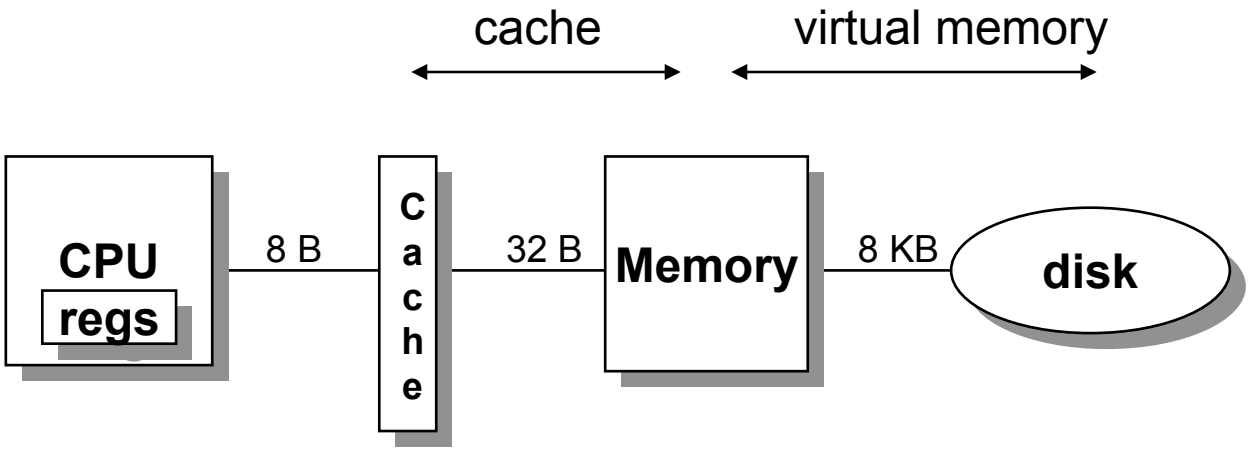
EE108B
Lecture 15
Virtual Memory II

Christos Kozyrakis
Stanford University
http://eeclass.stanford.edu/ee108b

Announcements

- PA2.2 due on 3/6

Review: Levels in Memory Hierarchy

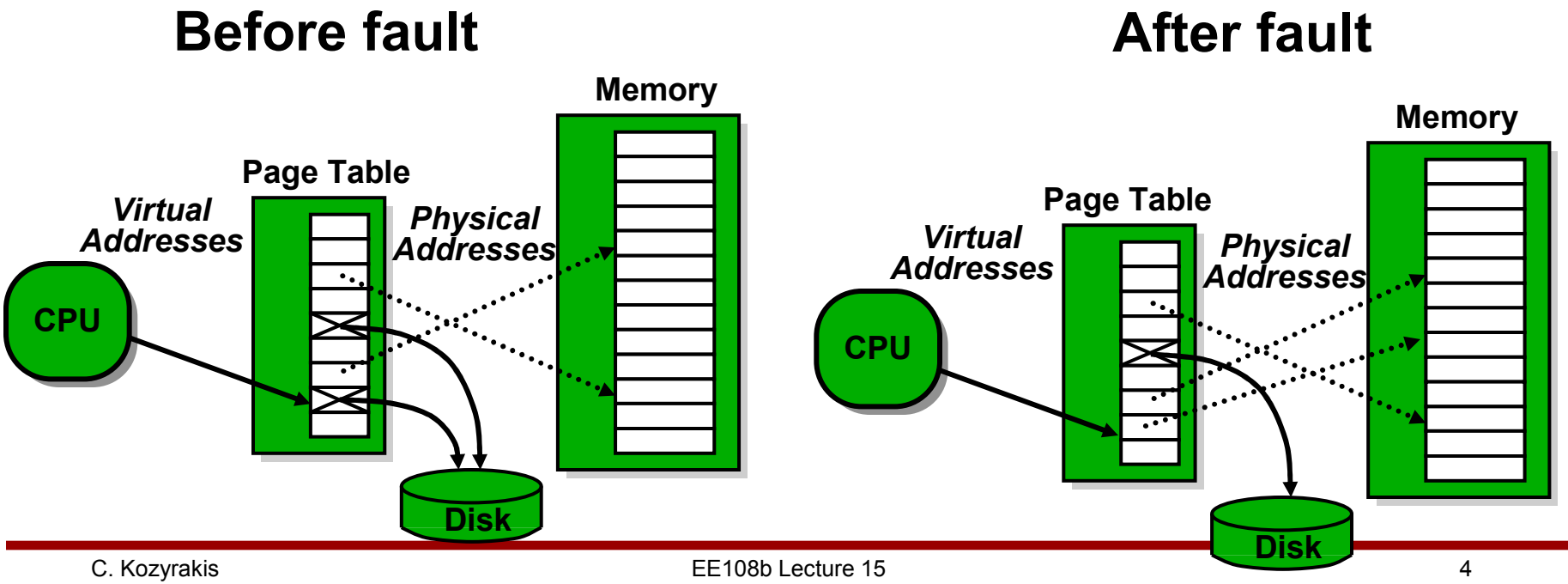


	Register	Cache	Memory	Disk Memory
size:	128 B	<4MB	< 16 GB	> 100 GB
Speed(cycles):	0.5-1	1-20	80-100	5-10 M
$/Mbyte:		$30/MB	$0.128/MB	$0.001/MB
block size:	8 B	32 B	8 KB	

larger, slower, cheaper

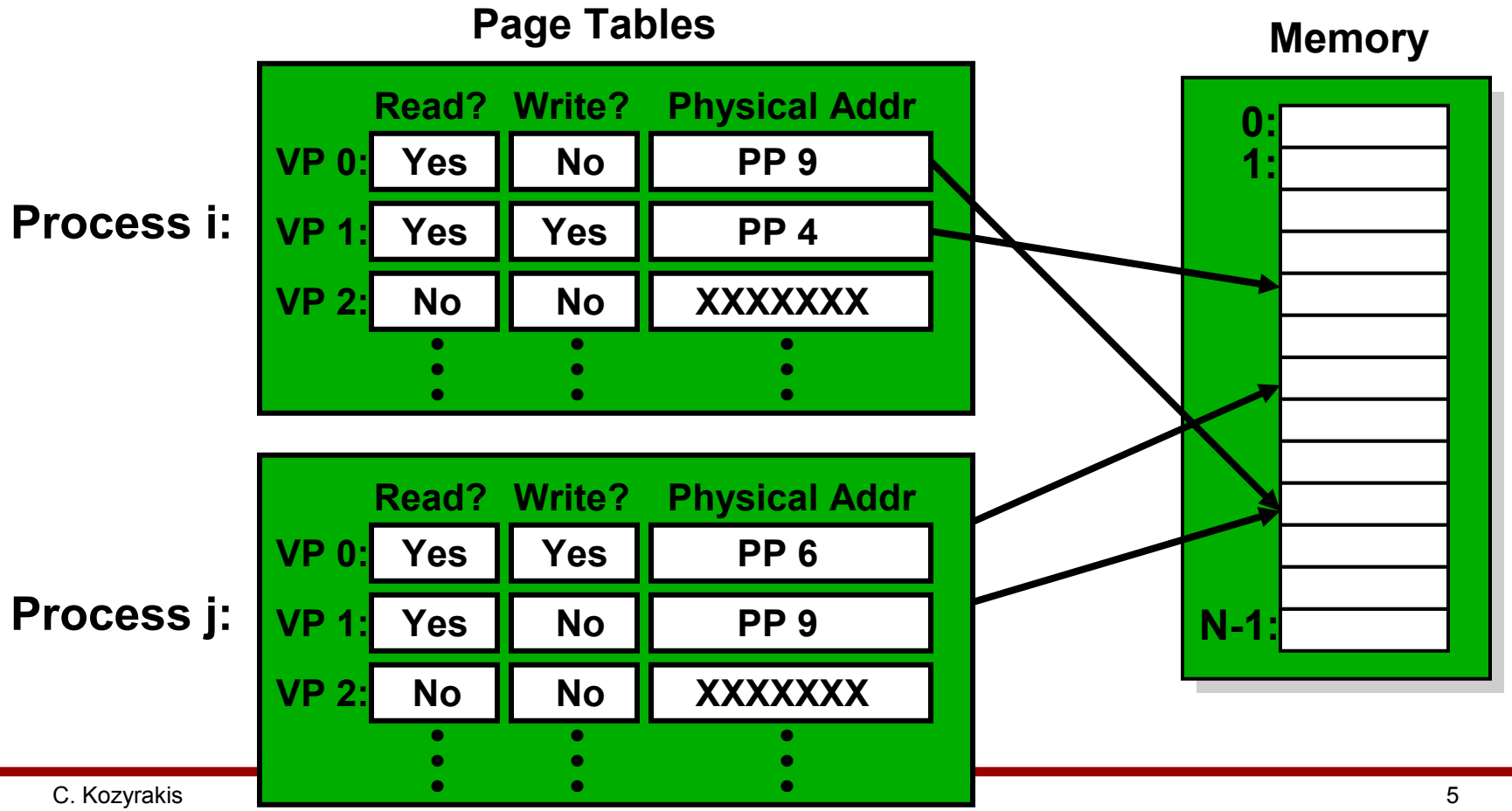
Review: Page Faults (Similar to "Cache Misses")

- What if an object is on disk rather than in memory?
  - Page table entry indicates virtual address not in memory
  - OS exception handler invoked to move data from disk into memory
    - current process suspends, others can resume
    - OS has full control over placement, etc.



Review: Page Tables

- Page table entry contains access rights information
  - hardware enforces this protection (trap into OS if violation occurs)



Review: Page Table Problems

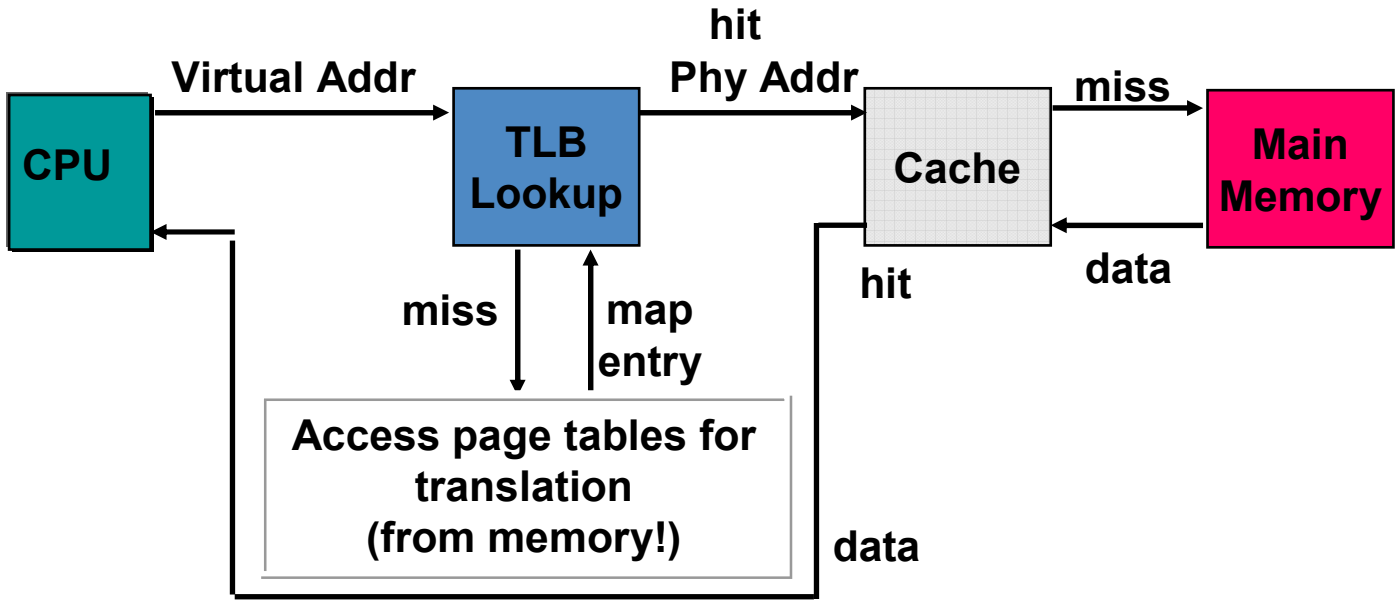
- "Internal" Fragmentation resulting from fixed size pages since not all of page will be filled with data
  - Problem gets worse as page size is increased
- Each data access now takes two memory accesses
  - First memory reference uses the page table base register to lookup the frame
  - Second memory access to actually fetch the value
- Page tables are large
  - Consider the case where the machine offers a 32 bit address space and uses 4 KB pages
    - Page table contains $2^{32} / 2^{10} = 2^{22}$ entries
    - Assume each entry contains ~32 bits
    - Page table requires 4 MB of RAM per process!

