UC Berkeley
Teaching Professor
Dan Garcia

CS61C

Great Ideas in Computer Architecture

(a.k.a. Machine Structures)

UC Berkeley
Teaching Professor
Lisa Yan

A **UC-wide strike of GSIs, readers, and other academic workers** has started. This is the outcome of a 98% positive vote of UAW academic worker union members to authorize a strike, in the absence of an agreement between UC and UAW over key issues including academic workers' salaries, benefits, job security, and support for working parents.

Virtual Memory III: Interrupts/Exceptions, Translation Lookaside Buffer

For more resources (and how it affects CS61C):
https://edstem.org/us/courses/25846/discussion/2141094

OS: Supervisor Mode, Exceptions [continued]

- OS: Supervisor Mode, Exceptions [continued]
- OS: Boot [recorded]
- Caches vs. Virtual Memory
- Translation Lookaside Buffer
- TLBs in the Datapath
- VM Performance

Traps Handle Exceptions/Interrupts

Review

- **The trap handler is code that services interrupts/exceptions.**

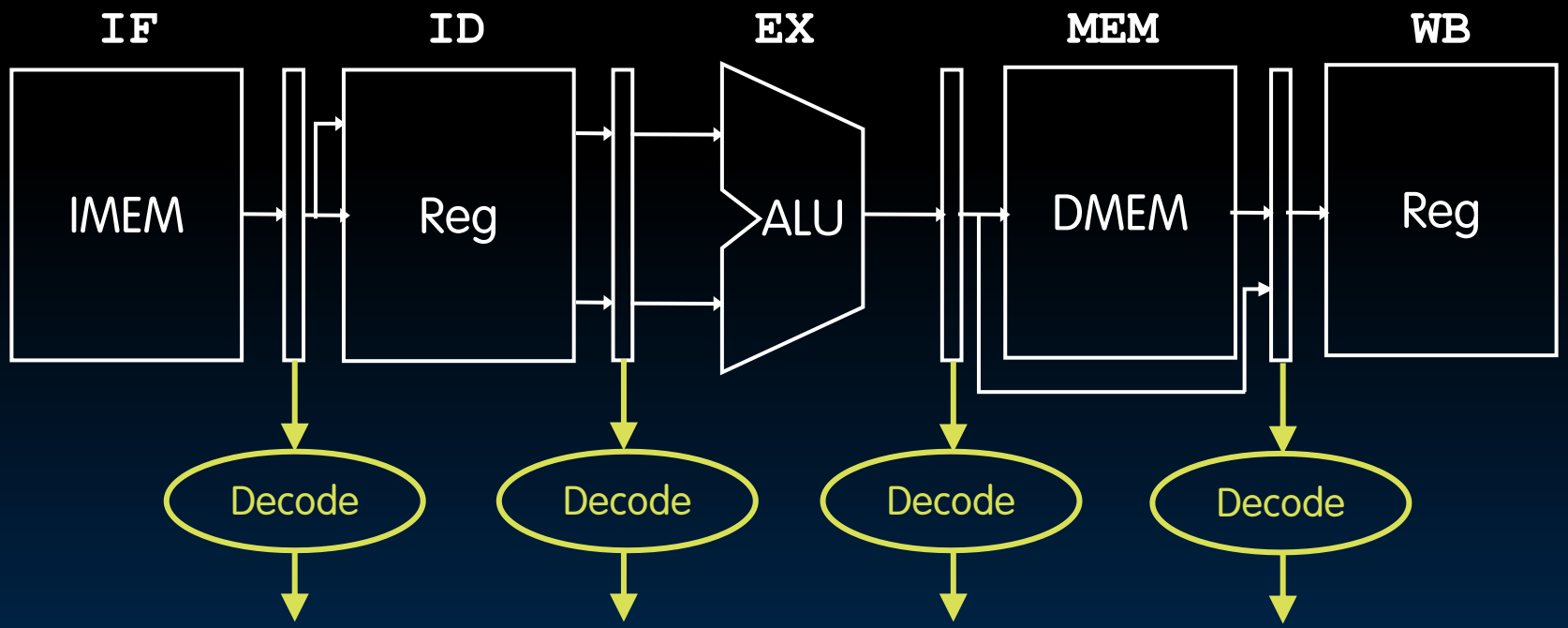
asynchronous, external — synchronous, during (e.g. page fault)

1. **Complete all instructions before the faulting instruction.**
2. **Flush all instructions after the faulting instruction.**
   - Like pipeline hazard: convert to noops/"bubbles."
   - Also flush faulting instruction.
3. **Transfer execution to trap handler (runs in supervisor mode).**
   - Optionally *return* to original program and re-execute instruction.

If the trap handler returns, then from the program's point of view it must look like nothing has happened!

Exceptions in a 5-Stage Pipeline

- Traps are handled similarly to pipeline hazards.
- In RISC-V, the exception cause can be inferred by the faulting instruction and its current pipeline stage.

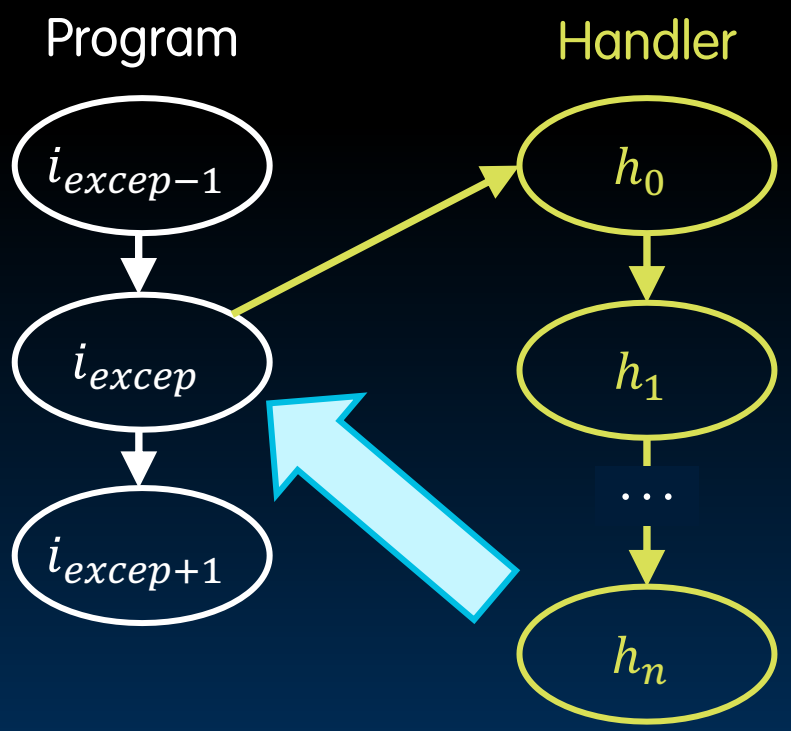


The Trap Handler

1. **Save the state of the current program.**
   - Save ALL of the registers!
2. **Determine what caused the exception/interrupt.**
3. **Handle exception/interrupt, then do one of two things:**

Continue execution of the program:

4. **Restore program state.**
5. **Return control to the program.**

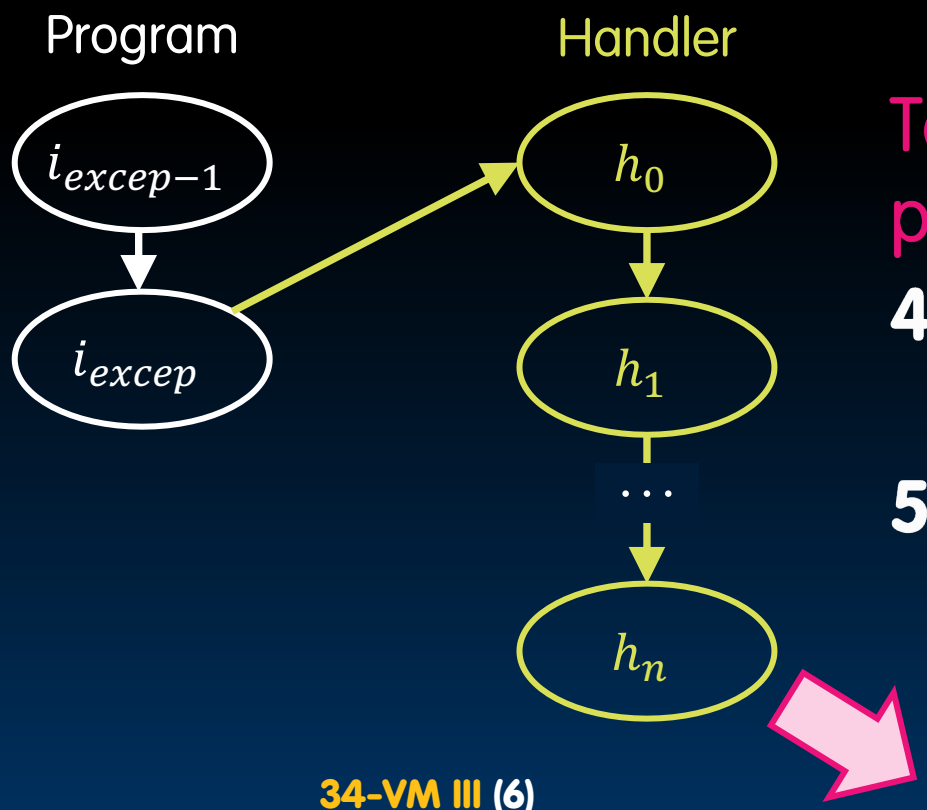


The Trap Handler

1. **Save the state of the current program.**
   - Save ALL of the registers!
2. **Determine what caused the exception/interrupt.**
3. **Handle exception/interrupt, then do one of two things:**

