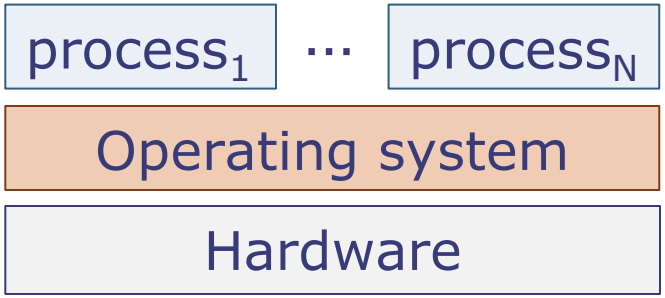
Lecture 17
Virtual Memory 1

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.)



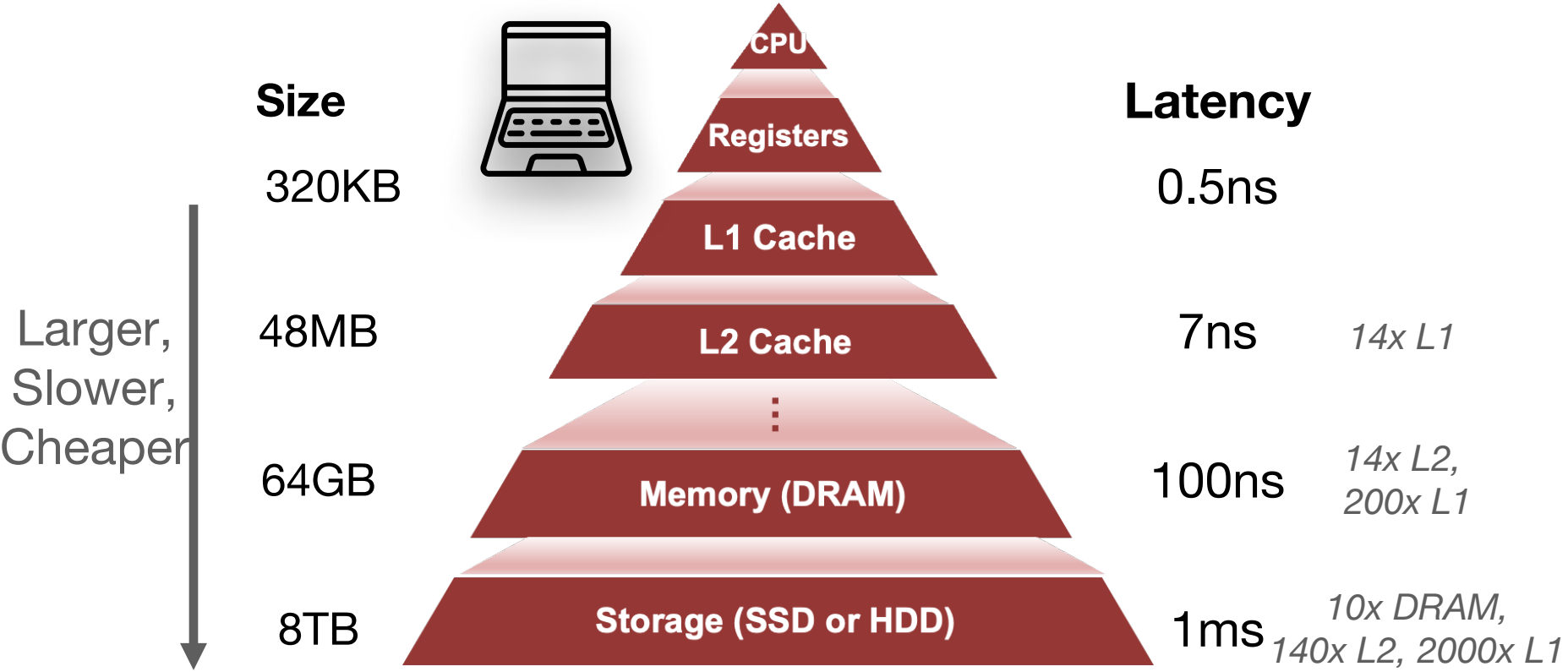
- Key enabling technologies:
  - **User** mode + **supervisor** mode
  - **Exceptions** to safely transition into supervisor mode
  - **Virtual memory** to abstract the storage resources of the machine

Last lecture

Today

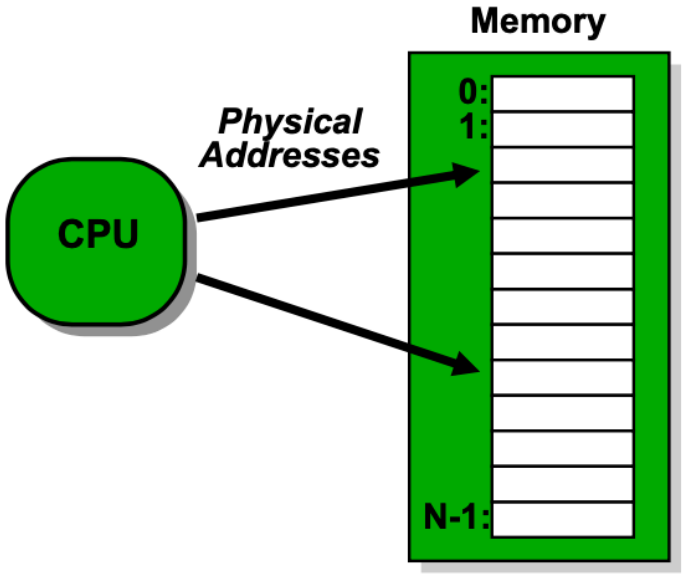
Memory Hierarchy

- Data transfer between layers
  - Register <-> Cache: Processor bit-width (e.g., 32/64bits)
  - Cache <-> Memory: Cache line (e.g., 64 bytes)
  - Memory <-> Disk: Page of virtual memory (e.g., 4 KB)

