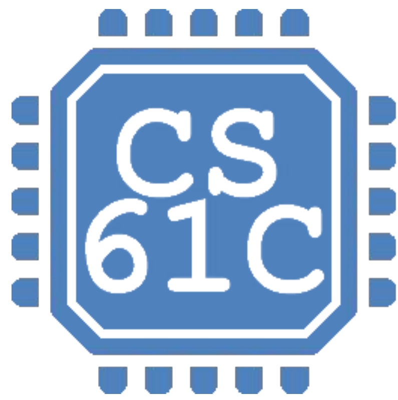


Great Ideas in Computer Architecture

Virtual Memory

Instructor: Nick Riasanovsky

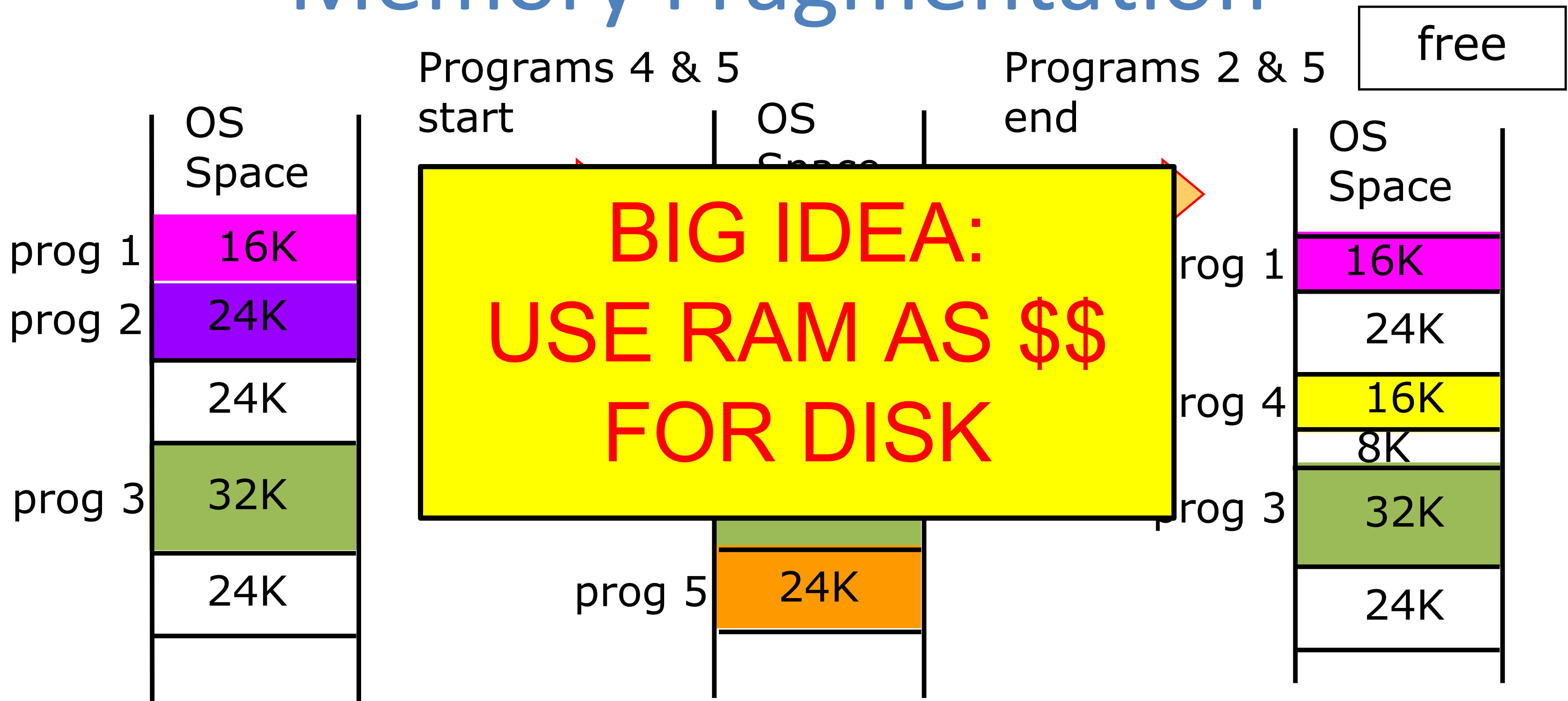
Review of Last Lecture

- The role of the Operating System
  - Booting a computer: BIOS, bootloader, OS boot, initialization
- Base and bounds for multiple processes
  - Simple, but doesn’t give us everything we want
- Virtual memory bridges memory and disk
  - Provides illusion of independent address spaces to processes and protects them from each other

Agenda

- Virtual Memory and Page Tables
- Administrivia
- Translation Lookaside Buffer (TLB)
- VM Performance
- VM Wrap-up

Memory Fragmentation



What if we want to run process 6, and we need 32K of space?

As programs start and end, memory becomes fragmented. Therefore, if we ever need a large chunk of memory, programs will need to be moved.

Question: What kind of cache should memory be?

(A) Direct Mapped, write back/write allocate

(B) Direct Mapped, write through/no write allocate

(C) Fully Associative, write back/write allocate

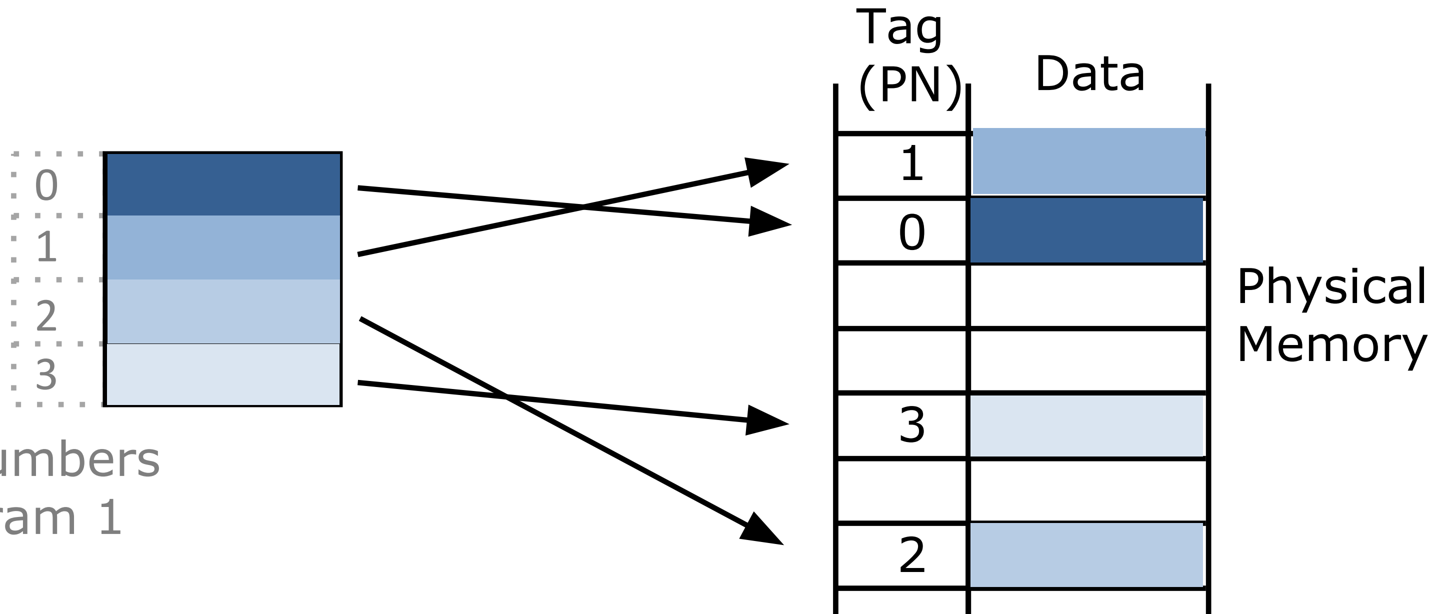
(D) Fully Associative, write thorugh/no write allocate

Page Number : Offset

- Program's view of memory can be broken up into pages by splitting processor issued addresses into:

