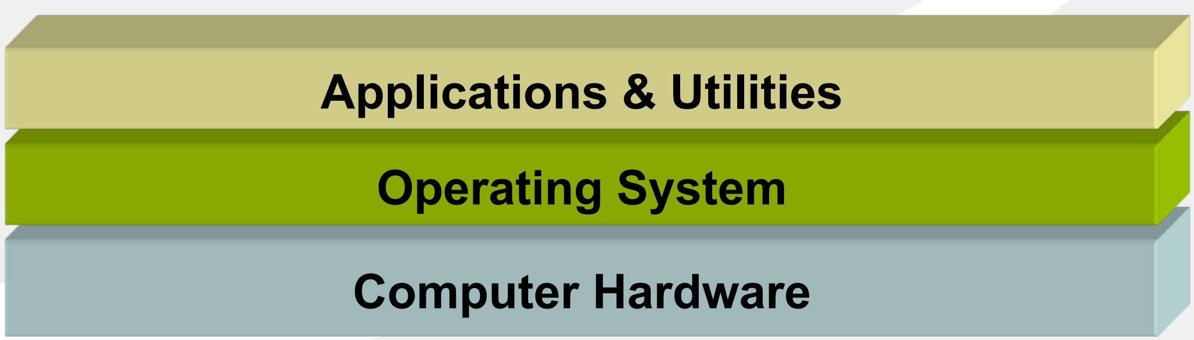
Basics of Operating Systems

What is an Operating System?

- Trusted software interposed between the hardware and application/utilities to improve efficiency and usability
  - Most computing systems have some form of operating systems
  - Hard to use computer systems without OS



How did OS evolve?

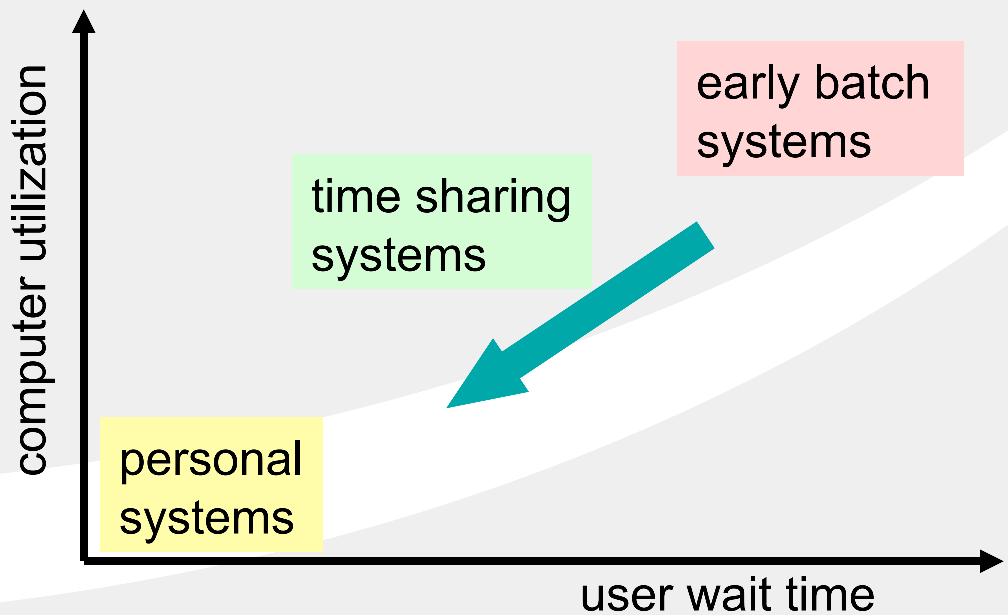
- In the beginning….
  - Computers cost millions of $$
  - Hard to operate (done using console switches)
  - Hard to program – particularly device I/O

How did OS evolve?

- Expensive machines cannot idle... utilization should be very high
  - Batch processing was devised
  - Programming done offline and subroutine were created for common tasks
  - I/O and processing overlap obtained via buffering and interrupts

How did OS evolve?

- OS evolution was driven by two important factors:
  - Cost of Computers (i.e., computer time)
  - Cost of People (i.e., user time)



What are the design concerns?

Systems	Design Concerns
Personal/Embedded systems (e.g., PDAs)	*Software has to be small (storage space limited), power consumption, screen size..*
Time-sharing systems (e.g., Lab workstations)	*Response time, CPU scheduling, data communication, security and integrity of programs and data…*
Batch systems (e.g., IBM mainframes, supercomputers)	*maximize CPU usage*

What are the major OS functions?

- Control access and provide interfaces
  - To the OS and devices attached to the system
  - Provide interfaces for human-machine and machine-machine transactions
- Manage resources
  - Mediate resource usage among different tasks
  - Implement policies

What are the major OS functions?

- Provide abstractions
  - Hide the peculiarities of the hardware.
  - Example: device independent I/O

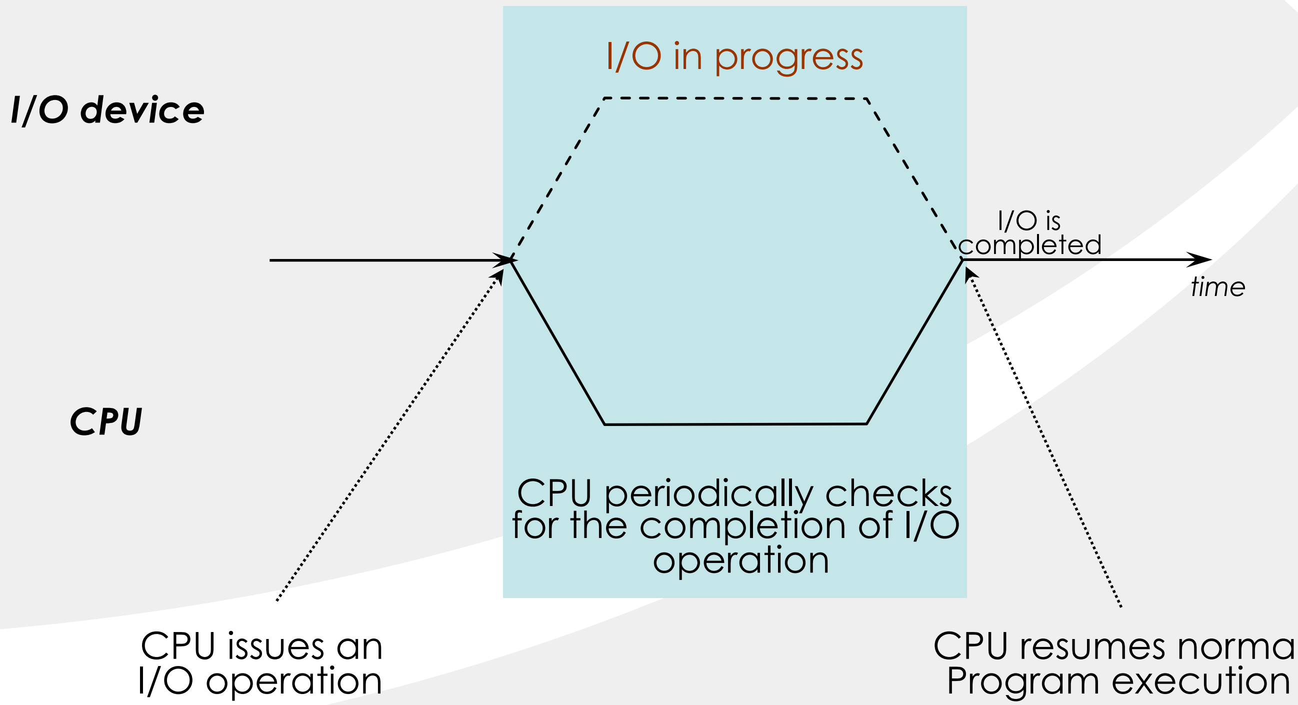
What is the access problem?

