Systems Programming

Virtual Memory: Systems

Byoungyoung Lee

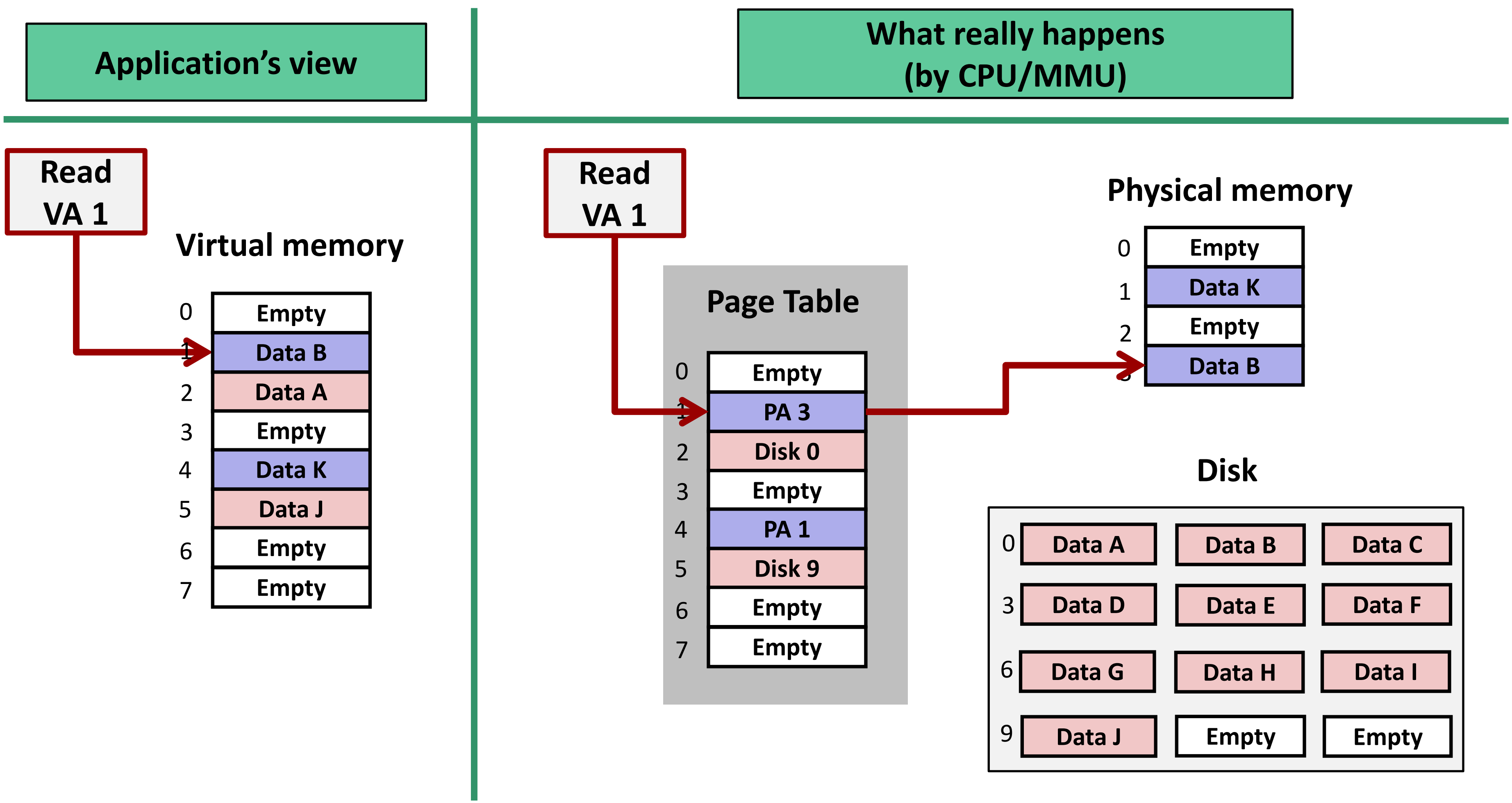
Seoul National University

byoungyoung@snu.ac.kr

https://lifeasageek.github.io

Lecture slides are prepared based on materials provided by CSAPP authors.

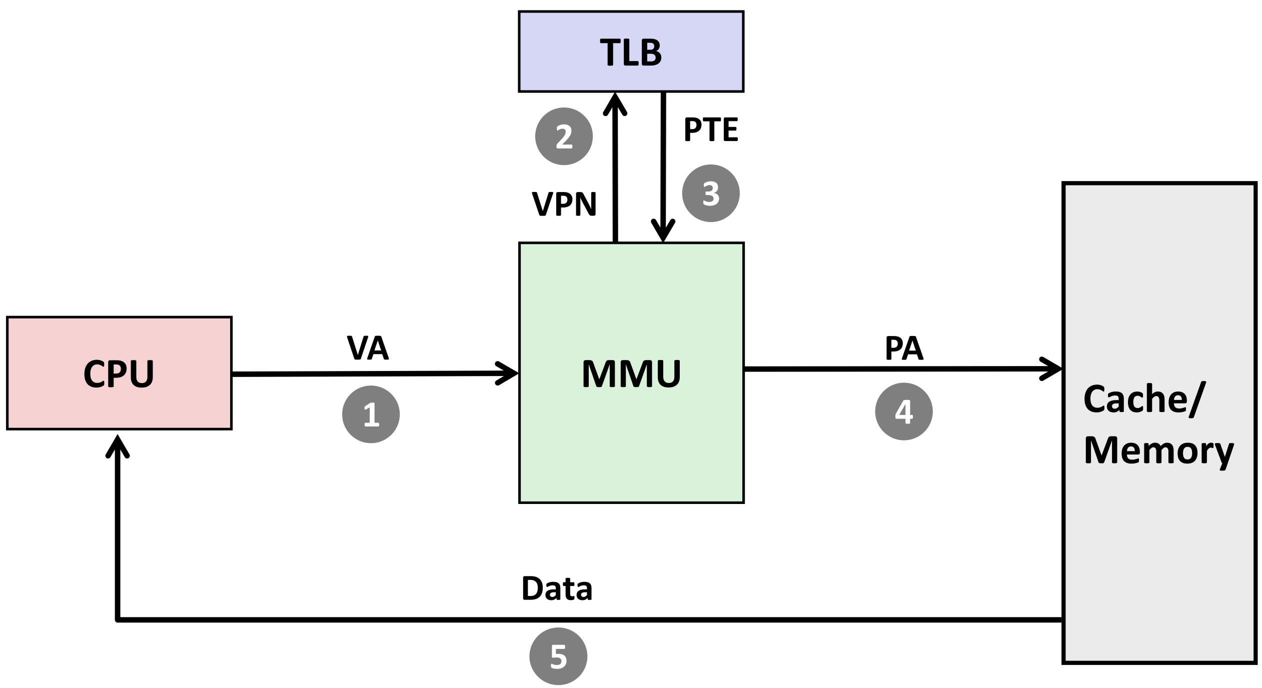
Review: Virtual Memory & Physical Memory