Continue execution of the program:

4. **Restore program state.**
5. **Return control to the program.**



Terminate the program:

4. **Free the program resources, etc.**
5. **Schedule a new program.**

Handling Context Switches

- **Recall the context switch:**
  - OS switches between processes (i.e., programs) by changing the internal state of the processor.
  - Allows a single processor to "simultaneously" run many programs.
- **At a high-level:**
  - The OS sets a timer. When it expires, perform a *hardware interrupt*.
  - Trap handler saves all register values, including:
    - Program Counter (PC)
    - *Page Table Register* (SPTBR in RV32I)
      - The memory *address* of the active process's page table.
  - Trap handler then loads in the next process's registers and returns to user mode.

Handling Page Faults

- **Recall** page faults:
  - An accessed page table entry has valid bit off → data is not in DRAM.
- **Page faults are handled by the trap handler.**
  - The *page fault exception handler* initiates transfers to/from disk and performs any page table updates.
  - (If pages needs to be swapped from disk, perform *context switch* so that another process can use the CPU in the meantime.)
    - (ideally need a "precise trap" so that resuming a process is easy.)
  - Following the page fault, *re-execute the instruction*.
- **Side note: Write protection violations also trigger exceptions.**

System Calls and Launching Applications

- **A system call (syscall) is a "software interrupt" that allows a program to request a service from the operating system.**
  - Similar to a function call, except now executed by kernel.
  - Examples:
    - Creating and deleting files; reading/writing files;
    - Accessing external devices (e.g., scanner);
    - `printf`, `malloc`, etc. (`ecall`s in RISC-V); etc.
    - Launch a new process
- **Suppose shell (a user process) wants to launch a new app:**
  - Shell *forks* (in Linux): a syscall that traps into the OS kernel process
  - OS (supervisor mode): Load program (see CALL); jump to start of `main`. Return to user mode.
  - Shell: "wait" for `main` to return (*join*)

Recorded if we don't get to it:
https://youtu.be/1mZ-ztcwbZk

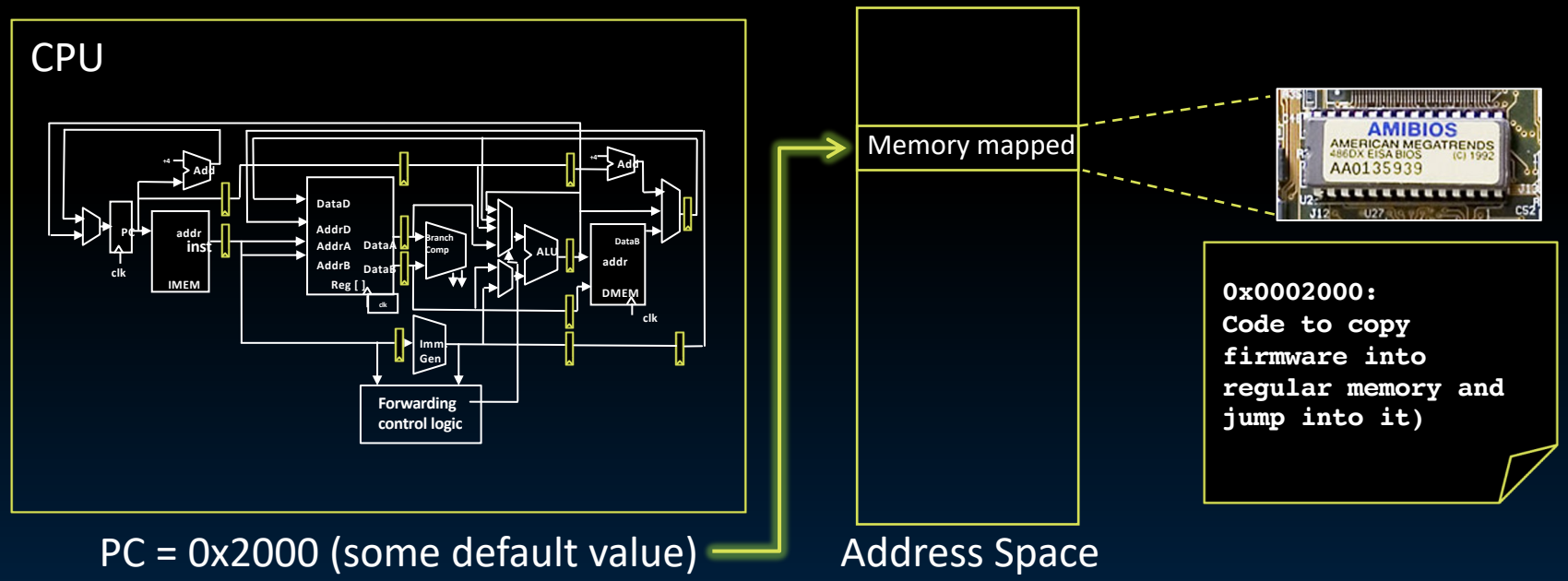
OS: Boot

- OS: Supervisor Mode, Exceptions [continued]
- OS: Boot [recorded]
- Caches vs. Virtual Memory
- The Translation Lookaside Buffer
- TLBs in the Datapath
- VM Performance

What Happens at Boot? (1/2)

(recorded)

- **When the computer switches on, it does the same as Venus:**
  - The CPU executes instructions from some start address stored in Flash ROM (Read-Only Memory).



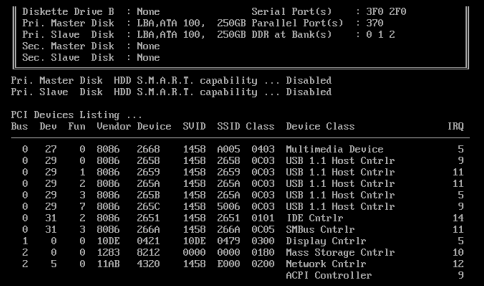
What Happens at Boot? (2/2)

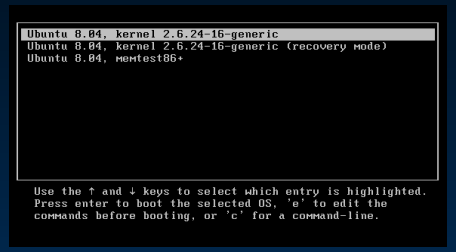
(recorded)

- **Then, the BIOS (Basic Input Output System) firmware loads the bootloader, which loads the OS kernel.**

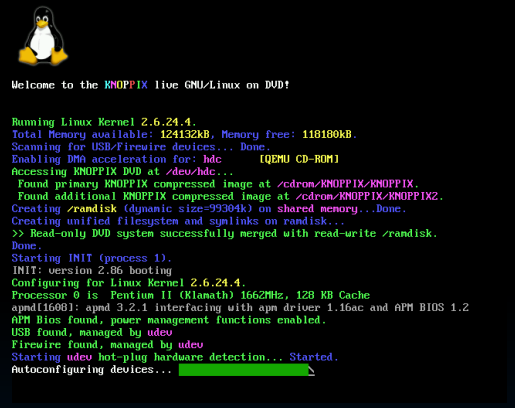
1. *BIOS*: Find a storage device and load the first sector (block of data).

2. *Bootloader*: (stored on, e.g., disk) Load the OS kernel from disk into a location in memory and jump into it.





3. *OS Boot*: Initialize services, drivers, etc.



