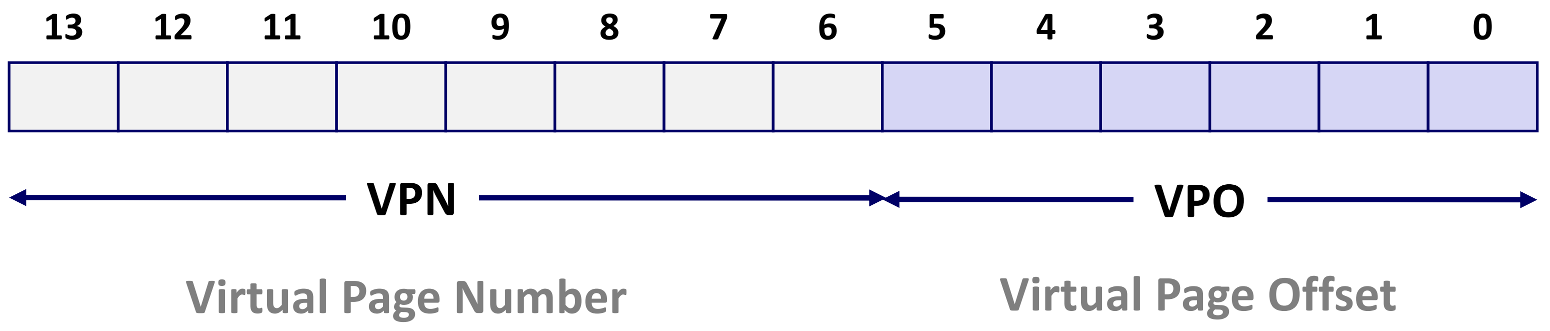
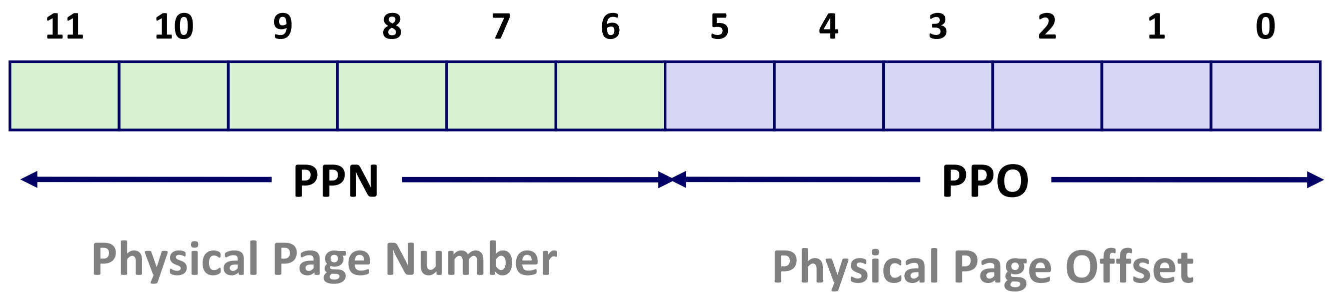
Today

- **Simple memory system example**
- Case study: Core i7/Linux memory system
- Memory mapping

Simple Memory System Example

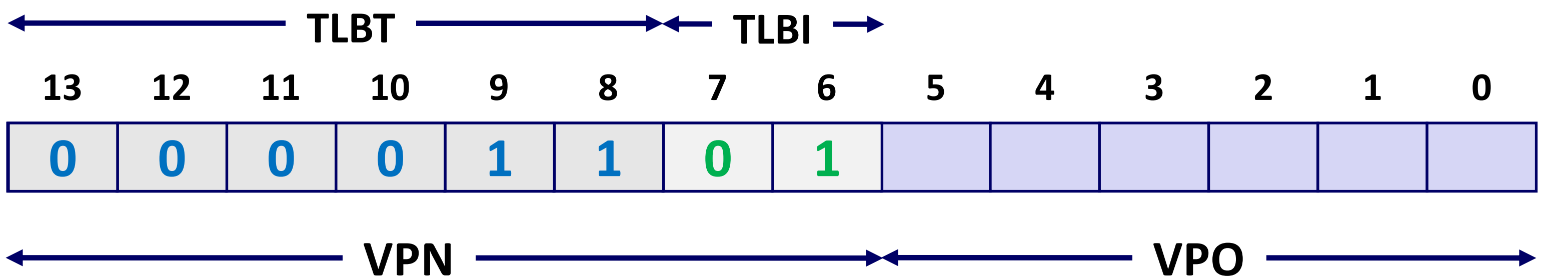
- **Addressing**
  - 14-bit virtual addresses
  - 12-bit physical address
  - Page size = 64 bytes





Simple Memory System TLB

- 16 entries
- 4-way associative



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

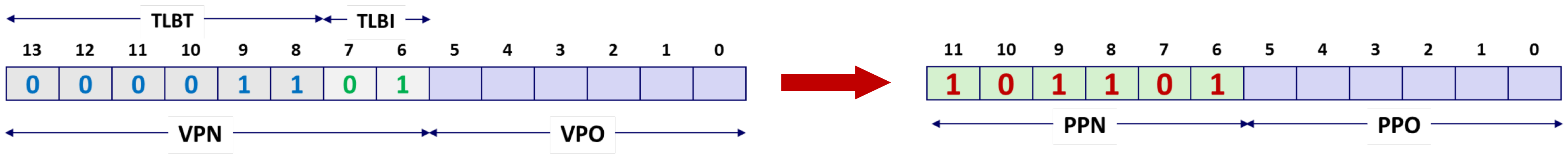
Simple Memory System Page Table

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

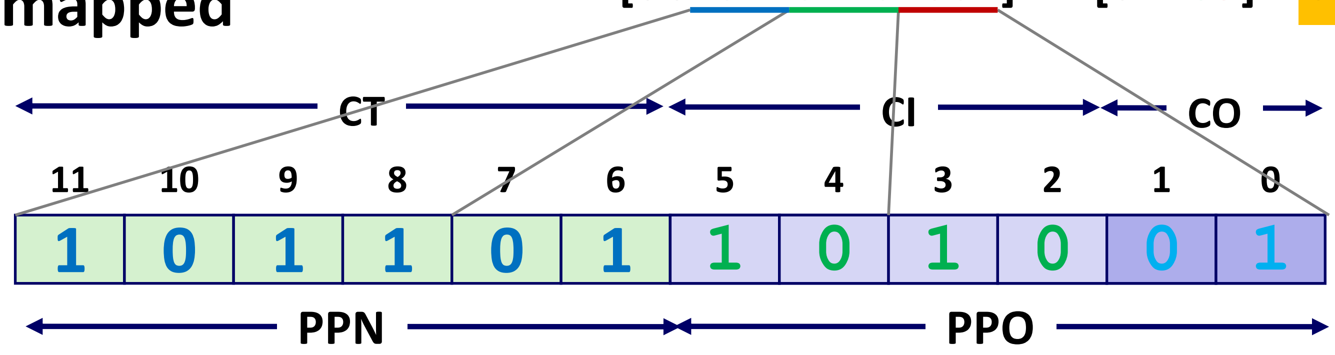
0x0D → 0x2D



Simple Memory System Cache

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

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

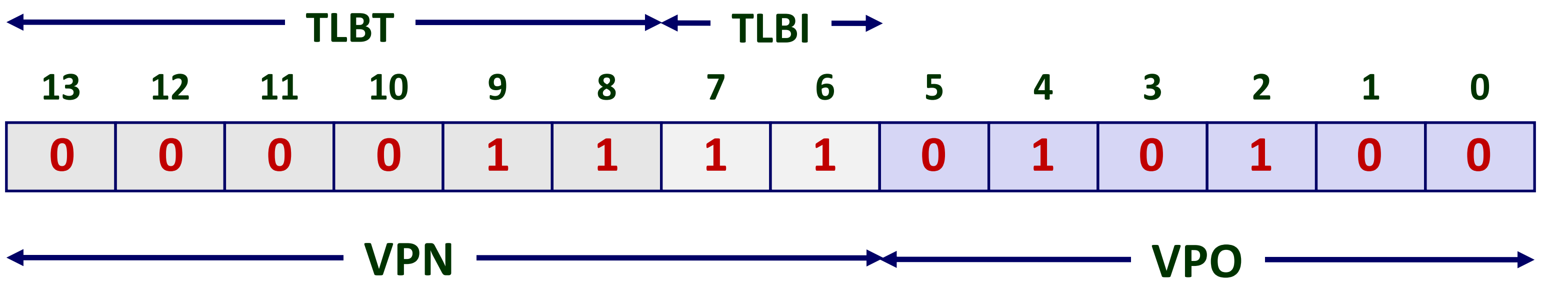


Idx	Tag	Valid	B0	B1	B2	B3
0	19	1	99	11	23	11
1	15	0	–	–	–	–
2	1B	1	00	02	04	08
3	36	0	–	–	–	–
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	–	–	–	–
7	16	1	11	C2	DF	03