Review:TLB Entries

- Just holds a cached Page Table Entry (PTE)
- Additional bits for LRU, etc. may be added

Virtual Page	Physical Frame	Dirty	LRU	Valid	Access Rights

- Optimization is to access TLB and cache in parallel

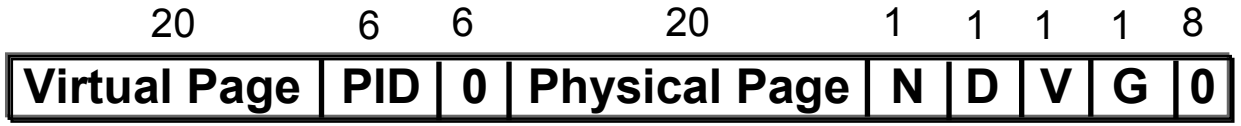


Review: TLB Miss Handler

- A TLB miss can be handled by software or hardware
  - Miss rate : 0.01% – 1%
  - Page tables are stored in regular physical memory
  - It is therefore likely that the data is in the L2 cache
- In software, a special OS handler looks up the value in the page table
  - Seven to ten instructions on R3000/R4000
- In hardware, microcode or a dedicated finite state machine (FSM) looks up the value
  - Advantages
    - No need to take an exception
    - Better performance but may be dominated by cache miss

TLB Case Study: MIPS R2000/R3000

- Consider the MIPS R2000/R3000 processors
  - Addresses are 32 bits with 4 KB pages (12 bit offset)
  - TLB has 64 entries, fully associative
  - Each entry is 64 bits wide:



PID	Process ID
N	Do not cache memory address
D	Dirty bit
V	Valid bit
G	Global (valid regardless of PID)

MIPS UTLB Miss Handler in Software

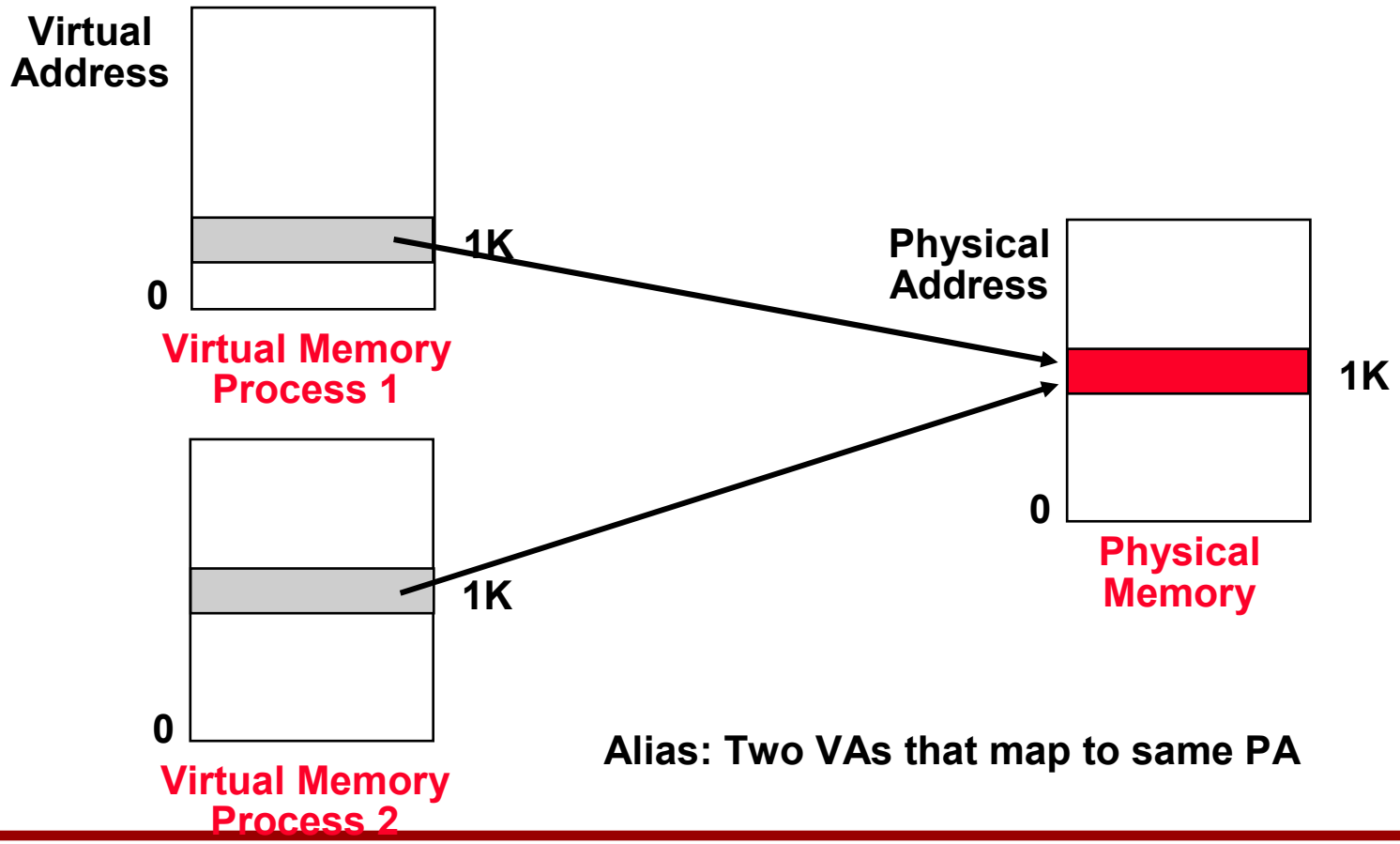
- In software, a special OS handler looks up the value in the page table

```
UTLBmiss:
  mfc0   $k1, Context # copy address of PTE to $k1
  lw     $k1, 0($k1)  # put PTE into $k1
  mtc0   $k1, Entrylo # put PTE into  CP0 Entrylo
  tlbwr               # put Entryhi, Entrylo into
                      # TLB at Random
  eret                # return from UTLB miss
```

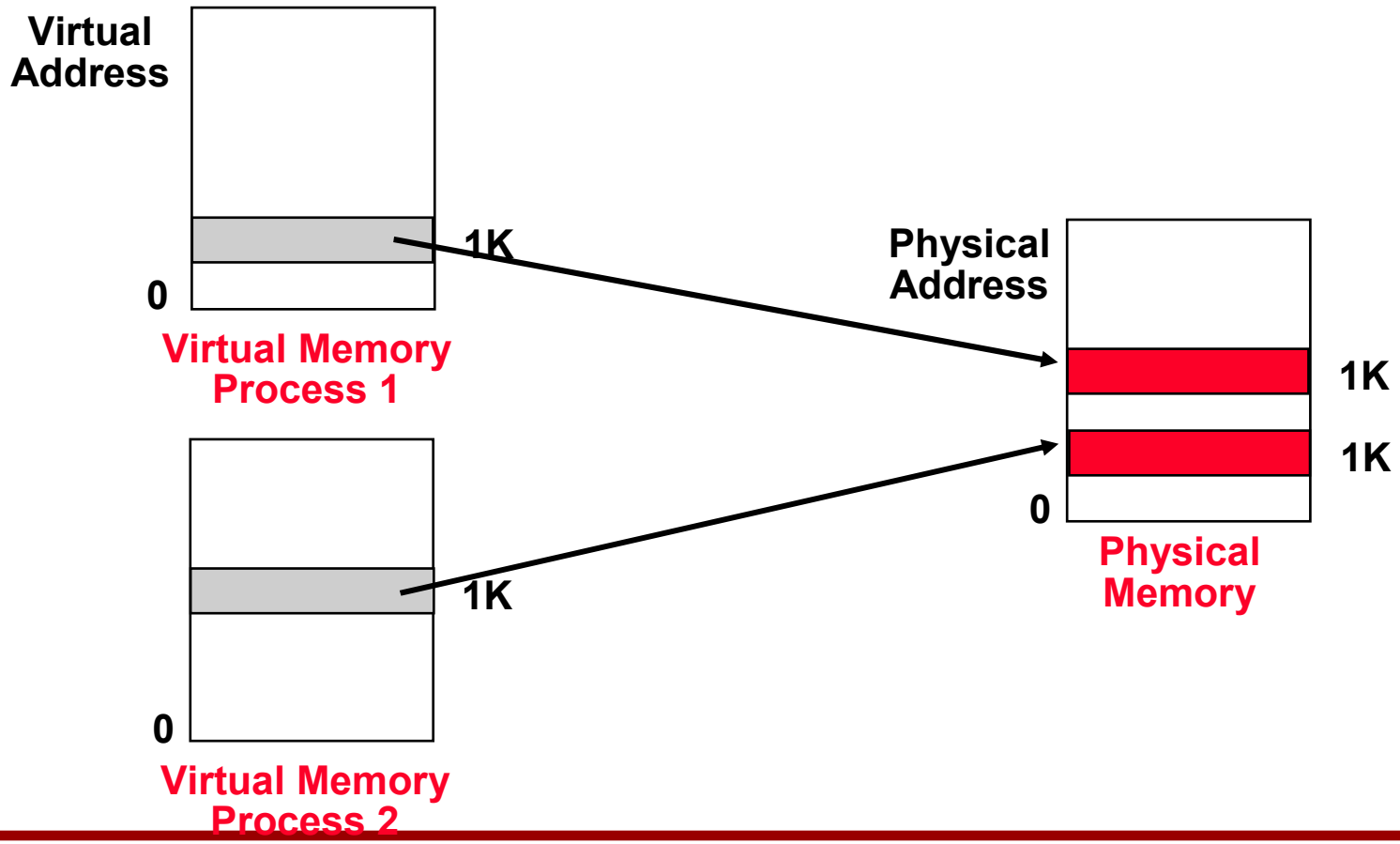
Page Sharing

- Another benefit of paging is that we can easily share pages by mapping multiple pages to the same physical frame
- Useful in many different circumstances
  - Useful for applications that need to share data
  - Read only data for applications, OS, etc.
  - Example: Unix `fork()` system call creates second process with a "copy" of the current address space
- Often implemented using *copy-on-write*
  - Processes have read-only access to the page
  - If any process wants to write, then a page fault occurs and the page is copied

Page Sharing (cont)



Copy-on-Write



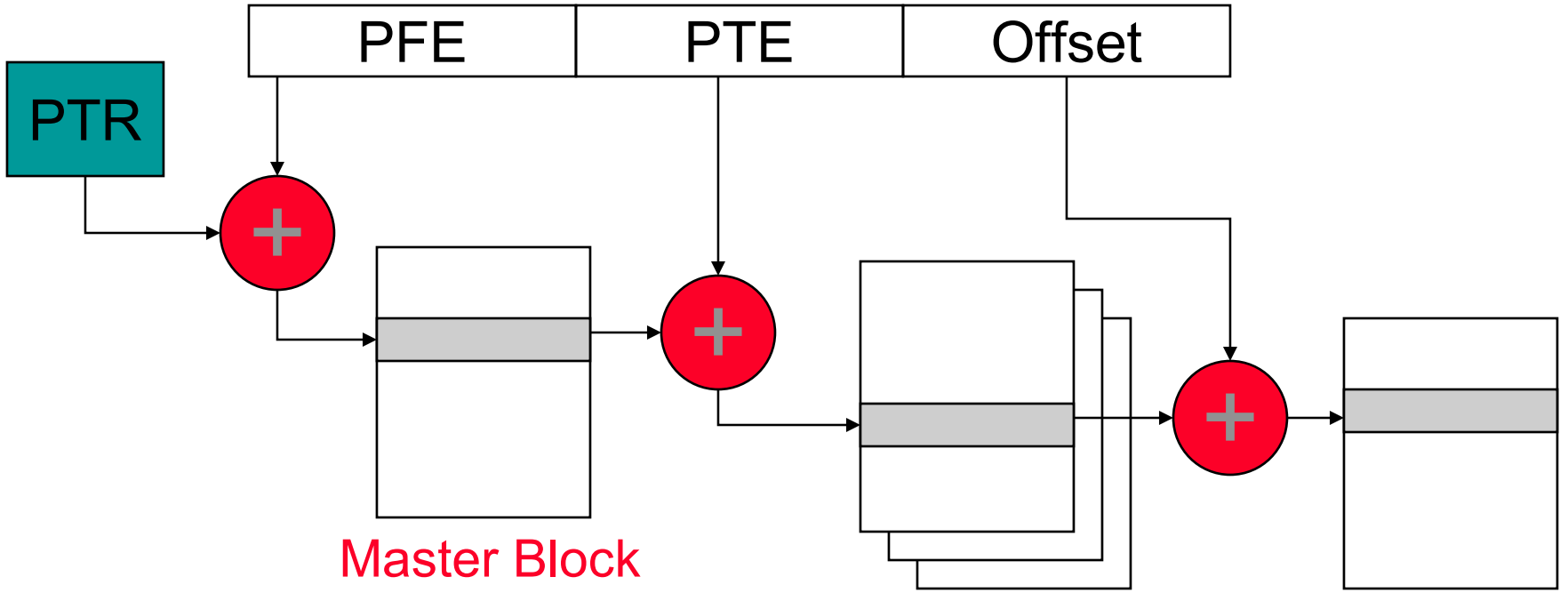
Page Table Size

- Page table size is proportional to size of address space
  - Even when applications use few pages, page table is still very large
- Example: Intel 80x86 Page Tables
  - Virtual addresses are 32 bits
  - Page size is 4 KB
  - Total number of pages: $2^{32} / 2^{12} = 1$ Million
  - Page Table Entry (PTE) are 32 bits wide
    - 20 bit Frame address
    - Dirty bit, accessed bit, valid (present) bit
    - 9 flag and unused bits
  - Total page table size is therefore $2^{20} \times 4$ bytes = 4 MB
  - But, only a small fraction of those pages are actually used!
- The large page table size is made even worse by the fact that the page table must be in memory
  - Otherwise, what would happen if the page table were suddenly swapped out?

