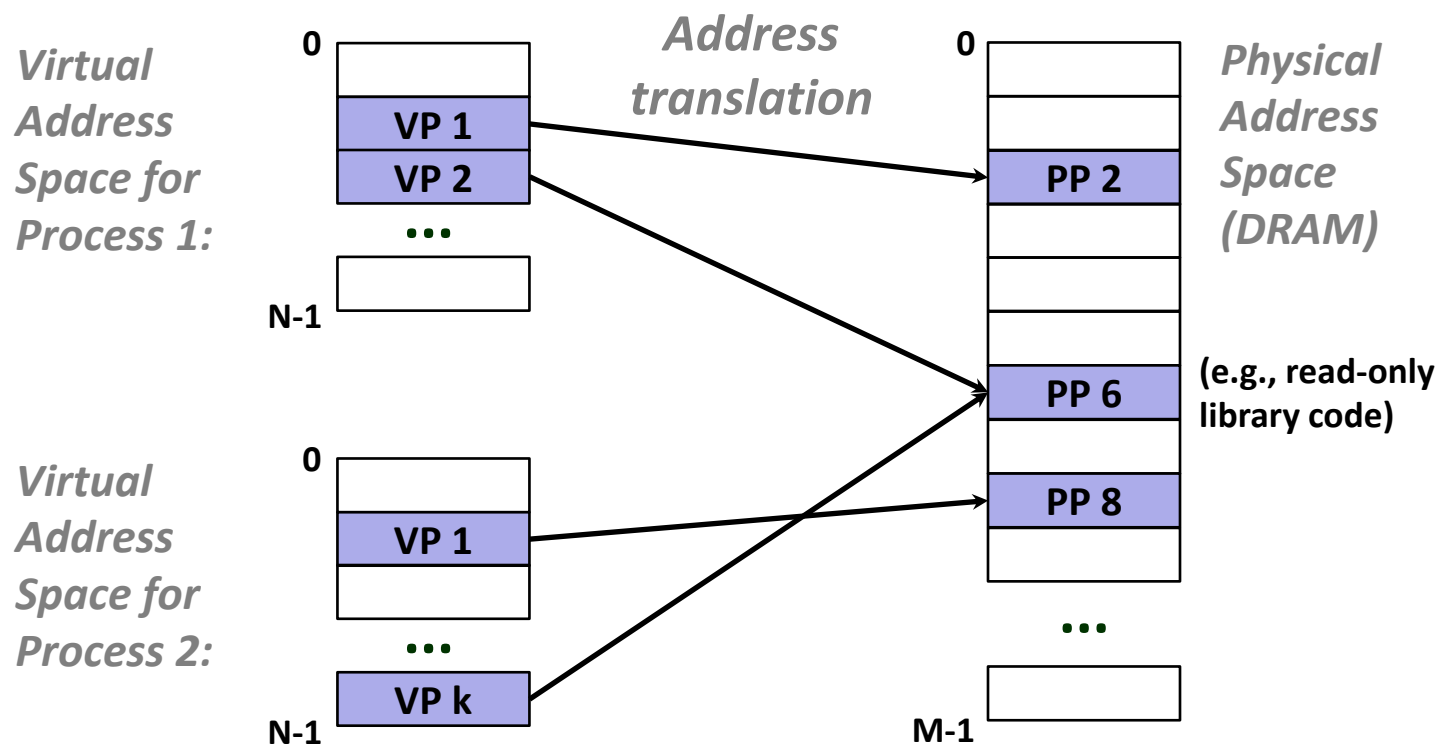


# Virtual Memory: Details

15-213/15-513: Introduction to Computer Systems  
18<sup>th</sup> Lecture, July 13, 2022

# Review: Virtual Addressing

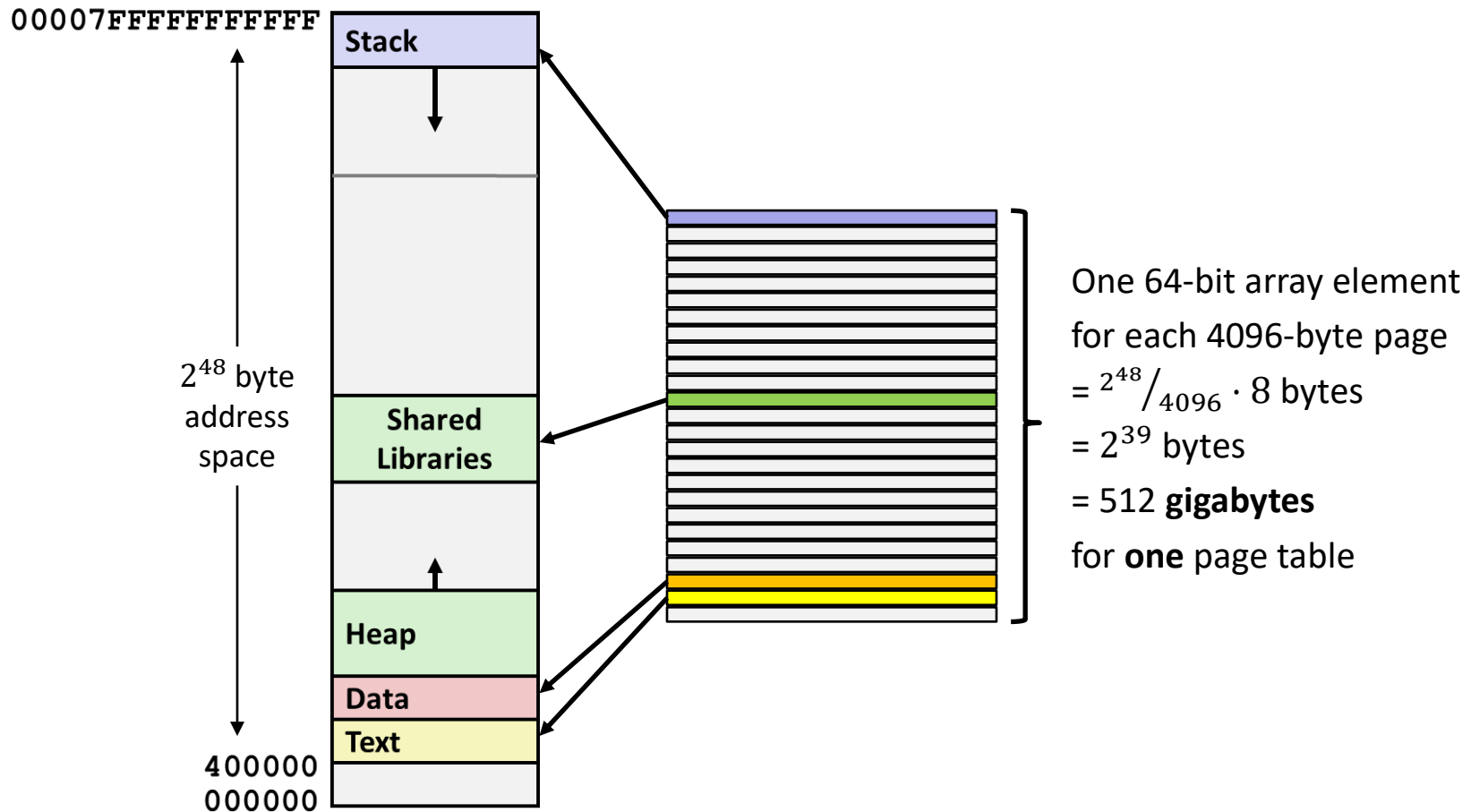
- Each process has its own *virtual address space*
- *Page tables* map virtual to physical addresses
- Physical memory can be shared among processes



