

Chapter 1: Introduction

Silberschatz et al, 8th or higher edition

Regarding Attendance

- I will call names, about 10 to 20, in a random manner
- You should confirm by saying “PRESENT”.

Outline

- What Operating Systems Do
- Computer-System Organization
- Computer-System Architecture
- Operating-System Structure
- Operating-System Operations
 - Process, Memory and Storage Management
 - Protection and Security
- Kernel data structures
- Computing environments
 - Distributed Systems
 - Special-Purpose Systems
- Computing Environments
- Open-Source Operating Systems

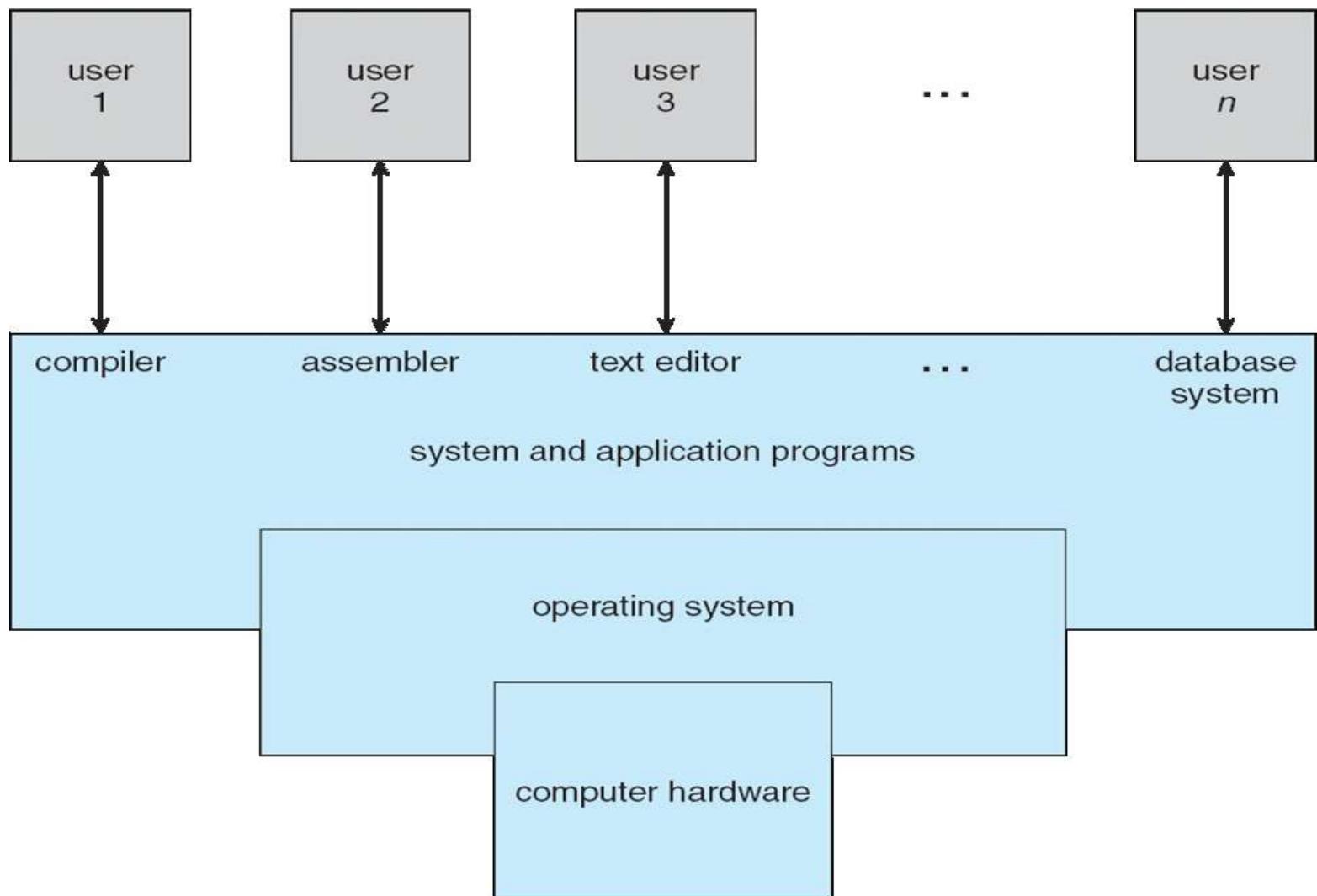
What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware
- Operating system goals:
 - Execute user programs and make solving user problems easier
 - Make the computer system convenient to use
 - Use the computer hardware in an efficient manner

Computer System Structure

- Computer system can be divided into four components:
 - Hardware – provides basic computing resources
 - ▶ CPU, memory, I/O devices
 - Operating system
 - ▶ Controls and coordinates use of hardware among various applications and users
 - Application programs –
 - ▶ define the ways in which the system resources are used to solve the computing problems of the users
 - ▶ Examples: Word processors, compilers, web browsers, database systems, video games
 - Users
 - ▶ People, machines, other computers

Four Components of a Computer System

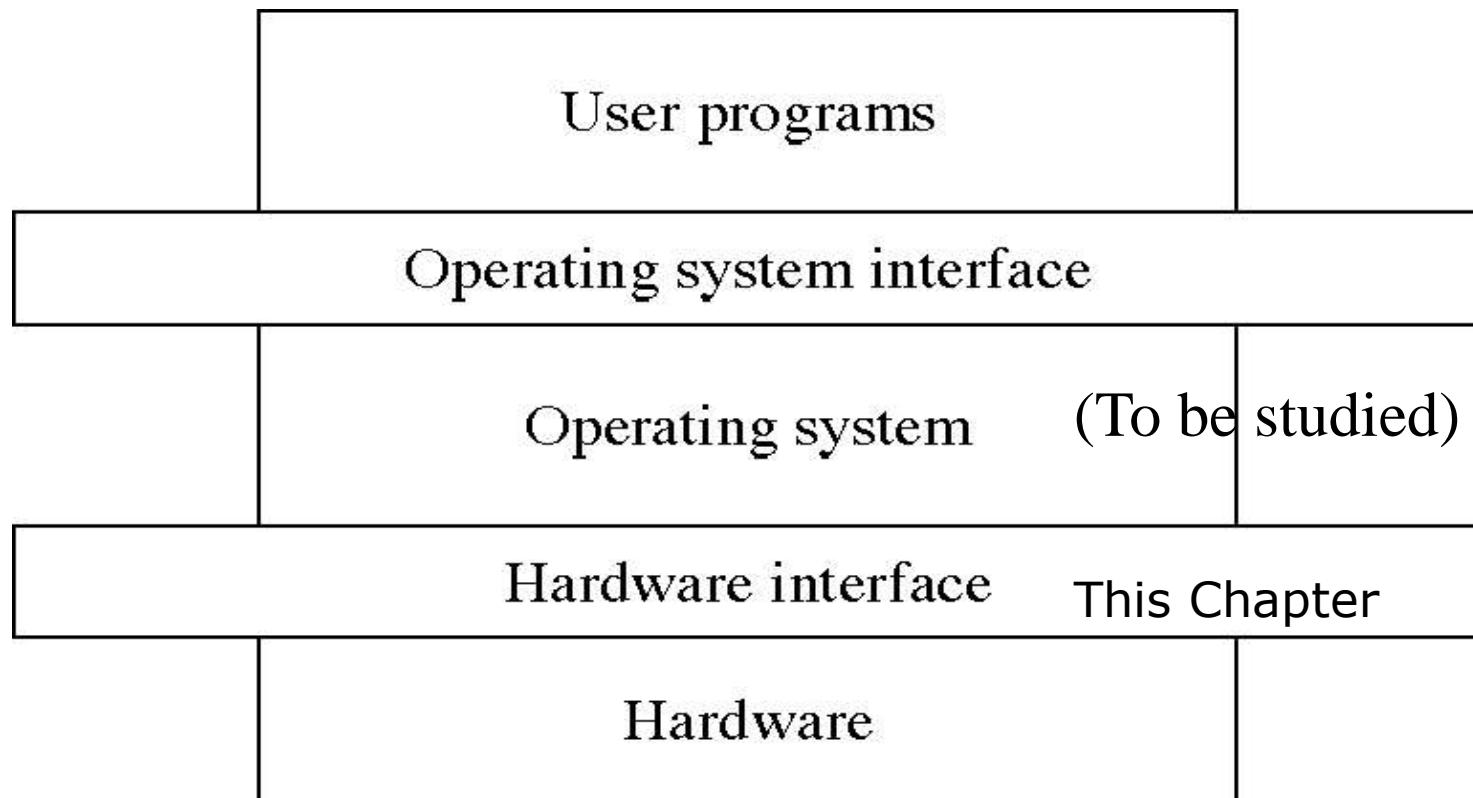


What Operating Systems Do

- Depends on the point of view
- Users view
 - Users want convenience, **ease of use**
 - Don't care about **resource utilization**
 - But shared computer such as **mainframe** or **minicomputer** must keep all users happy
- Users of dedicate systems such as **workstations** have dedicated resources but frequently use shared resources from **servers**
- Handheld computers are resource poor, optimized for usability and battery life
- Some computers have little or no user interface, such as embedded computers in devices and automobiles

Objectives of this chapter

- To provide a grand tour of the major operating systems components
- To provide coverage of basics of computer system organization



Operating System Definition

■ Systems view

- OS is a **resource allocator**
 - ▶ Manages all resources
 - ▶ Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
 - ▶ Controls execution of programs to prevent errors and improper use of the computer

Operating System Definition(Cont.)

- No universally accepted definition
 - “Everything a vendor ships when you order an operating system” is good approximation
 - ▶ But varies wildly
- “The one program running at all times on the computer” is the **kernel**. Everything else is either a system program (ships with the operating system) or an application program.

Computer Startup

- **bootstrap program** is loaded at power-up or reboot
 - Typically stored in ROM or EPROM, generally known as **firmware**
 - Initializes all aspects of system
 - Loads operating system kernel and starts execution

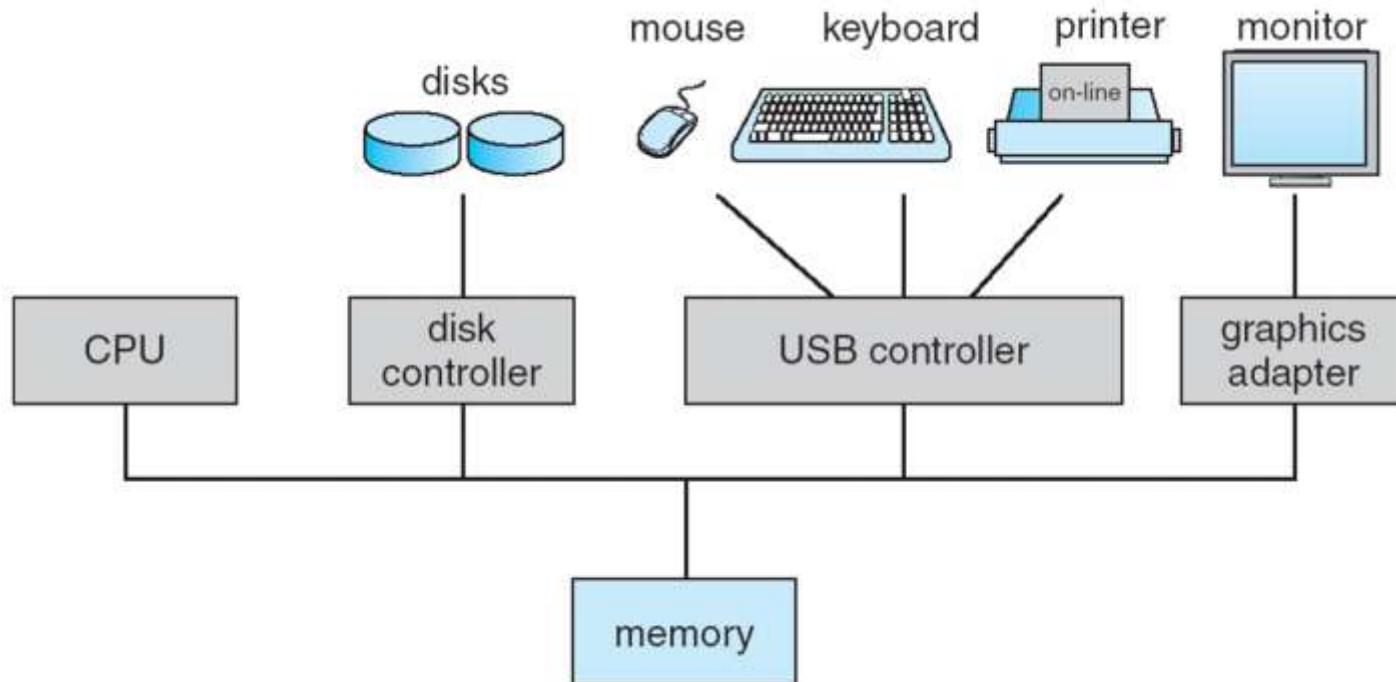
Outline

- What Operating Systems Do
- **Computer-System Organization**
- Computer-System Architecture
- Operating-System Structure
- Operating-System Operations
- Process, Memory and Storage Management
- Protection and Security
- Distributed Systems
- Special-Purpose Systems
- Computing Environments
- Open-Source Operating Systems

Computer System Organization

■ Computer-system operation

- One or more CPUs, device controllers connect through common bus providing access to shared memory
- Concurrent execution of CPUs and devices competing for memory cycles



Computer-Components

- Computer consists of processor, memory, and I/O components, with one or more modules of each type. These modules are connected through interconnection network.
- I/O devices and the CPU can execute concurrently.
- Each device controller is in-charge of a particular device type.
- Each device controller has a local buffer.
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller.
- **Device controller informs CPU that it has finished its operation by causing an *interrupt*.**

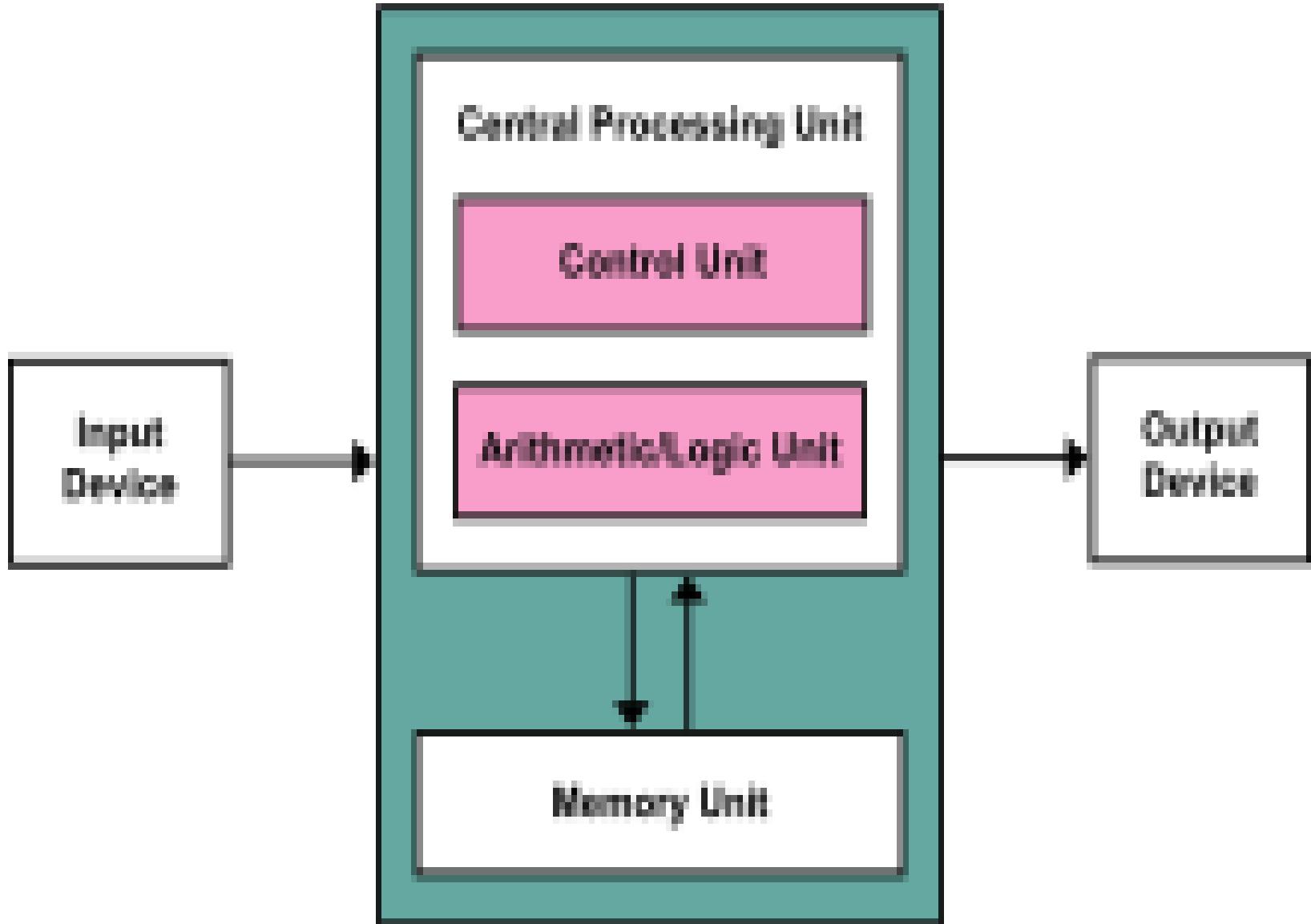
About the Interrupts

- In the next few slides, we will study the basics of interrupts by considering the architecture and operation of a simple computer.

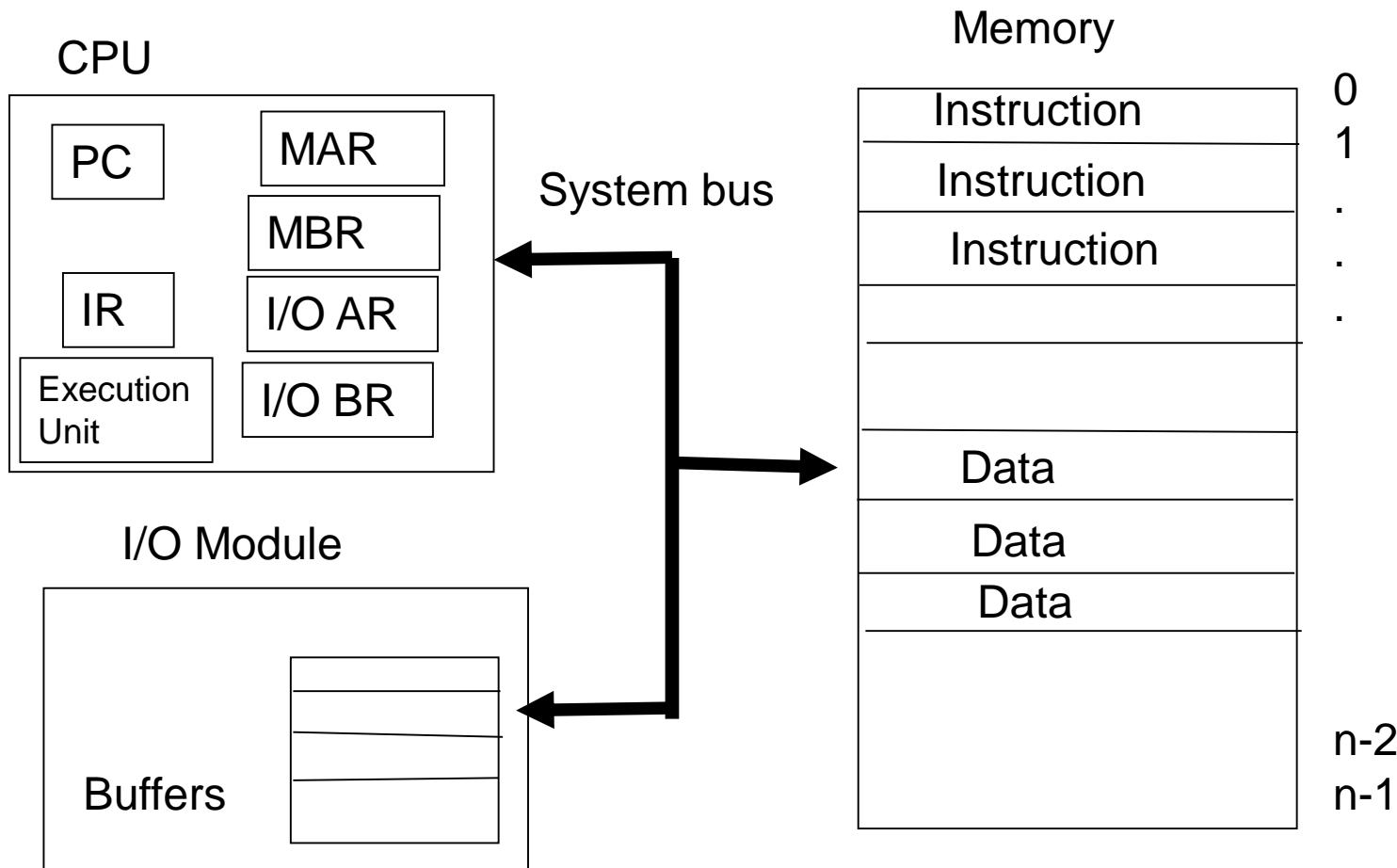
Von Neumann architecture

- The von Neumann architecture—also known as the von Neumann model or Princeton architecture—is a computer architecture based on a 1945 description by the mathematician and physicist John von Neumann and others in the First Draft of a Report on the EDVAC (Electronic Discrete Variable Automatic Computer). That document describes a design architecture for an electronic digital computer with
 - A processing unit that contains an arithmetic logic unit and processor registers
 - A control unit that contains an instruction register and program counter
 - Memory that stores **data and instructions**
 - External mass storage
 - Input and output mechanisms

Von Neumann architecture



Architecture of a simple computer



Architecture of a simple computer

- **Processor:** Controls the operation of the computer and performs data processing functions. It is called CPU.
- **Main memory:** Stores data and programs; it is volatile
- **I/O modules:** Moves data between the computer and external environment.
- **System bus:** mechanism of communication among processors, main memory, and I/O modules.

Simple Computer

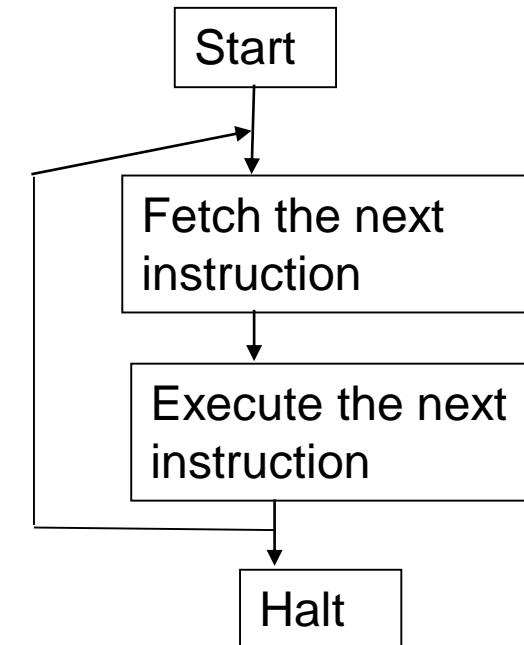
- Operation: Processor controls everything.
 - **MAR:** Memory address register: which specifies the address in memory for the next read or write.
 - **MBR:** Memory buffer register: which contains the data to be written into memory or which receives data read from memory.
 - **I/O AR:** Address of I/O device
 - **I/O BR:** Exchange of data between I/O and computer.

Simple Computer

- Processor Registers: Within the processor there is a set of registers that provide a level of memory that is faster and smaller than main memory.
 - User visible registers: Available to programmer
 - ▶ **Data registers:** Can be used by the programmer.
 - ▶ **Address Registers:** Contains Main memory address of data and instructions.
 - **Index register:** Index to base value
 - **Segment pointer:** It contains a reference to a particular segment.
 - **Stack pointer:** Points top of the stack.
 - **Control and status registers:** These are employed to control the operation of the processor. Differ from machine to machine.
 - ▶ MAR, MBR,I/O AR, and I/O BR
 - ▶ **Program Counter:** Contains the address of the instruction to be fetched.
 - ▶ **Instruction Register:** Contains the instruction most recently fetched.
 - ▶ **PSW:** Program Status word: It is a register or a set of registers.

Simple Computer...

- **PSW:** Program Status word: It is a register or a set of registers.
 - ▶ **Sign:** contains sign bit of last arithmetic operation
 - ▶ **Zero:** it is set if the result of arithmetic operation is zero
 - ▶ **Carry:** It is set if there is a carry or borrow.
 - ▶ **Equal:** If the compare result is equality
 - ▶ **Overflow:** It is set if the result is overflow.
 - ▶ **Interrupt enable/disable:** Used to disable or enable interrupts.
 - ▶ **Supervisor:** Indicates whether the processor is executing in supervisor or user mode.



■ Instruction execution:

- Program execution is the main function of the computer.
- Instruction fetch and execute

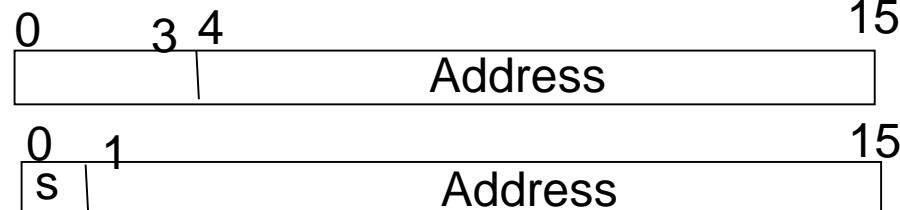
Simple computer...

- **Fetching:** Bringing the instructions from the main memory.
 - PC: Contains the address of the next instruction to be fetched.
 - ▶ Incremented unless told otherwise
- **Execute:**
 - The processor interprets the instructions and performs the required action.
 - ▶ **Processor-memory:** Transfer data from the memory
 - ▶ **Processor-I/O-** transfer data from peripheral device from the memory.
 - ▶ **Data processing:** Arithmetic and logic operations
 - ▶ **Control:** The instructions may specify the sequence of the next instruction to be fetched.

Simple Computer..

An Example:

Instruction format



PC: address of the instruction

Integer format

IR: Instruction being executed

Program

AC: Accumulator: temporary storage

1=0001=LOAD ADDRESS: Load AC from memory

LOAD 940

2=0010=STORE ADDRESS: Store AC to memory

ADD 941

3=0101=ADD ADDRESS: Add AC to memory

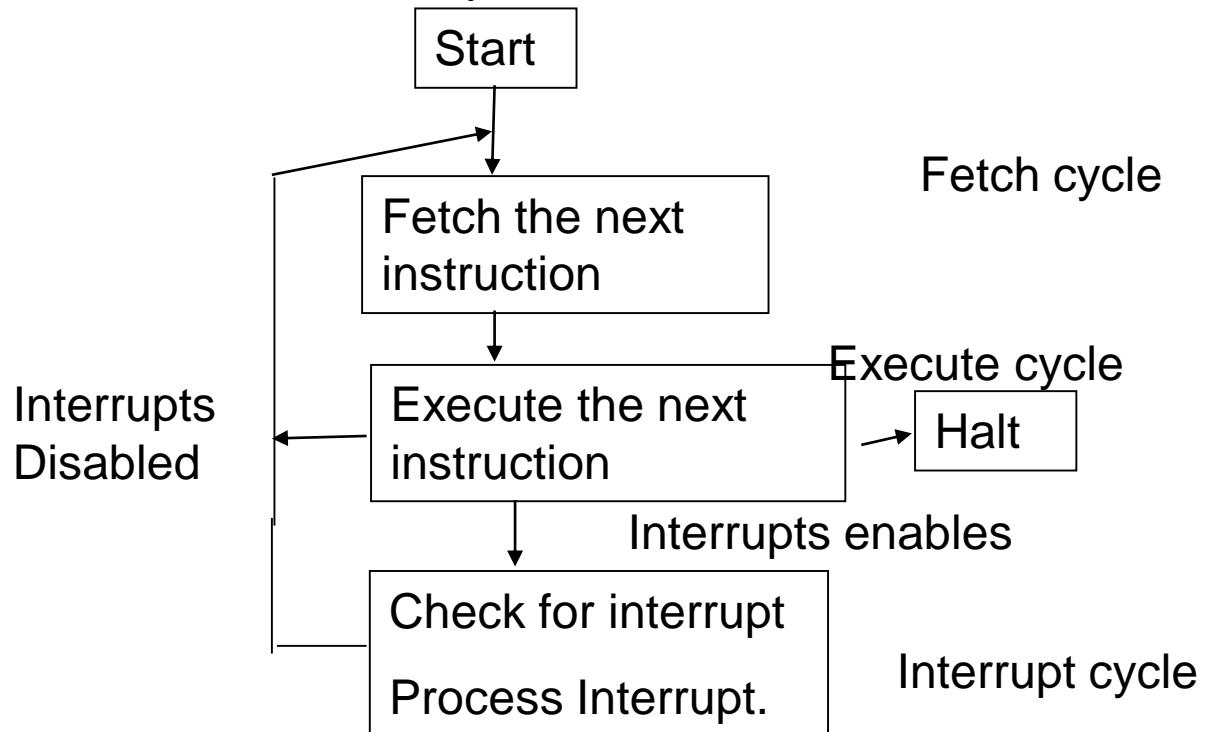
STORE 941

HLT

| | |
|-----|-------|
| 300 | 1 940 |
| 301 | 3 941 |
| 302 | 2 941 |
| | |
| 940 | |
| 941 | |

Interrupt processing

- To improve the performance, interrupts are provided.
- With interrupts, the processor can be engaged in executing other while an I/O operation is in progress.
- Interrupt cycle is added to the instruction cycle.



Common Functions of Interrupts

- Interrupt transfers the control to the interrupt service routine generally, through the *interrupt vector*, which contains the addresses of all the service routines.
- Interrupt architecture must save the address of the interrupted instruction.
- Incoming interrupts are *disabled* while another interrupt is being processed to prevent a *lost interrupt*.
- A *trap* is a software-generated interrupt caused either by an error or a user request.
- **An operating system is *interrupt driven*.**

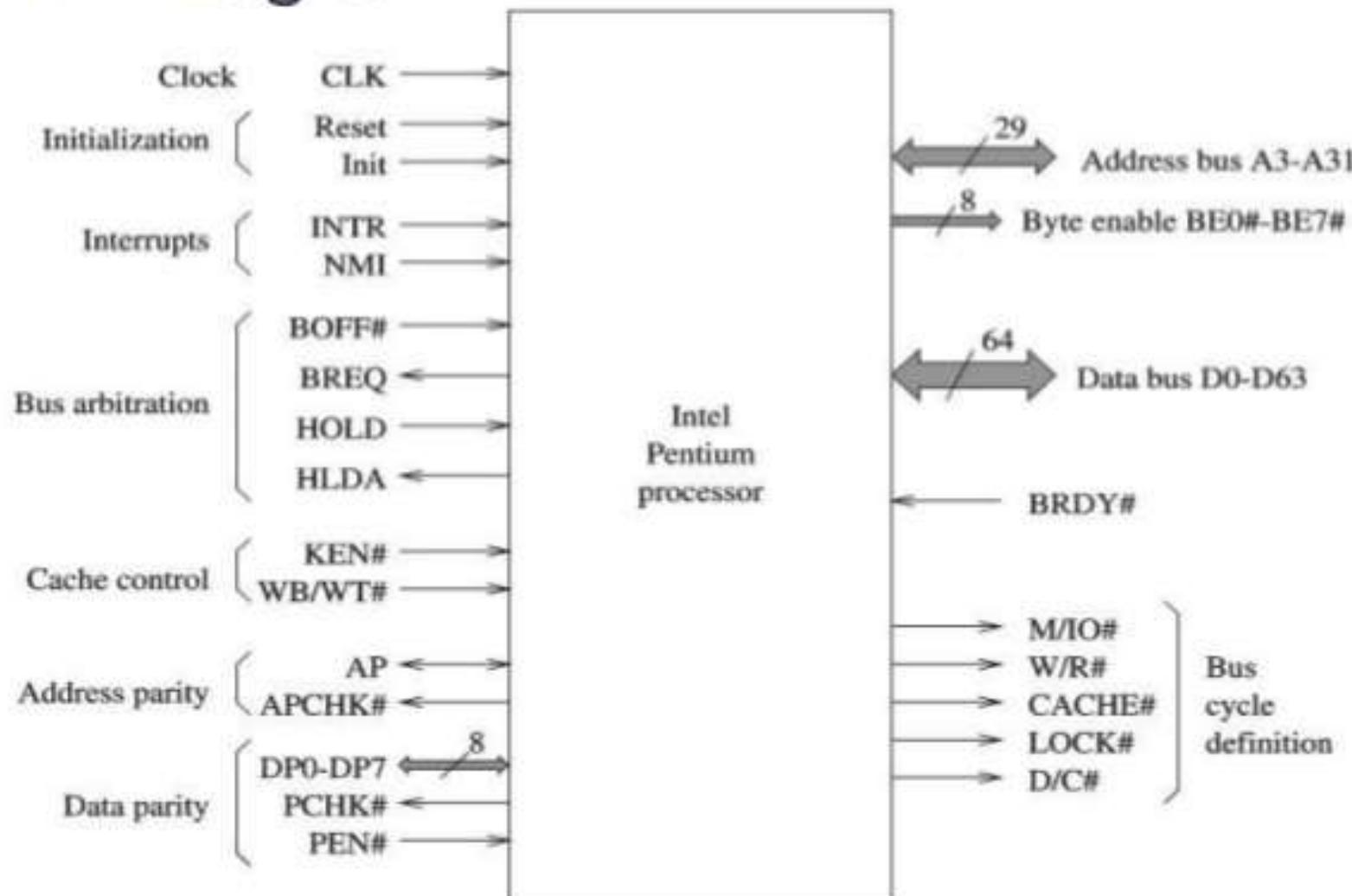
Interrupt Handling

- To start I/O operation, CPU loads the appropriate registers within the device controller
- The device controller examines the contents of these registers to determine what action to take.
- If it s a read operation the controller transfers the data into local buffer.
- Then it informs the CPU through interrupt.
- The operating system preserves the state of the CPU by storing registers and the program counter.

Interrupt Handling...

- **Interrupt handler:** The CPU hardware has a wire called interrupt request line; that CPU senses after executing every instruction.
- When a CPU senses a signal, it saves the PC, and PSW on a stack and jump to the interrupt handler routine at a fixed address in memory.
- **Interrupt vector:** It contains the memory addresses of specialized interrupt handlers.

Pin Diagram



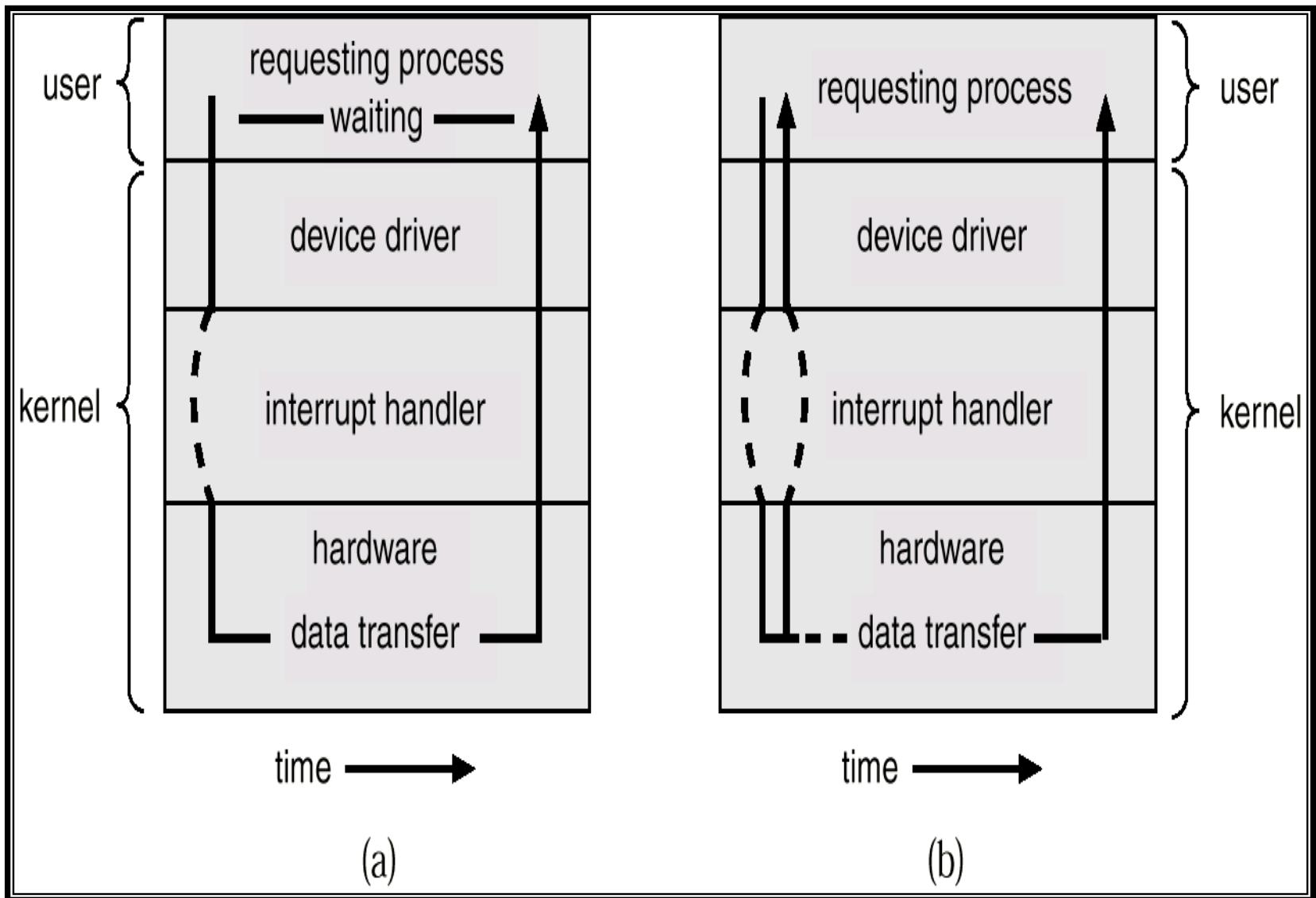
I/O Structure

- Two types of I/O
- **Synchronous I/O:** After I/O starts, control returns to user program only upon I/O completion.
 - Wait instruction idles the CPU until the next interrupt
 - Wait loop (contention for memory access).
 - At most one I/O request is outstanding at a time, no simultaneous I/O processing.
- **Asynchronous I/O:** After I/O starts, control returns to user program without waiting for I/O completion.
- *Device-status table* contains entry for each I/O device indicating its type, address, and state.
- Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt.

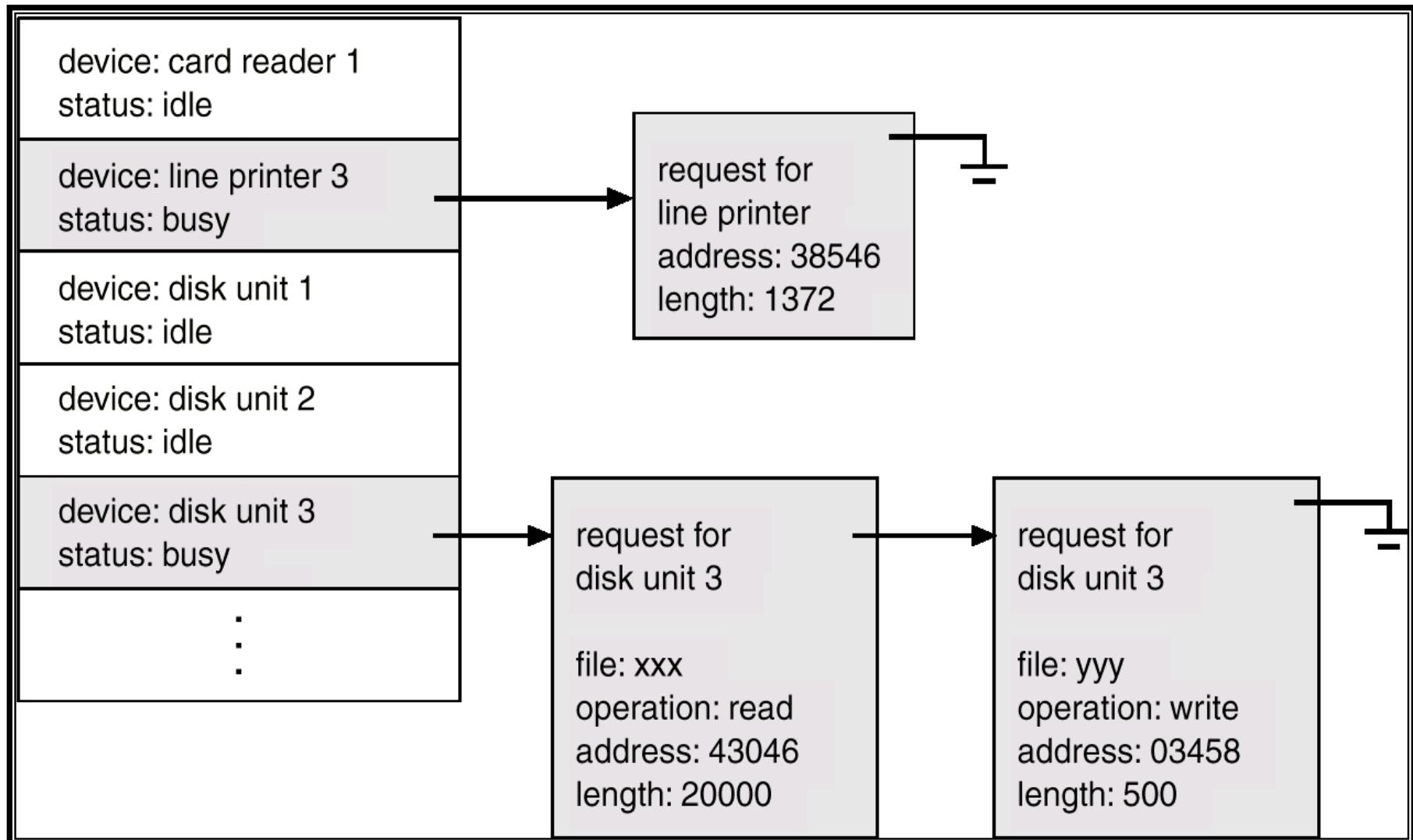
I/O Structure

- Waiting for I/O operation
 - Loop: Jump Loop
- The above loop continues until interrupt occurs.

Two I/O Methods

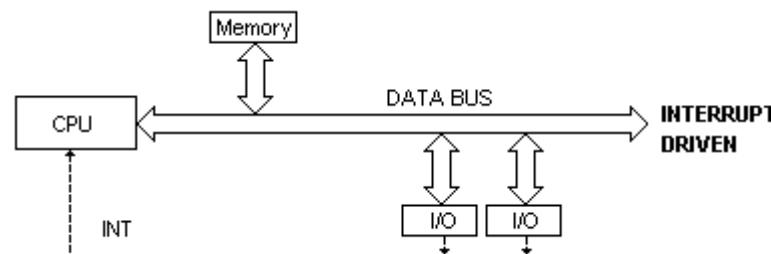


Device-Status Table



I/O communication techniques

- Three kinds of data transfer
 - **Program data transfer:** CPU checks the I/O status
 - The I/O module does not interrupt the processor.
 - Processor is responsible for extracting the data from main memory and shifting data out of main memory.
 - It is a time-consuming process that keeps the processor busy unnecessarily.
 - **Interrupt driven data transfer:** when I/O is ready it interrupts the CPU
 - In programmed I/O the processor repeatedly interrogate the status of the I/O module.
 - In Interrupt driven data transfer, the I/O module will interrupt the processor to request service when it is ready to exchange data with the processor.



Direct Memory Access (DMA) Structure

■ DMA:

- Used for high-speed I/O devices able to transmit information at close to memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- Only one interrupt is generated per block, rather than one interrupt per byte.

■ DMA: Processor issues a command to DMA module by sending the following information.

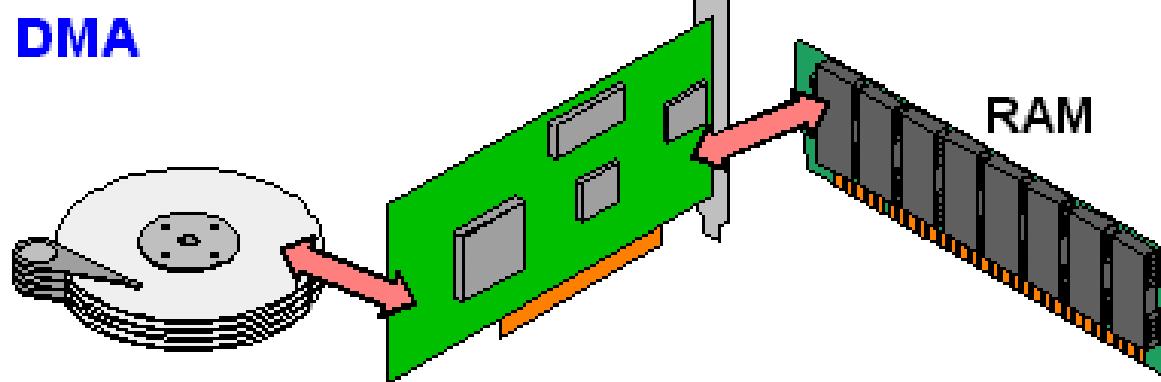
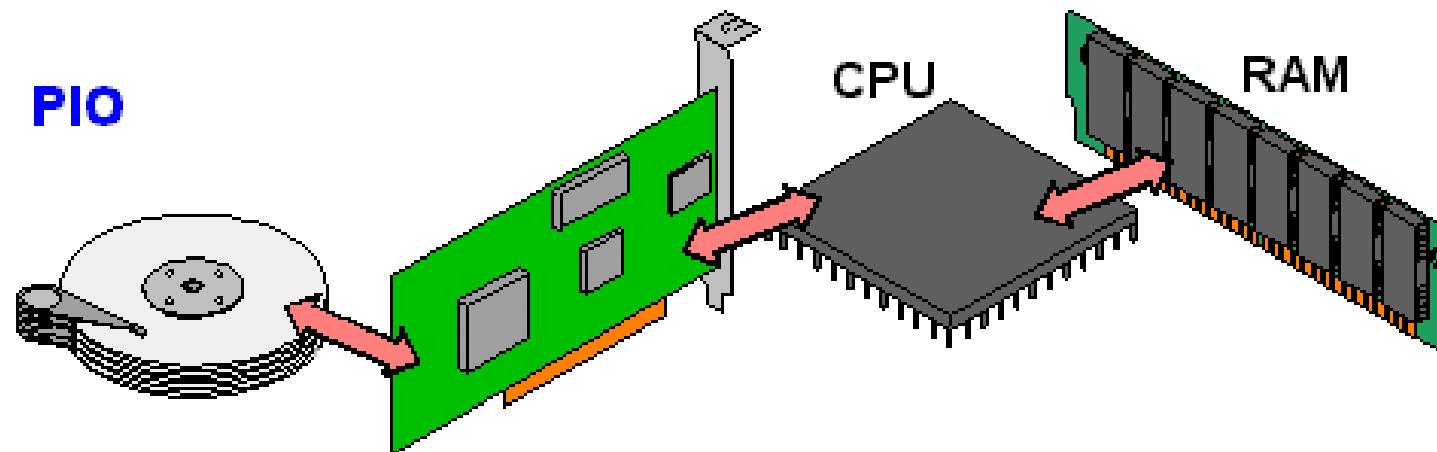
- Whether read or write is requested.
- The address of the I/O device involved.
- The starting location of the memory to read from or write to.
- The number of words to be read or written.

■ The processor continues other work.

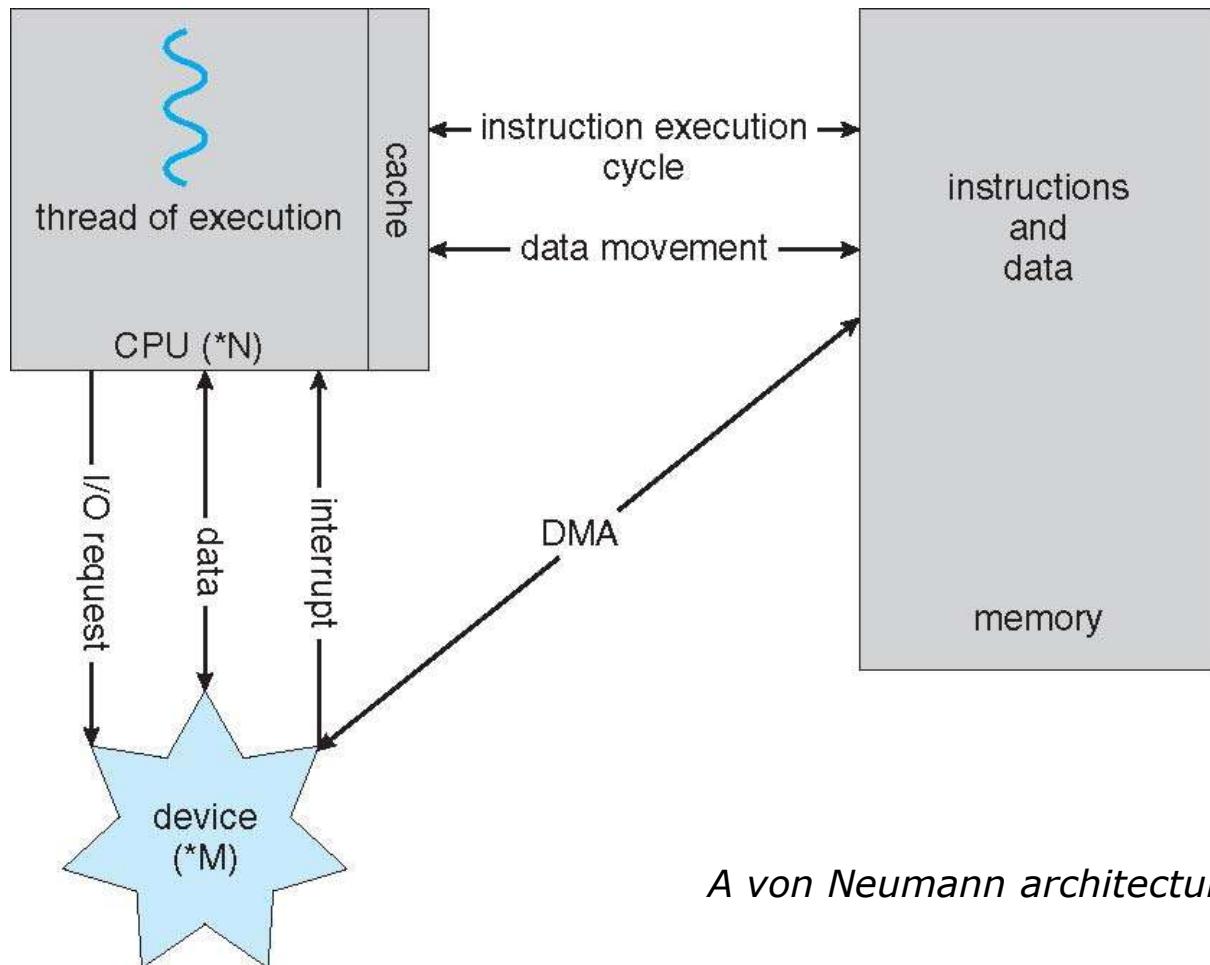
■ The DMA module transfers the data and interrupts the processor.

■ It takes the control of the bus to transfer data from memory.

■ The processor becomes slow, or must wait for the bus.



How a Modern Computer (Single Processor System) Works?



A von Neumann architecture

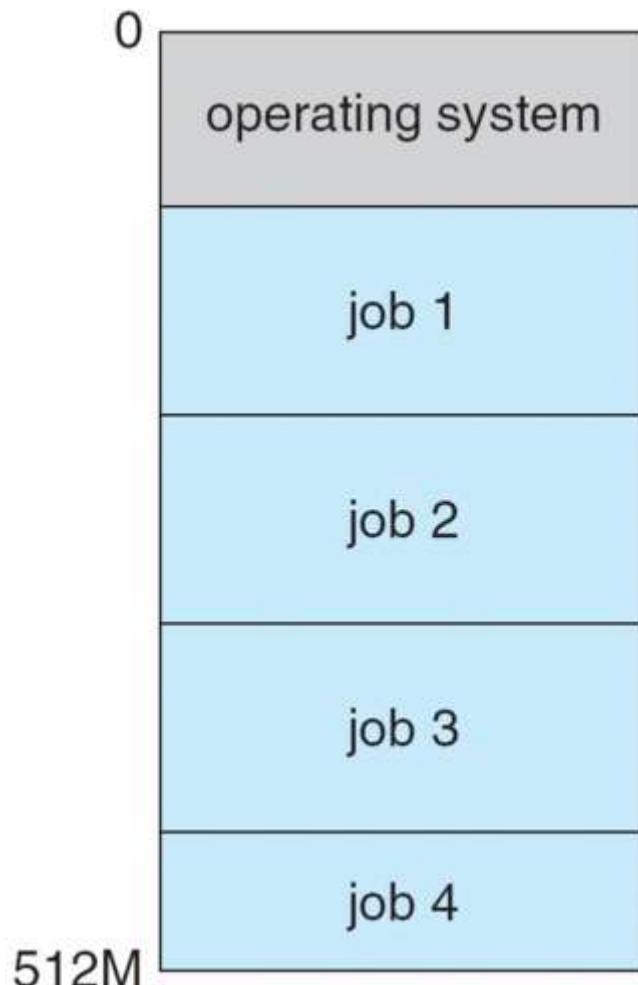
Outline

- What Operating Systems Do
- Computer-System Organization
- Computer-System Architecture
- **Operating-System Structure**
- Operating-System Operations
- Process, Memory and Storage Management
- Protection and Security
- Distributed Systems
- Special-Purpose Systems
- Computing Environments
- Open-Source Operating Systems

Operating System Structure

- **Multiprogramming** needed for efficiency
 - Single user cannot keep CPU and I/O devices busy at all times
 - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
 - A subset of total jobs in system is kept in memory
 - One job selected and run via **job scheduling**
 - When it has to wait (for I/O for example), OS switches to another job
- **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
 - **Response time** should be < 1 second
 - Each user has at least one program executing in memory \Rightarrow **process**
 - If several jobs ready to run at the same time \Rightarrow **CPU scheduling**
 - If processes don't fit in memory, **swapping** moves them in and out to run
 - **Virtual memory** allows execution of processes not completely in memory

Memory Layout for Multiprogrammed System



Operating system operations

- Dual mode operation
- Process Management
- Memory management
- Storage management
- Protection and Security

About Dual-mode operation

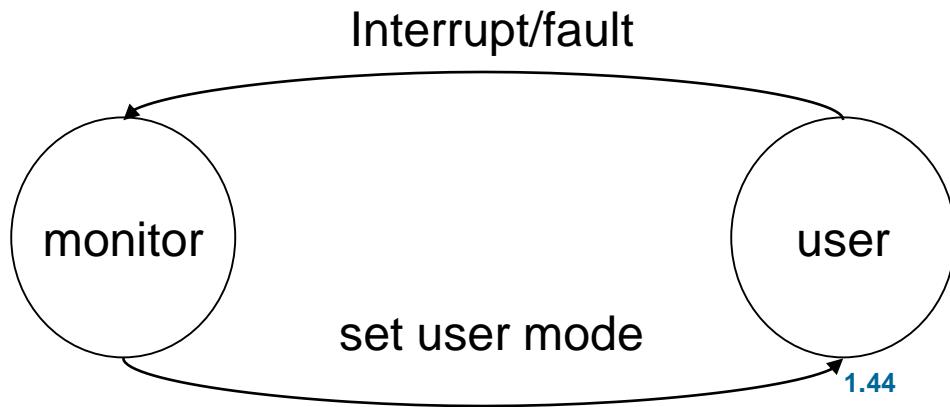
- In Early systems, programmers had complete control over the system.
- As OSs developed the control was given to OS.
- OS started performing many functions such as I/O.
- OS started sharing resources among several programs
- Multiprogramming put several programs at same time.
- Without sharing the error may cause problem to only one program.
- With sharing an error may cause problems to many programs.
 - Sometimes to OS itself.
 - OSs have to be protected from such incorrect programs.
- MS-DOS and MAC-OS allow this kind of error.
- Protection measures.
 - Dual-Mode Operation
 - I/O Protection
 - Memory Protection
 - CPU Protection

Dual-Mode Operation

- Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.
- Provide hardware support to differentiate between at least two modes of operations.
 1. *User mode* – execution done on behalf of a user.
 2. *Monitor mode* (also *kernel mode* or *system mode*) – execution done on behalf of operating system.
- This architectural enhancement is useful for many aspects of system operation.
- When system starts, hardware starts in monitor mode.
- OS is then loaded and OS starts user processes in user mode.

Dual-Mode Operation (Cont.)

- *Mode bit* added to computer hardware to indicate the current mode: monitor (0) or user (1).
- Whenever OS takes control the mode bit is 0.
- When an interrupt or fault occurs hardware switches to monitor mode.
- This dual-mode operation protects computer from errant users and errant users from other.
- This protection can be achieved by designating some of the instructions as privileged instructions.
- ***Privileged instructions can be issued only in monitor mode.***
 - **MS-DOS was written without mode bit**
 - **Pentium provides dual mode operation: so it provides greater protection.**



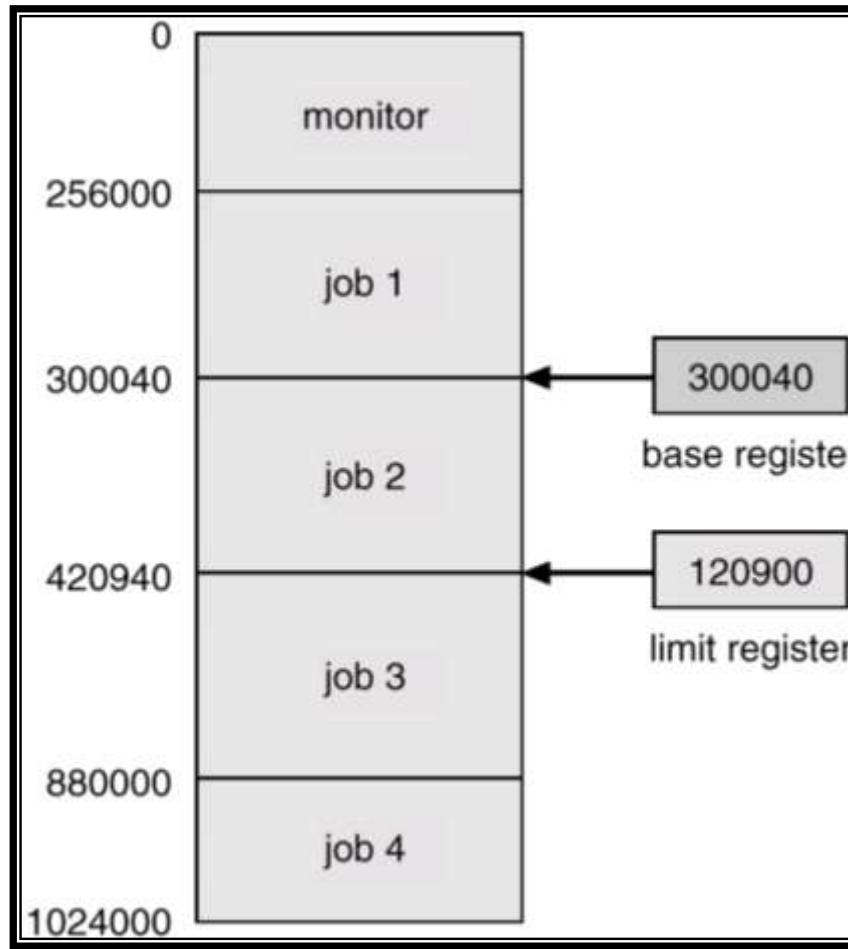
I/O Protection

- A user program may disrupt the normal operation
 - By issuing a illegal I/O operation.
- Solution: All I/O instructions are privileged instructions.
- Must ensure that a user program could never gain control of the computer in monitor mode
- A user program that, as part of its execution, stores a new address in the interrupt vector. How to protect it ?
- **Answer: Consider Interrupt instructions which modify interrupt vector as privileged instructions.**

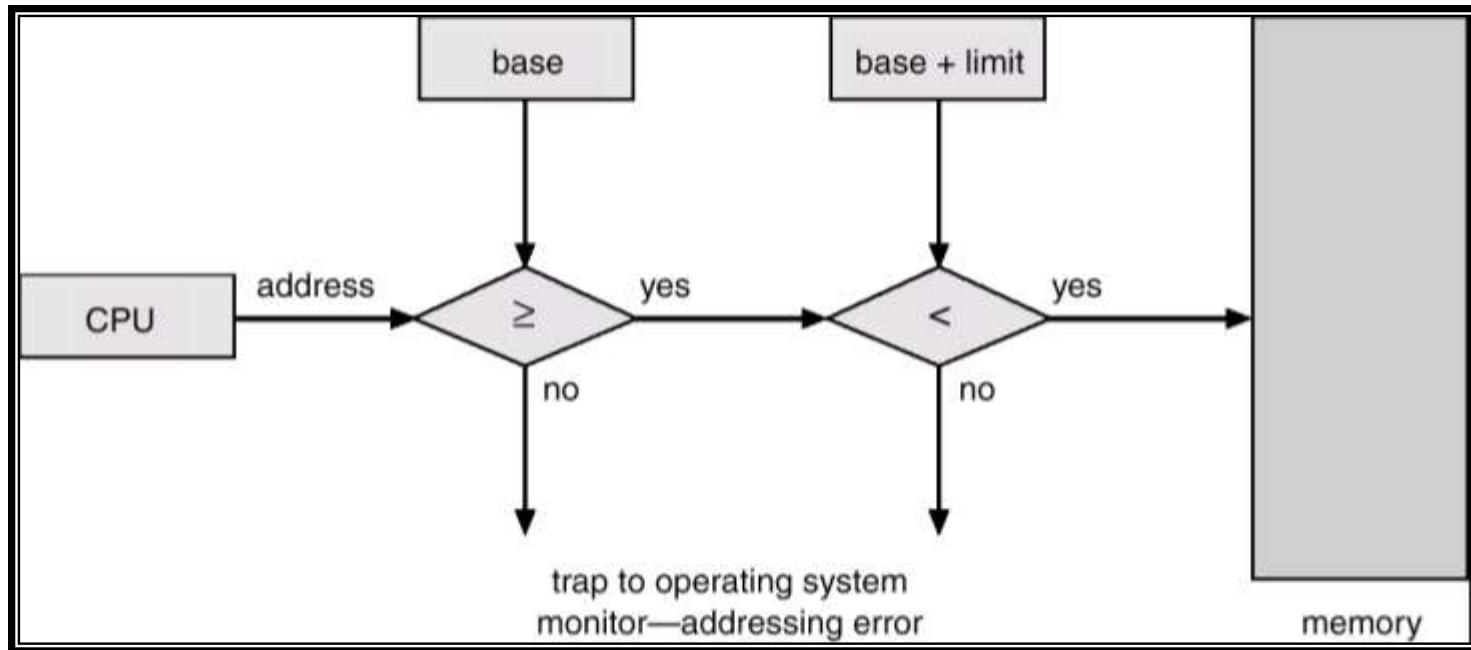
Memory Protection

- Must provide memory protection at least for the interrupt vector and the interrupt service routines.
- In order to have memory protection, add two registers that determine the range of legal addresses a program may access:
 - **Base register** – holds the smallest legal physical memory address.
 - **Limit register** – contains the size of the range
- Memory outside the defined range is protected.

Use of A Base and Limit Register



Hardware Address Protection



Hardware Protection

- When executing in monitor mode, the operating system has unrestricted access to both monitor and user's memory.
- The load instructions for the *base* and *limit* registers are privileged instructions.

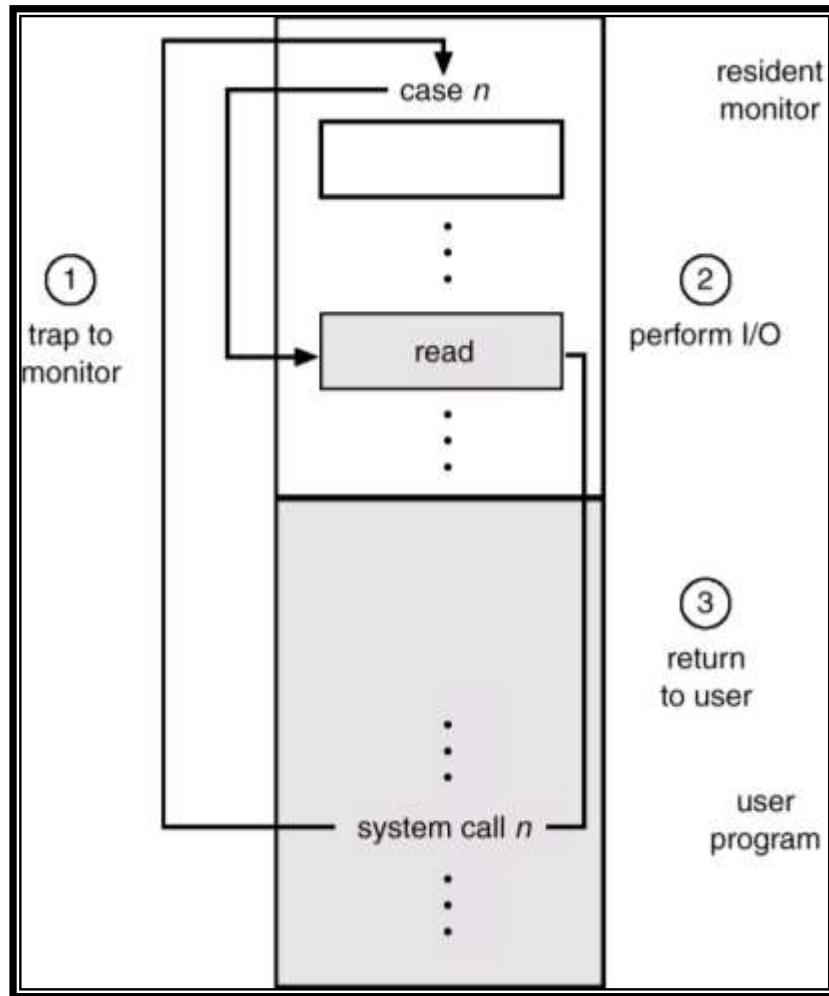
CPU Protection

- Ensures that OS maintains control.
- *Timer* – interrupts computer after a specified period to ensure operating system maintains control.
 - Timer is decremented every clock tick.
 - When timer reaches the value 0, an interrupt occurs.
- Timer commonly used to implement time sharing.
- Timer also used to compute the current time.
- Load-timer is a privileged instruction.

General system operation

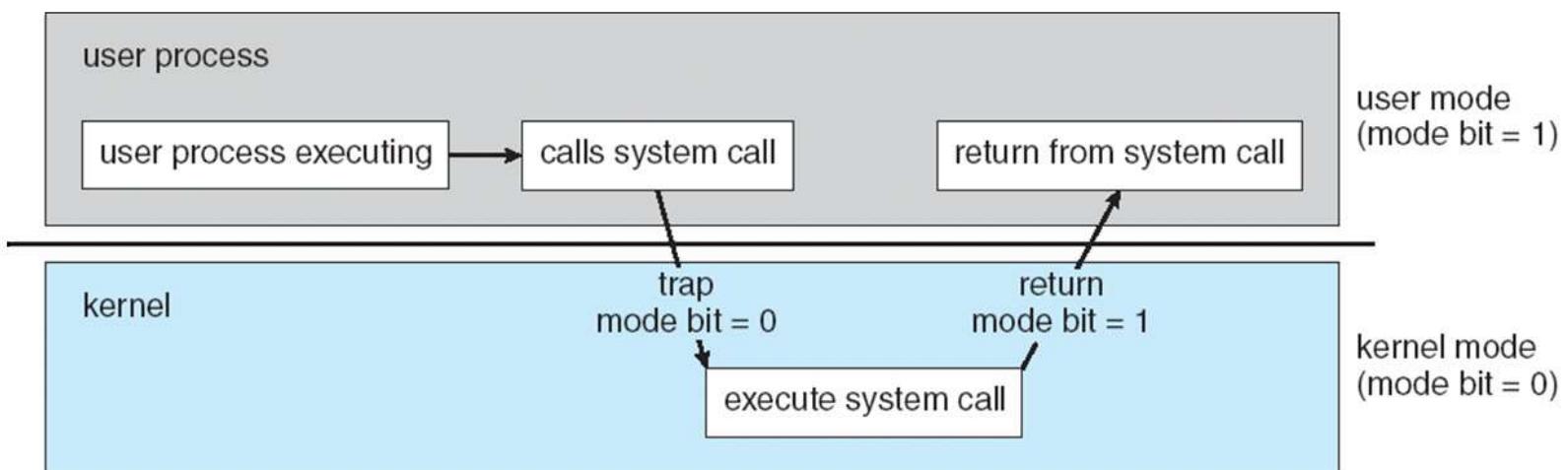
- To improve the utilization led to the development of multiprogramming and timesharing.
 - Resources of the computers are shared among many programs and processes
- Sharing led to allow OS to have control over I/O to provide continuous, consistent and correct operation.
- Dual mode of operation is introduced.
- I/O instructions and instructions to modify memory management are privileged instructions.
 - HLT instruction is privileged
 - The instructions to turn-on and turn-off are privileged.
- The instructions to change user mode to monitor mode are privileged.
- User must ask the monitor to do I/O. Such a request is called **system call**.
- When a system call is executed it is treated by the hardware as a software interrupt.
- Control is passed to interrupt vector to a service routine to the OS by setting the mode bit to monitor mode.
- The OS examines the request , and passes necessary information and checks correctness and executes the request.
- It returns the control to the statement after the system call.

General System Architecture...



Transition from User to Kernel/Supervisor Mode

- Timer to prevent infinite loop / process hogging resources
 - Set interrupt after specific period
 - Operating system decrements counter
 - When counter zero generate an interrupt
 - Set up before scheduling process to regain control or terminate program that exceeds allotted time



Outline

- What Operating Systems Do
- Computer-System Organization
- Computer-System Architecture
- Operating-System Structure
- Operating-System Operations
- **Process, Memory and Storage Management**
- **Protection and Security**
- Distributed Systems
- Special-Purpose Systems
- Computing Environments
- Open-Source Operating Systems

Process Management

- A process is a program in execution. It is a unit of work within the system.
Program is a *passive entity*, process is an *active entity*.
- Process needs resources to accomplish its task
 - CPU, memory, I/O, files
 - Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one **program counter** specifying location of next instruction to execute
 - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
 - Concurrency by multiplexing the CPUs among the processes / threads

Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

Memory Management

- All data in memory before and after processing
- All instructions in memory in order to execute
- Memory management determines what is in memory when
 - Optimizing CPU utilization and computer response to users
- **Memory management activities**
 - Keeping track of which parts of memory are currently being used and by whom
 - Deciding which processes (or parts thereof) and data to move into and out of memory
 - Allocating and deallocating memory space as needed

Storage Management

- OS provides uniform, logical view of information storage
 - Abstracts physical properties to logical storage unit - **file**
 - Each medium is controlled by device (i.e., disk drive, tape drive)
 - ▶ Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- **1. File-System management**
 - Files usually organized into directories
 - Access control on most systems to determine who can access what
 - OS activities include
 - ▶ Creating and deleting files and directories
 - ▶ Primitives to manipulate files and dirs
 - ▶ Mapping files onto secondary storage
 - ▶ Backup files onto stable (non-volatile) storage media

Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
 - Free-space management
 - Storage allocation
 - Disk scheduling
- Some storage need not be fast
 - Tertiary storage includes optical storage, magnetic tape
 - Still must be managed – by OS or applications
 - Varies between WORM (write-once, read-many-times) and RW (read-write)

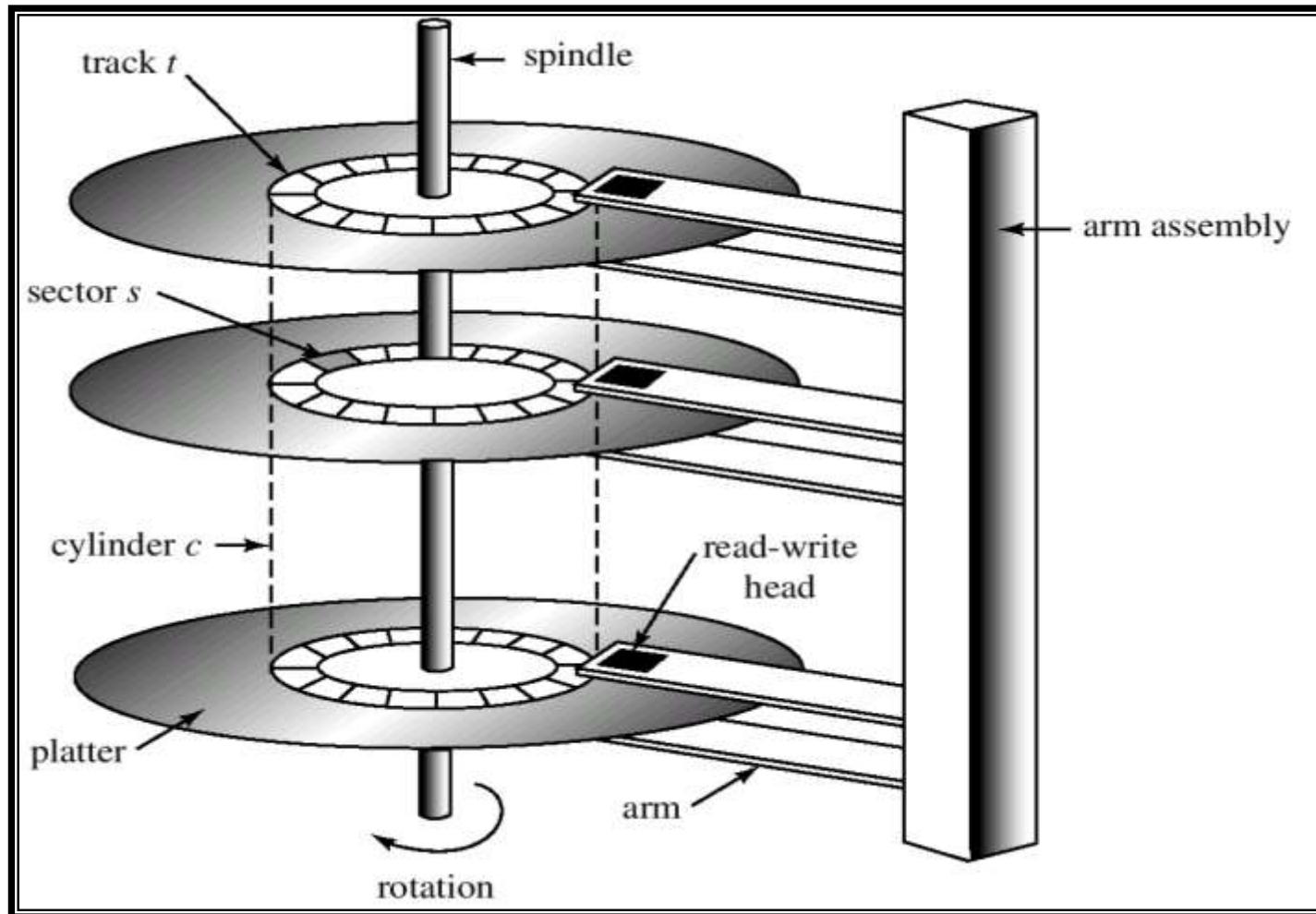
I/O Subsystem

- One purpose of OS is to hide peculiarities of hardware devices from the user
- I/O subsystem responsible for
 - Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs)
 - General device-driver interface
 - Drivers for specific hardware devices

Storage Structure

- Main memory – only large storage media that the CPU can access directly
 - **Random access**
 - Typically **volatile**
- Secondary storage – extension of main memory that provides large **nonvolatile** storage capacity
- Magnetic disks – rigid metal or glass platters covered with magnetic recording material
 - Disk surface is logically divided into **tracks**, which are subdivided into **sectors**
 - The **disk controller** determines the logical interaction between the device and the computer

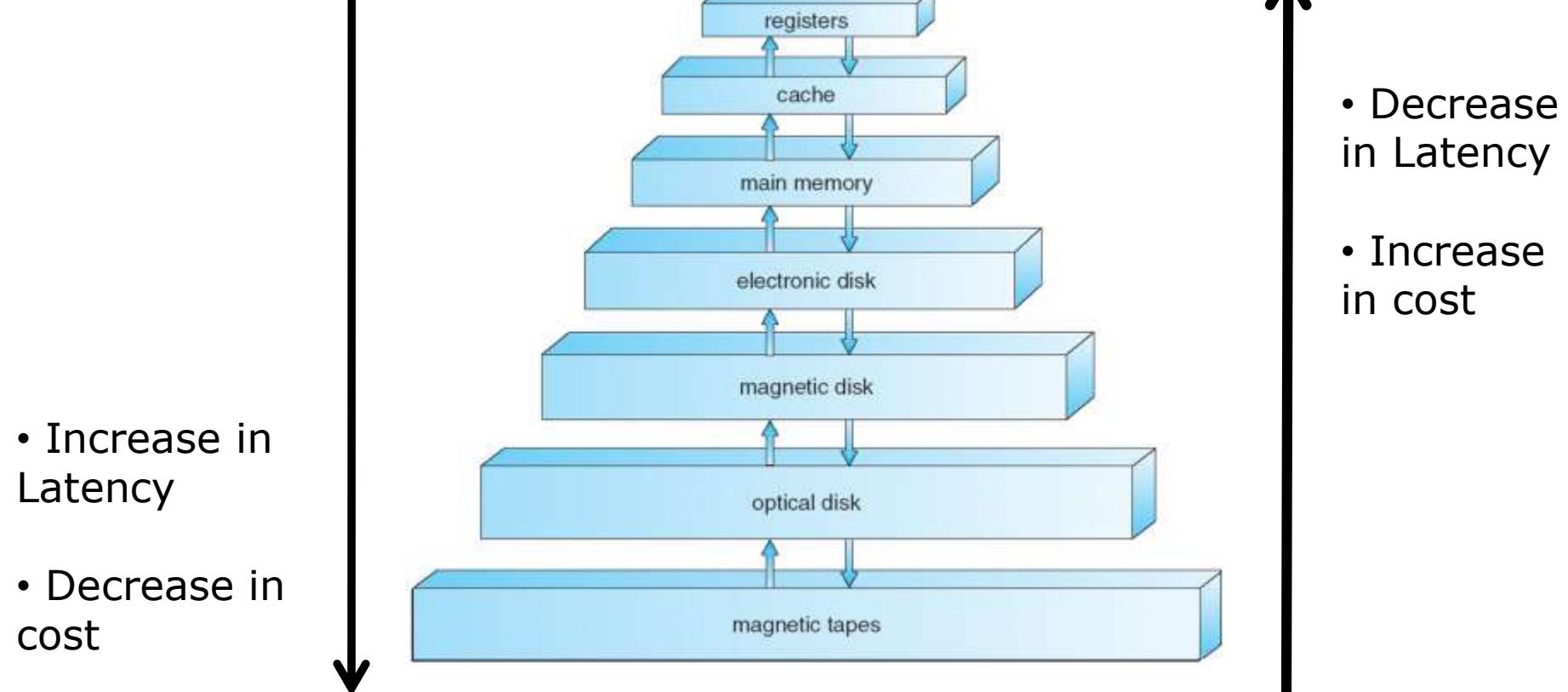
Moving-Head Disk Mechanism



Storage Hierarchy

- Storage systems organized in hierarchy
 - Speed
 - Cost
 - Volatility
- **Caching** – copying information into faster storage system; main memory can be viewed as a *cache* for secondary storage

Storage-Device Hierarchy



Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
 - If it is, information used directly from the cache (fast)
 - If not, data copied to cache and used there
- Cache smaller than storage being cached
 - Cache management is an important design problem
 - Cache size and replacement policy

Performance of Various Levels of Storage

- Movement between levels of storage hierarchy can be explicit or implicit

| Level | 1 | 2 | 3 | 4 |
|---------------------------|---|-------------------------------|------------------|------------------|
| Name | registers | cache | main memory | disk storage |
| Typical size | < 1 KB | > 16 MB | > 16 GB | > 100 GB |
| Implementation technology | custom memory with multiple ports, CMOS | on-chip or off-chip CMOS SRAM | CMOS DRAM | magnetic disk |
| Access time (ns) | 0.25 – 0.5 | 0.5 – 25 | 80 – 250 | 5,000.000 |
| Bandwidth (MB/sec) | 20,000 – 100,000 | 5000 – 10,000 | 1000 – 5000 | 20 – 150 |
| Managed by | compiler | hardware | operating system | operating system |
| Backed by | cache | main memory | disk | CD or tape |

Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
 - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
 - User identities (**user IDs**, security IDs) include name and associated number, one per user
 - User ID then associated with all files, processes of that user to determine access control
 - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file
 - **Privilege escalation** allows user to change to effective ID with more rights

Kernel Data Structures

- Lists
 - Single, double, circular
- Stacks
 - Last in first out
- Queues
 - First in first out
- Trees
 - Binary search tree, balanced tree (B-tree)
- Hash functions and maps
 - Input to hash function is data and output is numeric value, which can be used as an index.
- Bitmaps
 - Status of n items.
 - Example: Consider 1000 disk blocks. We use 1000 bits to know the status. Each of 1000 bits represents the status of one block (occupied (1) or unoccupied (0)).

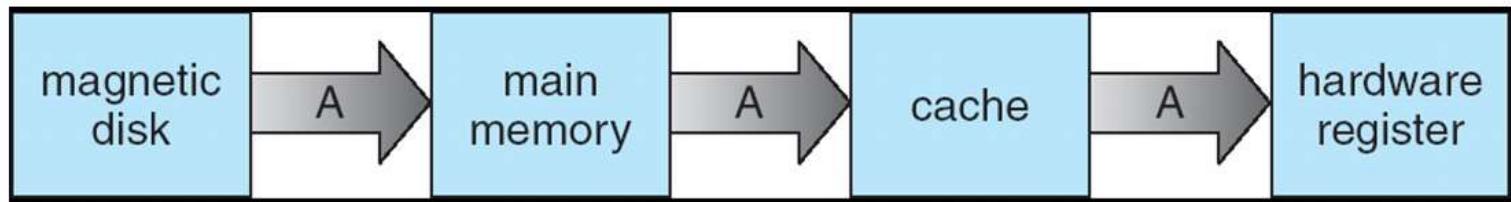
Computer Architectures

Multi-processor systems

- Also known as parallel systems or tightly coupled systems
- Advantages
 - Increased throughput:
 - ▶ Number of tasks finished per unit time.
 - Economy of scale
 - ▶ Total cost is less due to sharing of resources, like disk, mass storage and power supplies.
 - Increased reliability
 - ▶ One processor will not halt the system
 - ▶ Graceful degradation
 - The service depends on the available hardware
 - ▶ Fault tolerant
 - The service level does not decrease. Duplicate components.
 - » HP nonstop

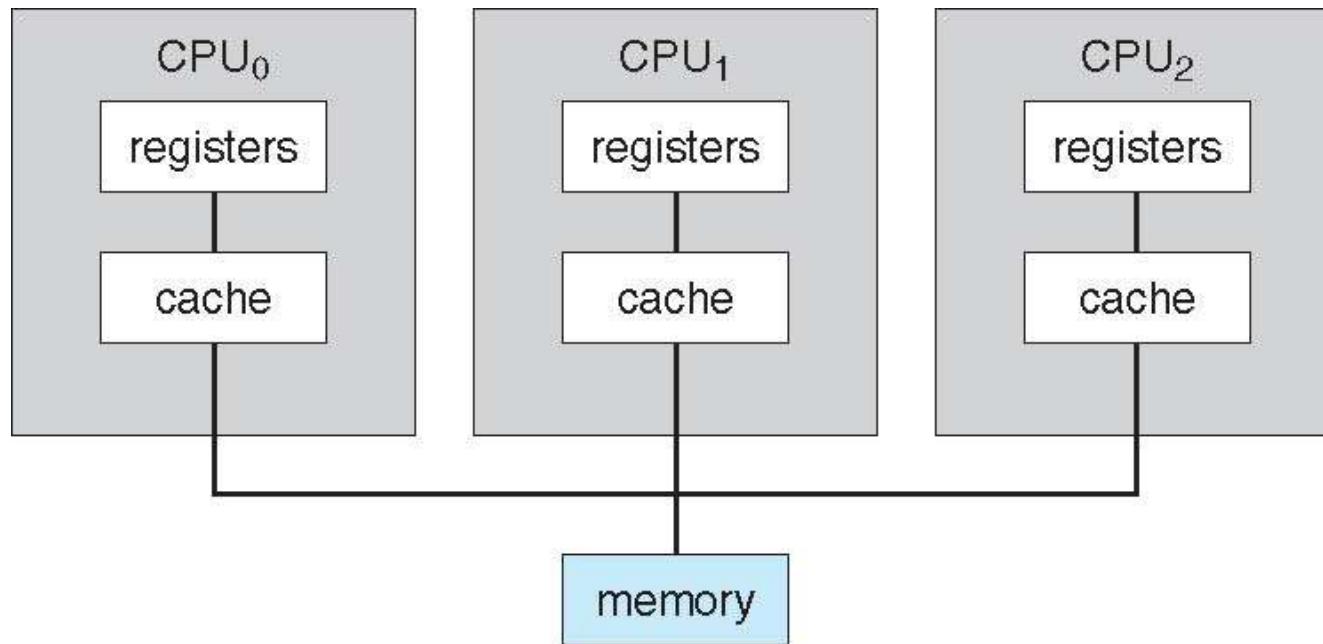
Migration of Integer A from Disk to Register

- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy



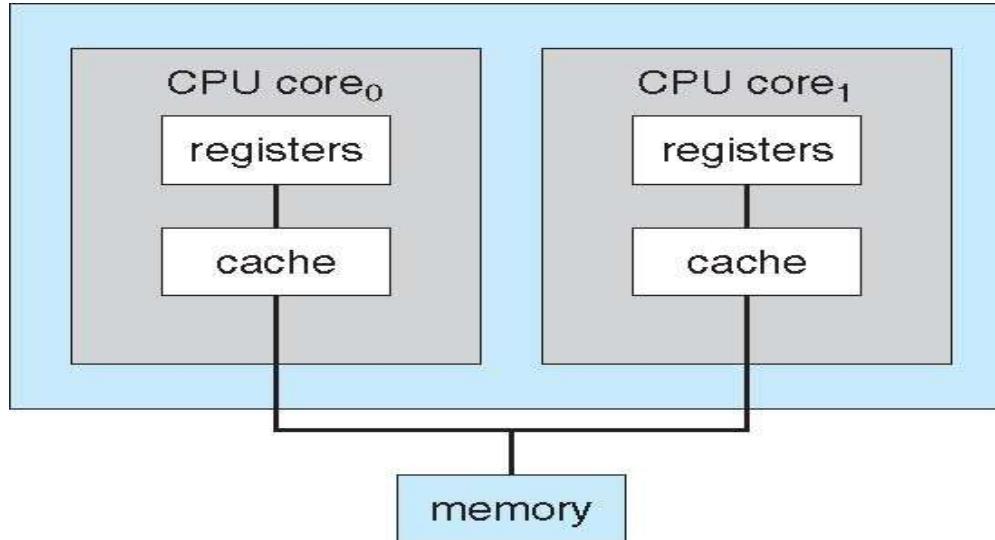
- Multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex
 - Several copies of a datum can exist
 - Various solutions covered in Chapter 17

Symmetric Multiprocessing Architecture



- Symmetric multiprocessing: each processor performs all tasks
 - Solaris
- Asymmetric multiprocessing: each processor is assigned a specific task
- Memory access model is changed from Uniformed memory access (UMA) to Non uniformed memory access (NUMA):

Recent Trend: A Dual-Core Design



- Including multiple computing cores on a single chip.
 - Less power, communication faster
- However, it puts a pressure on OS designers and application programmers
- Blade servers:
 - Each blade server runs the OS independently
 - Multiple processor boards are placed in the same chassis.

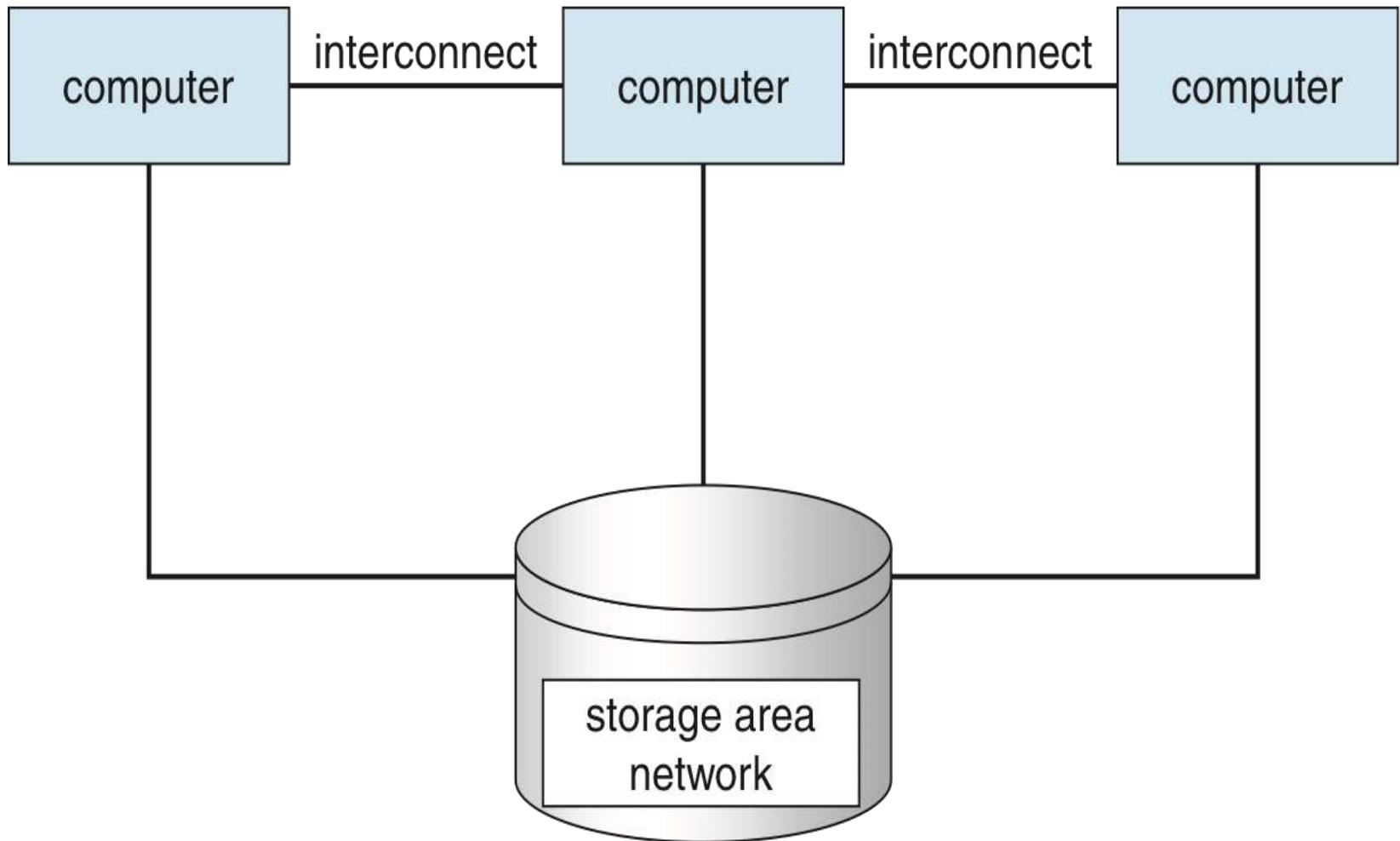
Blade Servers



Clustered Systems

- Like multiprocessor systems, but multiple systems working together
 - Usually sharing storage via a **storage-area network (SAN)**
 - Provides a **high-availability** service which survives failures
 - ▶ **Asymmetric clustering** has one machine in hot-standby mode
 - ▶ **Symmetric clustering** has multiple nodes running applications, monitoring each other
 - Some clusters are for **high-performance computing (HPC)**
 - ▶ Applications must be written to use **parallelization**

Clustered Systems



Outline

- What Operating Systems Do
- Computer-System Organization
- Computer-System Architecture
- Operating-System Structure
- Operating-System Operations
- Process, Memory and Storage Management
- Protection and Security
- **Distributed Systems**
- **Special-Purpose Systems**
- **Computing Environments**
- **Open-Source Operating Systems**

Computing Environments

Distributed Computing

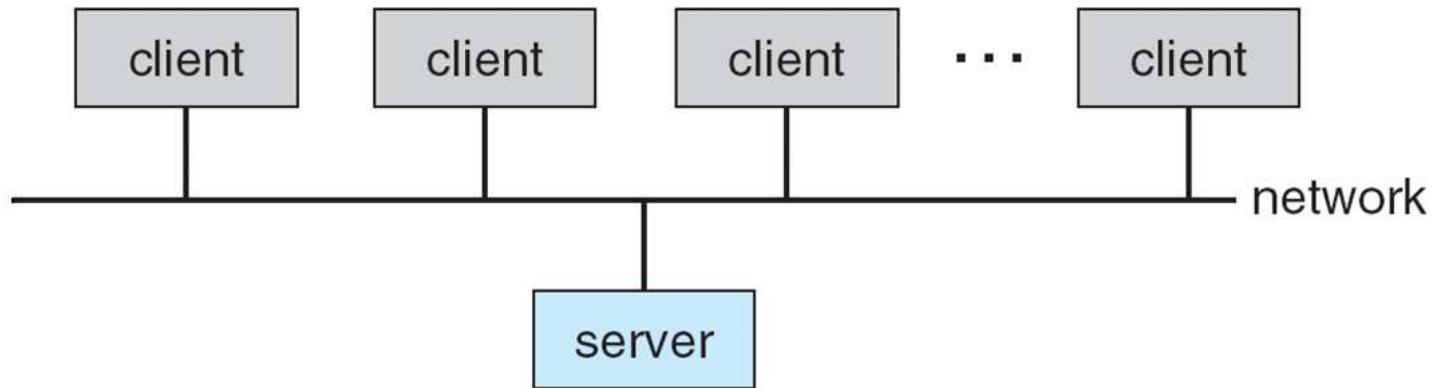
- Collection of separate, possibly heterogeneous, systems networked together
 - Network is a communications path
 - Local Area Network (**LAN**)
 - Wide Area Network (**WAN**)
 - Metropolitan Area Network (**MAN**)
- **Network Operating System** provides features between systems across network
 - Communication scheme allows systems to exchange messages
 - Illusion of a single system

Network Environments

- Traditional computer
 - Blurring over time
 - Office environment
 - ▶ PCs connected to a network, terminals attached to mainframe or minicomputers providing batch and timesharing
 - ▶ Now portals allowing networked and remote systems access to same resources
 - Home networks
 - ▶ Used to be single system, then modems
 - ▶ Now firewalled, networked

Client-server computing

- Client-Server Computing
 - Dumb terminals supplanted by smart PCs
 - Many systems now **servers**, responding to requests generated by **clients**
 - ▶ **Compute-server** provides an interface to client to request services (i.e., database)
 - ▶ **File-server** provides interface for clients to store and retrieve files



Real-time embedded systems

- Real-time embedded systems most prevalent form of computers
 - Vary considerable, special purpose, limited purpose OS, **real-time OS**
- Multimedia systems
 - Streams of data must be delivered according to time restrictions
 - Quality of service
- Handheld systems
 - PDAs, smart phones, limited CPU, memory, power
 - Reduced feature set OS, limited I/O

Peer-to-Peer Computing

- Another model of distributed system
- P2P does not distinguish clients and servers
 - Instead all nodes are considered peers
 - May each act as client, server or both
 - Node must join P2P network
 - ▶ Registers its service with central lookup service on network, or
 - ▶ Broadcast request for service and respond to requests for service via **discovery protocol**
 - Examples include *Napster* and *Gnutella*

Virtualization

- It allows one operating system to run as an application of other operating system
- Emulation

Cloud Computing

- Delivers computing, storage, and applications as a service across network
- Employs thousands of servers and network

Web-Based Computing

- Web has become ubiquitous
- PCs most prevalent devices
- More devices becoming networked to allow web access
- New category of devices to manage web traffic among similar servers:
load balancers
- Use of operating systems like Windows 95, client-side, have evolved into Linux and Windows XP, which can be clients and servers

Open-Source Operating Systems

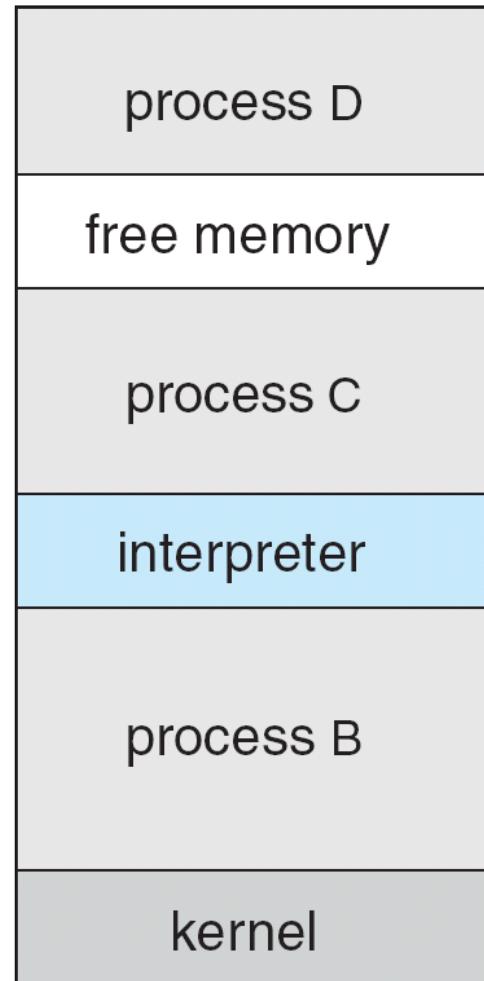
- Operating systems made available in source-code format rather than just binary **closed-source**
- Counter to the **copy protection** and **Digital Rights Management (DRM)** movement
- Started by **Free Software Foundation (FSF)**, which has “copyleft” **GNU Public License (GPL)**
- Examples include **GNU/Linux** and **BSD UNIX** (including core of **Mac OS X**), and many more

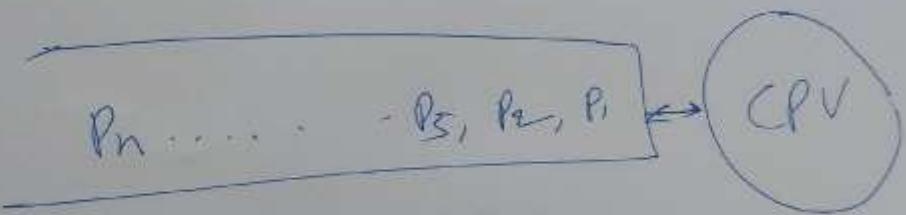
End of Chapter 1

Chapter 3: Process Concept

- Process Concept
- Process Scheduling
- Operations on Processes
- Cooperating Processes
- Interprocess Communication

FreeBSD Running Multiple Programs





An Analogy

■ Example

- ☛ Various workers (males, females, skilled, unskilled, semi-skilled) are working on some job.
- ☛ Question: How to organize them effectively to improve the performance ?
- ☛ Sitting worker is similar to program.
 - ☒ Does not use any tools and resources.
- ☛ Working worker is similar to process
 - ☒ Uses tools, interacts with other,
 - ☒ If he is working with others, cooperation and synchronization is required. Otherwise accidents might occur.
 - ☒ If he is working alone, no cooperation is needed.
- ☛ To manage working workers, the manager should have some information about the working worker.
 - ☒ What kind of tools he is using.
 - ☒ The nature of the job, and status of the job.
 - ☒
- ☛ In computer system
 - ☒ Operating system is similar to manager
 - ☒ The sitting workers are similar to the programs reside on the disk.
 - ☒ The working workers are processes
 - ☒ CPU is an expensive tool which should not be kept idle. It is OK if some workers wait for expensive tool.

Process Concept

- Early systems
 - ☞ One program at a time was executed and a single program has a complete control.
- Modern OSs allow multiple programs to be loaded in to memory and to be executed concurrently.
- This requires firm control over execution of programs.
- The notion of process emerged **to control the execution of programs.**
- A process
 - ☞ Unit of work
 - ☞ Program in execution
- OS consists of a collection of processes
 - ☞ OS processes executes system code.
 - ☞ User processes executes user code.
- By switching CPU between processes, the OS can make the computer more productive.

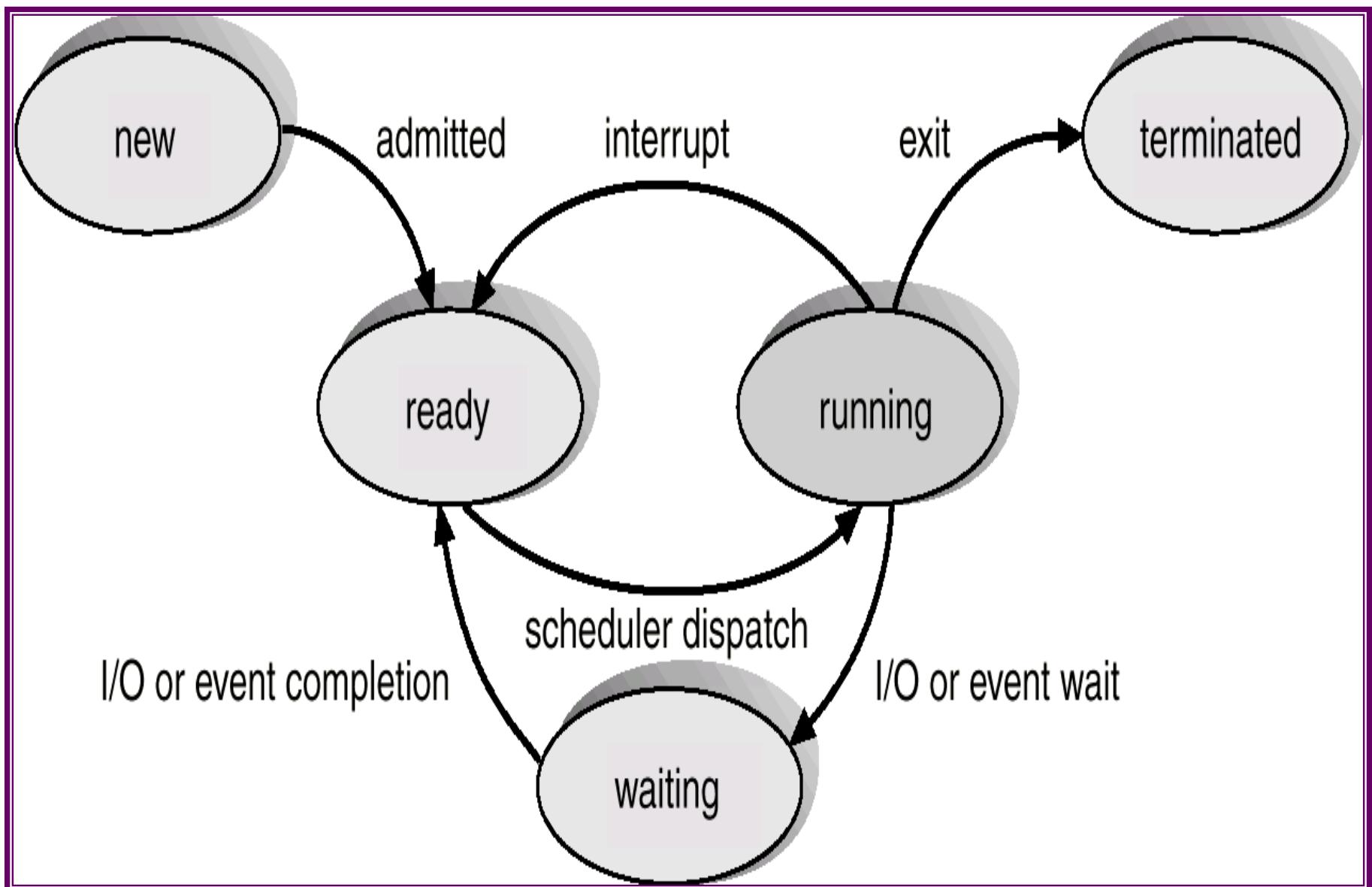
Process Concept...

- A program is simply a text
- Process (task or job) includes the current activity.– a program in execution; process execution must progress in sequential fashion.
- The components of a process are
 - ☞ The program to be executed
 - ☞ The data on which the program will execute
 - ☞ The resources required by the program– such as memory and file (s)
 - ☞ The status of execution
 - ☞ program counter
 - ☞ Stack
- A program is a passive entity, and a process is an active entity with the value of the PC. It is an execution sequence.
- Multiple processes can be associated with the same program (editor). They are separate execution sequences.
- For CPU, all processes are similar
 - ☞ Batch Jobs and user programs/tasks
 - ☞ Word file
 - ☞ Internet browser
 - ☞ System call
 - ☞ Scheduler
 - ☞

Process State

- As a process executes, it changes *state*
- *Each process can be in one of*
 - ☞ **new**: The process is being created.
 - ☞ **running**: Instructions are being executed.
 - ☞ **waiting**: The process is waiting for some event to occur.
 - ☞ **ready**: The process is waiting to be assigned to a process.
 - ☞ **terminated**: The process has finished execution.
- The names may differ between OSs.

Diagram of Process State



State change

- **Null → New :** a new process is created to execute the program
 - ☞ New batch job, log on
 - ☞ Created by OS to provide the service
- **New → ready:** OS will move a process from prepared to ready state when it is prepared to take additional process.
- **Ready → Running:** when it is a time to select a new process to run, the OS selects one of the process in the ready state.
- **Running → terminated:** The currently running process is terminated by the OS if the process indicates that it has completed, or if it aborts.
- **Running → Ready:** The process has reached the maximum allowable time or interrupt.
- **Running → Waiting:** A process is put in the waiting state, if it requests something for which it must wait.
 - ☞ Example: System call request.
- **Waiting → Ready:** A process in the waiting state is moved to the ready state, when the event for which it has been waiting occurs.
- **Ready → Terminated:** If a parent terminates, child process should be terminated
- **Waiting → Terminated:** If a parent terminates, child process should be terminated

Process Control Block (PCB)

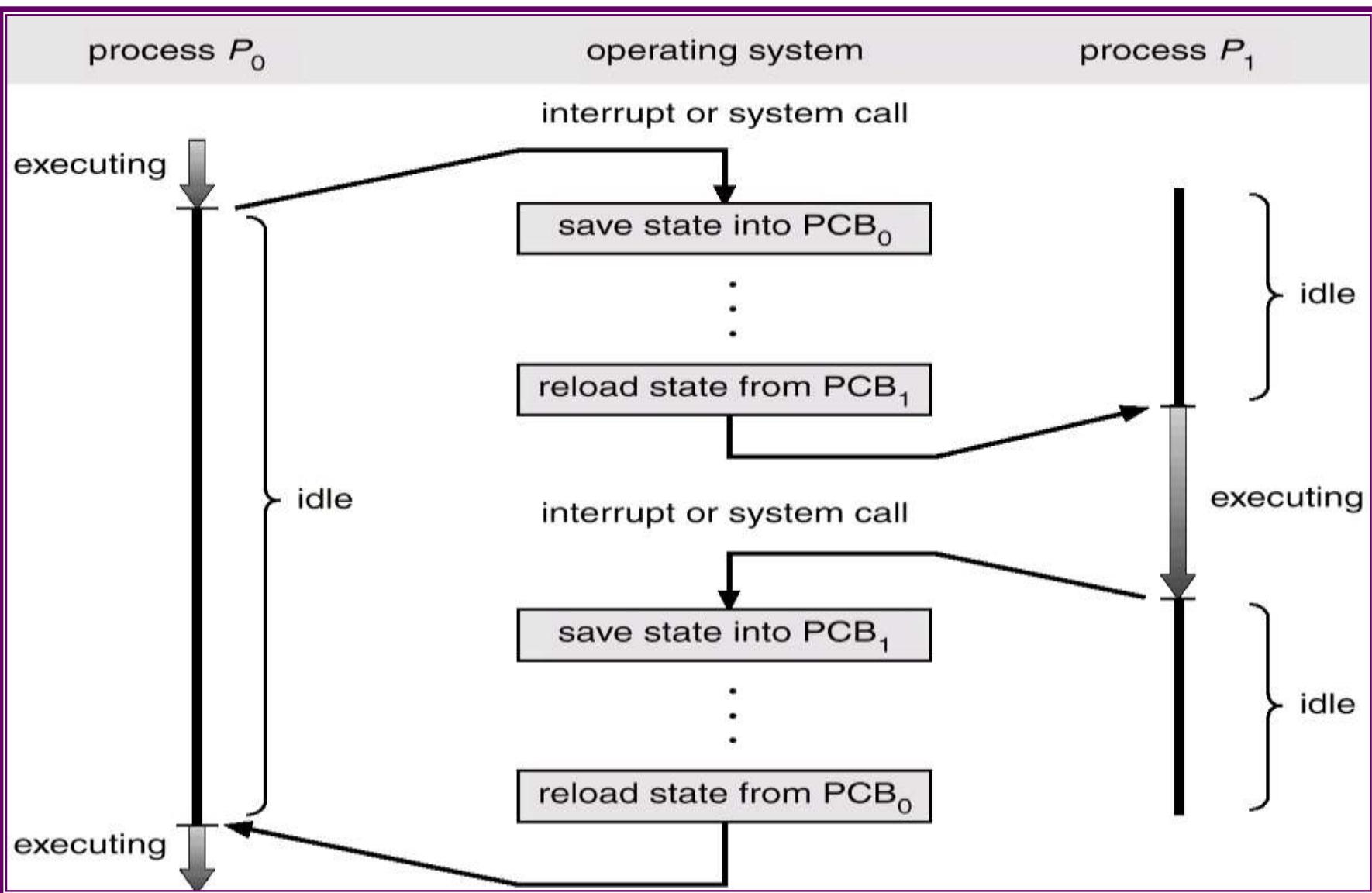
Information associated with each process.

- **Process state:** new, ready, running,...
- **PC:** address of the next instruction to execute
- **CPU registers:** includes data registers, stacks, condition-code information, etc
- **CPU scheduling information:** process priorities, pointers to scheduling queues, etc.
- **Memory-management information:** locations including value of base and limit registers, page tables and other virtual memory information.
- **Accounting information:** the amount of CPU and real time used, time limits, account numbers, job or process numbers etc.
- **I/O status information:** List of I/O devices allocated to this process, a list of open files, and so on

Process Control Block (PCB)

| | |
|---------------------------|------------------------|
| pointer | process state |
| | process number |
| | program counter |
| registers | |
| memory limits | |
| list of open files | |
| • • • | |

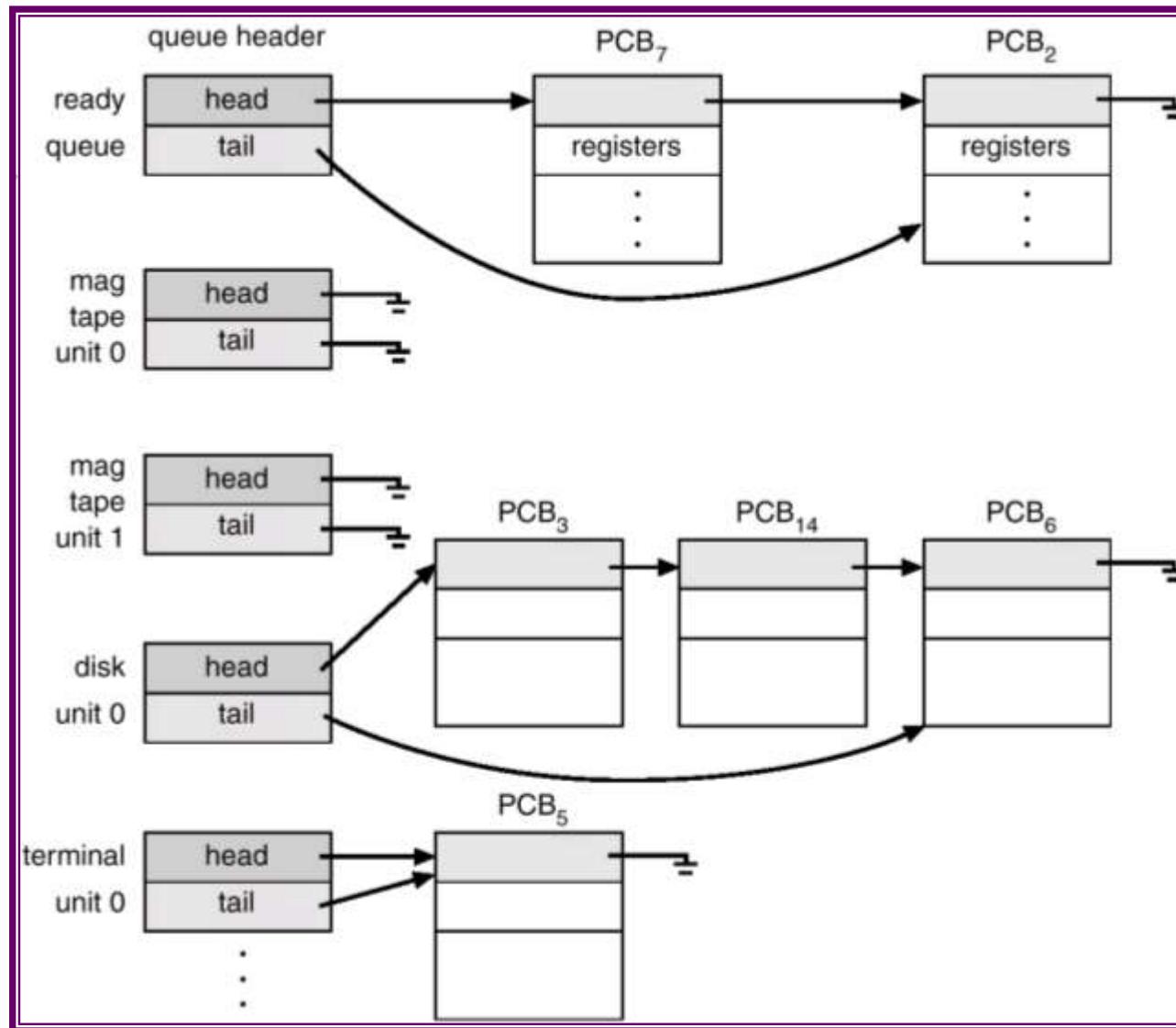
CPU Switch From Process to Process



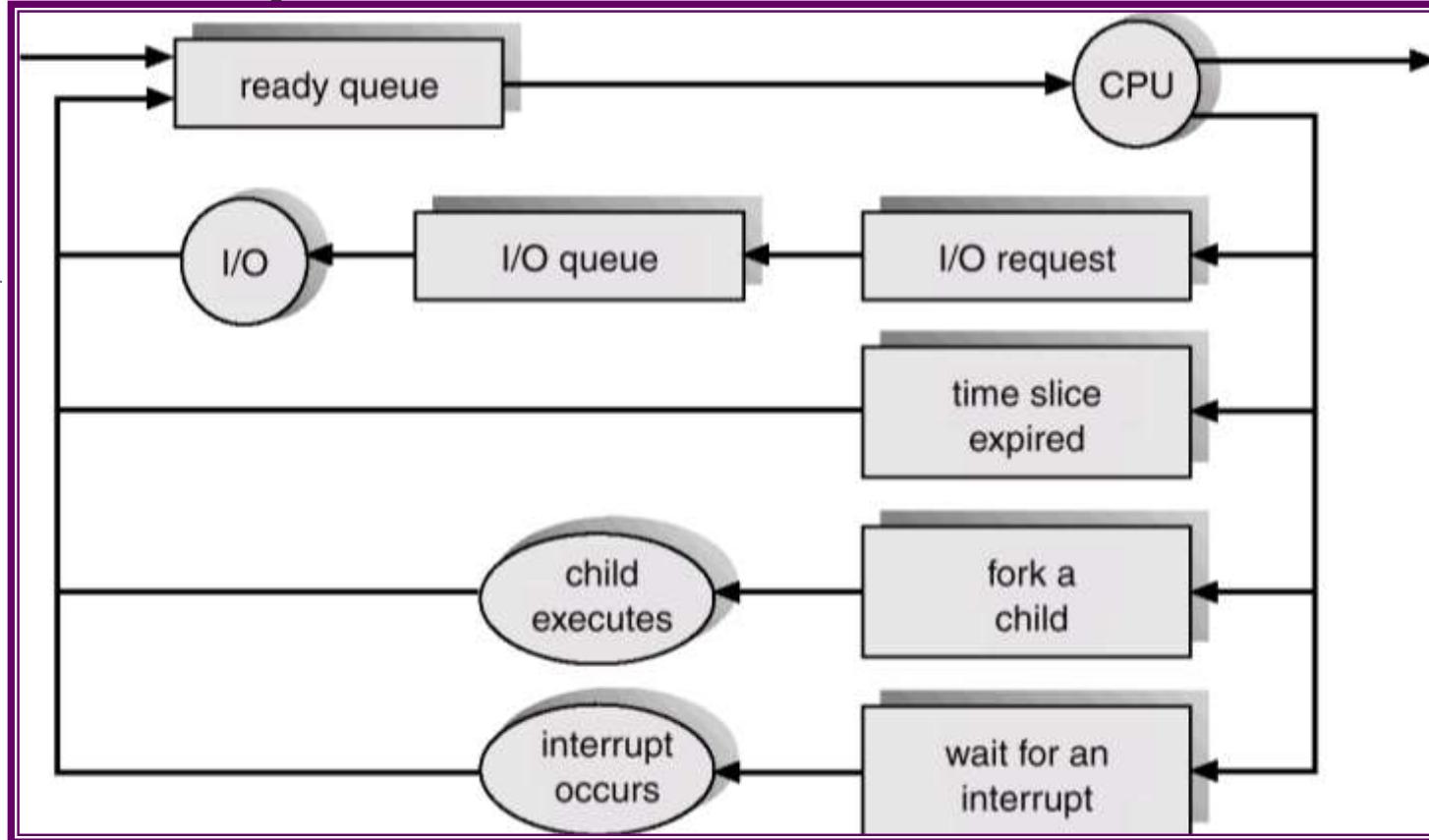
Process Scheduling Queues

- Scheduling is to decide which process to execute and when
- The objective of multi-program
 - ☞ To have some process running at all times.
- **Timesharing:** Switch the CPU frequently so that users can interact with the program while it is running.
- If there are many processes, the rest have to wait until CPU is free.
- Scheduling is to decide which process to execute and when.
- Scheduling queues:
 - ☞ Job queue – set of all processes in the system.
 - ☞ Ready queue – set of all processes residing in main memory, ready and waiting to execute.
 - ☞ Device queues – set of processes waiting for an I/O device.
 - ☞ Each device has its own queue.
- Process migrates between the various queues during its life time.

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling



- A new process is initially put in the ready queue
- Once a process is allocated CPU, the following events may occur
 - A process could issue an I/O request
 - A process could create a new process
 - The process could be removed forcibly from CPU, as a result of an interrupt.
- When process terminates, it is removed from all queues. PCB and its other resources are de-allocated.

Schedulers

- A process migrates between the various scheduling queues throughout its lifetime.
- The OS must select a process from the different process queues in some fashion. The selection process is carried out by a scheduler.
- In a batch system the processes are spooled to mass-storage device.
- **Long-term scheduler (or job scheduler)** – selects which processes should be brought into the ready queue.
 - ☞ It may take long time
- **Short-term scheduler (or CPU scheduler)** – selects which process should be executed next and allocates CPU.
 - ☞ It is executed at least once every 100 msec.
 - ☞ If 10 msec is used for selection, then 9 % of CPU is used (or wasted)
- The long-term scheduler executes less frequently.
- The long-term scheduler controls degree of multiprogramming.
- **Multiprogramming:** the number of processes active in the system.
- If MPL is stable: average rate of process creation= average departure rate of processes.

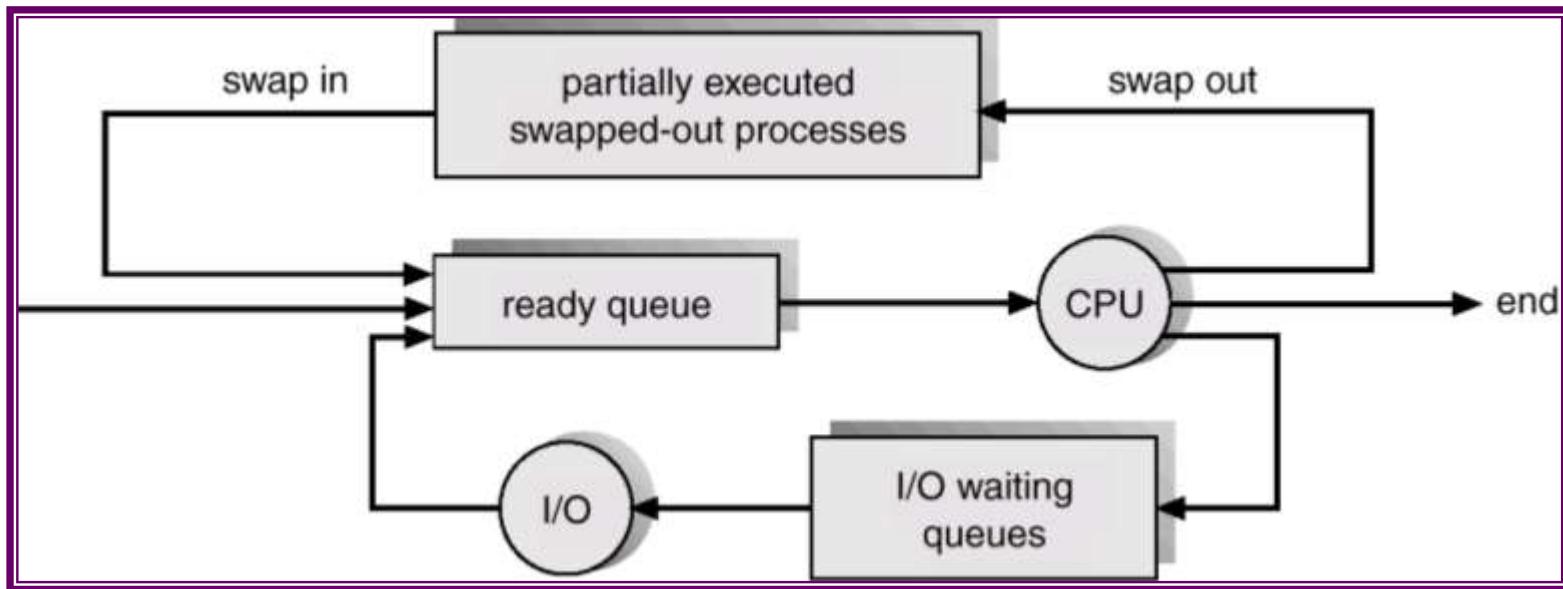
Schedulers...

- The long-term scheduler should make a careful selection.
- Most processes are either I/O bound or CPU bound.
 - ☞ **I/O bound process** spends more time doing I/O than it spends doing computation.
 - ☞ **CPU bound process** spends most of the time doing computation.
- The LT scheduler should select a good mix of I/O-bound and CPU-bound processes.
- Example:
 - ☞ If all the processes are I/O bound, the ready queue will be empty
 - ☞ If all the processes are CPU bound, the I/O queue will be empty, the devices will go unutilized and the system will be imbalanced.
- Best performance: best combination of CPU-bound and I/O-bound process.

Schedulers...

- Some OSs introduced a medium-term scheduler using swapping.
 - ☞ Key idea: it can be advantageous, to remove the processes from the memory and reduce the multiprogramming.
- **Swapping:** removal of process from main memory to disk to improve the performance. At some later time, the process can be reintroduced into main memory and its execution can be continued when it left off.
- Swapping improves the process mix (I/O and CPU), when main memory is unavailable.

Addition of Medium Term Scheduling



Context Switch

- Context switch is a task of switching the CPU to another process by saving the state of old process and loading the saved state for the new process
- **Context** of old process is saved in PCB and loads the saved context of old process.
- Context-switch time is **overhead**; the system does no useful work while switching.
- **New structures threads were incorporated.**
- Time dependent on hardware support.
 - ☞ 1 to 1000 microseconds

Operations on Processes: process creation

- A system call is used to create process.
 - ☞ Assigns unique id
 - ☞ Space
 - ☞ PCB is initialized.
- The creating process is called parent process.
- Parent process creates children processes, which, in turn create other processes, forming a tree of processes.
 - ☞ In UNIX **pagedaemon**, **swapper**, and **init** are children of root process. Users are children of **init** process.
- A process needs certain resources to accomplish its task.
 - ☞ CPU time, memory, files, I/O devices.
- When a process creates a new process,
 - ☞ Resource sharing possibilities.
 - ☒ Parent and children share all resources.
 - ☒ Children share a subset of parent's resources.
 - ☒ Parent and child share no resources.
 - ☞ Execution possibilities
 - ☒ Parent and children execute concurrently.
 - ☒ Parent waits until children terminate.

Process Creation (Cont.)

- ☞ Address space
 - ☞ Child duplicate of parent.
 - ☞ Child has a program loaded into it.
- UNIX examples
 - ☞ **fork** system call creates new process
 - ☞ **exec** system call used after a **fork** to replace the process' memory space with a new program.
 - ☞ The new process is a copy of the original process.
 - ☞ The exec system call is used after a fork by one of the two processes to replace the process memory space with a new program.
- DEC VMS
 - ☞ Creates a new process, loads a specified program into that process, and starts it running.
- WINDOWS NT supports both models:
 - ☞ Parent address space can be duplicated or
 - ☞ parent can specify the name of a program for the OS to load into the address space of the new process.

UNIX: fork() system call

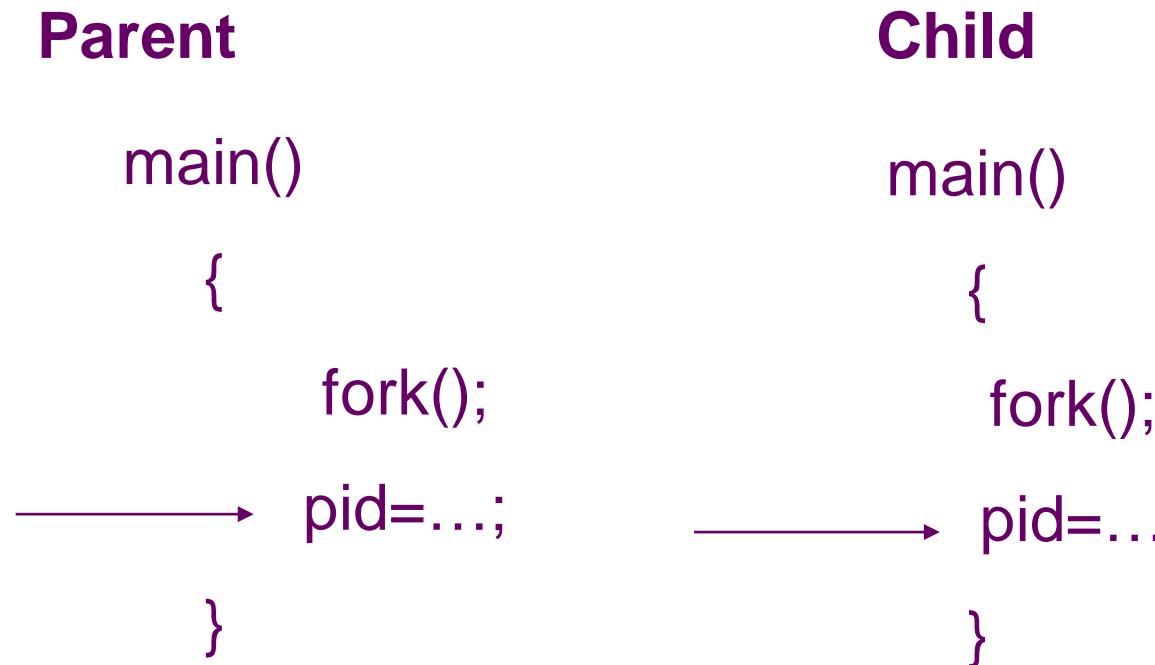
- fork() is used to create processes. It takes no arguments and returns a process ID.
- fork() creates a new process which becomes the child process of the caller.
- After a new process is created, both processes will execute the next instruction following the fork() system call.
- When checking the return value, we have to distinguish the parent from the child.
- fork()
 - ☞ If it returns a negative value, the creation is unsuccessful.
 - ☞ Returns 0 to the newly created child process.
 - ☞ Returns positive value to the parent.
- Process ID is of type pid_t defined in sys/types.h
- getpid() can be used to retrieve the process ID.
- The new process consists of a copy of address space of the original process.

UNIX: fork() system call

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#define MAX_COUNT 200
#define BUF_SIZE 100
void main(void)
{
    pid_t pid;
    int i;
    char buf[BUF_SIZE];
    fork();
    pid=getpid();
    for(i=1; i<=MAX_COUNT;i++)
    {
        sprintf(buf,"This line is from pid %d, value=%d\n",pid,i);
        write(1,buf,strlen(buf));
    }
}
```

UNIX: fork() system call

- If the fork() is executed successfully, Unix will
 - ☞ Make two identical copies of address spaces; one for the parent and one for the child.
 - ☞ Both processes start their execution at the next statement after the fork().



UNIX: fork() system call

```
#include <stdio.h>
#include <sys/types.h>
#define MAX_COUNT 200
void ChildProcess(void);
void ParentProcess(void);
#define BUF_SIZE 100
void main(void)
{
    pid_t pid;
    pid=fork();
    if (pid==0)
        ChildProcess();
    else
        ParentProcess();
}
```

```
Void ChildProcess(void)
{
    int i;
    for(i=1;i<=MAX_COUNT;i++)
    {
        printf(buf,"This line is from child, value=%d\n",i);
        Printf(" *** Child Process is done ***\n");
    }
}

Void ParentProcess(void)
{
    int i;
    for(i=1;i<=MAX_COUNT;i++)
    {
        printf(buf,"This line is from parent, value=%d\n",i);
        printf(" *** Parent Process is done ***\n");
    }
}
```

UNIX: fork() system call

Parent

```
void main(void)
{
    pid=fork();
    if (pid==0)
        ChildProcess();
    else
        ParentProcess();
}
Void ChildProcess(void)
{
}

Void ParentProcess(void)
{



}
```

PID=3456

Child

```
void main(void)
{
    pid=fork();
    if (pid==0)
        ChildProcess();
    else
        ParentProcess();
}
}

Void ChildProcess(void)
{
}

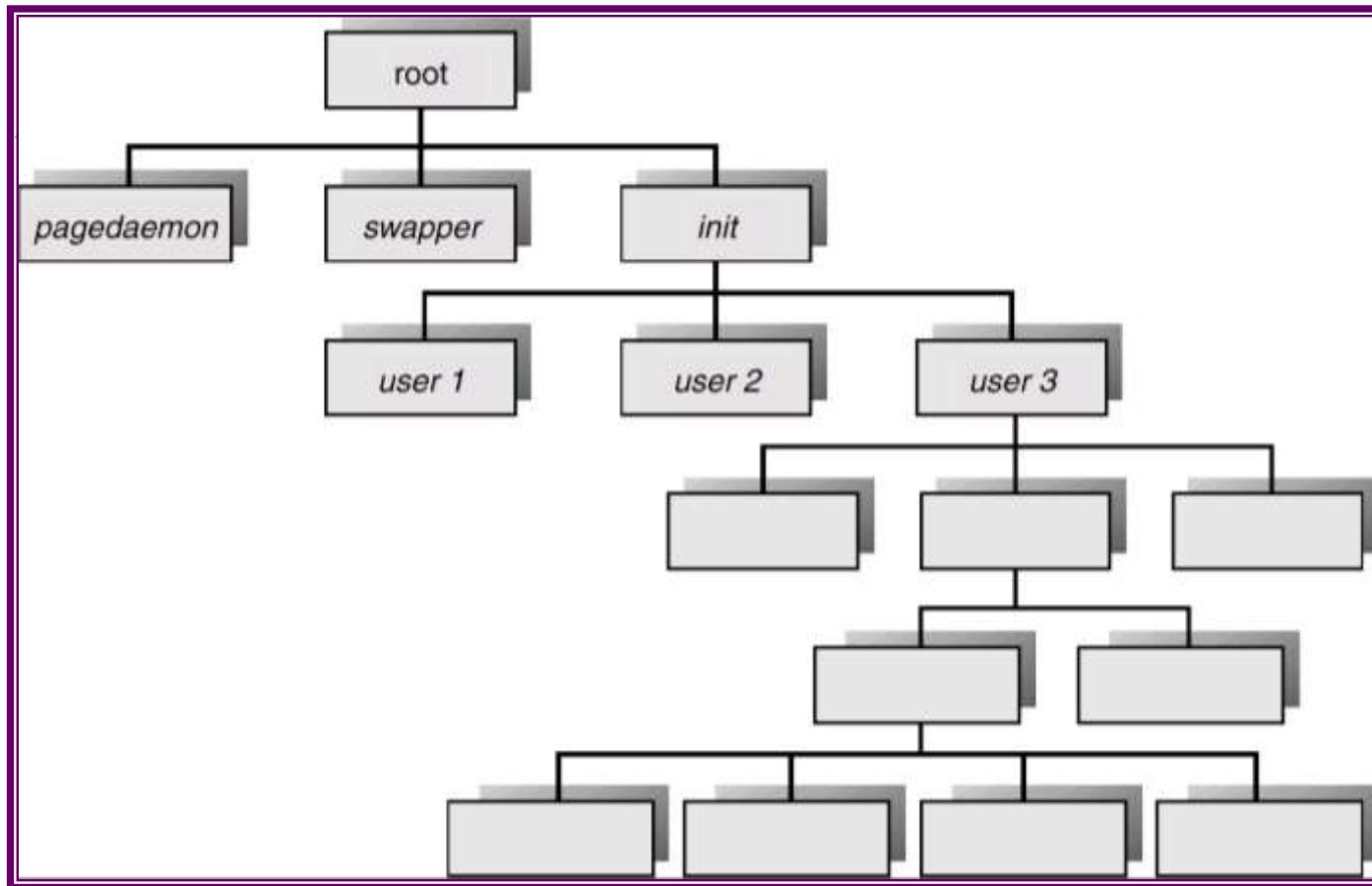
Void ParentProcess(void)
{



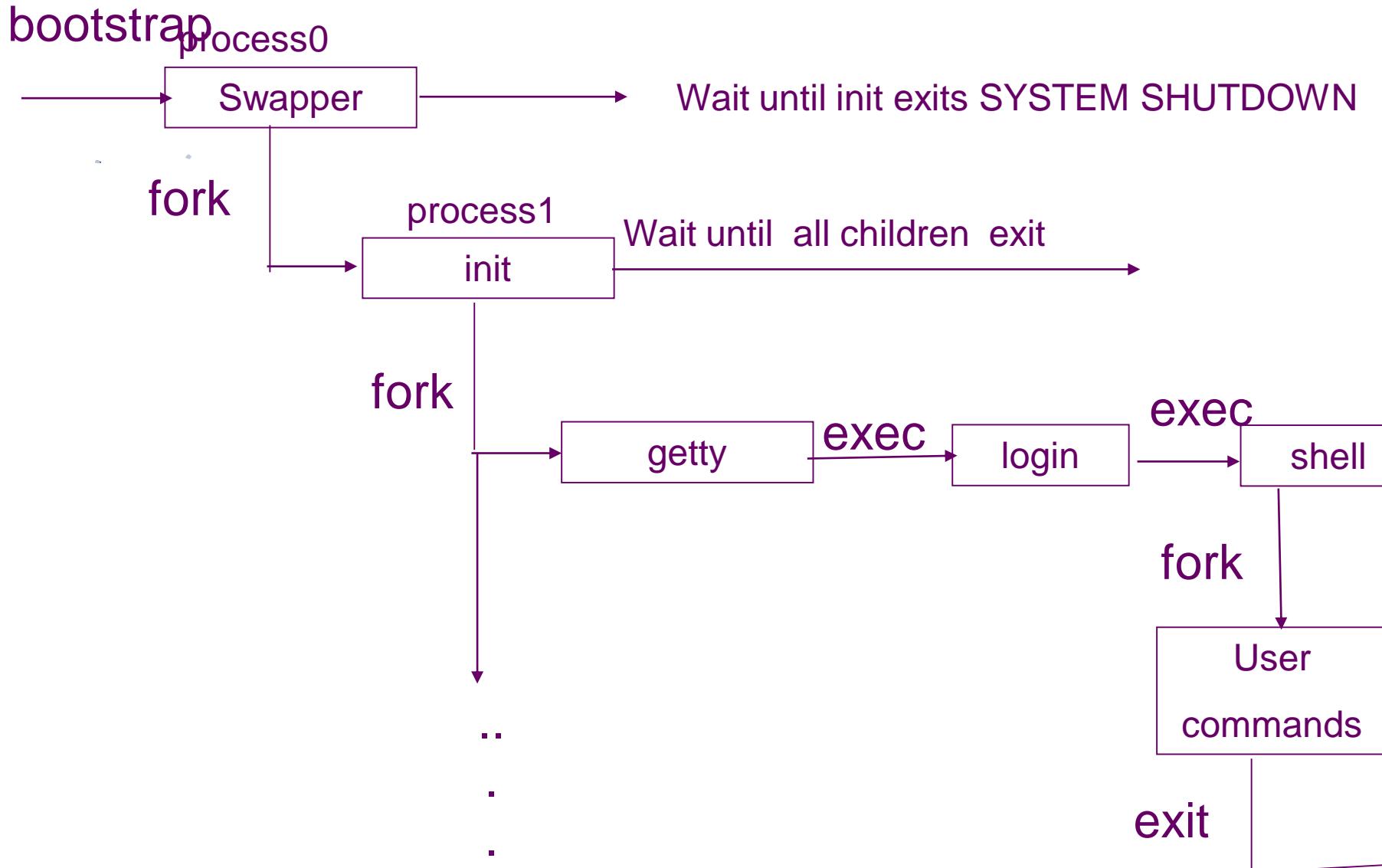
}
```

PID=0

Processes Tree on a UNIX System



UNIX system initialization



Process Termination

- Process executes last statement and asks the operating system to decide it (**exit**).
 - ☞ Output data from child to parent (via **wait**).
 - ☞ Process' resources are deallocated by operating system.
- Parent may terminate the execution of children processes (**abort**).
 - ☞ Child has exceeded allocated resources.
 - ☞ Task assigned to child is no longer required.
 - ☞ Parent is exiting.
 - ☞ Operating system does not allow child to continue if its parent terminates.
 - ☞ Cascading termination.

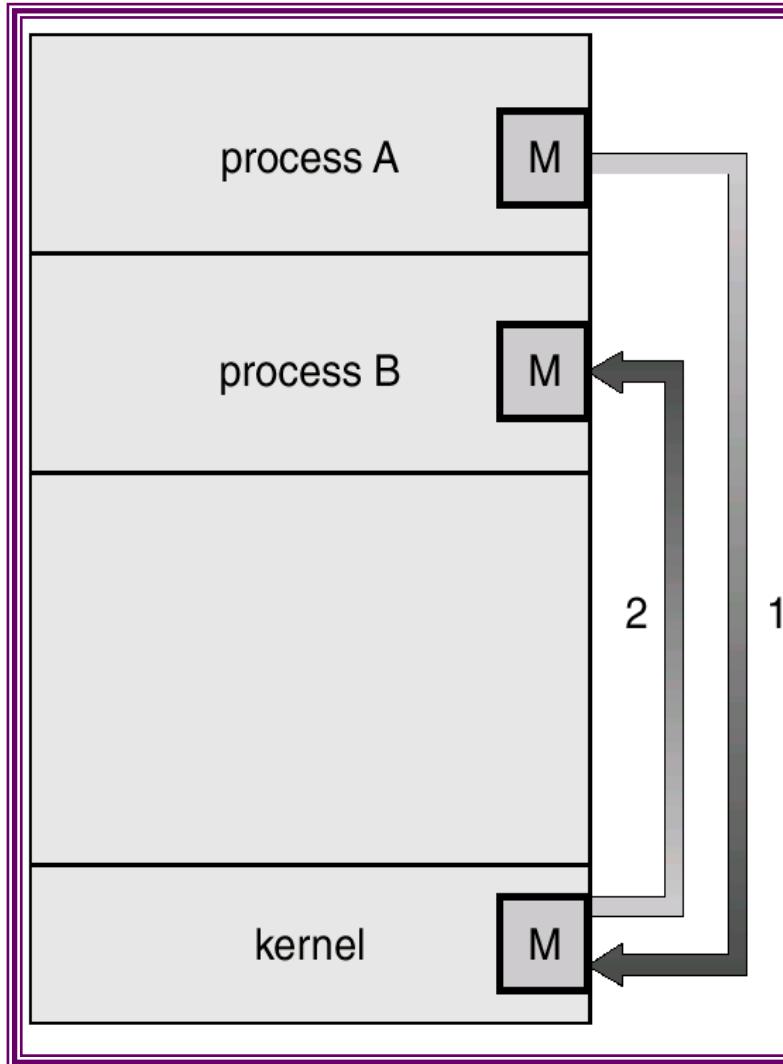
Cooperating Processes

- The processes can be independent or cooperating processes.
- *Independent* process **cannot** affect or be affected by the execution of another process.
- *Cooperating* process **can** affect or be affected by the execution of another process
- Advantages of process cooperation
 - ☞ Information sharing
 - ☞ Computation speed-up
 - ☞ Break into several subtasks and run in parallel
 - ☞ Modularity
 - ☞ Constructing the system in modular fashion.
 - ☞ Convenience
 - ☞ User will have many tasks to work in parallel
 - Editing, compiling, printing

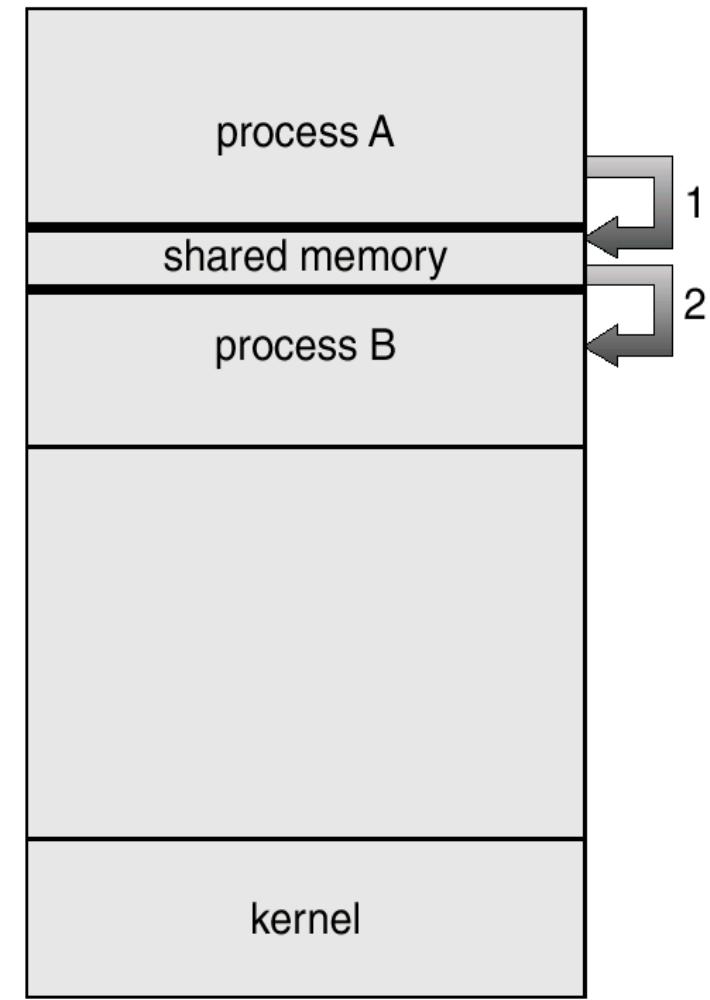
Inter-process Communication (IPC)

- IPC facility provides a mechanism to allow processes to communicate and synchronize their actions.
- Processes can communicate through **shared memory or message passing**.
 - ☞ Both schemes may exist in OS.
- The Shared-memory method requires communication processes to share some variables.
- The responsibility for providing communication rests with the programmer.
 - ☞ The OS only provides shared memory.
- Example: producer-consumer problem.

Communication Models



Msg Passing



Shared Memory

Producer-Consumer Problem: Shared memory

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process.
 - ☞ *unbounded-buffer* places no practical limit on the size of the buffer.
 - ☞  Producer can produce any number of items.
 - ☞  Consumer may have to wait
 - ☞ *bounded-buffer* assumes that there is a fixed buffer size.

Bounded-Buffer – Shared-Memory Solution

- Shared data

```
#define BUFFER_SIZE 10
Typedef struct {
    ...
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

in points to next free position.

out points to first full position.

Bounded-Buffer – Producer Process

```
item nextProduced;  
  
while (1) {  
    /* Produce an item in nextProduced */  
    while (((in + 1) % BUFFER_SIZE) == out)  
        ; /* do nothing */  
    buffer[in] = nextProduced;  
    in = (in + 1) % BUFFER_SIZE;  
}
```

Bounded-Buffer – Consumer Process

```
item nextConsumed;  
  
while (1) {  
    while (in == out)  
        ; /* do nothing */  
    nextConsumed = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
    /* Consume the item in  
nextConsumed */  
}
```

Inter-process Communication (IPC): Message Passing System

- **Message system** – processes communicate with each other without resorting to shared variables.
- If P and Q want to communicate, a communication link exists between them.
- OS provides this facility.
- IPC facility provides two operations:
 - ☞ **send(message)** – message size fixed or variable
 - ☞ **receive(message)**
- If P and Q wish to communicate, they need to:
 - ☞ establish a *communication link* between them
 - ☞ exchange messages via send/receive
- Implementation of communication link
 - ☞ physical (e.g., shared memory, hardware bus)
 - ☞ logical (e.g., logical properties)
 - We are concerned with logical link.

Fixed and variable message size

■ Fixed size message

- ☞ Good for OS designer
- ☞ Complex for programmer

■ Variable size messages

- ☞ Complex for the designer
- ☞ Good for programmer

Methods to implement a link

- Direct or Indirect communication
- Synchronous or asynchronous communication
- Automatic or explicit buffering

Direct Communication

- Processes must name each other explicitly:
 - ☞ **send** (P , message) – send a message to process P
 - ☞ **receive**(Q , message) – receive a message from process Q
- Properties of communication link
 - ☞ Links are established automatically.
 - ☞ A link is associated with exactly one pair of communicating processes.
 - ☞ Between each pair there exists exactly one link.
 - ☞ The link may be unidirectional, but is usually bi-directional.
- This exhibits both symmetry and asymmetry in addressing
 - ☞ Symmetry: Both the sender and the receiver processes must name the other to communicate.
 - ☞ Asymmetry: Only sender names the recipient, the recipient is not required to name the sender.
 - The send and receive primitives are as follows.
 - Send (P , message) – send a message to process P .
 - Receive(id, message) – receive a message from any process.
- **Disadvantages: Changing a name of the process creates problems.**

Indirect Communication

- The messages are sent and received from mailboxes (also referred to as ports).
- A mailbox is an object
 - ☞ Process can place messages
 - ☞ Process can remove messages.
- Two processes can communicate only if they have a shared mailbox.
- Operations
 - ☞ create a new mailbox
 - ☞ send and receive messages through mailbox
 - ☞ destroy a mailbox
- Primitives are defined as:
send(A, message) – send a message to mailbox A
receive(A, message) – receive a message from mailbox A

Indirect Communication

- Mailbox sharing
 - ☞ P_1 , P_2 , and P_3 share mailbox A.
 - ☞ P_1 , sends; P_2 and P_3 receive.
 - ☞ Who gets a message ?
- Solutions
 - ☞ Allow a link to be associated with at most two processes.
 - ☞ Allow only one process at a time to execute a receive operation.
 - ☞ Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.
- Properties of a link:
 - ☞ A link is established if they have a shared mailbox
 - ☞ A link may be associated with more than two boxes.
 - ☞ Between a pair of processes they may be number of links
 - ☞ A link may be either unidirectional or bi-directional.
- OS provides a facility
 - ☞ To create a mailbox
 - ☞ Send and receive messages through mailbox
 - ☞ To destroy a mail box.
- The process that creates mailbox is a owner of that mailbox
- The ownership and send and receive privileges can be passed to other processes through system calls.

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports).

- ☞ Each mailbox has a unique id.
 - ☞ Processes can communicate only if they share a mailbox.

- Example:

- Producer process:

```
repeat
```

```
....
```

```
    Produce an item in nextp
```

```
...
```

```
    send(consumer,nextp);
```

```
until false;
```

- Consumer process

- ☞ **repeat** **receive(producer, nextc);.... Consume the item in nextc ... until false;**

Synchronous or asynchronous

- Message passing may be either blocking or non-blocking.
- **Blocking** is considered **synchronous**
- **Non-blocking** is considered **asynchronous**
- **send** and **receive** primitives may be either blocking or non-blocking.
 - ☞ **Blocking send:** The sending process is blocked until the message is received by the receiving process or by the mailbox.
 - ☞ **Non-blocking send:** The sending process sends the message and resumes operation.
 - ☞ **Blocking receive:** The receiver blocks until a message is available.
 - ☞ **Non-blocking receive:** The receiver receives either a valid message or a null.

Automatic and explicit buffering

- A link has some capacity that determines the number of messages that can reside in it temporarily.
- Queue of messages is attached to the link; implemented in one of three ways.
 1. Zero capacity – 0 messages
Sender must wait for receiver (rendezvous).
 2. Bounded capacity – finite length of n messages
-Sender must wait if link full.
 3. Unbounded capacity – infinite length
Sender never waits.
- In non-zero capacity cases a process does not know whether a message has arrived after the send operation.
- The sender must communicate explicitly with receiver to find out whether the later received the message.
- Example: Suppose P sends a message to Q and executes only after the message has arrived.
- Process P:
 - send (Q, message) : send message to process Q
 - receive(Q,message) : Receive message from process Q
- Process Q
 - Receive(P,message)
 - Send(P,"ack")

Exception conditions

- When a failure occurs error recovery (exception handling) must take place.
- Process termination
 - ☞ A sender or receiver process may terminate before a message is processed. (may be blocked forever)
 - ☞ A system will terminate the other process or notify it.
- Lost messages
 - ☞ Messages may be lost over a network
 - ☞ Timeouts; restarts.
- Scrambled messages
 - ☞ Message may be scrambled on the way due to noise
 - ☞ The OS will retransmit the message
 - ☞ Error-checking codes (parity check) are used.

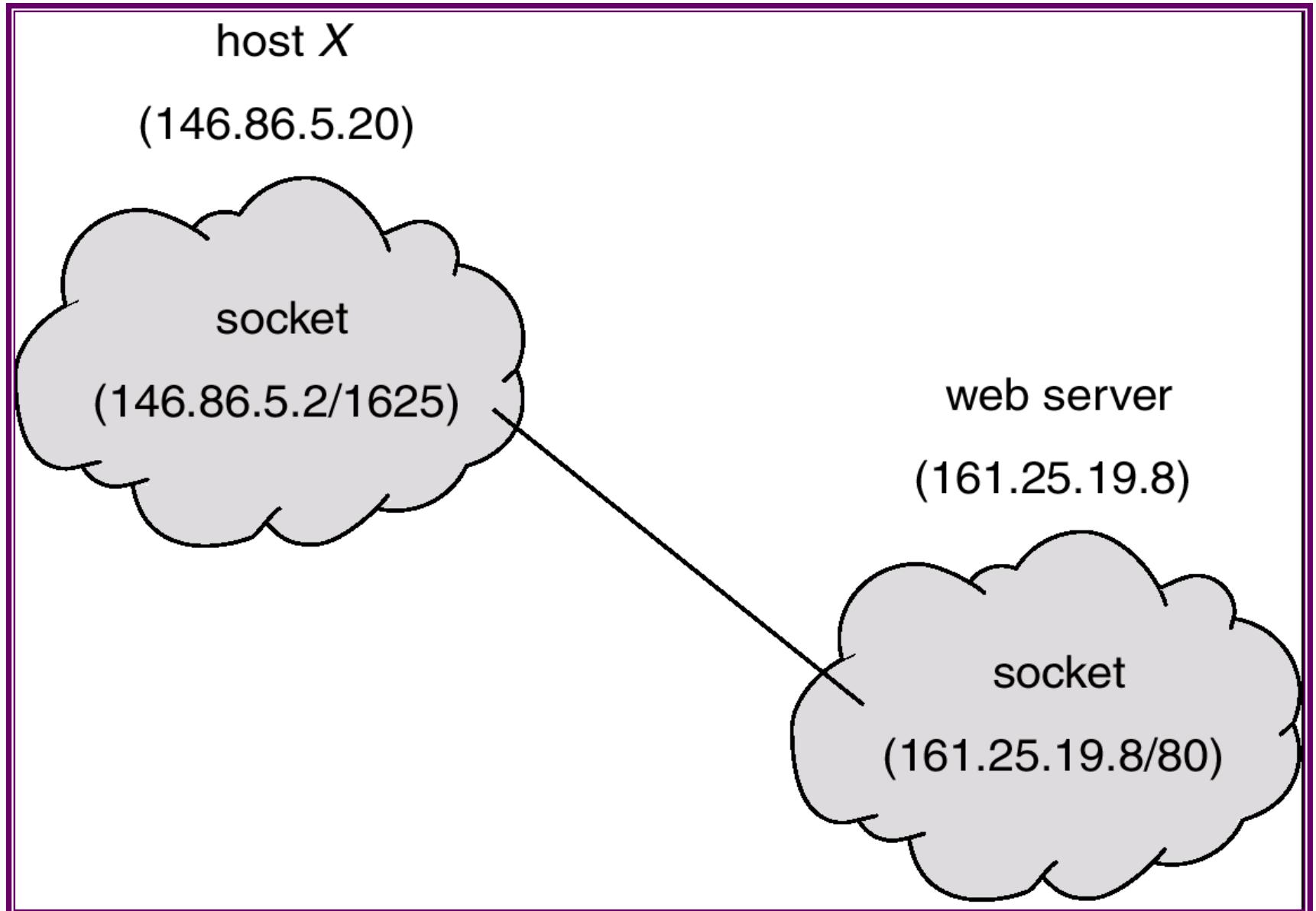
Client-Server Communication

- Sockets
- Remote Procedure Calls
- Pipes

Sockets

- A socket is defined as an *endpoint for communication*.
- A pair of processes communicating over a network employees a pair of sockets— one for each process.
- Socket: Concatenation of IP address and port
- The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
- **Servers implementing specific services listen to well-known ports**
 - ☞ **telnet server listens to port 80**
 - ☞ **ftp server listens to port 21**
 - ☞ **http server listens to port 80.**
- **The ports less than 1024 are used for standard services.**
- **The port for socket is an arbitrary number greater than 1024.**
- Communication consists between a pair of sockets.

Socket Communication



Remote Procedure Calls

- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems.
- **Stubs** – client-side proxy for the actual procedure on the server.
- The client-side stub locates the server and *marshals* the parameters.
 - ☞ Marshalling: Parameters must be marshaled into a standard representation.
 - ☞ The server-side stub receives this message, unpacks the marshaled parameters, and performs the procedure on the server.
- There are several issues
 - ☞ Local procedure call can fail and RPCs can fail and re-executed
 - ☞ One way to address this problem

Exactly once semantics (Local procedure calls)

- that processing of message will happen only **once**. It is difficult to implement in distributed system.

At most once semantics (Remote procedure calls)

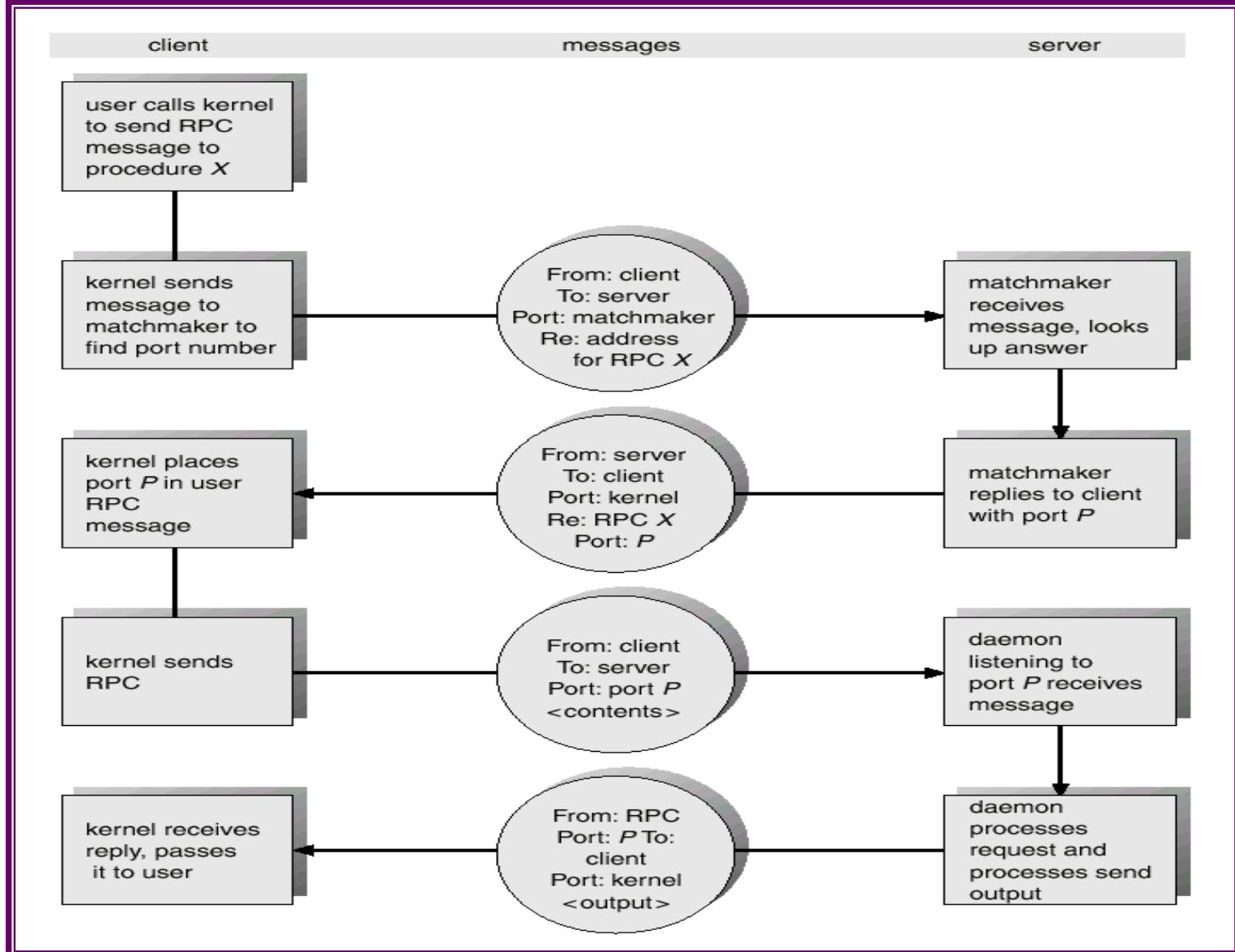
- when sending a message from a sender to a receiver there is no guarantee that a given message will be delivered.
- Server will act and send ack and identifies repeated messages. Client may send messages more than once until it receives ack.

- ☞ Binding issues: linking loading and execution

Fixed or rendezvous

- Applications: distributed file systems

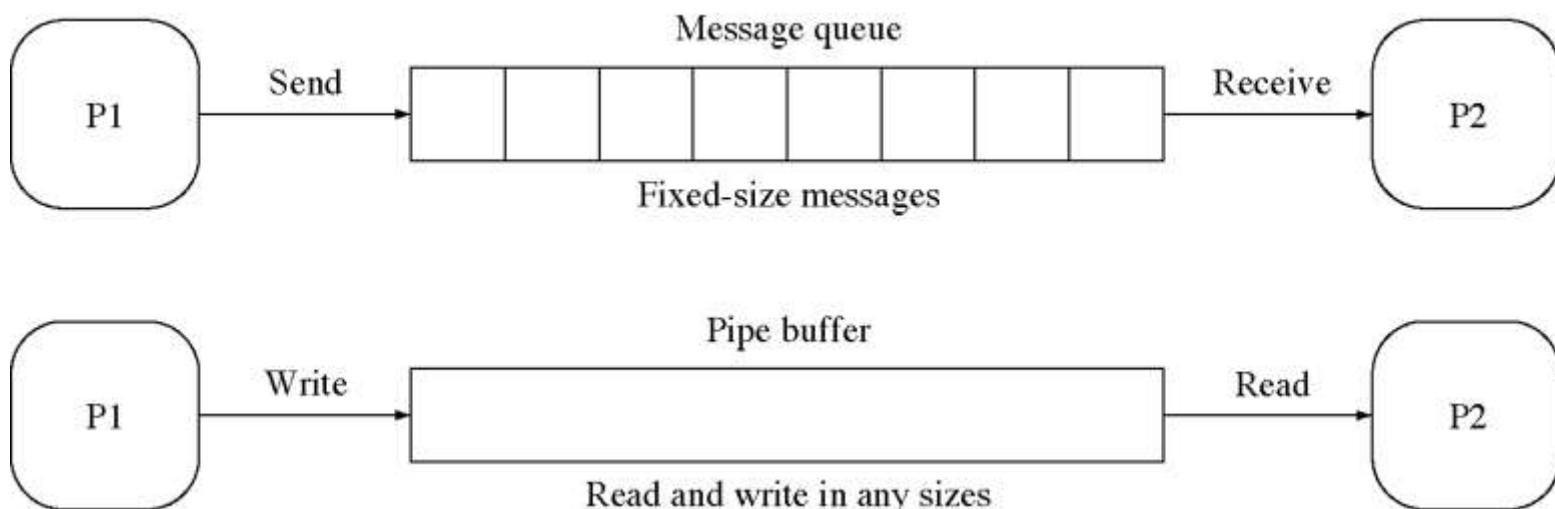
Execution of RPC



Pipes

- Pipe: another IPC mechanism
 - ☞ uses the familiar file interface
 - ☞ not a special interface (like messages)
- Connects an open file of one process to an open file of another process
 - ☞ Often used to connect the standard output of one process to the standard input of another process

Messages and pipes compared



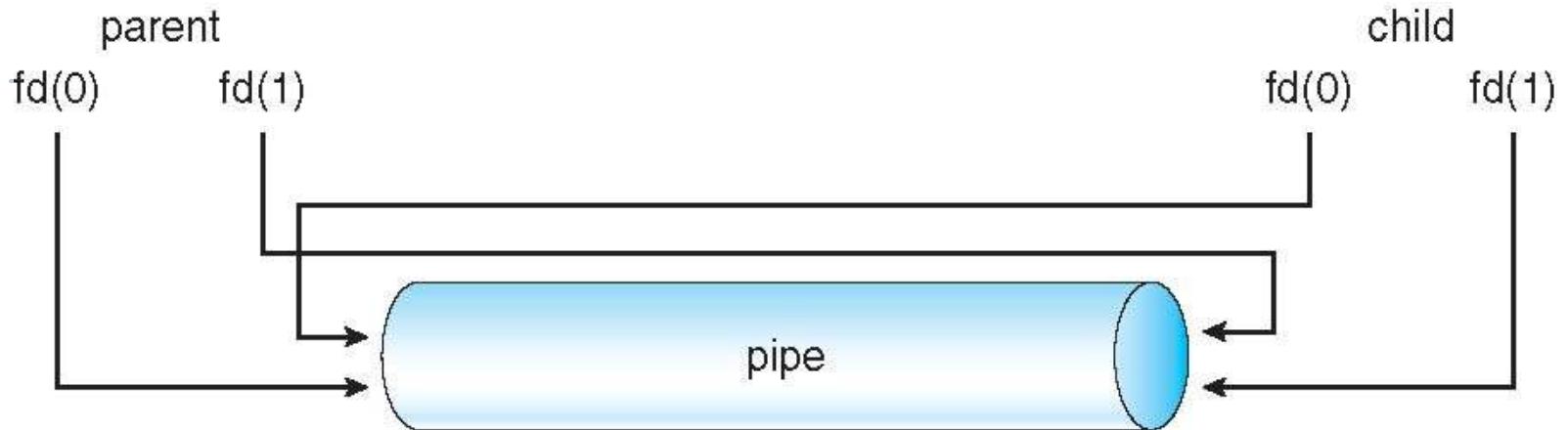
More about Pipes

- Acts as a conduit allowing two processes to communicate
 -
 -
- Issues
 - ☞ Is communication unidirectional or bidirectional?
 - ☞ In the case of two-way communication, is it half or full-duplex?
 - ☞ Must there exist a relationship (i.e. parent-child) between the communicating processes?
 - ☞ Can the pipes be used over a network?

Ordinary Pipes

- Ordinary Pipes allow communication in standard producer-consumer style
- Producer writes to one end (the *write-end* of the pipe)
- Consumer reads from the other end (the *read-end* of the pipe)
- Ordinary pipes are therefore unidirectional
- Require parent-child relationship between communicating processes
- Ordinary pipe can not be accessed from outside the process that creates it.
- Pipe is a special type of file.

Ordinary Pipes



- Unix function to create pipes: `pipe(int fd[])`
- `fd[0]` is the read end. `fd[1]` is write end.

Named Pipes

- Named Pipes are more powerful than ordinary pipes
- Communication is bidirectional
- No parent-child relationship is necessary between the communicating processes
- Several processes can use the named pipe for communication
- Provided on both UNIX and Windows systems

Pipes in Practice

- ls | more

Chapter 4: Multithreaded programming

- Introduction
- Processes and threads
- Multithreading Models
- Threading Issues
- Pthreads
- Solaris 2 Threads
- WINDOWS2000
- LINUX
- JAVA

References

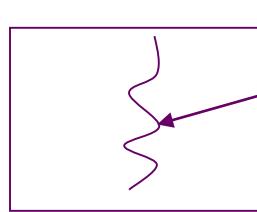
- Chapter 4 of Operating Systems Concepts, 8th edition, Silberschatz, Galvin, Gagne, Wiley
- Chapter 4, Operating systems, Fourth edition, William Stallings, Pearson education.

Threads: introduction

- Process has two characteristics
 - ☞ Resource ownership
 - ☒ Virtual address space is allocated
 - ☒ From time to time a process may be assigned main memory plus other resources such as I/O channels, I/O devices and files.
 - ☒ The OS performs a protection function to prevent unwanted interference between processes with respect to resources.
 - ☞ Scheduling/execution
 - ☒ The execution of a process follows an execution path (trace) through one or more programs.
 - ☒ The execution can be interleaved with that of other processes.
 - ☒ It is an entity that is scheduled and dispatched by the operating system.
- The two characteristics are independent
 - ☞ The unit of dispatching is called thread
 - ☞ The unit of resource ownership is called as a process or task.
- Examples: writing of file to disk
 - ☞ Compute on one batch of data while reading another batch of data.

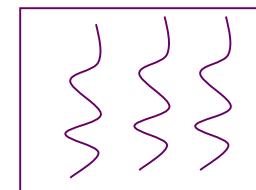
Multi-threading

- Multithreading refers to the ability on an operating system to support multiple threads of execution within a single process.
- Traditionally there is a single thread of execution per process.
 - ☞ Example: MSDOS supports a single user process and single thread.
 - ☞ UNIX supports multiple user processes but only support one thread per process.
- Multithreading
 - ☞ Java run time environment is an example of one process with multiple threads.
 - ☞ Examples of supporting multiple processes, with each process supporting multiple threads
 - ☞ Windows 2000, Solaris, Linux, Mach, and OS/2



Instruction
trace

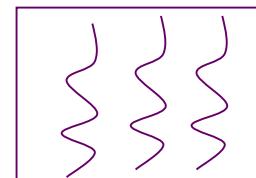
One process one thread



One process multiple threads



Multiple processes,
one thread per process



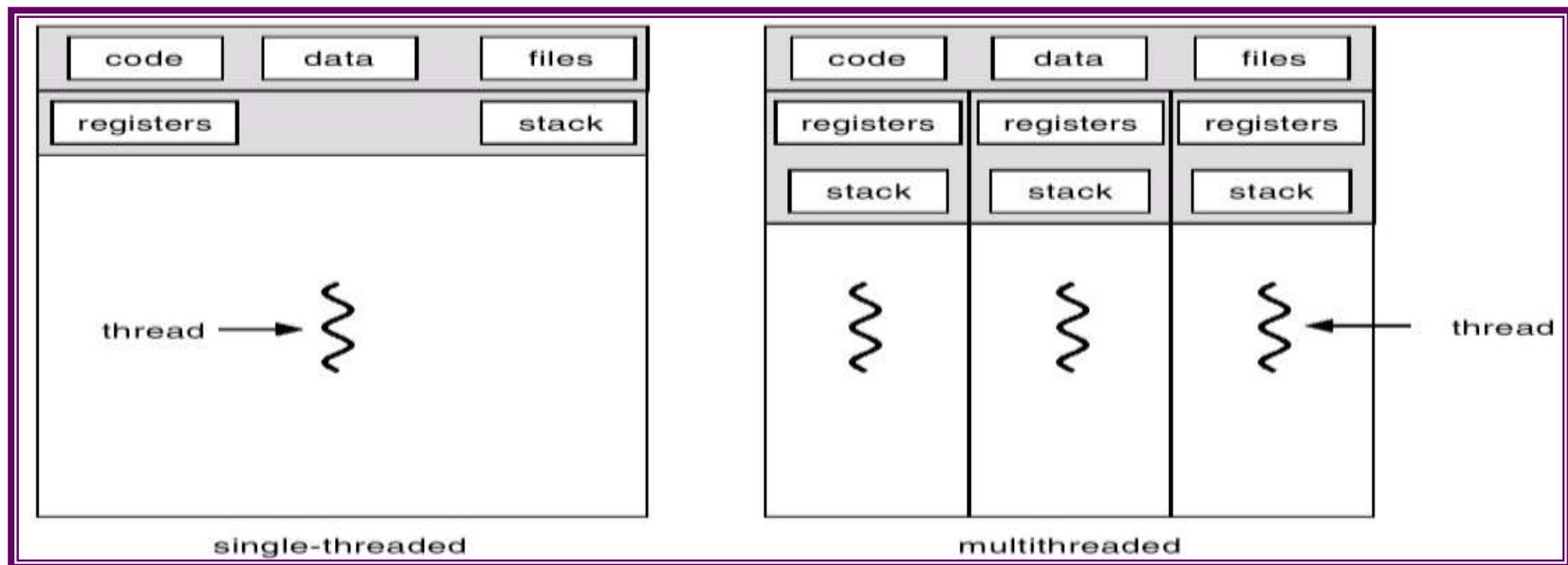
Multiple processes,
multiple threads per process

Process and Thread

- The following are associated with the process.
 - ☞ A virtual address space that holds the process image.
 - ☞ Protected access to processors, and others processes, files, and I/O resources (devices and channels)
- Within a process there may be one or more threads, each with the following
 - ☞ A thread execution state (Running, Ready, etc.)
 - ☞ Individual execution state; one way to view a thread is an independent program counter operating within a process.
 - ☞ Each thread has a control block, with a state (Running/Blocked/etc.), saved registers, instruction pointer
 - ☞ Separate stack
 - ☞ **Shares memory and files with other threads that are in that process**

Single and Multithreaded Processes

- Faster to create a thread than a process
- In a single threaded process model, the representation of a process includes its PCB, user address space, as well as user and kernel stacks.
 - When a process is running. The contents of these registers are controlled by that process, and the contents of these registers are saved when the process is not running.
- In a multi threaded environment
 - There is a single PCB and address space,
 - However, there are separate stacks for each thread as well as separate control blocks for each thread containing register values, priority, and other thread related state information.



Threads: Benefits

- All the threads of a process share the state and the resources of the process.
- They reside in the same address space and have access to the same data.
 - ☞ When one thread alters an item of data in memory, other threads see the results whenever they access the item.
- Key Benefits

☞ Responsiveness:

-  Allow program to continue to run even a part of it is blocked or performing lengthy operation.
 - Example: a multi-threaded application still allow user interaction in one thread while an image is being loaded in another thread.

☞ Resource sharing:

application can have different threads of activity all within the address space.

☞ Economy:

-  It takes less time to create a new thread in an existing process than to create a new process.
 - 10 times improvement in the speed: MAC over UNIX
 - In Solaris creating a process is 30 times slower than is creating a thread and context switching is 30 times slower.

Threads: Benefits...

☞ Economy...

- ☞ It takes less time to terminate a thread than a process.
- ☞ It takes less time to switch between two threads within the same process.
- ☞ Communication is easy with threads.
 - Communication between the processes required OS intervention.
 - Communication between threads can be through shared memory without OS intervention.

☞ Utilization of multi processor architectures

- ☞ Multithreading exploits multi-processor architectures very well
 - ☞ Each thread may be running parallel on a different processor.
 - ☞ A single threaded process can only run on single CPU, no matter how many are available.
 - ☞ Multithreading increases concurrency.
 - ☞ Single processor architecture only creates illusion of parallelism.
-
- If there is an application or function that should be implemented as a set of related units of execution, it is far more efficient to do as a collection of threads rather than a collection of processes.

Examples of use of threads

■ Single user multiprocessing system

☞ **Foreground/Background**

 In a spreadsheet program, one thread could display menus and read user input, while another thread executes user commands and updates the spreadsheet.

☞ **Asynchronous Processing** – Backing up in background

 Design of word processor write RAM buffer to disk once in every minutes.

☞ **Faster Execution** – Read one set of data while processing another set

 A multithreaded process can compute one batch of data while reading the next batch from a device.

 One a multiprocessor system multiple threads from the same process may be able to execute simultaneously.

☞ **Modular program structure**

 Programs that involve variety of activities or a variety of resources and destinations of input and output are easier to design.

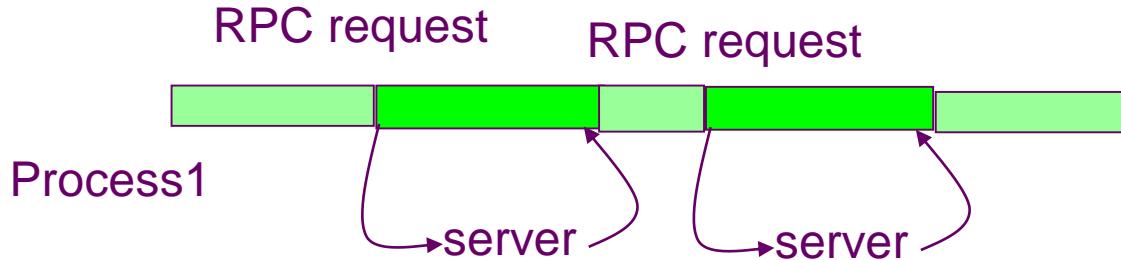
Thread Scheduling and dispatching

- Scheduling and dispatching is done at thread level.
- The actions that affect all the threads in the process should be managed at the process level.
- Suspension involves swapping of address space out of memory
 - ☞ Since all the threads in the process share the address space, all threads must enter the suspend state at the same time.
 - ☞ Termination of a process terminates all the threads in the process.

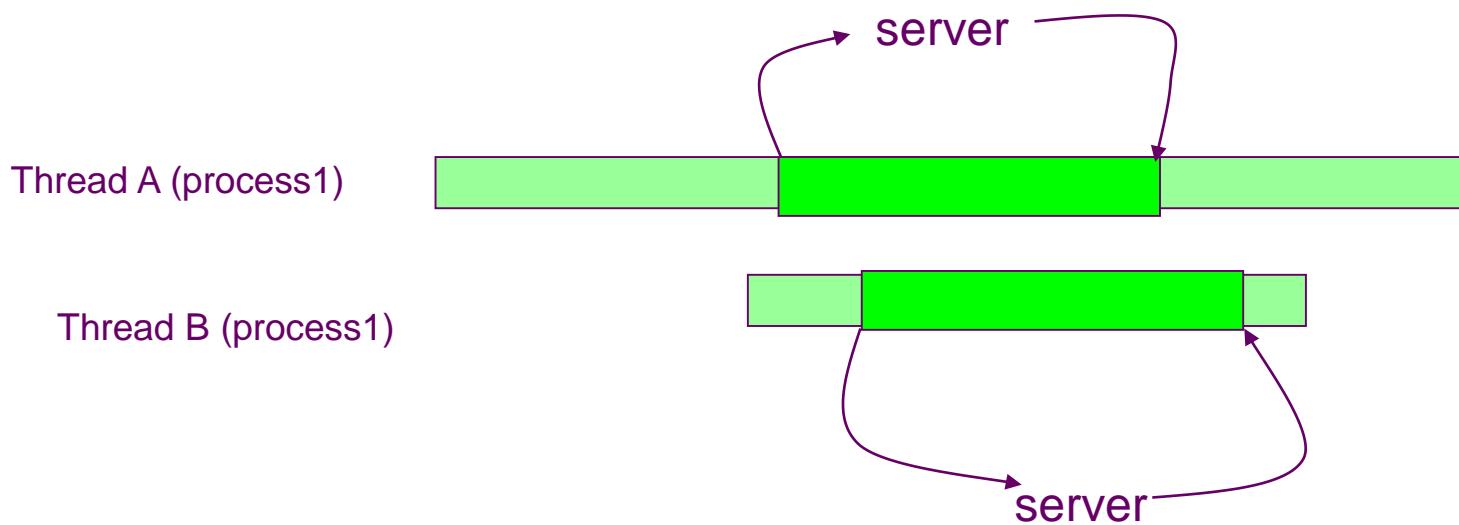
Thread Scheduling and dispatching

- Thread operations
 - ☞ *Spawn* – Creating a new thread
 - ☞ *Block* – Waiting for an event
 - ☞ *Unblock* – Event happened, start new
 - ☞ *Finish* – This thread is completed
- Generally a thread can block without blocking the remaining threads in the process
 - ☞ Allow the process to start two operations at once, each thread blocks on the appropriate event
- Must handle synchronization between threads
- System calls or local subroutines
 - ☞ Thread generally responsible for getting/releasing locks, etc.

Thread interleaving example



RPC using single thread



RPC using one thread per server

Multicore Programming

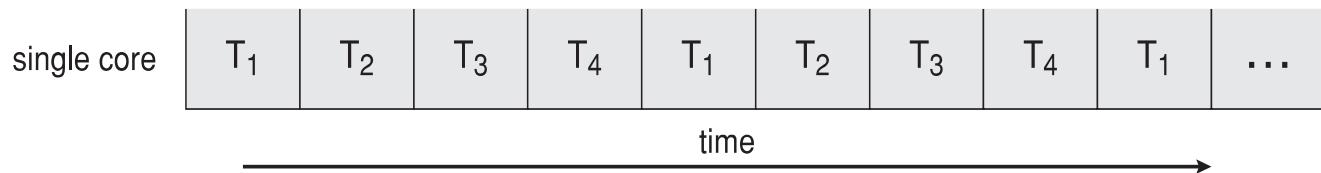
- **Multicore** or **multiprocessor** systems putting pressure on programmers, challenges include:
 - ☞ **Dividing activities**
 - ☞ **Load Balance: Equal load on every core**
 - ☞ **Data splitting**
 - ☞ **Data dependency**
 - ☞ **Testing and debugging**
- **Parallelism** implies a system can perform more than one task simultaneously
- **Concurrency** supports more than one task making progress
 - ☞ Single processor / core, scheduler providing concurrency

Multicore Programming (Cont.)

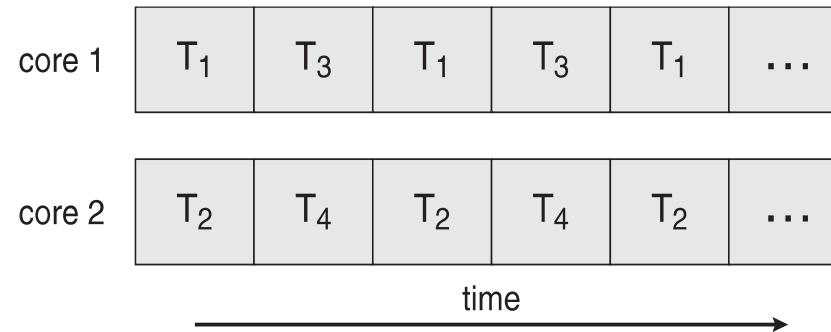
- Types of parallelism
 - ☞ **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
 - ☞ **Task parallelism** – distributing threads across cores, each thread performing unique operation
- As # of threads grows, so does architectural support for threading
 - ☞ CPUs have cores as well as ***hardware threads***
 - ☞ Consider Oracle SPARC T4 with 8 cores, and 8 hardware threads per core

Concurrency vs. Parallelism

- Concurrent execution on single-core system:



- Parallelism on a multi-core system:



Multithreading Models

- Many systems provide support for both user and kernel threads resulting different multi threading models. The following are common models.

☞ **Many-to-One**

☞ **One-to-One**

☞ **Many-to-Many**

Managing threads

- Support for threads can be provided at either user level or at kernel.

- ☞ **User-level threads**

- ☞ **Kernel-level threads**

User-level Threads

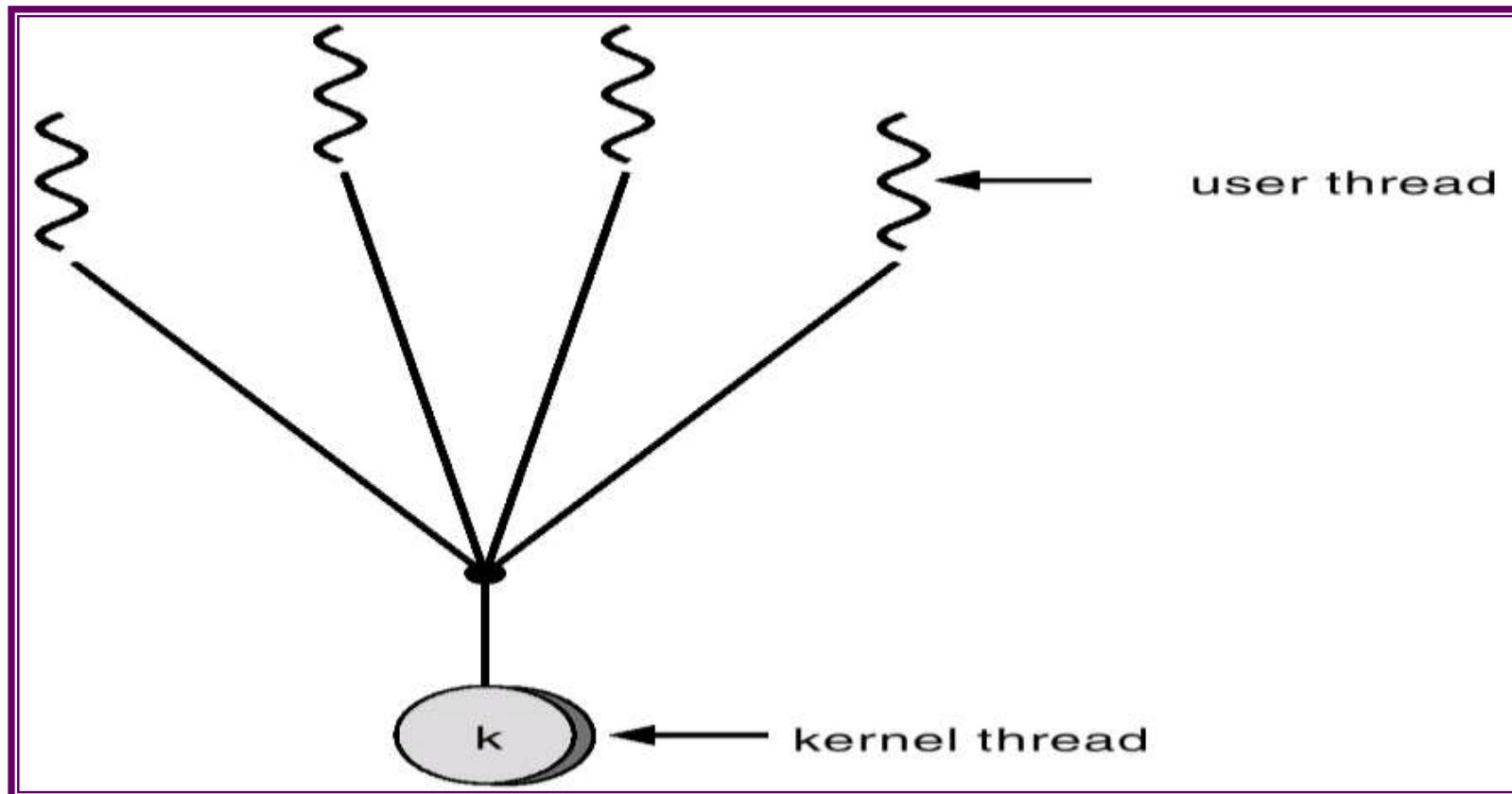
- Thread management is done by user-level threads library
- Application creates/manages all threads using a library
- System schedules the process as a unit
 - ☞ Scheduling, etc. is all in user space (faster)
 - ☞ Scheduling can be application specific
 - ☞ Does not require O.S. support
- Has problems with blocking system calls
- Cannot support multiprocessing
- Examples
 - POSIX *Pthreads*
 - Mach *C-threads*
 - Solaris *threads*

Kernel-level Threads

- Supported by the Kernel
- Kernel handles managing threads
- Easier to support multiple processors
- Kernel itself may be multithreaded
- Need user/kernel mode switch to change threads
- Examples
 - Windows 95/98/NT/2000
 - Solaris
 - Tru64 UNIX
 - BeOS
 - Linux
- Other Approaches
 - ☞ Each kernel thread may have multiple user threads

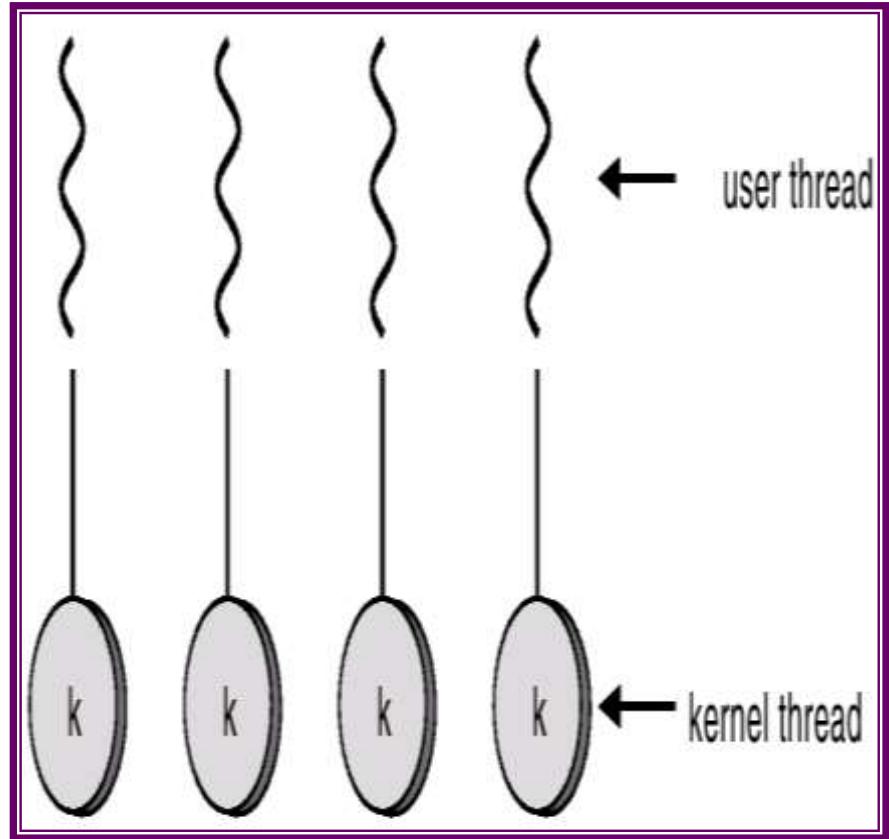
Many-to-One

- Many user-level threads mapped to single kernel thread.
- Thread management is done in user space.
- Entire process will block if a thread makes a blocking system call.
- Used on systems that do not support kernel threads.



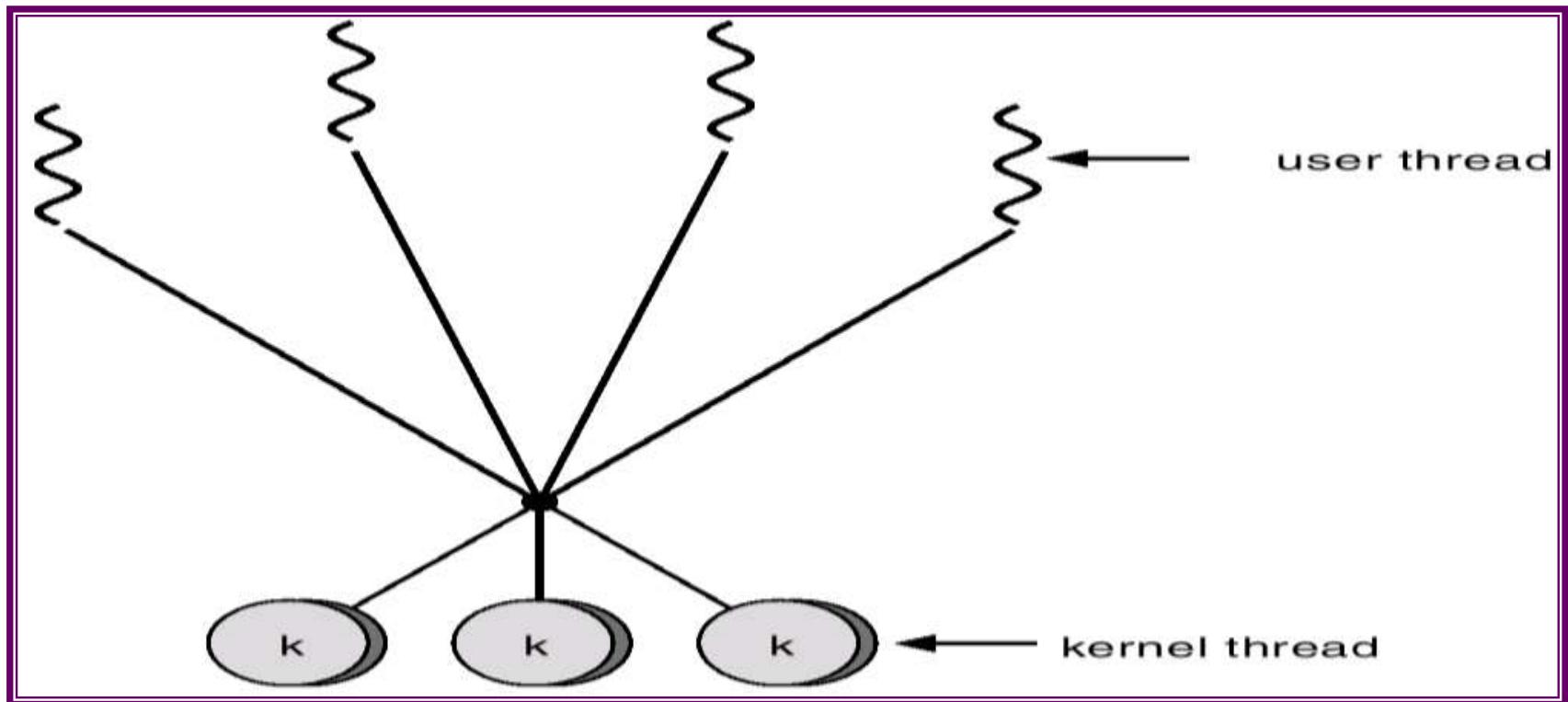
One-to-One

- Each user-level thread maps to kernel thread.
- Provides more concurrency than many-one model
- Allows other thread to run if a thread makes blocking system call
- Drawback: creating user thread requires creating the corresponding kernel thread.
- Most systems restrict the number of threads supported by the system
- Examples
 - Windows 95/98/NT/2000
 - OS/2

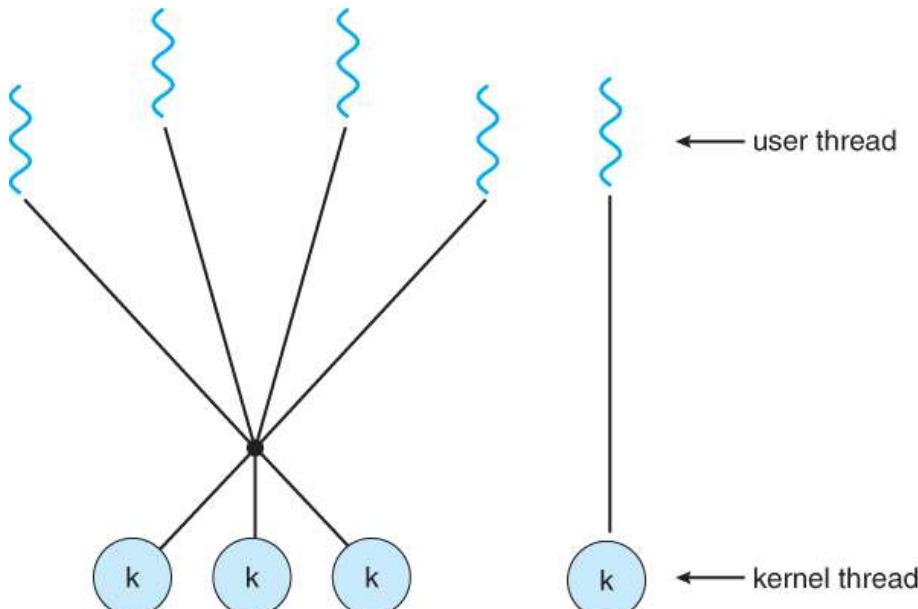


Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads.
- Allows the operating system to create a sufficient number of kernel threads.
- When thread performs a blocking system call, the kernel can schedule another thread for execution.
- Solaris 2 and Windows NT/2000 with the *ThreadFiber* package



Two-level Many to Many model



■ Allows

- ☞ multiplexing of many user-level threads to a small number or equal number of kernel threads, and
- ☞ User level thread to be bound to a kernel thread

Threading Issues

- Some of the issues to be considered for multi-threaded programs
 - ☞ Semantics of fork() and exec() system calls.
 - ☞ Thread cancellation.
 - ☞ Signal handling
 - ☞ Thread pools
 - ☞ Thread-specific data

Threading Issues...

■ Semantics of fork() and exec() system calls.

- ☞ In multithreaded program the semantics of the fork and exec systems calls change.
- ☞ If one thread calls fork, there are two options.
 - ☞ New process can duplicate all the threads or new process is a process with single thread
 - ☞ Some systems have chosen two versions of fork.

■ Thread cancellation.

- ☞ Task of terminating thread before its completion
 - ☞ Example: if multiple threads are searching database, if one gets the result others should be cancelled.
- ☞ **Asynchronous cancellation**
 - ☞ One thread immediately terminates the target thread.
- ☞ **Deferred cancellation**
 - ☞ The target thread can periodically check if it should terminate.

Threading Issues: Signal handling

- A signal is used in UNIX systems to notify a process that a particular event has occurred.
- A signal may be received synchronously or asynchronously.
- All signals follow specific patterns
 - ☞ A signal is generated by the occurrence of a particular event.
 - ☞ A generated signal is delivered to a process
 - ☞ On delivered, the signal must be handled.
- Examples: illegal memory access or division by zero.
- In single threaded program, it is easy, But in multithreaded program several options exist.
 - ☞ Deliver a signal to specific thread
 - ☞ Deliver a signal to every thread in the process
 - ☞ Deliver the signal to certain threads in the process.
 - ☞ Assign a specific thread to receive all signals for the process.
-  Solaris 2
 - It depends on the type of the signal
 - ☞ (<control>C) should be sent to all the threads.

Threading Issue: Thread pools

- Consider a scenario of multithreading a web server.
- Whenever a server receives a request, it creates a separate thread.
 - ☞ First problem is creating thread before hand and discarding it after completion.
 - ☞ Second problem is unlimited threads could exhaust system resources such as CPU time and main memory.
- One solution to this issue is **thread pools**
 - ☞ The idea is create a number of threads at process startup and place them in a pool, where they sit and wait for work.
 - ☞ If the pool has no available thread, the server waits until one is free.
- Benefits:
 - ☞ It is faster to service with an exiting thread than waiting to create a new thread.
 - ☞ A thread pool limits the number of threads at any point which is important on systems that can not support a large number of concurrent threads.

Threading Issues: Thread specific data

- Threads share the data
- Sharing is one of the benefits of multiprogramming.
- However, each thread might need its own copy of certain data in some circumstances.
- Such data is called thread-specific data.
 - ☞ For example in transaction processing system, each transaction may be assigned a unique number.
 - ☞ To associate each thread with its unique identifier, we could use thread specific data.

Threading Issues: Scheduler activations

- Communication between the kernel and the thread library.
- In many to many model, intermediate data structure is placed between kernel and user-level threads.
- To the user-thread library, the LWP appears to be a virtual processor on which the application can schedule a user thread to run.
- If the kernel thread blocks, the LWP blocks as well.
- Typically, the application requires many LWPs run efficiently.
 - ☞ One LWP is required for each concurrent blocking system call.
 - ☞ Suppose, five file reads occur concurrently, five LWPs are needed. If the process has four LWPs, then fifth must wait for one of the LWPs.
- For communication, the scheme “scheduler activation” is used.
- Scheduler activation: upcalls
 - ☞ The kernel informs application about certain events which is called **upcall** procedure
 - ☞ Upcalls are handled with thread library “**upcall handler**”
 - ☞ The kernel makes an **upcall** to the application regarding blocking of thread.
 - ☞ The upcall handler saves the state of blocking thread relinquishes the virtual processor on which blocking thread is running.
 - ☞ The upcall handler schedules another thread.
 - ☞ When waiting event occurs, the kernel makes another upcall to thread library informing it that the previously blocked thread is now eligible to run.

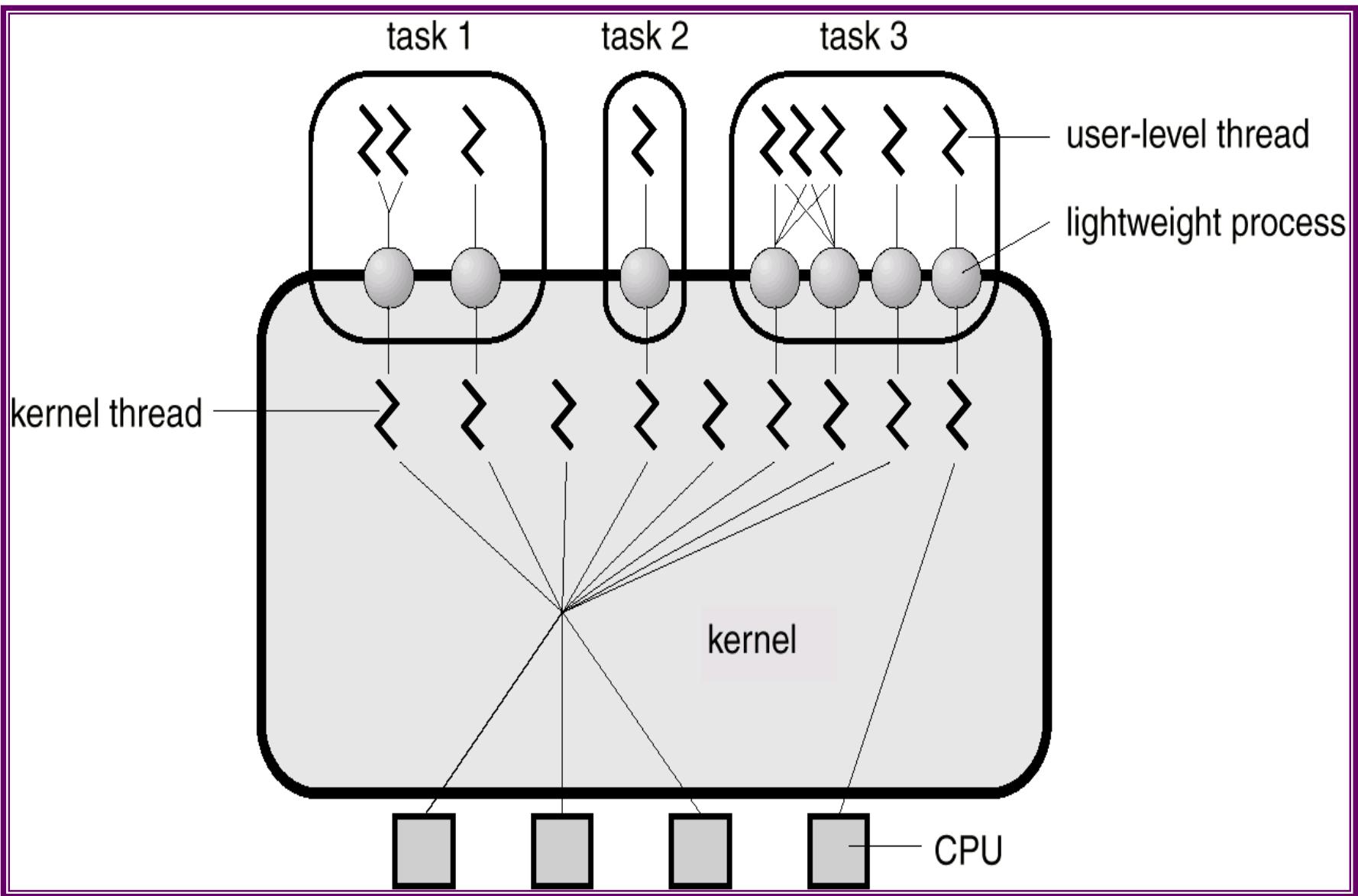
Pthreads

- a POSIX (Portable Operating System Interface for uniX) standard (IEEE 1003.1c) API for thread creation and synchronization.
- API specifies behavior of the thread library, implementation is up to development of the library.
- Common in UNIX operating systems.
 - ☞ MULTICS: Multiplexed Information and Computing System
 - ☞ UNICS: Uniplexed Operating and Computing System
 - ☞ UNICS → UNIX
 - ☞ LINUX: A UNIX- like operating system named after Linus Torvalds (Norwegian programmer) who initiated the work. LINUX is free and source code is open. Anyone can work on LINUX and post new code to improve it.

Solaris threads

- Implements Many-to-Many mapping
- Solaris uses four thread related concepts:
 - ☞ **Process:** Normal Unix process which includes user's address space, stack, and PCB
 - ☞ **User-level threads((ULTs):** Implemented through a thread library through the address of a process. These are invisible to OS.
 - ☞ **Lightweight processes (LWP):** mapping between ULTs and kernel threads.
 - ☞ Each LWP supports one or more ULTs and maps to one kernel thread. LWPs can be scheduled independently.
 - ☞ **Kernel threads:** These are fundamental entities that can be scheduled and dispatched to run on one of the system processors.

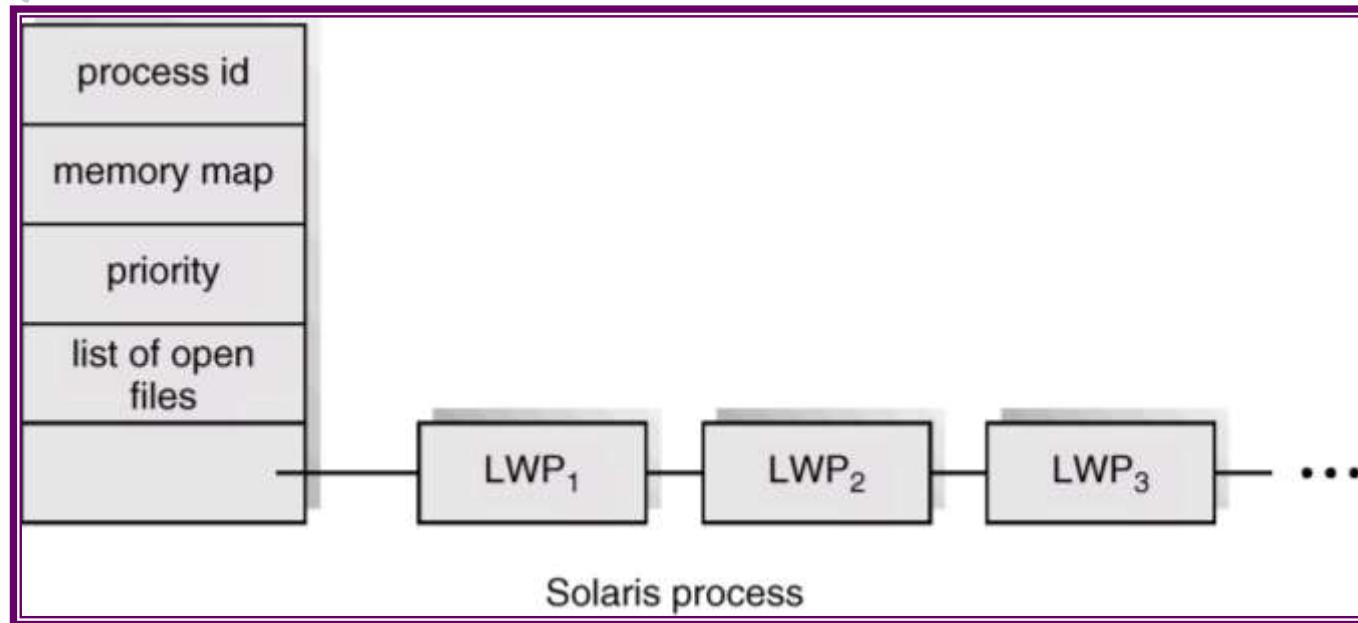
Solaris 2 Threads



Data structures of Solaris threads

- A user level thread contains a thread ID; register set, stack and its priority. None of these are kernel resources. All exists in user space.
- An LWP has a register set for the user-level thread it is running, as well as memory and account information.
 - ☞ A LWP is a kernel data structure and it resides in kernel space.
- A kernel thread has only a small data structure and a stack. The data structure includes
 - ☞ a copy of the kernel registers,
 - ☞ a pointer to the LWP to which it is attached,
 - ☞ priority and scheduling information.

Solaris Process



Windows 2000 Threads

- Implements the one-to-one mapping.
- Each thread contains
 - a thread id
 - register set
 - separate user and kernel stacks
 - private data storage area

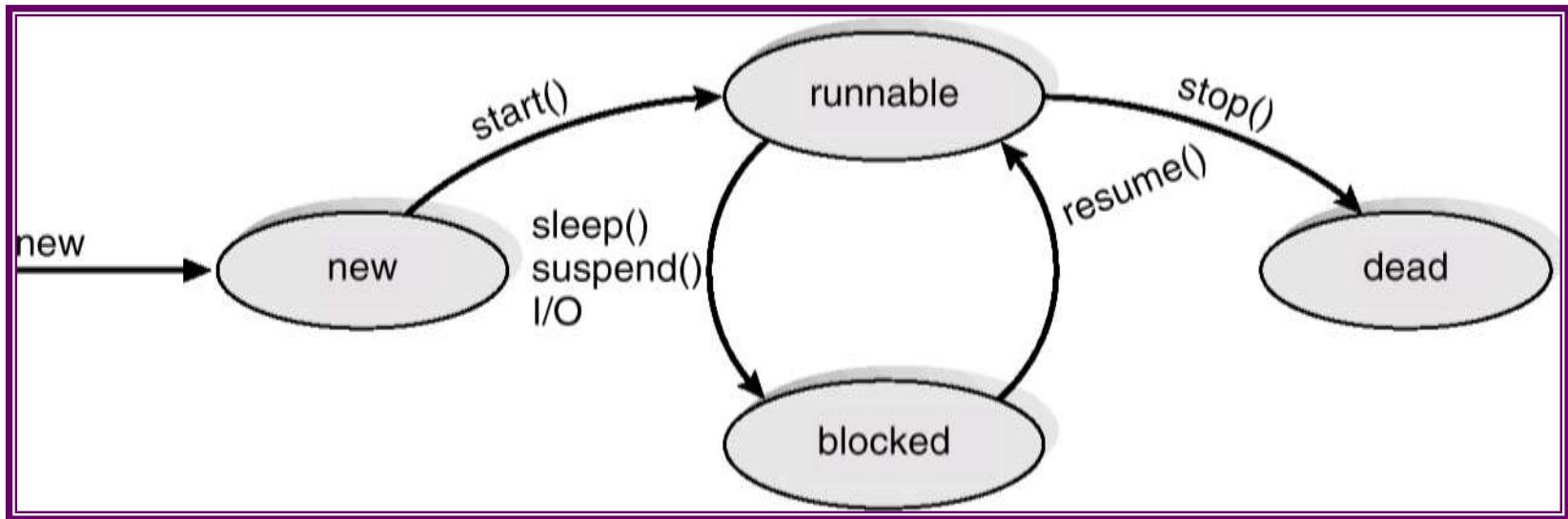
Linux Threads

- Linux refers to them as *tasks* rather than *threads*.
- Thread creation is done through `clone()` system call.
- `Clone()` allows a child task to share the address space of the parent task (process)

Java Threads

- Java threads may be created by:
 - ☞ Extending Thread class
 - ☞ Implementing the Runnable interface
- Java threads are managed by the JVM.
- Difficult to classify as a user or kernel threads.

Java Thread States



Process Synchronization

- Background
- The Critical-Section Problem
- Synchronization Hardware
- Semaphores
- Classical Problems of Synchronization
- Monitors
- Synchronization in Solaris 2 & Windows 2000

Process cooperation and synchronization

■ Process Synchronization

- ☞ ...mechanisms to ensure the orderly execution of cooperating processes that share a logical address space, so that data consistency is maintained.

■ Why do process cooperate ?

- ☞ Modularity: breaking up a system into several subsystems
 - ☞ E.g, an interrupt handler and device driver that need to communicate.
- ☞ Convenience: users might want to have several processes to share data
- ☞ Speed up: a single program is run as several sub-programs

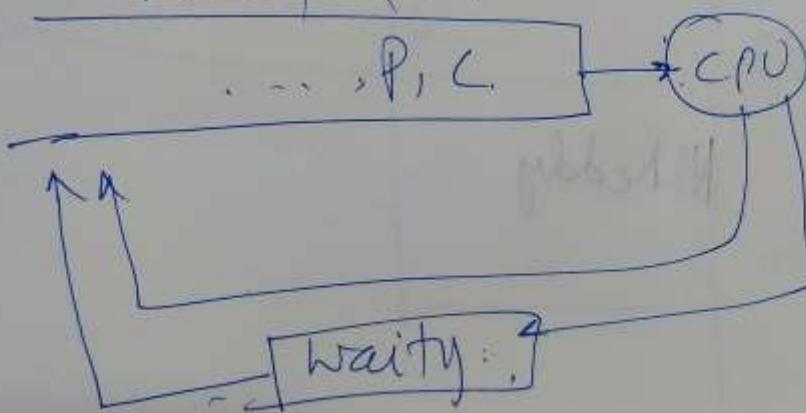
■ How do processes co-operate ?

- ☞ Communication abstraction: producers and consumers
 - ☞ Producers produce a piece of information
 - ☞ Customers use this information.

Background

- Concurrent access to shared data may result in data inconsistency.
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes.
 - ☞ Shared-memory solution to bounded-buffer problem allows at most $n - 1$ items in buffer at the same time. A solution, where all N buffers are used is not simple.
 - ☞ Suppose that we modify the producer-consumer code by adding a variable *counter*, initialized to 0 and incremented each time a new item is added to the buffer

Ready Queue



Bounded-Buffer

■ Shared data

```
#define BUFFER_SIZE 10
typedef struct {
    ...
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
int counter = 0;
```

Bounded-Buffer

■ Producer process

```
item nextProduced;  
  
while (1) {  
    while (counter == BUFFER_SIZE)  
        ; /* do nothing */  
    buffer[in] = nextProduced;  
    in = (in + 1) % BUFFER_SIZE;  
    counter++;  
}
```

Bounded-Buffer

■ Consumer process

```
item nextConsumed;
```

```
while (1) {
    while (counter == 0)
        ; /* do nothing */
    nextConsumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
    counter--;
}
```

Bounded Buffer

- Although, both the producer and consumer routines are correct separately they may not function correctly when executed concurrently.
- The statements

counter++;

counter--;

must be performed *atomically*.

- Atomic operation means an operation that completes in its entirety without interruption.

Bounded Buffer

- The statement “**count++**” may be implemented in machine language as:

register1 = counter

register1 = register1 + 1

counter = register1

- The statement “**count—**” may be implemented as:

register2 = counter

register2 = register2 - 1

counter = register2

Bounded Buffer

- If both the producer and consumer attempt to update the buffer concurrently, the assembly language statements may get interleaved.
- Interleaving depends upon how the producer and consumer processes are scheduled.