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

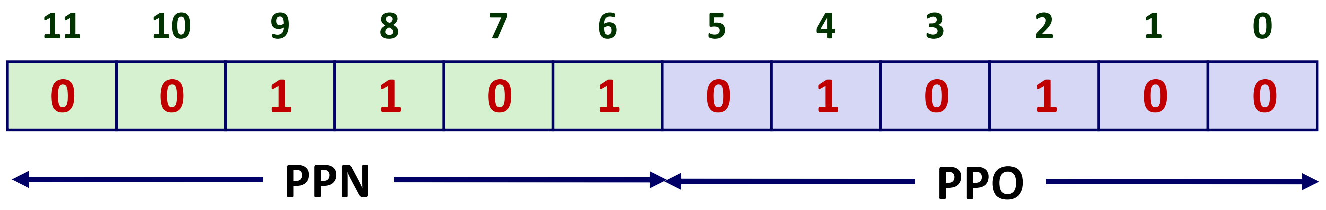


VPN 0x0F TLBI 0x3 TLBT 0x03 TLB Hit? Y Page Fault? N PPN: 0x0D

TLB

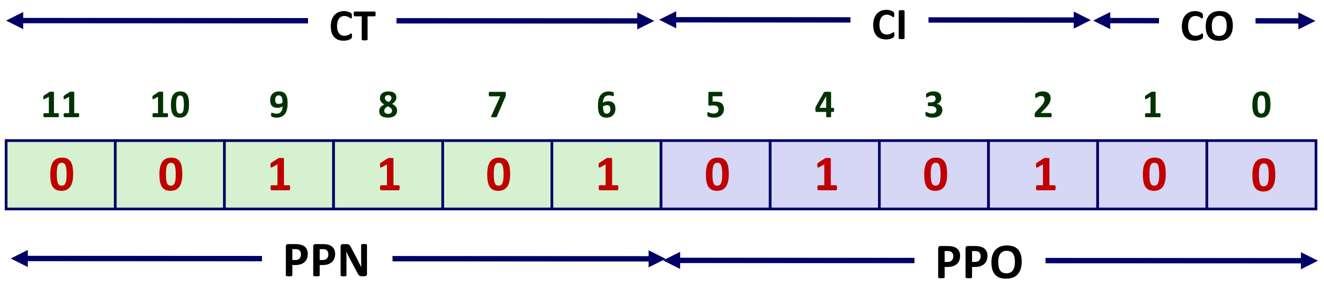
Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0

Physical Address



Address Translation Example

Physical Address



CO _0_ CI 0x5 CT 0x0D Hit? _Y_ Byte: 0x36

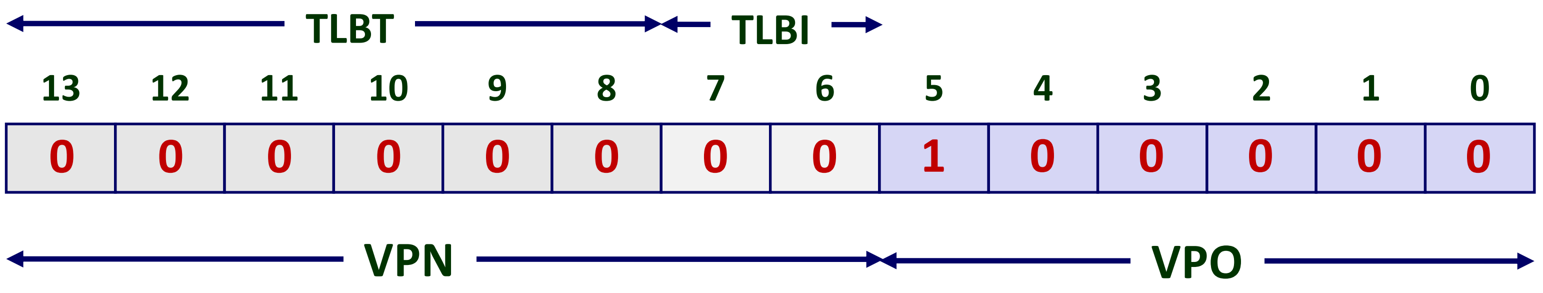
Cache

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	–	–	–	–

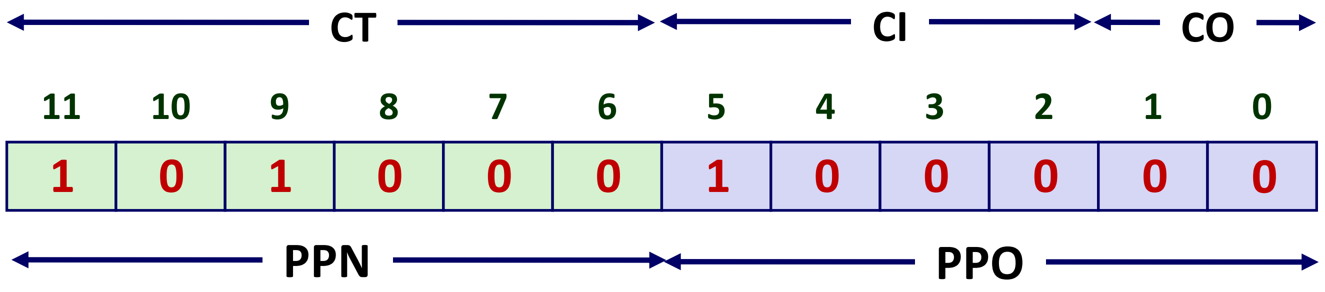
Address Translation Example: TLB/Cache Miss

Virtual Address: 0x0020



VPN 0x00 TLBI 0 TLBT 0x00 TLB Hit? N Page Fault? N PPN: 0x28

Physical Address



CO 0 CI 0x8 CT 0x28 Hit? __ Byte: _____

Page table

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

Address Translation Example: TLB/Cache Miss

Cache

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	–	–	–	–

Physical Address

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

CO_0_ CI 0x8 CT 0x28 Hit? N Byte: Mem

Virtual Memory Exam Question

Problem 5. (10 points):

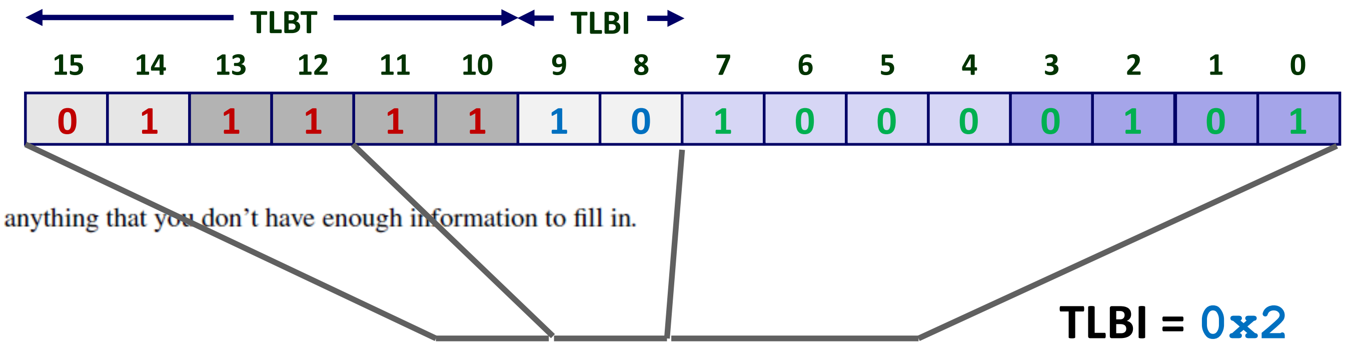
Assume a System that has

1. A two way set associative TLB
2. A TLB with 8 total entries
3. 2^8 byte page size
4. 2^{16} bytes of virtual memory
5. one (or more) boats

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

TLB

Index	Tag	PPN	Valid
0	0x13	0x30	1
	0x34	0x58	0
1	0x1F	0x80	0
	0x2A	0x72	1
2	0x1F	0x95	1
	0x20	0xAA	0
3	0x3F	0x20	1
	0x3E	0xFF	0



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	0	1	0	0	0	0	1	0	1

A. Use the TLB to fill in the table. Strike out anything that you don't have enough information to fill in.

Virtual Address	Physical Address
0x7E85	0x9585
0xD301	-------
0x4C20	0x3020
0xD040	-------
-------	0x5830

0x7E85 = 0x0111111010000101 $\rightarrow$

TLBI = 0x2

TLBT = 0x1F

0x7E85 → 0x9585

Exam: http://www.cs.cmu.edu/~213/oldexams/exam2b-s11.pdf (solution)

Quiz Time!