- A *page table* contains page table entries (PTEs) that map virtual pages to physical pages.

Review: Translation Lookaside Buffer (TLB)

- **A small cache of page table entries with fast access by MMU**

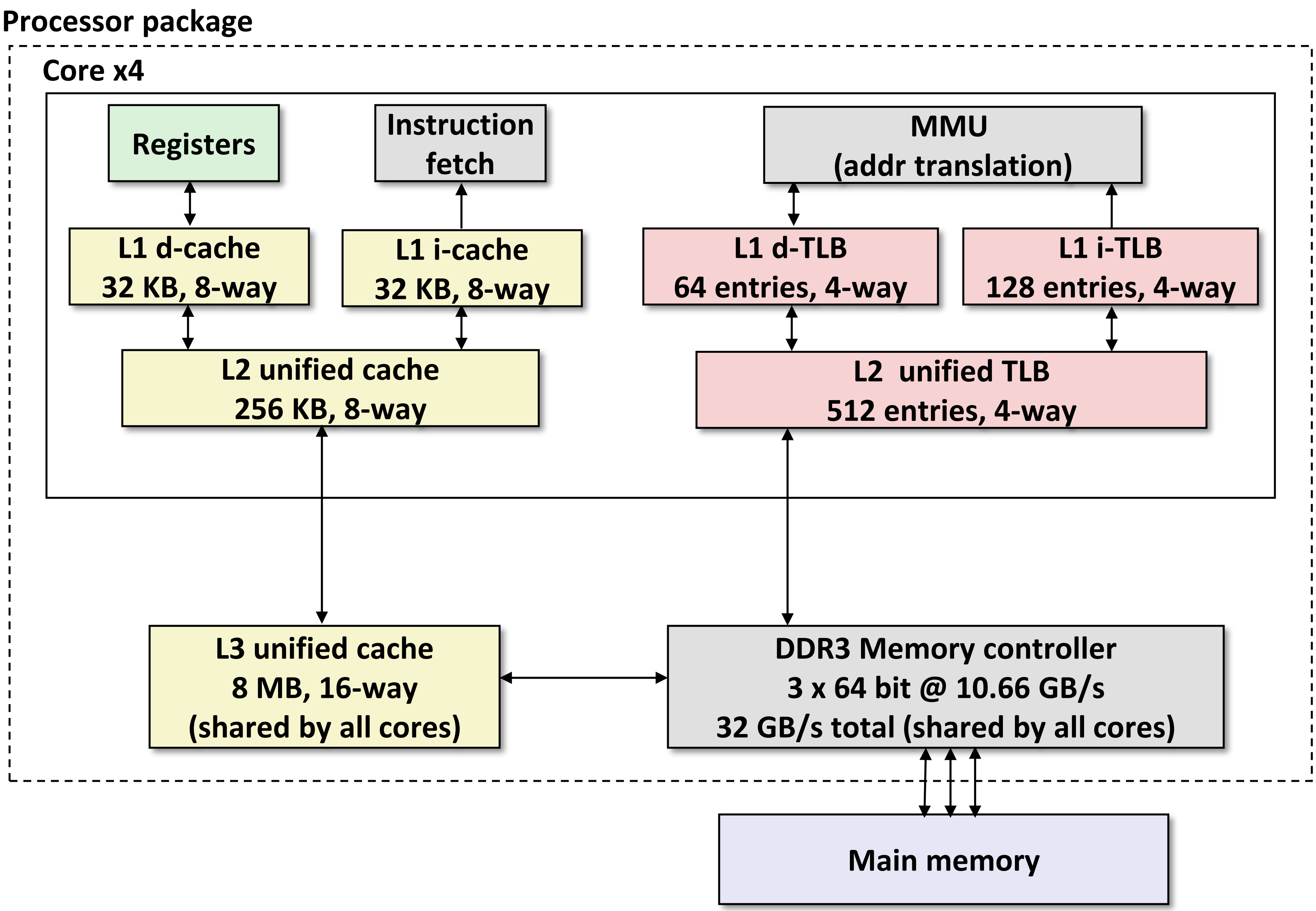


Typically, a TLB hit eliminates extra memory accesses required to do a page table lookup.

