Virtual Memory

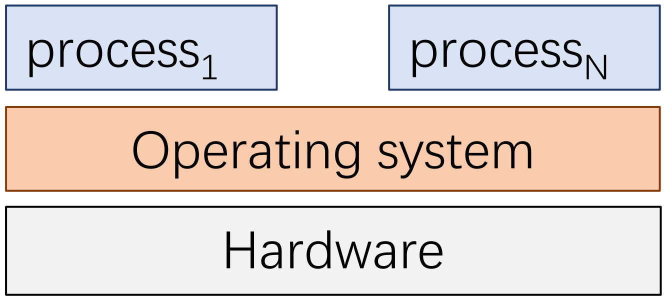
Xuhao Chen

cxh@mit.edu

Massachusetts Institute of Technology

CSAIL

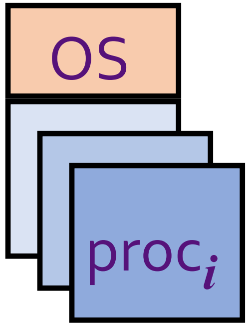
Reminder: Operating Systems



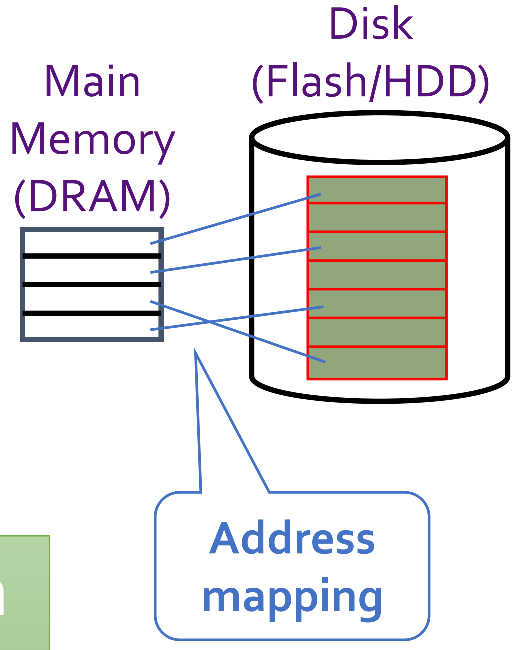
- Goals of OS:
  - **Protection and Privacy**: Process isolation
  - **Abstraction**: Hide away details of underlying hardware
  - **Resource Management**: Controls how processes share hardware resources (CPU, memory, disk, etc.)
- Key enabling technologies:
  - User mode + supervisor mode
  - Exceptions to safely transition into supervisor mode
  - **Virtual Memory** to abstract the storage resources of the machine

Virtual Memory (VM) Systems

Illusion of a large, private, uniform store

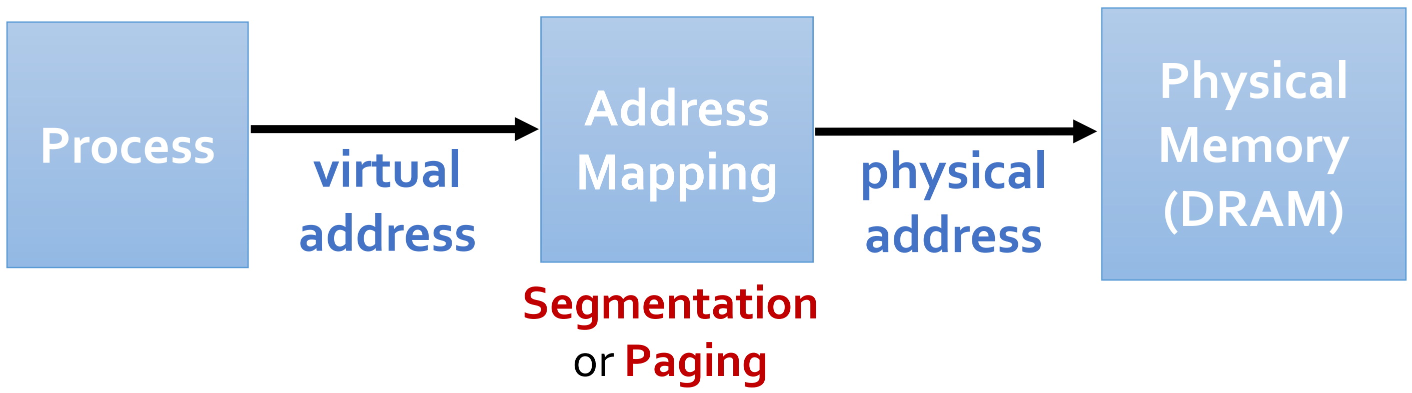


- Protection & Privacy
  - Each process has a private address space
- Demand Paging
  - Use main memory as a cache of disk
  - Enables running programs larger than main memory
  - Hides differences in machine configuration



