Preview

- Memory Management
  - With Mono-Process
  - With Multi-Processes
  - Multi-process with Fixed Partitions
  - Modeling Multiprogramming
  - Swapping
  - Memory Management with Bitmaps
  - Memory Management with Free-List

Memory Management

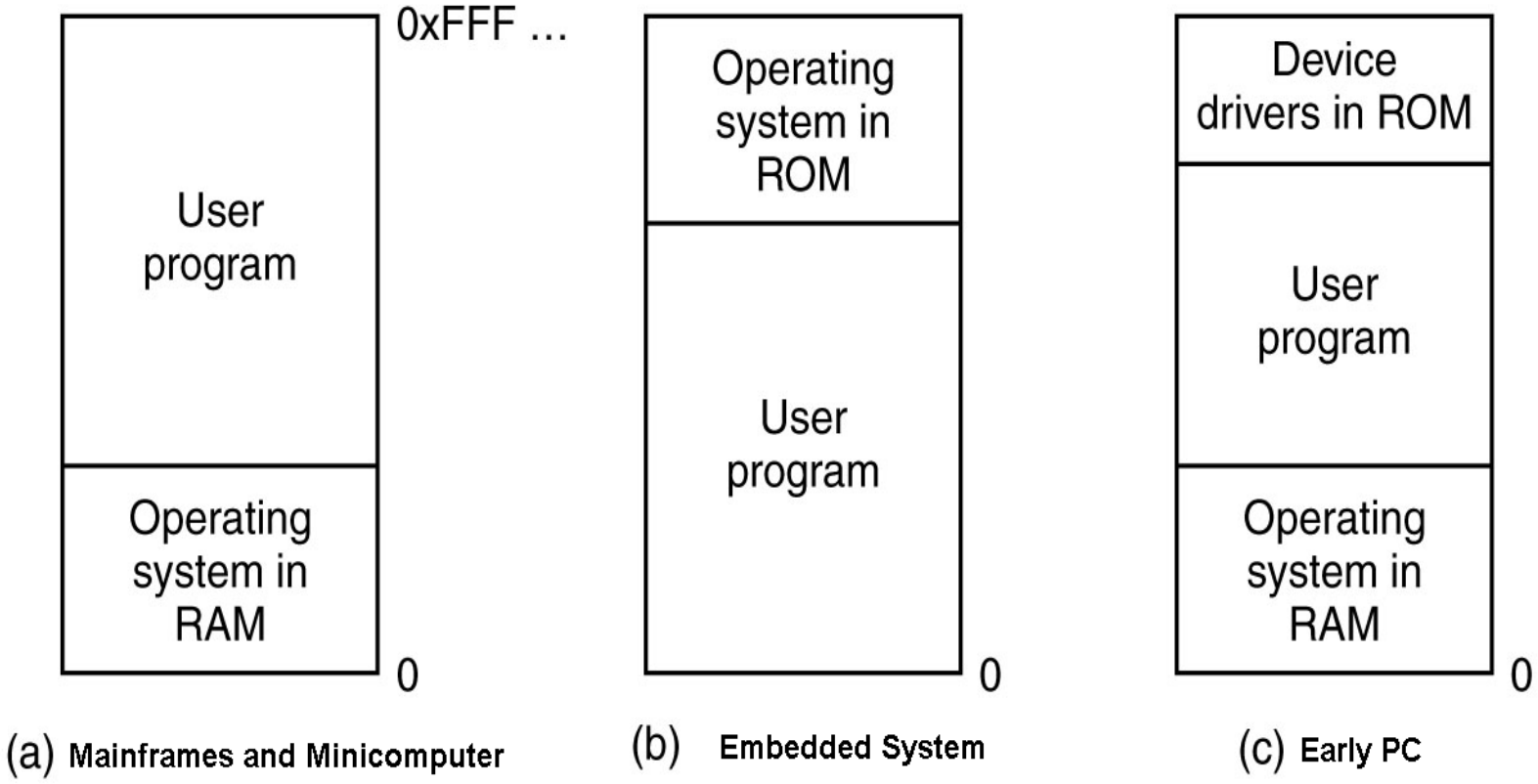
- Ideal Memory –
  - Infinitely Large
  - Infinitely Fast
  - Non-volatile
  - Inexpensive
- No such a memory, most computers has a memory hierarchy
- Memory hierarchy –
  - Small, fast, very expensive registers
  - Volatile, expensive cache memory
  - Volatile, megabyte medium-speed, medium-speed RAM
  - Non-volatile, hundreds or thousands of gigabits of slow cheap disk storage (Hard Disks)
- Memory management is a part of the operating system which manages the memory hierarchy

Memory Management (mono-process)

Mono-programming without Swapping or Paging

- Memory sharing only between a user program and the operating system in this system.
- Only one program could be loaded in the memory.
- When the program finishes its job, a scheduler (long term scheduler) chooses one job from the pool of jobs and loads it into the memory and starts to run.

Memory Management (mono-process)



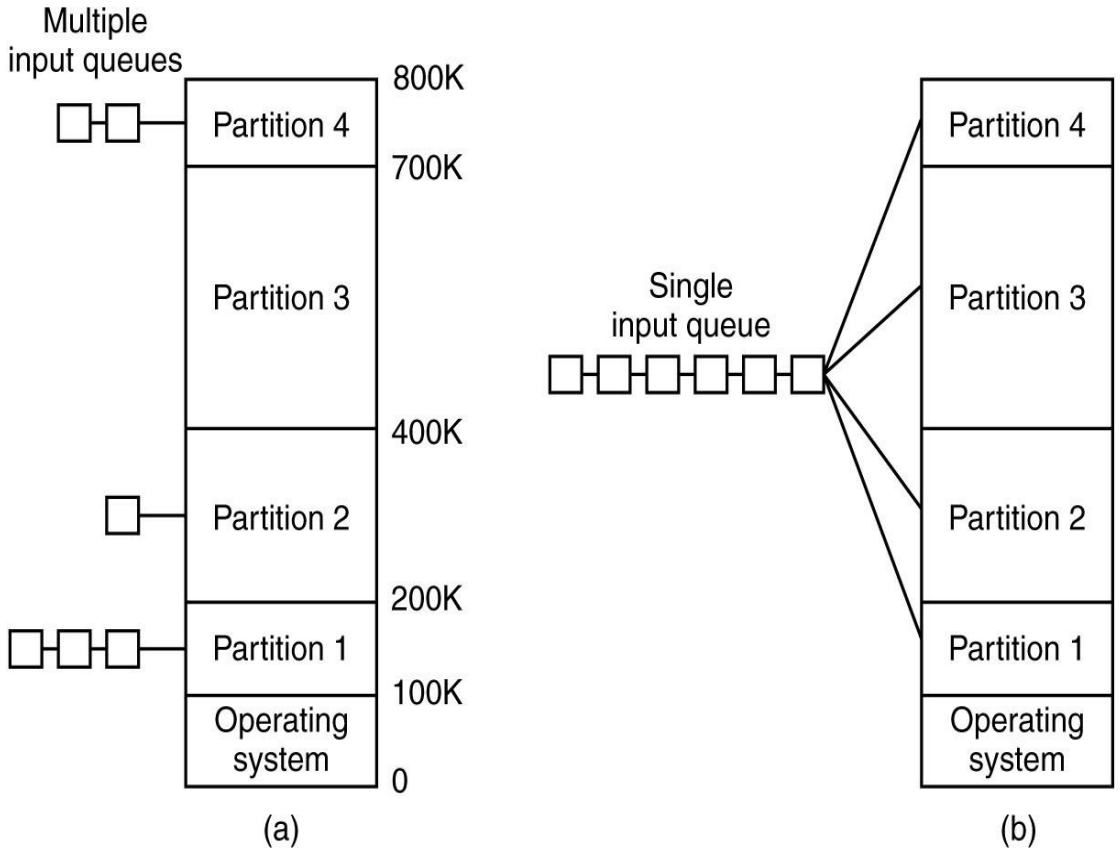
Memory Management (multi-process)

- Multiprogramming with Fixed Partition - Memory is divided into n (possibly unequal size) partitions – partition can be done manually when the system started.

1. Fixed memory partitions with separate input queues for each partition
2. Fixed memory partitions with a single input queue.

Memory Management

(multi-process with fixed partition)



Memory Management
(multi-process with fixed partition)

- Fixed memory partitions with **separate input queues** for each partition
  - When a job arrives, it can be put into the queue for the smallest partition that is large enough to hold it.
  - Disadvantage – Large partition queue is empty but many small jobs are waiting on the small partition queue.

Memory Management
(multi-process with fixed partition)

- Fixed memory partitions with a **single input queue**.
  - Whenever a partition becomes free, the closest to the queue that fits in it, is chosen and loaded into the empty partition and run – wasting memory. Or
  - Search the whole queue to find out a job which is the best fit to the free partition. – discriminate against small jobs.
    - One partition for small jobs
    - Count the time skipped by scheduler. If a job was skipped more than k time, it is automatically granted next time.

Modeling Multiprogramming

Simple Unrealistic Model

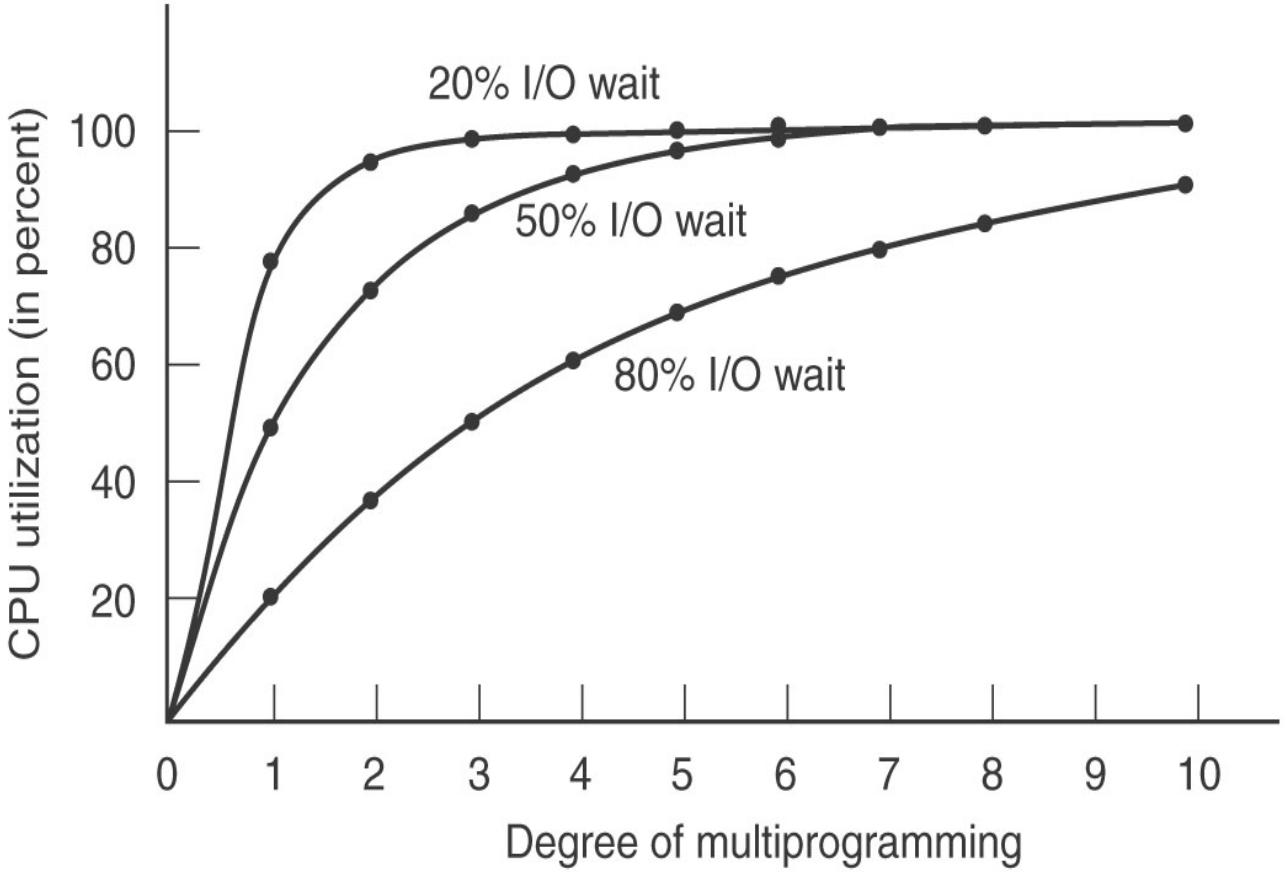
- Assumption: all five processes will never be waiting for I/O at the same time
- If the average processes use 20% computing time and 80% use for waiting I/O. With five processes in the memory at once, the CPU should be busy 100%

Modeling Multiprogramming

Probabilistic model

- p - a fraction of time a process is waiting for I/O.
- If there are n processes in the memory at once, the probability that all n processes are waiting for I/O is p^n.
- Then, the CPU utilization is given by
- CPU utilization = $1 - p^n$ (the probability that at least one of processes is using CPU)

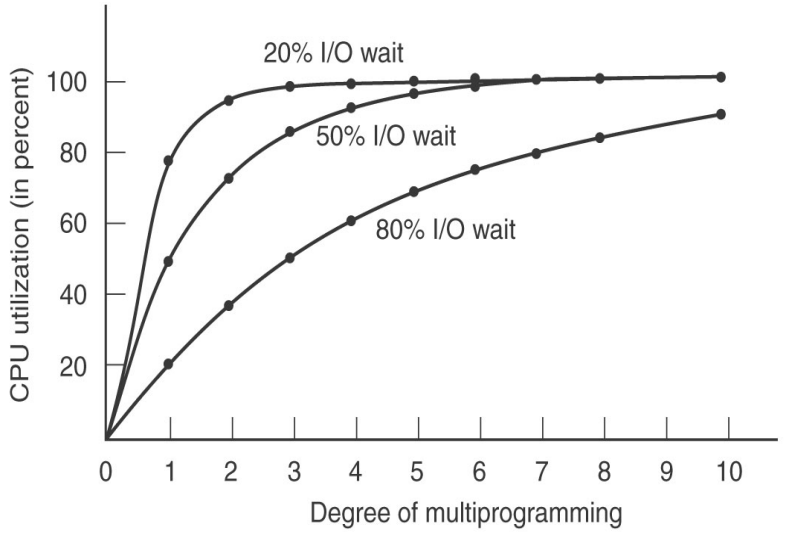
Modeling Multiprogramming



Modeling Multiprogramming

Ex) A computer has 32MB of memory, with operating system taking up 16 MB and each program taking up 4MB.

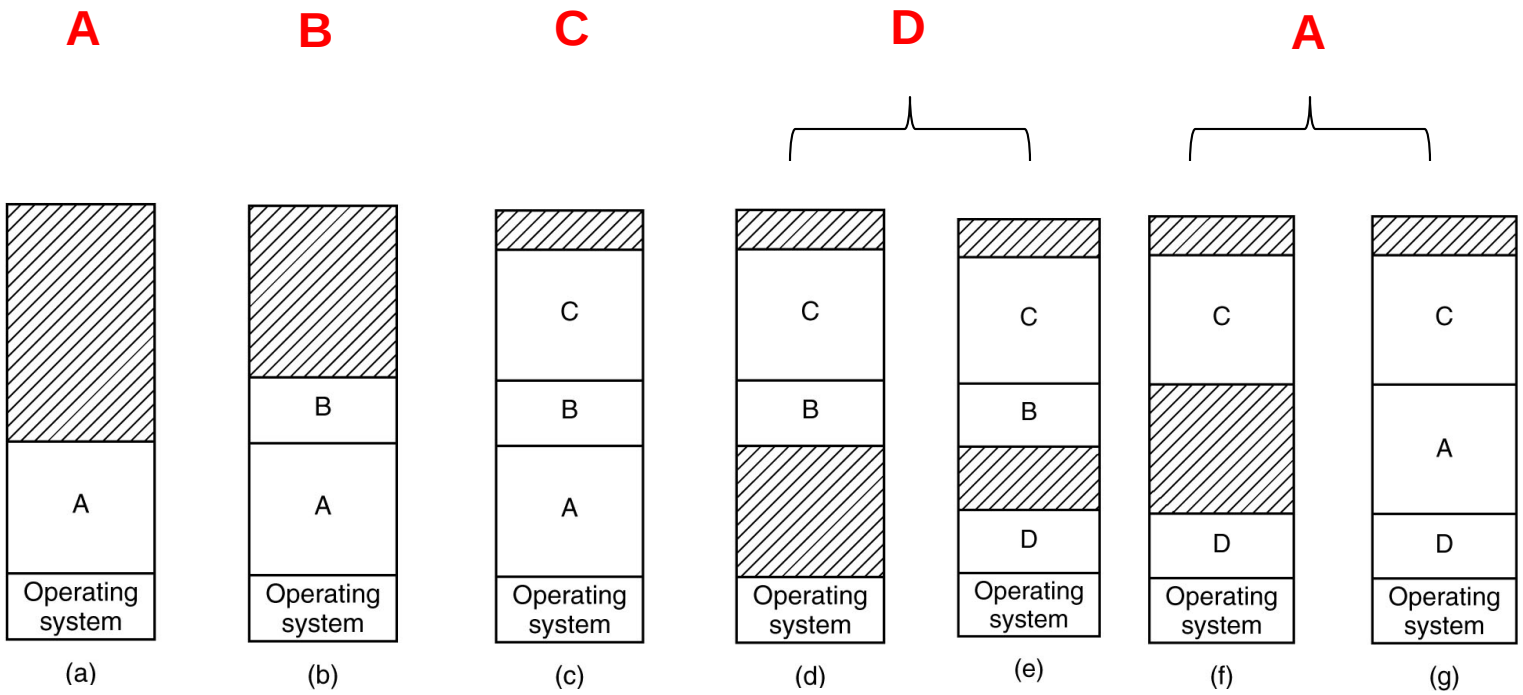
- 4 programs to be in memory at once (4 X 4 = 16MB).
- With an 80 % average I/O wait, we can have about 60 % of CPU utilization.
- Adding 16MB memory, (4 more programs: total 8 programs to be in the memory) we can get about 83% of CPU utilization.
- Adding yet another 16MB memory, (total 12 programs to be in the memory) we can get about 93% of CPU utilization



Memory Management

- There is not enough main memory to hold all the currently active processes, so excess processes must be kept on disk and brought in to run dynamically.
- Two general approaches for memory management (depending on the available hardware)
  1. **Swapping** – bringing in each process in its entirety, running it for a while, then putting it back on the disk.
  2. **Virtual Memory** – allows programs to run while partially loaded in the memory.

Swapping with Variable Partition



Swapping

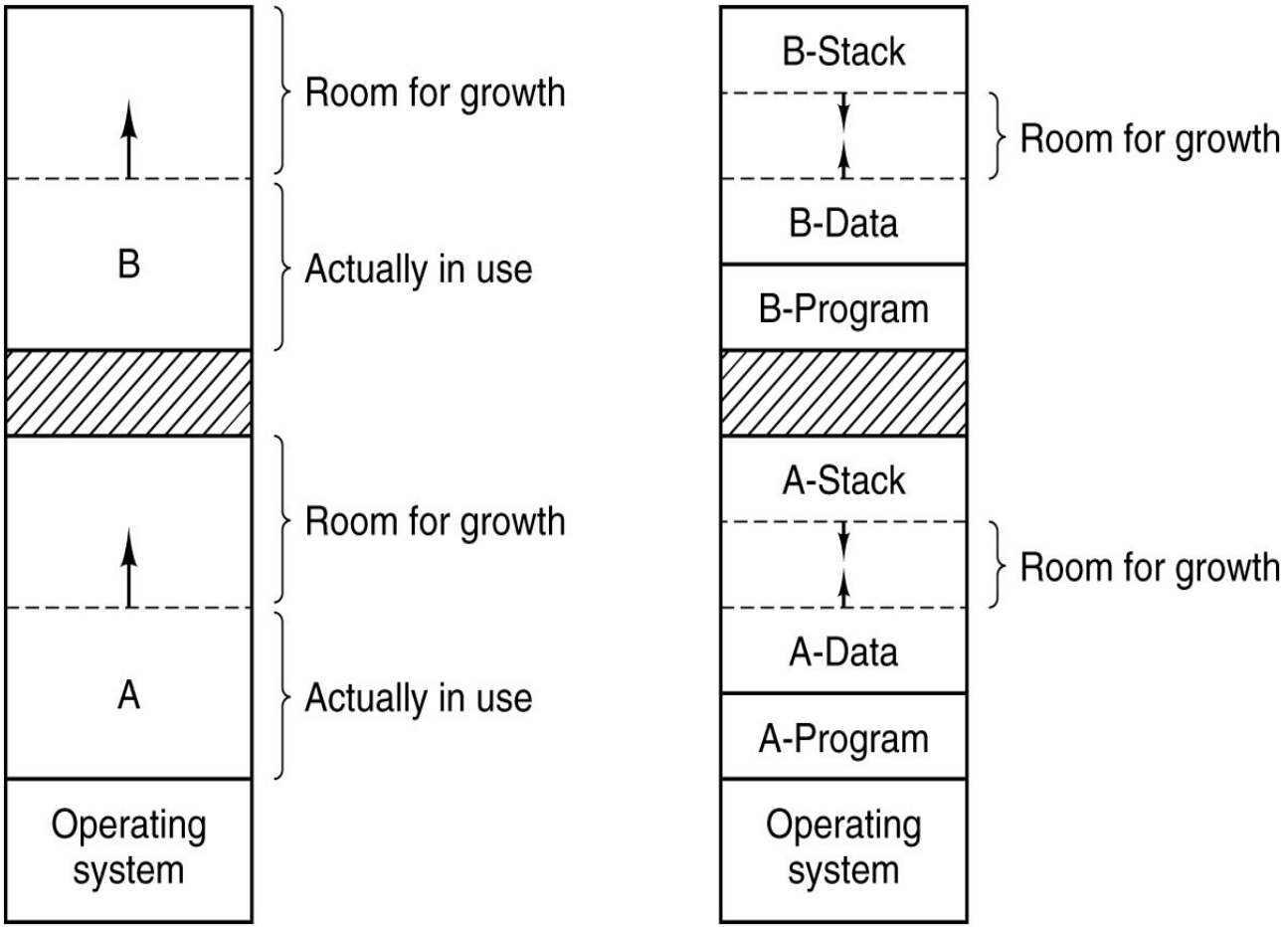
- Main difference between fixed partition and variable partition
  - Variable partition
    - The number, location, and size of the partitions vary dynamically
    - Need to keep track of partition information dynamically for memory allocation and de-allocation
    - Might create multiple holes – Combine them all into one big hole (memory compaction: expensive cycle)
  - Fixed partition
    - The number, location, and size of the partitions fixed.
    - OS knows the address of each process.
    - Simple to manage the memory.

Swapping

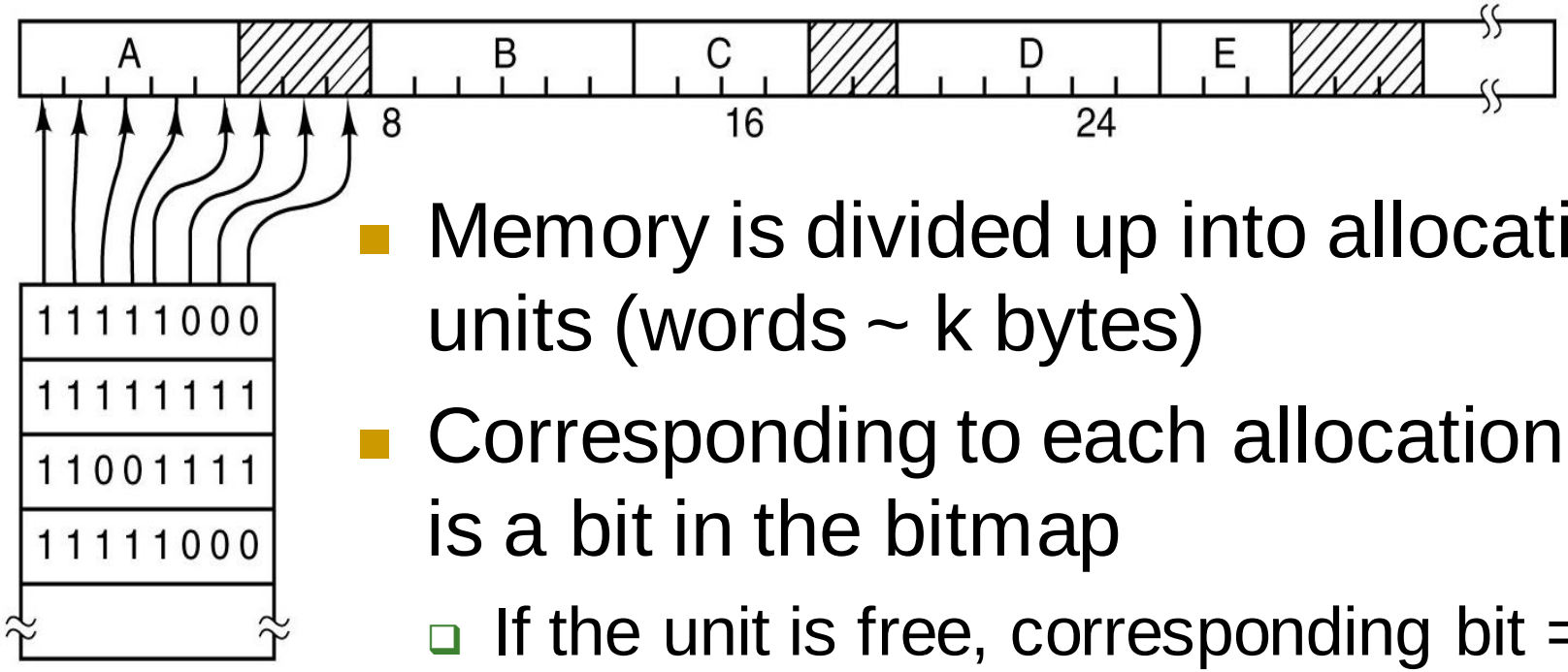
How much memory should be allocated for a process when it is created or swapped?

- If processes are created with a fixed size – simple
- But processes are usually changing their sizes by dynamically allocating memory – Dynamic memory allocation (heap), recursion
  - Allocate extra memory space for each process
  - Use adjacent hole for managing the growing size
- If there is not enough memory space for a process based on the dynamically changing size, the process might be swapped out or killed.

Swapping


Room for growth
B
Actually in use
Room for growth
A
Actually in use
Operating system
B-Stack
Room for growth
B-Data
B-Program
A-Stack
Room for growth
A-Data
A-Program
Operating system

Memory Management with Bitmaps



- Memory is divided up into allocation units (words ~ k bytes)
- Corresponding to each allocation unit is a bit in the bitmap
  - If the unit is free, corresponding bit = 0 and
  - If unit is occupied, corresponding bit = 1.

Memory Management with Bitmaps

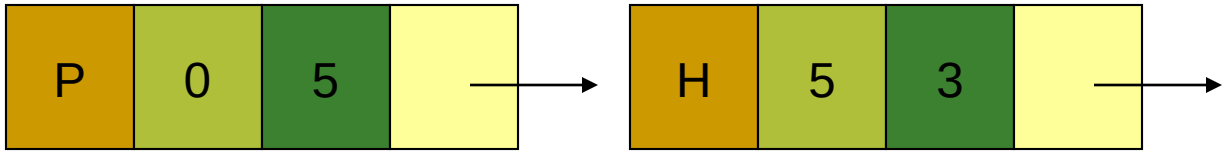
- Size of the allocation unit is an important design issue
  - The smaller the allocation unit, the larger the bitmap.
  - The larger the allocation unit, the smaller the bitmap – But memory may be wasted in the last unit of the process if the process size is not an exact multiple of the allocation unit.

Memory Management with Bitmaps

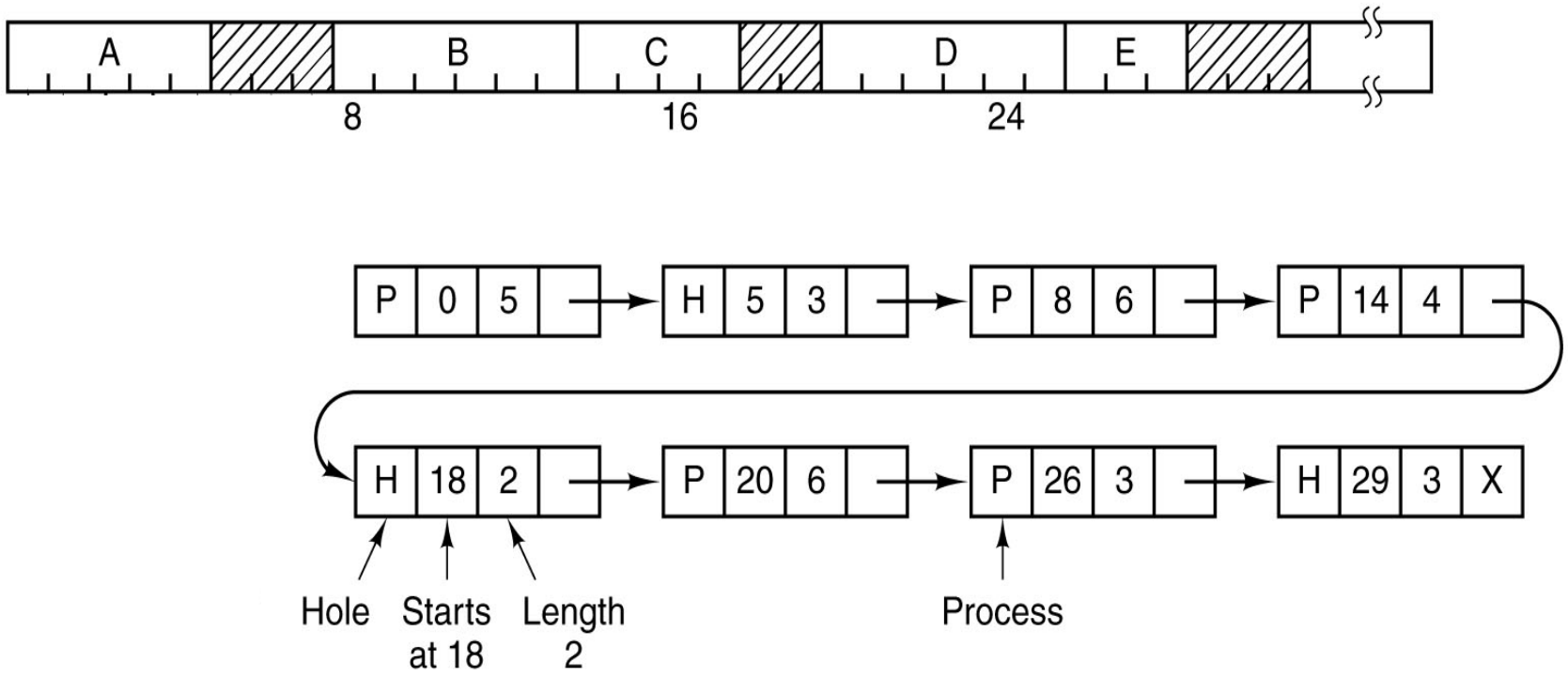
- Advantage with Bitmaps
  - Simple way to keep track of memory words in a fixed amount of memory since the size of bitmap depends only on the size of memory and size of the allocation units.
- Disadvantage with Bitmaps
  - To allocate memory for the process with k unit size, first, the memory manager need to search the bitmap to find out k consecutive 0 bits in the map.

