CS 61C: Great Ideas in Computer Architecture (Machine Structures)
Intro to Virtual Memory

Instructors:

Vladimir Stojanovic and Nicholas Weaver

http://inst.eecs.berkeley.edu/~cs61c/

Agenda

- Multiprogramming/time-sharing
- Introduction to Virtual Memory

Multiprogramming

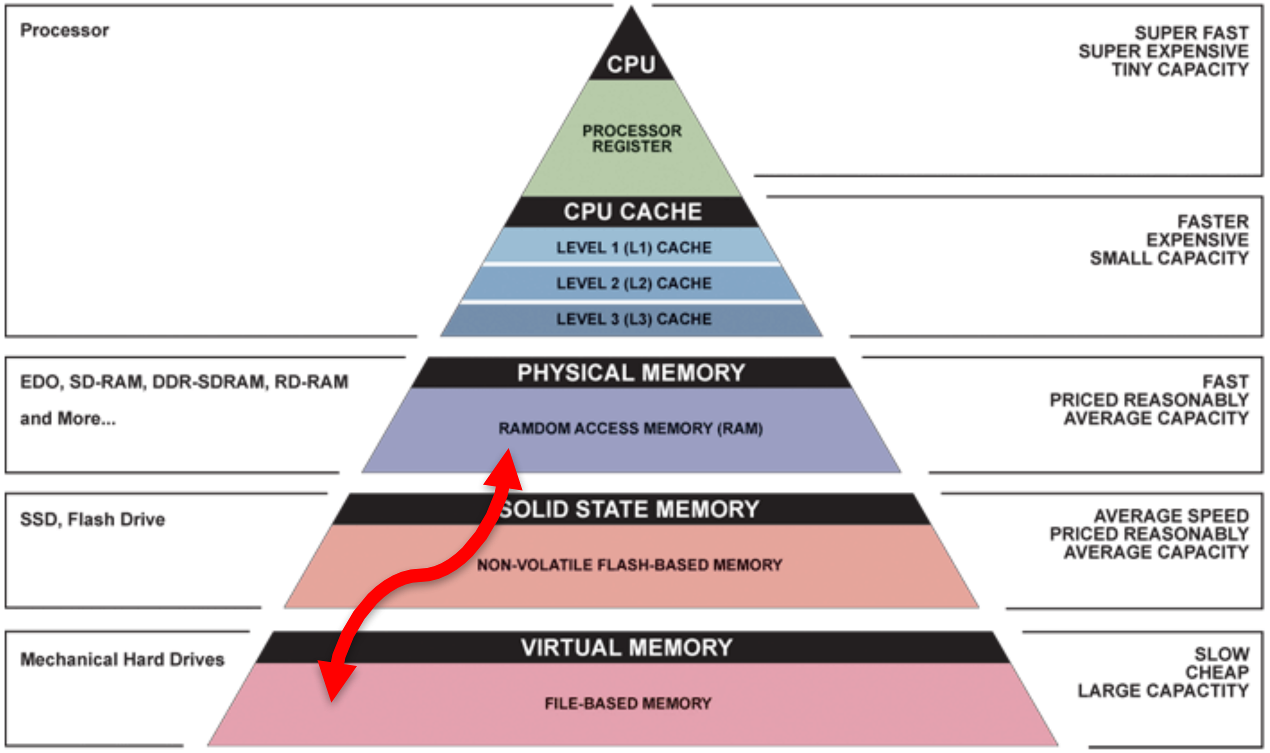
- The OS runs multiple applications at the same time.
  - But not really: have many more processes/threads than available cores
- Switches between processes very quickly. This is called a “context switch”.
- When jumping into process, set timer interrupt.
  - When it expires, store PC, registers, etc. (process state).
  - Pick a different process to run and load its state.
  - Set timer, change to user mode, jump to the new PC.
- Deciding what process to run is called scheduling.

Protection, Translation, Paging

- Supervisor mode does things that normal mode can't...
  - But...
- Supervisor mode is not enough to fully isolate applications from each other or from the OS.
  - Application could overwrite another application's memory.
  - Also, may want to address more memory than we actually have (e.g., for sparse data structures).
- Solution: Virtual Memory. Gives each process the illusion of a full memory address space that it has completely for itself.

What do we need Virtual Memory for? Reason 1: Adding Disks to Hierarchy

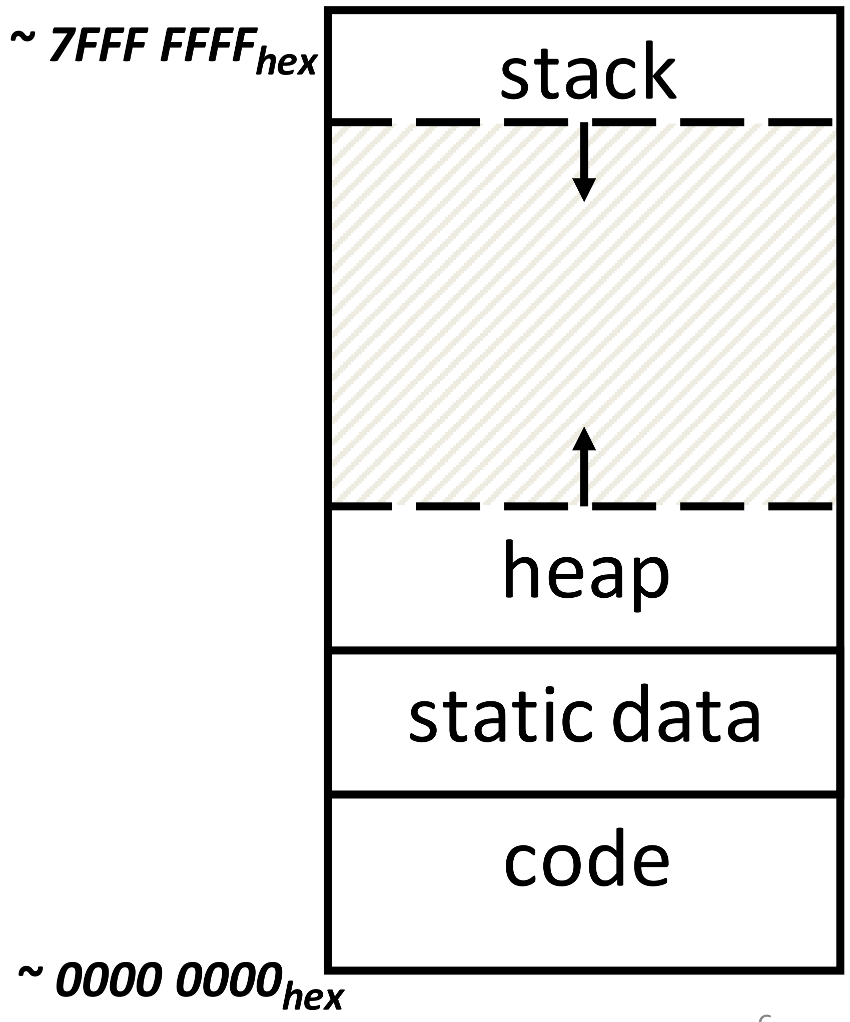
- Need to devise a mechanism to “connect” memory and disk in the memory hierarchy



