Virtual Memory

Hakim Weatherspoon
CS 3410
Computer Science
Cornell University

Cornell CIS
COMPUTING AND INFORMATION SCIENCE

[Weatherspoon, Bala, Bracy, McKee, and Sirer]

Where are we now and where are we going?

- How many programs do you run at once?

- a) 1
- b) 2
- c) 3-5
- d) 6-10
- e) 11+

Big Picture: Multiple Processes

- Can we execute ***more than one*** program at a time with our current RISC-V processor?

- a) Yes, no problem at all
- b) No, because memory addresses will conflict
- c) Yes, caches can avoid memory address conflicts
- d) Yes, our modified Harvard architecture avoids memory address conflicts
- e) Yes, because we have multiple processors (multiple cores)

Big Picture: Multiple Processes

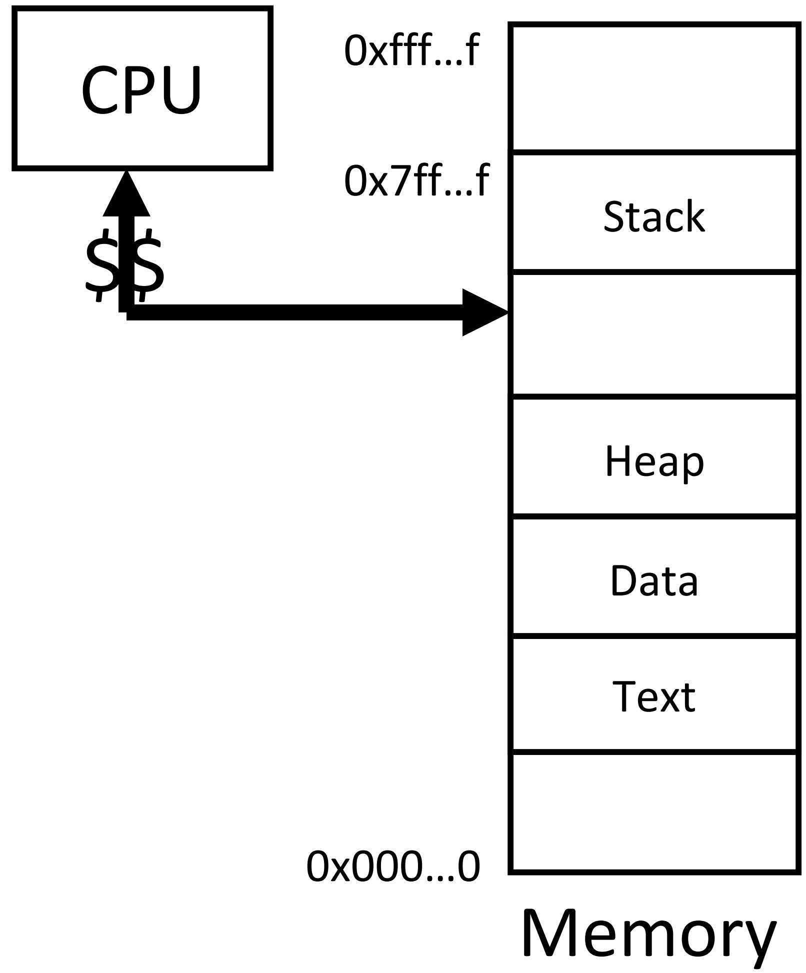
How to run multiple processes?

- *Time-multiplex* a single CPU core (multi-tasking)
  - Web browser, skype, office, … all must co-exist

- Many cores per processor (multi-core) or many processors (multi-processor)
  - Multiple programs run *simultaneously*

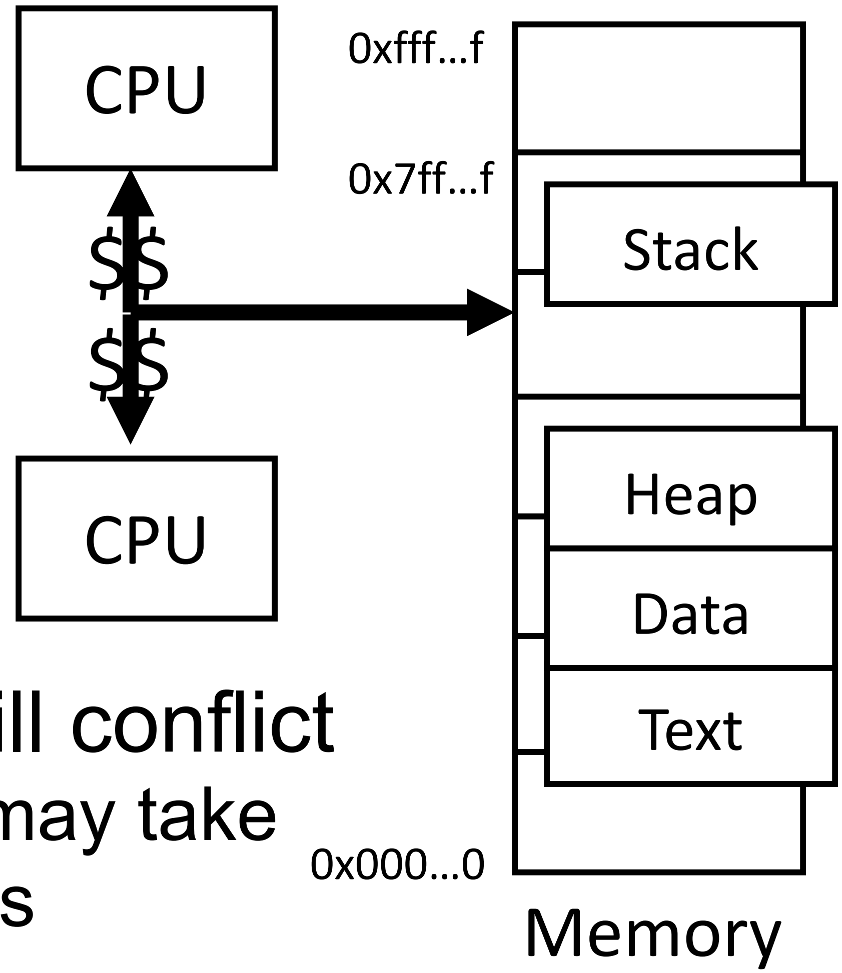
Processor & Memory

- CPU address/data bus...
- … routed through caches
- … to main memory
  - Simple, fast, but…



Multiple Processes

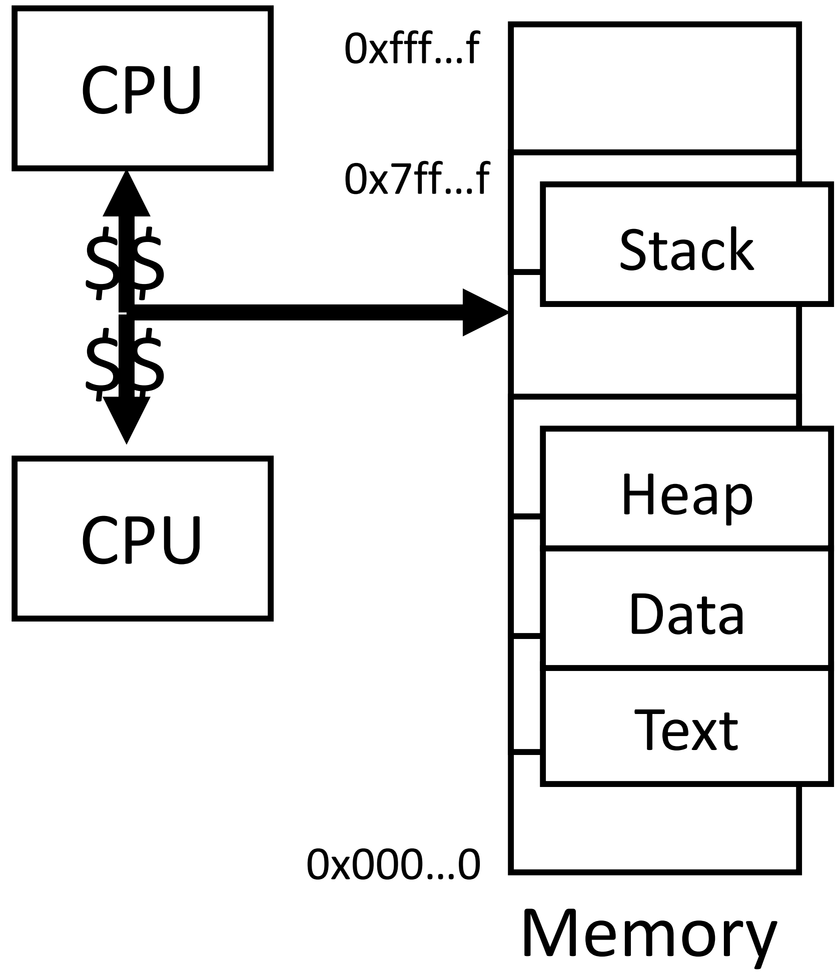
- Q: What happens when another program is executed concurrently on another processor?



- A: The addresses will conflict
  - Even though, CPUs may take turns using memory bus

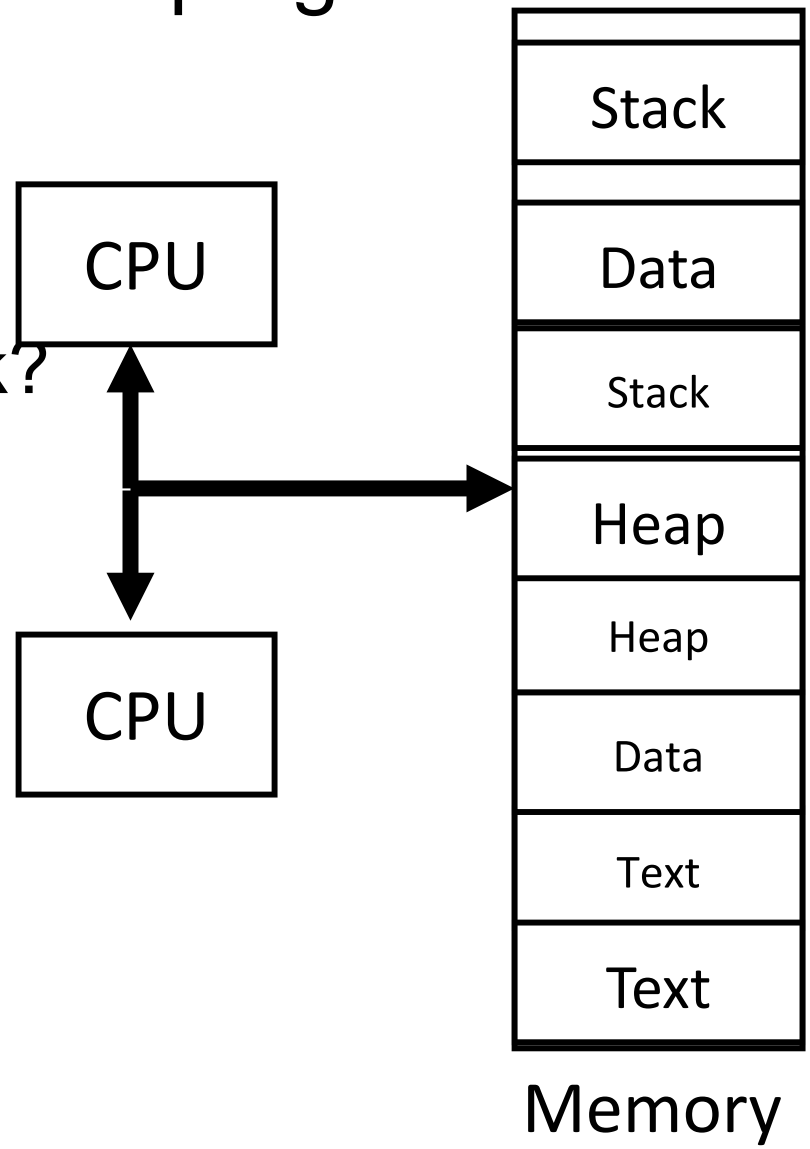
Multiple Processes

- Q: Can we relocate second program?

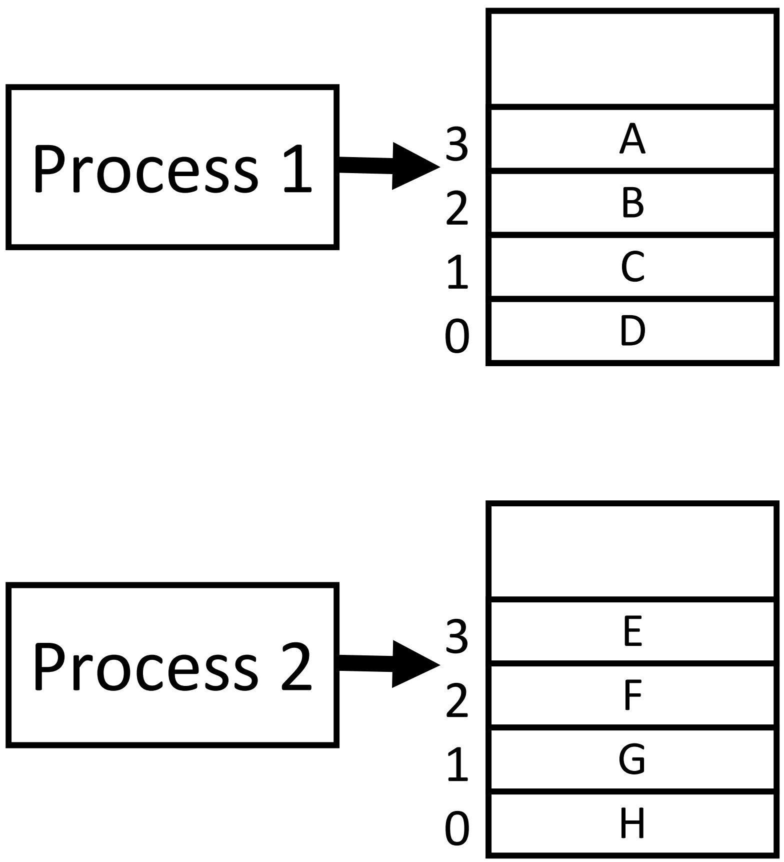


Solution? Multiple processes/processors

- Q: Can we relocate second program?
- A: Yes, but…
  - What if they don't fit?
  - What if not contiguous?
  - Need to recompile/relink?
  - …

