ECE437/CS481

M06C: VIRTUAL FILE SYSTEM & NETWORK FILE SYSTEM

CHAPTER 12.8

Xiang Sun

The University of New Mexico

Virtual File System

❑ Lots and lots of file system types

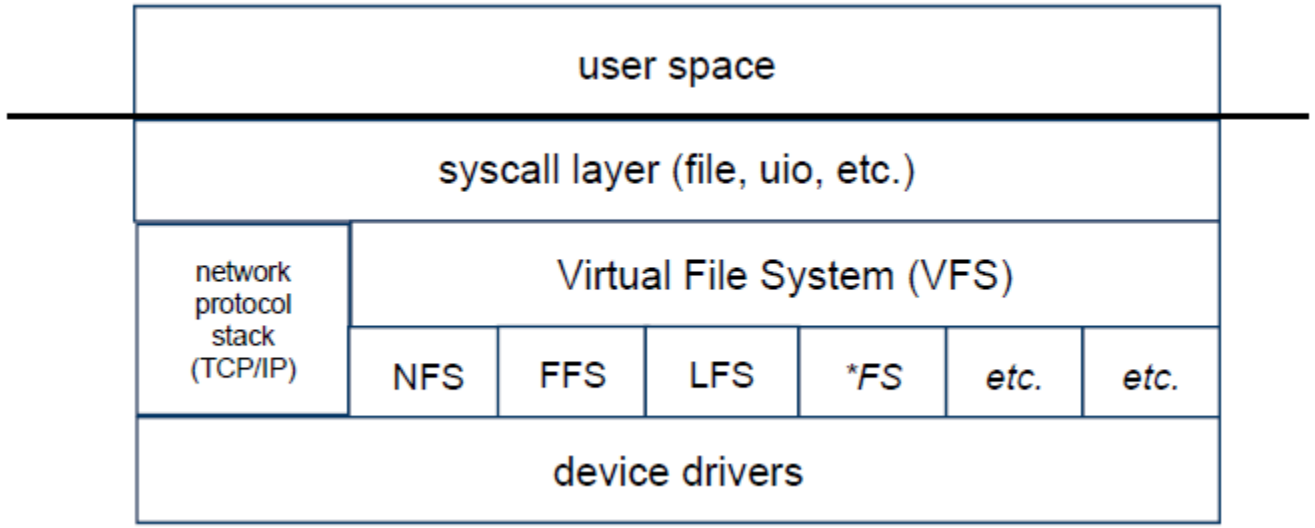
- Different file systems may have different directory structures, block sizes, etc.
- Is it possible to support more than one file system types on a single OS?

File System	Windows XP	Windows 7/8/10	macOS (10.6.4 and earlier)	macOS (10.6.5 and later)	Ubuntu Linux	Playstation 4	Xbox 360/One
NTFS	Yes	Yes	Read Only	Read Only	Yes	No	No/Yes
FAT32	Yes	Yes	Yes	Yes	Yes	Yes	Yes/Yes
exFAT	Yes	Yes	No	Yes	Yes (with ExFAT packages)	Yes (with MBR, not GUID)	No/Yes
HFS+	No	(read-only with Boot Camp)	Yes	Yes	Yes	No	Yes
APFS	No	No	No	Yes (macOS 10.13 or greater)	No	No	No
EXT 2, 3, 4	No	Yes (with third-party software)	No	No	Yes	No	Yes

Virtual File System

❑ Virtual file system

- A kernel software layer that handles all system calls related to file systems. Its main strength is providing a **common interface** to file systems.



- ✓ Processes make system calls to VFS.
- ✓ VFS separates file-system generic operations from implementation details, and then dispatches operations to appropriate file system implementation routines.

Virtual File System

- VFS system calls (~100)
  - filesystem operations – mounting, info, chroot, pivot_root
  - directory operations – chdir, getcwd, link, unlink, rename, symlink
  - file operations – open/close, read/write, stat, permissions, seek
    - chmod, chown, stat, creat, umask, dup, fcntl, truncate
    - read/write, readv/writev, pread/pwrite
  - memory mapping files – mmap, munmap, madvise, mlock
  - wait for input – poll, select
  - flushing – synch, fsync, msync, fdatasync
  - file locking – flock

© by Dr. X. Sun

Network File System

- Purpose: access disks attached to other computers

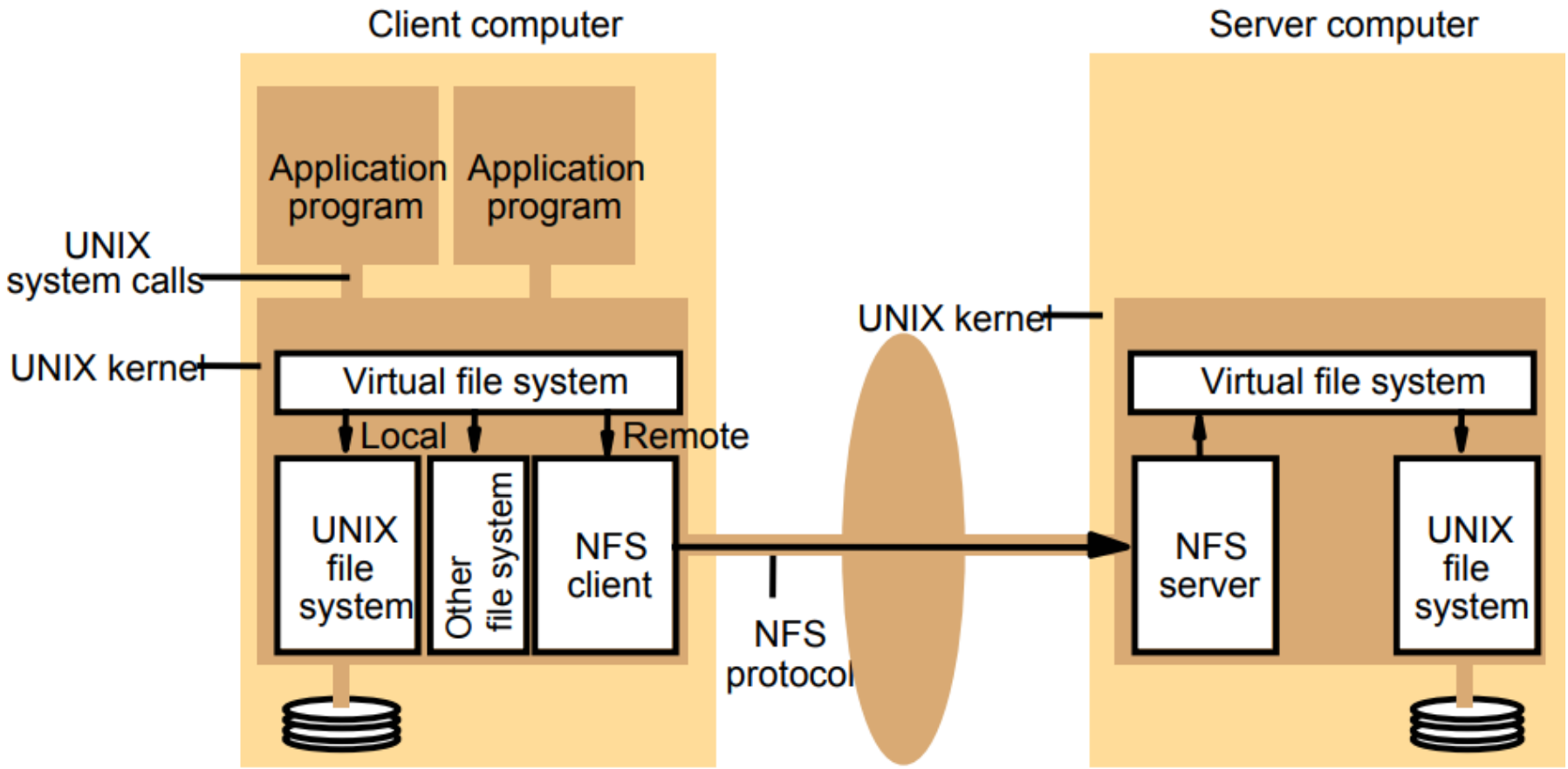
- NFS is a protocol for remote access to a file system
  - Does not implement a file system per se
  - NFS is a network protocol over TCP/UDP
    - ✓ Original implementations use UDP a transport layer protocol for low overhead.
    - ✓ Some newer implementations use TCP as a transport layer protocol for reliable communications.
  - Remote access is transparent to applications

- File system, OS, and architecture independent

- Implemented as a client/server architecture
  - Local file system (i.e., client) requests are forwarded to a remote server
  - Requests are implemented as remote procedure calls (RPCs)

© by Dr. X. Sun

Network File System

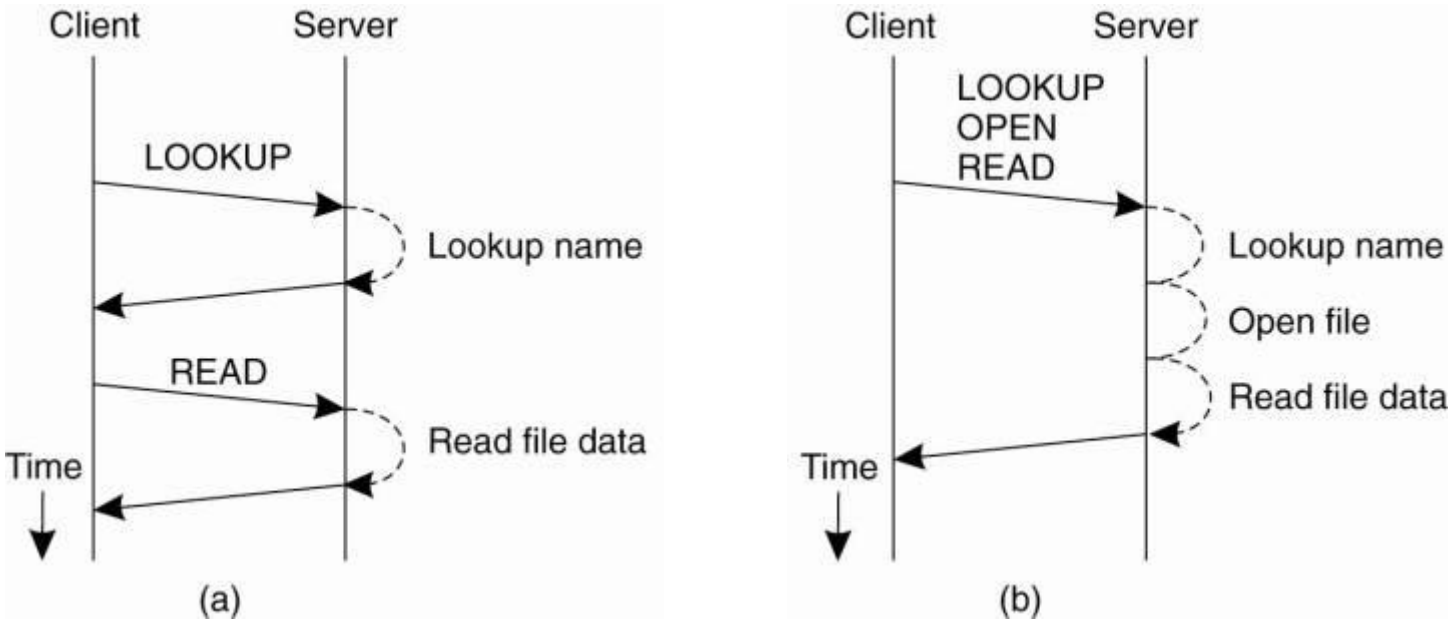


- NFS server check identity (e.g., uid, gid) for each access.
- NFS server is stateless, doesn't keep open files for clients
  - ✓ In NFS v4, stateful server are used.

Network File System

❑ Communications in NFS

- ✓ Every NFS operation (e.g., open(), read(), write()) can be implemented as a single remote procedure call (RPC) to a file server.
- ✓ In order to reduce the communications overhead (i.e., reduce the number of RPCs), compound procedures is applied (in NFS v4), where several RPCs can be grouped into a single request.

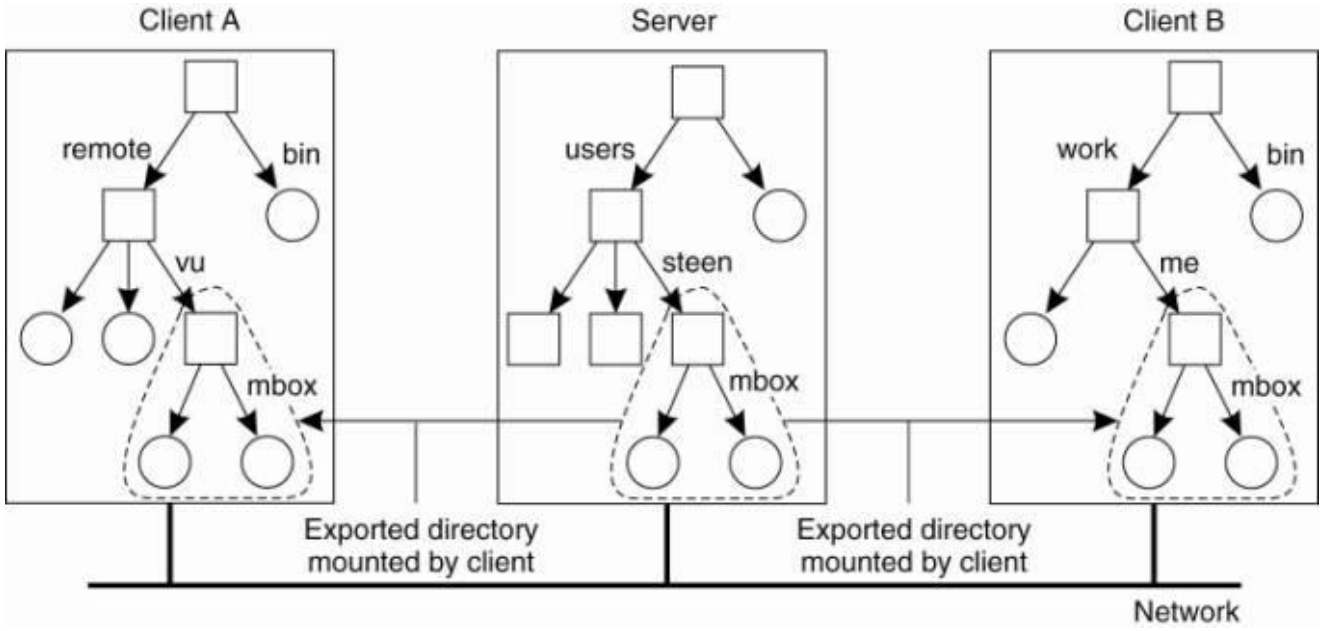


Network File System

❑ Naming in NFS

- The fundamental idea underlying the NFS naming model is to provide clients complete transparent access to a remote file system as maintained by a server. This transparency is achieved by letting a client be able to mount (part of) a remote file system into its own local file system.

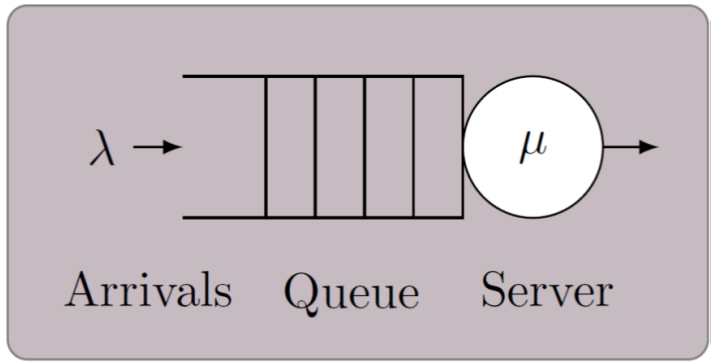
- A server exports a directory to a client, and the client mount the exported directory to its local name space.



Network File System

Synchronization in NFS

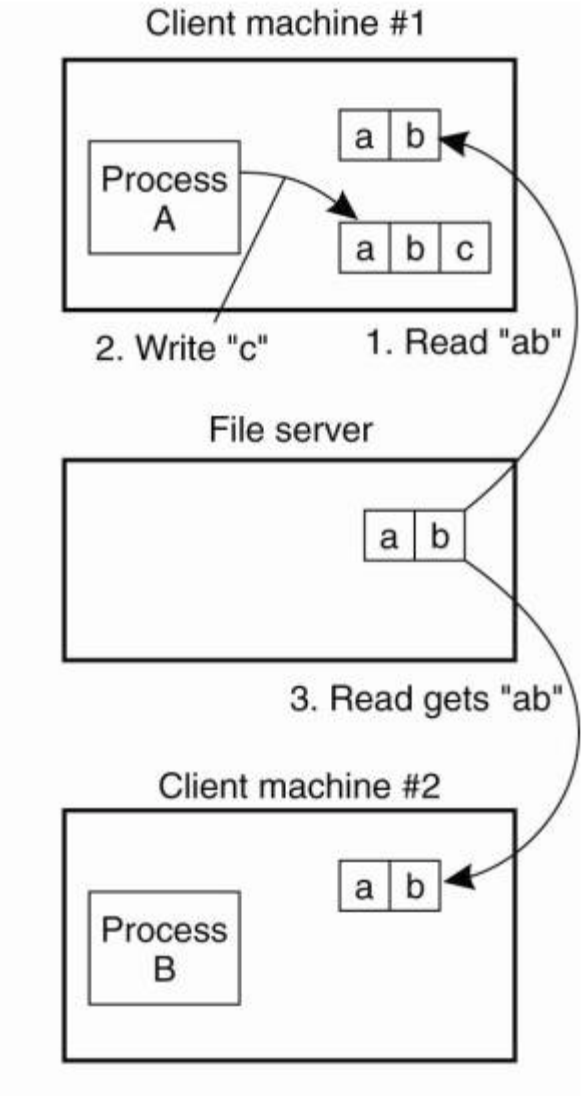
- Different clients mount the same part of file system in the server → files are shared among the clients.
- How to achieve synchronization if many clients try to read/write the same file?
- If every read/write go directly to the file server, the server can easily achieve the synchronization by processing them strictly sequentially.



- However, enabling the server to process every read and write may degrade the performance (in terms of throughput) of NFS, involving too many RPCs.

Network File System

- Synchronization in NFS
  - In order to improve the performance, clients are allowed to maintain local copies of heavily-used files in their private local caches. Thus, clients can read/write their local copies, and flush back the cached data to server when files are closed.
  - If a client locally modifies a cached file and shortly thereafter another client reads the file from the server, the second client will get an obsolete file.
  - There is no way to guarantee clients can read the up-to-date file in NFS.



ECE437/CS481

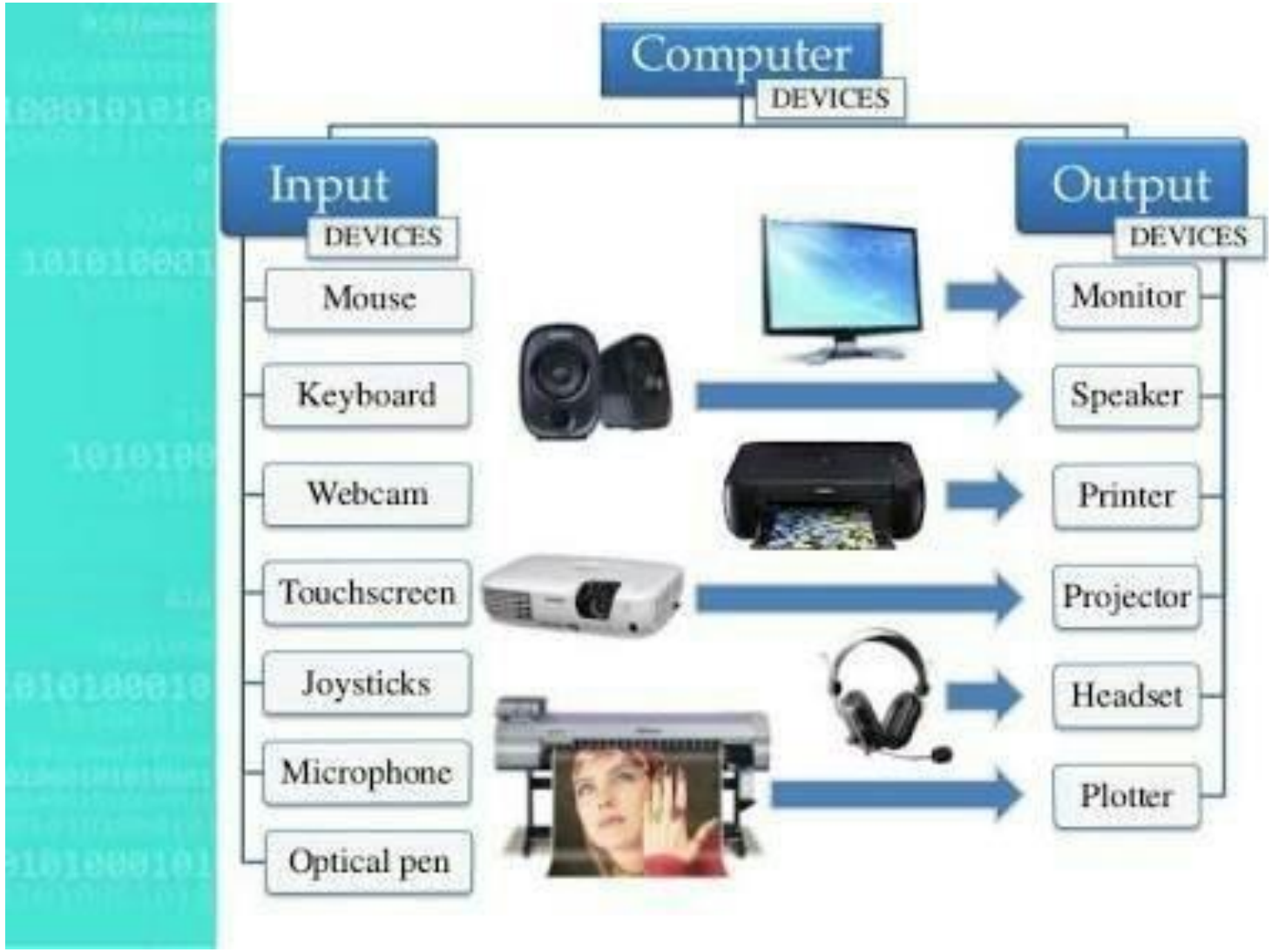
M06D: I/O SYSTEMS

CHAPTER 13

Xiang Sun

The University of New Mexico

Input and Output



Input and Output

- A computer's job is to process data
  - Computation (CPU, cache, and memory)
  - Move data into and out of a system (among I/O devices, memory and CPU)

- Challenges with I/O devices
  - Different categories: storage, networking, displays, etc.
  - Large number of device drivers to support
  - Device drivers run in kernel mode and can crash systems

- Goals of the OS
  - Provide a generic, consistent, convenient and reliable way to access I/O devices
  - As device-independent as possible
  - High performance I/O

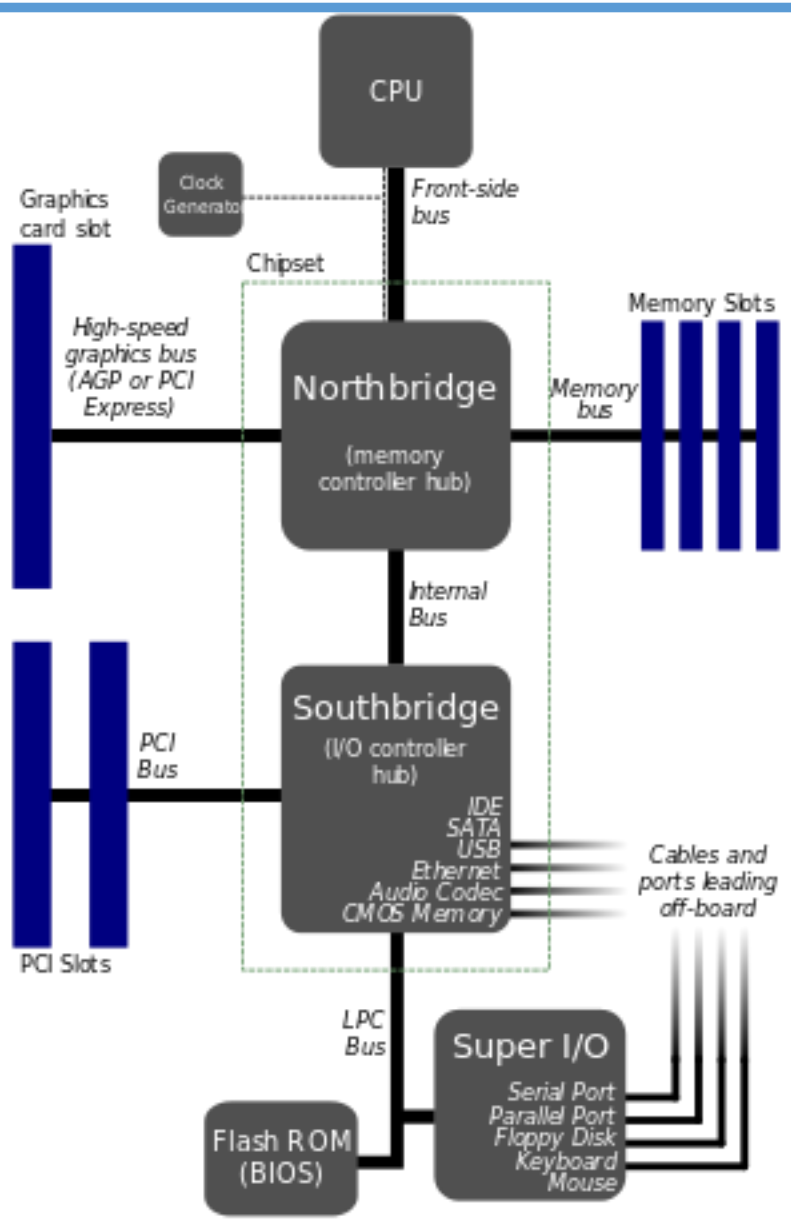
© by Dr. X. Sun

I/O System

❑ Hardware for data transferring

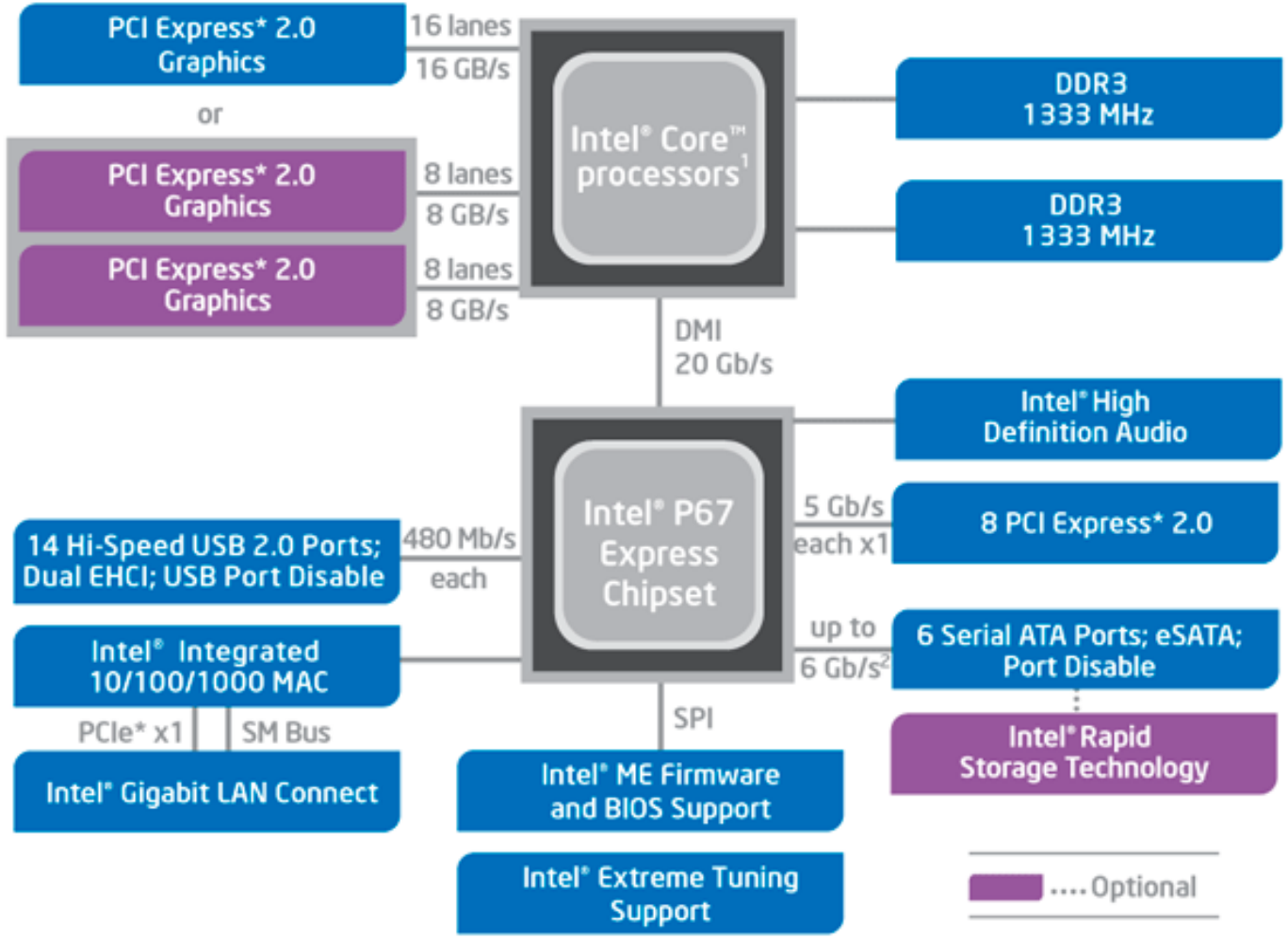
- **Northbridge**
  - ✓ Memory control hub
  - ✓ Links the CPU to very high-speed devices, e.g., memory and graphic card
  - ✓ Obsolete in modern computer architecture.

- **Southbridge**
  - ✓ I/O control hub
  - ✓ Connects to low-speed I/O devices, e.g., hard drives, USB devices, etc.



I/O System

- Communications between CPU and I/O devices
  - PCIe slot(s) for the GPU(s) and the DIMM slots for the RAM are now connected directly to the processor. The functionality previously provided by the Northbridge has now been integrated into the CPU.
  - Southbridge is referred to as the chipset and still handles the functionality that was handled by the Southbridge. However, it has better throughput than what the South Bridge used to have.

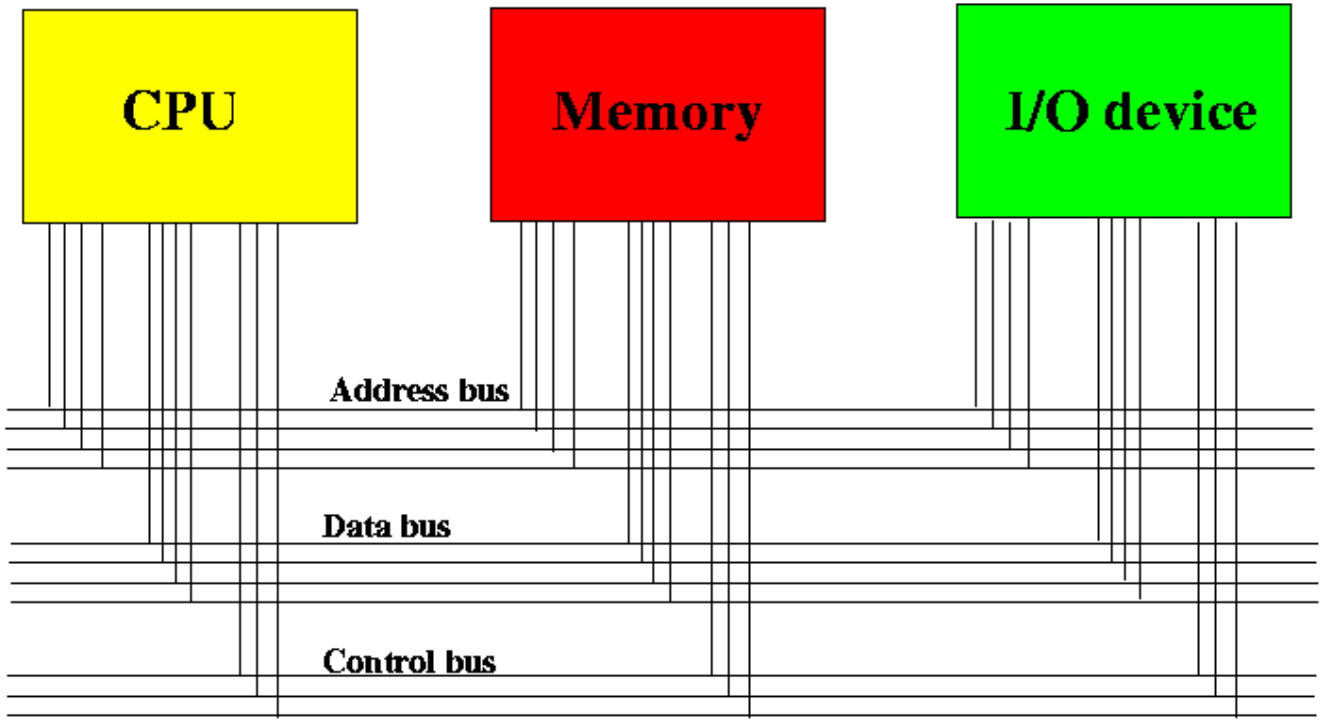


[1] Compatible with 2nd generation Intel® Core™ processor family

[2] All SATA ports capable of 3 Gb/s. 2 ports capable of 6 Gb/s.

I/O System

- Communications between CPU and I/O devices
  - Three types of buses
    - ✓ Data bus: transfer the data (information) between the CPU and other devices.
    - ✓ Address bus: identify the address of data
    - ✓ Control bus: identify the action on the address and data busses.



© by Dr. X. Sun

I/O System

❑ Addressing I/O devices

- CPU controls IO devices
  - ✓ by writing some command code into the command registers.
  - ✓ by reading their status registers to obtain the status (e.g., idle, busy) of I/O devices

```
                    +-------------+          +-------+
                    | Device controller |          |       |
+-----+             |                   |<-------->| Device |
|     |-------->| Control register |          |       |
| CPU |<--------| Status register  |          |       |
|     |<-------->| Data register    |          |       |
+-----+             |                   |          |       |
                    +-------------+          +-------+
```

- I/O devices need to be assigned a unique ID or address, just like memory.