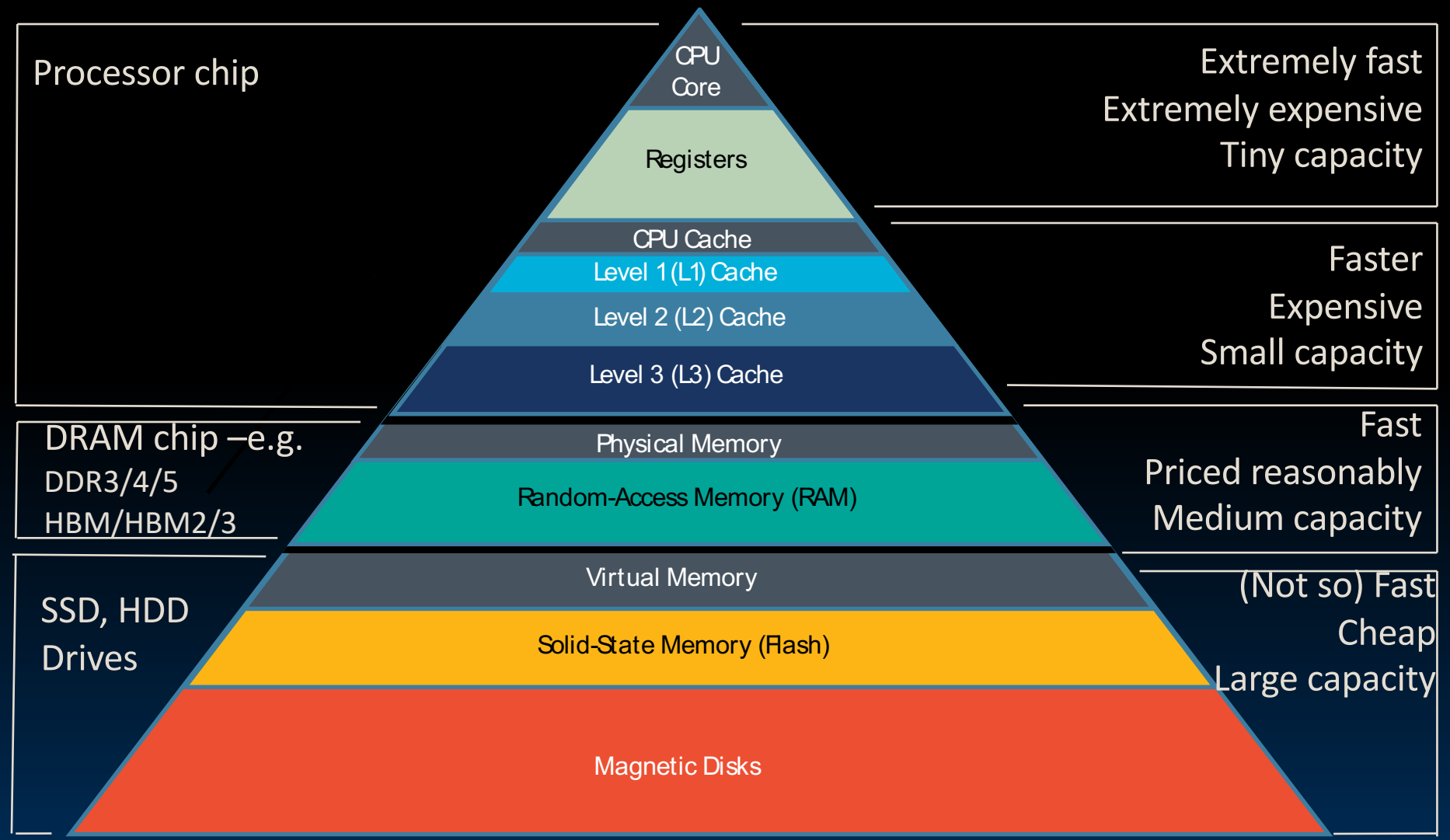
4. *Init*: Launch an application (e.g., Terminal/Desktop/…) that waits for input in loop.

Caches vs. Virtual Memory

- OS: Supervisor Mode, Exceptions [continued]
- OS: Boot [recorded]
- Caches vs. Virtual Memory
- The Translation Lookaside Buffer
- TLBs in the Datapath
- VM Performance

The Entire Modern Memory Hierarchy

Let's review the concepts of caches and memory.



Caches vs. Primary Memory

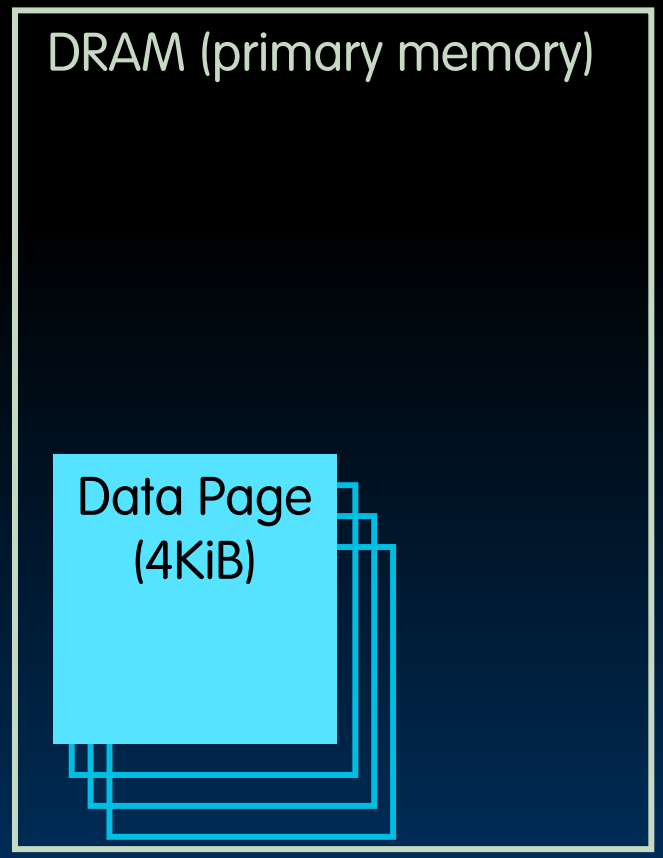
- **Blocks, pages, (bytes, words) are all units of memory.**
  - Caches: *blocks*
    - On modern systems, ~64B.
  - Memory: *pages*
    - On modern systems, ~4KiB.

