

Recitation 6

Midterm 2 Review!



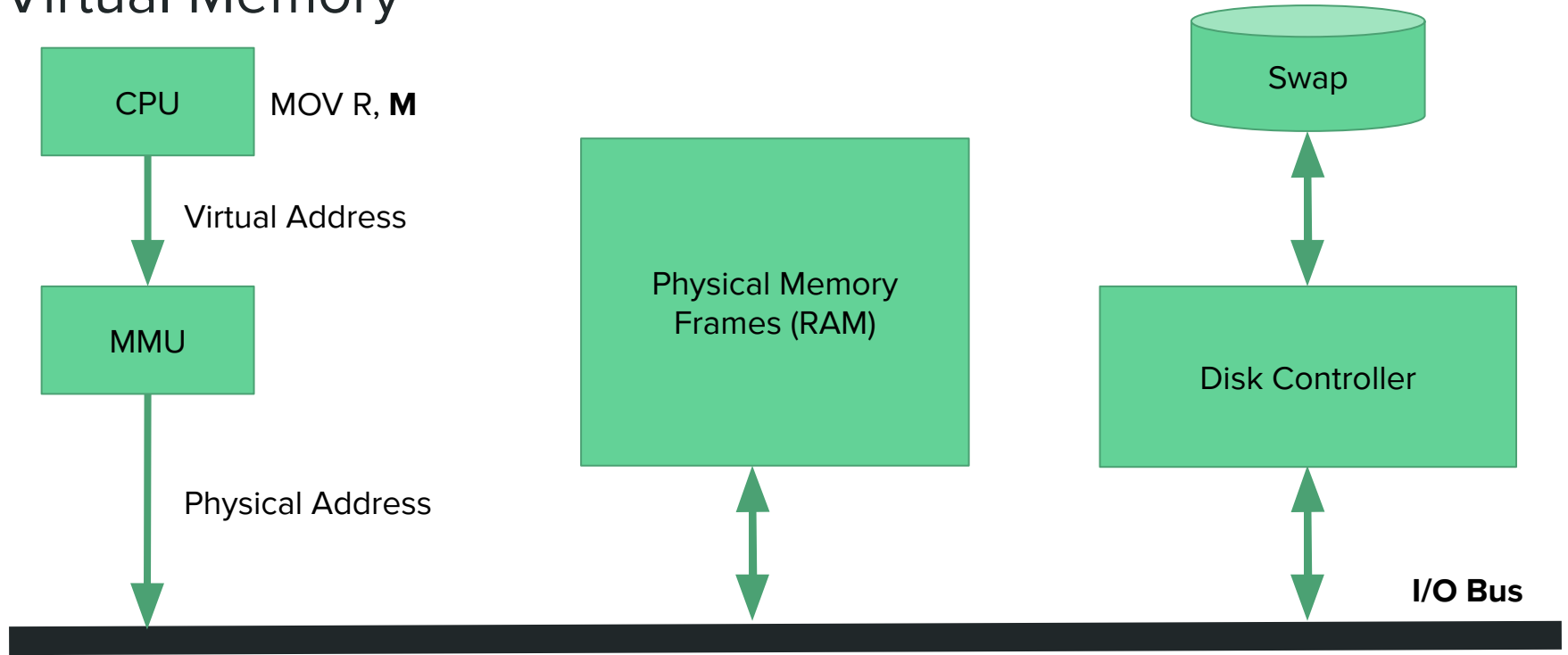
Topics Covered

- Virtual Memory
- Memory Allocation
- Threads & synchronization (lock and condition variables)
- Deadlock
- File System
- Processes
- Scheduling
- Caches