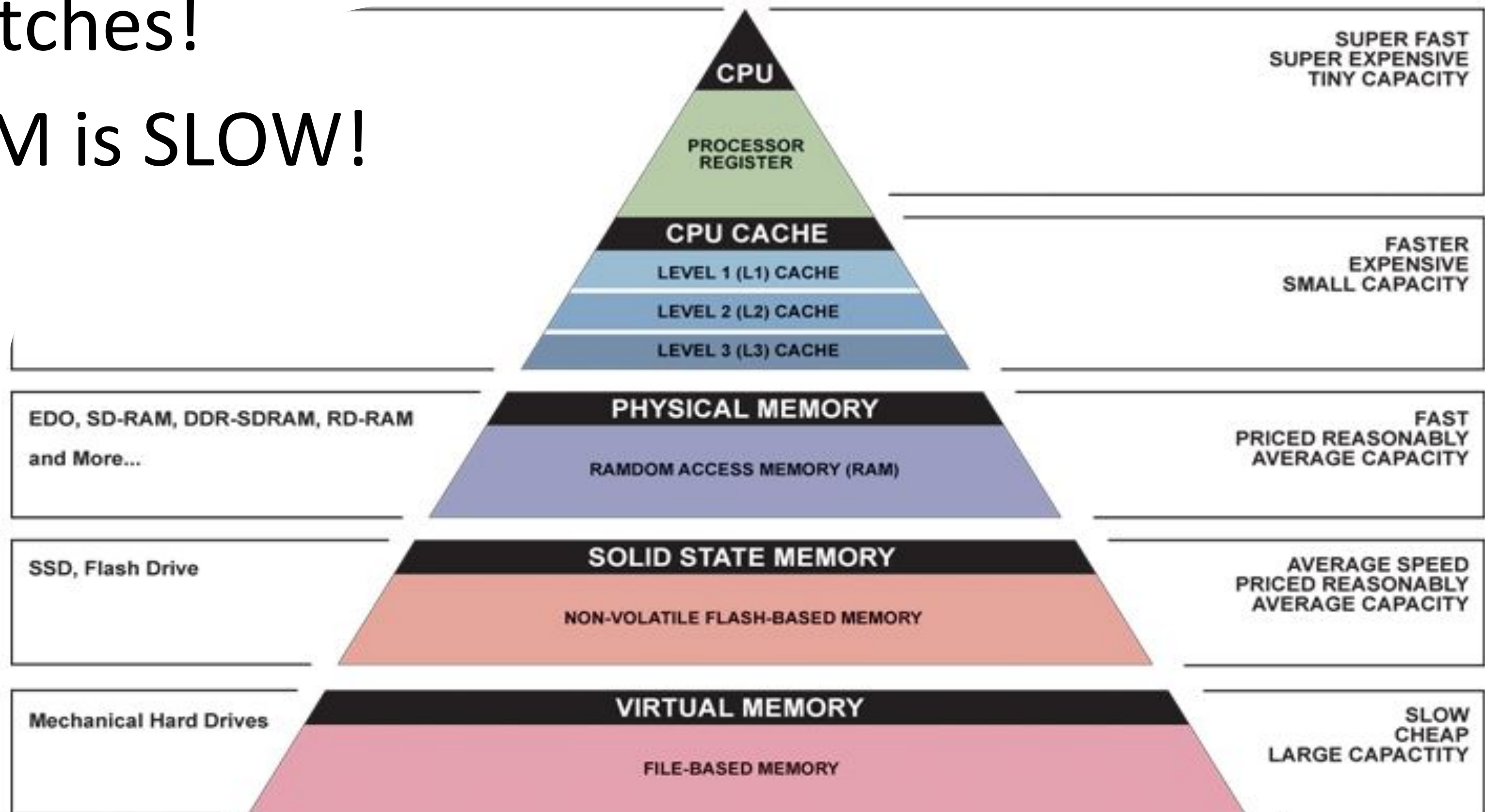
Memory as a Cache

- Allows storage of programs pages non-contiguously!



Issue: Associative Caches are SLOW

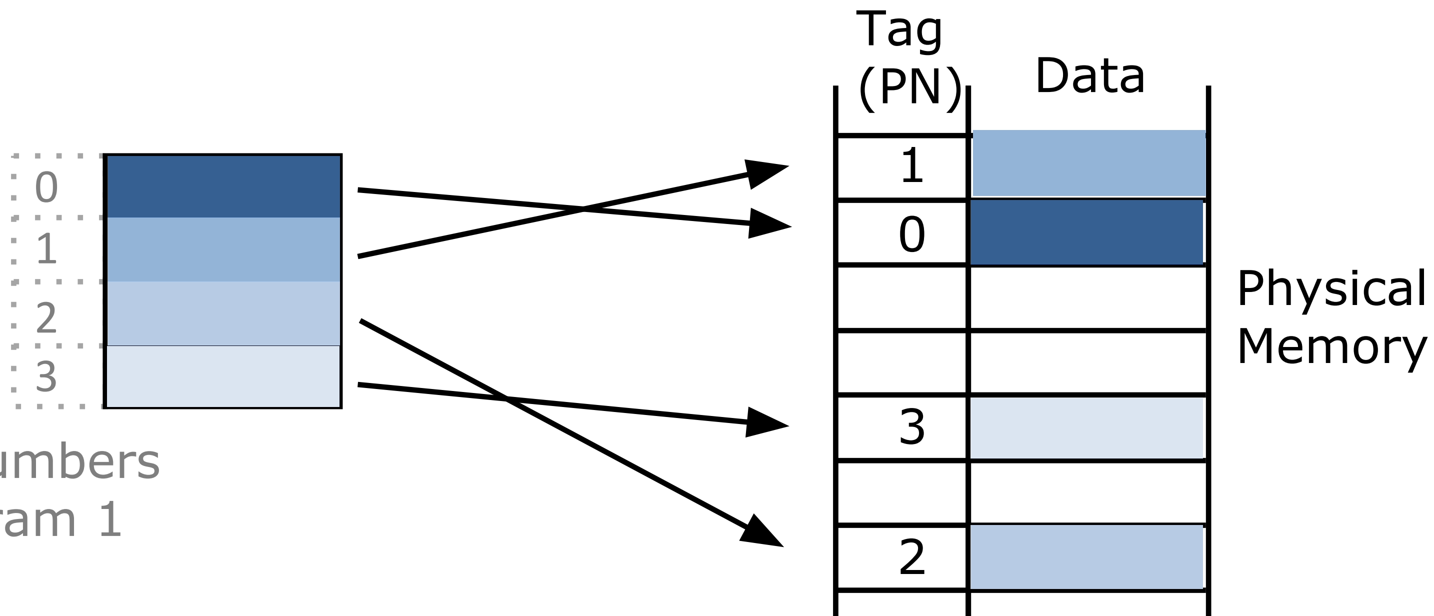
- Must check every entry to see if the tag matches!
- RAM is SLOW!



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Memory as a Cache

- Allows storage of programs pages non-contiguously!



Solution: Page Table

- Program's view of memory can be broken up into pages by splitting processor issued addresses into:

- A page table will enable translation from virtual to physical page

Virtual and Physical Page Numbers

VPN: like tag: responds to size of VIRTUAL memory

PPN: like block number: responds to size of PHYSICAL memory

Virtual Address

$\log_2$(Virtual Memory Size/Page Size)
= Virtual Address Bits – Offset Bits

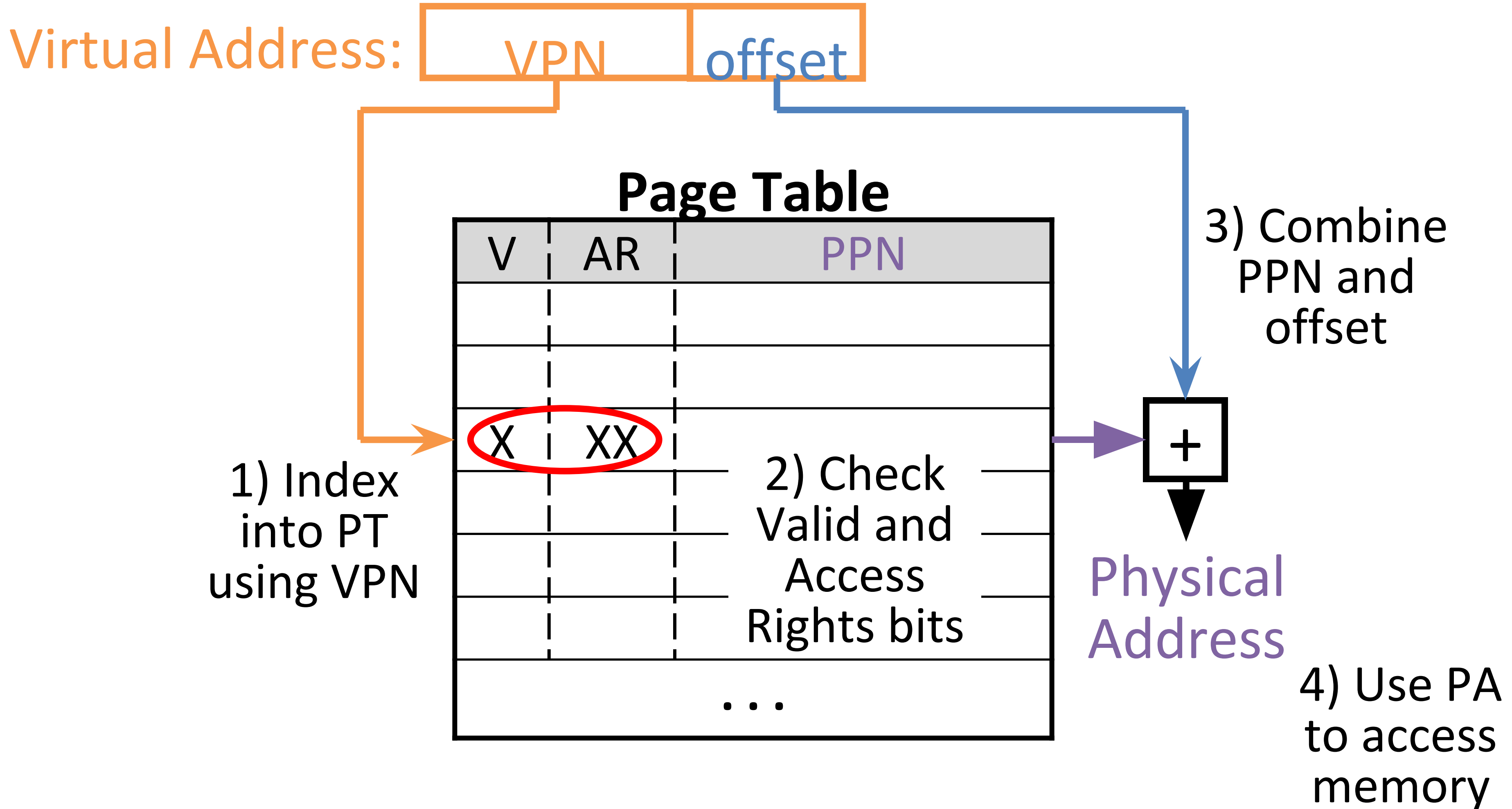
Physical Address

$\log_2$(Physical Memory Size/Page Size)
= Physical Address Bits – Offset Bits

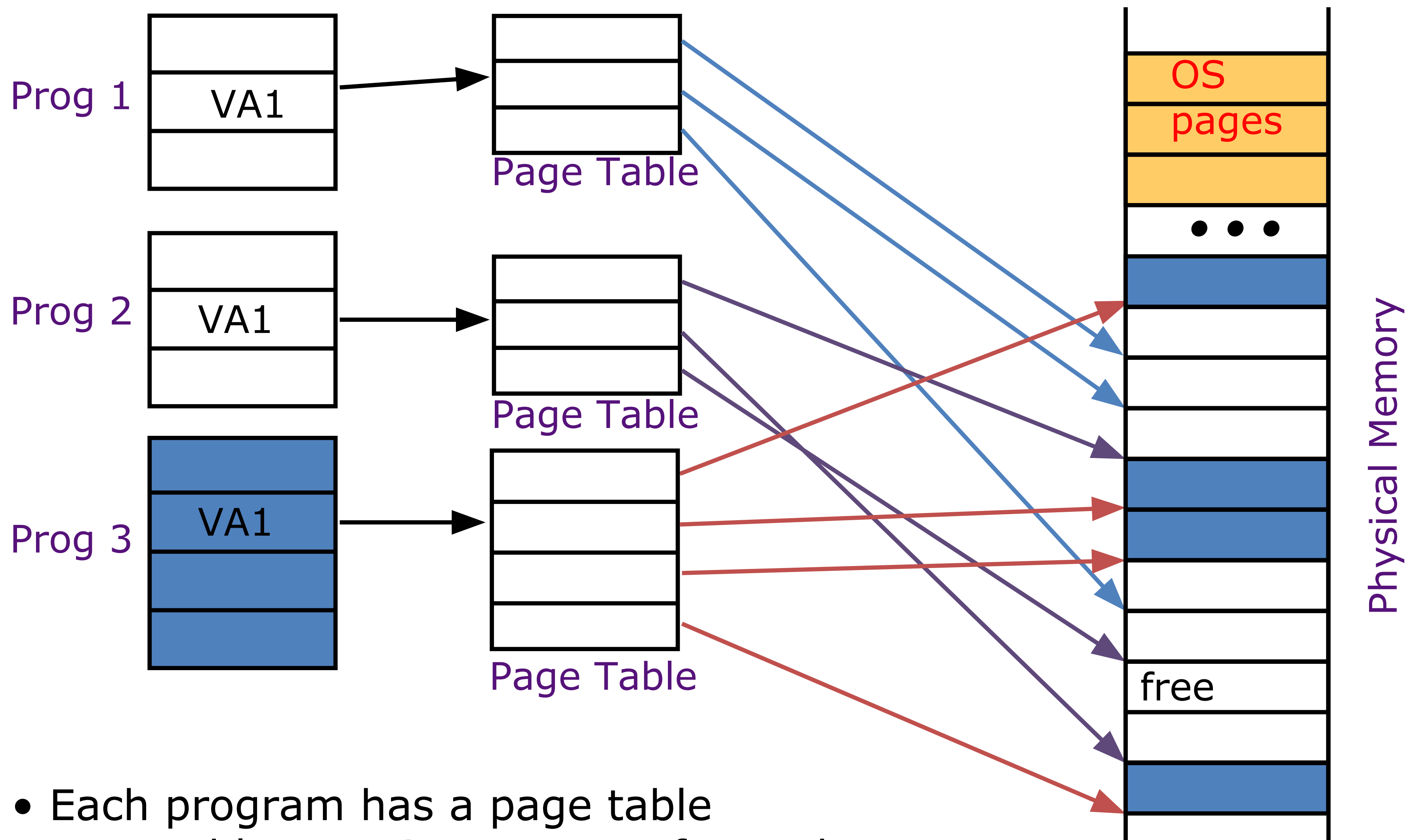
Page Table Entry Format

- Contains either PPN or indication not in main memory
- Valid = Valid page table entry
  - 1 → virtual page is in physical memory
  - 0 → OS needs to fetch page from disk
- Access Rights checked on every access to see if allowed (provides protection)
  - *Read*
  - *Write*
  - *Executable:* Can fetch instructions from page

Page Table Layout



Private Address Space per User



- Each program has a page table
- Page table contains an entry for each program page

Protection Between Processes

- With a bare system, addresses issued with loads/stores are real physical addresses
- This means any program can issue any address, therefore can access any part of memory, even areas which it doesn't own
  - Example: the OS data structures
- We should send all addresses through a mechanism that the OS controls, before they make it out to DRAM - *a translation mechanism*

How Big is the Page Table?

- 64 MiB RAM
- 32-bit virtual address space
- 1 KiB pages

Offset Bits = log_2 (1024) = 10

Virtual Page Bits = 32 - 10 = 22
(2^22 entries in the page table!)

Physical Page Bits = log_2 (2^26) - 10

Physical Page Bits = 26 - 10 = 16 (2 B)

Total Bytes = 2^22 * 2 = 2^23

Number of pages = 2^23 / 2^10 = 2^13 PAGES

(A) Less than 1 Page

(B) Less than 100 Pages

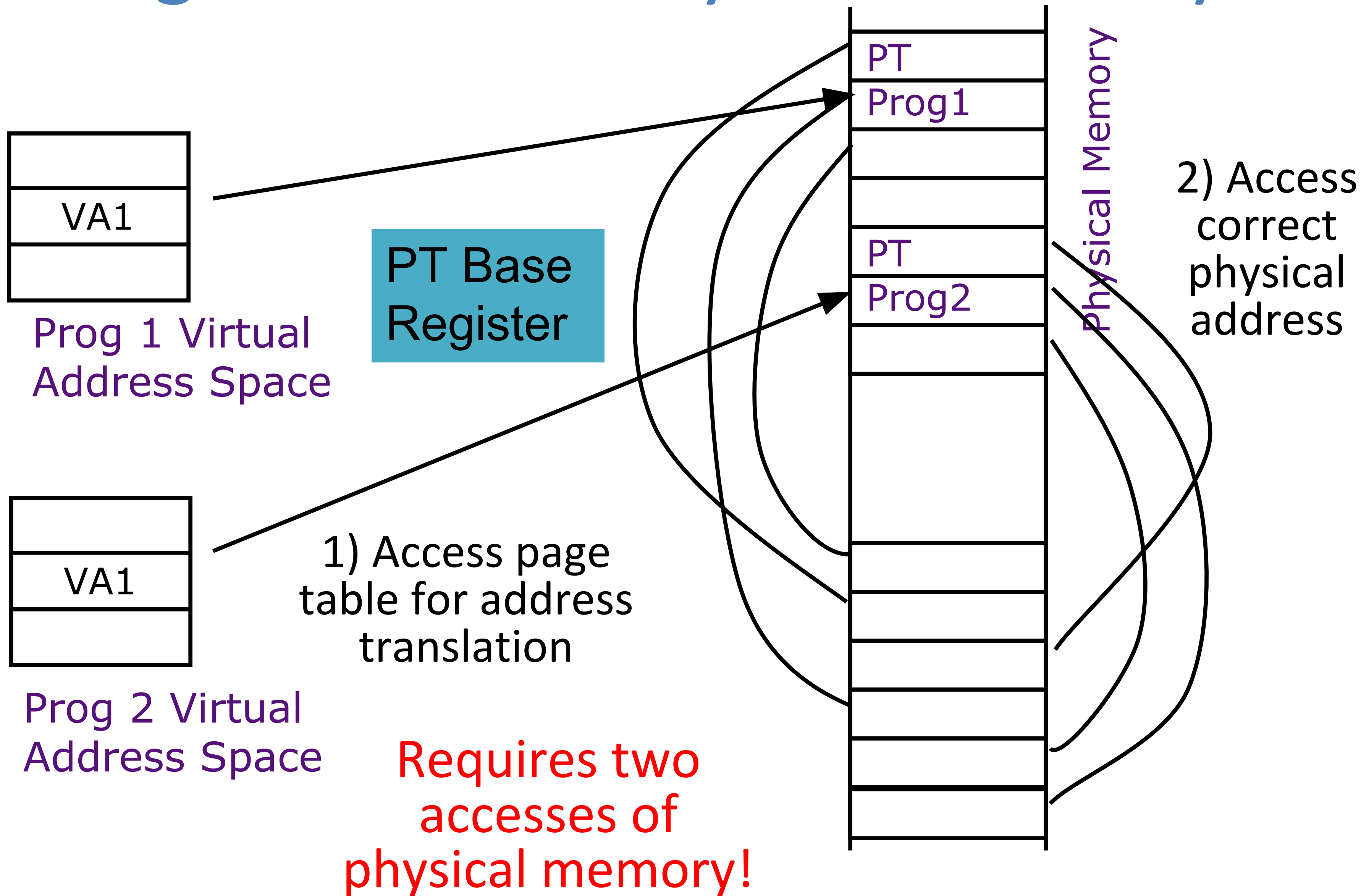
(C) Less than 1000 Pages

(D) **More than 1000 Pages**

Where Should Page Tables Reside?

