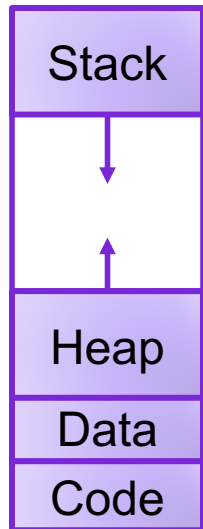


Lecture 18: Virtual Memory (cont'd)

CS 105

Review: Address Translation



Virtual Address

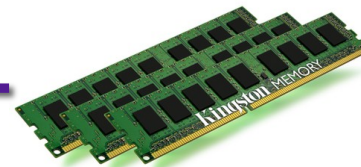


invalid

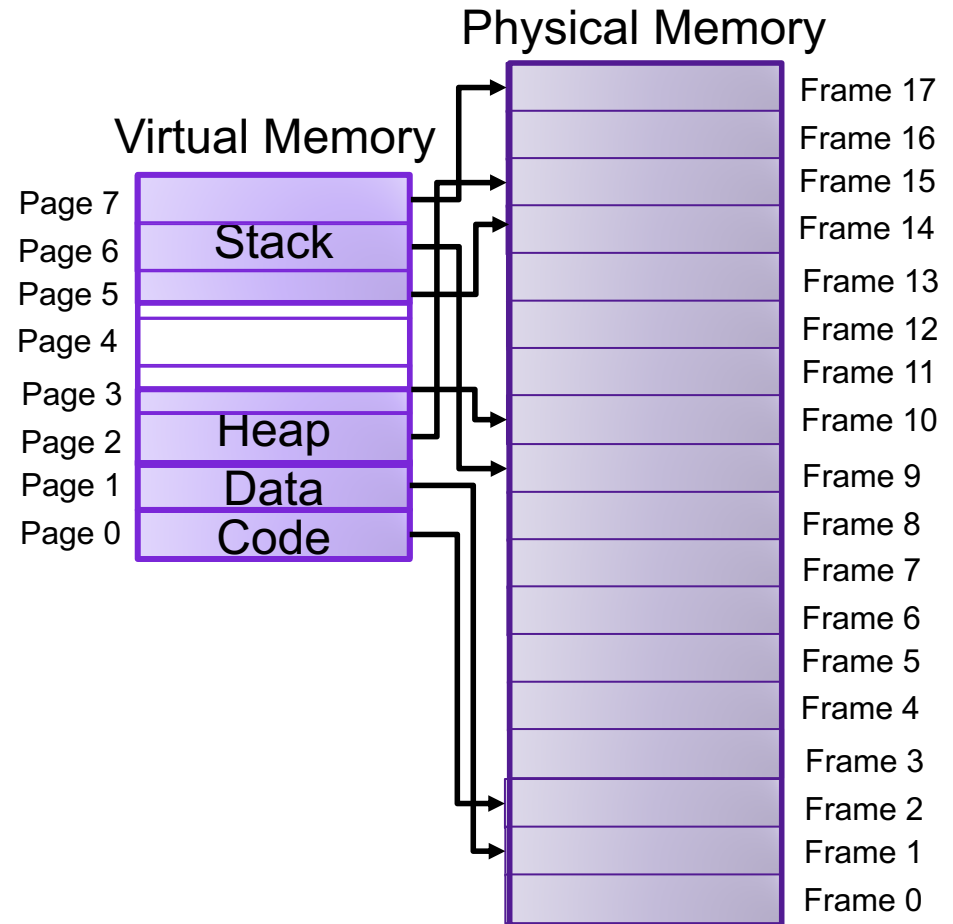
Exception

Physical Address

