

# Virtual Memory

**Hakim Weatherspoon**  
**CS 3410, Spring 2012**  
Computer Science  
Cornell University

P & H Chapter 5.4 (up to TLBs)

# Administrivia

---

Lab3 is due next Monday

HW5 is due *next* Tuesday

- Download updated version. Use updated version.

# Goals for Today

---

## Virtual Memory

- Address Translation
  - Pages, page tables, and memory mgmt unit
- Paging
- Role of Operating System
  - Context switches, working set, shared memory
- Performance
  - How slow is it
  - Making virtual memory fast
  - Translation lookaside buffer (TLB)
- Virtual Memory Meets Caching

---

# Virtual Memory

# 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

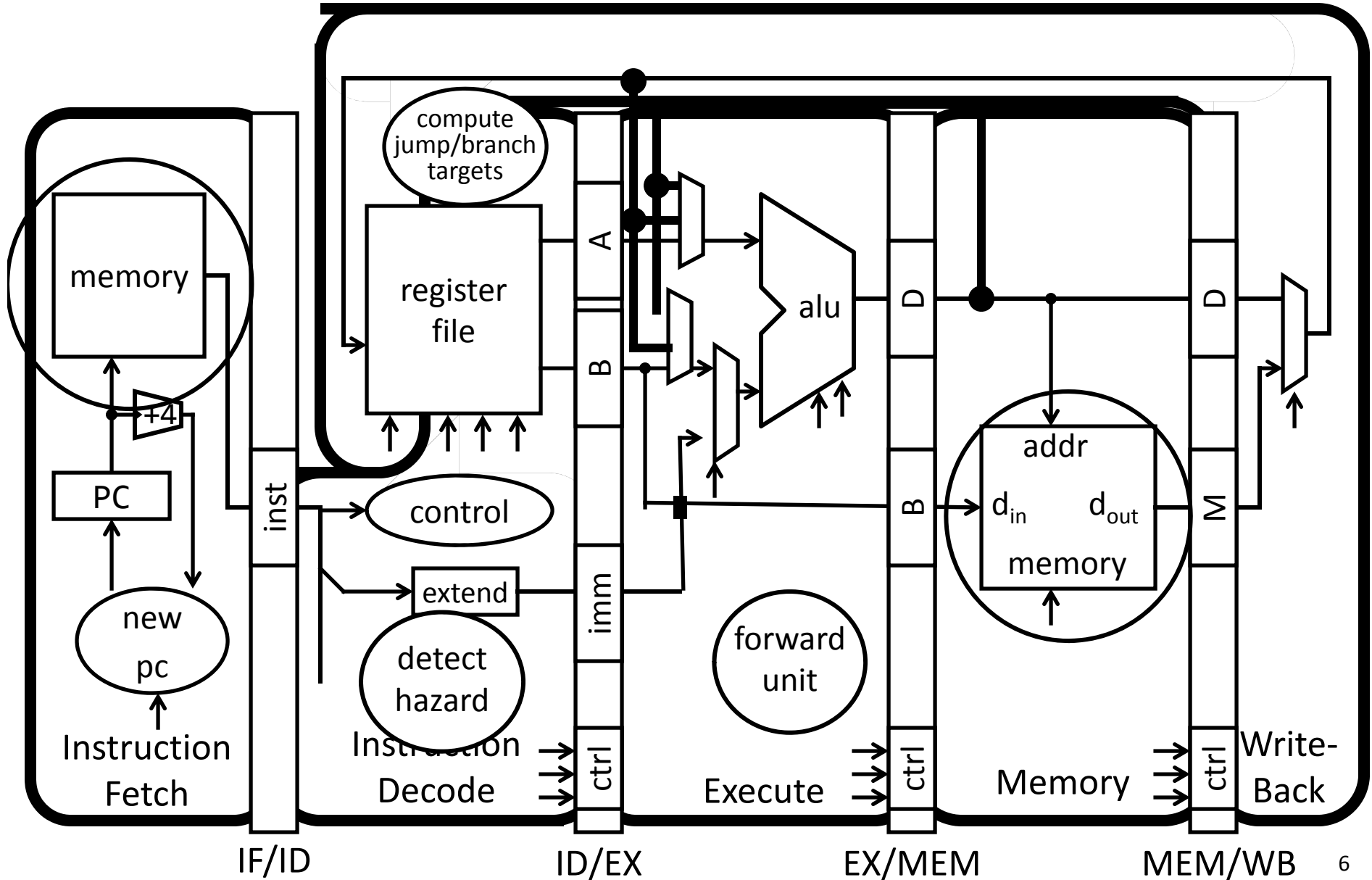
Many cores per processor (multi-core)

or many processors (multi-processor)

- Multiple programs run *simultaneously*

# Big Picture: (Virtual) Memory

Memory: big & slow vs Caches: small & fast





# Processor & Memory

CPU address/data bus...

... routed through caches

... to main memory

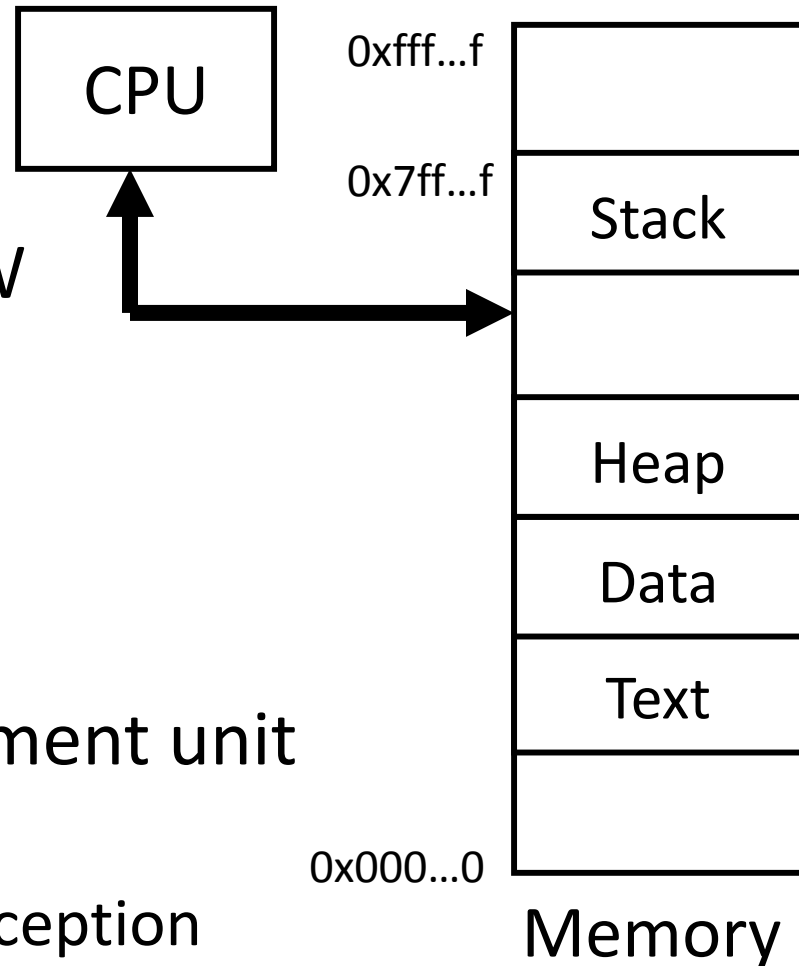
- Simple, fast, but...