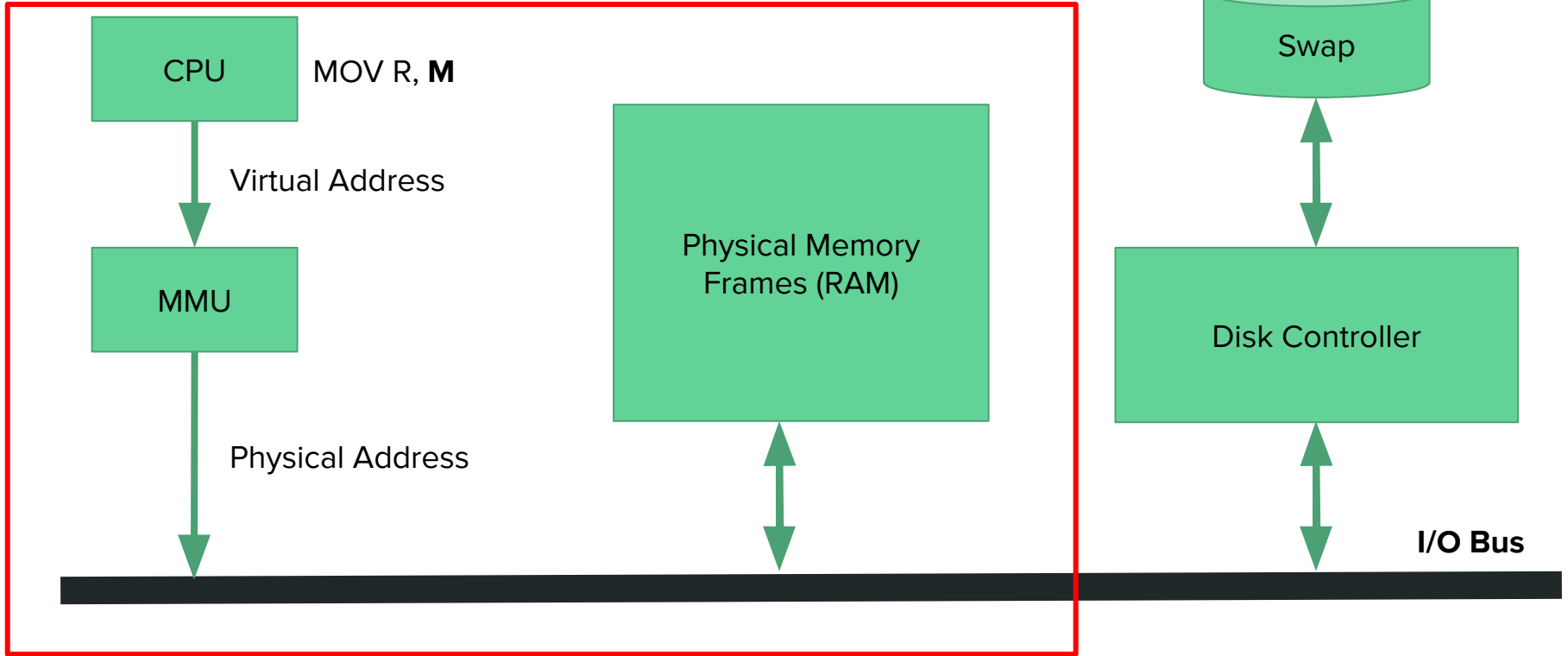
Virtual Memory

- Physical space is limited!
 - Bound by RAM and Hard Drive
- Want processes to “think” it has “unlimited” memory
 - **The x86-64 architecture (as of 2016) allows 48 bits for virtual memory and, for any given processor, up to 52 bits for physical memory. These limits allow memory sizes of 256 TiB (256×10244 bytes) and 4 PiB (4×10245 bytes), respectively.**
- Give virtual address space to each process, map each to actual physical memory, and process can actually access this memory.
- Uses special structures to achieve this translation
 - TLB
 - Page Tables
 - Multi level Page Table
 - Inverted Page Table

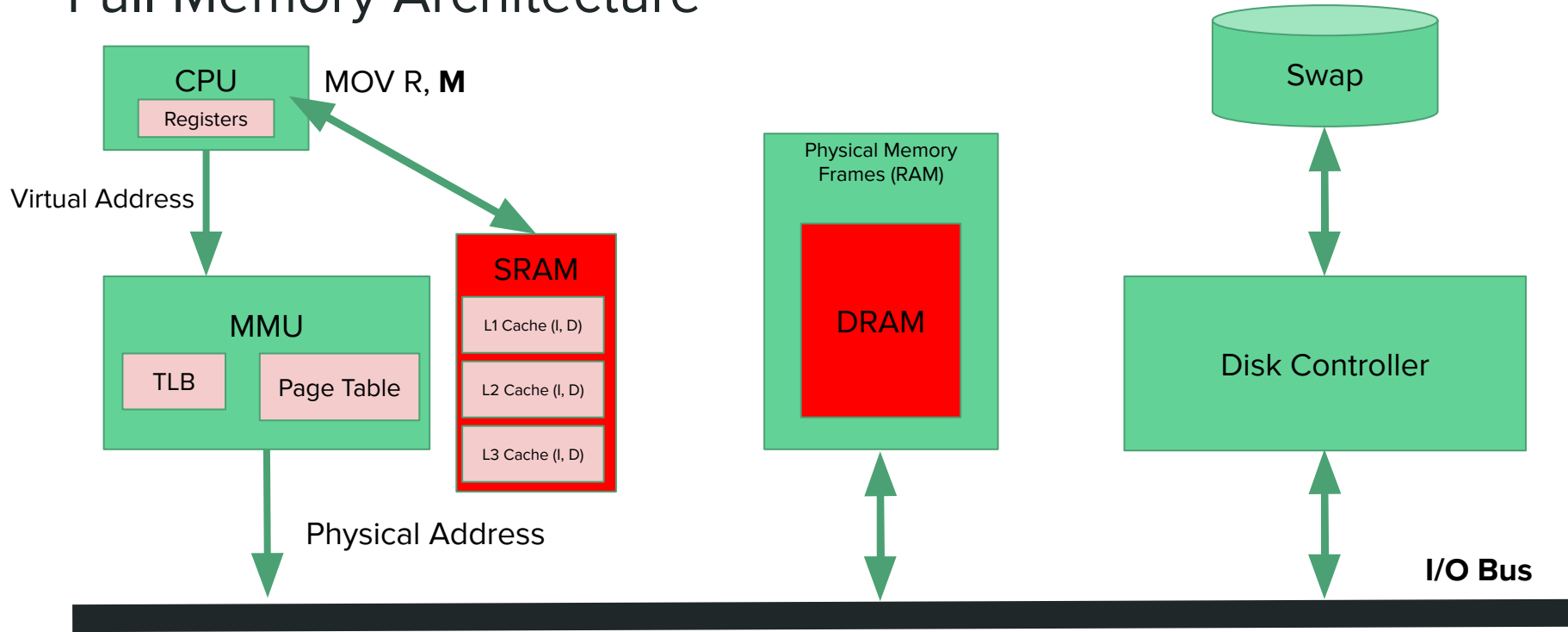
Virtual Memory



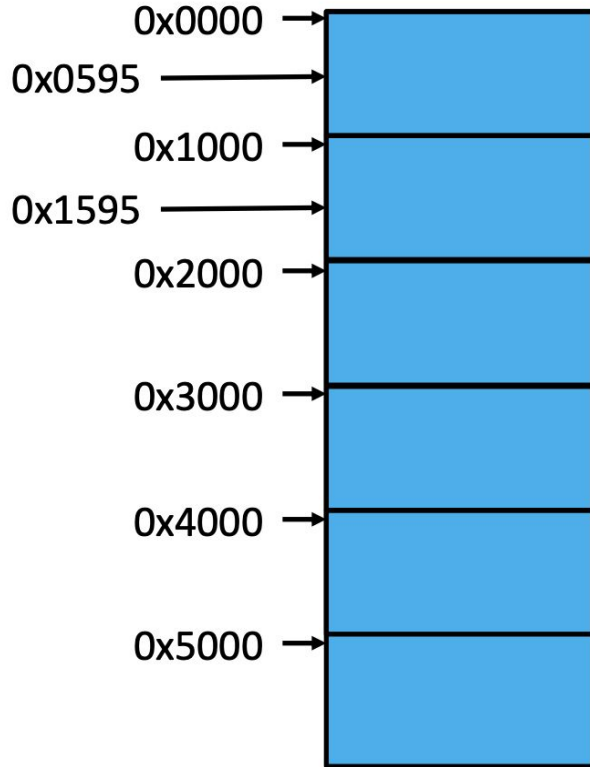
Virtual Memory



Full Memory Architecture



Memory Translation



What is the page size?

