Virtual Memory Overview

Computer Systems Programming, Spring 2023

Instructor: Travis McGaha

TAs:

Kevin Bernat	Jialin Cai
Mati Davis	Donglun He
Chandravaran Kunjeti	Heyi Liu
Shufan Liu	Eddy Yang

Poll Everywhere

pollev.com/tqm

- What order do following set of threads finish under:
  - First Come First Serve
  - Shortest Job First
  - Round Robin, Time Quantum = 3

Job	Arrival Time	Running Time
1	3	3
2	0	4
3	1	7

Poll Everywhere

pollev.com/tqm

- Midterm info:
  - When do you want the midterm to open & close
  - What ordering of lecture topics do you want for the week of the midterm?

Upcoming Due Dates

- HW2 (Threads) Due Monday February **2/27** @ 11:59 pm
  - Released
  - Due Date Extended
  - Due in one week

- Midterm
  - Take-home style on 3/1 or 3/2 ish: see lecture polls
  - Logistics to be released soon (next lecture)

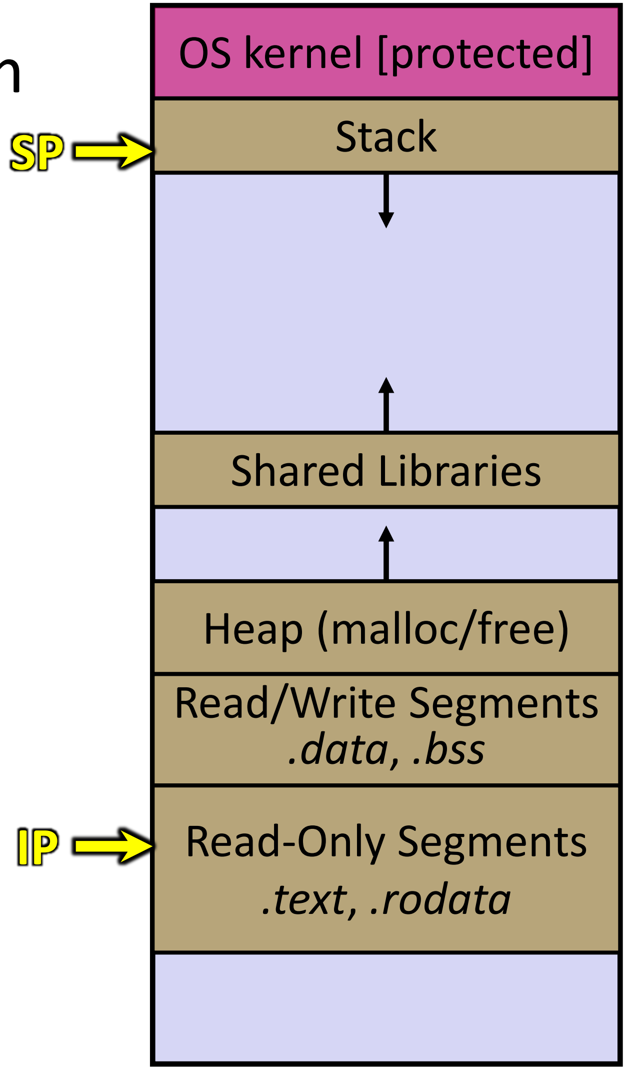
Lecture Outline

- **Motivation**
- Virtualization
- Caching

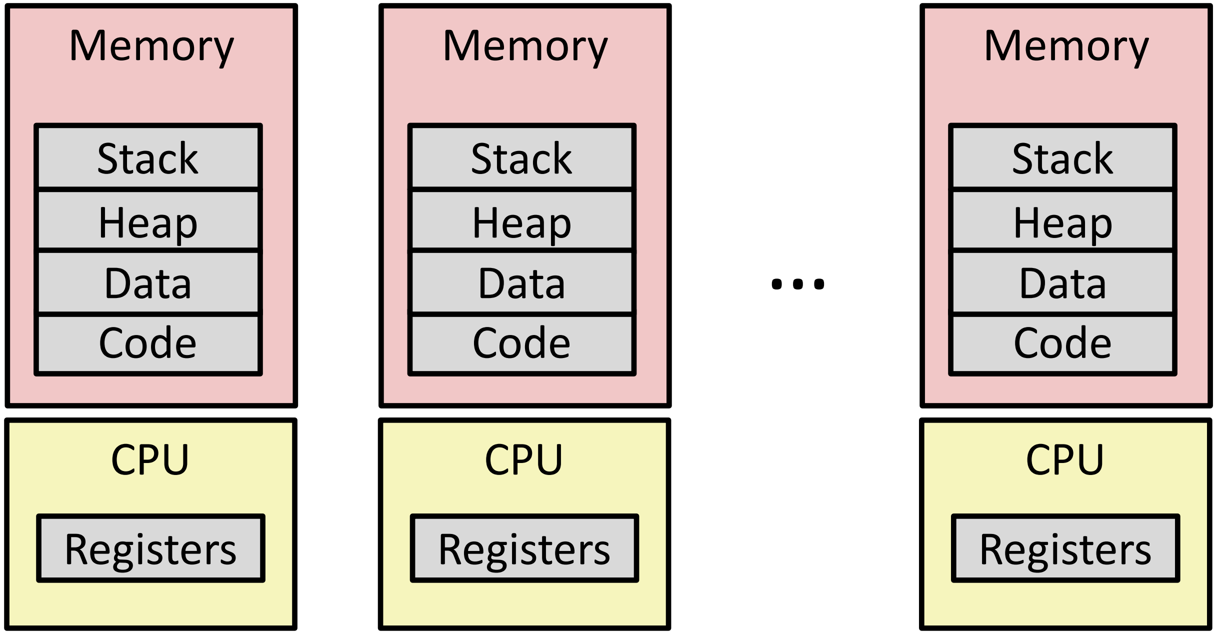
Review: Processes

- Definition: An instance of a program that is being executed
  (or is ready for execution)

- Consists of:
  - Memory (code, heap, stack, etc)
  - Registers used to manage execution
    (stack pointer, program counter, ...)
  - Other resources

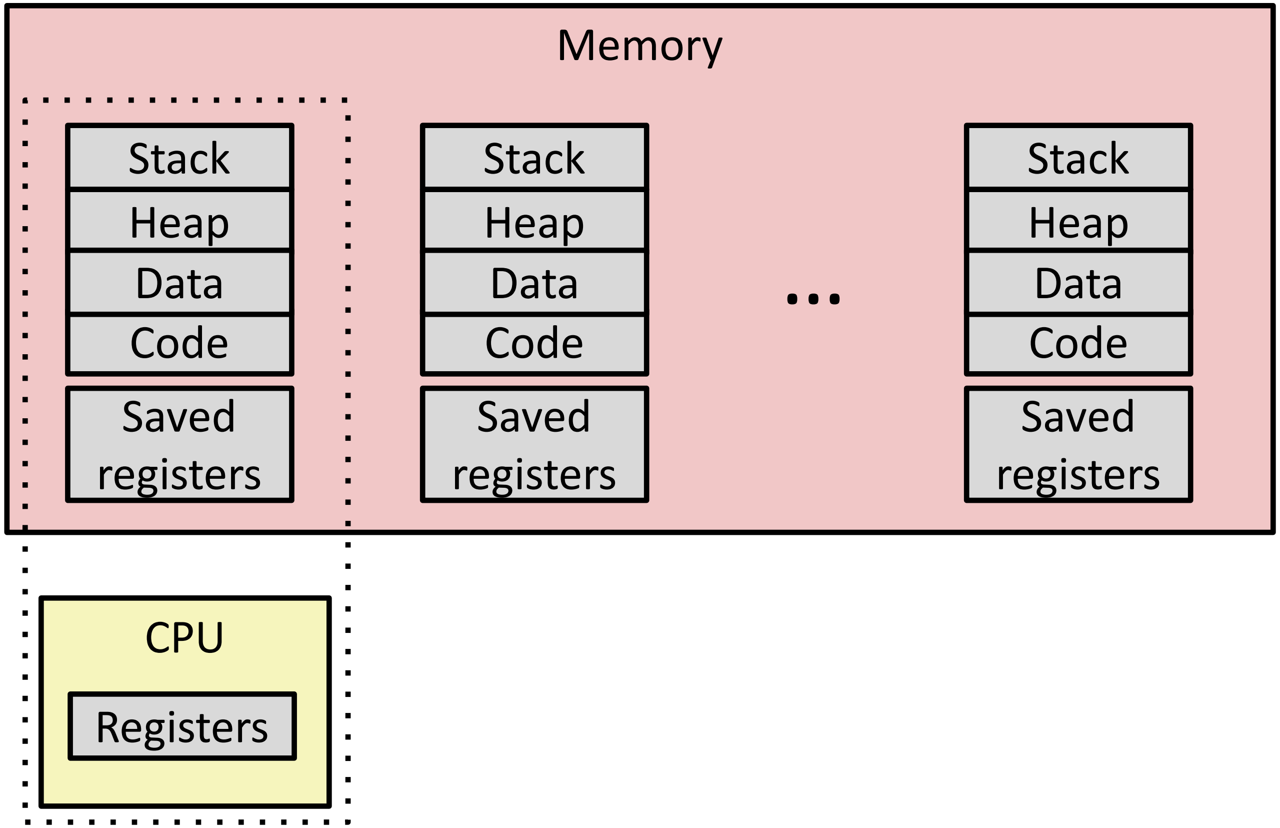


Multiprocessing: The Illusion



- Computer runs many processes simultaneously
  - Applications for one or more users
    - Web browsers, email clients, editors, …
  - Background tasks
    - Monitoring network & I/O devices