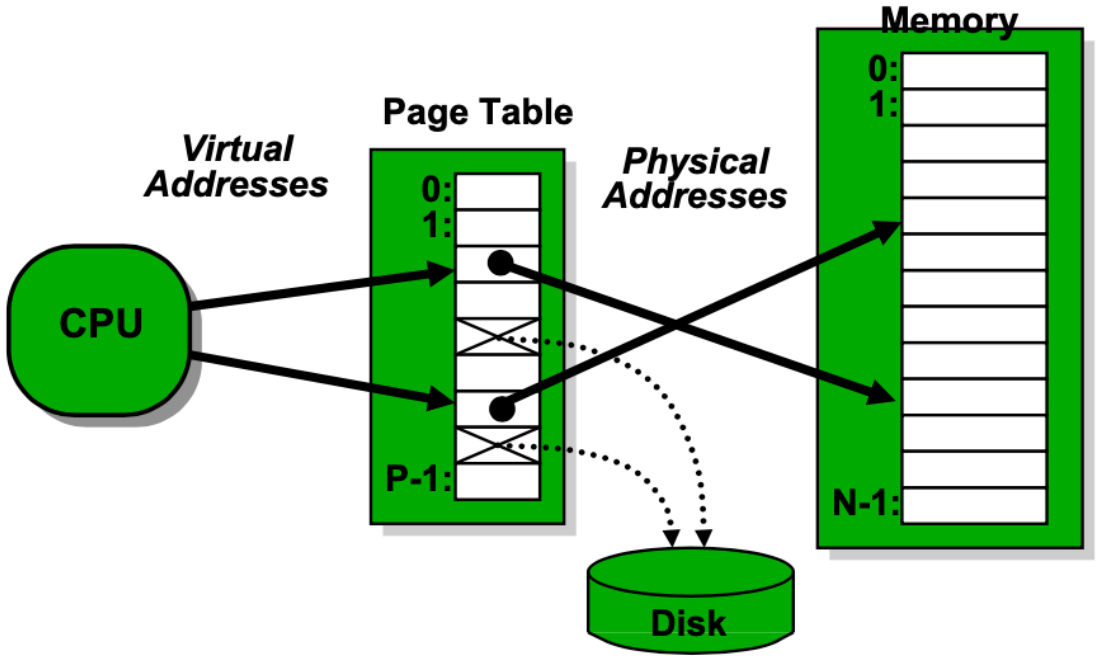


Systems with/without Virtual Memory

- System with Physical Memory only: Addresses generated by the CPU point directly to bytes in physical memory
- System with Virtual Memory: Hardware converts virtual addresses to physical addresses via an OS-managed lookup table (page table)



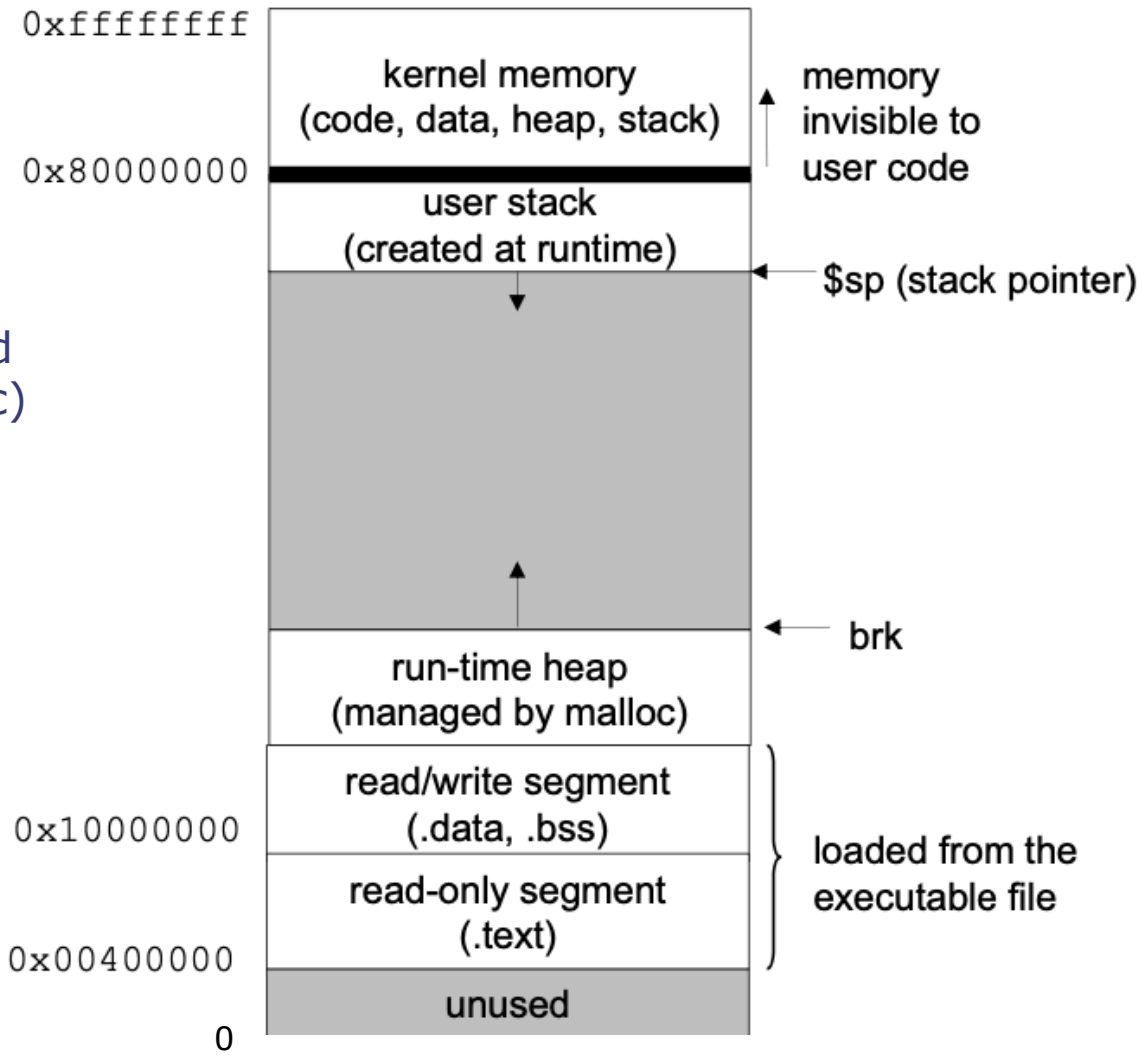
Physical memory only



Physical memory only

Process Address Space

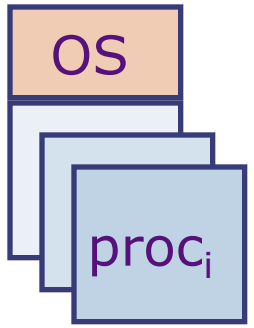
- Each process has its own private address space
  - Stack: Temporary data, such as local variables and grows/shrinks at runtime
  - Heap: Variables allocated dynamically (e.g., malloc)
  - .data: Initialized global and static variables
  - .bss: Uninitialized global and static variables
  - .text: Code segment