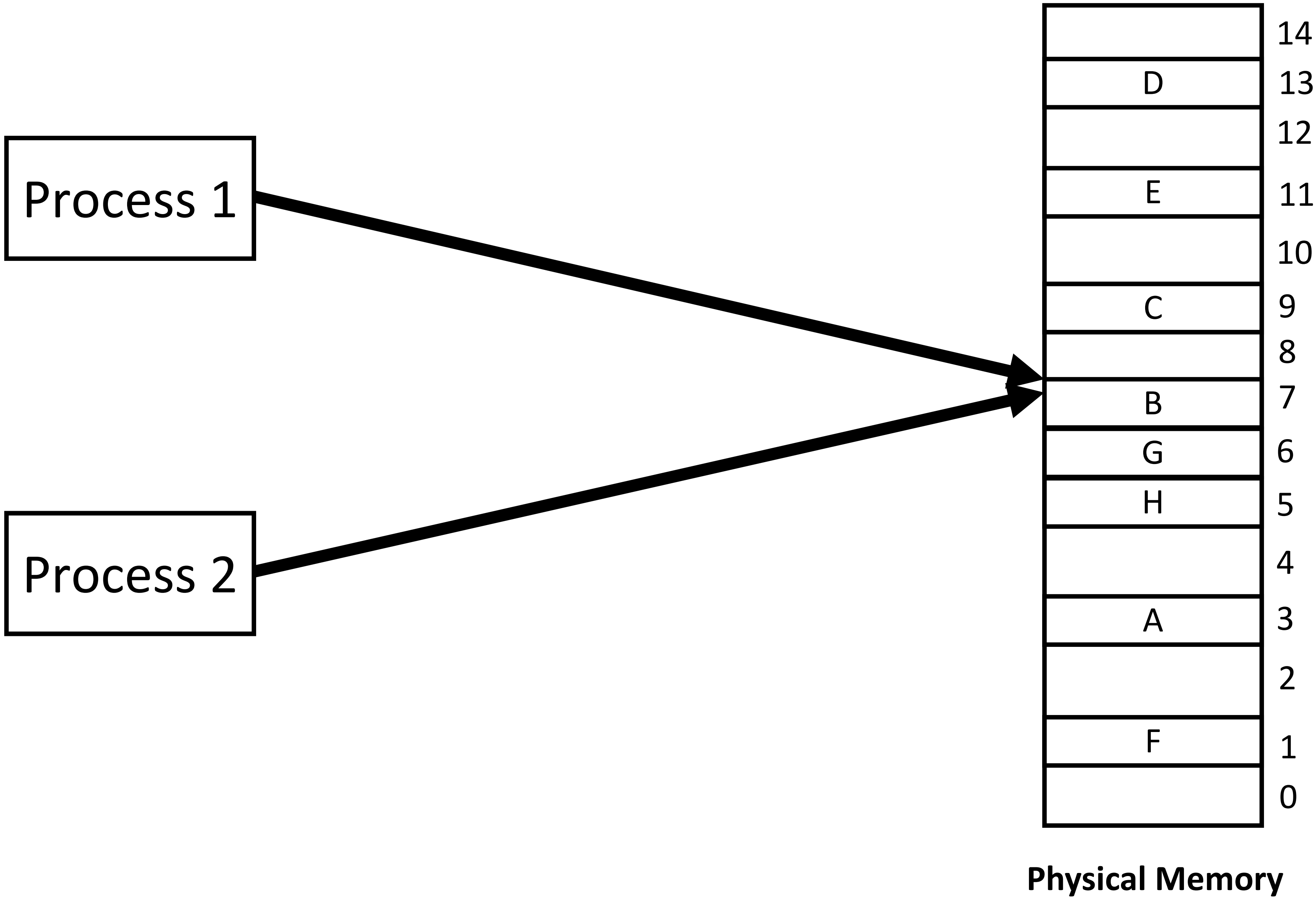


Big Picture: (Virtual) Memory

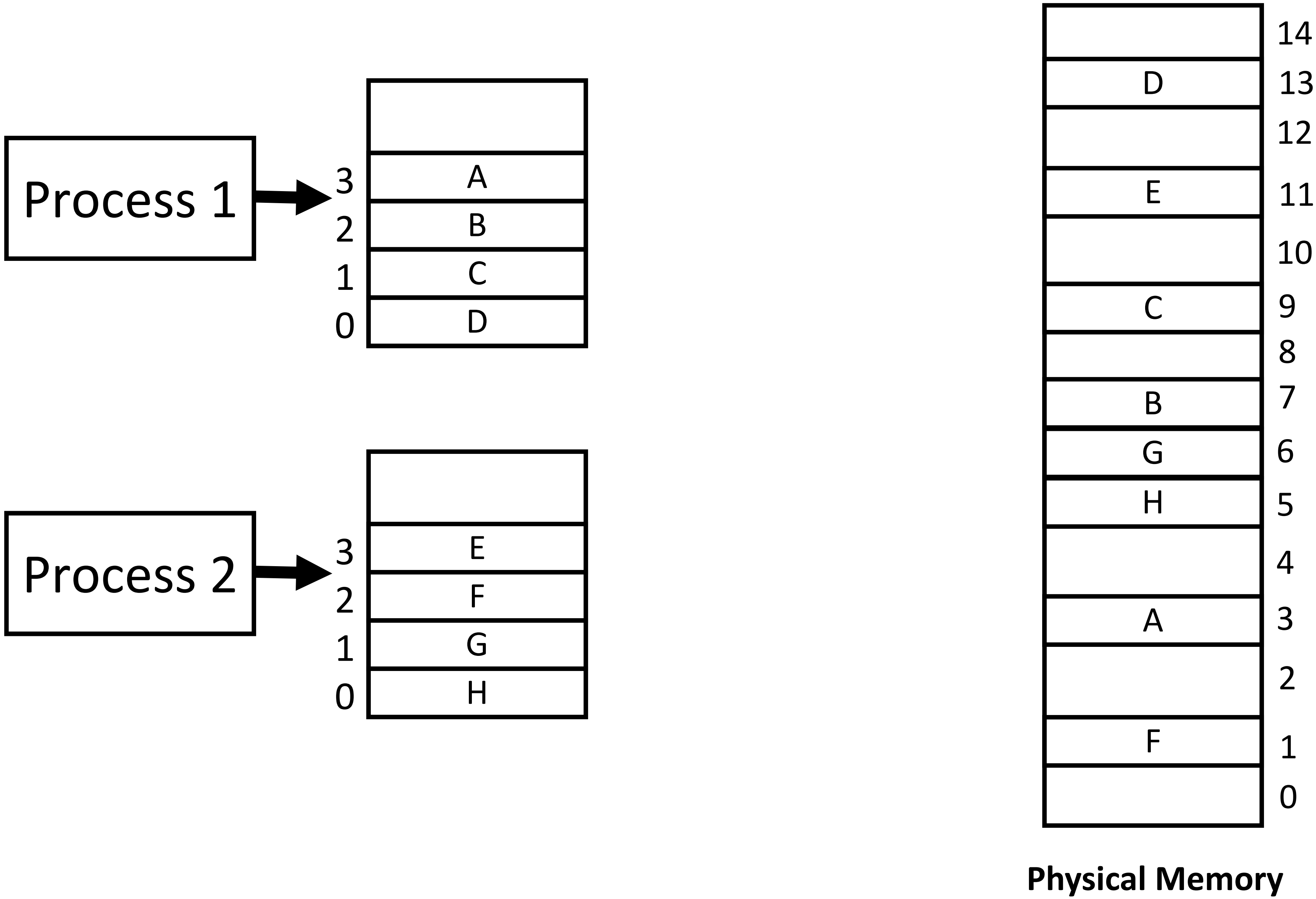


Give each process an illusion that it has exclusive access to entire main memory

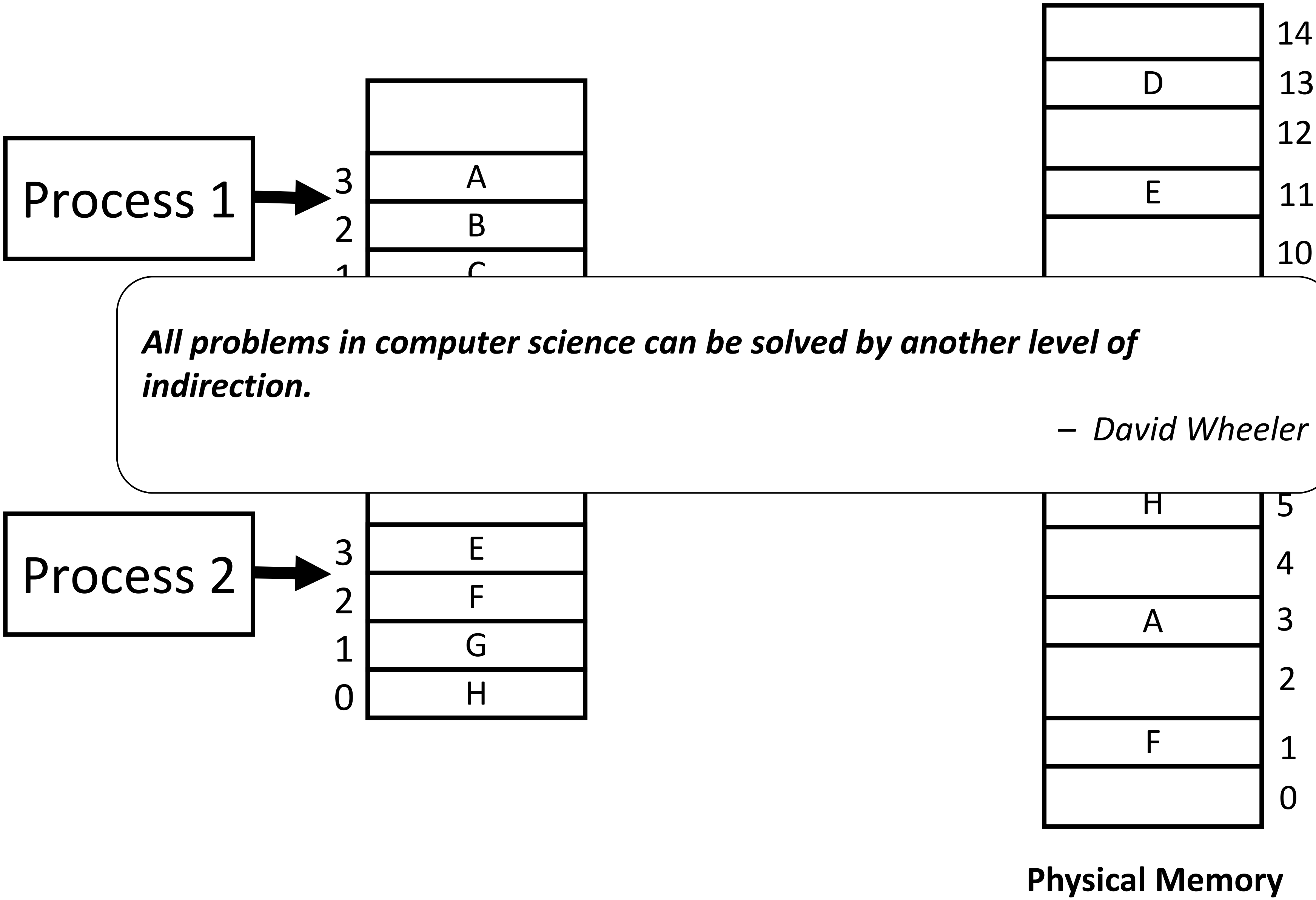
But In Reality…



How do we create the illusion?



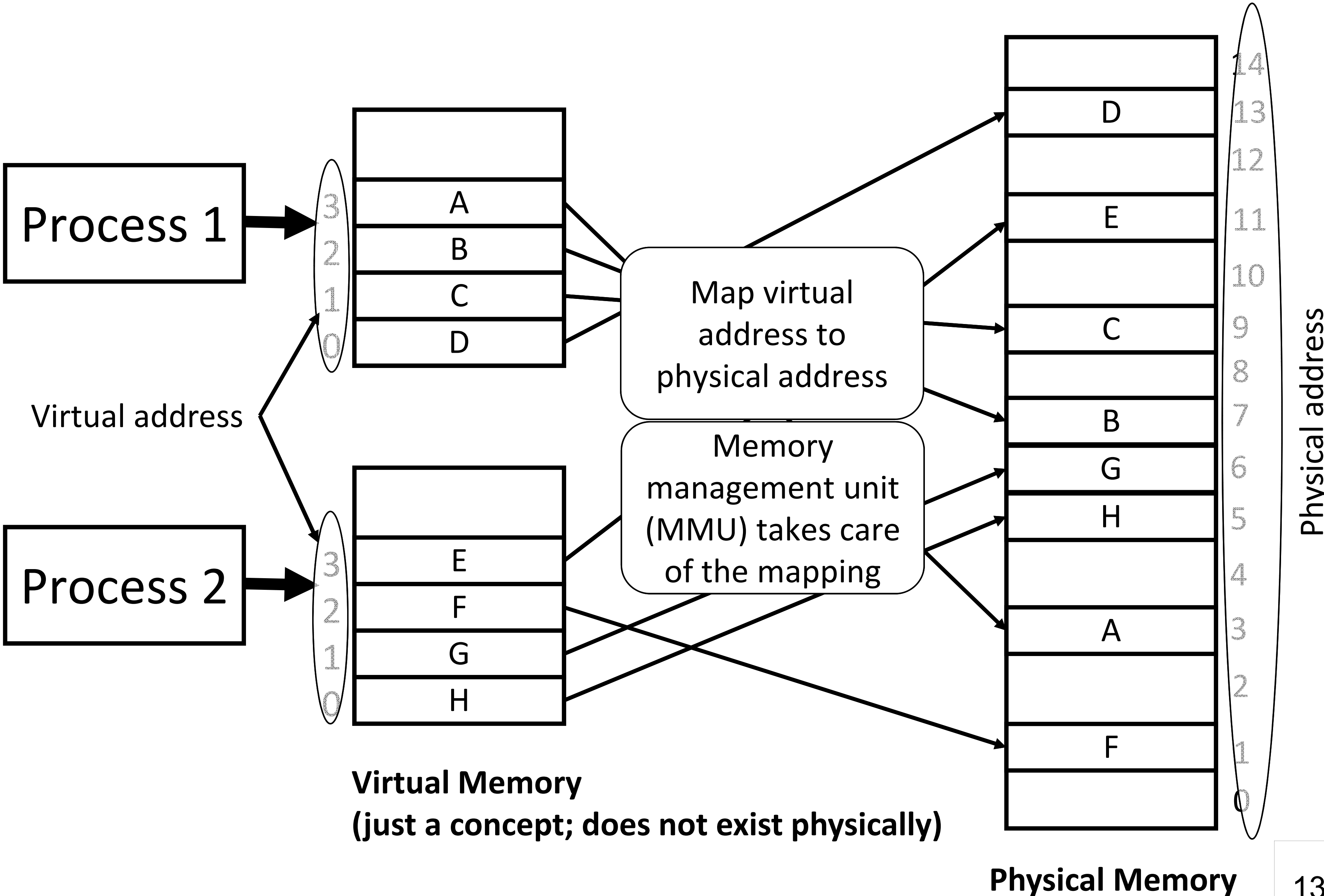
How do we create the illusion?