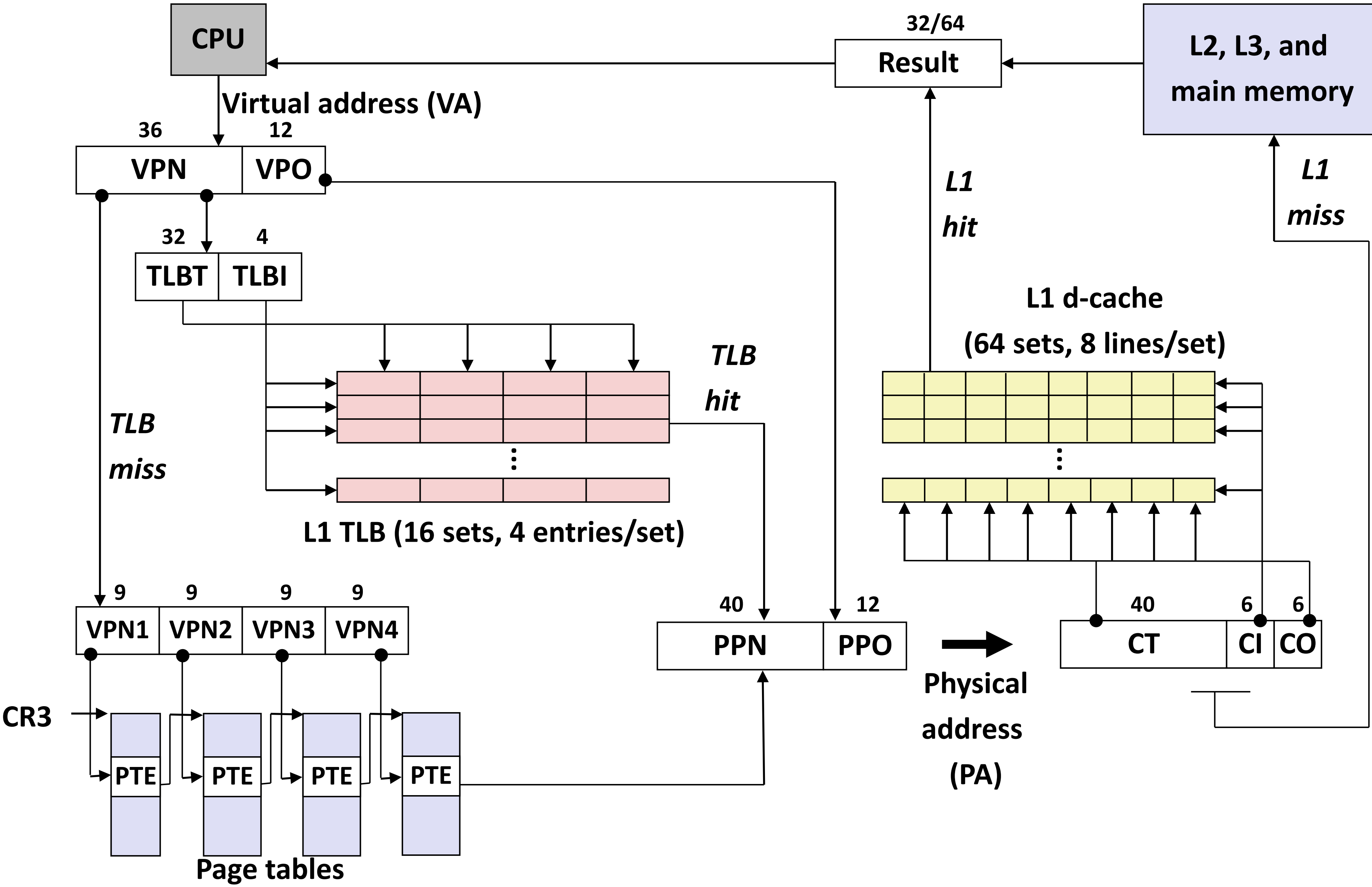
Today

- **Case study: Core i7/Linux memory system (CSAPP 9.7)**
- Memory mapping

Intel Core i7 Memory System



End-to-end Core i7 Address Translation



Core i7 Level 4 Page Table Entries

63	62 52	51 12	11 9	8	7	6	5	4	3	2	1	0
XD	Unused	Page physical base address	Unused	G		D	A	CD	WT	U/S	R/W	P=1

Available for OS (page location on disk)	P=0

Each entry references a 4K child page. Significant fields:

P: Child page is present in memory (1) or not (0)

R/W: Read-only or read-write access permission for child page

U/S: User or supervisor mode access

WT: Write-through or write-back cache policy for this page

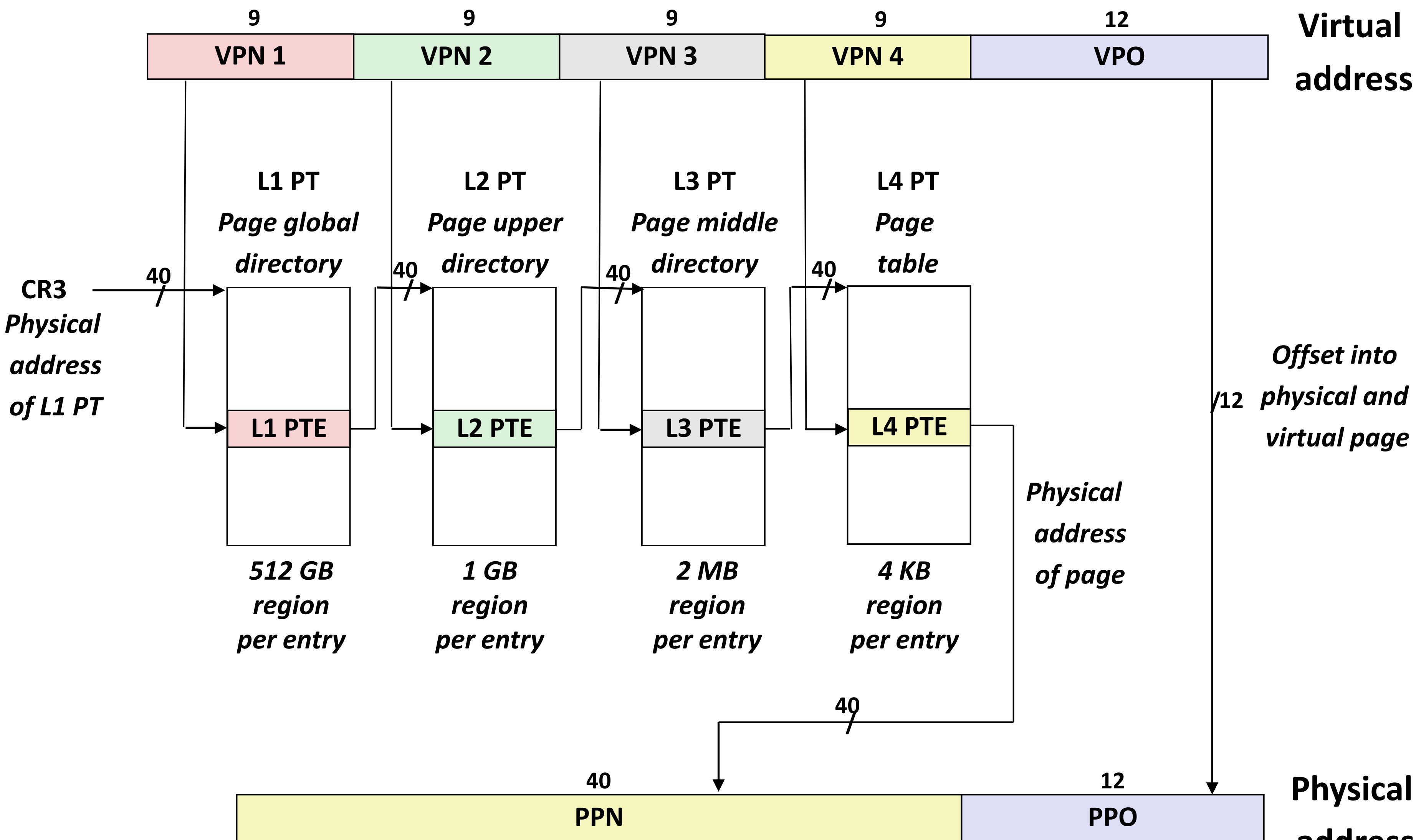
A: Reference bit (set by MMU on reads and writes, cleared by software)

