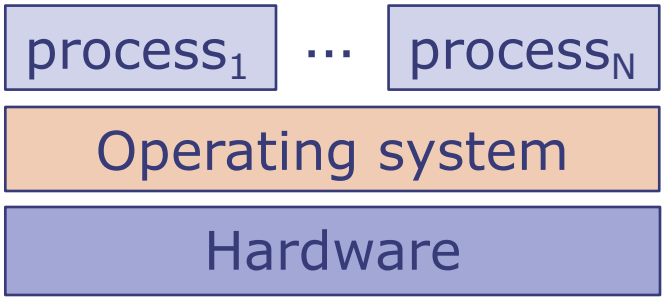
Virtual Memory

Silvina Hanono Wachman
Computer Science & Artificial Intelligence Lab
M.I.T.

Reminder: Operating Systems

- Goals of OS:
  - Protection and privacy: Processes cannot access each other's data
  - Abstraction: Hide away details of underlying hardware
    - e.g., processes open and access files instead of issuing raw commands to hard drive
  - Resource management: Controls how processes share hardware resources (CPU, memory, disk, etc.)



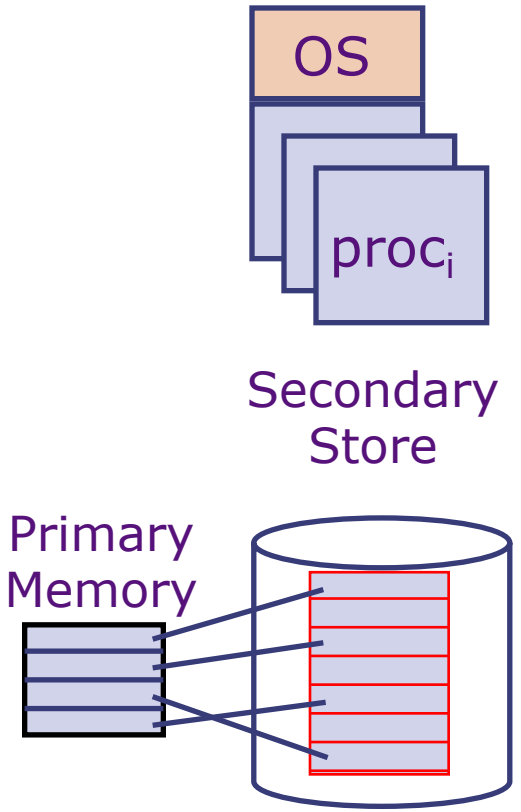
- Key enabling technologies:
  - User mode + supervisor mode
  - Interrupts to safely transition into supervisor mode
  - Virtual memory to abstract the storage resources of the machine

Last lecture

Today

Virtual Memory (VM) Systems
Illusion of a large, private, uniform store

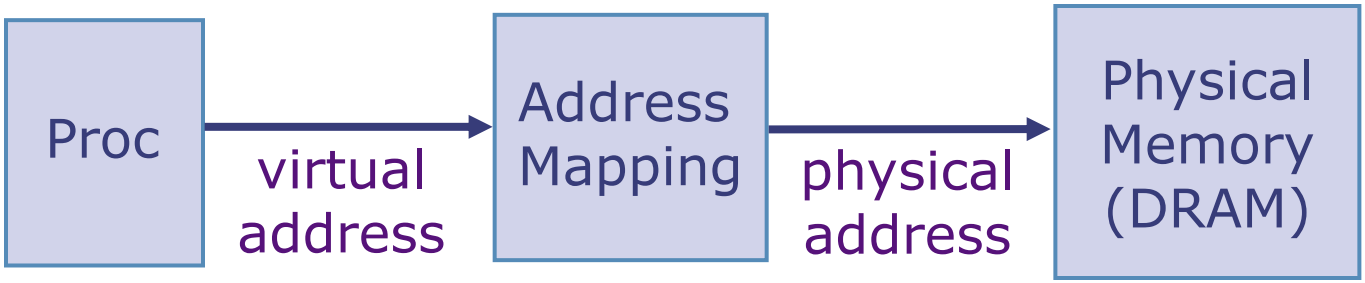
- Protection & Privacy
  - Each process has a private address space
- Demand Paging
  - Provides the ability to run programs larger than main memory
  - Hides differences in machine configuration