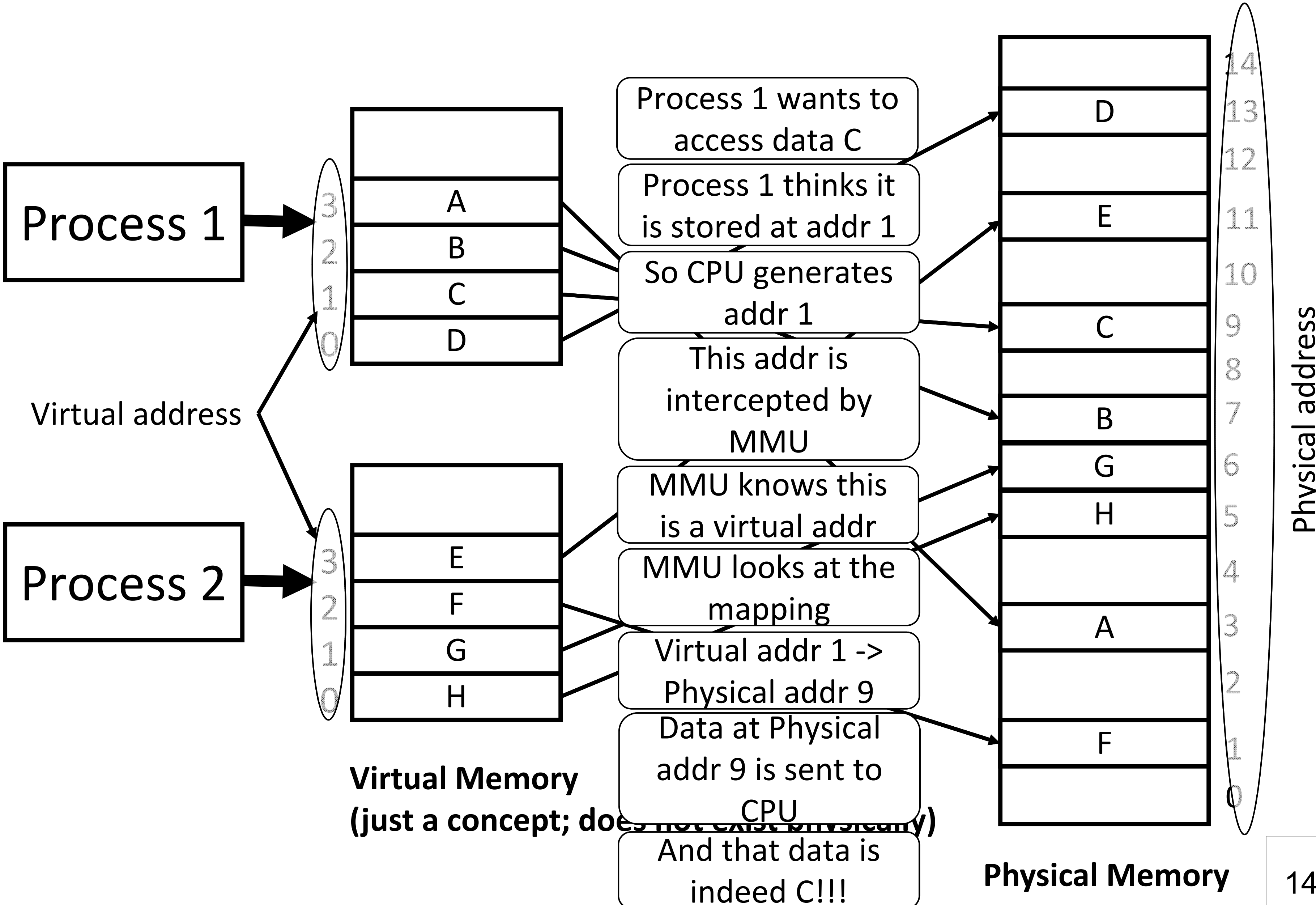
All problems in computer science can be solved by another level of indirection.

– *David Wheeler*

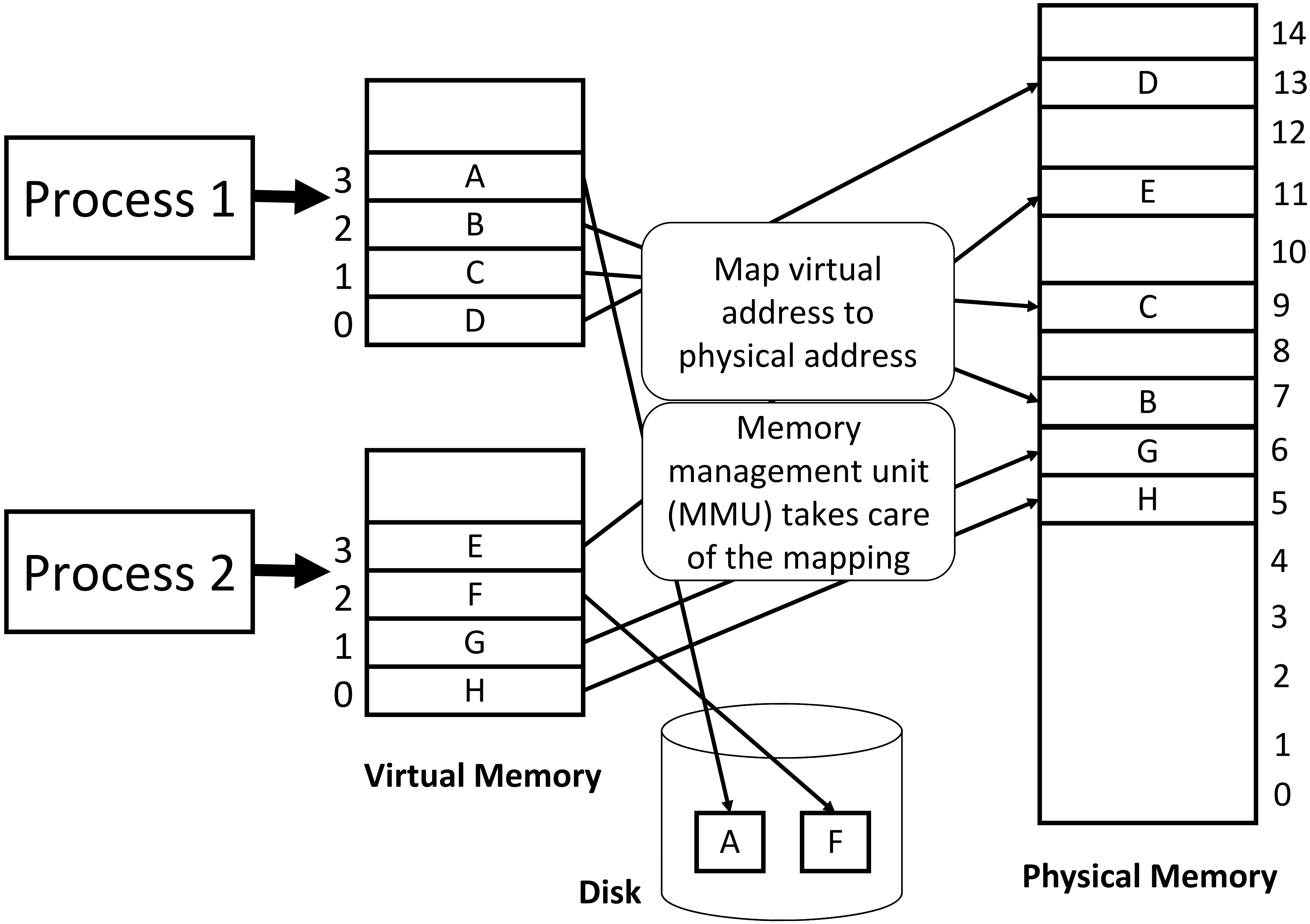
How do we create the illusion?



How do we create the illusion?

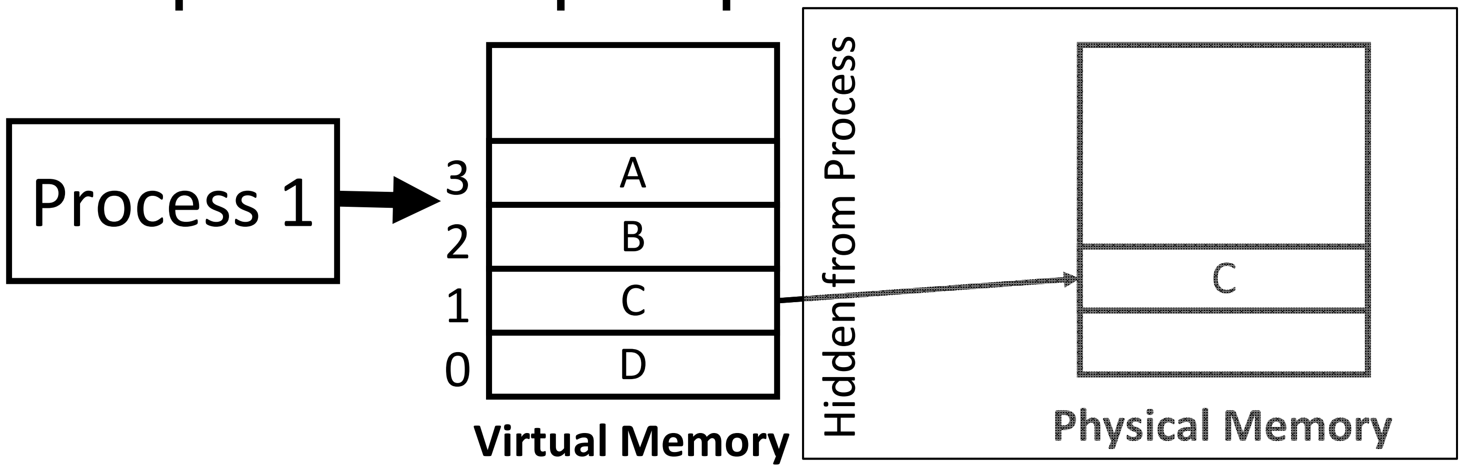


How do we create the illusion?



Big Picture: (Virtual) Memory

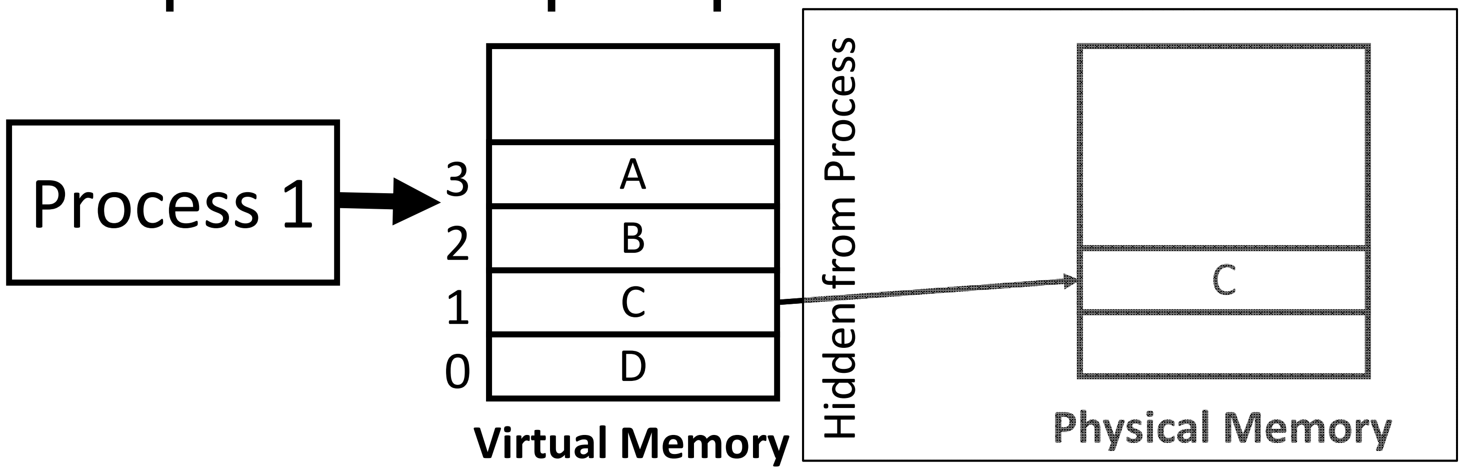
- From a process's perspective –



  - Process only sees the virtual memory
    - ✓Contiguous memory

Big Picture: (Virtual) Memory

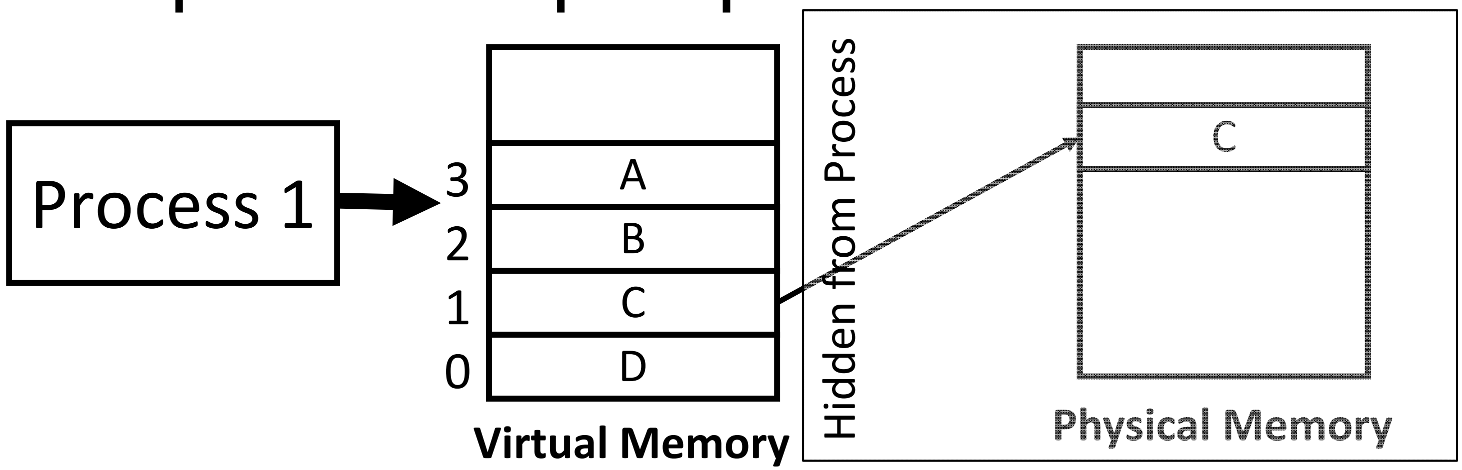
- From a process's perspective –



  - Process only sees the virtual memory
    - ✓ Contiguous memory
    - ✓ No need to recompile - only mappings need to be updated