Q: What happens for LW/SW to an invalid location?

- 0x00000000 (NULL)
- uninitialized pointer

A: Need a memory management unit (MMU)

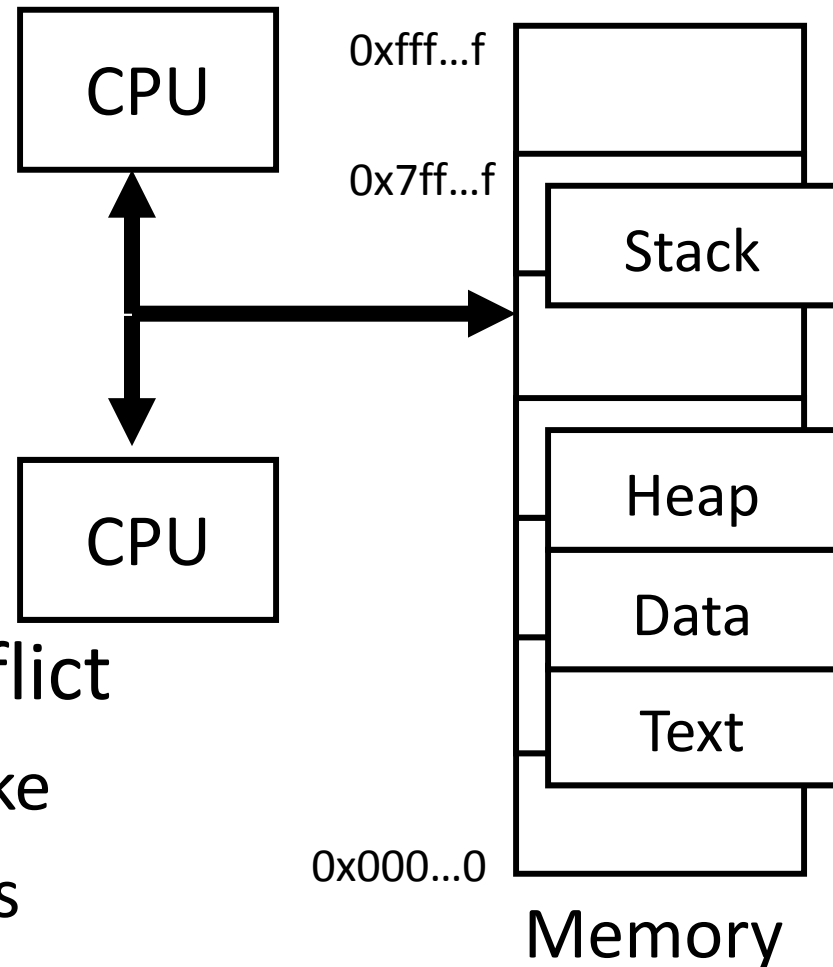
- Throw (and/or handle) an exception





# Multiple Processes

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

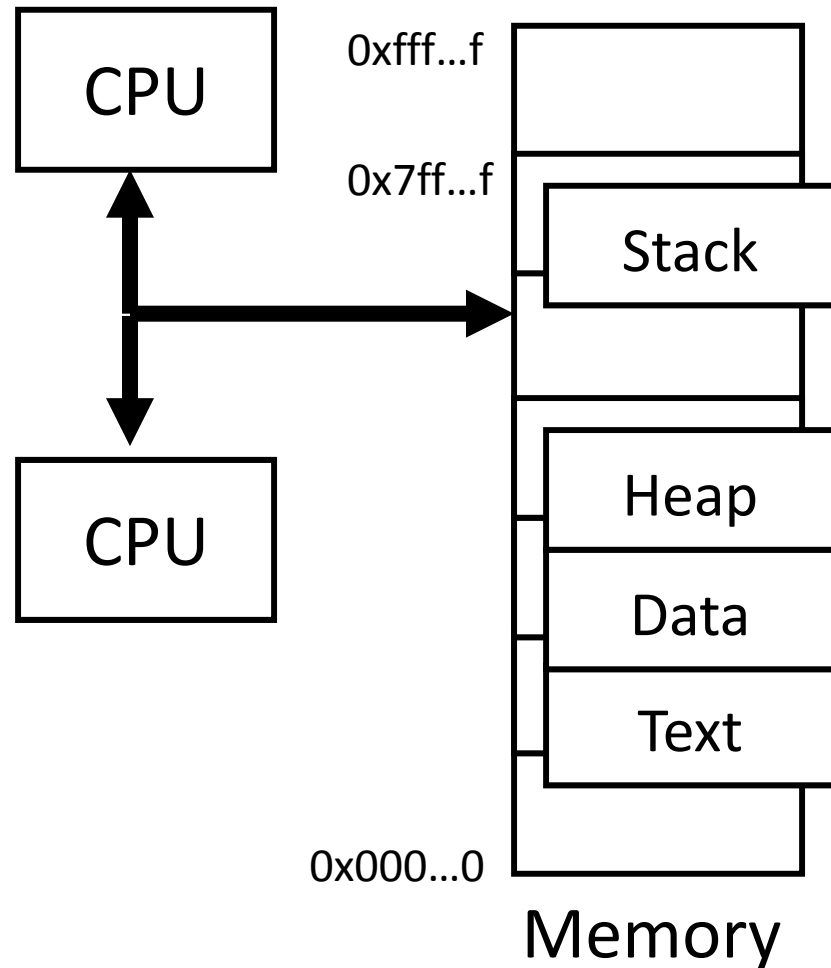


A: The addresses will conflict

- Even though, CPUs may take turns using memory bus

# Multiple Processes

Q: Can we relocate second program?