Memory Management with Free-List

- With free-list, maintain a linked list of allocated and free memory segments.
- Each entry in the list keeps information: **hole or process**, **starting address**, **length and a pointer** to the next entry



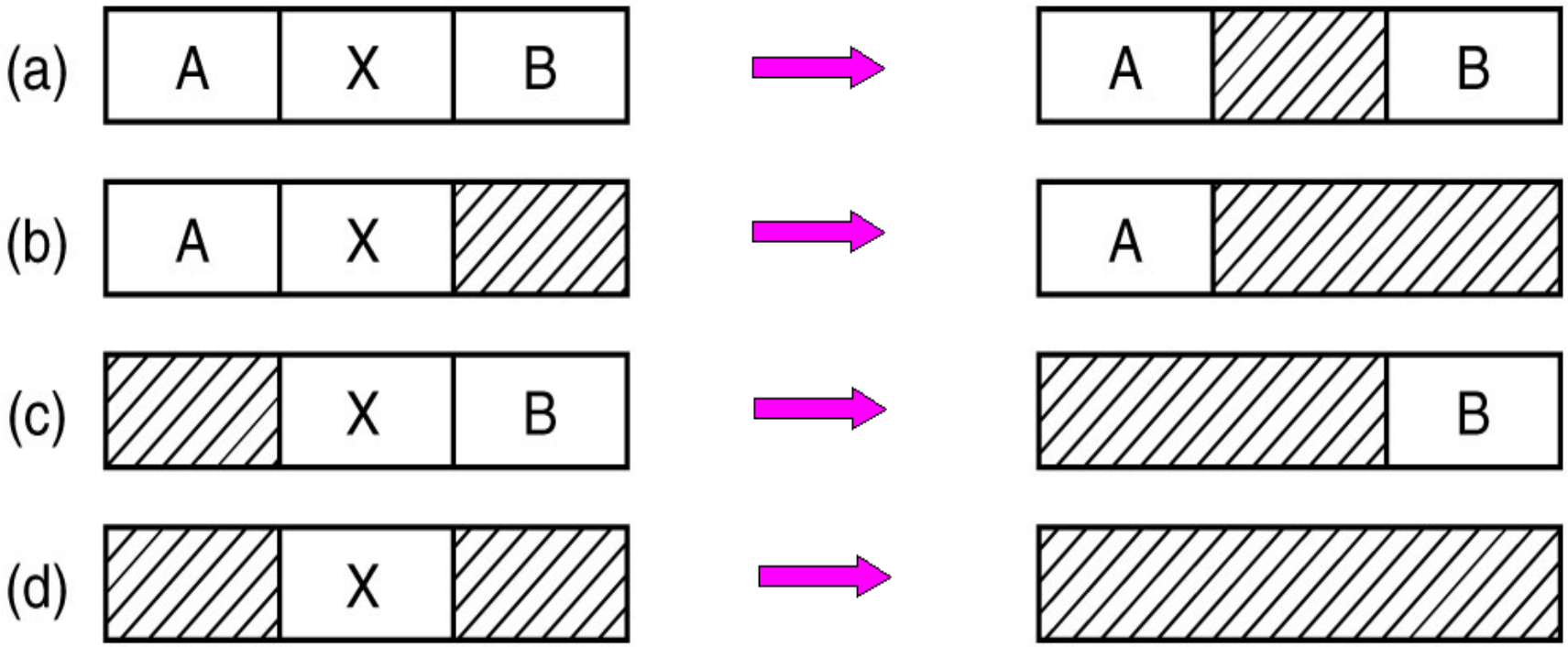
Memory Management with Free-List



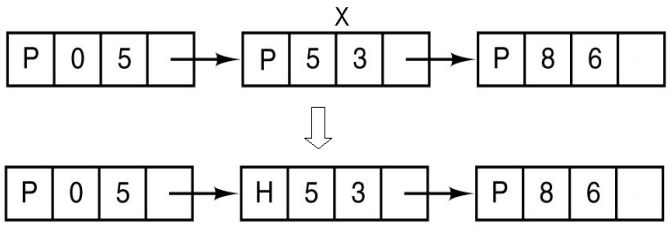
Memory Management with Free-List

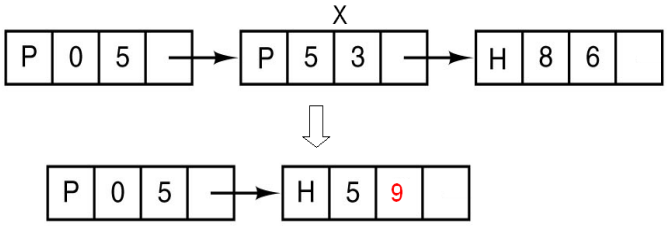
- If the segments list is kept sorted by address, it is simple to update the list when a process terminates or is swapped out from memory.

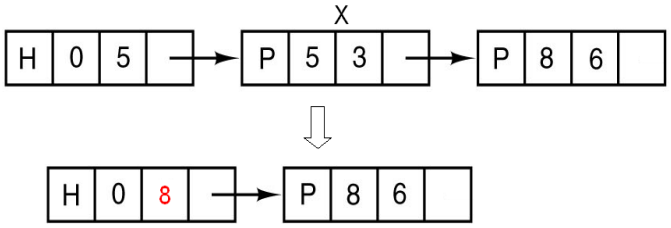
Memory Management with Free-List

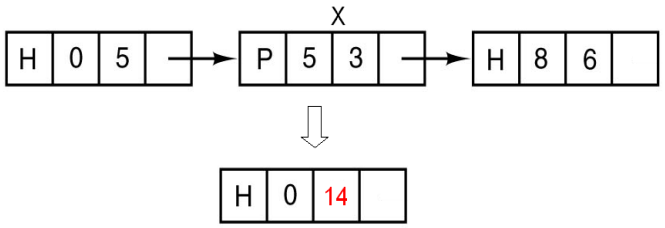


Memory Management with Free-List









Memory Management with Free-List

- Algorithms for allocating memory for a process (Assume that the list of segments is kept sorted by address value).
  - **First Fit** – The memory manager scans the list of segments from the beginning until it finds a hole that is big enough.
  - **Next Fit** – It works the same way as first fit. But next time it is called, it starts searching the list from the place where it left off last time.
  - **Best Fit** – It searches the entire list and takes the smallest hole that fit for the process.
  - **Worst Fit** – It always takes the largest free hole

Memory Management with Free-List

- Four algorithms can be more efficient by maintaining two lists: one for holes and another for processes.
  - The list of holes can be maintained with sorting by size. Best fit does not need to search entire list.
  - But still need extra effort for updating two lists based on the allocation and deallocation of memory.

Memory Management with Free-List

Quick Fit

- Maintains separate lists for holes based on the size of hole.
- It might have a table with n entries. Each entry is a pointer to the head of a certain size of holes list. Ex) First entry point to the head of list of 4-KB holes, second entry point to the head of list of 8-KB holes, ….
- Quick to finding a hole of required size
- Expensive to maintain the separate lists for hole based on the deallocation of memory

Virtual Memory

Motivation

- Initial idea of executing a program is that the entire logical address space of a process must be in physical memory before the process can access (swapping)
- But since size of a program grows tremendously based on the complexity of modern software, and limited size of physical memory, initial idea is not possible

Virtual Memory

Overlay

- Split a program into pieces (overlays)
- Overlay 0 starts running first; when it finishes its job, it will call another overlay.
- Overlays were kept on the disk and swapped in and out of memory by OS.
- The work of splitting the program into pieces had to be done by programmer
- Splitting up large program into small pieces is time consuming

Virtual Memory

Virtual Memory

- Virtual memory is a technique that allows the execution of processes that may not be entirely in memory
- The OS needs to keep tracking the parts of the program currently located in main memory and the rest on the disk.
- Virtual memory idea can be used in a multiprogramming system.

Virtual Memory with Paging

Paging

- Virtual address space is divided into units called **pages**.
- The corresponding units in the physical memory is called **page frames**.

the size of a page = the size of a page frame

- Size of page is usually between 512 byte and 64KB

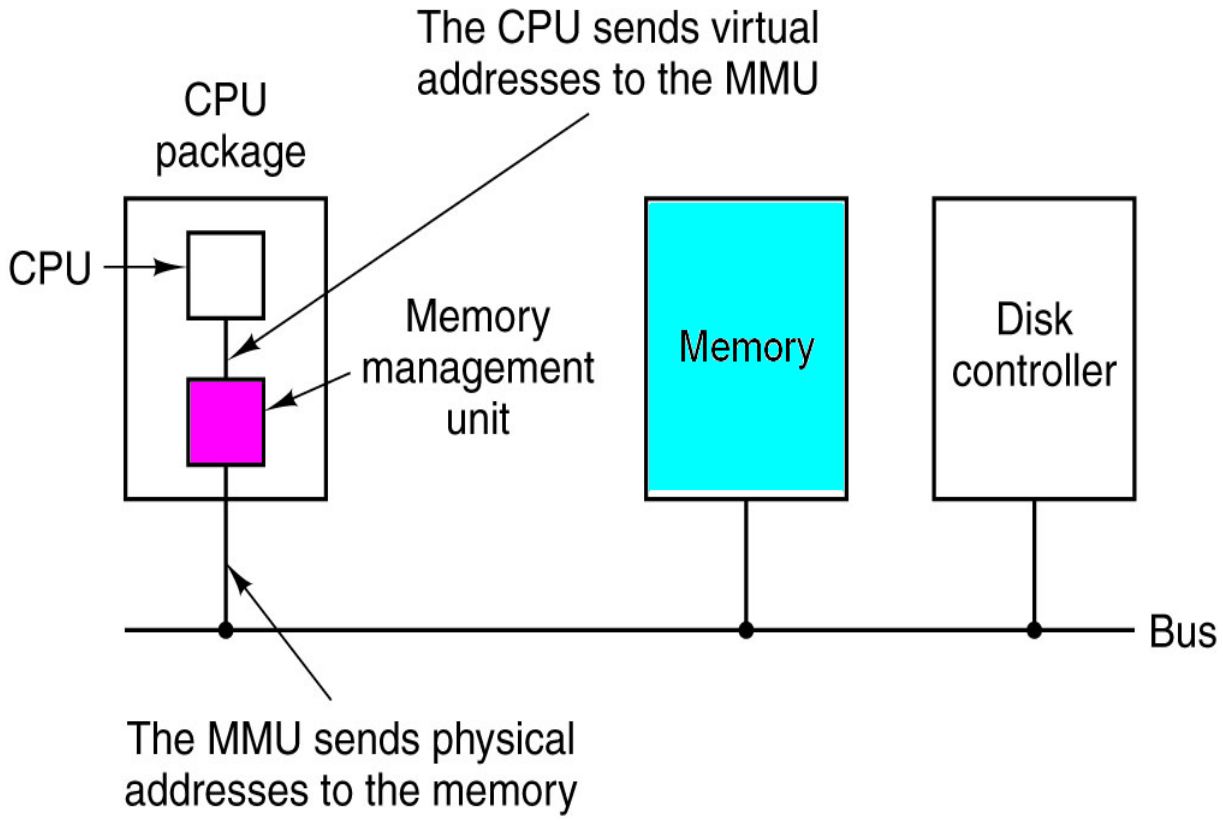
Virtual Memory with Paging

Mapping from Virtual Memory Address to Physical Memory Address

- When virtual memory is used, a virtual address (which is generated by a program) does not go directly onto the memory bus.

- Instead, they goes to an MMU (Memory Management Unit) that maps the virtual addresses onto the physical memory addresses based on the memory map (page table).

Virtual Memory with Paging

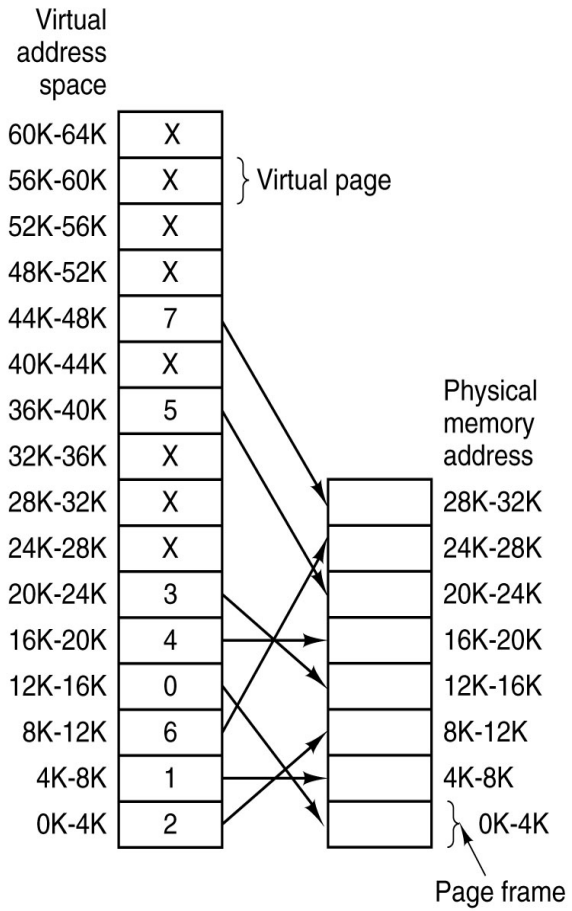


Virtual Memory with Paging

Example)

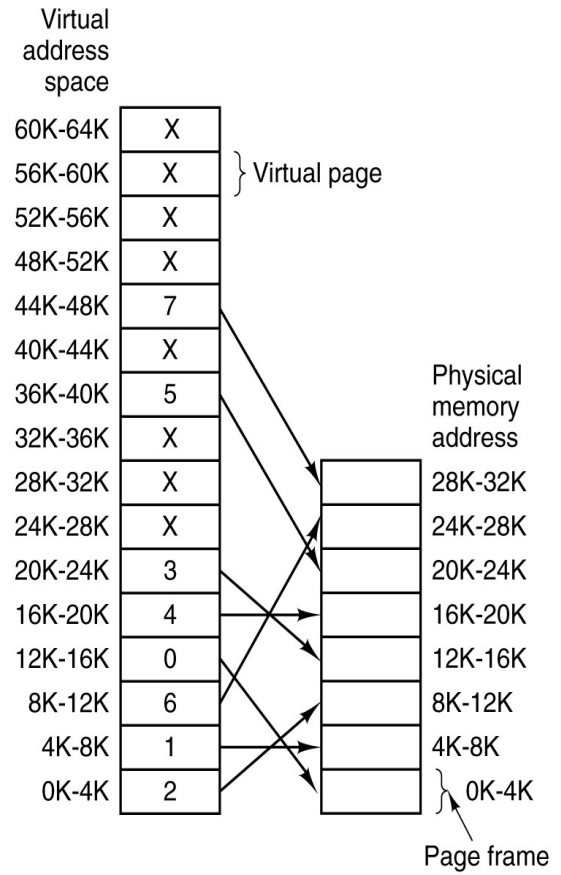
- A computer that can generate 16-bit address – 64KB virtual space ($0 \sim 2^{16}-1 = 0 \sim 64 \times 2^{10}$)
- A system is using the virtual memory with page size 4KB
- A computer has 32 KB memory
- There are 16 virtual pages and 8 (32/4) page frames.

Virtual Memory with Paging



Virtual Memory with Paging

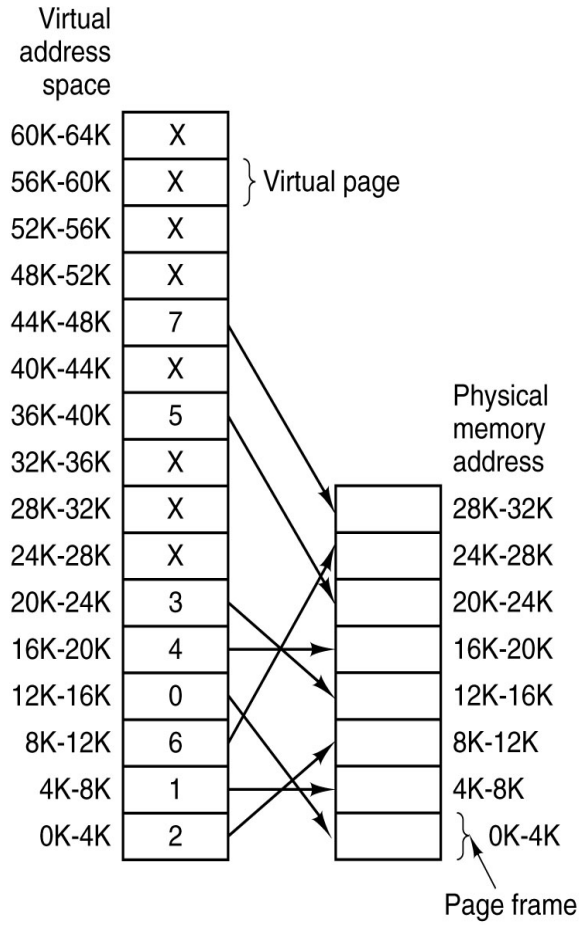
- **MOV REG, 0** ;;move content of address 0 to register.
  - virtual address 0 is sent to MMU.
  - MMU checks virtual address, 0 belongs to virtual page 0 (0 ~ 4095 (= 4×2^{10}).
  - MMU checks memory map such that page 0 is mapped to page frame 2 (8K ~ 12K)
  - MMU sends 8192 (= 2^{13}) physical address to address bus.



Virtual Memory with Paging

MOV REG, 8900

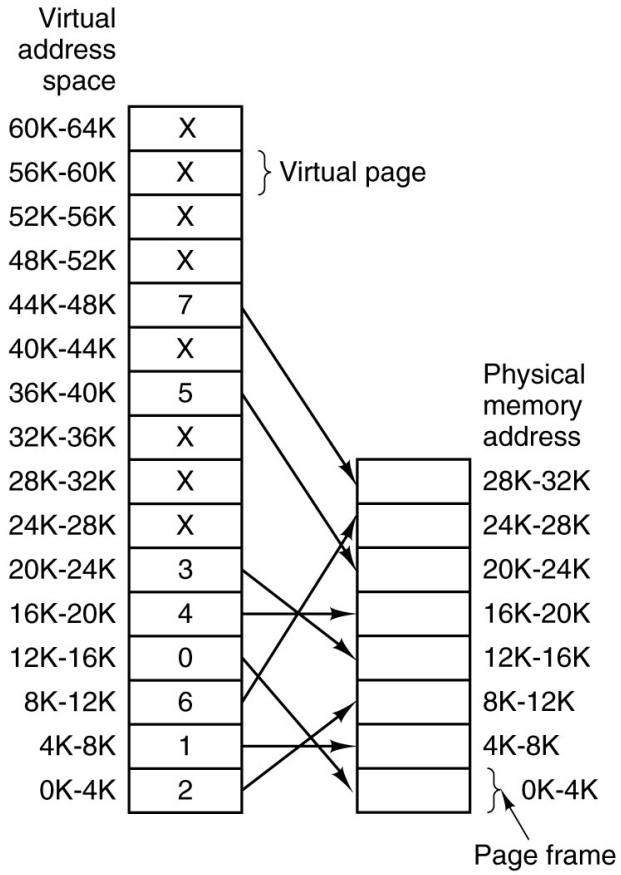
- Virtual address 8900 is send to MMU
- MMU calculates that address 8900 belongs to virtual page 2 (=between 8 KB ~ 12 KB) (8192~12288)
- MMU checks memory map such that virtual page 2 is mapped to physical frame 6 (24KB ~ 28 K: (24576 ~28672)
- MMU sends address 24576 + (8900 - 8192) = 25284 physical address to address



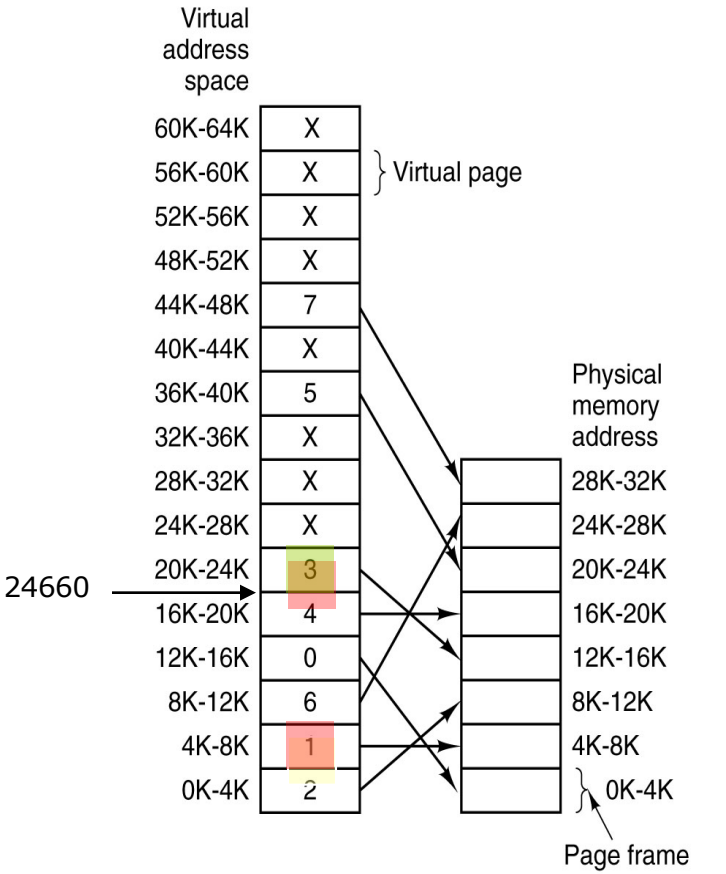
Virtual Memory with Paging

MOV REG 24660

- Virtual address **24660** is sent to MMU
- MMU calculates that address **24660** belongs to virtual page 6 (=between 24 KB ~ 28 KB: 24576 ~28672)
- MMU checks memory map such that virtual page 6 is not in physical space yet. (page fault)
- When a page fault occurs, operating system
  - Picks one of physical frame
  - Write back to disk and then
  - Fetch the page to the frame just freed.
  - Change the memory map.



Virtual Memory with Paging

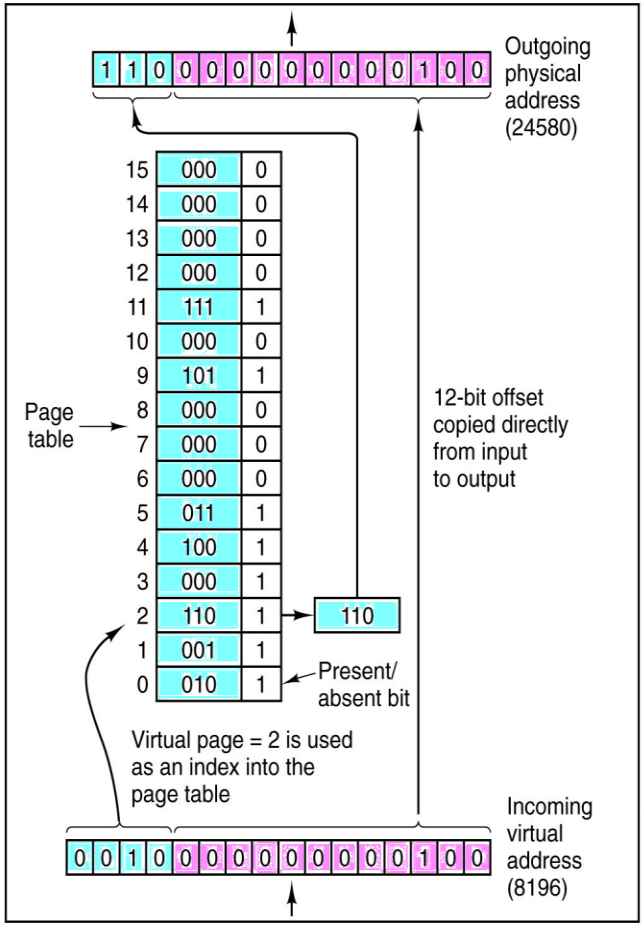


Virtual Memory with Paging

How to map virtual address into physical address by MMU with Previous Example

- The incoming 16 bit address is split into 4-bit page number and 12bit offset.
- With 4 bits page number, $2^4 = 16$ page number is available.
- The 4-bit page number is used as an index into the page table,
- With 12-bits offset, we can address $2^{12} = 4KB$ within a page.

Virtual Memory with Paging



Virtual Memory with Paging

How to map virtual address into physical address by MMU with Previous Example

- The incoming 16 bit address is split into
  - 4-bit page number
  - 12-bit offset.
- With 4-bit page number, $2^4 = 16$ page numbers are available.
- The 4-bit page number is used as an index into the page table,
- With 12-bits offset, we can address $2^{12} = 4KB$ within a page.

Page Table

- The purpose of the page table is to map virtual pages onto page frames.
- Two major issues with using page table
  - The page table can be extremely large
  - The mapping must be fast

Page Table

The page table can be extremely large

Ex) A modern computer with 32-bit virtual address and 4 KB page size

- $2^{32} = 2^{22}$ KB virtual space
- $2^{22} / 4 = 1048576$ pages
- Need 1048576 entries in the page table
- Since each process has its own virtual space, each process needs its own page table.!!!

Page Table

The mapping must be fast

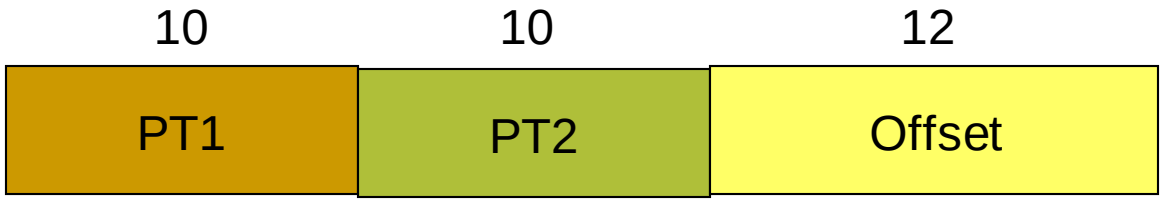
- Build a single table consisting of an array of fast hardware registers, with one entry for each virtual page, indexed by virtual page number. --- **very expensive**!!
- The page table can be entirely in main memory – all the hardware needs is a single register that points to the start of the page table --- needs one or more memory references to read page table entries.

Multilevel Page Table

- The basic idea of multilevel page table method is to avoid keeping all the page tables in memory all the time!!!

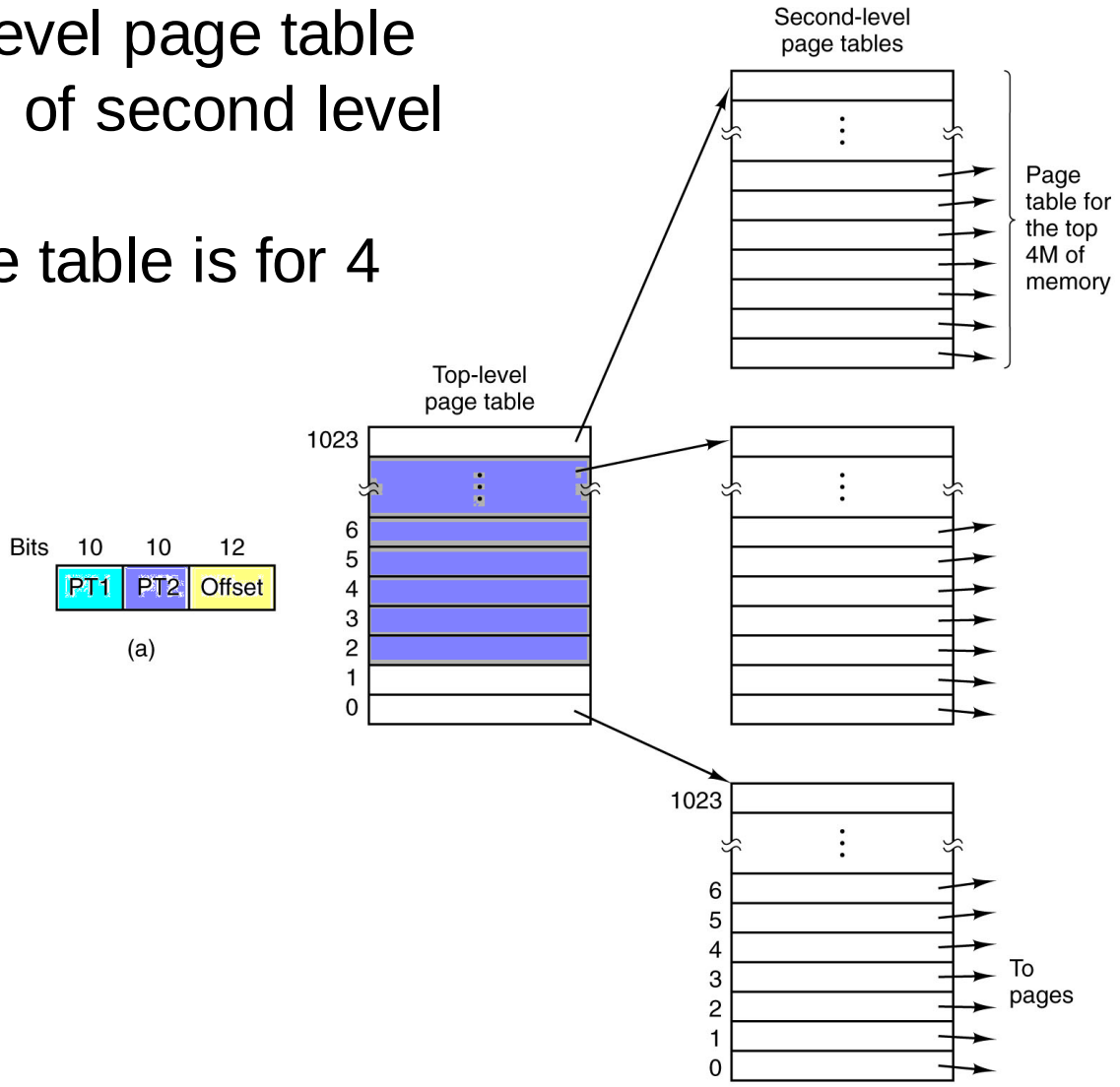
Ex) A system is using second level page tables

- The system generates 32-bit virtual address with a page size of 4KB.
- We have a 32-bit virtual address which is partitioned into
  - 10-bit PT1 field
  - 10-bit PT2 field
  - 12-bit offset field (for page size 4KB)



- Since offset is 12 bits, pages are 4KB and there are total 2^{20} of them.

