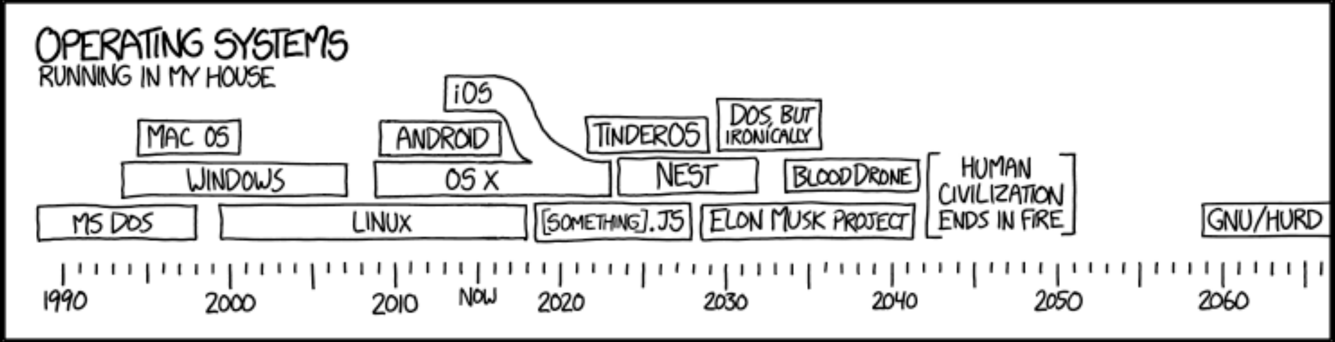
Great Ideas in Computer Architecture

Operating Systems, Virtual Memory Intro

Instructor: Stephan Kaminsky



Agenda

- OS Intro
- OS Boot Sequence and Operation
- Multiprogramming/time-sharing
- Introduction to Virtual Memory
- Summary

CS61C so far…

C Programs

```
#include <stdlib.h>

int fib(int n) {
  return
    fib(n-1) +
    fib(n-2);
}
```

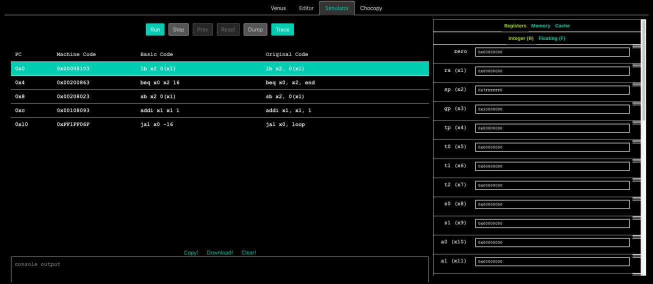
Project 1

RISC-V Assembly

```
.foo
lw   t0, 4(s0)
addi t1, t0, 3
beq  t1, t2, foo
nop
```

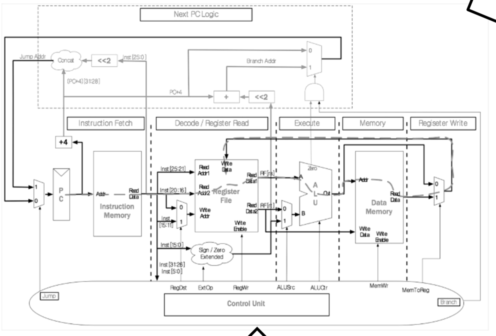
Labs

Project 2

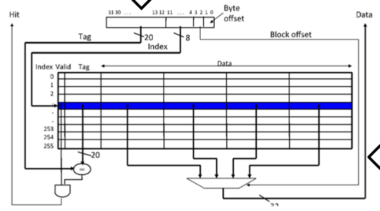


Project 3

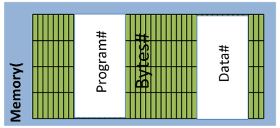
CPU



Caches

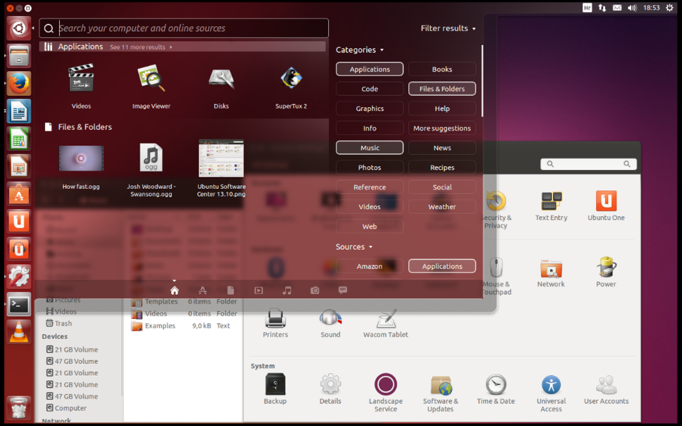


Memory



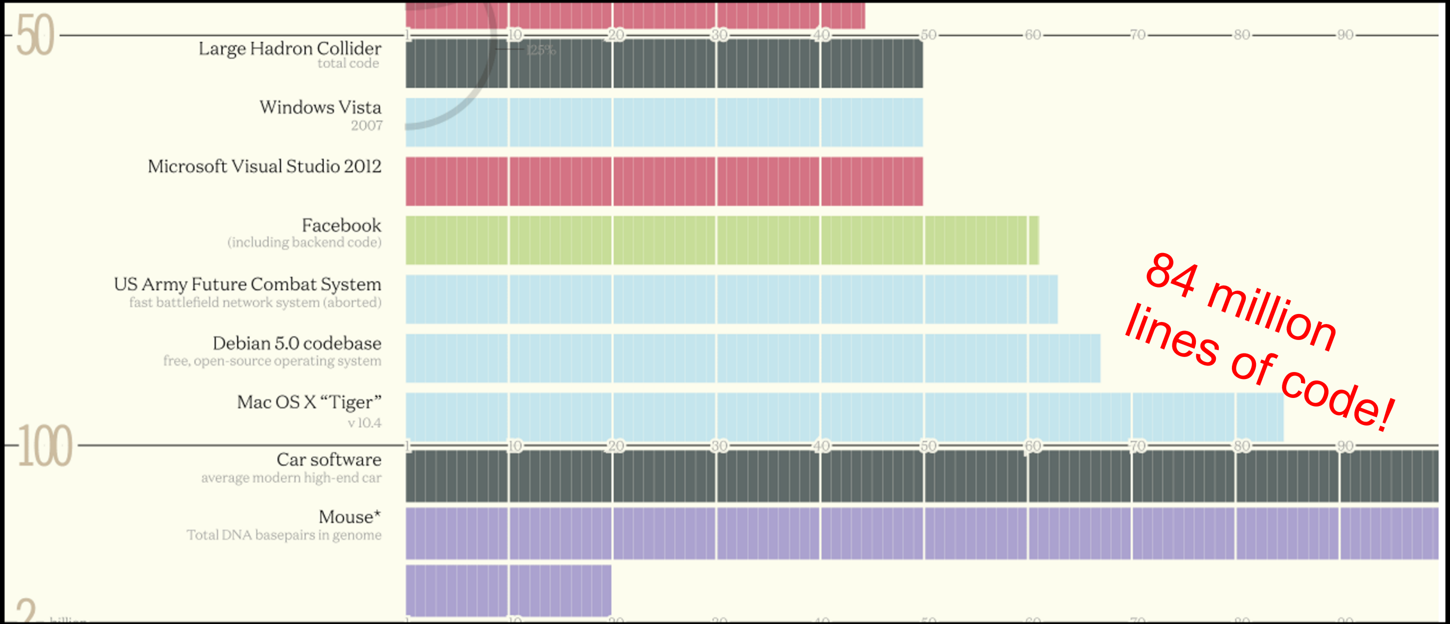
But wait…

- When we run Venus, it only executes one program and then stops.
- When I switch on my computer, I have many programs:



Yes, but that's just software! The Operating System (OS)

Well, "just software"



Codebases (in millions of lines of code). CC BY-NC 3.0 — David McCandless © 2015
http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

What is an operating system?

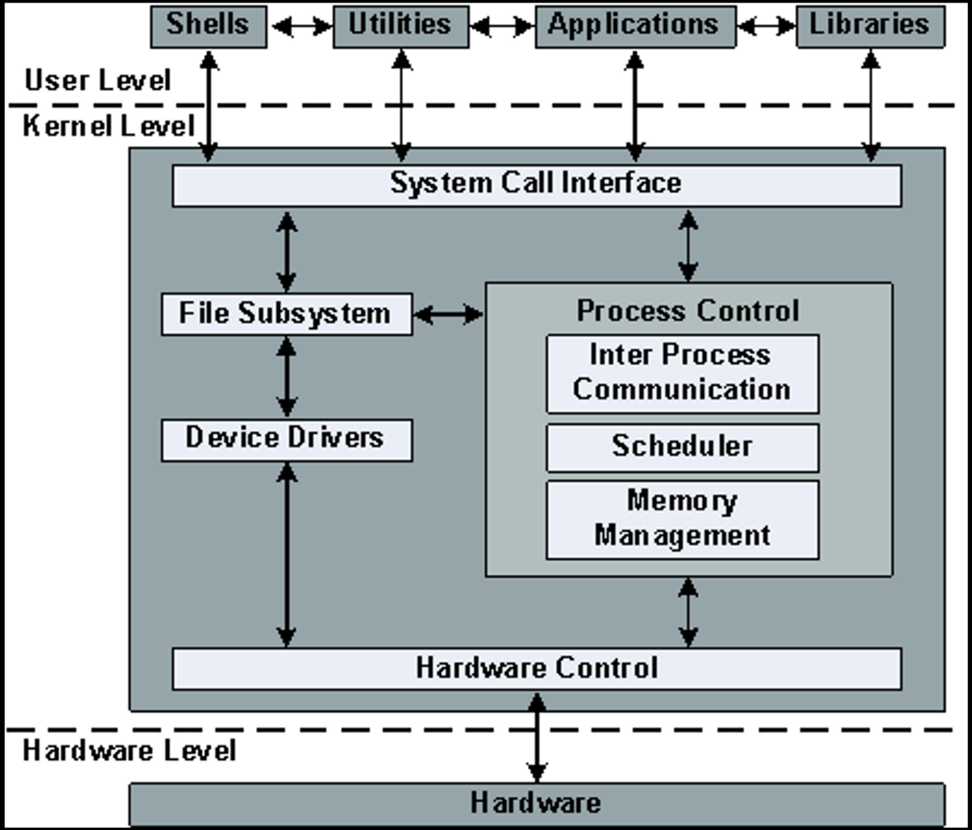
Applications (“Software”)
Operating System
Computer Hardware (CPU, SSD/HD, RAM, …)

Operating systems control <u>how software applications access and use hardware on your computer</u>. They provide a general interface for common actions (ex. reading/writing disk) and allow software to run without knowledge of the machine it lives on!

What does the OS do?