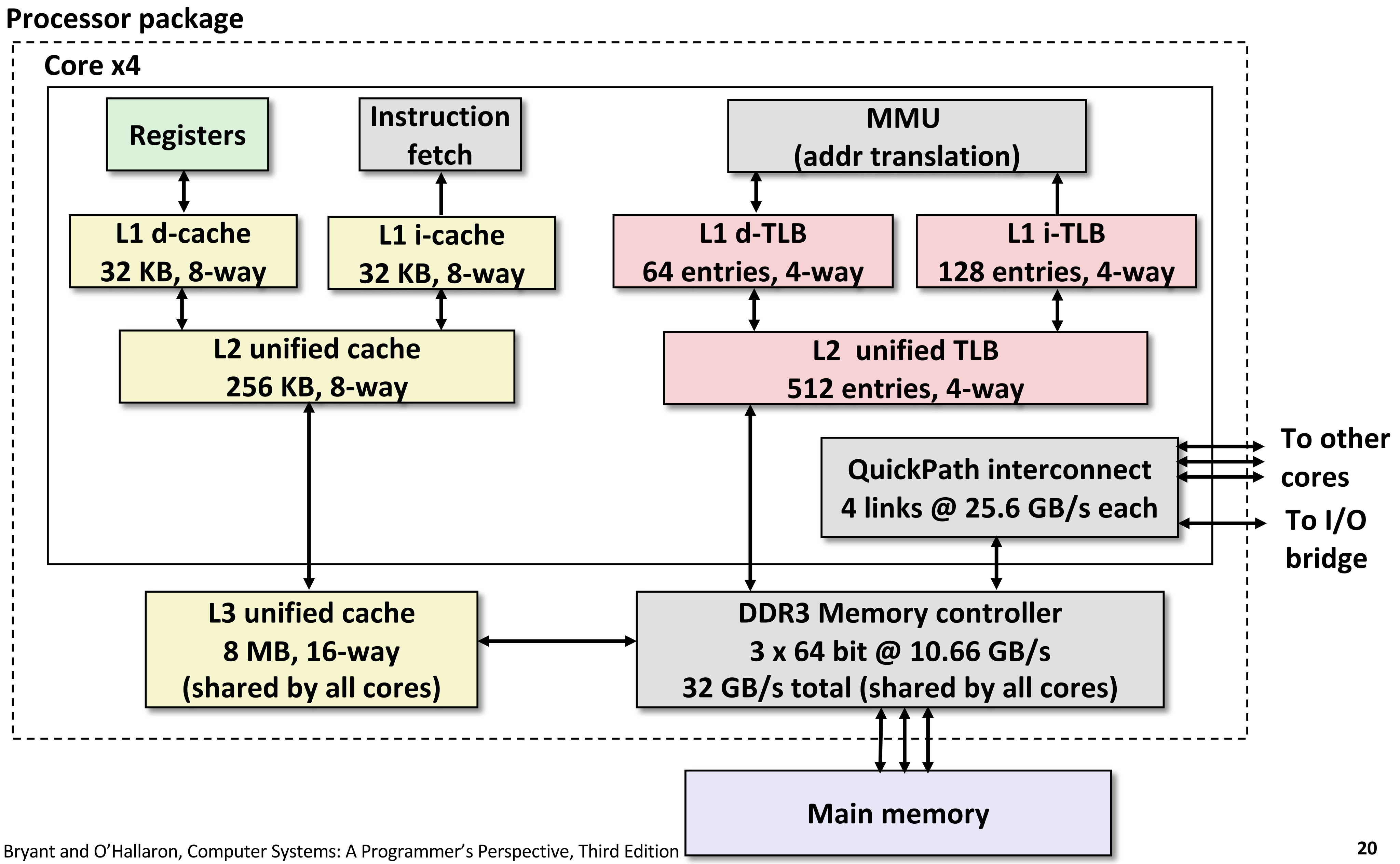
Check out:

https://canvas.cmu.edu/courses/10968

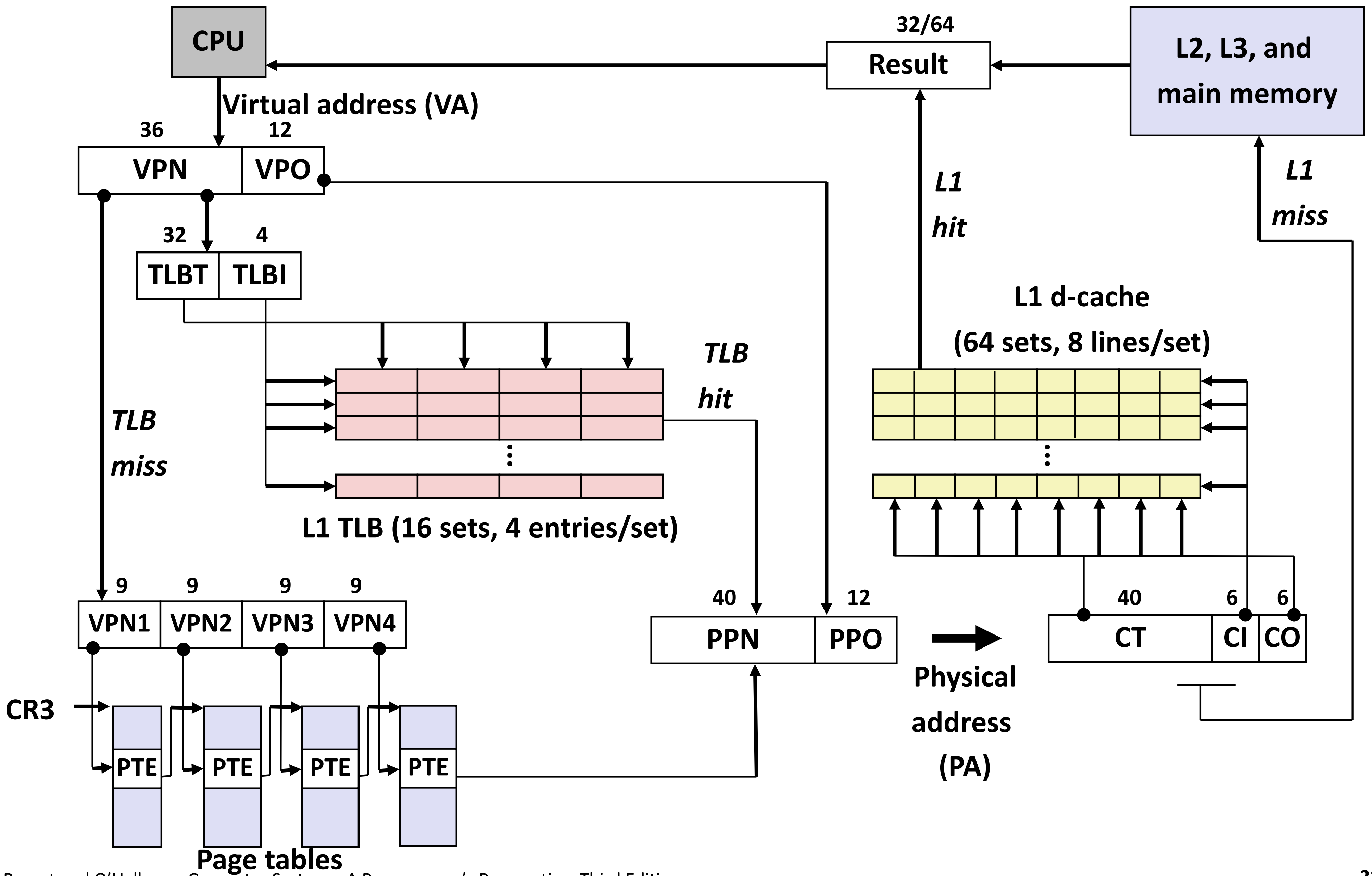
Today

- Simple memory system example
- Case study: Core i7/Linux memory system
- Memory mapping

