CS 61C:
Great Ideas in Computer Architecture
Virtual Memory

Instructors:

Krste Asanovic & Vladimir Stojanovic

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

Review

- Programmed I/O
- Polling versus Interrupts
- Asynchronous interrupts versus synchronous traps
- Precise interrupt looks like execution stopped at exactly one instruction, every instruction before finished, no instruction after started.
  - Simplify software view of interrupted state

You Are Here!

Software | *Hardware*

- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time
- Programming Languages

Harness Parallelism & Achieve High Performance

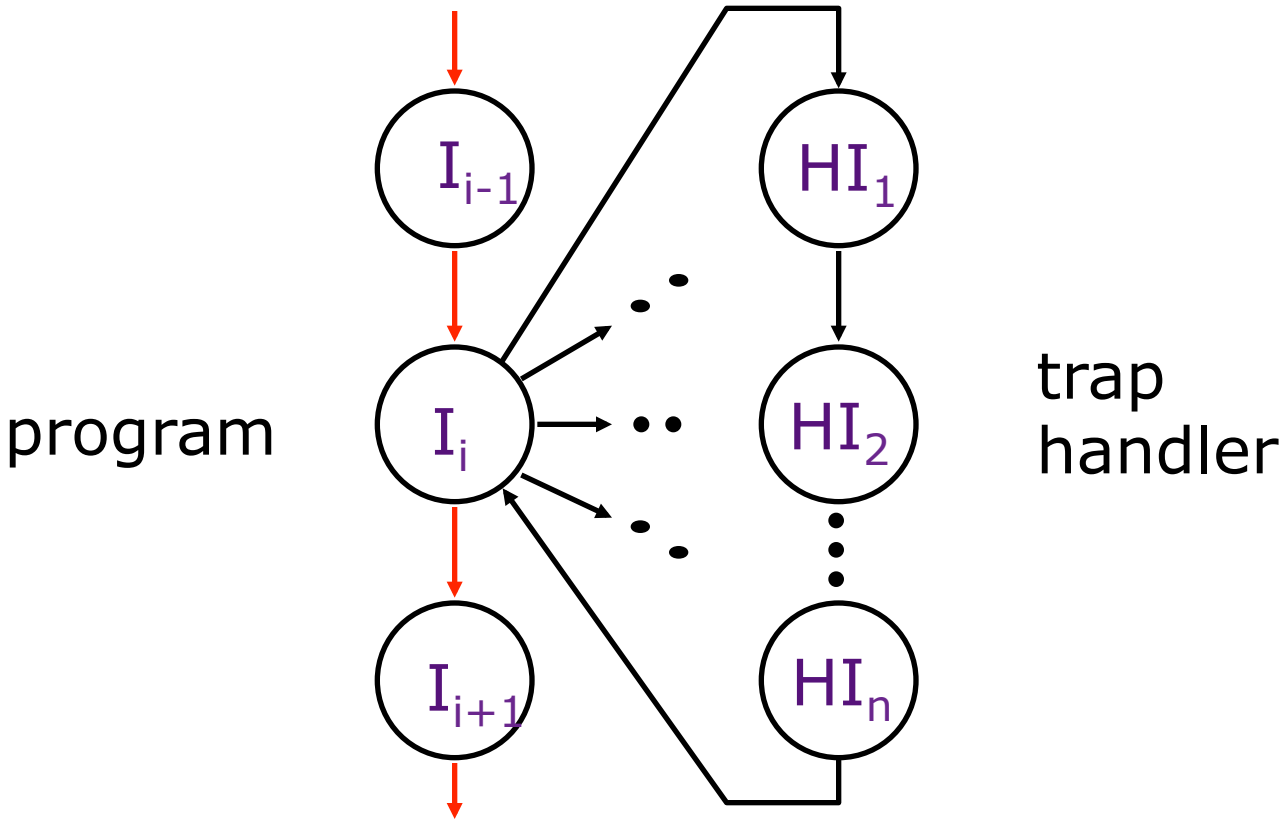
Today's Lecture

Warehouse Scale Computer

Smart Phone

Traps/Interrupts/Execeptions:

altering the normal flow of control



An *external or internal event* that needs to be processed by another (system) program. The event is usually unexpected or rare from program's point of view.