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

What do we need Virtual Memory for? Reason 2: Simplifying Memory for Apps

- Applications should see the straightforward memory layout we saw earlier ->
- User-space applications should think they own all of memory
- So we give them a **virtual** view of memory



What do we need Virtual Memory for? Reason 3: Protection Between Processes

- With a bare system, addresses issued with loads/stores are real **physical** addresses
- This means any program can issue any address, therefore can access any part of memory, even areas which it doesn't own
  - Ex: The OS data structures
- We should send all addresses through a mechanism that the OS controls, before they make it out to DRAM - **a translation mechanism**

VM + Supervisor Mode combine to Create Isolation

- Supervisor mode is ***only*** entered into at the trap handler
  - So its always known (and hopefully correct) code that is part of the core operating system
    - This is why "syscall" generates an exception
- Only Supervisor mode can ***change*** Virtual Memory mappings
  - So only the core of the operating system can bypass the protections imposed on memory
- These are the invariants necessary for isolation
  - Anything that can affect these invariants ***completely*** compromises the security of the system

Address Spaces

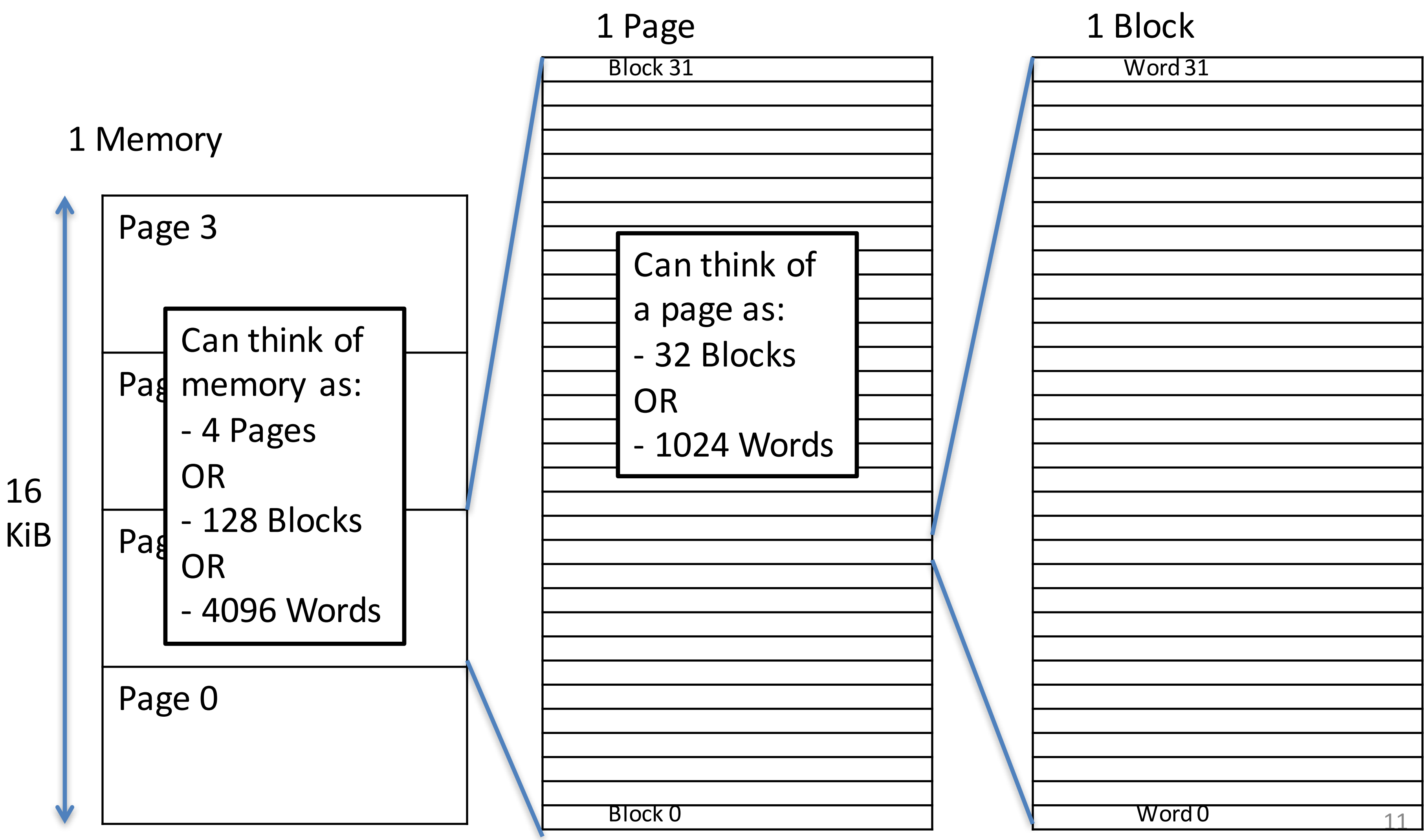
- The set of addresses labeling all of memory that we can access
- Now, 2 kinds:
  - **Virtual Address Space** - the set of addresses that the user program knows about
  - **Physical Address Space** - the set of addresses that map to actual physical cells in memory
    - Hidden from user applications
- So, we need a way to map between these two address spaces

Blocks vs. Pages

- In caches, we dealt with individual *blocks*
  - Usually ~64B on modern systems
  - We could “divide” memory into a set of blocks
- In VM, we deal with individual *pages*
  - Usually ~4 KB on modern systems
  - Now, we’ll “divide” memory into a set of pages
- Common point of confusion: Bytes, Words, Blocks, Pages are all just different ways of looking at memory!

Bytes, Words, Blocks, Pages

Ex: 16 KiB DRAM, 4 KiB Pages (for VM), 128 B blocks (for caches), 4 B words (for lw/sw)