- There are 2 techniques to allow IO devices to be addressed:
  - ✓ Standard I/O
  - ✓ Memory-mapped I/O

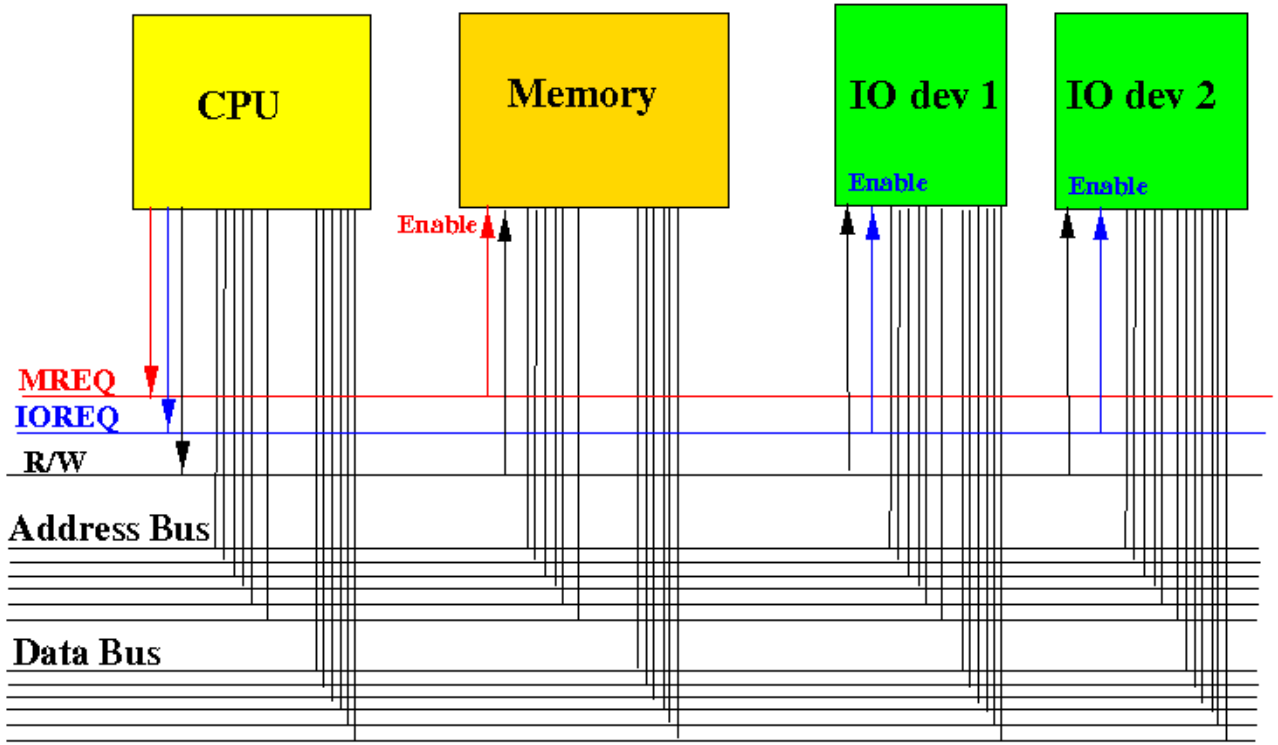
Main memory

Address	Value
0x00	01001010
0x01	10111010
0x02	01011111
0x03	00100100
0x04	01000100
0x05	10100000
0x06	01110100
0x07	01101111
0x08	10111011
...	...
0xFE	11011110
0xFF	10111011

© by Dr. X. Sun

❑ Addressing I/O devices—Standard I/O

- In the standard IO technique, IO devices and memory location have separate address spaces
  - ✓ Two "spaces" are separate if there is an external (explicit) signal that identify the space.
  - ✓ For example,
    - the MREQ signal is used to signal memory requests. When the CPU wants to access (read/write) a memory location, it asserts the MREQ signal.
    - the IOREQ signal is used to signal IO requests. When the CPU wants to access (read/write) an IO device, it asserts the IOREQ signal.



- **MOV assembler instruction** to access the memory - hence, when the CPU executes a load or store instruction, it will assert the **MREQ signal**.
- **IN** (input) and **OUT** (output) **assembler instruction** to access IO devices - hence, when the CPU executes a input or output instruction, it will assert the **IOREQ signal**.

© by Dr. X. Sun

I/O System

❑ Addressing I/O devices—Memory-mapped I/O

- IO devices and the main memory share the same address space.
  - ✓ One set of memory addresses is for main memory operations
  - ✓ The other set of memory address is for I/O operations.
  - ✓ The 2 sets of addresses must be disjoint (no overlap) !!!
  - ✓ For example, last 4 bits of the address = 1111, then the CPU wants to address an IO device. Otherwise, the CPU wants to address the memory

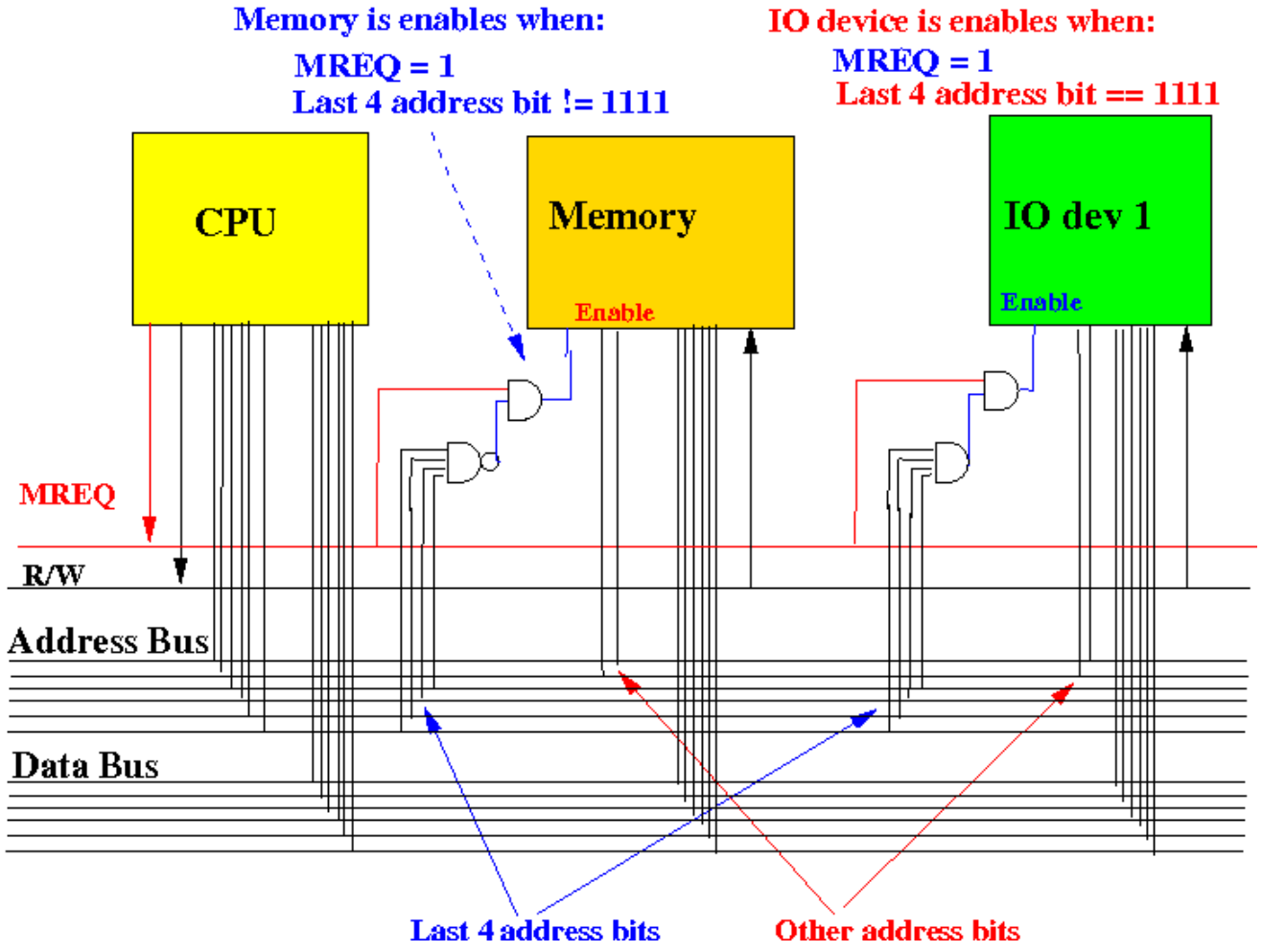
- Three types of addresses are provided:
  - ✓ Read only address
  - ✓ Write only address
  - ✓ Read and write address

Address (binary)	Use for
00000000 00000000 00000000 00000000	Memory
00000000 00000000 00000000 00000001	Memory
00000000 00000000 00000000 00000011	Memory
....	
11100000 00000000 00000000 00000000	Memory
11100000 00000000 00000000 00000000	Memory
....	
11101111 11111111 11111111 11111111	Memory
11110000 00000000 00000000 00000000	IO device
11110000 00000000 00000000 00000001	IO device
....	
11111111 11111111 11111111 11111111	IO device

© by Dr. X. Sun

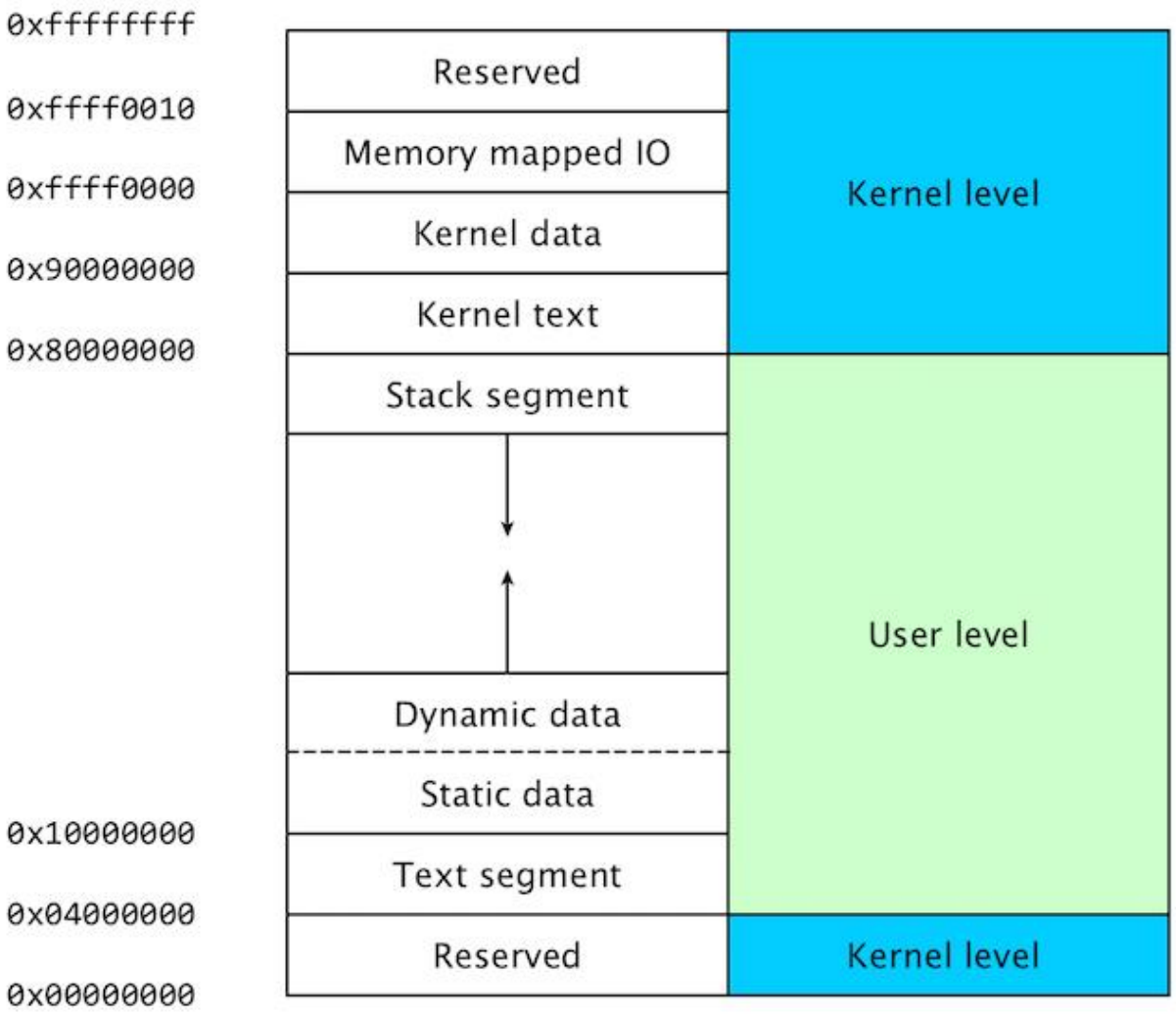
❑ Addressing I/O devices—Memory-mapped I/O

- The following circuit can determine whether the CPU wants to address the memory or an I/O device.
- When the CPU sends out an address, there is no way to tell by only looking at all the signals from the CPU if the address was for a memory location or for an IO device.
- That is, there is no need to have 2 different types of assembler instructions (one type for accessing memory and the other for accessing IO device).



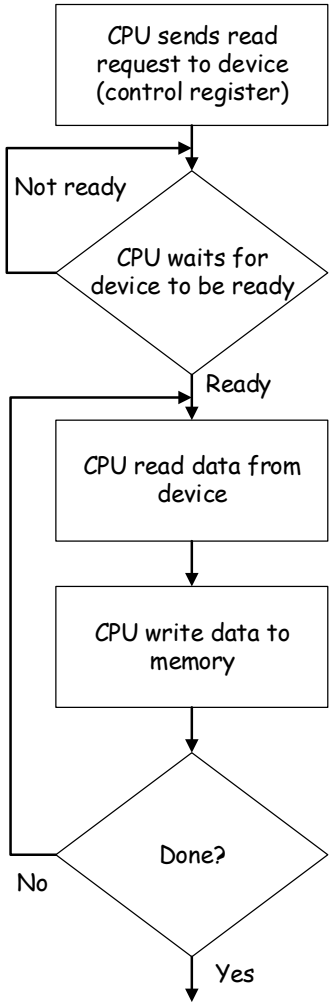
I/O System

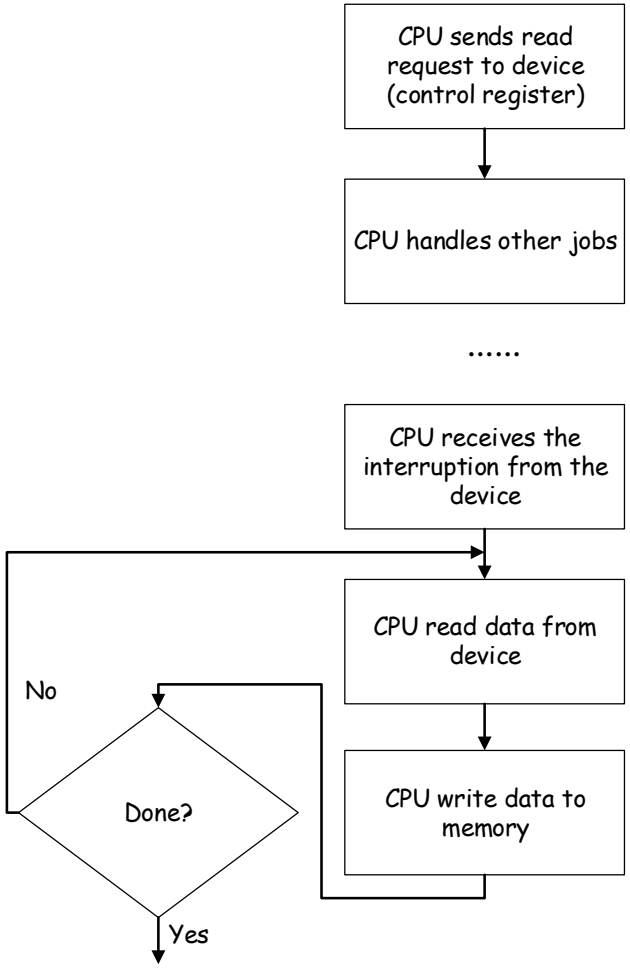
- Addressing I/O devices—Memory-mapped I/O

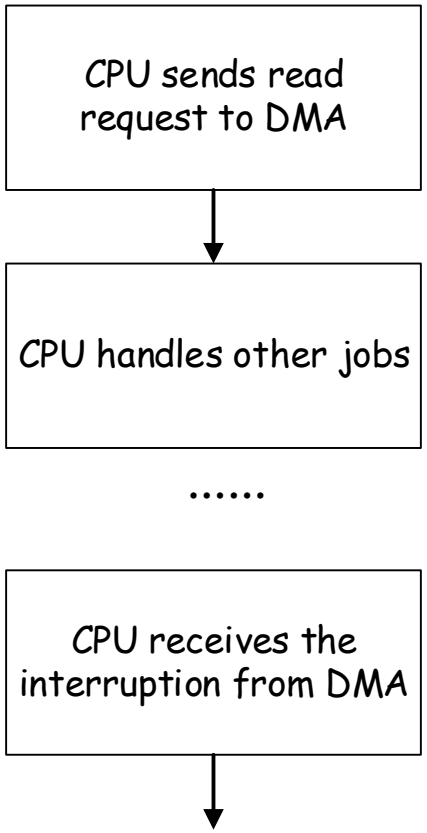


I/O System

- Data transferring between CPU and I/O devices
  - Recall: three methods of achieving data transferring:
    - ✓ Programmed I/O.
    - ✓ Interrupt- initiated I/O.
    - ✓ Direct memory access (DMA).

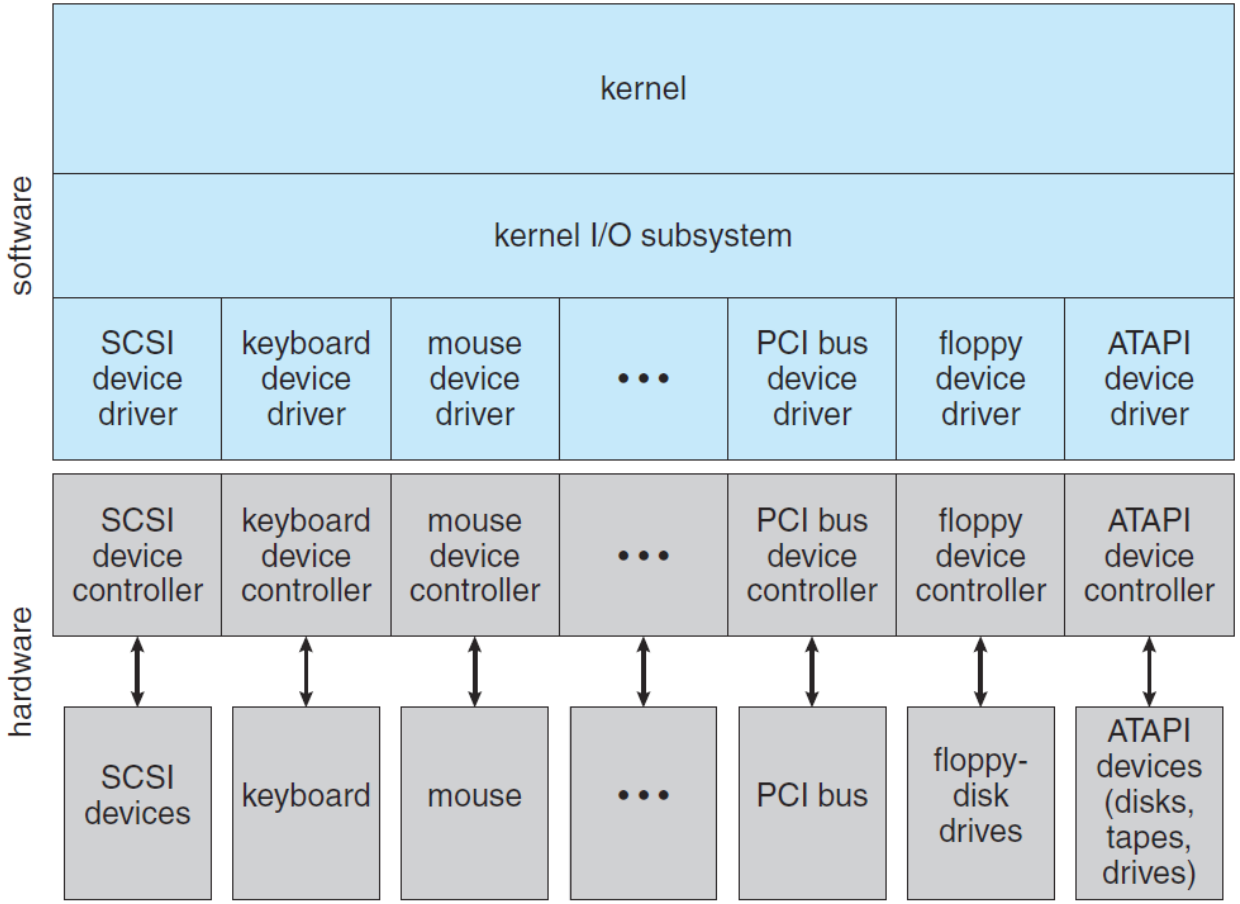






I/O System

- Hierarchical architecture of the I/O system



I/O System

Device controller—OS/user view

- Control register
- Data register
- Status register

Device driver

- Each type of device controller should have a specific device driver designed for it.
- A device driver can provide functionality through two separate driver components:
  - ✓ The upper-half part: interact with application and operating system—hardware independent.
  - ✓ The lower-half part: Provide the interface with device controller by accessing various registers.

Upper-half
- Device initialization
- Accept read-write request from the OS
- Check input parameters

Lower-half
- Start the device if necessary (e.g., start spinning the CD-ROM)
- Check if device is available: if not, wait
- Issue command(s)
- Wait for results
  - Busy wait (awakened by interrupt)
  - Block
- Check for possible errors
- Return results

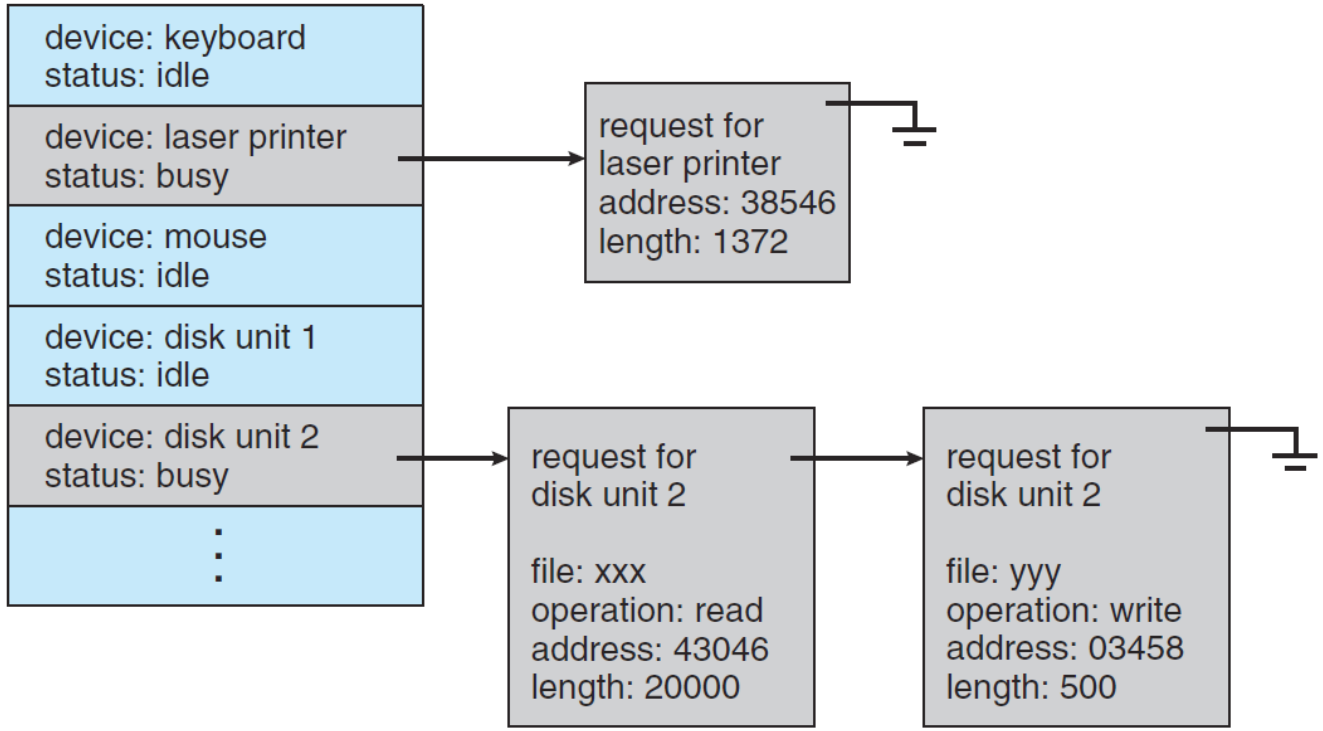
© by Dr. X. Sun

I/O System

Kernel I/O subsystem

1. I/O scheduling

- Kernel maintain a device-status table.
- If the device is busy with a request, the type of request and the related parameters will be stored in the table entry for that device.
- The OS may provide some scheduling algorithm to make IO operations more efficient.

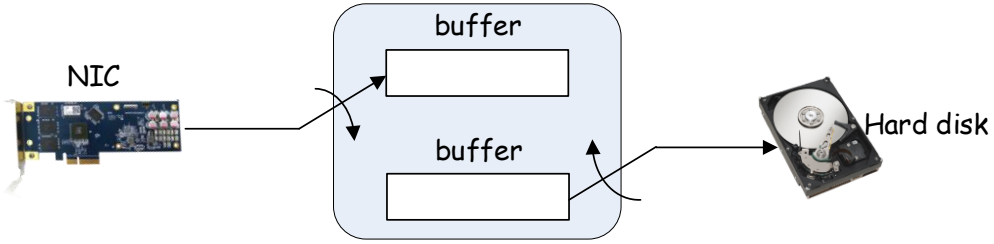


Kernel I/O subsystem

2. Buffering

- A buffer is a memory area that stores data being transferred between two devices or between a device and an application.

- The reasons of providing buffering includes:
  - ✓ cope with a speed mismatch between two devices. For example, a file received by the NIC (which has a lower speed) need to be stored on the hard disk (which has a higher speed).--- spooling



  - ✓ Provide adaptations for devices that have different data-transfer sizes. For example, in order to transmit a video from a disk to anther host via the NIC. The video has to be fragmented into small network packages.

Kernel I/O subsystem

3. Caching

- A cache is a memory area that stores copies of popular/recently accessed data. Access to the cached copy is more efficient than accessing to the original.
  - ✓ For instance, data stored in a disk could be cached in the memory (or even copied again in the CPU's secondary and primary caches) based on the hit rate.
  - ✓ A cache is different from a buffer: 1) different purposes; 2) a buffer may hold the only existing copy of a data; while a cache may have one of the copies of data.

4. Error handling

- I/O transferring can fail in many ways.
  - ✓ Transient failure
  - ✓ Permanent failure
- The kernel I/O subsystem can 1) deal with transient failures by attempting the data re-transmission; 2) return information to help with failure analysis.

© by Dr. X. Sun

I/O System

Kernel I/O subsystem

5. I/O protection

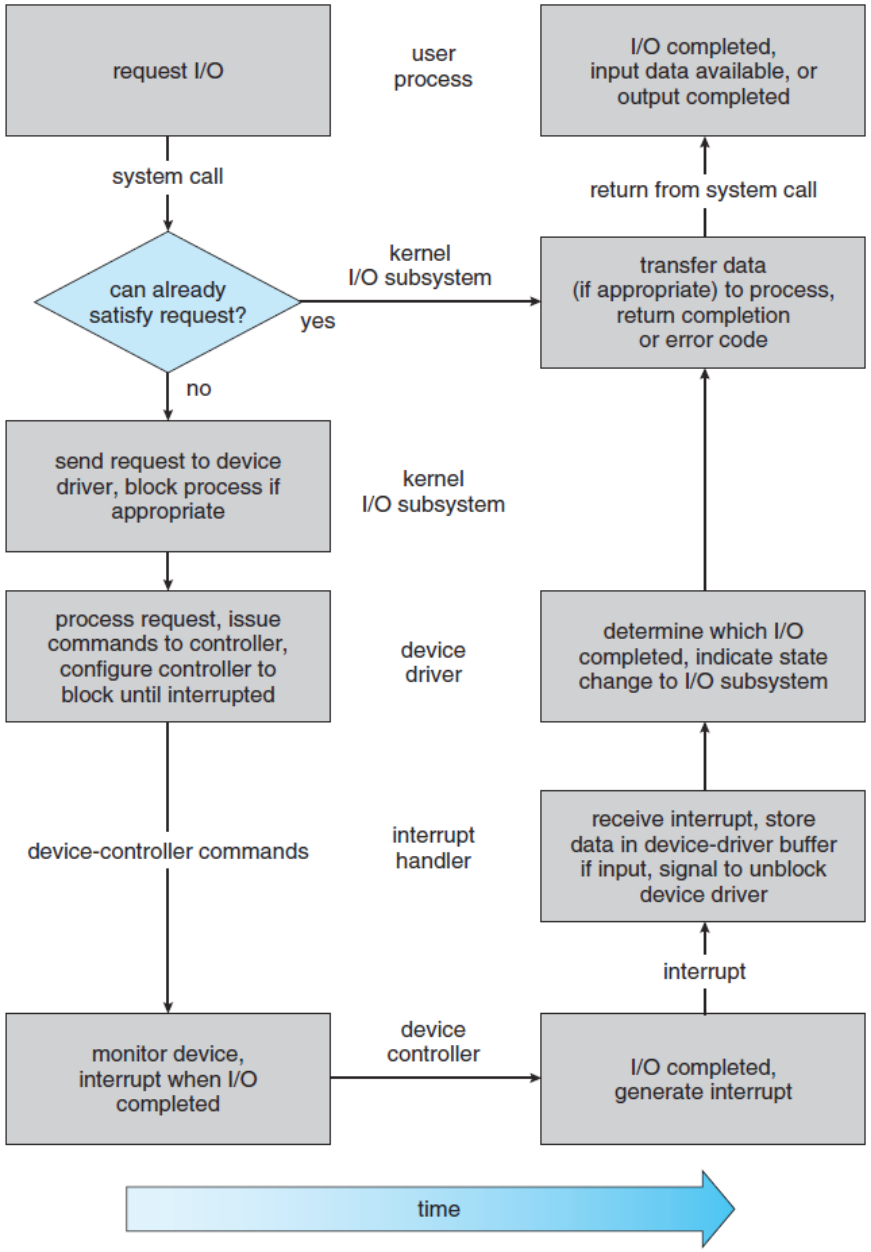
- Users cannot issue I/O instructions directly; they must do it through the operating system.
  - ✓ To conduct I/O operations, a user program executes a system call to request the OS to perform I/O on its behalf.
  - ✓ I/O subsystem in the OS validate the request. If it is, it process the request.
- Memory-mapped I/O locations must be protected from user access.

© by Dr. X. Sun

I/O System

❑ The life cycle of an I/O request

- ✓ A process issues a read() request, the kernel I/O subsystem first if the data are already available in the memory.
- ✓ If not, the process is removed from the run queue and is placed on the wait queue. Then, the kernel I/O subsystem sends the request to the related device driver.
- ✓ The device driver allocates space to receive the data, and sends the commands to the device controller by writing into the control register of the device.
- ✓ The device controller operates the device hardware to perform the data transfer. <assume that DMA is applied>
- ✓ Once data transfer is completed, DMA controller generates an interrupt to the interrupt handler, which signal the device driver.
- ✓ The device driver signals the kernel I/O subsystem that the I/O request has been completed.
- ✓ The kernel transfers data or return codes to the address space of the requesting process and moves the process from the wait queue back to the ready queue.



© by Dr. X. Sun

ECE437/CS481

M06E: DISK SCHEDULING

CHAPTER 10.1-10.4

Xiang Sun

The University of New Mexico

I/O Scheduling for Block Devices

- Mass Storage structure
  - Magnetic disk provide bulk of secondary storage
  - Drivers rotate at 3,600 to 10,000 times per second

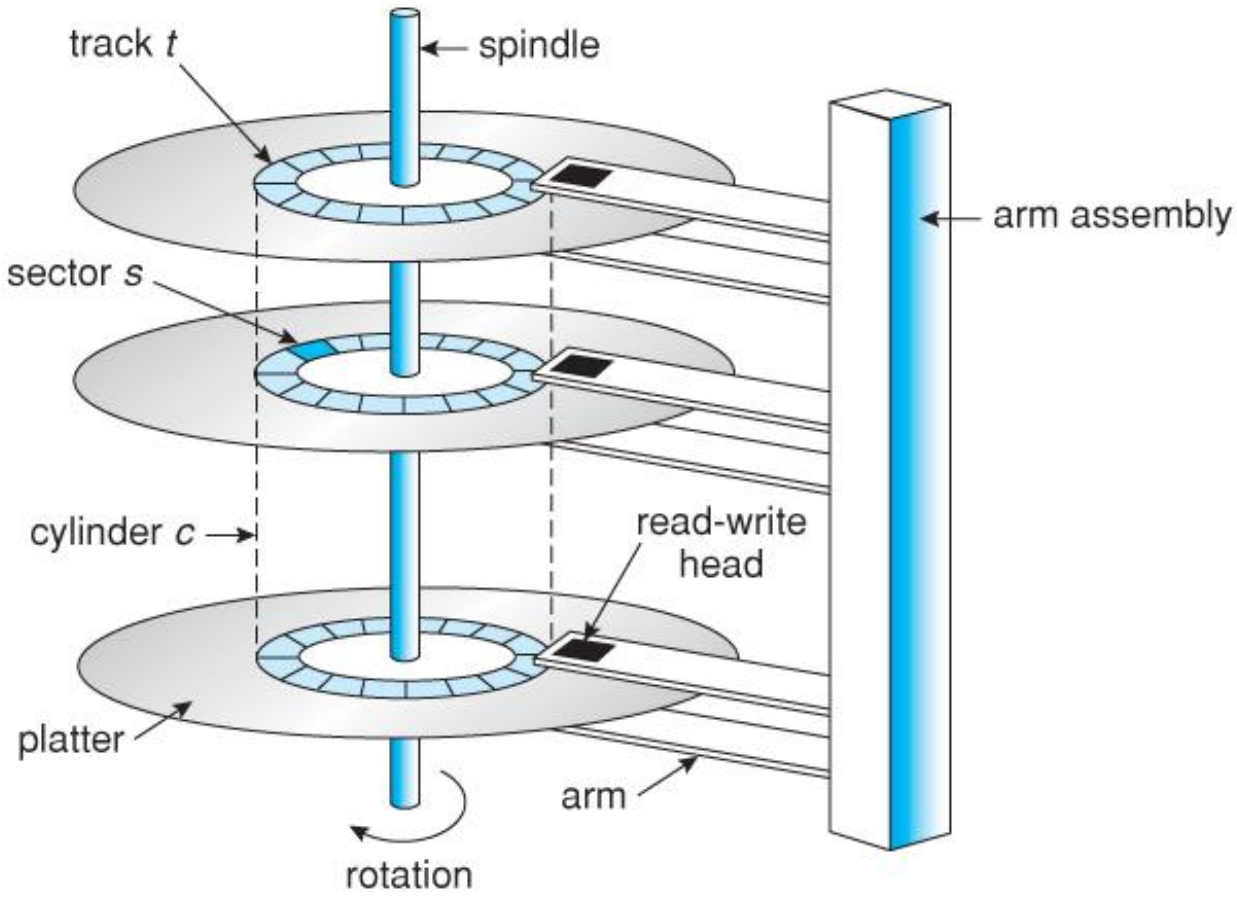
- Hard drive is attached to computer via I/O bus
  - ATA: Parallel ATA (PATA), Serial ATA (SATA)

  - SCSI (Small Computer System Interface)
  - USB

© by Dr. X. Sun

I/O Scheduling for Block Devices

❑ Recall Disk Structure



- **Sector**: Minimum unit (normally, 512 Byte for old hard disks; 4 KB for new hard disks) for data storage.
  - ✓ Use `cat /sys/block/sda/queue/physical_block_size` to see the size of sector for your hard disk.
- **Cluster**: Group of sectors
- **Track**: One cycle on a platter
- **Cylinder**: A semantic shape that consists of same level tracks on different platters.