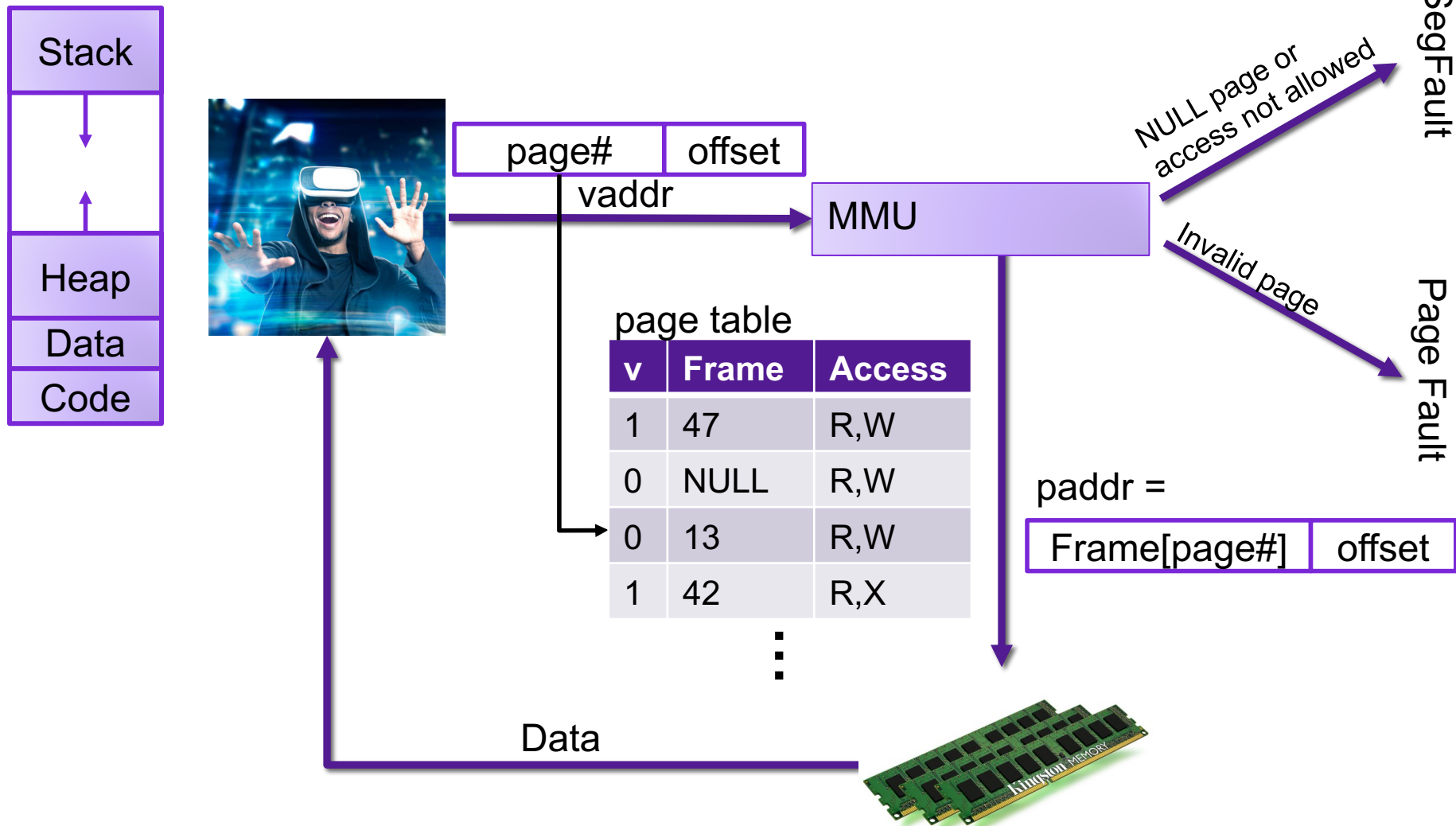
Data



Review: Paging



Review: Virtual Pages

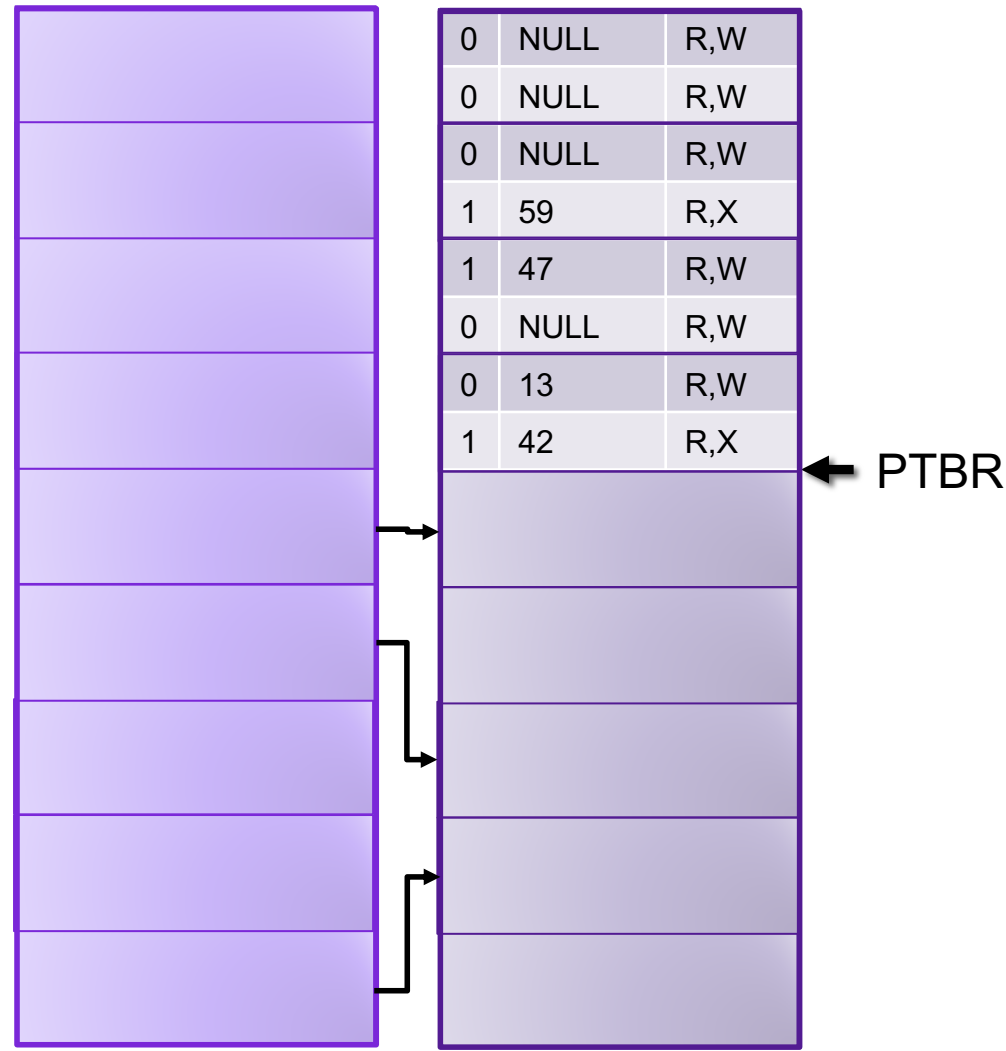


Review: Problems with Paging

- **Memory Consumption:** page table is really big
 - Example: consider 64-bit address space, 4KB (2^{12}) page size, assume each page table entry is 8 bytes.
 - Larger pages increase internal fragmentation
- **Performance:** every data/instruction access requires *two* memory accesses:
 - One for the page table
 - One for the data/instruction