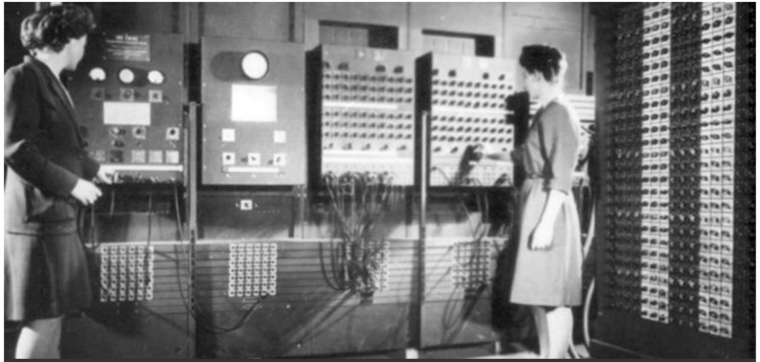
- One of the first things that runs when your computer starts (right after firmware/bootloader)
- Loads, runs and manages programs:
  - Multiple programs at the same time (time-sharing)
  - Isolate programs from each other (isolation)
  - Multiplex resources between applications (e.g., devices)
- Services: File System, Network stack, etc.
- Finds and controls all the devices in the machine in a general way (using “device drivers”)

What is an operating system?



Have we always had OS?

Operating “systems” used to just be “operators”; these were people, usually women!



Two women operating ENIAC (U.S. Army)

As late as the 1960s many people perceived computer programming as a natural career choice for savvy young women. Even the trend-spotters at Cosmopolitan Magazine urged their fashionable female readership to consider careers in programming. In an article titled “The Computer Girls,” the magazine described the field as offering better job opportunities for women than many other professional careers. As computer scientist Dr. Grace Hopper told a reporter, programming was “just like planning a dinner. You have to plan ahead and schedule everything so that it’s ready when you need it.... Women are ‘naturals’ at computer programming.” James Adams, the director of education for the Association for Computing Machinery, agreed: “I don’t know of any other field, outside of teaching, where there’s as much opportunity for a woman.”

What’s “different” about various OS?

- Different experience for the user
  - Organisation, appearance, etc.
- Different interfaces for applications
  - Windows software won’t run on Mac OS!
- Different levels of licensing, availability, hardware support
  - Linux is open source
  - MacOS can only run on Mac machines (sorta…)
  - Windows can be purchased independently of a Microsoft computer

Unix based, or … not

- In CS we often prefer systems that are "Unix-based" but what does that mean?
  - Unix was developed in AT&T's Bell Labs back in the mid-to-late 1960's.
  - Built modularly, strong file system core
  - MacOS, Linux descended from this! Berkeley (BSD) played a part!
- Windows developed independently of this unix craze!

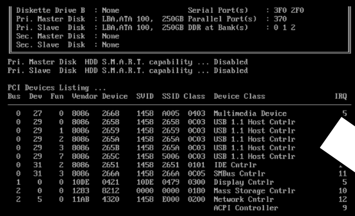
Agenda

- OS Intro
- OS Boot Sequence and Operation
- Multiprogramming/time-sharing
- Introduction to Virtual Memory
- Summary

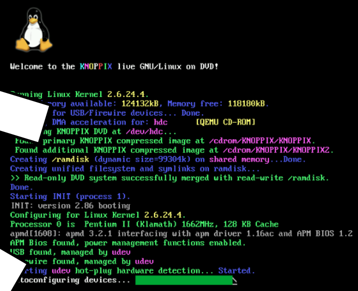
What happens at boot?

- When the computer switches on, it does the same as Venus: the CPU executes instructions from some start address (stored in Flash ROM)

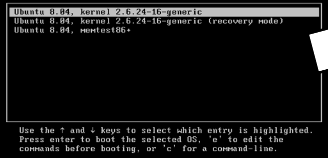
1. BIOS: Find a storage device and load first sector (block of data)



4. Init: Launch an application that waits for input in loop (e.g., Terminal/Desktop/...



2. Bootloader (stored on, e.g., disk): Load the OS *kernel* from disk into a location in memory and jump into it.



3. OS Boot: Initialize services, drivers, etc.

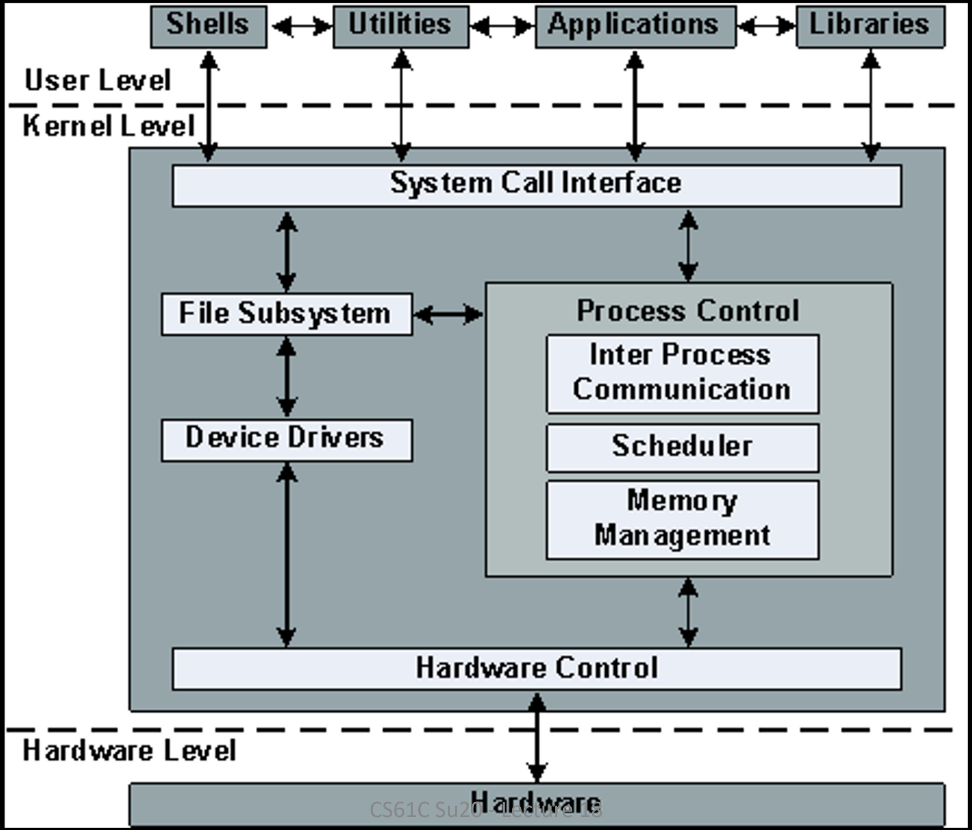
Launching Applications

- Applications are called “processes” in most OSs.
- Created by another process calling into an OS routine (using a “syscall”, more details later).
  - Depends on OS, but Linux uses fork (see OpenMP threads) to create a new process, and execve to load application.
- Loads executable file from disk (using the file system service) and puts instructions & data into memory (.text, .data sections), prepare stack and heap.
- Set argc and argv, jump into the main function.

Supervisor Mode

- If something goes wrong in an application, it can crash the entire machine. What about malware, etc.?
- The OS may need to enforce resource constraints to applications (e.g., access to devices).
- To protect the OS from the application, CPUs have a supervisor mode bit (also need isolation, more later).
  - You can only access a subset of instructions and (physical) memory when not in supervisor mode (user mode).
  - You can change out of supervisor mode using a special instruction, but not into it (unless there is an interrupt).

What is an operating system?



Syscalls

- How to switch back to OS? OS sets timer interrupt, when interrupts trigger, drop into supervisor mode.
- What if we want to call into an OS routine? (e.g., to read a file, launch a new process, send data, etc.)
  - Need to perform a syscall: set up function arguments in registers, and then raise software interrupt
  - OS will perform the operation and return to user mode
- This way, the OS can mediate access to all resources, including devices, the CPU itself, etc.

Syscalls in Venus

- Venus provides many simple syscalls using the `ecall` RISC-V instruction
- How to issue a syscall?
  - Place the syscall number in a0
  - Place arguments to the syscall in the a1 register
  - Issue the `ecall` instruction
- This is how your RISC-V code has been able to produce output all along
- `ecall` details depend on the ABI (Application Binary Interface)

Example Syscall

- Let's say we want to print an integer stored in `s3`:

Print integer is syscall #1

```
li    a0, 1
add  a1, s3, x0
ecall
```

Venus's Environmental Calls

ID (a0)	Name	Description
1	print_int	prints integer in `a1`
4	print_string	prints the null-terminated string whose address is in `a1`
9	sbrk	allocates `a1` bytes on the heap, returns pointer to start in `a0`
10	exit	ends the program
11	print_character	prints ASCII character in `a1`
17	exit2	ends the program with return code in `a1`

More can be found here: https://github.com/ThaumicMekanism/venus/wiki/Environmental-Calls

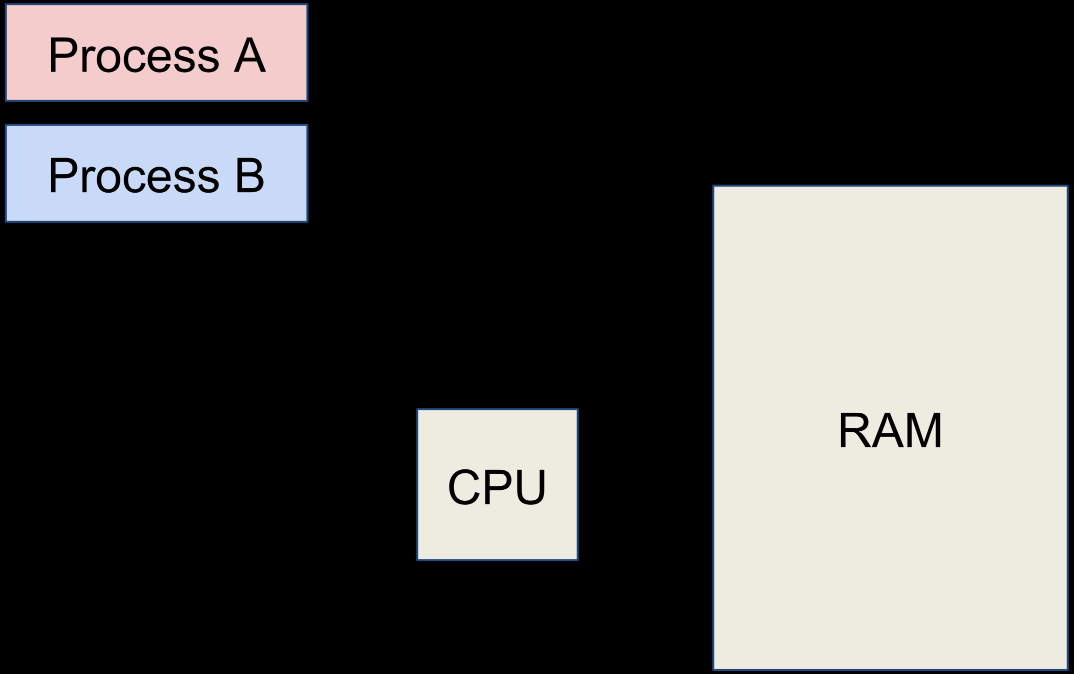
Agenda

- OS Intro
- OS Boot Sequence and Operation
- Multiprogramming/time-sharing
- Introduction to Virtual Memory
- Summary