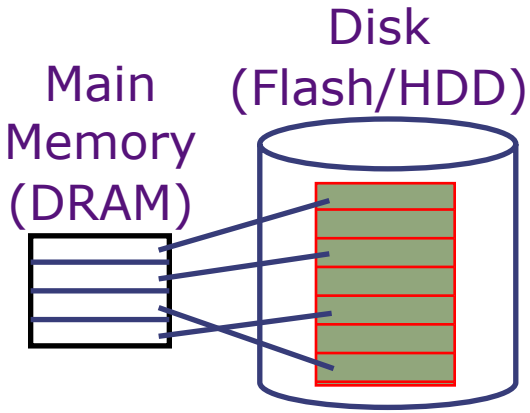


Reference: Stanford EE108b

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

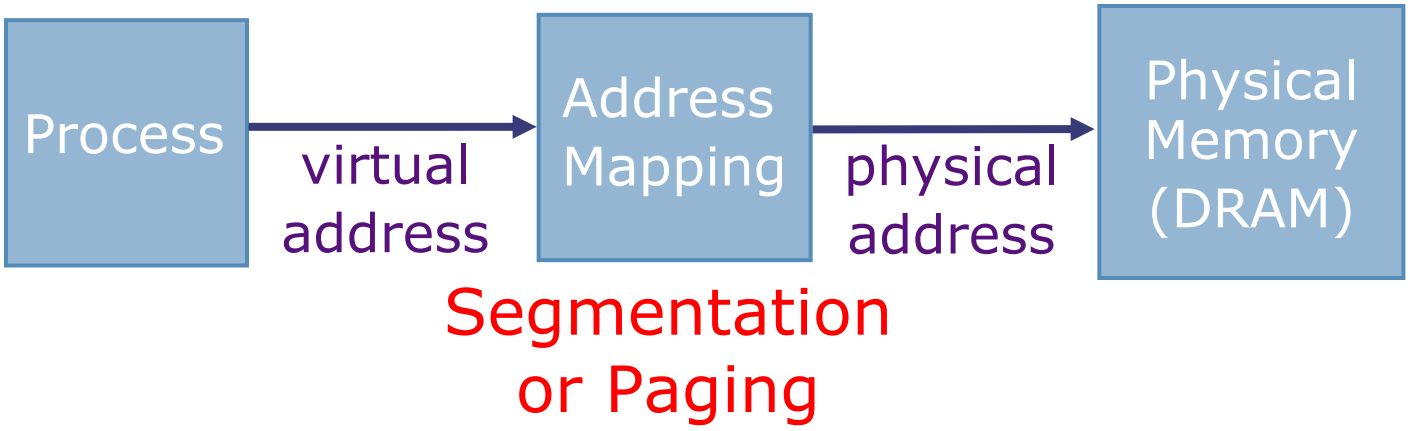
- Protection & Privacy
  - Each process has a **private** address space
- Demand Paging
  - Use main memory as a **cache** of disk
  - Enables running programs **larger** than main memory
  - Hides the **differences** in machine configuration





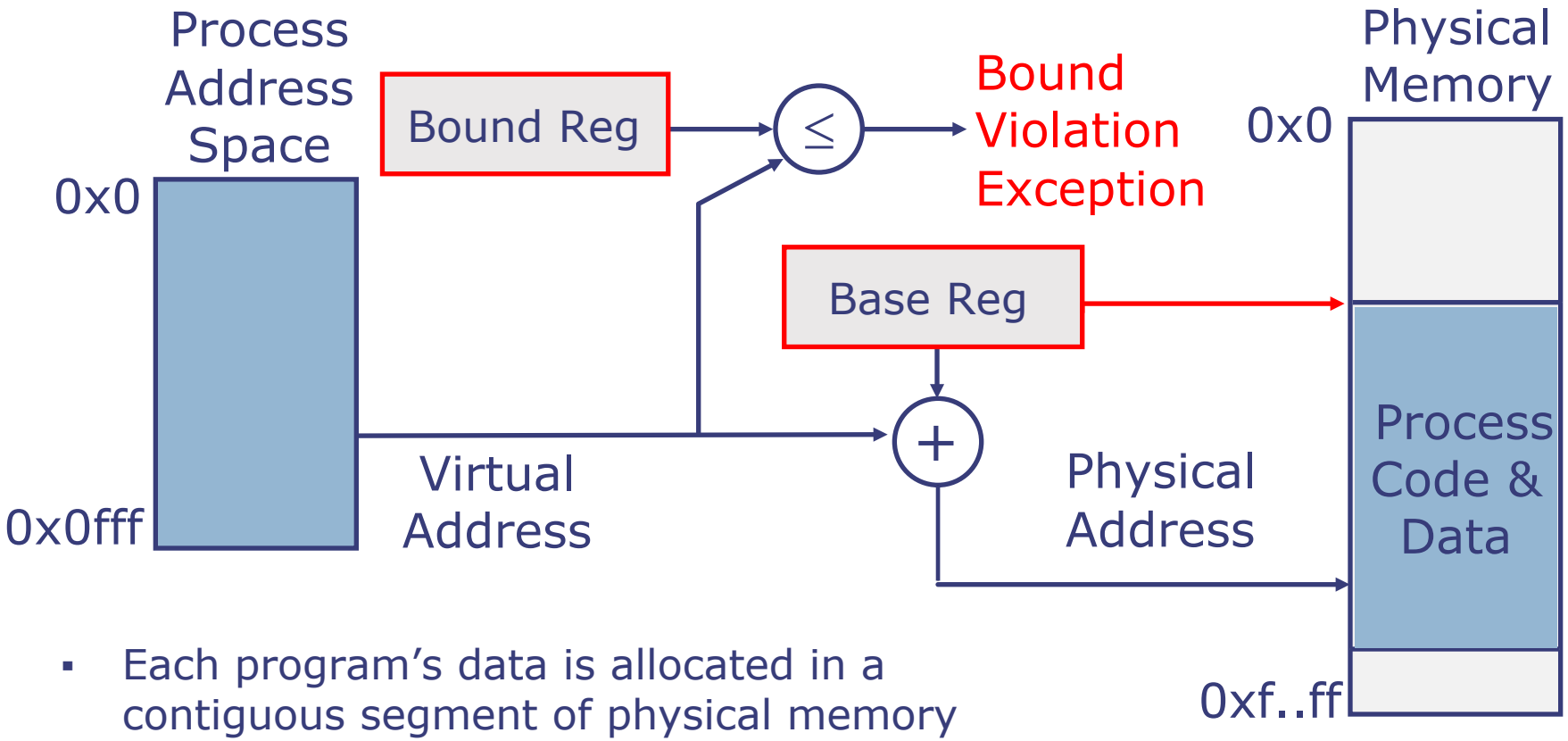
The price of VM is address translation on each memory reference