Bounded Buffer

- Assume **counter** is initially 5. One interleaving of statements is:

producer: **register1 = counter** ($register1 = 5$)

producer: **register1 = register1 + 1** ($register1 = 6$)

consumer: **register2 = counter** ($register2 = 5$)

consumer: **register2 = register2 - 1** ($register2 = 4$)

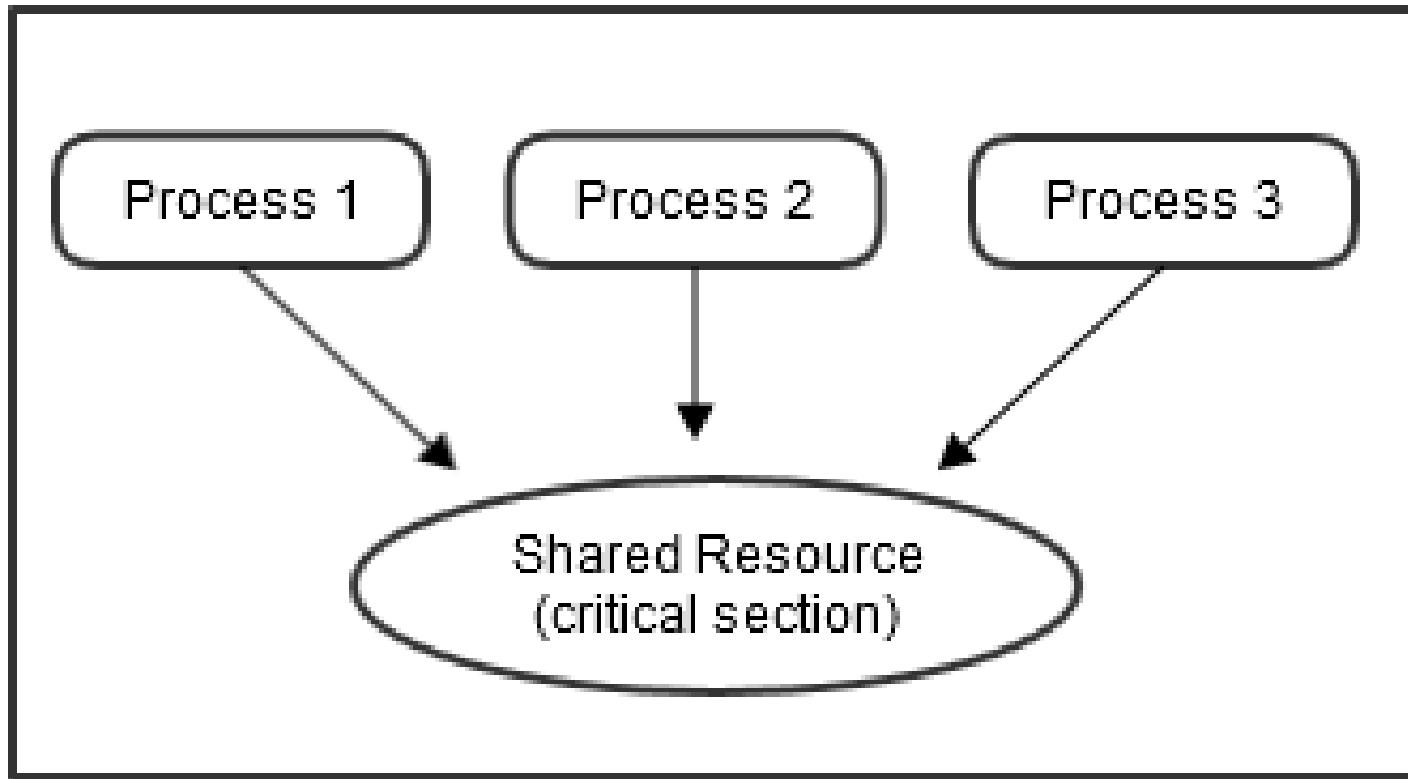
producer: **counter = register1** ($counter = 6$)

consumer: **counter = register2** ($counter = 4$)

- The value of **count** may be either 4 or 6, where the correct result should be 5.

Race Condition

- **Race condition:** The situation where several processes access – and manipulate shared data concurrently. The final value of the shared data depends upon which process finishes last.
- To prevent race conditions, concurrent processes must be **synchronized**.



The Critical-Section Problem

- n processes all competing to use some shared data
- Each process has a code segment, called *critical section*, in which the shared data is accessed.
- Problem – ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section.

Solution to Critical-Section Problem

- A solution to critical section problem must satisfy the following conditions.
 - ☞ **Mutual Exclusion.** If process P_i is executing in its critical section, then no other processes can be executing in their critical section.
 - ☞ **Progress.** At least one process requesting entry into CS will be able to enter it if there is no other process in it..
 - ☞ **Bounded Waiting.** No process waits indefinitely to enter CS once it has requested entry.
- Assume that each process executes at a nonzero speed
- No assumption concerning relative speed of the n processes.

Two approaches

- Several kernel-level processes may active at a time
 - ☞ Example: Data structure “List of open files”
- Kernel developers should ensure that OS is free from race conditions.
- Two approaches are used
- Non-preemptive kernel
 - ☞ A non-preemptive kernel does not allow a process running in the kernel mode to be preempted.
 - ☞ Kernel mode process runs until it exists kernel mode, blocks, or voluntarily yields the control of CPU
 - ☞ Free from race conditions
- Preemptive kernel
 - ☞ A preemptive kernel allows a process to be pre-empted while it is running in kernel mode.
 - ☞ Should be carefully designed
 - ☞ Difficult to design especially in SMP
- Why we prefer preemptive kernels ?
 - ☞ Suitable for realtime programming
 - ☞ More responsive as kernel mode process can not run for a longer time.
- WINDOWS XP, WINDOWS 2000, Prior to LINUX 2.6 are non-preemptive
- Solaris and IRIX are preemptive

Mutual exclusion: Software approaches

- Software approaches can be implemented
- Assume elementary mutual exclusion at the memory access level.
 - ☞ Simultaneous access to the same location in main memory are serialized in some order.
- Beyond this, no other support in the hardware, OS, programming language is assumed.

Two process solution

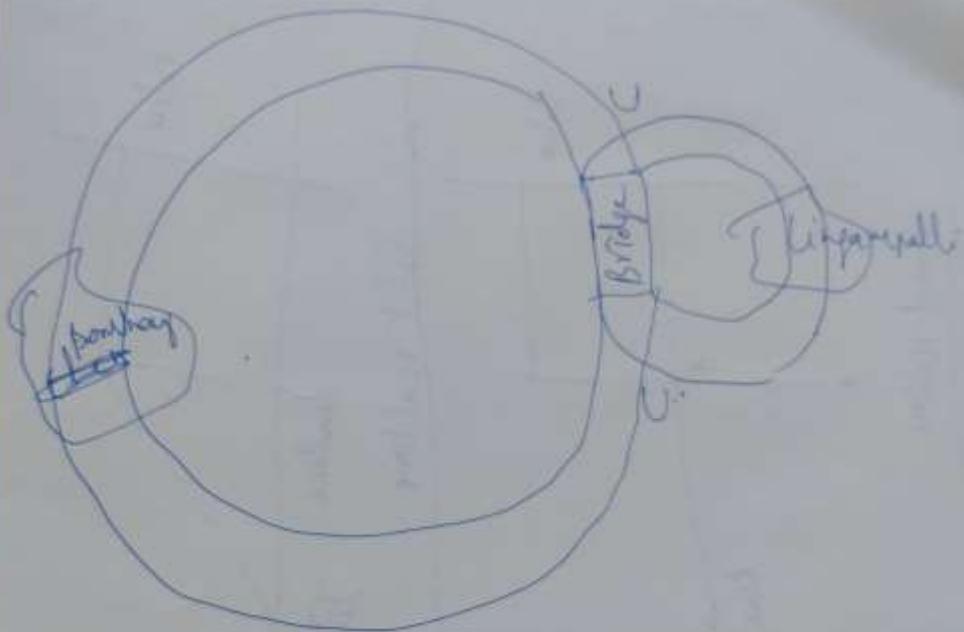
Initial Attempts to Solve Problem

Dekker's algorithm

- Reported by Dijkstra, 1965.
- Only 2 processes, P_0 and P_1
- General structure of process P_i (other process P_j)

```
do {  
    entry section  
    critical section  
    exit section  
    reminder section  
} while (1);
```

- Processes may share some common variables to synchronize their actions.



Algorithm 1

- Shared variables:

 - ☞ **int turn;**
initially **turn = 0**

- Turn variable

 - **P0**
while (turn != 0) ;
/* Do nothing */
critical section
turn = 1;
remainder section

 - **P1**
while (turn != 1);
/* Do nothing */
critical section
turn = 0;
remainder section

 - ☞ Shared variable *turn* indicates who is allowed to enter next, can enter if *turn = me*
 - ☞ On exit, point variable to other process
 - ☞ Deadlock if other process never enters

- +Satisfies mutual exclusion: Only one process can enter in CS

- -It does not satisfy the progress requirement, as it requires strict alternation of processes to enter CS.

- The pace of execution is dictated by slower process.

- If *turn=0*, P1 is ready to enter into CS, P1 can not do so, even though P0 may be in the RS.

- If one process fails in CS or RS, other process is blocked permanently.

Algorithm 2

- Problem with Alg1
 - ☞ It does not retain sufficient information about the state of each process.
 - ☞ Alg1 remembers only which process is allowed to enter the CS.
- To solve this problem, variable turn is replaced by **boolean flag[2]**; flag[0] is for P0; and flag[1] is for P1.
- Each process may examine the other's flag but may not alter it.
- When a process wishes to enter CS, it periodically checks other's flag until that flag is false (other process is not in CS)
- The process sets its own flag true and enters CS.
- When it leaves CS, it sets its flag to false.

Algorithm 2...

- initially **flag [0] = flag [1] = false.**

- **P0**

```
while ( flag[1] ) ;      while ( flag[0] )
/* Do nothing */        /* Do nothing */
flag[0] = true;          flag[1] = true;
critical section       critical section
flag[0] = false;         flag[1] = false;
```

- **P1**

```
while ( flag[0] ) ;
/* Do nothing */
flag[1] = true;
critical section
flag[1] = false;
```

- Mutual exclusion is satisfied.
- If one process fails outside CS the other process is not blocked.
- Sometimes, the solution is worst than previous solution.
 - ☞ It does not even **guarantee ME.**

- ☞ P0 executes the **while** statement and finds flag[1] set to false.
- ☞ P1 executes the **while** statement and finds flag[0] set to false.
- ☞ P0 sets flag[0] to true and enters its CS.
- ☞ P1 sets flag[1] to true and enters its CS.

Algorithm 3

- Interchange the first two statements.
- Busy Flag Modified

P0

```
flag[0] = true;  
while ( flag[1] );  
/* Do nothing */  
critical section  
flag[0] = false;
```

P1

```
flag[1] = true;  
while ( flag[0] );  
/* Do nothing */  
critical section  
flag[1] = false;
```

- Guarantees ME
- Both processes set their flags to true before either has executed the **while** statement, then each will think the other has entered CS causing deadlock.

Correct solution (1)

- Combining the key ideas of previous algorithms
- Dekker's Algorithm
 - ☞ Use *flags* for mutual exclusion, *turn* variable to break deadlock
 - ☞ Handles mutual exclusion, deadlock, and starvation
- Dekker's Algorithm
- Initial state: flag[0]=flag[1]=false; turn=1

P0

```
flag[0] = true;  
while ( flag[1] )  
    if (turn==1)  
    {  
        flag[0]=false;  
        while (turn==1)  
            /* do nothing */  
        flag[0]=true;  
    }  
/* critical section */  
turn=1;  
flag[0] = false;  
remainder section
```

P1

```
flag[1] = true;  
while ( flag[0] )  
if (turn==0)  
{  
    flag[1]=false;  
    while (turn==0)  
        /* do nothing */  
    flag[1]=true;  
}  
/* critical section */  
turn=0;  
flag[1] = false;  
remainder section
```

Correct solution (2)

- Peterson's Algorithm
- Initial state: flag[0]=flag[1]=false;

P0

```
flag[0] = true;  
turn = 1;  
while ( flag[1] && turn==1)  
    /* Do Nothing */;  
critical section  
flag[0] = false;  
remainder section
```

P1

```
flag[1] = true;  
turn = 0;  
while ( flag[0] && turn==0)  
    /* Do nothing */;  
critical section  
flag[1] = false;  
remainder section
```

Correct solution

- We need to show that
 - ☞ ME is preserved
 - ☞ The progress requirement is satisfied
 - ☞ The bounded-waiting requirement is met.
- **ME is preserved**
 - ☞ If both processes enter the CS both $\text{flag}[0]==\text{flag}[1]==\text{true}$
 - ☞ Both could not execute while loop successfully as turn is either 0 or 1.
- **Progress.**
 - ☞ While P1 exits CS it sets $\text{flag}[1]=\text{false}$, allowing P0 to enter CS.
 - ☞ P1 and P0 will enter the CS (Progress)
- **Bounded waiting:** P1 will enter the CS after at most one entry by P0 and vice versa.

Multi-process solution: Bakery Algorithm

Critical section for n processes

- Based on scheduling algorithm commonly used in bakeries.
 - ☞ On entering the store the customer receives the number.
 - ☞ The customer with the lowest number is served.
 - ☞ Customers may receive the same number, then the process with the lowest name is served first.
- Before entering its critical section, process receives a number. Holder of the smallest number enters the critical section.
- If processes P_i and P_j receive the same number, if $i < j$, then P_i is served first; else P_j is served first.
- The numbering scheme always generates numbers in increasing order of enumeration; i.e., 1,2,3,3,3,3,4,5...

Bakery Algorithm

- var: choosing: array[0...n-1] of boolean.
- Notation \leq lexicographical order (ticket #, process id #)
 - ☞ $(a,b) < (c,d)$ if $a < c$ or if $a = c$ and $b < d$
 - ☞ $\max(a_0, \dots, a_{n-1})$ is a number, k , such that $k \geq a_i$ for $i = 0, \dots, n - 1$
- Shared data

boolean choosing[n];

int number[n];

Data structures are initialized to **false** and **0** respectively

Bakery Algorithm

```
do {  
    choosing[i] = true;  
    number[i] = max(number[0], number[1], ..., number [n – 1])+1;  
    choosing[i] = false;  
    for (j = 0; j < n; j++) {  
        while (choosing[j]) ;  
        while ((number[j] != 0) && (number[j,j] < number[i,i])) ;  
    }  
    critical section  
    number[i] = 0;  
    remainder section  
} while (1);
```

- Consider Pi in its CS and Pk is trying to enter CS
- When Pk enters second while statement for j=i, it finds that
 - ☞ number[i] ≠ 0
 - ☞ (number[i],i) < (number[k],k)
 - ☞ So it waits until Pi leaves CS
- FCFS is followed.

Mutual exclusion: hardware solution

- In the uni-processor system, it is sufficient to prevent a process from being interrupted.

```
while (true){  
/* disable interrupts */  
/* Critical section */  
/* enable interrupts */  
/* remainder */  
}
```

- Since CS can not be interrupted ME is guaranteed.
- The efficiency decreases
- It can not work in multi-processor environments
 - ☞ More than one process is executing at a time.

Special machine instructions

- In multi-processor configuration, several processes share access to a common main memory.
- At the hardware level, access to a memory location excludes any other access to that same memory location.
- Processor designers have proposed several machine instructions to carry out two actions atomically (single cycle).
 - ☞ **Reading and writing**
 - ☞ **swapping**

Test and set instruction

- Test and modify the content of a word atomically

```
boolean testset (int i)
```

```
{  
    if (i==0)  
    {  
        i=1;  
        return true;  
    }  
    else  
    {  
        return false  
    }  
}
```

- This instruction sets the value of ‘i’, if the value=0 and returns true. Otherwise the value is not changed and false is returned.

Mutual Exclusion with Test-and-Set

- Shared data:

```
boolean lock = false;
```

- `void P(int i)`

```
do {
```

```
    while (TestAndSet(lock)==false)
```

```
        /* do nothing */;
```

critical section

```
    lock = false;
```

remainder section

```
}
```

```
void main()
```

```
{
```

```
lock=false;
```

```
parbegin(P1(), P(2),...,P(n));
```

```
}
```

Test-and-Set: Correctness

■ Mutual exclusion

- ☞ A shared variable lock is set to false
- ☞ The only process P_i that enters CS that finds lock as false and sets it to true.
- ☞ All other processes trying to enter CS go into a busy waiting mode and finds lock as false.
- ☞ When process leaves C it resests lock to false.
- ☞ When P_i exits lock is set to false so the next process P_j to execute instruction find test-and-set=false and will enter the CS.

■ Progress

- ☞ Trivially true

■ Unbounded waiting

- ☞ Possible since depending on the timing of evaluating the test-and-set primitive.
- ☞ Does not guarantee fairness.

Swap instruction

- Atomically swap two variables.

```
void swap(boolean &a, boolean &b) {  
    boolean temp = a;  
    a = b;  
    b = temp;  
}
```

Mutual Exclusion with Swap

- Shared data (initialized to **false**):

```
boolean lock;  
boolean waiting[n];
```

- Process P_i

```
do {  
    key = true;  
    while (key == true)  
        Swap(lock,key);  
    critical section  
    lock = false;  
    remainder section  
}
```

SWAP: Correctness

- Similar to Test-and-set
- Mutual exclusion
- Progress
 - ☞ Trivially true
- Unbounded waiting
 - ☞ Possible since depending on the timing of evaluating the test-and-set primitive.
 - ☞ Does not guarantee fairness.

Can we get bounded waiting ?

- Introduce a boolean array called waiting of size n and boolean variable key

- Entry

- ☞ waiting[i]:=true;
 - ☞ key:=true;
 - ☞ while (waiting[i] and key) do
 - ☞ key := test-and-set(lock);
 - ☞ waiting[i]:=false;
 - ☞ execute CRITICAL SECTION

- Exit

- ☞ Find the next process j that has waiting[j]=1 stepping through waiting.
 - ☞ Set waiting[j]:=false;
 - ☞ Process P_j immediately enter the CS.
 - ☞ If no process exists, set lock=false;

Can we get bounded waiting ?....

- Every (interested) Pi executes that test&set at least once.
- Pi enters the critical section provided:
 - ☞ Key is false in which case there is no process in CS.
- Or
 - ☞ If it was waiting, because waiting[i] was reset to false by the unique process that was blocking it in the critical section.
 - ☞ Either of the above events occur exactly once and hence mutual exclusion.

Properties of machine instruction approach

+ve

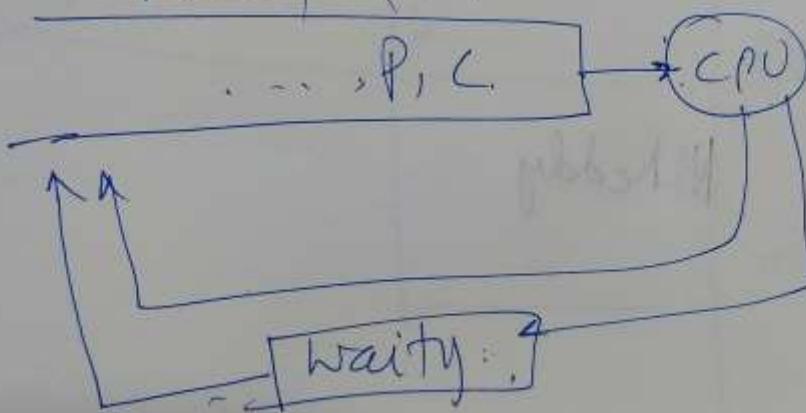
- ☞ Any number of processes
- ☞ Simple and easy
- ☞ Can support multiple CSs.

-ve

☞ **Busy waiting** is employed

- ☞ The process is waiting and consuming processor time.
- ☞ Starvation is possible.
 - ☞ The selection of waiting process is arbitrary.
- ☞ Deadlock is possible due to priority
 - ☞ P1 enters CS and interrupted by higher priority process P2 which is trying to enter CS.
 - ☞ P2 can not get CS unless P1 is out and P1 can not be dispatched due to low priority.

Ready Queue



Mutex Locks

- Test-set locks are also called mutex llocks
- Mutex means mutual exclusion.

Semaphores: Dijkstra; 1965

- Two and more processes can cooperate by means of simple signals, such that a process is forced to stop at a specified place until it has received a specific signal.
- For signaling, special variables called semaphores are used
- A semaphore is a synchronization tool.
- A semaphore is an integer variable that is accessed only through two standard atomic operations: **wait** and **signal**.
- To transmit a signal to semaphore S, a process executes the primitive *signal(S)* primitive.
- To receive a signal via semaphore S, the process executes *wait(S)* primitive.

Semaphores: Dijkstra 1965

Classical or first definition

- A semaphore is initialized to a non-negative value
- The **wait** operation decrements the semaphore value. If the integer value is negative the process waits.
- The **signal** operation increments the semaphore value. If the value is not positive, then process which is blocked by a wait operation gets the access to CS.
- **The wait and signal are assumed to be atomic.**
- Semaphore S – integer variable
- can only be accessed via two indivisible (atomic) operations
 - wait (S):*
while $S \leq 0$ do no-op;
 $S--;$
 - signal (S):*
 $S++;$

Critical Section of n Processes

- Shared data:

```
semaphore mutex; // initially mutex = 1
```

- Process P_i :

```
do {  
    wait(mutex);  
    critical section  
    signal(mutex);  
    remainder section  
} while (1);
```

- Modifications to the integer value of the semaphore in the wait and signal operations must be executed indivisibly.

Semaphore Implementation

- The classical definition requires busy waiting.
- While a process is in CS, the other process must loop continuously in the entry code.
- Busy waiting wastes CPU cycles.
- This type of semaphore is called **spinlock**: process spins while waiting for a lock.
 - ☞ Advantage of spinlock: no context switch
 - ☞ When locks are expected to be held for short times, spinlocks are useful.
- To overcome the need for busy waiting, we can modify the definition of the wait and signal semaphore operations.
- If a process executes wait operation and finds the semaphore operation is not positive, it must wait.
 - ☞ Rather than busy waiting it must **block** itself.
 - ☞ The **block** operation puts the process into waiting queue of semaphore and process is switched to waiting state.
- A process that is blocked waiting on a semaphore S, should be restarted when some other process executes signal operation.
- The process is restarted with **wakeup** operation.

Semaphore Implementation

- Define a semaphore as a record

```
typedef struct {  
    int value;  
    struct process *L;  
} semaphore;
```

- Assume two simple operations:

- ☞ **block** suspends the process that invokes it.
 - ☞ **wakeup(*P*)** resumes the execution of a blocked process **P**.

Implementation

- Semaphore operations now defined as

wait(S):

```
S.value--;
if (S.value < 0) {
    add this process to S.L;
    block;
}
```

signal(S):

```
S.value++;
if (S.value <= 0) {
    remove a process P from S.L;
    wakeup(P);
}
```

- Wait and signal operations are system calls.

Semaphore as a General Synchronization Tool

- Execute B in P_j only after A executed in P_i
- Use semaphore $flag$ initialized to 0
- Code:

| | | |
|----------------|--------------|-------|
| P_i | | P_j |
| : | | : |
| A | $wait(flag)$ | |
| $signal(flag)$ | | B |

Deadlock and Starvation

- **Deadlock** – two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes.
- Let S and Q be two semaphores initialized to 1

| | |
|--------------|--------------|
| P_0 | P_1 |
| $wait(S);$ | $wait(Q);$ |
| $wait(Q);$ | $wait(S);$ |
| \vdots | \vdots |
| $signal(S);$ | $signal(Q);$ |
| $signal(Q)$ | $signal(S);$ |

- **Starvation** – indefinite blocking. A process may never be removed from the semaphore queue in which it is suspended.

Two Types of Semaphores

- *Counting semaphore* – integer value can range over an unrestricted domain.
- *Binary semaphore* – integer value can range only between 0 and 1; can be simpler to implement.

Binary Semaphores

- A binary semaphore is a semaphore with an integer value that can range only between 0 and 1
- It is simple to implement.
- Type binary semaphore =**record**

```
    value:(0,1)
    queue: list of processes
  end;
```
- var s: binary semaphore
- **waitB(s):**

```
  If s.value=1 then
    s.value=0
  else
    begin
      place this process in s.queue;
      block this process;
    end;
```
- **signalB(s):**

```
  If s.queue is empty then
    s.value=1
  else
    begin
      remove (wakeup) the process from
      s.queue;
      place this process in the ready list.
    end;
```

Implementing S as a Binary Semaphore

- Can implement a counting semaphore S as a binary semaphore.
- Data structures:
binary-semaphore S1, S2;
int C:
- Initialization:
S1 = 1
S2 = 0
C = initial value of semaphore S

☞ *wait* operation

```
wait(S1);
C--;
if (C < 0) {
    signal(S1);
    wait(S2);
}
signal(S1);
```

☞ *signal* operation

```
wait(S1);
C++;
if (C <= 0)
    signal(S2);
else
    signal(S1);
```

Counting semaphores

wait(S):

```
S.value--;
if (S.value < 0) {
    add this process to
    S.L;
    block;
}
```

signal(S):

```
S.value++;
if (S.value <= 0) {
    remove a process P
    from S.L;
    wakeup(P);
}
```

Classical Problems of Synchronization

- Bounded-Buffer Problem
- Readers and Writers Problem
- Dining-Philosophers Problem

Bounded-Buffer Problem

- Used to illustrate the power of synchronization techniques
- We assume that the buffer consists of n buffers, each capable of holding an item.
- The **mutex** semaphore provides mutual exclusion access to buffer which is initialized to the value 1.
- The **empty** and **full** semaphores count the number of empty and full buffers which are initialized to n and zero respectively.
- Shared data
semaphore full, empty, mutex;
- Initially:
full = 0, empty = n, mutex = 1

Bounded-Buffer Problem Producer Process

```
do {  
    ...  
    produce an item in nextp  
    ...  
wait(empty);  
wait(mutex);  
    ...  
    add nextp to buffer  
    ...  
signal(mutex);  
signal(full);  
} while (1);
```

Bounded-Buffer Problem Consumer Process

```
do {  
    wait(full)  
    wait(mutex);  
  
    ...  
    remove an item from buffer to nextc  
  
    ...  
    signal(mutex);  
    signal(empty);  
  
    ...  
    consume the item in nextc  
  
    ...  
} while (1);
```

- Producer is producing full buffers for the consumer and consumer is producing empty buffers for the consumer.

Readers-Writers Problem

- Problem: A data object (file or record) is shared among several concurrent processes.
 - ☞ Some want to read and others want to update it.
- **Readers:** processes interested in reading.
- **Writers:** processes interested in writing.
- Two readers can access shared data object simultaneously.
- But a writer and reader can access shared data object simultaneously
 - ☞ problems may occur!
- To protect from these problems, writers should have an exclusive access to the shared object.
- This synchronization problem is referred to as readers-writers problem.
- It is a different kind of synchronization problem.
- The readers-writers problem has several variations.
 - ☞ Simple one: No reader will be kept waiting unless writer has obtained permission to write.

Readers-Writers Problem

- Semaphores used: **mutex** and **wrt**
- The semaphore **wrt** is common to reader and writer.
- Semaphore **mutex** is used to update **readcount**.
- **readcount** keeps track of how many are reading the object.
- Shared data

semaphore mutex, wrt;

Initially

mutex = 1, wrt = 1, readcount = 0

Readers-Writers Problem Writer Process

wait(wrt);

...

writing is performed

...

signal(wrt);

Readers-Writers Problem Reader Process

```
wait(mutex);
readcount++;
if (readcount == 1)
    wait(wrt);
signal(mutex);
```

...
reading is performed

```
...
wait(mutex);
readcount--;
if (readcount == 0)
    signal(wrt);
signal(mutex);
```

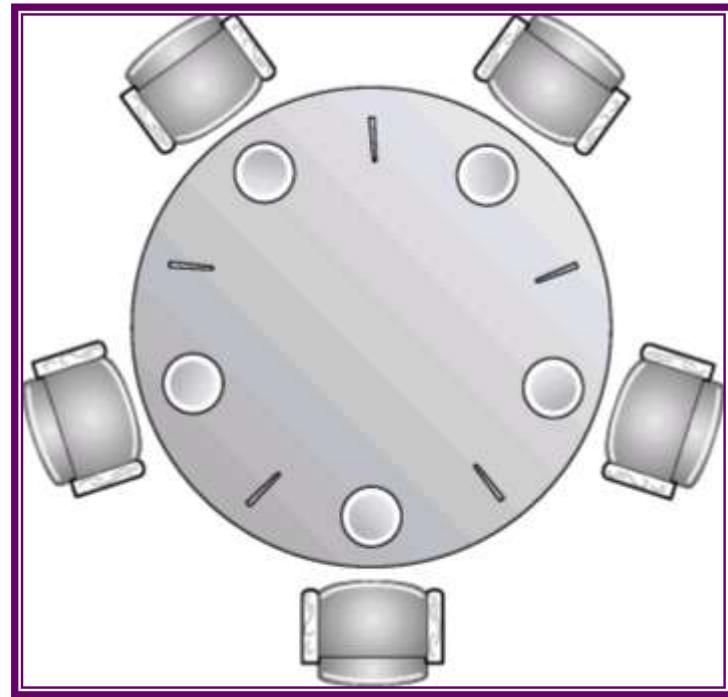
- Writers can be starved if there is a continuous sequence of readers.

Readers-Writers Problem

- Can the producer/consumer problem is considered as case of the readers/writers problem with a writer is a producer and reader is a consumer ?
- The answer is no
- The producer is not just a writer
 - ☞ It must read queue of pointers to determine where to write the next item and it must determine if the buffer is full.
- Similarly the consumer is not a reader
 - ☞ It must adjust queue pointers to show that it has removed a unit from the buffer.

Dining-Philosophers Problem

- Five philosophers spend their lives on thinking and eating.
 - They share a common circular table surrounded by five chairs.
 - Five single chopsticks are available.
 - Whenever a philosopher wants to eat, he tries to pick up two chopsticks that are closest to him/her.
 - A philosopher can not pick the chopstick in the hand of neighbor.
 - After finishing, the philosopher puts back the chopsticks and starts thinking.
-
- It is simple representation of the need to allocate several resources among several processes in a **deadlock and starvation free manner**.



Dining-Philosophers Problem

- Shared data

```
semaphore chopstick[5];
```

Initially all values are 1

- Philosopher i :

```
do {  
    wait(chopstick[i])  
    wait(chopstick[(i+1) % 5])  
    ...  
    eat  
    ...  
    signal(chopstick[i]);  
    signal(chopstick[(i+1) % 5]);  
    ...  
    think  
    ...  
} while (1);
```

- The solution creates a deadlock

Barbershop Problem

- 3 barbers, each with a barber chair
 - ☞ Haircuts may take varying amounts of time
- Sofa can hold 4 customers, max of 20 in shop
- Customers wait outside if necessary
- When a chair is empty:
 - ☞ Customer sitting longest on sofa is served
 - ☞ Customer standing the longest sits down
- After haircut, go to cashier for payment
 - ☞ Only one cash register
 - ☞ Algorithm has a separate cashier, but often barbers also take payment
 - ☞ This is also a critical section

Barbershop Problem

- The main body of the program activates 50 customers, 3 barbers, and the cashier process. Synchronization operators.
 - ☞ Shop and sofa capacity: the capacity of shop and the capacity of the sofa are governed by the semaphores **max_capacity** and **sofa**.
 - ☒ When customer enters max_capacity decremented by one.
 - ☒ When a customer leaves it is incremented.
 - ☒ Wait and signal operations are surround the actions of sitting and getting_up from sofa.
 - ☞ **Barber chair capacity:**
 - ☒ There are three barber chairs; the semaphore **barber_chair** assures that no more than three customers attempt to obtain service at a time.
 - ☒ A customer will not get up from the sofa until at least one chair is free.
 - ☞ **Ensuring customers are in the barber chair:** The semaphore **cust_ready** provides a wakeup signal for a sleeping barber indicating that the customer has just taken the chair.
 - ☞ **Holding customers in barber chair:** once seated the customer remain in the chair until the barber gives the signal that haircut is complete, using the semaphore **finished**.
 - ☞ **Limiting one customer to a barber chair:** the semaphore **barber_chair** is intended to limit the number of customers in barber chairs to three. The semaphore **leave_b_chair** is used to synchronize sitting.
 - ☞ **Paying and receiving:** payment and receipt semaphores are used to synchronize the operations.
 - ☞ **Coordinating barber and cashier functions:** To save money the barber shop does not employ a separate cashier. Each barber is required to perform that task when not cutting hair. The semaphore **coord** ensures the barbers perform only one task at a time.

Barbershop Problem

| Semaphore | Wait operation | Signal operation |
|----------------------|--|---|
| max_capacity | Customer waits for a room to enter shop. | Exiting customer signals customer waiting to enter |
| sofa | Customer waits for seat on sofa | Customer leaving sofa signals customer waiting for sofa |
| barber_chair | Customer waits for empty barber chair | Barber signals when that barber's chair is empty |
| Cust_read | Barber waits until customer is in the chair | Customer signals barber that customer is in the chair |
| finished | Customer waits until his haircut is complete. | Barber signals when done cutting hair of his customer. |
| leave_b_chair | Barber waits until customer gets up from the chair | Customer signals barber when customer gets up from chair. |
| payment | Cashier waits for a customer to pay | Customer signals cashier that he has paid. |
| receipt | Customer waits for a receipt for a payment | Cashier signals that payment has been accepted. |
| coord | Wait for a barber resource to be free to be free perform either the hair cutting or cashiering function. | Signal that a barber resource is free. |

Barbershop

```
program barbershop1;
var max_capacity: semaphore (:=20);
sofa: semaphore (:=4);
barber_chair, coord: semaphore (:=3);
cust_ready, leave_b_chair, payment, receipt: semaphore (:=0)

procedure customer;
var custnr: integer;
begin
  wait (max_capacity );
  enter shop;
  wait( sofa );
  sit on sofa;
  wait( barber_chair );
  get up from sofa;
  signal( sofa );
  sit in barber chair;
  wait( mutex2 );
  signal( cust_ready );
  wait( finished[custnr] );
  leave barber chair;
  signal( leave_b_chair );
  pay;
  signal( payment );
  wait( receipt );
  exit shop;
  signal( max_capacity );
end;

procedure barber;
var b_cust: integer
begin
  repeat
    wait( cust_ready );
    cut hair;
    end;
    signal( coord );
    signal( finsihed[b_cust] );
    wait( leave_b_chair );
    signal( barber_chair );
  forever
end;

procedure cashier;
begin
  repeat
    wait( payment );
    wait( coord );
    accept payment;
    signal( coord );
    signal( receipt );
  forever
end;

Void main()
{
  count=0;
  Parbegin {customer... 50 times,...customer,
  Barber, barber,barber, cashier)
}
```

```

/* program barbershop1 */
semaphore max_capacity = 20;
semaphore sofa = 4;
semaphore barber_chair = 3;
semaphore coord = 3;
semaphore cust_ready = 0, finished = 0, leave_b_chair = 0, payment= 0, receipt = 0;

void customer()
{
    wait(max_capacity);
    enter_shop();
    wait(sofa);
    sit_on_sofa();
    wait(barber_chair);
    get_up_from_sofa();
    signal(sofa);
    sit_in_barber_chair();
    signal(cust_ready);
    wait(finished);
    leave_barber_chair();
    signal(leave_b_chair);
    pay();
    signal(payment);
    wait(receipt);
    exit_shop();
    signal(max_capacity)
}

void main()
{
    parbegin (customer, . . . 50 times, . . . customer, barber, barber, barber, cashier);
}

```

Figure 16: An unfair barbershop

Another method to avoid the unfairness is to number the barber chairs so that less semaphores are needed, but how? Think about it!

Barbershop Problem

- The preceding solution is unfair.
- The customers are servers in the order they enter the shop.
- If one barber is very fast and one of the customer is quite bald.
- The problem can be solved with more semaphores.
 - ☞ We assign unique customer number is to each customer.
 - ☞ The semaphore mutex1 protects access to global variable count.
- The semaphore finished is refined to be an array of 50 semaphores.
 - ☞ Once a customer seated in a barber chair, he executes `wait(finished[custnt])` to wait in his own unique semaphore.
- Please see the solution in William Stallling book (pp 229-234)

Fair Barbershop

```

program barbershop2;
var
max_capacity: semaphore (:=20);
sofa: semaphore (:=4);
barber_chair, coord: semaphore (:=3);
mutex1, mutex2: semaphore (:=1);
cust_ready, leave_b_chair, payment, receipt: semaphore (:=0)
finished: array [1..50] of semaphore (:=0);
count: integer;

procedure customer;
var custnr: integer;
begin
  wait (max_capacity );
  enter shop;
  wait( mutex1 );
  count := count + 1;
  custnr := count;
  signal( mutex1 );
  wait( sofa );
  sit on sofa;
  wait( barber_chair );
  get up from sofa;
  signal( sofa );
  sit in barber chair;
  wait( mutex2 );
  enqueue1( custnr );
  signal( cust_ready );
  signal( mutex2 );
  wait( finished[custnr] );
  leave barber chair;
  signal( leave_b_chair );
  pay;
  signal( payment );
  wait( receipt );
  exit shop;
  signal( max_capacity );
end;

procedure barber;
var b_cust: integer
begin
  repeat
    wait( cust_ready );
    wait( coord );
    wait( mutex2 );
    dequeue1( b_cust );
    signal( mutex2 );
    wait( coord );
    cut hair;
    signal( coord );
    signal( finsihed[b_cust] );
    wait( leave_b_chair );
    signal( barber_chair );
  forever
  end;
end;

procedure cashier;
begin
  repeat
    wait( payment );
    accept payment;
    signal( coord );
    signal( receipt );
  forever
  end;
end;

Void main()
{
  count=0;
  Parbegin {customer... 50 times,...customer,
  Barber, barber,barber, cashier)
}

```

```

/* program barbershop2 */
semaphore max_capacity = 20;
semaphore sofa = 4;
semaphore barber_chair = 3, coord = 3;
semaphore mutex1 = 1, mutex2 = 1;
semaphore cust_ready = 0, leave_b_chair = 0, payment = 0, receipt = 0;
semaphore finished [50] = {0};
int count;

void customer()
{
    int custnr;
    wait(max_capacity);
    enter_shop();
    wait(mutex1);
    count++;
    custnr = count;
    signal(mutex1);
    wait(sofa);
    sit_on_sofa();
    wait(barber_chair);
    get_up_from_sofa();
    signal(sofa);
    sit_in_barber_chair();
    wait(mutex2);
    enqueue1(custnr);
    signal(cust_ready);
    signal(mutex2);
    wait(finished[custnr]);
    leave_barber_chair();
    signal(leave_b_chair);
    pay();
    signal(payment);
    wait(receipt);
    exit_shop();
    signal(max_capacity)
}

void main()
{
    count := 0;
    parbegin (customer, . . . 50 times, . . . customer, barber, barber, barber,
              cashier);
}

```

Figure 17: An fair barbershop

Implementing wait() and signal() in Multi-processor Systems

- Disabling interrupts will not work.
- Spinlock is the solution
 - ☞ With this we have moved busy waiting from entry section to critical sections of application programs.

Implementation of Semaphores

- wait and signal operations are atomic.
- No two processes should execute wait and signal on the same semaphore at the same time
 -
- Good Solution: implement through hardware or firmware.
- Other solutions
 - ☞ Ensure that only process manipulates “wait” and “signal” operations.
 - ☞ One can use Dekker’s algorithm or Peterson’s algorithm
 - ☞ Substantial processing overhead
 - ☞ Use one of the hardware supported schemes
 - ☞ Test and set
 - ☞ disabling interrupts (single processor)

About busy waiting

- Note: we have not eliminated the busy waiting with wait() and signal() completely.
 - ☞ Moved busy waiting from entry section to critical sections of application programs.
 - ☞ Furthermore, we have limited busy waiting to critical sections of wait() and signal() operations.
 - ☞ The wait and signal code is very short the amount of busy waiting involved is short.

Two possible implementations of Semaphores

```
Wait(s)
{
    while(!testset(s.flag)
        /* do nothing */
    s.count--;
    if (s.count <0)
    {
        place this process in s.queue;
        block this process (set s.flag to 0)
    }
    else
        s.flag=0;
}
```

```
Signal(s)
{
    while(!testset(s.flag)
        /* do nothing */
    s.count++;
    if (s.count <= 0)
    {
        remove a process P from s.queue;
        Place a process P in the ready list
    }
    s.flag=0;
}
```

```
Wait(s)
{
    Inhibit interrupts
    s.count--;
    if (s.count <0)
    {
        place this process in s.queue;
        block this process allow interrupts
    }
    else
        allow interrupts;
}
```

```
Signal(s)
{
    Inhibit interrupts;
    s.count++;
    if (s.count <= 0)
    {
        remove a process P from s.queue;
        Place a process P in the ready list
    }
    allow interrupts;
}
```

With Interrupts

Problem with semaphores

- Incorrect use may result in timing errors
- These errors are difficult to detect as these occur if only particular sequence occurs.
- Missing or reverse order.
- It is difficult to produce correct program using semaphores.
- The wait and signal operations are scattered throughout the program and it is difficult to see overall effect of these operations on the semaphores.

Problem with semaphores (cont.)

■ Problems

- ☞ Suppose a process interchanges the order in which wait and signal operations on the semaphore are executed

signal (mutex)

CS

wait (mutex)

 Several processes may be executing in their CS simultaneously.

- ☞ Suppose that a process replaces signal(mutex) with wait(mutex)

signal (mutex)

CS

signal (mutex)

wait(mutex)

CS

wait(mutex)

 Deadlock will occur.

- ☞ Suppose a process omits wait(mutex) or signal(mutex) or both.

 ME is violated or deadlock occurs.

■ A critical region and monitor concept is introduced to address this problem

High level synchronization constructs

- To deal with the type of errors caused by semaphores, high-level constructs have been introduced.
 - ☞ Critical region
 - ☞ Monitor
- Assumption
 - ☞ Process contains some local data
 - ☞ Local data can be accessed by only the sequential program that is encapsulated within the same process.
 - ☞ Process can not access the local data of another process.

Monitors

- It is an high level synchronization construct.
- A monitor is a software module consisting of one or more procedures, an initialization sequence and local data.
- Main characteristics
 - 1. The local data variables are accessible only by the monitor's procedures and not by any external procedure.
 - 2. A process enters monitor by entering one of its procedures.
 - 3. Only one process may be executing (active) in the monitor at any time; any other process that has invoked the monitor is suspended while waiting for the monitor to become available.
- 1&2 → object oriented characteristics.
- By enforcing one procedure at a time, the monitor enforces ME facility.

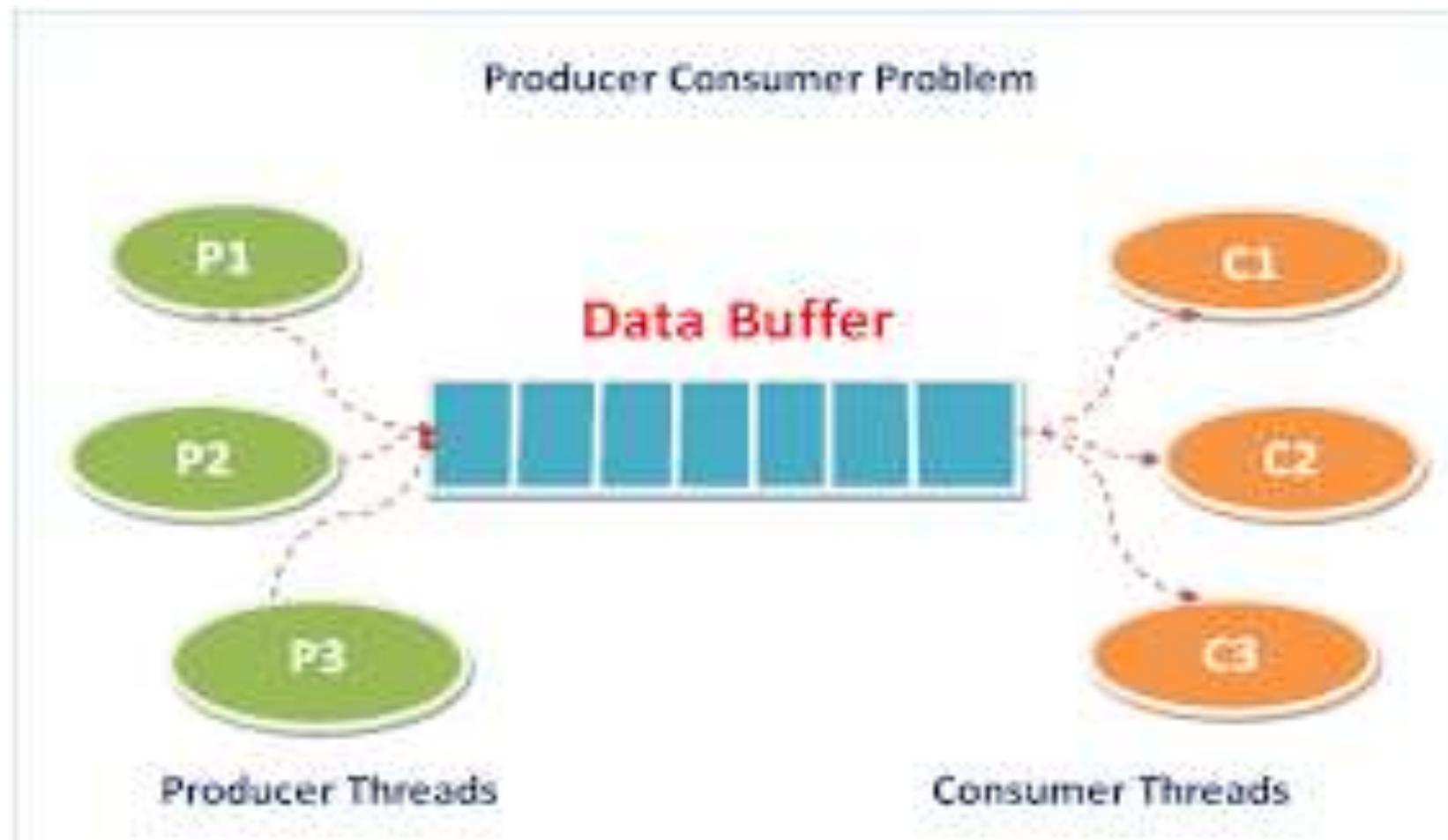
Monitors

- High-level synchronization construct that allows the safe sharing of an abstract data type among concurrent processes.
- A shared data resource can be protected by placing in the monitor.

```
monitor monitor-name
{
    shared variable declarations
    procedure body P1 (...) {
        ...
    }
    procedure body P2 (...) {
        ...
    }
    procedure body Pn (...) {
        ...
    }
    {
        initialization code
    }
}
```

Two kinds of waiting:

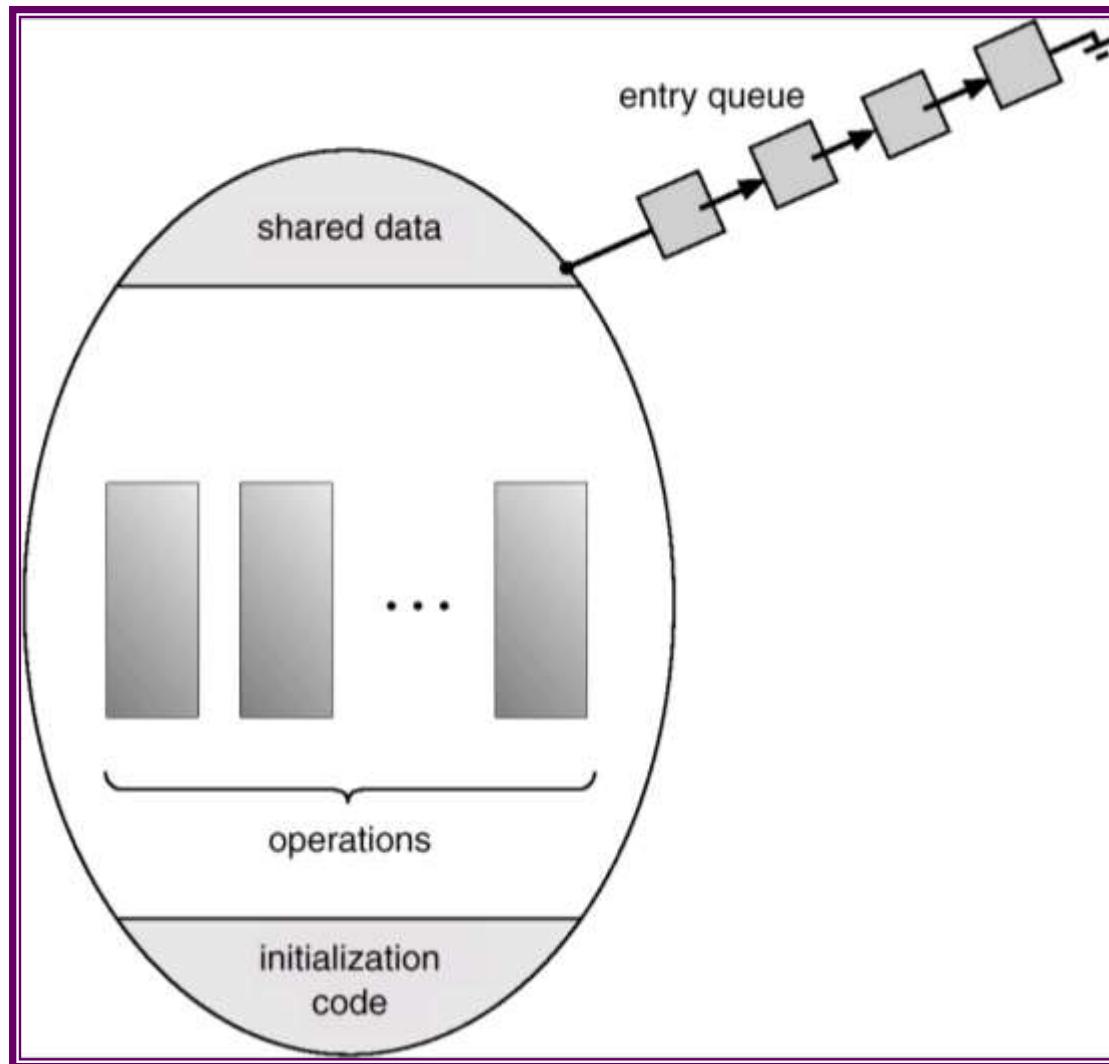
- Mutual exclusion waiting: to avoid race condition
- Conditional waiting: to avoid inconsistency



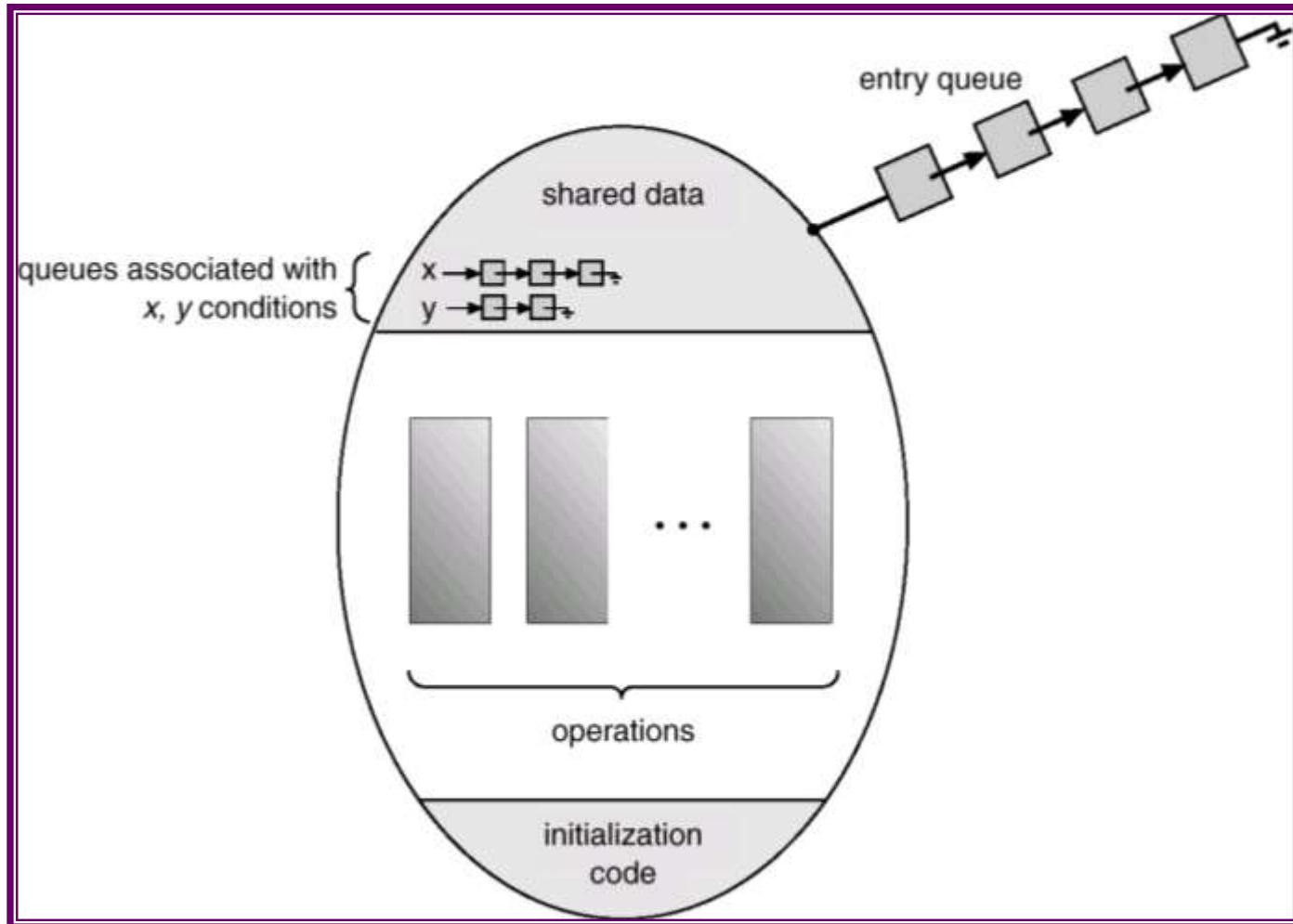
Monitors

- To allow a process to wait within the monitor, a **condition** variable must be declared, as
condition x, y;
- Condition variable can only be used with the operations **wait** and **signal**.
 - ☞ The operation
x.wait();
means that the process invoking this operation is suspended until another process invokes
x.signal();
 - ☞ The **x.signal** operation resumes exactly one suspended process. If no process is suspended, then the **signal** operation has no effect.

Schematic View of a Monitor



Monitor With Condition Variables



Monitors

- In case of monitors, the monitor construct itself provides ME, but synchronization is provided by the programmer.
- In case of semaphore, both ME and synchronization are provided by the programmer.
- Also , in case of monitors also, it is possible to make mistakes in the synchronization of monitors.
- For example if csignal function is omitted, the processes entering corresponding queue are permanently hung up.
- However, since all synchronization functions are confined to monitor, it is easier to verify the synchronization and detect bugs.
- Once a monitor is correctly programmed, access to the protected resource is correct from all processes.
- With semaphores, resources access is correct only if all of the processes that access the resource are programmed correctly.

Dining Philosophers Example

Deadlock free solution

- A philosopher is allowed to pick up his chopsticks only if both of them were available.
- We introduce three states:
 - ☞ Enum {thinking, hungry, eating} state[5]
- Philosopher i can set the variable `state[i]=eating` only if her two neighbors are not eating: (`state[(i+4)%5]!=eating`) and (`state((i+1)%5)!=eating`)
- We also declare condition `self[5]`;
 - ☞ Philosopher i can delay himself when he is hungry, but unable to obtain the chopsticks he needs.

Dining Philosophers Example

Deadlock free solution

```
■ monitor dp
{
    enum {thinking, hungry, eating} state[5];
    condition self[5];
    void pickup(int i)          // following slides
    void putdown(int i) // following slides
    void test(int i)           // following slides

    void init() {
        for (int i = 0; i < 5; i++)
            state[i] = thinking;
    }
}
```

Dining Philosophers

```
void pickup(int i) {
    state[i] = hungry;
    test[i]; // if left and right of i are not eating, then eat.
    if (state[i] != eating)
        self[i].wait();
}

void putdown(int i) {
    state[i] = thinking;
    // test left and right neighbors
    test((i+4) % 5);
    test((i+1) % 5);
}
```

Dining Philosophers

```
void test(int i) {  
    if ( (state[(i + 4) % 5] != eating) &&  
        (state[i] == hungry) &&  
        (state[(i + 1) % 5] != eating)) {  
        state[i] = eating;  
        self[i].signal();  
    }  
}
```

Dining Philosophers

```
dp.pickup[i];
```

....

```
eat
```

....

```
dp.putdown(i);
```

- It is easy to show that no two neighbors are eating simultaneously and no deadlocks will occur.
- However, it is possible for a philosopher to starve to death.

Monitor Implementation Using Semaphores

- Variables

```
semaphore mutex; // (initially = 1)
semaphore next; // (initially = 0)
int next-count = 0;
// number of processes suspended on next.
```

- Each external procedure F will be replaced by
wait(mutex);

...

body of F ;

...

```
if (next-count > 0)
    signal(next)
else
    signal(mutex);
```

- Mutual exclusion within a monitor is ensured.

Monitor Implementation

- For each condition variable **x**, we have:

```
semaphore x-sem; // (initially = 0)  
int x-count = 0;
```

- The operation **x.wait** can be implemented as:

```
x-count++;  
if (next-count > 0)  
    signal(next);  
else  
    signal(mutex);  
wait(x-sem);  
x-count--;
```

Monitor Implementation

- The operation **x.signal** can be implemented as:

```
if (x-count > 0) {  
    next-count++;  
    signal(x-sem);  
    wait(next);  
    next-count--;  
}
```

Monitor Implementation

- *Conditional-wait construct: x.wait(c);*
 - ☞ **c** – integer expression evaluated when the **wait** operation is executed.
 - ☞ value of **c** (*a priority number*) stored with the name of the process that is suspended.
 - ☞ when **x.signal** is executed, process with smallest associated priority number is resumed next.
- Check two conditions to establish correctness of system:
 - ☞ User processes must always make their calls on the monitor in a correct sequence.
 - ☞ Must ensure that an uncooperative process does not ignore the mutual-exclusion gateway provided by the monitor, and try to access the shared resource directly, without using the access protocols.

Solaris 2 Synchronization

- Solaris2 OS is designed to provide real-time computing, be multithreaded and support multiple processors.
- To control access to critical regions Solaris 2 implements
 - ☞ Adaptive mutexes
 - ☞ Condition variables
 - ☞ Semaphores
 - ☞ Reader-writer locks
 - ☞ Turnstiles
- Adaptive mutex protects access to every critical data item.
 - ☞ On multiprocessor system an adaptive mutex starts as a standard semaphore implemented as a spinlock.
 - ☞ Adaptive mutex is used to protect only data that are accessed by short-code segments (few hundred instructions).
- For longer code segments, condition variables and semaphores are used.
- Reader-writers locks are used to access data that is accessed frequently in read-only manner.
 - ☞ Semaphores serialize the access
 - ☞ When there are many readers and few writers r-w locks are efficient.
- Solaris 2 uses turnstiles to order list of threads waiting to acquire either an adaptive mutex or a reader-writer lock.

Solaris 2 Synchronization

- Uses *turnstiles* to order the list of threads waiting to acquire either an adaptive mutex or reader-writer lock.
 - ☞ It is a queue structure containing threads blocked in a lock.
- To prevent a priority inversion, turnstiles are organized into priority inheritance protocol.
 - ☞ When a lower priority thread holds a lock, and higher priority thread blocks, the lower priority thread inherits the priority of the higher-priority thread.

Windows 2000 Synchronization

- Multi-threaded kernel
 - ☞ Real-time applications and multiple processors
- Uses interrupt masks to protect access to global resources on uni-processor systems.
- Uses *spinlocks* on multiprocessor systems.
 - ☞ Kernel ensures that a thread will never be preempted while holding a spinlock.
- Also provides *dispatcher objects* which may act as mutexes and semaphores and events.
- Dispatcher objects may also provide *events*. An event acts much like a condition variable which may notify a waiting thread when a desired condition occurs.

Transactional Memory

- Multi-core applications increases risk of race conditions and deadlocks.
- Techniques proposed: locks, semaphores and monitors
- Transactional memory provides alternative strategy
- A memory transaction is a sequence of memory read and write transactions that are atomic. If all the operations are completed the memory transaction is committed. Otherwise, the operations must be aborted. Such a feature can be added to programming language.
- Traditional way

```
☞ Update () {  
    Acquire();  
    /* Modify shared data */  
    release(); }
```

- Suppose we add an atomic operation

```
☞ Update() {  
    Atomic { }  
}
```

Atomic transactions

- A group pf statements should be executed as a logical unit.
- More than mutual exclusion
- Notion of transaction has been emerged
 - ☞ ACID properties
 - ☞ Atomicity, consistency, Isolation and durability
- Atomicity: all or nothing
- Consistency: one consistent state to another consistent state
- Isolation: Parallel execution is serial
- Durability: Changes are permanent
- Two-phase locking and log based recovery methods are followed.
- Will be studied in database systems course

Transactional Memory

■ Advantage

☞ System is responsible for guaranteeing the atomicity.

■ Transactional memory systems can be implemented in either software or in hardware.

☞ STM: software transactional memory

 appropriate code is inserted by compiler

☞ HTM: Hardware transactional memory

 Uses cache coherency protocols to support transaction memory

Critical Regions

- High-level synchronization construct
- A shared variable v of type T , is declared as:
 $v: shared T$
- Variable v accessed only inside statement
region v when B do S

where B is a boolean expression.

- While statement S is being executed, no other process can access variable v .
- The expression B is Boolean expression which governs the access to the critical region.

Critical Regions

- Regions referring to the same shared variable exclude each other in time.
- When a process tries to execute the region statement, the Boolean expression B is evaluated. If B is true, statement S is executed. If it is false, the process is delayed until B becomes true and no other process is in the region associated with v .
- If the following two statements are executed concurrently, it will be equivalent to the serial execution “S1 followed by S2” or “S2 followed by S1”.
 - ☞ Region v when (true) S1;
 - ☞ Region v when (true) S2;
- CR construct guards against simple errors associated with the semaphore solution.

Example – Bounded Buffer

- Shared data:

```
struct buffer {  
    int pool[n];  
    int count, in, out;  
}
```

Bounded Buffer Producer Process

- Producer process inserts **nextp** into the shared buffer

```
region buffer when( count < n) {  
    pool[in] = nextp;  
    in:= (in+1) % n;  
    count++;  
}
```

Bounded Buffer Consumer Process

- Consumer process removes an item from the shared buffer and puts it in **nextc**

```
region buffer when (count > 0) {  
    nextc = pool[out];  
    out = (out+1) % n;  
    count--;  
}
```

Implementation region x when B do S

- Can be implemented using semaphores.

Implementing S as a Binary Semaphore

- Can implement a counting semaphore S as a binary semaphore.
- Data structures:
binary-semaphore S1, S2;
int C:
- Initialization:
S1 = 1
S2 = 0
C = initial value of semaphore S

☞ *wait* operation

```
wait(S1);
C--;
if (C < 0) {
    signal(S1);
    wait(S2);
}
signal(S1);
```

☞ *signal* operation

```
wait(S1);
C++;
if (C <= 0)
    signal(S2);
else
    signal(S1);
```

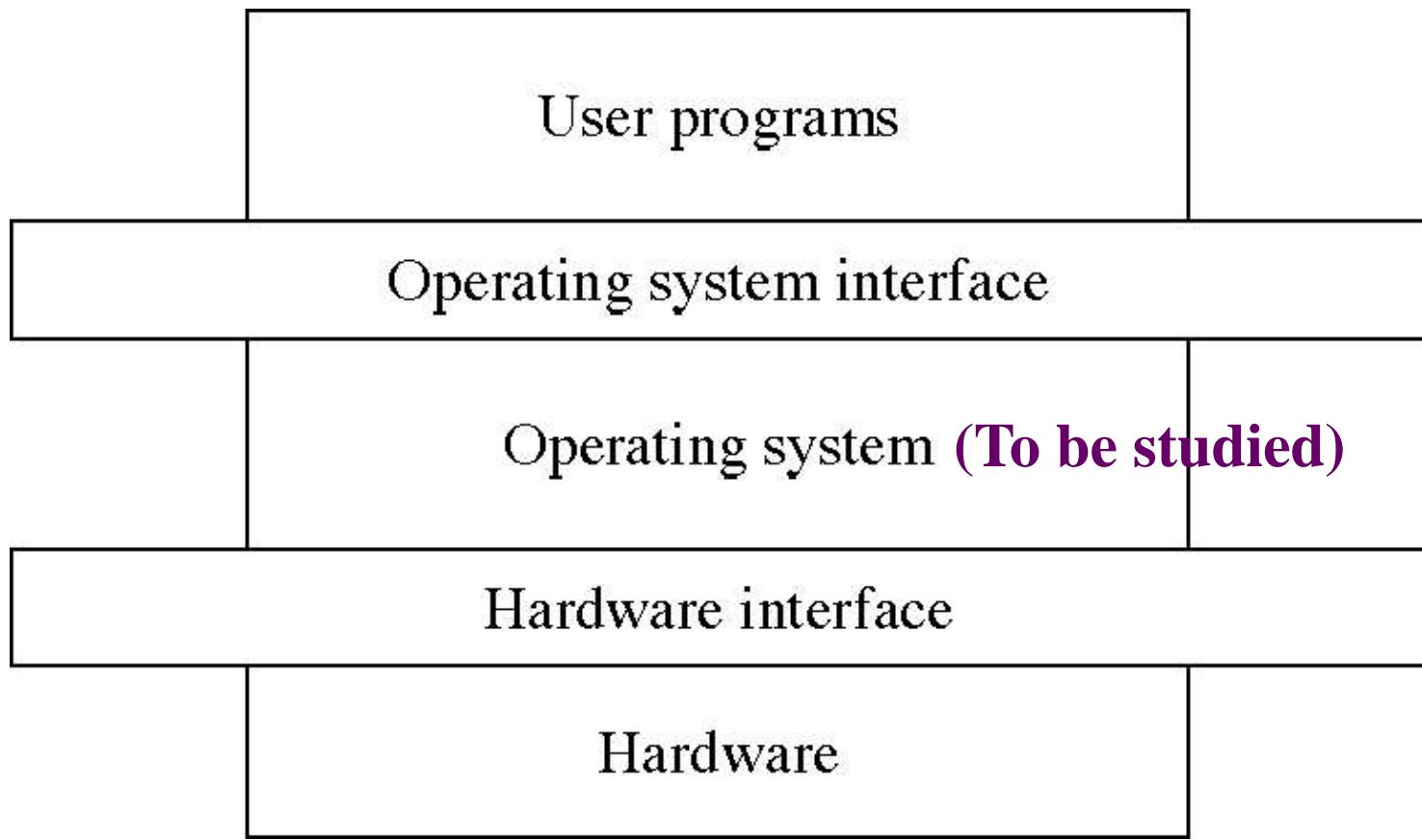
Counting semaphores

wait(S):

```
S.value--;
if (S.value < 0) {
    add this process to
    S.L;
    block;
}
```

signal(S):

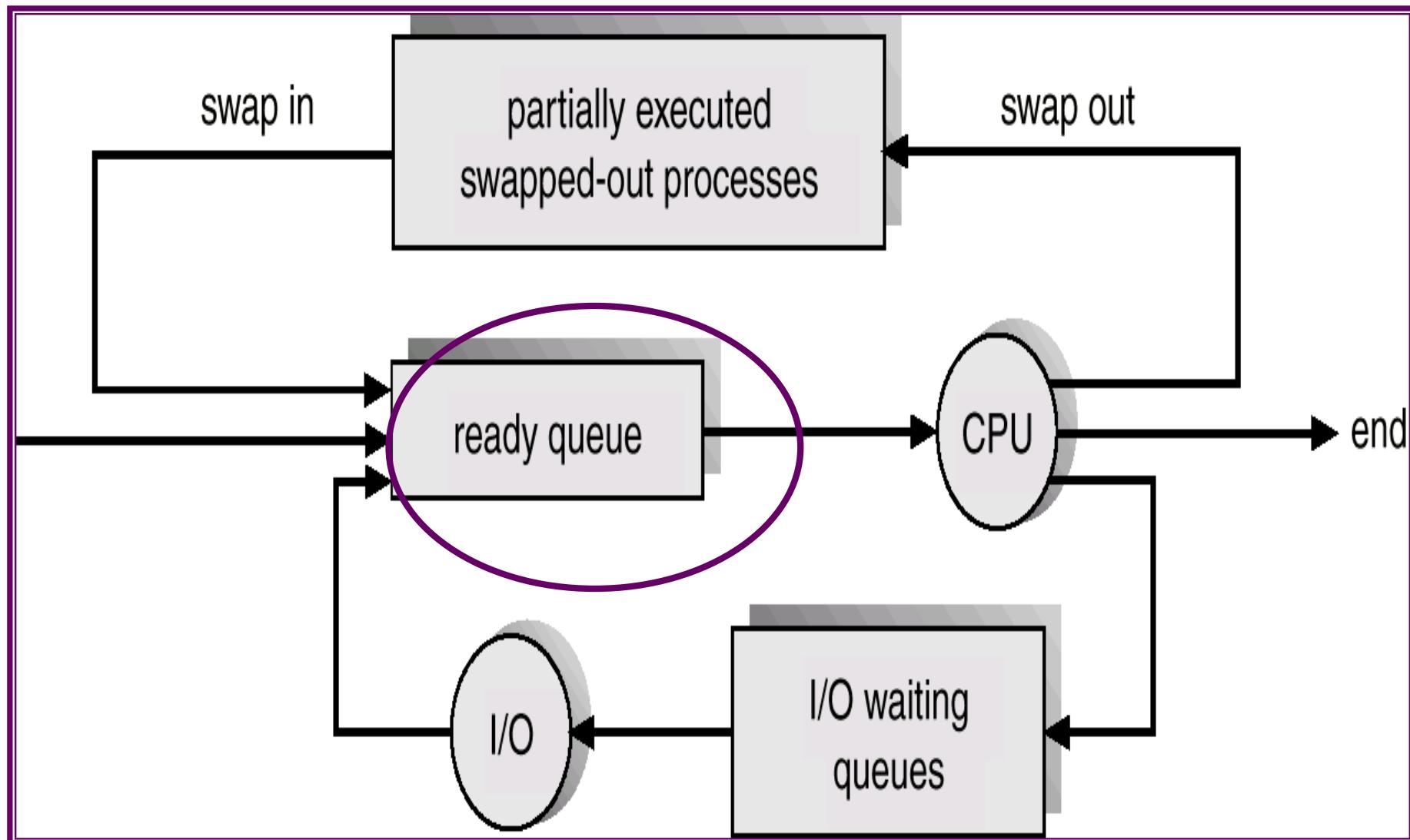
```
S.value++;
if (S.value <= 0) {
    remove a process P
    from S.L;
    wakeup(P);
}
```



Process Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Real-Time Scheduling
- Algorithm Evaluation
- Process Scheduling Models

Scheduling

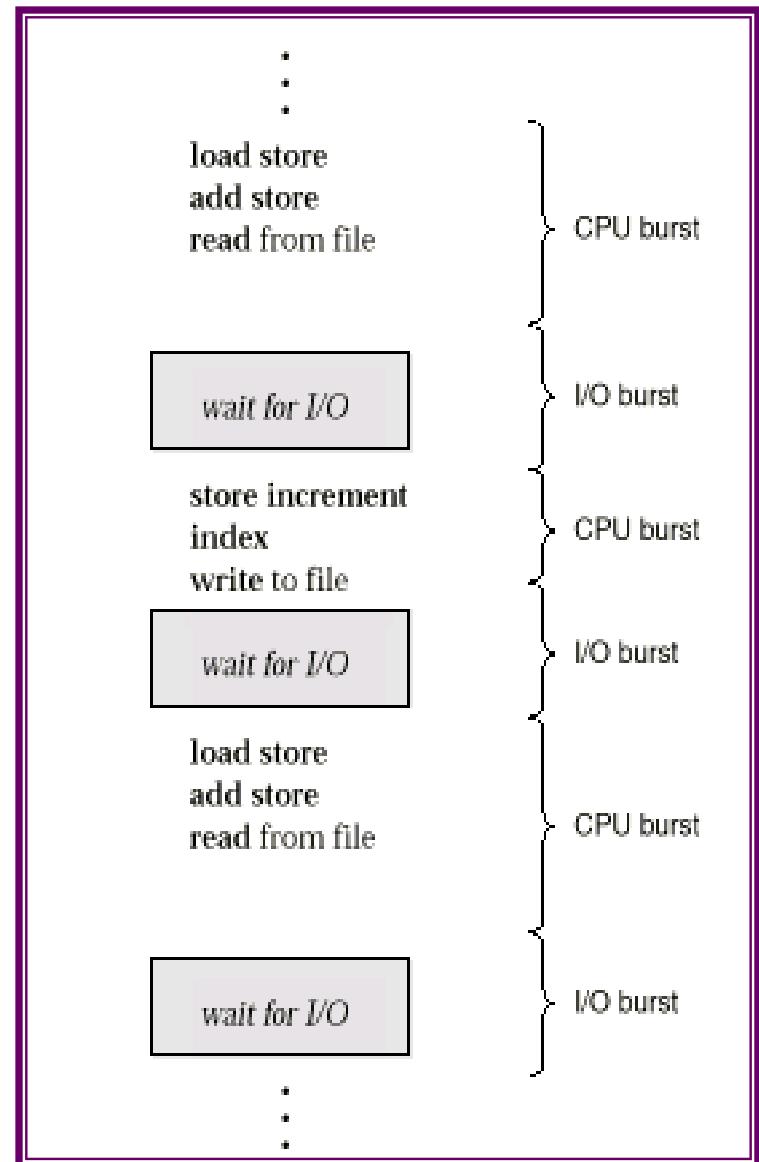


Basic Concepts

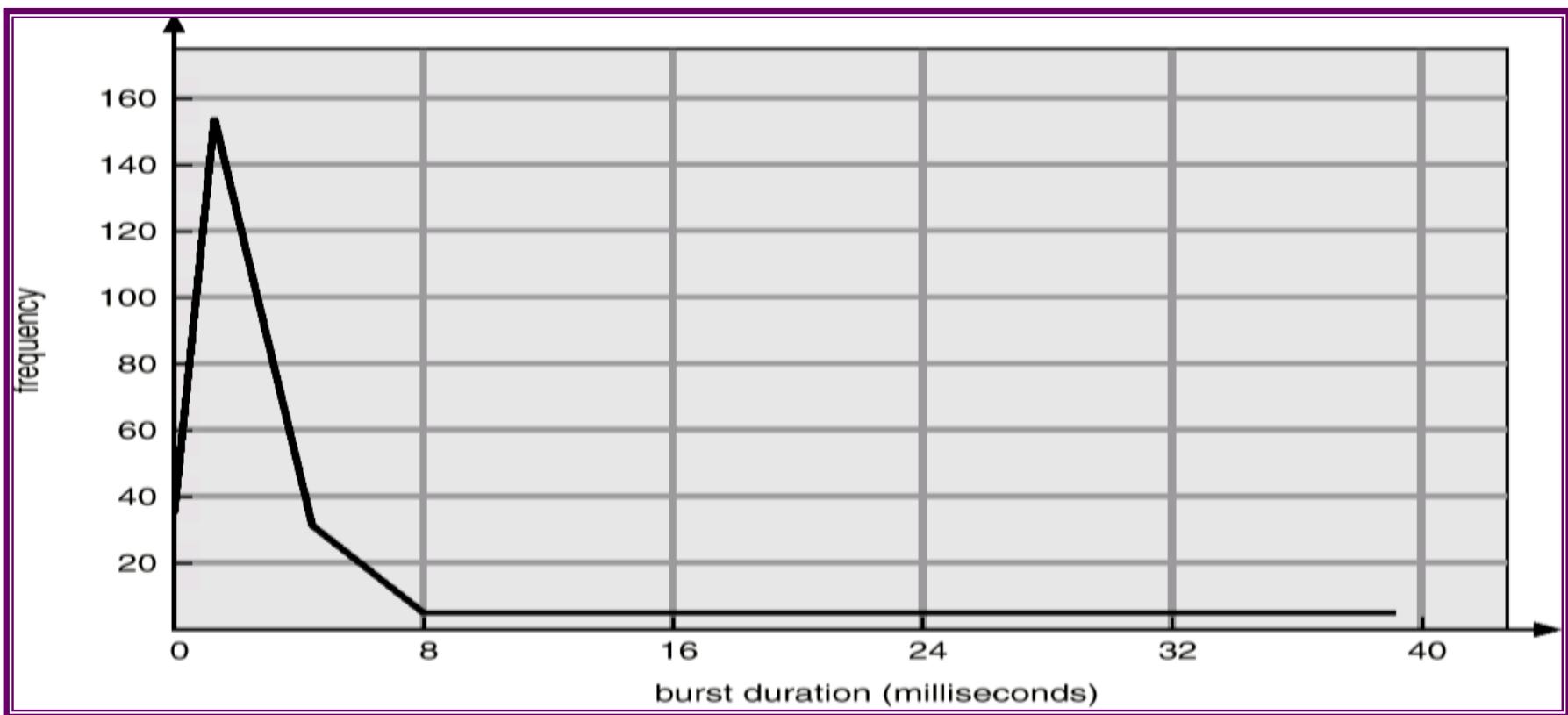
- CPU scheduling is the basis of multi-programmed OS.
- In the uni-processor system only one process runs at a time. Other processes must wait until CPU is free and can be rescheduled.
- Idea: the process is executed until it must wait for completion of I/O request.
- When one process has to wait, OS takes CPU away from that process and gives the CPU to another process.
- All the computer resources are scheduled before use.
- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait.
- CPU burst distribution

Observed property

- Process execution consists of a cycle of execution and I/O wait.
- CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait.
- Process execution starts with CPU-burst. That is followed by I/O burst.
- The execution ends with CPU burst.
- CPU burst distribution



Histogram of CPU-burst Times



Many short CPU bursts and a few long CPU bursts.

CPU and I/O bound processes

- An I/O bound program would have many short CPU bursts.
- A CPU bound program would have long CPU bursts.

CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- Short-term scheduler
- CPU scheduling decisions may take place when a process:
 1. Switches from running to waiting state (due to I/O).
 2. Switches from running to ready state (due to interrupt)
 3. Switches from waiting to ready. (Completion of I/O)
 4. Terminates.
- Scheduling under 1 and 4 is *non-preemptive*.
 - ☞ Once a CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.
 - ❑ WINDOWS 3.1 and MAC
- All other scheduling is *preemptive*.
 - ☞ The process can be preempted and another process can be run.
 - ❑ UNIX
 - ❑ Difficult to implement.
 - ❑ (Refer the book for the issues)

Designing Preemptive Scheduling based OS is difficult

- It incurs cost to access shared data
- Affects the design of OS
 - ☞ Interrupts occur at arbitrary time, changing of data modified by kernel
- Section of code modified by interrupts should be guarded
- But, all modern OSs are preemptive scheduling due to performance advantages as compared to non-preemptive scheduling

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
 - ☞ switching context
 - ☞ switching to user mode
 - ☞ jumping to the proper location in the user program to restart that program
- *Dispatch latency* – time it takes for the dispatcher to stop one process and start another running.

Scheduling Criteria

- Scheduling algorithm favors one process to another.
- User oriented criteria
 - ☞ **Turnaround time** – amount of time to execute a particular process
 - ⌚ The interval from the time of submission of a process to the time of completion.
 - ⌚ The sum of periods spent in waiting to get into memory, waiting in the ready queue, executing on CPU and doing I/O
 - ☞ **Response time** – amount of time it takes from when a request was submitted until the first response is produced, **not FINAL/LAST OUTPUT** (for time-sharing environment)
 - ☞ **Deadlines:** When the process completion deadlines can be specified, the scheduling discipline should subordinate their goals to that of maximizing the percentage of deadlines met.

Scheduling Criteria

- System oriented
 - ☞ **Throughput** – # of processes that complete their execution per time unit
 - ☞ **CPU utilization** – keep the CPU as busy as possible; percentage of time the processor is busy.
 - ☞ It may range from 0 to 100 percent.
 - ☞ In a real system, it should range from 40 percent (lightly loaded) to 90 percent (heavily loaded)
 - ☞ **Waiting time:** It is the sum of the periods spent waiting in the ready queue.
- System oriented: other
 - ☞ **Fairness:** In the absence of guidance from user or other system supplied guidance, processes should be treated the same, and no process should suffer starvation.
 - ☞ **Enforcing priorities:** When processes are assigned priorities, the scheduling policy should favor higher priority processes.
 - ☞ **Balancing resources:** The scheduling policy should keep the resources of the system busy. Processes that underutilize the stressed resources should be favored which involves long term scheduling.

Optimization Criteria

■ Maximize

- ☞ CPU utilization
- ☞ Throughput

■ Minimize

- ☞ turnaround time
- ☞ waiting time
- ☞ response time



(Some suggest variance of response time)

- ☞ Power consumption (Mobile)

Scheduling Algorithms

- FIFO
- SJF
- Priority
- Round Robin
- Multilevel
- Multilevel feedback

First-Come, First-Served (FCFS) Scheduling

- The process that requests CPU first is allocated the CPU first.

| <u>Process</u> | <u>Burst Time</u> |
|----------------|-------------------|
| P_1 | 24 |
| P_2 | 3 |
| P_3 | 3 |

- Suppose that the processes arrive in the order: P_1, P_2, P_3
The Gantt Chart for the schedule is:



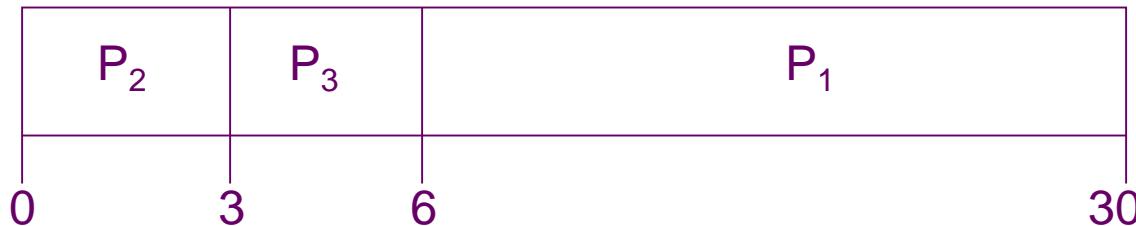
- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

$$P_2, P_3, P_1.$$

- The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case.
- *Dynamic environment*
 - ☞ **Convoy effect:** short process behind long process

FCFS Scheduling (Cont.)

■ Scenarios

- ☞ One CPU bound process and many I/O bound process
- ☞ CPU bound process will hold the CPU for a long time
- ☞ All I/O processes are in ready queue
 - ☞ The I/O devices are idle
- ☞ This is a convoy effect
 - ☞ Several processes wait for one big process.

■ FCFS algorithm is non-preemptive

■ Positive Points

- ☞ code is simple to write and understand

■ Negative points

- ☞ It is not good for timesharing systems
- ☞ Average waiting time may be too long.
- ☞ Convoy effect results in lower CPU and device utilization.
- ☞ Penalizes short processes; penalizes I/O bound processes
- ☞ Response time may be high, especially there is a large variance in process execution times.

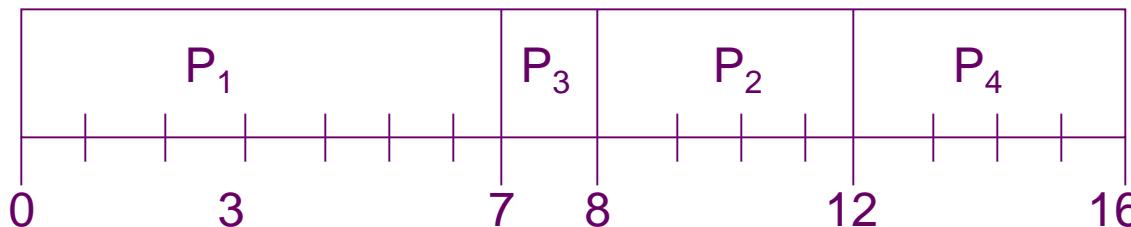
Shortest-Job-First (SJF) Scheduling

- When CPU is available it is assigned to the process that has the smallest next CPU burst.
- If the bursts are equal, FCFS is used to break the tie.
- Associate with each process the length of its **next CPU burst**. Use these lengths to schedule the process with the shortest time.
- Two schemes:
 - ☞ **Non-preemptive** – once CPU is given to the process it cannot be preempted until it completes its CPU burst.
 - ☞ **Preemptive** – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as Shortest-Remaining-Time-First (SRTF).
- **SJF is optimal** – gives minimum average waiting time for a given set of processes.

Example of Non-Preemptive SJF

| <u>Process</u> | <u>Arrival Time</u> | <u>Burst Time</u> |
|----------------|---------------------|-------------------|
| P_1 | 0.0 | 7 |
| P_2 | 2.0 | 4 |
| P_3 | 4.0 | 1 |
| P_4 | 5.0 | 4 |

- SJF (non-preemptive)

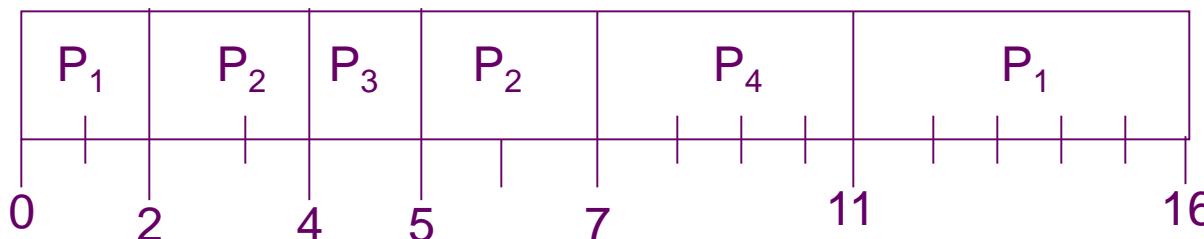


- Average waiting time = $(0 + 6 + 3 + 7)/4 = 4$

Example of Preemptive SJF

| <u>Process</u> | <u>Arrival Time</u> | <u>Burst Time</u> |
|----------------|---------------------|-------------------|
| P_1 | 0.0 | 7 |
| P_2 | 2.0 | 4 |
| P_3 | 4.0 | 1 |
| P_4 | 5.0 | 4 |

- SJF (preemptive)



- Average waiting time = $(9 + 1 + 0 + 2)/4 = 3$

Determining Length of Next CPU Burst

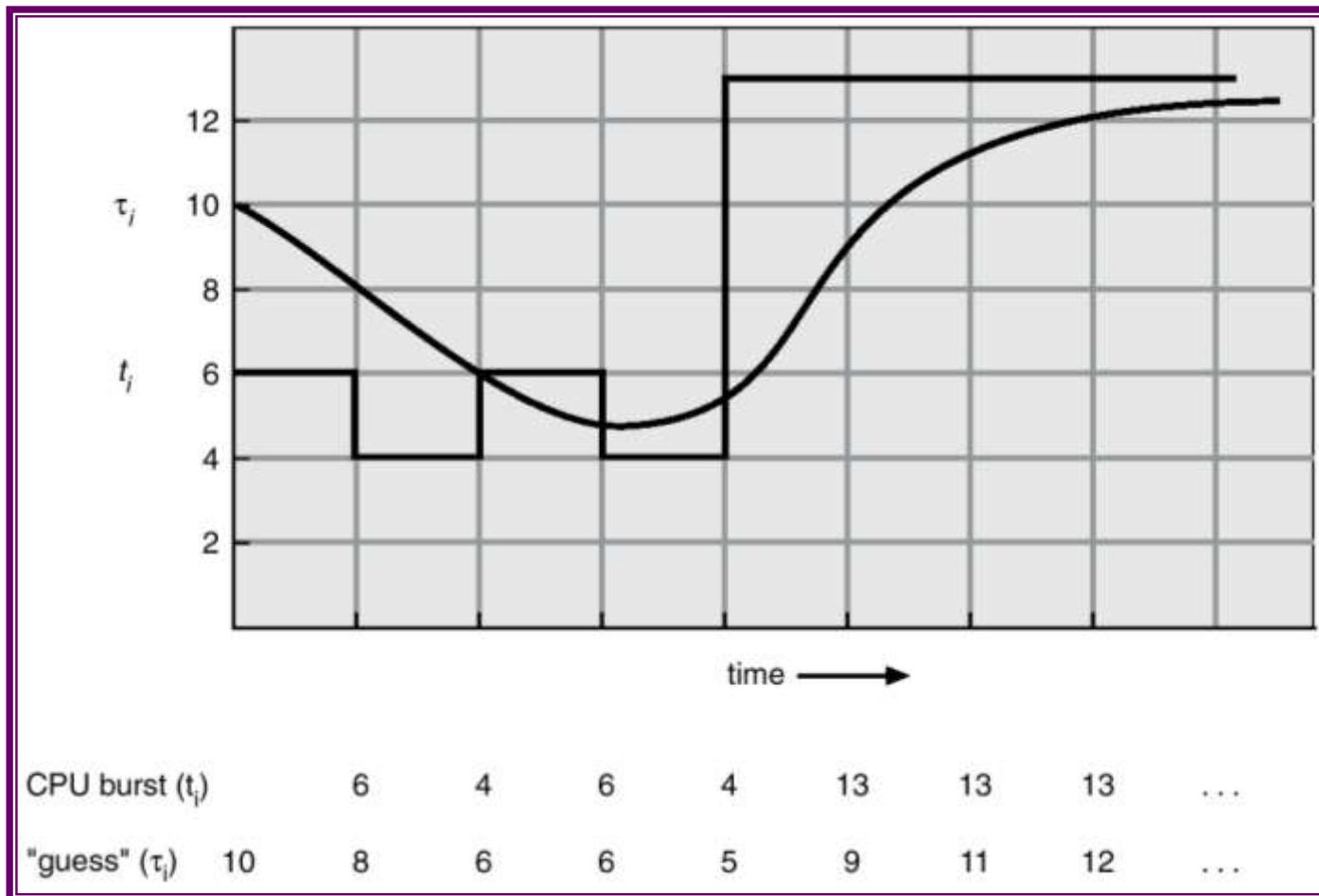
- Difficulty: Knowing the next CPU burst of each process.
- Solution: approximation.
- Simplest approximation: running average of each “burst” for each process.
 - ☞ $\tau_{n+1} = \frac{1}{n} \sum t_i$ (i=1 to n)
 - ☞ Where, t_i processor execution time for the i'th instance of this process (processor execution time for batch job, processor burst time for interactive job)
 - ☞ τ_i ; predicted value for the ith instance
 - ☞ τ_1 ; predicted value for the first instance (not calculated)
- To avoid recalculating the entire summation each time we can rewrite $\tau_{n+1} = \frac{1}{n} t_n + \frac{n-1}{n} \tau_n$
- The above formula gives equal weight to each instance
- Normally most recent references likely reflect future behavior
- Common technique is exponential averaging.

Exponential averaging

1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$
4. Define :

$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n.$$

Prediction of the Length of the Next CPU Burst



Examples of Exponential Averaging

$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n.$$

■ $\alpha = 0$

- ☞ $\tau_{n+1} = \tau_n$; Recent history does not count.
- ☞ If $\alpha \rightarrow 0$; greater weight is given to past values.
- ☞ If $\alpha \rightarrow 1$; greater weight is given to recent values.

■ $\alpha = 1$

- ☞ $\tau_{n+1} = t_n$
- ☞ Only the actual last CPU burst counts.

■ If we expand the formula, we get:

$$\begin{aligned}\tau_{n+1} &= \alpha t_n + (1 - \alpha) \alpha t_{n-1} + \dots \\ &\quad + (1 - \alpha)^{j-1} \alpha t_{n-j} + \dots \\ &\quad + (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor.
- If $\alpha = 0.8$, then $\tau_{n+1} = 0.8 t_n + 0.16 t_{n-1} + 0.032 t_{n-2} + \dots$

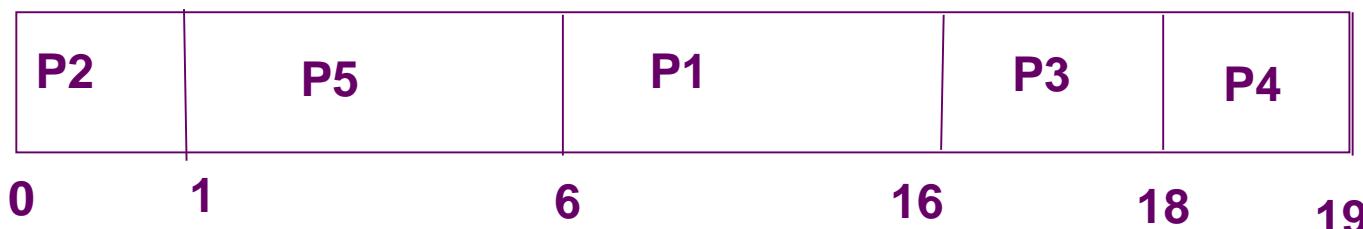
Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer \equiv highest priority).
- Equal priority process is scheduled in FCFS order.
- SJF is a priority scheduling where priority is the predicted next CPU burst time.

Example: Priority scheduling

| <u>Process</u> | <u>Burst time</u> | <u>Priority</u> |
|----------------|-------------------|-----------------|
| P_1 | 10 | 3 |
| P_2 | 1 | 1 |
| P_3 | 2 | 3 |
| P_4 | 1 | 4 |
| P_5 | 5 | 2 |

- Priority
- Average waiting time = 8.2 msec



Priority scheduling...

- Priority scheduling can be either **preemptive** or **non-preemptive**.
- **A preemptive priority scheduling algorithm** will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.
- **A non-preemptive priority** scheduling algorithm will simply put the new process at the head of the ready queue.

Priority scheduling: Problem

- **Problem ≡ Starvation** – low priority processes may never execute.
- In an heavily loaded system, a stream of high priority processes can prevent low priority process from ever getting the CPU.
- **Solution ≡ Aging** – as time progresses increase the priority of the process.
 - ☞ Ex: if priorities range from 0 (low)-127 (high), every 15 minutes we can increment the priority of the waiting process.

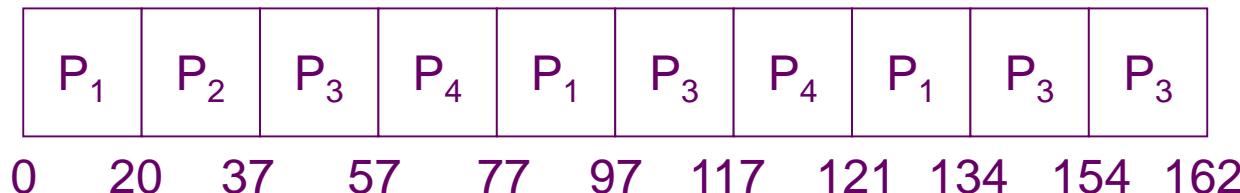
Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- The ready queue is treated as a circular queue.
 - ☞ If the CPU burst is less than 1 time quantum, the process leaves the system.
 - ☞ If the CPU burst is greater than the time quantum, interrupt occurs and context switch will be executed.
- If there are n processes in the ready queue and the time quantum is q , then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.

Example of RR with Time Quantum = 20

| <u>Process</u> | <u>Burst Time</u> |
|----------------|-------------------|
| P_1 | 53 |
| P_2 | 17 |
| P_3 | 68 |
| P_4 | 24 |

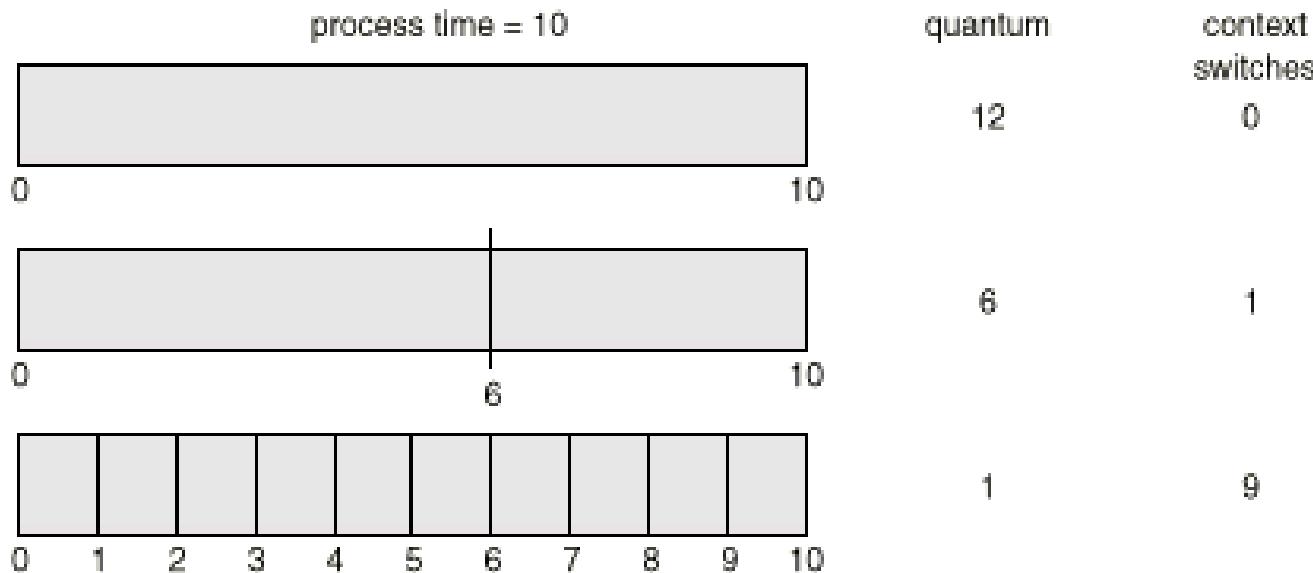
- The Gantt chart is:



- Typically, higher average turnaround than SJF, but better response.

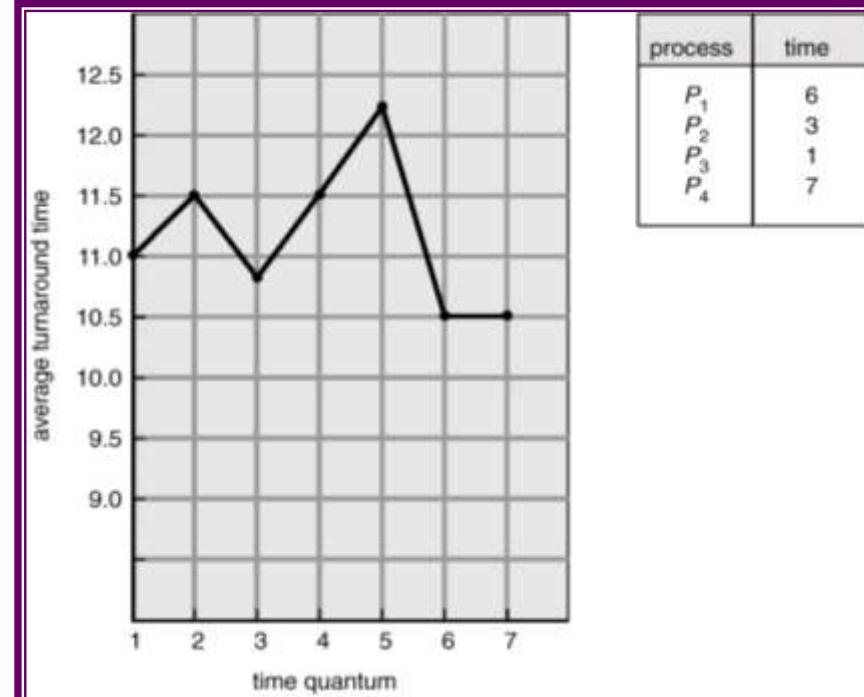
Round Robin Performance

- Performance depends on the size of the time quantum.
- If the time quantum is large, RR policy is the same as the FCFS policy.
- If the time quantum is too small, there will be more number of context switches.



RR performance..

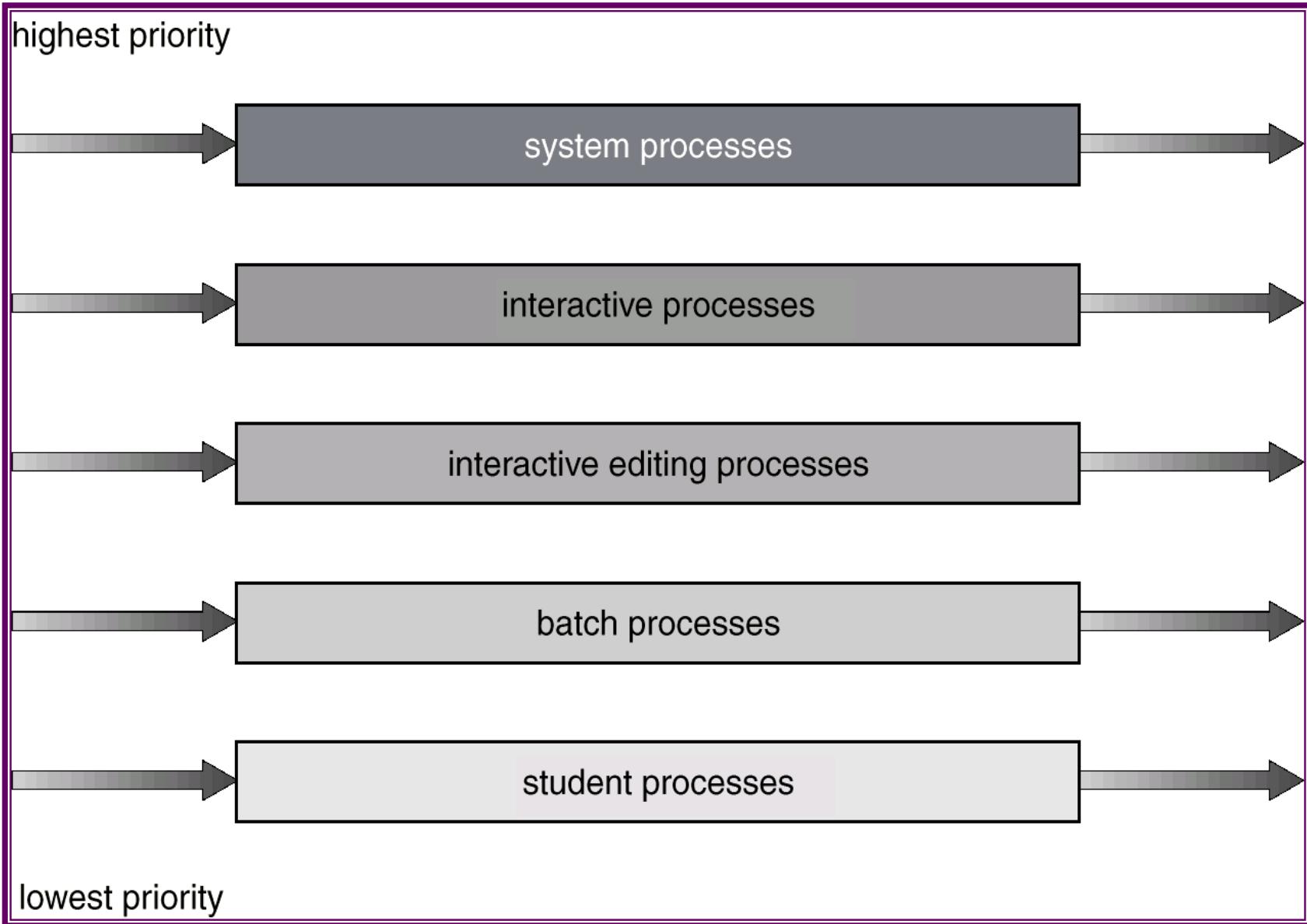
- If the context switch time is 10 % of time quantum, then 10 % of CPU time will be spent on the context switch.
- However, if we increase the time quantum the average turnaround time may not be improved.
- In general the average turnaround time will be improved if process finishes next CPU burst within single time quantum.
- Performance Summary:
 - ☞ q large \Rightarrow FIFO
 - ☞ q small \Rightarrow more number of context switches.
 - ☞ q must be large with respect to context switch, otherwise overhead is too high.
- **Rule of thumb:** 80 % of the CPU bursts should be shorter than the time quantum.



Multilevel Queue

- Ready queue is partitioned into several separate queues:
Example: foreground (interactive); background (batch)
 - ☞ Foreground processes may have priority over background processes.
- Each queue has its own scheduling algorithm,
foreground – RR
background – FCFS
- Scheduling must be done between the queues.
 - ☞ Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
 - ☞ Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
 - ☞ 20% to background in FCFS

Multilevel Queue Scheduling



Multilevel Queue

- Each queue has absolute priority over lower-priority queues.
 - ☞ No process in the batch queue can run unless the above queues are empty.
 - ☞ If the interactive process enters while the batch process is running, the batch process would be preempted.
 - ☞ Solaris 2 uses a form of this algorithm.
- Another policy is time slice:
 - ☞ Each queue gets certain portion of CPU time.
 - ☞ The foreground queue may receive 80% of CPU time, and background queue receives 20% of CPU time.
- Aging: A process can move between the various queues; aging can be implemented this way.

Multilevel Feedback Queue Scheduling

- Process moving is allowed
- The basic idea is to separate processes with different CPU-burst characteristics.
 - ☞ If a process uses too much CPU time, it will be demoted to lower priority queue.
 - ☞ If a process waits too long, it will be promoted to higher priority queue.
- ☞ Aging prevents starvation

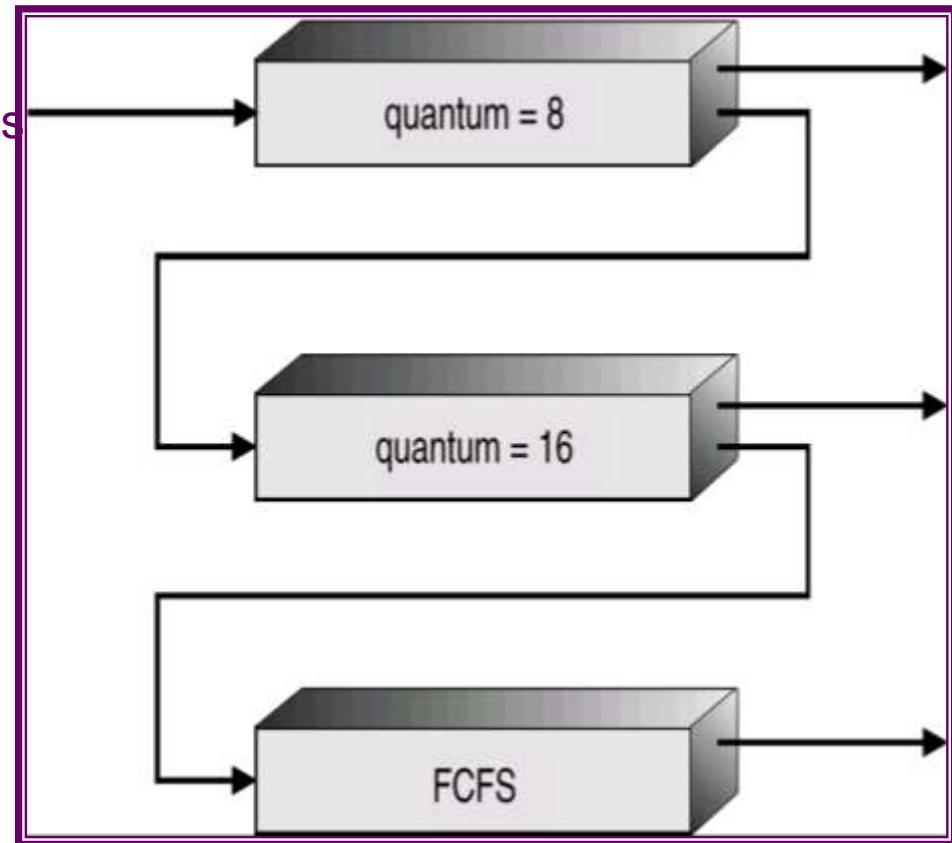
Example of Multilevel Feedback Queue

- Three queues:

- ☞ Q_0 – time quantum 8 milliseconds
 - ☞ Q_1 – time quantum 16 milliseconds
 - ☞ Q_2 – FCFS

- Scheduling

- ☞ A new job enters queue Q_0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1 .
 - ☞ At Q_1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2 .



- The process that arrives in Q_0 will preempt a process in Q_2 .
- Highest priority is given to the process of having 8 msec CPU burst.

Multilevel Feedback Queue Scheduling

- Multilevel-feedback-queue scheduler defined by the following parameters:
 - ☞ number of queues
 - ☞ scheduling algorithms for each queue
 - ☞ method used to determine when to upgrade a process
 - ☞ method used to determine when to demote a process
 - ☞ method used to determine which queue a process will enter when that process needs service
- Multi-level is a general scheduling algorithm
- It can be configured to match a specific system under design.
 - ☞ It requires some means of selecting the values for all the parameters.

Thread Scheduling

- Distinction between user-level and kernel-level threads
- When threads supported, threads scheduled, not processes
- Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP
 - ☞ Known as **process-contention scope (PCS)** since scheduling competition is within the process
 - ☞ Typically done via priority set by programmer
- Kernel thread scheduled onto available CPU is **system-contention scope (SCS)** – competition among all threads in system

Pthread Scheduling

- API allows specifying either PCS or SCS during thread creation
 - ☞ PTHREAD_SCOPE_PROCESS schedules threads using PCS scheduling
 - ☞ PTHREAD_SCOPE_SYSTEM schedules threads using SCS scheduling
- Can be limited by OS – Linux and Mac OS X only allow PTHREAD_SCOPE_SYSTEM

Multiple-Processor Scheduling

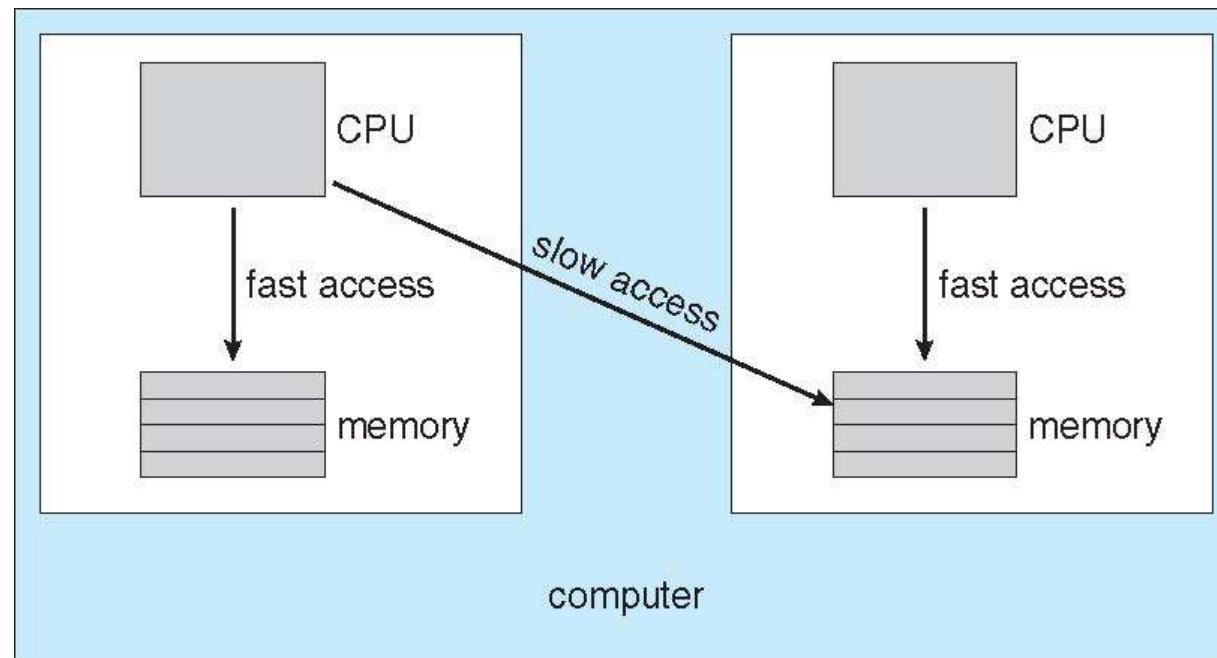
- CPU scheduling more complex when multiple CPUs are available
- **Homogeneous processors** within a multiprocessor
 - ☞ The processors are identical
- **Asymmetric multiprocessing** – only one processor accesses the system data structures, alleviating the need for data sharing
 - ☞ Master server,
 - ☞ simple to implement
- **Symmetric multiprocessing (SMP)** – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes
 - ☞ Currently, most common

Multiple-Processor Scheduling: Issues

- **Processor affinity** – process has affinity for processor on which it is currently running
 - ☞ **soft affinity:** Efforts will be made to run the process on the same CPU, but not guaranteed.
 - ☞ **hard affinity:** Process do not migrate among the processors.
 - ☞ Variations including **processor sets**
 - ☞ Processor set is assigned to a process. It can run on any processor.

NUMA and CPU Scheduling

NUMA: Non-Uniform Memory Access : CPU has faster access to some parts of main memory than to other parts. It occurs in the systems with combined CPU and memory boards.



Note that memory-placement algorithms can also consider affinity

Multiprocessor Scheduling: Load Balancing

- Load balancing attempts to keep the workload evenly distributed across all the processors in an SMP system.
- Push migration and pull migration
 - ☞ Push migration
 - ☞ A specific process periodically checks the load on each processor and evenly distributes the processes.
 - ☞ Pull migration
 - ☞ Idle processor pulls a waiting task from a busy processor.

Multiprocessor Scheduling: Load Balancing

- Load balancing attempts to keep the workload evenly distributed across all the processors in an SMP system.
- Push migration and pull migration
 - ☞ Push migration
 - ☞ A specific process periodically checks the load on each processor and evenly distributes the processes.
 - ☞ Pull migration
 - ☞ Idle processor pulls a waiting task from a busy processor.

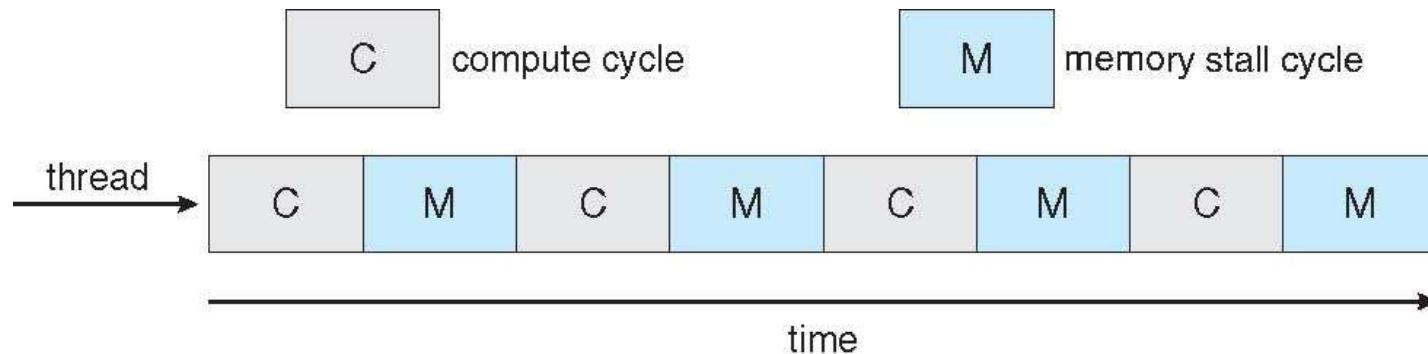
Multiprocessor Scheduling: Load Balancing

- Load balancing attempts to keep the workload evenly distributed across all the processors in an SMP system.
- Push migration and pull migration
 - ☞ Push migration
 - ☞ A specific process periodically checks the load on each processor and evenly distributes the processes.
 - ☞ Pull migration
 - ☞ Idle processor pulls a waiting task from a busy processor.

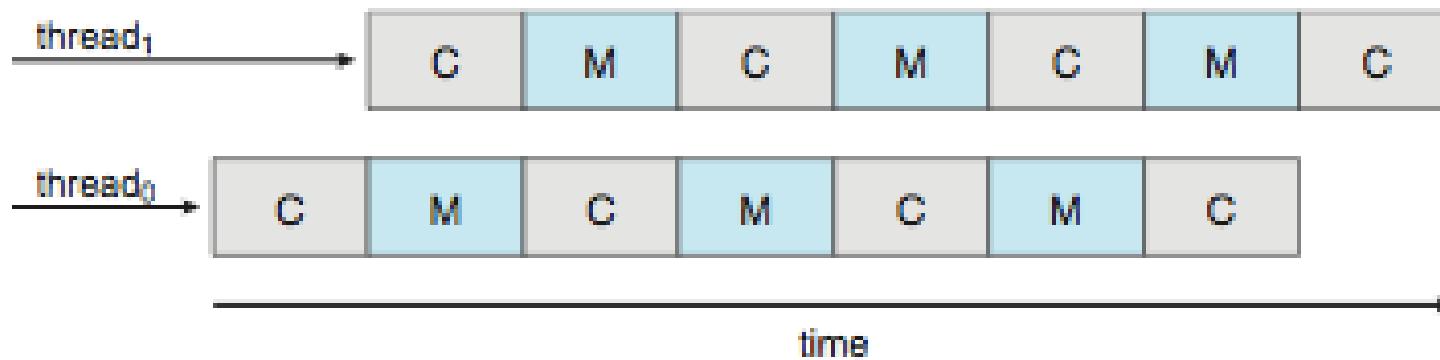
Multicore Processors

- Recent trend is to place multiple processor cores on same physical chip
- Faster and consumes less power
- Multiple threads per core also growing
 - ☞ Memory stall: When a processor accesses main memory, it spends significant amount of time in waiting.
 - ☞ Takes advantage of memory stall to make progress on another thread while memory retrieve happens

Multithreaded Multicore System



- Two threads are associated with one core



- One core may have many logical processors
- A dual core may have four logical processors

Out-of-order execution

- **out-of-order execution** (or more formally **dynamic execution**) is a paradigm used in most high-performance central processing units to make use of instruction cycles that would otherwise be wasted.
- In this paradigm, a processor executes instructions in an order governed by the availability of input data and execution units, rather than by their original order in a program.
- In doing so, the processor can avoid being idle while waiting for the preceding instruction to complete and can, in the meantime, process the next instructions that are able to run immediately and independently.
- Reference: https://en.wikipedia.org/wiki/Out-of-order_execution

Virtualization and Scheduling

- Virtualization software schedules multiple guests onto CPU(s)
- Each guest doing its own scheduling
 - ☞ Not knowing it doesn't own the CPUs
 - ☞ Can result in poor response time
 - ☞ Can effect time-of-day clocks in guests
- Can undo good scheduling algorithm efforts of guests

Algorithm Evaluation

- How to select CPU scheduling algorithm ?
- First problem is selection off criteria
 - ☞ CPU utilization, response time, or throughput
- To select an algorithm we have to select relative importance of these measures.
 - ☞ Maximize CPU utilization under the constraint that maximum response time is one second.
 - ☞ Maximize throughput such that turnaround time is linearly proportional to total execution time.
- After selecting the criteria various algorithms have to be evaluated.
- The following methods are followed for evaluation
 - ☞ Analytical modeling
 - ▀ Deterministic modeling
 - ▀ Queuing models
 - ☞ Simulation
 - ☞ Implementation

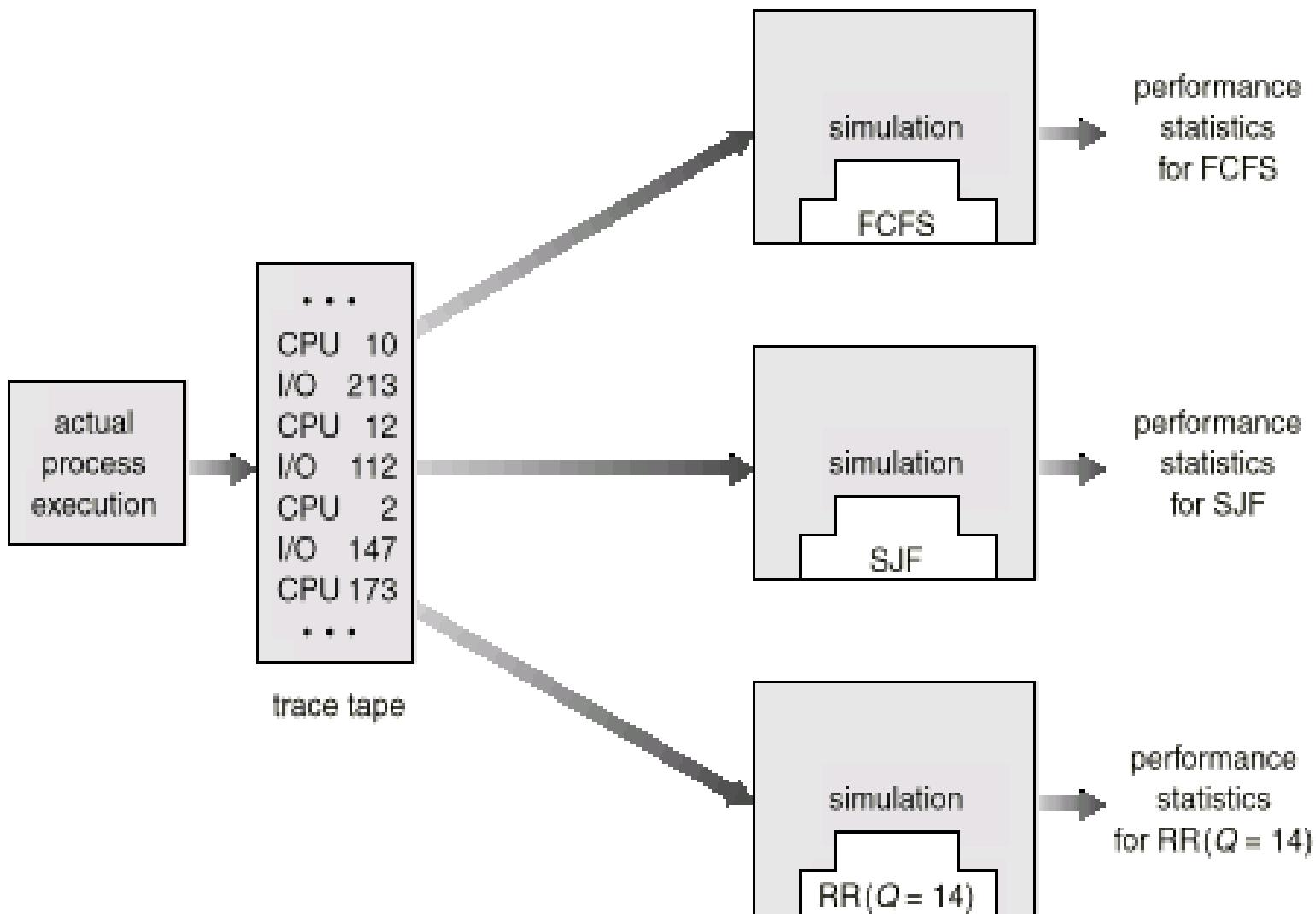
Analytical modeling

- Analytic evaluation uses the given algorithm and the system workload to produce a formula or number that evaluates the performance of the algorithm for that workload.
- One type of analytic evaluation is deterministic modeling.
 - ☞ It takes a predetermined workload and defines the performance of each algorithm for that workload.
 - ☞ + point: Fast
 - ☞ - point: specific to the load
- Queuing models (Reference: https://en.wikipedia.org/wiki/Queueing_theory)
 - ☞ No predetermined workload which is very difficult to get.
 - ☞ However, distributions of CPU and I/O bursts can be approximated.
 - ☞ Result is a probability of next CPU burst.
 - ☞ Variation of every variable is approximated to one distribution.
 - ☞ This area of study is called queuing network analysis.
 - ☞ Little's law: $n = \lambda * W$, where n = number of processes in the queue, λ is arrival rate, and w is average waiting time. (When a system is in steady state).
- +ve points: no cost; pencil and paper
- -ve points: too complex equations; only approximation to real system

Simulation

- For accurate evaluation, simulation can be used.
- Involve programming model of the computer system.
- Software data structures represent the major data structures of the system.
- There is a simulation clock which is a variable.
- Whenever clock is changes, system state is modified
 - ☞ Activities of the devices, the processes, and the scheduler.
- Random number generator can be used to generate data sets by following appropriate distributions.
 - ☞ Processes
 - ☞ CPU-burst times
 - ☞ Arrivals
 - ☞ Departures
- **Trace** can be used which is created by monitoring the real system.
- This area of study is called simulation.

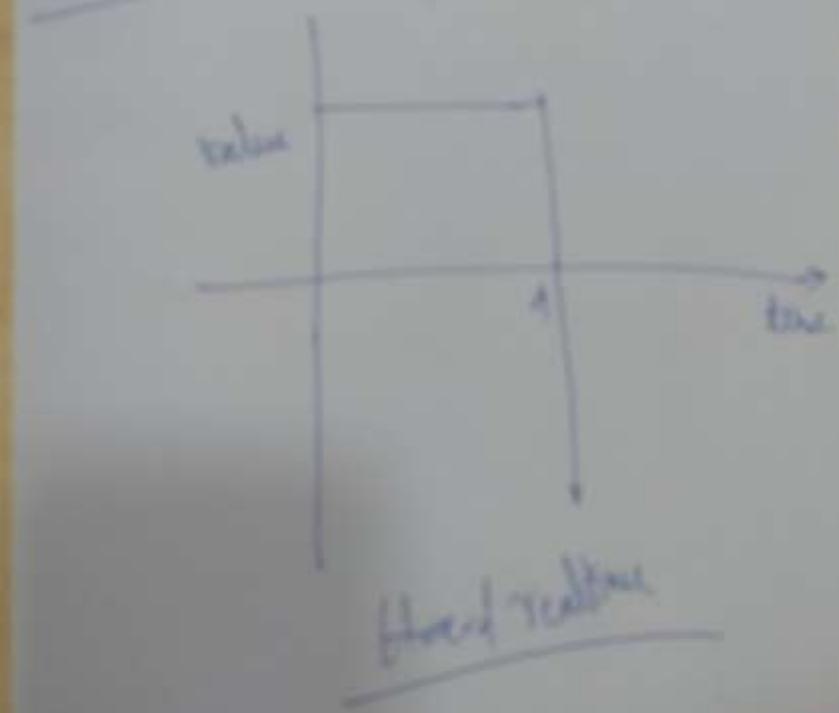
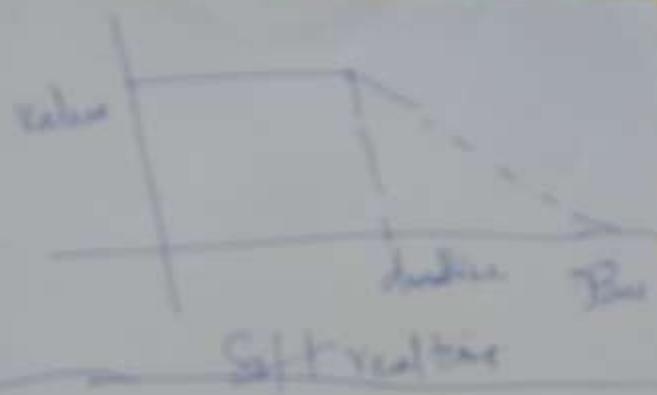
Evaluation of CPU Schedulers by Simulation



Implementation

- Simulation is also of limited accuracy.
- Correct way is code the algorithm and run in an OS.
- (-ve) Cost of this approach
- (-ve) reaction of users due to constant change

- In reality all the three methods are used.
 - ☞ Analytical
 - ☞ Simulation
 - ☞ Implementation



Real-Time Scheduling

- *Hard real-time* systems – required to complete a critical task within a guaranteed amount of time.
- *Soft real-time* computing – requires that critical processes receive priority over less fortunate ones.

Hard real-time systems

- Process is submitted along with a statement of amount of time in which it needs to complete or I/O.
- The scheduler may admit the process or reject the process as impossible.
- The main concept is resource reservation.
 - ☞ The scheduler must know exactly how long each OS function takes to perform.
- It is difficult to realize this guarantee on general purpose OSs.
- Therefore , hard real-time systems are composed of special purpose software running on hardware dedicated to their critical process.

Soft real-time systems

- Computing is less restrictive.
- Requires that critical process should receive priority over lesser priority processes.
- Adding soft real-time functionality may cause problems
 - ☞ Unfair allocation of resources may result in longer delays or even starvation for some processes.
- A soft real-time system is nothing but a general purpose OS that can also support multimedia, high-speed interactive graphics and so on.
 - ☞ Multimedia and high-speed interactive graphics would not work acceptably in an environment that does not support soft real-time computing.

Soft real-time systems...

- Implementation requires careful design of the scheduler and other related aspects.
- System must have priority scheduling.
- Real time processes must receive highest priority and priority should not degrade over time.
 - ☞ Aging is disallowed
- Dispatch latency must be small.
 - ☞ Which is very difficult as some system calls are complex.
 - ☞ For example, many OSs are forced to wait for a system call to complete to execute context switch.
 - ☞ So to keep dispatch latency low, insertion of **preemption points** in system call code is one solution.
 - ☞ At preemption point, the process checks whether any high priority process is waiting.
 - ☞ Another method is to make entire kernel pre-emptable.

Soft real-time systems...

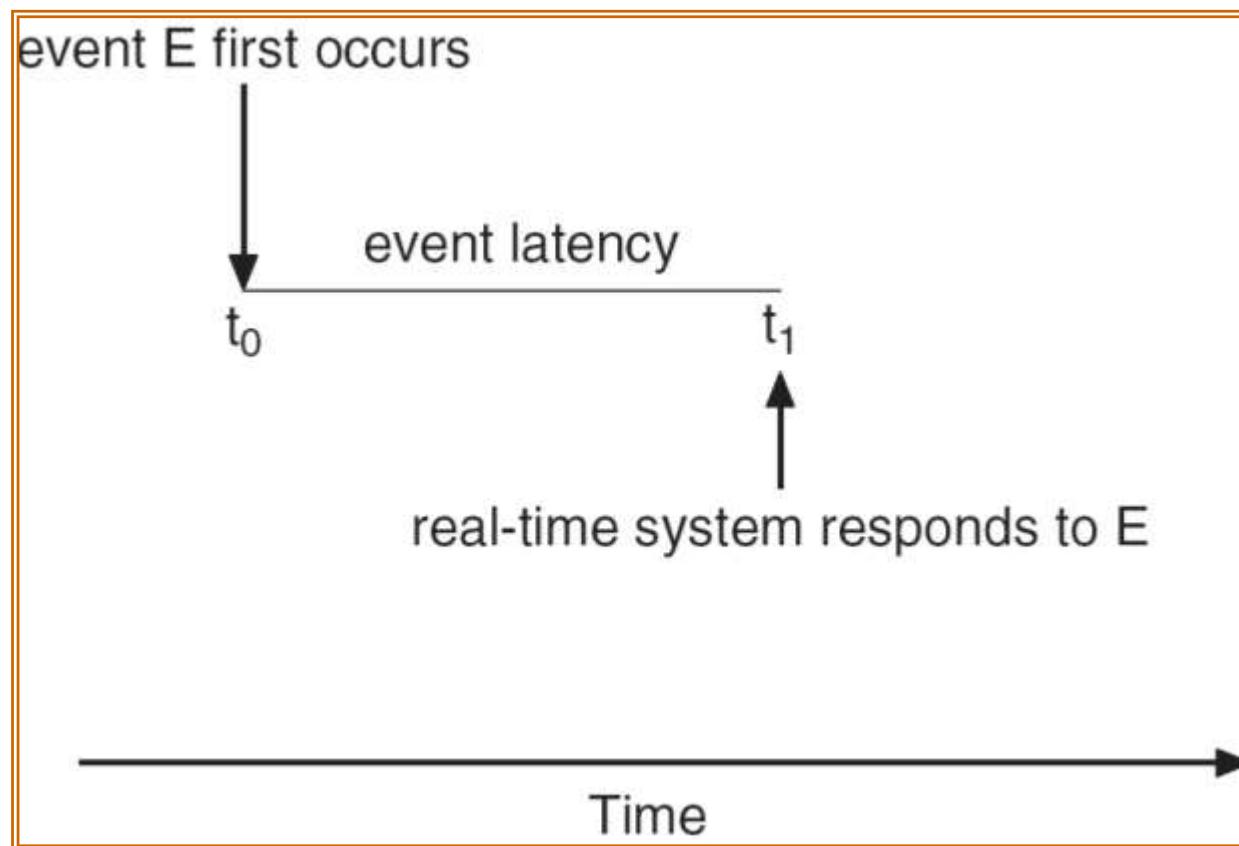
- If higher priority process needs to read or modify kernel data which is currently accessed by lower priority process, then **higher priority process would be waiting for a lower priority process to finish.**
 - This problem can be solved with priority-inheritance protocol
 - ☞ All the processes which are sharing a resource with high priority processes inherit the high priority until they are done with that resource.
 - ☞ After finishing, priority reverts to original value.
 - So the conflict phase of dispatch latency has two components
 - ☞ Preemption of any process running in the kernel
 - ☞ Release by low-priority process resources needed by high-priority resources.
-  In Solaris dispatch latency with preemption is 2 msec and without preemption is 100 msec.

Implementing Real-Time Operating Systems

- In general, real-time operating systems must provide:
 - (1) Preemptive, priority-based scheduling
 - (2) Preemptive kernels
 - (3) Latency must be minimized

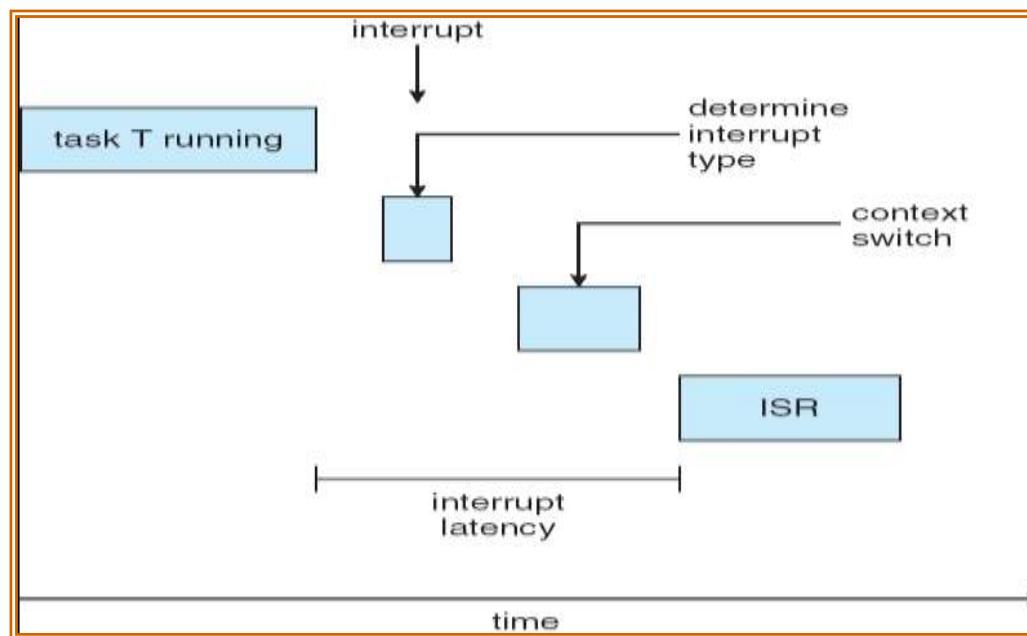
Minimizing Latency

- **Event latency** is the amount of time from when an event occurs to when it is serviced.



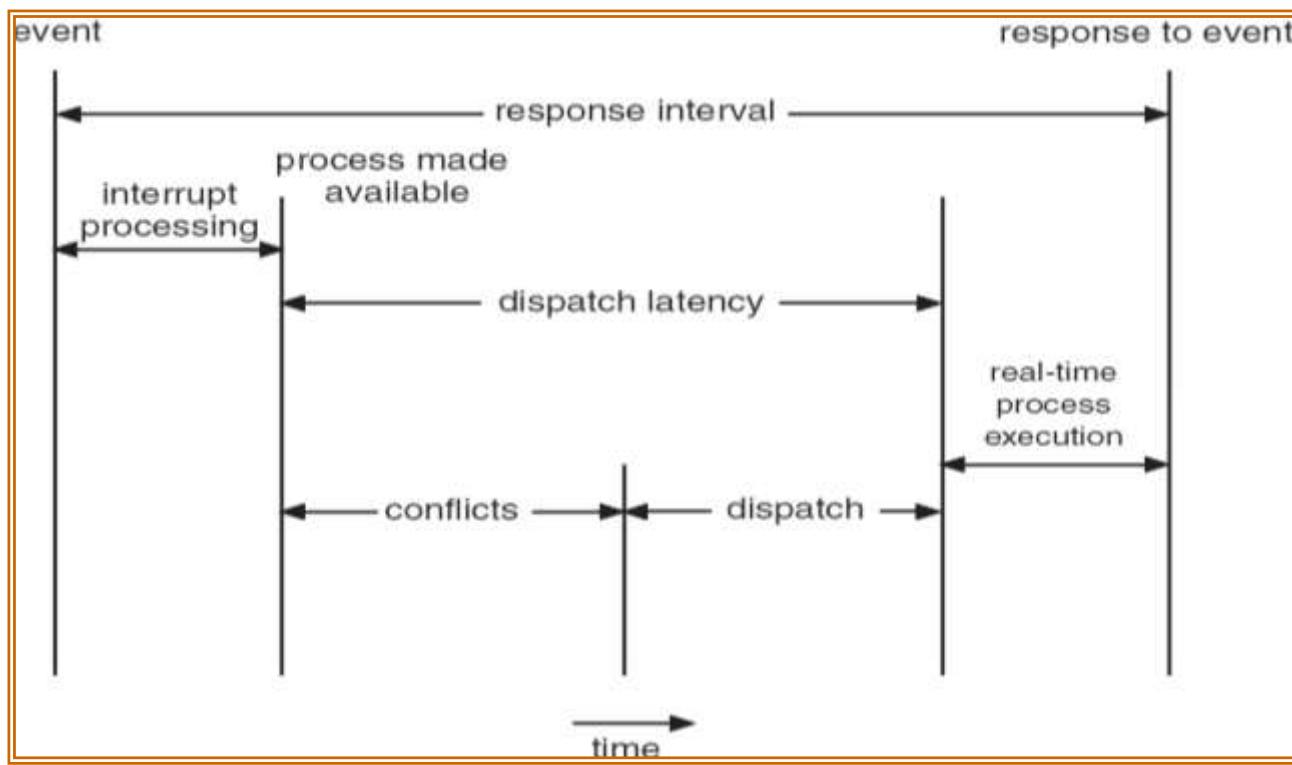
Interrupt Latency

- Interrupt latency is the period of time from when an interrupt arrives at the CPU to when it is serviced.



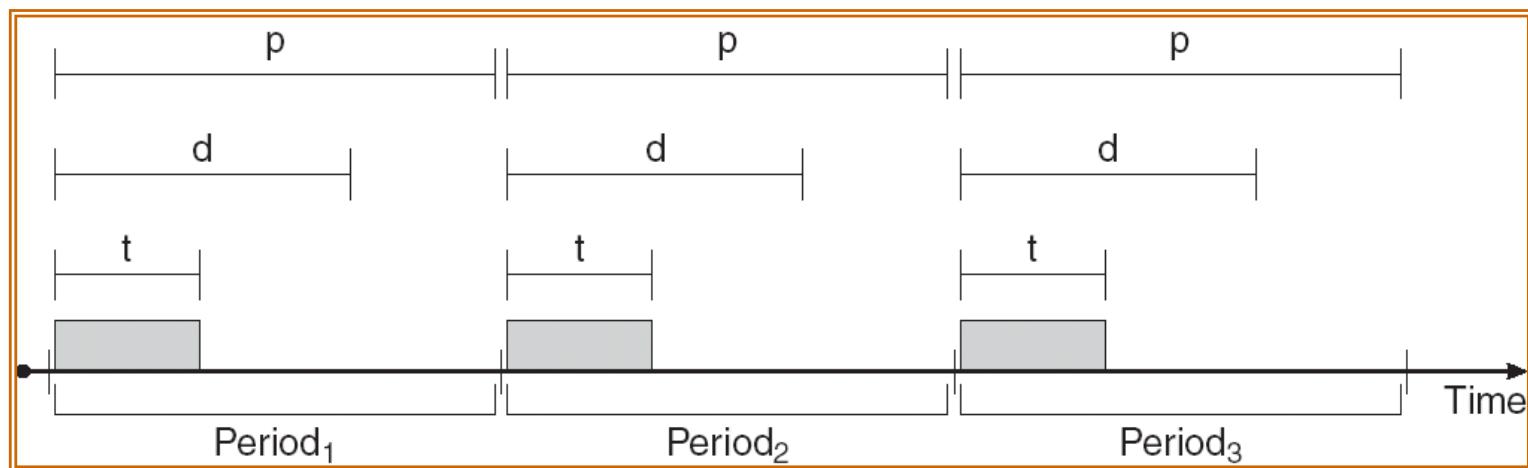
Dispatch Latency

- **Dispatch latency** is the amount of time required for the scheduler to stop one process and start another.



Real-Time CPU Scheduling

- Periodic processes require the CPU at specified intervals (periods)
- p is the duration of the period
- d is the deadline by when the process must be serviced
- t is the processing time



Rate Montonic Scheduling

- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority
- P_1 is assigned a higher priority than P_2 .

Earliest Deadline First Scheduling

- Priorities are assigned according to deadlines:
the earlier the deadline, the higher the priority;
the later the deadline, the lower the priority.

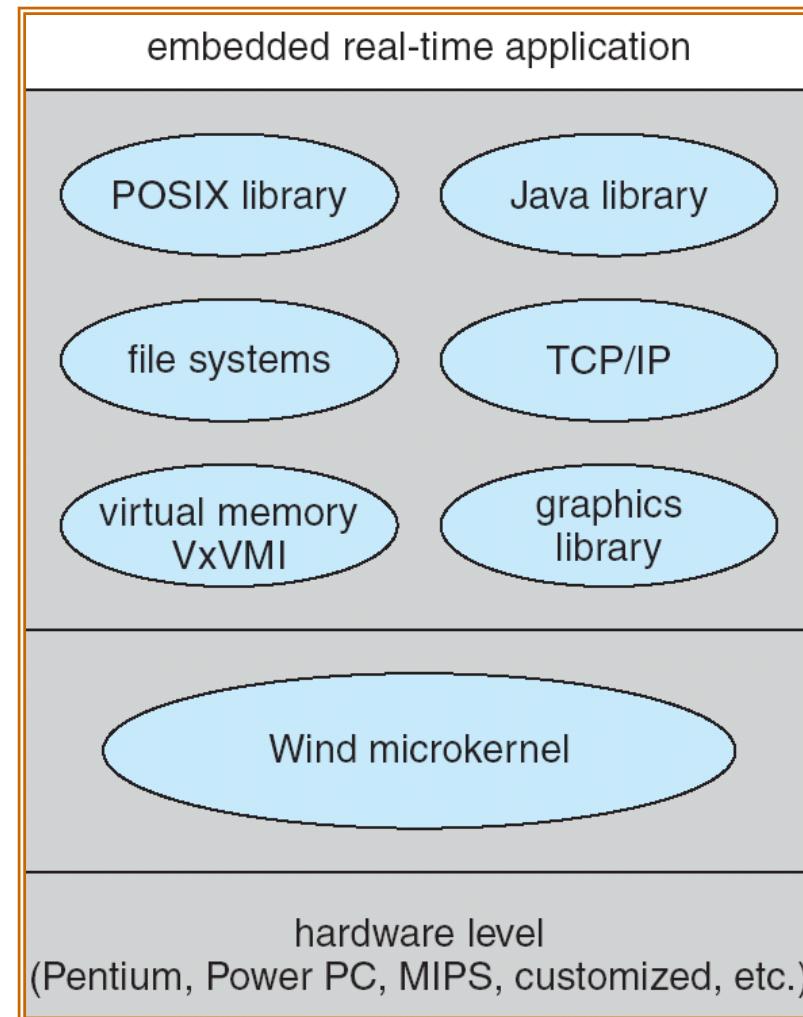
Proportional Share Scheduling

- T shares are allocated among all processes in the system.
- An application receives N shares where $N < T$.
- This ensures each application will receive N / T of the total processor time.

Pthread Scheduling

- The Pthread API provides functions for managing real-time threads.
- Pthreads defines two scheduling classes for real-time threads:
 - (1) SCHED_FIFO - threads are scheduled using a FCFS strategy with a FIFO queue. There is no time-slicing for threads of equal priority.
 - (2) SCHED_RR - similar to SCHED_FIFO except time-slicing occurs for threads of equal priority.

VxWorks 5.0



Wind Microkernel

<https://en.wikipedia.org/wiki/VxWork>

- **VxWorks** is a real-time operating system (RTOS) developed as proprietary software by Wind River Systems, a wholly owned subsidiary of TPG Capital, US.
- First released in 1987, VxWorks is designed for use in embedded systems requiring real-time, deterministic performance and, in many cases,
 - ☞ safety and security certification, for industries, such as aerospace and defense, medical devices, industrial equipment, robotics, energy, transportation, network infrastructure, automotive, and consumer electronics.
- The Wind microkernel provides support for the following:
 - (1) Processes and threads;
 - (2) preemptive and non-preemptive round-robin scheduling;
 - (3) manages interrupts (with bounded interrupt and dispatch latency times);
 - (4) shared memory and message passing interprocess communication facilities.

Solaris 2 scheduling

- Priority based process scheduling
- Four classes of scheduling: Real time, System, Time sharing and Interactive
- Each class includes different priorities and scheduling algorithms.
- A process starts with one LWP and is able to create new LWPs as needed.
- Each LWP inherits the scheduling class and priority of the parent process.
- Default scheduling class is time sharing.
- The scheduling policy for time sharing dynamically alters priorities and assigns time slices of different lengths using a multilevel feedback queue.
- Inverse relationship between priorities and time slices.
- Interactive process receives high priority
- CPU bound process receives low priority.

Solaris 2 scheduling

- Solaris 2 uses system class to run kernel processes: scheduler and paging daemon.
- The priority of a system process does not change.
- The scheduling class for system class is not time slice.
 - ☞ It runs until it blocks or preempted by a higher priority thread.
- Threads in realtime class are given the highest priority to run among all classes.
 - ☞ Allows a guaranteed response time
 - ☞ In general few processes belong to real time class.
- The selected thread runs until one of the following occurs.
 - ☞ It blocks
 - ☞ It uses its time slice (if it is not a system thread)
 - ☞ It is preempted by a higher-priority thread.
- If multiple threads have same priority, the scheduler uses round-robin queue

Dispatch Table

■ Fields of dispatch table

👉 Priority

☰ Higher number indicates a higher priority

👉 Time quantum

☰ Time quantum of associated priority

👉 Time quantum expired

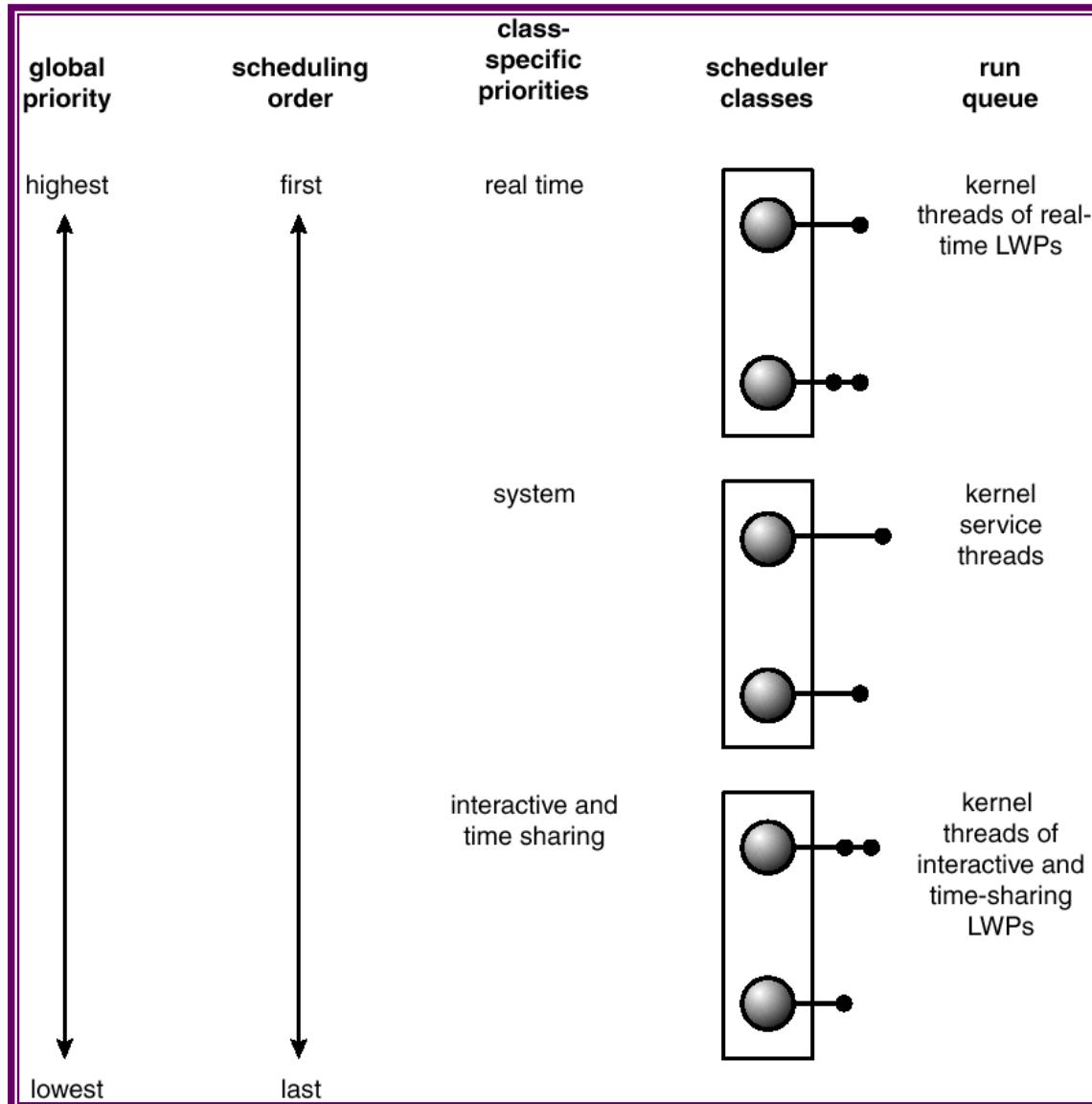
☰ New priority if it consumes entire time quantum
(lowered)

👉 Return from Sleep

☰ Priority of thread returning from sleep

| priority | time quantum | time quantum expired | return from sleep |
|----------|--------------|----------------------|-------------------|
| 0 | 200 | 0 | 50 |
| 5 | 200 | 0 | 50 |
| 10 | 160 | 0 | 51 |
| 15 | 160 | 5 | 51 |
| 20 | 120 | 10 | 52 |
| 25 | 120 | 15 | 52 |
| 30 | 80 | 20 | 53 |
| 35 | 80 | 25 | 54 |
| 40 | 40 | 30 | 55 |
| 45 | 40 | 35 | 56 |
| 50 | 40 | 40 | 58 |
| 55 | 40 | 45 | 58 |
| 59 | 20 | 49 | 59 |

Solaris 2 Scheduling



WINDOWS 2000/XP scheduling

- WINDOWS 2000 schedules threads using a priority-based, preemptive scheduling algorithm.
- Scheduler ensures that highest priority thread always run.
- Scheduler is called dispatcher.
- A thread will run until
 - ☞ Preempted by high priority thread
 - ☞ Until it terminates
 - ☞ Until its time quantum ends
 - ☞ Until it calls a blocking system call.
- The dispatcher uses 32-level priority scheme.
- Priorities are divided into two classes
 - ☞ Variable class (1 to 15)
 - ☞ Real-time class (16-31)
- Priority 0 is used for memory management.
- The dispatcher uses a queue for each scheduling priority and traverses from highest to lowest.
- If no ready thread is found, the dispatcher executes a special thread called idle thread.

WINDOWS 2000 scheduling

- WINDOWS 2000 identifies several priority classes. These include:
 - REALTIME_PRIORITY_CLASS
 - HIGH_PRIORITY_CLASS
 - ABOVE_NORMAL_PRIORITY_CLASS
 - NORMAL_PRIORITY_CLASS
 - BELOW_NORMAL_PRIORITY_CLASS
 - IDLE_PRIORITY_CLASS
- Except REALTIME_PRIORITY_CLASS all other are variable class priorities; the priority the thread belong to this class may change.
- Within each priority class there is a relative priority
 - TIME_CRITICAL
 - HIGHEST
 - ABOVE_NORMAL
 - NORMAL
 - BELOW_NORMAL
 - LOWEST
 - IDLE
- The priority of each thread is based upon the priority class it belongs to and the relative priority within the class.
- In addition each thread has a base priority which is equal to normal priority.
- The initial priority is typically the base priority of the class

Windows 2000 Priorities

| | real-time | high | above normal | normal | below normal | idle priority |
|---------------|-----------|------|--------------|--------|--------------|---------------|
| time-critical | 31 | 15 | 15 | 15 | 15 | 15 |
| highest | 26 | 15 | 12 | 10 | 8 | 6 |
| above normal | 25 | 14 | 11 | 9 | 7 | 5 |
| normal | 24 | 13 | 10 | 8 | 6 | 4 |
| below normal | 23 | 12 | 9 | 7 | 5 | 3 |
| lowest | 22 | 11 | 8 | 6 | 4 | 2 |
| idle | 16 | 1 | 1 | 1 | 1 | 1 |

WINDOWS 2000 scheduling

- When a thread quantum runs out, the thread is interrupted; if the thread is in variable-priority class its priority is lowered.
- When the variable priority thread is released from wait operation, the dispatcher boosts its priority.
 - ☞ Good response time to interactive threads.
 - ☞ Keeps I/O devices busy.
 - ☞ CPU bound processes use CPU cycles background.
- When a process moves to foreground, the time quantum is increased by some factor---typically by three.

LINUX

- Two separate process scheduling algorithms
 - ☞ Time sharing algorithm for fair preemptive scheduling among multiple processes.
 - ☞ Other is designed for real-time tasks
- LINUX allows only user processes to preempt
- A process can not be preempted in kernel mode, even real-time process is available to run.
- Every process is assigned with scheduling class, that defines which of the algorithms to apply to the process.
- First class is for time-sharing processes
 - ☞ LINUX uses prioritized credit based algorithm
 - ☞ Each process possesses a certain number of scheduling credits.
 - ☞ While choosing, the process with most credits is selected.
 - ☞ Every time the interrupt occurs the process loses one credit
 - ☞ When its credits reaches zero it is suspended and another process is chosen.

LINUX (earlier than 2.5)

- If no runnable process have any credits, LINUX performs re-crediting operation. It adds credits to every process
 - ☞ Credits=(Credits/2)+priority
- This algorithm uses two factors: process history and priority.
- Suspended process will accumulate more credits.
- It also gives high priority to interactive processes
- Linux implements two real-time scheduling classes
 - ☞ FCFS and round robin.
- In the FCFS class the process runs until it blocks.
- LINUX real-time scheduling is soft—rather than hard.
- The scheduler guarantees priorities, but kernel never gives guarantees
 - ☞ Kernel process can not preempted

Deadlocks

■ Outline:

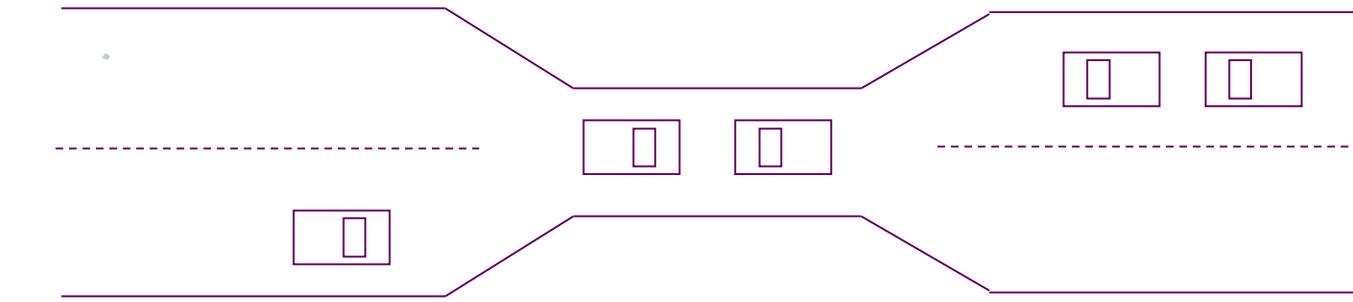
- ☞ Deadlock problem
- ☞ System Model
- ☞ Deadlock Characterization
- ☞ Methods for Handling Deadlocks
- ☞ Deadlock Prevention
- ☞ Deadlock Avoidance
- ☞ Deadlock Detection
- ☞ Recovery from Deadlock
- ☞ Combined Approach to Deadlock Handling

The Deadlock Problem

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.
- Example
 - ☞ System has 2 tape drives.
 - ☞ P_1 and P_2 each hold one tape drive and each needs another one.
- Example
 - ☞ semaphores A and B , initialized to 1

| | |
|------------------|----------------|
| P_0 | P_1 |
| <i>wait (A);</i> | <i>wait(B)</i> |
| <i>wait (B);</i> | <i>wait(A)</i> |

Bridge Crossing Example



- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.

System Model

- A set of processes is in a deadlock state when every process in the set is waiting for an event that can be caused by another process in the set.
- Events:
 - ☞ Resource acquisition and release.
- Resource types R_1, R_2, \dots, R_n
 - ☞ *Physical resources: CPU cycles, memory space, I/O devices*
 - ☞ *Logical resources: files, semaphores, and monitors*
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
 - ☞ Request: if the request is not granted , then it must wait.
 - ☞ Use: The process can operate on the resource.
 - ☞ Release: The process releases the resources.
- Multi-threaded programs are good candidates for deadlock because multiple threads compete for shared resources.

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use at least one resource. Resources are non sharable.
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.
- **No preemption:** Resources can not be preempted; that is a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- **Circular wait:** there exists a set $\{P_0, P_1, \dots, P_{n-1}, P_n, P_0\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2 , ..., P_{n-1} is waiting for a resource that is held by P_n , and P_n is waiting for a resource that is held by P_0 .

ALL FOUR CONDITIONS MUST HOLD.

Resource-Allocation Graph

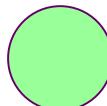
A tool to describe the deadlock.

A set of vertices V and a set of edges E .

- V is partitioned into two types:
 - ☞ $P = \{P_1, P_2, \dots, P_n\}$, the set consisting of all the processes in the system.
 - ☞ $R = \{R_1, R_2, \dots, R_m\}$, the set consisting of all resource types in the system.
- request edge – directed edge $P_i \rightarrow R_j$
- assignment edge – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph (Cont.)

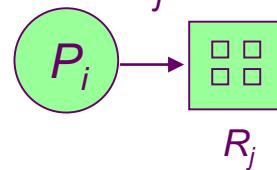
- Process



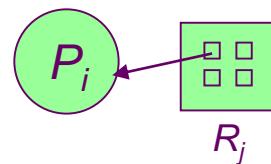
- Resource Type with 4 instances



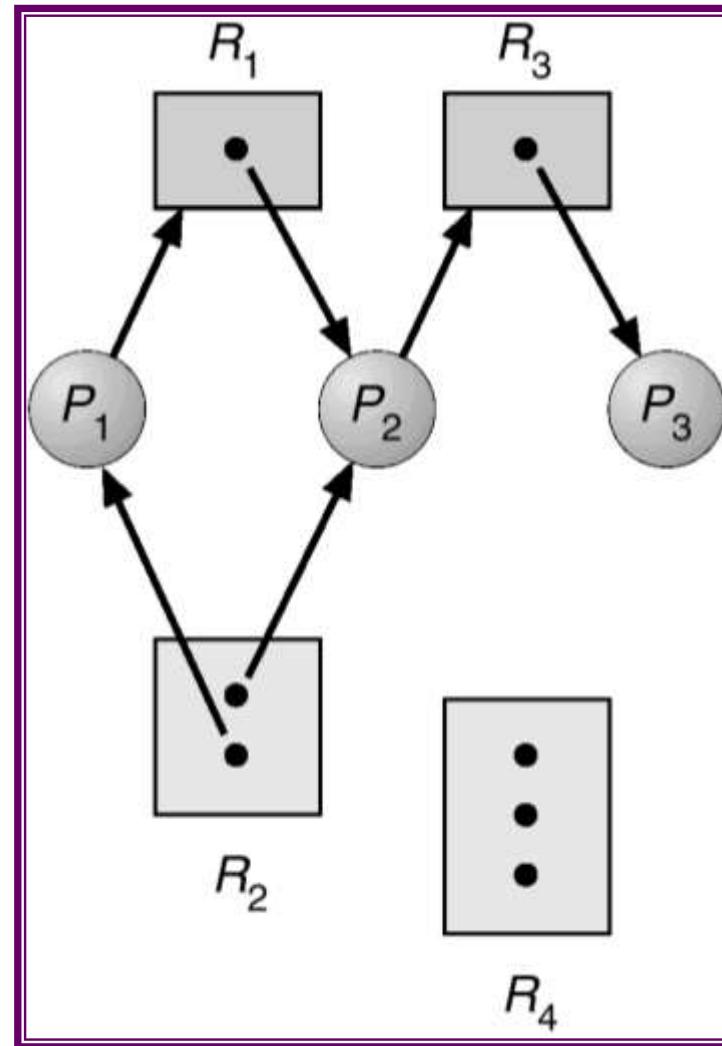
- P_i requests instance of R_j



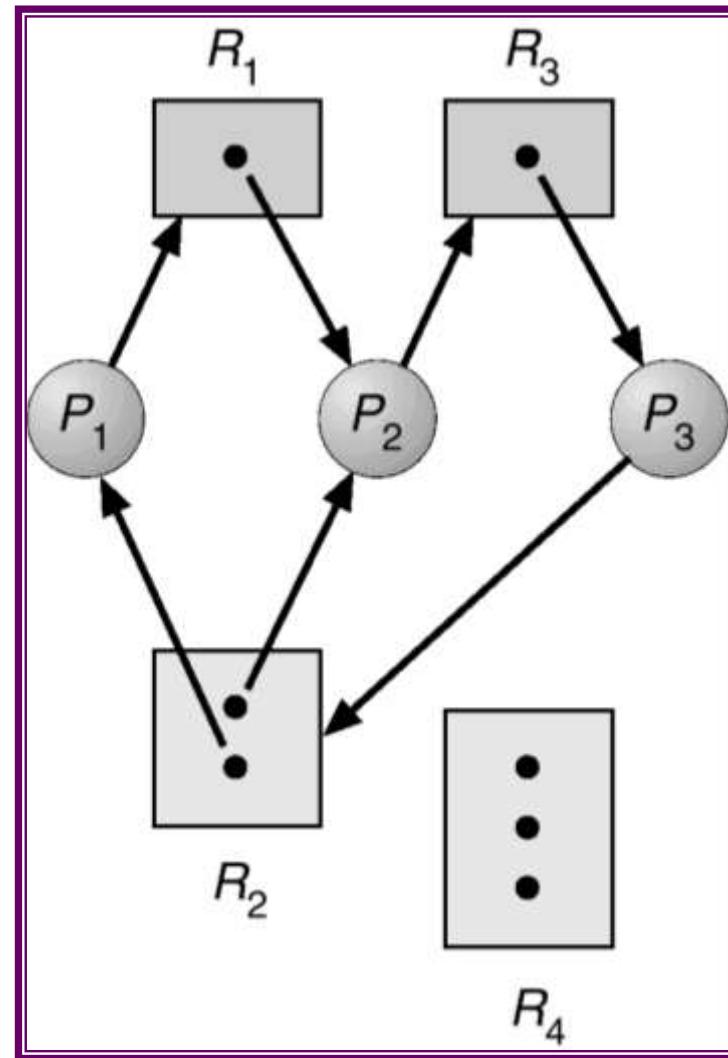
- P_i is holding an instance of R_j



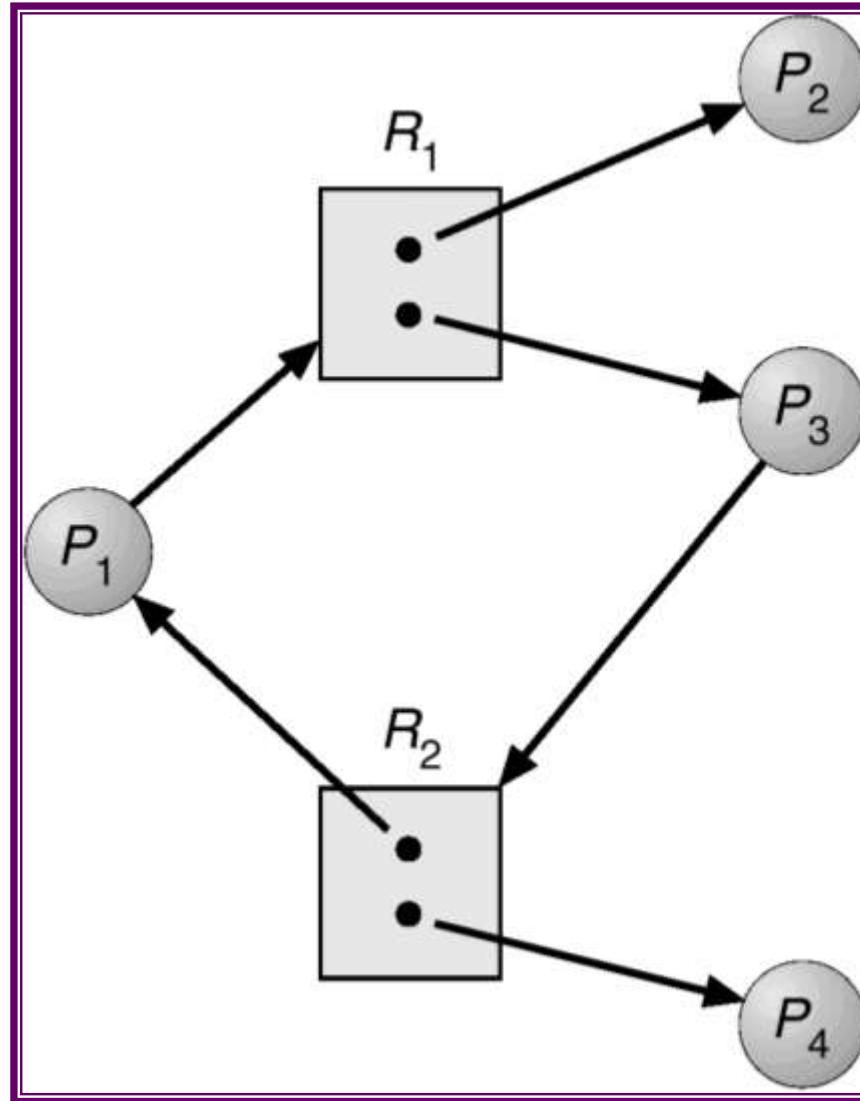
Example of a Resource Allocation Graph



Resource Allocation Graph With A Deadlock



Resource Allocation Graph With A Cycle But No Deadlock



Basic Facts

- If graph contains no cycles \Rightarrow no deadlock.
- If graph contains a cycle \Rightarrow
 - ☞ if only one instance per resource type, then deadlock.
 - ☞ if several instances per resource type, possibility of deadlock.

Methods for Handling Deadlocks

■ Three ways

- ☞ Ensure that the system will *never* enter a deadlock state.
- ☞ Allow the system to enter a deadlock state and then recover.
- ☞ Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

Methods for Handling Deadlocks

- To ensure that deadlocks never occur, the system can use either deadlock prevention and deadlock-avoidance schemes.
 - ☞ Deadlock prevention: set of methods for ensuring that at least one of the necessary conditions can not hold.
 - ☞ Deadlock avoidance: Requires that operating system be given in advance additional information concerning which resources a process will request and use during its life time.
- Deadlock detection: examines the state of the system to determine whether a deadlock has occurred or not.
- Alternatively, assume that deadlock would not occur. Resort to manual recovery when performance degrades due to deadlock.

Deadlock Prevention

- Deadlock prevention is a set of methods for ensuring that at least one of the necessary conditions can not hold.

Deadlock Prevention

Restrain the ways request can be made.

- **Mutual Exclusion** – It is not possible to prevent deadlocks through ME, as some resources are intrinsically non-sharable.
- **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources.
 - ☞ Protocol 1: Requires a process to request and be allocated all its resources before it begins execution.
 - ▣ System calls requesting resources for a process precede all other system calls.
 - ☞ Protocol 2: Allow process to request resources only when the process has none.
 - ▣ A process can request some resources and use them. Before, it can request additional resources, it must release all the resources that it is currently allocated.

Deadlock Prevention

■ Hold and Wait – Example

- ☞ Consider a process that copies data from tape drive to a disk file and then prints the results to the printer.
- ☞ Protocol1: If all the resources are requested at the beginning of a process, the process must request the tape drive, disk file, and printer.
 - ☞ It will the printer during entire execution, even though it needs at the end.
- ☞ Protocol 2: It copies data from tape drive to disk file and releases them. It gain requests disk file and printer.

■ Disadvantages:

- ☞ Resource utilization is very low
 - ☞ Resource may be allocated but unused for a long time.
- ☞ Starvation is possible
 - ☞ A process that needs several resources may have to wait indefinitely, at least one of the resources that it needs is always allocated to another process.
 - ☞ Inefficient
 - ☞ Delays process initiation
 - ☞ Future resource requirements must be known

Deadlock Prevention (Cont.)

■ No Preemption –

- ☞ Protocol: If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
- ☞ Preempted resources are added to the list of resources for which the process is waiting.
- ☞ Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.
- ☞ Possible for CPU registers and memory space whose state can be restored later, but not possible for printers and tape drives.

Deadlock Prevention (Cont.)

■ Circular Wait –

- ☞ $F:R \rightarrow N$ is a one-to-one function, where N is a set of natural numbers.
- ☞ **Protocol 1:** impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.
 - ☞ A process initially can request any number of instances of type R_i . Then the process can request instances of a resource type R_j if and only if $F(R_j) > F(R_i)$.
- ☞ **Protocol 2:** Whenever a process requests an instance of resource type R_j , it has released any other resources R_i such that $F(R_i) >= F(R_j)$.
- ☞ Ordering

Deadlock prevention

■ Circular Wait:

- “**witness**” system call is implemented in BSD UNIX for lock order verifier
- “**witness**” uses mutual-exclusion locks to protect critical regions and maintains the relationship of lock orders in the system
- Suppose thread_one aquires locks in the order first_mutex and second_mutex. Then **witness** records that first_mutex should be obtained before second_mutex. If thread_two later aquires locks out of order, **witness** generates a warning message.

```
thread_one
{
    lock(first_mutex);
    lock(second_mutex);
    .
    .
}
```

```
thread_two:
{
    lock(second_mutex);
    lock (first_mutex);
    .
    .
}
```

Deadlock Avoidance

- Deadlock prevention algorithms
 - ☞ Low device utilization, and reduced system throughput.
- Alternative method is get additional information about the processes
 - ☞ Resources currently available, resources currently allocated to a process, and future requests and releases of each process.
- Various algorithms differ about amount and type of information required.

Deadlock Avoidance

Requires that the system has some additional *a priori* information available.

- Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need.
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes.

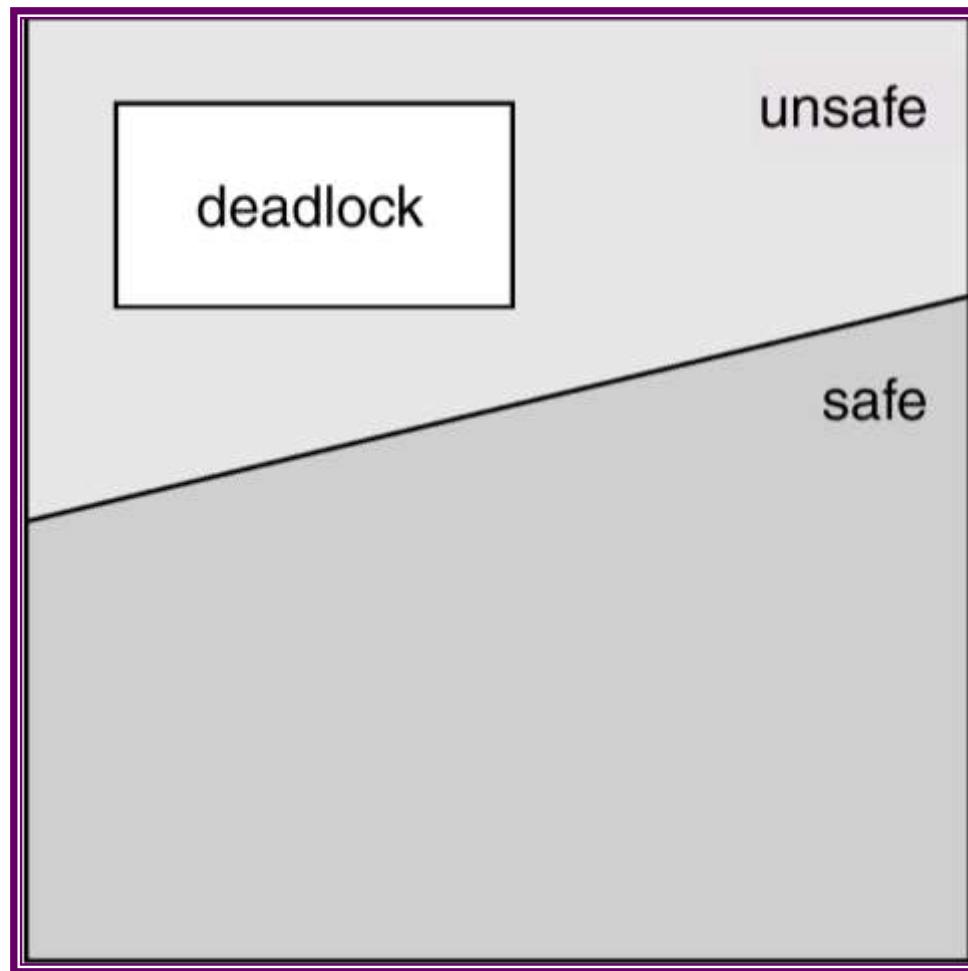
Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a *safe state*.
- System is in safe state if there exists a safe sequence of all processes.
- Sequence $\langle P_1, P_2, \dots, P_n \rangle$ is safe if for each P_i , the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j , with $j < i$.
 - ☞ If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.
 - ☞ When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
 - ☞ When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

Basic Facts

- If a system is in safe state \Rightarrow no deadlocks.
- If a system is in unsafe state \Rightarrow possibility of deadlock.
- Avoidance \Rightarrow ensure that a system will never enter an unsafe state.

Safe, Unsafe , Deadlock State



Example

- Consider a system with 12 magnetic tape drives and three processes P0,P1,P2.
- P0 requires 10 tape drives, P1 may need 4, and P2 may need up to 9 tape drives.
- Suppose at time t0, P0 is holding 5 tape drives, P1 holding 2, and P2 is holding 2 tape drives.
- There are 3 free tape drives

| | Maximum needs | allocated | current needs |
|----|---------------|-----------|---------------|
| P0 | 10 | 5 | 5 |
| P1 | 4 | 2 | 2 |
| P2 | 9 | 2 | 7 |

- At t0, system is in safe condition.
- $\langle P1, P0, P2 \rangle$ satisfies safety condition.
- Suppose, at t1, P2 requests and is allocated 1 more tape drive.
 - ☞ The system is no longer in safe state.
- The mistake was in granting the request of P2 for one more tape drive.
- The main idea of avoidance algorithm is to ensure the system is always in a safe state.

Two algorithms for Deadlock Avoidance

- Resource-Allocation Graph algorithm
- Bankers algorithm

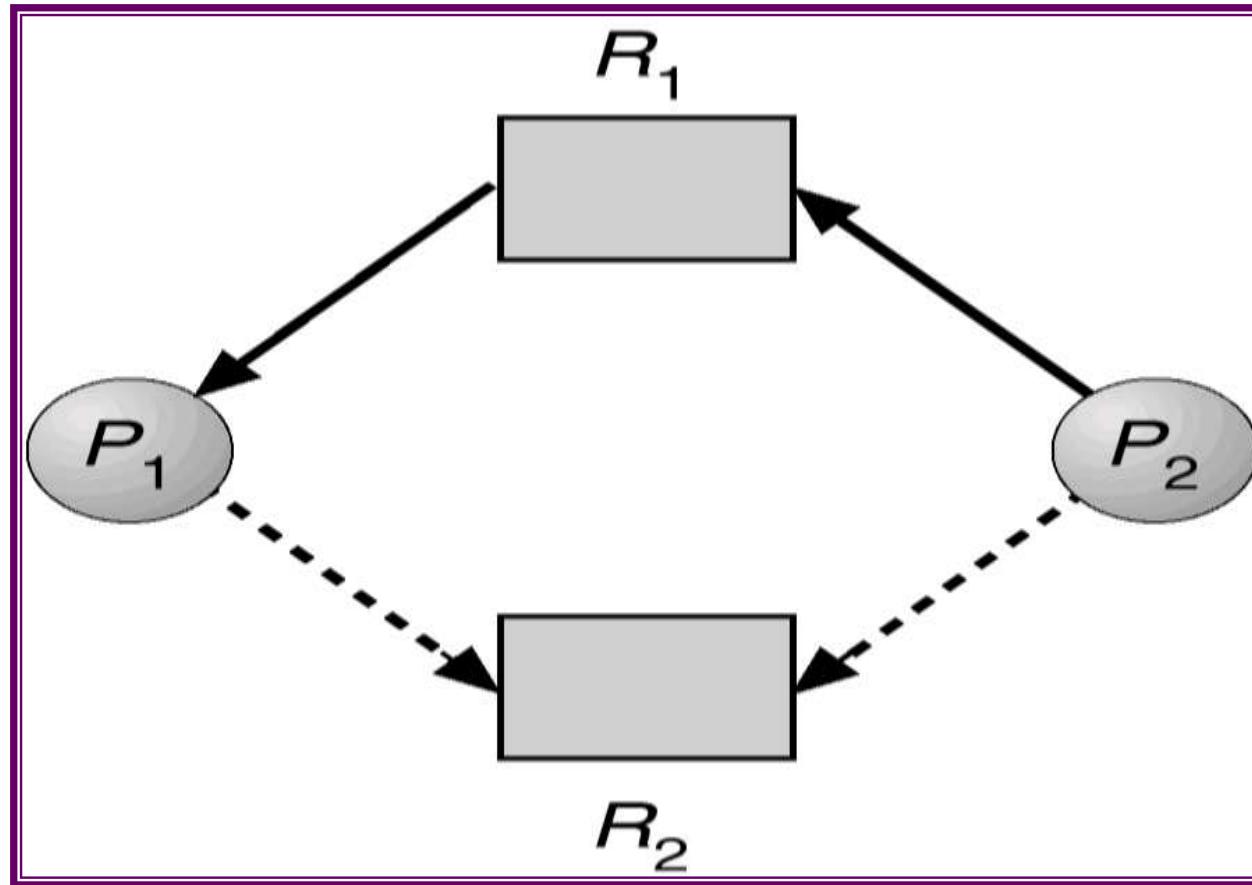
Resource-Allocation Graph Algorithm

- If we have a resource-allocation (RA) system with one instance of each resource type, we can use this approach.
- *Claim edge* $P_i \rightarrow R_j$ indicated that process P_i may request resource R_j ; represented by a dashed line.
- Claim edge converts to request edge when a process requests a resource.
- When a resource is released by a process, assignment edge reconverts to a claim edge.
- Resources must be claimed *a priori* in the system.
- Before the process starts executing, all its claim edges must appear in the RA graph.
- Suppose a process P_i requests a resource R_j . The request can be granted if converting the request edge $P_i \rightarrow R_j$ to an assignment edge $R_j \rightarrow P_i$ does not result in the formation of a cycle in the RA graph.

Resource-Allocation Graph Algorithm

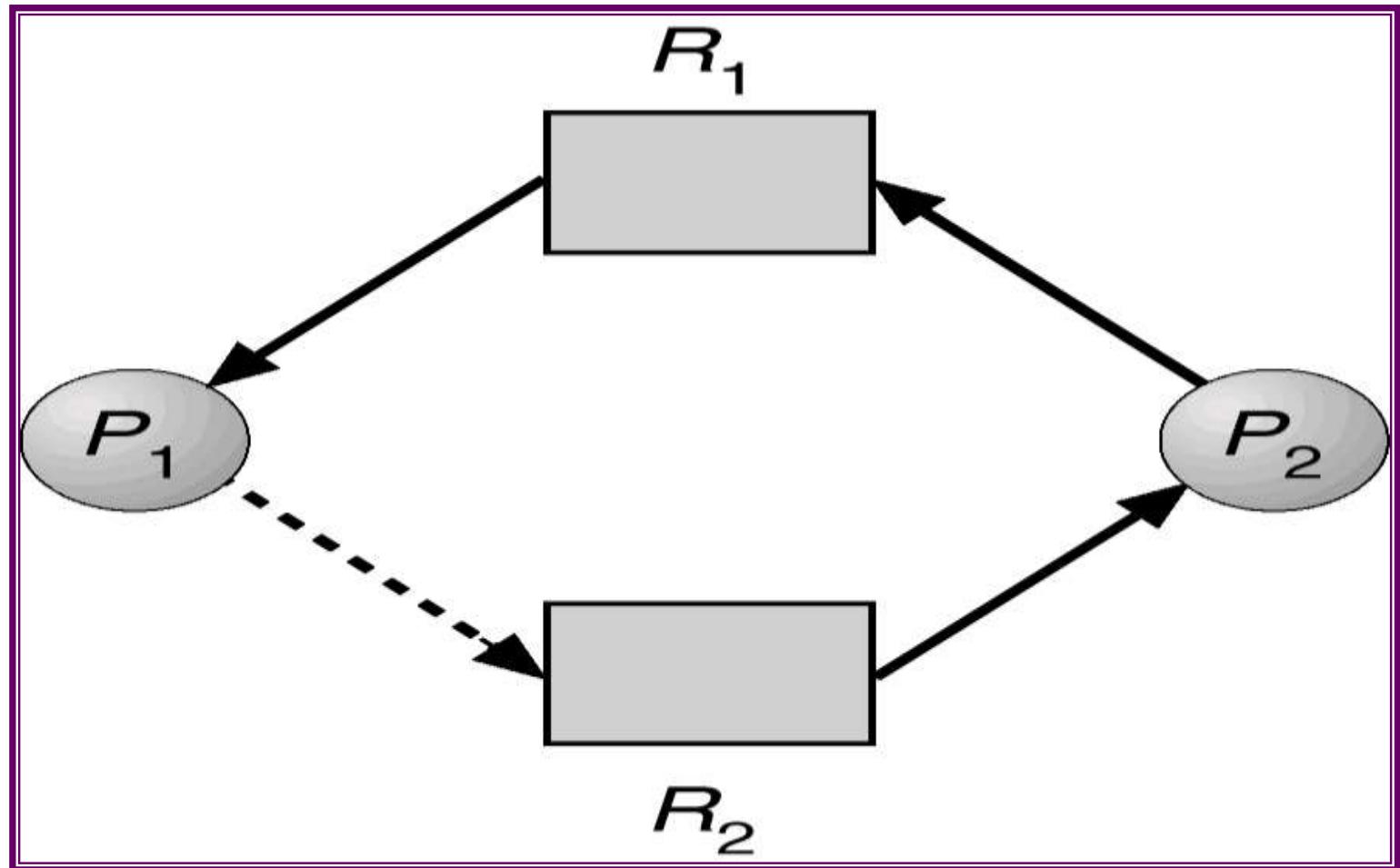
- Cycle detection algorithm is used to detect a cycle.
- If there are n processes detecting a cycle requires an order of n^*n operations.

Resource-Allocation Graph For Deadlock Avoidance



Suppose P2 requests R2

Unsafe State In Resource-Allocation Graph



Cycle in RA graph! So system is in an unsafe state. So we can not allocate R2 to P2.

Banker's Algorithm

- Bank never allocates its available cash such that it no longer satisfy the needs of all customers.
- Multiple instances.
- Each process must a priori claim maximum use.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.

Data Structures for the Banker's Algorithm

Let n = number of processes, and m = number of resources types.

- **Available:** Vector of length m . If $\text{available}[j] = k$, there are k instances of resource type R_j available.
- **Max:** $n \times m$ matrix. If $\text{Max}[i,j] = k$, then process P_i may request at most k instances of resource type R_j .
- **Allocation:** $n \times m$ matrix. If $\text{Allocation}[i,j] = k$ then P_i is currently allocated k instances of R_j .
- **Need:** $n \times m$ matrix. If $\text{Need}[i,j] = k$, then P_i may need k more instances of R_j to complete its task.

$$\text{Need}[i,j] = \text{Max}[i,j] - \text{Allocation}[i,j].$$

- Notation:
 - ☞ Let X and Y be vectors of length n .
 - ☞ We say $X \leq Y$ if and only if $X[i] \leq Y[i]$ for all $i=1,2,\dots,n$.
 - ☞ For example, if $X=(1,7,3,2)$ and $Y=(0,3,2,1)$ then $Y \leq X$.
 - ☞ $Y < X$ if $Y \leq X$ and $Y \neq X$.

Resource-Request Algorithm for Process P_i

Request = request vector for process P_i . If $\text{Request}_i[j] = k$ then process P_i wants k instances of resource type R_j .

1. If $\text{Request}_i \leq \text{Need}_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $\text{Request}_i \leq \text{Available}$, go to step 3. Otherwise P_i must wait, since resources are not available.
3. Pretend to allocate requested resources to P_i by modifying the state as follows:

$$\text{Available} = \text{Available} - \text{Request}_i;$$

$$\text{Allocation}_i = \text{Allocation}_i + \text{Request}_i;$$

$$\text{Need}_i = \text{Need}_i - \text{Request}_i;$$

- **If safe** \Rightarrow the resources are allocated to P_i .
- **If unsafe** $\Rightarrow P_i$ must wait, and the old resource-allocation state is restored

Safety Algorithm

1. Let $Work$ and $Finish$ be vectors of length m and n , respectively. Initialize:

$Work = Available$

$Finish[i] = false$ for $i = 1, 2, \dots, n$.

2. Find and i such that both:

(a) $Finish[i] = false$

(b) $Need_i \leq Work$

If no such i exists, go to step 4.

3. $Work = Work + Allocation_i$,

$Finish[i] = true$

go to step 2.

4. If $Finish[i] == true$ for all i , then the system is in a safe state.

- Safety algorithm may require an order of $m \times n^2$ operations.

Example of Banker's Algorithm

- 5 processes P_0 through P_4 ; 3 resource types A (10 instances), B (5 instances, and C (7 instances).
- Snapshot at time T_0 :

| | <u>Allocation</u> | | | <u>Max</u> | | | <u>Available</u> | | |
|-------|-------------------|----------|----------|------------|----------|----------|------------------|----------|----------|
| | <i>A</i> | <i>B</i> | <i>C</i> | <i>A</i> | <i>B</i> | <i>C</i> | <i>A</i> | <i>B</i> | <i>C</i> |
| P_0 | 0 | 1 | 0 | 7 | 5 | 3 | 3 | 3 | 2 |
| P_1 | 2 | 0 | 0 | 3 | 2 | 2 | | | |
| P_2 | 3 | 0 | 2 | 9 | 0 | 2 | | | |
| P_3 | 2 | 1 | 1 | 2 | 2 | 2 | | | |
| P_4 | 0 | 0 | 2 | 4 | 3 | 3 | | | |

Example (Cont.)

- The content of the matrix. Need is defined to be Max – Allocation.

| | <u>Need</u> | | |
|-------|-------------|---|---|
| | A | B | C |
| P_0 | 7 | 4 | 3 |
| P_1 | 1 | 2 | 2 |
| P_2 | 6 | 0 | 0 |
| P_3 | 0 | 1 | 1 |
| P_4 | 4 | 3 | 1 |

- The system is in a safe state since the sequence $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ satisfies safety criteria.

Example P_1 Request (1,0,2) (Cont.)

- Check that Request \leq Available (that is, $(1,0,2) \leq (3,3,2)$ \Rightarrow true.

| | <u>Allocation</u> | | | <u>Need</u> | | | <u>Available</u> | | |
|-------|-------------------|---|---|-------------|---|---|------------------|---|---|
| | A | B | C | A | B | C | A | B | C |
| P_0 | 0 | 1 | 0 | 7 | 4 | 3 | 2 | 3 | 0 |
| P_1 | 3 | 0 | 2 | 0 | 2 | 0 | | | |
| P_2 | 3 | 0 | 1 | 6 | 0 | 0 | | | |
| P_3 | 2 | 1 | 1 | 0 | 1 | 1 | | | |
| P_4 | 0 | 0 | 2 | 4 | 3 | 1 | | | |

- Executing safety algorithm shows that sequence $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ satisfies safety requirement.
- Can request for (3,3,0) by P_4 be granted?
- Can request for (0,2,0) by P_0 be granted?

Deadlock Detection

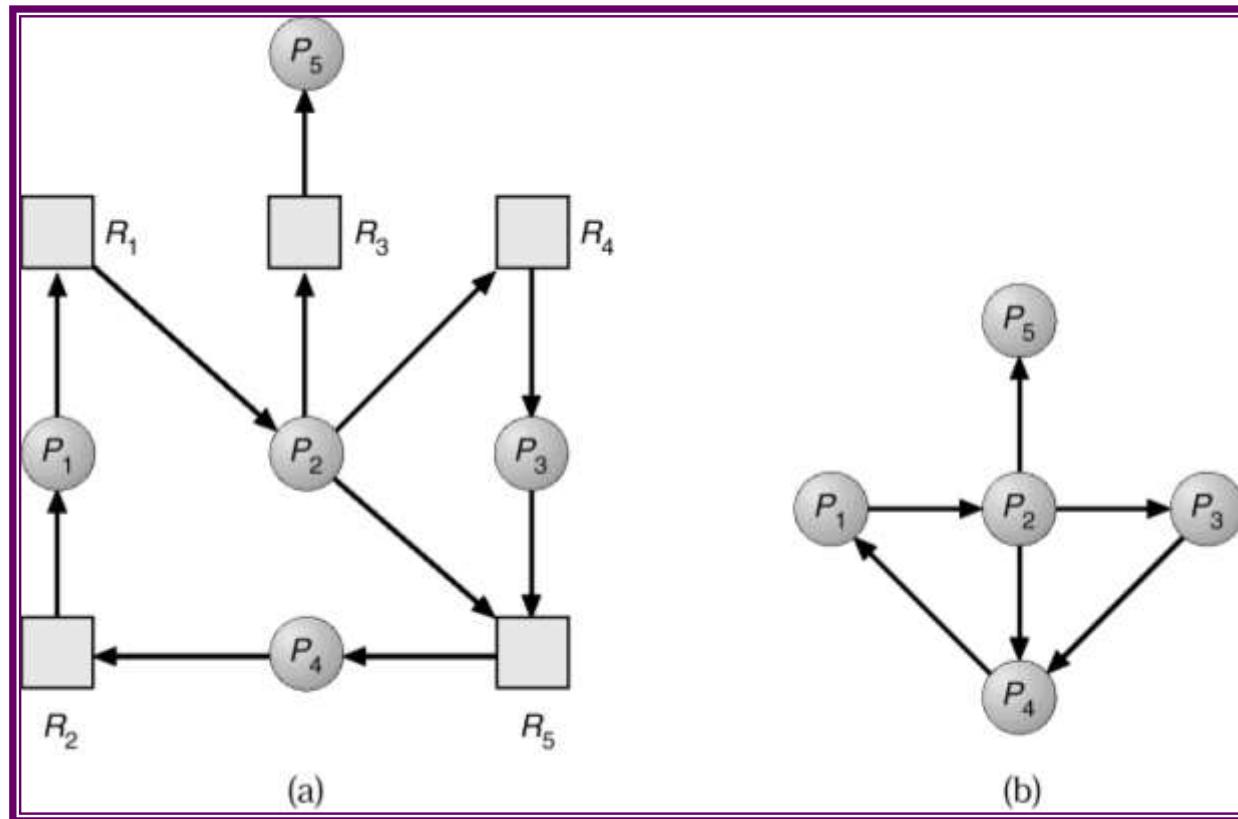
- Allow system to enter deadlock state
- Detection algorithm
 - ☞ Resource allocation graph algorithm
 - ☞ Detection algorithm (similar to Banker's algorithm)
- Recovery scheme

Single Instance of Each Resource

Graph based approach

- Maintain *wait-for* graph
 - ☞ Nodes are processes.
 - ☞ $P_i \rightarrow P_j$ if P_i is waiting for P_j .
- Periodically invoke an algorithm that searches for a cycle in the graph.
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph.

Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph

Corresponding wait-for graph

Several Instances of a Resource Type

- **Available:** A vector of length m indicates the number of available resources of each type.
- **Allocation:** An $n \times m$ matrix defines the number of resources of each type currently allocated to each process.
- **Request:** An $n \times m$ matrix indicates the current request of each process. If $\text{Request}[i_j] = k$, then process P_i is requesting k more instances of resource type. R_j .

Detection Algorithm

1. Let $Work$ and $Finish$ be vectors of length m and n , respectively Initialize:

- (a) $Work = Available$
- (b) For $i = 1, 2, \dots, n$, if $Allocation_i \neq 0$, then $Finish[i] = \text{false}$; otherwise, $Finish[i] = \text{true}$.

2. Find an index i such that both:

- (a) $Finish[i] == \text{false}$
- (b) $Request_i \leq Work$

If no such i exists, go to step 4.

Detection Algorithm (Cont.)

3. $Work = Work + Allocation_i$
 $Finish[i] = true$
go to step 2.
4. If $Finish[i] == \text{false}$, for some i , $1 \leq i \leq n$,
then the system is in deadlock state.
Moreover, if $Finish[i] == \text{false}$, then P_i is
deadlocked.
Algorithm requires an order of $O(m \times n^2)$
operations to detect whether the system is in
deadlocked state.

Example of Detection Algorithm

- Five processes P_0 through P_4 ; three resource types A (7 instances), B (2 instances), and C (6 instances).
- Snapshot at time T_0 :

| | <u>Allocation</u> | <u>Request</u> | <u>Available</u> |
|-------|-------------------|----------------|------------------|
| | A B C | A B C | A B C |
| P_0 | 0 1 0 | 0 0 0 | 0 0 0 |
| P_1 | 2 0 0 | 2 0 2 | |
| P_2 | 3 0 3 | 0 0 0 | |
| P_3 | 2 1 1 | 1 0 0 | |
| P_4 | 0 0 2 | 0 0 2 | |

- Sequence $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ will result in $Finish[i] = \text{true}$ for all i .

Example (Cont.)

- P_2 requests an additional instance of type C.

Request

| | A | B | C |
|-------|---|---|---|
| P_0 | 0 | 0 | 0 |
| P_1 | 2 | 0 | 1 |
| P_2 | 0 | 0 | 1 |
| P_3 | 1 | 0 | 0 |
| P_4 | 0 | 0 | 2 |

- State of system?

- ☞ Can reclaim resources held by process P_0 , but insufficient resources to fulfill other processes' requests.
- ☞ Deadlock exists, consisting of processes P_1 , P_2 , P_3 , and P_4 .

Detection-Algorithm Usage

- When, and how often, to invoke depends on:
 - ☞ How often a deadlock is likely to occur?
 - ☞ How many processes will need to be rolled back?
 - ☞ one for each disjoint cycle
- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes “caused” the deadlock.
- Invoking for every request is expensive
 - ☞ Invoke once per hour; when CPU utilization drops below 40 %.

Recovery from Deadlock: Process Termination

■ Several alternatives exist.

- ☞ Inform the operator
- ☞ Automatic: system will recover from the deadlock automatically
 - ☞ Two options exist for automatically breaking a deadlock
 - Process termination
 - Resource preemption

Recovery from Deadlock: Process Termination

- Abort all deadlocked processes.
 - ☞ More expensive
- Abort one process at a time until the deadlock cycle is eliminated.
 - ☞ Overhead: after the abort of each process detection algorithm have to be invoked.
- Aborting a process is not easy.
 - ☞ Printing or updating a file
- In which order should we choose to abort? (should incur minimum cost)
 - Priority of the process.
 - How long process has computed, and how much longer to completion.
 - Resources the process has used.
 - Resources process needs to complete.
 - How many processes will need to be terminated.
 - Is process interactive or batch?

Recovery from Deadlock: Resource Preemption

- We preempt some resources from one process and give it to other processes. Until the deadlock cycle is broken.
- For preemption three issues are to be addressed
 - ☞ Selecting a victim – minimize cost.
 - ❑ Number of resources the deadlocked process is holding, time consumed by it for execution.
 - ☞ Rollback – return to some safe state, restart process for that state.
 - ❑ Rollback as far as necessary to break the deadlock.
 - ☞ Starvation – same process may always be picked as victim,
 - ❑ Solution: include number of rollbacks in cost factor.

Deadlock Handling: summary

■ Three basic approaches

☞ **Prevention:** Use some protocol to prevent deadlocks, ensuring the system will never enter a deadlock state.

 Low resource utilization

☞ **Avoidance:** Use a priori information; do not allow the system to enter unsafe state.

 Needs a priori information about future requests.

☞ **Detection:** Allow the system to enter deadlock state: then recover.

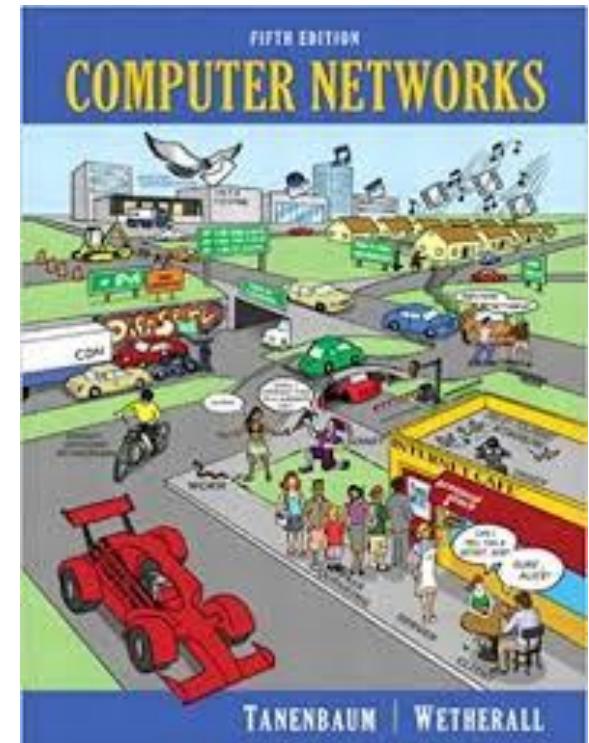
 Select the victim based on the cost factors.

Part Three: Storage Management

- The programs are in the Main memory
 - □ We need a memory management scheme to manage several processes in the main memory
- Main memory is very small
 - Secondary storage is used to support main memory
- We also need file management for on-line storage of access to both data and programs that reside on disk

Announcements

- Topics to be covered
 - memory management (2 classes),
 - virtual memory (2 classes),
 - Disk scheduling and file management (1 class),
 - Protection, security, media management (1 class)
 - overview of computer networks (6 classes).
- Procure the following book.



Memory Management

- Memory is shared by multiple processes
- We discuss various ways to manage memory.
- Outline
 - Background
 - Address binding
 - Dynamic loading and linking
 - Overlays
 - Swapping
 - Memory management approaches
 - Contiguous Allocation
 - Paging
 - Segmentation
 - Segmentation with Paging

Background

- Memory is a central part of modern computer systems
 - Large array of bytes and words each with its own address.
 - CPU fetches and executes instructions
 - After the operation, the result is stored back into memory
 - Memory unit sees only stream of addresses.
-
- Program resides on the disk.
 - Program must be brought into memory and placed within a process for it to be run.
 - *Input queue* – collection of processes on the disk that are waiting to be brought into memory to run the program.
 - User programs go through several steps before being run.

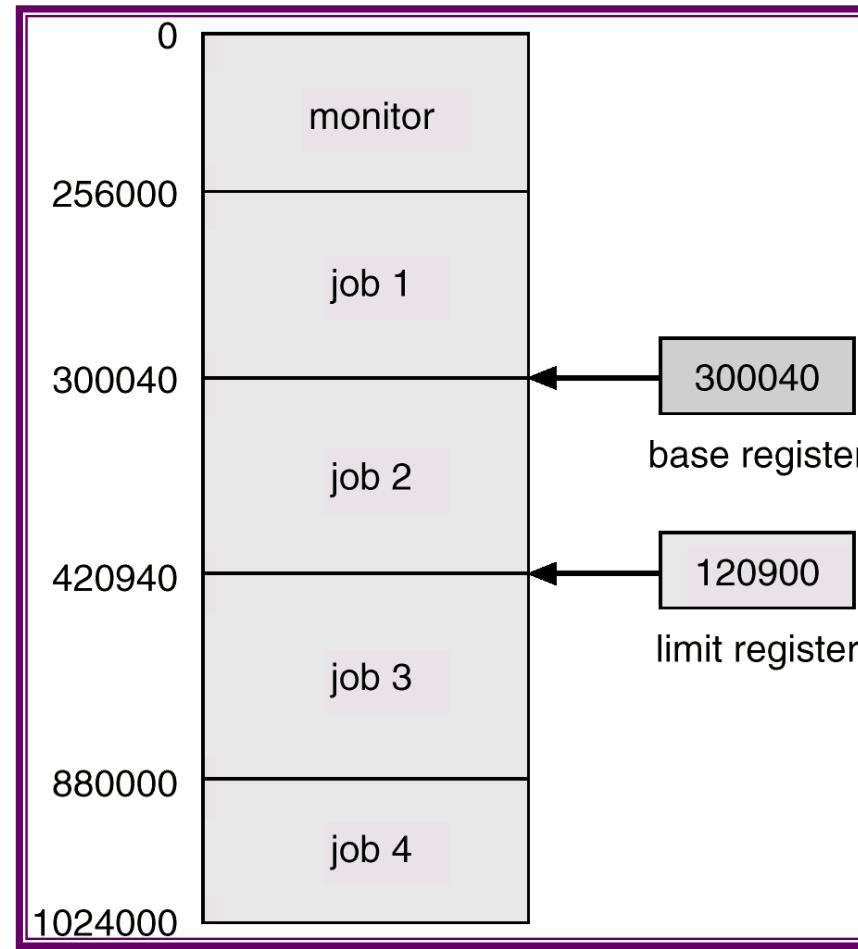
Basic hardware

- Ensuring correct operation
 - Each process has a separate memory space
- Base and limit registers
 - Every address should be compared.

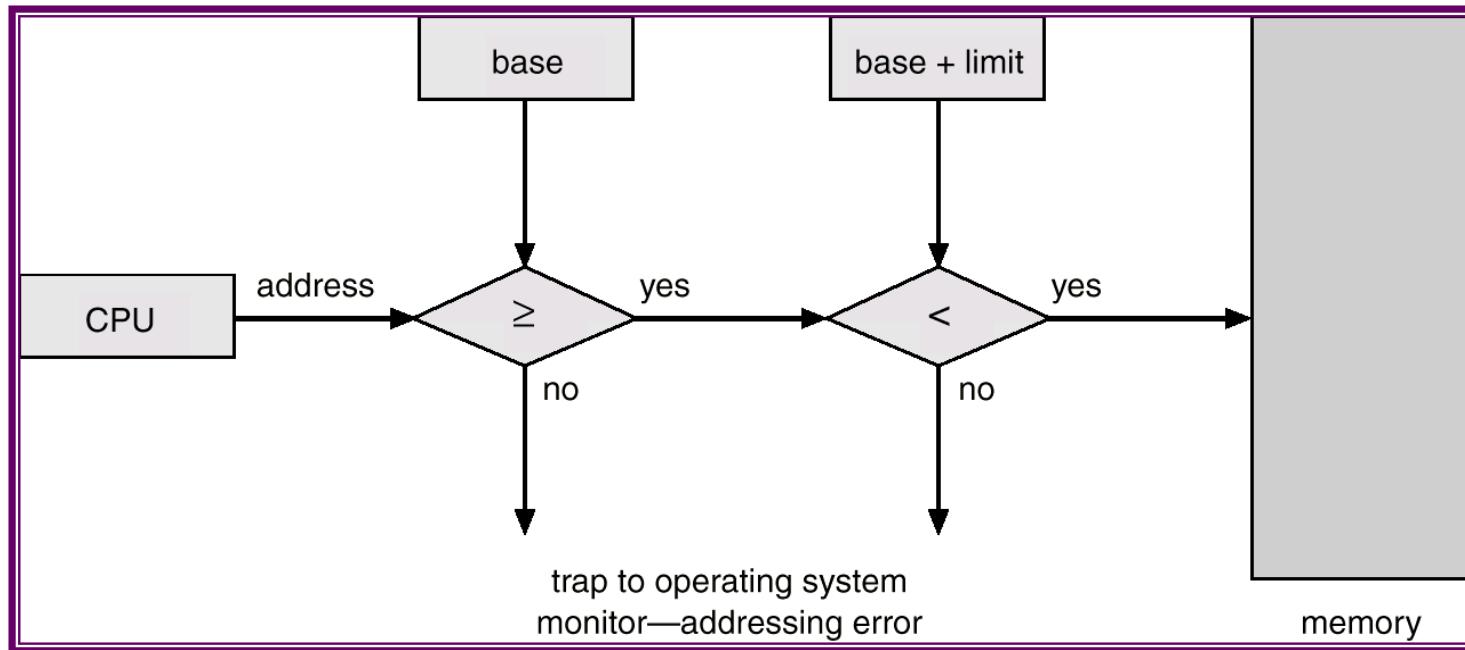
Memory Protection

- Must provide memory protection at least for the interrupt vector and the interrupt service routines.
- In order to have memory protection, add two registers that determine the range of legal addresses a program may access:
 - **Base register** – holds the smallest legal physical memory address.
 - **Limit register** – contains the size of the range
- Memory outside the defined range is protected.

Use of A Base and Limit Register



Hardware Address Protection



Memory Protection

- When executing in monitor mode, the operating system has unrestricted access to both monitor and user's memory.
- The load instructions for the *base* and *limit* registers are privileged instructions.

Address binding

- Most systems allow user process to reside in any part of the physical memory.
- Address space starts at 00000; However, the first address of user process may not start at 00000.
- So user program will go through several steps.
- Addresses in the source program are symbolic.
 - Example: Count
- A compiler will bind these addresses to re-locatable addresses.
 - Example: 14 bytes from the beginning of the module.
- A linkage editor or a loader will bind these re-locatable addresses to absolute addresses.
- A binding is a mapping from one address space to another.

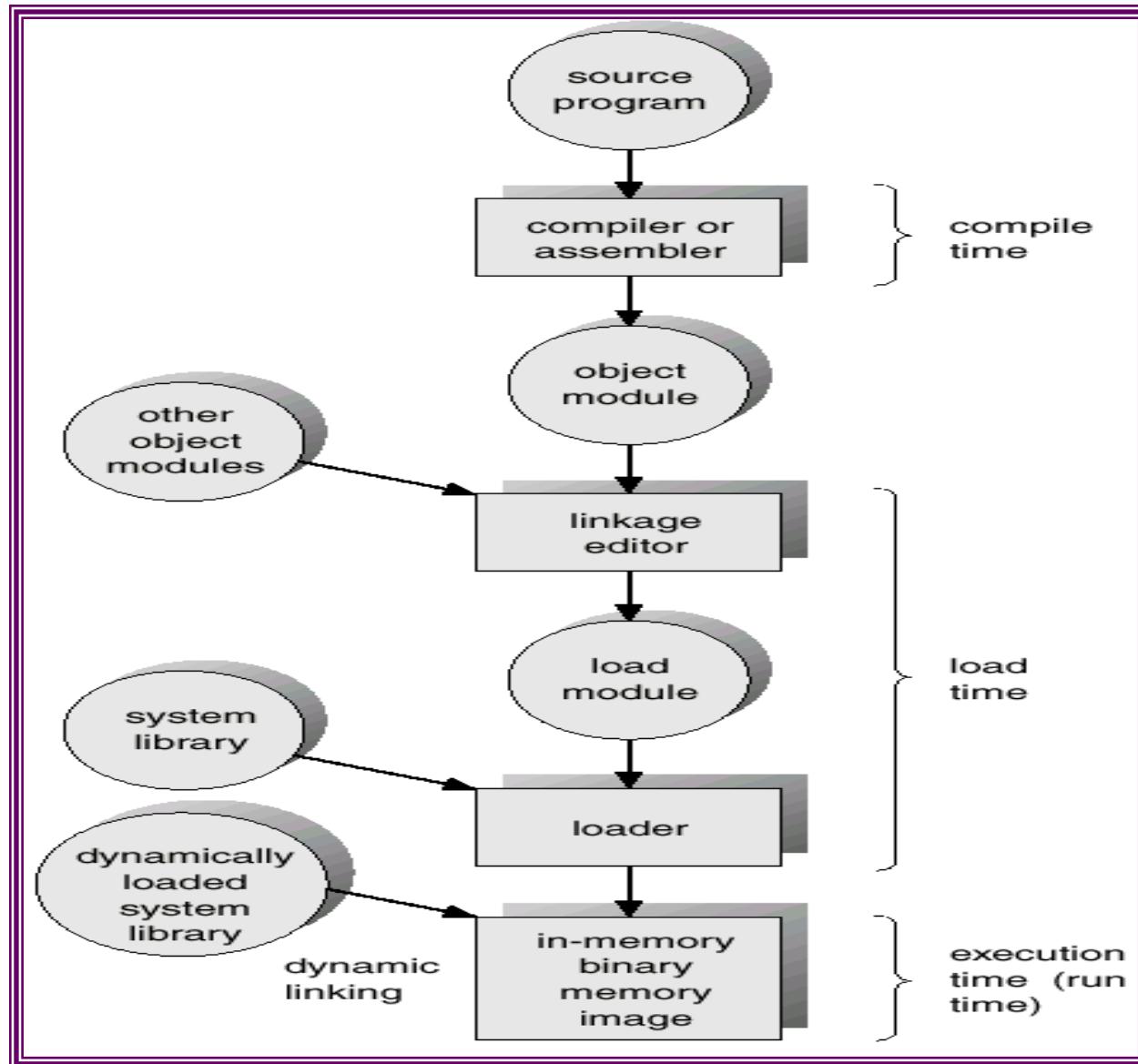
Binding of Instructions and Data to Memory

Address binding of instructions and data to memory addresses can happen at three different stages.

- **Compile time:** If memory location known a priori, absolute code can be generated; must recompile code if starting location changes.
 - MSDOS.COM-format programs are absolute code bound
- **Load time:** Compiler must generate *relocatable* code if memory location is not known at compile time.
 - Final binding is delayed until load time.
 - If the starting address changes, we need only to reload the user code to incorporate the changed value.
- **Execution time:** Binding delayed until run time if the process can be moved during its execution from one memory segment to another.
 - Need hardware support for address maps (e.g., *base* and *limit registers*).
 - Most OSs use this method.

In this chapter we study about various binding schemes and corresponding hardware support.

Multi-step Processing of a User Program



Dynamic Loading

- Dynamic loading is used to obtain better memory-space utilization
- Routine is not loaded until it is called
- All routines are kept on disk in a re-locatable load format
 - Better memory-space utilization; unused routine is never loaded.
 - Useful when large amounts of code are needed to handle infrequently occurring cases.
 - No special support from the operating system is required.
 - Implemented through program design.