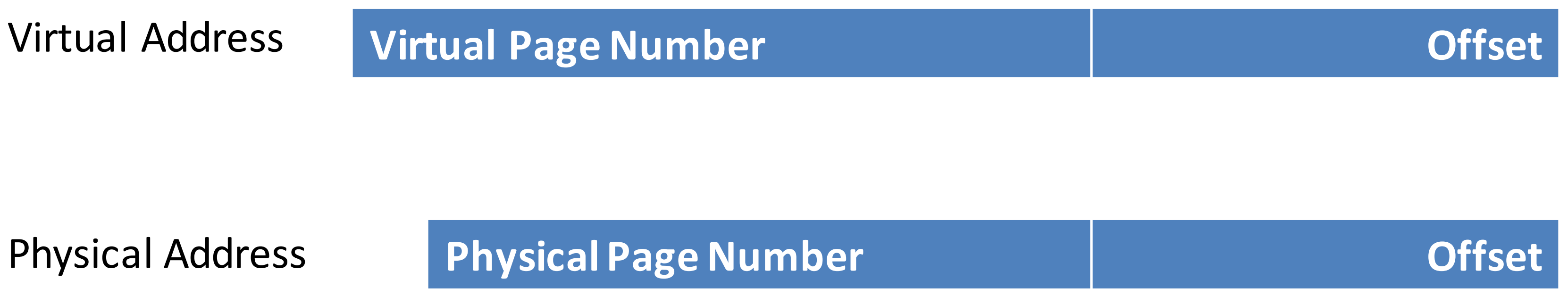


Can think of memory as:
- 4 Pages
OR
- 128 Blocks
OR
- 4096 Words

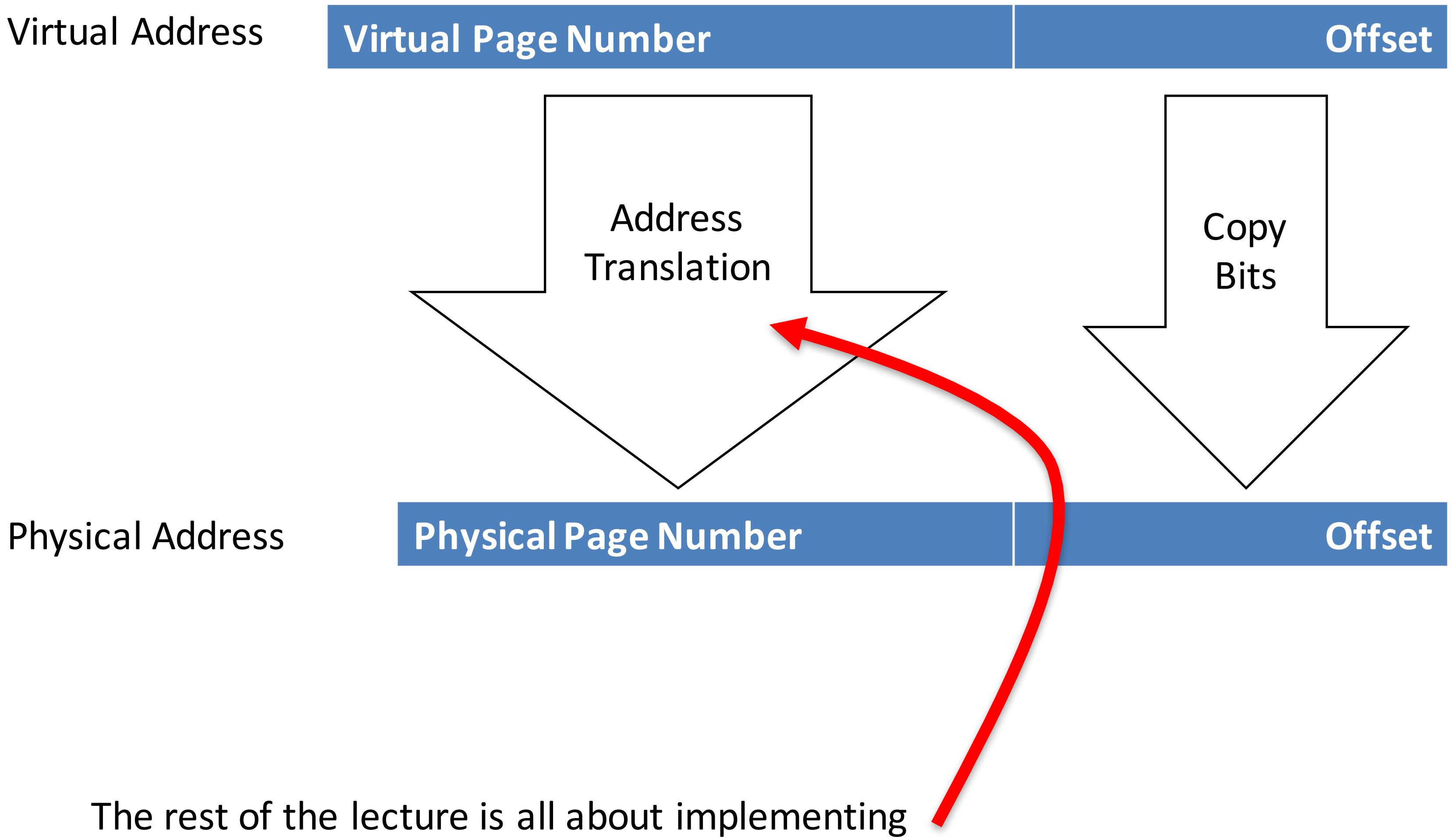
Can think of a page as:
- 32 Blocks
OR
- 1024 Words

Address Translation

- So, what do we want to achieve at the hardware level?
  - Take a Virtual Address, that points to a spot in the Virtual Address Space of a particular program, and map it to a Physical Address, which points to a physical spot in DRAM of the whole machine