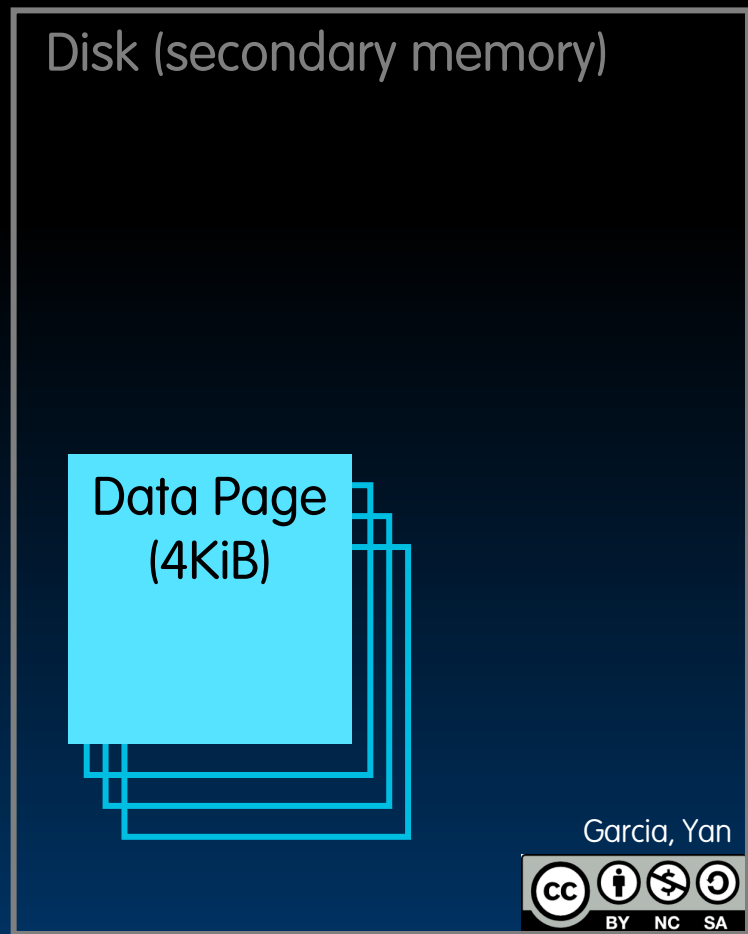
L1 Cache

V	Tag	Data
		Block (64B)
…	…	…

L2 Cache

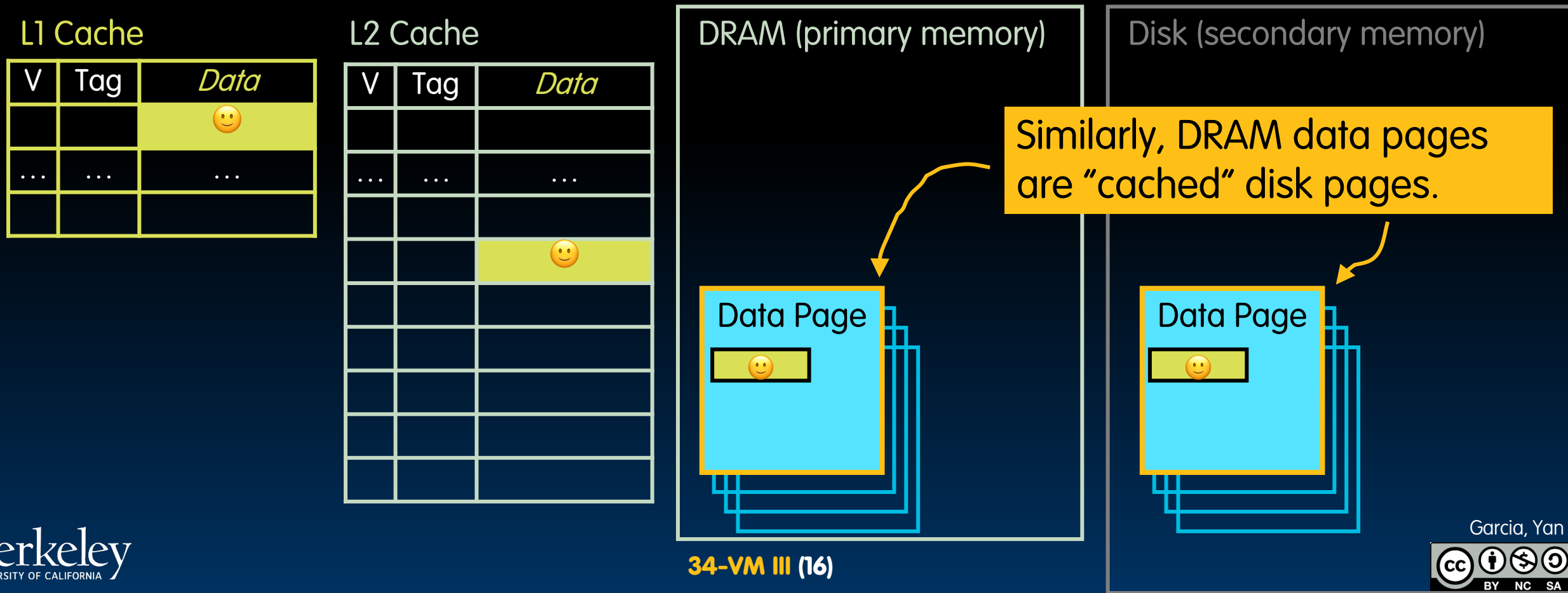
V	Tag	Data
		Block (64B)
…	…	…





Caches vs. Page Tables

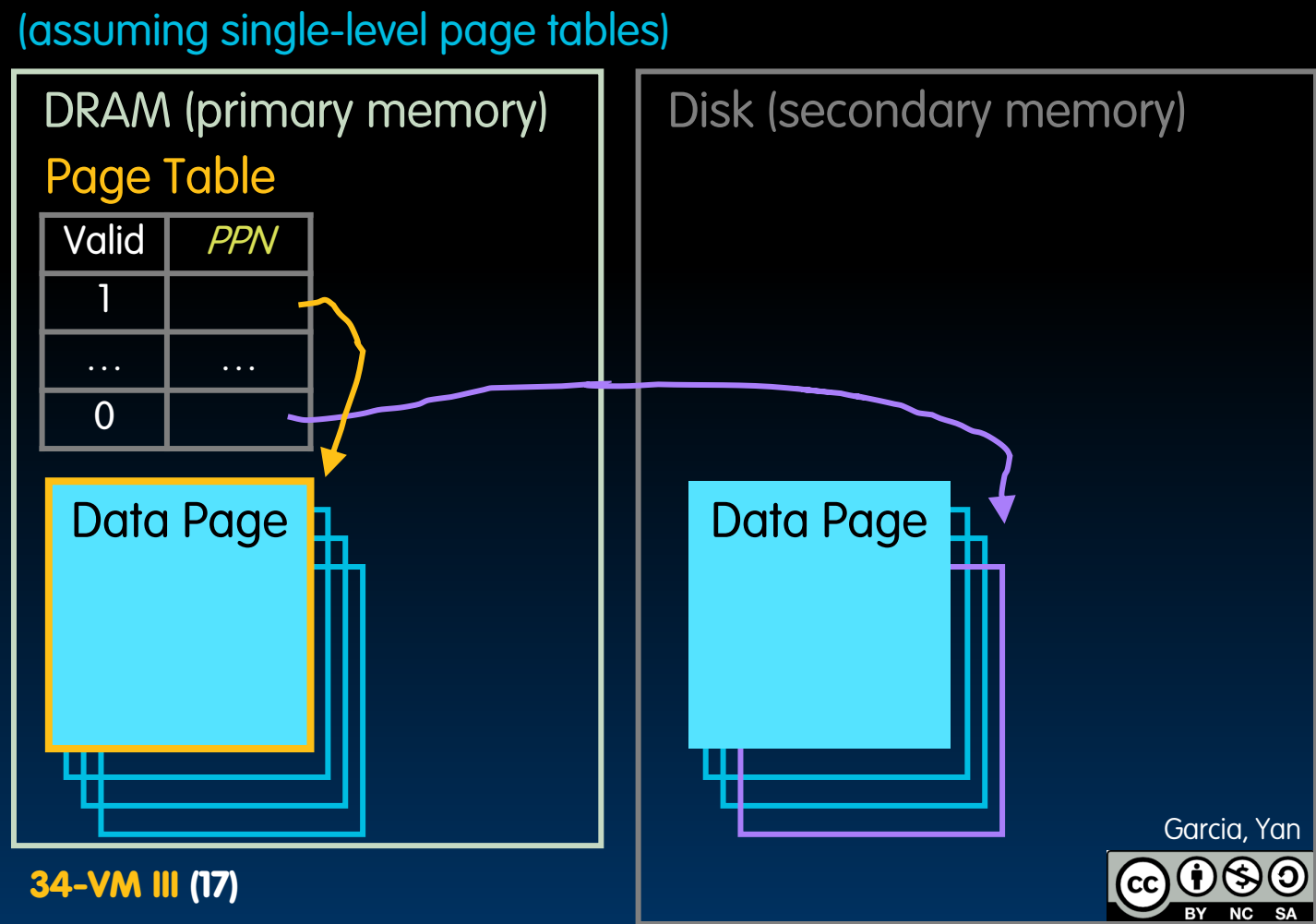
"Cache" Paradigm: Data at each level is a quick-access copy of data at a lower level in the memory hierarchy.



Caches vs. Page Tables

- **A Page Table translates addresses.**
  - Page tables store physical page numbers, *not data*.

- **Page tables facilitate Demand Paging.**
  - Cache data pages in memory.
  - Access disk pages only when needed by the process.
  - Page Table keeps track of page status/location.



Caching vs. Demand Paging

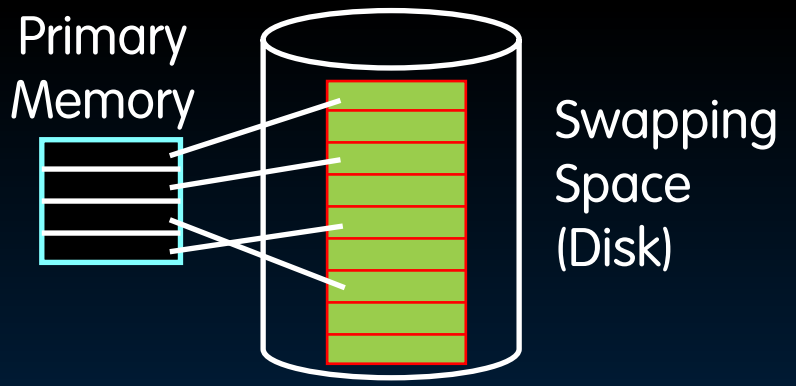
	Caching	Demand Paging
▫ Memory Unit	Block	Page
▪ Size	32B to 64B	4KiB to 8KiB
▪ Miss	Cache Miss	Page Fault
▫ Associativity	Direct-mapped, N-way Set associative, or fully associative	Fully associative (i.e., disk pages can be placed anywhere in DRAM)
▫ Replacement policy	Least Recently Used (LRU) or random	LRU (most common), or FIFO, or random
▫ Write policy	Write-through or write-back	Write-back

Translation Lookaside Buffer

- OS: Supervisor Mode, Exceptions [continued]
- OS: Boot [recorded]
- Caches vs. Virtual Memory
- The Translation Lookaside Buffer
- TLBs in the Datapath
- VM Performance

Modern Virtual Memory Systems

- **Modern Virtual Memory Systems use address translation to provide the illusion of a large, private, and uniform storage.**
  1. Privacy means *Protection*:
     - Several users/processes, each with their own private address space.
  2. Uniform storage means *Demand Paging*:
     - The ability to run programs larger than primary memory (DRAM).
     - Hides difference in machine configurations.