Dynamic Linking

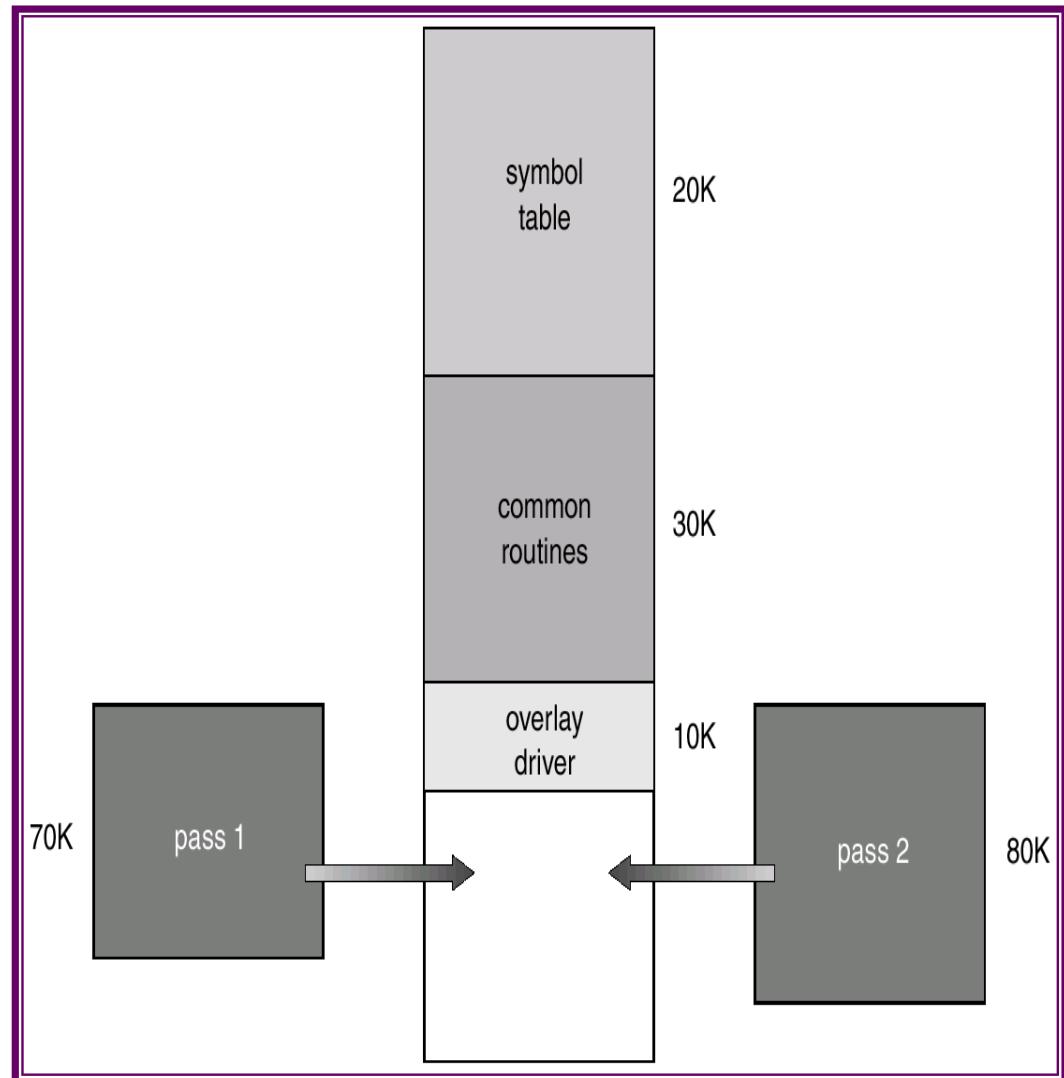
- Some OSs support static linking.
 - System language libraries are treated like any other object module and loader combines them into binary program image.
- In dynamic linking, the linking is postponed until execution time.
- Small piece of code, *stub*, used to locate the appropriate memory-resident library routine.
- Stub replaces itself with the address of the routine, and executes the routine.
- Operating system needed to check if routine is in processes' memory address.
- Dynamic linking is particularly useful for libraries.
 - Library updates; different versions
- Needs support from OS.

Overlays

- To allow a process to be larger than main memory Overlays can be used.
- Keep in memory only those instructions and data that are needed at any given time.
- Needed when process is larger than amount of memory allocated to it.
- Implemented by user, no special support needed from operating system, programming design of overlay structure is complex.
- Example: assembler
 - Pass1: 70K
 - Pass2: 80K
 - Symbol table: 20 K
 - Common routines: 30 K
 - To load everything we need 200K of memory.
 - With 150 K we can not run the program.
 - Two overlays are defined:
 - Overlay A: Symbol table, common routine and pass1
 - Overlay B: Symbol table, common routines and pass2.

Overlays for a Two-Pass Assembler

- After finishing Pass1, the overlay driver reads overlay B, overwriting overlay A, and transfers control to pass 2.
- Special loading and linking algorithms are needed to construct overlays.
- Writing of overlay driver requires complete knowledge of the program structure.
- Limited to microcomputers and embedded systems
 - Lack hardware support for memory management.



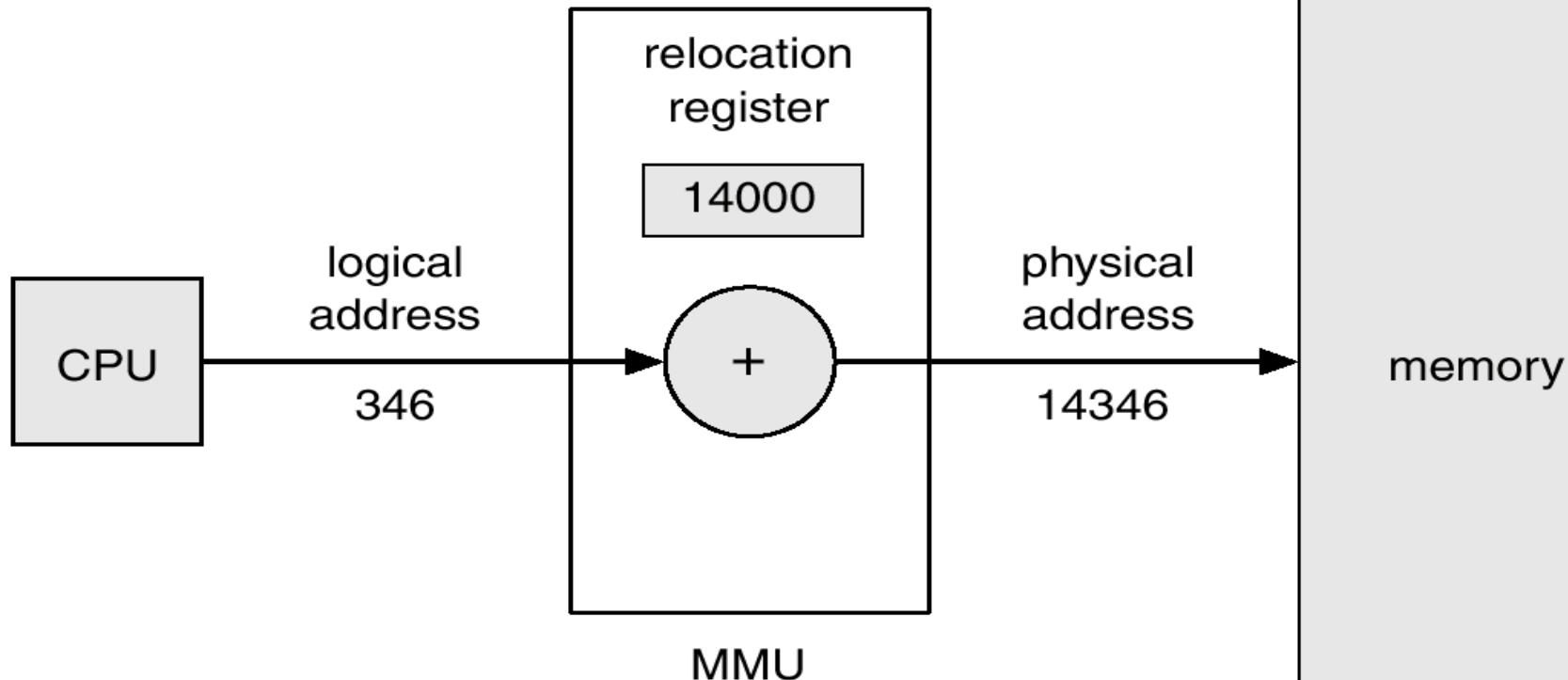
Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate *physical address space* is central to proper memory management.
 - *Logical address* – generated by the CPU; also referred to as *virtual address*.
 - Set of logical addresses generated by a program is called logical address space.
 - *Physical address* – address seen by the memory unit.
 - Set of physical addresses corresponds to logical space is called physical address space.
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme.

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address.
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
- The user program deals with *logical* addresses; it never sees the *real* physical addresses.

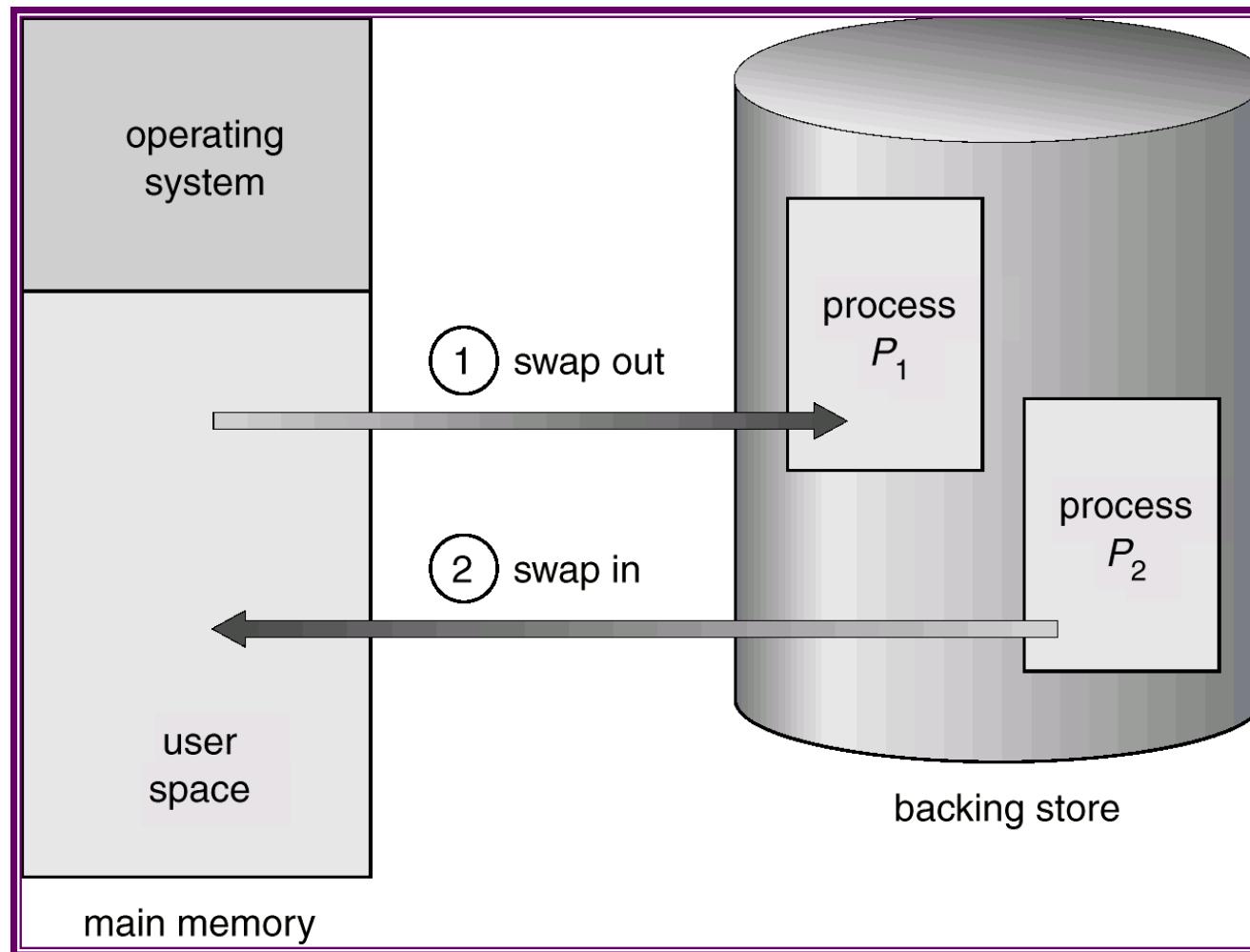
Dynamic relocation using a relocation register



Swapping

- Whenever the CPU scheduler decides to execute a process, it calls the dispatcher.
- The dispatcher checks whether the process is in main memory.
- If the process is not, and there is no free memory, the dispatcher swaps out a process currently in main memory and swaps in the desired process.
- Backing store – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.
- *Roll out, roll in* – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed.
- Major part of swap time is transfer time; total transfer time is directly proportional to the *amount* of memory swapped.
- When a process which is swapped-out swaps-in
 - Binding during assembly and loading
 - Process can not be moved to different locations.
 - Binding during execution
 - Process can be moved to different locations
- Modified versions of swapping are found on many systems, i.e., UNIX, Linux, and Windows.

Schematic View of Swapping



Swapping...

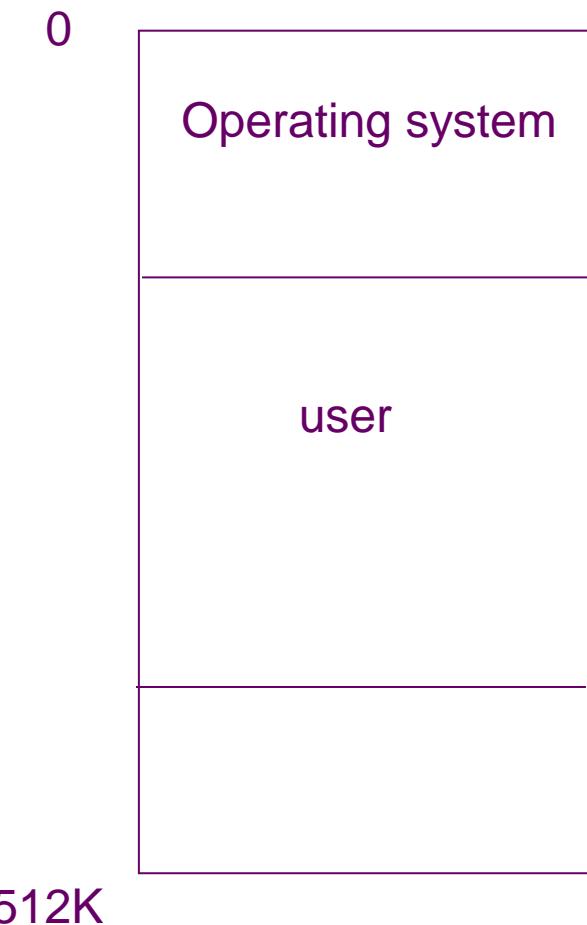
- A process should be idle to swap.
 - It should not have any pending I/Os.
- Solutions
 - Never swap a process with pending I/Os
 - Difficult if the process is accessing asynchronously.
 - Execute I/O operations into OS buffers.
 - Transfers between OS buffers to process occur only when the process is swapped in.
- UNIX
 - Swapping is normally disabled.
 - However, swapping starts if many processes are running and using threshold amount of memory.
 - Swapping is halted when the load is reduced.
- WINDOWS
 - OS does not provide full swapping.
 - The scheduler decides when the process should be swapped out.
 - When the user selects the process, the process is swapped-in.

Outline

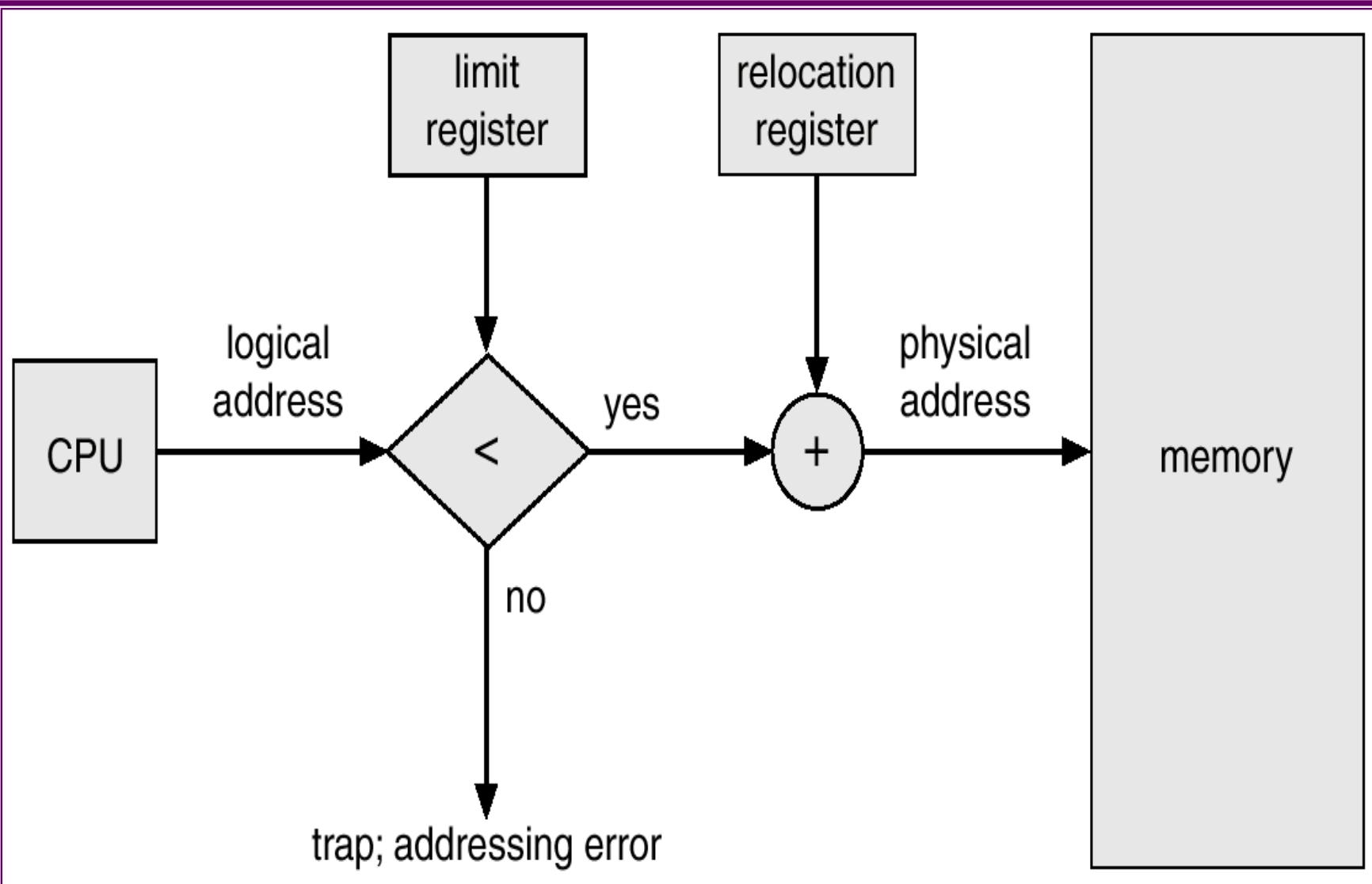
- Background
 - Address binding
 - Dynamic loading and linking
 - Overlays
 - Swapping
- **Memory management approaches**
 - **Contiguous Allocation**
 - **Paging**
 - **Segmentation**
 - **Segmentation with Paging**

Contiguous Allocation

- Main memory usually into two partitions:
 - Resident operating system, usually held in low memory with interrupt vector.
 - User processes then held in high memory.
- Single-partition allocation
 - Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data.
 - Relocation register contains value of smallest physical address; limit register contains range of logical addresses – each logical address must be less than the limit register.
- When a CPU scheduler selects a process from execution, the dispatcher loads the relocation and limit registers with correct values as a part of context switch.



Hardware Support for Relocation and Limit Registers

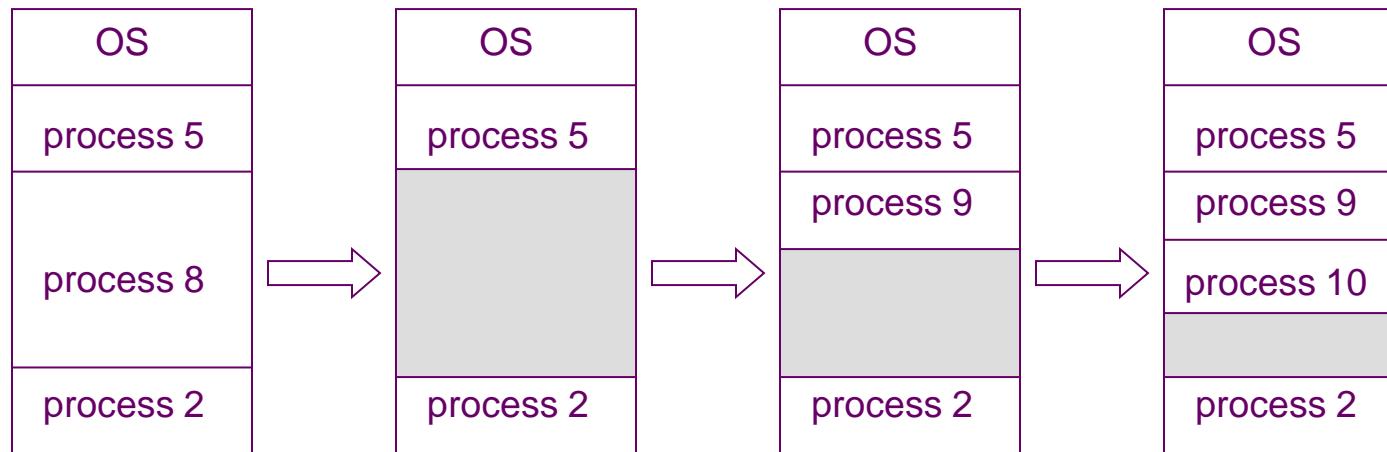


Contiguous Allocation (Cont.)

- Multiple-partition allocation
- How to allocate available main-memory to the various processes ?
- Simple solution (fixed partition solution): partition the memory
 - Number of programs= Number of processes.
 - Not efficient
- Improved solution:
 - Generalization of fixed partition solution.
- OS keeps a table indicating which parts of memory are available and which are occupied.
- Initially all memory available for a user process.
- *Free partition or Hole* – block of available memory; holes of various size are scattered throughout memory.
 - When a process arrives, it is allocated memory from free partitions large enough to accommodate it.
 - Operating system maintains information about:
 - a) allocated partitions
 - b) free partitions (hole)

Contiguous Allocation (Cont.)

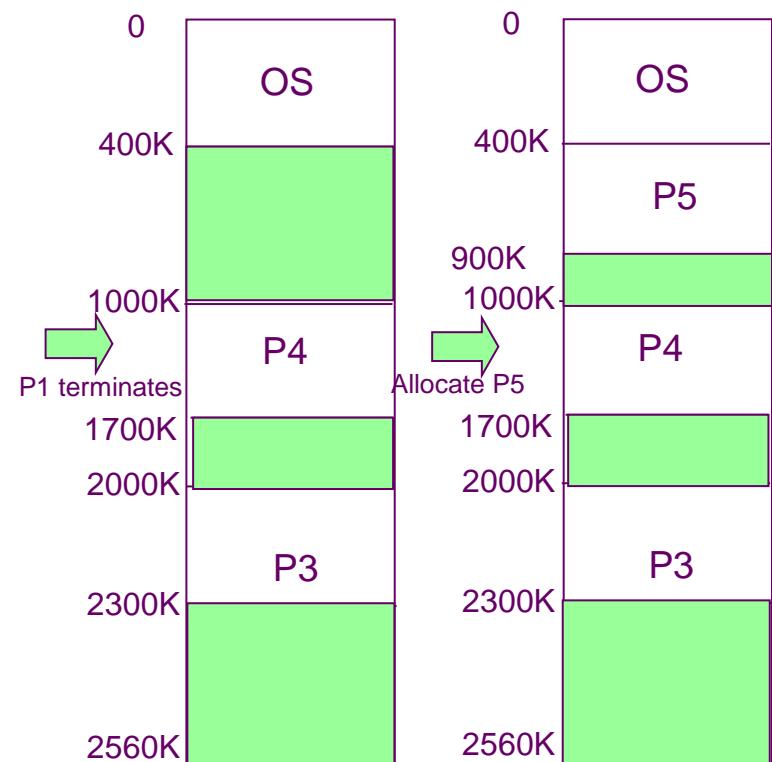
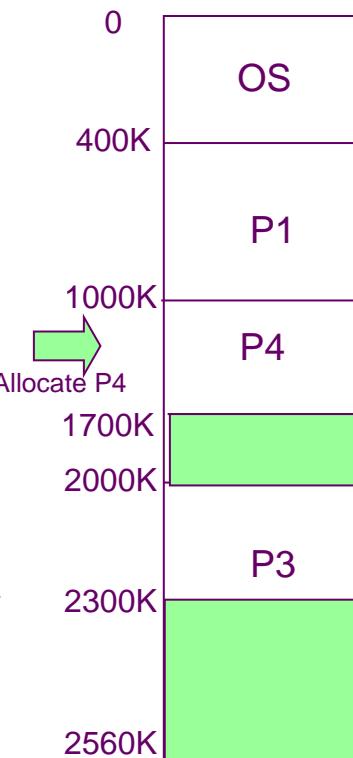
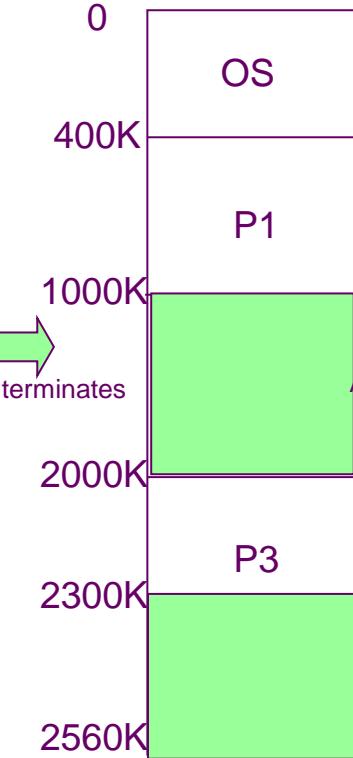
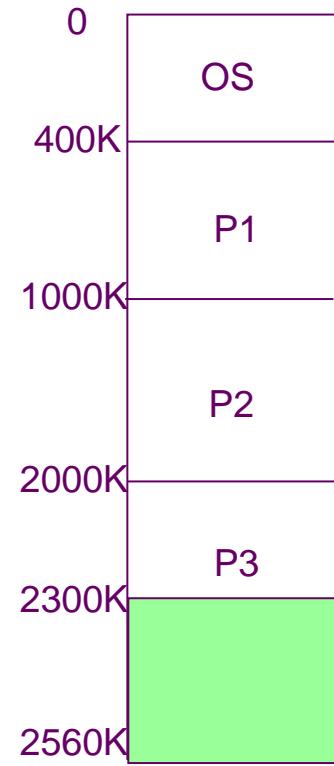
- At any time a set of holes of various sizes scattered throughout the memory
- A free partition large enough for the requesting process is searched.
- If the free partition is too large it is split into two: one is allocated to the arriving process and the other is returned to the set of partitions.
- When a process terminates, it releases its block of memory, which is then placed back in the set of free partitions.
- If the new hole is adjacent to other partitions, we merge these partitions to form a larger hole.
- Example:



Contiguous Allocation (Cont.)

- Example: 2560K of memory available and a resident OS of 400K. Allocate memory to processes P1...P4 following FCFS.
- Shaded regions are holes
- Initially P1, P2, P3 create first memory map.

| Process | Memory | time |
|---------|--------|------|
| P1 | 600K | 10 |
| P2 | 1000K | 5 |
| P3 | 300K | 20 |
| P4 | 700K | 8 |
| P5 | 500K | 15 |



Dynamic Storage-Allocation Problem

Algorithms to search for a free partition of size n.

- **First-fit:** Allocate the *first* free partition that is big enough.
- **Best-fit:** Allocate the *smallest* partition that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.
- **Worst-fit:** Allocate the *largest* partition; must also search entire list. Produces the largest leftover hole.

First-fit and best-fit better than worst-fit in terms of speed and storage utilization.

Fragmentation

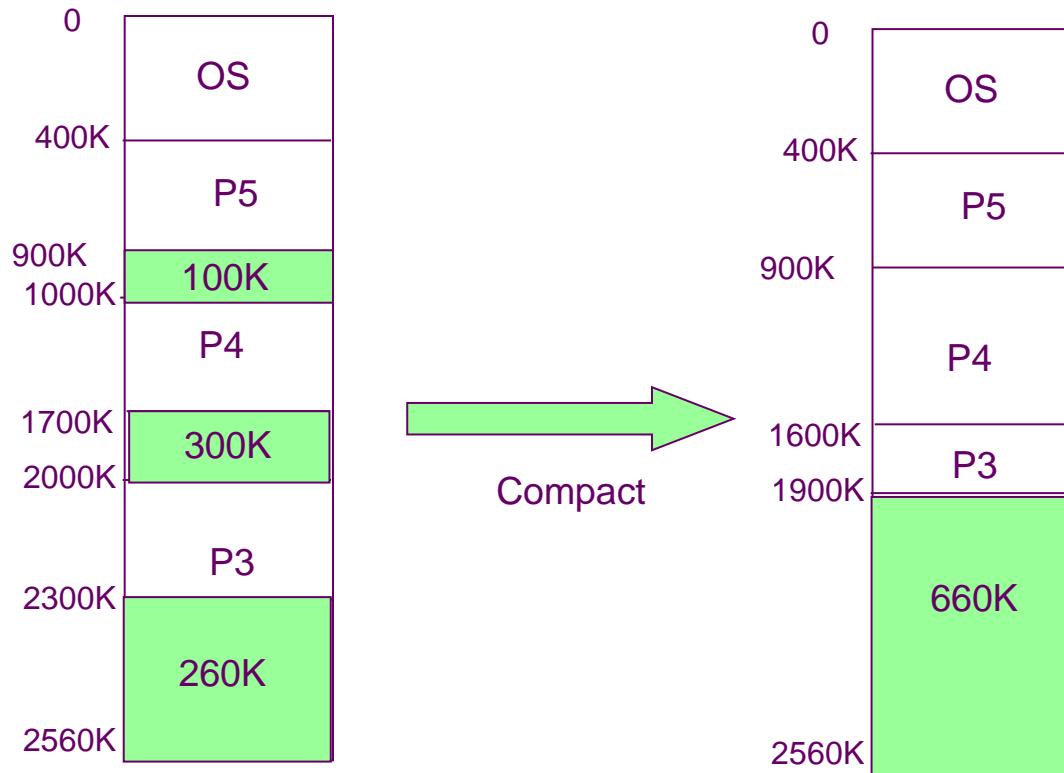
- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous.
 - 50 % rule: Given N allocated blocks 0.5 blocks will be lost due to fragmentation.
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.
 - Consider the hole of 18,464 bytes and process requires 18462 bytes.
 - If we allocate exactly the required block, we are left with a hole of 2 bytes.
 - The overhead to keep track of this free partition will be substantially larger than the hole itself.
 - Solution: allocate very small free partition as a part of the larger request.

Solution to fragmentation

- Compaction
- Paging
- Segmentation

Compaction

- Reduce external fragmentation by compaction
 - Shuffle memory contents to place all free memory together in one large block.
 - Compaction is possible *only* if relocation is dynamic, and is done at execution time.
 - I/O problem
 - Latch job in memory while it is involved in I/O.
 - Do I/O only into OS buffers.
 - Compaction depends on cost.

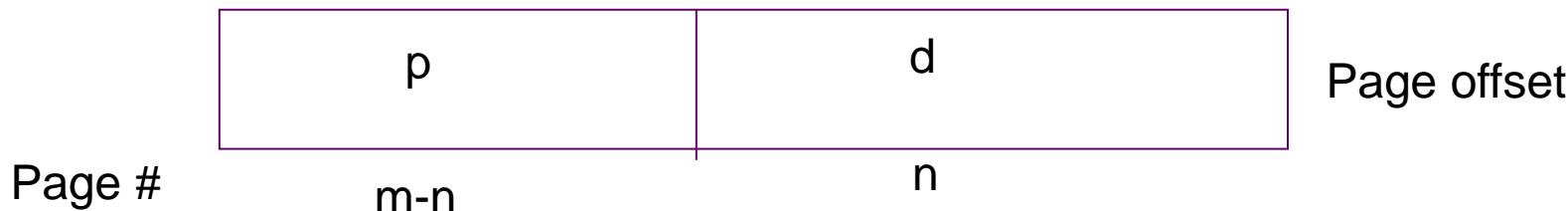


Paging

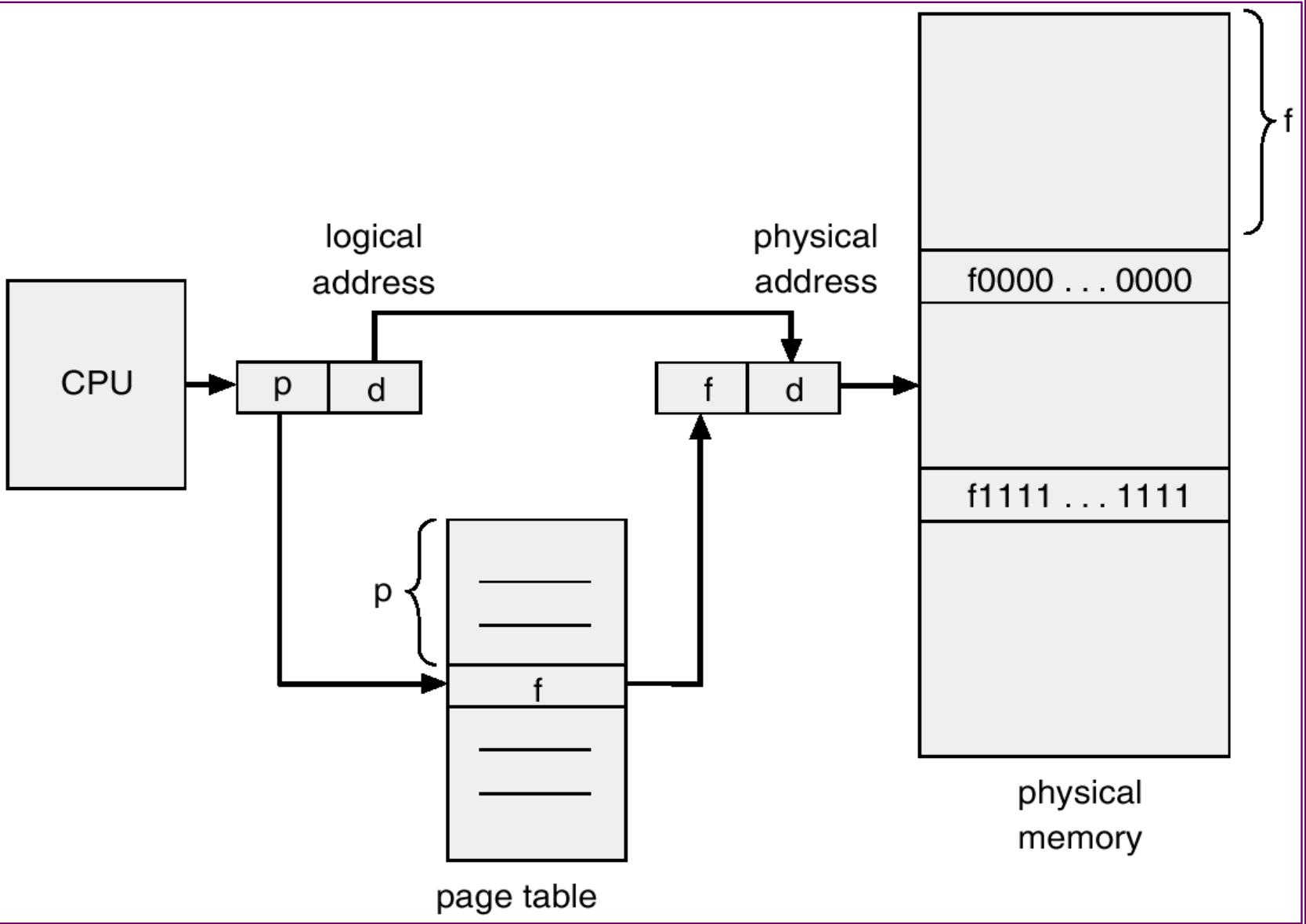
- Solution to external fragmentation
 - Permit the logical address space of the process to be non contiguous, allowing a process to be allocated physical memory whenever the latter is available.
 - Paging is a solution.
- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available.
 - Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8192 bytes).
 - Divide logical memory into blocks of same size called **pages**.
 - Keep track of all free frames.
- To run a program of size ' n ' pages, need to find ' n ' free frames and load program.
- Set up a page table to translate logical to physical addresses.
- Internal fragmentation.

Address Translation Scheme

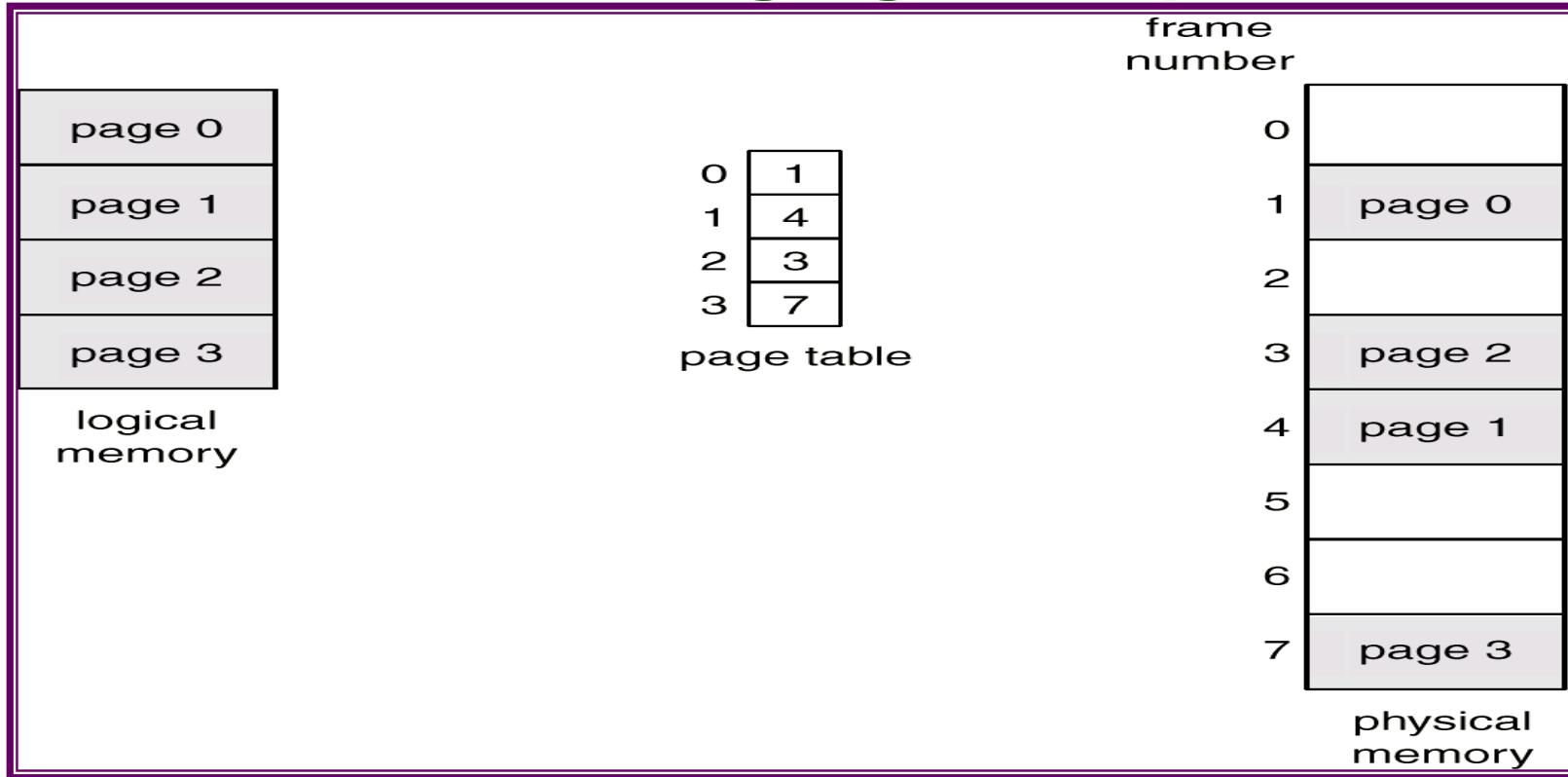
- Address generated by CPU is divided into:
 - *Page number (p)* – used as an index into a *page table* which contains base address of each page in physical memory.
 - *Page offset (d)* – combined with base address to define the physical memory address that is sent to the memory unit.
- Page number is an index to the page table.
- The page table contains base address of each page in physical memory.
- The base address is combined with the page offset to define the physical address that is sent to the memory unit.
- The size of a page is typically a power of 2.
 - 512 –8192 bytes per page.
- The size of logical address space is 2^m and page size is 2^n address units.
- Higher m-n bits designate the page number
- n lower order bits indicate the page offset.



Address Translation Architecture

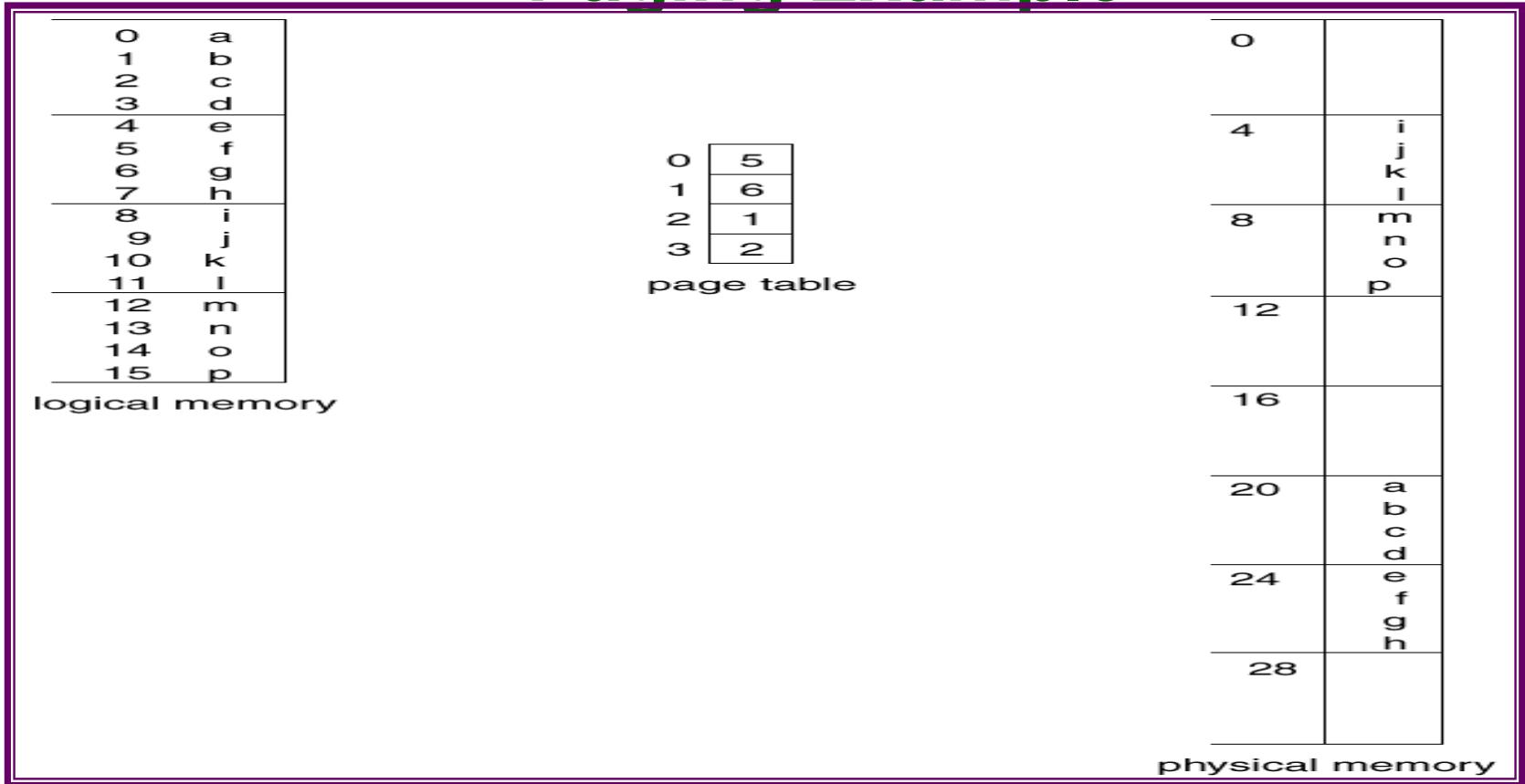


Paging Example



- Page size= 4 bytes; Physical memory=32 bytes (8 pages)
- Logical address “x” maps to
(frame number of the corresponding page*page size)+x
 - Logical address 0 maps $1*4+0=4$
 - Logical address 3 maps to= $1*4+3=7$
 - Logical address 4 maps to = $4*4+0=16$
 - Logical address 13 maps to= $7*4+1=29$.

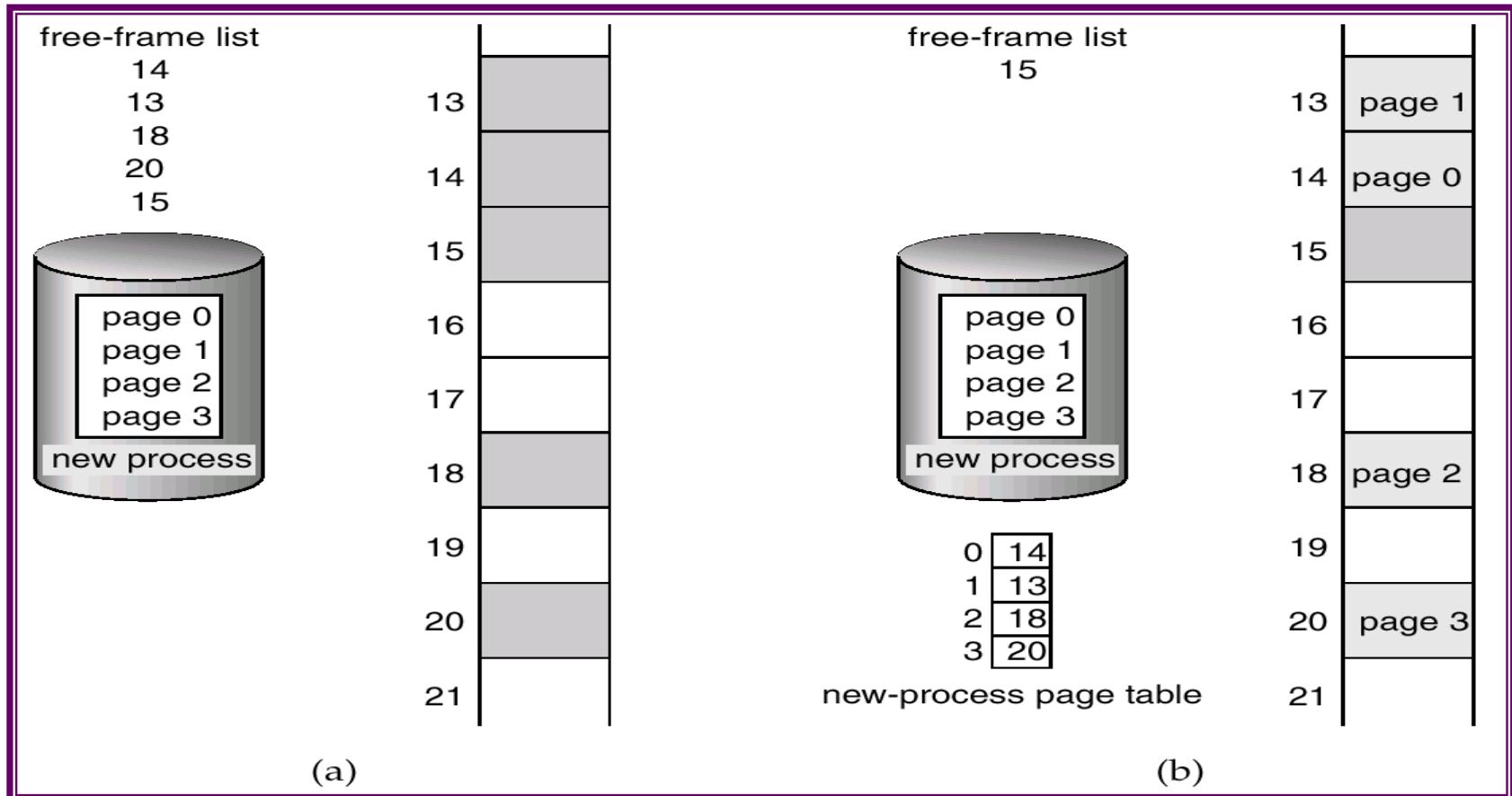
Paging Example



- Page size= 4 bytes; Physical memory=32 bytes (8 pages)
- Logical address “x” maps to
(frame number of the corresponding page*page size)+x
- Logical address 0 maps 5*4+0=20
- Logical address 3 maps to= 5*4+3=23
- Logical address 4 maps to =6*4+0=24
- Logical address 13 maps to= 2*4+1=9.

Free Frames

- When a process arrives the size in pages is examined
- Each page of process needs one frame.
- If n frames are available these are allocated, and page table is updated with frame number.



Before allocation

After allocation

More about Paging

- Paging separates user's view of memory and the actual physical memory.
- User program views memory as single contiguous space.
- In fact, user program is scattered throughout physical (main) memory.
- OS maintains the copy of the page table for each process. This copy is used to translate logical addresses to physical addresses.

- With paging we have no external fragmentation.
- We may have some internal fragmentation.
- In general, page sizes of 2K or 4K bytes.

Two issues

- Speed
- Space

Implementation of Page Table

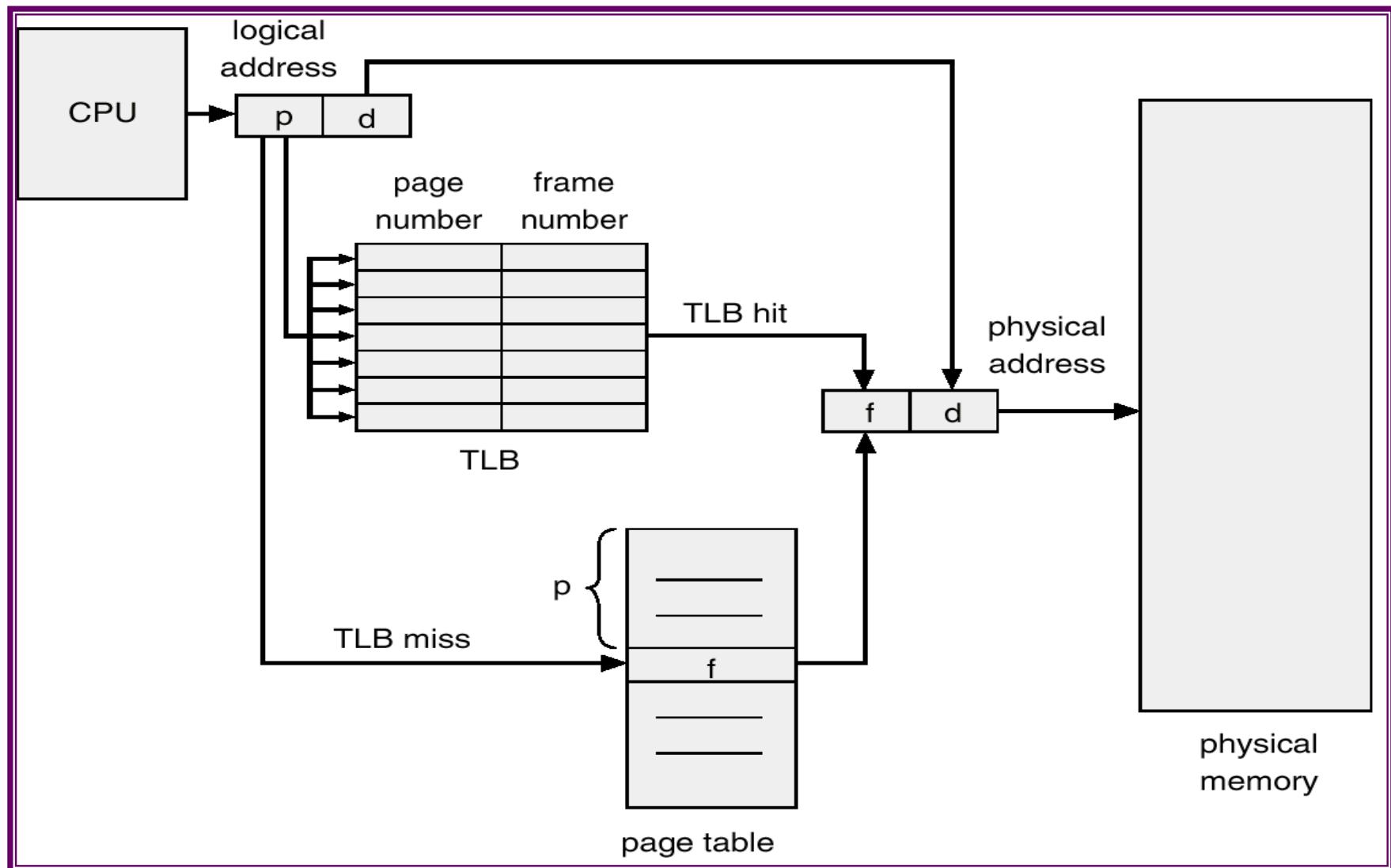
- Two options: Page table can be kept in registers or main memory
- Page table is kept in main memory due to bigger size.
 - Ex: address space = 2^{32} to 2^{64}
 - Page size= 2^{12}
 - Page table= $2^{32} / 2^{12} = 2^{20}$
 - If each entry consists of 4 bytes, the page table size = 4MB.
- *Page-table base register* (PTBR) points to the page table.
- *Page-table length register* (PRLR) indicates size of the page table.
- PTBR, and PRLR are maintained in the registers.
- Context switch means changing the contents of PTBR and PRLR.
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.
 - Memory access is slowed by a factor of 2.
 - **Swapping might be better !**
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called *associative memory* or *translation look-aside buffers (TLBs)*

Translation look-aside buffer (TLB) to address Speed issue

- TLB is an associative memory – parallel search
- A set of associative registers is built of especially high-speed memory.
- Each register consists of two parts: key and value
- When associative registers are presented with an item, it is compared with all keys simultaneously.
- If corresponding field is found, corresponding field is output.
- Associative registers contain only few of page table entries.
 - When a logical address is generated it is presented to a set of associative registers.
 - If yes, the page is returned.
 - Otherwise memory reference to page table mode. Then that page # is added to associative registers.
- Address translation (A' , A'')
 - If A' is in associative register, get frame # out.
 - Otherwise get frame # from page table in memory
- It may take 10 % longer than normal time.
- % of time the page # is found in the associative registers is called hit ratio.

| Page # | Frame # |
|--------|---------|
| | |
| | |
| | |
| | |

Paging Hardware With TLB



Address Space Identifiers (ASIDs)

- Some TLBs store ASIDs in each TLB entry.
- ASID uniquely identifies each process.
- ASID mechanism allows the TLB to contain entries for several different processes simultaneously.
- If TLB does not support separate ASIDs, every time new page table is selected, the TLB must be flushed (erased) to ensure that next executing process does not use wrong translation information.

Effective Access Time

- If it takes 20 nsec to search the associative registers and 100 nsec to access memory and hit ratio is 80 %, then,
 - Effective access time=hit ratio*Associate memory access time +miss ratio* memory access time.
 - $0.80 * 120 + 0.20 * 220 = 140$ nsec.
 - 40 % slowdown.
- For 98-percent hit ratio, we have
 - Effective access time= $98 * 120 + 0.02 * 220$
= 122 nanoseconds
= 22 % slowdown.

Memory Protection

- Memory protection implemented by associating protection bit with each frame.
- *One bit can be assigned to indicate read and write or read-only*
 - *An attempt to write read-only page causes hardware trap to OS.*
- *More bits can be added to provide read-only, read-write or execute-only protection.*
- *Valid-invalid* bit attached to each entry in the page table:
 - “valid” indicates that the associated page is in the process’ logical address space, and is thus a legal page.
 - “invalid” indicates that the page is not in the process’ logical address space.
 - Illegal addresses can be trapped with valid/invalid bit.
 - Some systems implement page table length register (PTLR) in case of internal fragmentation.
 - PTLR is used to check whether address is in valid range or not.

Valid (v) or Invalid (i) Bit In A Page Table

00000

| |
|--------|
| page 0 |
| page 1 |
| page 2 |
| page 3 |
| page 4 |
| page 5 |

10,468

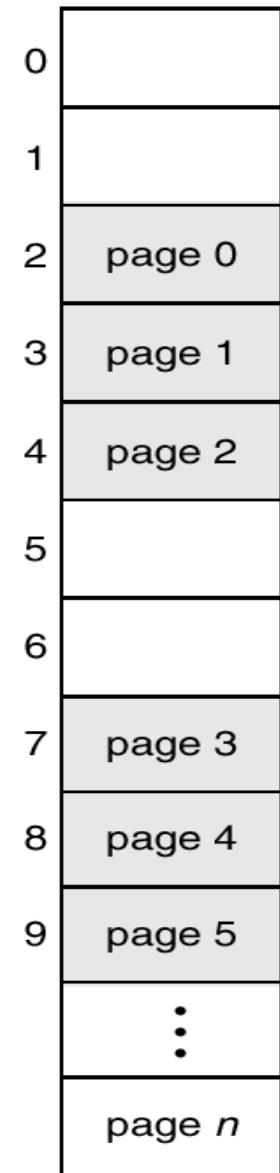
12,287

frame number

valid–invalid bit

| | | |
|---|---|---|
| 0 | 2 | v |
| 1 | 3 | v |
| 2 | 4 | v |
| 3 | 7 | v |
| 4 | 8 | v |
| 5 | 9 | v |
| 6 | 0 | i |
| 7 | 0 | i |

page table



Page Table: Space Issue

- Page table:
 - Each process has a page table associated with it.
 - The page table has a slot for each logical address regardless of its validity.
 - Since table is sorted by virtual address, OS calculates the value directly.
- **However, it consumes more space.**
- Example regarding space issue: Modern computer systems support a very large address space: 2^{32} to 2^{64} .
 - Consider a system with 32-bit logical address space. If the page size is 4K, then the page table size is 2^{12} entries.
 - If each entry consists of 4 bytes, then each process may need 4 mega bytes for a page table.

Page Table Structure

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

- Modern computer systems support a very large address space: 2^{32} to 2^{64} .
 - Consider a system with 32-bit logical address space. If the page size is 4K, then the page table size is 2^{12} entries.
 - If each entry consists of 4 bytes, then each process may need 4 mega bytes.
- Solution: Break up the logical address space into multiple page tables.
- A simple technique is a two-level page table.
 - Page table itself is paged.

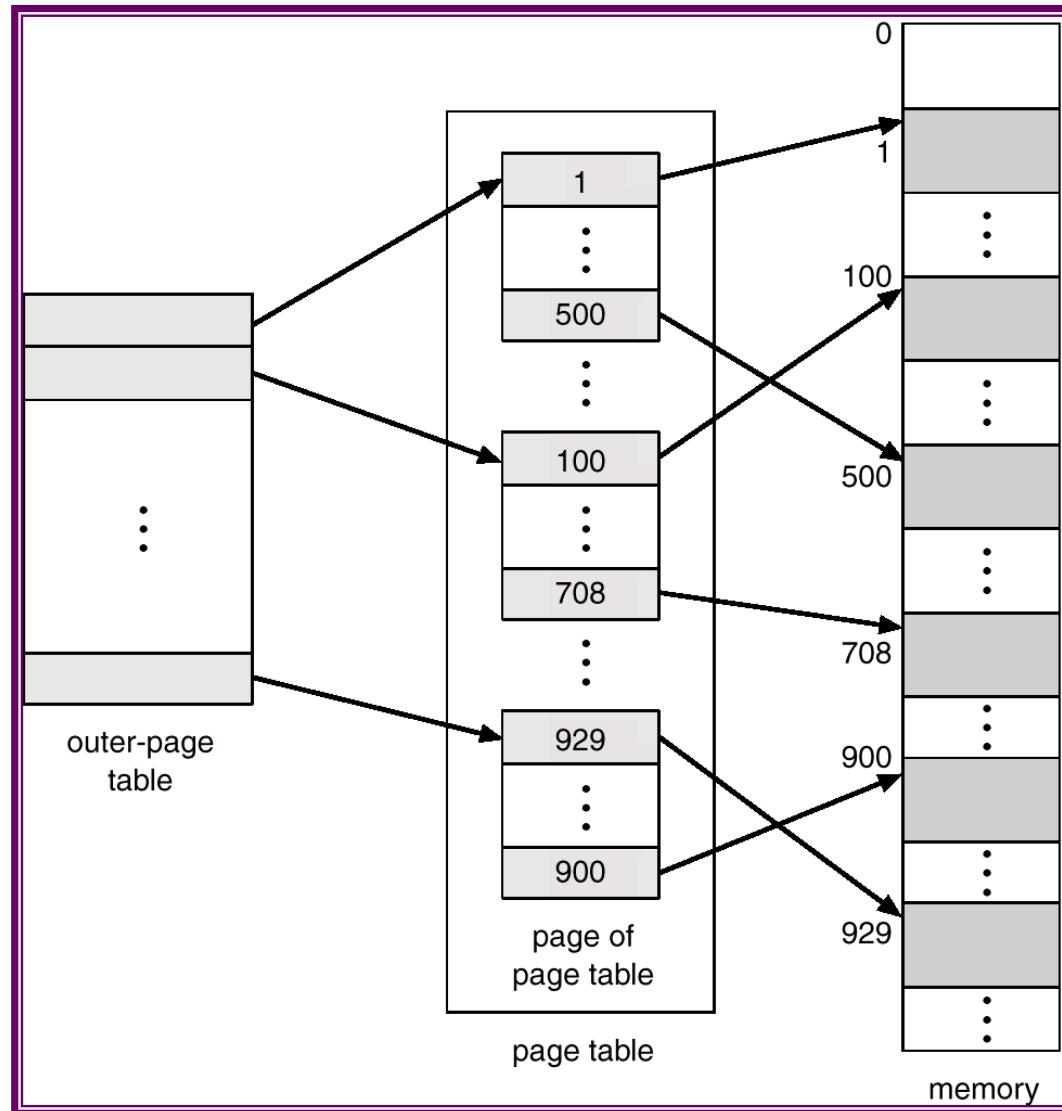
Two-Level Paging Example

- A logical address (on 32-bit machine with 4K page size) is divided into:
 - a page number consisting of 20 bits.
 - a page offset consisting of 12 bits.
- Since the page table is paged, the page number is further divided into:
 - a 10-bit page number.
 - a 10-bit page offset.
- Thus, a logical address is as follows:

| page number | | page offset |
|-------------|-------|-------------|
| p_i | p_2 | d |
| 10 | 10 | 12 |

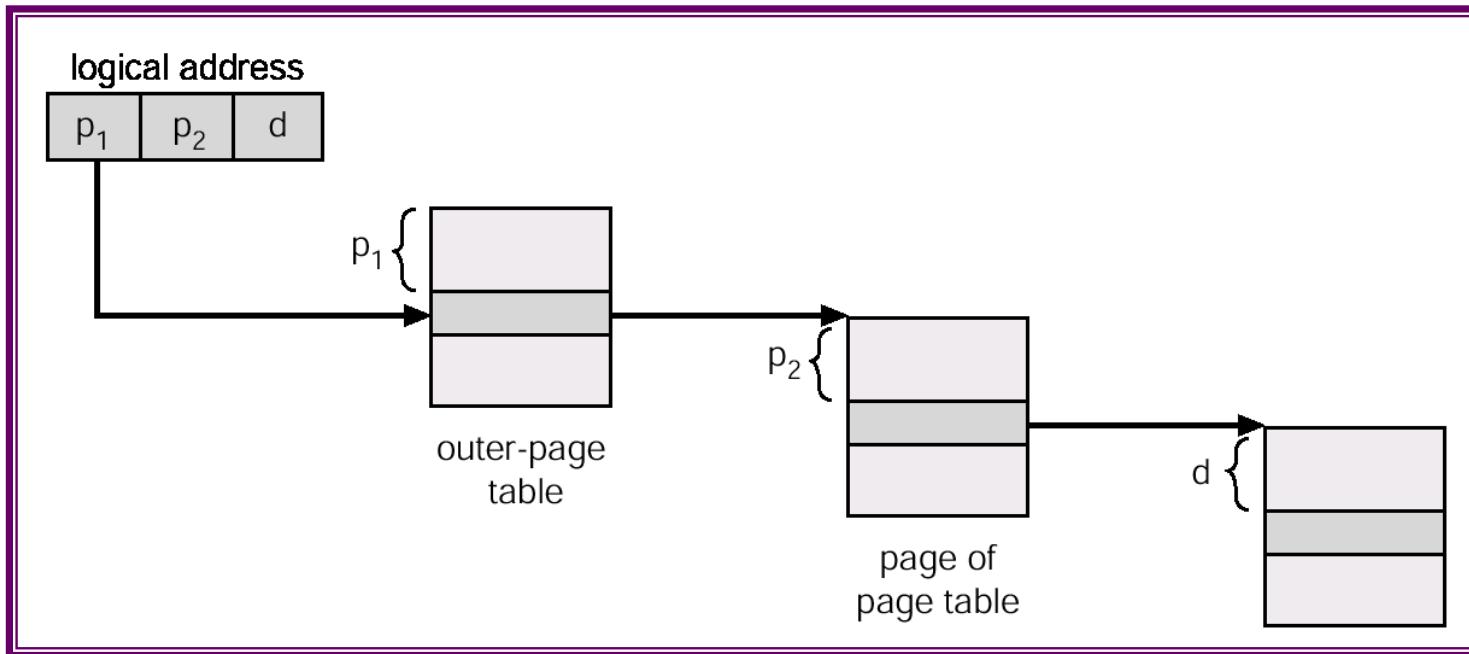
where p_i is an index into the outer page table, and p_2 is the displacement within the page of the outer page table.

Two-Level Page-Table Scheme



Address-Translation Scheme

- Address-translation scheme for a two-level 32-bit paging architecture



Address-Translation Scheme

- VAX 32-bit architecture supports a variation of two-level paging scheme
- SPARC 32 bit architecture supports 3-level paging scheme.
- 32-bit Motorola 68030 supports a 4-level paging scheme.
- Tradeoff: size versus speed
 - For 64-bit architectures hierarchical page tables are inappropriate.
 - For example, the 64-bit UltraSPARC would require seven levels of paging.

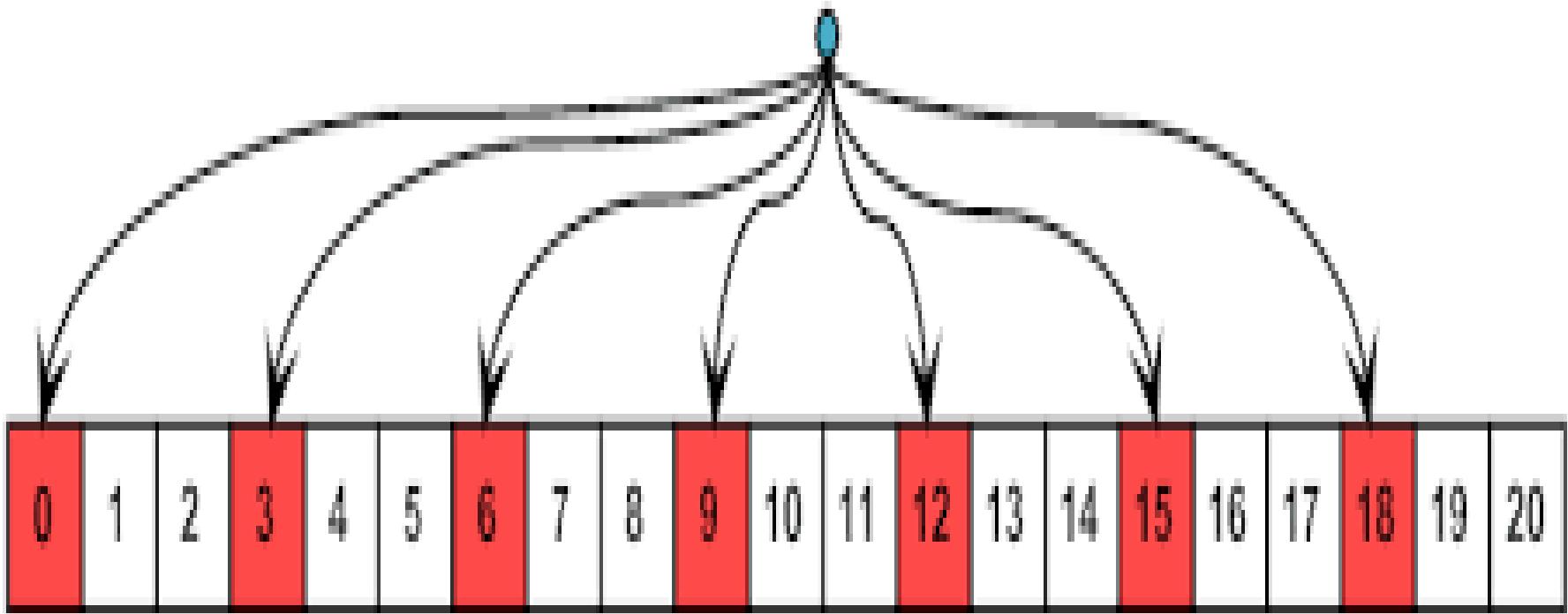
Hashed Page Tables

- Common in address spaces > 32 bits.
- The virtual page number is hashed into a page table. This page table contains a chain of elements hashing to the same location.
- Virtual page numbers are compared in this chain searching for a match. If a match is found, the corresponding physical frame is extracted.

About Hash Tables

- A hash function takes a search key as an argument and computes an integer range 0 to $B-1$, where B is a number of buckets.
 - A bucket array, which is an array indexed from 0 to $B-1$, holds headers of B linked lists, one for each bucket of the array.
 - If the record has a search key K , we store the record by linking it to the bucket list for the bucket numbered $h(K)$, where h is the hash function.
- Common hash function
 - Remainder of K/B

$$\{30, 33, 36, 39, 42, 45, \dots, 111, 114, 117\}$$
$$H_i = h(k_i) \bmod 21$$



Secondary Storage Hash Tables

- Bucket contains sequence of blocks
-
- First block can be found given “i”.
- If a bucket overflows, a chain of overflow blocks are added.

Bucket

- Bucket is a logical concept
 - □ It is the concept of hashing
 - □ Physically implemented as a sequence of blocks
 - □ Ideal= 1 block

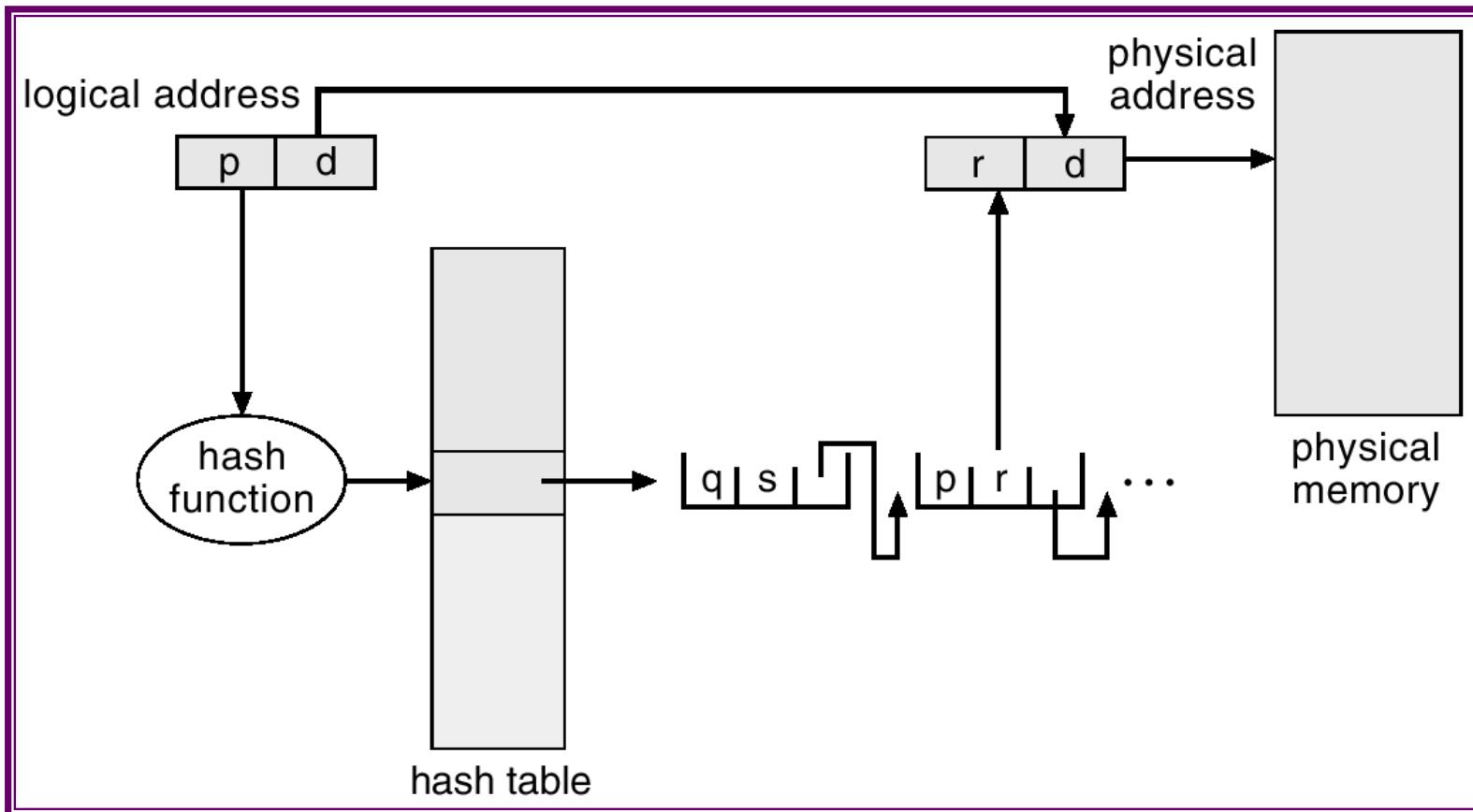
Insertion into a Hash Table

- When a new record with search key K must be inserted, we compute $h(K)$.
- If the bucket number $h(K)$ has the space, we insert the record into the block for this bucket or into one of the chain of blocks.
- If there is no space, we add extra block.

Hash-table deletion

- Go to the bucket number $h(K)$ and search for records with that search key. Delete if we find any data.
 - Consolidate (optional)

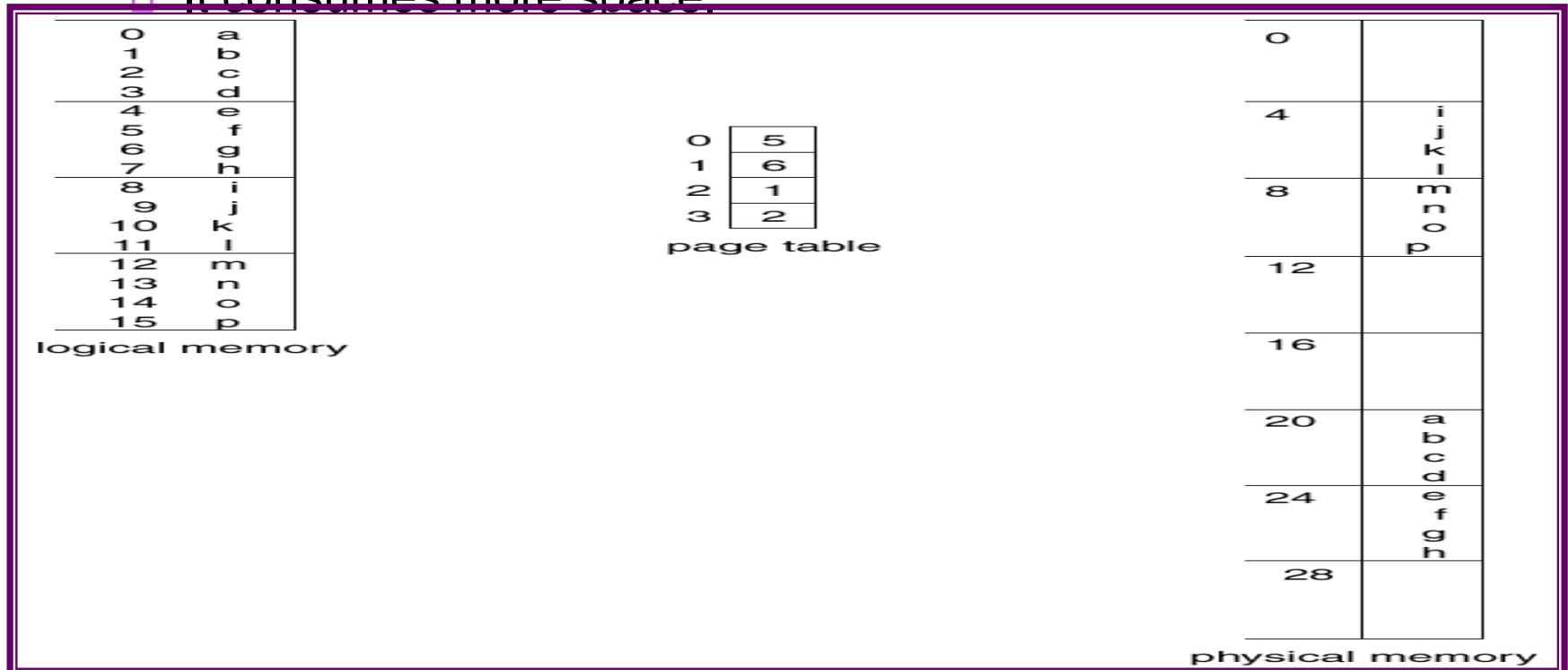
Hashed Page Table



Inverted Page Table

□ Page table:

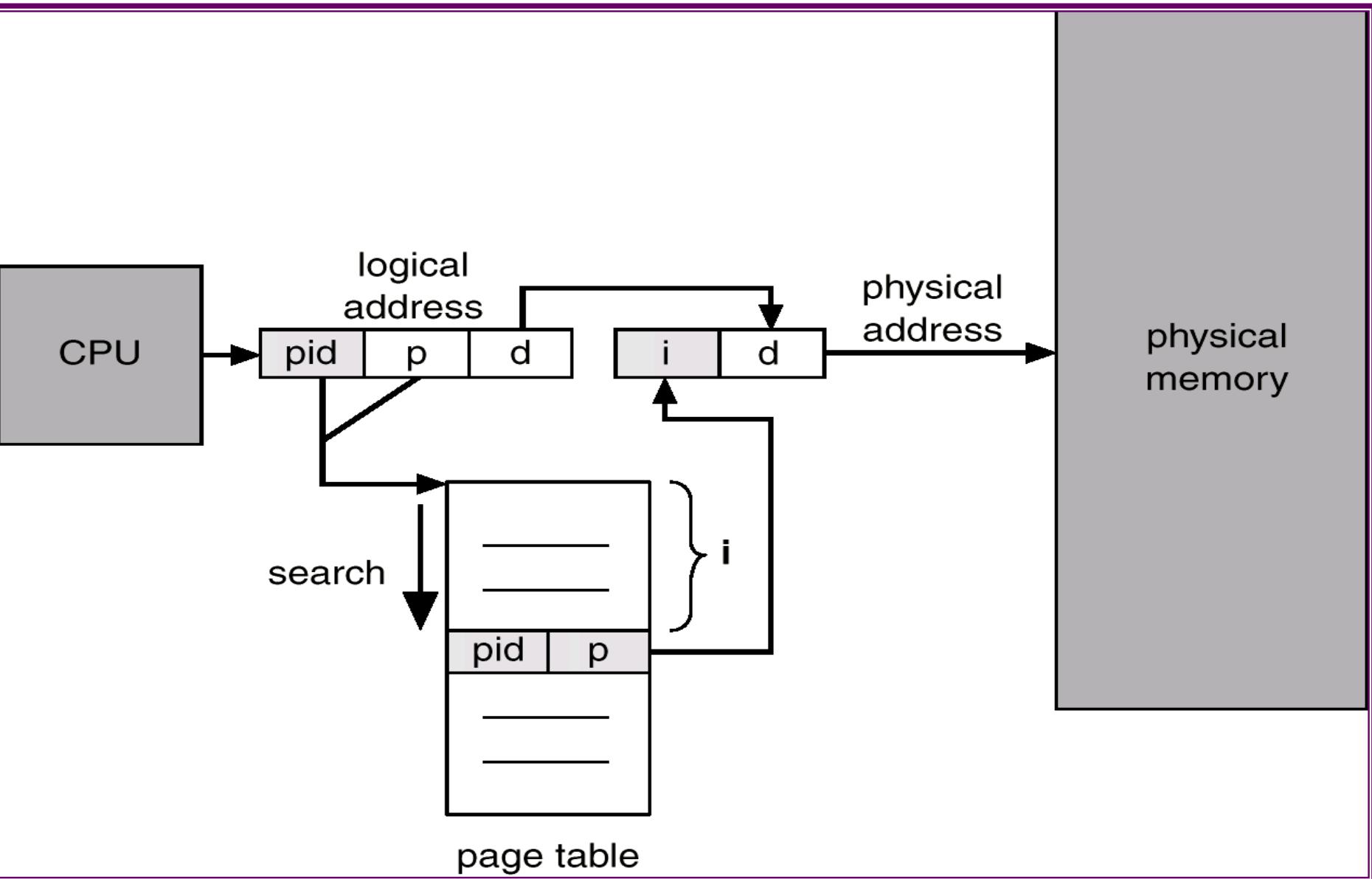
- Each process has a page table associated with it.
- The page table has a slot for each logical address regardless of its validity.
- Since table is sorted by virtual address, OS calculates the value directly.
- It consumes more space.



Inverted Page Table

- Solution: Inverted page table
- One entry for each real page of memory.
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.
- There is only one page table in the system.
- Each virtual address space in the the system consists of a triple <Process-id, page #, offset>
- When a memory reference occurs part of <process-id, page#> is presented to memory subsystem.
- The inverted page table is searched for a match.
- Use hash table to limit the search to one — or at most a few — page-table entries.
- It is implemented in 64-bit UltraSPARC and PowerPC.
- Decreases amount of memory but increases time needed to search the table.

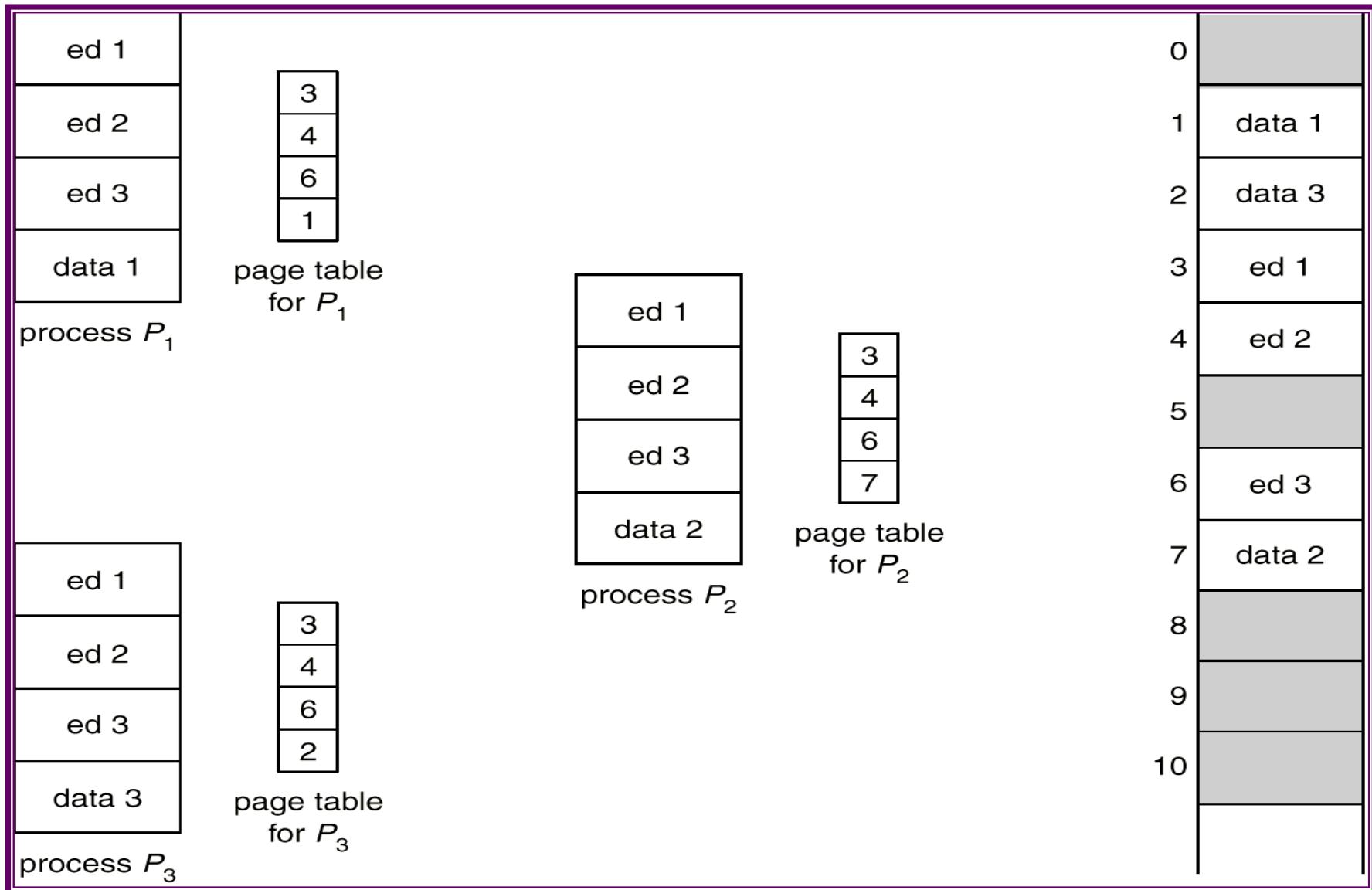
Inverted Page Table Architecture



Shared Pages

- Advantage of paging
 - Paging allows sharing of common code.
- Shared code
 - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
 - Shared code must appear in the same location in the logical address space of all processes.
- Private code and data
 - Each process keeps a separate copy of the code and data.
 - The pages for the private code and data can appear anywhere in the logical address space.
- For example if a system supports text editor and supports 40 users. If the text editor consists of 150K and 50K of data space we need 8000K for the 40 users.
- If the code reentrant it can be shared.
 - Reentrant code is non-self modifying code
 - It never changes during execution
 - Ex: compilers, window system, DBS and so on can be shared.

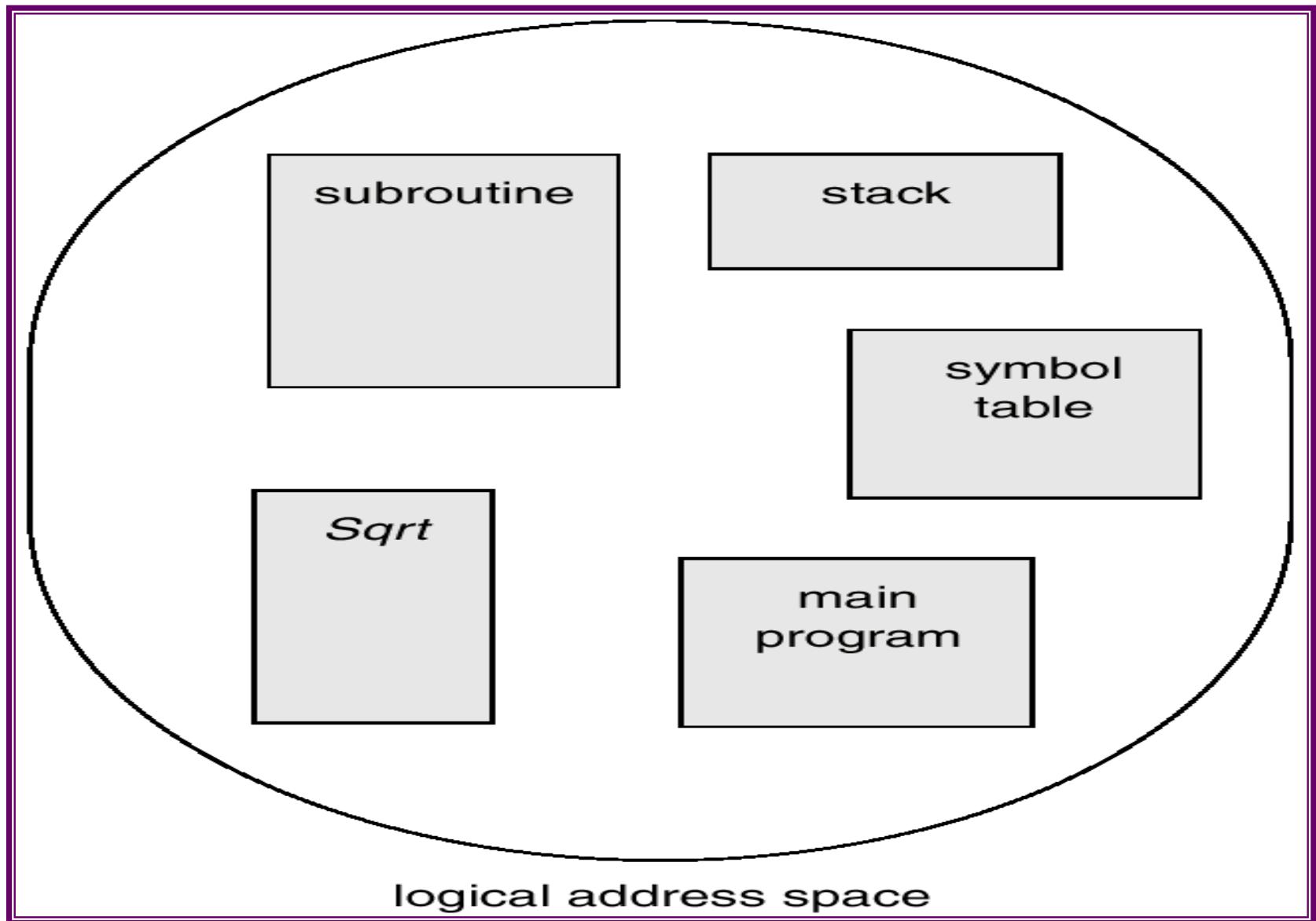
Shared Pages Example



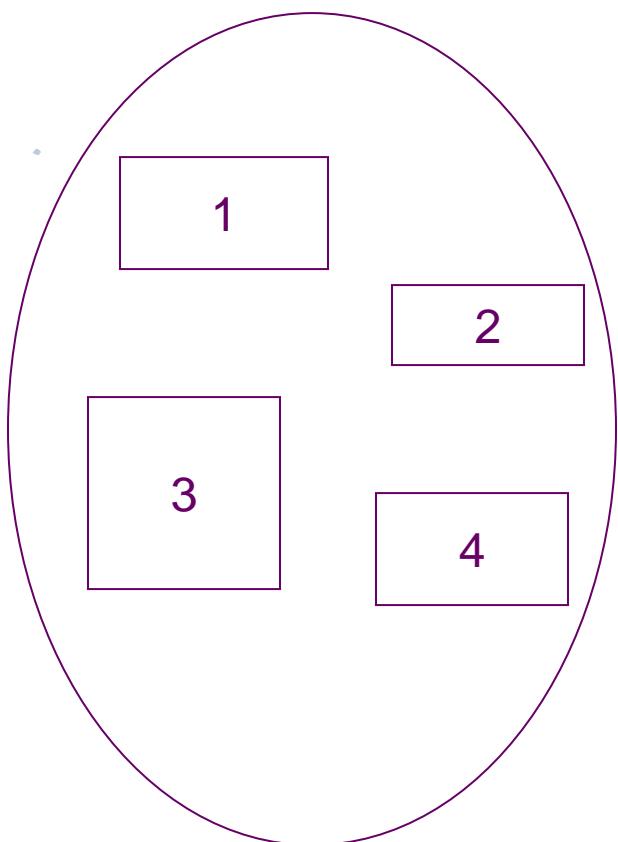
Segmentation

- Paging issues
 - Space: Size of page table
 - Searching time
- Segmentation is a memory-management scheme that supports user view of memory.
- Users prefer to view memory as a collection of variable-sized segments without any ordering.
- Logical address space is a collection of segments.
 - Each segment has name and length.
 - The address specify segment name and length.
- A program is a collection of segments. A segment is a logical unit such as:
 - main program,
 - procedure,
 - function,
 - method,
 - object,
 - local variables, global variables,
 - common block,
 - stack,
 - symbol table, arrays

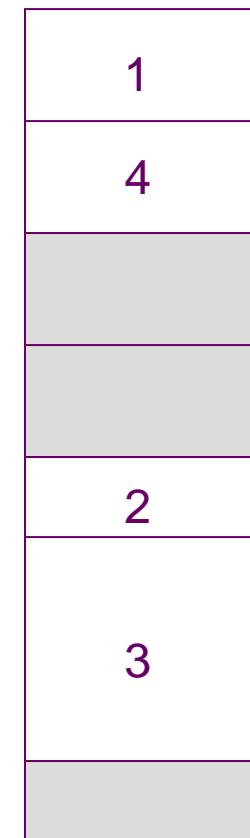
User's View of a Program



Logical View of Segmentation



user space



physical memory space

Segmentation Architecture

- Logical address consists of a two tuple:
 - $\langle \text{segment-number}, \text{offset} \rangle$,
- *Segment table* – maps two-dimensional physical addresses; each table entry has:
 - base – contains the starting physical address where the segments reside in memory.
 - *limit* – specifies the length of the segment.
- *Segment-table base register (STBR)* points to the segment table's location in memory.
- *Segment-table length register (STLR)* indicates number of segments used by a program;
 - segment number s is legal if $s < \text{STLR}$.

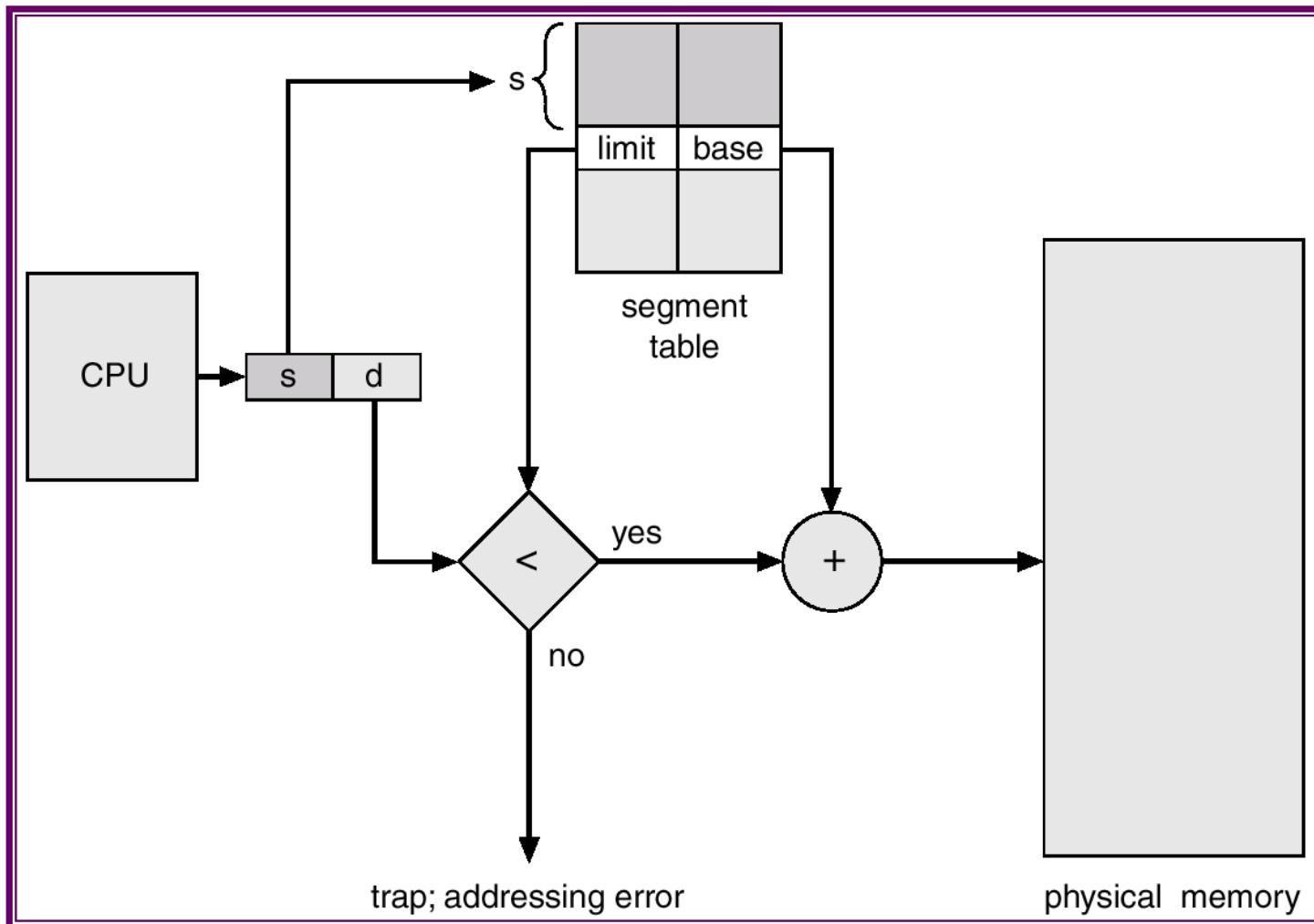
Segmentation Architecture (Cont.)

- Relocation.
 - □ dynamic
 - □ by segment table
- Sharing.
 - shared segments
 - same segment number
- Allocation.
 - first fit/best fit
 - external fragmentation

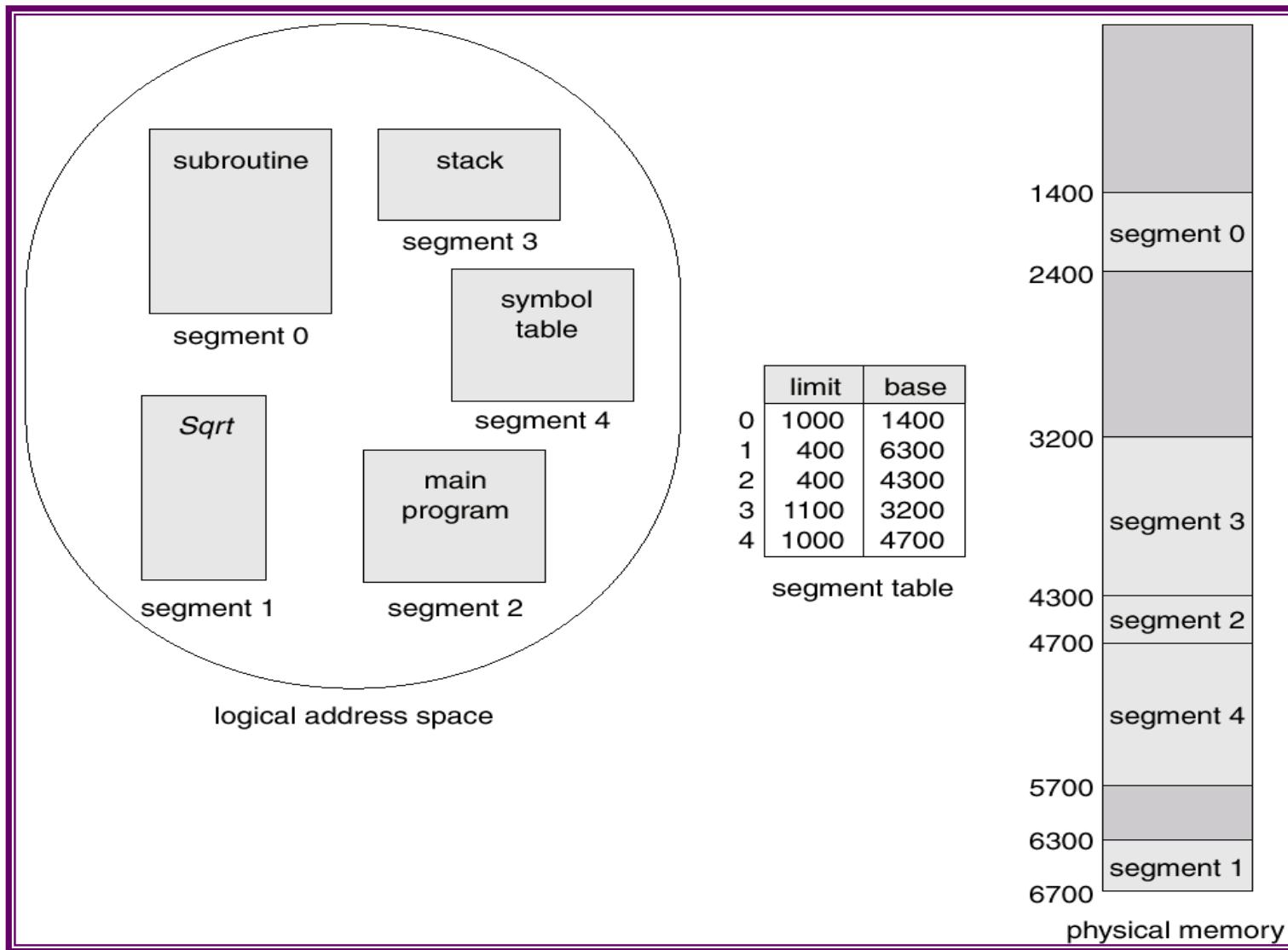
Segmentation Architecture (Cont.)

- Protection. With each entry in segment table associate:
 - validation bit = 0 \Rightarrow illegal segment
 - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level.
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem.
- A segmentation example is shown in the following diagram

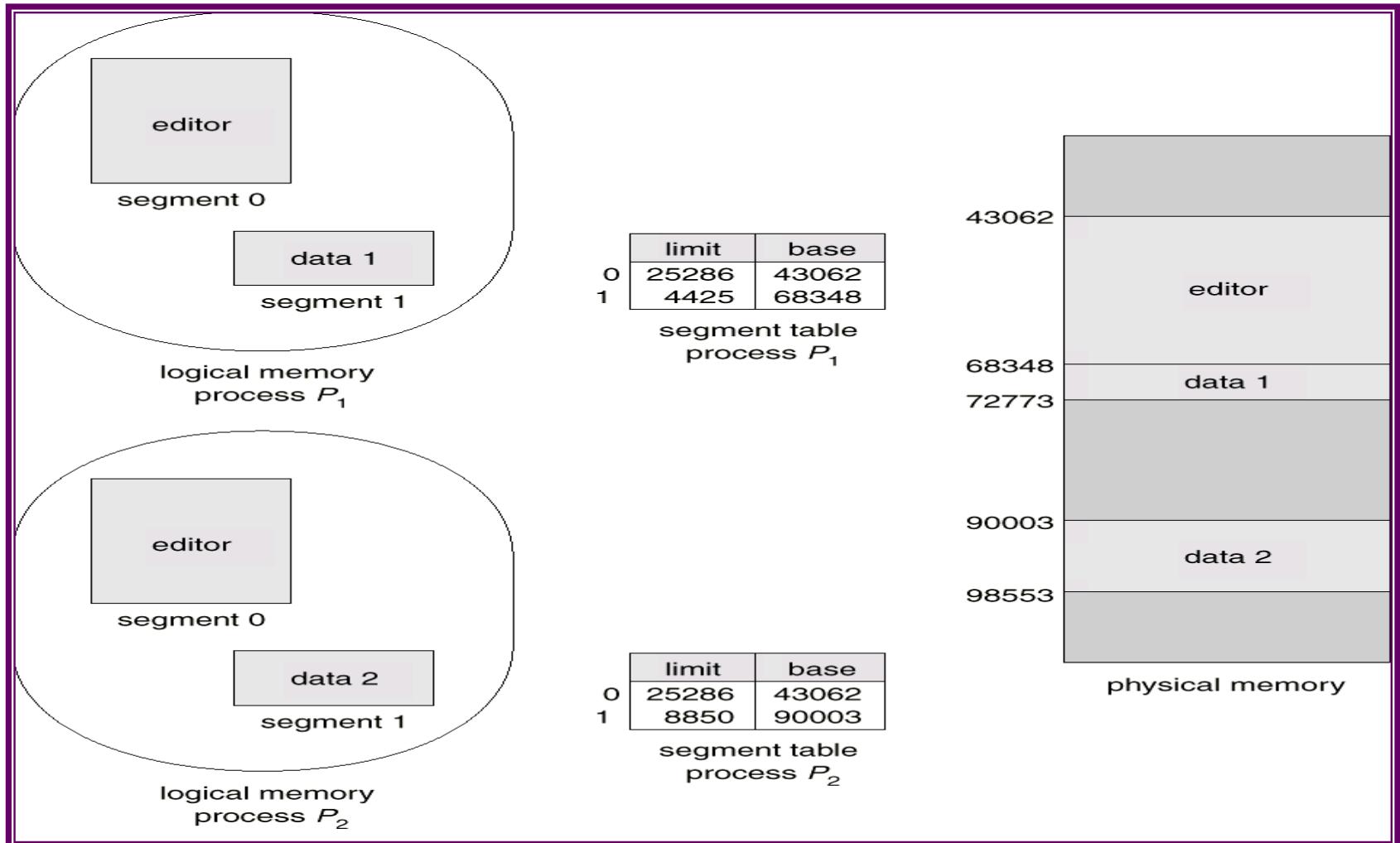
Segmentation Hardware



Example of Segmentation



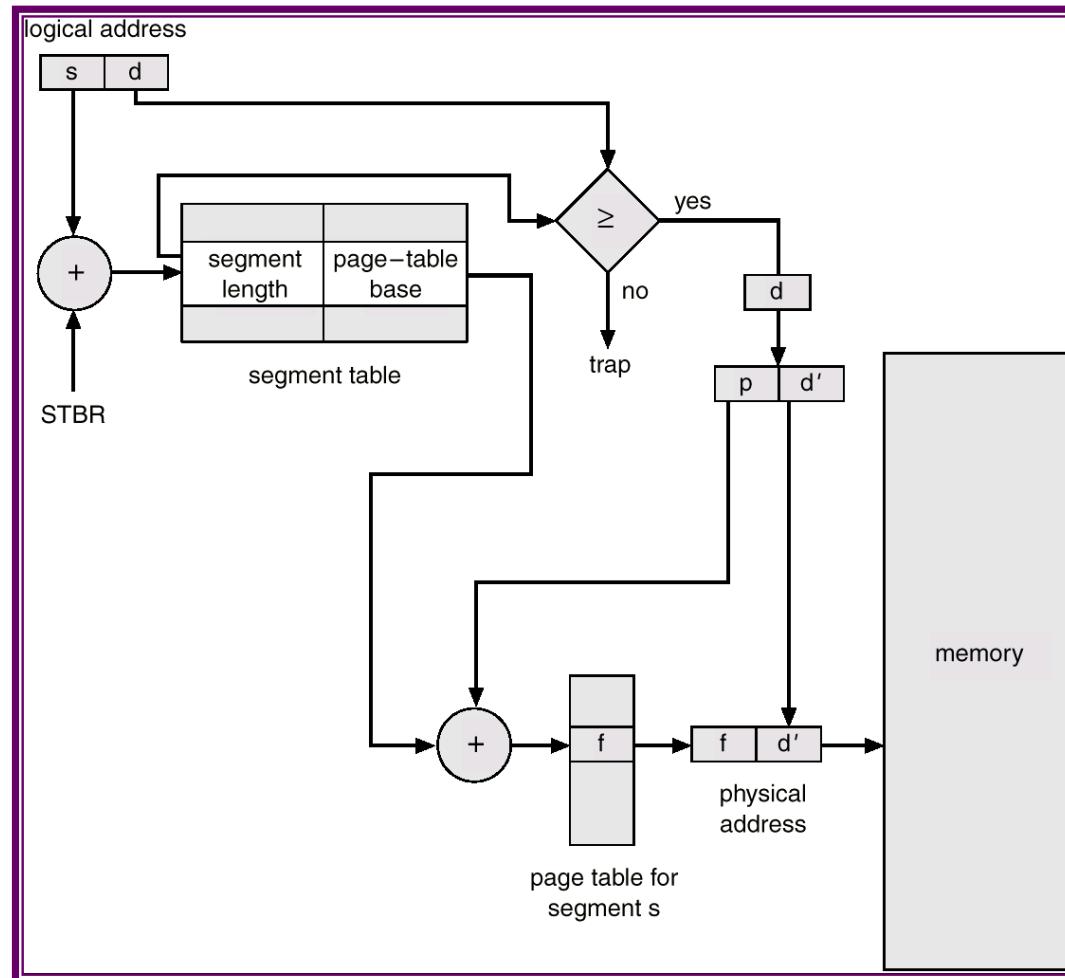
Sharing of Segments



Segmentation with Paging – MULTICS

- The MULTICS system solved problems of external fragmentation and lengthy search times by paging the segments.
- Solution differs from pure segmentation in that the segment-table entry contains not the base address of the segment, but rather the base address of a *page table* for this segment.

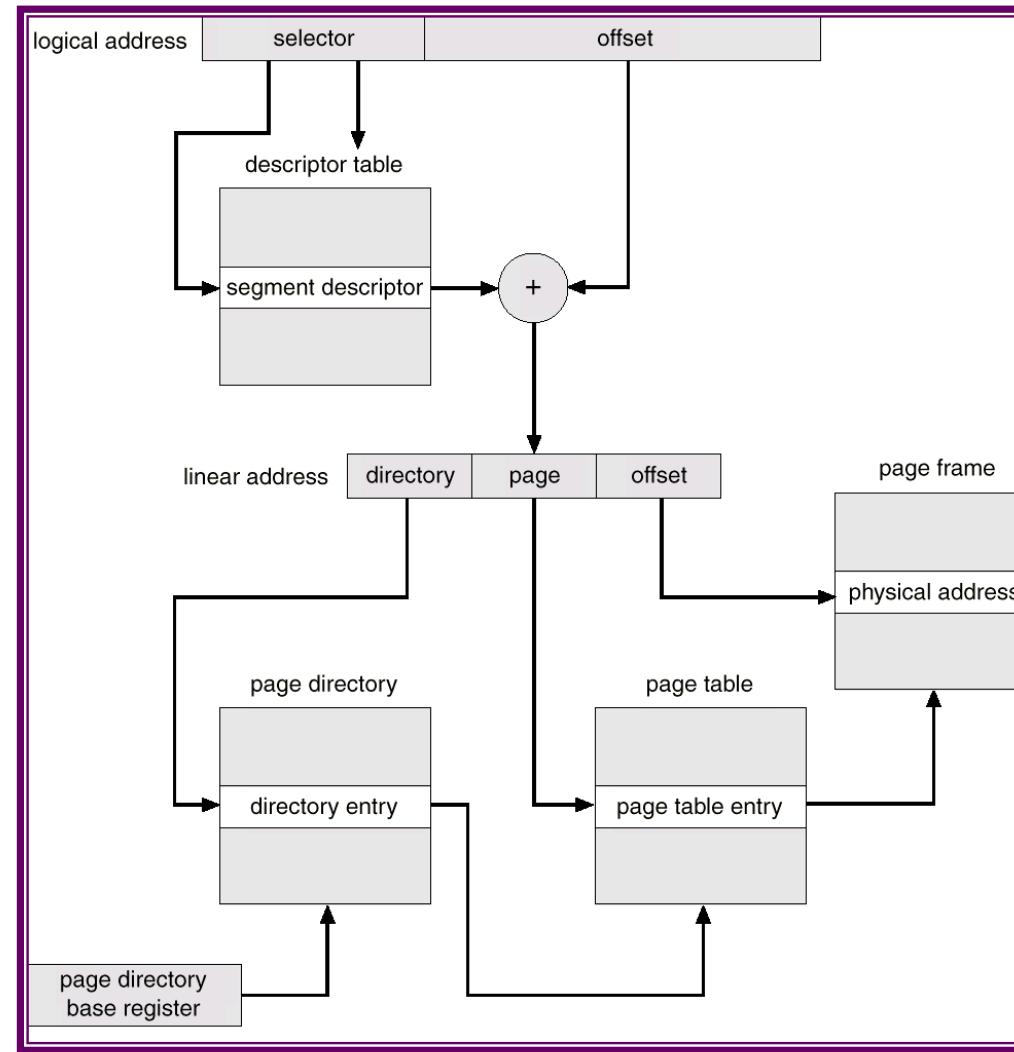
MULTICS Address Translation Scheme



Segmentation with Paging – Intel 386

- As shown in the following diagram, the Intel 386 uses segmentation with paging for memory management with a two-level paging scheme.

Intel 30386 Address Translation



Chapter: Virtual Memory

- Background
- Demand Paging
- Process Creation
- Page Replacement
- Allocation of Frames
- Thrashing
- Operating System Examples

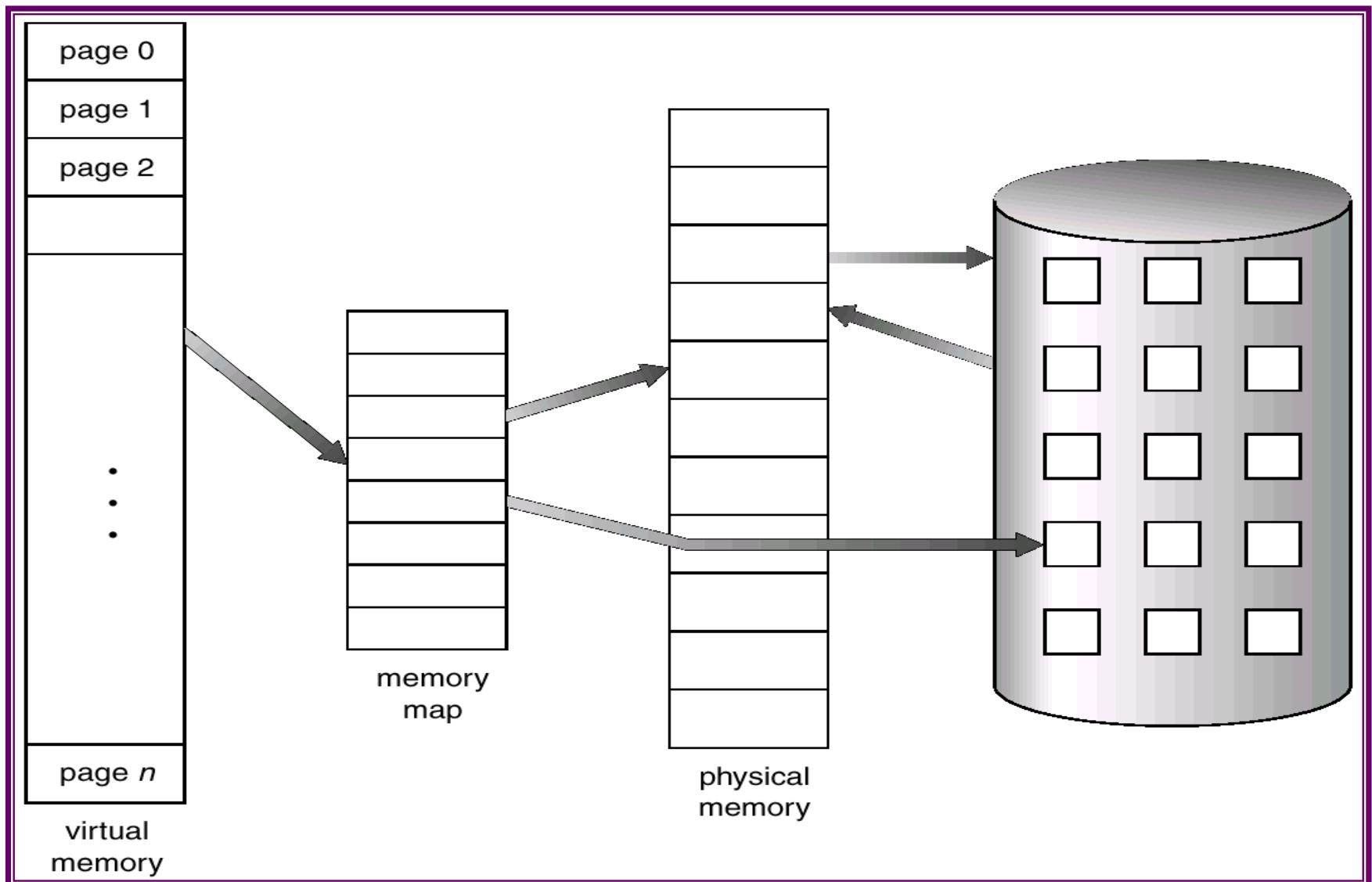
Background

- First requirement for execution: Instructions must be in physical memory
 - **One Approach:** Place entire logical address in main memory.
 - Overlays and dynamic loading may relax this criteria.
 - But the size of **the program is limited to size of main memory.**
- Normally entire program may not be needed in main memory.
 - Programs have error conditions.
 - Arrays, lists, and tables may be declared by 100 by 100 elements, but seldom larger than 10 by 10 elements.
 - Assembler program may have room for 3000 symbols, although average program may contain less than 200 symbols.
 - Certain portions or features of the program are used rarely.
- Benefits of the ability to execute program that is partially in memory:
 - User can write programs and software for entirely large virtual address space.
 - More programs can run at the same time.
 - Less I/O would be needed to load or swap each user program into memory.

Background

- Virtual memory is a technique that allows the execution of processes that may not be completely in memory.
 - Programs are larger than main memory.
 - VM abstract main memory into an extremely large, uniform array of storage.
- Separation of user logical memory from physical memory.
 - Only part of the program needs to be in memory for execution.
 - Logical address space can therefore be much larger than physical address space.
 - Allows address spaces to be shared by several processes.
 - Allows for more efficient process creation.
 - Frees the programmer from memory constraints.
- Virtual memory can be implemented via:
 - Demand paging
 - Demand segmentation
- We only cover demand paging.
- For demand segmentation refer research papers.
 - IBM OS/2, Burroughs' computer systems

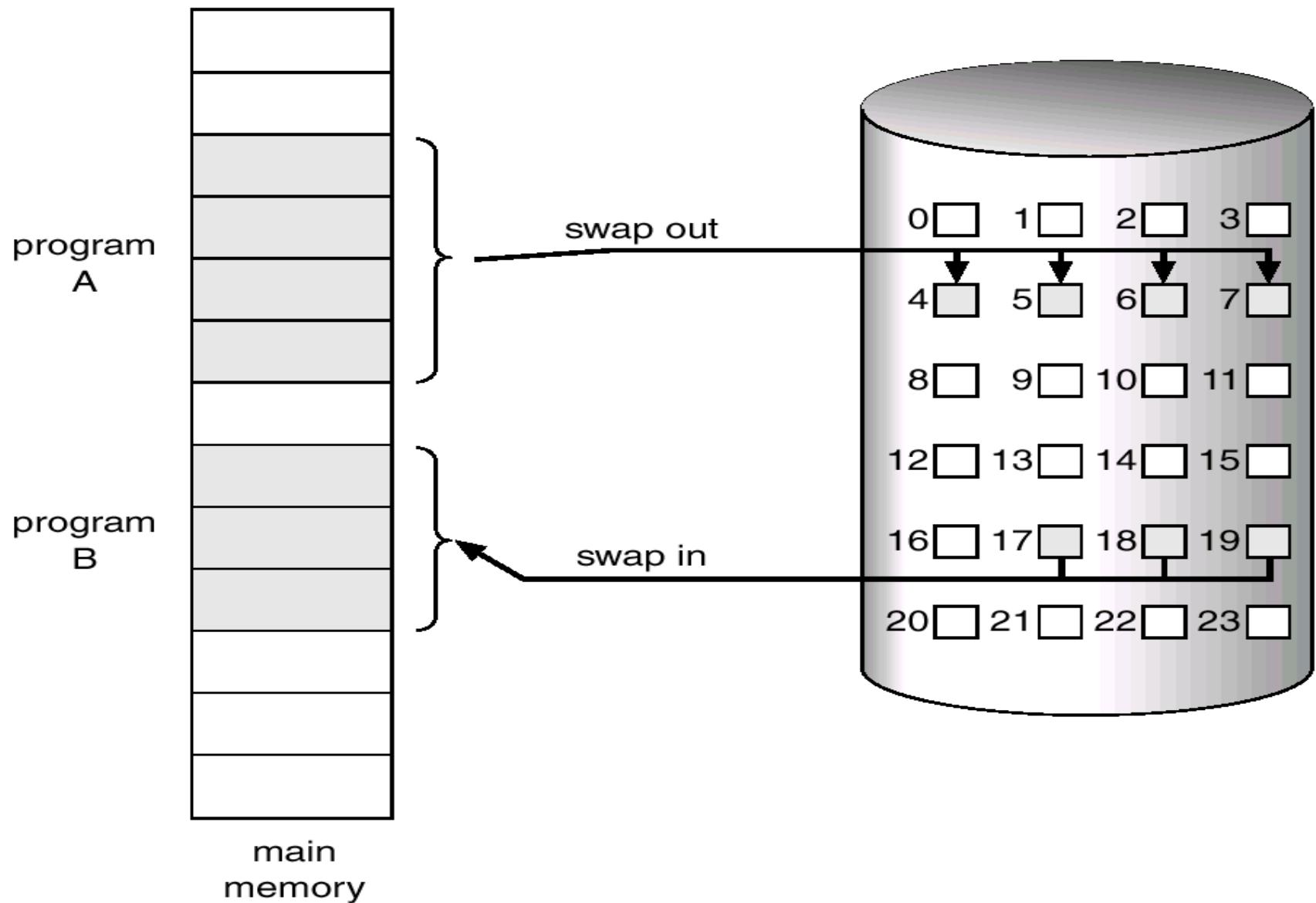
Virtual Memory That is Larger Than Physical Memory



Demand Paging

- Paging system with swapping.
 - When we execute a process we swap into memory (next fig).
- For demand paging, we use lazy swapper or pager.
 - Never swaps a page into memory unless required.
 - Bring a page into memory only when it is needed.
 - Less I/O needed
 - Less memory needed
 - Faster response
 - More users
- Page is needed \Rightarrow reference to it
 - invalid reference \Rightarrow abort
 - not-in-memory \Rightarrow bring to memory

Transfer of a Paged Memory to Contiguous Disk Space



Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated ($1 \Rightarrow$ in-memory, $0 \Rightarrow$ not-in-memory)
- Initially valid–invalid but is set to 0 on all entries.
- Example of a page table snapshot.

| Frame # | valid-invalid bit |
|---------|-------------------|
| | 1 |
| | 1 |
| | 1 |
| | 1 |
| | 0 |
| | |
| | 0 |
| | 0 |

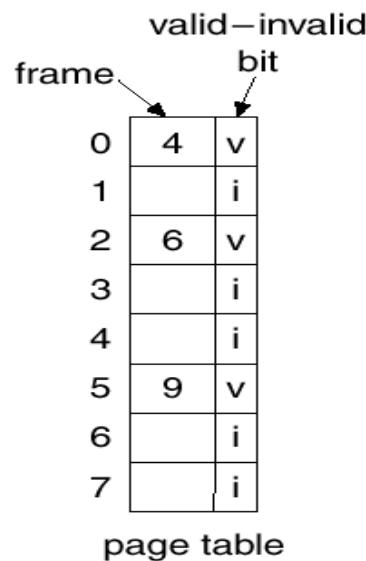
page table

- During address translation, if valid–invalid bit in page table entry is $0 \Rightarrow$ page fault.

Page Table When Some Pages Are Not in Main Memory

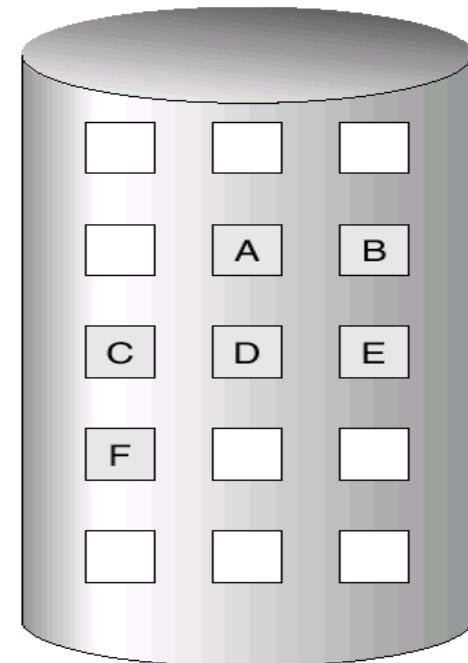
| | |
|---|---|
| 0 | A |
| 1 | B |
| 2 | C |
| 3 | D |
| 4 | E |
| 5 | F |
| 6 | G |
| 7 | H |

logical memory



| | |
|----|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | A |
| 5 | |
| 6 | C |
| 7 | |
| 8 | |
| 9 | F |
| 10 | |
| 11 | |
| 12 | |
| 13 | |
| 14 | |
| 15 | |

physical memory

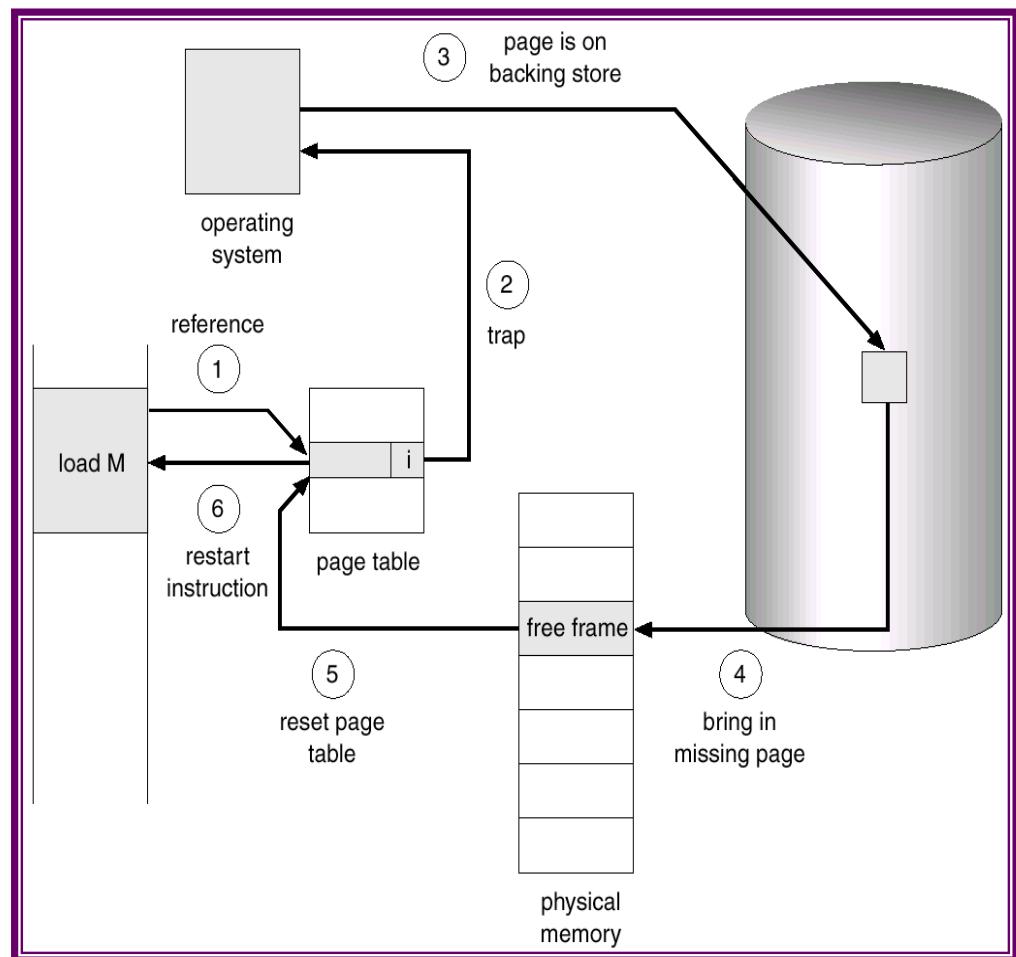


Page Fault

- If there is ever a reference to a page, first reference will trap to OS \Rightarrow page fault
- OS looks at another table to decide:
 - Invalid reference
 - \Rightarrow abort.
 - Just not in memory.
 - Get empty frame.
 - Swap page into frame.
 - Reset tables, validation bit = 1.
 - Restart instruction:

Steps in Handling a Page Fault

1. Check the internal table, to determine whether this reference is valid or invalid.
2. If the reference is invalid, then terminate.
3. Find free frame.
4. Schedule disk operation.
5. Modify the internal table to indicate that page is in main memory.
6. Restart the instruction.



What happens if there is no free frame?

- Page replacement – find some page in memory, but not really in use, swap it out.
 - algorithm
 - performance – want an algorithm which will result in minimum number of page faults.
- Same page may be brought into memory several times.

Hardware support to demand paging

- Theoretically some programs access several pages of new memory causing multiple page faults per instruction.
 - But analysis of program show locality of reference.
- Hardware support to demand paging
 - Page table
 - Secondary memory; high speed disk

Software support to demand paging

- Additional software is also required.
 - Restarting of instruction after page fault.
 - Page fault could occur any time during execution.
 - Ex: Add the contents of A and B and replace the result in C
 - 1.Fetch and decode the instruction
 - Fetch A
 - Fetch B
 - Add A and B
 - Store the sum in C.
 - If the page is faulted if we try to store C, we have to restart the instruction.

Software support to demand paging...

- Difficulty occurs if the instruction modifies several different locations.
- In IBM 360/370 MVC (Move character) instruction, we can move 256 bytes from one location to another.
 - source and destination may overlap
- If the page fault occurs after partial moving, we can not redo the instruction, if regions overlap.
- Solution:
 - 1. Use micro code to access both ends of blocks
 - If page fault is going to occur, it will occur.
 - 2. Use temporary registers to hold the values of temporary registers
 - If a page fault occurs old values are written back to memory, restoring the memory state to before the instruction was started.

Hardware support and software support to demand paging...

- Also, similar difficulty occurs in machines that use special addressing modes. Uses register as a pointer
 - Auto-increment: increment after using
 - auto-decrement. Decrements before using
 - MOV (R2)+, -(R3)
 - If the page fault occurs while storing in R3, we have to restart the instruction by restoring the values of R2 and R3.
 - if the instruction modifies several different locations.
- Solution: use status register to record the register number and amount modified so that OS can undo the effect of partially executed instruction that causes a page fault.
- **EVERY THING SHOULD BE TRANSPARENT TO USER**

Performance of Demand Paging

- Let p be the probability of page fault.
- Page Fault Rate $0 \leq p \leq 1.0$
 - if $p = 0$ no page faults
 - effective access time=memory access time.
 - if $p = 1$, every reference is a fault
- Effective Access Time (EAT)
$$\text{EAT} = (1 - p) \times \text{memory access} + p \text{ (page fault overhead)}$$
- Major operations during page fault:
 - Trap to OS; save user registers and process state; issue a disk read; wait for interrupt from disk; wait for the CPU; restore the process status;

Demand Paging Example

- Memory access time = 100 nanoseconds
- Page fault service time = 25 milliseconds
- Effective access time (EAT) = $(1 - p) \times 100 + p$
(25 msec)
 - $= 100 + 24,999,900 \times p$
- EAT is directly proportional to page-fault rate.
- It is important to keep the page-fault rate low.
 - Otherwise EAT increases and slowing the process execution dramatically.

Advantages of VM: Process Creation

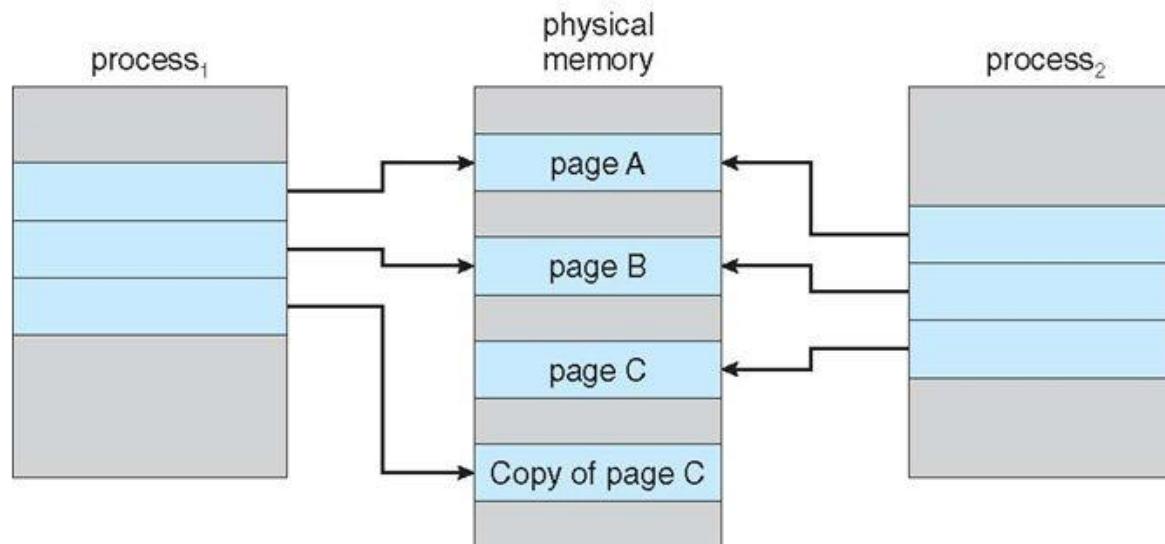
- Virtual memory allows other benefits during process creation:
 - Copy-on-Write
 - Memory-Mapped Files

Copy-on-Write

- Copy-on-Write (COW) allows both parent and child processes to initially *share* the same pages in memory.
 - If either process modifies a shared page, only then the page copied.
- COW allows more efficient process creation as only modified pages are copied.
- Free pages are allocated from a *pool* of zeroed-out pages.

Copy on Write

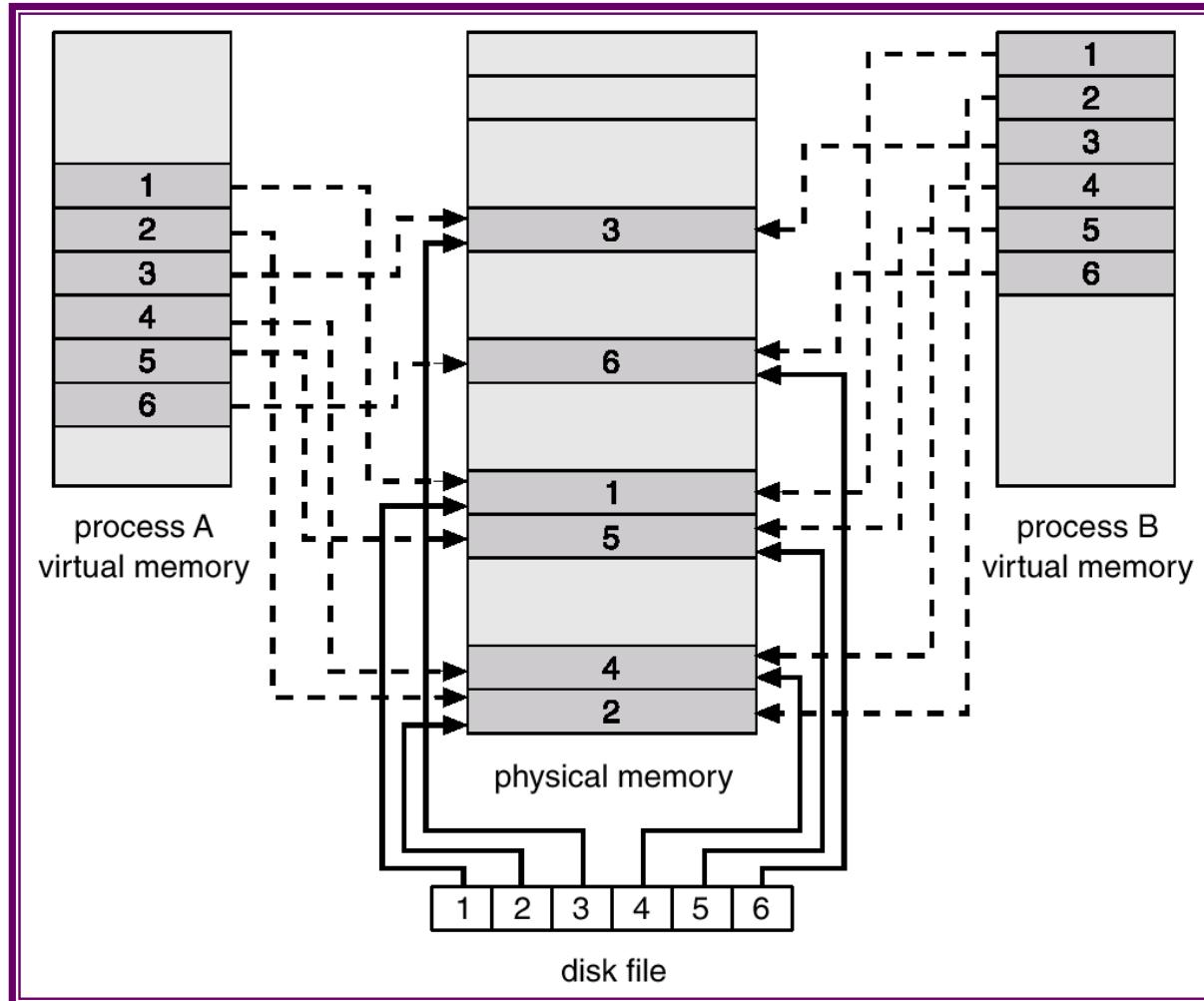
- Recall: the `fork()` system call creates a child process that is a duplicate of its parent
- Since the child might not modify its parents pages, we can employ the copy-on-write technique:
 - The child initially shares all pages with the parent.
 - If either process modifies a page, then a copy of that page is created.



Memory-Mapped Files

- Memory-mapped file I/O allows file I/O to be treated as routine memory access by *mapping* a disk block to a page in memory.
- A file is initially read using demand paging. A page-sized portion of the file is read from the file system into a physical page. Subsequent reads/writes to/from the file are treated as ordinary memory accesses.
- Simplifies file access by treating file I/O through memory rather than **read()** **write()** system calls.
- Also allows several processes to map the same file allowing the pages in memory to be shared.

Memory Mapped Files



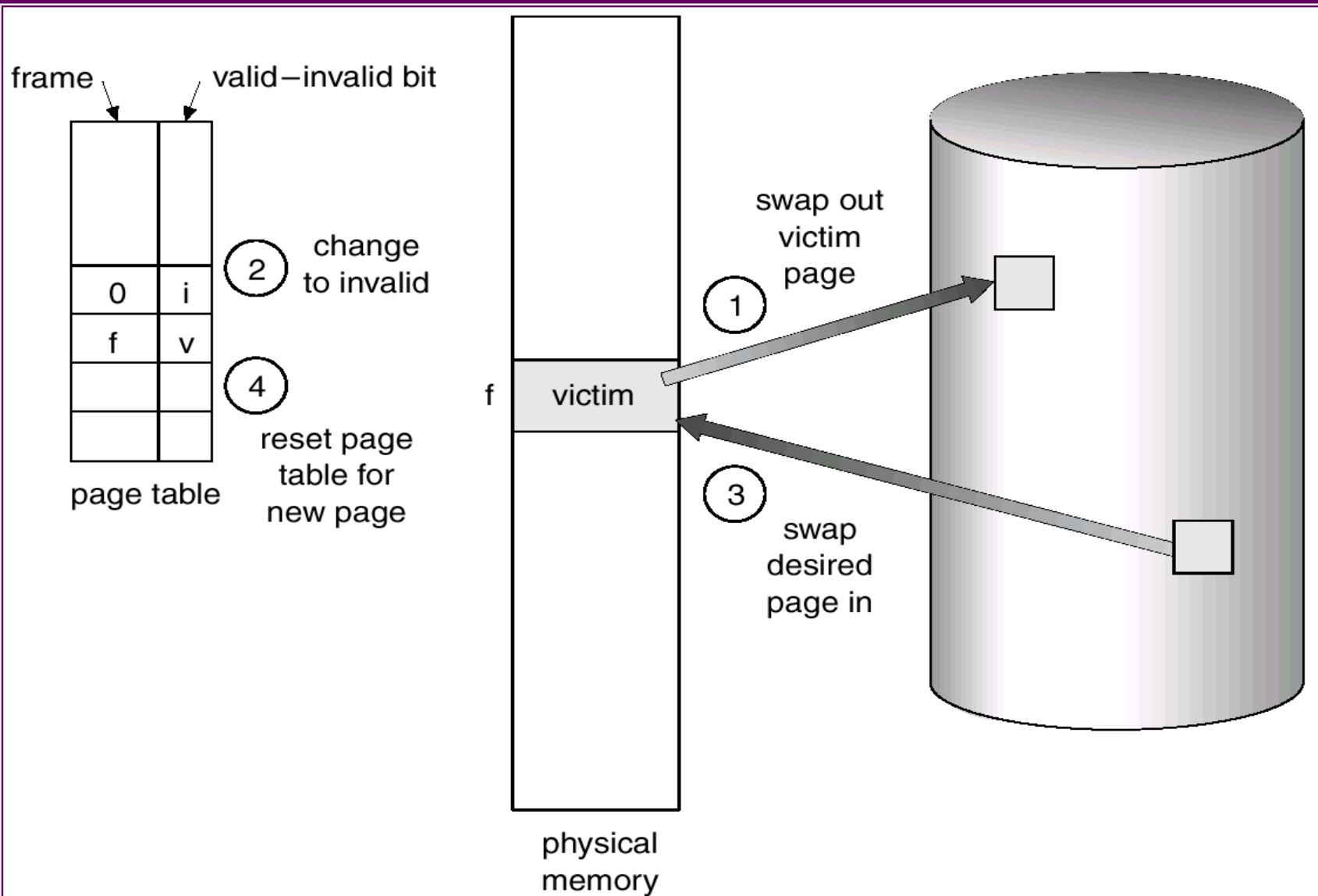
Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.

Basic Page Replacement

1. Find the location of the desired page on disk.
2. Find a free frame:
 - If there is a free frame, use it.
 - If there is no free frame, use a page replacement algorithm to select a *victim* frame.
3. Read the desired page into the (newly) free frame. Update the page table.
4. Restart the process.

Page Replacement



Reducing overhead: modify bit

- Use *modify (dirty) bit* to reduce overhead of page transfers – only modified pages are written to disk.
- Modify bit can be used to reduce the overhead with the help of hardware.
- It is set indicating the page has been modified.
- While replacing a page
 - If the modify bit is set, we must write that page to disk.
 - If it is not set we can avoid overwriting it, if is not overwritten.

Plan for remaining topics from today (16 Oct; Friday)

| | |
|--|---|
| 16 Oct., Friday | Virtual memory |
| 19 Oct., Monday (1130-1230) | Disk scheduling and RAID |
| 19 Oct., Monday (130 -230) | Overview of File management, Protection |
| 21 Oct., Wednesday (1130-12.30) | Quiz 6 (Topics: Memory management, Virtual Memory) |
| 23 Oct., Friday (1130-12.30) | Overview of computer security and media management |
| 26 Oct, Monday (1130-1230) | Intro. to computer networks |
| 26 Oct. Monday (130-230) | Computer networks (NWs): Physical layer |
| 28 Oct. Wednesday (1130-1230) | Quiz 7 (Topics: Disk scheduling and RIAD, Computer Security, File management, Protection, Security, Media management) |
| 2 Nov (Monday) (1130-1230) | Computer networks: Data link layer |
| 2 Nov (Monday) (130-230) | Computer networks: Data link layer |
| 4 Nov., (Wednesday) (1130-1230) | Computer networks: Network Layer |
| 6 Nov., (Friday) (1130-1230) | Computer networks : Network layer |
| 9 Nov., (Monday) (1130-1230) | Computer networks: Network layer |
| 11 Nov. (Wednesday) (1130-1230) | Quiz 8 (Topics: Intro., Data link and Network layers) |
| 13 Nov (Friday) (1130-1230) | Computer networks: Transport layer |
| 16 Nov (Monday) 1130-1230 | Computer networks: Transport layer |
| 18 Nov (Wednesday) (1130-1230) (LAST CLASS) | Computer networks: Presentation layer, Computer networks: Application layer |

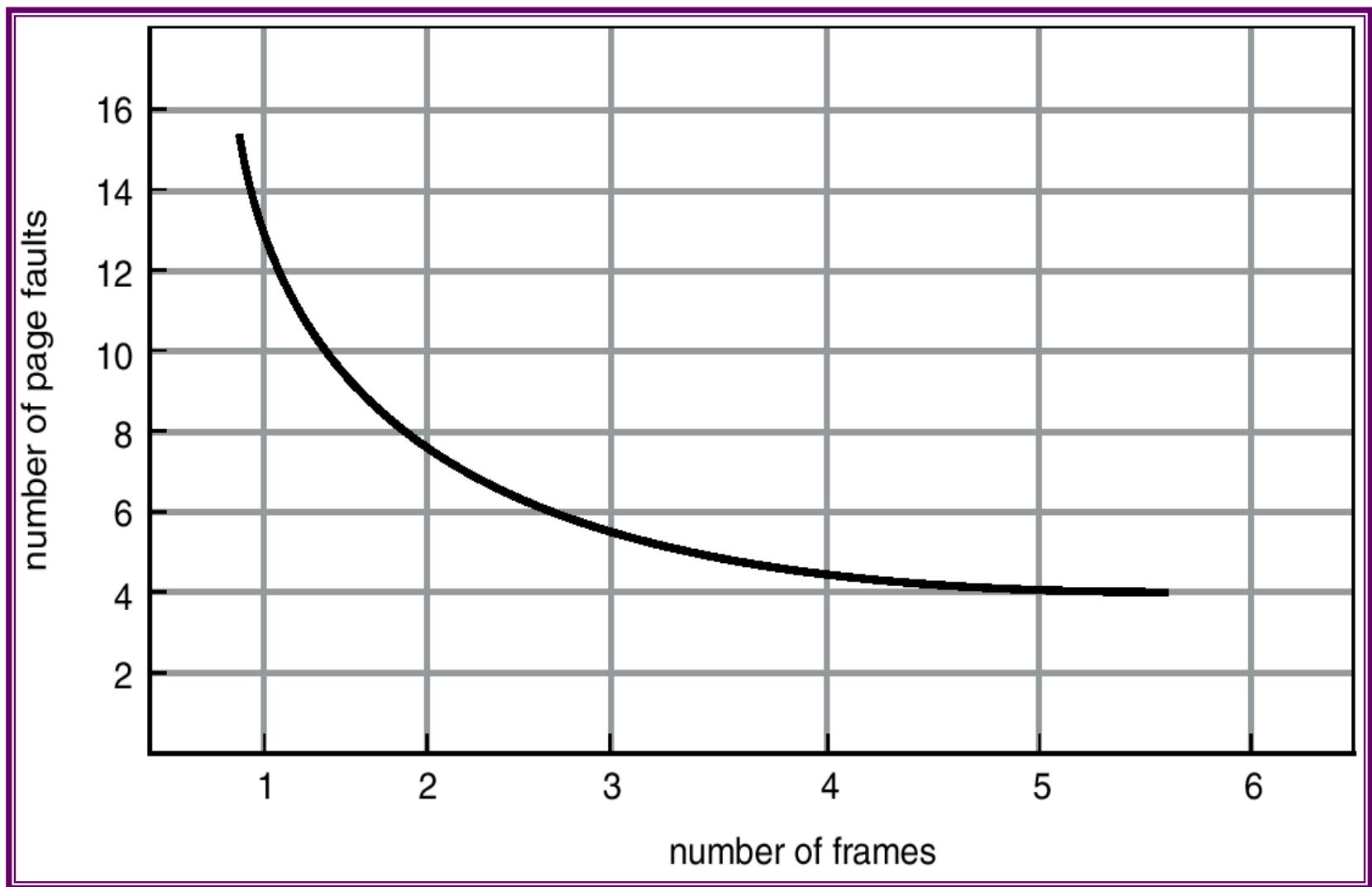
Page Replacement Algorithms

- Page replacement completes the separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.
 - Enormous virtual memory can be provided on a smaller physical memory.
- Two major problems are solved to implement demand paging
 - **Frame allocation algorithm**
 - If multiple processes exist in memory we have to decide the number frames for each process
 - **Page replacement algorithm**
 - We have to select a frame that is to be replaced.

Page Replacement Algorithms

- Want lowest page-fault rate.
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.
- In all our examples, the reference string is
 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
- Address reference divided by page size.
 - If the address reference is 0432 and page size is 100 then the reference number is $0432/100=4$

Graph of Page Faults Versus The Number of Frames



First-In-First-Out (FIFO) Algorithm

- Oldest page is replaced; FIFO queue; replace the head of the queue
 - Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

| | | | |
|---|---|---|---|
| 1 | 1 | 4 | 5 |
| 2 | 2 | 1 | 3 |
| 3 | 3 | 2 | 4 |

9 page faults

- 4 frames

| | | | |
|---|---|---|---|
| 1 | 1 | 5 | 4 |
| 2 | 2 | 1 | 5 |
| 3 | 3 | 2 | |
| 4 | 4 | 3 | |

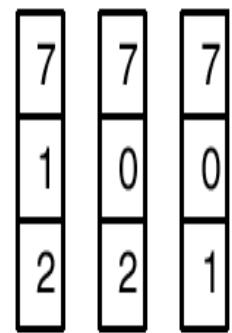
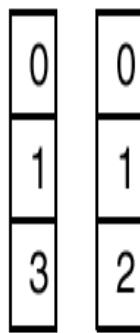
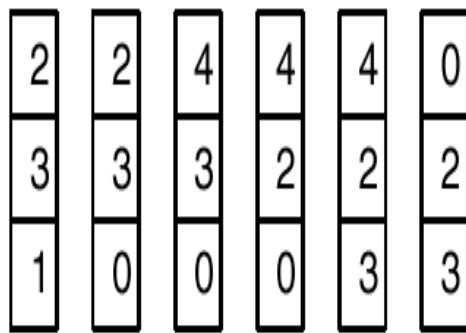
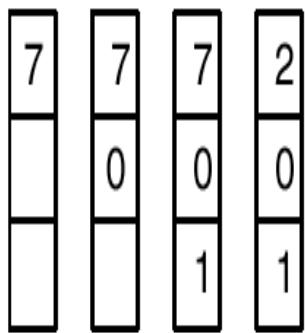
10 page faults

- FIFO Replacement – Belady's Anomaly
 - more frames \Rightarrow more page faults

FIFO Page Replacement

reference string

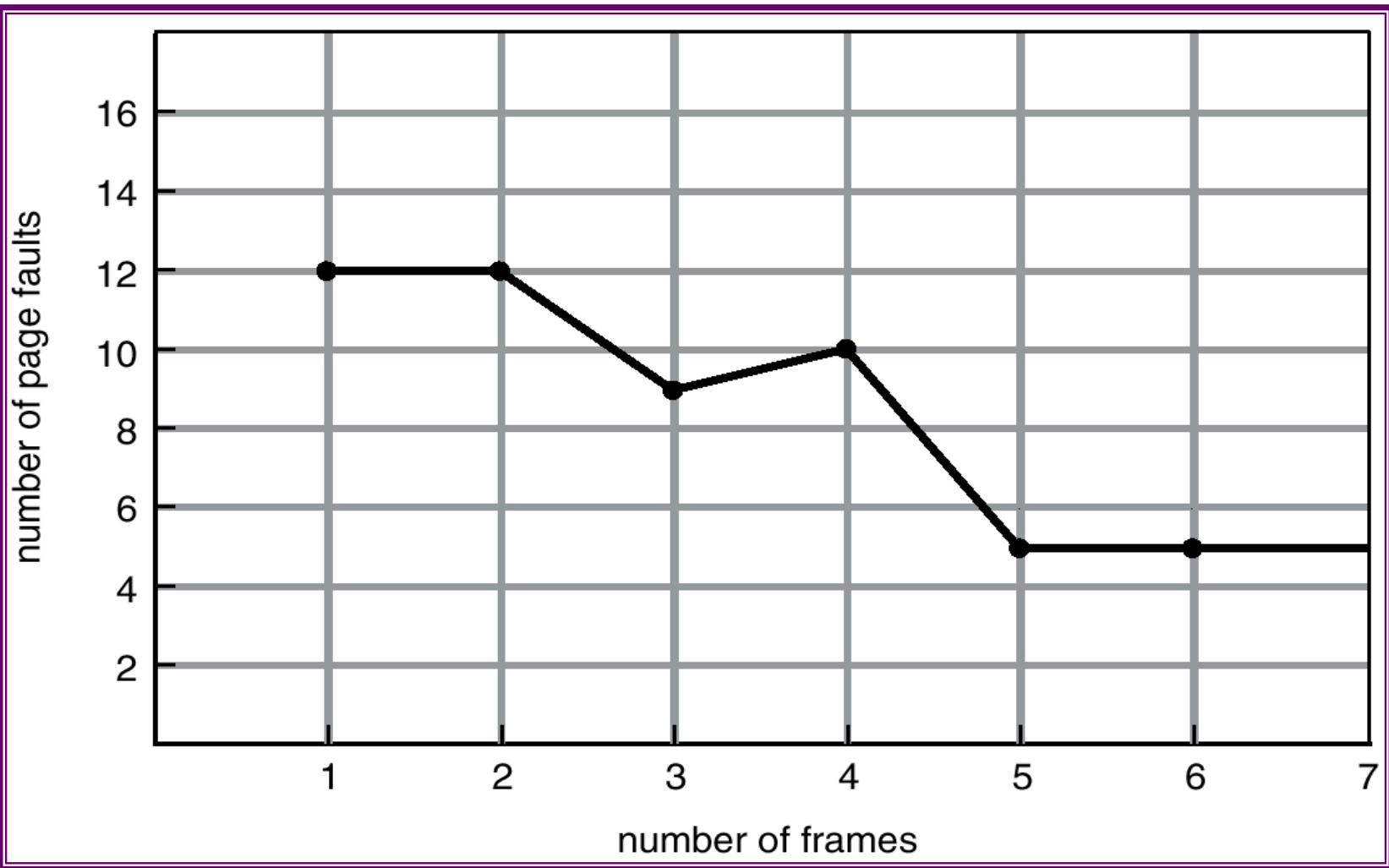
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

FIFO Illustrating Belady's Anomaly

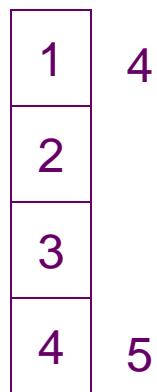
For some algs, page fault rate increases with number of pages.



Optimal Algorithm

- Replace page that will not be used for longest period of time.
- 4 frames example

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



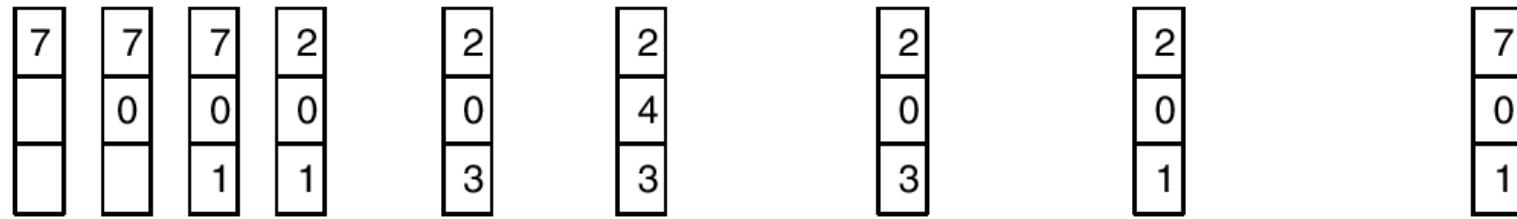
6 page faults

- How do you know this?
- Used for measuring how well your algorithm performs.

Optimal Page Replacement

reference string

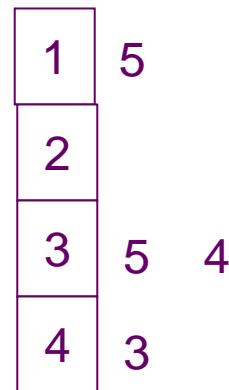
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

Least Recently Used (LRU) Algorithm

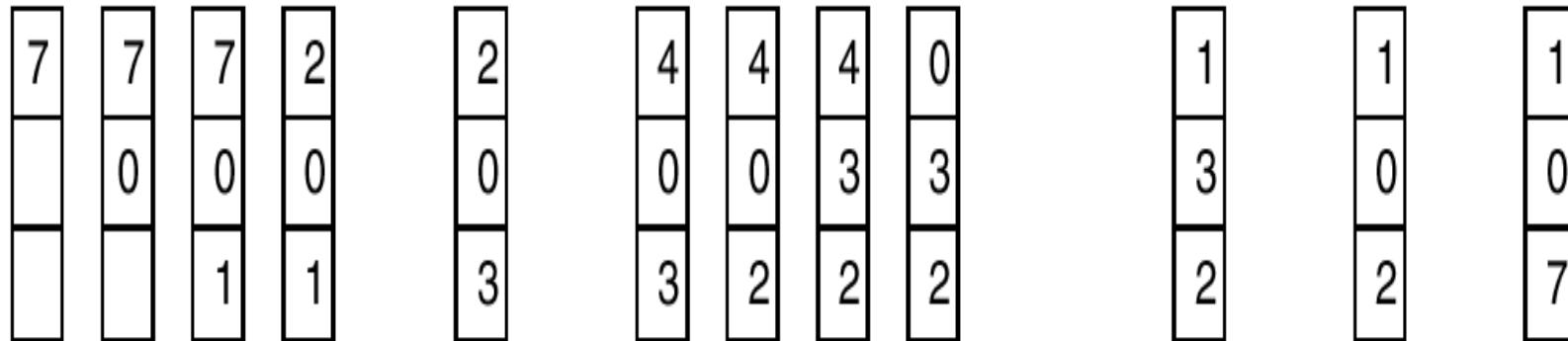
- The page that has not been used for longest period of time is replaced.
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



LRU Page Replacement

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

- The performance is good. But, How to Implement ?

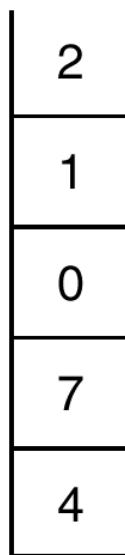
LRU Algorithm Implementation

- Counter implementation
 - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
 - When a page needs to be changed, look at the counters to determine which are to change.
 - Issues:
 - Search pf page table to find LRU page, Overflow of clock,..
- Stack implementation – keep a stack of page numbers in a double link form:
 - Page referenced:
 - move it to the top
 - requires 6 pointers to be changed
 - Update is expensive
 - No search for replacement
 - Top is the most recently used page and bottom is the LRU page.

Use Of A Stack to Record The Most Recent Page References

reference string

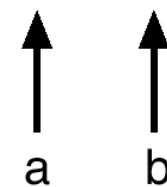
4 7 0 7 1 0 1 2 1 2 7 1 2



stack before a



stack after b



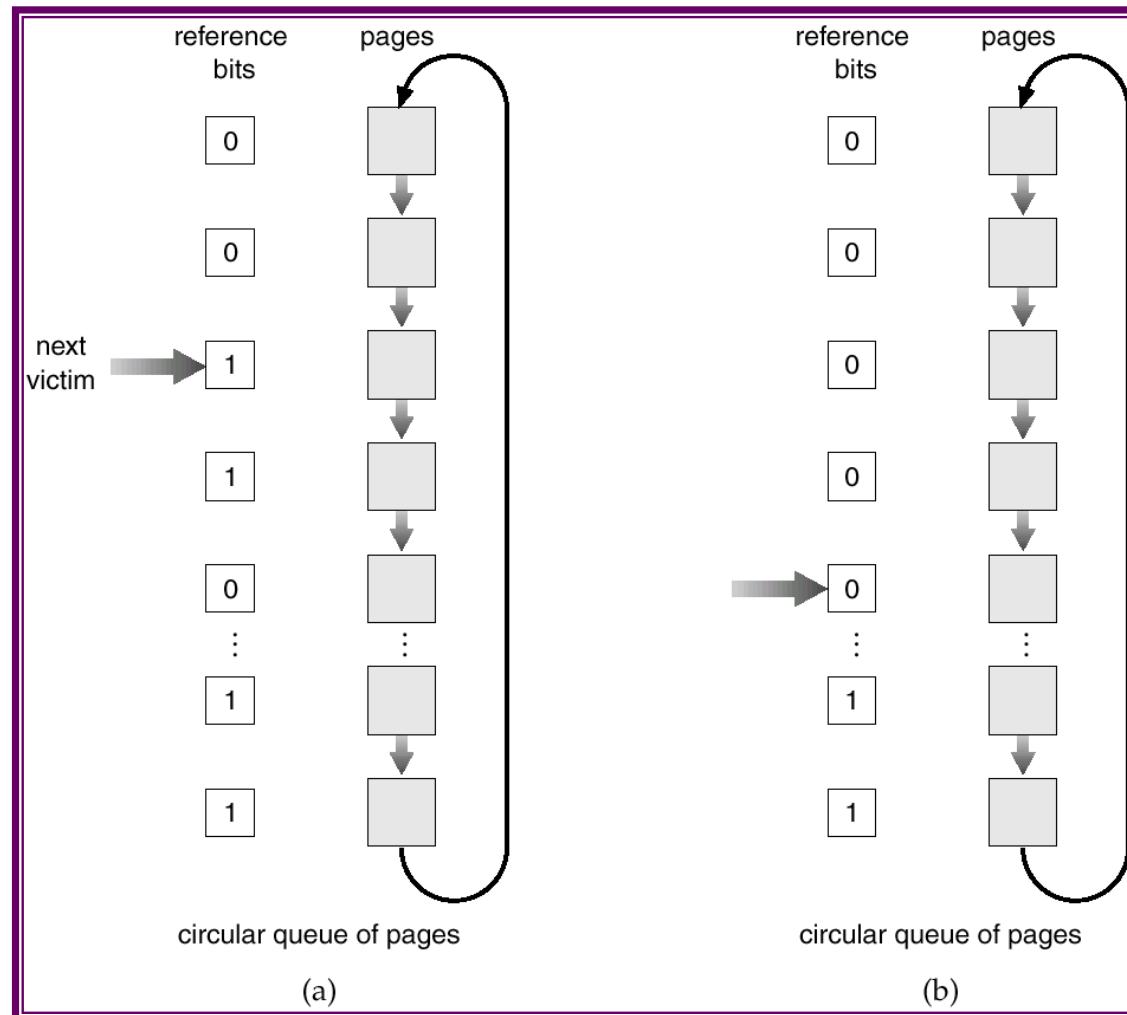
Performance issue: Stack and Counters

- The updating of stack or clock must be done on every memory reference.
- If we use interrupt for every reference, to allow software to update data structures, it would slow every reference by a factor of 10.
 - Few systems tolerate such degradation in performance.
- Sol:
 - Systems follow LRU approximation implemented through hardware.

LRU Approximation Algorithms

- Reference bit
 - With each page associate a bit, initially = 0
 - When page is referenced bit set to 1.
 - Replace the one which is 0 (if one exists). We do not know the order, however.
- Additional ordering:
 - By maintaining 8-bit byte for each page.
 - Shifting can be used to record the history (interrupt to OS for every 100 msec).
- Second chance
 - Need one reference bit.
 - Clock replacement.
 - If page to be replaced (in clock order) has reference bit = 1. then:
 - set reference bit 0; arrival time is set to current time.
 - leave page in memory.
 - replace next page (in clock order), subject to same rules.
 - Similar to FIFO if all bits are set.

Second-Chance (clock) Page-Replacement Algorithm



Enhanced-second chance algorithm

- Use reference bit and modify bit as an ordered pair.
 - (0,0) neither recently used nor modified – best page to replace.
 - (0,1) not recently used but modified- not quite good; to be written before replacement.
 - (1,0) recently used but clean – it probably used again soon.
 - (1,1) recently used and modified- it probably will be used again soon; to be written before replacement.
- Each page is one of four classes.

Other algorithms: Counting Algorithms

- Keep a counter of the number of references that have been made to each page.
- LFU Algorithm: replaces page with smallest count.
- MFU Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used.
- MFU and LFU are not used
 - Implementation is expensive
 - Do not approximate OPT well

Allocation of Frames

- Each process needs **minimum** number of pages.
- If there is a single process, entire available memory can be allocated.
- Multi-programming puts two or more processes in memory at same time.
- We must allocate minimum number of frames to each process.
- Two major allocation schemes.
 - fixed allocation
 - priority allocation

Minimum number of Frames

- Each process needs **minimum** number of pages.
- Minimum # of frames is defined based on the computer architecture.
 - Equal to maximum number of memory references per instruction.
 - LOAD may refer indirect reference that could also reference an indirect address.
 - # of frames = number of indirections.
 - Counter can be used to measure the number of indirections and trap the OS.
 - MVC in IBM370 machine requires 6 frames.
 - Two major allocation schemes.
 - fixed allocation
 - priority allocation

Fixed Allocation

- Equal allocation – e.g., if 100 frames and 5 processes, give each 20 pages.
- Proportional allocation – Allocate according to the size of process.
 - s_i = size of process p_i
 - $S = \sum s_i$
 - m = total number of frames
 - a_i = allocation for $p_i = \frac{s_i}{S} \times m$
 $m = 64$
 $s_1 = 10$
 $s_2 = 127$
 $a_1 = \frac{10}{137} \times 64 \approx 5$
 $a_2 = \frac{127}{137} \times 64 \approx 59$

Priority Allocation

- Use a proportional allocation scheme using priorities rather than size.
- If process P_i generates a page fault,
 - select for replacement one of its frames.
 - select for replacement a frame from a process with lower priority number.

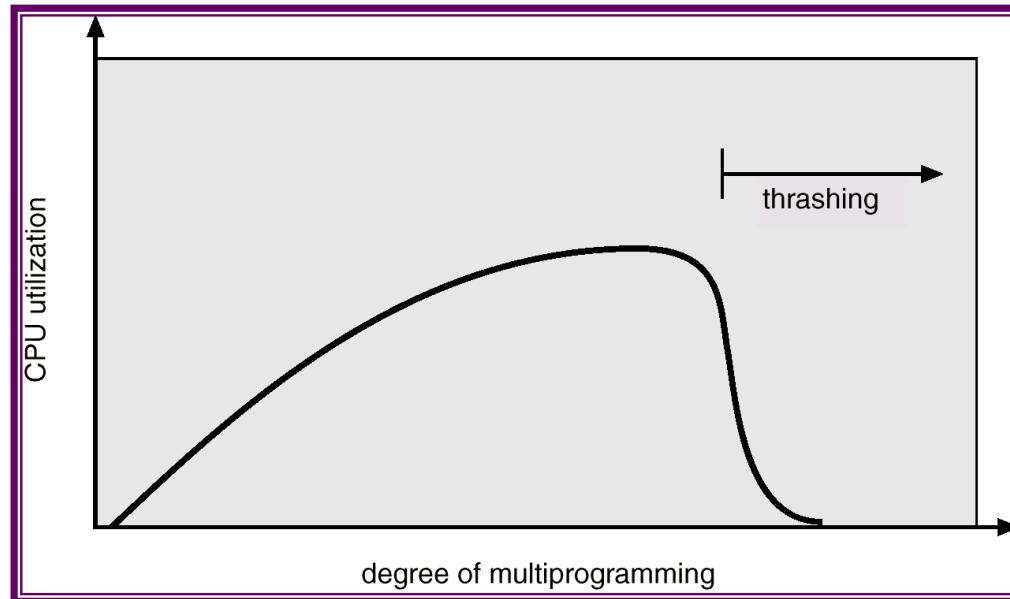
Global vs. Local Allocation

- **Global** replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another.
- **Local** replacement – each process selects from only its own set of allocated frames.
- With local replacement, # of frames does not change.
 - Performance depends on the paging behavior of the process.
 - Free frames may not be used
- With global replacement, a process can take a frame from another process.
 - Performance depends not only paging behavior of that process, but also paging behavior of other processes.
- In practice global replacement is used.

Thrashing

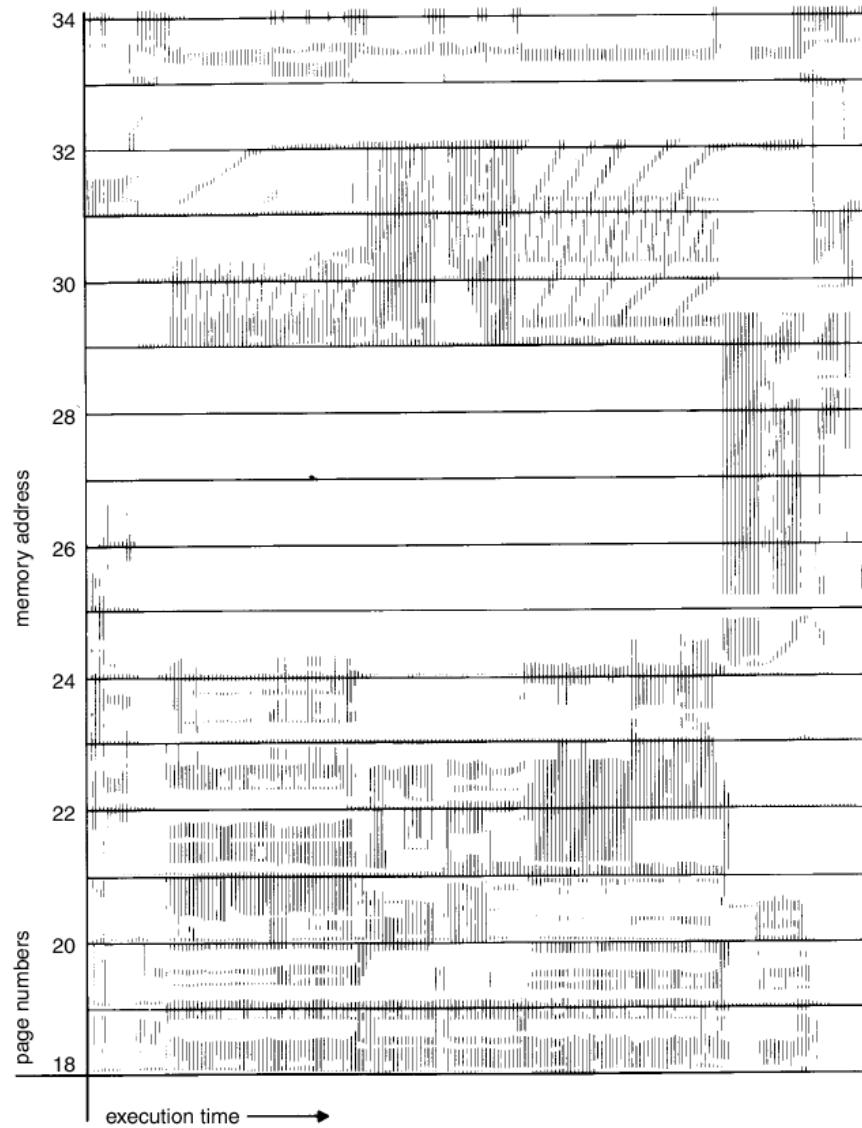
- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
 - □ low CPU utilization.
 - operating system thinks that it needs to increase the degree of multiprogramming.
 - another process is added to the system.
- Thrashing is High paging activity.
- **Thrashing** ≡ a process is spending more time in swapping pages in and out.
- If the process does not have # of frames equivalent to # of active pages, it will very quickly page fault.
- Since all the pages are in active use it will page fault again.

Thrashing



- Why does paging work?
Locality model
 - Process migrates from one locality to another.
 - Localities may overlap.
- Why does thrashing occur?
 Σ size of locality > total memory size

Locality In A Memory-Reference Pattern



Causes of thrashing

- OS monitors CPU utilization
 - If it is low, increases the degree of MPL
- Consider that a process enters new execution phase and starts faulting.
- It takes pages from other processes
- Since other processes need those pages, they also fault, taking pages from other processes.
- The queue increases for paging device and ready queue empties
- CPU utilization decreases.
- Solution: provide process as many frames as it needs.
- But how we know how many frames it needs ?
- Locality model provides hope.

Locality model

- Locality is a set of pages that are actively used together.
- A program is composed of several different localities which may overlap.
 - Ex: even when a subroutine is called it defines a new locality.
- The locality model states that all the programs exhibit this memory reference structure.
- **This is the main reason for caching and virtual memory!**
- **If we allocate enough frames to a process to accommodate its current locality, it faults till all pages are in that locality are in the MM. Then it will not fault.**
- **If we allocate fewer frames than current locality, the process will thrash.**

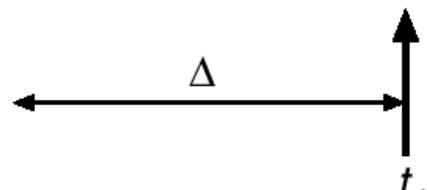
Working-Set Model

- Based on locality
- Define a parameter Δ ;
 - $\Delta \equiv$ working-set window \equiv a fixed number of page references
Example: 10,000 instruction
- Most recent references are examined
- WSS_i (working set of Process P_i) =
total number of pages referenced in the most recent Δ (varies in time)
 - if Δ too small will not encompass entire locality.
 - if Δ too large will encompass several localities.
 - if $\Delta = \infty \Rightarrow$ will encompass entire program.
- $D = \sum WSS_i \equiv$ total demand frames
- if $D > m \Rightarrow$ Thrashing
- Policy: if $D > m$, then suspend one of the processes.

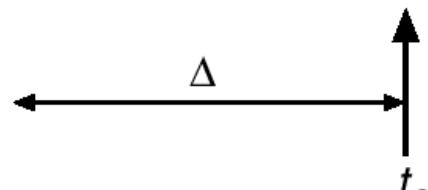
Working-set model

page reference table

... 2 6 1 5 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 ...



$$WS(t_1) = \{1, 2, 5, 6, 7\}$$



$$WS(t_2) = \{3, 4\}$$

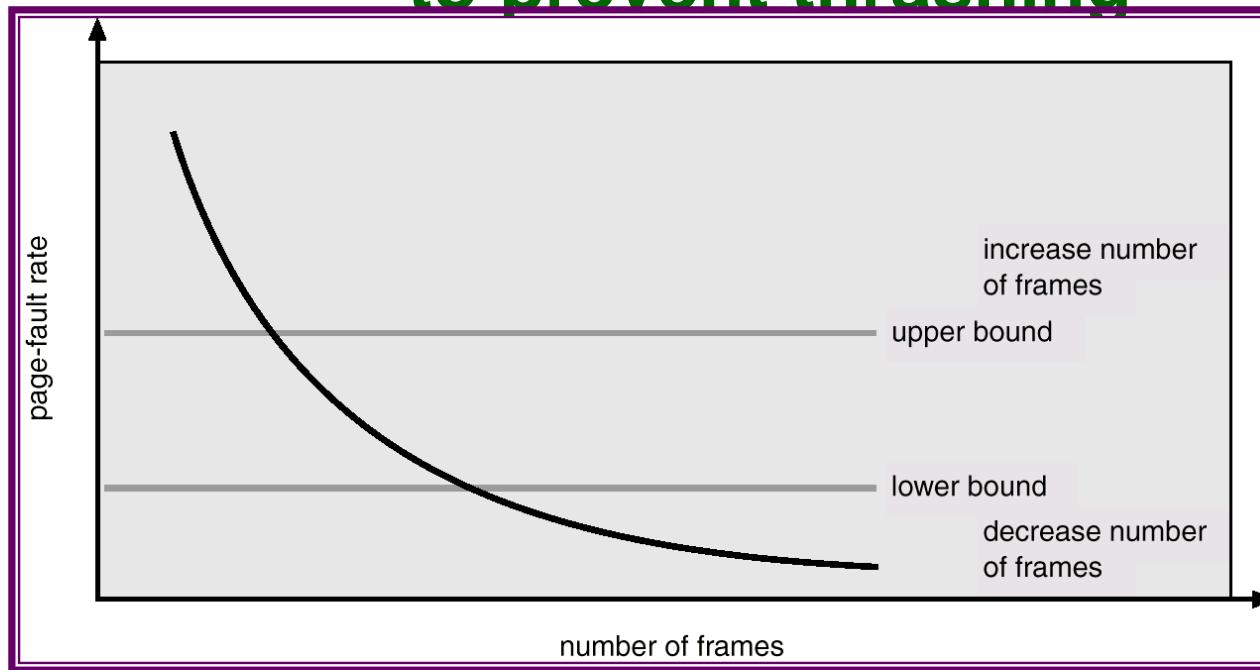
Working Set

- OS monitors the WS of each process allocates to that working set enough frames equal to WS size.
- If there are enough extra frames, another process can be initiated.
- If $D > m$, OS suspends a process, and its frames are allocated to other processes.
- The WS strategy prevents thrashing by keeping MPL as high as possible.
- However, we have to keep track of working set.

Keeping track of Working Set

- Approximate with interval timer + a reference bit
- Example: $\Delta = 10,000$
 - Timer interrupts after every 5000 time units.
 - Keep in memory 2 bits for each page.
 - Whenever a timer interrupts copy and sets the values of all reference bits to 0.
 - If one of the bits in memory = 1 \Rightarrow page in working set.
- Why is this not completely accurate?
 - We can not tell when the reference was occurred.
 - Accuracy can be increased by increasing frequency of interrupts which also increases the cost.

Page-Fault Frequency approach to prevent thrashing



- Thrashing has a high page-fault rate.
- Solution: Control or establish “acceptable” page-fault rate.
 - If page fault rate is too low (below lower bound), process loses frame.
 - If page fault rate is too high (exceeds upper bound), process gains frame.
- Process is suspended if no free frames are available. The freed frames are distributed among other processes.

Allocation of Kernel memory

- Buddy allocation
 - Allocates fixed size segment consisting of physically contiguous pages
 - Uses power-of-2 allocator

Adjacent buddies can be joined to meet a bigger request
- Slab allocation
 - A slab is made up of one or more physically contiguous pages.
 - Use different caches for different size/kernel data structure.
 - Caches are mapped to slabs.

Other Considerations

- The selection of a page replacement algorithm and allocation policy are major decisions. There are many other considerations as well.
- Prepaging
 - Bring entire WS to the memory to prevent high level of initial paging.
- Page size selection
 - Fragmentation
 - Memory is better utilized if we have a small page size as pages are units of allocation.
 - table size
 - Large page size is desirable to decrease table size.
 - I/O overhead
 - Minimize I/O time argues for a larger page size.
 - Locality
 - With smaller page size, the locality will be improved.

Other Considerations (Cont.)

- **TLB Reach –**
 - Hit ratio should be increased.
 - □ The amount of memory accessible from the TLB.
 - Ideally, the working set of each process is stored in the TLB. Otherwise there is a high degree of page faults.
 - **TLB Reach = (TLB Size) X (Page Size)**
 - **Increase the Page Size.** This may lead to an increase in fragmentation as not all applications require a large page size.
 - **Provide Multiple Page Sizes.** This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation.

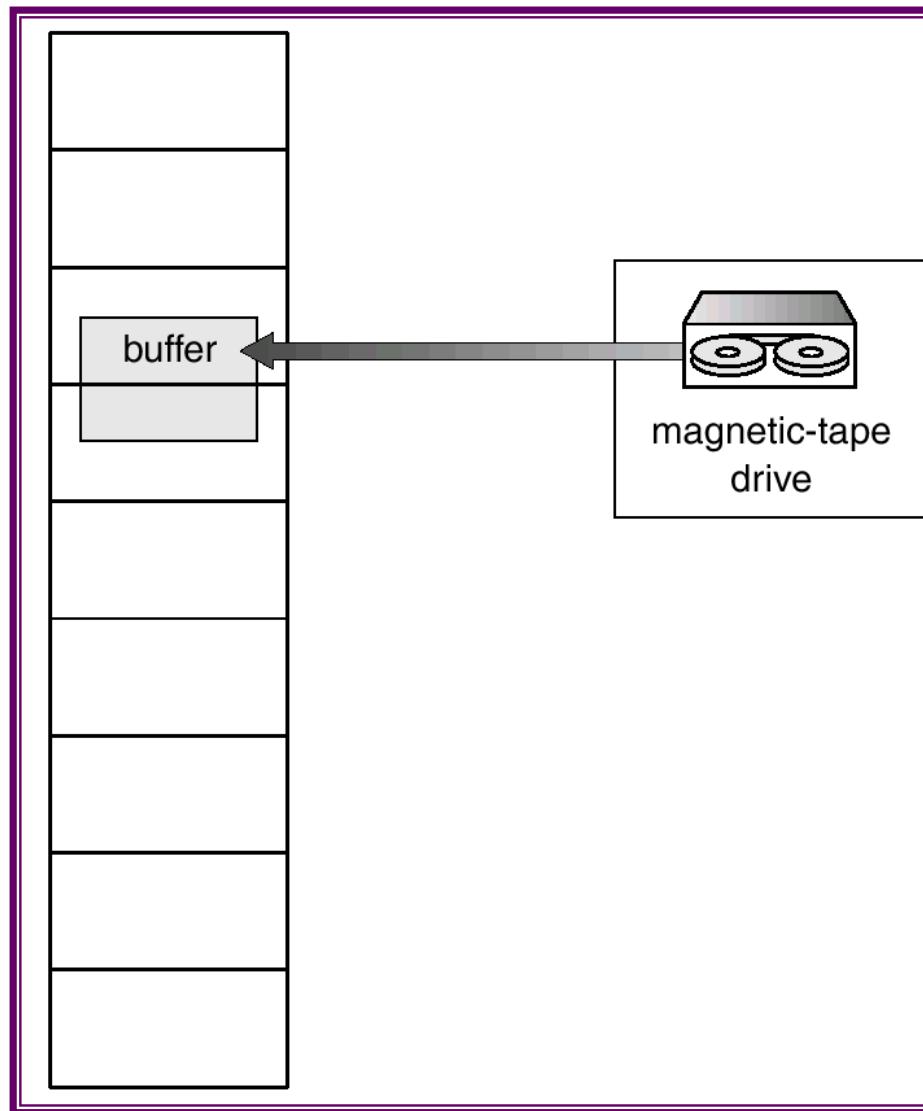
Other Considerations (Cont.)

- System performance can be improved if the user is completely aware of the paged nature of memory.
- Program structure
 - `int A[][] = new int[1024][1024];`
 - Each row is stored in one page
 - Program 1
$$\begin{aligned} &\text{for (j = 0; j < A.length; j++)} \\ &\quad \text{for (i = 0; i < A.length; i++)} \\ &\quad \quad A[i,j] = 0; \end{aligned}$$
1024 x 1024 page faults
 - Program 2
$$\begin{aligned} &\text{for (i = 0; i < A.length; i++)} \\ &\quad \text{for (j = 0; j < A.length; j++)} \\ &\quad \quad A[i,j] = 0; \end{aligned}$$
1024 page faults
- Careful selection of data structures and programming structures can increase locality, lower the page fault rate and the number of pages in the working set.
- Java (no pointers) has better locality of reference than C or C++ due to pointers that randomize page references.

Other Considerations (Cont.)

- **I/O Interlock** – Pages must sometimes be locked into memory.
- Consider I/O. Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm.

Reason Why Frames Used For I/O Must Be In Memory



Operating System Examples

- Windows NT
- Solaris 2

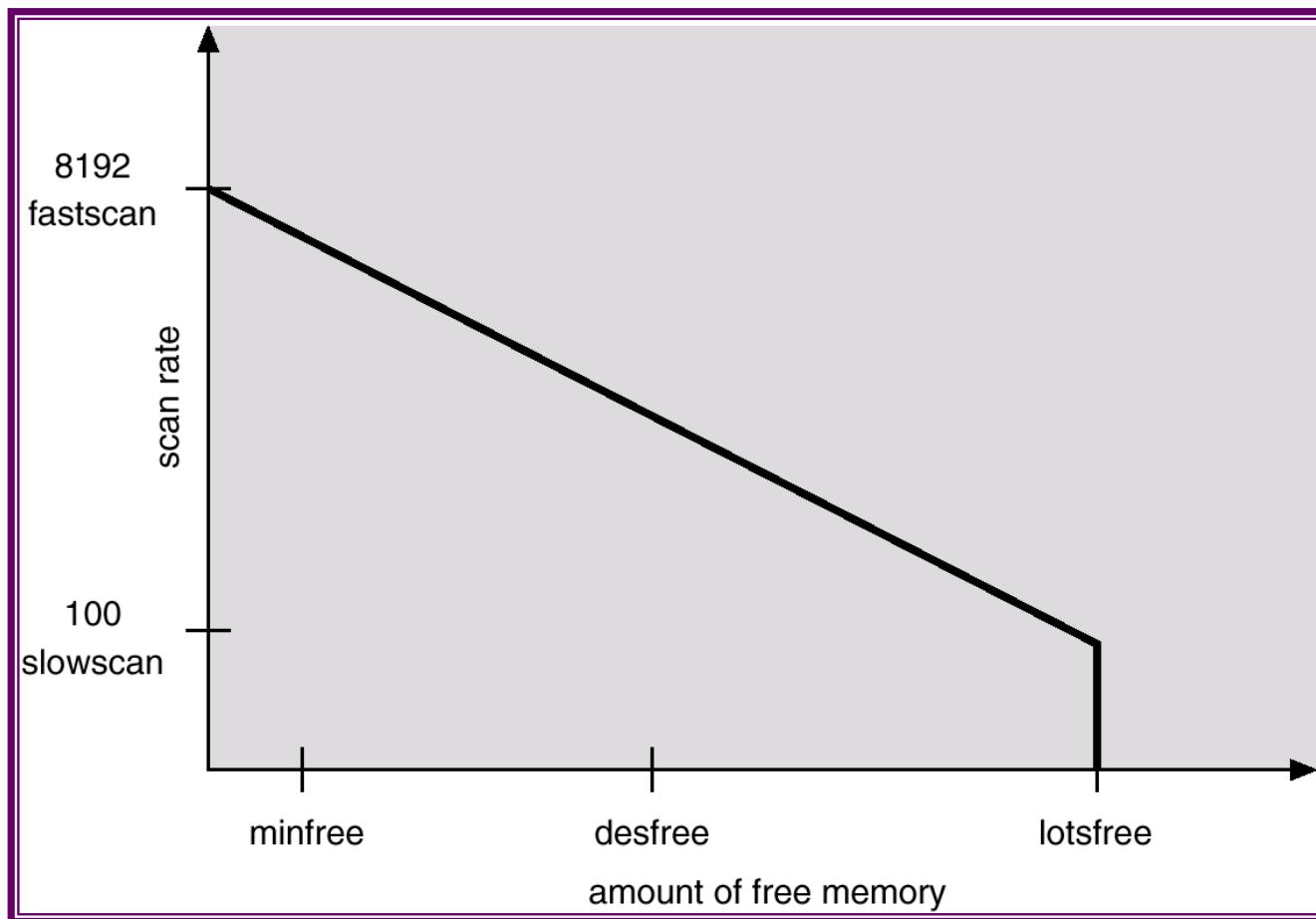
Windows NT

- Uses demand paging with **clustering**. Clustering brings in pages surrounding the faulting page.
- Processes are assigned **working set minimum** and **working set maximum**.
- Working set minimum is the minimum number of pages the process is guaranteed to have in memory.
- A process may be assigned as many pages up to its working set maximum.
- When the amount of free memory in the system falls below a threshold, **automatic working set trimming** is performed to restore the amount of free memory.
- Working set trimming removes pages from processes that have pages in excess of their working set minimum.

Solaris 2

- Maintains a list of free pages to assign faulting processes.
- **Lotsfree** – threshold parameter to begin paging.
- Paging is performed by *pageout* process.
- Pageout scans pages using modified clock algorithm.
- **Scanrate** is the rate at which pages are scanned. This ranged from **slowscan** to **fastscan**.
- Pageout is called more frequently depending upon the amount of free memory available.

Solar Page Scanner



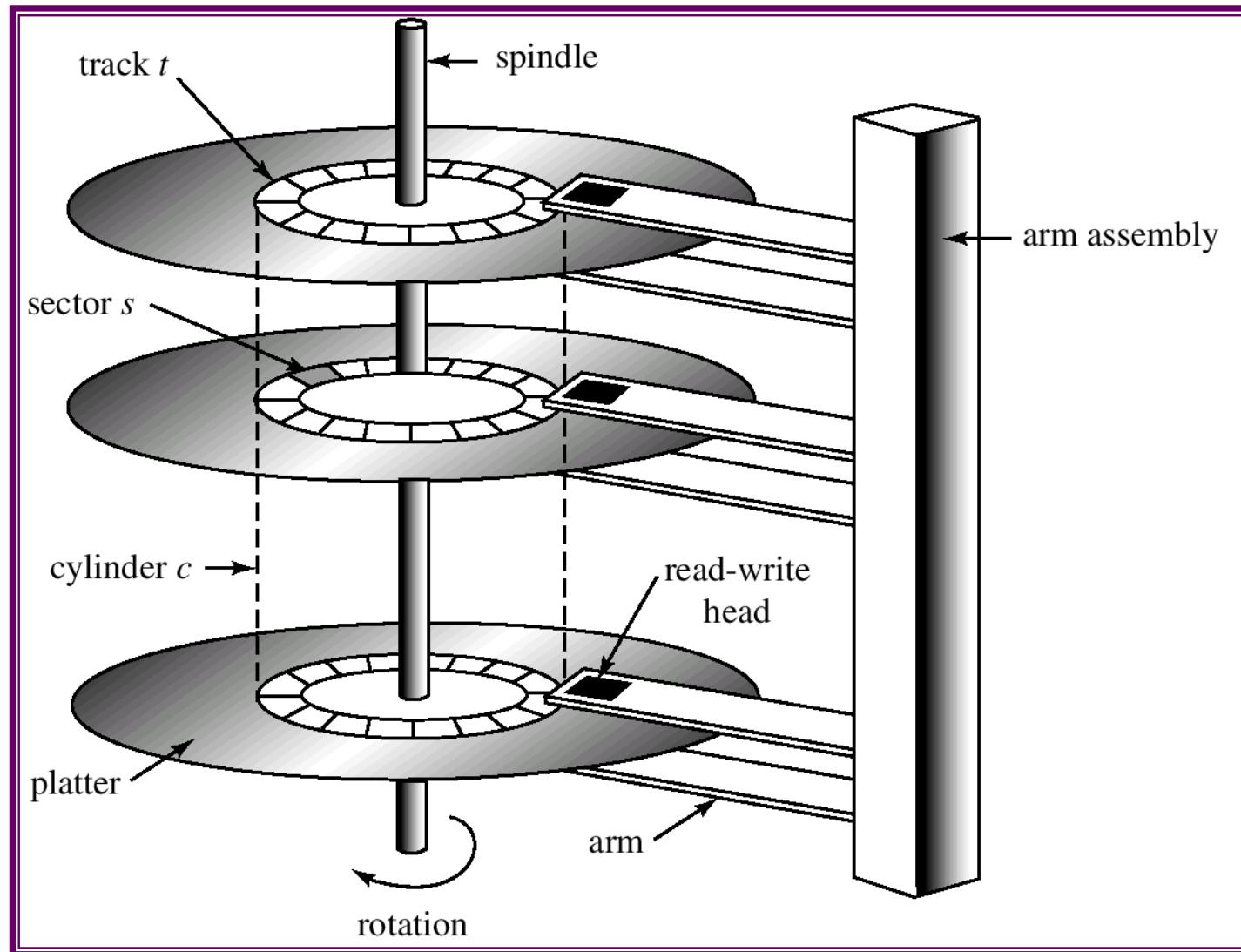
Secondary Storage Structure

- Disk Structure
- Disk Scheduling
- Disk Management
- RAID Structure

Disk Structure

- Disk drives are addressed as large 1-dimensional arrays of *logical blocks*, where the logical block is the smallest unit of transfer.
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially.
 - ◆ Sector 0 is the first sector of the first track on the outermost cylinder.
 - ◆ Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

Moving-Head Disk Mechanism



Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth.
- Access time has two major components
 - ◆ *Seek time* is the time for the disk are to move the heads to the cylinder containing the desired sector.
 - ◆ *Rotational latency* is the additional time waiting for the disk to rotate the desired sector to the disk head.
- Minimize seek time
- Seek time \approx seek distance
- Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

Disk Scheduling (Cont.)

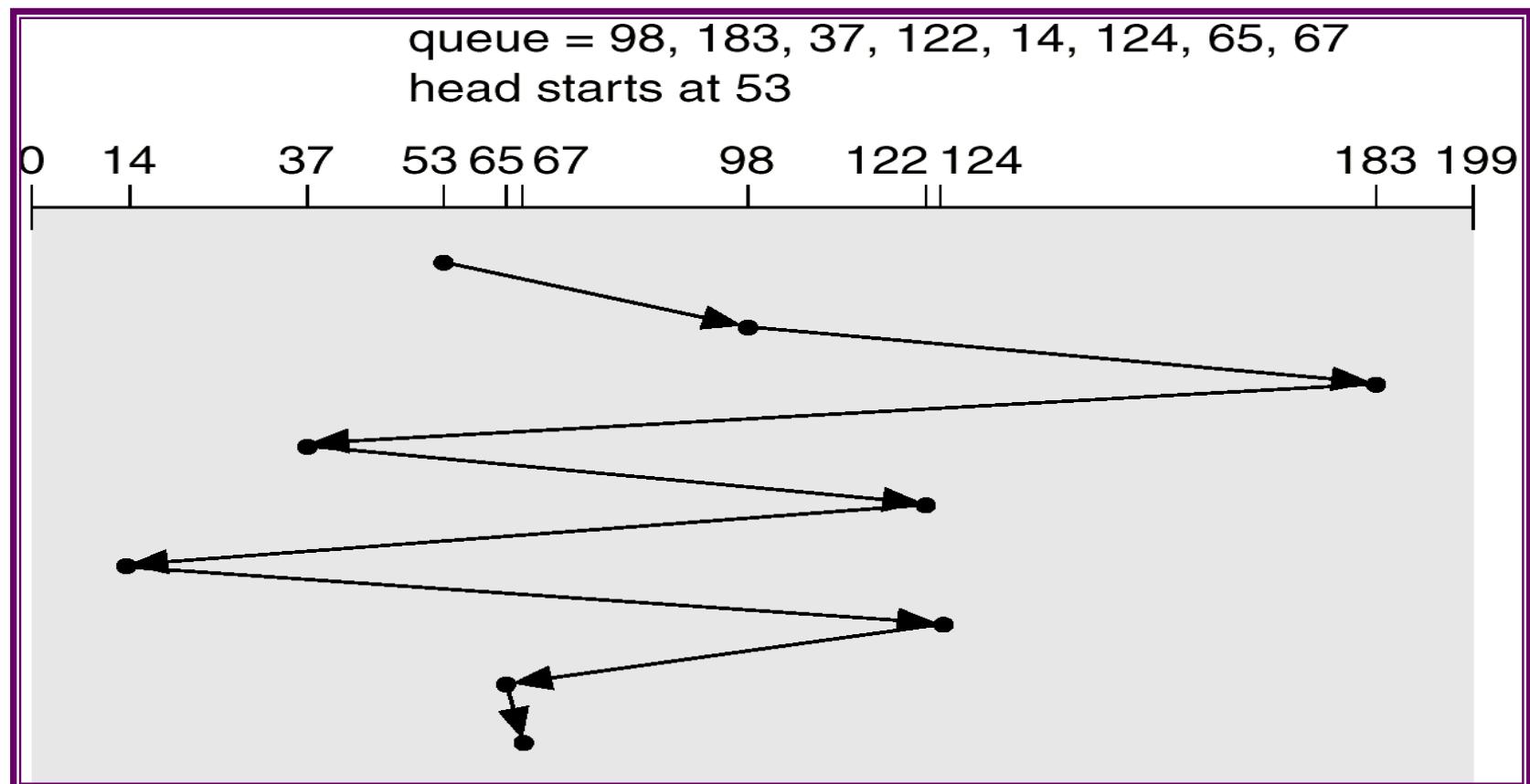
- Whenever a process needs I/O or from the disk, it issues a system call to the OS with the following information.
 - ◆ Whether the operation is input or output
 - ◆ What the disk address for the transfer is
 - ◆ What the memory address for the transfer is
 - ◆ What the number of bytes to be transferred is.
- Several algorithms exist to schedule the servicing of disk I/O requests.
- We illustrate them with a request queue (0-199).

98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53

FCFS

- FCFS
- +ve points: fair
- -ve points: slow
- Illustration shows total head movement of 640 cylinders.

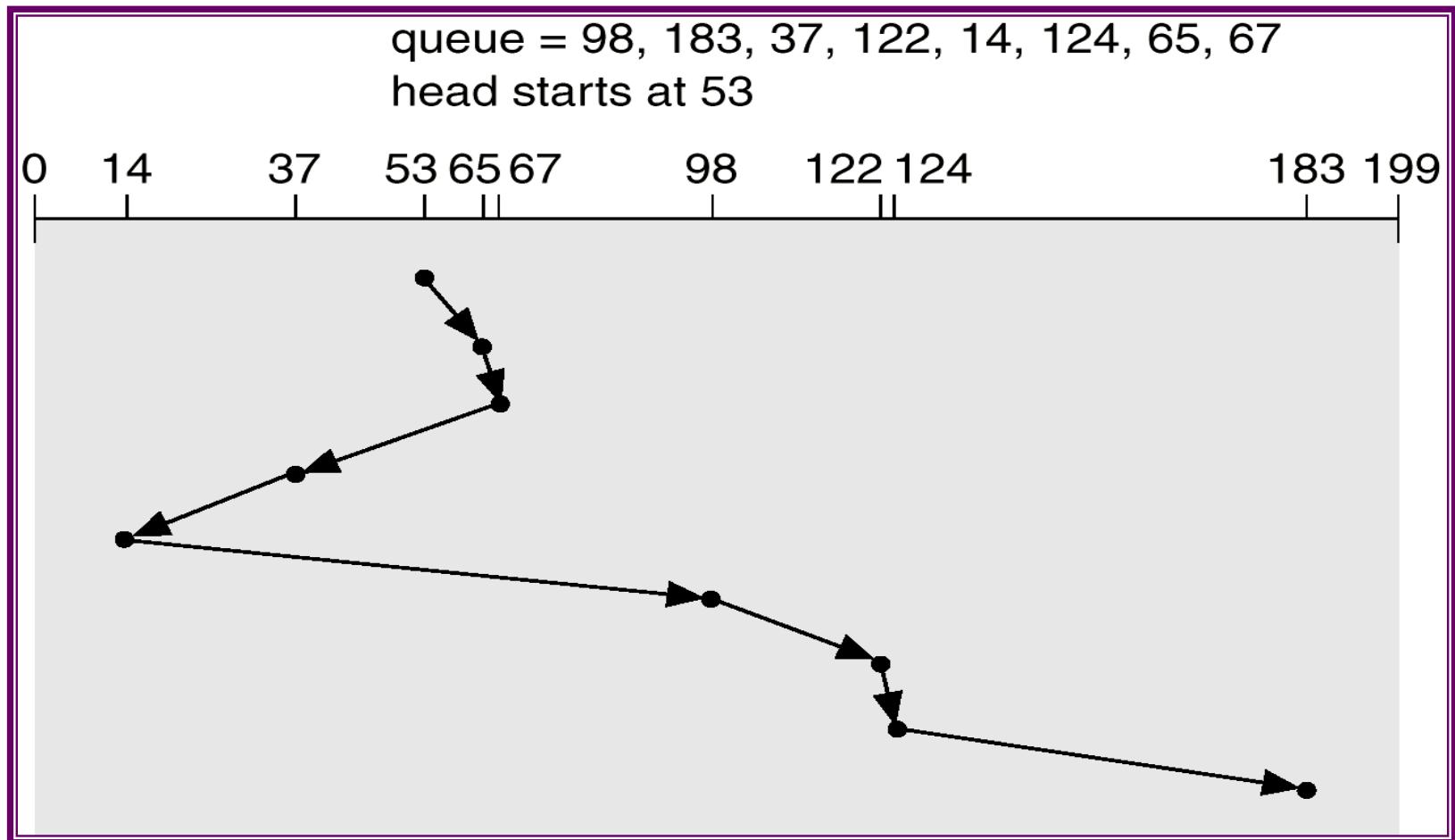


Shortest-seek-time-first (SSTF) algorithm

- Selects the request with the minimum seek time from the current head position.
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.
- Illustration shows total head movement of 236 cylinders.

SSTF (Cont.)

Total head movement=236 cylinders

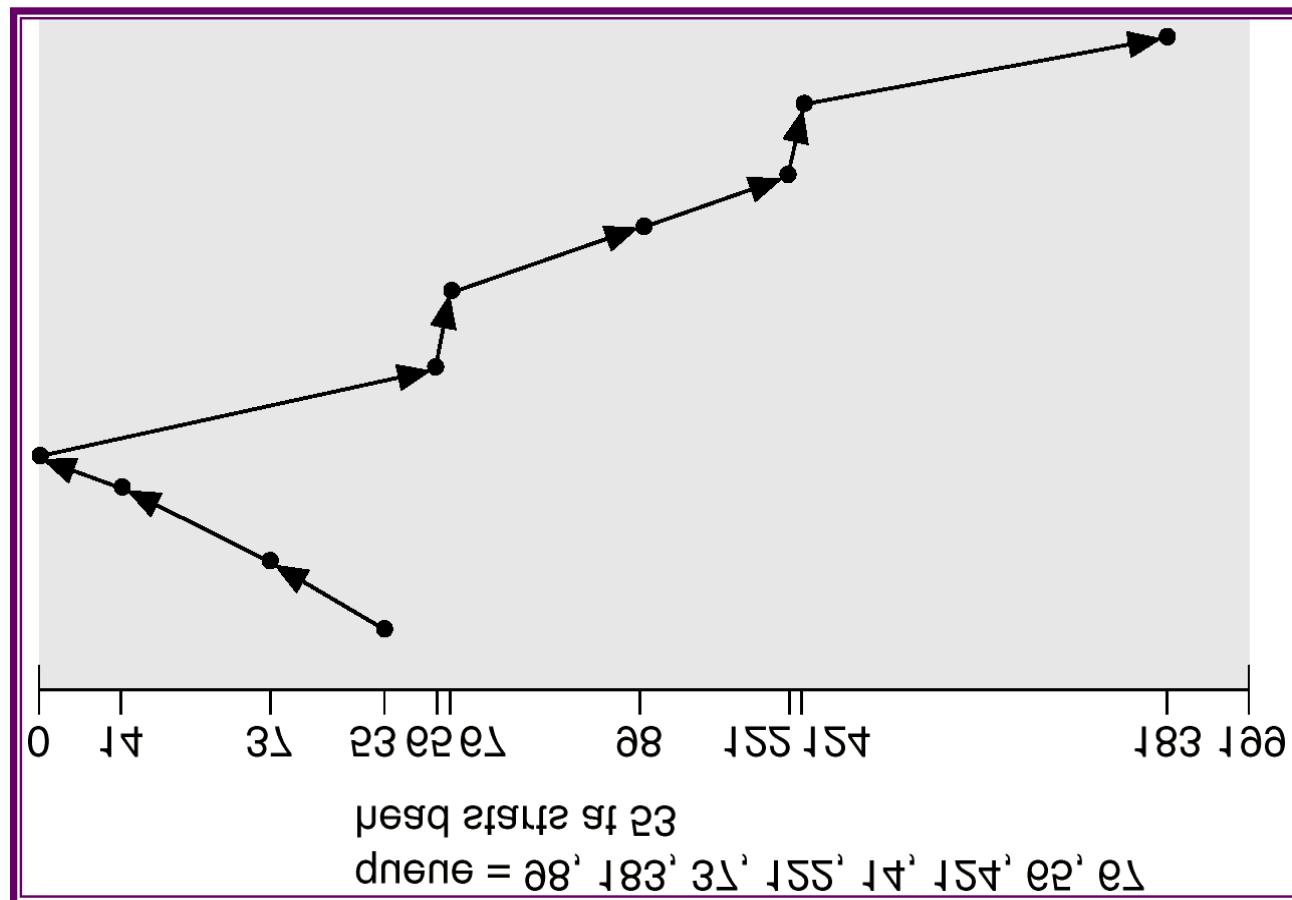


SCAN or Elevator algorithm

- The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.
- Sometimes called the *elevator algorithm*.
 - ◆ This algorithm does not give uniform wait time.
- Illustration shows total head movement of 208 cylinders.

SCAN (Cont.)

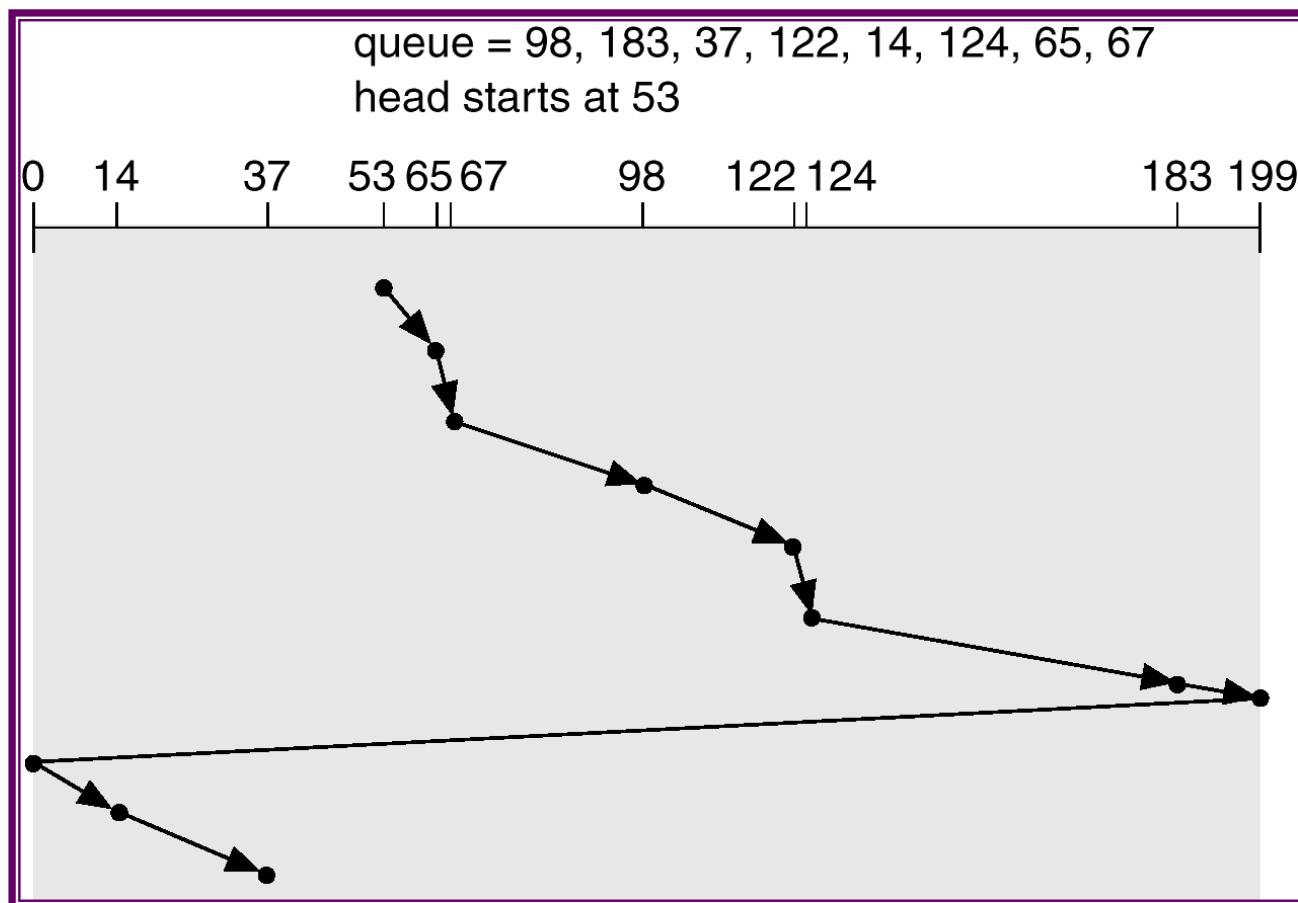
Total head movement=208 cylinders



C-SCAN

- Provides a more uniform wait time than SCAN.
- The head moves from one end of the disk to the other, servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

C-SCAN (Cont.)

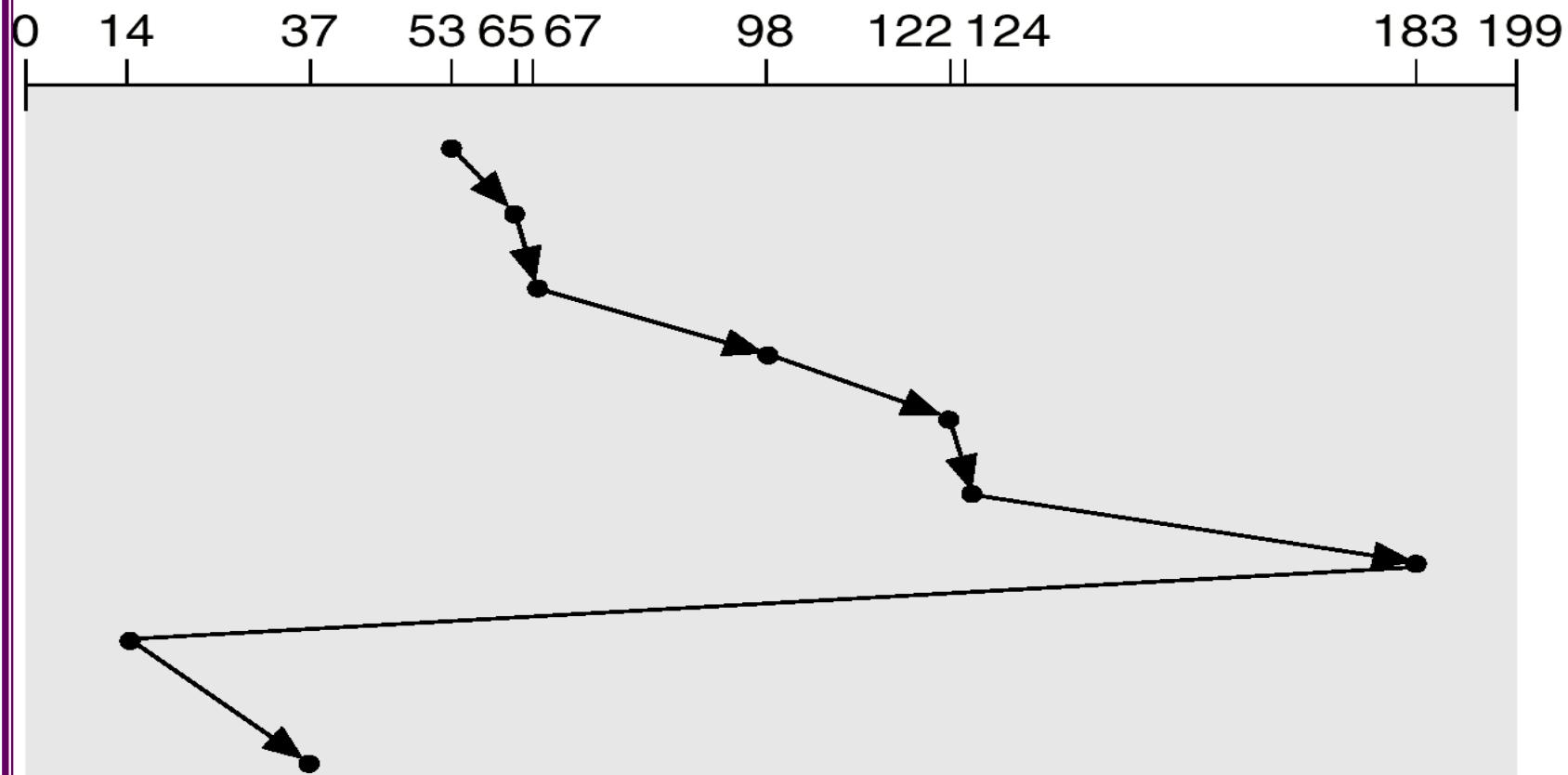


C-LOOK

- Version of C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.

C-LOOK (Cont.)

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



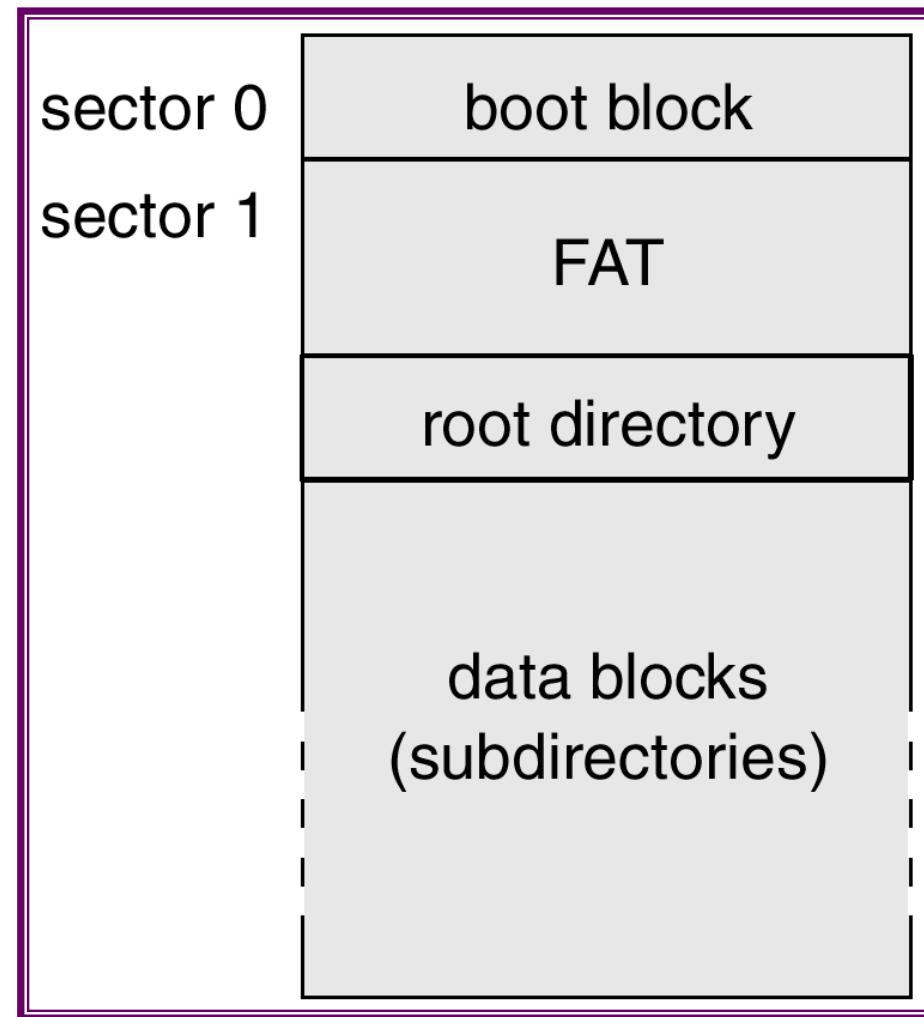
Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk.
- Performance depends on the number and types of requests.
- Requests for disk service can be influenced by the file-allocation method.
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary.
- Either SSTF or LOOK is a reasonable choice for the default algorithm.