The price of VM is **address translation** on each memory reference

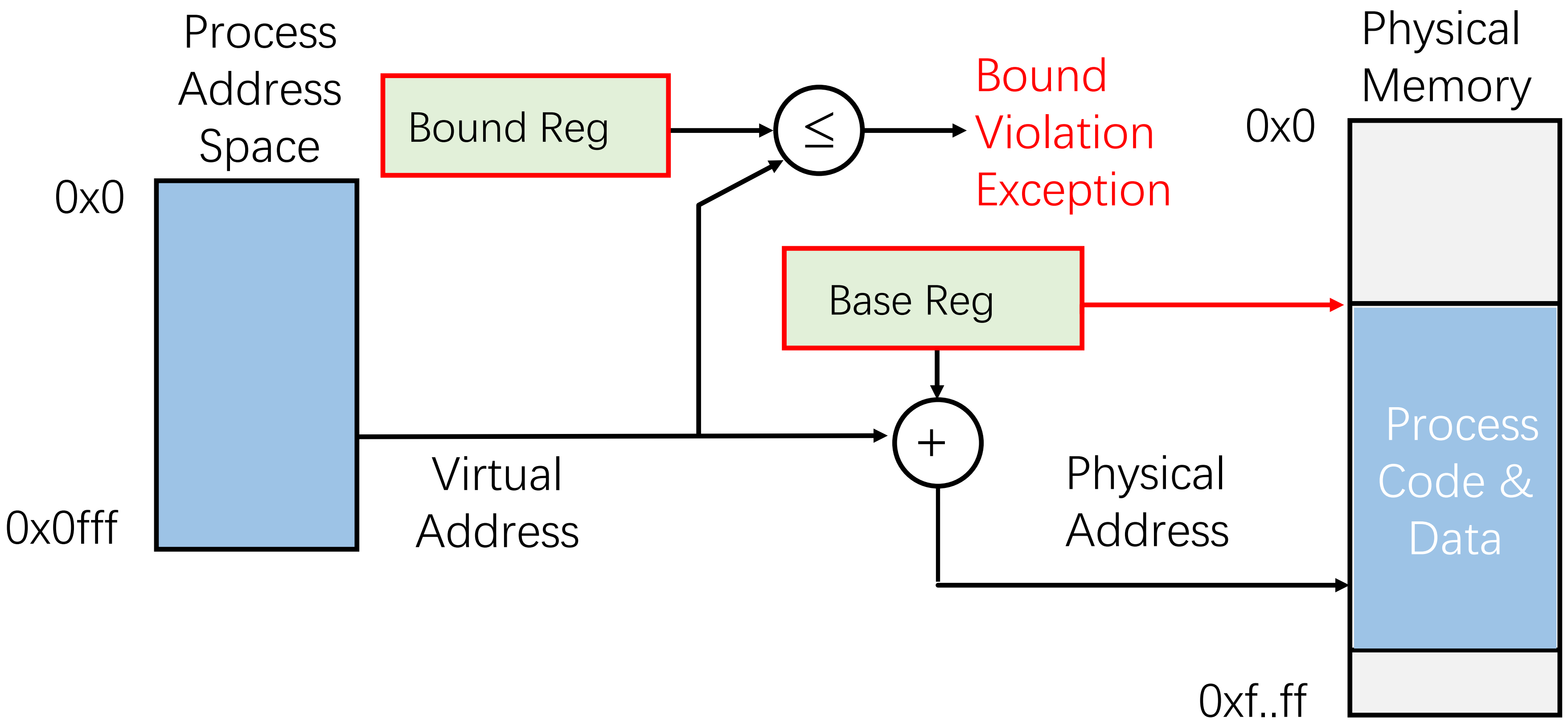
Virtual Address vs. Physical Address



- **Virtual address**
  - Address generated by the process
  - Specific to the process's private address space

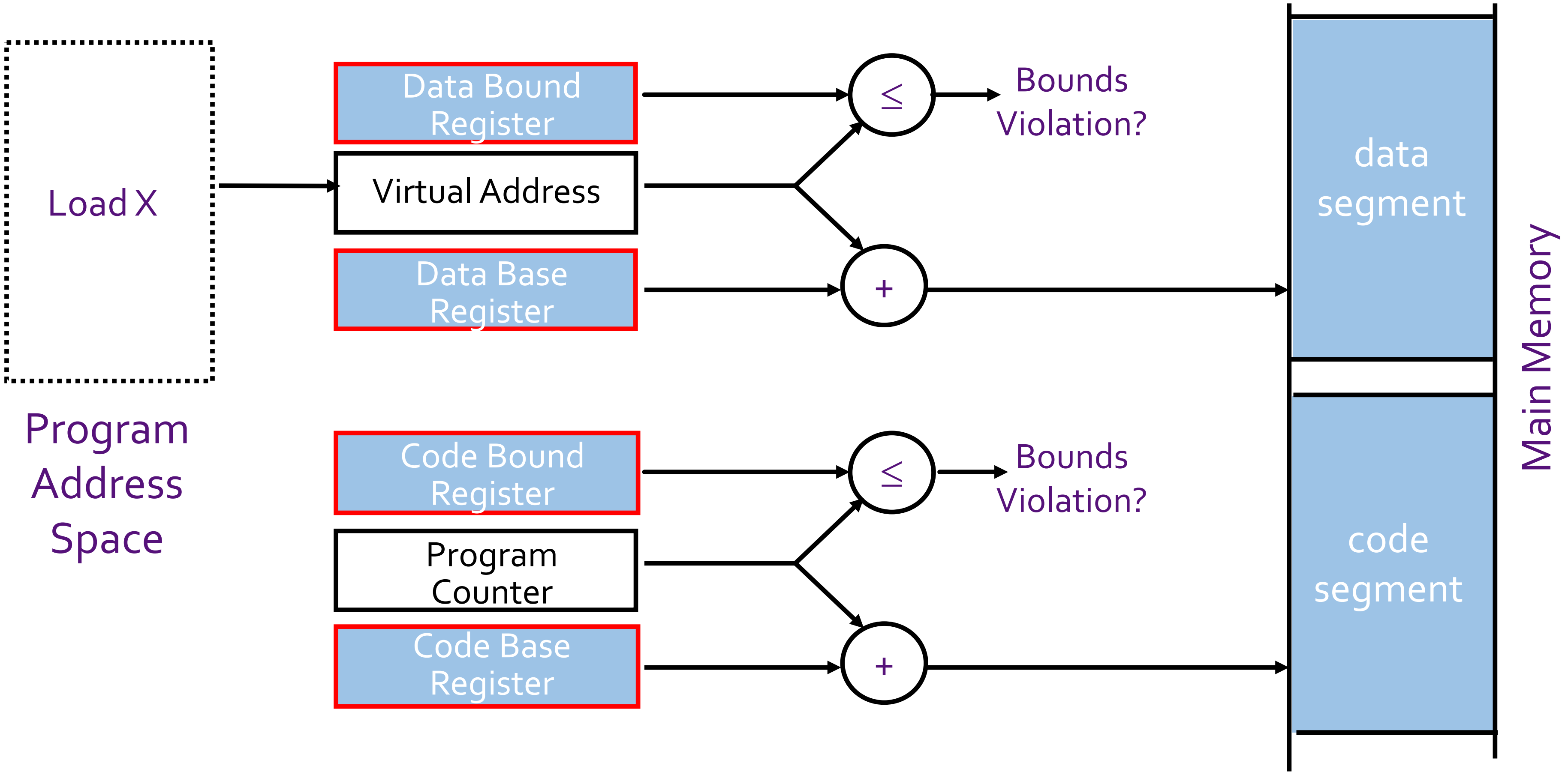
- **Physical address**
  - Address used to access physical (hardware) memory
  - **OS** specifies mapping of virtual addresses into physical addresses