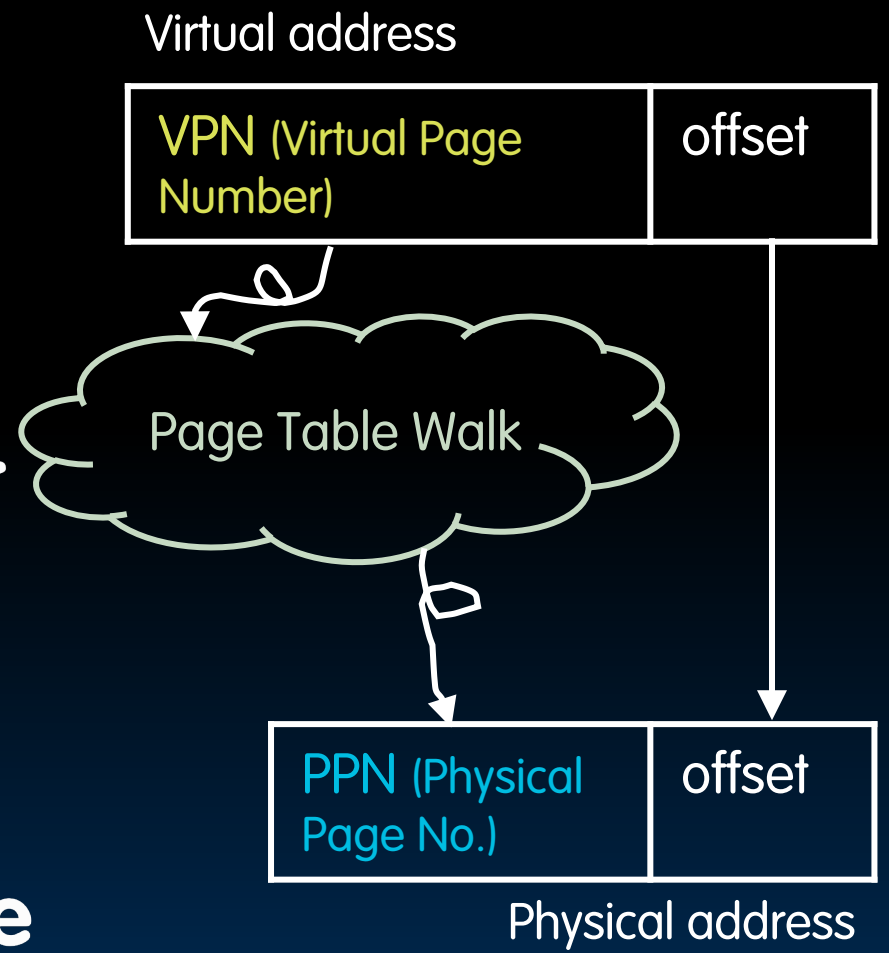


- **Price: Address translation on each memory reference.**

Page tables in memory significantly increase average memory access time!

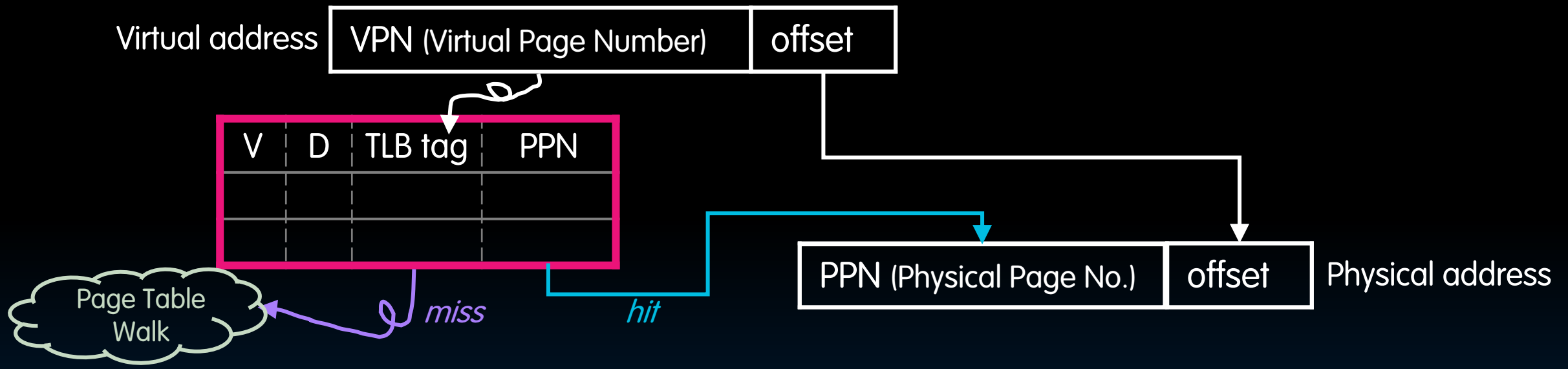
Speeding Up Address Translation

- **Good Virtual Memory design should be fast (~1 clock cycle) and space efficient.**
  - Every instruction/data access needs address translation.
- **But if page tables are in memory, then we must perform a page table walk per instruction/data access:**
  - Single-level page table: 2 memory accesses.
  - Two-level page table: 3 memory accesses.
- **Solution: Cache some translations in the Translation Lookaside Buffer (TLB).**



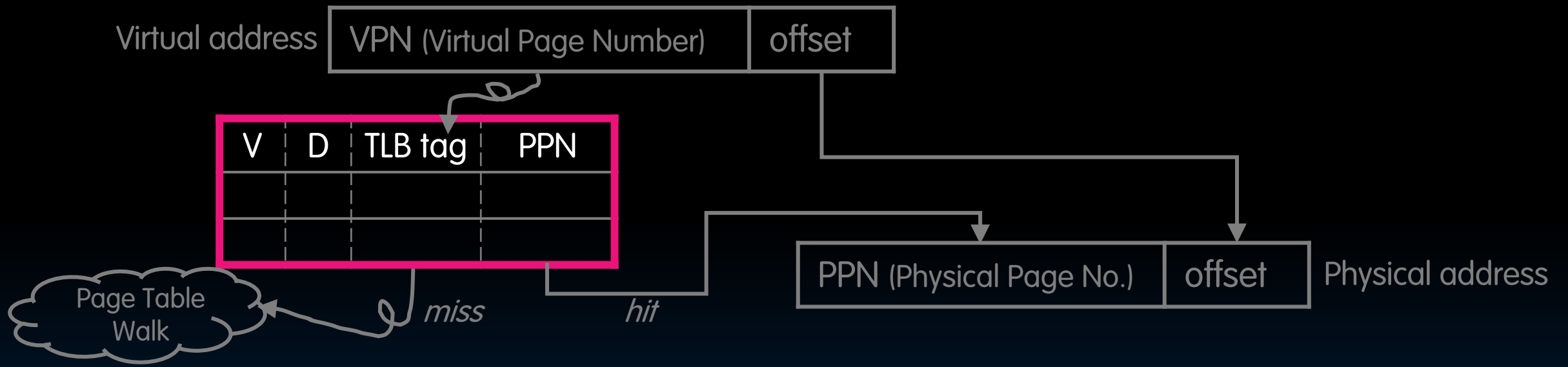
The TLB Is a Cache for Address Translations

- **The Translation Lookaside Buffer (TLB) caches page table entries.**
  - TLB *hit*: → Single-cycle translation ✅
  - TLB *miss*: → Page table walk to refill.



The TLB Is a Cache for Address Translations

- **The** Translation Lookaside Buffer (TLB) **caches page table entries.**
  - TLB *hit*: → Single-cycle translation ✅
  - TLB *miss*: → Page table walk to refill.

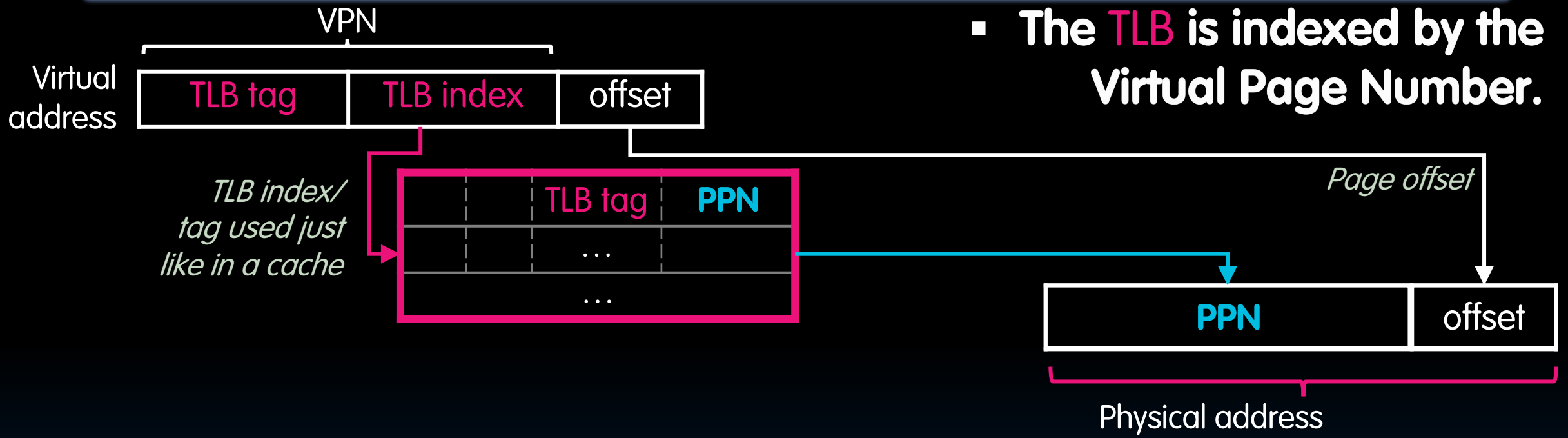


- TLB Reach: **Size of largest virtual address space that can be simultaneously mapped by the TLB.**
- **TLB design: 38-128 entries.**
  - Typically *fully associative* (increase TLB reach by minimizing conflicting entries).
  - Random/FIFO replacement policy.