4KB / 4096 bytes

How many bits represent page offset?

12 bits

How many bits represent each page?

4 bits

How many pages?

16 pages

How many addresses per page?

4096

Least Recently Used (LRU)

- Memory is limited
- Which line of memory to evict/replace when we run out of memory?
 - LRU
- Advantages of LRU
 - Generally good performance, we are evicting a page that is “least” frequently used
 - Reduces number of page faults
- Disadvantages of LRU
 - Quite costly to find the LRU page
- Think of a scenario where LRU may actually hurt performance
 - Sequential Access: If some sequential access pattern forces LRU to evict and re-allocate parts of memory, we will have poor performance

Memory Allocation

- How to use memory the most 'efficient' way?
 - Best fit, first fit, worst fit
 - What are the pros and cons?
- Buddy Algorithm, Slab Allocator
 - What are the characteristics of each allocation method?
 - What do they assume?
 - How are internal and external fragmentation handled in each case?

Memory - Sample Question 1

A 32-bit byte-addressable system with page size of 2KiB and 2GiB of physical memory. How many bits are there for page offset bits? Virtual page bits, physical address bits? Physical page number bits?

Virtual page number bits: 21 bits

Page Offset bits: 11 bits

Physical page number bits: 20 bits

Physical page address bits: 31 bits

Virtual address (32)

Virtual page number (21)

Page offset (11)

Physical address (31)

Physical page number (20)

Page offset (11)

Memory - Sample Question 2

True or False

1. Worst-fit is always the least efficient strategy.
2. Best-Fit is slower than First-Fit because it must search the entire list every time it is called.
3. If we reduce page size, we can limit external fragmentation
4. Buddy algorithm can significantly reduce external fragmentation

Memory - Sample Question 2

True or False

1. Worst-fit is always the least efficient strategy.
 - a. **False! Worst-fit can sometimes help reduce external fragmentation**
2. Best-Fit is slower than First-Fit because it must search the entire list every time it is called.
 - a. **True! First-Fit returns as long as it finds a hole that is large enough. Therefore this option is correct**
3. If we reduce page size, we can limit external fragmentation
 - a. **False! Reducing page size generally limits internal fragmentation**
4. Buddy algorithm can significantly reduce external fragmentation
 - a. **True! Though same is not true for internal fragmentation**

Threads

- Lightweight compared to processes
- Shared Memory
- Efficient context switching
- Concurrency

Threads

- Lightweight compared to processes
 - Shared Memory
 - Efficient context switching
 - **Concurrency**
-
- Lots of ISSUES with concurrency and synchronization!!!!

Issues with Concurrency

- Data Races
- Deadlocks
- Producer Consumer Problem
- Reader / Writer Problem
- Dining Philosophers
- Methods to prevent these
 - Peterson's solution
 - Mutexes
 - TSL
 - Condition Variables
 - Monitors

Make sure to understand what each problem states, and what each measures do to prevent any issues.

What can go wrong? How do they achieve concurrency? Atomic actions? Deadlock Prevention?

Threads & Concurrency - Sample Problem 1

Consider the following implementation of Peterson's Algorithm. Why will this not work? What needs to change?

```
bool flag[2] = {false, false};  
int turn;
```

```
P0:   flag[0] = true;  
P0_gate: turn = 1;  
      while (flag[0] && turn == 1)  
      {  
          // busy wait  
      }  
      // critical section  
      ...  
      // end of critical section  
      flag[0] = false;
```

```
P1:   flag[1] = true;  
P1_gate: turn = 0;  
      while (flag[1] && turn == 0)  
      {  
          // busy wait  
      }  
      // critical section  
      ...  
      // end of critical section  
      flag[1] = false;
```


Threads & Concurrency - Sample Problem 1

Consider the following implementation of Peterson's Algorithm. Why will this not work? What needs to change?

```
bool flag[2] = {false, false};  
int turn;
```

```
P0:   flag[0] = true;  
P0_gate: turn = 1;  
      while (flag[1] && turn == 1)  
      {  
          // busy wait  
      }  
      // critical section  
      ...  
      // end of critical section  
      flag[0] = false;
```

```
P1:   flag[1] = true;  
P1_gate: turn = 0;  
      while (flag[0] && turn == 0)  
      {  
          // busy wait  
      }  
      // critical section  
      ...  
      // end of critical section  
      flag[1] = false;
```

Threads & Concurrency - Sample Problem 2

Which of the strategies would work to break the circular wait in the Dining Philosophers problem?

a. Assign each chopstick a distinct number [0 to N] in counterclockwise order. Each philosopher “i” first

grabs the right chopstick and then the left chopstick.

b. Each philosopher “i” grabs chopstick “i” and then chopstick “ $i+1 \bmod N$ ”

c. Each philosopher “i” grabs chopstick “i” and then chopstick “i+1”

d. Assign each chopstick the numbers 1 and 2 in alternate order. Each philosopher “i” grabs the adjacent chopstick of lower number and then the resource of a higher number.

