UC Berkeley
Teaching Professor
Dan Garcia

CS61C

Great Ideas in Computer Architecture

(a.k.a. Machine Structures)

UC Berkeley
Lecturer
Justin Yokota

Virtual Memory II

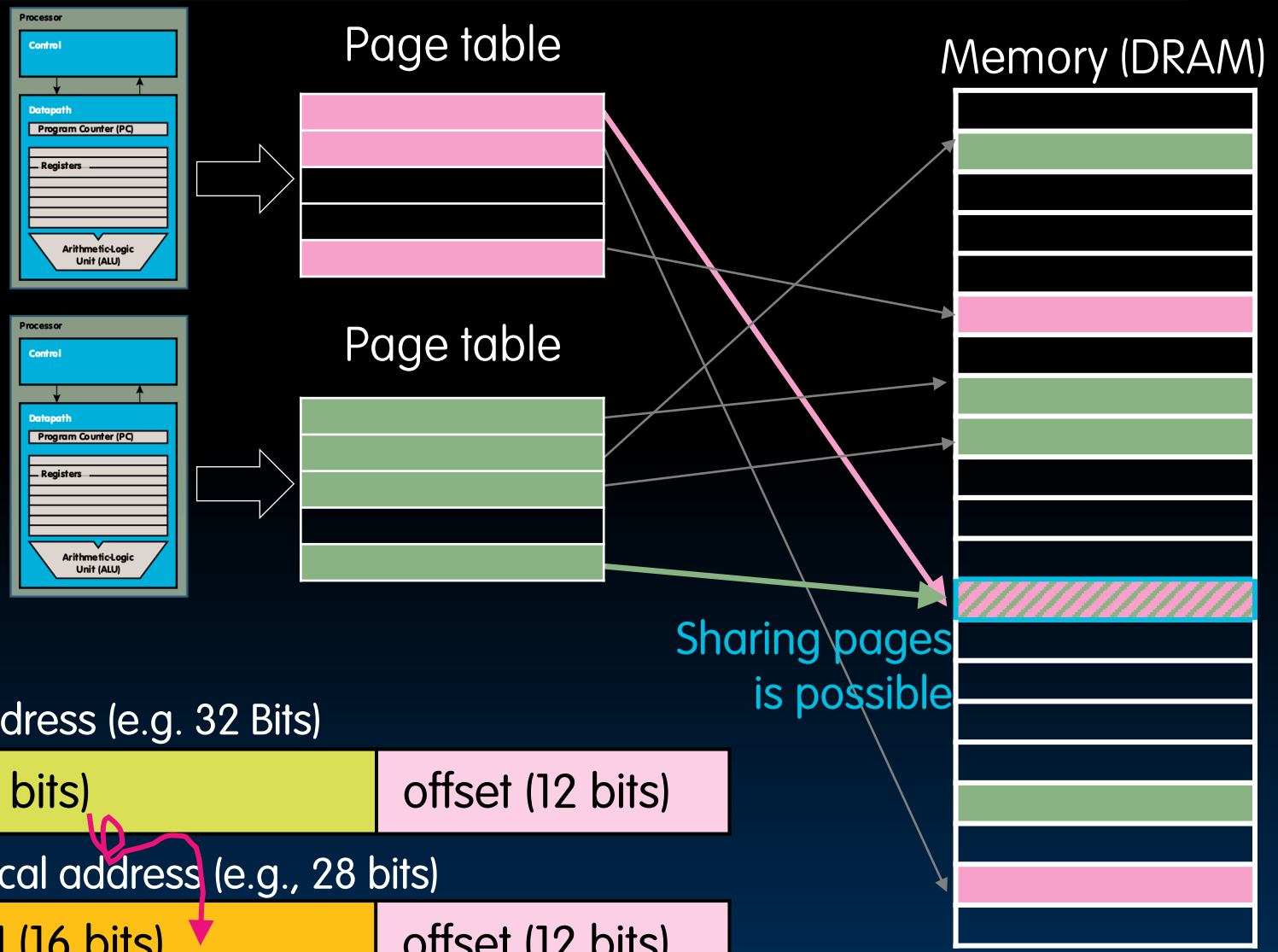
Page Table Details: Page Faults, Write Policy

- Page Table Details: Page Faults, Write Policy
- OS: Supervisor Mode, Exceptions
- Caches vs. Virtual Memory
- Translation Lookaside Buffer