Segmentation: *Base-and-Bound* Address Translation



- Each program's data is allocated in a contiguous segment of physical memory
- Physical address = Virtual Address + Segment Base
- Bound register provides safety and isolation
- Base and Bound registers should not be accessed by user programs (only accessible in supervisor mode)

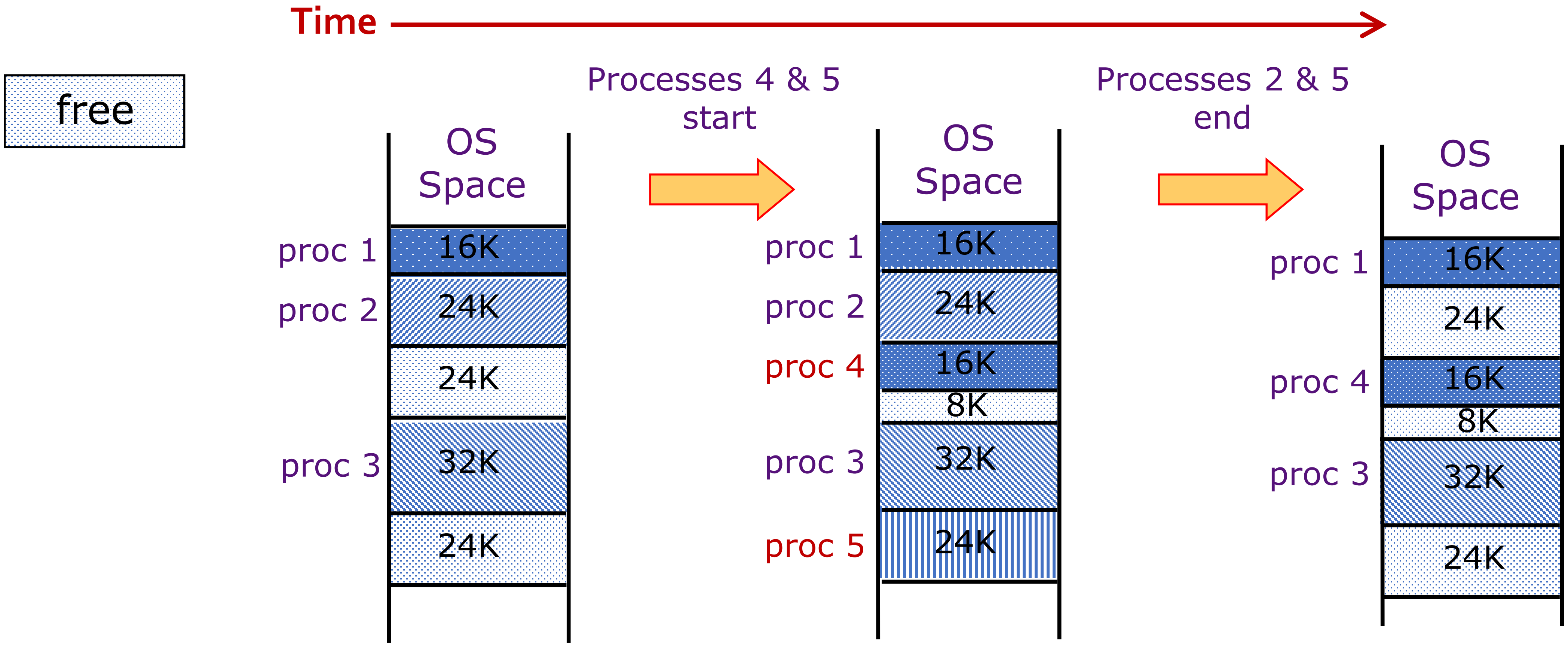
Separate Segments for Code and Data



Pros of this separation?