Names for Memory Locations



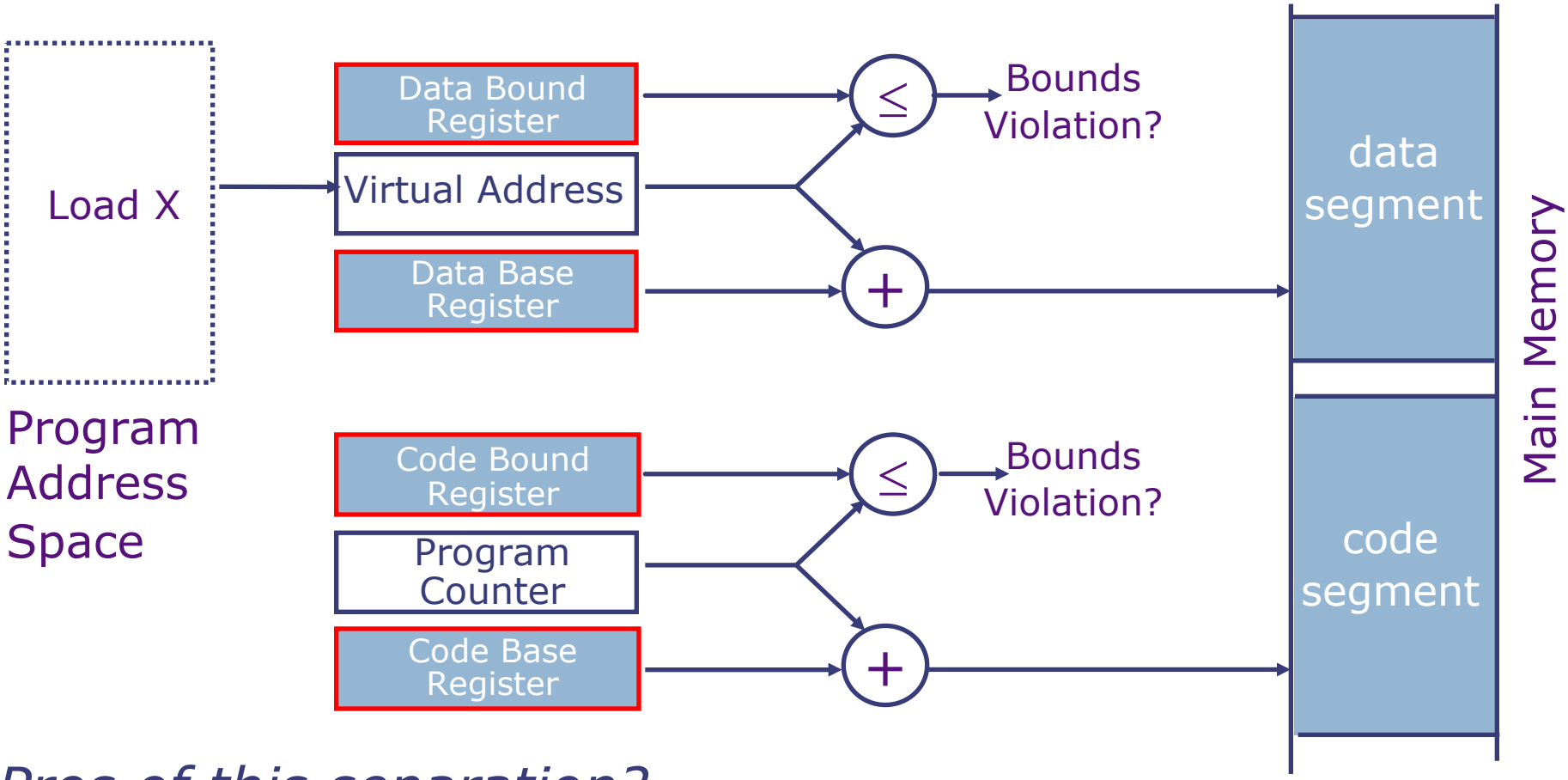
- Virtual address
  - Address generated by the process
  - Specific to the process's private address space
- Physical address
  - Address used to access physical (hardware) memory
  - Operating system specifies mapping of virtual addresses into physical addresses

Segmentation (Base-and-Bound) Address Translation



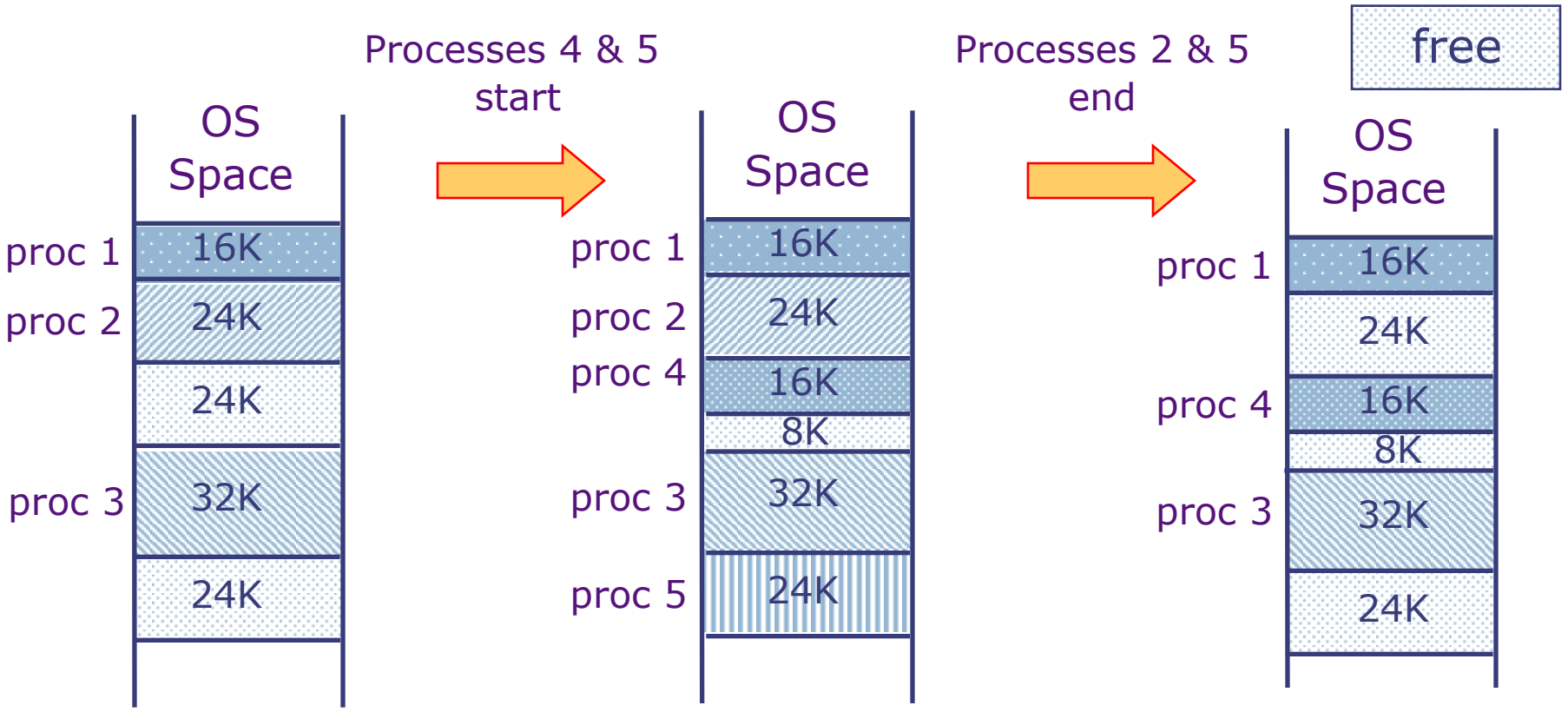
- Each program's data is allocated in a contiguous segment of physical memory
- Physical address = Virtual Address + Segment Base
- Bound register provides safety and isolation
- Base and Bound registers should not be accessed by user programs (only accessible in supervisor mode)

Separate Segments for Code and Data



Pros of this separation?