Disk Management

- *Low-level formatting, or physical formatting* —
Dividing a disk into sectors that the disk controller
can read and write.
- To use a disk to hold files, the operating system still
needs to record its own data structures on the disk.
 - ◆ *Partition* the disk into one or more groups of cylinders.
 - ◆ *Logical formatting* or “making a file system”.
- Boot block initializes system.
 - ◆ The bootstrap is stored in ROM.
 - ◆ *Bootstrap loader* program.
- Methods such as *sector sparing* used to handle
bad blocks.

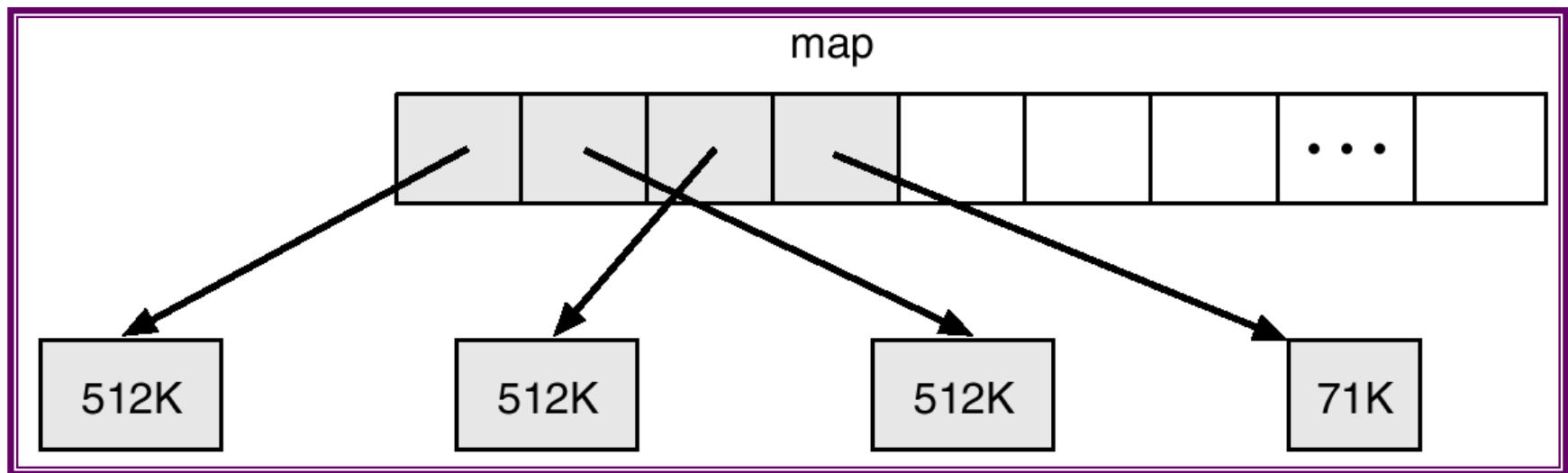
MS-DOS Disk Layout



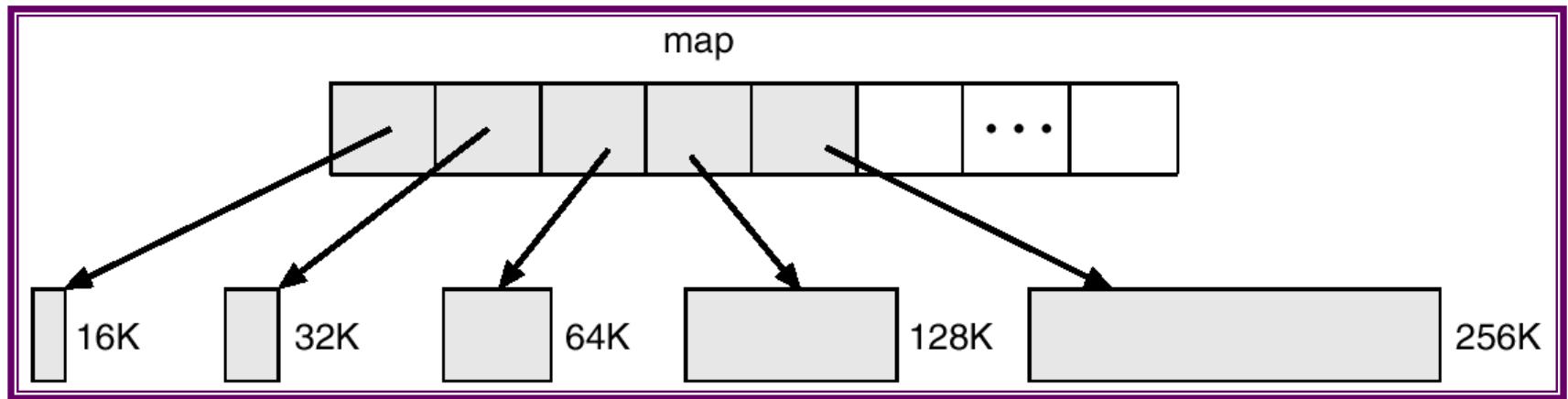
Swap-Space Management

- Swap-space — Virtual memory uses disk space as an extension of main memory.
- Swap-space can be carved out of the normal file system, or, more commonly, it can be in a separate disk partition.
- Swap-space management
 - ◆ 4.3BSD allocates swap space when process starts; holds *text segment* (the program) and *data segment*.
 - ◆ Kernel uses *swap maps* to track swap-space use.
 - ◆ Solaris 2 allocates swap space only when a page is forced out of physical memory, not when the virtual memory page is first created.

4.3 BSD Text-Segment Swap Map



4.3 BSD Data-Segment Swap Map



RAID Structure

■ Problem with disk

- ◆ Data transfer rate is limited by serial access.
- ◆ Reliability

■ Solution to both problems: Redundant arrays of inexpensive disks (RAID)

- In the past RAID (combination of cheap disks) is alternative for large and expensive disks.
- Today: RAID is used for their higher reliability and higher data-transfer rate.
- So the I in RAID stands for “independent” instead of ‘inexpensive’.
 - ◆ So RAID stands for **Redundant Arrays of Independent Disks**.
- RAID is arranged into six different levels.

RAID: Improvement of Reliability via Redundancy

- The chance that some disk out of N disk fail is much higher than single disk.
- Each failure of disk leads to loss of data
- Solution: Redundancy.
 - ◆ Store extra information which can be used in the event of failure.
- Simplest: Mirrored disks
 - ◆ Each logical disk consists of two physical disks.
 - ◆ Read from any disk
 - ◆ Write to both disks.

RAID: improvement in Performance via Parallelism.

- # of reads per unit time can be increased.
- With multiple disks we can improve the transfer rate with data stripping.
- Bit level Data stripping:
 - ◆ Splitting the bits of each byte across multiple disks.
 - ◆ If we have array of 8 disks, we write bit i of every byte to disk i.
 - ◆ The array of eight disks can be treated as single disk that are eight times normal size and eight times the access rate.
 - ◆ Every disk participates in the read.
 - ◆ But each access can read eight times as many data.
- Block level stripping: Blocks of files are stripped across multiple disks.
- Two goals
 - ◆ Increase the throughput of multiple small accesses.
 - ◆ Reduce the response time of large accesses.

Bit-level data stripping

- 0001010 0
- 0001110 1
- 0011001 1
- 0000110 0
-
■

Block-level stripping

- Block-level stripping is the logical extension of bit-level stripping.
- For example first block contains 1000 bytes. So it is a sequence of 7000 bits (ignoring 8th bit which is a parity bit). The first block is stored in Disk1. Similarly, other blocks are stored in Disk2, Disk3,..., Disk7 respectively. In Disk8, the parity of D1 to D7 blocks are stored.
- Disk1: 7000 bits of block1
- Disk2: 7000 bits of block2
- Disk3: 7000 bits of block3
- .
- .
- Disk7= 7000 bits of block7
- D8= 7000 bit parity (parity of 7 bits (combination of first bit of each block)).

- So if any of the disk fails, we can easily recover the failed data by accessing other 7 disks.

RAID levels

- RAID level 0: Disk arrays with stripping at the level of blocks, but without any redundancy (no mirroring and no parity)
 - ◆ Block level stripping
- RAID level1: Disk mirroring.
 - ◆ Block level striping
- RAID level2:
 - ◆ Error detection with parity bits.
 - ◆ Error correcting stores two or more parity bits.
 - ◆ Data can be stripped among disks and parity bits are stored in other disks.
 - ◆ Bit level stripping

RAID levels

■ RAID level 3: bit-interleaved parity organization

- ◆ Disk controllers can detect whether a sector has been read correctly.
- ◆ The bits of failed sector can be recovered by computing the parity of the remaining bits.

■ RAID Level 3 is similar to RAID level 2 but less expensive

- ◆ One disk overhead.

■ RAID level 2 is not used in practice.

■ Advantages of RAID level 3 over level 1 (mirroring)

- ◆ One parity disk is needed; reducing the storage overhead
- ◆ Transfer rate is same.

■ RAID 3 performance problem

- ◆ Computing and writing parity

RAID levels

- RAID level 4: block-interleaved parity organization
 - ◆ Uses block-level stripping
 - ◆ Keeps parity block on separate disk
 - ◆ If one disk fails the parity block and other blocks can be used to recover the failed disk.
- A block read accesses only one disk
 - ◆ Data transfer rate for each process is slower, However multiple read requests can be carried out in parallel.
 - ◆ Write results into two writes: block and corresponding parity.
- RAID level 5: Block interleaved distributed parity
 - ◆ Parity is distributed among all $N+1$ disks.
 - ◆ For each block one disk stores parity and others store data.
- RAID level 6:
 - ◆ Similar to RAID 5, but stores extra redundant information to guard against multiple disk failures.
- RAID 0+1 and RAID 1+0
 - ◆ Combination of RAID levels 0 and 1

RAID Levels



(a) RAID 0: non-redundant striping



(b) RAID 1: mirrored disks



(c) RAID 2: memory-style error-correcting codes



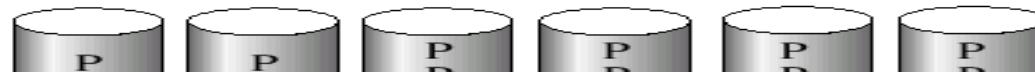
(d) RAID 3: bit-interleaved Parity



(e) RAID 4: block-interleaved parity

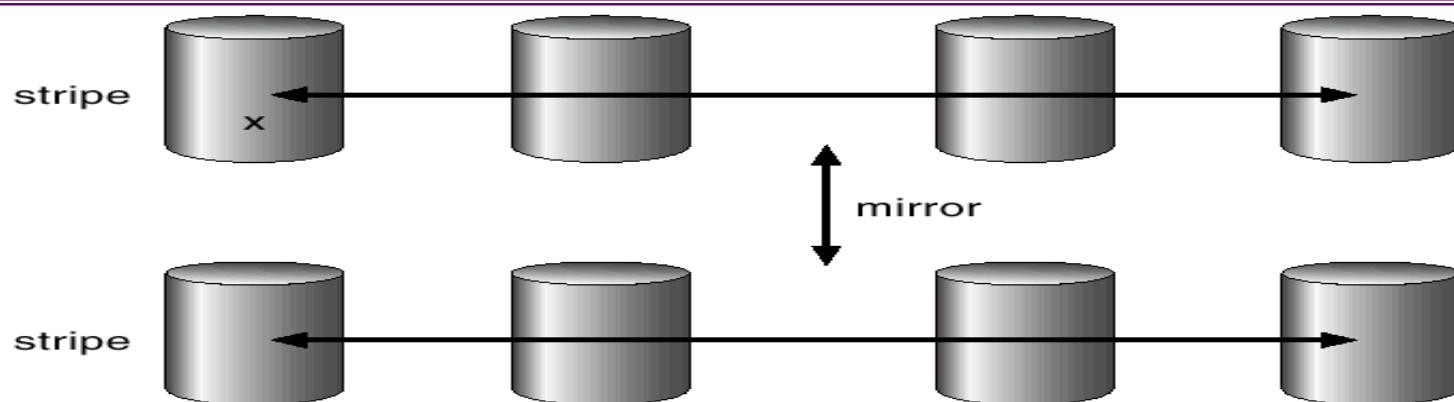


(f) RAID 5: block-Interleaved distributed parity

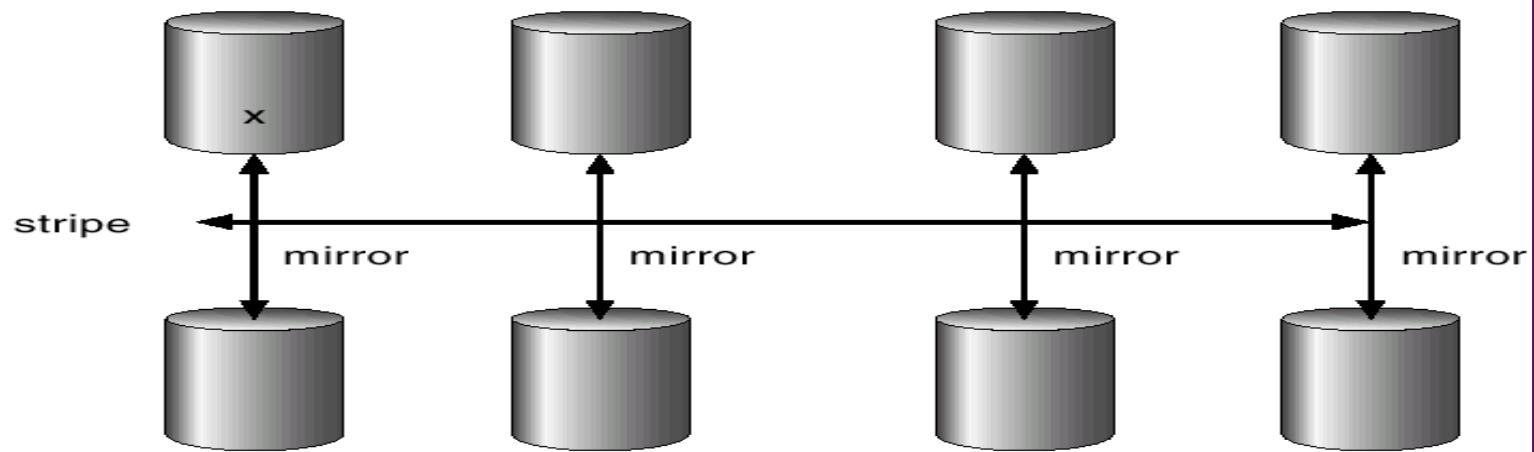


(g) RAID 6: P + Q redundancy

RAID (0 + 1) and (1 + 0)



a) RAID 0 + 1 with a single disk failure



b) RAID 1 + 0 with a single disk failure

Selecting a RAID level

- In case of a failure, the time to rebuild data vary with the RAID level.
- RAID level 0: used for high performance applications where data loss is not critical.
- RAID level 1: Popular for applications that require high reliability with fast recovery.
- RAID 0+1 and 1+0 are used where performance and reliability are important.
- RAID 1+0 fault tolerance is more
- RAID level 5 is used to store large volume of data.
- RAID level 6 is not supported.
- Hot spare disks can be used to reduce human intervention.

- The concepts of RAID can be generalized to other systems
 - ◆ Arrays of tapes
 - ✓ Recovery of damaged tape
 - ◆ Broadcast of data over wireless systems.
 - ✓ Receiver can reconstruct the information with parity information.

File System Implementation

Two aspects

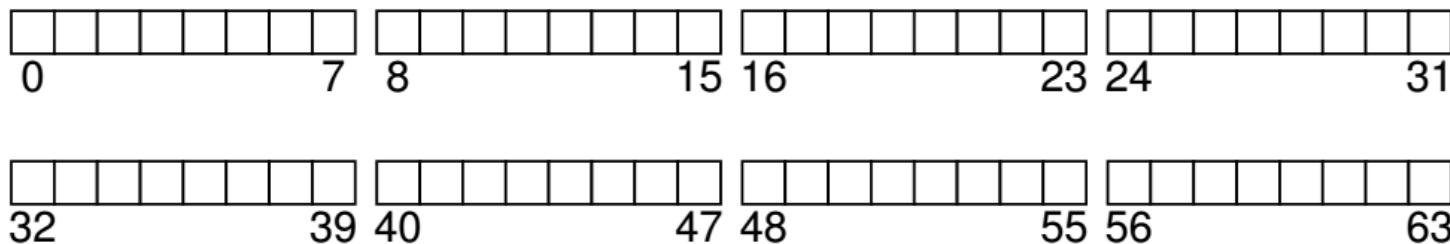
- Simplified version of a typical UNIX file system
 - Introduce some of the basic on-disk structures, access methods, and various policies that you will find in many file systems today.
 - Other file systems
 - AFS (the Andrew File System)
 - ZFS (Sun's Zettabyte File System)
- Two aspects
 - **data structures** of the file system.
 - What types of on-disk structures are utilized by the file system to organize its data and metadata?
 - **access methods**.
 - How does it map the calls made by a process, such as open(), read(), write(), etc., onto its structures?
 - Which structures are read during the execution of a particular system call? Which are written?
 - How efficiently are all of these steps performed?

Outline

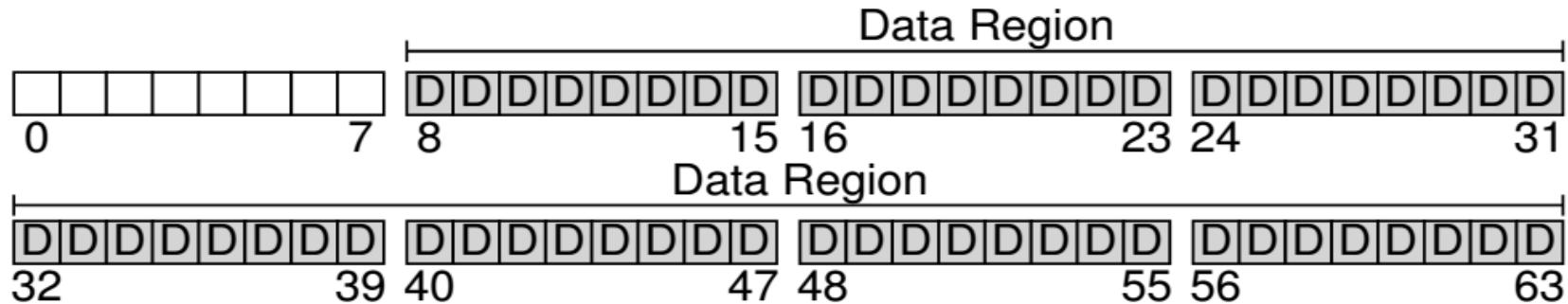
- Explain about simple file system
- i-node
- Directory organization
- Access paths

Overall Organization

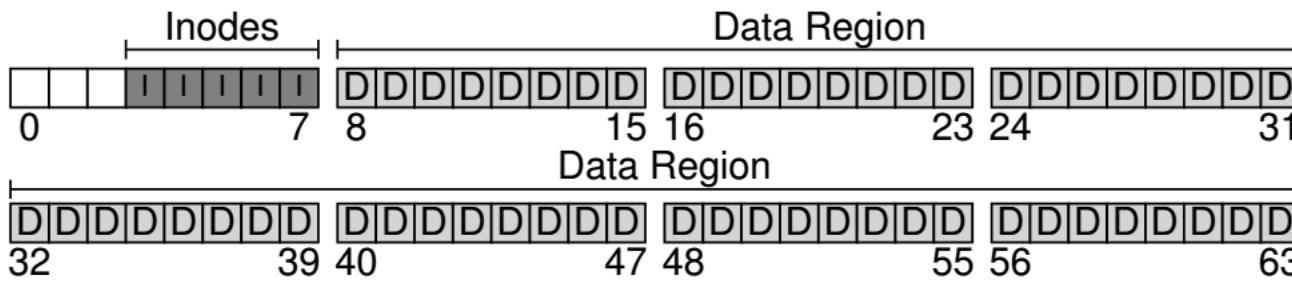
- Disk is divided into blocks
 - For example, the block size is 4k.
- Assume a small disk has 64 small blocks.



- We will allocate 56 blocks for data

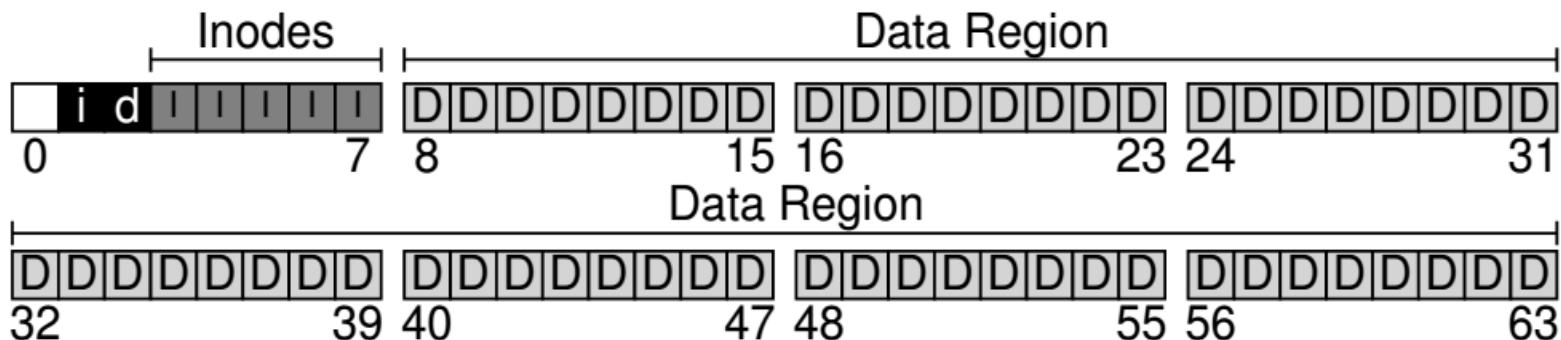


- File system has to track information about files
- To store this information i-nodes are used
- Five blocks are assigned to i-nodes.
- The inodes are typically not that big, for example 128 or 256 bytes.
- Assuming 256 bytes per inode, a 4-KB block can hold 16 inodes, and our file system above contains 80 total inodes.



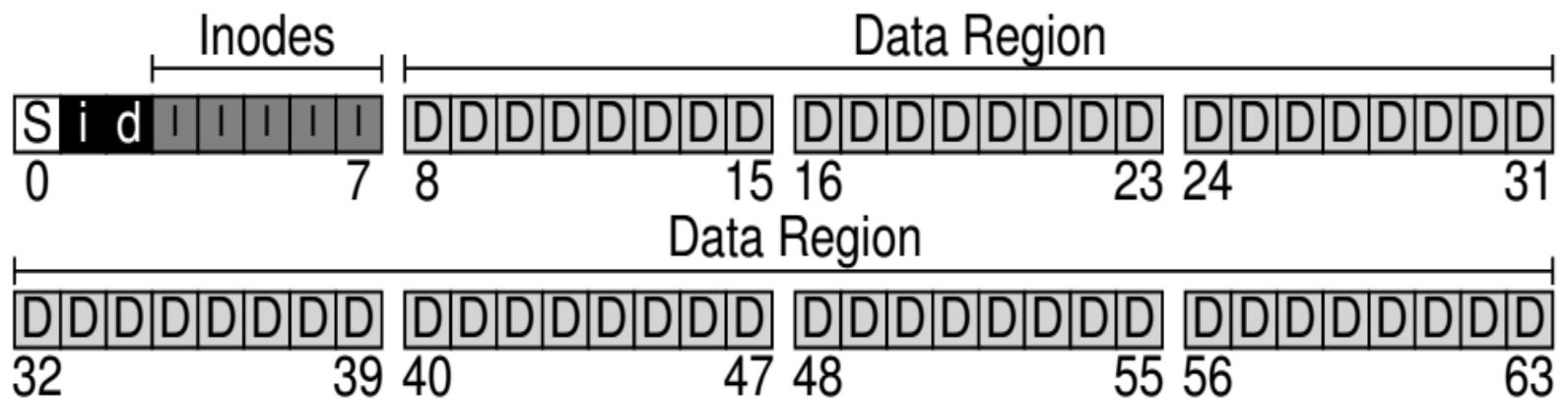
Tracking of Files: bit maps

- We have to track whether inodes or data blocks are free or allocated.
 - Bitmaps are used
 - One for data region and one for inode region



Super Block

- The superblock contains information about this particular file system, including, for example, how many inodes and data blocks are in the file system (80 and 56, respectively in this instance), where the inode table begins (block 3), and so forth. It will likely also include a magic number of some kind to identify the file system type (in this case, vsfs).



Mounting of a File System

- Reading of super block and initialize the parameters and attaching the volume to the file system tree

File Organization

- i-node is the short form of index node
- Each inode is implicitly referred to by a number (called the **inumber**), which we've earlier called the **low-level name** of the file.
- In vsfs (and other simple file systems), given an i-number, you should directly be able to calculate where on the disk the corresponding inode is located.

The Inode Table (Closeup)

| | | | | iblock 0 | iblock 1 | iblock 2 | iblock 3 | iblock 4 |
|-------|--------|--------|------|---|---|---|---|----------|
| Super | i-bmap | d-bmap | | 0 1 2 3 16 17 18 19 32 33 34 35 48 49 50 51 64 65 66 67 | 4 5 6 7 20 21 22 23 36 37 38 39 52 53 54 55 68 69 70 71 | 8 9 10 11 24 25 26 27 40 41 42 43 56 57 58 59 72 73 74 75 | 12 13 14 15 28 29 30 31 44 45 46 47 60 61 62 63 76 77 78 79 | |
| 0KB | 4KB | 8KB | 12KB | 16KB | 20KB | 24KB | 28KB | 32KB |

- To read inode number 32, the file system would first calculate the offset into the inode region ($32 \cdot \text{sizeof(inode)} \text{ or } 8192$), add it to the start address of the inode table on disk (inodeStartAddr = 12KB), and thus arrive upon the correct byte address of the desired block of inodes: 20KB.
- Recall that disks are not byte addressable, but rather consist of a large number of addressable sectors, usually 512 bytes. Thus, to fetch the block of inodes that contains inode 32, the file system would issue a read to sector $20 \times 1024 / 512$, or 40, to fetch the desired inode block. More generally, the sector address iaddr of the inode block can be calculated as follows:

```
blk = (inumber * sizeof(inode_t)) / blockSize;
sector = ((blk * blockSize) + inodeStartAddr) / sectorSize;
```

| Size | Name | What is this inode field for? |
|-------------|-------------|---|
| 2 | mode | can this file be read/written/executed? |
| 2 | uid | who owns this file? |
| 4 | size | how many bytes are in this file? |
| 4 | time | what time was this file last accessed? |
| 4 | ctime | what time was this file created? |
| 4 | mtime | what time was this file last modified? |
| 4 | dtime | what time was this inode deleted? |
| 2 | gid | which group does this file belong to? |
| 2 | links_count | how many hard links are there to this file? |
| 4 | blocks | how many blocks have been allocated to this file? |
| 4 | flags | how should ext2 use this inode? |
| 4 | osd1 | an OS-dependent field |
| 60 | block | a set of disk pointers (15 total) |
| 4 | generation | file version (used by NFS) |
| 4 | file_acl | a new permissions model beyond mode bits |
| 4 | dir_acl | called access control lists |

Figure 40.1: Simplified Ext2 Inode

Supporting Big files

- Multi-level index
- Instead of pointing to data block, i-node points to block of pointers.
- Notion of extents are used
 - An extent is simply a disk pointer plus a length (in blocks); thus, instead of requiring a pointer for every block of a file, all one needs is a pointer and a length to specify the on-disk location of a file.
- Link based approaches

Directory Organization

- A directory has list of pairs (entry name and i-node number)
- File systems treat directories as special type of file.

For example, assume a directory `dir` (inode number 5) has three files in it (`foo`, `bar`, and `foobar`), and their inode numbers are 12, 13, and 24 respectively. The on-disk data for `dir` might look like this:

| inum | recflen | strlen | name |
|------|---------|--------|--------|
| 5 | 4 | 2 | . |
| 2 | 4 | 3 | .. |
| 12 | 4 | 4 | foo |
| 13 | 4 | 4 | bar |
| 24 | 8 | 7 | foobar |

Access Paths: Reading and Writing

- Reading and writing require several I/Os

| | data bitmap | inode bitmap | root inode | foo inode | bar inode | root data | foo data | bar data[0] | bar data[1] | bar data[2] |
|-----------|----------------|-----------------|---------------|--------------|--------------|--------------|-------------|----------------|----------------|----------------|
| open(bar) | | | read | | | | read | | | |
| | | | | read | | | | read | | |
| | | | | | read | | | | read | |
| read() | | | | | read | | | | read | |
| | | | | | | write | | | | |
| | | | | | read | | | | | |
| read() | | | | | | | read | | | |
| | | | | | | write | | | | |
| | | | | | read | | | | | |
| read() | | | | | | | | read | | |
| | | | | | | write | | | | |

Figure 40.3: File Read Timeline (Time Increasing Downward)

| | data bitmap | inode bitmap | root inode | foo inode | bar inode | root data | foo data | bar data[0] | bar data[1] | bar data[2] |
|----------------------|----------------|-----------------|---------------|--------------|---------------|--------------|-------------|----------------|----------------|----------------|
| create (/foo/bar) | | read write | read | read | | read | read | | | |
| | | | | | read write | | | write | | |
| | | | | | write | read | | | | |
| write() | | read write | | | | | | write | | |
| | | | | | write | read | | | | |
| | | | | | | read | | | write | |
| write() | | read write | | | | | | | write | |
| | | | | | write | read | | | | |
| | | | | | | read | | | | write |
| write() | | read write | | | | | | | | write |
| | | | | | write | | | | | |

Figure 40.4: File Creation Timeline (Time Increasing Downward)

THE CRUX: HOW TO REDUCE FILE SYSTEM I/O COSTS

Even the simplest of operations like opening, reading, or writing a file incurs a huge number of I/O operations, scattered over the disk. What can a file system do to reduce the high costs of doing so many I/Os?

Techniques: Caching and buffering

- Use caching or DRAM to cache important blocks

Topics of this Lecture

■ Overview

- File system interface
- File system implementation
- Multimedia systems
- Protection
- Security

File System

■ Objectives

- Function of file systems
- Describe the interfaces to file systems
- Discuss design trade-offs
- Explore file system protection

File Concept

- Contiguous logical address space
- File is an abstraction by OS.
- These are mapped to physical devices

- Types:
 - Data
 - ▶ numeric
 - ▶ character
 - ▶ binary
 - Program

File Structure

- None - sequence of words, bytes
- Simple record structure
 - Lines
 - Fixed length
 - Variable length
- Complex Structures
 - Formatted document
 - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters
- Who decides:
 - Operating system
 - Program

Common terms related to files

■ Common terms

- Field is the basic element of data
- A record is collection of related fields
- A file is a collection of similar records
- A database is a collection of related data/files

■ Typical operations to be supported

- Retrieve_all: Retrieve all the records in the file
- Retrieve_one: Retrieve one record
- Retrieve_next: Retrieve next
- Retrieve_previous: Retrieve previous
- Insert_One
- Delete_One
- Update_one
- Retrieve_few

File Management Systems (FMS)

- A file management system is the set of system software that provides services and applications in the use of files. Users can access files only through FMS.
- This relieves the programmer of necessity of developing special-purpose software for each application.
- Objectives of FMS
 - To meet the data management needs and requirements of the user: storage of data and ability to perform operations
 - Optimize the performance regarding throughput and response time
 - To provide I/O support for variety of storage devices
 - To prevent the potential lost of destroyed data
 - To provide I/O support for multiple users.

Minimal set of requirements of FMS

- Each user should be able to create, delete, read, change files
- Each user may have controlled access to other user's files
- Each user should control access to his/her files
- Each user should be able to restructure the files
- Each user should be able to move data between files.
- Each user should be able to backup and recover the user's files in case of damage
- Each user should be able to access the files using symbolic names.

File Attributes

- **Name** – only information kept in human-readable form
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on the disk

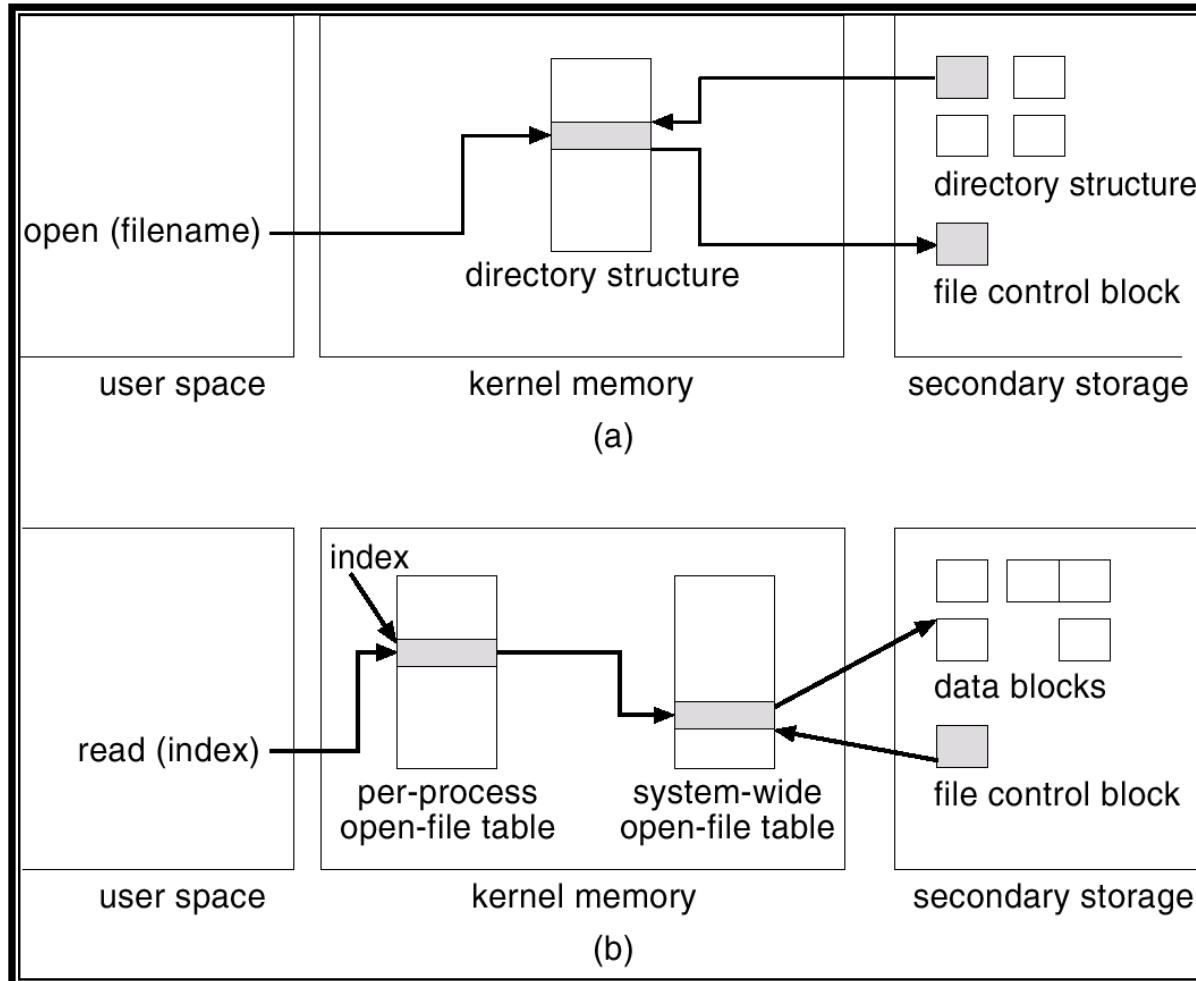
File Operations

- A file is a abstract data type.
- File operations are as follows
 - Create
 - Write : File position pointer
 - Read
 - file seek – reposition the pointer within file
 - Delete: Search a directory
 - Truncate: Keep the attributes and delete the content.
 - Open(F_i) – search the directory structure on disk for entry F_i , and move the content of entry to memory
 - Close (F_i) – move the content of entry F_i in memory to directory structure on disk

Open Files

- The OS keeps a small table “Open-file table” containing information about all open files
- The open() system call returns a pointer to the entry in Open-file table.
- OS maintains two kinds of internal tables
 - A per-process table: tracks all the files the process has open
 - ▶ Current pointer is found here.
 - A system-wide table: An entry in per-process tables points to a system-wide open-file table.
 - ▶ The system-wide table contains process-independent information
 - Location of file, access dates and file size.
- Several pieces of data are needed to manage open files:
 - File pointer: pointer to last read/write location, per process that has the file open
 - File-open count: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
 - Disk location of the file: cache of data access information
 - Access rights: per-process access mode information

In-Memory File System Structures



(a) File open (b) File read

Open File Locking

- Provided by some operating systems and file systems
- Mediates access to a file
- Similar to reader-writer locks
- A **shared lock** is similar to reader lock
- An **exclusive lock** is similar to writer lock.
- Programmers should be careful in holding exclusive locks for a longer times.
- Mandatory or advisory:
 - **Mandatory** – access is denied depending on locks held and requested
 - **Advisory** – processes can find status of locks and decide what to do

File Types – Name, Extension

| file type | usual extension | function |
|----------------|--------------------------------|---|
| executable | exe, com, bin or none | read to run machine-language program |
| object | obj, o | compiled, machine language, not linked |
| source code | c, cc, java, pas, asm, a | source code in various languages |
| batch | bat, sh | commands to the command interpreter |
| text | txt, doc | textual data, documents |
| word processor | wp, tex, rrf, doc | various word-processor formats |
| library | lib, a, so, dll, mpeg, mov, rm | libraries of routines for programmers |
| print or view | arc, zip, tar | ASCII or binary file in a format for printing or viewing |
| archive | arc, zip, tar | related files grouped into one file, sometimes compressed, for archiving or storage |
| multimedia | mpeg, mov, rm | binary file containing audio or A/V information |

File structures

- How many file structures should be supported by OS ?
 - More file structures implies more complexity
- Suppose if OS supports five file structures, code should be provided for all.
- UNIX considers file as a sequence of 8-bit bytes.

Access Methods

■ Sequential Access

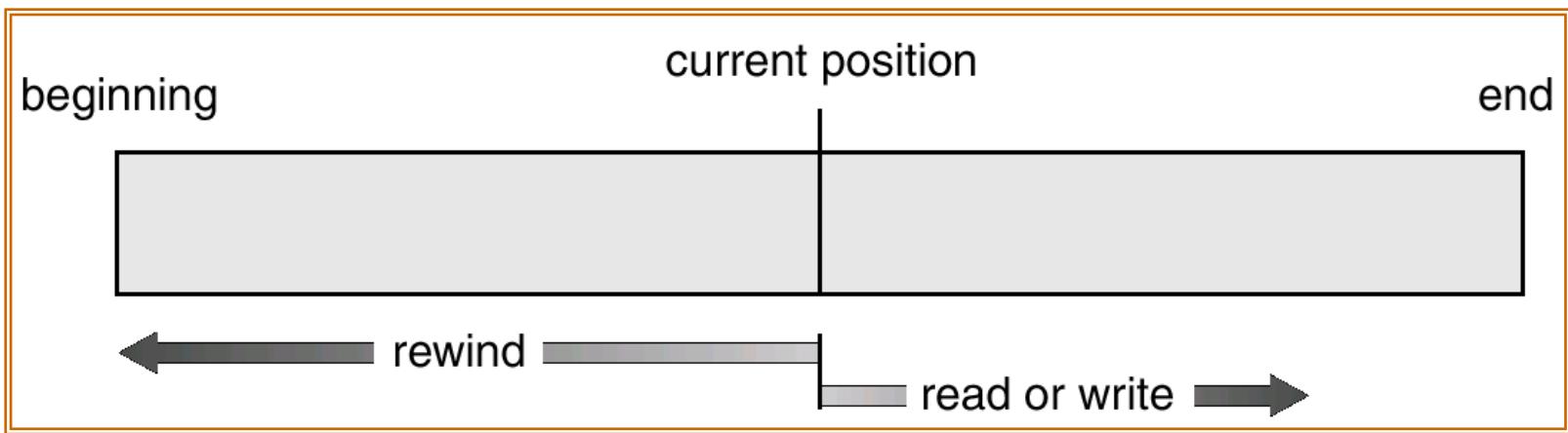
read next
write next
reset
no read after last write
(rewrite)

■ Direct Access

read n
write n
position to n
read next
write next
rewrite n

n = relative block number

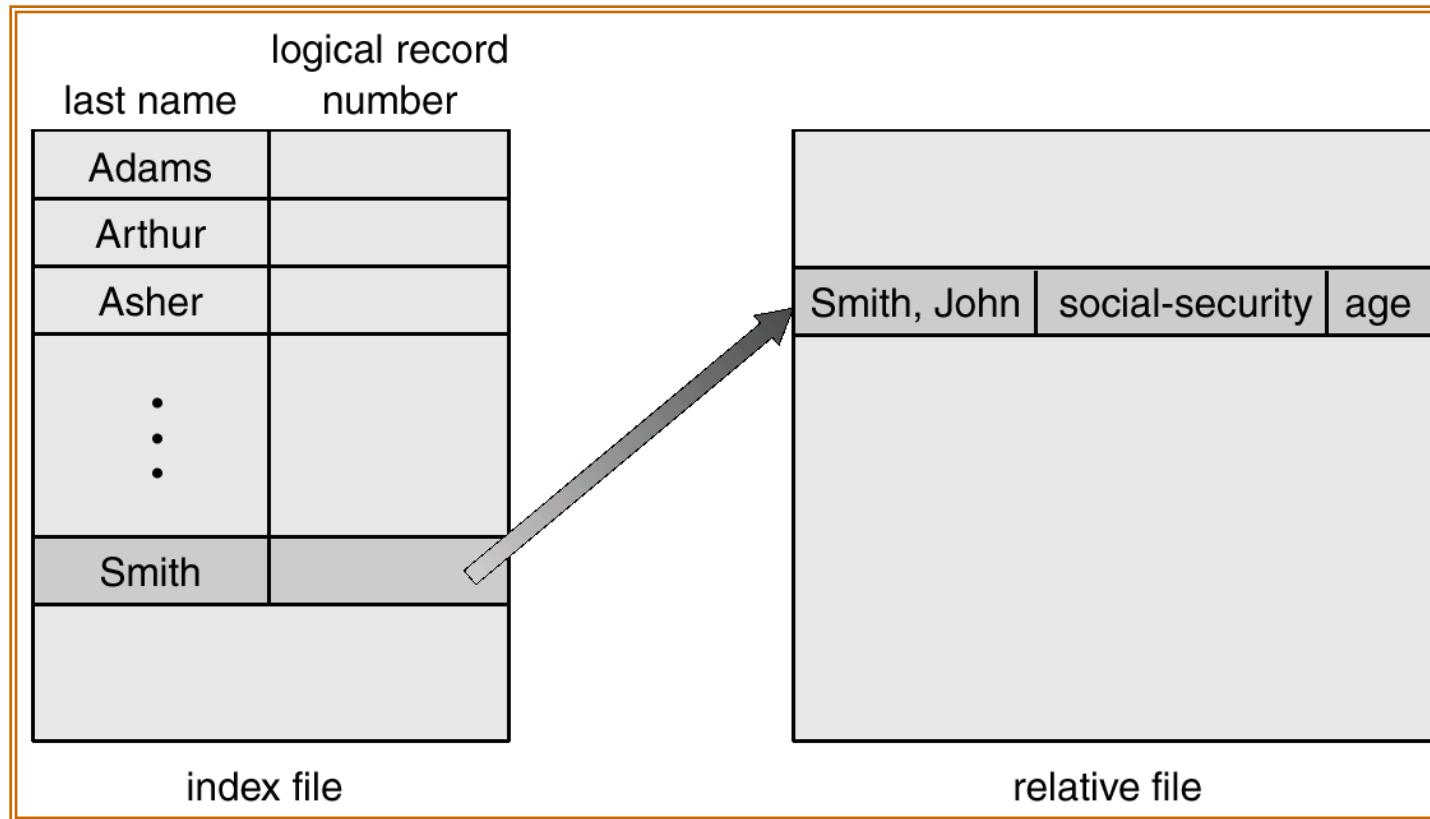
Sequential-access File



Simulation of Sequential Access on a Direct-access File

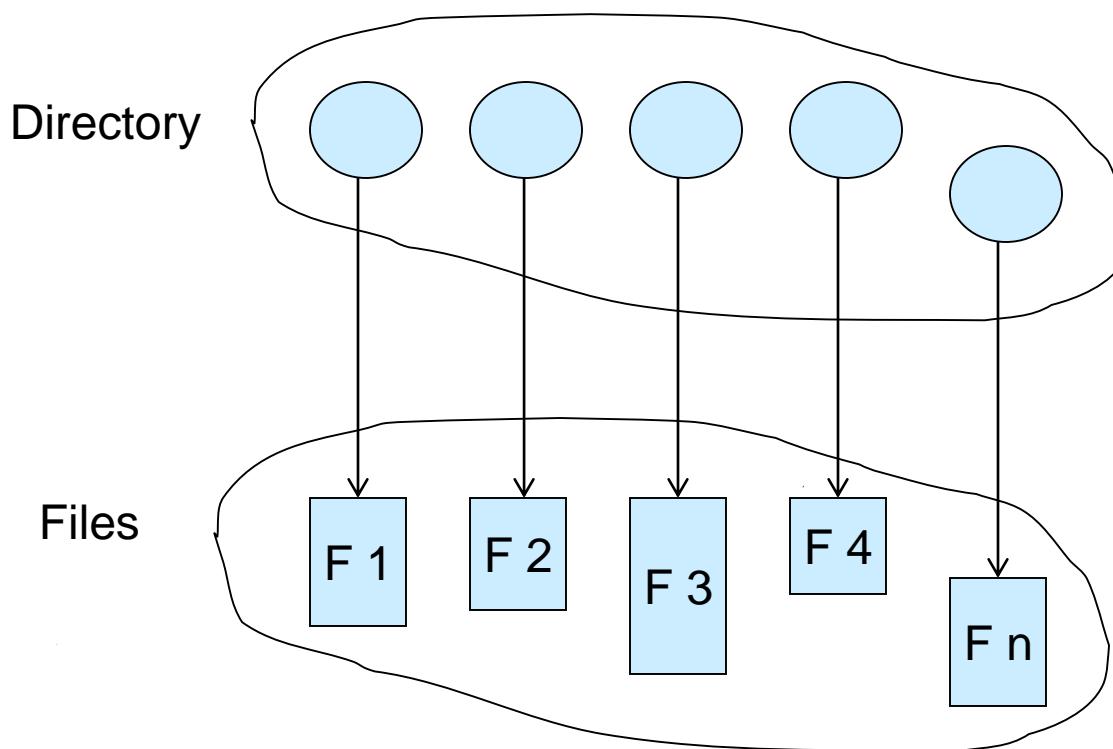
| sequential access | implementation for direct access |
|-------------------|----------------------------------|
| <i>reset</i> | $cp = 0;$ |
| <i>read next</i> | $read cp;$ $cp = cp+1;$ |
| <i>write next</i> | $write cp;$ $cp = cp+1;$ |

Example of Index and Relative Files



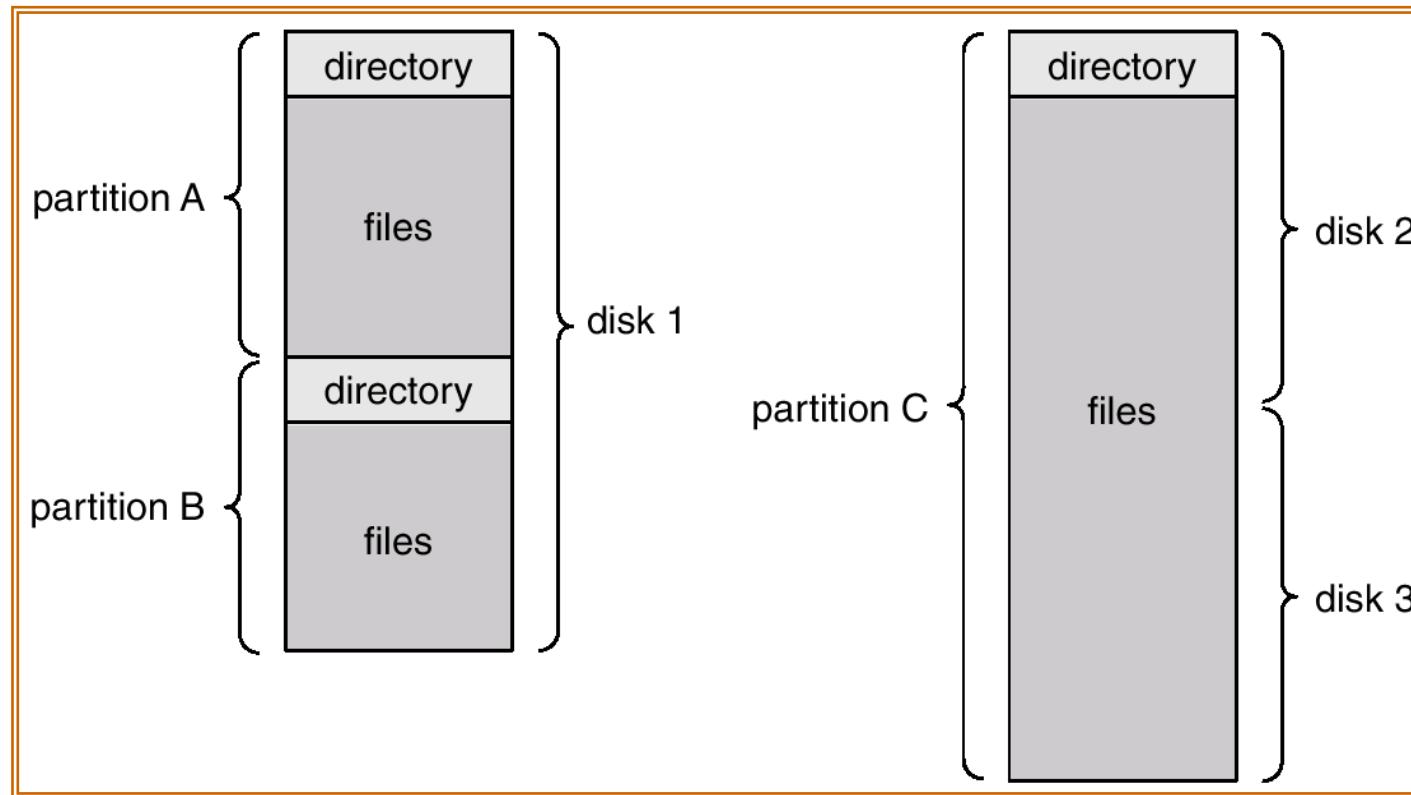
Directory Structure

- A collection of nodes containing information about all files



Both the directory structure and the files reside on disk
Backups of these two structures are kept on tapes

A Typical File-system Organization



Information in a Device Directory

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed (for archival)
- Date last updated (for dump)
- Owner ID
- Protection information (discuss later)

Operations Performed on Directory

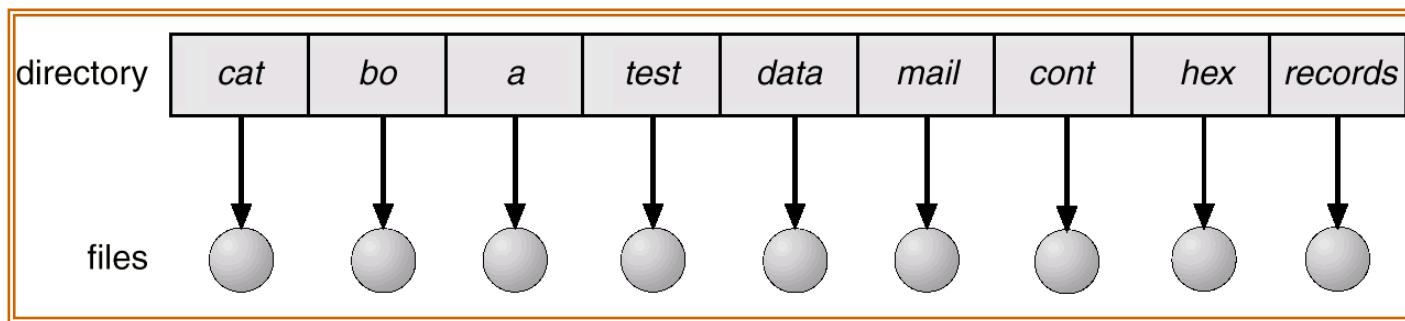
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Organize the Directory (Logically) to Obtain

- Efficiency – locating a file quickly
- Naming – convenient to users
 - Two users can have same name for different files
 - The same file can have several different names
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, ...)

Single-Level Directory

- A single directory for all users

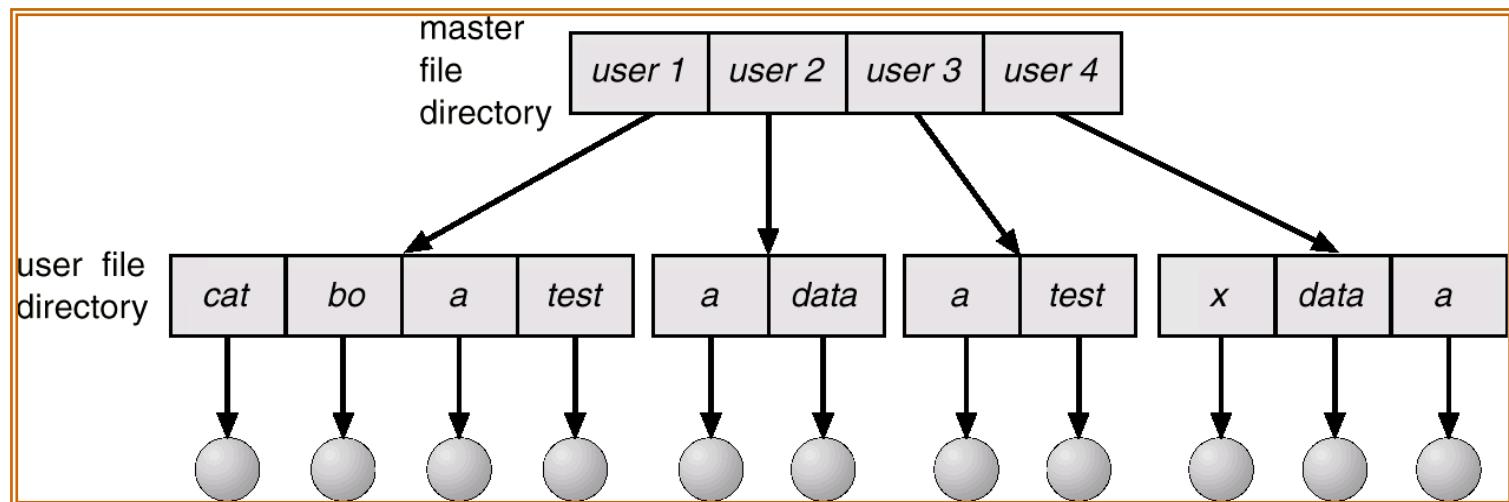


Naming problem

Grouping problem

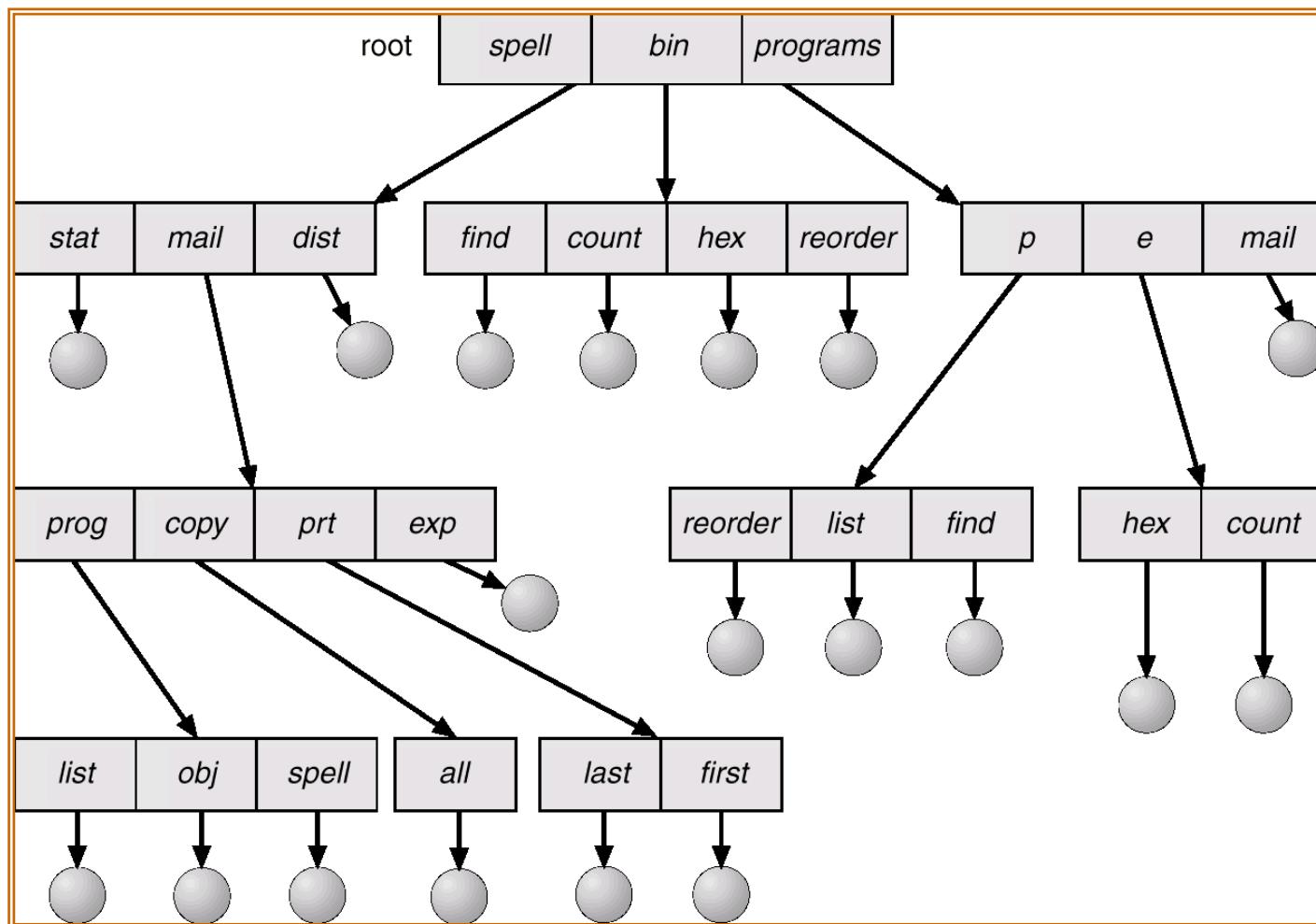
Two-Level Directory

- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories



Tree-Structured Directories (Cont)

- Efficient searching
- Grouping Capability
- Current directory (working directory)
 - `cd /spell/mail/prog`
 - `type list`

Tree-Structured Directories (Cont)

- **Absolute** or **relative** path name
- Creating a new file is done in current directory
- Delete a file

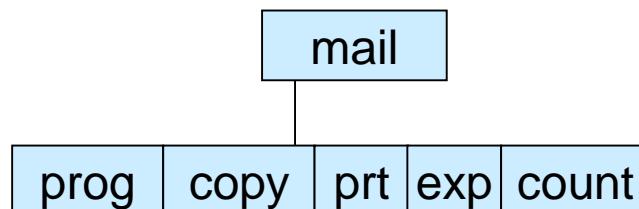
rm <file-name>

- Creating a new subdirectory is done in current directory

mkdir <dir-name>

Example: if in current directory /mail

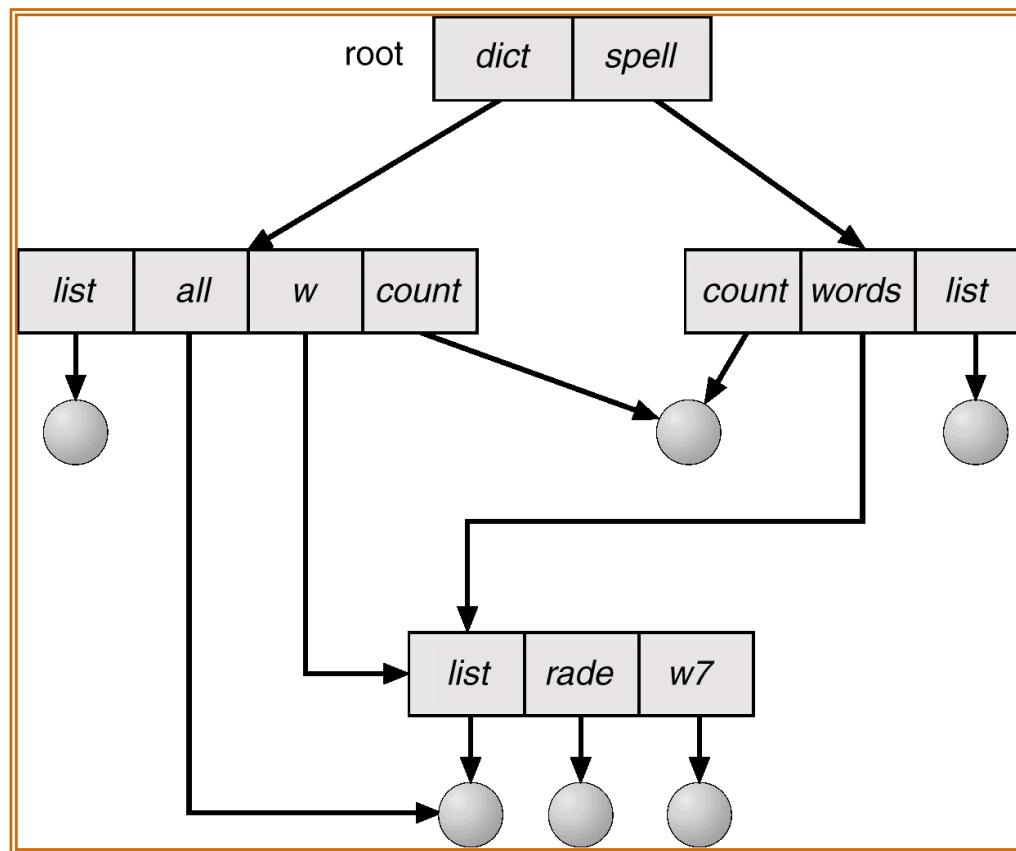
mkdir count



Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”

Acyclic-Graph Directories

- Have shared subdirectories and files



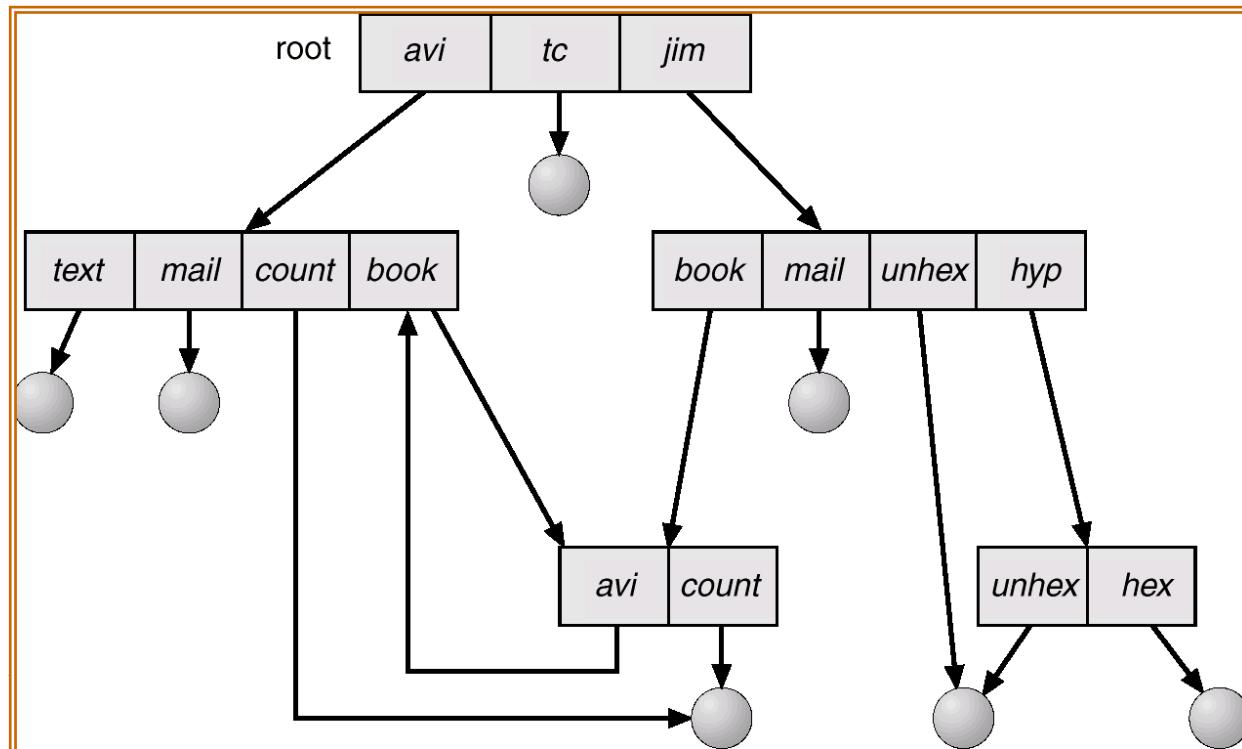
Acyclic-Graph Directories (Cont.)

- Two different names (aliasing)
- If *dict* deletes *list* \Rightarrow dangling pointer

Solutions:

- Backpointers, so we can delete all pointers
Variable size records a problem
- Backpointers using a daisy chain organization
- Entry-hold-count solution

General Graph Directory



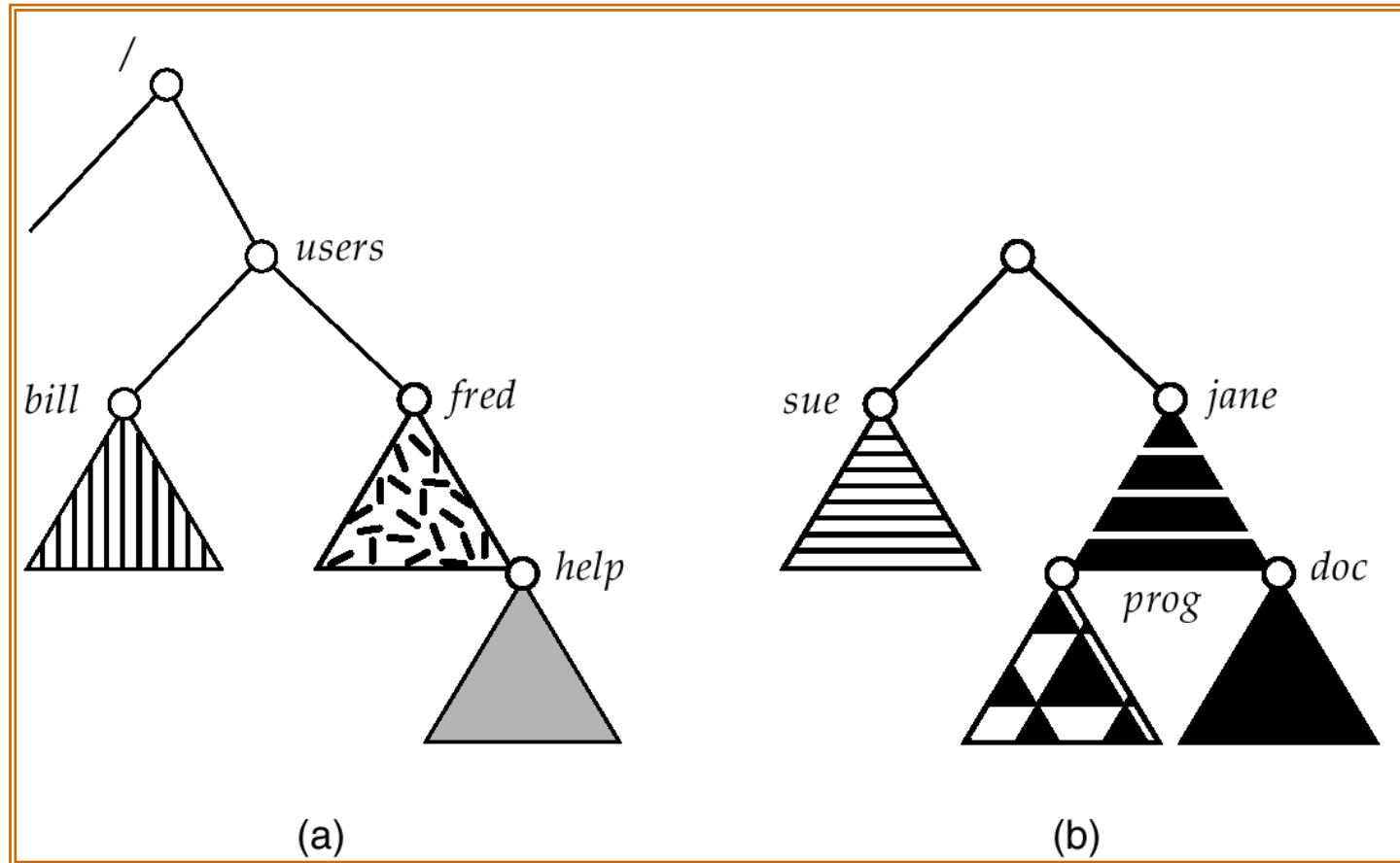
General Graph Directory (Cont.)

- How do we guarantee no cycles?
 - Allow only links to file not subdirectories
 - Garbage collection
 - Every time a new link is added use a cycle detection algorithm to determine whether it is OK

File System Mounting

- A file system must be **mounted** before it can be accessed
- An unmounted file system (i.e. Fig. 11-11(b)) is mounted at a **mount point**

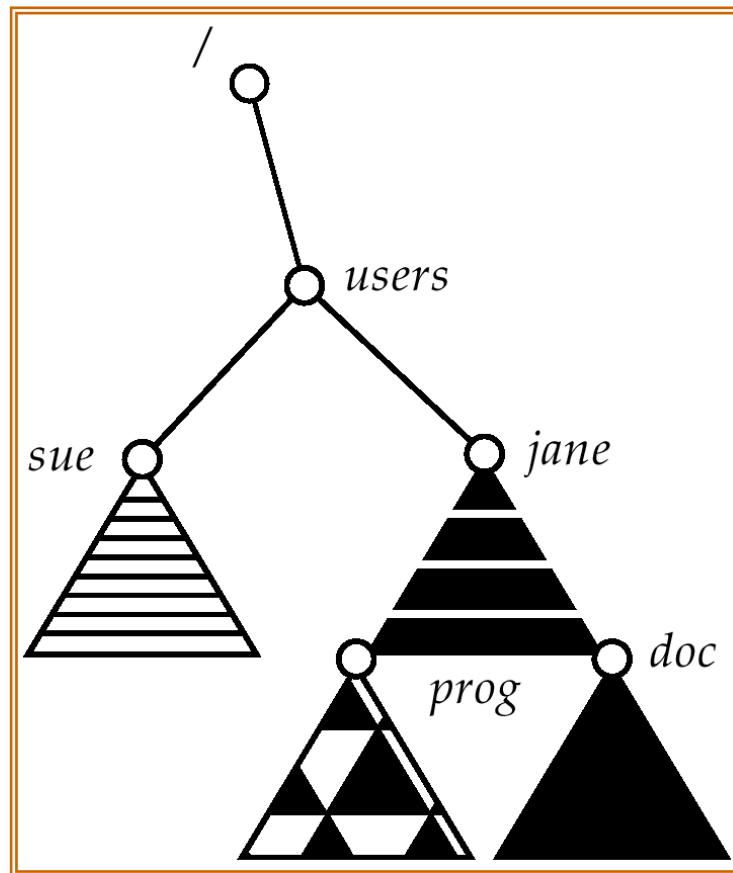
(a) Existing. (b) Unmounted Partition



(a)

(b)

Mount Point



File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method

File Sharing – Multiple Users

- **User IDs** identify users, allowing permissions and protections to be per-user
- **Group IDs** allow users to be in groups, permitting group access rights

File Sharing – Remote File Systems

- Uses networking to allow file system access between systems
 - Manually via programs like FTP
 - Automatically, seamlessly using **distributed file systems**
 - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers
 - Server can serve multiple clients
 - Client and user-on-client identification is insecure or complicated
 - **NFS** is standard UNIX client-server file sharing protocol
 - **CIFS** is standard Windows protocol
 - Standard operating system file calls are translated into remote calls
- Distributed Information Systems (**distributed naming services**) such as LDAP, DNS, NIS implement unified access to information needed for remote computing

File Sharing – Failure Modes

- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve state information about status of each remote request
- Stateless protocols such as NFS include all information in each request, allowing easy recovery but less security

File Sharing – Consistency Semantics

- **Consistency semantics** specify how multiple users are to access a shared file simultaneously
 - Similar to process synchronization algorithms
 - ▶ Tend to be less complex due to disk I/O and network latency (for remote file systems)
 - Andrew File System (AFS) implemented complex remote file sharing semantics
 - Unix file system (UFS) implements:
 - ▶ Writes to an open file visible immediately to other users of the same open file
 - ▶ Sharing file pointer to allow multiple users to read and write concurrently
 - AFS has session semantics
 - ▶ Writes to an open file by a user are not visible immediately to other users that have the same file open.
 - ▶ Writes only visible to sessions starting after the file is closed

Protection

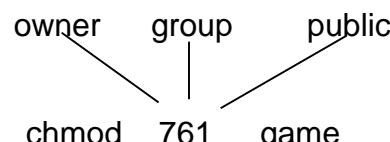
- File owner/creator should be able to control:
 - what can be done
 - by whom
- Types of access
 - Read
 - Write
 - Execute
 - Append
 - Delete
 - List

Access Lists and Groups

- Mode of access: read, write, execute
- Three classes of users

| | | | |
|------------------|---|---|-------|
| | | | RWX |
| a) owner access | 7 | ⇒ | 1 1 1 |
| | | | RWX |
| b) group access | 6 | ⇒ | 1 1 0 |
| | | | RWX |
| c) public access | 1 | ⇒ | 0 0 1 |

- Ask manager to create a group (unique name), say G, and add some users to the group.
- For a particular file (say *game*) or subdirectory, define an appropriate access.



Attach a group to a file

chgrp G game

File System Implementation

- File System Structure
- File System Implementation
- Directory Implementation
- Allocation Methods
- Free-Space Management
- Efficiency and Performance
- Recovery
- Log-Structured File Systems
- NFS

File-System Structure

■ File structure

- ◆ Logical storage unit
- ◆ Collection of related information

■ File system resides on secondary storage (disks).

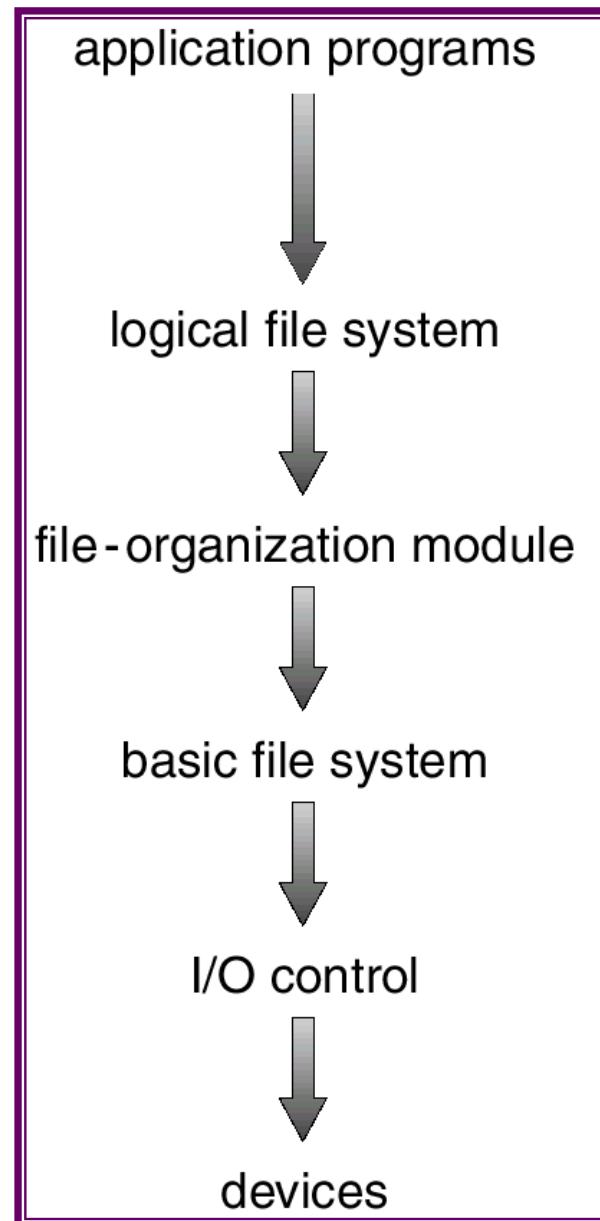
■ File system organized into layers.

■ *File control block* – storage structure consisting of information about a file.

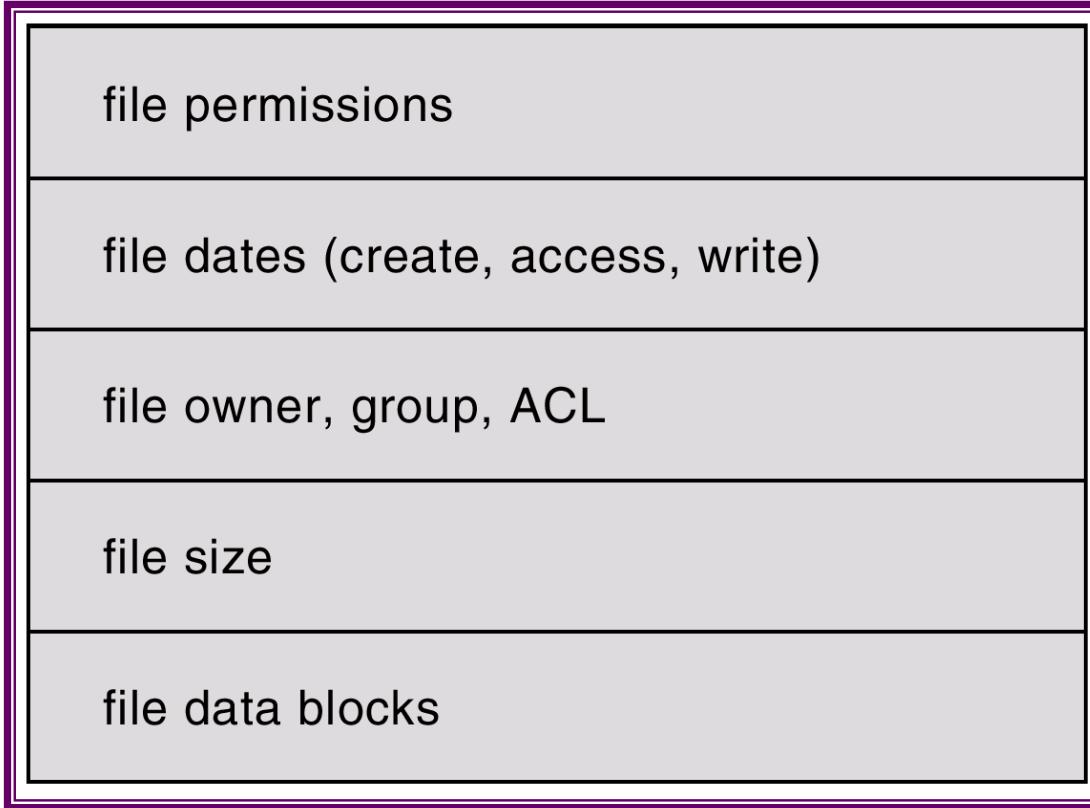
Layered file system

- Device drivers: transfer information from main memory and disk system
- Device driver writes data into I/O controller's memory
- The basic file system issues generic commands to appropriate device drivers.
- The file organization module knows about files and logical blocks.
 - ◆ It can translate logical block addresses to physical block addresses.
- The logical file system manages meta-information.
 - ◆ File system structure (excludes contents)
 - ◆ Manages directory structure
 - ◆ It maintains file structure via file control blocks (FCBs).
 - ✓ Ownership, permissions, ACL (access control list) and location of the contents.

Layered File System



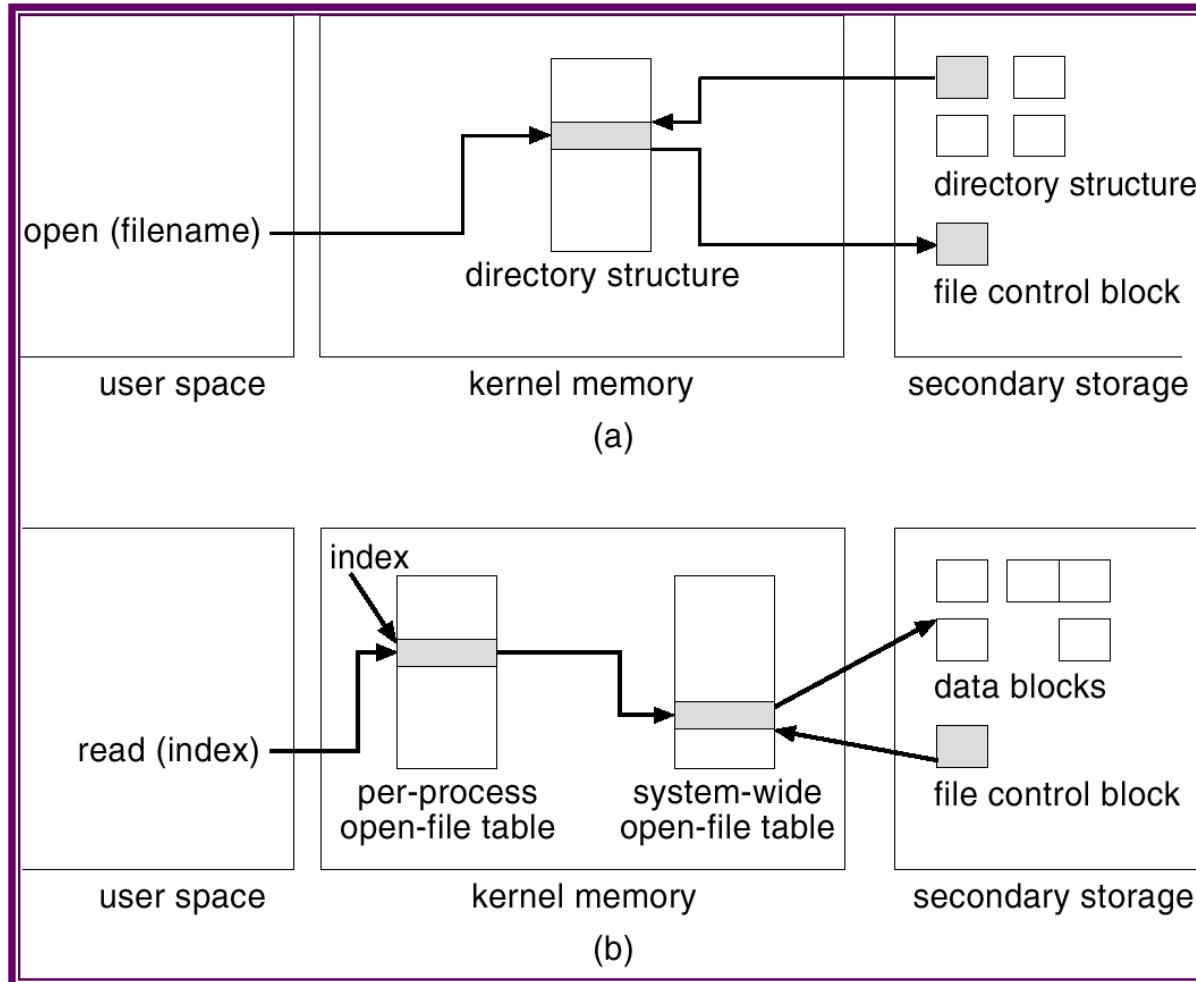
A Typical File Control Block



In-Memory File System Structures

- The following figure illustrates the necessary file system structures provided by the operating systems.
- Figure 12-3(a) refers to opening a file.
- Figure 12-3(b) refers to reading a file.

In-Memory File System Structures

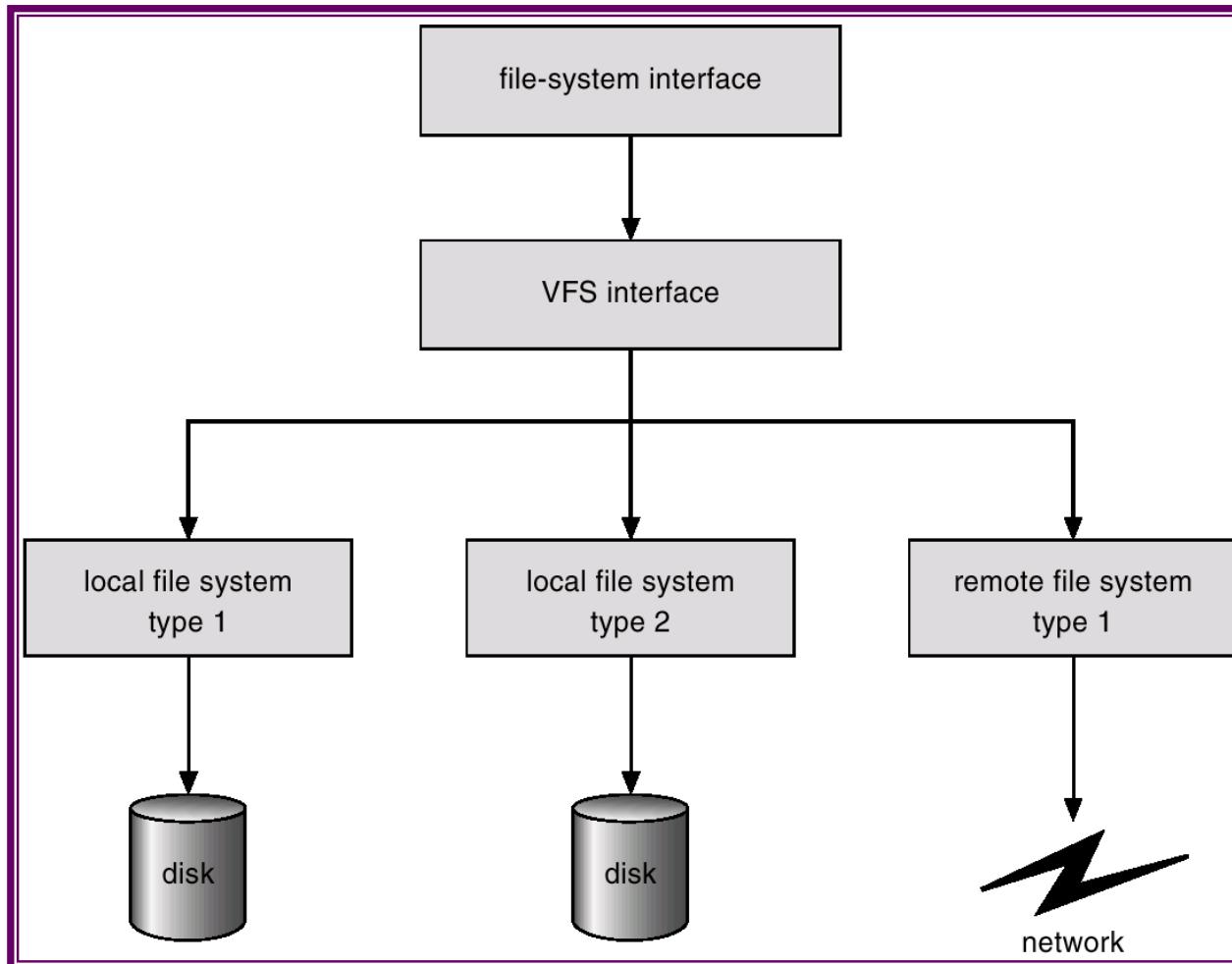


(a) File open (b) File read

Virtual File Systems

- Virtual File Systems (VFS) provide an object-oriented way of implementing file systems.
- VFS allows the same system call interface (the API) to be used for different types of file systems.
- The API is to the VFS interface, rather than any specific type of file system.

Schematic View of Virtual File System



Directory Implementation

■ Linear list of file names with pointer to the data blocks.

- ◆ simple to program
- ◆ time-consuming to execute

■ Hash Table – linear list with hash data structure.

- ◆ decreases directory search time
- ◆ *collisions* – situations where two file names hash to the same location
- ◆ fixed size

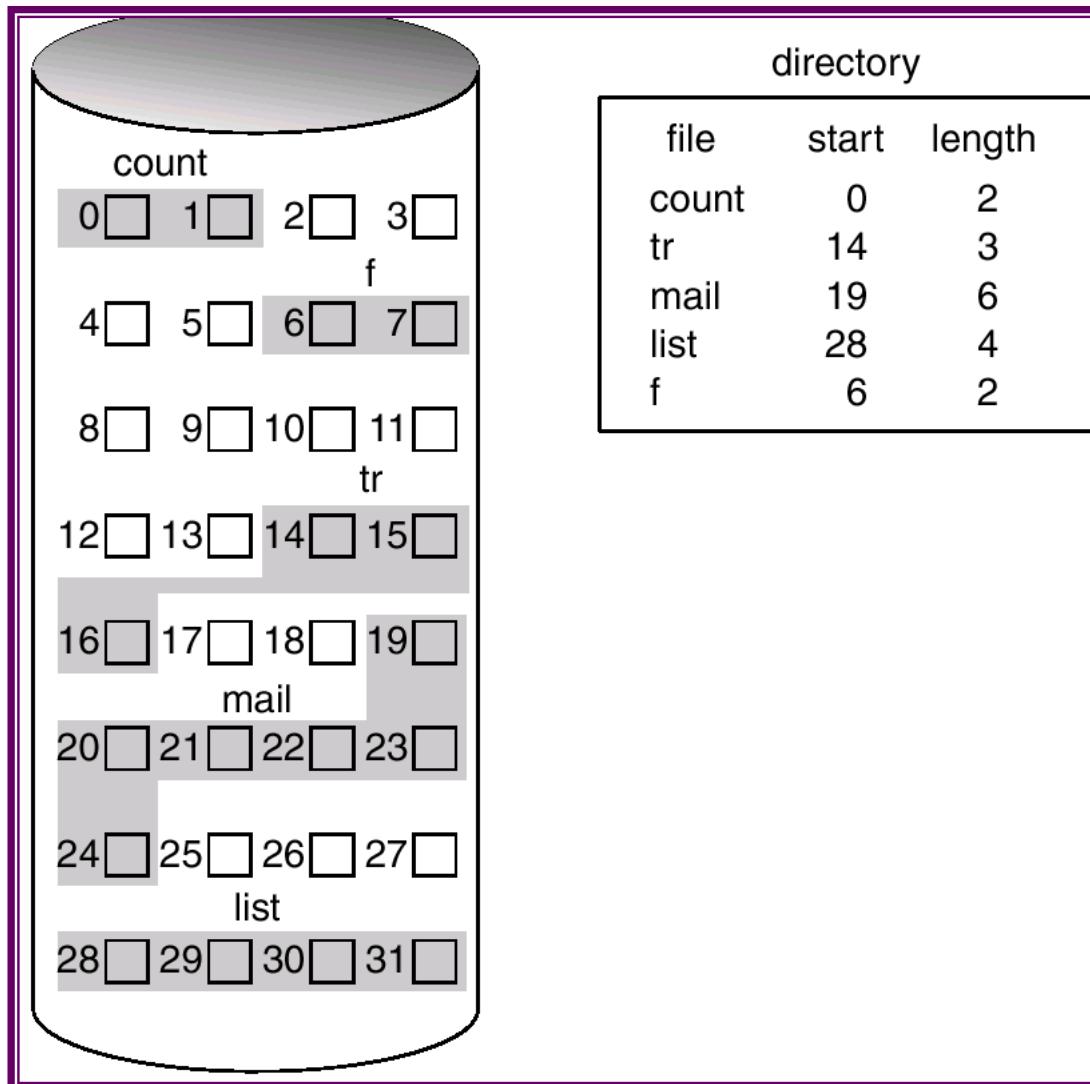
Allocation Methods

- An allocation method refers to how disk blocks are allocated for files:
- Contiguous allocation
- Linked allocation
- Indexed allocation

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk.
- Simple – only starting location (block #) and length (number of blocks) are required.
- Random access.
- Wasteful of space (dynamic storage-allocation problem).
- Files cannot grow.

Contiguous Allocation of Disk Space

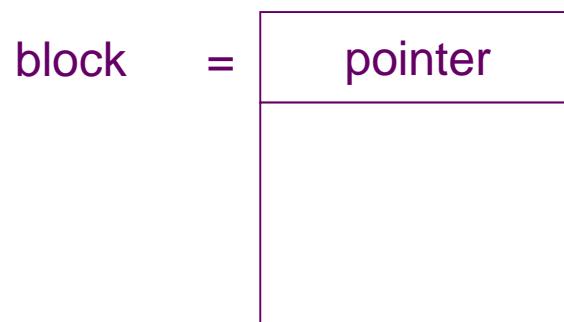


Extent-Based Systems

- Many newer file systems (I.e. Veritas File System) use a modified contiguous allocation scheme.
- Extent-based file systems allocate disk blocks in **extents**.
- An **extent** is a contiguous block of disks. Extents are allocated for file allocation. A file consists of one or more extents.

Linked Allocation

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk.



Linked Allocation (Cont.)

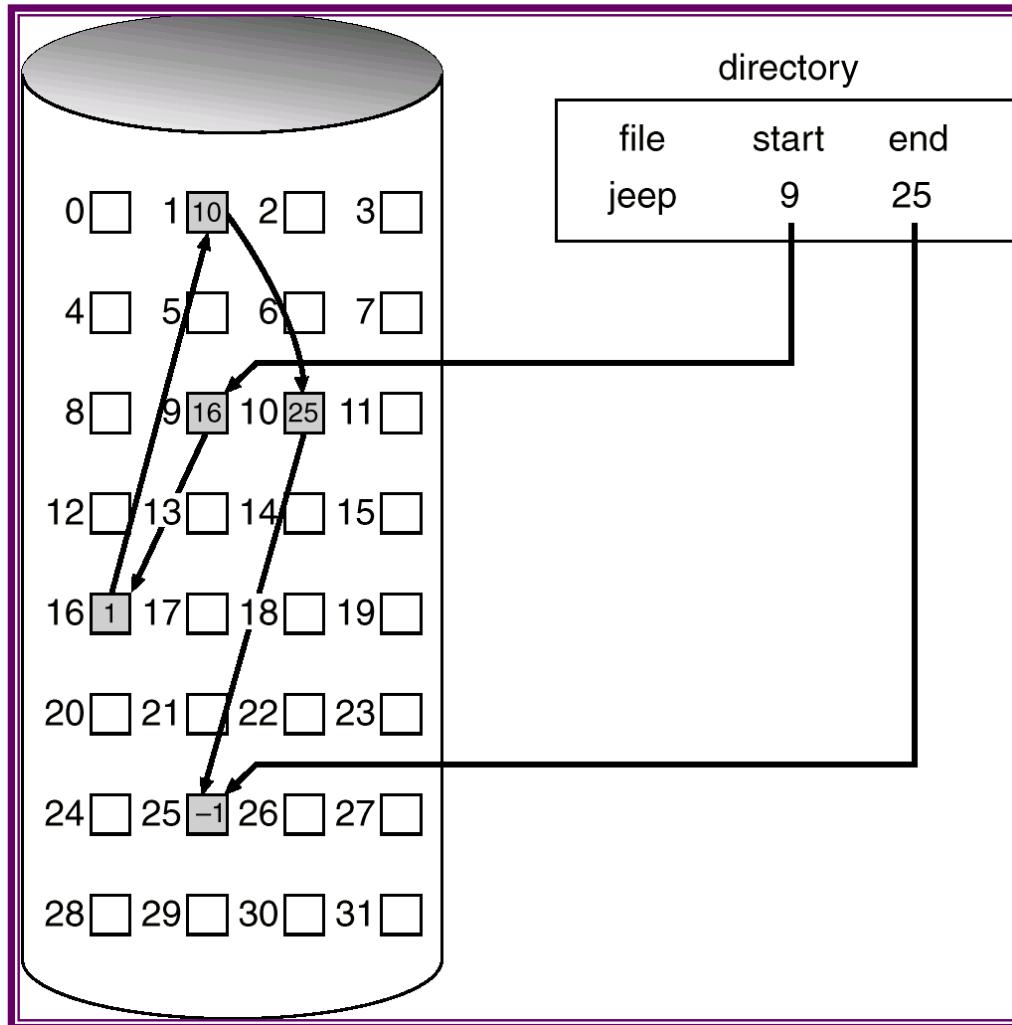
- Simple – need only starting address
- Free-space management system – no waste of space
- No random access
- Mapping

Block to be accessed is the Qth block in the linked chain of blocks representing the file.

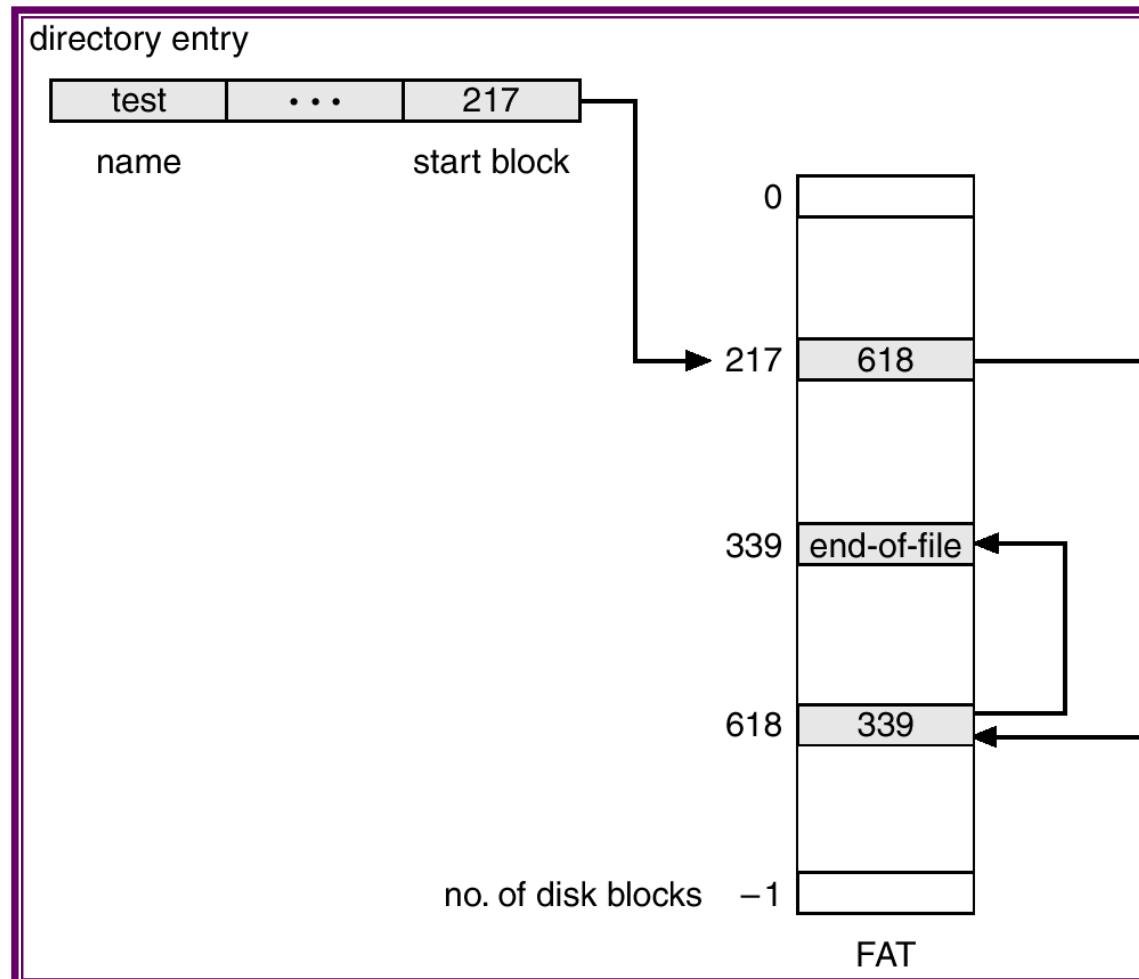
Displacement into block = R + 1

File-allocation table (FAT) – disk-space allocation used by MS-DOS and OS/2.

Linked Allocation

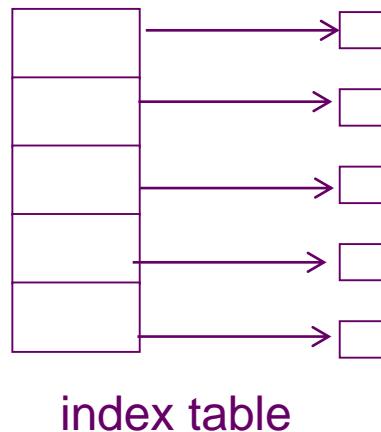


File-Allocation Table

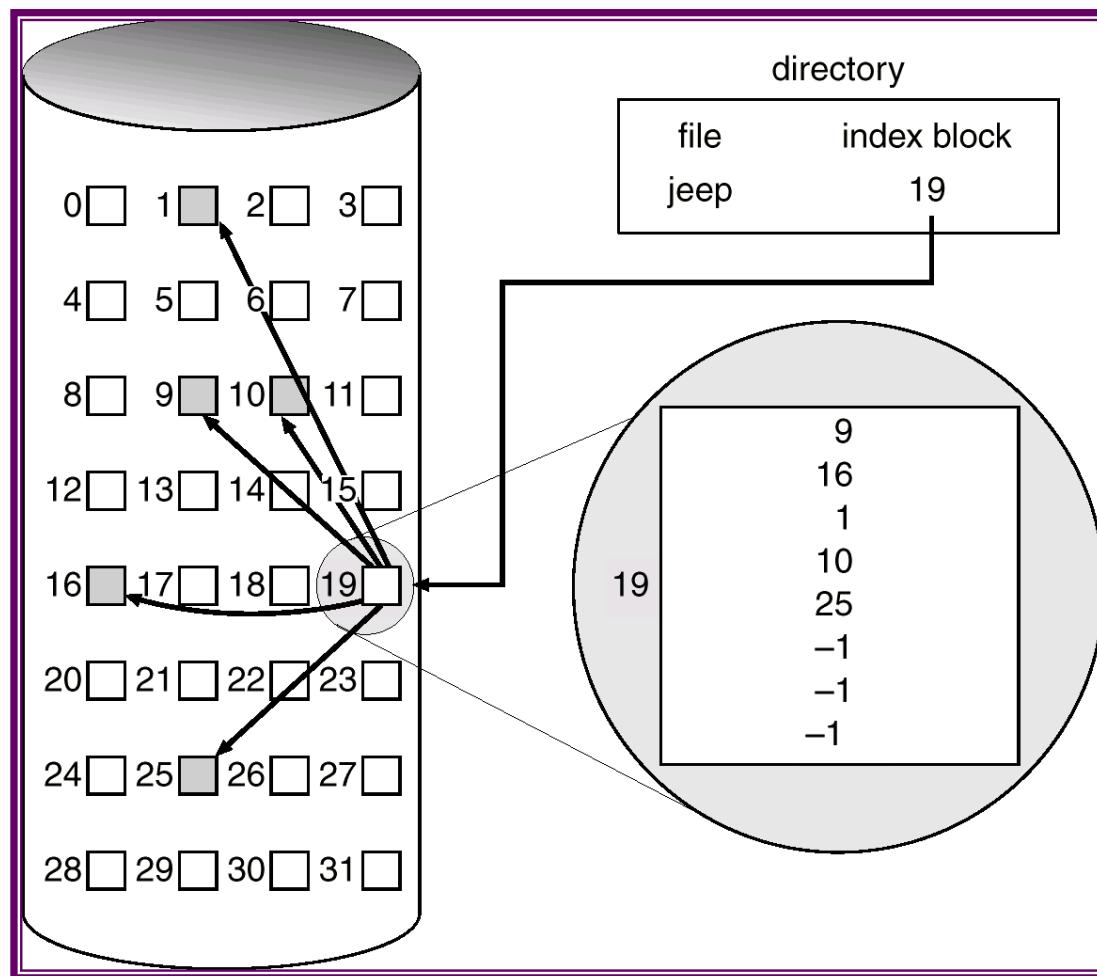


Indexed Allocation

- Brings all pointers together into the *index block*.
- Logical view.



Example of Indexed Allocation



Indexed Allocation (Cont.)

- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block.
- Mapping from logical to physical in a file of maximum size of 256K words and block size of 512 words. We need only 1 block for index table.

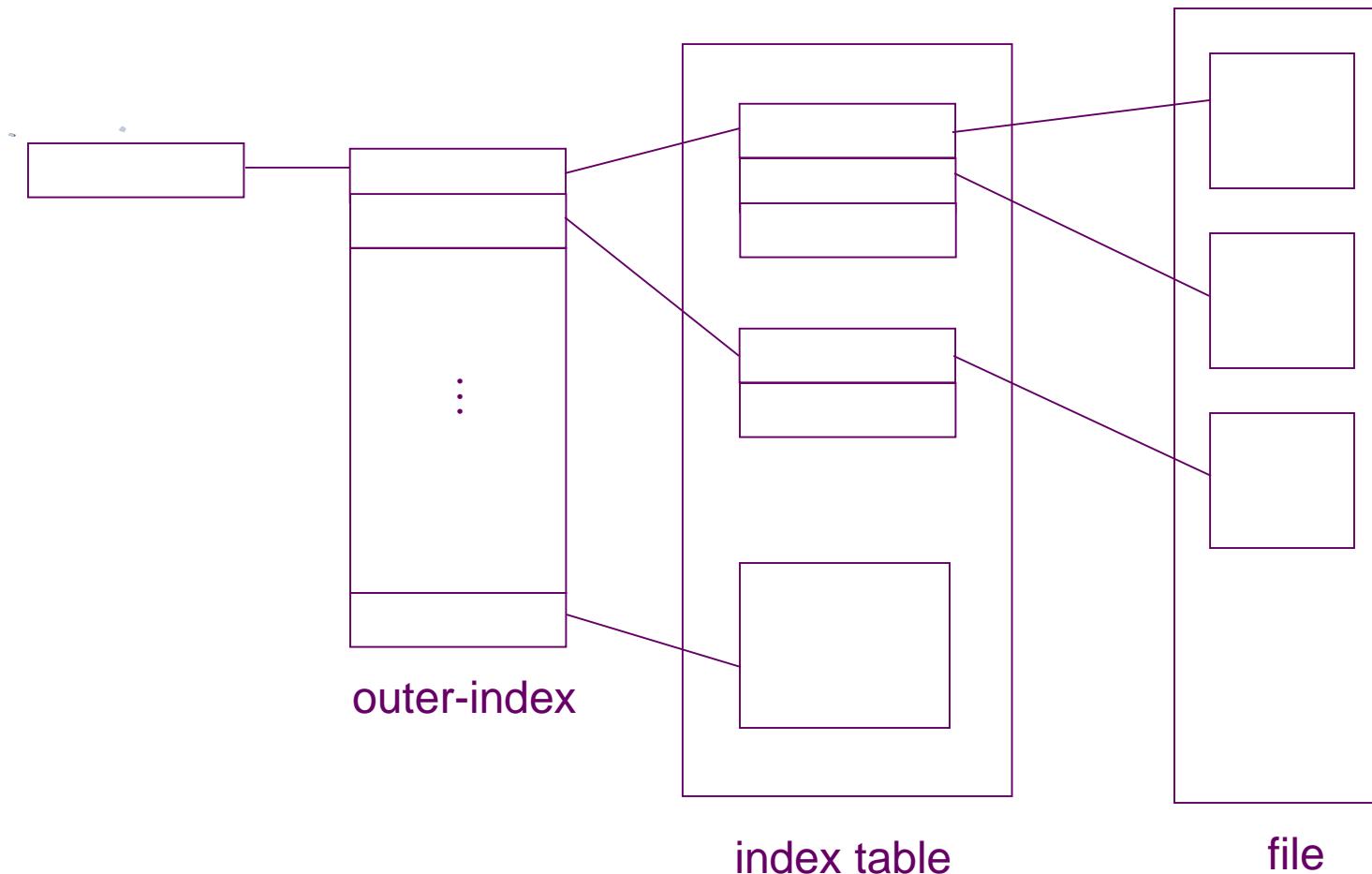
Indexed Allocation – Mapping (Cont.)

- Mapping from logical to physical in a file of unbounded length (block size of 512 words).
- Linked scheme – Link blocks of index table (no limit on size).

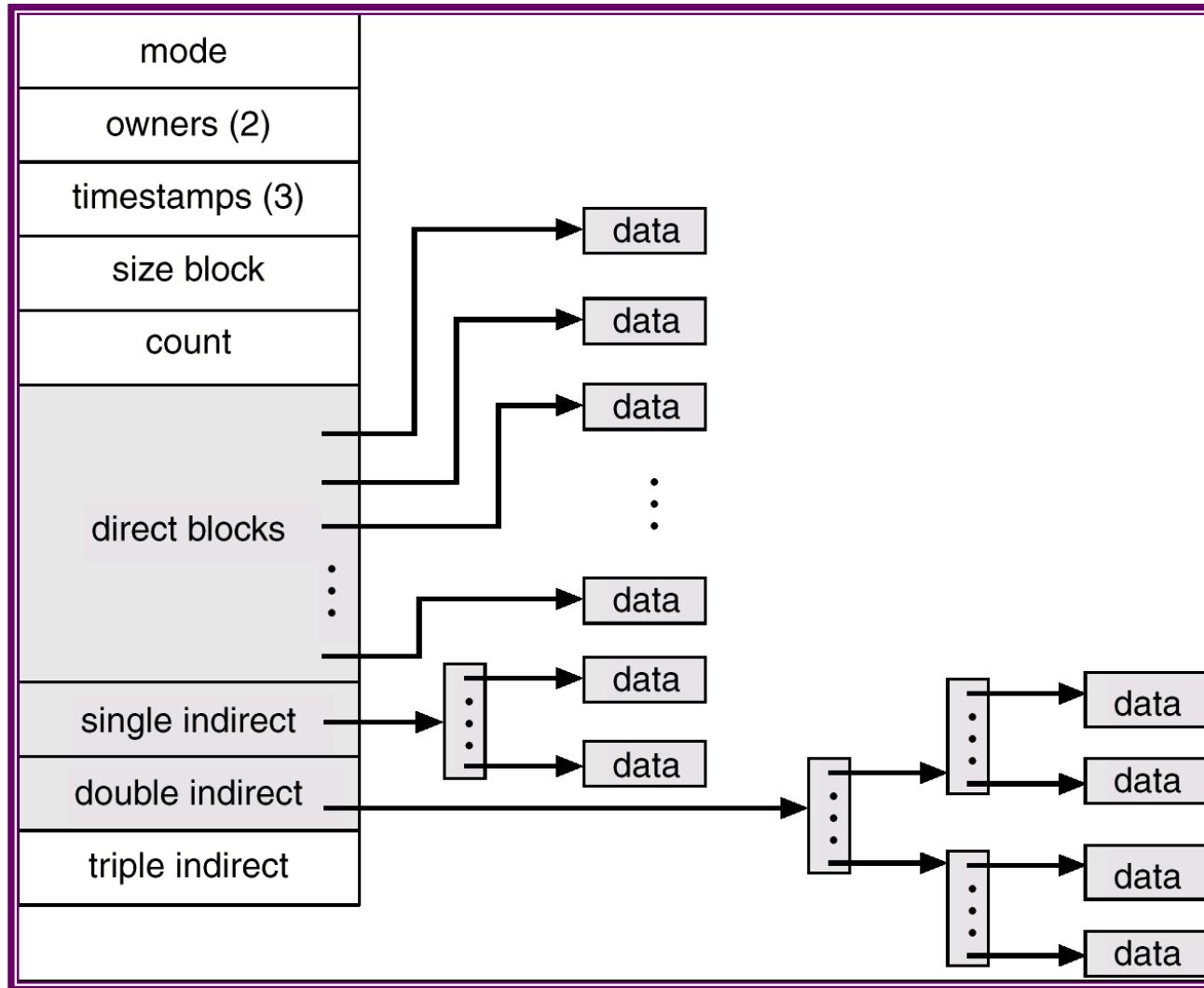
Indexed Allocation – Mapping (Cont.)

- Two-level index

Indexed Allocation – Mapping (Cont.)



Combined Scheme: UNIX (4K bytes per block)



Free-Space Management

- Bit vector (n blocks)



$$\text{bit}[i] = \begin{cases} 0 & \Rightarrow \text{block}[i] \text{ free} \\ 1 & \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

Block number calculation

$$(\text{number of bits per word}) * \\ (\text{number of 0-value words}) + \\ \text{offset of first 1 bit}$$

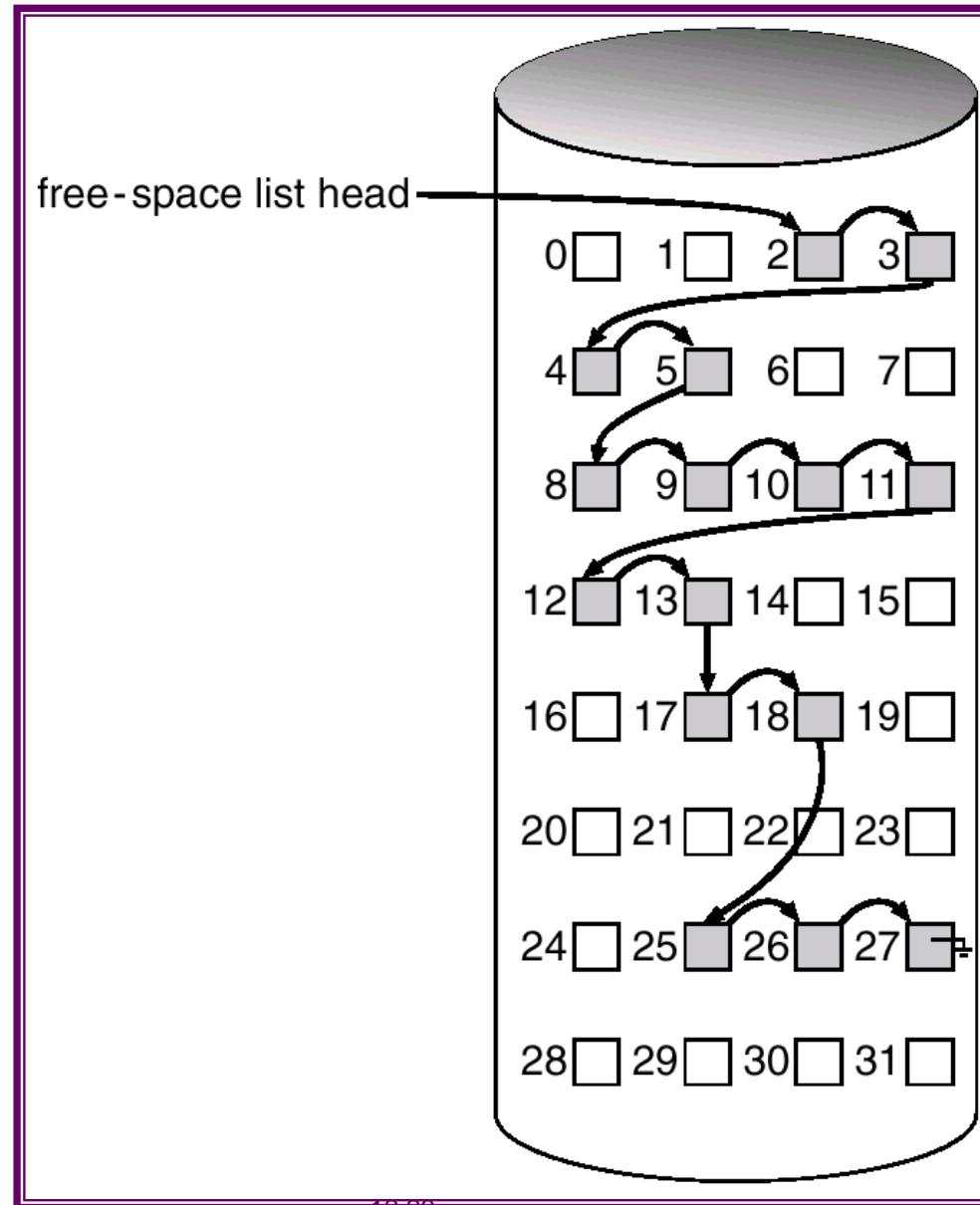
Free-Space Management (Cont.)

- Bit map requires extra space. Example:
 - block size = 2^{12} bytes
 - disk size = 2^{30} bytes (1 gigabyte)
 - $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)
- Easy to get contiguous files
- Linked list (free list)
 - ◆ Cannot get contiguous space easily
 - ◆ No waste of space
- Grouping
- Counting

Free-Space Management (Cont.)

- Need to protect:
 - ◆ Pointer to free list
 - ◆ Bit map
 - ✓ Must be kept on disk
 - ✓ Copy in memory and disk may differ.
 - ✓ Cannot allow for $\text{block}[i]$ to have a situation where $\text{bit}[i] = 1$ in memory and $\text{bit}[i] = 0$ on disk.
 - ◆ Solution:
 - ✓ Set $\text{bit}[i] = 1$ in disk.
 - ✓ Allocate $\text{block}[i]$
 - ✓ Set $\text{bit}[i] = 1$ in memory

Linked Free Space List on Disk



Efficiency and Performance

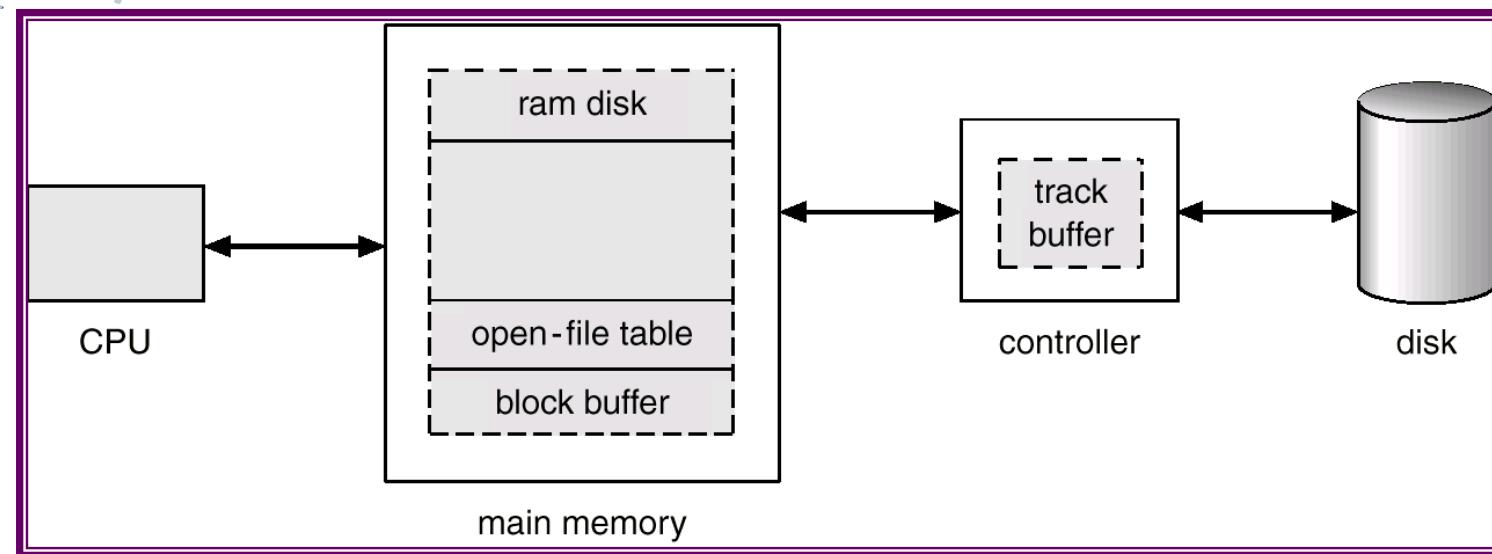
■ Efficiency dependent on:

- ◆ disk allocation and directory algorithms
- ◆ types of data kept in file's directory entry

■ Performance

- ◆ disk cache – separate section of main memory for frequently used blocks
- ◆ free-behind and read-ahead – techniques to optimize sequential access
- ◆ improve PC performance by dedicating section of memory as virtual disk, or RAM disk.

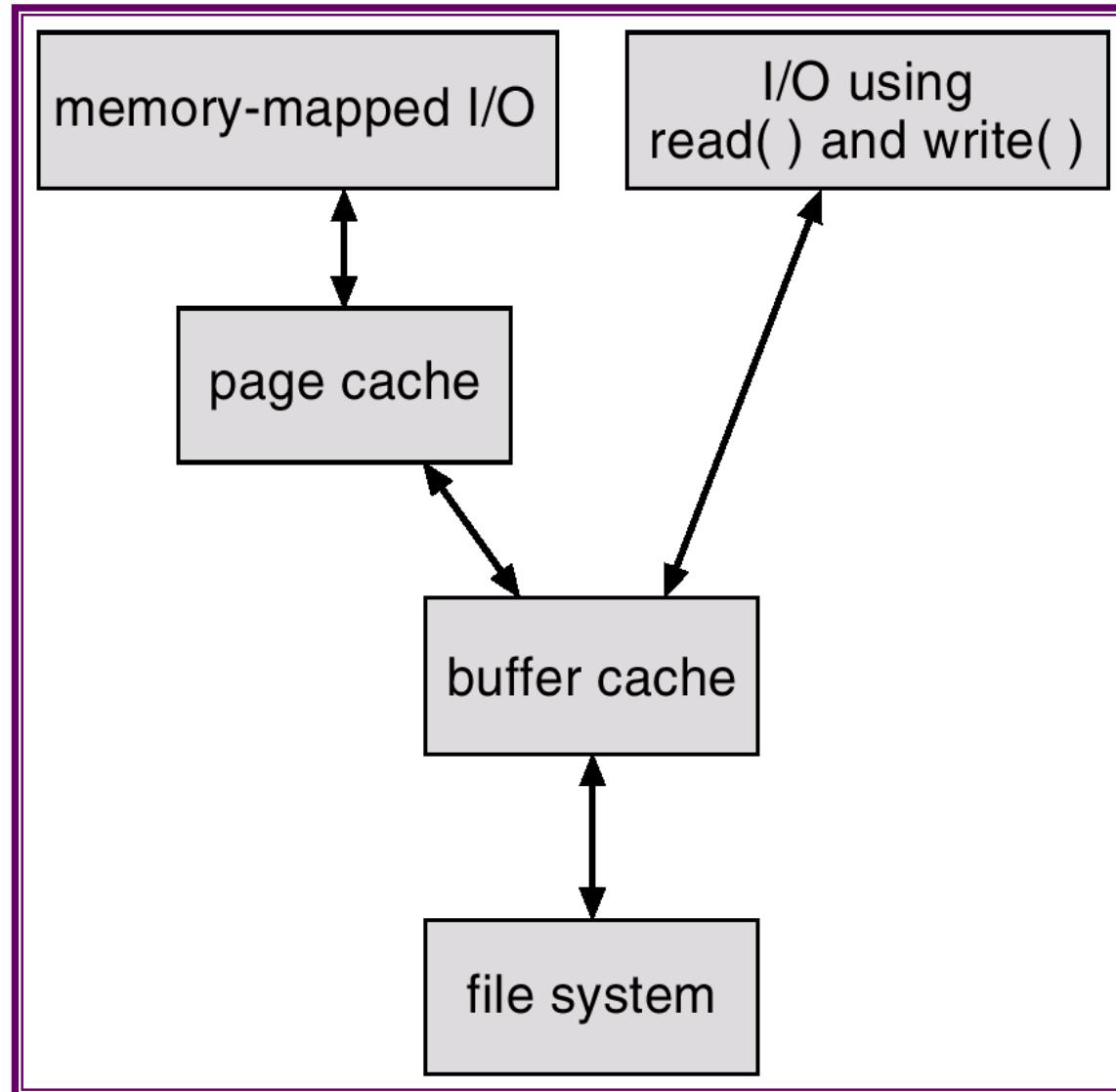
Various Disk-Caching Locations



Page Cache

- A **page cache** caches pages rather than disk blocks using virtual memory techniques.
- Memory-mapped I/O uses a page cache.
- Routine I/O through the file system uses the buffer (disk) cache.
- This leads to the following figure.

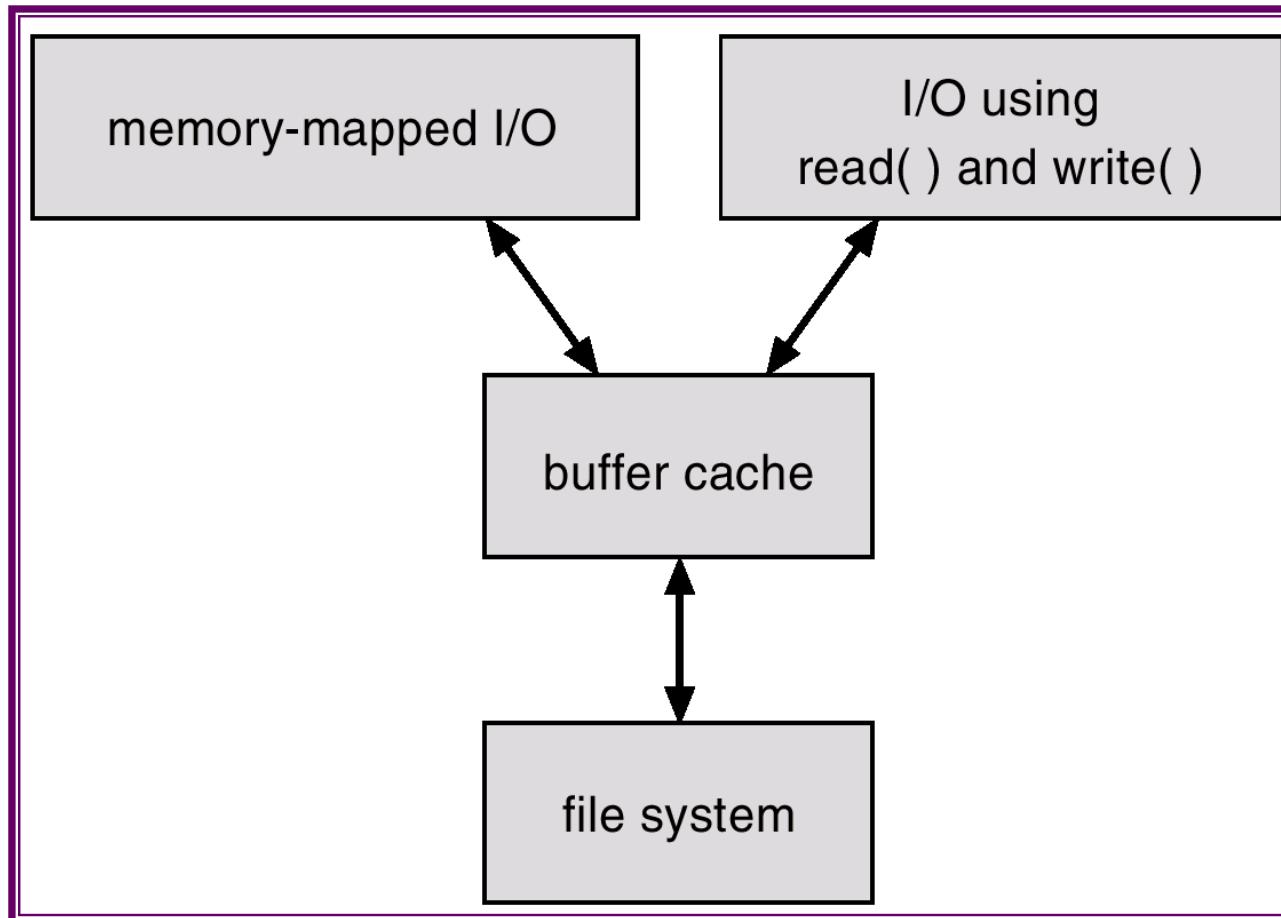
I/O Without a Unified Buffer Cache



Unified Buffer Cache

- A unified buffer cache uses the same page cache to cache both memory-mapped pages and ordinary file system I/O.

I/O Using a Unified Buffer Cache



Recovery

- Consistency checking – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies.
- Use system programs to *back up* data from disk to another storage device (floppy disk, magnetic tape).
- Recover lost file or disk by *restoring* data from backup.

Log Structured File Systems

- Log structured (or journaling) file systems record each update to the file system as a **transaction**.
- All transactions are written to a **log**. A transaction is considered **committed** once it is written to the log. However, the file system may not yet be updated.
- The transactions in the log are asynchronously written to the file system. When the file system is modified, the transaction is removed from the log.
- If the file system crashes, all remaining transactions in the log must still be performed.

The Sun Network File System (NFS)

- An implementation and a specification of a software system for accessing remote files across LANs (or WANs).
- The implementation is part of the Solaris and SunOS operating systems running on Sun workstations using an unreliable datagram protocol (UDP/IP protocol and Ethernet).

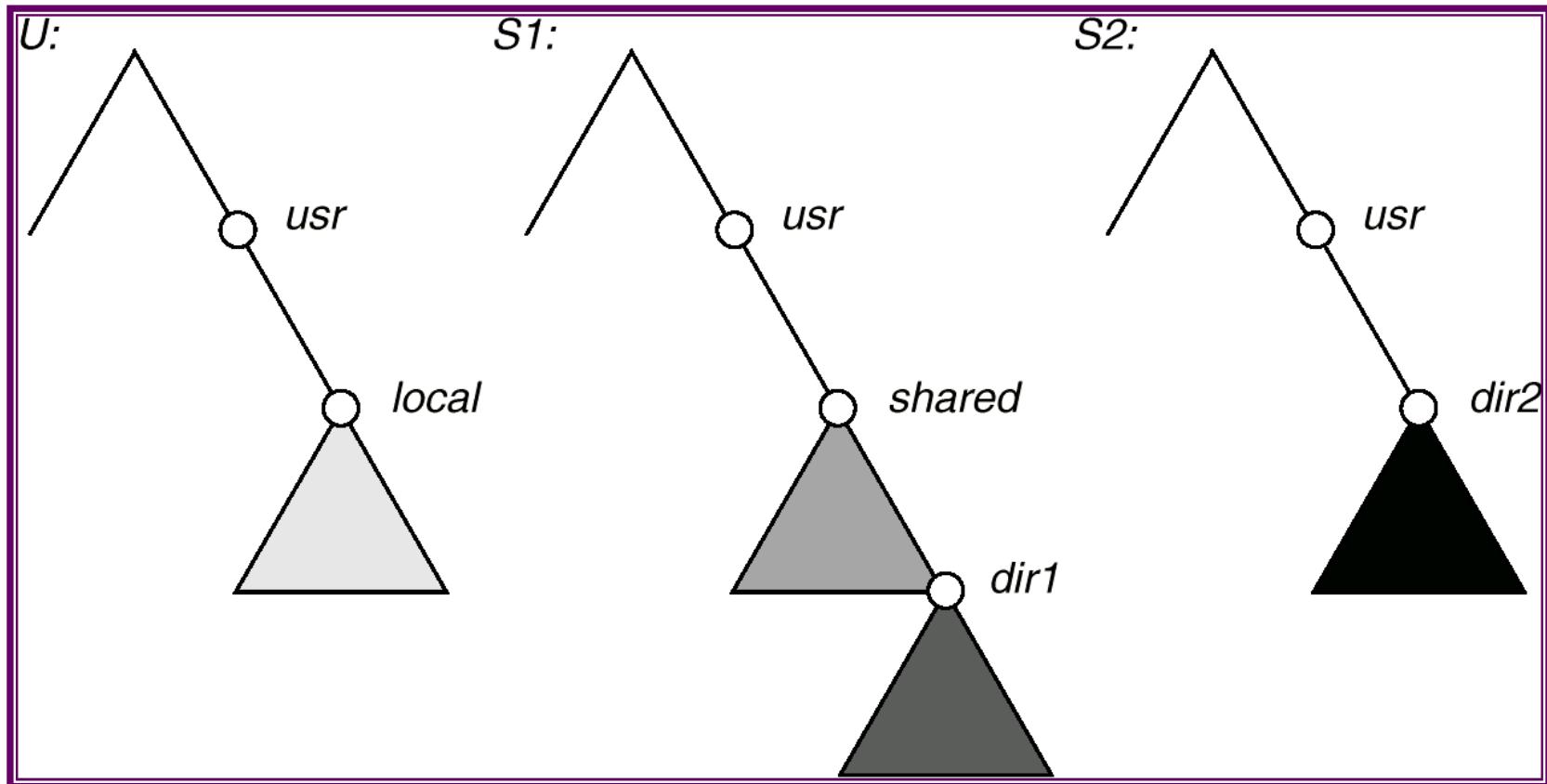
NFS (Cont.)

- Interconnected workstations viewed as a set of independent machines with independent file systems, which allows sharing among these file systems in a transparent manner.
 - ◆ A remote directory is mounted over a local file system directory. The mounted directory looks like an integral subtree of the local file system, replacing the subtree descending from the local directory.
 - ◆ Specification of the remote directory for the mount operation is nontransparent; the host name of the remote directory has to be provided. Files in the remote directory can then be accessed in a transparent manner.
 - ◆ Subject to access-rights accreditation, potentially any file system (or directory within a file system), can be mounted remotely on top of any local directory.

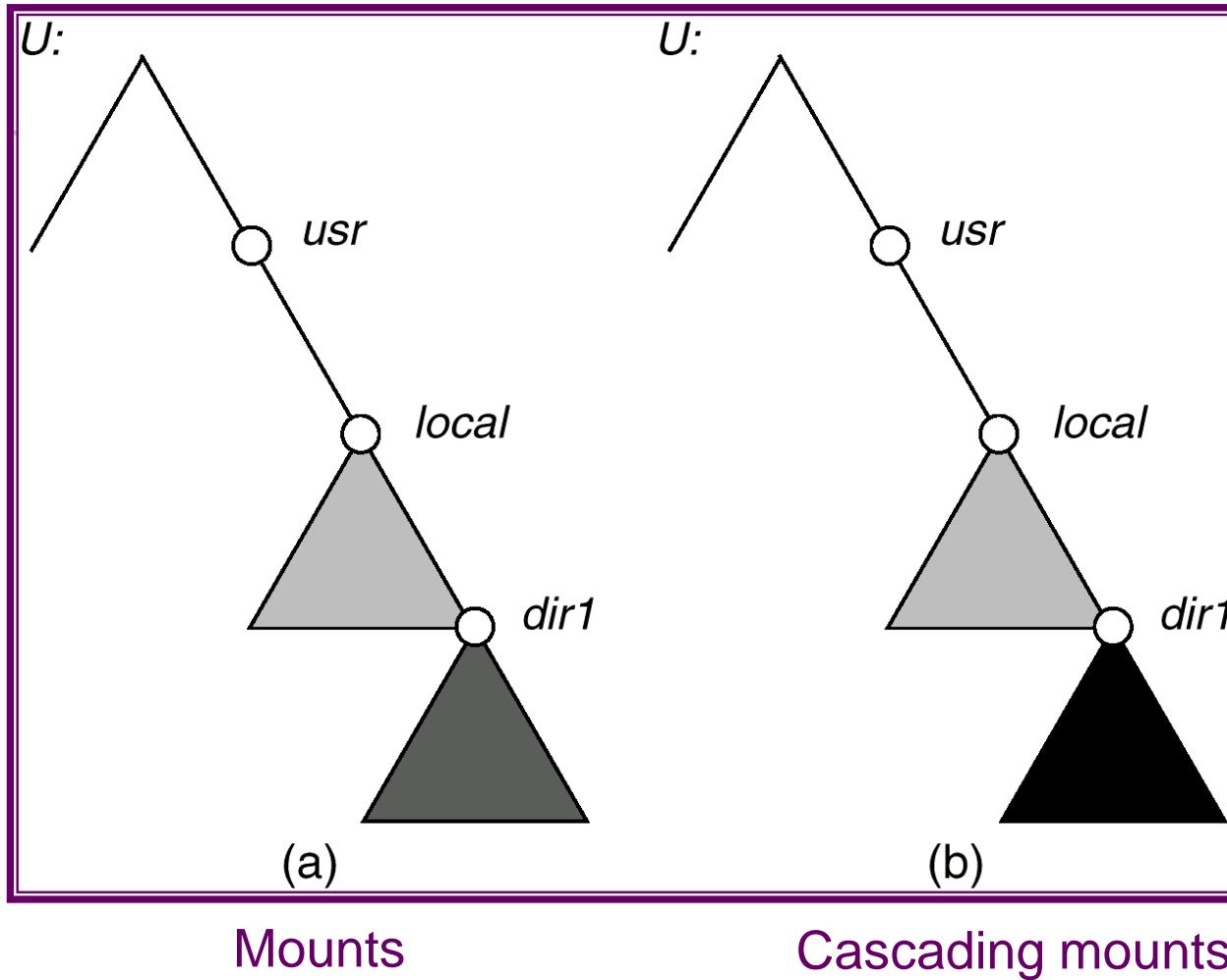
NFS (Cont.)

- NFS is designed to operate in a heterogeneous environment of different machines, operating systems, and network architectures; the NFS specifications independent of these media.
- This independence is achieved through the use of RPC primitives built on top of an External Data Representation (XDR) protocol used between two implementation-independent interfaces.
- The NFS specification distinguishes between the services provided by a mount mechanism and the actual remote-file-access services.

Three Independent File Systems



Mounting in NFS



NFS Mount Protocol

- Establishes initial logical connection between server and client.
- Mount operation includes name of remote directory to be mounted and name of server machine storing it.
 - ◆ Mount request is mapped to corresponding RPC and forwarded to mount server running on server machine.
 - ◆ *Export list* – specifies local file systems that server exports for mounting, along with names of machines that are permitted to mount them.
- Following a mount request that conforms to its export list, the server returns a *file handle*—a key for further accesses.
- File handle – a file-system identifier, and an inode number to identify the mounted directory within the exported file system.
- The mount operation changes only the user's view and does not affect the server side.

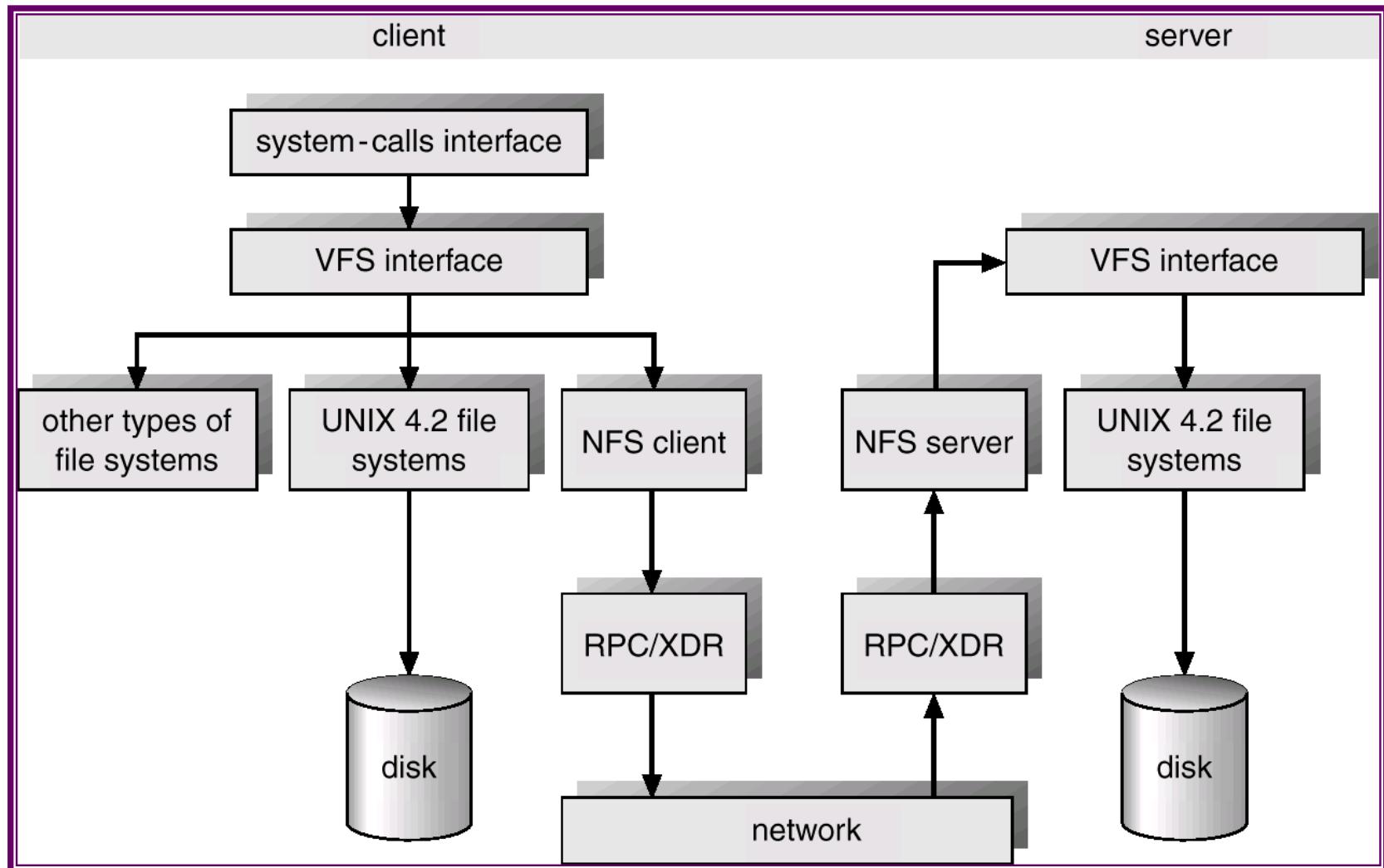
NFS Protocol

- Provides a set of remote procedure calls for remote file operations. The procedures support the following operations:
 - ◆ searching for a file within a directory
 - ◆ reading a set of directory entries
 - ◆ manipulating links and directories
 - ◆ accessing file attributes
 - ◆ reading and writing files
- NFS servers are *stateless*; each request has to provide a full set of arguments.
- Modified data must be committed to the server's disk before results are returned to the client (lose advantages of caching).
- The NFS protocol does not provide concurrency-control mechanisms.

Three Major Layers of NFS Architecture

- UNIX file-system interface (based on the **open**, **read**, **write**, and **close** calls, and file descriptors).
- *Virtual File System* (VFS) layer – distinguishes local files from remote ones, and local files are further distinguished according to their file-system types.
 - ◆ The VFS activates file-system-specific operations to handle local requests according to their file-system types.
 - ◆ Calls the NFS protocol procedures for remote requests.
- NFS service layer – bottom layer of the architecture; implements the NFS protocol.

Schematic View of NFS Architecture



NFS Path-Name Translation

- Performed by breaking the path into component names and performing a separate NFS lookup call for every pair of component name and directory vnode.
- To make lookup faster, a directory name lookup cache on the client's side holds the vnodes for remote directory names.

NFS Remote Operations

- Nearly one-to-one correspondence between regular UNIX system calls and the NFS protocol RPCs (except opening and closing files).
- NFS adheres to the remote-service paradigm, but employs buffering and caching techniques for the sake of performance.
- File-blocks cache – when a file is opened, the kernel checks with the remote server whether to fetch or revalidate the cached attributes. Cached file blocks are used only if the corresponding cached attributes are up to date.
- File-attribute cache – the attribute cache is updated whenever new attributes arrive from the server.
- Clients do not free delayed-write blocks until the server confirms that the data have been written to disk.

Multimedia Systems

Chapter: Multimedia Systems

- What is Multimedia
- Compression Techniques
- Requirements of Multimedia Kernels
- CPU Scheduling
- Disk Scheduling
- Network Management
- An Example: Cineblitz

Objectives

- To identify the characteristics of multimedia data
- To examine several algorithms used to compress multimedia data
- To explore the operating system requirements of multimedia data, including CPU and disk scheduling and network management

What is Multimedia?

- Multimedia data includes
 - audio and video clips (i.e. MP3 and MPEG files)
 - live webcasts
- Multimedia data may be delivered to
 - desktop PC's
 - handheld devices (PDAs, smart phones)

Media Delivery

- Multimedia data is stored in the file system like the ordinary data.
- However, multimedia data must be accessed with specific timing requirements.
- For example, video must be displayed at 24-30 **frames** per second. Multimedia video data must be delivered at a rate which guarantees 24-30 frames/second.
- **Continuous-media data** is data with specific rate requirements.

Streaming

- **Streaming** is delivering a multimedia file from a server to a client - typically the delivery occurs over a network connection.
- There are two different types of streaming:
 1. **Progressive download** - the client begins playback of the multimedia file as it is delivered. The file is ultimately stored on the client computer.
 2. **Real-time streaming** - the multimedia file is delivered to - but not stored on - the client's computer.

Real-time Streaming

- There are two types of real-time streaming:
 - (1) **Live streaming** - used to deliver a live event while it is occurring.
 - (2) **On-demand streaming** - used to deliver media streams such as movies, archived lectures, etc. The events are not delivered in real-time.

Multimedia Systems

Characteristics

- Multimedia files can be quite large.
- Continuous media data may require very high data rates.
 - Consider a video of resolution 800*600. If we use 24 bits to represent colour, we have 2^{24} (about 16 million colours). A single frame requires $800*600*24=11,520,000$ bits of data. If the frames are displayed 30 frames per second we requires more than 345 Mbps bandwidth.
- Multimedia applications may be sensitive to timing delays during playback of the media.

Compression

- Because of the size and rate requirements of multimedia systems, multimedia files are often compressed into a smaller form.
- Basic idea
 - Lossy compression
 - Store the differences between successive frames.
- MPEG Compression:
 - (1) MPEG-1 - 352 X 240 @ 30 frames/second
 - (2) MPEG-2 - Used for compressing DVD and high-definition television (HDTV)
 - (3) MPEG-4 - Used to transmit audio, video, and graphics.
Can be delivered over very slow connections (56 Kbps)

Operating Systems Issues

- The operating system must guarantee the specific data rate and timing requirements of continuous media.
- Such requirements are known as **Quality-of-Service (QoS)** guarantees.

Parameters Defining QoS

- **Throughput** - the total amount of work completed during a specific time interval.
- **Delay** - the elapsed time from when a request is first submitted to when the desired result is produced.
- **Jitter** - the delays that occur during playback of a stream.
 - Due to lost frames.
 - Not acceptable for continuous media applications
- **Reliability** - how errors are handled during transmission and processing of continuous media.

Requirement of Multimedia Operating Systems

- There are three levels of QoS
 - (1) Best-effort service - the system makes a best effort with no QoS guarantees.
 - (2) Soft QoS - allows different traffic streams to be prioritized, however no QoS guarantees are made.
 - (3) Hard QoS - the QoS requirements are guaranteed.

Further QoS Issues

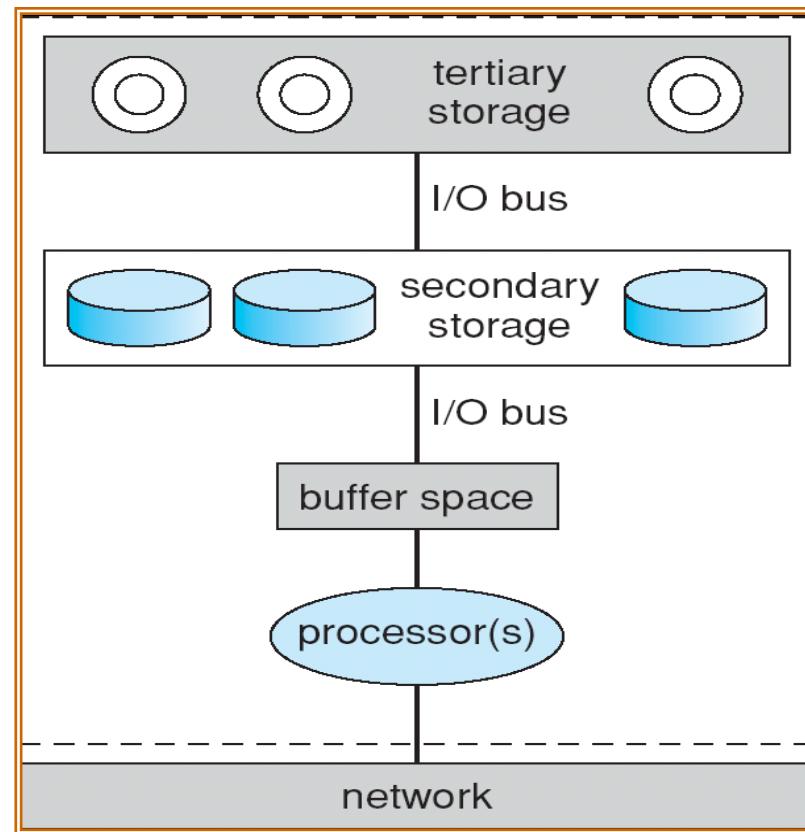
- QoS may be **negotiated** between the client and server.
- Operating systems often use an **admission control** algorithm that admits a request for a service only if the server has sufficient resources to satisfy the request.
 - Resource reservation
 - Requests arrive with associated QOS.
 - Assigns a resource manager for each type of resource.
 - The resource manager rejects the service if it can not allocate resources to meet QOS.

QoS Guarantees

- Guaranteeing QoS has the following effects in a computer system:
 - (1) CPU processing
 - (2) Scheduling
 - (3) File systems
 - (4) Network protocols

Figure 20.1

Resources on a file server



CPU Scheduling

- Multimedia systems require **hard realtime** scheduling to ensure critical tasks will be serviced within timing deadlines.
- Most hard realtime CPU scheduling algorithms assign realtime processes static priorities that do not change over time.

Disk Scheduling

- Disk scheduling algorithms must be optimized to meet the timing deadlines and rate requirements of continuous media.
- Earliest-Deadline-First (EDF) Scheduling
- SCAN-EDF Scheduling

Disk Scheduling (cont)

- The EDF scheduler uses a queue to order requests according to the time it must be completed (its deadline.)
- SCAN-EDF scheduling is similar to EDF except that requests with the same deadline are ordered according to a SCAN policy.

Deadline and cylinder requests for SCAN-EDF scheduling

| request | deadline | cylinder |
|---------|----------|----------|
| A | 150 | 25 |
| B | 201 | 112 |
| C | 399 | 95 |
| D | 94 | 31 |
| E | 295 | 185 |
| F | 78 | 85 |
| G | 165 | 150 |
| H | 125 | 101 |
| I | 300 | 85 |
| J | 210 | 90 |

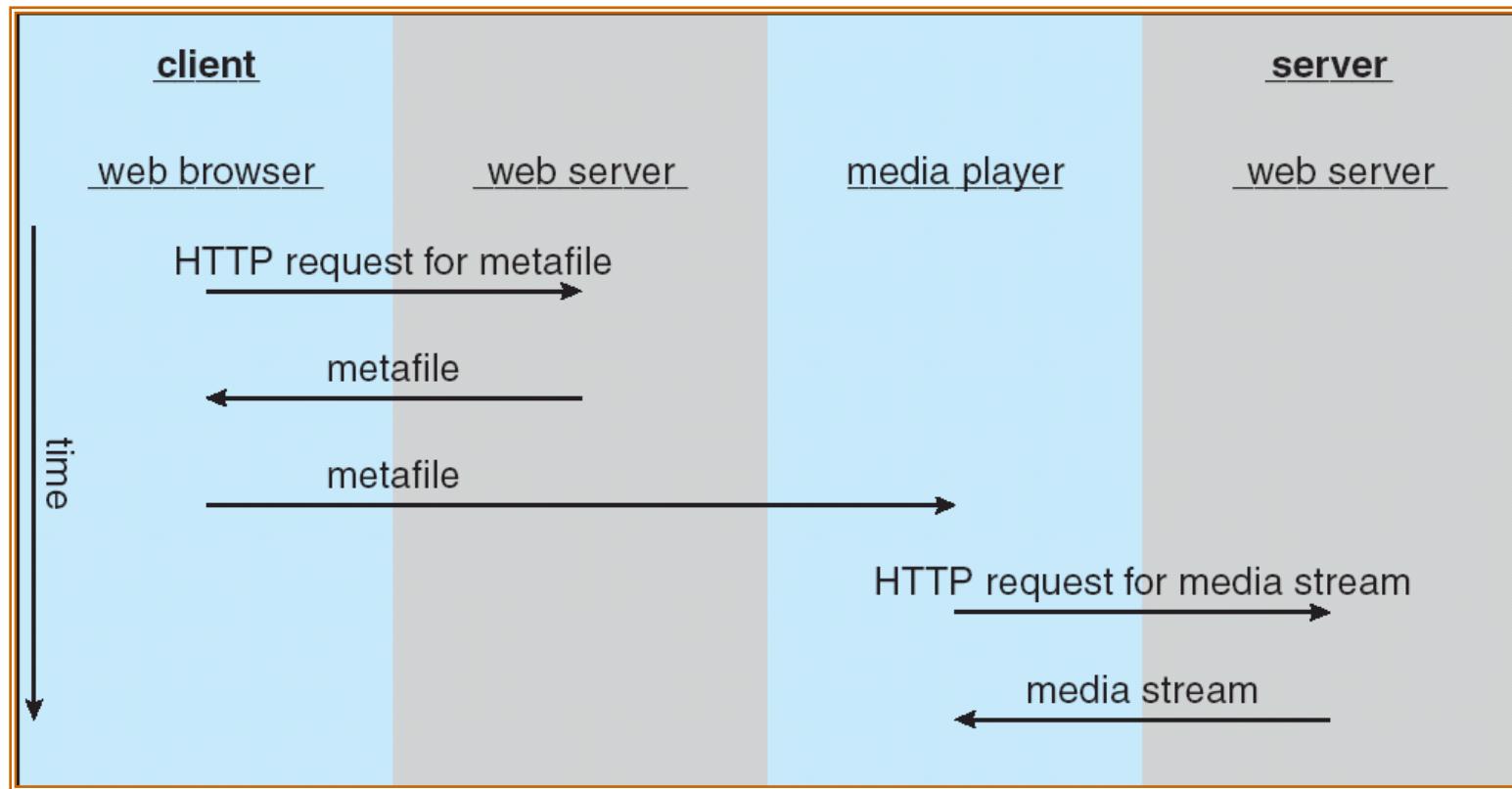
Network Management

- Three general methods for delivering content from a server to a client across a network:
 - (1) **Unicasting** - the server delivers the content to a single client.
 - (2) **Broadcasting** - the server delivers the content to all clients, regardless whether they want the content or not.
 - (3) **Multicasting** - the server delivers the content to a group of receivers who indicate they wish to receive the content.

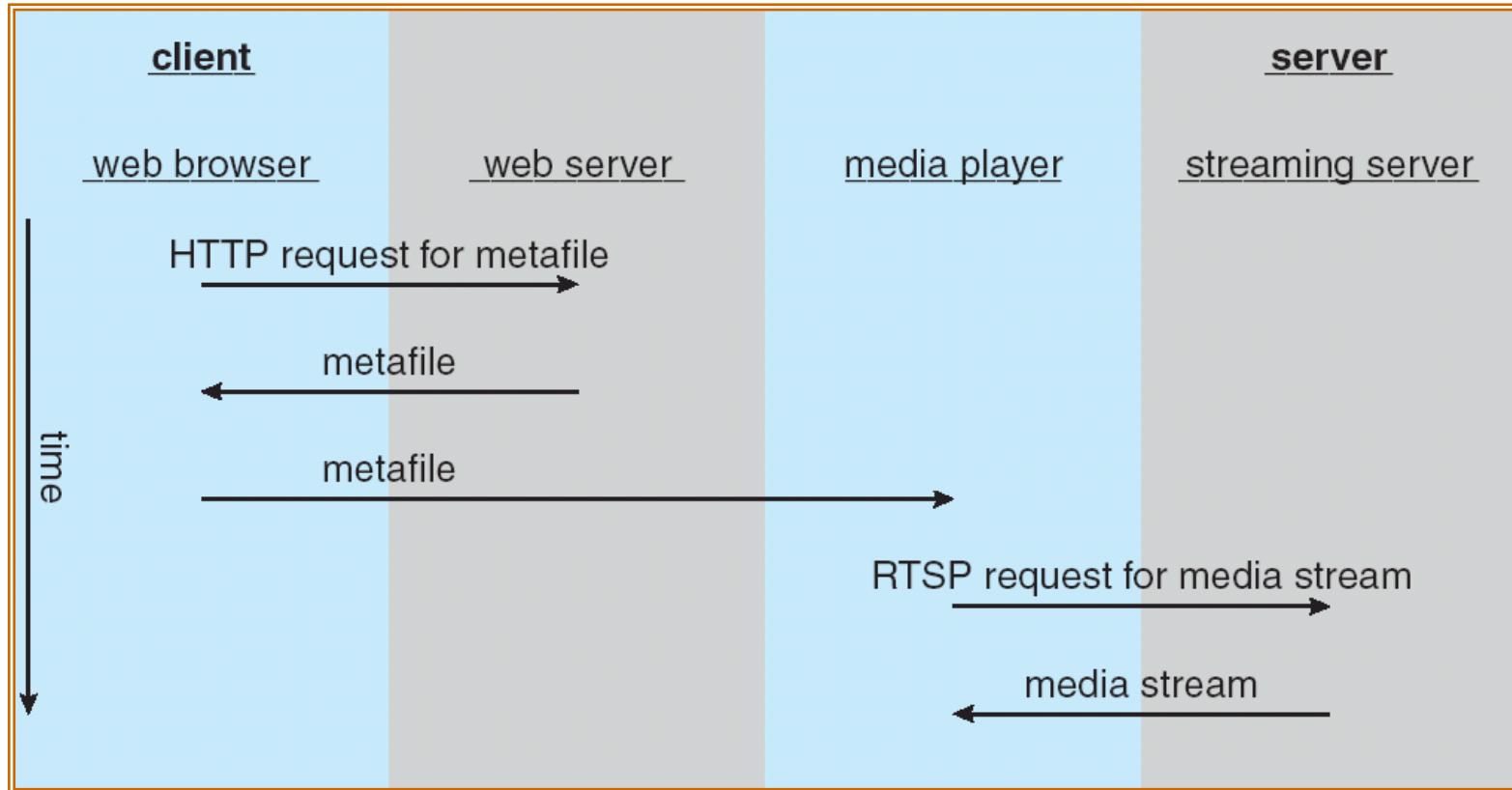
RealTime Streaming Protocol (RTSP)

- Standard HTTP is stateless whereby the server does not maintain the status of its connection with the client.

Streaming media from a conventional web server



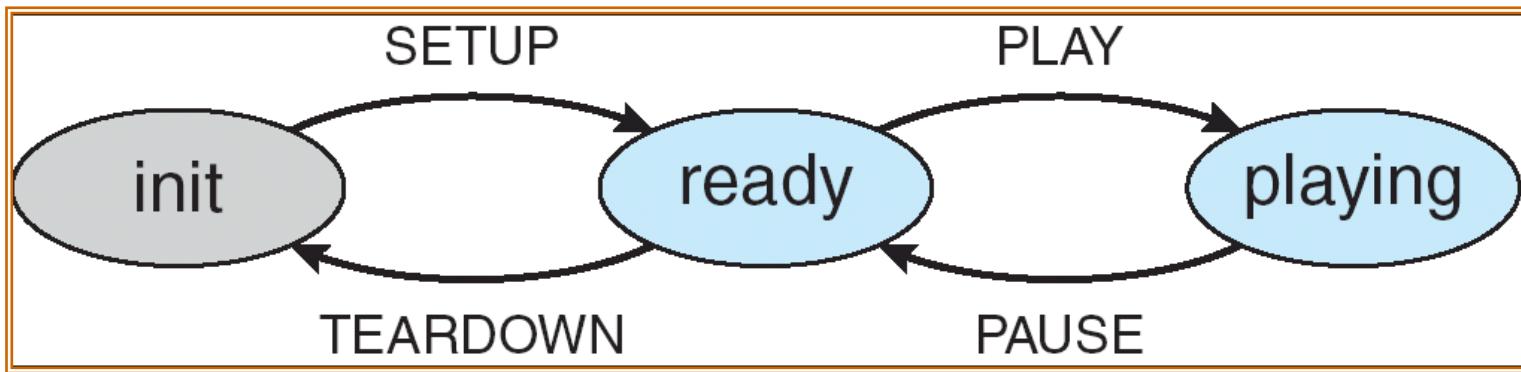
Realtime Streaming Protocol



RTSP States

- SETUP - the server allocates resources for a client session.
- PLAY - the server delivers a stream to a client session.
- PAUSE - the server suspends delivery of a stream.
- TEARDOWN - the server breaks down the connection and releases the resources allocated for the session.

RTSP state machine



CineBlitz Multimedia Server

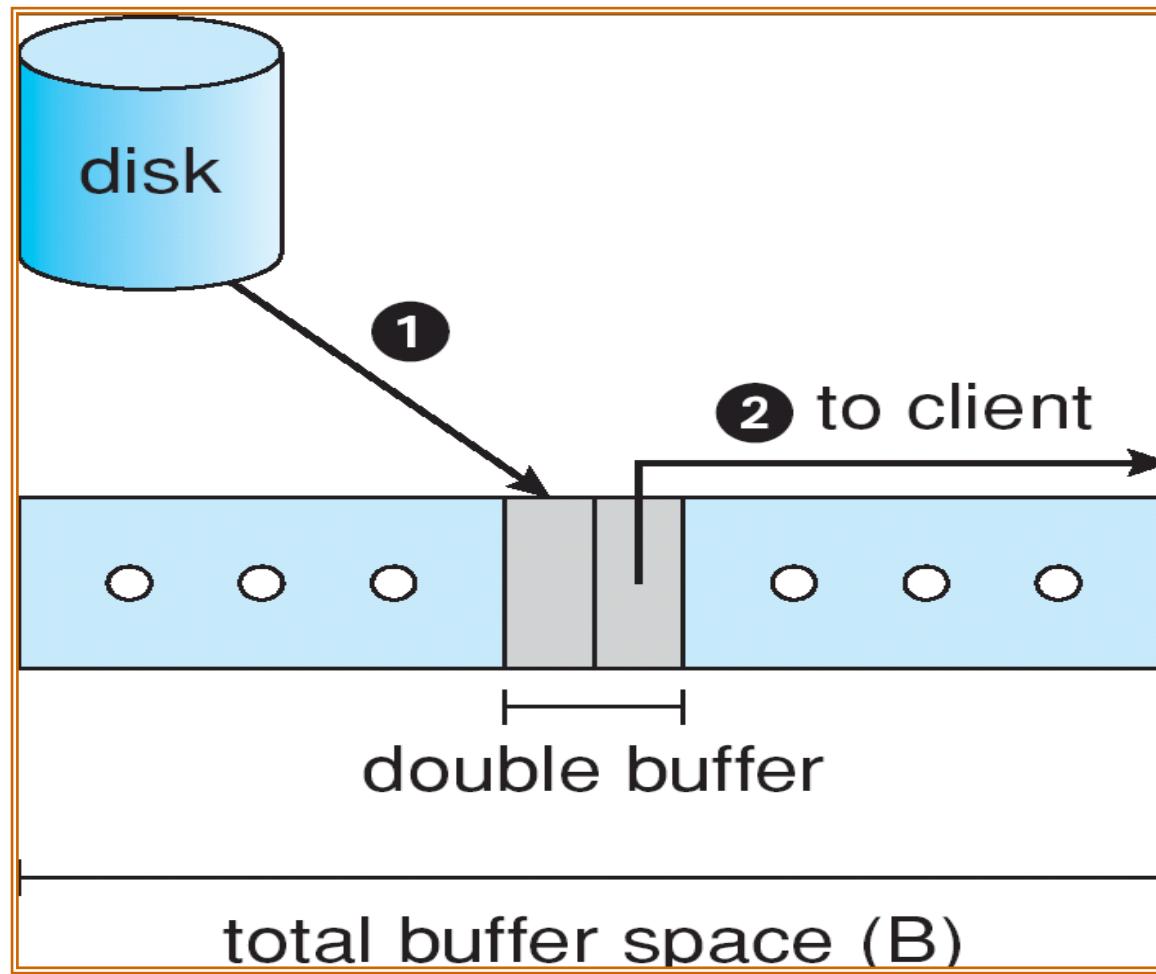
- CineBlitz supports both realtime and non-realtime clients.
- CineBlitz provides hard QoS guarantees to realtime clients using an admission control algorithm.
- The disk scheduler orders requests using C-SCAN order.

CineBlitz Admission Controller

- Total buffer space required for N clients where client has rate requirement of r_i

$$\sum_{i=1}^N 2 \times T \times r_i \leq B.$$

Double buffering in CineBlitz



CineBlitz Admission Controller (cont)

- If t_{seek} and t_{rot} are the worst-case seek and rotational delay times, the maximum latency for servicing N requests is

$$2 \times t_{seek} + \sum_{i=1}^N \left(\lceil \frac{T \times r_i}{b} \rceil + 1 \right) \times t_{rot}.$$

CineBlitz Admission Controller (cont)

- The CineBlitz admission controller only admits a new client if there is at least $2 \times T \times r_i$ bits of free buffer space and the following equation is satisfied

$$2 \times t_{seek} + \sum_{i=1}^N \left(\lceil \frac{T \times r_i}{b} \rceil + 1 \right) \times t_{rot} + \sum_{i=1}^N \frac{T \times r_i}{r_{disk}} \leq T.$$

System Protection

- Goals of Protection
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix
- Revocation of Access Rights
- Capability-Based Systems
- Language-Based Protection

Protection

- Protection must ensure that only those processes that have gained proper authorization from the OS can operate on memory segments, the CPU, and other resources.
- Operating system consists of a collection of objects, hardware or software
- Each object has a unique name and can be accessed through a well-defined set of operations.
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so.
- Enforcement of the policies governing resource usage.
- A protection system must have the flexibility to enforce a variety of policies that can be declared to it.

Goals of Protection

- Provide mechanisms for enforcement of policies.
 - ◆ Mechanisms determine how something will be done. Policies decide what will be done.
- Policies may change over time and can be decided by application programmer or system programmer.
- In this chapter we discuss the protection mechanism the OS should provide so that the application designers can design their own protection software.

Guiding principle

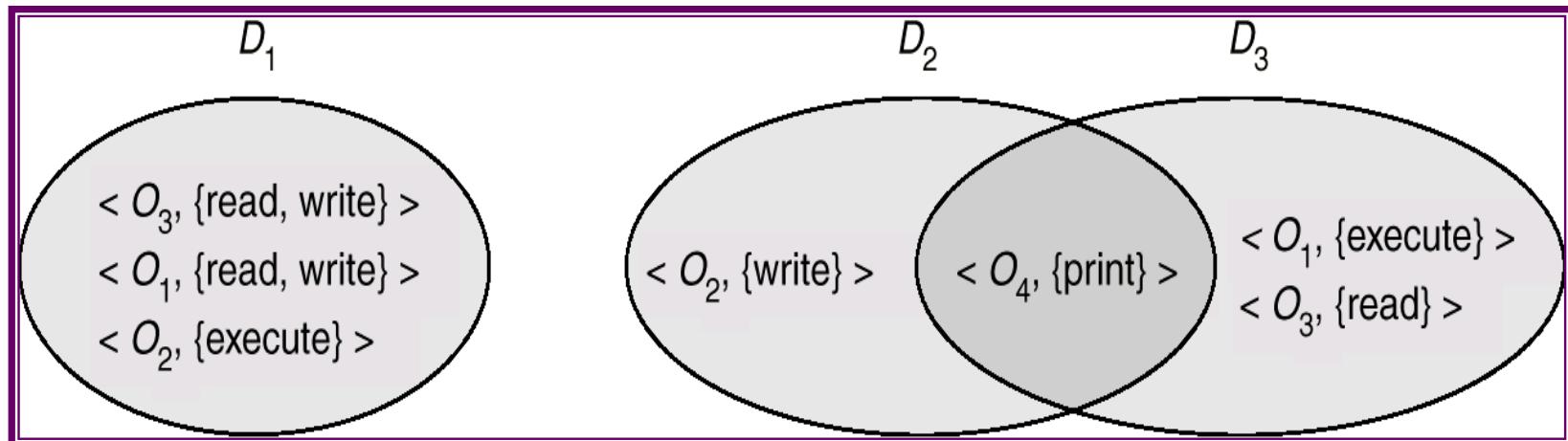
- Principle of least privilege
 - ◆ Dictates that programs, users and even systems be given just enough privileges to perform their task.
- Uses fine grained access controls
- Beneficial to create an audit trial
- Audit trial allows tracing of violations.

Domain Structure

- Computer system is a collection of processes and objects
- Objects are abstract data types
 - ◆ Hardware objects (cpu, memory segments, printers..) and software objects (files, programs, and semaphores).
- Each object is accessed by a well-defined and meaningful operations.
- Operations depend on the object
 - ◆ A CPU may only be executed on. Memory systems may read or written. Files can be open, close, read, write, executed, deleted.
- The process should access only those resources which it requires or allowed to access.
 - ◆ Need to know principle

Domain Structure

- A process operates within a **protection domain**.
- Domain specifies the resources a process may access
- Access right= The ability to execute an operation on an object
- Domain is a collection of access rights.
- Access-right = $\langle \text{object-name}, \text{rights-set} \rangle$
where *rights-set* is a subset of all valid operations that can be performed on the object.



Ways of realizing domains

- Each user may be a domain
 - ◆ Set of objects that can be accessed depends up on the identity of the user.
- Each process may be a domain
 - ◆ Set of objects that can be accessed depends on the identity of the process.
- Each procedure may be a domain
 - ◆ Set of objects that can be accessed corresponds to the local variables defined within the procedure.

Domain Implementation (UNIX)

■ System consists of 2 domains:

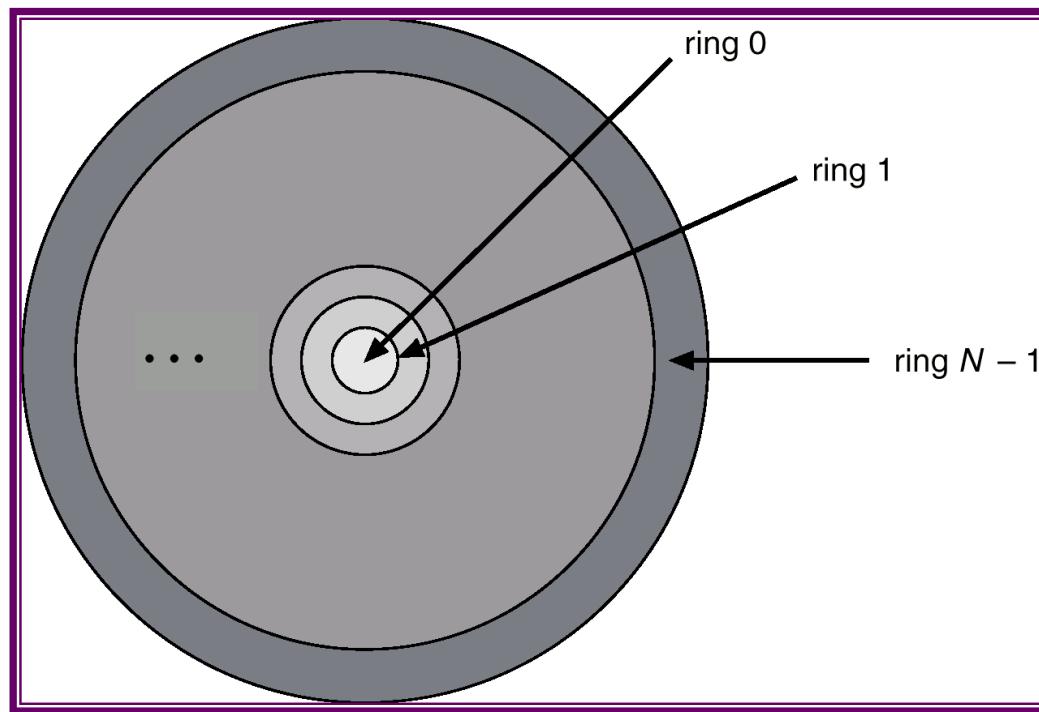
- ◆ User
- ◆ Supervisor

■ UNIX

- ◆ Domain = user-id
- ◆ Domain switch accomplished via file system.
 - ✓ Each file has associated with it a domain bit (setuid bit).
 - ✓ When file is executed and setuid = on, then user-id is set to owner of the file being executed. When execution completes user-id is reset.

Domain Implementation (Multics)

- Protection domains are organized hierarchically into a ring structure.
- Let D_i and D_j be any two domain rings.
- If $j < i \Rightarrow D_i \subseteq D_j$
- Protection system is more complex and less efficient.



Multics Rings

Access Matrix Method

- View protection as a matrix (*access matrix*)
- Rows represent domains
- Columns represent objects
- Each entry in the matrix consists of set of access rights.
- $\text{Access}(i, j)$ is the set of operations that a process executing in Domain_i can invoke on Object_j
- Access Matrix gives flexibility to implement various policies.

Access Matrix

| object domain | F_1 | F_2 | F_3 | printer |
|------------------|---------------|-------|---------------|---------|
| D_1 | read | | read | |
| D_2 | | | | print |
| D_3 | | read | execute | |
| D_4 | read write | | read write | |

Figure A

Use of Access Matrix

- If a process in Domain D_i tries to do “op” on object O_j , then “op” must be in the access matrix.
- Can be expanded to dynamic protection.
 - ◆ Operations to add, delete access rights.
 - ◆ Special access rights:
 - ✓ *owner of O_i ,*
 - ✓ *copy op from O_i to O_j*
 - ✓ *control – D_i can modify D_j access rights*
 - ✓ *transfer – switch from domain D_i to D_j*

Use of Access Matrix (Cont.)

- Access matrix design separates mechanism from policy.
 - ◆ Mechanism (how something will be done)
 - ✓ Operating system provides access-matrix + rules.
 - ✓ It ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced.
 - ◆ Policy (what will be done)
 - ✓ User dictates policy.
 - ✓ Who can access what object and in what mode.

Implementation of Access Matrix

- Each column = Access-control list for one object
Defines who can perform what operation.

Domain 1 = Read, Write

Domain 2 = Read

Domain 3 = Read

:

- Each Row = Capability List (like a key)
For each domain, what operations allowed on what objects.

Object 1 – Read

Object 4 – Read, Write, Execute

Object 5 – Read, Write, Delete, Copy

Access Matrix of Figure A With Domains as Objects

A process in domain D4 can switch to D1, and one in domain D1 can switch to D2

| object domain | F_1 | F_2 | F_3 | laser printer | D_1 | D_2 | D_3 | D_4 |
|------------------|---------------|-------|---------------|------------------|-------|--------|--------|--------|
| D_1 | read | | read | | | switch | | |
| D_2 | | | | print | | | switch | switch |
| D_3 | | read | execute | | | | | |
| D_4 | read write | | read write | | | switch | | |

Figure B

Access Matrix with Copy Rights

- A process executing in domain D2 can copy the read operation in to any entry associated with F2. Propagation may be limited.

| domain \ object | F_1 | F_2 | F_3 |
|-----------------|---------|-------|---------|
| D_1 | execute | | write* |
| D_2 | execute | read* | execute |
| D_3 | execute | | |

(a)

| domain \ object | F_1 | F_2 | F_3 |
|-----------------|---------|-------|---------|
| D_1 | execute | | write* |
| D_2 | execute | read* | execute |
| D_3 | execute | read | |

(b)

Access Matrix With *Owner* Rights

- Domain D1 is owner of F1 and Can add or delete any valid right in F1 column. Similarly D2 is owner of F2 and F3.

| object domain | F_1 | F_2 | F_3 |
|------------------|------------------|----------------|--------------------------|
| D_1 | owner execute | | write |
| D_2 | | read* owner | read* owner write* |
| D_3 | execute | | |

(a)

| object domain | F_1 | F_2 | F_3 |
|------------------|------------------|--------------------------|--------------------------|
| D_1 | owner execute | | |
| D_2 | | owner read* write* | read* owner write* |
| D_3 | | write | write |

(b)

Access Matrix: Switch control

A process executing in D2 could modify domain D4.

| object domain | F_1 | F_2 | F_3 | laser printer | D_1 | D_2 | D_3 | D_4 |
|------------------|-------|-------|---------|------------------|--------|--------|--------|-------------------|
| D_1 | read | | read | | | switch | | |
| D_2 | | | | print | | | switch | switch control |
| D_3 | | read | execute | | | | | |
| D_4 | write | | write | | switch | | | |

Confinement problem

- *The copy and owner rights provide us the mechanism to limit the propagation of access rights.*
 - *However, they do not give appropriate tools for preventing the propagation of information.*
 - The problem of guaranteeing that no information initially held in an object can migrate outside of its execution environment is called **the confinement problem**.
-
- **Confinement problem is unsolvable.**

Implementation of the Access Matrix

- *Global table of <domain, object, rights-set>*
 - ◆ *Table becomes large and additional I/O is needed*
- *Access list for every object*
 - ◆ *Each column can be implemented as a access list for the object. The list for each object consists of <Domain, rights-set>*
- *Capability List for domains*
 - ◆ *A capability list a domain is a list objects together with the operations allowed on those objects*

Capability-Based Systems

■ Hydra

- ◆ Fixed set of access rights known to and interpreted by the system.
- ◆ Interpretation of user-defined rights performed solely by user's program; system provides access protection for use of these rights.

■ Cambridge CAP System

- ◆ Data capability - provides standard read, write, execute of individual storage segments associated with object.
- ◆ Software capability -interpretation left to the subsystem, through its protected procedures.

Language-Based Protection

- Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources.
- Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable.
- Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system.

Protection in Java 2

- Protection is handled by the Java Virtual Machine (JVM)
- A class is assigned a protection domain when it is loaded by the JVM.
- The protection domain indicates what operations the class can (and cannot) perform.
- If a library method is invoked that performs a privileged operation, the stack is inspected to ensure the operation can be performed by the library.

Stack Inspection

| | | | |
|--------------------|---|--|---|
| protection domain: | untrusted applet | URL loader | networking |
| socket permission: | none | *.lucent.com:80, connect | any |
| class: | gui: ... get(url); open(addr); | get(URL u): ... doPrivileged { open('proxy.lucent.com:80'); } <request u from proxy> ... | open(Addr a): ... checkPermission(a, connect); connect (a); ... |

System Security

- The Security Problem
- Authentication
- Program Threats
- System Threats
- Securing Systems
- Intrusion Detection
- Encryption
- Windows NT

The Security Problem

- Security must consider external environment of the system, and protect it from:
 - ◆ unauthorized access.
 - ◆ malicious modification or destruction
 - ◆ accidental introduction of inconsistency.
 - ◆ These are management, rather than system, problems.
- Easier to protect against accidental than malicious misuse.
- We say that the system is secure if its resources are used and accessed as intended under all circumstances.

The Security Problem...

- Unfortunately total security can not be achieved.
- It is easier to protect against accidental misuse than malicious misuse.
- Intruder or cracker: attempt to breach the security
- Threat: potential of security violation such as discovery of vulnerability
- Attack: an attempt to break security
- Form of malicious access
 - ◆ Breach of confidentiality: Unauthorized reading of data (theft of information)
 - ◆ Breach of integrity: Unauthorized modification of data
 - ◆ Breach of availability: Unauthorized destruction of data
 - ◆ Theft of service: Unauthorized use of resources.
 - ◆ Denial of service: Preventing legitimate use of the system (denial of service)
- Absolute protection against malicious use is not possible, however cost of perpetrator can be sufficiently high to deter unauthorized attempts.

The Security Problem...

- Four levels of security measures must be taken.
 - ◆ Physical
 - ✓ Against armed or surreptitious entry by intruders.
 - ◆ Human
 - ✓ Careful screening of users to reduce the chance of unauthorized access.
 - ◆ Network
 - ✓ No one should intercept the data on the network.
 - ◆ Operating system
 - ✓ The system must protect itself from accidental or purposeful security breaches.
- A weakness at a high level of security allows circumvention of low-level measures.
- In the remainder of this chapter we discuss security at OS level

Security measures at OS level

■ User authentication

- ◆ Verifying the user's authentication

■ Program threats

- ◆ Misuse of programs unexpected misuse of programs.

■ System threats

- ◆ Worms and viruses

■ Intrusion detection

- ◆ Detect attempted intrusions or successful intrusions and initiate appropriate responses to the intrusions.

■ Cryptography

- ◆ Ensuring protection of data over network

Authentication

- User identity most often established through *passwords*, can be considered a special case of either keys or capabilities.
- Easy to understand and use
- However,
 - ◆ they can be easily guessed, illegally transferred
 - ◆ Visual or electronic monitoring, network sniffing
- Passwords must be kept secret.
 - ◆ Frequent change of passwords.
 - ◆ Use of “non-guessable” passwords.
 - ◆ Log all invalid access attempts.
- Passwords may also either be encrypted or allowed to be used only once.
 - ◆ For given a value of x , $f(x)$ is calculated and stored. It is difficult to guess x by seeing $f(x)$.
- One time passwords
 - ◆ Password is different for each session.
- Example: My mother's name is K...
 - ◆ Password “MmnisK”

Authentication...

■ Biometrics

- ◆ Palm and hand readers: temperature map, finger length, finger width and line patterns.
- ◆ Fingerprint readers.

■ Two factor authentication scheme

- ◆ Password plus fingerprint scan

Program Threats

■ Trojan Horse

- ◆ Code segment that misuses its environment.
- ◆ Exploits mechanisms for allowing programs written by users to be executed by other users.
 - ✓ Consider the use of “.” character in a search path. The “.” tells the shell to include the current directory. If the user sets current directory to a friends directory, the program runs in users domain and effects friends directory.

■ Trap Door

- ◆ The designer of the code might leave a hole in the software that only she is capable of using.
- ◆ Specific user identifier or password that circumvents normal security procedures.
- ◆ Could be included in a compiler.

■ Logic Bomb

- ◆ Security threat only under certain circumstances.

■ Stack and Buffer Overflow

- ◆ Exploits a bug in a program (overflow either the stack or memory buffers.)

Program Threats...

■ Stack and Buffer Overflow

- ◆ Exploits a bug in a program (overflow either the stack or memory buffers.)

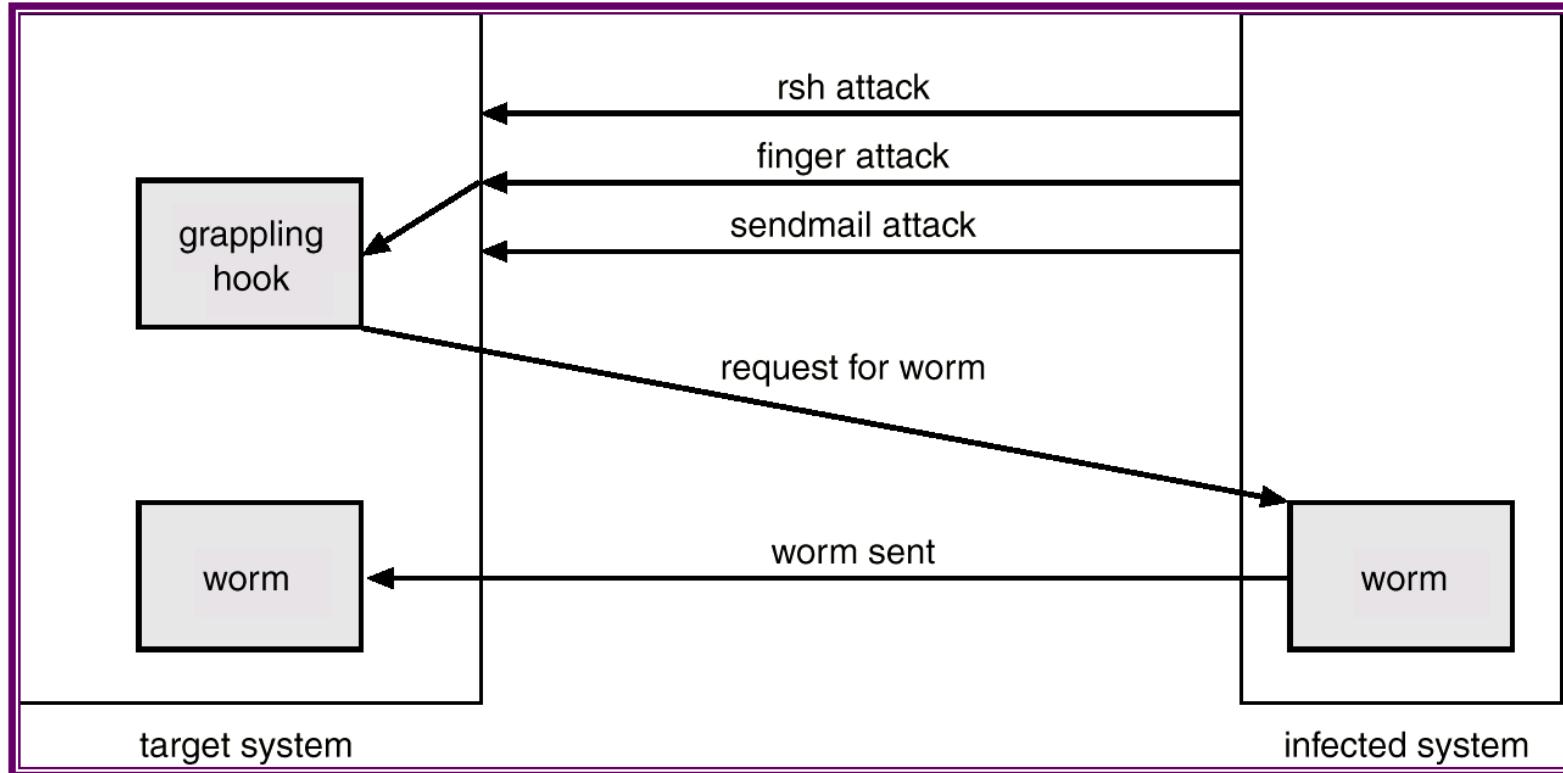
■ The attacker determines the vulnerability and writes a program to do the following.

- ◆ Overflow an input-field, command-line argument, or input buffer until it writes into the stack.
- ◆ Overwrite the current return address on the stack with the address of the exploit code in the next step.
- ◆ Write a simple set of code for the next space in the stack that includes commands that the attacker wishes to execute, for example, spawn a shell.

System Threats

- Worms – use spawn mechanism; standalone program
 - ◆ The worm spawns copies of itself, using up systems resources and perhaps locking out system use by all other processes.
- Internet worm
 - ◆ Exploited UNIX networking features (remote access) and bugs in *finger* and *sendmail* programs.
 - ◆ Grappling hook program uploaded main worm program.
- Viruses – fragment of code embedded in a legitimate program.
 - ◆ Mainly effect microcomputer systems.
 - ◆ Downloading viral programs from public bulletin boards or exchanging floppy disks containing an infection.
 - ◆ MSWORD (micros)
 - ✓ RTF format is safe
 - ◆ *Safe computing.*
- Denial of Service
 - ◆ Overload the targeted computer preventing it from doing any useful work.
 - ◆ Downloading of a page.
 - ◆ Partially started TCP/IP sessions could eat up all resources.
 - ◆ Difficult to prevent denial of service attacks.

The Morris Internet Worm



Threat Monitoring

- Check for suspicious patterns of activity – i.e., several incorrect password attempts may signal password guessing.
- Audit log – records the time, user, and type of all accesses to an object; useful for recovery from a violation and developing better security measures.
- Scan the system periodically for security holes; done when the computer is relatively unused.

Threat Monitoring (Cont.)

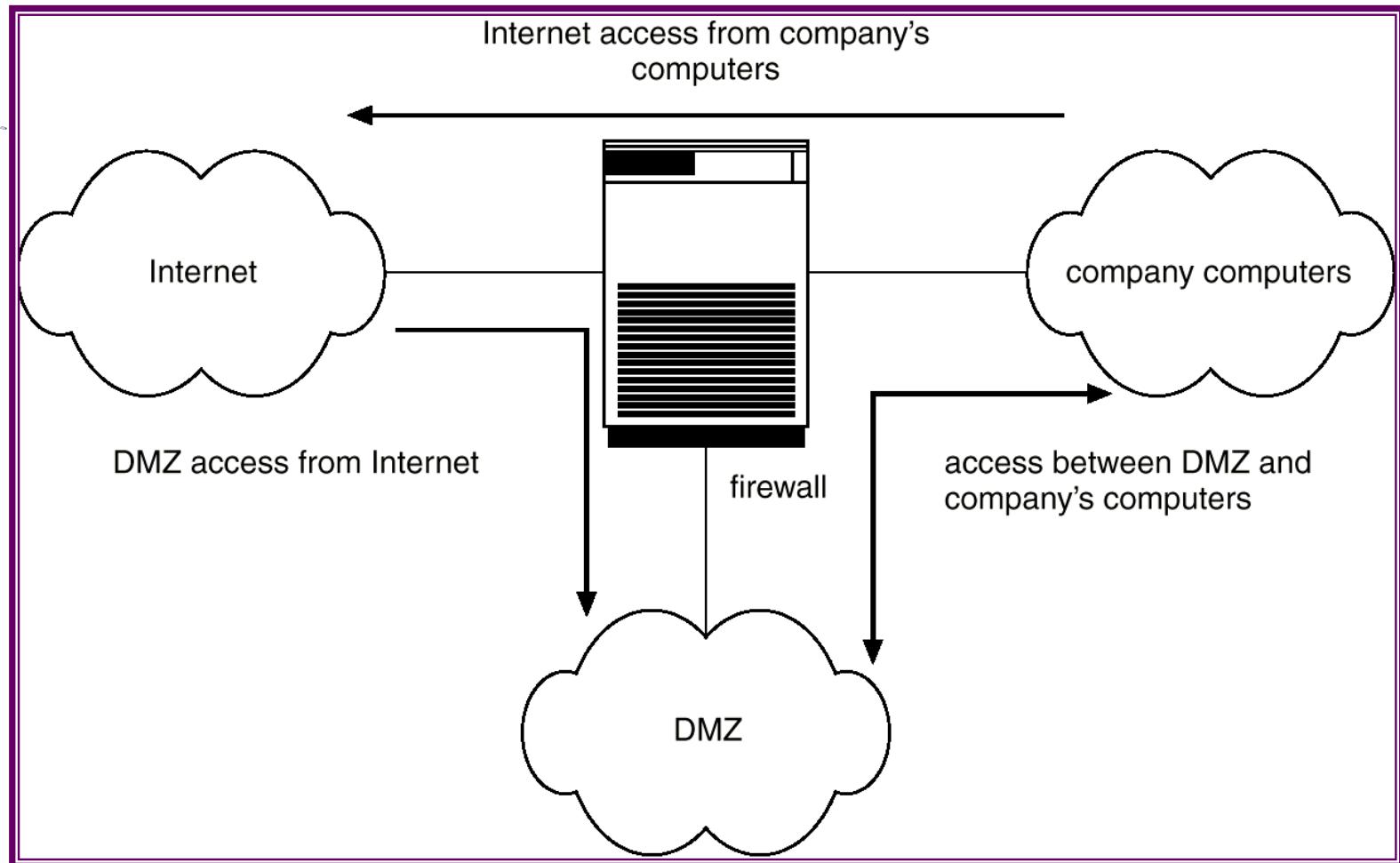
■ Check for:

- ◆ Short or easy-to-guess passwords
- ◆ Unauthorized set-uid programs
- ◆ Unauthorized programs in system directories
- ◆ Unexpected long-running processes
- ◆ Improper directory protections
- ◆ Improper protections on system data files
- ◆ Dangerous entries in the program search path (Trojan horse)
- ◆ Changes to system programs: monitor checksum values

FireWall

- A firewall is placed between trusted and untrusted hosts.
 - ◆ A firewall is a computer or router that sits between trusted and untrusted systems. It monitors and logs all connections.
- The firewall limits network access between these two security domains.
- **Spoofing:** An unauthorized host pretends to be an authorized host by meeting some authorization criterion.

Network Security Through Domain Separation Via Firewall



DMZ: Demilitarized zone

Intrusion Detection

- Detect attempts to intrude into computer systems.
- Wide variety of techniques
 - ◆ The time of detection
 - ◆ The type of inputs examined to detect intrusion activity
 - ◆ The range of response capabilities.
 - ✓ Alerting the administrator, killing the intrusion process, false resource is exposed to the attacker (but the resource appears to be real to the attacker) to gain more information about the attacker.
- The solutions are known as intrusion detection systems.
- Detection methods:
 - ◆ Auditing and logging.
 - ✓ Install logging tool and analyze the external accesses.
 - ◆ Tripwire (UNIX software that checks if certain files and directories have been altered – i.e. password files)
 - ✓ Integrity checking tool for UNIX.
 - ✓ It operates on the premise that a large class of intrusions results in anomalous modification of system directories and files.
 - ✓ It first enumerates the directories and files to be monitored for changes and deletions or additions. Later it checks for modifications by comparing signatures.
- System call monitoring
 - ◆ Detects when a process is deviating from expected system call behavior.

Data Structure Derived From System-Call Sequence

Open, read,mmap, mmap, open, getrlimit,mmap,close

| system call | distance = 1 | distance = 2 | distance = 3 |
|-------------|-----------------------|-------------------|-------------------|
| open | read getrlimit | mmap | mmap close |
| read | mmap | mmap | open |
| mmap | mmap open close | open getrlimit | getrlimit mmap |
| getrlimit | mmap | close | |
| close | | | |

Open, read,mmap, open, open, getrlimit,mmap,close

Cryptography

- Eliminate the need to trust the network.
- Cryptography enables a recipient of a message to verify that the message was created by some computer possessing a certain key.
- Keys are designed to be computationally infeasible to derive from the messages.

Encryption

- Encrypt clear text into cipher text.
- Properties of good encryption technique:
 - ◆ Relatively simple for authorized users to encrypt and decrypt data.
 - ◆ Encryption scheme depends not on the secrecy of the algorithm but on a parameter of the algorithm called the encryption key.
 - ◆ Extremely difficult for an intruder to determine the encryption key.
- *Data Encryption Standard* substitutes characters and rearranges their order on the basis of an encryption key provided to authorized users via a secure mechanism. Scheme only as secure as the mechanism.
- RSA : public/private key algorithm is popular

Computer Networks

Introduction

FIFTH EDITION

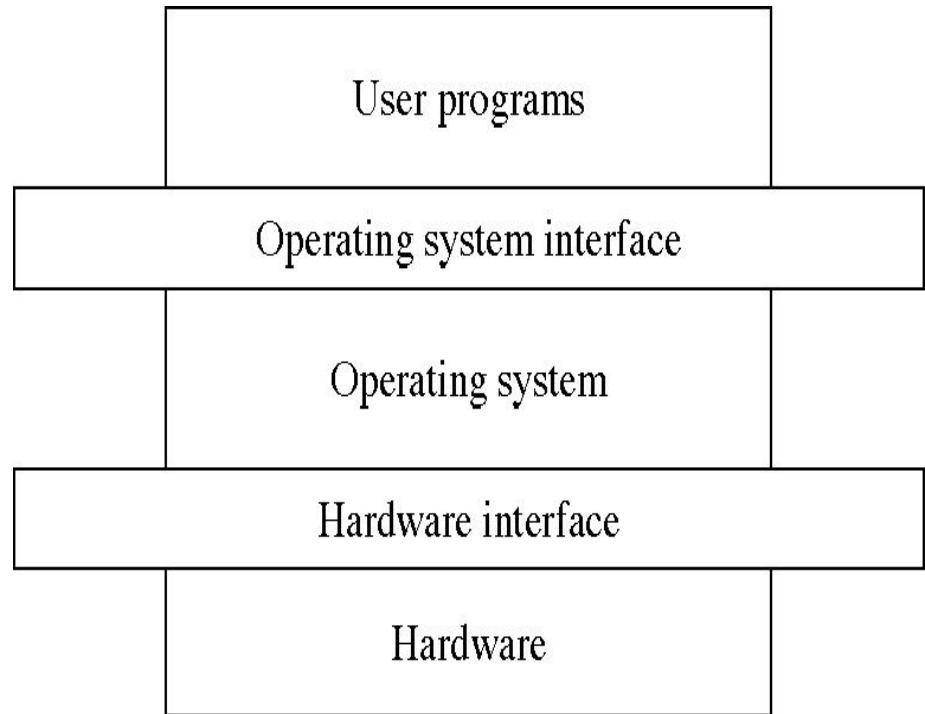
COMPUTER NETWORKS



TANENBAUM | WETHERALL

Topics completed: Operating system

- Input: hardware
-
- Output is system calls to develop applications
- Objectives: to improve throughput and response time, deadlines, waiting time



Topics to be completed: Networking

- Input: thousands of computers connected through communication channels
- Output is primitives to develop applications
- Objectives: to improve throughput, reliability, and response time, deadlines, waiting time

Networking Applications
(http, sendmail,...)

Networking Protocols (Interface)

Tens of thousands of computers and devices connected through communication channels distributed across the world
(thousands of kilometers)

How it is different from producer and consumer problem

- Producer places an item and consumer consumes an item from the buffer.
- Normally in the same computer
- Producer waits if the buffer is full, consumer waits if buffer is empty.
- How to enable producer and consumer to exchange data, suppose a video if
- Producer is in one computer
- Consumer is another computer
- Connected through long communication channel (thousands of kilometers)
- Issues
 - Links might not be reliable
 - All the data may not be delivered
 - So, More mechanisms are needed.

Layered model

Layered model

Divide the roles/issues among layers

Develop each layer

Lower layer provides services to higher layer.

Structure view: layered system design

A general philosophy that builds on the above approach

Decompose functionality into layers such that

Hardware is level 0, and layer t accesses functionality at layer $(t-1)$ or less

Access via appropriately defined system calls.

Advantages

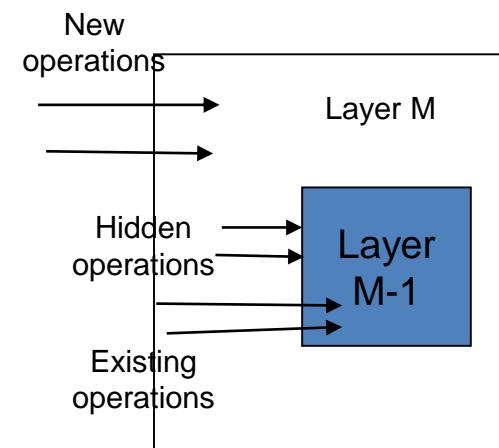
Modular design: well defined interfaces between layers

Prototyping/development

Association between function and layer eases overall OS design.

OS development and debugging is layer by layer.

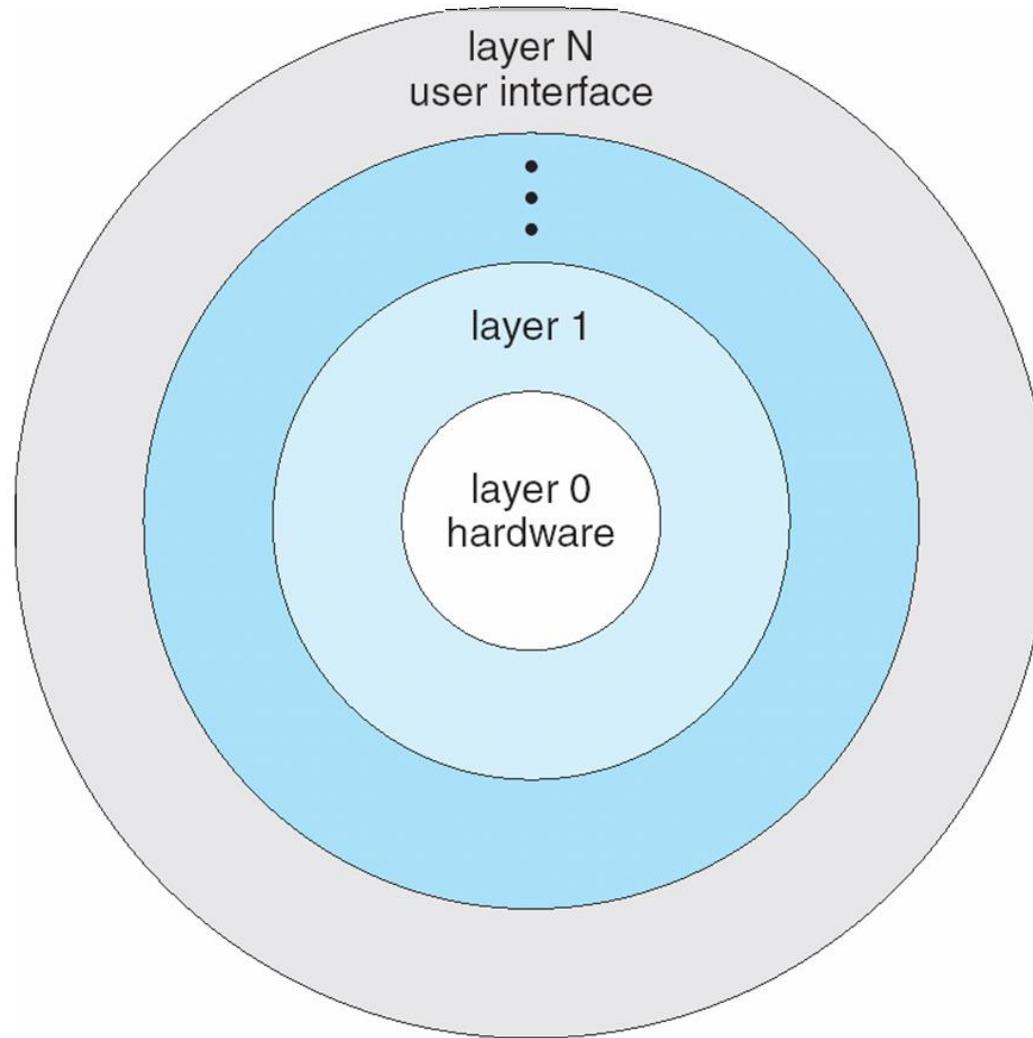
Simplifies debugging and system verification



Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

Layered Operating System



OS design hierarchy: hypothetical model

| Level | Name | Objects | Example operations |
|-------|---|---|--|
| 15 | Shell | User programming environment | Statements in shell language |
| 14 | Directories (maintains association between external and internal identifiers of system resources and objects) | Directories | Create, destroy, search, list, access rights |
| 13 | User Processes | User process | Fork, quit, kill, suspend, resume |
| 12 | Stream I/O | streams | open., close |
| 11 | Devices (access to external devices) | Printers, displays, and key boards | Create, destroy, open, close, read, write |
| 10 | File system (long storage of named files) | files | Create, destroy, open, close |
| 9 | Communications | pipes | Create, destroy, open, close, read, write |
| 8 | Capabilities | Capabilities | Create, Validate, Attenuate |
| 7 | Virtual memory (creating logical address space for programs) | Segments, pages | Read, write, fetch |
| 6 | Local secondary storage (position of read/write heads) | Blocks of data, device channels | Read, write, allocate, free |
| 5 | Primitive processes | Primitive process, semaphores, synchronization primitives | Suspend, resume, wait, signal |
| 4 | Interrupts | Interrupts handling programs | Invoke, mask, unmask, retry |
| 3 | Procedures | Procedures, call stack | Mark stack, call, return |
| 2 | Instruction set | Evaluation, stack, micro-program, interpreter | Load, store, add, subtract, branch |
| 1 | Electronic circuits | Registers, gates, busses | Clear, transfer, activate, complement |

Overview of this chapter

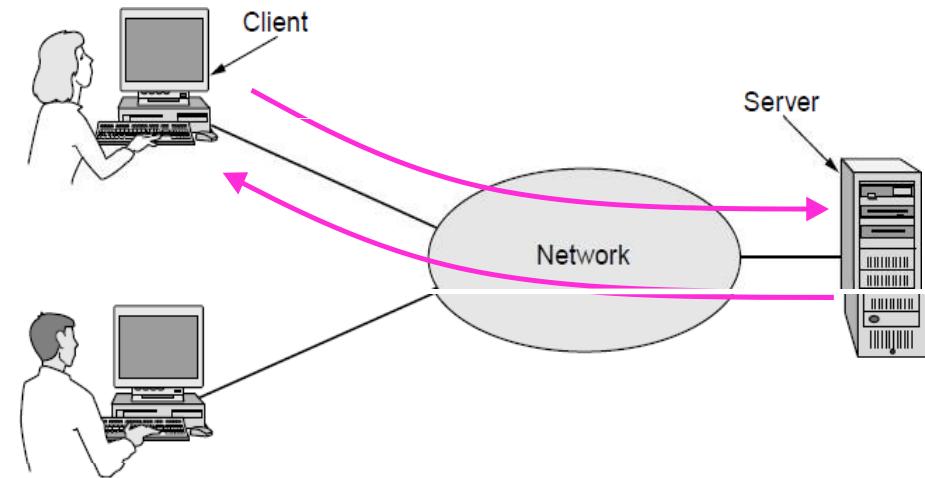
- Introduction
- Uses of Computer Networks
- Network Hardware
- Network Software
- Reference Models
- Example Networks
- Network Standardization
- Metric Units

Introduction

- 18th century is dominated by mechanical systems
- 19th century is the age of steam engine
- 20th century- Information Technology
 - Information gathering processing and distribution
 - Others: radio, television, telephones, communication satellites and Internet
- Computer network
 - Large number of separate but interconnected computers.
- Distributed system: A collection of independent computers which appears to users as single coherent system. Another layer such as world wide web is presented on the top of computer network
- In computer network, users are exposed to machines.

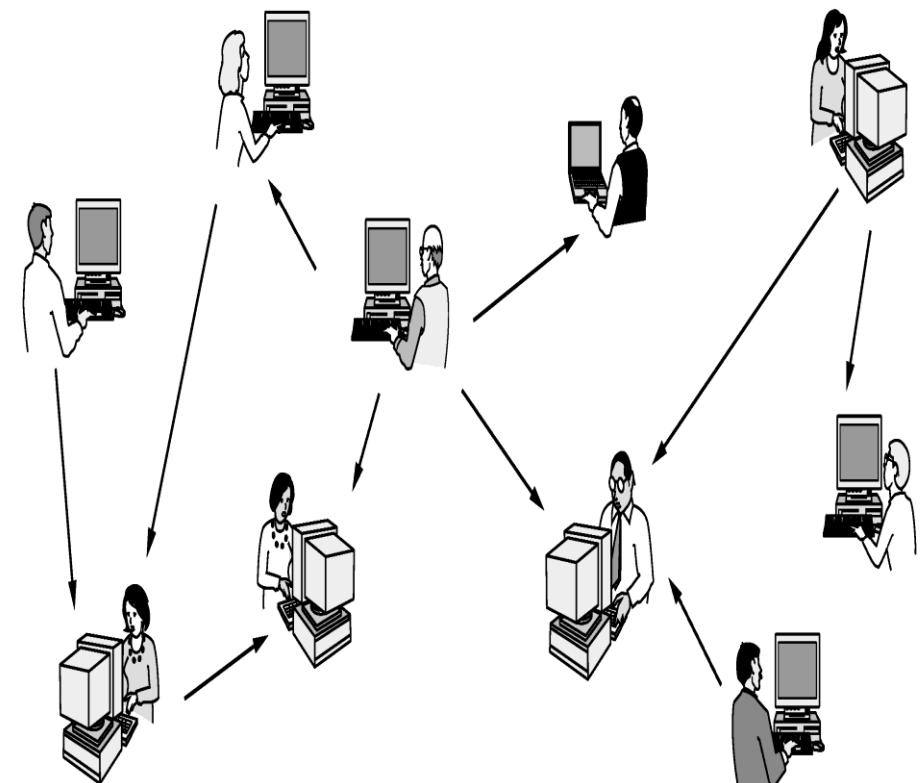
Users of computer networks

- Business applications
 - Resource sharing
 - To make programs, equipment, and data available to anyone on the network without regard to the location of the user.
 - Almost every office requires computer networks



Home Applications

- Homes contain many networked devices, e.g., computers, TVs, connected to the Internet by cable, DSL, wireless, etc.
- Home users communicate, e.g., social networks, consume content, e.g., video, and transact, e.g., auctions
 - Wikipedia, facebook, twitter, sms, IP television
- Some application use the peer-to-peer model in which there are no fixed clients and servers:



In a peer-to-peer system there are no fixed clients and servers.

| Tag | Full name | Example |
|------------|------------------------|--|
| B2C | Business-to-consumer | Ordering books online |
| B2B | Business-to-business | Car manufacturer ordering tires from supplier |
| G2C | Government-to-consumer | Government distributing tax forms electronically |
| C2C | Consumer-to-consumer | Auctioning second-hand products online |
| P2P | Peer-to-peer | Music sharing |

Some forms of e-commerce.

Mobile Users

- Tablets, laptops, and smart phones are popular devices; WiFi hotspots and 3G cellular provide wireless connectivity.
- Mobile users communicate, e.g., voice and texts, consume content, e.g., video and Web, and use sensors, e.g., GPS.
- Sensor networks
- Wireless and mobile are related but different:

| Wireless | Mobile | Typical applications |
|----------|--------|--|
| No | No | Desktop computers in offices |
| No | Yes | A notebook computer used in a hotel room |
| Yes | No | Networks in unwired buildings |
| Yes | Yes | Store inventory with a handheld computer |

Social Issues

- Network neutrality – no network restrictions
- Content ownership, e.g., DMCA (Digital Millennium Copyright Act) takedowns
- Anonymity and censorship
- Privacy, e.g., Web tracking and profiling
- Theft, e.g., botnets and phishing

Network Hardware

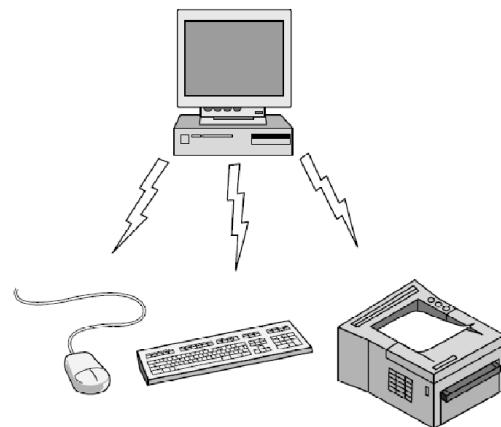
| Scale | Type |
|----------|--|
| Vicinity | PAN (Personal Area Network) » |
| Building | LAN (Local Area Network) » |
| City | MAN (Metropolitan Area Network) » |
| Country | WAN (Wide Area Network) » |
| Planet | The Internet (network of all networks) |

About networking technology

- Two dimensions
 - Transmission technology
 - Scale
- Two types of transmission technology
 - Broadcast links
 - Point to point links
- Point-to-Point links
 - One sender and one receiver
 - Connect individual pairs of links
 - They may have to visit some intermediate machines.
- Broadcast link
 - Communication link is shared by several machines
 - A packet is received by all machines. The address in the packet specifies intended recipient.
 - Broadcasting: message is sent to all machines
 - Multicasting: message is sent to a subset of nodes.