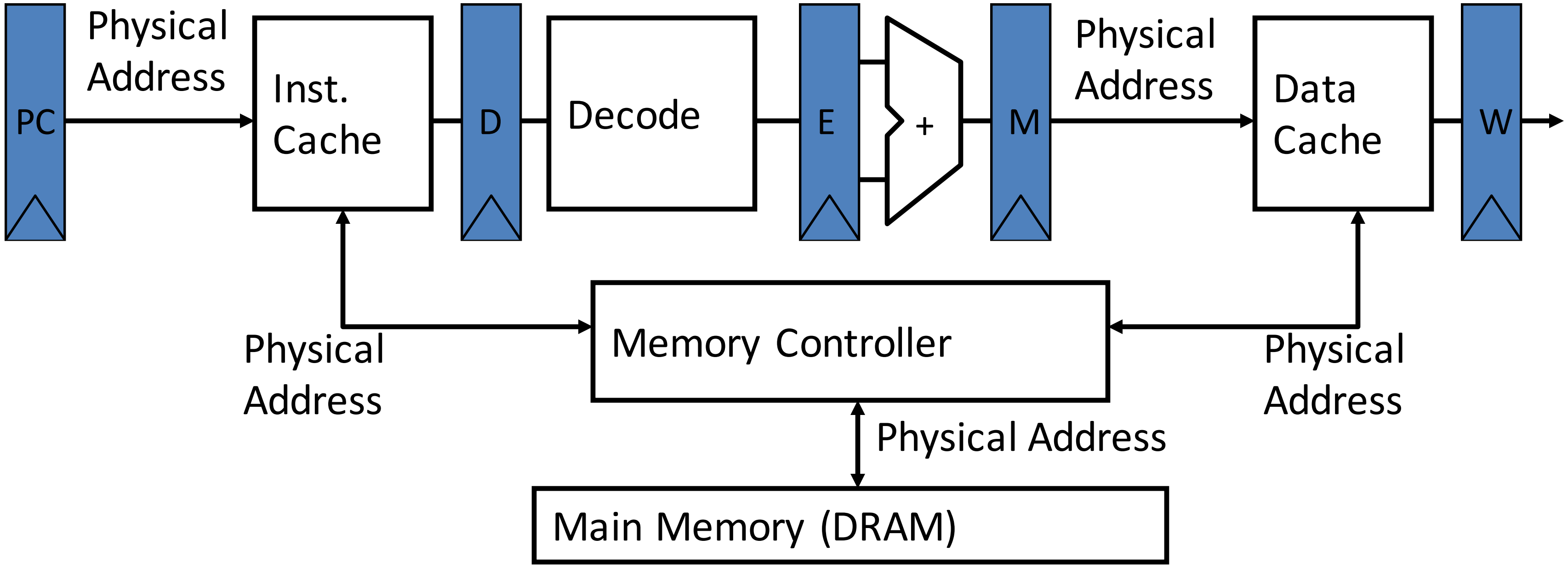


Address Translation



The rest of the lecture is all about implementing

“Bare” 5-Stage Pipeline



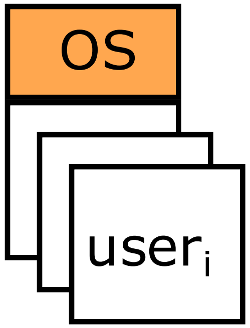
- In a bare machine, the only kind of address is a physical address

Modern *Virtual Memory* Systems

Illusion of a large, private, uniform store

Protection

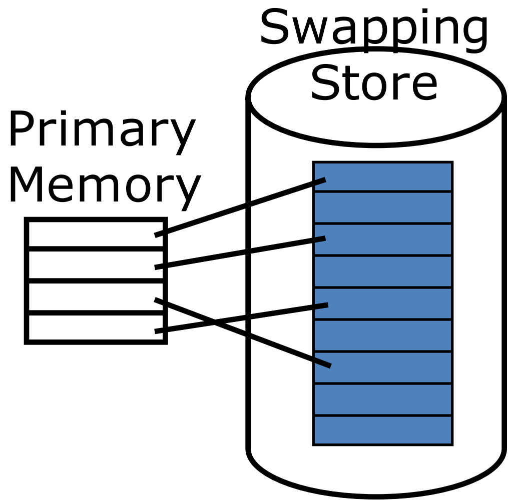
- several users, each with their private address space and one or more shared address spaces



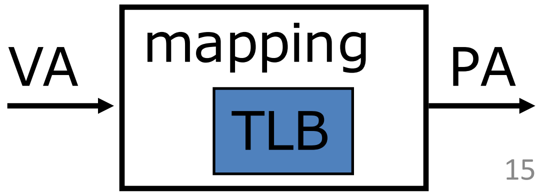
Demand Paging

- 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



Dynamic Address Translation

Motivation

Multiprogramming, multitasking: Desire to execute more than one process at a time (more than one process can reside in main memory at the same time).

Location-independent programs

Programming and storage management ease

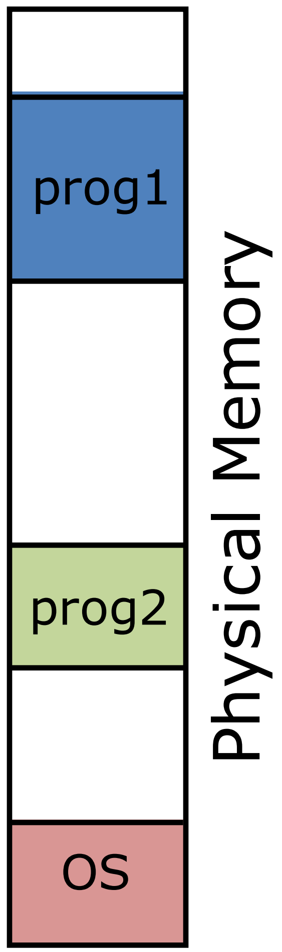
- *base register – add offset to each address*

Protection

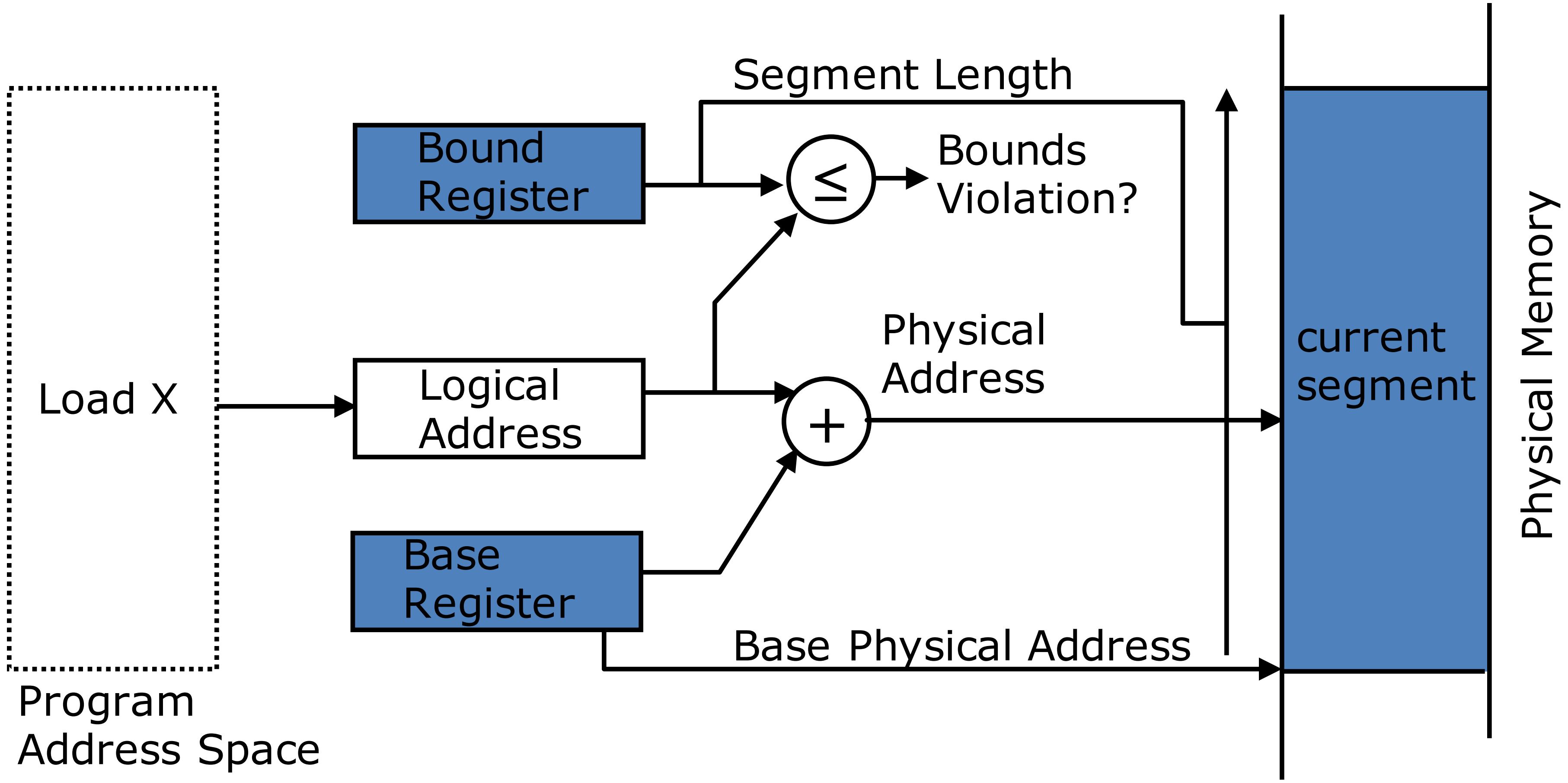
Independent programs should not affect each other inadvertently