Tag, Index, and Offset

TIO for Virtual Addresses and Physical Addresses are *unrelated*!

- **The TLB is indexed by the Virtual Page Number.**



Tag, Index, and Offset

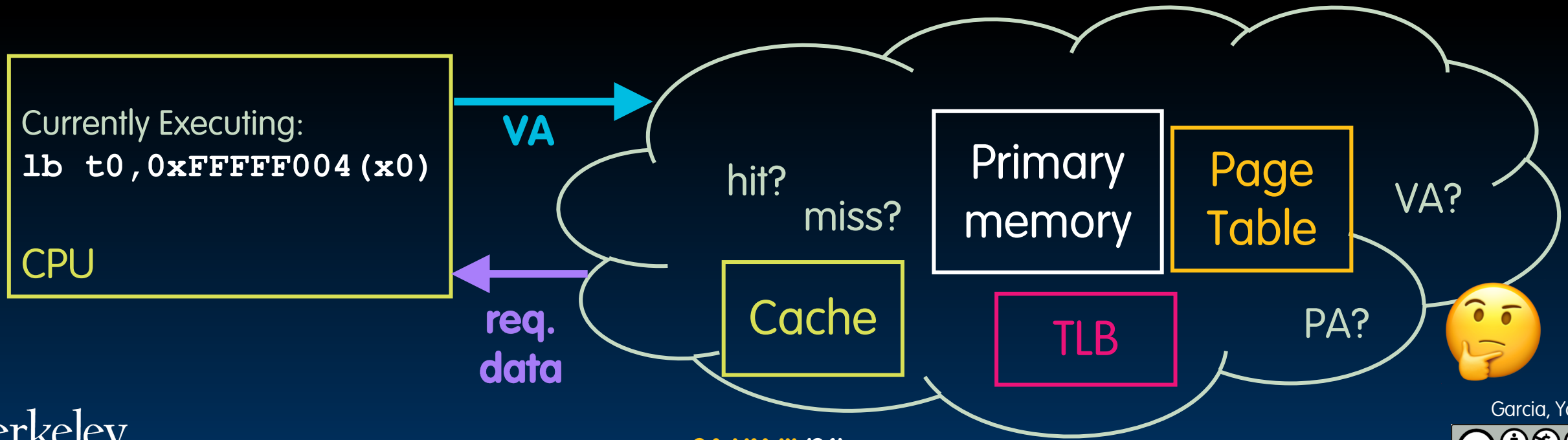
TIO for Virtual Addresses and Physical Addresses are *unrelated*!

- The TLB is indexed by the Virtual Page Number.

- The data cache is indexed by the Physical Address.

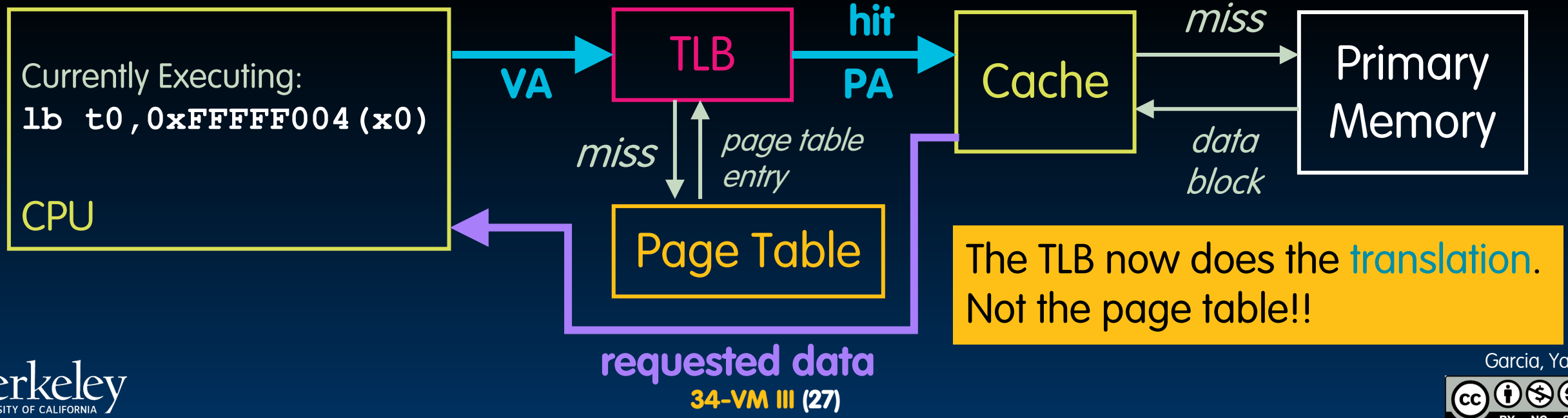
Memory Access: TLB, Cache, DRAM, Page Table?

1. Can a cache hold the requested data if the corresponding page is *not* in main memory?
2. On a memory reference, which block should we access first? When should we translate virtual addresses?



Memory Access: TLB, Cache, DRAM, Page Table

1. **Can a cache hold the requested data if the corresponding page is *not* in main memory?** No!
2. **On a memory reference, which block should we access first? When should we translate virtual addresses?**
   - We will assume *Physically Indexed, Physically Tagged* caches (other designs exist).
   - This means TLB first, then cache.

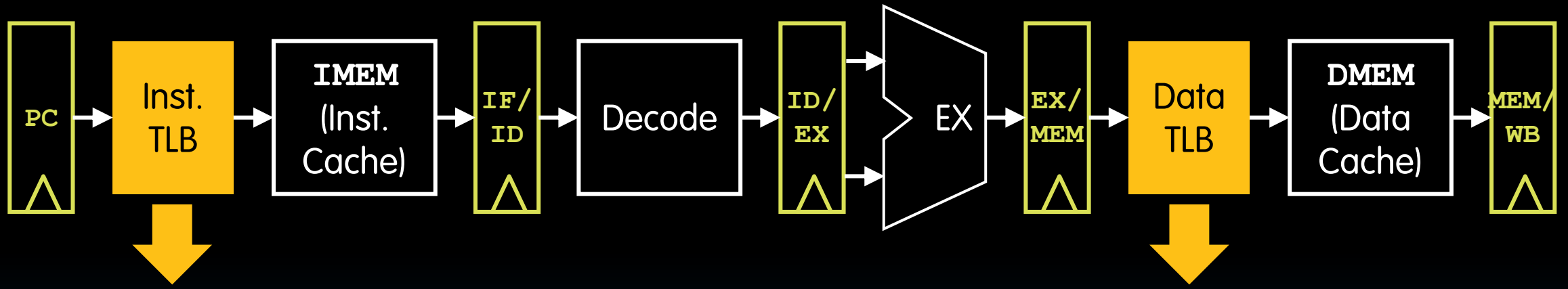


TLBs in the Datapath

- OS: Supervisor Mode, Exceptions [continued]
- OS: Boot [recorded]
- Caches vs. Virtual Memory
- The Translation Lookaside Buffer
- TLBs in the Datapath
- VM Performance