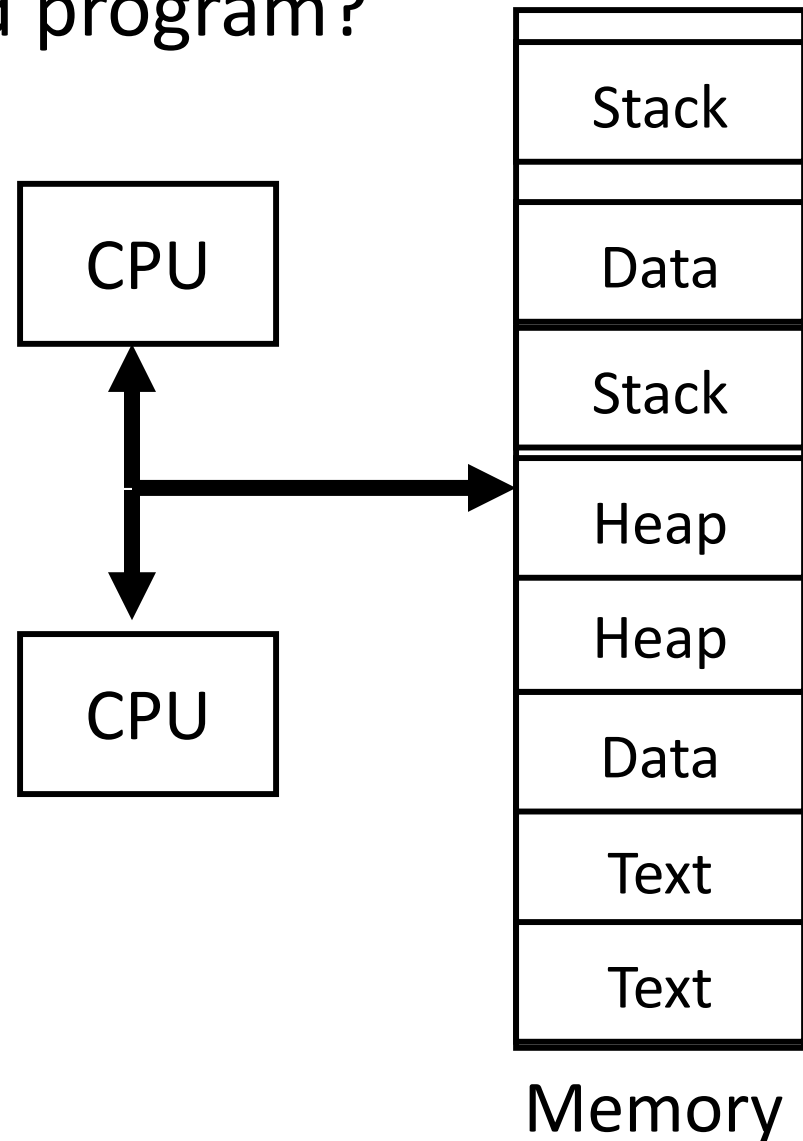


# Solution? Multiple processes/processors

Q: Can we relocate second program?

A: Yes, but...

- What if they don't fit?
- What if not contiguous?
- Need to recompile/relink?
- ...



---

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

- David Wheeler*
- or, Butler Lampson*
- or, Leslie Lamport*
- or, Steve Bellovin*

# Virtual Memory

---

## Virtual Memory: A Solution for All Problems

Each process has its own virtual address space

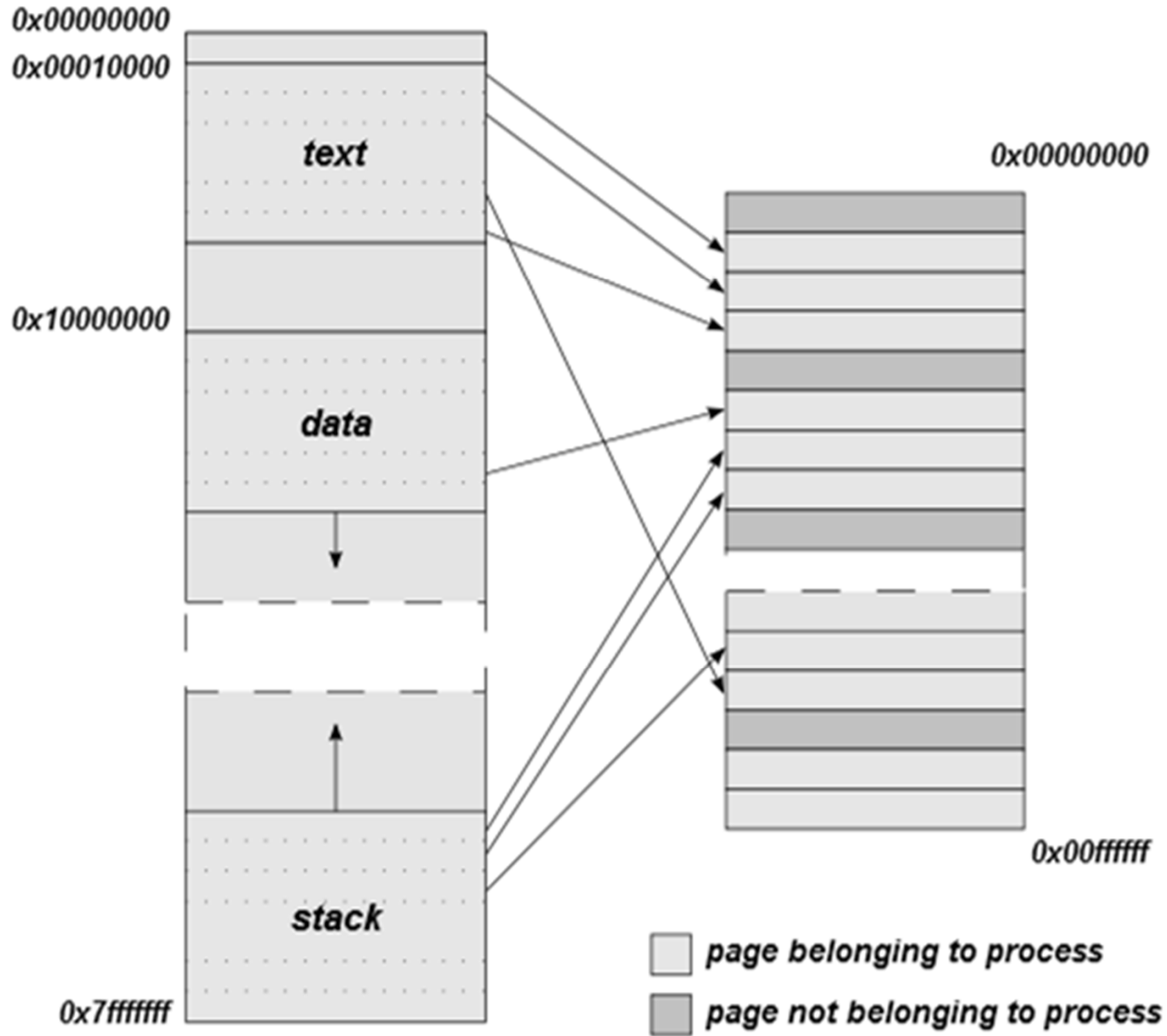
- Programmer can code as if they own all of memory