- 2^{10} index for top level page table
- 2^{10} index for each of second level page table
- second level page table is for 4 KB×2^{10} = 4 MB



Multilevel Page Table

Ex) virtual address 403004_{16} (=$000000000100000000110000000000100_2$ = 4206596_{10})

- PT1 = $0000000001_2 = 1_{10}$
- PT2 = $0000000011_2 = 3_{10}$
- Offset = $000000000100_2 = 4_{10}$

Multilevel Page Table

- The MMU uses PT1 to index into the top-level page table and obtain entry 1 (0000000001)
  - (physical address between 4M ~ 8M-1).
- Then the MMU uses PT2 to index into the second-level page table and obtain entry 3 (0000000011)
  - (Corresponds to address 12KB ~ 16KB-1 within 4M ~ 8M-1 chunk)
    address between 4M+12K ~ 4M+16K–1
    = 4206592 ~ 4210687
- If the page is in the memory, the page frame number taken from the second-level page table is combined with the offset (4) to construct a physical address.

Preview

- Page Replacement Algorithms
  - Optimal Algorithm
  - First In First Out (FIFO)
  - Not Recently Used Algorithm (NRU)
  - The Least Recently Used (LRU)
- Modeling Page Replacement Algorithm
  - Belady's Anomaly
  - Stack Algorithm
    - Model for Stack Algorithm
    - Property of Stack Algorithm
- Design Issues for Page System
- Segmentation
  - Segmentation Implementation
- Segmentation with Paging
  - Segmentation with Paging (MULTICS)
  - Segmentation with Paging (Pentium: self study)

Page Replacement Algorithms

- When a page fault occurs, OS needs to do the following steps
  1. Choose a victim page currently allocated in a page frame
  2. If the page has been modified in the memory, rewrite it to the disk.
  3. A page is allocated into the page frame which was used by the victim page
  4. Change page table.

  The first step depends on the replacement algorithm

Page Replacement Algorithms

- A process's Memory access can be characterized by a list of page number
- This list is called the **reference string**.
- A paging system can be characterized by three items
  1. The reference string of the executing process
  2. The page replacement algorithm
  3. The number of page frames available in memory

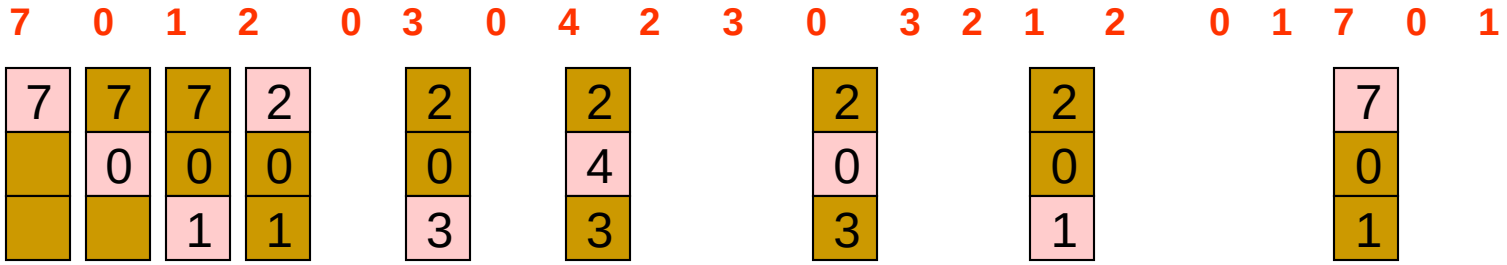
Page Replacement Algorithms

Optimal Algorithm

- Replace the page that will not be used for the longest period of time in the future.
- Optimal algorithm always guarantees the lowest possible page-fault rate for a fixed number of frames
- Unfortunately, there is no such an optimal replacement algorithm, as it requires future knowledge of the reference string.

Page Replacement Algorithms

Optimal Algorithm



9 page Faults

Page Replacement Algorithms

Not Recently Used (NRU)

- When a page fault occurs, the operating system inspects all the pages and classifies them into four groups based on the page table information (Modified bit, Reference bit).
  - Class 0: Not referenced, not modified
  - Class 1: not referenced, modified
  - Class 2: referenced, not modified
  - Class 3: referenced, modified
- The NRU algorithm removes a page at random from the lowest numbered nonempty class.

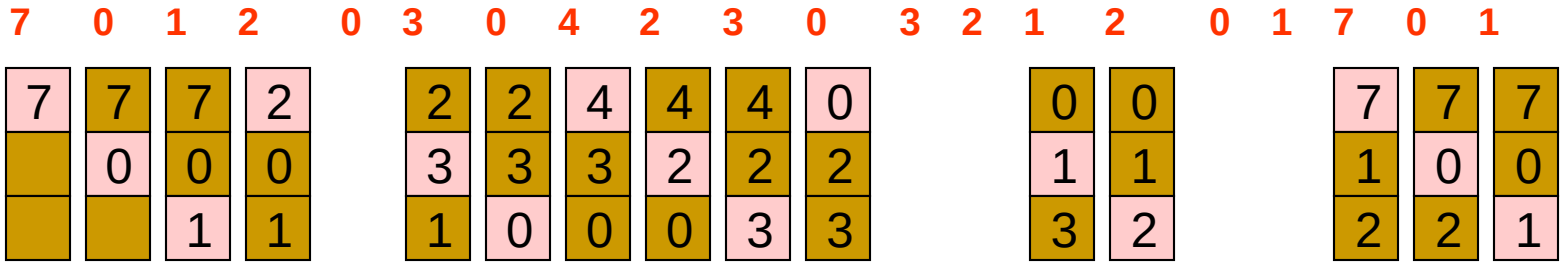
Page Replacement Algorithms

First In First Out (FIFO)

- Replace the oldest page frame
- The FIFO algorithm is easy to understand and easy to program.
- However, its performance is not always good.

Page Replacement Algorithms

First In First Out



15 Page Faults

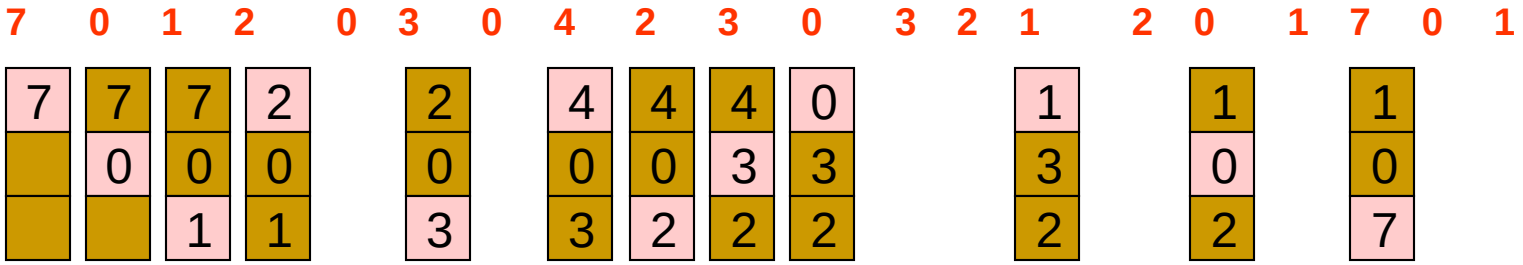
Page Replacement Algorithms

The Least Recently Used

- Replace the page that has not been used for the longest period of time
- This is the optimal page replacement algorithm looking backward in time.
- LRU algorithms works quite well but it may require substantial hardware assistance to keep track of the information.

Page Replacement Algorithms

Least Recently Used



12 Page Faults

Example Page Replacement Algorithms

Question:

(a) If FIFO page replacement is used with five page frames and eight pages, how many page faults will occur with the reference string 135732345051740 if the five frames are initially empty? Use a table showing the details of what pages are in the five page frames along with the reference string and mark it when a page fault occurred.

(b) Repeat the question for LRU

(c) Repeat the question for Optimal Algorithm.

Example Page Replacement Algorithms

(a) FIFO

1	3	5	7	3	2	3	4	5	0	5	1	7	4	0
1	1	1	1	1	1	1	4	4	4	4	4	4	4	4
	3	3	3	3	3	3	3	3	0	0	0	0	0	0
		5	5	5	5	5	5	5	5	5	1	1	1	1
			7	7	7	7	7	7	7	7	7	7	7	7
					2	2	2	2	2	2	2	2	2	2
x	x	x	x		x		x		x		x			

FIFO yields 8 page faults

Example Page Replacement Algorithms

(b) LRU

1	3	5	7	3	2	3	4	5	0	5	1	7	4	0
1	1	1	1	1	1	1	4	4	4	4	4	4	4	4
	3	3	3	3	3	3	3	3	3	3	3	7	7	7
		5	5	5	5	5	5	5	5	5	5	5	5	5
			7	7	7	7	7	7	0	0	0	0	0	0
					2	2	2	2	2	2	1	1	1	1
x	x	x	x		x		x		x		x	x		

LRU yields 9 page faults

Example Page Replacement Algorithms

(c) Optimal Algorithm

1	3	5	7	3	2	3	4	5	0	5	1	7	4	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	3	3	3	3	3	4	4	4	4	4	4	4	4
		5	5	5	5	5	5	5	5	5	5	5	5	5
			7	7	7	7	7	7	7	7	7	7	7	7
					2	2	2	2	0	0	0	0	0	0
x	x	x	x		x		x		x					

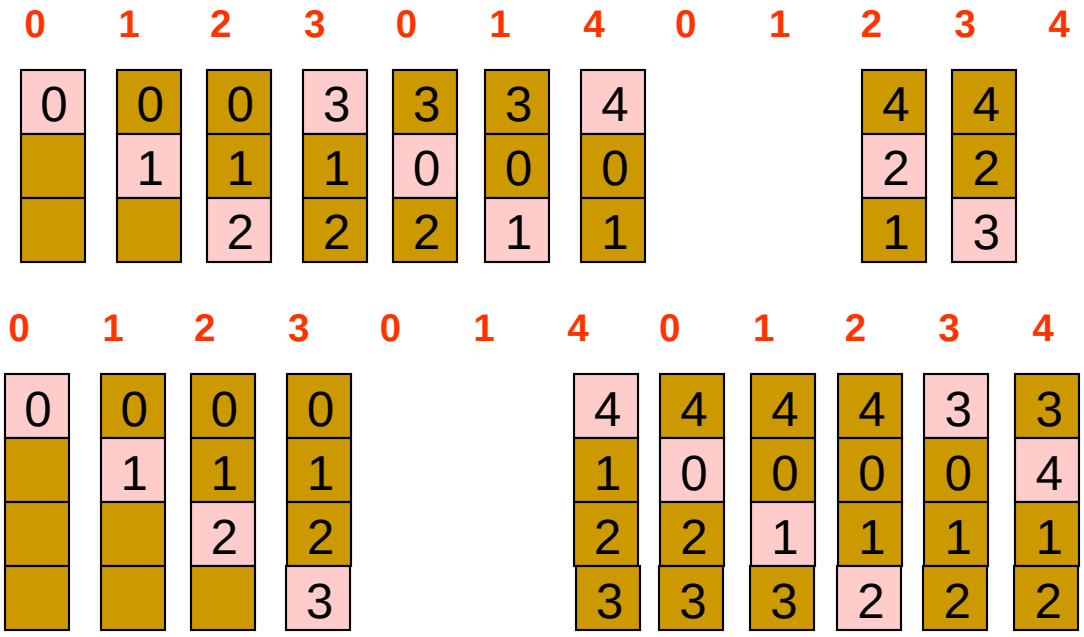
FIFO yields 7 page faults

Modeling Page Replacement Algorithm

Belady's Anomaly

- Intuitively, it might seem that the more page frames the memory has, the fewer page faults a program will cause. This is not always the case – Belady's Anomaly
- Belady shows the case with FIFO cases; the more page frames the memory has, the more page faults a program causes.

Modeling Page Replacement Algorithm (Belady's Anomaly)



Modeling Page Replacement Algorithm (Stack Algorithm)

A paging system can be characterized by three items

1. The reference string of the executing process
2. The page replacement algorithm
3. The number of page frames available in memory

Stack Algorithm

Model for Stack Algorithm

- Maintains an internal array M that keeps track of the state of memory.
- M has as many as *n* virtual memory pages.
- Top *m* entries contain all the pages currently in the memory (page frames).
- Bottom *n* - *m* entries contains all the pages that have been referenced once but have been page out and are not currently in memory

Stack Algorithm
(Model for Stack Algorithm)

Properties of the model

1. When a page is referenced, it is always moved to the top entry in M.
2. If the page referenced was already in M, all pages above it move down one position.
3. A transition from within the box to outside of it corresponds to a page being evicted from the memory.
4. The pages that were below the referenced page are not moved.

Stack Algorithm
(Property for Stack Algorithm)

Property of Stack Algorithm

- If any page replacement algorithm has the following property, we call it the stack algorithm

$$M(m,r) \subseteq M(m+1,r)$$

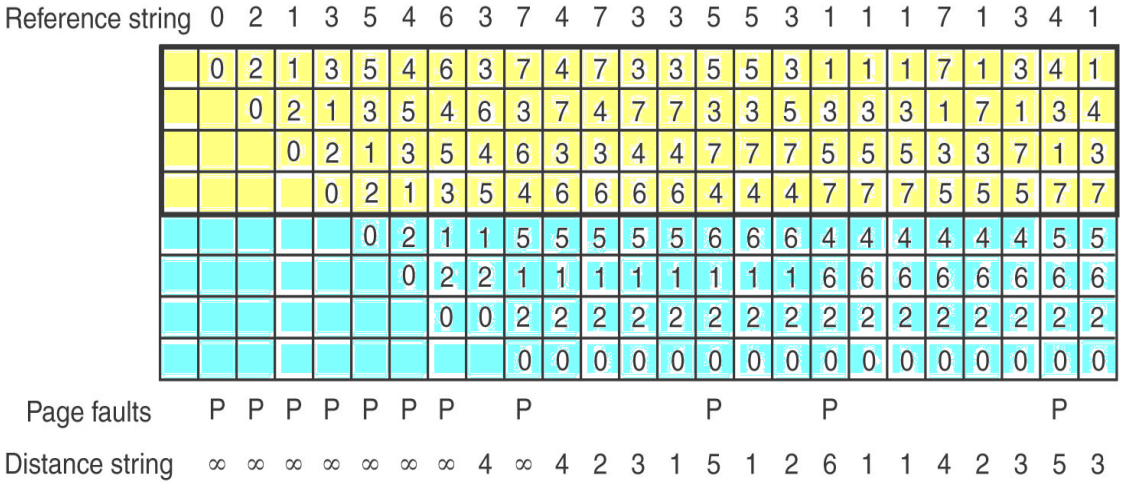
- If we increase memory size by one page frame and re-execute the process, at every point during the execution, all the pages that were present in the first run are also present in the second run, along with one additional page
- Stack algorithm does not suffer from Belady's Anomaly.

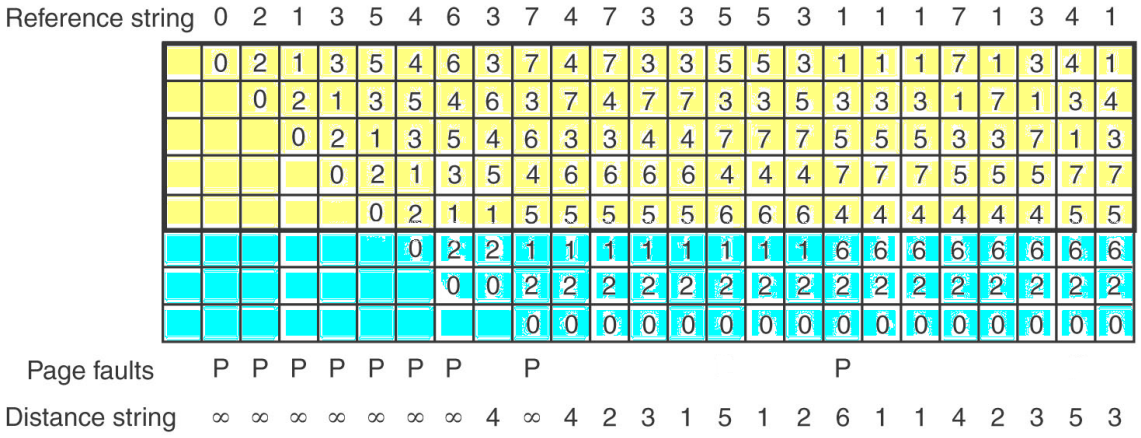
Stack Algorithm

LRU algorithm with the model

1. The reference strings: 0 2 1 3 5 4 6 3 7 4 7 3 3 5 3 1 1 1 7 2 3 4 1
2. The page replacement algorithm: LRU
3. The number of page frames available in memory:
    1. Virtual spaces: 8 pages
    2. Physical spaces: 4 pages or
    3. Physical spaces: 5 pages

Stack Algorithm





Design Issues for Paging System
(Local vs. Global Allocation)

Local versus Global allocation Policies

- With multiple processes competing for frames, we can classify page replacement algorithms into two categories: global replacement and local replacement.
- Global replacement allows a process to select a replacement frame from the set of all frames.
- Local replacement requires that each process select from only its own set of allocated frame

Design Issues for Paging System (Load Control)

Load Control

- Even though we use best replacement algorithm with global allocation policies, if the combined working sets of all processes exceed the capacity of memory, thrashing can be expected.
- Possible concern is reducing the degree of multiprogramming – swap out some of the processes onto the disk.

Design Issues for Paging System (Page Size)

- Smaller page size means smaller internal fragmentation.
- Larger size will cause more unused program to be in memory than a small page size.
- Small pages means that programs will need many pages (hence a large page tables).
- Many small pages results in wasting time for the seek and rotation.
- Some systems need to load the page table into the hardware registers to execute. – many small pages, need more page table loading time.

Design Issues for Paging System (Page Size)

Mathematical Analysis

S: average size of process (byte)

P: the size of page (byte)

E: Each page table entry needs (byte) per page.

$\frac{S}{P}$: Average number of pages per process

$\frac{S}{P} \times E$: Average page table space (for each process)

$\frac{P}{2}$: the wasted memory (in average) in the last page of the process

Design Issues for Paging System (Page Size)

$\frac{S}{P}$: Average number of pages per process

$\frac{S}{P} \times E$: Average page table space (for each process)

$\frac{P}{2}$: the wasted memory (in average) in the last page of the process

Total overhead by page table and internal fragmentation loss is:

$$Overhead(P) = \frac{SE}{P} + \frac{P}{2}$$

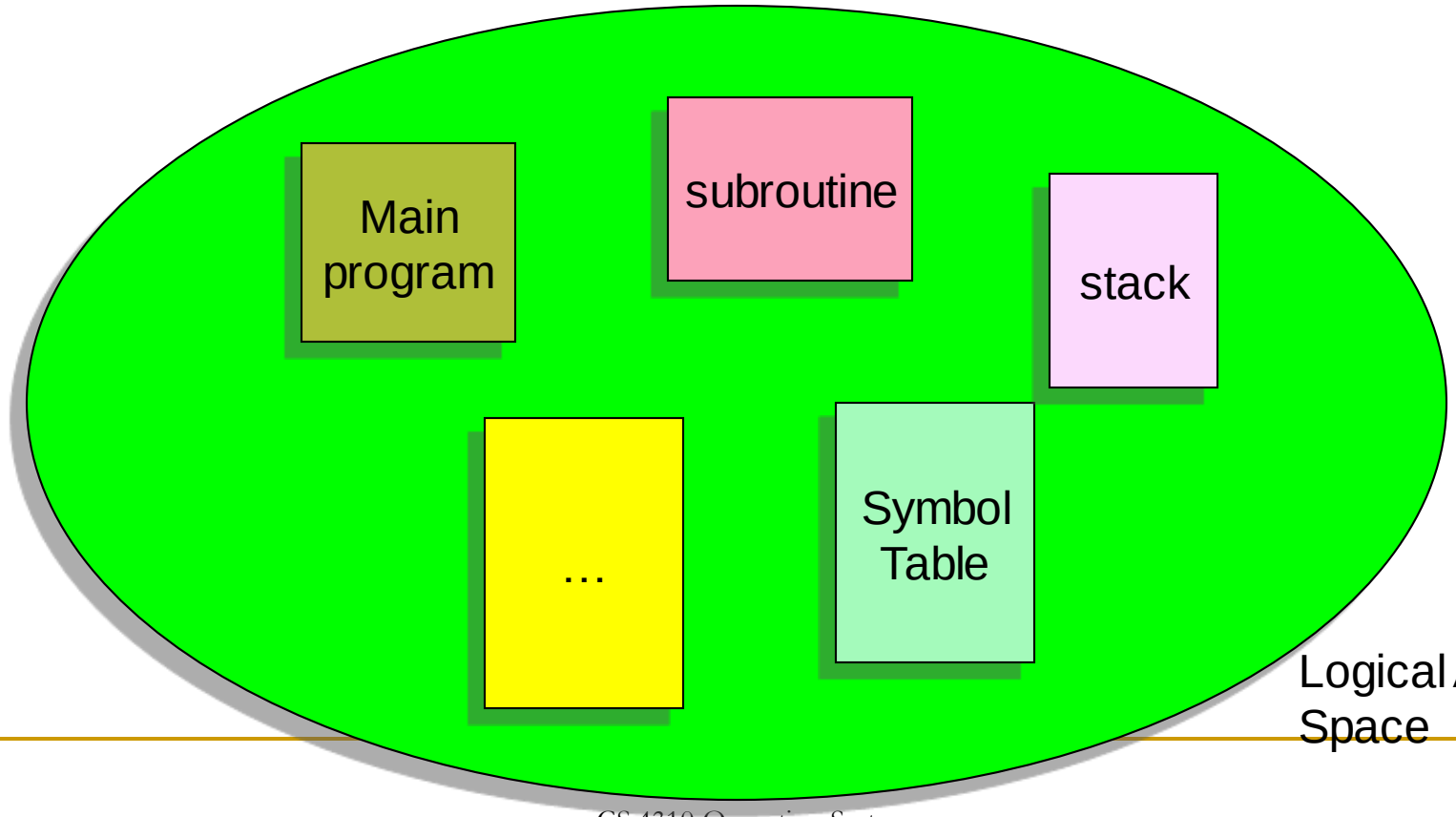
$$Overhead'(P) = -\frac{SE}{P^2} + \frac{1}{2} = 0$$

$$P = \sqrt{2SE} : optimal \text{ page size}$$

Segmentation

- Segmentation is a logical entity, which the programmer knows and uses as a logical entity.
- Segmentation is a memory management scheme that supports the user's view of memory!!

Segmentation
(User's View of Memory)



Segmentation

- A logical address space is a collection of segments. Each segment has a name and a length.
- Each segment might have a different size.
- A segment might contain a procedure, an array, a stack, or a collection of scalar variables; but usually it does not contain a mixture of different types
- Normally, the user program is compiled, and the compiler automatically constructs segments reflecting the input program.

Segmentation

Ex) Pascal compiler might create separate segments for following:

- The global variables;
- The procedure call stack, to store parameters and return address
- The code portion of each procedure or function
- The local variables of each procedure and function

Segmentation Implementation Hardware

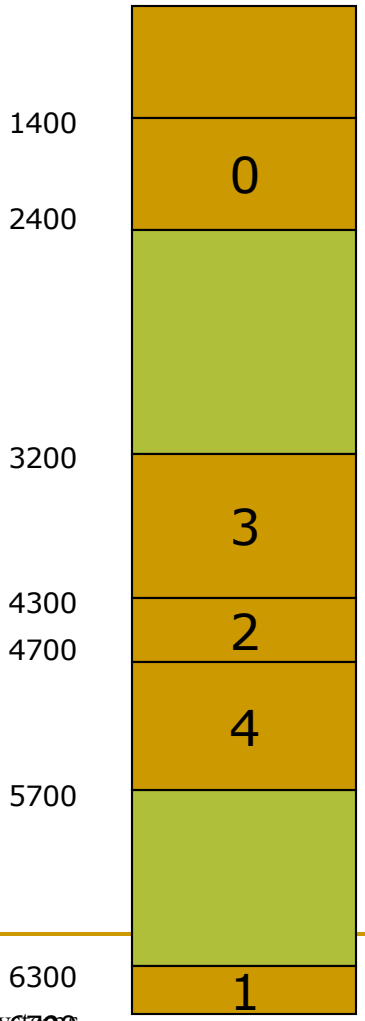
	limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Ex) A reference to segment 3 to byte 852 is mapped to

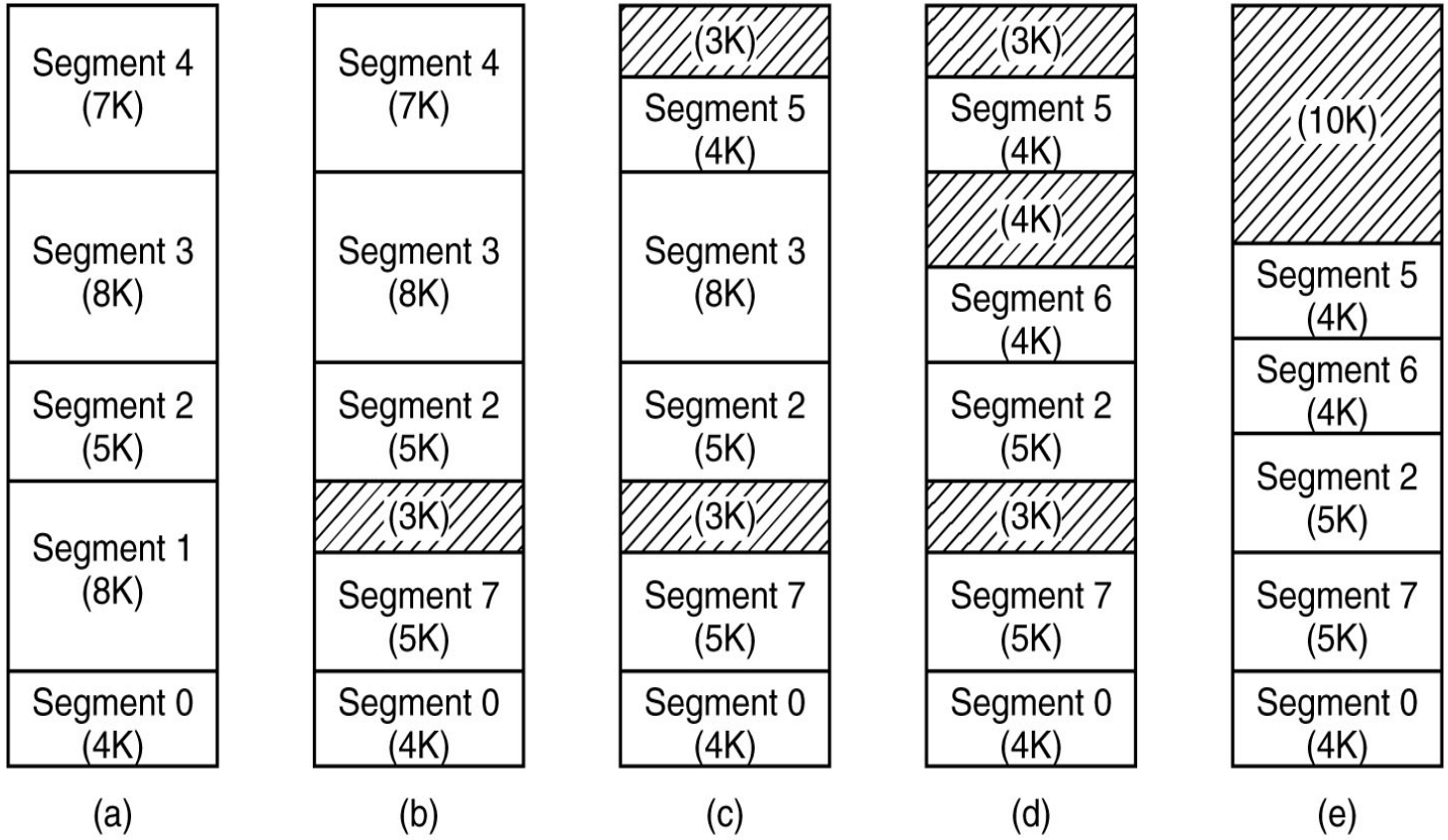
3200 + 852 = 4052

Ex) A reference to segment 0 to byte 1222

would result in error



Disadvantages with Segmentation
(External Fragmentation)



Segmentation with Paging

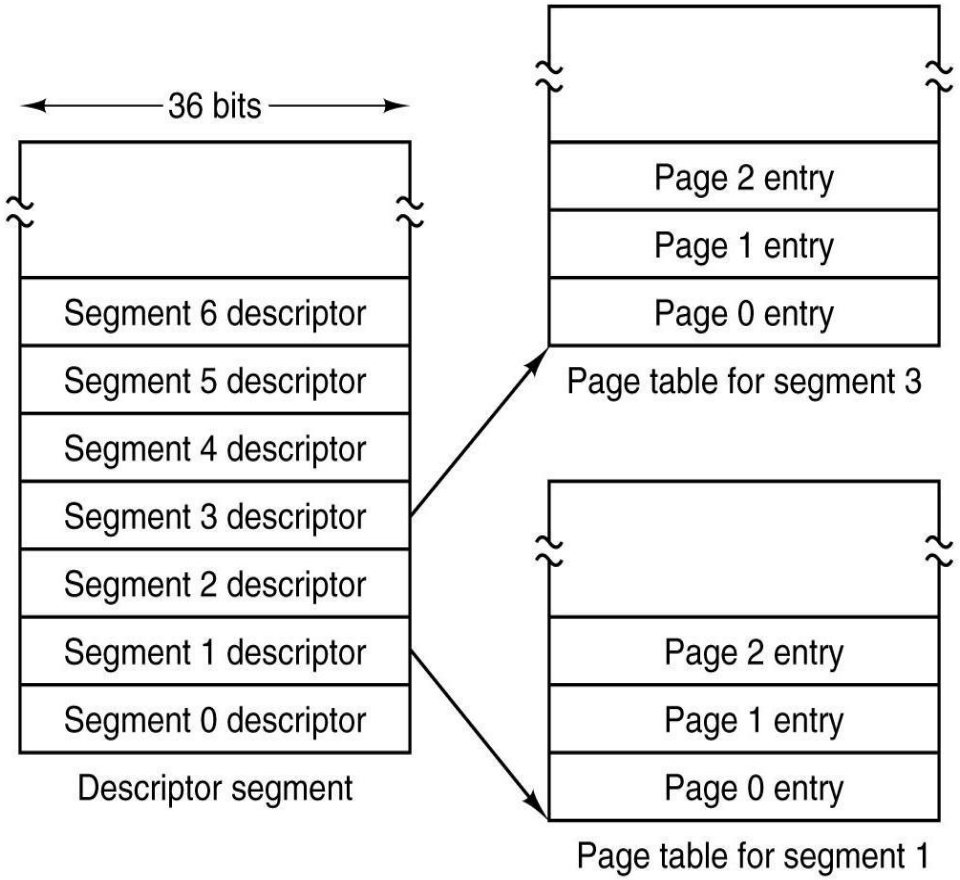
Reason for segmentation with paging

- If the segments are large, then keeping them in the physical memory might be wasting memory space.
- If a segment's virtual space is larger than physical space, it is not even possible to keep them in the physical memory.
- A solution is the segmentation with paging – each segment is divided into pages.

