Virtual Memory: Details

15-213/14-513/15-513: Introduction to Computer Systems
17th Lecture, November 1, 2022

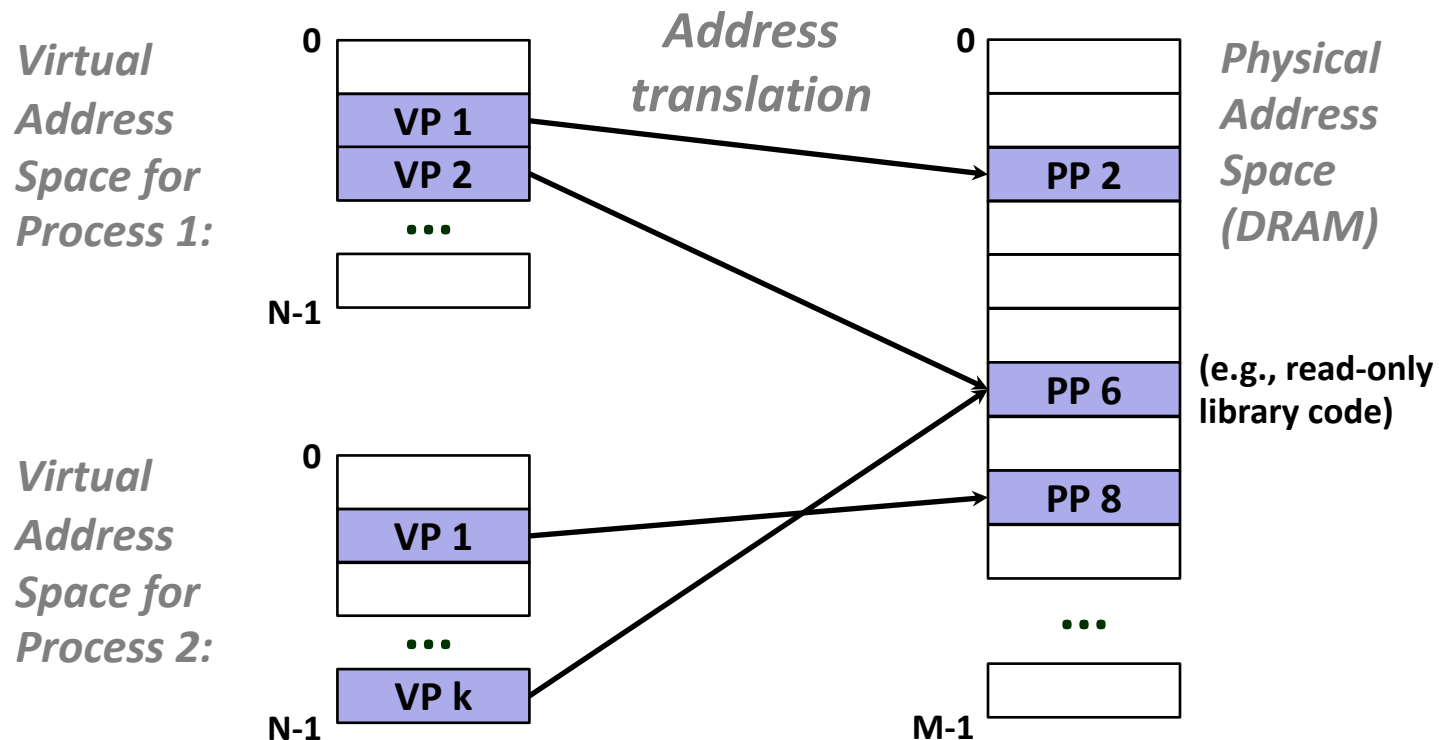**Instructors:**

Dave Andersen (15-213)

Zack Weinberg (15-213)

Brian Railing (15-513)
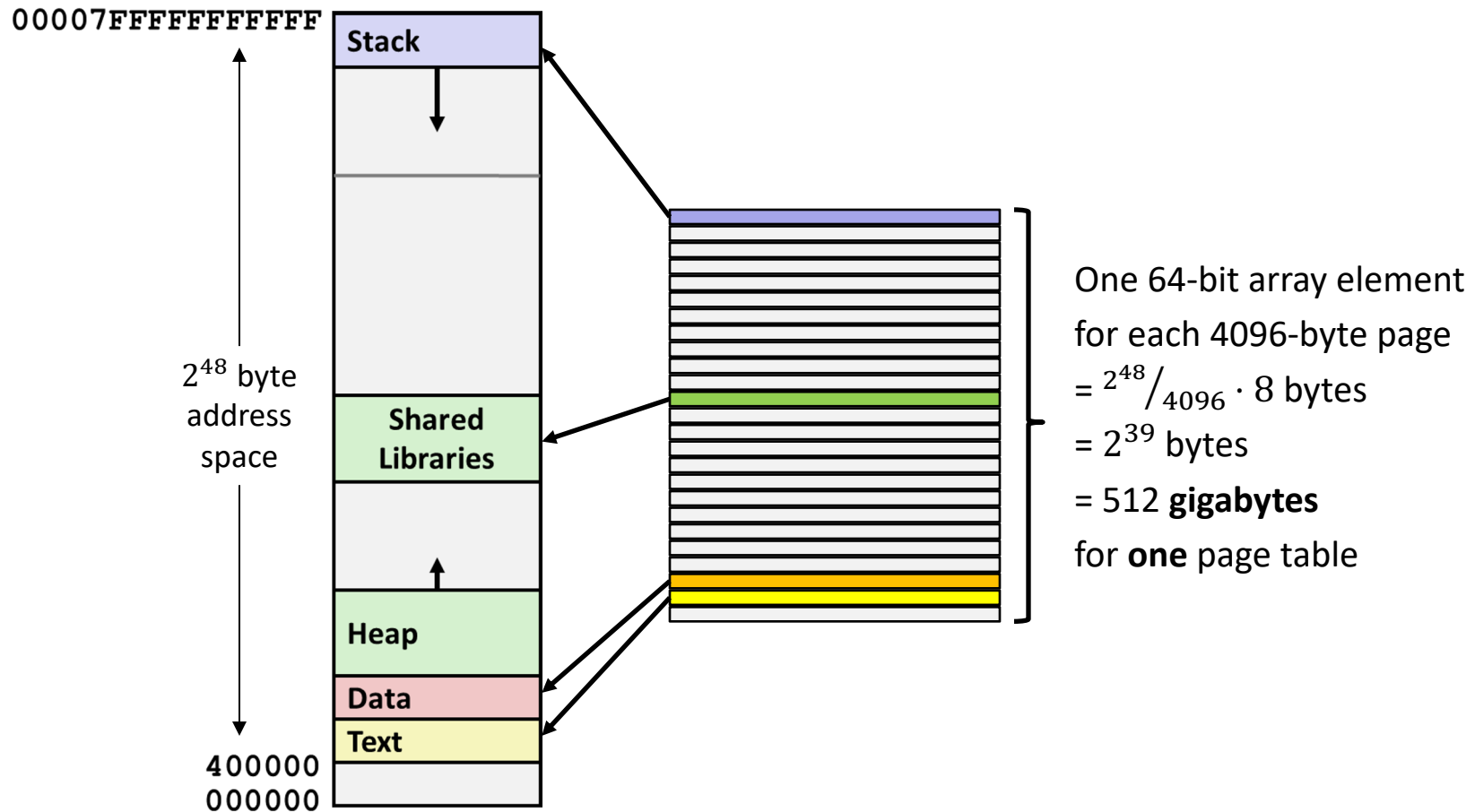
David Varodayan (14-513)

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**

Virtual Address Space for Process 1:

0

VP 1
VP 2
...

N-1

Address translation

0

PP 2

PP 6

PP 8

...

M-1

Physical Address Space (DRAM)

(e.g., read-only library code)

Virtual Address Space for Process 2:

0

VP 1

...