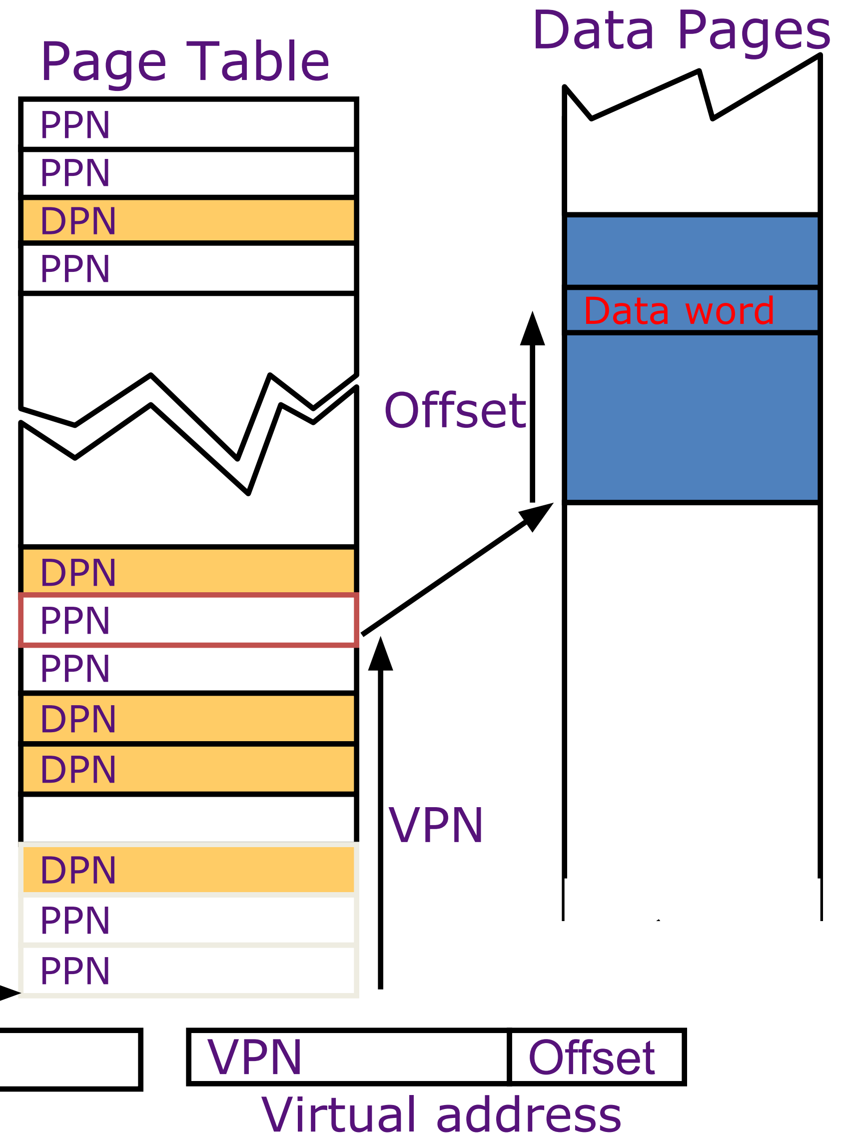
- Space required by the page tables (PT) is proportional to the address space, number of users, ...
  - ⇒ *Too large to keep in registers, or caches….*
- Idea: Keep PTs in the main memory
  - How can we find the page table in memory if the page table is how we learn Physical addresses?????
  - PT Base Register: stores Physical Address of current Page Table

Page Tables in Physical Memory

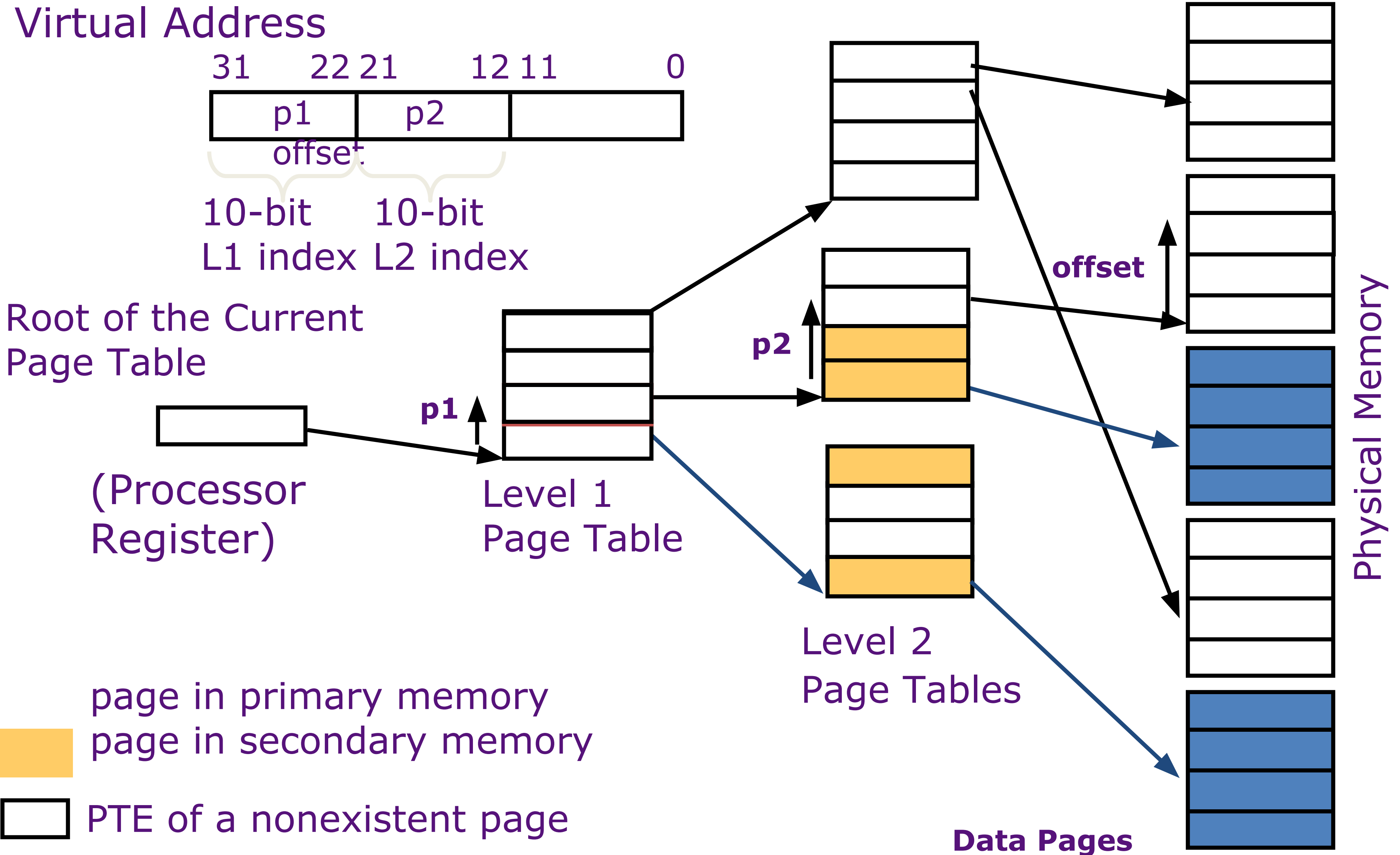


Linear Page Table

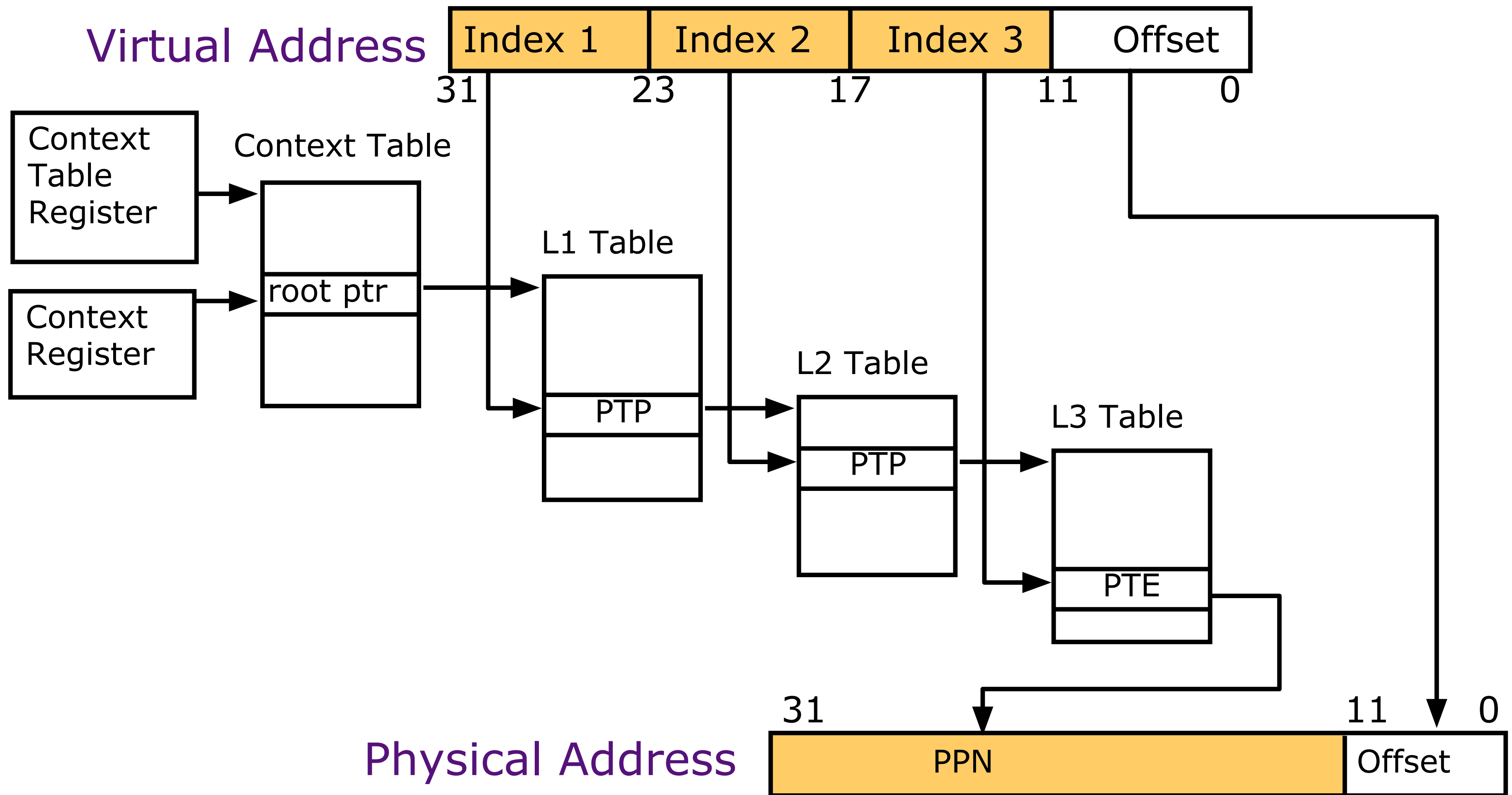
- Page Table Entry (PTE) contains:
  - 1 bit to indicate if page exists
  - And either PPN or DPN:
    - PPN (physical page number) for a memory-resident page
    - DPN (disk page number) for a page on the disk
  - Status bits for protection and usage (read, write, exec)
- OS sets the Page Table Base Register whenever active user process changes