D: Dirty bit (set by MMU on writes, cleared by software)

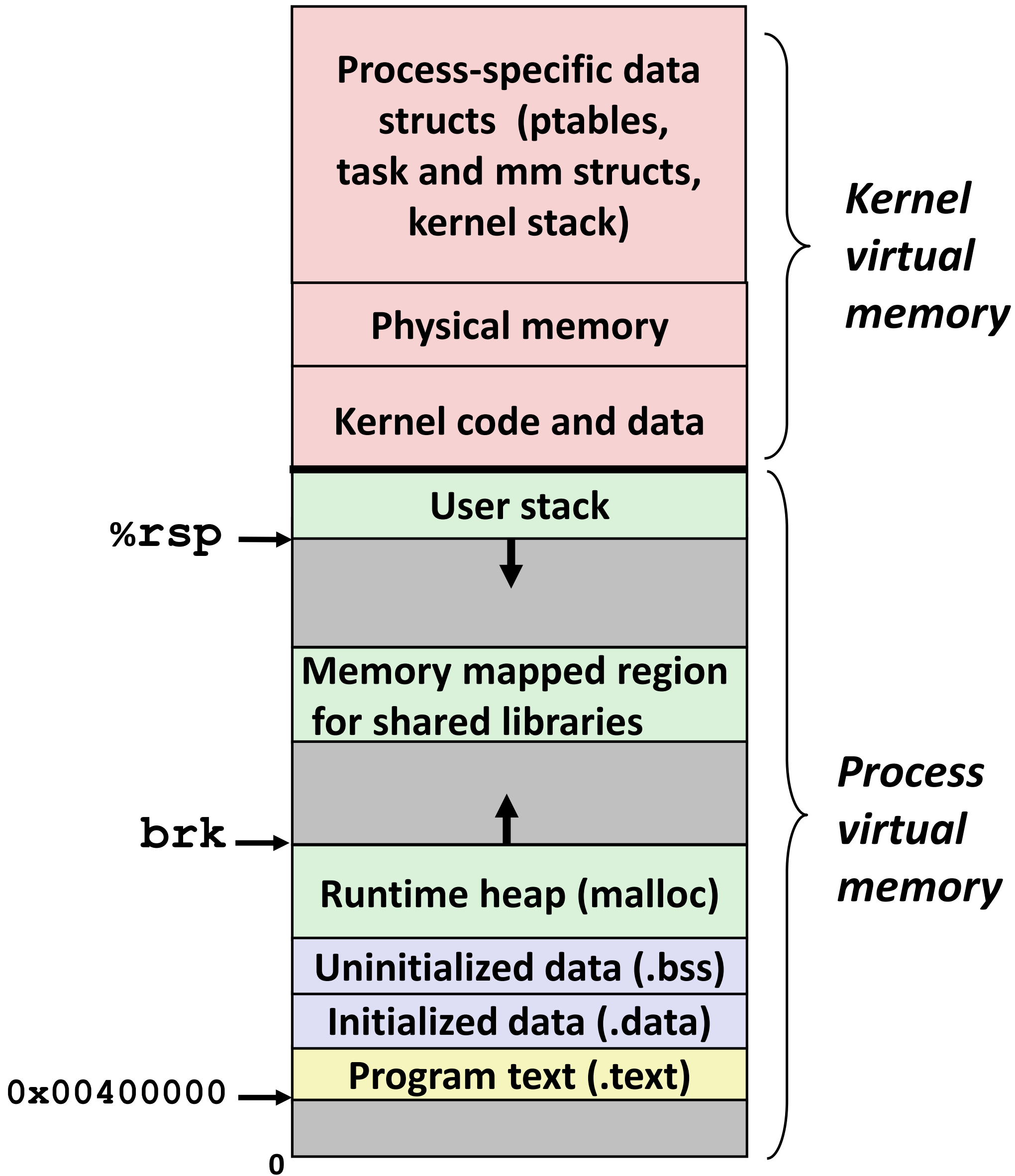
Page physical base address: 40 most significant bits of physical page address
(forces pages to be 4KB aligned)

XD: Disable or enable instruction fetches from this page.