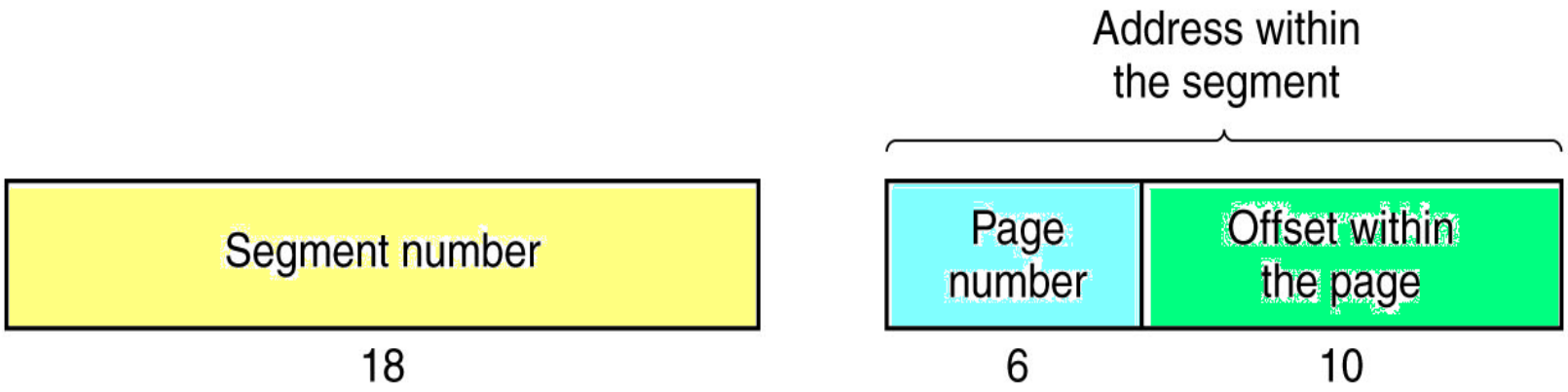
Segmentation with Page (MULTICS)

- For each program a virtual memory of up to 2^{18} segments are allowed.
- Each program has segmentation table

Segmentation with Page (MULTICS)

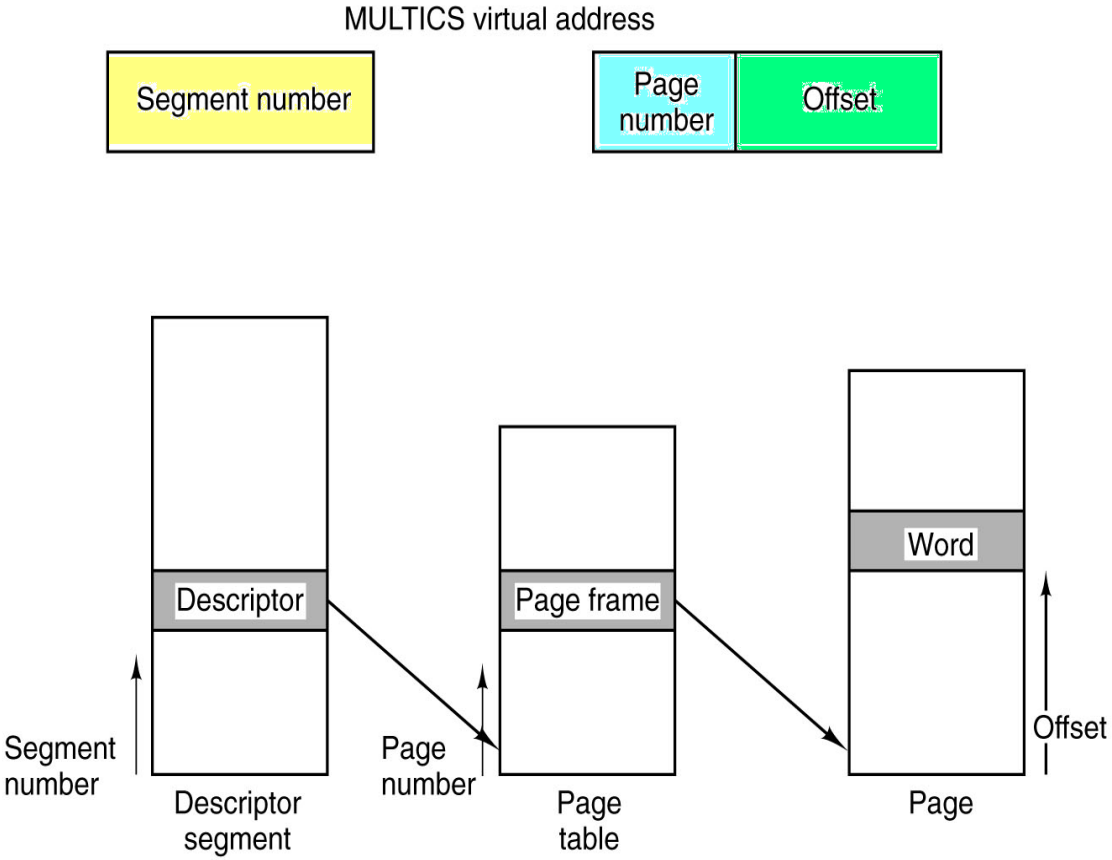


Segmentation with Page (MULTICS)



34 bit Virtual Address

Segmentation with Page (MULTICS)



Segmentation with Page (MULTICS)

When a memory reference occurs, the following takes place

1. The segment number is address for the segment descriptor in the segmentation table
2. Check whether segment's page table is in or not. If it is not in, segmentation fault occurs (OS need to handle it). If it is in go to next
3. Check whether the page is in the memory or not. If the page is not in, page fault (OS need to handle it). If the page is in memory, go to next step
4. The offset in the virtual memory address is added with page frame number. It is sent through address bus to main memory (physical space)
5. Read/write

Preview

- File Systems
- File
  - File Name
  - File Structure
  - File Types
  - File Access
  - File Attributes
  - File Operation
- Directories
- Directory Operations
- File System Layout
- Implementing File
  - Contiguous Allocation
  - Linked List Allocation
  - Linked List Allocation with FAT (File Allocation Table)
- Implementing Directories

File Systems

Three essential requirements for long term information storage

1. To store a very large amount of information
2. To store information permanently
3. To share the information with multiple processes

File Name

<file_name>.<extension>

length: 255

UNIX: case sensitive

DOS: not case sensitive

File Naming

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

Typical file extensions.

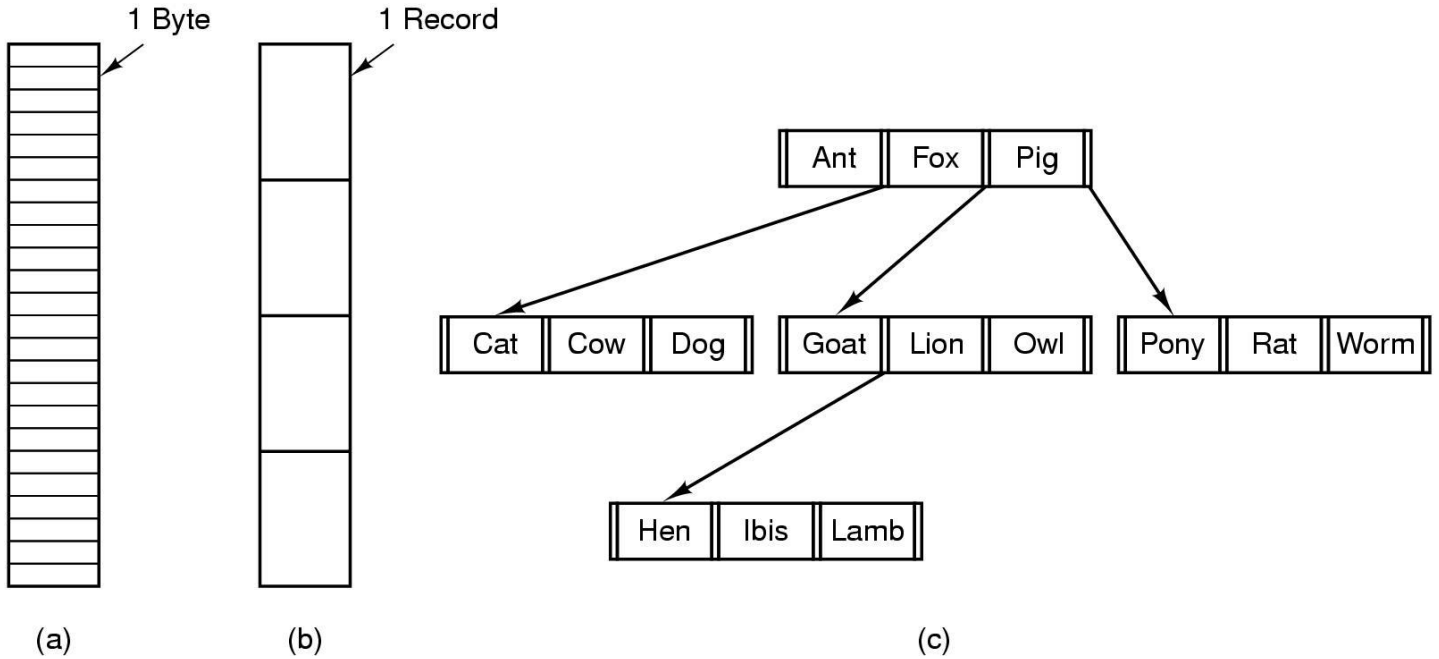
File Structure

Files can be structured

- File is an un-structured sequence of bytes – OS does not know what is in the file (UNIX, Window)

- File is a sequence of records
  - Used in the 2nd Generation main frame computer
  - 80-column punched card – files consist of 80-character records.
- File consists of a tree of records

File Structure



Three kinds of files

a. byte sequence
b. record sequence
c. tree

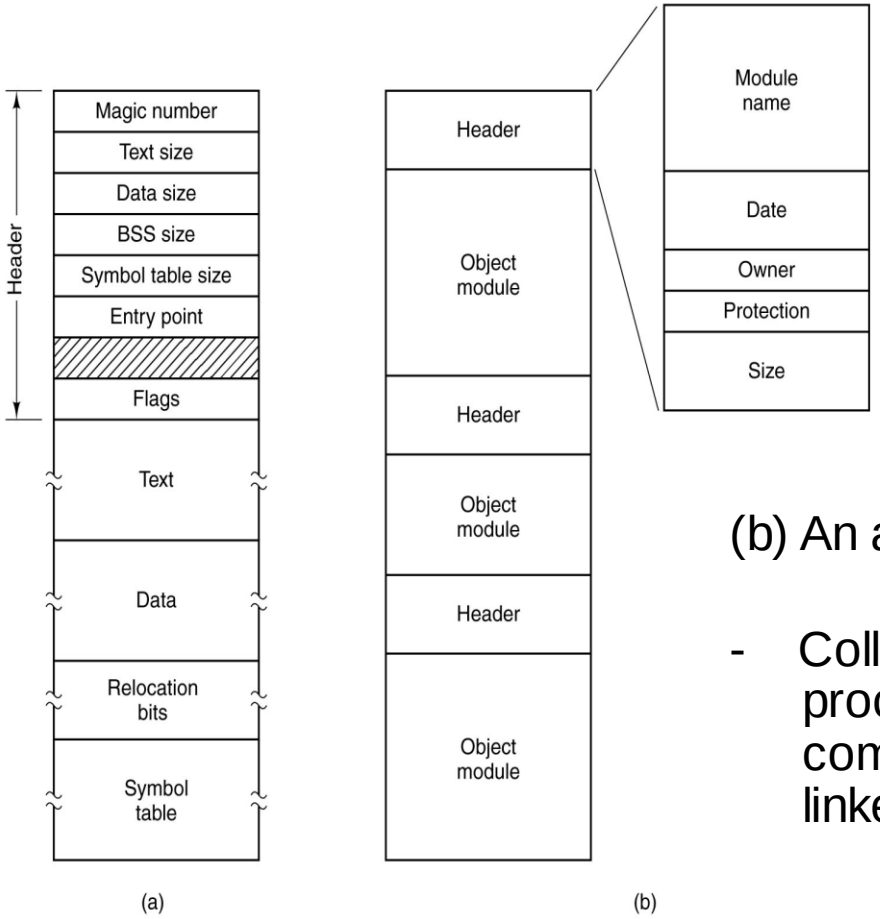
File Types

- **Regular Files** – for user's information
  - ASCII file – line of text
    - A line is terminated by carriage return or special character
  - Binary file
    - executable file - OS can execute a file if it has the proper format
    - archive file – consists of a collection of library procedures compiled but not linked
- **Directories** – system files for maintaining the structure of the file systems

Executable Files

- Executable Files
  - Header – all information regarding execution
    - Starts with a so-called ***magic number***, identifying the file as an executable file
    - the address where execution starts
  - Text
  - Data

File Type (Binary Files)



(a) An executable file from an early version of UNIX

- Magic number to identify the file as an executable file

(b) An archive from UNIX

- Collection of library procedures (modules) compiled but not linked

File Access

- Sequential Access –
  - read all bytes/records from the beginning
  - cannot jump around, could rewind or back up
  - *Read operation* changes pointer to the next location for read. (magnetic tape)
- Random Access –
  - bytes/records read in any order
  - *Seek operation* can move the pointer to the location (disk)

File Attributes

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Possible file attributes

File Operations

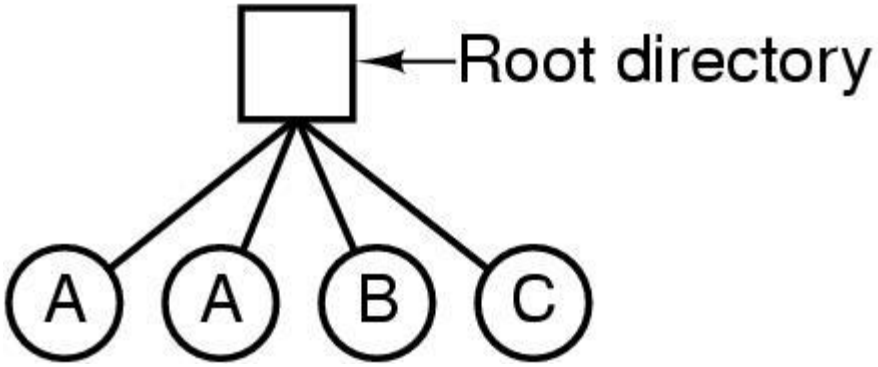
1. Create
2. Delete
3. Open
4. Close
5. Read
6. Write
7. Append
8. Seek
9. Get attributes
10. Set Attributes
11. Rename

Directories

To keep track of files, file systems have directories which themselves are files.

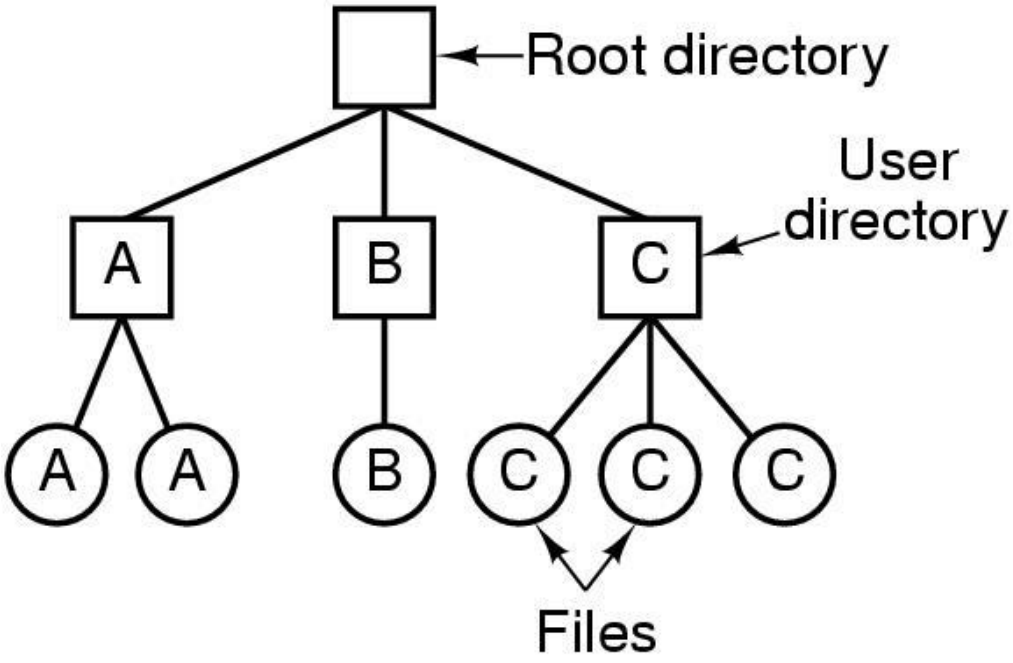
- Single-level directory System – one directory containing all files
- Two-level directory System – Giving each user a private directory
- Hierarchical directory System – User can create a directory to group their files in a logical way

Single-Level Directory Systems



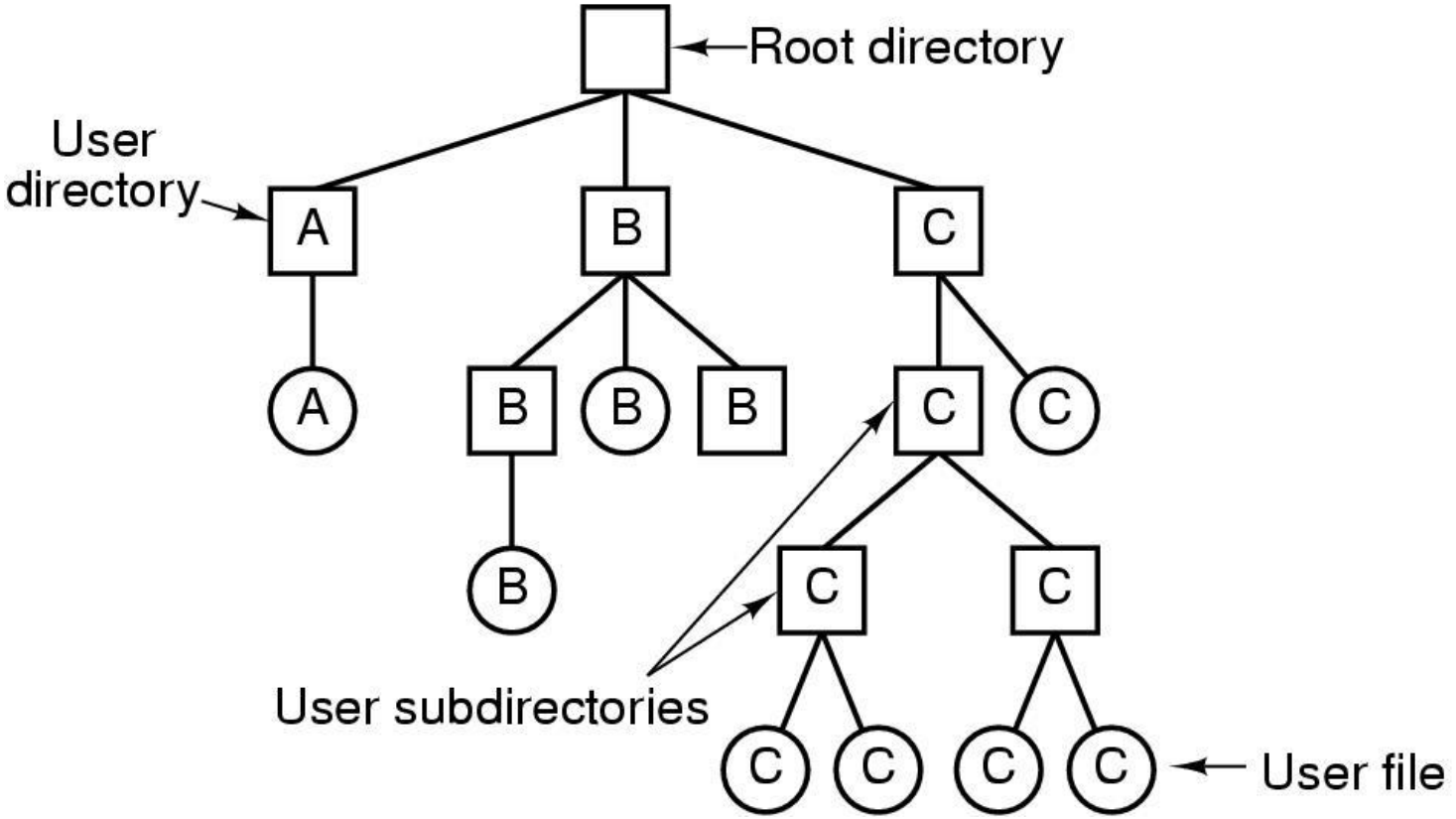
- A single level directory system
  - contains 4 files
  - owned by 3 different people, A, B, and C

Two-level Directory Systems



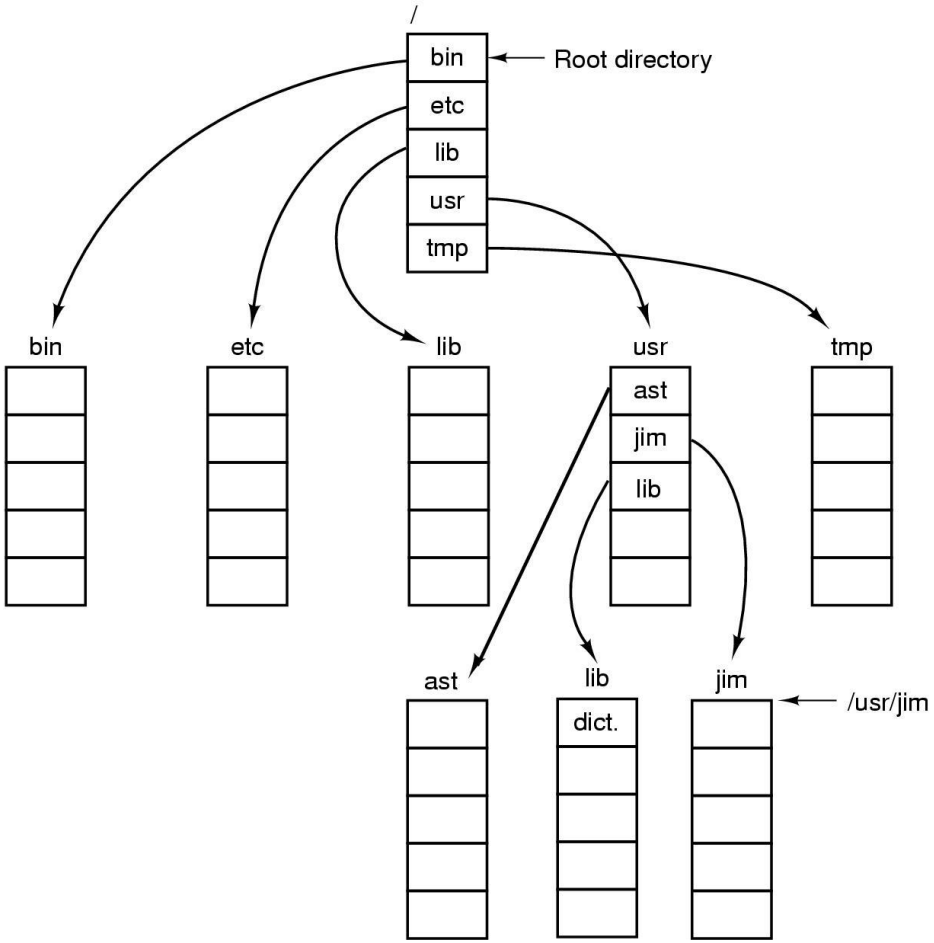
Letters indicate *owners* of the directories and files

Hierarchical Directory Systems



A hierarchical directory system

Path Names

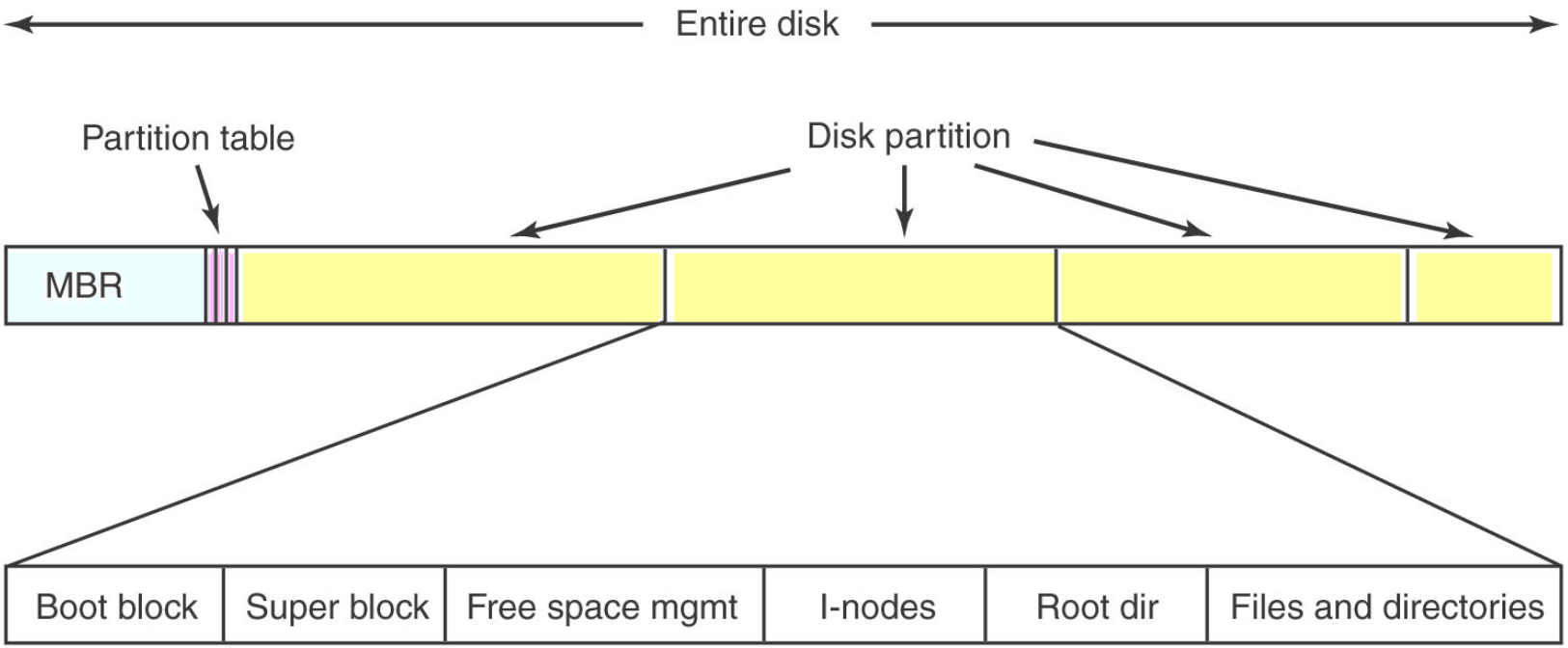


A UNIX directory tree

Directory Operations

1. Create
2. Delete
3. Opendir
4. Closedir
5. Readdir
6. Rename
7. Link
8. Unlink

File System Layout



A possible file system layout

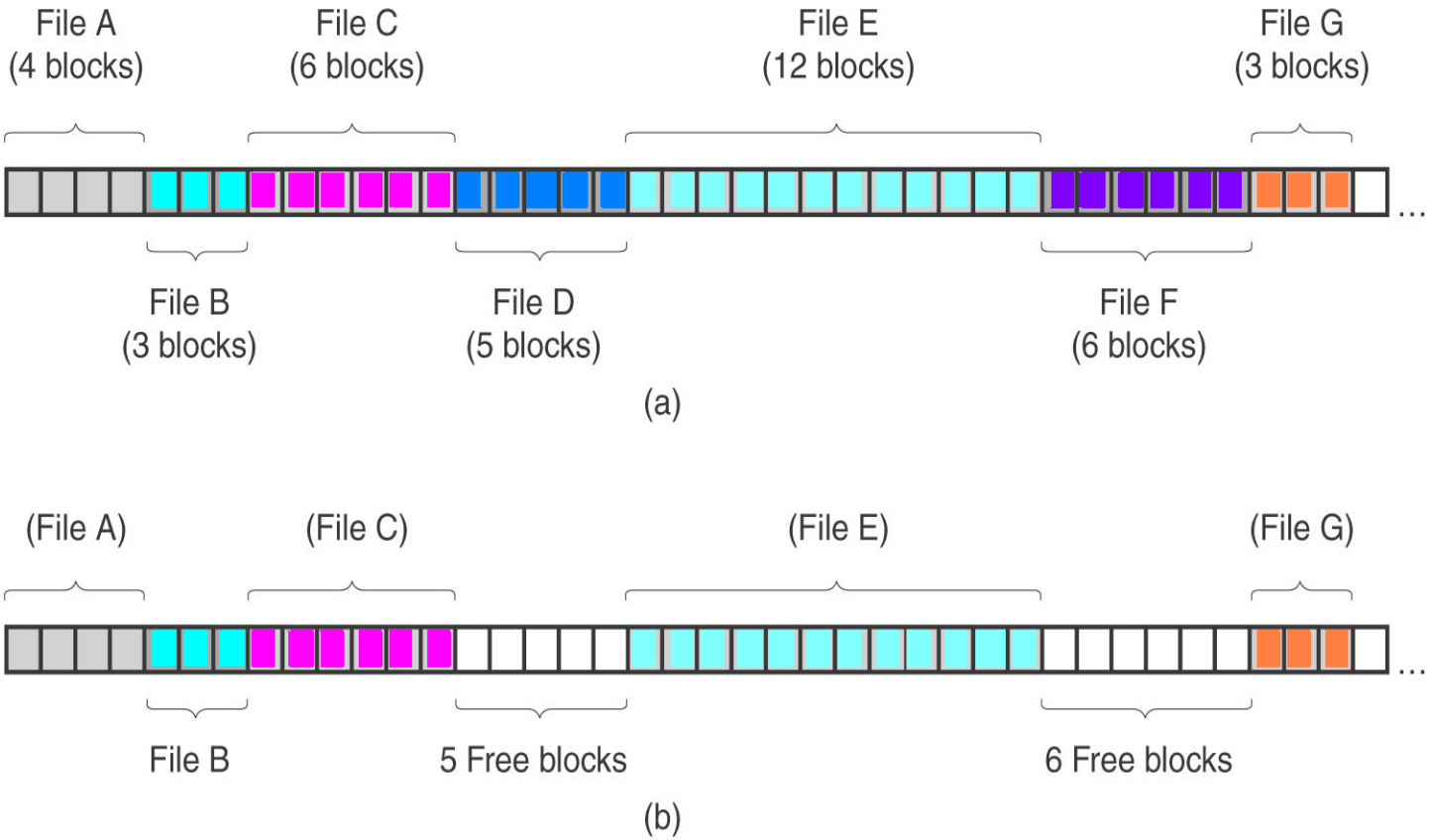
File System Layout

- When the Computer is Booted
  - The BIOS reads in and executes the MBR.
  - MBR (Master Boot Record)
    - Sector 0 of the disk
    - Used to boot the computer
    - Locates the active partition
    - Read in the first block (boot block) and execute it
    - The program in the boot block loads the OS contained in that partition (active partition)
  - Partition Table
    - Gives the starting and ending addresses of each table
    - One of the partitions is marked as active

File System Layout

- The System Layout of a disk partition varies strongly from file system to file system.
  - Super block
    - Key parameters about the file system, e.g. magic number to identify file system type, number of blocks in the file system…
  - I-node
    - An array of data structure, one per file, telling all about the file
  - Root directory
    - Top of the file system tree.