On-the-fly at runtime, for each memory access

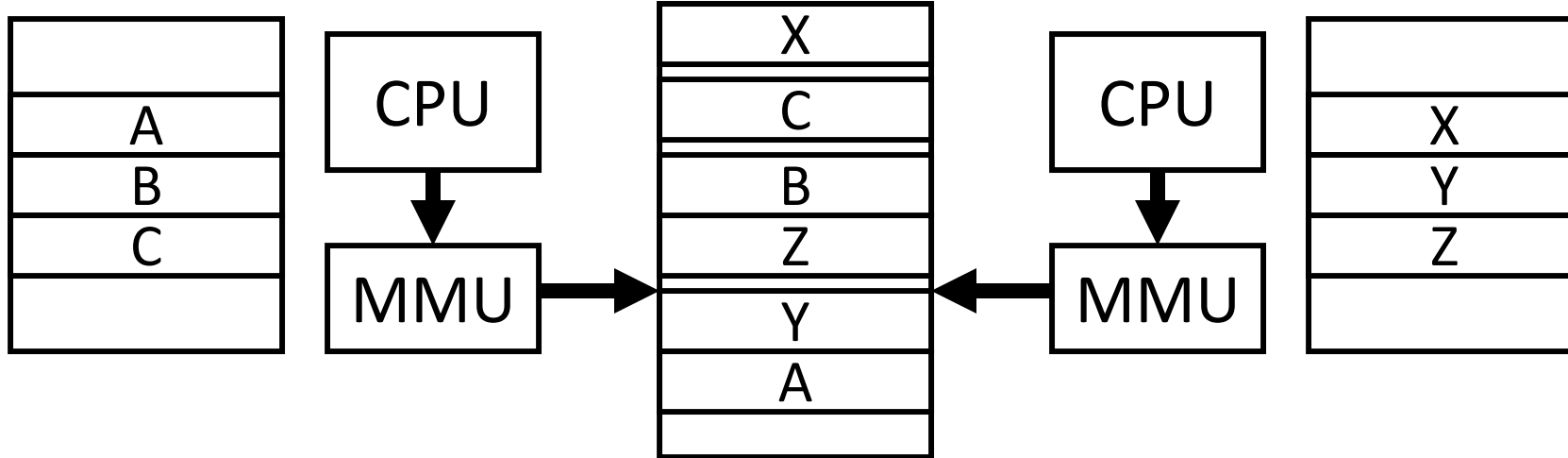
- all access is *indirect* through a virtual address
- translate fake virtual address to a real physical address
- redirect load/store to the physical address

### Virtual address space

### Physical address space



# Address Space



Programs load/store to virtual addresses

Actual memory uses physical addresses

Memory Management Unit (MMU)

- Responsible for translating on the fly
- Essentially, just a big array of integers:  
`paddr = PageTable[vaddr];`

# Virtual Memory Advantages

## Advantages

### Easy relocation

- Loader puts code anywhere in physical memory
- Creates 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...



---

Address Translation  
Pages, Page Tables, and  
the Memory Management Unit (MMU)

# Address Translation

Attempt #1: How does MMU translate addresses?

```
paddr = PageTable[vaddr];
```

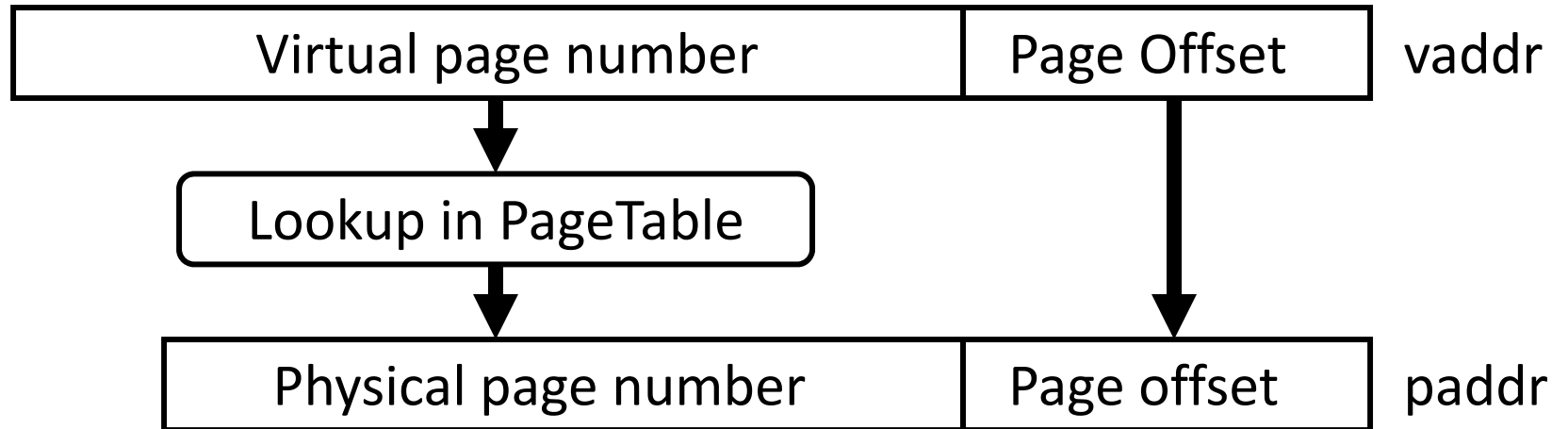
Granularity?

- Per word...
- Per block...
- Variable.....

Typical:

- 4KB – 16KB pages
- 4MB – 256MB jumbo pages

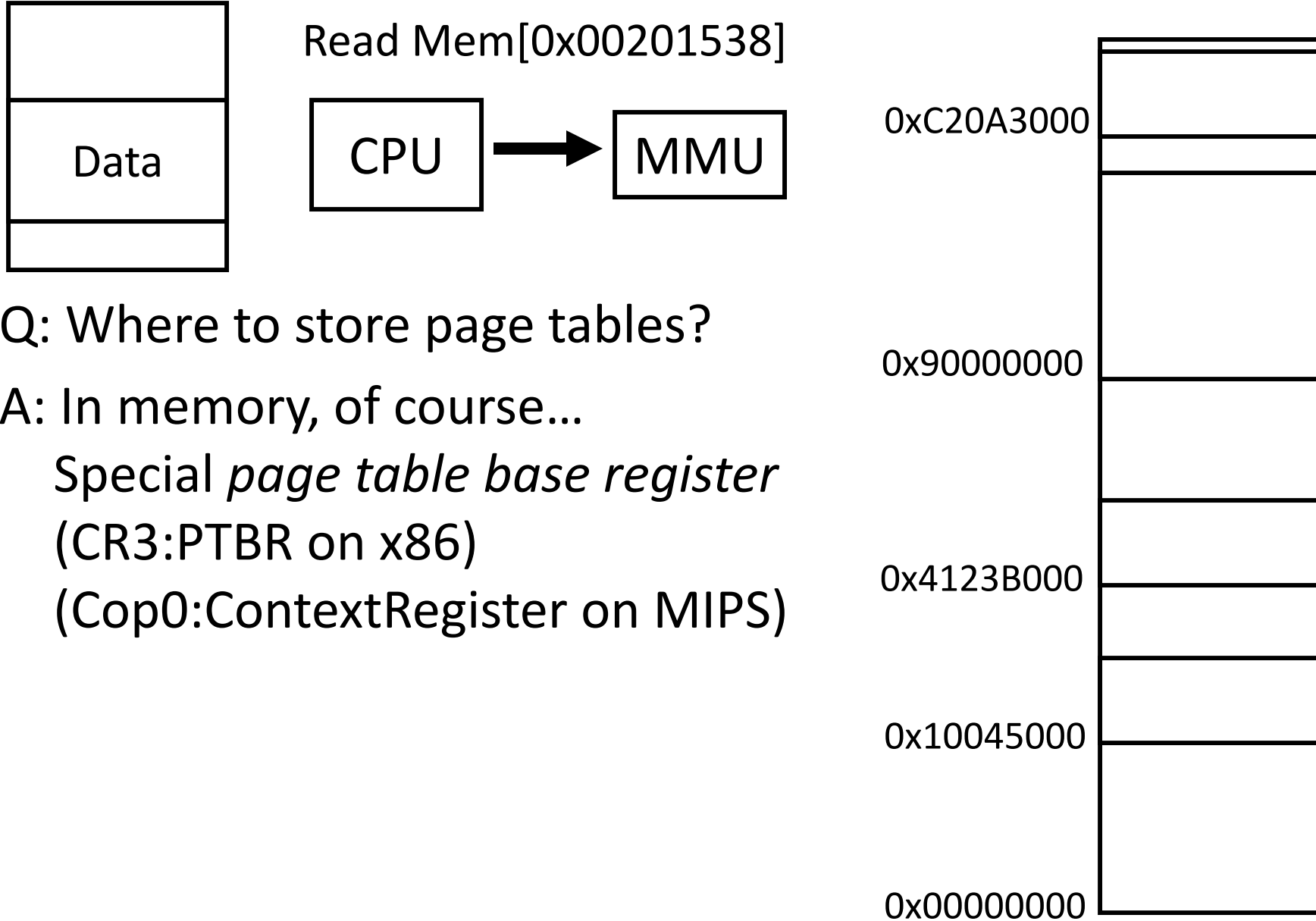
# Address Translation



**Attempt #1:** For any access to virtual address:

- Calculate virtual page number and page offset
- Lookup physical page number at `PageTable[vpn]`
- Calculate physical address as `ppn:offset`

# Simple PageTable



Q: Where to store page tables?

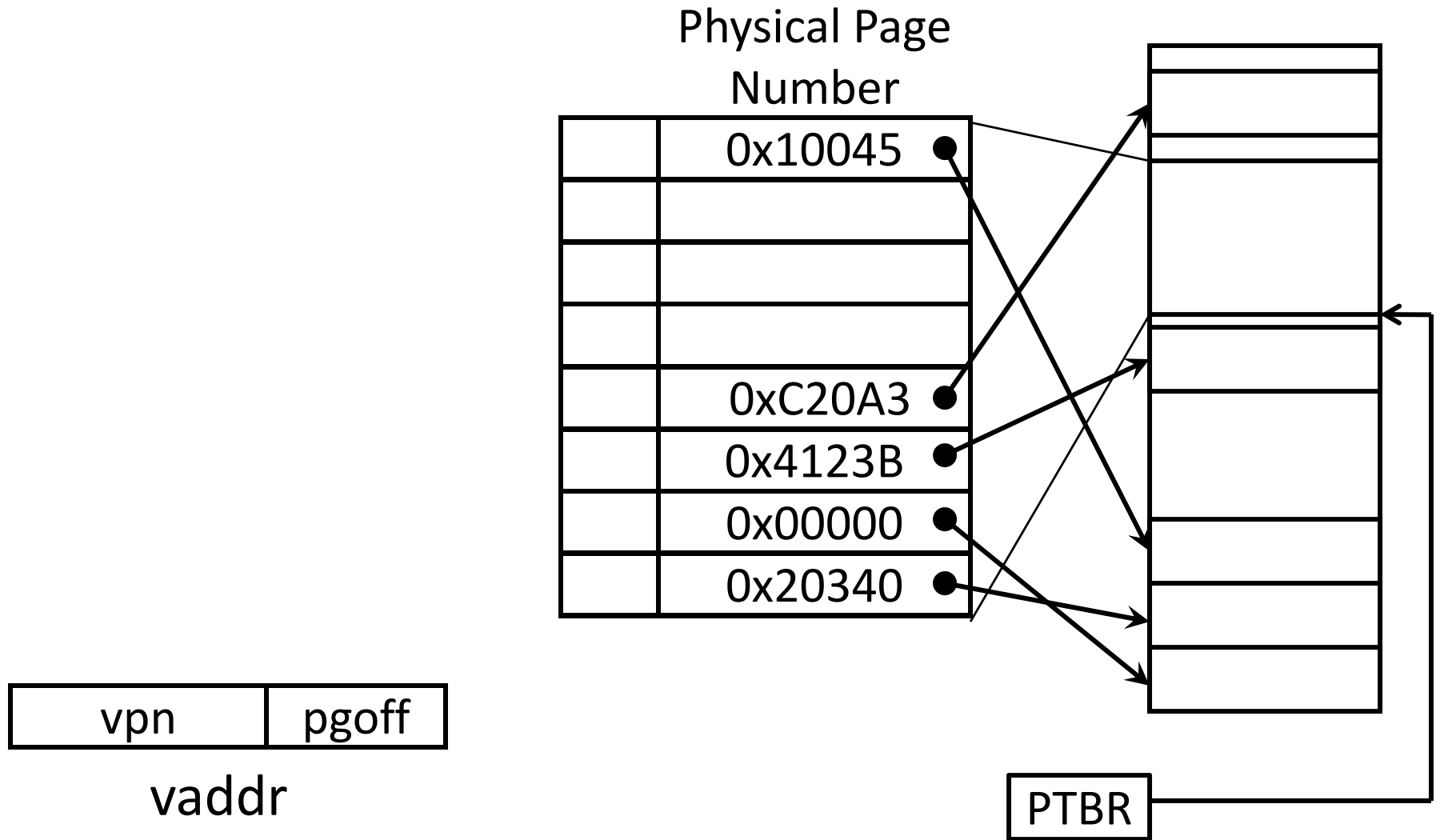
A: In memory, of course...