Hierarchical Page Table

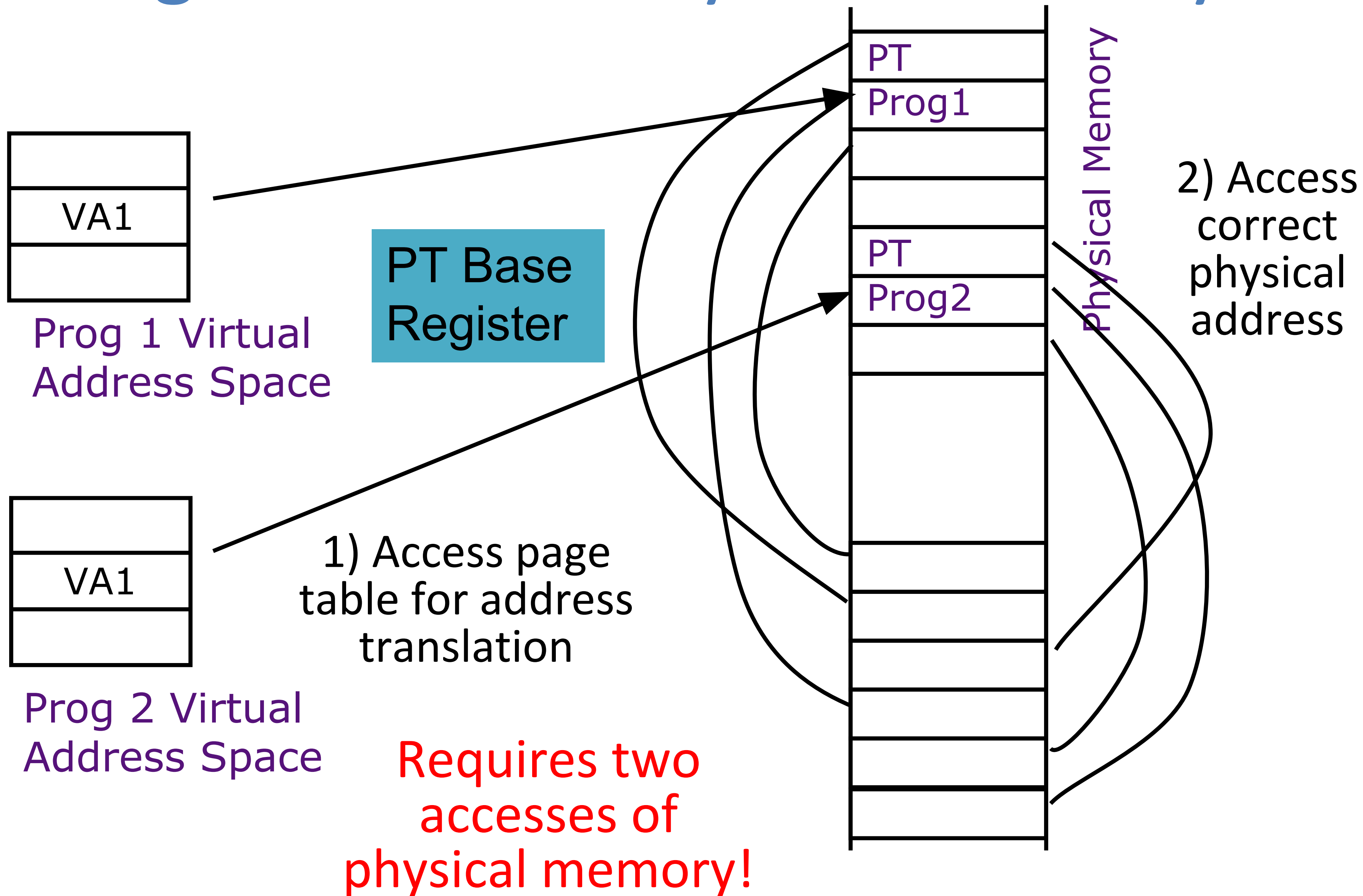


Hierarchical Page Table Walk: SPARC v8

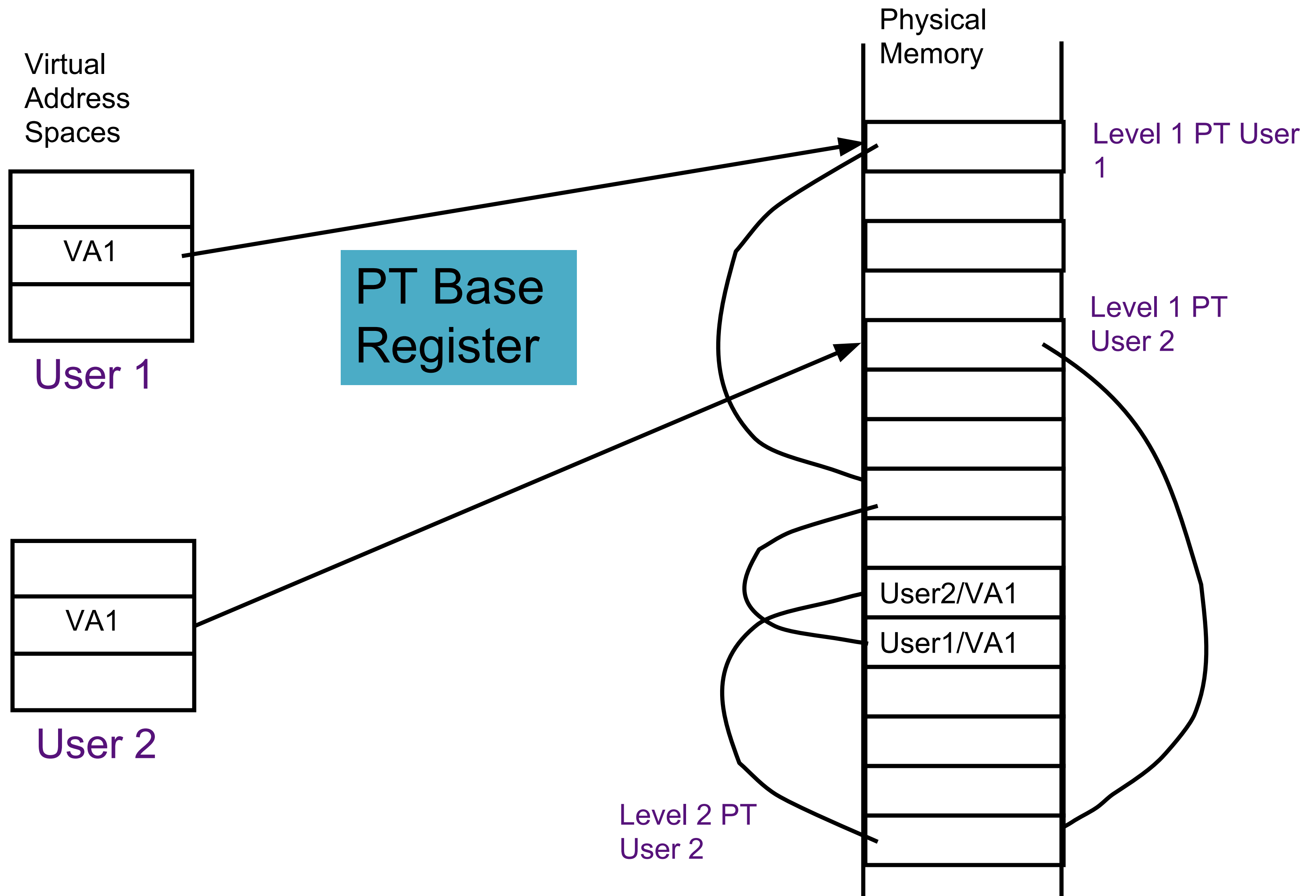


MMU does this table walk in hardware on a TLB miss

Page Tables in Physical Memory



Two-Level Page Tables in Physical Memory



Agenda

- Virtual Memory and Page Tables
- Administrivia
- Translation Lookaside Buffer (TLB)
- VM Performance
- VM Wrap-up

Administrivia

- Proj4 due on Friday (8/03)
  - Hold off on submissions for now
- HW7 released
- Guerilla Session on Wed. @Cory 540AB, 4-6p
- Regrade requests are open for MT2 until Friday
- The final will be 8/09 7-10PM @VLSB 2040/2060!

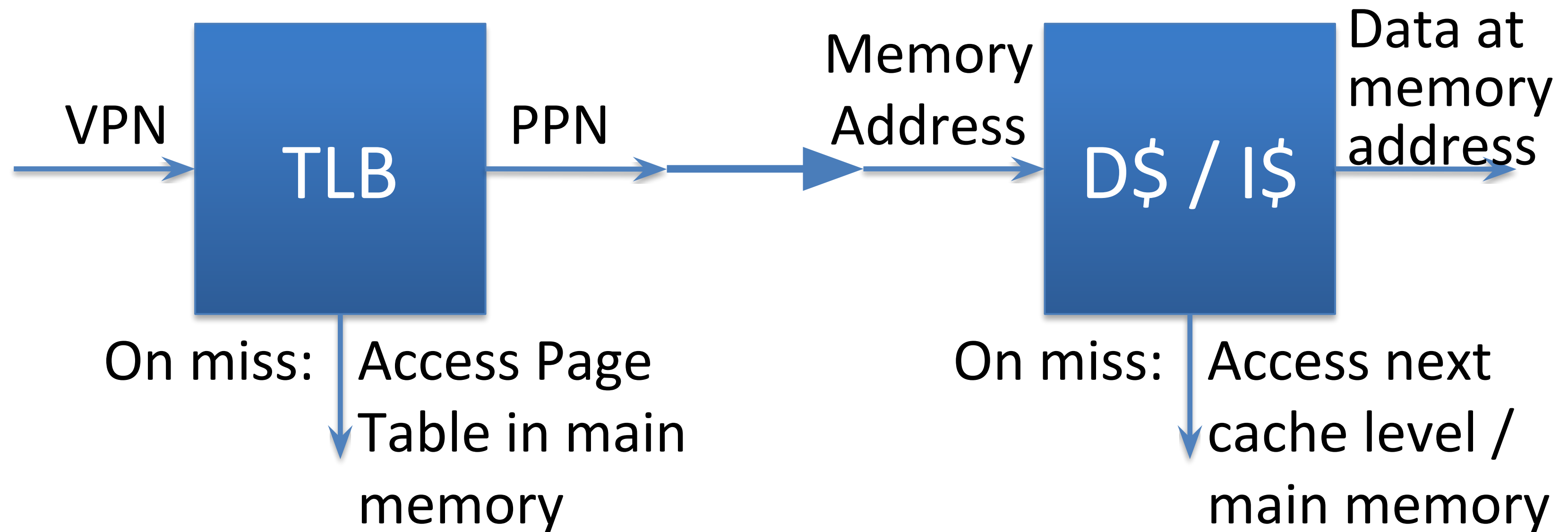
Agenda

- Virtual Memory
- Page Tables
- Administrivia
- Translation Lookaside Buffer (TLB)
- VM Performance
- VM Wrap-up