Implementing Files (Contiguous Allocation)



(a) Contiguous allocation of disk space for 7 files

(b) State of the disk after files *D* and *F* have been removed

Implementing Files (Contiguous Allocation)

Two significant advantages with contiguous file allocation

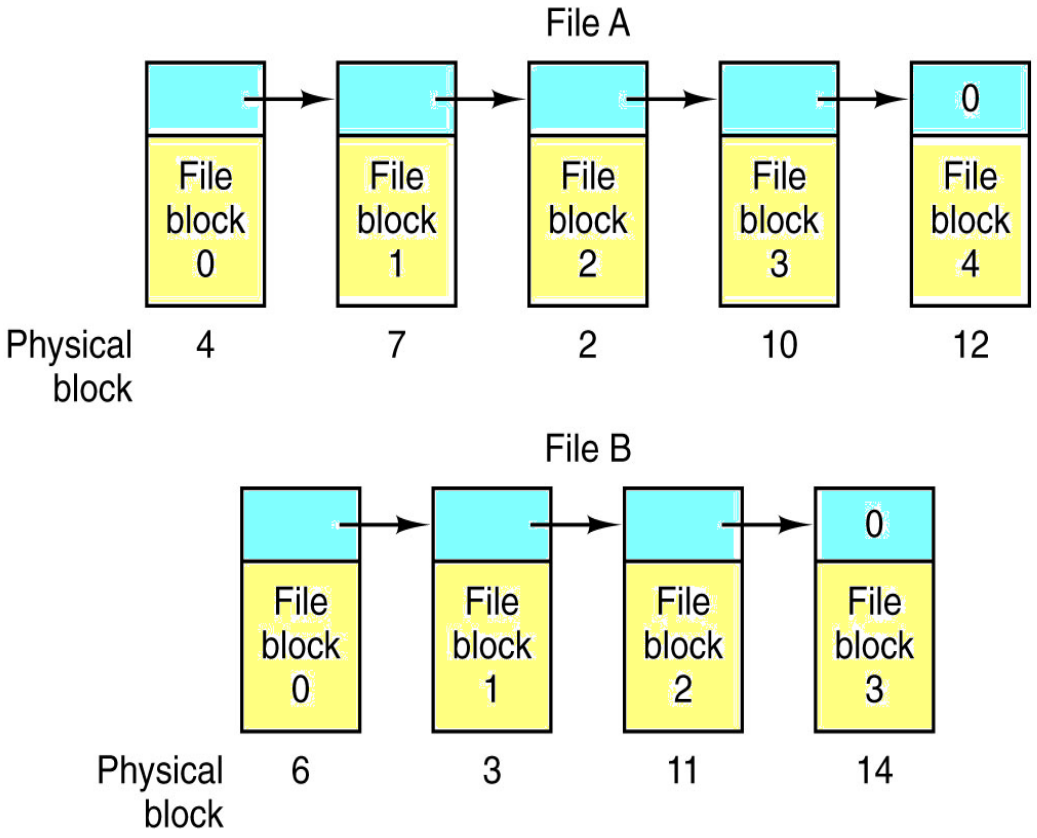
- **Simple implementation** because keeping track of where a file's blocks are is reduced to remembering two numbers:
  - disk address of the first block
  - the number of blocks in the file
- **The read performance is excellent**
  - only one seek is needed to read the entire file.

Implementing Files (Contiguous Allocation)

Drawback with the contiguous allocation

- Fragmentation
- In order to choose a hole for a new file, we need to know its (final) size. Otherwise, there is a problem.

Implementing Files (Linked List Allocation)



Storing a file as a linked list of disk blocks

Implementing Files (Linked List Allocation)

Disadvantage

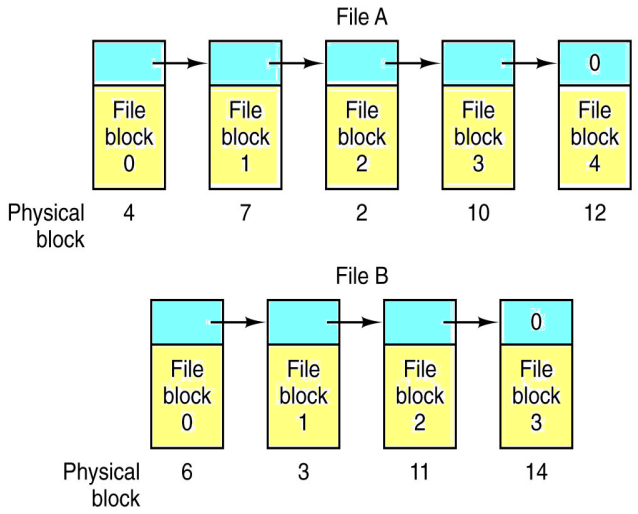
- To get to block n, the OS has to start at the beginning and read the n-1 blocks prior to it, one at a time.

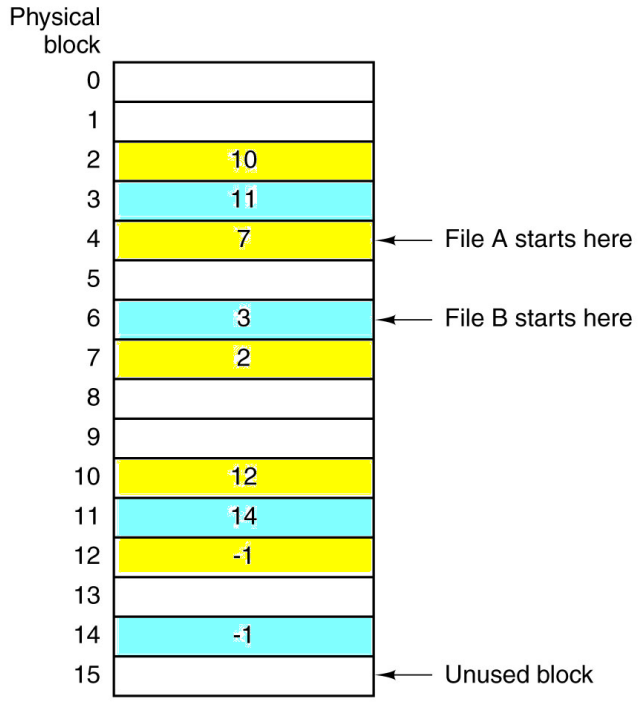
  $\Rightarrow$ SLOW!

- The pointer takes up a few bytes

  $\Rightarrow$ the amount of data storage in a block is no longer a power of 2

Implementing File
(Linked List Allocation with File Allocation Table)





Linked list allocation using a file allocation table in RAM

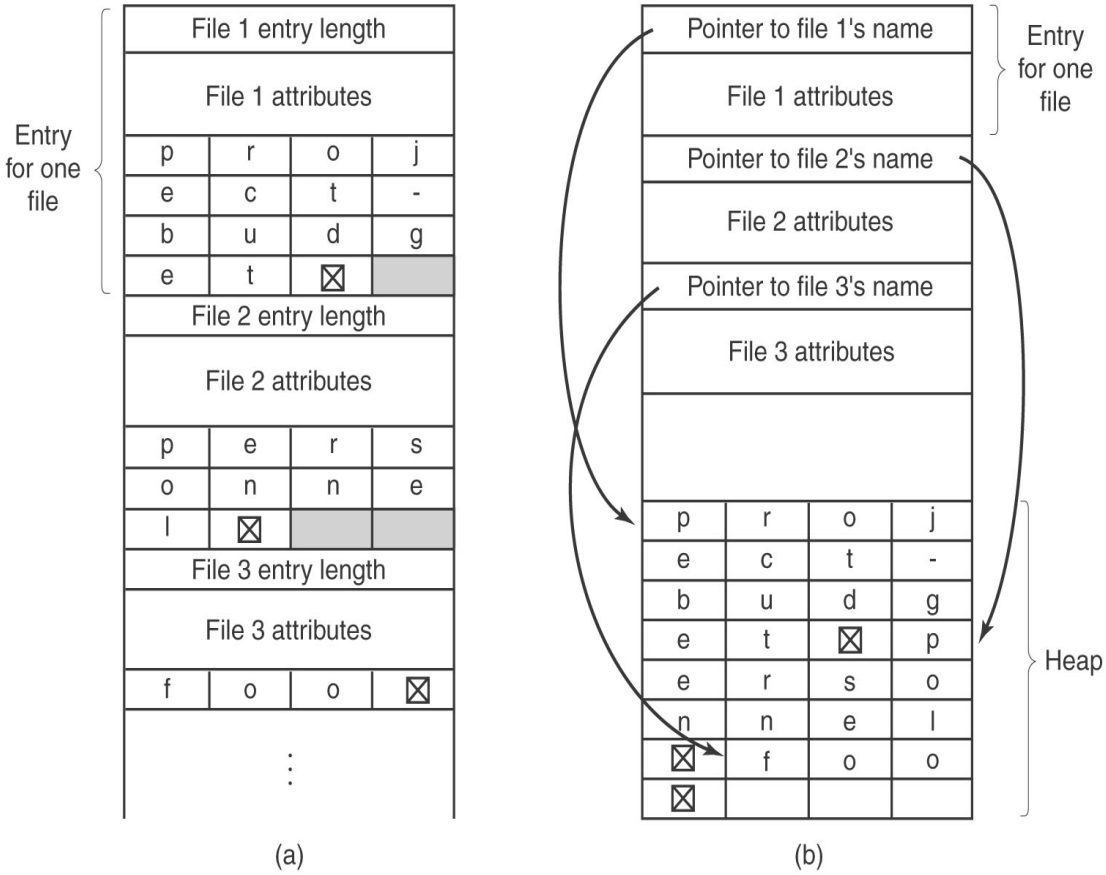
Implementing File
(Linked List Allocation with File Allocation Table)

- All link information for each file is written into the File Allocation Table (FAT) which is located in main memory.

Primary disadvantage

- The entire table must be in the memory all the time
  - Example: 20 GB Hard Drive with 1KB block size, needs 20 million entries for the FAT. If one entry takes 3 bytes, we need 60MB for the FAT.

Implementing Directories (long file names)



- Two ways of handling long file names in directory
  (a) In-line (b) In a heap

Implementing Directories

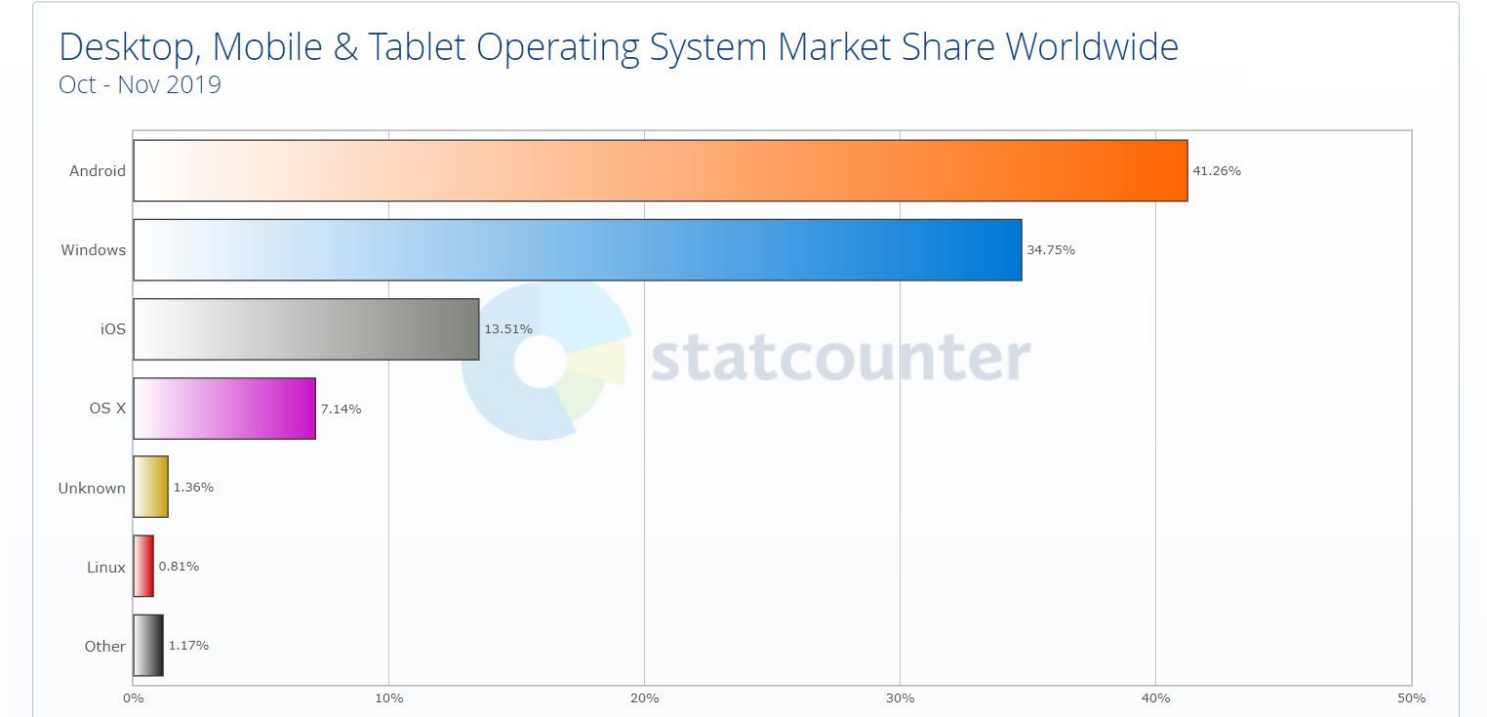
Searching Directories

- If a directory is designed in a linear way, for extremely long directory, linear searching can be slow.
- To speed up the search
  - Use hash table in each directory
  - Use cache

Chapter 9: Security

Android	Windows	iOS	OS X	Unknown	Linux
41.22%	34.83%	13.53%	7.1%	1.36%	0.81%

Desktop, Mobile & Tablet Operating System Market Share Worldwide - October 2019



https://gs.statcounter.com/os-market-share/desktop-mobile-tablet/worldwide/#monthly-201910-201911-bar

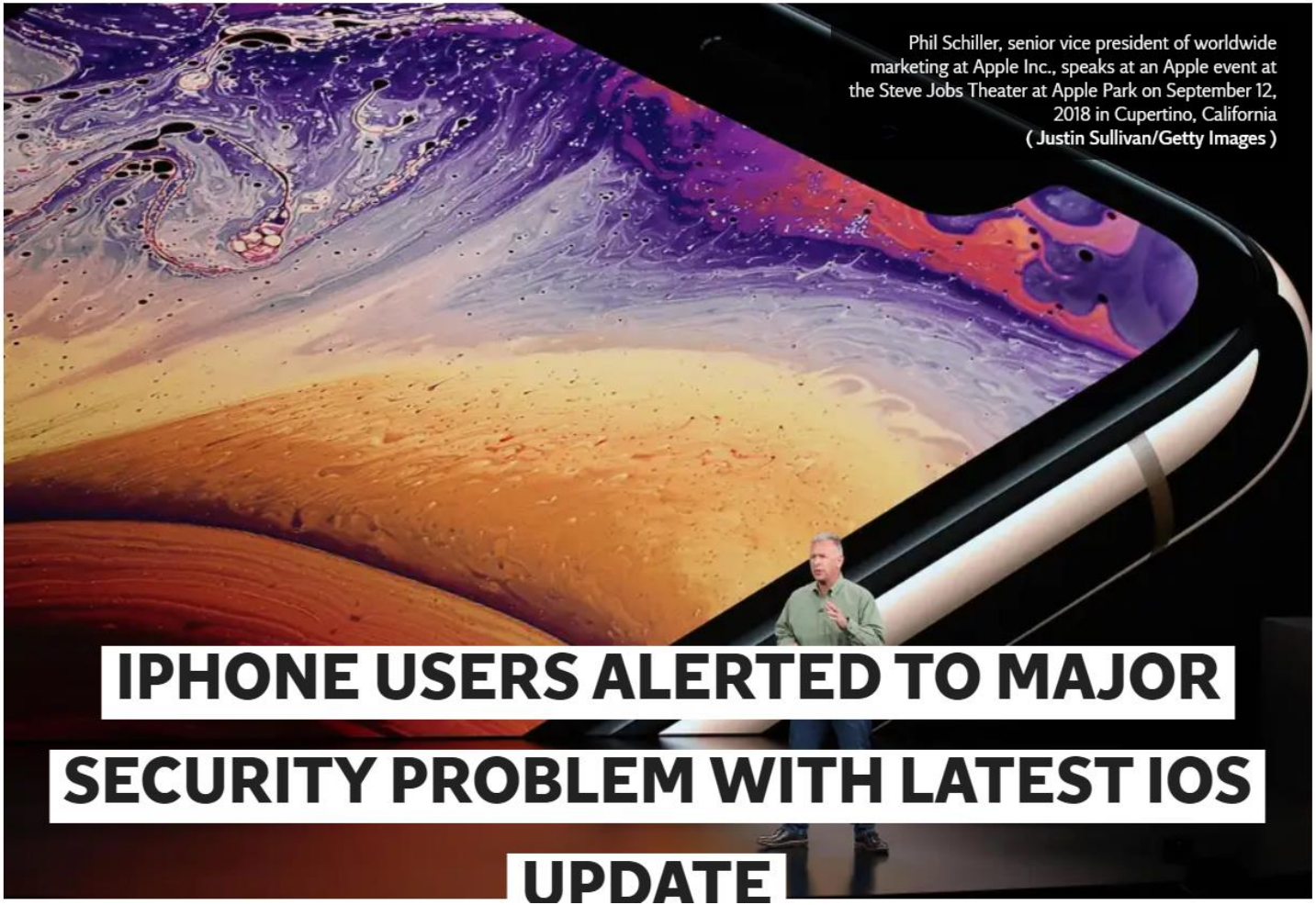
PREEMPTIVE STRIKES —

Google enlists outside help to clean up Android's malware mess

New App Defense Alliance tries solving longstanding Play Store malware problem.

LILY HAY NEWMAN, WIRED.COM - 11/9/2019, 3:30 AM

https://arstechnica.com/gadgets/2019/11/google-enlists-outside-help-to-clean-up-androids-malware-mess/



https://www.independent.co.uk/life-style/gadgets-and-tech/news/iphone-ios-update-124-latest-download-apple-ipad-software-security-bug-a9071361.html

https://www.cnn.com/2019/09/24/tech/microsoft-windows-security-threat/index.html

Chapter 9 Outline

- The Security Environment
- OS security
- Controlling access to resources
- Basics of cryptography
- Authentication
- Insider attacks
- Malware
- Defenses

The Security Environment

- **GOALS:**
  - **Confidentiality** is concerned with having data remain a secret.
  - **Integrity** ensures that unauthorized users should not be able to modify any data without owner's permission.
  - **Availability** promotes usability and prevents an entity from disturbing the system.

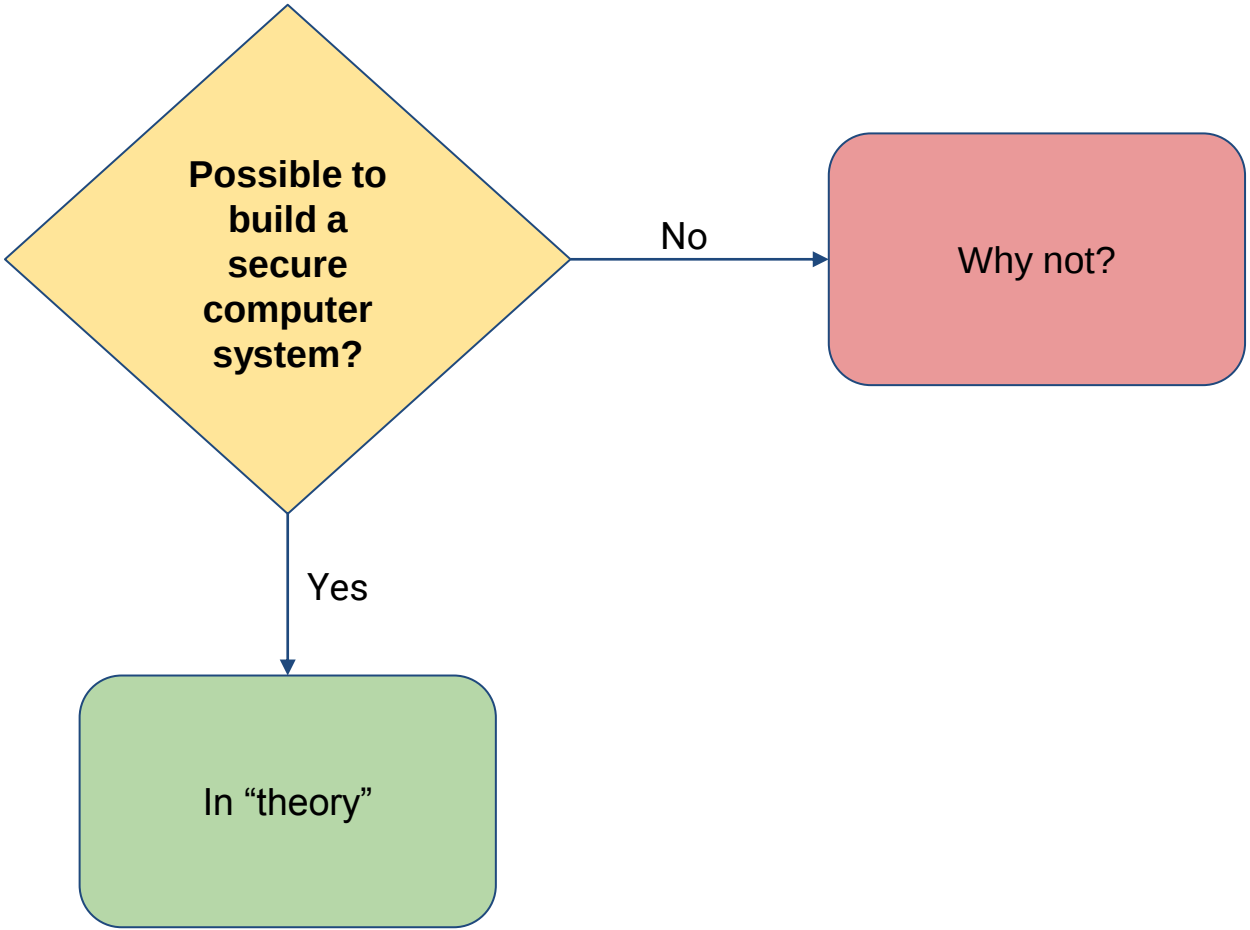
- **THREATS:**
  - Exposure of data
  - Tampering with data
  - Denial of Service

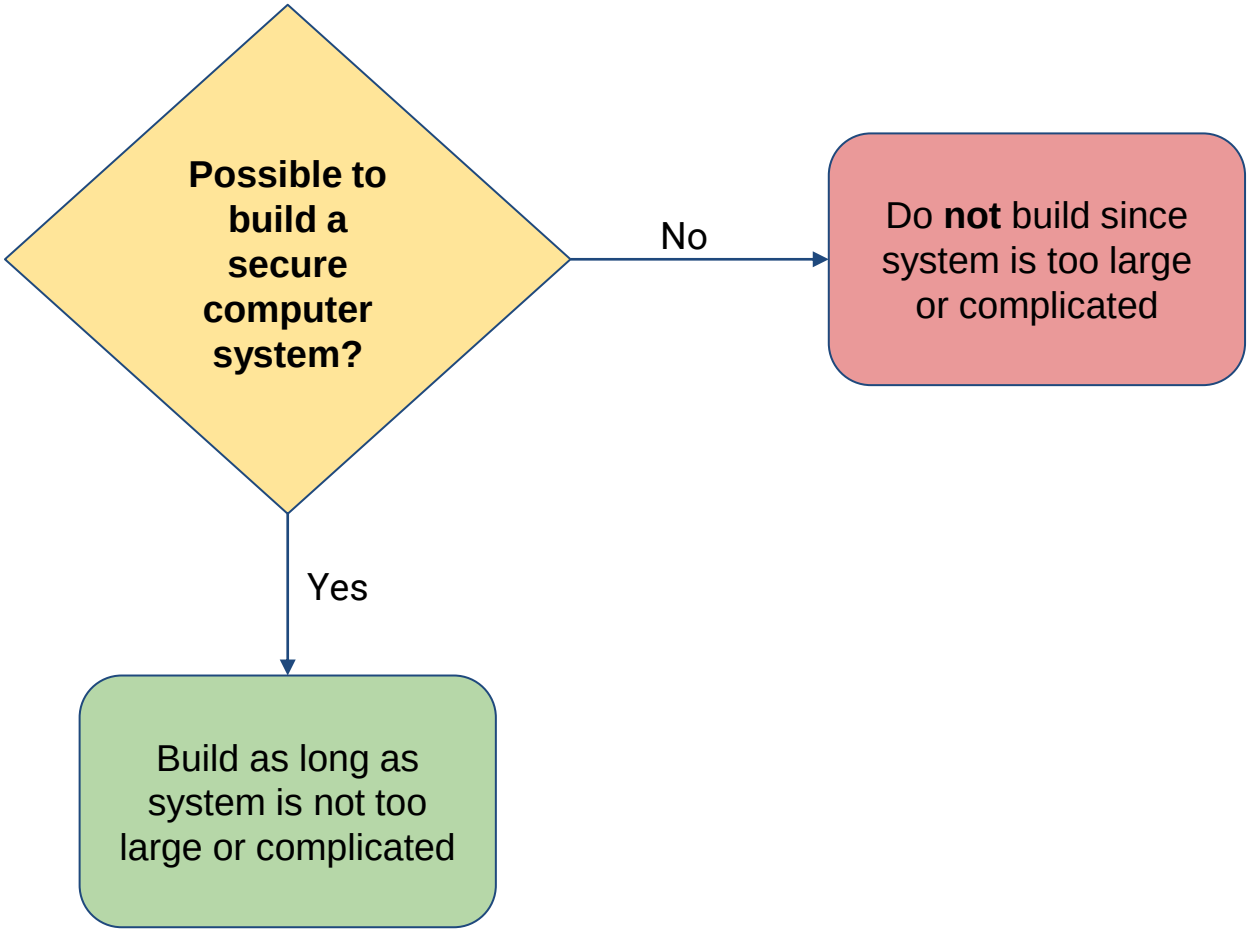
Bad Habits

- Setting PIN codes or passwords as something “easy” to remember:
  - PIN: `1111`
  - Password: `p@55w0rd`
- Choosing complicated passwords that require users to write them down.
- Keeping sensitive data on USB sticks.

