

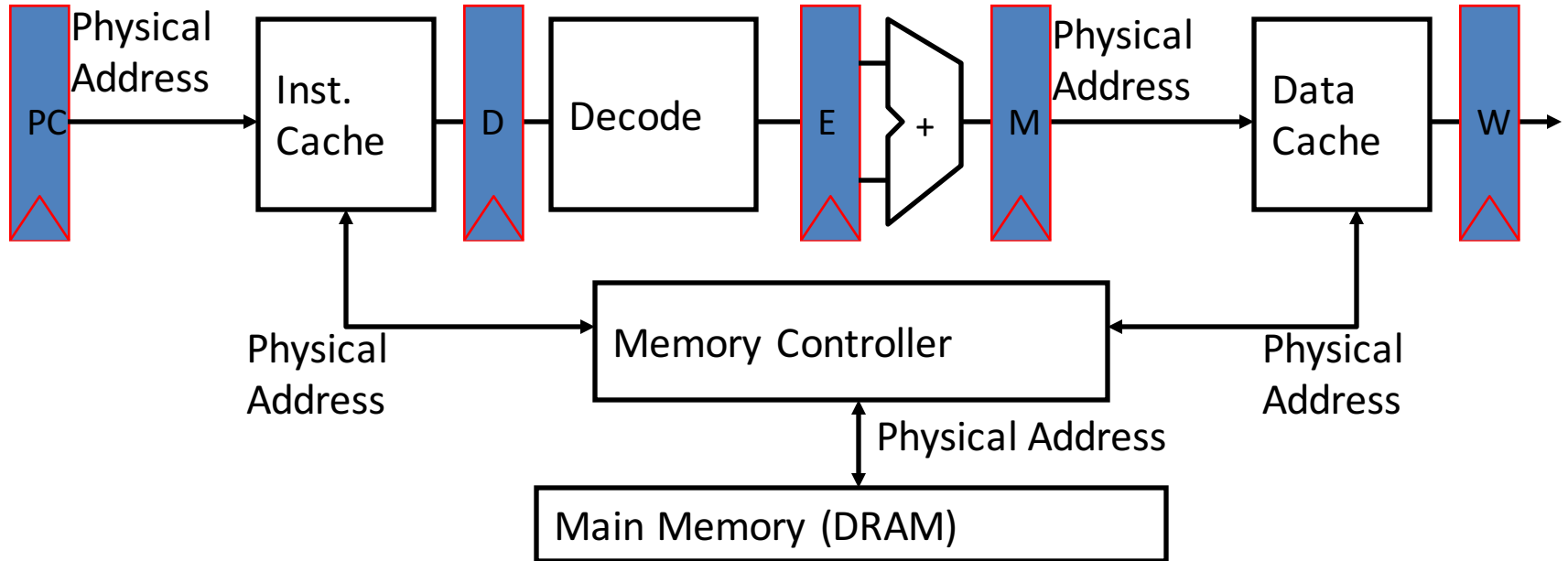
CS 61C:
Great Ideas in Computer Architecture
Virtual Memory Cont.

Instructors:

Vladimir Stojanovic & Nicholas Weaver

<http://inst.eecs.berkeley.edu/~cs61c/>

“Bare” 5-Stage Pipeline



- In a bare machine, the only kind of address is a physical address

Address Translation

- So, what do we want to achieve at the hardware level?
 - Take a Virtual Address, that points to a spot in the Virtual Address Space of a particular program, and map it to a Physical Address, which points to a physical spot in DRAM of the whole machine

Virtual Address

Virtual Page Number

Offset

Physical Address

Physical Page Number

Offset

Address Translation

Virtual Address

Virtual Page Number

Offset

Address
Translation

Copy
Bits

Physical Address

Physical Page Number

Offset

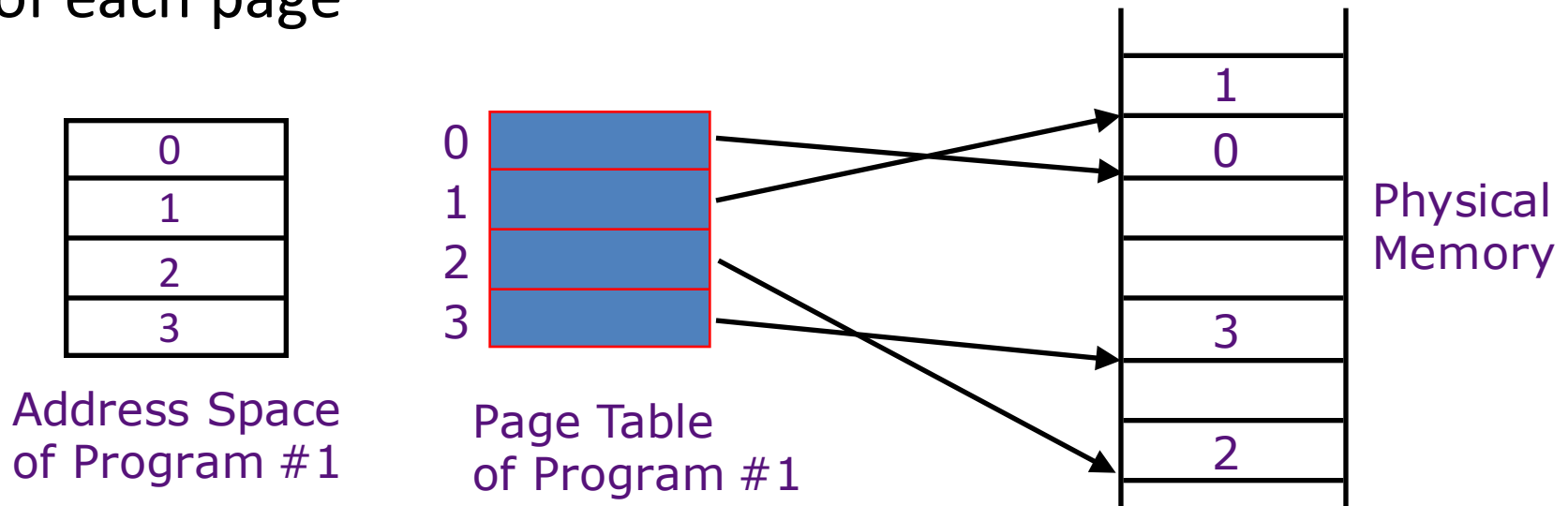
The rest of the lecture is all about implementing