Paging the Page Tables

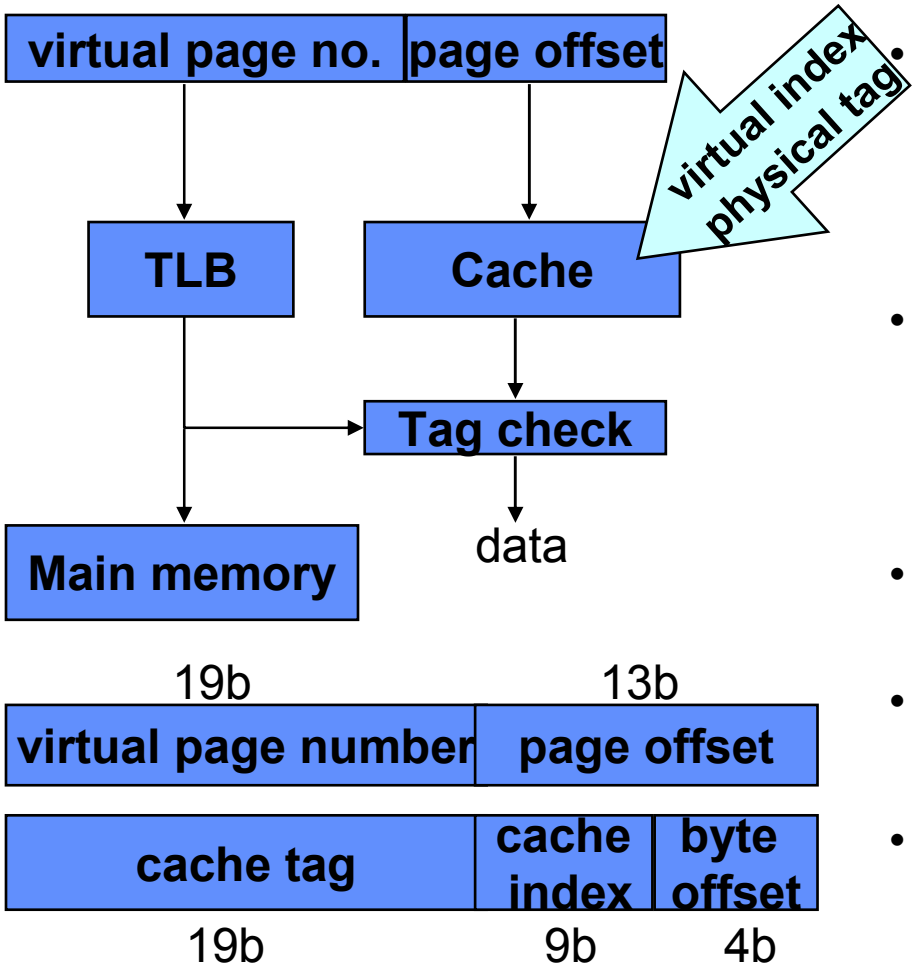
- One solution is to use multi-level page tables
  - If each PTE is 4 bytes, then each 4 KB frame can hold 1024 PTEs
  - Use an additional “master block” 4 KB frame to index frames containing PTEs
    - Master frame has 1024 PFEs (Page Frame Entries)
    - Each PFE points to a frame containing 1 KB PTEs
    - Each PTE points to a page
  - Now, only the “master block” must be fixed in memory

Two-level Paging



- Problems
  - –Multiple page faults
    - •Accessing a PTE page table can cause a page fault
    - •Accessing the actual page can cause a second page fault
  - –TLB plays an even more important role

Virtual Memory and Caches: TLB and Cache in Parallel



- Translation & cache access in 1 cycle
  - Access cache with untranslated part of address
  - Tag check used physical address
- Cache data is based on physical addresses
  - No problems with aliasing, I/O, multiprocessor snooping
- But only works when VPN bits not needed for cache lookup
- Cache Size <= Page Size * Assoc
  - i.e. Set Size <= Page Size
- Most common solution
  - We want L1 to be small anyway

Virtual Memory and Caches

TLB	Cache	Virtual memory	Possible?
hit	miss	hit	
miss	hit	hit	
miss	miss	hit	
miss	miss	miss	
hit	miss	miss	
hit	hit	miss	
miss	hit	miss	

Virtual Memory Overview

- Presents the illusion of a large address space to every application
  - Each process or program believes it has its own complete copy of physical memory (its own copy of the address space)
- Strategy
  - Pages not currently in use can be written back to secondary storage
  - When page ownership changes, save the current contents to disk so they can be restored later
- Costs
  - Address translation is still required to convert virtual addresses into physical addresses
  - Writing to the disk is a slow operation
  - TLB misses can limit performance

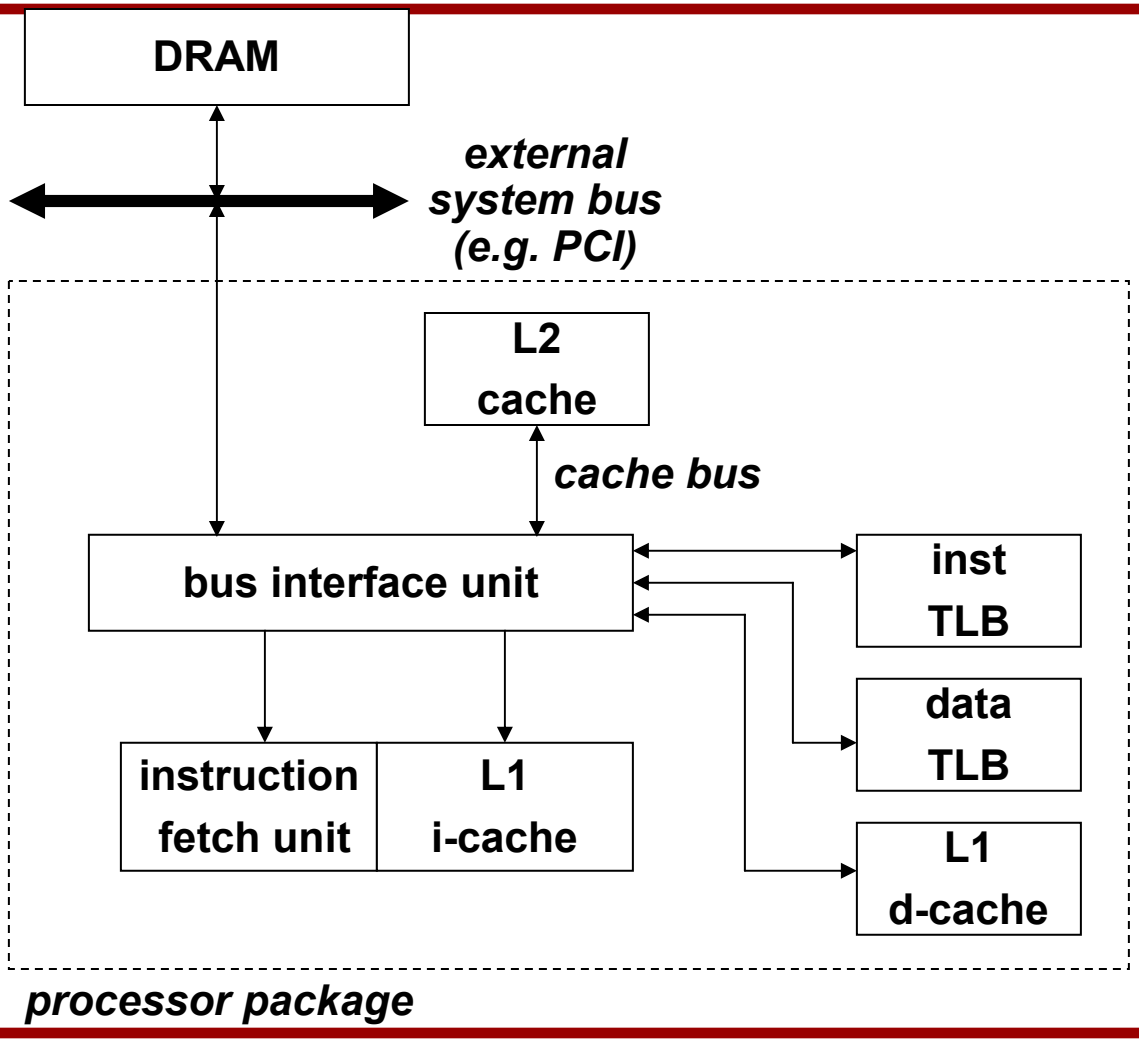
Real Stuff: Intel P6

- Internal Designation for Successor to Pentium
  - Which had internal designation P5
- Fundamentally Different from Pentium
  - Out-of-order, superscalar operation
  - Designed to handle server applications
    - Requires high performance memory system
- Resulting Processors
  - PentiumPro 200 MHz (1996)
  - Pentium II (1997)
    - Incorporated MMX instructions
    - L2 cache on same chip
  - Pentium III (1999)
    - Incorporated Streaming SIMD Extensions
  - Pentium M 1.6 GHz (2003)
    - Low power for mobile

Adapted from Computer Systems: APP

Bryant and O'Halloraon

P6 memory system

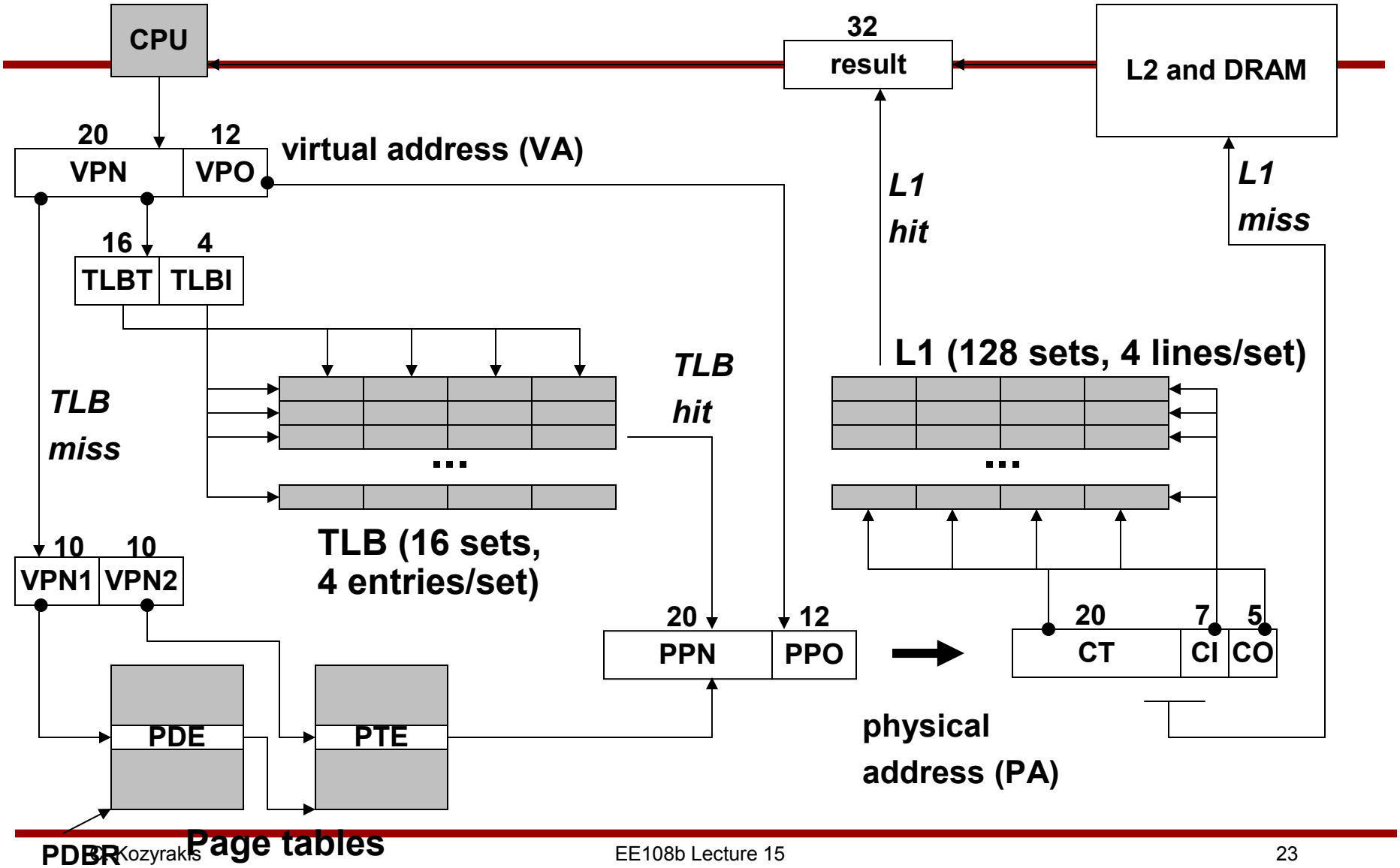


- 32 bit address space
- 4 KB page size
- L1, L2, and TLBs
  - 4-way set associative
- inst TLB
  - 32 entries
  - 8 sets
- data TLB
  - 64 entries
  - 16 sets
- L1 i-cache and d-cache
  - 16 KB
  - 32 B line size
  - 128 sets
- L2 cache
  - 128 KB – 2 MB

Review of abbreviations

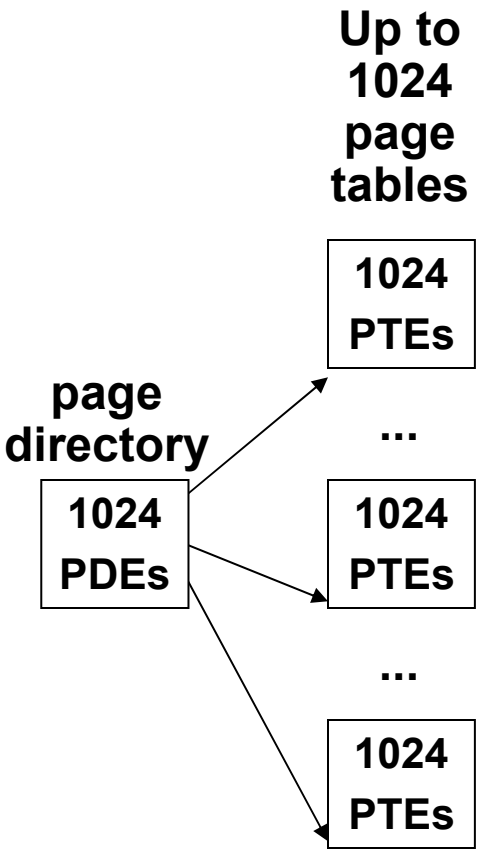
- Symbols:
  - Components of the virtual address (VA)
    - TLBI: TLB index
    - TLBT: TLB tag
    - VPO: virtual page offset
    - VPN: virtual page number
  - Components of the physical address (PA)
    - PPO: physical page offset (same as VPO)
    - PPN: physical page number
    - CO: byte offset within cache line
    - CI: cache index
    - CT: cache tag

Overview of P6 address translation

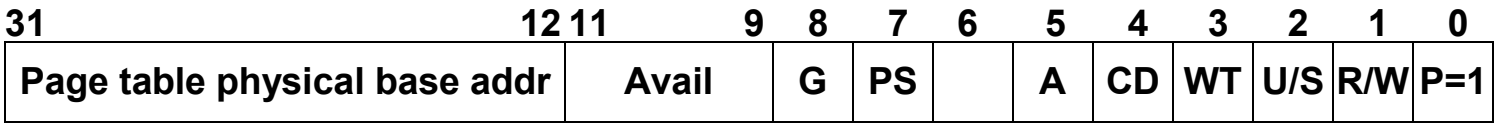


P6 2-level page table structure

- Page directory
  - 1024 4-byte page directory entries (PDEs) that point to page tables
  - One page directory per process.
  - Page directory must be in memory when its process is running
  - Always pointed to by PDBR
- Page tables:
  - 1024 4-byte page table entries (PTEs) that point to pages.
  - Page tables can be paged in and out.