Core i7 Page Table Translation



Virtual Address Space of a Linux Process



Today

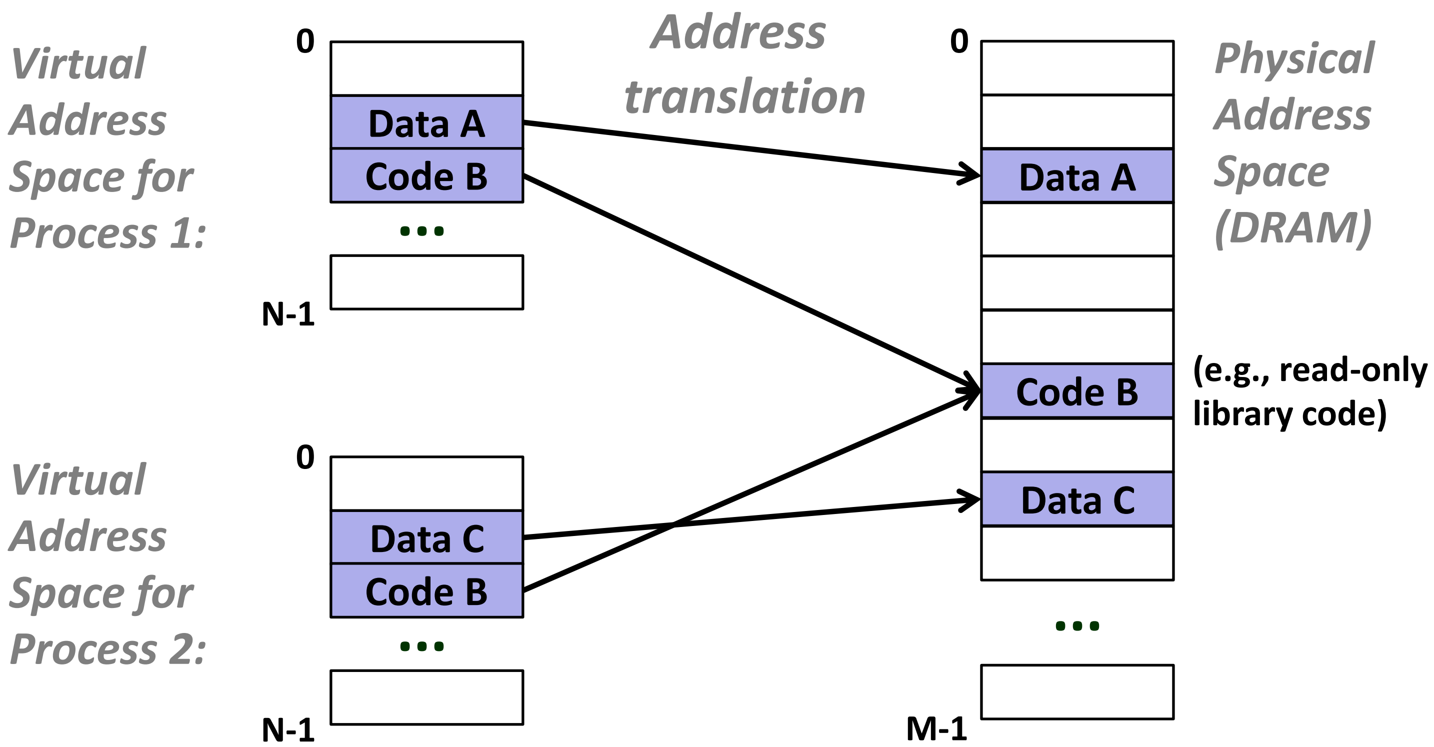
- Case study: Core i7/Linux memory system (CSAPP 9.7)
- **Memory mapping**

Memory Mapping

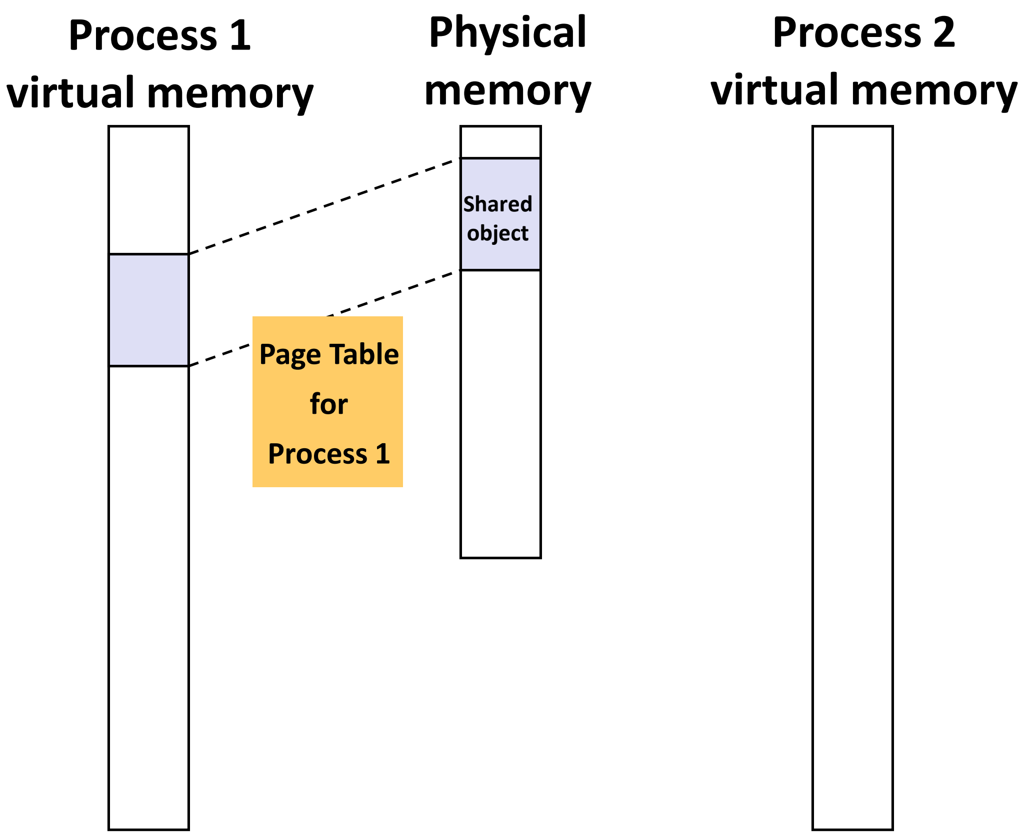
- **VM areas are be *backed by* (i.e., get its initial page from) :**
  - ***Regular file*** on disk (e.g., an executable object file)
    - Initial page bytes come from a section of a file
  - ***Anonymous file*** (e.g., nothing)
    - First fault will allocate a physical page full of 0's (***demand-zero page***)
    - Once the page is written to (***dirtied***), it is like any other page
- **Dirty pages are copied back and forth between memory and a special *swap file*.**