- User wants to access OS, why?
- User wants to access devices connected to the system
- It all starts with recognizing the user's intent to access!

Access problem: Handling I/O

- Programmed I/O

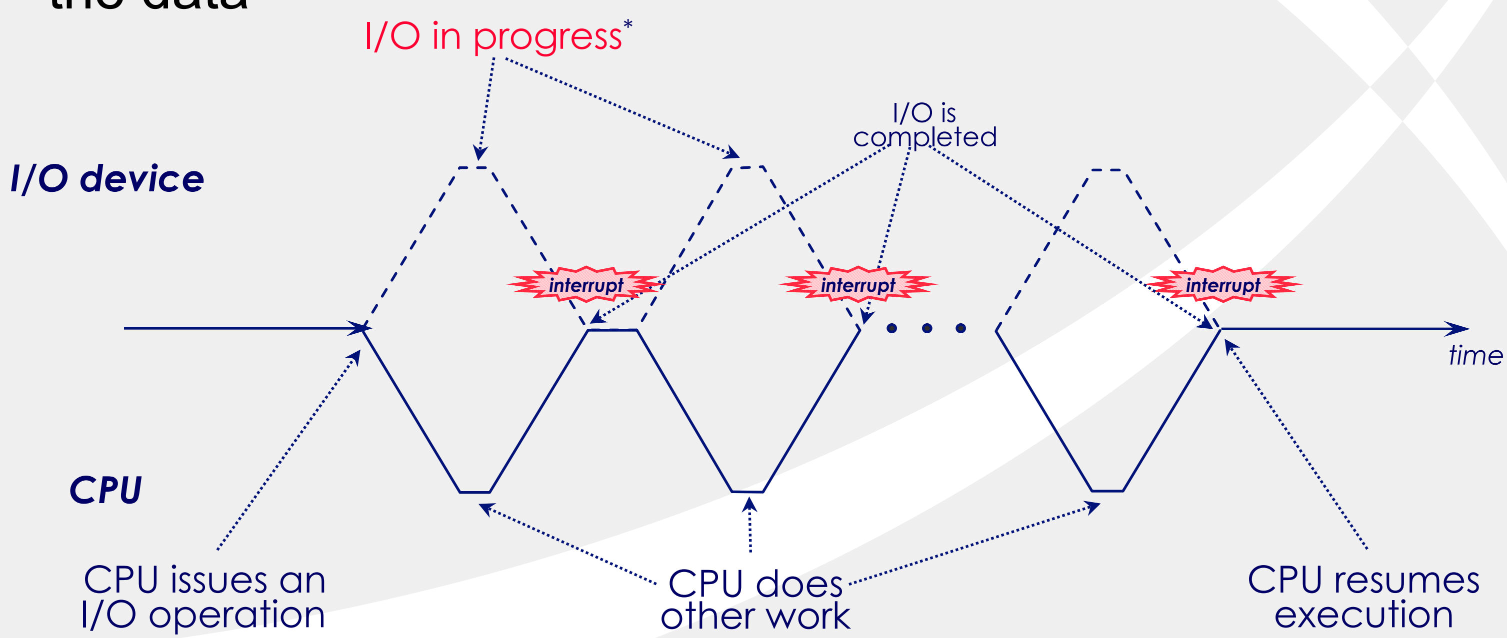
  The CPU transfers the data from (or to) the device buffers. After issuing an I/O operation the CPU continuously checks (polls) for its completion



Access problem: Handling I/O

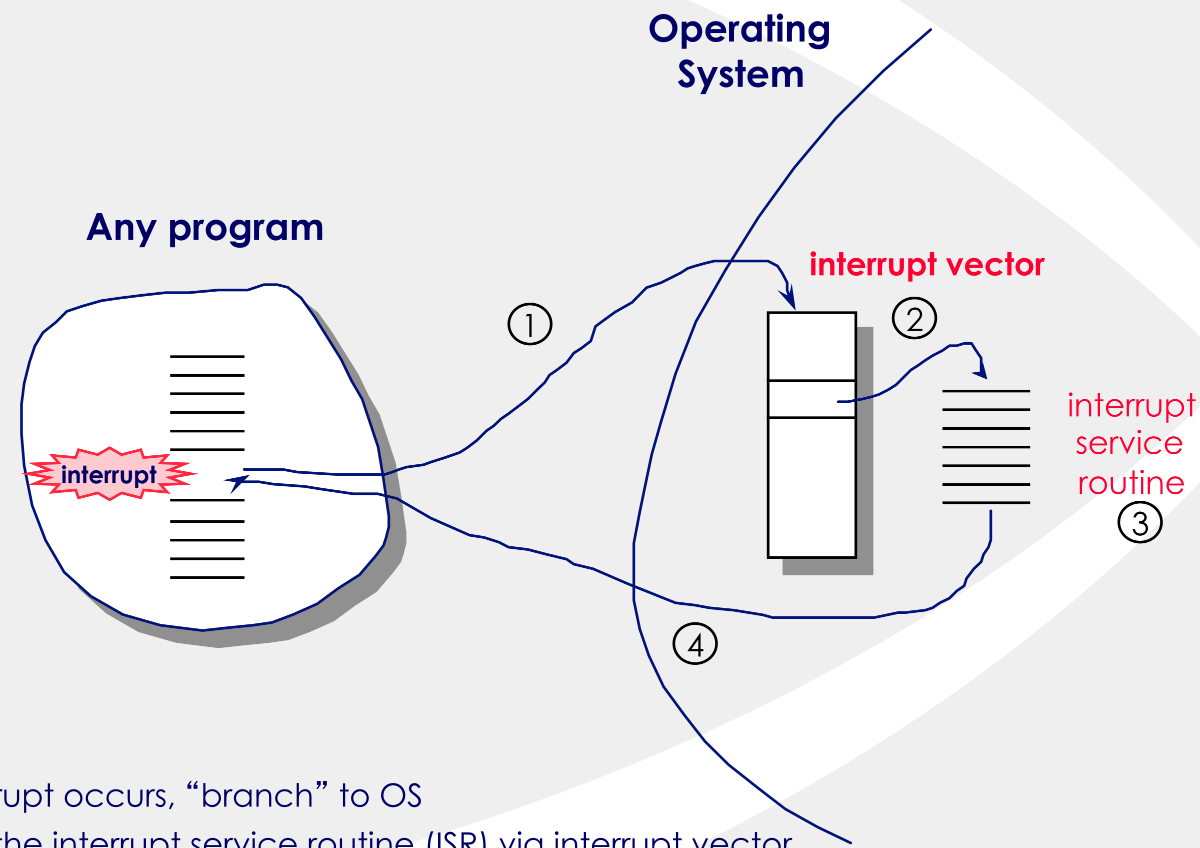
- Interrupt-driven I/O *(slow speed, character device)*

  The CPU issues an I/O operation and goes on; Device notifies (interrupts) the CPU as data arrives; CPU processes the data



* One unit of data (a byte or a word) is transferred.