The price of VM is address translation on each memory reference

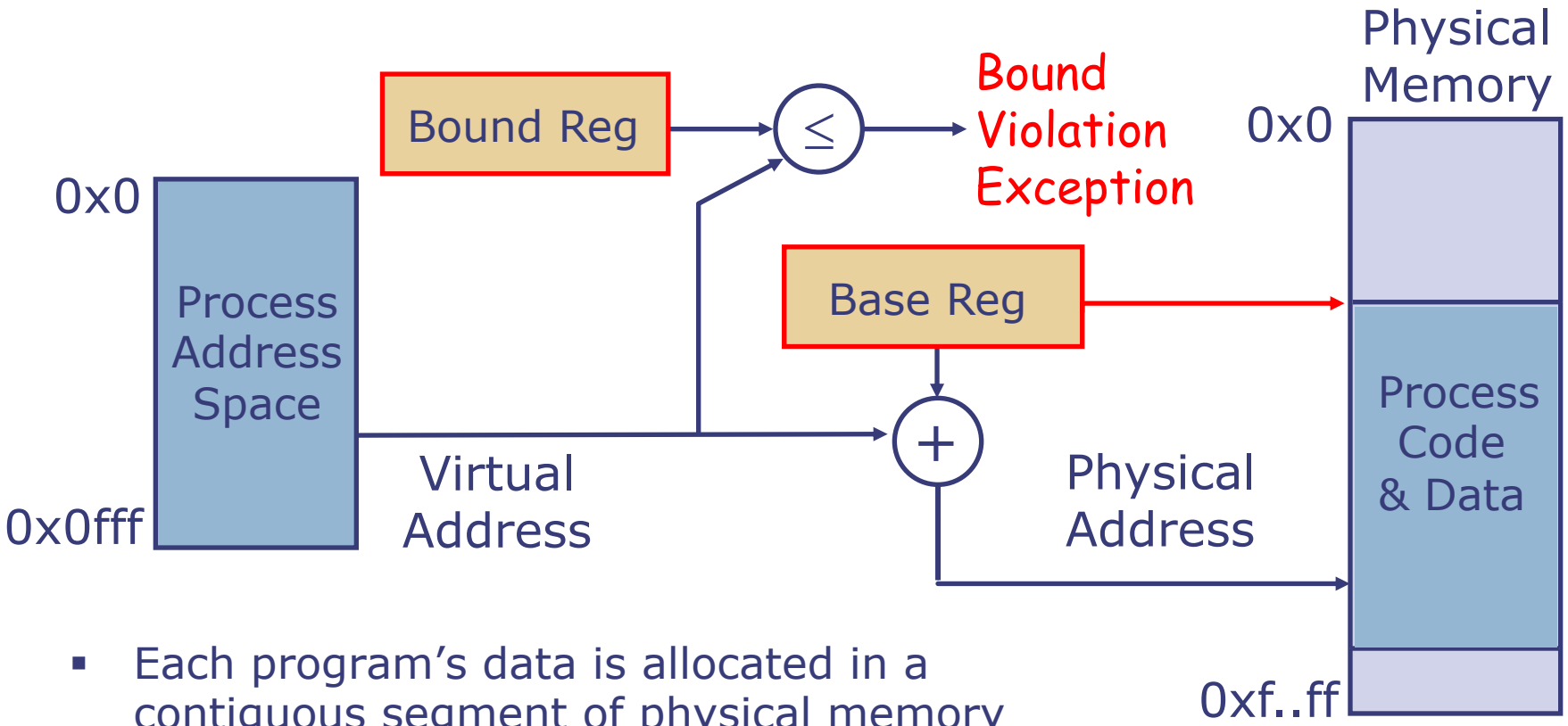
VM is needed for usability but can cause serious performance degradation

Names for Memory Locations



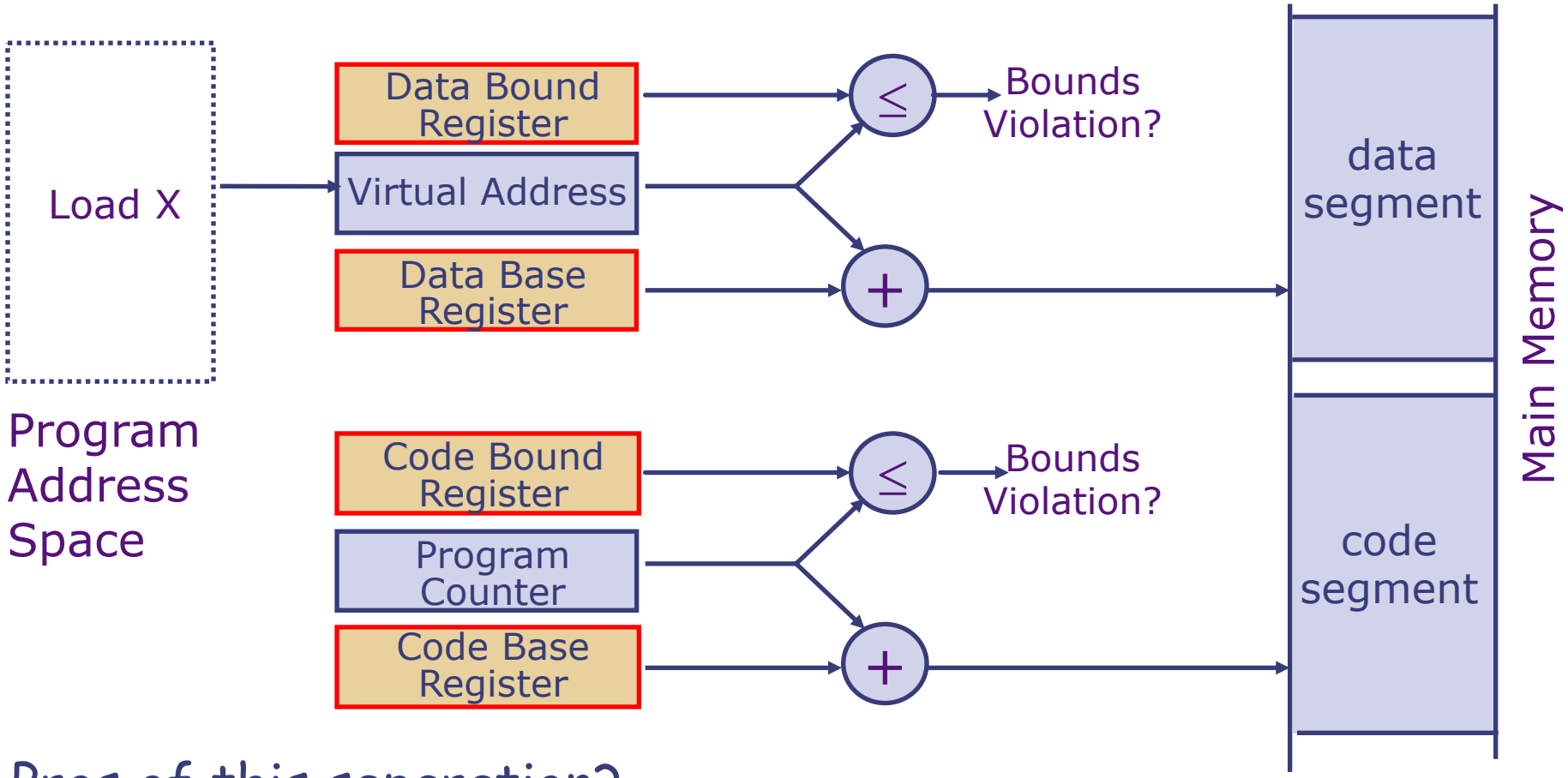
- Virtual address
  - Multiple processes can use the same virtual address to refer to different physical memory locations.
- Physical address
  - Operating system specifies mapping of virtual address into name for a physical memory location