Multiprocessing: The (Traditional) Reality

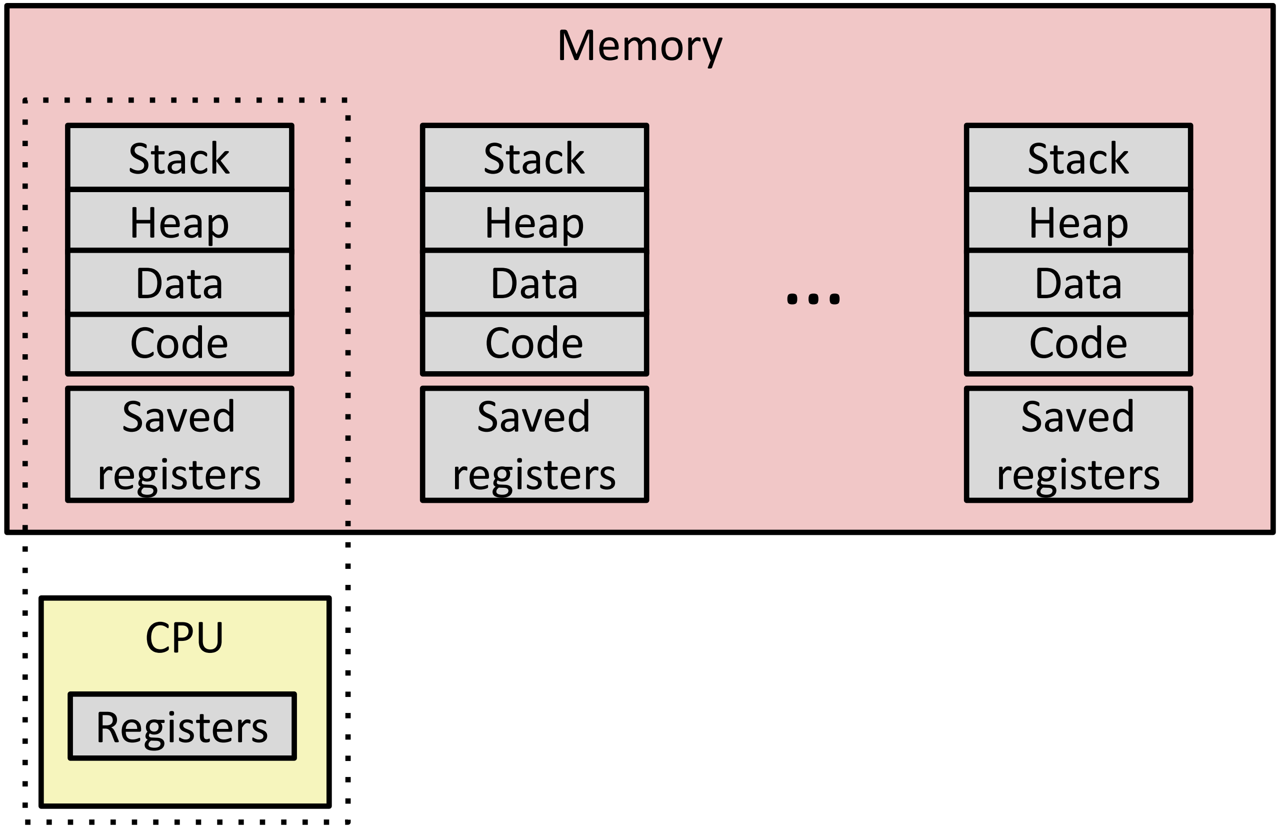


- Single processor executes multiple processes concurrently
  - Process executions interleaved (multitasking)
  - Address spaces managed by virtual memory system (later in course)
  - Register values for nonexecuting processes saved in memory

Memory (as we know it now)