# Today

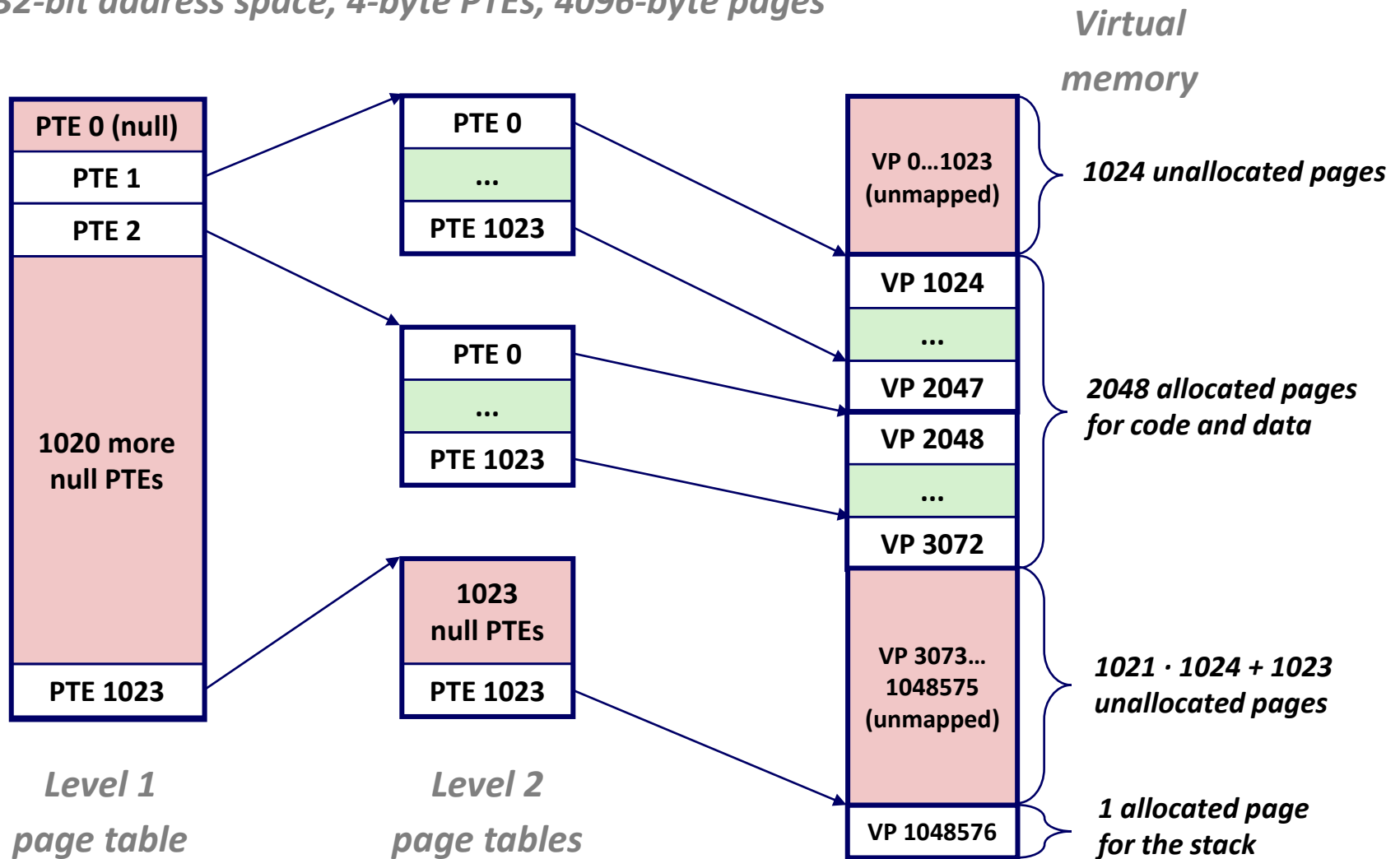
- **Multi-level page tables**
- **Translation lookaside buffers**
- **Activity 1**
- **Concrete examples of virtual memory systems**
  - “Simple memory system” from CSAPP 9.6.4
  - Intel Core i7
- **Activity 2**
- **Nifty things virtual memory makes possible**
  - Paging/swapping (disk as extra RAM)
  - Memory-mapped files (RAM as cache for disk)
  - Copy-on-write sharing
- **Activity 3**