(1) **Prevents buggy program from overwriting code**
(2) **Multiple processes can share code segment**

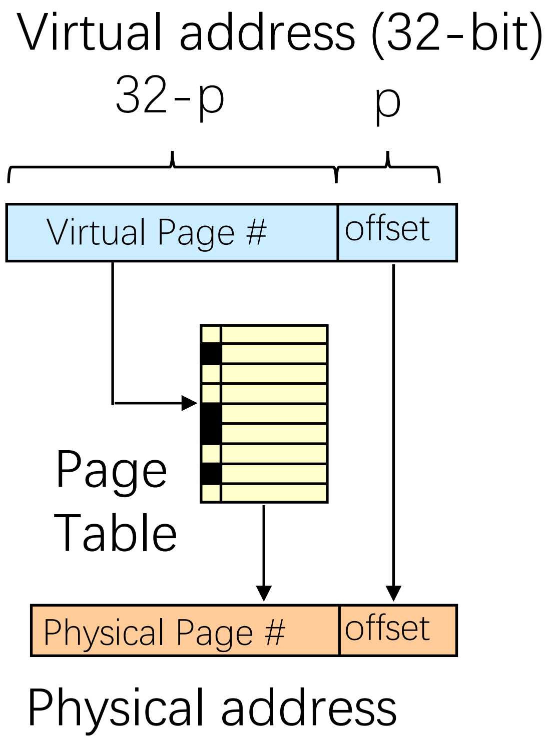
Memory Fragmentation



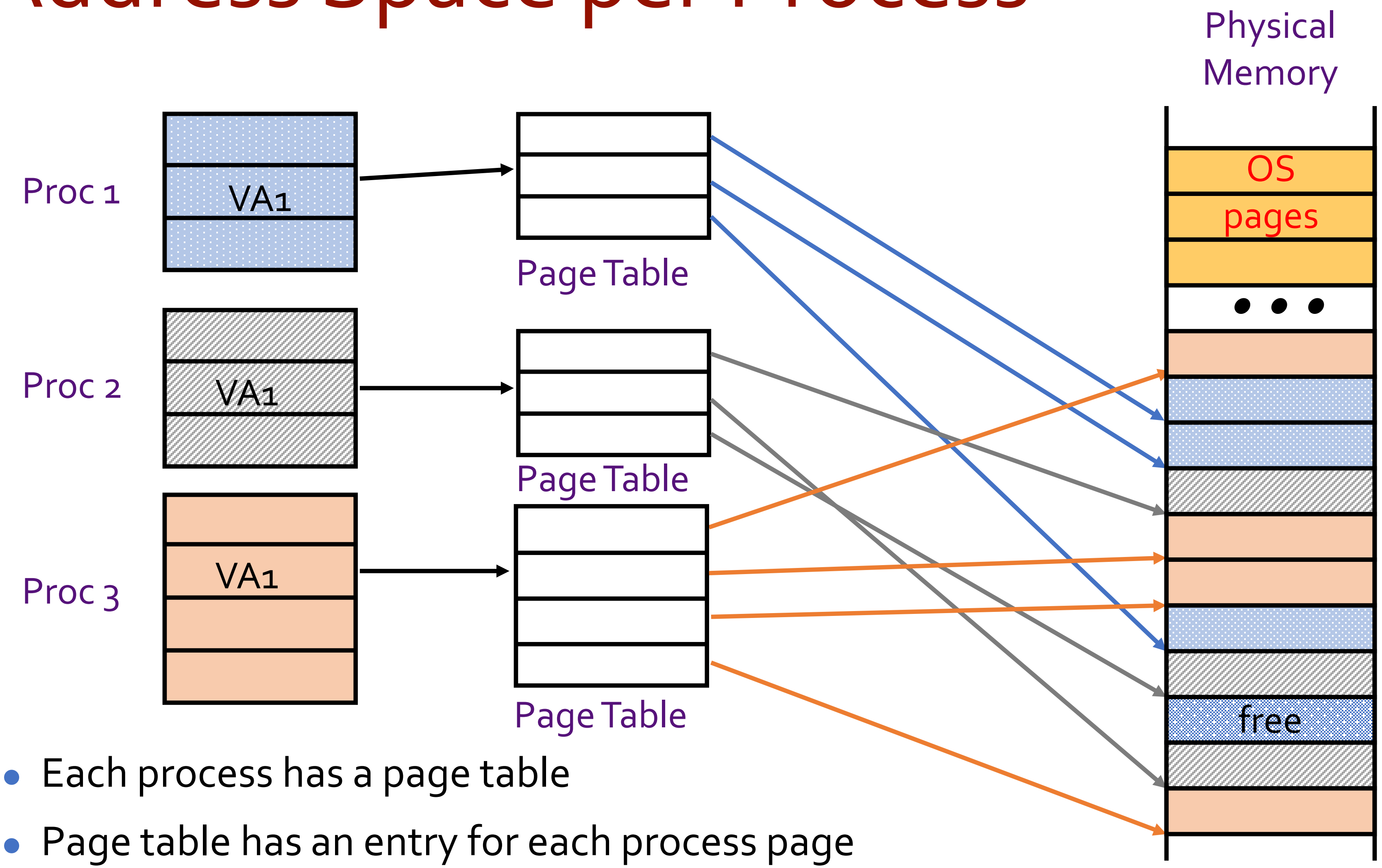
- As processes start and end, storage is "fragmented"
- At some point, segments have to be moved around to compact the free space

Paged Memory Systems

- Divide physical memory in *fixed-size blocks* called pages
  - Typical page size: 4KB
- Interpret each virtual address as a pair
  <virtual page number, offset>
- Use a Page Table to translate from virtual to physical page numbers
  - Page table contains the physical page number (i.e., starting physical address) for each virtual page number



Private Address Space per Process



- Each process has a page table
- Page table has an entry for each process page

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

Paging vs. Segmentation

- Pros of paging
  - Paging avoids fragmentation
  - Paging enables demand paging
    - Allows programs to use more VM than the machine's PM

- Cons of paging
  - Page tables are much larger than base & bound registers

- What if all the pages can't fit in DRAM?