How OS service interrupts?



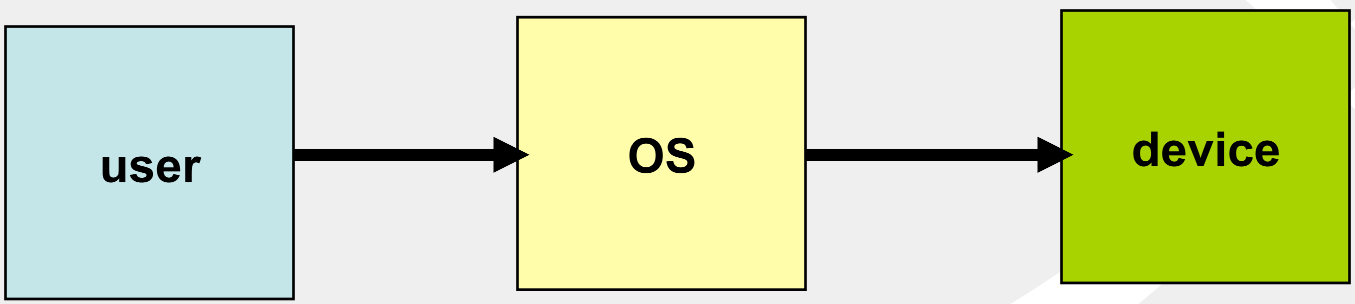
1. An interrupt occurs, “branch” to OS
2. Locate the interrupt service routine (ISR) via interrupt vector
3. Execute the ISR
4. Return to interrupted program

Access problem: processing data

- Once data is captured from device (e.g., keyboard) what is next?
- Imagine you want to write an word processor
  - Capture some keystrokes – for arrow and other control keys
  - Capture words (*cooked* or *raw* form?)

Accessing devices vs. OS services

- OS interposes itself between the hardware and applications/utilities/users



- Controlling access to OS → Controls device access

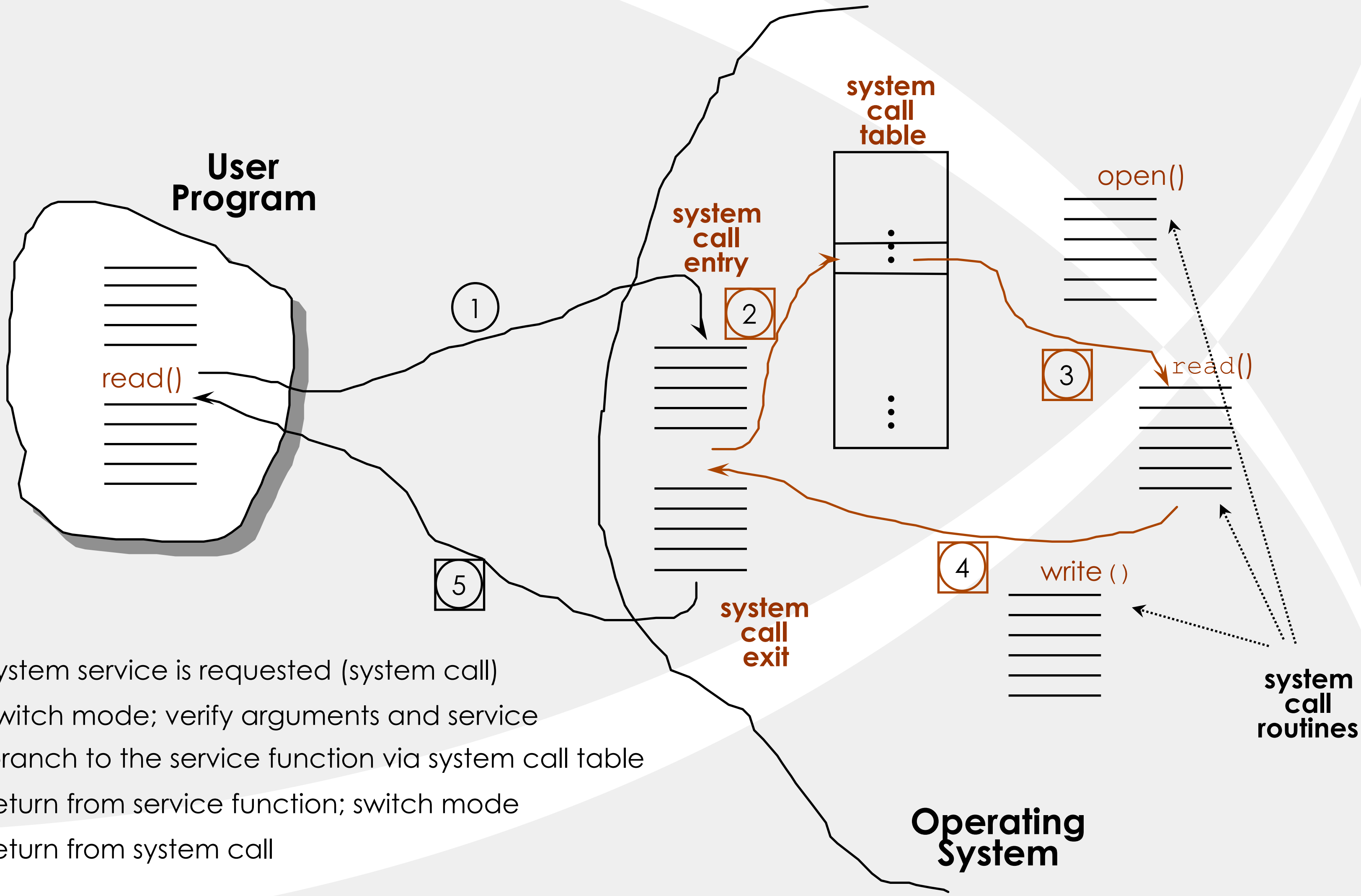
How to control access to OS?

- Provides a two level architecture:
  - Trusted mode
  - Untrusted mode
- The OS (at least the core part – called Kernel) runs in the trusted mode
- User applications/utilities run in the untrusted mode

System call via Trap

- OS provides system calls for accessing OS services from applications
- System calls are special procedure calls
  - Control transfer
  - Switch protection domain
- System calls also:
  - Validate the call
  - Allow authorized actions to take place

How system call is processed?



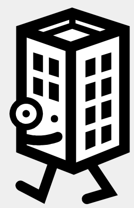
1. system service is requested (system call)
2. switch mode; verify arguments and service
3. branch to the service function via system call table
4. return from service function; switch mode
5. return from system call

Example system calls in UNIX

- Process control
  - *fork(), exec(), wait(), abort()*
- File manipulation
  - *chmod(), link(), stat(), creat()*
- Device manipulation
  - *open(), close(), ioctl(), select()*
- Information maintenance
  - *time(), acct(), gettimeofday()*
- Communications
  - *socket(), accept(), send(), recv()*

How does OS manage resources?