Review: Memory Management & Protection

- **Code and data can be isolated or shared among processes**

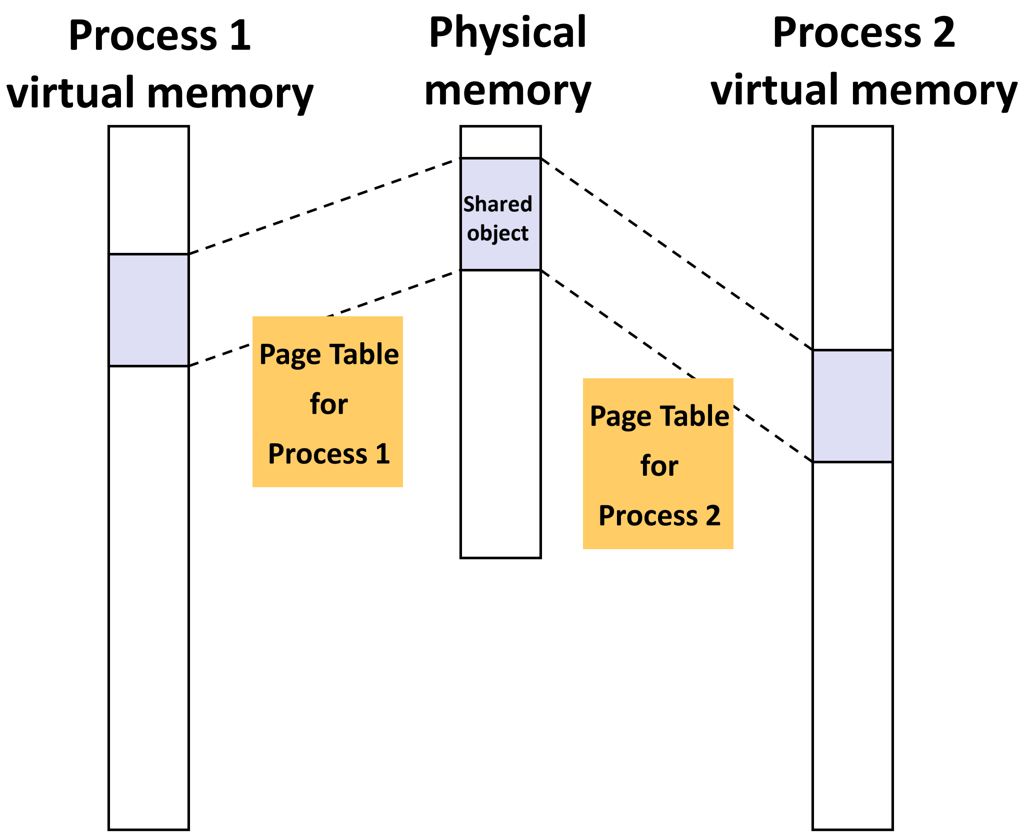


Sharing Revisited: Shared Objects



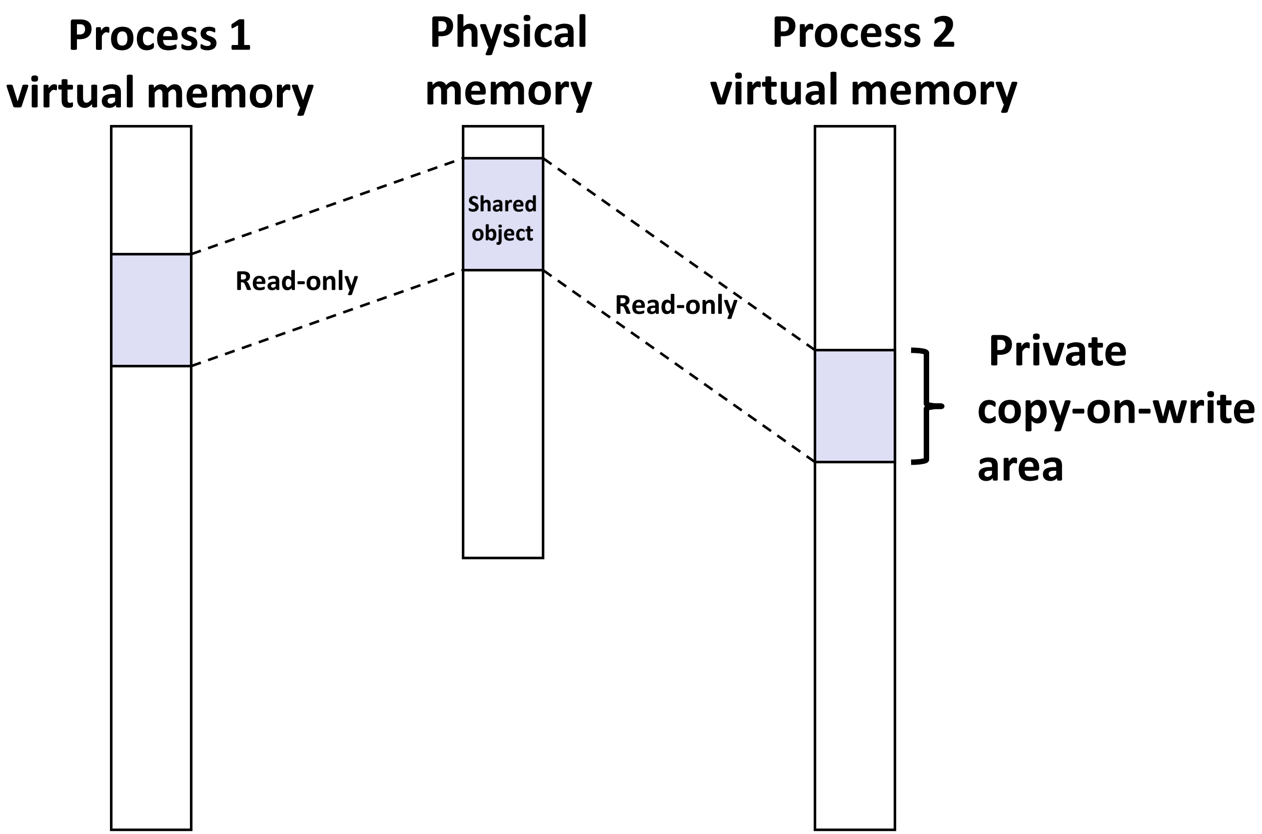
- Process 1 maps the shared object.