Paged Memory Systems

- Processor-generated address can be split into:

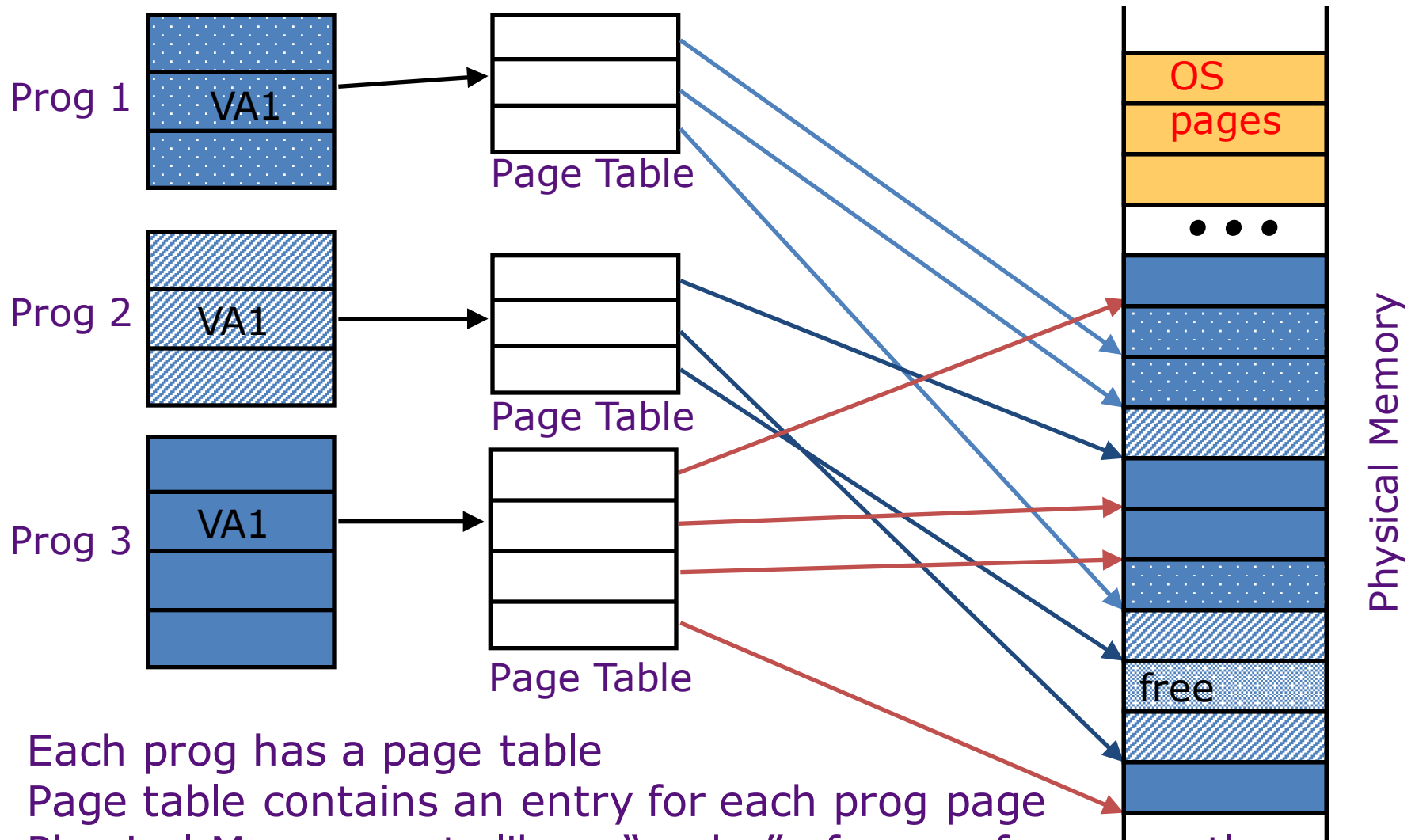


- A *page table* contains the physical address of the base of each page



Page tables make it possible to store the pages of a program non-contiguously.