*Special page table base register*

(CR3:PTBR on x86)

(Cop0:ContextRegister on MIPS)

# Summary



# Page Size Example

---

Overhead for VM Attempt #1 (example)

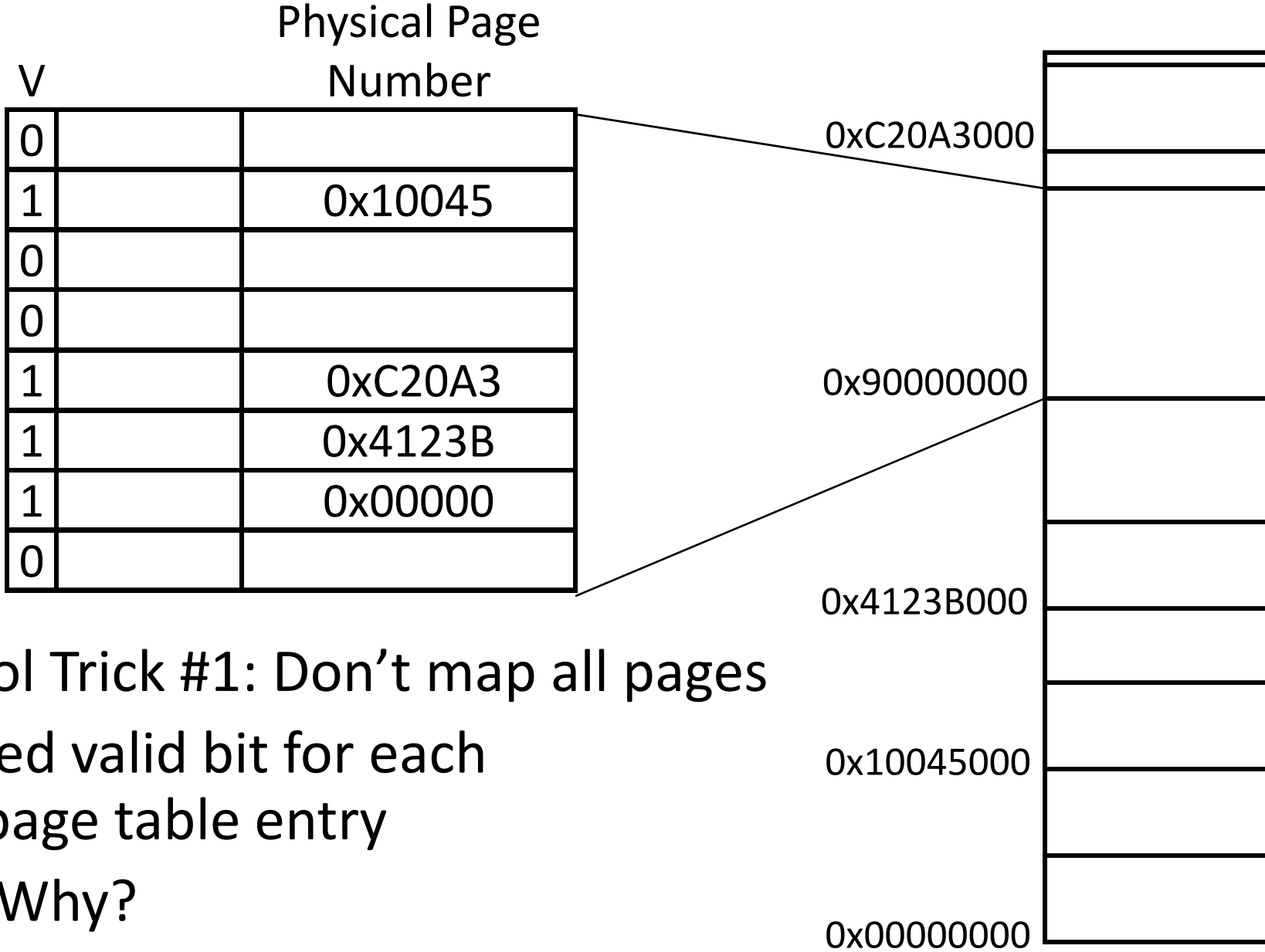
Virtual address space (for each process):

- total memory:  $2^{32}$  bytes = 4GB
- page size:  $2^{12}$  bytes = 4KB
- entries in PageTable?
- size of PageTable?

Physical address space:

- total memory:  $2^{29}$  bytes = 512MB
- overhead for 10 processes?

# Invalid Pages



Cool Trick #1: Don't map all pages

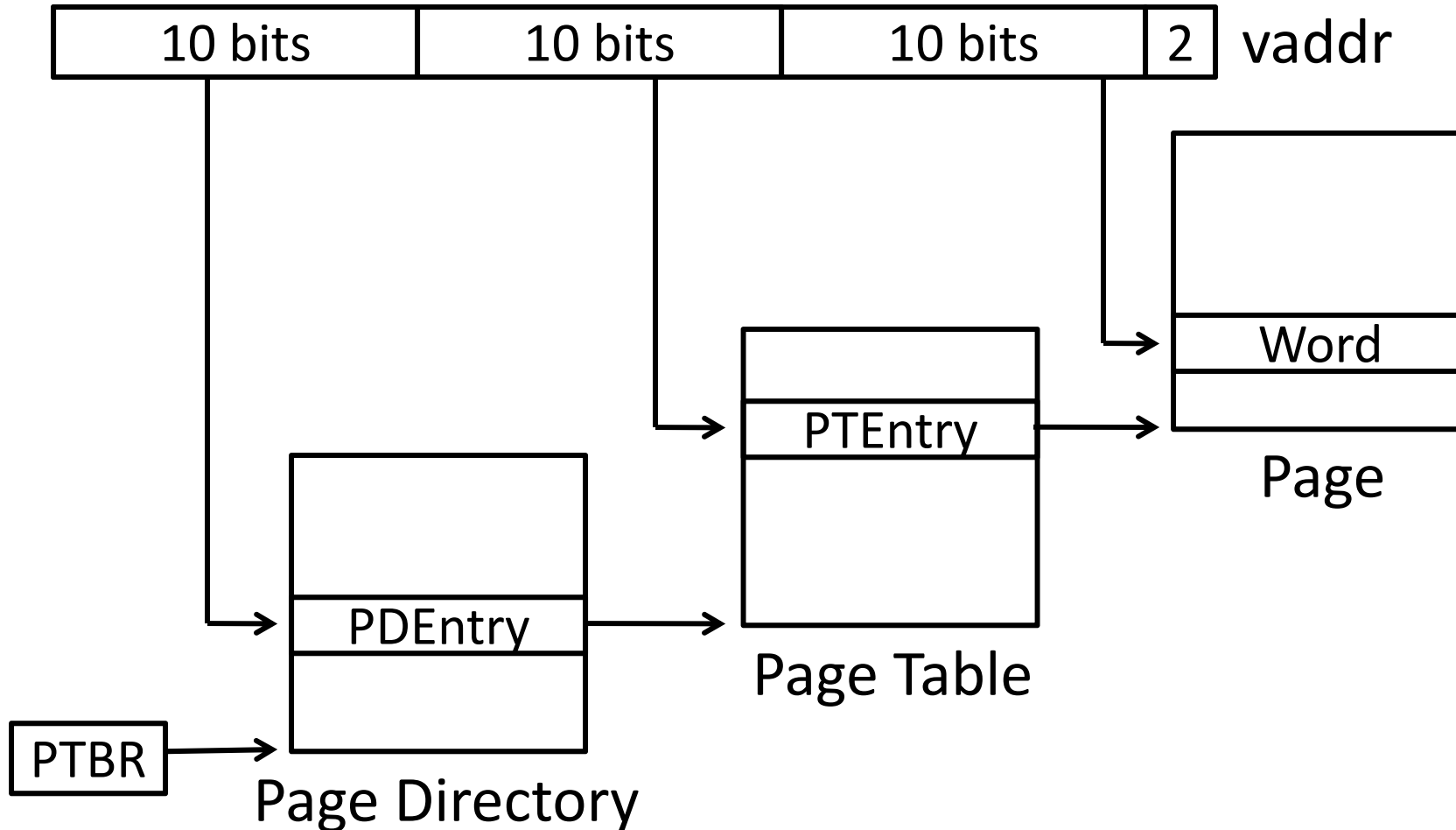
Need valid bit for each page table entry

Q: Why?