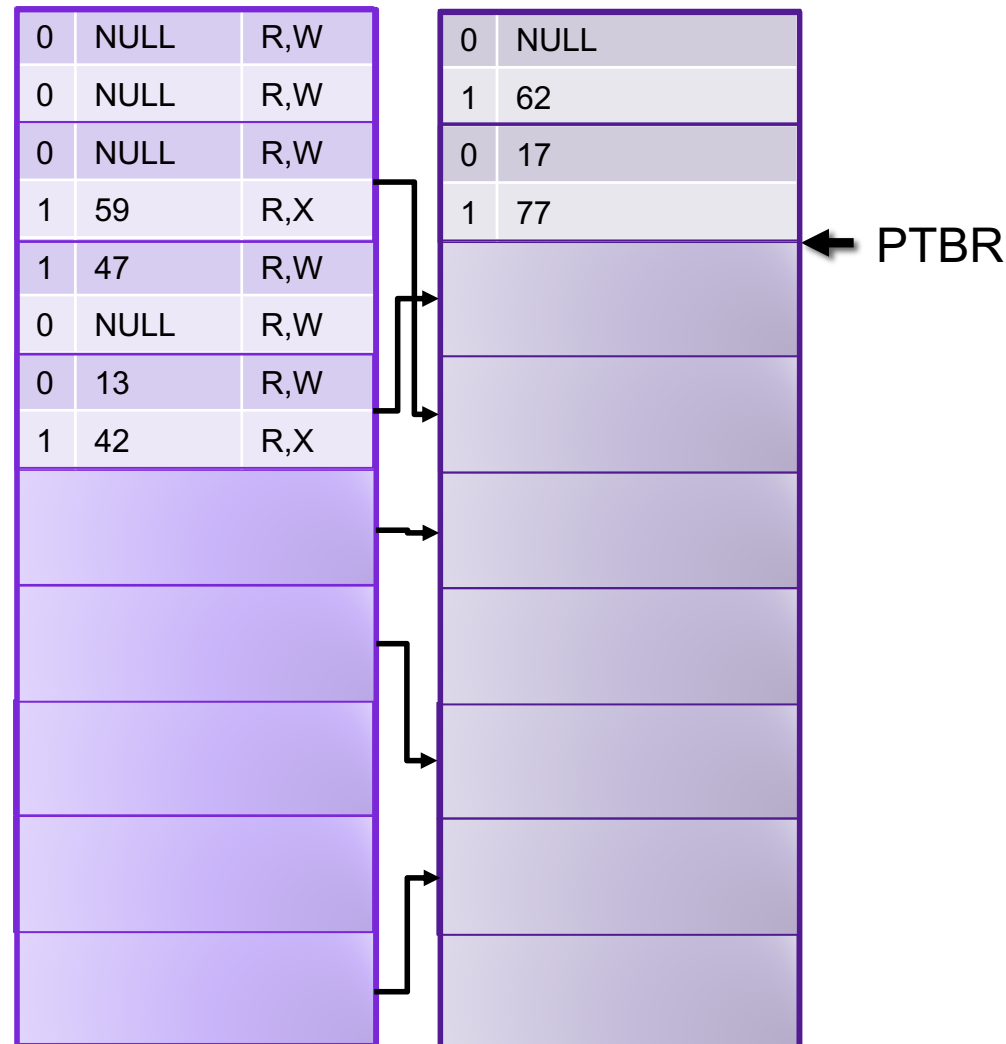
Traditional Paging

- page table is stored in physical memory
- implemented as array of page table entries
- Page Table Base Register (PTBR) stores physical address of beginning of page table
- Page table entries are accessed by using the page number as the index into the page table