Resources

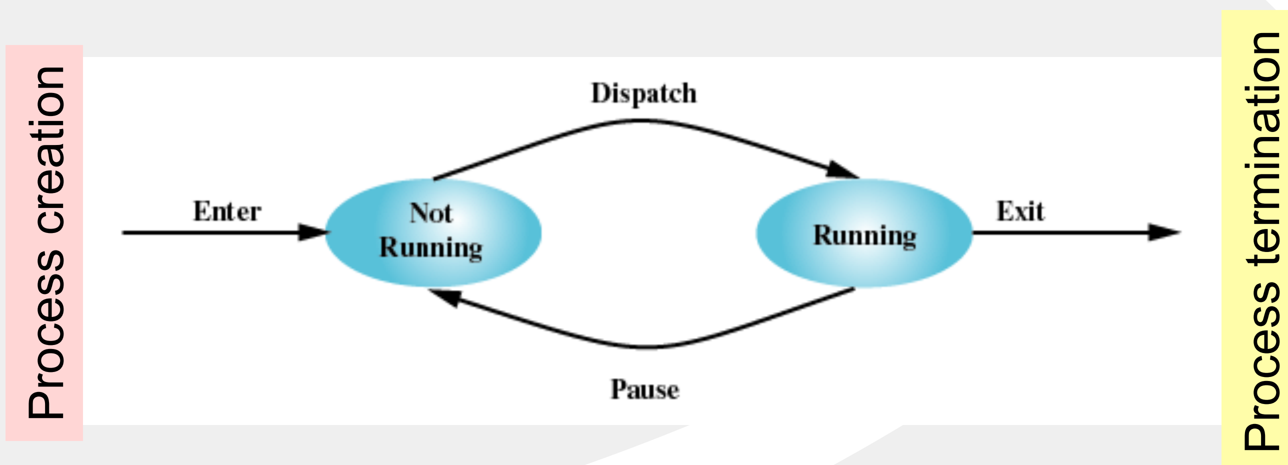


Resource Manager
(Operating System)

Recipients of the
Resources

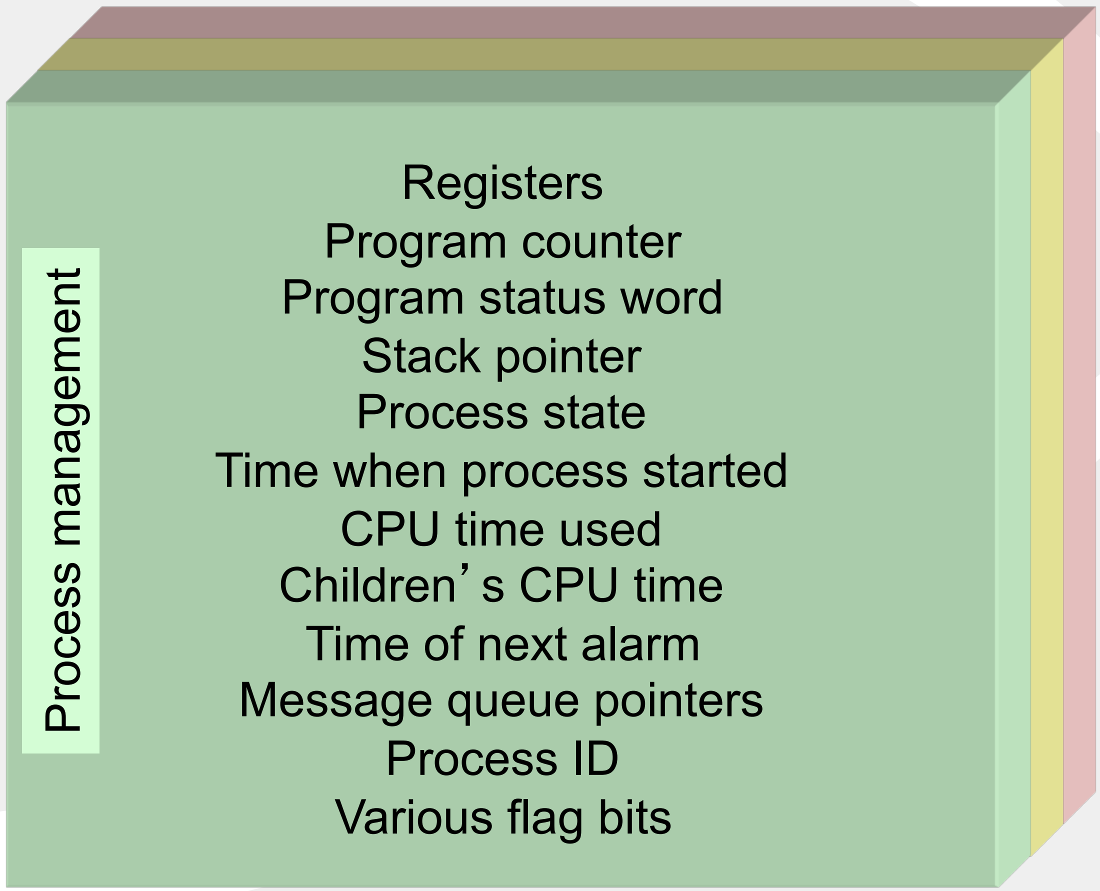
A simple process model

- Below is a two-state process model
  - Running (on the CPU)
  - Not running (waiting to get the CPU or at I/O)



Process: Avatar of the application

- A process represented process table entry

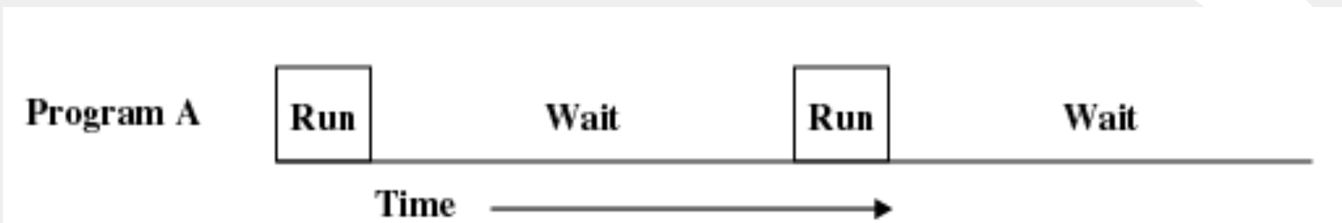


Resource mgmt.: Processor time

- Processor time is the primary resource managed by the OS
- How it is managed depends on the type of OS:
  - Batch vs Time-sharing
  - Uniprogramming vs Multiprogramming

Some examples

- Uniprogramming (only one program is running in the system)