# Beyond Flat Page Tables

Assume most of PageTable is empty

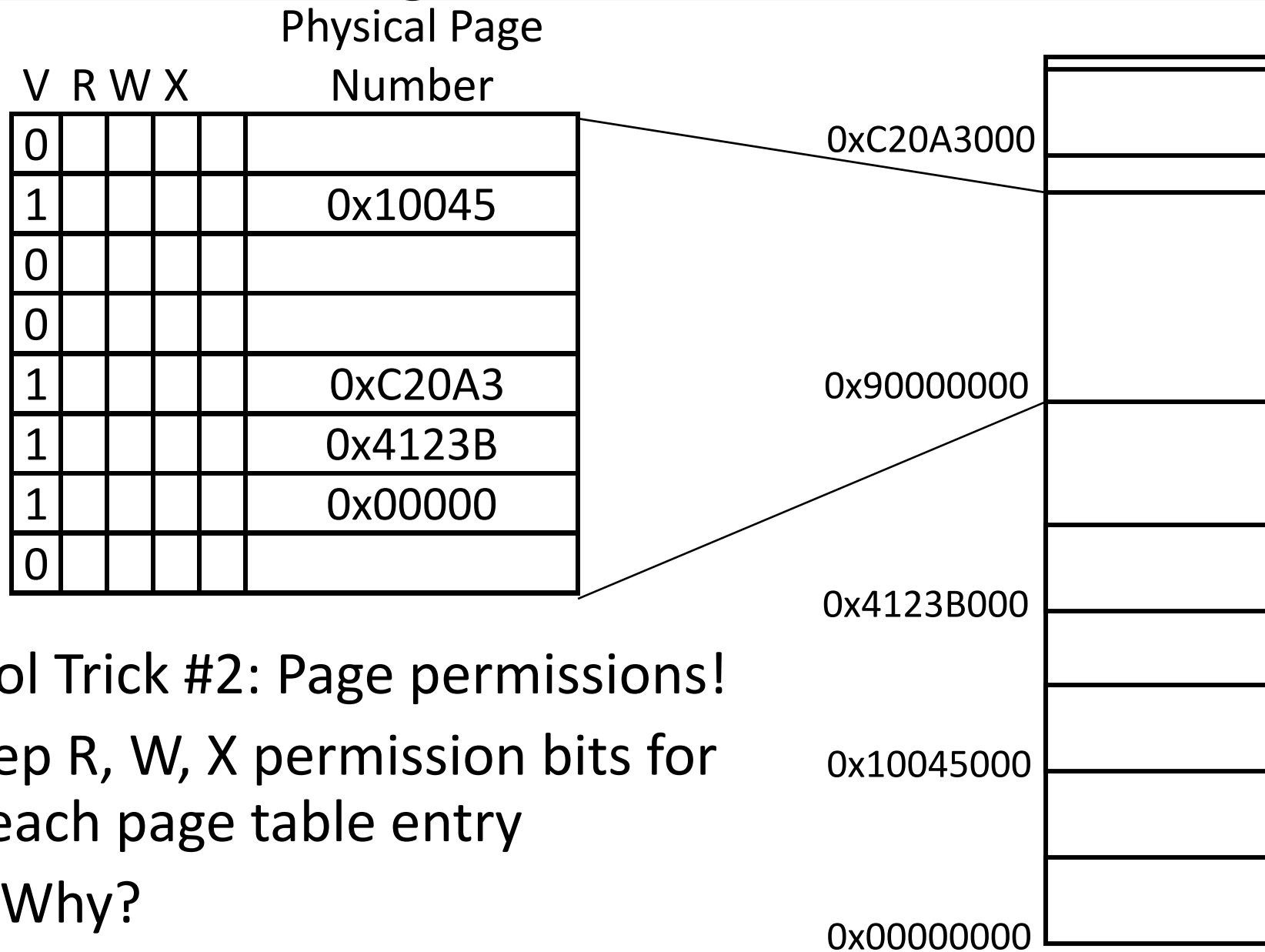
How to translate addresses? Multi-level PageTable