- *bound register – check range of access*

(Note: Multiprogramming drives requirement for resident *supervisor (OS)* software to manage context switches between multiple programs)



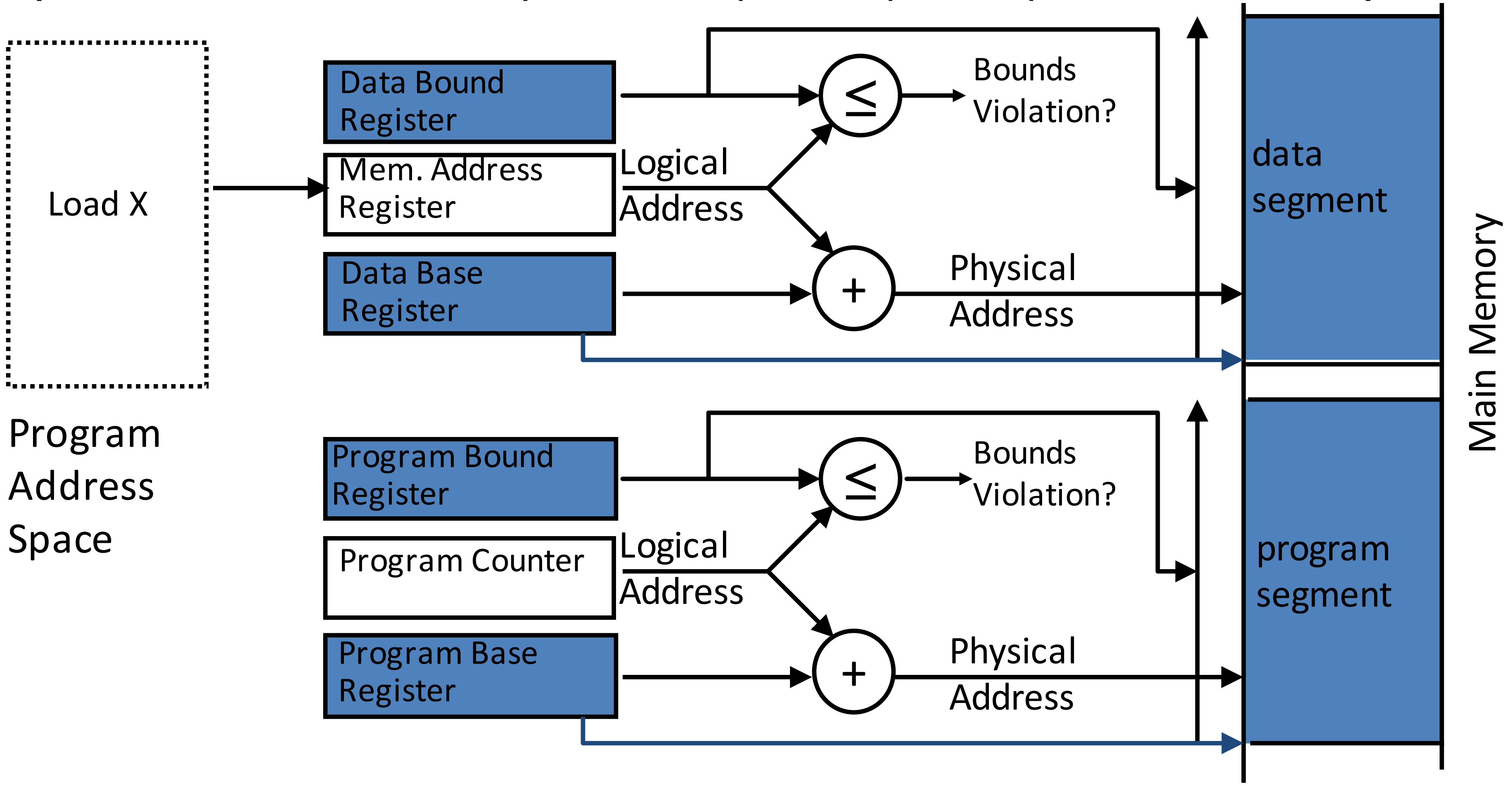
Simple Base and Bound Translation



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

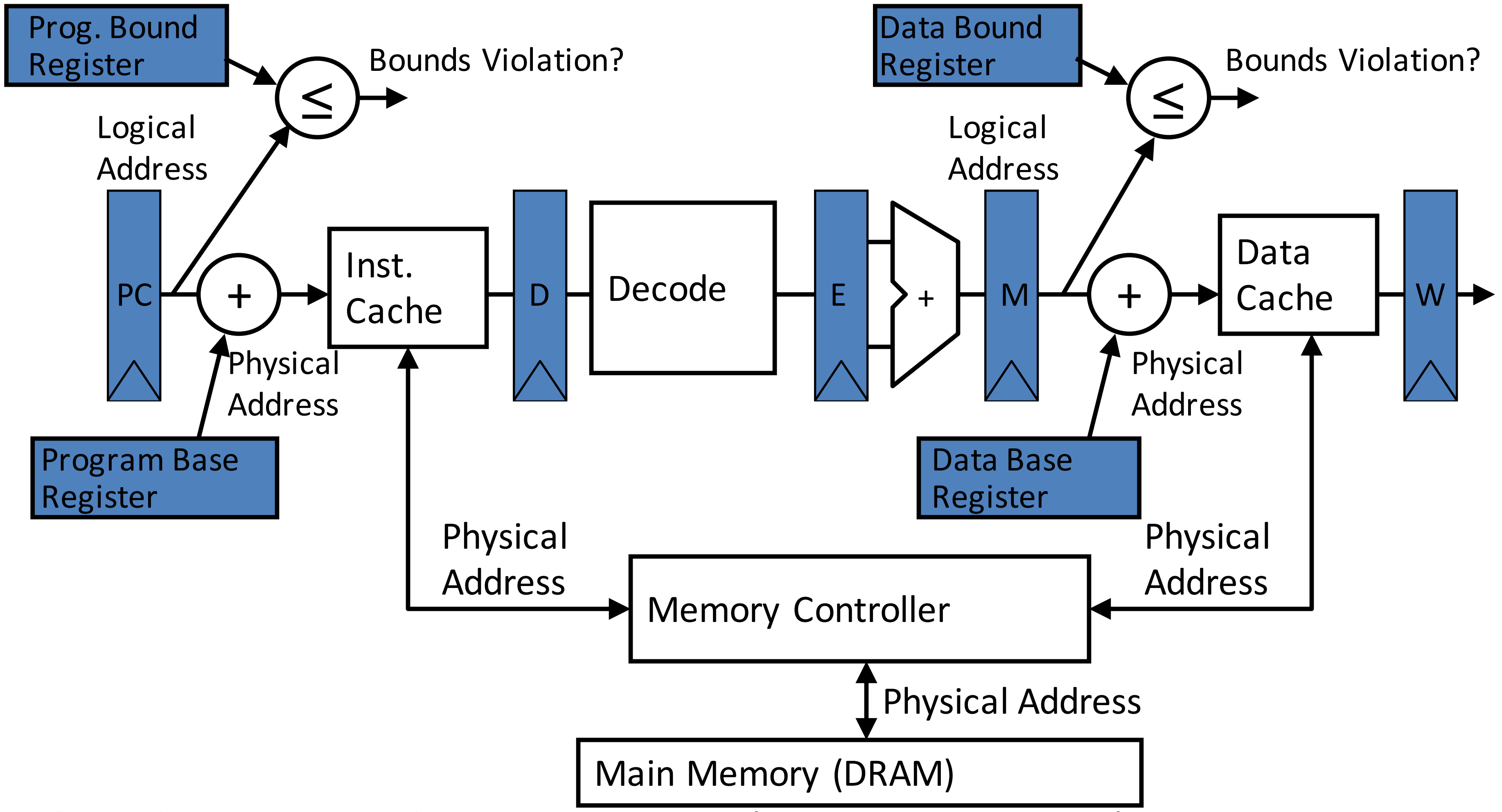
Separate Areas for Program and Data