Personal Area Network

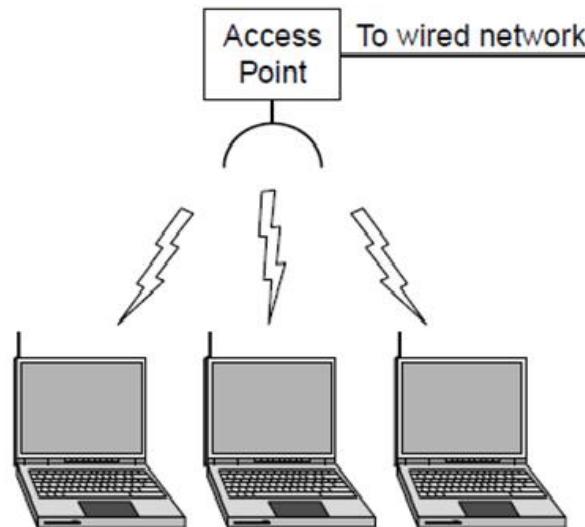
- Connect devices over the range of a person
- Example of a Bluetooth (wireless) PAN:



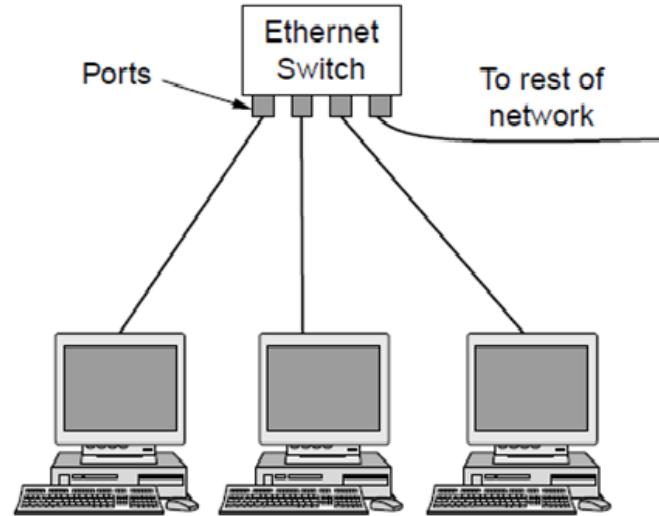
Local Area Networks

Connect devices in a home or office building

Called enterprise network in a company



Wireless LAN
with 802.11

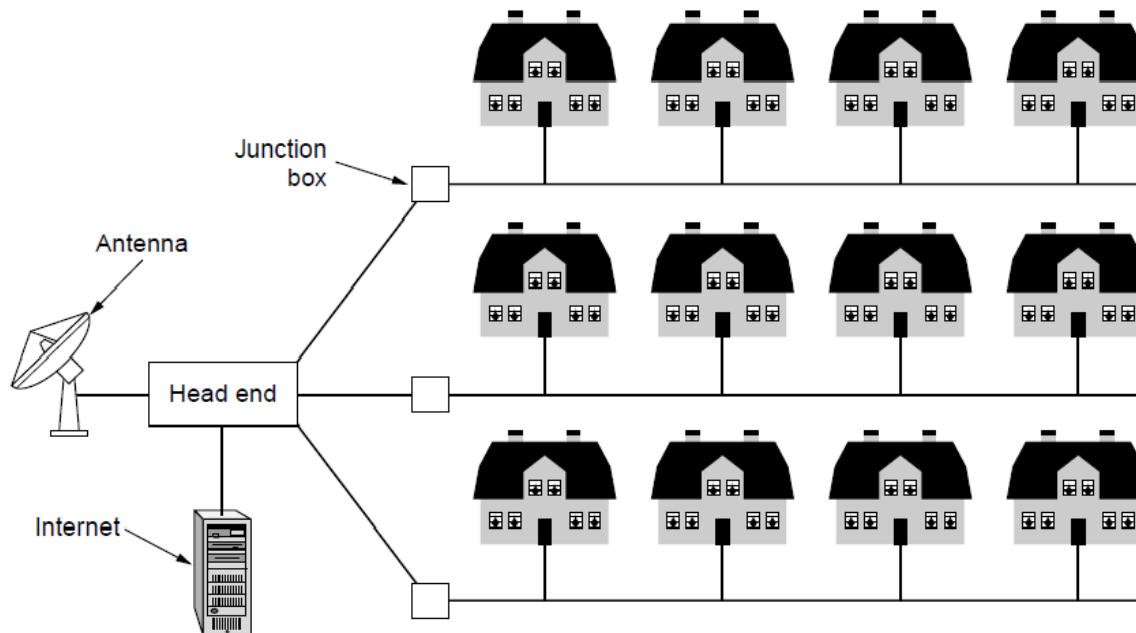


Wired LAN with
switched Ethernet

Metropolitan Area Networks

Connect devices over a metropolitan area

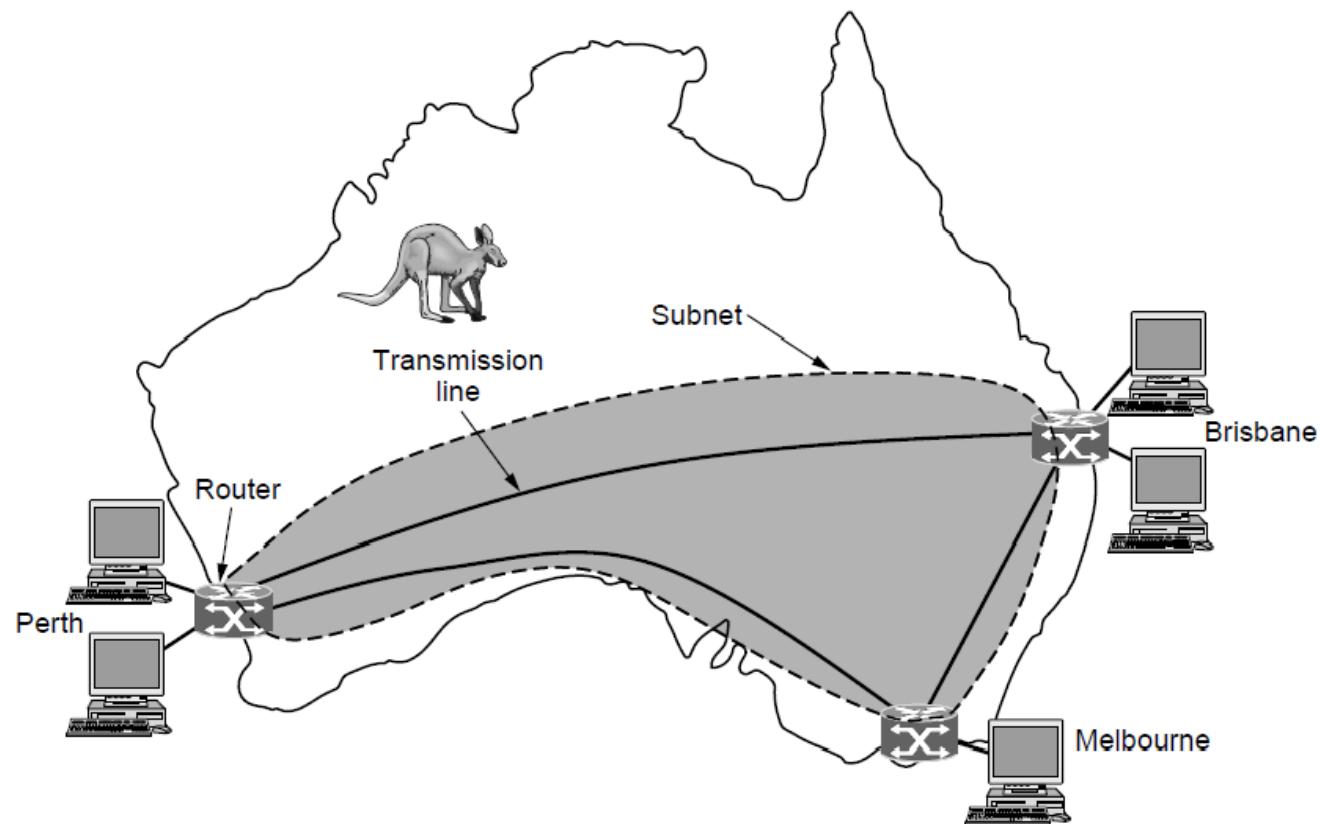
Example MAN based on cable TV:



Wide Area Networks (1)

Connect devices over a country

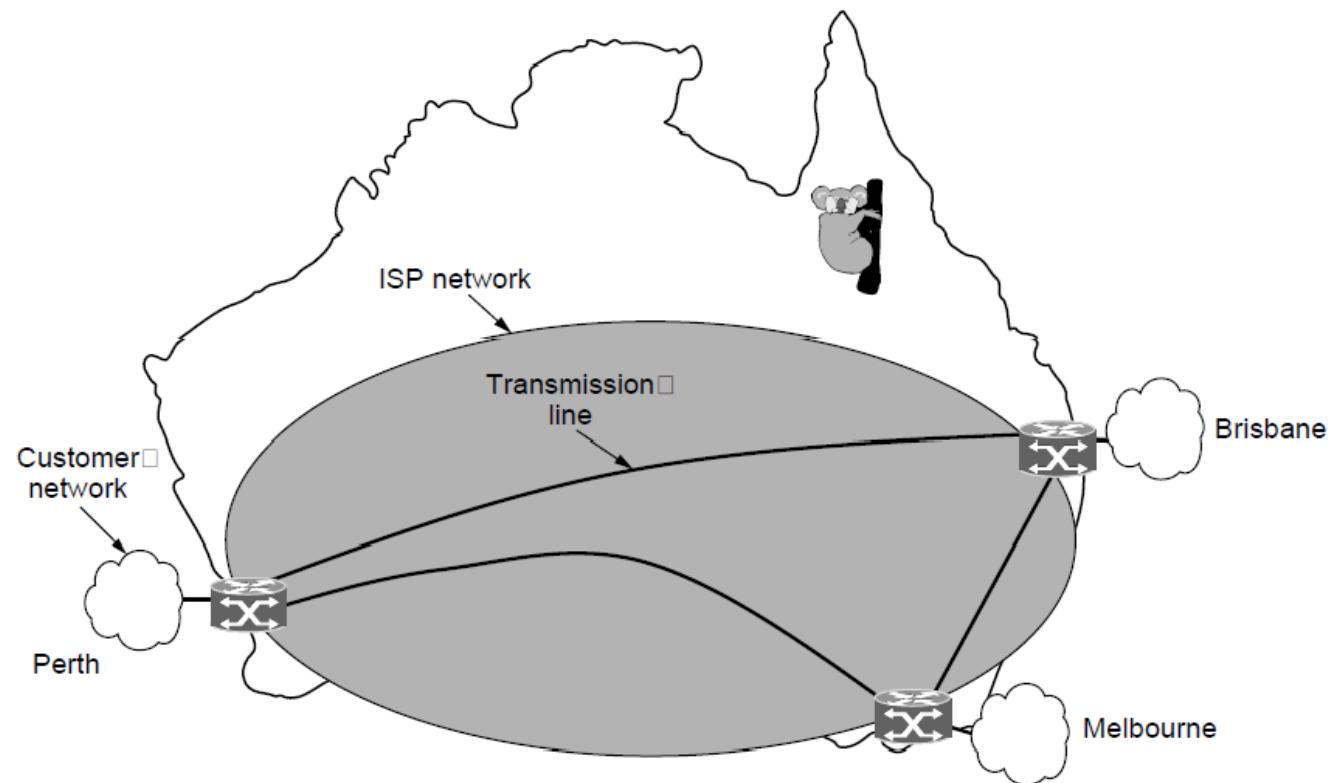
Example WAN connecting three branch offices:



Wide Area Networks (2)

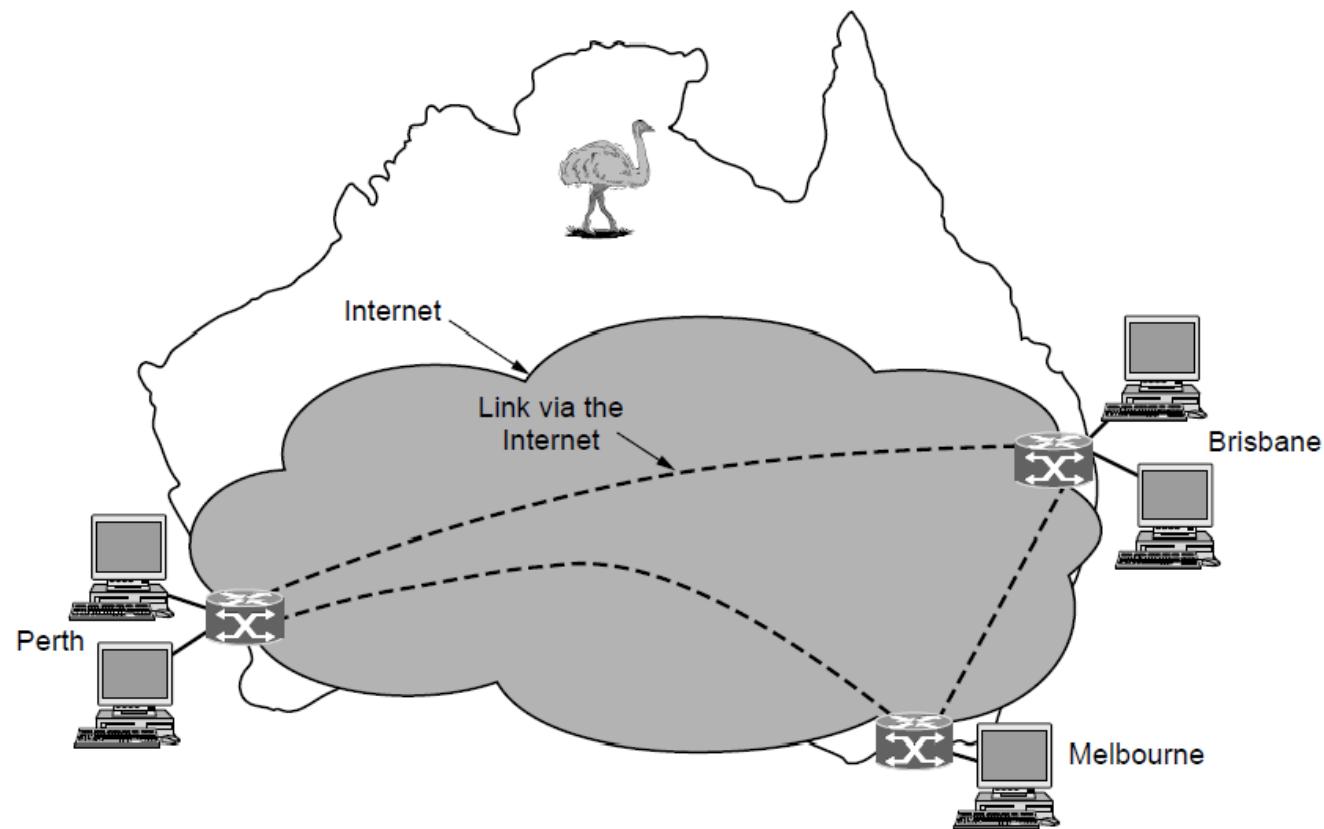
An ISP (Internet Service Provider) network is also a WAN.

Customers buy connectivity from the ISP to use it.



Wide Area Networks (3)

A VPN (Virtual Private Network) is a WAN built from virtual links that run on top of the Internet.



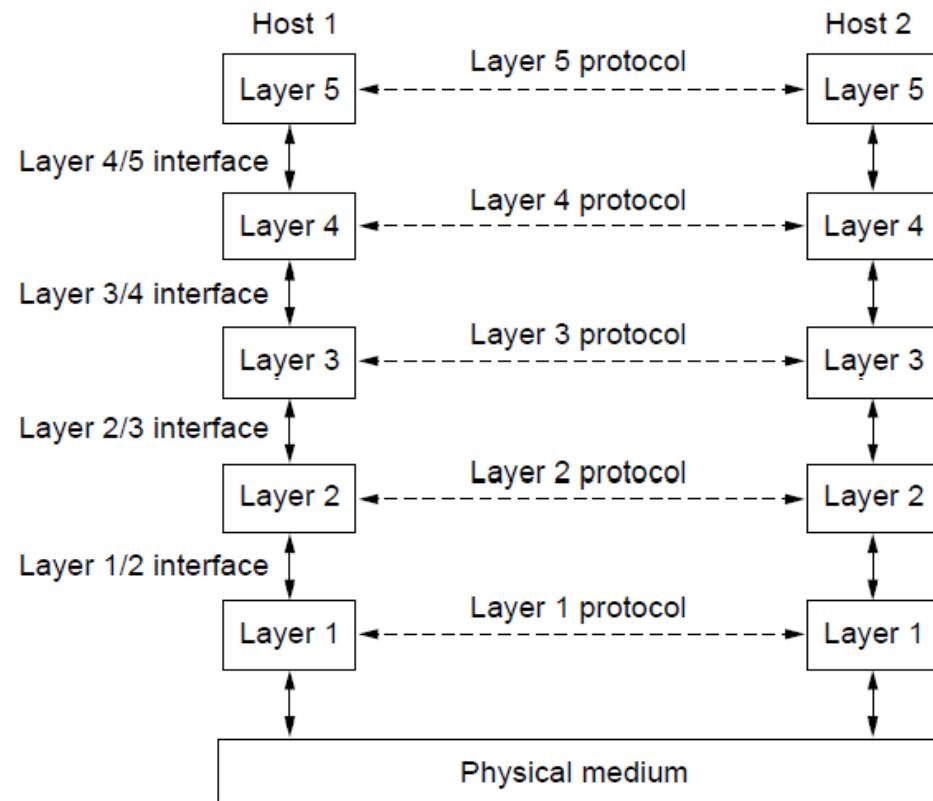
Network Software

- Protocol layers »
- Design issues for the layers »
- Connection-oriented vs. connectionless service »
- Service primitives »
- Relationship of services to protocols »

Protocol Layers (1)

Protocol layering is the main structuring method used to divide up network functionality.

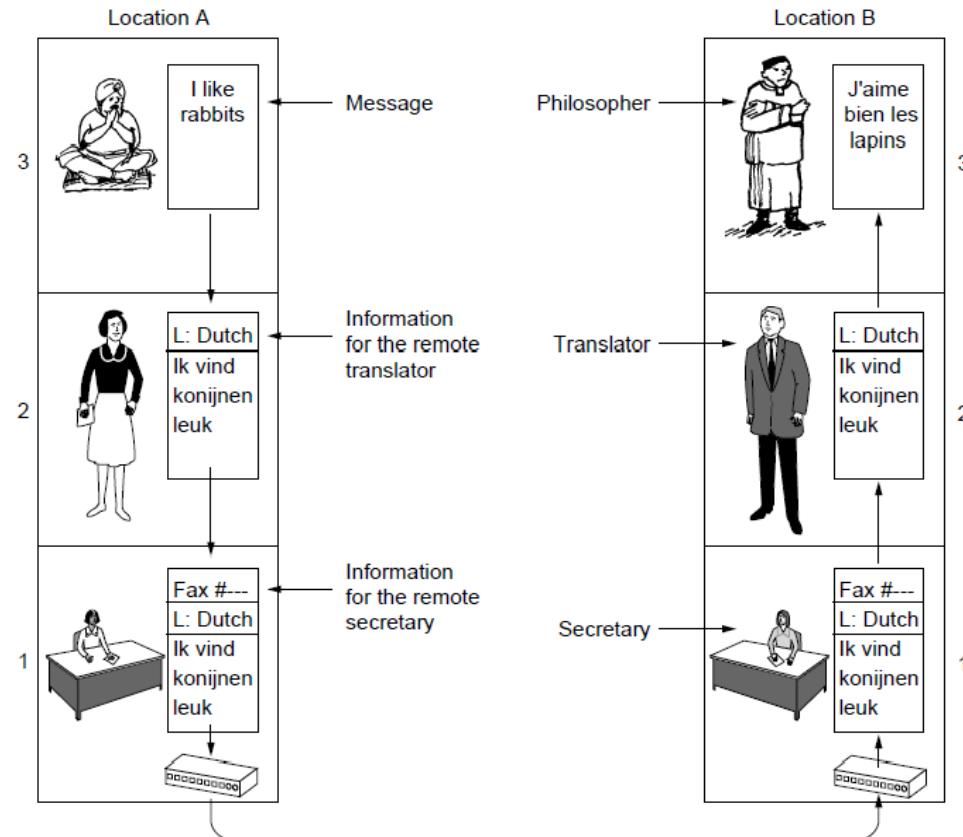
- Each protocol instance talks virtually to its peer
- Each layer communicates only by using the one below
- Lower layer services are accessed by an interface
- At bottom, messages are carried by the medium



Protocol Layers (2)

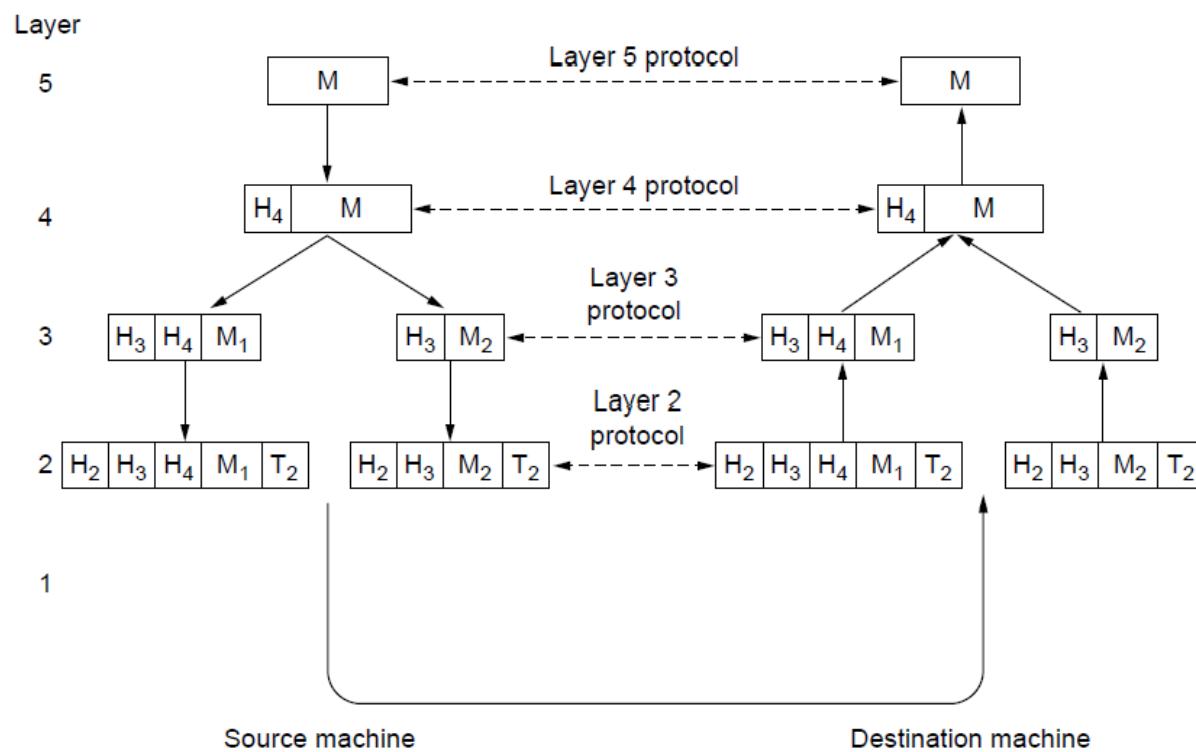
Example: the philosopher-translator-secretary architecture

Each protocol at different layers serves a different purpose



Protocol Layers (3)

Each lower layer adds its own header (with control information) to the message to transmit and removes it on receive



Layers may also split and join messages, etc.

Design Issues for the Layers

Each layer solves a particular problem but must include mechanisms to address a set of recurring design issues

| Issue | Example mechanisms at different layers |
|--|---|
| Reliability despite failures | Codes for error detection/correction (§3.2, 3.3) Routing around failures (§5.2) |
| Network growth and evolution | Addressing (§5.6) and naming (§7.1) Protocol layering (§1.3) |
| Allocation of resources like bandwidth | Multiple access (§4.2) Congestion control (§5.3, 6.3) |
| Security against various threats | Confidentiality of messages (§8.2, 8.6) Authentication of communicating parties (§8.7) |

Connection-Oriented vs. Connectionless

Service provided by a layer may be kinds of either:

- Connection-oriented, must be set up for ongoing use (and torn down after use), e.g., phone call
- Connectionless, messages are handled separately, e.g., postal delivery

| | Service | Example |
|---------------------|-------------------------|----------------------|
| Connection-oriented | Reliable message stream | Sequence of pages |
| | Reliable byte stream | Movie download |
| | Unreliable connection | Voice over IP |
| Connection-less | Unreliable datagram | Electronic junk mail |
| | Acknowledged datagram | Text messaging |
| | Request-reply | Database query |

Service Primitives (1)

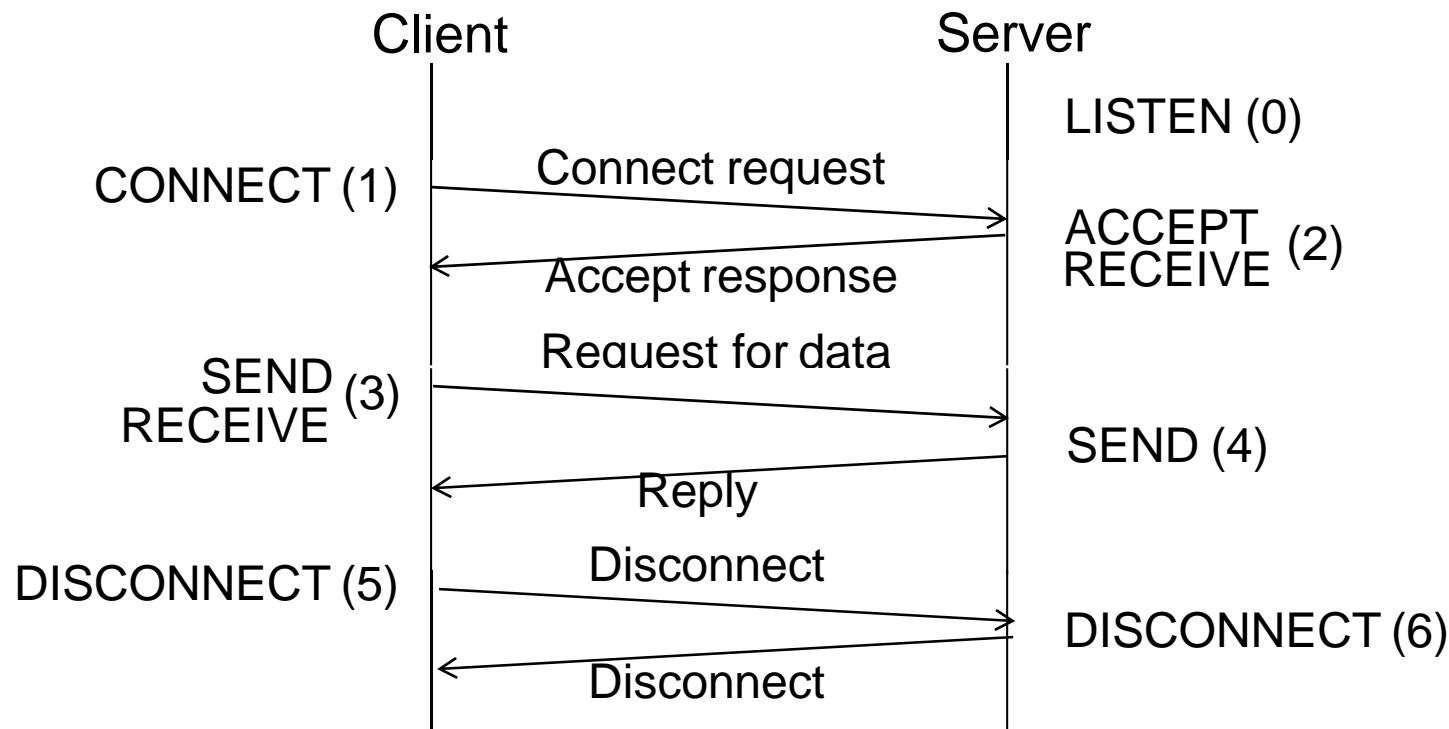
A service is provided to the layer above as primitives

Hypothetical example of service primitives that may provide a reliable byte stream (connection-oriented) service:

| Primitive | Meaning |
|------------|--|
| LISTEN | Block waiting for an incoming connection |
| CONNECT | Establish a connection with a waiting peer |
| ACCEPT | Accept an incoming connection from a peer |
| RECEIVE | Block waiting for an incoming message |
| SEND | Send a message to the peer |
| DISCONNECT | Terminate a connection |

Service Primitives (2)

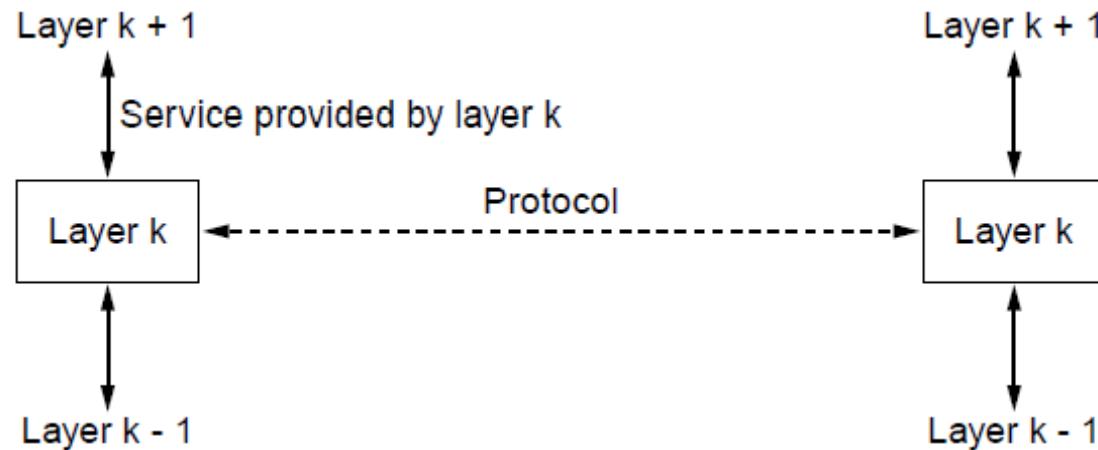
Hypothetical example of how these primitives may be used for a client-server interaction



Relationship of Services to Protocols

Recap:

- A layer provides a service to the one above [vertical]
- A layer talks to its peer using a protocol [horizontal]



Reference Models

Reference models describe the layers in a network architecture

- OSI reference model »
- TCP/IP reference model »
- Model used for this text »
- Critique of OSI and TCP/IP »

OSI (Open Systems Interconnection) Reference Model

A principled, international standard, seven layer model to connect different systems

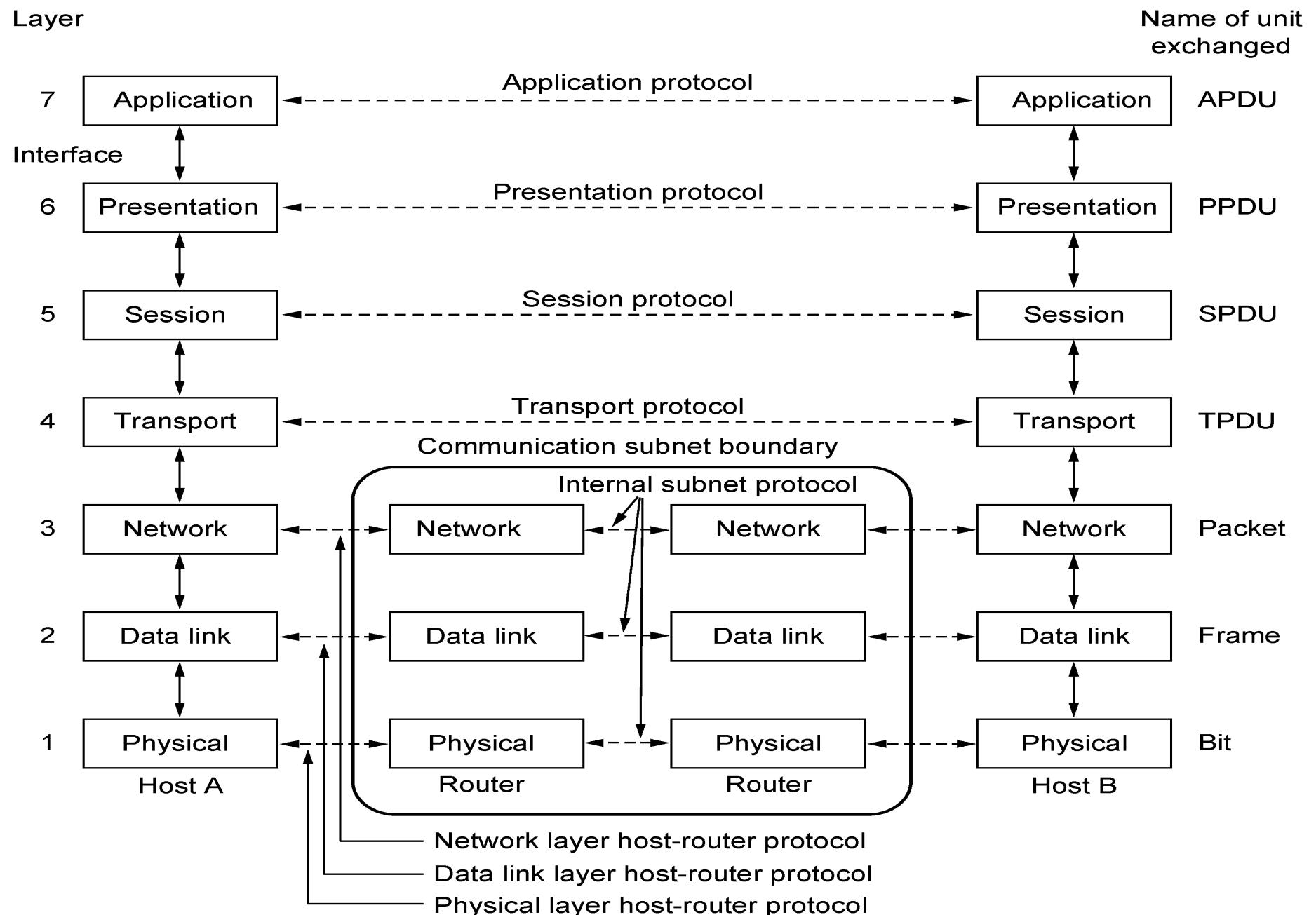
| | | |
|---|--------------|--------------------------------------|
| 7 | Application | – Provides functions needed by users |
| 6 | Presentation | – Converts different representations |
| 5 | Session | – Manages task dialogs |
| 4 | Transport | – Provides end-to-end delivery |
| 3 | Network | – Sends packets over multiple links |
| 2 | Data link | – Sends frames of information |
| 1 | Physical | – Sends bits as signals |

About Layers

- The Physical layer
 - Transmitting raw bits over a communication channel
 - One side sends 1/0 and the other side receives 1/0
 - Issue: What electrical signals should be used?
 - Design issues
 - Mechanical, electrical and timing interfaces
- The Data Link Layer
 - Transform a raw transmission facility into a like that appears free of undetected transmission errors.
 - Data frame and ack. Frame
 - Interface between fast transmitter and slow receiver
 - Broadcast networks: How to control access to shared channel?
 - LAN
- The network layer
 - Controls the operation of the subnet.
 - Issue: how packets are routed from source to destination?
 - Resolving bottlenecks; concession control
 - In broadcast networks, network layer is absent

About Layers

- The Transport layer
 - Accept data split into small units, if required, and pass it to network layer.
 - It isolates upper layer with the possible changes in the hardware technology.
 - It also determines the type of service: connection oriented or connectionless.
 - It is an end to end layer
 - One computer talks with other computer
 - In the lower layers the communication is between the source and intermediate computer.
- The Session Layer
 - Helps to establish sessions
 - Dialog control, token management, synchronization
- The presentation layer
 - Syntax and semantics of the information
 - It allows computers with different data representations to communicate
- The application layer
 - Set of protocols
 - Popular one is HTTP (Hypertext transfer protocol)
 - File transfer
 - Electronic mail
 - Network news

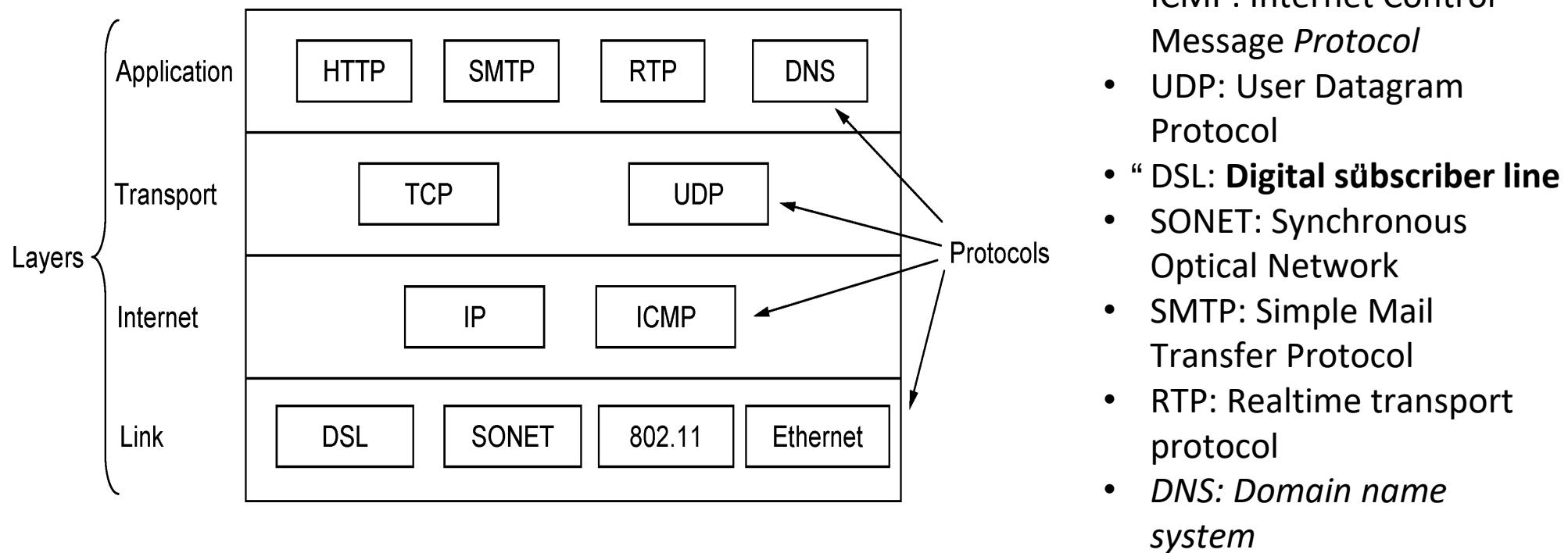


The OSI reference model.

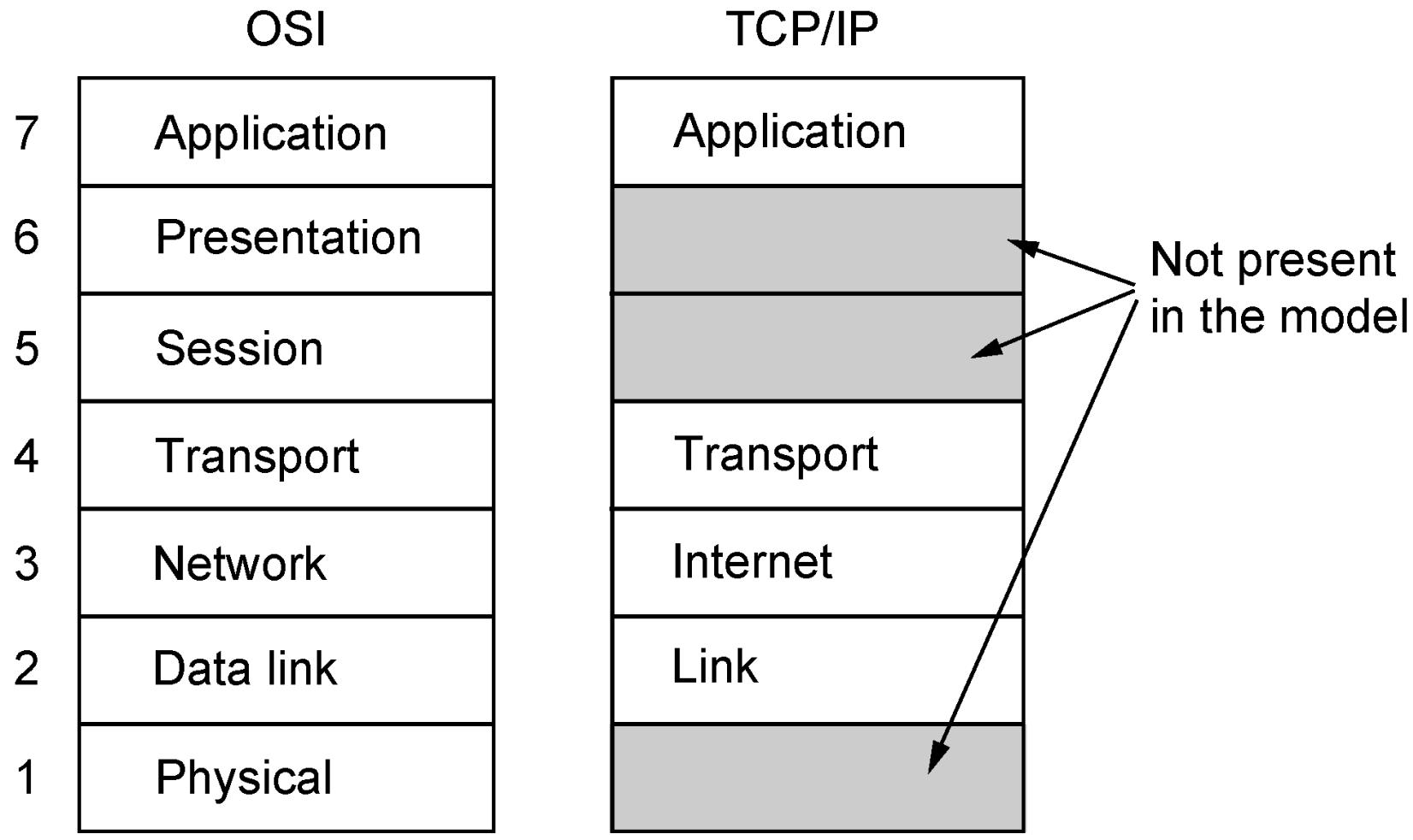
TCP/IP Reference Model

A four layer model derived from experimentation; omits some OSI layers and uses the IP as the network layer.

TCP/IP stands for Transmission Control Protocol/Internet Protocol and is a suite of communication protocols used to interconnect network devices on the internet.



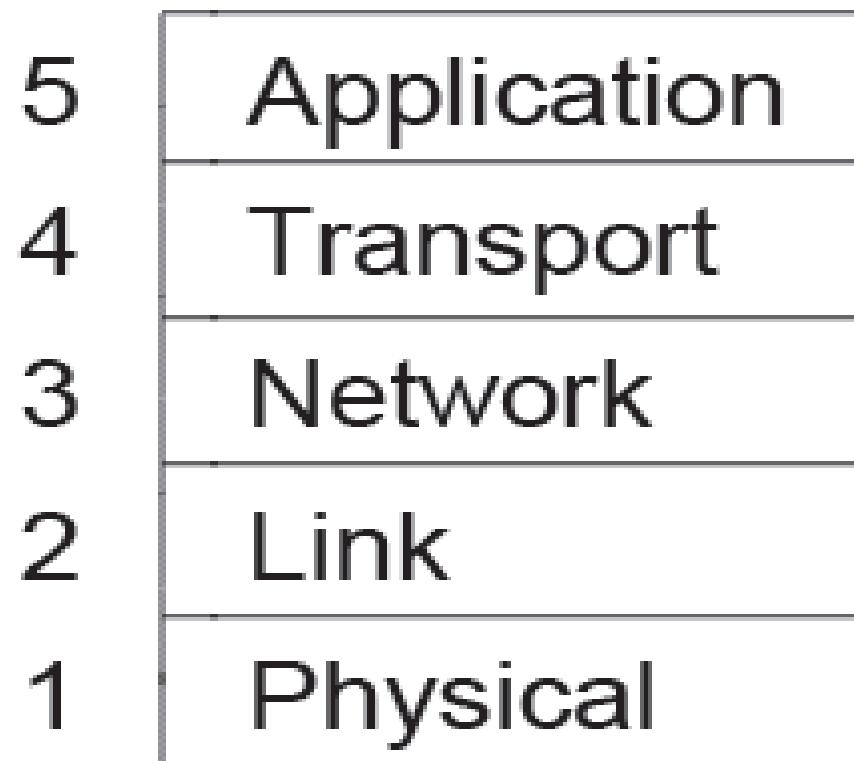
The TCP/IP model with some protocols we will study.



The TCP/IP reference model.

Model Used in this Book

It is based on the TCP/IP model but we call out the physical layer and look beyond Internet protocols.



Critique of OSI & TCP/IP

OSI:

- + Very influential model with clear concepts
- Models, protocols and adoption all bogged down by politics and complexity

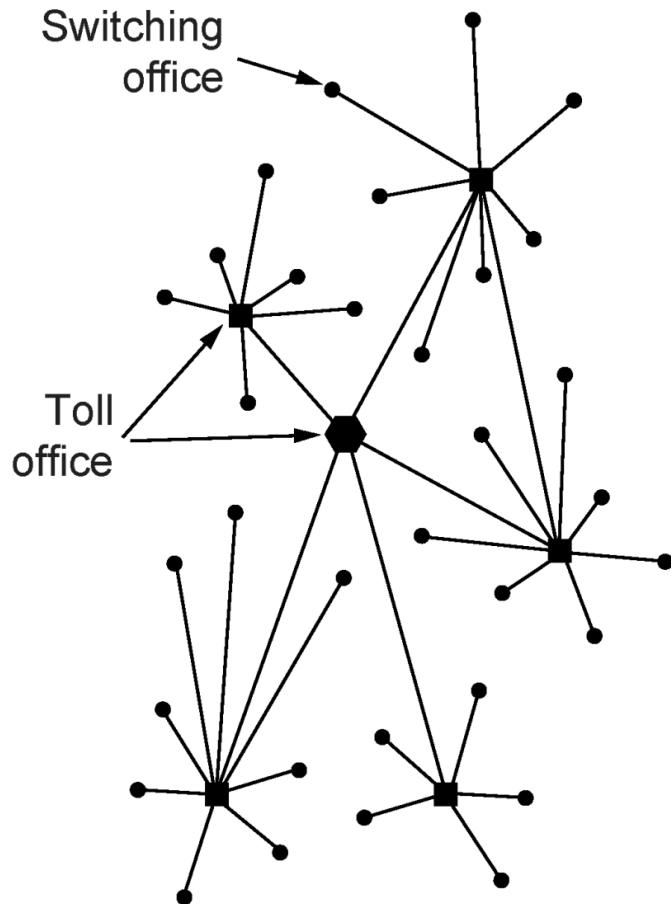
TCP/IP:

- + Very successful protocols that worked well and thrived
- Weak model derived after the fact from protocols

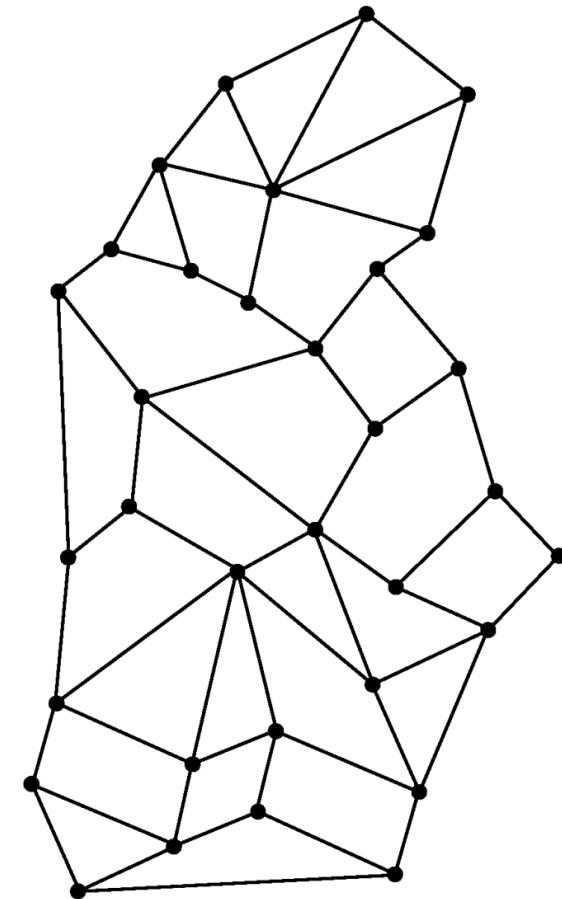
Example Networks

- The Internet »
- 3G mobile phone networks »
- Wireless LANs »
- RFID and sensor networks »

Internet



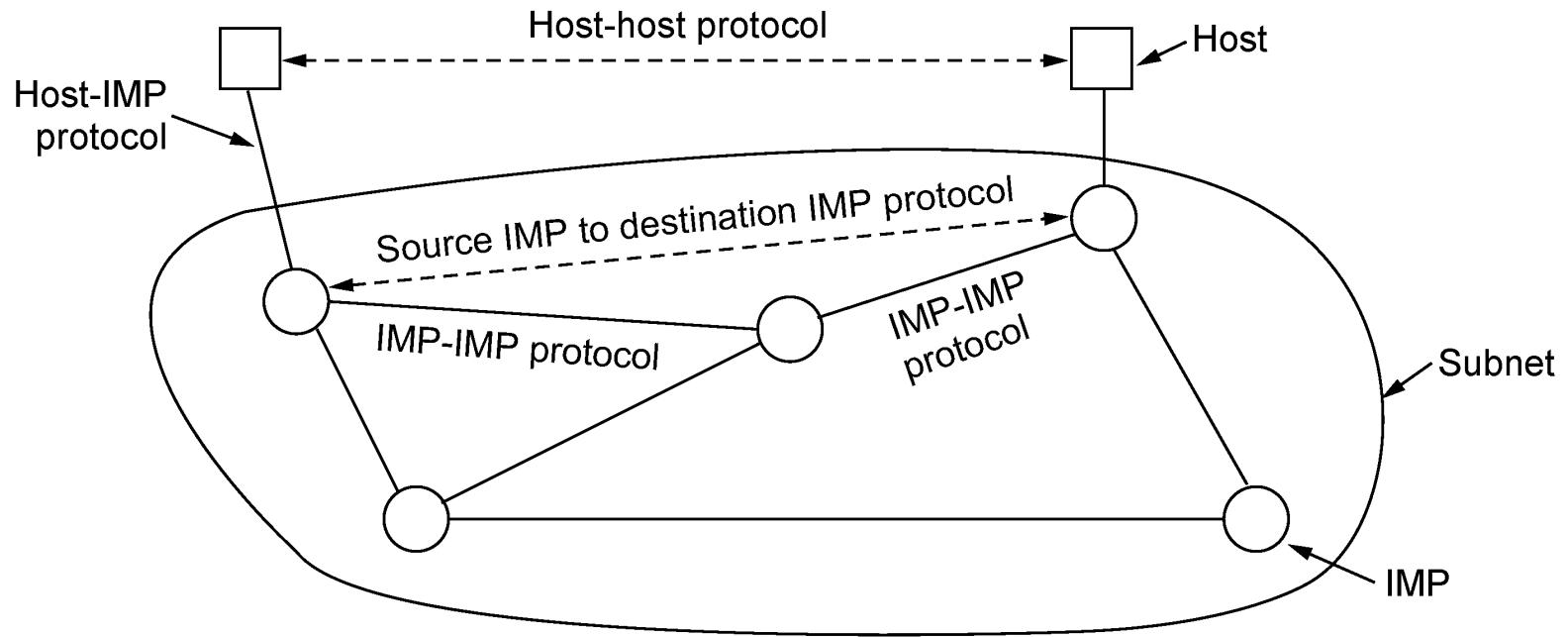
(a)



(b)

(a) Structure of the telephone system. (b) Baran's proposed distributed switching system.

Internet

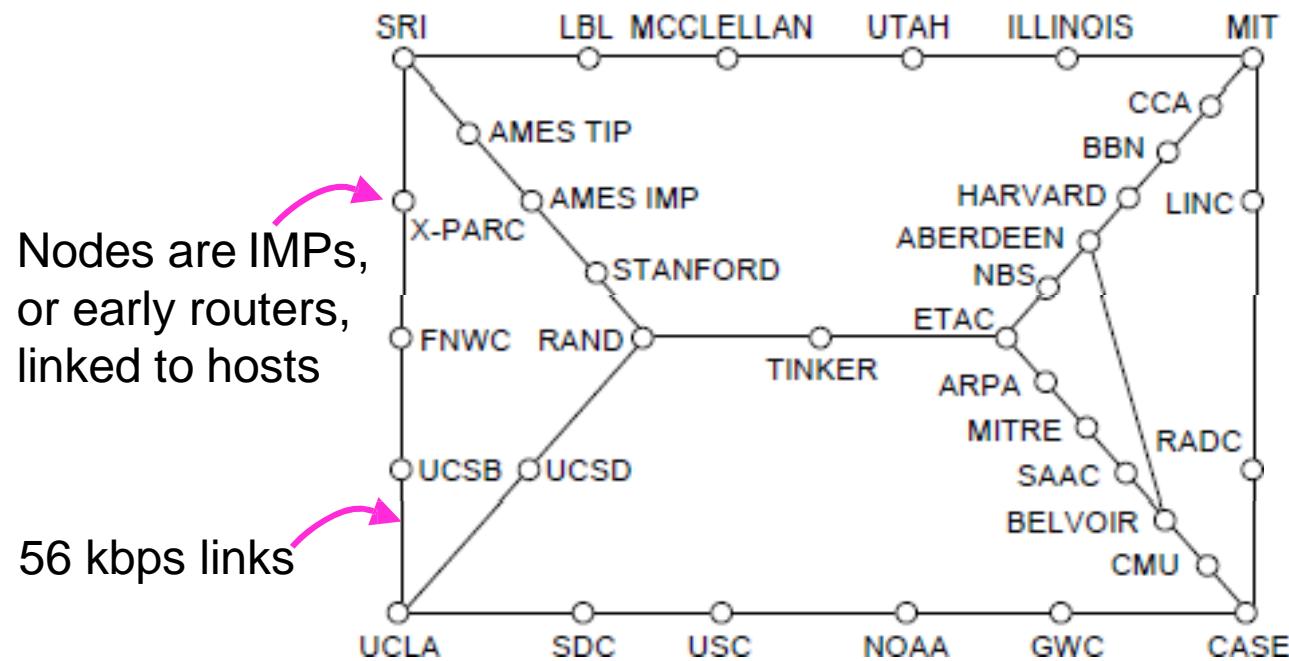


The original ARPANET design.

ARPANET: Advanced Research Projects Agency Network
IMP: Interface message processors

Internet

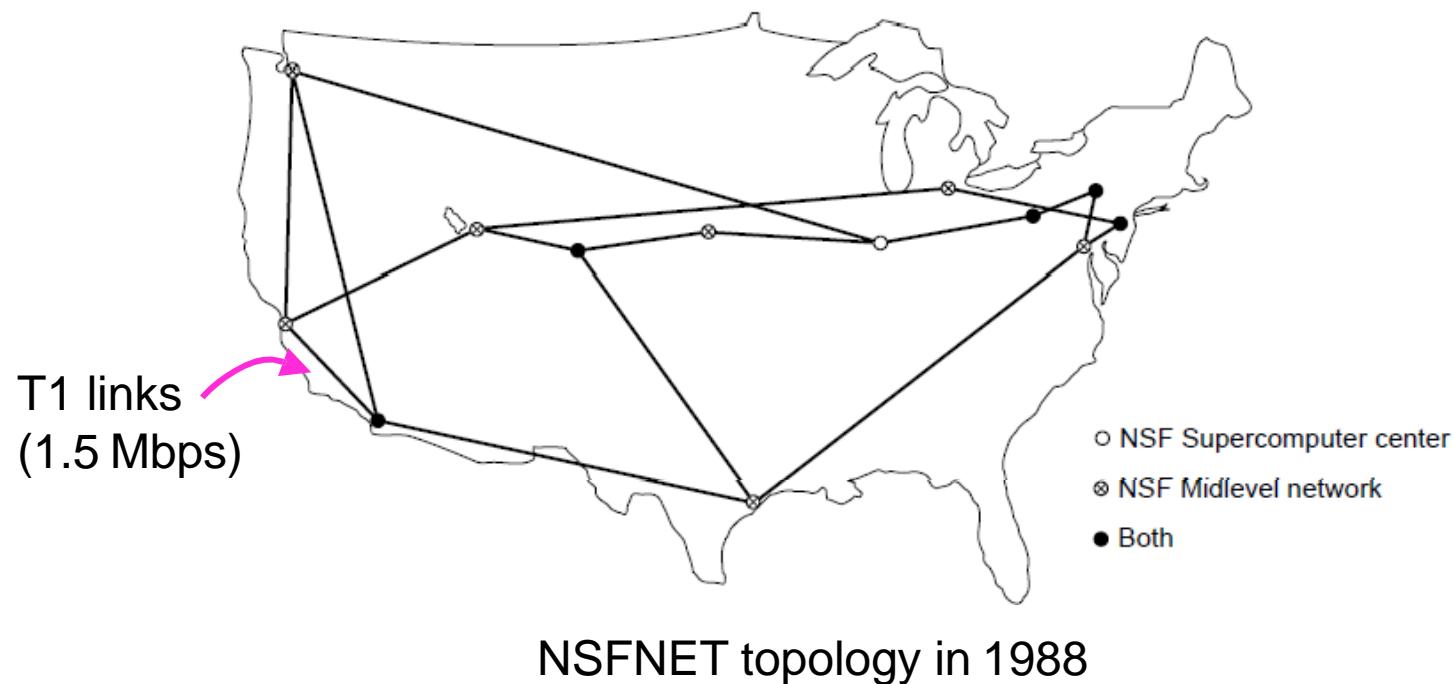
- Before the Internet was the ARPANET, a decentralized, packet-switched network based on Baran's ideas.
- DNS evolved (Domain name system)
 - A generalized distributed database system for storing variety of information related to naming.



ARPANET topology in Sept 1972.

Internet

- The early Internet used NSFNET (1985-1995) as its backbone; universities connected to get on the Internet



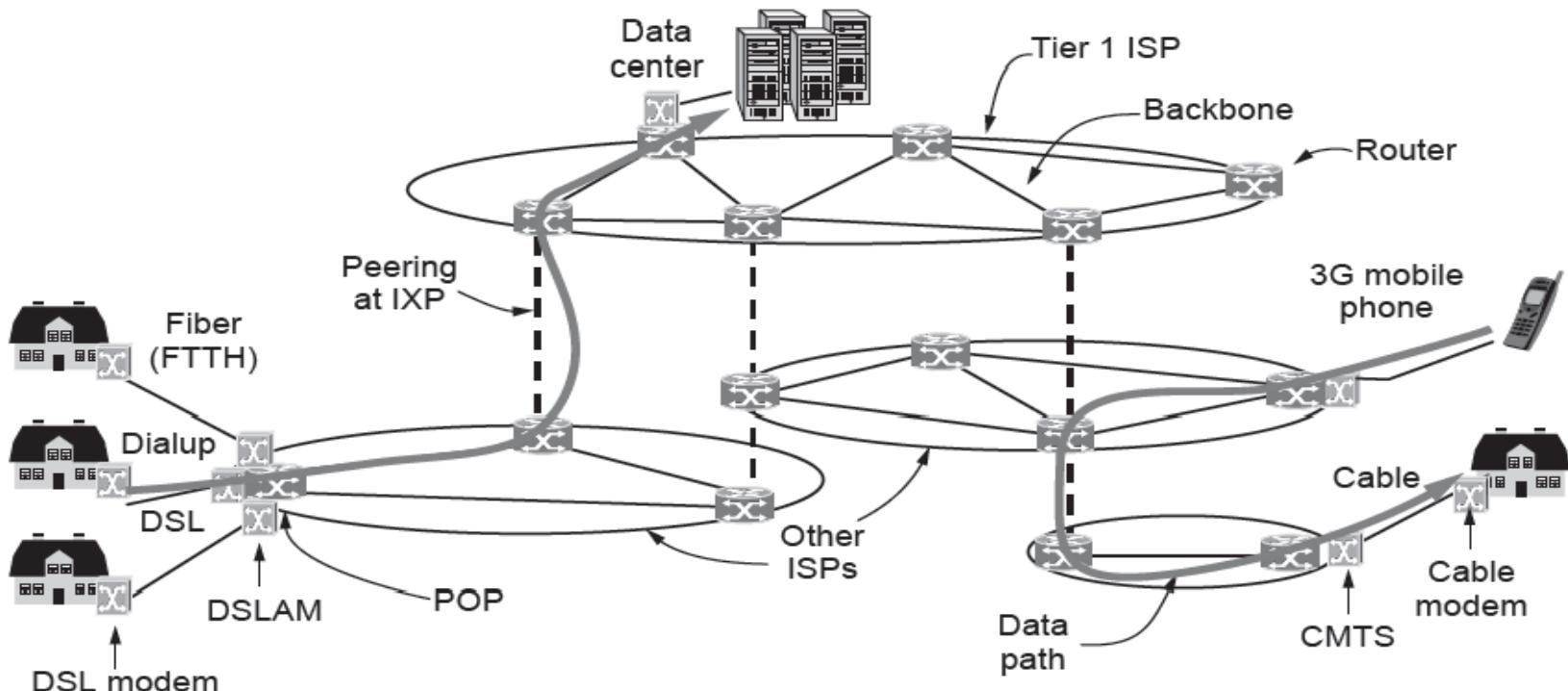
NSFNET: National Science Foundation Network

Internet

The modern Internet is more complex:

- ISP networks serve as the Internet backbone
- ISPs connect or peer to exchange traffic at IXPs
- Within each network routers switch packets
- Between networks, traffic exchange is set by business agreements
- Customers connect at the edge by many means
 - Cable, DSL, Fiber-to-the-Home, 3G/4G wireless, dialup
- Data centers concentrate many servers (“the cloud”)
- Most traffic is content from data centers (esp. video)
- The architecture continues to evolve

Internet (4)

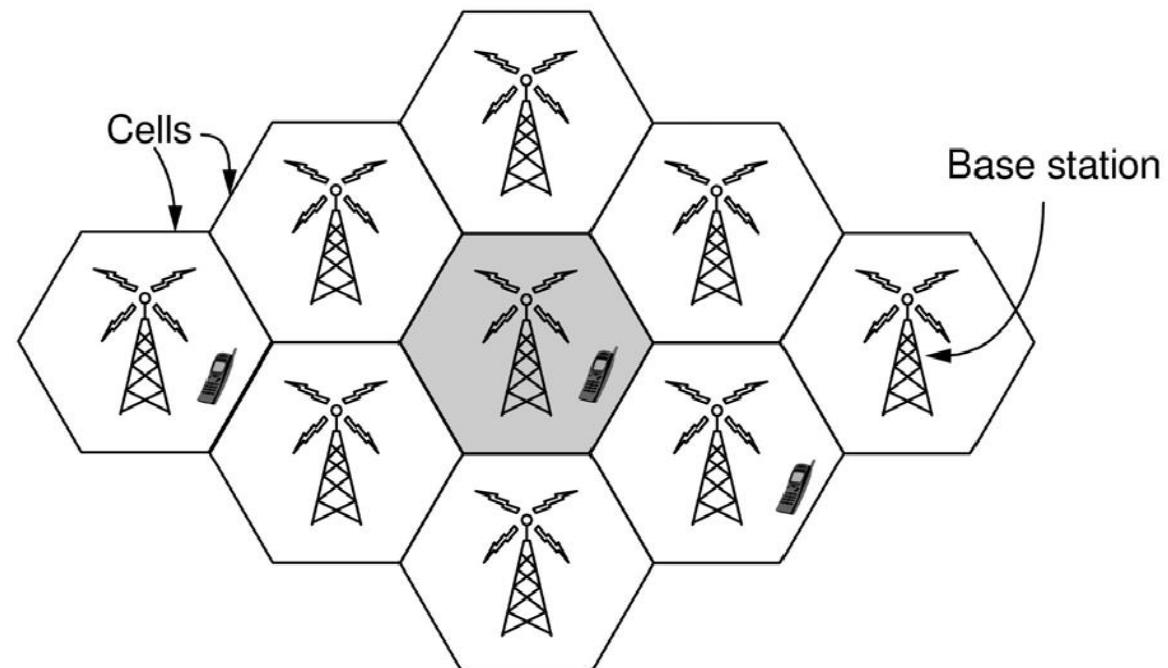


Architecture of the Internet

- DSLAM (Digital Subscriber Line Access Multiplexer)
- *Fiber to the home (FTTH)*
- A cable modem termination system (CMTS)
- Internet eXchange Point (IXP) : Internet Exchange Point (IXP) can earn money by charging each ISP that connects to it. The IXP charges each ISP based on the amount of traffic sent to or received from the IXP.

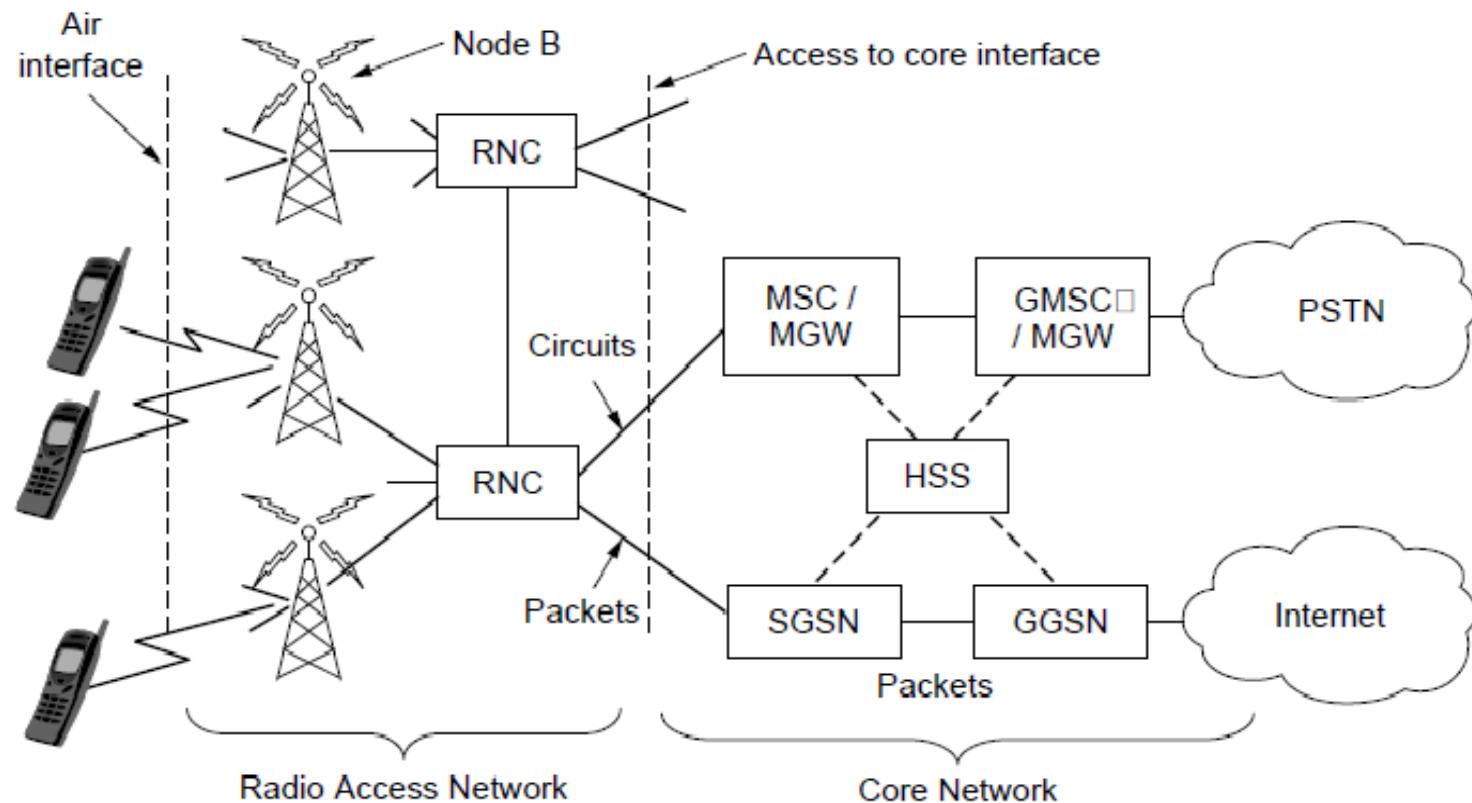
3G Mobile Phone Networks (1)

- 3G stands for third generation
- Evolved due to scarcity of spectrum
- 3G network is based on spatial cells; each cell provides wireless service to mobiles within it via a base station
- Coverage area is divided into cells.
- Within a cell, users are assigned channels which do not interfere with each other and do not cause much interference to adjacent cells.



3G Mobile Phone Networks (2)

Base stations connect to the core network to find other mobiles and send data to the phone network and Internet

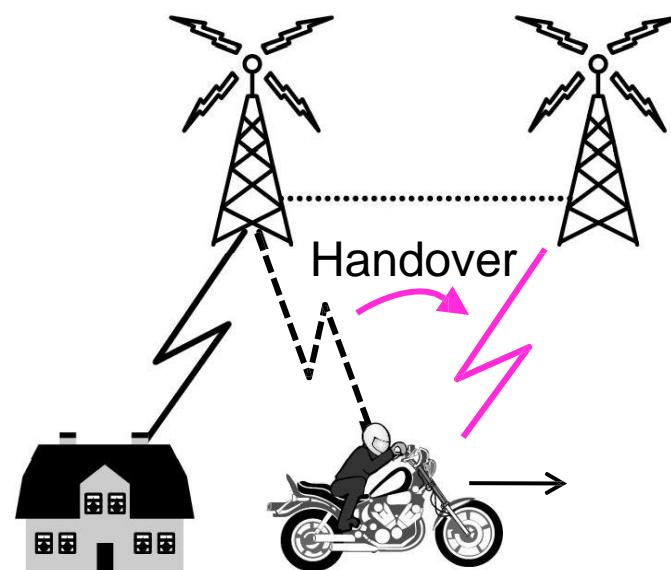


3G mobile networks

- Air interface
- It is based on UMTS (Universal Mobile Telecommunication System) or WCDMA (Wideband code division multiple access)
 - Can provide 14Mbps downlink and 6Mbps uplink
- Cellular base station together with the controller forms radio access network
- The controller node RNC (Radio Network Controller) controls the spectrum used.
- The rest of the mobile network carries traffic, which is called core network
 - MSC: mobile switching center
 - GMSC: Gateway mobile switching center
 - MGW: media gateway
 - PSTN: Public switched telephone network
- Data services
 - GPRS General Packet Radio Service
 - SGSN: Serving GPRS support node
 - GGSN: Gateway GPRS support node
- SIM card
 - Subscriber Identity module (SIM)
- 4G
 - LTE (Long term evolution)
 - WiMAX
- It can support both connection oriented and connectionless

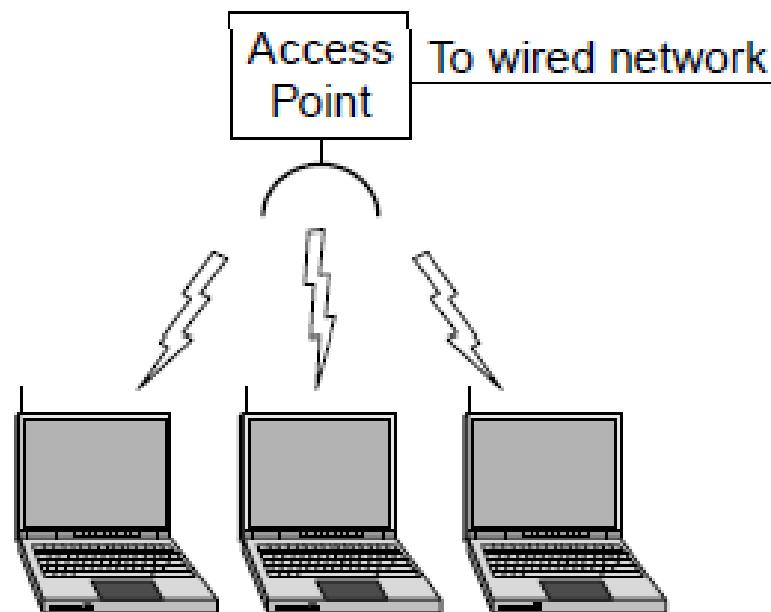
3G Mobile Phone Networks (3)

As mobiles move, base stations hand them off from one cell to the next, and the network tracks their location



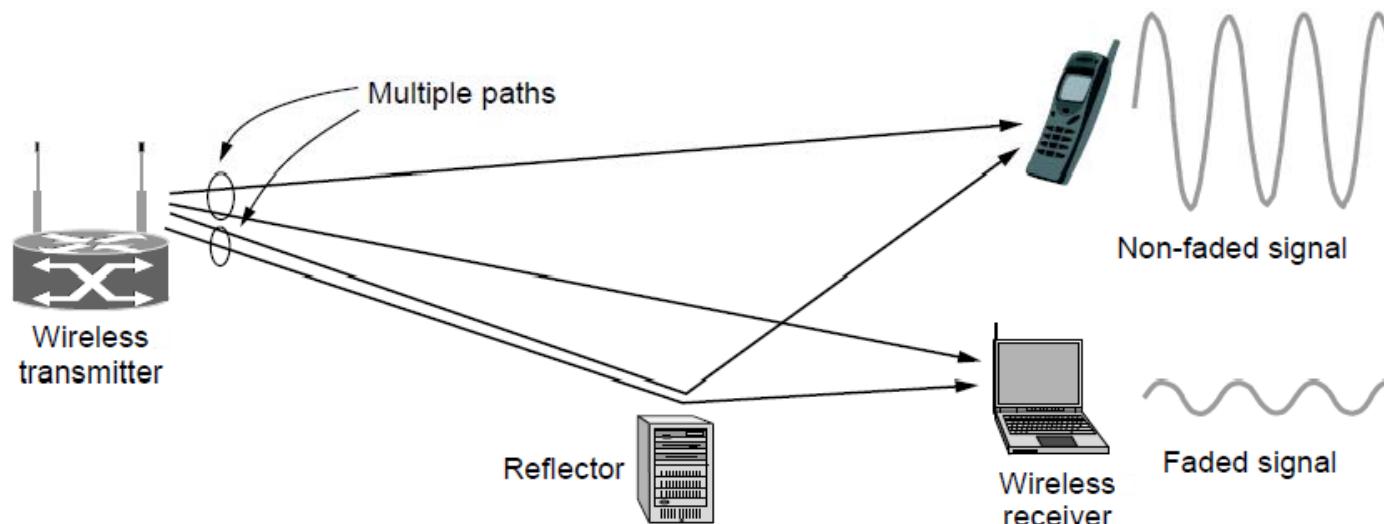
Wireless LANs (1)

In 802.11, clients communicate via an AP (Access Point) that is wired to the rest of the network.



Wireless LANs (2)

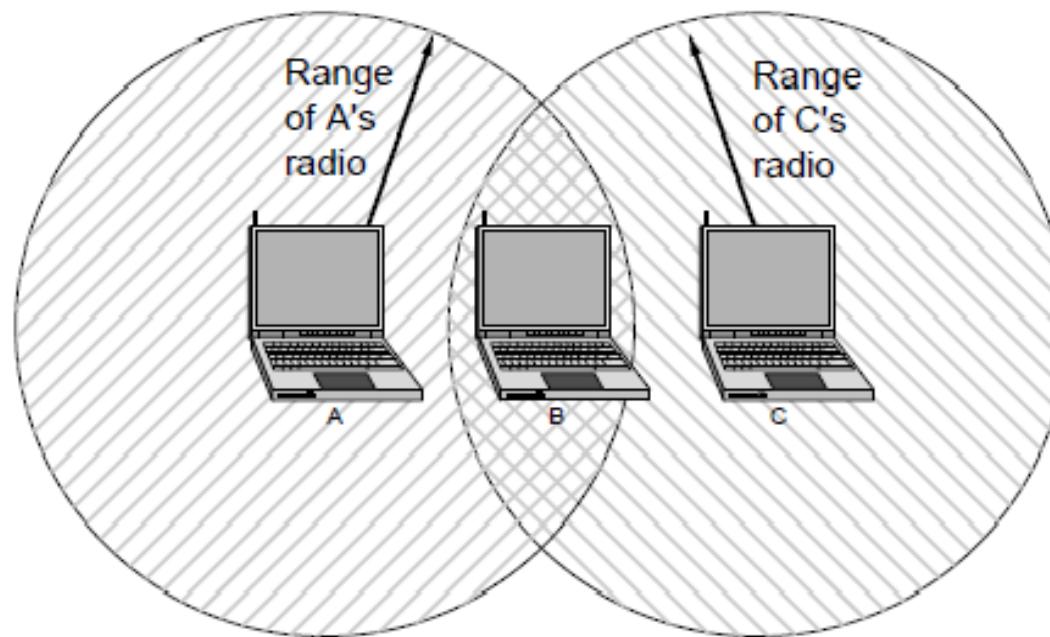
- Signals in the 2.4GHz ISM band vary in strength due to many effects, such as **multipath fading** due to reflections
 - requires complex transmission schemes, e.g., OFDM



Wireless LANs (3)

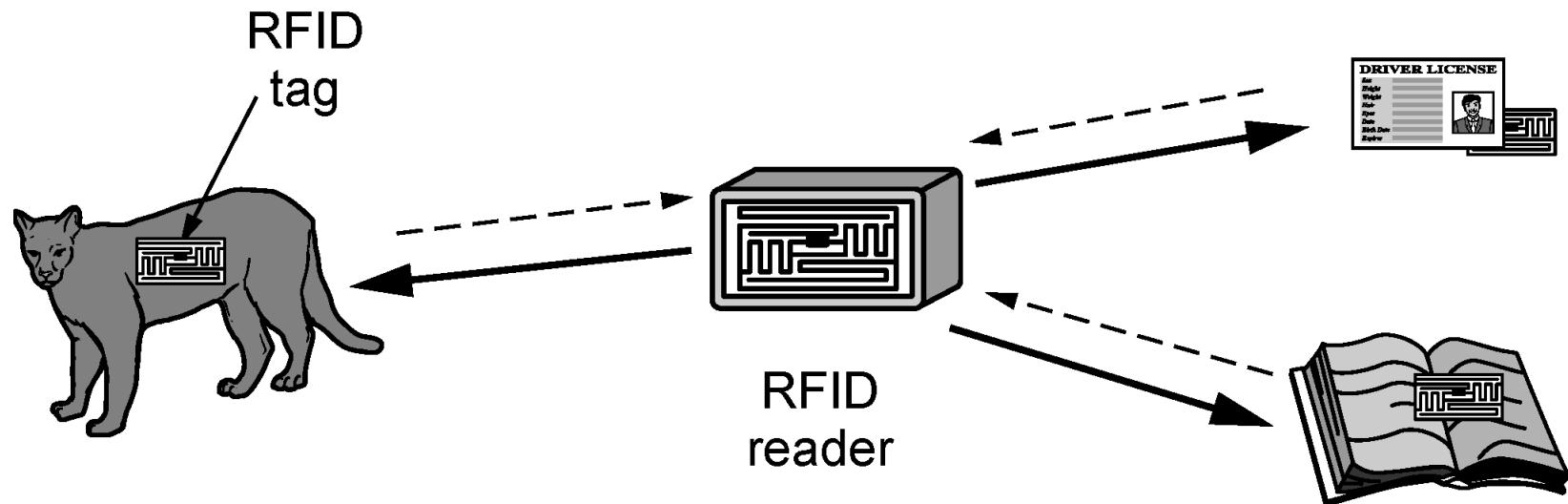
Radio broadcasts interfere with each other, and radio ranges may incompletely overlap

- CSMA (Carrier Sense Multiple Access) designs are used



RFID and Sensor Networks (1)

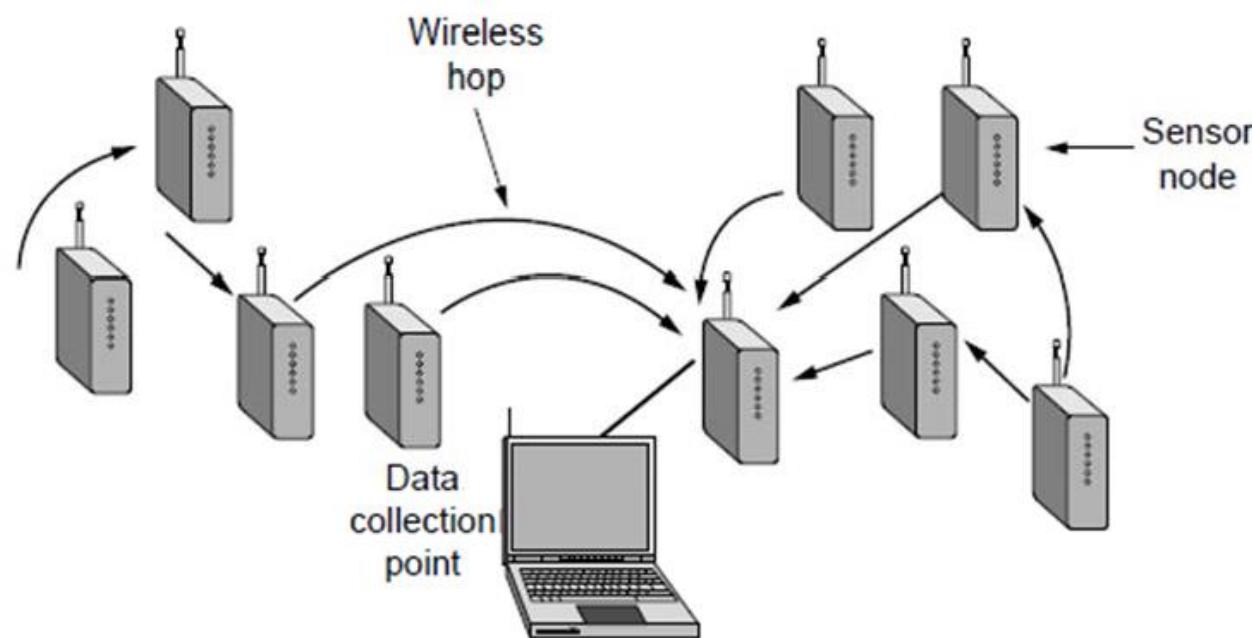
- RFID: Radio Frequency Identification
- Passive UHF RFID networks everyday objects:
 - Tags (stickers with not even a battery) are placed on objects
 - Readers send signals that the tags reflect to communicate
 - Tags communicate at distances of several meters
- HF RFID is short range, 1 meter
- LF RFID is used for animal tracking



RFID used to network everyday objects.

RFID and Sensor Networks (2)

- Sensor networks spread small devices over an area:
 - Devices send sensed data to collector via wireless hops
 - Sensor nodes are small computers which has the temperature, pressure, vibration and other sensors.
 - They have batteries. They can also harvest energy from Sun
 - They can communicate with each other. One of the node can become hop.
 - Organized into a **multihop network**



Network Standardization

Standards define what is needed for interoperability

Some of the many standards bodies:

| Body | Area | Examples |
|------|--------------------|--|
| ITU | Telecommunications | G.992, ADSL H.264, MPEG4 |
| IEEE | Communications | 802.3, Ethernet 802.11, WiFi |
| IETF | Internet | RFC 2616, HTTP/1.1 RFC 1034/1035, DNS |
| W3C | Web | HTML5 standard CSS standard |

ITU: International Telecommunications Union.

IRTF: Internet Research Task Force

IETF: Internet Engineering Task Force

iAB: Internet Architecture Board

Metric Units

- The main prefixes we use:

| Prefix | Exp. | prefix | exp. |
|--------|--------|---------------|-----------|
| K(ilo) | 10^3 | m(illi) | 10^{-3} |
| M(ega) | 10^6 | μ (micro) | 10^{-6} |
| G(iga) | 10^9 | n(ano) | 10^{-9} |

- Use powers of 10 for rates, powers of 2 for storage
 - E.g., 1 Mbps = 1,000,000 bps, 1 KB = 1024 bytes
- “B” is for bytes, “b” is for bits

| Exp. | Explicit | Prefix | Exp. | Explicit | Prefix |
|------------|--------------------|--------|-----------|-----------------------------------|--------|
| 10^{-3} | 0.001 | milli | 10^3 | 1,000 | Kilo |
| 10^{-6} | 0.000001 | micro | 10^6 | 1,000,000 | Mega |
| 10^{-9} | 0.000000001 | nano | 10^9 | 1,000,000,000 | Giga |
| 10^{-12} | 0.000000000001 | pico | 10^{12} | 1,000,000,000,000 | Tera |
| 10^{-15} | 0.000000000000001 | femto | 10^{15} | 1,000,000,000,000,000 | Peta |
| 10^{-18} | 0.0000000000000001 | atto | 10^{18} | 1,000,000,000,000,000,000 | Exa |
| 10^{-21} | 0.0000000000000001 | zepto | 10^{21} | 1,000,000,000,000,000,000,000 | Zetta |
| 10^{-24} | 0.0000000000000001 | yocto | 10^{24} | 1,000,000,000,000,000,000,000,000 | Yotta |

Summary

- Computer networks have many uses
 - Companies and individuals
- Networks can be divided into LANs, MANs, WANs and Internetworks.
- Wireless networks such as 3G and 802.11 LANs are becoming popular.
- Network software is built around protocols, the rules the processes communicate.
- Protocols are based on OSI or TCP/IP model
- TCP/IP
 - Link, network, transport and application layers
- Design issues
 - Reliability, delay, resource allocation, growth, security, and more
- Networks provide services
 - Connectionless best effort packet delivery
 - Connection oriented guaranteed delivery
- Well-known networks
 - Internet, the 3G mobile phone network, 802.11 LANs
- Internet is a collection of thousands of networks which employ TCP/IP stack
- New kinds of networks
 - Embedded sensor networks, networks based on sensor technology
- Standardization efforts
 - ITU-T, ISO, IEEE and IETF manage different parts of the standardization process.

The Physical Layer: an overview

Chapter 2

- Foundation on which other layers build
 - Properties of wires, fiber, wireless limit what the network can do
- We will cover theoretical basis of data transmission.
- Nature puts limits on the data sent over a channel.
- Properties of channels determine the performance
 - Throughput, latency, error rate
- We will cover three kinds of media
 - Guided (copper wire and fiber optics)
 - Wireless (terrestrial radio)
 - Satellite.
- Key problem is to send (digital) bits using only (analog) signals
 - This is called modulation
- We will cover communication system used by computer networks.

Application
Transport
Network
Link
Physical

The Physical Layer

- Theoretical Basis for Data Communications
- Guided Transmission Media
- Wireless Transmission
- Communication Satellites
- Digital Modulation and Multiplexing
- Public Switched Telephone Network
- Mobile Telephone System
- Cable Television

Theoretical Basis for Data Communication

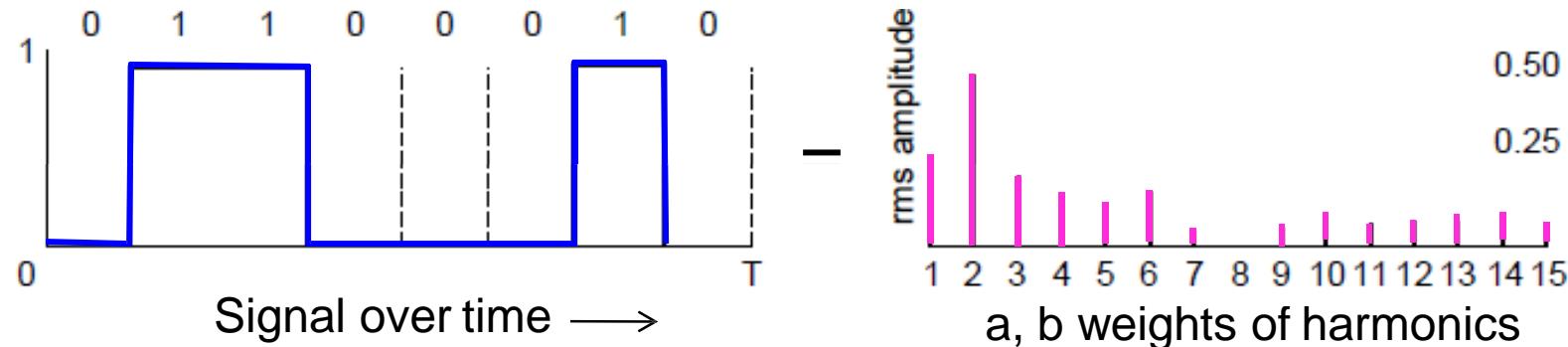
Communication rates have fundamental limits

- Fourier analysis
- Bandwidth-limited signals
- Maximum data rate of a channel

Fourier Analysis

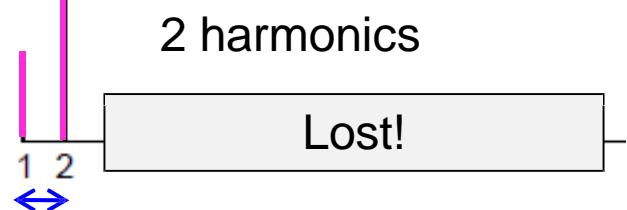
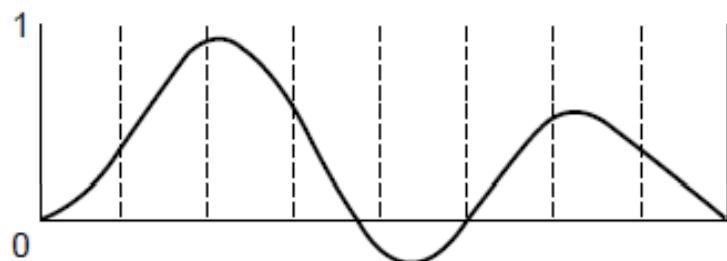
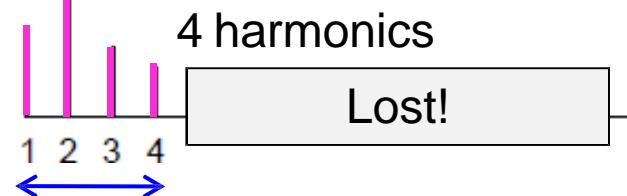
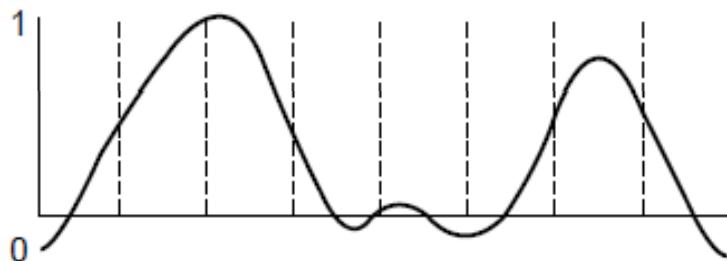
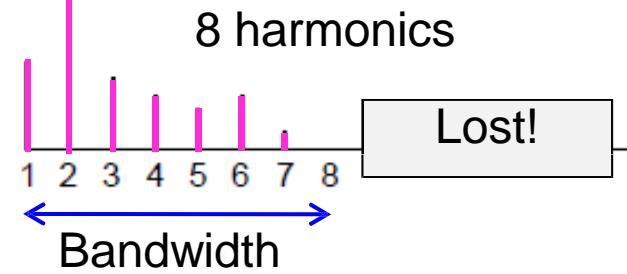
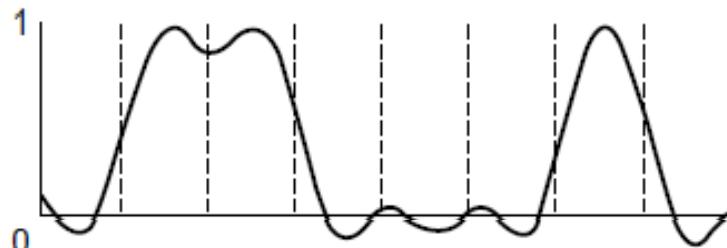
A time-varying signal can be equivalently represented as a series of frequency components (harmonics):

$$g(t) = \frac{1}{2}c + \sum_{n=1}^{\infty} a_n \sin(2\pi nft) + \sum_{n=1}^{\infty} b_n \cos(2\pi nft)$$



Bandwidth-Limited Signals

Having less bandwidth (harmonics) degrades the signal



Maximum Data Rate of a Channel

- 1924, Henry Nyquist
 - Perfect channel has a finite transmission capacity
 - He derived an equation expressing the maximum data rate of the channel
 - Nyquist's theorem relates the data rate to the bandwidth (B) and number of signal levels (V)

$$\text{Max. data rate} = 2B \log_2 V \text{ bits/sec}$$

- 1948, Shannon
 - Derived an equation expressing the maximum rate of the channel subject random noise.
 - Most important paper in information theory
- Shannon's theorem relates the data rate to the bandwidth (B) and signal strength (S) relative to the noise (N):

$$\text{Max. data rate} = B \log_2(1 + S/N) \text{ bits/sec}$$

Guided Transmission (Wires & Fiber)

Media have different properties, hence performance

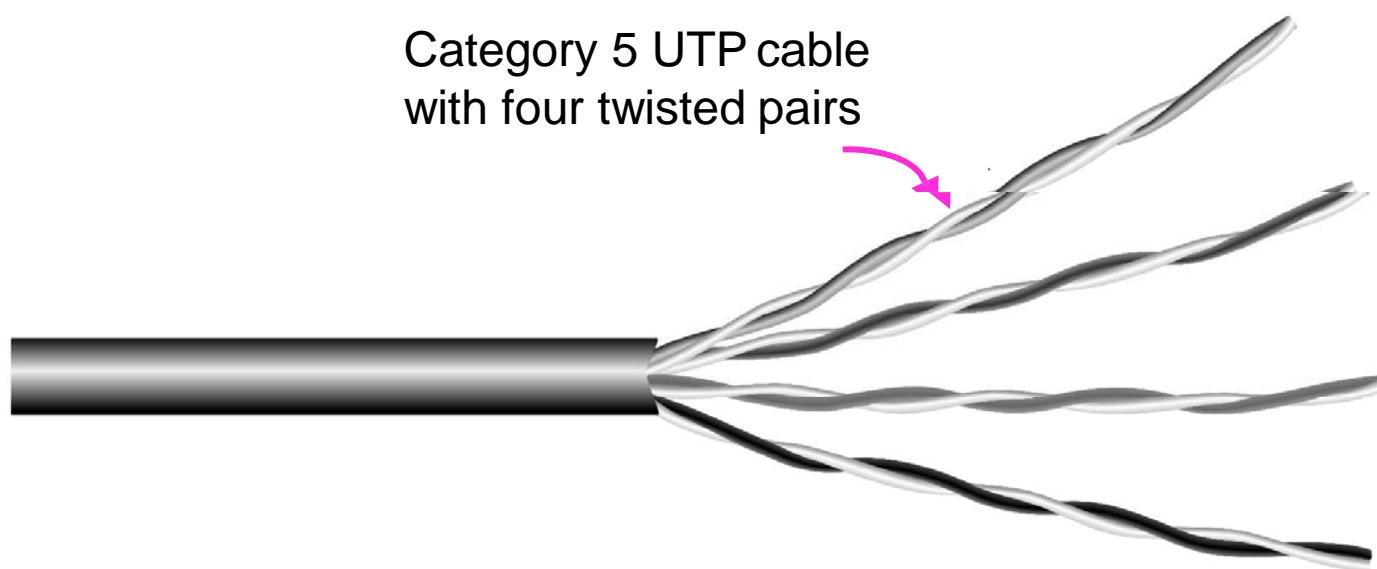
- Reality check
 - Storage media
- Wires:
 - Twisted pairs
 - Coaxial cable
 - Power lines
- Fiber cables

Reality Check: Storage media

- Send data on tape / disk / DVD for a high bandwidth link
 - Mail one box with 1000 800GB tapes (6400 Tera bits)
 - Takes one day to send (86,400 secs)
 - Data rate/bandwidth is 70 Gbps.
- Data rate is faster than long-distance networks! It is also cheap. But, the message delay is very poor.

Wires – Twisted Pair

- Very common; used in LANs, telephone lines
 - Twists reduce radiated signal (interference)
- Common application is telephone system and ADSL (Asymmetric Digital Subscriber Line)
- Can be used to transmit both analog and digital signals
- Bandwidth depends on the thickness of wire and distance travelled
- But, bandwidth of several megabits/sec can be achieved for few kilometers.
- Low cost and reasonable performance



Link Terminology

Full-duplex link

- Used for transmission in both directions at once
- e.g., use different twisted pairs for each direction

Half-duplex link

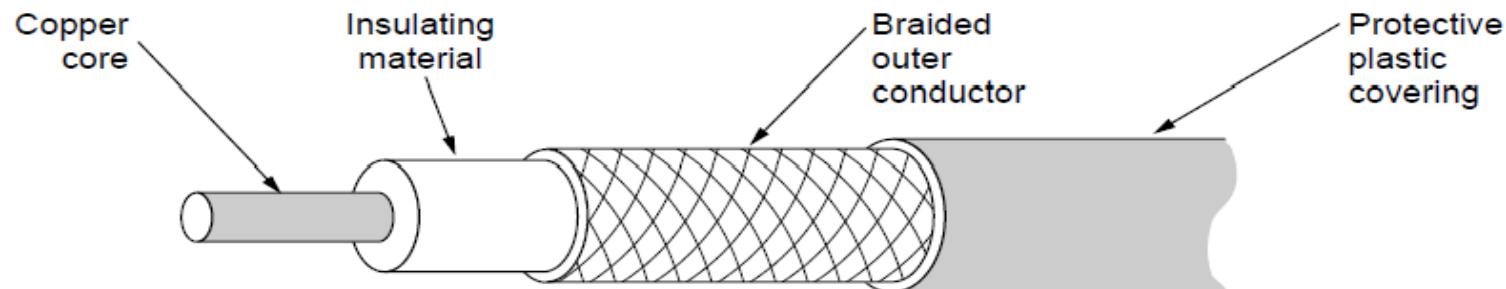
- Both directions, but not at the same time
- e.g., senders take turns on a wireless channel

Simplex link

- Only one fixed direction at all times; not common

Wires – Coaxial Cable (“Co-ax”)

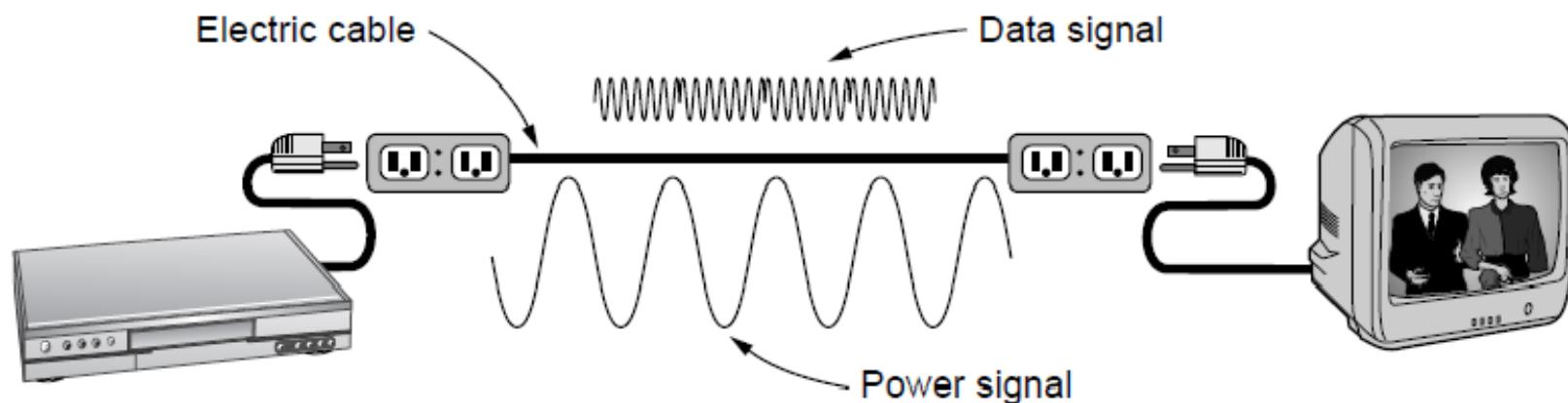
- A coaxial cable consists of still copper wire as a core, surrounded by an insulating material.
- Greater bandwidth than twisted pair
- Better shielding and more bandwidth for longer distances and higher rates than twisted pair.
- Bandwidth depends on cable quality and length
- Bandwidth is up to 1GHz
- Also common in long distance lines, cable lines, television networks.



Wires – Power Lines

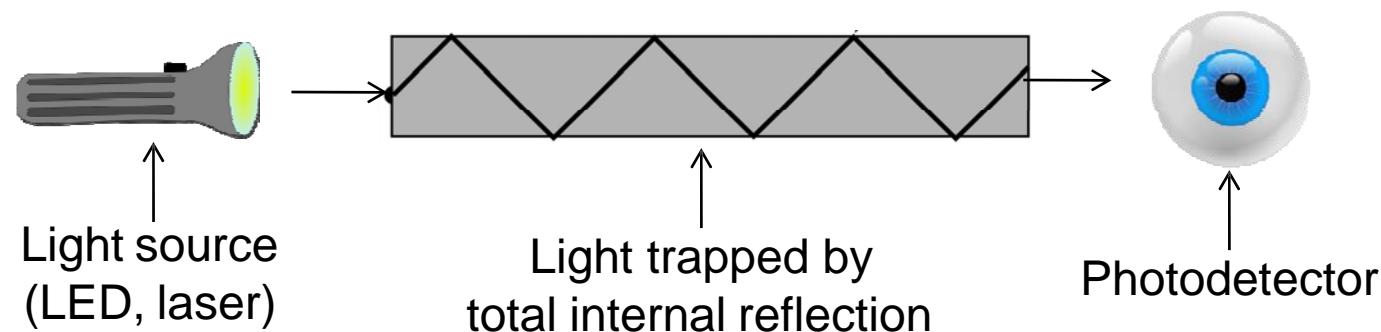
Household electrical wiring is another example of wires

- Convenient to use, but horrible for sending data
- 50-60 Hz
- Interference will be more, wires act like an antenna
- Efforts are going on



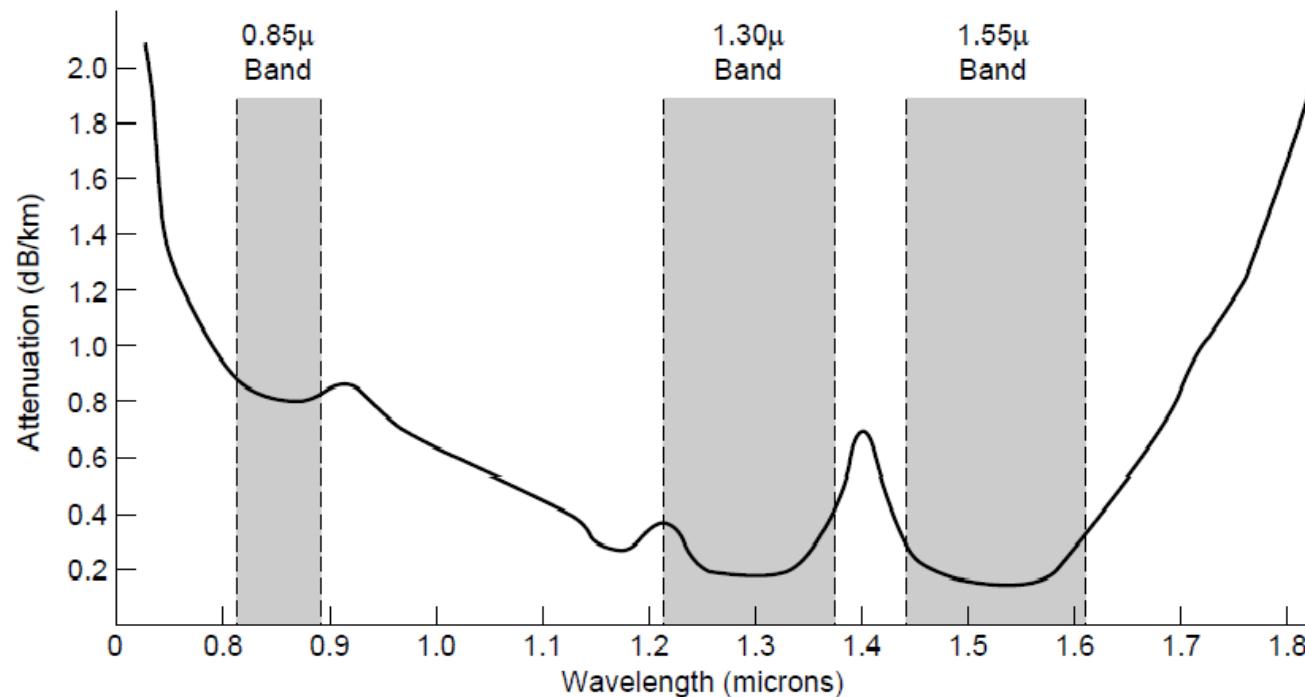
Fiber Cables (1)

- CPU speed has improved from 4.77 MHz (in the year 1981) to 3 GHz. Increased by a factor of 2500.
- Similarly, communication speed went from 45Mbps (telephone) to 100Gbps. Common for high rates and long distances and error rate went from 10^{-5} to zero.
- Achievable bandwidth with fiber is 50,000 Gbps and difficult to reach. So, many channels are employed over a single fiber.
- Light source, transmission medium and detector
 - Light pulses are transmitted (presence=1 absence=0)
 - Transmission medium is ultra-thin fiber of glass.
- Long distance ISP links, Fiber-to-the-Home
- Light carried in very long, thin strand of glass



Fiber Cables (2)

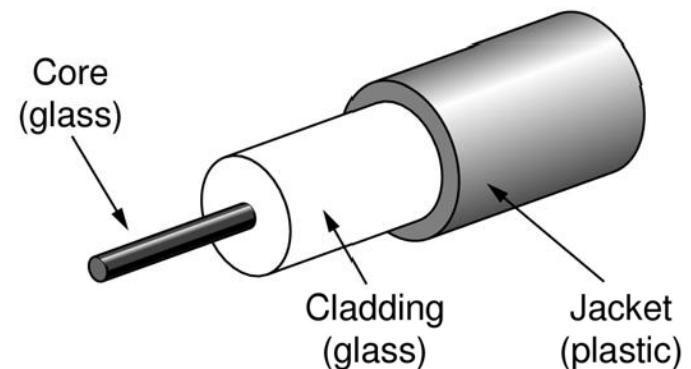
Fiber has enormous bandwidth (THz) and tiny signal loss – hence high rates over long distances



Fiber Cables (3)

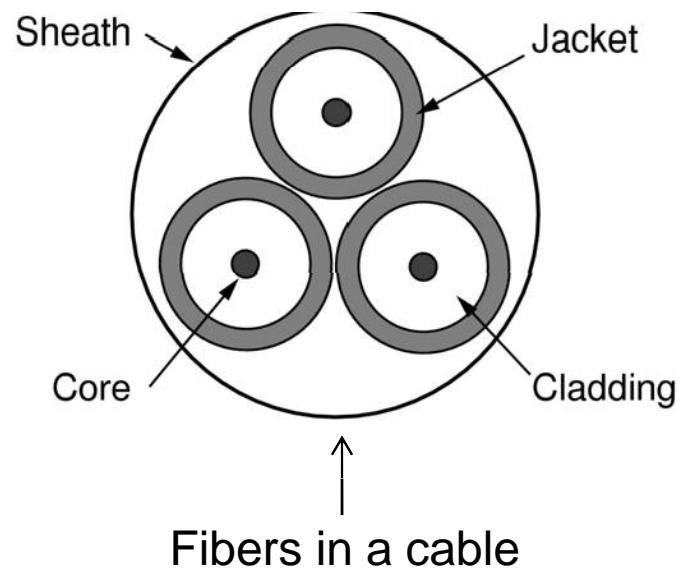
Single-mode

- Core so narrow (10um) light can't even bounce around
- Used with lasers for long distances, e.g., 100km



Multi-mode

- Other main type of fiber
- Light can bounce (50um core)
- Used with LEDs for cheaper, shorter distance links



Fiber Cables (4)

Comparison of the properties of wires and fiber:

| Property | Wires | Fiber |
|-------------|-------------------|-------------------|
| Distance | Short (100s of m) | Long (tens of km) |
| Bandwidth | Moderate | Very High |
| Cost | Inexpensive | Less cheap |
| Convenience | Easy to use | Less easy |
| Security | Easy to tap | Hard to tap |

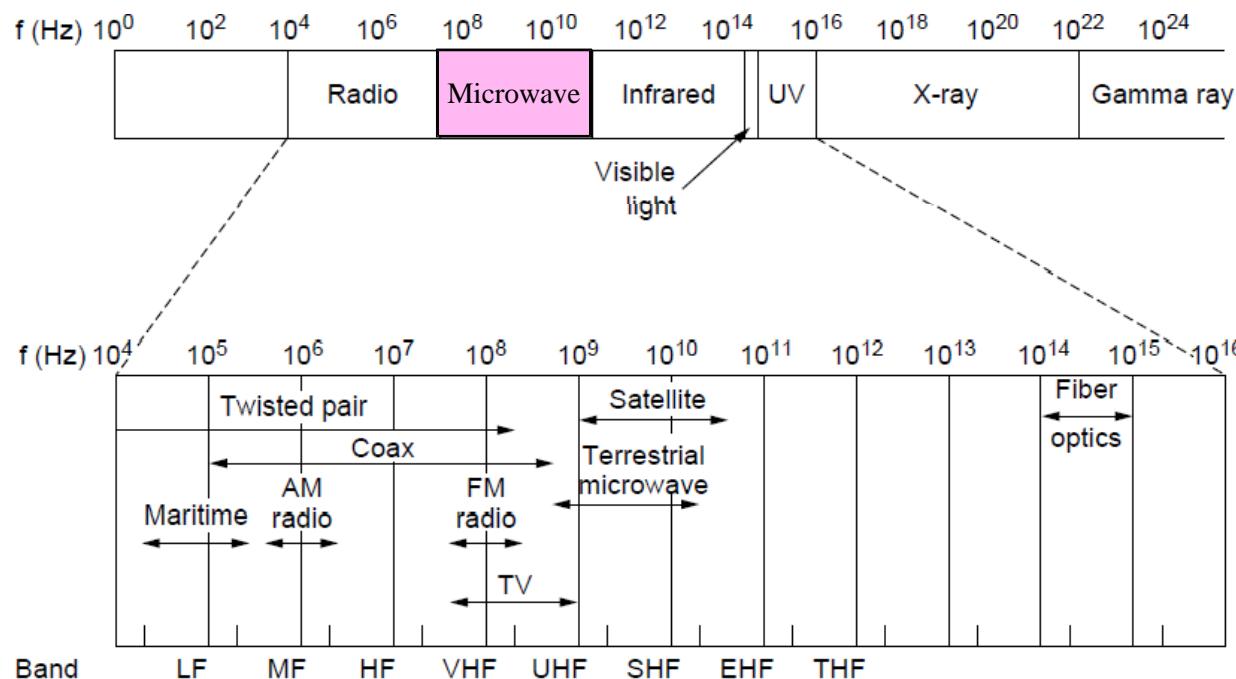
Wireless Transmission

- Electromagnetic Spectrum »
- Radio Transmission »
- Microwave Transmission »
- Light Transmission »
- Wireless vs. Wires/Fiber »

Electromagnetic Spectrum (1)

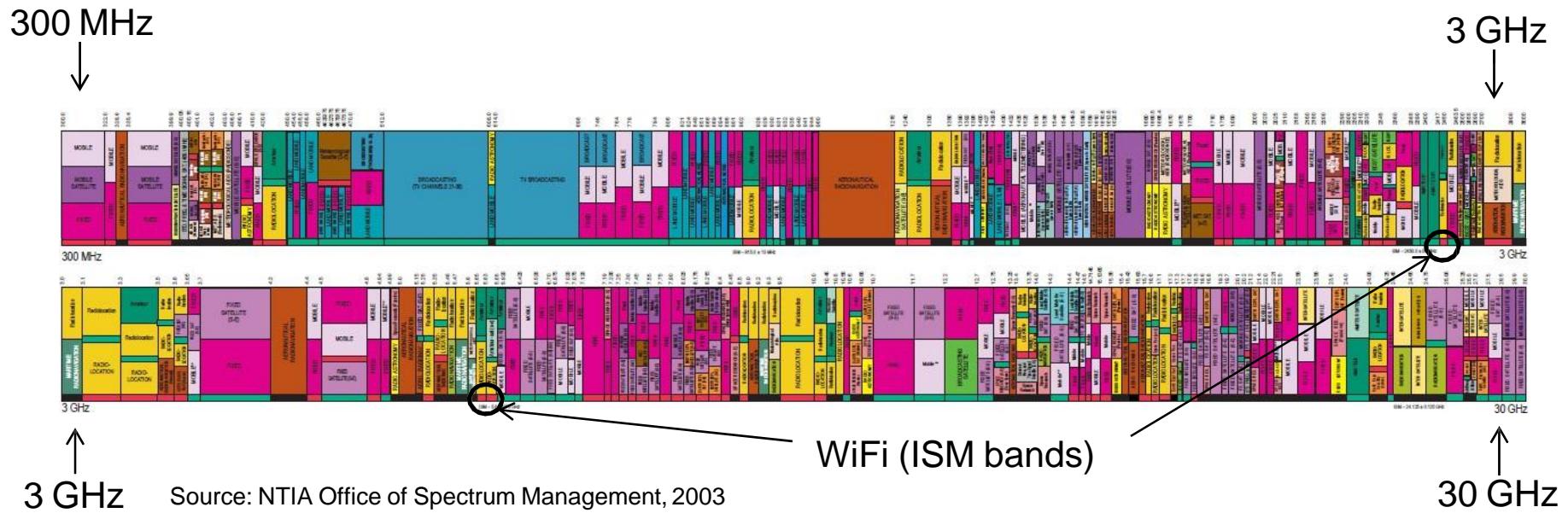
Different bands have different uses:

- Radio: wide-area broadcast; Infrared/Light: line-of-sight
- Microwave: LANs and 3G/4G; ← Networking focus



Electromagnetic Spectrum (2)

To manage interference, spectrum is carefully divided, and its use regulated and licensed, e.g., sold at auction.

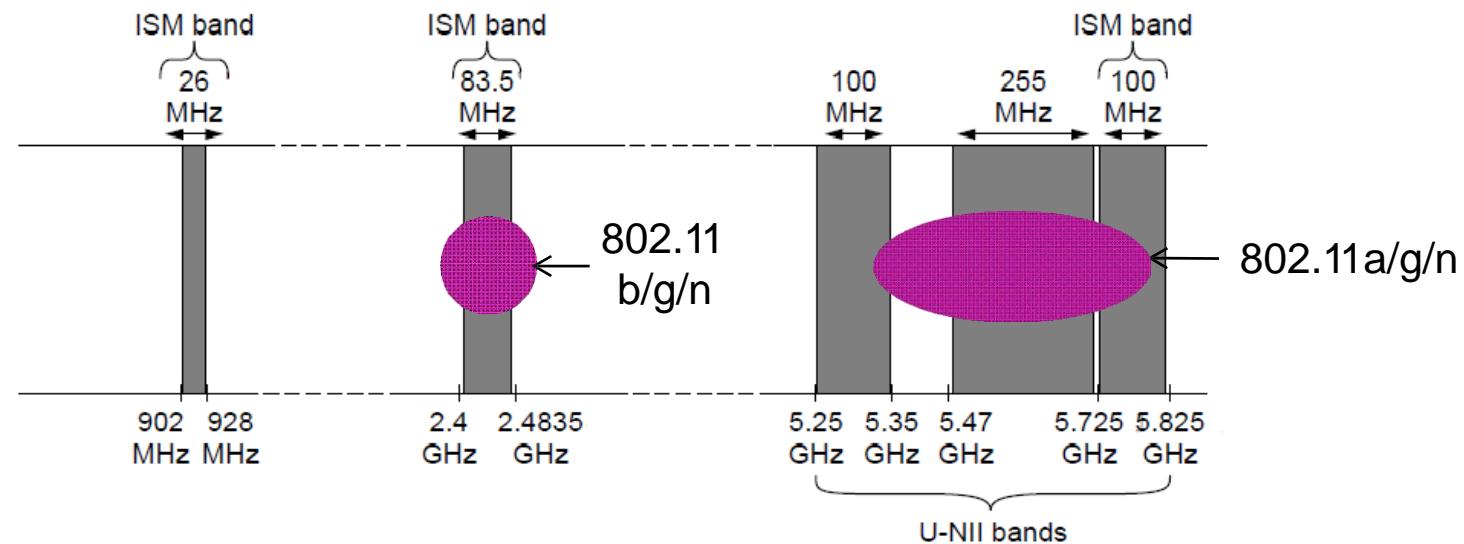


Part of the US frequency allocations

Electromagnetic Spectrum (3)

Fortunately, there are also unlicensed (“ISM”) bands:

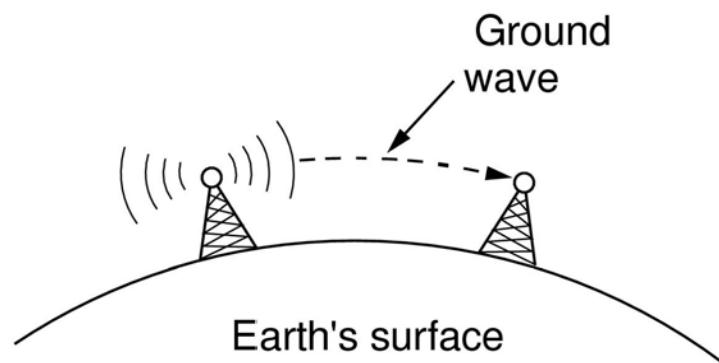
- Free for use at low power; devices manage interference
- Widely used for networking; WiFi, Bluetooth, Zigbee, etc.



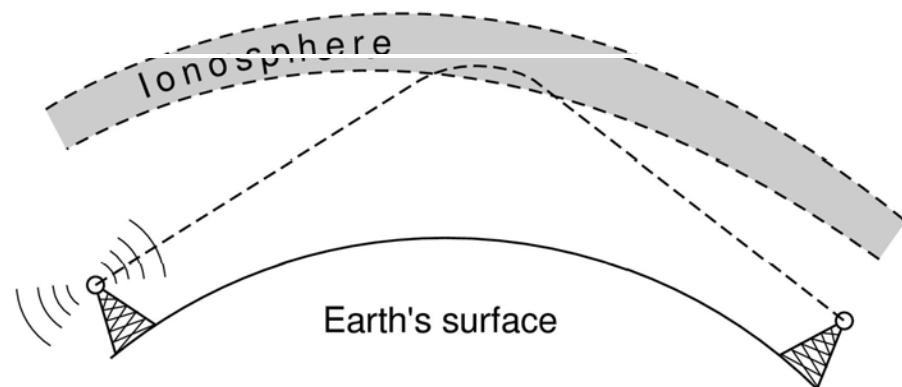
Industrial, scientific and medical networks (ISM networks)

Radio Transmission

- Radio signals penetrate buildings well and propagate for long distances with path loss (attenuation)
- At high frequencies, radio waves travel in a straight line.
- At all frequencies radio waves pass through obstacles, but power falls sharply



In the VLF, LF, and MF bands, radio waves follow the curvature of the earth

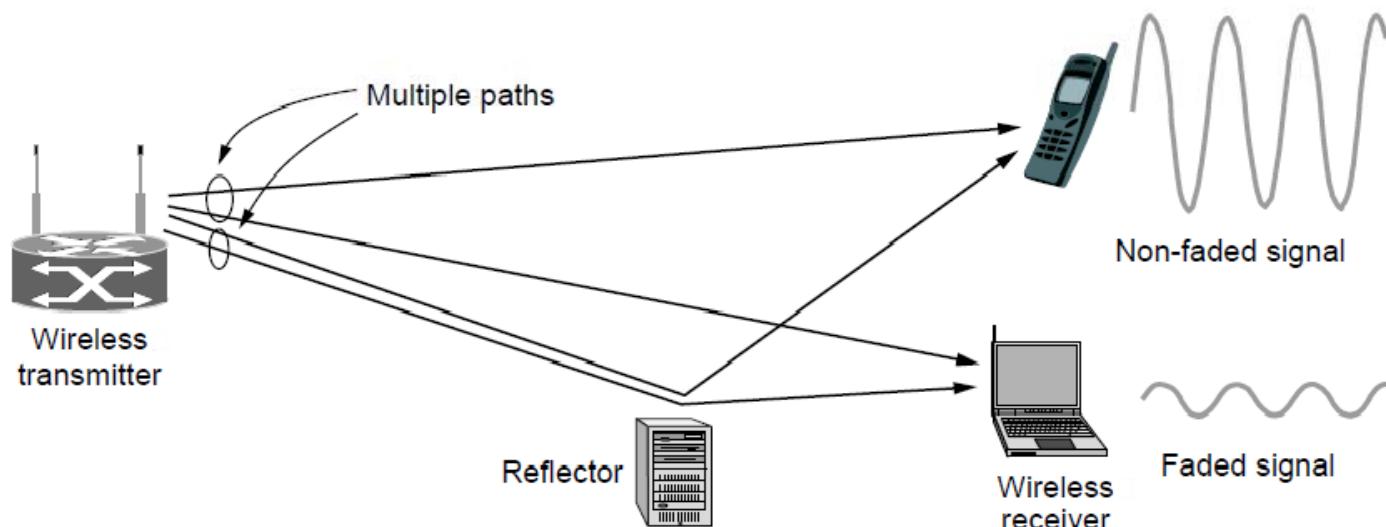


In the HF band, radio waves bounce off the ionosphere.

Microwave Transmission

Microwaves have much bandwidth and are widely used indoors (WiFi) and outdoors (3G, satellites)

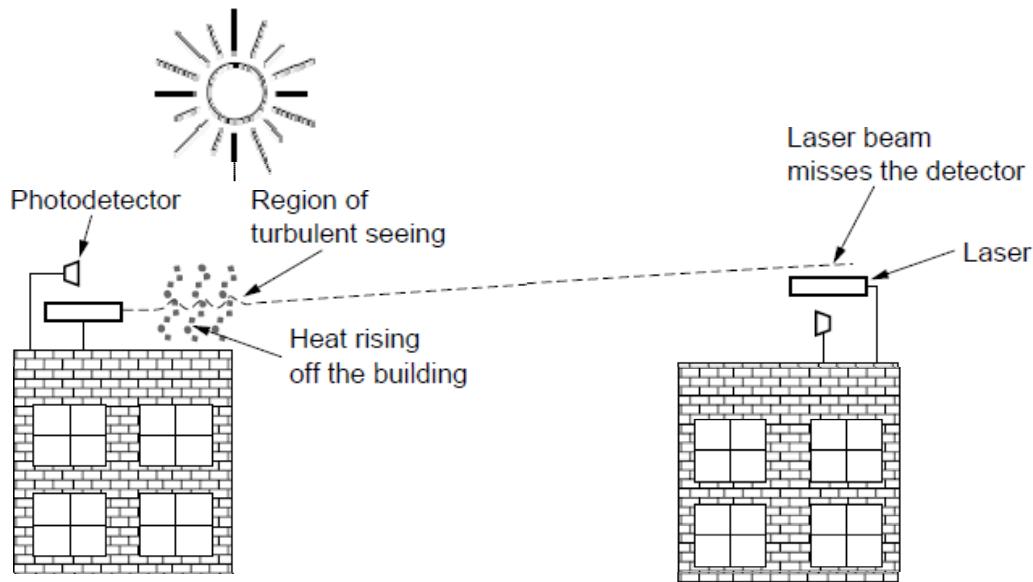
- Signal is attenuated/reflected by everyday objects
- Strength varies with mobility due multipath fading, etc.



Light Transmission

Line-of-sight light (no fiber) can be used for links

- Light is highly directional, has much bandwidth
- Use of LEDs/cameras and lasers/photodetectors



Wireless vs. Wires/Fiber

Wireless:

- + Easy and inexpensive to deploy
- + Naturally supports mobility
- + Naturally supports broadcast
- Transmissions interfere and must be managed
- Signal strengths hence data rates vary greatly

Wires/Fiber:

- + Easy to engineer a fixed data rate over point-to-point links
- Can be expensive to deploy, esp. over distances
- Doesn't readily support mobility or broadcast

Communication Satellites

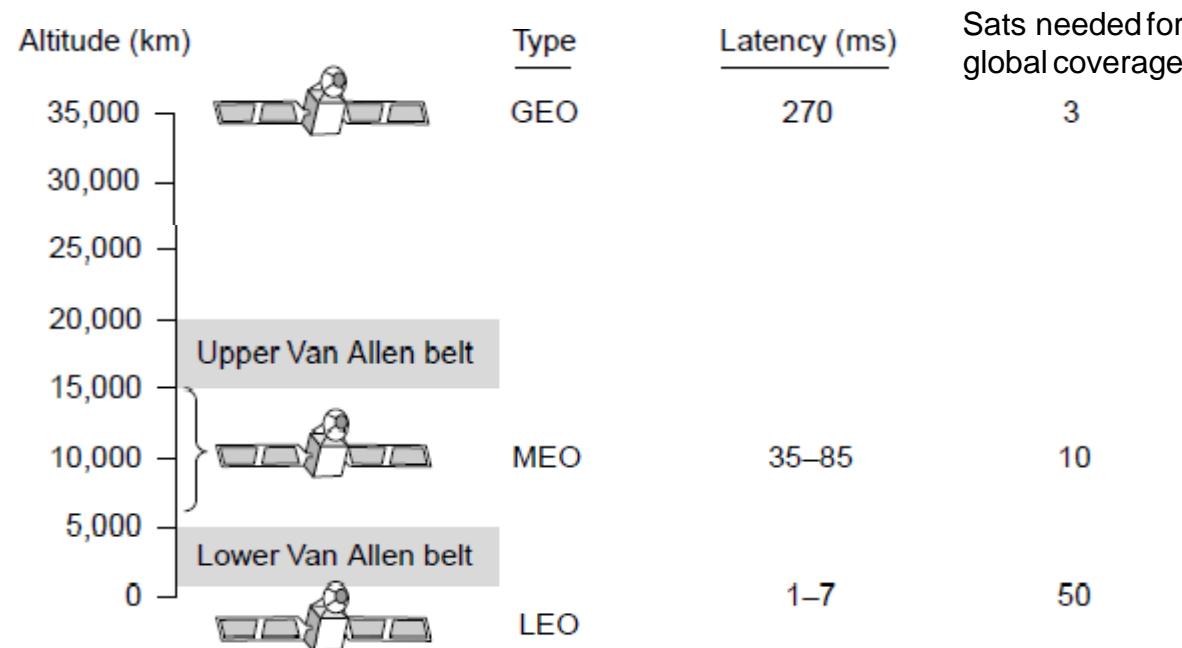
Satellites are effective for broadcast distribution and anywhere/anytime communications

- Kinds of Satellites »
- Geostationary (GEO) Satellites »
- Low-Earth Orbit (LEO) Satellites »
- Satellites vs. Fiber »

Kinds of Satellites

Satellites and their properties vary by altitude:

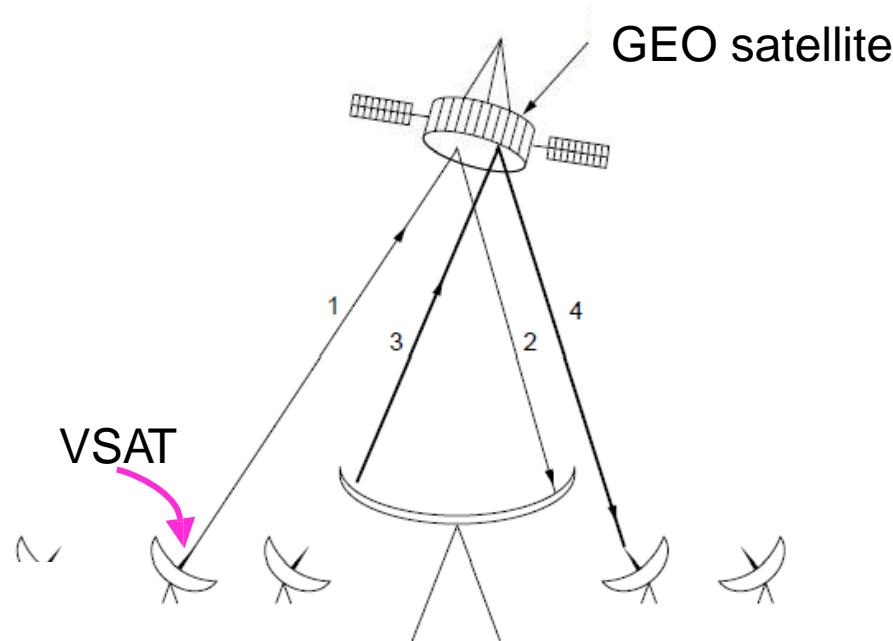
- Geostationary (GEO), Medium-Earth Orbit (MEO), and Low-Earth Orbit (LEO)



Geostationary Satellites

GEO satellites orbit 35,000 km above a fixed location

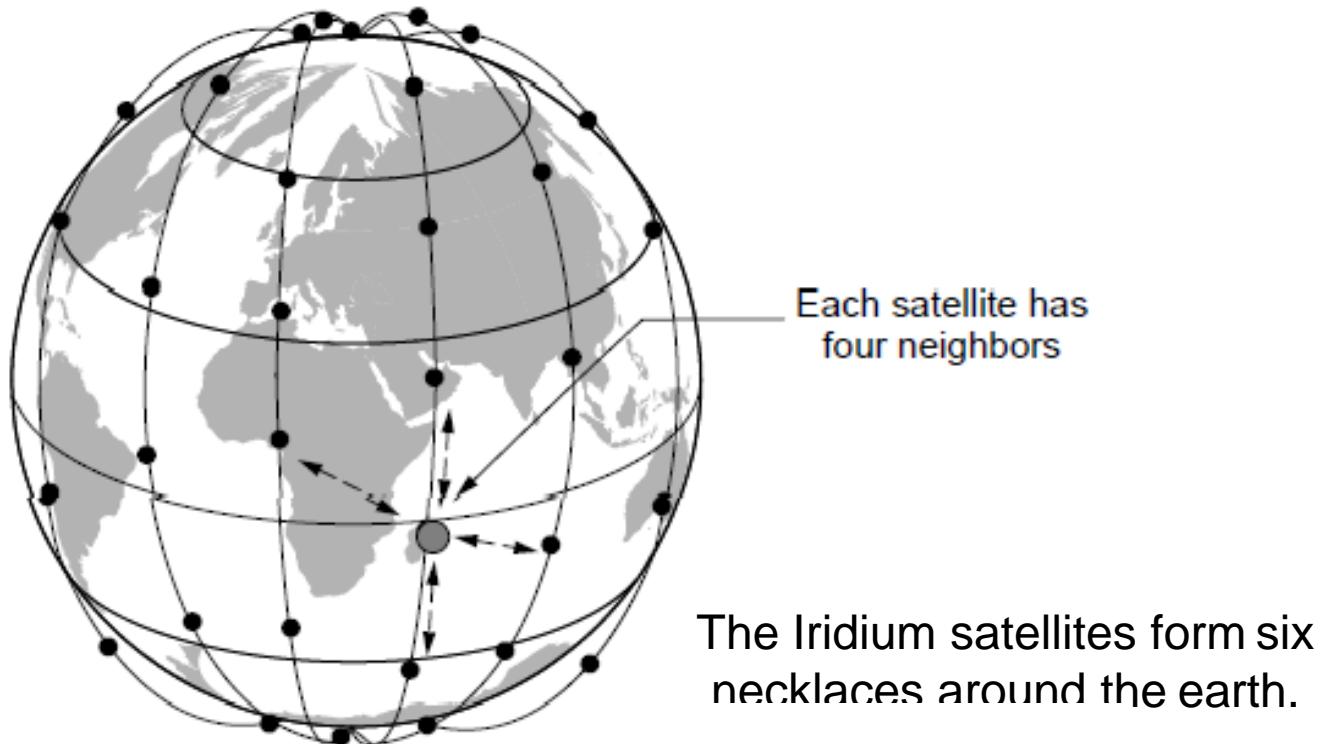
- VSAT (computers) can communicate with the help of a hub
- Different bands (L, S, C, Ku, Ka) in the GHz are in use but may be crowded or susceptible to rain.



A very small aperture terminal (VSAT) is a small-sized earth station used **in the transmit/receive of data, voice and video signals over a satellite communication network**, excluding broadcast television

Low-Earth Orbit Satellites

Systems such as Iridium use many low-latency satellites for coverage and route communications via them



Satellite vs. Fiber

Satellite:

- + Can rapidly set up anywhere/anytime communications (after satellites have been launched)
- + Can broadcast to large regions
- Limited bandwidth and interference to manage

Fiber:

- + Enormous bandwidth over long distances
- Installation can be more expensive/difficult

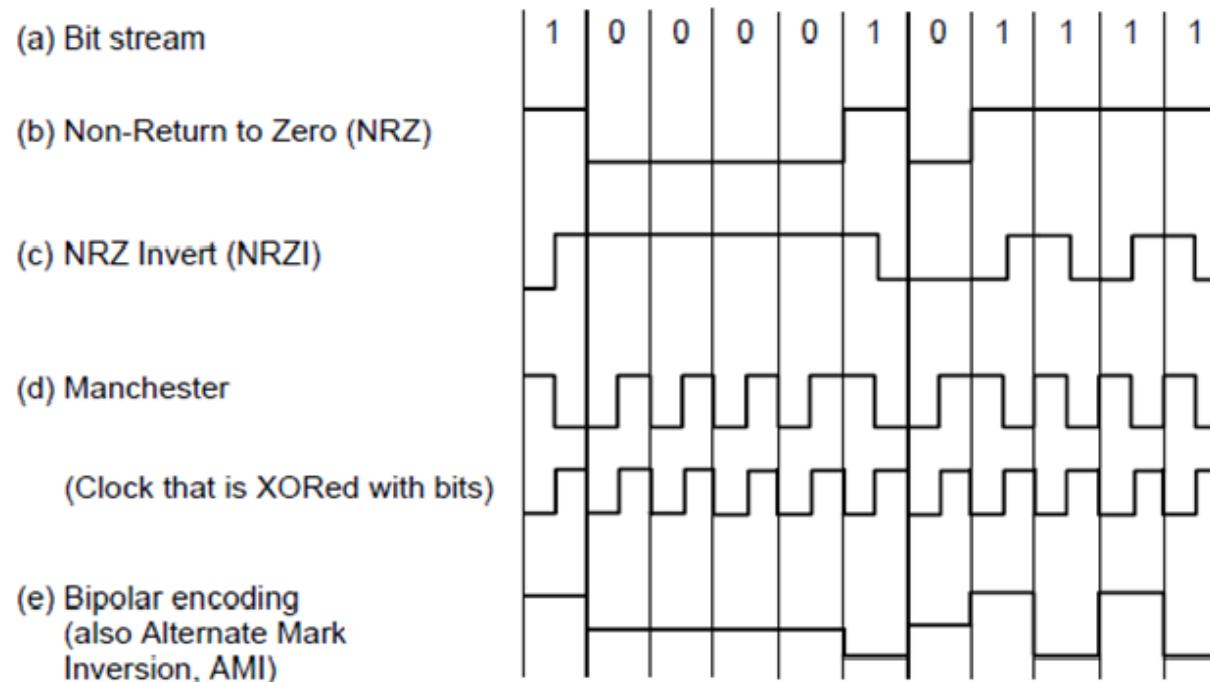
Digital Modulation and Multiplexing

- Process of converting between bits and signals is called modulation.
- multiplexing schemes share a channel among users.
 - Baseband Transmission »
 - Passband Transmission »
- Frequency Division Multiplexing »
- Time Division Multiplexing »
- Code Division Multiple Access »

Baseband Transmission

Line codes send symbols that represent one or more bits

- NRZ (**non-return-to-zero**) is the simplest, literal line code (+1V=“1”, -1V=“0”)
- Other codes tradeoff bandwidth and signal transitions



Four different line codes

Clock Recovery

To decode the symbols, signals need sufficient transitions

- Otherwise long runs of 0s (or 1s) are confusing, e.g.:



1 0 0 0 0 0 0 0 0 0 um, 0? er, 0?

Strategies:

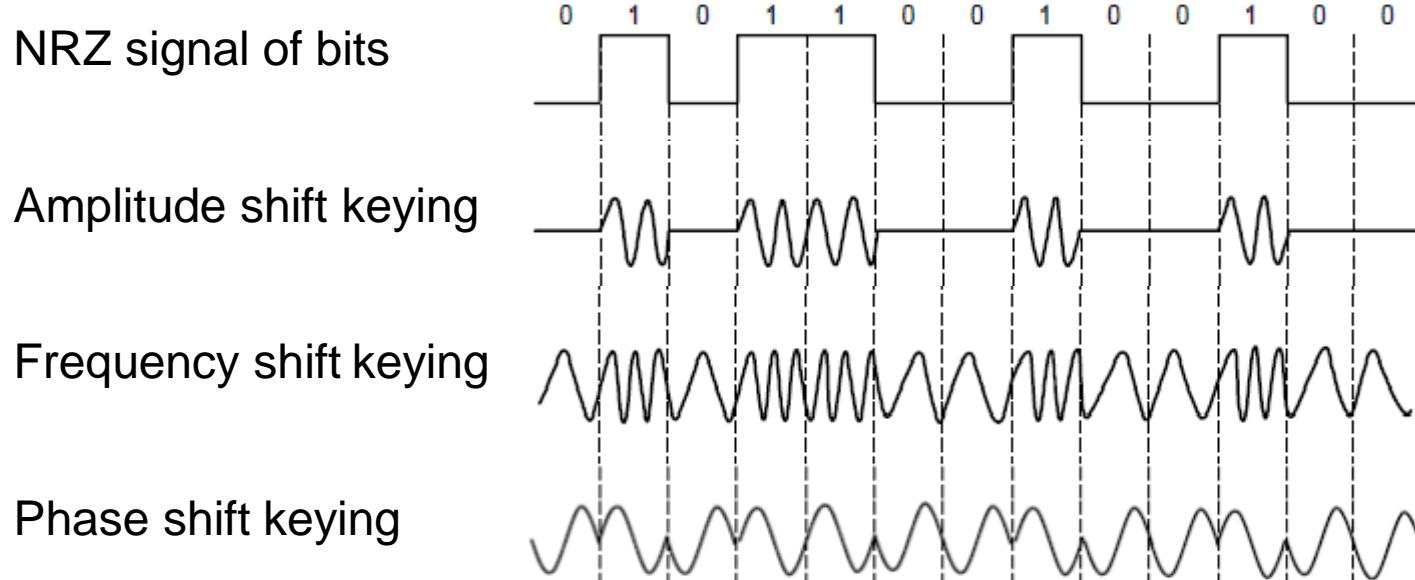
- Manchester coding, mixes clock signal in every symbol
- 4B/5B maps 4 data bits to 5 coded bits with 1s and 0s:

| Data | Code | Data | Code | Data | Code | Data | Code |
|------|-------|------|-------|------|-------|------|-------|
| 0000 | 11110 | 0100 | 01010 | 1000 | 10010 | 1100 | 11010 |
| 0001 | 01001 | 0101 | 01011 | 1001 | 10011 | 1101 | 11011 |
| 0010 | 10100 | 0110 | 01110 | 1010 | 10110 | 1110 | 11100 |
| 0011 | 10101 | 0111 | 01111 | 1011 | 10111 | 1111 | 11101 |

- Scrambler XORs tx/rx data with pseudorandom bits

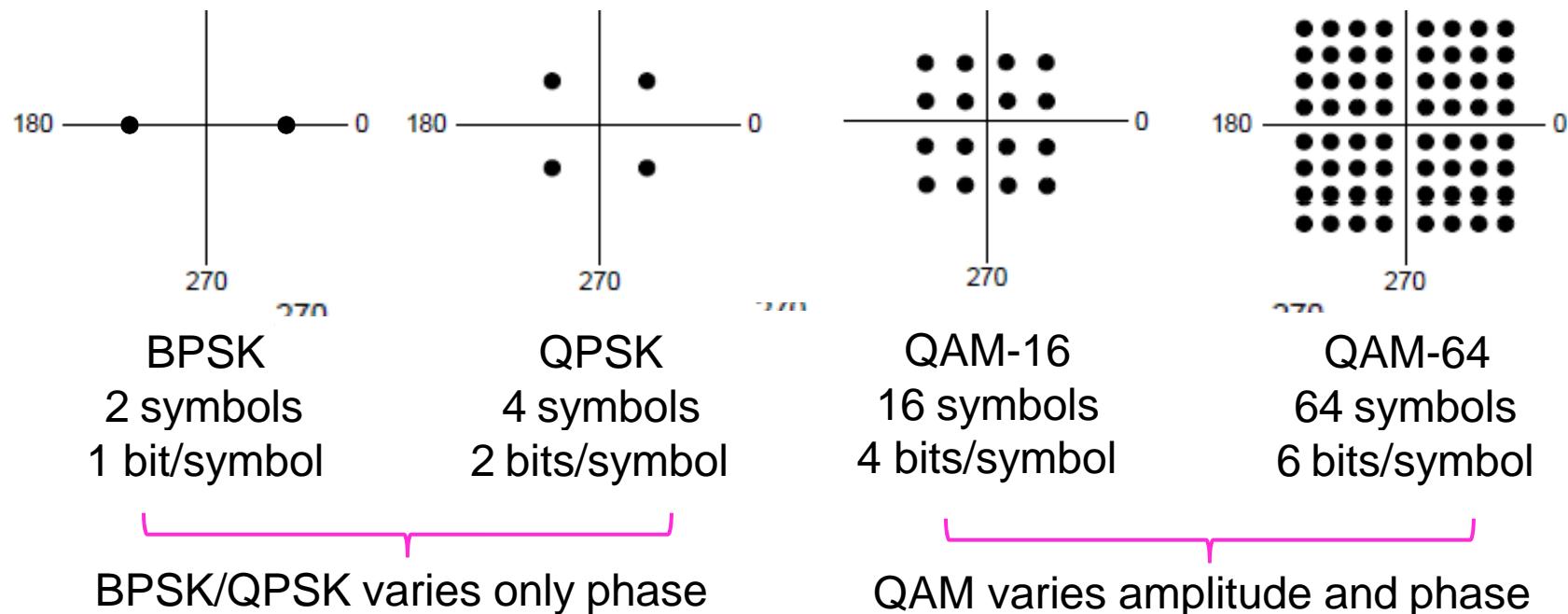
Passband Transmission (1)

Modulating the amplitude, frequency/phase of a carrier signal sends bits in a (non-zero) frequency range



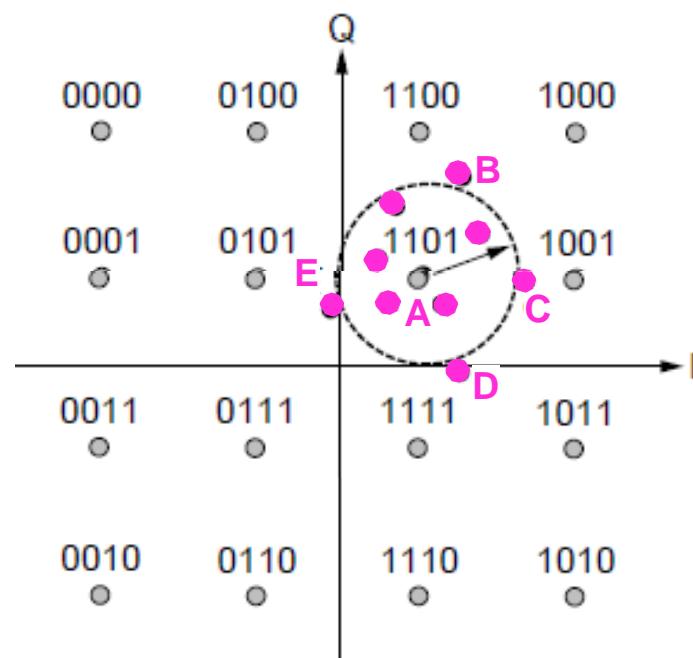
Passband Transmission (2)

Constellation diagrams are a shorthand to capture the amplitude and phase modulations of symbols:



Passband Transmission (3)

Gray-coding assigns bits to symbols so that small symbol errors cause few bit errors:

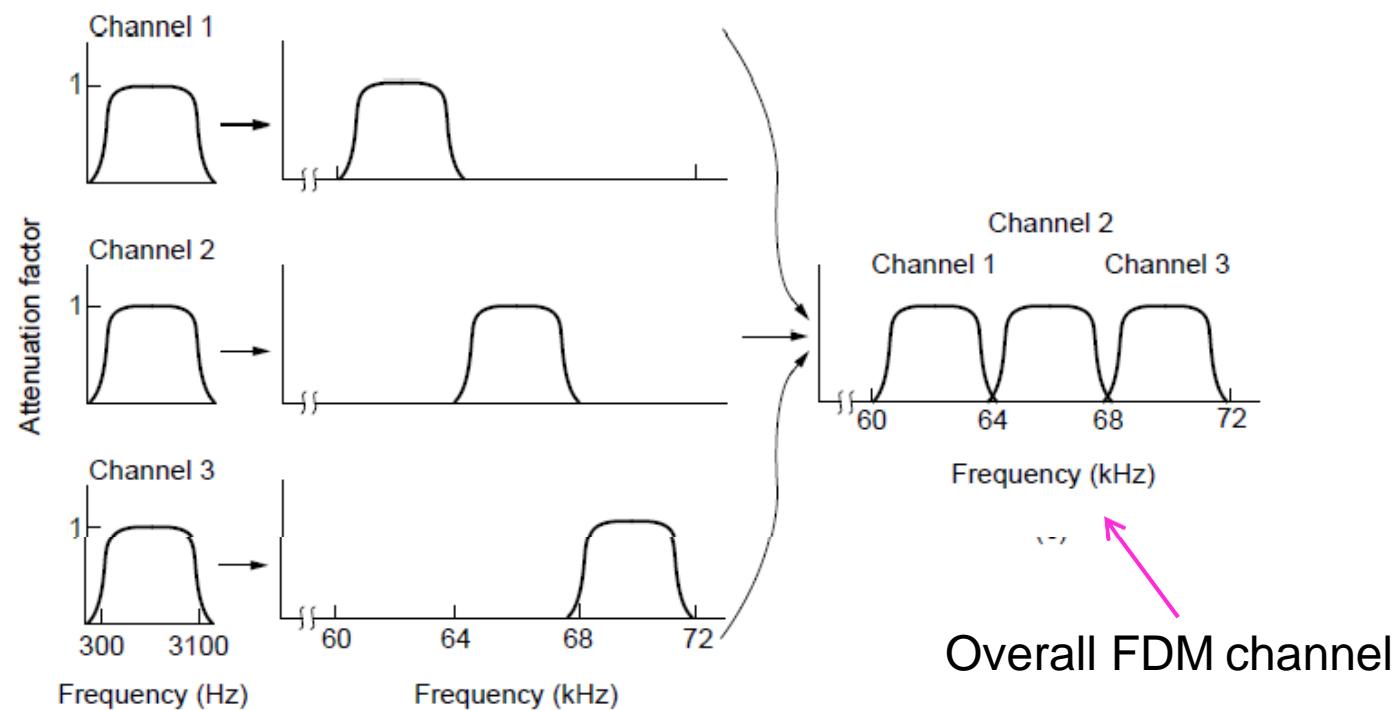


When 1101 is sent:

| Point | Decodes as | Bit errors |
|-------|-------------|------------|
| A | 1101 | 0 |
| B | <u>1100</u> | 1 |
| C | <u>1001</u> | 1 |
| D | <u>1111</u> | 1 |
| E | <u>0101</u> | 1 |

Frequency Division Multiplexing (1)

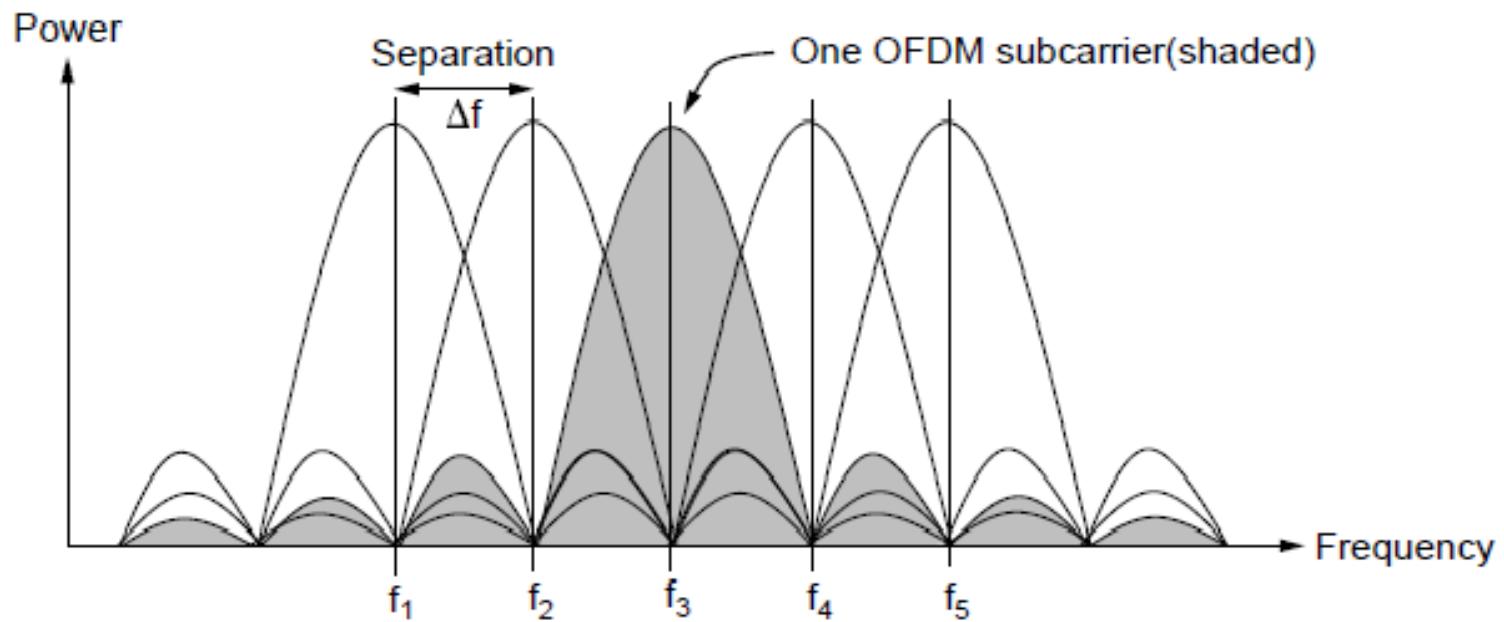
FDM (Frequency Division Multiplexing) shares the channel by placing users on different frequencies:



Frequency Division Multiplexing (2)

OFDM (Orthogonal FDM) is an efficient FDM technique used for 802.11, 4G cellular and other communications

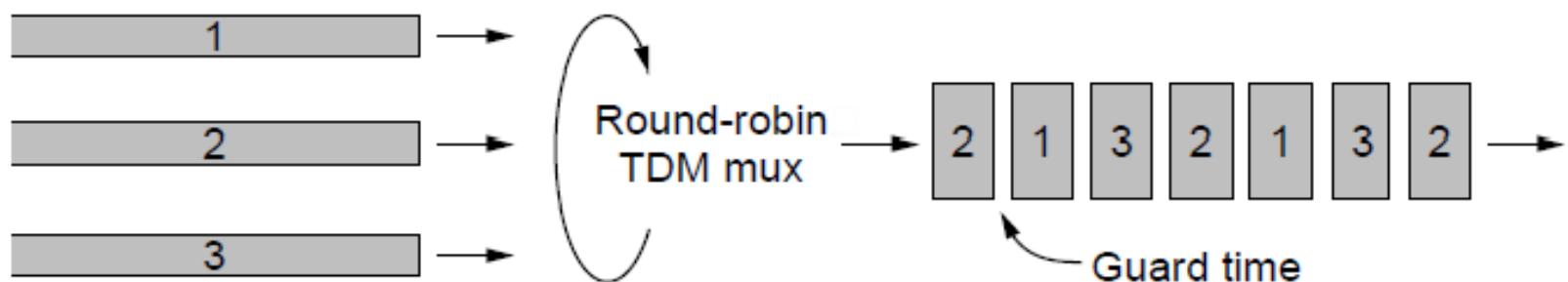
- Subcarriers are coordinated to be tightly packed



Time Division Multiplexing (TDM)

Time division multiplexing shares a channel over time:

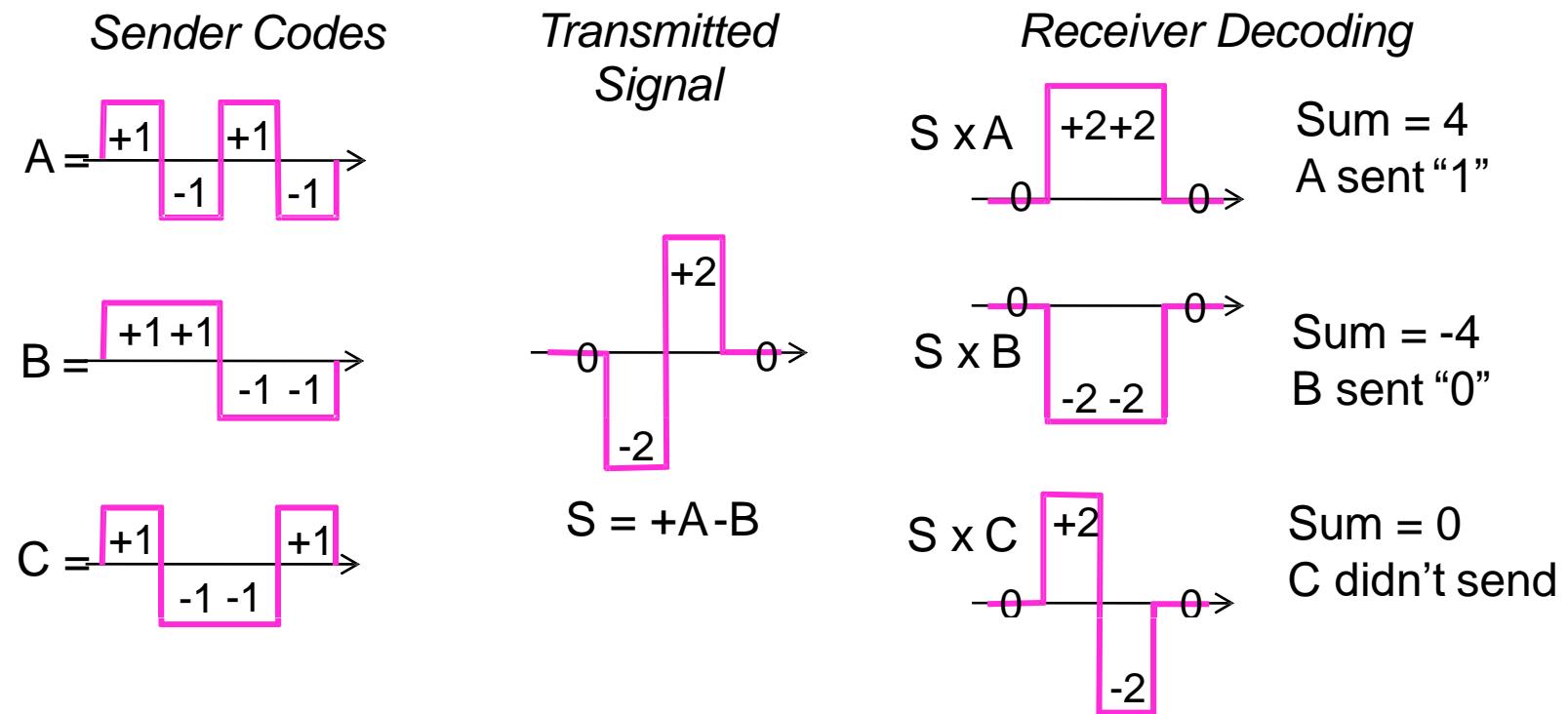
- Users take turns on a fixed schedule; this is not packet switching or STDM (Statistical TDM)
- Widely used in telephone / cellular systems



Code Division Multiple Access (CDMA)

CDMA shares the channel by giving users a code

- Codes are orthogonal; can be sent at the same time
- Widely used as part of 3G networks



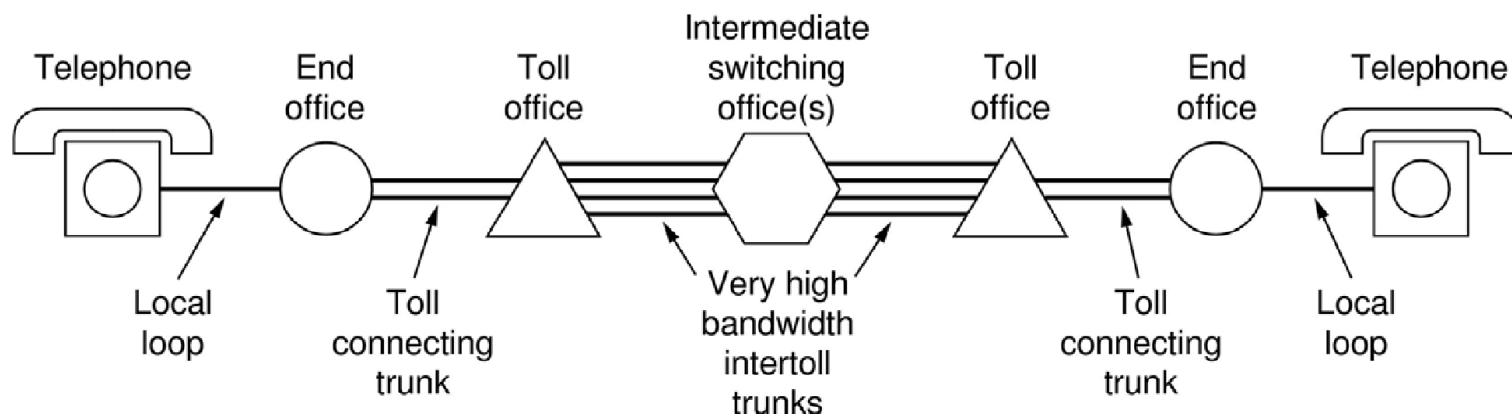
The Public Switched Telephone Network

- Structure of the telephone system »
- Politics of telephones »
- Local loop: modems, ADSL, and FTTH »
- Trunks and multiplexing »
- Switching »

Structure of the Telephone System

A hierarchical system for carrying voice calls made of:

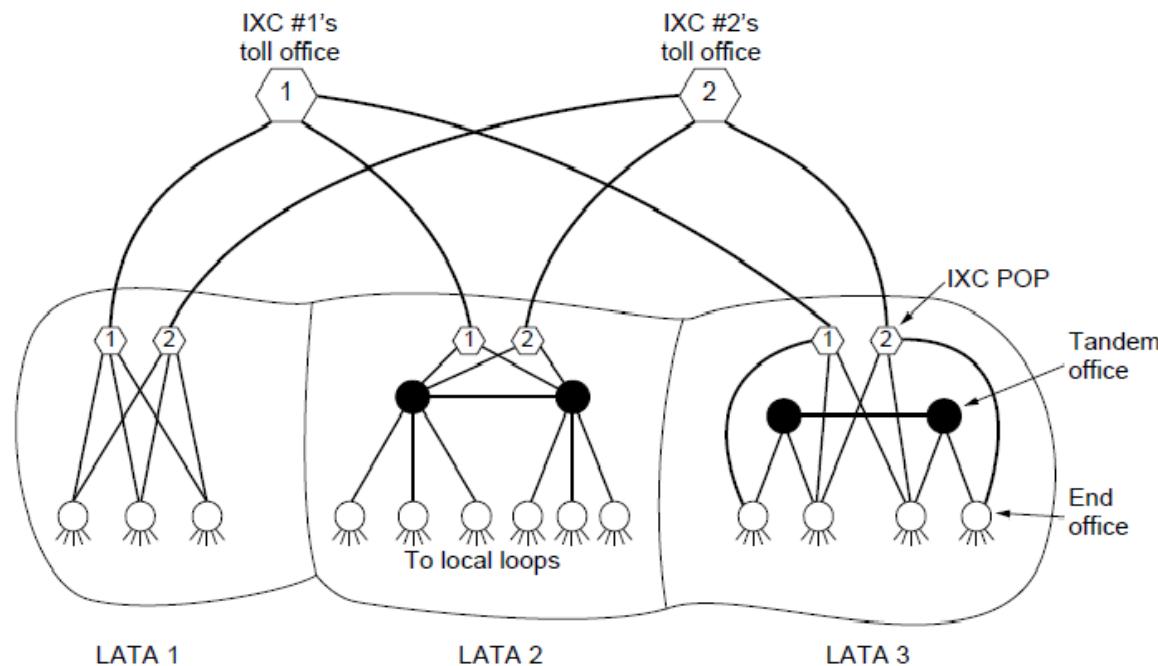
- Local loops, mostly analog twisted pairs to houses
- Trunks, digital fiber optic links that carry calls
- Switching offices, that move calls among trunks



The Politics of Telephones

In the U.S., there is a distinction for competition between serving a local area (LECs) and connecting to a local area (at a POP) to switch calls across areas (IXCs)

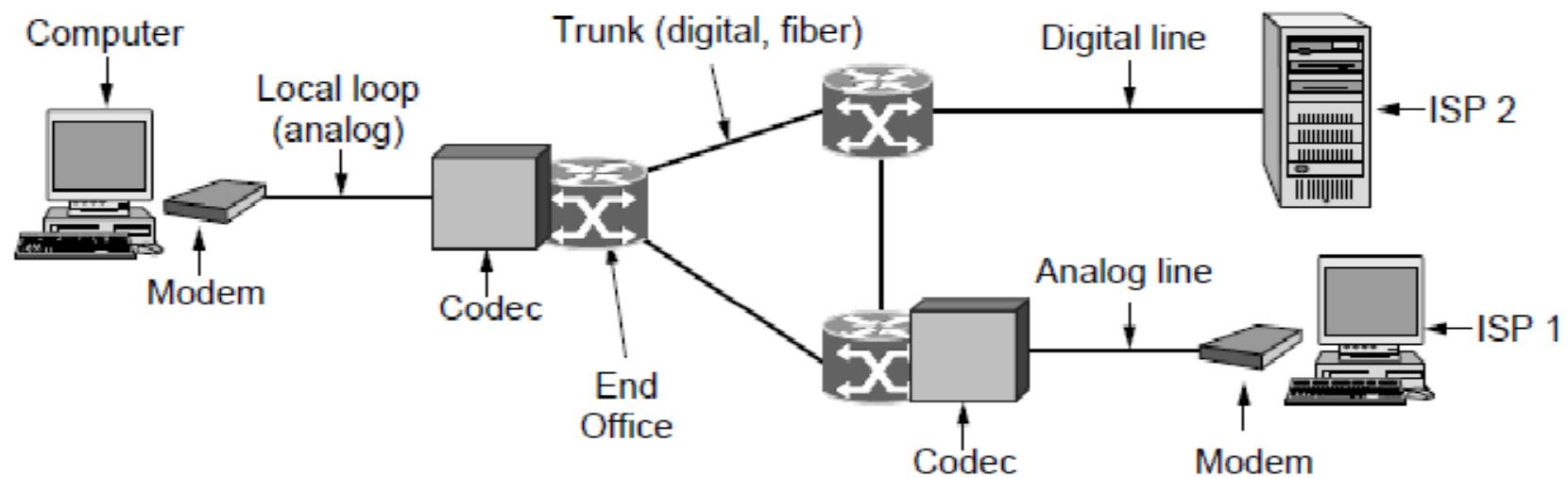
- Customers of a LEC can dial via any IXC they choose



Local loop (1): modems

Telephone modems send digital data over an 3.3 KHz analog voice channel interface to the POTS

- Rates <56 kbps; early way to connect to the Internet



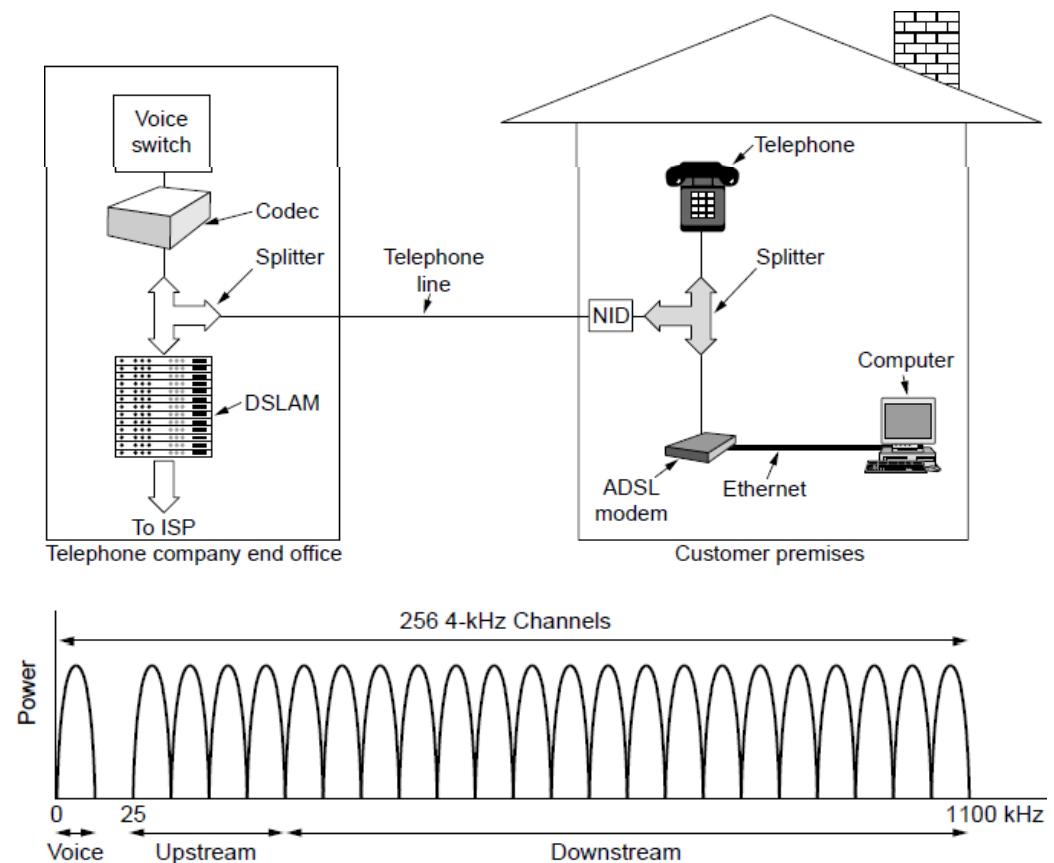
Local loop (2): Digital Subscriber Lines

DSL broadband sends data over the local loop to the local office using frequencies that are not used for POTS

- Telephone/computers attach to the same old phone line
- Rates vary with line
 - ADSL2 up to 12 Mbps
- OFDM is used up to 1.1 MHz for ADSL2
 - Most bandwidth down

OFDM: Orthogonal Frequency Division Multiplexing

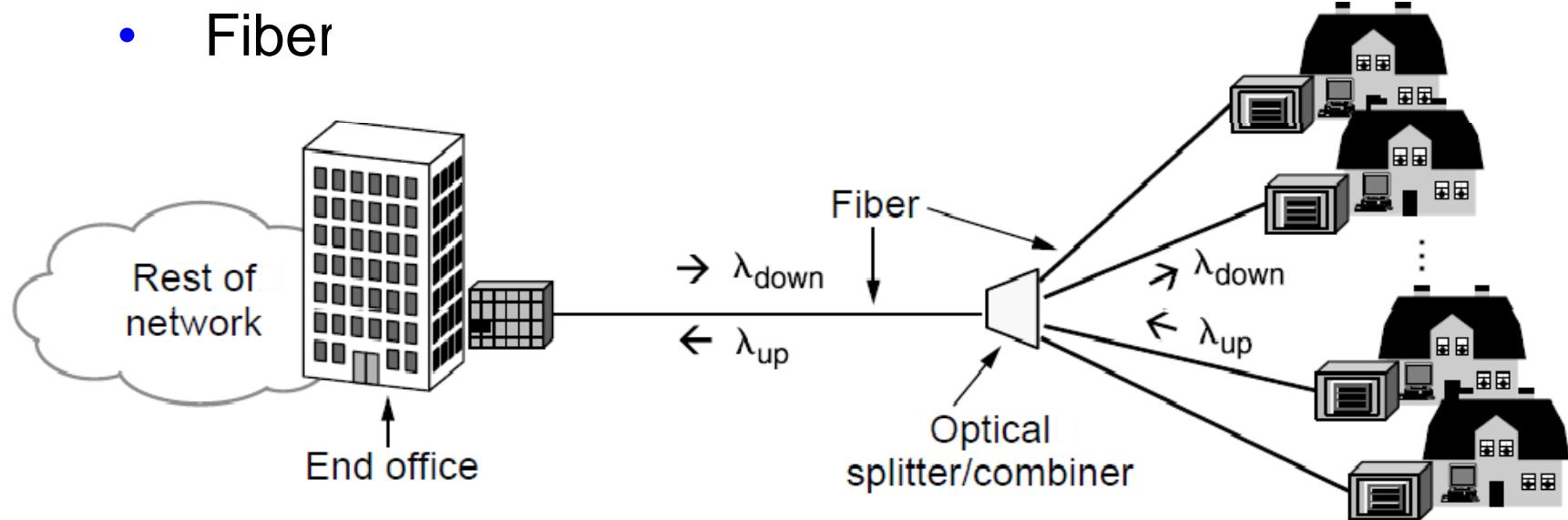
POTS: an analog voice transmission phone system implemented over copper twisted pair wires.



Local loop (3): Fiber To The Home

FTTH (Fiber To The Home) broadband relies on deployment of fiber optic cables to provide high data rates customers

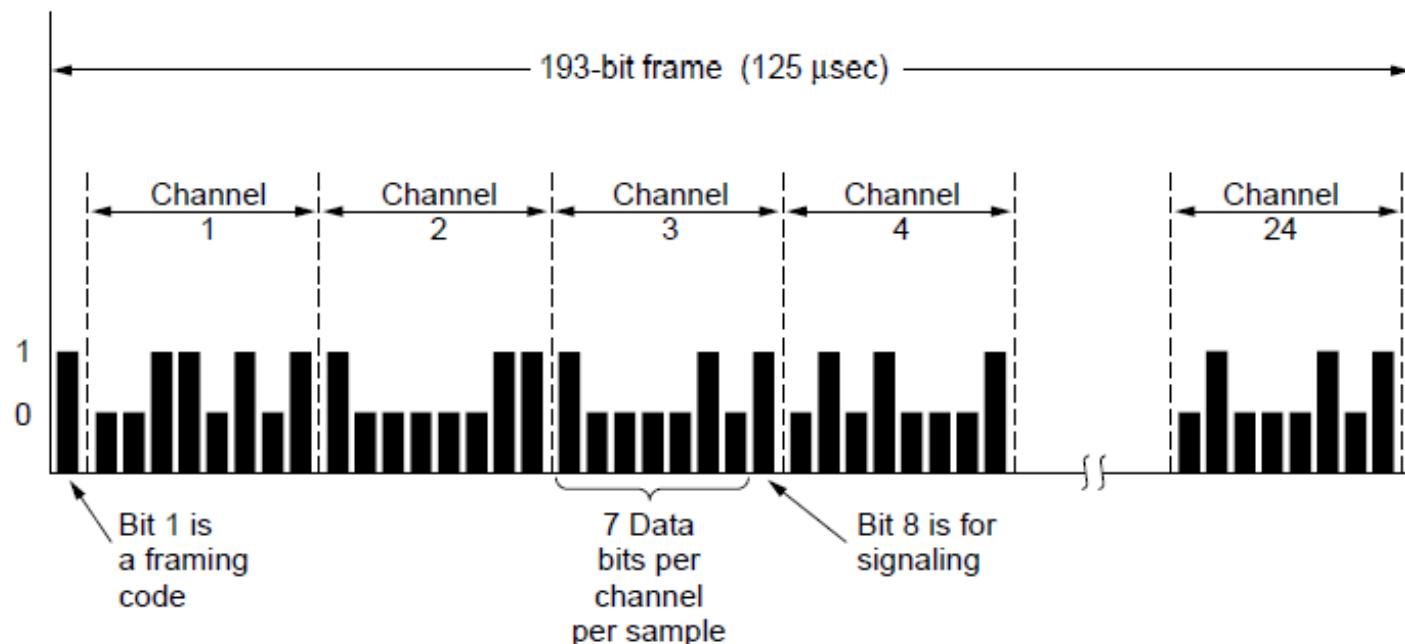
- One wavelength can be shared among many houses
- Fiber



Trunks and Multiplexing (1)

Calls are carried digitally on PSTN trunks using TDM

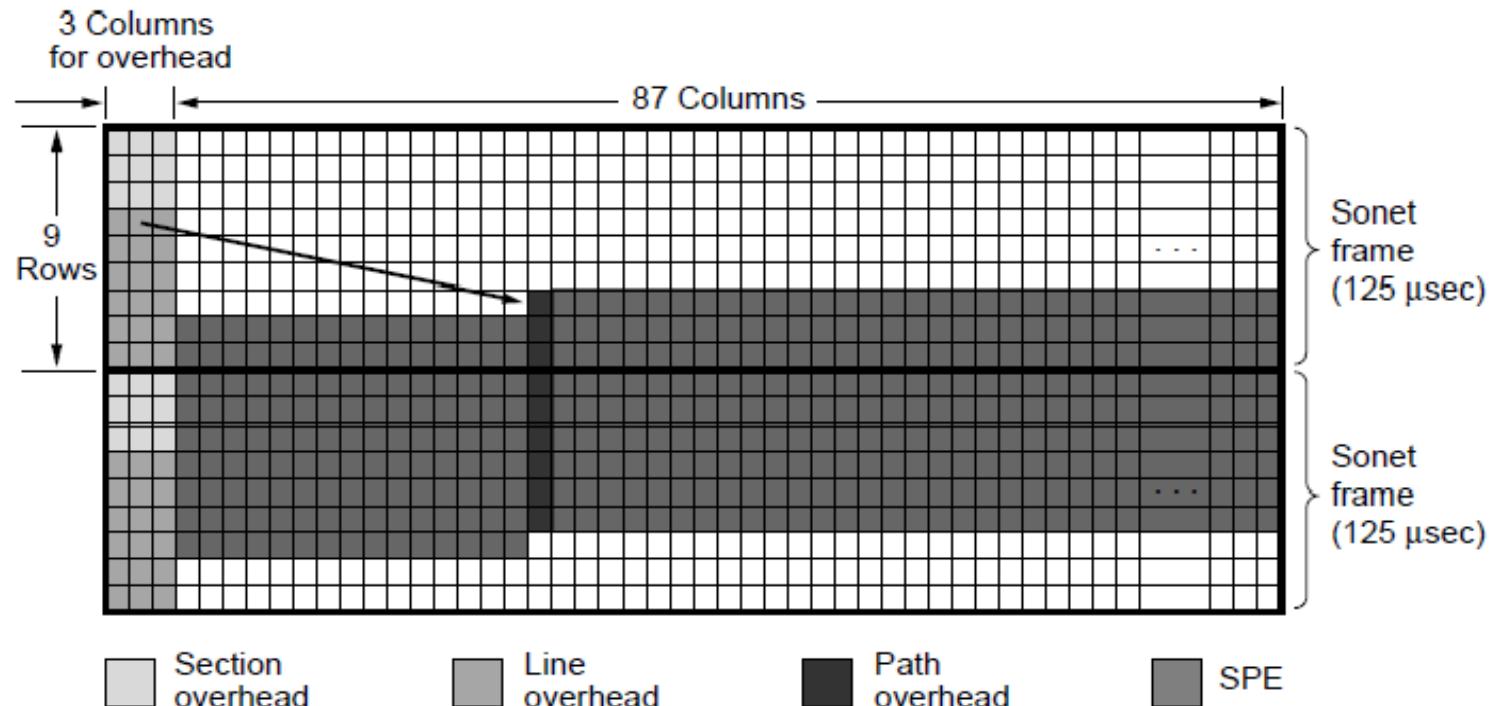
- A call is an 8-bit PCM sample each 125 μ s (64 kbps)
- Traditional T1 carrier has 24 call channels each 125 μ s (1.544 Mbps) with symbols based on AMI



Trunks and Multiplexing (2)

SONET (Synchronous Optical NETwork) is the worldwide standard for carrying digital signals on optical trunks

- Keeps 125 µs frame; base frame is 810 bytes (52Mbps)
- Payload “floats” within framing for flexibility



Trunks and Multiplexing (3)

Hierarchy at 3:1 per level is used for higher rates

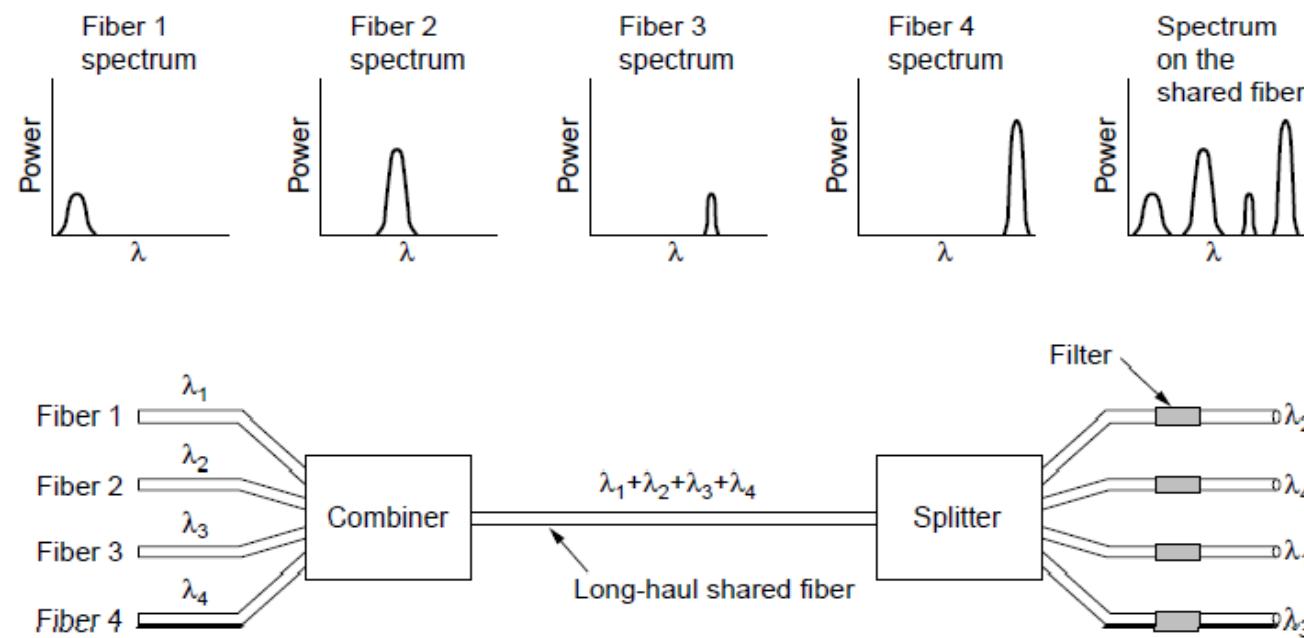
- Each level also adds a small amount of framing
- Rates from 50 Mbps (STS-1) to 40 Gbps (STS-768)

| SONET | | SDH | Data rate (Mbps) | | |
|------------|---------|---------|------------------|-----------|-----------|
| Electrical | Optical | Optical | Gross | SPE | User |
| STS-1 | OC-1 | | 51.84 | 50.112 | 49.536 |
| STS-3 | OC-3 | STM-1 | 155.52 | 150.336 | 148.608 |
| STS-12 | OC-12 | STM-4 | 622.08 | 601.344 | 594.432 |
| STS-48 | OC-48 | STM-16 | 2488.32 | 2405.376 | 2377.728 |
| STS-192 | OC-192 | STM-64 | 9953.28 | 9621.504 | 9510.912 |
| STS-768 | OC-768 | STM-256 | 39813.12 | 38486.016 | 38043.648 |

SONET/SDH rate hierarchy

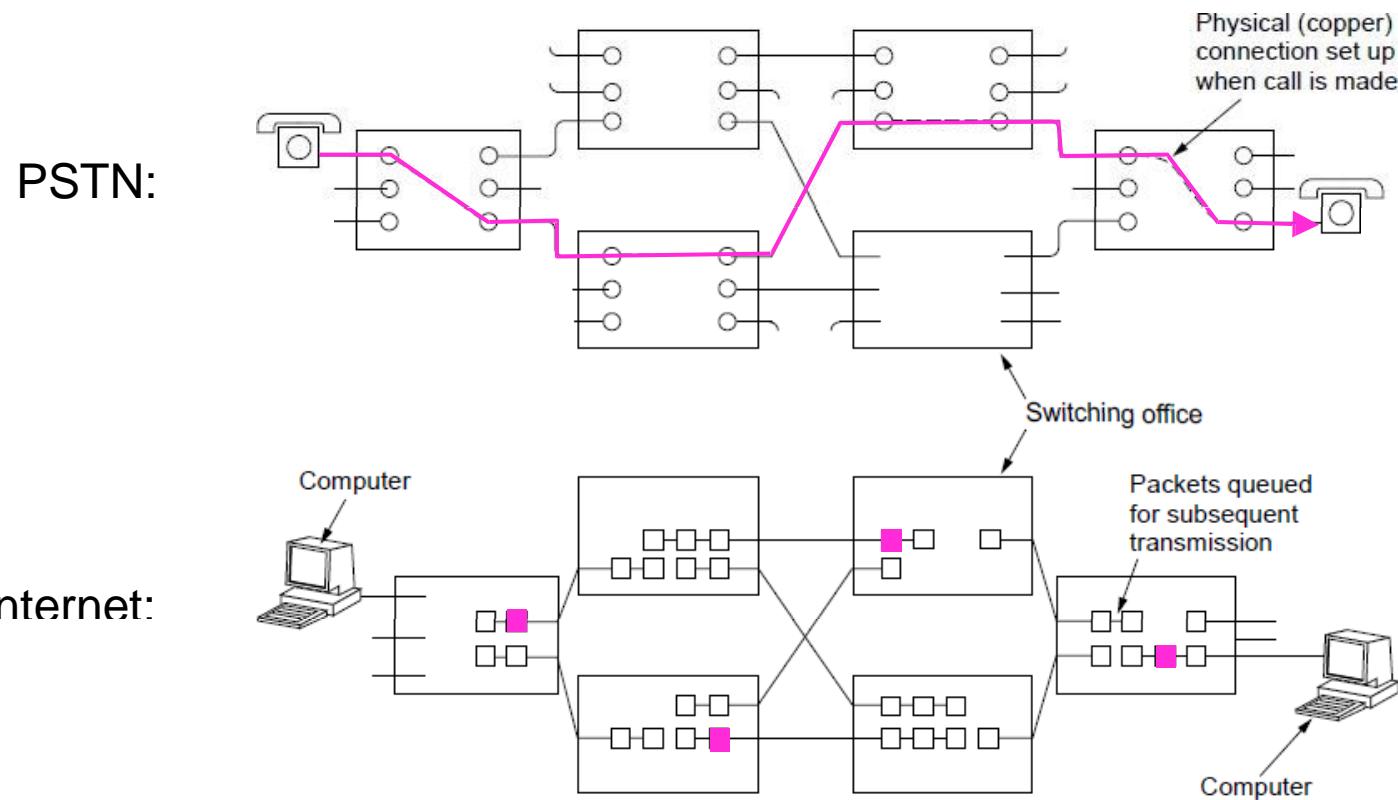
Trunks and Multiplexing (4)

WDM (Wavelength Division Multiplexing), another name for FDM, is used to carry many signals on one fiber:



Switching (1)

PSTN uses circuit switching; Internet uses packet switching



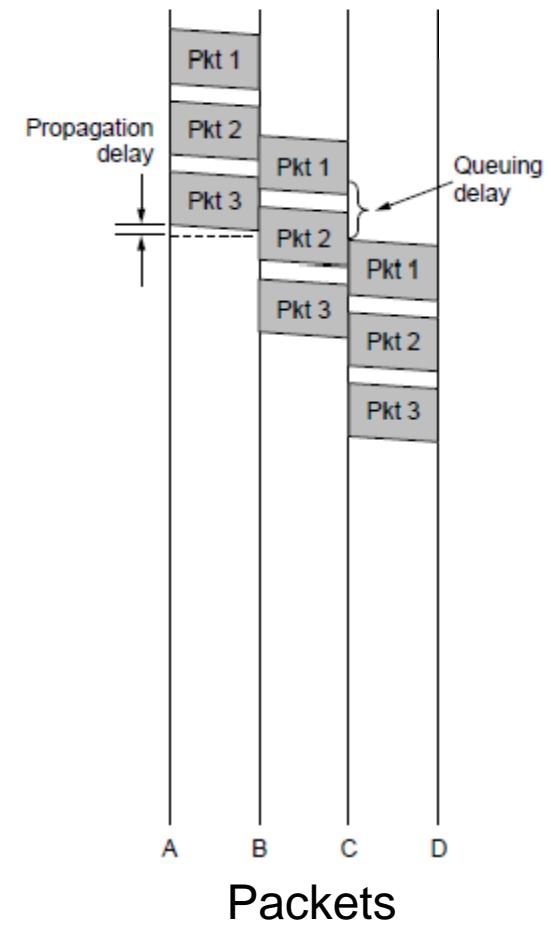
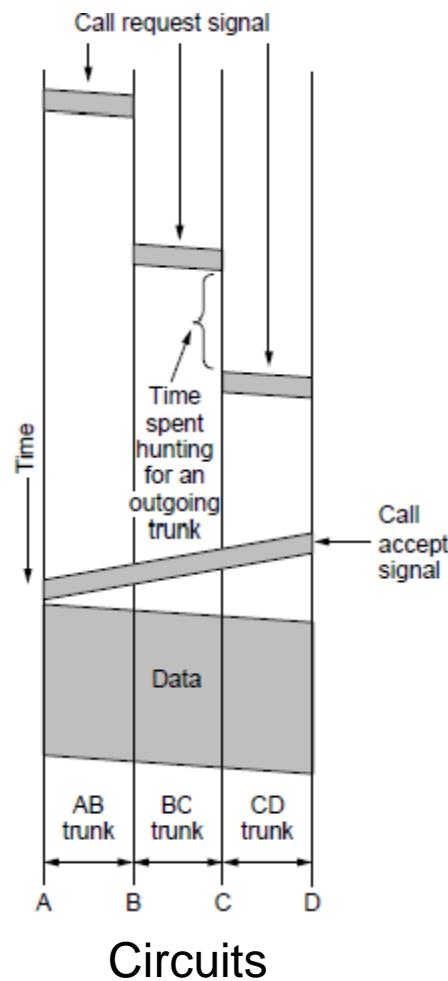
Switching (2)

Circuit switching requires call setup (connection) before data flows smoothly

- Also teardown at end (not shown)

Packet switching treats messages independently

- No setup, but variable queuing delay at routers



Switching (3)

Comparison of circuit- and packet-switched networks

| Item | Circuit switched | Packet switched |
|------------------------------------|------------------|-----------------|
| Call setup | Required | Not needed |
| Dedicated physical path | Yes | No |
| Each packet follows the same route | Yes | No |
| Packets arrive in order | Yes | No |
| Is a switch crash fatal | Yes | No |
| Bandwidth available | Fixed | Dynamic |
| Time of possible congestion | At setup time | On every packet |
| Potentially wasted bandwidth | Yes | No |
| Store-and-forward transmission | No | Yes |
| Charging | Per minute | Per packet |

Mobile Telephone System

- Generations of mobile telephone systems »
- Cellular mobile telephone systems »
- GSM, a 2G system »
- UMTS, a 3G system »

Generations of mobile telephone systems

1G, analog voice

- AMPS (Advanced Mobile Phone System) is example, deployed from 1980s. Modulation based on FM (as in radio).

2G, analog voice and digital data

- GSM (Global System for Mobile communications) is example, deployed from 1990s. Modulation based on QPSK.

3G, digital voice and data

- UMTS (Universal Mobile Telecommunications System) is example, deployed from 2000s. Modulation based on CDMA

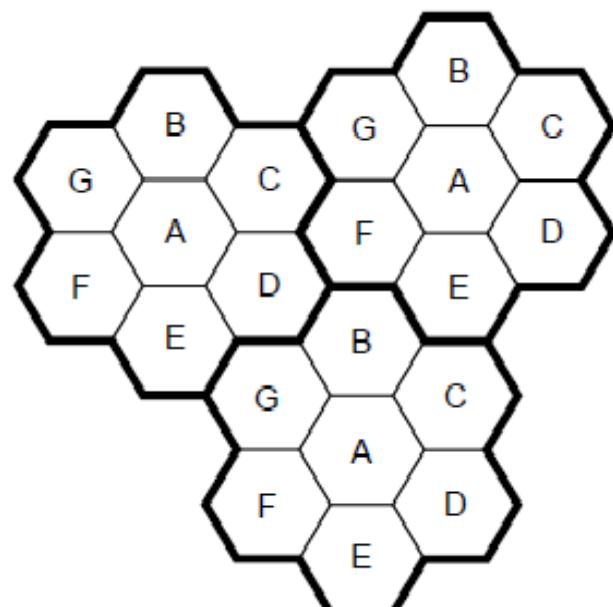
4G, digital data including voice

- LTE (Long Term Evolution) is example, deployed from 2010s. Modulation based on OFDM

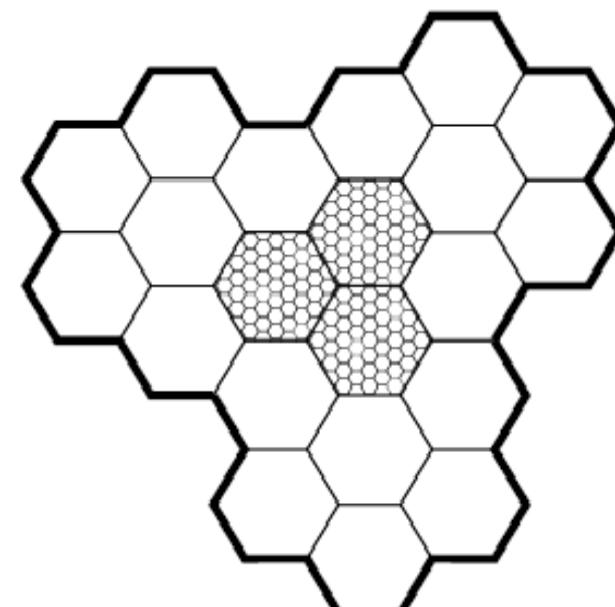
Cellular mobile phone systems

All based on notion of spatial regions called cells

- Each mobile uses a frequency in a cell; moves cause handoff
- Frequencies are reused across non-adjacent cells
- To support more mobiles, smaller cells can be used



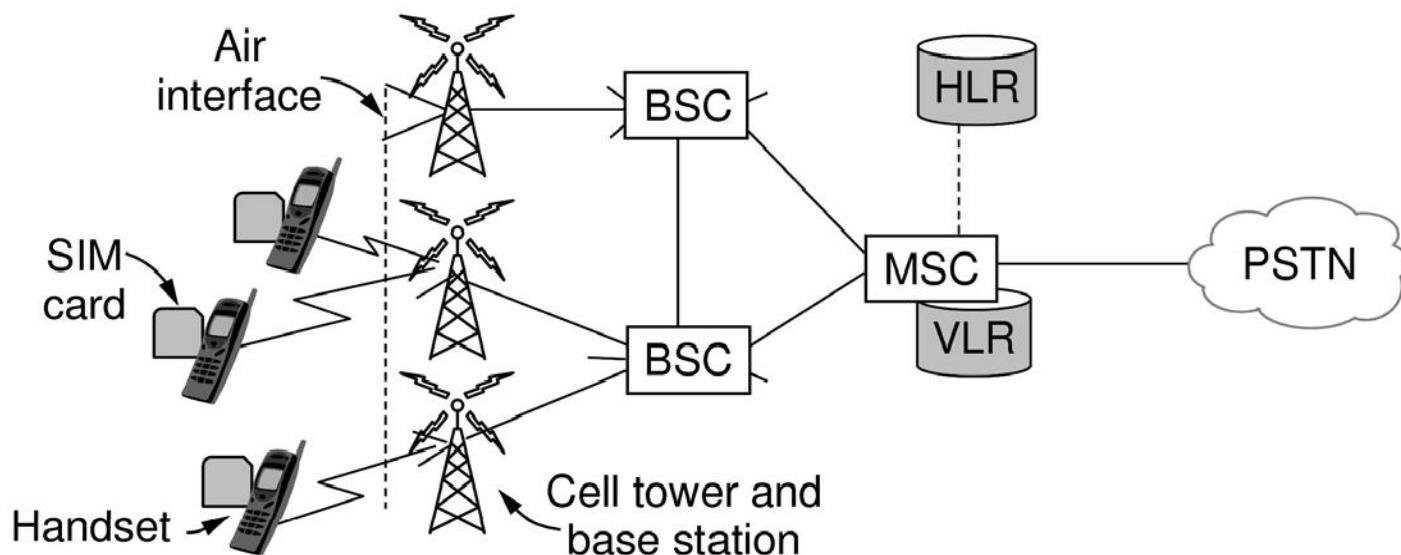
Cellular reuse pattern



Smaller cells for dense mobiles

GSM – Global System for Mobile Communications (1)

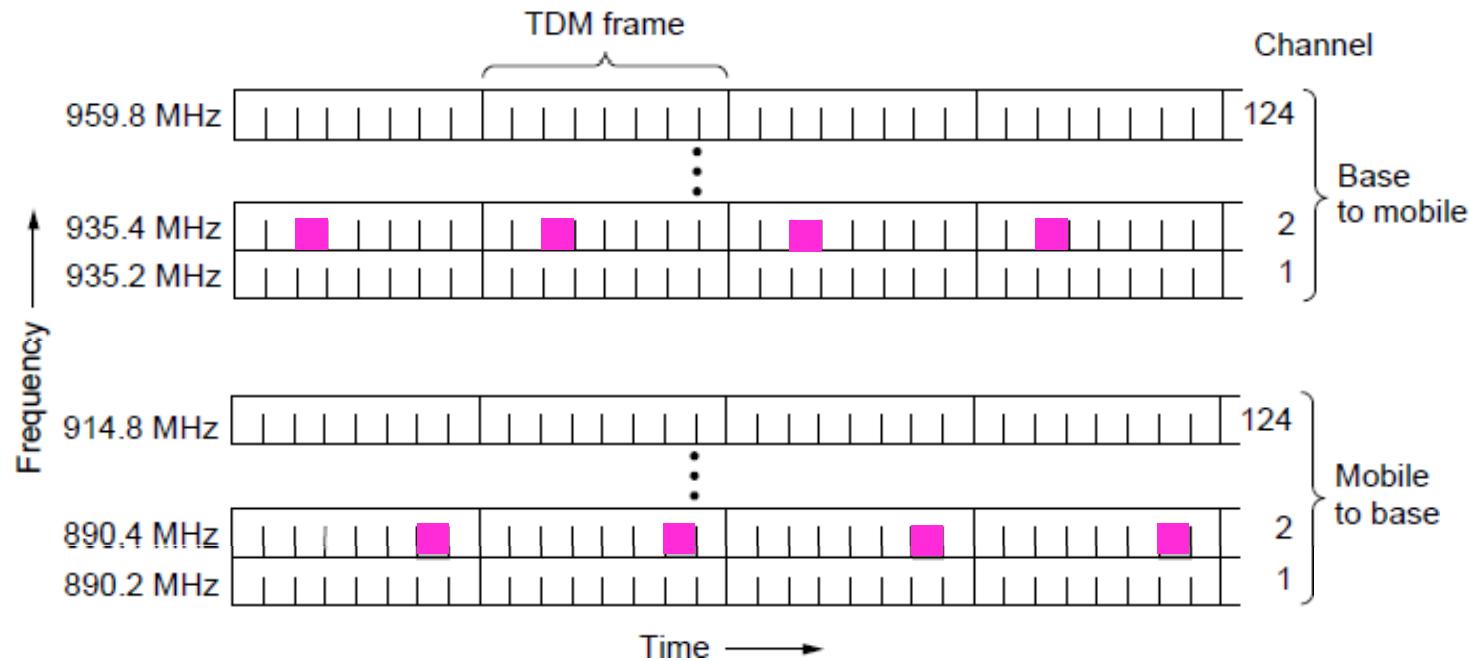
- Digital voice, Sophisticated technology
- Mobile is divided into handset and SIM card (Subscriber Identity Module) with credentials
- Mobiles tell their HLR (Home Location Register) their current whereabouts for incoming calls
- Cells keep track of visiting mobiles (in the Visitor LR)



GSM – Global System for Mobile Communications (2)

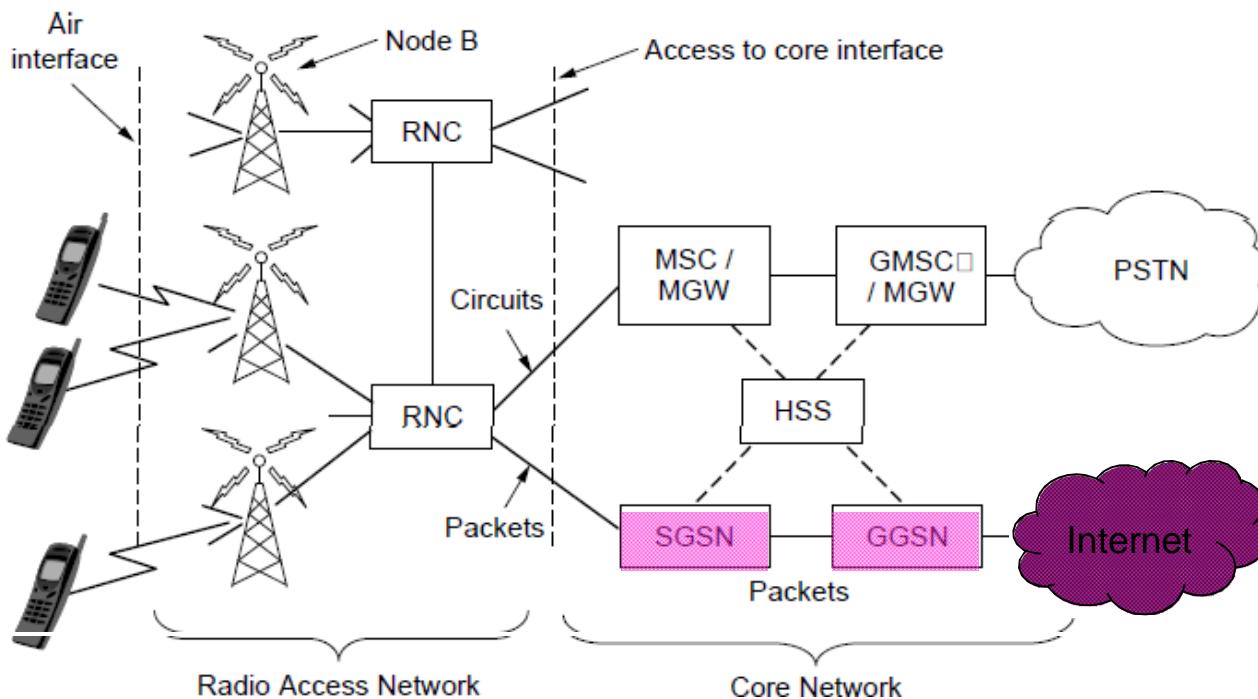
Air interface is based on FDM channels of 200 KHz divided in an eight-slot TDM frame every 4.615 ms

- Mobile is assigned up- and down-stream slots to use
- Each slot is 148 bits long, gives rate of 27.4 kbps



UMTS – Universal Mobile Telecommunications System (1)

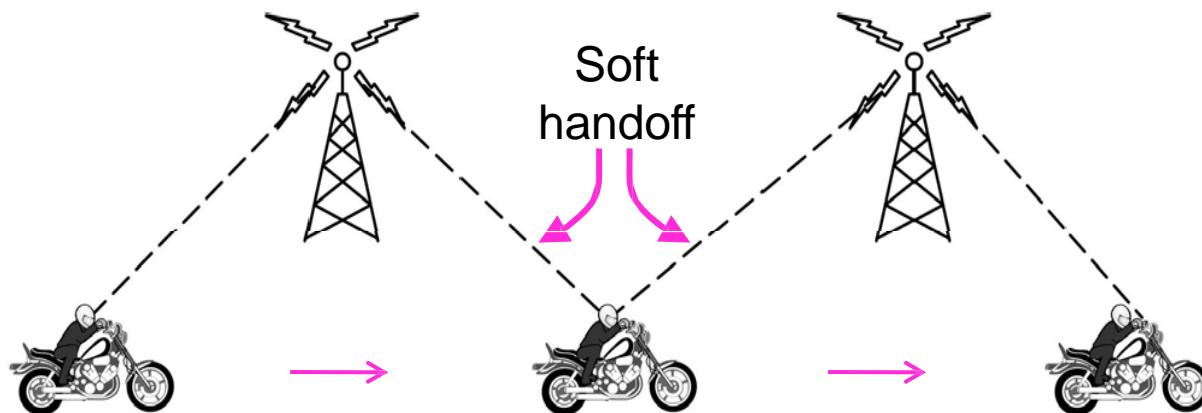
- Voice and data
- Architecture is an evolution of GSM; terminology differs
- Packets goes to/from the Internet via SGSN/GGSN



UMTS – Universal Mobile Telecommunications System (2)

Air interface based on CDMA over 5 MHz channels

- Rates over users <14.4 Mbps (HSPDA) per 5 MHz
- CDMA allows frequency reuse over all cells
- CDMA permits soft handoff (connected to both cells)



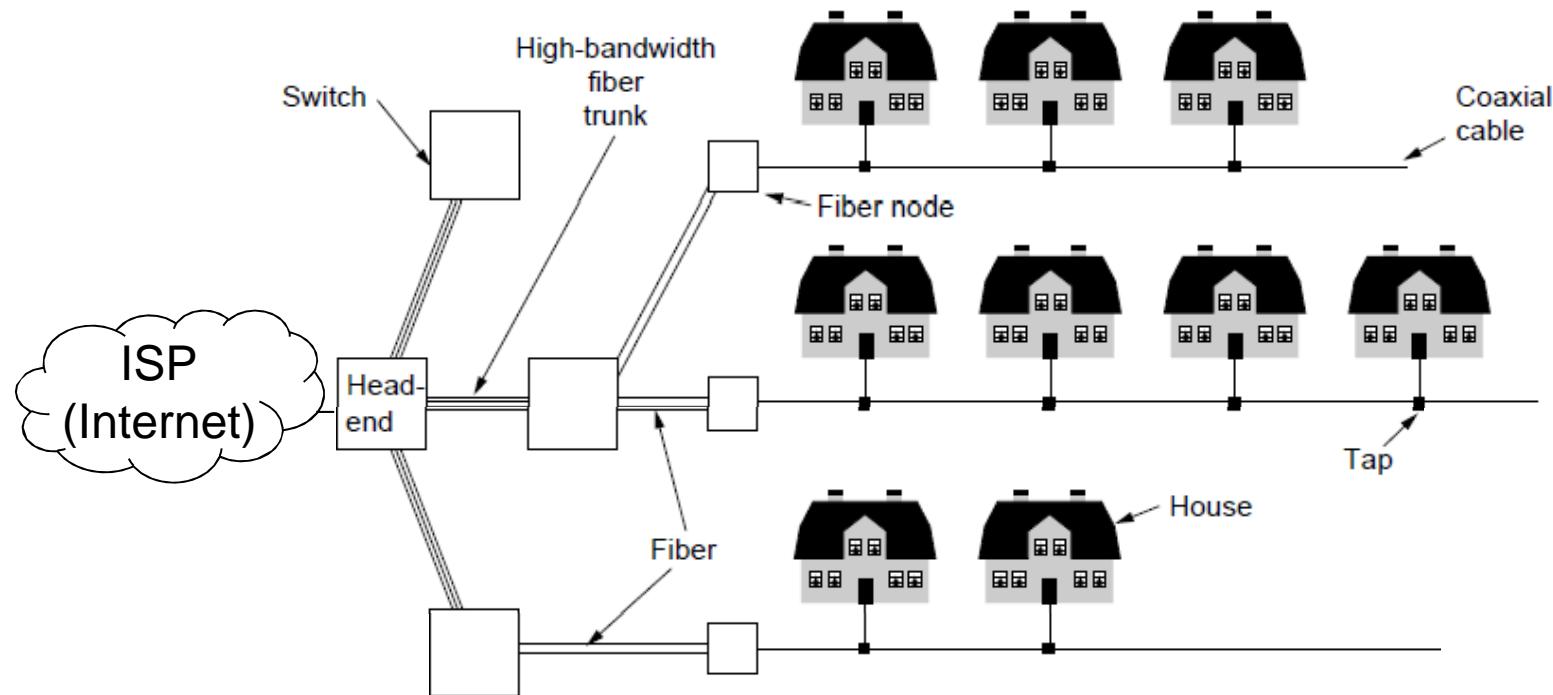
Cable Television

- Internet over cable »
- Spectrum allocation »
- Cable modems »
- ADSL vs. cable »

Internet over Cable

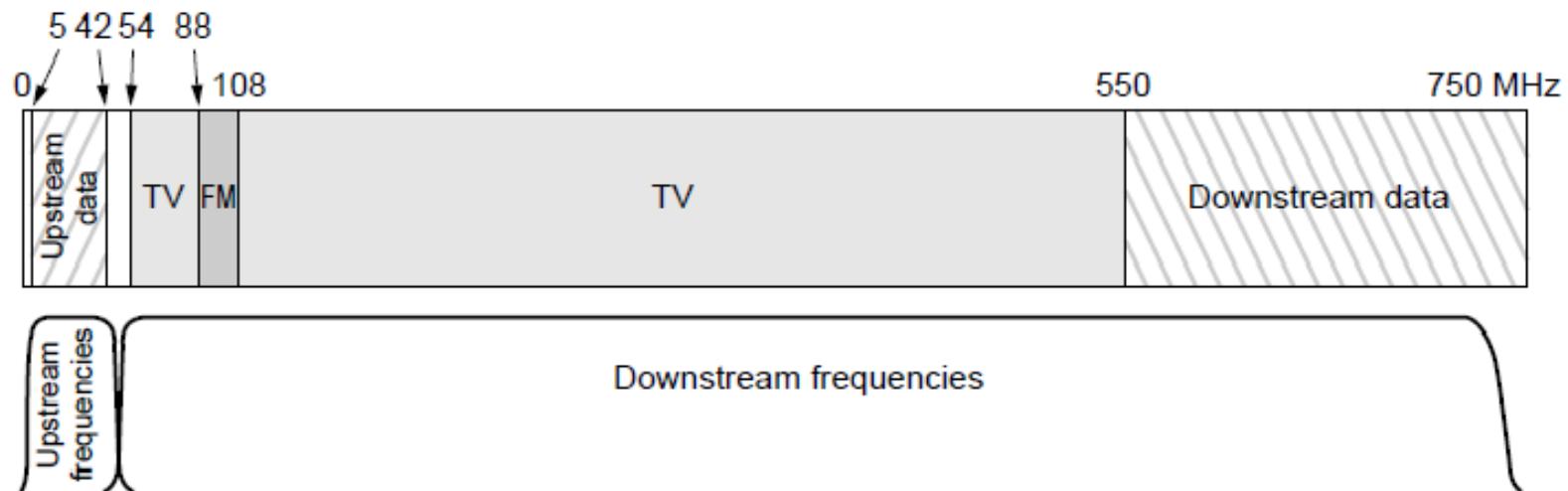
Internet over cable reuses the cable television plant

- Data is sent on the shared cable tree from the head-end, not on a dedicated line per subscriber (DSL)



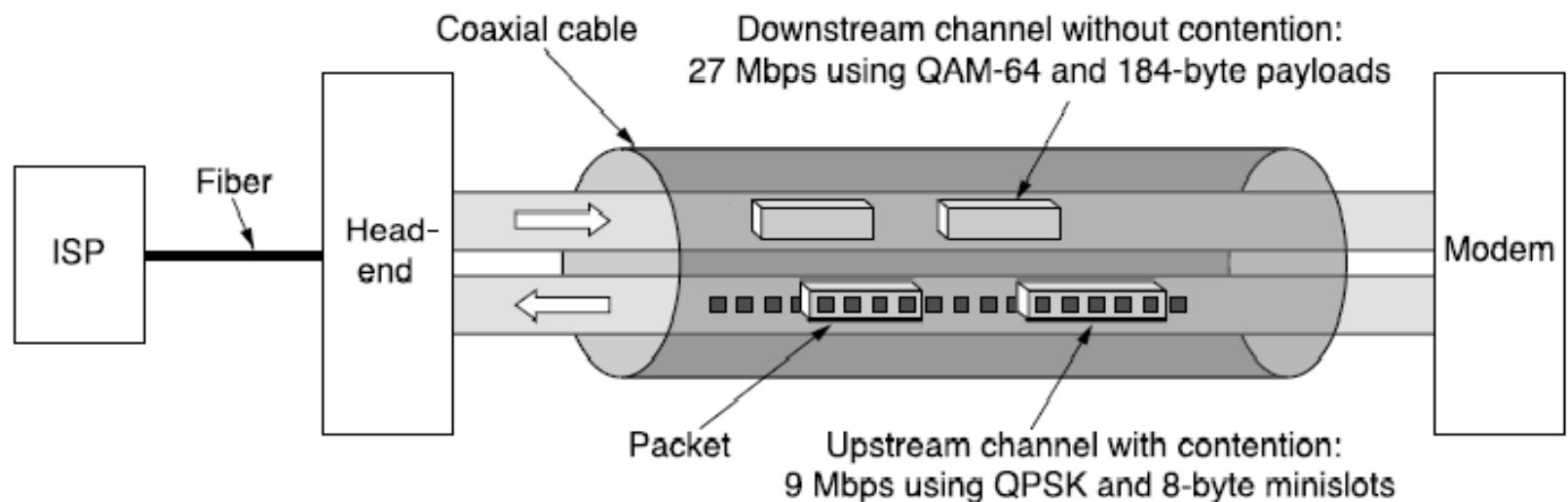
Spectrum Allocation

Upstream and downstream data are allocated to frequency channels not used for TV channels:



Cable Modems

- Internet access requires cable modem.
- Cable modems at customer premises implement the physical layer of the DOCSIS (data over cable service interface specification) standard
- Modem to computer is normally ethernet or USB. The other uses all of FDM, TDM and CDMA to share the bandwidth of the cable among the subscribers.



Cable vs. ADSL

Cable:

- + Uses coaxial cable to customers (good bandwidth)
- Data is broadcast to all customers (less secure)
- Bandwidth is shared over customers so may vary

ADSL:

- + Bandwidth is dedicated for each customer
- + Point-to-point link does not broadcast data
- Uses twisted pair to customers (lower bandwidth)

Summary

- Physical layer is the basis of all networks
- Nature imposes two limits to determine bandwidth
 - Nyquist limit which deals with noiseless channels
 - Shannon limit which deals with noisy channels
- Transmission media can be guided or unguided
 - Guided: twisted pair, coaxial cable, and fiber optics
 - Unguided: radio, microwaves, infrared, laser through air and satellites.
- Digital modulation methods send bits over guided and unguided media
- Line codes operate at baseband, and signals can be placed in a passband by modulating the amplitude, frequency, and phase of the carrier.
- Telephone system is the basis for wide area networks
- Main components: local loops, trunks, and switches
 - ADSL offers speeds upto 40Mbps better than local loop
 - PONN brings fiber to home even greater access rates than ADSL
- Trunks carry digital information. They are multiplexed with WDM and provide many high capacity links over fibers as well as TDM to share each high data rate link between users.

Summary

- Both circuit switching and packet switching is important
- For mobile applications, the fixed telephone system is not suitable.
- Mobile phones have gone through three generations: 1G, 2G, 3G
- Cable television system is alternative system for network access. Bandwidth depends on other users as it is shared among the users.