- The CPU directly uses an address to access a location in memory

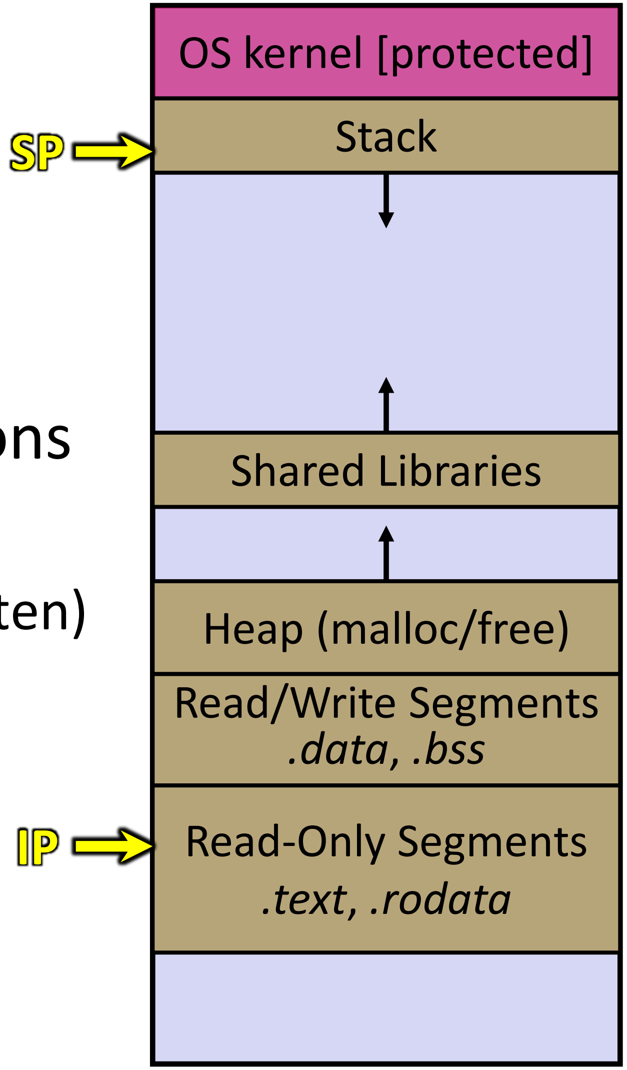
Problem 1: Sharing Memory



- How do we enforce process isolation?
  - Could one process just calculate an address into another process?

Problem 2: How do we segment things

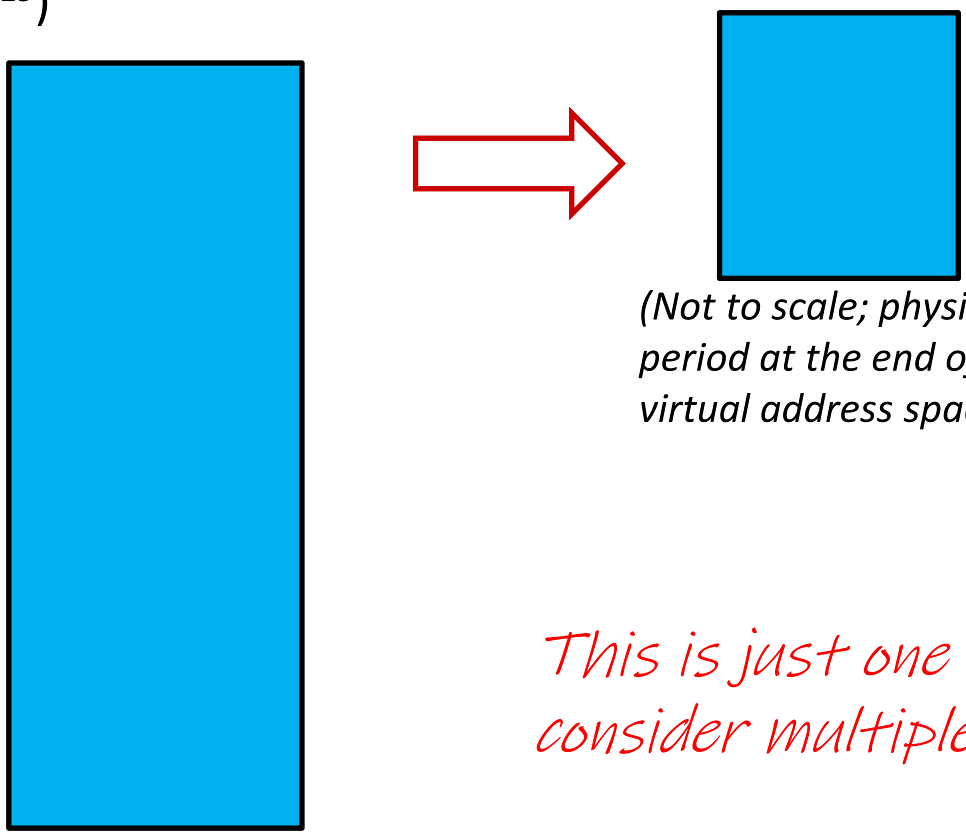
- A process' address space contains many different "segments"
- How do we keep track of which segment is which and the permissions each segment may have?
  - (e.g., that Read-Only data can't be written)



Problem 3: How does everything fit?