© by Dr. X. Sun

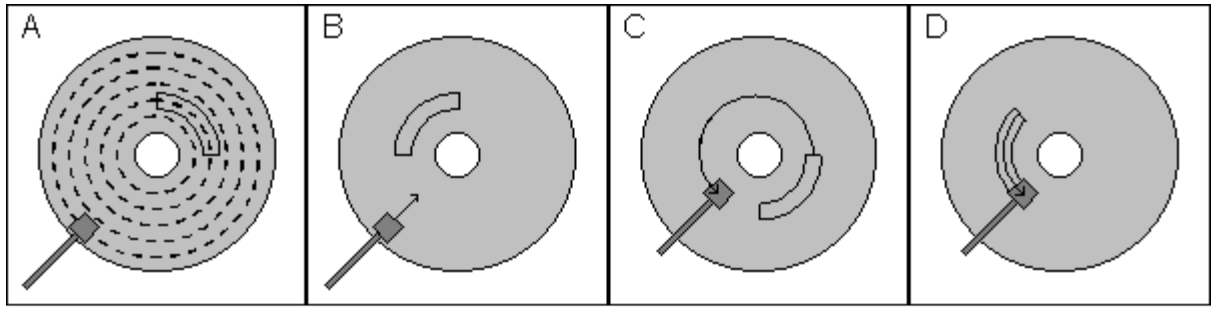
I/O Scheduling for Block Devices

❑ Response time

- ✓ Internal response time: the time for moving data between the disk surface and the controller on the drive.
- ✓ External response time : the time for moving data between the controller in the drive and the memory.

➢ Internal response time comprises

- ✓ Seek time: the time for moving disk arm to place the r/w head on the desired cylinder/track.
- ✓ Rotational latency: the time for desired sector to rotate under the disk head.
- ✓ Internal transfer time: the time for the disk platter to rotate until all the addressed sectors have passed under the head. The transfer time is directly proportional to the number of addressed sectors.
- ✓ Overhead: the time for searching the right platter, etc. The overhead is normally fixed.



Starting Position | Seek Time | Rotational Latency | Transfer Time

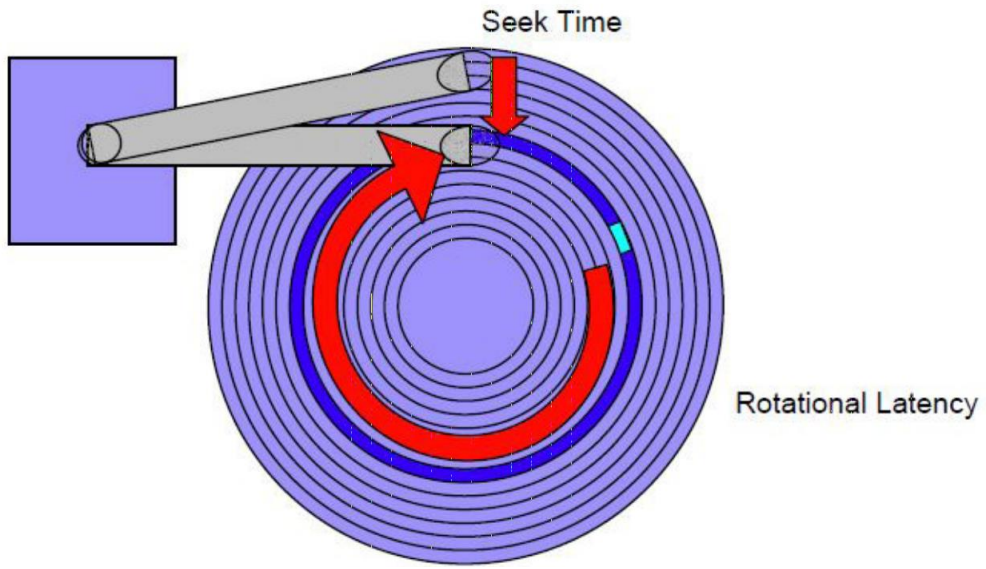
© by Dr. X. Sun

I/O Scheduling for Block Devices

❑ Seek time

- Manufacturers often report *average seek time* of 3-9 ms
  - ✓ These times average the time to seek from any track to any other track.

- In practice, the average seek time is often much lower.
  - ✓ For example, if the head is already on or near the desired track, then seek time is much smaller. In other words, locality is important!
  - ✓ Actual average seek times are often just 2-3ms.



© by Dr. X. Sun

I/O Scheduling for Block Devices

❑ Rotational latency

- Once the head is in place, we need to wait until the right sector is underneath the head.
  - ✓ This may require as little as no time (reading consecutive sectors) or as much as a full rotation (just missed it).
  - ✓ On average, for random reads/writes, we can assume that the disk spins halfway.
- Rotational delay is determined by how fast the disk platters spin.
  - ✓ Average rotational delay = 0.5 rotation ÷ rotational speed
  - ✓ For example, a 5400 RPM disk has an average rotational delay of:

    0.5 rotation / (5400 rotations/minute) x 60 secs/1 min ≈ 5.55ms

© by Dr. X. Sun

I/O Scheduling for Block Devices

- Transfer time
  - Time to read the bits in the target sector.
  - Average transfer time per sector= 1/RPM x 1/(avg # sectors/track) x 60 secs/1 min

- The overall internal response time is the sum of average seek time, rotational delay, transfer time, and overhead.

- Assume a disk has the following specifications.
  - An average seek time of 9ms
  - A 5400 RPM rotational speed
  - Avg number of sectors/track= 400
  - 0.2 ms of overheads

- How long does it take to read 2 continuous sectors?
  - The average rotational delay is 5.55 ms.
  - The transfer time will be about 2x1/5400x1/400x60≈ 0.05ms.
  - The internal response time is then 9ms + 5.55ms + 0.05ms + 0.2ms = 14.8ms. That's 14,800,000 CPU cycles for a 1GHz processor!

© by Dr. X. Sun

I/O Scheduling for Block Devices

- Disk scheduling
  - Disks are several orders of magnitude slower than main memory
    - ✓ The performance of disk I/O is vital for system
    - ✓ Internal response time >> external response time for reading/writing a sector
    - ✓ How to reduce the internal response time?
      - Rotational latency
      - Internal transfer time
      - Overhead
      - Seek time
  - Disk scheduling, minimizing seek distance (seek time)
    - ✓ Possibly reorder stream of read/write requests to improve performance.

© by Dr. X. Sun

I/O Scheduling for Block Devices

- Algorithms to schedule disk I/O requests
  - FCFS
  - SSTF
    - ✓ (Shortest-service-time-first): pick the request that requires the least movement of the head
  - SCAN
    - ✓ (back and forth over disk): good service distribution
  - C-SCAN
    - ✓ (Circular Scan): one way with fast return
  - Look
  - C-Look

  - Illustrate with a platter containing 200 tracks (#0-#199) and a request queue that buffers 8 read/write requests on the #98, #183, #37, #122, #14, #124, #65, #67 track.
  - Current head location: #53 track.

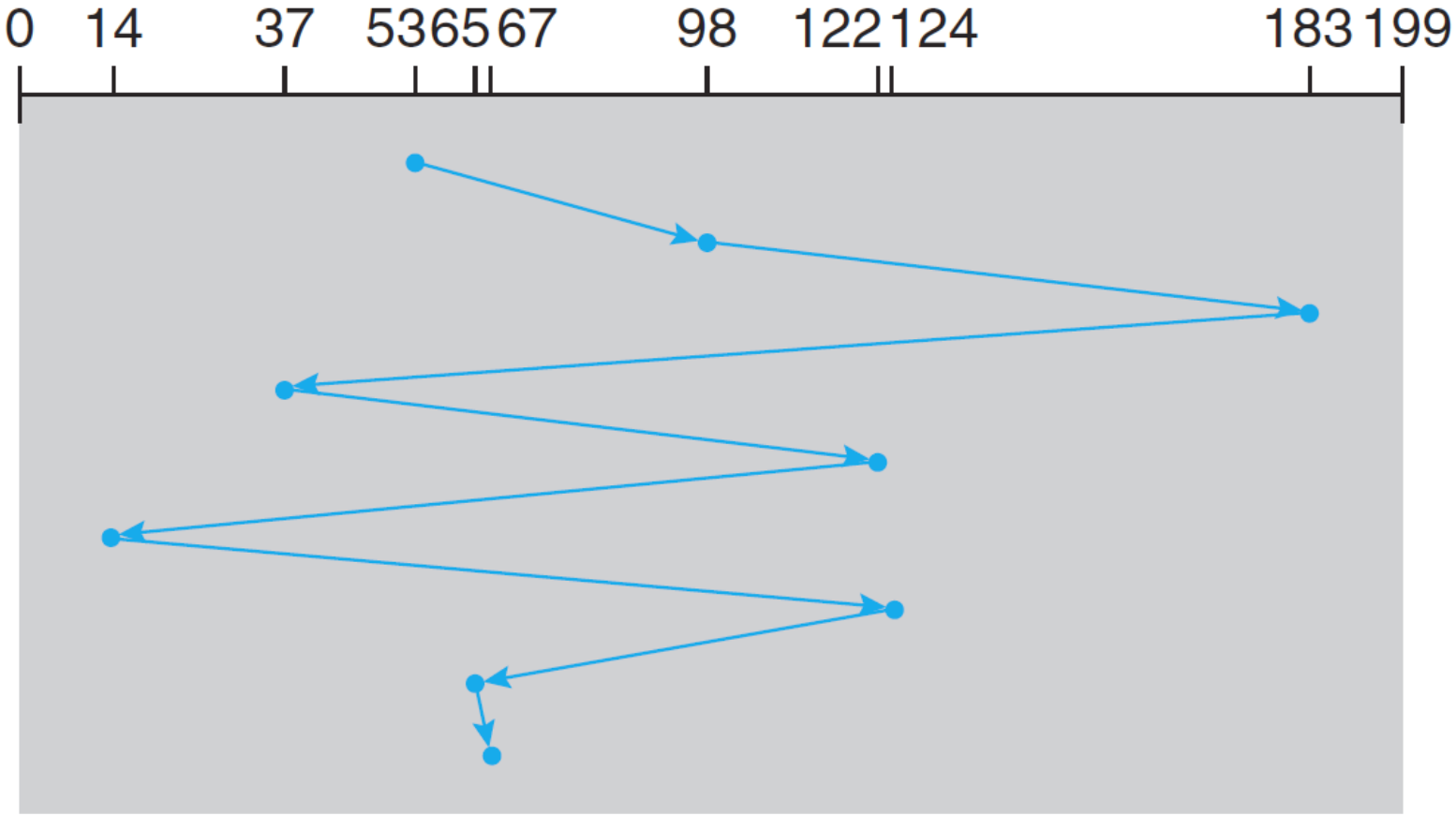
© by Dr. X. Sun

I/O Scheduling for Block Devices

❑ FCFS

- Most common baseline policy
- Total head movement of 640 tracks

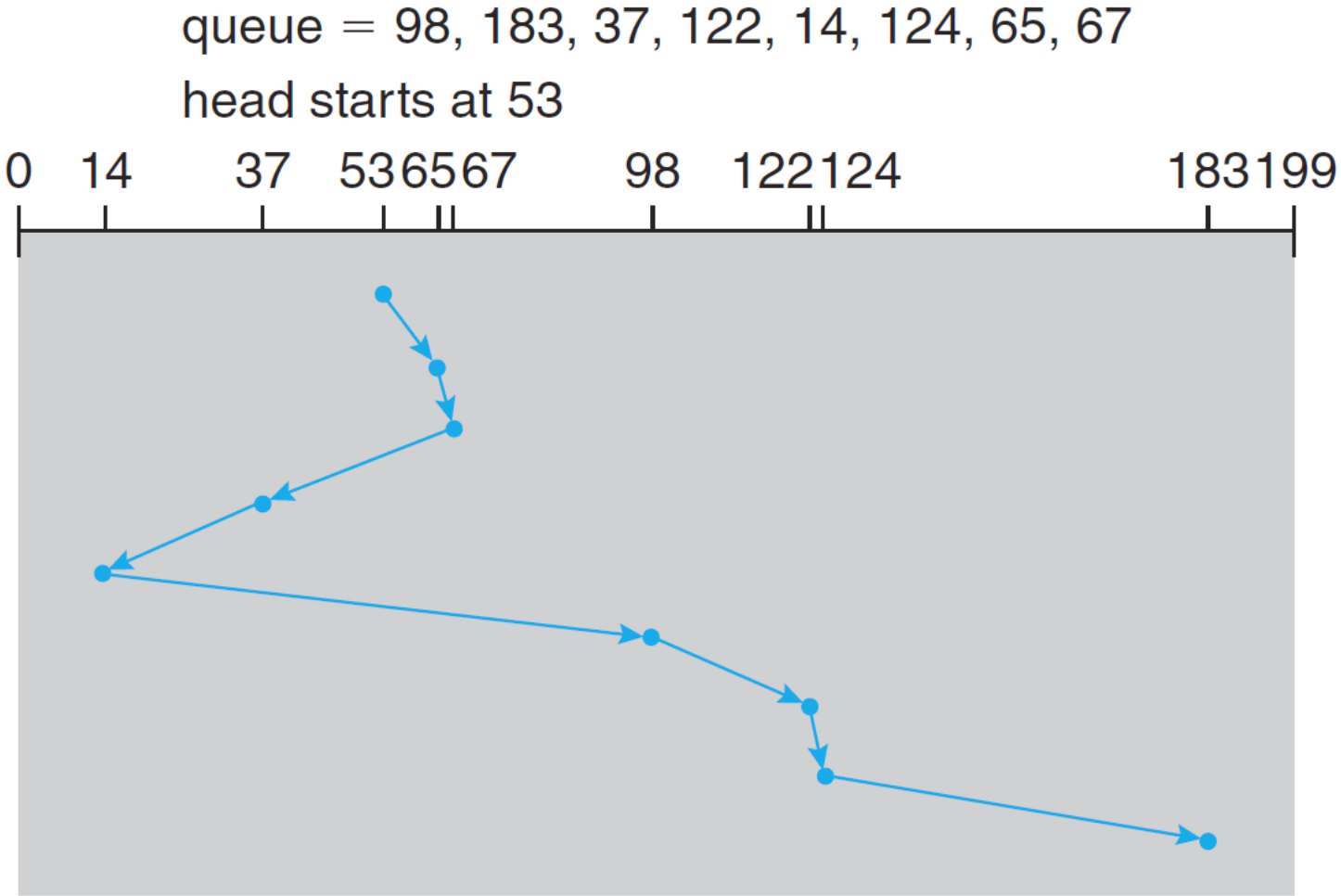
queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53



I/O Scheduling for Block Devices

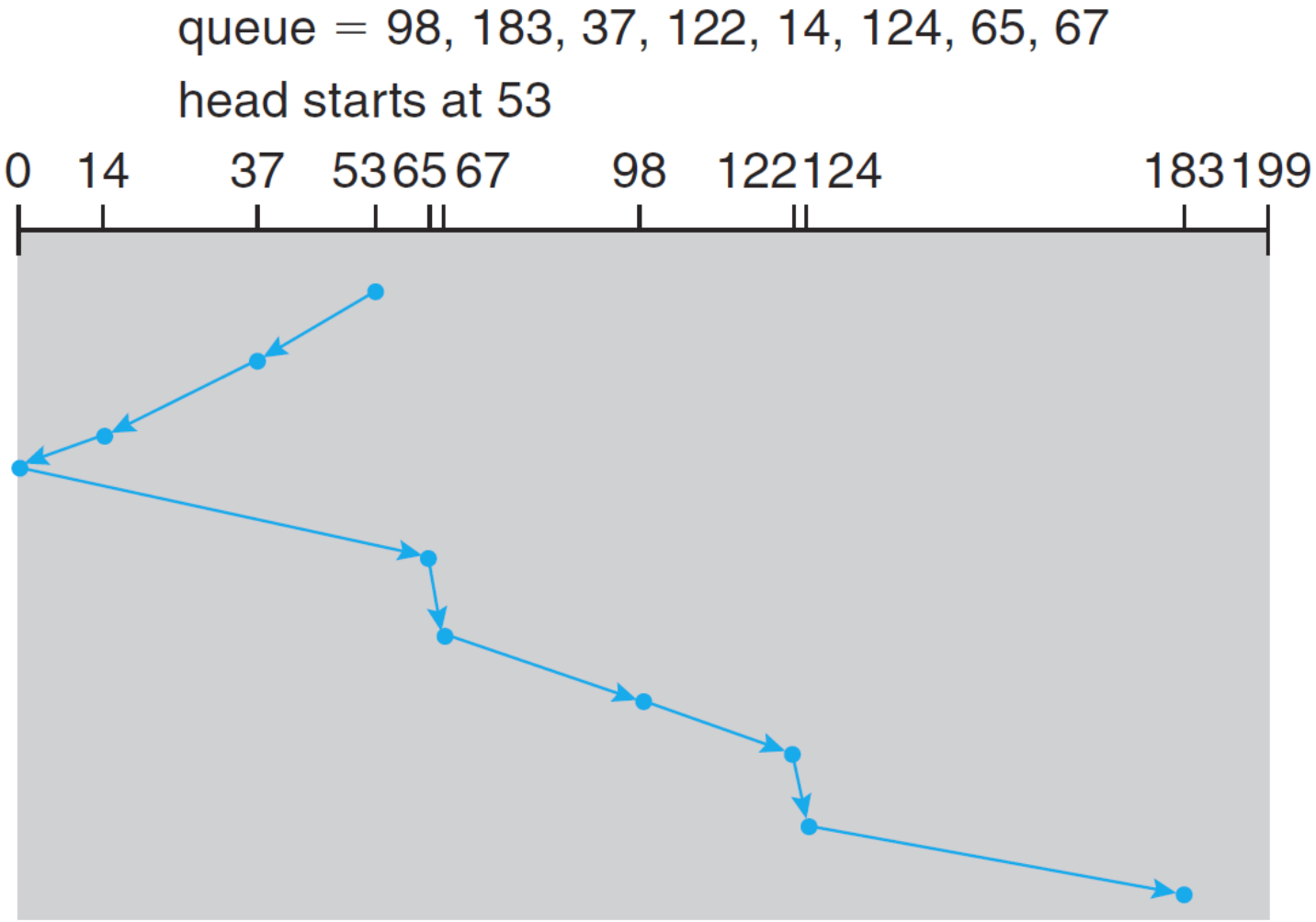
❑ SSTF

- Selects the request with the minimum seek time from the current head position.
- May cause starvation of some requests (e.g., 183 is scheduled last).
- Total head movement of 236 tracks.



I/O Scheduling for Block Devices

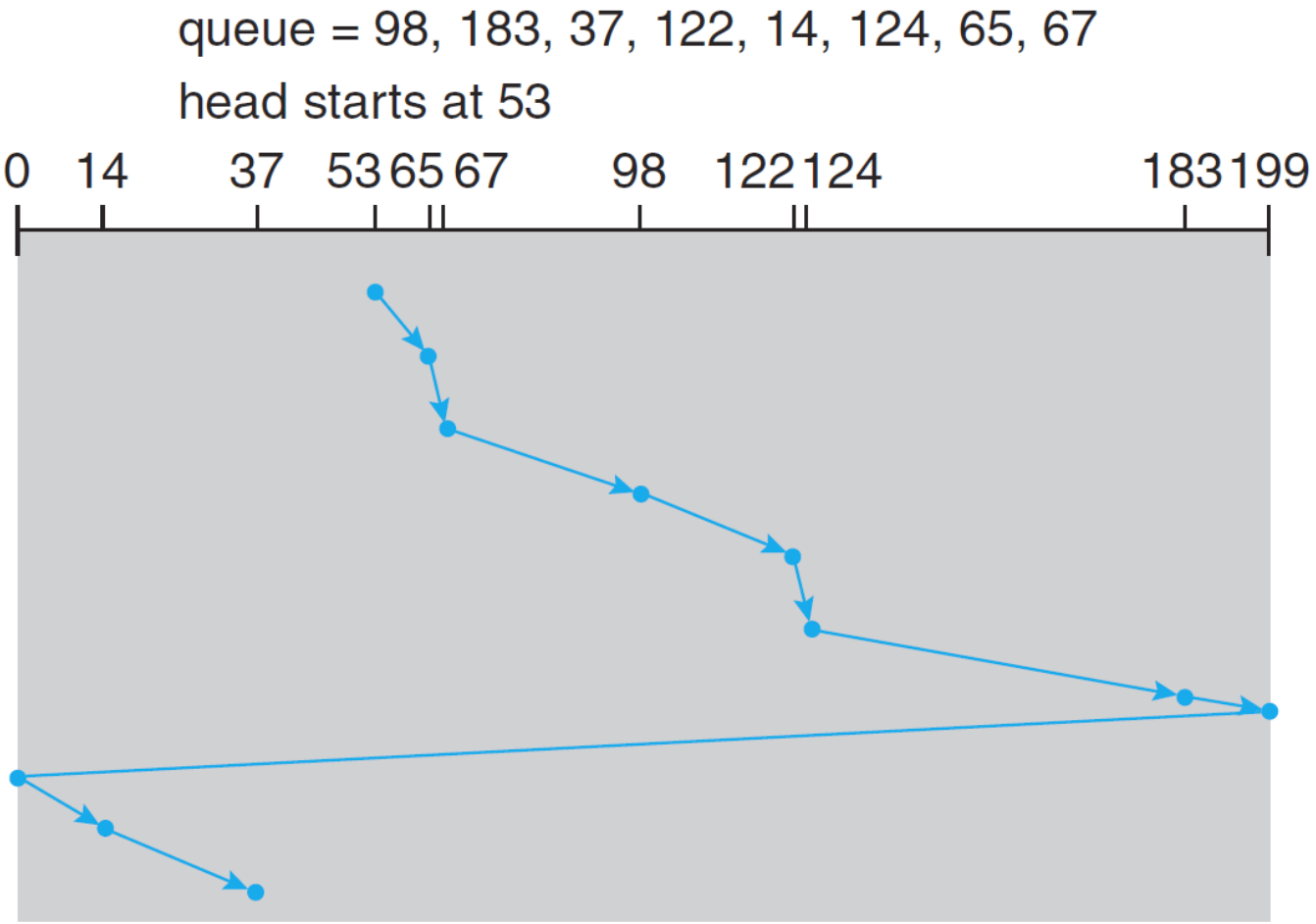
- SCAN- (or Elevator algorithm)
  - The disk arm continues to move to one end of the disk, and then move toward the other end, serving requests during the movement.
  - Assume initially moving backwards, total head movement of 236 tracks.



I/O Scheduling for Block Devices

❑ C-SCAN

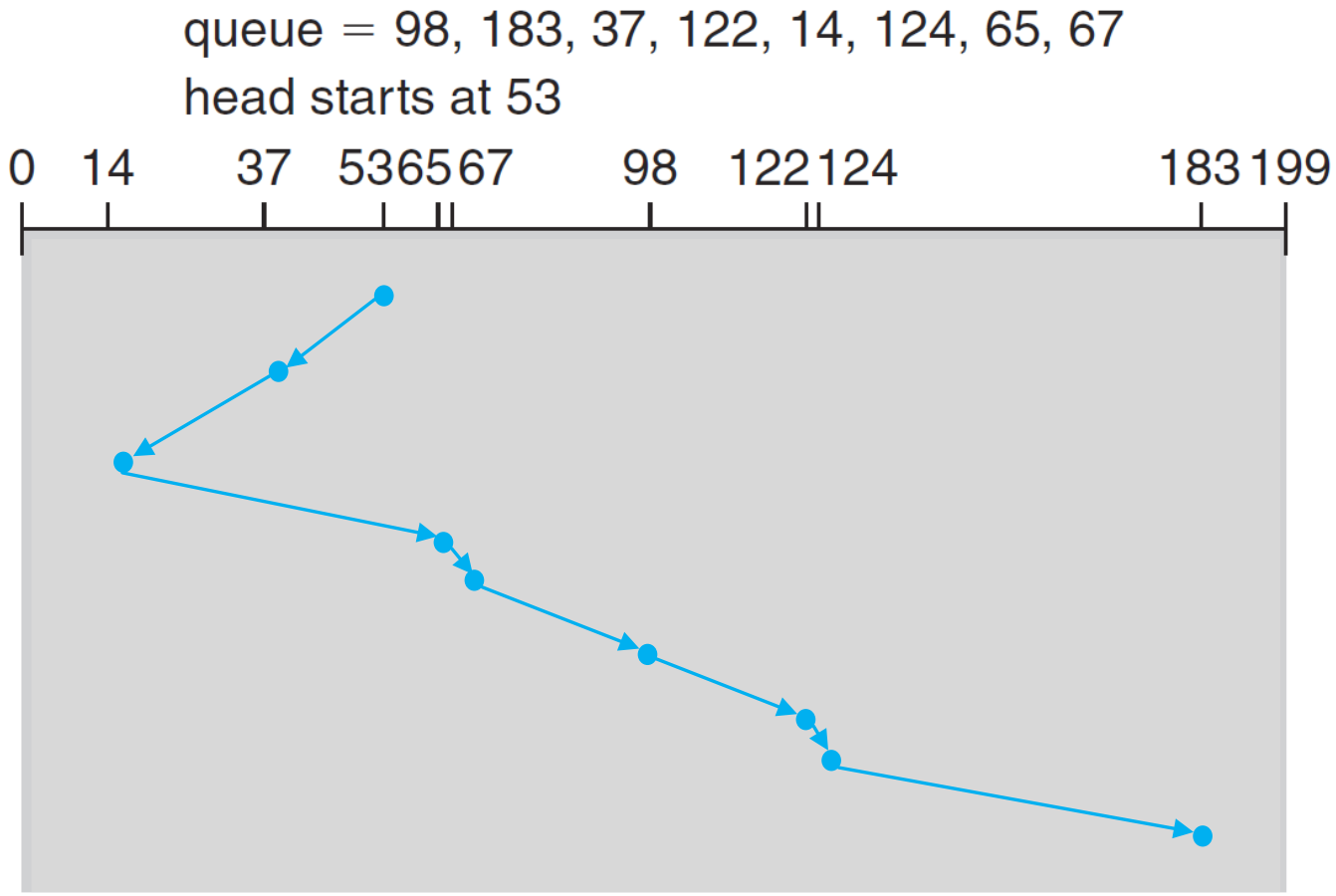
- Provides a more uniform wait time than SCAN. The head moves to one end of the disk, serving requests as it goes.
- When it reaches the end, it immediately returns to the other end of the disk, without servicing any requests on the return trip.
- Treats the tracks as a circular list that wraps around from the last track to the first one.
- Assume initially moving forwards, total head movement of 183(+199) tracks.



I/O Scheduling for Block Devices

❑ Look

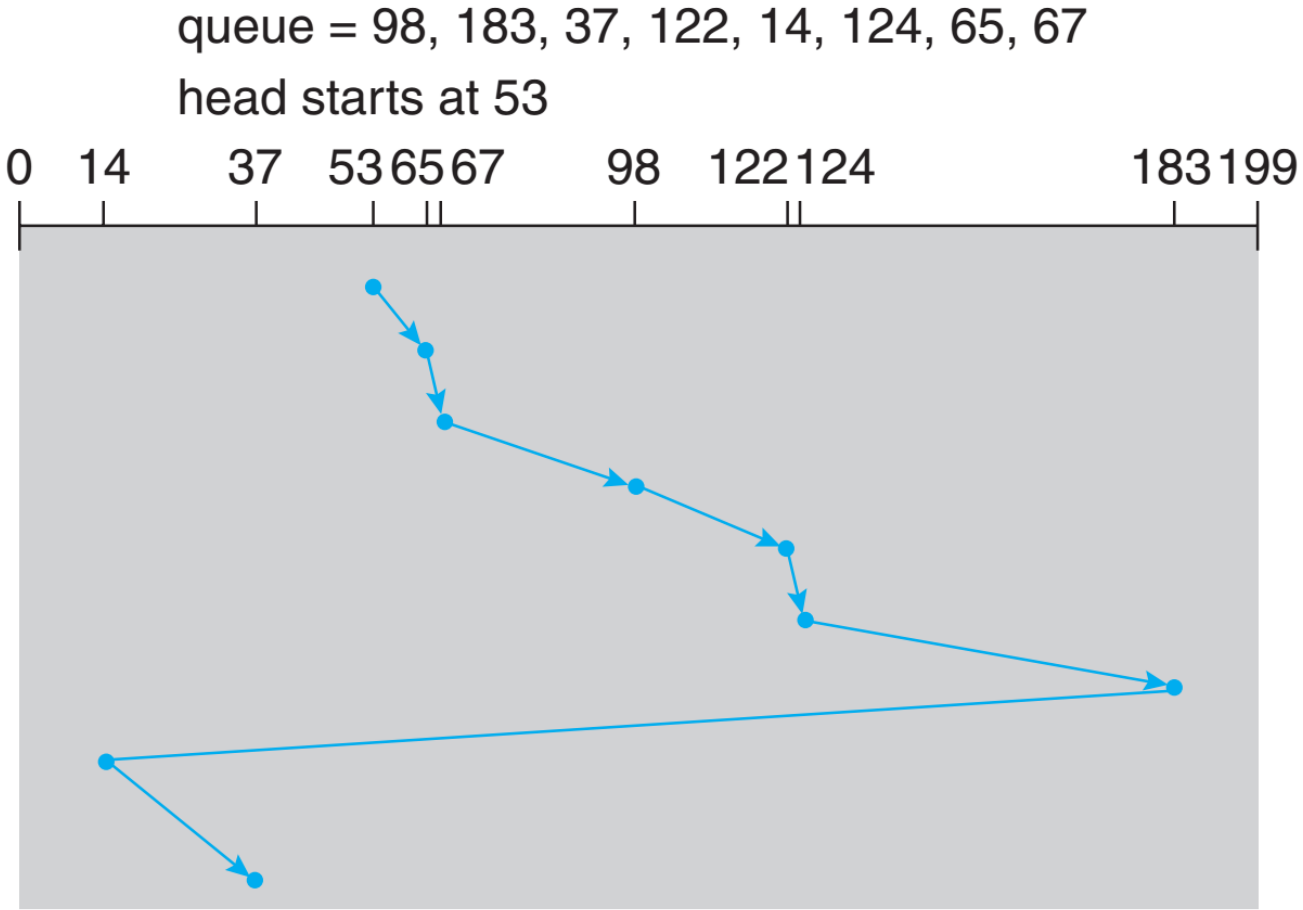
- Similar to SCAN.
- The disk arm only goes as far as the last request in each direction, without going all the way to the end of the disk.
- Assume initially moving backwards, sequence will be (53), 37, 14, 65, 67, 98, 122, 124, 183.
- Total head movement of 53-14+183-14=208 tracks.



I/O Scheduling for Block Devices

C-Look

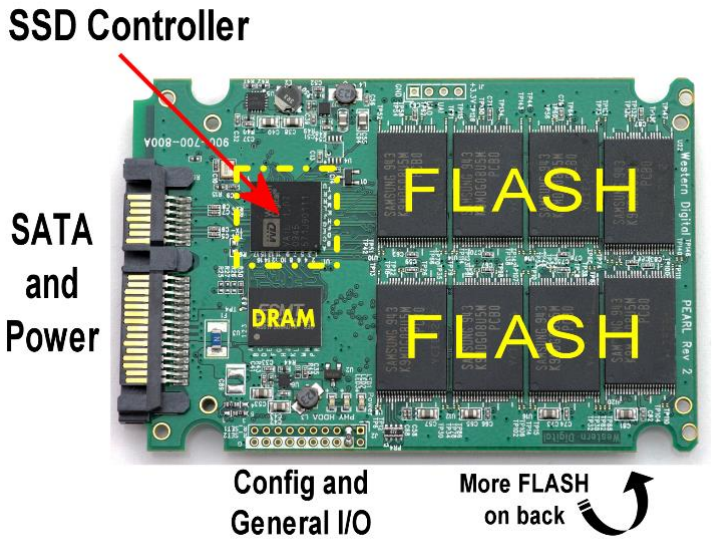
- Similar to C-SCAN.
- The disk arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.
- Assume initially moving forwards, total head movement of 153(+169) tracks.



© by Dr. X. Sun

I/O Scheduling for Block Devices

- Solid State Drive (SSD)
  - No moving mechanical components
  - Have lower internal response time.
  - Use flash memory
    - ✓ retaining data without power
    - ✓ No seek latency
    - ✓ Efficient random I/O, 100ns compared to 1-10ms with HDD
    - ✓ Limited life time by # of rewrites
    - ✓ Data loss from SSD may not be recovered
  - Small and light in weight
  - Generally require $\frac{1}{2}$ of the power of HDDs.



- In order to provision high reliability and performance, hybrid drives, which containing a large hard disk drive and an SSD, are applied.

ECE437/CS481

M06F: RAID

CHAPTER 10.7

Xiang Sun

The University of New Mexico

MULTIPLE DISKS: RAID

❑ Motivation

- Moore's law: CPU speed doubles every 18 month
- SRAM speed increases by 40-100% a year
- However, disk seek time only improves 5-10% a year

❑ Solution: Redundant Array of Independent Disks (RAID)

- Originally, known as Redundant Array of Inexpensive Disks (RAID), to combine multiple, small inexpensive disks drive into an array of disk drives which yields performance exceeding that of a Single, Large Expensive Drive(SLED).
- The array of drives appear to the computer as a single logical storage unit or drive.

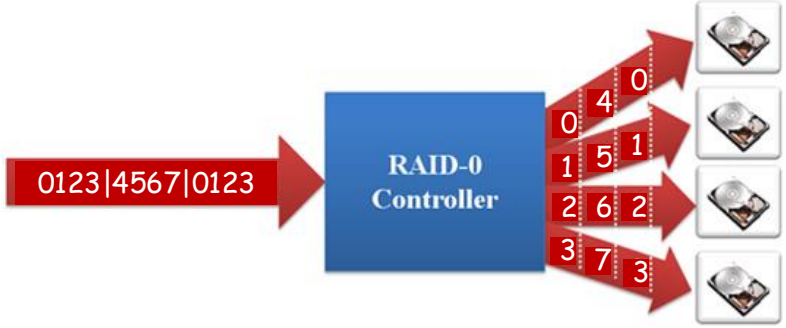
MULTIPLE DISKS: RAID

❑ Parallel disk systems

- How RAID improves performance as compared to a single drive?

- Solution—parallel disk systems, which is achieved by data stripping
  - ✓ Fundamental to RAID
  - ✓ A method of concatenating multiple drives into one logical storage unit
  - ✓ Splitting the bits of each byte across multiple disks: bit – level striping

    For example, an array of four disks, write bit i of each byte to disk i%4



  - ✓ The data transferring rate of the RAID is four times of a single drive
  - ✓ Similarly for splitting the blocks of a file across multiple disks: block-level striping

MULTIPLE DISKS: RAID

❑ Reliability via redundancy

- As the number of disks per component increases, the probability of failure also increases .
  - ✓ Suppose the mean time to failures (MTTF) of a single disk is 100,000 hrs.
  - ✓ What is the MTTF if there are N disks in a RAID system?

$$\mathrm{MTTF}=\frac{1}{1-\left(1-\frac{1}{100{,}000}\right)^{N}}$$

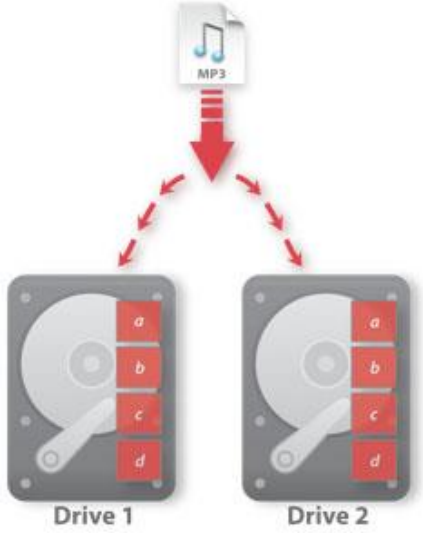
  - ✓ If N=100, then MTTF≈ 1000.5 hrs =41.6875 days !! ---Not enough
  - ✓ If N=2, then MTTF≈ 50,000.25 hrs ≈2084 days ≈ 5.7 years

- Solution?
  - ✓ Redundancy!