P6 page directory entry (PDE)



Page table physical base address: 20 most significant bits of physical page table address (forces page tables to be 4KB aligned)

Avail: available for system programmers

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

PS: page size 4K (0) or 4M (1)

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

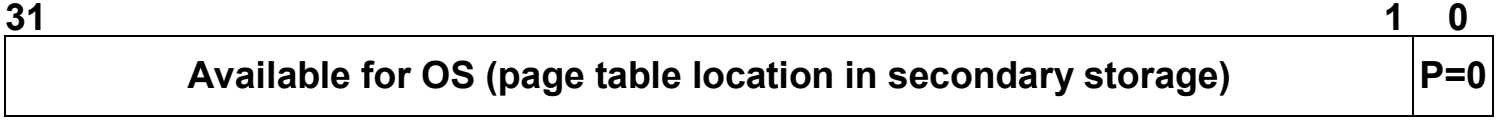
CD: cache disabled (1) or enabled (0)

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

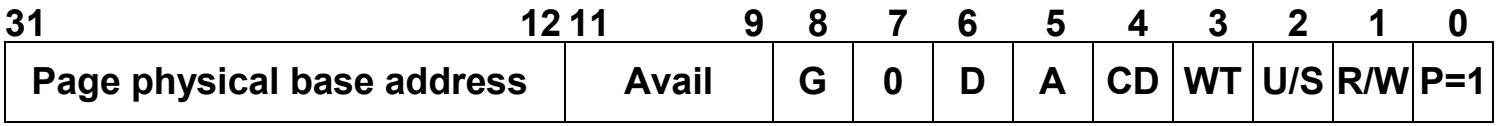
U/S: user or supervisor mode access

R/W: read-only or read-write access

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



P6 page table entry (PTE)



Page base address: 20 most significant bits of physical page address (forces pages to be 4 KB aligned)

Avail: available for system programmers

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

D: dirty (set by MMU on writes)

A: accessed (set by MMU on reads and writes)

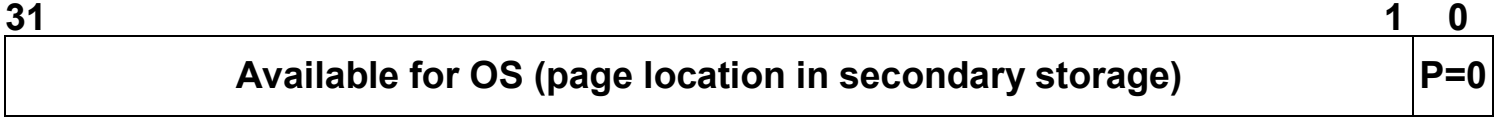
CD: cache disabled or enabled

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

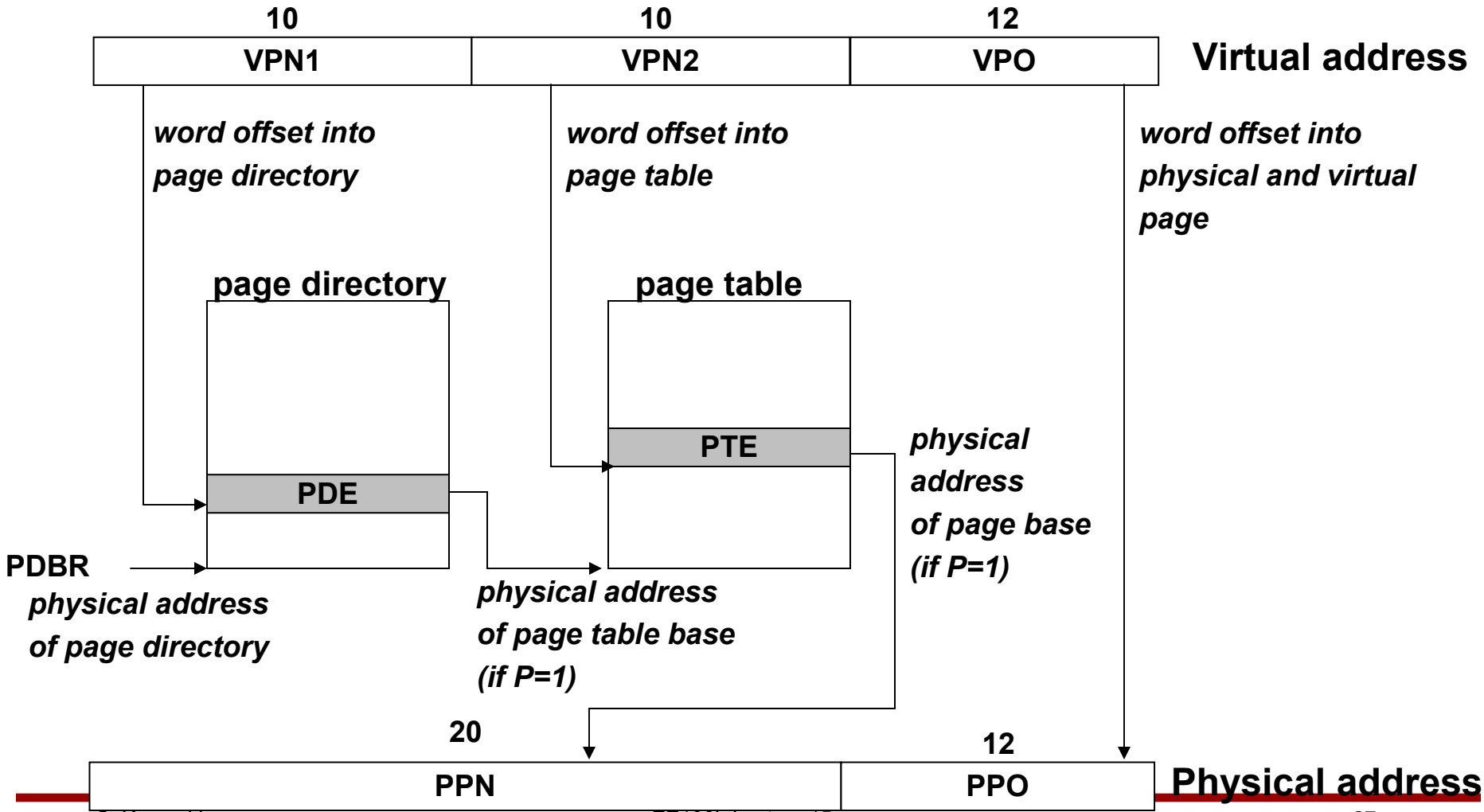
U/S: user/supervisor

R/W: read/write

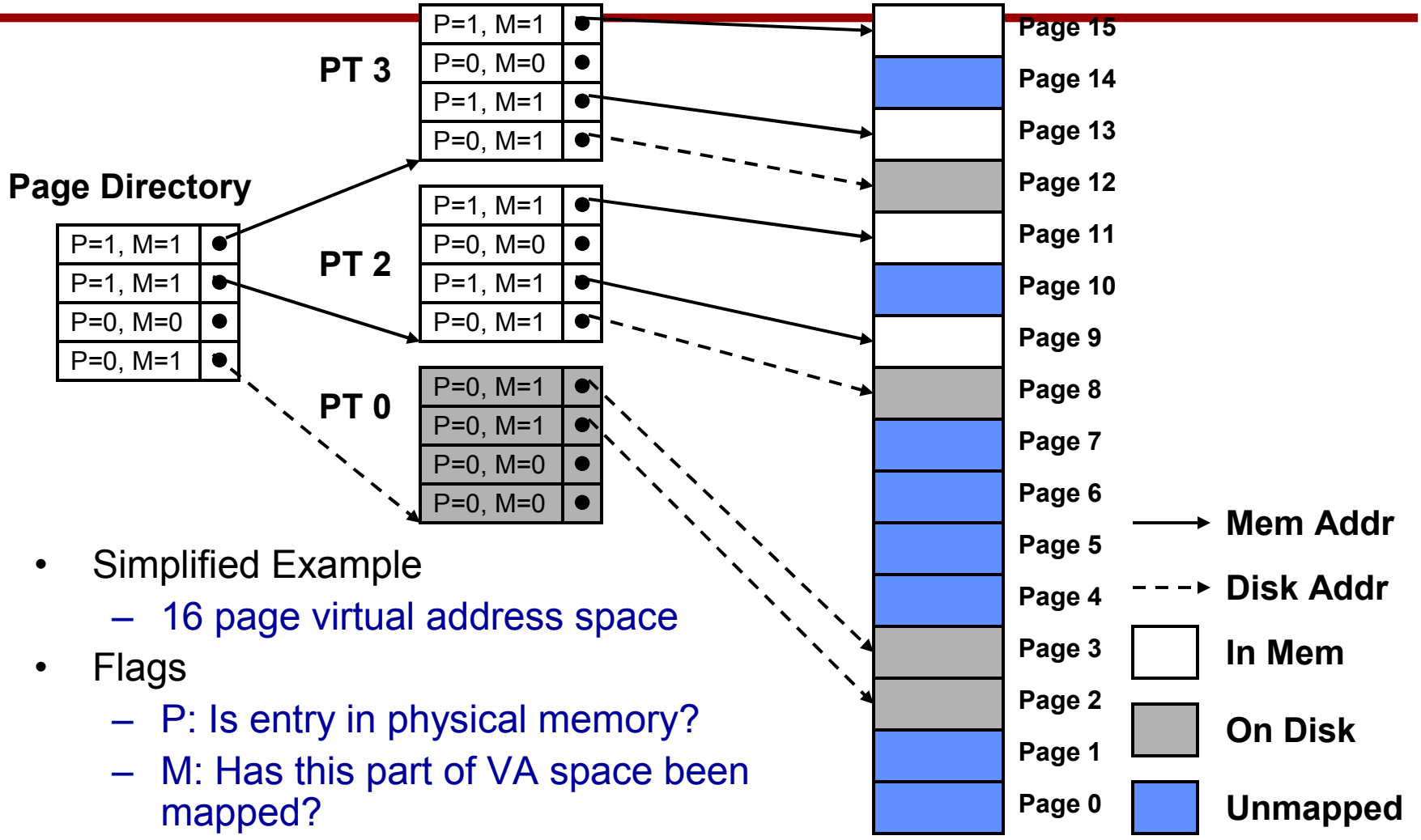
P: page is present in physical memory (1) or not (0)



How P6 page tables map virtual addresses to physical ones

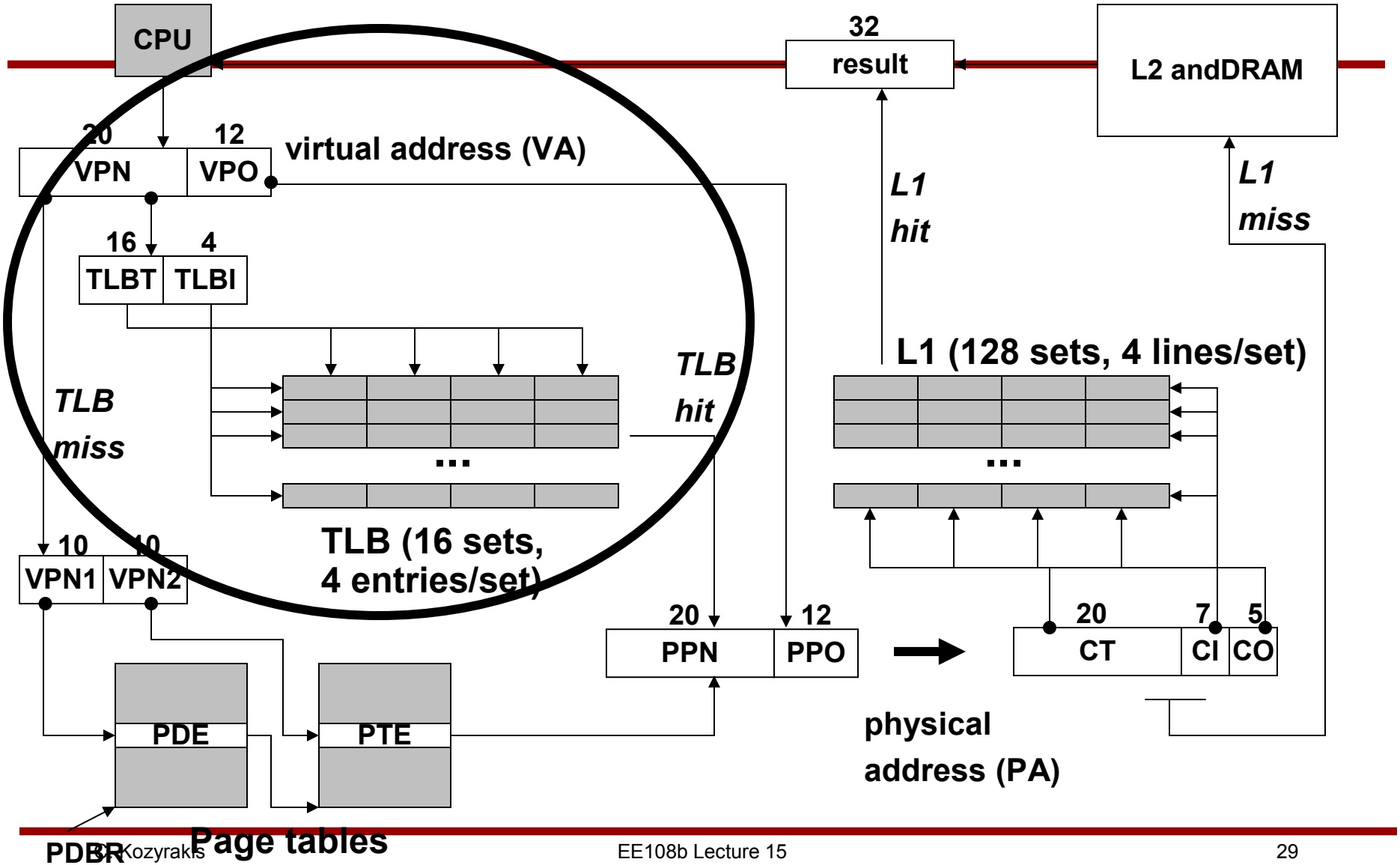


Representation of Virtual Address Space



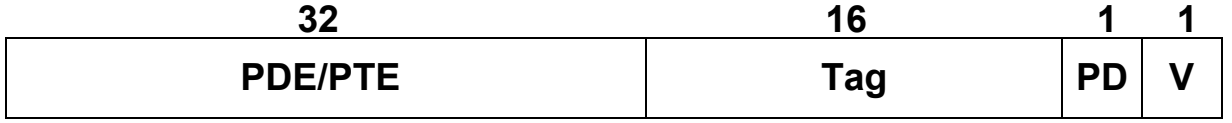
- Simplified Example
  - 16 page virtual address space
- Flags
  - P: Is entry in physical memory?
  - M: Has this part of VA space been mapped?