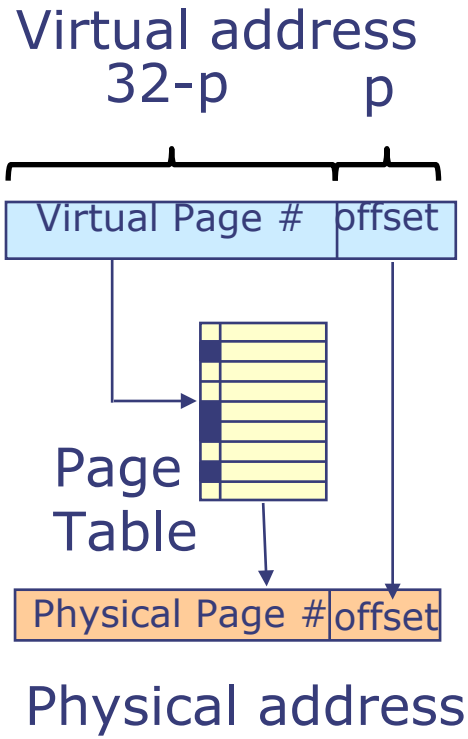
Memory Fragmentation



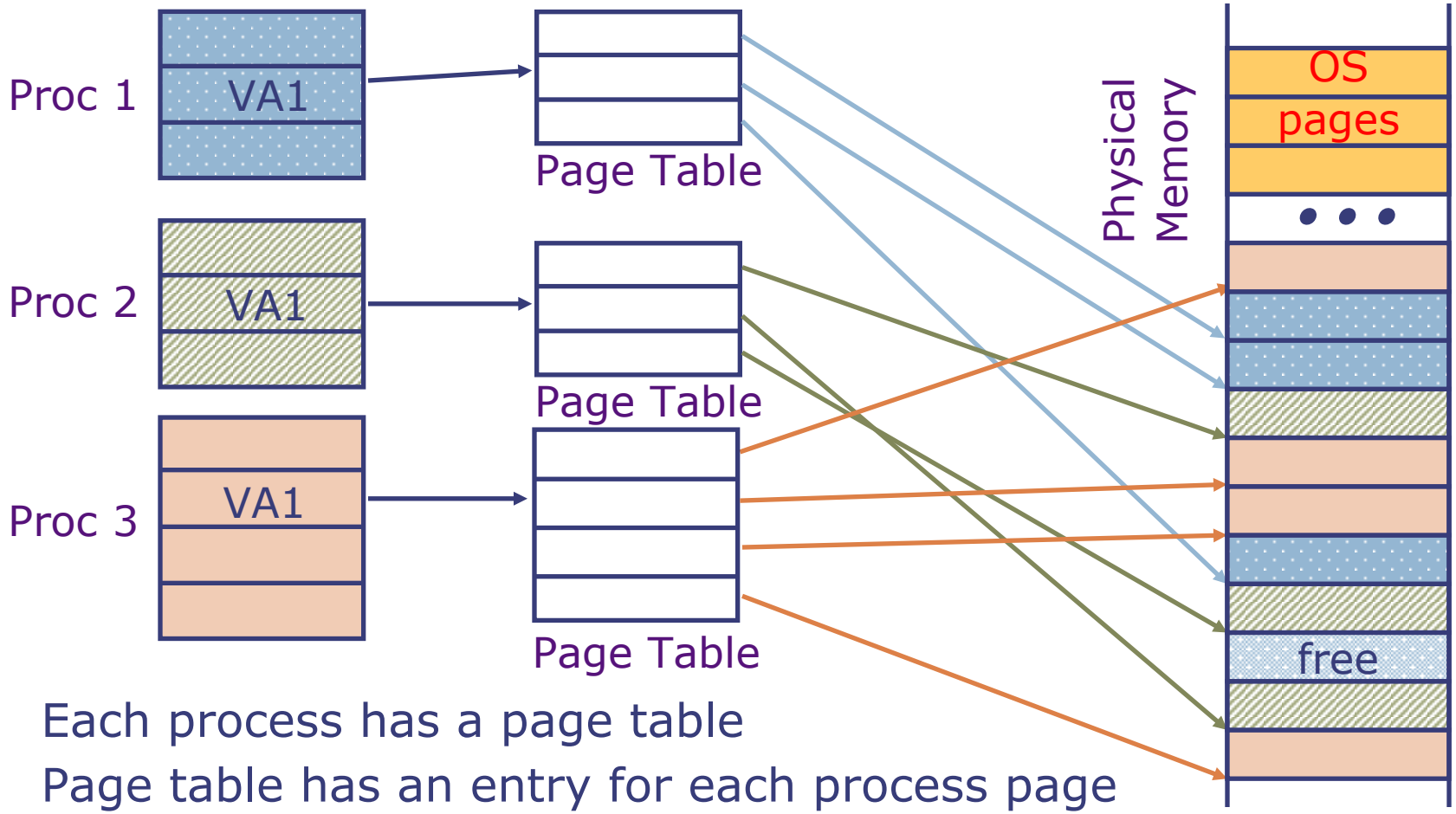
As processes start and end, storage is "fragmented". Therefore, at some point segments have to be moved around to compact the free space.

Paged Memory Systems

- Divide physical memory 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

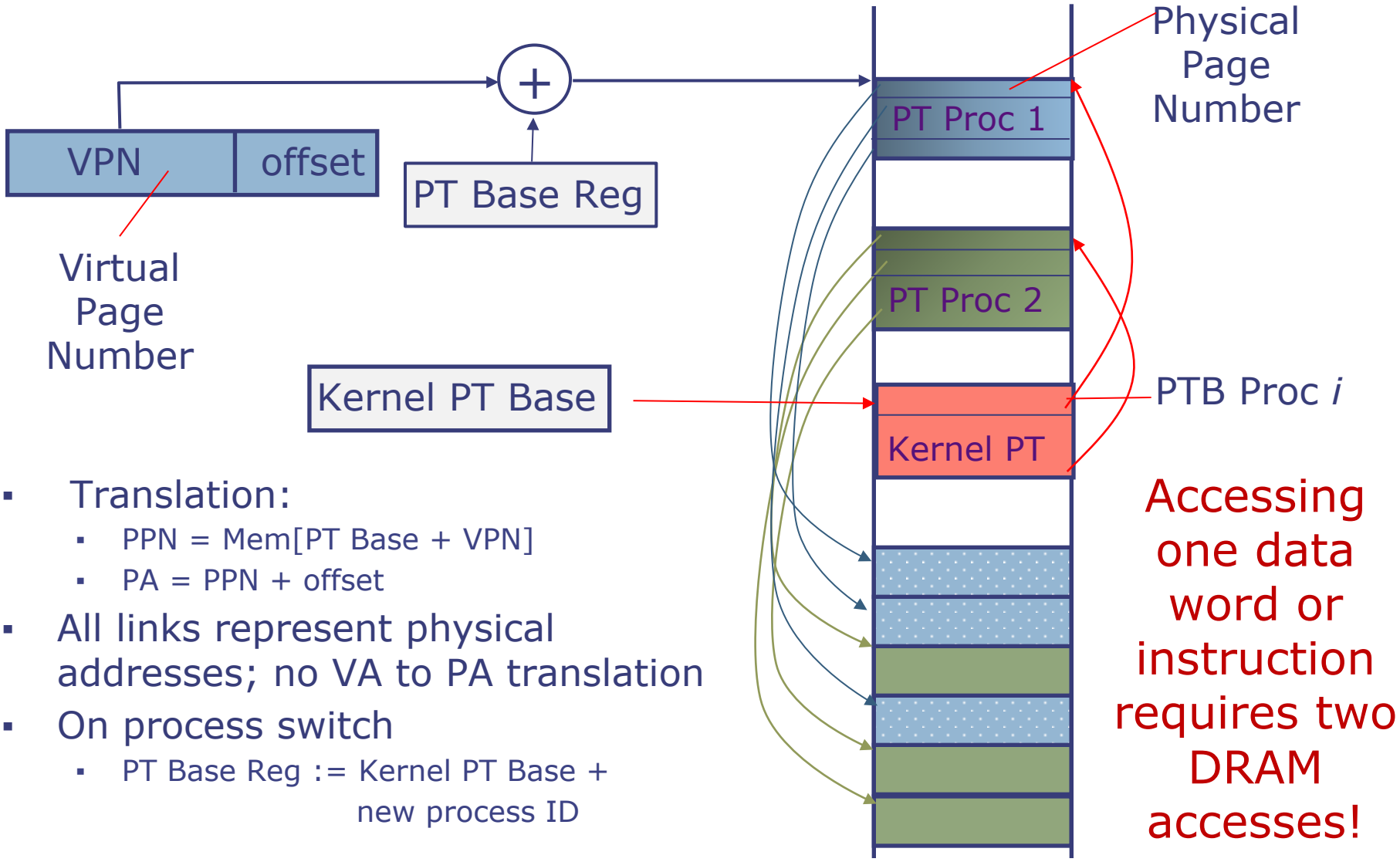
Pros and cons of paging vs segmentation?