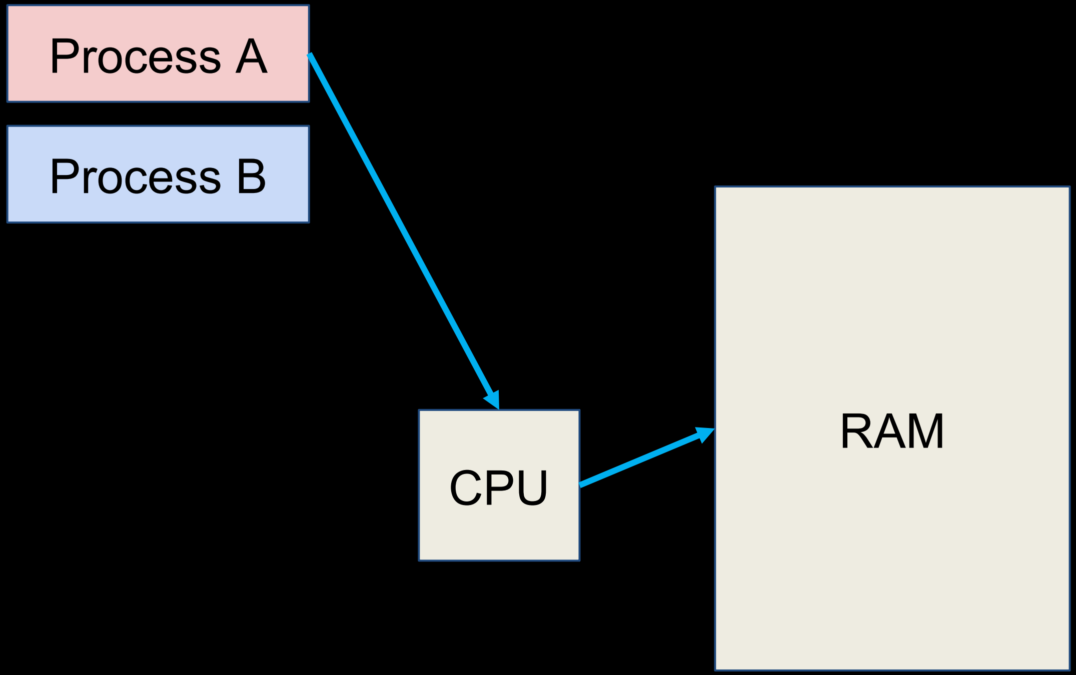
Multiprogramming

- OS runs multiple applications at the same time.
- But not really (unless have a core per process)
- Switches between processes very quickly. This is called a “context switch”.
- Deciding what process to run is called scheduling.
  - Programs can be scheduled in a variety of ways!
  - Most/least resources needed, “fastest” to run, most important, etc.