Base-and-Bound Address Translation aka Segmentation



- Each program's data is allocated in a contiguous segment of physical memory
- Physical address = Segment Base + Virtual Address
- 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

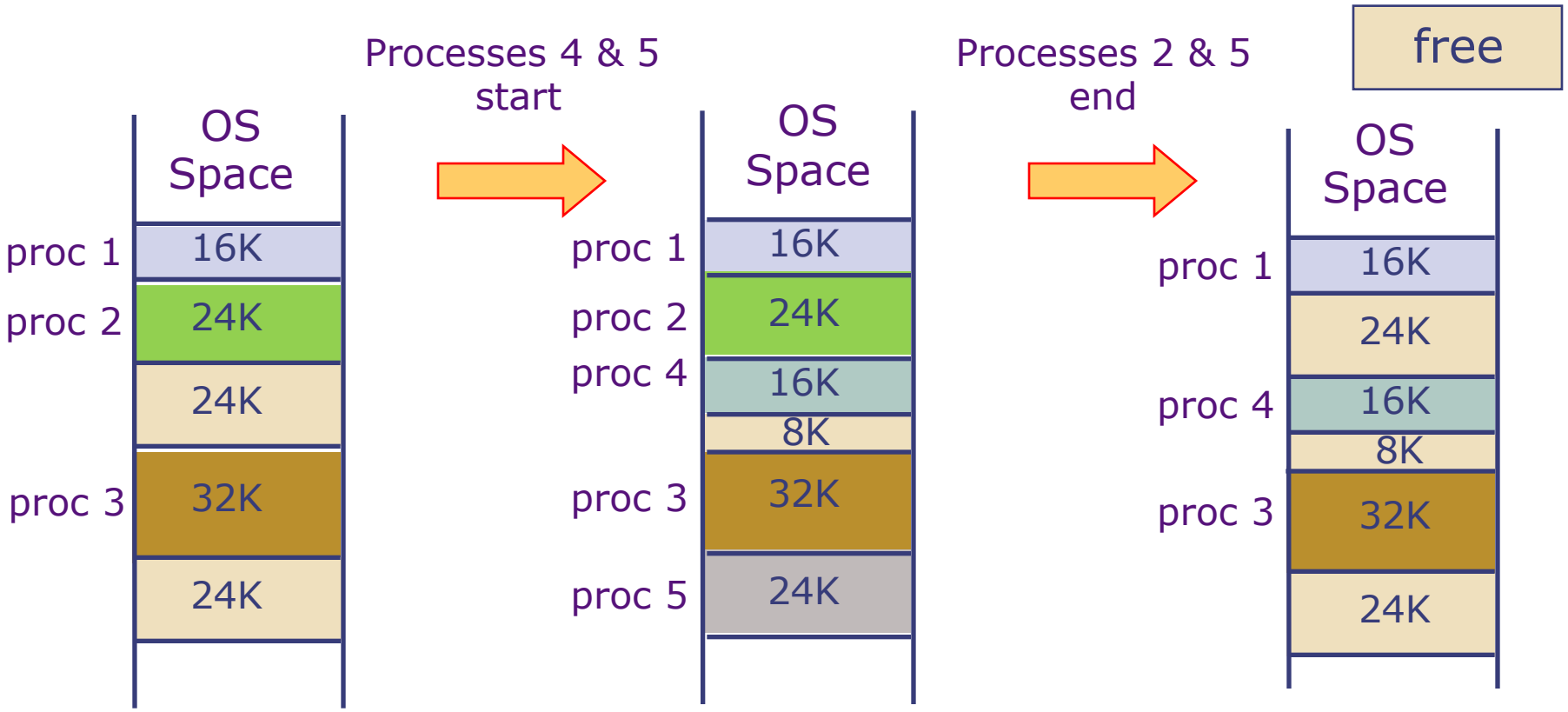


Pros of this separation?