- Paging: Fixed-size blocks
  - No external fragmentation O
  - Easy swapping O
  - Internal fragmentation X
  - Overhead of maintaining a large page table X

- Segmentation: Flexible block sizes
  - No translation overhead O
  - External fragmentation X

…where do we store the page tables?

Suppose Page Tables reside in memory

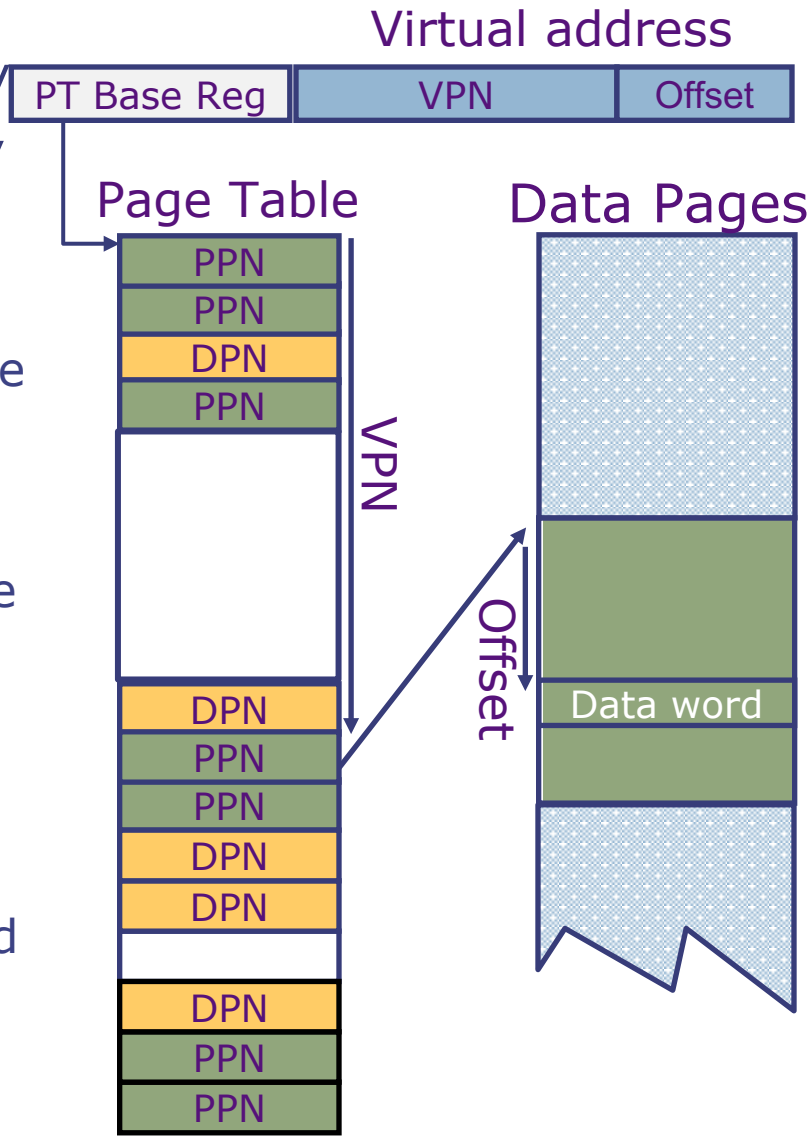


- Translation:
  - PPN = Mem[PT Base + VPN]
  - PA = PPN + offset
- All links represent physical addresses; no VA to PA translation
- On process switch
  - PT Base Reg := Kernel PT Base + new process ID

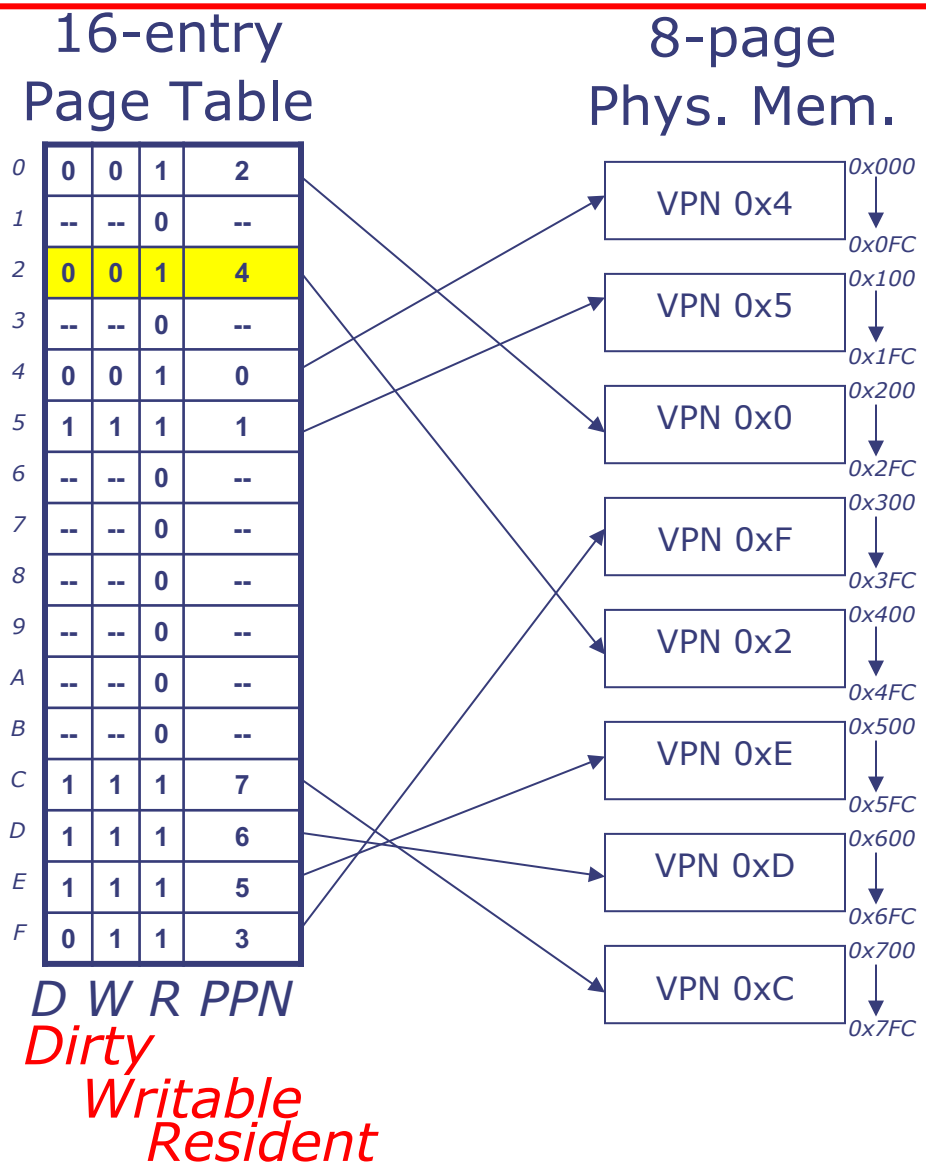
Accessing one data word or instruction requires two DRAM accesses!