# The problem (with one-level page tables)

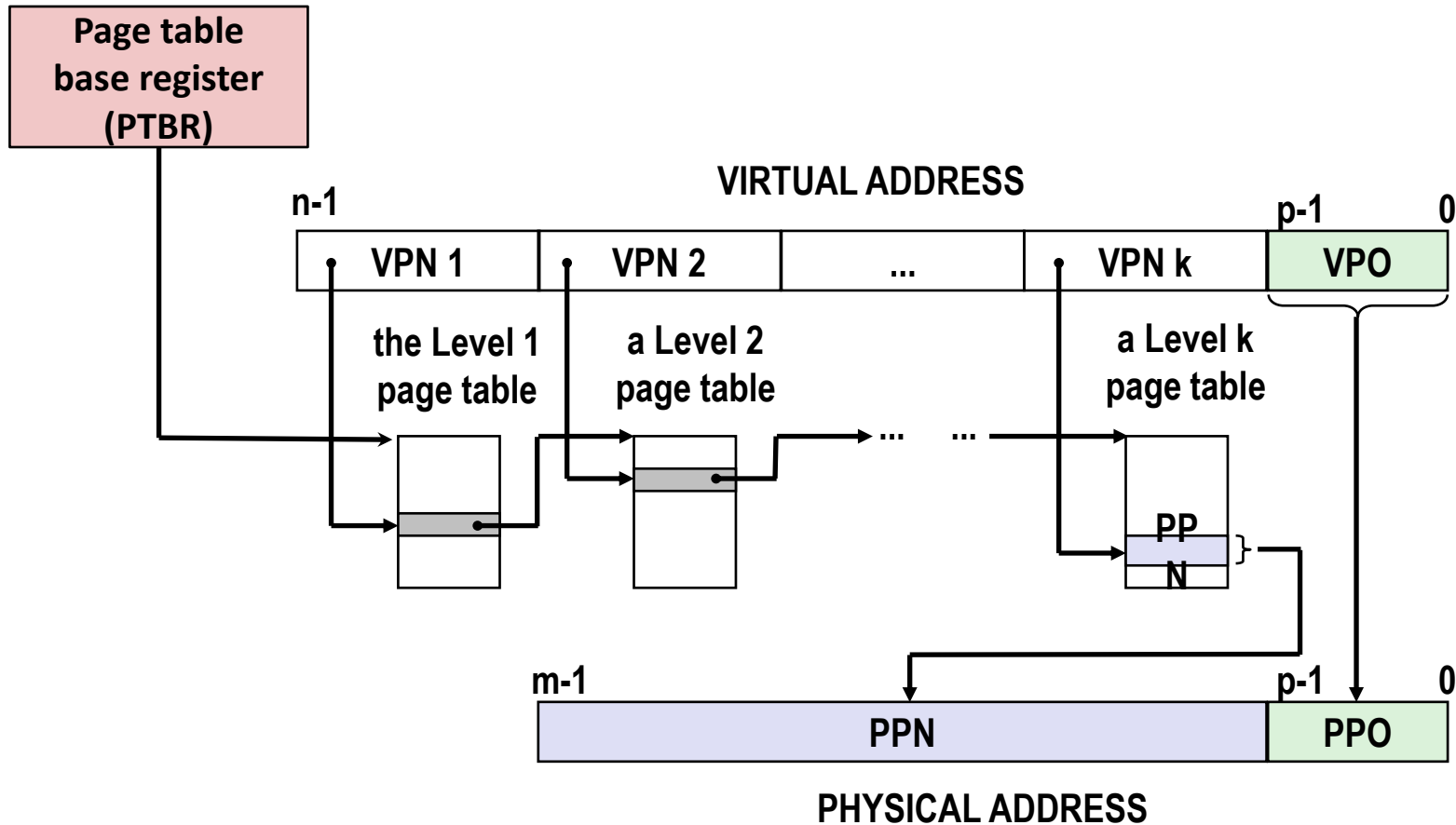


# A Two-Level Page Table Hierarchy

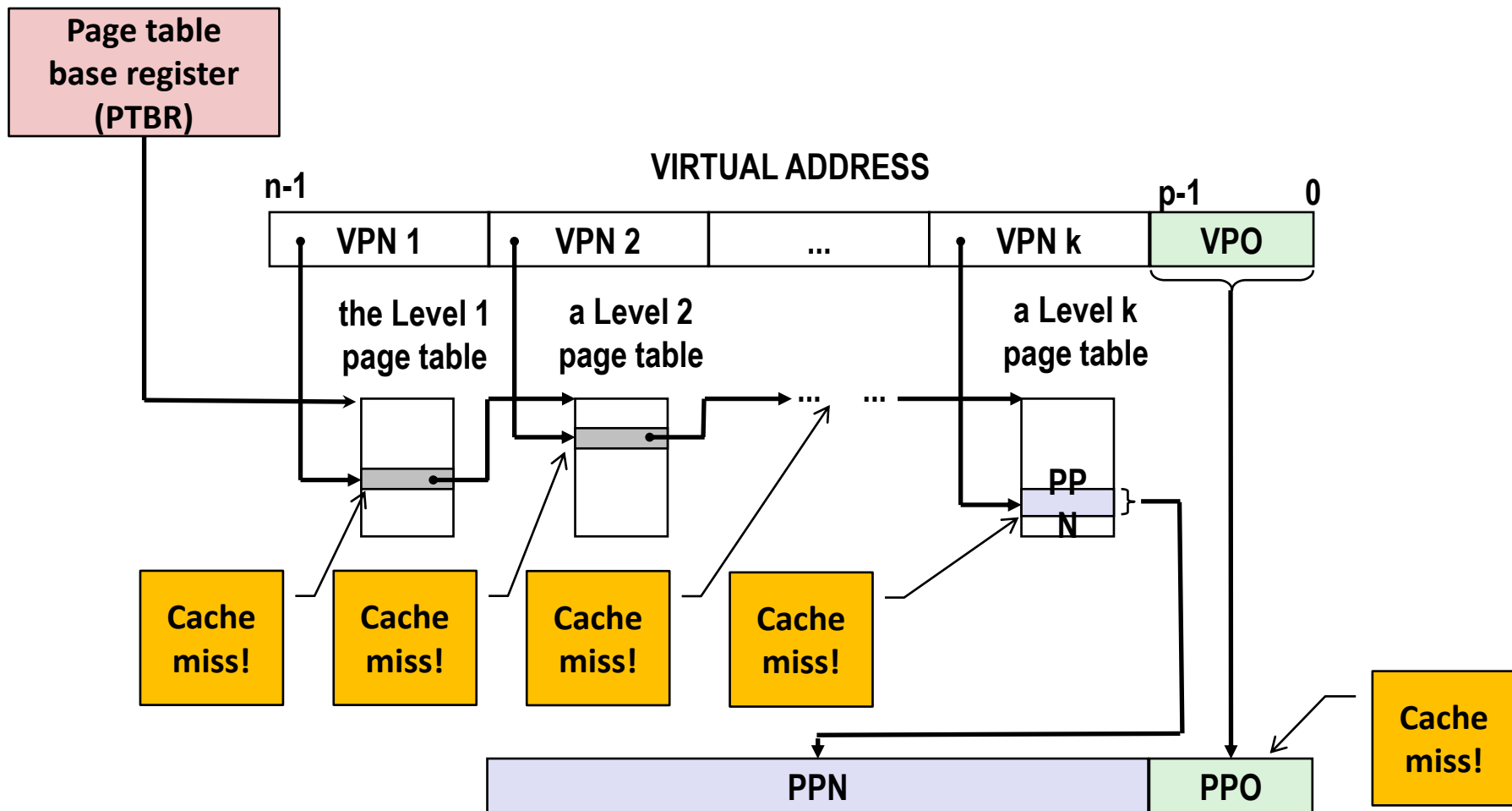
32-bit address space, 4-byte PTEs, 4096-byte pages



# Translating with a k-level Page Table



# The problem (with k-level page tables)



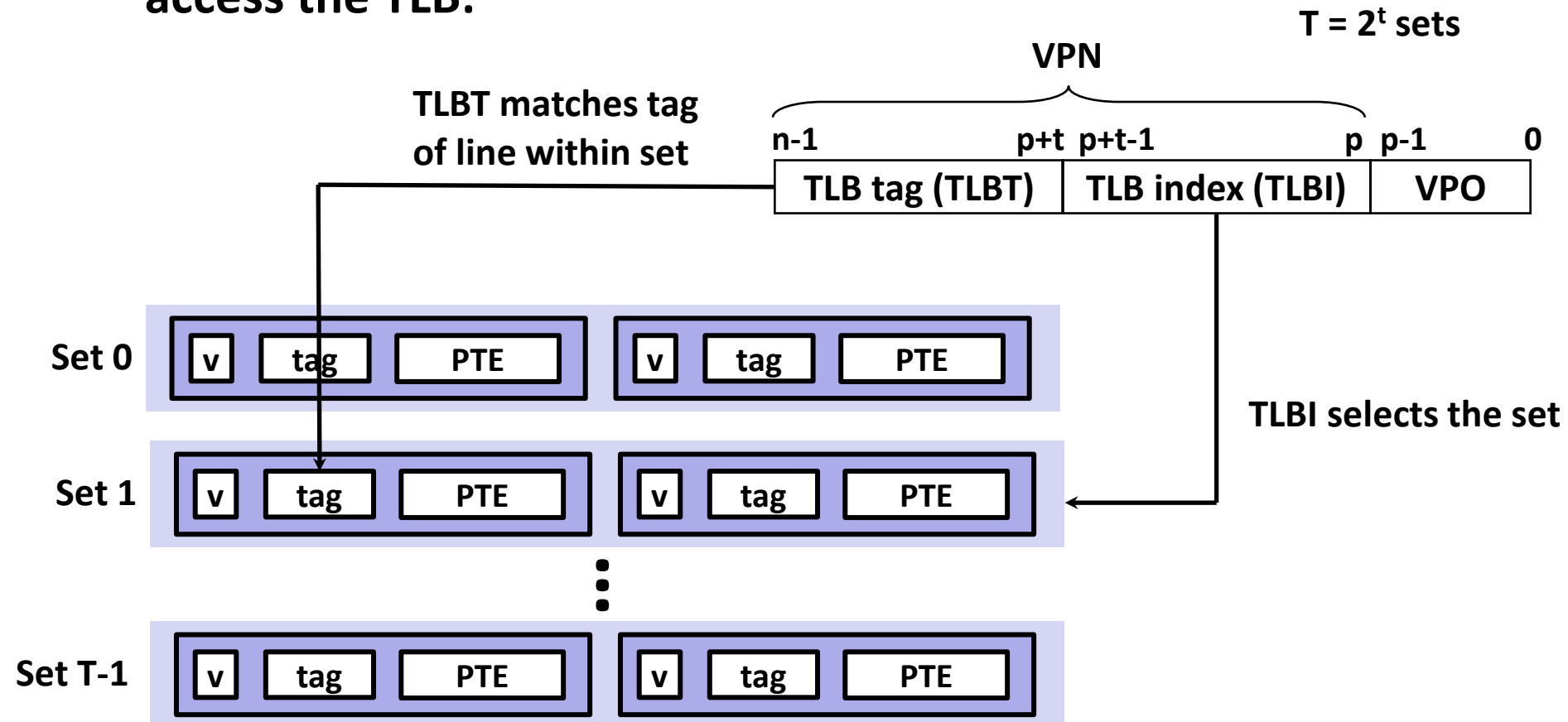
# Speeding up Translation with a TLB

- **Page table entries (PTEs) are cached like any other memory word**
  - PTEs may be evicted by other data references
  - PTE hit still costs cache delay
- **Solution: *Translation Lookaside Buffer* (TLB)**
  - Dedicated cache for page table entries
  - TLB hit = page table not consulted
  - Can be fairly small: one TLB entry covers 4k or more

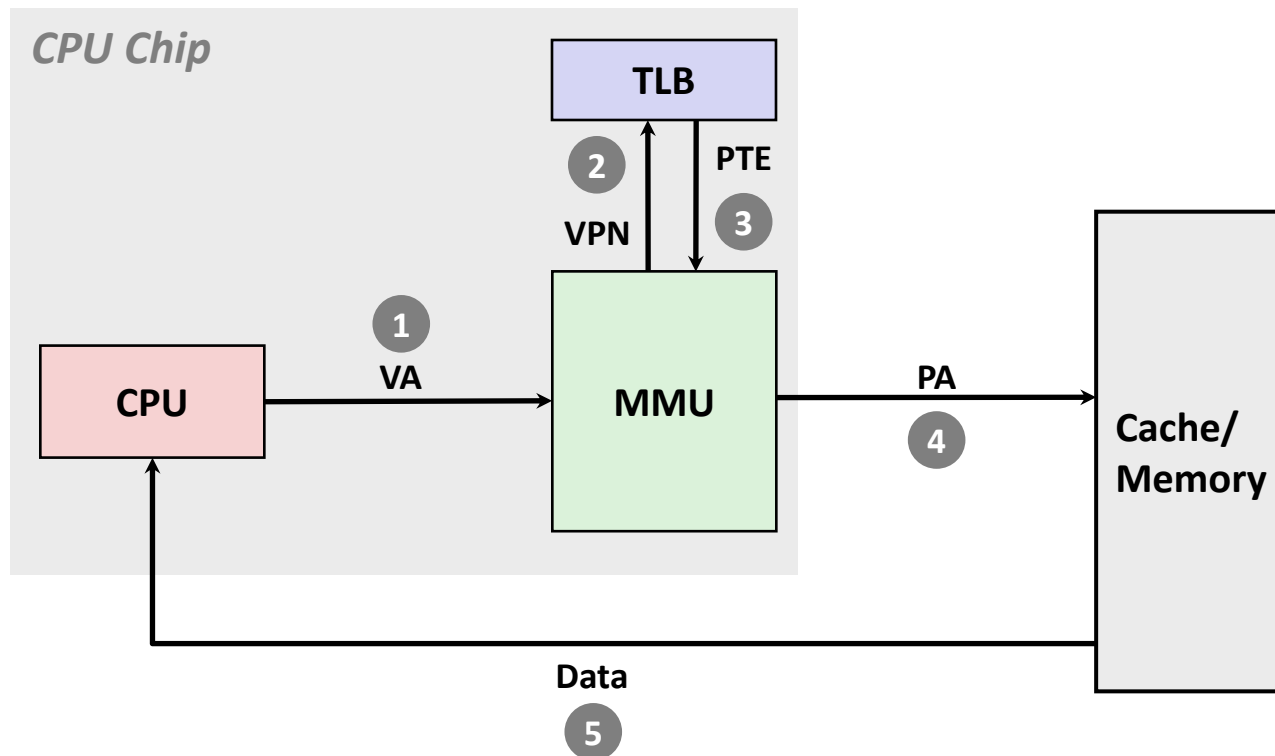


# Accessing the TLB

- MMU uses the VPN portion of the virtual address to access the TLB:

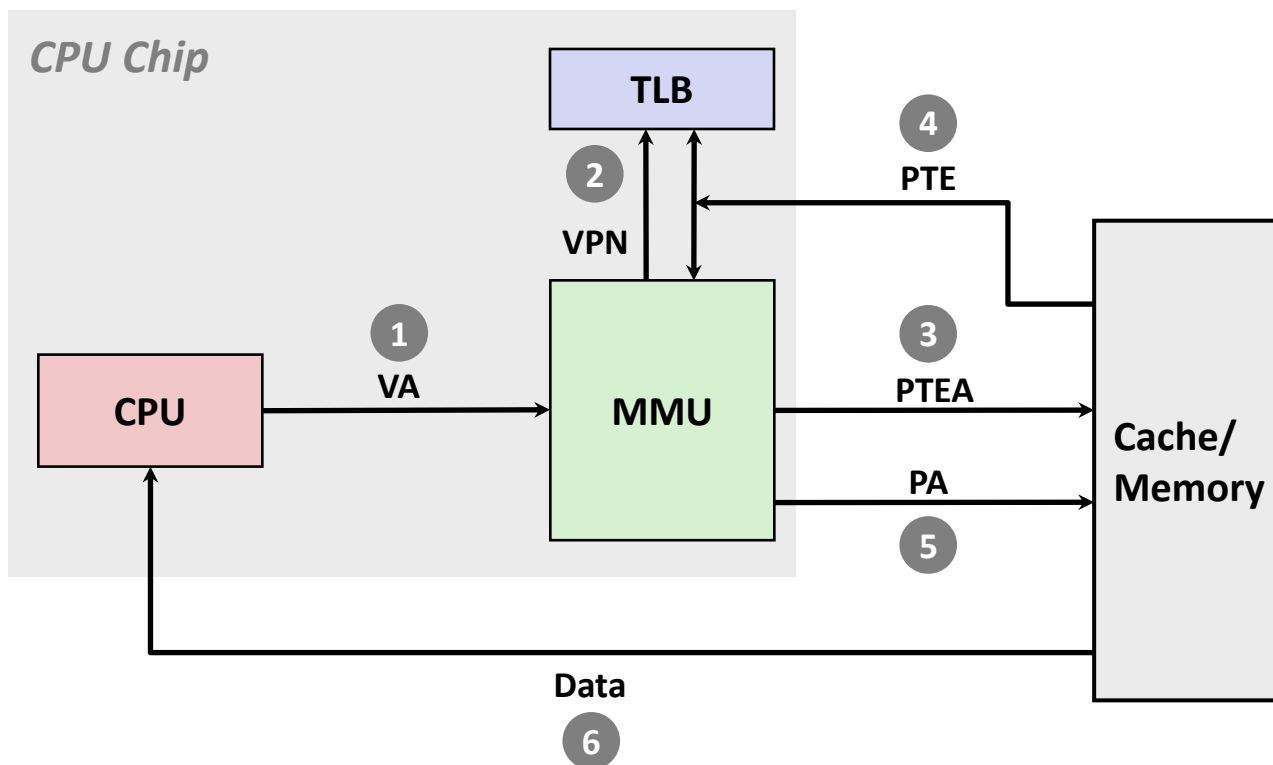


# TLB Hit



**A TLB hit eliminates memory accesses to the page table**

# TLB Miss



**A TLB miss incurs additional memory accesses (PTE lookup)**

Fortunately, TLB misses are rare. Why?

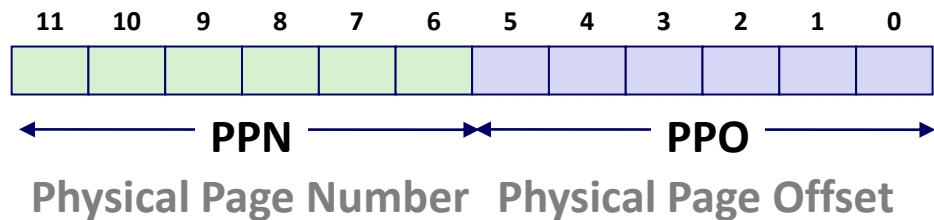
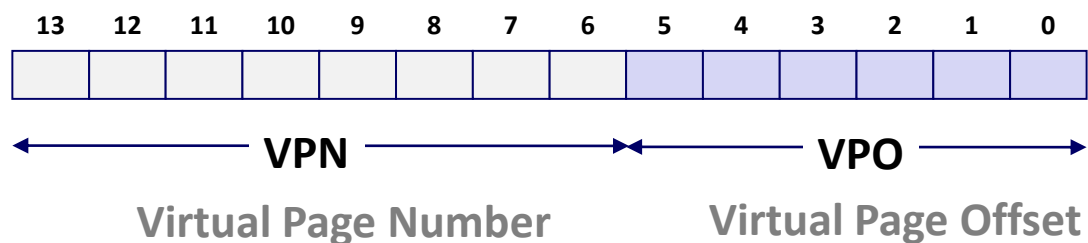
# Today

- Multi-level page tables
- Translation lookaside buffers
- **Activity 1**
- **Concrete examples of virtual memory systems**
  - “Simple memory system” from CSAPP 9.6.4
  - Intel Core i7
- Activity 2
- **Nifty things virtual memory makes possible**
  - Paging/swapping (disk as extra RAM)
  - Memory-mapped files (RAM as cache for disk)
  - Copy-on-write sharing
- **Activity 3**

# Simple Memory System Example

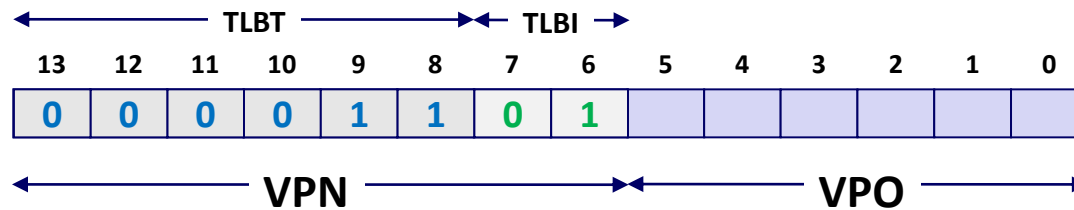
## ■ Addressing

- 14-bit virtual addresses
- 12-bit physical address
- Page size = 64 bytes



# Simple Memory System TLB

- 16 entries
- 4-way associative



$$\text{VPN} = 0b1101 = 0x0D$$

## Translation Lookaside Buffer (TLB)

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	-	0	09	0D	1	00	-	0	07	02	1
1	03	2D	1	02	-	0	04	-	0	0A	-	0
2	02	-	0	08	-	0	06	-	0	03	-	0
3	07	-	0	03	0D	1	0A	34	1	02	-	0

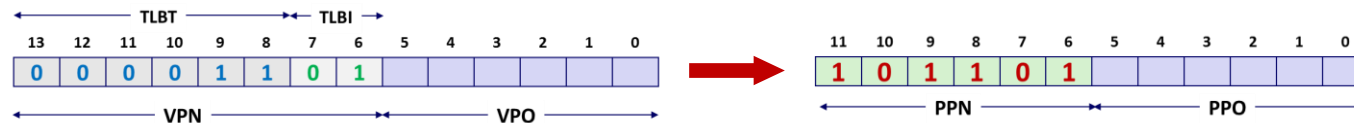
# Simple Memory System Page Table

- Only showing the first 16 entries (out of 256)

<i>VPN</i>	<i>PPN</i>	<i>Valid</i>
00	28	1
01	–	0
02	33	1
03	02	1
04	–	0
05	16	1
06	–	0
07	–	0