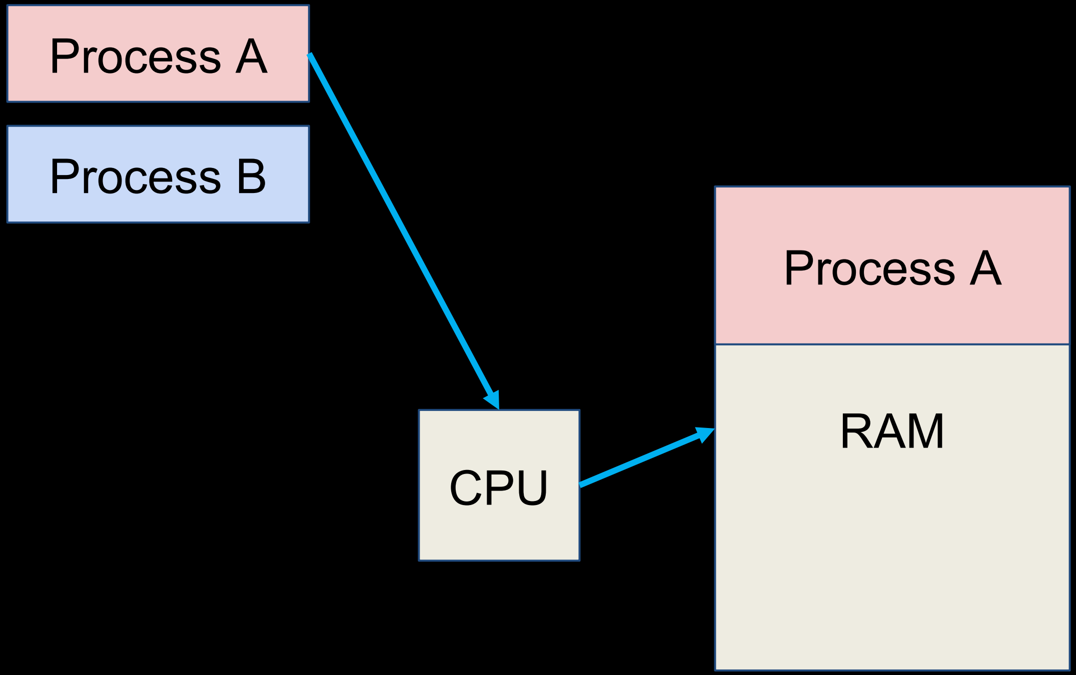
User mode: multiple applications



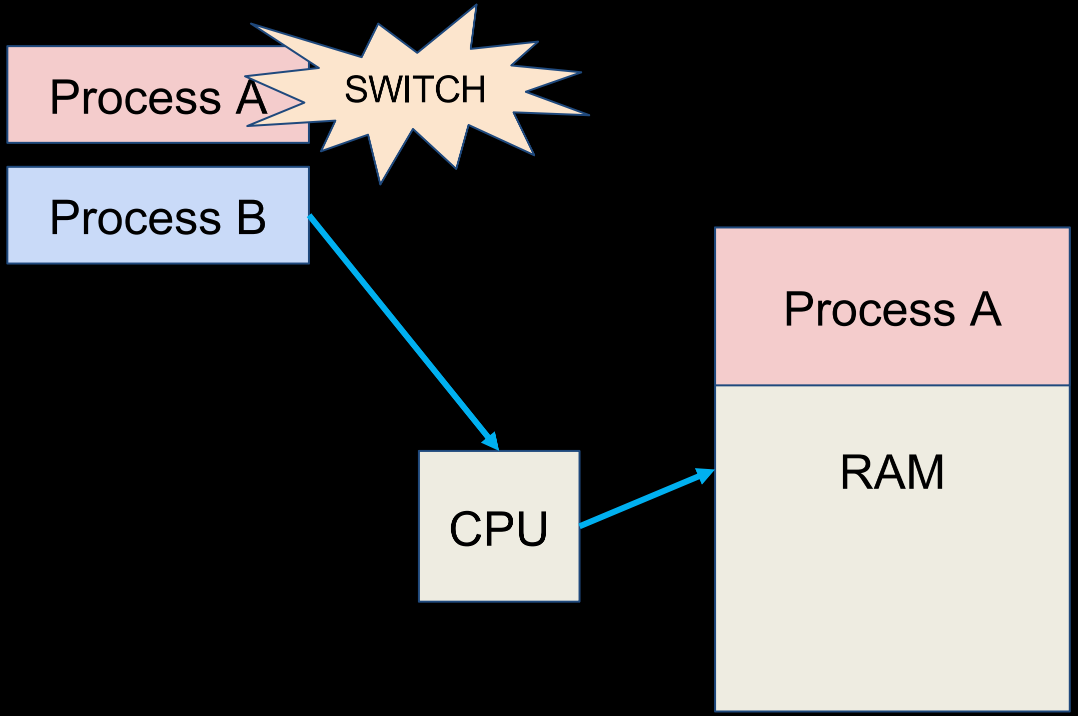
User mode: multiple applications



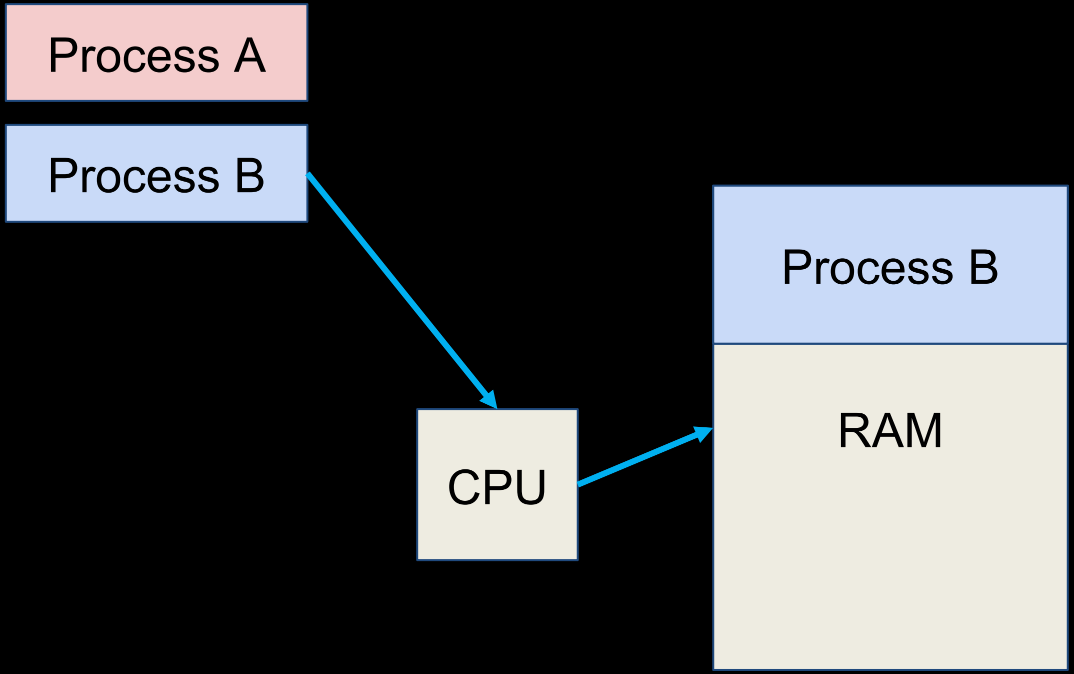
User mode: multiple applications



User mode: multiple applications

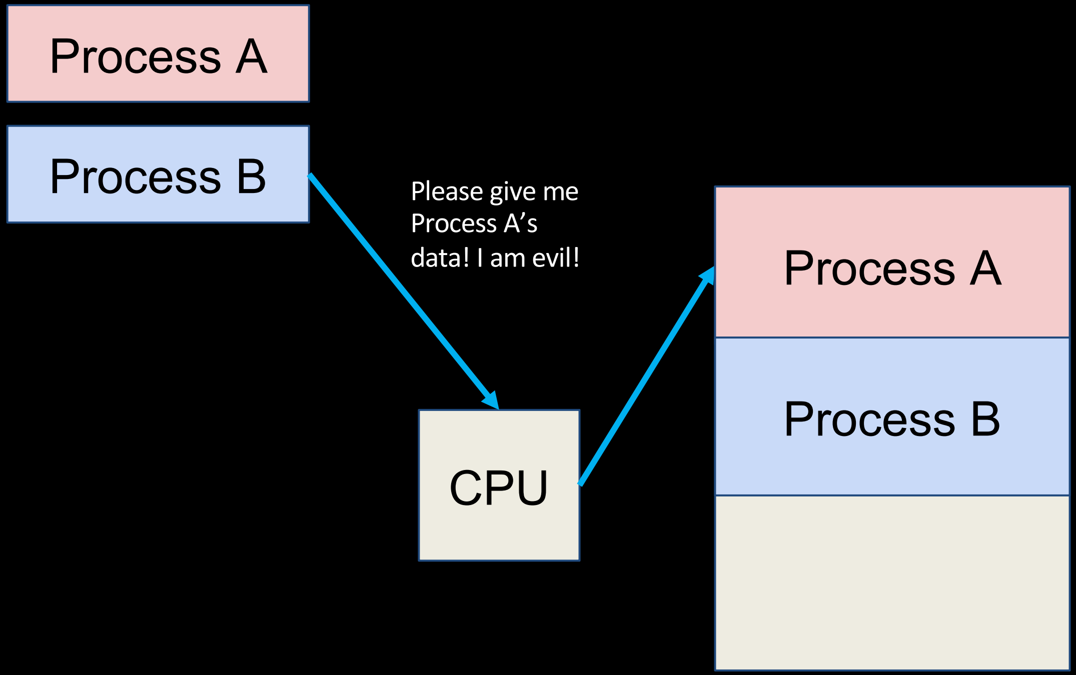


User mode: multiple applications

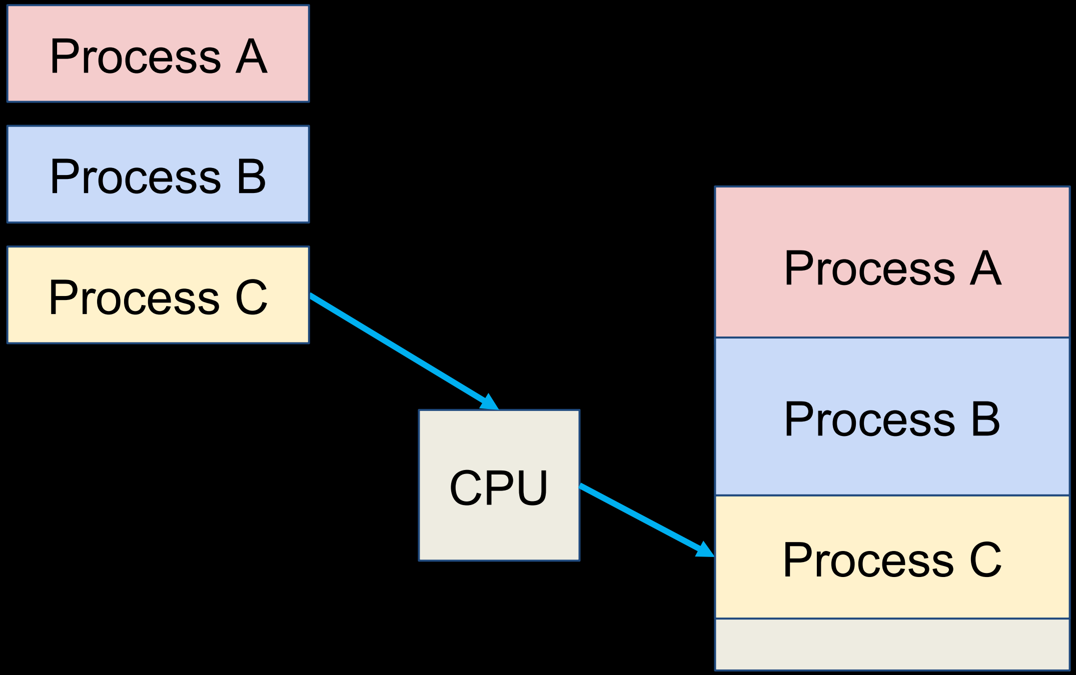


Wait… that shouldn't happen!

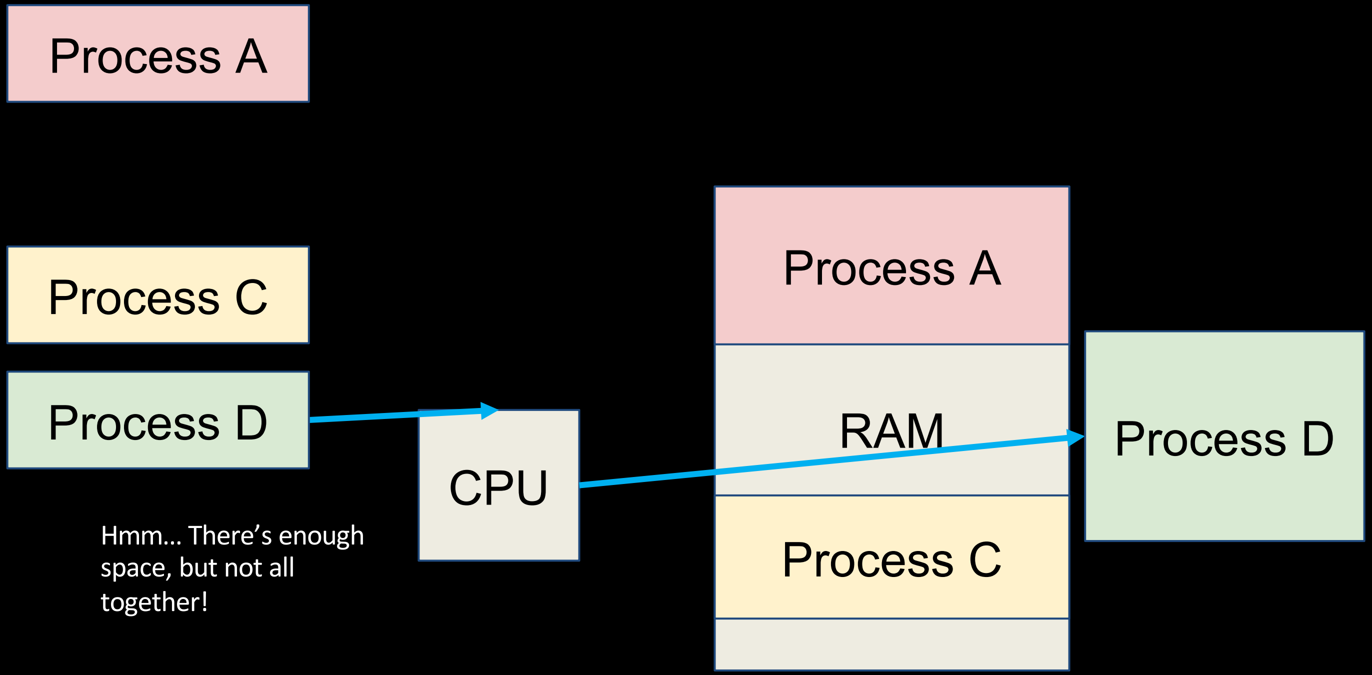
User mode: multiple applications



User mode: multiple applications

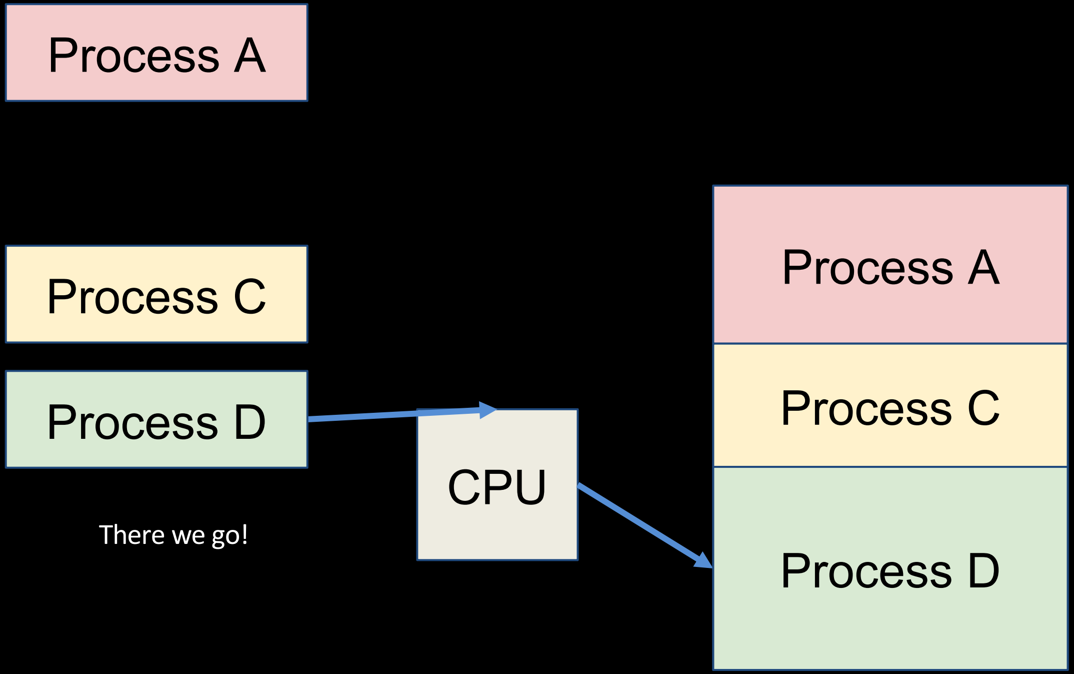


User mode: multiple applications



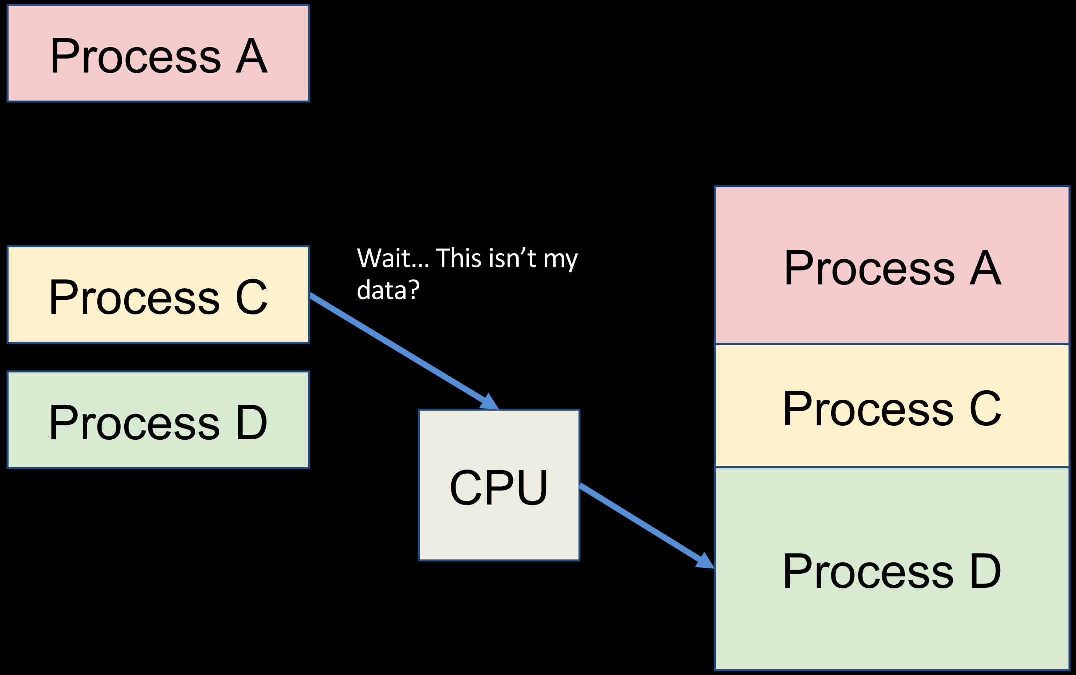
Hmm… There's enough space, but not all together!

User mode: multiple applications



There we go!