Big Picture: (Virtual) Memory

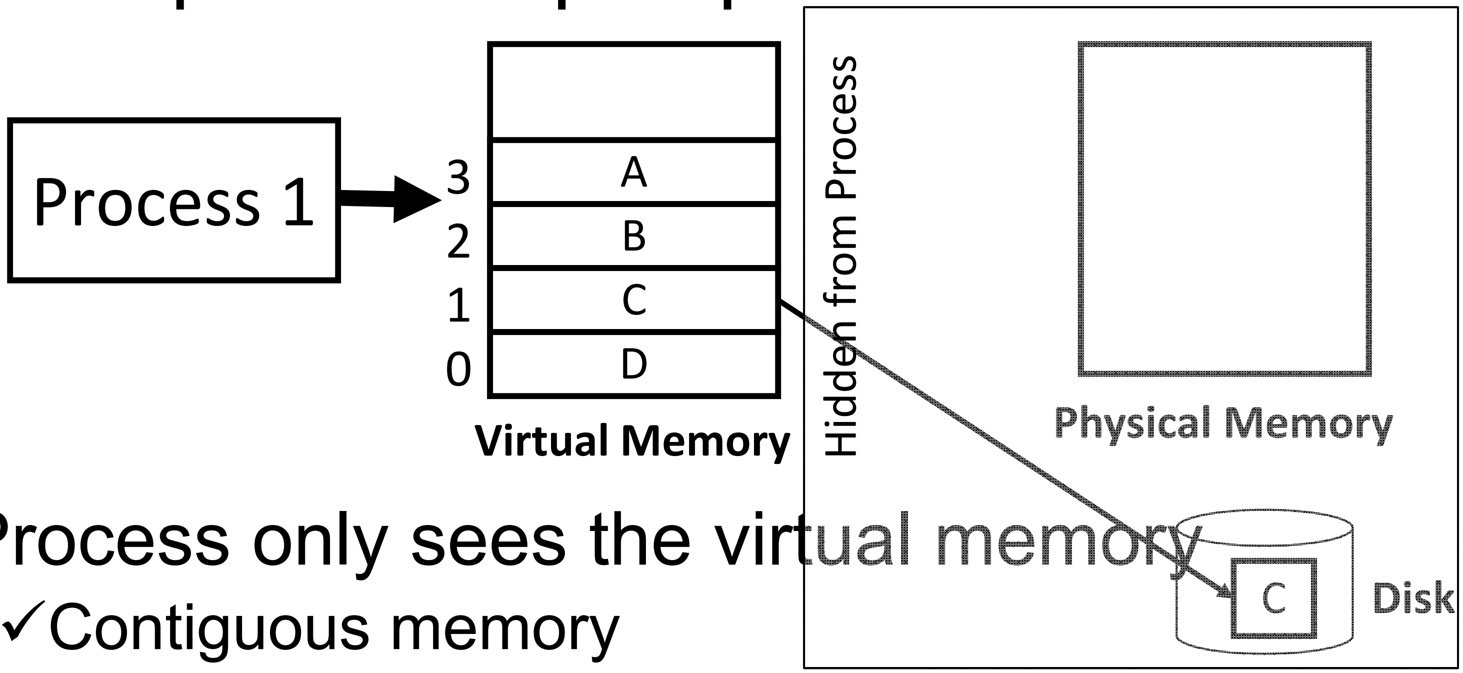
- From a process's perspective –



  - Process only sees the virtual memory
    - ✓Contiguous memory
    - ✓No need to recompile - only mappings need to be updated

Big Picture: (Virtual) Memory

- From a process's perspective –



  - Process only sees the virtual memory
    - ✓Contiguous memory
    - ✓No need to recompile - only mappings need to be updated
    - ✓When run out of memory, MMU maps data on disk in a transparent manner

Next Goal

- How does Virtual Memory work?

- i.e. How do we create the “map” that maps a virtual address generated by the CPU to a physical address used by main memory?

Next Goal (after spring break!)

- How does Virtual Memory work?

- i.e. How do we create the “map” that maps a virtual address generated by the CPU to a physical address used by main memory?

Have a great Spring Break!!!

Cornell University · Founded A.D. 1865 · Cornell CIS · Computing and Information Science

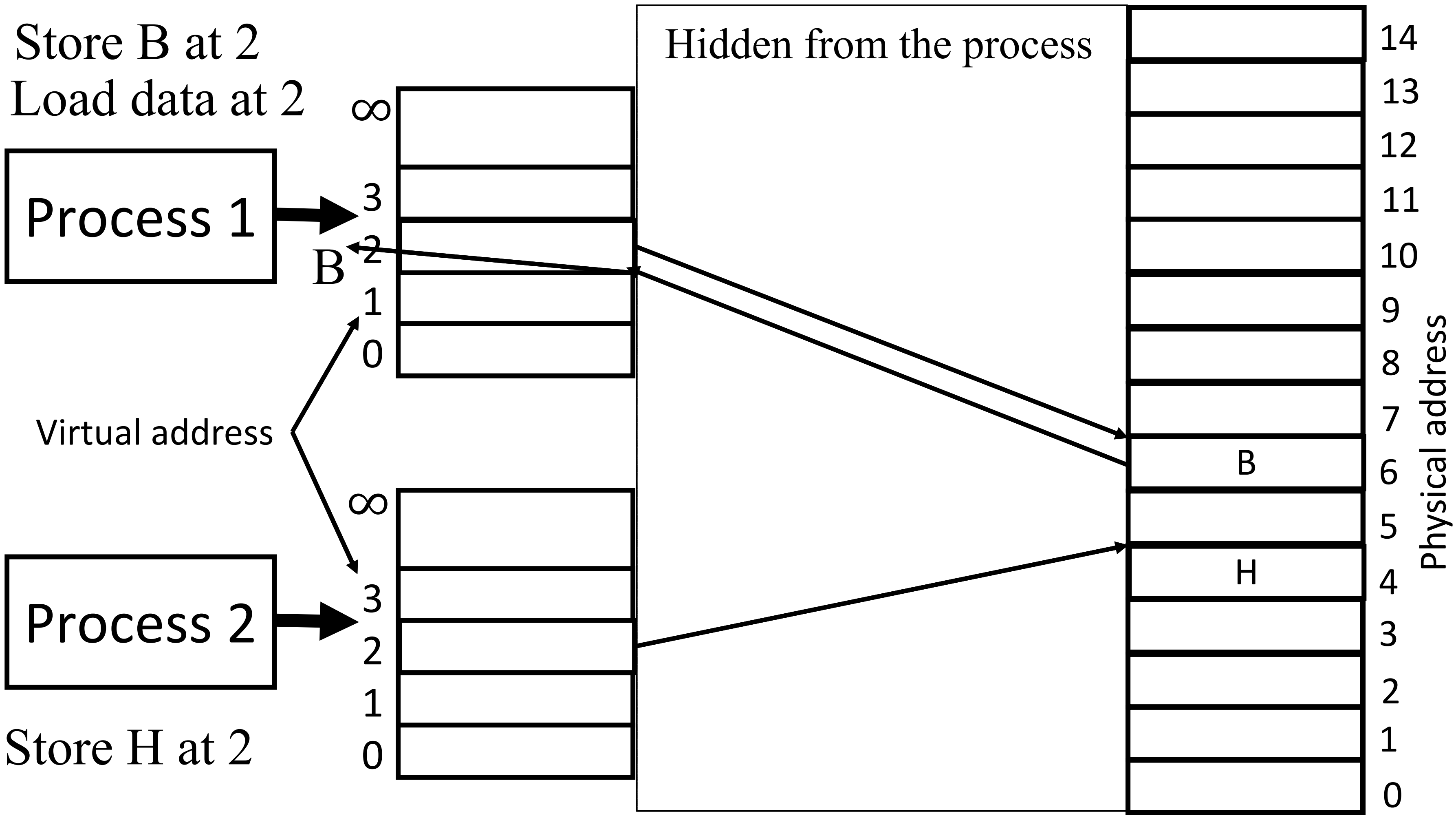
Virtual Memory Agenda

What is Virtual Memory?

How does Virtual memory Work?

- Address Translation
- Overhead
- Paging
- Performance

Virtual Memory: Recap



Virtual Memory
{just a set of numbers (addresses); does not exist physically}

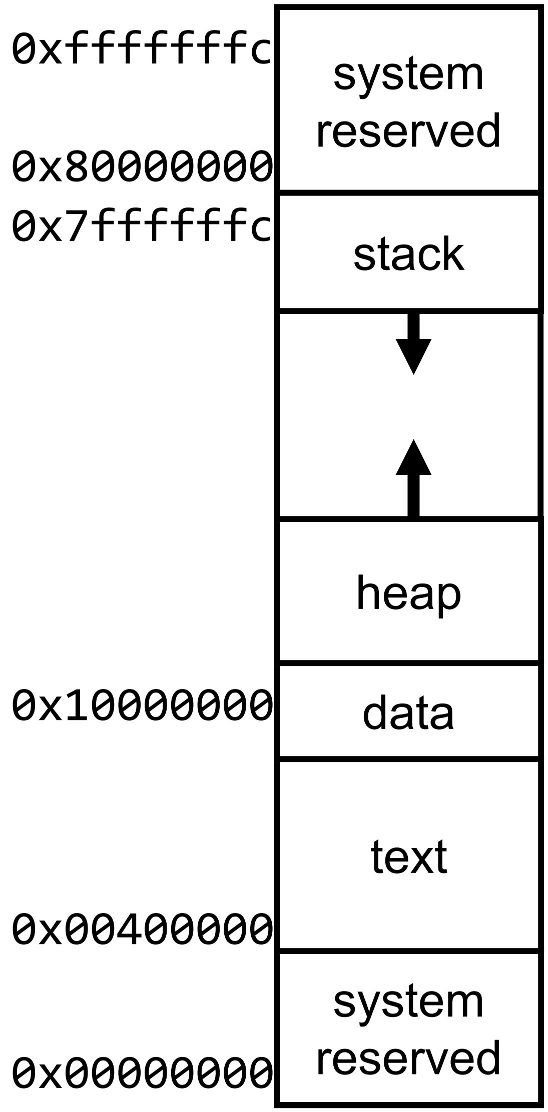
Physical Memory

Picture Memory as… ?

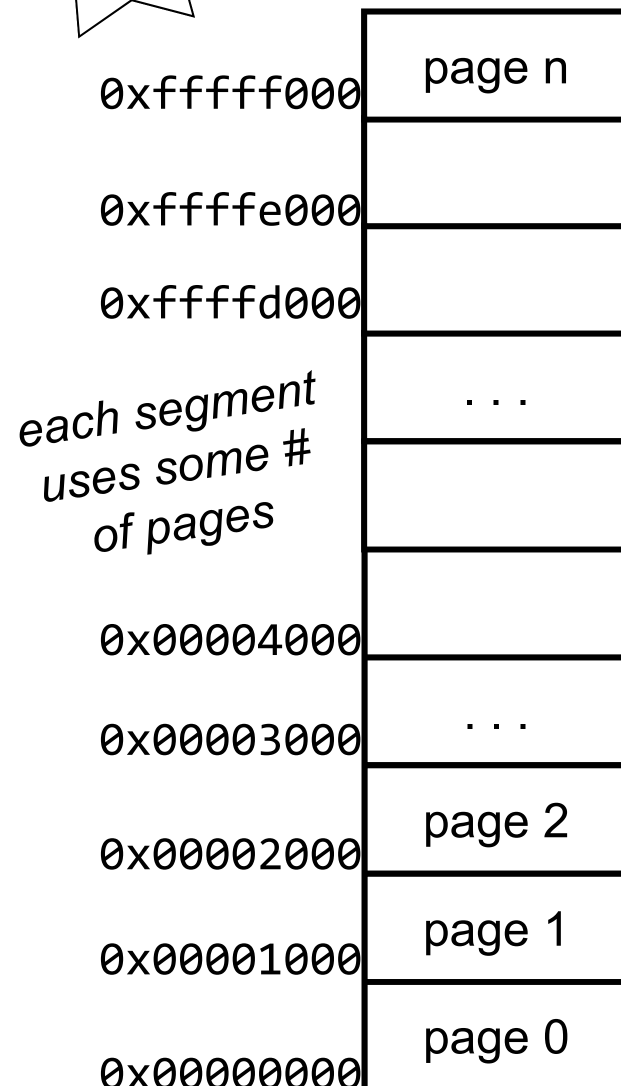
Byte Array:

addr	**data**
0xffffffff	xaa
	…
	…
	x00
	x00
	xef
	xcd
	xab
	xff
0x00000000	x00

Segments:

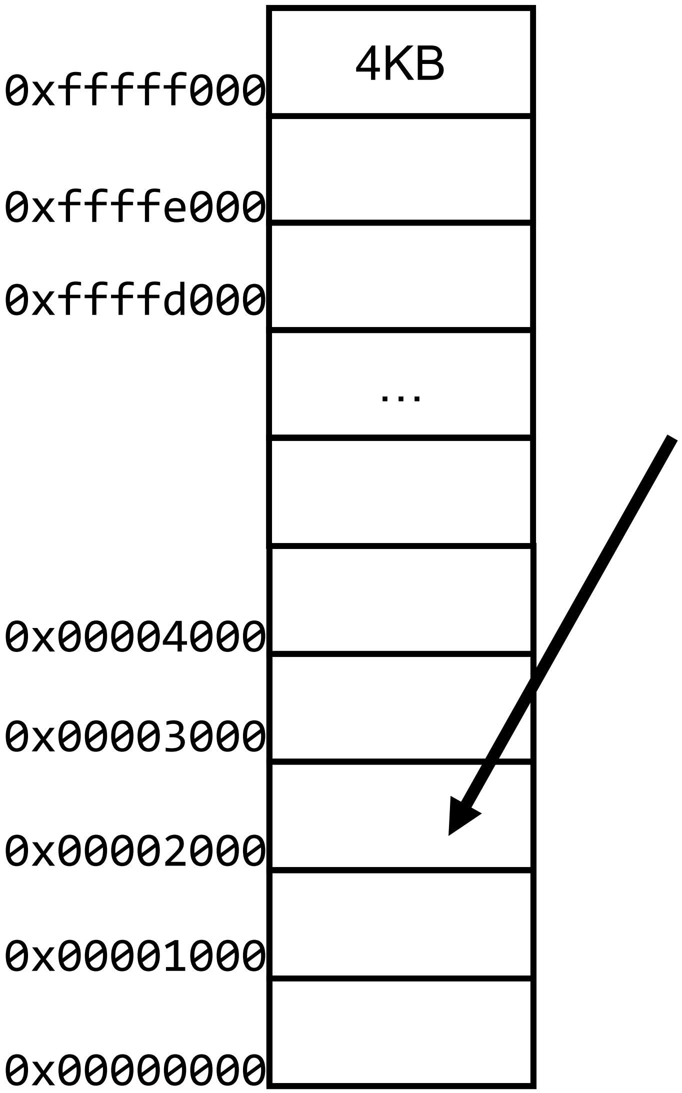


New! Page Array:



A Little More About Pages

Page Array:



Memory size = depends on system
say 4GB

Page size = 4KB (by default)

Then, # of pages = 2^20

Any data in a page # 2 has address of the form: `0x00002xxx`

Lower 12 bits specify which byte you are in the page:

```
0x00002200  = 0010 0000 0000
                 = byte 512
```

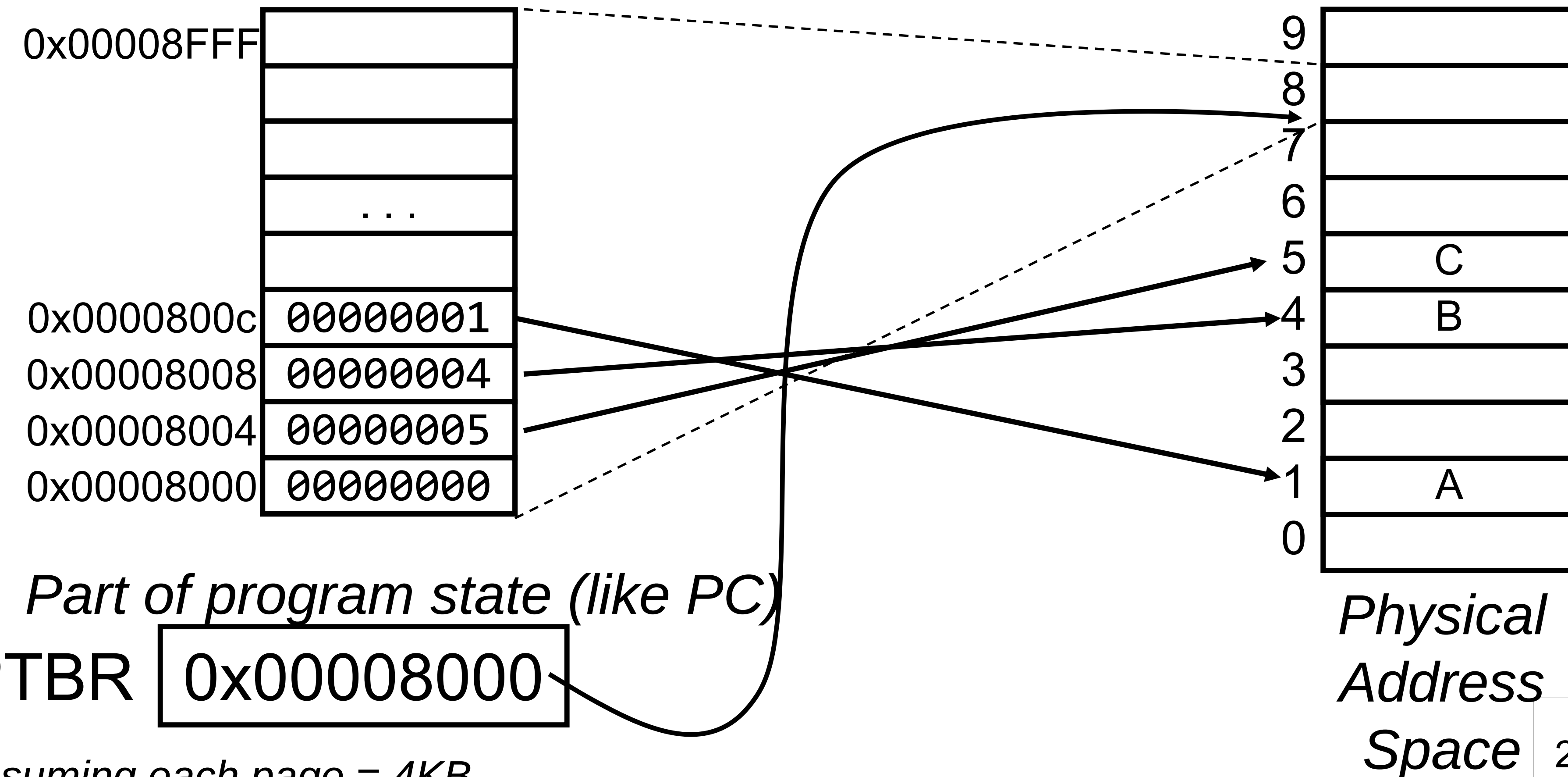
upper bits = page number (PPN)
lower bits = page offset

Page Table:Datastructure to store mapping

1 Page Table *per process*

Lives in Memory, *i.e. in a page (or more…)*

Location stored in **Page Table Base Register**

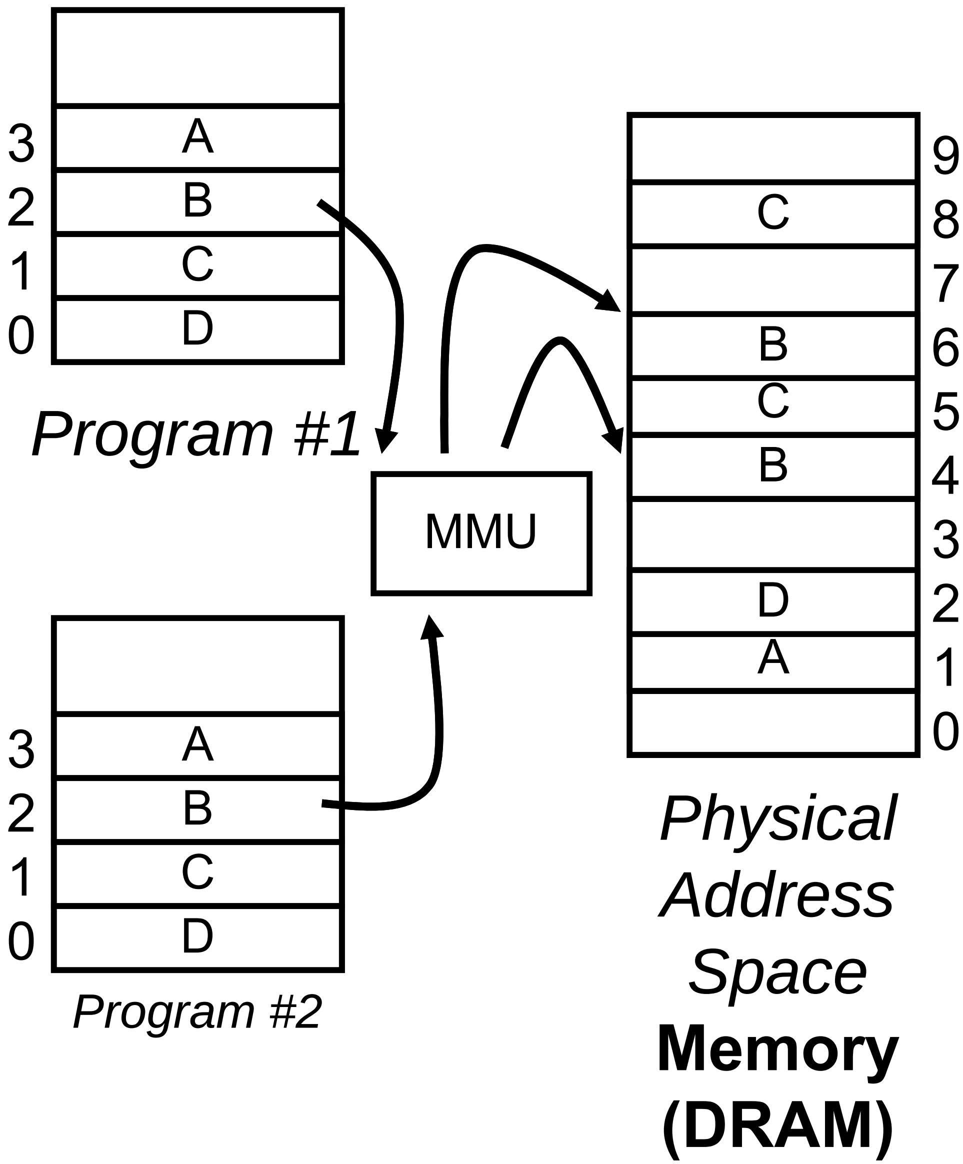


Part of program state (like PC)

PTBR $\boxed{0x00008000}$

Assuming each page = 4KB

Address Translator: MMU

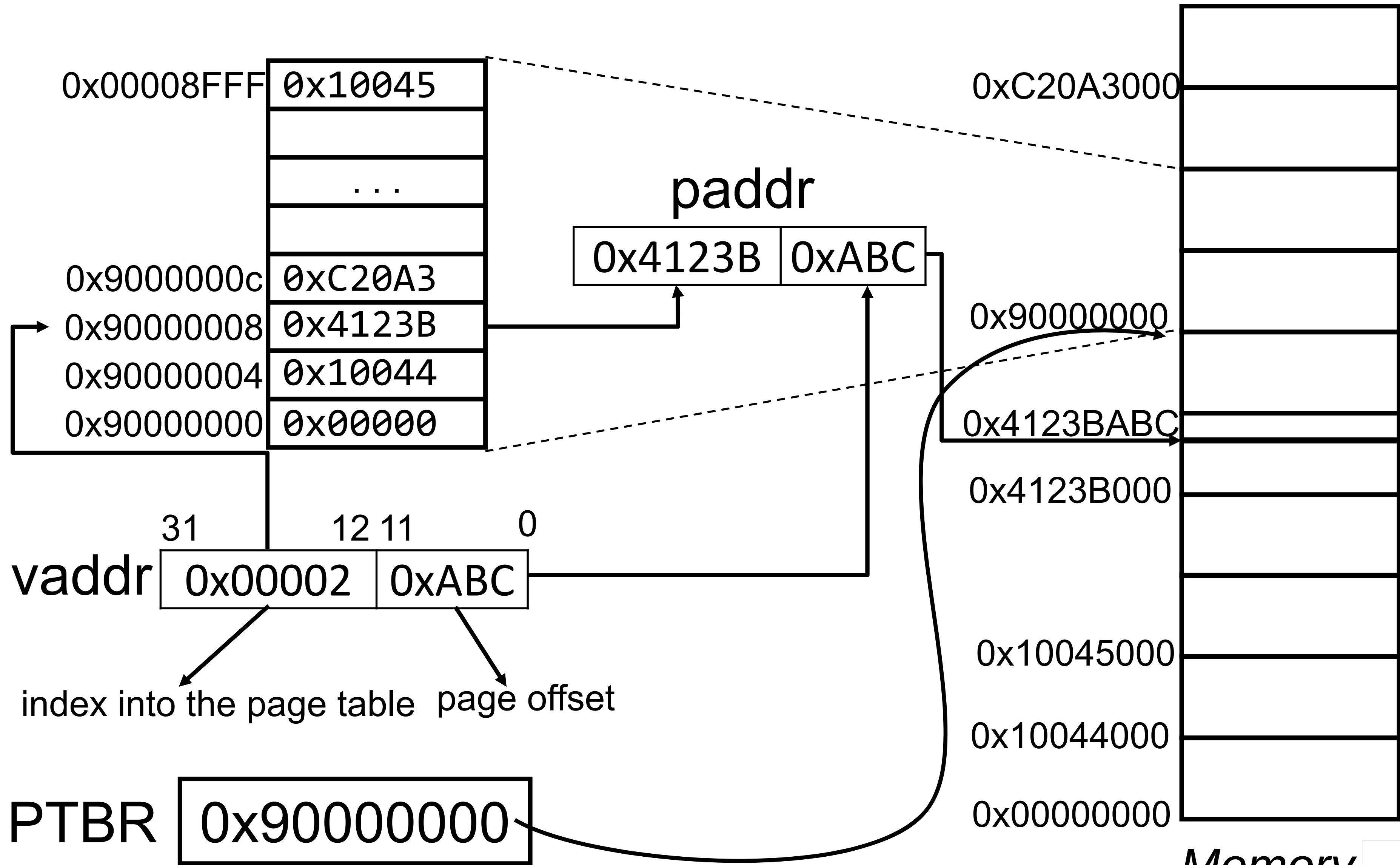


- Programs use virtual addresses
- Actual memory uses physical addresses

Memory Management Unit (MMU)

- HW structure
- Translates virtual → physical address on the fly

Simple Page Table Translation



Assuming each page = 4KB

General Address Translation

- What if the page size is not 4KB?
  → Page offset is no longer 12 bits

Clicker Question:
Page size is 16KB → how many bits is page offset?
(a) 12 (b) 13 $\boxed{\text{(c) 14}}$ (d) 15 (e) 16

- What if Main Memory is not 4GB?
  → Physical page number is no longer 20 bits

Clicker Question:
Page size 4KB, Main Memory 512 MB
→ how many bits is PPN?
(a) 15 (b) 16 $\boxed{\text{(c) 17}}$ (d) 18 (e) 19

Virtual Memory: Summary

Virtual Memory: a Solution for All Problems

- Each process has its own virtual address space
  - Program/CPU can access any address from $0 \ldots 2^N-1$ (N=number of bits in address register)
  - A process is a program being executed
  - Programmer can code as if they own all of memory

- On-the-fly at runtime, for each memory access
  - all accesses are *indirect* through a virtual address
  - map translate fake virtual address to a real physical address
  - redirect load/store to the physical address

Advantages of Virtual Memory

Easy relocation

- Loader puts code anywhere in physical memory
- Virtual mappings to give illusion of correct layout

Higher memory utilization

- Provide illusion of contiguous memory
- Use all physical memory, even physical address 0x0

Easy sharing

- Different mappings for different programs / cores

And more to come…

Takeaway

- All problems in computer science can be solved by another level of indirection.