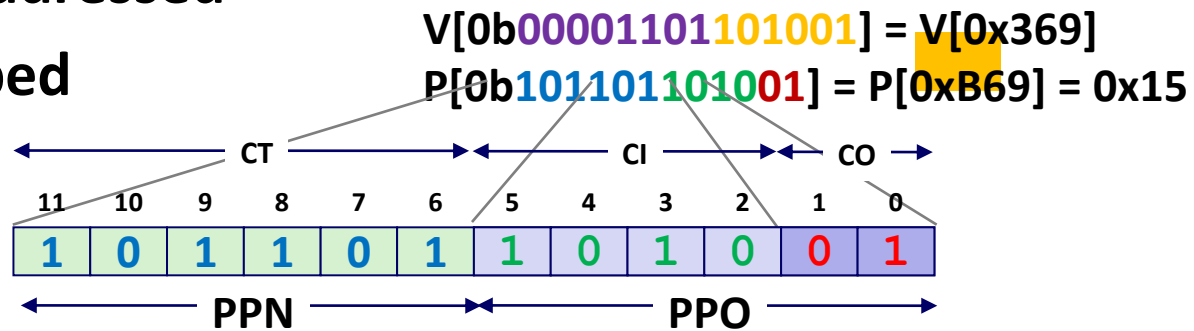
<i>VPN</i>	<i>PPN</i>	<i>Valid</i>
08	13	1
09	17	1
0A	09	1
0B	–	0
0C	–	0
0D	2D	1
0E	11	1
0F	0D	1

0x0D → 0x2D



# Simple Memory System Cache

- 16 lines, 4-byte cache line size
- Physically addressed
- Direct mapped



<i>Idx</i>	<i>Tag</i>	<i>Valid</i>	<i>B0</i>	<i>B1</i>	<i>B2</i>	<i>B3</i>
0	19	1	99	11	23	11
1	15	0	-	-	-	-
2	1B	1	00	02	04	08
3	36	0	-	-	-	-
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	-	-	-	-
7	16	1	11	C2	DF	03

<i>Idx</i>	<i>Tag</i>	<i>Valid</i>	<i>B0</i>	<i>B1</i>	<i>B2</i>	<i>B3</i>
8	24	1	3A	00	51	89
9	2D	0	-	-	-	-
A	2D	1	93	15	DA	3B
B	0B	0	-	-	-	-
C	12	0	-	-	-	-
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	-	-	-	-



# Address Translation Example

Virtual Address: 0x03D4

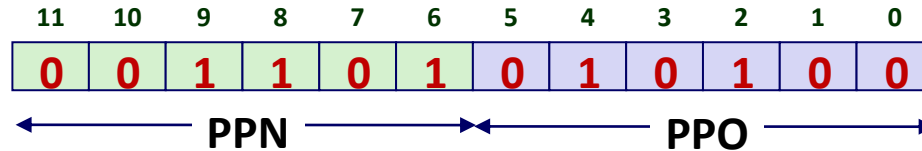


← **VPN** → ← **VPO** →  
 VPN 0x0F    TLBI 0x3    TLBT 0x03    TLB Hit? Y    Page Fault? N    PPN: 0x0D

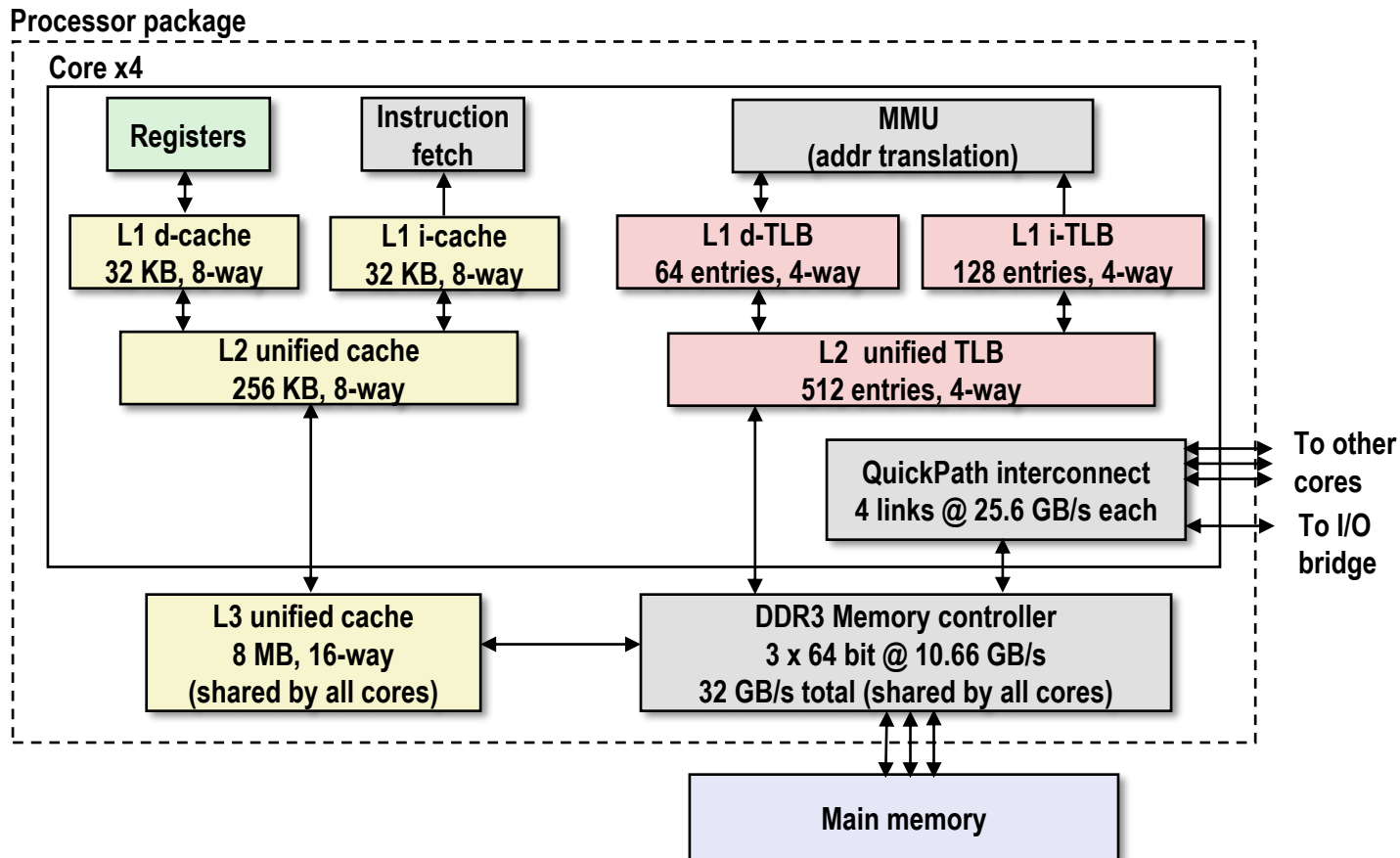
TLB

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	-	0	09	0D	1	00	-	0	07	02	1
1	03	2D	1	02	-	0	04	-	0	0A	-	0
2	02	-	0	08	-	0	06	-	0	03	-	0
3	07	-	0	03	0D	1	0A	34	1	02	-	0