Terminology

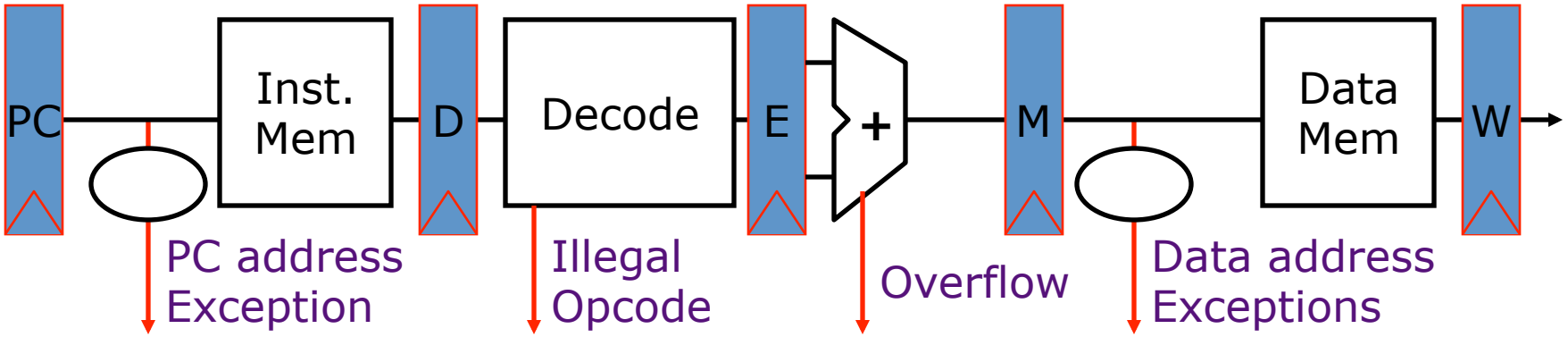
In CS61C (you'll see other definitions in use elsewhere):

- Interrupt – caused by an event external to current running program (e.g. key press, mouse activity)
  - Asynchronous to current program, can handle interrupt on any convenient instruction
- Exception – caused by some event during execution of one instruction of current running program (e.g., page fault, illegal instruction)
  - Synchronous, must handle exception on instruction that causes exception
- Trap – action of servicing interrupt or exception by hardware jump to "trap handler" code

Precise Traps

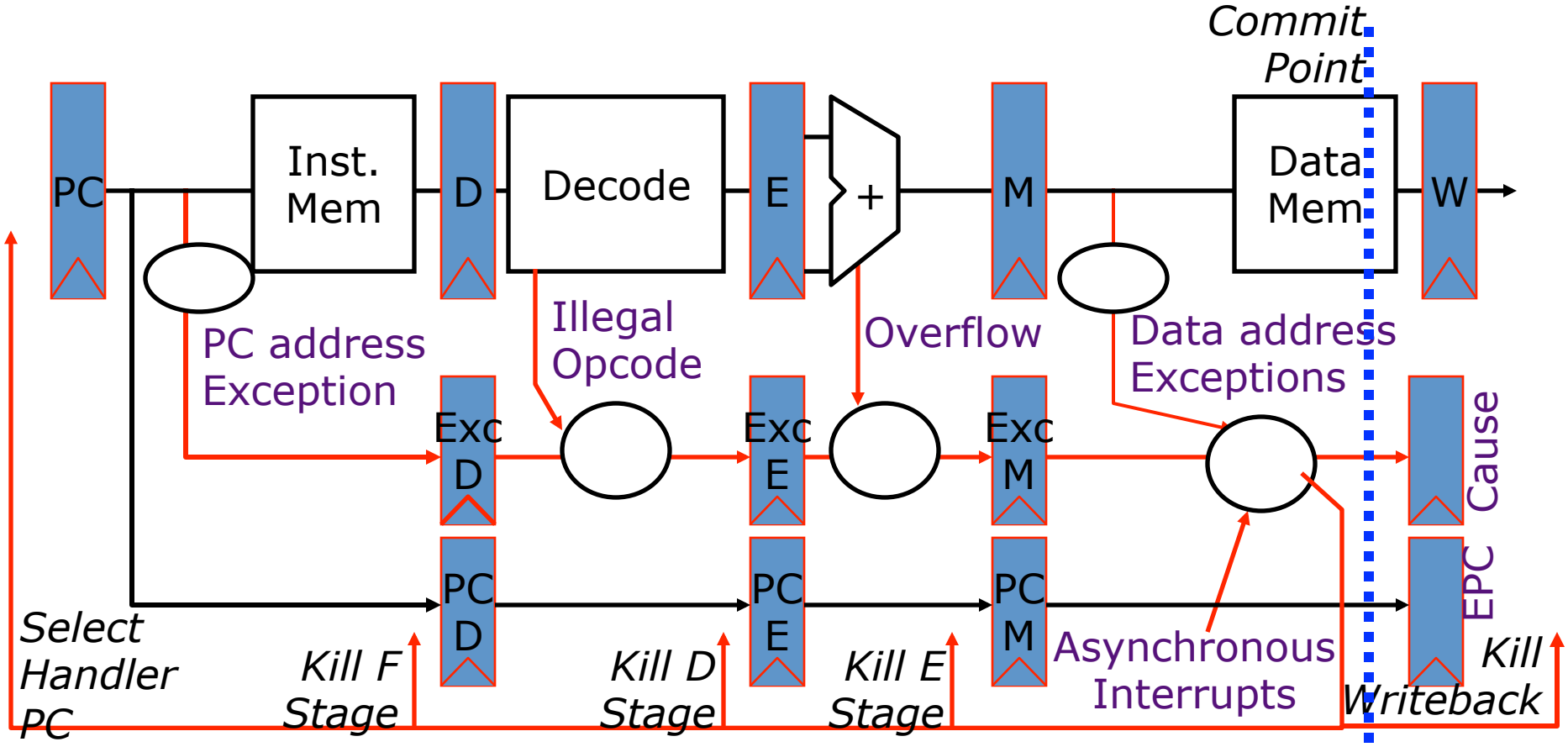
- Trap handler's view of machine state is that every instruction prior to the trapped one has completed, and no instruction after the trap has executed.
- Implies that handler can return from an interrupt by restoring user registers and jumping to EPC
  - Interrupt handler software doesn't need to understand the pipeline of the machine, or what program was doing!
  - More complex to handle trap caused by an exception
- Providing precise traps is tricky in a pipelined superscalar out-of-order processor!
  - But handling imprecise interrupts in software is even worse.