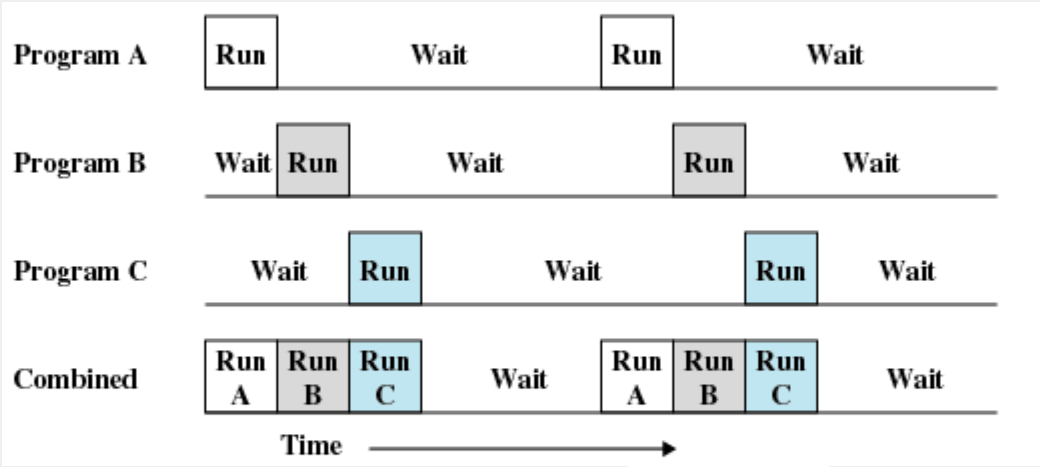


Read one record from file	15 µs
Execute 100 instructions	1 µs
Write one record to file	15 µs
TOTAL	31 µs

What is the CPU utilization?

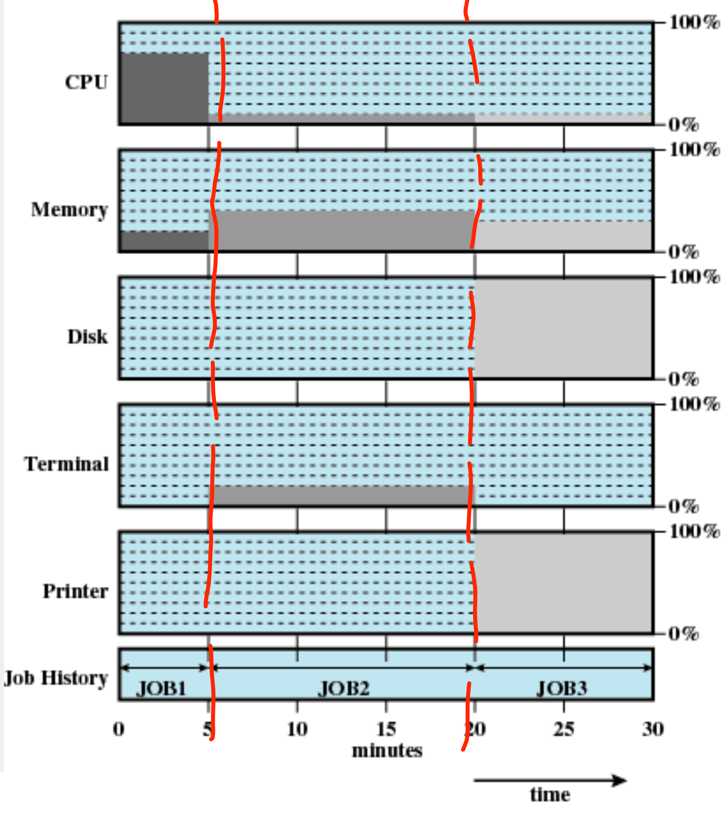
Some examples…

- Multiprogramming (multiple programs simultaneously loaded into the system)