© by Dr. X. Sun

MULTIPLE DISKS: RAID

- Redundancy achieved by mirroring
  - Duplicate every disk.
  - Logical disk consists of two physical disks
  - Every write is carried out on both disks
  - If one of the disk fails, data read from the other
  - Data permanently lost only if the second disk fails before the first failed disk is replaced



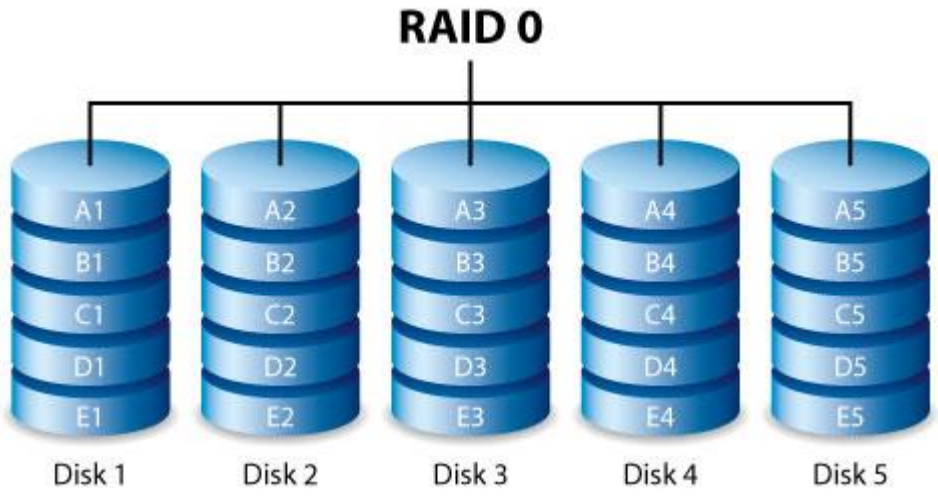
- The probability of failure for a mirrored disk system significantly reduces.
  - Suppose the MTTF of a single disk is 100,000 hrs, i.e., the probability of failure for a disk=$\frac{1}{100,000}$ per hr.
  - Assume that the mean time to replace a failure disk in a mirrored disk system is 10 hrs.
  - Thus, the probability of failure for a mirrored disk system p= $2 \times \left(\frac{1}{100,000}\right)^2 \times 10$
  - Thus, MTTF of the mirroring RAID system=$\frac{1}{p} = 57,000\ years$
- Main disadvantage: Most expensive approach.

© by Dr. X. Sun

MULTIPLE DISKS: RAID

- Design of RAID-six levels representing six alternative designs

  - RAID Level 0: Non-redundant Disk Striping
    - ✓ Multiple disks are connected to a single disk controller.
    - ✓ Data is striped to spread segments across multiple drives
    - ✓ Improvement of I/O performance, not data redundancy
      - provide a high transfer rate, by overlapping disk read and write
      - possible interspersed read/write
      - if one of drives fails, the whole system may fail.

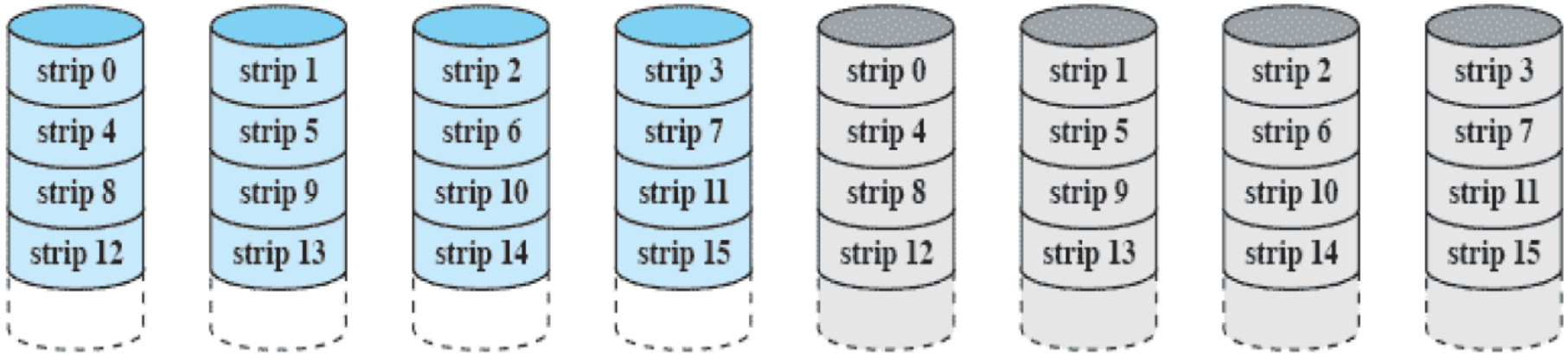


© by Dr. X. Sun

MULTIPLE DISKS: RAID

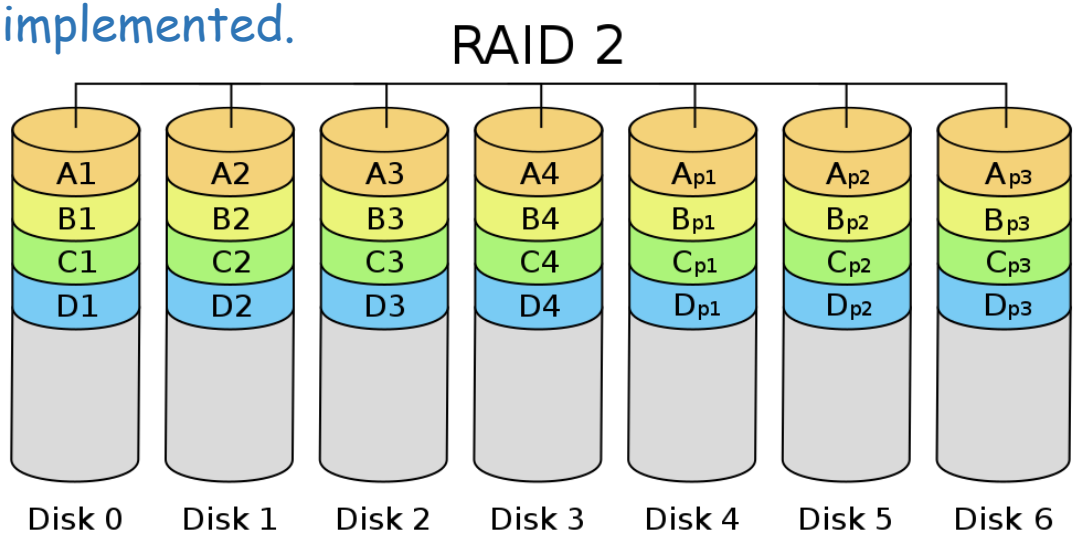
- Design of RAID-six levels representing six alternative designs

  - RAID Level 1: Disk Mirroring (or shadowing)
    - ✓ RAID 0 plus duplicated data stored in the identical drives, i.e., every main drive has a minor drive, backup/mirror.
    - ✓ Improvement of not only performance but also data redundancy.
      - expensive way to achieve data redundancy.
      - good reliability, if a single drive fails, its mirror drive takes over.



MULTIPLE DISKS: RAID

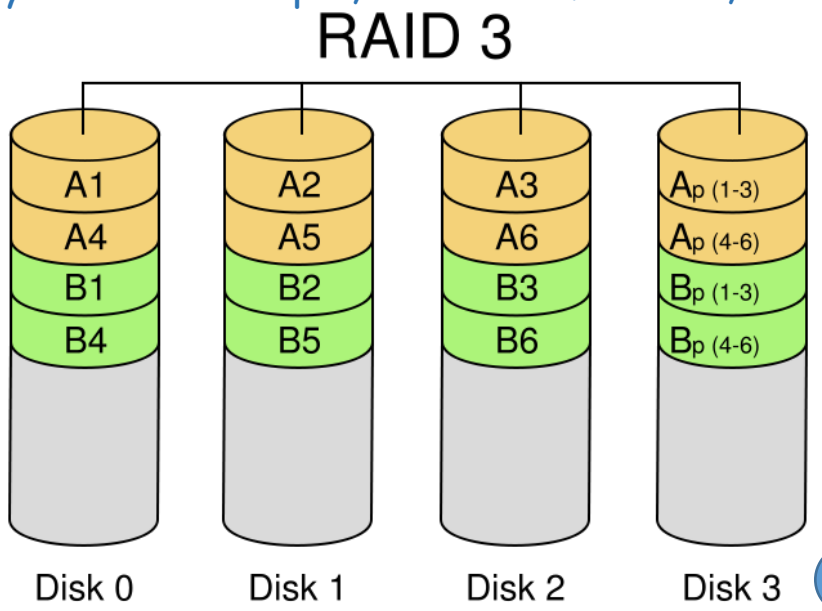
- Design of RAID-six levels representing six alternative designs
  - RAID Level 2: Bit Interleaving of Data with HECC (Hamming Error Correcting Code)
    - ✓ Requires fewer disks than Level-1 to provide redundancy, but still needs quite a few more disks
      - $m+n<=2^n$
      - m=4 data disks need n=3 check disks.
    - ✓ Write request requires all the disks.
    - ✓ Given high reliability of disks with the expense of complicated coding algorithm.
      - RAID Level 2 is an overkill and is never implemented.



MULTIPLE DISKS: RAID

- Design of RAID-six levels representing six alternative designs

  - RAID Level 3: Bit Interleaving of Data with Parity (Parallel Disk Array)
    - ✓ One parity disk is used to do on-the-fly parity generation and parity checking, capable of correcting any single and self-identifying failure.
      - Suppose we have 3 data disks and one parity disk. The sample bits in the data disks are: 0,1,1.
      - The parity bit is the XOR of these three data bits, which can be calculated by adding them up and writing a 0 if the sum is even and a 1 if it is odd. Here, the sum of Disk 0 through Disk 2 is "2", so the parity is 0.
      - Now if we attempt to read back this data, and find that Disk 2 gives a read error, we can reconstruct Disk 2 by conducting XOR of all the other disks, including the parity. In the example, the sum of Disk 0, 1 and Parity is "1", so the data on Disk 2 must be 1.
    - ✓ RAID Level 3 requires a minimum of 3 drives to implement.
      - # of redundant disks is 1 (constant, no matter how many drives).
      - works only if it is known which disk fails.
      - to access a single file block of data, must access all the disks. This allows good parallelism for a single file access, but doesn't allow multiple I/Os (i.e., multiple file access at a time).



© by Dr. X. Sun

MULTIPLE DISKS: RAID

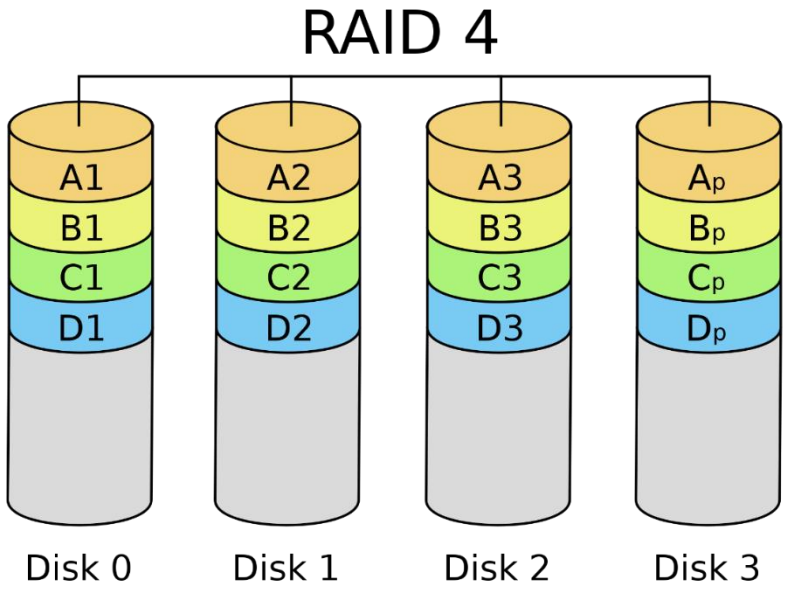
- Design of RAID-six levels representing six alternative designs

  - RAID Level 4: Block Interleaving of Data with Parity (Parallel Disk Array)
    - ✓ Similar to RAID Level 3, except that data is interleaved across disks of striping unit (striping width) rather than in bits.
    - ✓ Still use a single disk for parity; however, the parity is calculated over data from multiple striping units.

      - If an error detected, we may have to read other striping units on other disks to correct data.
      - What if we modify a data bit in a striping unit? Do we need to read striping units from other disks and recalculate the parity data?—No, can use the following formula.

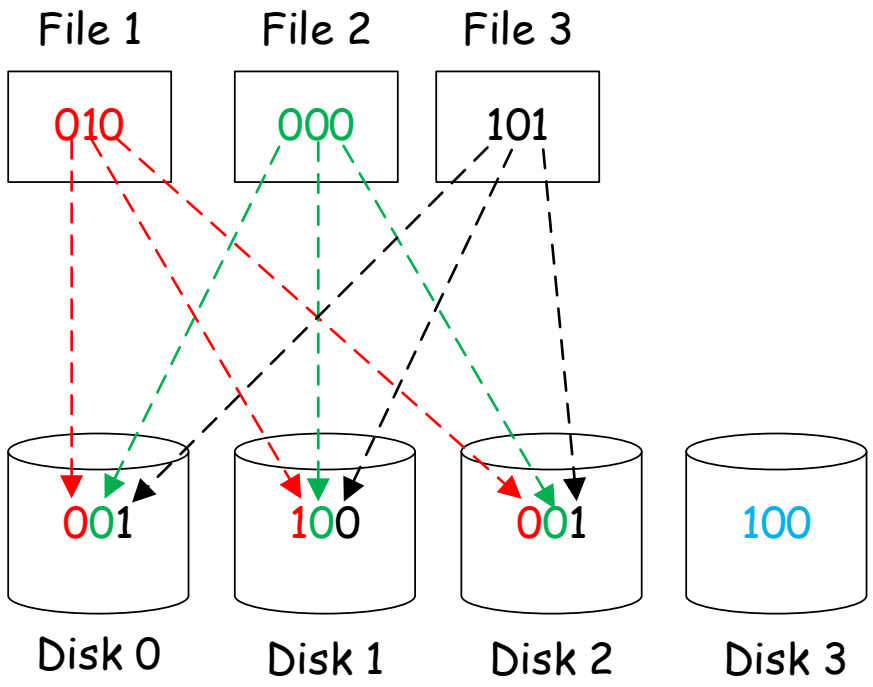
        new parity = (old data xor new data) xor old parity

        which requires 2 reads and 2 writes.
      - Allow multiple file reads at a time but may not allow parallelism for a single file access.
      - Only one write is allowed at a time.

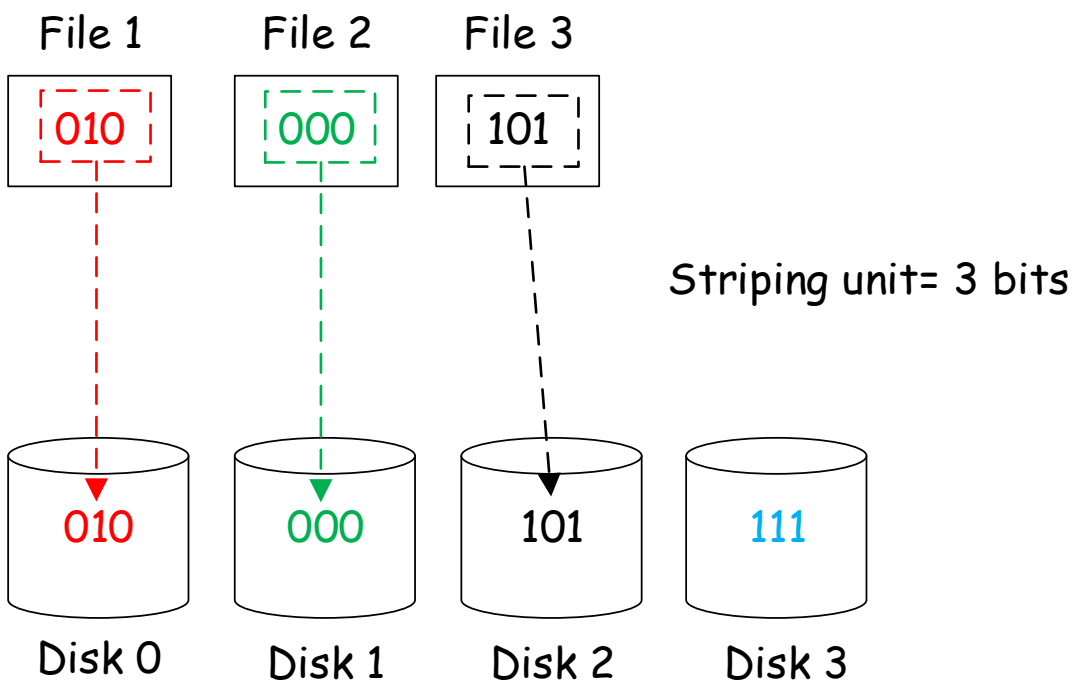


© by Dr. X. Sun

❑ Comparison between RAID Level-3 and Level-4

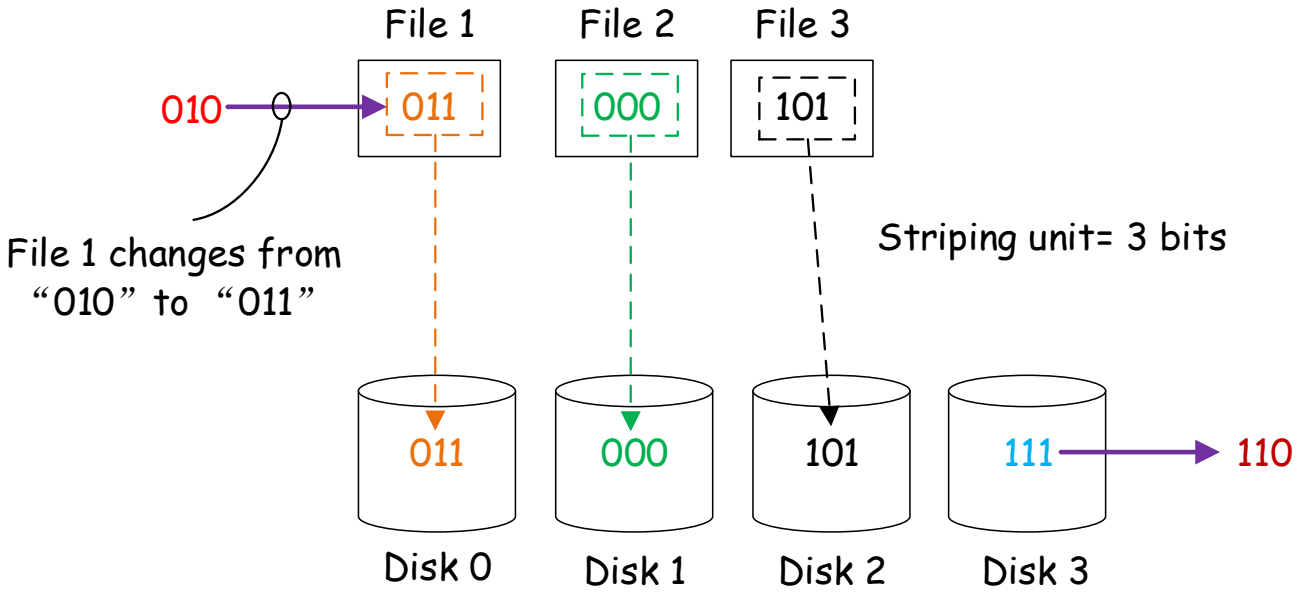


RAID Level-3



RAID Level-4

❑ Write on a striping unit in RAID Level-4



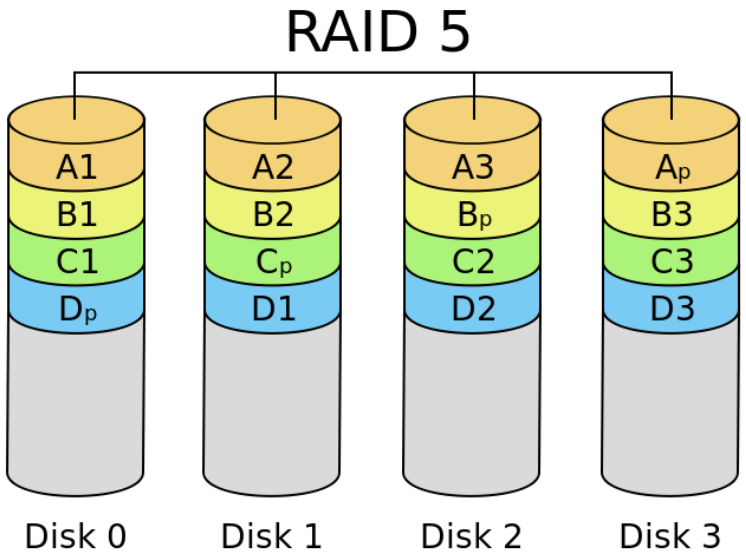
RAID Level-4

Old data 010 | New data 011 | Old parity 111

New parity=(old data xor new data) xor old parity
=(010 xor 011) xor 111=110

- Design of RAID-six levels representing six alternative designs

  - RAID Level 5: Block Interleaving with Distributed Parity
    - ✓ Level 5 is to solve the problem of one write being allowed at a time (bottleneck in parity disk) in Level 4.
      - e.g., writing a file on Disk 0 and writing on Disk 1 both require a write to the parity disk
    - ✓ In Level 5, there is no dedicated parity drive in Level 5. The parity bits are across all disks based on a round-robin manner.
      - e.g., writing on A1 and writing on B2 can be proceeded simultaneously.

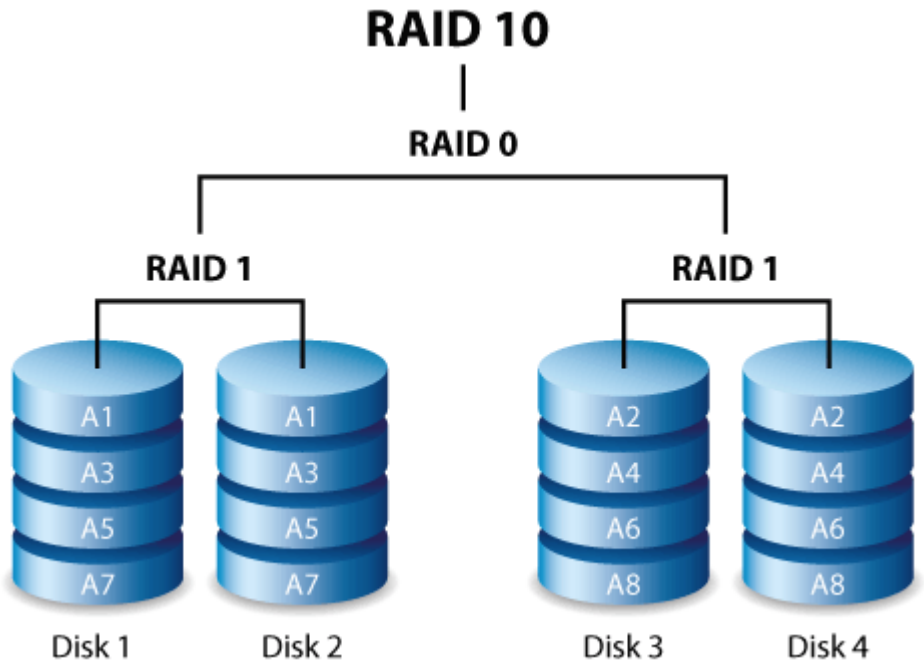


© by Dr. X. Sun

MULTIPLE DISKS: RAID

❑ RAID Level 10: Combine Level 0 and Level 1

- ✓ Using four disks as an example, RAID 10 creates two RAID 1 segments, and then combines them into a RAID 0 stripe.
- ✓ How about eight disks?



ECE437/CS481

M07A: MEMORY MANAGEMENT

CHAPTER 8.1-8.6

Xiang Sun

The University of New Mexico

Goals and Tools of Memory Management

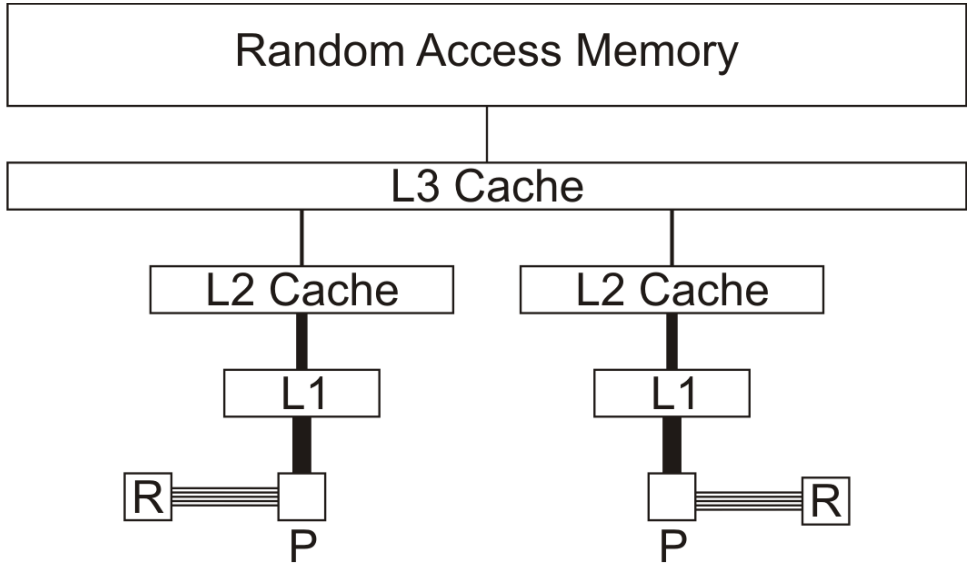
- Allocate memory resources among competing processes
  - maximizing memory utilization and system throughput.
- Provide isolation between processes
  - Addressability and protection: orthogonal
- Convenient abstraction for programming
  - and compilers, etc.
- Tool
  - Base and limit registers
  - Swapping
  - Segmentation
  - Paging, page tables and TLB
  - Virtual memory

© by Dr. X. Sun

Memory, cache, and CPU

❑ Architecture

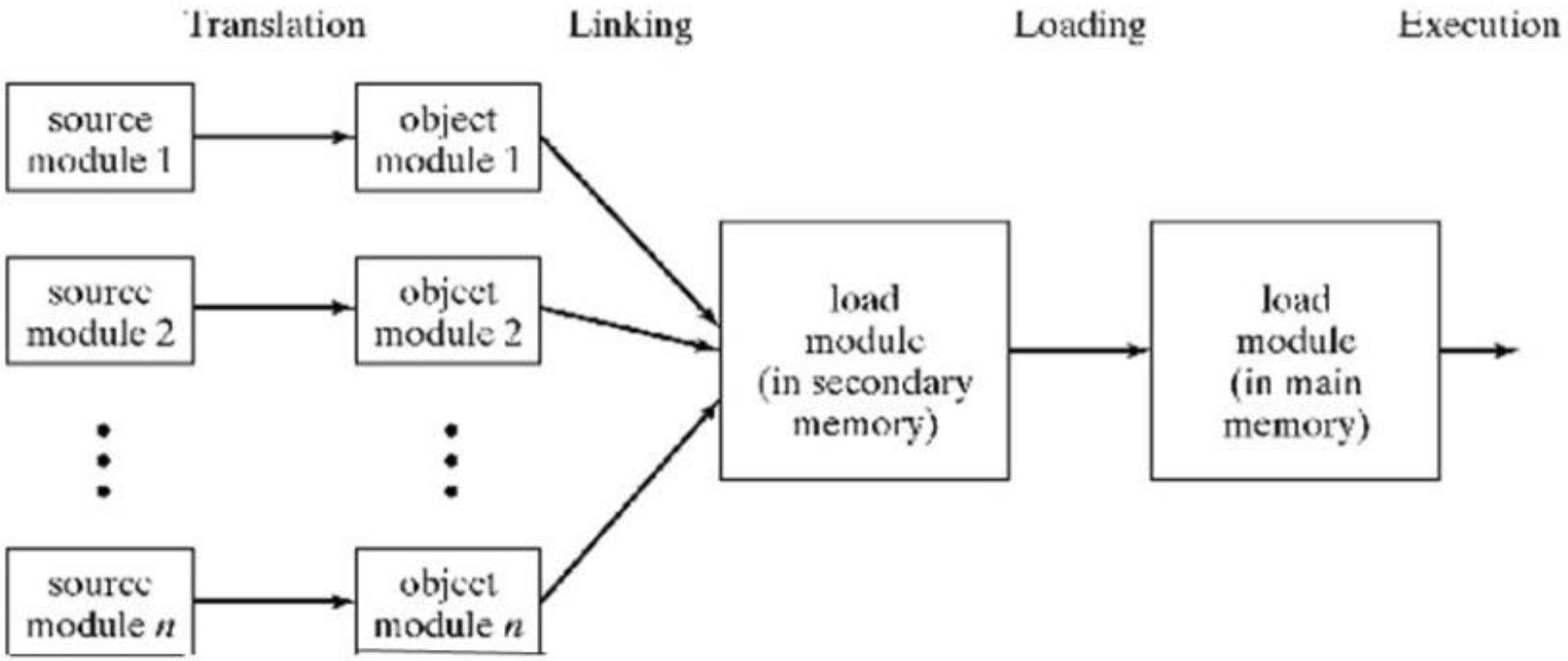
- To run a process, programs will be read from 2nd storage (disks) into memory.
- During the process lifetime, both instructions and data need moved between memory and CPUs.
- CPU can read/write
  - ✓ Registers--within a CPU clock
  - ✓ Memory--taking many cycles—memory stall
  - ✓ Cache--between main memory and CPU registers



Preparing Program for Execution

❑ Program Transformations

- Translation (Compilation) of program
- Linking of program with other object modules needed for its execution
- Loading of the modules in the memory for purpose of execution



© by Dr. X. Sun

Preparing Program for Execution

❑ A sample code segment

Source code in the text editor

```
...
static int gVar;
...
int proc_a(int arg){
        ...
        gVar = 7;
        put_record(gVar);
        ...
}
```

Relocatable file in secondary memory

Code Segment	
Relocate Address	**Generated Code**
0000	(Other module)
…	
1008	entry proc_a
…	
1220	load 7 , %rax
1224	store %rax , 3136
1228	push %rbx
1232	call 2334
…	
1399	(End of proc_a)
…	(Other modules)
2334	entry put_record
…	
2399	(End of put_record)
…	
2999	(Last location in the code segment)

Code Segment	
Relocate Address	**Generated Code**
…	
3136	[Space for gVar variable]
…	
4000	(last location in the data segment)

Load module in main module (The Program Loaded at Location 4000)

Absolute Address	Generated Code
0000	(Other process's programs)
4000	(Other modules)
…	
4000	(Other modules)
…	
5008	entry proc_a
…	
5220	load 7 , %rax
5224	store %rax, 7136
5228	push %rbx
5232	call 6334
…	
6399	(End of proc_a)
…	(Other modules)
6334	entry put_record
…	
6399	(End of put_record)
…	
6999	(Last location in the code segment)
7000	(first location in the data segment)
…	
7136	(Space for gVar variable)
…	
8000	(last location in the data segment)

- Relocate addr: the type of address that needs to be relocated during the linking procedure.
- Logical/Virtual addr: the type of address within a process after linking.
- Physical addr: the type of address in the physical memory.

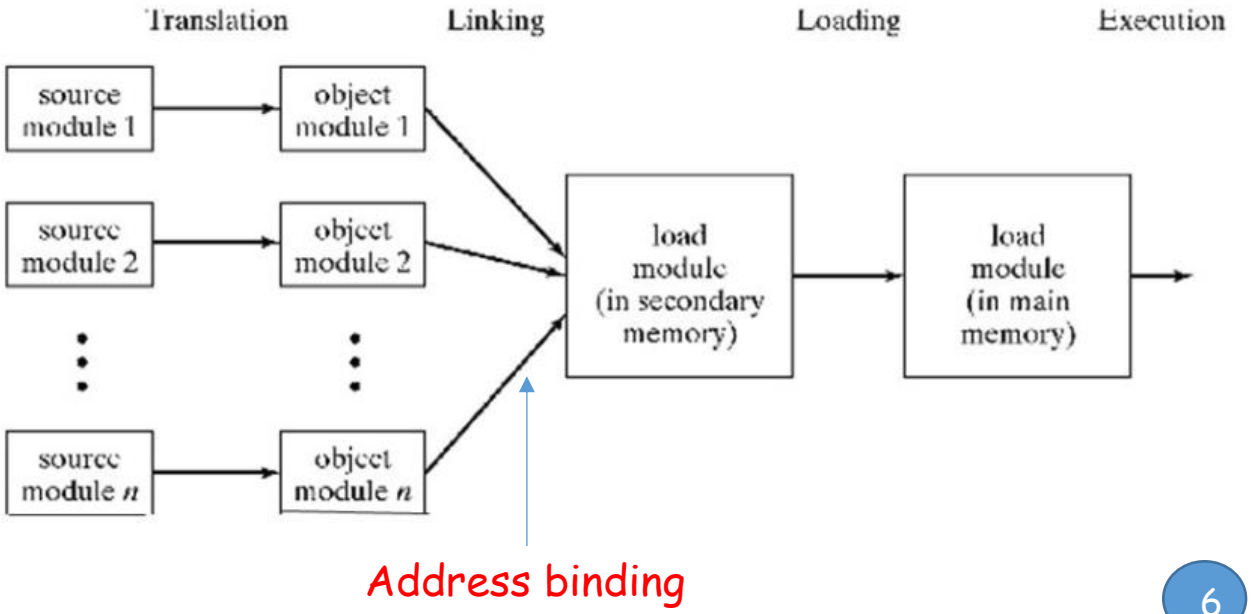
© by Dr. X. Sun

Address Binding

❑ Address binding

- ➢ Binding the logical address of codes and data to physical memory address (i.e., absolute address)

- ➢ The address binding can happen at three different stages
  - ✓ Compile time: the type of address in the ELF file stored in the secondary memory is the physical memory address.
    - Compiler has to know the main memory location in advance.
    - Compiler has to recompile the codes if the location is changed.