On a 64-bit machine, there are 2^{64} bytes, which is: 18,446,744,073,709,551,616 Bytes (1.844×10^{19})

Laptops usually have around 8GB which is 8,589,934,592 Bytes (8.589×10^{9})



(Not to scale; physical memory is smaller than the period at the end of the sentence compared to the virtual address space.)

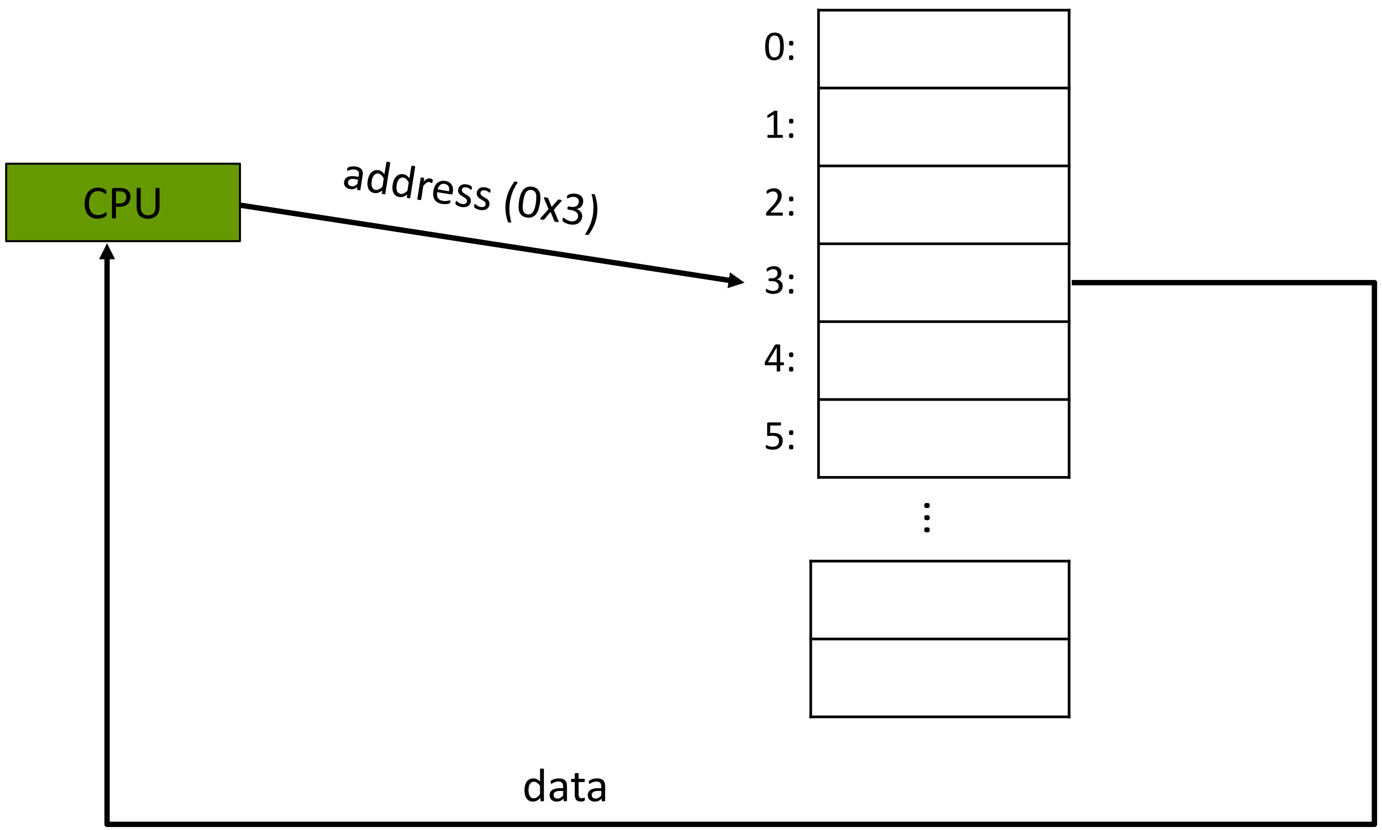
This is just one address space, consider multiple processes...