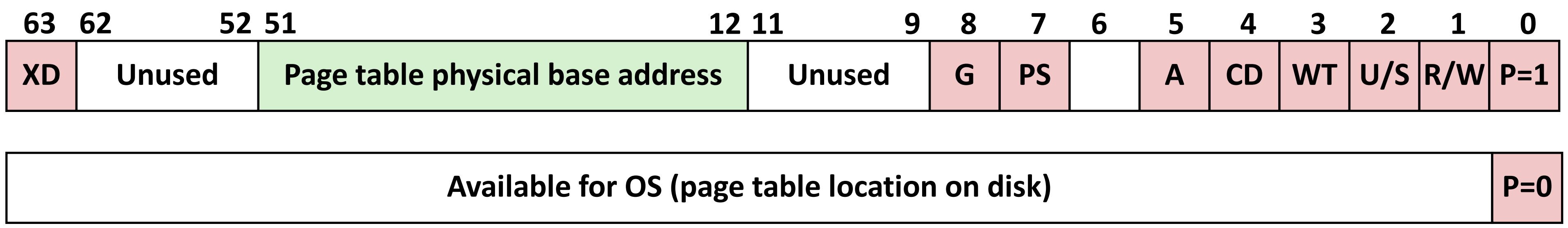
Intel Core i7 Memory System



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

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

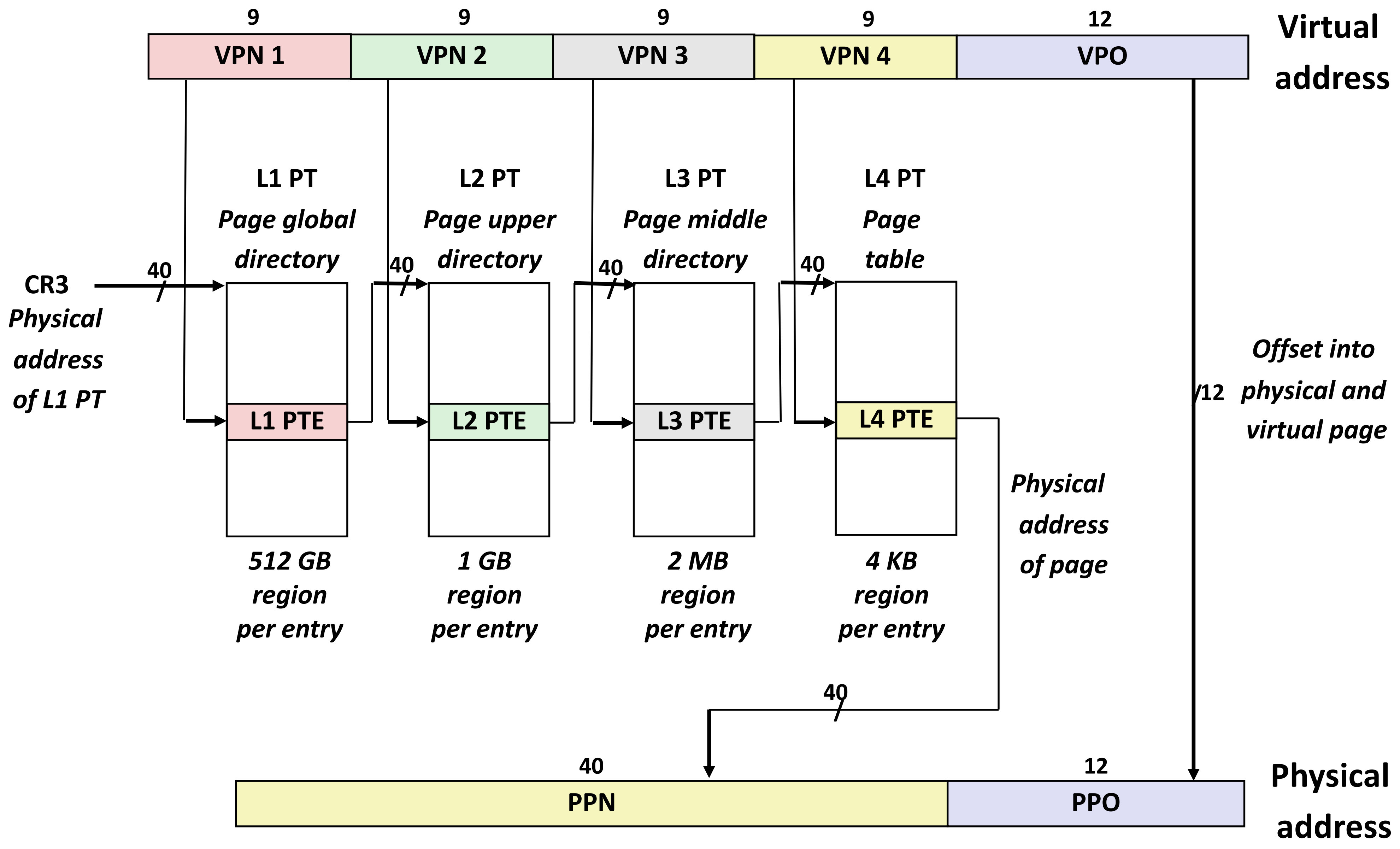
A: Reference bit (set by MMU on reads and writes, cleared by software)

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

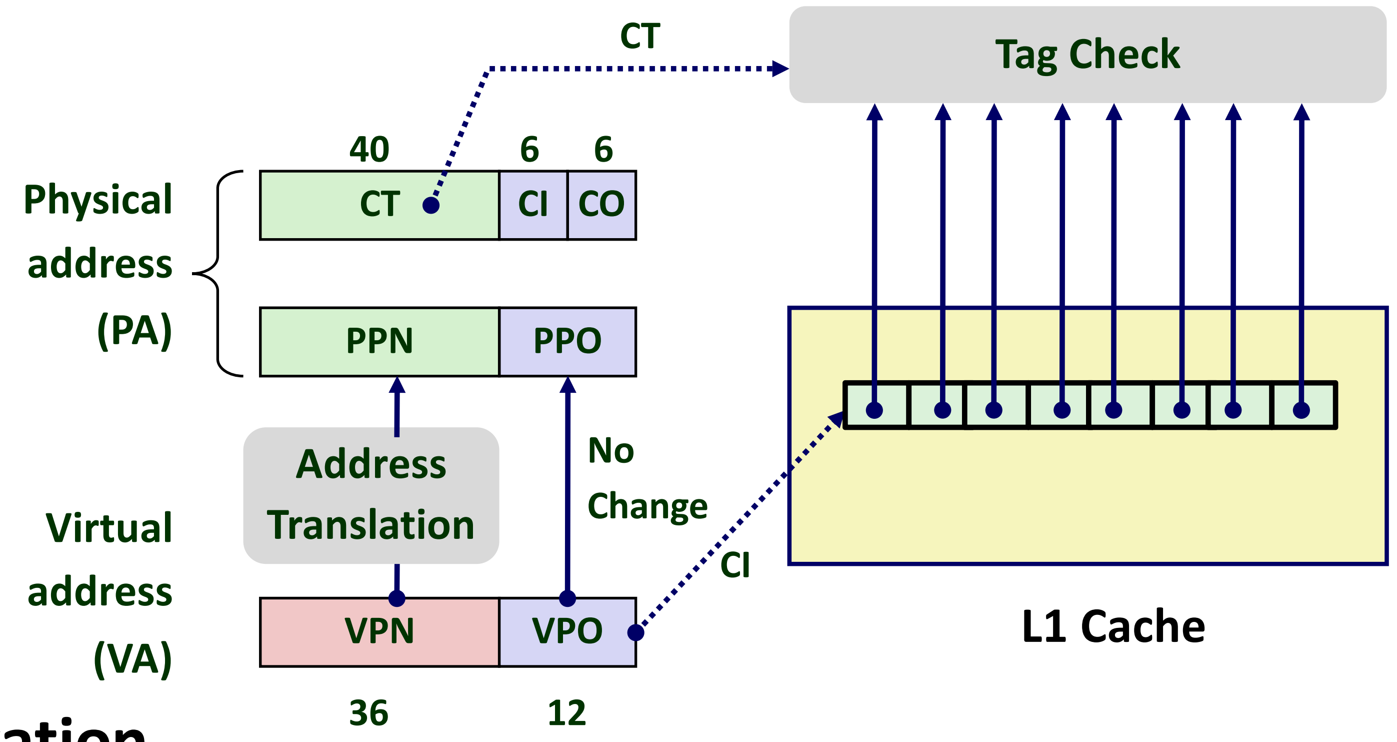
Page physical base address: 40 most significant bits of physical page address
(forces pages to be 4KB aligned)

XD: Disable or enable instruction fetches from this page.