VP k

N-1

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition
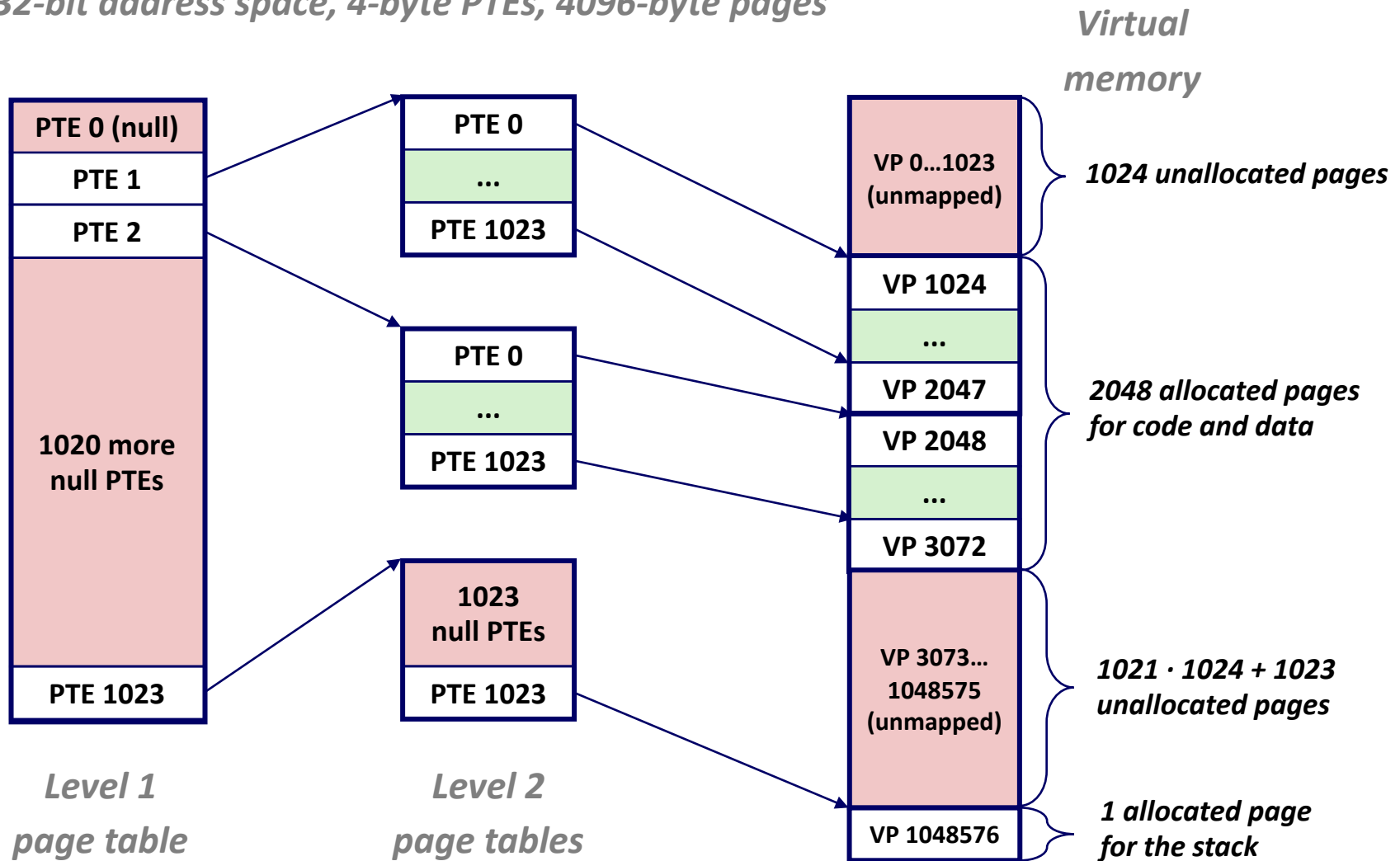
Today

- **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)