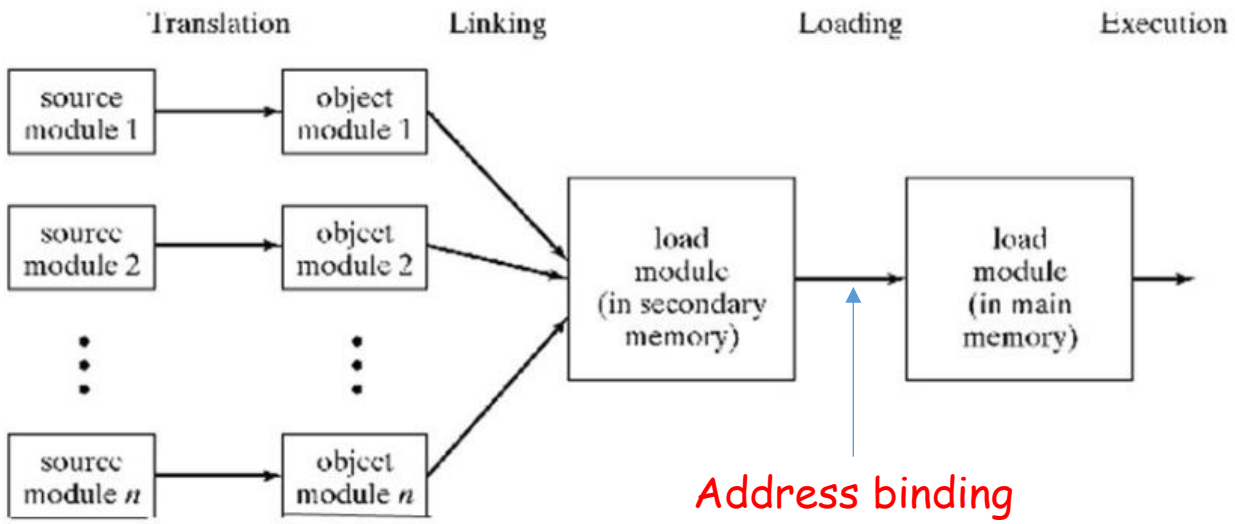


Address Binding

❑ Address binding

- Binding the logical address of codes and data to physical memory address (i.e., absolute address)

- The address binding can happen at three different stages
  - ✓ Loading time: the type of address in the ELF file stored in the secondary memory is the logical address. But the type of address in the virtual memory layout is the physical memory address.
    - Compiler does not know the main memory location.
    - The address binding happens when the program is loaded into the main memory.

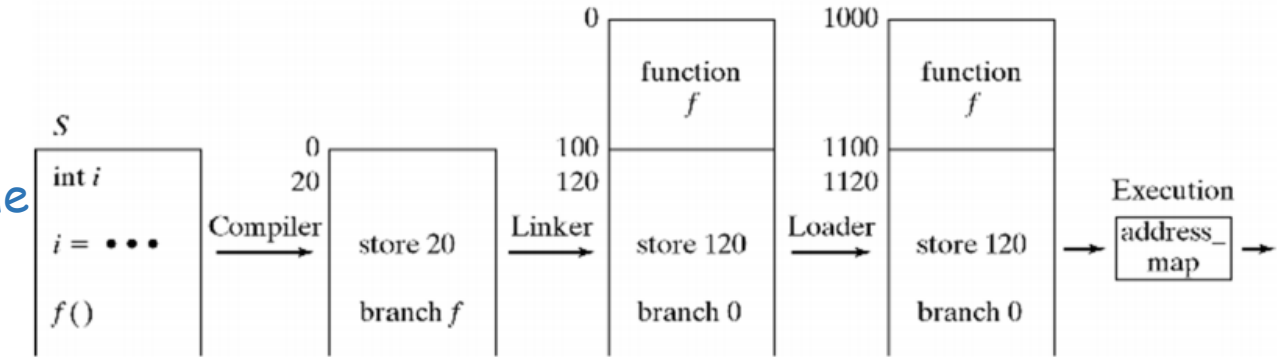


Address Binding

❑ Address binding

- Binding the logical address of codes and data to physical memory address (i.e., absolute address)

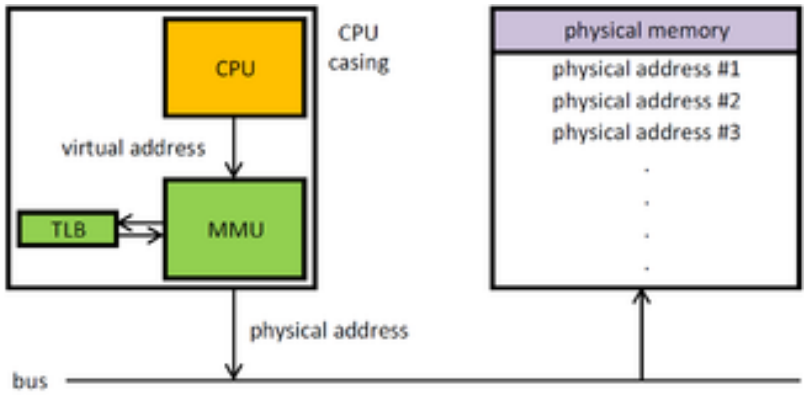
- The address binding can happen at three different stages
  - ✓ Running time: the type of address in the ELF file stored in the secondary memory and in the virtual memory layout is the logical address. But the logical address will be translated into the physical memory address during the execution.
    - Compiler does not know the main memory location.
    - The address binding happens when the process is executed.



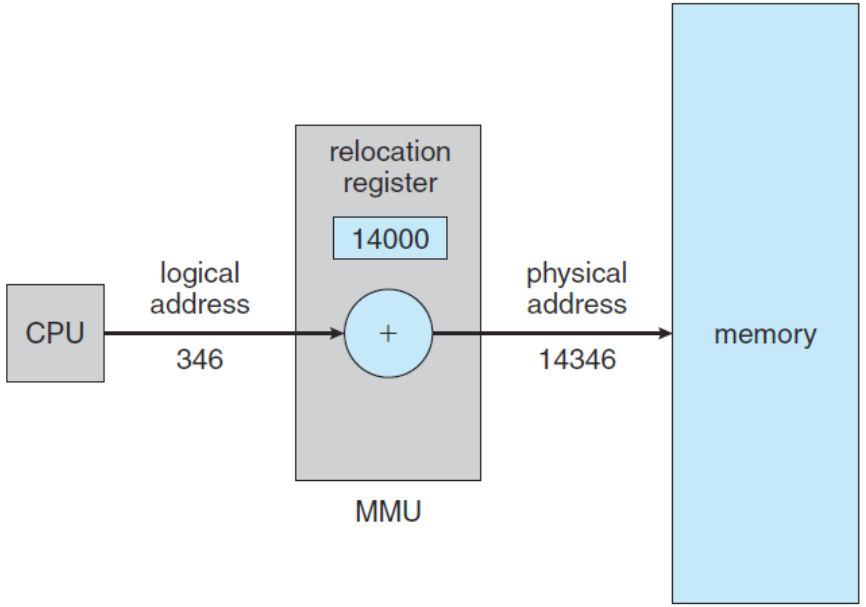
Address Binding

- Running time address binding
  - Running time mapping from logical address space into a separate physical address space
  - The mapping is achieved by Memory Management Unit (MMU, hardware support).
    - ✓ With MMU, the value in the relocation register is added to every address generated by a process at the time it is sent to memory

Physical address =
Logical address + relocation register



CPU: Central Processing Unit
MMU: Memory Management Unit
TLB: Translation lookaside buffer



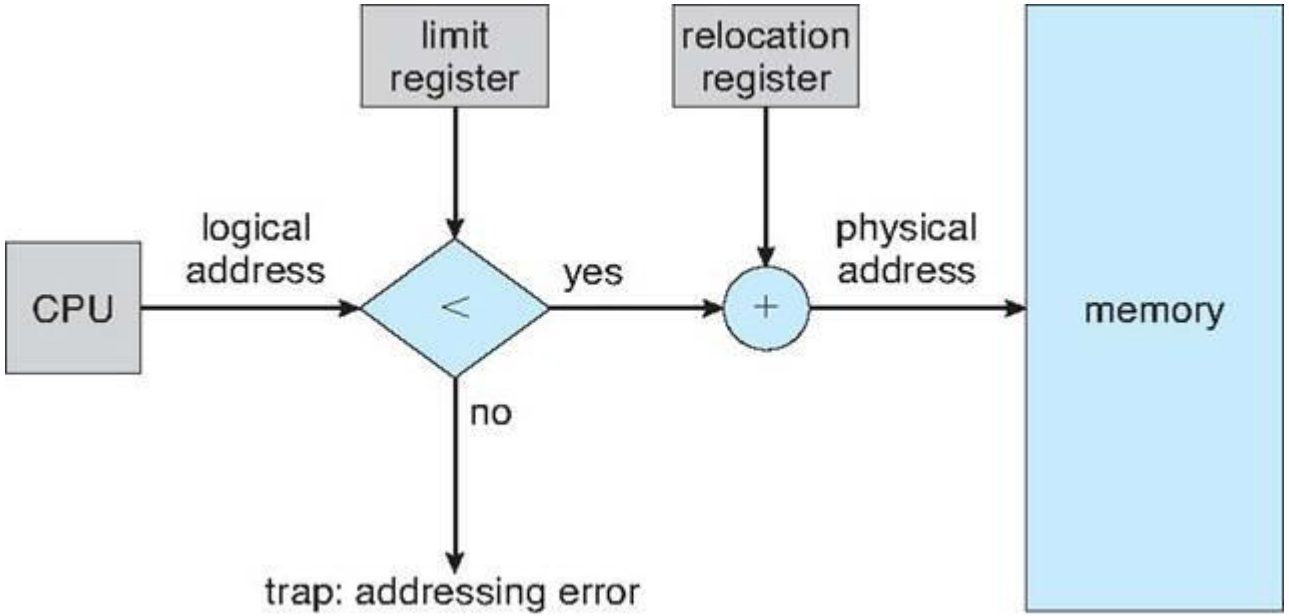
Address Binding

❑ Running time address binding

- In order to avoid the process in accessing other processes' memory spaces, the relocation/base register and the limit register are used.
  - ✓ Limit register contains range of logical addresses – each logical address must be less than the limit register

- Is logical address beyond the limit register?
  - If yes, "trap" to deal with memory access errors
  - Otherwise, go to physical address

Physical address =
Logical address + base/relocation register

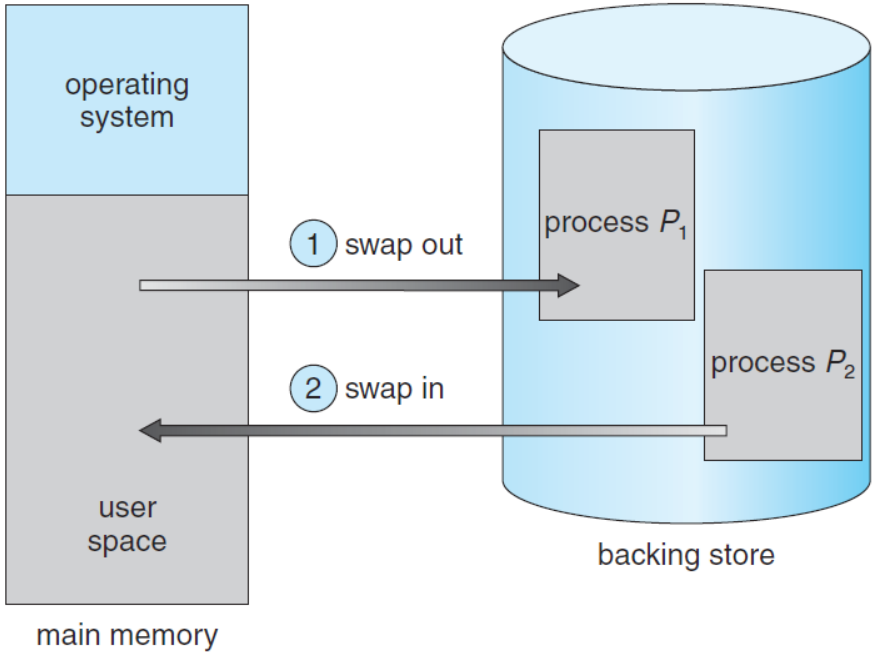


Swapping

❑ Swapping (Recall)

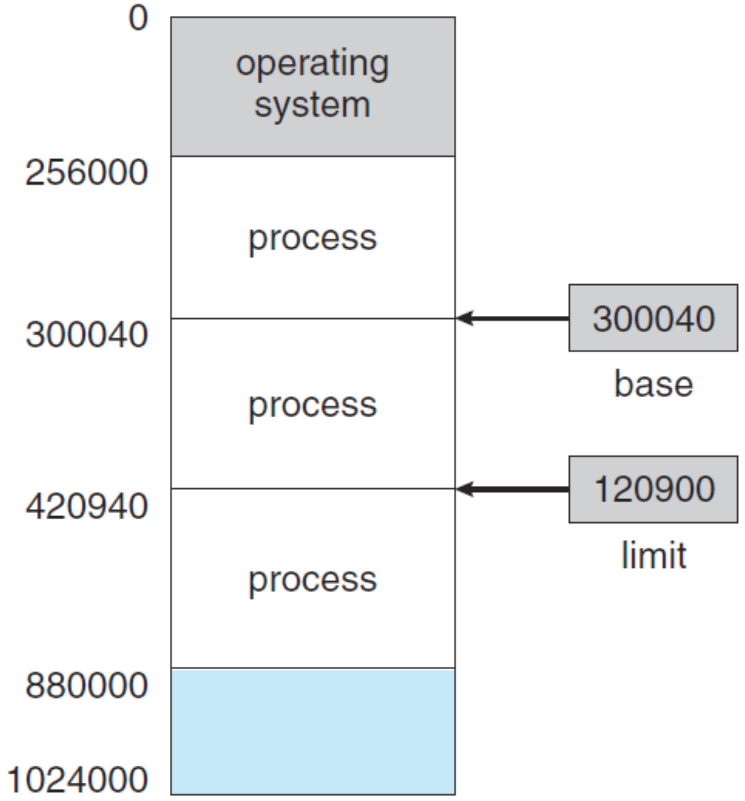
- A process can be swapped temporarily out of memory to the backing store (e.g., hard drives), and then brought back into memory for continued execution
  - ✓ Swap out: memory-to-backing store
  - ✓ Swap in: backing store-to-memory

- The short-term CPU scheduler (dispatcher) decides which process is executed next. After the decision, it checks if the desired process is in the ready queue in the memory.
  - ✓ If yes, does nothing.
  - ✓ If no, checks if there is enough memory space for the desired process to be swapped in.
    - If yes, swaps in the process
    - If no, swaps out a process currently in memory and swaps in the desired process



Memory Allocation

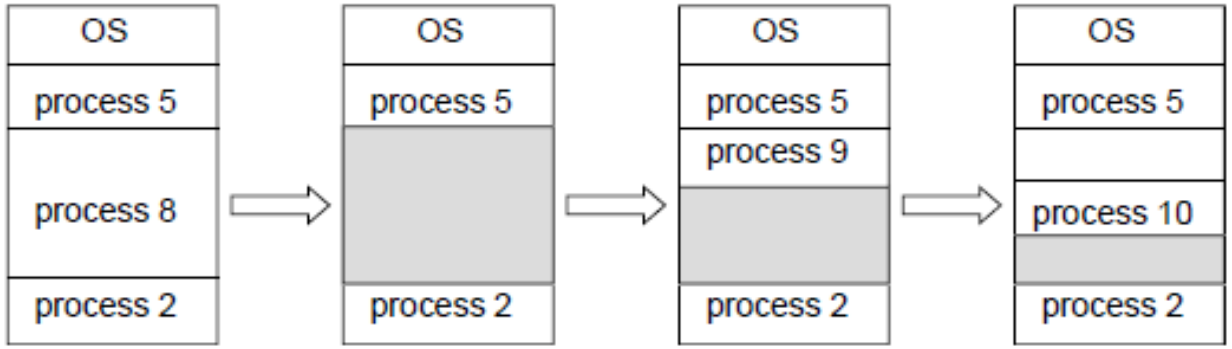
- Memory allocation
  - Main memory is divided into two partitions: Resident operating system (usually held in low memory) and user processes (held in high memory).
  - Main memory has limited resource, and thus must be efficiently allocated.
  - Contiguous memory allocation is one method.
    - ✓ Each process is contained in a partition with single contiguous section of memory
    - ✓ Memory accessing is protected by relocation + limit registers
    - ✓ MMU maps logical address dynamically



Memory Allocation

❑ Contiguous allocation

- Contiguous allocation isn't allocating memory spaces to processes sequentially.
  - ✓ Processes may terminate
  - ✓ Processes may be swapped in/out
  - ✓ New processes may arrive



- Main memory space may be separated into small holes.
  - ✓ Hole – a block of available memory; holes of various size are scattered throughout memory
  - ✓ When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - ✓ OS maintains information about:
    - Allocated memory blocks
    - Free memory blocks (holes), sorted by memory address or sizes

© by Dr. X. Sun

Memory Allocation

- How to satisfy a process requesting of n units memory space from a list of free holes
  - First fit
    - ✓ allocate the first hole that is big enough
    - ✓ may incur many small holes
  - Best fit
    - ✓ allocate the smallest hole that is big enough
    - ✓ may search entire free list
    - ✓ produces the smallest leftover hole
  - Worst fit
    - ✓ allocate the largest hole
    - ✓ must search entire free list
    - ✓ produces the largest leftover hole

© by Dr. X. Sun

Memory Allocation

- Contiguous allocation incurs external fragmentation

  - External fragmentation: memory free space can satisfy a request, but it is not contiguous.

  - Solution:

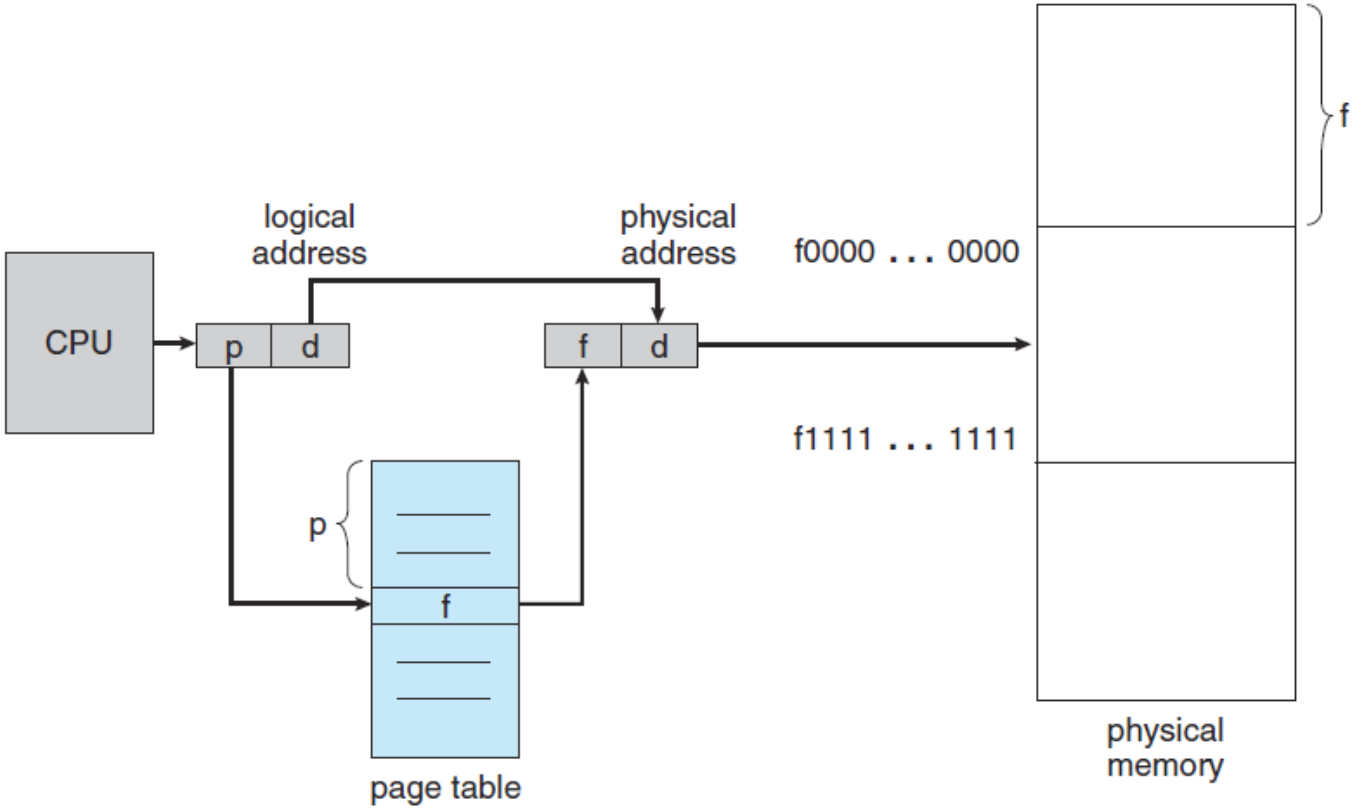
    a) Reduce external fragmentation by compaction
       - ✓ move allocated partitions (blocks) to place all holes together
       - ✓ compaction is possible only if relocation is dynamic (running time address binding)

    b) Permit non-contiguous allocation of logical address space---Paging
       - ✓ Divide physical memory into fixed-sized blocks called frames (the size is power of 2, between 512 (2^9) bytes and 1GB (2^{30}B))
       - ✓ Divide logical memory into blocks of same size called pages
       - ✓ Keep track of all free frames.
       - ✓ To run a program of n-page size, need to find n free frames in the memory and load program
       - ✓ Set up a page table to translate logical addresses to physical addresses

© by Dr. X. Sun

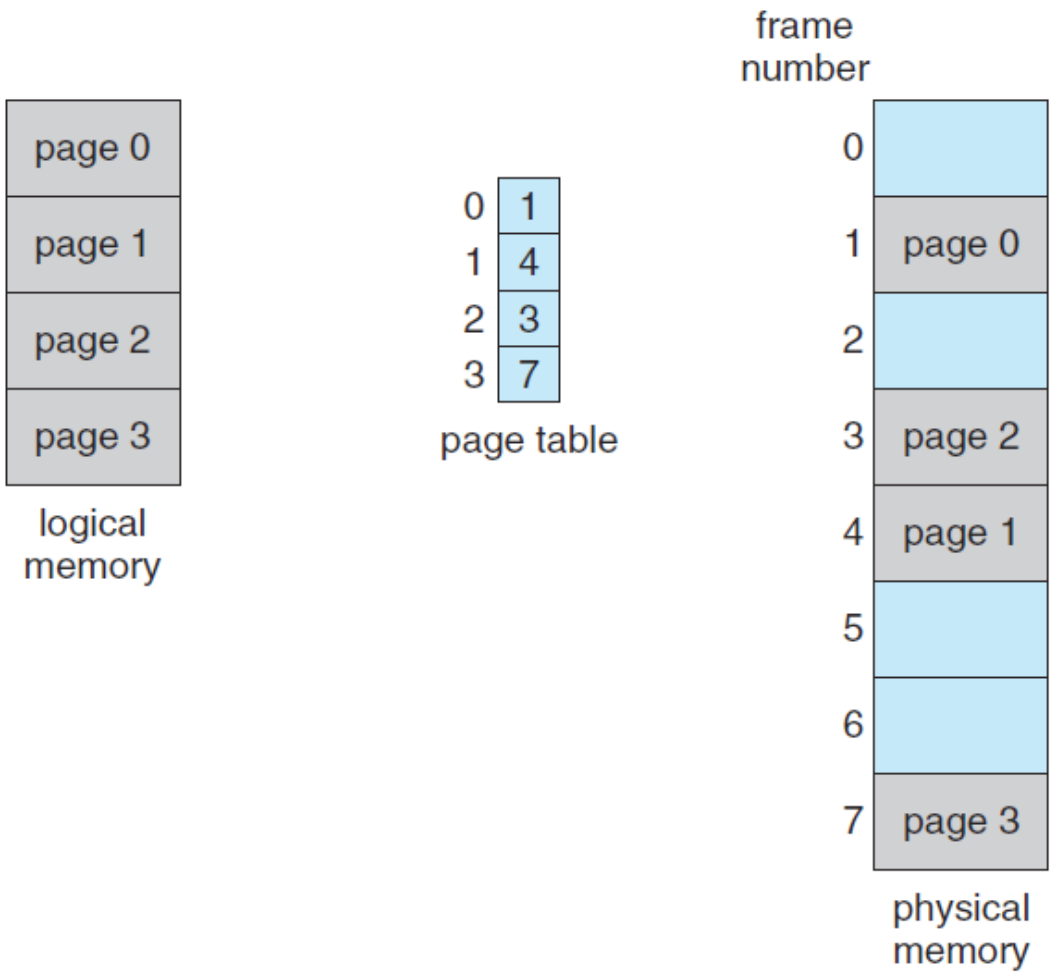
❑ Paging

- ➢ Address generated by CPU is divided into:
  - ✓ Page number (p) – used as an index into a page table which indicates base address of each frame in physical memory
    - Page table: mapping logical to physical addresses
  - ✓ Page offset (d) – combined with base address to define the physical memory address



Address Translation Scheme in Paging

- Example: logical page number → physical frame number



Address Translation Scheme in Paging

- The page size/frame size is defined by the hardware (2^9--2^{30}B).

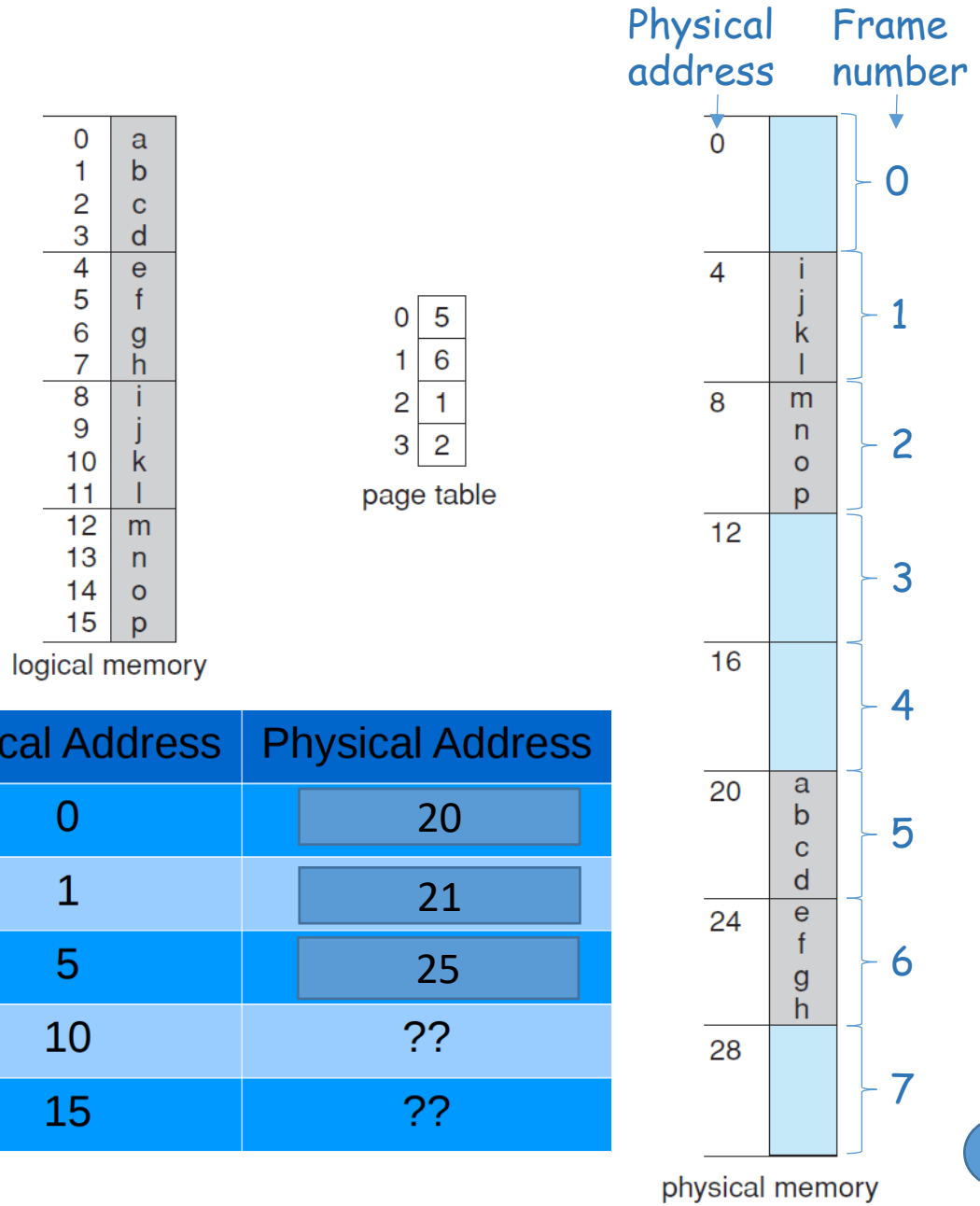
- If the size of the memory address space is 2^m and the size of a page is 2^n, then the high-order $m-n$ bits of a logical address designate the page number, and the n low-order bits designate the page offset.

page number	page offset
p	d
$m-n$	n

© by Dr. X. Sun

Address Translation Scheme in Paging

- If n=2 and m=5, how a logical memory is mapped into a physical memory?
- When we use a paging scheme, we have no external fragmentation--any free frame can be allocated to a process that needs it.
- Question: A 32-bit computer system uses 32-bit to represent a memory address, and the frame size is 4KB, what is the maximum physical memory address that can be addressed?



Logical Address	Physical Address
0	20
1	21
5	25
10	??
15	??

© by Dr. X. Sun

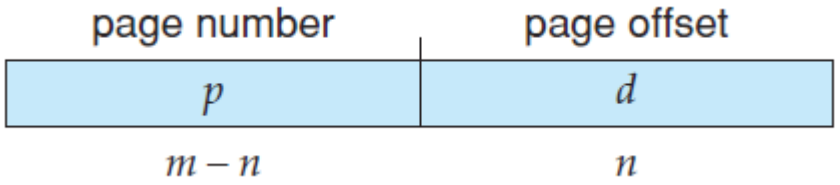
Paging

❑ Paging may incur internal fragmentation

- Internal fragmentation: allocated memory may be slightly larger than requested memory.

- In the paging system, frames are considered as smallest units allocated to processes. If the memory requirements of a process do not happen to coincide with page boundaries, the last frame allocated may not be completely full.
  - ✓ if page size is 2,048 bytes and a process requires 71,681 bytes memory, how many frames are needed?
  - ✓ 71,681/2,048=35 frames + 1 bytes→ the system would allocate 36 frames to the process and the last frame only utilize 1 byte (low memory utilization).

- In order to reduce internal fragmentation loss, we may choose smaller page size; however, a smaller page size (n) incurs a larger size of page table (m-n).



Paging

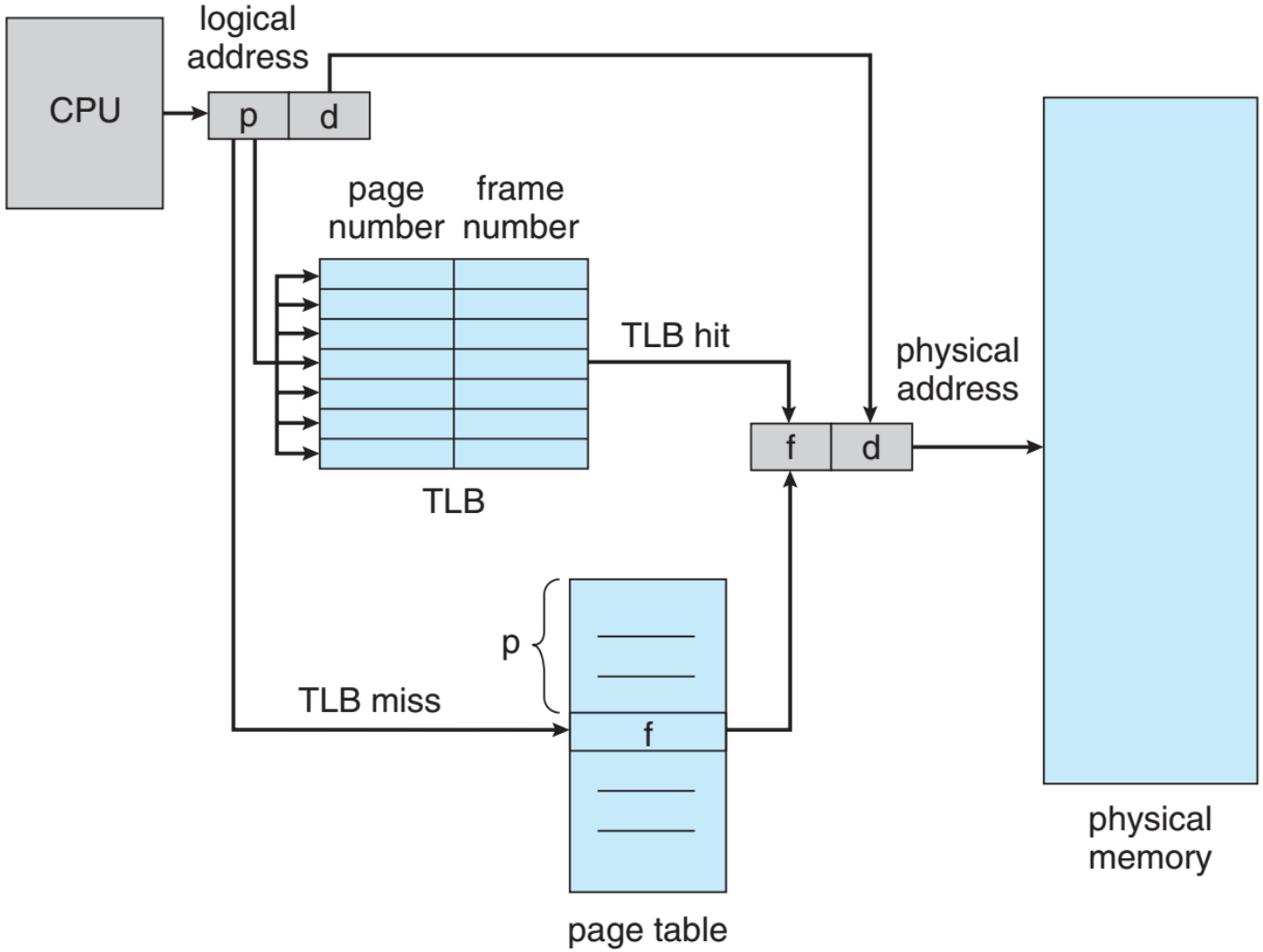
❑ Page table implementation

- Each process has its own page table in the main memory.
- The Page-table base register (PTBR) is used to point to the page table of the running process.
  - ✓ The PTBR is changed on each context switch.
- Every load/store requires two memory accesses
  - ✓ one for accessing page table and one for accessing the data/instruction.
  - ✓ Thus, memory access is slowed by a factor of 2, which is not efficient.
- Solution: use a special, small, fast-lookup hardware cache, called a translation look-aside buffer (TLB), to hold popular page table entries.
  - ✓ access to TLB should be very fast

© by Dr. X. Sun

Page table implementation with TLB

- ✓ The TLB contains only a few page-table entries (typically between 32 and 1,024 entries).
- ✓ When a logical address is generated by the CPU, its page number is presented to the TLB.
  - If the page number is found (know as a TLB hit), its frame number is immediately available and is used to access memory
  - If the page number is not in the TLB (known as a TLB miss), a memory reference to the page table must be made.



Page table implementation with TLB

- Problem: each process has its own page table; once a context switch happens, the OS changes PTBR pointer. How about page table entries cached in TLB?
  - ✓ TLB may include old entries that contain valid virtual addresses but have incorrect or invalid physical addresses left over from the previous process.

- Solution:
  a) TLB must be flushed/erased in each context switch
  b) Each entry in TLB is associated with an address-space identifier (ASID)
     - ✓ ASID uniquely identifies each process to provide address-space protection for that process
     - ✓ When the TLB attempts to resolve virtual page numbers, it ensures that the ASID for the currently running process matches the ASID associated with the virtual page. If the ASIDs do not match, the attempt is treated as a TLB miss.