Prevents buggy program from overwriting code

Multiple processes can share code segment

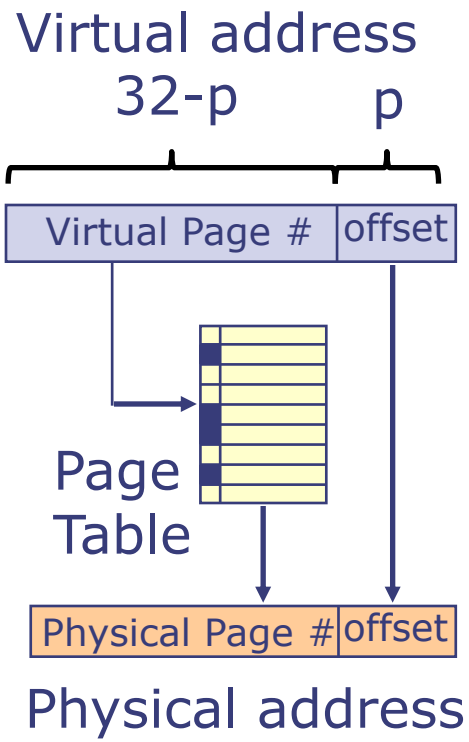
Memory Fragmentation



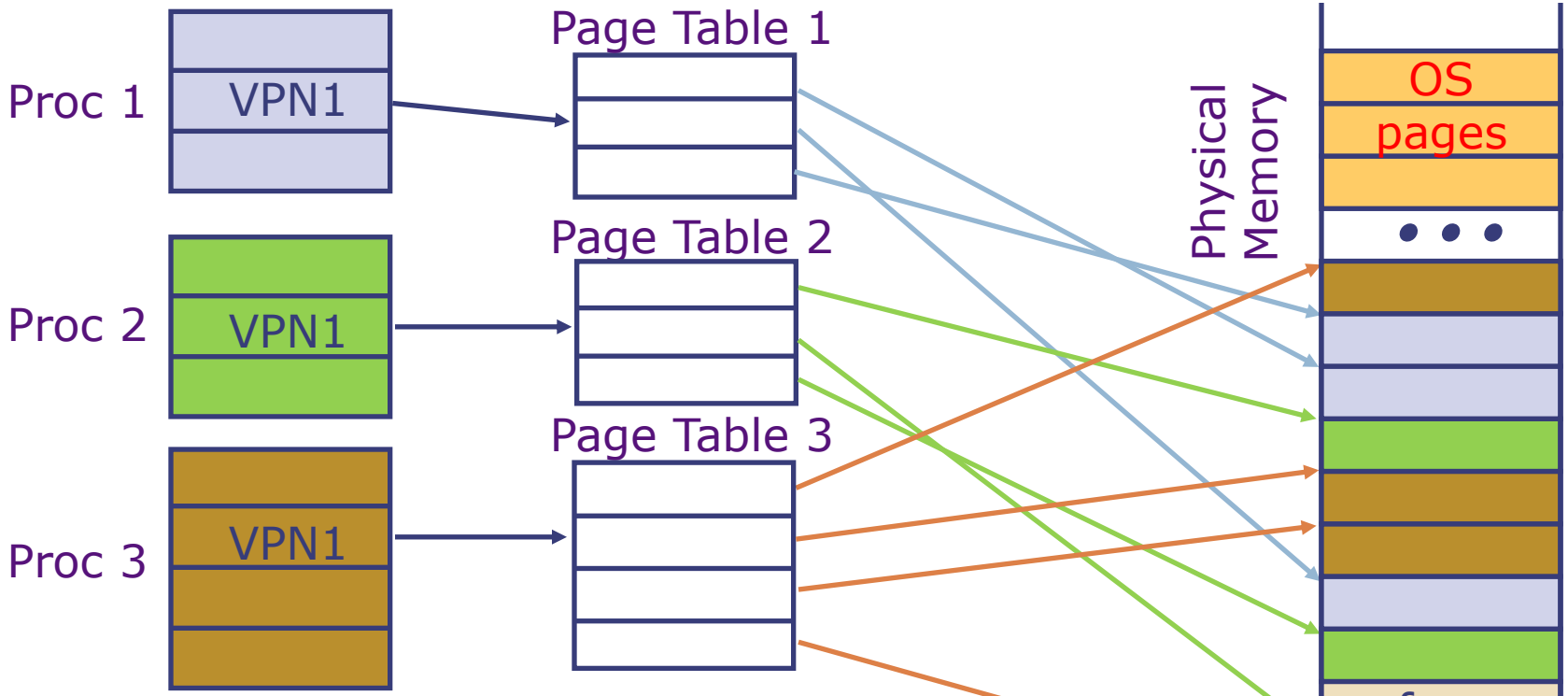
As processes start and end, storage is "fragmented". Therefore, at some point segments have to be moved around to compact the free space.

Paged Memory Systems

- Divide physical memory into 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
  - Number of entries in page table = 2^{32-p}