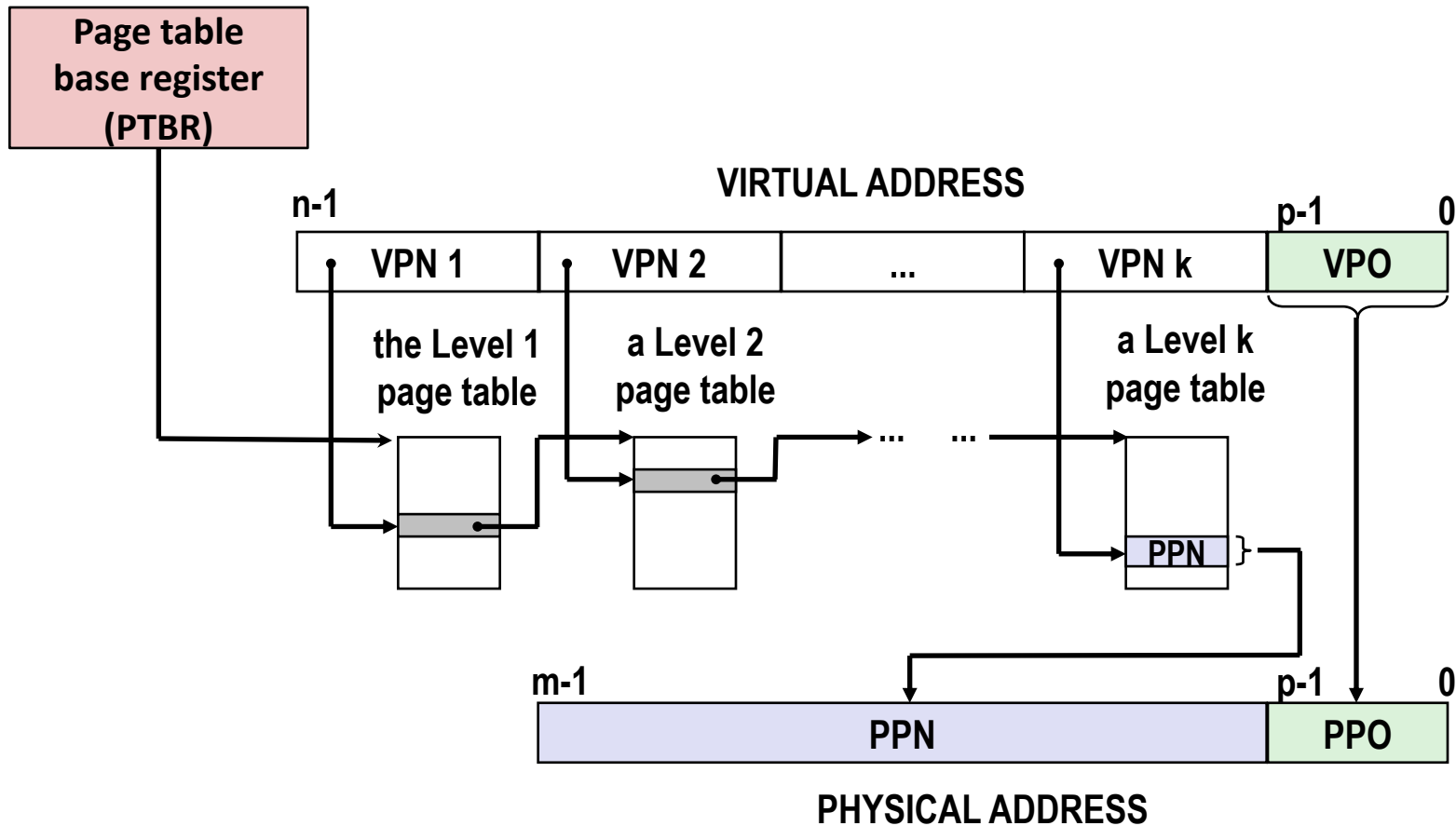00007FFFFFFFFFFF

Stack

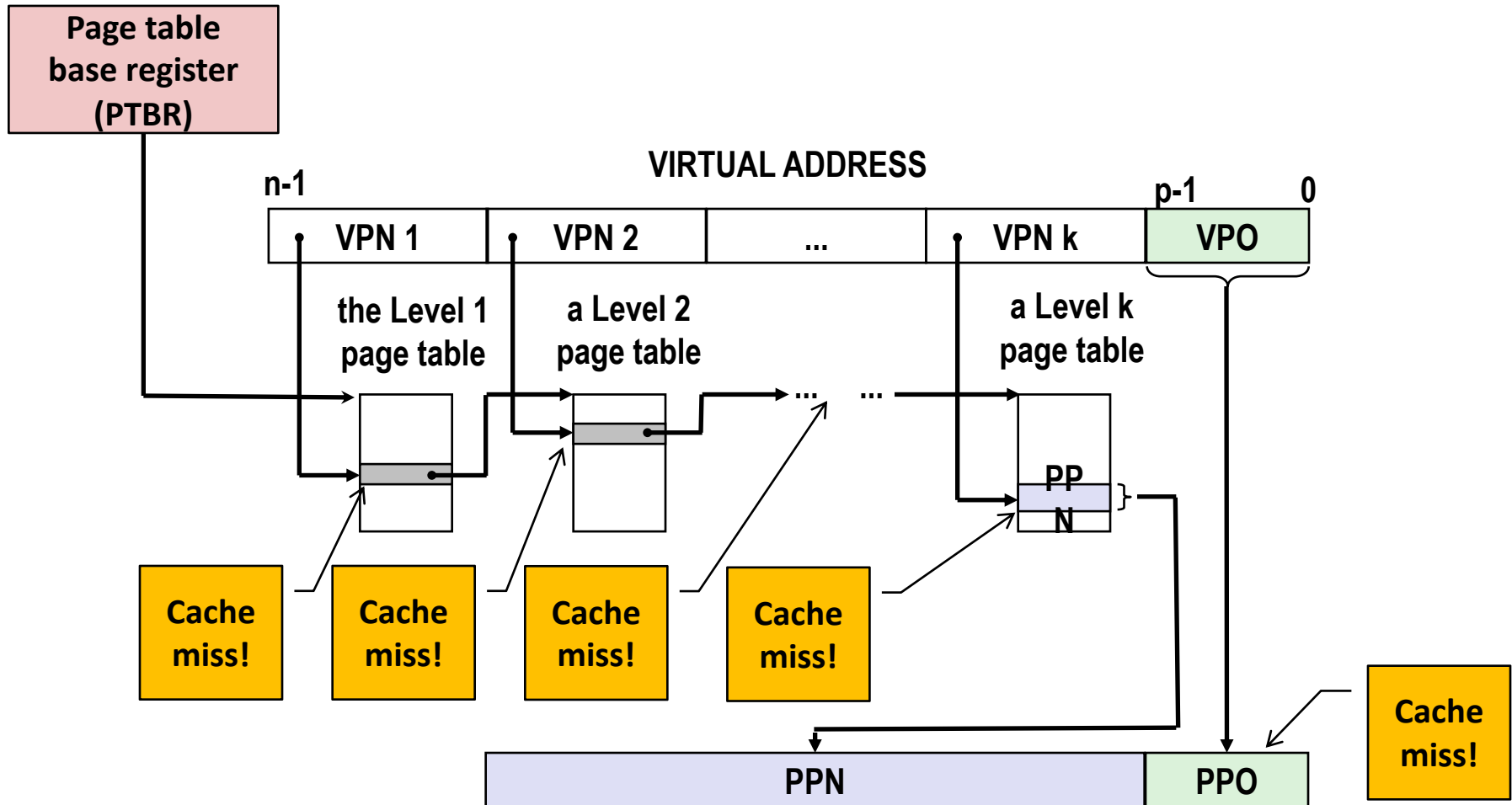2^{48} byte
address
space

Shared
Libraries

Heap

Data

Text

400000
000000

One 64-bit array element
for each 4096-byte page
= $^{2^{48}}/_{4096} \cdot 8$ bytes
= 2^{39} bytes
= 512 **gigabytes**
for **one** page table

A Two-Level Page Table Hierarchy

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

Virtual memory

Level 1 page table
PTE 0 (null)
PTE 1
PTE 2
1020 more null PTEs
PTE 1023

Level 2 page tables
PTE 0
...
PTE 1023

PTE 0
...
PTE 1023

1023 null PTEs
PTE 1023

VP 0...1023 (unmapped)   *1024 unallocated pages*

VP 1024
...
VP 2047
VP 2048
...
VP 3072   *2048 allocated pages for code and data*

VP 3073... 1048575 (unmapped)   *1021 · 1024 + 1023 unallocated pages*

VP 1048576   *1 allocated page for the stack*

Level 1 page table

Level 2 page tables

Translating with a k-level Page Table

Page table
base register
(PTBR)

VIRTUAL ADDRESS

| VPN 1 | VPN 2 | ... | VPN k | VPO |

n-1 ... p-1 ... 0

the Level 1
page table

a Level 2
page table

a Level k
page table

PPN

PHYSICAL ADDRESS

| PPN | PPO |

m-1 ... p-1 ... 0

The problem (with k-level page tables)

Page table base register (PTBR)

VIRTUAL ADDRESS

n-1				p-1	0
VPN 1	VPN 2	...	VPN k	VPO	

the Level 1 page table

a Level 2 page table

a Level k page table

PP N

Cache miss!

Cache miss!

Cache miss!

Cache miss!

Cache miss!

PPN	PPO

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
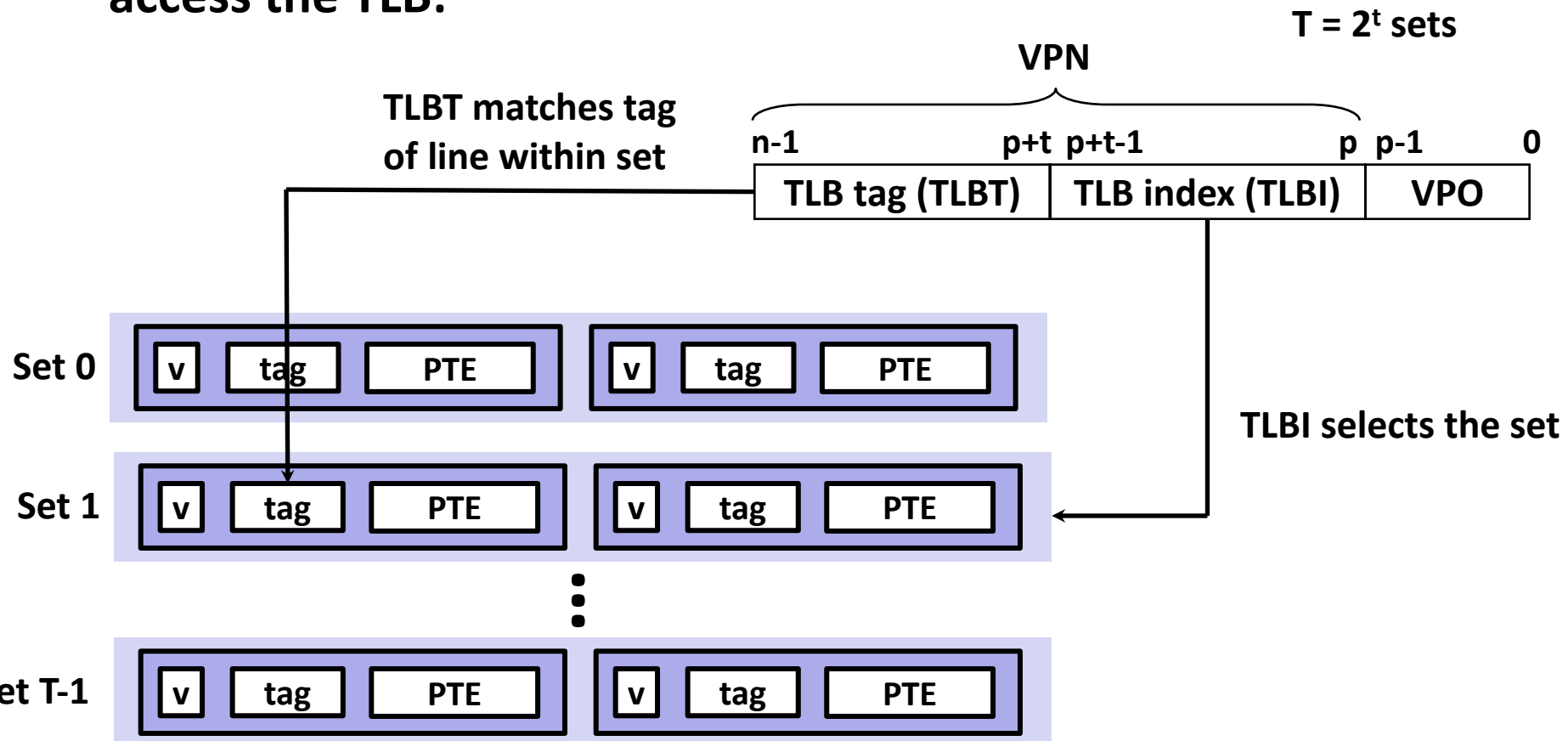- PTE hit still costs cache delay