(Scheme used on all Cray vector supercomputers prior to X1, 2002)



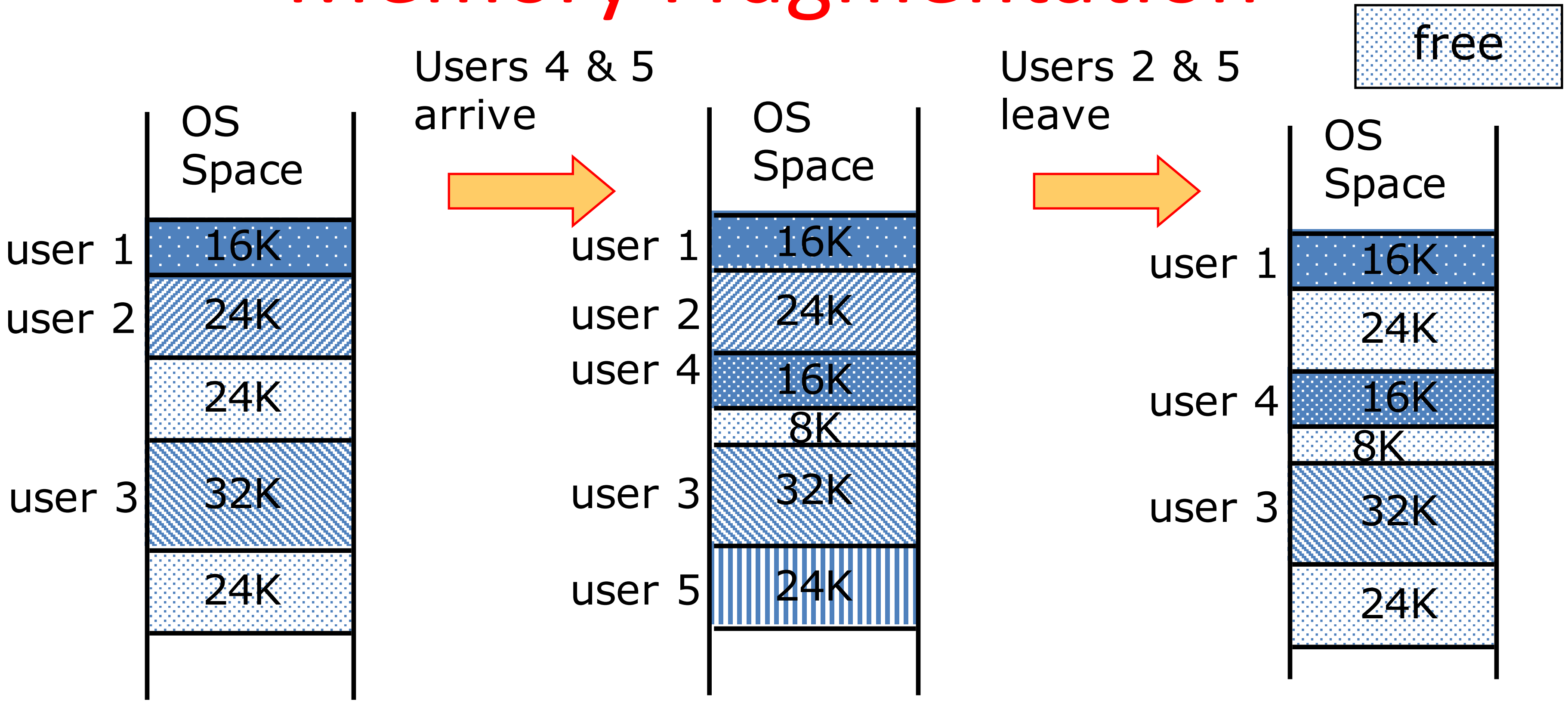
What is an advantage of this separation?

Base and Bound Machine



[Can fold addition of base register into (register+immediate) address calculation using a carry-save adder (sums three numbers with only a few gate delays more than adding two numbers)]