Private Address Space per Process

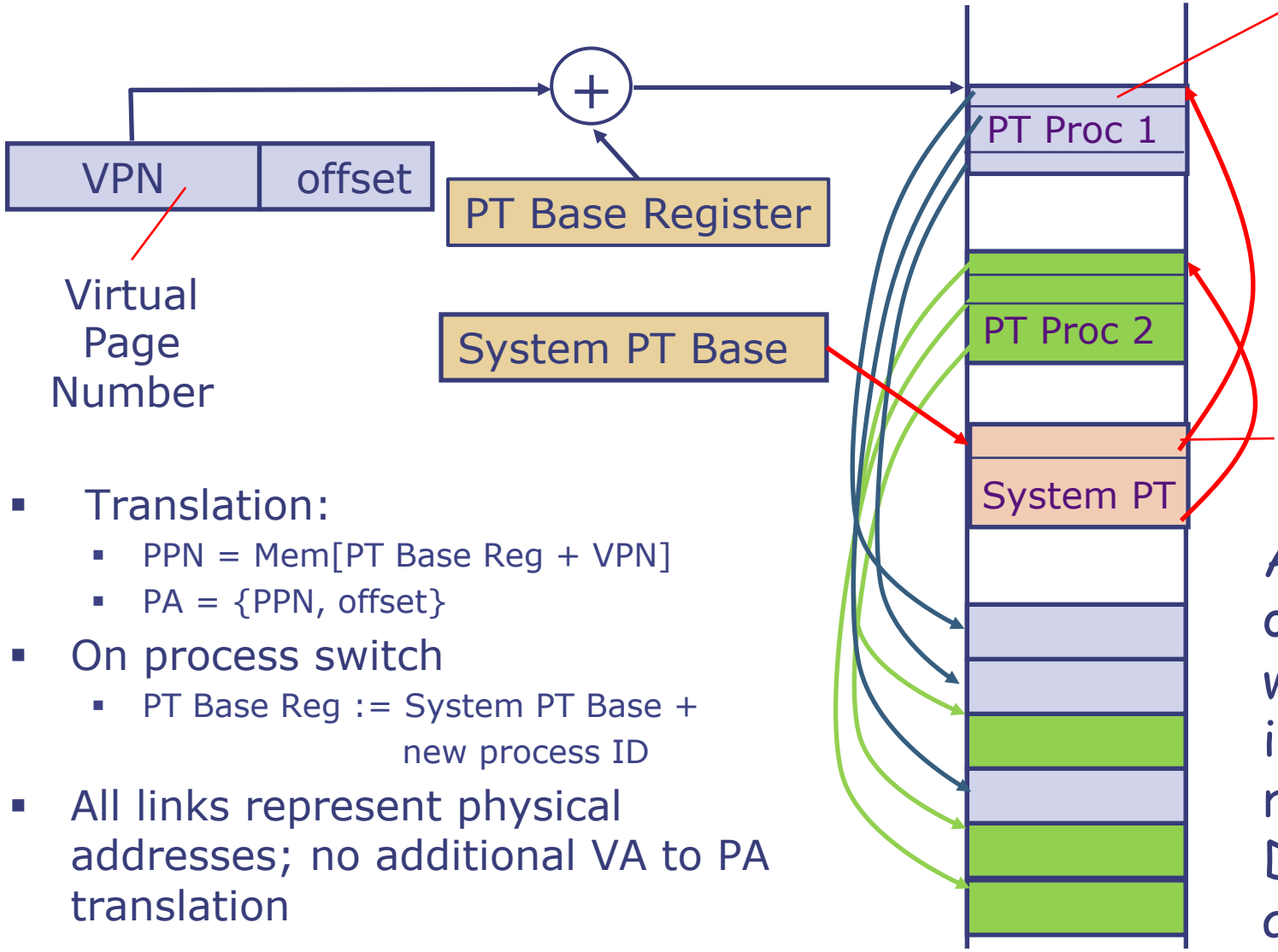


- Each process has a page table
- Page table has an entry for each process page
- An OS controlled PT base register points to the page table of the current process

Where do page tables reside?

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

Suppose Page Tables reside in memory



- Translation:
  - PPN = Mem[PT Base Reg + VPN]
  - PA = {PPN, offset}
- On process switch
  - PT Base Reg := System PT Base + new process ID
- All links represent physical addresses; no additional VA to PA translation

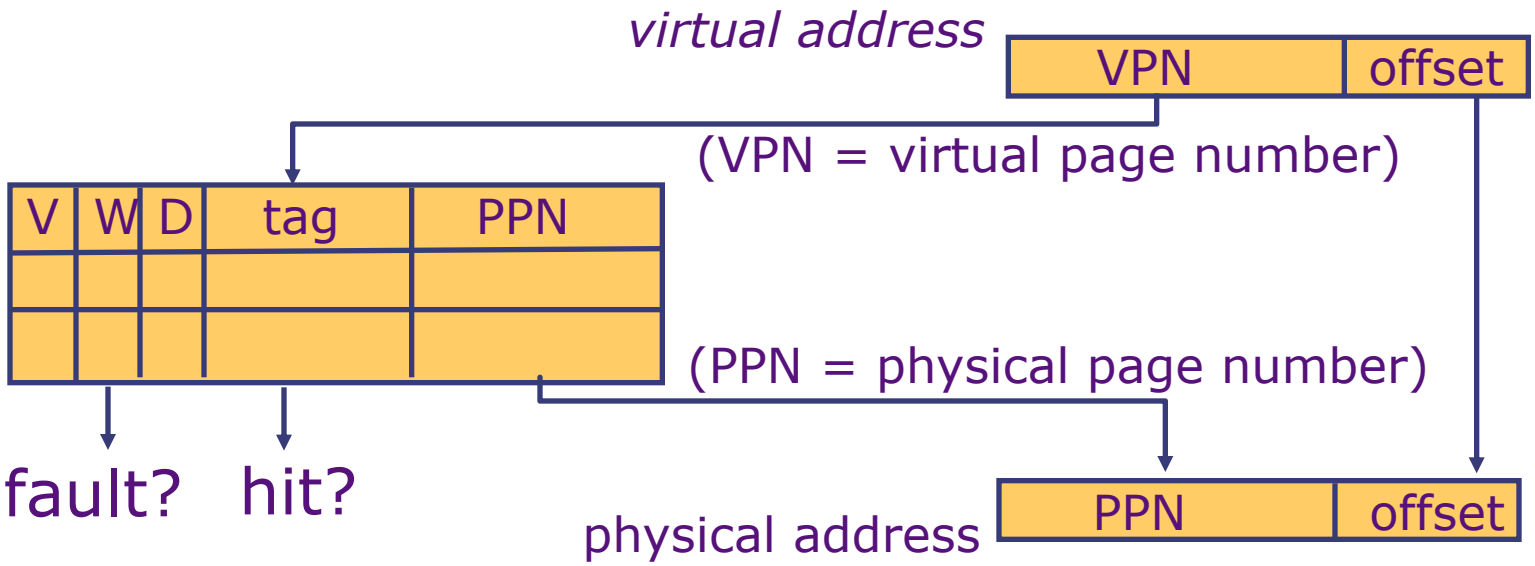
Accessing one data word or instruction requires two DRAM accesses!

Translation Lookaside Buffer (TLB)

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

Solution: *Cache VPN to PPN translations in TLB*

TLB hit ⇒ Single-cycle Translation
TLB miss ⇒ Access Page Table to refill TLB



TLB Designs

- Typically 32-128 entries, 4 to 8-way set-associative
  - Modern processors use a hierarchy of TLBs!
- Handling a TLB miss: Hardware looks up the page table (aka walking the page table) and loads the corresponding VPN→PPN entry into TLB
  - Can be expensive if the page is "missing" (to be explained)
    - RISC-V, x86 access page table in hardware
- TLB is flushed on a process switch: This makes restarting the process expensive
  - TLB flush can be avoided if process IDs are included in TLB entries (reduces the effective capacity of TLB)

Can we execute programs which do not fit in physical memory?