\* x86 does exactly this



# Page Permissions

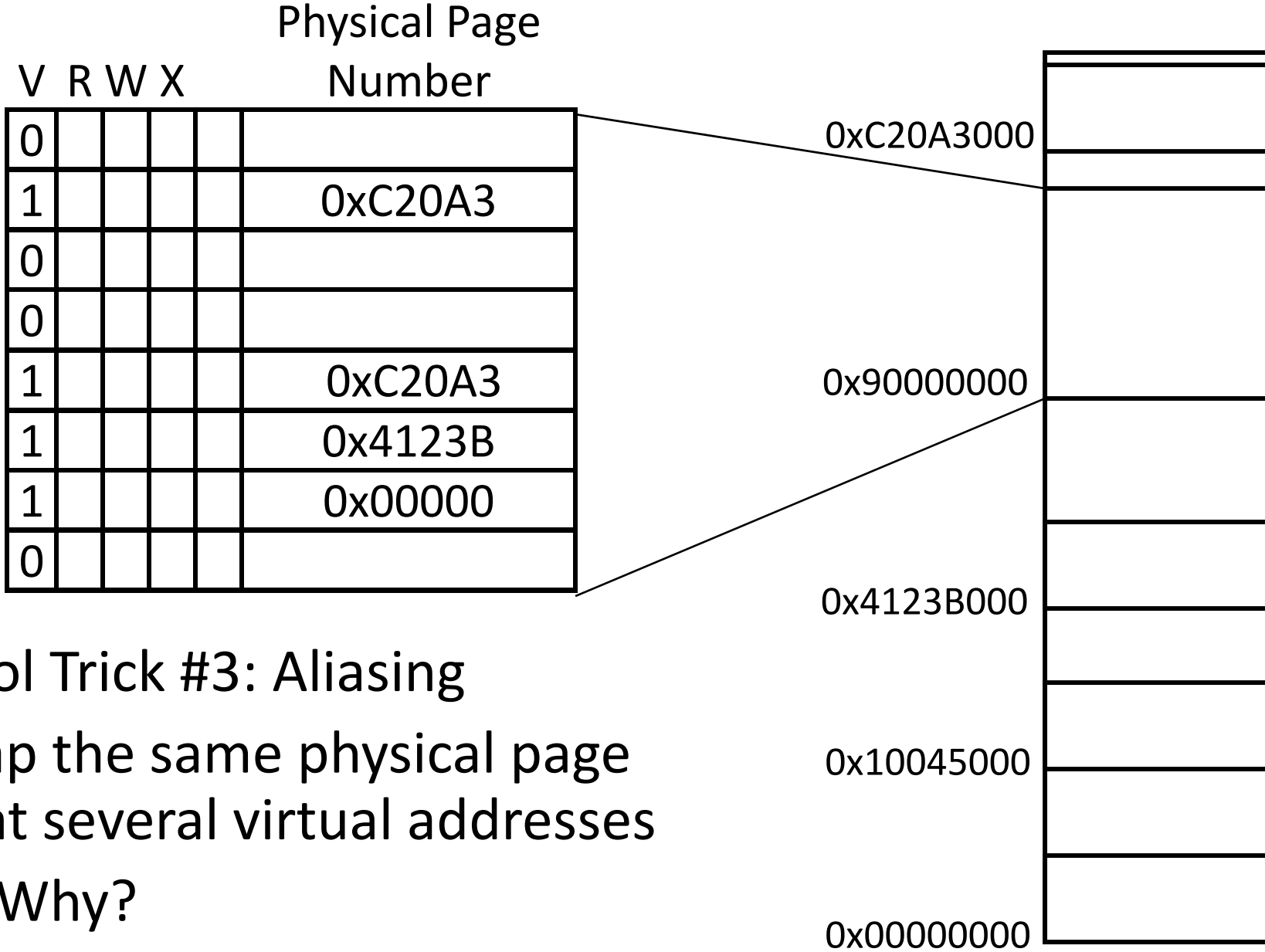


Cool Trick #2: Page permissions!

Keep R, W, X permission bits for each page table entry

Q: Why?

# Aliasing



Cool Trick #3: Aliasing

Map the same physical page  
at several virtual addresses

Q: Why?

---

# Paging

# Paging

Can we run process larger than physical memory?

- The “virtual” in “virtual memory”

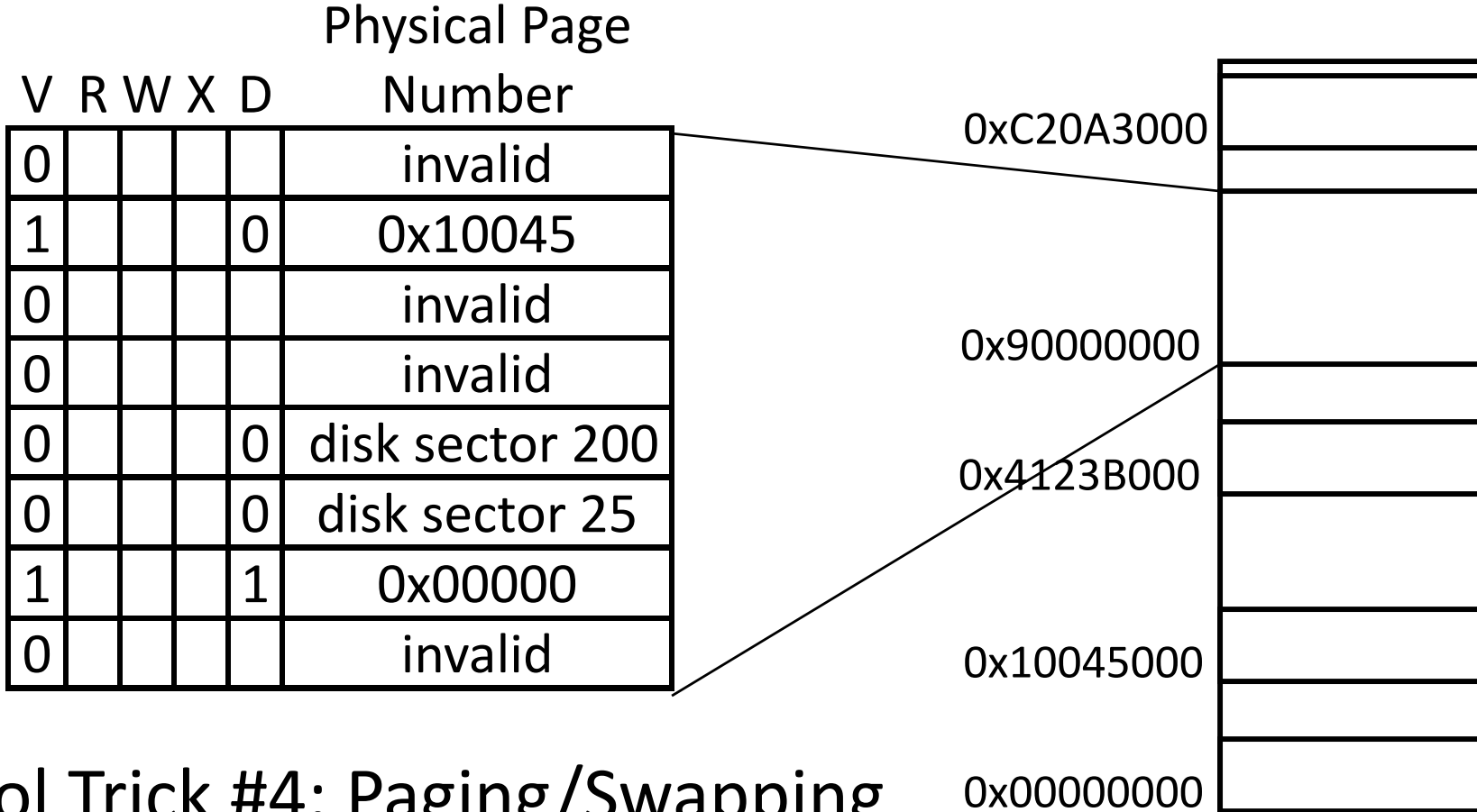
View memory as a “cache” for secondary storage

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

Assumes Temporal/Spatial Locality

- Pages used recently most likely to be used again soon

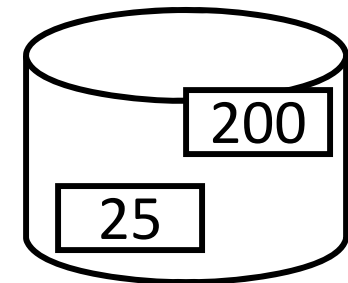
# Paging



Cool Trick #4: Paging/Swapping

Need more bits:

Dirty, RecentlyUsed, ...



# Summary

---

## Virtual Memory

- Address Translation
  - Pages, page tables, and memory mgmt unit
- Paging

## Next time

- Role of Operating System
  - Context switches, working set, shared memory
- Performance
  - How slow is it
  - Making virtual memory fast
  - Translation lookaside buffer (TLB)
- Virtual Memory Meets Caching