P6 TLB translation

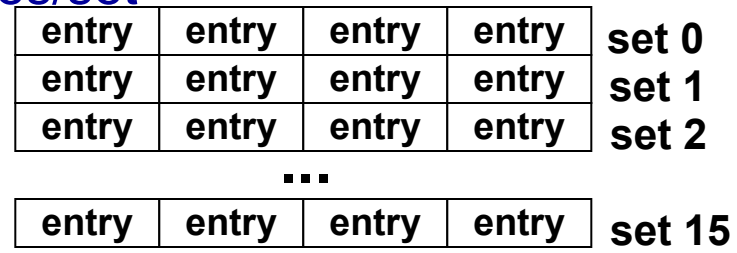


P6 TLB

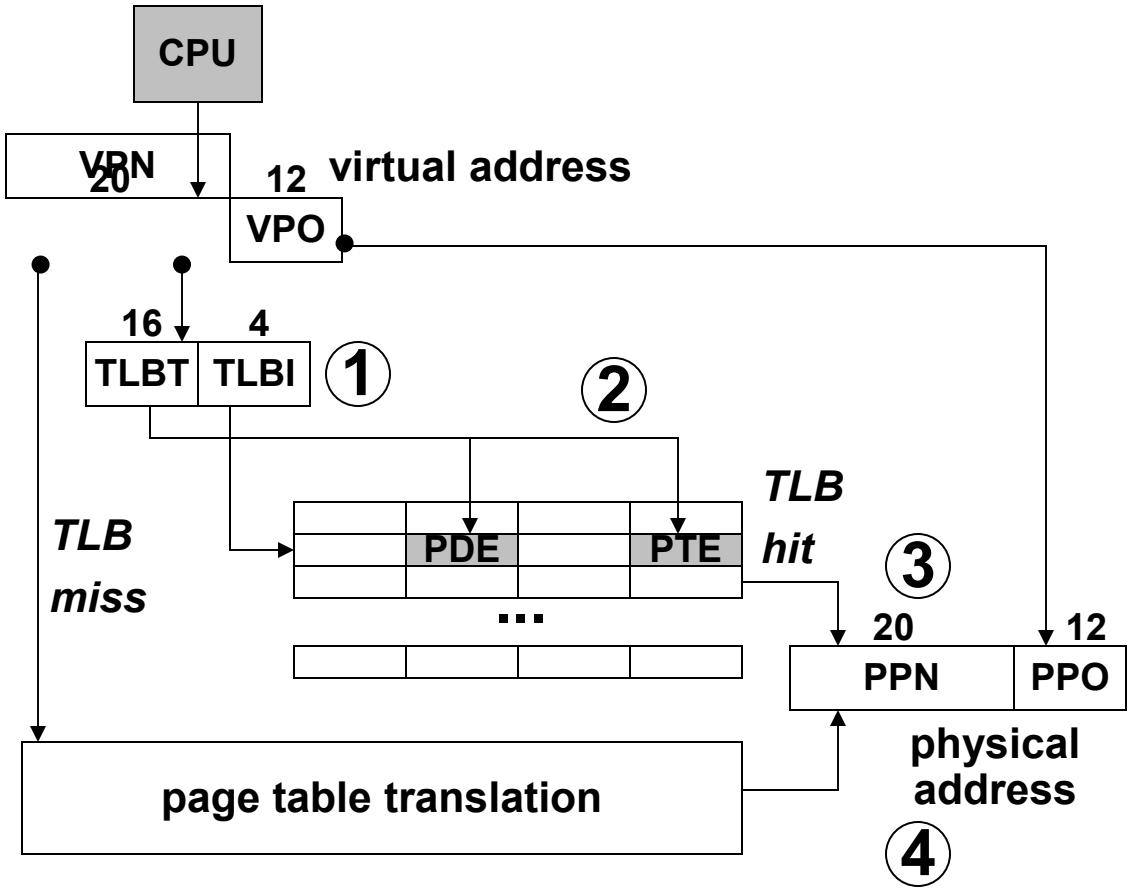
- TLB entry (not all documented, so this is speculative):



  - V: indicates a valid (1) or invalid (0) TLB entry
  - PD: is this entry a PDE (1) or a PTE (0)?
  - tag: disambiguates entries cached in the same set
  - PDE/PTE: page directory or page table entry
- Structure of the data TLB:
  - 16 sets, 4 entries/set

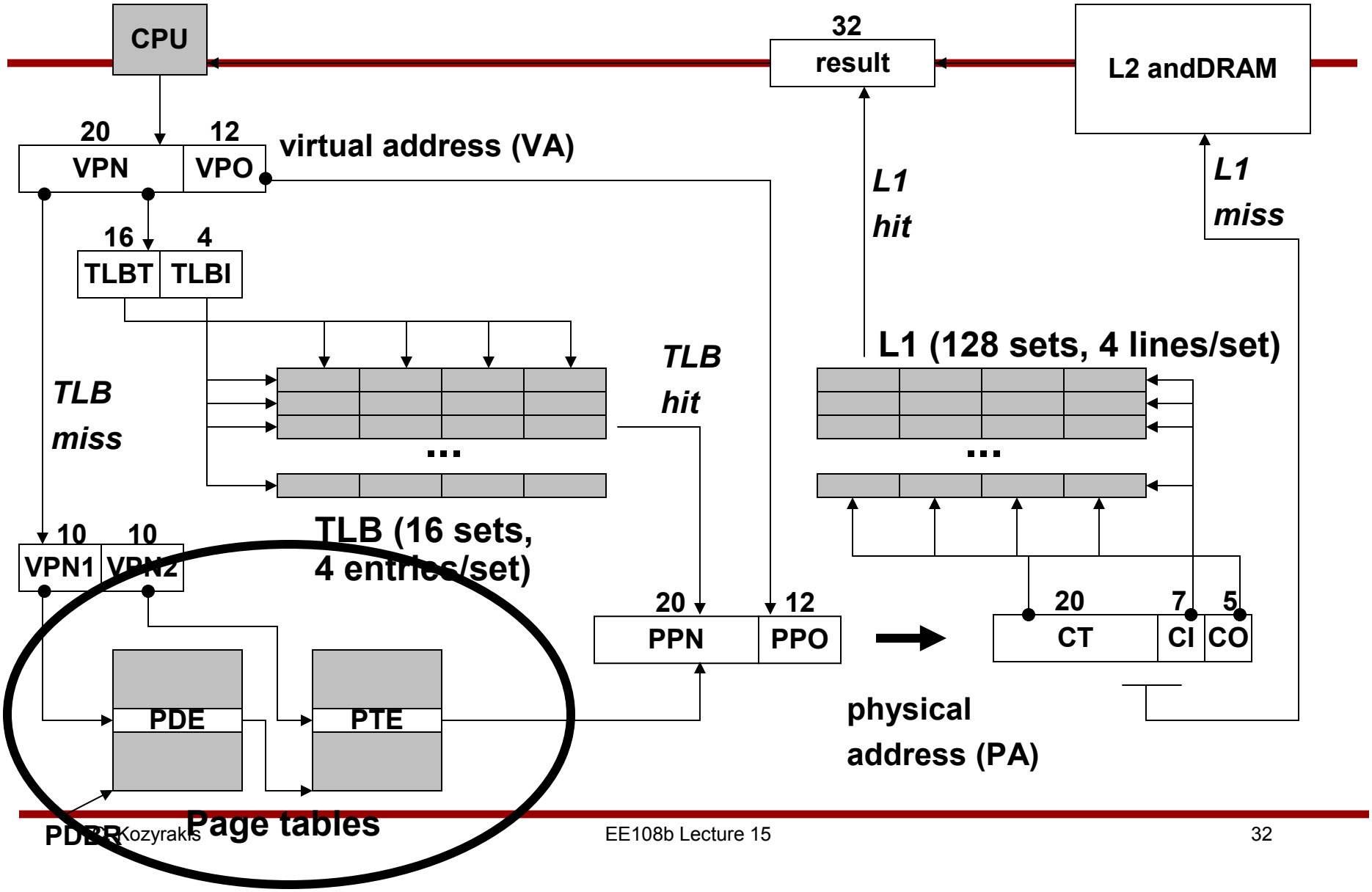


Translating with the P6 TLB

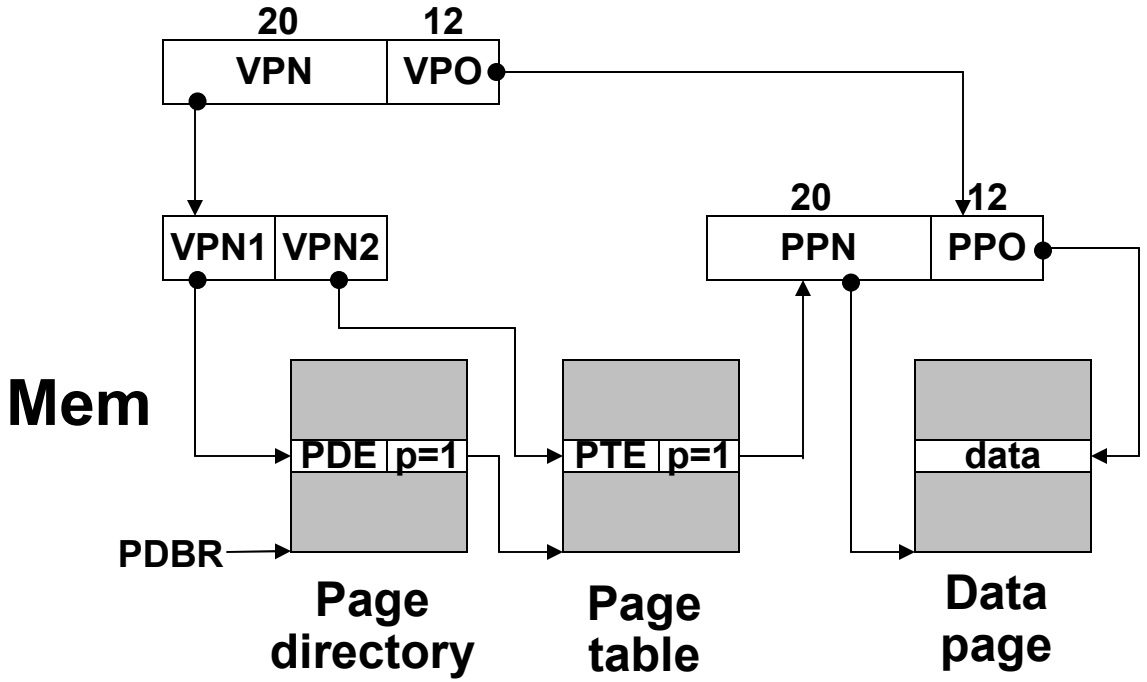


- 1. Partition VPN into TLBT and TLBI.
- 2. Is the PTE for VPN cached in set TLBI?
  - 3. Yes: then build physical address.
- 4. No: then read PTE (and PDE if not cached) from memory and build physical address.