Sharing Revisited: Shared Objects



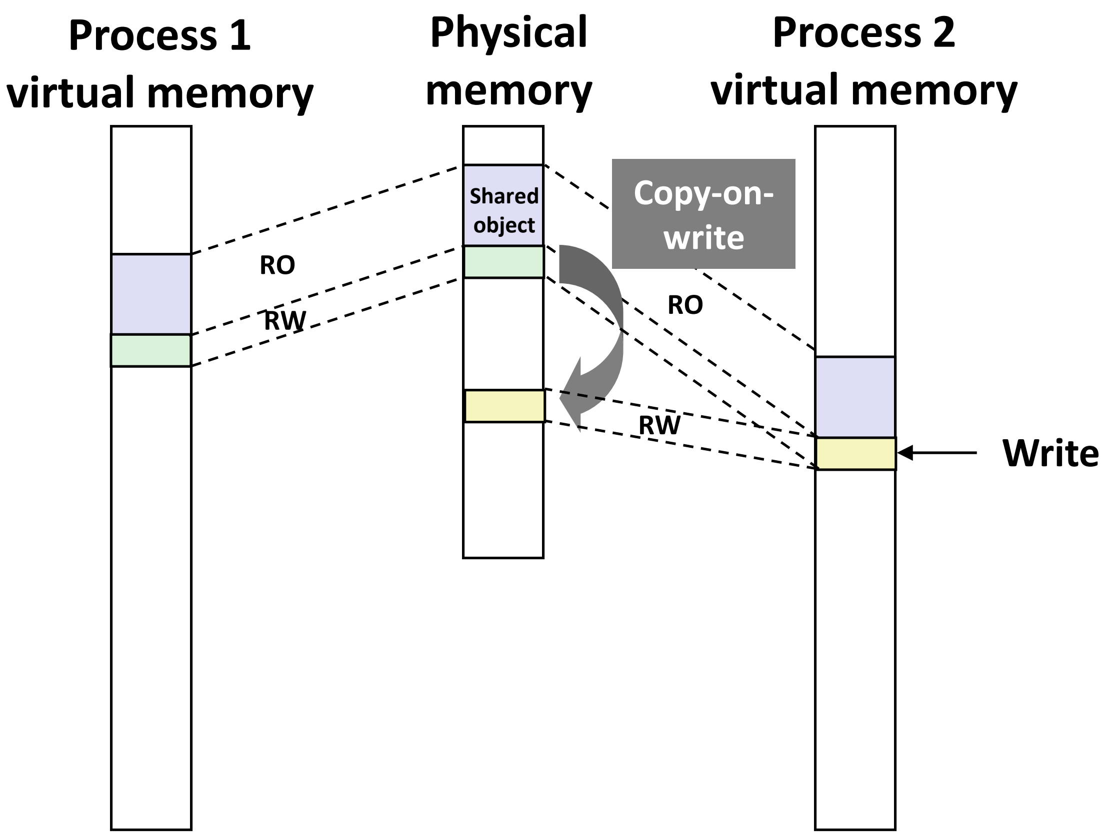
- Process 2 maps the same shared object.
- Notice how the virtual addresses can be different.

Sharing Revisited: Private Copy-on-write (COW) Objects



- Two processes mapping a ***private copy-on-write (COW)*** object
- Area flagged as private copy-on-write
- PTEs in private areas are flagged as read-only

Sharing Revisited: Private Copy-on-write (COW) Objects



- Instruction writing to private page triggers protection fault.
- Handler creates new R/W page, and copies the data (copy-on-write)
- Instruction restarts upon handler return.
- Conclusion: Copying deferred as long as possible!

Finding Shareable Pages

- **Kernel Same-Page Merging (deduplication)**
  - OS scans through all of physical memory, looking for duplicate pages
  - When found, merge into single copy, marked as copy-on-write
  - Implemented in Linux kernel in 2009
  - Especially useful in cloud machines, running many VMs