Core i7 Page Table Translation

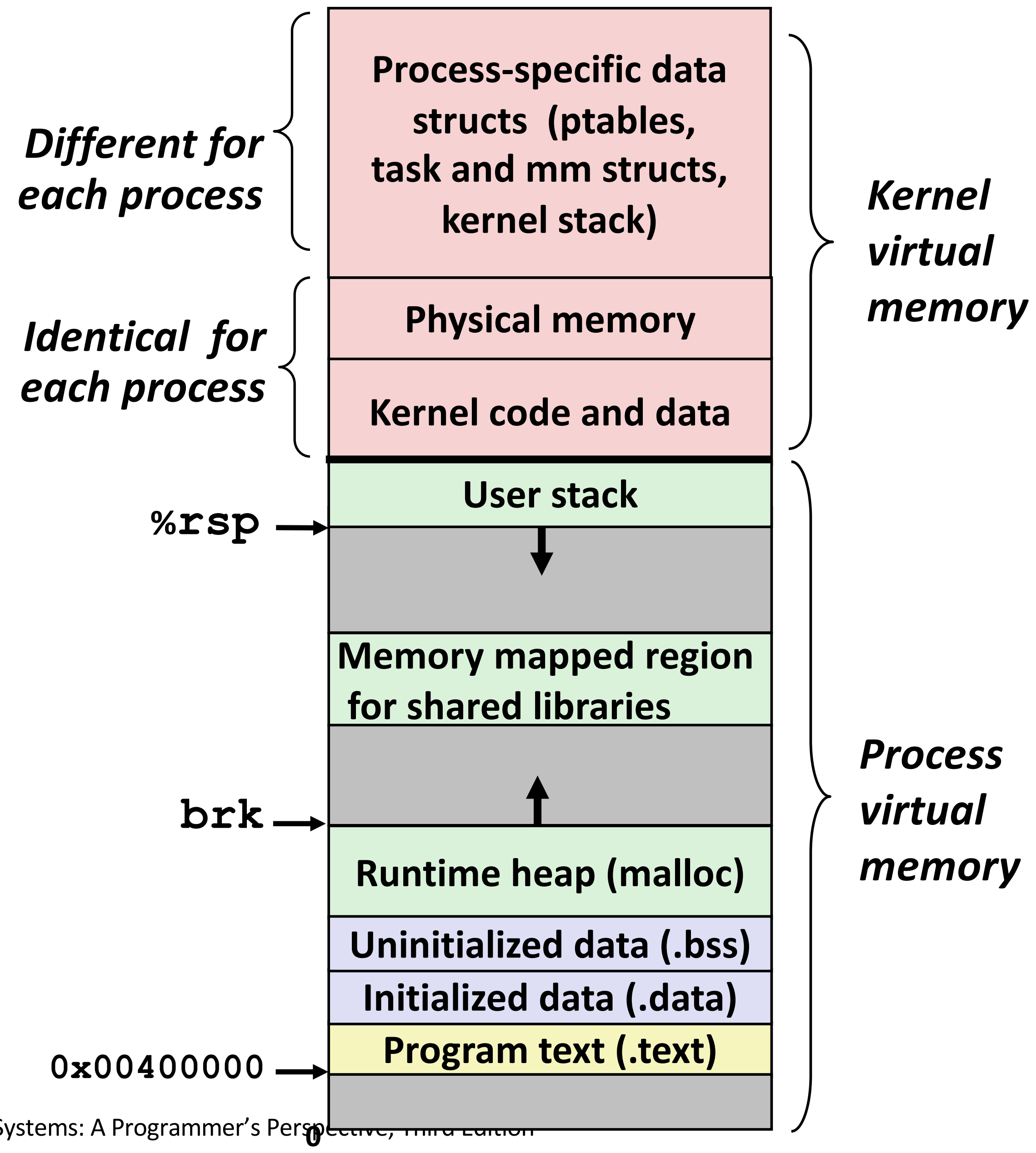


Cute Trick for Speeding Up L1 Access

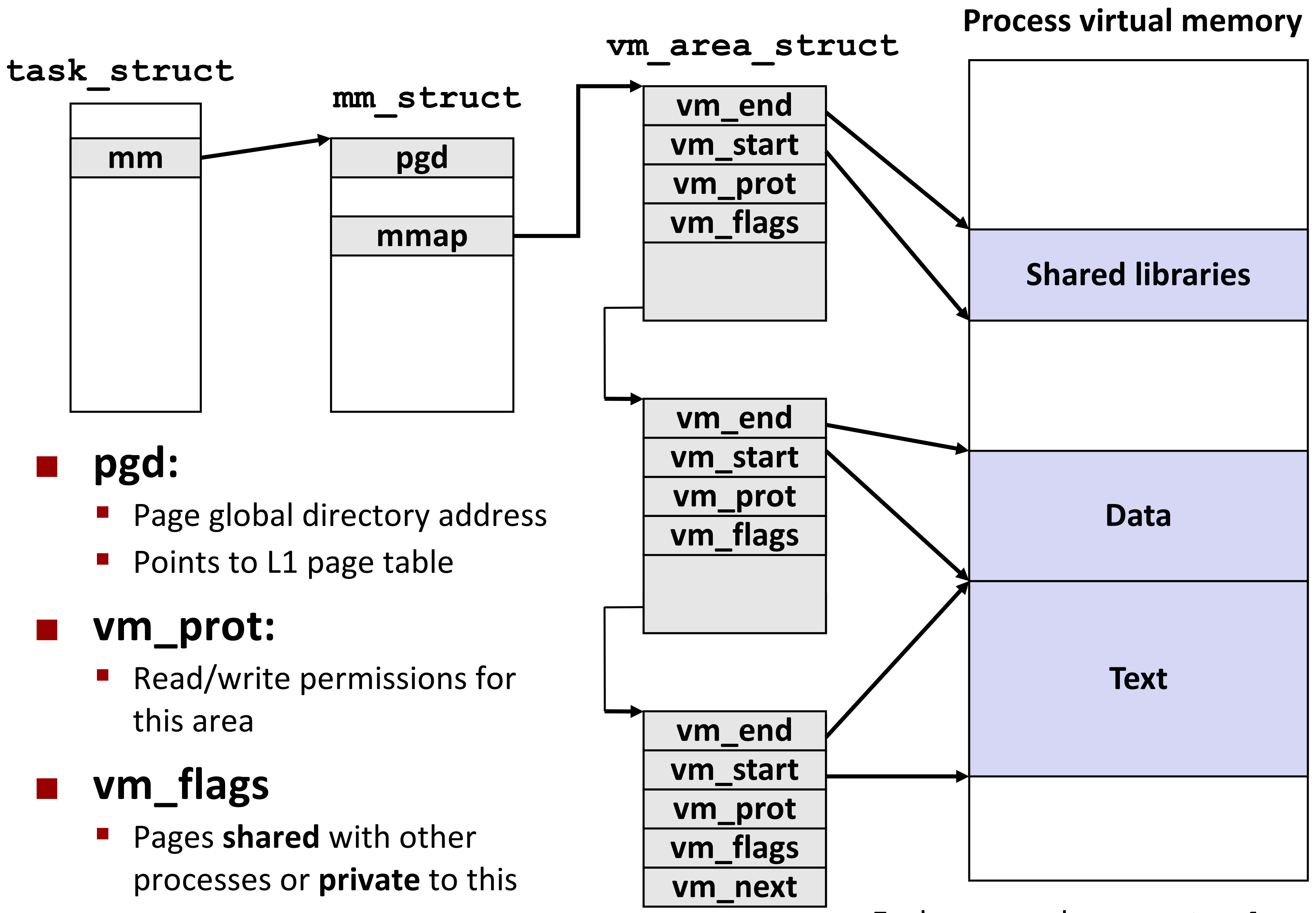


- **Observation**
  - Bits that determine CI identical in virtual and physical address
  - Can index into cache while address translation taking place
  - Generally we hit in TLB, so PPN bits (CT bits) available next
  - ***"Virtually indexed, physically tagged"***
  - Cache carefully sized to make this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