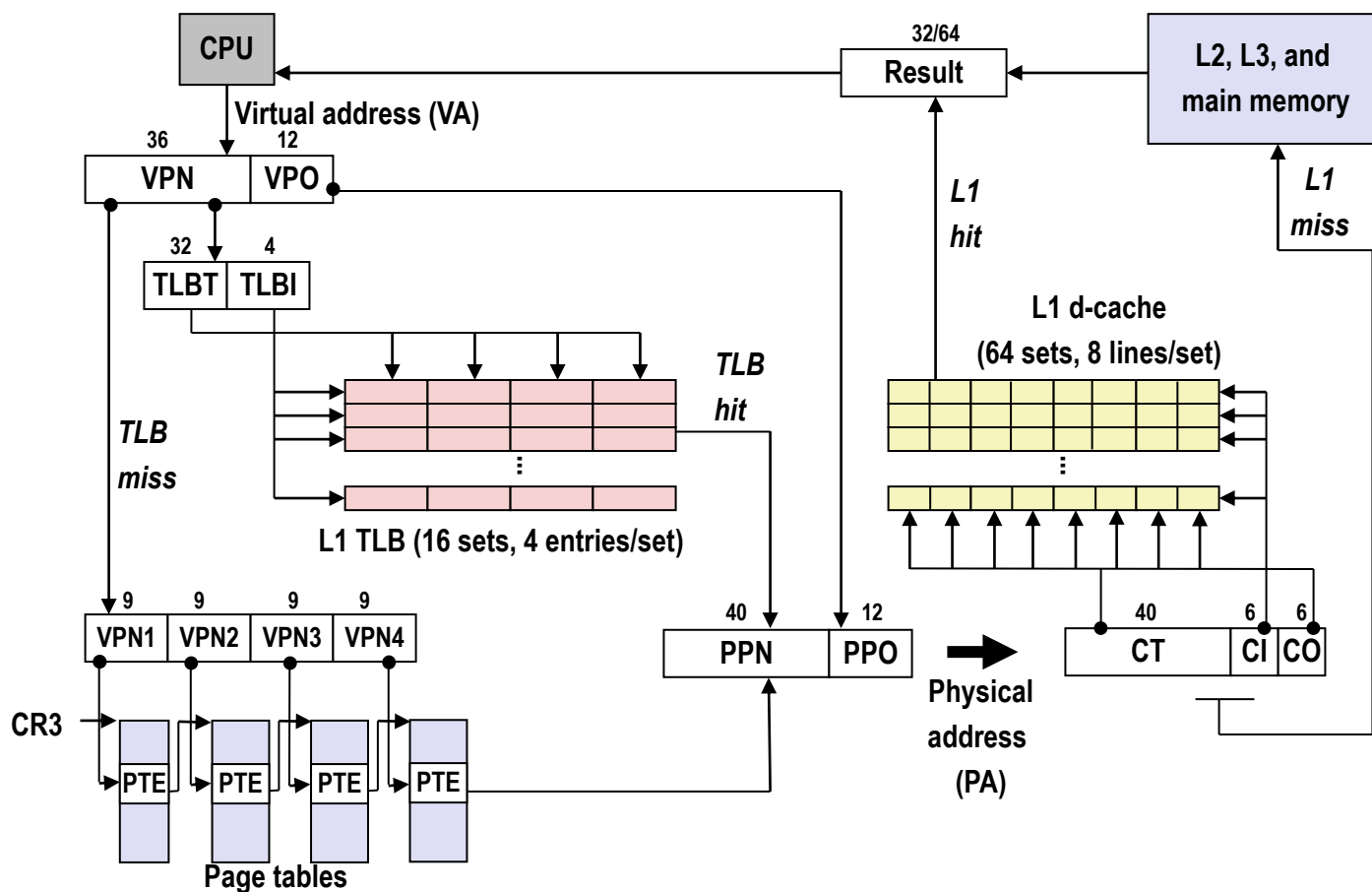
Physical Address



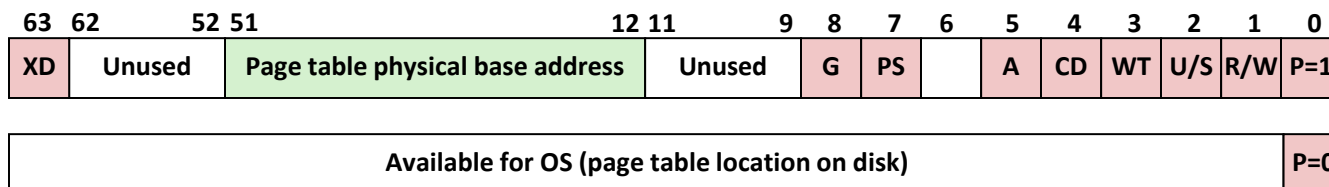
# Intel Core i7 Memory System



# End-to-end Core i7 Address Translation



# Core i7 Level 1-3 Page Table Entries



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

**P:** Child page table present in physical memory (1) or not (0).

**R/W:** Read-only or read-write access access permission for all reachable pages.

**U/S:** user or supervisor (kernel) mode access permission for all reachable pages.

**WT:** Write-through or write-back cache policy for the child page table.

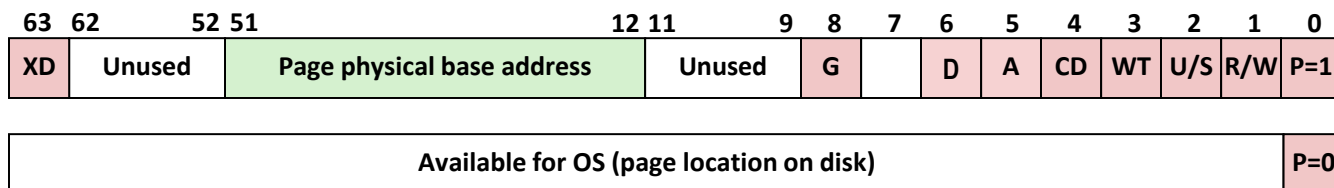
**A:** Reference bit (set by MMU on reads and writes, cleared by software).

**PS:** Page size either 4 KB or 4 MB (defined for Level 1 PTEs only).

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

**XD:** Disable or enable instruction fetches from all pages reachable from this PTE.

# Core i7 Level 4 Page Table Entries



**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)

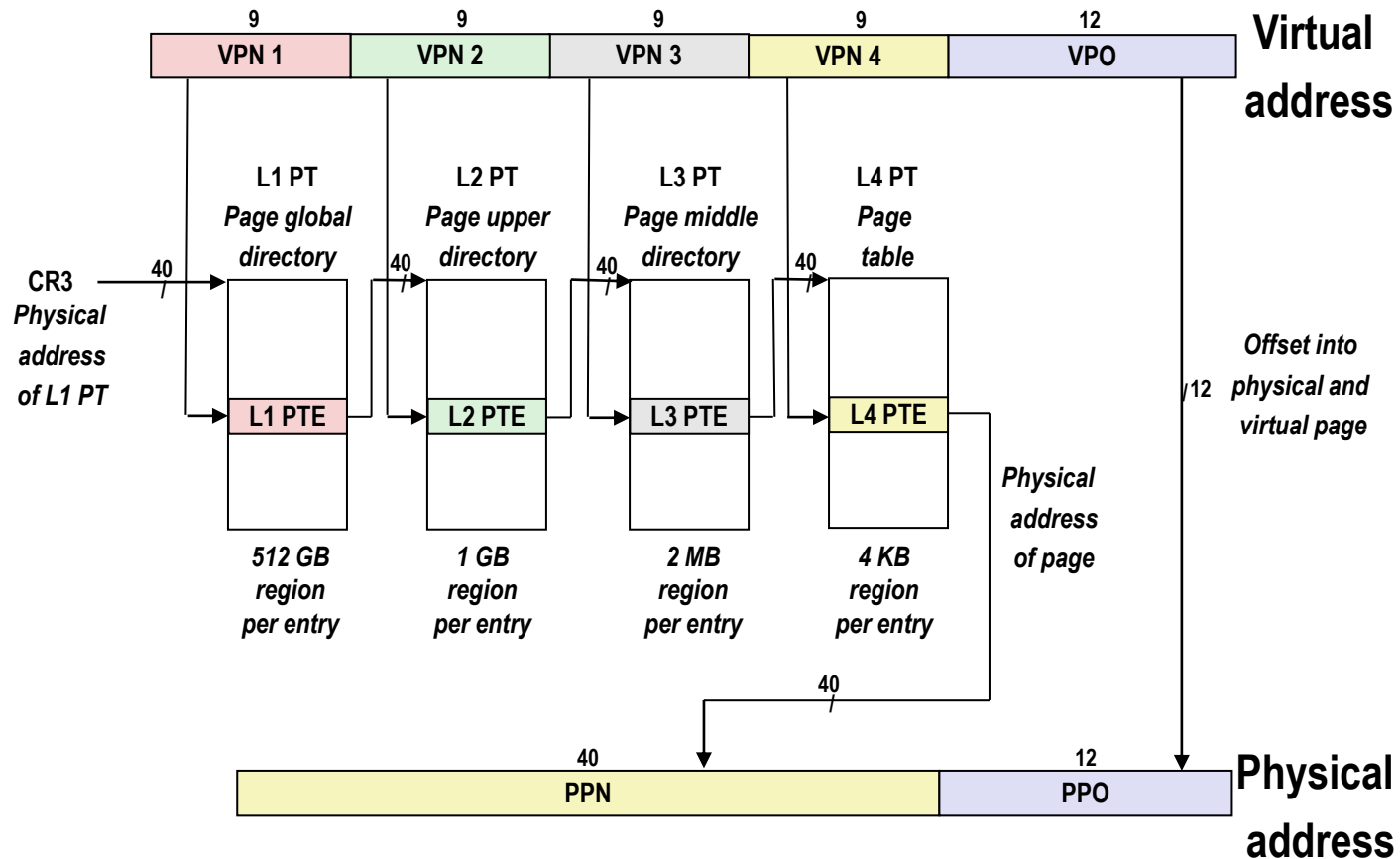
**D:** Dirty bit (set by MMU on writes, cleared by software)