Solution: *Translation Lookaside Buffer* (TLB)

- Dedicated cache for page table entries
- TLB hit = page table not consulted
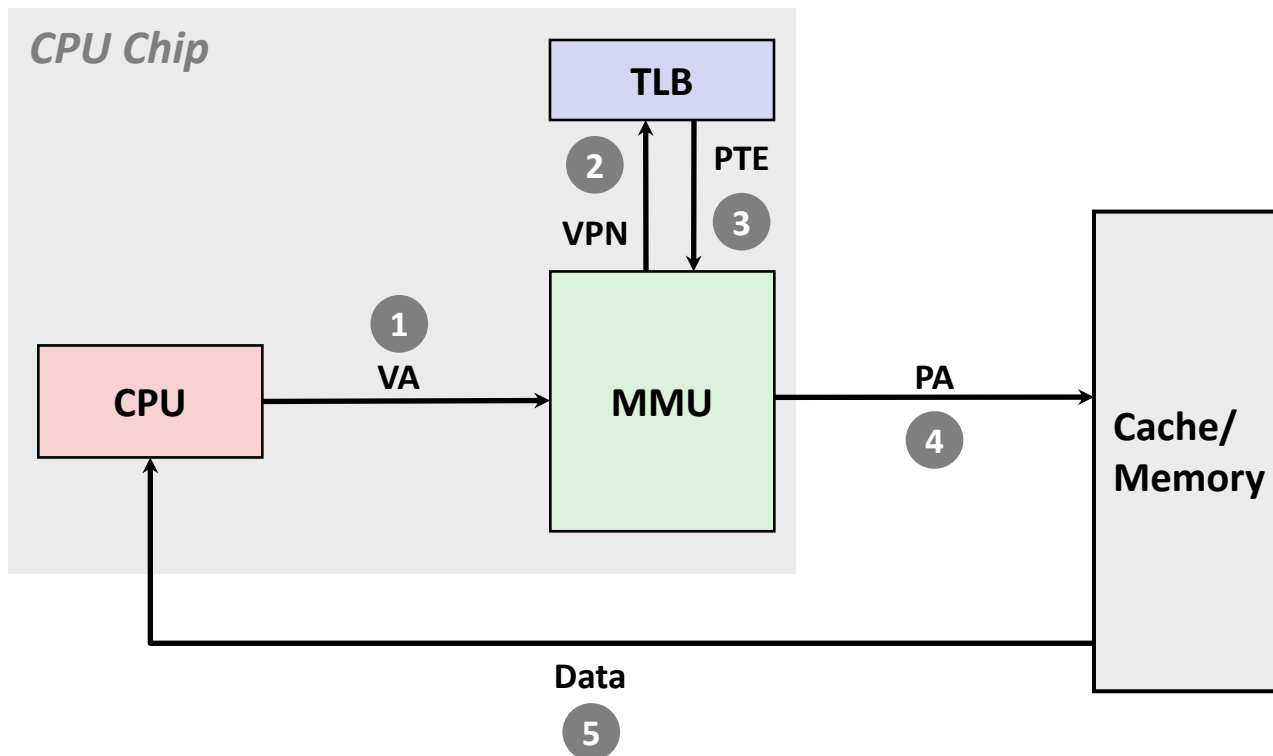- Can be fairly small: one TLB entry covers 4k or more

Accessing the TLB

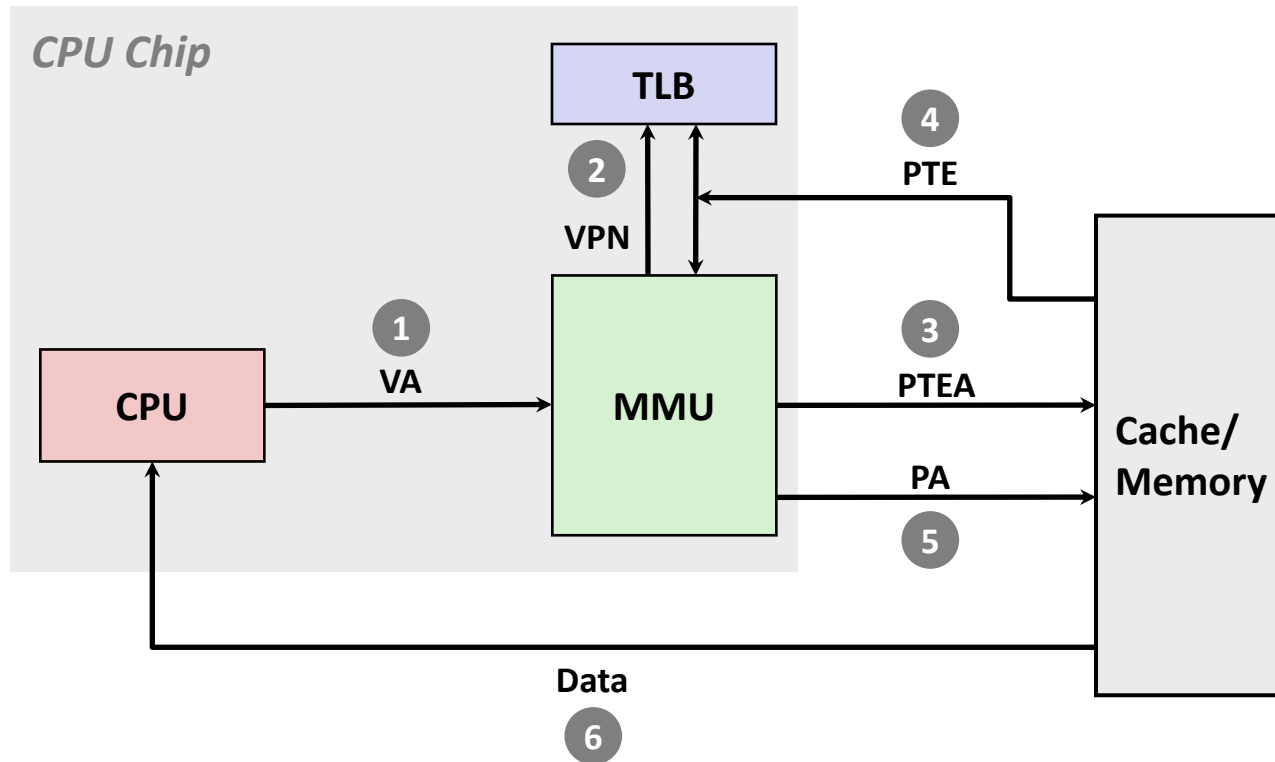**MMU uses the VPN portion of the virtual address to access the TLB:**