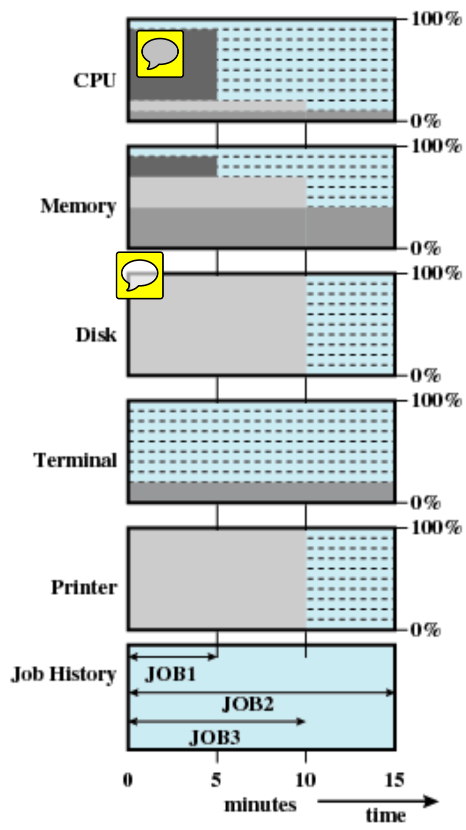


More examples

- Comparing the system utilizations

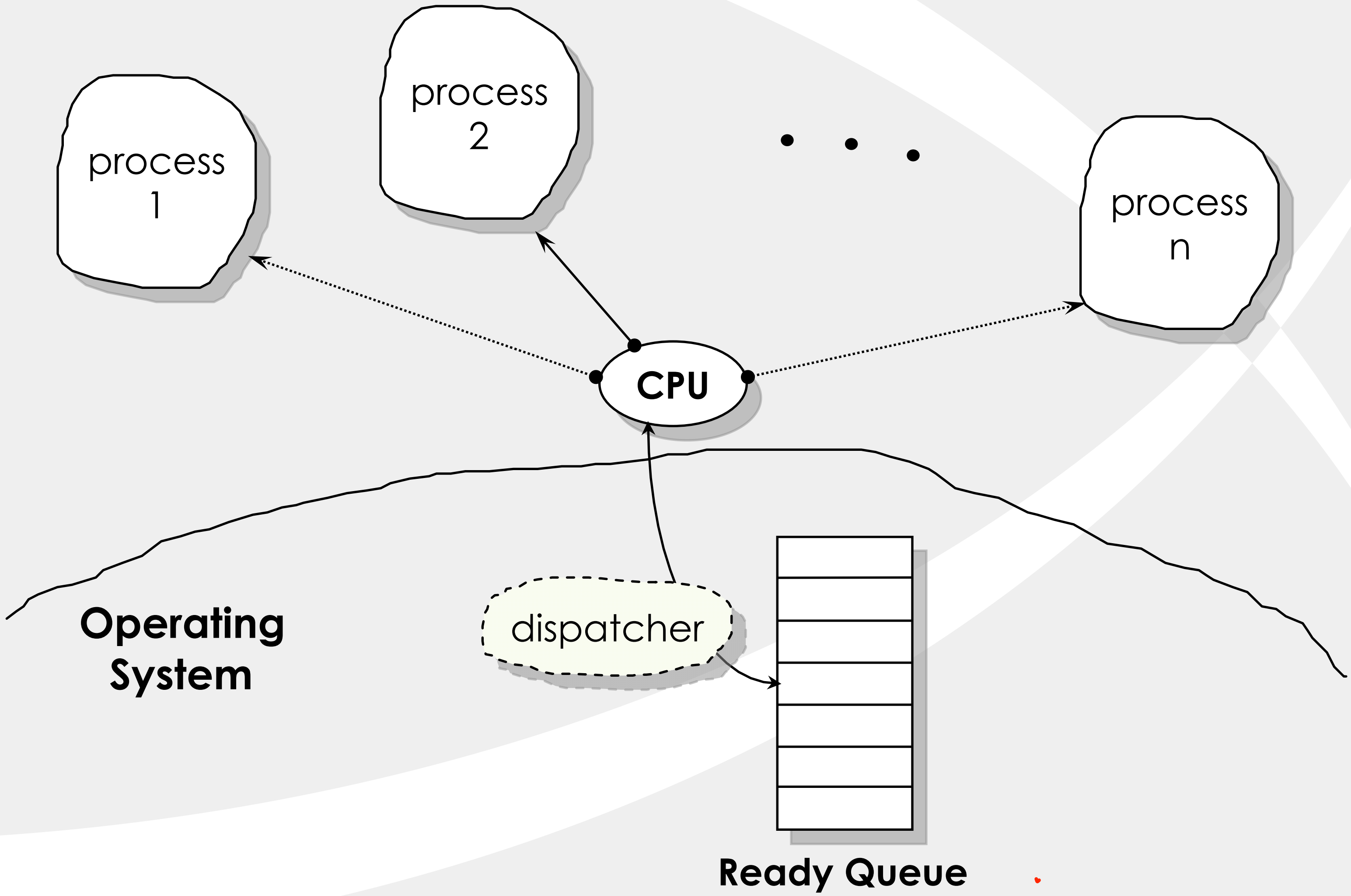


(a) Uniprogramming



(b) Multiprogramming

Process scheduling



Process scheduling objectives

- Fairness
  - Give equal and fair access to resources
- Differential responsiveness
  - Discriminate among different classes of jobs
- Efficiency
  - Maximize throughput, minimize response time, and accommodate as many users as possible

Why memory management?

- For multiprogramming:
  - multiple programs should co-exist in the memory
  - If programs don't co-exist (share the memory space) what will happen?
- With memory space sharing:
  - necessary to protect co-residing programs from each other
  - depending on the occupancy pattern, a program would not know the location until load time

Why is memory management?

- It provides:
  - Protection
  - Relocation

How does OS manage memory?

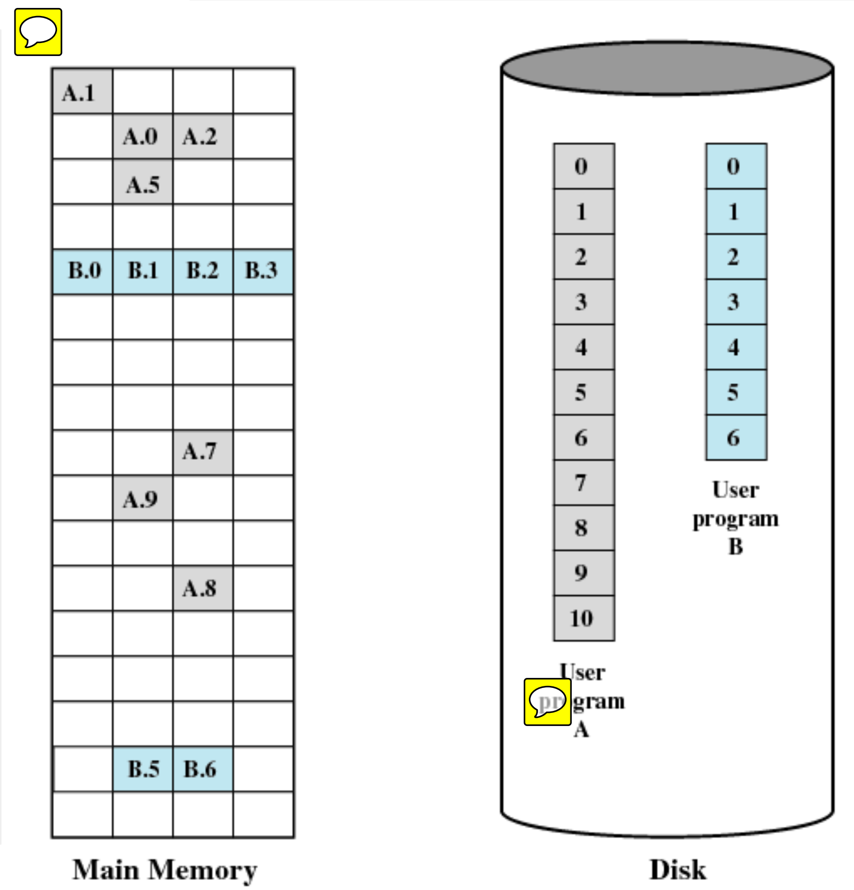
- Idea:
  - Divide the memory into partitions and allocate them to different processes
  - Partitions can be fixed size or variable size
- A process
  - Cannot access memory not allocated to it
  - Can request more memory from OS
  - Can release already held memory to OS
  - May only use a small portion of the allocated memory

How does OS manage memory?

- We can load more processes than that fitting the memory
- Where do the rest go? Answer: Virtual memory

Virtual memory in action

- Some memory chunks (pages) are in memory
- Rest in disk
- OS is responsible for retrieving the pages from and flushing the pages to disk



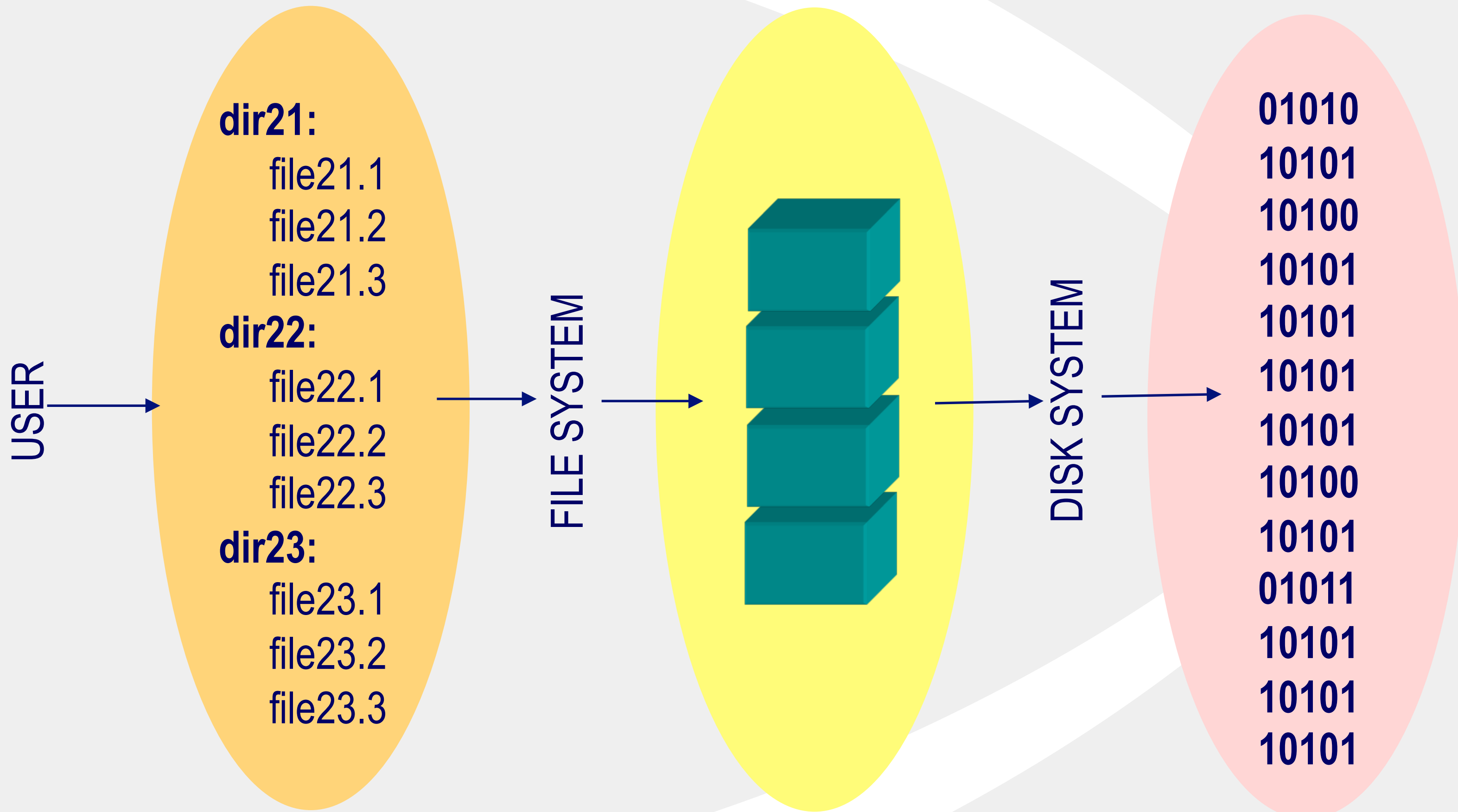
Main memory consists of a number of fixed-length frames, each equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.

Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages for all programs plus the operating system are on disk, as are files.

Storage management: overview

- Requirement:
  - Need to store data in persistent store
  - Organize the data such that it is easy to access and is protected
- What are the challenges?
  - Performance: secondary storage such as disks are very slow
  - Handle heterogeneity
  - Protect devices

Storage management: Data view

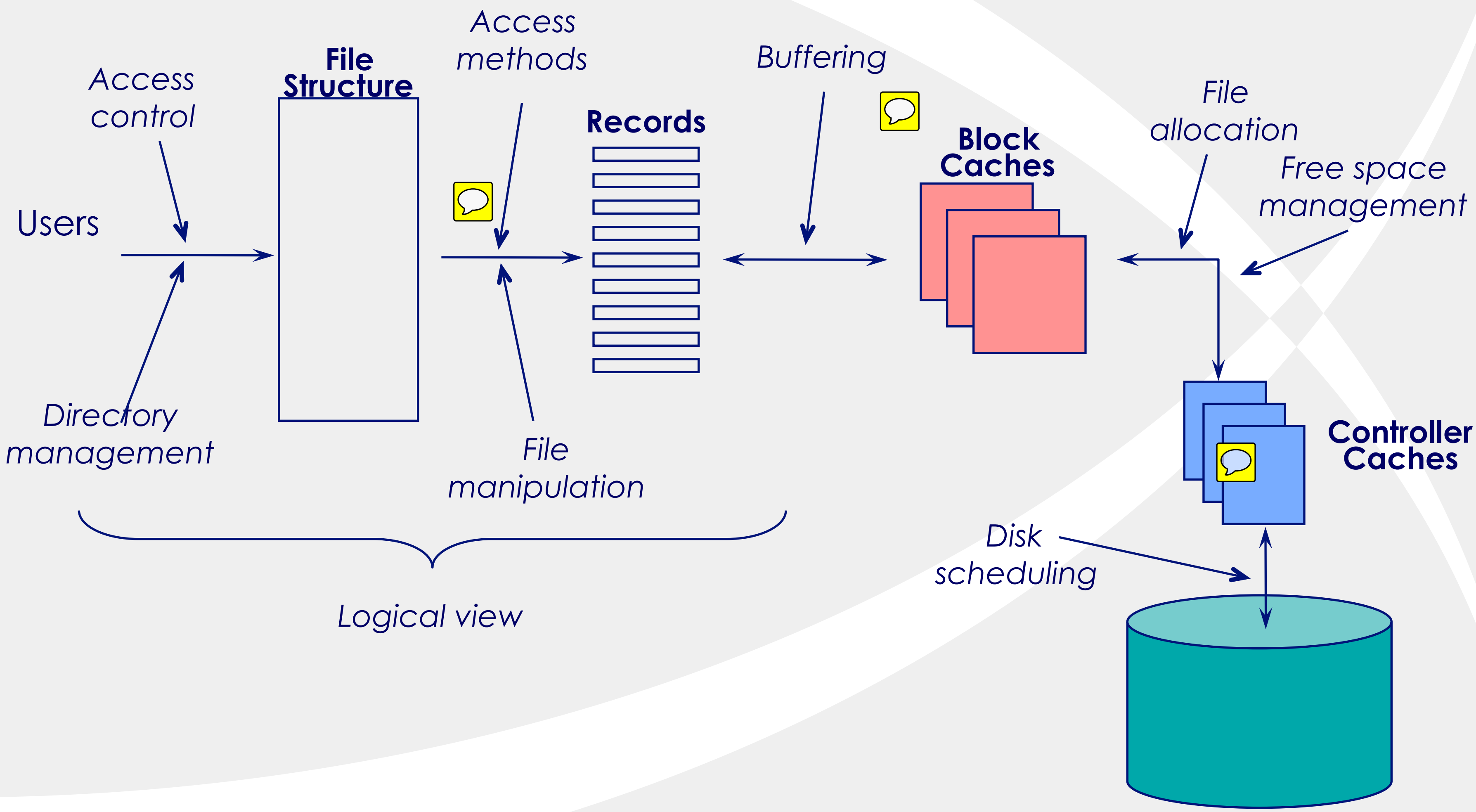


File system based view – content visible as files and directories

Block based view content considered as blocks of data

Physical view – all data just a stream of bits

Storage mgmt.: Component view



Abstractions provided by the OS

- Central to the abstractions provided by OS is
  - Processes or
  - Threads (lightweight processes)
- What are the functions provided for process?
  - Lifecycle management
  - Resource management
  - Inter-process communication

What is inter-process comm.?

- Processes:
  - Compete with one another for resources
  - Sometimes they also cooperate to get an application done! Can you give an example?
- Inter-process communication:
  - Key for cooperating processes that depend on each other

How to do inter-process comm.?