P6 page table translation

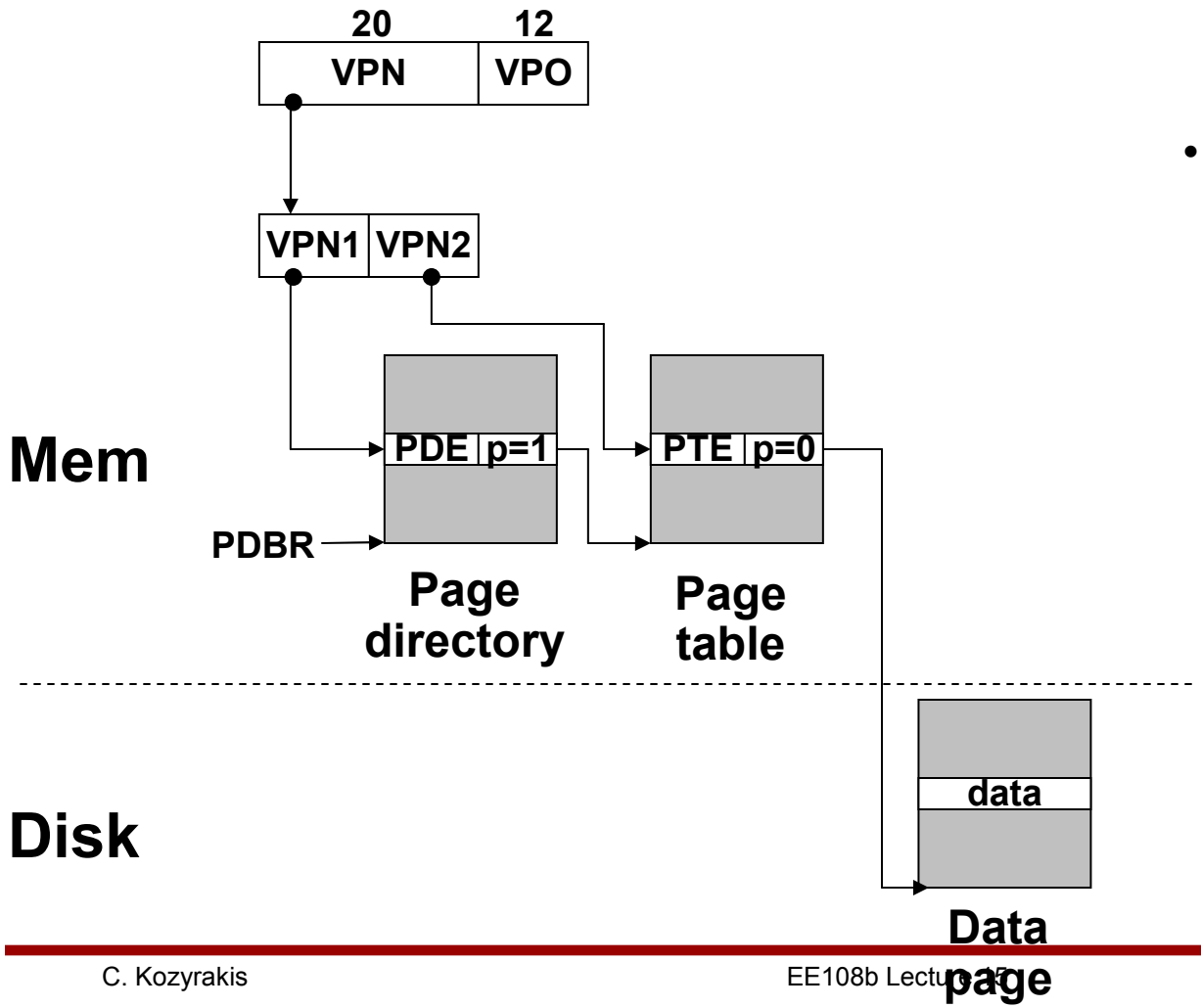


Translating with the P6 page tables (case 1/1)



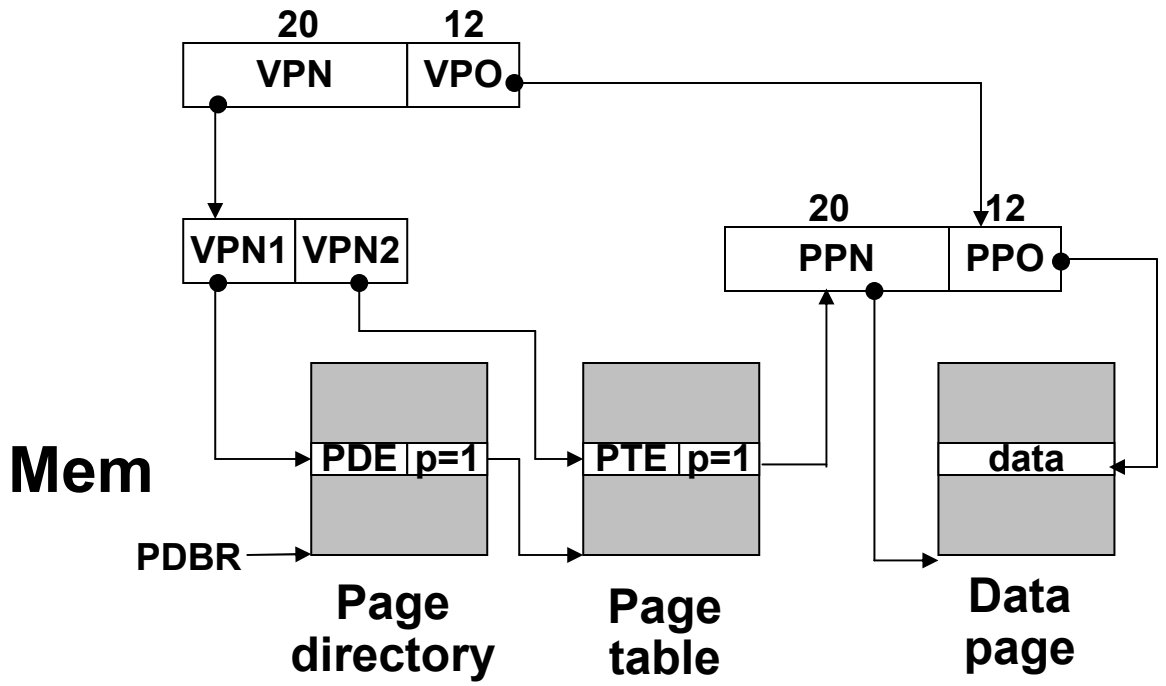
- Case 1/1: page table and page present.
- MMU Action:
  - MMU build physical address and fetch data word.
- OS action
  - none

Translating with the P6 page tables (case 1/0)



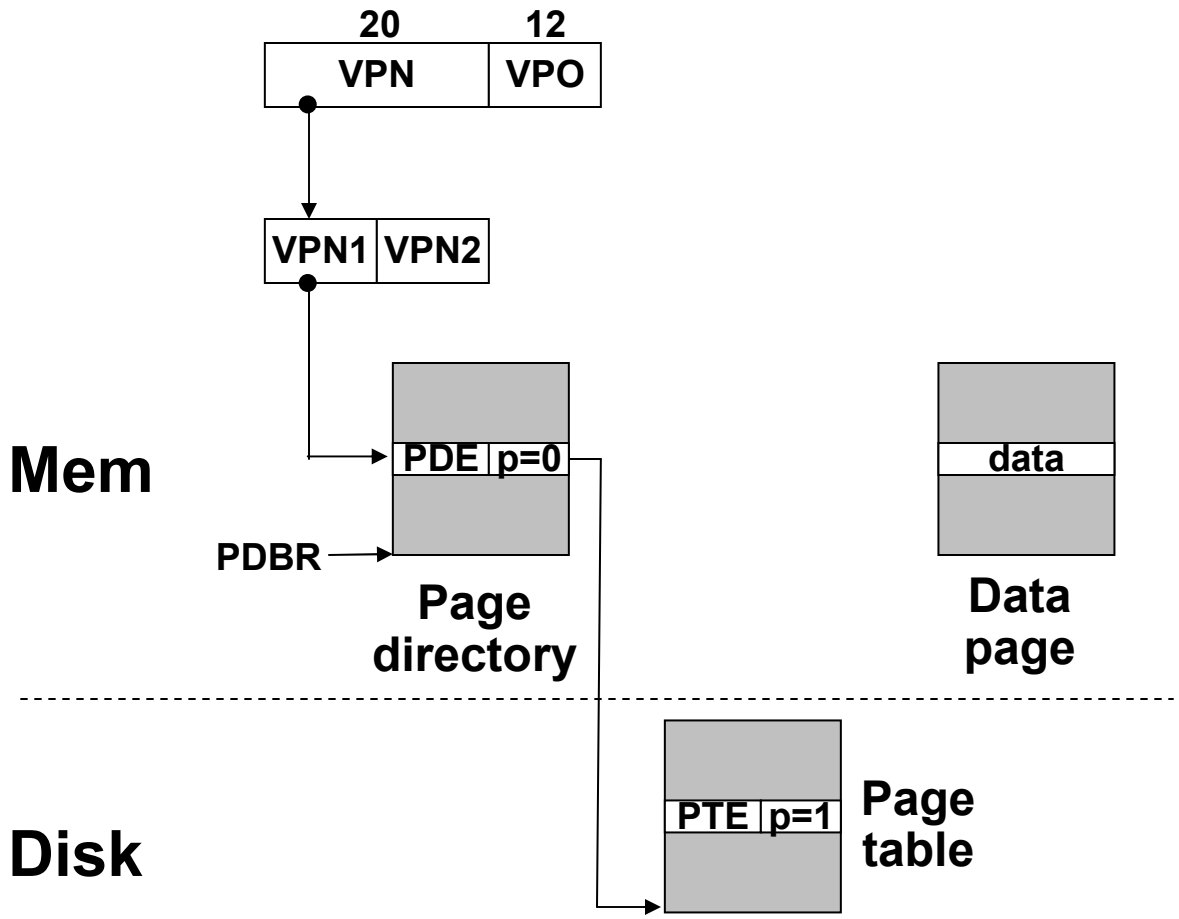
- Case 1/0: page table present but page missing.
- MMU Action:
  - page fault exception
  - handler receives the following args:
    - VA that caused fault
    - fault caused by non-present page or page-level protection violation
    - read/write
    - user/supervisor

Translating with the P6 page tables (case 1/0, cont)



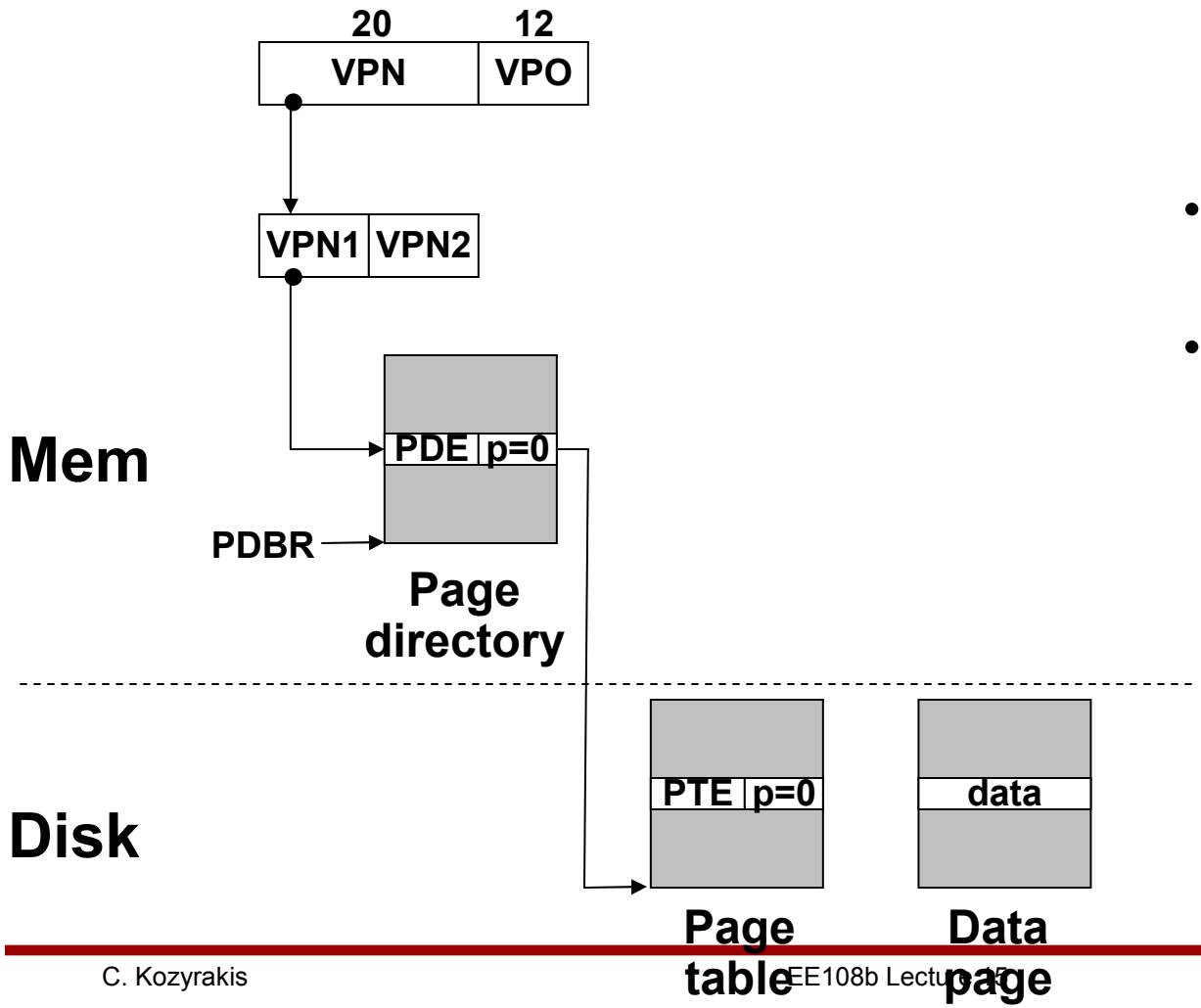
- OS Action:
  - Check for a legal virtual address.
  - Read PTE through PDE.
  - Find free physical page (swapping out current page if necessary)
  - Read virtual page from disk and copy to virtual page
  - Restart faulting instruction by returning from exception handler.