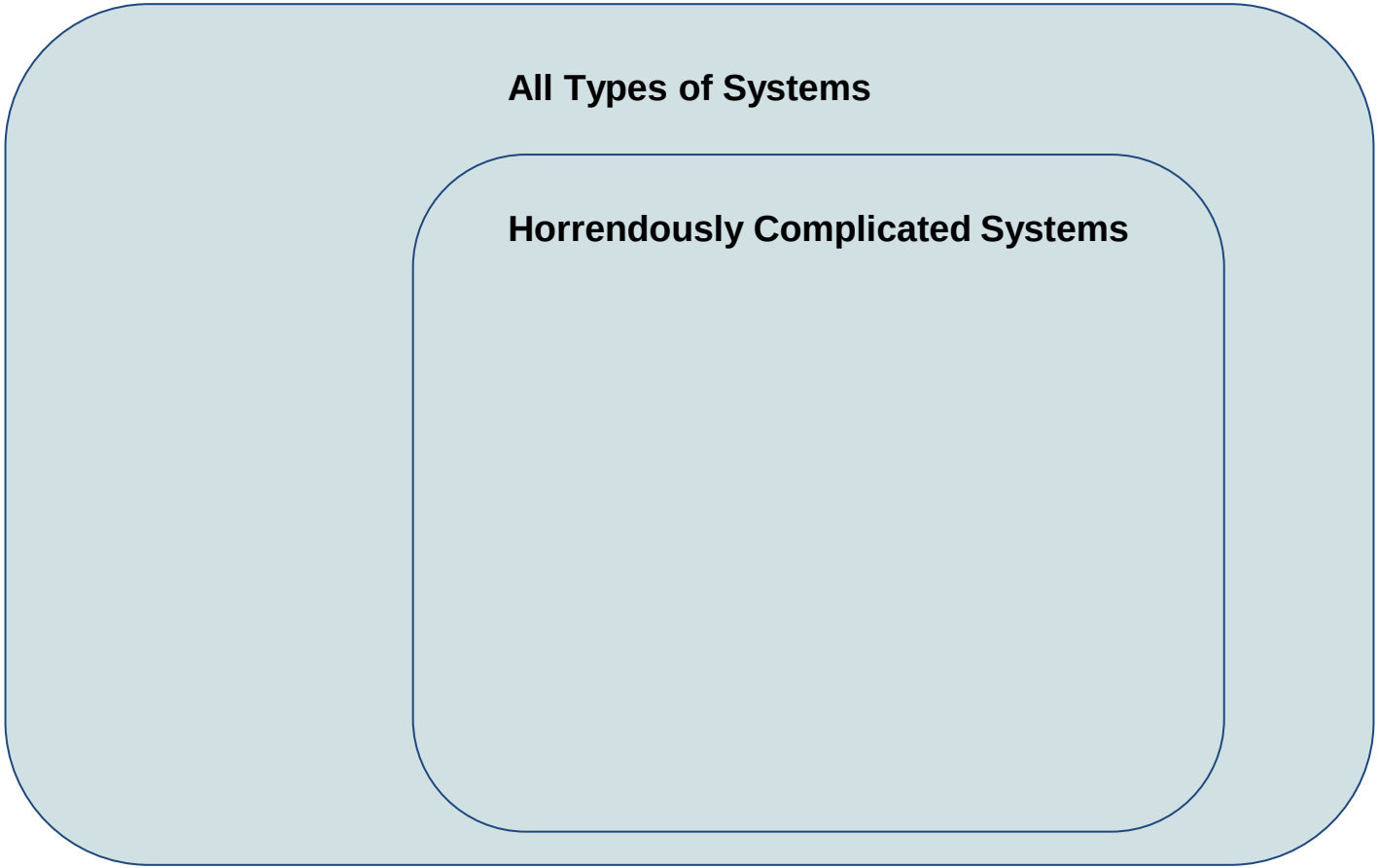
- Keeping personal credentials saved on guest computers.

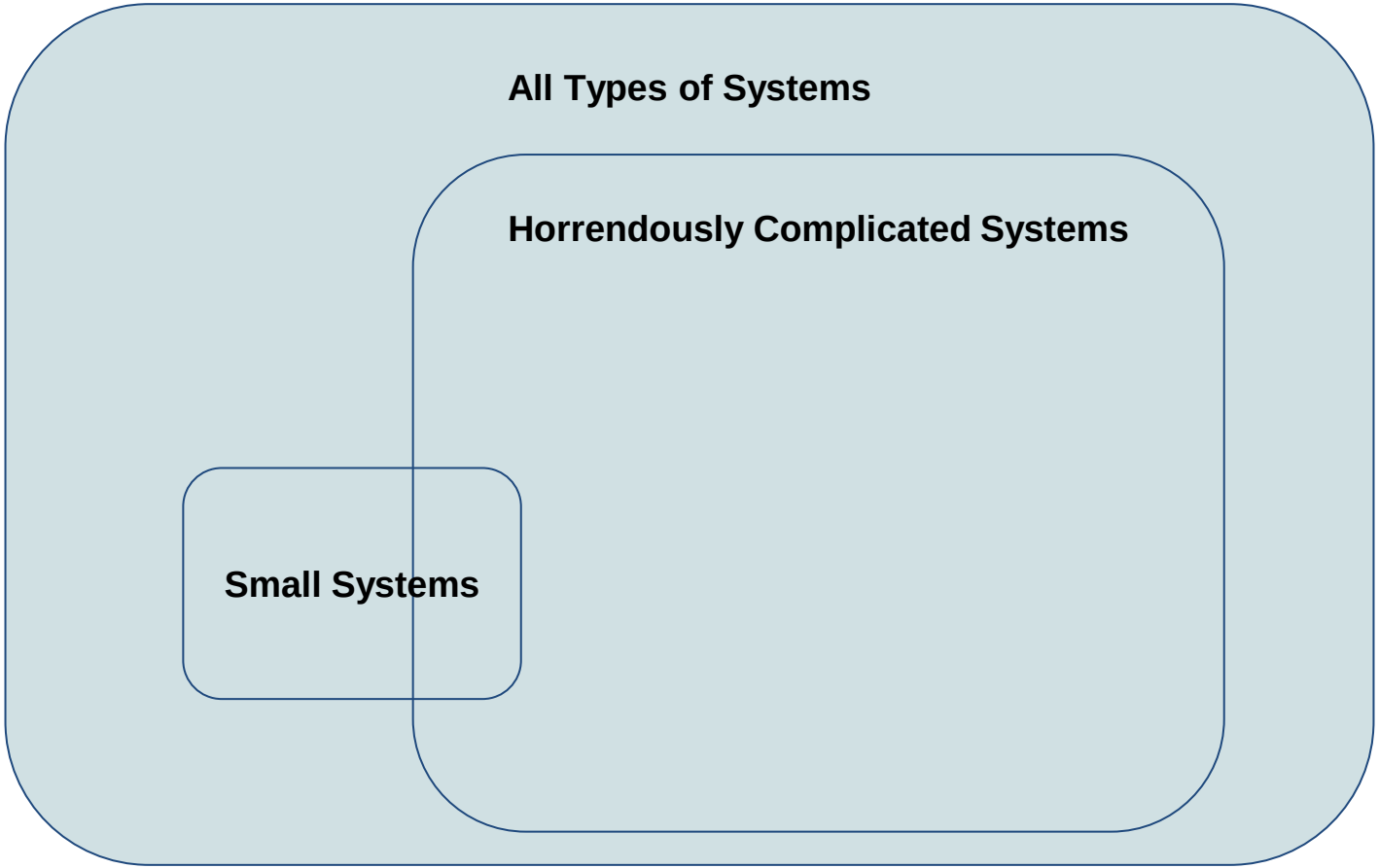
  (ex: classroom, library, workplace)

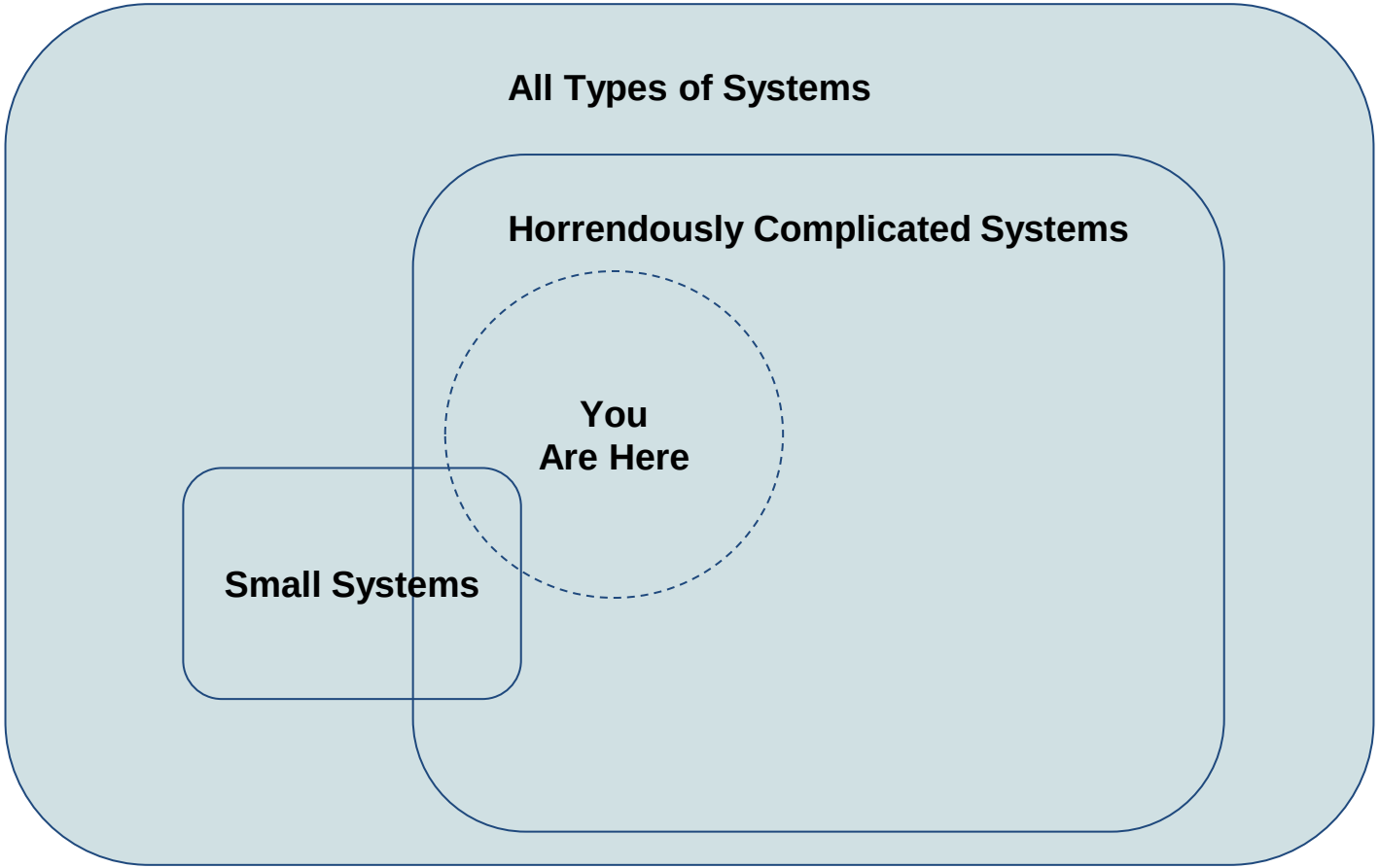

Possible to build a secure computer system?
No
Why not?
Yes
In “theory”


Possible to build a secure computer system?
No
Do **not** build since system is too large or complicated
Yes
Build as long as system is not too large or complicated

All Types of Systems


All Types of Systems
Horrendously Complicated Systems


All Types of Systems
Horrendously Complicated Systems
Small Systems


All Types of Systems
Horrendously Complicated Systems
You
Are Here
Small Systems

Can We Build Secure Systems?

- “In theory, yes”
- Computer systems today are complicated.
  - Current systems are not secure, but users have invested too much and will not throw them out
  - Only way to build a secure system is to keep it simple; features are the enemy of security!
- Misconception (courtesy of the respective marketing dept.)
  - more features, bigger features, and better features → happy customers
- Unintended consequences:
  - more complexity, code, bugs, and security errors.

Trusted Computing Base

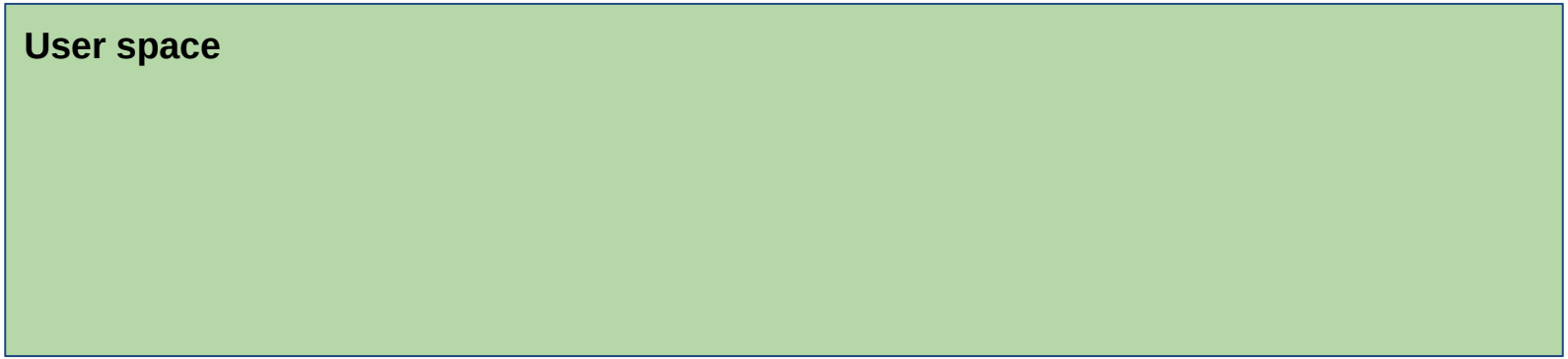
At the heart of every trusted system is a minimal **TCB** (**Trusted Computing Base**) consisting of the hardware and software necessary for enforcing all the security rules. If the trusted computing base is working to specification, the system security cannot be compromised, no matter what else is wrong.

Trusted Computing Base

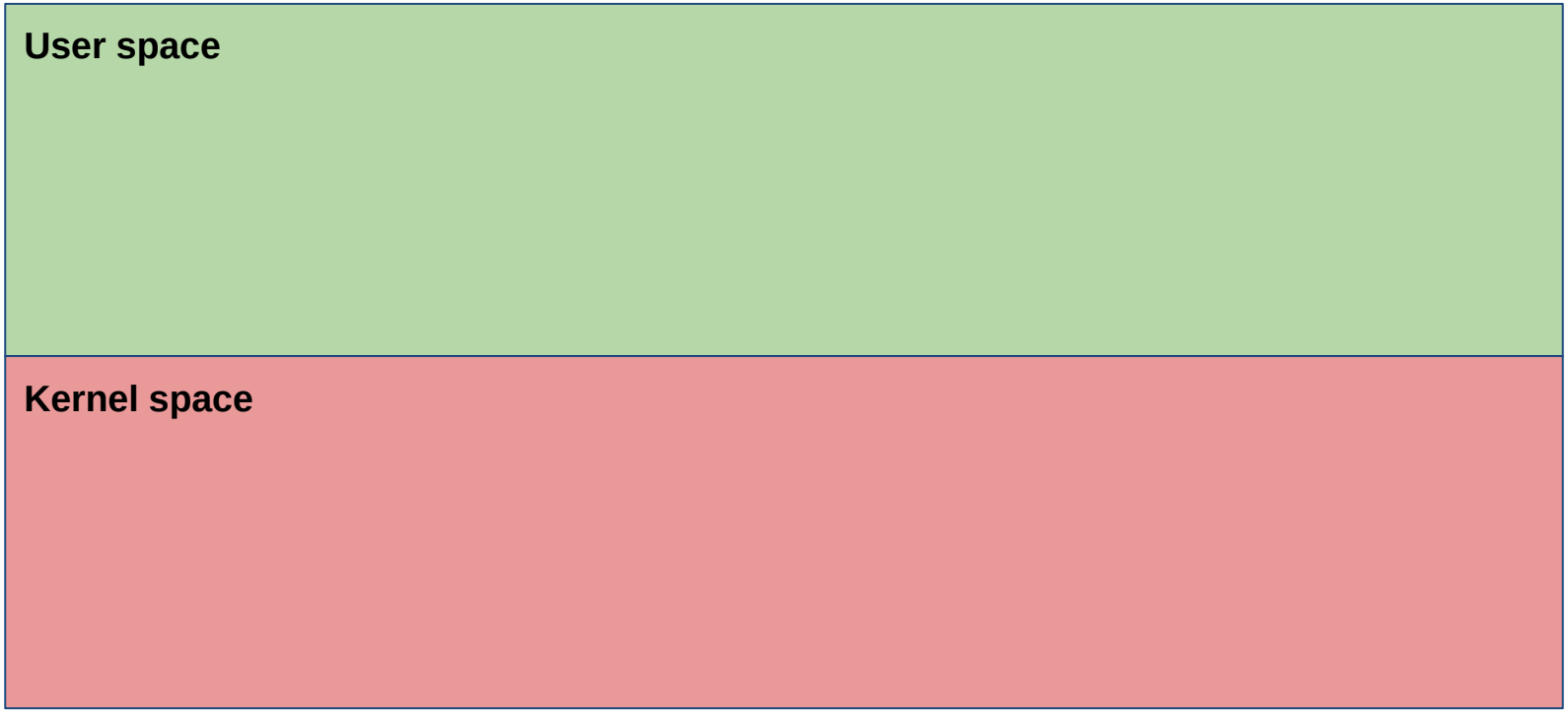
Operating System functions that are a part of the **TCB**:

- Process creation
- Process switching
- Memory management
- Part of file and I/O management

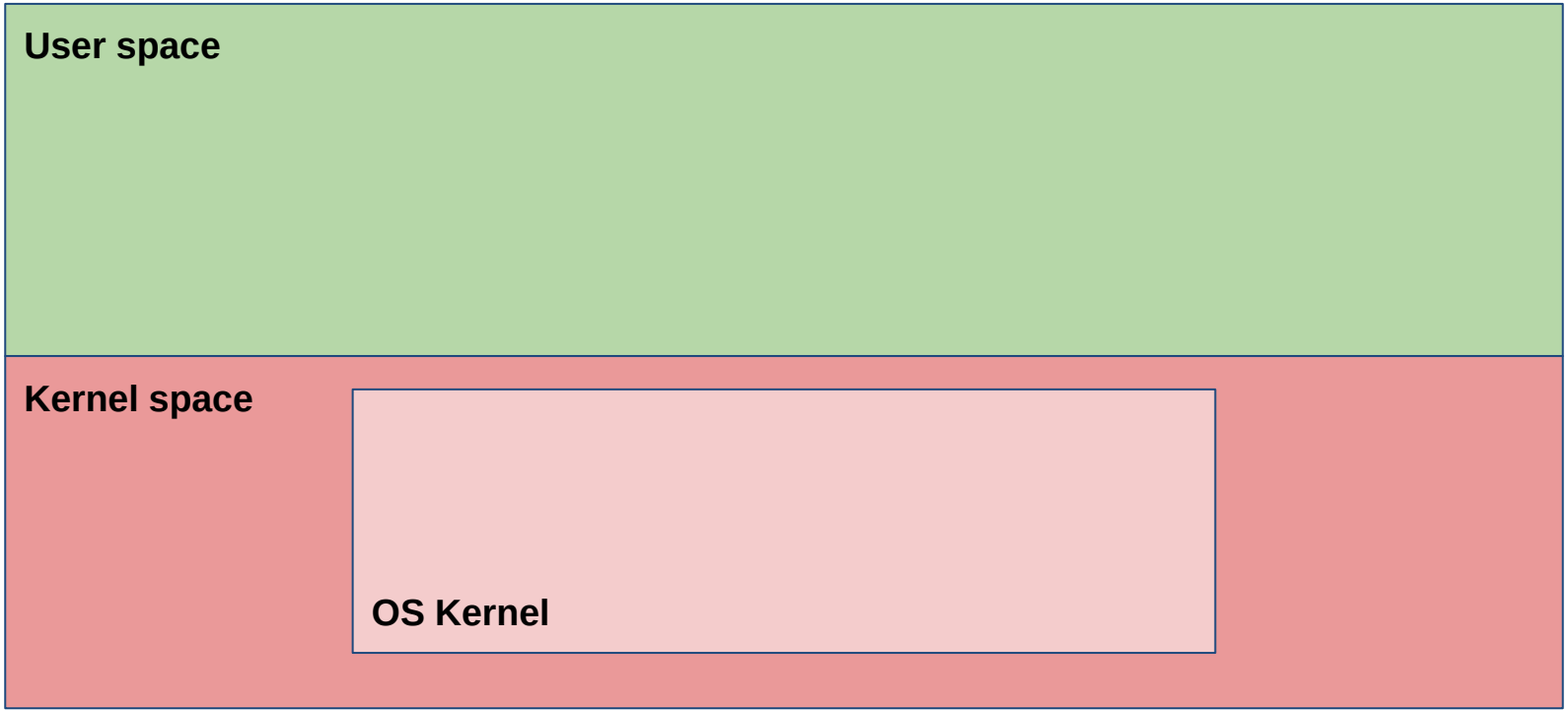
A Reference Monitor



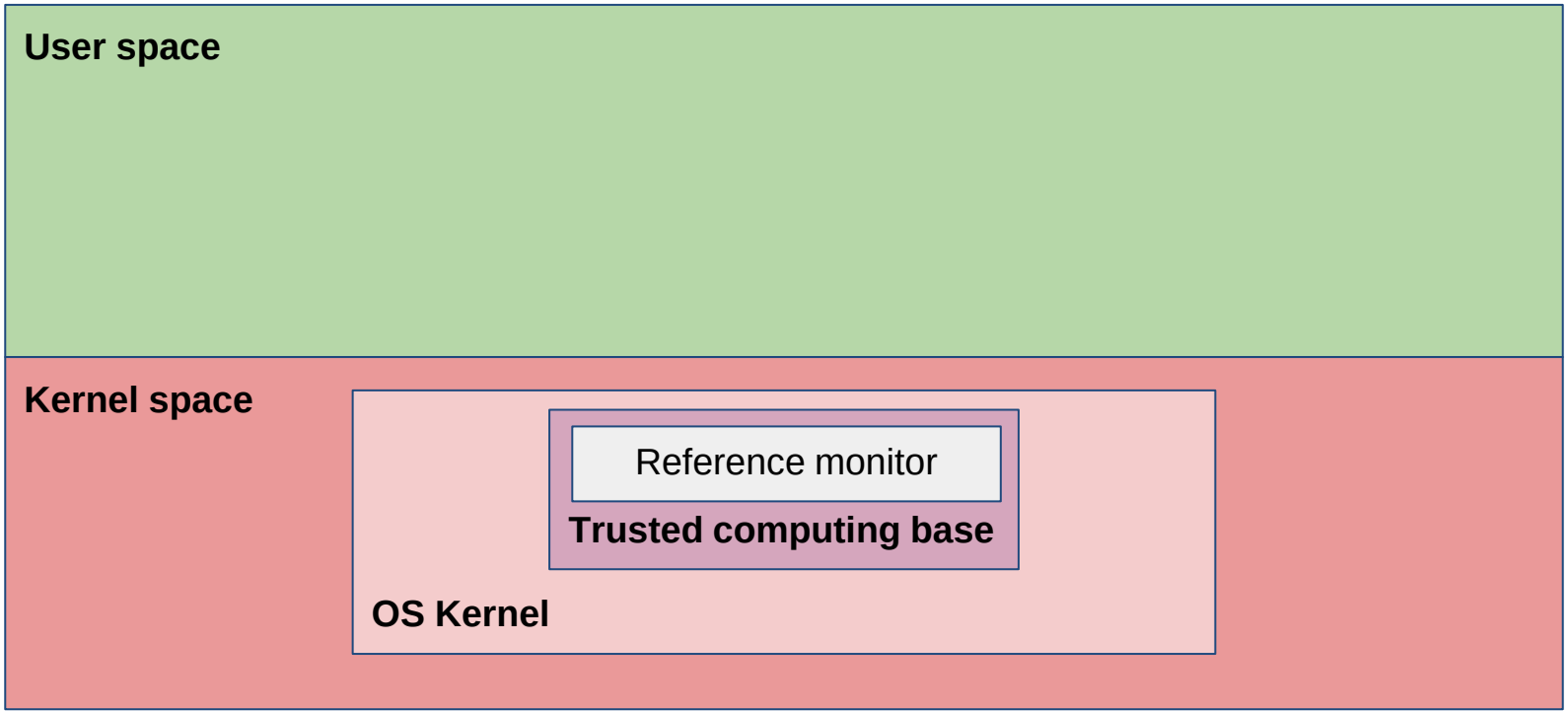
A Reference Monitor



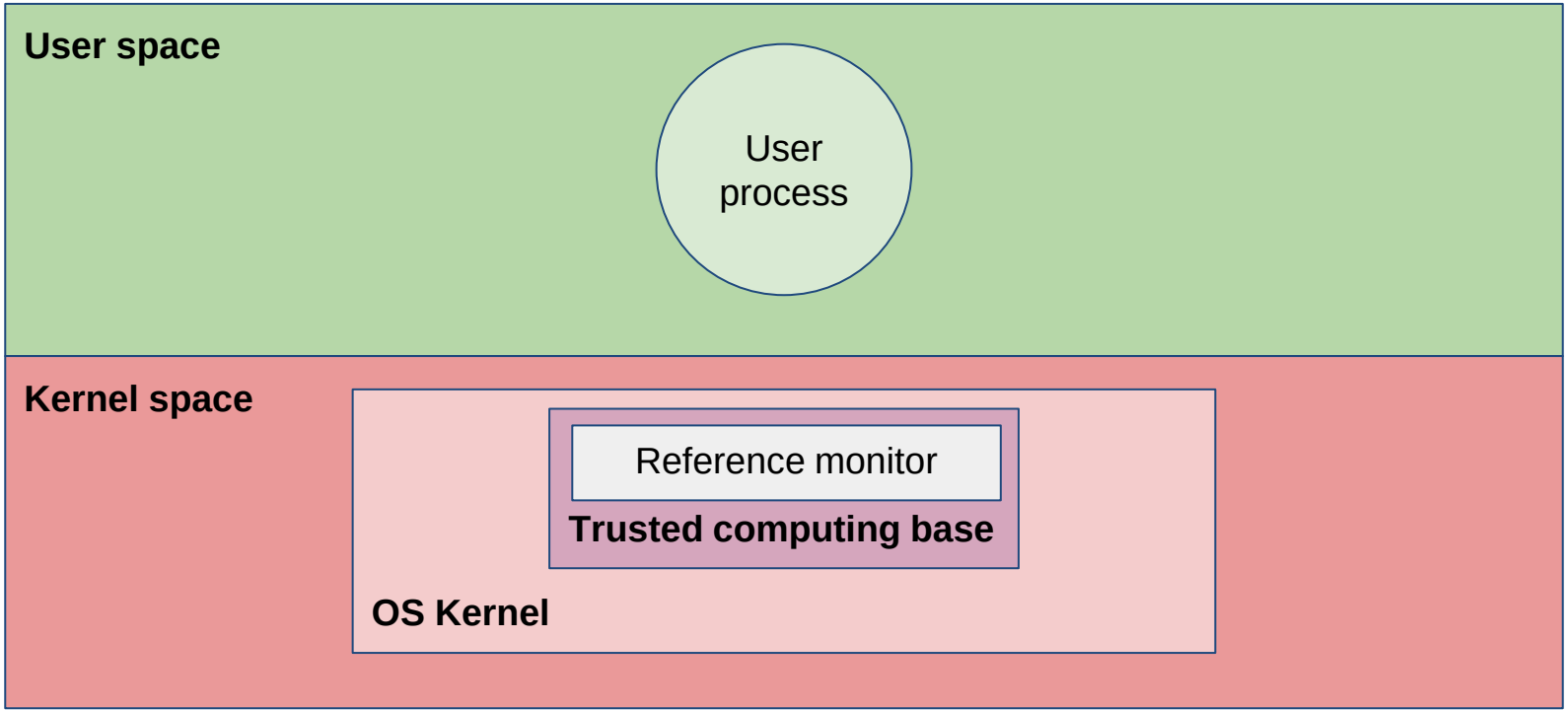
A Reference Monitor



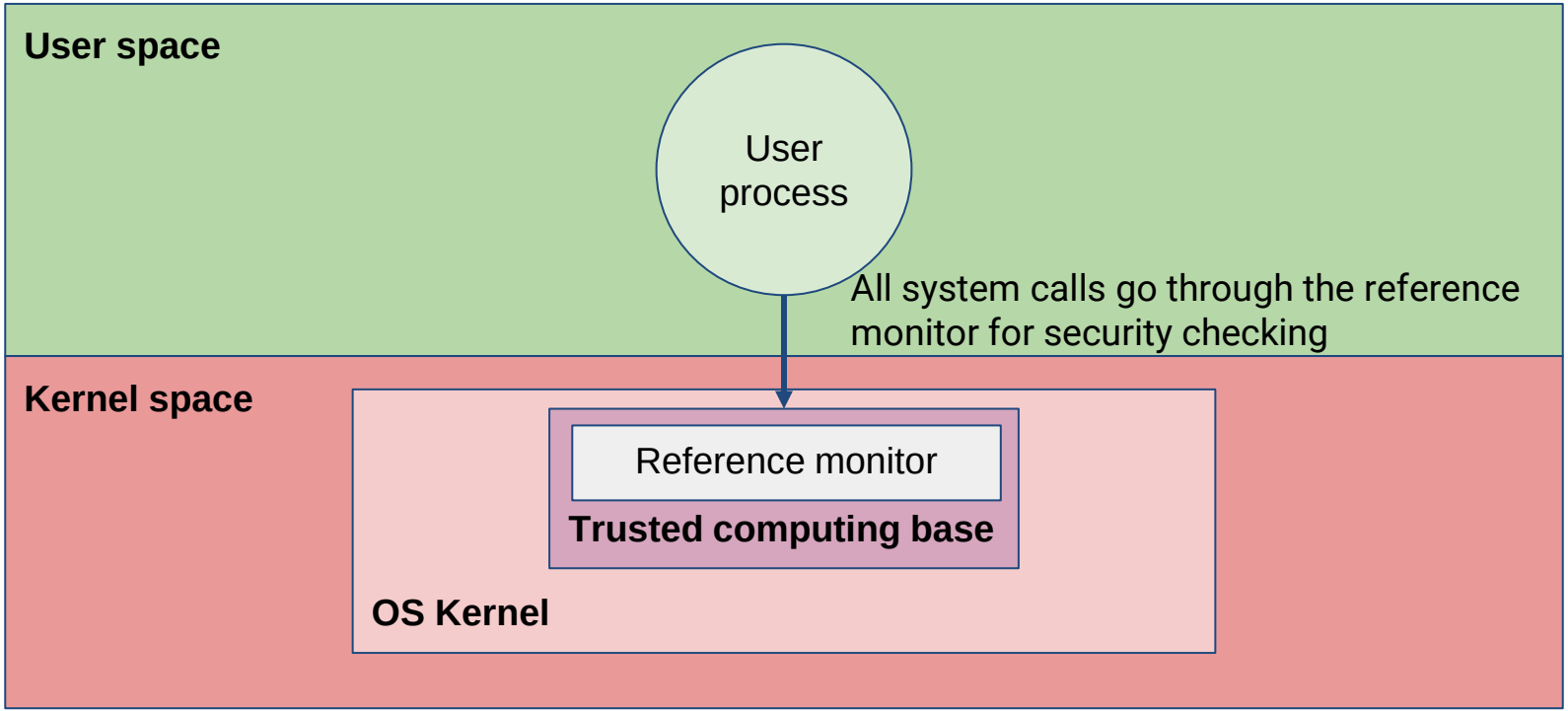
A Reference Monitor



A Reference Monitor



A Reference Monitor



Protection Domains

A **domain** is a set of pairs (object, rights), each pair specifies an object and some subset of the operations that can be performed on it. A right in this context means permission to perform one of the operations.