Virtual Address Space of a Linux Process


Different for each process
Identical for each process
Process-specific data structs (ptables, task and mm structs, kernel stack)
Physical memory
Kernel code and data
Kernel virtual memory
User stack
%rsp
Memory mapped region for shared libraries
brk
Runtime heap (malloc)
Uninitialized data (.bss)
Initialized data (.data)
Program text (.text)
0x00400000
0
Process virtual memory

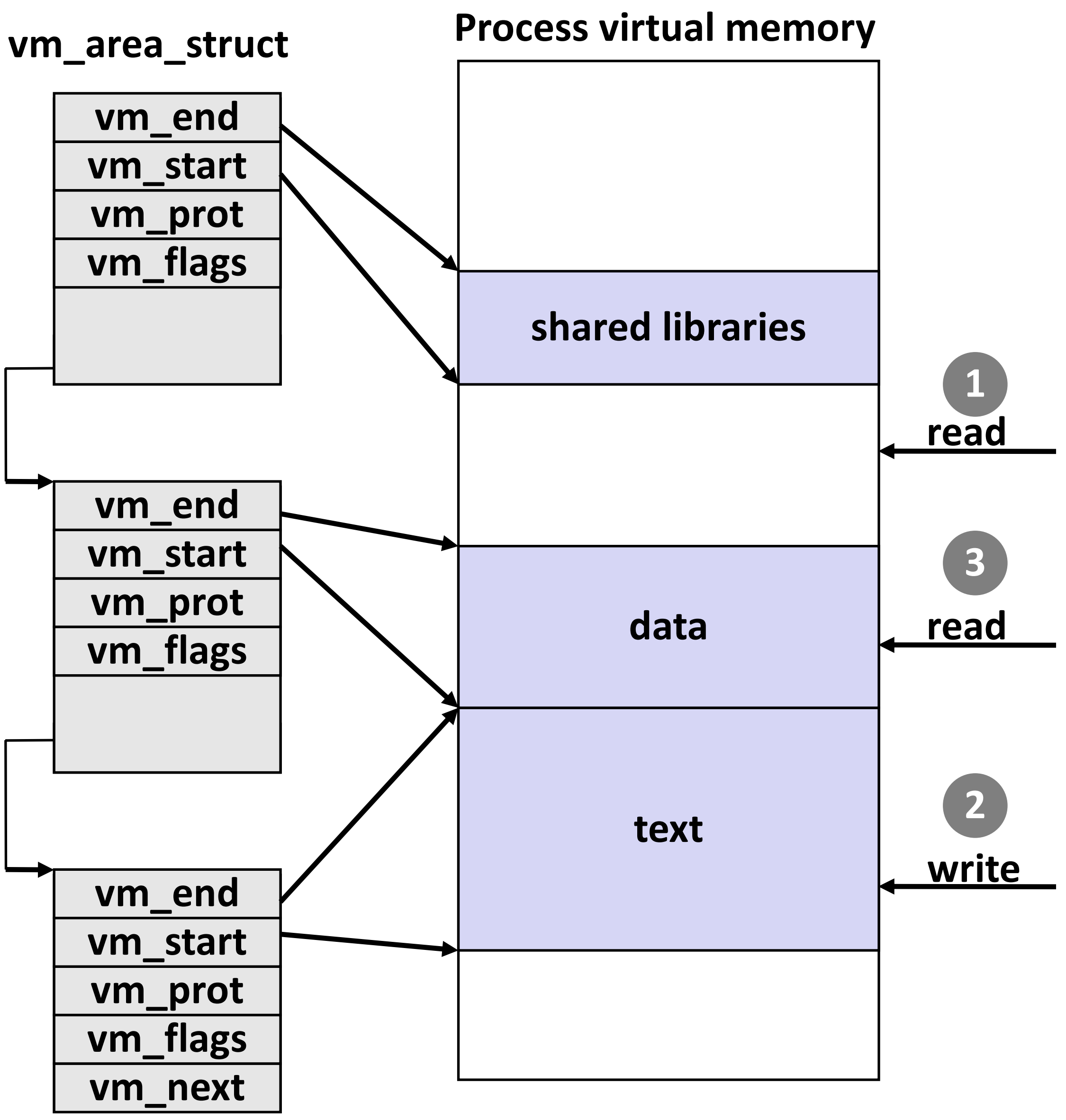
Linux Organizes VM as Collection of "Areas"



- **pgd:**
  - Page global directory address
  - Points to L1 page table
- **vm_prot:**
  - Read/write permissions for this area
- **vm_flags**
  - Pages **shared** with other processes or **private** to this process

Each process has own **`task_struct`**, etc

Linux Page Fault Handling



Segmentation fault:
accessing a non-existing page

Normal page fault

Protection exception:
e.g., violating permission by writing to a read-only page (Linux reports as Segmentation fault)

Today

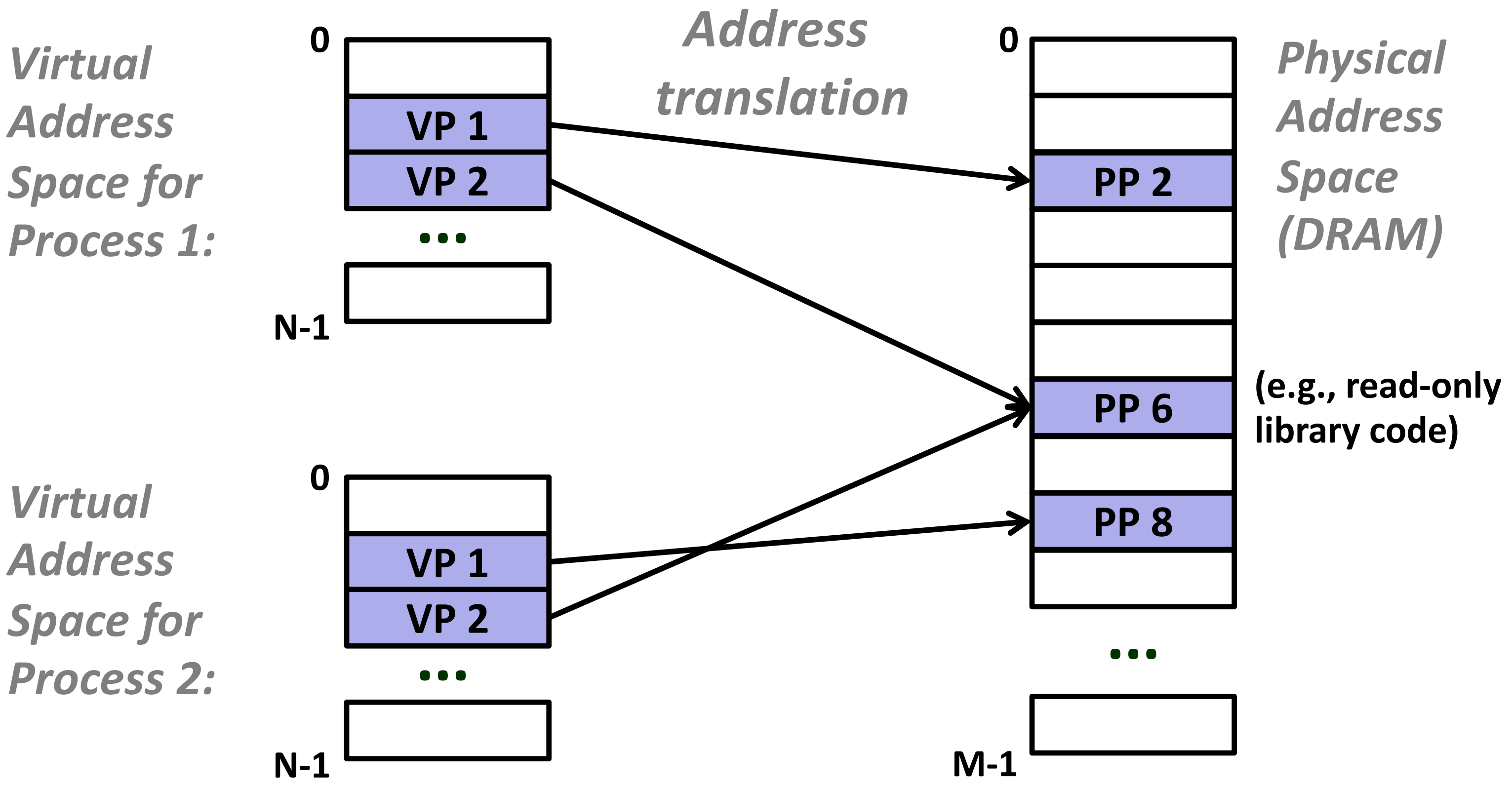
- Simple memory system example
- Case study: Core i7/Linux memory system
- **Memory mapping**