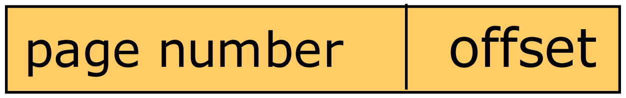
Memory Fragmentation



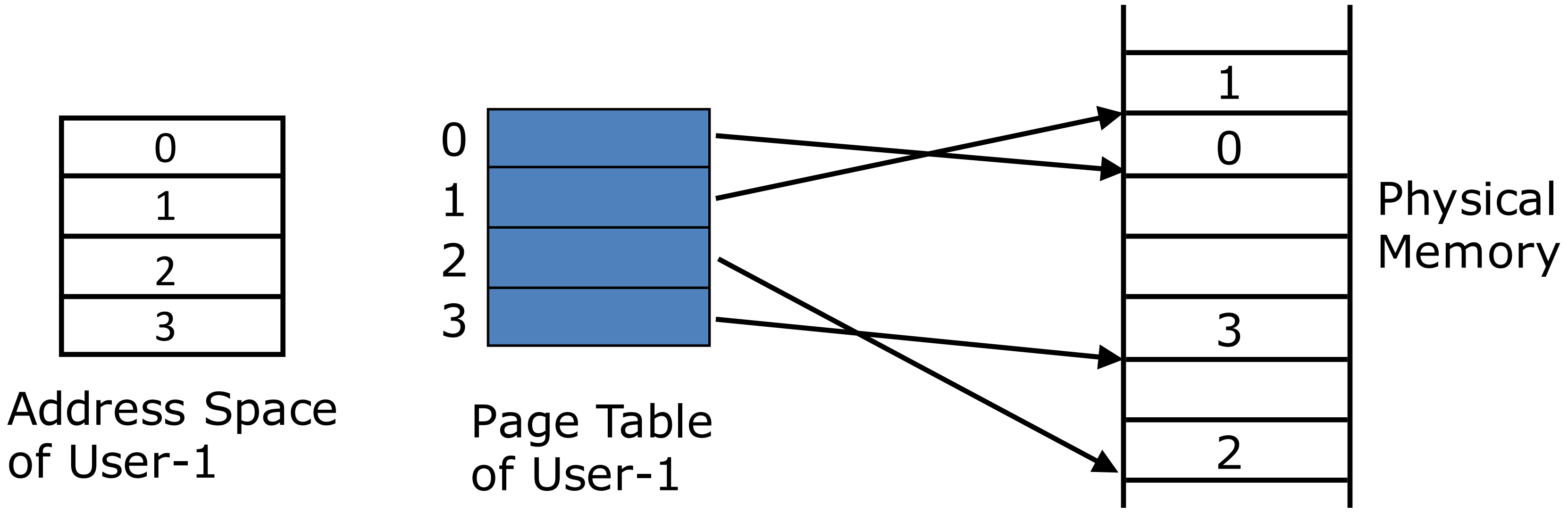
As users come and go, the storage is "fragmented". Therefore, at some stage programs have to be moved around to compact the storage.

Paged Memory Systems

- Processor-generated address can be split into:

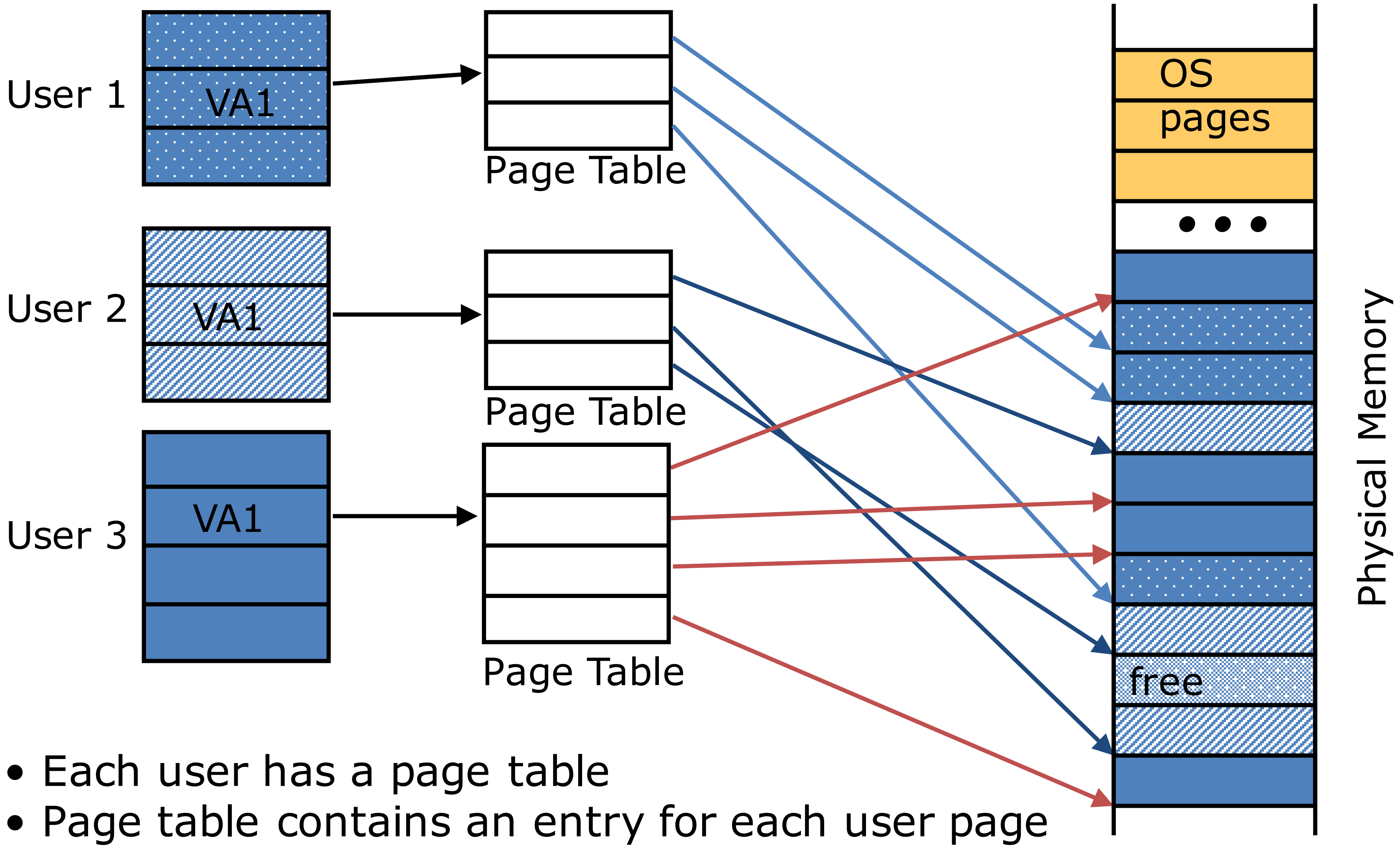


- A page table contains the physical address of the base of each page



Page tables make it possible to store the pages of a program non-contiguously.

Private Address Space per User



- Each user has a page table
- Page table contains an entry for each user page

Where Should Page Tables Reside?

- Space required by the page tables (PT) is proportional to the address space, number of users, ...
    - *Too large to keep in cpu registers*