Private (Virtual) Address Space per Program

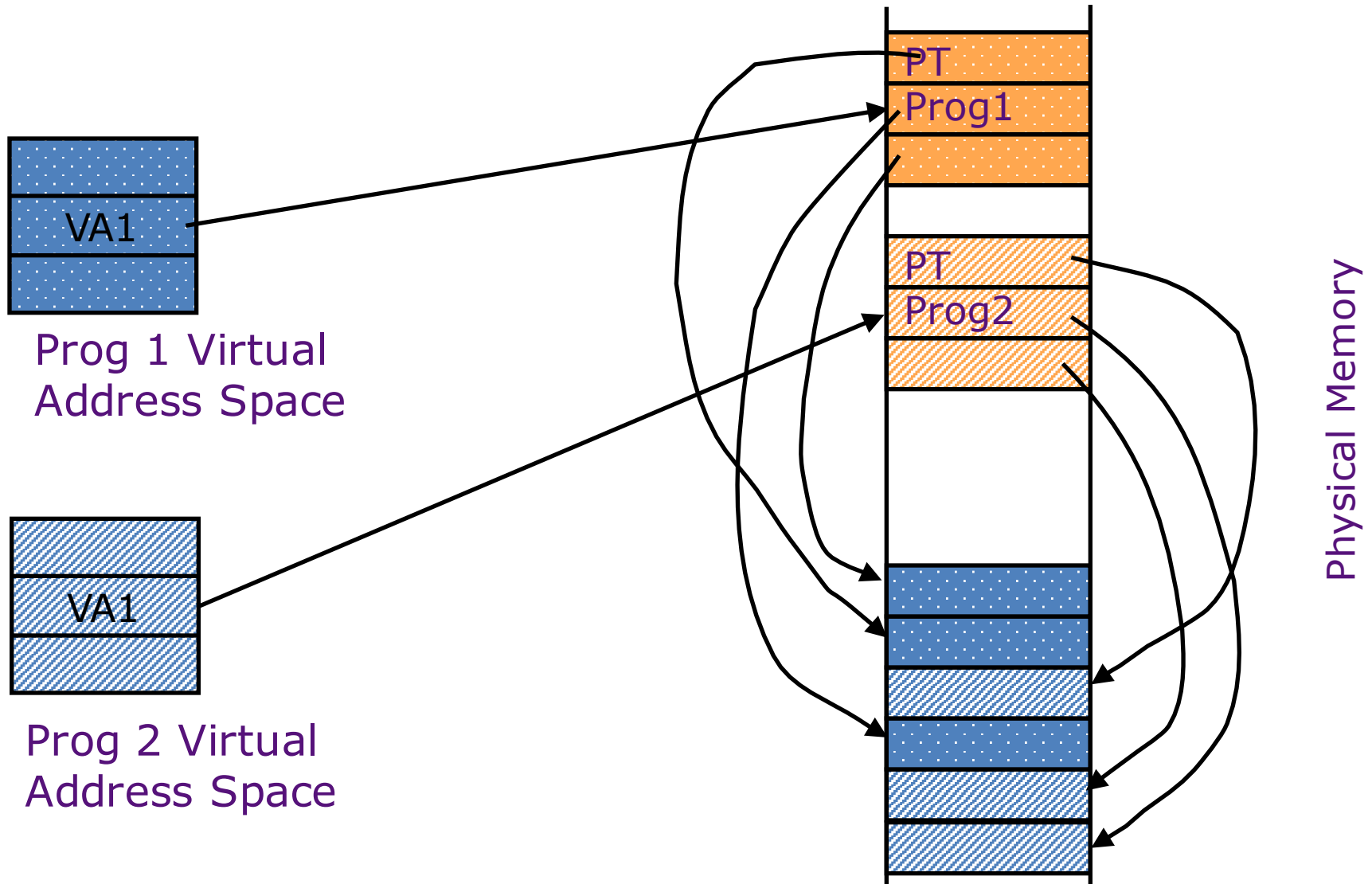


- Each prog has a page table
- Page table contains an entry for each prog page
- Physical Memory acts like a “cache” of pages for currently running programs. **Not recently used pages are stored in secondary memory, e.g. disk (in “swap partition”)**

Where Should Page Tables Reside?

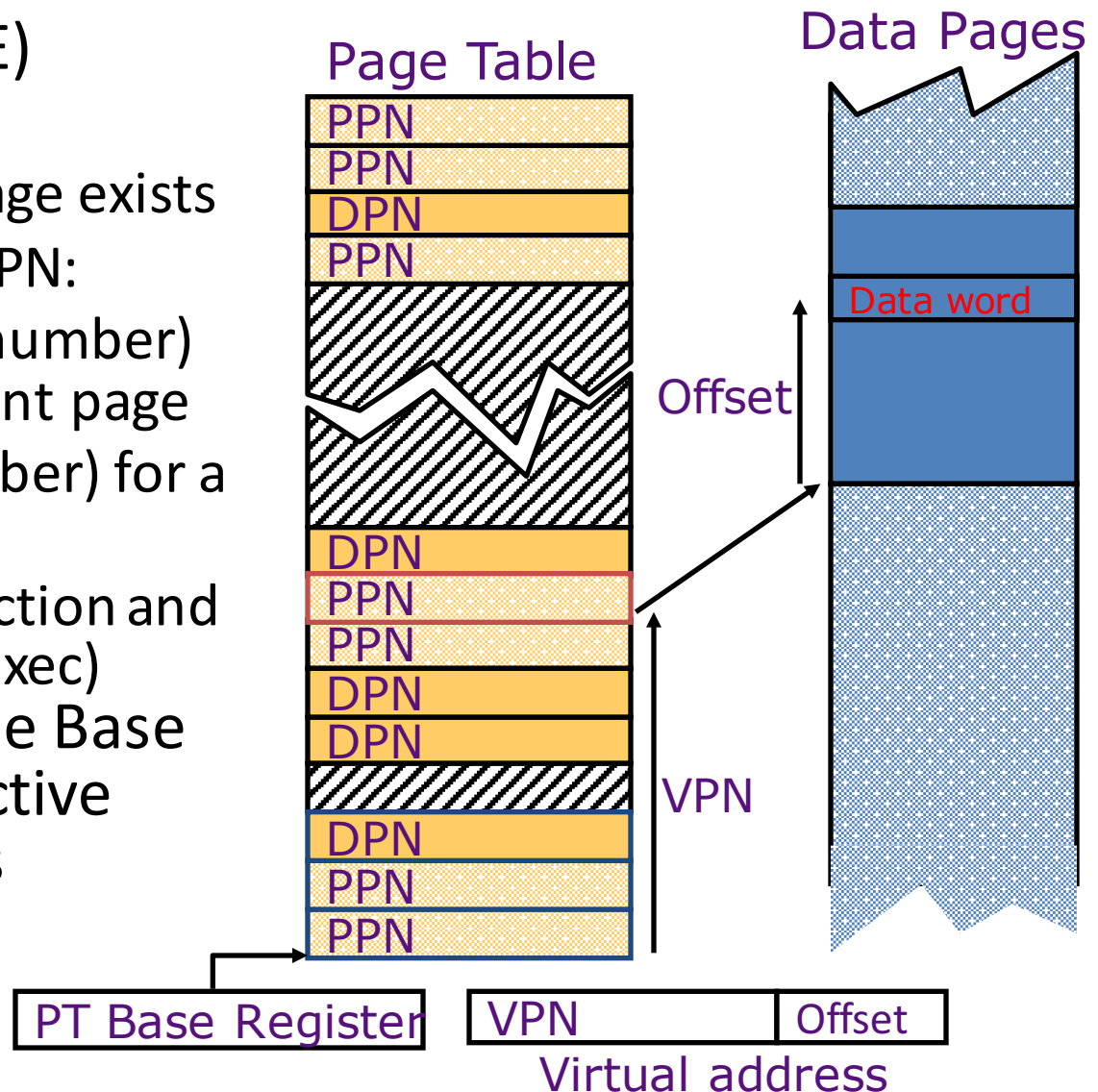
- Space required by the page tables (PT) is proportional to the address space, number of users, ...
 - ✿ *Too large to keep in registers inside CPU*
- Idea: Keep page tables in the main memory
 - Needs one reference to retrieve the page base address and another to access the data word
 - ✿ *doubles the number of memory references! (but we can fix this using something we already know about...)*

Page Tables in Physical Memory



Linear (simple) 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



Suppose an instruction references a memory page that isn't in DRAM?