Virtual Memory: Page Tables

Review

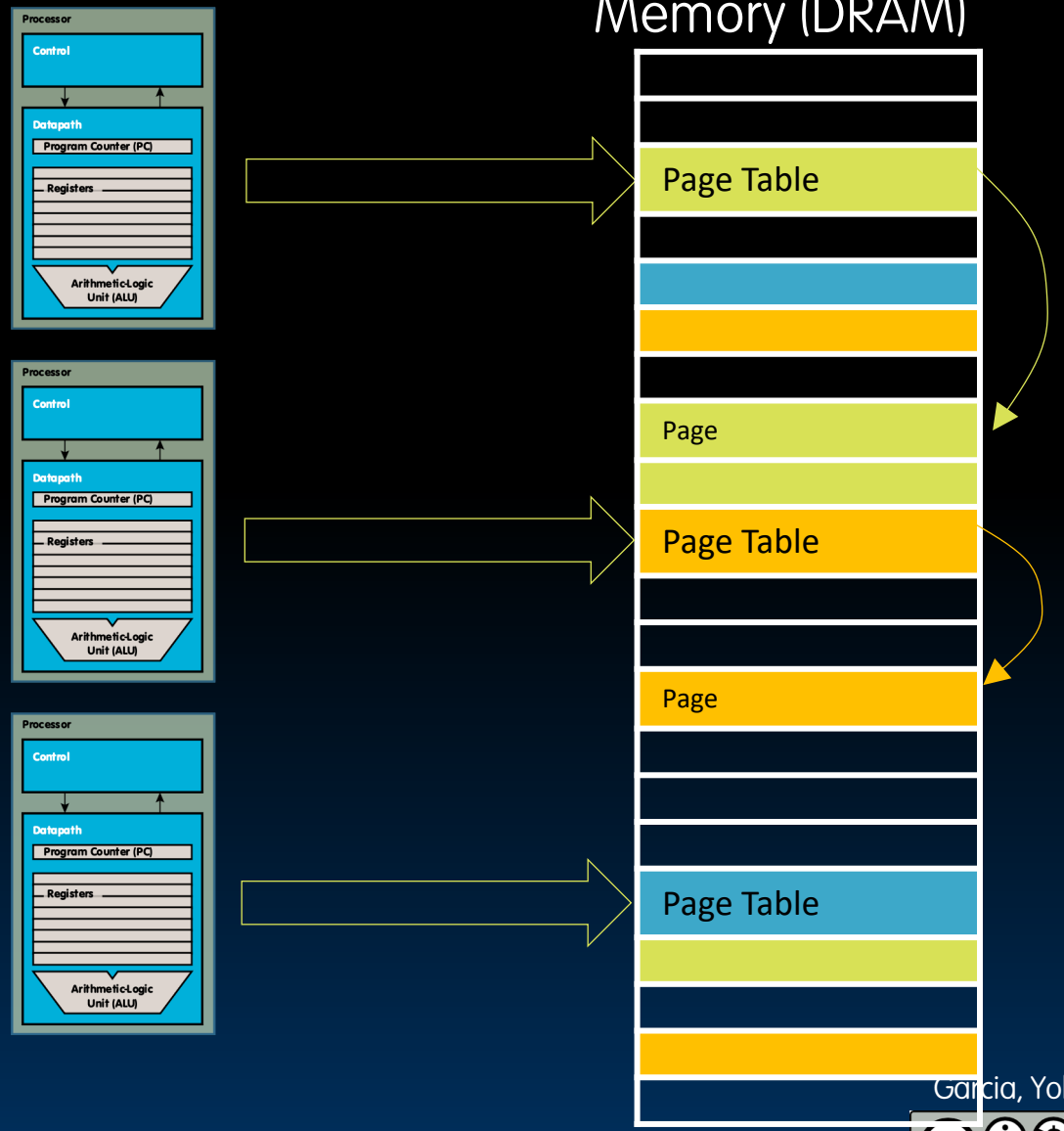
- **Each process has a dedicated page table.**
  - OS keeps track of which process is active.
- Isolation: **Assign processes different pages in DRAM**
  - Prevents (*protects*) a process from accessing other processes' data
  - Page tables managed by OS



We now have **two address spaces**: virtual (per process) and physical (memory).

Page Tables Are Stored in Memory

- **`lw/sw` require two memory accesses:**
  - Read page table (stored in main memory) to translate to physical address.
  - Read physical page, also in main memory.
- **To minimize space and time penalty:**
  - Transfer blocks (not words) between DRAM and processor cache.
  - Have a hierarchical multi-level page table
    - Not in scope this semester, see CS162 for more
  - Use a cache for frequently used page table entries… (more later, TLB)



Page Faults

- **Page table entries store status to indicate if the page is in memory (DRAM) or only on disk.**
  - On each memory access, check the page table entry "*valid*" status bit.
- **Valid → In DRAM**
  - Read/write data in DRAM
- **Not Valid → On disk**
  - Triggers a *Page Fault*; OS intervenes to allocate the page into DRAM.
  - If out of memory, first evict a page from DRAM.
  - Store evicted page to disk.
  - Read requested page from disk into DRAM.
  - Finally, read/write data in DRAM.

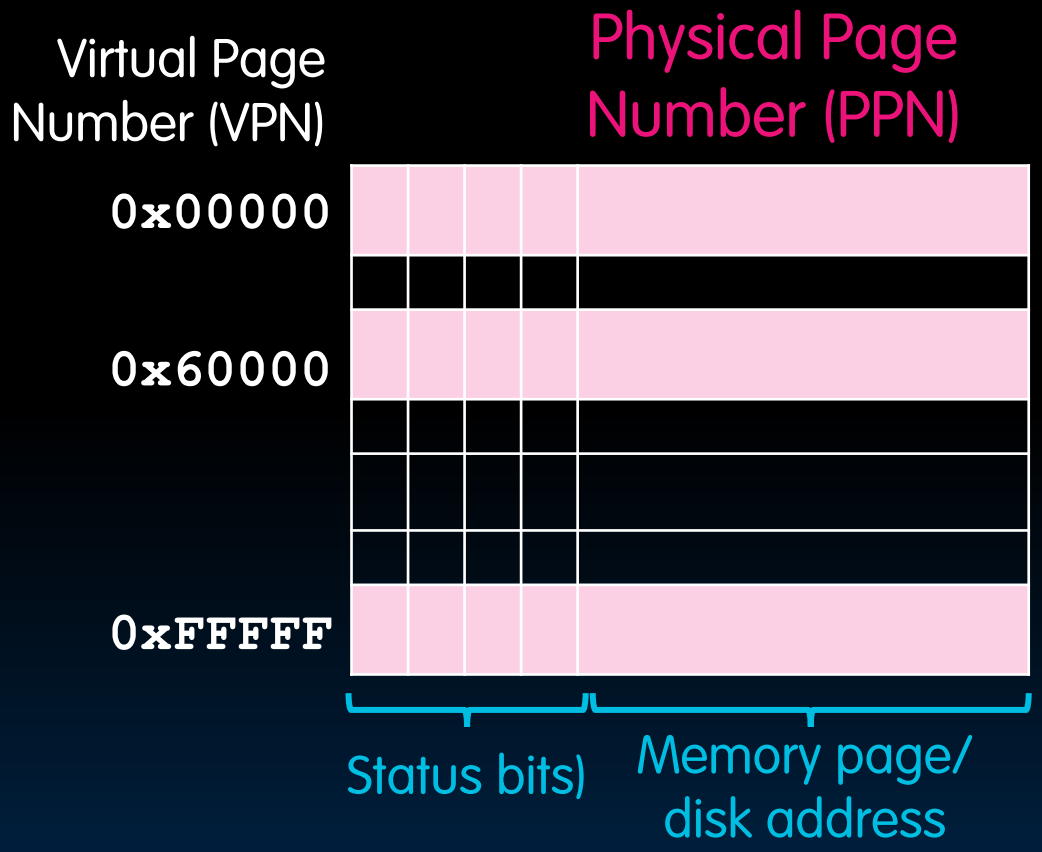
The *page replacement policy* (e.g., LRU/FIFO/random) is usually done in OS/software; this overheard << disk access time.

Memory's Write Policy?

- **DRAM acts like a "cache" for disk.**
  - Should writes always go directly to disk (write-through), or
  - Should writes only go to disk when page is evicted (write-back)?
- **Answer: All virtual memory systems use write-back.**
  - Disk accesses take too long!