Virtual Memory Problem

- 2 physical memory accesses per data access = SLOW!
- Since locality in pages of data, there must be locality in the translations of those pages
- **Build a separate cache for the Page Table**
  - For historical reasons, cache is called a *Translation Lookaside Buffer (TLB)*
  - Notice that what is stored in the TLB is NOT data, but the VPN $\longrightarrow$ PPN mapping translations

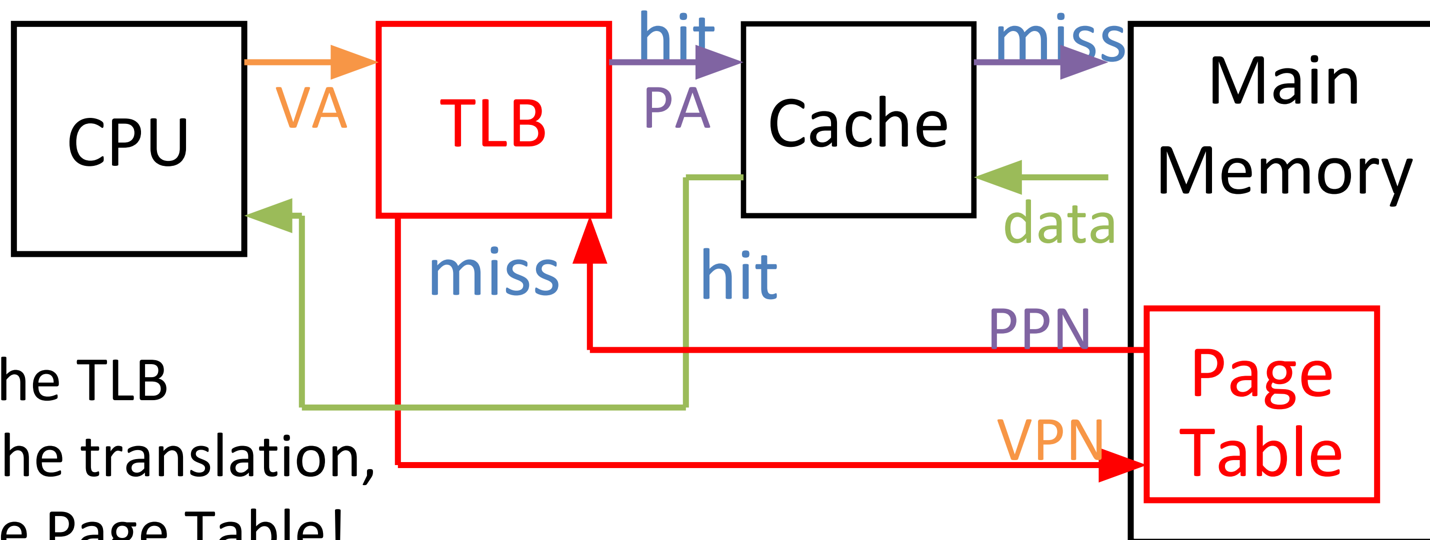
TLBs vs. Caches



- TLBs usually small, typically 32–128 entries
- TLB access time comparable to cache (much faster than accessing main memory)
- TLBs usually are fully/highly associativity

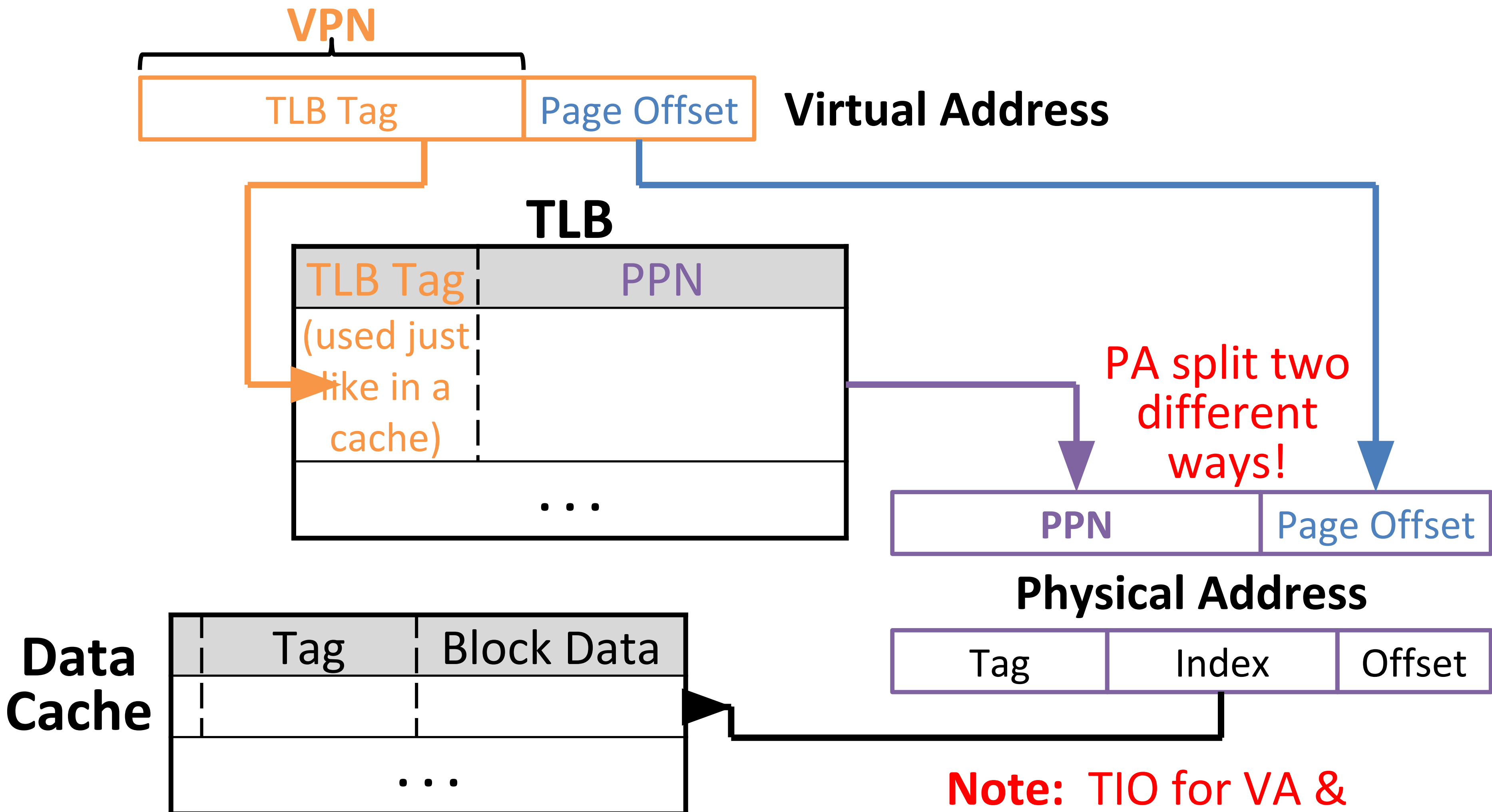
Where Are TLBs Located?

- Which should we check first: Cache or TLB?
  - Can cache hold requested data if corresponding page is not in physical memory? No – check PT first
  - With TLB first, does cache receive VA or PA?



Now the TLB does the translation, not the Page Table!

Address Translation Using TLB



Typical TLB Entry Format

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

- *Valid* whether that **TLB ENTRY** is valid (unrelated to PT)
- *Access Rights:* *Data* from the PT
- *Dirty:* Basically always use write-back, so indicates whether or not to write page to disk when replaced
- *Ref:* Used to implement LRU
  - Set when page is accessed, cleared periodically by OS
- *TLB Index:* VPN mod (# TLB sets)
- *TLB Tag:* VPN minus TLB Index (upper bits)
- *PPN: Data* from PT

Question: How many bits wide are the following?