$T = 2^t$ sets

VPN

TLBT matches tag of line within set

	TLB tag (TLBT)	TLB index (TLBI)	VPO
n-1	p+t p+t-1	p p-1	0

Set 0 | v | tag | PTE | | v | tag | PTE |

TLBI selects the set

Set 1 | v | tag | PTE | | v | tag | PTE |

⋮

Set T-1 | v | tag | PTE | | v | tag | PTE |

TLB Hit

CPU Chip

TLB

2   PTE

VPN   3

1

VA   CPU   MMU   PA   Cache/Memory

4

Data

5

A TLB hit eliminates memory accesses to the page table

TLB Miss

A TLB miss incurs additional memory accesses (PTE lookup)
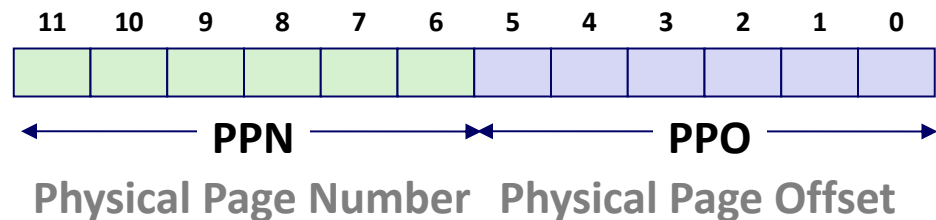Fortunately, TLB misses are rare. Why?
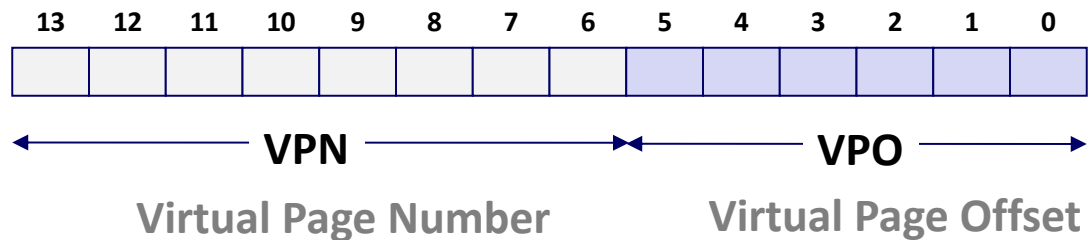
Today

- **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**
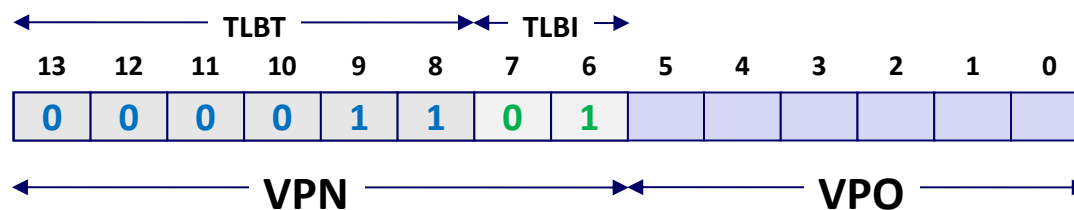
Simple Memory System Example

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

| 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

VPN ← → **VPO** →

Virtual Page Number　　**Virtual Page Offset**

| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

PPN ← → **PPO** →

Physical Page Number　Physical Page Offset

Simple Memory System TLB

- ## 16 entries
- ## 4-way associative

TLBT **TLBI**

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1						

VPN **VPO**

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)**

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

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

0x0D → 0x2D

	TLBT				TLBI									
13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	1						
		VPN							VPO				

→

11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	1						
	PPN					PPO					
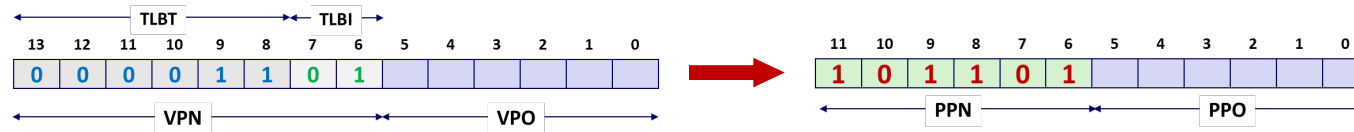
Simple Memory System Cache

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

V[0b00001101101001] = V[0x369]
P[0b101101101001] = P[0xB69] = 0x15

	CT						CI			CO	
11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	1	1	0	1	0	0	1

PPN ← → PPO
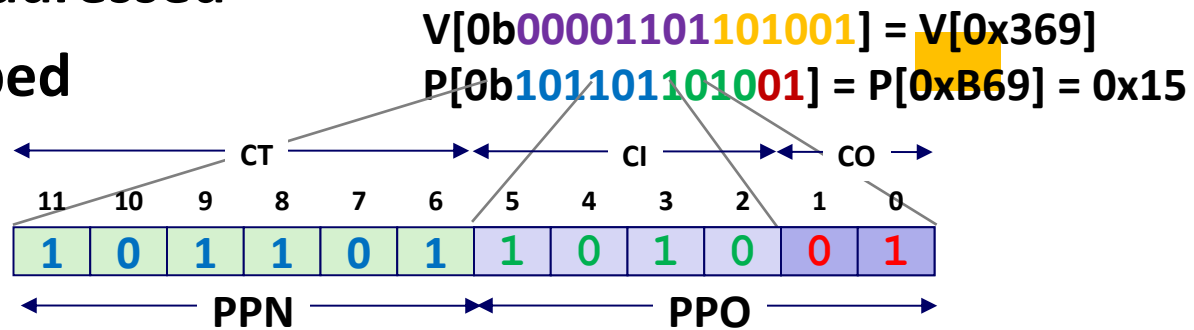
Idx	Tag	Valid	B0	B1	B2	B3
0	19	1	99	11	23	11
1	15	0	–	–	–	–
2	1B	1	00	02	04	08
3	36	0	–	–	–	–
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	–	–	–	–
7	16	1	11	C2	DF	03

Idx	Tag	Valid	B0	B1	B2	B3
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

TLBT ⟶ TLBI ⟶

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	0	1	0	1	0	0

⟵ 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

11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	0	1	0	0

⟵ PPN ⟶ ⟵ PPO ⟶
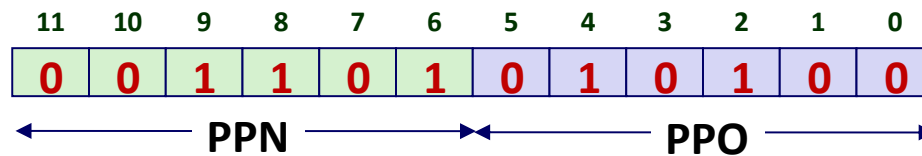
Intel Core i7 Memory System

Processor package

Core x4

Registers	**Instruction fetch**		**MMU (addr translation)**
L1 d-cache 32 KB, 8-way	**L1 i-cache 32 KB, 8-way**	**L1 d-TLB 64 entries, 4-way**	**L1 i-TLB 128 entries, 4-way**
L2 unified cache 256 KB, 8-way		**L2 unified TLB 512 entries, 4-way**	

QuickPath interconnect 4 links @ 25.6 GB/s each

To other cores

To I/O bridge

L3 unified cache 8 MB, 16-way (shared by all cores)

DDR3 Memory controller 3 x 64 bit @ 10.66 GB/s 32 GB/s total (shared by all cores)