Page Table Metadata: Status Bits

- **Write Protection Bit**
  - On: If process writes to page, trigger exception
- **Valid Bit**
  - On: Page is in RAM
- **Dirty Bit**
  - On: Page on RAM is more up-to-date than page on disk



OS: Supervisor Mode, Exceptions

- Page Table Details: Page Faults, Write Policy
- OS: Supervisor Mode, Exceptions
- Caches vs. Virtual Memory
- Translation Lookaside Buffer

Supervisor Mode vs. User Mode

- **If an application goes wrong (or rogue, e.g., malware), it could crash the entire machine!**
- **CPUs have a hardware supervisor mode (i.e., kernel mode).**
  - Set by a status bit in a special register.
  - An OS process in supervisor mode helps enforce constraints to other processes, e.g., access to memory, devices, etc.
  - Supervisor mode is a bit like "superuser"…
    - Errors in supervisory mode are often catastrophic (blue "screen of death", or "I just corrupted your disk").
- **By contrast, in user mode, a process can only access a subset of instructions and (physical) memory.**
  - Can change *out* of supervisor mode using a special instruction (e.g. `sret`).
  - Cannot change *into* supervisor mode directly; instead, HW interrupt/exception.
  - The OS mostly runs in user mode! Supervisor mode is used sparingly.

Exceptions and Interrupts

- **Exceptions**
  - Caused by an event *during* the execution of the current program.
  - *Synchronous*; must be handled immediately.
  - Examples:
    - Illegal instruction
    - Divide by zero
    - *Page fault*
    - Write protection violation

- **Interrupts** (more later)
  - Caused by an event *external* to the current running program.
  - *Asynchronous* to current program; does not need to be handled immediately (but should be soon).
  - Examples:
    - Key press
    - Disk I/O

Traps Handle Exceptions/Interrupts

- **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.

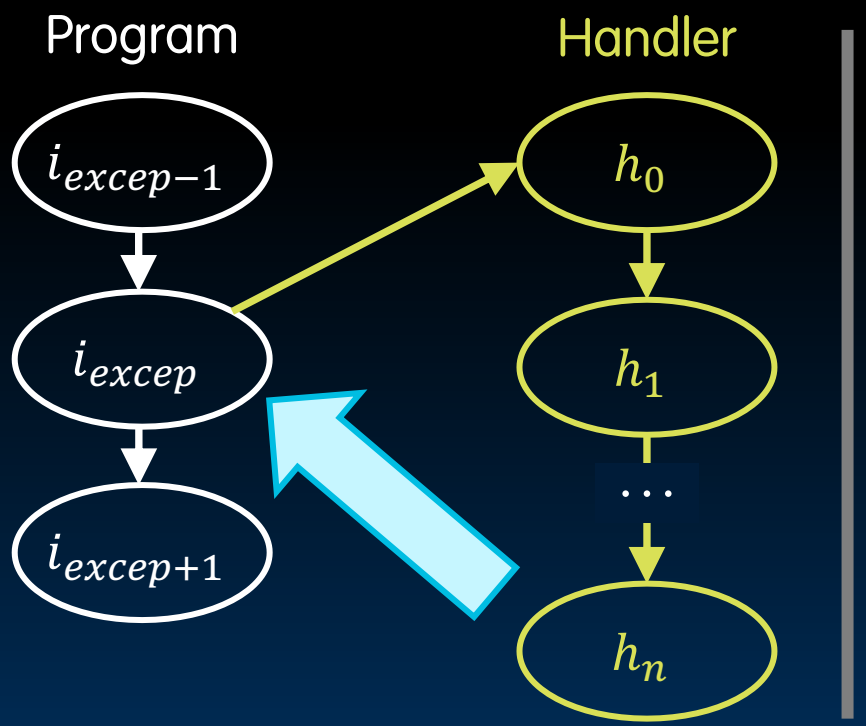
If the trap handler returns, then from the program's point of view it must look like nothing has happened!