Threads & Concurrency - Sample Problem 2

Which of the strategies would work to break the circular wait in the Dining Philosophers problem?

a. Assign each chopstick a distinct number [0 to N] in counterclockwise order. Each philosopher “i” first

grabs the right chopstick and then the left chopstick.

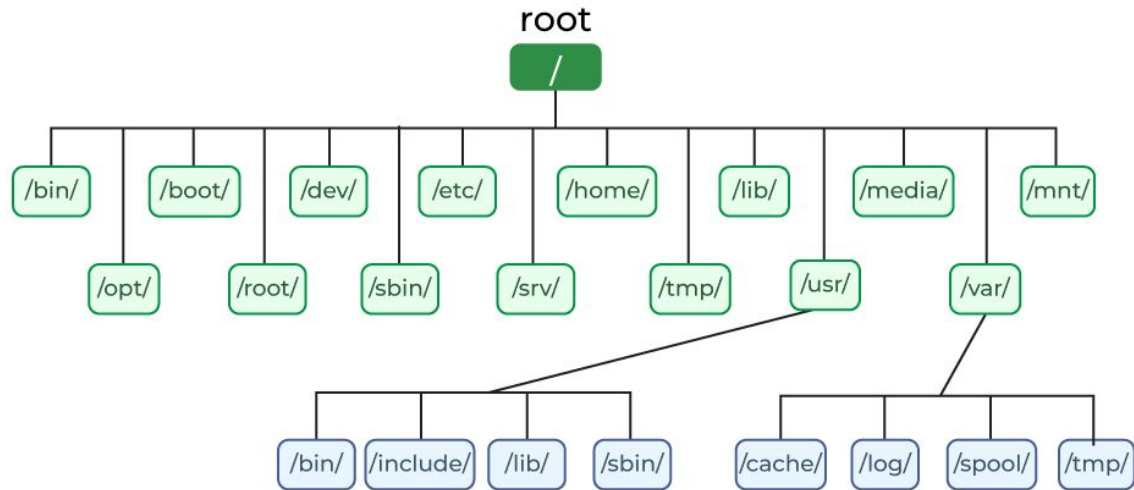
b. Each philosopher “i” grabs chopstick “i” and then chopstick “ $i+1 \bmod N$ ”

c. Each philosopher “i” grabs chopstick “i” and then chopstick “i+1”

d. Assign each chopstick the numbers 1 and 2 in alternate order. Each philosopher “i” grabs the adjacent chopstick of lower number and then the resource of a higher number.

File System

- A disk is just a random assortment of bits
- File systems help us organize them efficiently