Lecture Outline

- Motivation
- **Virtualization**
- Caching

This doesn’t work anymore

- The CPU directly uses an address to access a location in memory



Idea:

- We don’t need all processes to have their data in physical memory, **just the ones that are currently running**

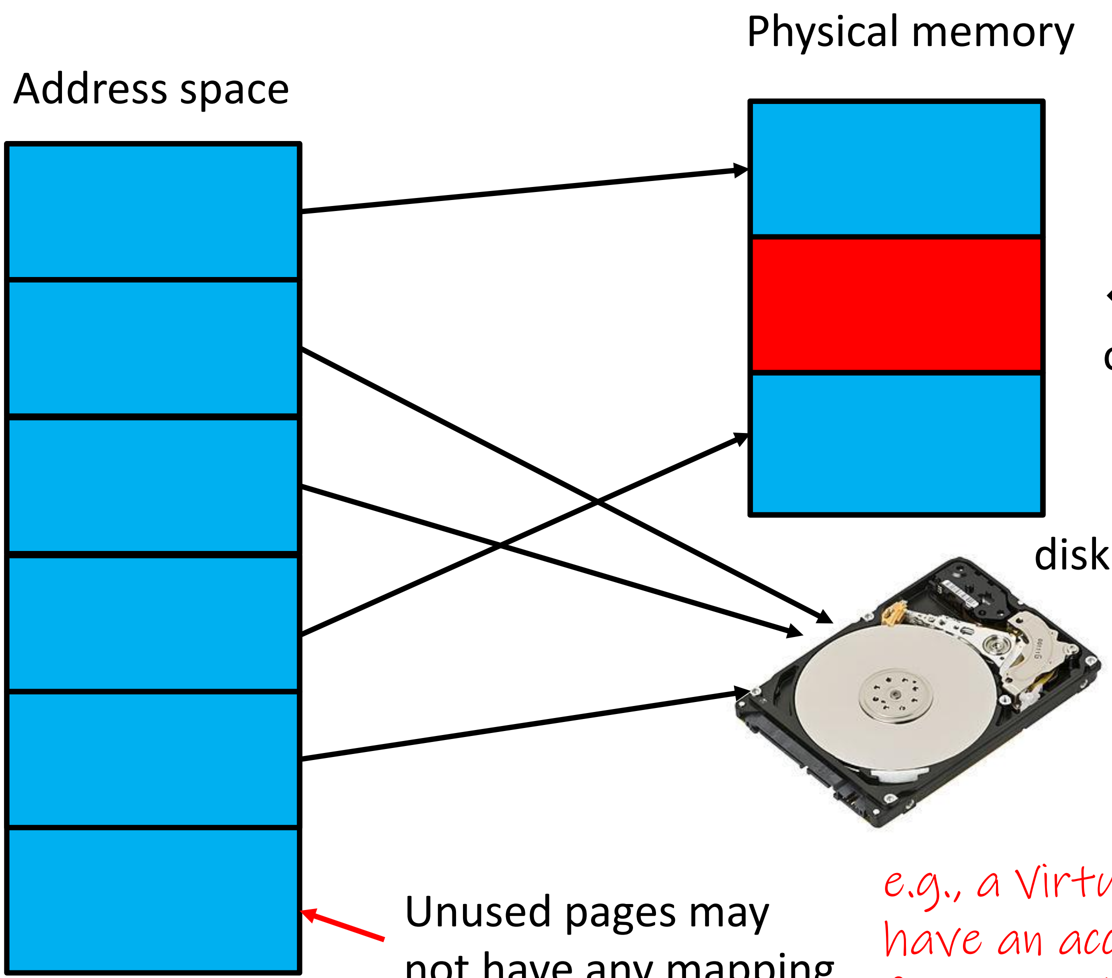
- For the process’ that are currently running: we don’t need all their data to be in physical memory, **just the parts that are currently being used**

- Data that isn’t currently stored in physical memory, can be stored elsewhere (disk).
  - Disk is "permanent storage" usually used for the file system
  - Disk has a longer access time than physical memory (RAM)

Pages

Pages are fixed size chunks ~4KB
(4 * 1024 = 4096 bytes)

- Memory can be split up into units called "pages"



Pages currently in use are stored in physical memory (RAM)

← Ram may contain pages from other active processes

Pages on physical storage are called a "Page Frame"