TLB

ASID	Page number	Frame number

© by Dr. X. Sun

Paging

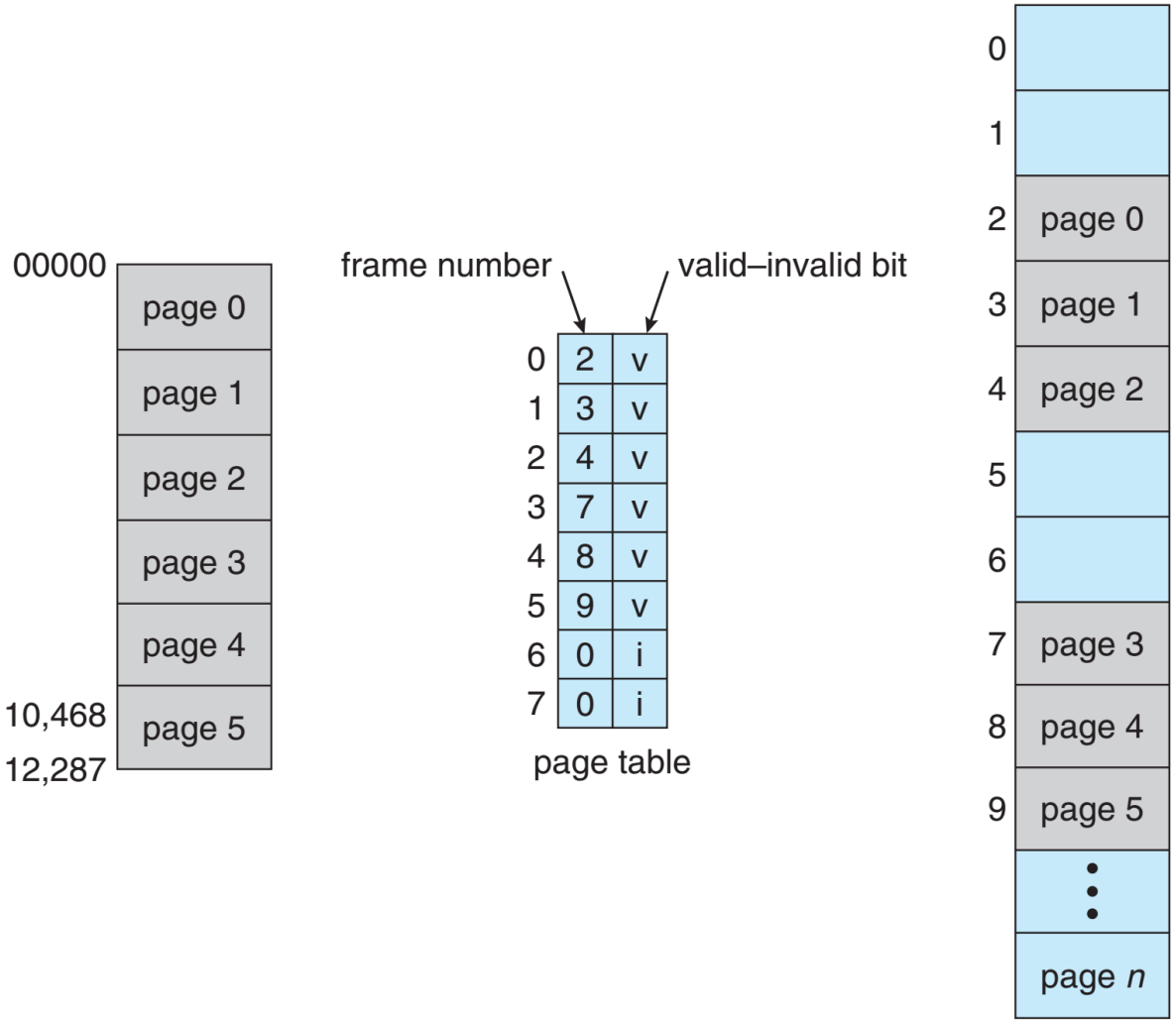
Memory protection

- Memory protection implemented by associating protection bits with each frame in the page table to indicate the frame is
  - ✓ read-only or
  - ✓ read-write access allowed
  - ✓ execute only
  - ✓ etc…
- Also, 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

© by Dr. X. Sun

Memory protection--Example

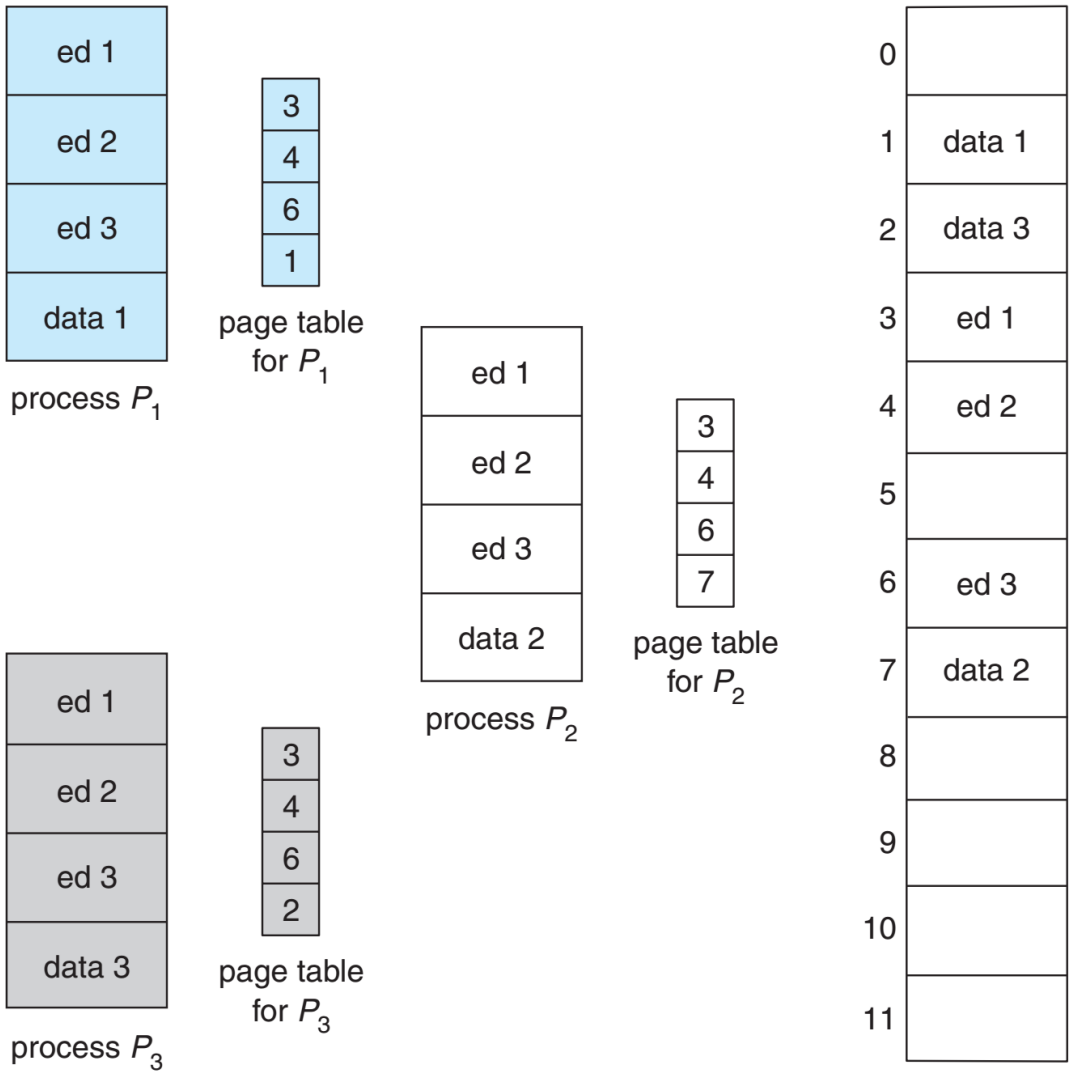
- A system has 14-bit address space
- A process uses addresses 0 – 10,468
- Page size is 2KB
- Accessing addresses on pages 6 and 7 is invalid
- Some systems use a page-table length register (PTLR) to indicate the size of the page table.
  - ✓ Check the value of PTLR to verify that the address is in the valid range for the process.



Paging

Share pages

- Paging allows the possibility of easily sharing code
- One copy of read-only (reentrant) code shared among processes
  - ✓ E.g., text editors, compilers, window systems
- Each process has private data
- Share code can significantly reduce memory usage.



Hierarchical Page Tables

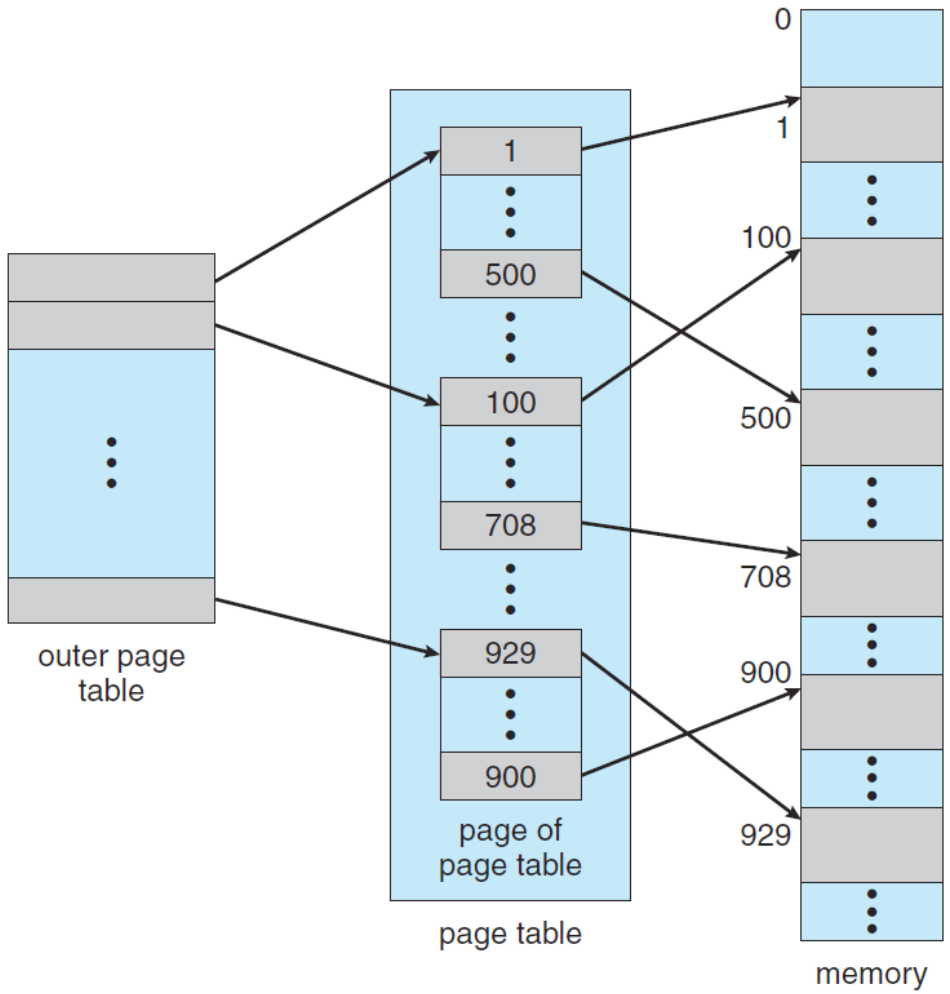
- The size of a page table is large
  - If a system with a 32-bit logical address space and the page size is 4 KB (2^{12}), then a page table may consist of up to $2^{32}/2^{12}=2^{20}$ entries.
  - If each entry needs 4 bytes, then the page table with 2^{20} entries needs 4 MB of physical space.
  - 4MB of memory space per process is a lot.
  - In addition, most of the processes only use a small portion of the address table (i.e., address space).

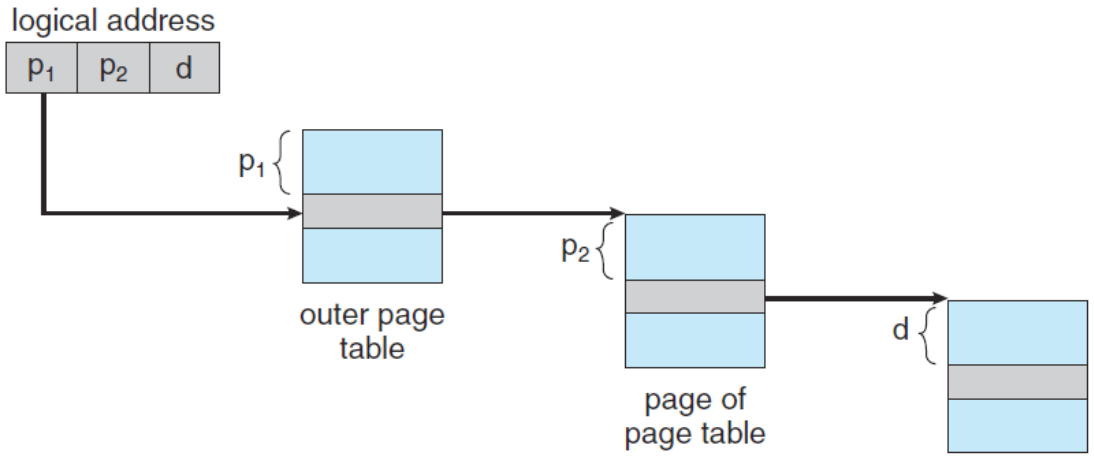
- Two-level page table (paged page-table)
  - One "Page number" column in the page table is further divided into two "Page number" columns, p1 and p2.
  - p1 – index of the entries in the outer page table
  - p2 – index of the entries in the inter page table

page number		page offset
p_i	p_2	d
10	10	12

© by Dr. X. Sun

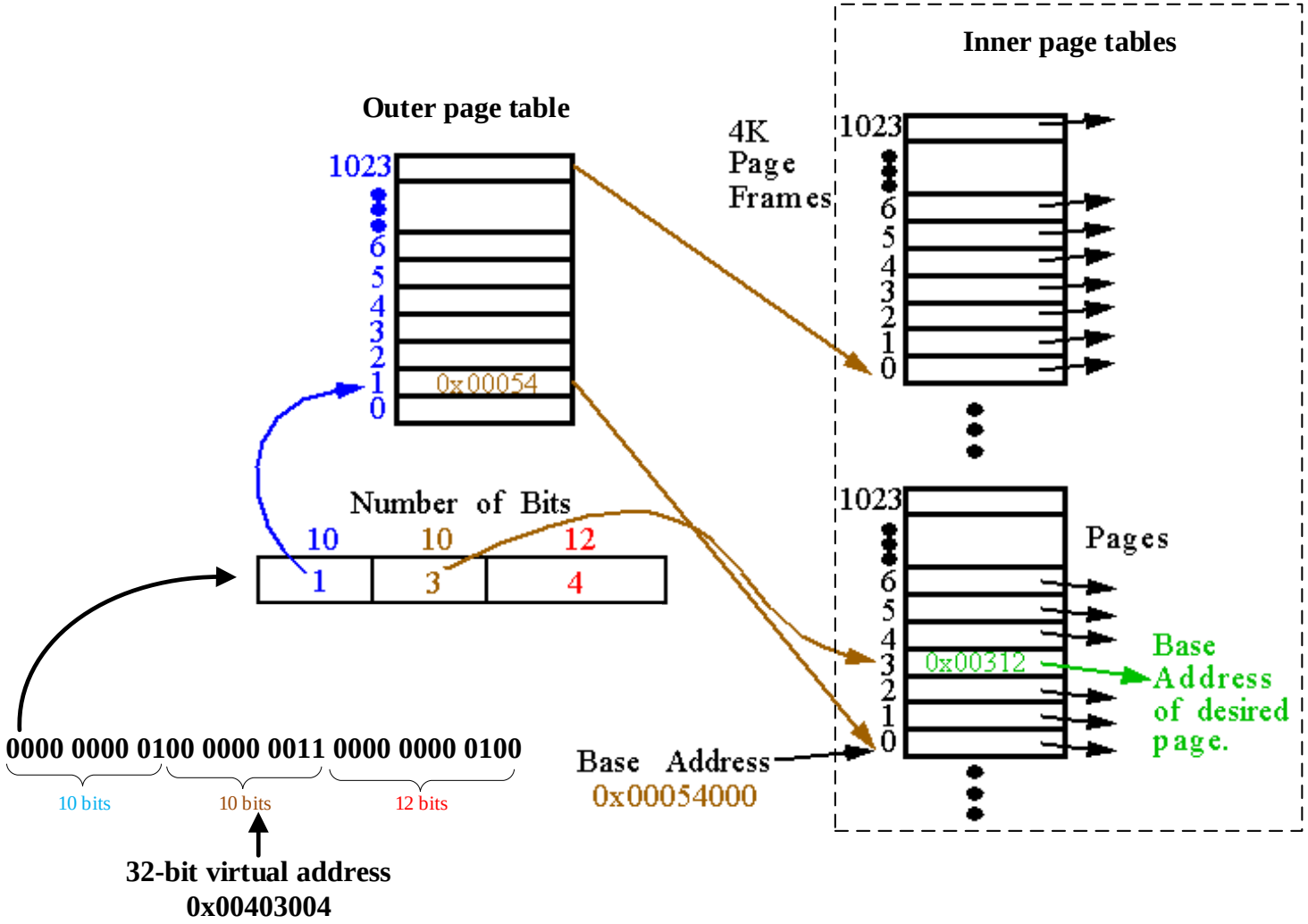
❑ Address-Translation Scheme for Two-Level Paging





© by Dr. X. Sun

Address-Translation Scheme for Two-Level Paging--example



Two-level page table

- Could two-level page table reduce the address space?
  - ✓ An outer page table has 2^{10} entries, each of which needs 4 bytes. Thus, the outer page table requires $4*2^{10}$=4KB
  - ✓ There are 2^{10} inner page tables, each of which has 2^{10} entries. Thus, the inner page tables require $4*2^{10}*2^{10}$=4MB
  - ✓ Recall that: single table page needs 4MB space.

- Two-level page table can reduce since most of inner page tables are NOT needed.
  - ✓ Suppose that a process needs 4MB memory space– that is only one inner page table is enough.
  - ✓ Thus, only one entry in the outer page table is marked as "valid" and associated to the specific inner page table. The rest of the entries in the outer page table are marked as "invalid" and are not associated to inner tables

Index	inner table number	valid/invalid bit
0	1	v
1	2	i
2	3	i
3	7	i
…	…	…

Index	frame number
0	11
1	12
2	13
3	7
…	…
1023	1442

  - ✓ Thus, only one outer page table and one inner page table are needed, which require 8KB!
  - ✓ Note that two-level page table requires three memory accesses for each load/store (if TLB is not considered). So, there is a tradeoff between saving memory space and accelerating address translation speed.

© by Dr. X. Sun

More levels in hierarchical paging

- Two level paging is not appropriate for 64-bit machines
  - Now a system has a 64-bit logical address space. If the page size is 4 KB (2^{12}), then a page table may consist of up to $2^{64}/2^{12}=2^{52}$ entries.
  - If one level paging is used and each entry needs 4 bytes, then the page table with 2^{52} entries needs 16 PB
  - If two level paging is used, the outer page table still needs $4*2^{42}$=16TB

outer page	inner page	offset
p_1	p_2	d
42	10	12

- What if three level paging?

2nd outer page	outer page	inner page	offset
p_1	p_2	p_3	d
32	10	10	12

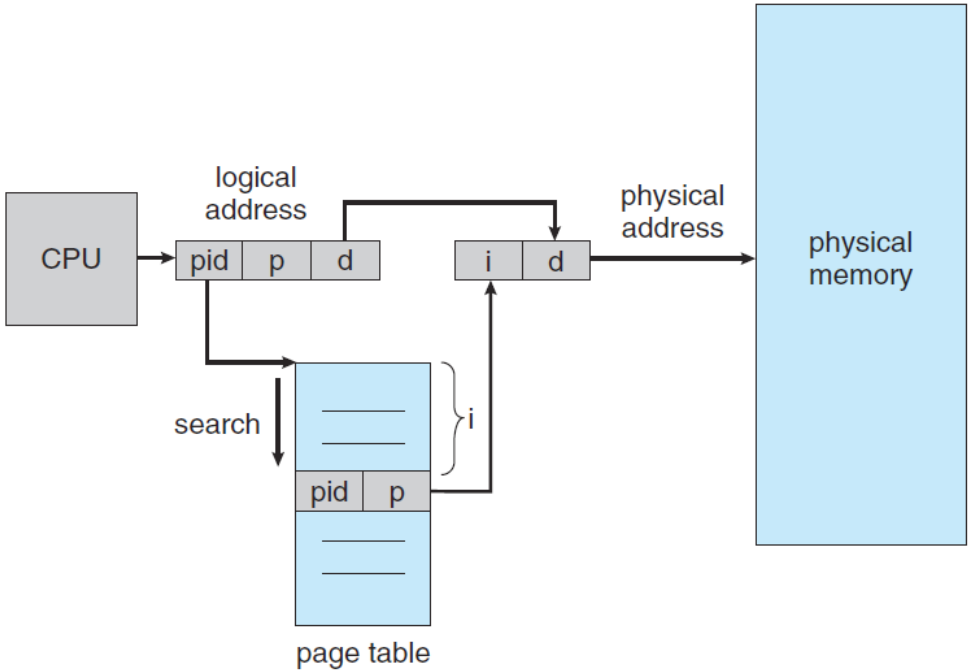
  - The 2nd outer page table still needs $4*2^{32}$=16GB

- We can go on increasing the number of paging levels to reduce the memory space for tables
  - but each level requires a memory access
  - n-level paging requires n+1 memory accesses for every load/store!
  - need another page table implementation...

© by Dr. X. Sun

❑ Inverted Page Table

- An inverted page table has one entry for each frame/page of memory.
  - ✓ If the physical memory size is 4GB and the frame/page size is 4KB, the number of frames/pages =4GB/4KB=2^{20}. That is, there are 2^{20} entries in the inverted page table. If each entry needs 4 bytes, then the inverted page table requires 4 MB memory space.
  - ✓ If the physical memory size is 8GB/16GB, the inverted page table requires 8MB/16MB memory space.
  - ✓ The size of inverted page table is determined by the physical memory size, NOT the total logical address space.

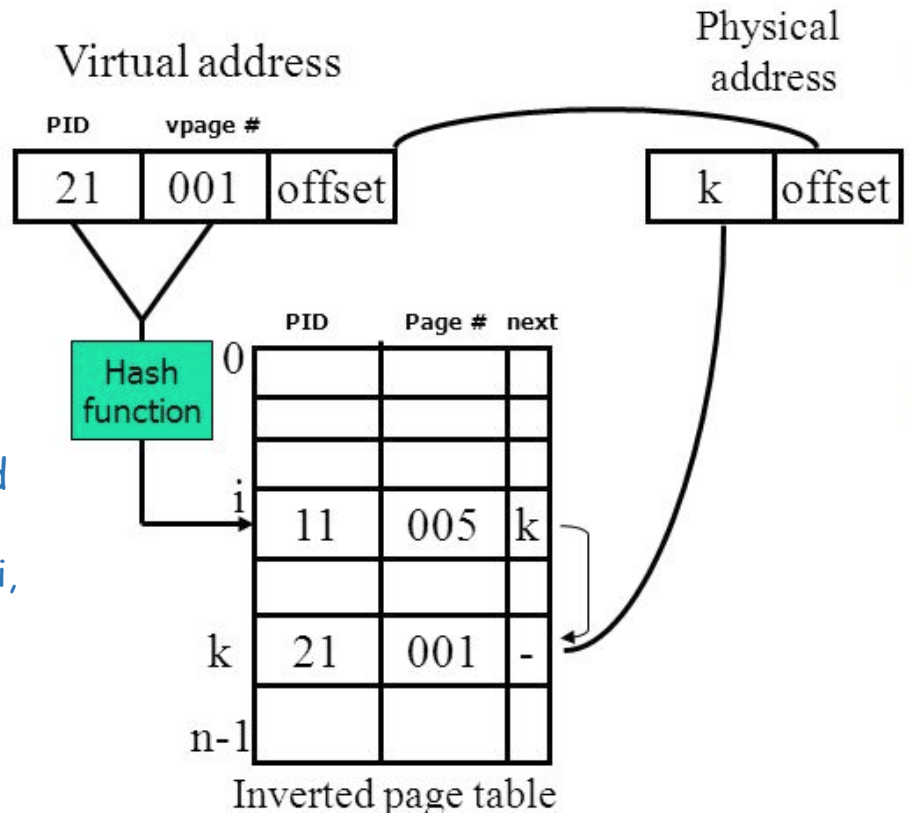


- Each logical address in the system consists of a triple <process-id, page-number, offset>.
- Each inverted page-table entry is a pair <process-id, page-number>
- For any address translation request, inverted page table is searched for a match <pid, p>.
- If a match is found—say, at entry i—then the physical address <i, offset> is generated.
- One inverted page table is shared among processes.
- Problem: long searching time.

© by Dr. X. Sun

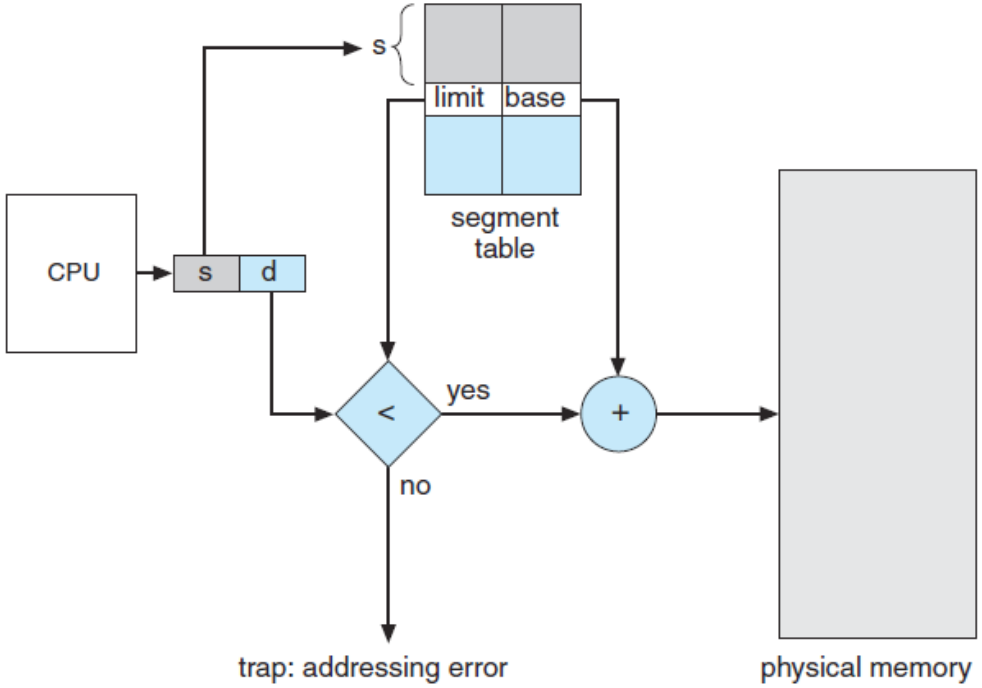
❑ Hashed inverted page table

- Basic idea: hash PID and page number, and the generated value is the index/address of the inverted page table page.
- Each entry in the new inverted page table comprises of a triple <PID, Page #, NEXT>
  - ✓ Next: pointer points to the next entry of the table.
  - ✓ The reason for adding "Next" is that different <PID, page#> may hash into the same value.
- Address translation procedure
  1. Hash PID and page number, and find the related entry
  2. The PID and page number in the logical address are compared with the 1st field and 2nd field of the entry, respectively.
     - If a match is found—say, at entry i—then the physical address <i, offset> is generated.
     - Otherwise, go to the next entry (which is indicated by the 3rd field of the entry) .



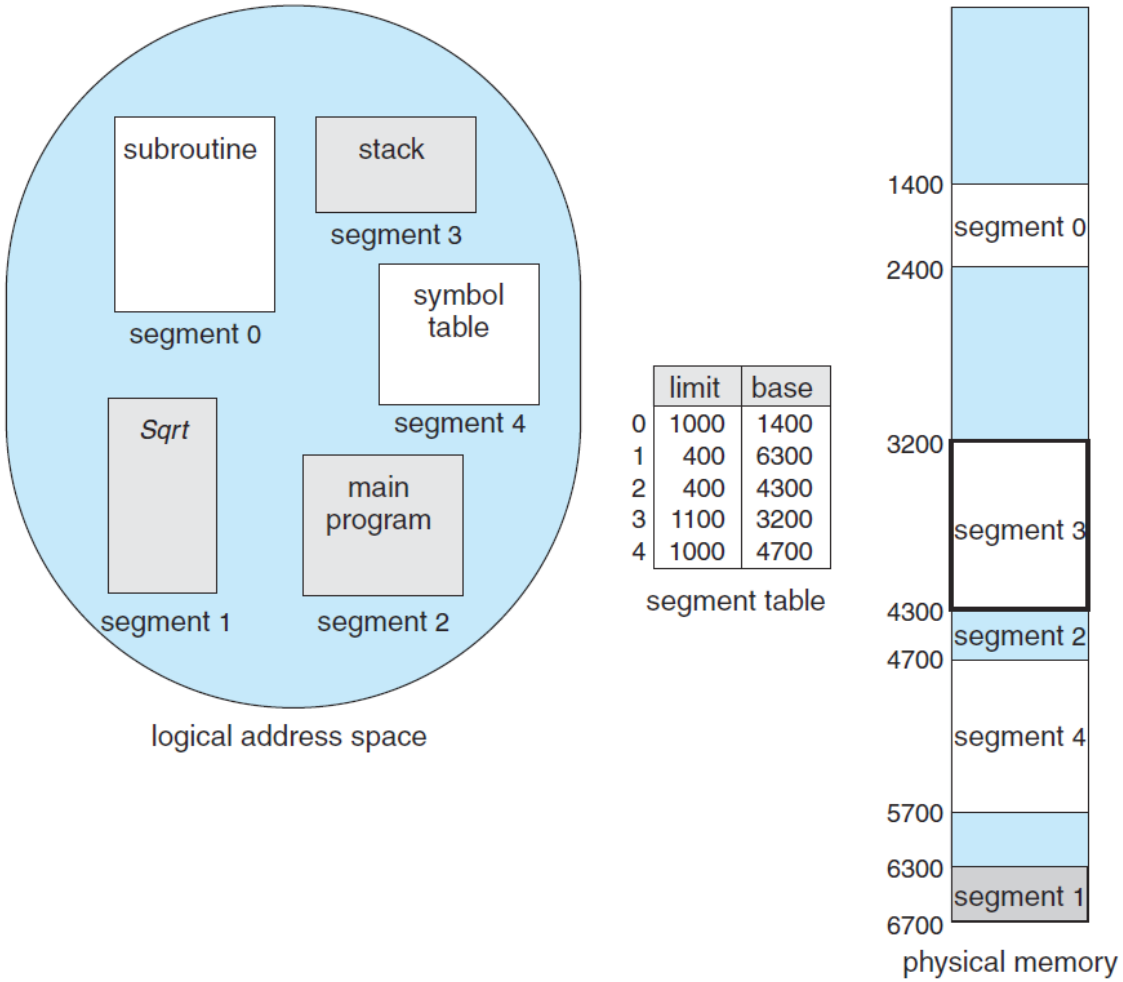
Segmentation

- Memory-management scheme that supports user view of memory
- A process is considered as a collection of segments
- Logical address is described as a pair: <segment number (s), offset (d)>
- Each process has a segment table (similar to the page table), which maps a logical address into a physical address.
  - Each entry in has a segment base and a segment limit
    - ✓ The segment base contains the starting physical address where the segment resides in memory
    - ✓ The segment limit specifies the length of the segment



Segmentation

❑ Example of segmentation



❑ Comparison between paging and segmentation

- A page is of fixed block size, while a segment is of variable size.
- In paging, the user specified address is converted into a page number and offset. In segmentation, the user specifies each address by two quantities a segment number and the offset.
- In paging, the hardware decides the page size. In segmentation, the segment size is specified by the user.

© by Dr. X. Sun

ECE437/CS481

M07B: MEMORY MANAGEMENT

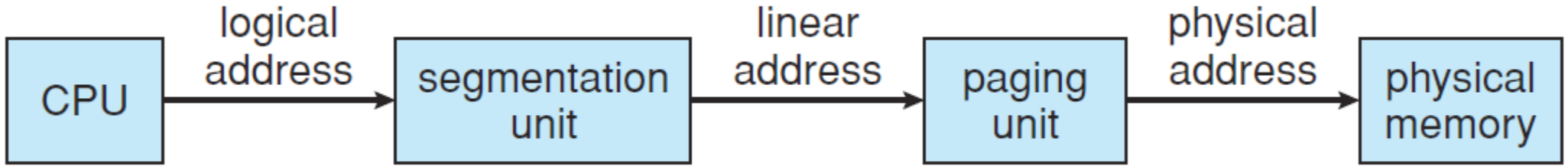
CHAPTER 8.7 & 9.8

Xiang Sun

The University of New Mexico

IA-32 architecture

- Memory management in IA-32 systems is divided into two components—segmentation and paging



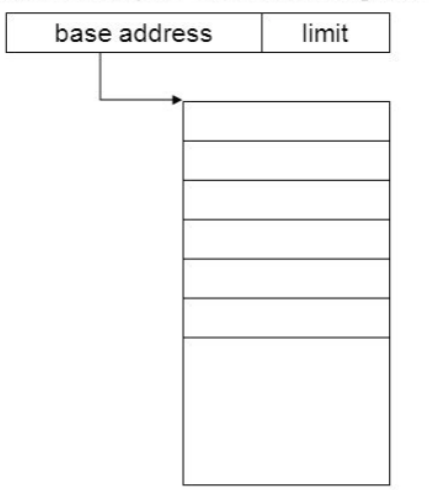
  - The CPU generates logical addresses, which are given to the segmentation unit.
  - The segmentation unit produces a linear address for each logical address.
  - The linear address is then given to the paging unit, which in turn generates the physical address in main memory.