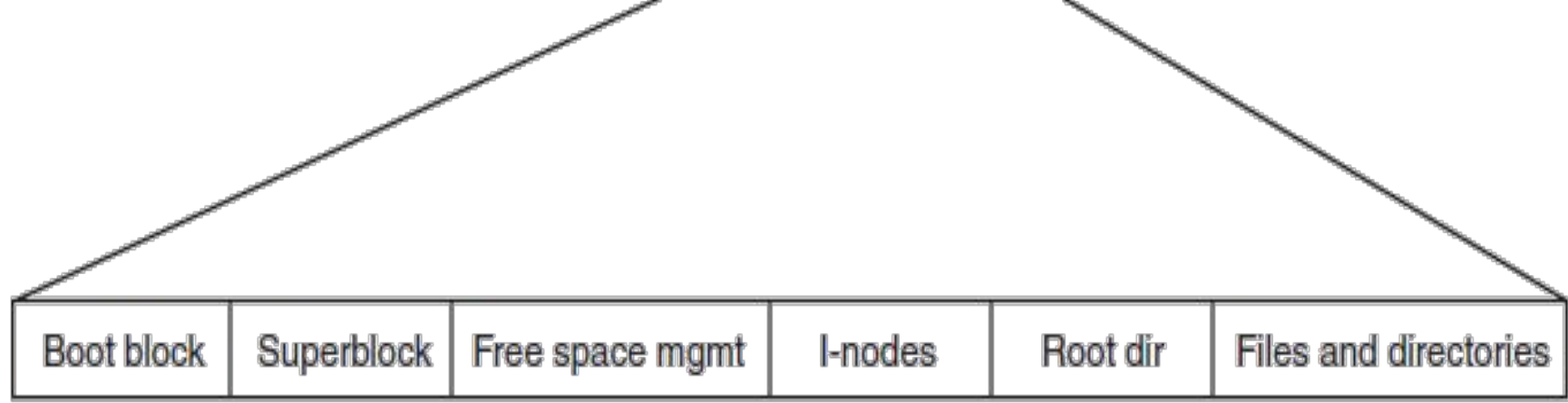


Entire disk



Partition table

Disk partition



Boot block

Superblock

Free space mgmt

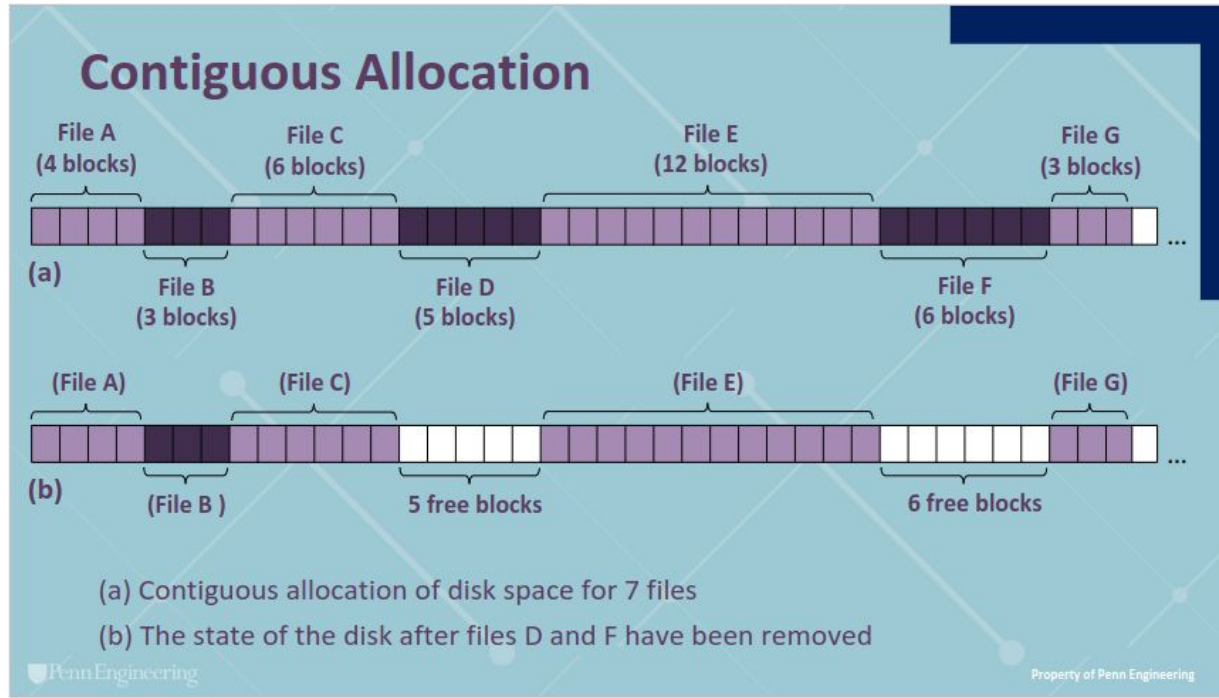
I-nodes

Root dir

Files and directories

File System - Contiguous Allocation

- Allocate memory for files together



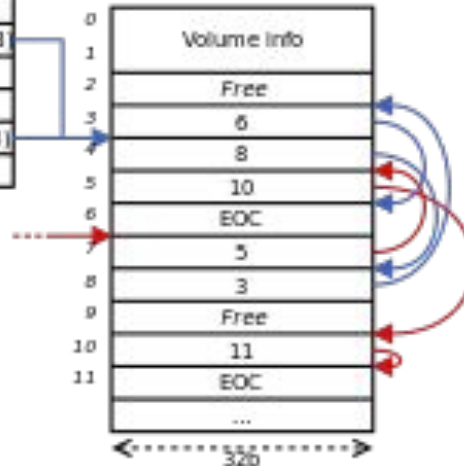
File System - FAT

- Linked-List File Organization

Directory table entry (32B)

Filename (8B)
Extension (3B)
Attributes (1B)
Reserved (1B)
Create time (3B)
Create date (2B)
Last access date (2B)
First cluster # (MSB, 2B)
Last mod. time (2B)
Last mod. date (2B)
First cluster # (LSB, 2B)
File size (4B)

File allocation table



File System - FAT

University of Pennsylvania L07: File System Intro CIS 3800, Spring 2024

FAT size

- ❖ A FAT is similar to a bitmap
 - A bitmap needs 1 bit per block
 - A FAT needs ~16-bits per block ☹️
- ❖ At least we don't need bitmap anymore!

- ❖ Grows a lot as the size of disk grows
 - As the disk grows, there are more blocks in the disk. We need more FAT entries, and each entry needs more bits. (To hold the block number. # of bits for block # grows to support more blocks)
 - **A FAT may be bigger than one block**
 - Since we need to keep the FAT in memory, this increases our memory consumption as well
 - FAT got fazed out for I-nodes (next lecture) because of this

56

File System - Inode



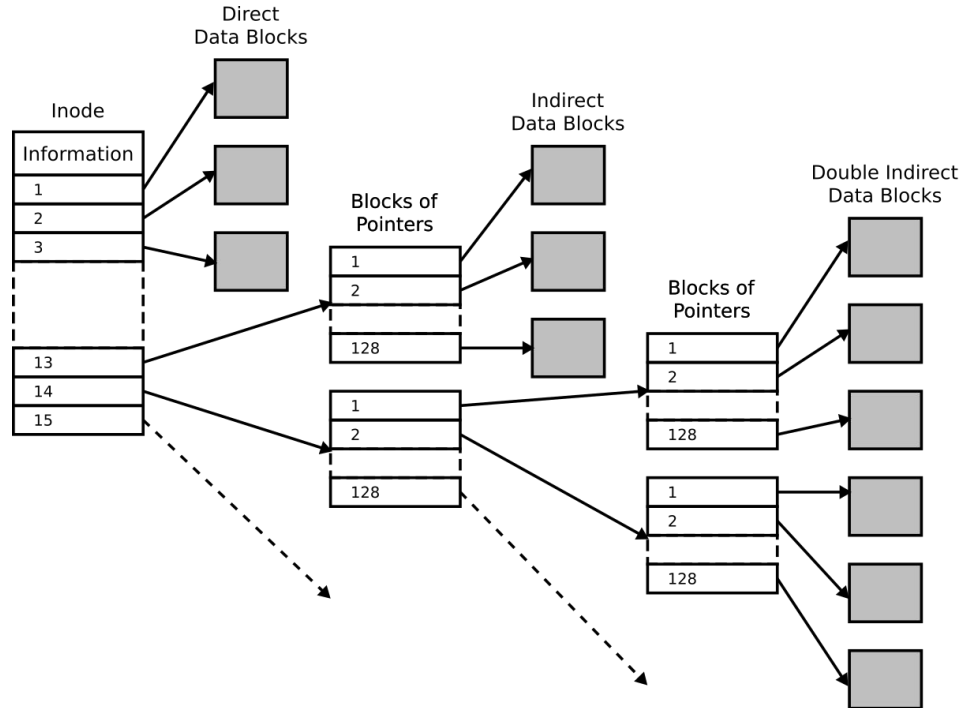
Inode motivation

- ❖ Idea: we usually don't care about ALL blocks in the file system, just the blocks for the currently open files
- ❖ Can we group the block numbers of a file together?
- ❖ Yes: we call these inodes:
 - Contains some metadata about the file and 12 physical block numbers corresponding to the first 12 logical blocks of a file

meta data
0 th phys block #
1 st phys block #
2 nd phys block #
3 rd phys block #
4 th phys block #
...
12 th phys block #

File System - Inodes

- Index Node Best Node



File System - Sample Problem 1

How many disk operations are needed to fetch the i-node for the file `/usr/ast/courses/os/handout.t`?