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.**

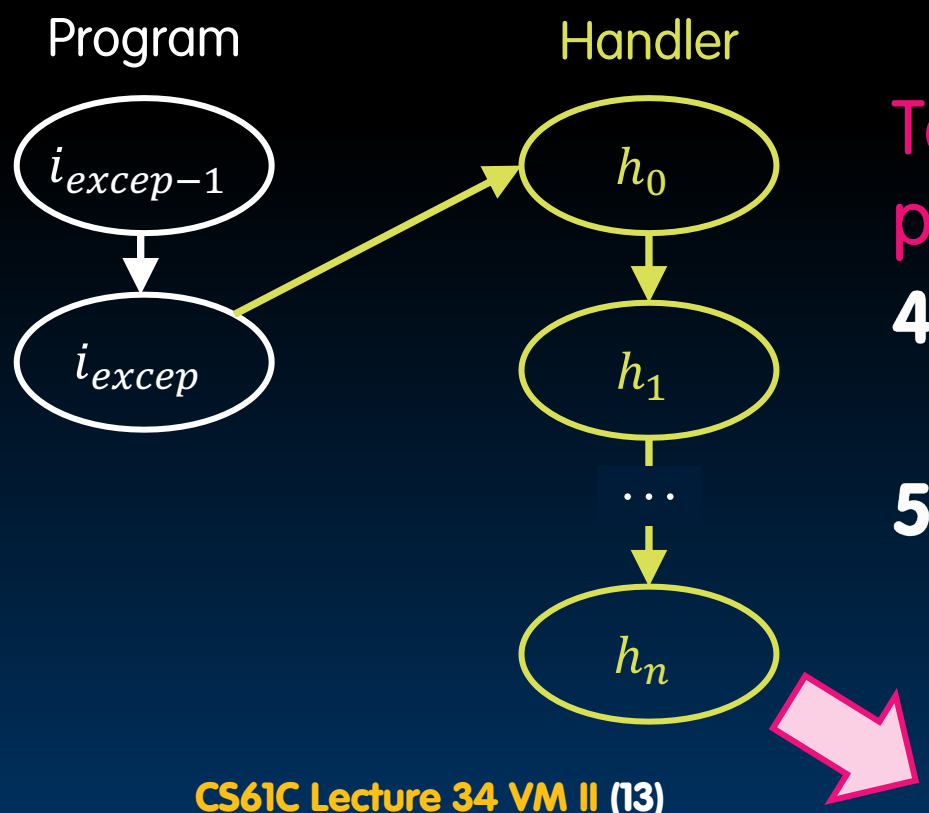


The Trap Handler

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

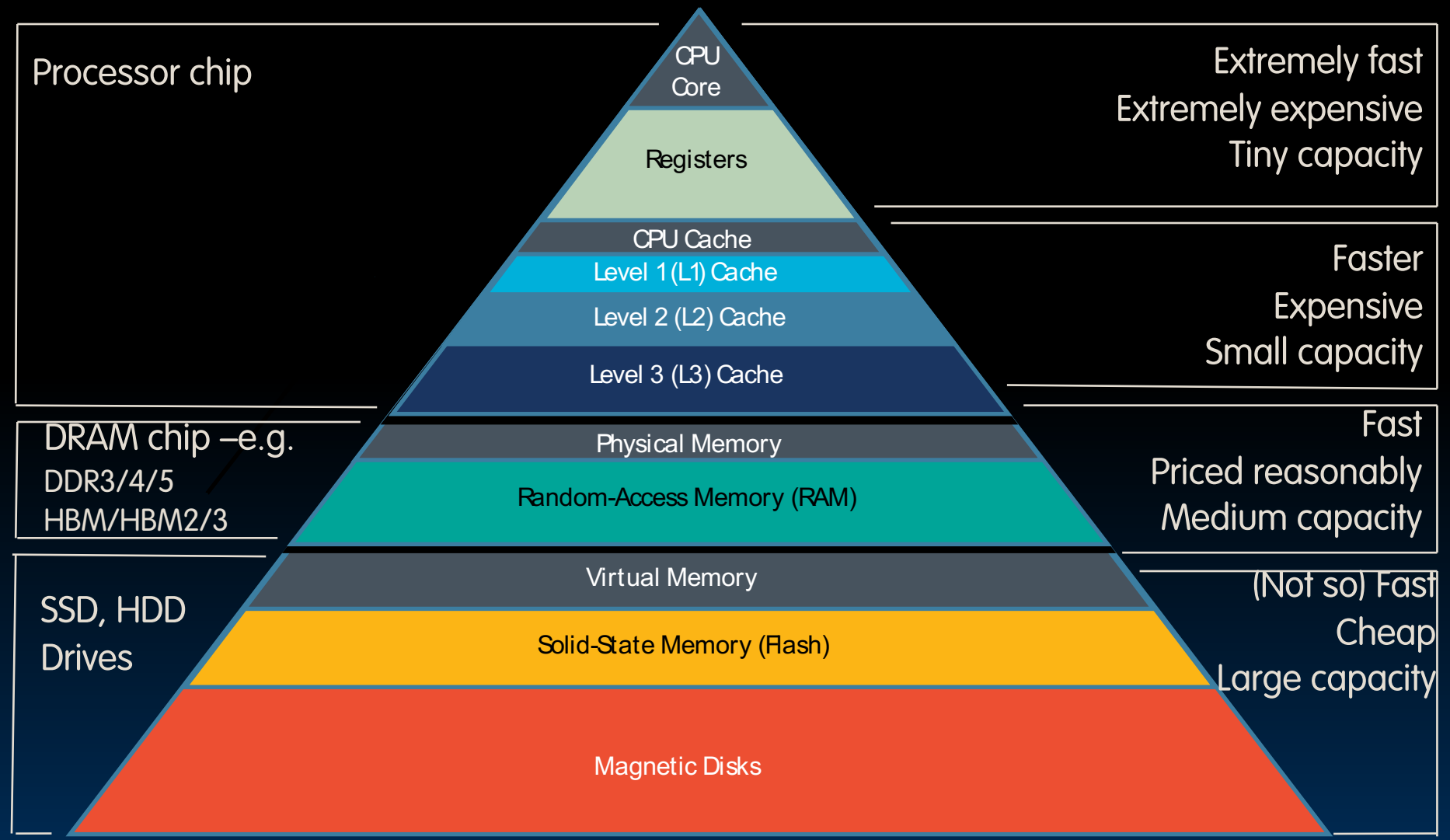
- **Recall** page faults:
  - An accessed page table entry has valid bit off → data is not in DRAM.
- **Page faults are handled by the trap handler.**
  - The *page fault exception handler* initiates transfers to/from disk and performs any page table updates.
  - (If pages needs to be swapped from disk,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

Caches vs. Virtual Memory

- Page Table Details: Page Faults, Write Policy
- OS: Supervisor Mode, Exceptions
- Caches vs. Virtual Memory
- Translation Lookaside Buffer

The Entire Modern Memory Hierarchy

Let's review the concepts of caches and memory.



Caches vs. Primary Memory

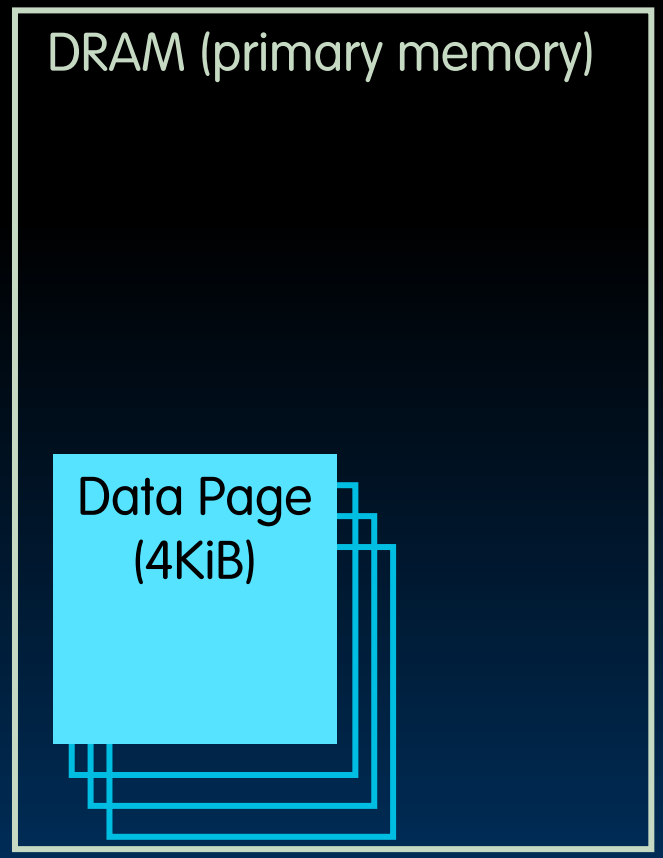
- **Blocks, pages, (bytes, words) are all units of memory.**
  - Caches: *blocks*
    - On modern systems, ~64B.
  - Memory: *pages*
    - On modern systems, ~4KiB.