© by Dr. X. Sun

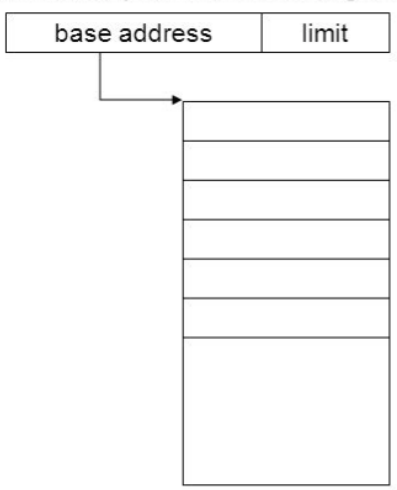
IA-32 architecture

❑ IA-32 Segmentation (logical address → linear address)

- The logical address space of a process is divided into two partitions
  - ✓ Private partition
    - consists of up to 8K segments that are private to a process
    - information about the private partition is kept in the local descriptor table (LDT)
  - ✓ Global partition
    - consists of up to 8K segments that are shared among all the processes
    - information about the global partition is kept in the global descriptor table (GDT)
  - ✓ Each entry in the LDT/GDT consists of an 8-byte segment descriptor with information about a particular segment, including the base address and limit of the segment



Global Descriptor Table

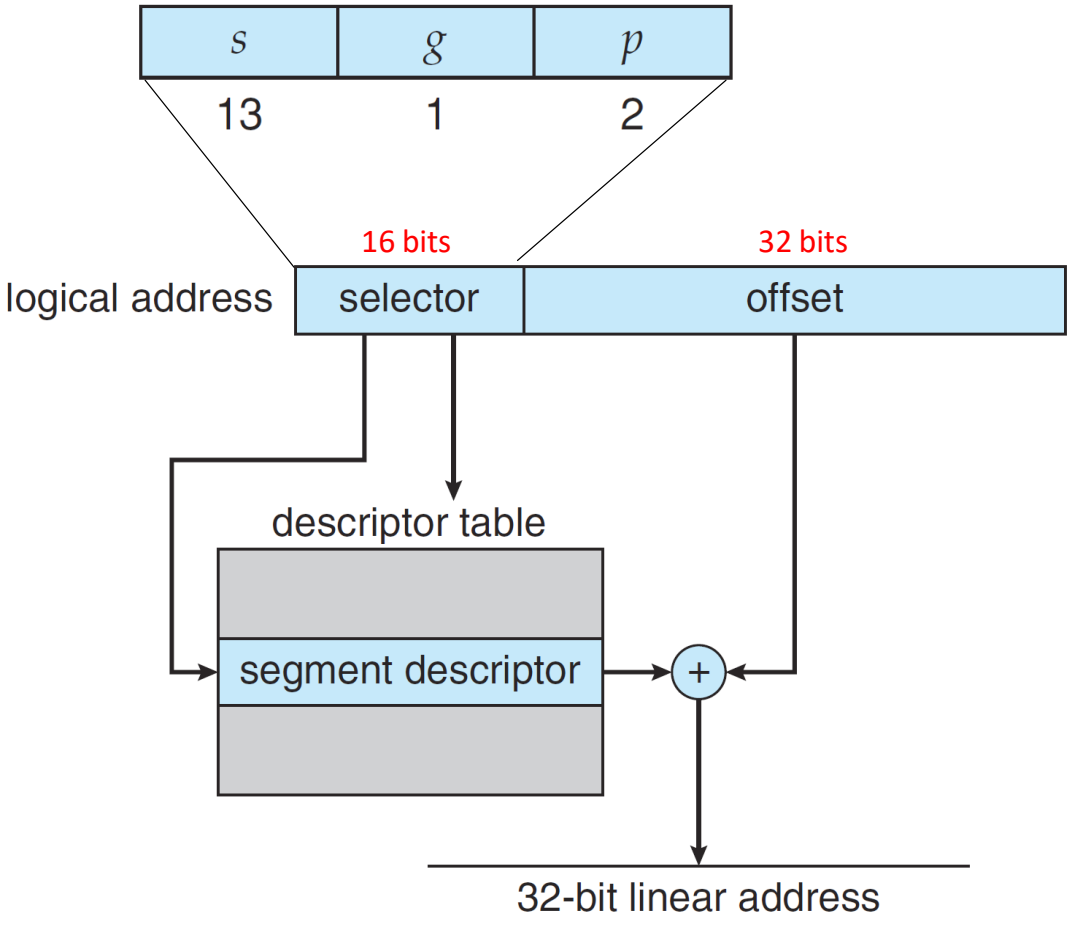


Local Descriptor Table

IA-32 architecture

❑ IA-32 Segmentation

- ➢ logical address is a pair <selector, offset>, where the selector is a 16-bit number
  - ✓ s designates the segment number
  - ✓ g indicates whether the segment is in the GDT or LDT
  - ✓ p deals with protection
  - ✓ The segment number in the logical address points to the appropriate entry (segment descriptor) in the LDT/GDT
  - ✓ The base and limit information in the segment descriptor are used to generate a 32-bit linear address



IA-32 architecture

❑ IA-32 Paging

- The IA-32 architecture allows a page size of either 4 KB or 4 MB
- For 4KB page size setup, IA-32 uses a two-level paging scheme

page number		page offset
p_1	p_2	d
10	10	12

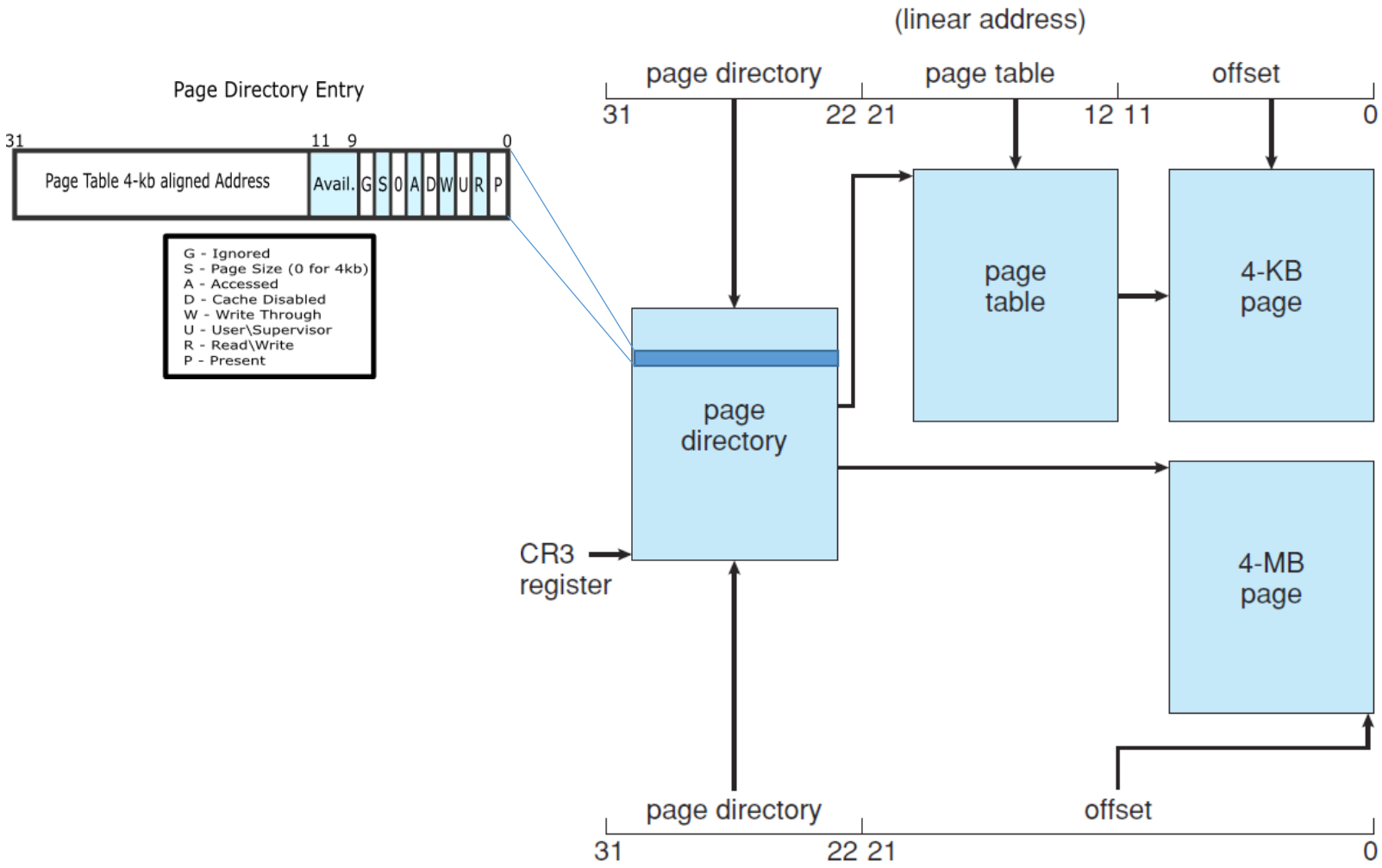
  - ✓ 10 bits (p_1) are used to index the outer page table (which is termed as page directory in IA-32)
  - ✓ 10 bits (p_2) are used to index the inner page table
  - ✓ 12 bits are the offset for a 4KB page

- For 4MB page size setup, IA-32 uses a one-level paging scheme
  - ✓ 10 bits (p_1) are used to index the page table (page directory)
  - ✓ 22 bits are the offset for a 4MB page

IA-32 architecture

IA-32 Paging

- The CR3 register (page-table base register) points to the page directory for the current process
- One entry in the page directory is the page size (PS) flag (8th bit).
  - If PS=1, the page directory points directly to the 4-MB page frame



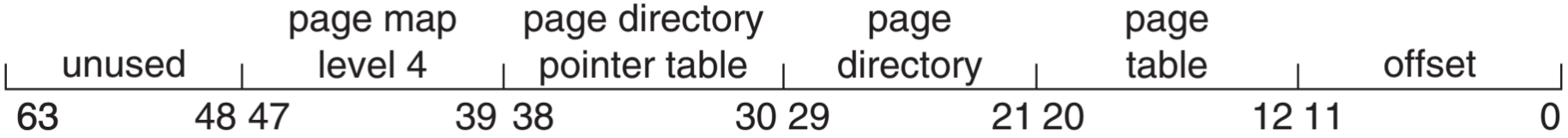
X86-64 architecture

- Intel developed IA-64 architecture--Itanium
  - The IA-64 Architecture was NOT widely adopted since IA-64 does not compatible with IA-32

- AMD also developed similar architecture, called X86-64 (AMD64) architecture
  - The 86-64 architecture offers several backward compatibility modes to execute IA-32 software. Intel adopted the architecture after the failure of Itanium.

- Support for a 64-bit address space yields 2^{64} addressable memory space.

- In practice, fewer than 64 bits are used for address representation in current designs. The x86-64 architecture currently provides a 48-bit logical address with support for page sizes of 4 KB, 2 MB, or 1 GB using four levels of paging hierarchy.



© by Dr. X. Sun

Kernel Memory Management

- Kernel memory is treated differently from user memory (e.g., paging scheme)
  - Different from the user space memory requests, the kernel memory requests
    - ✓ may high-frequency, small allocations (e.g., for kernel data structures, process control blocks, etc.) and large continuous blocks for certain operations (like I/O buffers, network packets, etc.).
    - ✓ need to be fast and low-latency.
  - Two methods to manage free kernel memory
    - ✓ Buddy system
    - ✓ Slab allocation

© by Dr. X. Sun

Kernel Memory Management

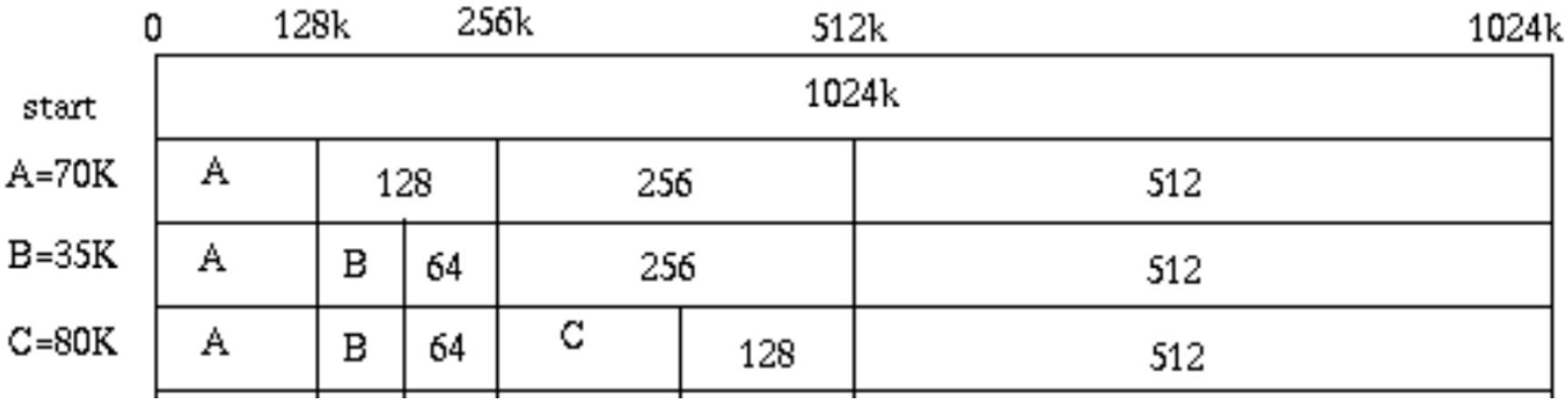
- Kernel memory allocation—Buddy system
  - Basic idea: the size of partition/page can be varied within a range
  - Used for UNIX kernel memory allocation
  - Detail metric
    - ✓ The size of a memory block could be 2^k, where $L \leq k \leq U$.
    - ✓ 2^L is the smallest block that can be allocated.
    - ✓ 2^U is the largest block that can be allocated.
    - ✓ Once there is an incoming memory request, a suitable block size would be searched by the order of 2^U … 2^L

© by Dr. X. Sun

Kernel Memory Management

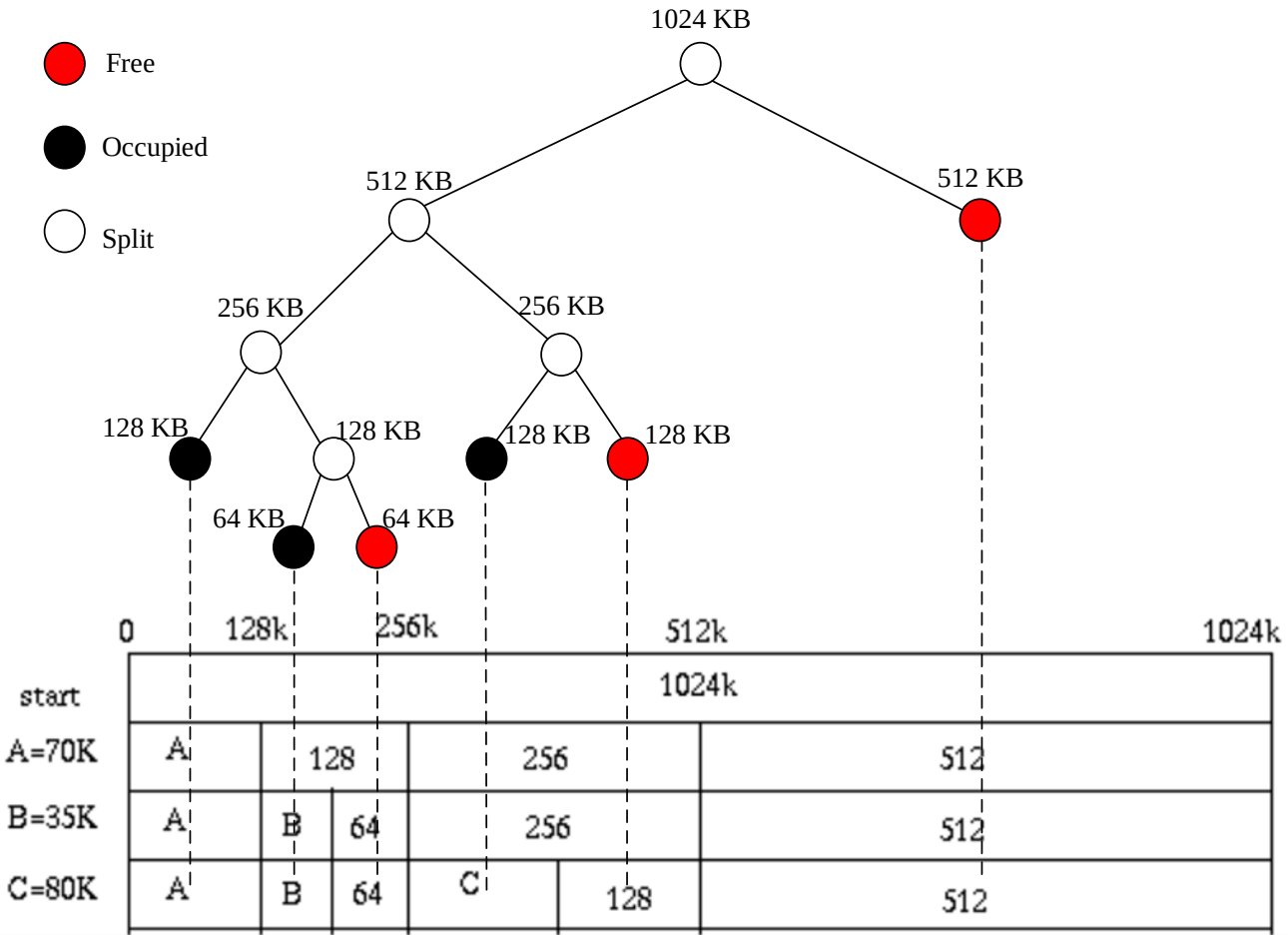
❑ Kernel memory allocation—Buddy system

- Assume that the free kernel memory space is 1024 KB.
- There are three incoming memory requests, i.e., A, B, and C, and the requested memory size is 70 KB, 35 KB, and 80 KB, respectively. The kernel will allocate the memory to the requests sequentially.
- The memory space is divided into two buddies, each 512 KB in size.
- One of the buddies is further divided into to buddies, each 256 in size.
- The process continues until a smallest buddy (which can satisfy the requirement of the request) is found.



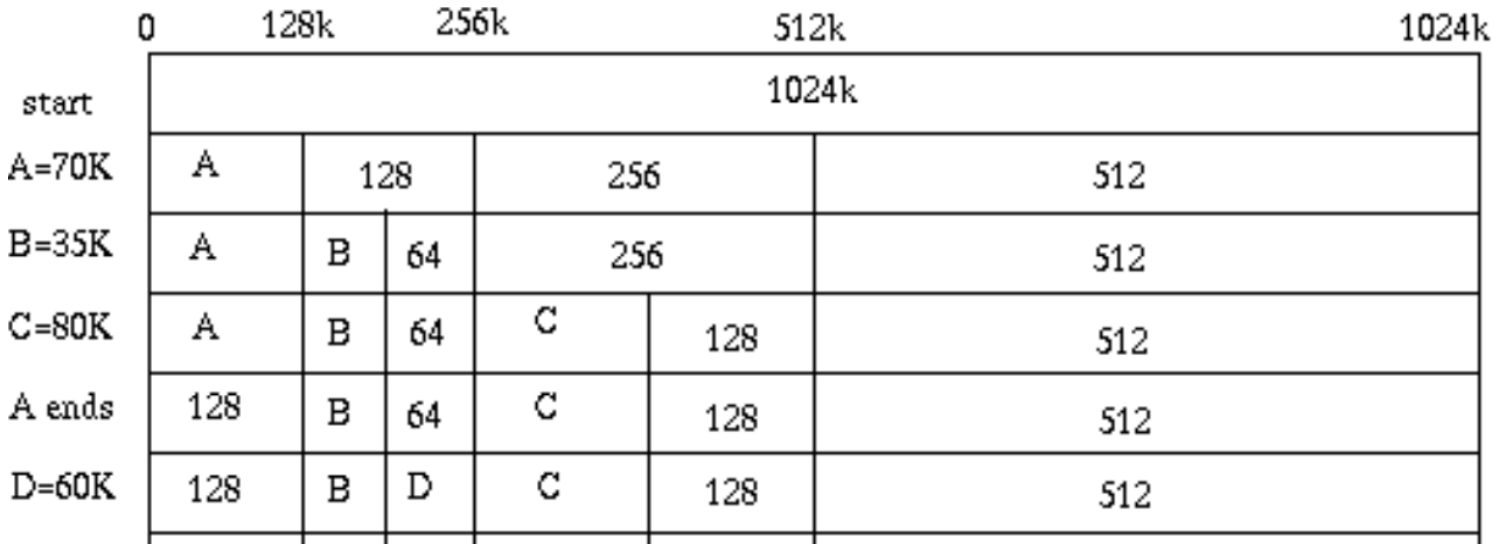
Kernel memory allocation—Buddy system

- The buddy system can be represented as a binary tree.
  - ✓ Each node represents a buddy.
  - ✓ The nodes in the same level have the same memory size.
  - ✓ The memory size of a child node = the memory size of its parent node/2.
  - ✓ Each branch ends up to a free/occupied node.



Kernel Memory Management

- Kernel memory allocation—Buddy system
  - What if A ends, and then a new incoming memory request D (with memory size 60 KB) shows up?
  - What if D=100 KB?



Kernel Memory Management

❑ Kernel memory allocation—Buddy system

- If occupied buddies are freed, the adjacent buddies can be combined together to form larger buddies----coalescing.

	0	128k		256k		512k	1024k
start	1024k						
A=70K	A	128		256		512	
B=35K	A	B	64	256		512	
C=80K	A	B	64	C	128	512	
A ends	128	B	64	C	128	512	
D=60K	128	B	D	C	128	512	
B ends	128	64	D	C	128	512	
D ends	256			C	128	512	
C ends	512					512	
end	1024k						

© by Dr. X. Sun

Kernel Memory Management

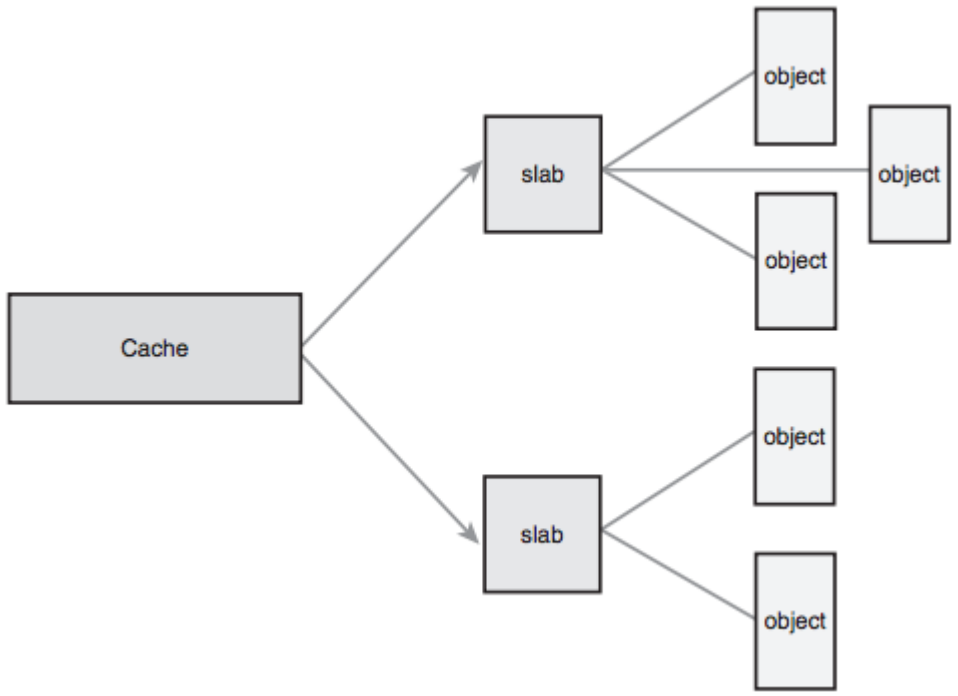
❑ Kernel memory allocation—Slab allocation

- Kernel modules and drivers often need to allocate temporary storage for non-persistent structures/objects (e.g., inode, PCBs, and mutex locks), which are uniform in size and are allocated and deallocated many times.
- Frequent allocation and deallocation is not efficient and may result in external fragmentation.
- Solution: frequently used data structures/objects tend to be allocated and freed often, so cache them. We need a global kernel caching allocator to manage individual caches.
- Slab allocator is design for the global kernel caching allocator.

© by Dr. X. Sun

Kernel memory allocation—Slab allocation

- Three entities are defined in the Slab layer.
  - ✓ Cache: a cache stores the data structures with the same type.
    - For example, one cache is for all the PCBs, whereas another cache is for inode.
  - ✓ Slab: a cache is divided into multiple slabs. Each slab comprises one or more contiguous physical memory frames.
  - ✓ Object: each slab contains a number of objects, which are the data structures being cached.
  - ✓ Each slab is in one of three states
    - Full slab: all objects in a slab are allocated.
    - Empty slab: all objects in a slab are free.
    - Partial slab: some objects in a slab are free.



ECE437/CS481

M07C: VIRTUAL MEMORY

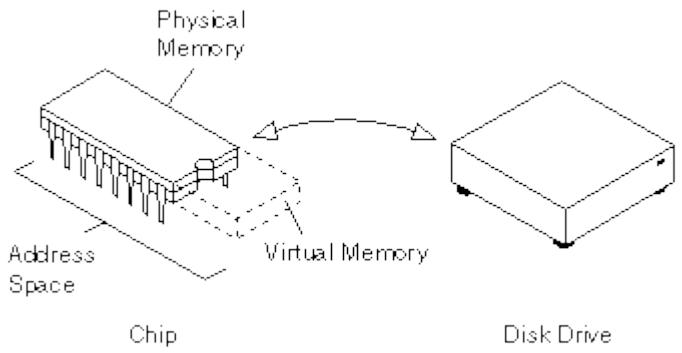
CHAPTER 9.1-9.4

Xiang Sun

The University of New Mexico

Virtual Memory

- Paging forms a foundation of virtual memory
  - ✓ Paging allows noncontiguous memory space allocation



- Virtual memory is more than paging
  - ✓ Virtual memory is a technique to allow the execution of processes not be completed in the main memory. That is, secondary memory (e.g., hard disk) can be used as a part of main memory.
  - ✓ Virtual memory technology frees programmers from concern over memory storage limitations.
  - ✓ Virtual memory can be implemented via Demand Paging.

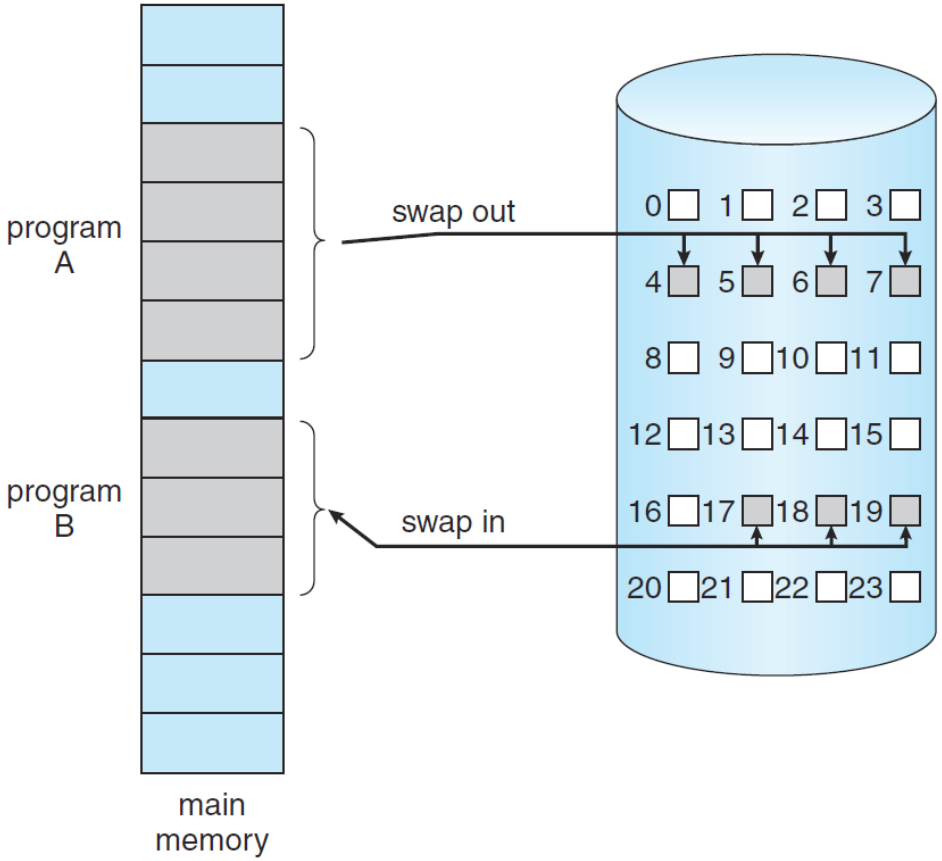
© by Dr. X. Sun

Virtual Memory

❑ Demand Paging

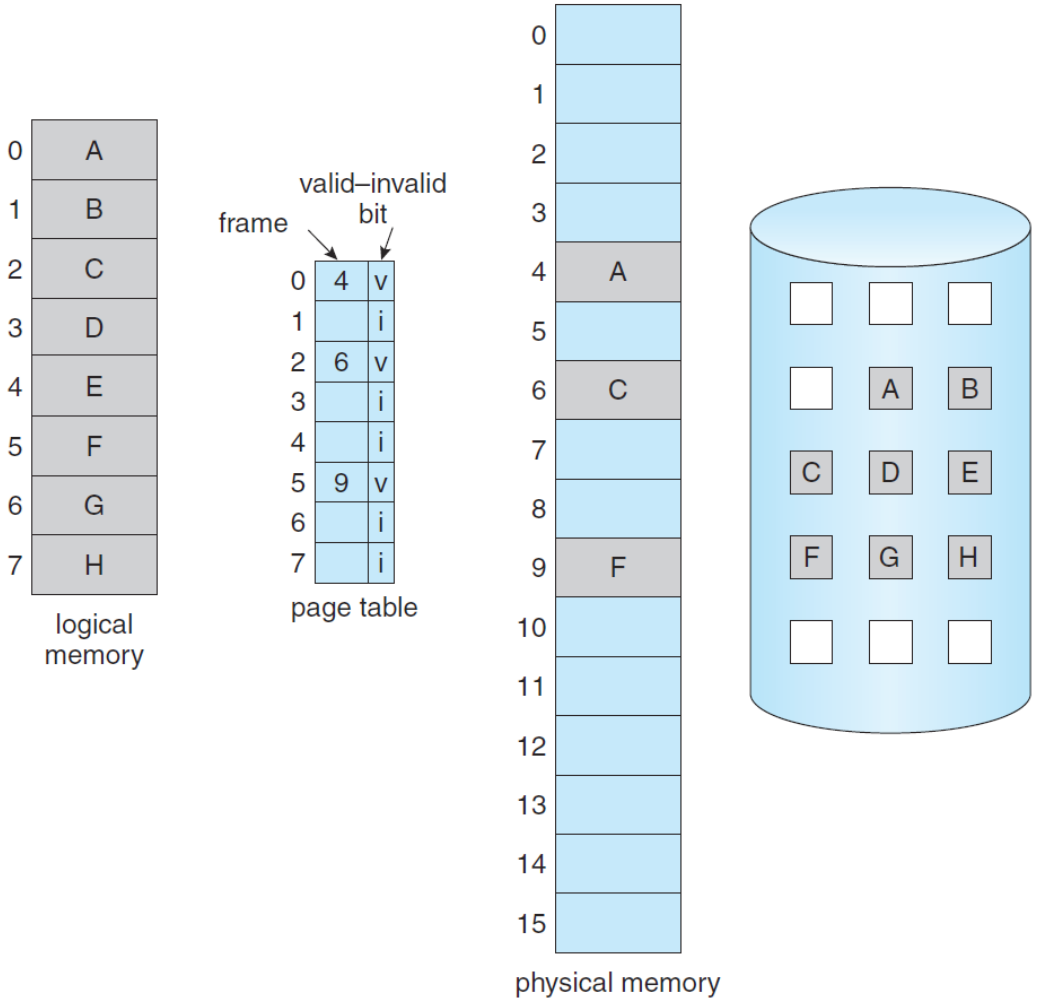
- Recall that: the pages can be swapped in/out between the main memory and disk.
  - ✓ For example, when the memory fills up, allocating a page in memory requires some other page to be swapped out from memory
  - ✓ The unit for swapping in/out is page

- Demand Paging: when a page is referenced (e.g., a page is needed by the CPU), and the page is NOT in the main memory, then the page is swapped into the main memory.



Demand Paging

- Demand paging requires hardware support to distinguish between pages in memory and those in disk.
  - ✓ Recall that: in a page table, there is a valid/invalid bit to check if the referenced page is a legal page (w.r.t. the running process) or not.
  - ✓ It can be used to identify the location of the referenced page.
    - v → in-memory
    - i → not-in-memory

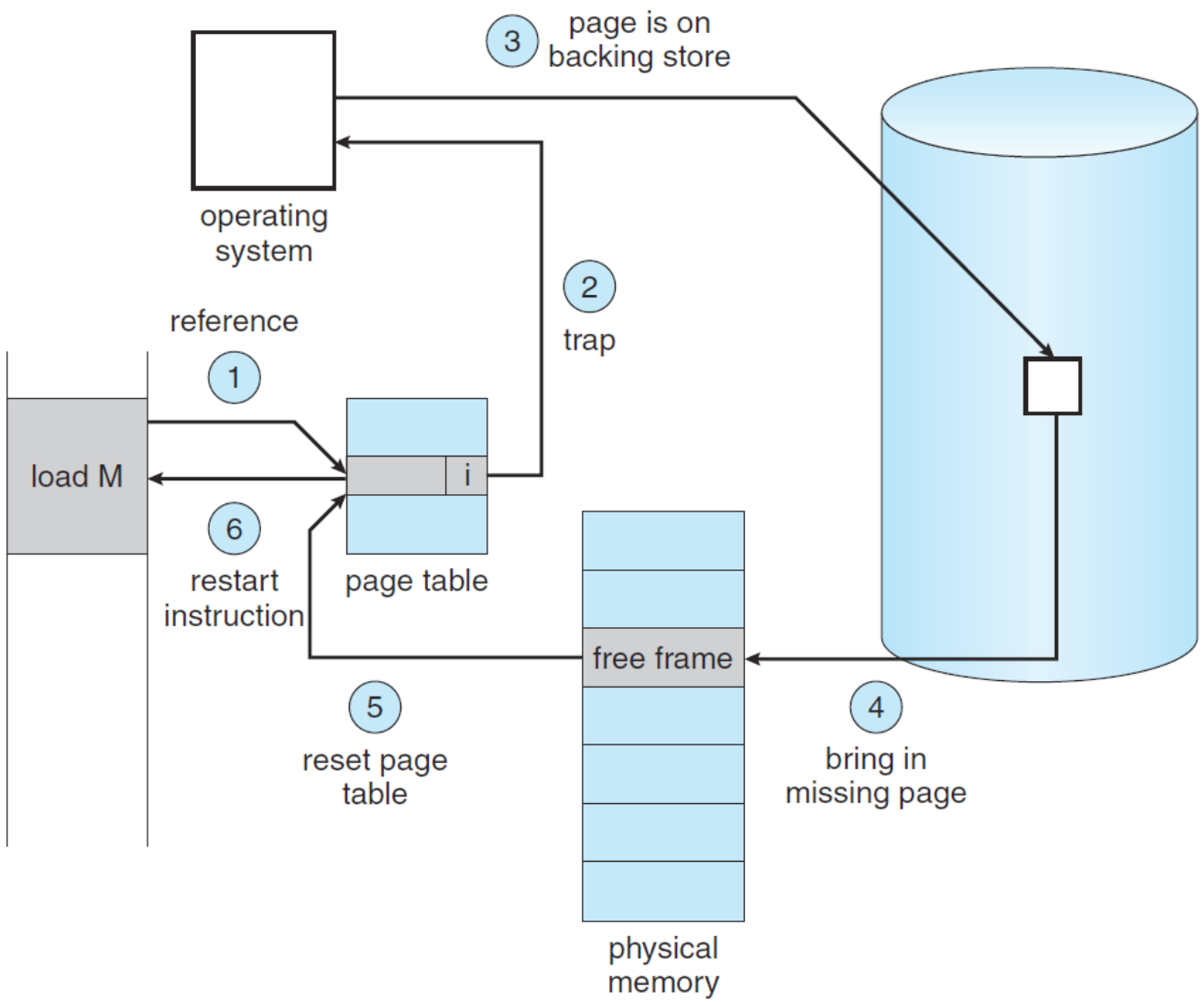


- During the address translation, if the valid/invalid bit is "i", a page fault will be raised.
  - ✓ A page fault is a type of exception raised by MMU
  - ✓ After raising the page fault, the kernel (interrupt handler) will deal with the page fault.