**G:** Global page (don't evict from TLB on task switch)

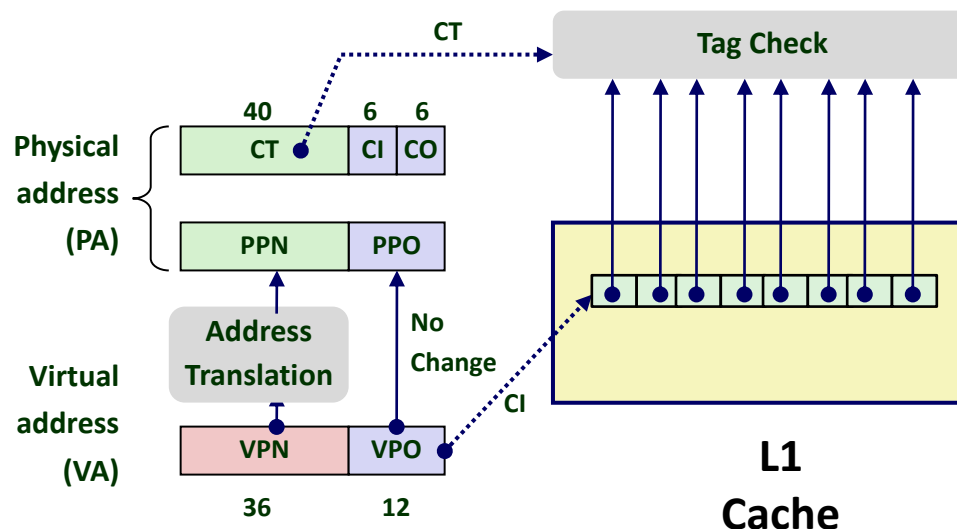
**Page physical base address:** 40 most significant bits of physical page address  
(forces pages to be 4KB aligned)

**XD:** Disable or enable instruction fetches from this page.

# Core i7 Page Table Translation



# Cute Trick for Speeding Up L1 Access



## ■ Observation

- Bits that determine CI identical in virtual and physical address
- Can index into cache while address translation taking place
- Generally we hit in TLB, so PPN bits (CT bits) available quickly
- “Virtually indexed, physically tagged”
- Cache carefully sized to make this possible

# Today

- Multi-level page tables
- Translation lookaside buffers
- Activity 1
- Concrete examples of virtual memory systems
  - “Simple memory system” from CSAPP 9.6.4
  - Intel Core i7
- **Activity 2**
- **Nifty things virtual memory makes possible**
  - Paging/swapping (disk as extra RAM)
  - Memory-mapped files (RAM as cache for disk)
  - Copy-on-write sharing
- Activity 3



# Paging (aka Swapping)

- **Use (part of) disk as additional working memory**
- **Adds another layer to the memory hierarchy, but...**
  - “Main memory” is 10–1000x slower than the caches
  - Disk is **10,000x** slower than main memory
  - Enormous miss penalty drives design
- **Consequences**
  - Large page (block) size: 4KB and bigger
  - Always write-back and fully associative
  - Managed entirely in software
    - Plenty of time to execute complex replacement algorithms

# Locality to the Rescue Again!

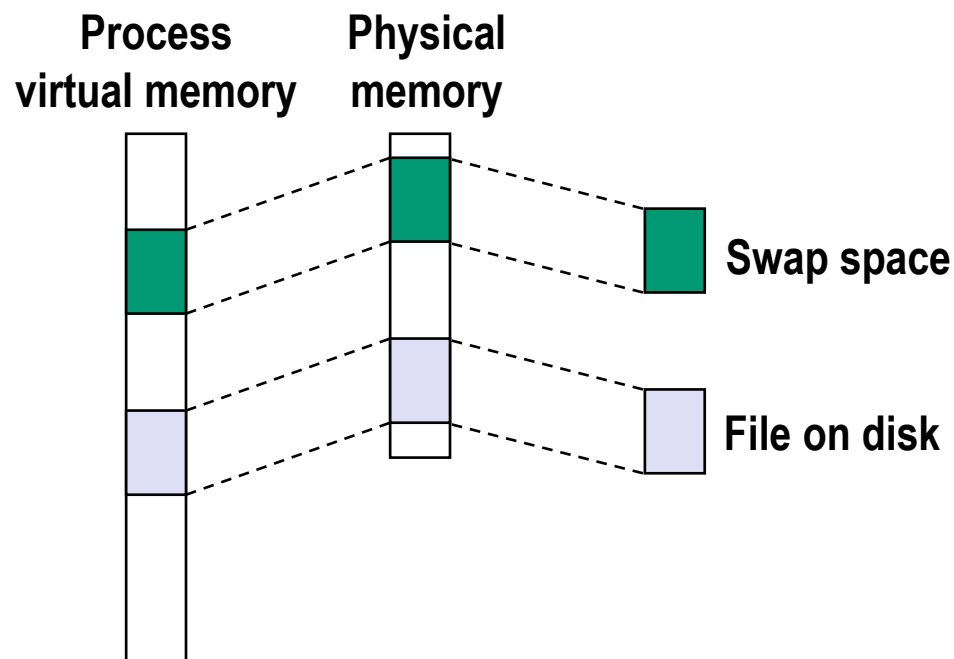
- Paging is terribly inefficient
- Only works because of locality
- At any point in time, programs tend to access a set of active virtual pages called the *working set*
  - Programs with good temporal locality will have small working sets
- If working set size  $<$  main memory size
  - Good performance after compulsory misses
- If working set size  $>$  main memory size
  - *Thrashing*: Performance meltdown, computer spends most of its time copying pages in and out of RAM
  - In the worst case, no forward progress at all (livelock)

# Memory-Mapped Files

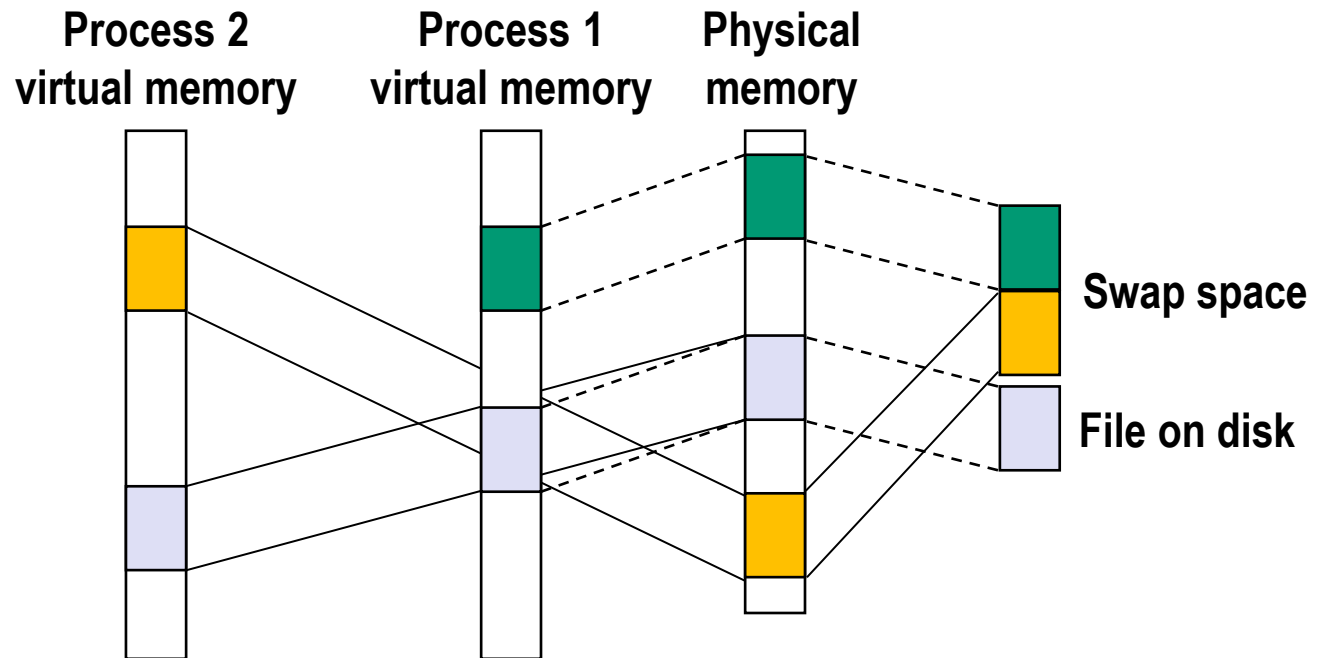
- **Paging = every page of a program's physical RAM is *backed* by some page of disk\***
- **Normally, those pages belong to *swap space***
- **But what if some pages were backed by ... files?**

\* This is how it used to work 20 years ago.  
Nowadays, not always true.