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

First solve for the T:
O = log_2(2^14) = 14
I = log_2(512 / 2) = 8
T = 40 - 14 - 8 = 18

Now solve for the size of a TLB entry.
Total Bits = 5 + 18 + PPN bits
PPN bits = log_2(2^36 / 2^14) = 22
Total Bits = 5 + 18 + 22 = 45

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

	TLB Tag	TLB Index	TLB Entry
(A)	12	14	38
(B)	18	8	45
(C)	14	12	40
(D)	17	9	43

Fetching Data on a Memory Read

1) Check TLB (input: VPN, output: PPN)
   - *TLB Hit:* Fetch translation, return PPN
   - *TLB Miss:* Check page table (in memory)
     - *Page Table Hit:* Load page table entry into TLB
     - *Page Table Miss (Page Fault):* Fetch page from disk to memory, update corresponding page table entry, then load entry into TLB
2) Check cache (input: PPN, output: data)
   - *Cache Hit:* Return data value to processor
   - *Cache Miss:* Fetch data value from memory, store it in cache, return it to processor

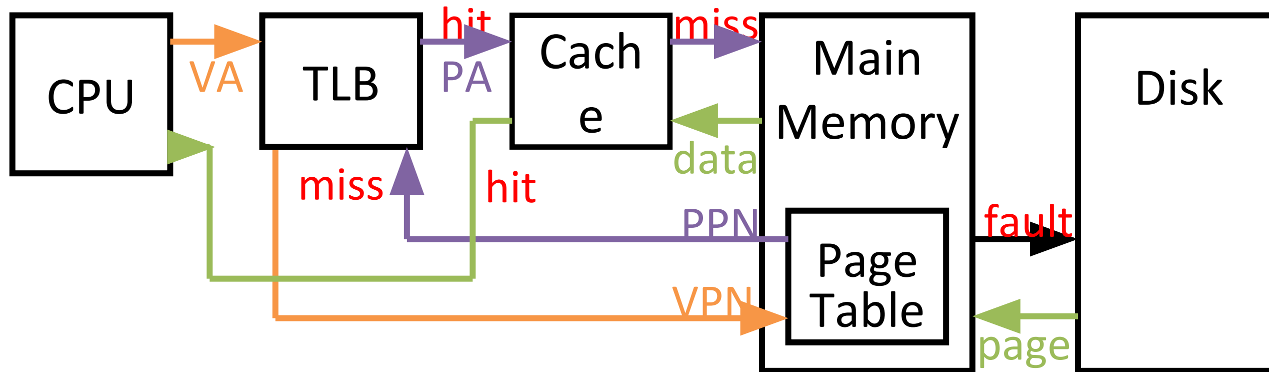
Page Faults

- Load the page off the disk into a free page of memory
  - Switch to some other process while we wait
- Interrupt thrown when page loaded and the process' page table is updated
  - When we switch back to the task, the desired data will be in memory
- If memory full, replace page (LRU), writing back if necessary, and update *both* page table entries
  - Continuous swapping between disk and memory called "thrashing"

Performance Metrics

- VM performance also uses Hit/Miss Rates and Miss Penalties
  - *TLB Miss Rate:* Fraction of TLB accesses that result in a TLB Miss
  - *Page Table Miss Rate:* Fraction of PT accesses that result in a page fault
- Caching performance definitions remain the same
  - Somewhat independent, as TLB will always pass PA to cache regardless of TLB hit or miss

Data Fetch Scenarios



- Are the following scenarios for a single data access possible?
  - TLB Miss, Page Fault — Yes
  - TLB Hit, Page Table Hit — No
  - TLB Miss, Cache Hit — Yes
  - Page Table Hit, Cache Miss — Yes
  - Page Fault, Cache Hit — No

Beat the Staff

- You have met the staff, but now it's your chance to beat them
- The staff is working on project 4 too, trying to get the fastest speedup they can
- We have a competition for the students with the best speedups, but we will have an additional reward for any students who Beat the Staff
- More details to come in a piazza post

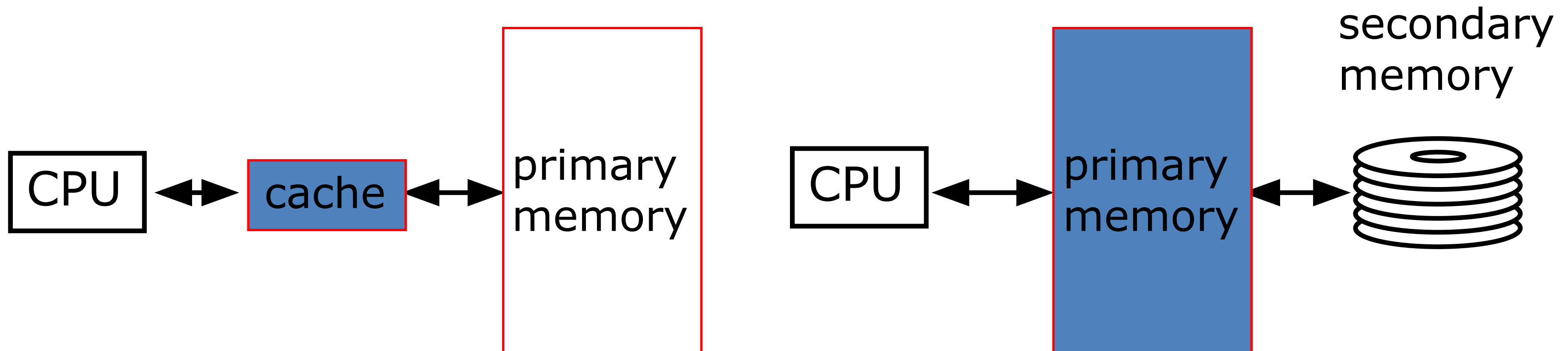
Agenda

- Virtual Memory
- Page Tables
- Administrivia
- Translation Lookaside Buffer (TLB)
- VM Performance
- VM Wrap-up

VM Performance

- Virtual Memory is the level of the memory hierarchy that sits *below* main memory
  - TLB comes *before* cache, but affects transfer of data from disk to main memory
  - Previously we assumed main memory was lowest level, now we just have to account for disk accesses
- Same CPI, AMAT equations apply, but now treat main memory like a mid-level cache

Typical Performance Stats



Caching	*Demand paging*
cache entry	page frame
cache block (≈32 bytes)	page (≈4 Ki bytes)
cache miss rate (1% to 20%)	page miss rate (<0.001%)
cache hit (≈1 cycle)	page hit (≈100 cycles)
cache miss (≈100 cycles)	page fault (≈5M cycles)

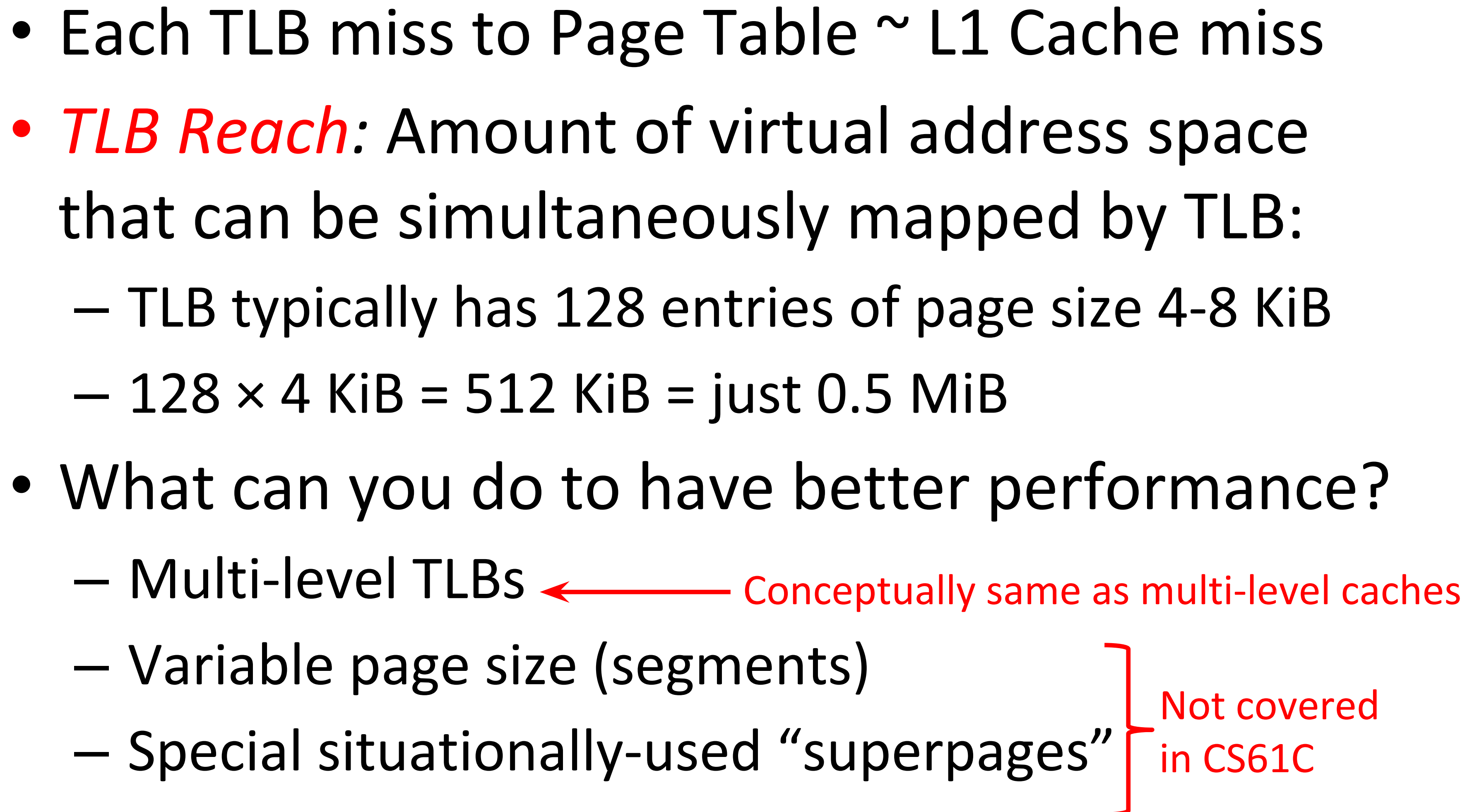
Impact of Paging on AMAT (1/2)

- Memory Parameters:
  - L1 cache hit = 1 clock cycles, hit 95% of accesses
  - L2 cache hit = 10 clock cycles, hit 60% of L1 misses
  - DRAM = 200 clock cycles (≈100 nanoseconds)
  - Disk = 20,000,000 clock cycles (≈10 milliseconds)
- Average Memory Access Time (no paging):
  - $1 + 5\% \times 10 + 5\% \times 40\% \times 200 = 5.5$ clock cycles
- Average Memory Access Time (with paging):
  - 5.5 (AMAT with no paging) + ?