Protection Domains

Format: object[rights]

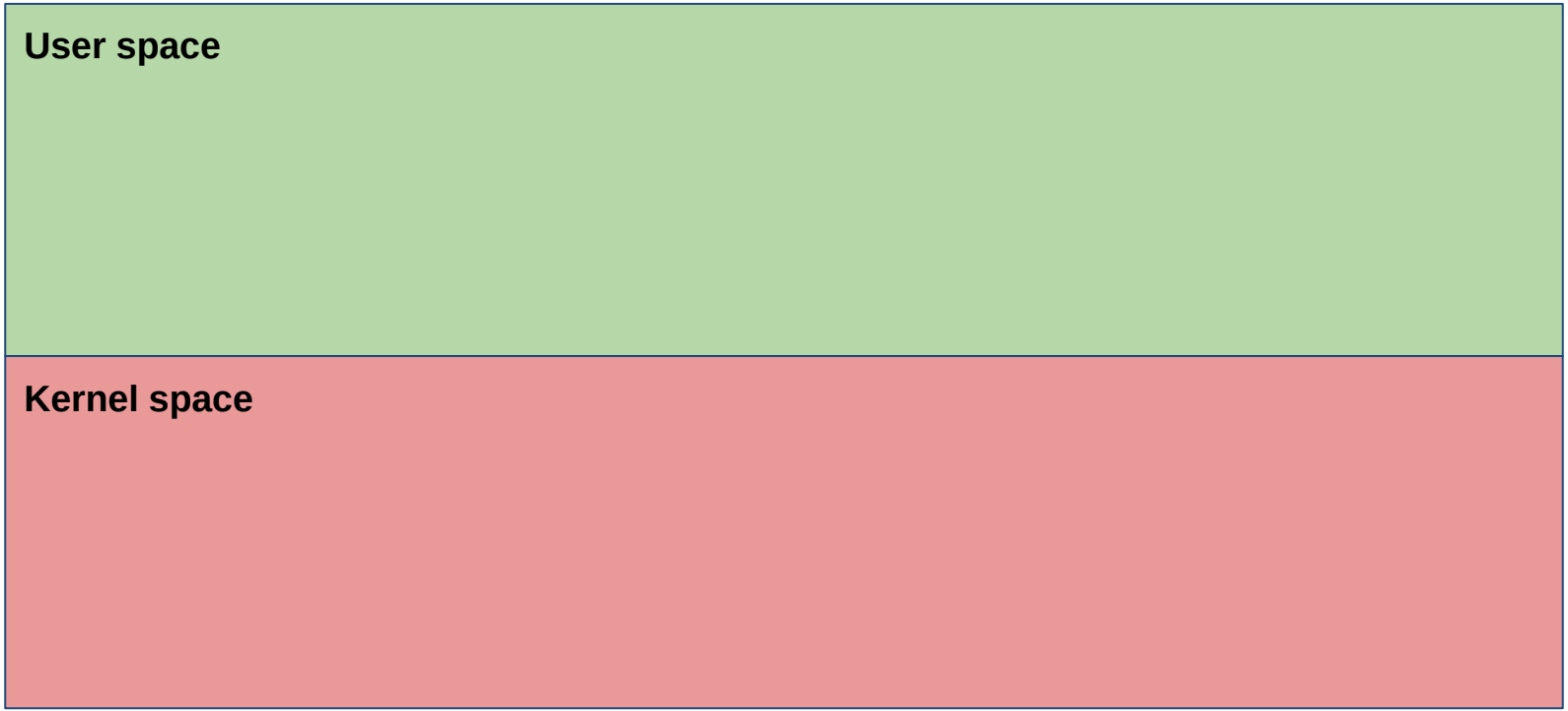
File1[R]

Access Control Lists (ACL)

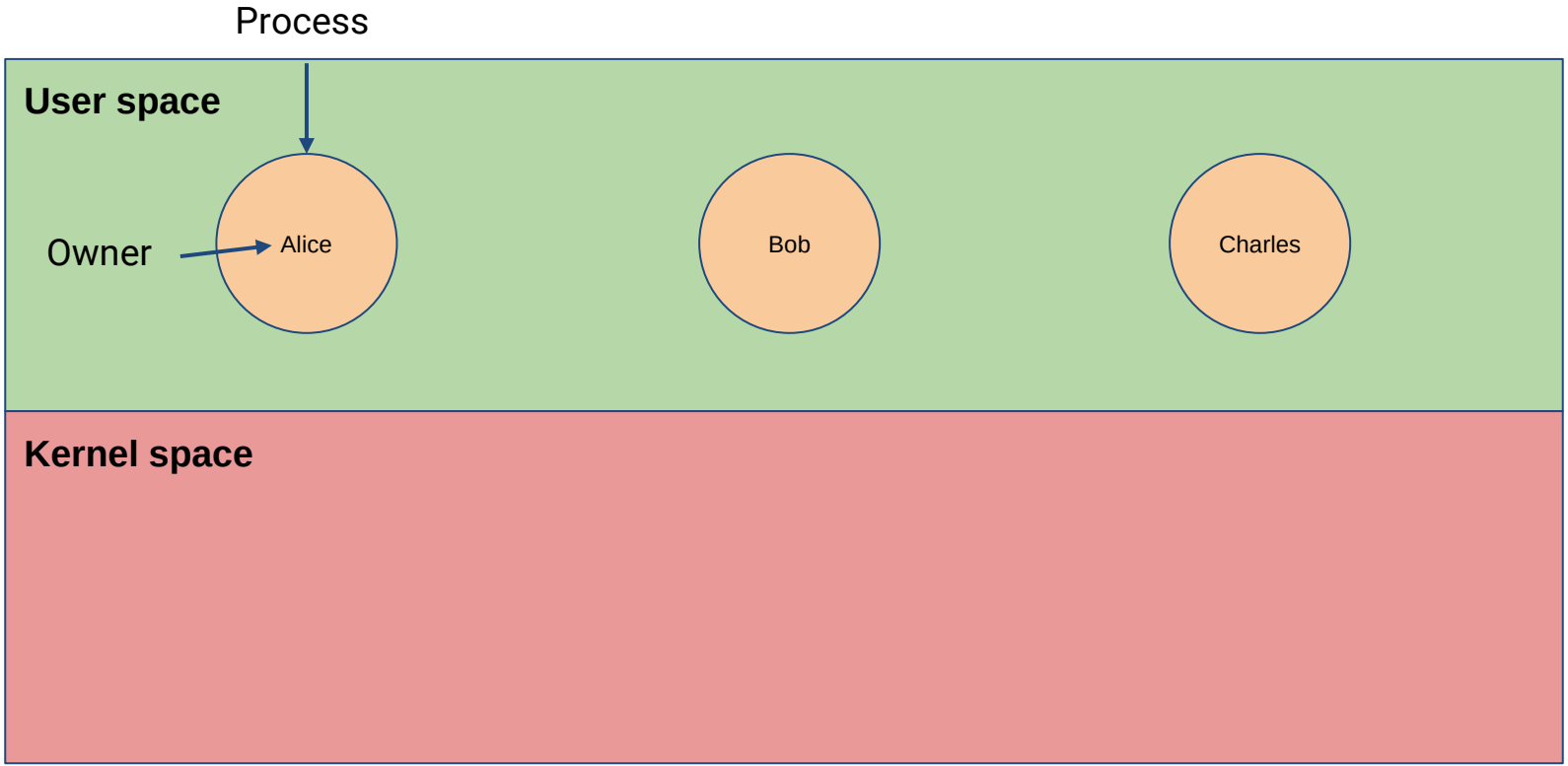
Two techniques in managing file access.

- First Technique: associates with each object an ordered list containing all the domains that may access the object.
- Second Technique: each process contains a capability list that indicate their respective access rights.

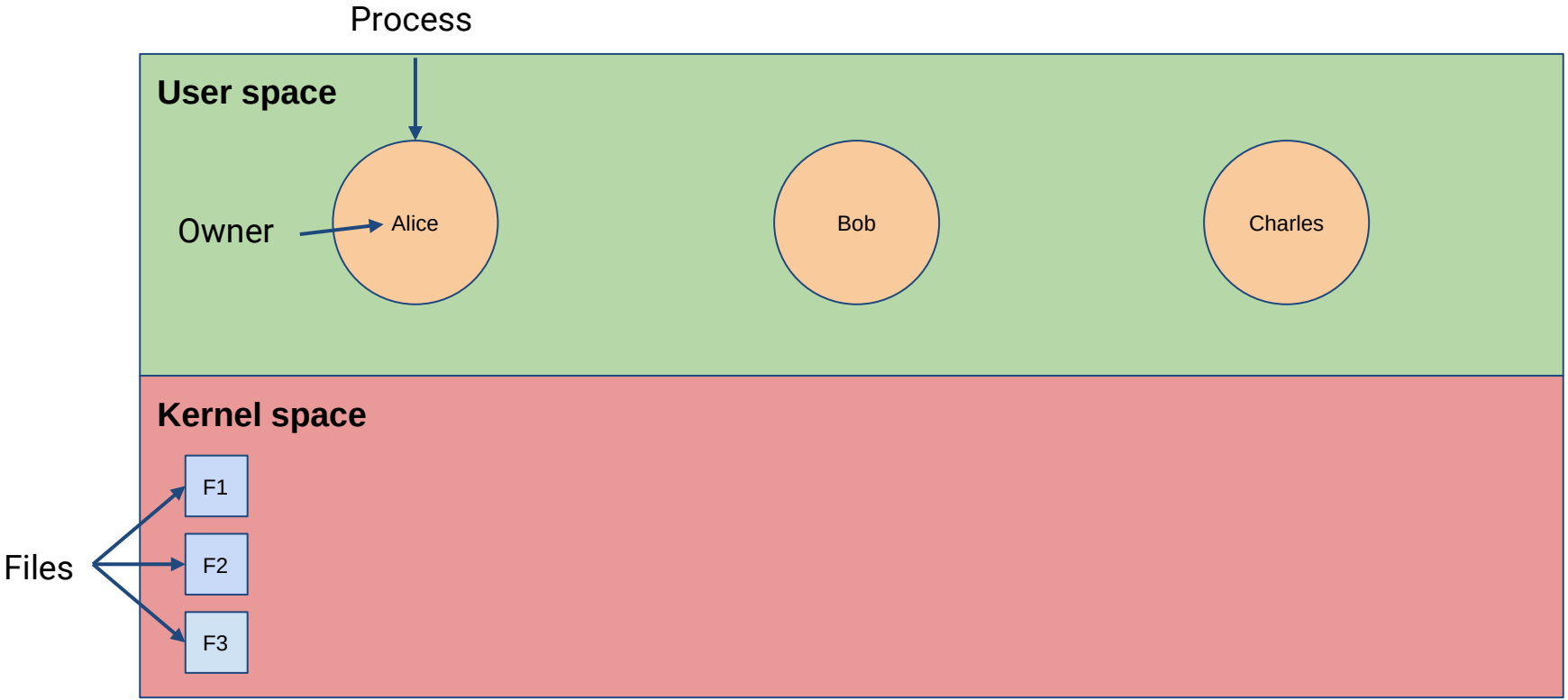
Access Control Lists



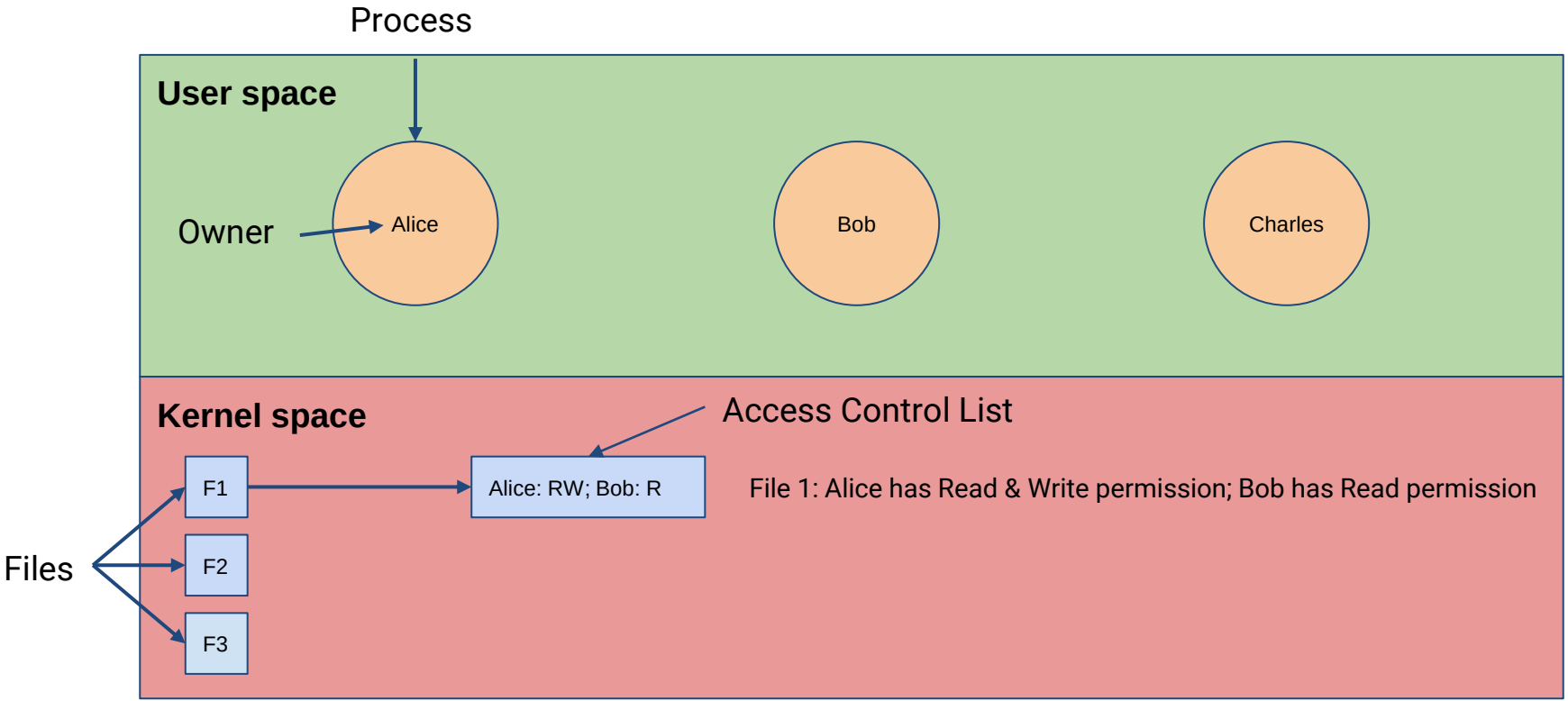
Access Control Lists



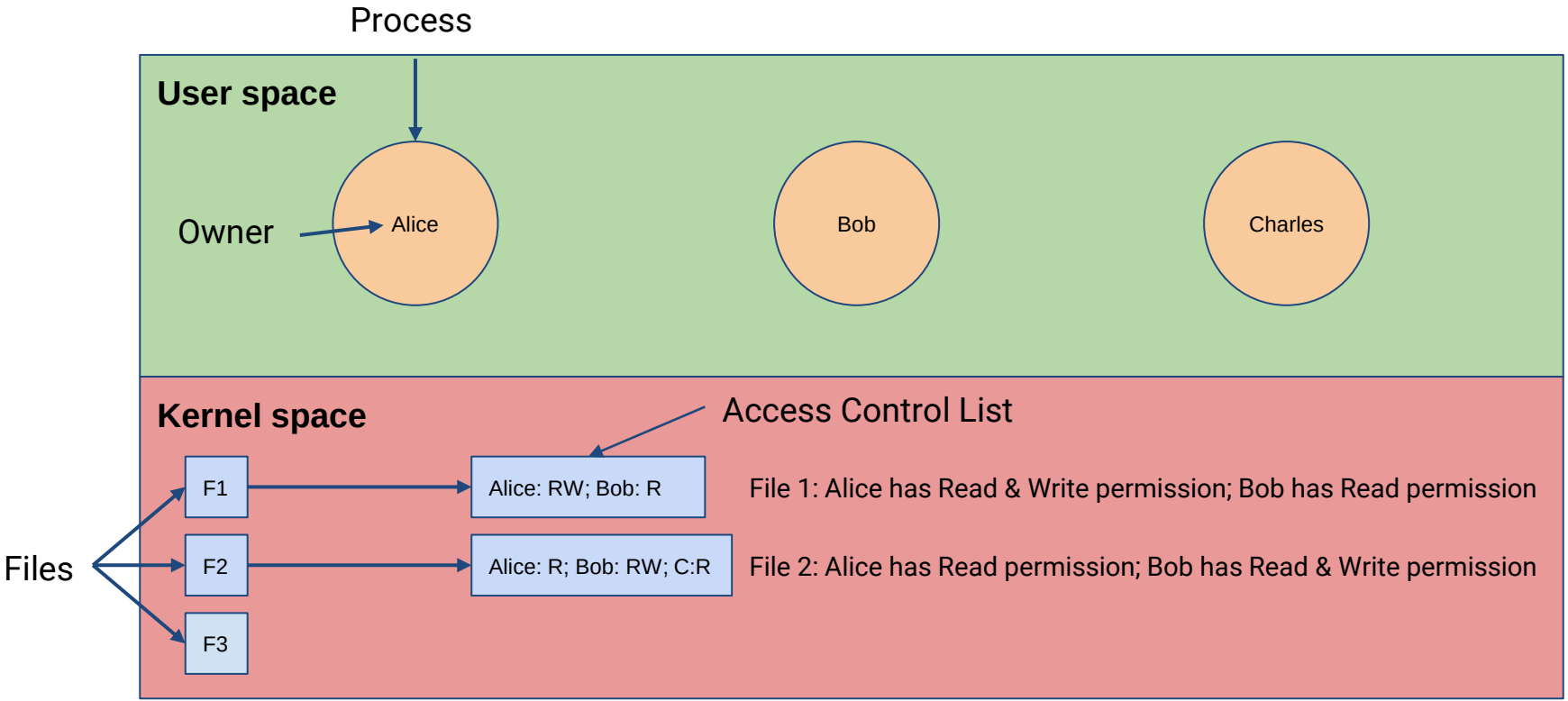
Access Control Lists



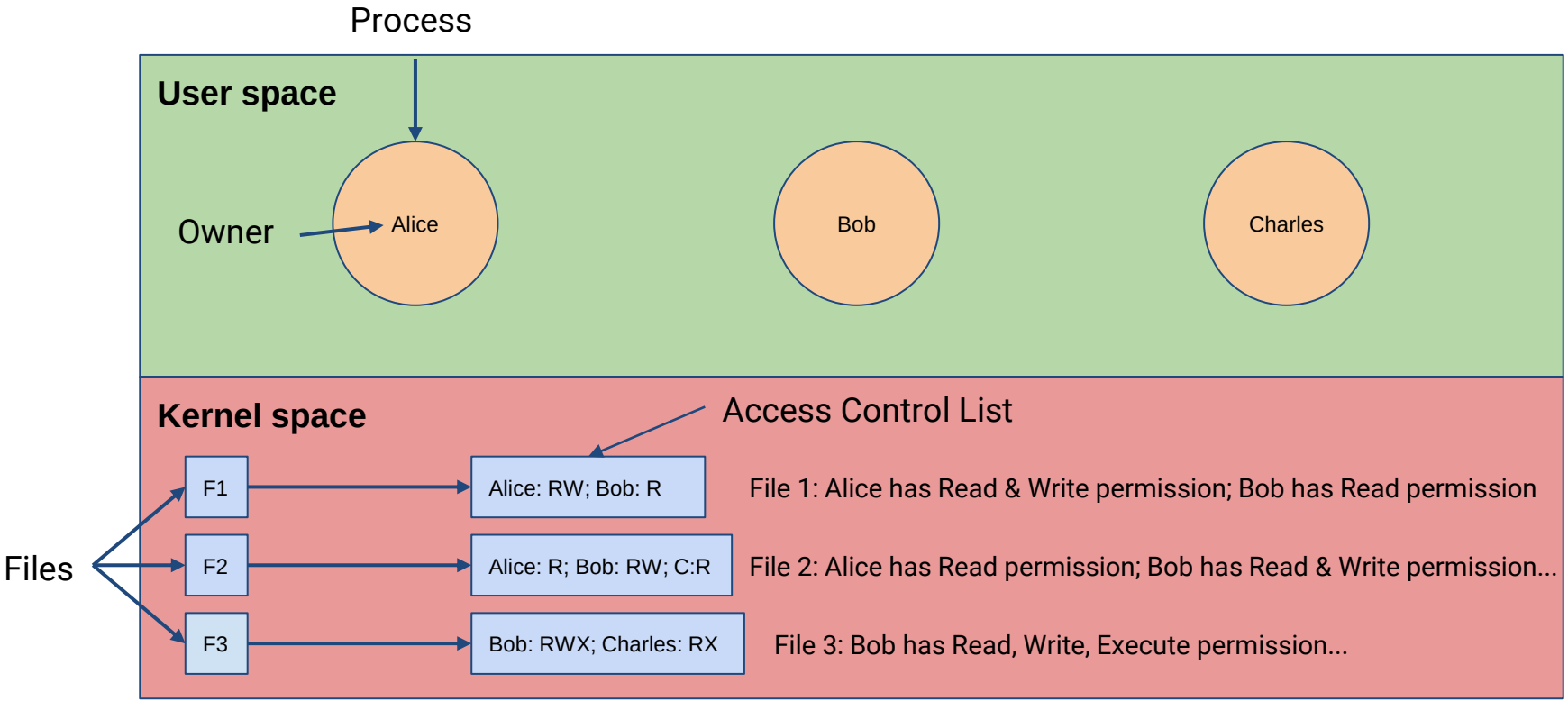
Access Control Lists



Access Control Lists



Access Control Lists

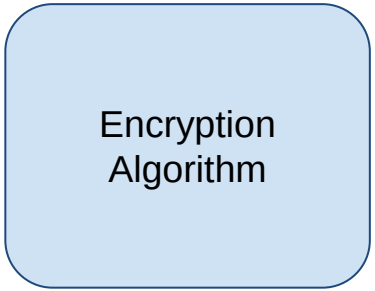


Cryptography

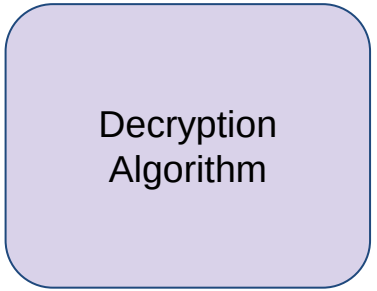
Concerned with transmitting original information into ciphertext to safeguard against the public who are not welcomed to access original contents.

- **Encryption** & **Decryption** algorithms take two parameters

```
encryption(plaintxt,encryptionKey)
```




```
decryption(ciphertxt,decryptionKey)
```

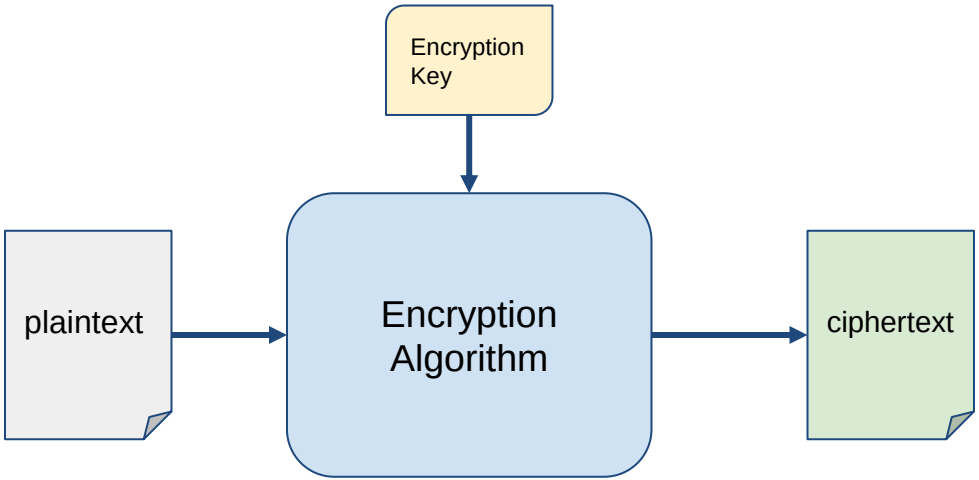


Cryptography

- Suppose Professor Young wants to send a message to his students.
  - Professor Young uses encryption algorithm by passing in plaintext and encryption key
  - Students use decryption algorithm by passing in cipher text and decryption key

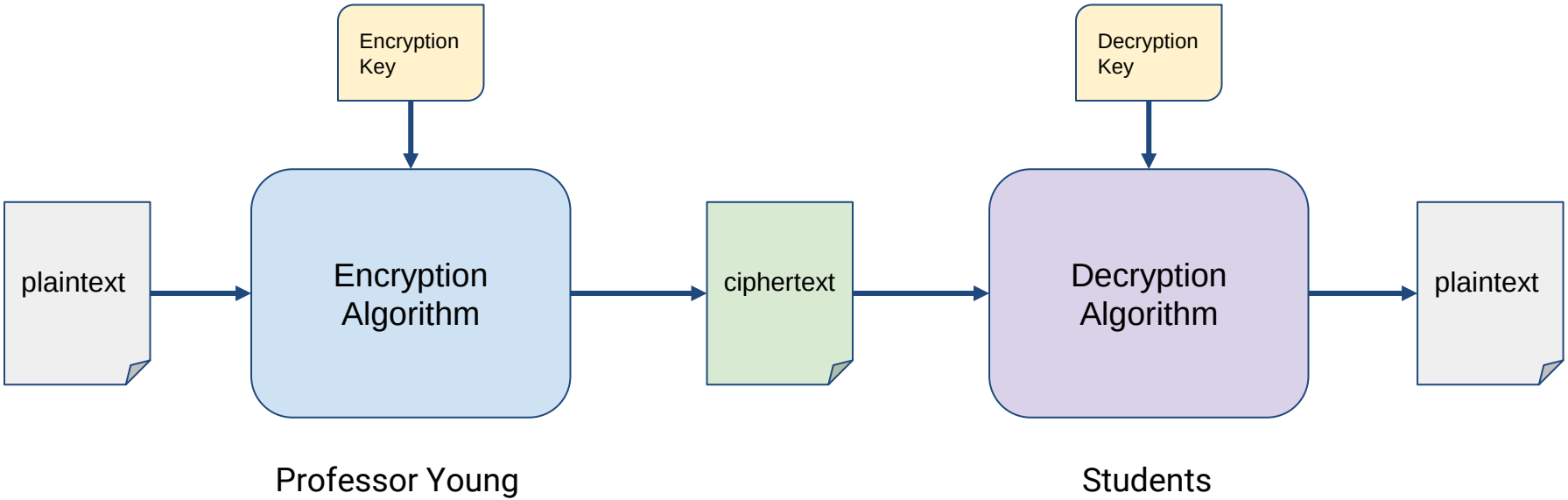
Cryptography

- Suppose Professor Young wants to send a message to his students.
  - Professor Young uses encryption algorithm by passing in plaintext and encryption key to generate cipher text.



Cryptography

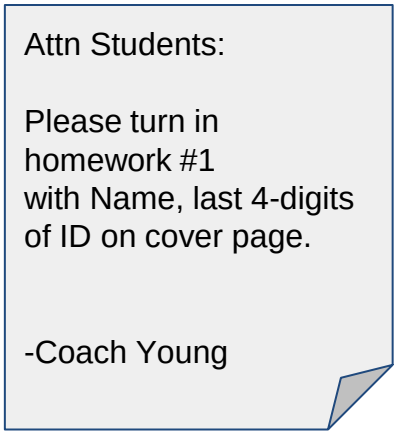
- Suppose Professor Young wants to send a message to his students.
  - Professor Young uses encryption algorithm by passing in plaintext and encryption key
  - Students use decryption algorithm by passing in cipher text and decryption key



Cryptography

- Suppose Professor Young wants to send a message to his students.
  - Professor Young uses encryption algorithm by passing in plaintext and encryption key
  - Students use decryption algorithm by passing in cipher text and decryption key

- **End result:** students decrypt the ciphertext with their keys to read original message.



Digital Signatures

A way to represent a document as a unique identity can be achieved through a secure hash algorithm (SHA).
Suppose a hashing function **f** and it's input **x** exists, where **x** represents a document, then let:

$$f(x) = y$$

Where **y** represents a unique digital signature or a message digest of **x**.

Digital Signatures

$$f(x) = y$$

If we were given **y**, it would be infeasible to get the contents of **x**.
Thus, we refer to these secure hash algorithms as one-way functions.