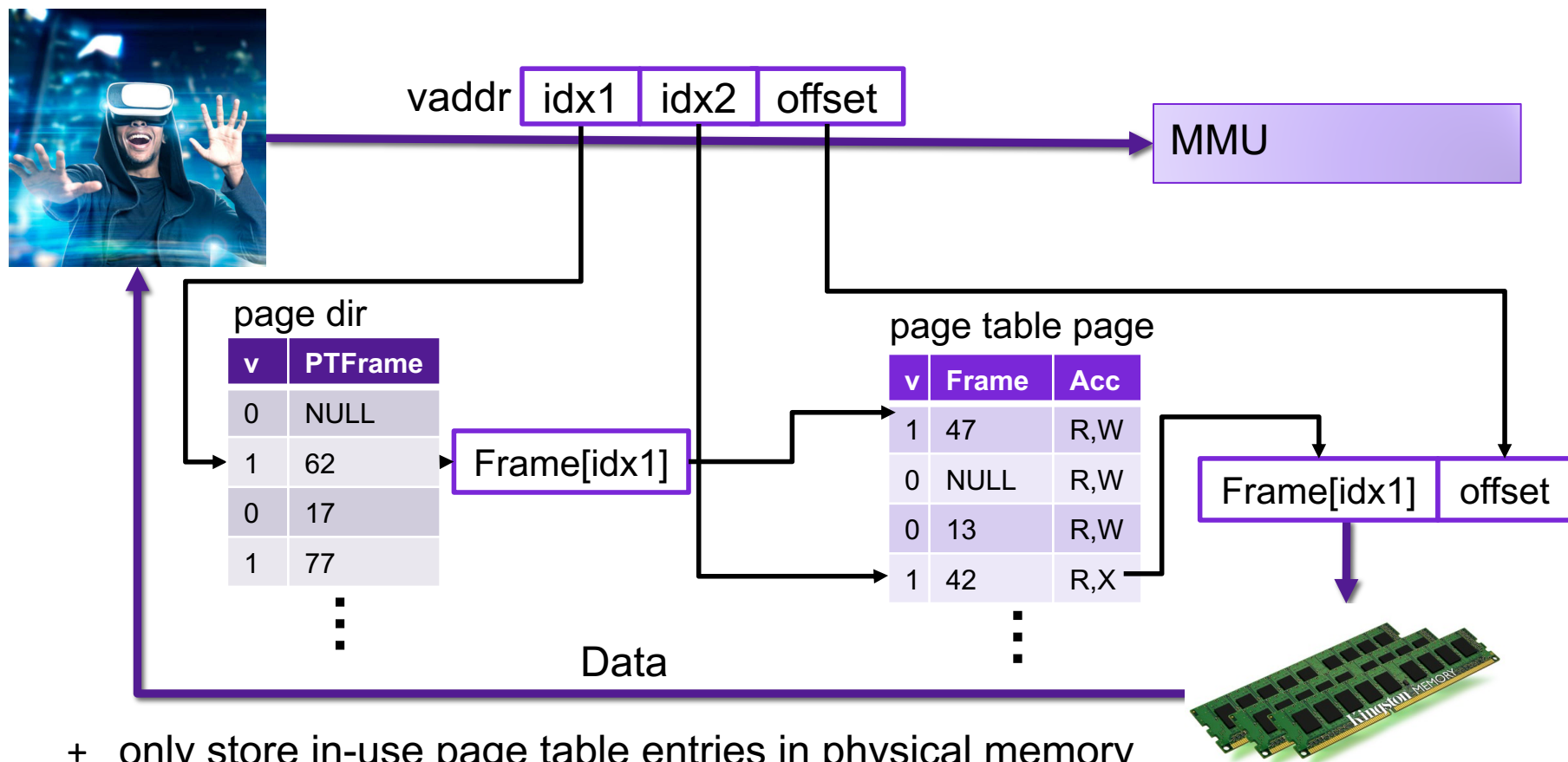


Two-level Page Tables

- page table is stored in virtual memory pages
- page directory is stored in physical memory (page table for the page table)
- Implemented as array of page directory entries
- Page Table Base Register (PTBR) stores physical address of beginning of page directory



Two-level Page Tables



- + only store in-use page table entries in physical memory
- + easier to allocate page table
- more memory accesses

Exercise 1: Two-level Page Tables

- Assume you are working on an architecture with a 32-bit virtual address space in which idx1 is 4 bits, idx2 is 12 bits, and offset is 16 bits.

4 bit idx1	12 bit idx2	16 bit offset
------------	-------------	---------------
- How big is a page in this architecture? **2^{16} bytes = 64 KB**
- How big is a page table entry in this architecture? **16 bytes**

Exercise 2: Two-level Page Tables


Assume you are still working on that architecture.

4 bit idx1 | 12 bit idx2 | 16 bit offset

Compute the physical address corresponding to each of the virtual address (or answer "invalid"):

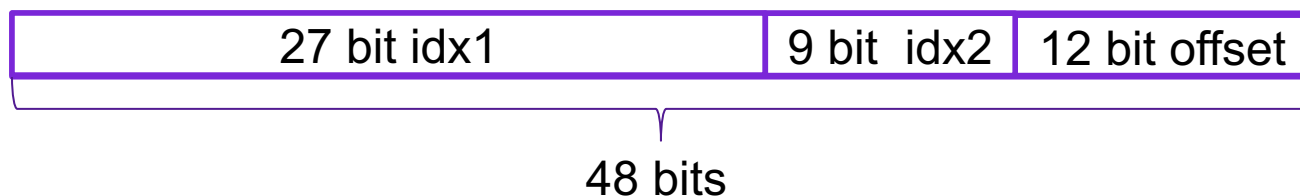
- a) 0x00000000 **0x00470000**
- b) 0x20022002 **invalid**
- c) 0x10015555 **0xCAFE5555**

	v	PTFrame
0x0	1	0x0
0x1	1	0x2
0x2	0	NULL
0x3	0	NULL
		⋮
0xF	0	NULL

Frame	v	Frame	Acc	
Frame 0				
	0x0	1	0x0047	R,W
	0x1	0	NULL	R,W
	0x2	0	0x0013	R,W
	0x3	1	0x0042	R,X
			⋮	
Frame 1				
				