Impact of Paging on AMAT (2/2)

- Average Memory Access Time (with paging) =
  - $5.5 + 5\% \times 40\% \times (1-HR_{Mem}) \times 20{,}000{,}000$
- AMAT if $HR_{Mem} = 99\%$?
  - $5.5 + 0.02 \times 0.01 \times 20{,}000{,}000 = 4005.5$ (≈728x slower)
  - 1 in 20,000 memory accesses goes to disk: 10 sec program takes 2 hours!
- AMAT if $HR_{Mem} = 99.9\%$?
  - $5.5 + 0.02 \times 0.001 \times 20{,}000{,}000 = 405.5$
- AMAT if $HR_{Mem} = 99.9999\%$
  - $5.5 + 0.02 \times 0.000001 \times 20{,}000{,}000 = 5.9$

Impact of TLBs on Performance



Agenda

- Virtual Memory
- Page Tables
- Administrivia
- Translation Lookaside Buffer (TLB)
- VM Performance
- VM Wrap-up

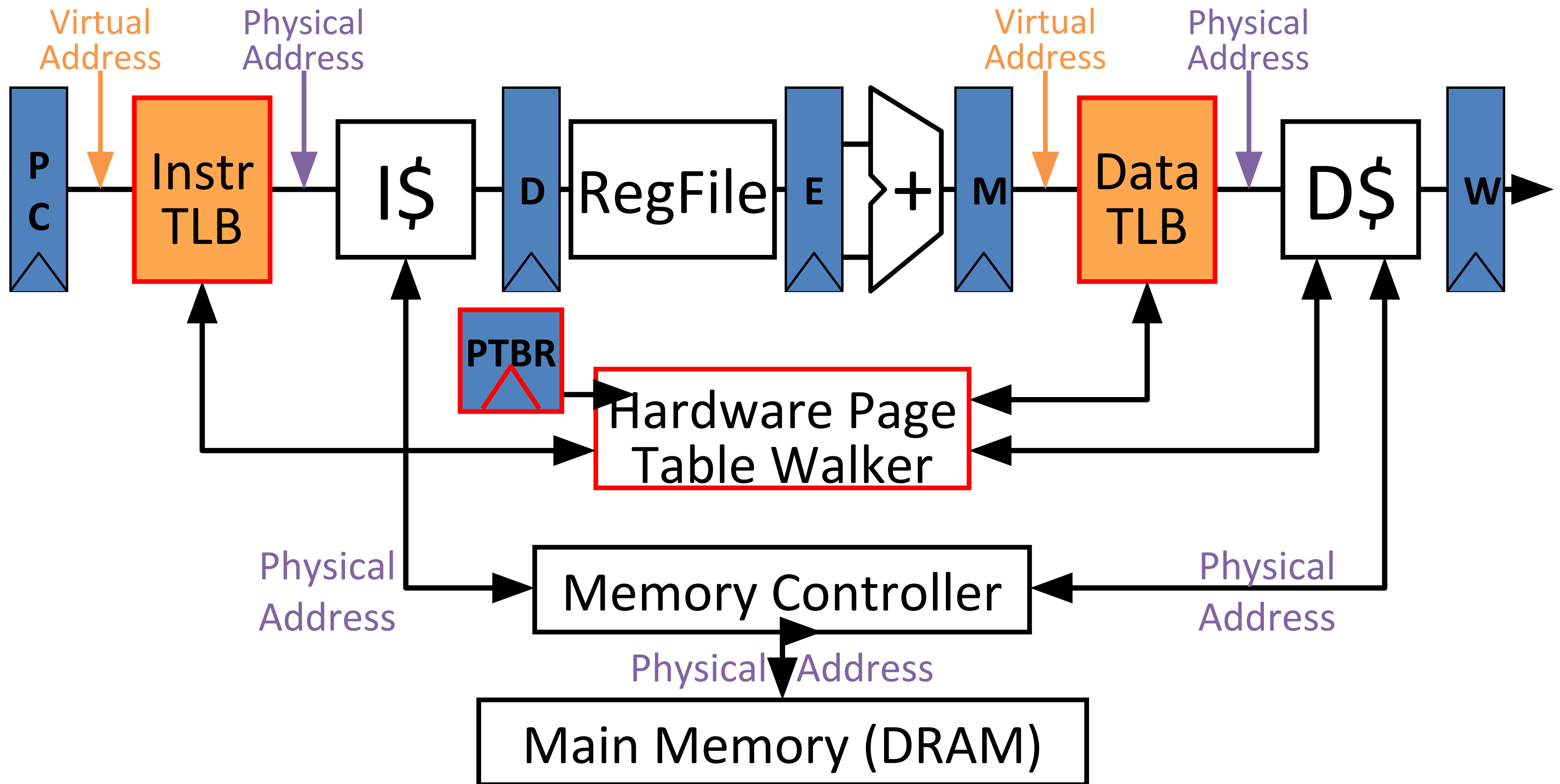
Hardware/Software Support for Memory Protection

- Different tasks can share parts of their virtual address spaces
  - But need to protect against errant access
  - Requires OS assistance
- Hardware support for OS protection
  - Privileged supervisor mode (a.k.a. *kernel mode*)
  - Privileged instructions
  - Page tables and other state information only accessible in supervisor mode
  - System call exception (e.g. `ecall` in RISC-V)

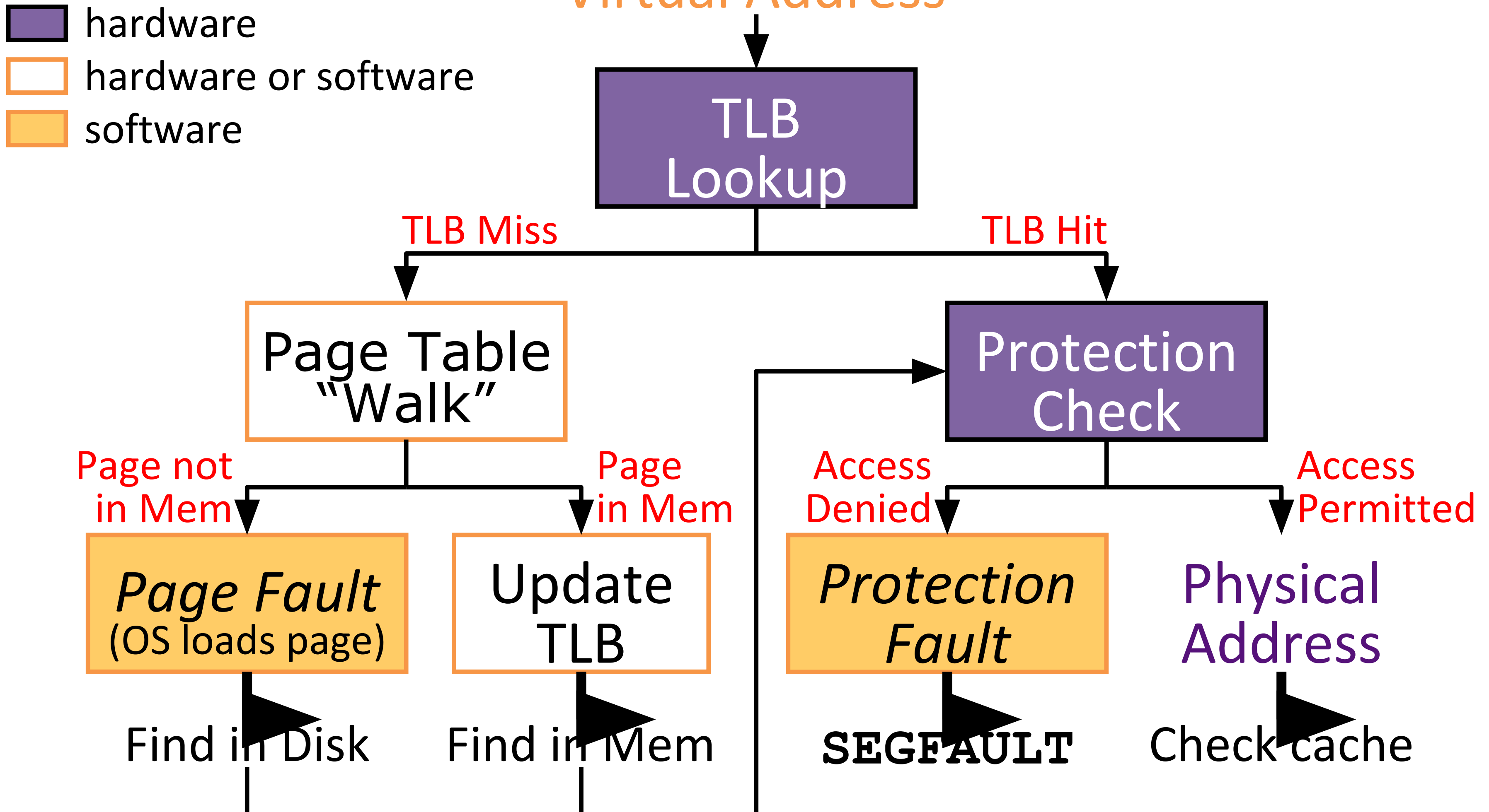
Context Switching

- How does a single processor run many programs at once?
- *Context switch:* Changing of internal state of processor (switching between processes)
  - Save register values (and PC) and change value in Page Table Base register
- What happens to the TLB?
  - Current entries are for a different process (similar VAs, though!)
  - Set all entries to invalid on context switch

Page-Based Virtual-Memory Machine



Address Translation



Summary

- User program view:
  - Contiguous memory
  - Start from some set VA
  - “Infinitely” large
  - Is the only running program
- Reality:
  - Non-contiguous memory
  - Start wherever available memory is
  - Finite size
  - Many programs running simultaneously
- Virtual memory provides:
  - Illusion of contiguous memory
  - All programs starting at same set address
  - Illusion of ~ infinite memory (2^{32} or 2^{64} bytes)
  - Protection, Sharing
- Implementation:
  - Divide memory into chunks (pages)
  - OS controls page table that maps virtual into physical addresses
  - memory as a cache for disk
  - TLB is a cache for the page table