User mode: multiple applications



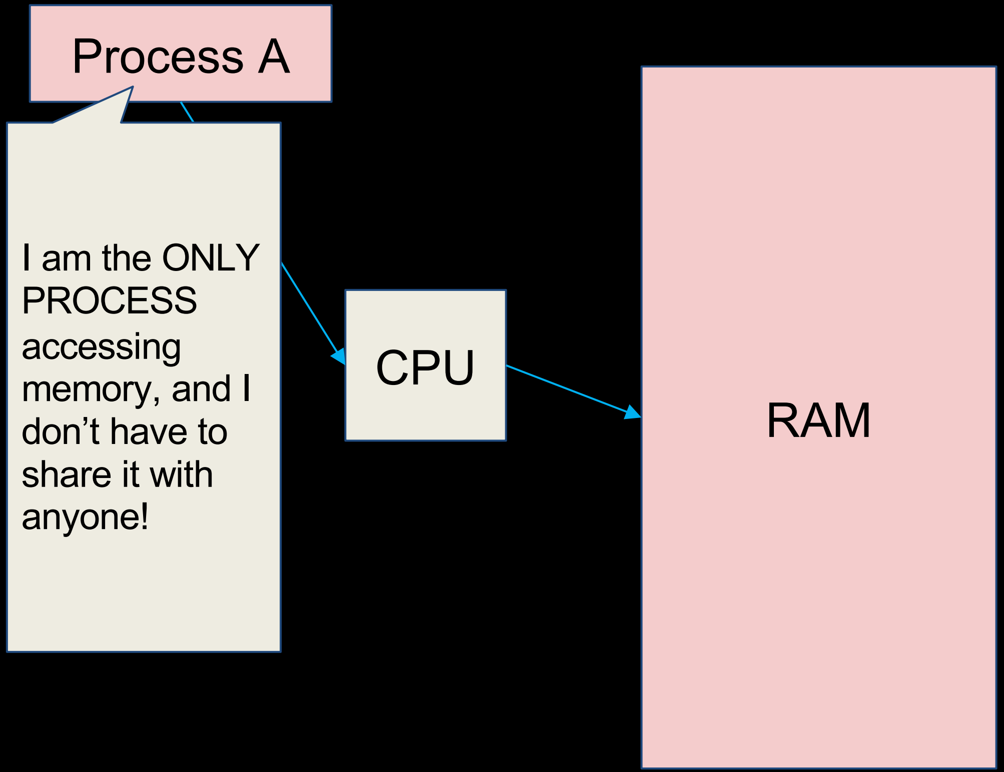
Protection, Translation, Paging

- Supervisor mode does not fully isolate applications from each other or from the OS.
  - Application could overwrite another application's memory.
  - Remember the linker in CALL: application assumes that code is in certain location. How to prevent overlaps?
  - May want to address more memory than we actually have (e.g., for sparse data structures).
- Solution: Virtual Memory. Give each process the illusion of a full memory address space that it has completely to itself.

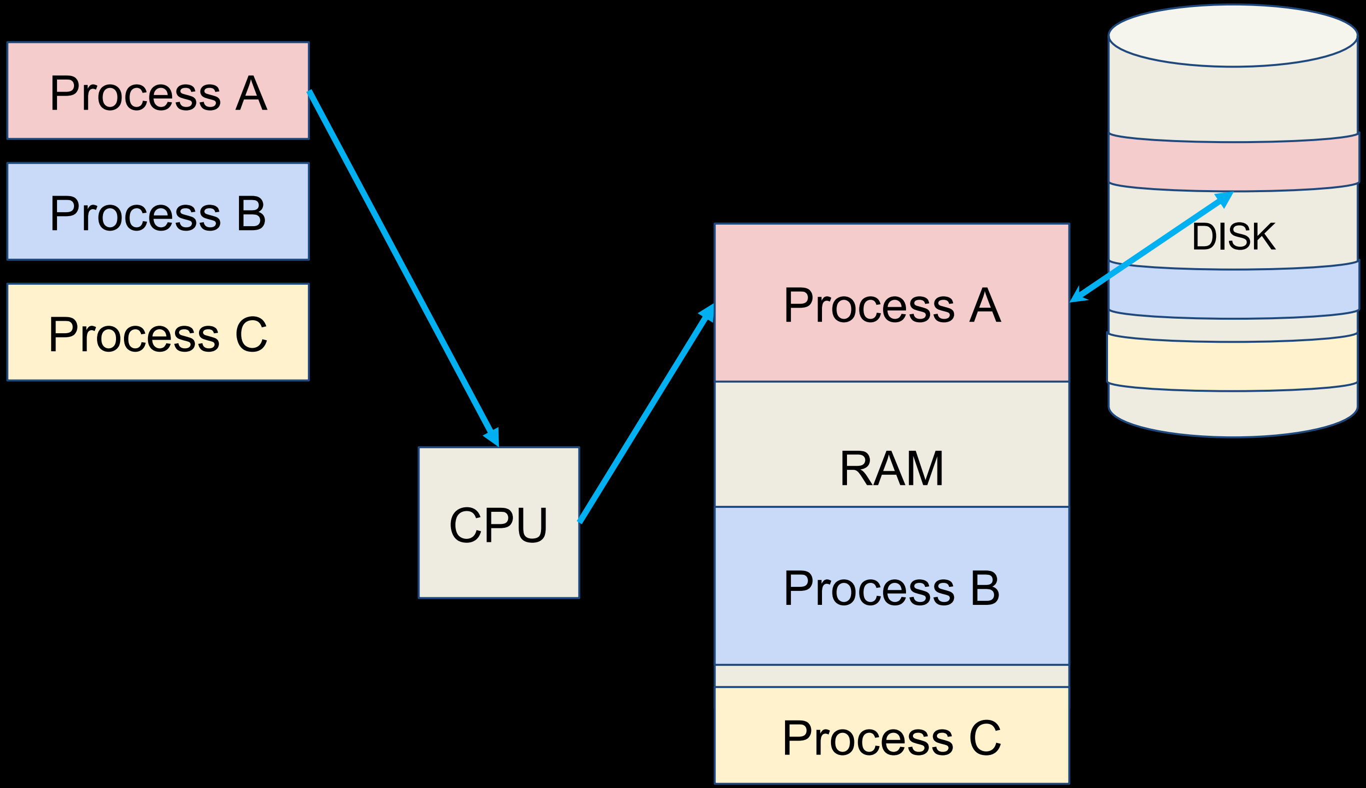
Virtual Memory

- From here on out, we'll be working with two different memory spaces:
  - **Virtual Memory (VM)**: A large (~infinite) space that a process believes it, and only it, has access to
  - **Physical Memory (PM)**: The limited RAM space your computer must share among all processes and processors
- Goals:
  - Process/program isolation
  - Make transition from infinite to finite seamless, or not noticeable to the program
  - Translate between VM, PM addresses

Virtual: The Illusion!

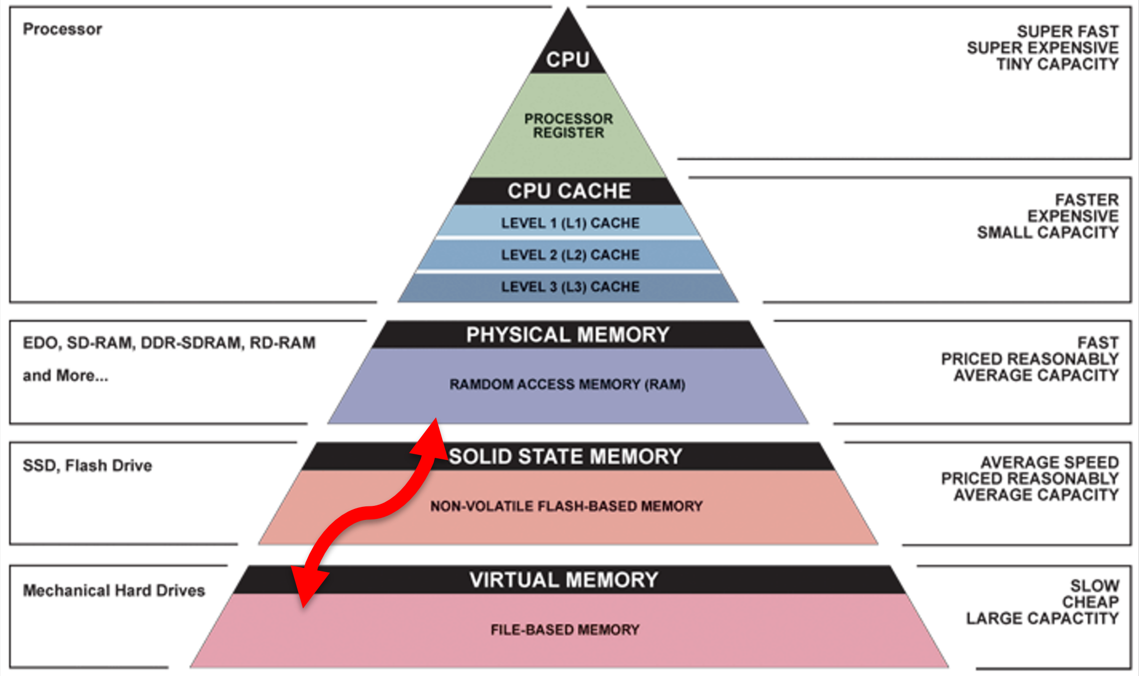


Physical: The Reality!



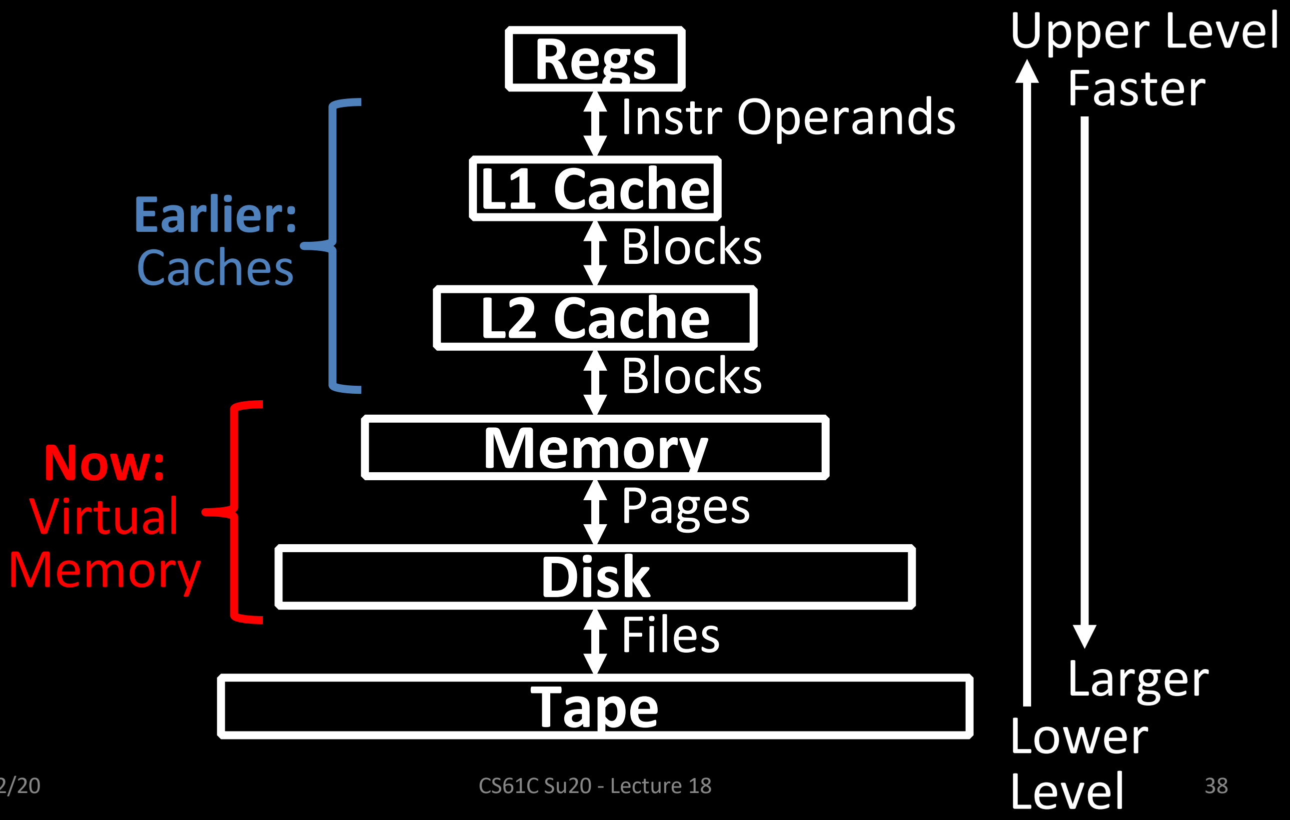
Adding Disks to Hierarchy

- Use VM as a mechanism to “connect” memory and disk in the memory hierarchy



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Memory Hierarchy



Agenda

- OS Intro
- Administrivia
- OS Boot Sequence and Operation
- Multiprogramming/time-sharing
- Introduction to Virtual Memory
- Summary