Translating with the P6 page tables (case 0/1)



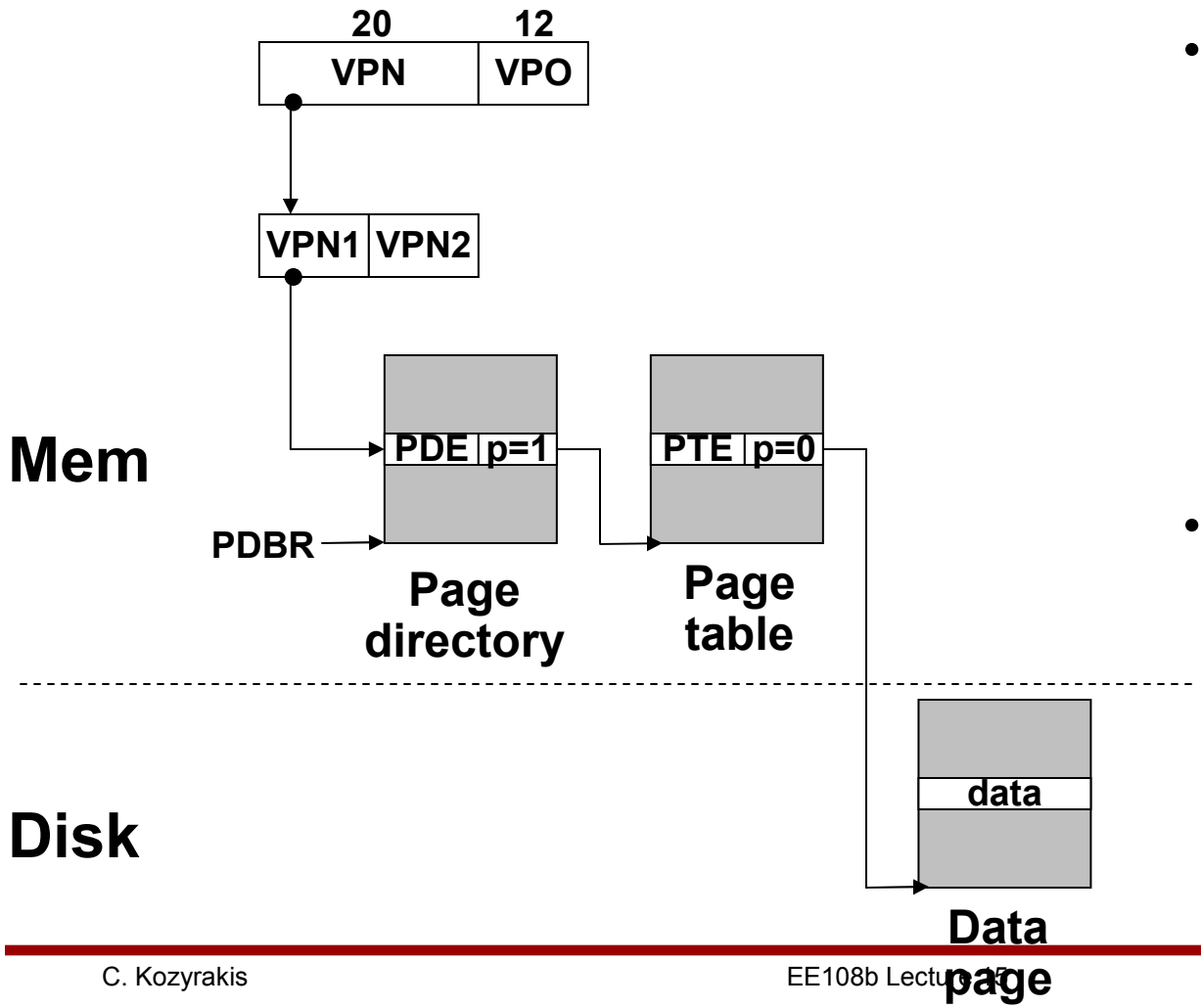
- Case 0/1: page table missing but page present.
- Introduces consistency issue.
  - potentially every page out requires update of disk page table.
- Linux disallows this
  - if a page table is swapped out, then swap out its data pages too.

Translating with the P6 page tables (case 0/0)



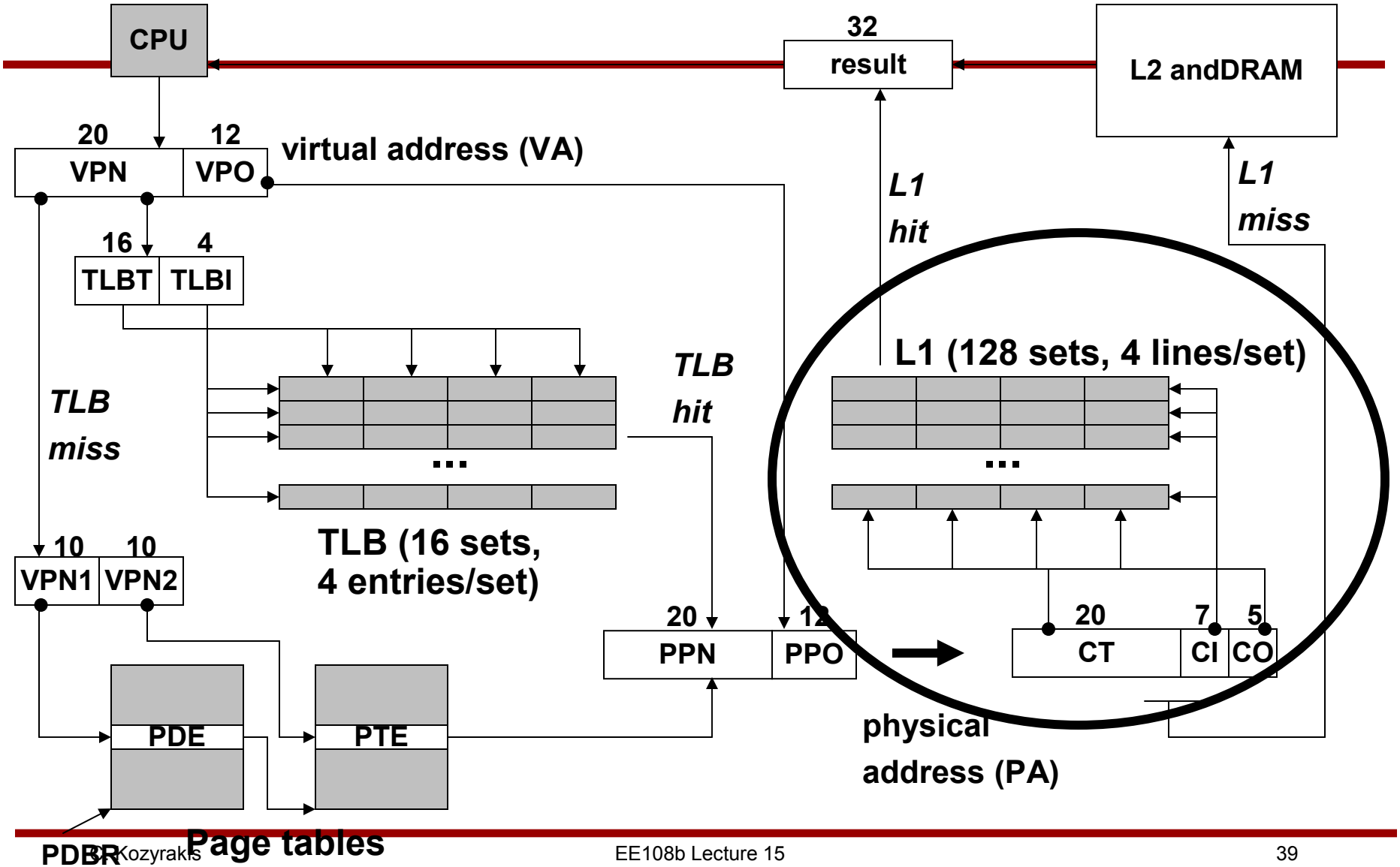
- Case 0/0: page table and page missing.
- MMU Action:
  - page fault exception

Translating with the P6 page tables (case 0/0, cont)

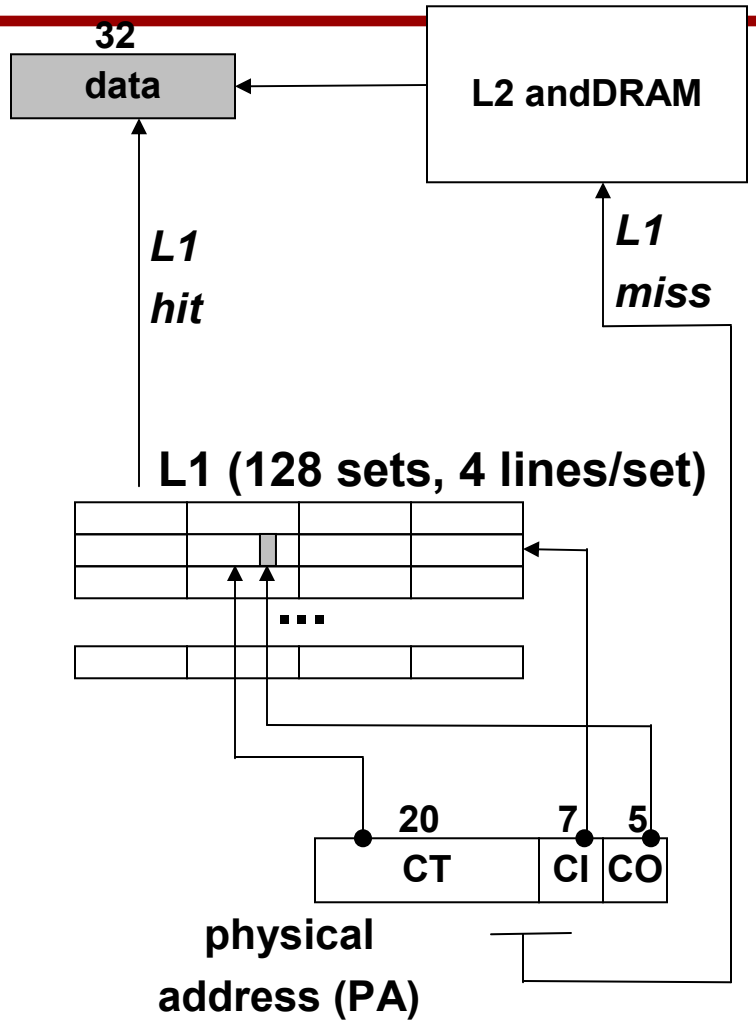


- OS action:
  - swap in page table.
  - restart faulting instruction by returning from handler.
- Like case 0/1 from here on.

P6 L1 cache access

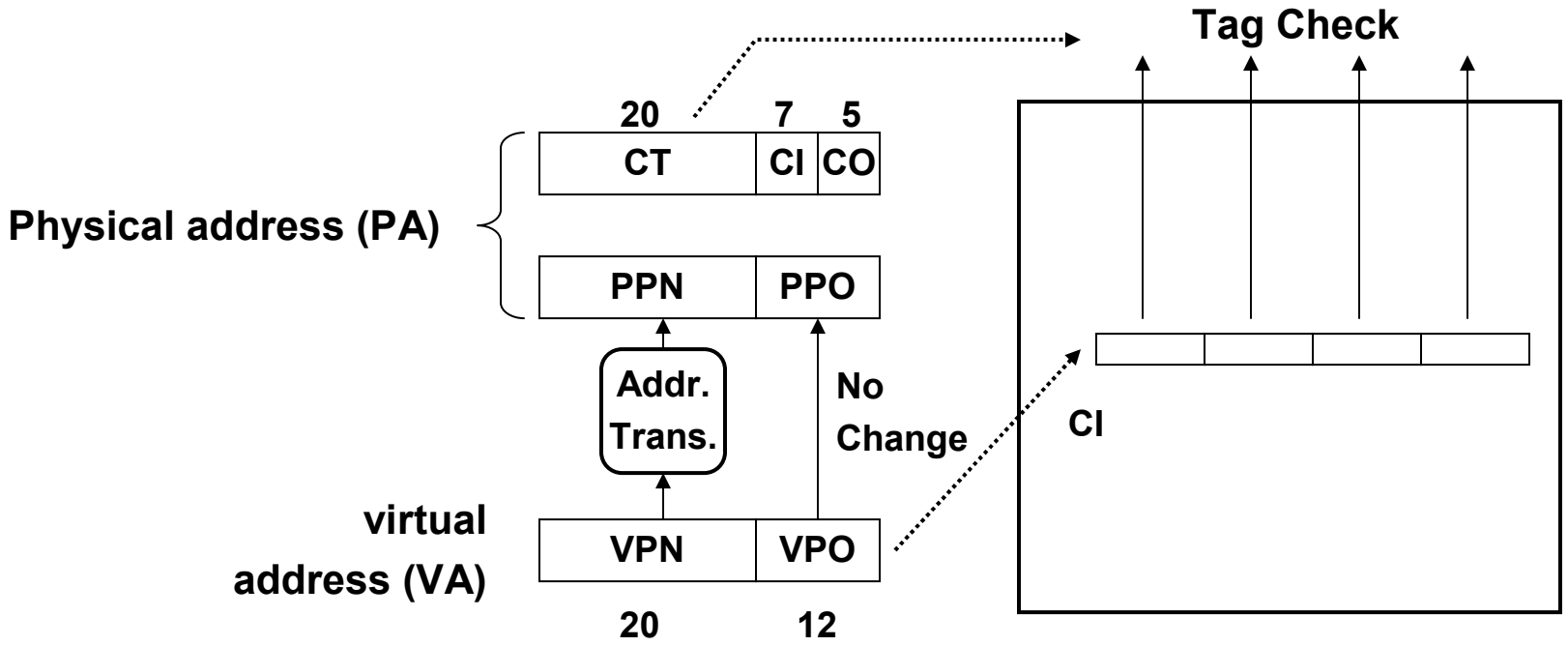


L1 cache access



- Partition physical address into CO, CI, and CT.
- Use CT to determine if line containing word at address PA is cached in set CI.
- If no: check L2.
- If yes: extract word at byte offset CO and return to processor.