Virtual Memory

❑ Procedures in handling page fault

1. The page is looked up in the page table to determine if the referenced page is valid or not (hardware).
2. If the page is valid, the CPU can access the related frame in the main memory. If the page is not valid, a page fault is generated.
3. Find the referenced page in the hard disk.
4. Find a free frame in the main memory (may need to free up a page). Schedule a disk operation to read the referenced page into the newly allocated frame.
5. When the frame is filled, modify the page table.
6. Restart the instruction that was interrupted.



Effective access time (EAT) in demand paging

- EAT: the average time of accessing a memory page.
- Assume that
  1) memory access time = 200 ns (nanoseconds)
  2) average page-fault service time = 8 ms (milliseconds)
  3) probability of having a page fault is *p*—page-fault rate

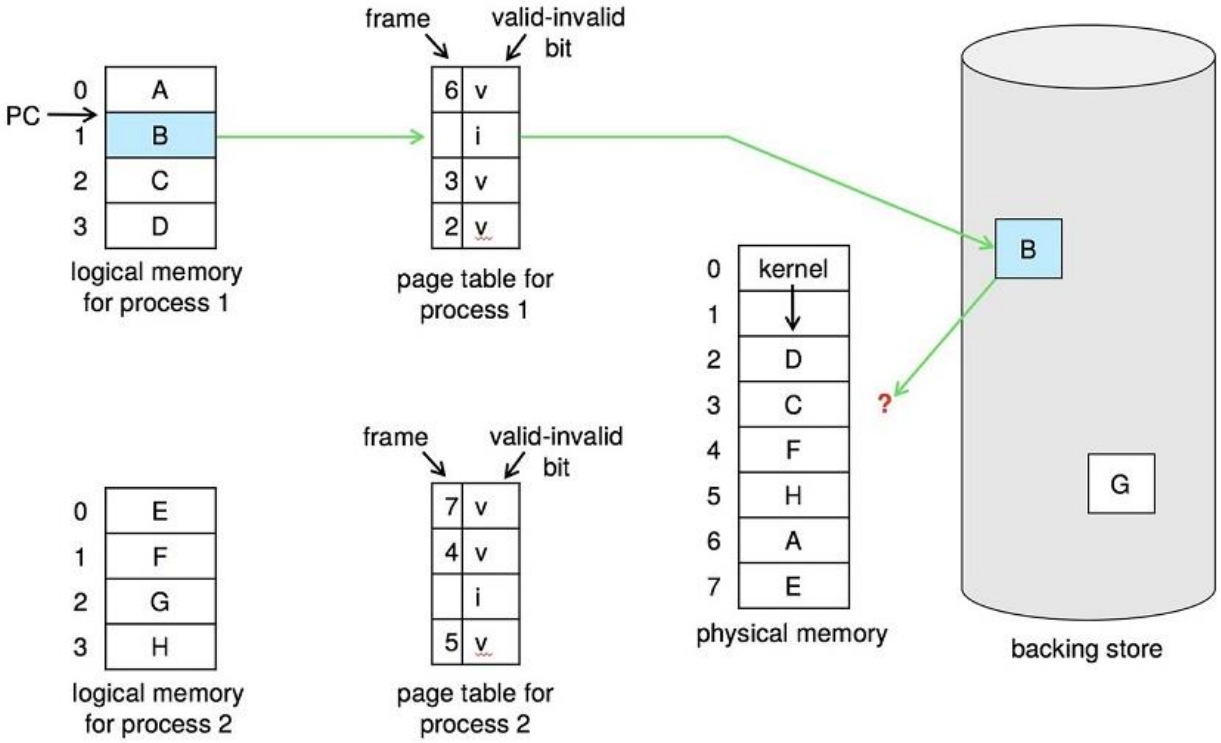
- EAT = 200ns × 2 + (8ms+200ns × 2)×p
  =(400 + 8,000,400×p) ns

- If p=1/1,000,
  EAT=400+8,000.4=8,400.4 ns ≈ 8.4 μs (microsecond)
  This is a slowdown by a factor of 21!!

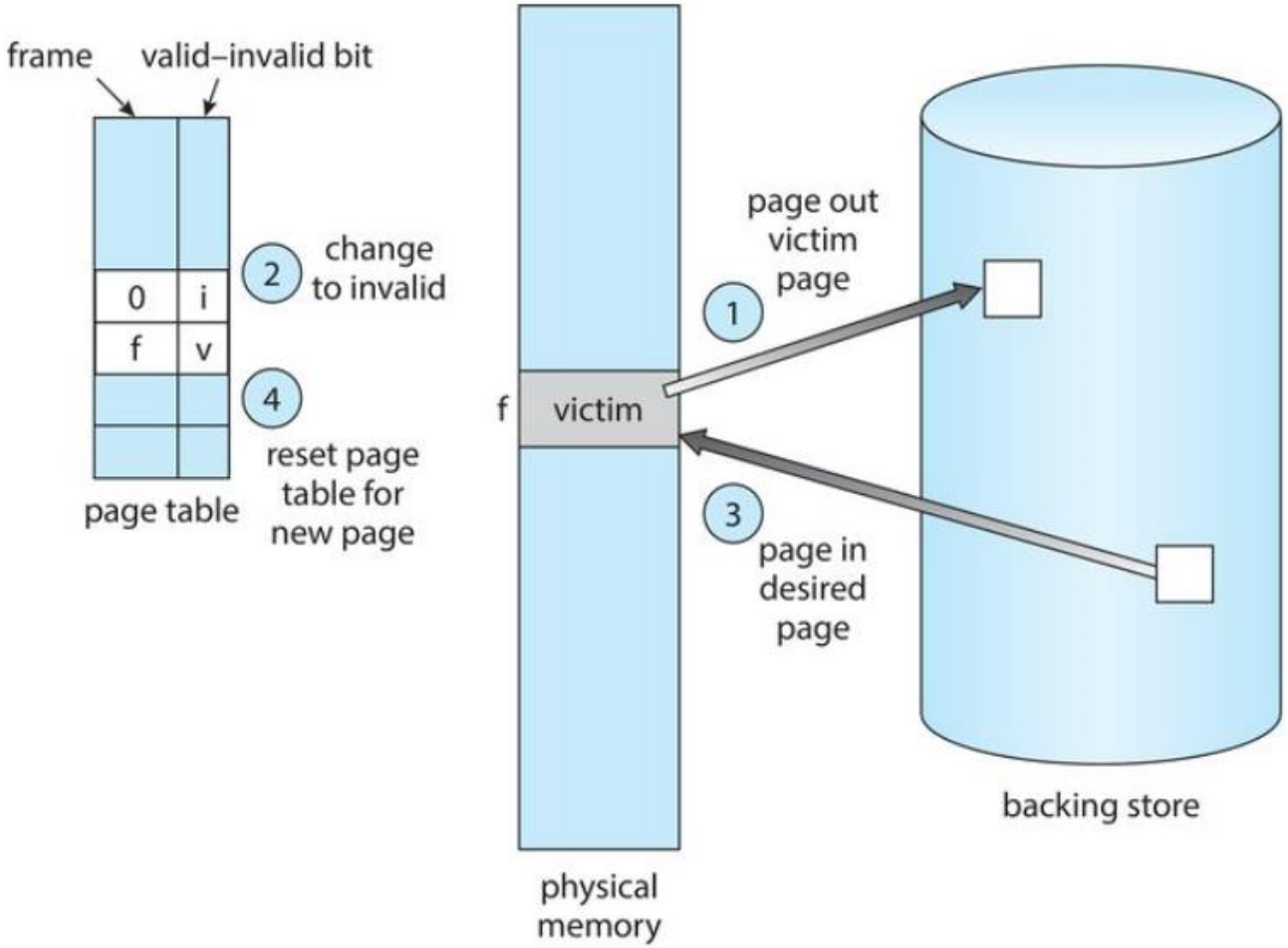
© by Dr. X. Sun

Page replacement in virtual memory

- In order to increase the degree of multiprogramming, we are over-allocating memory to processors. So, some pages are swapped out of the physical memory.
- If a page fault occurs and there is no free frame in the main memory, then how to swap the referenced page in the memory?---page replacement
- Page replacement: find a suitable page in memory, and swap it out to the hard disk.



- Basic steps of page replacement
  - Victim frame is the selected frame (by the page replacement algorithm) to be swapped out



Page replacement algorithm

- Design metric
  - ✓ will result in the lowest page-fault rate
  - ✓ will introduce the minimum runtime overhead
    - overhead of executing the replacement algorithm

- Evaluate algorithm by running it on
  - ✓ a particular string of memory references (reference string)
  - ✓ 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

© by Dr. X. Sun

Page replacement algorithm

- FIFO Page Replacement---If a page must be replaced, the oldest page is chosen.
  - Reference string: **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**
  - 3 frames

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

  - Page fault = 9/12

indicate page fault

© by Dr. X. Sun

Page replacement algorithm

- FIFO Page Replacement---If a page must be replaced, the oldest page is chosen.
  - Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
  - 4 frames

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

  - Page fault = 10/12 indicate page fault

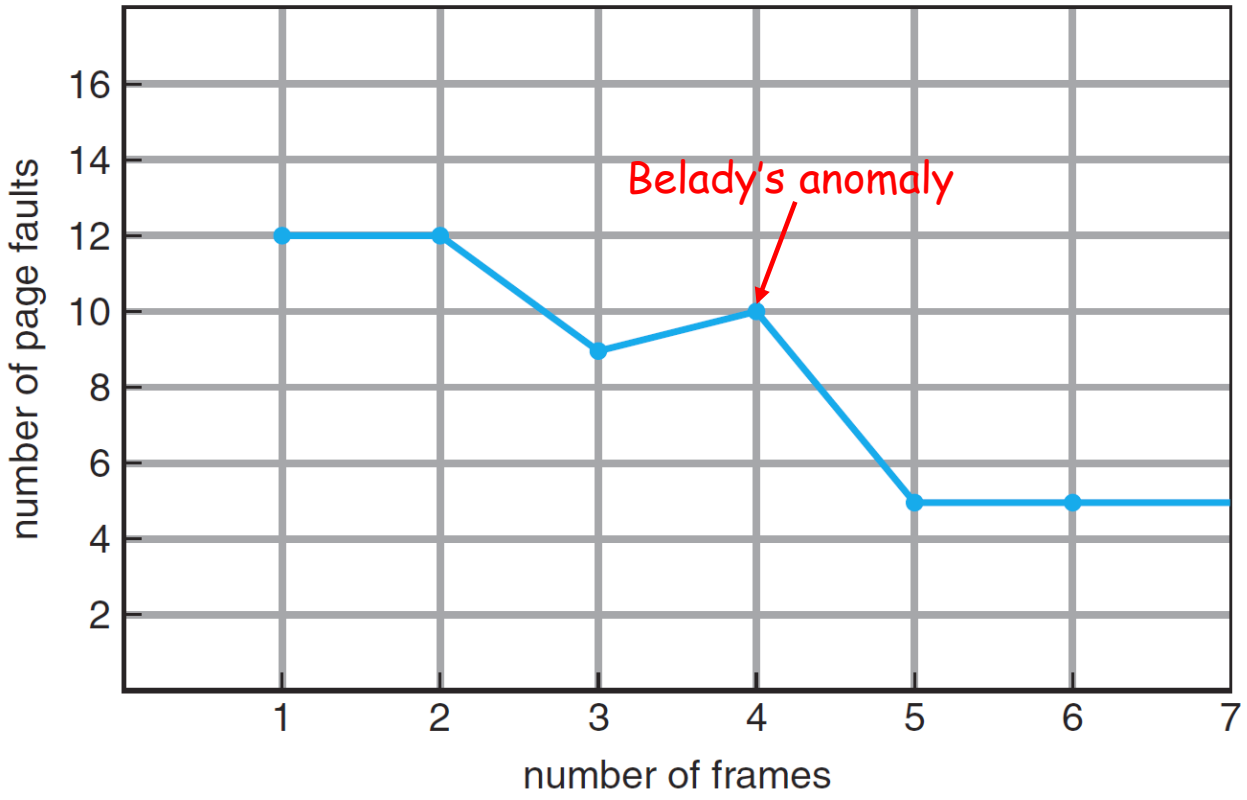
© by Dr. X. Sun

Page replacement algorithm

- FIFO Page Replacement---If a page must be replaced, the oldest page is chosen.
  - observation: more frames NOT → lower page faults.

    \+ easy to implement

    \- not satisfactory performance



© by Dr. X. Sun

Page replacement algorithm

- Optimal Page Replacement ---looking forward to the future: replace the page which will not be referenced for the longest time.

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

- Page fault = 6/12

© by Dr. X. Sun

Page replacement algorithm

- Optimal Page Replacement ---looking forward to the future: replace the page which will not be used for the longest time.
  - \+ Optimal
  - \- Almost impossible to implement
  - Used for measuring other algorithms' performance
  - If looking forward is not implementable, then let's looking backwards

© by Dr. X. Sun

Page replacement algorithm

- Least Recently Used (LRU) Page Replacement ---Looking backwards: replace the page which was least recently used.

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

- Page fault = 8/12

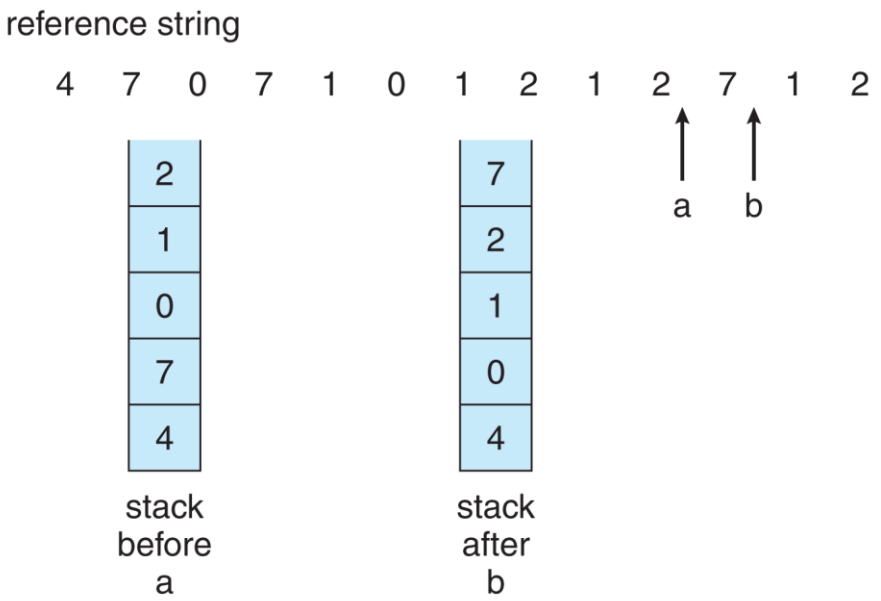
© by Dr. X. Sun

LRU (least Recently Used) replacement algorithm implementation

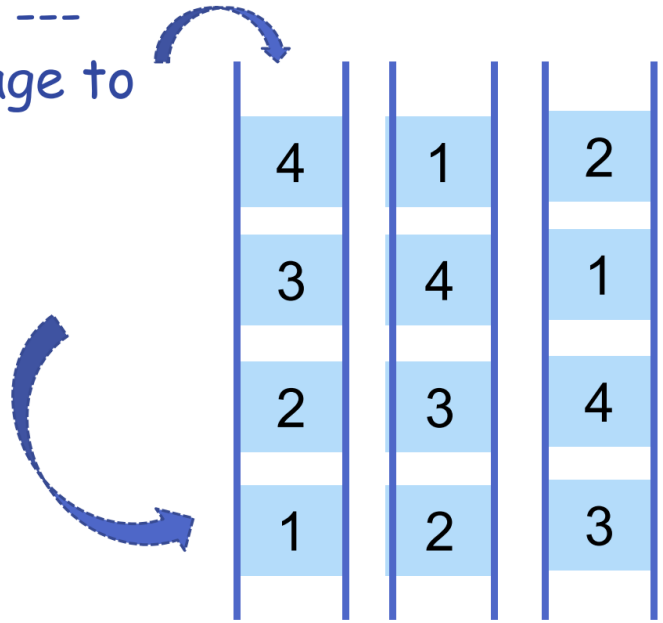
- Counters
  - ✓ Each page-table entry is associated with a time-of-use field.
  - ✓ Whenever a refence to a page is made, the current time (in the clock register) are copied to the time-of-use field.
  - ✓ The page with the smallest value in the time-of-use field (i.e., least recent used) will be replaced.

  + Simple to understand
  - High overhead: 1) conduct a write to memory (page table) for each memory access; 2) search the entire page table to find the LRU page.

- Stack
  - ✓ Use a stack to keep page numbers being referenced.
  - ✓ Whenever a page is referenced, it is removed from the stack and put on the top.
  - ✓ The LRU page is always at the bottom.
  - ✓ Lower overhead, but entries need to be removed from the middle of the stack (e.g., "7")



- LRU (least Recently Used) replacement algorithm implementation
  - Link List—keep a double-linked list of page frames
    - memory reference time --- move the referenced page to the tail
    - replacement time --- pick up one at the head
    - link list operations are expensive



Reference string: **1, 2, 3, 4, 1, 2**

© by Dr. X. Sun

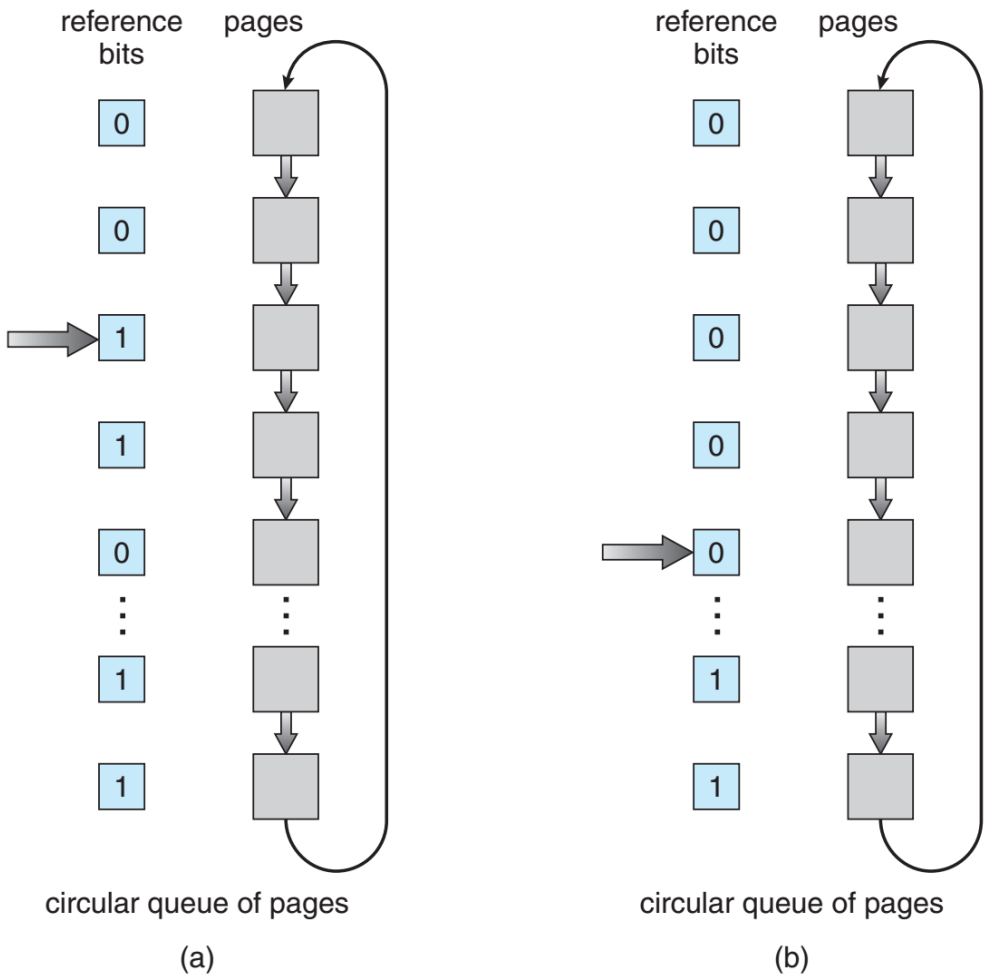
Some Approximated-LRU Algorithms ---Not-recently-used

- Reference bit – R only
  - ✓ one reference bit R per page table entry
  - ✓ memory reference time --- set R = 1
  - ✓ replacement time --- randomly pick up one with R is 0, reset all Rs to 0
- Reference bit – R & Modification bit - M
  - ✓ reference bit R + modification bit M per page table entry
  - ✓ memory reference time --- set R = 1, and set M = 1 if write
  - ✓ replacement time ---
    - randomly pick up one with the minimal value of R&M
    - reset all Rs to 0

© by Dr. X. Sun

Some Approximated-LRU Algorithms --- Not-recently-used

- Reference bit + FIFO - second chance
  - ✓ keep a circular list of all page frames
  - ✓ one reference bit R per page table entry
  - ✓ keep a current inspection point
  - ✓ memory reference time --- set R=1
  - ✓ replacement time ---
    - if the current entry with R=1
      a) set R=0;
      b) advance the point, "give it a second chance"
    - else
      the current page is the "victim page"; replace the "victim page" with the new page.



Some Approximated-LRU Algorithms ---Not-recently-used

- Reference bit + aged history
  - ✓ one reference bit R + counter C (8 bits; initially, "00000000"") per page table entry
  - ✓ memory reference time --- set R=1
  - ✓ at each regular interval (e.g., 100 ms)---1) shift the reference bit into the high-order bit of its count C for each page entry; 2) shift the other bits right by 1 bit; 3) discard the low-order bit; 4) set R=0;
    - e.g., if C=00000000 and R=1, after one interval, C=10000000 and R=0.
  - ✓ replacement time ---
    - replace the page with the smallest value of C

© by Dr. X. Sun

Other page replacement algorithms

- Counting-based page replacement
  - ✓ One counter per page table entry. The counter records the number of references that have been made to its page.
    - Least frequently used algorithm: pick the page with the smallest value of counter being replaced.
    - Most frequently used algorithm: pick the page with the maximum value of counter being replaced
- Page-Buffering Algorithm
  - ✓ Construct a pool of free page pool in advance (before a page fault incurs).

© by Dr. X. Sun

ECE437/CS481

M07D: VIRTUAL MEMORY

CHAPTER 9.5-9.4

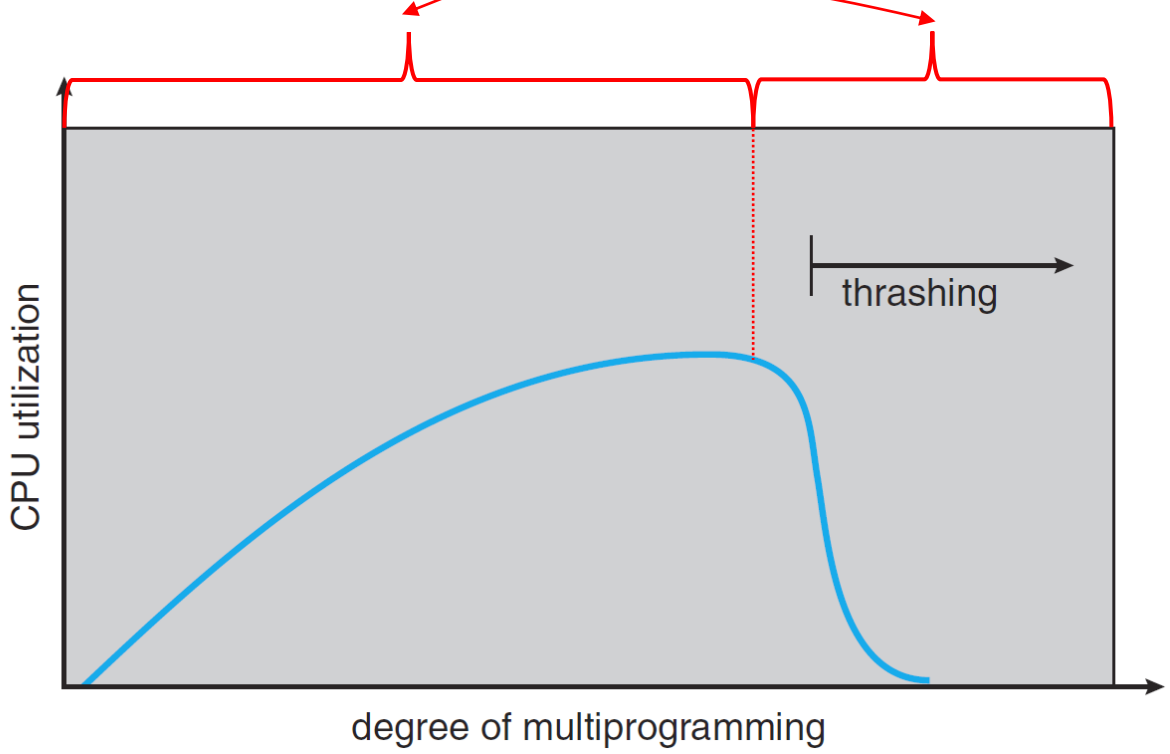
Xiang Sun

The University of New Mexico

Thrashing

- As the degree of multiprogramming increases, more processes will be brought into the main memory
  - The average CPU utilization may increase.
  - The size of memory space allocated to each process may reduce.
    - → If reduced, more pages of a process will be stored in the hard drive.
    - → The page fault rate increases.
    - → The OS spends more time on swapping in/out pages.
    - → Reduce the CPU utilization.

- Thrashing ➔ A process is busy in swapping pages in and out.



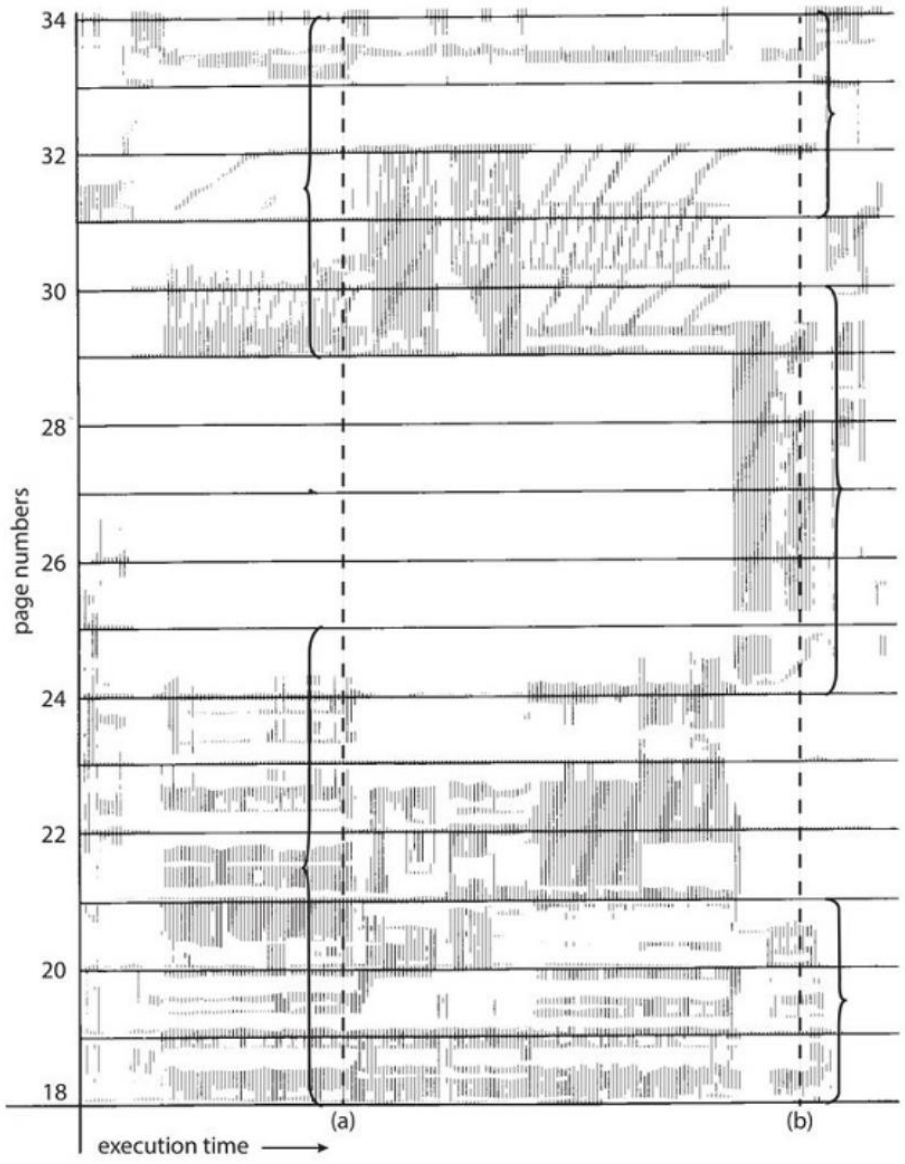
Thrashing

- How to mitigate process thrashing?
  - Basic idea: 1) prefetch the memory pages of a process that will be referenced in the near future; 2) if the memory space is not enough, suspend the process.
    - ✓ Figuring out the memory pages of a process to be referenced soon is difficult.
  - Locality feature of a process—a set of pages for a process are actively used together
  - Two types of locality
    - ✓ Temporal locality
      - A process's pages, which are referenced recently, will likely be referenced in the near future.
      - E.g., looping, counting/reduction variables, etc.
    - ✓ Spatial locality
      - One page of a process is referenced, and its nearby pages will be referenced in the near future.
      - E.g., array, sequential code.

© by Dr. X. Sun

Thrashing

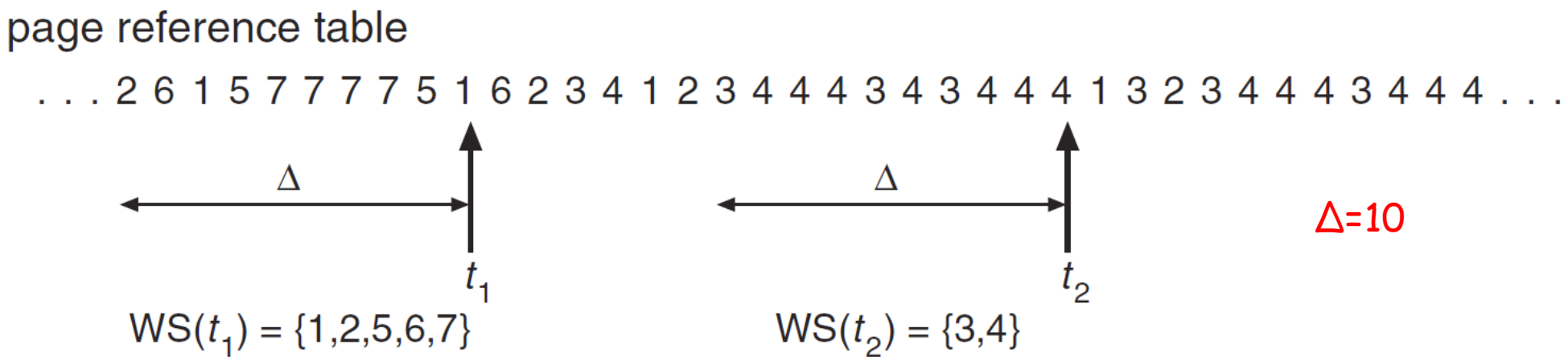
- Locality in a memory-reference pattern
  - In time slot (a), the spatial locality is [18,19,20,21,22,23,24,29,30,33]
  - In time slot (b), the spatial locality is [18, 19, 20, 24, 25, 26, 27, 28, 29, 31, 32, 33].
  - The locality may change over time.



© by Dr. X. Sun

Thrashing

- Based on the locality feature, the working set model is designed to figure out the need of a process in the near future.
  - One parameter, working set window Δ, is used to monitor the recent referenced pages.



  - The accuracy of the working set depends on the value of Δ.
    - ✓ If Δ is too small, it will not cover the entire locality.
    - ✓ If Δ is too large, it may overlap several localities.
    - ✓ If Δ is infinite, the working set includes all the pages during the process execution.

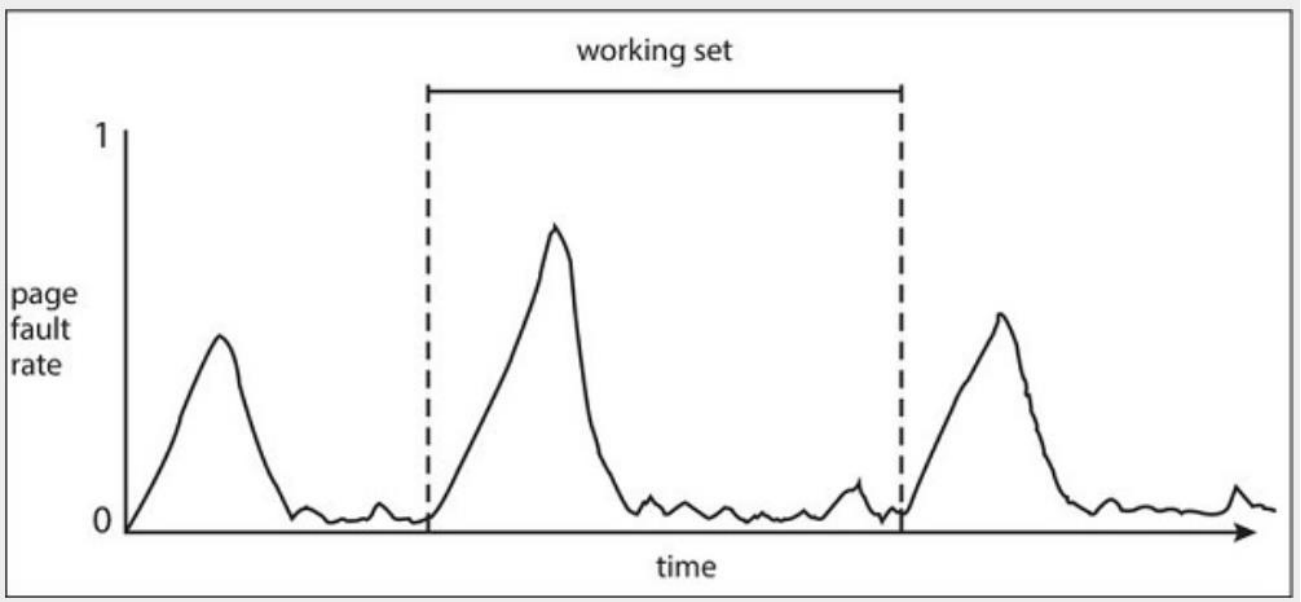
Thrashing

- Procedures of using working set to mitigate the process thrashing.
  - The OS monitors the working set of each running process and tries to allocate the frames to each running process, i.e., # of frames allocated to process $i = |WS_i|$, where $|WS_i|$ indicates the size of the working set for process i.
  - Check if the total number of free frames (denoted as D) is larger than the requirements, i.e., D > $\sum_i$ |WSi| ?
    - ✓ If yes, another process can be initiated (by allocating frames to the process).
    - ✓ If no, the OS selects a process(es) to suspend.

© by Dr. X. Sun

Thrashing

- Relationship between the working set of a process and its page fault rate.
  - As mentioned before, the locality/working set of a process may vary over time.
  - The page fault rate of a process transits between peaks and valleys over time. Here,
    - ✓ The peaks indicate the pages in working set changes, and thus the OS has to bring the pages from the hard disks into memory (demand paging)
    - ✓ The valleys indicate all the pages in the working set have swapped in the memory.



© by Dr. X. Sun