- Where do we store the page tables?

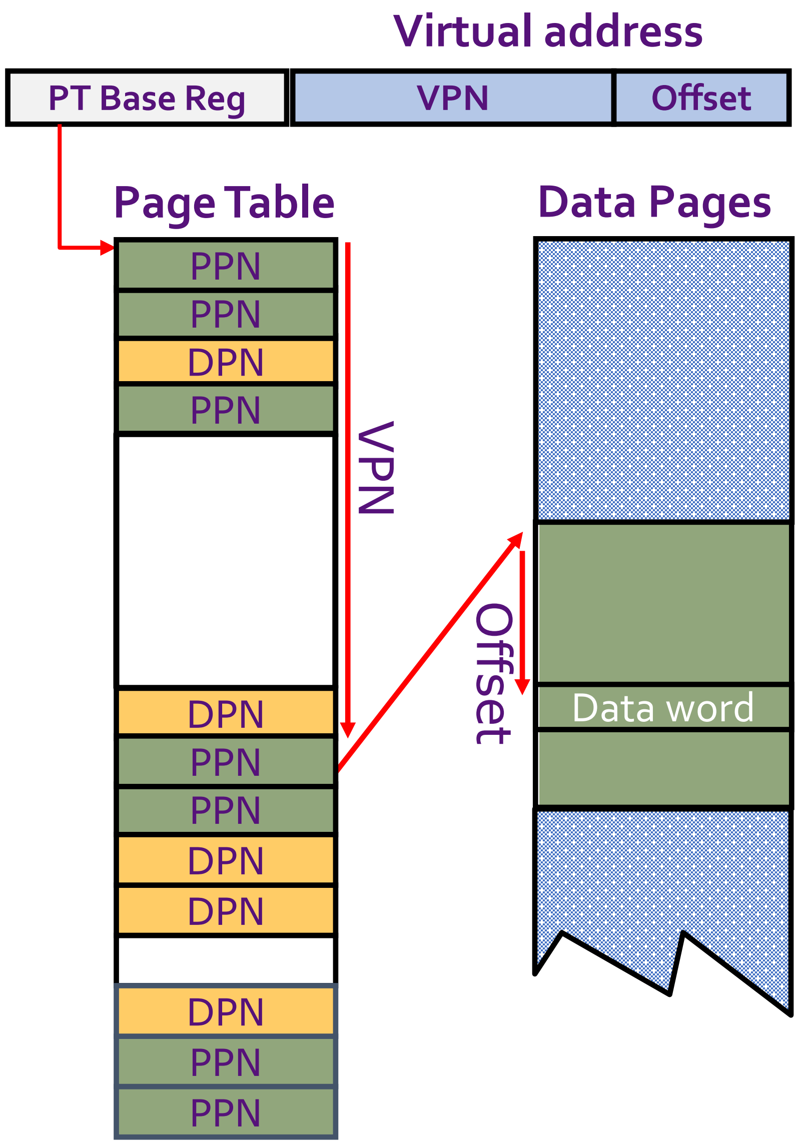
Two Remaining Questions

- What if all the pages can’t fit in DRAM?

- Where do we store the page tables?

Demand Paging: using main memory as a cache of disk

- All the pages of the processes may not fit in main memory → DRAM is backed up by ***swap space*** on disk.
- Page Table Entry (PTE) contains:
  - A resident bit to indicate if the page exists in main memory
  - PPN (physical page number) for a memory-resident page
  - DPN (disk page number) for a page on the disk
  - Protection and usage bits
- Even if all pages fit in memory, demand paging allows bringing only what is needed from disk
  - When a process starts, all code and data are on disk; bring pages in as they are accessed