- Need a map to translate a “fake” virtual address (generated by CPU) to a “real” physical Address (in memory)

- Virtual memory is implemented via a “Map”, a ***PageTage,*** that maps a ***vaddr*** (a virtual address) to a ***paddr*** (physical address):

  paddr = PageTable[vaddr]

Feedback

- How much did you love today's lecture?
  A: As much as Melania loves Trump
  B: As much as Kanye loves Kanye
  C: Somewhere in between, but closer to A
  D: Somewhere in between, but closer to B
  E: I am incapable of loving anything ☹

Virtual Memory Agenda

What is Virtual Memory?

How does Virtual memory Work?

- Address Translation
- Overhead
- Paging
- Performance

Page Table Overhead

- How large is PageTable?
- Virtual address space (for each process):
  - Given: total virtual memory: 2^{32} bytes = 4GB
  - Given: page size: 2^{12} bytes = 4KB
  - **# entries in PageTable?** 2^{20} = 1 million entries
  - **size of PageTable?** PTE size = 4 bytes
    →PageTable size = 4 x 2^{20} = 4MB
- Physical address space:
  - total physical memory: 2^{29} bytes = 512MB
  - overhead for 10 processes?

10 x 4MB = 40 MB of overhead!

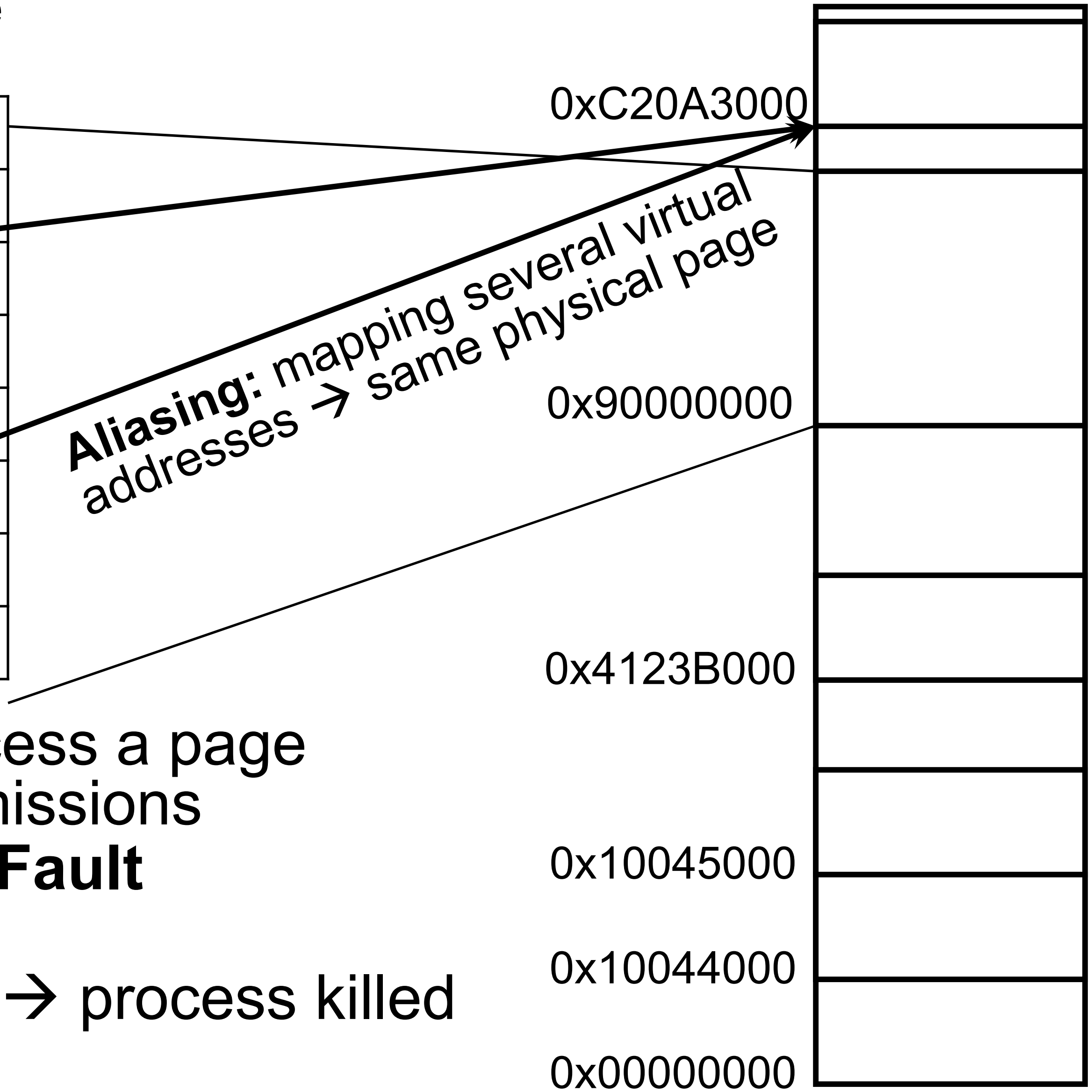
- 40 MB /512 MB = 7.8% overhead, space due to PageTable

But Wait... There's more!

- Page Table Entry won't be just an integer
- Meta-Data
  - Valid Bits
    - *What PPN means "not mapped"?* No such number…
    - **At first:** not all virtual pages will be in physical memory
    - **Later:** might not have enough physical memory to map all virtual pages
  - Page Permissions
    - R/W/X permission bits for each PTE
    - **Code:** read-only, executable
    - **Data:** writeable, not executable

Less Simple Page Table

V	R	W	X	Physical Page Number
0				
1	1	1	0	0xC20A3
0				
0				
1	1	0	0	0xC20A3
1				0x4123B
1				0x10044
0				



Process tries to access a page without proper permissions

Segmentation Fault

Example:
Write to read-only? → process killed

Now how big is this Page Table?

```
struct pte_t page_table[2^20]
Each PTE = 8 bytes
```

How many pages in memory will the page table take up?

Clicker Question:

(a) 4 million (2^{22}) pages
(b) 2048 (2^{11}) pages
(c) 1024 (2^{10}) pages
(d) 4 billion (2^{32}) pages
(e) 4K (2^{12}) pages

Assuming each page = 4KB

Now how big is this Page Table?

```
struct pte_t page_table[2^20]
Each PTE = 8 bytes
```

How many pages in memory will the page table take up?

Clicker Question:

(a) 4 million (2^{22}) pages

(b) 2048 (2^{11}) pages

(c) 1024 (2^{10}) pages

(d) 4 billion (2^{32}) pages

(e) 4K (2^{12}) pages

Assuming each page = 4KB

Wait, how big is this Page Table?

`page_table[`2^{20}`] = 8x`2^{20}` =`2^{23}` bytes`
`(Page Table = 8 MB in size)`

How many pages in memory will the page table take up? $2^{23}/2^{12} = 2^{11}$ `2K pages!`

Clicker Question:

(a) 4 million (2^{22}) pages
(b) 2048 (2^{11}) pages
(c) 1024 (2^{10}) pages
(d) 4 billion (2^{32}) pages
(e) 4K (2^{12}) pages

Assuming each page = 4KB

Takeaway

- All problems in computer science can be solved by another level of indirection.
- Need a map to translate a “fake” virtual address (generated by CPU) to a “real” physical Address (in memory)

- Virtual memory is implemented via a “Map”, a ***PageTage,*** that maps a ***vaddr*** (a virtual address) to a ***paddr*** (physical address):
- ***paddr = PageTable[vaddr]***

- A page is constant size block of virtual memory. Often, the page size will be around 4kB to reduce the number of entries in a PageTable.

- We can use the PageTable to set Read/Write/Execute permission on a per page basis. Can allocate memory on a per page basis. Need a valid bit, as well as Read/Write/Execute and other bits.
- But, overhead due to PageTable is significant.

Next Goal

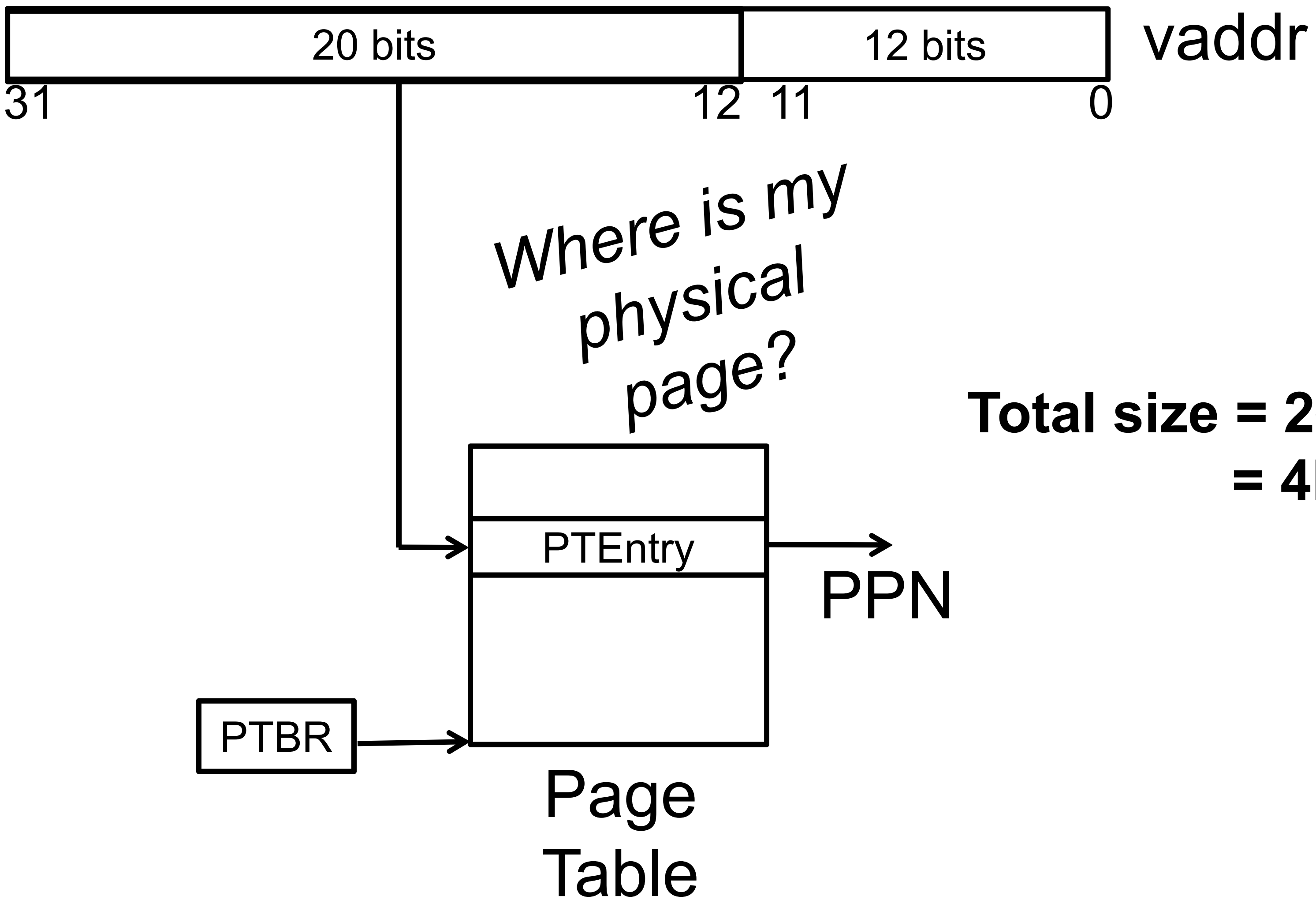
- How do we reduce the size (overhead) of the PageTable?

Next Goal

- How do we reduce the size (overhead) of the PageTable?

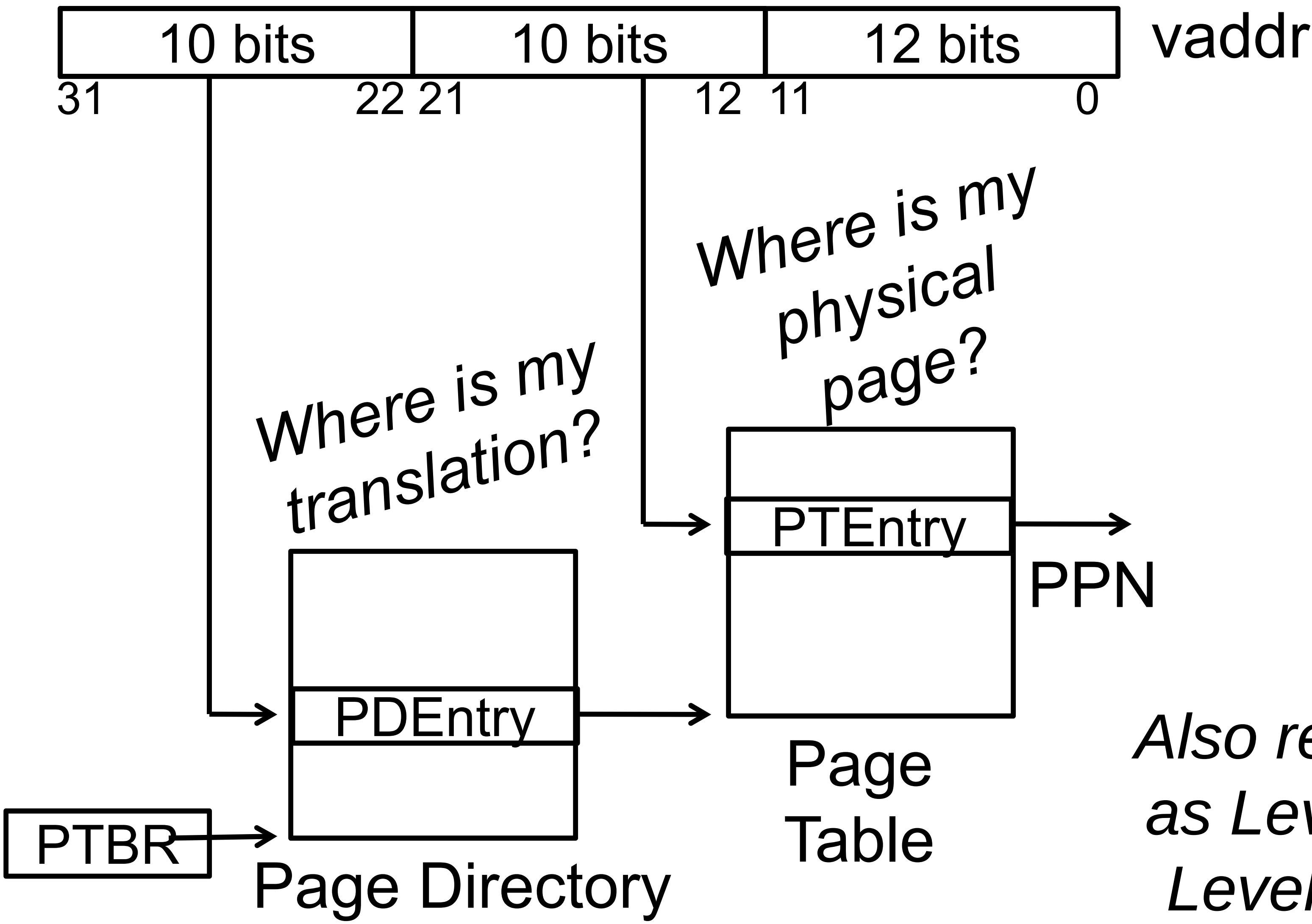
- A: Another level of indirection!!