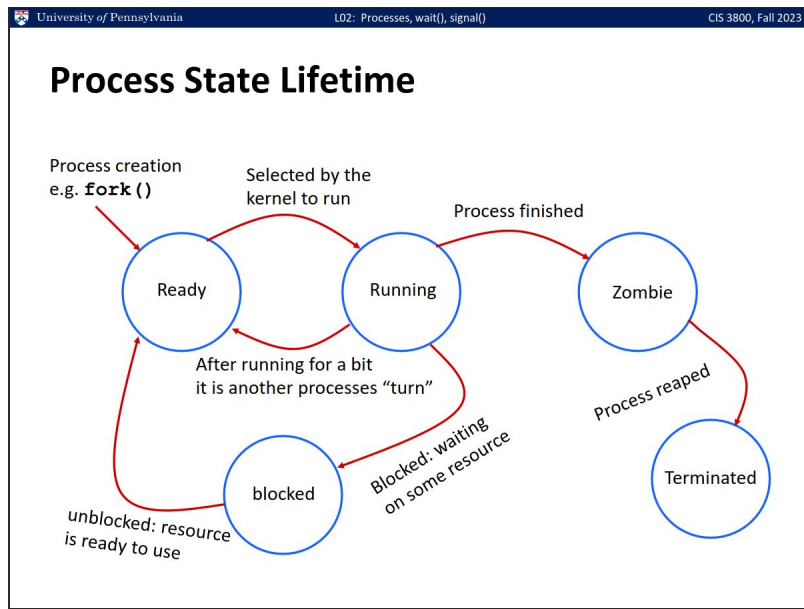
Only root directory is in RAM currently

File System - Sample Problem 2

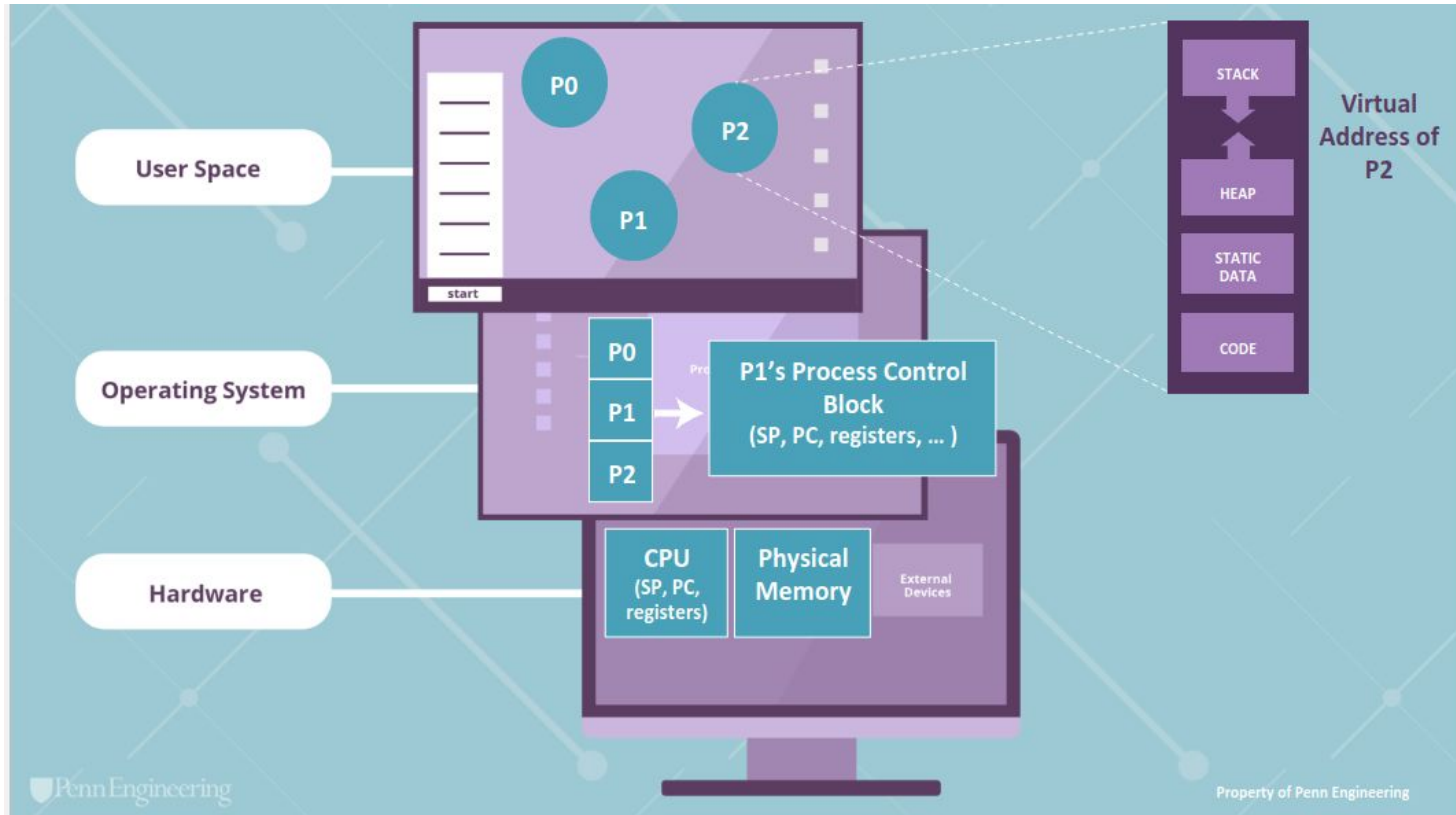
18. Consider a file whose size varies between 4 KB and 4 MB during its lifetime. Which of the three allocation schemes (contiguous, linked and table/indexed) will be most appropriate?