- We get an exception of type “page fault”
- Page fault handler does the following:
 - If virtual page doesn't yet exist, assign an unused page in DRAM, or if page exists ...
 - Initiate transfer of the page we're requesting from disk to DRAM, assigning to an unused page
 - If no unused page is left, a *page currently in DRAM is selected to be replaced* (based on usage)
 - The replaced page is written (back) to disk, page table entry that maps that VPN->PPN is marked as invalid/DPN
 - Page table entry of the page we're requesting is updated with a (now) valid PPN

Size of Linear Page Table

With 32-bit memory addresses, 4-KiB pages:

- ✈ $2^{32} / 2^{12} = 2^{20}$ virtual pages per user, assuming 4-Byte PTEs,
- ✈ 2^{20} PTEs, i.e, 4 MiB page table per user!

Larger pages?

- Internal fragmentation (Not all memory in page gets used)
- Larger page fault penalty (more time to read from disk)

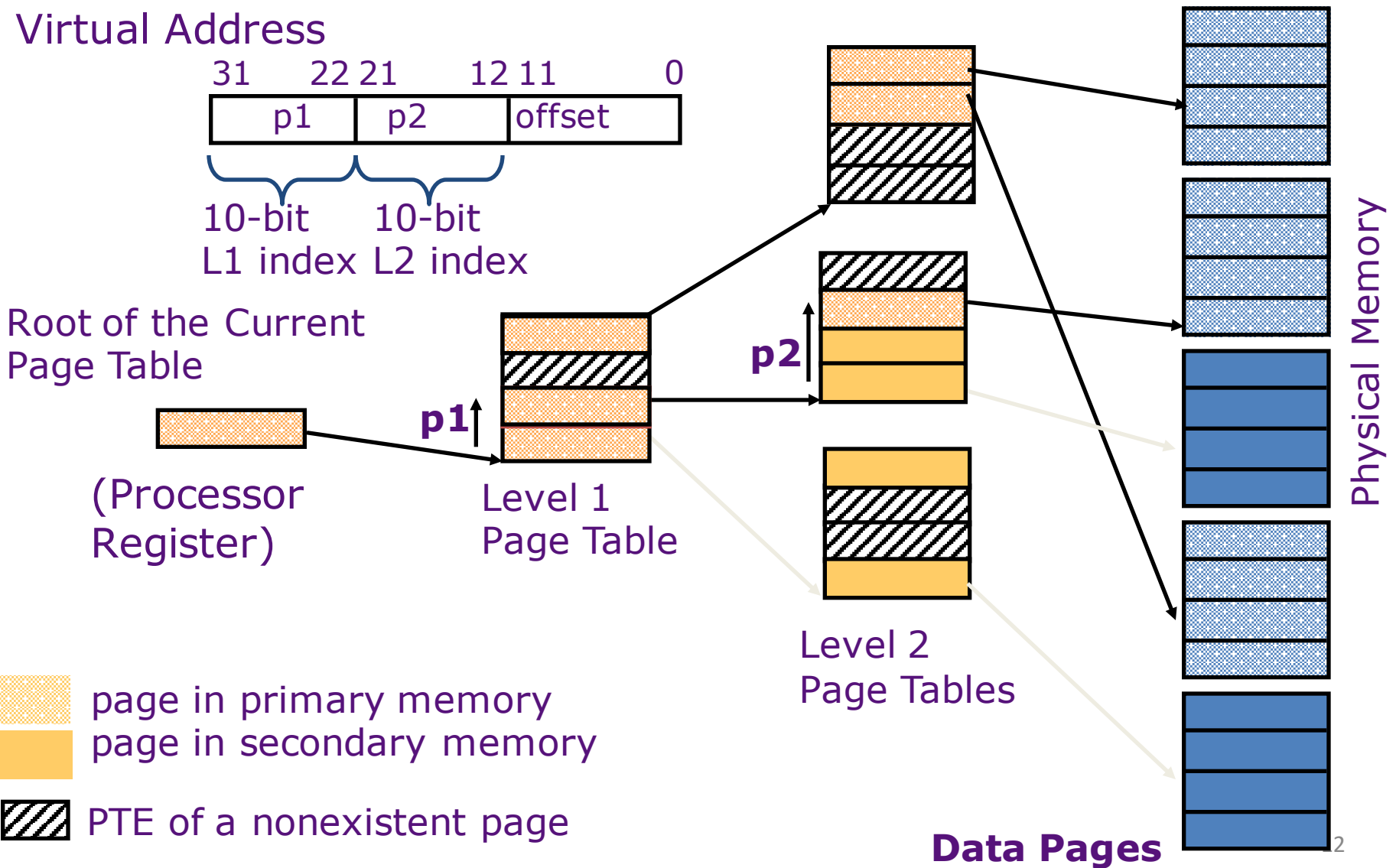
What about 64-bit virtual address space???

- Even 1MiB pages would require 2^{44} 8-Byte PTEs (128 TiB!)

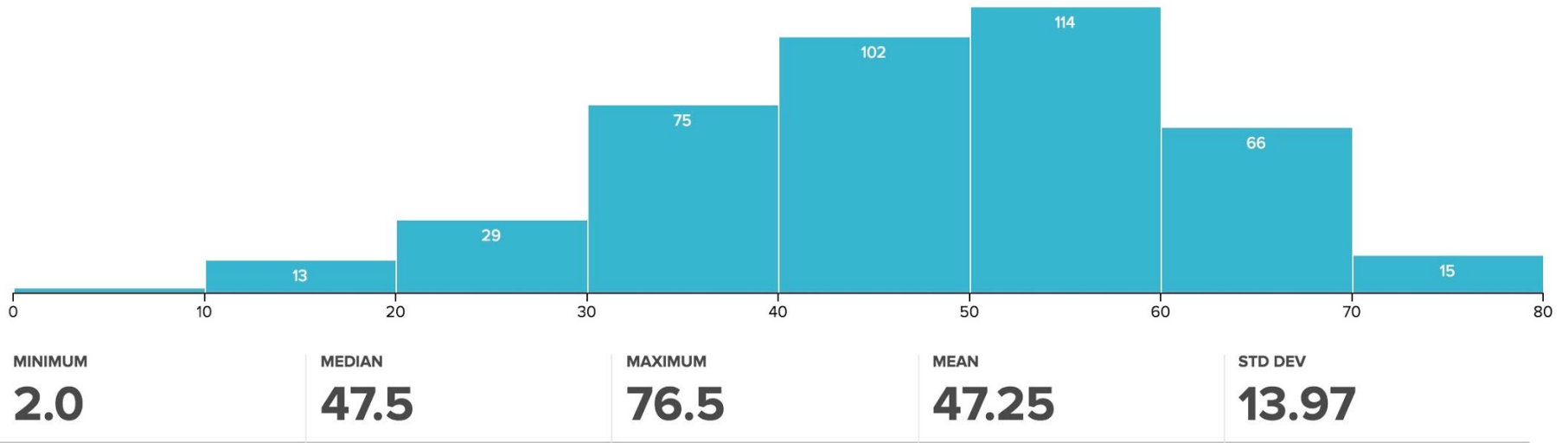
What is the “saving grace” ?