Memory Mapping

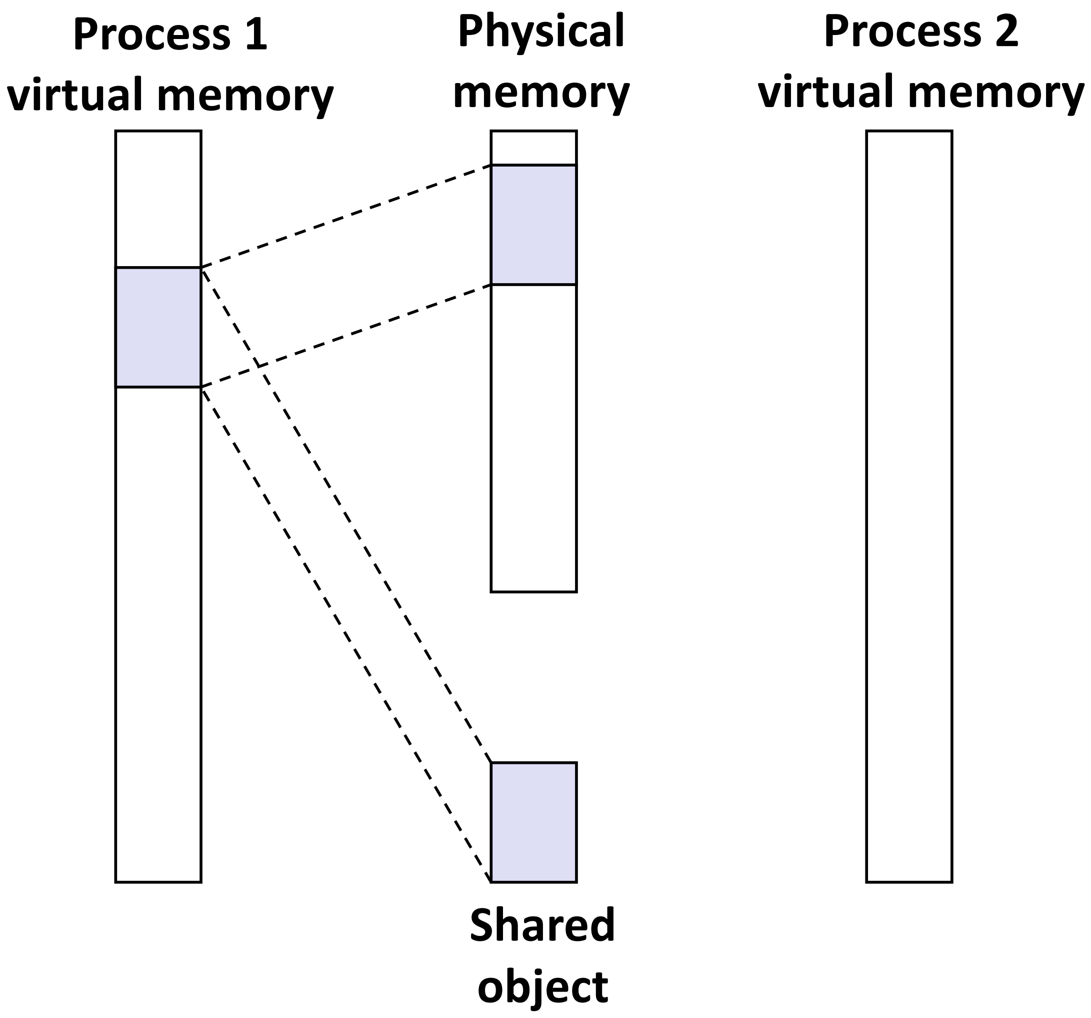
- **VM areas initialized by associating them with disk objects.**
  - Called ***memory mapping***
- **Area can be *backed by* (i.e., get its initial values from) :**
  - ***Regular file*** on disk (e.g., an executable object file)
    - Initial page bytes come from a section of a file
  - ***Anonymous 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

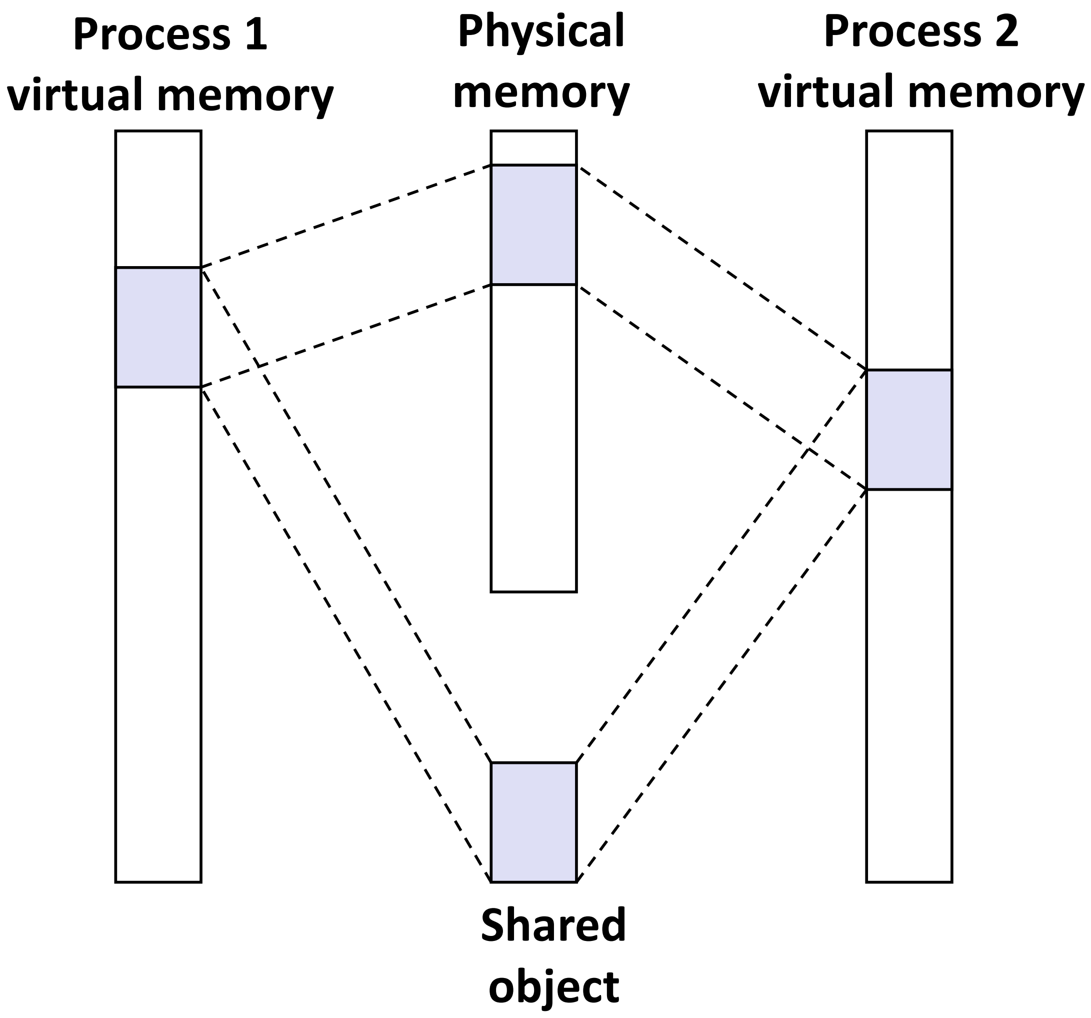

Virtual Address Space for Process 1:
0
VP 1
VP 2
...
N-1
Address translation
0
PP 2
PP 6
PP 8
...
M-1
Physical Address Space (DRAM)
(e.g., read-only library code)
Virtual Address Space for Process 2:
0
VP 1
VP 2
...
N-1