Trap Handling in 5-Stage Pipeline



- How to handle multiple simultaneous exceptions in different pipeline stages?
- How and where to handle external asynchronous interrupts?

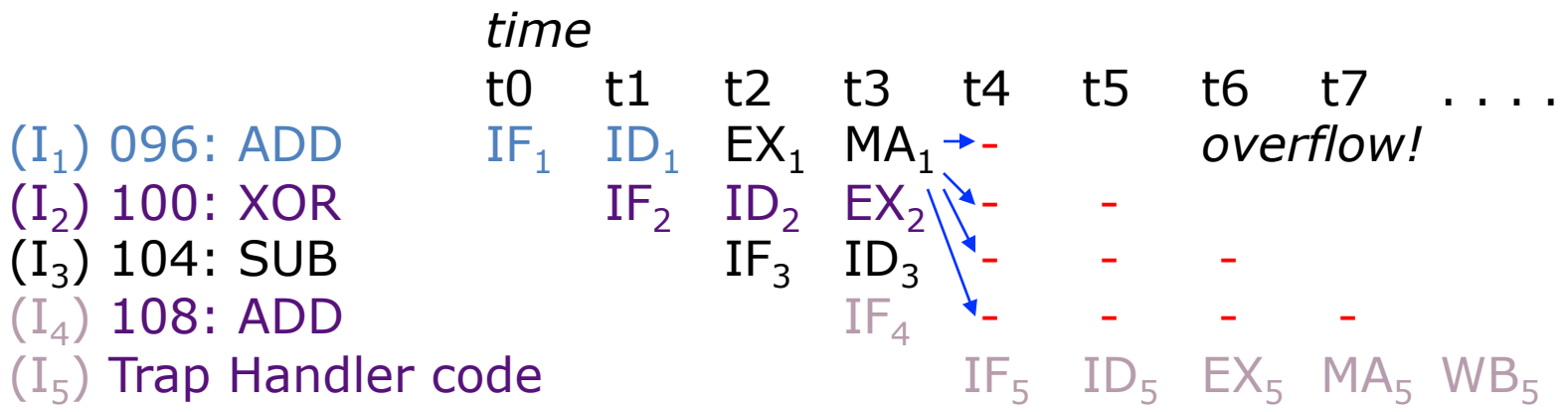
Save Exceptions Until Commit



Handling Traps in In-Order Pipeline

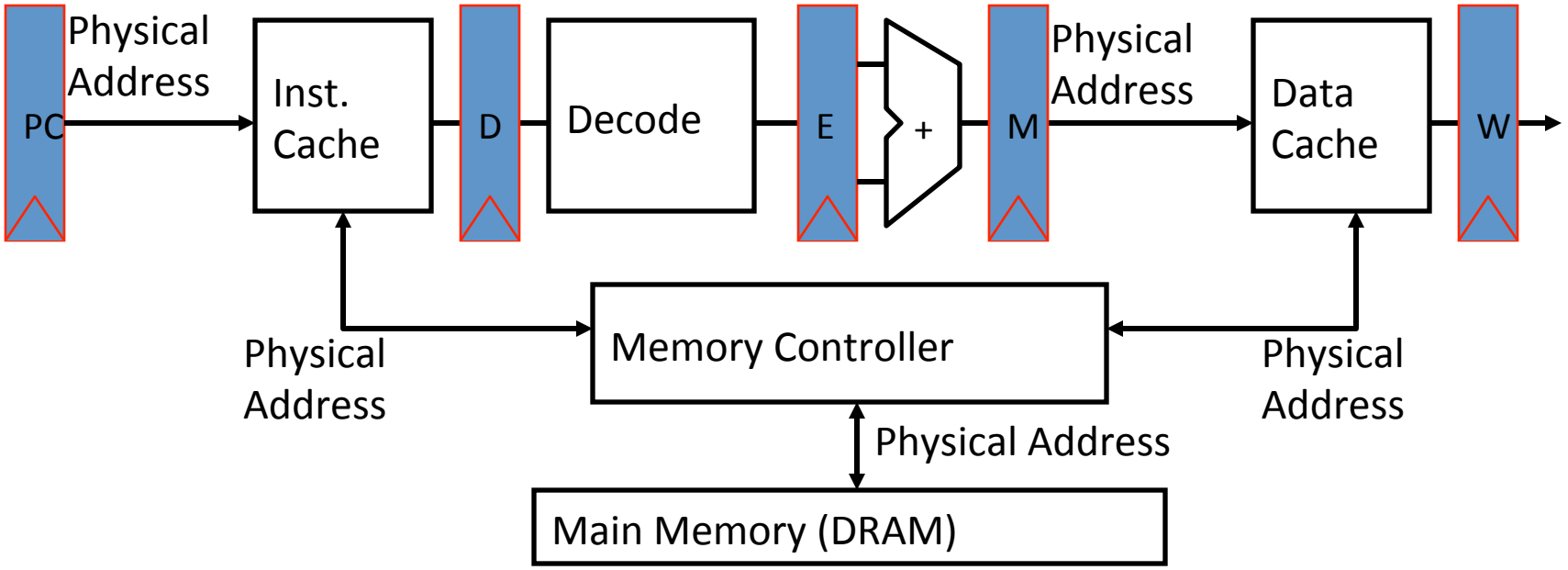
- Hold exception flags in pipeline until commit point (M stage)
- Exceptions in earlier pipe stages override later exceptions *for a given instruction*
- Inject external interrupts at commit point (override others)