End

Chapter 2

The Datalink Layer

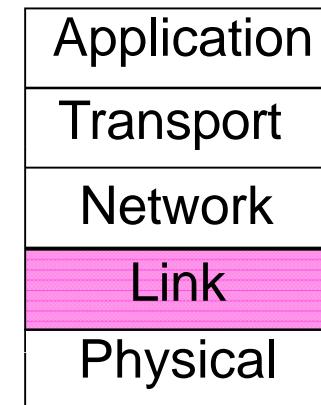
Chapter 3

- How to transmit information among two machines between two adjacent machines in a reliable and efficient manner?
 - Conceptually two machines are connected like a wire (coaxial cable, telephone line, or wireless channel)
 - There is a non-zero propagation delay
 - Communication channel makes errors

The Data Link Layer

Responsible for delivering **frames** of information over a single link

- Handles transmission errors and regulates the flow of data
- Provides well-defined service to network layer
- Dealing with transmission errors
- Regulates data between high-speed sender and slow speed receiver



Outline

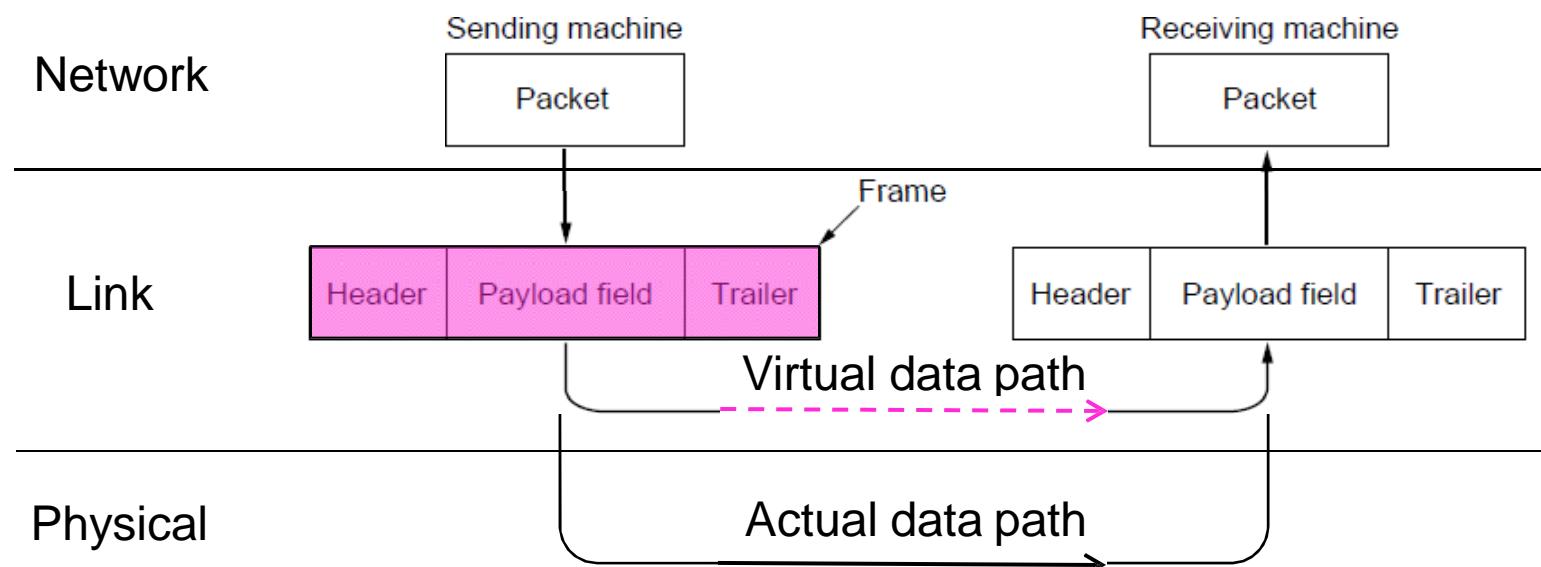
- Data Link Layer Design Issues
- Error Detection and Correction
- Elementary Data Link Protocols
- Sliding Window Protocols
- Example Data Link Protocols

Data Link Layer Design Issues

- Frames »
- Possible services »
- Framing methods »
- Error control »
- Flow control »

Frames

- Link layer accepts packets from the network layer, and encapsulates them into frames that it sends using the physical layer; reception is the opposite process



Possible Services

Unacknowledged connectionless service

- Frame is sent with no connection / error recovery
- Ethernet is an example

Acknowledged connectionless service

- Frame is sent with retransmissions if needed
- Example is 802.11 (WiFi)

Acknowledged connection-oriented service

- First phase: Connection is set-up
- Second phase: Frames are transmitted
- Third phase: connection is released.

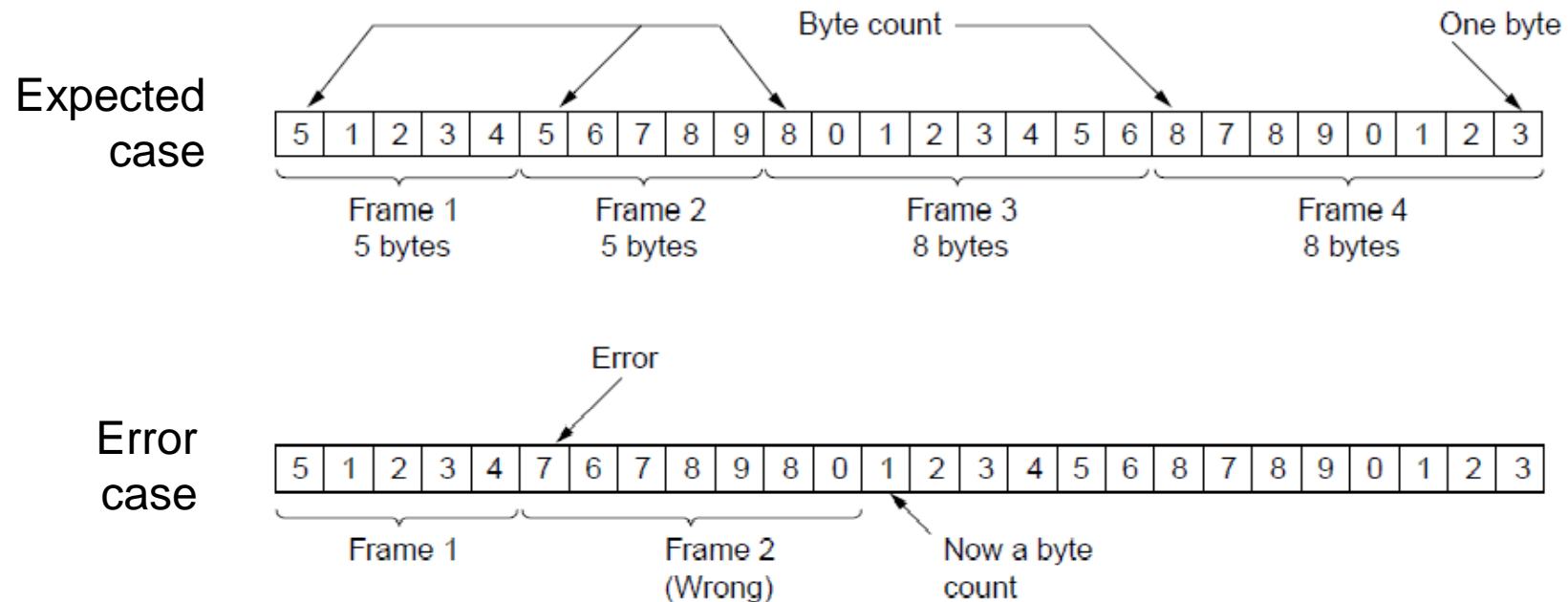
Framing Methods

- Datalink layer must use services provided by physical layer
 - The physical layer may add some redundant signals, but the transmission may not be error free
 - It is up to datalink layer to detect, if necessary, correct errors.
- Approach
 - Break the bitstream into discrete frames, compute the checksum and include checksum with the frame. When the frame arrives, checksum is recomputed. If it is different, steps will be taken to deal with the issue.
- Four methods of framing
 - Byte count »
 - Flag bytes with byte stuffing »
 - Flag bits with bit stuffing »
 - Physical layer coding violations
 - Use non-data symbol to indicate frame

Framing – Byte count

Frame begins with a count of the number of bytes in it

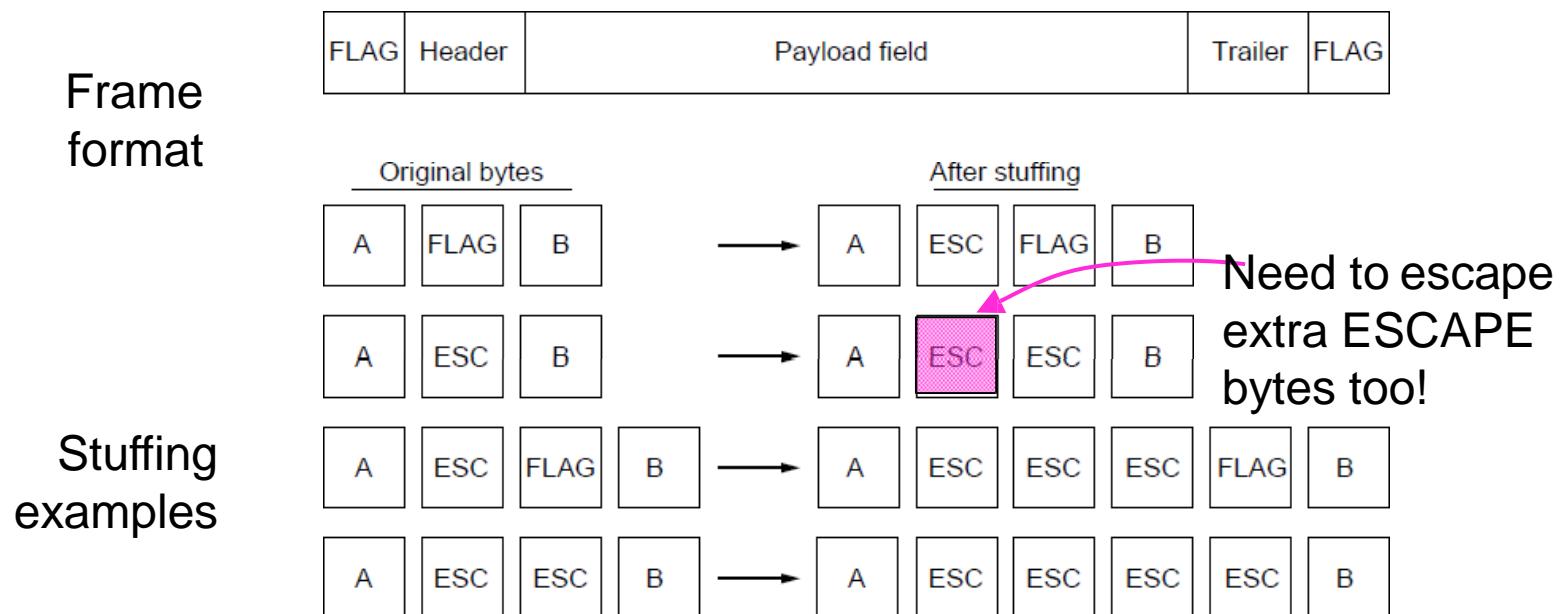
- Simple, but difficult to resynchronize after an error



Framing – Byte stuffing

Special flag bytes delimit frames; occurrences of flags in the data must be stuffed (escaped), at the both end of frame

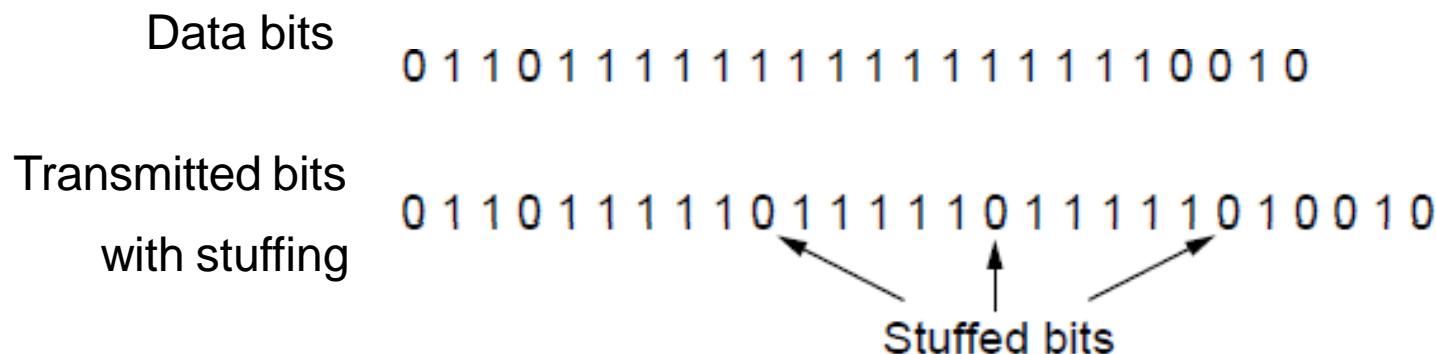
- Longer, but easy to resynchronize after error



Framing – Bit stuffing

Stuffing done at the bit level:

- Frame flag has six consecutive 1s (not shown)
- On transmit, after five 1s in the data, a 0 is added
- On receive, a 0 after five 1s is deleted



About Error Control

- Issue: How to make sure all frames are eventually delivered to the network layer in order?
- For connectionless service, it is OK if the sender keeps sending the frames continuously
- However, for reliable, connection-oriented service, it would not be fine at all.
- Approach
 - Error control repairs frames that are received in error
 - Requires errors to be detected at the receiver
 - Feedback in terms of positive and negative acks.
 - Timer protects against lost acknowledgements
 - Typically retransmit the unacknowledged frames
 - When frames are transmitted multiple times, there is an issue
 - Assign sequence numbers

Design Issue: Flow Control

How to prevent a fast sender from out-pacing a slow receiver?

- Feedback based flow control
 - Receiver sends back information to the sender giving it the permission to send the data
- Rate-based flow control
 - Built-in mechanism that limits the rate.
 - Overall, a set of well-defined rules are formed
 - Receiver gives feedback on the data it can accept
 - Rare in the Link layer as NICs run at “wire speed”
 - Receiver can take data as fast as it can be sent

Flow control is a topic in the Link and Transport layers.

Error Detection and Correction

Error codes add structured redundancy to data so errors can be either detected, or corrected.

Error correction codes:

- Hamming codes »
- Binary convolutional codes »
- Reed-Solomon and Low-Density Parity Check codes
 - Mathematically complex, widely used in real systems

Error detection codes:

- Parity »
- Checksums »
- Cyclic redundancy codes »

Error Bounds – Hamming distance

Code turns data of n bits into codewords of $n+k$ bits

Hamming distance is the minimum bit flips to turn one valid codeword into any other valid one.

- Example with 4 codewords of 10 bits ($n=2, k=8$):
 - 0000000000, 0000011111, 1111100000, and 1111111111
 - Hamming distance is 5

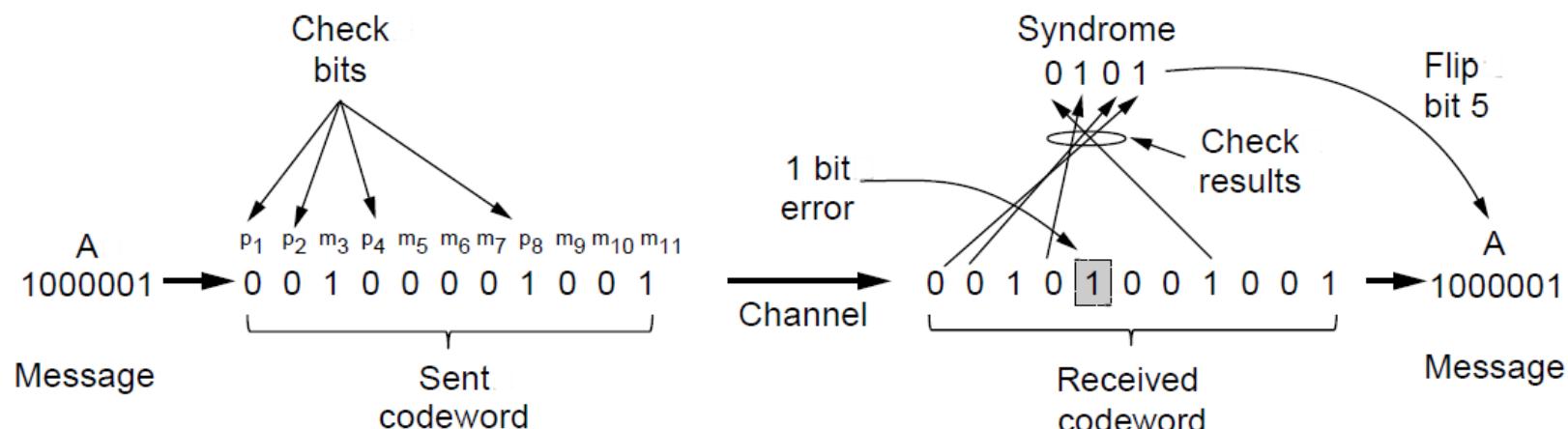
Bounds for a code with distance:

- $2d+1$ – can correct d errors (e.g., 2 errors above)
- $d+1$ – can detect d errors (e.g., 4 errors above)

Error Correction – Hamming code

Hamming code gives a simple way to add check bits and correct up to a single bit error:

- Check bits are parity over subsets of the codeword
- Recomputing the parity sums (syndrome) gives the position of the error to flip, or 0 if there is no error

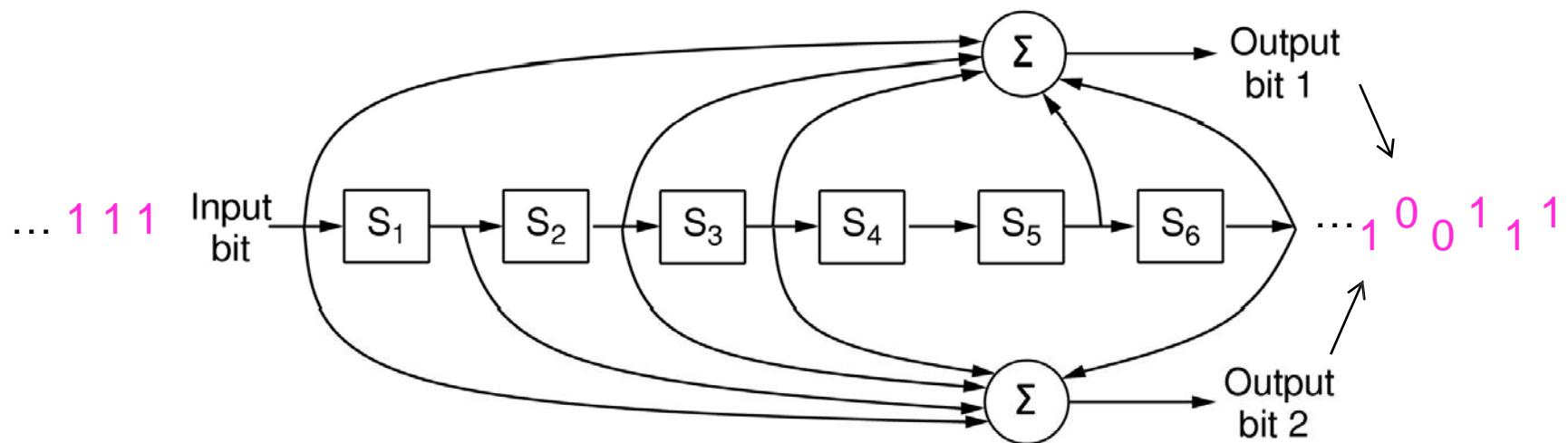


(11, 7) Hamming code adds 4 check bits and can correct 1 error

Error Correction – Convolutional codes

Operates on a stream of bits, keeping internal state

- Output stream is a function of all preceding input bits
- Bits are decoded with the Viterbi algorithm



Popular NASA binary convolutional code (rate = $\frac{1}{2}$) used in 802.11

Error Detection – Parity (1)

Parity bit is added as the modulo 2 sum of data bits

- Equivalent to XOR; this is even parity
- Ex: 1110000 → 11100001
- Detection checks if the sum is wrong (an error)

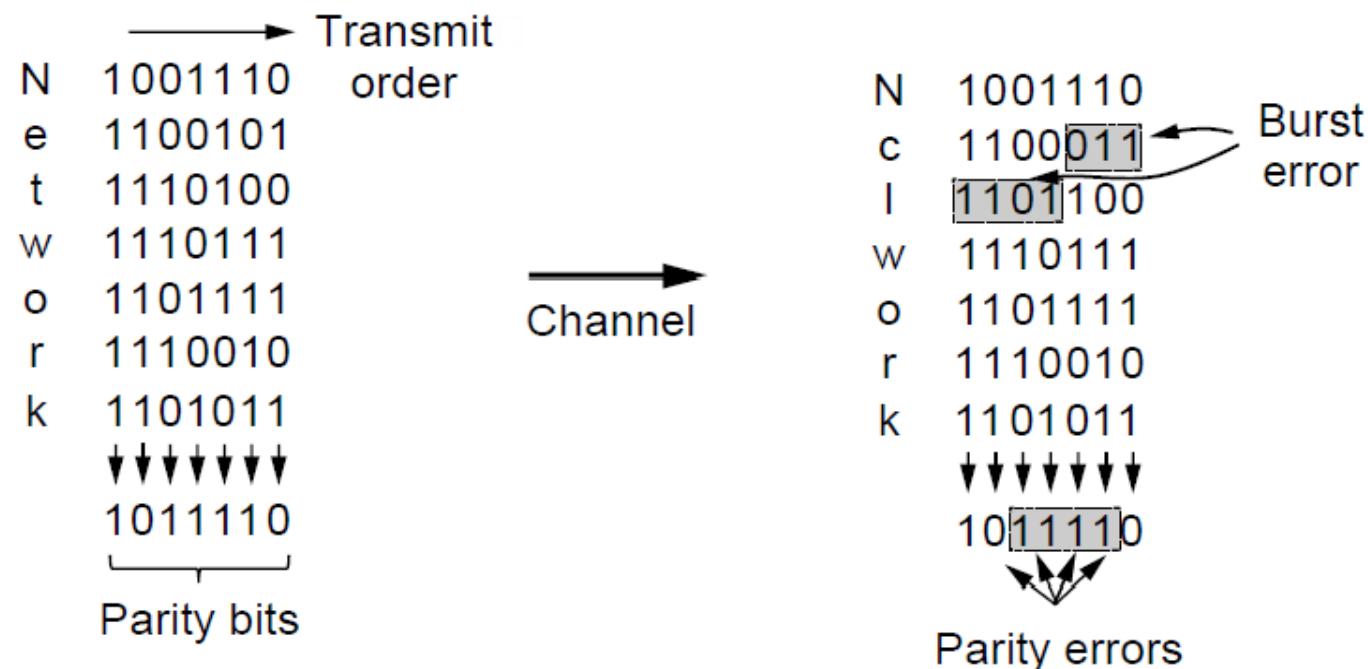
Simple way to detect an *odd* number of errors

- Ex: 1 error, 11100101; detected, sum is wrong
- Ex: 3 errors, 11011001; detected sum is wrong
- Ex: 2 errors, 11101101; *not detected*, sum is right!
- Error can also be in the parity bit itself
- Random errors are detected with probability $\frac{1}{2}$

Error Detection – Parity (2)

Interleaving of N parity bits detects burst errors up to N

- Each parity sum is made over non-adjacent bits
- An even burst of up to N errors will not cause it to fail



Error Detection – Checksums

Checksum treats data as N-bit words and adds N check bits that are the modulo 2^N sum of the words

- Ex: Internet 16-bit 1s complement checksum

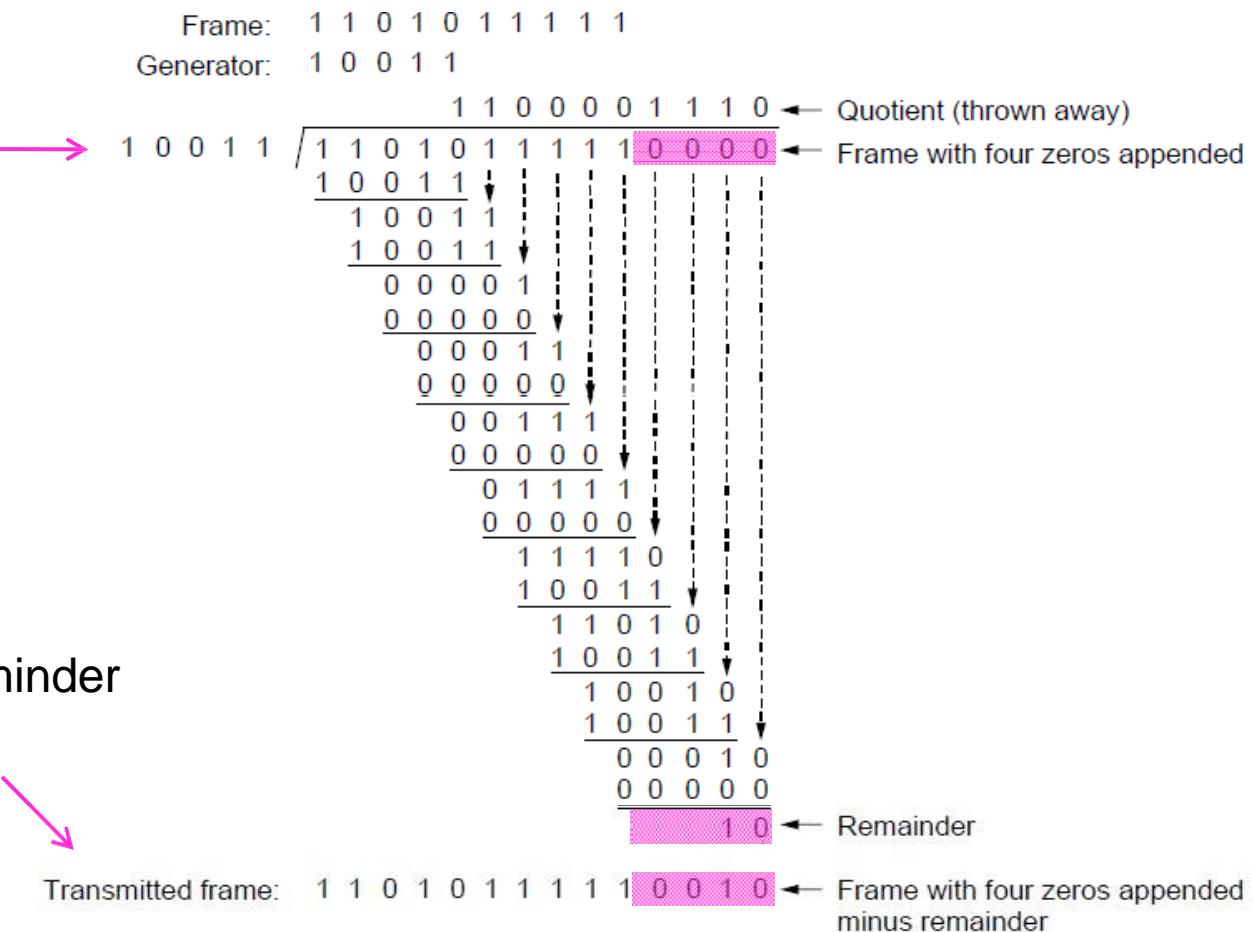
Properties:

- Improved error detection over parity bits
- Detects bursts up to N errors
- Detects random errors with probability $1-2^{-N}$
- Vulnerable to systematic errors, e.g., added zeros

Error Detection – CRCs (1) Cyclic Redundancy Checks

Adds bits so that transmitted frame viewed as a polynomial is evenly divisible by a generator polynomial

Start by adding
0s to frame
and try dividing



Error Detection – CRCs (2)

Based on standard polynomials:

- Ex: Ethernet 32-bit CRC is defined by:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$$

- Computed with simple shift/XOR circuits

Stronger detection than checksums:

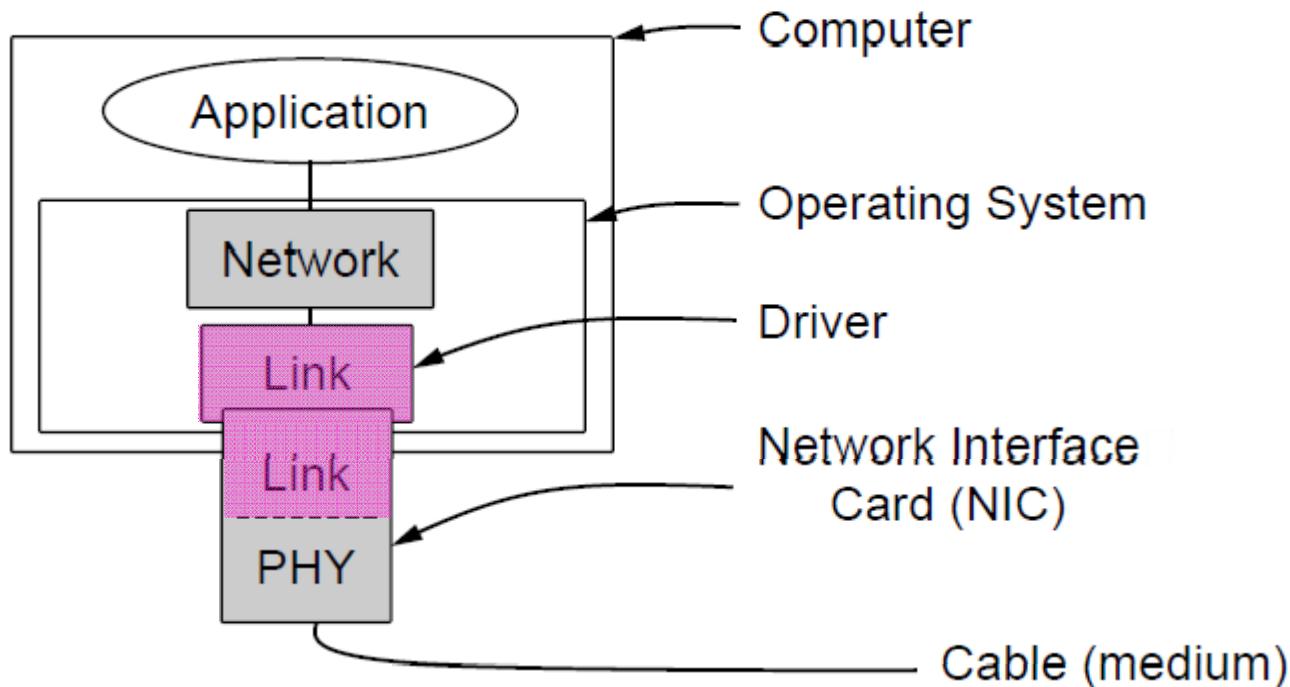
- E.g., can detect all double bit errors
- Not vulnerable to systematic errors

Elementary Data Link Protocols

- Link layer environment »
- Utopian Simplex Protocol »
- Stop-and-Wait Protocol for Error-free channel »
- Stop-and-Wait Protocol for Noisy channel »

Link layer environment (1)

Commonly implemented as NICs (Network Interface Cards) and OS drivers; network layer (IP) is often OS software



Link layer environment (2)

Link layer protocol implementations use library functions

- See code (`protocol.h`) for more details

| Group | Library Function | Description |
|-----------------|---|--|
| Network layer | <code>from_network_layer(&packet)</code> <code>to_network_layer(&packet)</code> <code>enable_network_layer()</code> <code>disable_network_layer()</code> | Take a packet from network layer to send Deliver a received packet to network layer Let network cause “ready” events Prevent network “ready” events |
| Physical layer | <code>from_physical_layer(&frame)</code> <code>to_physical_layer(&frame)</code> | Get an incoming frame from physical layer Pass an outgoing frame to physical layer |
| Events & timers | <code>wait_for_event(&event)</code> <code>start_timer(seq_nr)</code> <code>stop_timer(seq_nr)</code> <code>start_ack_timer()</code> <code>stop_ack_timer()</code> | Wait for a packet / frame / timer event Start a countdown timer running Stop a countdown timer from running Start the ACK countdown timer Stop the ACK countdown timer |

Utopian Simplex Protocol

An optimistic protocol (p1) to get us started

- Assumes no errors, and receiver as fast as sender
- Considers one-way data transfer

```
void sender1(void)
{
    frame s;
    packet buffer;

    while (true) {
        from_network_layer(&buffer);
        s.info = buffer;
        to_physical_layer(&s);
    }
}
```

Sender loops blasting frames

```
void receiver1(void)
{
    frame r;
    event_type event;

    while (true) {
        wait_for_event(&event);
        from_physical_layer(&r);
        to_network_layer(&r.info);
    }
}
```

Receiver loops eating frames

- That's it, no error or flow control ...

Stop-and-Wait – Error-free channel

Protocol (p2) ensures sender can't outpace receiver:

- Receiver returns a dummy frame (ack) when ready
- Only one frame out at a time – called stop-and-wait
- We added flow control!

```
void sender2(void)
{
    frame s;
    packet buffer;
    event_type event;

    while (true) {
        from_network_layer(&buffer);
        s.info = buffer;
        to_physical_layer(&s);
        wait_for_event(&event);
    }
}
```

Sender waits to for ack after
passing frame to physical layer

```
void receiver2(void)
{
    frame r, s;
    event_type event;
    while (true) {
        wait_for_event(&event);
        from_physical_layer(&r);
        to_network_layer(&r.info);
        to_physical_layer(&s);
    }
}
```

Receiver sends ack after passing
frame to network layer

Stop-and-Wait – Noisy channel (1)

ARQ (Automatic Repeat reQuest) adds error control

- Receiver acks frames that are correctly delivered
- Sender sets timer and resends frame if no ack.

For correctness, frames and acks must be numbered

- Else receiver can't tell retransmission (due to lost ack or early timer) from new frame
- For stop-and-wait, 2 numbers (1 bit) are sufficient

Stop-and-Wait – Noisy channel (2)

Sender loop (p3):

Send frame (or retransmission)

Set timer for retransmission

Wait for ack or timeout

If a good ack then set up for the next frame to send (else the old frame will be retransmitted)

```
void sender3(void) {  
    seq_nr next_frame_to_send;  
    frame s;  
    packet buffer;  
    event_type event;  
  
    next_frame_to_send = 0;  
    from_network_layer(&buffer);  
    while (true) {  
        s.info = buffer;  
        s.seq = next_frame_to_send;  
        → to_physical_layer(&s);  
        → start_timer(s.seq);  
        → wait_for_event(&event);  
        if (event == frame_arrival) {  
            from_physical_layer(&s);  
            if (s.ack == next_frame_to_send) {  
                stop_timer(s.ack);  
                from_network_layer(&buffer);  
                inc(next_frame_to_send);  
            }  
        }  
    }  
}
```

Stop-and-Wait – Noisy channel (3)

Receiver loop (p3):

Wait for a frame

If it's new then take
it and advance
expected frame

Ack current frame

```
void receiver3(void)
{
    seq_nr frame_expected;
    frame r, s;
    event_type event;

    frame_expected = 0;
    while (true) {
        wait_for_event(&event);
        if (event == frame_arrival) {
            from_physical_layer(&r);
            if (r.seq == frame_expected) {
                to_network_layer(&r.info);
                inc(frame_expected);
            }
            s.ack = 1 - frame_expected;
            to_physical_layer(&s);
        }
    }
}
```

Sliding Window Protocols

- Sliding Window concept »
- One-bit Sliding Window »
- Go-Back-N »
- Selective Repeat »

Sliding Window Protocols

- In the previous protocols, data frames are transmitted in one direction only
- In most practical situations, data frames have to be transmitted in both directions.
- Each link has a forward channel and reverse channel. The capacity of reverse channel is wasted.
- Alternative
 - Use the same link in both directions.
 - Data frames of A and B are intermixed with ack frames of A and B.
- Piggybacking
 - Temporarily delaying ack so that they can be hooked into the next outgoing frame.
- However, piggybacking increases the delay.
 - Use hybrid
 - If possible use piggybacking otherwise send a separate ack frame.

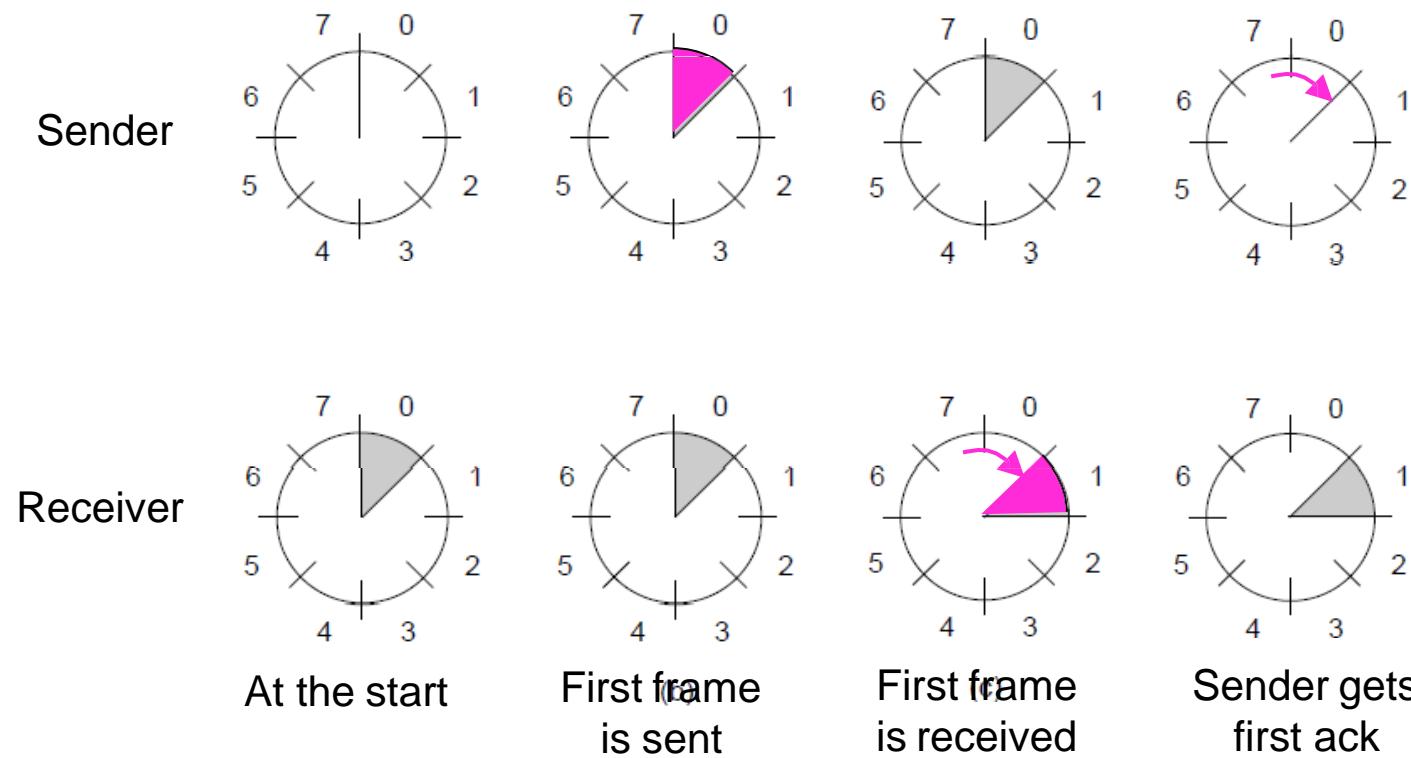
Sliding Window Protocols

- Summary of sliding window protocols
 - Sender maintains a sequence numbers for frames (sending window)
 - Receiver also maintains a receiving window.
 - Whenever a packet is received from network layer, it gives a highest sequence number and the upper edge of the window is advanced by one.
 - When ack, arrives lower edge of the window is advanced by one.
 - Sender needs n buffers to hold unack frames

Sliding Window concept (2)

A sliding window advancing at the sender and receiver

- Ex: window size is 1, with a 3-bit sequence number.



Sliding Window concept (3)

Larger windows enable pipelining for efficient link use

- Stop-and-wait ($w=1$) is inefficient for long links
- Best window (w) depends on bandwidth-delay (BD)
- Want $w \geq 2BD+1$ to ensure high link utilization

Pipelining leads to different choices for errors/buffering

- We will consider Go-Back-N and Selective Repeat

One-Bit Sliding Window (1)

Transfers data in both directions with stop-and-wait

- Piggybacks acks on reverse data frames for efficiency
- Handles transmission errors, flow control, early timers

Each node is sender
and receiver (p4):

Prepare first frame

Launch it, and set timer

```
void protocol4 (void) {  
    seq_nr next_frame_to_send;  
    seq_nr frame_expected;  
    frame r, s;  
    packet buffer;  
    event_type event;  
  
    next_frame_to_send = 0;  
    frame_expected = 0;  
    from_network_layer(&buffer);  
    s.info = buffer;  
    s.seq = next_frame_to_send;  
    s.ack = 1 - frame_expected;  
    to_physical_layer(&s);  
    start_timer(s.seq);  
    . . .
```

One-Bit Sliding Window (2)

Wait for frame or timeout

If a frame with new data
then deliver it

If an ack for last send then
prepare for next data frame

(Otherwise it was a timeout)

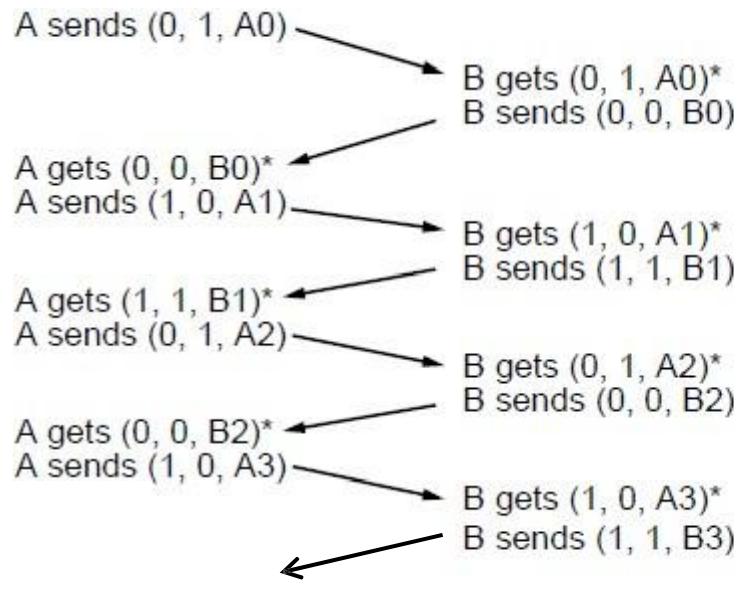
Send next data frame or
retransmit old one; ack
the last data we received

```
    . . .
while (true) {
    → wait_for_event(&event);
    if (event == frame_arrival) {
        from_physical_layer(&r);
        if (r.seq == frame_expected) {
            to_network_layer(&r.info);
            inc(frame_expected);
        }
        if (r.ack == next_frame_to_send) {
            stop_timer(r.ack);
            from_network_layer(&buffer);
            inc(next_frame_to_send);
        }
        s.info = buffer;
        s.seq = next_frame_to_send;
        s.ack = 1 - frame_expected;
        → to_physical_layer(&s);
        start_timer(s.seq);
    }
}
```

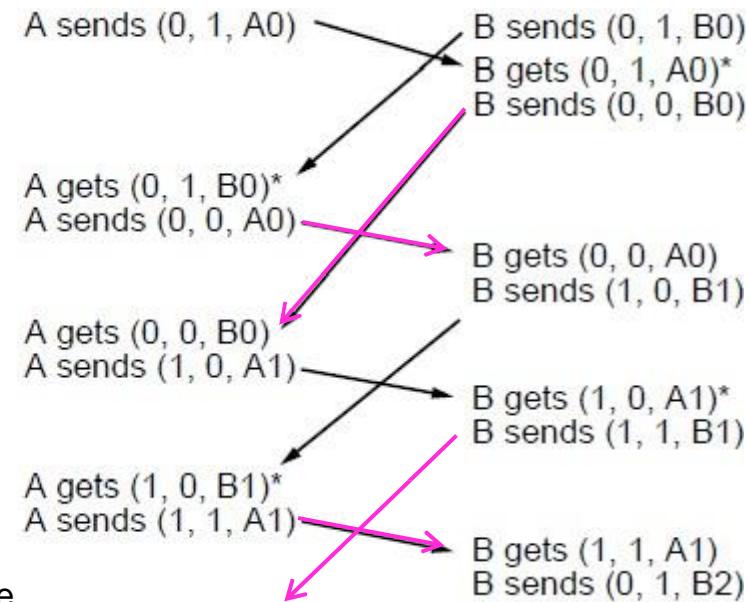
One-Bit Sliding Window (3)

Two scenarios show subtle interactions exist in p4:

- Simultaneous start [right] causes correct but slow operation compared to normal [left] due to duplicate transmissions.



Normal case



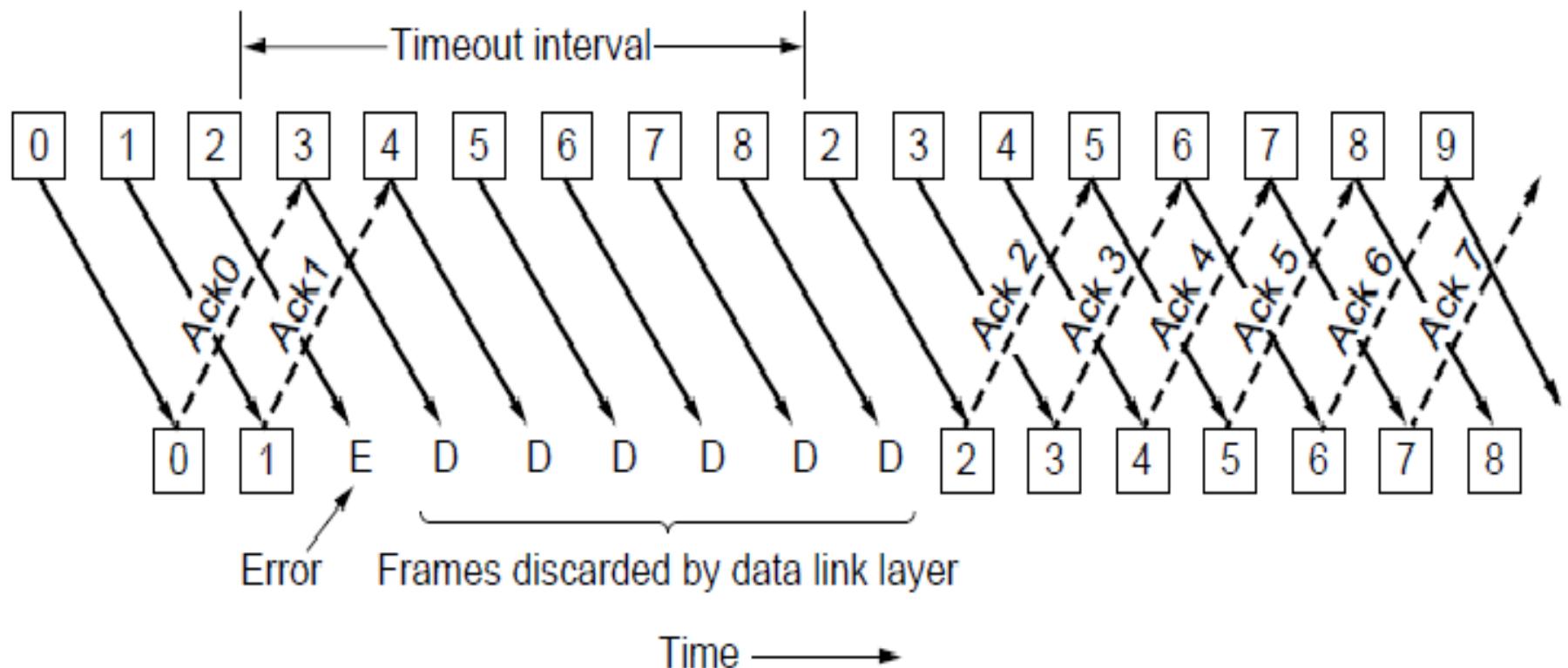
Correct, but poor performance

Notation is (seq, ack, frame number). Asterisk indicates frame accepted by network layer .

Go-Back-N Sliding window protocol

Receiver only accepts/acks frames that arrive in order:

- Discards frames that follow a missing/errored frame
- Sender times out and resends all outstanding frames



Go-Back-N Protocol

Tradeoff made for Go-Back-N:

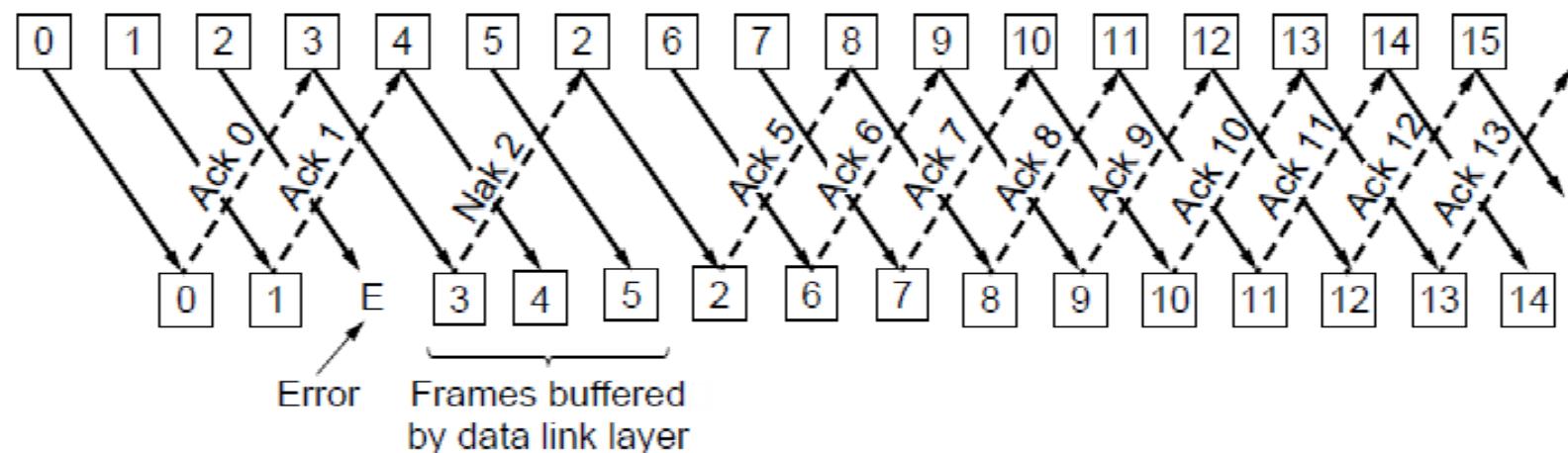
- Simple strategy for receiver; needs only 1 frame
- Wastes link bandwidth for errors with large windows; entire window is retransmitted

Implemented as p5 (see code in book)

Selective Repeat Sliding Window Protocol

Receiver accepts frames anywhere in receive window

- Cumulative ack indicates highest in-order frame
- NAK (negative ack) causes sender retransmission of a missing frame before a timeout resends window



About Selective Repeat

Tradeoff made for Selective Repeat:

- More complex than Go-Back-N due to buffering at receiver and multiple timers at sender
- More efficient use of link bandwidth as only lost frames are resent (with low error rates)

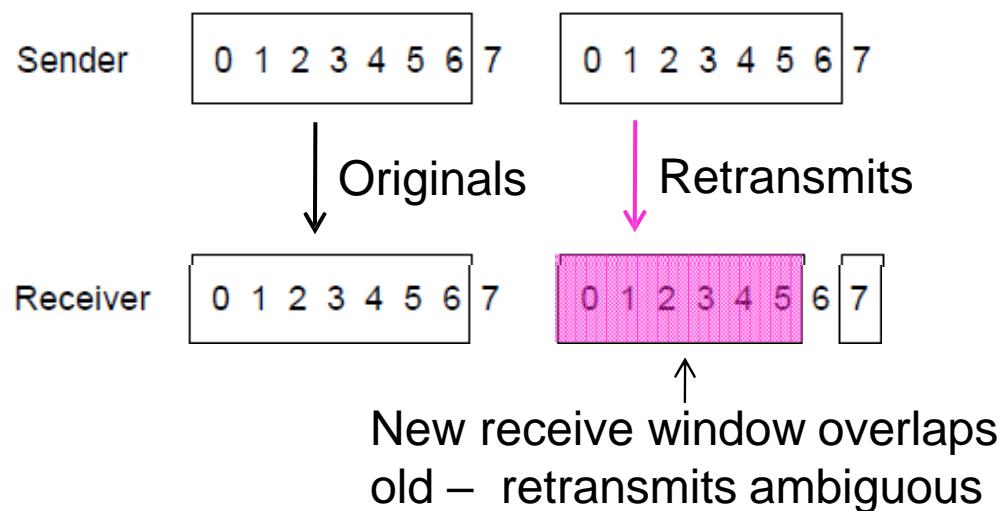
Implemented as p6 (see code in book)

Selective Repeat

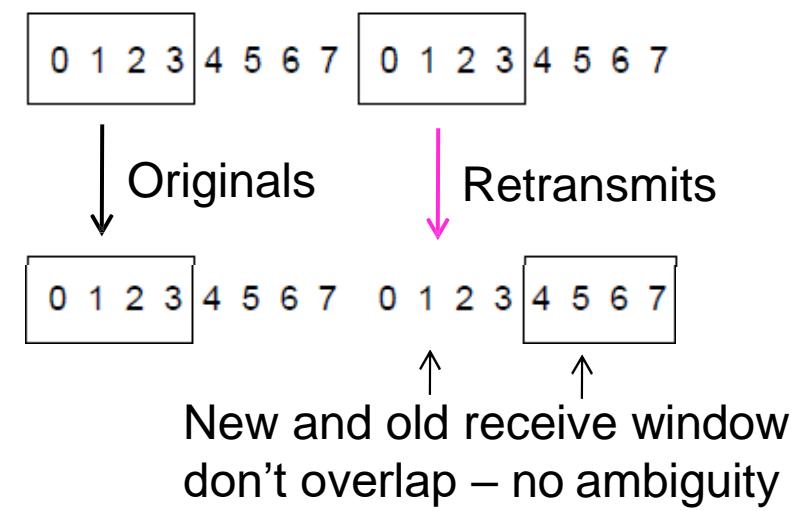
For correctness, we require:

- Sequence numbers (s) at least twice the window (w)

Error case ($s=8$, $w=7$) – too few sequence numbers



Correct ($s=8$, $w=4$) – enough sequence numbers



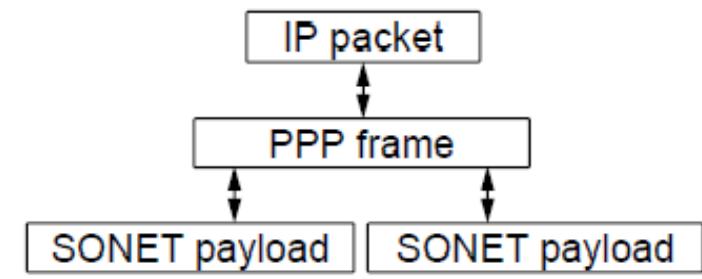
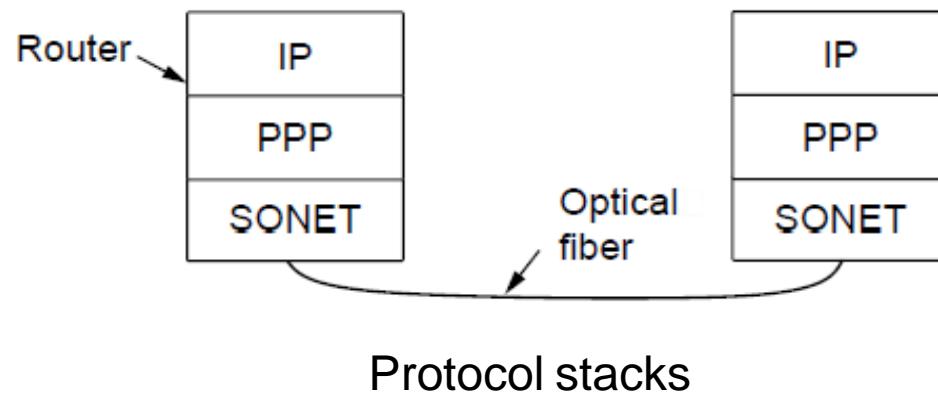
Example Data Link Protocols

- Packet over SONET (Synchronous Optical Network (SONET))
- PPP (Point-to-Point Protocol) »
- ADSL (Asymmetric Digital Subscriber Loop) »

Packet over SONET

Packet over SONET is the method used to carry IP packets over SONET optical fiber links

- Uses PPP (Point-to-Point Protocol) for framing

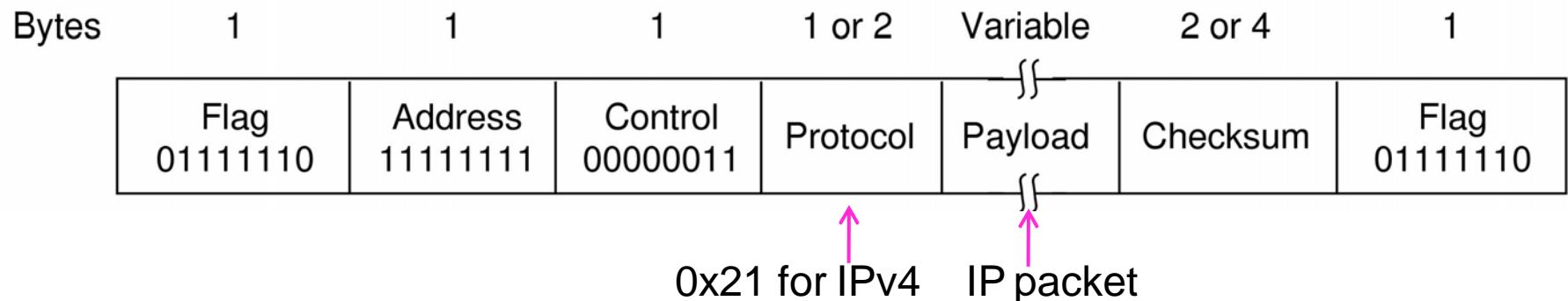


PPP frames may be split
over SONET payloads

PPP (1)

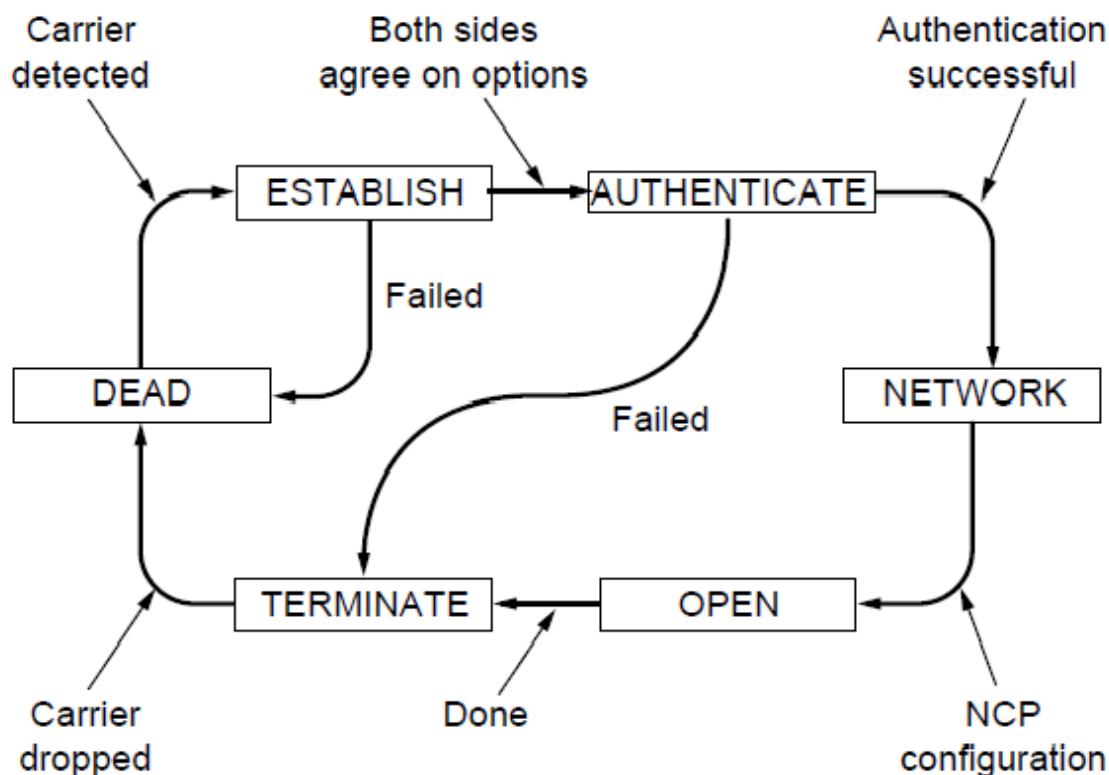
PPP (Point-to-Point Protocol) is a general method for delivering packets across links

- Framing uses a flag (0x7E) and byte stuffing
- “Unnumbered mode” (connectionless unacknowledged service) is used to carry IP packets
- Errors are detected with a checksum



PPP (2)

A link control protocol brings the PPP link up/down

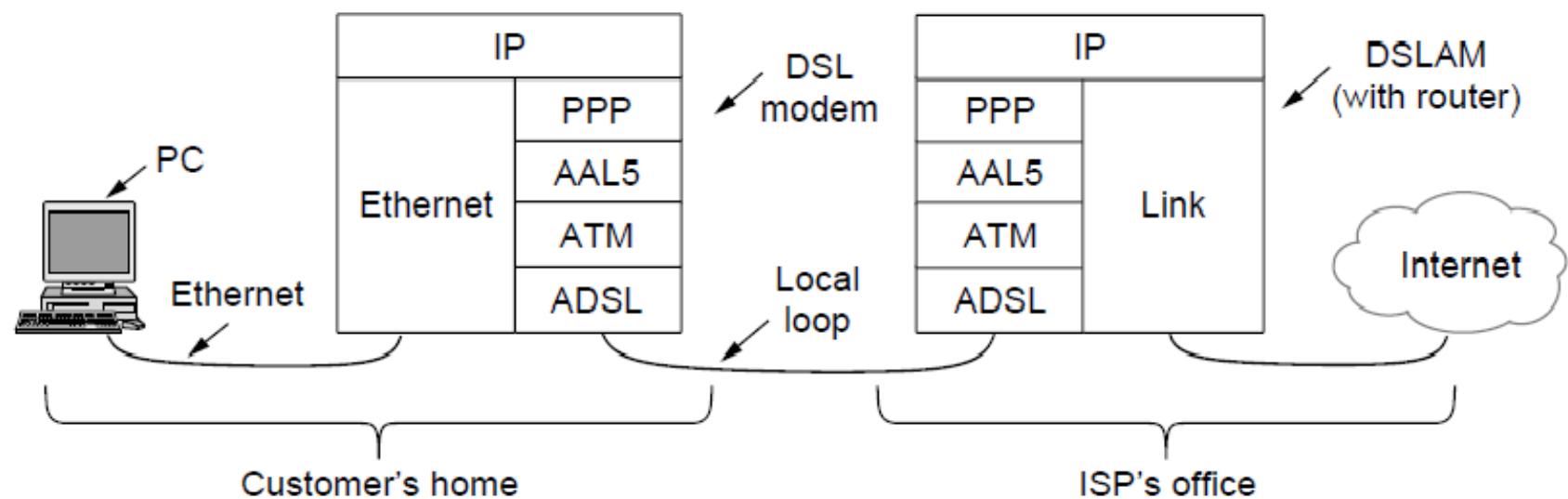


State machine for link control

ADSL (1)

Widely used for broadband Internet over local loops

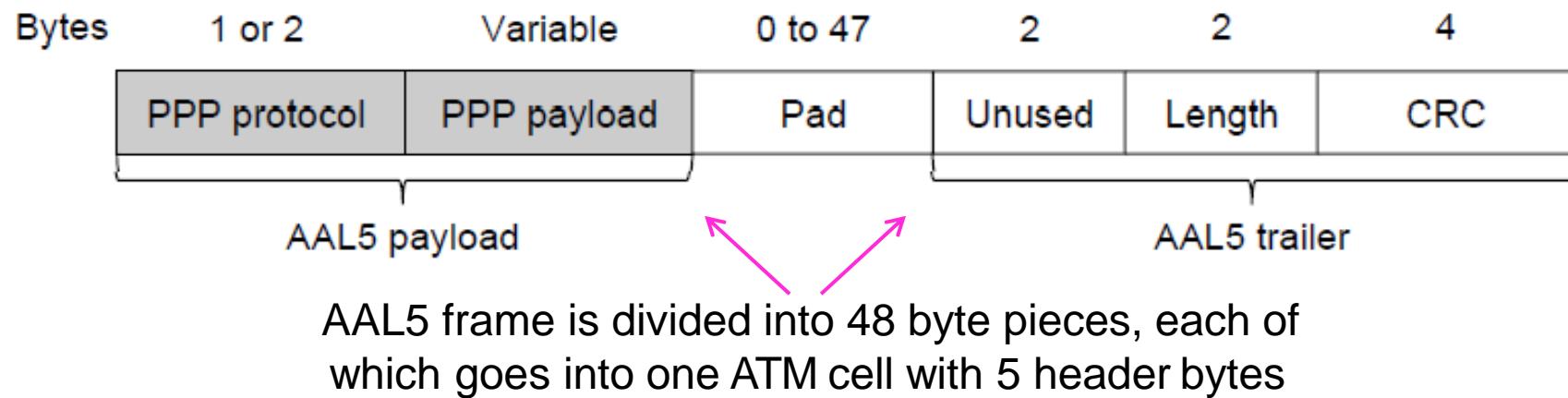
- ADSL runs from modem (customer) to DSLAM (ISP)
- IP packets are sent over PPP and AAL5/ATM (over)



ADSL (2)

PPP data is sent in AAL5 frames over ATM cells:

- ATM is a link layer that uses short, fixed-size cells (53 bytes); each cell has a virtual circuit identifier
- AAL5 is a format to send packets over ATM
- PPP frame is converted to a AAL5 frame (PPPoA)



End

Chapter 3

The Medium Access Control Sublayer

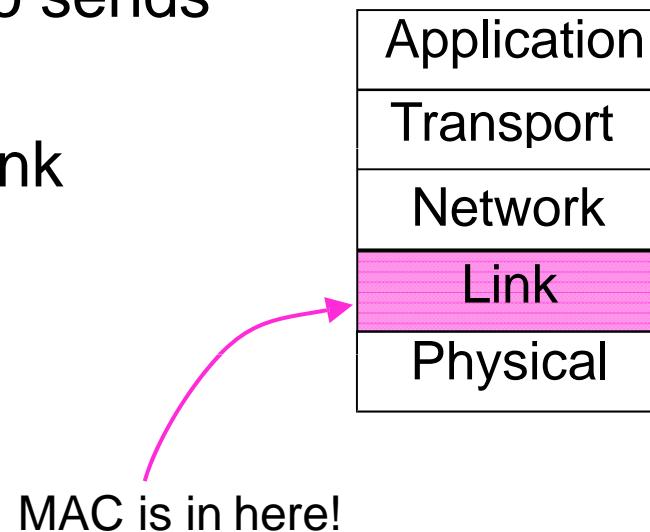
Chapter 4

- Networks links can be divided into two categories:
 - Point-to-point connections
 - WAN is point-to-point links
 - Broadcast channels/multiaccess channels or random access channels
 - For fixed channel and traffic from N users
 - Wireless is a broadcast channel
 - Analogy
 - Conference call with six people with different telephones
- Medium access control sublayer is a sublayer in data link layer
- Multi-access channels and LANs are closely related
 - So, we discuss about LANs
- MAC sublayer is the bottom of the datalink layer

The MAC Sublayer

Responsible for deciding who sends next on a multi-access link

- An important part of the link layer, especially for LANs



Medium Access Control Sublayer

- Channel Allocation Problem
- Multiple Access Protocols
- Ethernet
- Wireless LANs
- Broadband Wireless
- Bluetooth
- RFID
- Data Link Layer Switching

Channel Allocation Problem

- Channel connects one user to other user/users
 - Portion of wireless spectrum, single wire, optical fiber
- Two options:
 - Static Channel allocation
 - Dynamic channel allocation
- Static allocation
 - For fixed channel and traffic from N users
 - Divide up bandwidth using FTM, TDM, CDMA, etc.
 - This is a static allocation, e.g., FM radio
 - The static allocation performs poorly for bursty traffic
 - Allocation to a user will sometimes go unused

Static Allocation: Performance of FDM

- FDM
 - If there are N users, the bandwidth is divided into N equal-sized portions, with each user being assigned one portion.
 - Issues
 - When number of senders increase, FDM suffers
- If the spectrum is cut up into N regions and fewer than N users are communicating, a large portion is wasted.
- Analysis: single channel
 - Channel capacity= C bps;
 - Mean time delay= T ;
 - frame arrival rate = λ frames/sec,
 - the frames vary with the average length of= $1/\mu$ bits.
 - With this the service rate of the channel is μC frames/sec.
 - As per the standard queuing theory $T = 1/(\mu C - \lambda)$
 - https://en.wikipedia.org/wiki/Queueing_theory
- Allocation for dividing the channel into N channels
 - If we divide the channel into N channels, each can transmit with the capacity of C/N bps; the mean arrival rate is λ/N frames/sec. Then,
 - $T_N = 1/(\mu(C/N) - (\lambda/N)) = N/(\mu C - \lambda) = NT$
- Same arguments will apply for other modes of dividing the channel

Dynamic Channel Allocation

Dynamic allocation gives the channel to a user when they need it. Potentially N times as efficient for N users.

Schemes vary with assumptions:

| Assumption | Description | Implication |
|----------------------------|---|--|
| Independent traffic | The expected number of frames per unit time is constant. After generation the station is blocked till the frame is delivered. | Often not a good model, but permits analysis |
| Single channel | A single channel is available for all stations. All stations can send and receive | No external way to coordinate senders |
| Observable collisions | If two frames are transmitted simultaneously, the signal is garbled. All stations can detect. | Needed for reliability; mechanisms vary |
| Continuous or slotted time | Time may be continuous or slotted. Frame transmissions can begin at the beginning of the slot. | Slotting may improve performance |
| Carrier sense | Stations can tell if the channel is in use | Can improve performance if available |

Multiple Access Protocols

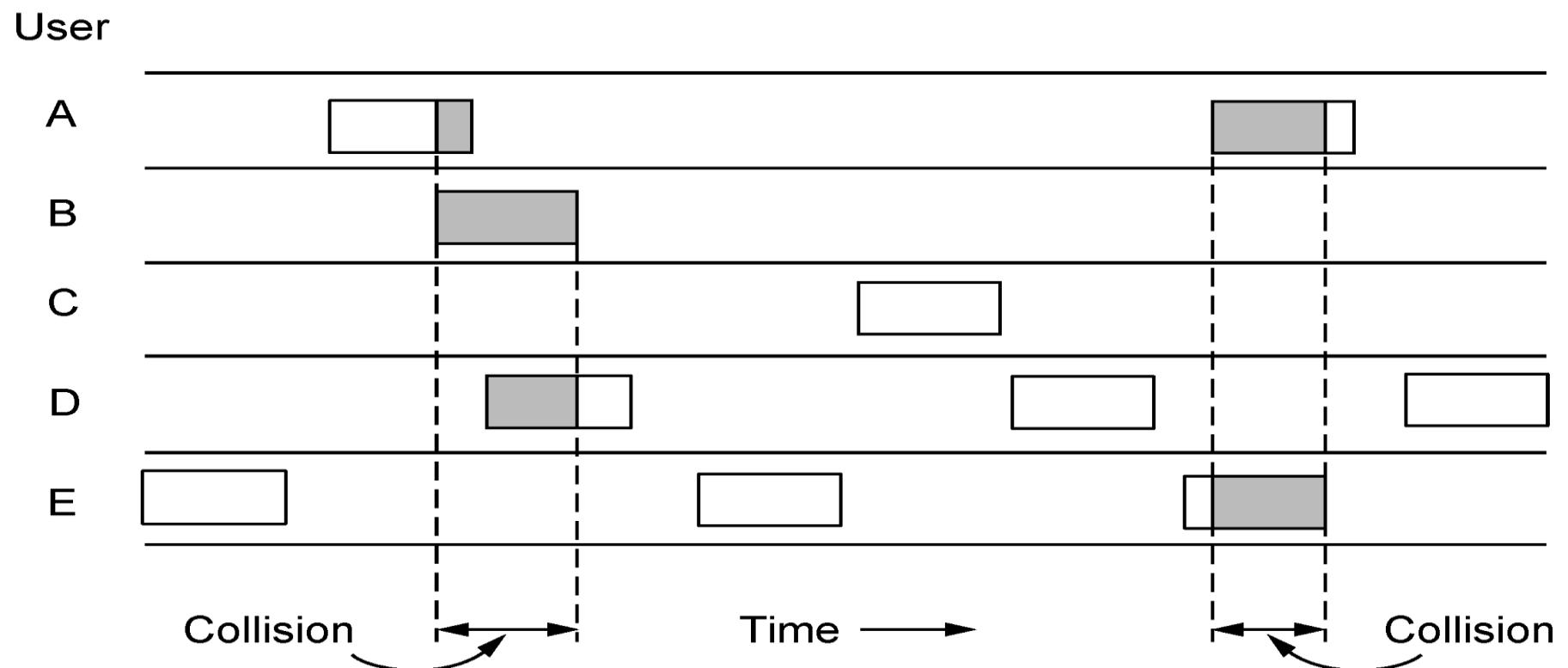
- ALOHA »
- CSMA (Carrier Sense Multiple Access) »
- Collision-free protocols »
- Limited-contention protocols ^{1/(μC · λ)} »
- Wireless LAN protocols »

PURE ALOHA

- **ALOHAnet**, also known as the **ALOHA System**, or simply **ALOHA**, was a pioneering computer networking system developed at the University of Hawaii.
- ALOHAnet became operational in June, 1971, providing the first public demonstration of a wireless packet data network.
- ALOHA originally stood for “Additive Links On-line Hawaii Area”

PURE ALOHA

- Pure ALOHA
 - Users transmit frames whenever they have data;
 - If the frame is destroyed (Collision occurs), users waits a random amount of time and sends it again.



In pure ALOHA, frames are transmitted at completely arbitrary times.

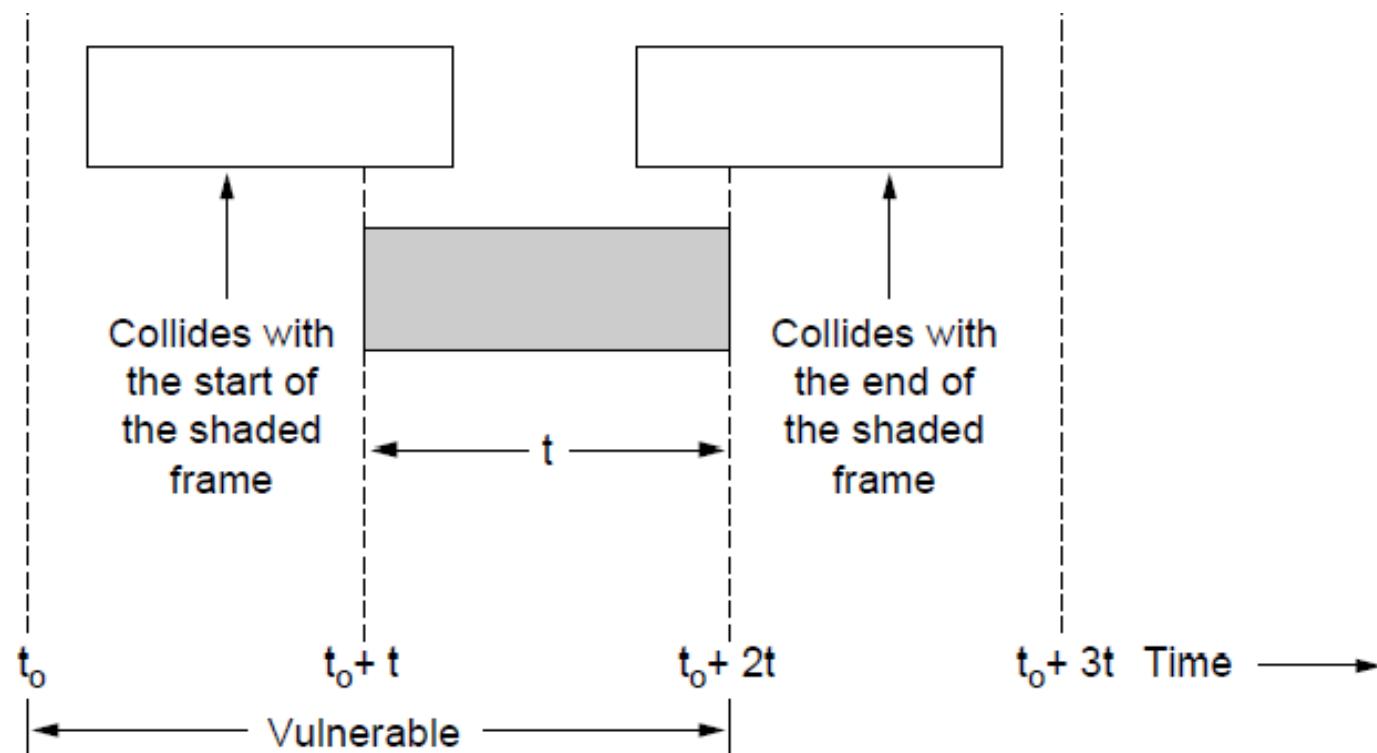
Efficiency of Pure ALOHA

- What fraction of frames escape collisions in chaotic situation?
 - Consider N users typing at the terminals, when the line is finished, user waits for the response.
- Let “frame time” denote the time to transmit the frame.
- Frame generation is modelled with a Poisson distribution with a mean of N frames per frame time (https://en.wikipedia.org/wiki/Poisson_distribution)
- If $N > 1$, user is generating frames higher rate than the channel can handle.
- For reasonable throughput we expect that $0 < N < 1$.
- Station also generated retransmissions
- Old and new frames are modeled by a Poisson distribution with a mean of G frames per frame time.
- Clearly, $G \geq N$, So, at high load there will be many collisions.
- Throughput= $G * P_0$, where P_0 is the probability that a frame does not suffer a collision.
- Question: Under what conditions, the frame arrive undamaged?
- The probability that k frames are generated during the given frame time in which G frames are expected is given by Poisson distribution
 - $\Pr[k] = \frac{G^k e^{-G}}{k!}$
 - The probability of 0 frames is e^{-G}
- The probability of no frames being initiated during the entire vulnerable period is thus given by $P_0 = e^{-2G}$; Using $S = GP_0$, we get $S = G e^{-2G}$

ALOHA (2)

Collisions happen when other users transmit during a vulnerable period that is twice the frame time

- Synchronizing senders to slots can reduce collisions



Slotted ALOHA

- Divide the time into discrete intervals. Each interval corresponding to one frame.
 - Allow a special station to emit a pip at the start of each interval.
 - Station is not permitted to send at any time. It has to wait for the slot.

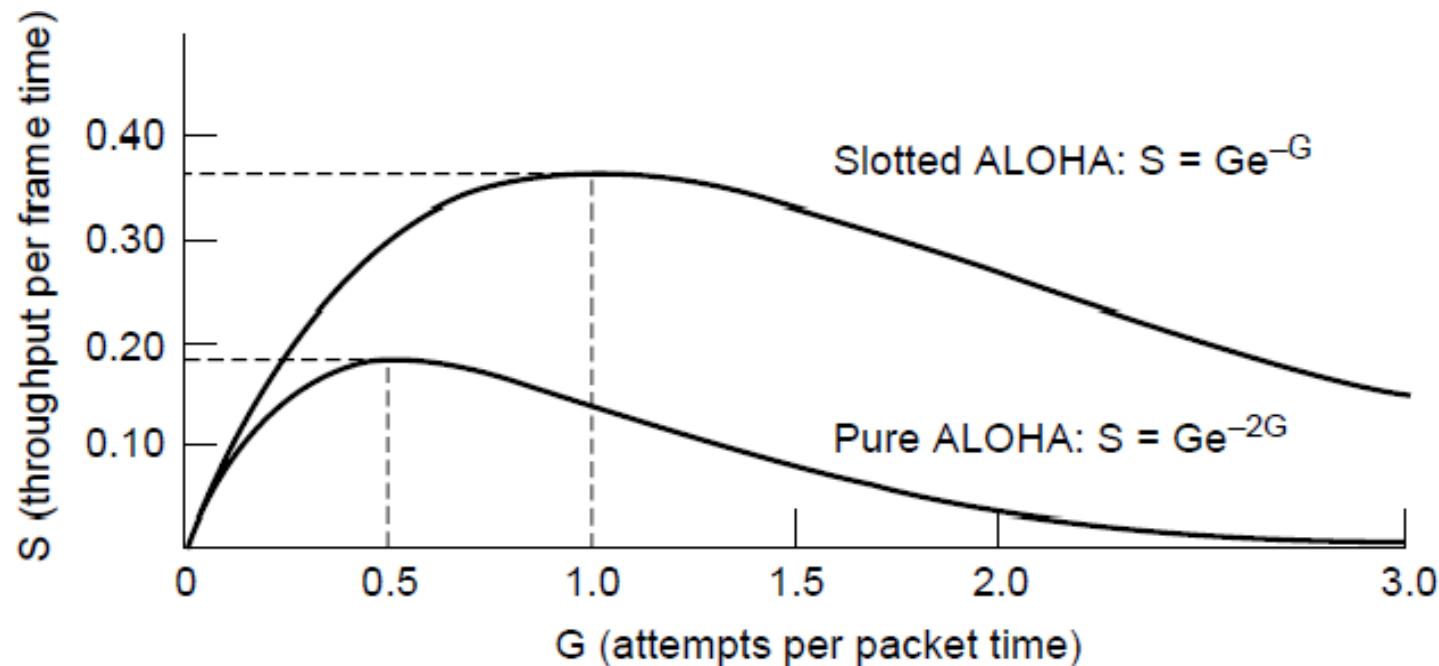
Efficiency of Slotted ALOHA

- In Slotted ALOHA, in contrast to Pure ALOHA, a station is not permitted to send whenever user types a line, it is required to wait for the beginning of the next slot. This halves the vulnerability period.
- The probability of no other traffic during the same slot as our test frame is there is then e^{-G} , which leads to $S=G e^{-G}$
- Slotted ALOHA peaks at $G=1$, the throughout is $1/e =$ about 0.368
- The probability of a transmission requiring exactly k attempts (i.e., $k-1$ collisions and one success) $P_k = e^{-G} (1 - e^{-G})^{k-1}$
- The expected number of retransmissions, E , per line types at a terminal is them
$$E = \sum_{k=1}^{\infty} k P_k = \sum_{k=1}^{\infty} k e^{-G} (1 - e^{-G})^{k-1} = e^G$$
- A small increase in the channel load can drastically reduce the performance.

ALOHA

Slotted ALOHA is twice as efficient as pure ALOHA

- Low load wastes slots, high loads causes collisions
- Efficiency up to $1/e$ (37%) for random traffic models



Carrier Sense Multiple Access (CSMA)

- CSMA (Carrier Sense Multiple Access) improves on ALOHA by sensing the channel!
- User doesn't send if it senses someone else

Variations on what to do if the channel is busy:

- 1-persistent (greedy) sends as soon as idle
- Nonpersistent waits a random time then tries again
- p-persistent sends with probability p when idle

1-Persistent CSMA Protocol

- When a station has data to send, it listens to the channel
- If the channel is idle, the station sends the data
- If the channel is busy, the station waits until it becomes idle.
- If the collision occurs, the station waits for random amount of time and starts over again.
- It is called 1-persistent, because, the station transmits with a probability of 1 when it finds the channel idle.
- Issues: when two stations ready, collisions occur, if they are not so impatient.

Nonpersistent CSMA Protocol

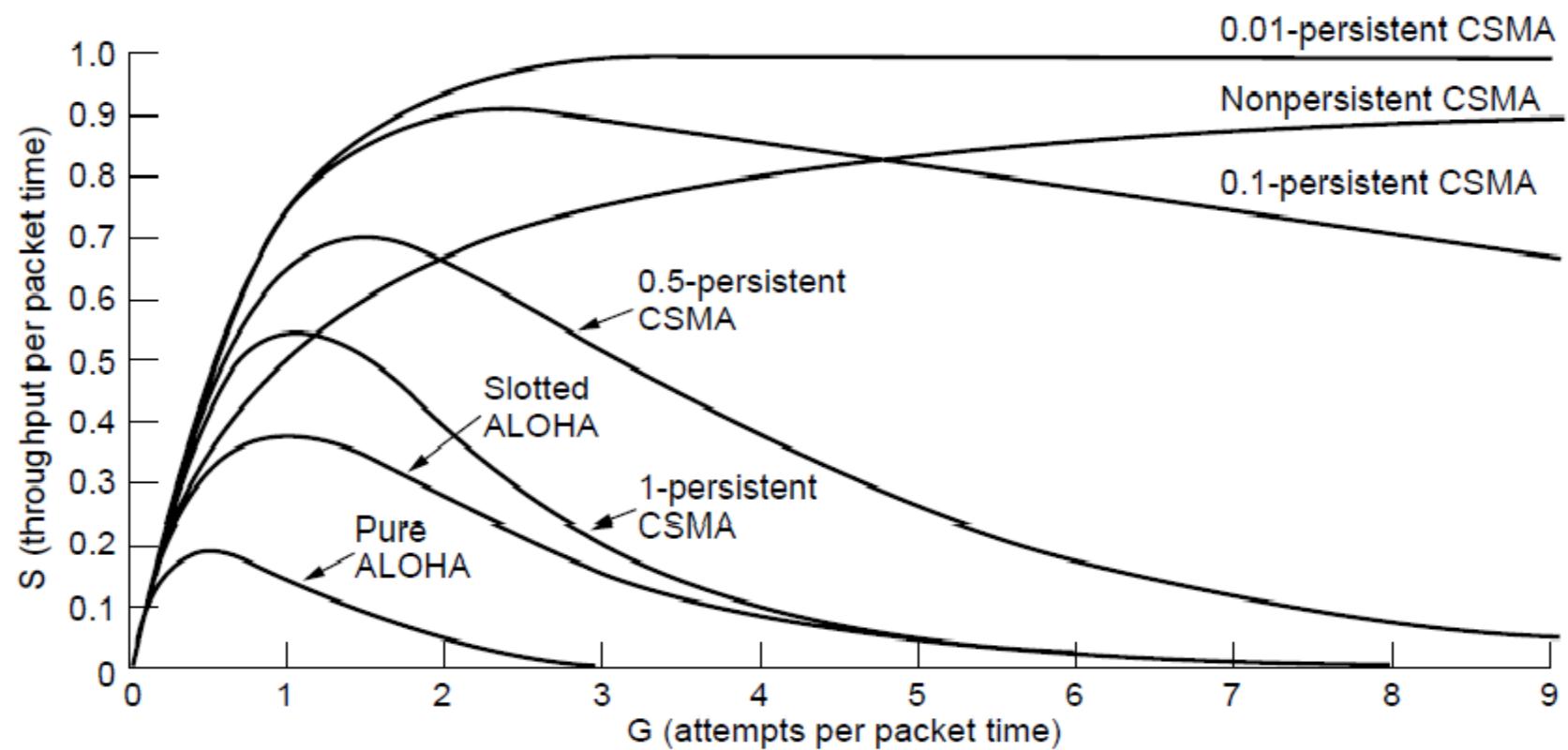
- Attempt has been made to be less greedy than 1-persistent
- When a station has data to send, it listens to the channel
- If the channel is idle, the station sends the data
- If the channel is busy,
 - the station does not make effort to seize it immediately after detecting the end of transmission. Instead, it waits for random period of time and repeats the algorithm
- Advantages: It has better utilization than 1-persistent CSMA

p-persistent CSMA Protocol

- Applies to slotted channels
- When a station has data to send, it listens to the channel
- If the channel is idle, the station sends the data with probability p . With a probability $q=1-p$, it defers until the next slot. If that slot is idle it either transmits or defers with probability p and q . This step repeats until either the frame has transmitted or other station has begun transmitting.
- If the slot is busy, it considers as a Collision, that it, it waits for random time and starts again.
- If the station initially senses that the slot is busy, it waits until the next slot and applies the above algorithm
- Advantages: Improved Utilization over nonpersistent CSMA

CSMA – Persistence

CSMA outperforms ALOHA, and being less persistent is better under high load



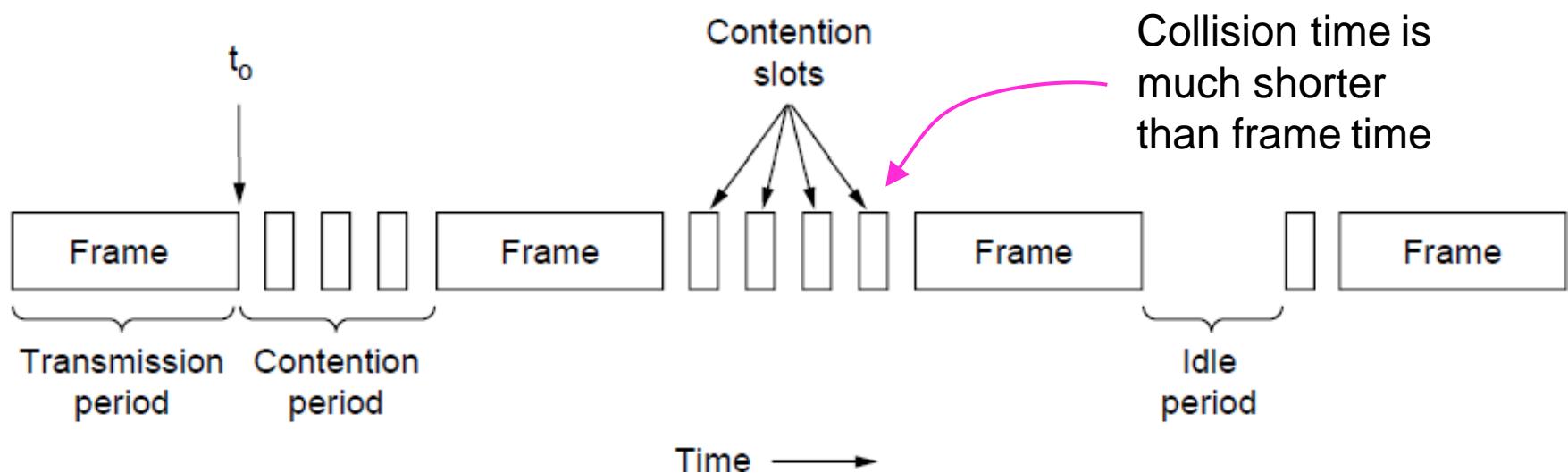
CSMA with Collision detection

- In CSMA, if two stations sense a channel to be idle and begin transmitting, the signals will collide.
- Alternative: Stations quickly detect collision and abruptly stop transmitting.
 - It saves time and bandwidth
- If a station detects collision, it aborts its transmission, waits a random period of time and tries again.
- CSMA/CD consists of alternating contention and transmission periods and idle periods (when all stations are quiet).
- If the channel transmits $2*t$, whether t is the signal propagation time between the two farthest stations. (for 1KM co-axial cable, $t=5 \mu\text{sec}$).
- Performance is greatly improved if frame time is much larger than the propagation time.

CSMA (3) – Collision Detection

CSMA/CD improvement is to detect/abort collisions

- Reduced contention times improve performance



Collision-Free Protocols

- In CSMA/CD, collisions still occur during the contention period.
- It will effect the performance with large t (cable is long)

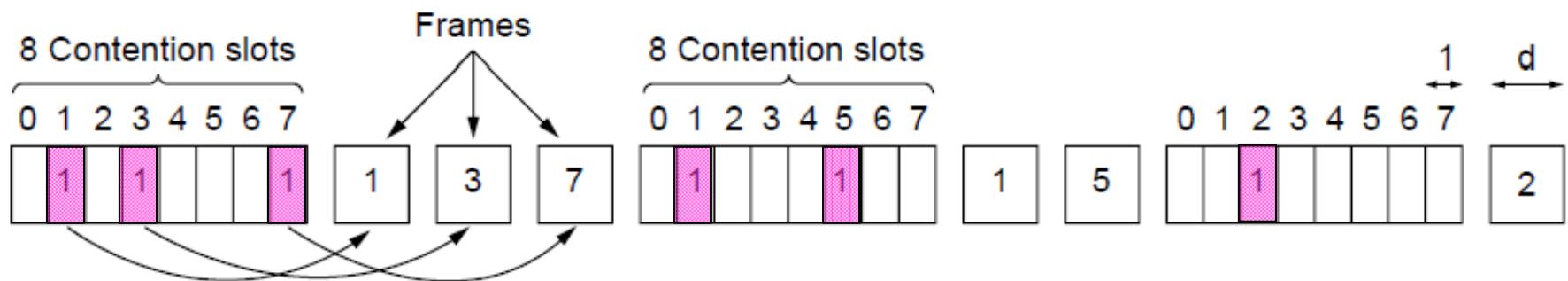
Bitmap Collision-free protocol

Collision-free protocols avoid collisions entirely

- Senders must know when it is their turn to send

The basic bit-map protocol:

- Contention period consists of N slots.
- Sender set a bit in contention slot if they have data
- Senders send in turn; everyone knows who has data
- Stations transmit data in the numerical order.



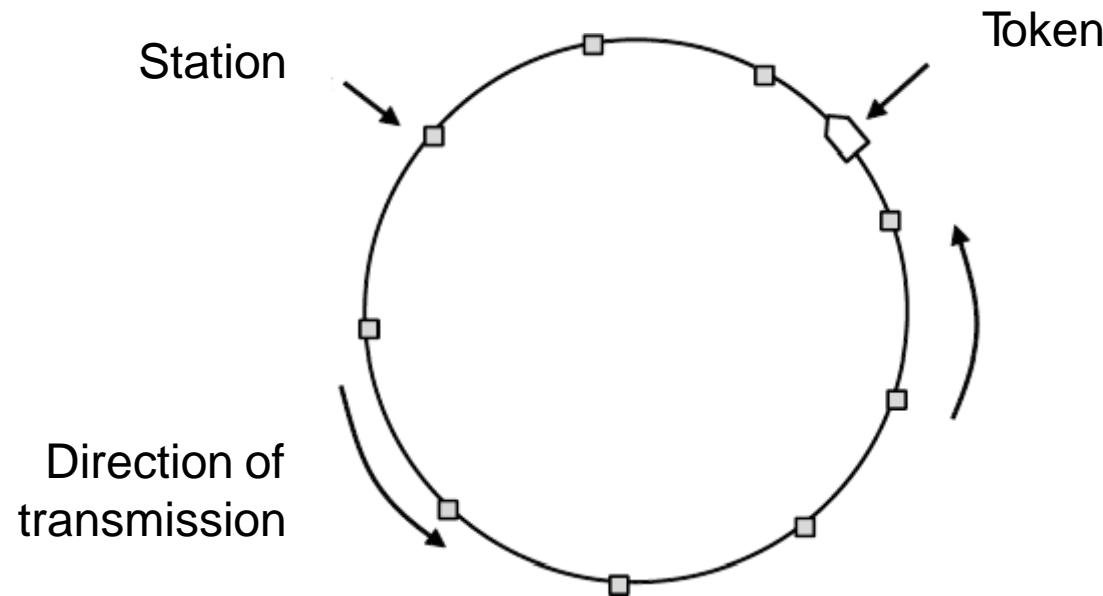
Performance:

- Low load: bit-map gets repeated. On an average, the station has to wait $N/2$ slots.
- High numbers slots rarely have to wait for next scan.
- Lower numbers channels have to wait for $1.5N$ slots and higher numbered channels have to wait for $0.5N$ slots.

Token Ring-Collision free protocol

Token sent round ring defines the sending order

- Station with token may send a frame before passing
- Idea can be used without ring too, e.g., token bus

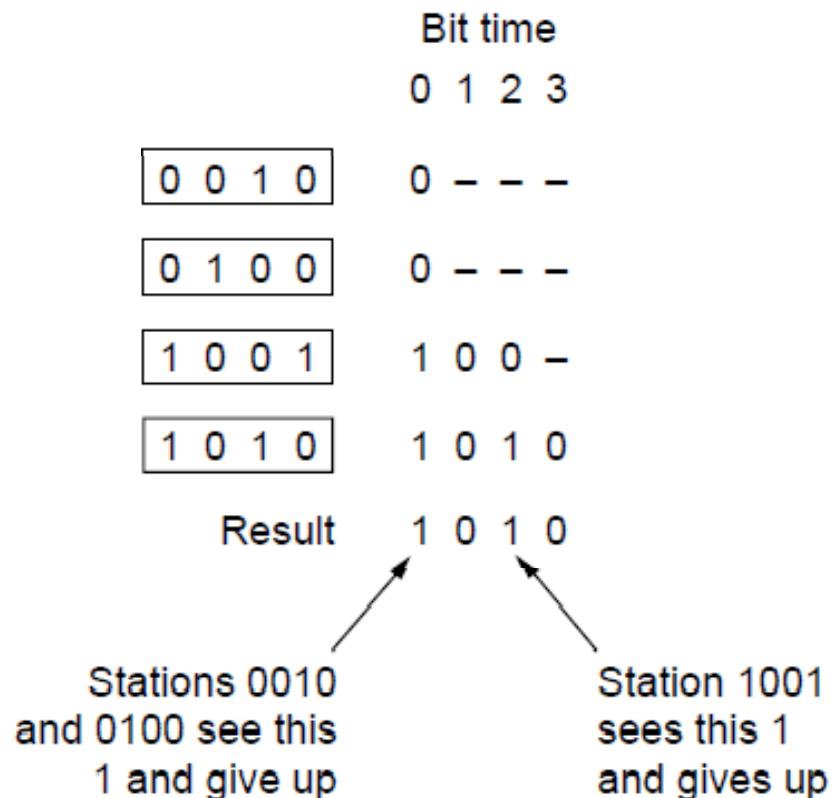


Countdown-Collision Free Protocol

-Bitmap does not scale to thousands of stations

-Binary countdown improves on the bitmap protocol

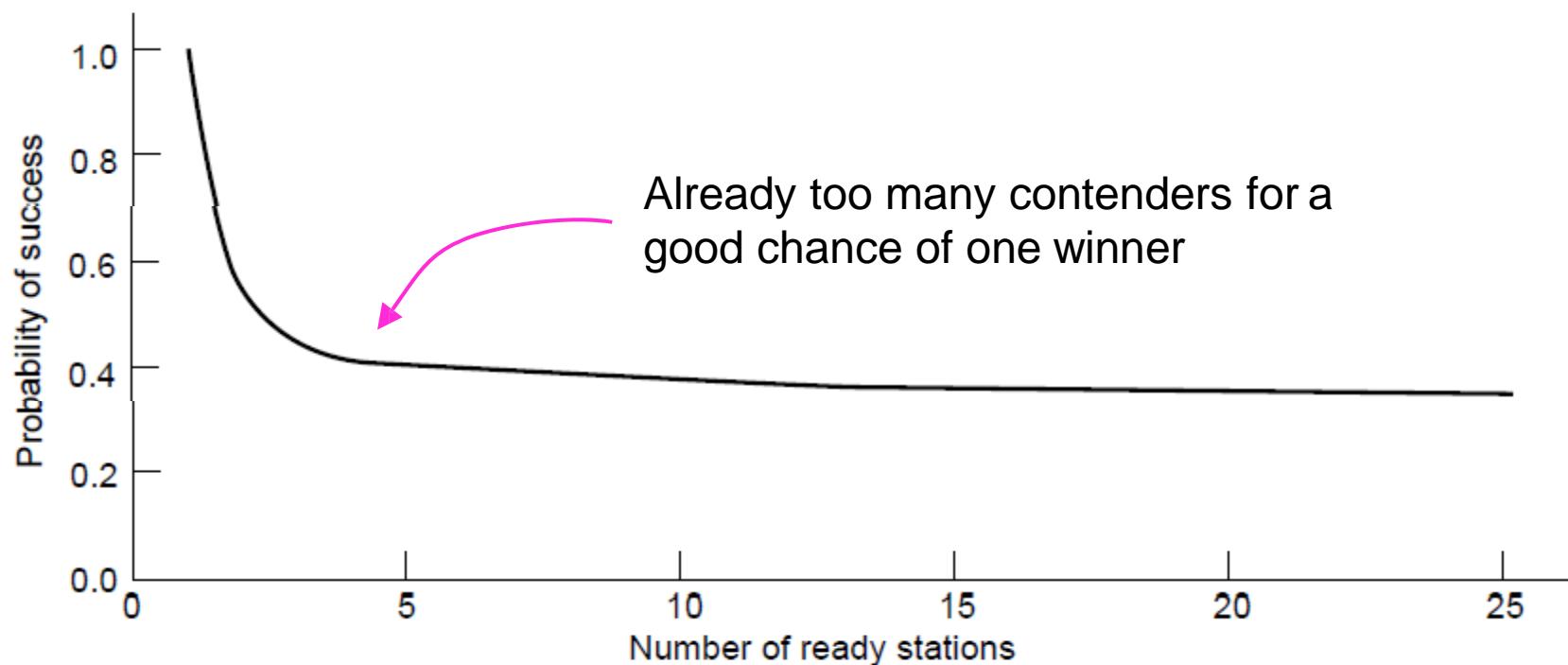
- Stations send their address in contention slot ($\log N$ bits instead of N bits)
- Medium ORs bits; stations give up when they send a “0” but see a “1”
- Station that sees its full address is next to send



Limited-Contention Protocols

Idea is to divide stations into groups within which only a very small number are likely to want to send

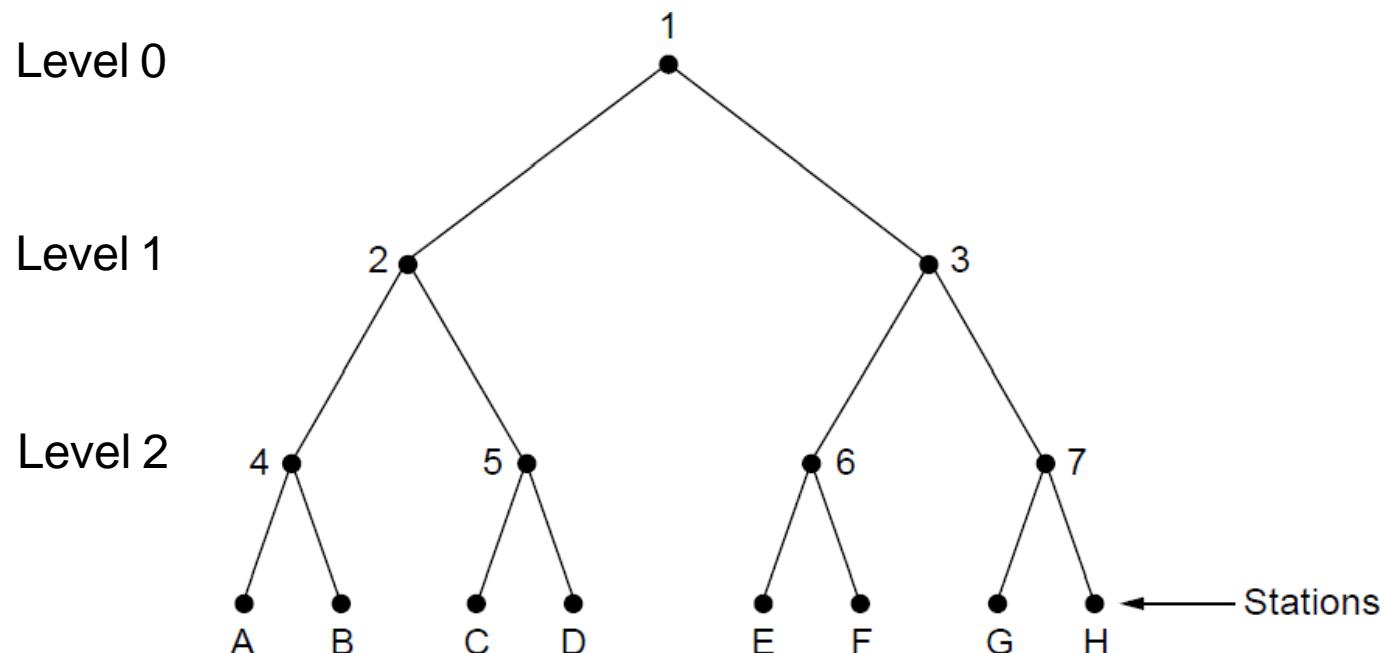
- Avoids wastage due to idle periods and collisions



Limited Contention (2) –Adaptive Tree Walk

Tree divides stations into groups (nodes) to poll

- Depth first search under nodes with poll collisions
- Start search at lower levels if >1 station expected



Wireless LAN Protocols (1)

Wireless has complications compared to wired.

Nodes may have different coverage regions

- Leads to hidden and exposed terminals

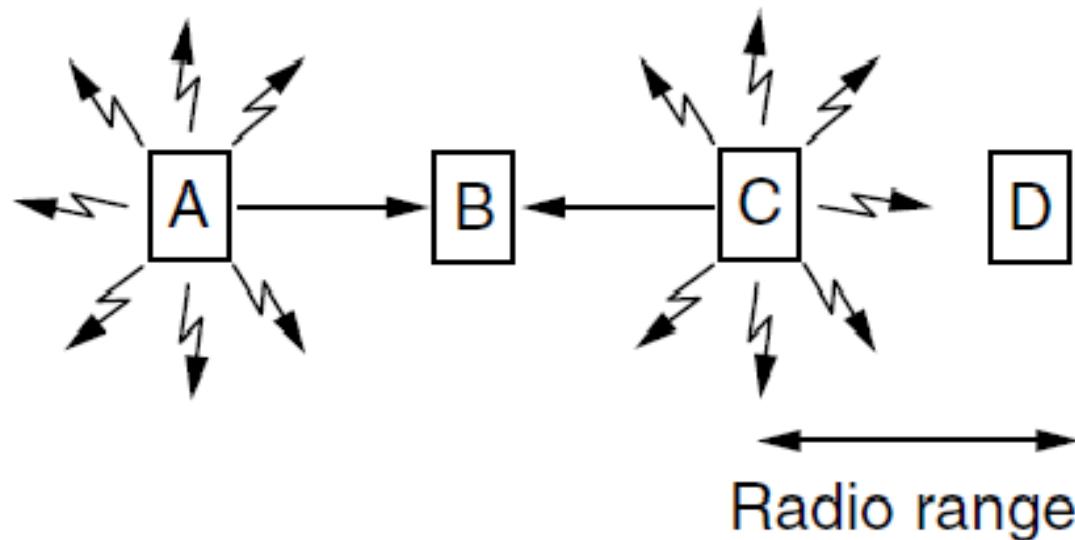
Nodes can't detect collisions, i.e., sense while sending

- Makes collisions expensive and to be avoided

Wireless LANs (2) – Hidden terminals

Hidden terminals are senders that cannot sense each other but nonetheless collide at intended receiver

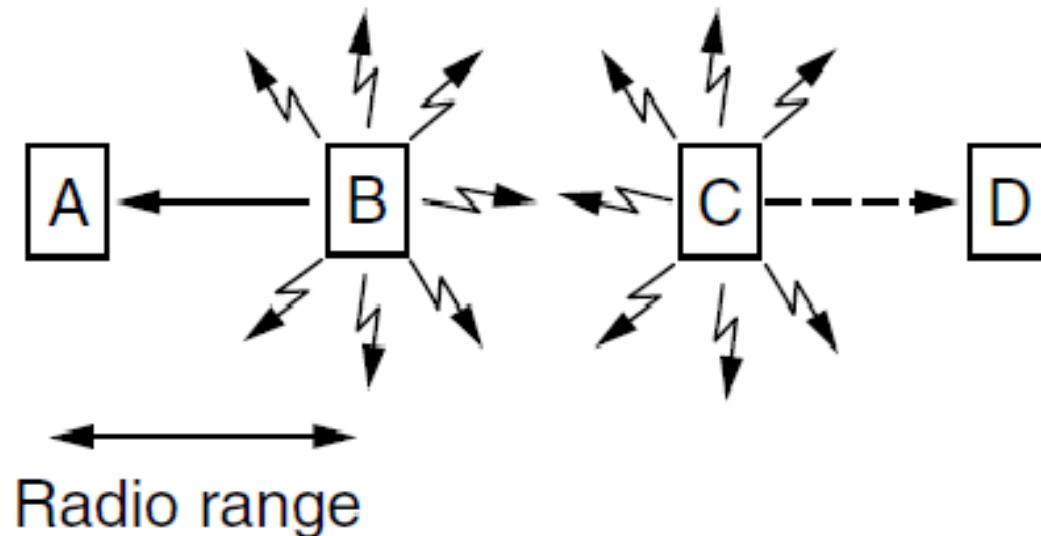
- Want to prevent; loss of efficiency
- A and C are hidden terminals when sending to B



Wireless LANs (3) – Exposed terminals

Exposed terminals are senders who can sense each other but still transmit safely (to different receivers)

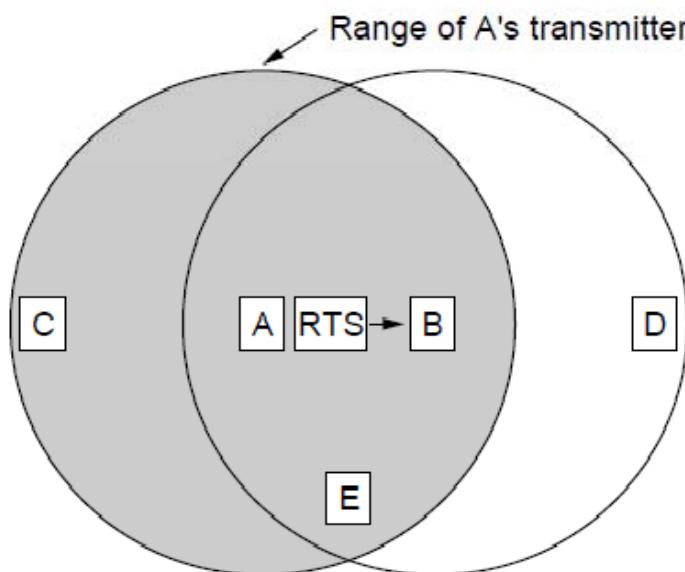
- Desirably concurrency; improves performance
- $B \rightarrow A$ and $C \rightarrow D$ are exposed terminals



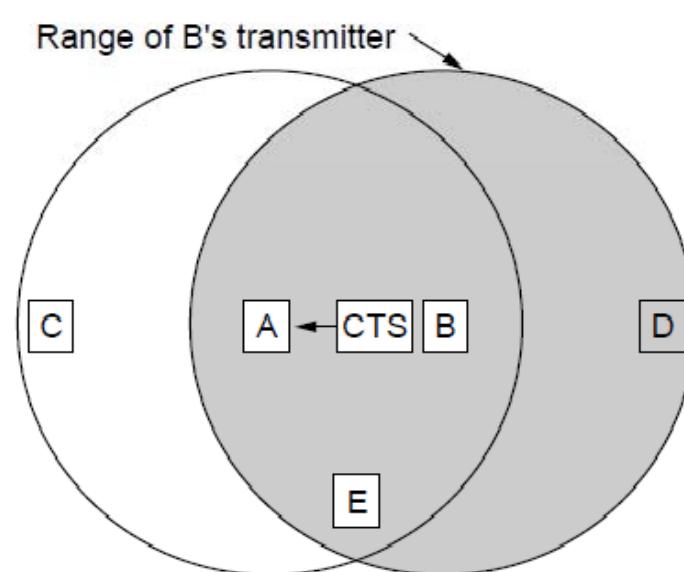
Wireless LANs (4) – MACA

MACA protocol grants access for A to send to B:

- A sends RTS to B [left]; B replies with CTS [right]
- A can send with exposed but no hidden terminals



A sends RTS to B; C and E
hear and defer for CTS



B replies with CTS; D and
E hear and defer for data

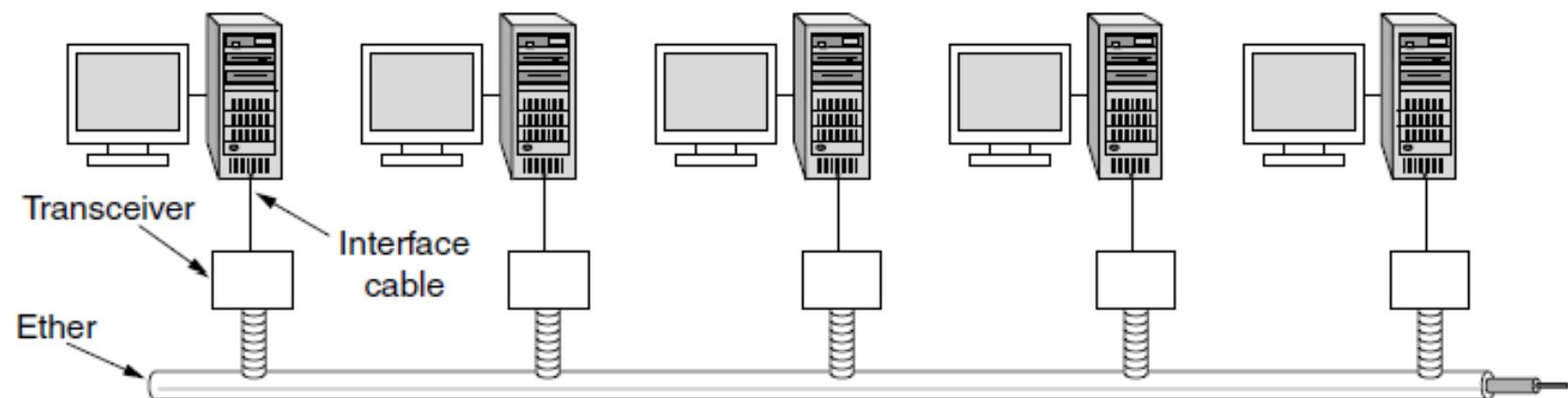
Ethernet

- Classic Ethernet »
- Switched/Fast Ethernet »
- Gigabit/10 Gigabit Ethernet »

Classic Ethernet (1) – Physical Layer

One shared coaxial cable to which all hosts attached

- Up to 10 Mbps, with Manchester encoding
- Hosts ran the classic Ethernet protocol for access



Classic Ethernet (2) – MAC

MAC protocol is 1-persistent CSMA/CD (earlier)

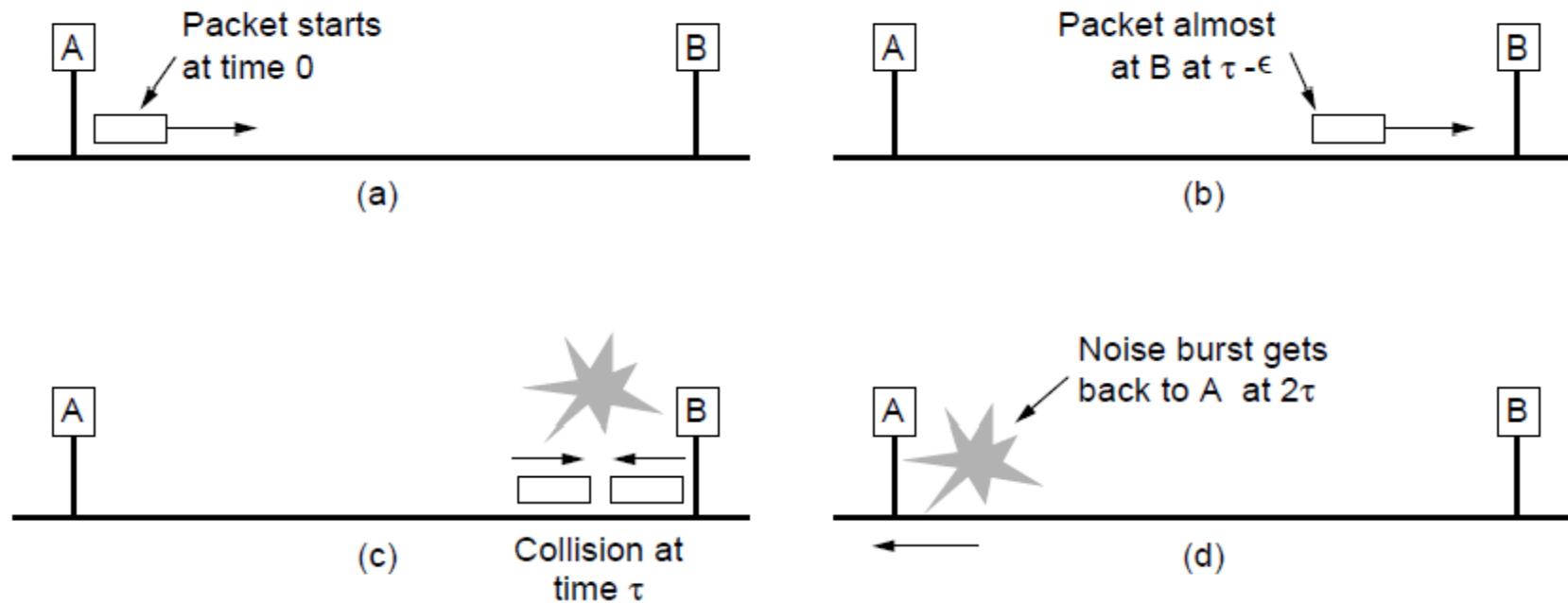
- Random delay (backoff) after collision is computed with BEB (Binary Exponential Backoff)
- Frame format is still used with modern Ethernet.

| | Bytes | 8 | 6 | 6 | 2 | 0-1500 | 0-46 | 4 | |
|-------------------|-------|----------|---------------------|---------------------|----------------|--------|------|-----------|-----------|
| Ethernet (DIX) | | Preamble | Destination address | Source address | Type | Data | Pad | Check-sum | |
| IEEE 802.3 | | Preamble | S o F | Destination address | Source address | Length | Data | Pad | Check-sum |

Classic Ethernet (3) – MAC

Collisions can occur and take as long as 2τ to detect

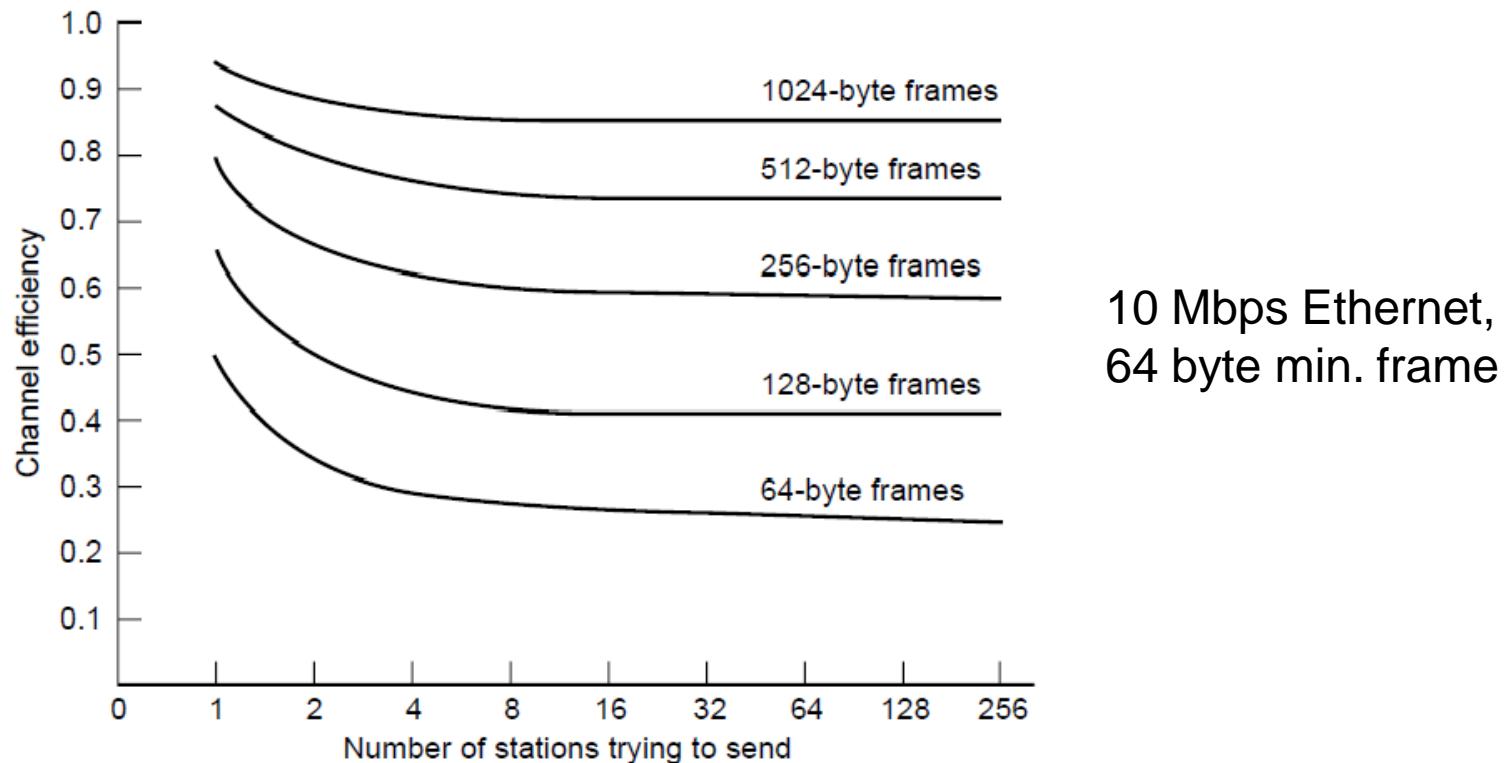
- τ is the time it takes to propagate over the Ethernet
- Leads to minimum packet size for reliable detection



Classic Ethernet (4) – Performance

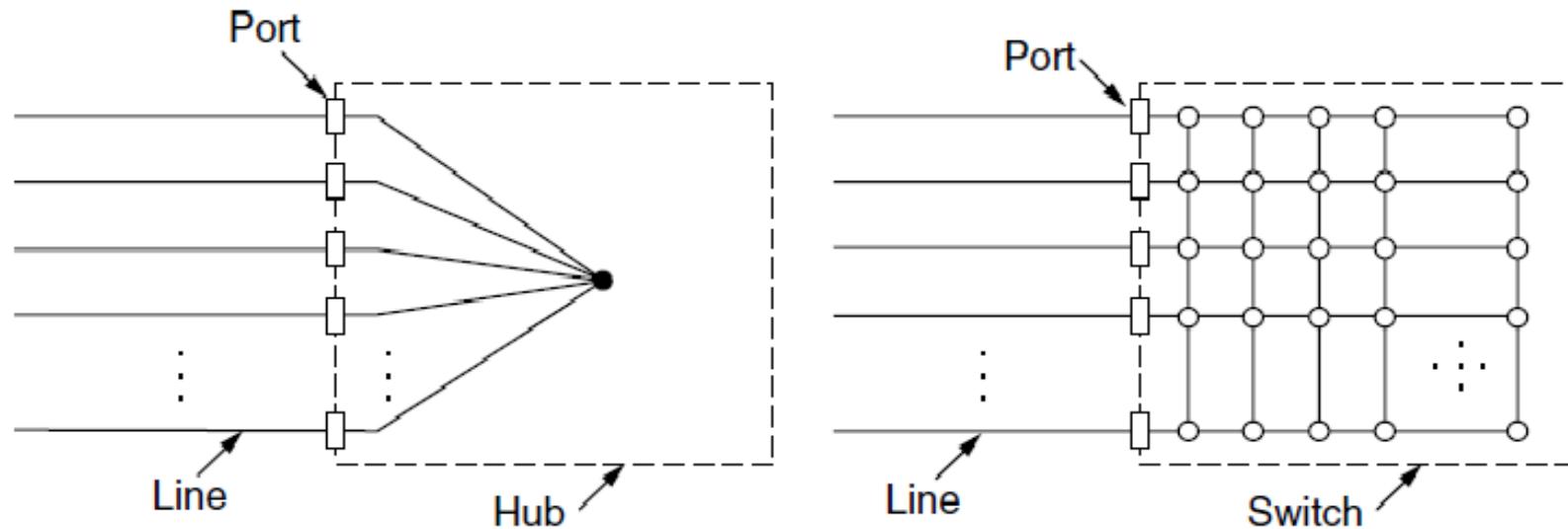
Efficient for large frames, even with many senders

- Degrades for small frames (and long LANs)



Switched/Fast Ethernet (1)

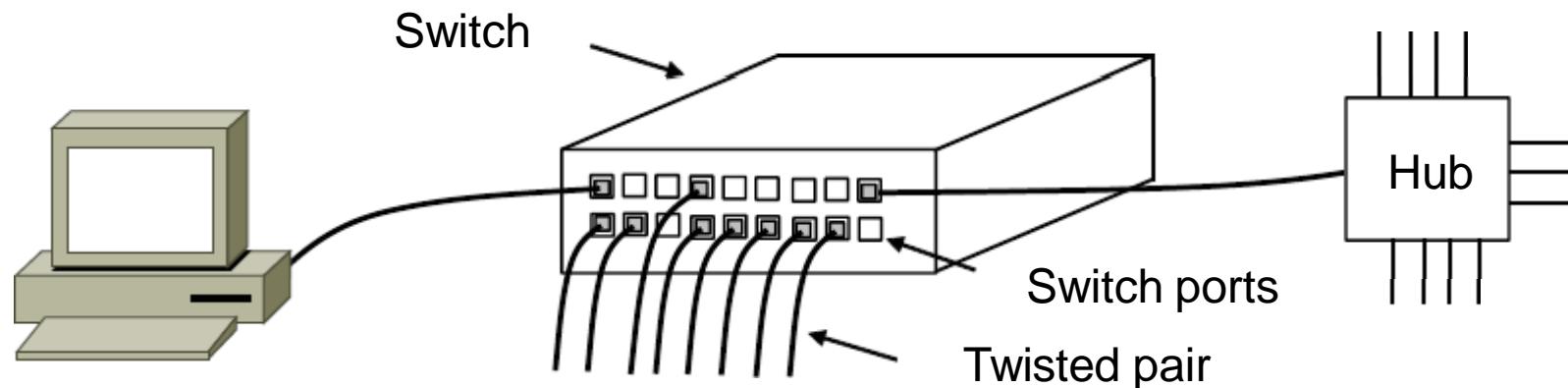
- Hubs wire all lines into a single CSMA/CD domain
- Switches isolate each port to a separate domain
 - Much greater throughput for multiple ports
 - No need for CSMA/CD with full-duplex lines



Switched/Fast Ethernet (2)

Switches can be wired to computers, hubs and switches

- Hubs concentrate traffic from computers
- More on how to switch frames the in 4.8



Switched/Fast Ethernet (3)

Fast Ethernet extended Ethernet from 10 to 100 Mbps

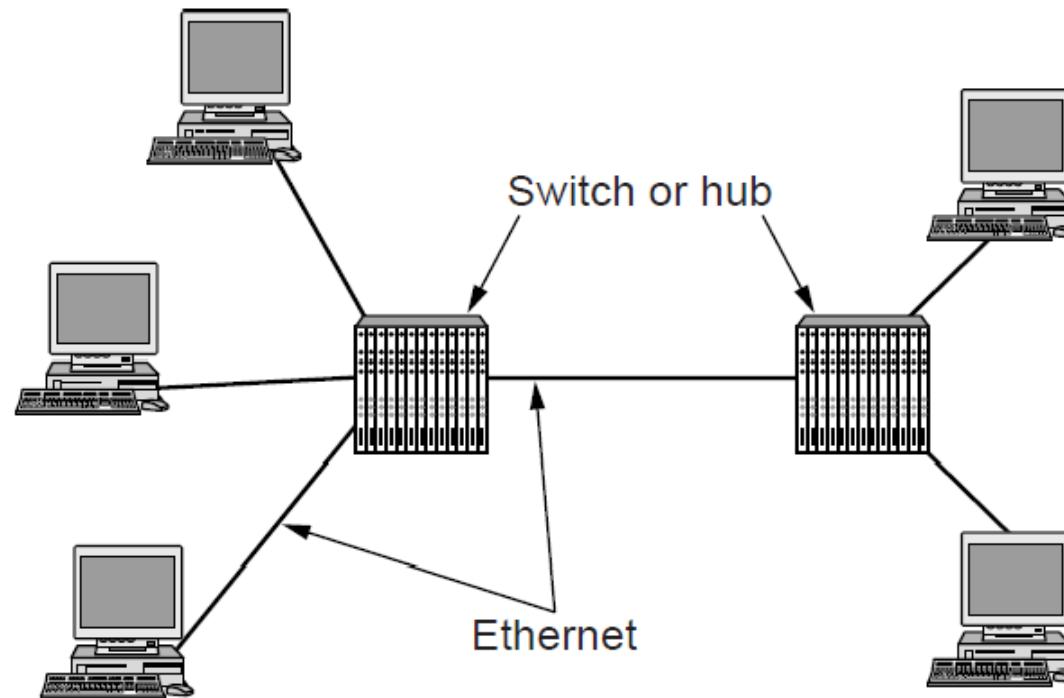
- Twisted pair (with Cat 5) dominated the market

| Name | Cable | Max. segment | Advantages |
|------------|--------------|--------------|-------------------------------------|
| 100Base-T4 | Twisted pair | 100 m | Uses category 3 UTP |
| 100Base-TX | Twisted pair | 100 m | Full duplex at 100 Mbps (Cat 5 UTP) |
| 100Base-FX | Fiber optics | 2000 m | Full duplex at 100 Mbps; long runs |

Gigabit / 10 Gigabit Ethernet (1)

Switched Gigabit Ethernet is now the garden variety

- With full-duplex lines between computers/switches



Gigabit / 10 Gigabit Ethernet (1)

- Gigabit Ethernet is commonly run over twisted pair

| Name | Cable | Max. segment | Advantages |
|-------------|----------------|--------------|---|
| 1000Base-SX | Fiber optics | 550 m | Multimode fiber (50, 62.5 microns) |
| 1000Base-LX | Fiber optics | 5000 m | Single (10 μ) or multimode (50, 62.5 μ) |
| 1000Base-CX | 2 Pairs of STP | 25 m | Shielded twisted pair |
| 1000Base-T | 4 Pairs of UTP | 100 m | Standard category 5 UTP |

- 10 Gigabit Ethernet is being deployed where needed

| Name | Cable | Max. segment | Advantages |
|-------------|-------------------|--------------|--------------------------------|
| 10GBase-SR | Fiber optics | Up to 300 m | Multimode fiber (0.85 μ) |
| 10GBase-LR | Fiber optics | 10 km | Single-mode fiber (1.3 μ) |
| 10GBase-ER | Fiber optics | 40 km | Single-mode fiber (1.5 μ) |
| 10GBase-CX4 | 4 Pairs of twinax | 15 m | Twinaxial copper |
| 10GBase-T | 4 Pairs of UTP | 100 m | Category 6a UTP |

- 40/100 Gigabit Ethernet is under development

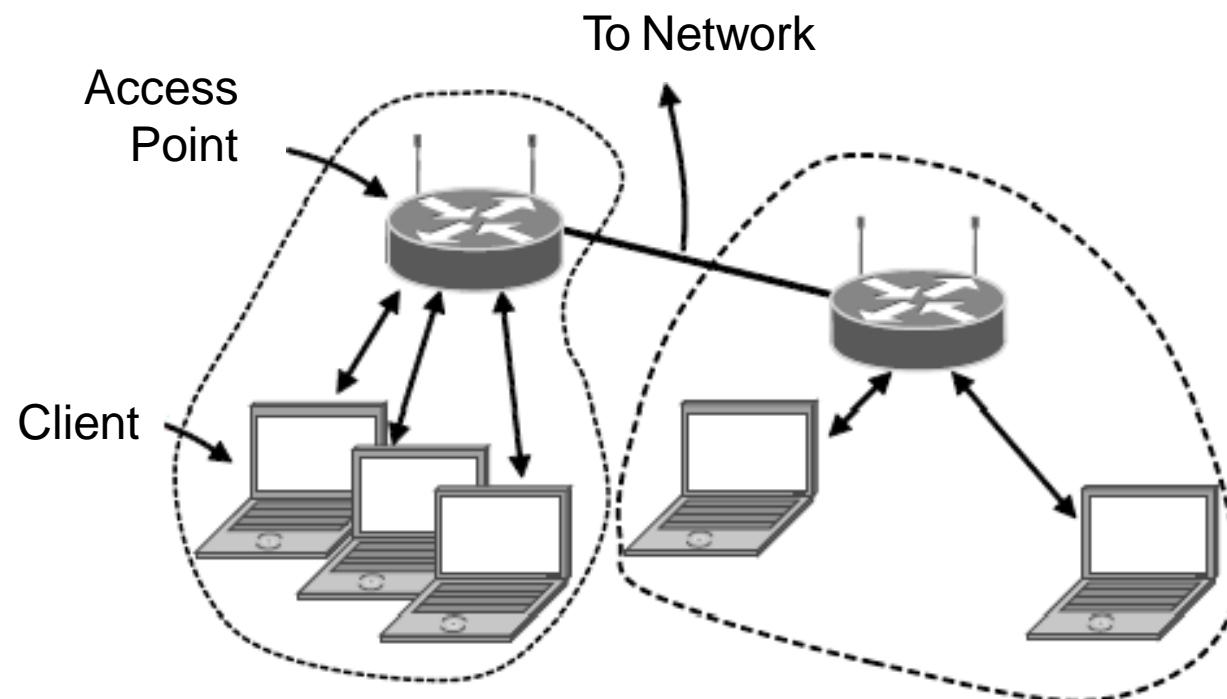
Wireless LANs

- 802.11 architecture/protocol stack »
- 802.11 physical layer »
- 802.11 MAC »
- 802.11 frames »

802.11 Architecture/Protocol Stack (1)

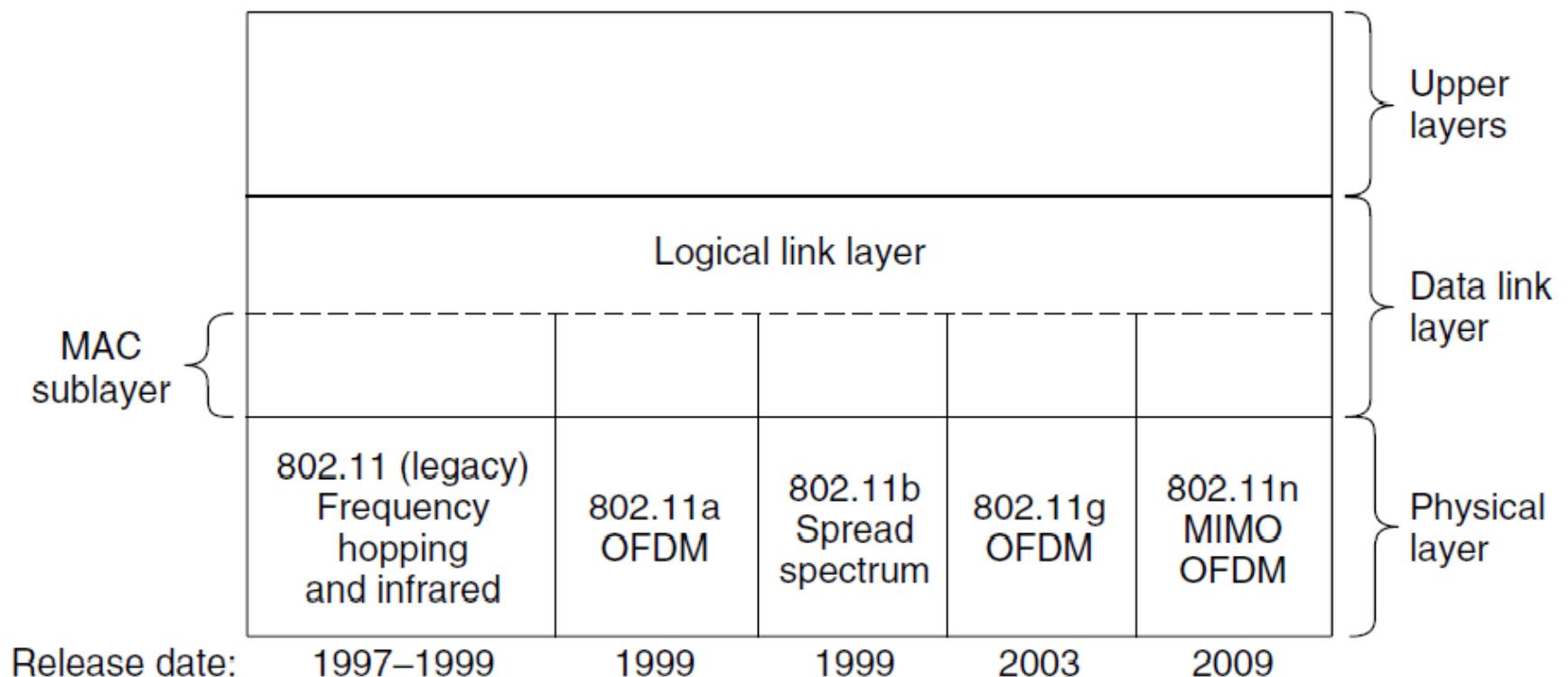
Wireless clients associate to a wired AP (Access Point)

- Called infrastructure mode; there is also ad-hoc mode with no AP, but that is rare.



802.11 Architecture/Protocol Stack (2)

MAC is used across different physical layers



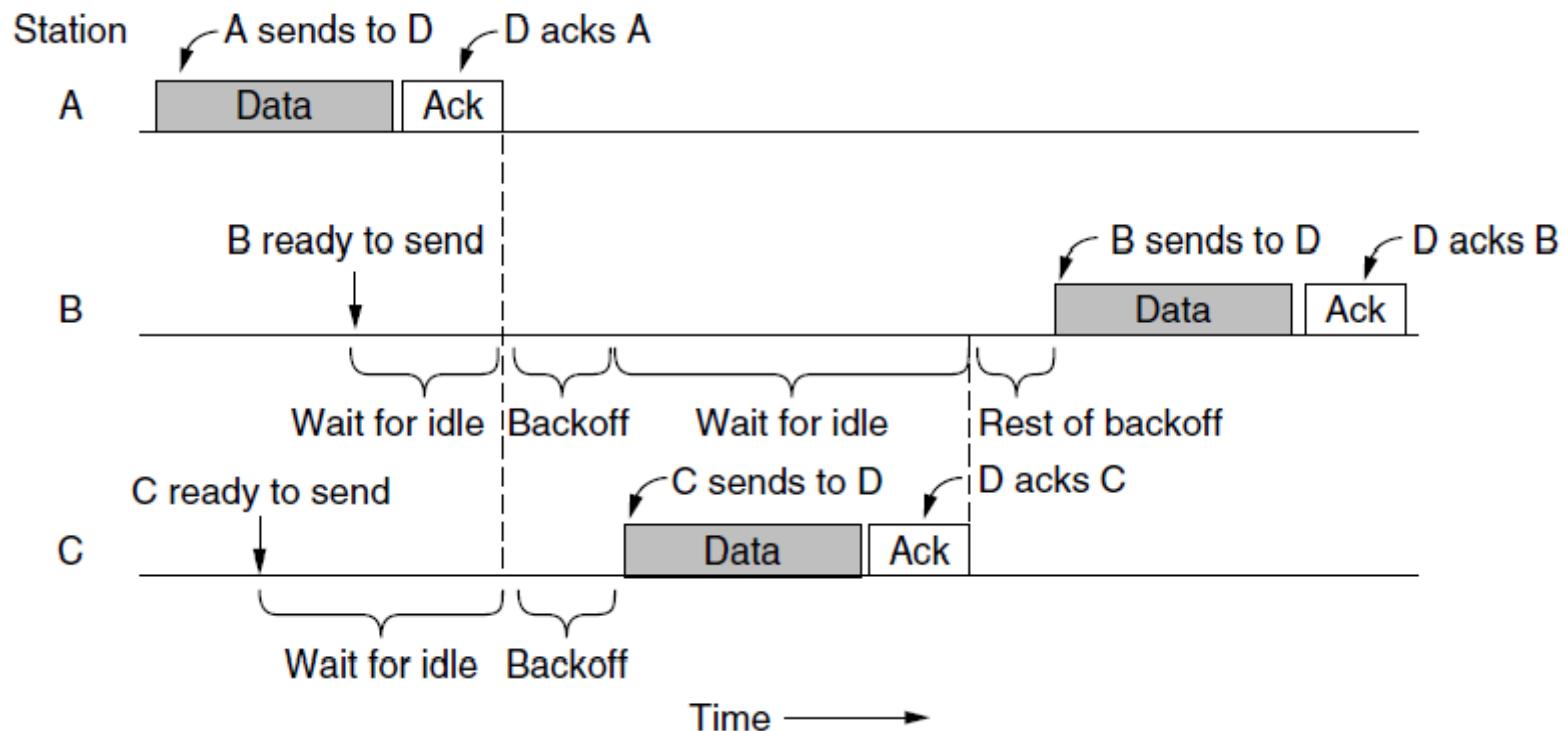
802.11 physical layer

- NICs are compatible with multiple physical layers
 - E.g., 802.11 a/b/g

| Name | Technique | Max. Bit Rate |
|---------|---------------------------|---------------|
| 802.11b | Spread spectrum, 2.4 GHz | 11 Mbps |
| 802.11g | OFDM, 2.4 GHz | 54 Mbps |
| 802.11a | OFDM, 5 GHz | 54 Mbps |
| 802.11n | OFDM with MIMO, 2.4/5 GHz | 600 Mbps |

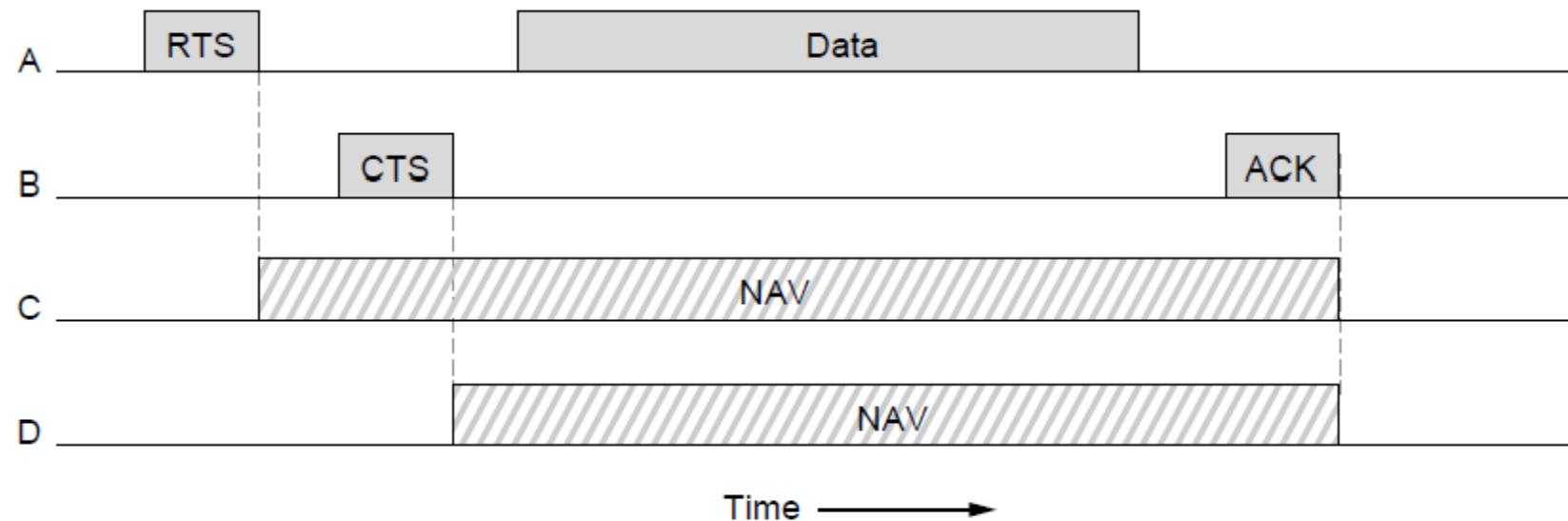
802.11 MAC (1)

- CSMA/CA inserts backoff slots to avoid collisions
- MAC uses ACKs/retransmissions for wireless errors



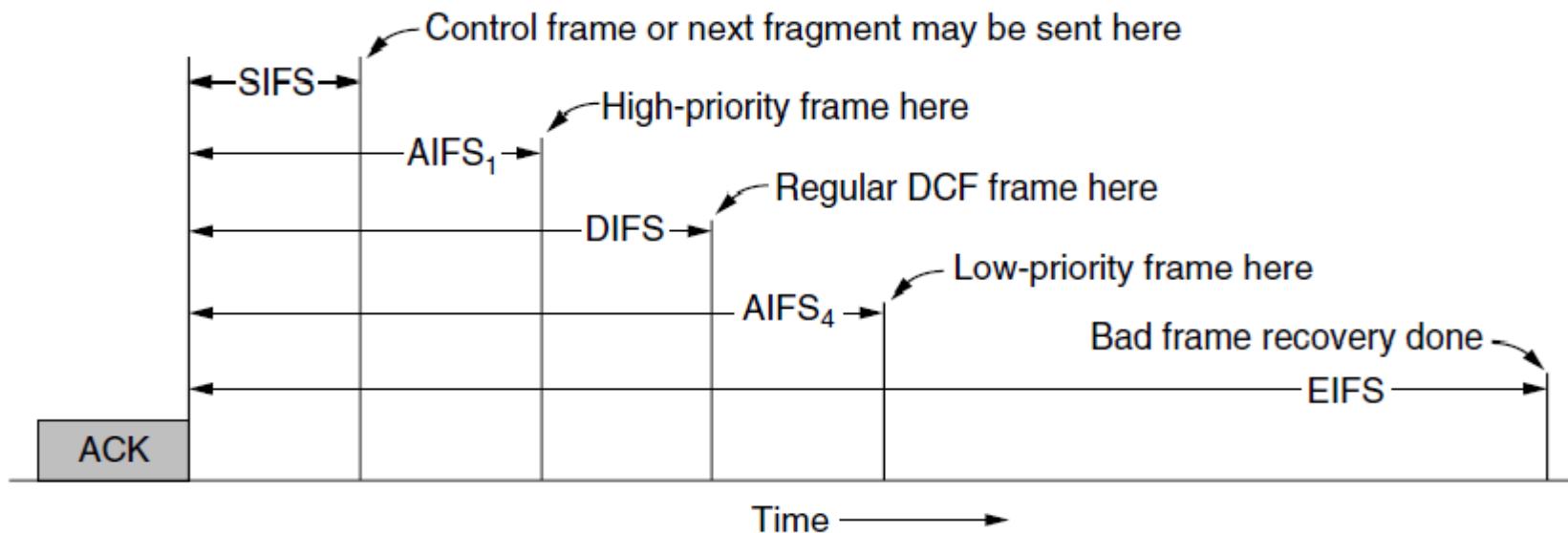
802.11 MAC (2)

Virtual channel sensing with the NAV and optional RTS/CTS (often not used) avoids hidden terminals



802.11 MAC (3)

- Different backoff slot times add quality of service
 - Short intervals give preferred access, e.g., control, VoIP
- MAC has other mechanisms too, e.g., power save



802.11 Frames

- Frames vary depending on their type (Frame control)
- Data frames have 3 addresses to pass via APs

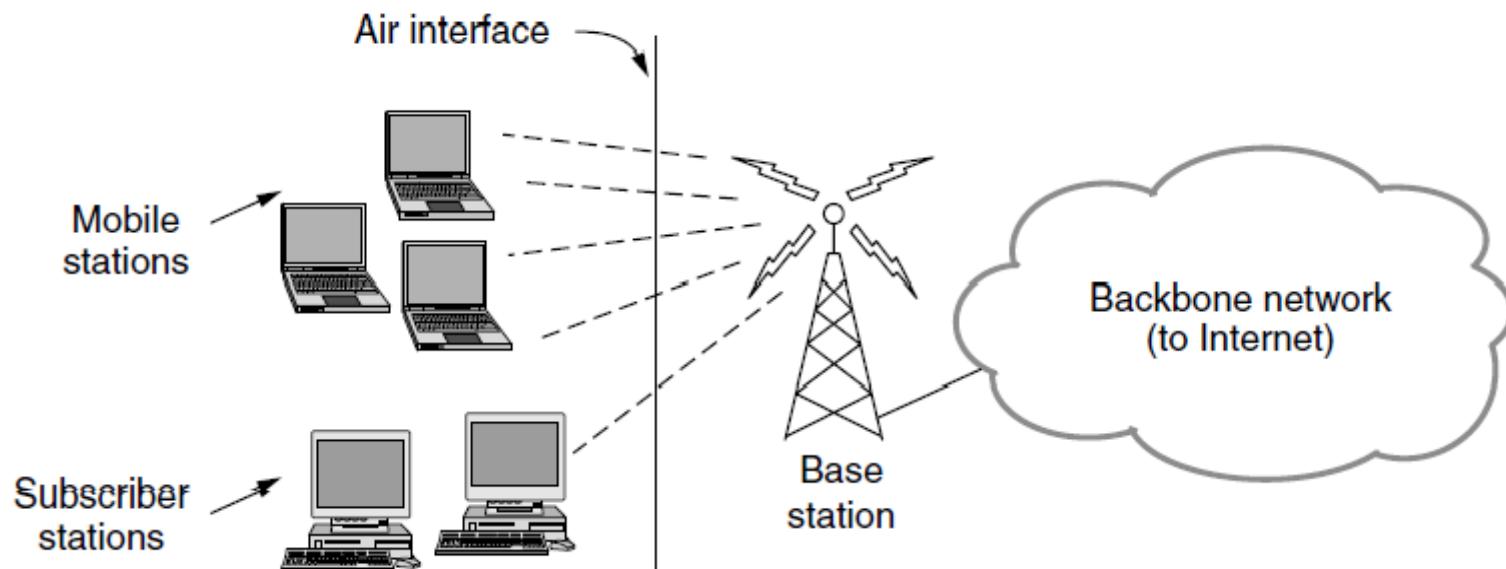
| Bytes | 2 | 2 | 6 | 6 | 6 | 2 | 0–2312 | 4 | |
|-------|---------------|-----------|--------------------------|----------------------------|-----------|------------|-----------|----------------|-------|
| | Frame control | Duration | Address 1 (recipient) | Address 2 (transmitter) | Address 3 | Sequence | Data | Check sequence | |
| Bits | 2 | 2 | 4 | 1 | 1 | 1 | 1 | 1 | |
| | Version = 00 | Type = 10 | Subtype = 0000 | To DS | From DS | More frag. | Retry | Pwr. mgt. | |
| | | | | | | | More data | Protected | Order |

Broadband Wireless

- 802.16 Architecture / Protocol Stack »
- 802.16 Physical Layer »
- 802.16 MAC »
- 802.16 Frames »

802.16 Architecture/Protocol Stack (1)

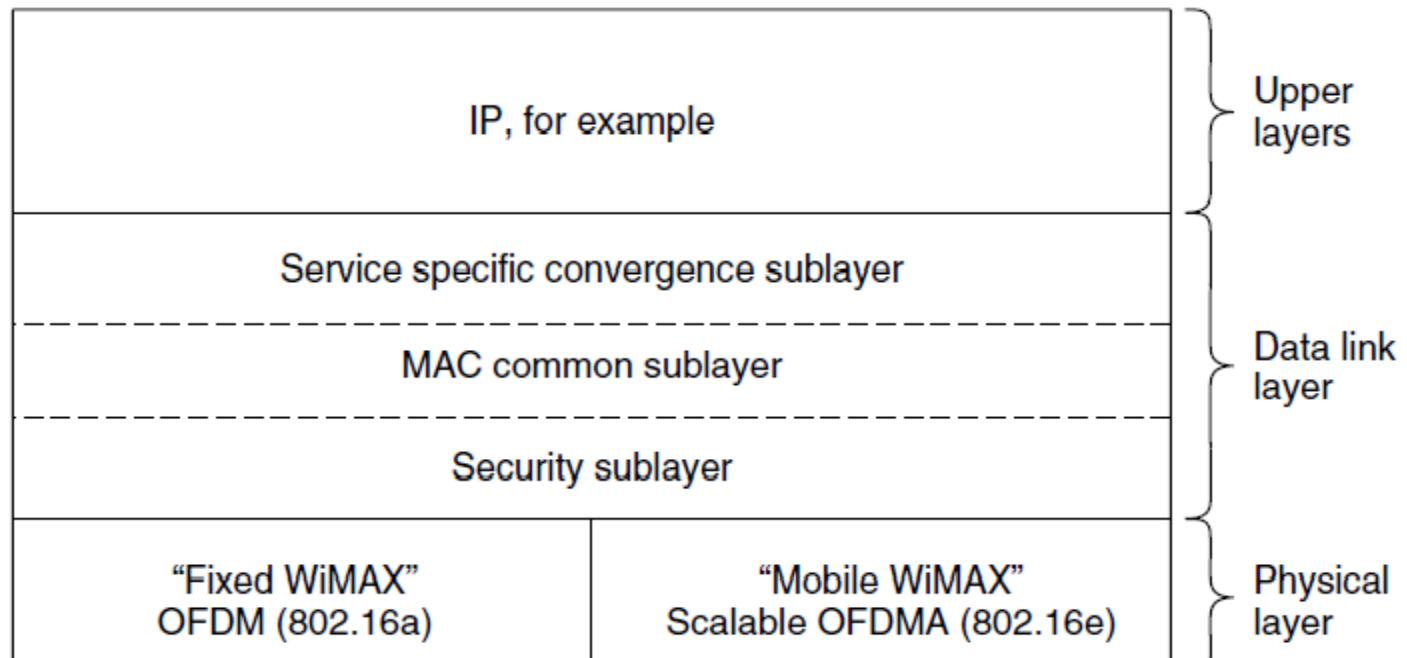
Wireless clients connect to a wired basestation (like 3G)



802.16 Architecture/Protocol Stack (2)

MAC is connection-oriented; IP is connectionless

- Convergence sublayer maps between the two



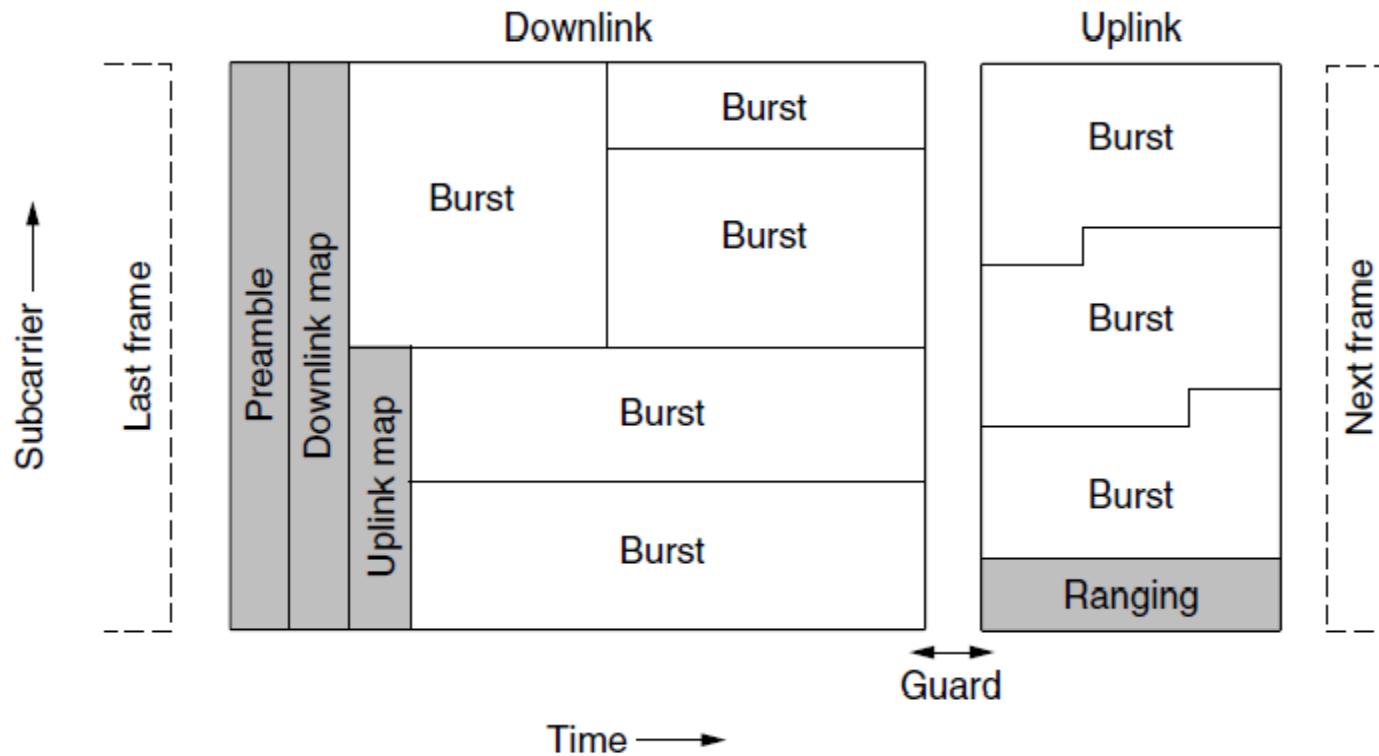
Release date:

2003

2005

802.16 Physical Layer

Based on OFDM; base station gives mobiles bursts (subcarrier/time frame slots) for uplink and downlink



802.16 MAC

Connection-oriented with base station in control

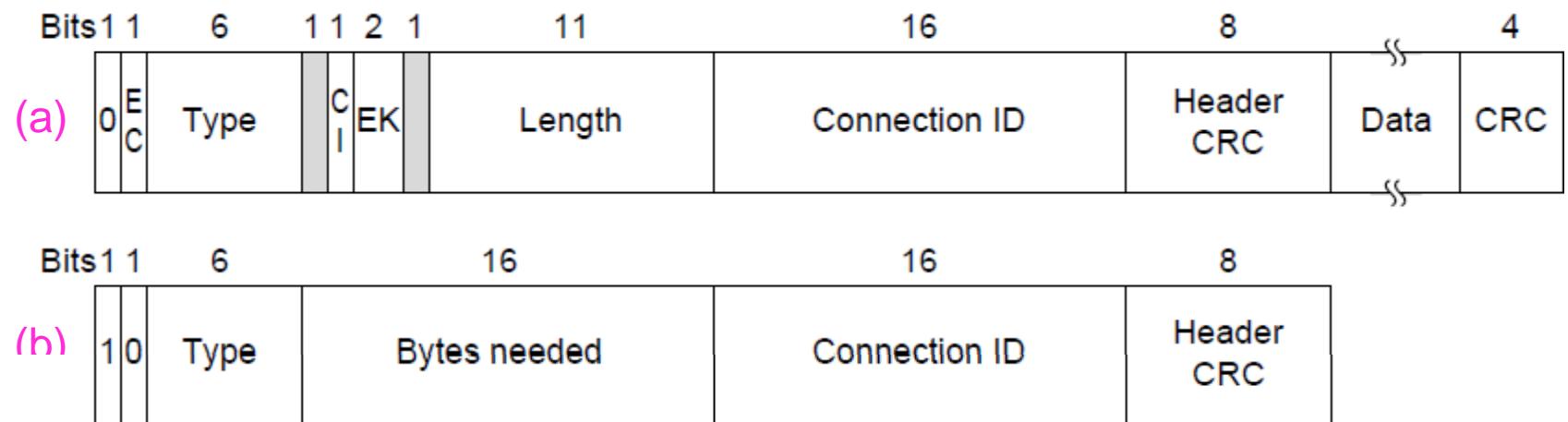
- Clients request the bandwidth they need

Different kinds of service can be requested:

- Constant bit rate, e.g., uncompressed voice
- Real-time variable bit rate, e.g., video, Web
- Non-real-time variable bit rate, e.g., file download
- Best-effort for everything else

802.16 Frames

- Frames vary depending on their type
- Connection ID instead of source/dest addresses



(a) A generic frame. (b) A bandwidth request frame

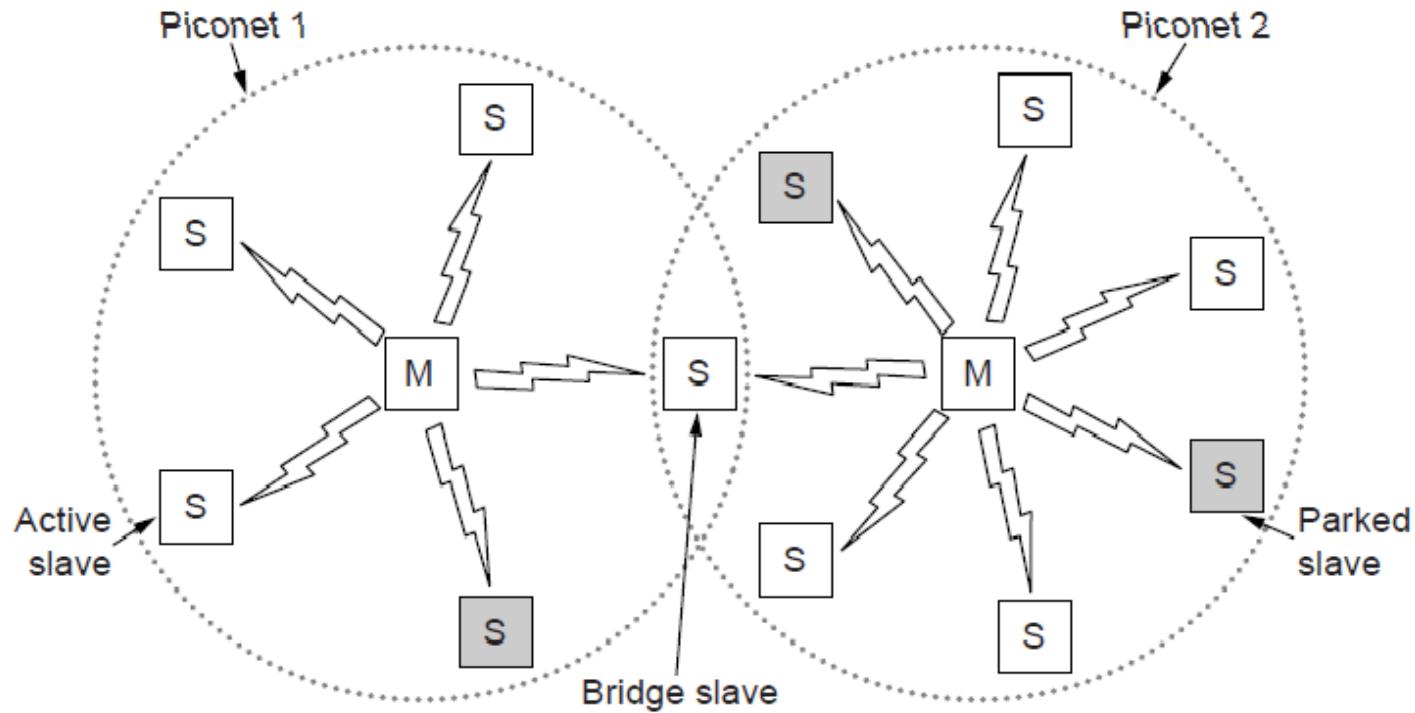
Bluetooth

- Bluetooth Architecture »
- Bluetooth Applications / Protocol »
- Bluetooth Radio / Link Layers »
- Bluetooth Frames »

Bluetooth Architecture

Piconet master is connected to slave wireless devices

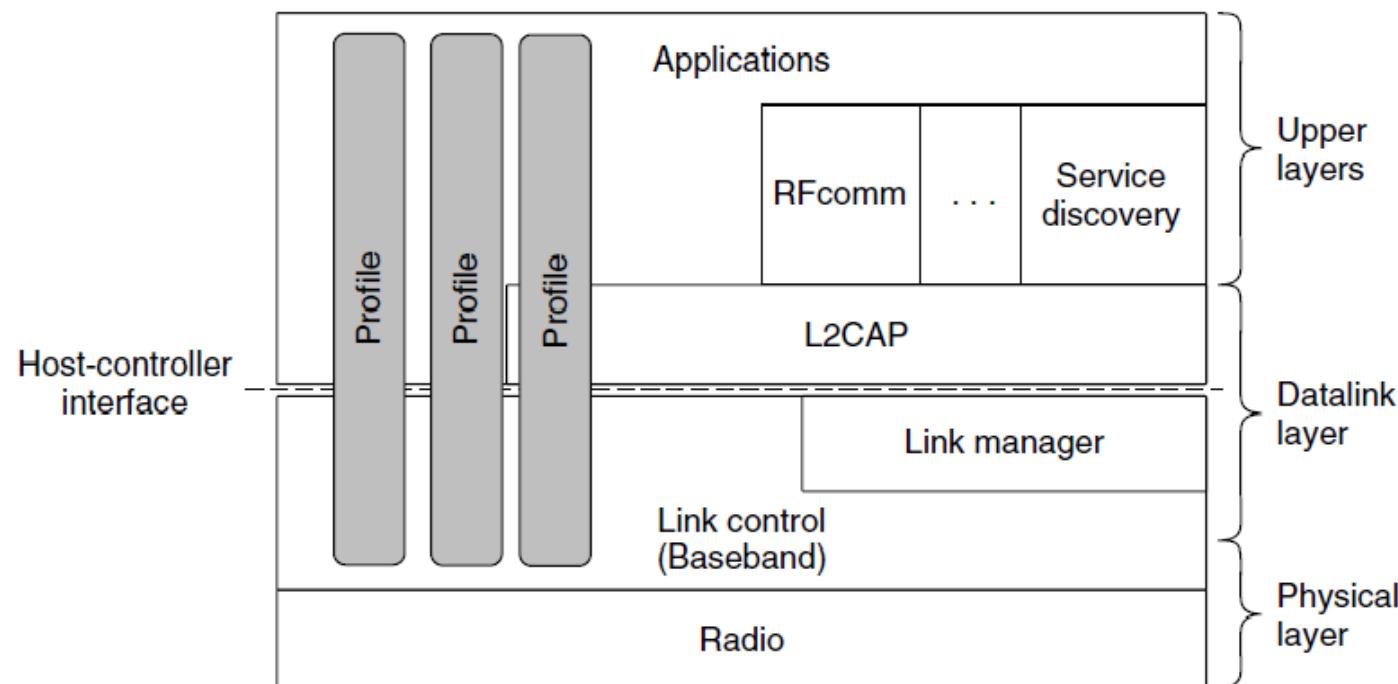
- Slaves may be asleep (parked) to save power
- Two piconets can be bridged into a scatternet



Bluetooth Applications / Protocol Stack

Profiles give the set of protocols for a given application

- 25 profiles, including headset, intercom, streaming audio, remote control, personal area network, ...



Bluetooth Radio / Link Layers

Radio layer

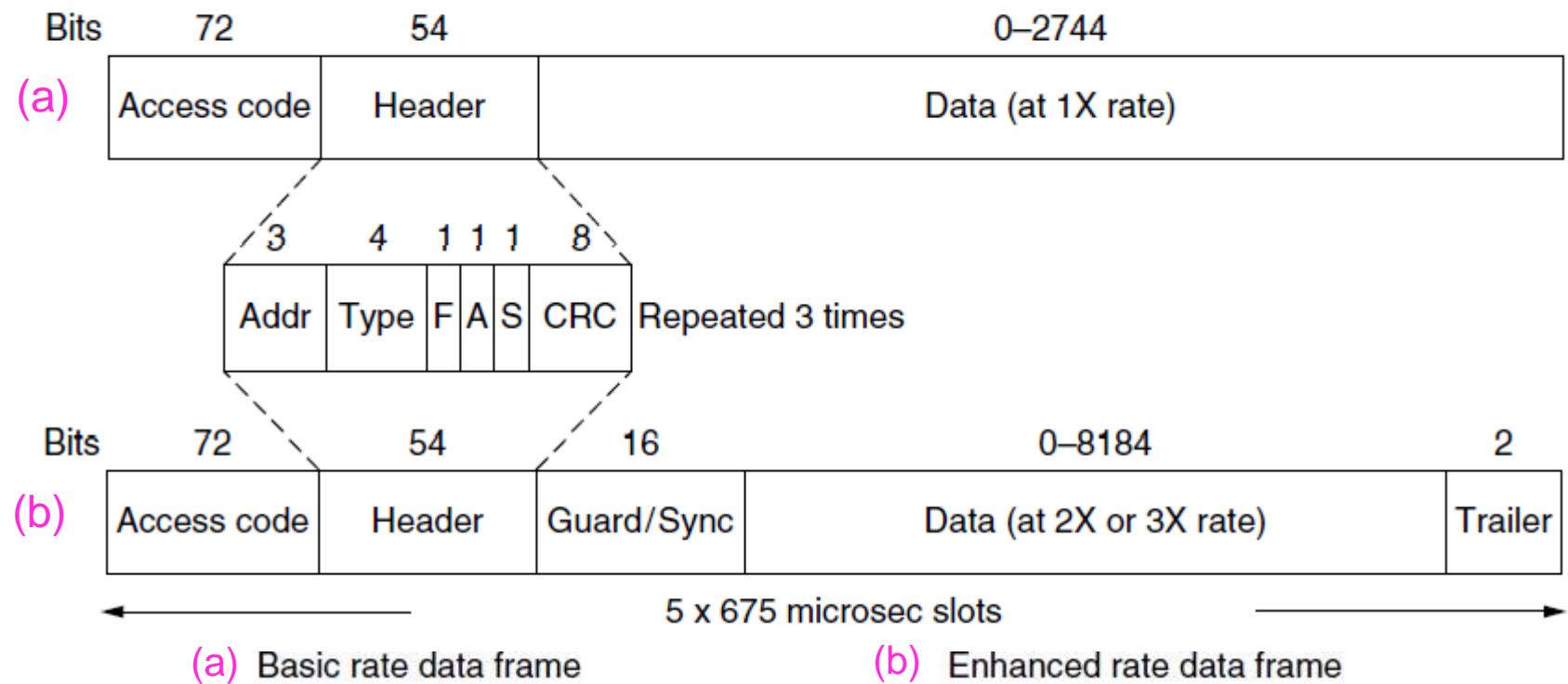
- Uses adaptive frequency hopping in 2.4 GHz band

Link layer

- TDM with timeslots for master and slaves
- Synchronous CO for periodic slots in each direction
- Asynchronous CL for packet-switched data
- Links undergo pairing (user confirms passkey/PIN) to authorize them before use

Bluetooth Frames

Time is slotted; enhanced data rates send faster but for the same time; addresses are only 3 bits for 8 devices

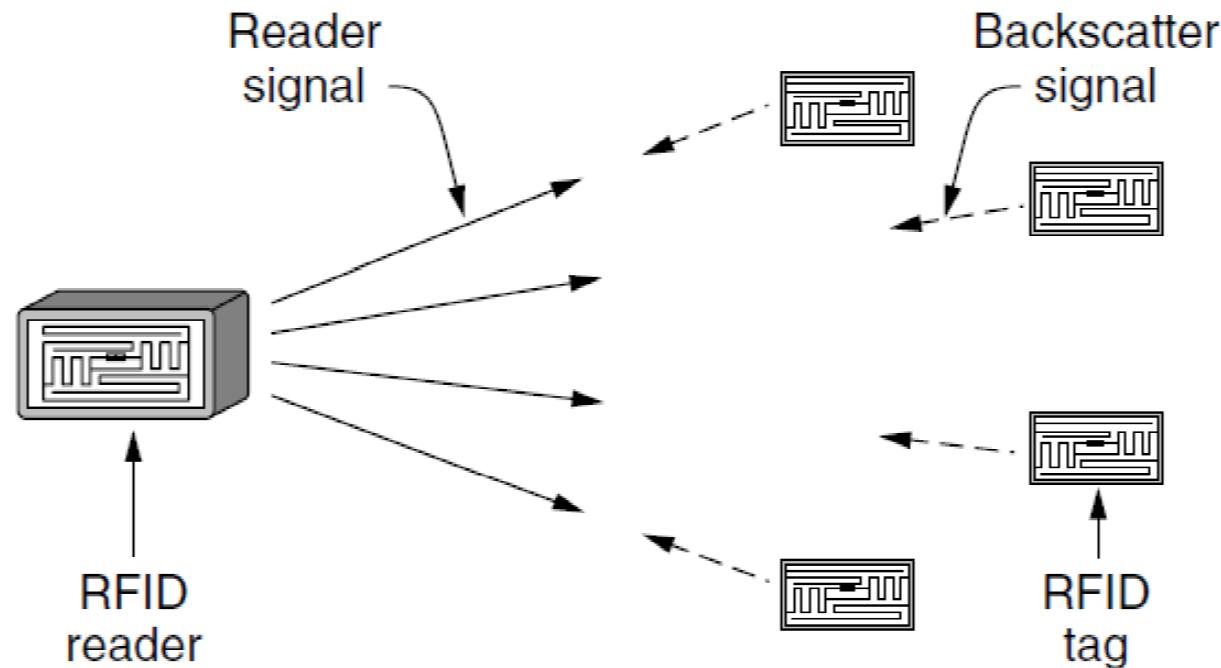


RFID

- Gen 2 Architecture »
- Gen 2 Physical Layer »
- Gen 2 Tag Identification Layer »
- Gen 2 Frames »

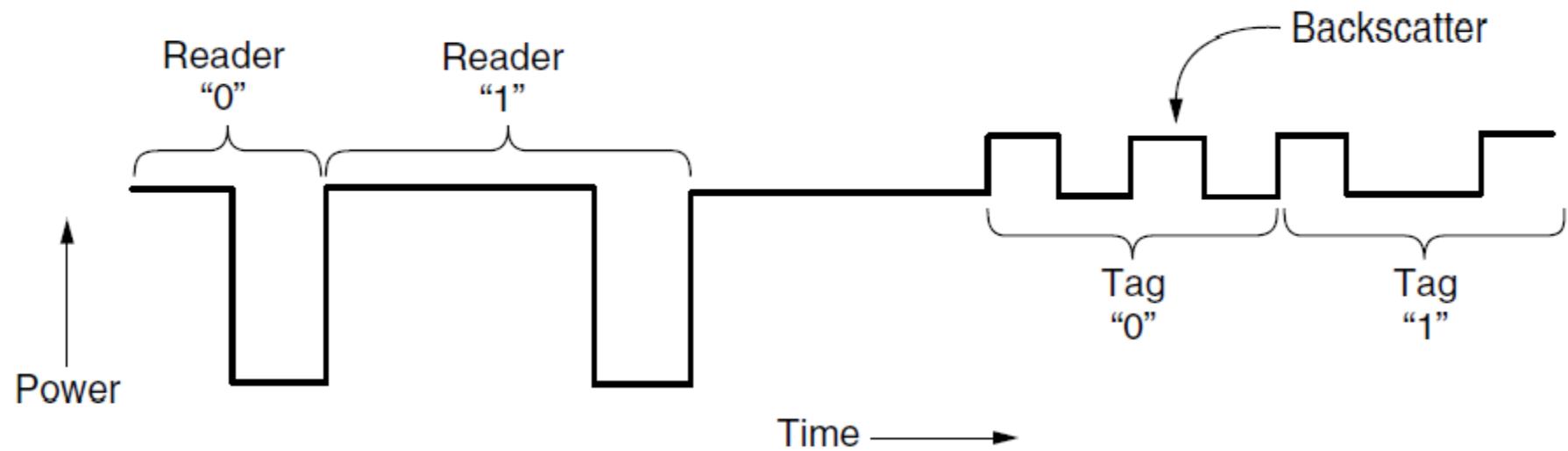
Gen 2 Architecture

Reader signal powers tags; tags reply with backscatter



Gen 2 Physical Layer

- Reader uses duration of on period to send 0/1
- Tag backscatters reader signal in pulses to send 0/1



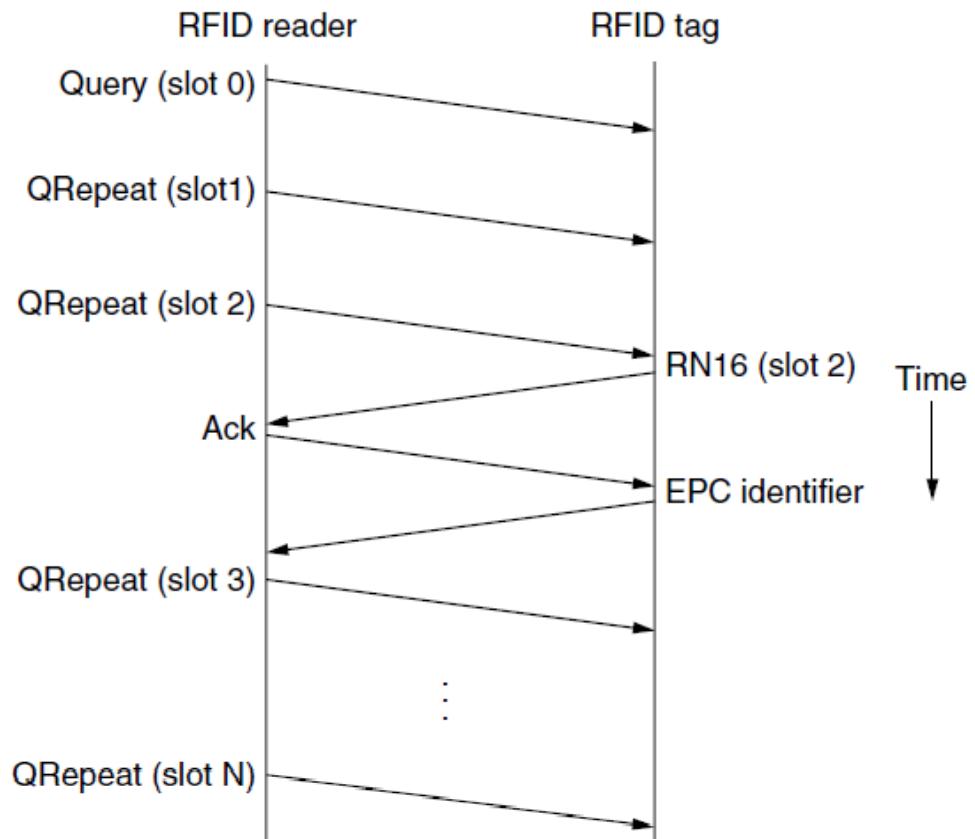
Gen 2 Tag Identification Layer

Reader sends query and sets slot structure

Tags reply (RN16) in a random slot; may collide

Reader asks one tag for its identifier (ACK)

Process continues until no tags are left



Gen 2 Frames

- Reader frames vary depending on type (Command)
 - Query shown below, has parameters and error detection
- Tag responses are simply data
 - Reader sets timing and knows the expected format



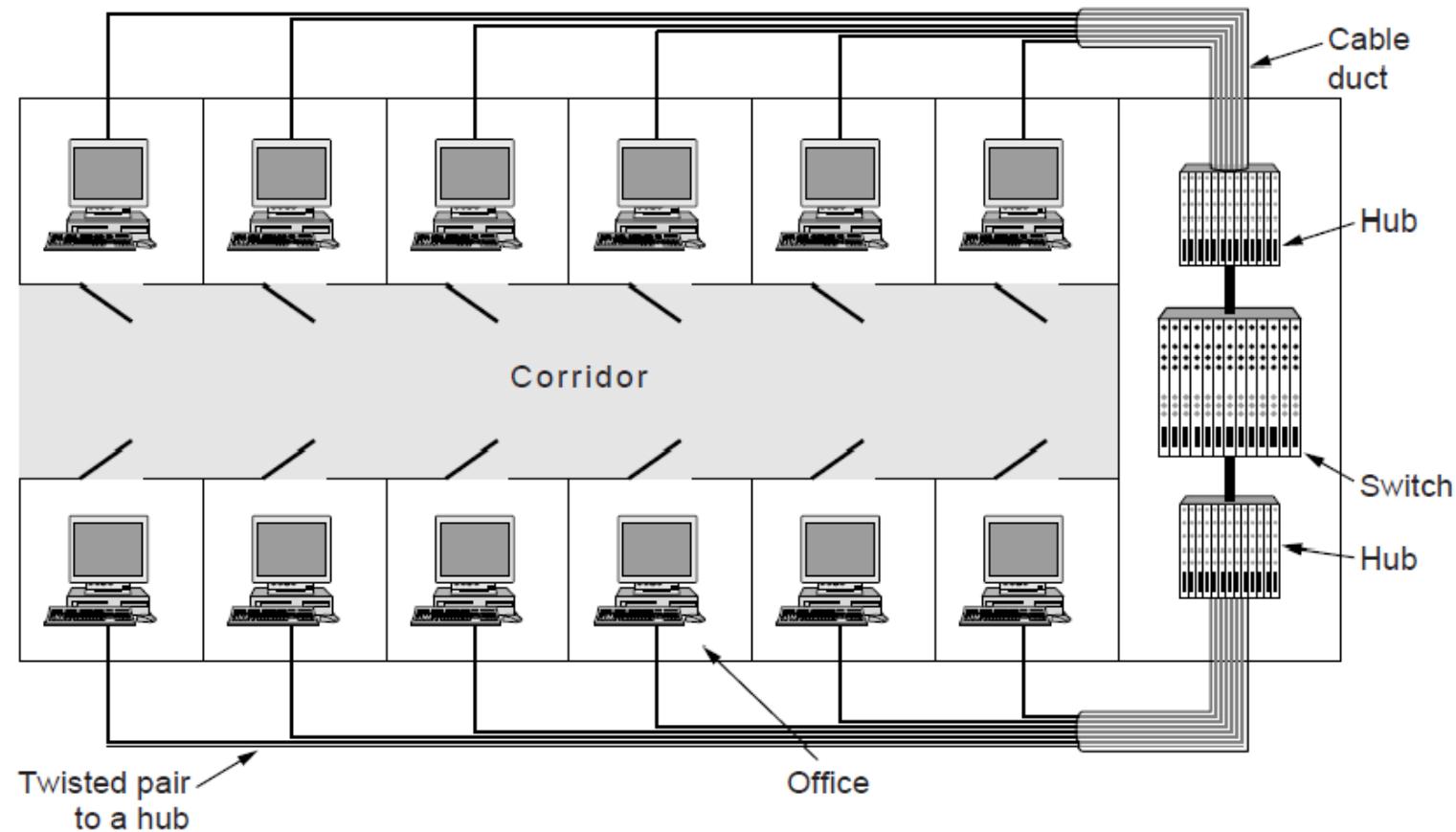
Data Link Layer Switching

- Uses of Bridges »
- Learning Bridges »
- Spanning Tree »
- Repeaters, hubs, bridges, .., routers, gateways »
- Virtual LANs »

Uses of Bridges

Common setup is a building with centralized wiring

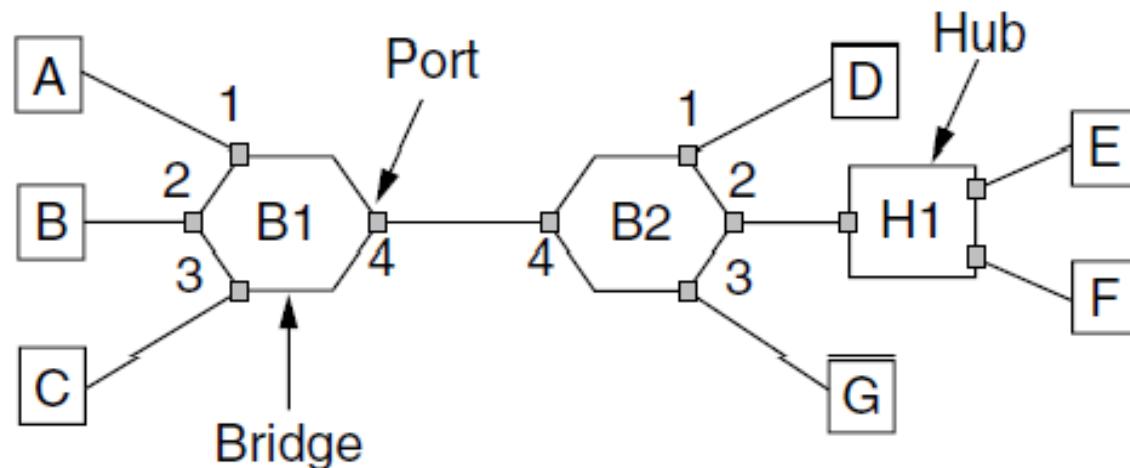
- Bridges (switches) are placed in or near wiring closets



Learning Bridges (1)

A bridge operates as a switched LAN (not a hub)

- Computers, bridges, and hubs connect to its ports



Learning Bridges (2)

Backward learning algorithm picks the output port:

- Associates source address on frame with input port
- Frame with destination address sent to learned port
- Unlearned destinations are sent to all other ports

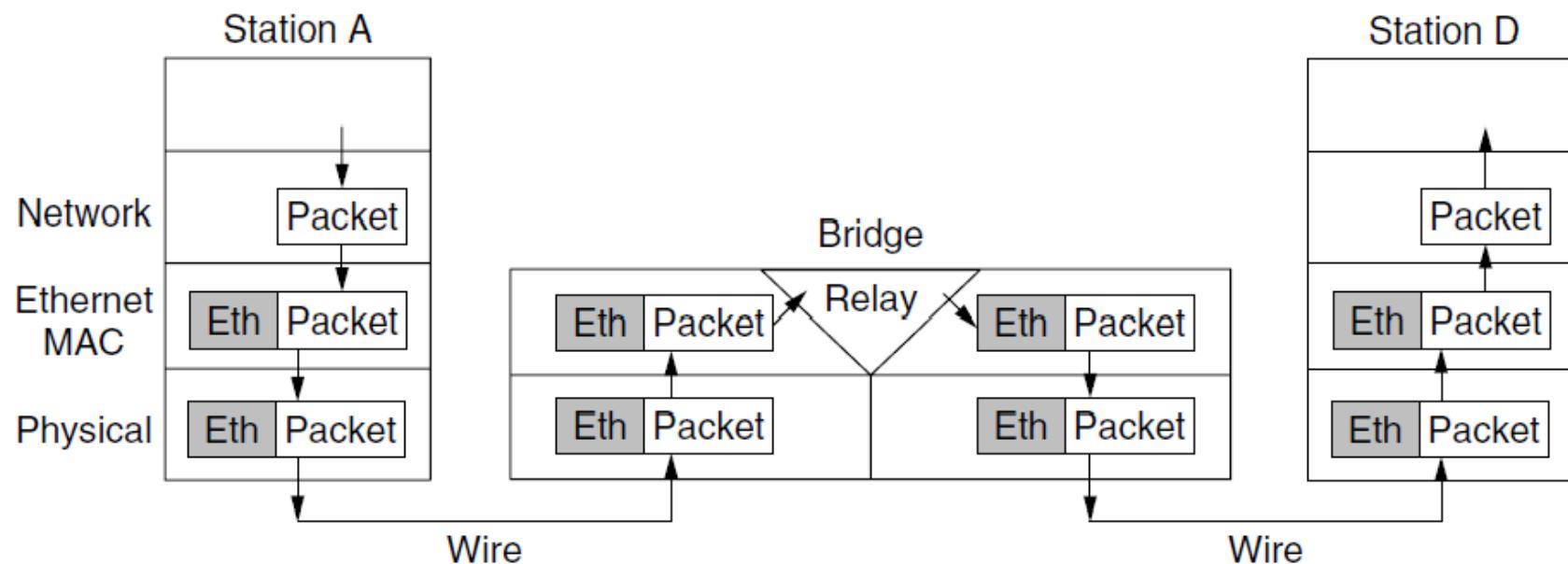
Needs no configuration

- Forget unused addresses to allow changes
- Bandwidth efficient for two-way traffic

Learning Bridges (3)

Bridges extend the Link layer:

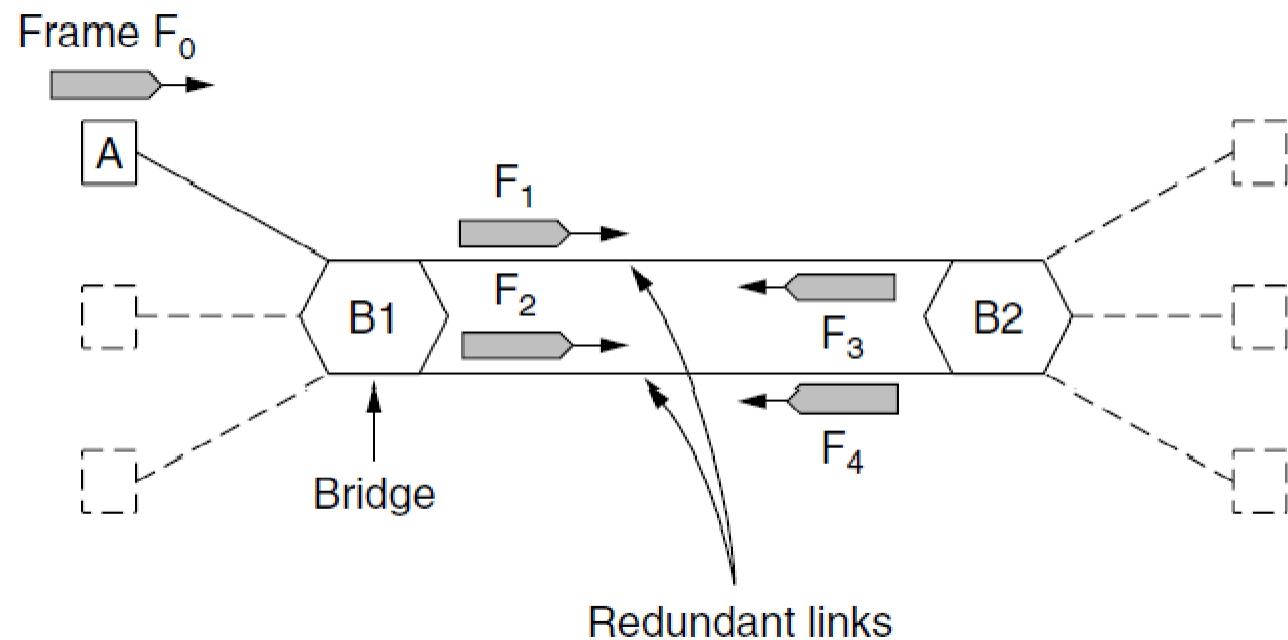
- Use but don't remove Ethernet header/addresses
- Do not inspect Network header



Spanning Tree (1) – Problem

Bridge topologies with loops and only backward learning will cause frames to circulate for ever

- Need spanning tree support to solve problem



Spanning Tree (2) – Algorithm

- Subset of forwarding ports for data is used to avoid loops
- Selected with the spanning tree distributed algorithm by Perlman

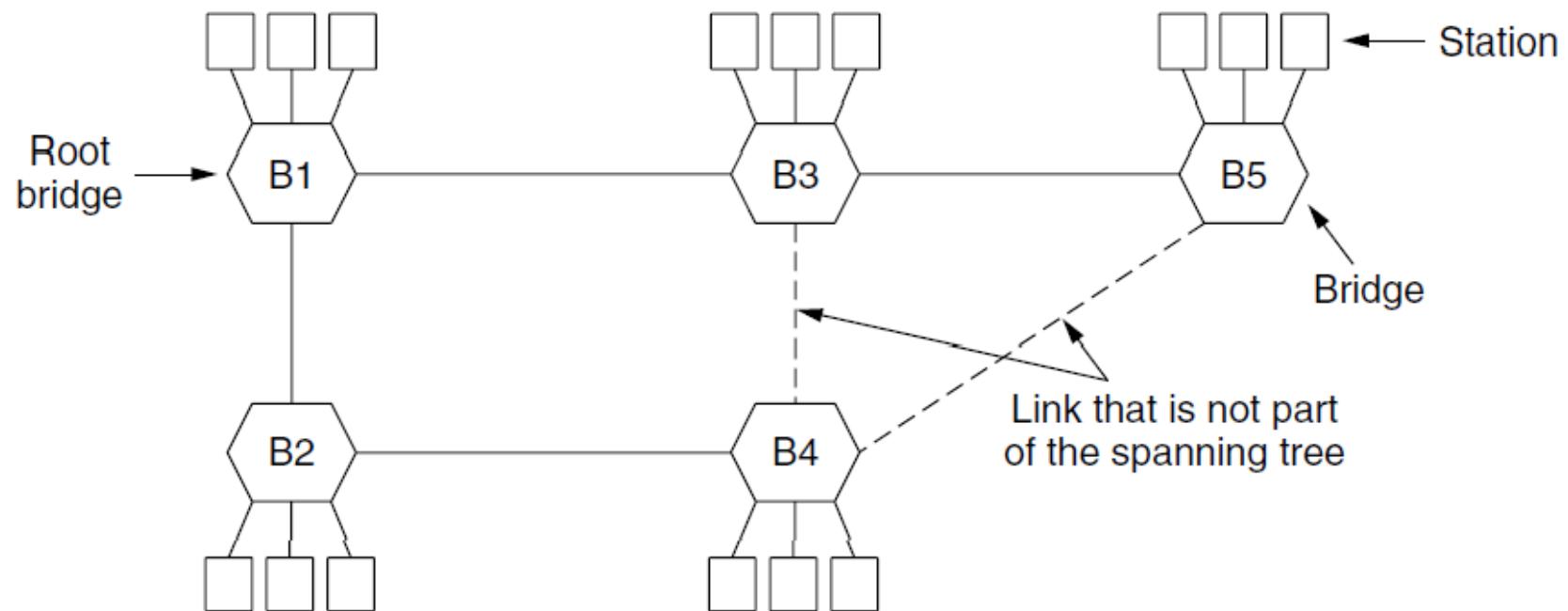
*I think that I shall never see
A graph more lovely than a tree.
A tree whose crucial property
Is loop-free connectivity.
A tree which must be sure to span.
So packets can reach every LAN.
First the Root must be selected
By ID it is elected.
Least cost paths from Root are traced
In the tree these paths are placed.
A mesh is made by folks like me
Then bridges find a spanning tree.*

– Radia Perlman, 1985.

Spanning Tree (3) – Example

After the algorithm runs:

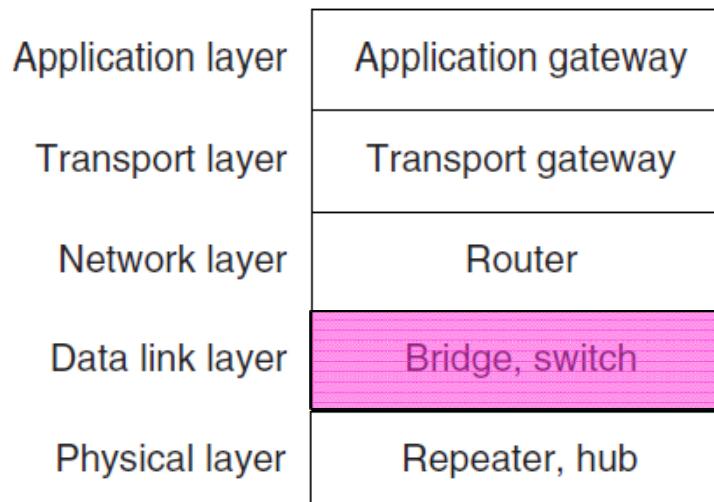
- B1 is the root, two dashed links are turned off
- B4 uses link to B2 (lower than B3 also at distance 1)
- B5 uses B3 (distance 1 versus B4 at distance 2)



Repeaters, Hubs, Bridges, Switches, Routers, & Gateways

Devices are named according to the layer they process

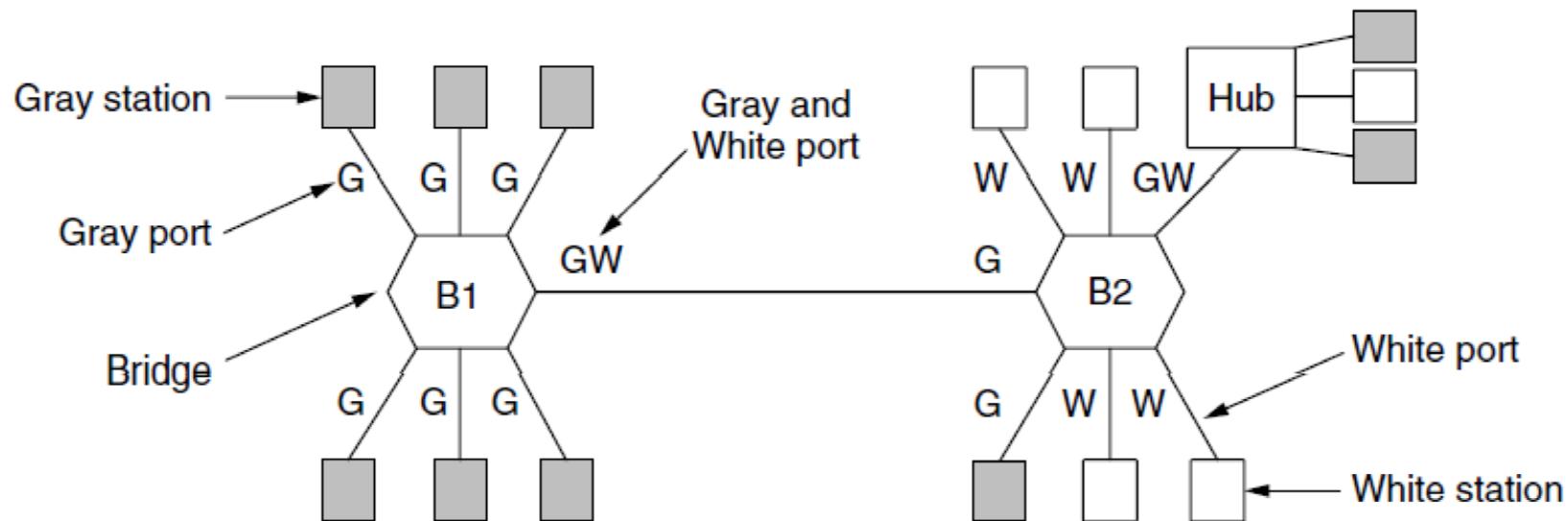
- A bridge or LAN switch operates in the Link layer



Virtual LANs (1)

VLANs (Virtual LANs) splits one physical LAN into multiple logical LANs to ease management tasks

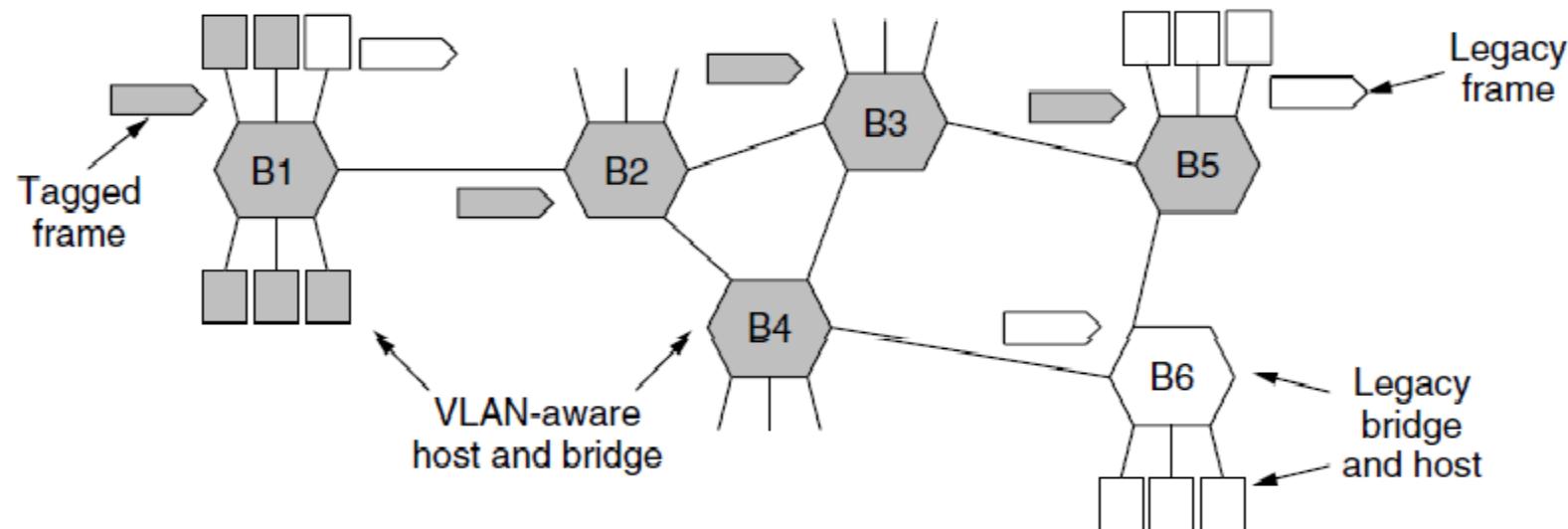
- Ports are “colored” according to their VLAN



Virtual LANs (2) – IEEE 802.1Q

Bridges need to be aware of VLANs to support them

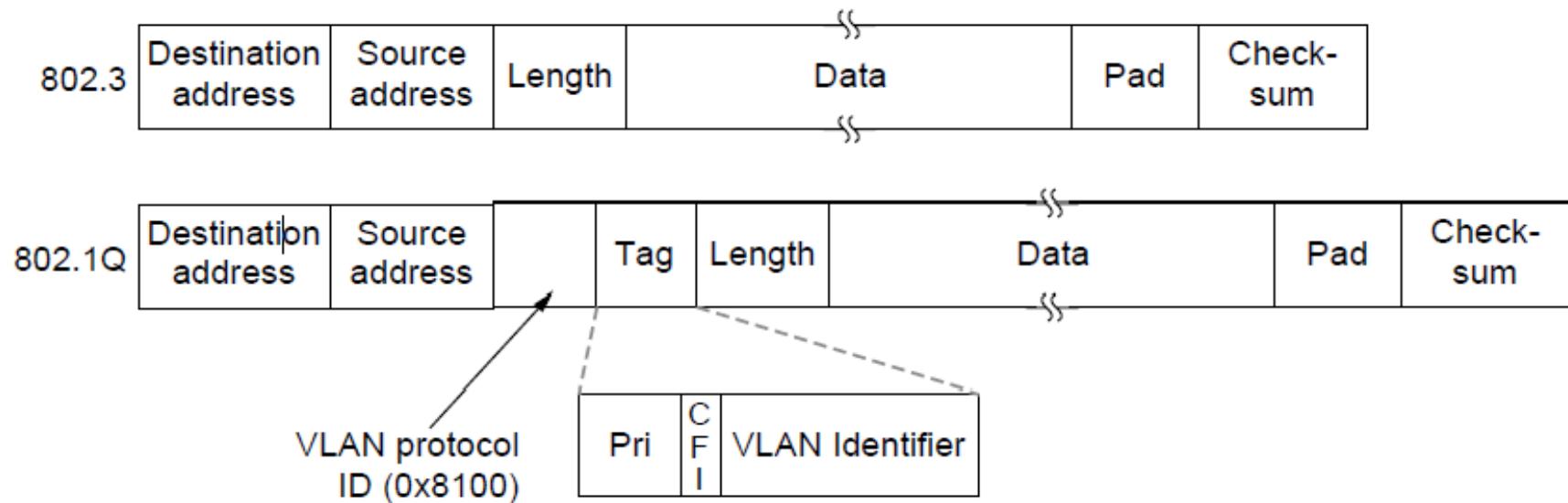
- In 802.1Q, frames are tagged with their “color”
- Legacy switches with no tags are supported



Virtual LANs (3) – IEEE 802.1Q

802.1Q frames carry a color tag (VLAN identifier)

- Length/Type value is 0x8100 for VLAN protocol



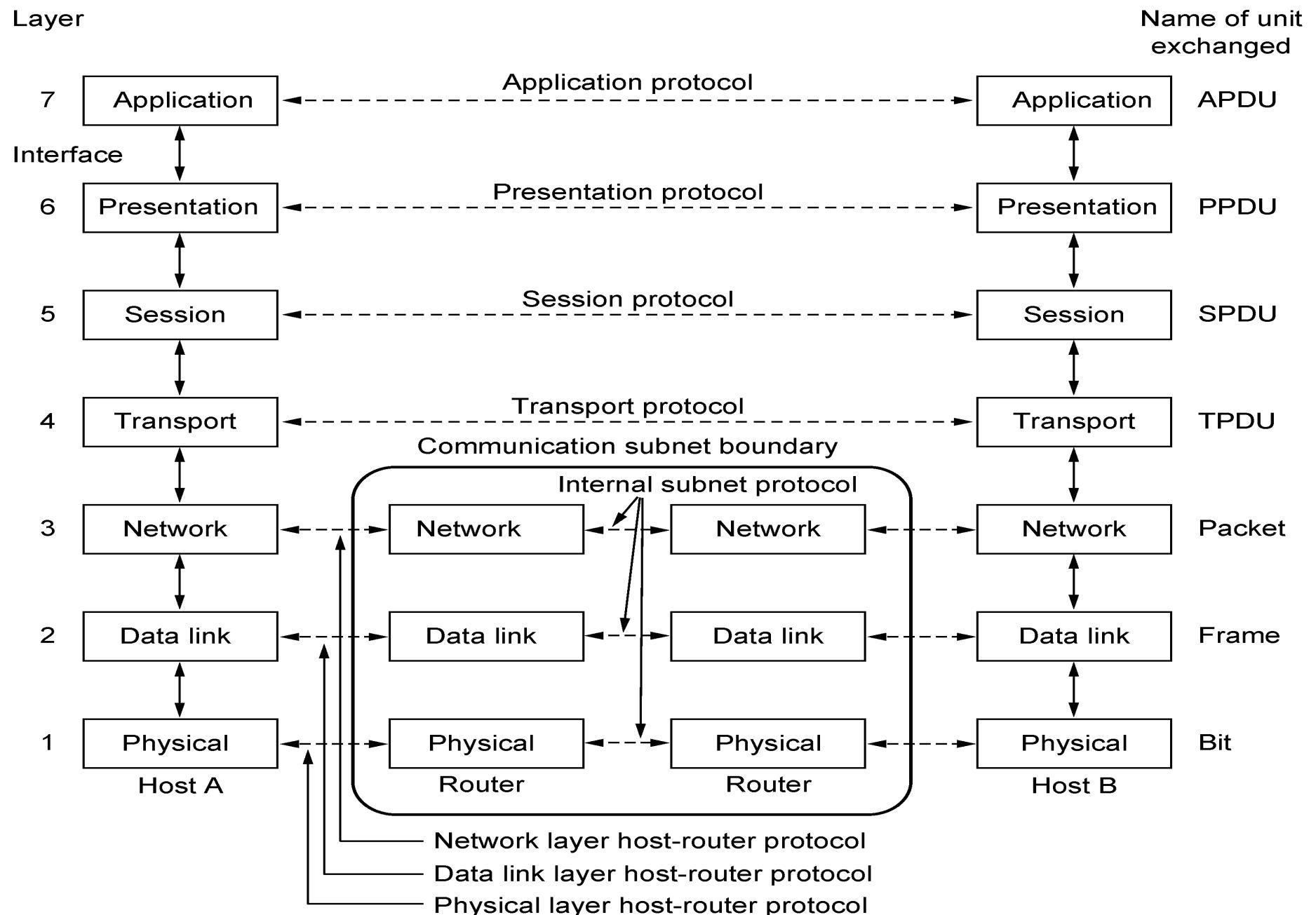
End

Chapter 4

Network Layer

Chapter 5

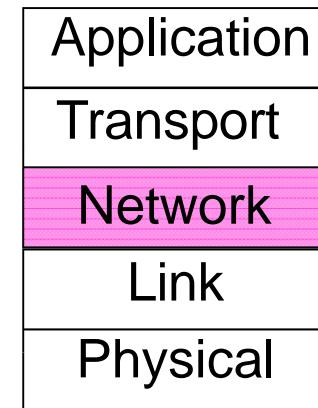
- Responsible for getting packets from the source to destination through **INTERMEDIATE HOPS**
- Data link layer: only between two points
- The network layer deals with end to end transmission
 - Should know the topology of the network
 - Should not overload some routes and underload other routes.
 - Should deal with different networks



The OSI reference model.

The Network Layer

Responsible for delivering packets between endpoints over multiple links



Network Layer

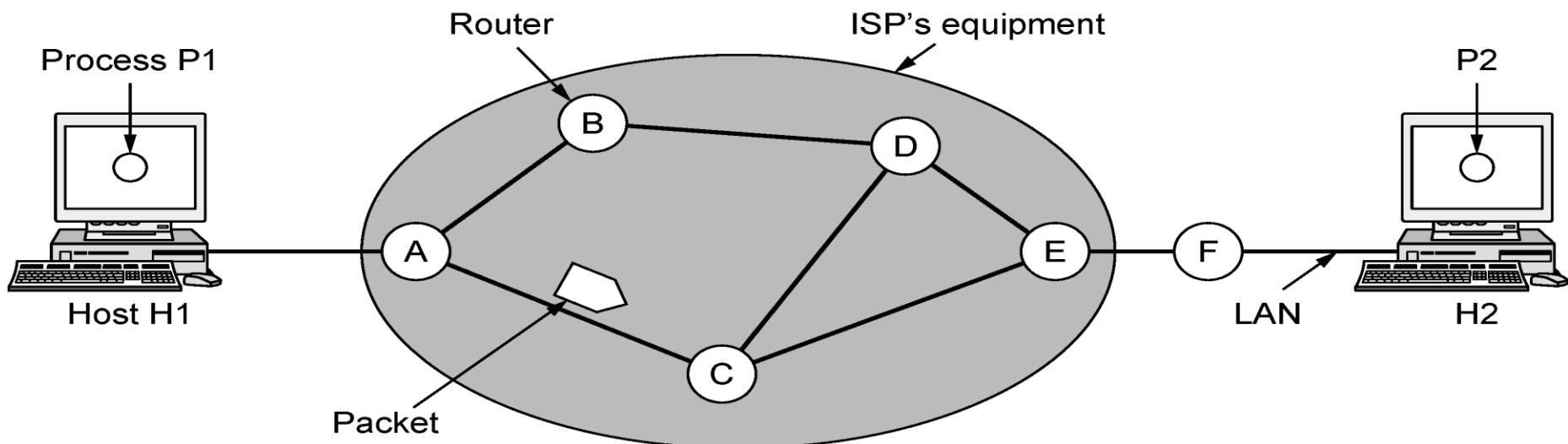
- Design Issues
- Routing Algorithms
- Congestion Control
- Quality of Service
- Internetworking
- Network Layer of the Internet

Design Issues

- Store-and-forward packet switching »
- Connectionless service – datagrams »
- Connection-oriented service – virtual circuits »
- Comparison of virtual-circuits and datagrams »

Store-and-Forward Packet Switching

- Components of the network layer
 - ISP equipment (routers connected by transmission lines)
 - Host H1 is directly connected to one of the router
 - Home computer plugged into MODEM
 - H2 is a LAN, with an office Internet with a router, F, owned and operated by the customer.
 - The router has a leased line to the ISP's equipment.
 - F is not part of ISP, but it runs the same algorithms as the ISP's routers.
- Host sends the packet to the nearest router, which will be forwarded to the next router along the path until it reaches the destination.



The environment of the network layer protocols.

Services Provided to the Transport Layer

- Properties of services
 - The services should be independent of router technology
 - Transport layer should be shielded from the number, type, and topology of the routers present.
 - The network addresses made available to transport layer should use a uniform numbering plan across LANs and WANs.
- The freedom resulted into two thoughts
 - First: Network service should be connectionless with SEND PACKET and RECEIVE PACKET and little more.
 - No ordering and flow control should be done.
 - Each packet should carry a destination address.
 - Second: network should be reliable and connection-oriented service based on the experience of telephone service.
 - Without connections, quality of service is difficult to achieve especially for real-time traffic.
- First, connectionless service has dominated.
- Later, connection-orient service dominated
- Currently, both are widely used.

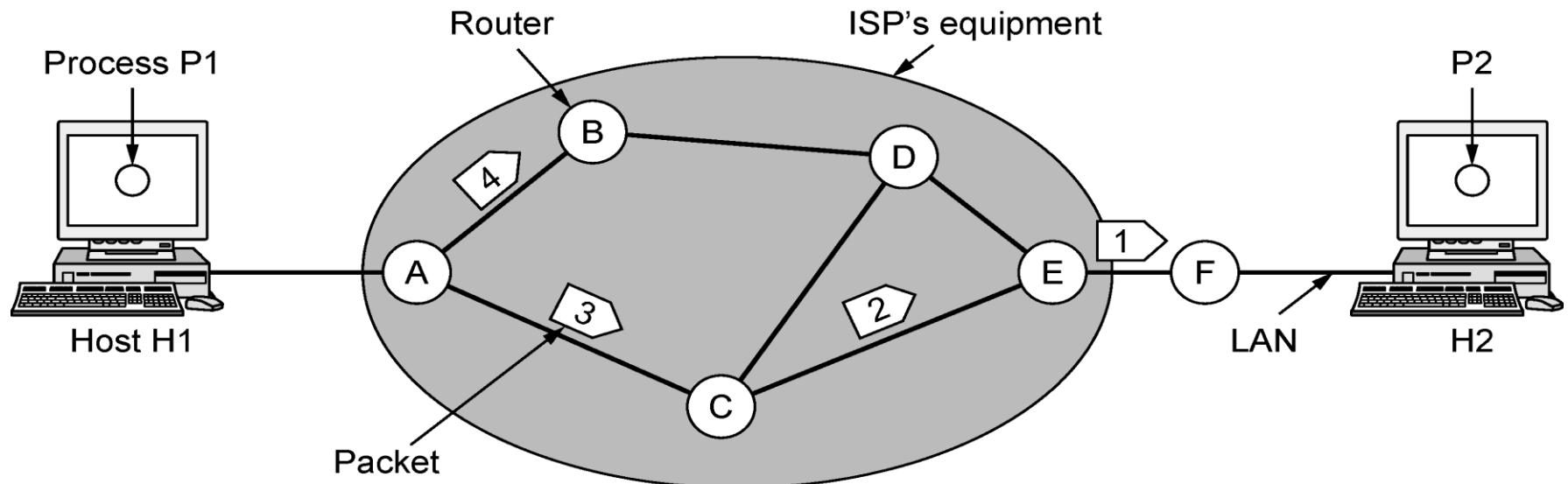
Connectionless service: datagrams

- Packets are injected as datagrams (analogy is telegram).
- No advance setup is needed
- Network is called datagram network.
 - H1 handover long message to transport layer which instructs to deliver packet to P2 on H2. The transport layer runs in H1 (in the operating system). It prepends transport header and handover it to network layer (another procedure)
- Network layer breaks it into packets and sends each of them to router A using point-to-point protocol, PPP.
- ISP decided where to send packets. It has to take routing decisions.
- The algorithm which does routing decisions are called **routing algorithm**.
- In this chapter, we study routing algorithms.
- Internet protocol (IP) is the basis of the Internet, individually forwards each packet. 32 bits in IPv4 and 128 bits in IPv6.