Sharing Revisited: Shared Objects



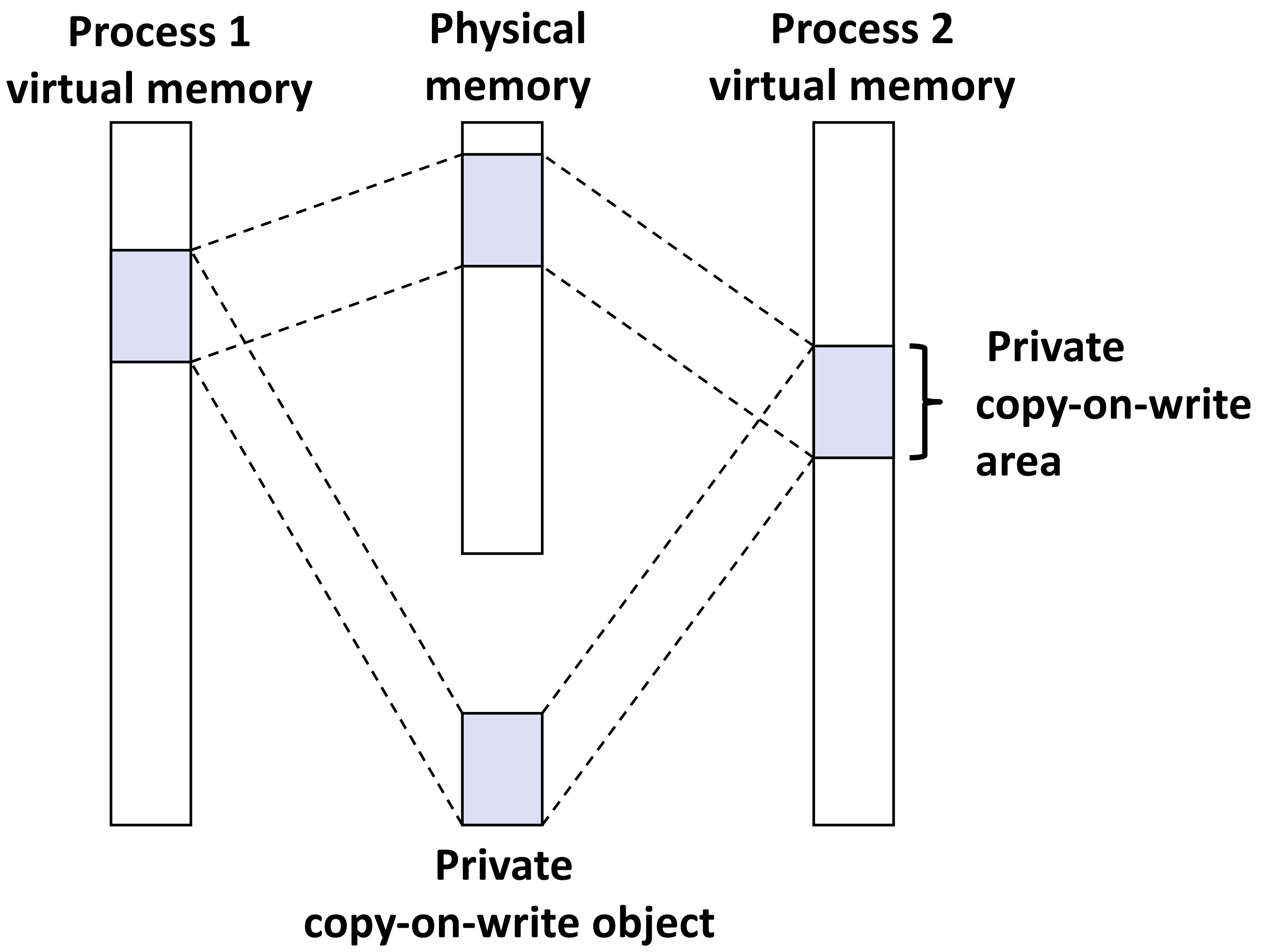
- Process 1 maps the shared object (on disk).

Sharing Revisited: Shared Objects



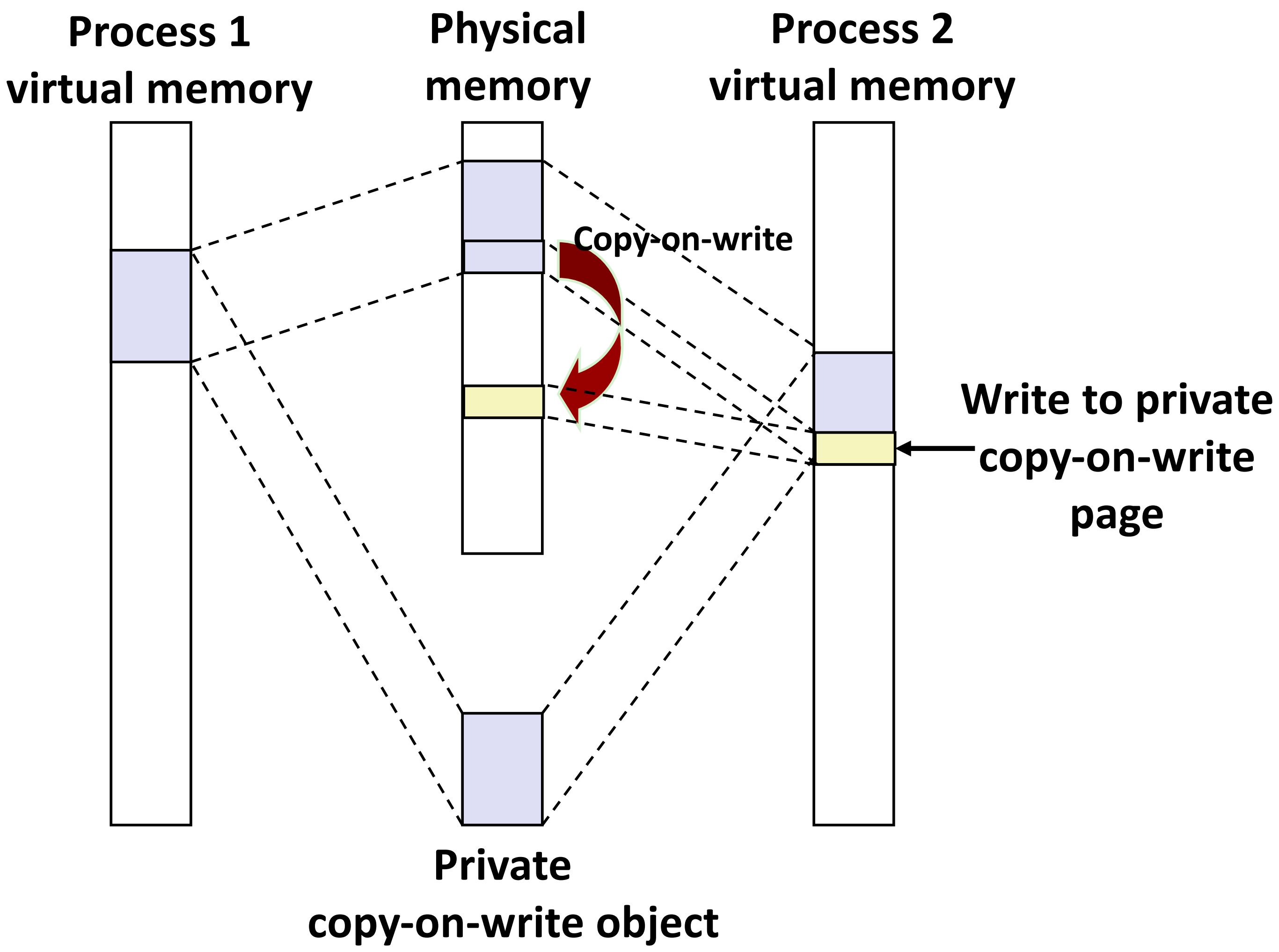
- Process 2 maps the same shared object.
- Notice how the virtual addresses can be different.
- But, difference must be multiple of page size

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.
- Instruction restarts upon handler return.
- Copying deferred as long as possible!

Finding Shareable Pages

- **Kernel Same-Page Merging**
  - 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
  - Limited to pages marked as likely candidates
  - Especially useful when processor running many virtual machines