Most processes only use a set of high address (stack), and a set of low address (instructions, heap)

Hierarchical Page Table – exploits sparsity of virtual address space use



MT2 Grades



59%

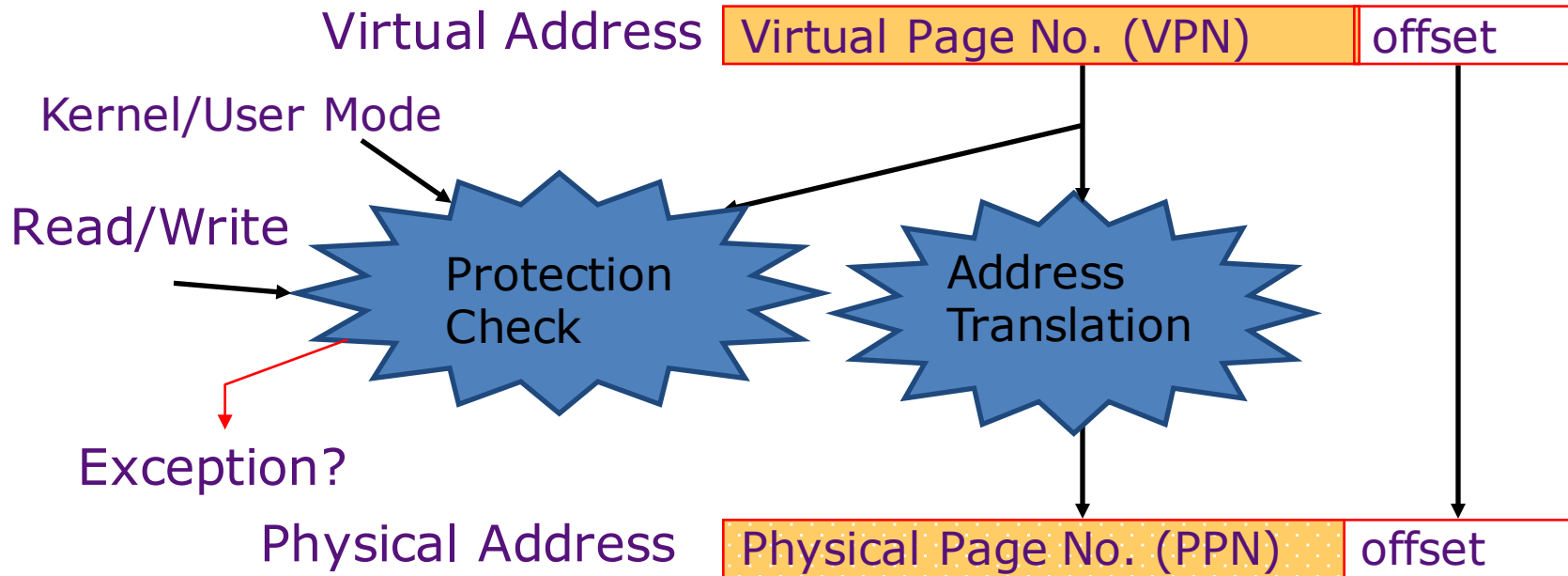
Administrivia

- Upcoming Lecture Schedule
 - 04/18: VM (today)
 - 04/20: I/O: DMA, Disks
 - 04/22: Networking
 - 04/25: Dependability: Parity, ECC (+ HKN reviews)
 - 04/27: RAID
 - Last day of new material
 - 04/29: Summary, What's Next?

Administrivia

- Project 4 programming competition rules posted
- Project 5 released – due on 4/26
- Guerrilla Session: Virtual Memory
 - Wed 4/20 3 - 5 PM @ 241 Cory
 - Sat 4/22 1 - 3 PM @ 521 Cory
- Last HW (3) Virtual Memory
 - Due 05/01

Address Translation & Protection



- Every instruction and data access needs address translation and protection checks

A good VM design needs to be fast (~ one cycle) and space efficient

Translation Lookaside Buffers (TLB)

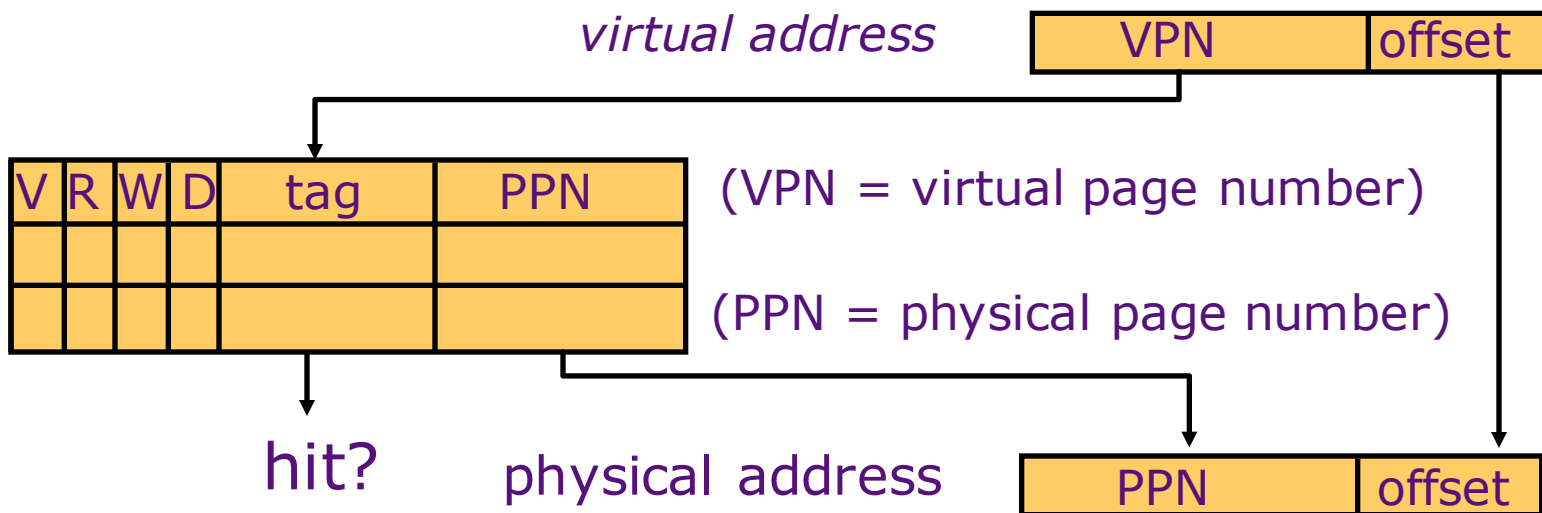
Address translation is very expensive!