Main memory

End-to-end Core i7 Address Translation
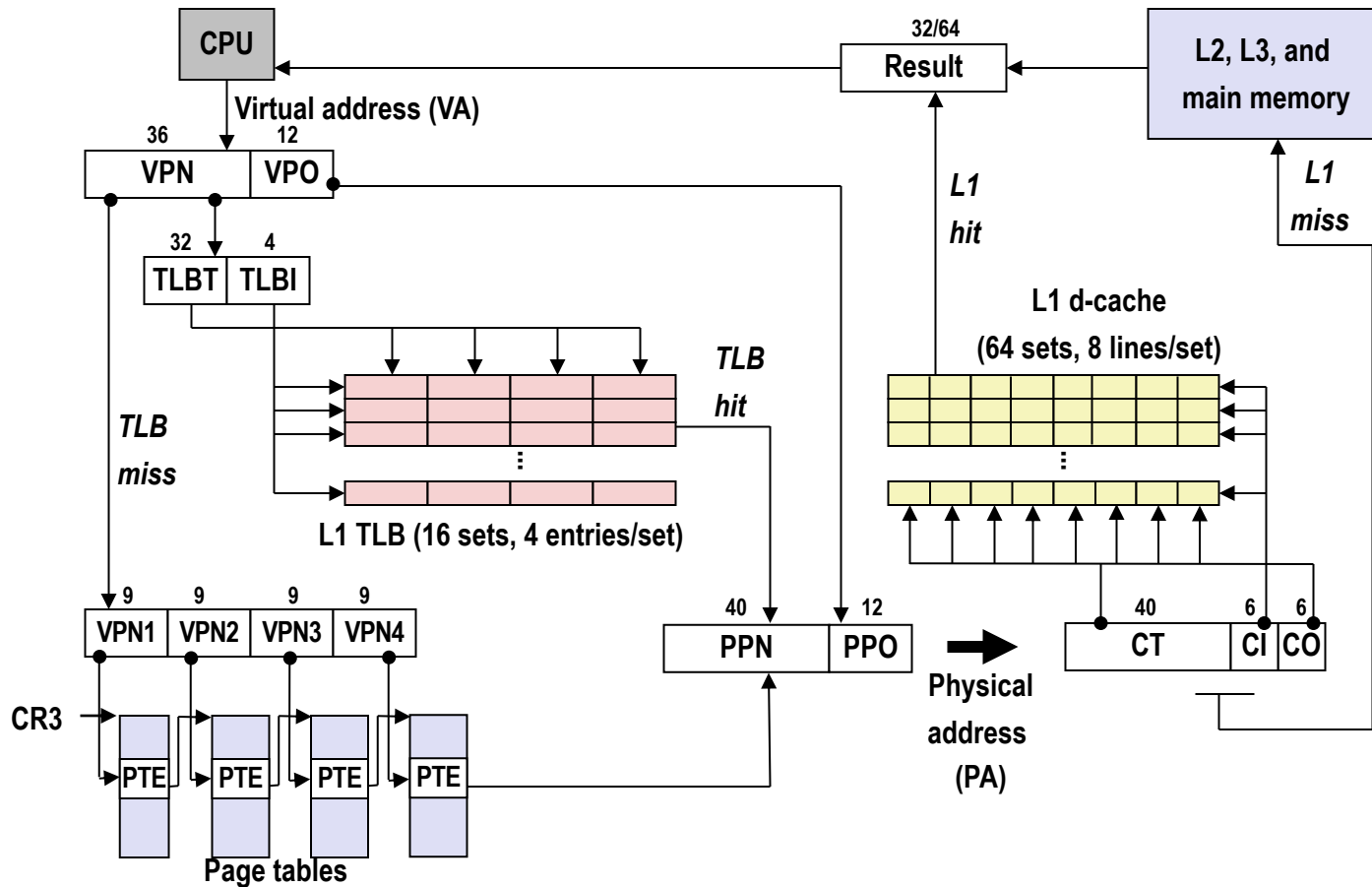
Core i7 Level 1-3 Page Table Entries

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

Available for OS (page table location on disk)	P=0
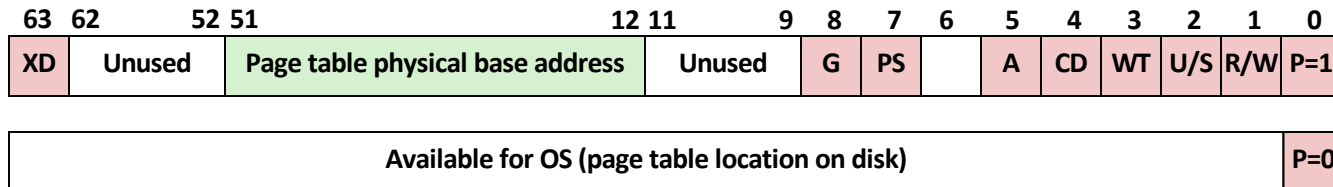
Each entry references a 4K child page table. Significant fields:

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

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

	P=0
Available for OS (page location on disk)	P=0
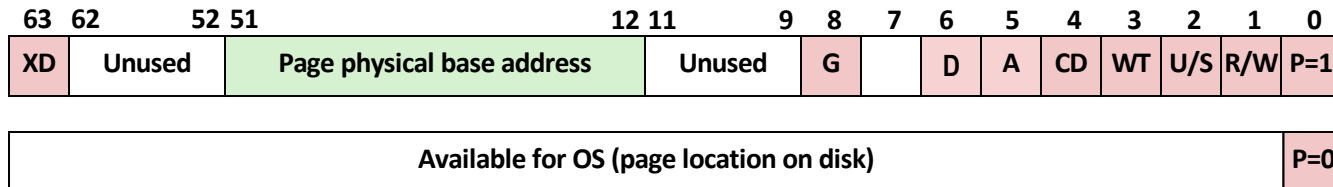
Each entry references a 4K child page. Significant fields:

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)

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
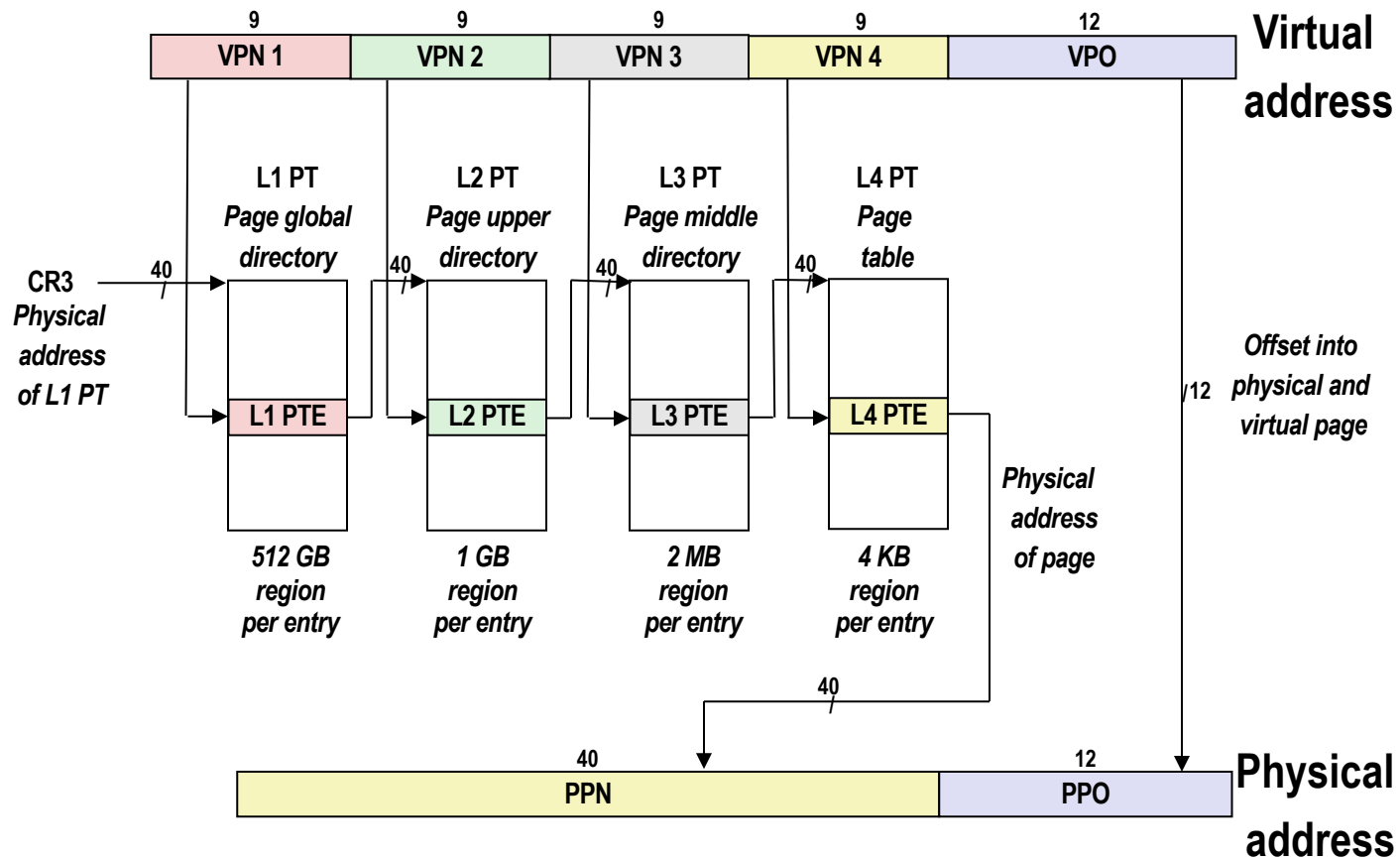
Cute Trick for Speeding Up L1 Access
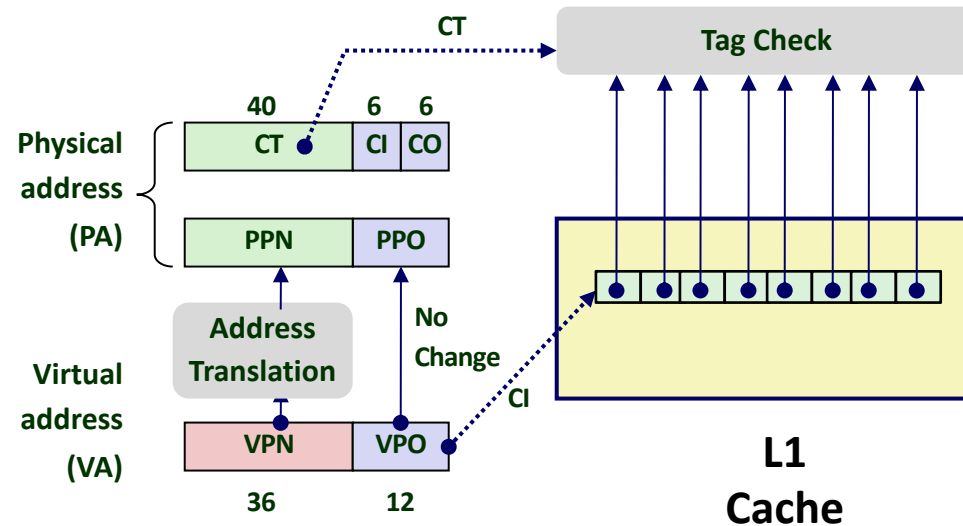
- **Observation**
  - Bits that determine CI identical in virtual and physical address
  - Can index into cache while address translation taking place
  - Generally we hit in TLB, so PPN bits (CT bits) available quickly
  - "Virtually indexed, physically tagged"
  - Cache carefully sized to make this 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**