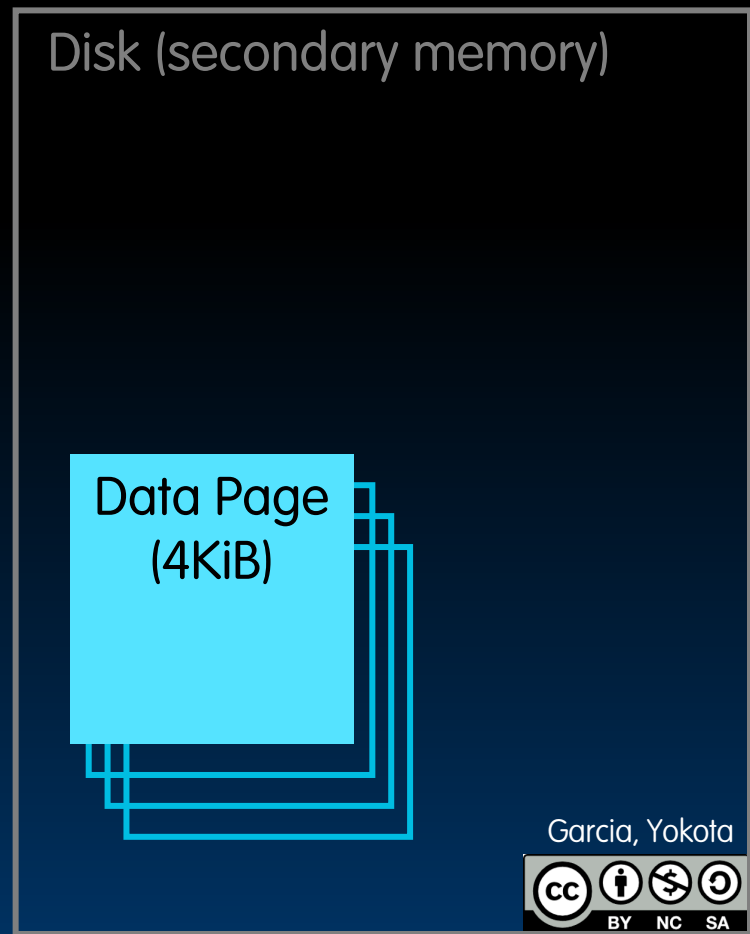
L1 Cache

V	Tag	Data
		Block (64B)
…	…	…

L2 Cache

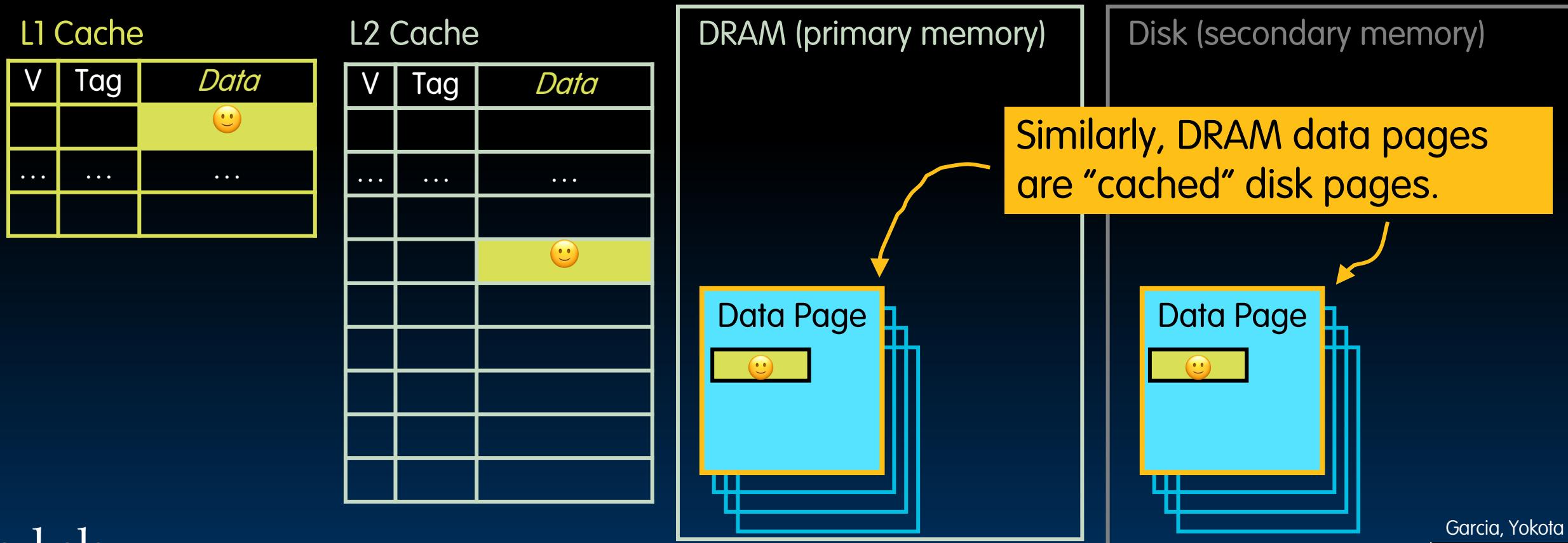
V	Tag	Data
		Block (64B)
…	…	…





Caches vs. Page Tables

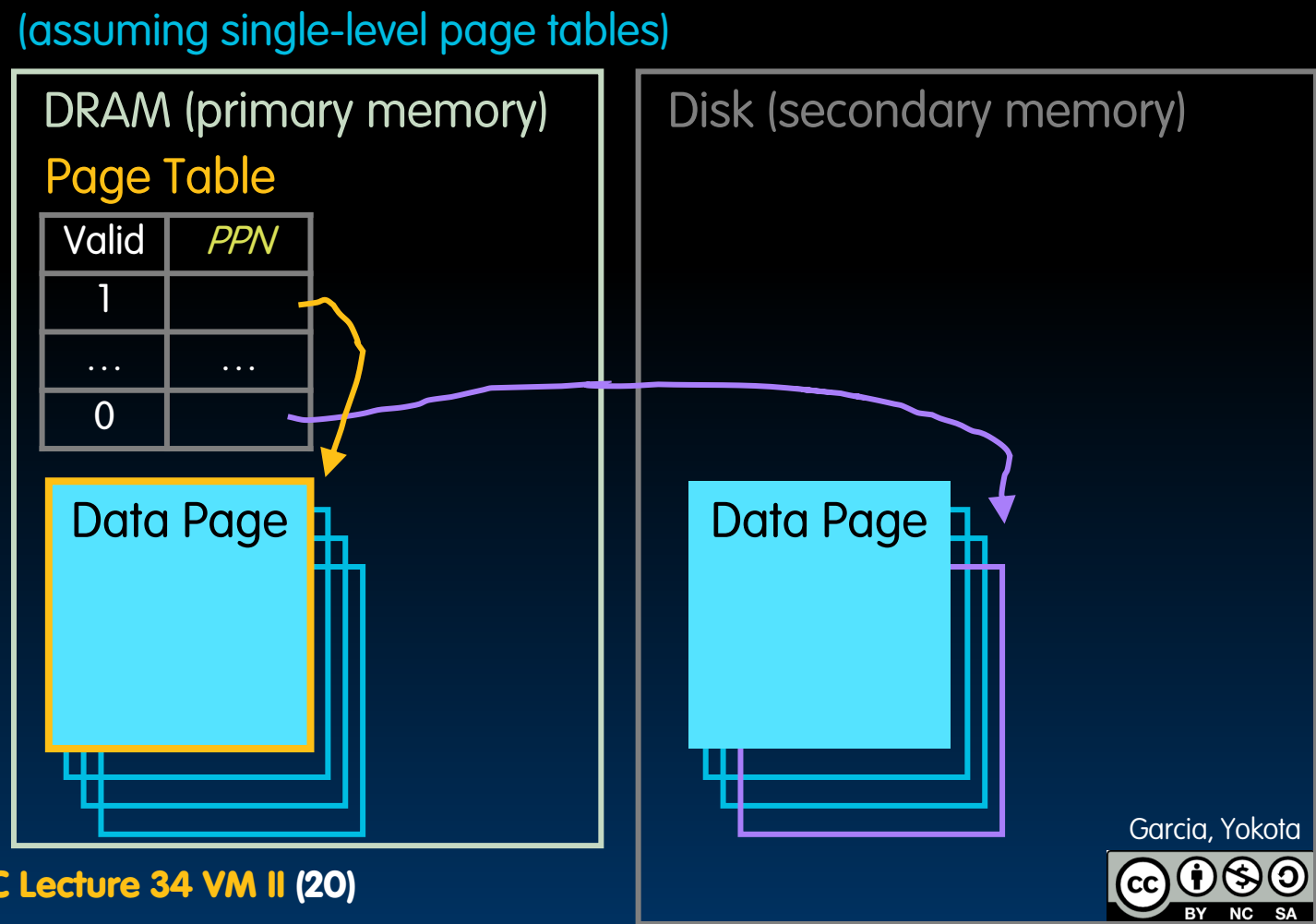
"Cache" Paradigm: Data at each level is a quick-access copy of data at a lower level in the memory hierarchy.



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

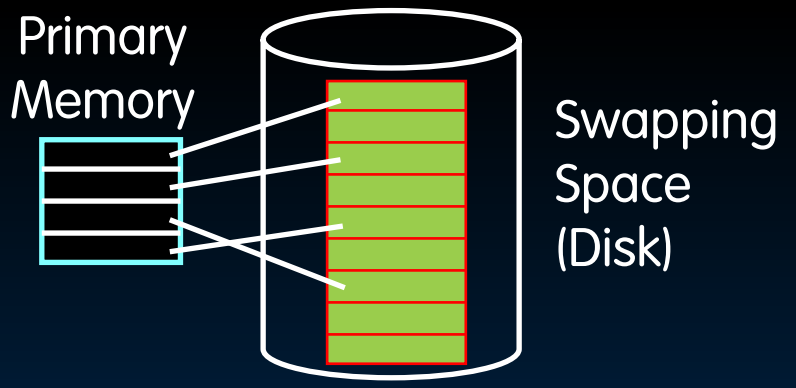
	Caching	Demand Paging
▫ Memory Unit	Block	Page
▪ Size	32B to 64B	4KiB to 8KiB
▪ Miss	Cache Miss	Page Fault
▫ Associativity	Direct-mapped, N-way Set associative, or fully associative	Fully associative (i.e., disk pages can be placed anywhere in DRAM)
▫ Replacement policy	Least Recently Used (LRU) or random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

Translation Lookaside Buffer

- Page Table Details: Page Faults, Write Policy
- OS: Supervisor Mode, Exceptions
- Caches vs. Virtual Memory
- Translation Lookaside Buffer

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.