Single-Level Page Table



Total size = 2^{20} * 4 bytes
= 4MB

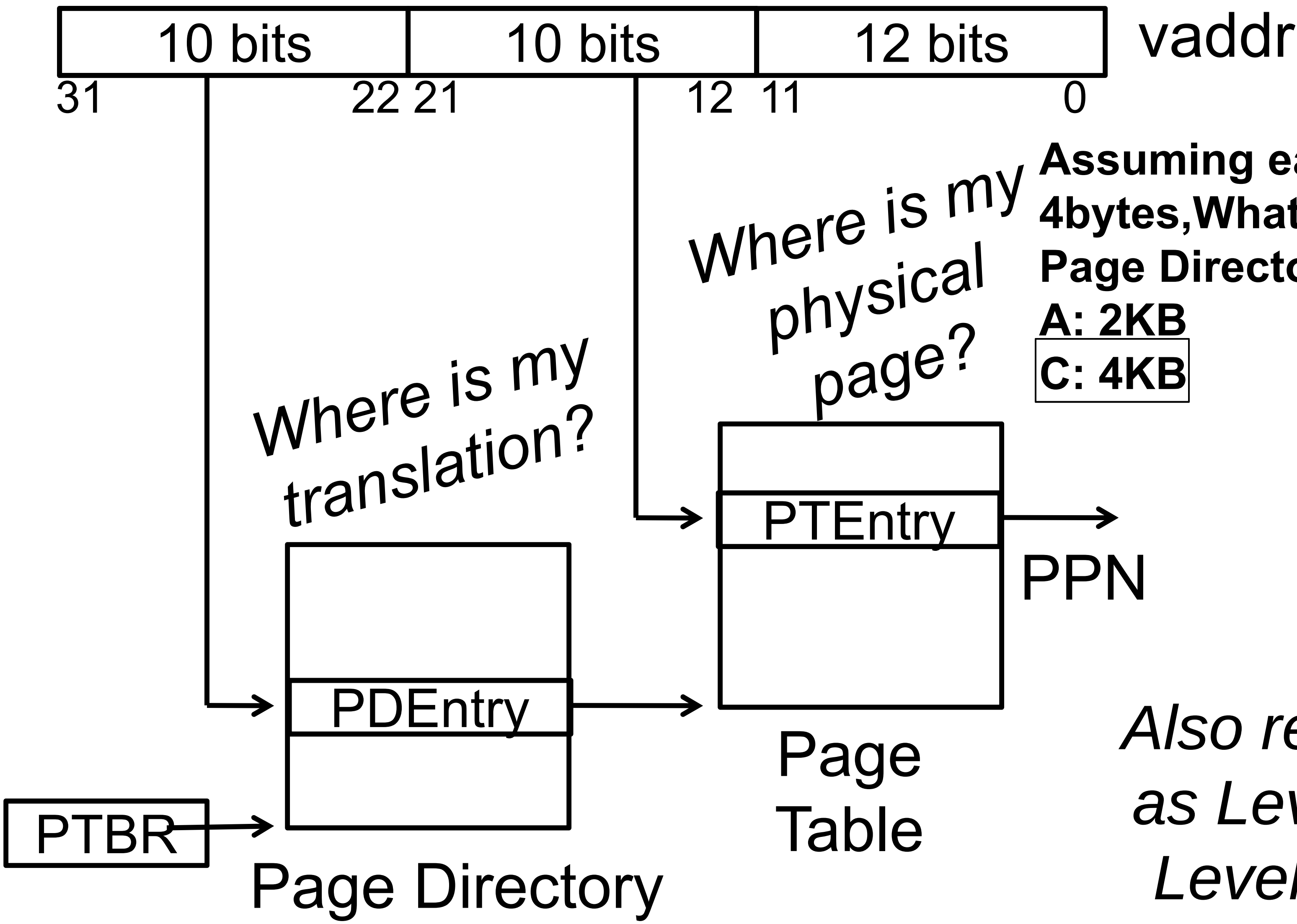
Multi-Level Page Table



Also referred to as Level 1 and Level 2 Page Tables

* *Indirection to the Rescue, AGAIN!*

Multi-Level Page Table



Assuming each entry is 4bytes,What is the size of Page Directory?

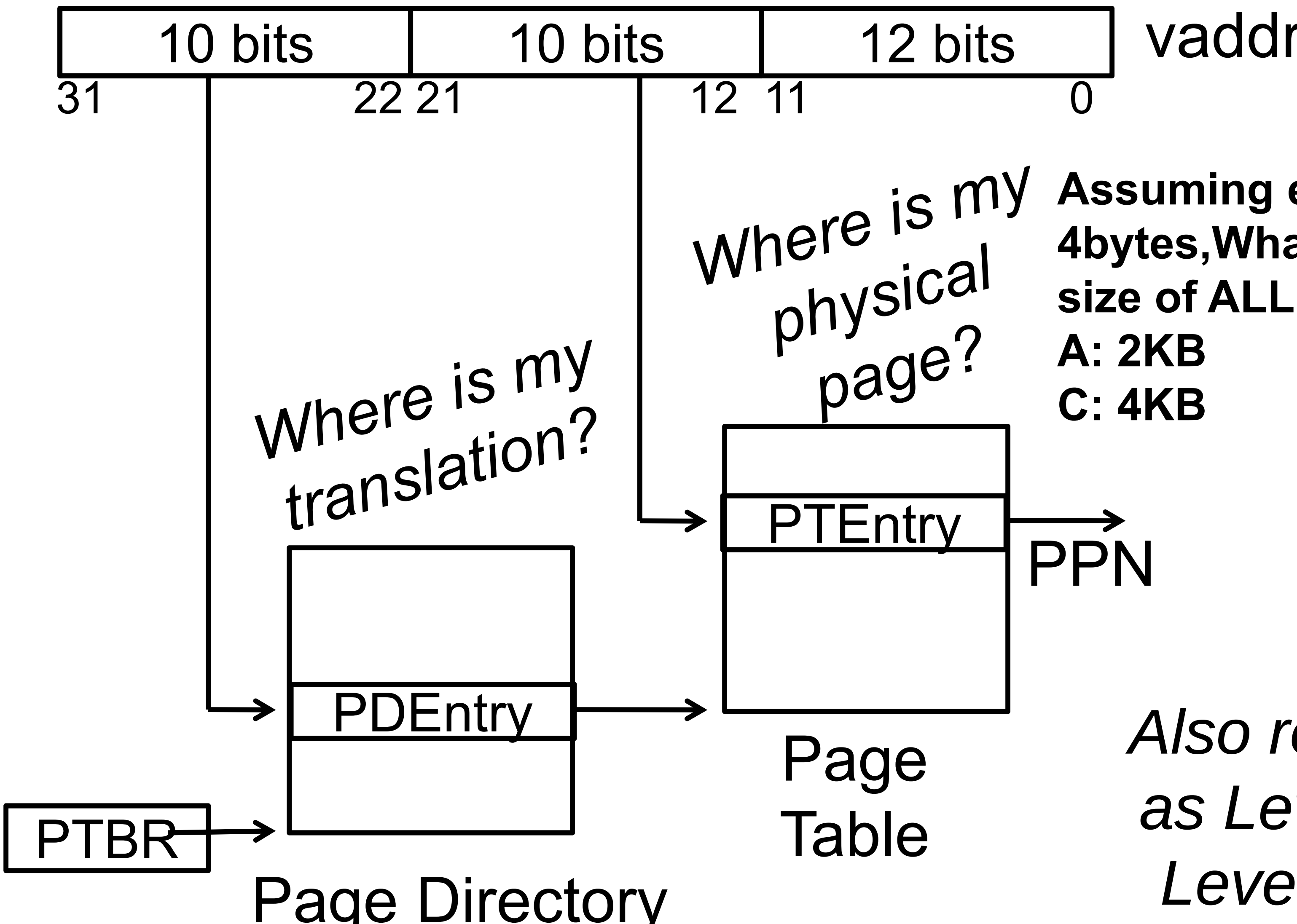
A: 2KB **B: 2MB**

C: 4KB **D: 4MB**

Also referred to as Level 1 and Level 2 Page Tables

* *Indirection to the Rescue, AGAIN!*

Multi-Level Page Table



Assuming each entry is 4bytes,What is the total size of ALL Page tables?

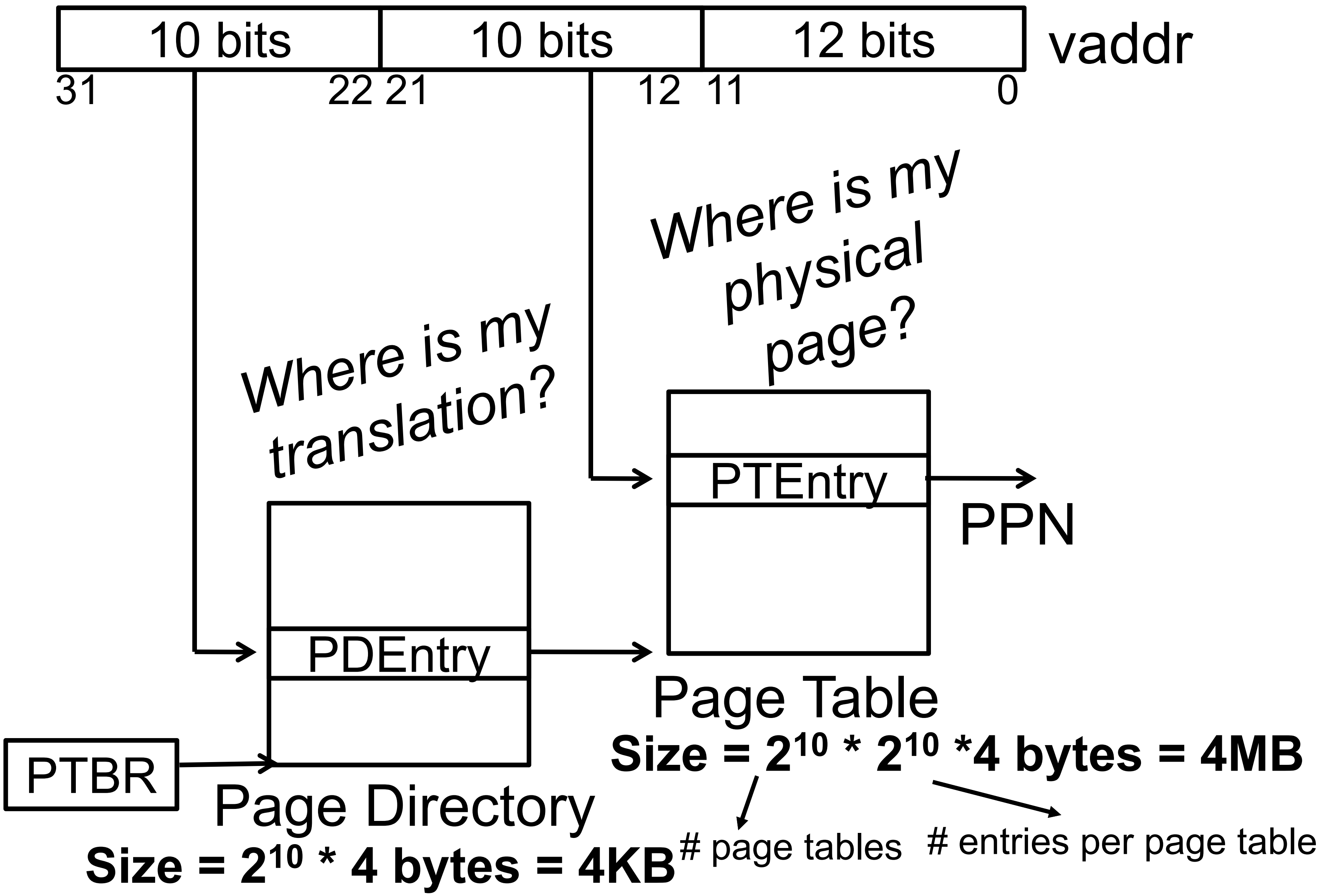
A: 2KB **B: 2MB**

C: 4KB $\boxed{\textbf{D: 4MB}}$

Also referred to as Level 1 and Level 2 Page Tables

* *Indirection to the Rescue, AGAIN!*

Multi-Level Page Table



Multi-Level Page Table

Doesn't this take up more memory than before?

- YES, but..

Benefits

- Don't need 4MB contiguous physical memory
- Don't need to allocate every PageTable, only those containing valid PTEs

Drawbacks

- Performance: Longer lookups

Virtual Memory Agenda

What is Virtual Memory?

How does Virtual memory Work?

- Address Translation
- Overhead
- Paging
- Performance

Paging

What if process requirements > physical memory?

Virtual starts earning its name

Memory acts as a cache for secondary storage (disk)

- Swap memory pages out to disk when not in use
- Page them back in when needed

Courtesy of Temporal & Spatial Locality (again!)