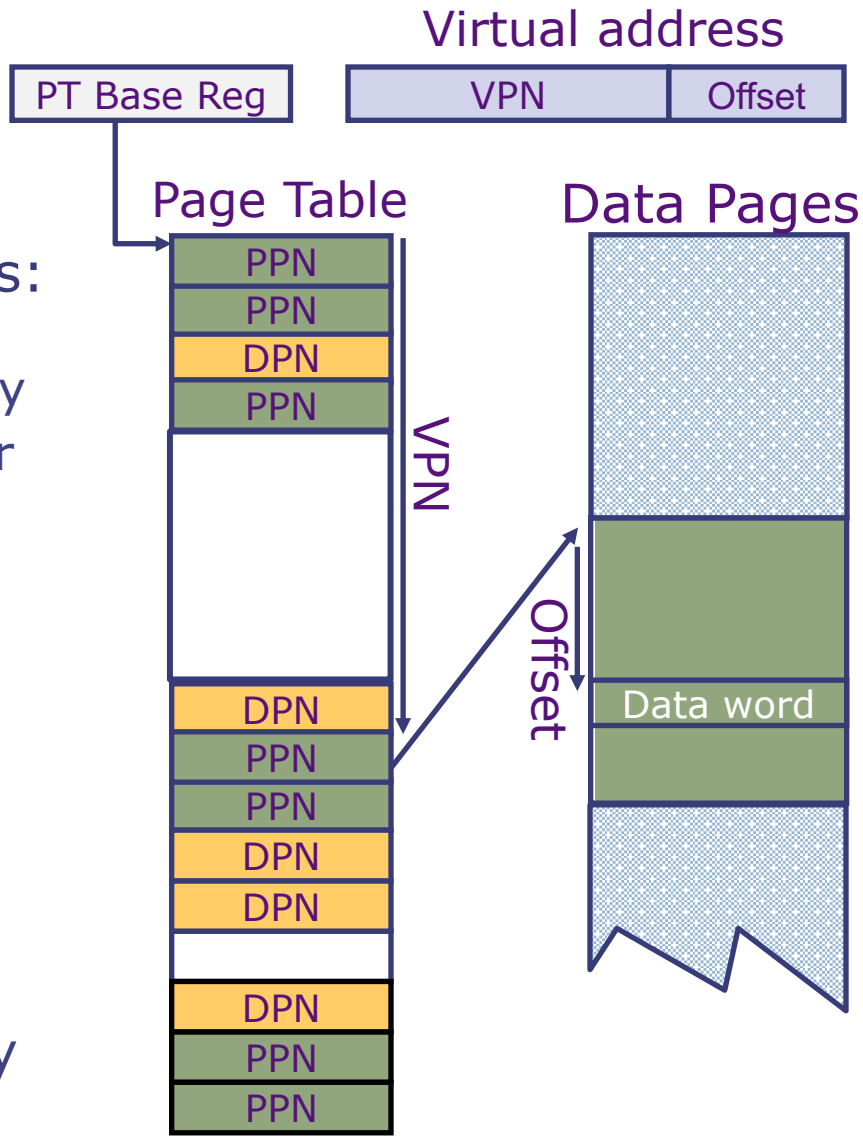
yes

The program and its data reside in the secondary storage (Disk)) and we bring its pages into the memory (DRAM) on an "as needed" basis

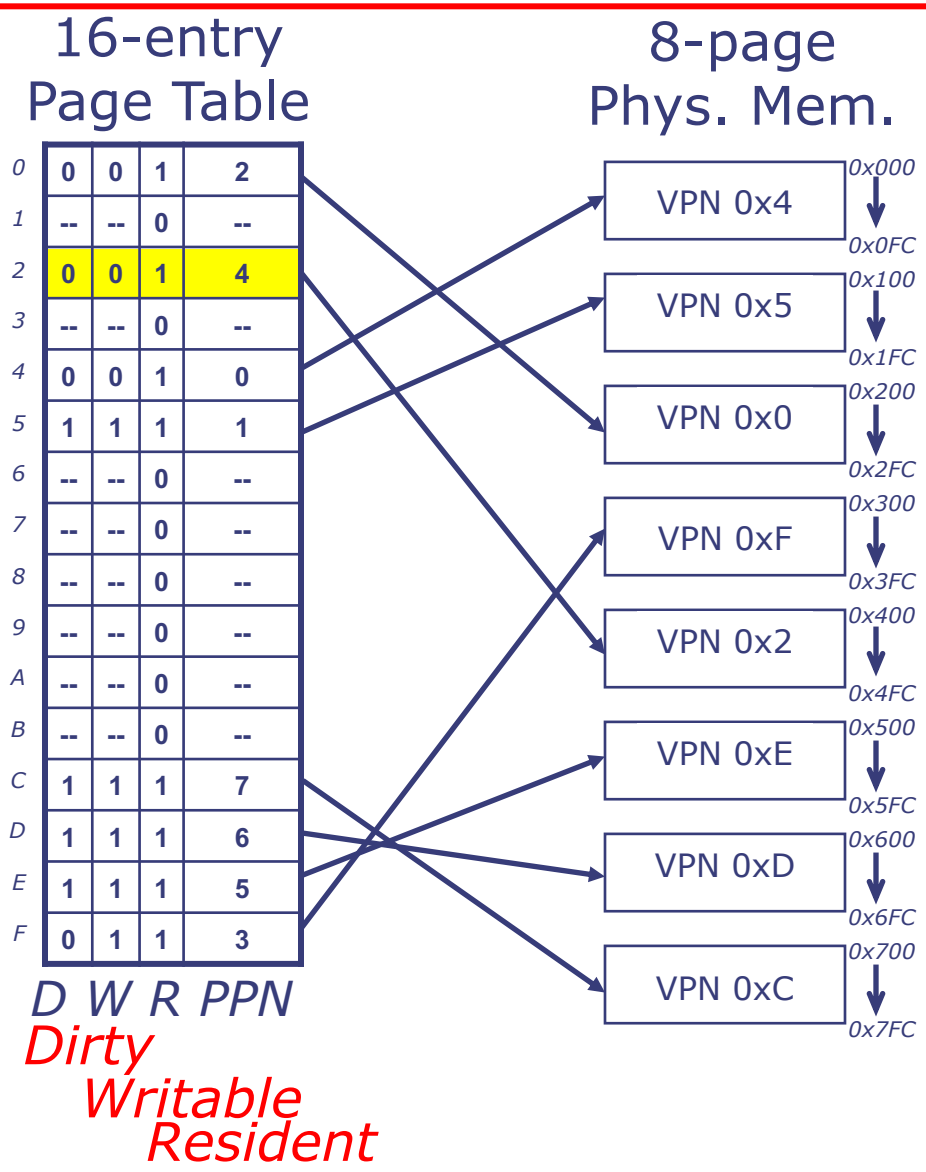
Demand Paging

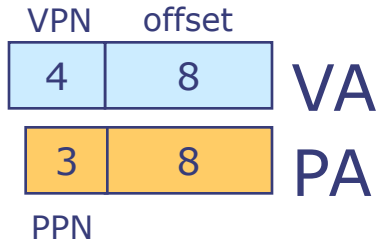
Demand Paging
Using main memory as a cache for disk