Pages not currently in use are stored on disk

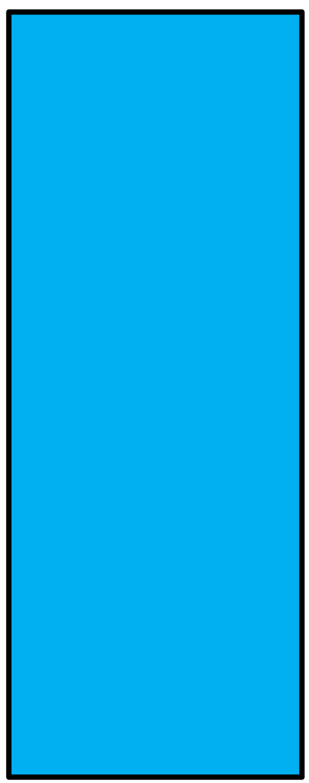
Unused pages may not have any mapping

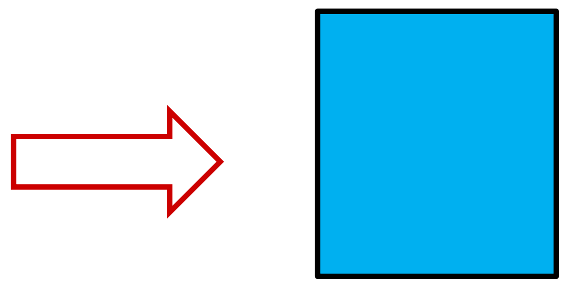
e.g., a Virtual page may not have an accompanying page frame until the page is used

Unused Pages

On a 64-bit machine, there are 2^{64} bytes, which is: 18,446,744,073,709,551,616 Bytes (1.844×10^{19})

Laptops usually have around 8GB which is 8,589,934,592 Bytes (8.589×10^{9})





(Not to scale; physical memory is smaller than the period at the end of the sentence compared to the virtual address space.)

As I write this slide, PowerPoint is using 212.7MB which is: 223,032,115 Bytes (2.230×10^{7})

Some programs don't need 2^{64} bytes, so several pages may never be used

Indirection

- "Any problem in computer science can be solved by adding another level of indirection."
  - David wheeler, inventor of the subroutine (e.g. functions)

- The ability to indirectly reference something using a name, reference or container instead of the value itself. A flexible mapping between a name and a thing allows chagcing the thing without notifying holders of the name.
  - May add some work to use indirection
  - Example: Phone numbers can be transferred to new phones

- Idea: instead of directly referring to physical memory, add a level of indirection

Definitions

Sometimes called "virtual memory" or "virtual address space"

- **Addressable Memory**: the total amount of memory that can be theoretically be accessed based on:
  - number of addresses ("address space")
  - bytes per address ("addressability")