In a two-level page table, each reference becomes several memory accesses

Solution: *Cache some translations in TLB*

TLB hit 🦋 *Single-Cycle Translation*

TLB miss 🦋 *Page-Table Walk to refill*



TLB Designs

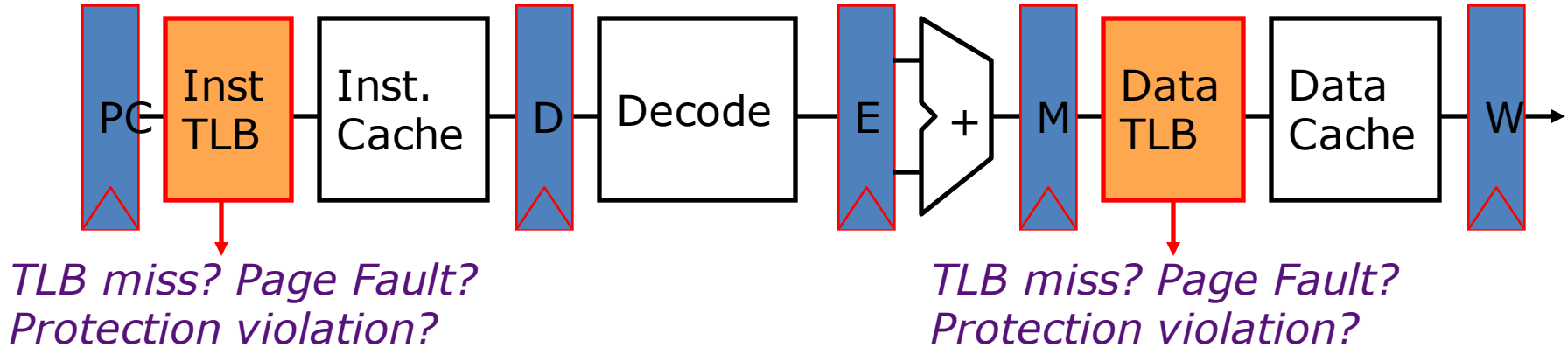
- Typically 32-128 entries, usually fully associative
 - Each entry maps a large page, hence less spatial locality across pages => more likely that two entries conflict
 - Sometimes larger TLBs (256-512 entries) are 4-8 way set-associative
 - Larger systems sometimes have multi-level (L1 and L2) TLBs
- Random or FIFO replacement policy
- “TLB Reach”: Size of largest virtual address space that can be simultaneously mapped by TLB

Example: 64 TLB entries, 4KiB pages, one page per entry

TLB Reach =

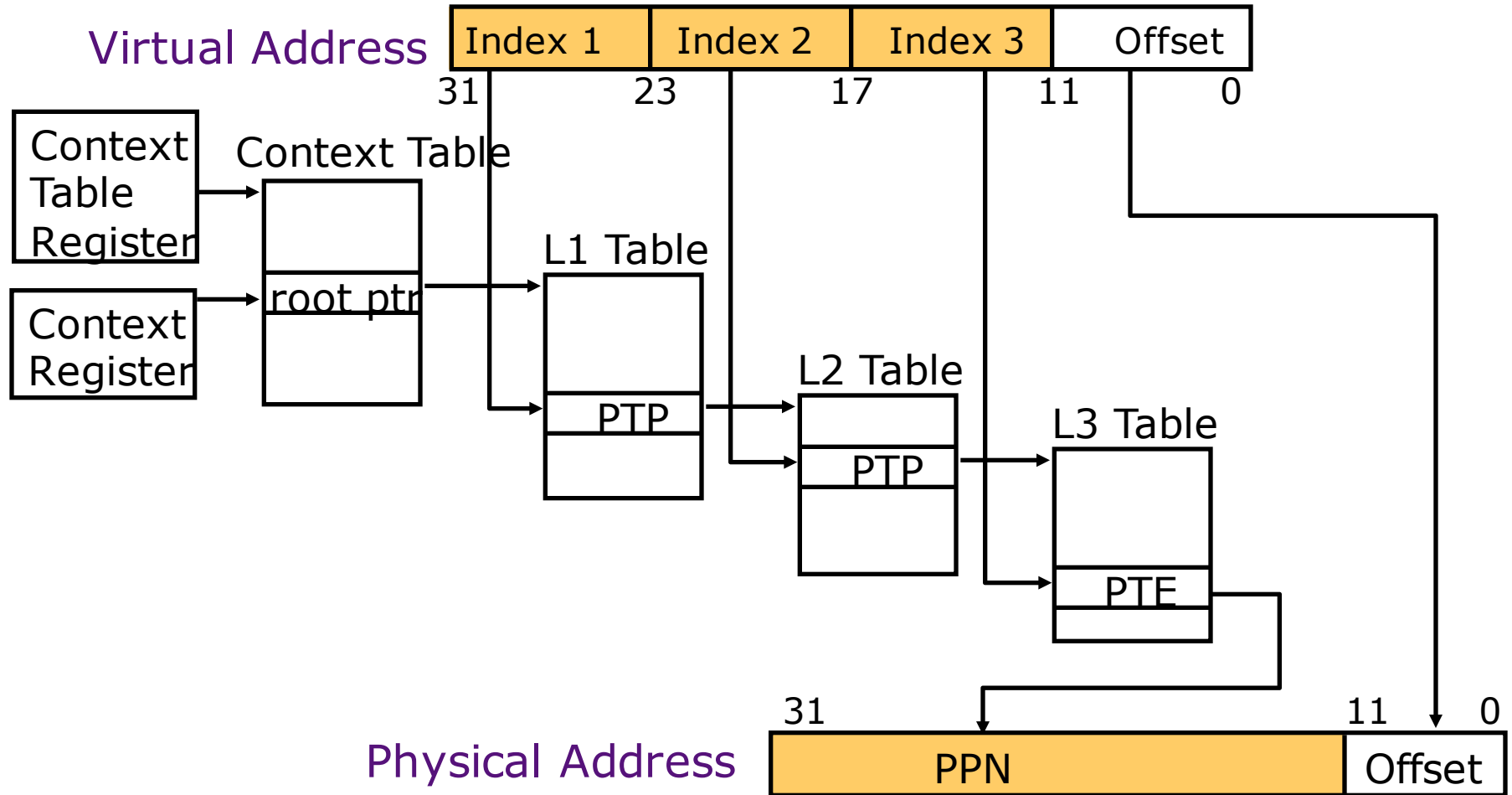
_____?