Virtual Memory and the CPU Pipeline

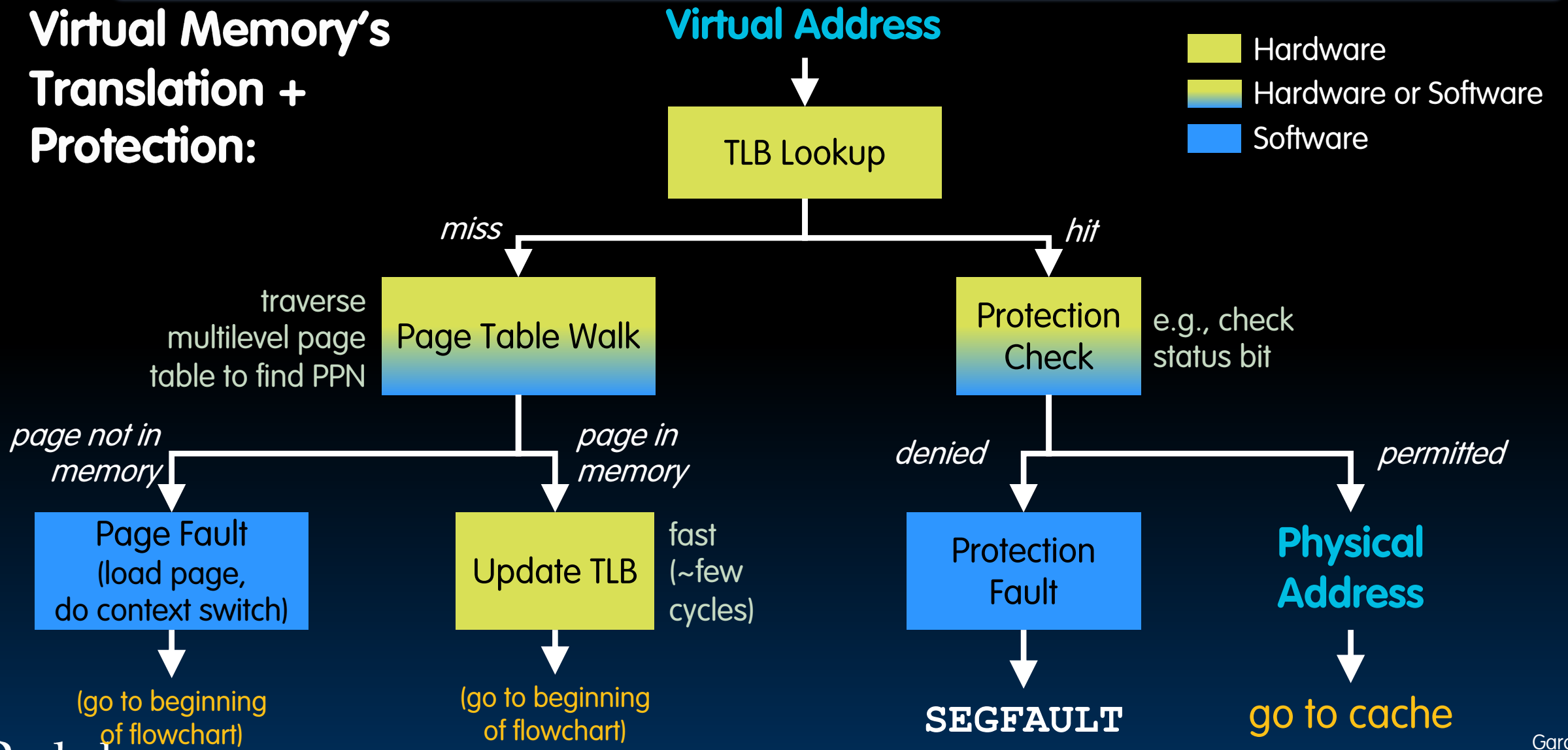
Virtual Memory = address translation + protection + demand paging.



- **Each instruction/data access = address translation + functional checks.**
- **Should handle:**
  1. *TLB Miss*: Needs a mechanism to refill TLB (usually done in hardware).
  2. *Page Fault* (i.e., page on disk)
     - Needs a *precise trap* so that software handler can easily re-execute instruction after page retrieval.
  3. *Protection violation* check
     - A violation may abort the process, e.g., `SEGFAULT`.

Virtual Memory Action Flowchart

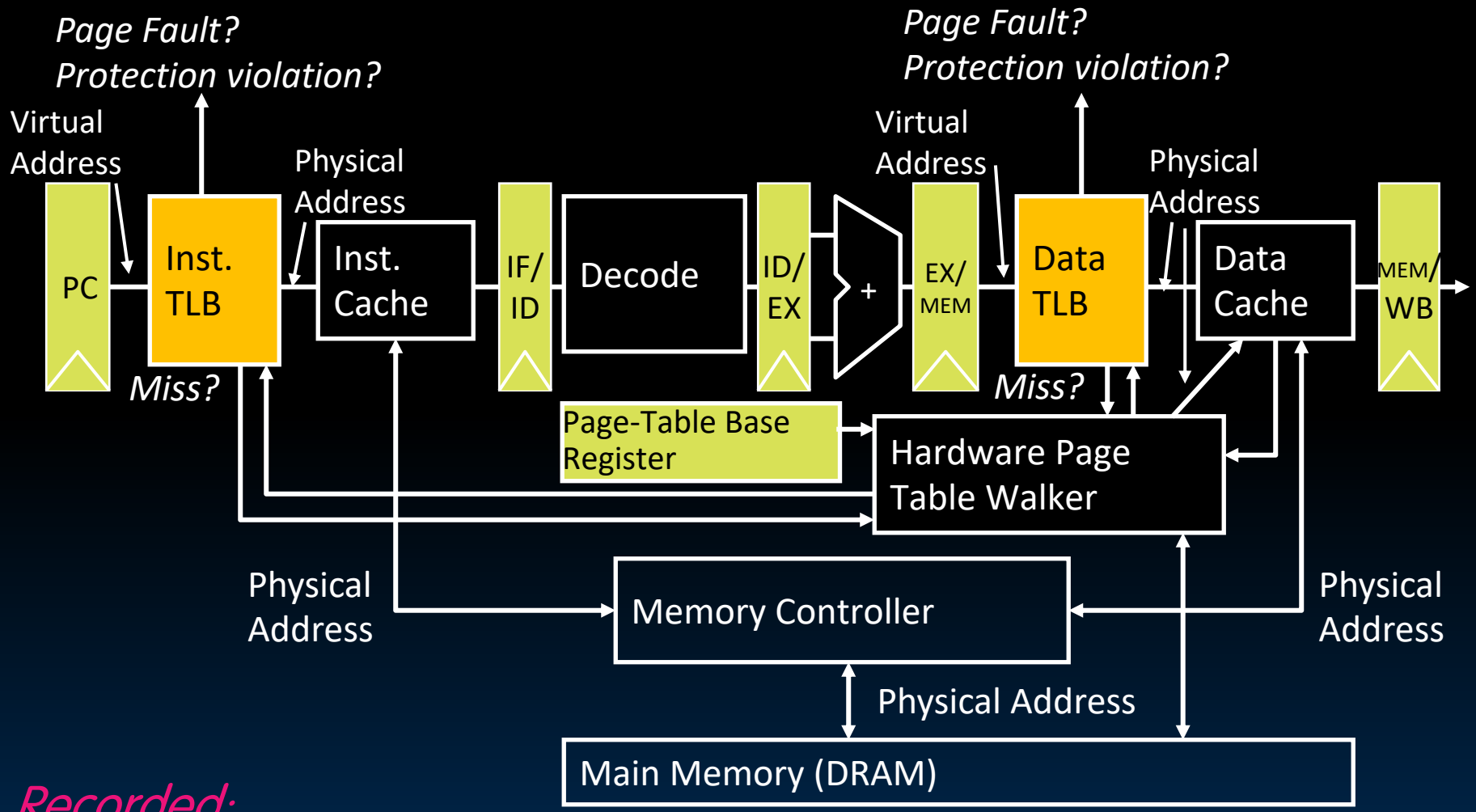
Virtual Memory's Translation + Protection:



Handling Context Switches and TLBs

- **Context switches should be fast. Avoid DRAM/disk updates.**
  - Keep all page tables for all currently running processes in DRAM.
  - Instead, ensure that all TLB entries refer to the *active process*.
- **The high-level context switch:**
  - The OS sets a timer. When it expires, perform a *hardware interrupt*.
  - Trap handler saves all register values, including:
    - Program Counter (PC)
    - *Page Table Register* (SPTBR in RV32I)
      - The memory *address* of the active process's page table.
  - Trap handler also sets all TLB entries to invalid. (other strategies exist)
  - Trap handler then loads in the next process's registers and returns to user mode.