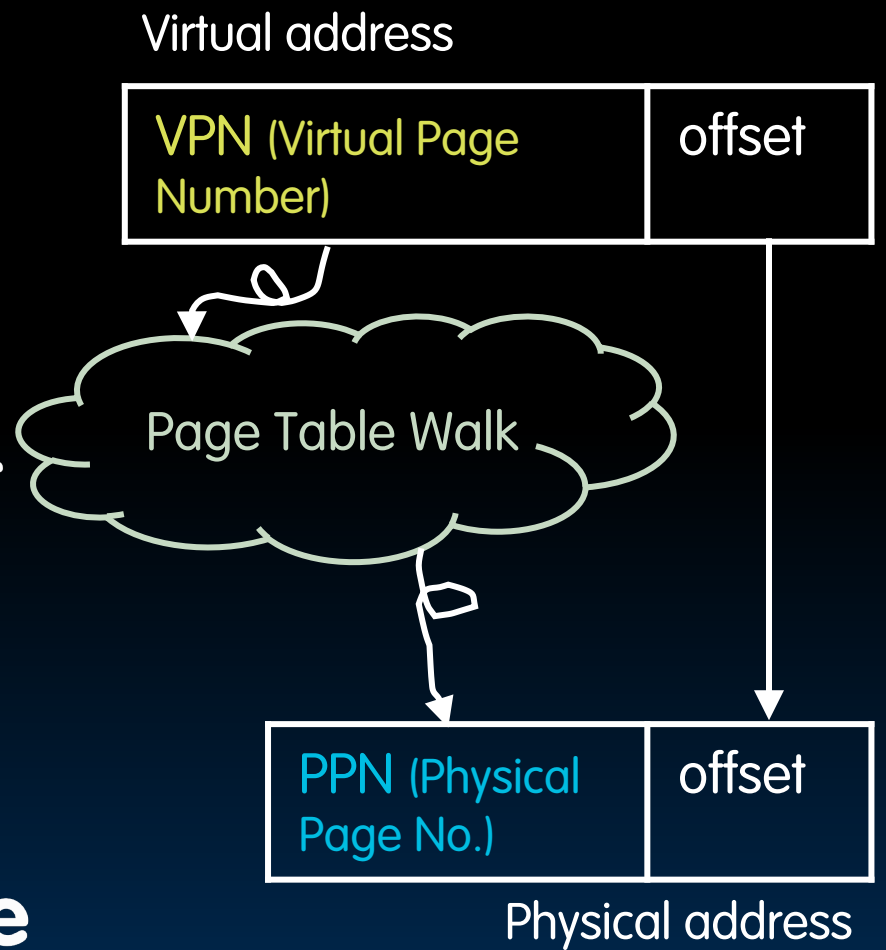


- **Price: Address translation on each memory reference.**

Page tables in memory significantly increase average memory access time!

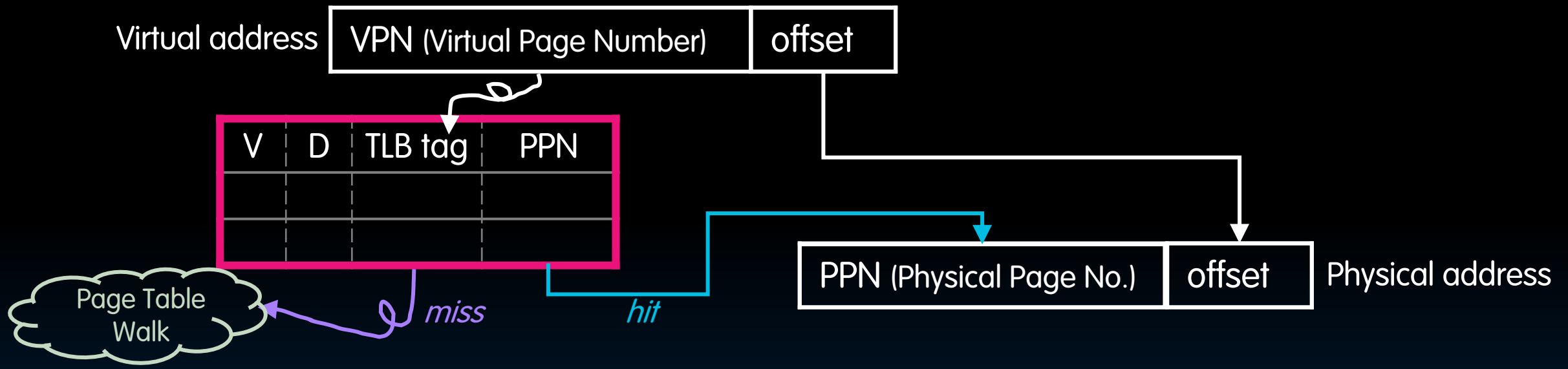
Speeding Up Address Translation

- **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)**.**



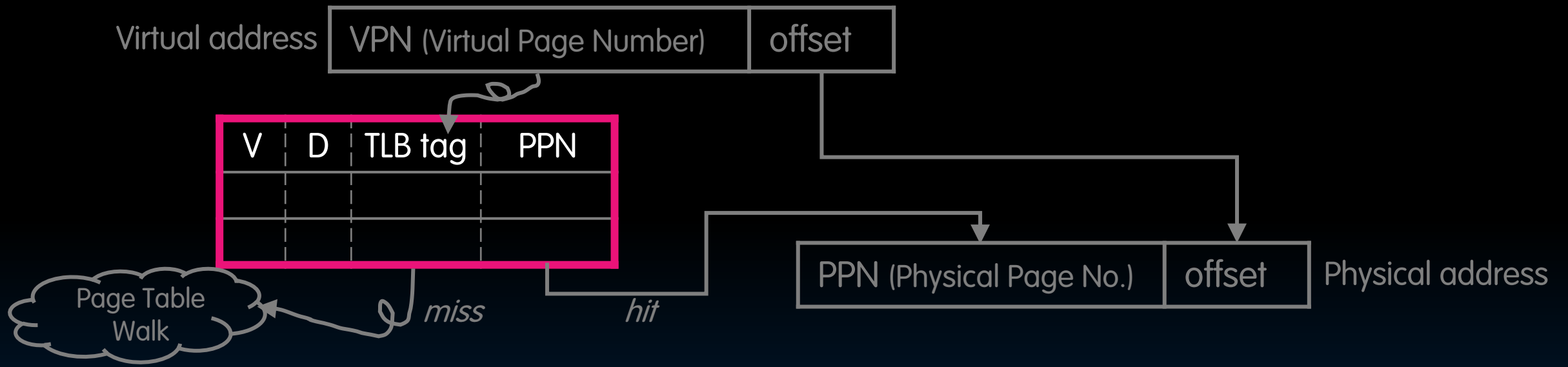
The TLB Is a Cache for Address Translations

- **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.