Activity 2

Hint: Write down the address parts (starting w/tag) in binary. You can build up the address and then convert it back to hex at the end.

Python makes a good hex converter if you want to double check your brain version:

hex(0b01110001) → '0x71'

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.
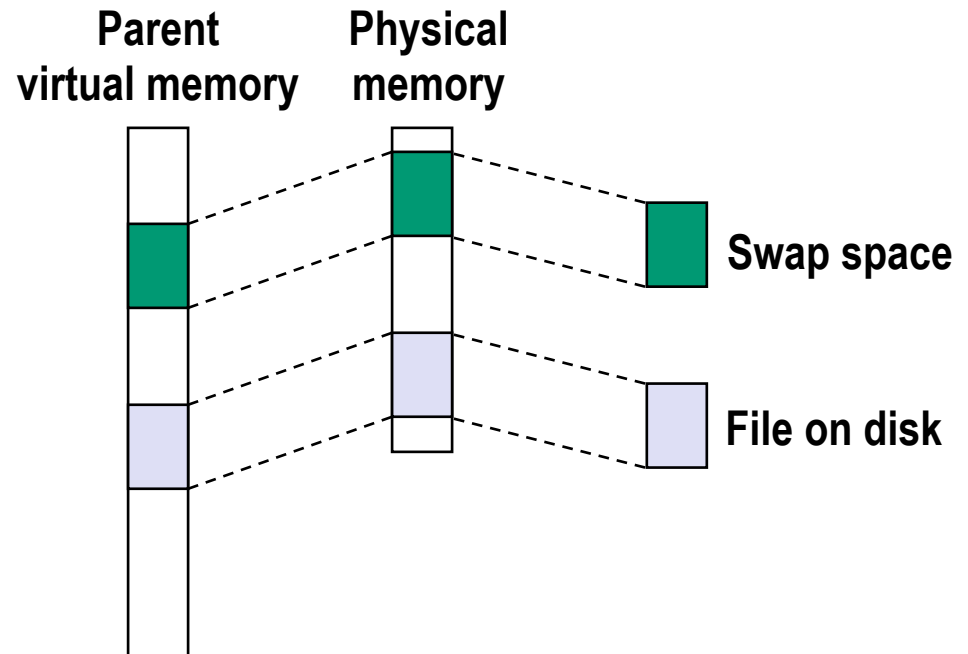Nowadays, not always true.

Memory-Mapped Files

Process virtual memory   **Physical memory**

Swap space

File on disk

Memory-Mapped Files

**Process 2
virtual memory**　　　**Process 1
virtual memory**　　　**Physical
memory**

Swap space

File on disk
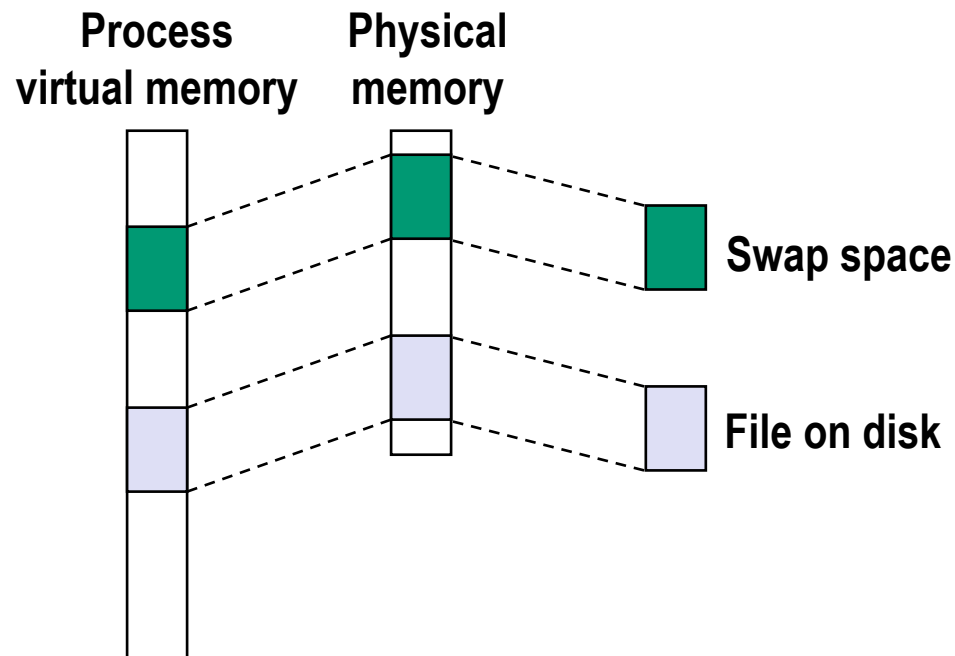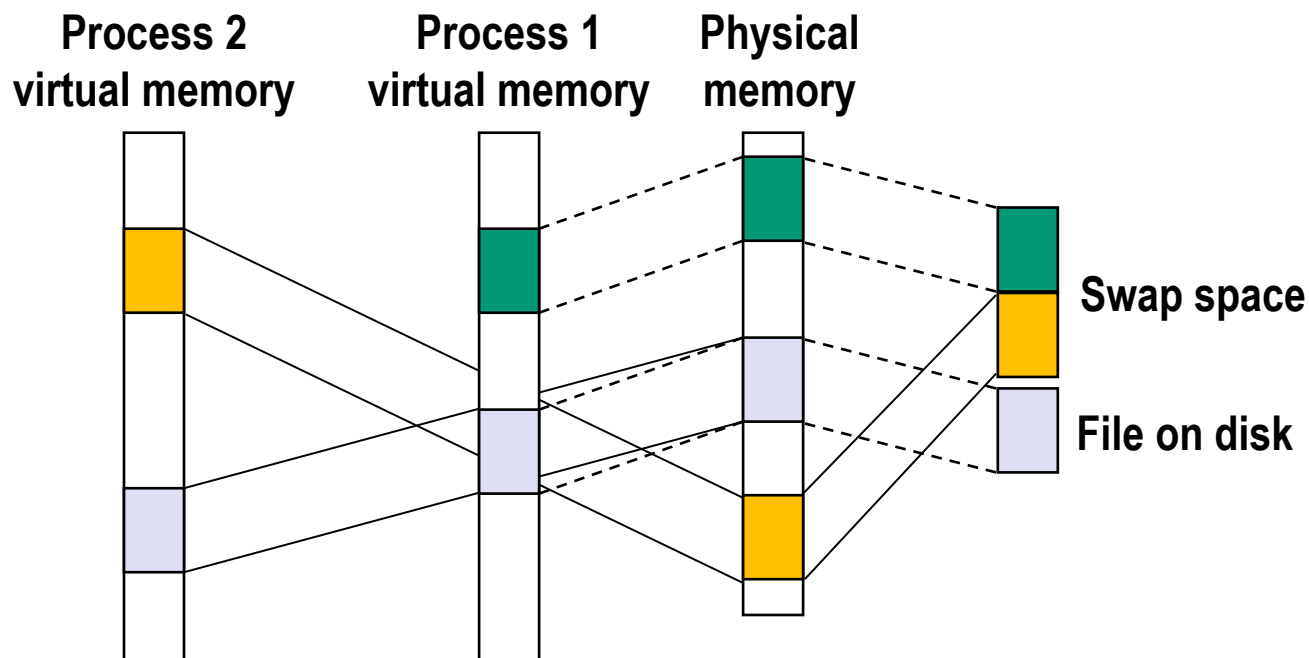
Copy-on-write sharing