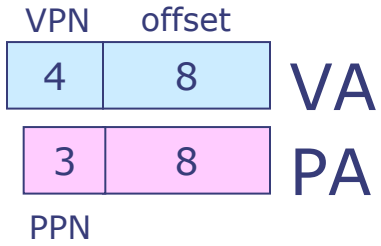
Demand Paging
Using main memory as a cache of disk

- 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 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





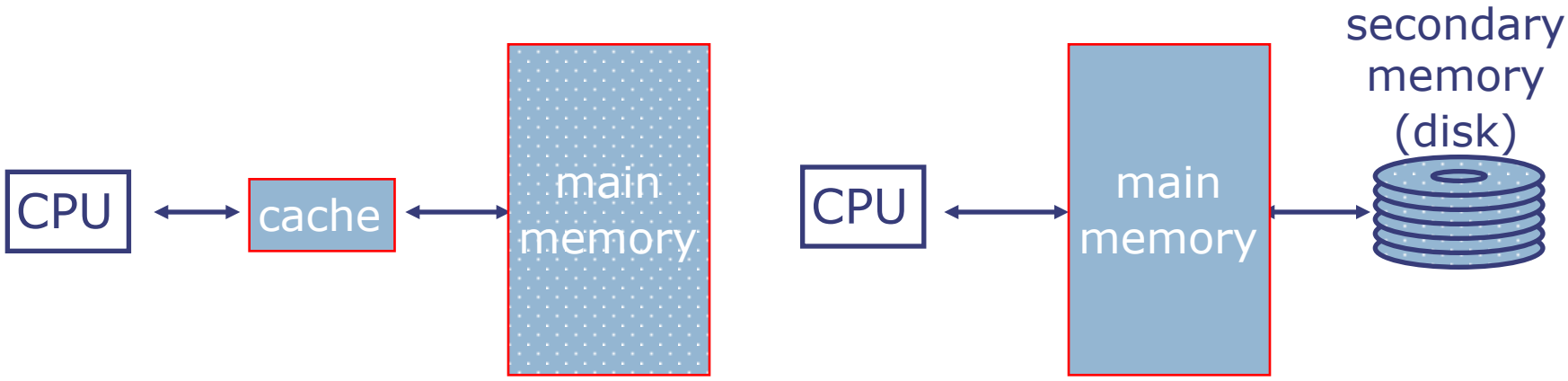
Setup:

- 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

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*

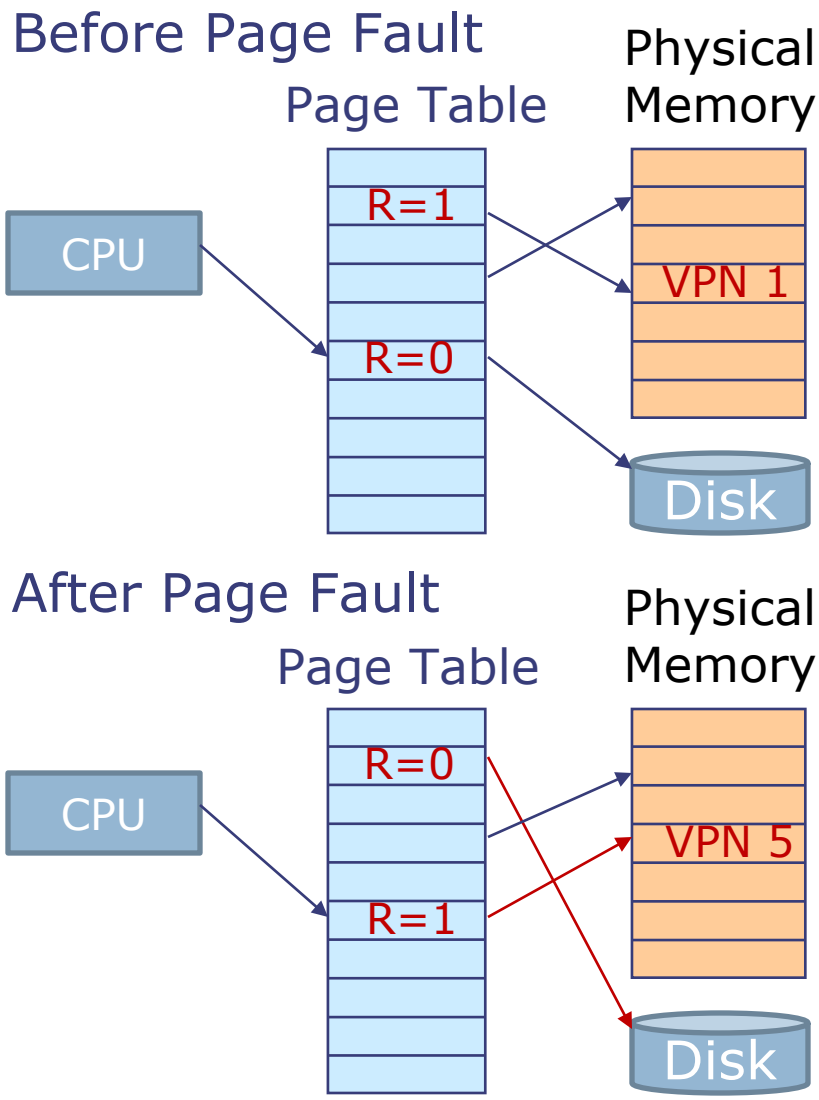
Demand paging

- 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*

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

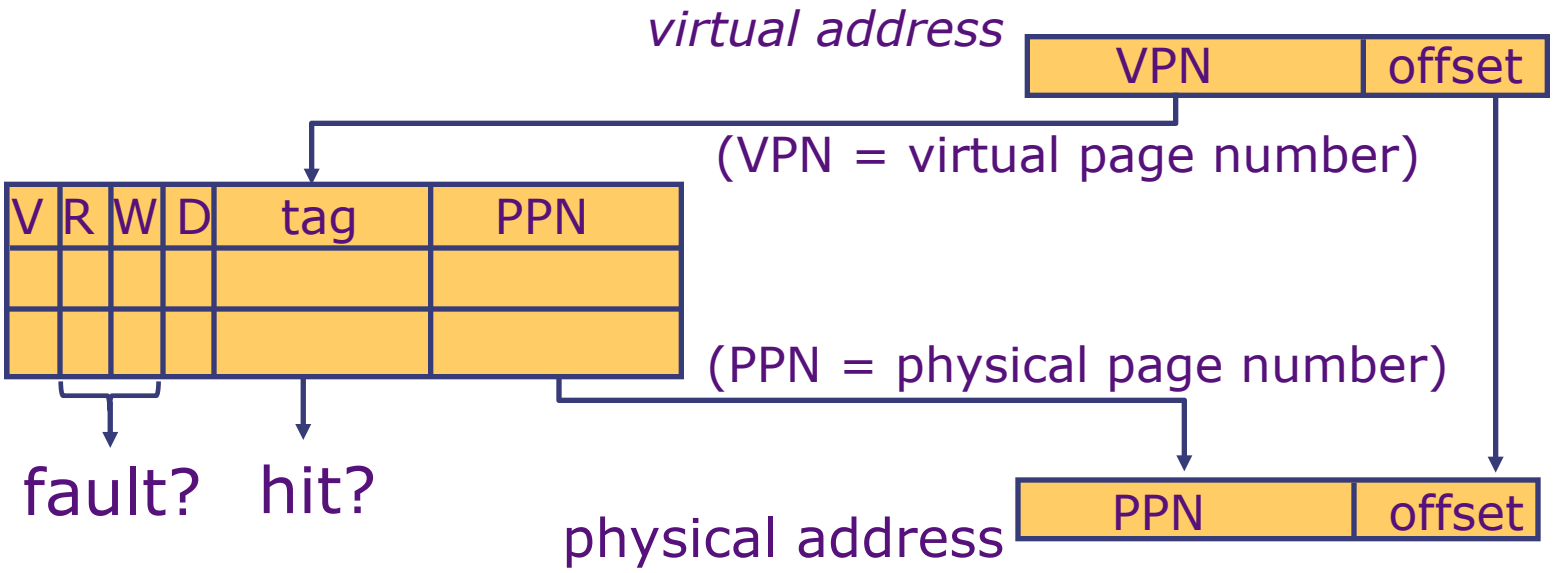


Translation Lookaside Buffer (TLB)

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*



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	V	R	D	PPN
0	1	0	0	7
6	1	1	1	2
1	1	1	1	9
3	1	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
	…		

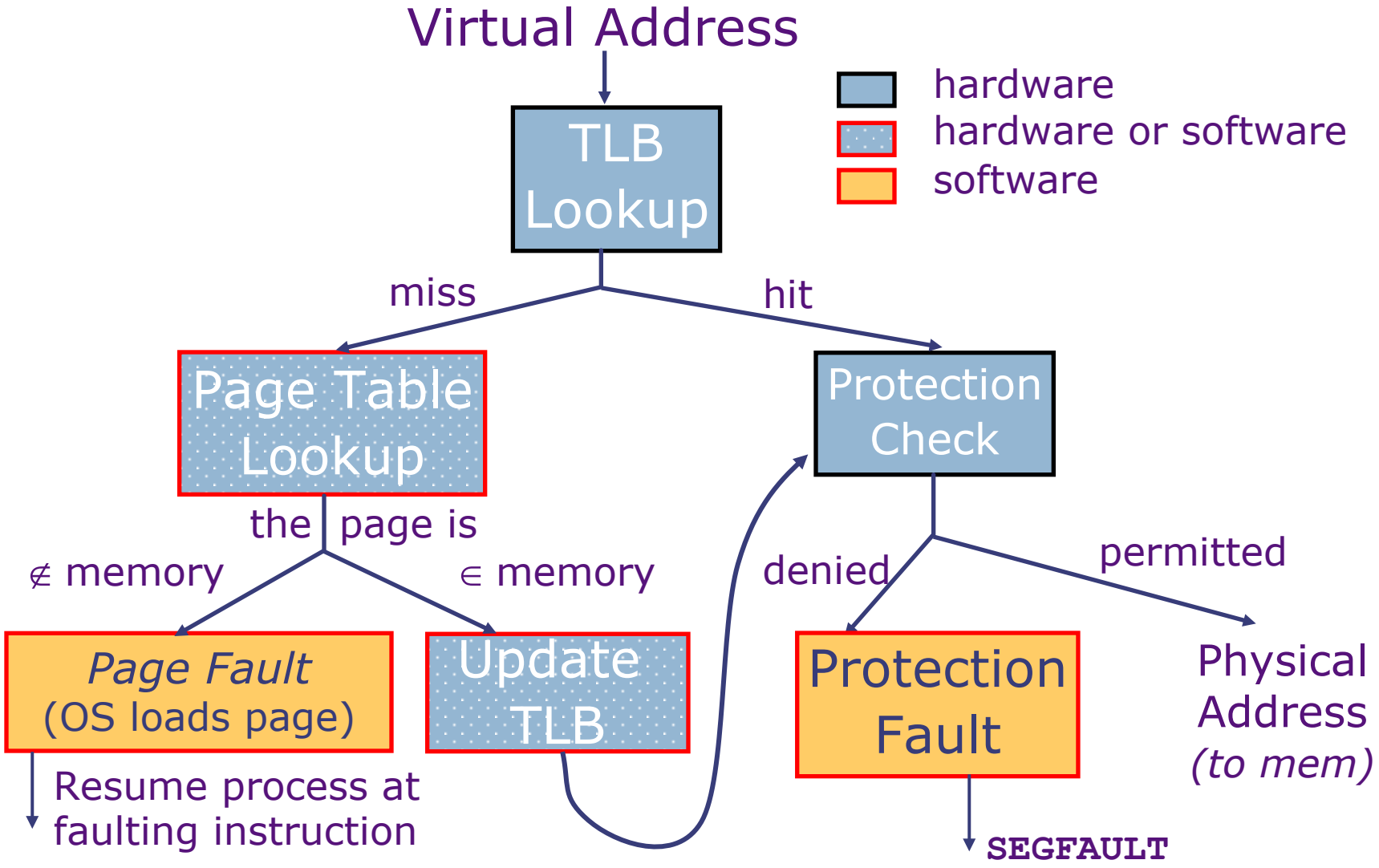
1. How many pages can be stored in physical memory at once? $2^{24}/2^{10}=2^{14}$
2. How many entries are there in the page table? $2^{32}/2^{10}=2^{22}$
3. How many bits per entry in the page table? (Assume each entry has PPN, resident bit, dirty bit) $14+1+1=16$
4. How many pages does page table take? $2*2^{22}/2^{10}=2^{13}$
6. What is the physical address for virtual address 0x1804? What components are involved in the translation? 0x804
7. Same for 0x1080
8. Same for 0x0FC

TLB Designs

- Typically 32-128 entries, 4 to 8-way set-associative
  - Modern processors use a hierarchy of TLBs (e.g., 128-entry L1 TLB + 2K-entry L2 TLB)
- Switching processes is expensive because TLB has to be flushed
  - Alternatively, include process ID in TLB entries to avoid flushing
- Handling a TLB miss: Look up the page table (a.k.a. "walk" the page table). If the page is in memory, load the VPN→PPN translation in the TLB. Otherwise, cause a page fault
  - Page faults are always handled in software
  - But page walks are usually handled in hardware using a *memory management unit (MMU)*
    - RISC-V, x86 access page table in hardware

Address Translation

Putting it all together



Summary

- Benefits of virtual memory:
  - Protection and privacy: Private address space per process
  - Demand paging: Can use main memory as a cache of disk

- Base and bound address translation: 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

Thank you!

Next lecture: Virtual Memory 2