Frame 2				
	0x0	0	0x002A	R
	0x1	1	0xCAFE	R,W
	0x2	0	NULL	R,W
	0x3	0	13	R,W
			⋮	

Multi-level Page Tables

- Problem: How big does the page directory get? **128 MB**
 - Assume you have a 48-bit address space
 - Assume you have 4KiB pages
 - Assume you have 8 byte page table entries/page directory entries



- Goal: Page Table Directory should fit in one frame
- **Multi-level page tables:** add additional level(s) to tree

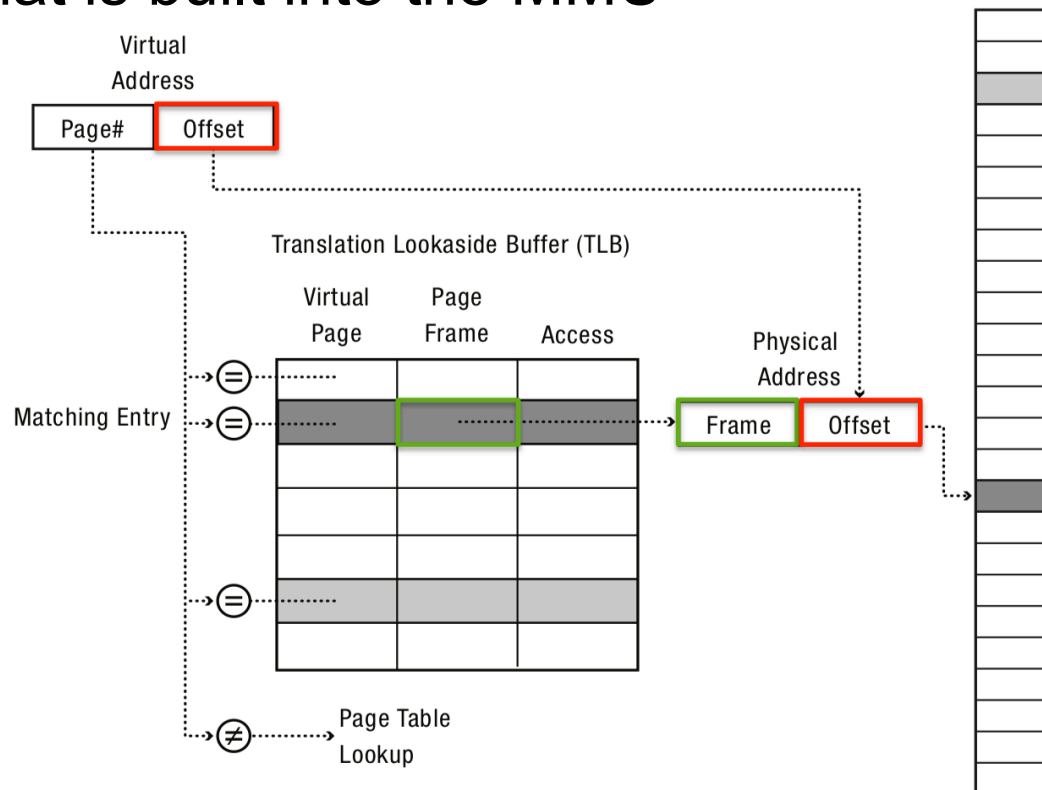


Review: Problems with Paging

- **Memory Consumption:** page table is really big
 - Example: consider 64-bit address space, 4KB (2^{12}) page size, assume each page table entry is 8 bytes.
 - Larger pages increase internal fragmentation
- **Performance:** every data/instruction access requires ~~two~~^{five} memory accesses:
 - One for ~~the page table~~ each of the four levels of page table
 - One for the data/instruction

Translation-Lookaside Buffer (TLB)

- General idea: if address translation is slow, cache some of the answers
- **Translation-lookaside buffer** is an address translation cache that is built into the MMU