Example: Virtual → Physical Translation

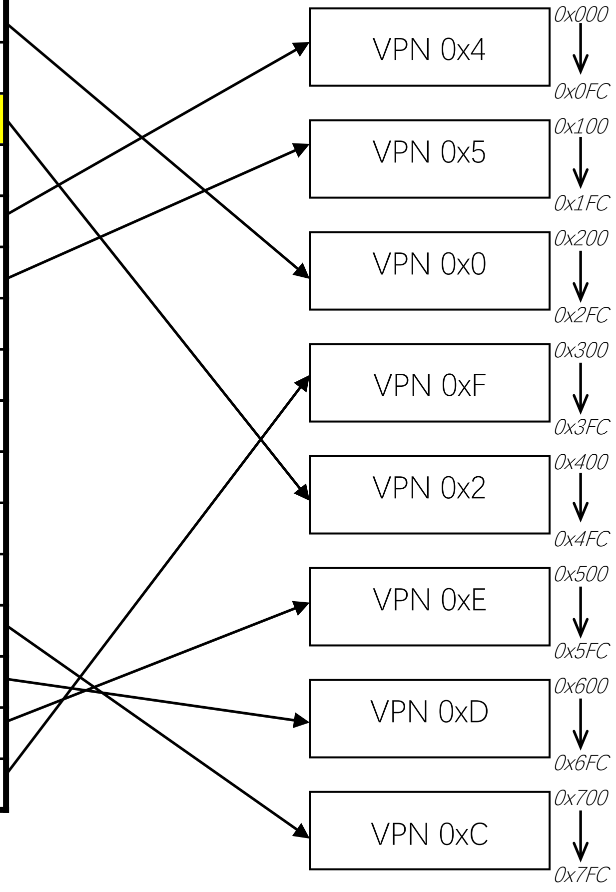
16-entry
Page Table

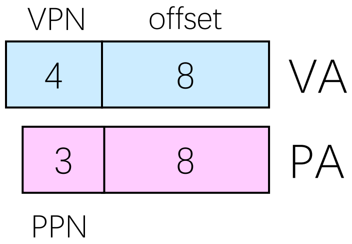
	D	W	R	PPN
0	0	0	1	2
1	--	--	0	--
2	0	0	1	4
3	--	--	0	--
4	0	0	1	0
5	1	1	1	1
6	--	--	0	--
7	--	--	0	--
8	--	--	0	--
9	--	--	0	--
A	--	--	0	--
B	--	--	0	--
C	1	1	1	7
D	1	1	1	6
E	1	1	1	5
F	0	1	1	3

D W R PPN

Dirty
Writable
Resident

8-page
Phys. Mem.





Setup:

- 256 bytes/page (2^8)
- 16 virtual pages (2^4)
- 8 physical pages (2^3)
- 12-bit VA (4 vpn, 8 offset)
- 11-bit PA (3 ppn, 8 offset)

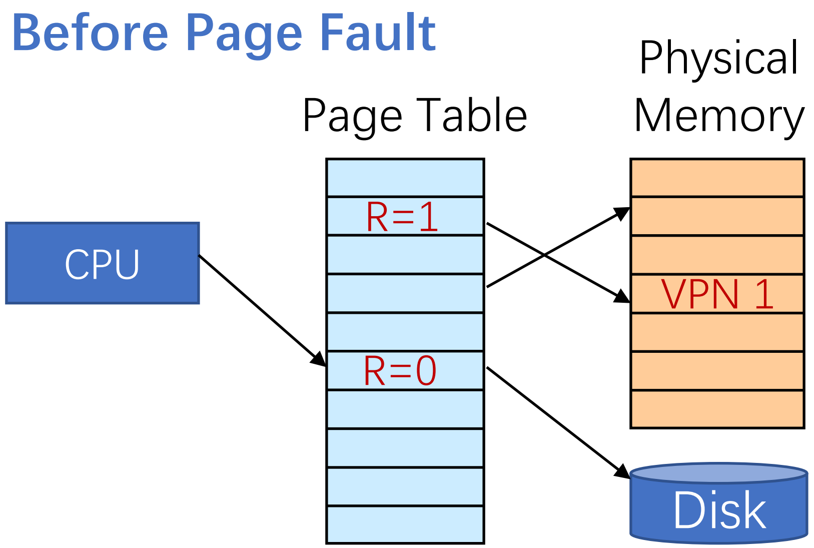
```
lw 0x2C8(x0)
```

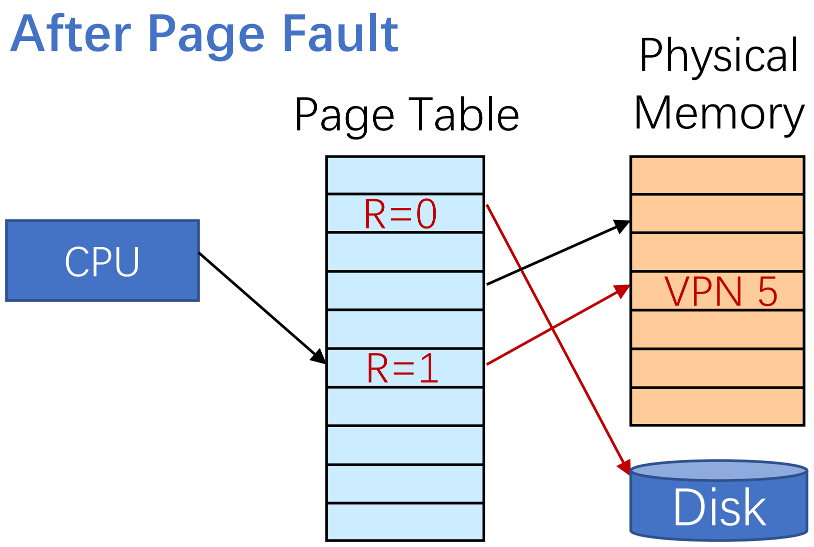
VA = 0x2C8, PA = **0x4C8**

VPN = 0x2
→ PPN = 0x4

Page Faults

- An access to a page that does not have a valid translation causes a page fault exception
- OS page fault handler is invoked, handles miss:
  - Choose a page to replace, write it back if dirty. Mark page as no longer resident
  - Read page from disk into available physical page
  - Update page table to show new page is resident
  - Return control to program, which re-executes memory access