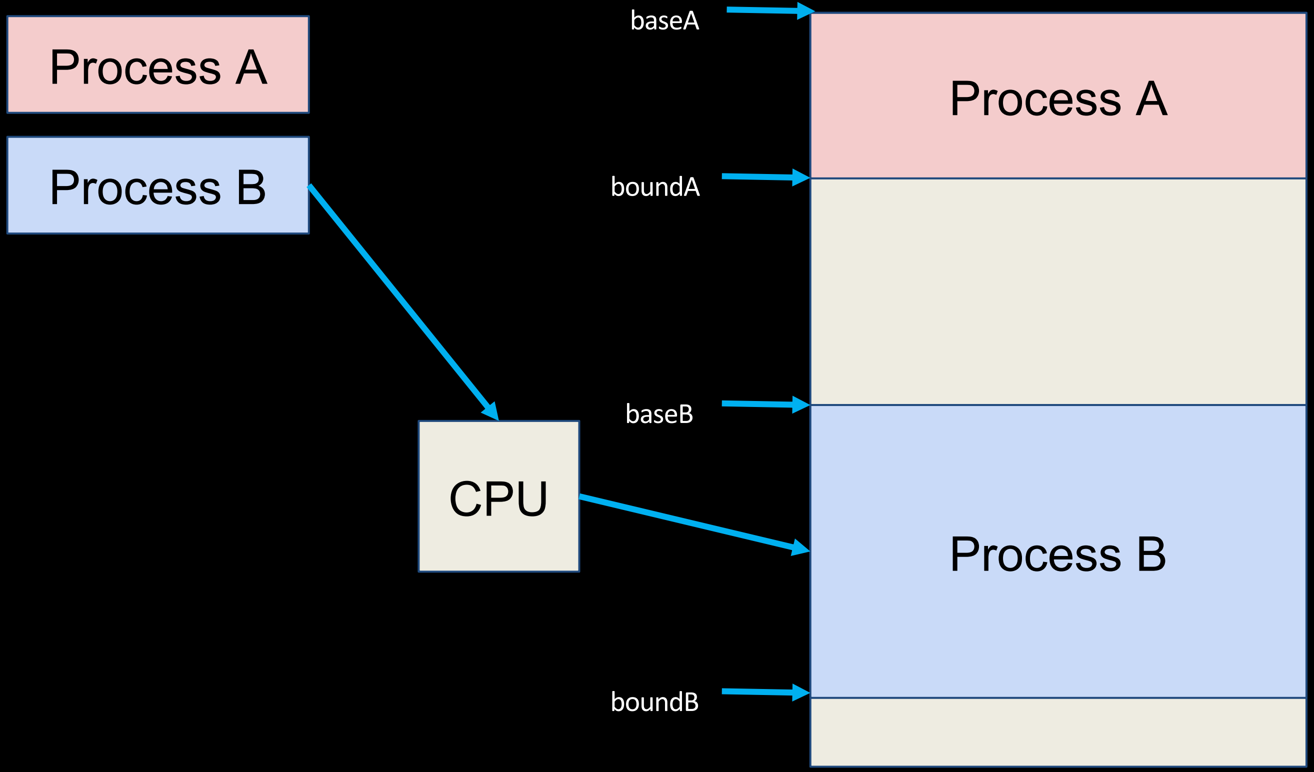
Virtual Memory Goals

- Allow multiple processes to simultaneously occupy memory and provide *protection*: Don't let programs read/write each other's memory
- Give each program the illusion that it has its own private address space
  - Suppose code starts at address 0x00400000, then different processes each think their code resides at that same address!
  - Each program must have a different view of memory

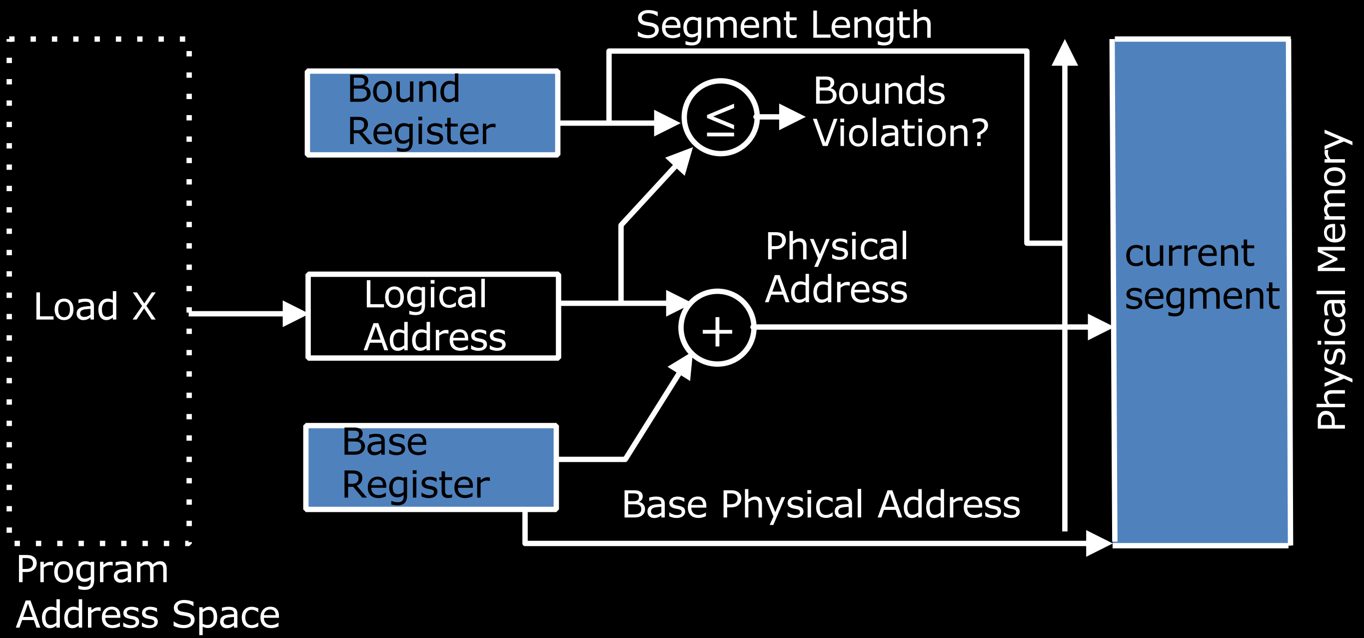
Segmented Memory

- Divide RAM into segments with a “base” and “bound”
  - Each program has access to its segment only!
- Program has a virtual address range 0x0...0 - 0xF...FE
  - To get location of data in segment (physical address), add to base value!

Segmented Memory



Simple Base and Bound Translation

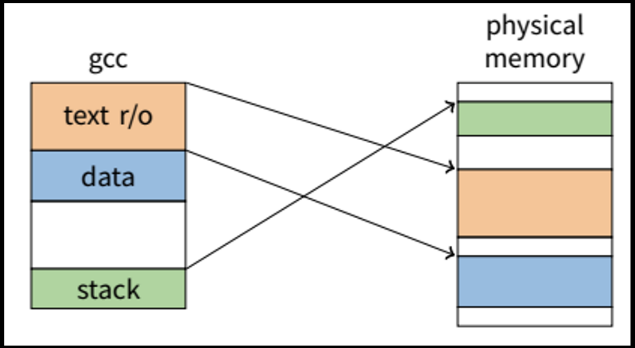


Base and bounds registers are visible/accessible only when processor is running in *supervisor mode*

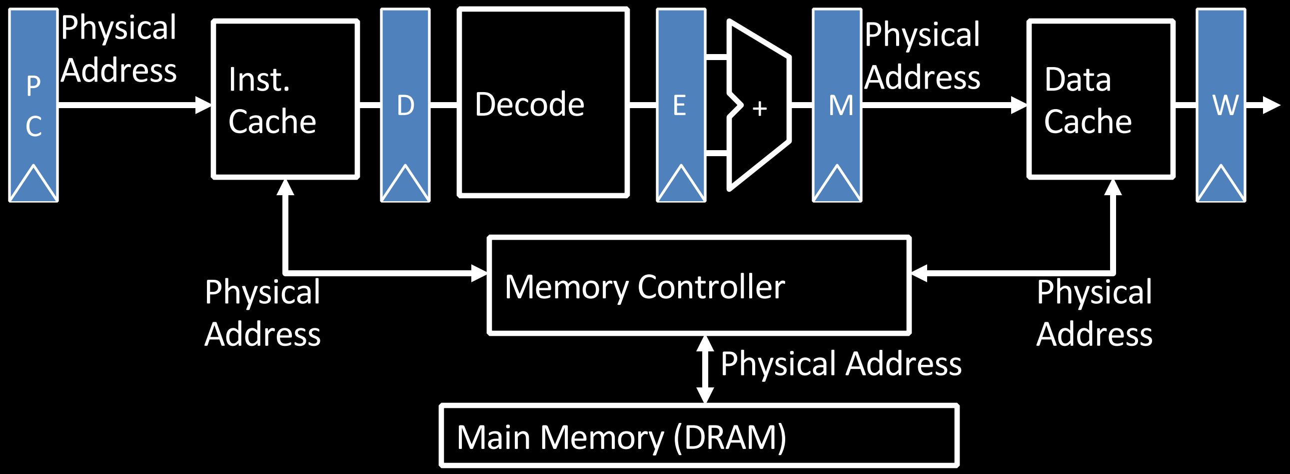
Where have we seen segments before?

Stack, Heap, Static, Code, etc. !

Base & bound model was used to keep segments independent of each other on earlier machines; still used in some memory models (x86) today!