VM-related events in pipeline



- Handling a TLB miss needs a hardware or software mechanism to refill TLB
 - usually done in hardware now
- Handling a page fault (e.g., page is on disk) needs a *precise* trap so software handler can easily resume after retrieving page
- Handling protection violation may abort process

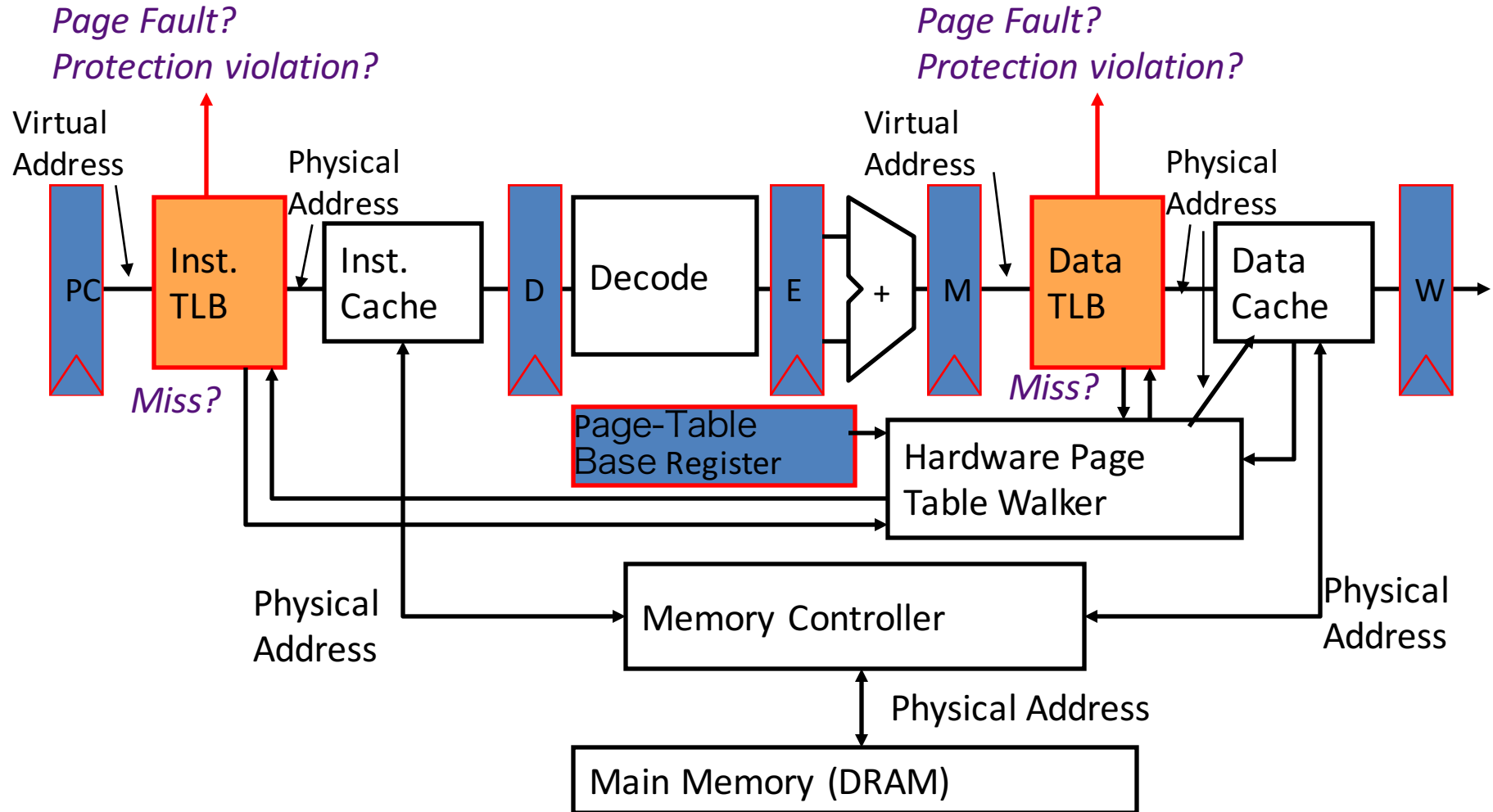
Hierarchical Page Table Walk: SPARC v8



MMU does this table walk in hardware on a TLB miss

Page-Based Virtual-Memory Machine

(Hardware Page-Table Walk)