Digital Signatures

f(x) = y

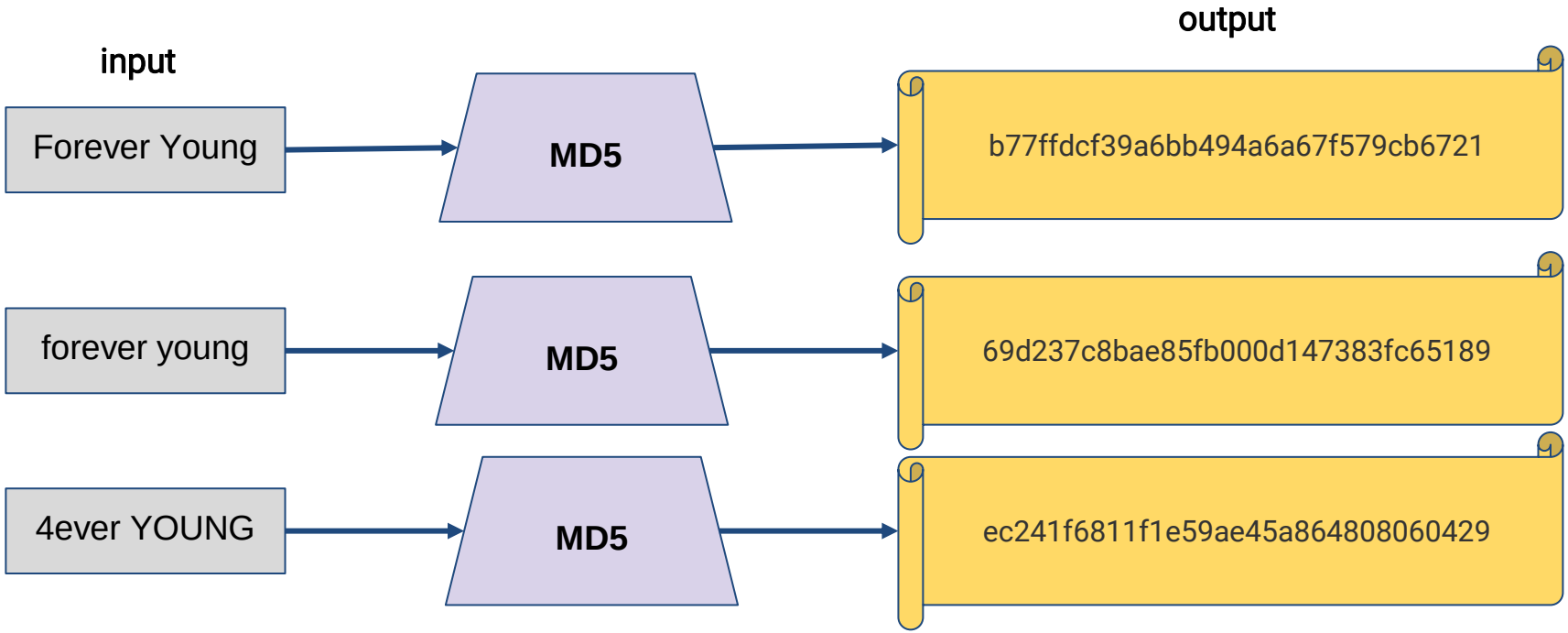
x **f(x)** **y**

Digital Signatures

Let's take a look at two widely mentioned secure hash algorithms called: **MD5** and **SHA-1**.

Notice that slight changes to the input completely alters the hashed result

MD5 example:

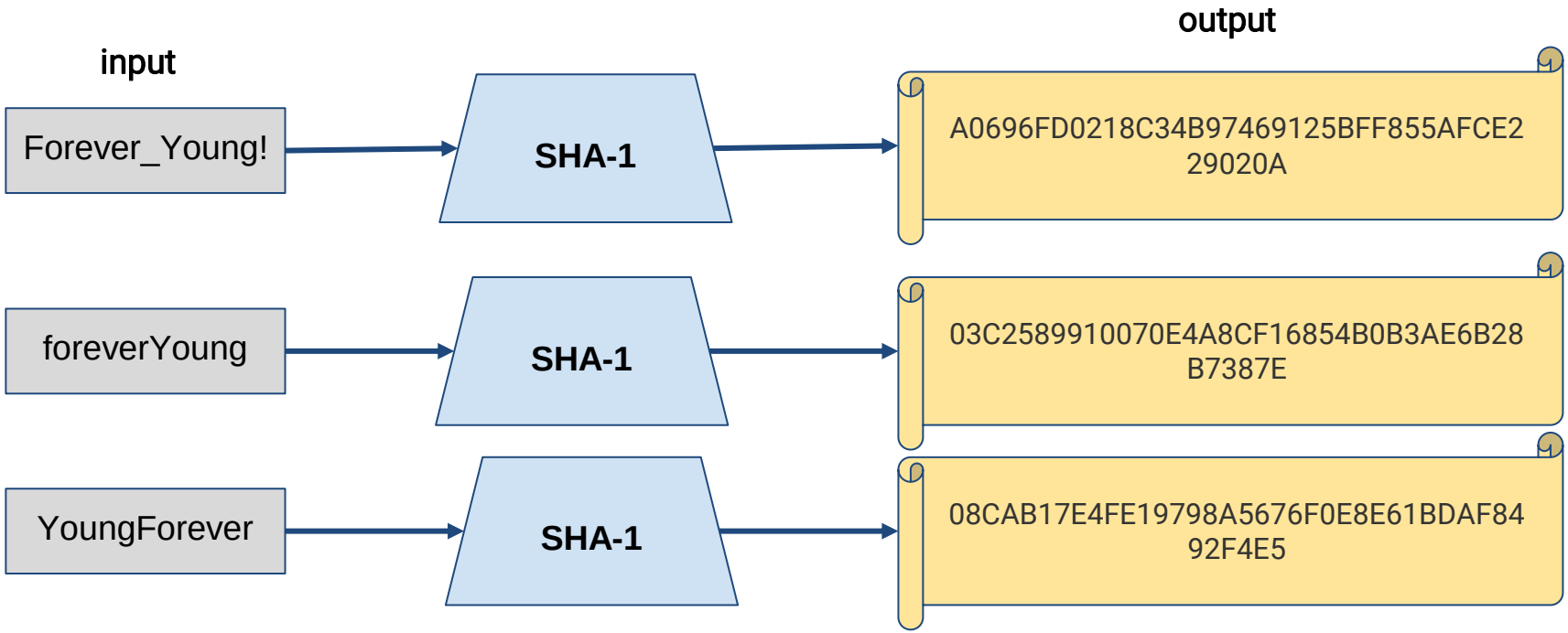


Digital Signatures

Let's take a look at two widely mentioned secure hash algorithms called: **MD5** and **SHA-1**.

Notice that slight changes to the input completely alters the hashed result

SHA-1 example:



Digital Signatures

An **MD5** algorithm produces a 16-byte result, whereas **SHA-1** produces a 20 byte result.

Practical infeasibility: By brute force method of guessing…

MD5 requires 2^{128} number of guesses
&
SHA-1 requires 2^{160} number of guesses to find a match

Perspective Scale:

2^{25} = 33,554,432
2^{50} = 1,125,899,906,842,624
2^{100} = 1,267,650,600,228,229,401,496,703,205,376
2^{160} = **<A VERY LARGE NUMBER>**

Digital Signatures

Despite the Practical Infeasibility of guessing a hash value,
MD5 and **SHA-1** are no longer considered secure,
from the courtesy of supercomputers.

Today's industry standards suggest **SHA-256** and **SHA-512**

Authentication

Every secured computer system must require all users to be authenticated at login time.

Most methods of authenticating rely on 3 principles:

- Something the user knows.
- Something the user has.
- Something the user is.

Types of Authentication

- **One-time password:** Each login uses a new password, utilizes one-way hashing to generate a list of passwords.
- **Challenge-Response Authentication:** Users required to select a few security questions and provide answers to the respective question.
    - What's the name of your first pet?
    - What's your mother's maiden name?
    - How fast is the universe expanding?
- **Authentication Using Physical Object:** Users required to physical object that would prompt a password (ex: ATM card, smart card, phone, etc,.)

Types of Authentication

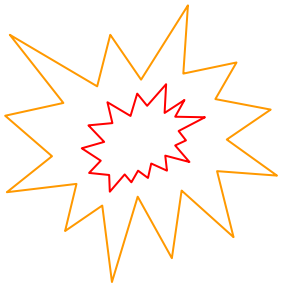
- **Biometrics:** measures physical characteristics of user for authentication (ex: fingerprint, voice, etc,.)

Insider Attacks

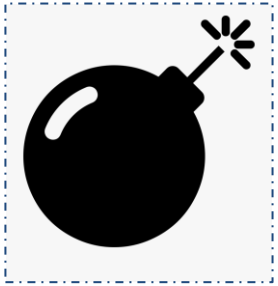
“Inside jobs” are executed by programmers and other employees of a company running the computer to be protected or making critical software. Insiders have special knowledge that the outsiders do not have.

Types:

- **Logic Bombs**
- **Back Doors**
- **Login Spoofing**



Insider Attacks: Logic Bombs



Scenario: A programmer who is insecure of their job position might be inclined at blackmailing the company.

Plan of attack: A piece of source code written, secretly inserted in the production system. A daily password, which only the employee knows about, must be entered daily to disarm the “bomb”.

Consequence: If the employee is fired, then the bomb detonates. Detonation might involve erasing files at random, hard-to-detect changes to key programs, or encrypting essential files.

https://www.netclipart.com/isee/iowbJo_bomb-png-picture-bomb-icon-png/

Insider Attacks: Back Doors

Scenario: A programmer wants to abuse his/her power to steal valuable information.

Plan of attack: Code is inserted into the software by the programmer to bypass login authentication.

Consequence: Once software deploys to the public, the programmer has access to many accounts since he/she can bypass login.

Insider Attacks: Login Spoofing

Scenario: Perpetrator is attempting to collect other people's passwords.

Plan of attack: Find a popular public place with multiple computer access points (ex: library, CS lab/classroom, LAN cafe, etc,.) and install a software that mimics a login screen.

Consequence: Unsuspecting users enter their password into the phony login screen and is collected. Phony login screen terminates as if user incorrectly entered their password, thereby not raising suspicion.

Malware Types

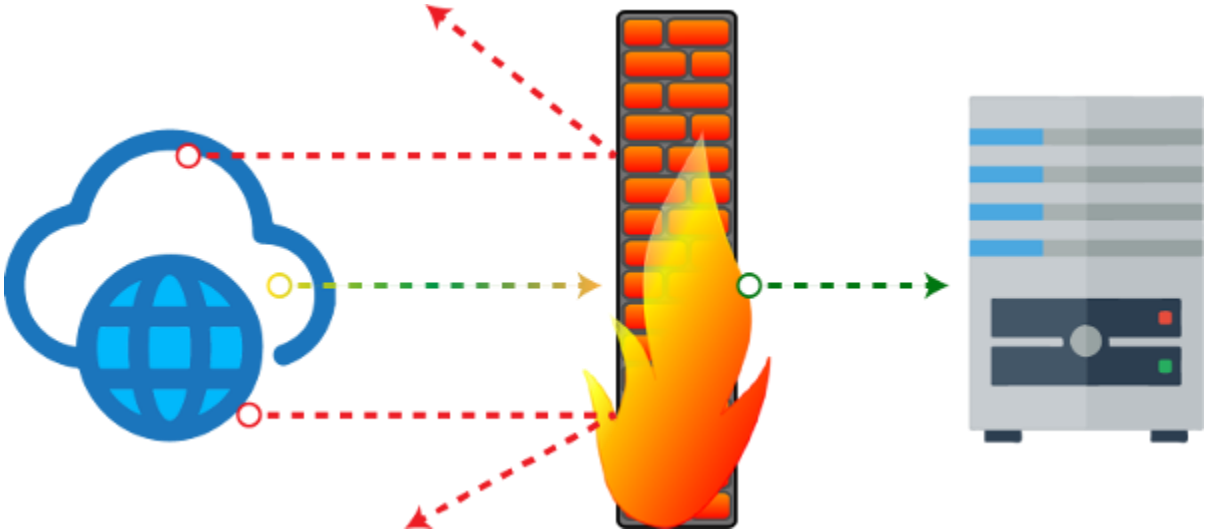
Malware: malicious software designed to harm a computer system.

- **Trojan Horses** - malware disguised as a useful program.
- **Virus** - a program that can reproduce itself by attaching code to another program. Depends on host program to spread itself.
- **Worms** - a malicious, self replicating program that does not require a host program to run.
- **Spyware** - secretly loaded onto a PC without owner's knowledge and runs in the background doing things behind the owner's back.
- **Rootkits** - program or set of programs and files that attempts to conceal its existence; can contain a subset of malwares described above.

http://pixelartmaker.com/art/4b0f046ad6dfc32

Defenses: Firewall

- **Firewalls** - A system emplaced to maintain security; a mechanism to keep “good” bits in and “bad” bits out.



https://www.gajshield.com/index.php/product/productfeatures

Defenses: Antivirus Techniques

- **Virus Scanners** - programs that attempt to detect and isolate virus signatures. Once an antivirus program is installed on a customer's machine, the first thing it does is scan every executable file on the disk looking for any of the viruses in the database of known viruses.

- **Integrity Checking** - using a checksum to validate files.

- **Behavioral Checking** - antivirus program living in memory, monitors all system calls.

Review

Chapter 3. Deadlock

- Deadlock Strategies
  - Just ignore
  - Detection and Recover
    - Deadlock Detection Algorithm
    - Recover

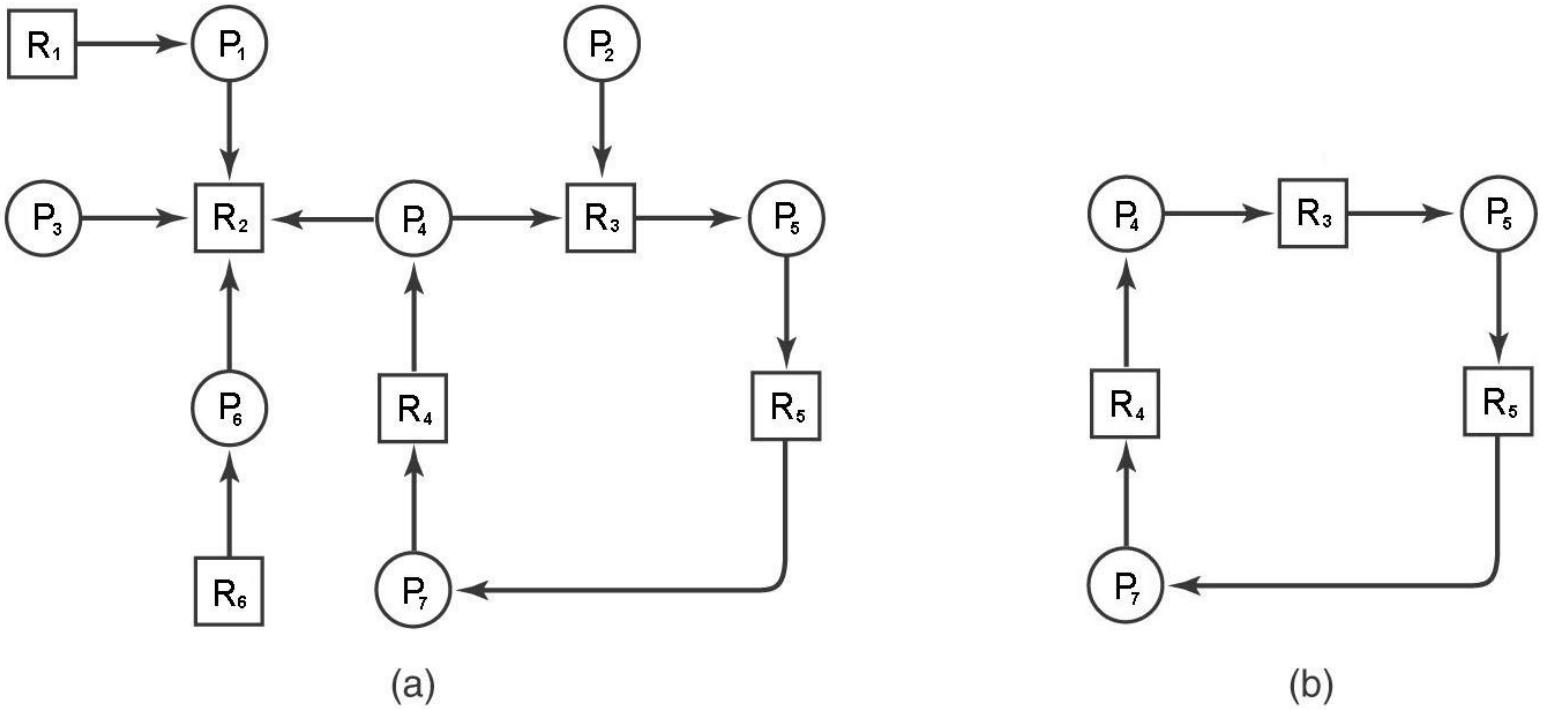
Deadlock Condition

A deadlock situation can arise if and only if the following four conditions hold simultaneously in a system. (Coffman et al.)

1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait

Deadlock Detection and Recovery
(Detection with one resource of each type)

A cycle – means *deadlock*



Deadlock Detection and Recovery
(Detection with Multiple resources of each type)

We need matrices for deadlock detection algorithm.

- Existing resource matrix – At present, the number of resources per each type.
- Available resource matrix – At present, the number of resources per each type available, with some of the resources are assigned to processes (not available).
- Current allocation matrix – At present, which resources are currently held by processes
- Request matrix – At present, how many resources are needed for processes to finish their job.

Deadlock Detection and Recovery
(Detection with Multiple resource of each type)

E Resources in existence $(E_1, E_2, E_3, \ldots, E_m)$

A Resources available $(A_1, A_2, A_3, \ldots, A_m)$

C Current allocation matrix

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix}$$

Row n is current allocation to process n

R Request matrix

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$$

Row 2 is what process 2 needs

We have the following equation from the four matrix structures.

$$\sum_{i=1}^{n} C_{i,j} + A_j = E_j$$

Deadlock Detection and Recovery
(Detection with Multiple resource of each type)

- The deadlock detection algorithm is based on comparing vectors.
- Let's define the relation A ≤ B between two vector A and B means that each element of A is less than or equal to the corresponding element of B.

Ex)

A = (1, 2, 0 2), B = (1, 3, 0 2): Is A ≤ B true? Yes

A = (1, 2, 0, 2), B = (2, 0, 0, 0): Is A ≤ B true? No

Deadlock Detection and Recovery
(Detection with Multiple resource of each type)

A Deadlock detection algorithm

1. All processes start with unmarked.
2. Look for an unmarked process P_i, for which the i^{th} row of R is less than or equal to A.
3. If such a process is found, add the i^{th} row of C to A, mark the process and go back to step 1.
4. If no such process exists, the algorithm terminates.

Preview

- Deadlock Avoidance
  - Safe and Unsafe state
  - Banker's Algorithm
- Deadlock Prevention
  - Attack Mutual Exclusion
  - Attack Hold and Wait
  - Attack No-Preemption
  - Attack Circular Wait

Deadlock Avoidance

- The system must be able to decide whether granting a resource is safe or not and only make allocation when it is safe.
- Is there any algorithm that can always avoid deadlock by making the right choice all the time?
  - Yes, when certain information is available in advance.

Deadlock Avoidance
(Safe and Unsafe State)

A state is said to be a safe state

- If it is not deadlocked and
- there is some scheduling order in which every process can run to completion even if all of them suddenly request their maximum number of resources immediately.

Deadlock Avoidance (Safe and Unsafe State)

The State in (a) is Safe

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

(b)

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5

(c)

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0

(d)

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7

(e)

Deadlock Avoidance (Safe and Unsafe State)

The State in (a) is Safe & The State in (b) is Unsafe

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4

(d)

Banker's Algorithm for a Single Resource (Dijkstra)

- Modeled on the way a small-town banker might need to deal with a group of customers.
- What banker's algorithm does is check to see whether granting the request leads to an unsafe state or not.
- If granting the request leads to an unsafe state, the request is denied.

Banker's Algorithm for a Single Resource (Dijkstra)

Ex)

- There are four customers A, B, C, D, each of whom has been granted a certain amount of credit units.

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

(a)

	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

(b)

	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

(c)

Three resource allocation states: (a) Safe (b) Safe (c) Unsafe

- The banker knows that not all customers will need their maximum credit immediately, so banker has reserved only 10 units rather than 22.

Banker's Algorithm for Multiple Resource

Using three matrices for checking a safe state

- Available resource matrix **(A)** – At present, how many number of resources per each type are available (Note that some resources are assigned to processes and they are not available).
- Request matrix (R) – At present, how many resources (at most) are needed for processes to finish their jobs.
- Current allocation matrix (C) – At present, how many resources are currently held by processes.

Banker's Algorithm for Multiple Resource

The algorithm for checking if a state is safe or not

1. Look for a row of vector R, whose unmet resource needs are all smaller than or equal to A. If no such row exists, the system might have a deadlock later as some processes might never run to completion.
2. Assume the process (of the row) requests all the resources it needs and finishes. Mark that process as terminated and add all its resources to the A vector.
3. Repeat step 1 and 2 until either all processes are terminated (safe state) or until a deadlock occurs (unsafe).

Banker's Algorithm for Multiple Resource

E – Existing resources, P – Possessed resources, Available resources

Process	Tape drives	Plotters	Scanners	CD ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Scanners	CD ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

E = (6342)
P = (5322)
A = (1020)

Banker's Algorithm for Multiple Resource

- E = (6, 3, 4, 2)
- A = (1, 0, 2, 0)

$$C = \begin{bmatrix} 3011 \\ 0100 \\ 1110 \\ 1101 \\ 0000 \end{bmatrix}, R = \begin{bmatrix} 1100 \\ 0112 \\ 3100 \\ 0010 \\ 2110 \end{bmatrix}$$

Deadlock Prevention

- Actually, deadlock avoidance is essentially impossible, since it requires information about future requests which is not known.
- Deadlock prevention is a method to attack one of four deadlock condition.
  1. Mutual exclusion
  2. Hold and wait
  3. No preemption
  4. Circular wait

Deadlock Prevention
(Attacking Mutual Exclusion)

Attacking Mutual Exclusion

- The mutual-exclusion condition must hold for non-preemptive resources such as printer, CD writer.
- But preemptive resources do not require mutual exclusion such as read only file.

Deadlock Prevention

(Attacking the hold and wait)

Attacking the hold and wait

Approach 1)

- Each process requests all resources before starting execution.
- If everything is available, all resources (requested by the process) are located and finish its job.
- If some resources are not available, no resources are allocated to the process.

Deadlock Prevention

(Attacking the hold and wait)

Problem with Approach 1)

- Many processes do not know how many resources they will need before start execution.
- If possible, Banker's algorithm can be applied.
- Resources cannot be used optimally

Ex)

- P_1 running for its completion, holding resources R_1, R_2, and R_3.
- P_1 currently using R_1, but R_2 and R_3 are idle.
- P_2 needs only R_2 to finish its job, but R_2 is held by P_1.

Deadlock Prevention

(Attacking the hold and wait)

Attacking the hold and wait

Approach 2)

- Allows a process to request resources only when the process has none.
- To get a new resource, first, release all the resources currently holds and request all at time same time.

Disadvantage with Approach 2

- Starvation is possible – a process that needs several popular resources may have to wait indefinitely since at least one of the resources that it needs is always allocated to some other process.

Deadlock Prevention
(Attacking No Preemption)

Attacking No Preemption

Approach 1)

- If a process holding some resources requests another resources that cannot be immediately allocated it, then all resources currently being hold are preempted.
- Preempted resources are intentionally released and enter the available resources list.
- The process will be restarted only when it can regain its old resources as well as the new ones that it is requesting

Deadlock Prevention
(Attacking No Preemption)

Attacking No Preemption

Approach 2)

- If a process requests some resources, first checks whether they are available or not
- If they are, allocate them to the process.
- If they are not available, check whether they are allocated to some other process that is waiting for additional resources.
- If so, preempt the desired resources from the waiting process and allocate them to the requesting process.
- If the resources are not either available or held by waiting process, the requesting process must wait.

Deadlock Prevention
(Attacking the Circular Wait Condition)

Approach 1)

- A process can hold only one resource.
- If a process needs another resource, the process needs to release the first one.
- But sometimes this approach is not acceptable

Approach 2)

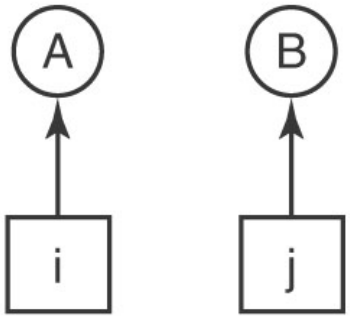
- Global number is provided to each resource
- A process can request resources whenever they want, but all requests must made in numerical order.

Deadlock Prevention

(Attacking the Circular Wait Condition)

1. Imagesetter
2. Scanner
3. Plotter
4. Tape drive
5. CD Rom drive

(a)



(b)

- Only deadlock can occur if the process A makes a request of resource j and the process B makes a request of resource i. but it can never happen since either i > j or i < j.

Multiple Processor Scheduling

- Scheduling is concerned with the allocation of scarce resources to activities so as to optimize one or more performance measures.

- Depending on the situation, resources and activities can take many different forms. Resources may be machines in a workshop, runways at an airport, crews at a construction site, CPU and I/O devices in a computing system, and so on.

- Activities may be operations in a production process, landings and take offs at airport, stages in a construction project, executions of computer programs, and so on.

- Performance measures can also take many forms. One performance measure may be the minimization of late jobs, while another performance measure may be the average completion time of the jobs.

Deterministic models & the alpha|beta|gamma notation

$$\alpha \mid \beta \mid \gamma$$

- α describes the machine environment.
- β describes the processing characteristics and constraints of the jobs.
- γ describes the performance measures to optimize.

The number of jobs is denoted by n

The number of machines is denoted by m

The subscript j refers to a job

The subscript i refers to a machine

P_j denotes the processing time at job j.

1.
   - α - machine environment
   - **1** – single machine
   - $\mathbf{P_m}$ – identical machines in parallel

     Ex. P_2 – 2 identical machines in parallel
   - **P** – arbitrary no. of identical machines
   - $\mathbf{Q_m}$ – uniform machines in parallel
     - Each machine has its own speed. The speed of machine i is denoted by $V_{i.}$
     - Job j will take P_j/V_i units of real time to finish
   - **Q** – arbitrary no. of uniform machines

- **R_m** – unrelated machines in parallel.

  Each machine can process different jobs at different speeds

  Machine i can process job j at speed V_{ij}

  Job j will take Pj/V_{ij} units of real time to finish

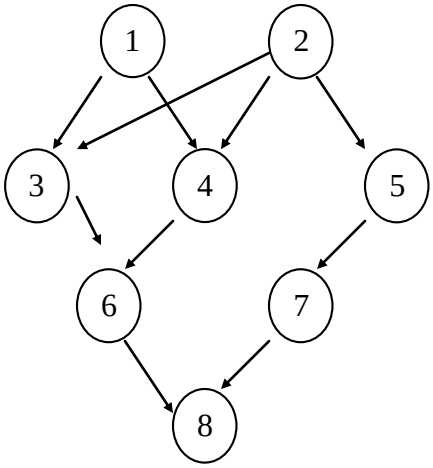
- **R** – arbitrary no. of related machines.

2.

- β - job characteristics, processing times and restrictions
- Release dates (**r_j**) – If this symbol is in the β field, job j may not start its processing before r_j. If this symbol is not in β field, job j may start at any time.
- Deadline (**$\overline{dj}$**) – Job j must be completed by its deadline ($\overline{dj}$). This is <u>hard</u> deadline.
- Preemptions (**pmtn**) – Preemption means that a job can be removed from the machine before it is completed. It will resume processing later on, possibly on a different machine. If this symbol is not in the β field, non-preemptive scheduling is implied. Preemptive will not cause loss of time.

- Precedence constraints (**prec**) – If this symbol is not in the β field, it means that the jobs are independent to each other. Otherwise, they have precedence relationships among each other.

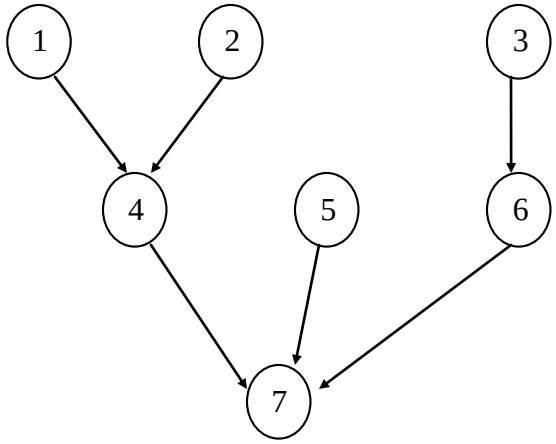
  Most general **prec** is an arbitrary directed acyclic graph.



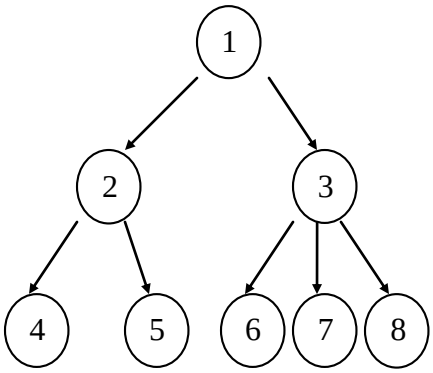
Predecessor

Successor

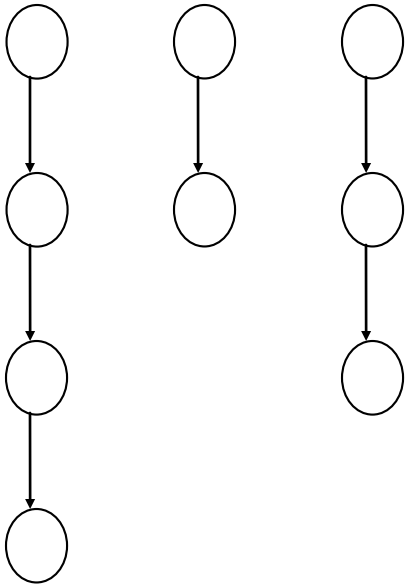
DAG - Directed Acyclic Graph



in-tree



out-tree



chains

Independent jobs

- γ - performance measures

  The performance measures we want to optimize are always a function of the completion time of the job, which of course depends on the schedule.

  C_j – completion time of the job j.

- Makespan or Schedule Length (C_{max})

  $C_{max} = \max \{C_1, C_2, \ldots, C_n\}$

- Total Completion Time (ΣC_j) (flow time)

- Total Weighted Completion Time ($\Sigma W_j C_j$)

  Each job has a weight associated with it. Job j has weight W_j

- Maximum Lateness (Lmax)

  Each job has a due date (soft deadline)

  Due date of job j is dj

  Lj = Cj – dj (can be positive, zero, or negative)

  $Lmax = \max \{L_1, L_2, \ldots, Ln\}$

  For this and later performance measures, each job has a due date.

- Number of Late Jobs (Σ Uj)

  Uj = 1 if Cj > dj

  = 0 otherwise

- Weighted Number of Late Jobs (Σ Wj Uj)
- Total Tardiness (Σ Tj)

  Tj = max {0, Cj – dj}

- Total Weighted Tardiness (ΣWj Tj)

Examples

1. $1 \mid chains \mid \Sigma Wj\ Cj$

2. $P2 \mid prec, Pj = 1 \mid Cmax$

3. $P \mid pmtn, \overline{dj} \mid \Sigma Cj$

4. $1 \mid \mid \Sigma Tj$

5. $P \mid prmp \mid Lmax$

6. $1 \mid r_j, \overline{dj} \mid \Sigma Cj$