Speeding Up L1 Access



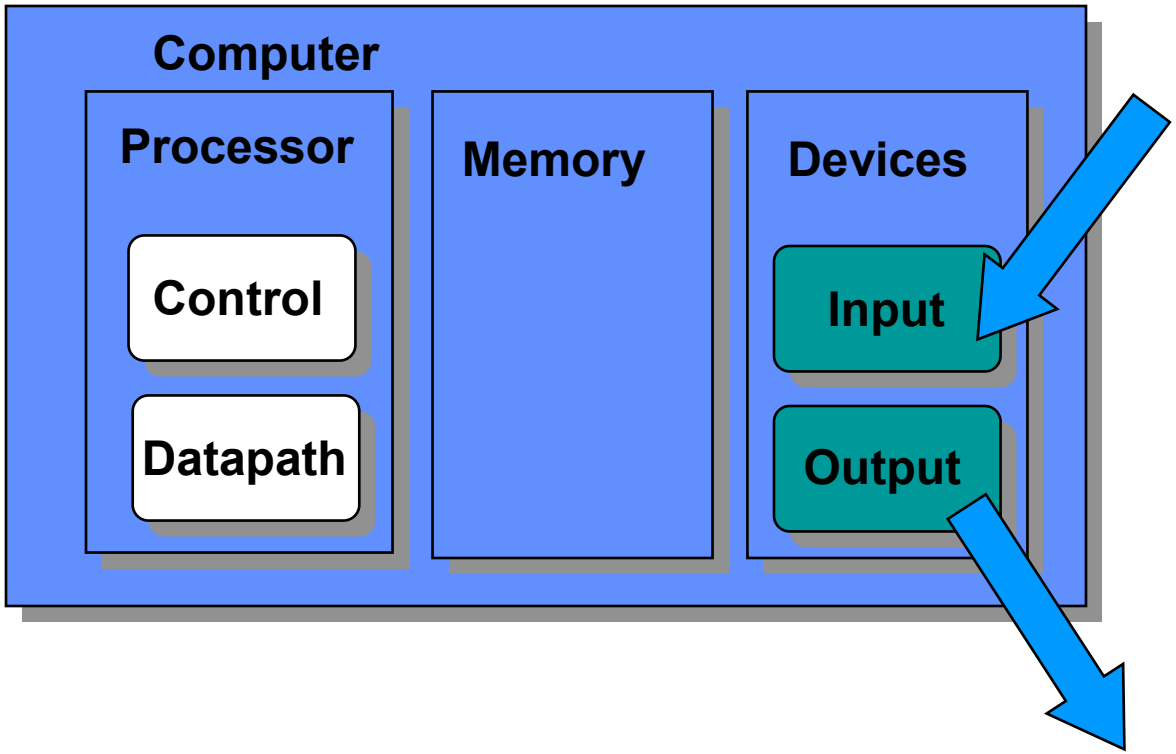
- Observation
  - Bits that determine CI identical in virtual and physical address
  - Can index into cache while address translation being performed
  - Then check with CT from physical address
  - “Virtually indexed, physically tagged”

Review: Memory Hierarchy Framework

- Placing a Block
  - Direct mapped – Element hashed to a single location
  - Set Associative – Element can go in one of N locations
  - Fully Associative – Element can go anywhere
- Finding a Block
  - Index with partial map (cache) or full map (page table)
  - Index and search (set associative)
  - Search (fully associative)
- Replacing a Block
  - Random – Pick one of the elements to replace
  - Least Recently Used (LRU) – Use bits to track usage
- Writing
  - Write back – Data is written only on eviction
  - Write through – Each write is passed through to lower level

Locality makes this work

Five Components



- Datapath
- Control
- Memory
- Input
- Output

Outline

- I/O Systems and Performance
  - Types and characteristics of I/O devices
  - Magnetic disks
- Buses
  - Bus types and bus operation
  - Bus arbitration
- Interfacing the OS and I/O devices
  - Operating System's role in handling I/O devices
  - Delegating I/O responsibility by the CPU
- I/O workloads and performance

Today's Lecture

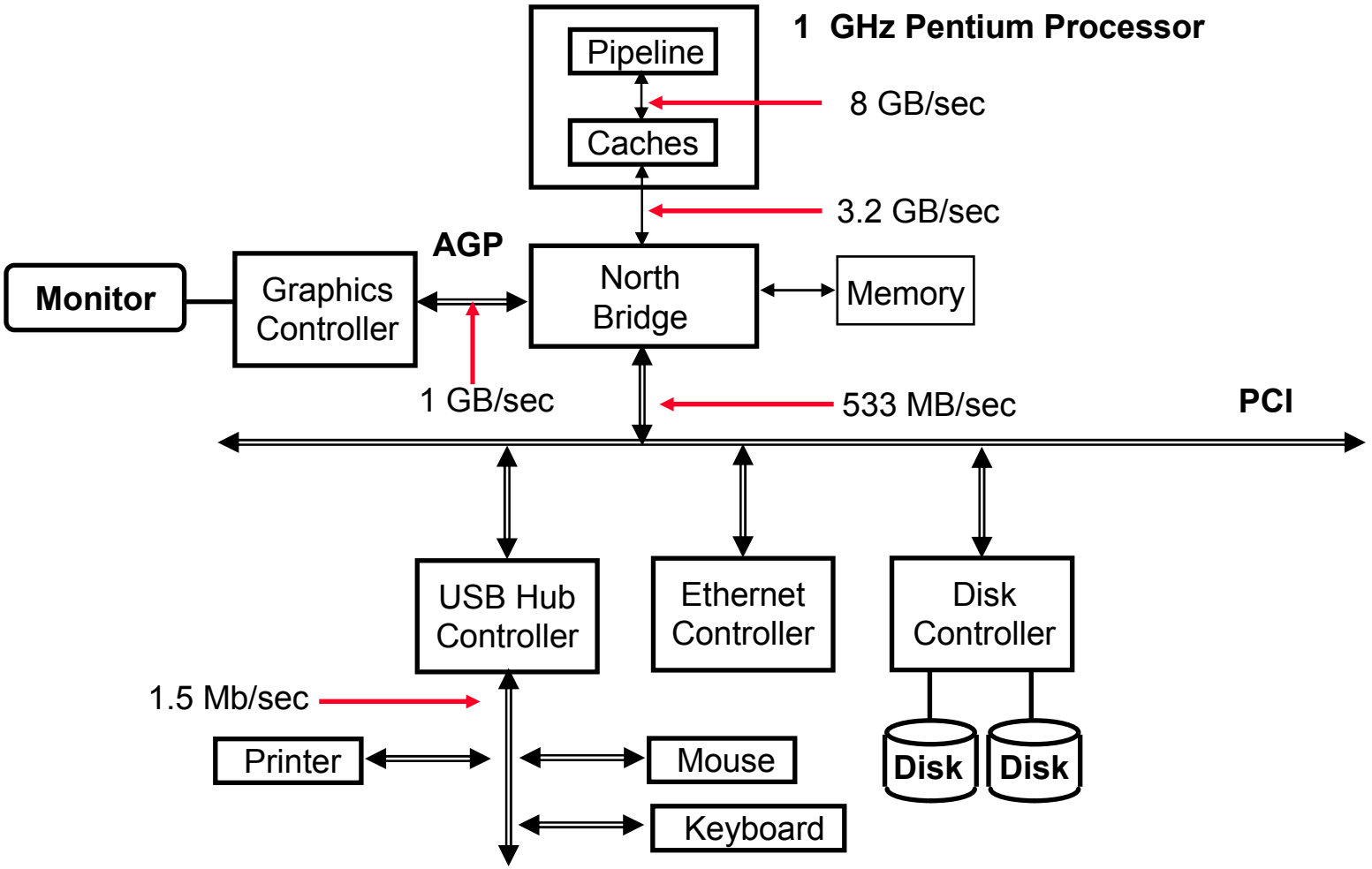
- I/O overview
- I/O performance metrics
- High performance I/O devices
  - Disk

Diversity of Devices

Device	Behavior	Partner	Data Rate (KB/sec)
Keyboard	Input	Human	0.01
Mouse	Input	Human	0.02
Line Printer	Output	Human	1.00
Laser Printer	Output	Human	100.00
Graphics	Output	Human	100,000.00
Network-LAN	Communication	Machine	10,000.00
Floppy disk	Storage	Machine	50.00
Optical Disk	Storage	Machine	10, 000.00
Magnetic Disk	Storage	Machine	30,000.00

- Behavior refers to what I/O device does
- Since I/O connects two things, partner refers to the object on the other end of the connection

Speeds and Feeds of a PC System



I/O System Design Issues

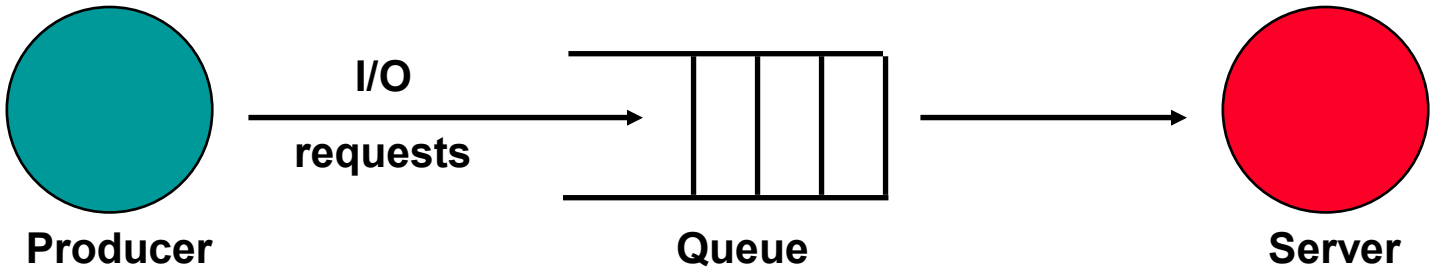
- Performance
  - Is throughput or response time more critical?
  - Huge diversity of devices means wide performance spectrum
  - I/O device performance tends to be technology driven
  - I/O system performance also depends on OS, software, bus performance, etc
- Expandability
- Resilience in the face of failure

Throughput vs. Response time

- Throughput
  - Aggregate measure of amount of data moved per unit time, averaged over a window
  - Sometimes referred to as bandwidth
    - Example: Memory bandwidth
    - Example: Disk bandwidth

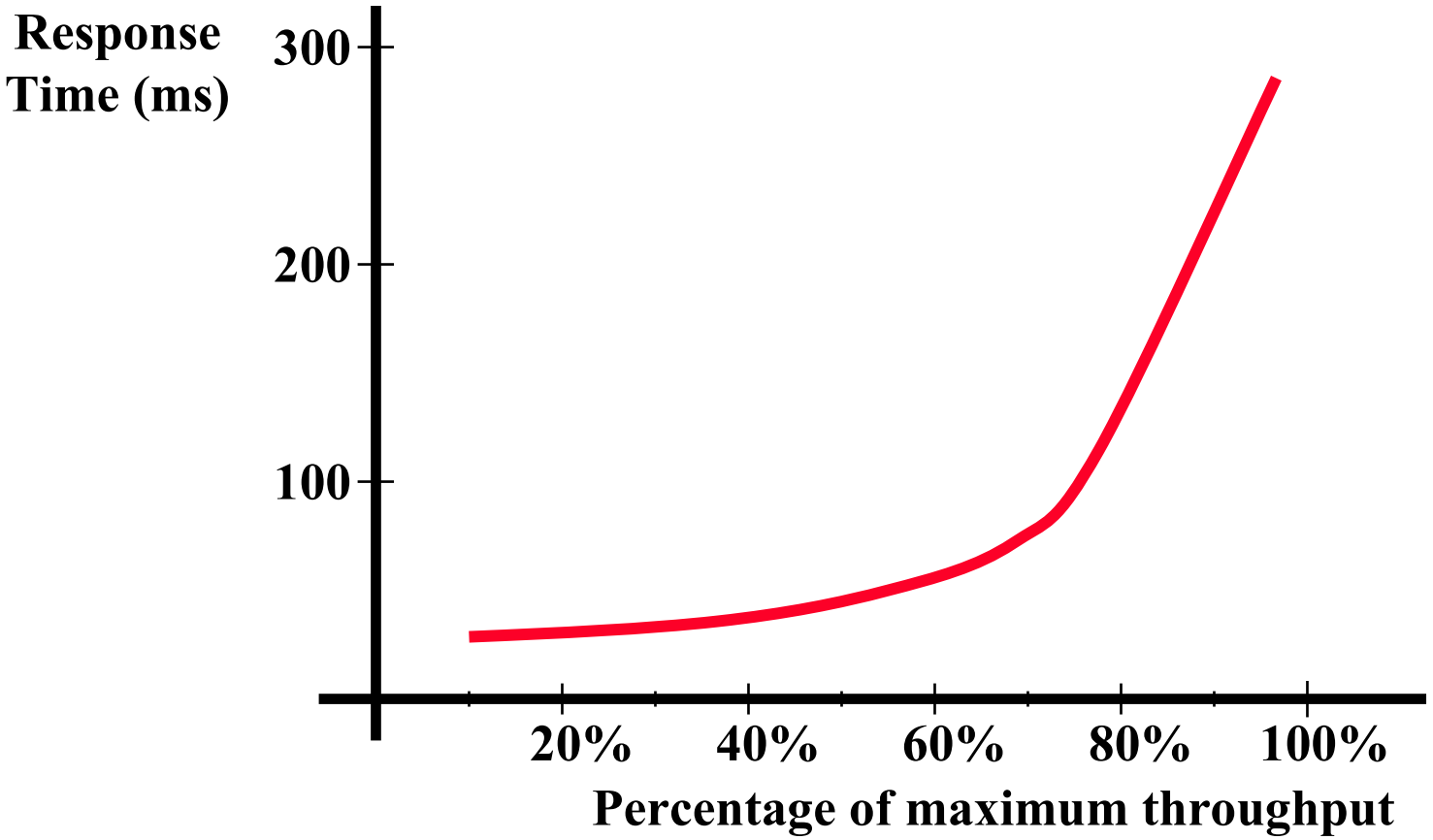
- Response time
  - Response time to do a single I/O operation
    - Example: Write a block of bytes to disk
    - Example: Send a data packet over the network

Producer-Server Model



- Throughput is the number of tasks completed by the server per unit time
- Response time (latency) is the elapsed time between tasks entering queue and tasks completed by the server
- Tradeoff between throughput and response time
  - Highest possible throughput is when the server is always busy and the queue is never empty
  - Fastest response time is when the queue is empty and the server is idle when the task arrives

Throughput vs. Response Time



I/O Devices

- I/O devices leverage various implementation techniques
  - Magnetic disks