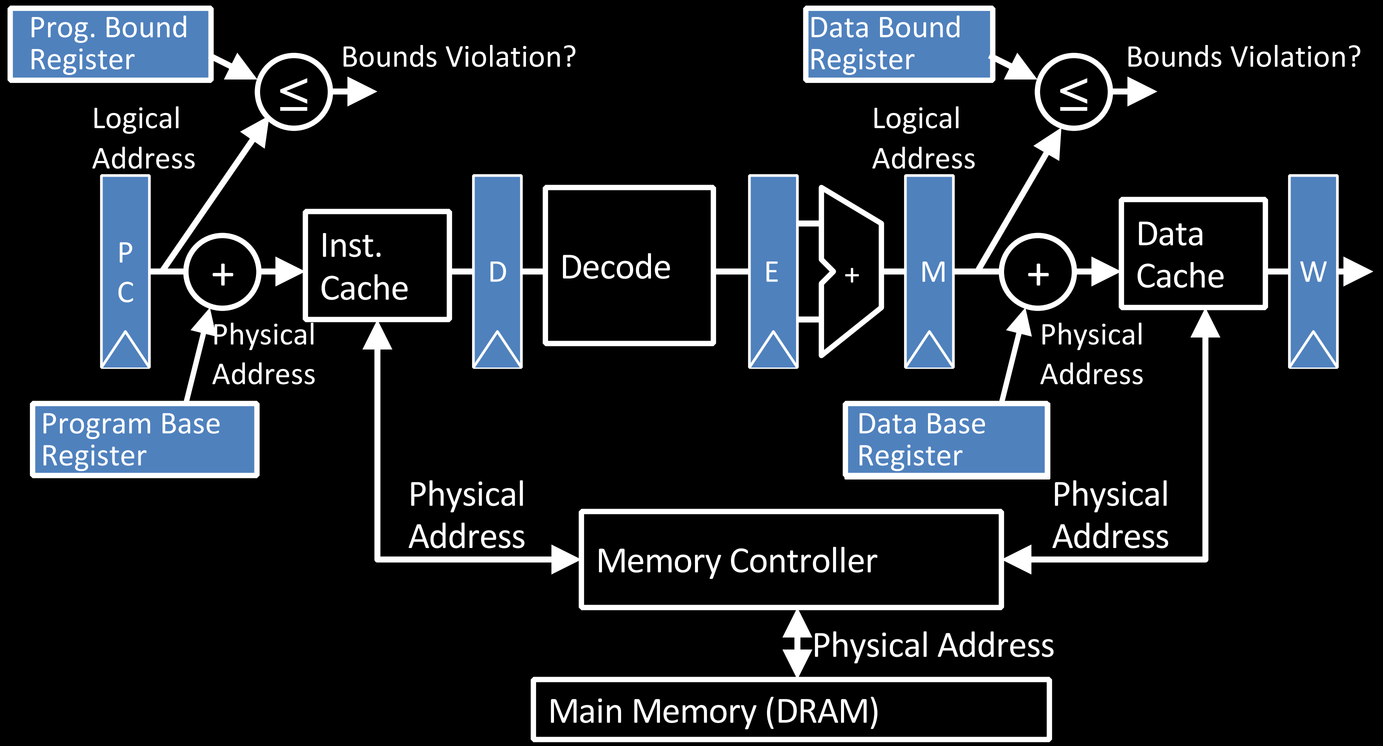


"Bare" 5-Stage Pipeline



- In a bare machine, the only kind of address is a physical address

Base and Bound Machine



Base & Bound: Problems!

- What if we need more space than our segment allows?
  - Increase segment? What if no more RAM?
    - Use disk? How do we decide what data to move in or out? How much data?
- What if we require 500MB of space, but RAM is fragmented, so there isn’t a contiguous chunk available?
  - How can we fix this?

Think to yourself!

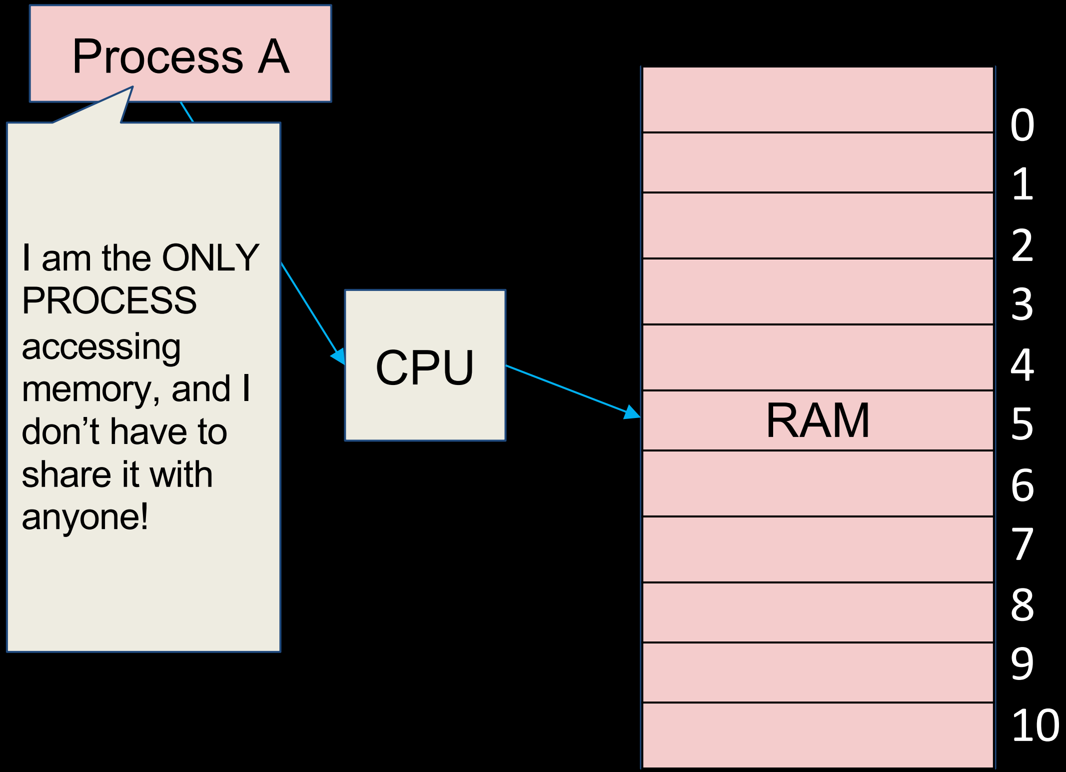
How can we fix our problems with base & bound? is there a better scheme?

- Must provide protection b/t processes
  - Programs can only find/access their own data
- Must be able to work with more data than RAM can hold
  - Swap to and from disk
- Must not fragment memory in an unusable way
  - If need 500MB, and have 500MB available overall, should be usable

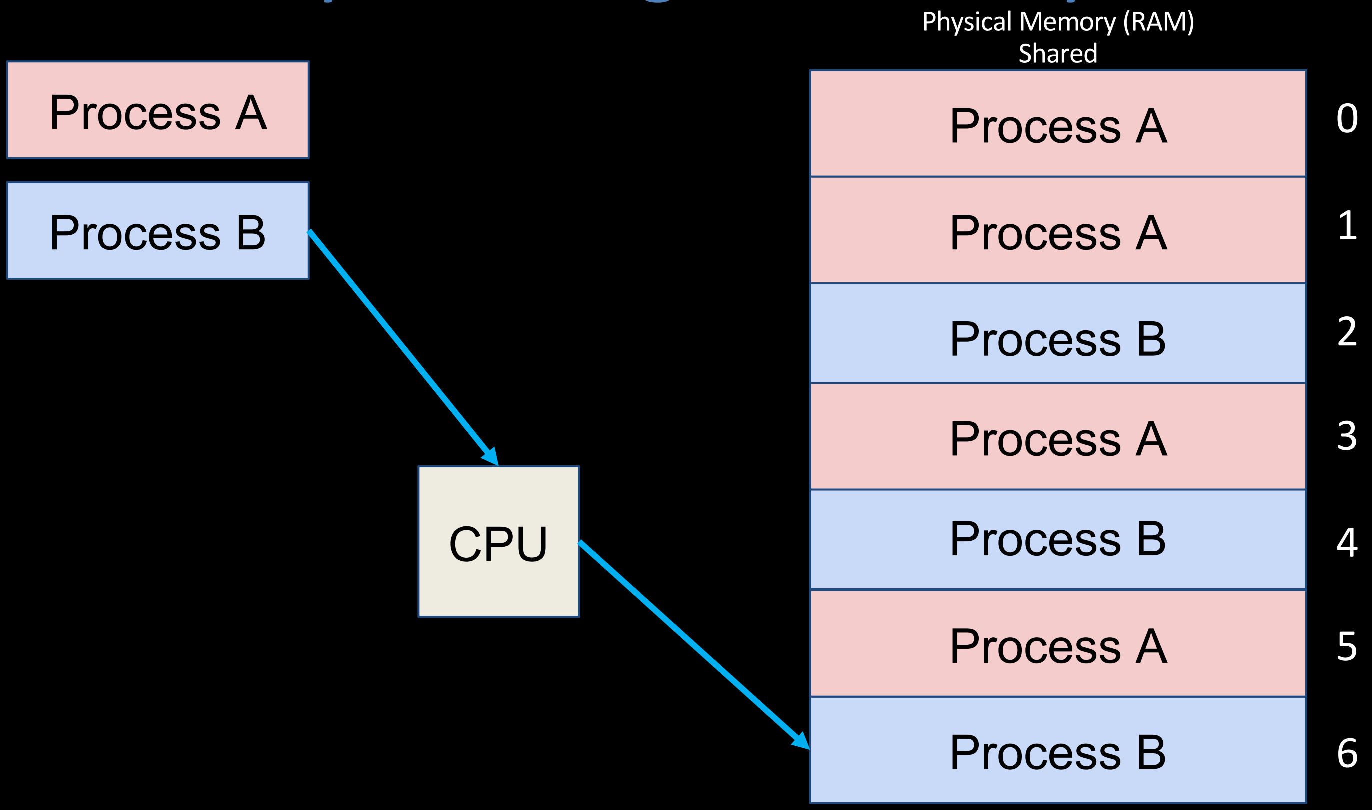
Paged Memory!

- Instead of having segments of various sizes, let’s divide physical memory and virtual memory into equal units called pages!
  - Pages are all the same size, regardless of program, and RAM is an integer multiple of pages.
  - Page size is the same in both virtual and physical memory
- What does our memory layout look like now?

Virtual: The Illusion!



Physical: Paged Memory



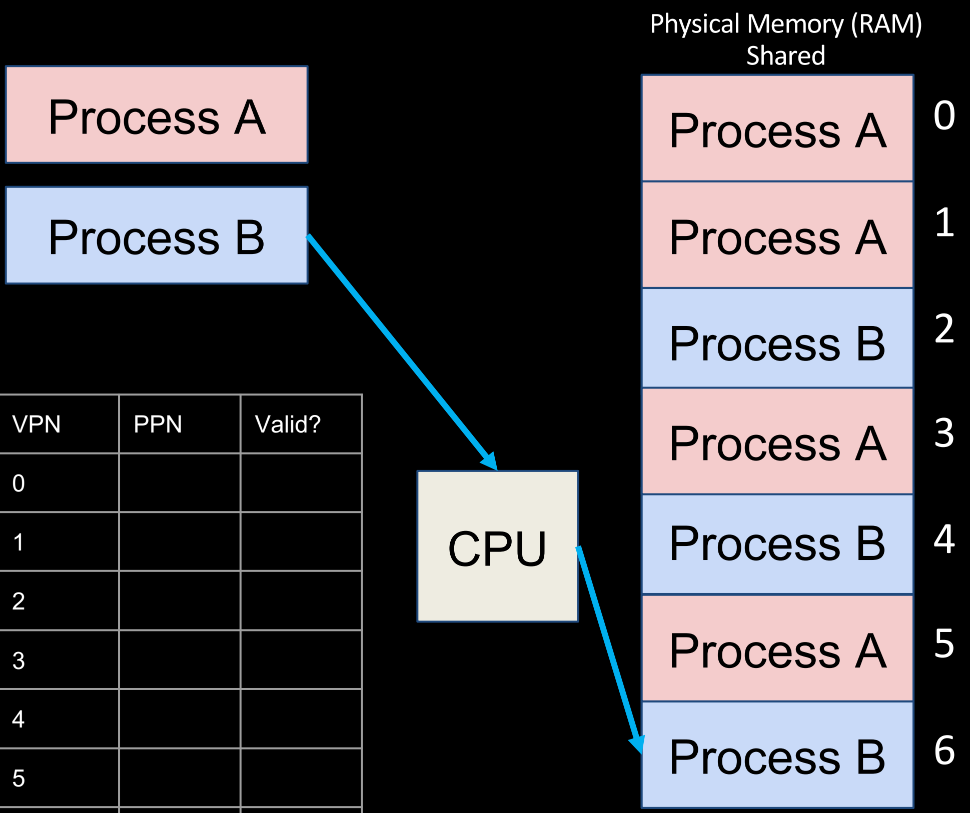
Paged Memory!