- All the pages of the processes may not fit in main memory. Therefore, DRAM is backed up by *swap space* on disk.
- Page Table Entry (PTE) contains:
  - A resident bit (R) 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; pages are brought in as they are demanded (accessed)



Example: Virtual → Physical Translation





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

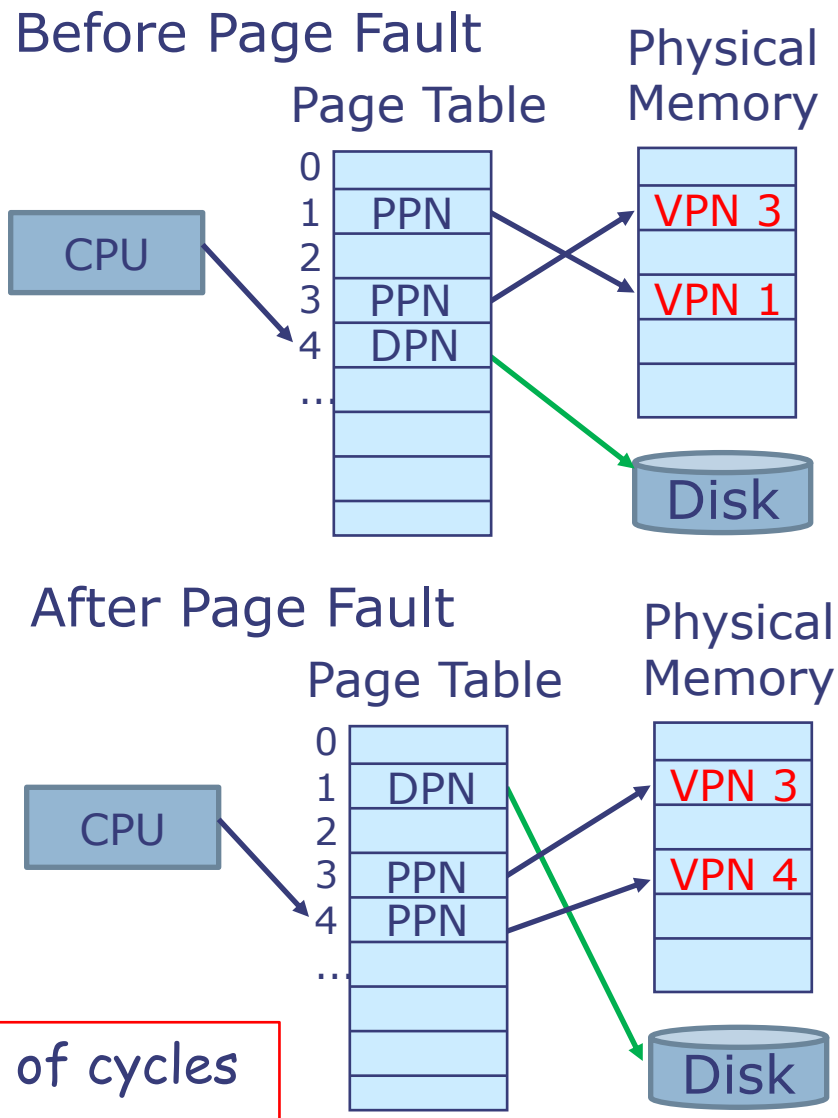
VPN = 0x2

→ PPN = 0x4

Page Faults

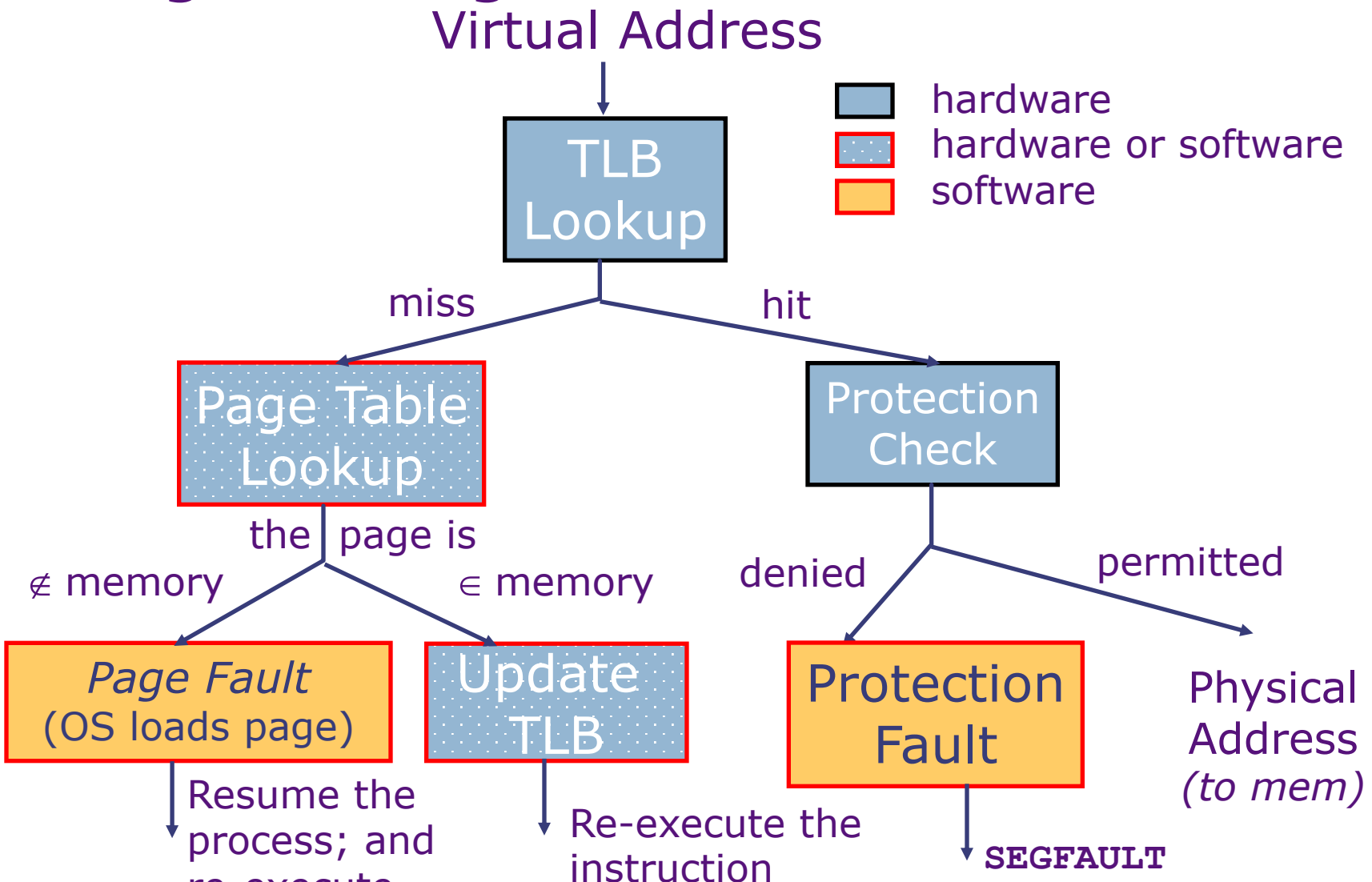
An access to a page that is not in main memory causes a page fault exception (e.g., Page 4). OS page fault handler is invoked, handles miss:

- Choose a page to replace, write it back if dirty (e.g., page VP1). 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



Page faults take millions of cycles to process

Address Translation:
putting it all together



Example: TLB and Page Table

Suppose
- Virtual memory of 2^{32} bytes
- Physical memory of 2^{24} bytes
- Page size is 2^{10} (1 K) bytes
- 4-entry fully associative TLB

TLB

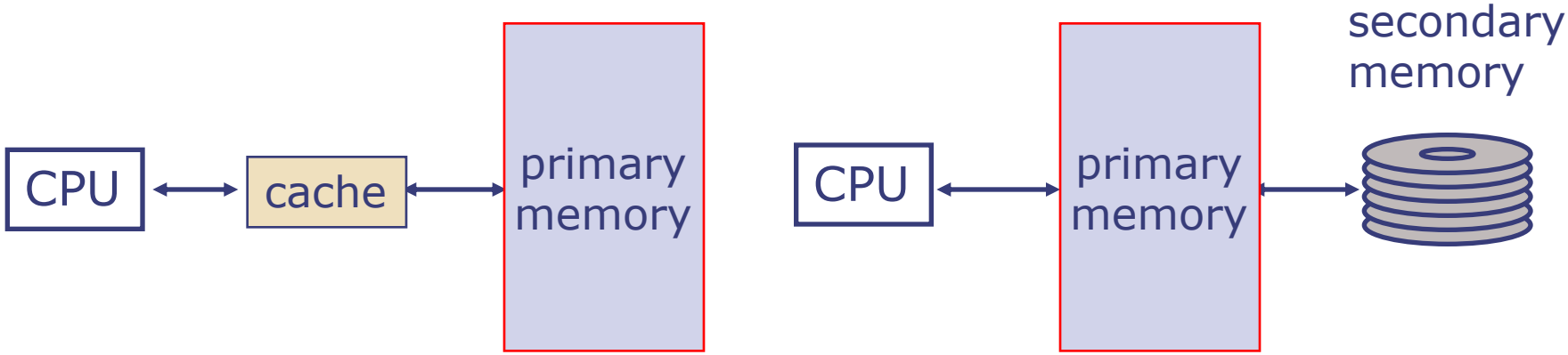
Tag	Data		