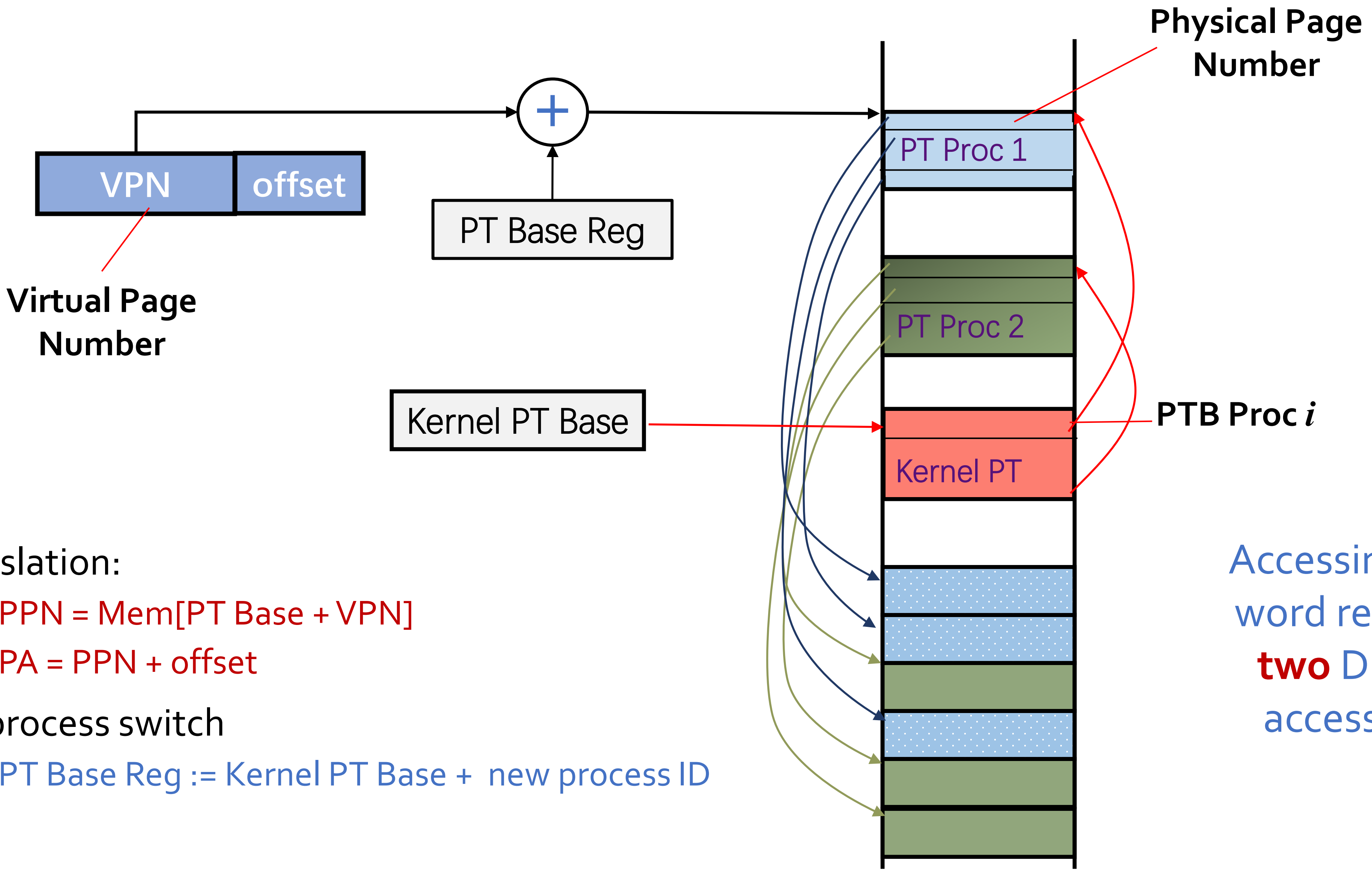




Two Remaining Questions

- What if all the pages can’t fit in DRAM?
- Where do we store the page tables?

Suppose Page Tables reside in Memory



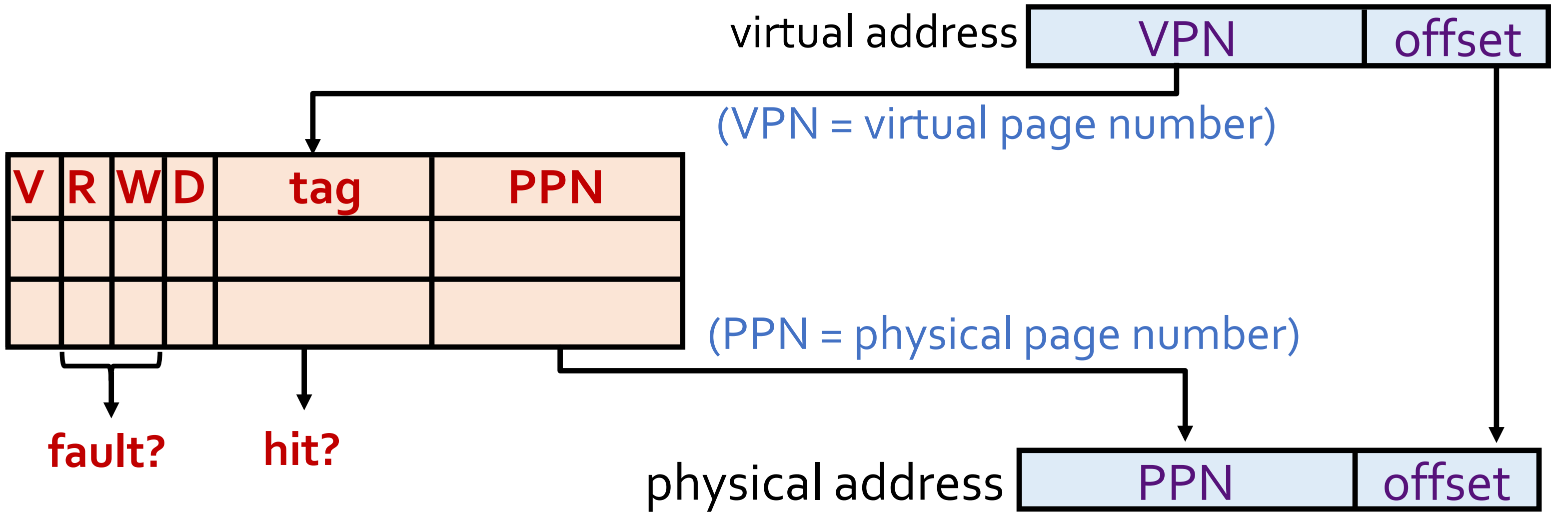
- Translation:
  - PPN = Mem[PT Base + VPN]
  - PA = PPN + offset
- On process switch
  - PT Base Reg := Kernel PT Base + new process ID

Accessing **one** word requires **two** DRAM accesses!!!