Exercise 3: TLB

TLB												
idx	v	tag	PPN	v	tag	PPN	v	tag	PPN	v	tag	PPN
0	1	03	B	0	07	6	1	28	3	0	01	F
1	1	31	0	0	12	3	1	3E	4	1	0B	1
2	0	2A	A	0	11	1	1	1F	8	1	07	5
3	1	07	3	0	2A	A	0	1E	2	0	21	B

- Assume you are running on an architecture with a one-level page table with 4096 byte pages. For each of the following virtual addresses, determine whether the address translation is stored in the TLB. If so, give the corresponding physical address
 - 0x7E37C
 - 0x16A48

Exercise 3: TLB

TLB												
idx	v	tag	PPN	v	tag	PPN	v	tag	PPN	v	tag	PPN
0	1	03	B	0	07	6	1	28	3	0	01	F
1	1	31	0	0	12	3	1	3E	4	1	0B	1
2	0	2A	A	0	11	1	1	1F	8	1	07	5
3	1	07	3	0	2A	A	0	1E	2	0	21	B

- Assume you are running on an architecture with a one-level page table with 4096 byte pages. For each of the following virtual addresses, determine whether the address translation is stored in the TLB. If so, give the corresponding physical address

- 0x7E37C **0x837C**

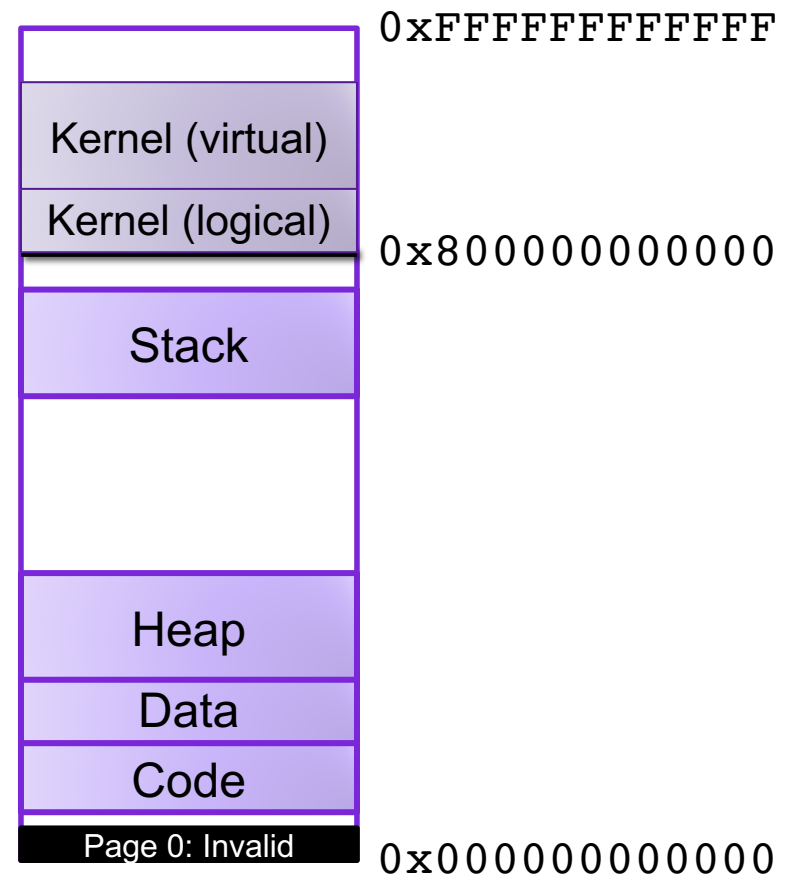
011111	10
--------	----

- 0x16A48 **TLB miss**

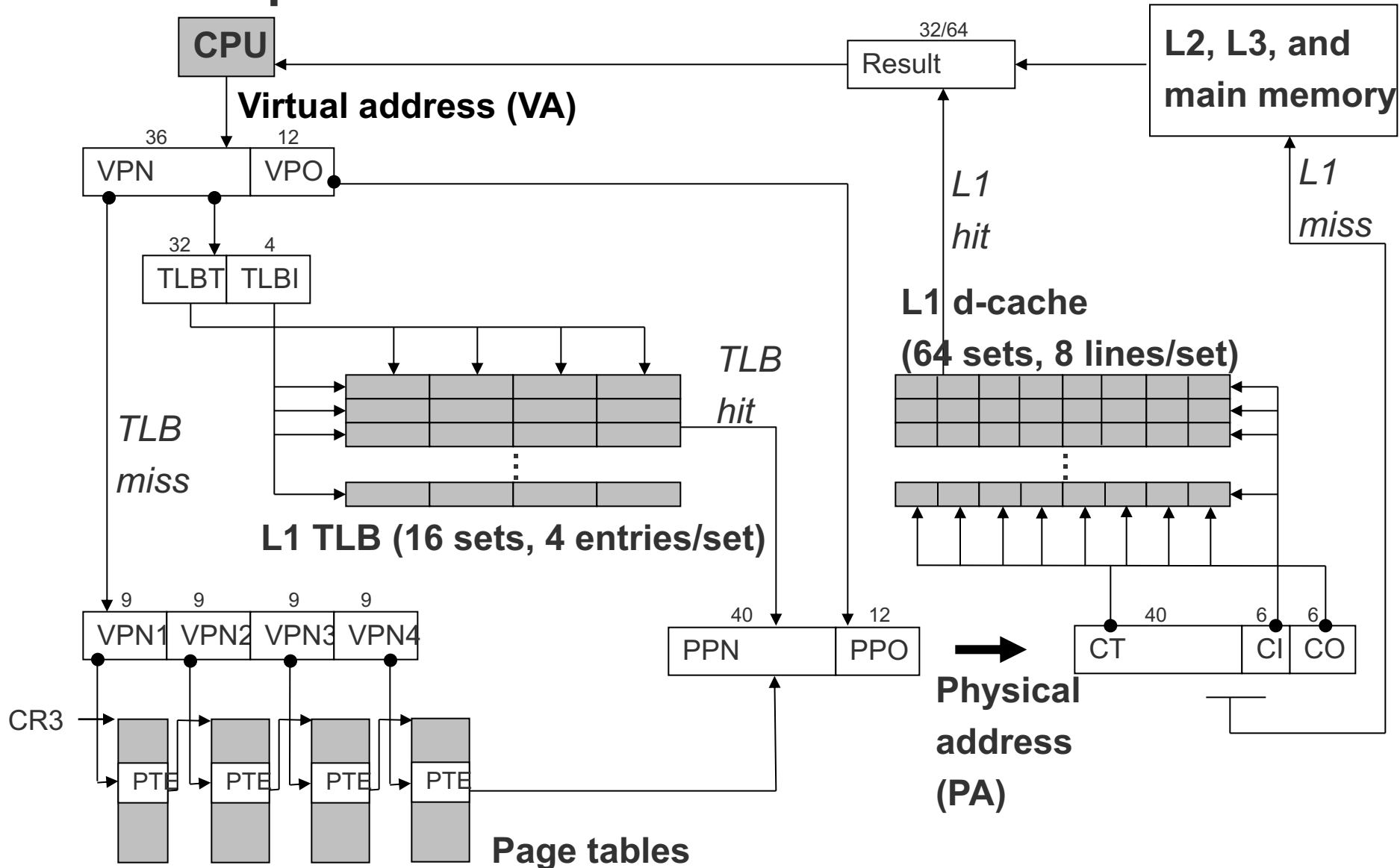
000101	10
--------	----

Example: The Linux x86 Address Space

- Use "only" 48-bit addresses (top 16 bits not used)
- 4KiB pages by default
 - supports larger "superpages"
- Four-level page table
- Physical memory stores memory pages, memory-mapped files, cached file pages
- Updates are periodically written to disk by background processes
- Page eviction algorithm uses variant of LRU called 2Q
 - approximates LRU with clock
 - maintains two lists (active/inactive)
- Stack is marked non-executable
- Virtual address of stack/heap start are randomized each time process is initialized



Example: Core i7 Address Translation



Exercise 4: Feedback

1. Rate how well you think this recorded lecture worked
 1. Better than an in-person class
 2. About as well as an in-person class
 3. Less well than an in-person class, but you still learned something
 4. Total waste of time, you didn't learn anything
2. How much time did you spend on this video (including exercises)?
3. Do you have any particular questions you'd like me to address in this week's problem session?
4. Do you have any other comments or feedback?