VPN	R	D	PPN
0	0	0	3
6	1	1	2
1	1	1	9
3	0	0	5

Page Table

VPN	R	D	PPN
0	0	0	7
1	1	1	9
2	1	0	0
3	0	0	5
4	1	0	5
5	0	0	3
6	1	1	2
7	1	0	4
8	1	0	1
	…		

1. How many pages can be stored in physical memory at once? 2^{14}
2. How many entries are there in the page table? 2^{22}
3. How many bits per entry in the page table? (Assume each entry has PPN, resident bit, dirty bit) 16
4. How many pages does page table take? 2^{23} bytes = 2^{13} pages
5. What fraction of virtual memory can be resident? $1/2^{8}$
6. What is the physical address for virtual address 0x1804? What components are involved in the translation? [VPN=6] 0x804
7. Same for 0x1080 [VPN=4] 0x1480
8. Same for 0x0FC [VPN=0] page fault

Caching vs. Demand Paging



Caching

- cache entry
- cache block (~32 bytes)
- cache miss rate (1% to 20%)
- cache hit (~1 cycle)
- cache miss (~100 cycles)
- a miss is handled
  - in *hardware*

Demand paging

- page frame
- page (~4K bytes)
- page miss rate (<0.001%)
- page hit (~100 cycles)
- page miss (~5M cycles)
- a miss is handled
  - mostly in *software*

Thank you!

Next lecture: Pipelining