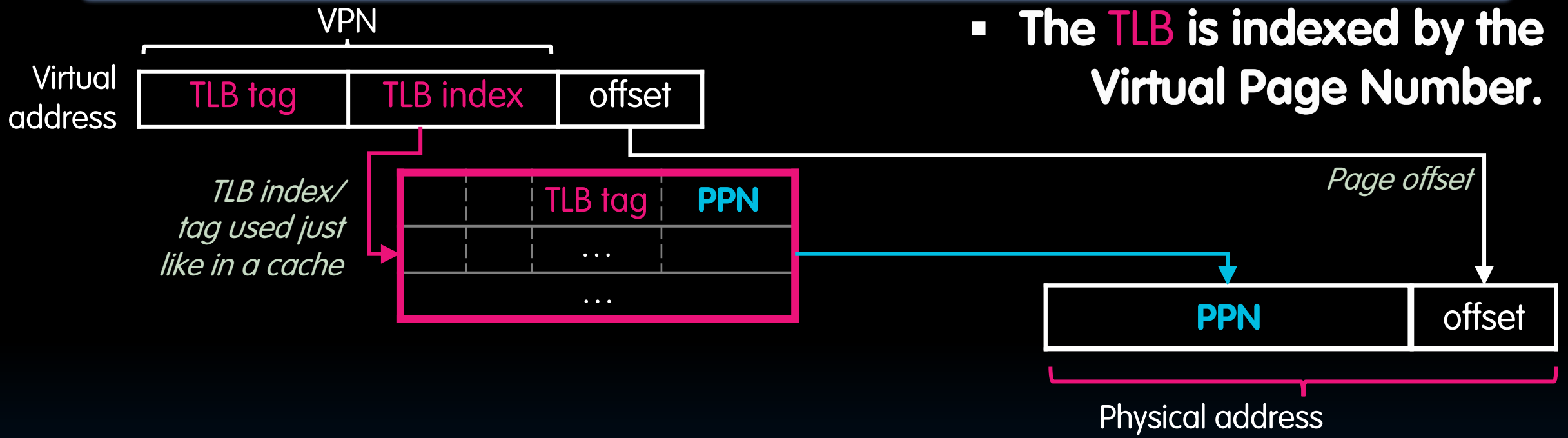


- TLB Reach: **Size of largest virtual address space that can be simultaneously mapped by the TLB.**
- **TLB design: 38-128 entries.**
  - Typically *fully associative* (increase TLB reach by minimizing conflicting entries).
  - Random/FIFO replacement policy.

Tag, Index, and Offset

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

- The TLB is indexed by the Virtual Page Number.



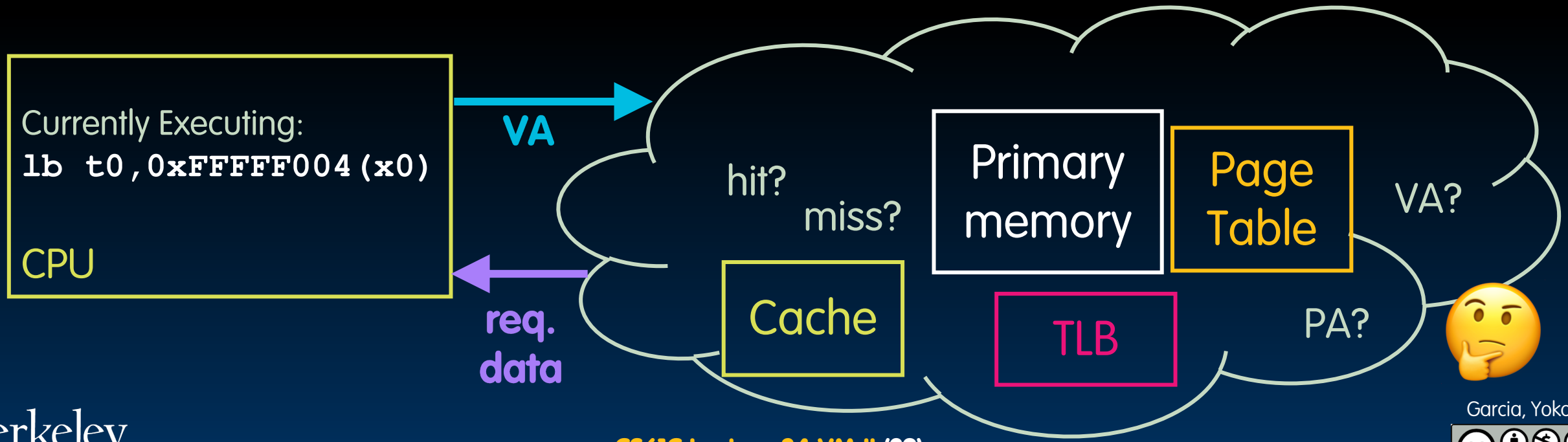
Tag, Index, and Offset

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.

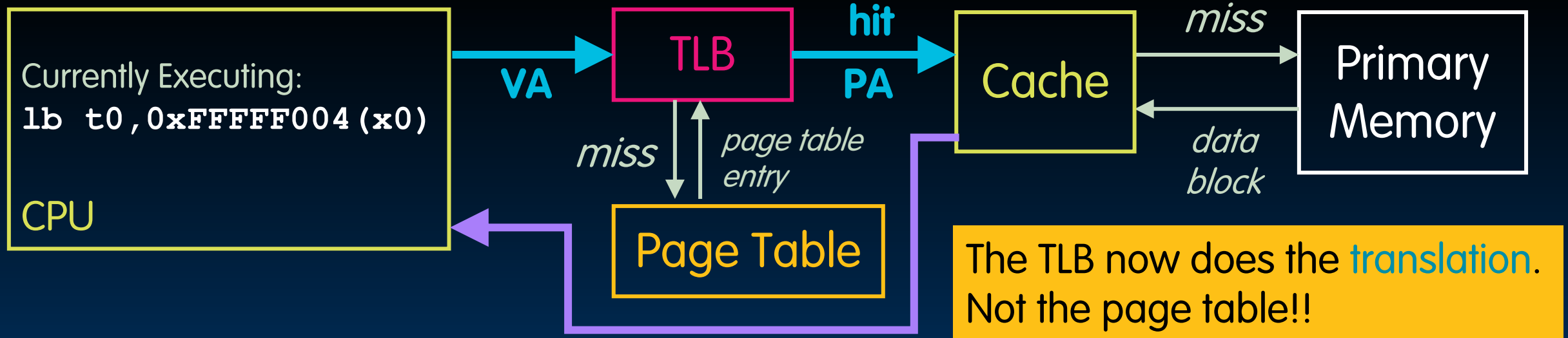
Memory Access: TLB, Cache, DRAM, Page Table?

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.



The TLB now does the translation. Not the page table!!

Virtual Memory Summary

- **Virtual Memory is a wonderful abstraction!**
  - Run multiple processes, each with independent protected 4GiB memory
  - Have physical memory smaller than or bigger than virtual memory
  - Page tables live in memory, time and space expensive
  - Valid (in memory vs on disk) and Dirty status bits (write back!)
- **Trap Handler runs in Supervisor Mode**
  - Handles exceptions (sync, e.g., page faults) or interrupts (async)
  - Swaps pages out on page fault with context switch so no lost time
- **Caches are caches for memory, Memory is cache for disk**
- **TLB is a cache of the page table**
  - Most recent page table entries stored
- **Cache is physically addressed (uses PA not VA)**
- **Want to make your computer faster?**
  - Buy more memory! (can run more programs "resident", avoid going to Pluto)