# Memory-Mapped Files



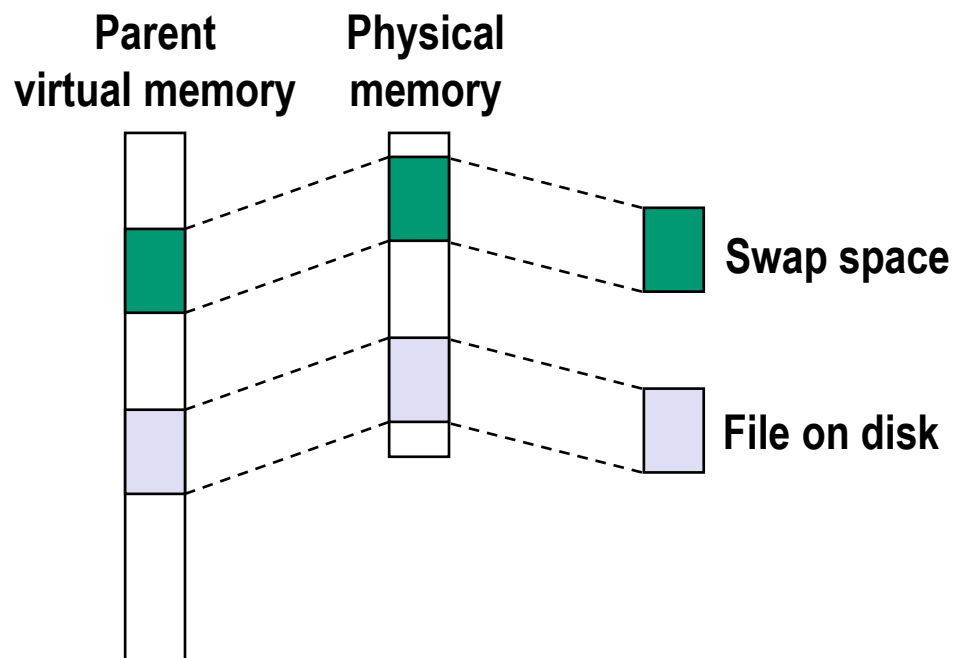
# Memory-Mapped Files



# Copy-on-write sharing

- `fork` creates a new process by copying the entire address space of the parent process

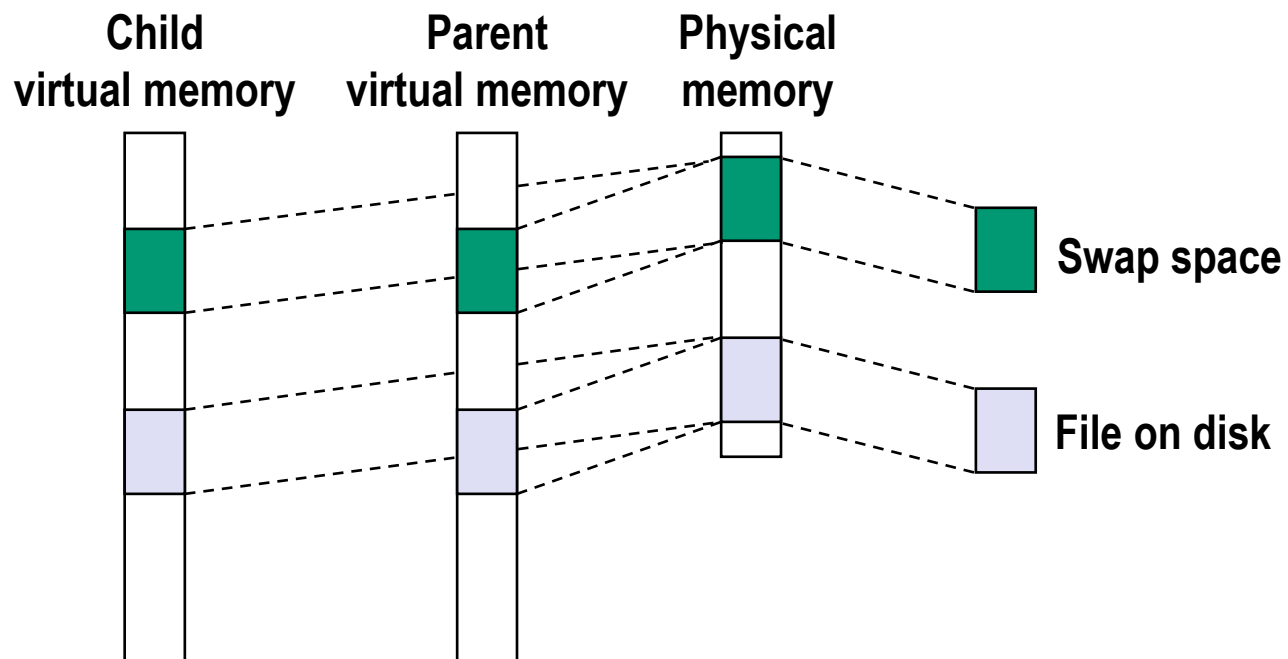
- That sounds slow
- It *is* slow



- **Clever trick:**

- Just duplicate the page tables
- Mark everything read only
- Copy only on write faults

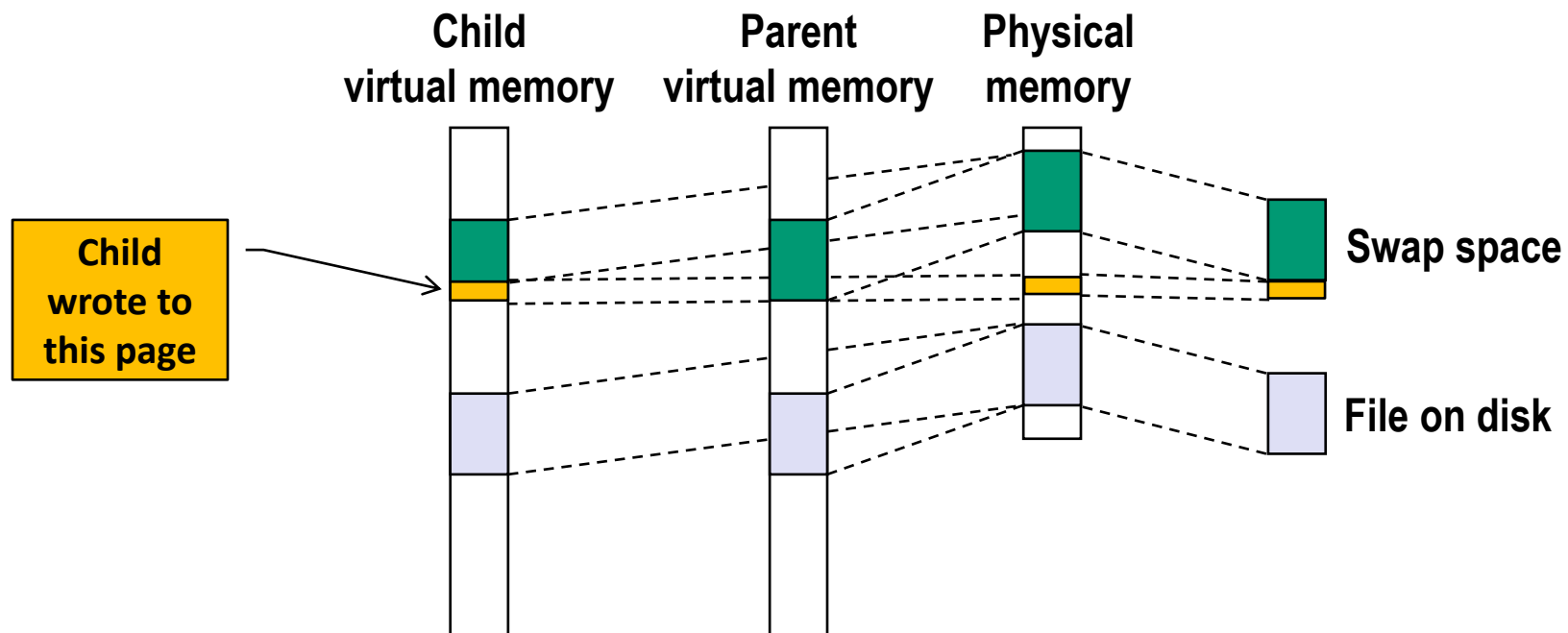
# Copy-on-write sharing



## ■ Clever trick:

- Just duplicate the page tables
- Mark everything read only
- Copy only on write faults

# Copy-on-write sharing



## ■ Clever trick:

- Just duplicate the page tables
- Mark everything read only
- Copy only on write faults



# Today

- Multi-level page tables
- Translation lookaside buffers
- Activity 1
- Concrete examples of virtual memory systems
  - “Simple memory system” from CSAPP 9.6.4
  - Intel Core i7
- Activity 2
- Nifty things virtual memory makes possible
  - Paging/swapping (disk as extra RAM)
  - Memory-mapped files (RAM as cache for disk)
  - Copy-on-write sharing
- **Activity 3**