A Full, Page-Based Virtual Memory Machine (optional)



(Assume page tables are held in untranslated physical memory)

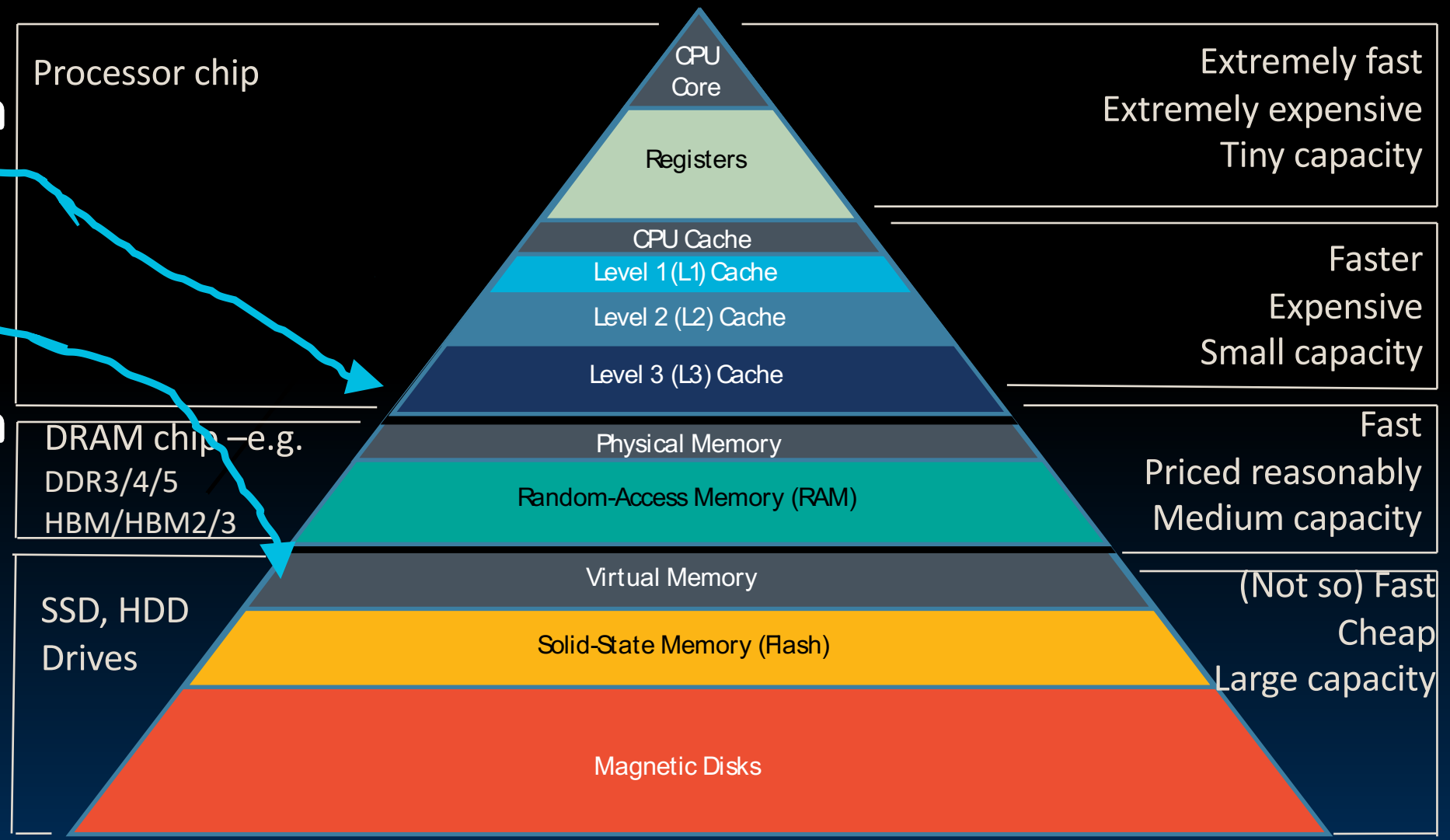
Recorded:
https://youtu.be/eVIsejli9hU

Virtual Memory Performance

- OS: Supervisor Mode, Exceptions [continued]
- OS: Boot [recorded]
- Caches vs. Virtual Memory
- The Translation Lookaside Buffer
- TLBs in the Datapath
- VM Performance

The Entire Modern Memory Hierarchy

- **Cache policies manages memory between cache and DRAM.**
- **Virtual Memory manages memory between DRAM and disk.**
  - TLB comes *before* the cache but affects transfer of data between DRAM/disk.



Average Memory Access Time (AMAT)

- **Recall: Performance is Average Memory Access Time (AMAT).**

$$AMAT = (hit\ time) + (miss\ rate) \times (miss\ penalty)$$

- **Previously, we treated main memory as the lowest level.**
- **Now with demand paging (virtual memory):**
  - Disk is lowest level.
  - Main memory is like a mid-level cache.
- **Main memory also has a hit rate:**
  - Hit rate = 1 – Page Fault Rate

🤔Design question: What is a reasonable hit rate?

AMAT, DRAM only

L1 cache
- Hit Time: 1 cycle
- Hit Rate: 95%

L2 cache
- Hit Time: 10 cycles
- Hit Rate: 60% (of L1 misses)

DRAM
- Hit Time: 200 cycles (≈ 100 ns if 2GHz)

- **Average Memory Access Time with DRAM only:**

$$AMAT_{no} = 1 + 5\% \times (\text{L1 miss penalty})$$
$$= 1 + 5\% \times \left(10 + 40\% \times (\underbrace{\text{L2 miss penalty}}_{\text{200 cycles}})\right) = \text{5.5 clock cycles}$$

AMAT w/Demand Paging

L1 cache
- Hit Time: 1 cycle
- Hit Rate: 95%

L2 cache
- Hit Time: 10 cycles
- Hit Rate: 60% (of L1 misses)

DRAM
- Hit Time: 200 cycles (≈ 100 ns if 2GHz)
- **Hit Rate: HR_{mem}???**

Disk
- Hit Time: 20,000,000 cycles (≈ 10 ms if 2GHz)

- **Average Memory Access Time with DRAM only:**

$$AMAT_{no} = 1 + 5\% \times (\text{L1 miss penalty})$$
$$= 1 + 5\% \times \big(10 + 40\% \times (\underbrace{\text{L2 miss penalty}}_{200\text{ cycles}})\big) = \boxed{5.5 \text{ clock cycles}}$$

- **Average Memory Access Time with demand paging:**

$$AMAT_{dp} = 1 + 5\% \times \big(10 + 40\% \times (200 + (1 - HR_{mem}) \times 20{,}000{,}000)\big)$$
$$= \underbrace{5.5}_{AMAT_{no}} + \underbrace{(5\% \times 40\% \times (1 - HR_{mem}) \times 20{,}000{,}000)}_{\text{performance cost of demand paging (disk access)}} \quad \text{(distributive property)}$$

AMAT w/Demand Paging

L1 cache
- Hit Time: 1 cycle
- Hit Rate: 95%

L2 cache
- Hit Time: 10 cycles
- Hit Rate: 60% (of L1 misses)

DRAM
- Hit Time: 200 cycles (≈ 100 ns if 2GHz)
- **Hit Rate: HR_{mem}???**

Disk
- Hit Time: 20,000,000 cycles (≈ 10 ms if 2GHz)

- **Average Memory Access Time with demand paging:**

$$AMAT_{dp} = 5.5 + (5\% \times 40\% \times (1 - HR_{mem}) \times 20{,}000{,}000)$$

A. $HR_{mem} = 99\%$

B. $HR_{mem} = 99.9\%$

C. $HR_{mem} = 99.9999\%$

Which proposed DRAM hit rate minimizes the performance cost of disk?

AMAT w/Demand Paging

L1 cache
- Hit Time: 1 cycle
- Hit Rate: 95%

L2 cache
- Hit Time: 10 cycles
- Hit Rate: 60% (of L1 misses)

DRAM
- Hit Time: 200 cycles (≈ 100 ns if 2GHz)
- **Hit Rate: HR_{mem}???**

Disk
- Hit Time: 20,000,000 cycles (≈ 10 ms if 2GHz)

- **Average Memory Access Time with demand paging:**

$$AMAT_{dp} = 5.5 + (5\% \times 40\% \times (1 - HR_{mem}) \times 20{,}000{,}000)$$

A. $HR_{mem} = 99\%$ $AMAT_A = 5.5 + (0.02 \times (.01) \times 20{,}000{,}000) = 4{,}005.5$ cycles
- 1 in 20,000 memory accesses goes to disk → 10 second program takes 20 hours!!

B. $HR_{mem} = 99.9\%$ $AMAT_B = 5.5 + (0.02 \times (.001) \times 20{,}000{,}000) = 405.5$ cycles

C. $HR_{mem} = 99.9999\%$ $AMAT_C = 5.5 + (0.02 \times (.000001) \times 20{,}000{,}000)$
$= 5.9$ cycles ✅

A reasonable *page fault rate* is ≪ 0.01%.

That's it for VM.
Happy Monday!

Question: How Many Bits in the TLB Entry?

Practice

- 16 KiB pages
  - 40-bit virtual addresses
  - 64 GiB physical memory
  - 2-way set associative TLB with 512 entries

VPN = 26 bits, PPN = 22 bits

Valid	Dirty	Ref	Access Rights	TLB Tag	PPN
X	X	X	XX		

Question: How Many Bits in the TLB Entry?

Practice

- 16 KiB pages
  - 40-bit virtual addresses
  - 64 GiB physical memory
  - 2-way set associative TLB with 512 entries

VPN = 26 bits, PPN = 22 bits

Valid	Dirty	Ref	Access Rights	TLB Tag	PPN
1	1	1	2	18	22

TLB entry: 45 bits