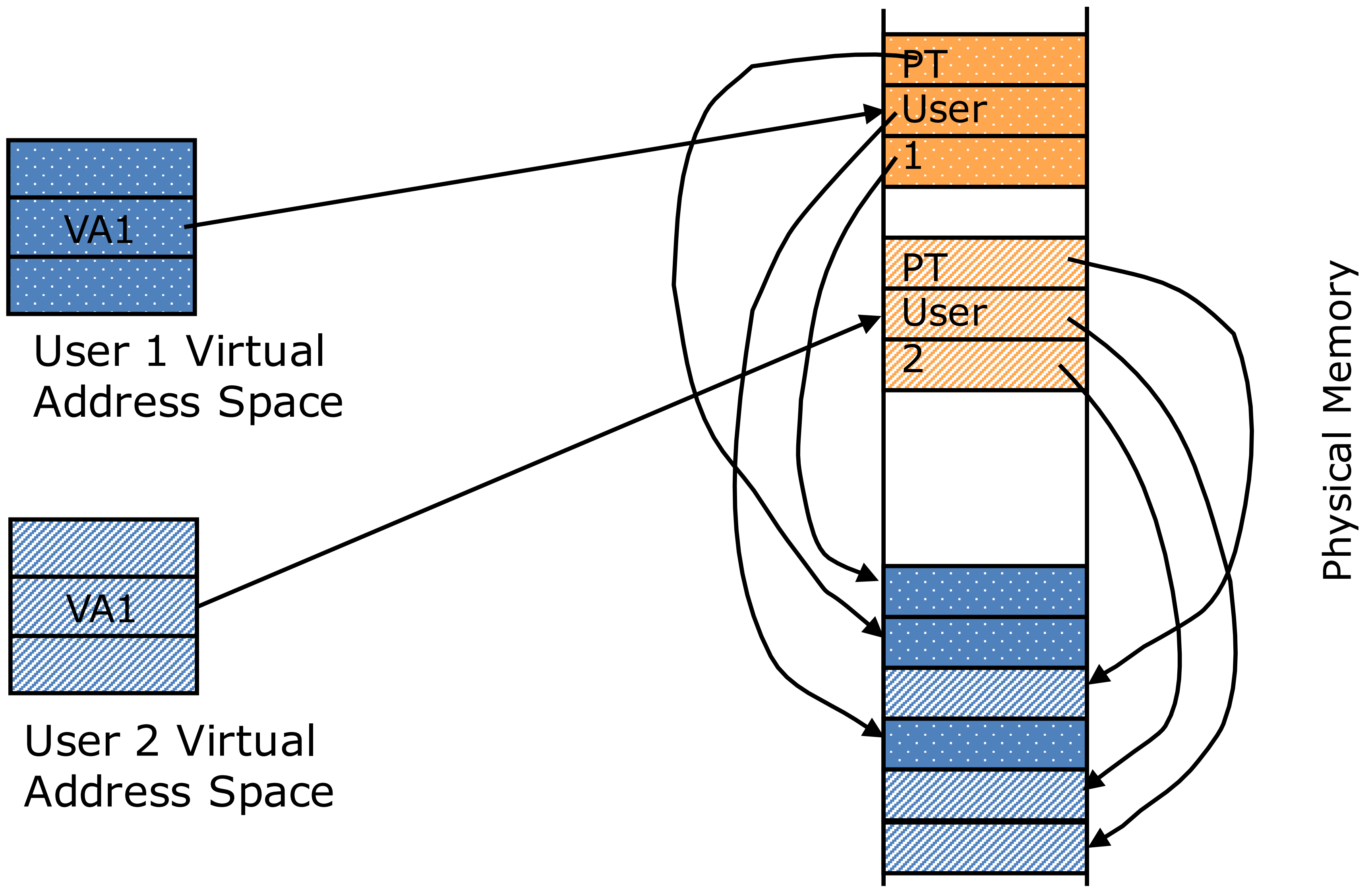
- Idea: Keep PTs in the main memory
    - Needs one reference to retrieve the page base address and another to access the data word
        - *doubles the number of memory references!*

Caching helps

Automatic caching if the processor uses full page tables

Manual caching controlled by the OS with the *Translation Lookaside Buffer (TLB)*

Page Tables in Physical Memory



Page Table Tricks…

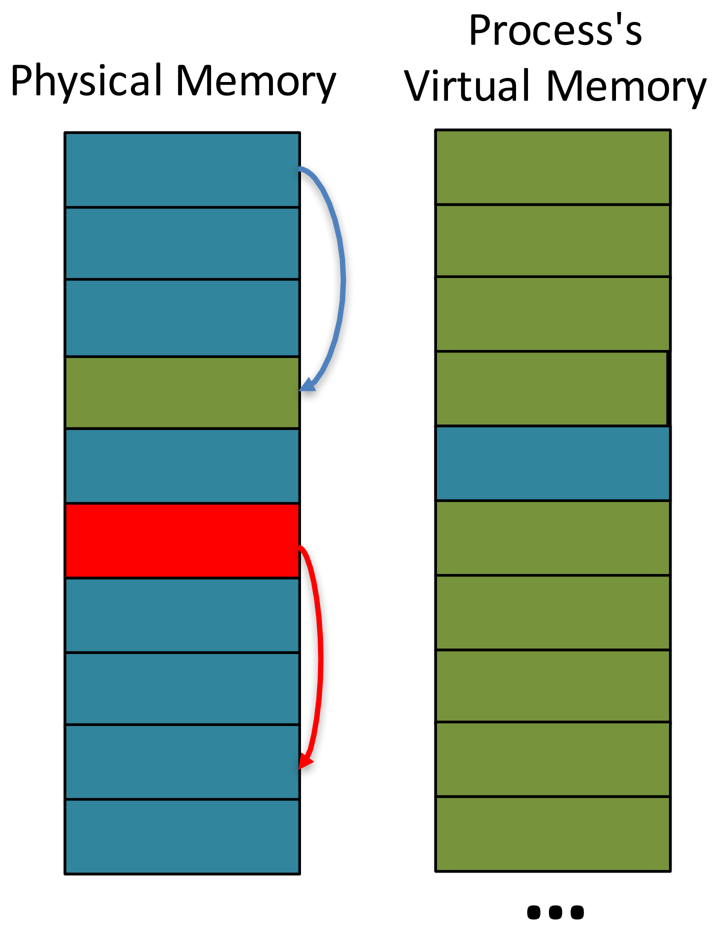
- Can actually have multiple processes referring to the same physical memory
  - Enables "shared memory" between processes
- Page table entry can say "this exists on disk"
  - When such memory is accessed it triggers an exception instead
  - The operating system can copy the data into memory ("swap it in") and then resume the trapped instruction
    - How it gives the illusion of infinite memory
- Can use that same method to efficiently read files
  - File is "memory mapped", when read it is simply paged in like other
  - Allows an efficient method to handle large files conveniently
  - When data is changed can use the same method used to "swap out" unused memory

The Ultimate Page-Table Trick: Rowhammer

- An ***unspeakably cool*** security vulnerability…
- DRAM (unless you pay for error correcting (ECC) memory) is actually unreliable
  - Can repeatedly read/write the same location ("hammer the row" and eventually cause an error in ***some physically distinct memory location***
- Can tell the OS "I want to map this same block of memory at multiple addresses in my process…"
  - Which creates additional page table entries
- Enter ***Rowhammer***
  - It seems all vunerabilities get named now, but this one is cool enough to deserve a name!

How RowHammer Works

- Step 1: Allocate a single page of memory
- Step 2: Make the OS make a gazillion page-table entries pointing to the same page
- Step 3: Hammer the DRAM until one of those entries gets corrupted
  - Now causes that memory page to point to a set of page table entries instead
- Step 4: ***Profit***
  - Well, the ability to read and write to any physical address in the system, same difference



Clicker Question…

- So how cool is this?
  - A -> Supercool
  - E -> Eh, whatever

In Conclusion

- Once we have a basic machine, it’s mostly up to the OS to use it and define application interfaces.
- Hardware helps by providing the right abstractions and features (e.g., Virtual Memory, I/O).
- If you want to learn more about operating systems, you should take CS162!
- What’s next in CS61C?
  - More details on I/O
  - More about Virtual Memory