Processes

- A self contained program which the operating system manages
- Includes its own set of virtual memory and file descriptors
- Mostly self contained

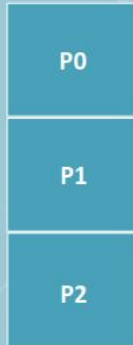


Processes



Processes

Contents of the PCB



P1's PCB

Execution Context

- Stack pointer
- Program counter
- Compute register values
- Segment register values
- Status (running, blocked, ready)

Memory Context

- Pointer to page table

Open File Descriptor Table

FD	Devices
STDIN	...
STDOUT	
STDERR	
File1	
File2	

Process

Signals

- ❖ A Process can be interrupted with various types of signals
 - This interruption can occur in the middle of most code
- ❖ Each signal type has a different meaning, number associated with it, and a way it is handled

- ❖ Examples:

- | | | |
|------------------|---|--------------------------------|
| ▪ SIGCHLD | → | Default: ignore |
| ▪ SIGINT | | |
| ▪ SIGKILL | → | Default: terminate the process |
| ▪ SIGALRM | | |
| ▪ SIGSEGV | → | Default: terminate & core dump |

Processes - Sample Problem 1

- When an interrupt or a system call transfers control to the operating system, a kernel stack area separate from the stack of the interrupted process is generally used. Why?

Processes - Sample Problem 2

- If we check out the diagram from before, what possible states or transitions are missing from the diagram?

Scheduling

- Scheduling is how we decide which process gets to run when
- What might be some goals of scheduling?

Scheduling

Goals

- ❖ The scheduler will have various things to prioritize
- ❖ Some examples:
 - ❖ Minimizing wait time
 - Get threads started as soon as possible
 - ❖ Minimizing latency
 - Quick response times and task completions are preferred
 - ❖ Maximizing throughput
 - Do as much work as possible per unit of time
 - ❖ Maximizing fairness
 - Make sure every thread can execute fairly
- ❖ These goals depend on the system and can conflict

Scheduling

Example of SJF

1 CPU

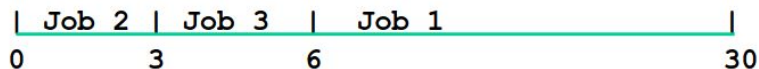
Job 2 arrives slightly after job 1.

Job 3 arrives slightly after job 2

- ❖ Same example workload with three “jobs”:

Job 1: 24 time units; Job 2: 3 units; Job 3: 3 units

- ❖ FCFS schedule:



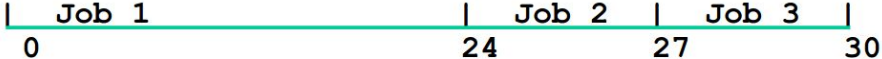
- ❖ Total waiting time: $6 + 0 + 3 = 9$
- ❖ Average waiting time: 3
- ❖ Total turnaround time: $30 + 3 + 6 = 39$
- ❖ Average turnaround time: $39/3 = 13$

Scheduling

University of Pennsylvania LIS: Scheduling & File System Intro CIS 3800, Fall 2023

Example of FCFS

1 CPU
Job 2 arrives slightly after job 1.
Job 3 arrives slightly after job 2

- ❖ Example workload with three “jobs”:
Job 1: 24 time units; Job 2: 3 units; Job 3: 3 units
- ❖ FCFS schedule:


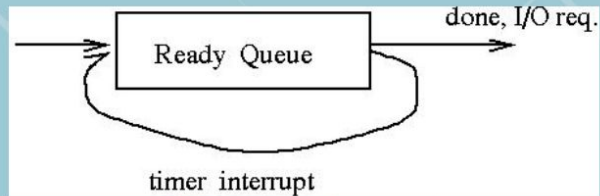
| Job 1 | Job 2 | Job 3 |
0 24 27 30
- ❖ Total waiting time: $0 + 24 + 27 = 51$
- ❖ Average waiting time: $51/3 = 17$
- ❖ Total turnaround time: $24 + 27 + 30 = 81$
- ❖ Average turnaround time: $81/3 = 27$

13

Scheduling

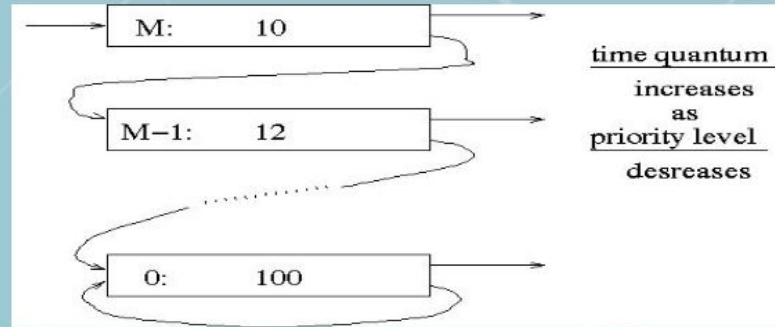
RR - Round Robin

- *Preemptive* version of FCFS
- Circular ready queue
 - arriving jobs are placed at end
 - dispatcher selects first job in queue and runs until completion of CPU burst, or until **time quantum** expires
 - if quantum expires, job is again placed at end



Scheduling

RR variant: Multi-Level Feedback (FB)



- Each priority level has a ready queue, and a time quantum
- Process enters highest priority queue initially, and (next) lower queue with each timer interrupt (penalized for long CPU usage)
- Bottom queue is standard Round Robin
- Process in a given queue not scheduled until all higher queues are empty

Scheduling

Why is quantum value so important?