Connectionless Service – Datagrams

Packet is forwarded using destination address inside it

- Different packets may take different paths



A's table (initially) A's table (later)

| | |
|---|---|
| A | - |
| B | B |
| C | C |
| D | B |
| E | C |
| F | C |

Dest. Line

C's table

| | |
|---|---|
| A | A |
| B | A |
| C | - |
| D | E |
| E | E |
| F | E |

E's table

| | |
|---|---|
| A | C |
| B | D |
| C | C |
| D | D |
| E | - |
| F | F |

Routing within a datagram network.

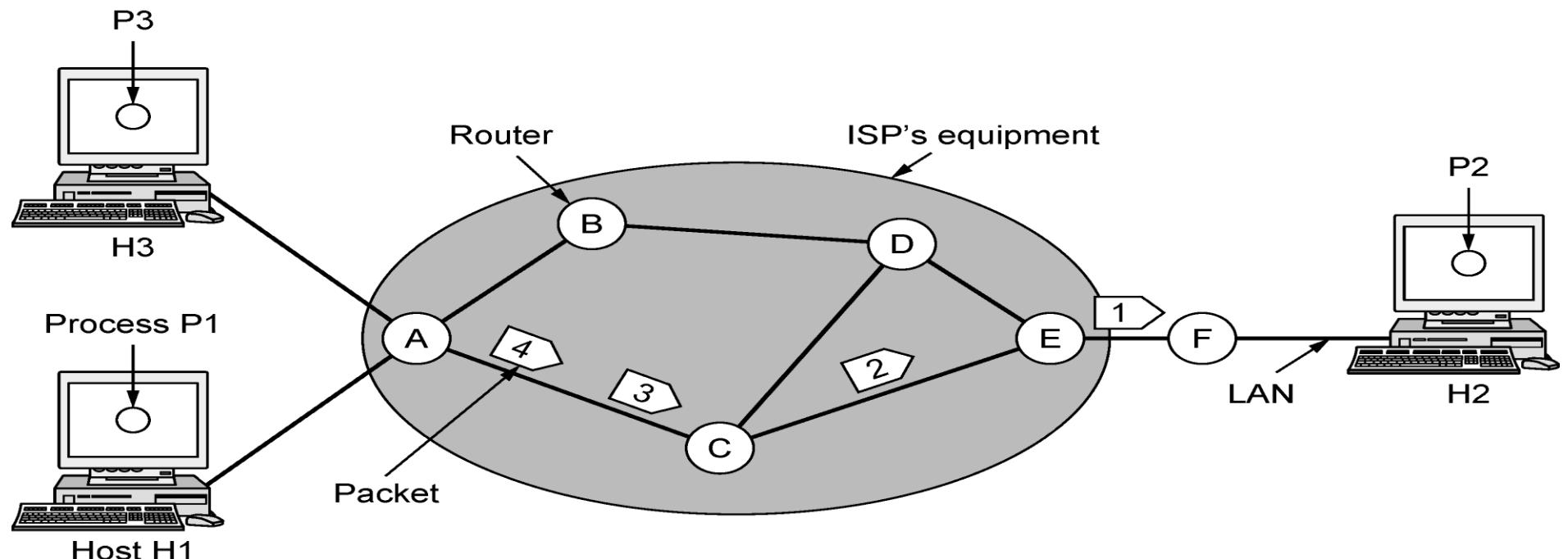
Connection-oriented: Virtual Circuits

- Idea: Avoid having a new route for every packet.
- Establish a connection. When a connection is established, the connection set-up is stored in tables.
 - The route is used for all traffic flowing over the connection.
- H1 established connection 1 with host H2. The first line of A's table says that if a pack arrives with connection 1 it should be forwarded to C and given connection id as 1.
- If H3 also wants to establish a connection to H2, it chooses a connection identifier 1 and tells the network to establish a connection.
- There is a conflict as A has already used id 1. So, A assigns a different identifier. This process is called label switching. An example of the protocol is MPLS (MultiProtocol label Switching). It wraps packets with 20-bit connection identifier or label.

Connection-Oriented – Virtual Circuits

Packet is forwarded along a virtual circuit using tag inside it

- Virtual circuit (VC) is set up ahead of time



| A's table | |
|-----------|-------|
| In | Out |
| H1 1 | C 1 |
| H3 1 | C 2 |

| C's table | |
|-----------|-------|
| In | Out |
| A 1 | E 1 |
| A 2 | E 2 |

| E's table | |
|-----------|-------|
| In | Out |
| C 1 | F 1 |
| C 2 | F 2 |

Routing within a virtual-circuit network.

Comparison of Virtual-Circuits & Datagrams

| Issue | Datagram network | Virtual-circuit network |
|---------------------------|--|--|
| Circuit setup | Not needed | Required |
| Addressing | Each packet contains the full source and destination address | Each packet contains a short VC number |
| State information | Routers do not hold state information about connections | Each VC requires router table space per connection |
| Routing | Each packet is routed independently | Route chosen when VC is set up; all packets follow it |
| Effect of router failures | None, except for packets lost during the crash | All VCs that passed through the failed router are terminated |
| Quality of service | Difficult | Easy if enough resources can be allocated in advance for each VC |
| Congestion control | Difficult | Easy if enough resources can be allocated in advance for each VC |

Routing Algorithms (1)

- Optimality principle »
- Shortest path algorithm »
- Flooding »
- Distance vector routing »
- Link state routing »
- Hierarchical routing »
- Broadcast routing »
- Multicast routing »
- Anycast routing »
- Routing for mobile hosts »
- Routing in ad hoc networks »

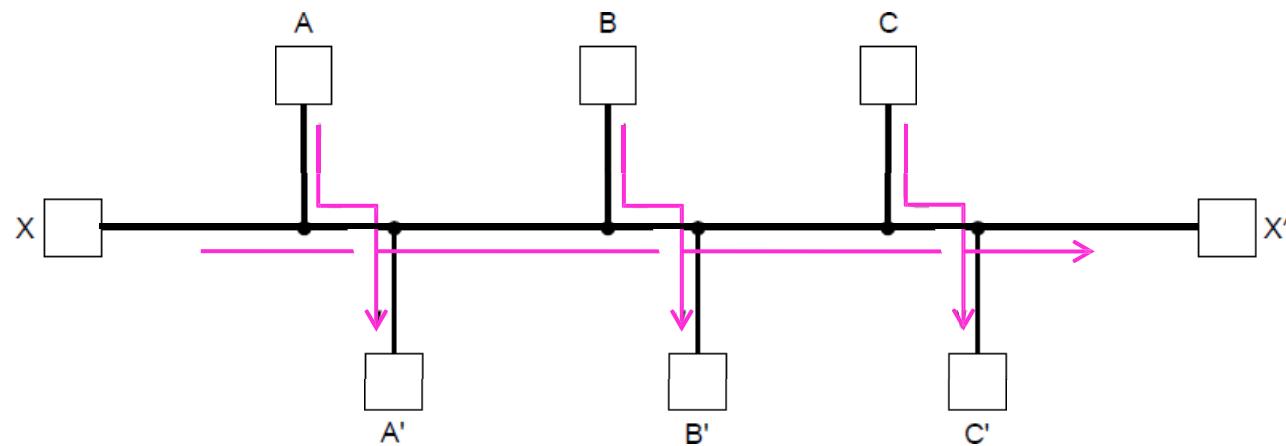
Routing Algorithms

- The decision has to be made for every packet as best route may be changed over last time.
- Network layer design
 - Algorithms and data structures
- Routing and forwarding
 - Forwarding: a process which takes each packet and forwards it to outgoing line.
 - Routing: A process which updates the routing tables.
 - Routing algorithms play a major role.
- Properties of routing algorithm
 - Correctness, simplicity:
 - Robustness: it should run for years; hosts fail, routers crash, software fails
 - Stability: Routing algorithm should converge and reach equilibrium.
 - fairness, efficiency: Difficult to achieve: For example, in case of high traffic between A and A', B and B', C and C', the traffic between X and X' will be shut off. So compromise is required.

Routing Algorithms (2)

Routing is the process of discovering network paths

- Model the network as a graph of nodes and links
- Decide what to optimize (e.g., fairness vs efficiency)
- Update routes for changes in topology (e.g., failures)



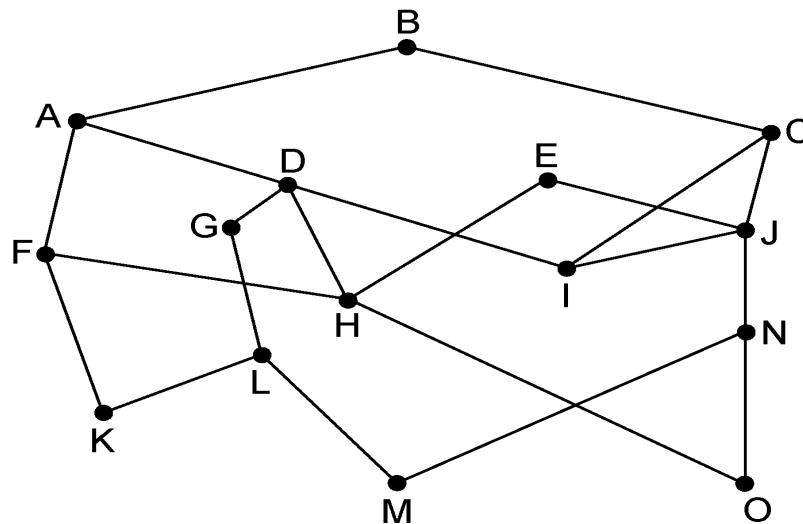
Forwarding is the sending of packets along a path

Routing Algorithms

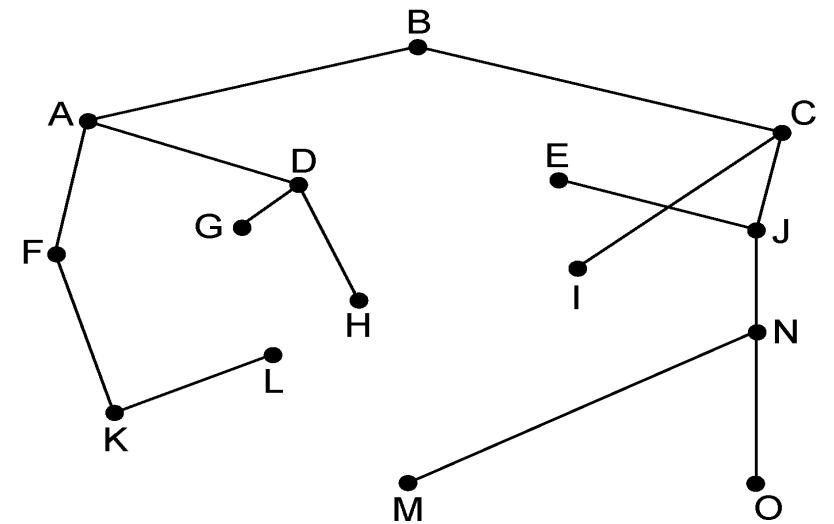
- Nonadaptive algorithms
 - Decisions do not depend on the state or topology
 - When the network is booted, routing tables are fixed.
 - Also called static routing algorithms
- Adaptive algorithms
 - Decisions are based on the state or topology
 - Dynamic routing algorithms

The Optimality Principle

- Each portion of a best path is also a best path; the union of them to a router is a tree called the sink tree
- Best means fewest hops in the example
- Set of all optimal routes from all sources to a given destination form a **sink tree**
- Goal of routing algorithm: discover and use sink trees for all the routers.
- The optimality principle acts as a Benchmark for comparison



(a)



(b)

(a) A network. (b) A sink tree for router B.

Shortest Path Algorithm (1)

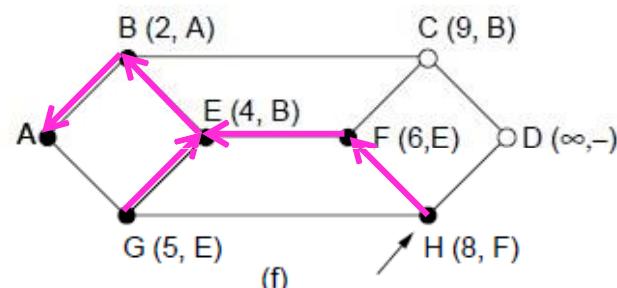
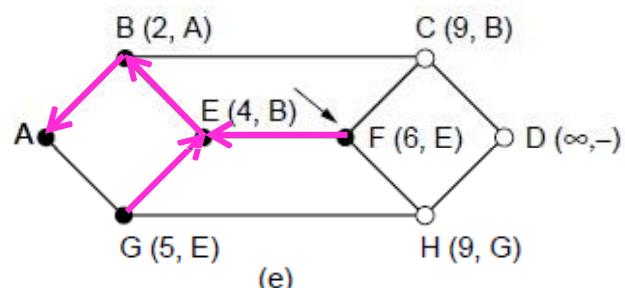
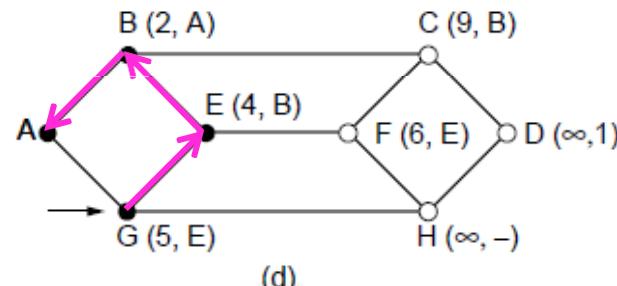
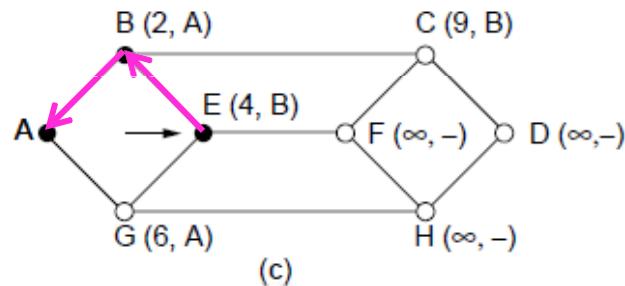
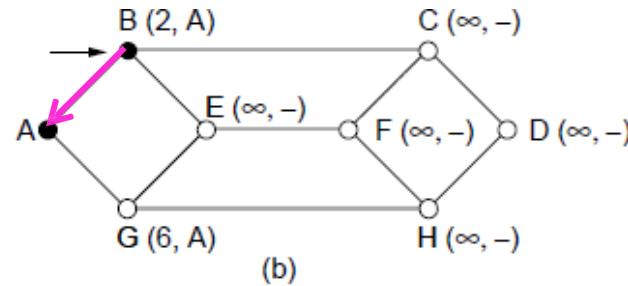
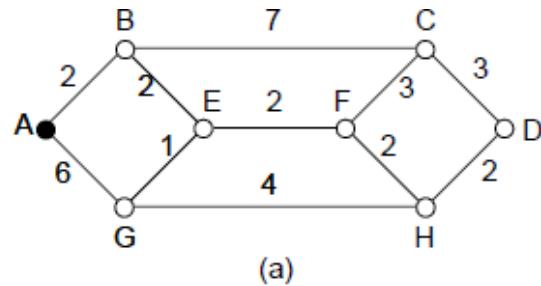
Dijkstra's algorithm computes a sink tree on the graph:

- Each link is assigned a non-negative weight/distance
- Shortest path is the one with lowest total weight
- Using weights of 1 gives paths with fewest hops

Algorithm:

- Start with sink, set distance at other nodes to infinity
- Relax distance to other nodes
- Pick the lowest distance node, add it to sink tree
- Repeat until all nodes are in the sink tree

Shortest Path Algorithm (2)



A network and first five steps in computing the shortest paths from A to D. Pink arrows show the sink tree so far.

Shortest Path Algorithm (3)

```
...  
for (p = &state[0]; p < &state[n]; p++) {  
    p->predecessor = -1;  
    p->length = INFINITY;  
    p->label = tentative;  
}  
state[t].length = 0; state[t].label = permanent;  
k = t;  
do {  
    for (i = 0; i < n; i++)  
        if (dist[k][i] != 0 && state[i].label == tentative) {  
            if (state[k].length + dist[k][i] < state[i].length) {  
                state[i].predecessor = k;  
                state[i].length = state[k].length + dist[k][i];  
            }  
        }  
    ...  
}
```

} Start with the sink, all other nodes are unreachable

} Relaxation step. Lower distance to nodes linked to newest member of the sink tree

Shortest Path Algorithm (4)

```
    . . .
    k = 0; min = INFINITY;
    for (i = 0; i < n; i++)
        if (state[i].label == tentative && state[i].length < min) {
            min = state[i].length;
            k = i;
        }
    state[k].label = permanent;
} while (k != s);
```

Find the lowest distance, add it to the sink tree, and repeat until done

Flooding

- A simple method to send a packet to all network nodes
 - Each node floods a new packet received on an incoming link by sending it out all of the other links
- Generates vast numbers of duplicate packets.
- Method 1:
 - Flooding with an hop count: Put hop counter and decrement hop counter for each node.
 - -ve: generates exponential number of duplicate packets
- Method 2
 - Track the packets. Stop, if it is already flooded, i.e., avoid sending second time.
 - Put a sequence number for each source in the packet. Maintain the list at each node.

Flooding

- Flooding is not practical, but has several advantages
 - It ensures that packet reaches every node.
 - Flooding is tremendously robust.
 - Even if large number of routers are damaged, the packet reaches the destination.
 - Flooding produces shorter delay
- Flooding can be used to compare other algorithms.

Distance Vector Routing (1)

- Dynamic
- Distance vector is a distributed routing algorithm
 - Shortest path computation is split across nodes
 - Each router maintains the table giving the best known distance to each destination

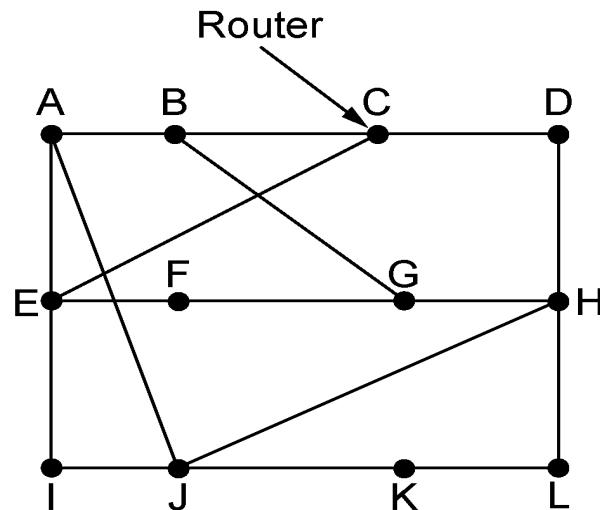
Bellman-Ford Algorithm (followed in ARPANET):

- Each node knows the distance of links to its neighbors
- Each node advertises vector of lowest known distances to all neighbors
- Each node uses received vectors to update its own
- Repeat periodically

Distance Vector Routing (1)

- Once every T sec, each router sends to each neighbor a list of estimated delays to the destination. It also receives similar list from its neighbors and calculated the new routing table.
- Example
 - Four columns show the delay vectors received from neighbors. A claims to have a 12 msec delay to B, 25msec delay to C, and 40 sec delay to D and so on.
 - Suppose, J has measured or estimated its delay to its neighbors A, I,H, and K as 8, 10, 12, and 6 respectively.
 - J computes time delay to G as follows.
 - One route: $J, A, \dots, G = 8 (JA) + 18 (AG) = 26$
 - Second route: $J \ I \ \dots \ G = 41 (31+10)$
 - Third route: $J \ H \ \dots \ G = 18 (6+12)$
 - Fourth route: $J \ K \ \dots \ G = 37 (31+6)$
 - The best route is 18.

Distance Vector Routing (2)



(a)

New estimated delay from J

| To | A | I | H | K | Line |
|----|----|----|----|----|--------|
| A | 0 | 24 | 20 | 21 | 8 A |
| B | 12 | 36 | 31 | 28 | 20 A |
| C | 25 | 18 | 19 | 36 | 28 I |
| D | 40 | 27 | 8 | 24 | 20 H |
| E | 14 | 7 | 30 | 22 | 17 I |
| F | 23 | 20 | 19 | 40 | 30 I |
| G | 18 | 31 | 6 | 31 | 18 H |
| H | 17 | 20 | 0 | 19 | 12 H |
| I | 21 | 0 | 14 | 22 | 10 I |
| J | 9 | 11 | 7 | 10 | 0 - |
| K | 24 | 22 | 22 | 0 | 6 K |
| L | 29 | 33 | 9 | 9 | 15 K |

JA delay is 8 JI delay is 10 JH delay is 12 JK delay is 6

Vectors received from J's four neighbors

New routing table for J

(b)

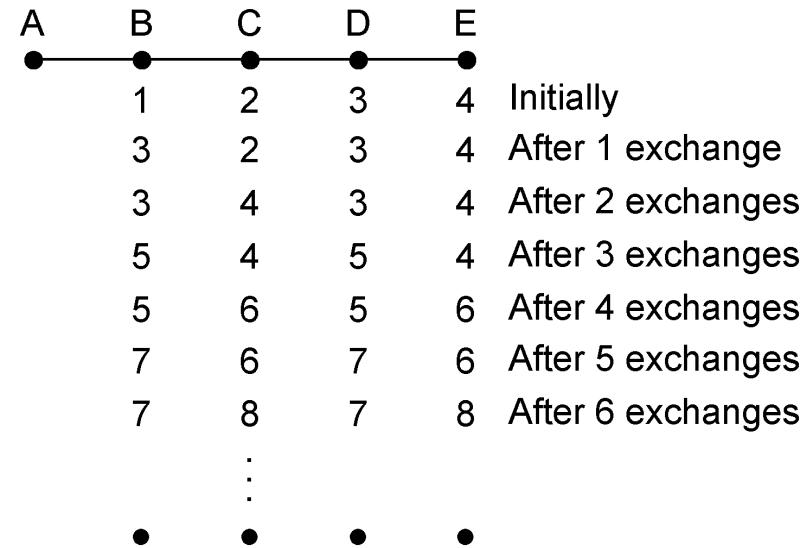
(a) A network. (b) Input from *A, I, H, K*, and the new routing table for *J*.

The Count-to-Infinity Problem

- Convergence: Though it converges to correct answer, it converges slowly.
- Responds rapidly to good news, but very slowly to bad news.
- In Figure (a) shows how good news spreads quickly.
- Figure (b) demonstrates count to infinity problem.



(a)



(b)

The count-to-infinity problem.

Link State Routing (1)

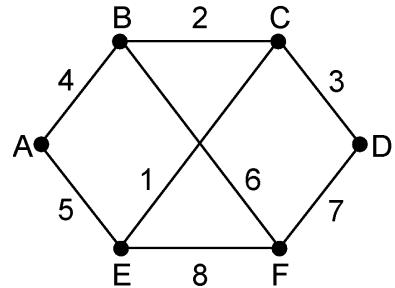
- Distance vector routing is replaced by link state routing.
- More computation but simpler dynamics
- Widely used in the Internet (OSPF, ISIS)

Algorithm: Each router

1. Discovers its neighbors and learn their network addresses.
 - After booting, node sends HELLO packet to each point-to-point line.
2. Set the distance or cost metric to each of its neighbors
3. Construct a packet with the learning.
4. Send this packet and receive packet from all other routers.
5. Compute the shortest path to every other router

Link State Routing (2) – LSPs

LSP (Link State Packet) for a node lists neighbors and weights of links to reach them



(a)

| Link | State | Packets |
|-------|-------|---------|
| A | B | E |
| Seq. | Seq. | Seq. |
| Age | Age | Age |
| B 4 | B 2 | A 5 |
| A 4 | C 3 | C 1 |
| C 2 | D 3 | F 8 |
| F 6 | F 7 | F 8 |
| E 5 | E 1 | E 8 |

(b)

(a) A network. (b) The link state packets for this network.

Link State Routing (3) – Reliable Flooding

Seq. number and age are used for reliable flooding

- New LSPs are acknowledged on the lines they are received and sent on all other lines
- Example shows the LSP database at router B

| Source | Seq. | Age | Send flags | | | ACK flags | | | Data |
|--------|------|-----|------------|---|---|-----------|---|---|------|
| | | | A | C | F | A | C | F | |
| A | 21 | 60 | 0 | 1 | 1 | 1 | 0 | 0 | |
| F | 21 | 60 | 1 | 1 | 0 | 0 | 0 | 1 | |
| E | 21 | 59 | 0 | 1 | 0 | 1 | 0 | 1 | |
| C | 20 | 60 | 1 | 0 | 1 | 0 | 1 | 0 | |
| D | 21 | 59 | 1 | 0 | 0 | 0 | 1 | 1 | |

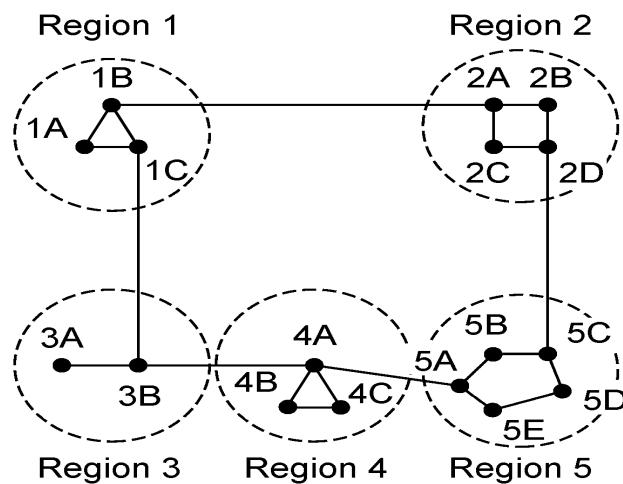
The packet buffer for router B in Fig. 5-12(a).

Hierarchical Routing

- As network grows, the router tables grow proportionately.
 - More memory and CPU time
 - More bandwidth
- After certain limit, it is not possible to send routing table to every other router.
- Hierarchical routing
 - Routers are divided into regions.
 - Each router knows how to route packets to other routers within the region.
 - Hierarchy may consists of multiple levels.

Hierarchical Routing

Hierarchical routing reduces the work of route computation but may result in slightly longer paths than flat routing



(a)

| Full table for 1A | | |
|-------------------|------|------|
| Dest. | Line | Hops |
| 1A | — | — |
| 1B | 1B | 1 |
| 1C | 1C | 1 |
| 2A | 1B | 2 |
| 2B | 1B | 3 |
| 2C | 1B | 3 |
| 2D | 1B | 4 |
| 3A | 1C | 3 |
| 3B | 1C | 2 |
| 4A | 1C | 3 |
| 4B | 1C | 4 |
| 4C | 1C | 4 |
| 5A | 1C | 4 |
| 5B | 1C | 5 |
| 5C | 1B | 5 |
| 5D | 1C | 6 |
| 5E | 1C | 5 |

(b)

| Hierarchical table for 1A | | |
|---------------------------|------|------|
| Dest. | Line | Hops |
| 1A | — | — |
| 1B | 1B | 1 |
| 1C | 1C | 1 |
| 2 | 1B | 2 |
| 3 | 1C | 2 |
| 4 | 1C | 3 |
| 5 | 1C | 4 |

(c)

Hierarchical routing.

Broadcast routing

- Broadcast sends a packet to all nodes
- Simple algorithm: send the packet to each destination.
 - It is slow and wastes bandwidth
 - Source should have a list of all destinations.
- Multi-destination routing
 - Each packet contains a list of destinations or a bitmap indicating desired destinations.
 - The router generates new copy of the packet for each output line and includes in the packet only those destinations which are on the line.
 - After sufficient number of hops the packet will have only one destination
 - Bandwidth is used efficiently.

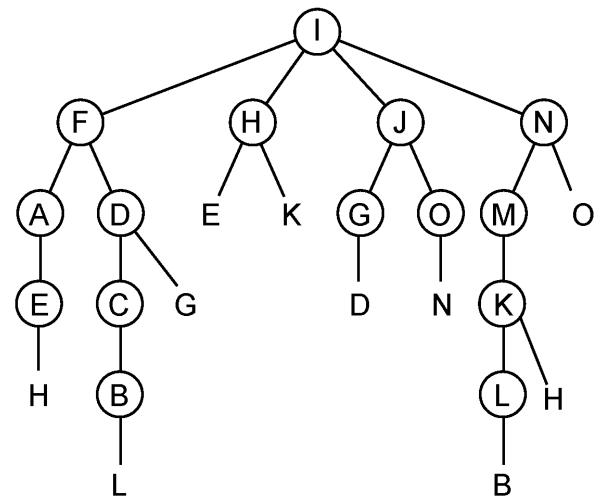
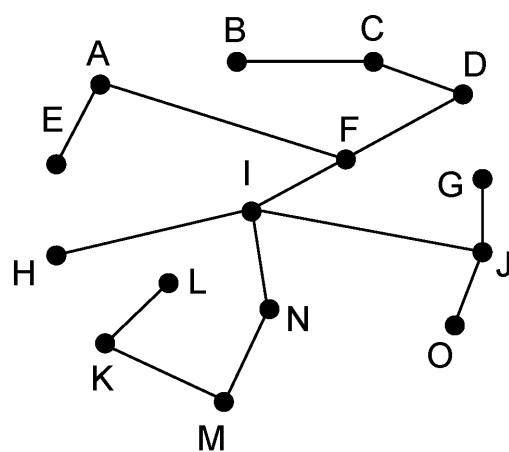
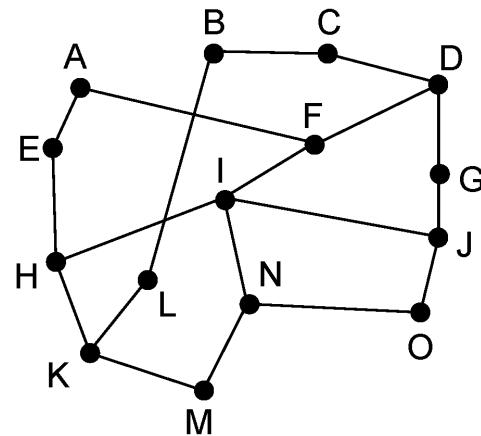
Broadcast routing

- Flooding is another broad cast technique
- Reverse path forwarding (figure (c))
- When a broadcast packet arrives to the router, the router checks to see if the packet arrived on the link that is normally used for sending packets toward the source of the broadcast.
 - If so, there is a chance that broadcast packet might have followed the excellent route.
 - The router forward the packet to all links except incoming links.
 - If the packet arrives from other link than the preferred one, the pack is discarded as a likely duplicate.
 - After 5 hops, with 24 packets broadcasting terminates.
 - +ve: easy to implement and efficient; no sequence numbers are required.
- Spanning tree algorithm: figure (b)
 - Minimum of 14 packets
 - -ve: each router should have knowledge of spanning tree

Broadcast Routing

Broadcast sends a packet to all nodes

- RPF (Reverse Path Forwarding): send broadcast received on the link to the source out all remaining links
- Alternatively, can build and use sink trees at all nodes



(a)

(b)

(c)

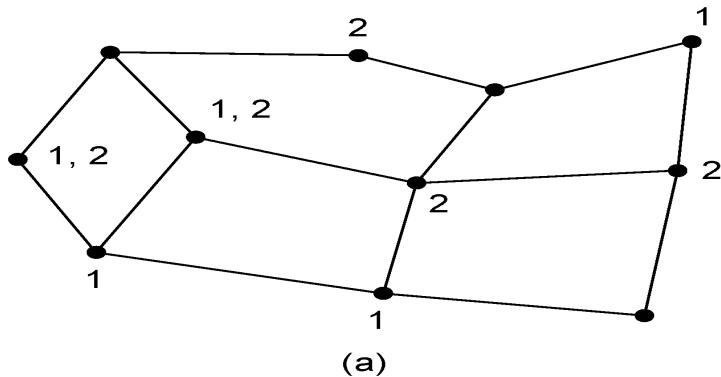
Reverse path forwarding. (a) A network. (b) Sink tree for router *I*. (c) The tree built by reverse path forwarding from *I*.

Multicast Routing (1) – Dense Case

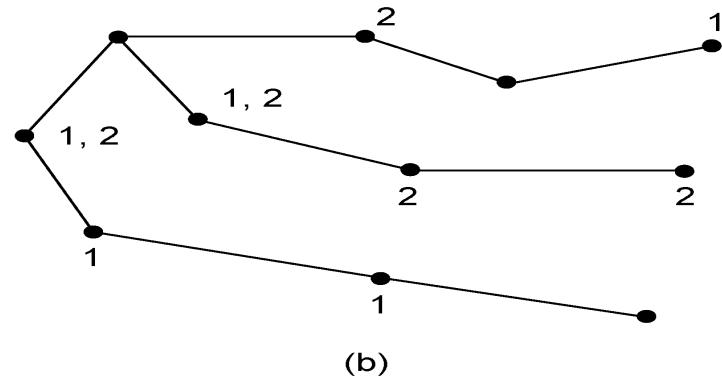
- Send a message to well-defined group, numerically large by small than the size of the network
 - Example: live video
- Multicast sends to a subset of the nodes called a group
- Routing algorithm: multicast routing
- All multicasting schemes require a way to create and destroy groups and identify which routers are the members of the group.

Multicast Routing (1) – Dense Case

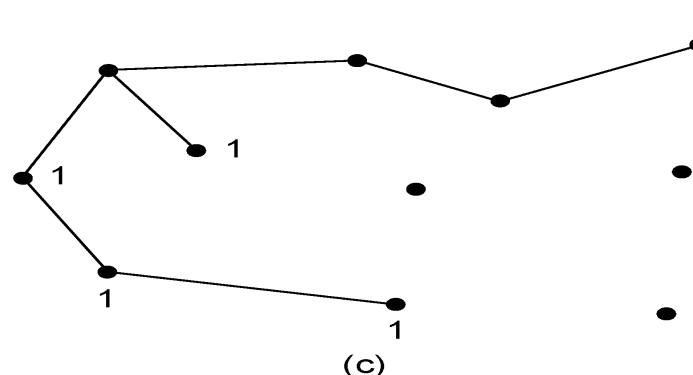
- If the group is dense, employ broadcasting
 - Remove those nodes which are not the members in the spanning tree.
 - Multicasting spanning tree
 - Use link state routing to identify the nodes.



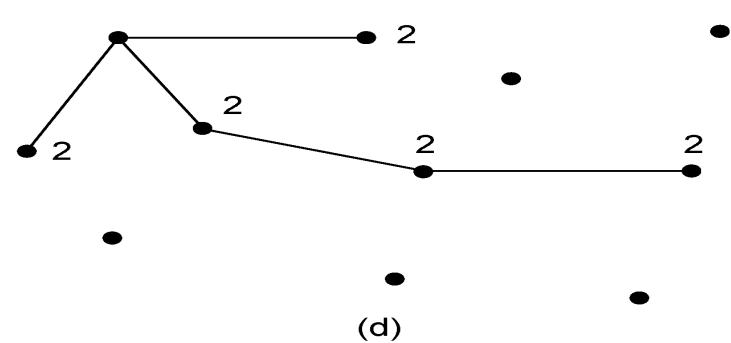
(a)



(b)



(c)



(d)

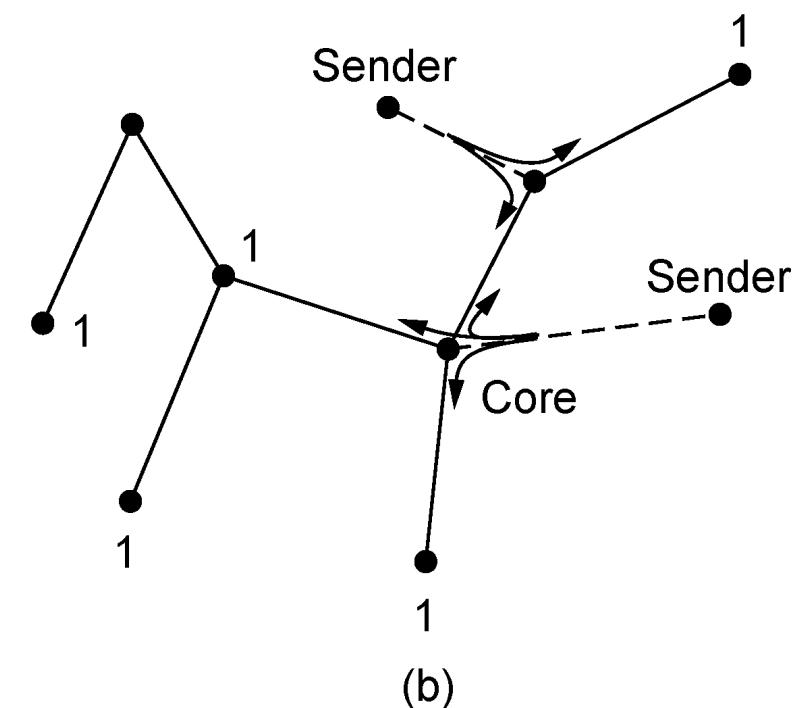
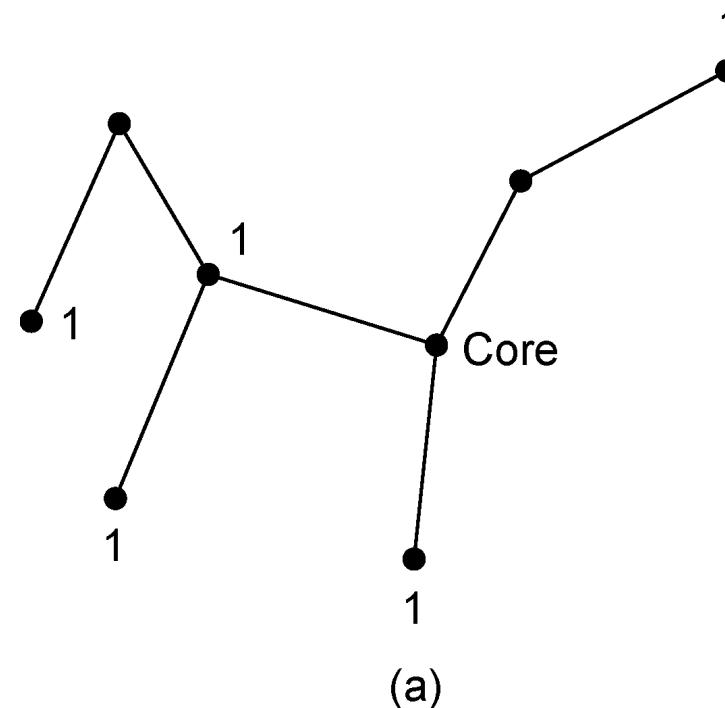
(a) A network. (b) A spanning tree for the leftmost router.

(c) A multicast tree for group 1. (d) A multicast tree for group 2.

Multicast Routing (2) – Sparse Case

CBT (Core-Based Tree) uses a single tree to multicast

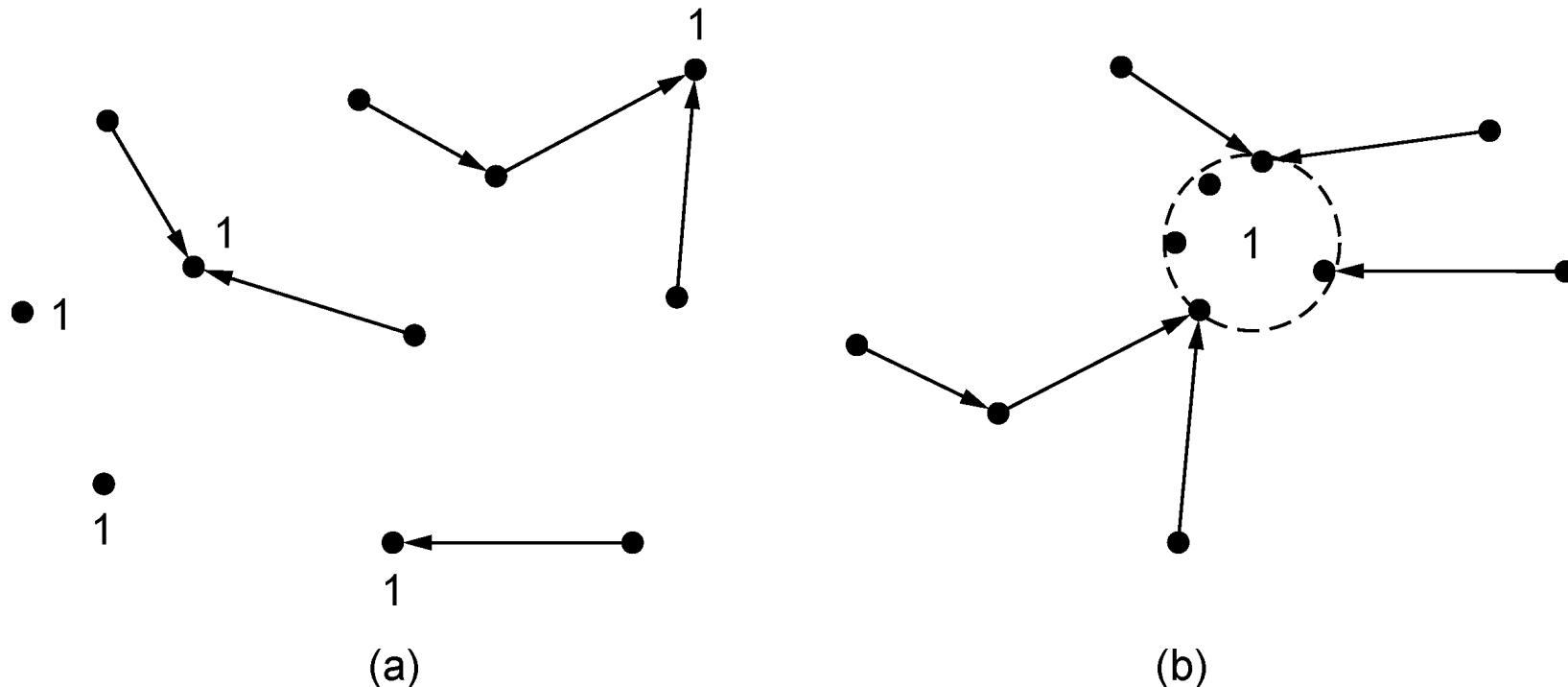
- All the members agree on a tree.
- Tree is the sink tree from core node to group members
- Multicast heads to the core until it reaches the CBT



(a) Core-based tree for group 1. (b) Sending to group 1.

Anycast Routing

- Anycast sends a packet to one (nearest) group member
 - Used for time of the day, DNS service
 - Falls out of regular routing with a node in many places

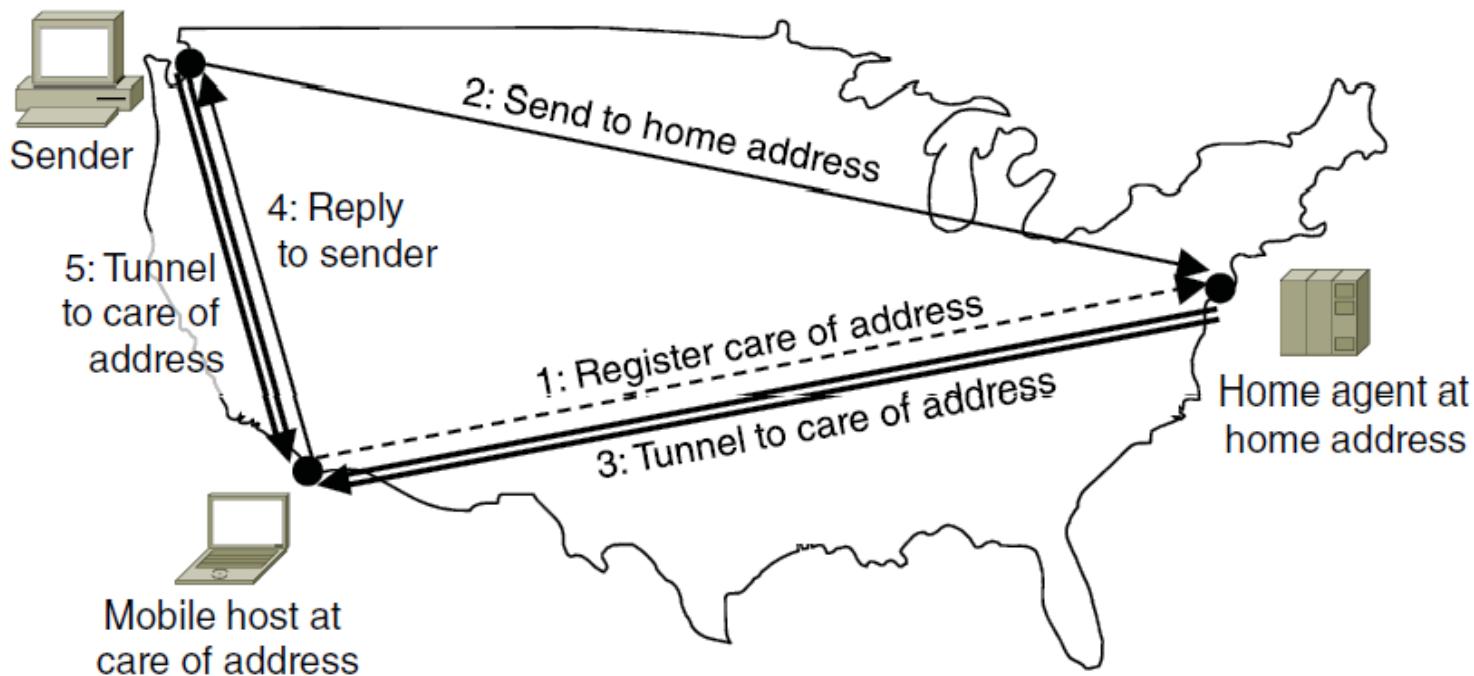


(a) Anycast routes to group 1. (b) Topology seen by the routing protocol.

Routing for Mobile Hosts

Mobile hosts can be reached via a home agent

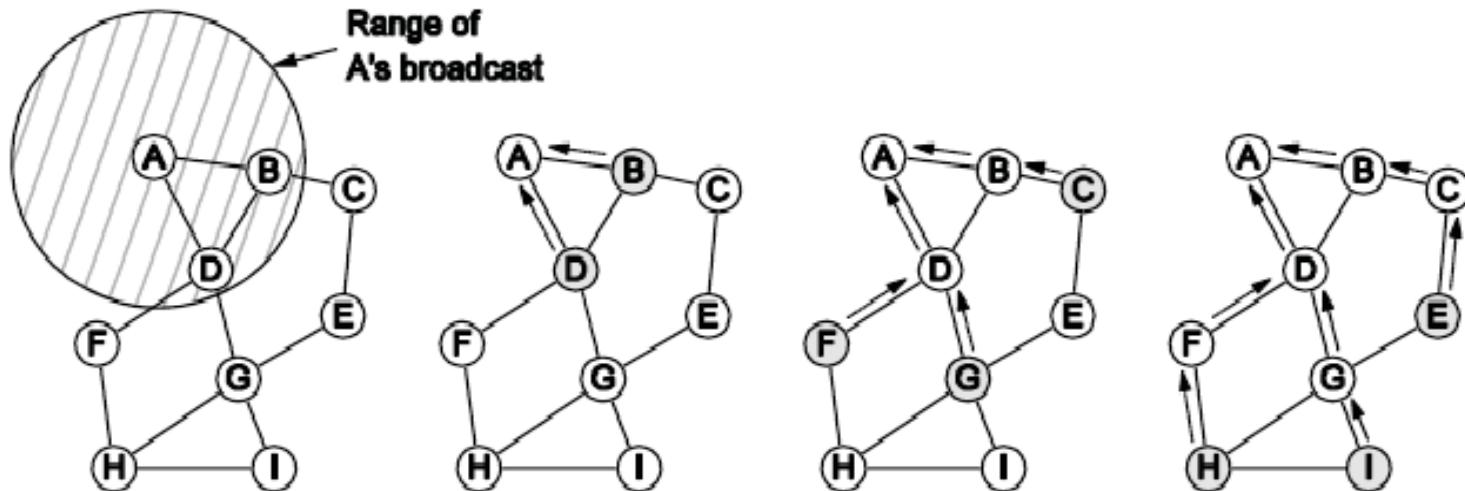
- Fixed home agent tunnels packets to reach the mobile host; reply can optimize path for subsequent packets
- No changes to routers or fixed hosts



Routing in Ad Hoc Networks

The network topology changes as wireless nodes move

- Routes are often made on demand.



A's starts to
find route to I

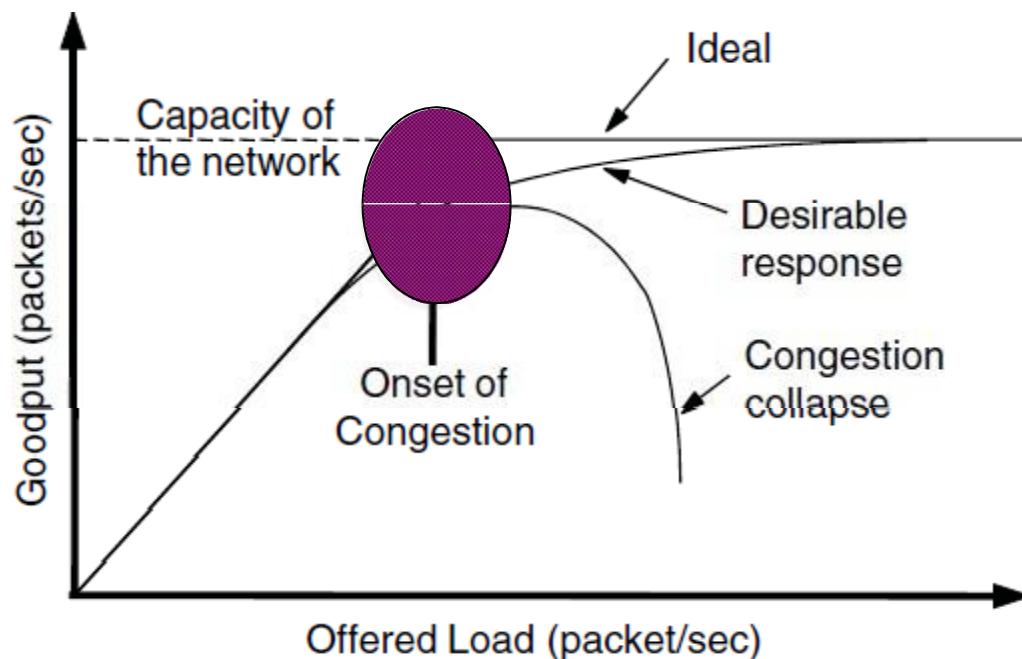
A's broadcast
reaches B & D

B's and D's
broadcast
reach C, F & G

C's, F's and G's
broadcast
reach H & I

Congestion Control (1)

- Congestion is a situation: Too many packets present in the network causes network delay and loss
- Both network and transport layers share the responsibility to control congestion.
- Effective way is to reduce the load at the transport layer.
- Goodput (=useful packets) trails offered load



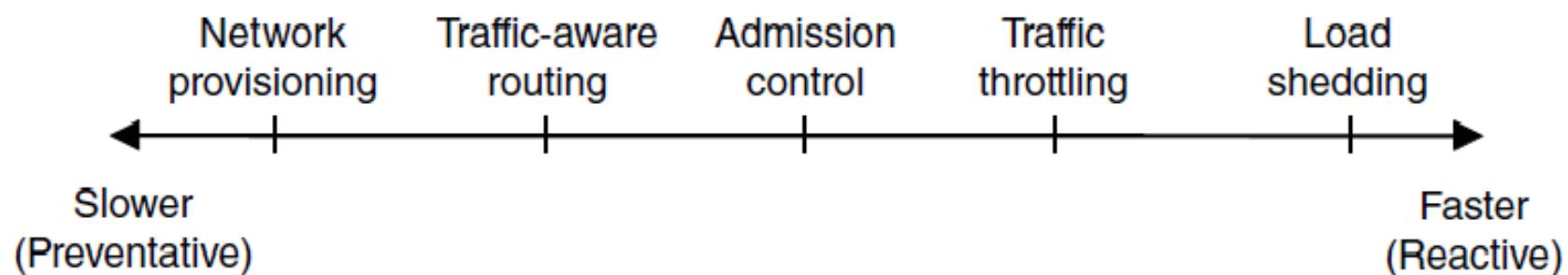
Congestion Control (2)

- Network experiences a congestion collapse
- If routers will have infinite memory, congestion gets worse
 - Packets gets timed out.
- Difference between congestion control and flow control
- Congestion control is related to whole network.
Involves the behavior of all the hosts and routers.
- Flow control relates to the traffic between a particular sender and a particular receiver.
- Best method is to get host a “slow down”.

Congestion Control (3) – Approaches

Network must do its best with the offered load

- Different approaches at different timescales
- Nodes should also reduce offered load (Transport)



Congestion Control (1)

Handling congestion is the responsibility of the Network and Transport layers working together

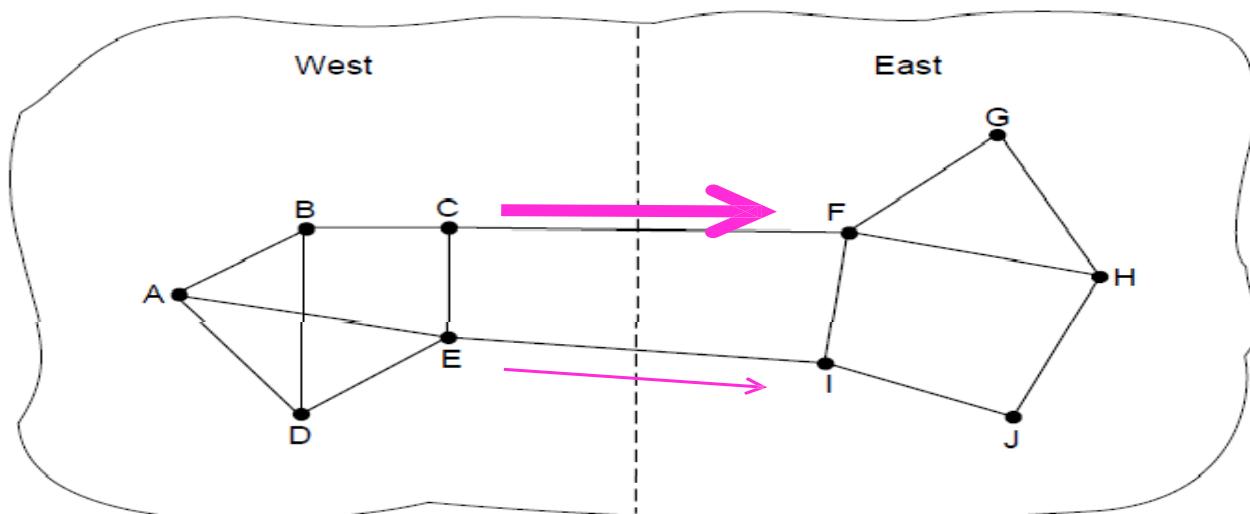
- We look at the Network portion here
- Traffic-aware routing »
- Admission control »
- Traffic throttling »
- Load shedding »

Network provisioning

- Links and routers that are heavily utilized are upgraded at the earliest opportunity.
 - By considering short-term and long-term traffic trends.

Traffic-Aware Routing

- Choose routes depending on traffic, not just topology
- Link weight is a function of the link bandwidth, propagation delay, measured load (average queuing delay)
- Routing tables are updated accordingly.
- E.g., use *EI* for West-to-East traffic if *CF* is loaded
- -ve: routing tables may oscillate wildly leading to erratic routing and other problems

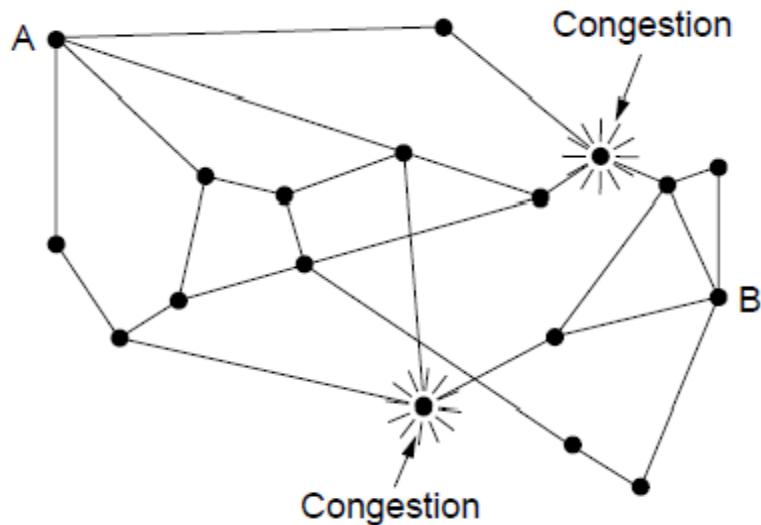


Admission Control

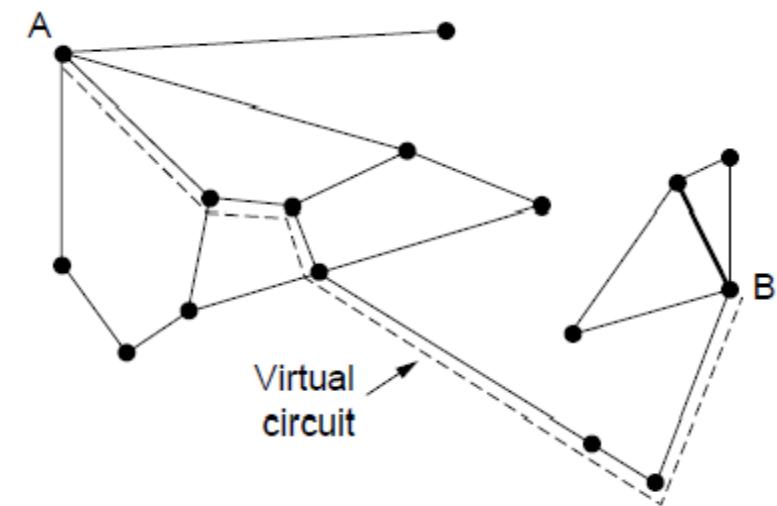
- Idea: do not set up a new virtual circuit unless the network can carry the added traffic without becoming congested.
 - Attempt to set up a virtual circuit may fail.
 - Example: telephone system
- Issue is handling **bursty traffic**
 - Video is easy to handle
 - Traffic due to web browsing is very bursty
- Admission control can be combined with traffic ware routing
- Redraw the network topology based on information about congested routers.
 - For example, normally connection between router A and B passes through one of the congested routers. Based on the information, we can redraw the network.

Admission Control

- Admission control allows a new traffic load only if the network has sufficient capacity, e.g., with virtual circuits
- Can combine with looking for an uncongested route



Network with some congested nodes



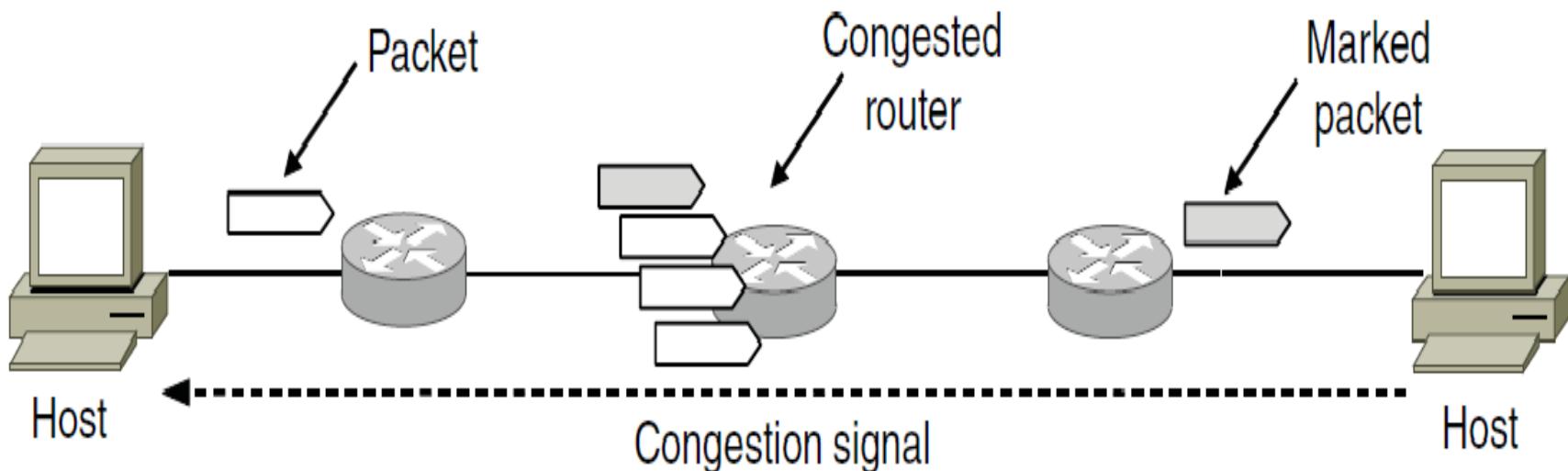
Uncongested portion and route *AB* around congestion

Traffic Throttling

- When the congestion is imminent, network should tell the senders to throttle back their transmissions and slow down.
- Congestion avoidance
- Congested routers signal hosts to slow down traffic for Datagram networks and virtual circuit networks.
- Two problems have to be solved for each approach
- First: router must determine when congestion is approaching, before it has arrived.
 - Queuing delay captures congestion experienced by packets
- Second: routers must send timely feedback to the senders that are causing congestion.
 - ECN (Explicit Congestion Notification) marks packets and receiver returns signal to sender

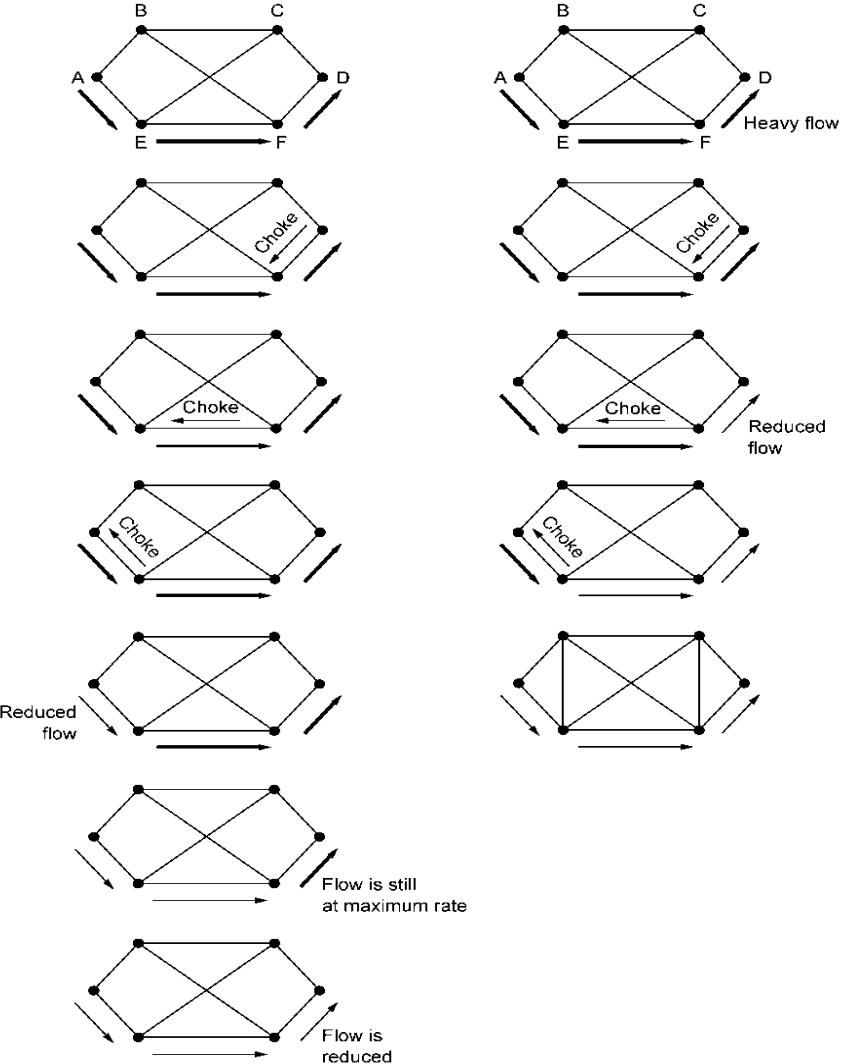
Traffic Throttling: feedback mechanisms

- Choke packets
 - Router selects the congested packet and sends the choked packet back to source host.
 - When a source gets the choked packet, it is supposed to reduce the 50% of traffic.
- Explicit congestion notification (ECN)
 - The router can tag any packet it forwards to signal that it is experiencing a congestion.
 - When a network delivers a packet, the receiver can inform the sender about congestion through ack. message. The design is called Explicit Congestion Notification (ECN).



Traffic Throttling: feedback mechanisms

- With ECN method, many new packets may be transmitted after the congestion.
- Alternative approach: Hop-by-Hop Backpressure
 - Have the choke packet take effect at every hop it passes through
 - Whenever a packet reaches F, F is required to reduce the flow to D.
- Example:
 - As soon as choke packet reaches F, F is required to reduce the flow to D.
 - It gives the D immediate relief
 - Similarly when a choke packet reaches E, E has to reduce the flow to F.



(a) A choke packet that affects only the source. (b) A choke packet that affects each hop it passes through.

Load Shedding (1)

- When all else fails, network will drop packets (shed load)
- When routers are being inundated by packets that they can not handle, they just throw them away.
- Key question
 - Which packets to drop?
 - File transfer: old packet is important
 - WINE policy
 - Realtime traffic: new packet is important
 - MILK policy
- Application must mark the importance on the packets.
- **Random Early Detection**
 - Routers maintain running average of queue length
 - When a queue length exceeds, router starts dropping the packets at random.
 - Sender will notice the loss as there are no ack. And slow down
 - Better performance.

Load Shedding (1)

- **Random Early Detection**
 - Routers maintain running average of queue length
 - When an average queue length exceeds, router starts dropping the packets at random.
 - Fast senders will have more packet drop
 - Sender will notice the loss as there are no ack. and slow down
 - Better performance.

Quality of Service

- So far, we have made effort to improve network performance
- However, some applications require stronger performance guarantees.
 - Example: Multimedia applications:
- We will focus of providing network performance with a sharp focus on providing quality of service.
- One solution is overprovisioning
 - More money is required
- With quality of service mechanisms, the network can honor performance guarantees at the cost of turning down some requests.
- Issues
 - What applications need from the network?
 - How to regulate the traffic that enters the network?
 - How to reserve the resources at routers to guarantee performance?
 - Whether the network can safely accept more traffic
- No single technique deals efficiently all these issues..

Quality of Service

- Application requirements »
- Traffic shaping »
- Packet scheduling »
- Admission control »
- Integrated services »
- Differentiated services »

Parameters Defining QoS

- **Throughput/bandwidth** - the total amount of work completed during a specific time interval.
- **Delay** - the elapsed time from when a request is first submitted to and when the desired result is produced.
- **Jitter** - the delays that occur during the playback of a stream.
 - Due to lost frames.
 - Not acceptable for continuous media applications
 - Random variation of transmission time
- **Reliability/loss** - how errors are handled during transmission and processing of continuous media.

Application Requirements (1)

Different applications care about different properties

- We want all applications to get what they need

| Application | Bandwidth | Delay | Jitter | Loss |
|-------------------|-----------|--------|--------|--------|
| Email | Low | Low | Low | Medium |
| File sharing | High | Low | Low | Medium |
| Web access | Medium | Medium | Low | Medium |
| Remote login | Low | Medium | Medium | Medium |
| Audio on demand | Low | Low | High | Low |
| Video on demand | High | Low | High | Low |
| Telephony | Low | High | High | Low |
| Videoconferencing | High | High | High | Low |

Stringency of applications' quality-of-service requirements.

Application Requirements (2)

- Bandwidth:
 - E-mail, audio, remote login not need much bandwidth. But file sharing and video in all forms need high bandwidth.
- Delay:
 - File transfer, email, audio, and video are not delay sensitive. That is if the packets are delayed uniformly across all by a few seconds, no harm is done. So, playing audio or video files from a server does not require low delay.
 - But, interactive applications, web access surfing and remote login are more delay sensitive. Telephony, and video conferencing are strictly delay sensitive
- Jitter
 - E-mail, file sharing and web access not sensitive to irregular arrival of packets.
 - Remote login is sensitive
 - Video and audio are extremely sensitive to Jitter
- Loss:
 - E-mail, file sharing, web access and remote login have more stringent requirements on loss and bits should be delivered correctly.
 - Audio and video applications tolerate lost packets.
 - People do not notice short pauses or skipped frames.

Application Requirements (3)

- Network support different categories of QoS
 - Constant bit rate (telephony)
 - Simulate a wire by providing uniform bandwidth and a uniform delay
 - Real-time variable bit rate (e.g., compressed video conferencing)
 - Sending a complex frame requires many bits, where a shot of white ball requires less bits.
 - Non-real-time variable bit rate (e.g., watching a movie on demand)
 - Buffering can be used
 - Available bit rate (e.g., file transfer)
 - Use the available bandwidth.

Application Requirements (2)

Network provides service with different kinds of QoS (Quality of Service) to meet application requirements

| Network Service | Application |
|---------------------------------|--------------------|
| Constant bit rate | Telephony |
| Real-time variable bit rate | Videoconferencing |
| Non-real-time variable bit rate | Streaming a movie |
| Available bit rate | File transfer |

Example of QoS categories from ATM networks

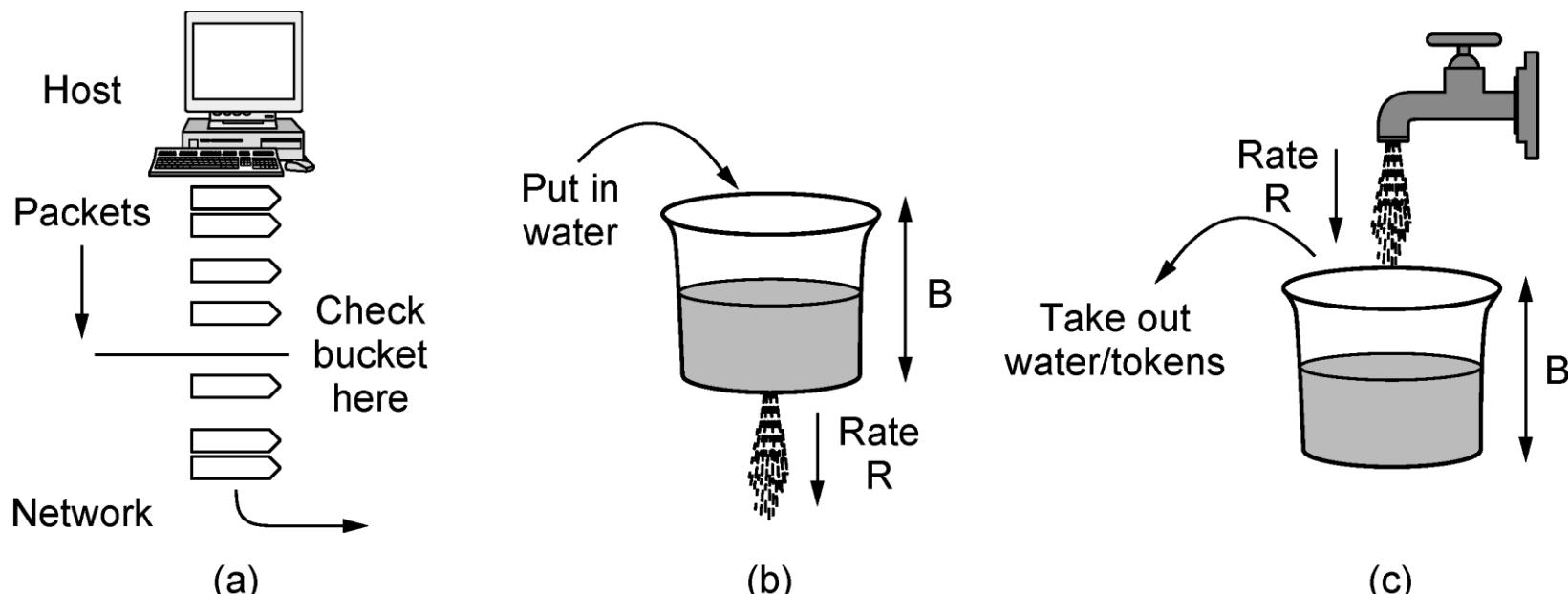
Traffic Shaping (1)

- We must know that traffic is being guaranteed
- However, traffic in data is BURSTY!
- Bursts in the traffic are difficult to handle than constant rate traffic as they can fill the buffers and cause packets to be lost.
- Traffic shaping
 - regulates the average rate and burstiness
- Monitoring traffic flow is called “Traffic policing”
- Traffic shaping and Traffic policing are important for real-time data (audio and video).

Leaky bucket algorithm

- Sliding window controls the data to be transmitted.
- Leaky bucket algorithm
 - Imagine a bucket with a small hole at the bottom.
 - No matter the rate the data enters, the outflow is constant.
 - Once the bucket is full with a capacity B , any additional water splits over the sides and lost.
 - Algorithm
 - Each host is connected to the network containing a leaky bucket.
 - If the packet arrives when the bucket is full, it is queued until enough water leaks out to hold it or be discarded.

Leaky and Token buckets



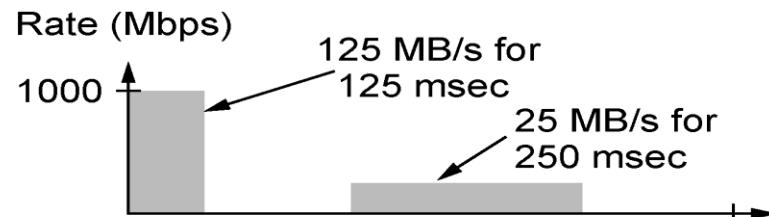
(a) Shaping packets. (b) A leaky bucket. (c) A token bucket.

Token bucket algorithm

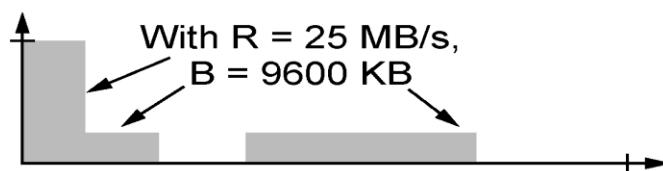
- Imagine a bucket that is being filled. A tap is running at the rate R and the bucket has a capacity of B , as before.
- To send a packet, we must able to take water or tokens as out of bucket.
- If the bucket is empty, we must wait till tokens arrive.
- No more than a fixed number of tokens can accumulate in the bucket.
- Both leaky and token bucket algorithms smoothen the traffic by limiting the rate of flow.

Traffic Shaping (3)

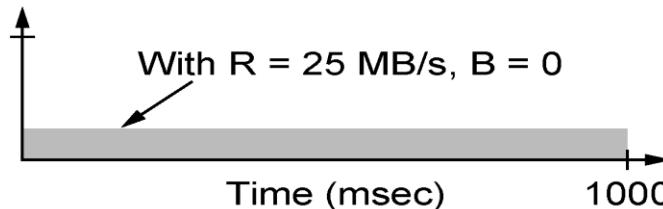
- Both leaky and token bucket algorithms smoothen the traffic by limiting the rate of flow.



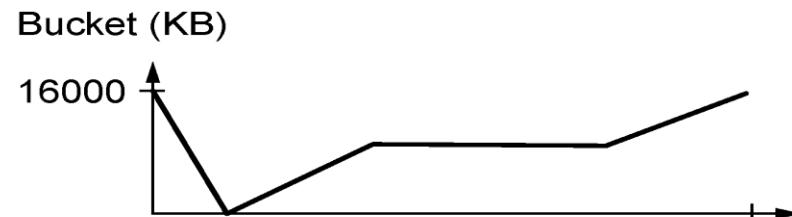
(a)



(b)



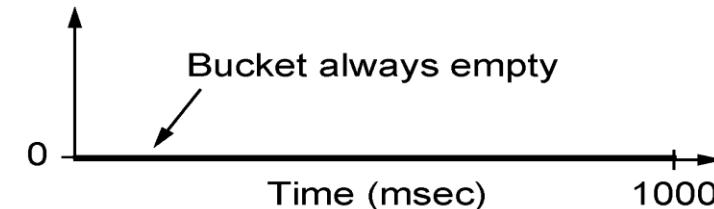
(c)



(d)



(e)



(f)

(a) Traffic from a host. Output shaped by a token bucket of rate 200 Mbps and capacity (b) 9600 KB and (c) 0 KB. Token bucket level for shaping with rate 200 Mbps and capacity (d) 16,000 KB, (e) 9600 KB, and (f) 0 KB.

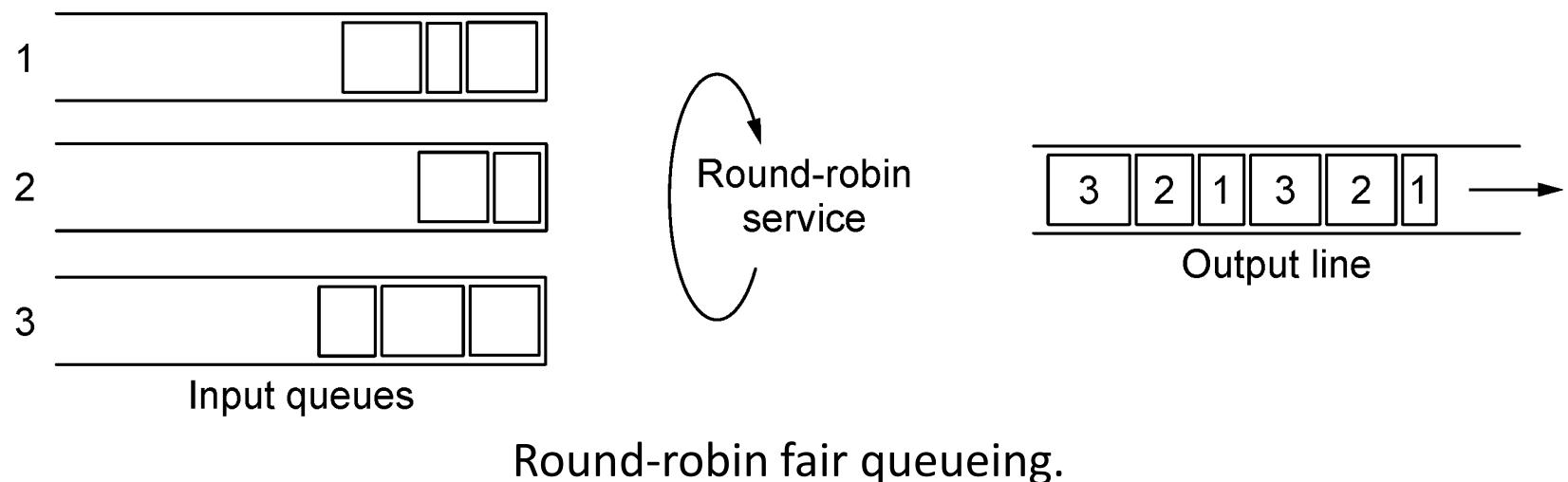
- Host generates the traffic pattern as shown in Figure (a). The pattern is bursty.
 - Average rate over one second is 200Mbps, even though the host sends a burst of 16,000 KB at the top speed of 1000 Mbps (1/8 of second).
 - Routers can only accept data until only the buffers are filled. The buffer size is 9600 KB, smaller than the traffic burst. Some of the packets will be dropped.
- With token bucket, we can shape the traffic. The output of token bucket is shown in Figure (b). The host can send full throttle for a short while, until it has drained the bucket. It will cut back 200Mbps until the burst has been sent.
- The level of token bucket is given in Figure (e)
- We can also shape the traffic to be less bursty. Figure (c) shows the output of a token bucket with $R=200$ Mbps and a capacity of zero. This is the extreme case. The corresponding bucket level Figure (f) is always empty.
- Figure (d) shows the bucket level for a token bucket with $R=200$ Mbps and a capacity of $B=16,000$ KB.
- This is the smallest token bucket through which the traffic passes unaltered.
- Leaky and token buckets are easy to implement.

Packet Scheduling (1)

- Regulating the traffic is OK.
- However, to provide performance guarantee, we must reserve sufficient resources along the route for performance guarantee.
- Packet scheduling algorithms
 - Allocate router resources among the packets of a flow and between competing flows.
- Three kinds of resources are reserved
 - Bandwidth
 - If flow requires 1Mbps and bandwidth is 2Mbps, one should not try to get three flows through a line.
 - Buffer space
 - Purpose of buffer is to absorb the small bursts. To have quality of service, some buffers are reserved for a specific flow.
 - CPU cycles
 - Making sure that CPU is not loaded

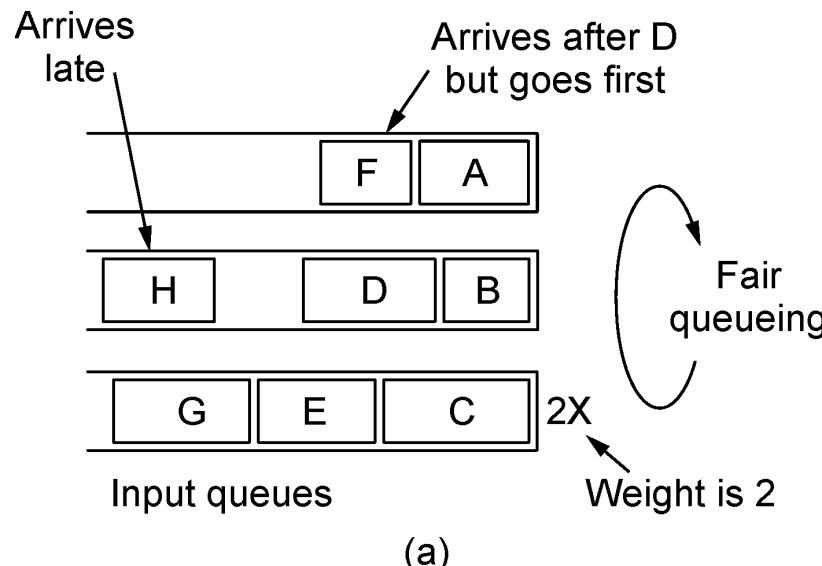
Packet Scheduling (1)

- FIFO: First-Come-First-Serve
 - Drop newly arrived packets
 - Tail drop behavior
 - Simple to implement
 - Not good for achieving quality of service in case of multiple flows
- Fair queuing round-robin
 - Routers have separate queues, one for each flow.
 - When line becomes idle, the router scans queues in a round robin fashion.



Packet Scheduling (2)

- Byte-byte round robin.
 - Compute the virtual time that is the number of round at which each packet would finish being sent.
 - It gives all hosts the same priority
- Weighted Fair Scheduling (WFG)
 - Number of bytes per round will be weight (W) of the flow.
 - $F_i = \max(A_i, F_{i-1}) + L_i/W$
 - A_i is the arrival time, F_i is the finish time, L_i is the length of the packet



(a) Weighted Fair Queueing. (b) Finishing times for the packets.

| Packet | Arrival time | Length | Finish time | Output order |
|--------|--------------|--------|-------------|--------------|
| A | 0 | 8 | 8 | 1 |
| B | 5 | 6 | 11 | 3 |
| C | 5 | 10 | 10 | 2 |
| D | 8 | 9 | 20 | 7 |
| E | 8 | 8 | 14 | 4 |
| F | 10 | 6 | 16 | 5 |
| G | 11 | 10 | 19 | 6 |
| H | 20 | 8 | 28 | 8 |

(b)

Admission Control (1)

- Admission control takes a traffic flow specification and decides whether the network can carry it
- Sets up packet scheduling to meet QoS

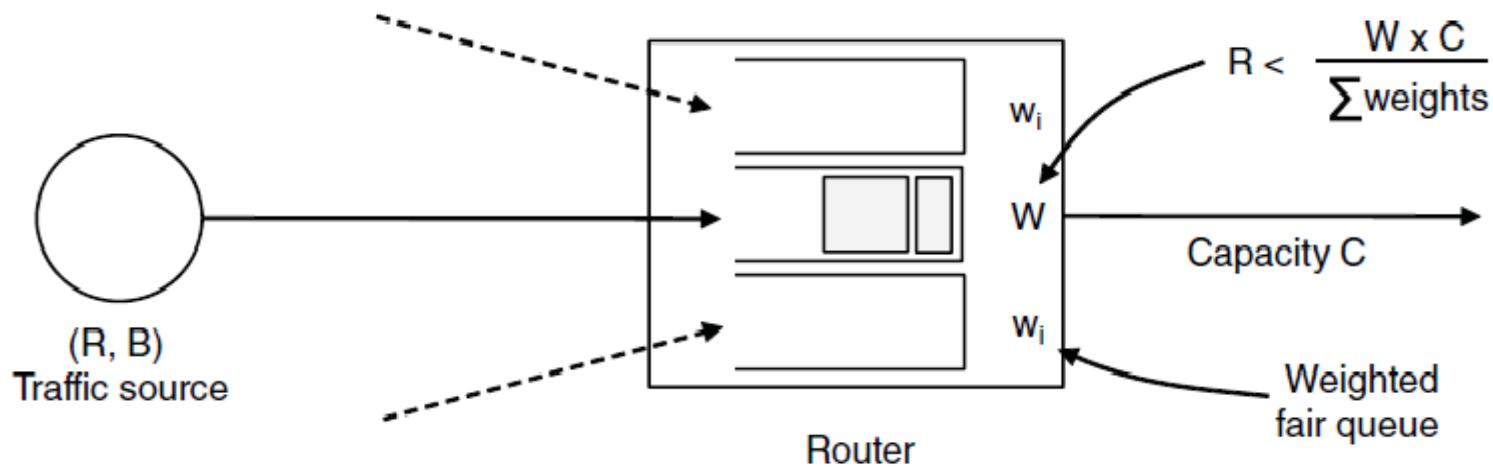
| Parameter | Unit |
|---------------------|-----------|
| Token bucket rate | Bytes/sec |
| Token bucket size | Bytes |
| Peak data rate | Bytes/sec |
| Minimum packet size | Bytes |
| Maximum packet size | Bytes |

Example flow specification

Admission Control (2)

Construction to guarantee bandwidth B and delay D:

- Shape traffic source to a (R, B) token bucket
- Run WFQ (Waited Fair Queue) with weight W / all weights > R/capacity
- Holds for all traffic patterns, all topologies



Network Layer in the Internet (1)

- IP Version 4 »
- IP Addresses »
- IP Version 6 »
- Internet Control Protocols »
- Label Switching and MPLS »
- OSPF—An Interior Gateway Routing Protocol »
- BGP—The Exterior Gateway Routing Protocol »
- Internet Multicasting »
- Mobile IP »

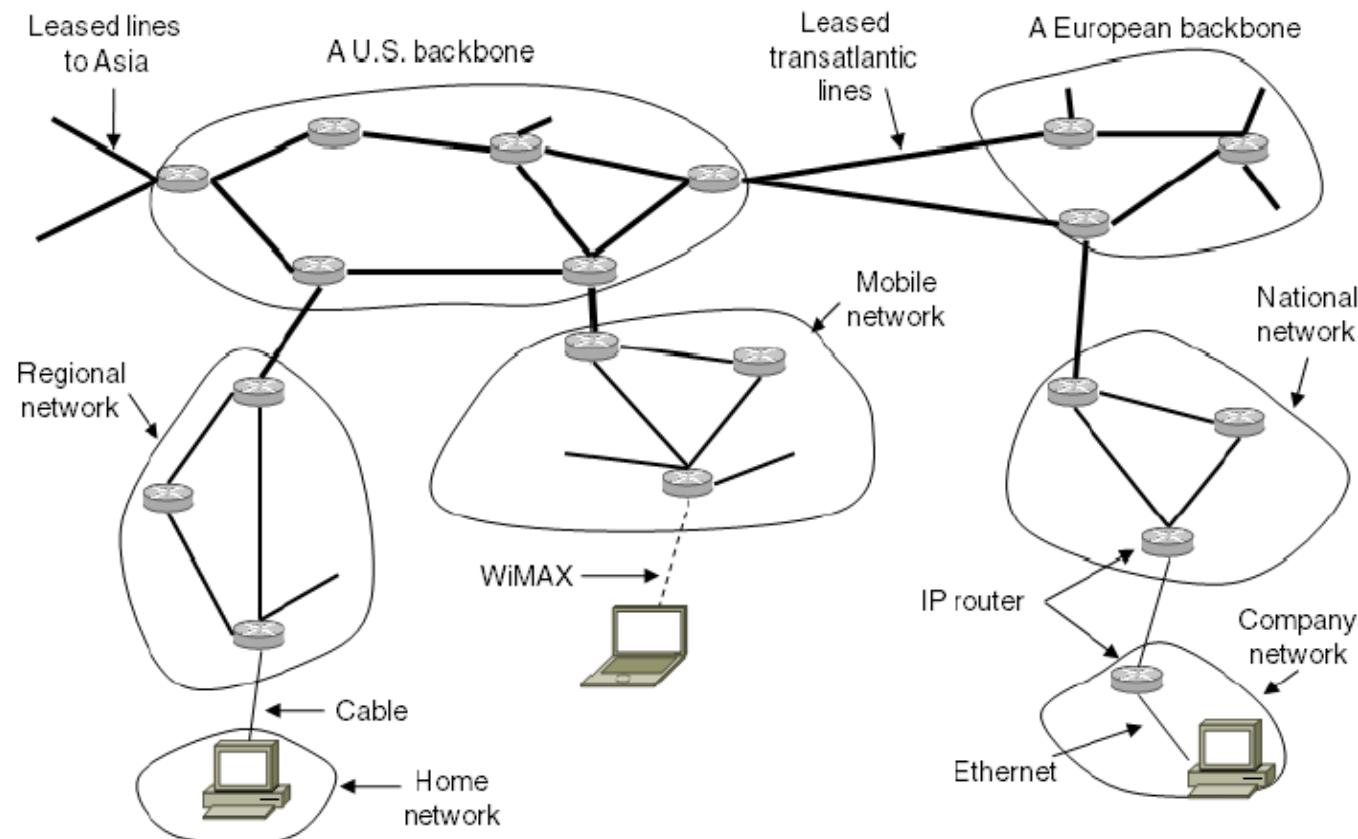
Network Layer in the Internet (2)

IP has been shaped by guiding principles:

- Make sure it works
- Keep it simple
- Make clear choices
- Exploit modularity
- Expect heterogeneity
- Avoid static options and parameters
- Look for good design (not perfect)
- Strict sending, tolerant receiving
- Think about scalability
- Consider performance and cost

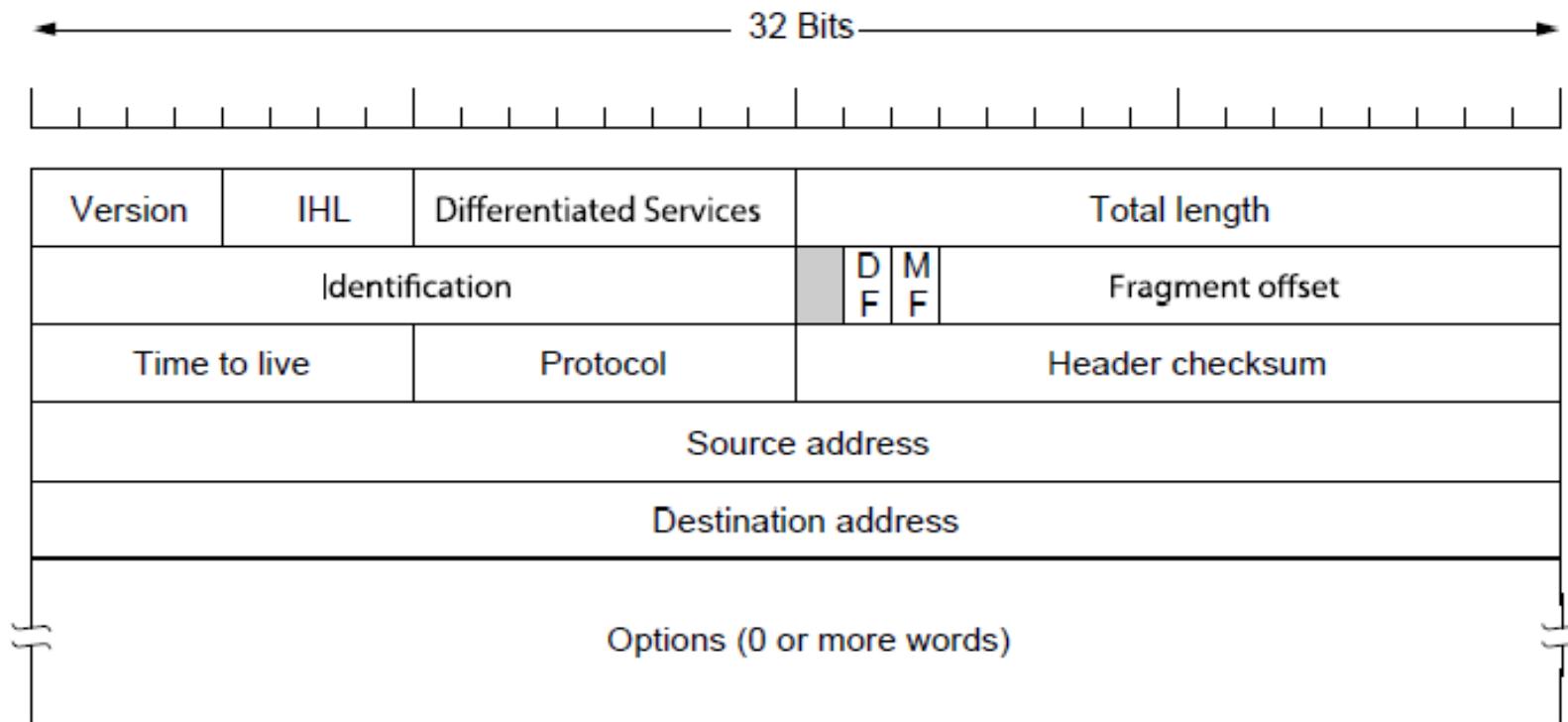
Network Layer in the Internet (3)

Internet is an interconnected collection of many networks that is held together by the IP protocol



IP Version 4 Protocol (1)

IPv4 (Internet Protocol) header is carried on all packets and has fields for the key parts of the protocol:



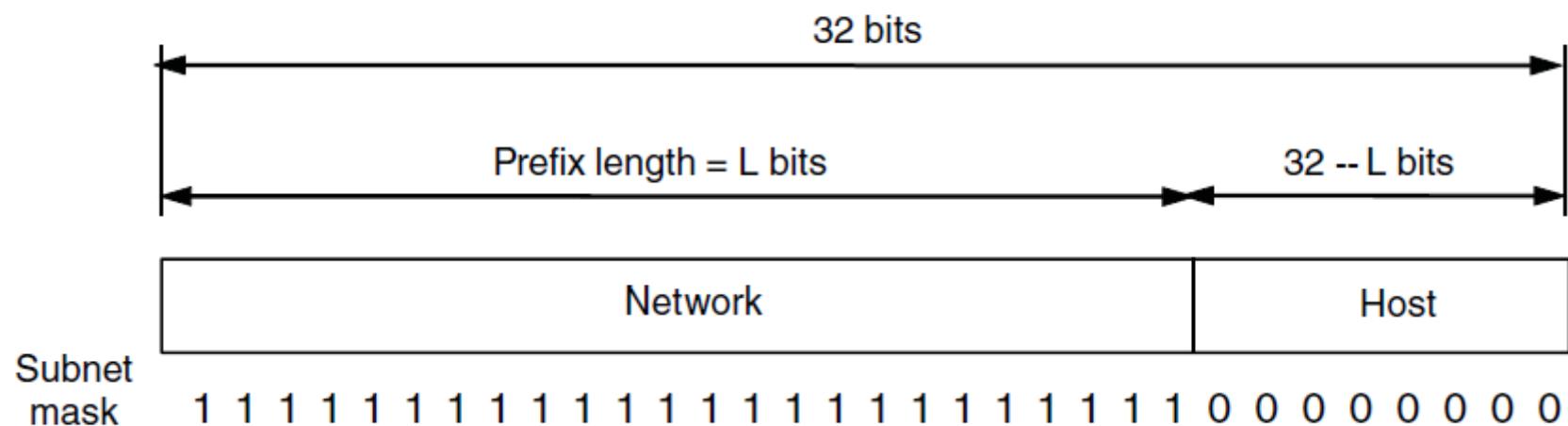
IP Version 4 Protocol (1)

- Version filed
- IHL: header length
- Differentiated services: type of service
- Total length: every thing in the datagram
- Identification field: which packet the newly arrived fragment belong to
 - All the fragments of the same packet have the same identification code.
- Next field: not defined
- DF: do not fragment
- MF more fragments
 - It is set to know whether all the fragments of datagram have arrived.
- Fragment offset
 - Where in the current packet this packet belongs
- TTL: Time to live
 - To limit packet life time
 - Decrementated after each hop
- Protocol: UDP or TCP
- Header checksum
- Source and destination address indicate the IP address of source and destination
- Options field
 - For other ideas
 - Security
 - Strict route
 - Set of routes
 - Loose routes
 - Set of routes

IP Addresses (1) – Prefixes

Addresses are allocated in blocks called prefixes

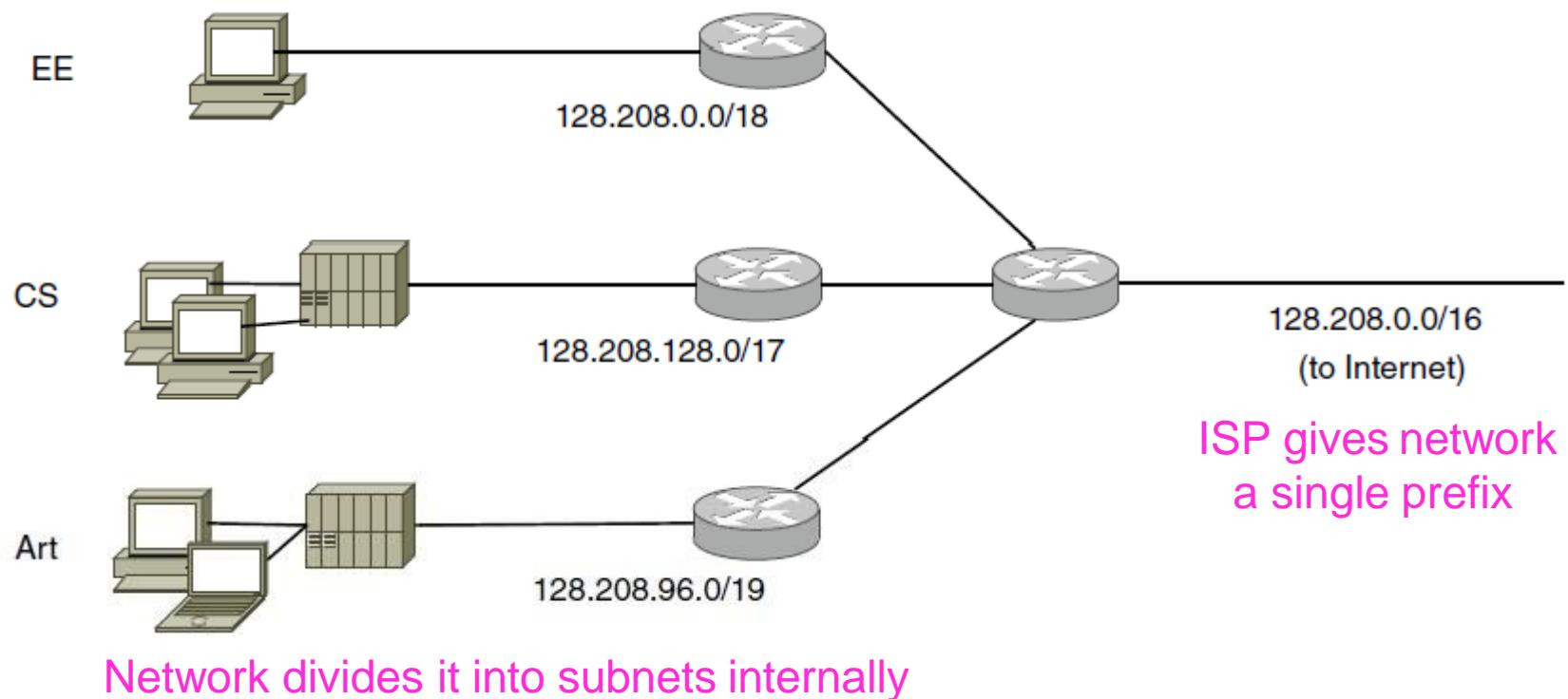
- Prefix is determined by the network portion
- Has 2^L addresses aligned on 2^L boundary
- Written address/length, e.g., 18.0.31.0/24



IP Addresses (2) – Subnets

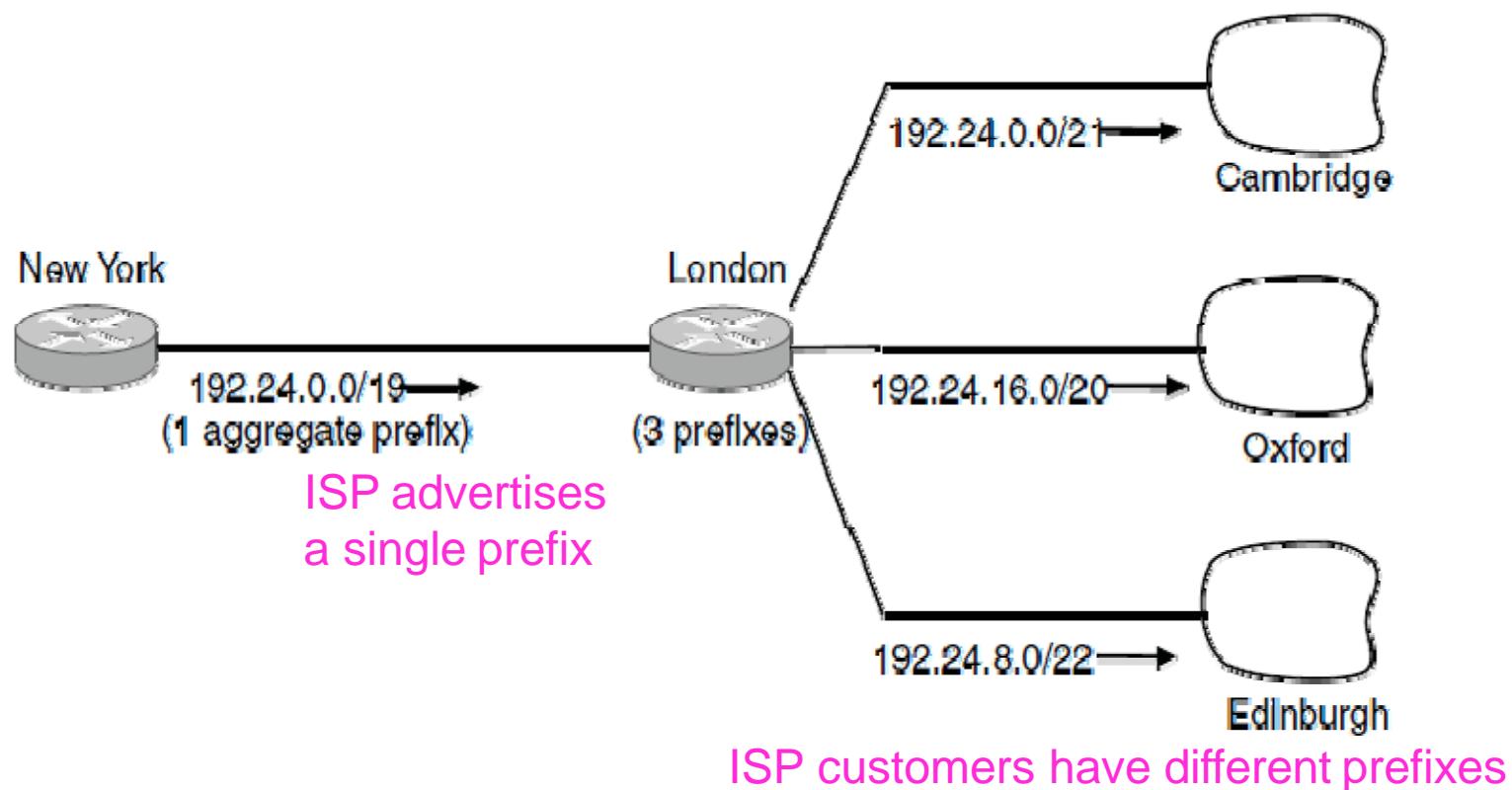
Subnetting splits up IP prefix to help with management

- Looks like a single prefix outside the network



IP Addresses (3) – Aggregation

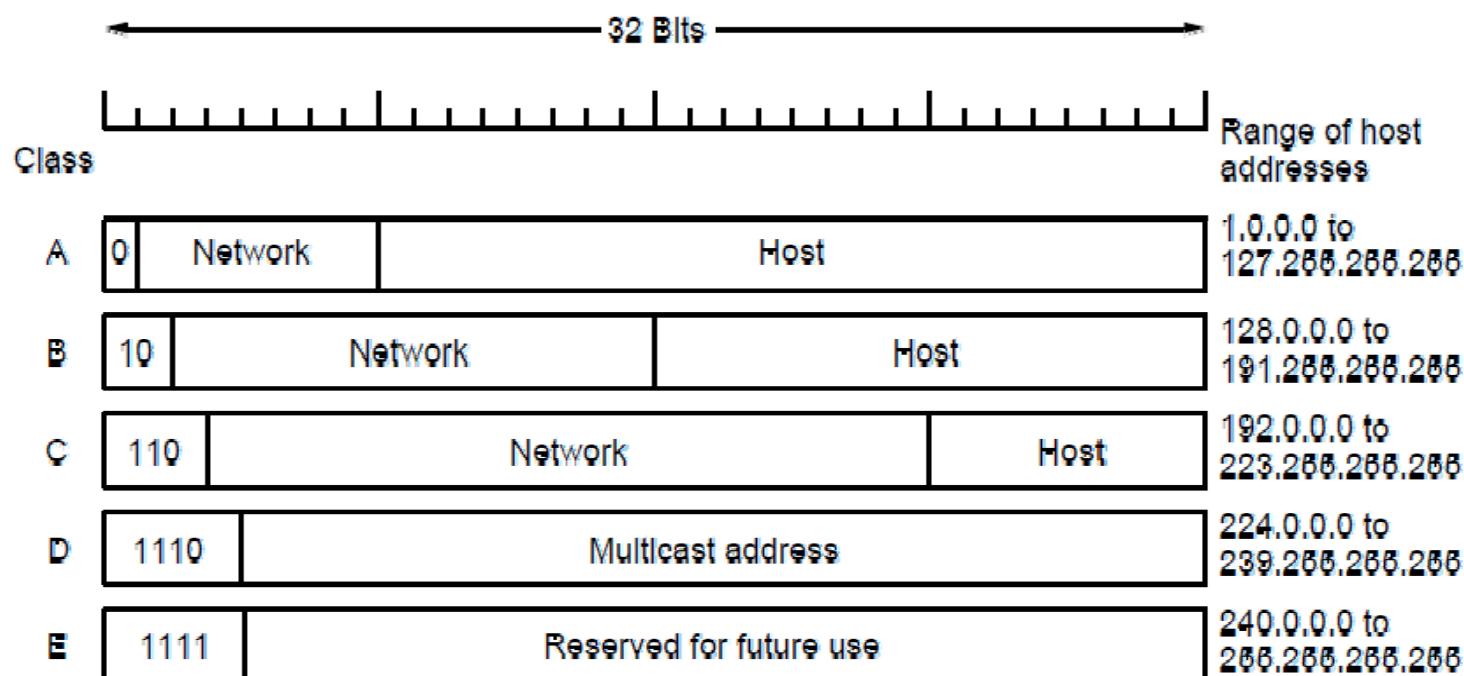
Aggregation joins multiple IP prefixes into a single larger prefix to reduce routing table size



IP Addresses (5) – Classful Addressing

Old addresses came in blocks of fixed size (A, B, C)

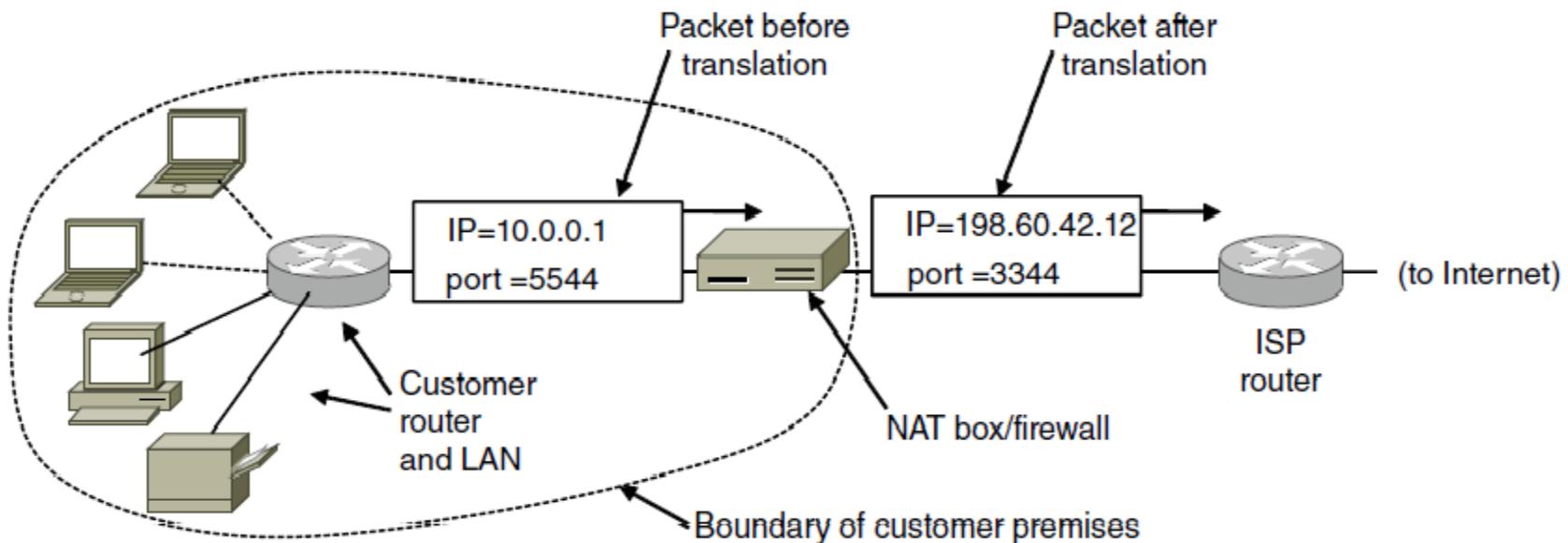
- Carries size as part of address, but lacks flexibility
- Called classful (vs. classless) addressing



IP Addresses (6) – NAT

NAT (Network Address Translation) box maps one external IP address to many internal IP addresses

- Uses TCP/UDP port to tell connections apart
- Violates layering; very common in homes, etc.



Internet Control Protocols (1)

IP works with the help of several control protocols:

- ICMP is a companion to IP that returns error info
 - Required, and used in many ways, e.g., for traceroute
- ARP finds Ethernet address of a local IP address
 - Glue that is needed to send any IP packets
 - Host queries an address and the owner replies
- DHCP assigns a local IP address to a host
 - Gets host started by automatically configuring it
 - Host sends request to server, which grants a lease

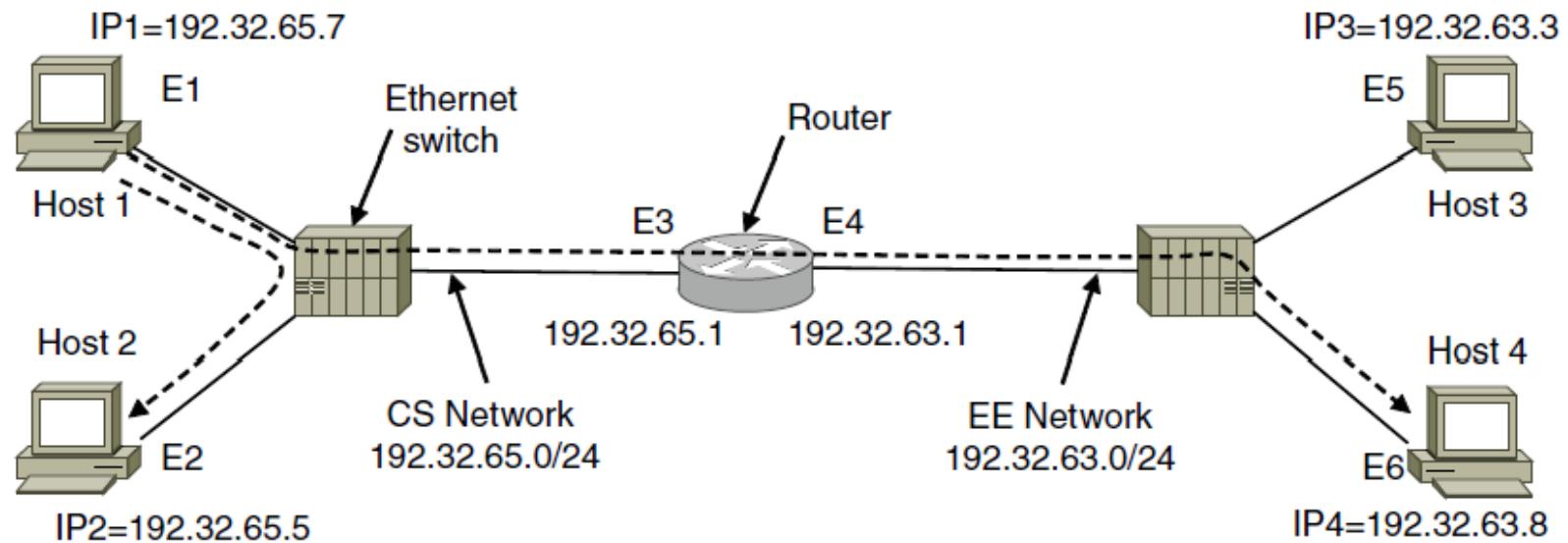
Internet Control Protocols (2)

Main ICMP (Internet Control Message Protocol) types:

| Message type | Description |
|-----------------------------------|----------------------------------|
| Destination unreachable | Packet could not be delivered |
| Time exceeded | Time to live field hit 0 |
| Parameter problem | Invalid header field |
| Source quench | Choke packet |
| Redirect | Teach a router about geography |
| Echo and Echo reply | Check if a machine is alive |
| Timestamp request/reply | Same as Echo, but with timestamp |
| Router advertisement/solicitation | Find a nearby router |

Internet Control Protocols (3)

ARP (Address Resolution Protocol) lets nodes find target Ethernet addresses [pink] from their IP addresses

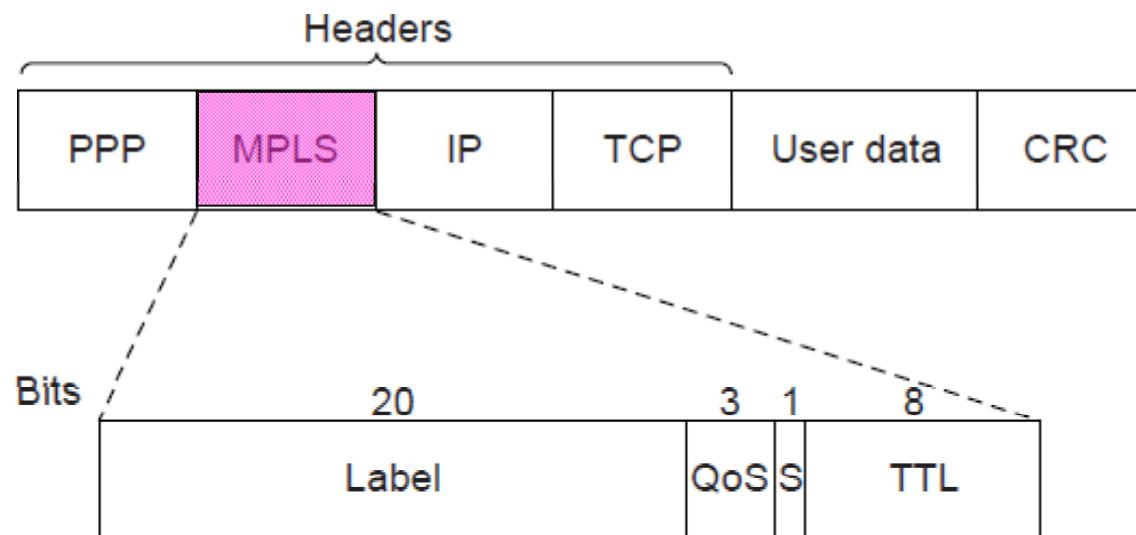


| Frame | Source IP | Source Eth. | Destination IP | Destination Eth. |
|------------------------|-----------|-------------|----------------|------------------|
| Host 1 to 2, on CS net | IP1 | E1 | IP2 | E2 |
| Host 1 to 4, on CS net | IP1 | E1 | IP4 | E3 |
| Host 1 to 4, on EE net | IP1 | E4 | IP4 | E6 |

Label Switching and MPLS (1)

MPLS (Multi-Protocol Label Switching) sends packets along established paths; ISPs can use for QoS

- Path indicated with label below the IP layer



End

Chapter 5

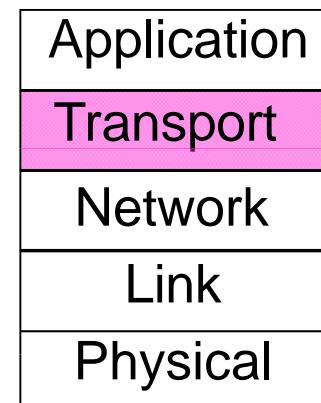
Transport Layer

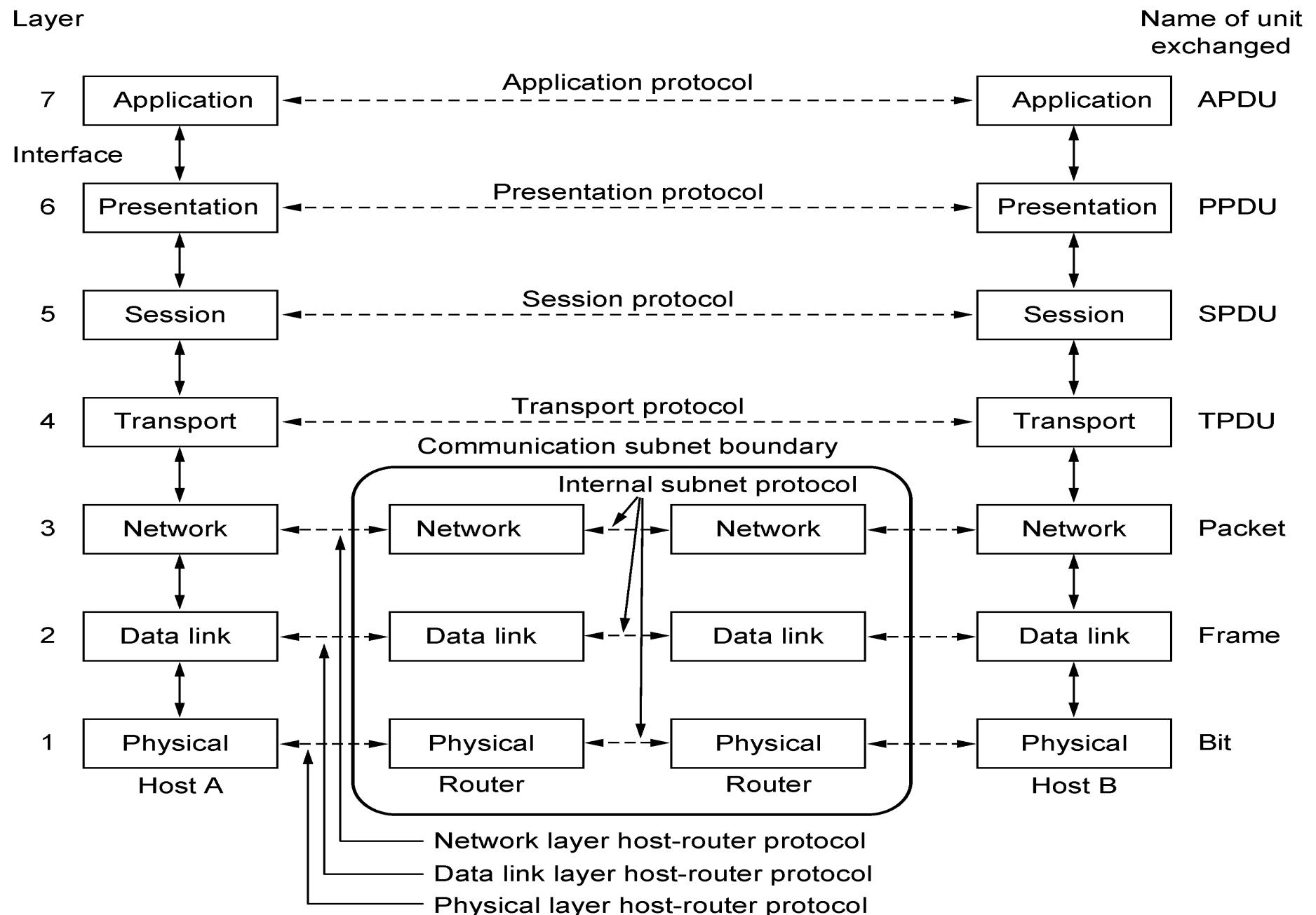
Chapter 6

- Transport Service
- Elements of Transport Protocols
- Congestion Control
- Internet Protocols – UDP
- Internet Protocols – TCP
- Delay-Tolerant Networking

The Transport Layer

- Responsible for delivering data across networks with the desired reliability or quality
- Provide data support from a process on a source machine to a process on destination machine





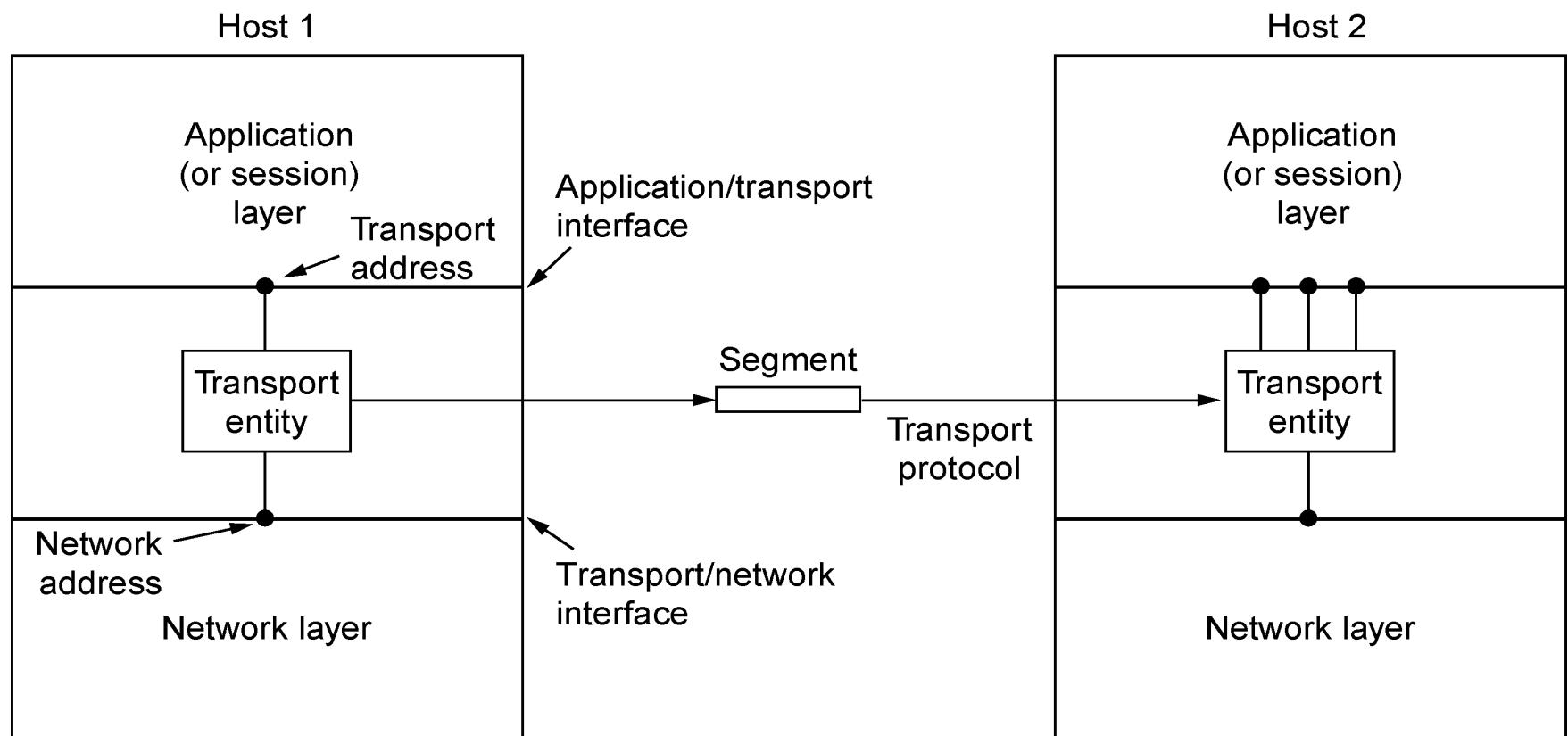
The OSI reference model.

Transport Service

- Services Provided to the Upper Layer »
- Transport Service Primitives »
- Berkeley Sockets »
- Socket Example: Internet File Server »

Services Provided to the Upper Layers (1)

- Goal
 - To provide efficient, reliable and cost-effective data transmission service to its users, i.e., to a process in the application layer
- Transport layer adds reliability to the network layer and offers
 - connectionless (e.g., UDP) and
 - connection-oriented (e.g., TCP) service to applications

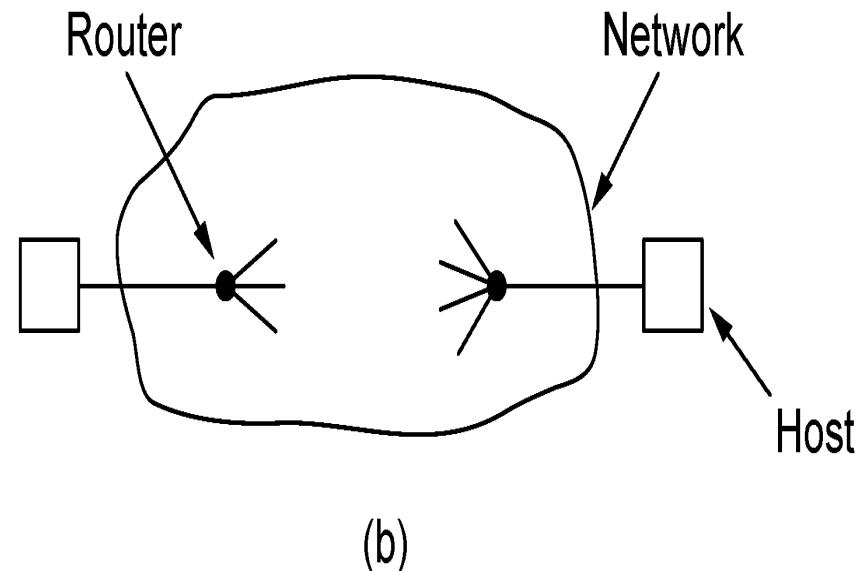
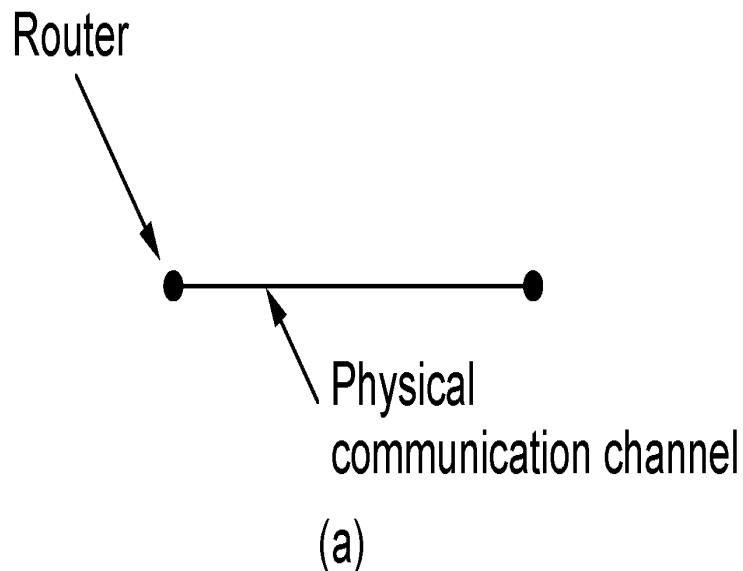


Why different layer?

- Transport service is similar to network layer. Then, why two distinct layers?
 - Transport layer runs entirely on users machine
 - Network layer runs on routers, which are operated by the carrier.
 - What happens if the network layer offers inadequate service?
 - What if it frequently loses packets?
 - What happens if the routers crash time to time?

About Transport Layer

- Essence of the transport service
 - More reliability to applications
- Users have no control over network layer.
- Transport layer can use retransmissions.
- Transport primitives are independent of network primitives.
 - Network service calls may be different for each network



(a) Environment of the data link layer. (b) Environment of the transport layer.

Transport Service Primitives (1)

- Example:
- Two processes in a single machine connected through a pipe
 - Process A puts the data and Process B consumes
- Connection between them is 100 % perfect
 - **No acks, no lost packets, congestion or any thing.**
- Transport layer provides similar service in the network for connection-oriented service **with acks, lost packets, congestion and other mechanisms**
- Transport layer also provides unreliable datagram service
- Another difference
 - Network service is used by transport entities
 - Few users write their own transport entities
 - Few users writes programs using network services
 - Many programs use transport primitives.

Transport Service Primitives (1)

- Primitives that applications might call to transport data for a simple connection-oriented service:
 - Client calls CONNECT, SEND, RECEIVE, DISCONNECT
 - Server calls LISTEN, RECEIVE, SEND, DISCONNECT

| Primitive | Packet sent | Meaning |
|------------|--------------------|--|
| LISTEN | (none) | Block until some process tries to connect |
| CONNECT | CONNECTION REQ. | Actively attempt to establish a connection |
| SEND | DATA | Send information |
| RECEIVE | (none) | Block until a DATA packet arrives |
| DISCONNECT | DISCONNECTION REQ. | Request a release of the connection |

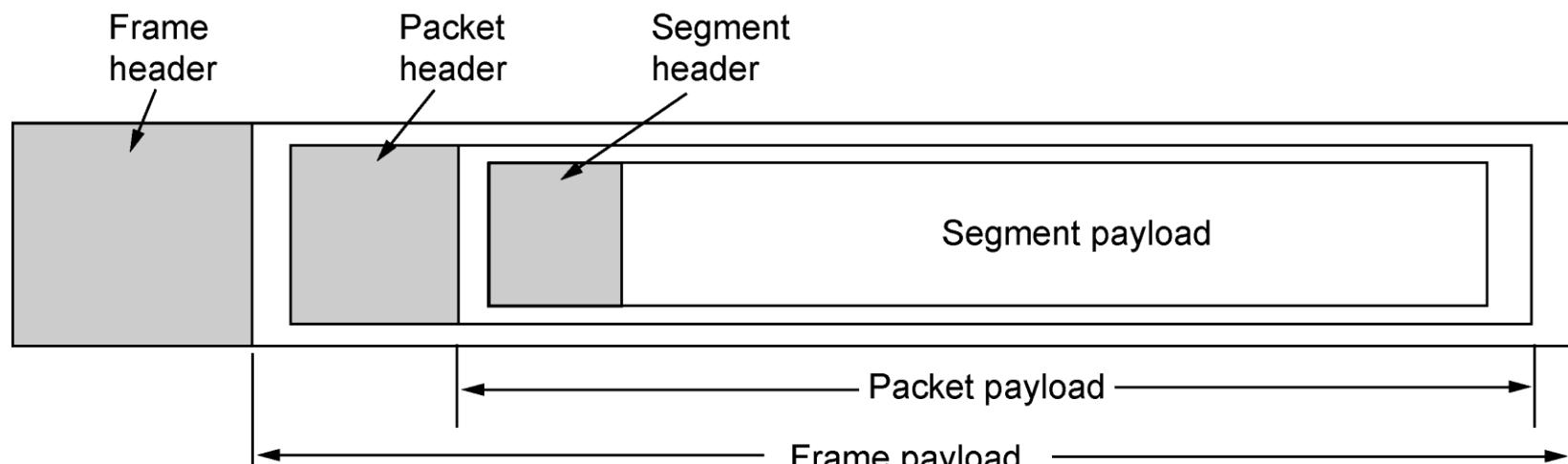
The primitives for a simple transport service.

Transport Service Primitives (1)

- LISTEN: calling a library procedure that makes a system call that blocks the server until a client tunes up.
- CONNECT: When client wants to talk to the server, it executes a CONNECT primitive. It results into CONNECT REQ. segment sent to the server.
- The server unblocks the server and sends CONNECTION ACCEPTED segment back to the client.
- Data can be exchanged using the SEND and RECEIVE primitives.
- Every data packet is acknowledged.
- When a connection is no longer needed, transport user issues a DISCONNECT primitive. Upon the arrival, the connection is released.
- Each direction is closed independently.

Services Provided to the Upper Layers (2)

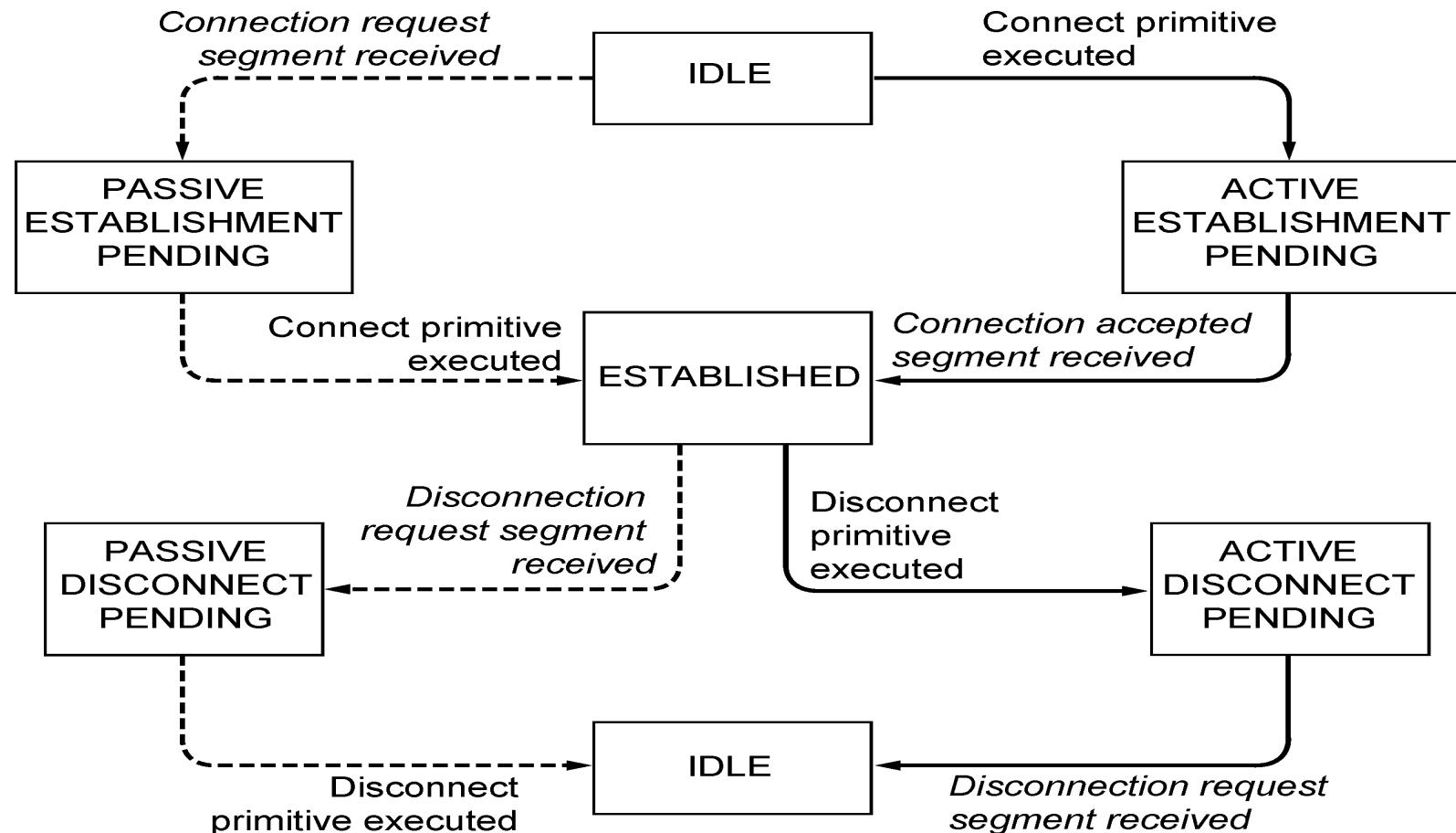
- Segments are contained in the packet sent by network layer, in turn packets are contained in frames exchanged by data link layer.



Nesting of segments, packets, and frames.

Transport Service Primitives (2)

State diagram for a simple connection-oriented service



A state diagram for a simple connection management scheme. Transitions labeled in italics are caused by packet arrivals. The solid lines show the client's state sequence. The dashed lines show the server's state sequence.

Berkeley Sockets

Very widely used primitives started with TCP on UNIX

- Notion of “sockets” as transport endpoints
- Like simple set plus SOCKET, BIND, and ACCEPT

| Primitive | Meaning |
|-----------|---|
| SOCKET | Create a new communication endpoint |
| BIND | Associate a local address with a socket |
| LISTEN | Announce willingness to accept connections; give queue size |
| ACCEPT | Passively establish an incoming connection |
| CONNECT | Actively attempt to establish a connection |
| SEND | Send some data over the connection |
| RECEIVE | Receive some data from the connection |
| CLOSE | Release the connection |

The socket primitives for TCP.

Socket Example – Internet File Server (1)

Client code

```
...
if (argc != 3) fatal("Usage: client server-name file-name");
h = gethostbyname(argv[1]);
if (!h) fatal("gethostbyname failed");
s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s <0) fatal("socket");
memset(&channel, 0, sizeof(channel));
channel.sin_family= AF_INET;
memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
channel.sin_port= htons(SERVER_PORT);

c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
if (c < 0) fatal("connect failed");
...
}
```

Get server's IP address

Make a socket

Try to connect

Socket Example – Internet File Server (2)

Client code (cont.)

```
    . . .
    write(s, argv[2], strlen(argv[2])+1);      } Write data (equivalent to
                                                send)
while (1) {
    bytes = read(s, buf, BUF_SIZE);
    if (bytes <= 0) exit(0);
    write(1, buf, bytes);
}
```

Loop reading (equivalent to receive) until no more data;
exit implicitly calls close

Socket Example – Internet File Server (3)

Server code

Socket Example – Internet File Server (4)

Server code

```
...  
while (1) {  
    sa = accept(s, 0, 0);  
    if (sa < 0) fatal("accept failed");  
    read(sa, buf, BUF_SIZE);  
    /* Get and return the file. */  
    fd = open(buf, O_RDONLY);  
    if (fd < 0) fatal("open failed");  
    while (1) {  
        bytes = read(fd, buf, BUF_SIZE);  
        if (bytes <= 0) break;  
        write(sa, buf, bytes);  
    }  
    close(fd);  
    close(sa);  
}
```

Block waiting for the next connection

Read (receive) request and treat as file name

Write (send) all file data

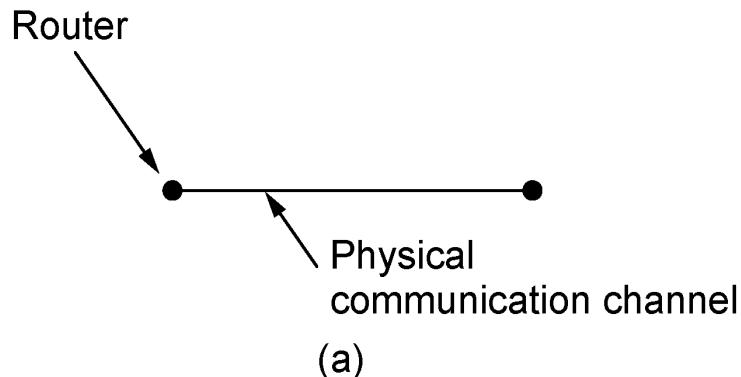
Done, so close this connection

Elements of Transport Protocols

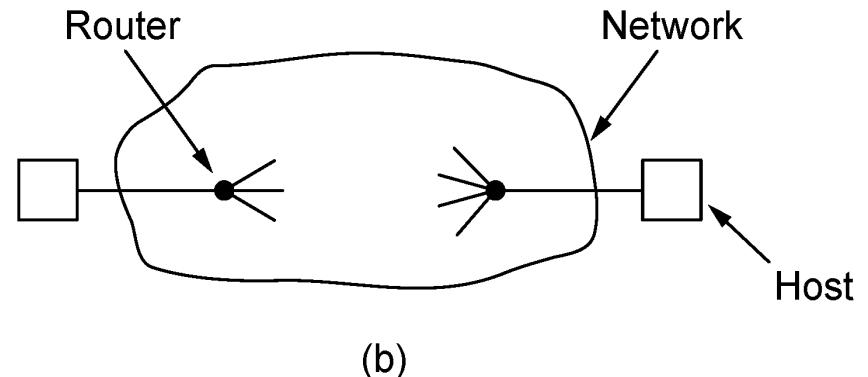
- Addressing »
- Connection establishment »
- Connection release »
- Error control and flow control »
- Multiplexing »
- Crash recovery »

About Transport Protocols

- Transport protocols resemble data link layer protocols
 - Both have to deal with error control, sequencing, and flow control, among other issues
- Significant differences exist due to environmental differences
- Differences
 - One: Data link layer is on direct connection whereas transport layer is on entire network.
 - Two: Delay is more in transport layer, more failures
 - Three: Number of connections are more in transport layer



(a) Environment of the data link layer.

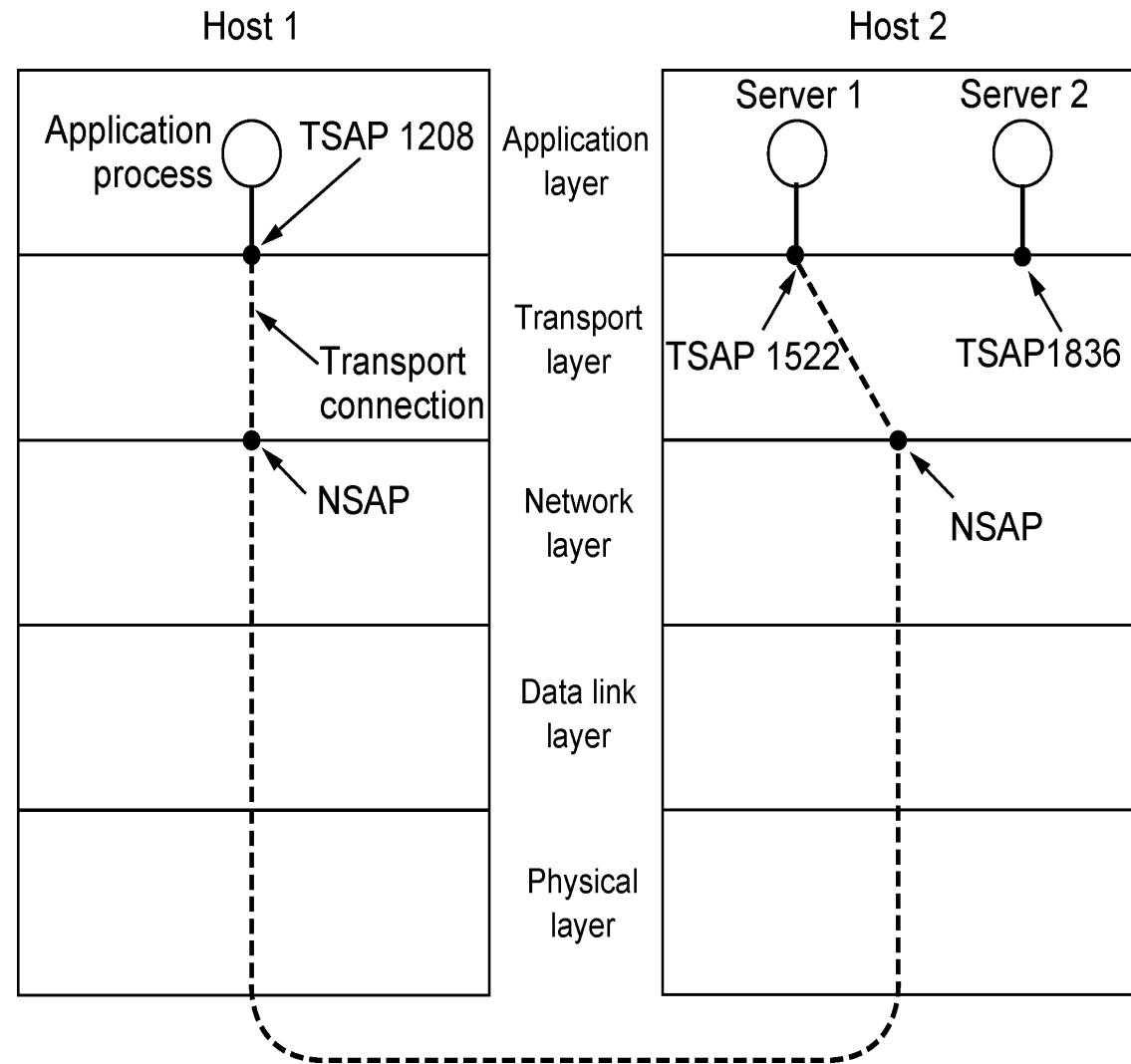


(b)

(b) Environment of the transport layer.

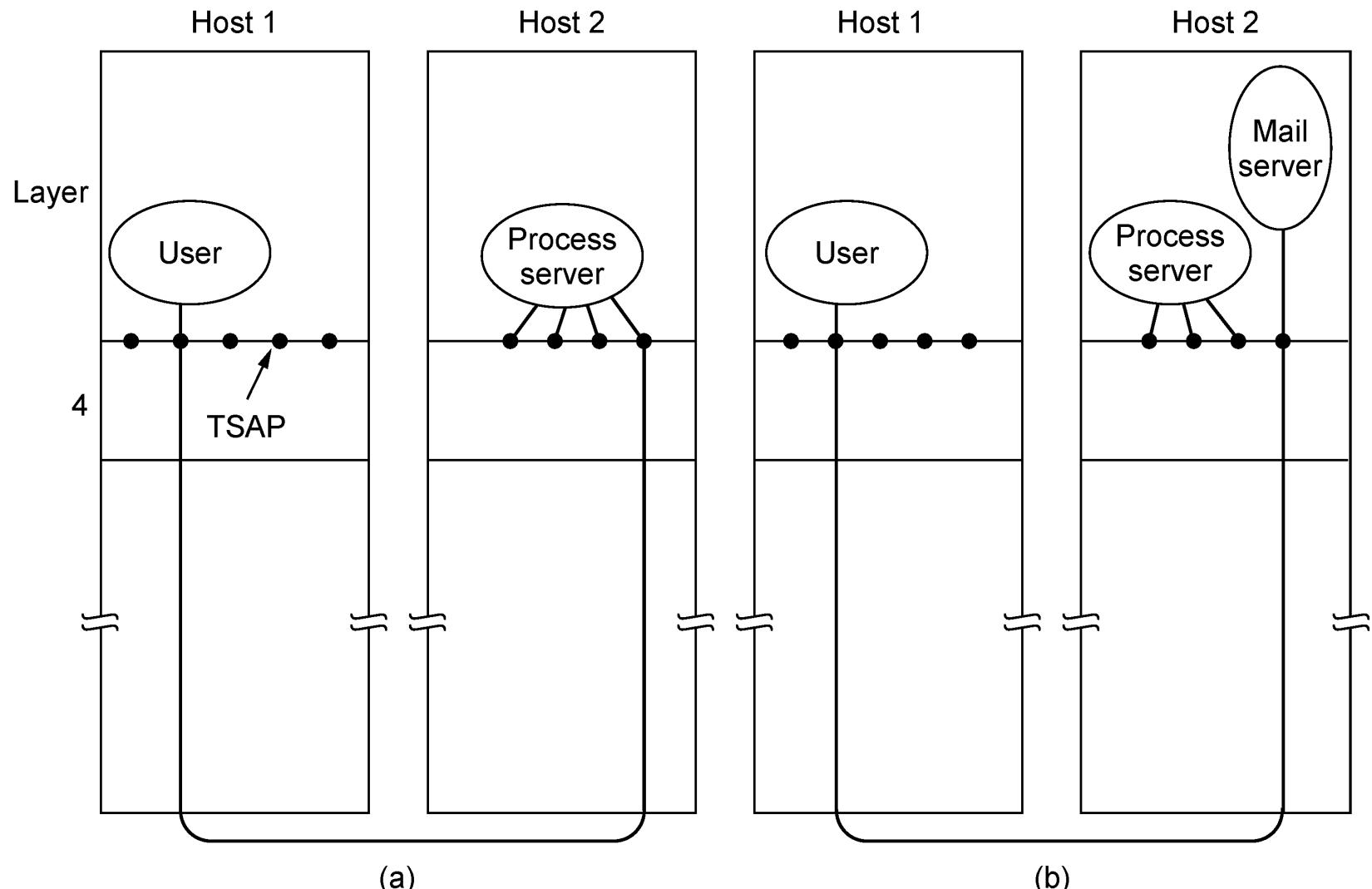
Addressing

- Transport layer adds TSAPs (Transport Service Access Points)
- Multiple clients and servers can run on a host with a single network (IP) address
- TSAPs are ports for TCP/UDP



TSAPs, NSAPs, and transport connections.

- Process server: inetd in UNIX
- Each machine runs a special process server.
- The process server spawns a thread and connects to the related server.



How a user process in host 1 establishes a connection with a mail server in host 2 via a process server.

Connection Establishment (1)

- Key problem is to ensure reliability even though packets may be lost, corrupted, delayed, and duplicated
 - Don't treat an old or duplicate packet as new
 - (Use ARQ (Automatic Repeat Request) and checksums for loss/corruption)
- Key issue
 - Delayed duplicates
 - User sends packets, they may follow longest route.
 - User time out expires. Send the packets again.
 - Receiver receives both old and new packets
- Approaches
 - Restricted network login
 - Putting a hop counter in each packet
 - Timestamping each packet
- We should also ensure that all ack. are dead.

Connection Establishment (1)

Approach:

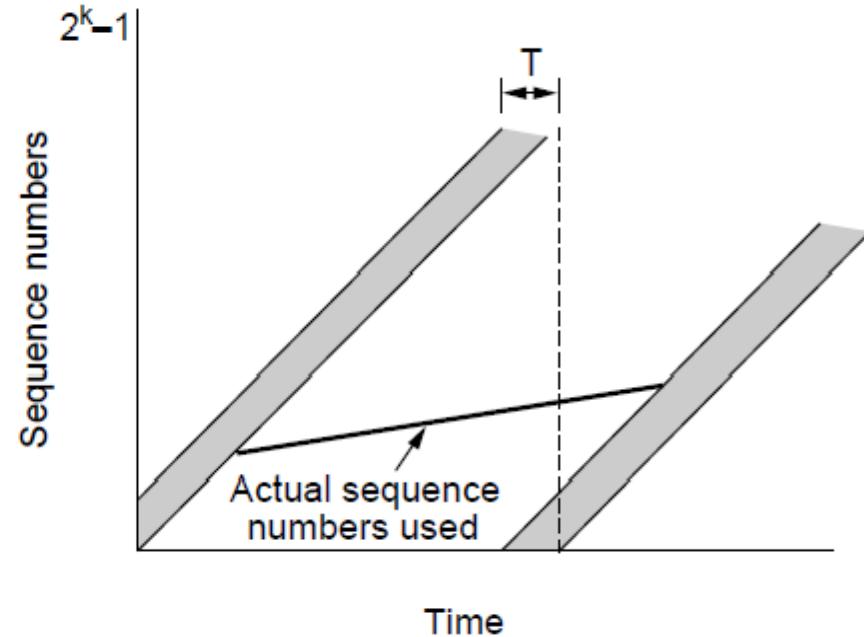
- Don't reuse sequence numbers within twice the MSL (Maximum Segment Lifetime) of $2T=240$ secs
- We can use the clock
- Once the transport entities agree to initial sequence number, any sliding window protocol can be used.
 - It discards duplicate packets
- Issues
 - How to keep packet sequence numbers out of forbidden region?
 - If the host sends too much data too fast, it results sequence numbers to enter into forbidden curve.
 - If the data rate is less than the clock rate, we also get into the trouble.
 - We have to remember the sequence numbers.
- Three-way handshake is proposed to solve the problem

Connection Establishment (2)

- Use a sequence number space large enough that it will not wrap, even when sending at full rate
 - Clock (high bits) advances & keeps state over crash



Need seq. number not to wrap within T seconds



Need seq. number not to climb too slowly for too long

Connection Establishment

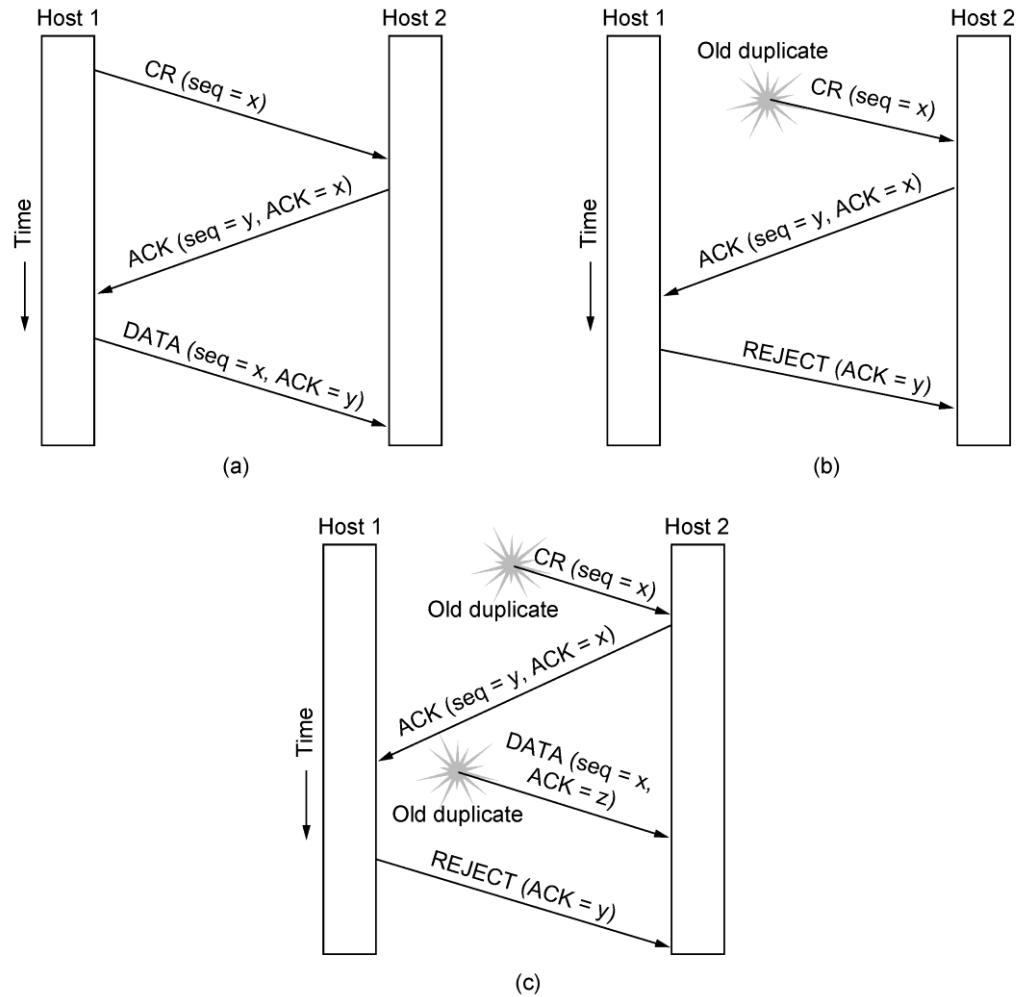
Three-way handshake used for initial packet

- Since no state from previous connection
- Both hosts contribute fresh seq. numbers
- CR = Connect Request
- Protocol
 - Host 1 chooses a sequence number and sends CR
 - Host 2 send CR with ack. and sending its own sequence number.
 - Host 1 ack. the choice of sequence numbers by host 2

Connection Establishment (4)

Three-way handshake
protects against odd cases:

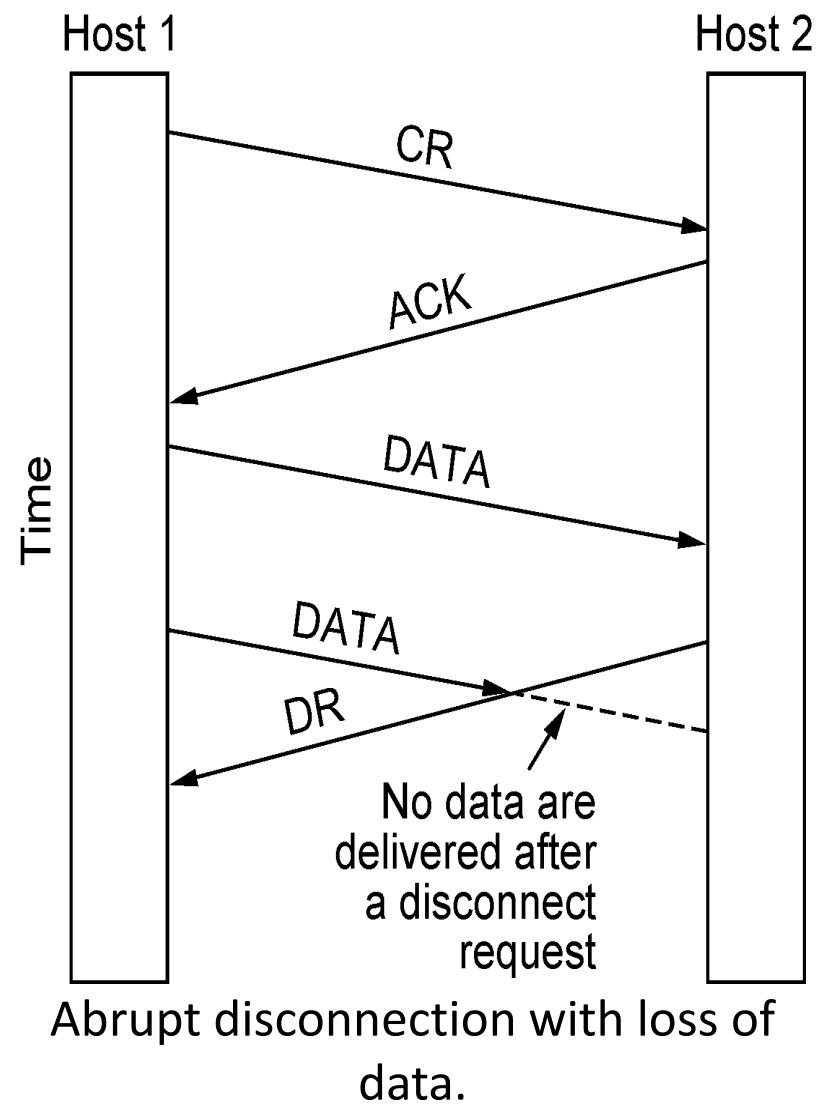
- a) Duplicate CR, Spurious ACK does not connect
- b) Duplicate CR and DATA. Same plus DATA will be rejected (wrong ACK).
- TCP uses three-way handshake to establish connections



Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST.
(a)Normal operation. (b)Old duplicate CONNECTION REQUEST appearing out of nowhere. (c)Duplicate CONNECTION REQUEST and duplicate ACK

Connection Release (1)

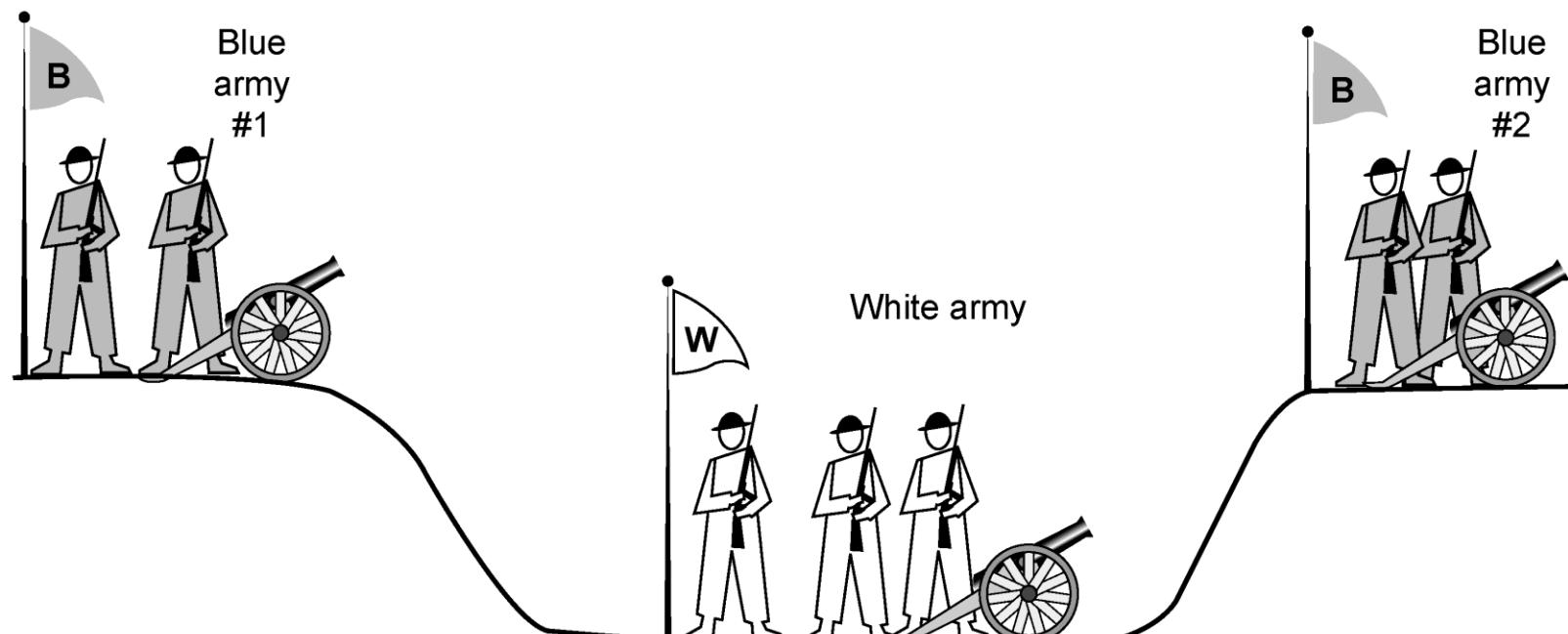
- Key problem is to ensure reliability while releasing
- Two kinds of releases
 - Asymmetric release
 - Like a telephone system. When one party hangs up, connection is released
 - Asymmetric release (when one side breaks connection) is abrupt and may lose data
 - Symmetric release
 - Two separate releases
- Issue
 - How to avoid data loss?
 - Use symmetric release



Connection Release (2)

Symmetric release (both sides agree to release) can't be handled solely by the transport layer

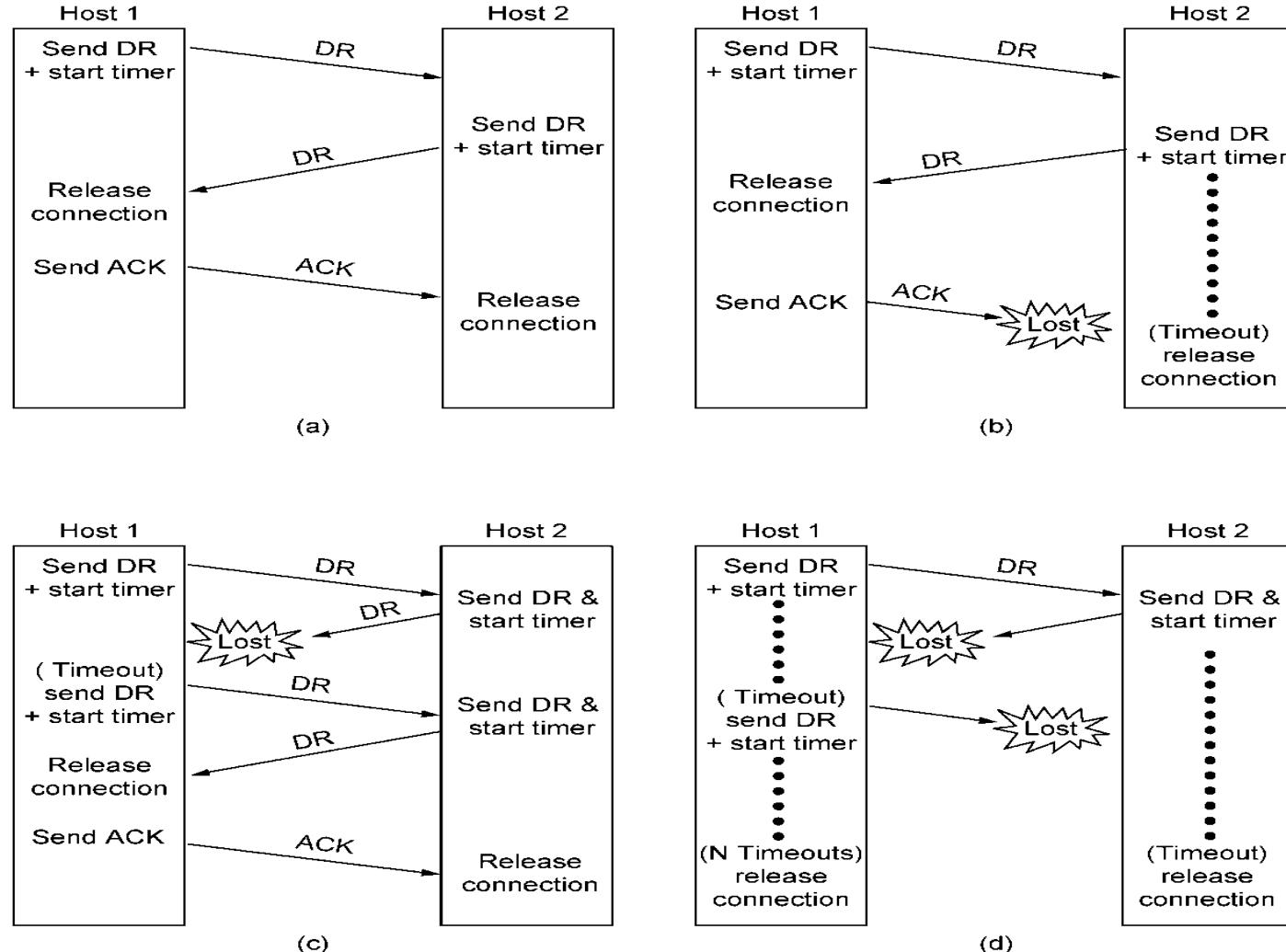
- Two-army problem shows pitfall of agreement
- Blue army units wants to synchronize the attack.
- No one is sure that his reply is got through.
- Three-way/four-way/... will not help.



The two-army problem.

Connection Release (4)

Error cases are handled with timer and retransmission



Four protocol scenarios for releasing a connection. (a) Normal case of three-way handshake. (b) Final lost. (c) Response lost. (d) Response lost and subsequent lost.

Error Control and Flow Control (1)

- Foundation for error control is a sliding window (from Link layer) with checksums and retransmissions
- Flow control manages buffering at sender/receiver
 - Issue is that data goes to/from the network and applications at different times
 - Window tells sender available buffering at receiver
 - Makes a variable-size sliding window

Error Control and Flow Control (1)

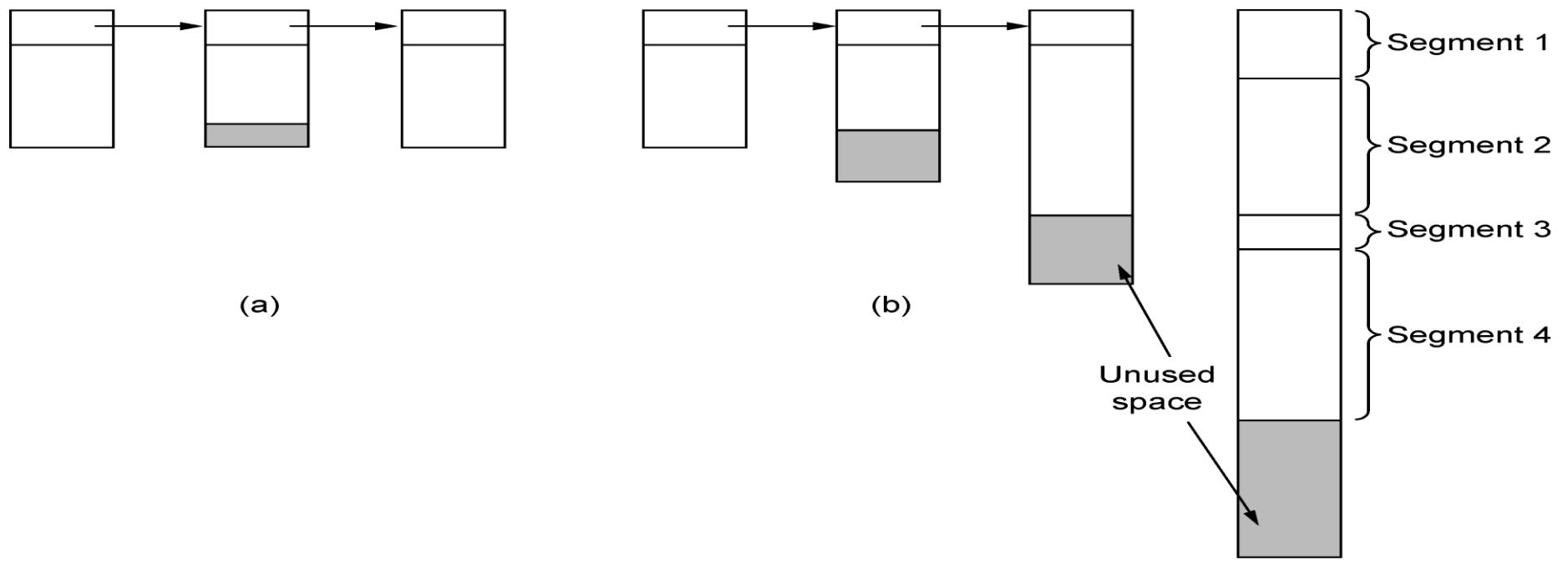
- Recap about data link layer mechanisms
 - A frame carries an error-correcting code (CRC or checksum)
 - A frame carries sequence number to identify itself and retransmitted by sender until receives an ack of successful receipt. This is called ARQ (Automatic Repeat Request)
 - There is a maximum number of frames the sender will allow. If the maximum is one, it is called stop-wait.
 - Sliding window protocol enable bidirectional data transfer.

Error Control and Flow Control (1)

- Difference
 - Link layer checksum protects frames
 - Transport layer checksum protects segments
- A larger sliding window has to be used for Transport layer
 - Host may need a substantial amount of buffer space
 - To hold the data of unacknowledged segments
 - Receiver may not buffer it
- How to organize the buffer pool is the question.
 - Fixed
 - Variable-size
 - Chained large buffer
- Dynamic buffer allocation scheme is used.

Error Control and Flow Control (2)

Different buffer strategies trade efficiency / complexity

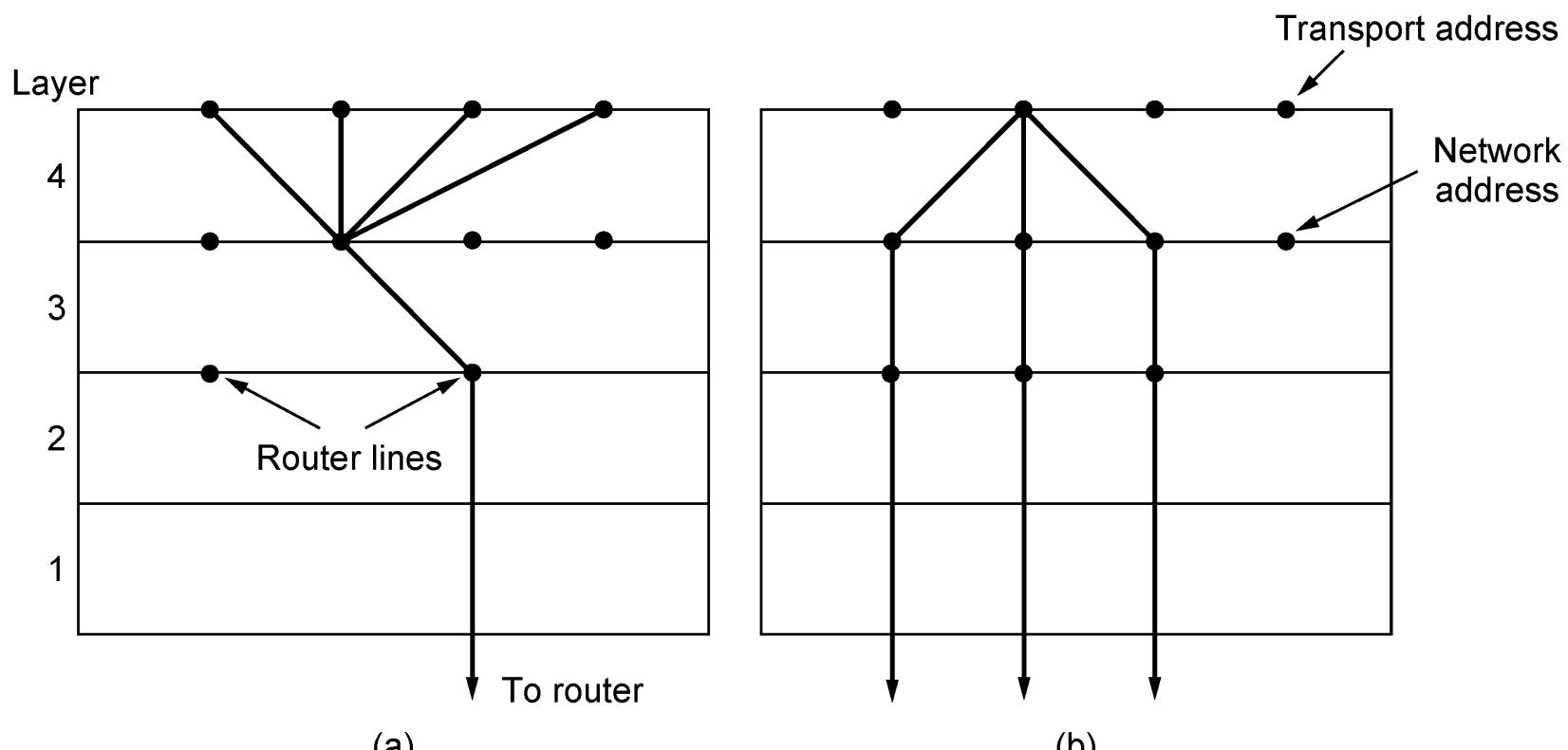


(a) Chained fixed-size buffers. (b) Chained variable-sized buffers. (c) One large circular buffer per connection.

Multiplexing

Kinds of transport / network sharing that can occur:

- Multiplexing: connections share a network address
- Inverse multiplexing: addresses share a connection



(a)Multiplexing. (b)Inverse multiplexing.

Crash Recovery

Application needs to help recovering from a crash

- Transport can fail since A(ck) / W(rite) not atomic
- Difficult problem

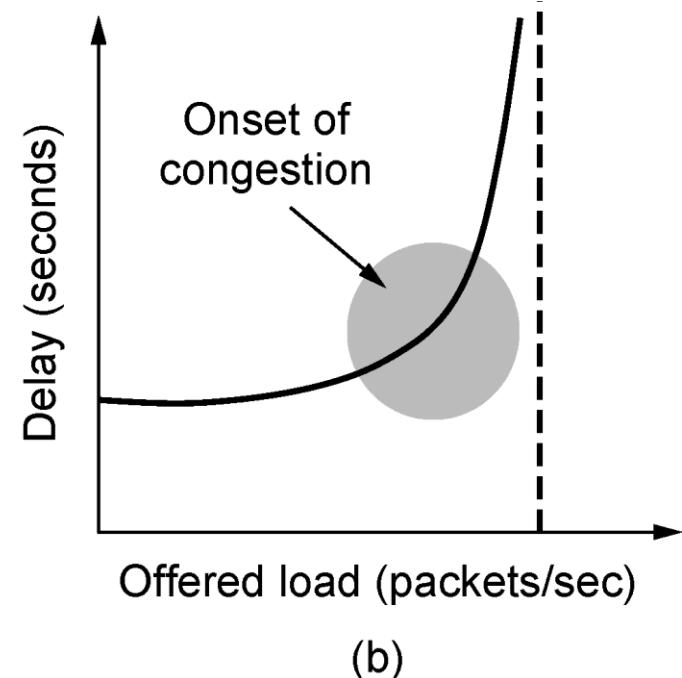
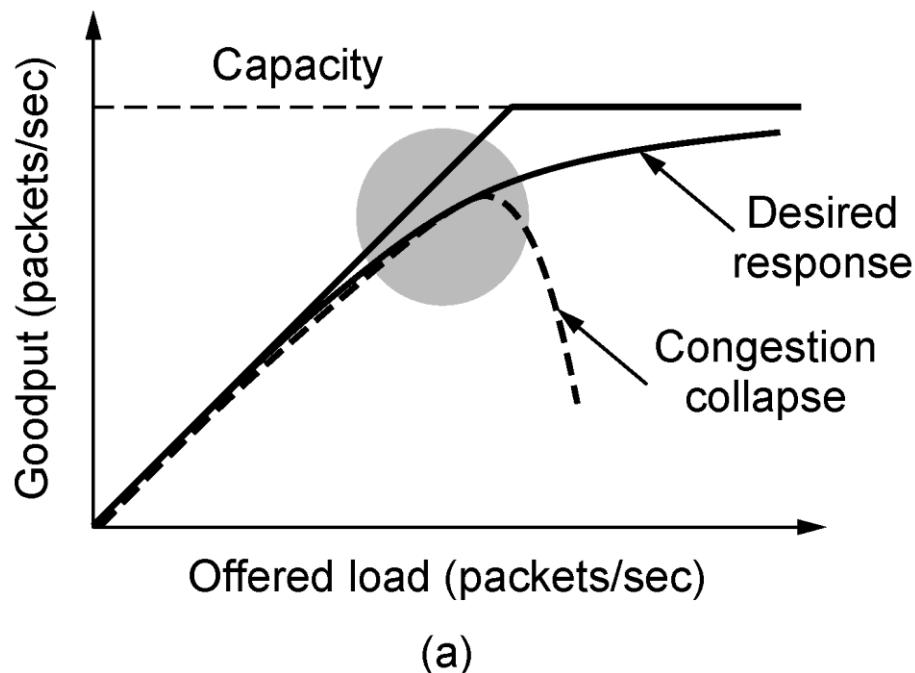
Congestion Control

Two layers are responsible for congestion control:

- Transport layer, controls the offered load [here]
 - Network layer, experiences congestion [previous]
-
- Desirable bandwidth allocation »
 - Regulating the sending rate »
 - Wireless issues »

Desirable Bandwidth Allocation (1)

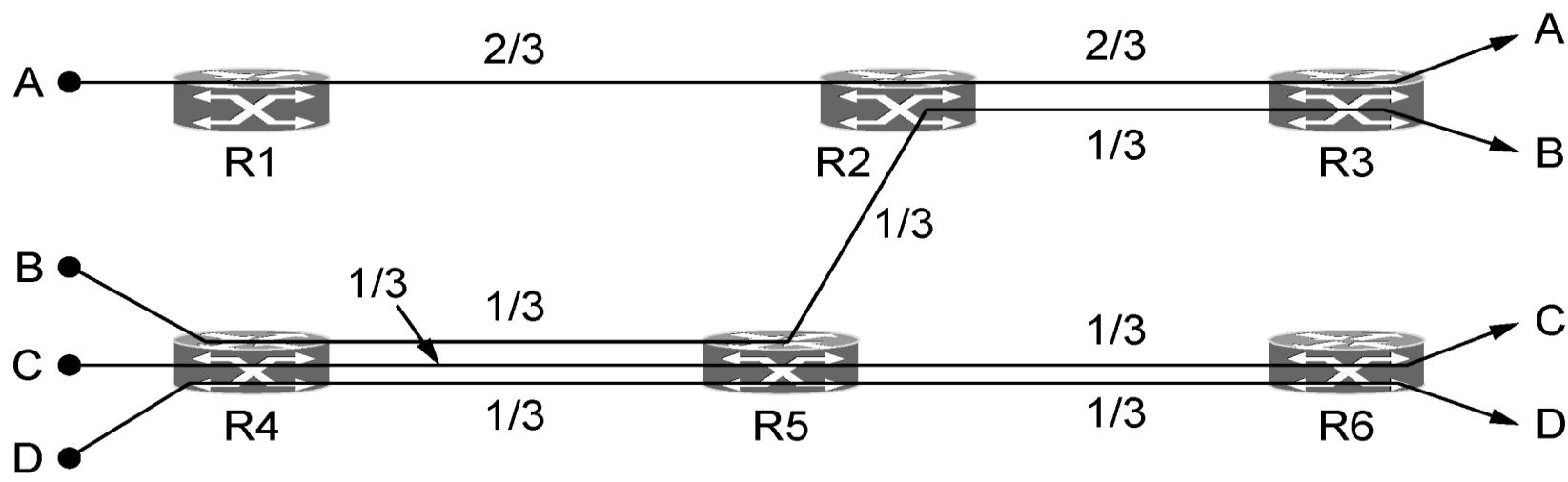
Efficient use of bandwidth gives high goodput, low delay



(a) Goodput and (b) delay as a function of offered load.

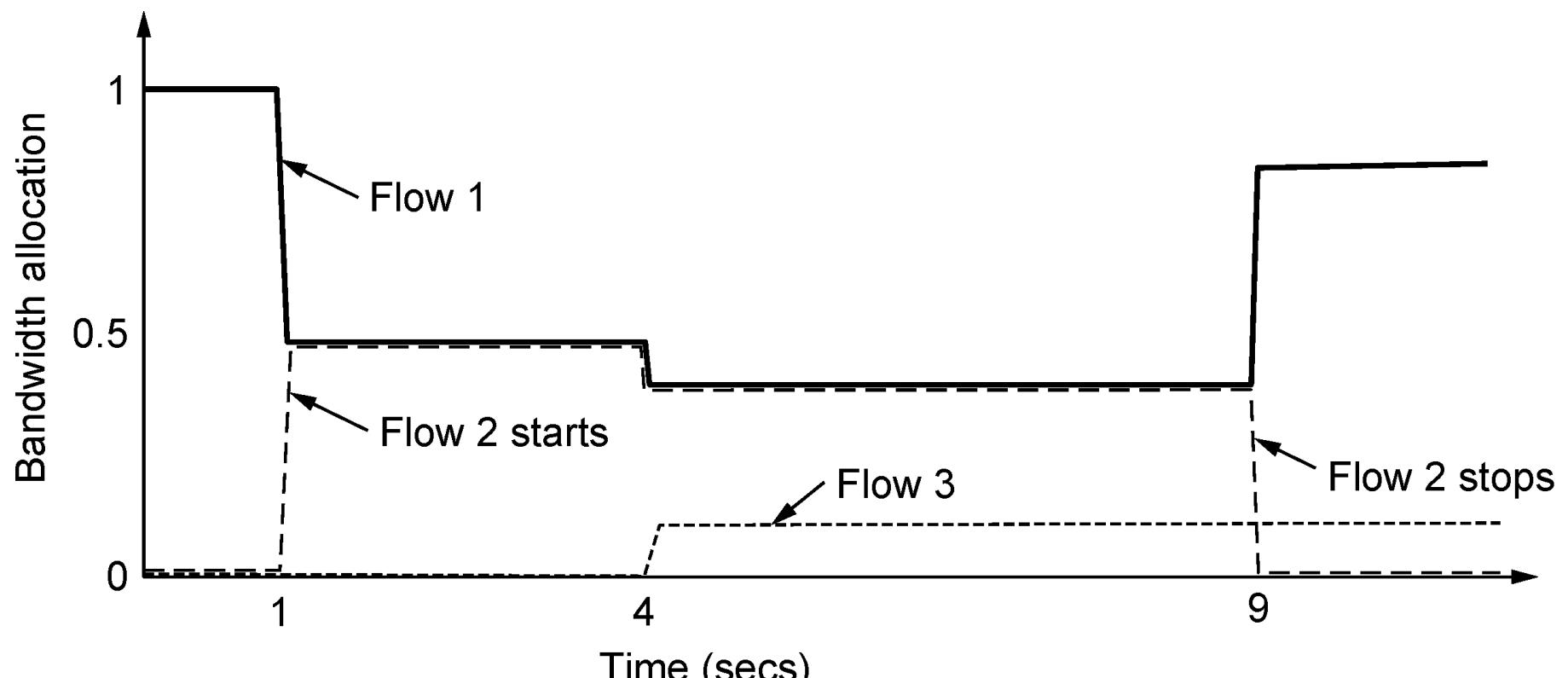
Desirable Bandwidth Allocation (2)

- Fair use gives bandwidth to all flows (no starvation)
 - Max-min fairness gives equal shares of bottleneck
 - An allocation is min-max fair, if the bandwidth given to one flow can not be increased without decreasing the bandwidth given to another flow with the allocation that is no longer.
 - Increase the bandwidth of the flow makes situation worse.



Desirable Bandwidth Allocation (3)

We want bandwidth levels to converge quickly when traffic patterns change

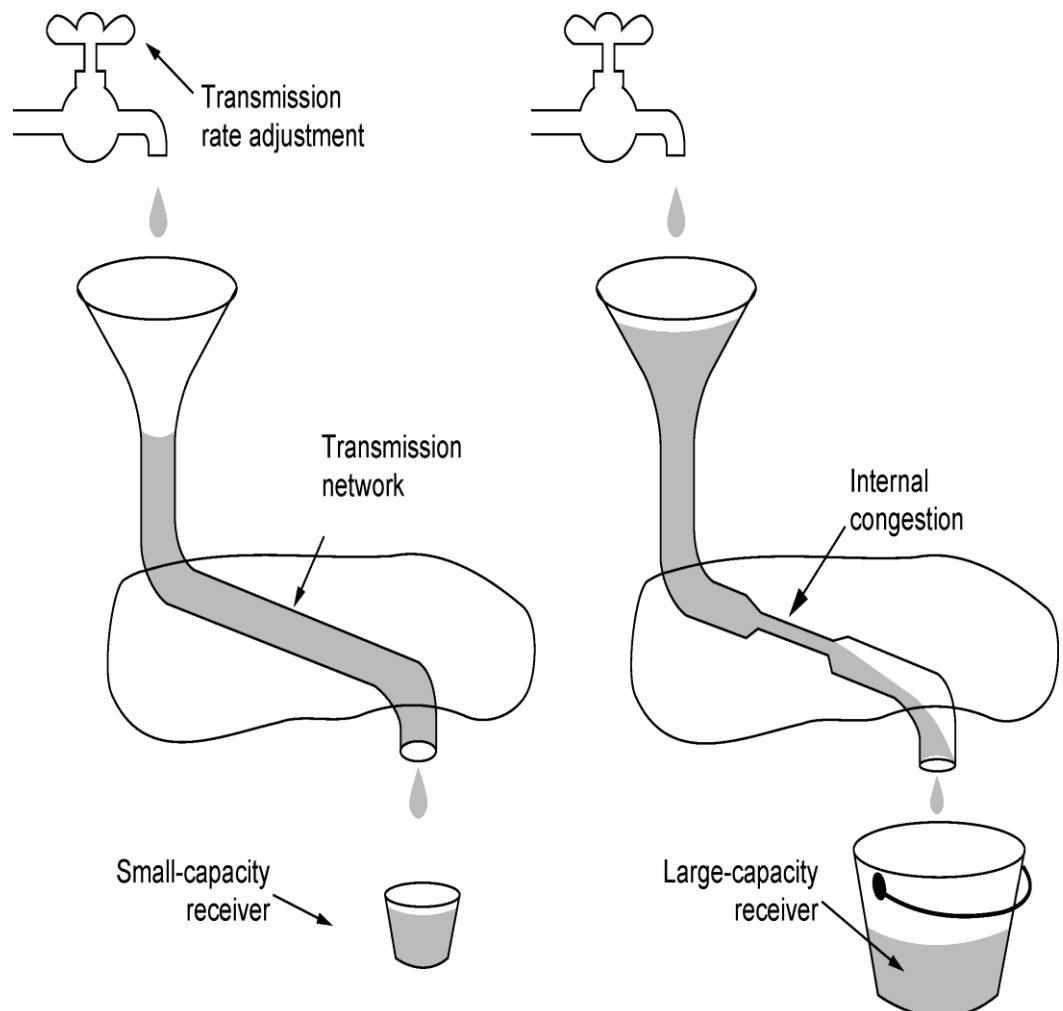


Changing bandwidth allocation over time.

Regulating the Sending Rate (1)

Sender may need to slow down for different reasons:

- Flow control, when the receiver is not fast enough [right]
- Congestion, when the network is not fast enough [over]



- (a) A fast_(a) network feeding a low-capacity_(b) receiver.
- (b) A slow network feeding a high-capacity receiver.

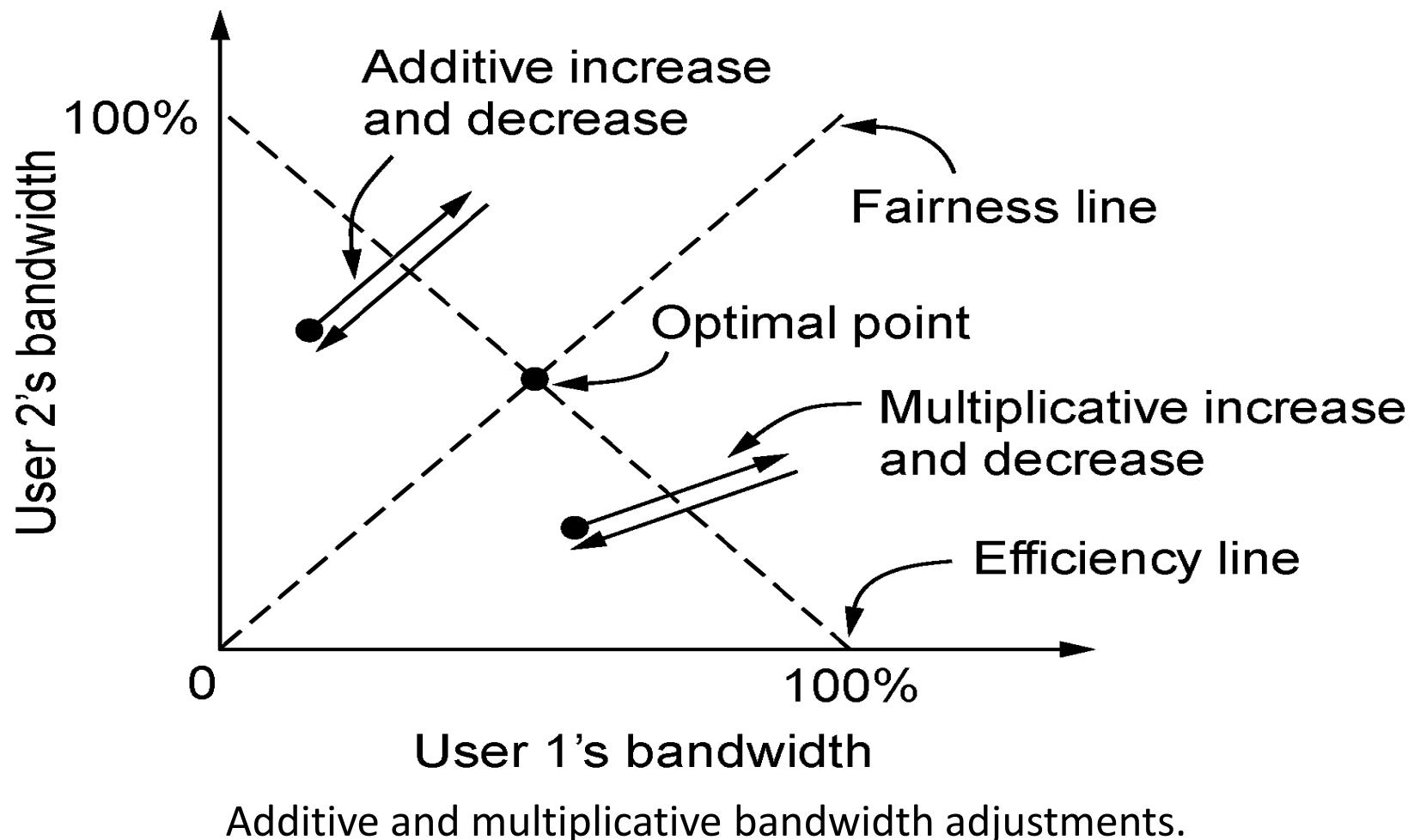
Regulating the Sending Rate (3)

Different congestion signals the network may use to tell the transport endpoint to slow down (or speed up)

| Protocol | Signal | Explicit? | Precise? |
|--------------|--------------------|-----------|----------|
| XCP | Rate to use | Yes | Yes |
| TCP with ECN | Congestion warning | Yes | No |
| FAST TCP | End-to-end delay | No | Yes |
| CUBIC TCP | Packet loss | No | No |
| TCP | Packet loss | No | No |

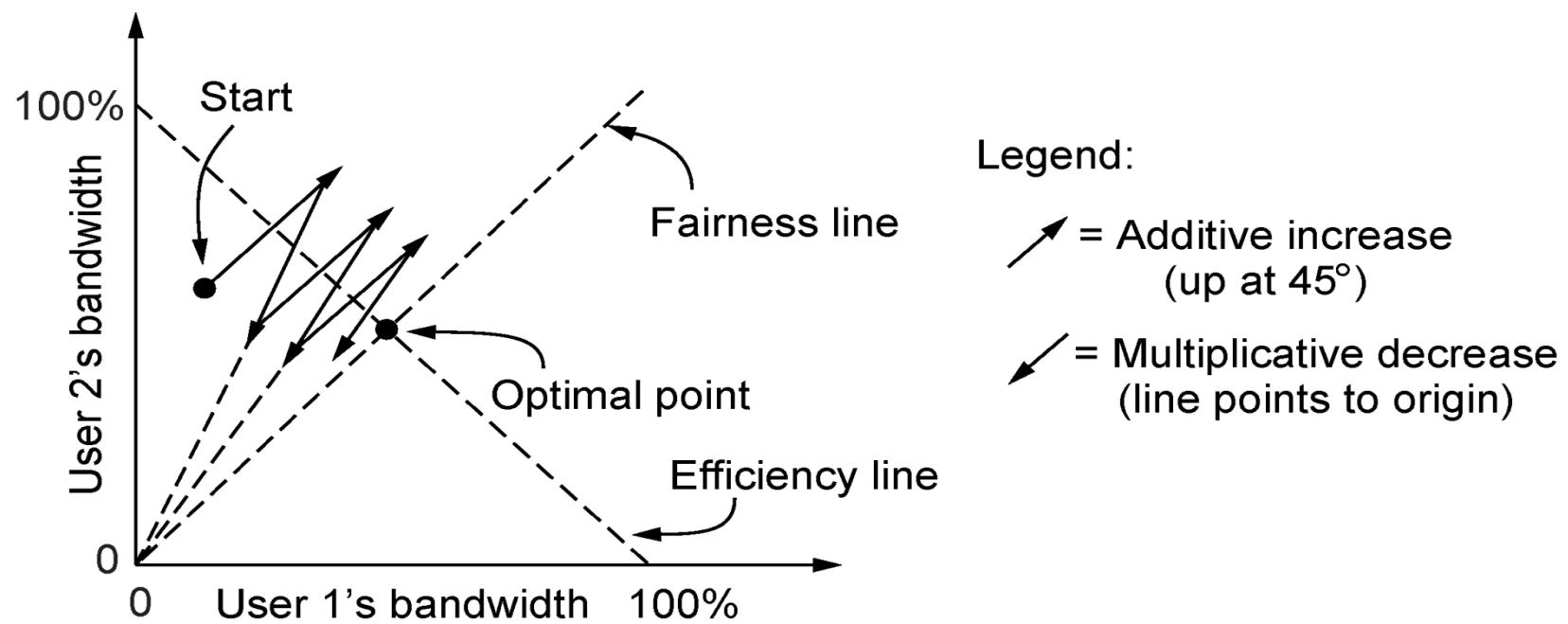
Regulating the Sending Rate (3)

If two flows increase/decrease their bandwidth in the same way when the network signals free/busy they will not converge to a fair allocation.



Regulating the Sending Rate (4)

- The AIMD (Additive Increase Multiplicative Decrease) control law does converge to a fair and efficient point!
 - In the absence of congestion sender should increase the rate
 - In the presence of congestion signal senders to decrease the rate
- TCP uses AIMD for this reason



Additive Increase Multiplicative Decrease (AIMD) control law.

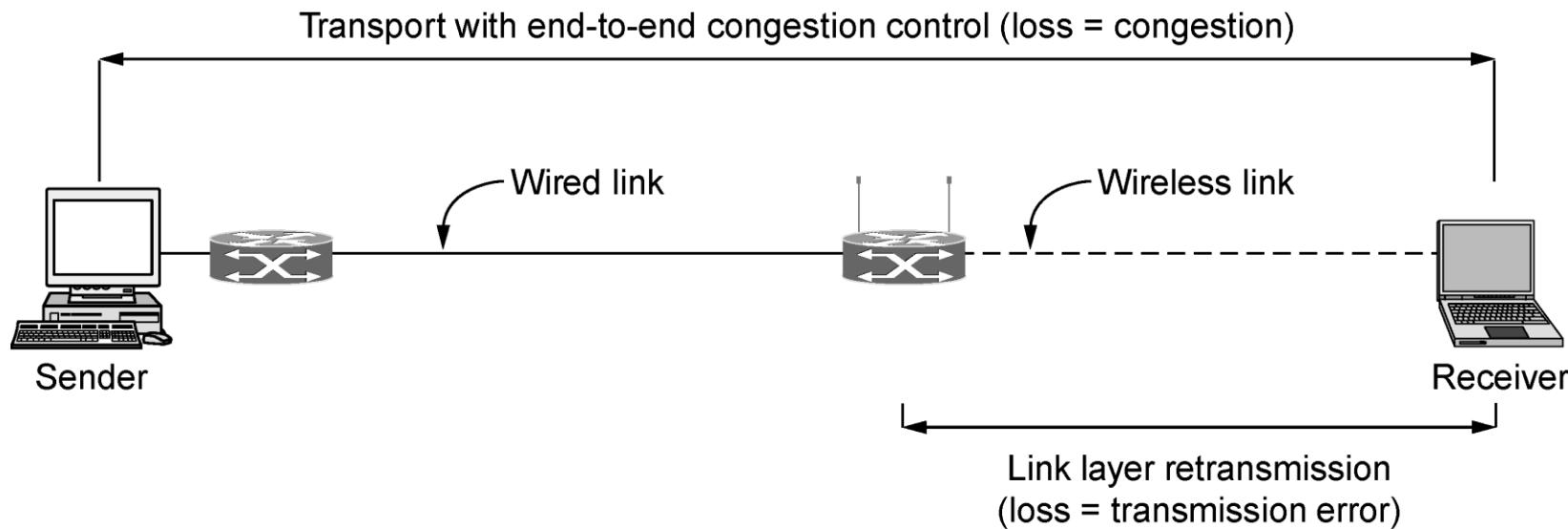
Wireless Issues

Wireless links lose packets due to transmission errors

- Do not want to confuse this loss with congestion
- Or connection will run slowly over wireless links!

Strategy:

- Wireless links use ARQ, which masks errors



Congestion control over a path with a wireless link.

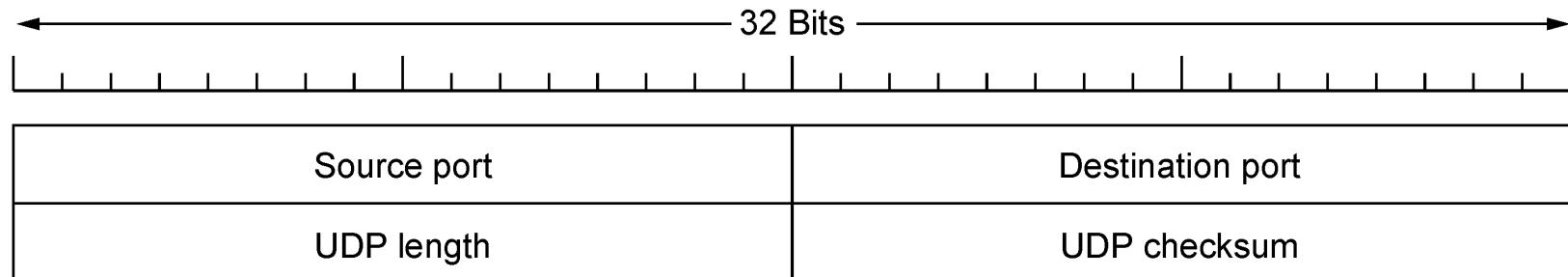
Internet Protocols – UDP

- Introduction to UDP »
- Remote Procedure Call »
- Real-Time Transport »

Introduction to UDP (1)

UDP (User Datagram Protocol) is a shim over IP

- Header has ports (TSAPs), length and checksum.

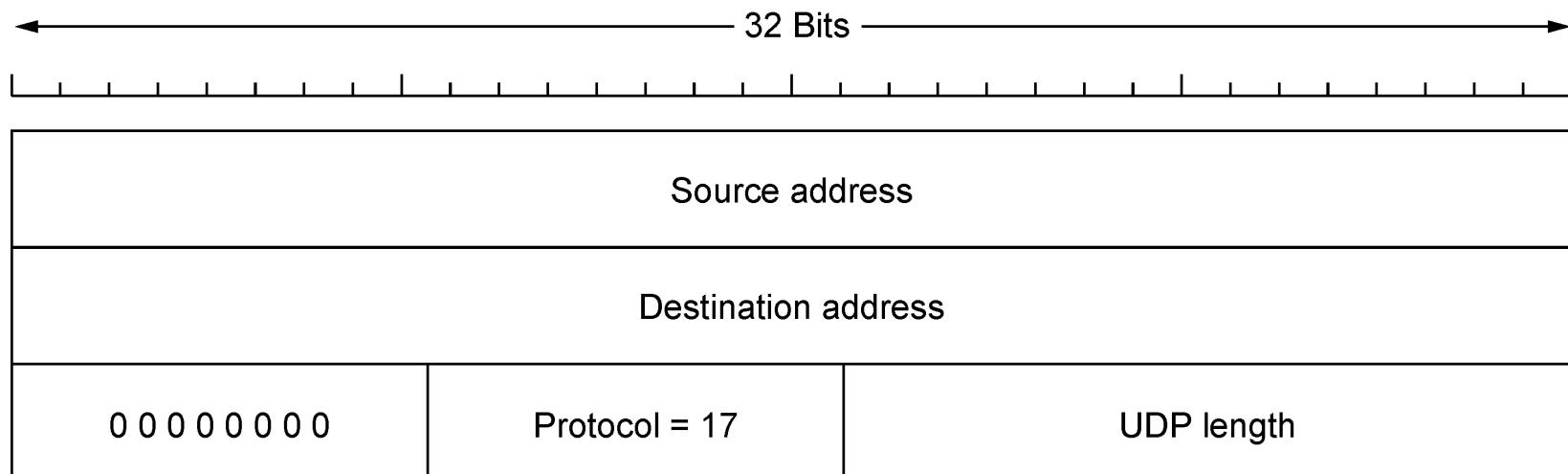


The UDP header.

Introduction to UDP (2)

Checksum covers UDP segment and IP pseudoheader

- Fields that change in the network are zeroed out
- Provides an end-to-end delivery check

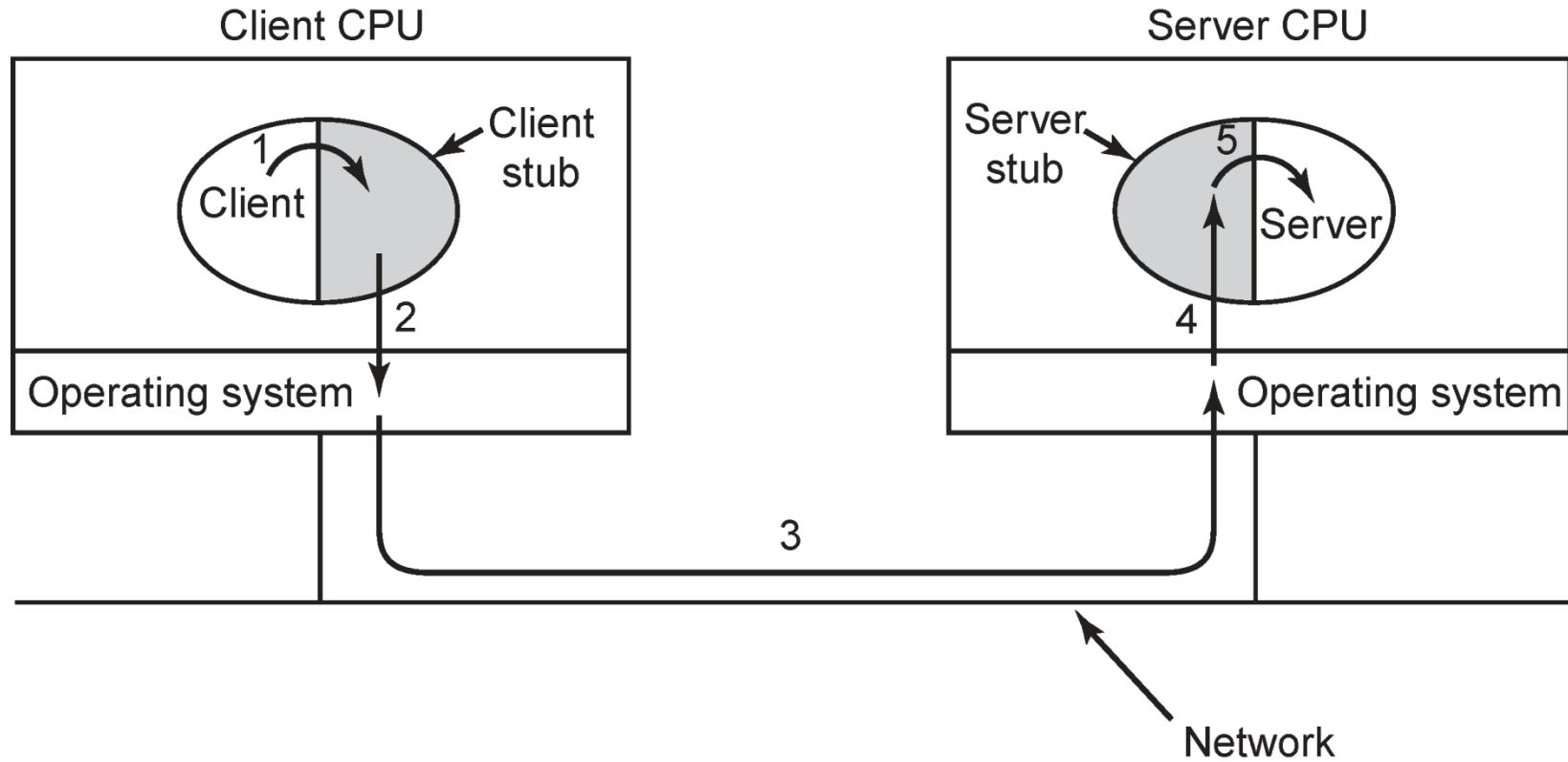


The IPv4 pseudoheader included in the UDP checksum.

RPC (Remote Procedure Call)

RPC connects applications over the network with the familiar abstraction of procedure calls

- Stubs package parameters/results into a message
- UDP with retransmissions is a low-latency transport

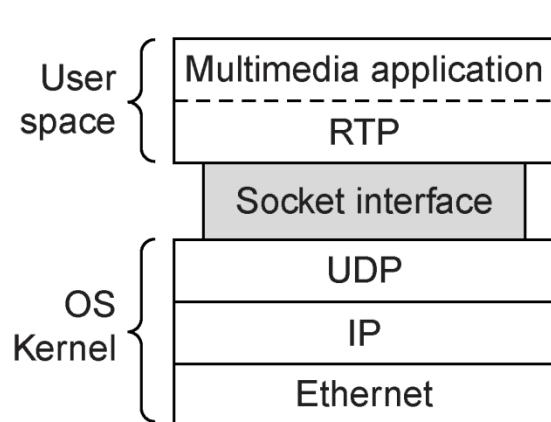


Steps in making a remote procedure call. The stubs are shaded.

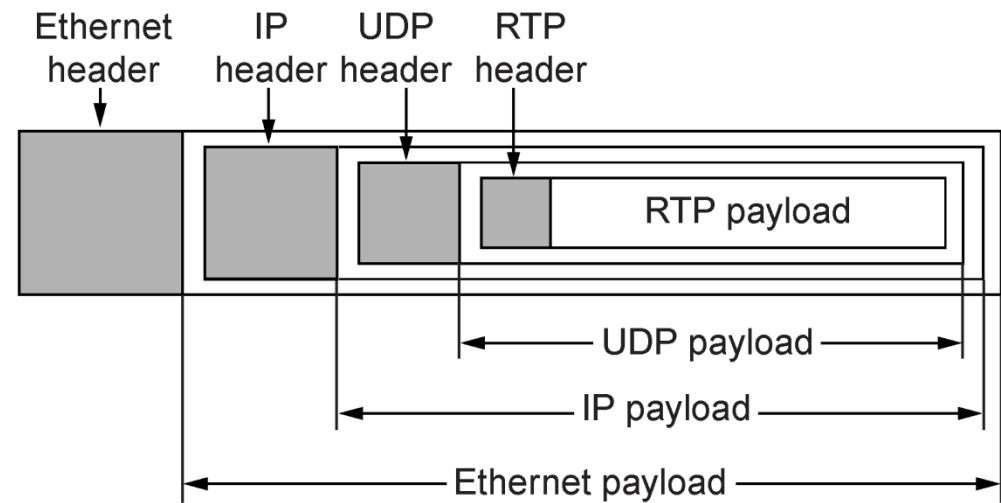
Real-Time Transport (1)

RTP (Real-time Transport Protocol) provides support for sending real-time media over UDP

- Often implemented as part of the application



(a)



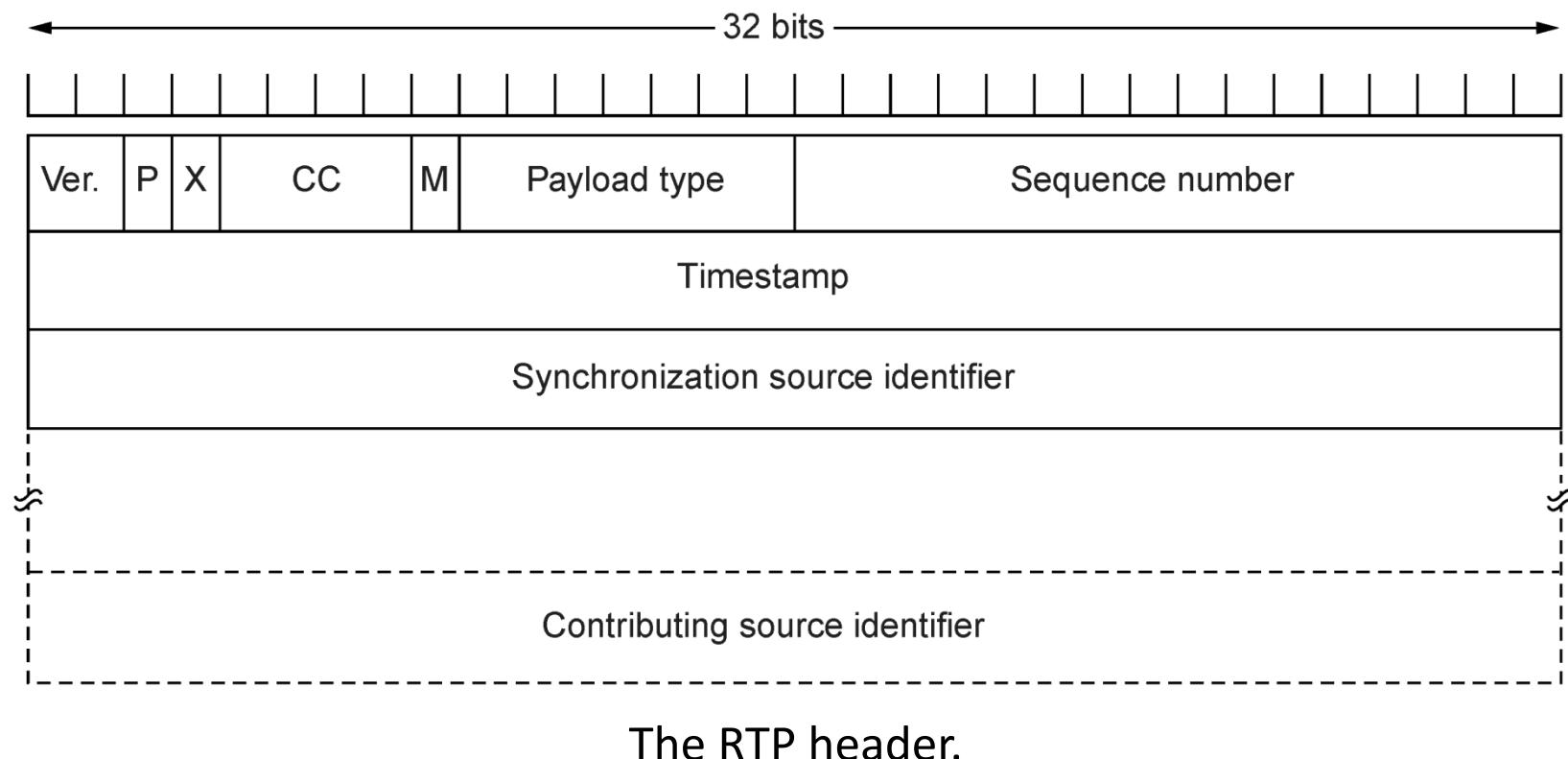
(b)

(a) The position of RTP in the protocol stack. (b) Packet nesting.