- Each program has access to one or more pages
- Pages do not have to be contiguous, or next to each other. Pages are not organised by program

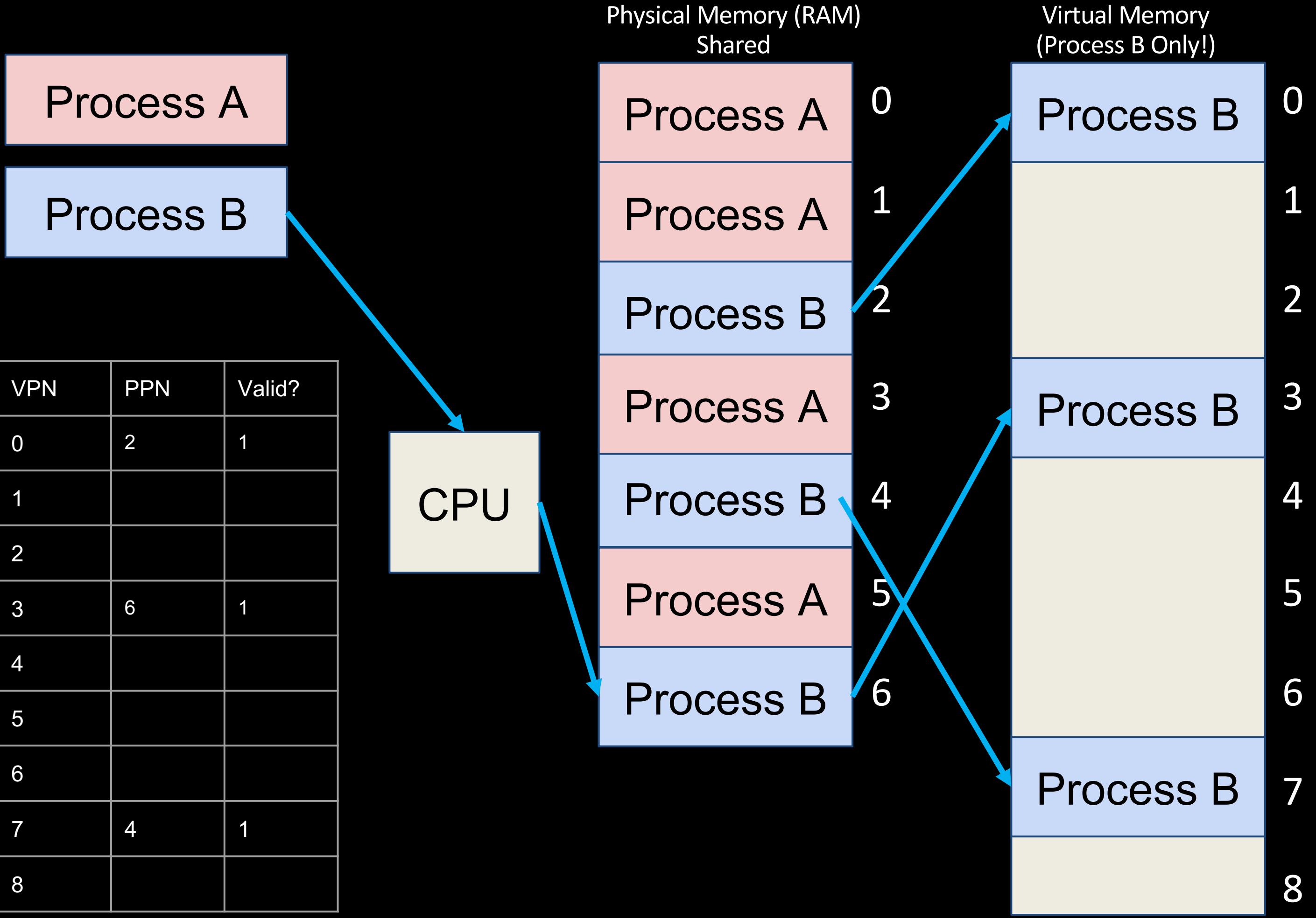
- How do we continue to enforce protection?
- How do we find a piece of data given our virtual address between 0x0..0 and 0xF...FE?

Page protection: Page tables!

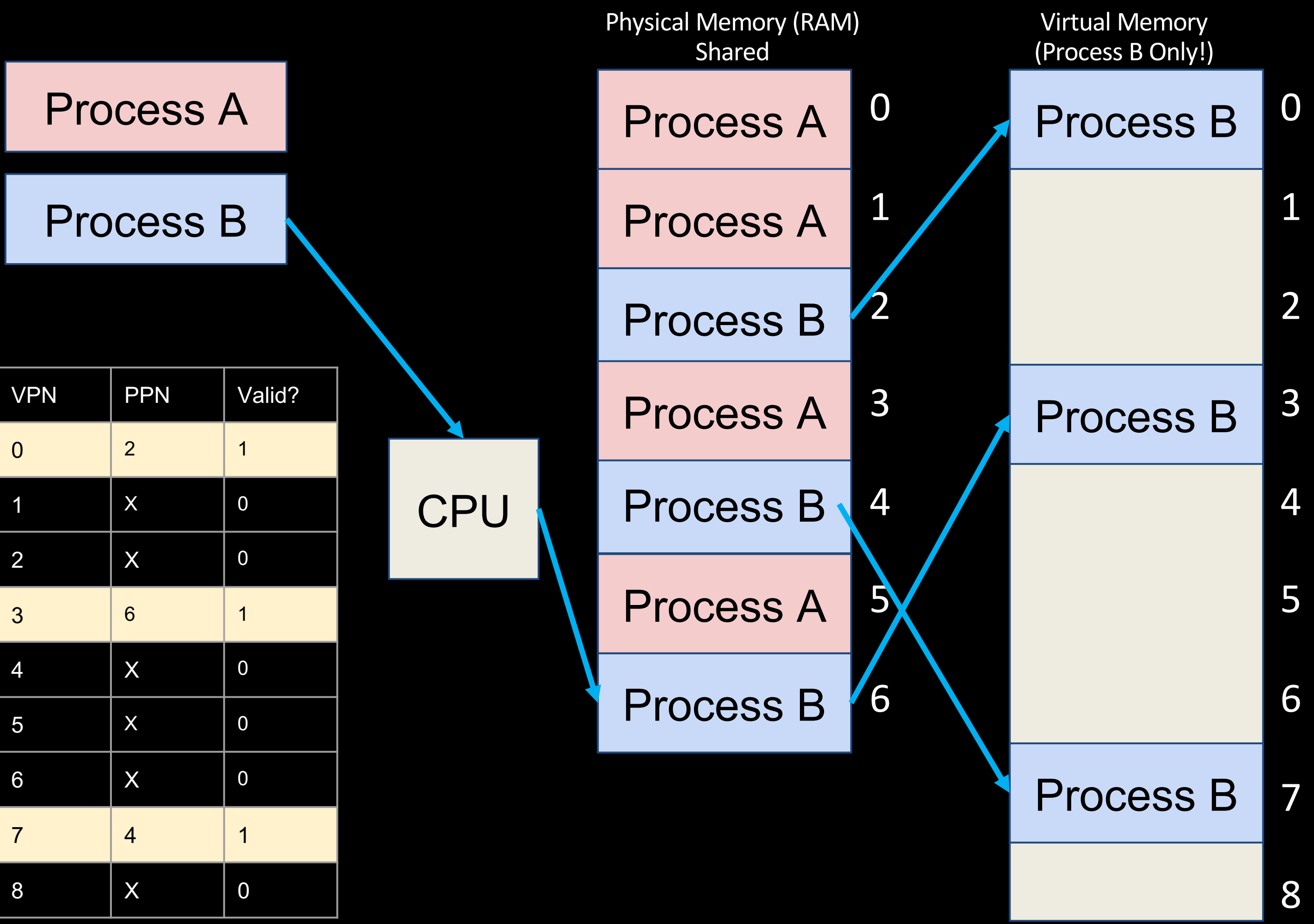
- Per program, maintain a list (or table) of pages in physical memory that they own
  - For each page, note the data it contains
    - Sufficient to just note virtual page number! This will tell us the range of addresses!
- Just like base and bound: use this table to translate addresses so programs cannot reach physical addresses outside of their assigned range!



VPN	PPN	Valid?
0		
1		
2		
3		
4		
5		
6		
7		
8		



VPN	PPN	Valid?
0	2	1
1		
2		
3	6	1
4		
5		
6		
7	4	1
8		



VPN	PPN	Valid?
0	2	1
1	X	0
2	X	0
3	6	1
4	X	0
5	X	0
6	X	0
7	4	1
8	X	0

Modern Virtual Memory Systems

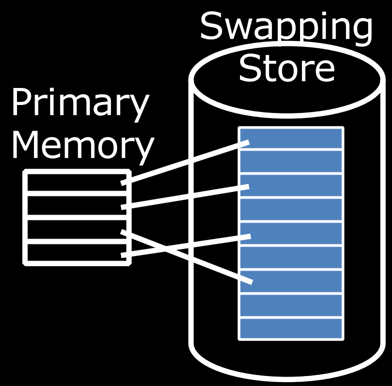
Illusion of a large, private, uniform store

Protection

several programs, each with their private address space and one or more shared address spaces

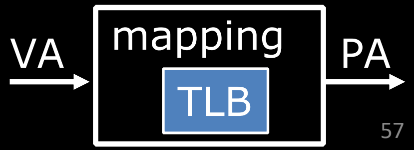
Demand Paging

Provides the ability to run programs larger than the primary memory



Hides differences in machine configurations

The price is address translation on each memory reference



Virtual Memory Goals

- Next level in the memory hierarchy:
  - Provides program with illusion of a very large main memory:
  - Working set of “pages” reside in main memory - others reside on disk.
- Also allows OS to share memory, protect programs from each other
- Today, more important for protection vs. just another level of memory hierarchy
- Each process thinks it has all the memory to itself
- (Historically, it predates caches)

Agenda

- OS Intro
- Administrivia
- OS Boot Sequence and Operation
- Multiprogramming/time-sharing
- Introduction to Virtual Memory
- Summary

Summary

- The role of the Operating System
  - Booting a computer: BIOS, bootloader, OS boot, initialization
- Base and bounds for multiple processes
  - Simple, but doesn't give us everything we want
- Virtual memory bridges memory and disk
  - Provides illusion of independent address spaces to processes and protects them from each other