- Context switching is expensive
 - hundreds of instructions on most CPUs
 - if quantum is too small, much of the CPU time will be wasted on context switching overhead
 - ... and the machine will seem slower than it really is
- Interactivity requires rapid response time
 - Interactive processes need service quickly, especially after I/O occurs
 - if quantum is too large, waiting time is increased
 - ... and the machine will seem slower than it really is
- Quantum must be \gg context switch cost, but \ll desired response time

Scheduling - Sample Problem 1

- Five jobs are waiting to be run. Their expected run times are 9, 6, 3, 5, and X. In what order should they be run to minimize average response time? (Your answer will depend on X.)

Scheduling - Sample Problem 2

round-robin scheduling algorithm.

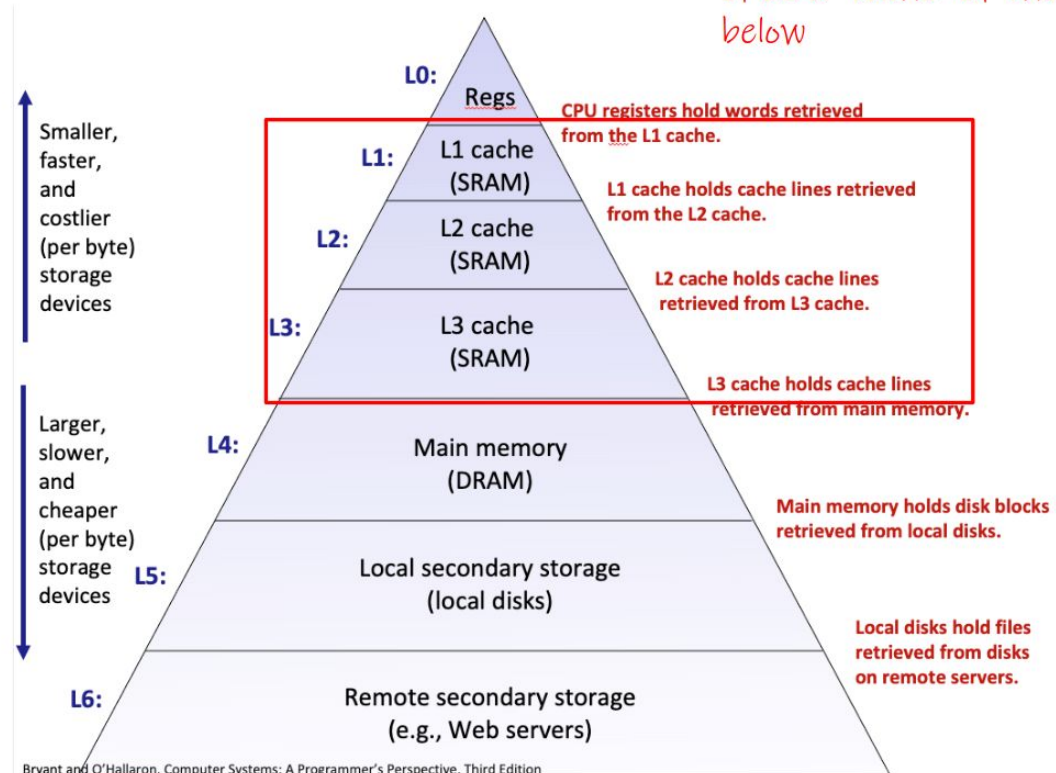
43. Measurements of a certain system have shown that the average process runs for a time T before blocking on I/O. A process switch requires a time S , which is effectively wasted (overhead). For round-robin scheduling with quantum Q , give a formula for the CPU efficiency for each of the following:

- (a) $Q = \infty$
- (b) $Q > T$
- (c) $S < Q < T$
- (d) $Q = S$
- (e) Q nearly 0

Caches

- Keep recently used memory nearby to be used again

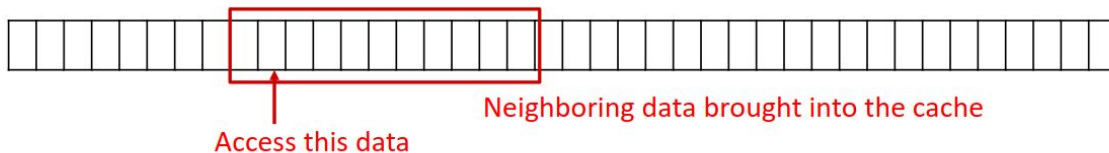
Memory Hierarchy



Caches

Cache Lines

- ❖ Imagine memory as a big array of data:



- ❖ Just like we did with pages, we can split these into 64-byte “lines” or “blocks” (64 bytes on most architectures)
 - This means bottom 6 bits of an address are the offset into a line
 - The top 58 bits of the address specify the “line” number
- ❖ When we access data at an address, we bring the whole cache line (cache block) into the L1 Cache
 - Data next to address access is thus also brought into the cache!

Caches - Sample Problem 1

11. Consider the following C program:

```
int X[N];  
int step = M; /* M is some predefined constant */  
for (int i = 0; i < N; i += step) X[i] = X[i] + 1;
```

- (a) If this program is run on a machine with a 4-KB page size and 64-entry TLB, what values of M and N will cause a TLB miss for every execution of the inner loop?
- (b) Would your answer in part (a) be different if the loop were repeated many times? Explain.

Caches - Sample Problem 2

Explain the concept of *cache hit* and *cache miss*?