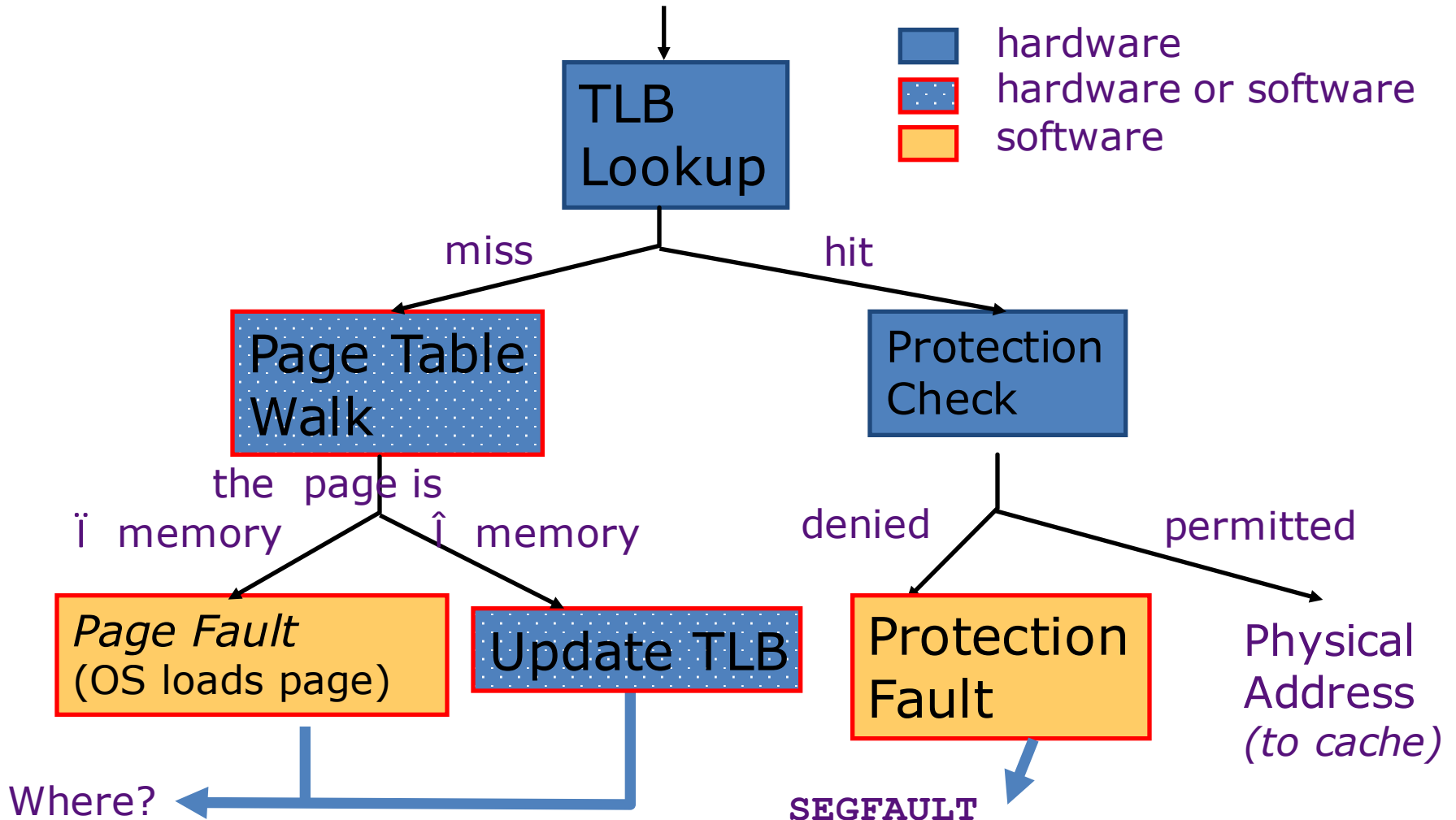


- Assumes page tables held in untranslated physical memory

Address Translation:

putting it all together


Virtual Address

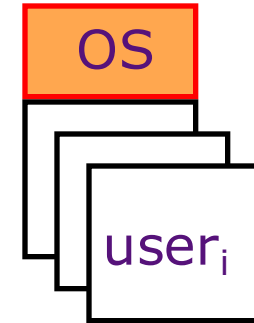


Modern Virtual Memory Systems

Illusion of a large, private, uniform store

Protection & Privacy

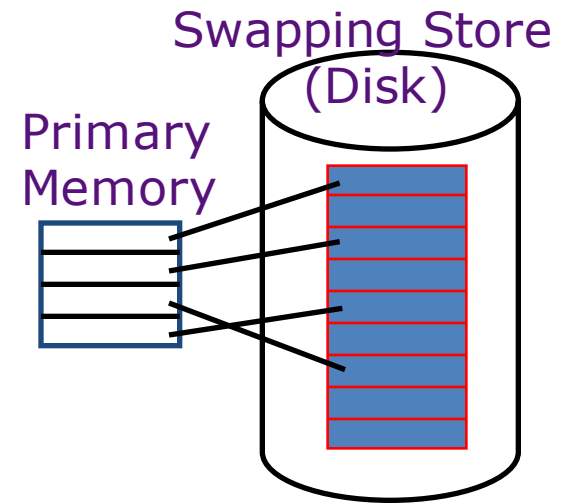
several users, each with their private address space and one or more shared address spaces
page table  name space



Demand Paging

Provides the ability to run programs larger than the primary memory

Hides differences in machine configurations



The price is address translation on each memory reference



Clicker Question

Let's try to extrapolate from caches... Which one is false?

- A. # offset bits in V.A. = $\log_2(\text{page size})$
- B. # offset bits in P.A. = $\log_2(\text{page size})$
- C. # VPN bits in V.A. = $\log_2(\# \text{ of physical pages})$
- D. # PPN bits in P.A. = $\log_2(\# \text{ of physical pages})$
- E. A single-level page table contains a PTE for every possible VPN in the system

Conclusion: VM features track historical uses

- **Bare machine, only physical addresses**
 - One program owned entire machine
- **Batch-style multiprogramming**
 - Several programs sharing CPU while waiting for I/O
 - Base & bound: translation and protection between programs (not virtual memory)
 - Problem with external fragmentation (holes in memory), needed occasional memory defragmentation as new jobs arrived

Conclusion: VM features track historical uses

- **Time sharing**
 - More interactive programs, waiting for user. Also, more jobs/second.
 - Motivated move to fixed-size page translation and protection, no external fragmentation (but now internal fragmentation, wasted bytes in page)
 - Motivated adoption of virtual memory to allow more jobs to share limited physical memory resources while holding working set in memory
- **Virtual Machine Monitors**
 - Run multiple operating systems on one machine
 - Idea from 1970s IBM mainframes, now common on laptops
 - e.g., run Windows on top of Mac OS X
 - Hardware support for two levels of translation/protection
 - Guest OS virtual -> Guest OS physical -> Host machine physical
 - Also basis of Cloud Computing
 - Virtual machine instances on EC2