  - Many security issues have been found related to deduplication ☹

User-Level Memory Mapping

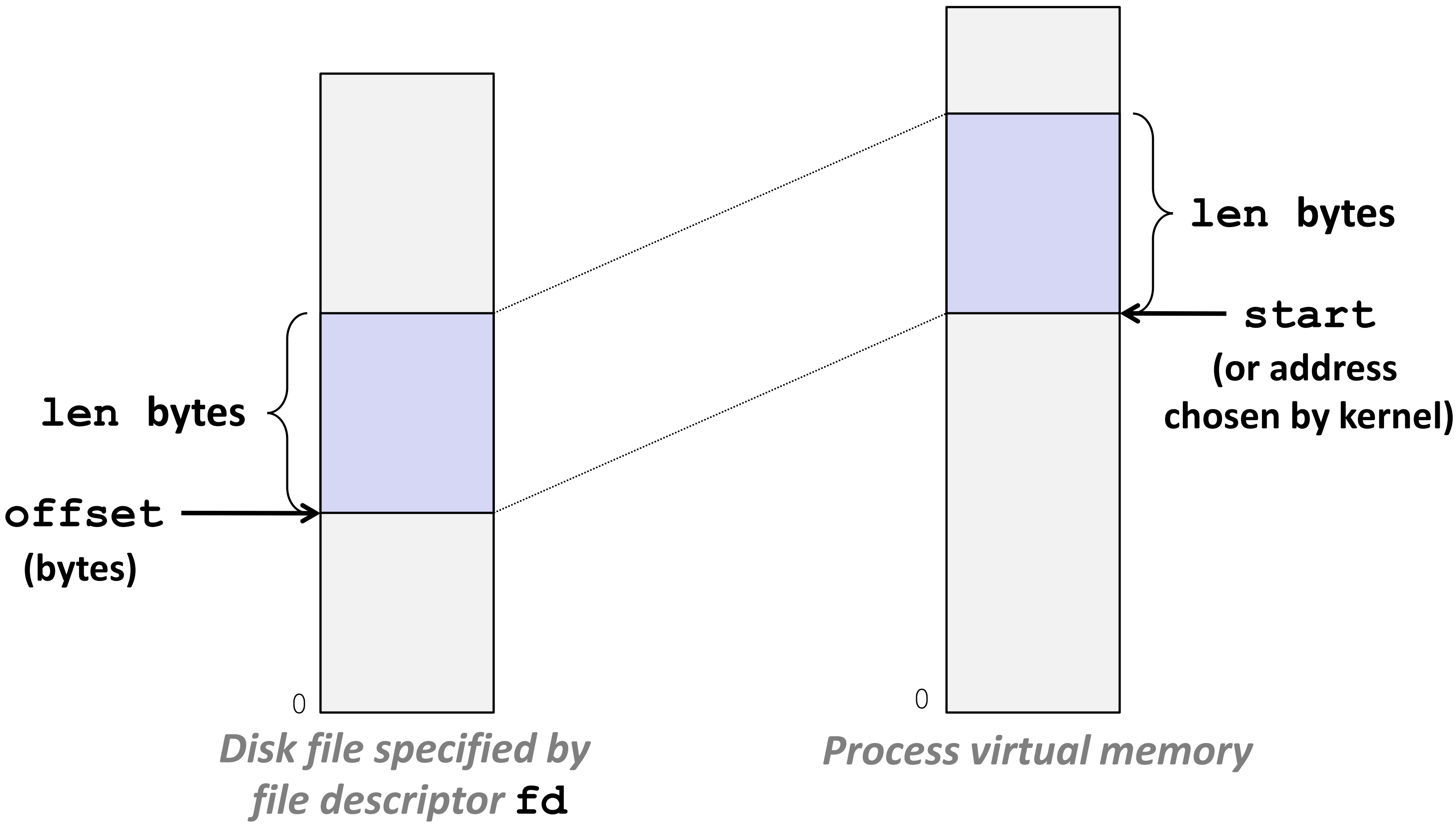
```
void *mmap(void *start, int len,
           int prot, int flags, int fd, int offset)
```

- **Map `len` bytes starting at offset `offset` of the file specified by file description `fd`, preferably at address `start`**
  - **`start:`** may be 0 for "pick an address"
  - **`prot`**: PROT_READ, PROT_WRITE, PROT_EXEC, ...
  - **`flags`**: MAP_ANON, MAP_PRIVATE, MAP_SHARED, ...

- **Return a pointer to start of mapped area (may not be `start`)**

User-Level Memory Mapping

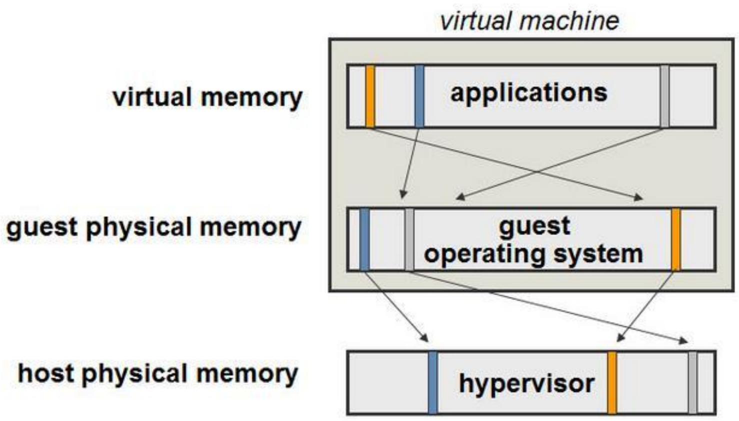
```
void *mmap(void *start, int len,
           int prot, int flags, int fd, int offset)
```

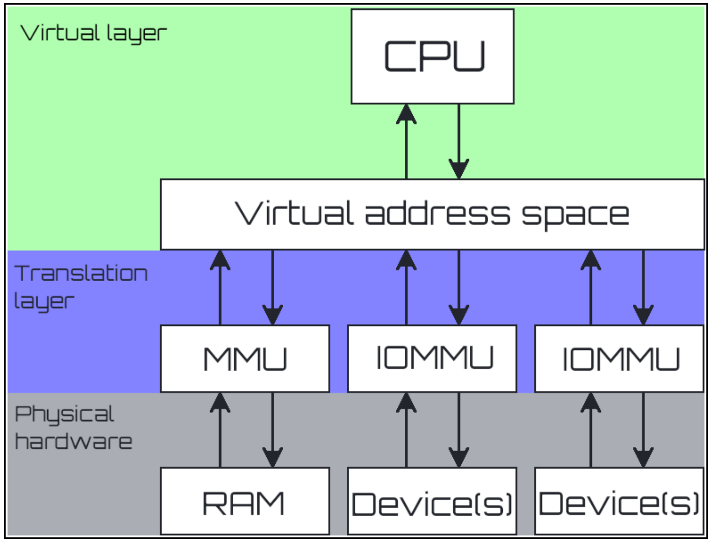


Uses of mmap

- **Reading big files**
  - Uses paging mechanism to bring files into memory
- **Shared data structures**
  - When call with `MAP_SHARED` flag
    - Multiple processes have access to same region of memory
    - Risky!
- **File-based data structures**
  - E.g., database
  - Give `prot` argument `PROT_READ | PROT_WRITE`
  - When unmap region, file will be updated via write-back
  - Can implement load from file / update / write back to file

Virtual Memory in Real-world





https://raddinox.com/gpu-passthrough-to-windows-11-using-libvirt-qemu

Summary

- **VM requires hardware support**
  - Exception handling mechanism
  - TLB
  - Various control registers
- **VM requires OS support**
  - Managing page tables
  - Implementing page replacement policies
  - Managing file system
- **VM enables many capabilities**
  - Loading programs from memory
  - Providing memory protection