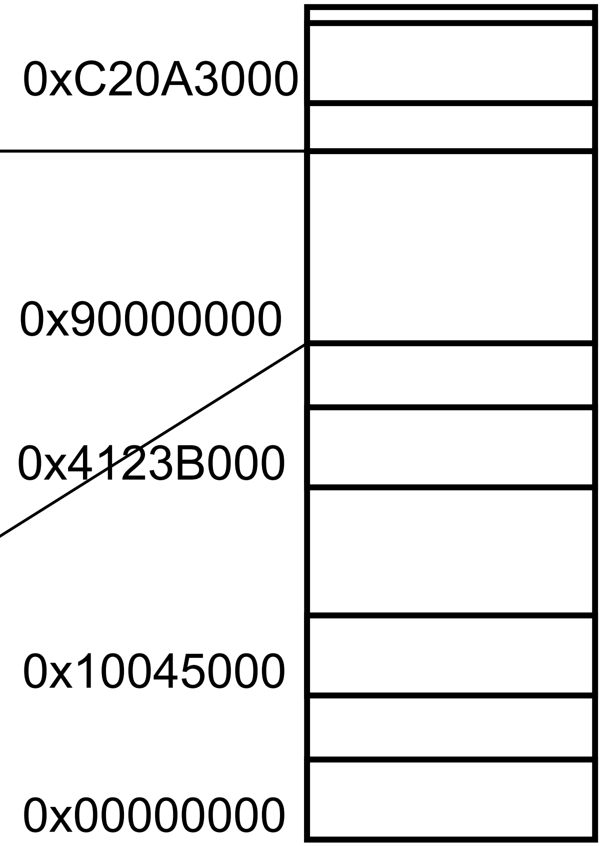
- Pages used recently mostly likely to be used again

More Meta-Data:

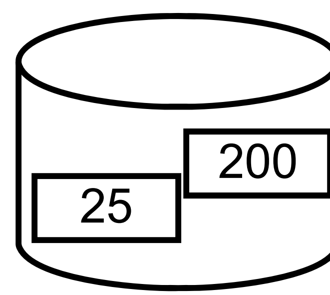
- Dirty Bit, Recently Used, *etc.*
- OS may access this meta-data to choose a victim

Paging

V	R	W	X	D	*Physical Page Number*
0					--
1	1	0	1	0	0x10045
0					--
0					--
0				0	disk sector 200
0				0	disk sector 25
1	1	1	0	1	0x00000
0					--



Example: accessing address beginning with 0x00003 (PageTable[3]) results in a Page Fault which will page the data in from disk sector 200



Page Fault

Valid bit in Page Table = 0

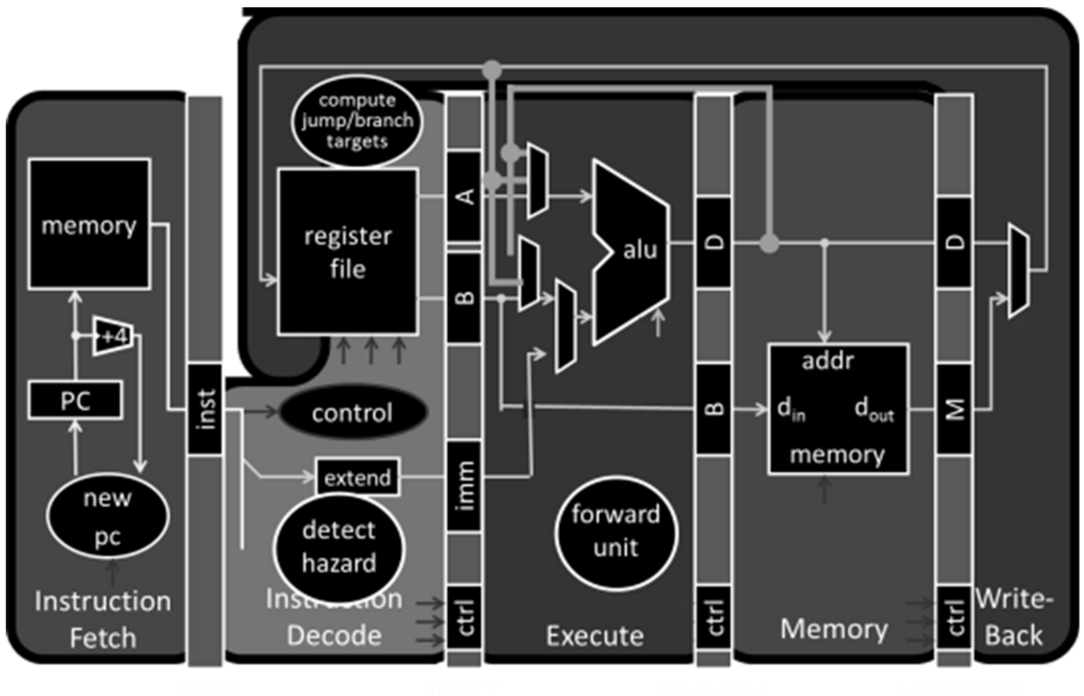
→ means page is not in memory

OS takes over:

- Choose a physical page to replace
  - **“Working set”**: refined LRU, tracks page usage
- If dirty, write to disk
- Read missing page from disk
  - Takes so long (~10ms), OS schedules another task

Performance-wise page faults are *really* bad!

RISC-V Processor Milestone



RISC-V Processor Milestone Celebration

Virtual Memory Agenda

What is Virtual Memory?

How does Virtual memory Work?

- Address Translation
- Overhead
- Paging
- Performance

Watch Your Performance Tank!

For *every instruction*:

- MMU translates address (virtual → physical)
  - Uses PTBR to find Page Table in memory
  - Looks up entry for that virtual page
- Fetch the instruction using physical address
  - Access Memory Hierarchy (I$ → L2 → Memory)

- Repeat at Memory stage for load/store insns
  - Translate address
  - **Now** you perform the load/store

Performance

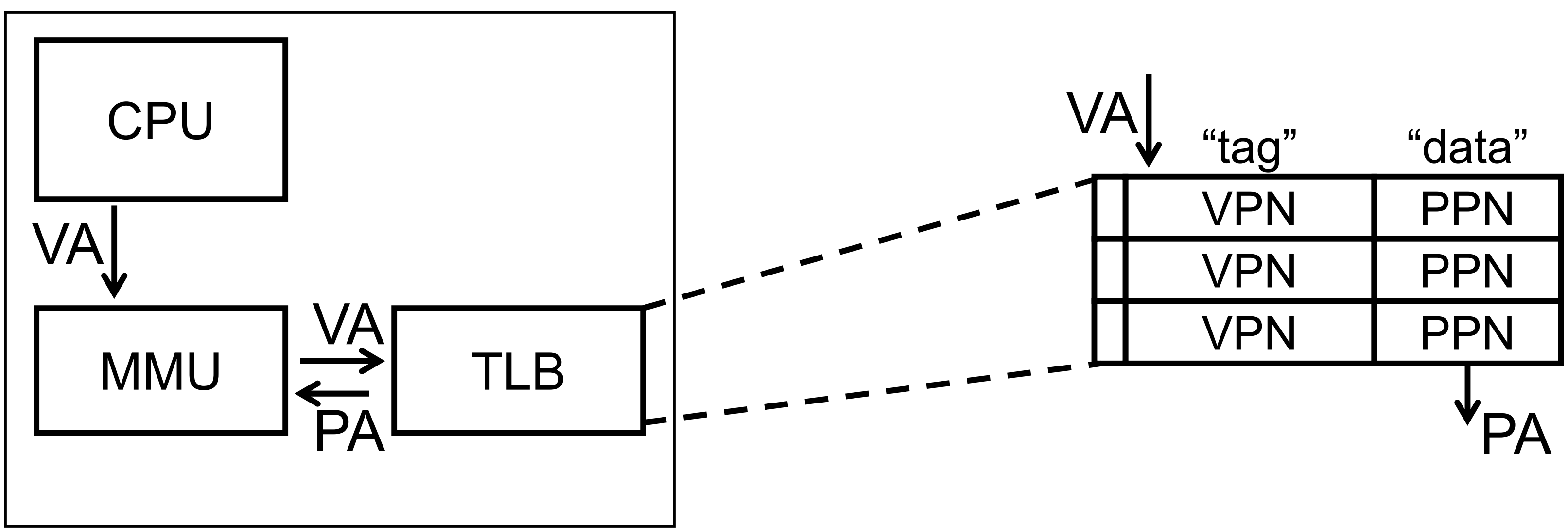
- Virtual Memory Summary
- PageTable for each process:
  - Page
    - Single-level (e.g. 4MB contiguous in physical memory)
    - or multi-level (e.g. less mem overhead due to page table),
    - …
  - every load/store translated to physical addresses
  - page table miss: load a swapped-out page and retry instruction, or kill program
- Performance?
  - terrible: memory is already slow
    translation makes it slower
- Solution?
  - A cache, of course

Next Goal

- How do we speedup address translation?

Translation Lookaside Buffer (TLB)

- Small, fast cache
- Holds VPN→PPN translations
- Exploits temporal locality in pagetable
- TLB Hit: huge performance savings
- TLB Miss: invoke TLB miss handler
    - *Put translation in TLB for later*



TLB Parameters

Typical

- very small (64 – 256 entries) → *very fast*
- fully associative, or at least set associative

Example: Intel Nehalem TLB

- 128-entry L1 Instruction TLB, 4-way LRU
- 64-entry L1 Data TLB, 4-way LRU
- 512-entry L2 Unified TLB, 4-way LRU

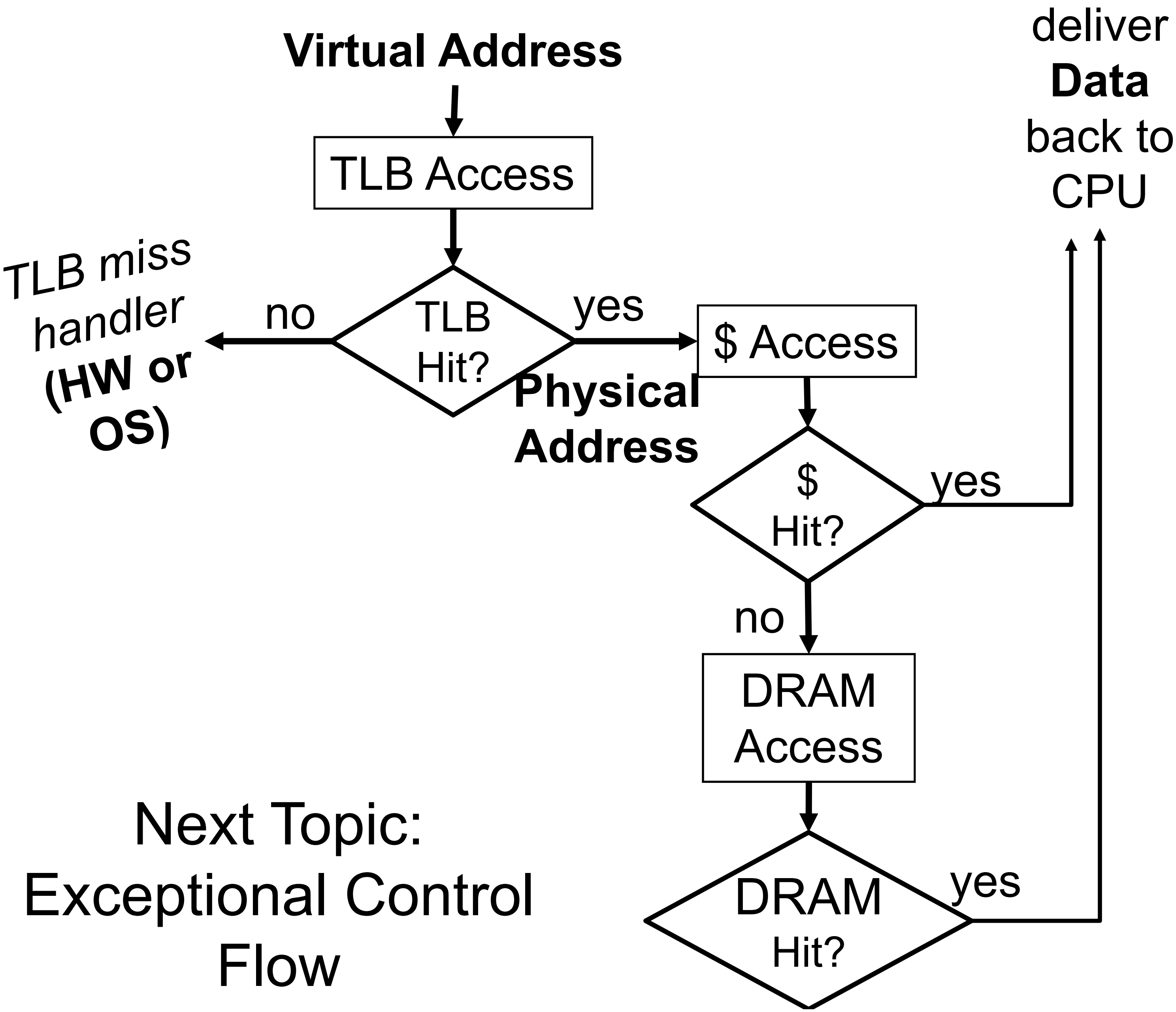
TLB to the Rescue!

For every instruction:

- Translate the address (virtual → physical)
  - CPU checks TLB
  - That failing, walk the Page Table
    - Use PTBR to find Page Table in memory
    - Look up entry for that virtual page
    - Cache the result in the TLB
- Fetch the instruction using physical address
  - Access Memory Hierarchy (I$ → L2 → Memory)

- Repeat at Memory stage for load/store insns
  - CPU checks TLB, translate if necessary
  - **Now** perform load/store

Translation in Action



Next Topic:
Exceptional Control
Flow

Takeaways

Need a map to translate a “fake” virtual address (from process) to a “real” physical Address (in memory).

The map is a **Page Table*:*** `ppn = PageTable[vpn]`

A page is constant size block of virtual memory. Often ~4KB to reduce the number of entries in a PageTable.

Page Table can enforce Read/Write/Execute permissions on a per page basis. Can allocate memory on a per page basis. Also need a valid bit, and a few others.

Space overhead due to Page Table is significant.
Solution: another level of indirection!
Two-level of Page Table significantly reduces overhead.

Time overhead due to Address Translations also significant.
Solution: caching! **Translation Lookaside Buffer (TLB)** acts as a cache for the Page Table and significantly improves performance.