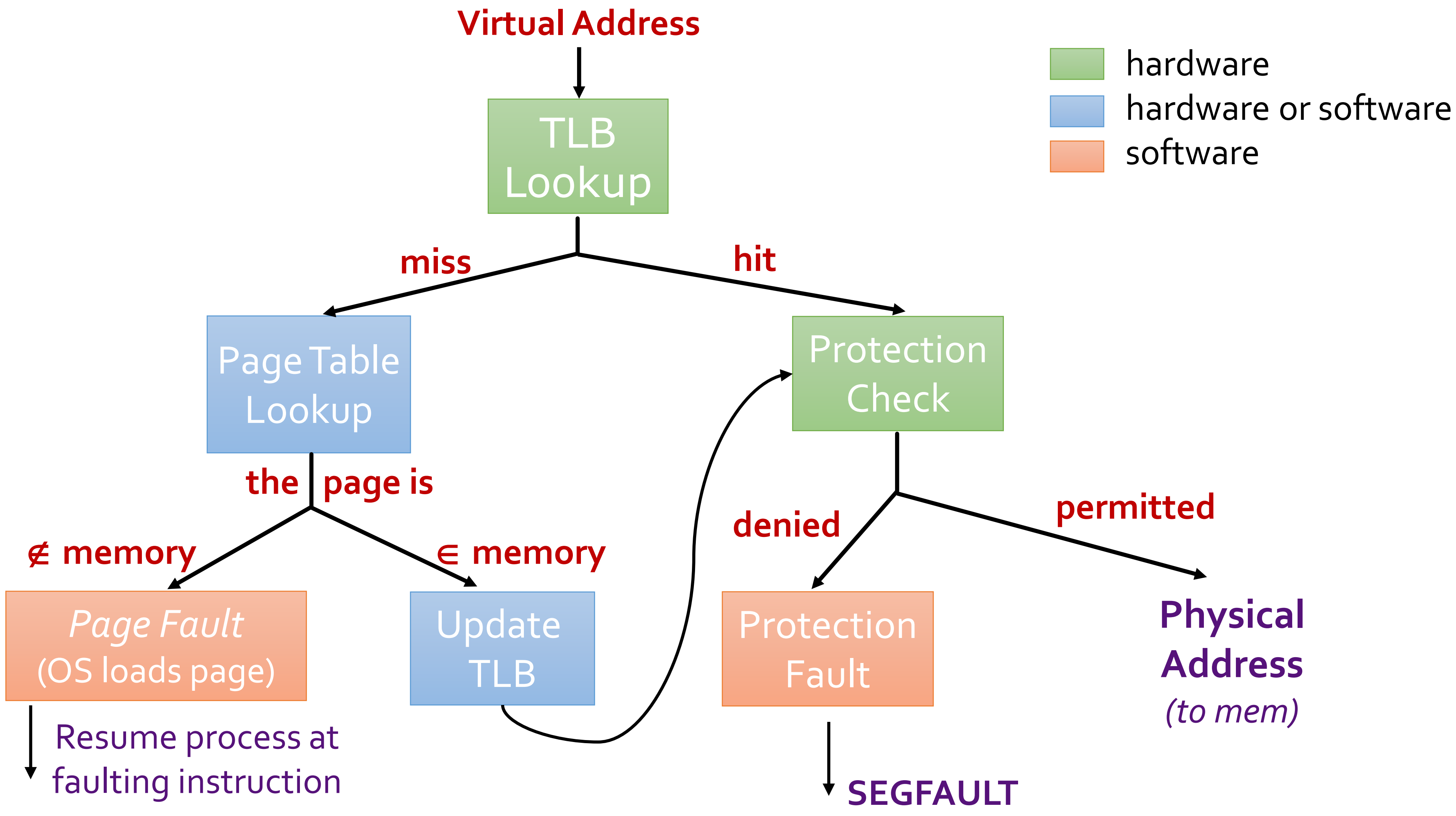
Translation Lookaside Buffer (TLB)

Problem: Address translation is very expensive!
Each reference requires accessing page table

Solution: **Cache translations in TLB**
TLB hit → *Single-cycle translation*
TLB miss → *Access page table to refill TLB*



Address Translation: *Putting it all together*



Summary

- Virtual memory benefits:
  - Protection and Privacy: Private address space per process
  - Demand Paging: use main memory as a cache of disk
- Segmentation: Each process address space is a contiguous block (a segment) in physical memory
  - Simple: Base and bound registers
  - Suffers from fragmentation, no demand paging
- Paging: Each process address space is stored on multiple fixed-size pages. A page table maps virtual to physical pages
  - Avoids fragmentation
  - Enables demand paging: pages can be in main memory or disk
  - Requires a page table access on each memory reference
- TLBs make paging efficient by caching the page table

Thank you!

Q & A