IT MAY NOT EXIST ON REAL HARDWARE

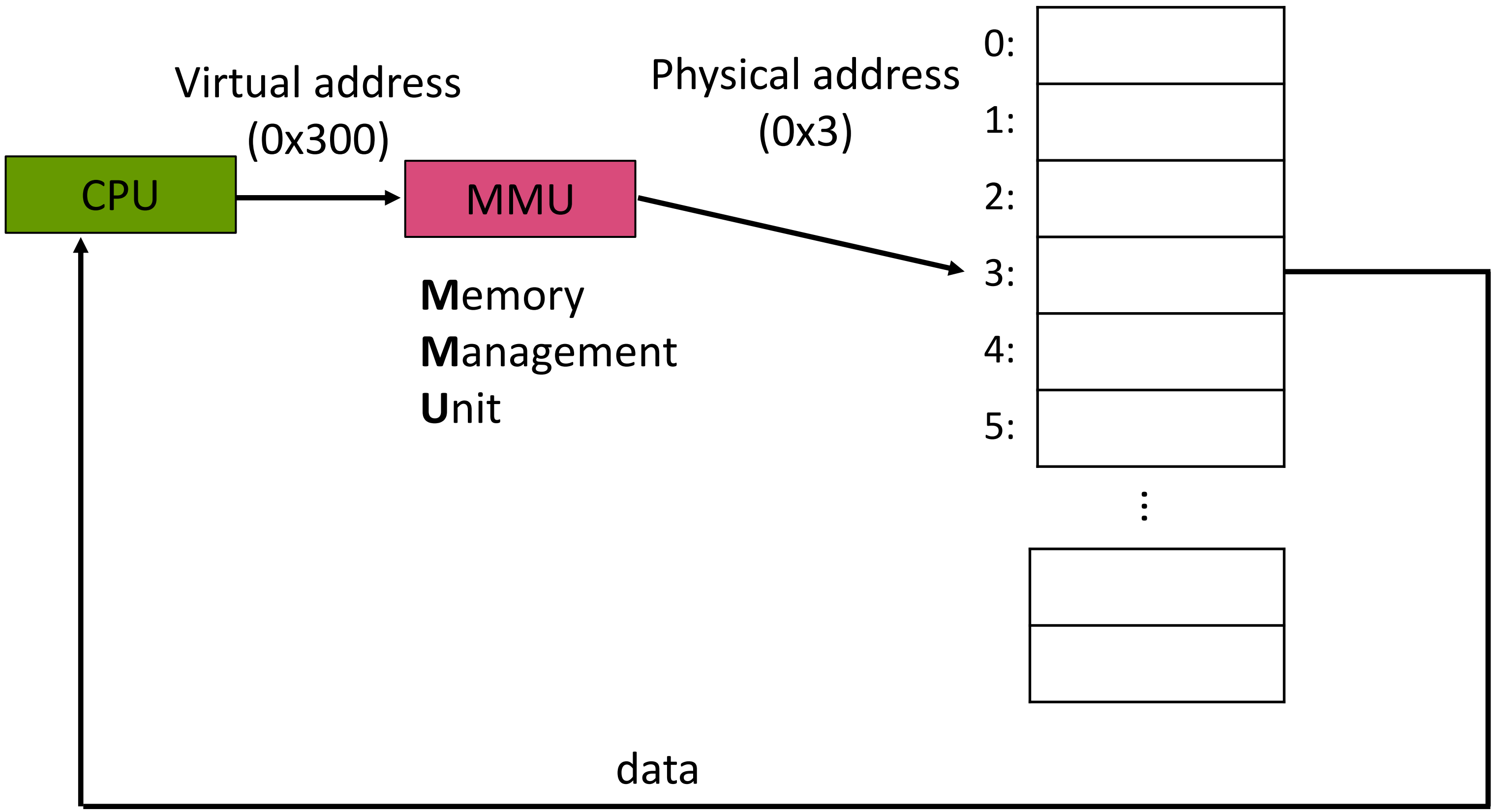
- **Physical Memory**: the total amount of memory that is physically available on the computer

Adding Addressable Memory + Physical Memory doesn't make sense

- **Virtual Memory**: An abstraction technique for making memory look larger than it is and hides many details from the programs.

Virtual Address Translation

- Programs don't know about physical addresses; virtual addresses are translated into them by the MMU



Page Tables

More details about translation on Wednesday

- Virtual addresses can be converted into physical addresses via a page table.

- There is one page table per processes, managed by the MMU

Virtual page #	Valid	Physical Page Number
0	0	null
1	1	0
2	1	1
3	0	disk

Valid determines if the page is in physical memory

If a page is on disk, MMU will fetch it

Lecture Outline

- Motivation
- Virtualization
- **Caching**

Problem: Paging Replacement

- We don't have space to store all active pages in physical memory.
- If we need to load in a page from disk, how do we decide which page in physical memory to "evict"
- Goal: Minimize the number of times we have to go to disk. It takes a while to go to disk.

Paging Replacement Algorithms

- Simple Algorithms:
  - Random choice
    - “dumbest” method, easy to implement
  - FIFO
    - Replace the page that has been in physical memory the longest

- Both could evict a page that is used frequently and would require going to disk to retrieve it again.

(Theoretically) Optimal Algorithm

- If we knew the precise sequence of requests for pages in advance, we could optimize for smallest overall number of faults
  - Always replace the page to be used at the farthest point in future
  - Optimal (but unrealizable since it requires us to know the future)

- Off-line simulations can estimate the performance of a page replacement algorithm and can be used to measure how well the chosen scheme is doing

- Optimal algorithm can be approximated by using the past to predict the future

Least Recently Used (LRU)

- Assume pages used recently will be used again soon
  - Throw out page that has been unused for longest time
- Past is usually a good indicator for the future
- LRU has significant overhead:
  - A timestamp for *each* memory access that is updated in the page table
  - Sorted list of pages by timestamp

How to Implement LRU?

- Counter-based solution:
  - Maintain a counter that gets incremented with each memory access
  - When we need to evict a page, pick the page with lowest counter
- List based solution
  - Maintain a linked list of pages in memory
  - On every memory access, move the accessed page to end
  - Pick the front page to evict
- HashMap and LinkedList
  - Maintain a hash map and a linked list
  - The list acts the same as the list-based solution
  - The HashMap has keys that are the page number, values that are pointers to the nodes in the linked list to support O(1) lookup