- We can use two approaches:
  - Shared memory approach:
  - Message passing approach

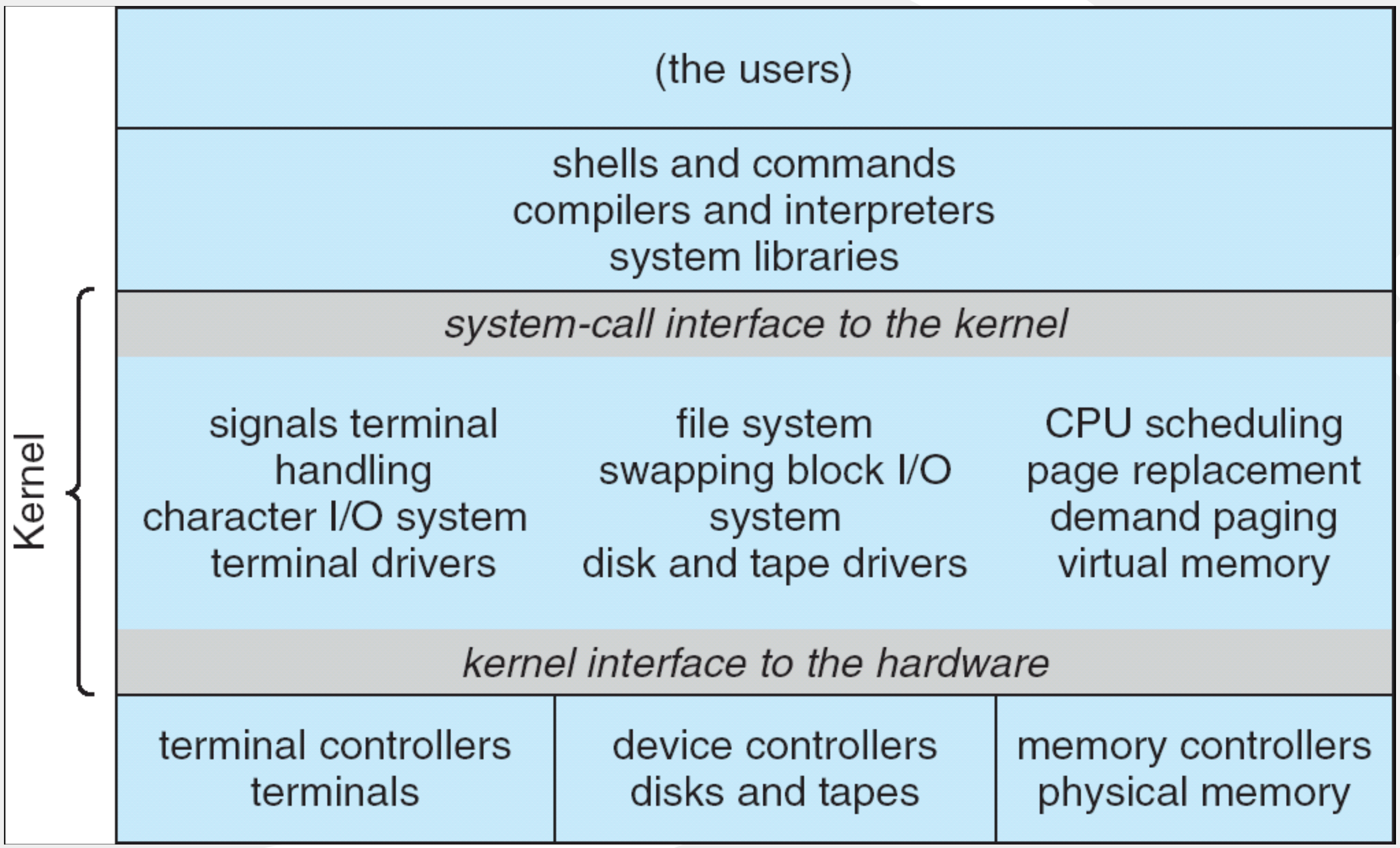
What is an OS architecture?

- How do we build an OS that has all the above functions?
  - We need an architecture for the OS

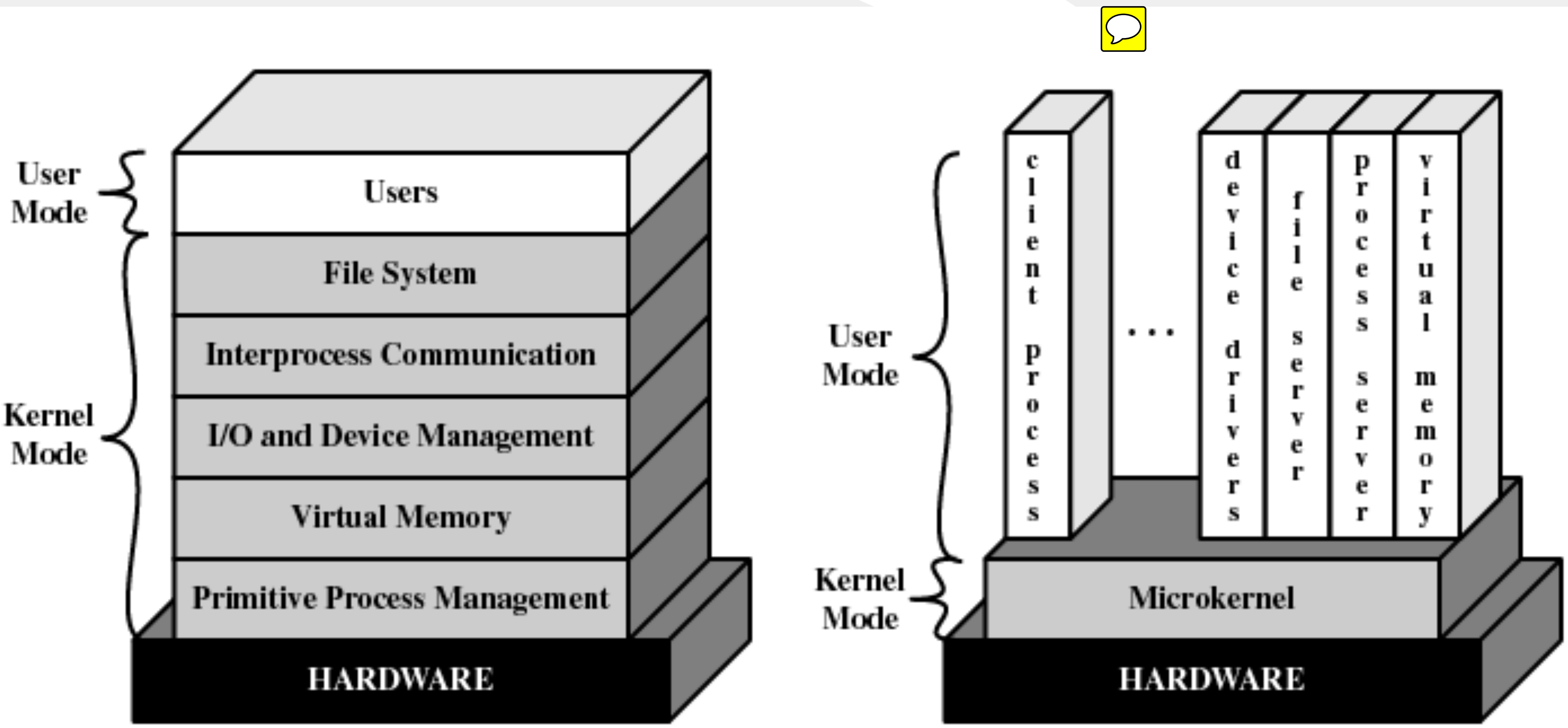
UNIX

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring.
  - Systems programs
  - Kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

UNIX System Structure



Layered Vs Microkernel



(a) Layered kernel

(b) Microkernel

With and without virtual machines