- **`fork` creates a new process by copying the entire address space of the parent process**
  - That sounds slow
  - It *is* slow

Parent virtual memory   Physical memory

Swap space

File on disk

- **Clever trick:**
  - Just duplicate the page tables
  - Mark everything read only
  - Copy only on write faults

Copy-on-write sharing

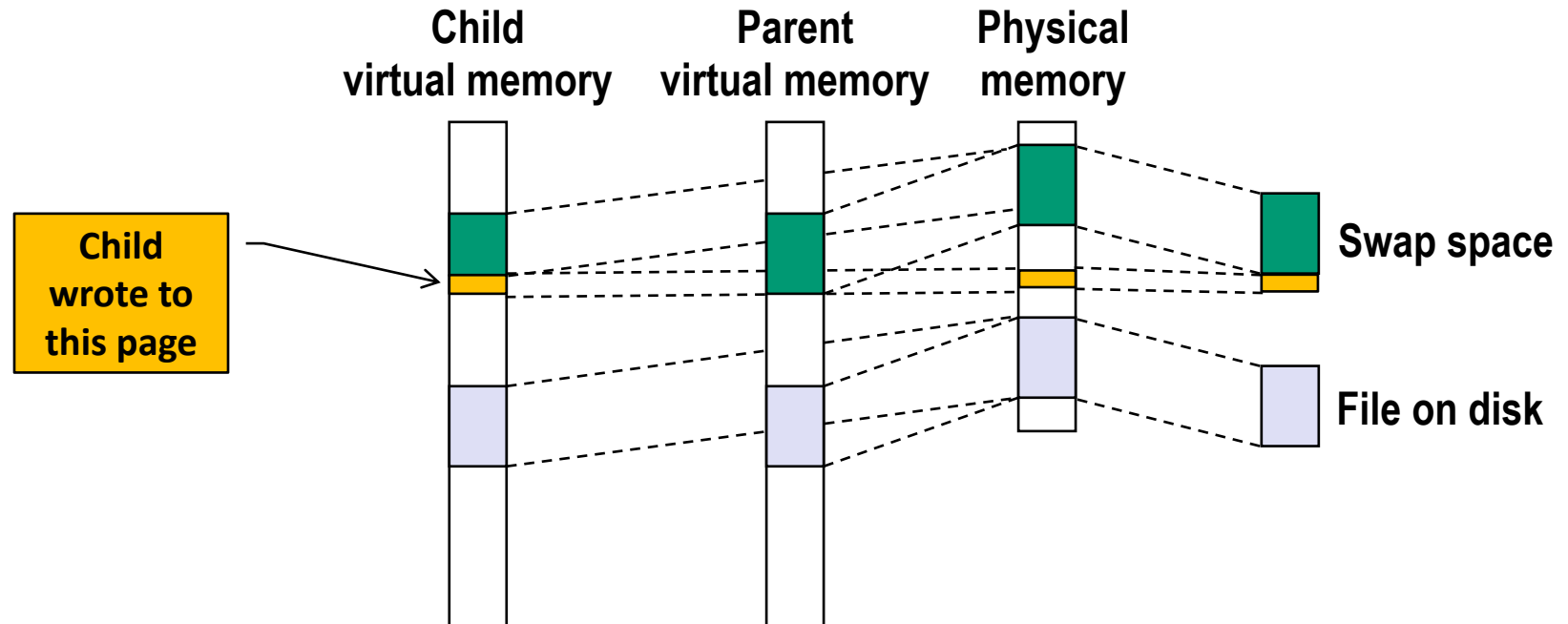Child virtual memory / Parent virtual memory / Physical memory / Swap space / File on disk
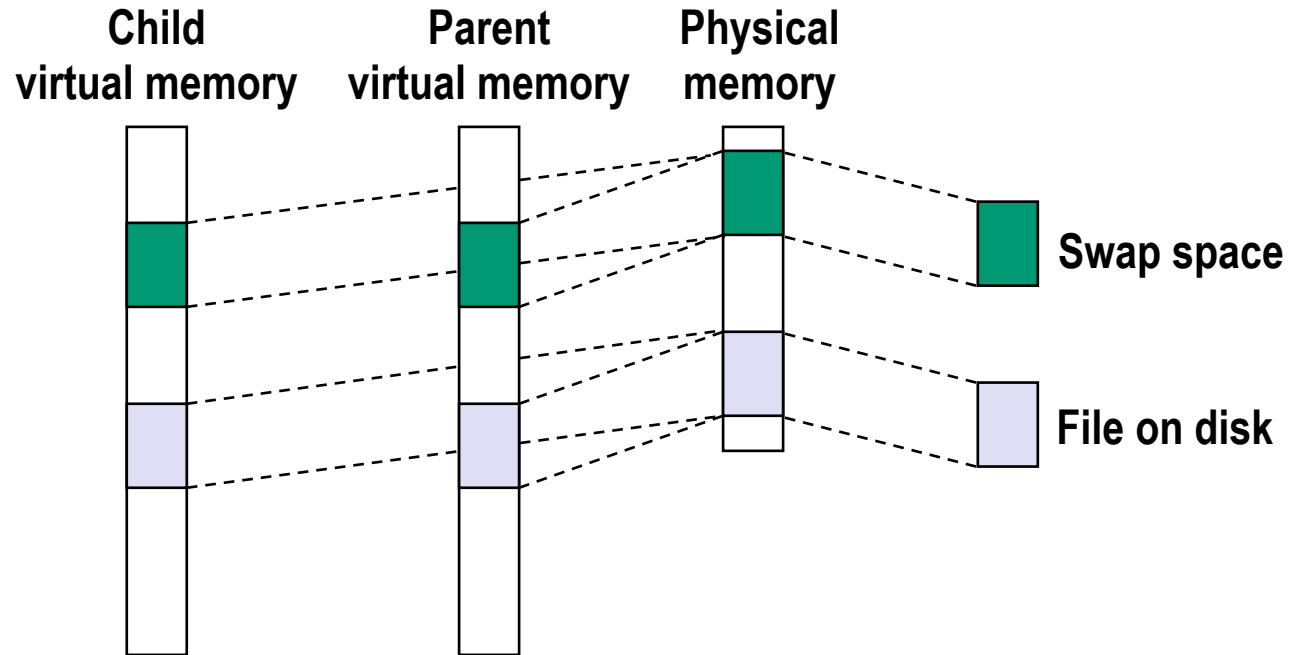
- **Clever trick:**
  - Just duplicate the page tables
  - Mark everything read only
  - Copy only on write faults

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**