Real-Time Transport (2)

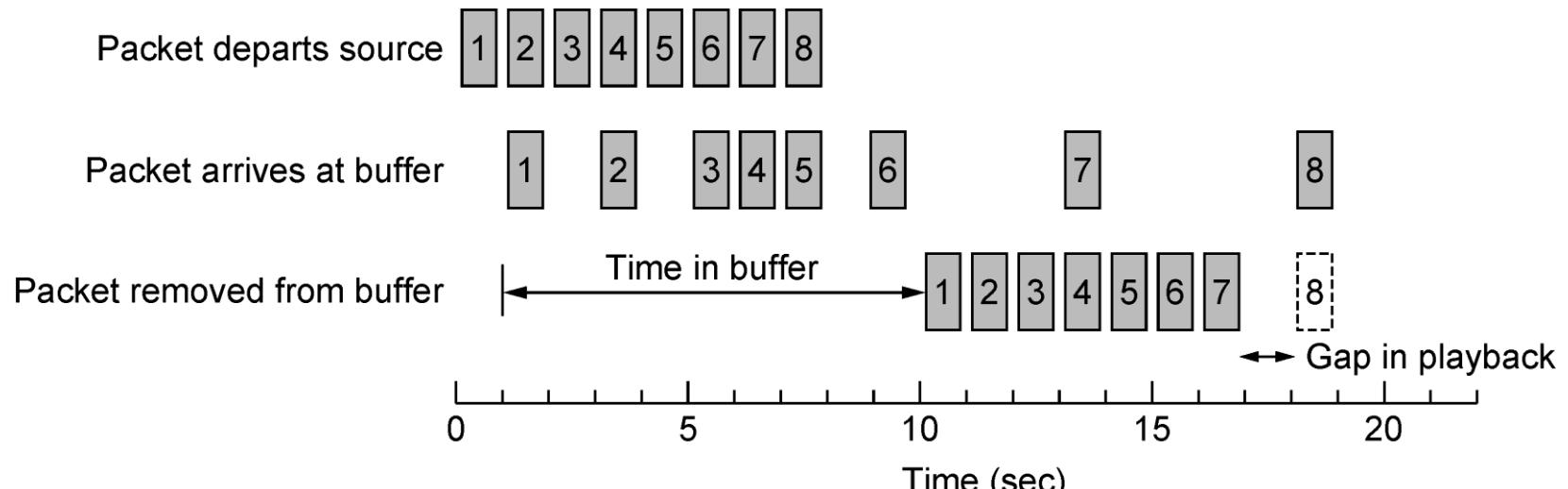
RTP header contains fields to describe the type of media and synchronize it across multiple streams

- RTCP sister protocol helps with management tasks



Real-Time Transport (3)

Buffer at receiver is used to delay packets and absorb jitter so that streaming media is played out smoothly

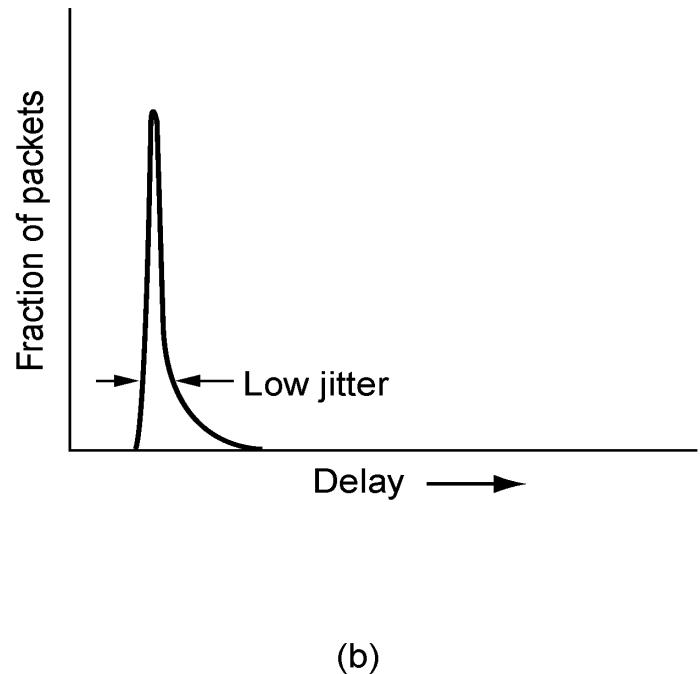
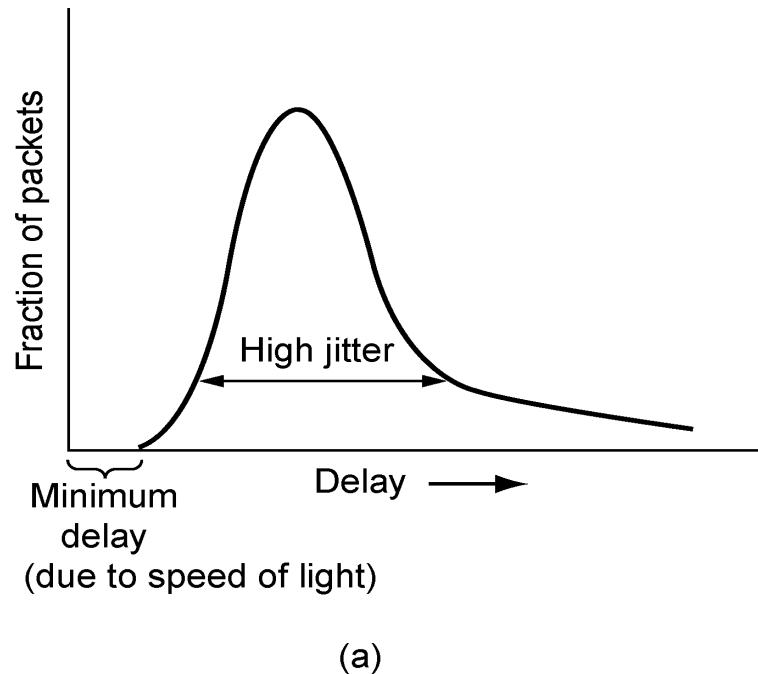


Smoothing the output stream by buffering packets.

Real-Time Transport (3)

High jitter, or more variation in delay, requires a larger playout buffer to avoid playout misses

- Propagation delay does not affect buffer size



(a) High jitter. (b) Low jitter.

The Internet Transport Control Protocols: TCP

- TPC (transport Control Protocol) is designed to provide reliable connection over unreliable internetwork.
- Internetwork
 - Different topologies, bandwidths, delays, packet sizes, and other parameters.
 - TCP is designed to be robust against many types of failures.
- The TCP stream may accept user data stream from local processes, breaks them up into pieces not exceeding 64KB (fits in the Ethernet frame) and sends this piece as a separate TCP datagram.
- When datagrams arrive at a machine, they are given to the TCP entity, which reconstructs the original byte streams.
- The IP layer gives no guarantee that datagrams will be delivered properly and any indication of how fast these will be delivered.

The Internet Transport Control Protocols: TCP

- It is up to the TCP to send datagrams fast enough to make enough capacity but not cause congestion and to timeout and retransmit messages that are not delivered.
- It should also assemble in the order.

The Internet Transport Control Protocols: TCP

- The TCP service model »
- The TCP segment header »
- TCP connection establishment »
- TCP connection state modeling »
- TCP sliding window »
- TCP timer management »
- TCP congestion control »

The TCP Service Model (1)

- TCP service is obtained by both the sender and the receiver creating end points called sockets.
- Each socket has a socket number consisting of of IP address and port number of the machine.
- Port is the TCP name for a TSAP.
- A connection must be explicitly established between a socket on one machine and a socket on other machine.
- Port numbers below 1024 are reserved for well-known services.
 - They can only be started by privileged users
- Other port numbers 1024 to 49151 are reserved for unprivileged users.

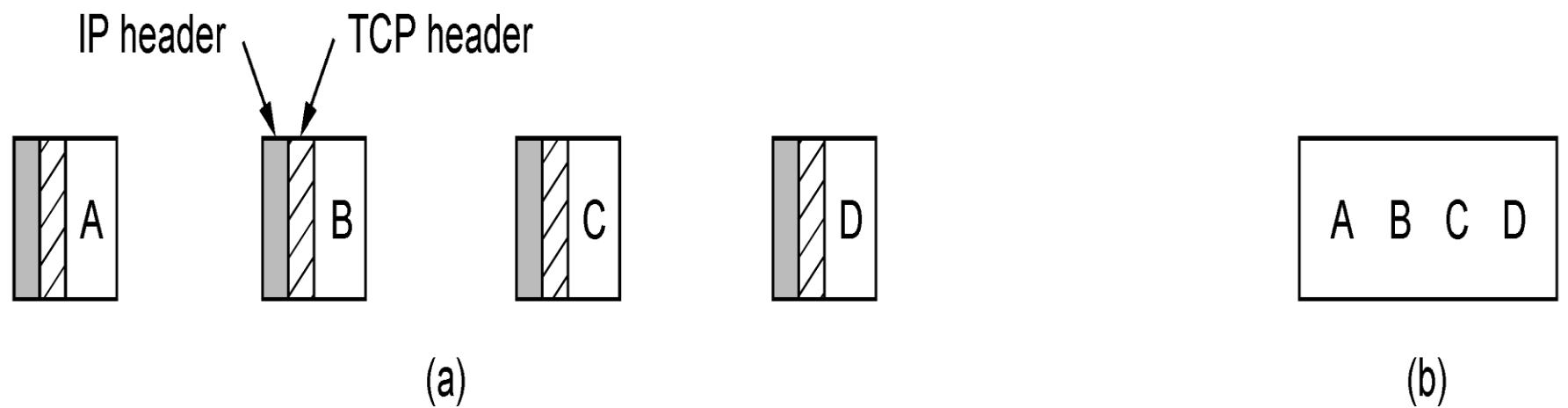
The TCP Service Model (1)

- TCP provides applications with a reliable byte stream between processes; it is the workhorse of the Internet
 - Popular servers run on well-known ports

| Port | Protocol | Use |
|--------|----------|--------------------------------------|
| 20, 21 | FTP | File transfer |
| 22 | SSH | Remote login, replacement for Telnet |
| 25 | SMTP | Email |
| 80 | HTTP | World Wide Web |
| 110 | POP-3 | Remote email access |
| 143 | IMAP | Remote email access |
| 443 | HTTPS | Secure Web (HTTP over SSL/TLS) |
| 543 | RTSP | Media player control |
| 631 | IPP | Printer sharing |

The TCP Service Model (2)

- Applications using TCP see only the byte stream [right] and not the segments [left] sent as separate IP packets



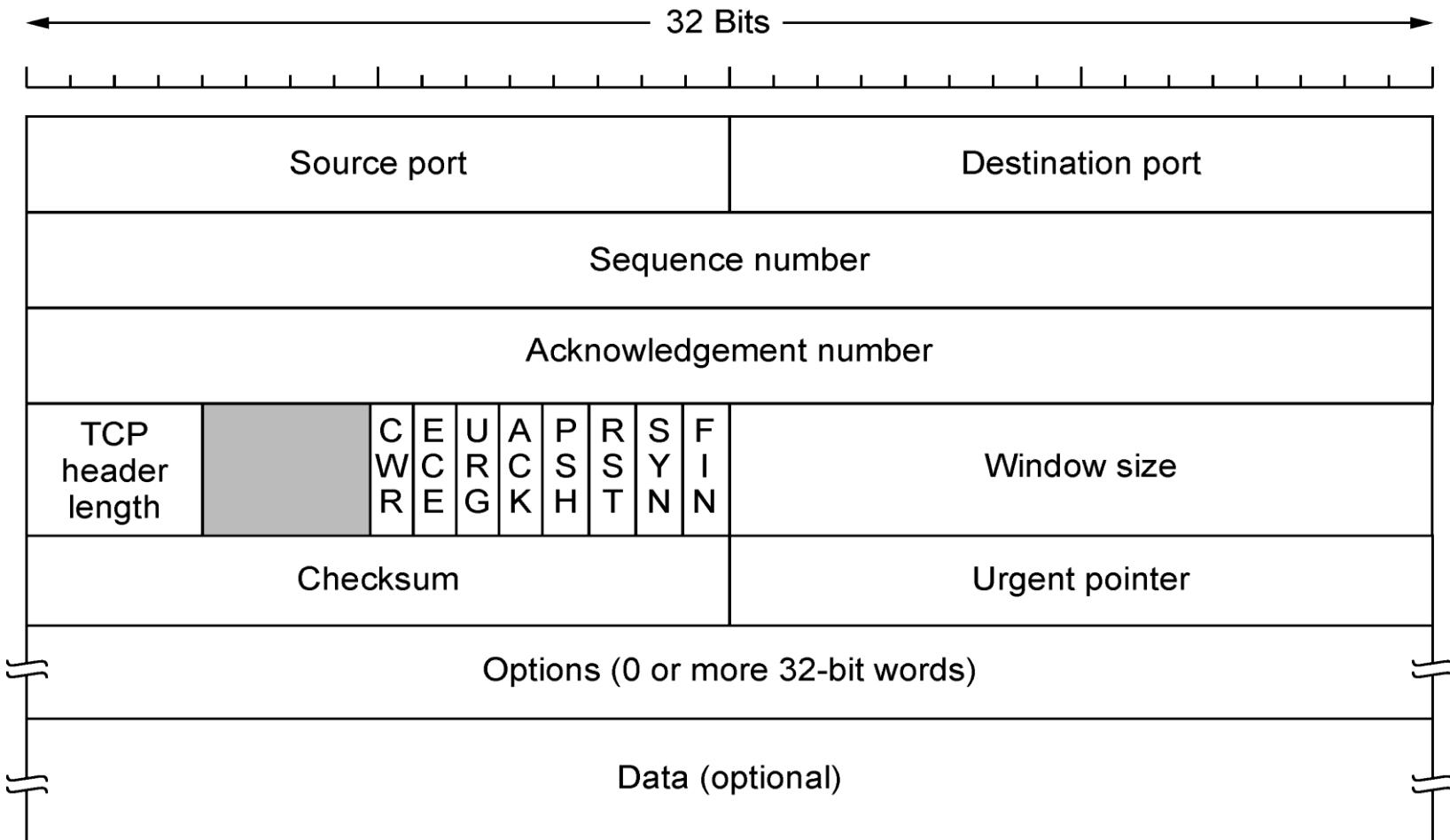
(a) Four 512-byte segments sent as separate IP datagrams. (b) The 2048 bytes of data delivered to the application in a single call.

Overview of The TCP Protocol

- Data is exchanged in the form is segments.
- Each segment has 20 byte header +optional part followed by data.
- Each segment must fit into 65515 byte payload (including header).
- Basic protocol
 - Sliding window protocol with a dynamic window size
 - When a sender transmits a segment, it also starts a timer.
 - The receiver send ack segment with the expected sequence number.
 - If the sender timer goes off, before the ack is received, the sender retransmits the message again.
- Issues
 - Out of order arrival of segments
 - Sender times out and retransmits

The TCP Segment Header

- TCP header includes addressing (ports), sliding window (seq. / ack. number), flow control (window), error control (checksum) and more.



The TCP header.

The TCP Segment Header

- Port addresses
- Sequence number
- Ack number
- TCP header length: number of 32-bit words in TCP header.
- 1 bit flags
 - CWR and ECE are used for signal congestion when ECN is used.
 - ECE is set to signal an ECN-Echo to a TCP sender.
 - CWR (Congestion Window Reduced) is set by sender to inform the receiver that it has reduced the load and can stop sending ECN-Echo
 - URG is set to 1 if the urgent pointer is used.
 - ACK is set to one to indicate ACK is valid.
 - PSH indicates pushed data. Do not wait for complete data.
 - RST bit is set to abruptly reset connection.
 - SYN bit is used to establish connection.
 - FIN bit is used to release connection.

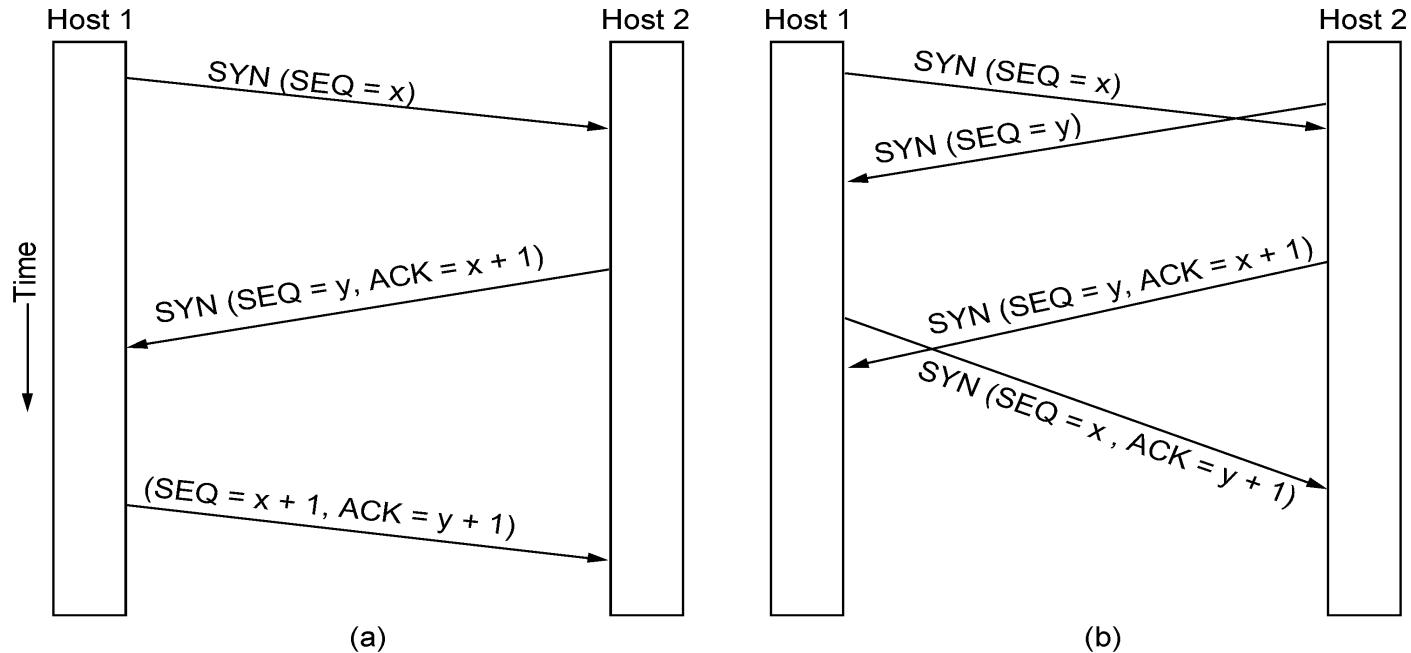
Overview of The TCP Protocol

- **Window-size (WIN)** tells how many bytes may be sent starting at the byte acknowledged.
 - If WIN=0, the sender must stop.
- Options
 - MSS (Maximum segment size)
 - Using larger segment is better as the header data can be amortized over more data
 - Timestamp
 - SACK (Selective acknowledgement)
 - Lets the receiver to send a ranges of sequence numbers that it has received.

TCP Connection Establishment

TCP sets up connections with the three-way handshake

- Release is symmetric, also as described before



(a) TCP connection establishment in the normal case.

(b) Simultaneous connection establishment on both sides.

TCP Connection Release

- To release a connection, either party can send a TCP segment with the FIN bit set.
- When FIN is acknowledged, the direction is shutdown.
 - Other direction may continue.
- To avoid two-army problem, timers are used
 - If a response to a FIN is not forthcoming within two packet lifetimes, the sender pf FIN releases the connection.
 - The other side will time out as well.

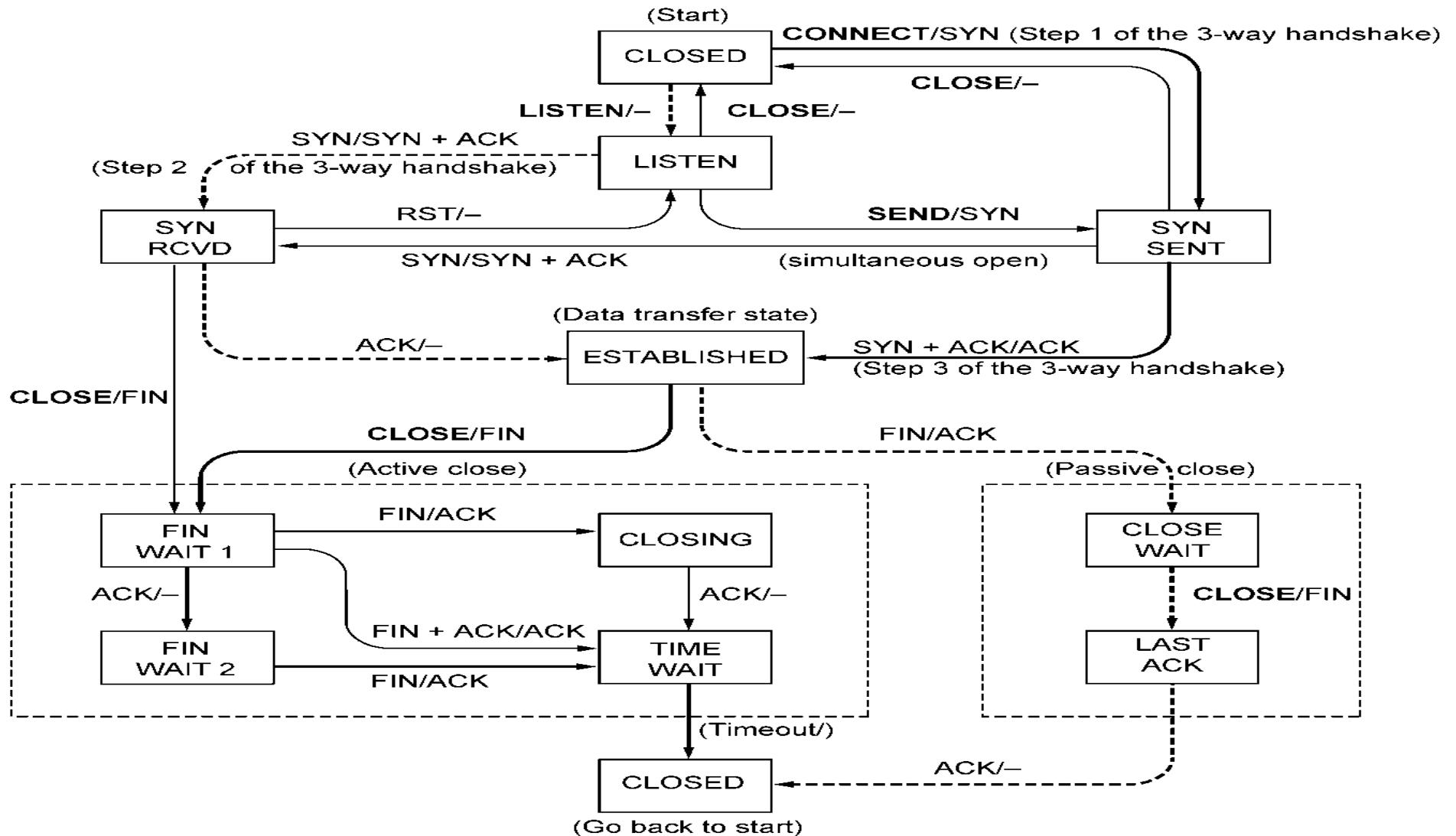
TCP Connection State Modeling (1)

The TCP connection finite state machine has more states than our simple example from earlier.

| State | Description |
|-------------|--|
| CLOSED | No connection is active or pending |
| LISTEN | The server is waiting for an incoming call |
| SYN RCVD | A connection request has arrived; wait for ACK |
| SYN SENT | The application has started to open a connection |
| ESTABLISHED | The normal data transfer state |
| FIN WAIT 1 | The application has said it is finished |
| FIN WAIT 2 | The other side has agreed to release |
| TIME WAIT | Wait for all packets to die off |
| CLOSING | Both sides have tried to close simultaneously |
| CLOSE WAIT | The other side has initiated a release |
| LAST ACK | Wait for all packets to die off |

The states used in the TCP connection management finite state machine.

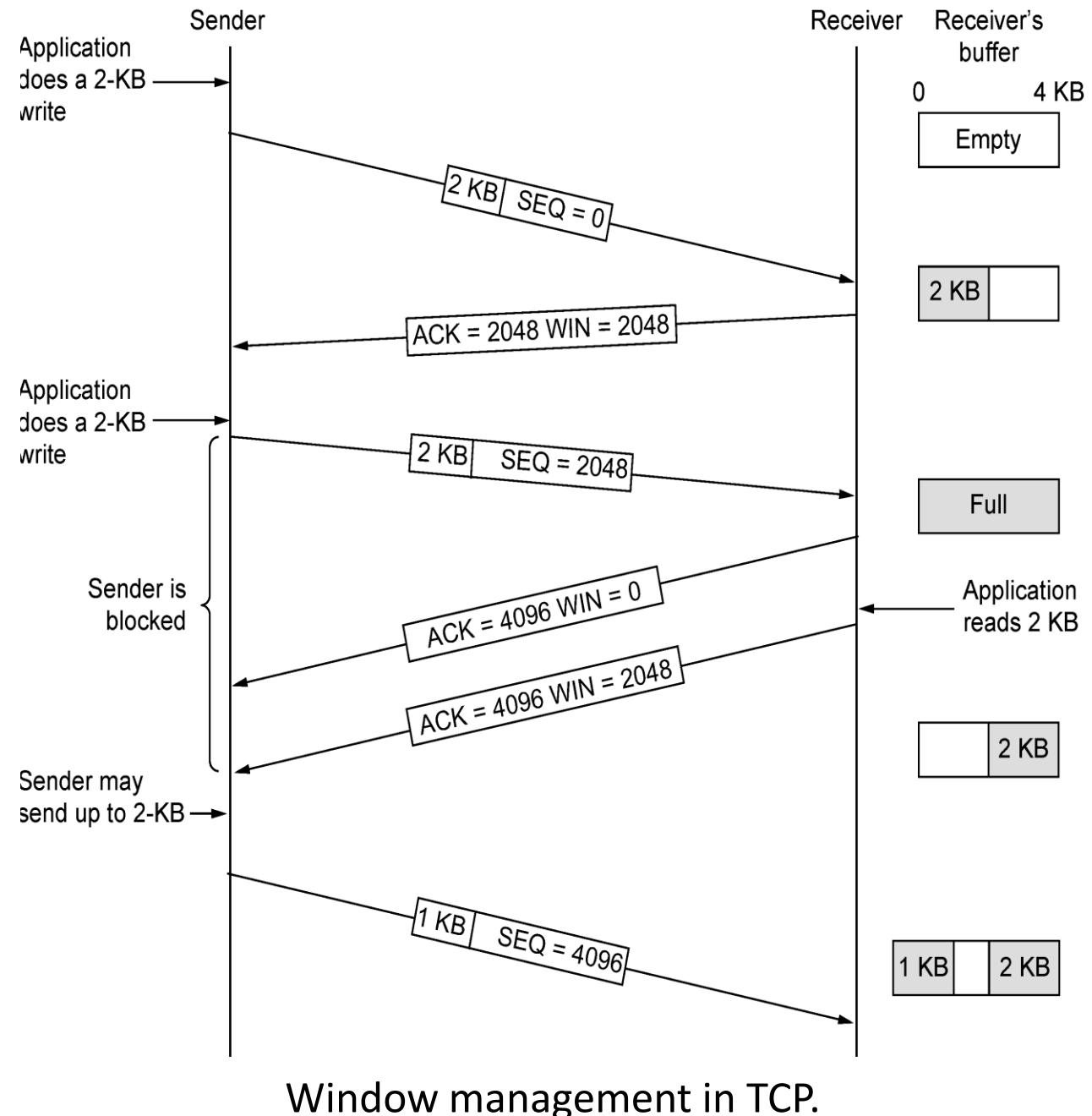
TCP Connection State Modeling (2)



TCP connection management finite state machine. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events. Each transition is labeled with the event causing it and the action resulting from it, separated by a slash.

TCP Sliding Window (1)

- TCP adds flow control to the sliding window as before
- ACK + WIN is the sender's limit



TCP Sliding Window

- If the sender transmits 2048 data, receiver stores in the buffer and sends ack.
 - Sender sends ACK and $WIN=2048$
 - Sender sends another 2KB.
 - As the buffer is full, receiver sends ack with $WIN=0$
 - Sender should stop sending
 - Sender resumes after receiver send ACK with $WIN=2048$.
-
- Window probe
 - Sender may send 1-byte segment to force the receiver to renounce the next byte expected.
 - This is to prevent the deadlock, if the window updates ever gets lost.

TCP Sliding Window..

- Senders need not transmit data as soon as it receives from the application.
- Receivers need not send ACKs as soon as possible.
- This freedom can be used to improve performance.
- Issue:
 - some times each character creates a 21 byte segment, which it gives to IP 41-byte IP datagram.
 - At the receiver side, TCP immediately sends a 40-byte ack.
 - 162 bytes of bandwidth are used and four segments are sent for each character typed.

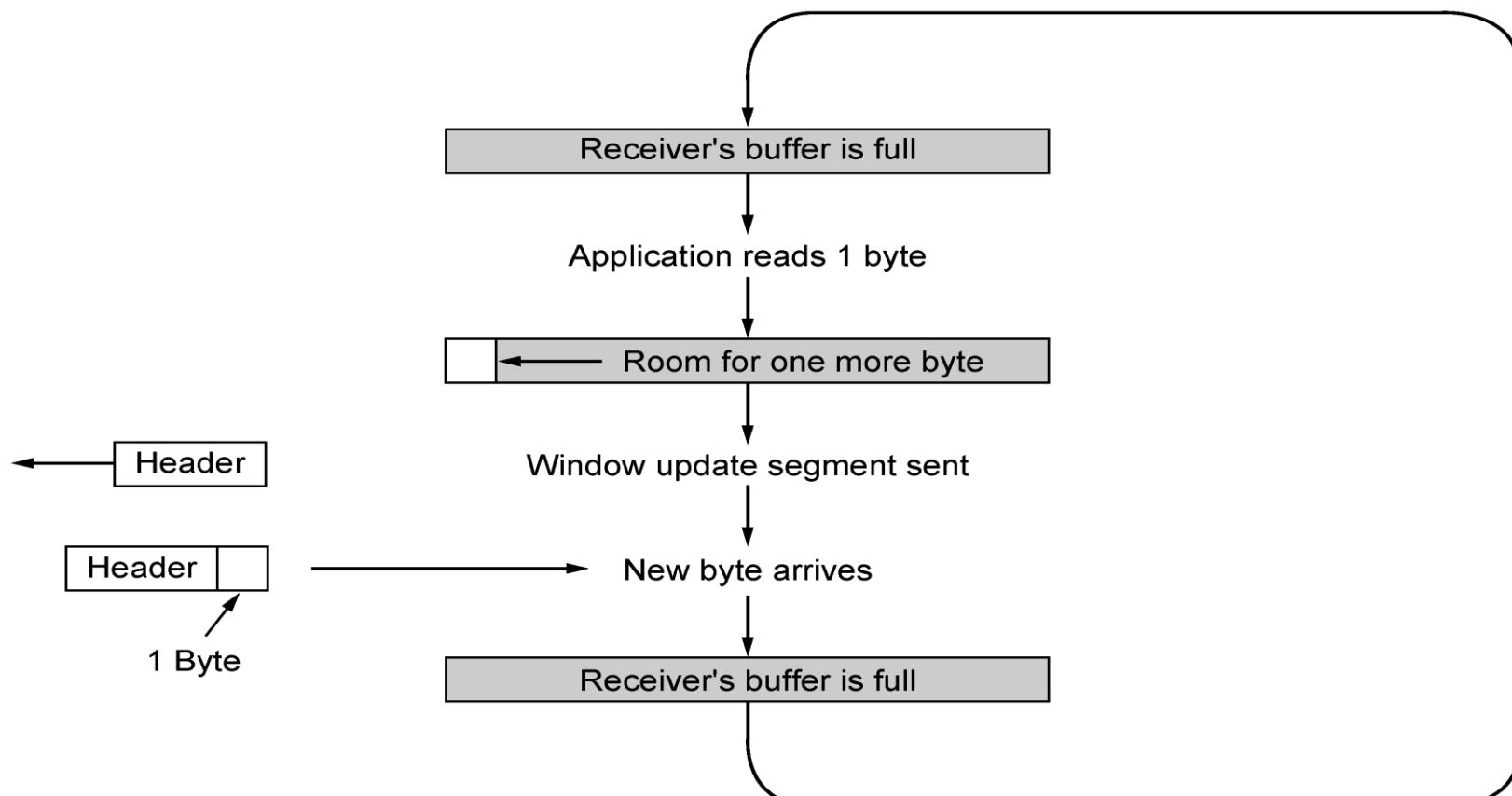
TCP Sliding Window..

- Solution: Delayed acks.
 - Delay ack up to 500 msec.
 - But if sender sends small packets, we will have the same problem.
- Nagle's algorithm
 - Send the first piece and buffer the rest until the first piece is ack.
 - Send all the buffered data in one TCP segment.
 - This is widely used algorithm, but for some applications, like interactive games, it is not applicable.
- Silly window syndrome
 - Receiving end only processes one byte at a time.
- Clark's algorithm
 - Wait until decent amount of space is available and advertise that instead.

TCP Sliding Window (2)

Need to add special cases to avoid unwanted behavior

- E.g., silly window syndrome [below]



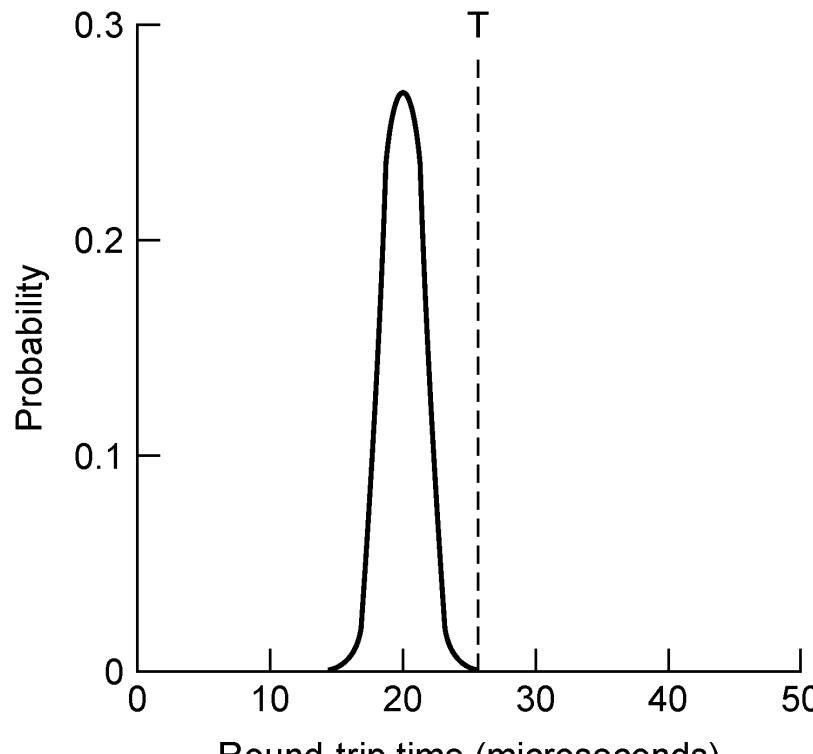
Silly window syndrome.

Receiver application reads single bytes, so sender always sends one byte segments

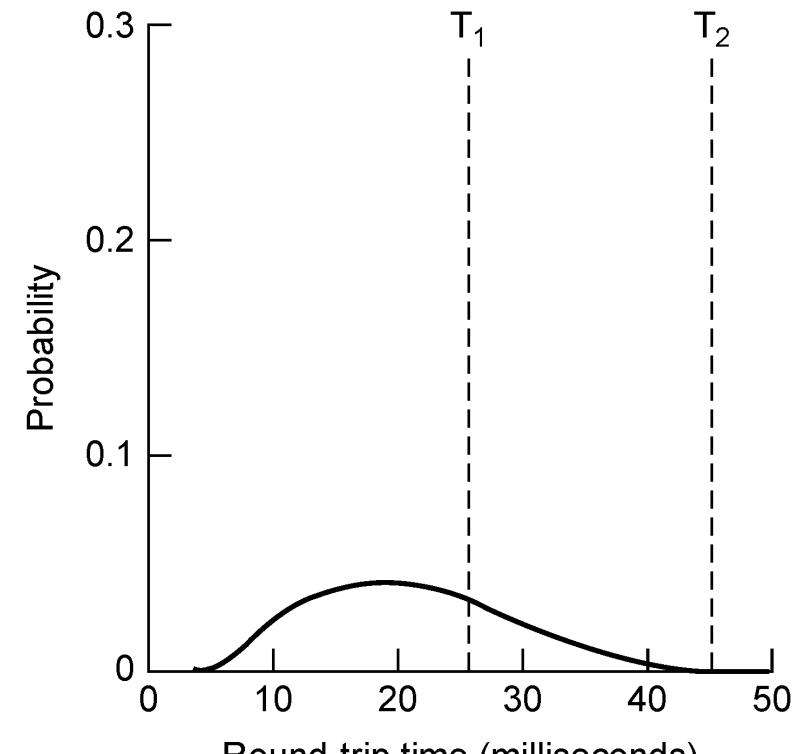
TCP Timer Management

TCP estimates retransmit timer from segment RTTs

- Tracks both average and variance (for Internet case)
- Timeout is set to average plus $4 \times$ variance



(a)



(b)

- (a) Probability density of acknowledgement arrival times in the data link layer.
(b) Probability density of acknowledgement arrival times for TCP.

TCP Timer Management

- Probability density function varies significantly.
- Variance is significant.
 - Things can change quickly.
- Change T based on the performance
- Jacobson algorithm
 - TCP maintains a variable SRTT (Smoothened round trip time)
 - TCP measures how long the ack took
 - It updates $SRTT = \alpha SRTT + (1 - \alpha) SRTT$
 - The formula is called EWMA (exponentially Weighted Moving Average).
- Improved formula
 - Sensitive to variance in round trip times

TCP Timer Management

- Two more timers
- Persistence timer
 - Receiver sends ack asking sender to wait.
 - Later receiver updates the window, the packet with the update is lost
 - Situation: Both sender and receiver are waiting.
 - When persistence time goes off, the sender retransmits the probe. If it is still 0, the timer is initialized and cycle repeats
- Keepalive timer
 - If the connection is idle for a long time, one side will check whether the other side is live or not. If it fails to respond, the connection is terminated.

TCP Congestion Control (1)

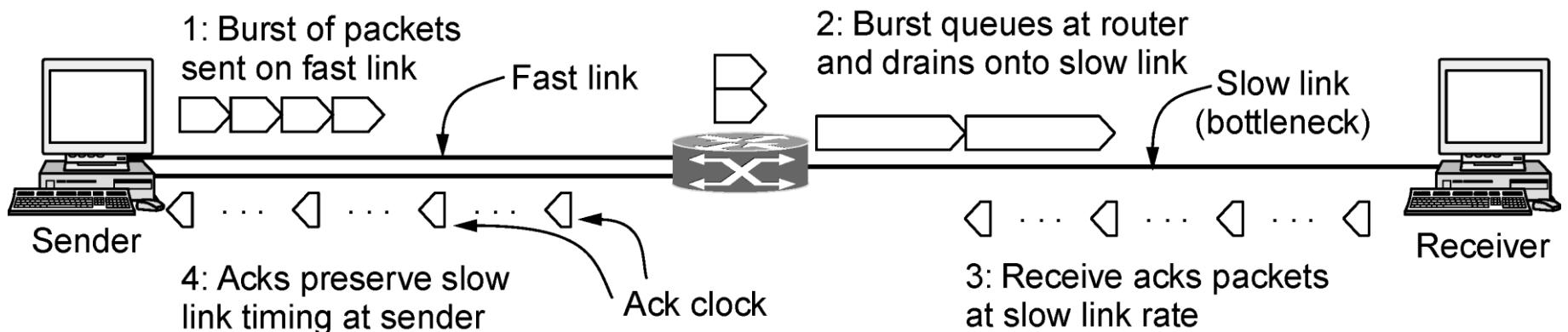
TCP uses AIMD with loss signal to control congestion

- Implemented as a congestion window (cwnd) for the number of segments that may be in the network
- Uses several mechanisms that work together

TCP Congestion Control (2)

Congestion window controls the sending rate

- Rate is cwnd / RTT; window can stop sender quickly
- ACK clock (regular receipt of ACKs) paces traffic and smoothes out sender bursts

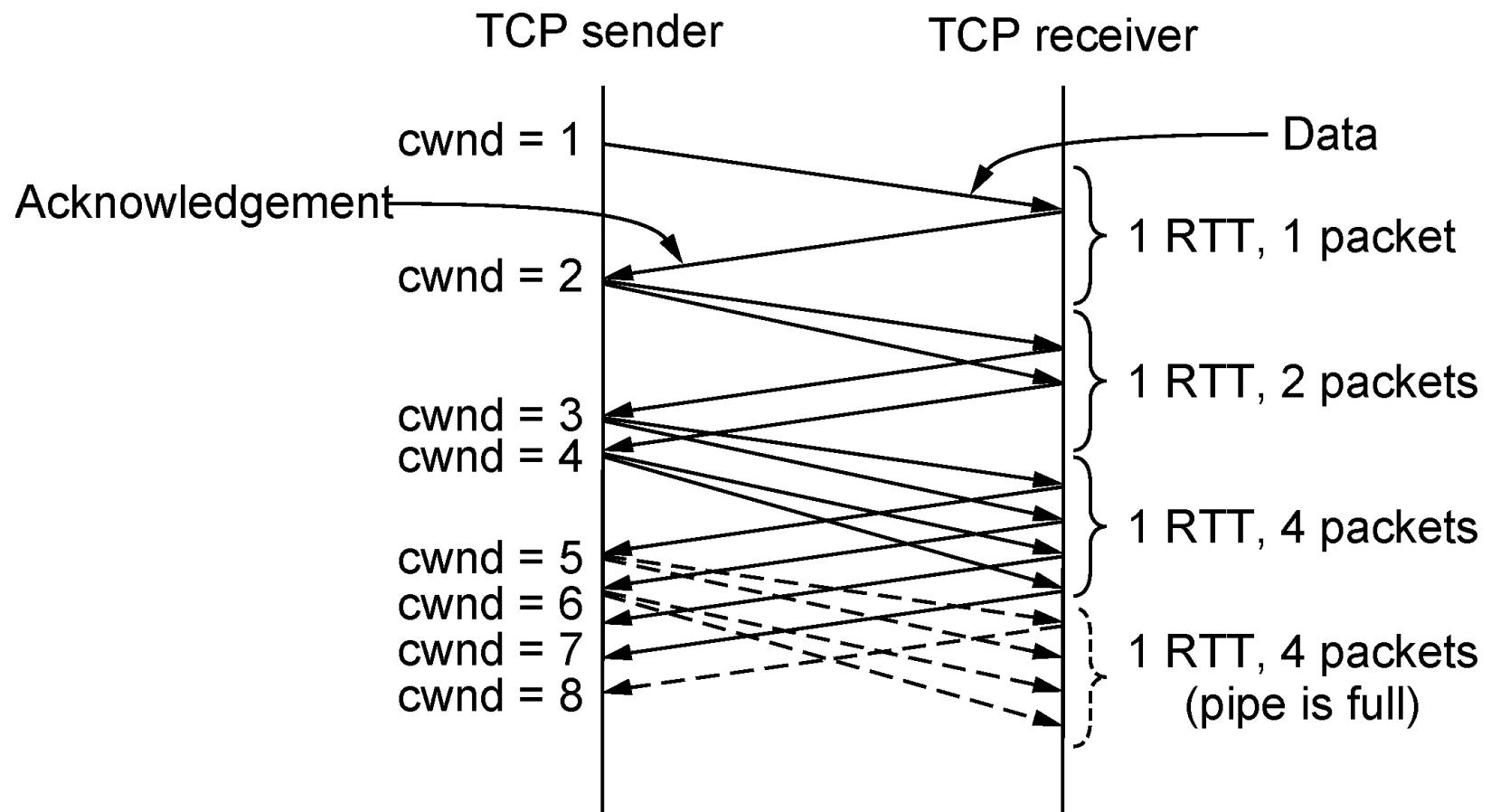


A burst of packets from a sender and the returning ack clock.

TCP Congestion Control (3)

Slow start grows congestion window exponentially

- Doubles every RTT while keeping ACK clock going



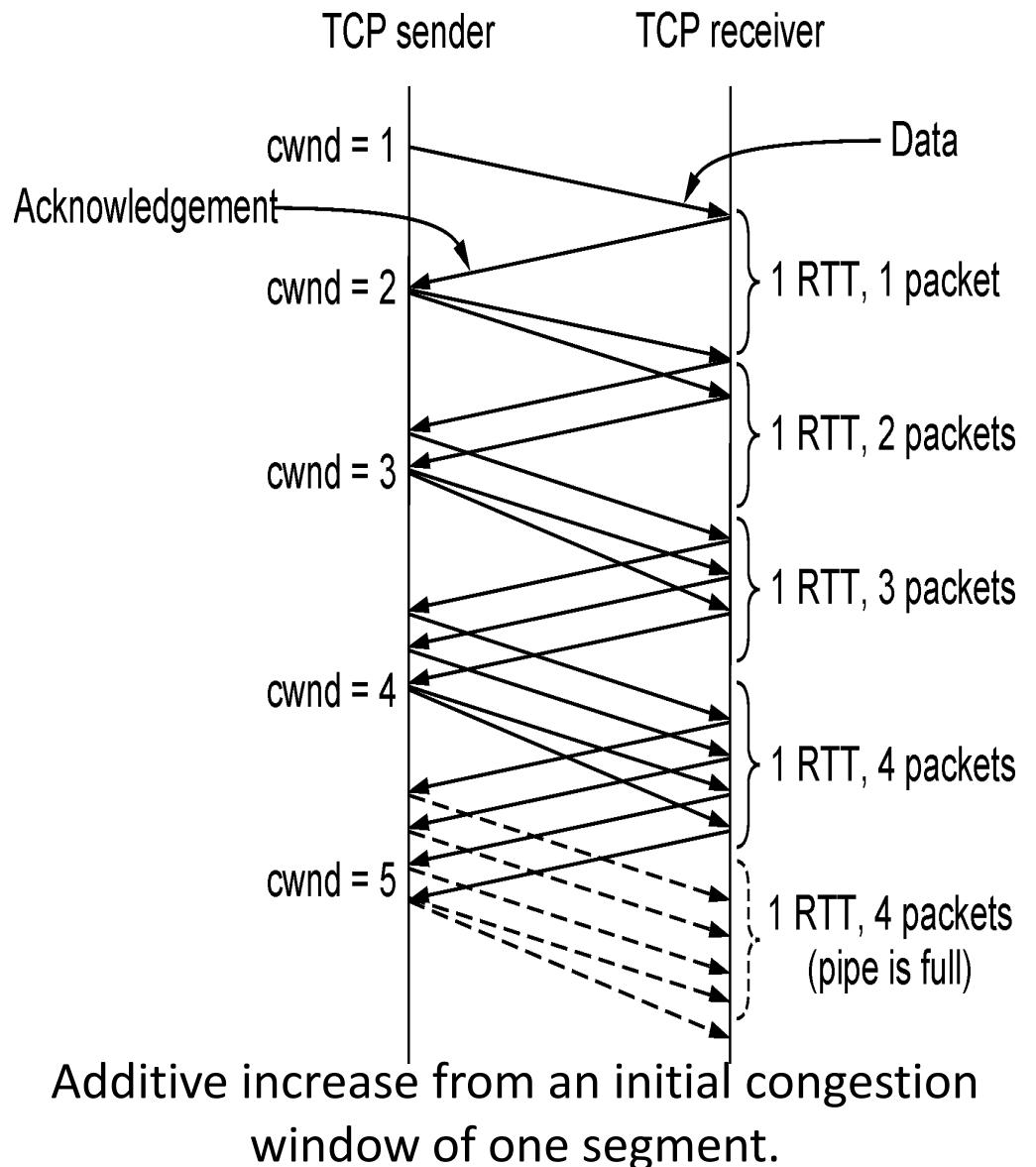
Slow start from an initial congestion window of one segment.

TCP Congestion Control (4)

Additive increase grows

cwnd slowly

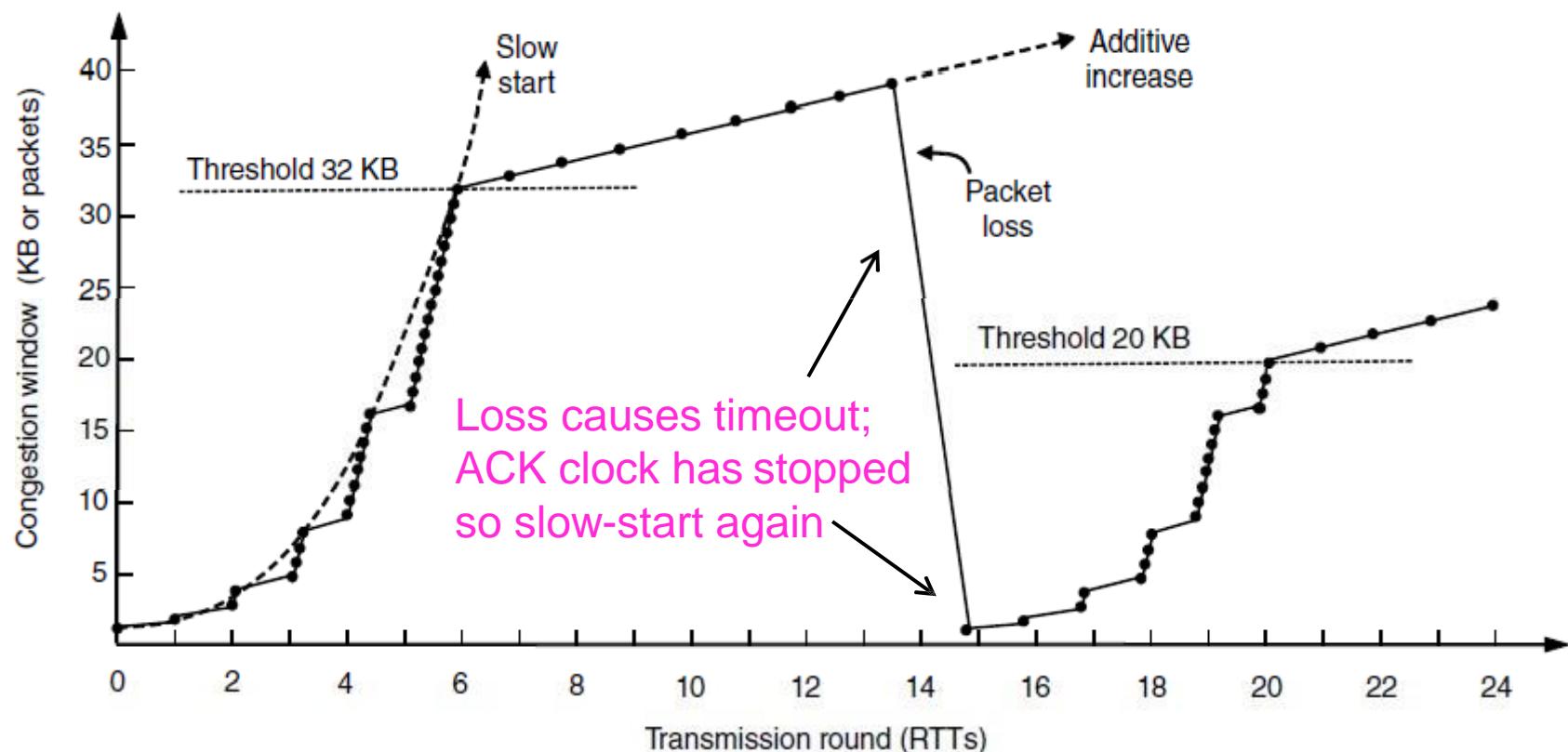
- Adds 1 every RTT
- Keeps ACK clock



TCP Congestion Control (5)

Slow start followed by additive increase (TCP Tahoe)

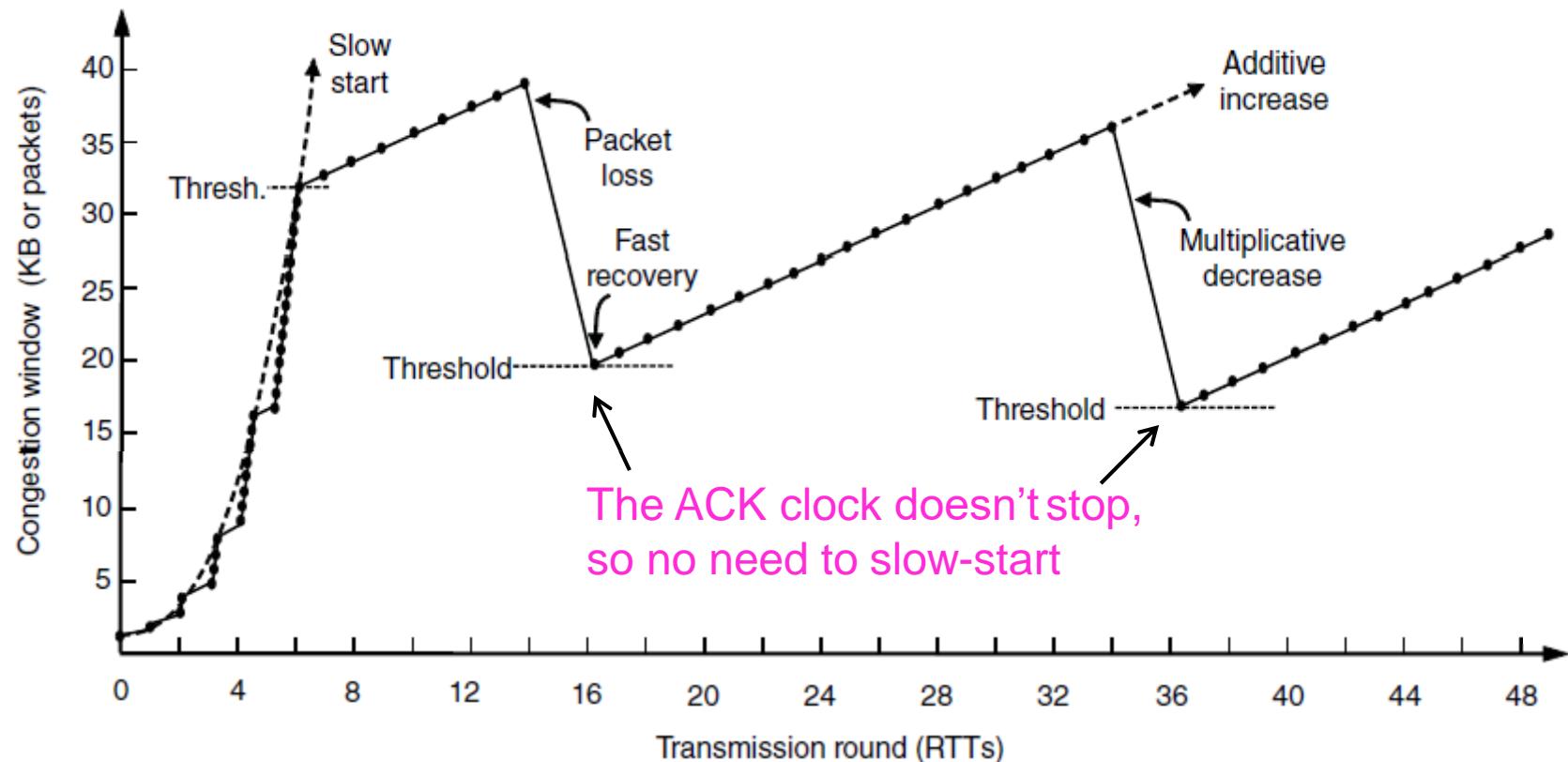
- Threshold is half of previous loss cwnd



TCP Congestion Control (6)

With fast recovery, we get the classic sawtooth (TCP Reno)

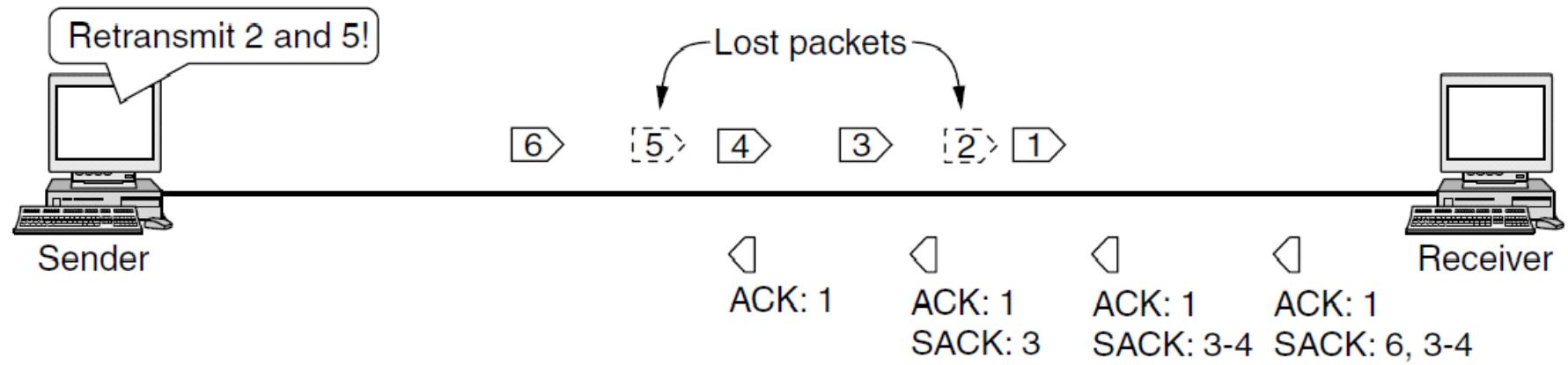
- Retransmit lost packet after 3 duplicate ACKs
- New packet for each dup. ACK until loss is repaired



TCP Congestion Control (7)

SACK (Selective ACKs) extend ACKs with a vector to describe received segments and hence losses

- Allows for more accurate retransmissions / recovery



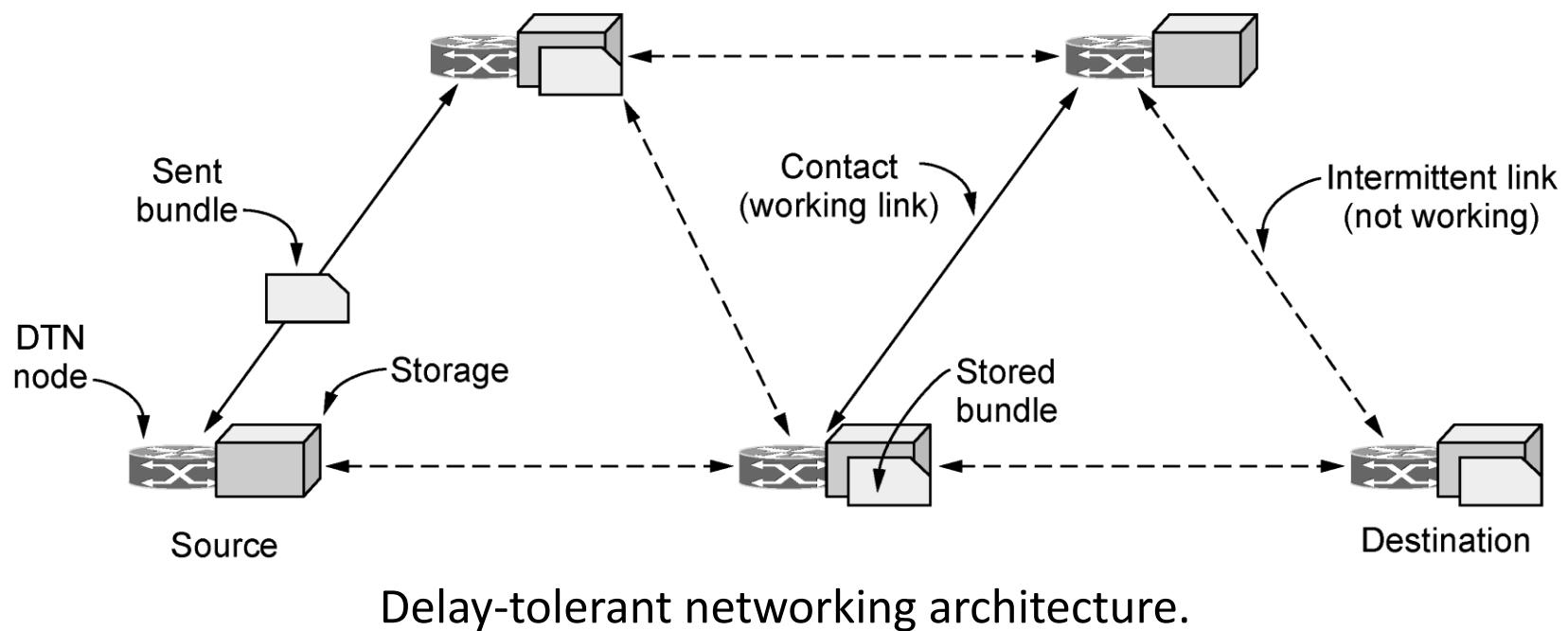
No way for us to know that 2 and 5 were lost with only ACKs

Delay Tolerant Networking

- Assumption so far:
 - Sender and receiver are continuously connected by some working path.
- Other applications
 - Space network where satellites pass in and out of range of ground stations.
 - Networks involve submarine, buses, mobile phones have a intermittent connectivity
- DTNs (Delay Tolerant Networks) store messages inside the network until they can be delivered
 - DTNs (Delay Tolerant Networks) store messages inside the network until they can be delivered
 - This model accepts delays like postal system.
 - Several applications
 - Copying of data to data centers
 - Sending traffic during off times.
 - Doing backups at night
 - It is possible to send huge datasets during off-time.

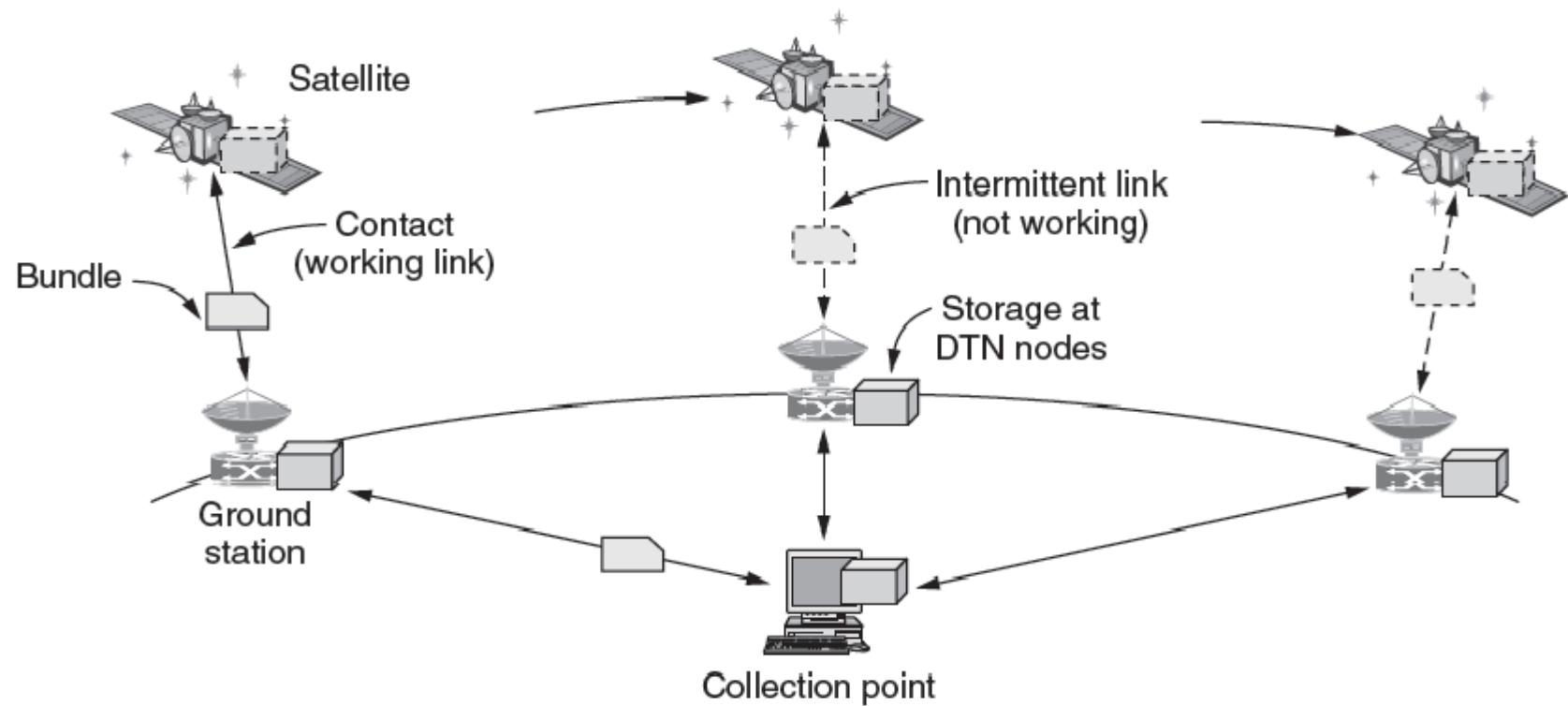
DTN Architecture (1)

- Messages called bundles are stored at DTN nodes while waiting for an intermittent link to become a contact
 - DTN nodes are equipped with storage
 - Bundles might wait hours, not milliseconds as in routers
 - May be no working end-to-end path at any time



DTN Architecture (2)

Example DTN connecting a satellite to a collection point



End

Chapter 6

Application Layer

Chapter 7

- DNS – Domain Name System
- Other services
- About Network Security
- Conclusion of the course

The Application Layer

Uses transport services to build distributed applications

| |
|-------------|
| Application |
| Transport |
| Network |
| Link |
| Physical |

DNS – Domain Name System

The DNS resolves high-level human readable names for computers to low-level IP addresses

- DNS name space »
- Domain Resource records »
- Name servers »

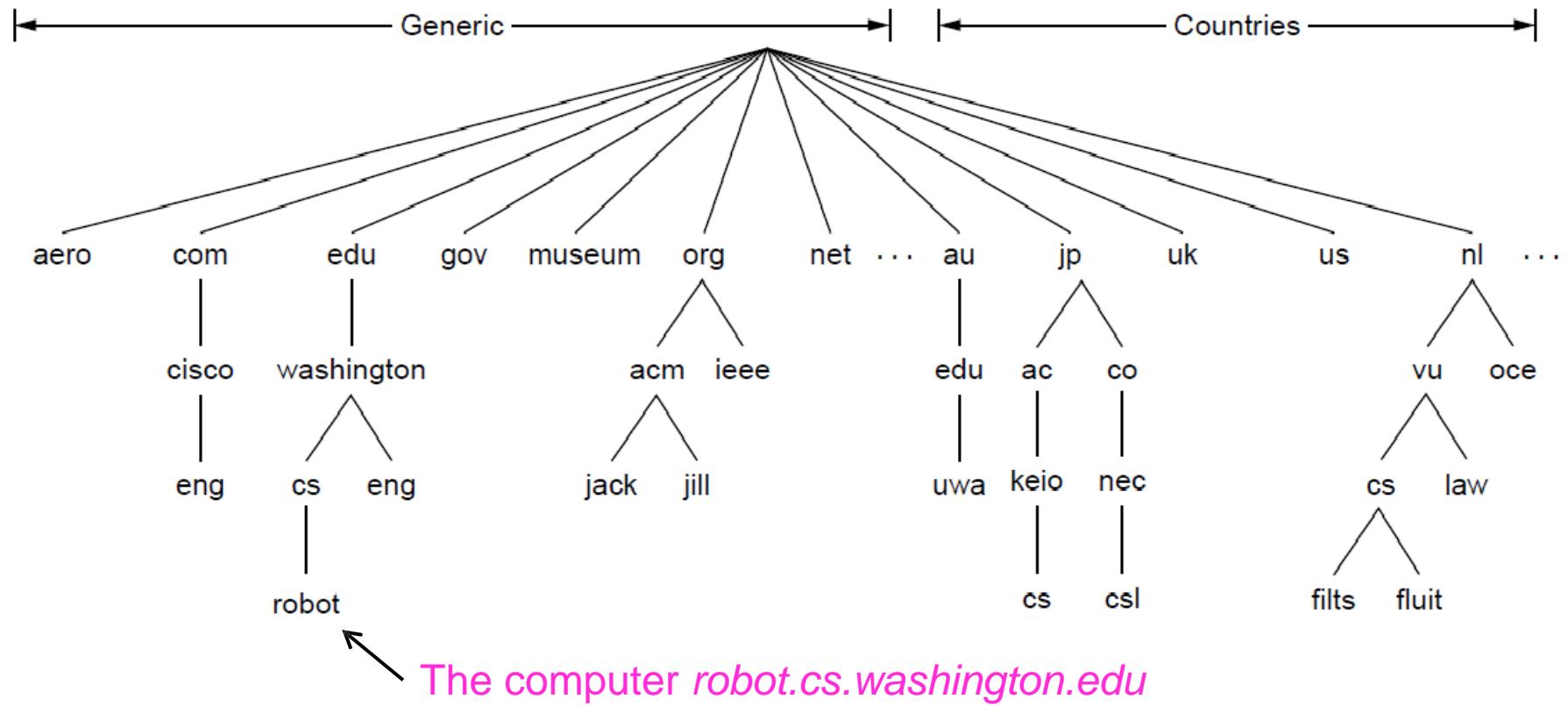
About DNS

- Moving web pages from 128.111.24.41 means that if the company moves the web server to another machine with different IP address, all programs should reflect this change.
- So, a high level, readable names were introduced in order to decouple machine names from machine addresses

The DNS Name Space (1)

DNS namespace is hierarchical from the root down

- Different parts delegated to different organizations



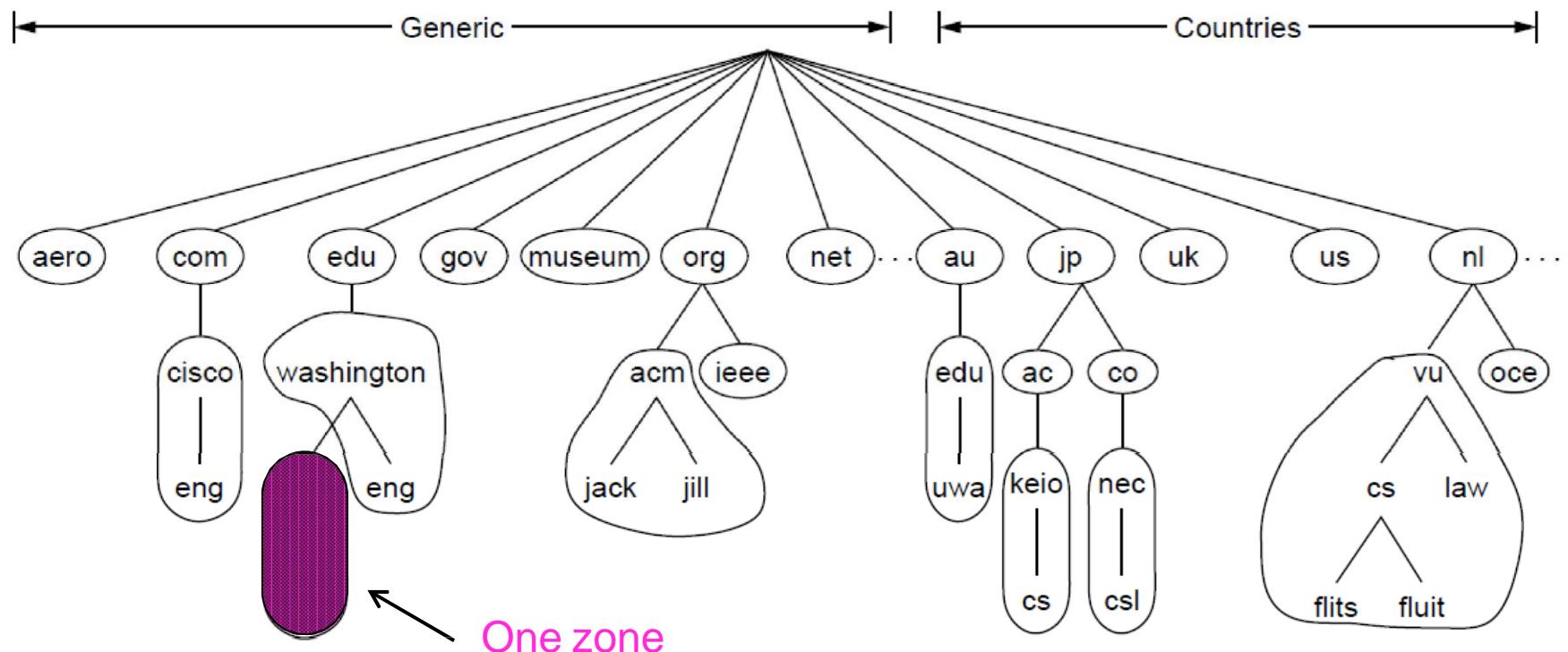
The DNS Name Space (2)

- Generic top-level domains are controlled by ICANN (Internet Corporation for Assigned Names and Numbers) who appoints registrars to run them

| Domain | Intended use | Start date | Restricted? |
|--------|-----------------------------|------------|-------------|
| com | Commercial | 1985 | No |
| edu | Educational institutions | 1985 | Yes |
| gov | Government | 1985 | Yes |
| int | International organizations | 1988 | Yes |
| mil | Military | 1985 | Yes |
| net | Network providers | 1985 | No |
| org | Non-profit organizations | 1985 | No |
| aero | Air transport | 2001 | Yes |
| biz | Businesses | 2001 | No |
| coop | Cooperatives | 2001 | Yes |
| info | Informational | 2002 | No |
| museum | Museums | 2002 | Yes |
| name | People | 2002 | No |
| pro | Professionals | 2002 | Yes |
| cat | Catalan | 2005 | Yes |
| jobs | Employment | 2005 | Yes |
| mobi | Mobile devices | 2005 | Yes |
| tel | Contact details | 2005 | Yes |
| travel | Travel industry | 2005 | Yes |
| xxx | Sex industry | 2010 | No |

Name Servers (1)

Name servers contain data for portions of the name space called zones (circled).



Name Servers (2)

Finding the IP address for a given hostname is called resolution and is done with the DNS protocol.

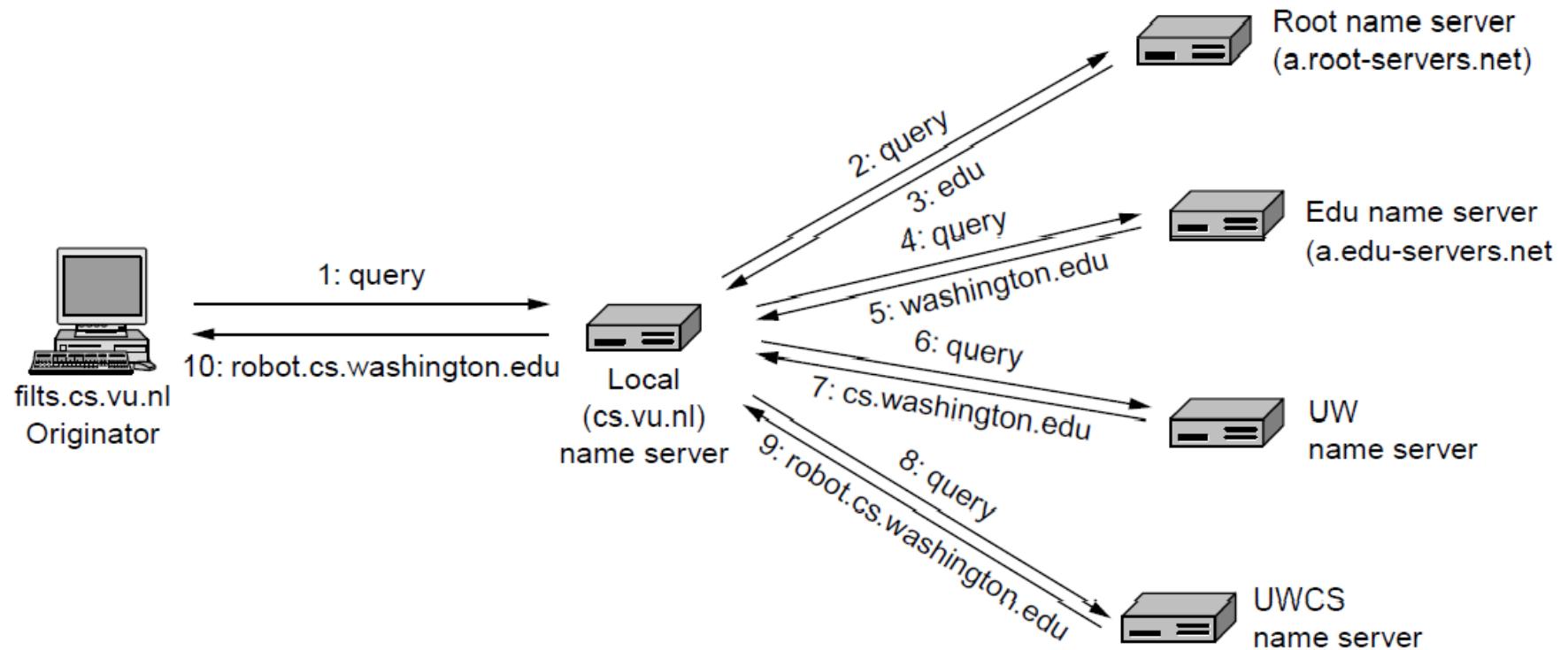
Resolution:

- Computer requests local name server to resolve
- Local name server asks the root name server
- Root returns the name server for a lower zone
- Continue down zones until name server can answer

DNS protocol:

- Runs on UDP port 53, retransmits lost messages
- Caches name server answers for better performance

Name Servers (3)



Example of a computer looking up the IP for a name

Other Applications

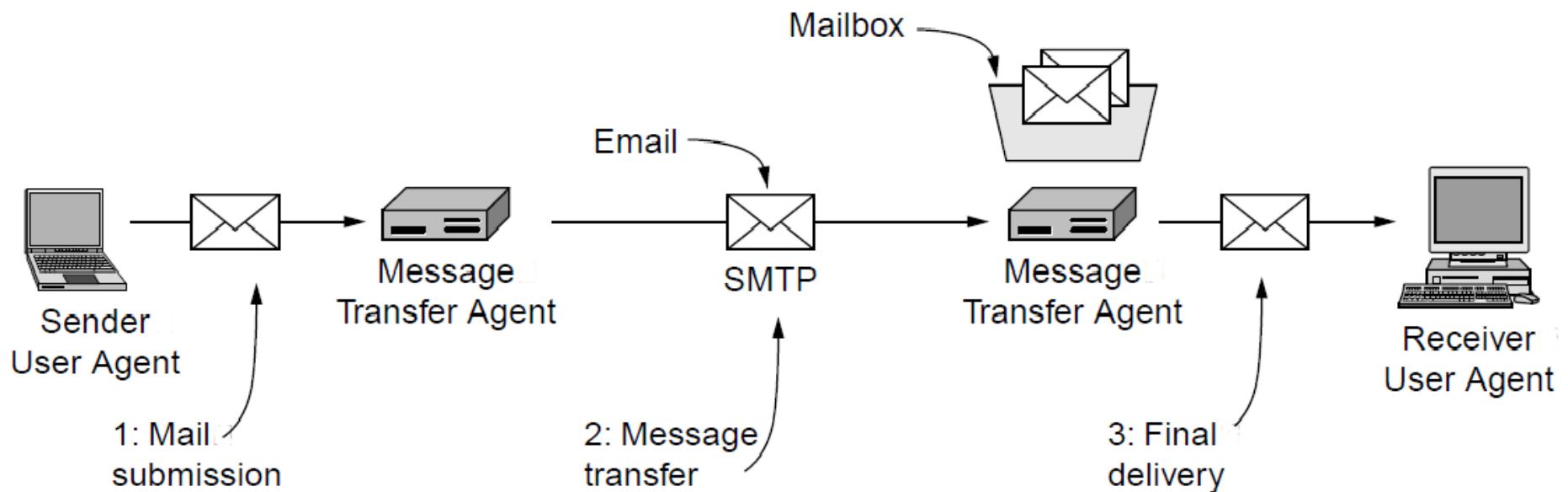
- E-mail
- World Wide Web
- HTTP
- Streaming Audio and Video
- Content Delivery

Electronic Mail

- Architecture and services »
- The user agent »
- Message formats »
- Message transfer »
- Final delivery »

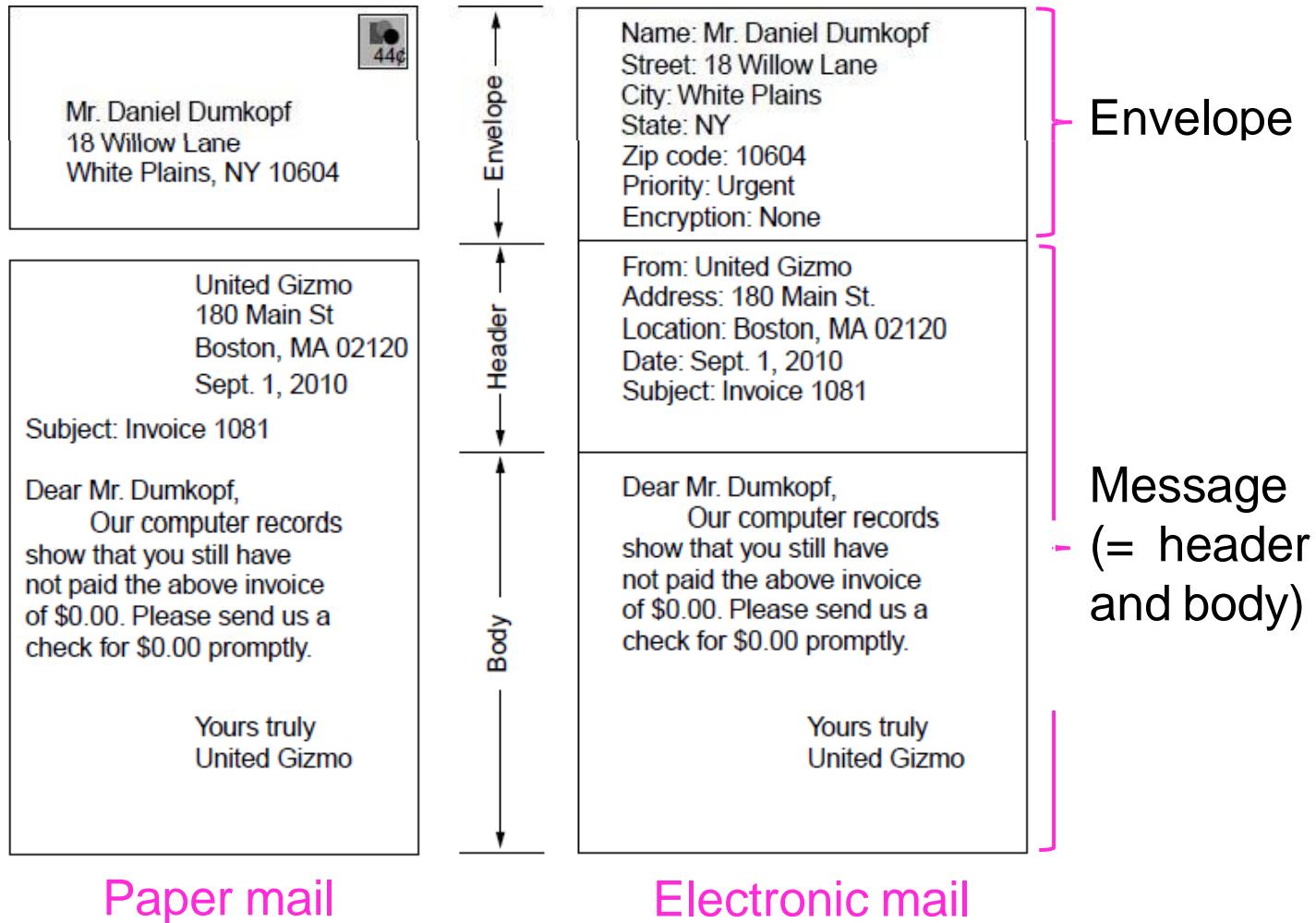
Architecture and Services (1)

The key components and steps (numbered) to send email



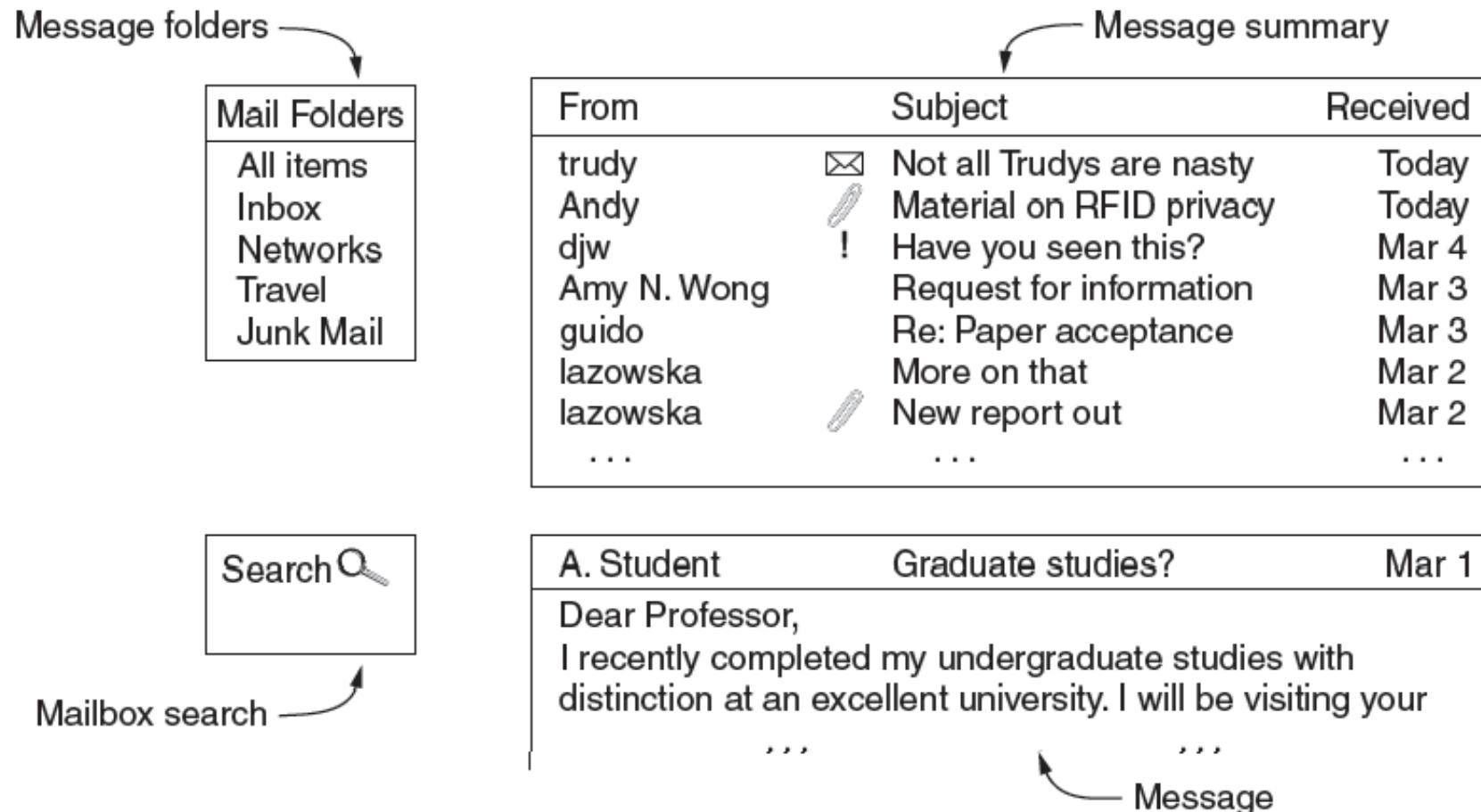
Architecture of the email system

Architecture and Services (2)



The User Agent

What users see – interface elements of a typical user agent



Message Formats (1)

Header fields related to message transport; headers are readable ASCII text

| Header | Meaning |
|--------------|---|
| To: | Email address(es) of primary recipient(s) |
| Cc: | Email address(es) of secondary recipient(s) |
| Bcc: | Email address(es) for blind carbon copies |
| From: | Person or people who created the message |
| Sender: | Email address of the actual sender |
| Received: | Line added by each transfer agent along the route |
| Return-Path: | Can be used to identify a path back to the sender |

Message Formats (2)

Other header fields useful for user agents

| Header | Meaning |
|---------------|---|
| Date: | The date and time the message was sent |
| Reply-To: | Email address to which replies should be sent |
| Message-Id: | Unique number for referencing this message later |
| In-Reply-To: | Message-Id of the message to which this is a reply |
| References: | Other relevant Message-Ids |
| Keywords: | User-chosen keywords |
| Subject: | Short summary of the message for the one-line display |

Message Formats (3)

MIME header fields used to describe what content is in the body of the message

| Header | Meaning |
|----------------------------|--|
| MIME-Version: | Identifies the MIME version |
| Content-Description: | Human-readable string telling what is in the message |
| Content-Id: | Unique identifier |
| Content-Transfer-Encoding: | How the body is wrapped for transmission |
| Content-Type: | Type and format of the content |

Message Formats (4)

Common MIME content types and subtypes

| Type | Example subtypes | Description |
|-------------|--------------------------------------|-------------------------------|
| text | plain, html, xml, css | Text in various formats |
| image | gif, jpeg, tiff | Pictures |
| audio | basic, mpeg, mp4 | Sounds |
| video | mpeg, mp4, quicktime | Movies |
| model | vrml | 3D model |
| application | octet-stream, pdf, javascript, zip | Data produced by applications |
| message | http, rfc822 | Encapsulated message |
| multipart | mixed, alternative, parallel, digest | Combination of multiple types |

Message Formats (5)

Putting it all together:
a multipart message
containing HTML and
audio alternatives.

One part
(HTML)

Another
(audio)

```
From: alice@cs.washington.edu
To: bob@ee.uwa.edu.au
MIME-Version: 1.0
Message-Id: <0704760941.AA00747@cs.washington.edu>
Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
Subject: Earth orbits sun integral number of times
```

This is the preamble. The user agent ignores it. Have a nice day.

```
--qwertyuiopasdfghjklzxcvbnm
Content-Type: text/html

<p>Happy birthday to you<br>
Happy birthday to you<br>
Happy birthday dear <b> Bob </b><br>
Happy birthday to you</p>

--qwertyuiopasdfghjklzxcvbnm
Content-Type: message/external-body;
access-type="anon-ftp";
site="bicycle.cs.washington.edu";
directory="pub";
name="birthday.snd"

content-type: audio/basic
content-transfer-encoding: base64
--qwertyuiopasdfghjklzxcvbnm--
```

Message Transfer (1)

Messages are transferred with SMTP (Simple Mail Transfer Protocol)

- Readable text commands
- Submission from user agent to MTA on port 587
- One MTA to the next MTA on port 25
- Other protocols for final delivery (IMAP, POP3)

Message Transfer (2)

Sending a message:

- From Alice to Bob
- SMTP commands
are marked pink

```
S: 220 ee.uwa.edu.au SMTP service ready
C: HELO abcd.com
S: 250 cs.washington.edu says hello to ee.uwa.edu.au
C: MAIL FROM: <alice@cs.washington.edu>
S: 250 sender ok
C: RCPT TO: <bob@ee.uwa.edu.au>
S: 250 recipient ok
C: DATA
S: 354 Send mail; end with "." on a line by itself
C: From: alice@cs.washington.edu
C: To: bob@ee.uwa.edu.au
C: MIME-Version: 1.0
C: Message-Id: <0704760941.AA00747@ee.uwa.edu.au>
C: Content-Type: multipart/alternative; boundary=qwertuuiopasdfghjklzxcvbnm
C: Subject: Earth orbits sun integral number of times
C:
C: This is the preamble. The user agent ignores it. Have a nice day.
C:
C: --qwertuuiopasdfghjklzxcvbnm
C: Content-Type: text/html
C:
C: <p>Happy birthday to you
C: Happy birthday to you
    . . . (rest of message) . . .
C: --qwertuuiopasdfghjklzxcvbnm
C: .
S: 250 message accepted
C: QUIT
S: 221 ee.uwa.edu.au closing connection
```

Message Transfer (3)

Common SMTP extensions (not in simple example)

| Keyword | Description |
|------------|---|
| AUTH | Client authentication |
| BINARYMIME | Server accepts binary messages |
| CHUNKING | Server accepts large messages in chunks |
| SIZE | Check message size before trying to send |
| STARTTLS | Switch to secure transport (TLS; see Chap. 8) |
| UTF8SMTP | Internationalized addresses |

Final Delivery (1)

User agent uses protocol like IMAP for final delivery

- Has commands to manipulate folders / messages [right]

Alternatively, a Web interface (with proprietary protocol) might be used

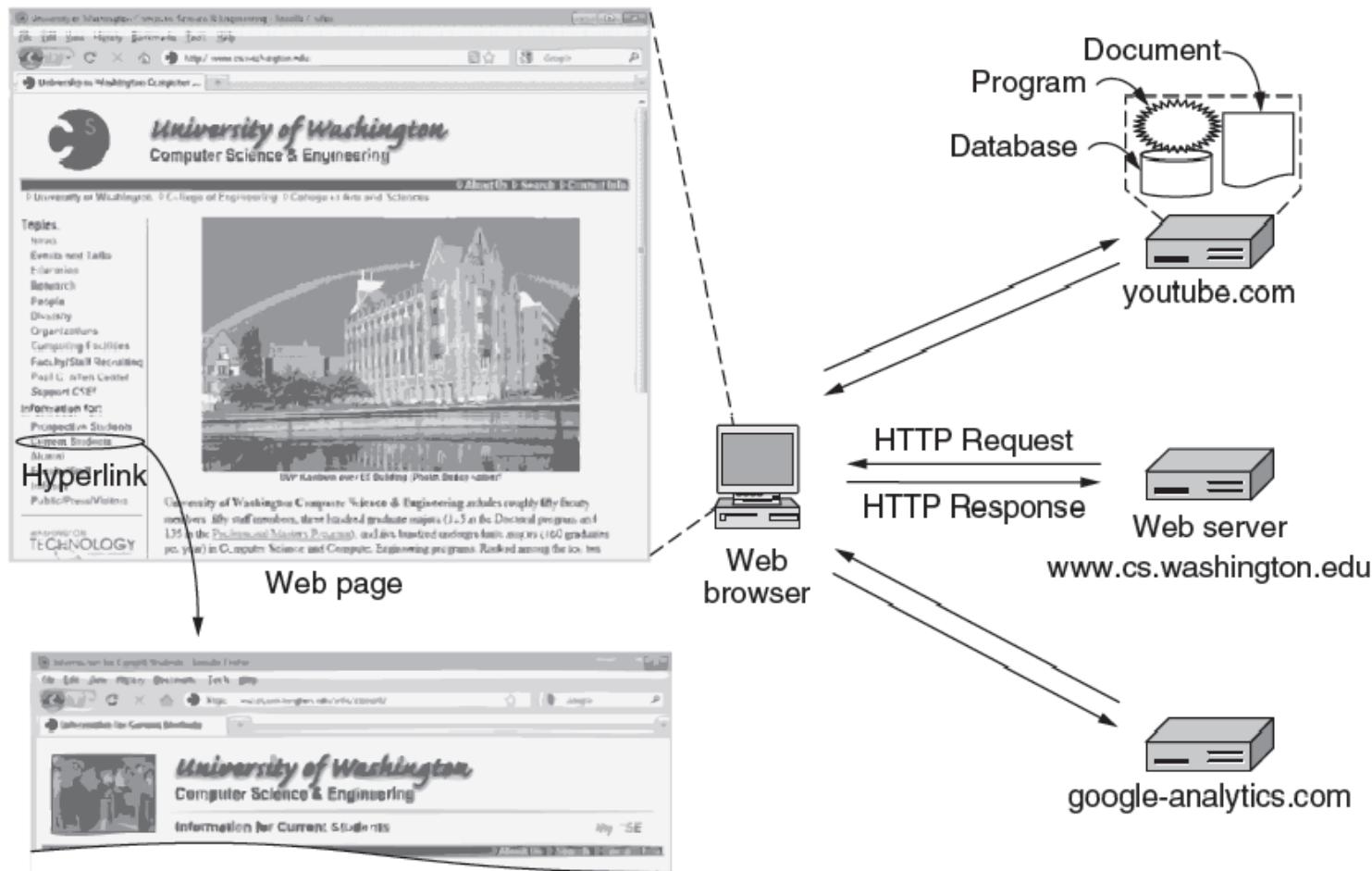
| Command | Description |
|--------------|---|
| CAPABILITY | List server capabilities |
| STARTTLS | Start secure transport (TLS; see Chap. 8) |
| LOGIN | Log on to server |
| AUTHENTICATE | Log on with other method |
| SELECT | Select a folder |
| EXAMINE | Select a read-only folder |
| CREATE | Create a folder |
| DELETE | Delete a folder |
| RENAME | Rename a folder |
| SUBSCRIBE | Add folder to active set |
| LIST | List the available folders |
| LSUB | List the active folders |
| STATUS | Get the status of a folder |
| APPEND | Add a message to a folder |
| CHECK | Get a checkpoint of a folder |
| FETCH | Get messages from a folder |
| SEARCH | Find messages in a folder |
| STORE | Alter message flags |
| COPY | Make a copy of a message in a folder |
| EXPUNGE | Remove messages flagged for deletion |
| UID | Issue commands using unique identifiers |
| NOOP | Do nothing |
| CLOSE | Remove flagged messages and close folder |
| LOGOUT | Log out and close connection |

The World Wide Web

- Architectural overview »
- Static Web pages »
- Dynamic pages and Web applications »
- HTTP – HyperText Transfer Protocol »
- The mobile Web »
- Web search »

Architectural Overview (1)

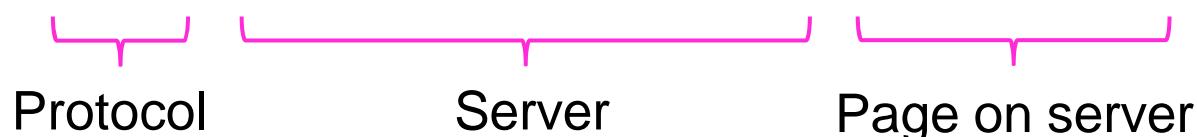
HTTP transfers pages from servers to browsers



Architectural Overview (2)

Pages are named with URLs (Uniform Resource Locators)

- Example: <http://www.phdcomics.com/comics.php>



Our focus →

| Name | Used for | Example |
|--------|-------------------------|-------------------------------------|
| http | Hypertext (HTML) | http://www.ee.uwa.edu/~rob/ |
| https | Hypertext with security | https://www.bank.com/accounts/ |
| ftp | FTP | ftp://ftp.cs.vu.nl/pub/minix/README |
| file | Local file | file:///usr/suzanne/prog.c |
| mailto | Sending email | mailto:JohnUser@acm.org |
| rtsp | Streaming media | rtsp://youtube.com/montypython.mpg |
| sip | Multimedia calls | sip:eve@adversary.com |
| about | Browser information | about:plugins |

Common URL protocols

Architectural Overview (3)

Steps a client (browser) takes to follow a hyperlink:

- Determine the protocol (HTTP)
- Ask DNS for the IP address of server
- Make a TCP connection to server
- Send request for the page; server sends it back
- Fetch other URLs as needed to display the page
- Close idle TCP connections

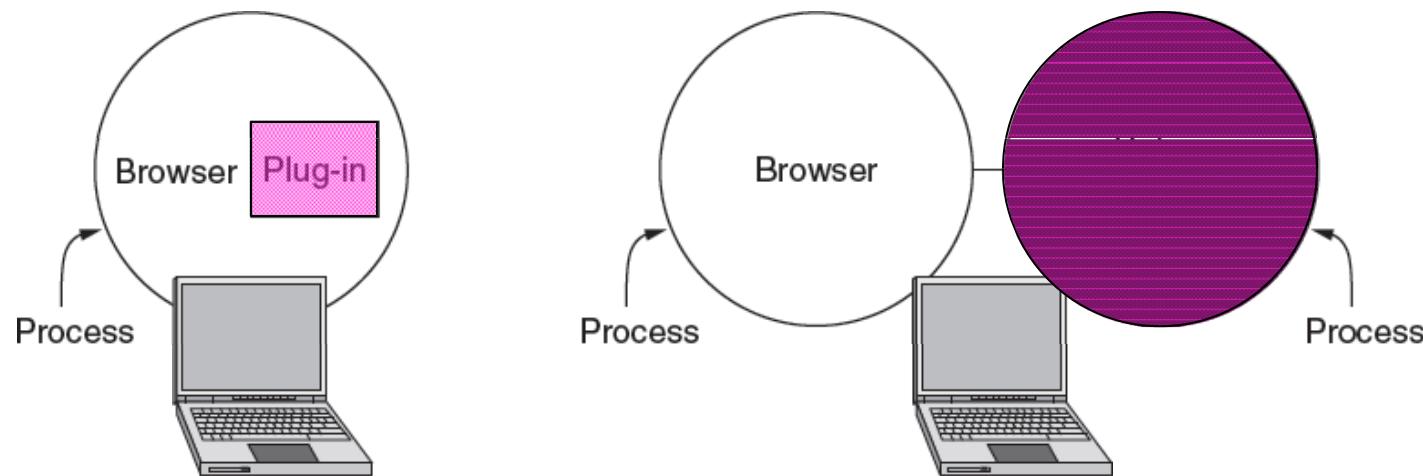
Steps a server takes to serve pages:

- Accept a TCP connection from client
- Get page request and map it to a resource (e.g., file name)
- Get the resource (e.g., file from disk)
- Send contents of the resource to the client.
- Release idle TCP connections

Architectural Overview (4)

Content type is identified by MIME types

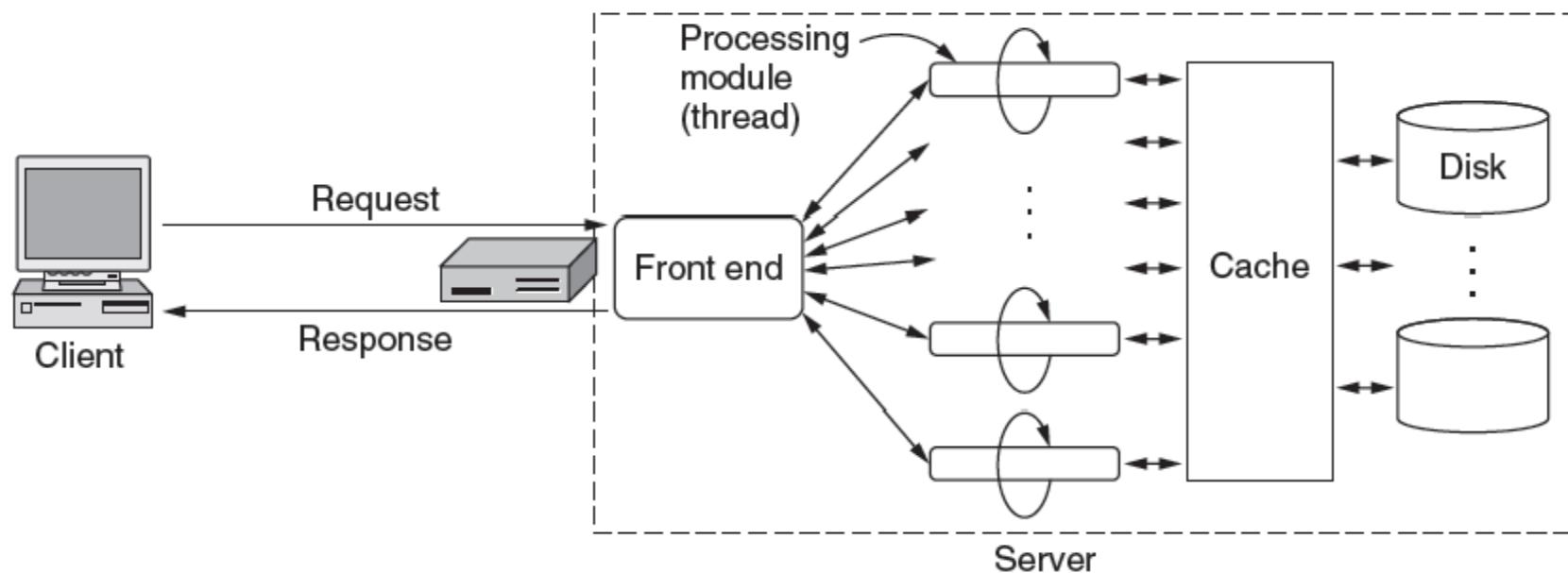
- Browser takes the appropriate action to display
- Plug-ins / helper apps extend browser for new types



Architectural Overview (5)

To scale performance, Web servers can use:

- Caching, multiple threads, and a front end



Architectural Overview (6)

Server steps, revisited:

- Resolve name of Web page requested
- Perform access control on the Web page
- Check the cache
- Fetch requested page from disk or run program
- Determine the rest of the response
- Return the response to the client
- Make an entry in the server log

Architectural Overview (7)

Cookies support stateful client/server interactions

- Server sends cookies (state) with page response
- Client stores cookies across page fetches
- Client sends cookies back to server with requests

| Domain | Path | Content | Expires | Secure |
|-----------------|------|------------------------------|----------------|--------|
| toms-casino.com | / | CustomerID=297793521 | 15-10-10 17:00 | Yes |
| jills-store.com | / | Cart=1-00501;1-07031;2-13721 | 11-1-11 14:22 | No |
| aportal.com | / | Prefs=Stk:CSCO+ORCL;Spt:Jets | 31-12-20 23:59 | No |
| sneaky.com | / | UserID=4627239101 | 31-12-19 23:59 | No |

Examples of cookies

Static Web Pages (1)

Static Web pages are simply files

- Have the same contents for each viewing

Can be visually rich and interactive nonetheless:

- HTML that mixes text and images
- Forms that gather user input
- Style sheets that tailor presentation
- Vector graphics, videos, and more (over) . . .

Static Web Pages (2)

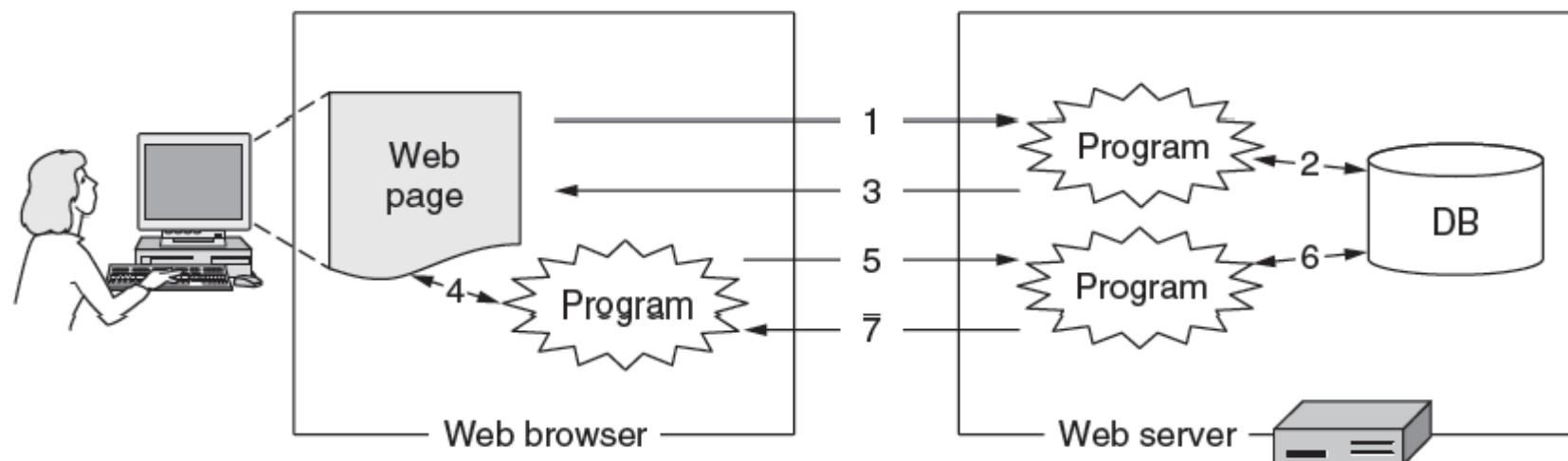
Progression of features through HTML 5.0

| Item | HTML 1.0 | HTML 2.0 | HTML 3.0 | HTML 4.0 | HTML 5.0 |
|------------------------|----------|----------|----------|----------|----------|
| Hyperlinks | x | x | x | x | x |
| Images | x | x | x | x | x |
| Lists | x | x | x | x | x |
| Active maps & images | | x | x | x | x |
| Forms | | x | x | x | x |
| Equations | | | x | x | x |
| Toolbars | | | x | x | x |
| Tables | | | x | x | x |
| Accessibility features | | | | x | x |
| Object embedding | | | | x | x |
| Style sheets | | | | x | x |
| Scripting | | | | x | x |
| Video and audio | | | | | x |
| Inline vector graphics | | | | | x |
| XML representation | | | | | x |
| Background threads | | | | | x |
| Browser storage | | | | | x |
| Drawing canvas | | | | | x |

Dynamic Pages & Web Applications (1)

Dynamic pages are generated by programs running at the server (with a database) and the client

- E.g., PHP at server, JavaScript at client
- Pages vary each time like using an application



Dynamic Pages & Web Applications (2)

Web page that gets form input and calls a server program

```
<html>
<body>
<form action="action.php" method="post">
<p> Please enter your name: <input type="text" name="name"> </p>
<p> Please enter your age: <input type="text" name="age"> </p>
<input type="submit">
</form>
</body>
</html>
```

PHP server program that creates a custom Web page

```
<html>
<body>
<h1> Reply: </h1>
Hello <?php echo $name; ?>.
Prediction: next year you will be <?php echo $age + 1; ?>
</body>
</html>
```

PHP calls

Resulting Web page (for inputs “Barbara” and “32”)

```
<html>
<body>
<h1> Reply: </h1>
Hello Barbara.
Prediction: next year you will be 33
</body>
</html>
```

Dynamic Pages & Web Applications (3)

JavaScript program produces result page in the browser

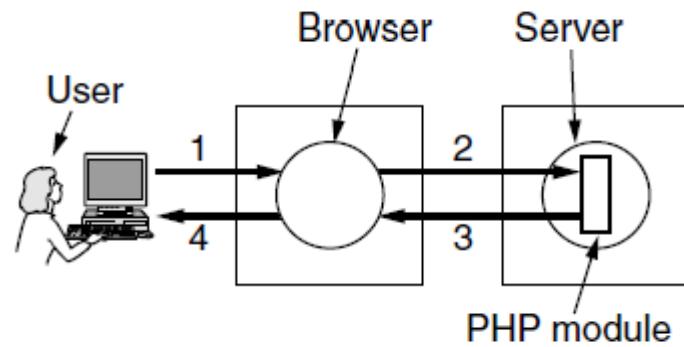
```
<html>
<head>
<script language="javascript" type="text/javascript">
function response(test_form) {
    var person = test_form.name.value;
    var years = eval(test_form.age.value) + 1;
    document.open();
    document.writeln("<html> <body>");
    document.writeln("Hello " + person + ".<br>");
    document.writeln("Prediction: next year you will be " + years + ".");
    document.writeln("</body> </html>");
    document.close();
}
</script>
</head>

<body>
<form>
Please enter your name: <input type="text" name="name">
<p>
Please enter your age: <input type="text" name="age">
<p>
<input type="button" value="submit" onclick="response(this.form)">
</form>
</body>
</html>
```

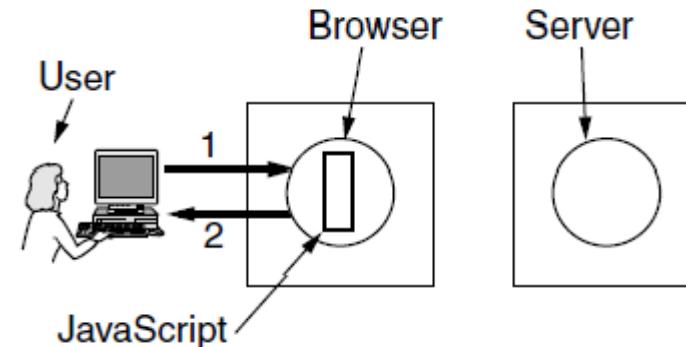
First page with form, gets input and calls program above

Dynamic Pages & Web Applications (4)

The difference between server and client programs



Server-side scripting with PHP



Client-side scripting with JavaScript

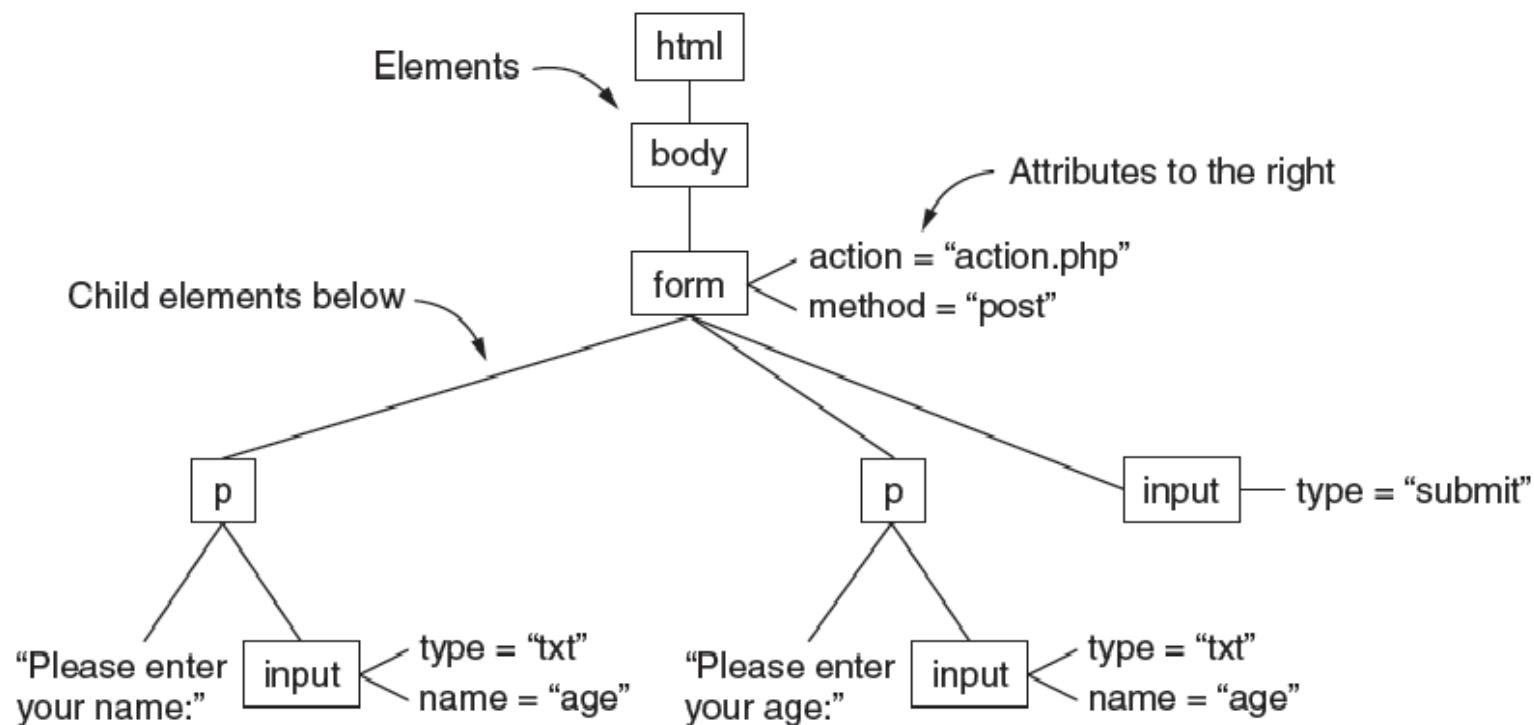
Dynamic Pages & Web Applications (5)

Web applications use a set of technologies that work together, e.g. AJAX:

- HTML: present information as pages.
- DOM: change parts of pages while they are viewed.
- XML: let programs exchange data with the server.
- Asynchronous way to send and retrieve XML data.
- JavaScript as a language to bind all this together.

Dynamic Pages & Web Applications (6)

The DOM (Document Object Model) tree represents Web pages as a structure that programs can alter



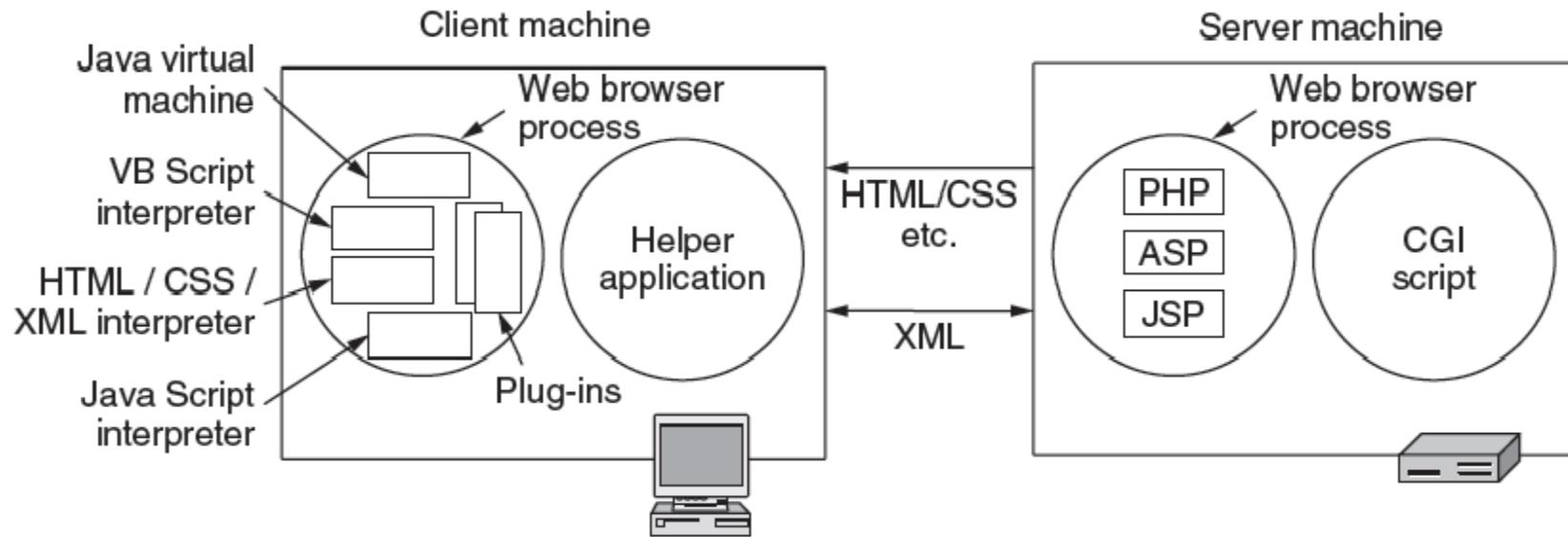
Dynamic Pages & Web Applications (7)

XML captures document structure, not presentation like HTML. Ex:

```
<?xml version="1.0" ?>
<book_list>
  <book>
    <title> Human Behavior and the Principle of Least Effort </title>
    <author> George Zipf </author>
    <year> 1949 </year>
  </book>
  <book>
    <title> The Mathematical Theory of Communication </title>
    <author> Claude E. Shannon </author>
    <author> Warren Weaver </author>
    <year> 1949 </year>
  </book>
  <book>
    <title> Nineteen Eighty-Four </title>
    <author> George Orwell </author>
    <year> 1949 </year>
  </book>
</book_list>
```

Dynamic Pages & Web Applications (8)

Web applications use a set of technologies, revisited:



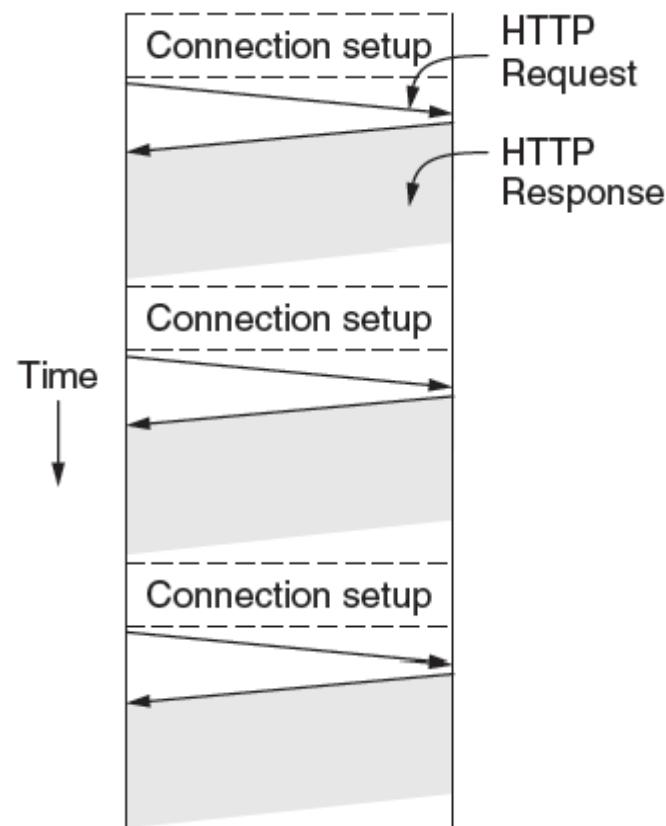
HTTP (1)

HTTP (HyperText Transfer Protocol) is a request-response protocol that runs on top of TCP

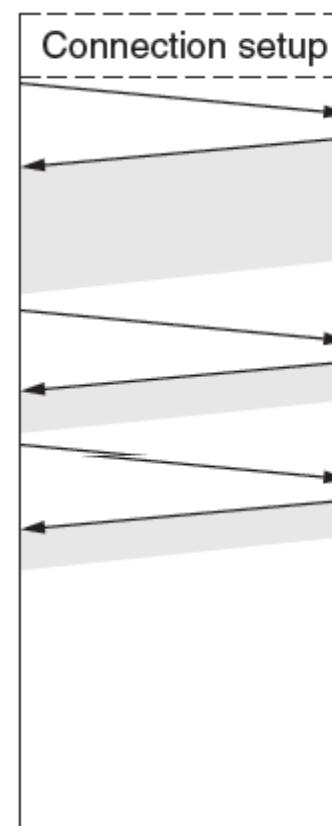
- Fetches pages from server to client
- Server usually runs on port 80
- Headers are given in readable ASCII
- Content is described with MIME types
- Protocol has support for pipelining requests
- Protocol has support for caching

HTTP (2)

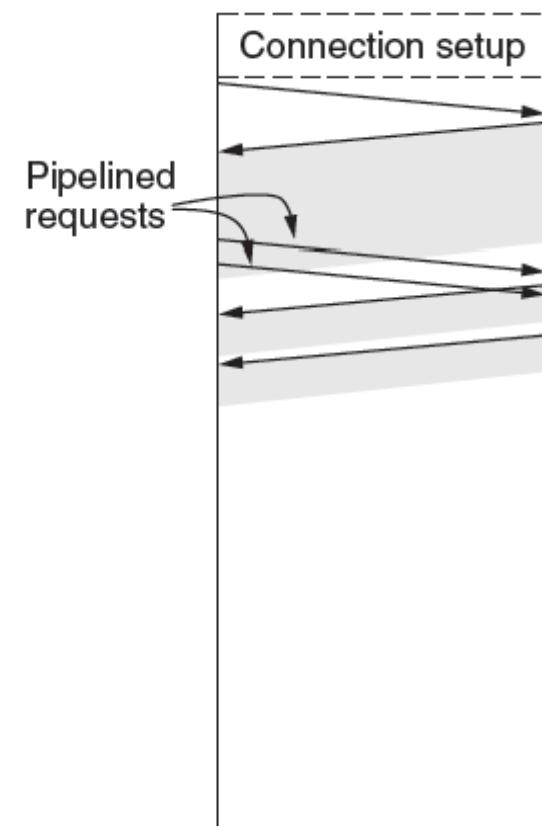
HTTP uses persistent connections to improve performance



One connection for
each request



Sequential requests
on one connection



Pipelined requests
on one connection

HTTP (3)

HTTP has several request methods.

| Method | Description |
|---------|---------------------------|
| GET | Read a Web page |
| HEAD | Read a Web page's header |
| POST | Append to a Web page |
| PUT | Store a Web page |
| DELETE | Remove the Web page |
| TRACE | Echo the incoming request |
| CONNECT | Connect through a proxy |
| OPTIONS | Query options for a page |

Fetch a page → GET
Used to send input data to a server program → POST

HTTP (4)

Response codes tell the client how the request fared:

| Code | Meaning | Examples |
|-------------|----------------|--|
| 1xx | Information | 100 = server agrees to handle client's request |
| 2xx | Success | 200 = request succeeded; 204 = no content present |
| 3xx | Redirection | 301 = page moved; 304 = cached page still valid |
| 4xx | Client error | 403 = forbidden page; 404 = page not found |
| 5xx | Server error | 500 = internal server error; 503 = try again later |

HTTP (5)

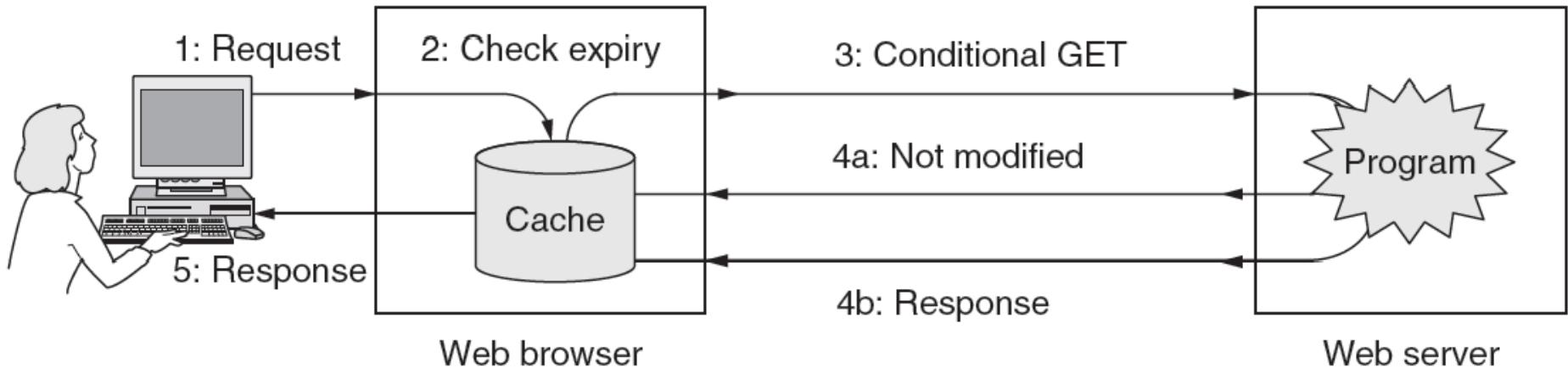
Many headers carry key information:

| Function | Example Headers |
|---|---|
| Browser capabilities (client → server) | User-Agent, Accept, Accept-Charset, Accept-Encoding, Accept-Language |
| Caching related (mixed directions) | If-Modified-Since, If-None-Match, Date, Last-Modified, Expires, Cache-Control, ETag |
| Browser context (client → server) | Cookie, Referer, Authorization, Host |
| Content delivery (server → client) | Content-Encoding, Content-Length, Content-Type, Content-Language, Content-Range, Set-Cookie |

HTTP (6)

HTTP caching checks to see if the browser has a known fresh copy, and if not if the server has updated the page

- Uses a collection of headers for the checks
- Can include further levels of caching (e.g., proxy)



The Mobile Web

Mobiles (phones, tablets) are challenging as clients:

- Relatively small screens
- Limited input capabilities, lengthy input.
- Network bandwidth is limited
- Connectivity may be intermittent.
- Computing power is limited

Strategies to handle them:

- Content: servers provide mobile-friendly versions; transcoding can also be used
- Protocols: no real need for specialized protocols; HTTP with header compression sufficient

Web Search

Search has proved hugely popular, in tandem with advertising that has proved hugely profitable

- A simple interface for users to navigate the Web

Search engine requires:

- Content from all sites, accessed by crawling. Follow links to new pages, but beware programs.
- Indexing, which benefits from known and discovered structure (such as XML) to increase relevance

Streaming Audio and Video

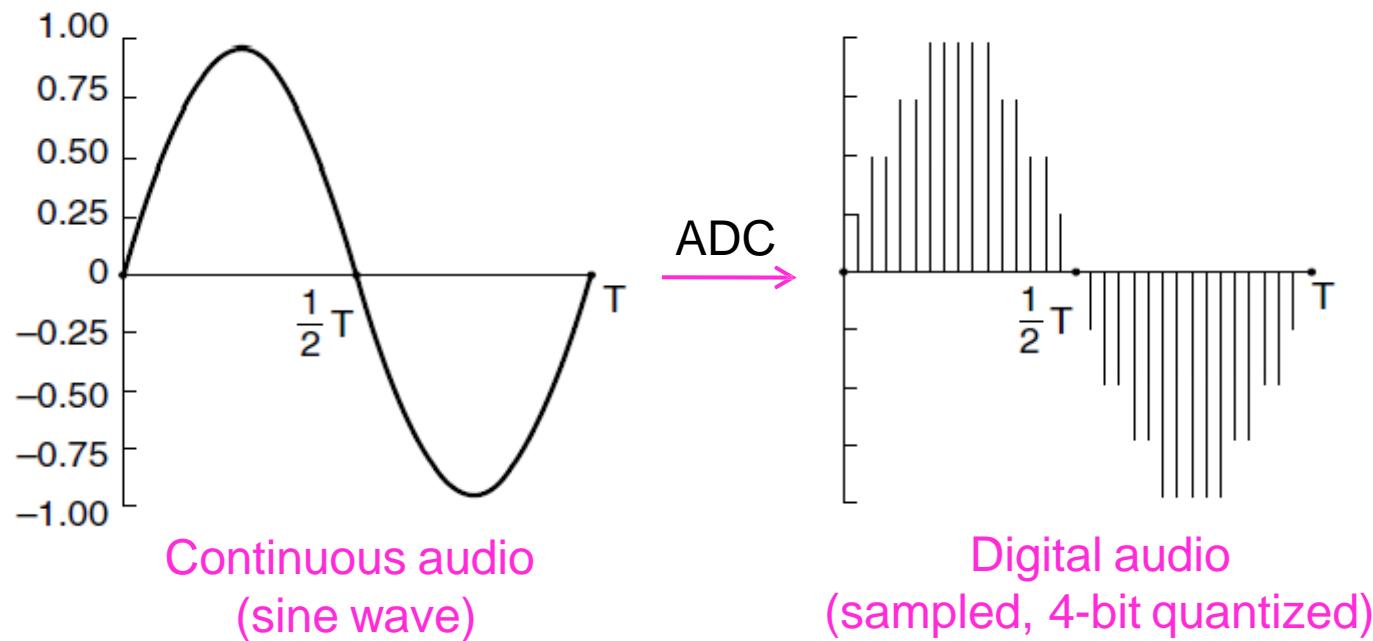
Audio and video have become key types of traffic,
e.g., voice over IP, and video streaming.

- Digital audio »
- Digital video »
- Streaming stored media »
- Streaming live media »
- Real-time conferencing »

Digital Audio (1)

ADC (Analog-to-Digital Converter) produces digital audio from a microphone

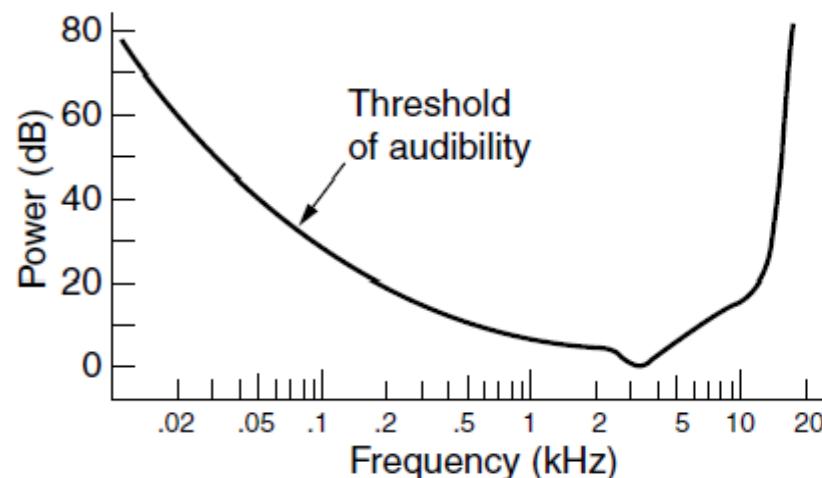
- Telephone: 8000 8-bit samples/second (64 Kbps); computer audio is usually better quality (e.g., 16 bit)



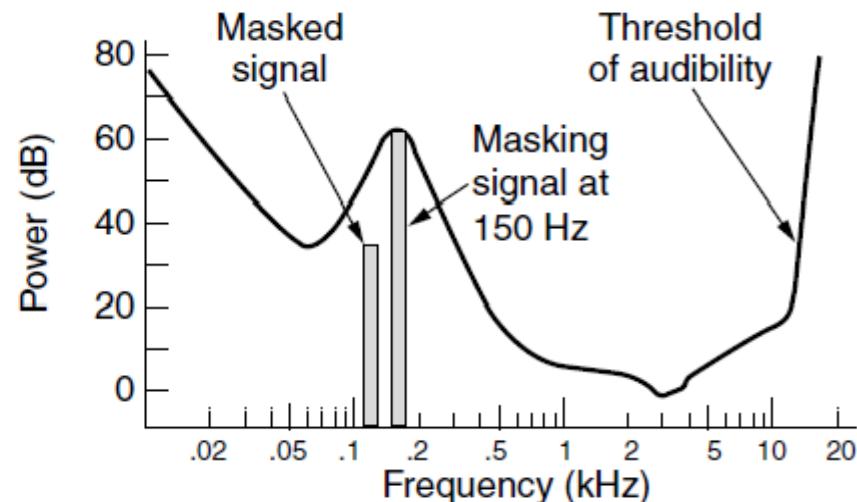
Digital Audio (2)

Digital audio is typically compressed before it is sent

- Lossy encoders (like AAC) exploit human perception
- Large compression ratios (can be >10X)



Sensitivity of the ear
varies with frequency



A loud tone can mask
nearby tones

Digital Video (1)

Video is digitized as pixels (sampled, quantized)

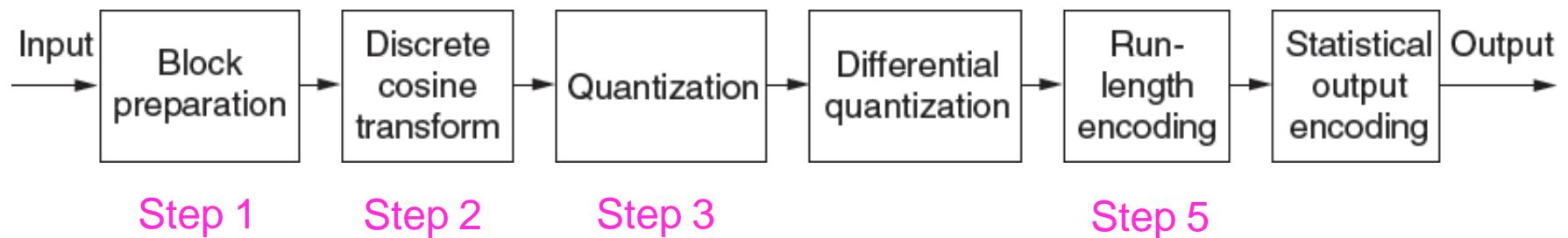
- TV quality: 640x480 pixels, 24-bit color, 30 times/sec

Video is sent compressed due to its large bandwidth

- Lossy compression exploits human perception
 - E.g., JPEG for still images, MPEG, H.264 for video
- Large compression ratios (often 50X for video)
- Video is normally > 1 Mbps, versus >10 kbps for speech and >100 kbps for music

Digital Video (2)

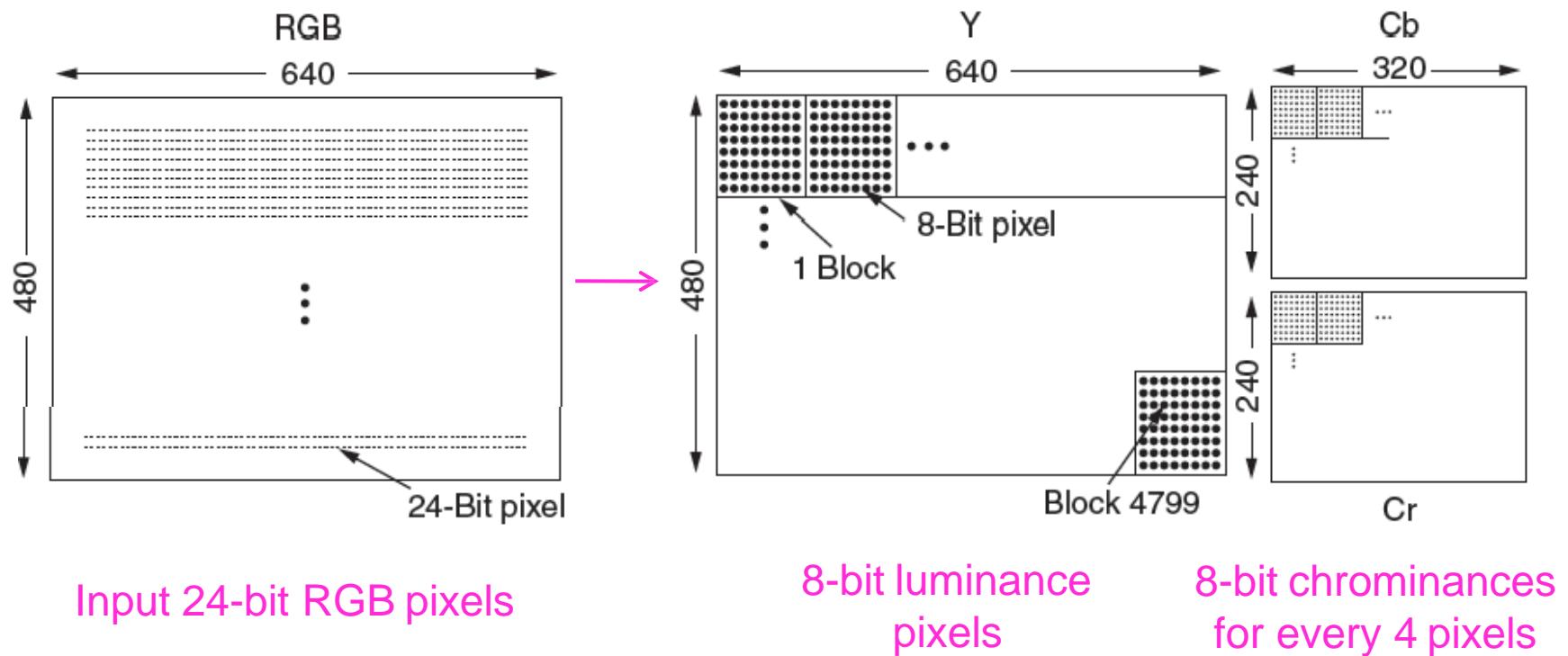
JPEG lossy compression sequence for one image:



Digital Video (3)

Step 1: Pixels are mapped to luminance/chrominance (YCbCr) color space and chrominance is sub-sampled

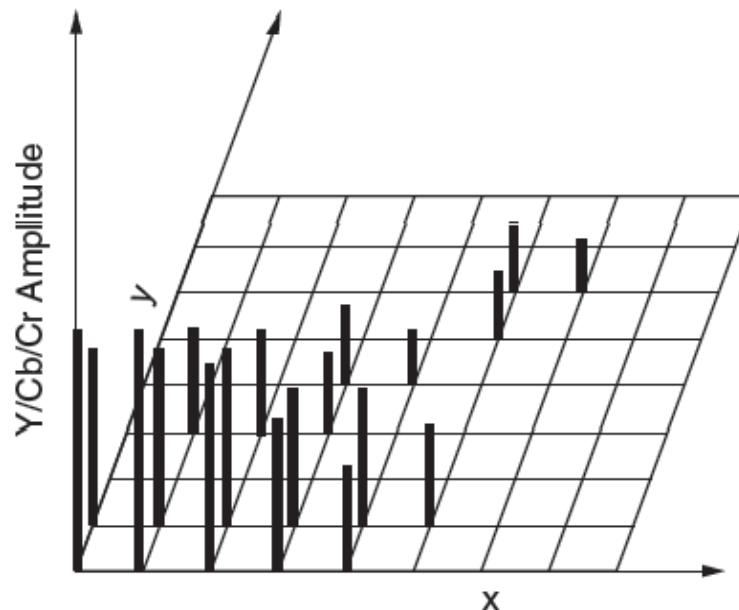
- The eye is less sensitive to chrominance



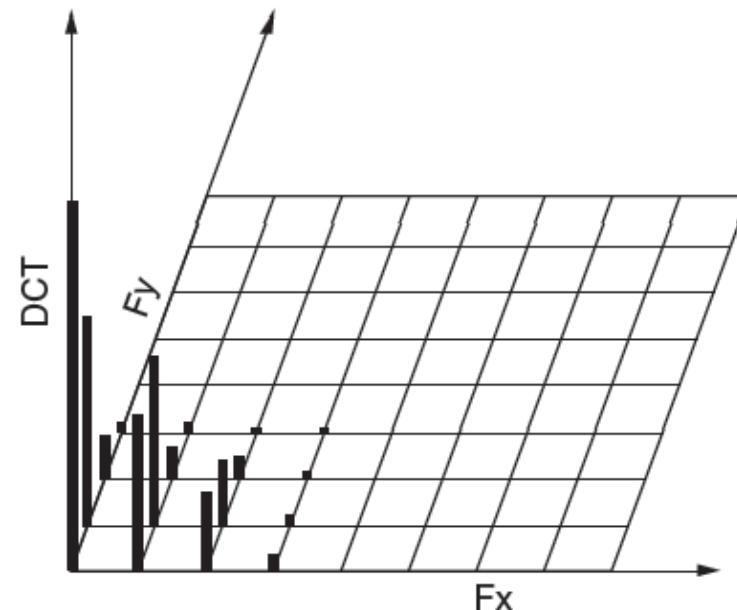
Digital Video (4)

Step 2: Each component block is transformed to spatial frequencies with DCT (Discrete Cosine Transformation)

- Captures the key image features



One component block



Transformed block

Digital Video (5)

Step 3: DCT coefficients are quantized by dividing by thresholds; reduces bits in higher spatial frequencies

- Top left element is differenced over blocks (Step 4)

DCT coefficients

| | | | | | | | |
|-----|----|----|----|---|---|---|---|
| 150 | 80 | 40 | 14 | 4 | 2 | 1 | 0 |
| 92 | 75 | 36 | 10 | 6 | 1 | 0 | 0 |
| 52 | 38 | 26 | 8 | 7 | 4 | 0 | 0 |
| 12 | 8 | 6 | 4 | 2 | 1 | 0 | 0 |
| 4 | 3 | 2 | 0 | 0 | 0 | 0 | 0 |
| 2 | 2 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Quantization table

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 1 | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| 1 | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| 2 | 2 | 2 | 4 | 8 | 16 | 32 | 64 |
| 4 | 4 | 4 | 4 | 8 | 16 | 32 | 64 |
| 8 | 8 | 8 | 8 | 8 | 16 | 32 | 64 |
| 16 | 16 | 16 | 16 | 16 | 16 | 32 | 64 |
| 32 | 32 | 32 | 32 | 32 | 32 | 32 | 64 |
| 64 | 64 | 64 | 64 | 64 | 64 | 64 | 64 |

Quantized coefficients

| | | | | | | | |
|-----|----|----|---|---|---|---|---|
| 150 | 80 | 20 | 4 | 1 | 0 | 0 | 0 |
| 92 | 75 | 18 | 3 | 1 | 0 | 0 | 0 |
| 26 | 19 | 13 | 2 | 1 | 0 | 0 | 0 |
| 3 | 2 | 2 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Input

/

Thresholds

=

Output

Digital Video (6)

Step 5: The block is run-length encoded in a zig-zag order. Then it is Huffman coded before sending (Step 6)

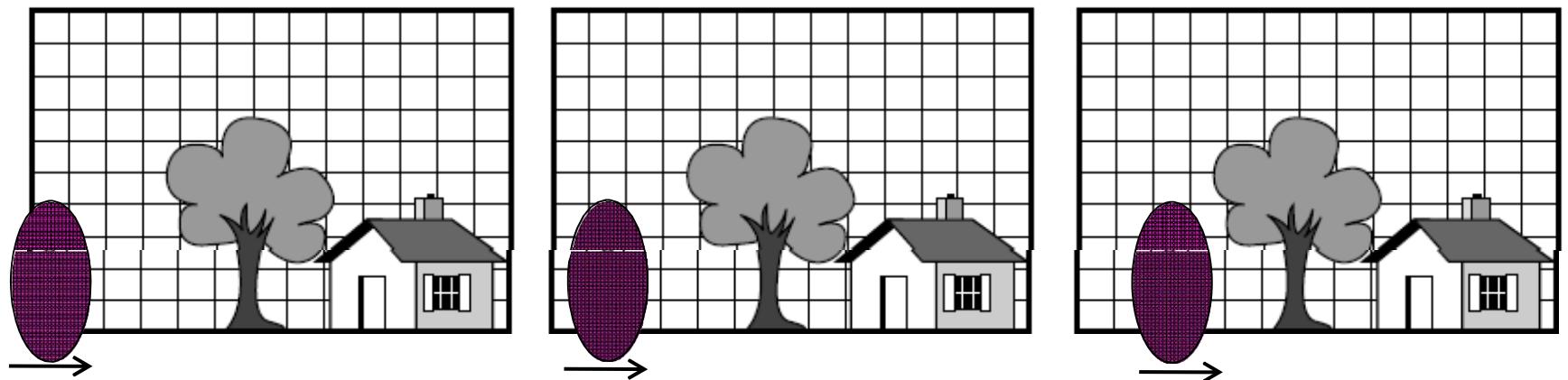
| | | | | | | | |
|-----|----|----|---|---|---|---|---|
| 150 | 80 | 20 | 4 | 1 | 0 | 0 | 0 |
| 92 | 75 | 18 | 3 | 1 | 0 | 0 | 0 |
| 26 | 19 | 13 | 2 | 1 | 0 | 0 | 0 |
| 3 | 2 | 2 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Order in which the block coefficients are sent

Digital Video (7)

MPEG compresses over a sequence of frames, further using motion tracking to remove temporal redundancy

- I (Intra-coded) frames are self-contained
- P (Predictive) frames use block motion predictions
- B (Bidirectional) frames may base prediction on future frame

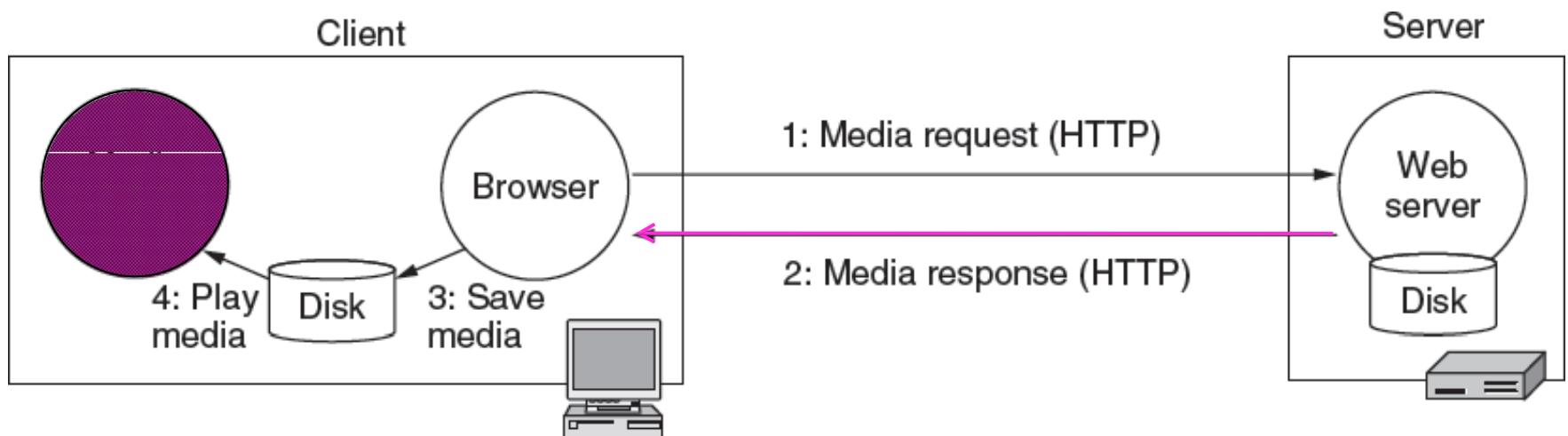


Three consecutive frames with stationary and moving components

Streaming Stored Media (1)

A simple method to stream stored media, e.g., for video on demand, is to fetch the video as a file download

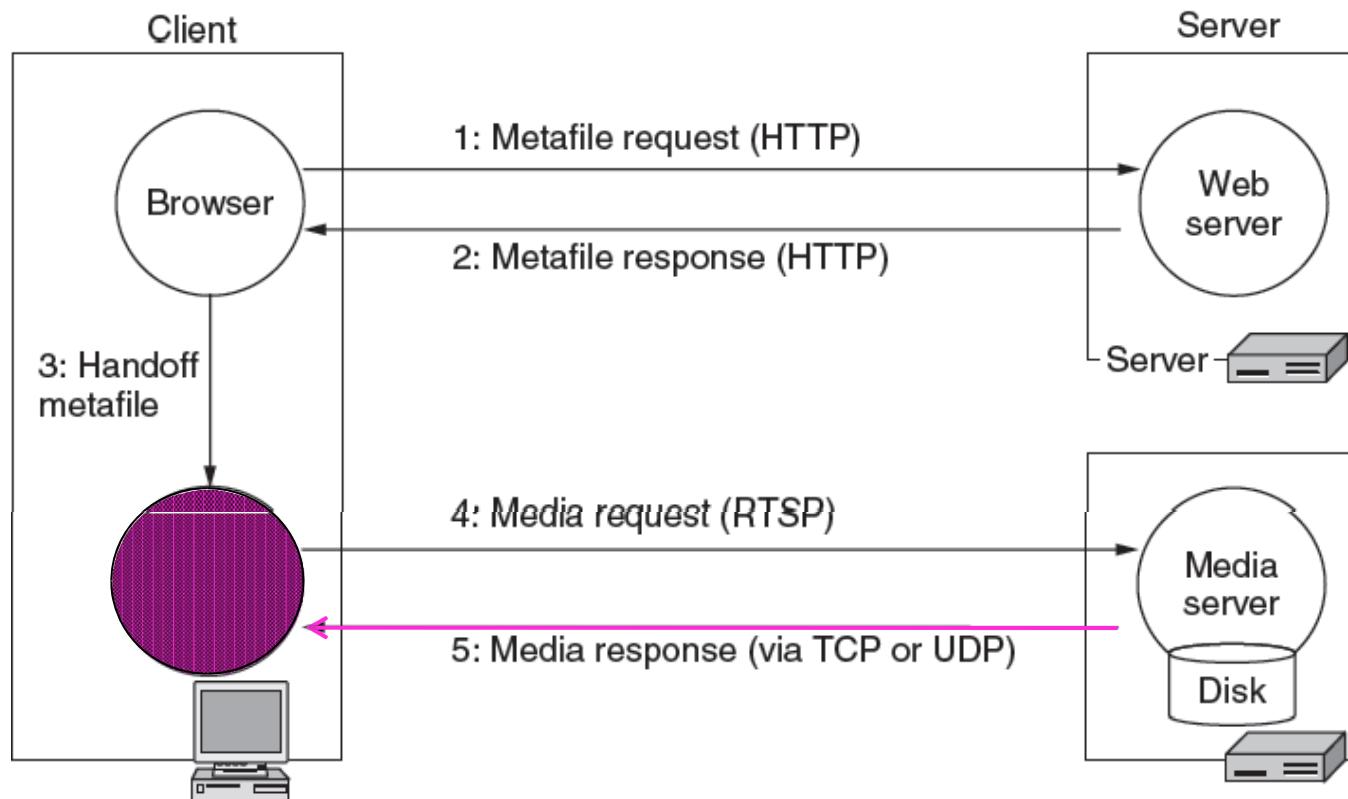
- But has large startup delay, except for short files



Streaming Stored Media (2)

Effective streaming starts the playout during transport

- With RTSP (Real-Time Streaming Protocol)



Streaming Stored Media (3)

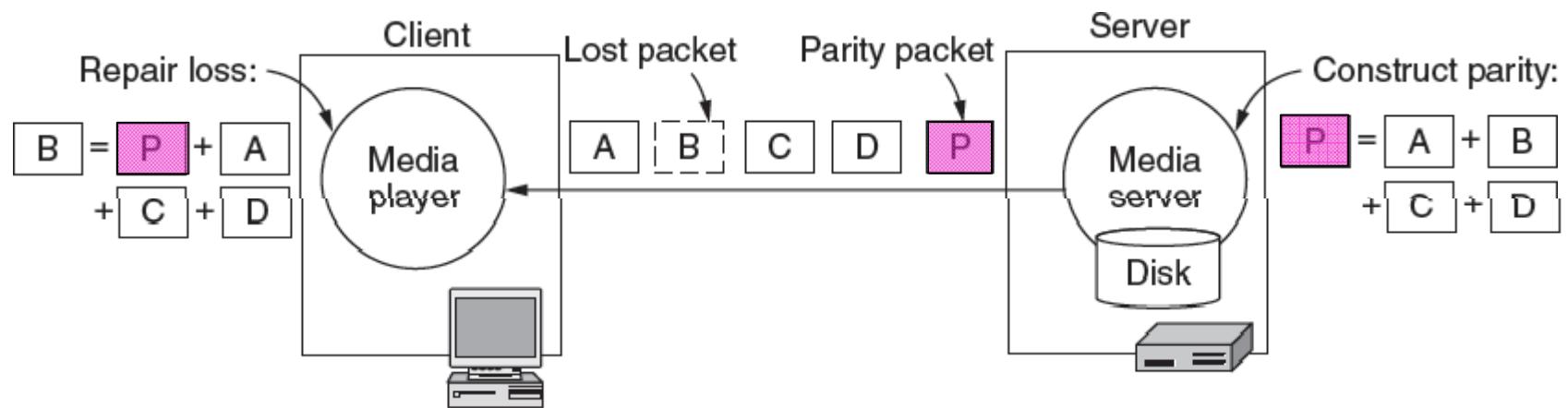
Key problem: how to handle transmission errors

| Strategy | Advantage | Disadvantage |
|------------------------------|---------------------|--|
| Use reliable transport (TCP) | Repairs all errors | Increases jitter significantly |
| Add FEC (e.g., parity) | Repairs most errors | Increases overhead, decoding complexity and jitter |
| Interleave media | Masks most errors | Slightly increases decoding complexity and jitter |

Streaming Stored Media (4)

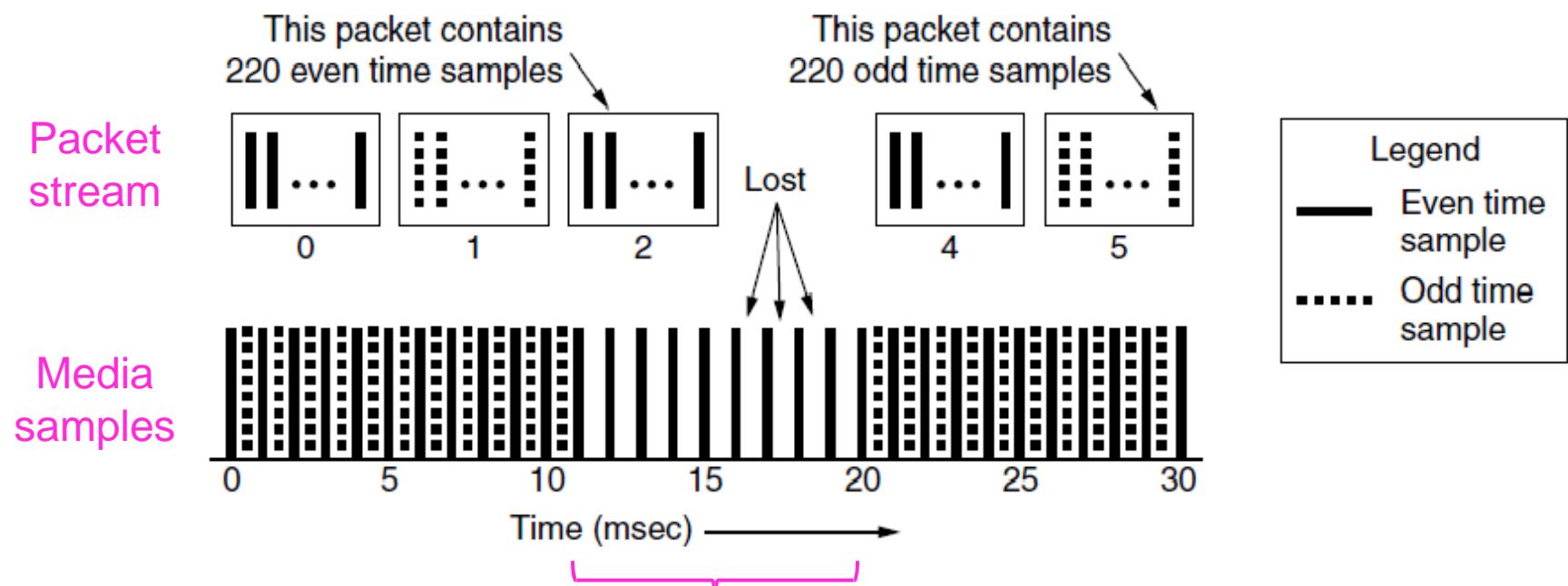
Parity packet can repair one lost packet in a group of N

- Decoding is delayed for N packets



Streaming Stored Media (5)

Interleaving spreads nearby media samples over different transmissions to reduce the impact of loss

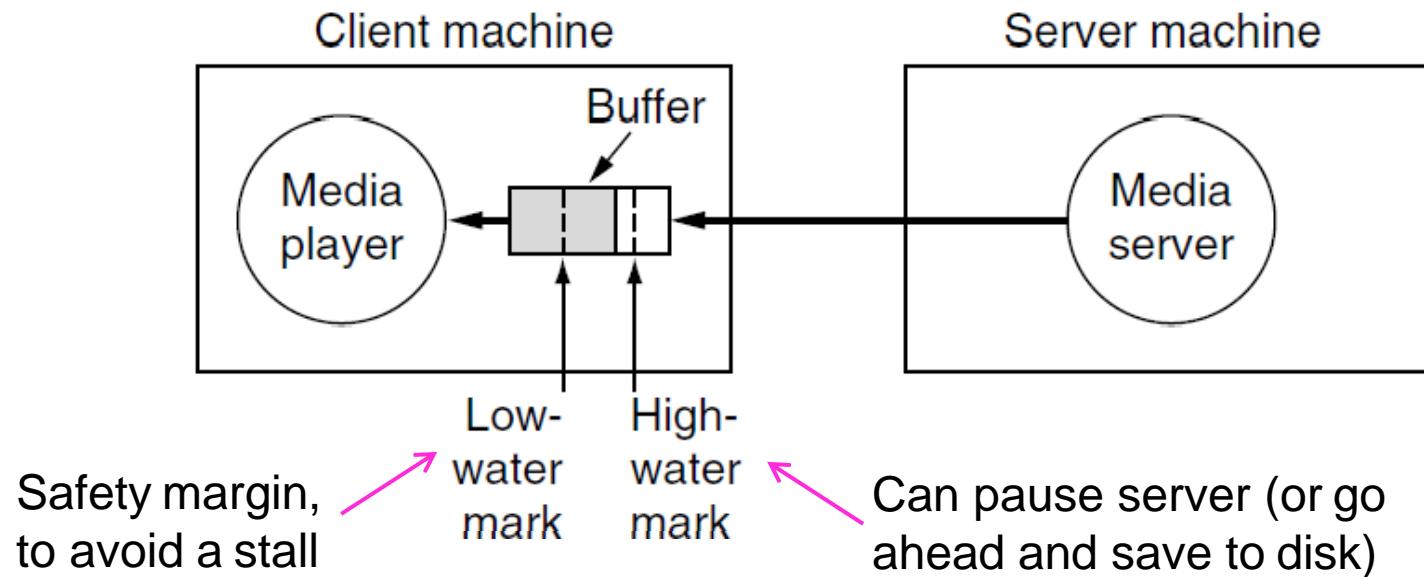


Loss reduces temporal resolution; doesn't leave a gap

Streaming Stored Media (6)

Key problem: media may not arrive in time for playout due to variable bandwidth and loss/retransmissions

- Client buffers media to absorb jitter; we still need to pick an achievable media rate



Streaming Stored Media (7)

RTSP commands

- Sent from player to server to adjust streaming

| Command | Server action |
|----------------|---|
| DESCRIBE | List media parameters |
| SETUP | Establish a logical channel between the player and the server |
| PLAY | Start sending data to the client |
| RECORD | Start accepting data from the client |
| PAUSE | Temporarily stop sending data |
| TEARDOWN | Release the logical channel |

Streaming Live Media (1)

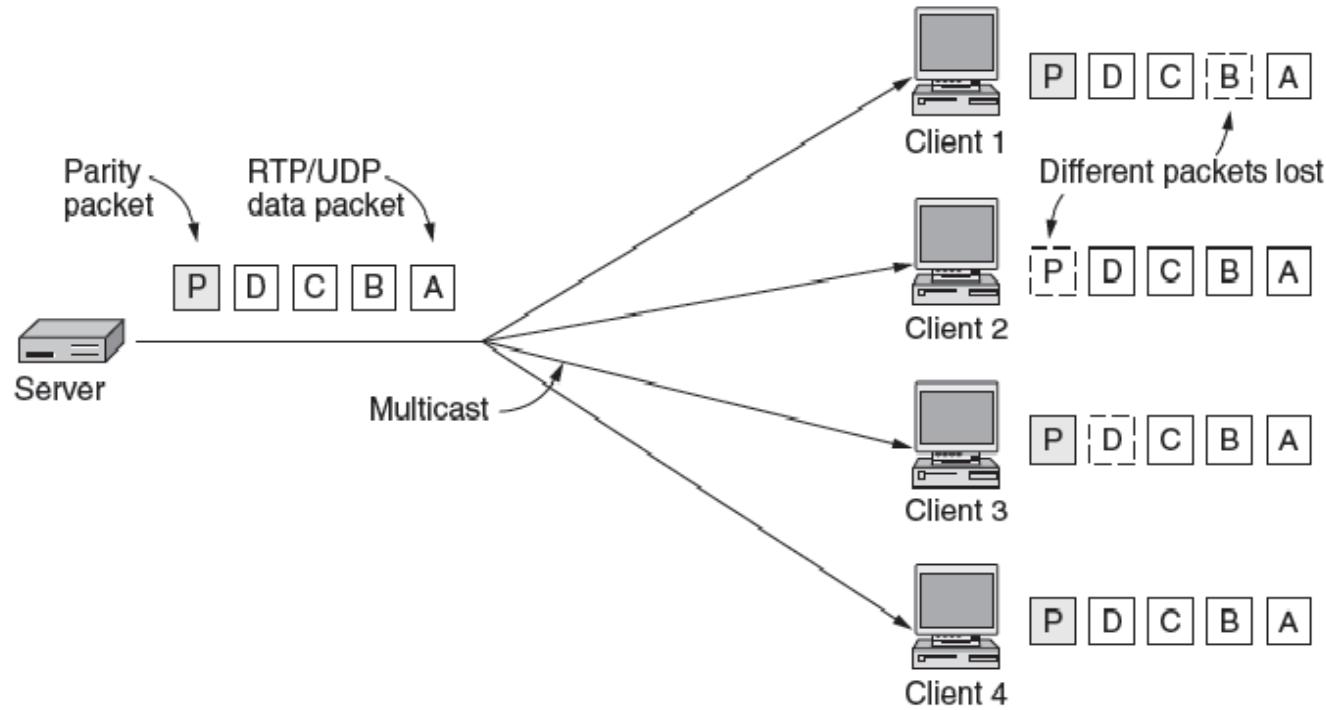
Streaming live media is similar to the stored case plus:

- Can't stream faster than "live rate" to get ahead
 - Usually need larger buffer to absorb jitter
- Often have many users viewing at the same time
 - UDP with multicast greatly improves efficiency. It is rarely available, so many TCP connections are used.
 - For very many users, content distribution is used [later]

Streaming Live Media (2)

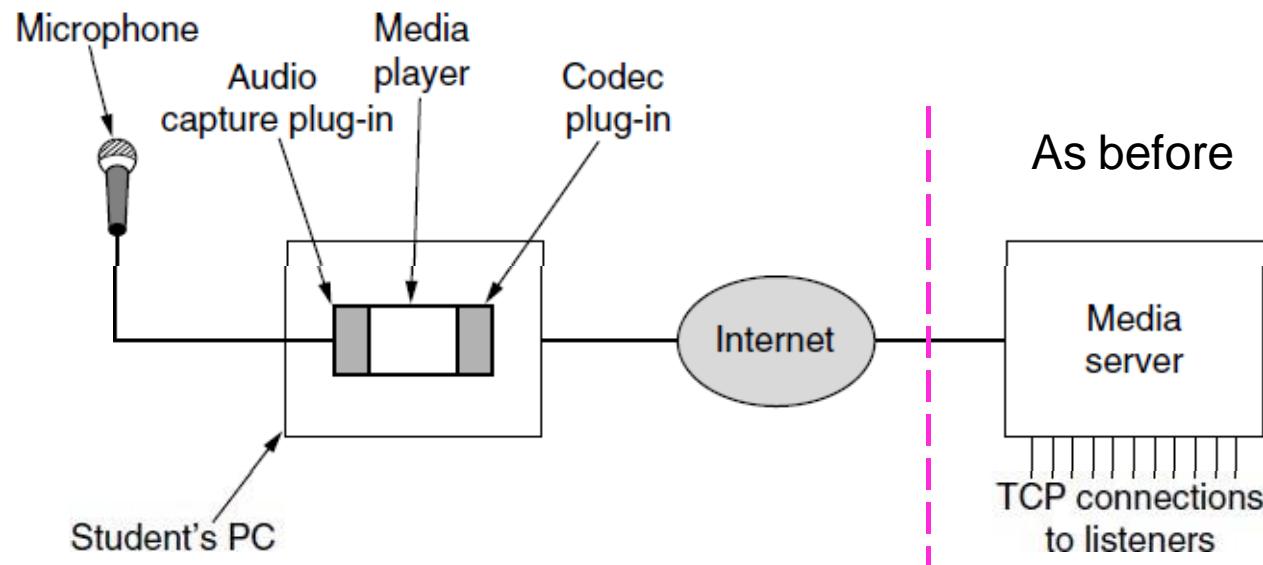
With multicast streaming media, parity is effective

- Clients can each lose a different packet and recover



Streaming Live Media (2)

Production side of a student radio station.



Real-Time Conferencing (1)

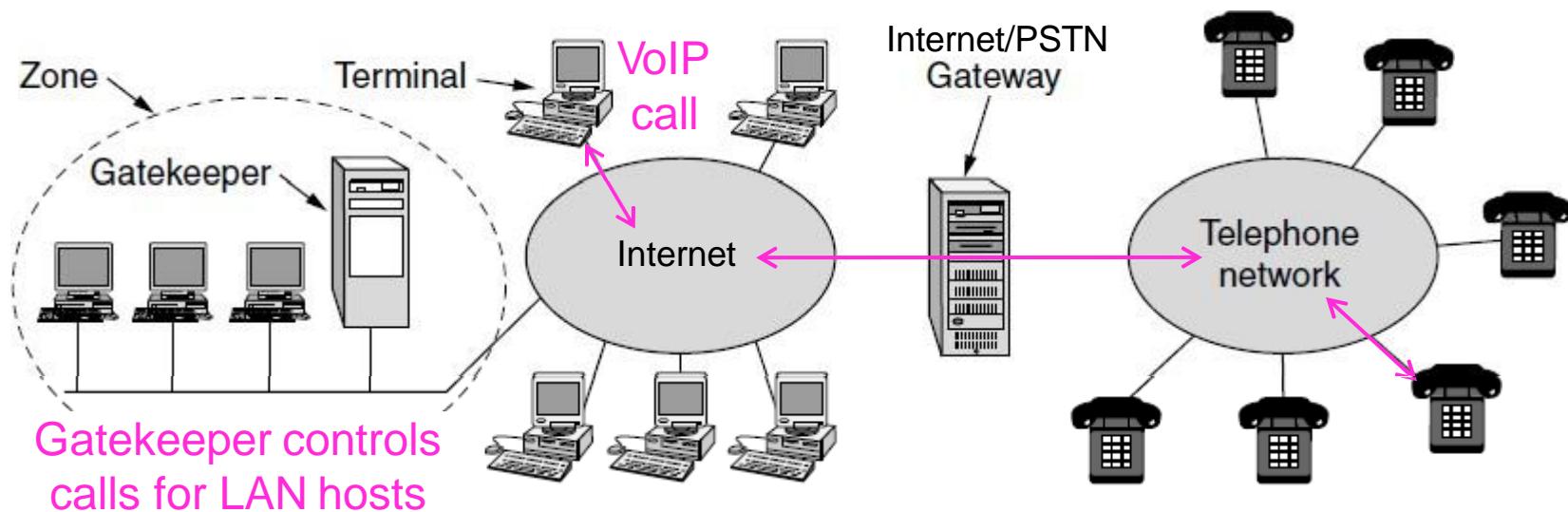
Real-time conferencing has two or more connected live media streams, e.g., voice over IP, Skype video call

Key issue over live streaming is low (interactive) latency

- Want to reduce delay to near propagation
- Benefits from network support, e.g., QoS
- Or, benefits from ample bandwidth (no congestion)

Real-Time Conferencing (2)

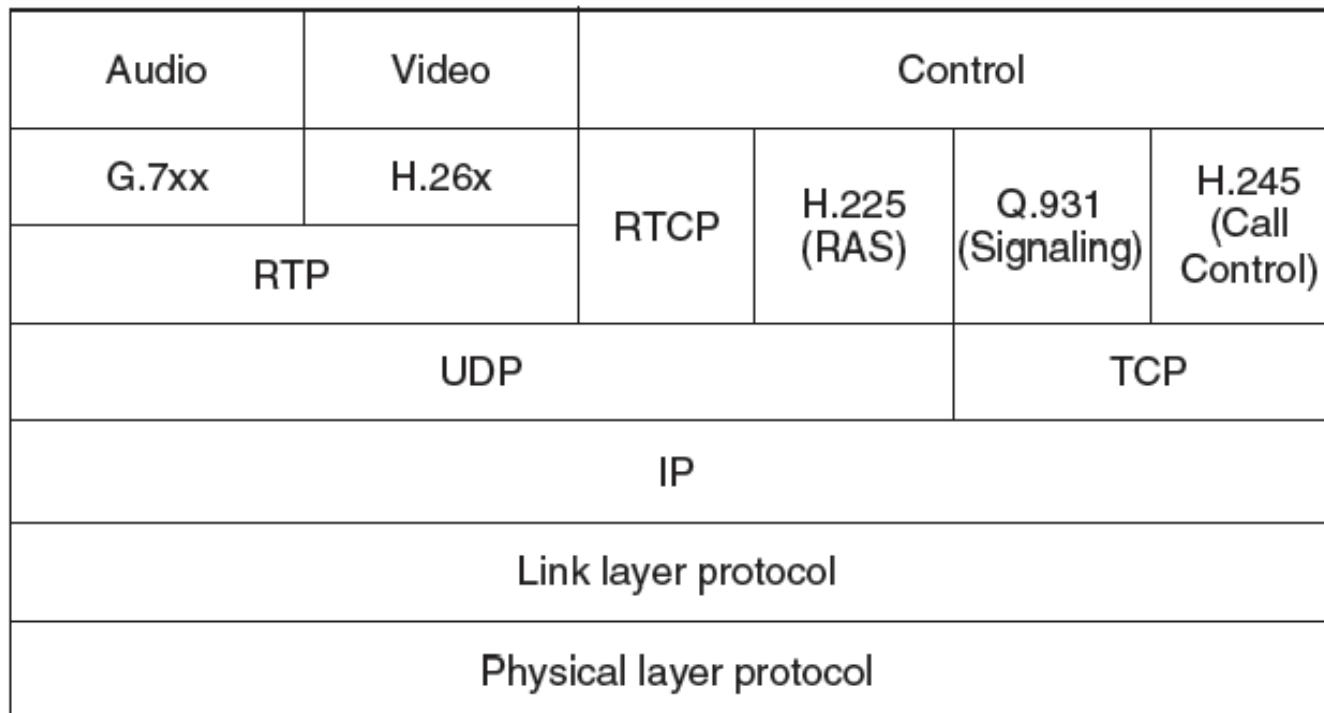
H.323 architecture for Internet telephony supports calls between Internet computers and PSTN phones.



Real-Time Conferencing (3)

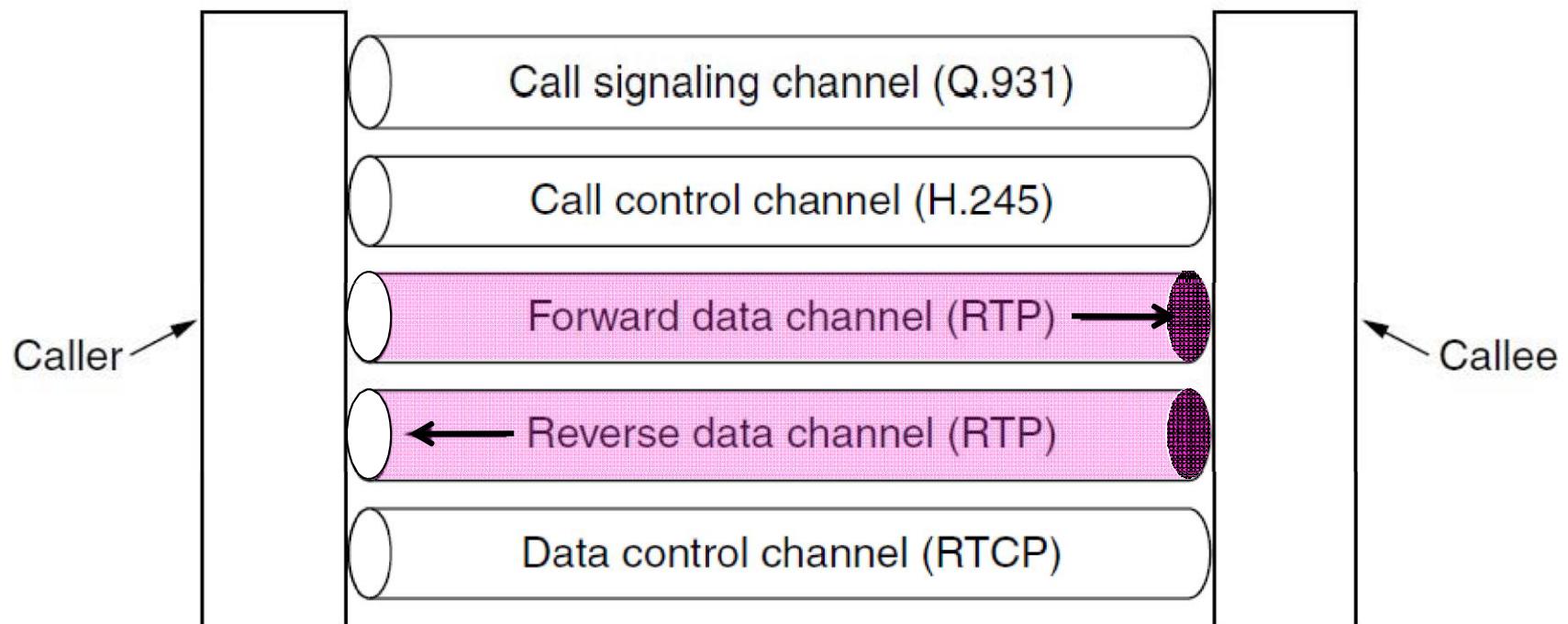
H.323 protocol stack:

- Call is digital audio/video over RTP/UDP/IP
- Call setup is handled by other protocols (Q.931 etc.)



Real-Time Conferencing (4)

Logical channels that make up an H.323 call



Real-Time Conferencing (5)

SIP (Session Initiation Protocol) is an alternative to H.323 to set up real-time calls

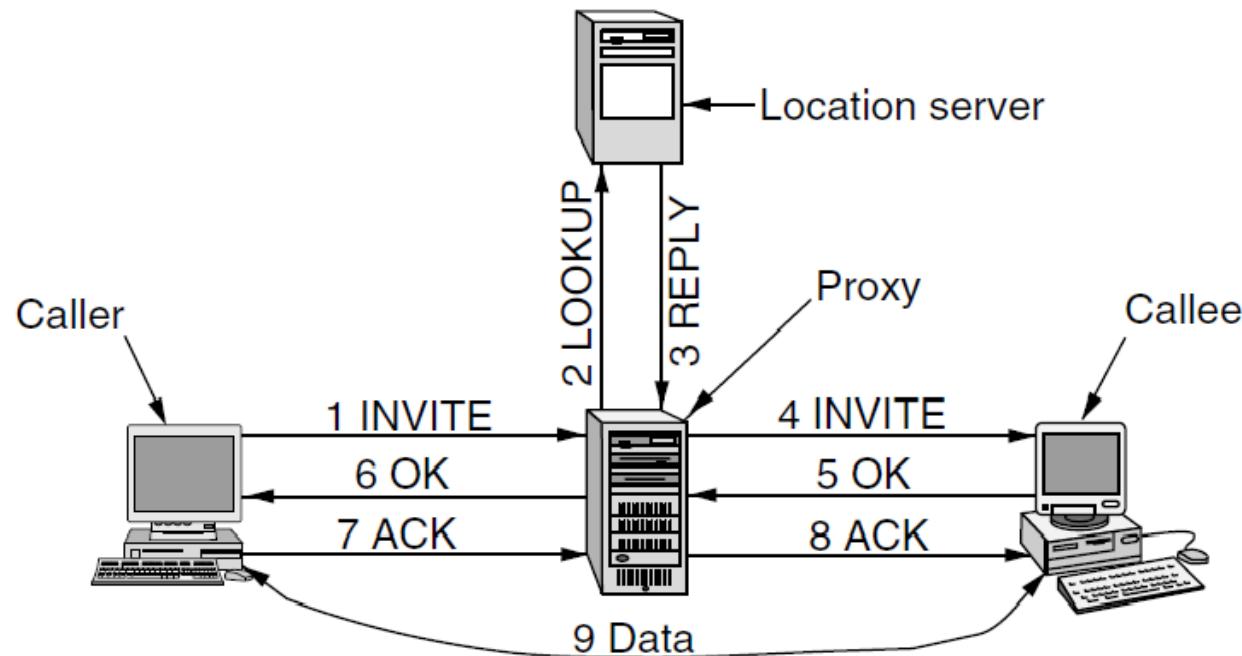
- Simple, text-based protocol with URLs for addresses
- Data is carried with RTP / RTCP as before

| Method | Description |
|----------|---|
| INVITE | Request initiation of a session |
| ACK | Confirm that a session has been initiated |
| BYE | Request termination of a session |
| OPTIONS | Query a host about its capabilities |
| CANCEL | Cancel a pending request |
| REGISTER | Inform a redirection server about the user's current location |

Real-Time Conferencing (6)

Setting up a call with the SIP three-way handshake

- Proxy server lets a remote callee be connected
- Call data flows directly between caller/callee



Real-Time Conferencing (7)

| Item | H.323 | SIP |
|-----------------------------|-------------------------|-------------------------|
| Designed by | ITU | IETF |
| Compatibility with PSTN | Yes | Largely |
| Compatibility with Internet | Yes, over time | Yes |
| Architecture | Monolithic | Modular |
| Completeness | Full protocol stack | SIP just handles setup |
| Parameter negotiation | Yes | Yes |
| Call signaling | Q.931 over TCP | SIP over TCP or UDP |
| Message format | Binary | ASCII |
| Media transport | RTP/RTCP | RTP/RTCP |
| Multiparty calls | Yes | Yes |
| Multimedia conferences | Yes | No |
| Addressing | URL or phone number | URL |
| Call termination | Explicit or TCP release | Explicit or timeout |
| Instant messaging | No | Yes |
| Encryption | Yes | Yes |
| Size of standards | 1400 pages | 250 pages |
| Implementation | Large and complex | Moderate, but issues |
| Status | Widespread, esp. video | Alternative, esp. voice |

Comparison of H.323 and SIP.

Content Delivery

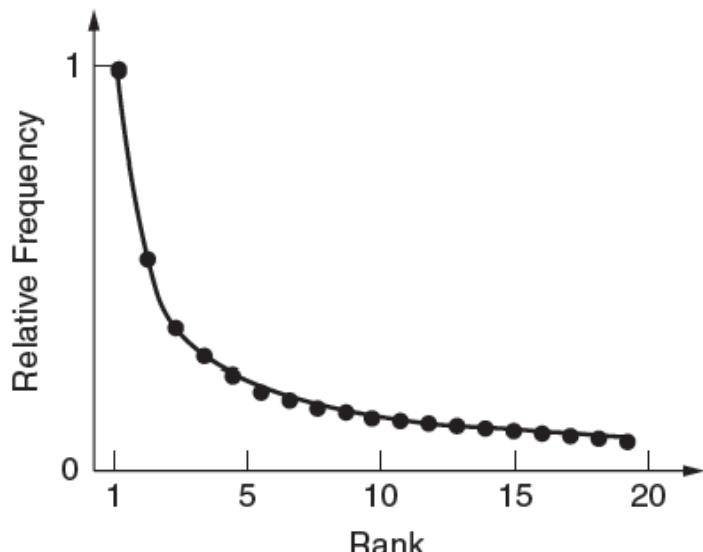
Delivery of content, especially Web and video, to users is a major component of Internet traffic

- Content and Internet traffic »
- Server farms and Web proxies »
- Content delivery networks »
- Peer-to-peer networks »

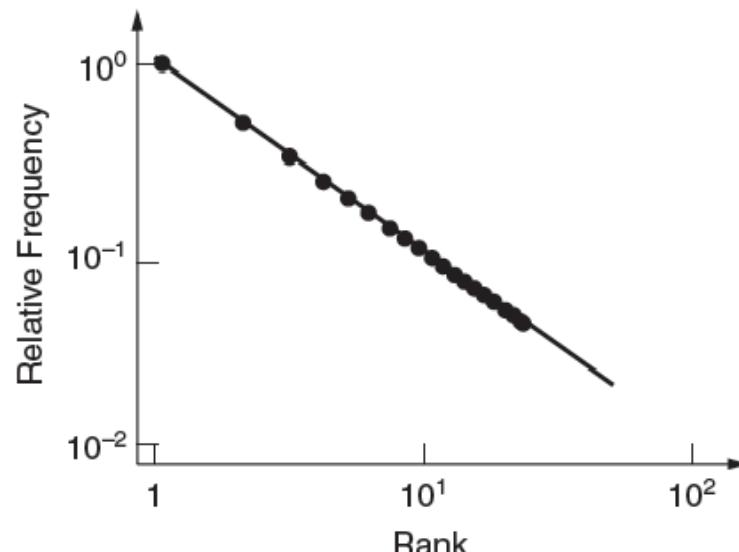
Content and Internet Traffic

Internet traffic:

1. Shifts seismically (email → FTP → Web → P2P → video)
2. Has many small/unpopular and few large/popular flows – mice and elephants



Zipf popularity distribution, $1/k$

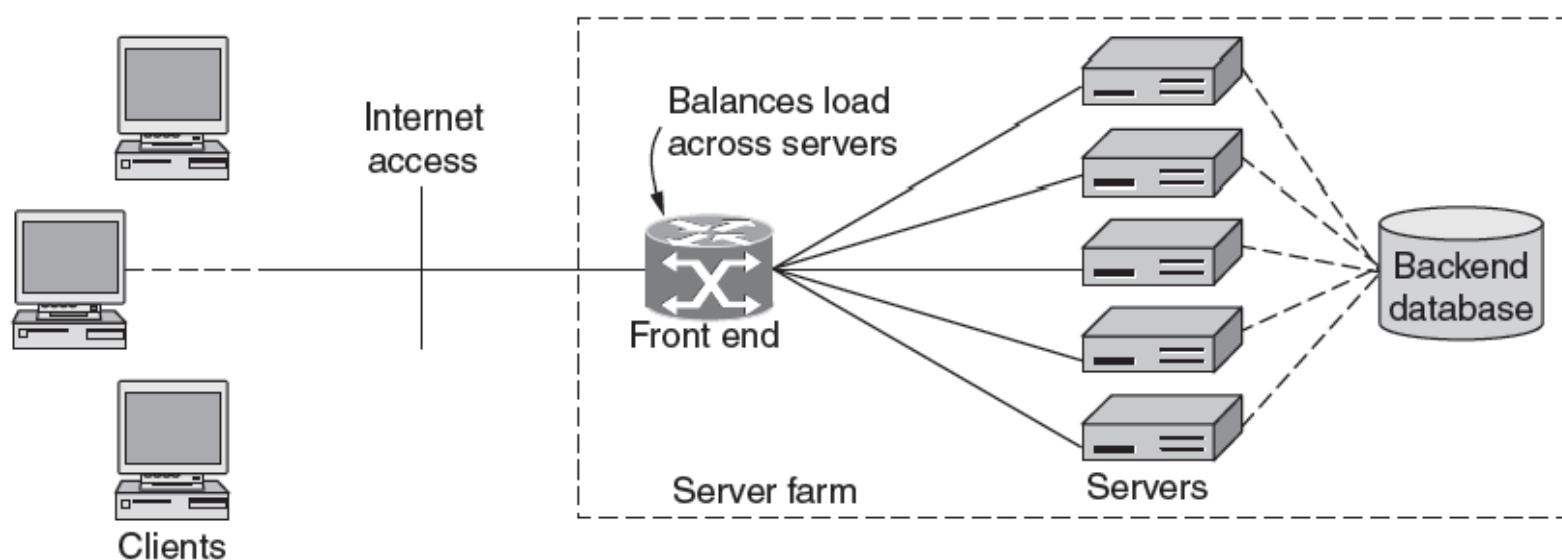


Shows up as a line on log-log plot

Server Farms and Web Proxies (1)

Server farms enable large-scale Web servers:

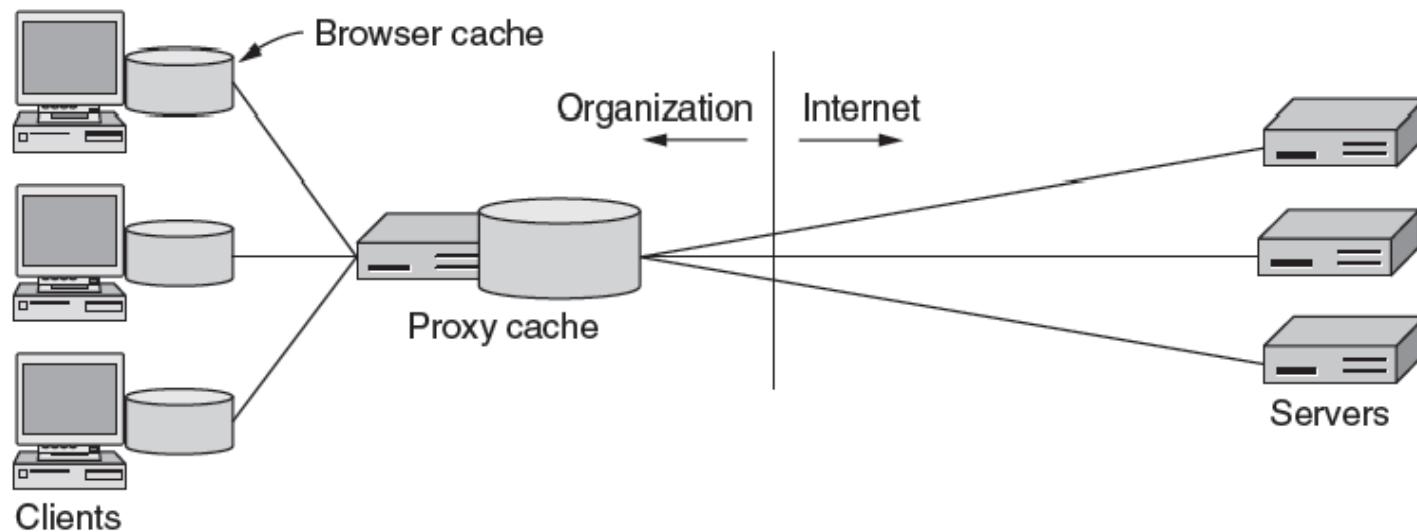
- Front-end load-balances requests over servers
- Servers access the same backend database



Server Farms and Web Proxies (2)

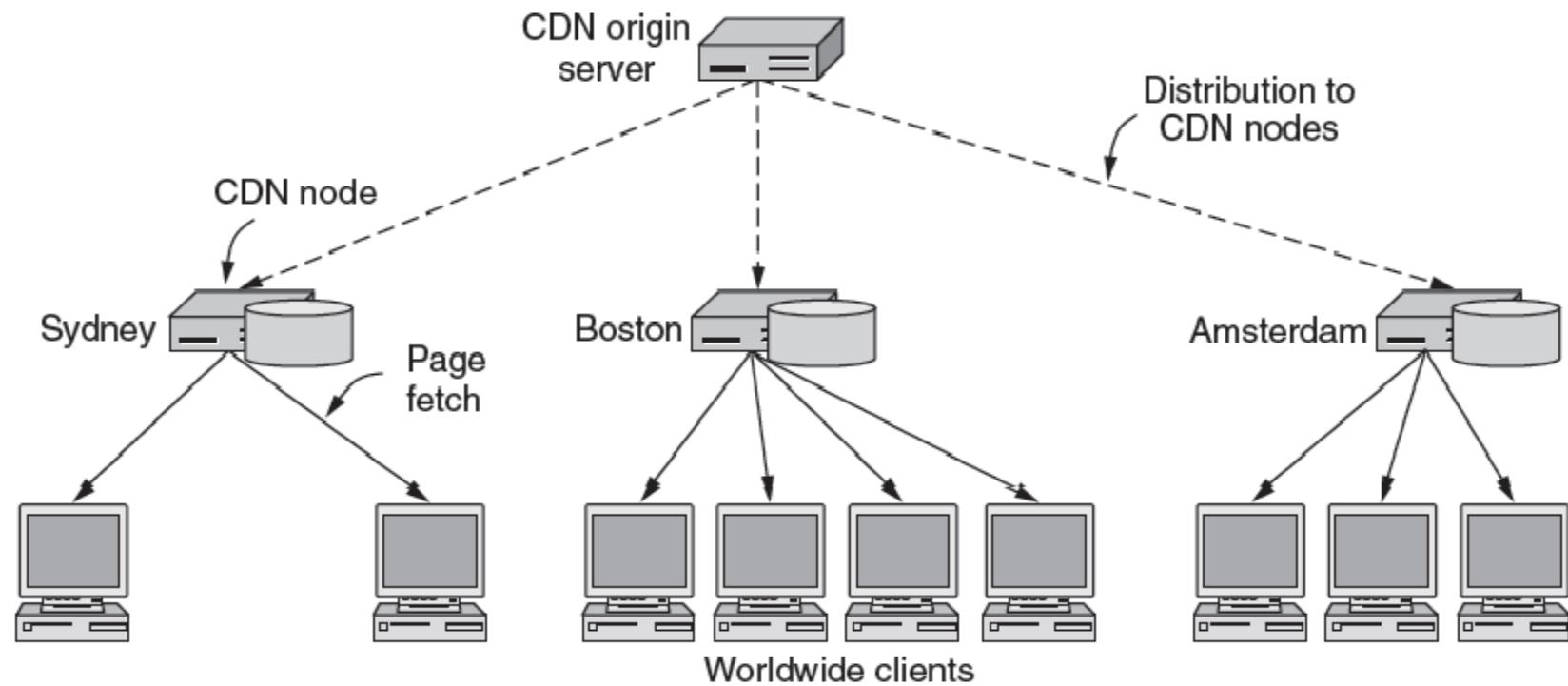
Proxy caches help organizations to scale the Web

- Caches server content over clients for performance
- Also implements organization policies (e.g., access)



CDNs – Content Delivery Networks (1)

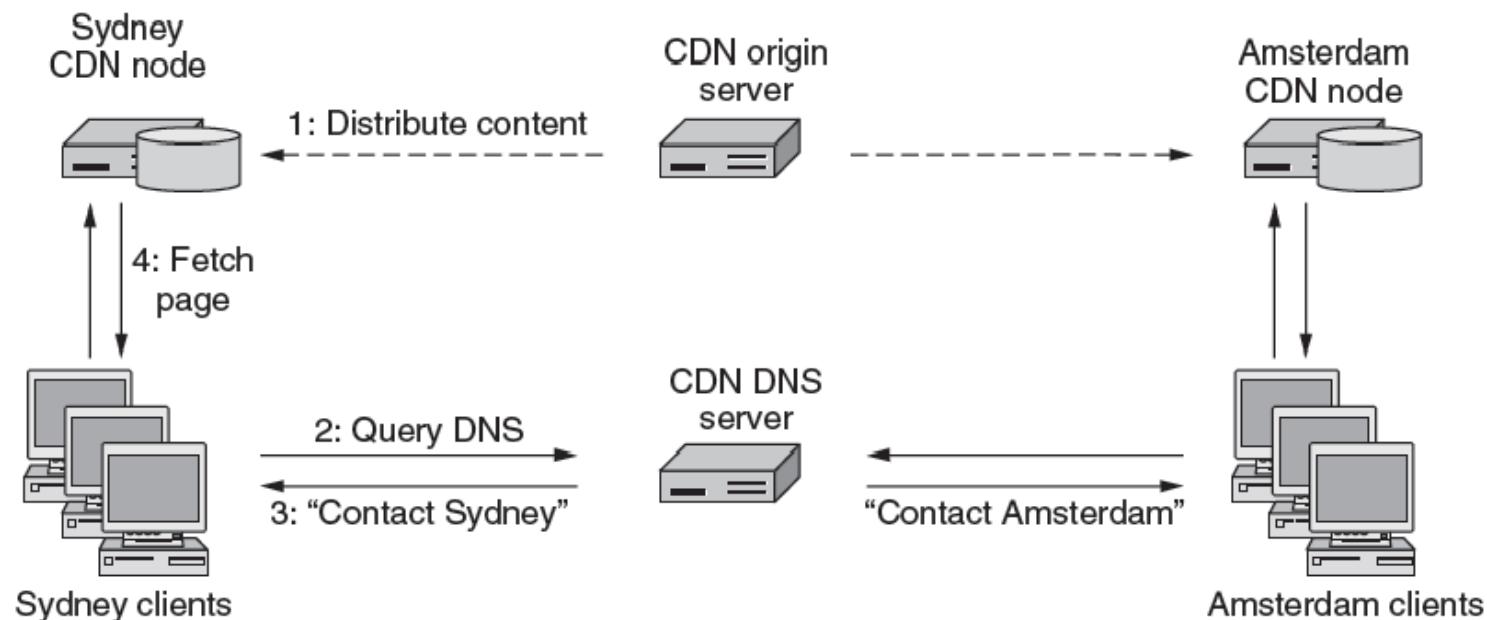
CDNs scale Web servers by having clients get content from a nearby CDN node (cache)



Content Delivery Networks (2)

Directing clients to nearby CDN nodes with DNS:

- Client query returns local CDN node as response
- Local CDN node caches content for nearby clients
and reduces load on the origin server



Content Delivery Networks (3)

Origin server rewrites pages to serve content via CDN

```
<html>
<head> <title> Fluffy Video </title> </head>
<body>
<h1> Fluffy Video's Product List </h1>
<p> Click below for free samples. </p>
<a href="koalas.mpg"> Koalas Today </a> <br>
<a href="kangaroos.mpg"> Funny Kangaroos </a> <br>
<a href="wombats.mpg"> Nice Wombats </a> <br>
</body>
</html>
```

Traditional Web page on server

```
<html>
<head> <title> Fluffy Video </title> </head>
<body>
<h1> Fluffy Video's Product List </h1>
<p> Click below for free samples. </p>
<a href="http://www.cdn.com/fluffyvideo/koalas.mpg"> Koalas Today </a> <br>
<a href="http://www.cdn.com/fluffyvideo/kangaroos.mpg"> Funny Kangaroos </a> <br>
<a href="http://www.cdn.com/fluffyvideo/wombats.mpg"> Nice Wombats </a> <br>
</body>
</html>
```

Page that distributes content via CDN

Peer-to-Peer Networks (1)

P2P (Peer-to-Peer) is an alternative CDN architecture with no dedicated infrastructure (i.e., servers)

- Clients serve content to each other as peers

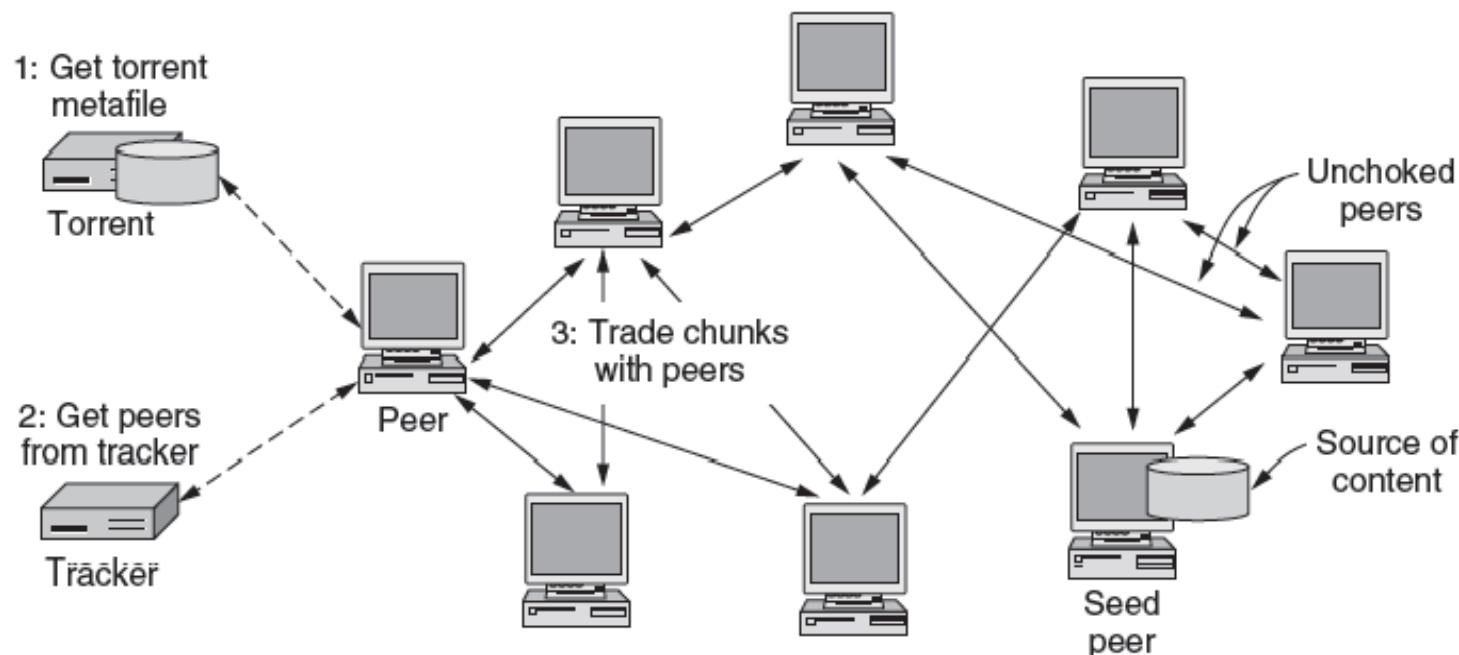
Challenges when servers are removed:

1. How do peers find each other?
2. How do peers support rapid content downloads?
3. How do peers encourage each other to upload?

Peer-to-Peer Networks (2)

BitTorrent lets peers download torrents

- Peers find each other via Tracker in torrent file
- Peers swap chunks (parts of content) with partners, preferring those who send most quickly [2]
- Many peers speed download; preference helps uploads [3]



Peer-to-Peer Networks (3)

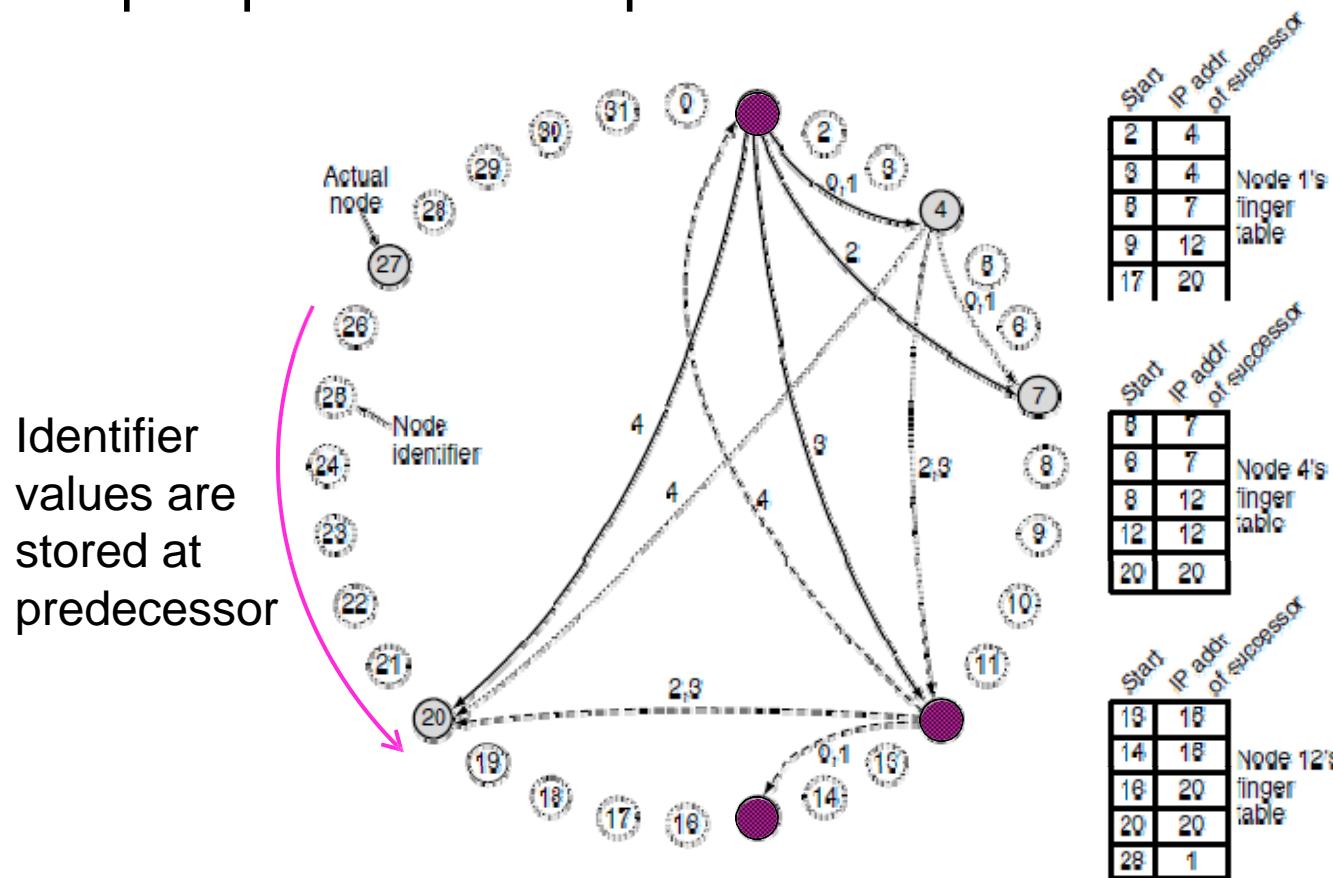
Distributed Hash Tables (DHTs) are a fully distributed index that scales to very many clients/entries

- Need to follow $O(\log N)$ path for N entries
- Can use as Tracker to find peers with no servers [1]
- Look up torrent (identifier) in DHT to find IP of peers
- Kademlia is used in BitTorrent

Peer-to-Peer Networks (3)

A Chord ring of 32 identifiers. Finger tables [at right, and as arcs] are used to navigate the ring.

- Example: path to look up 16 from 1 is $1 \rightarrow 12 \rightarrow 15$



About Network Security

Some different adversaries and security threats

- Different threats require different defenses

| Adversary | Goal |
|-------------|---|
| Student | To have fun snooping on people's email |
| Cracker | To test out someone's security system; steal data |
| Sales rep | To claim to represent all of Europe, not just Andorra |
| Businessman | To discover a competitor's strategic marketing plan |
| Ex-employee | To get revenge for being fired |
| Accountant | To embezzle money from a company |
| Stockbroker | To deny a promise made to a customer by email |
| Con man | To steal credit card numbers for sale |
| Spy | To learn an enemy's military or industrial secrets |
| Terrorist | To steal germ warfare secrets |

Cryptography

- Cryptography is a fundamental building block for security mechanisms.
- Another course

Objectives of CS&NW course

- Understand the operational part of any computer.
learn the important concepts which have been evolved for building modern operating systems and networking protocols.
- Understanding the general principles of OS design.
- Focus on general-purpose, multi-user systems.
Emphasis on widely applicable concepts rather than any specific features of any specific OS.
- Understanding problems, solutions and design choices.
- Understanding the structure of specific OSs (UNIX, LINUX, WINDOWS 10 and so on) and TCP/IP

Course topics

Introduction (1 week)

Process and thread management (2 weeks)

CPU Scheduling (1 week)

Process Synchronization (2 weeks)

Deadlocks (1 week)

Memory management (1.5 weeks)

Virtual Memory (1.5 weeks)

File Systems (1 week)

Protection and Security (1 week)

Networking (3 weeks)

References

Text book:

Silberschatz, A, Galvin, P, Gagne, G. Operating System Concepts, Addison-Wesley (8th or latest edition).

Computer Networks (5th Edition) Andrew S. Tanenbaum, David J. Wetherall Prentice Hall.

Other Reference BOOKS:

William Stallings, Operating systems, Prentice-Hall, 1998.

Operating Systems, Gary Nutt, Pearson Education

Charles Crowley, Operating Systems: A design-oriented approach, Tata McGraw-Hill, 1997.

Operating Systems: Concepts and Design, Milan Milenkovic, TATA McGRAW-HILL

Tanenbaum, A., Modern Operating Systems, Prentice-Hall, second edition, 2000.

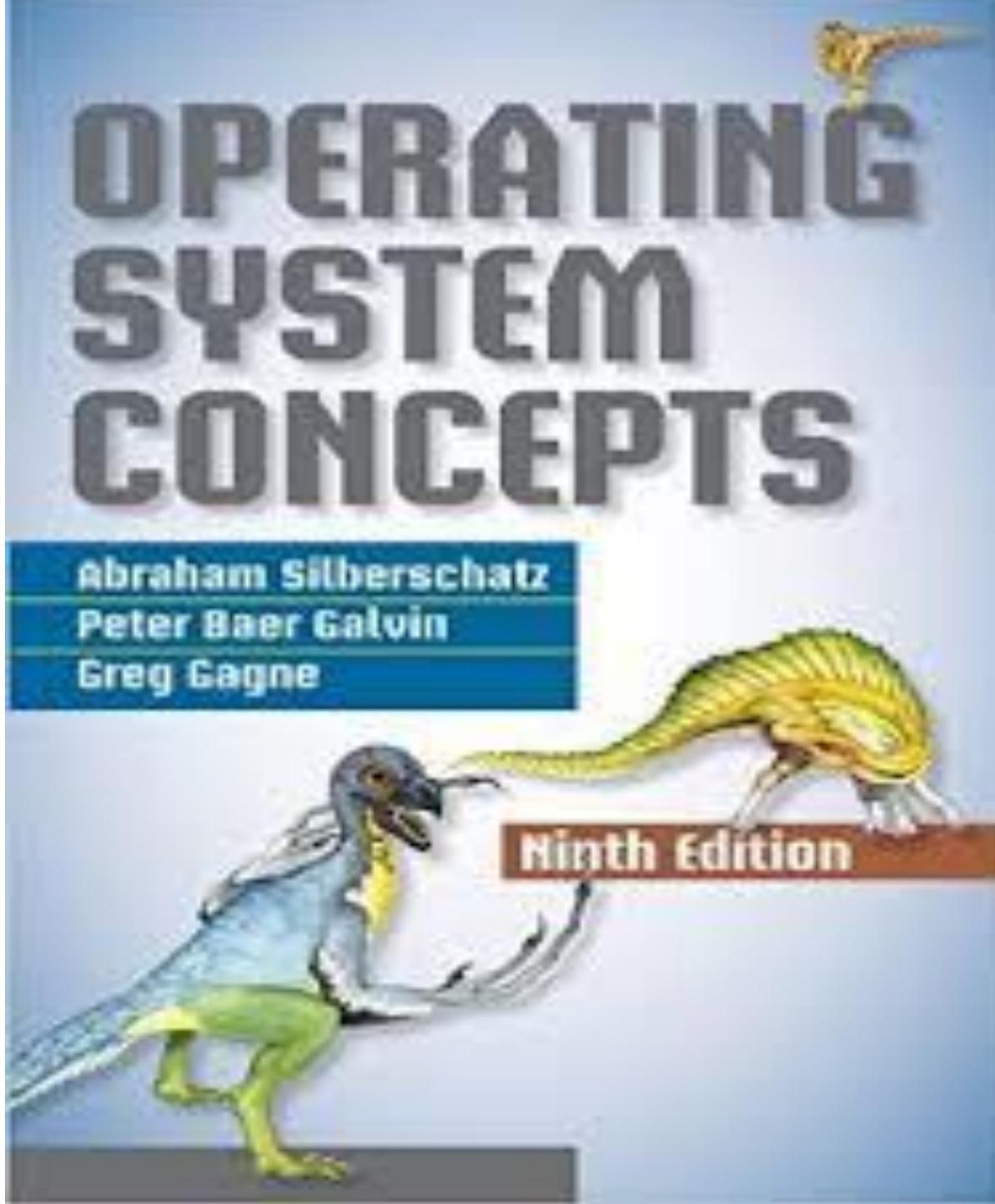
OPERATING SYSTEM CONCEPTS

Abraham Silberschatz

Peter Baer Galvin

Greg Gagne

Ninth Edition



FIFTH EDITION

COMPUTER NETWORKS



TANENBAUM | WETHERALL

OUTCOME

- After completing the course, the students will understand
 - fundamental concepts of several computer operating systems (SOLARIS, LINUX, WINDOWS, MAC, Adroid,...) and network based services (Skype, Google Hangouts,...)
 - the solutions/options to interesting problems which have been encountered by the designers of the preceding operating systems and networking protocols, and
 - the critical role of the operation system and networking in designing several computer and network based systems like database systems, expert systems, web based information systems, multi-media systems, embedded systems, internet services and so on.

END