- If exception/interrupt at commit: update Cause and EPC registers, kill all stages, inject handler PC into fetch stage

Trap Pipeline Diagram



Virtual Memory

“Bare” 5-Stage Pipeline



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

Dynamic Address Translation

Motivation

- In early machines, I/O operations were slow and each word transferred involved the CPU
- Higher throughput if CPU and I/O of 2 or more programs were overlapped.
  - *How?⇒ multiprogramming with DMA I/O devices, interrupts*

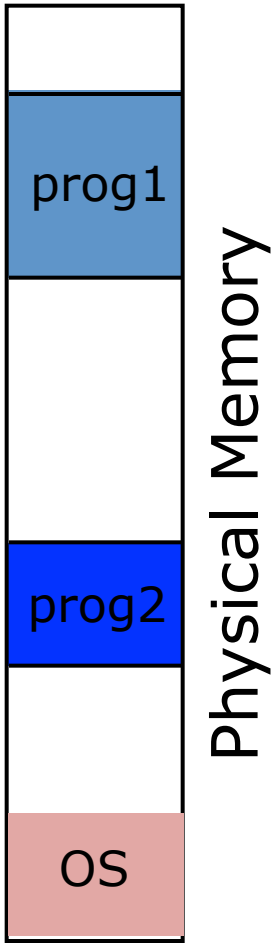
Location-independent programs

- Programming and storage management ease ⇒ need for a *base register*

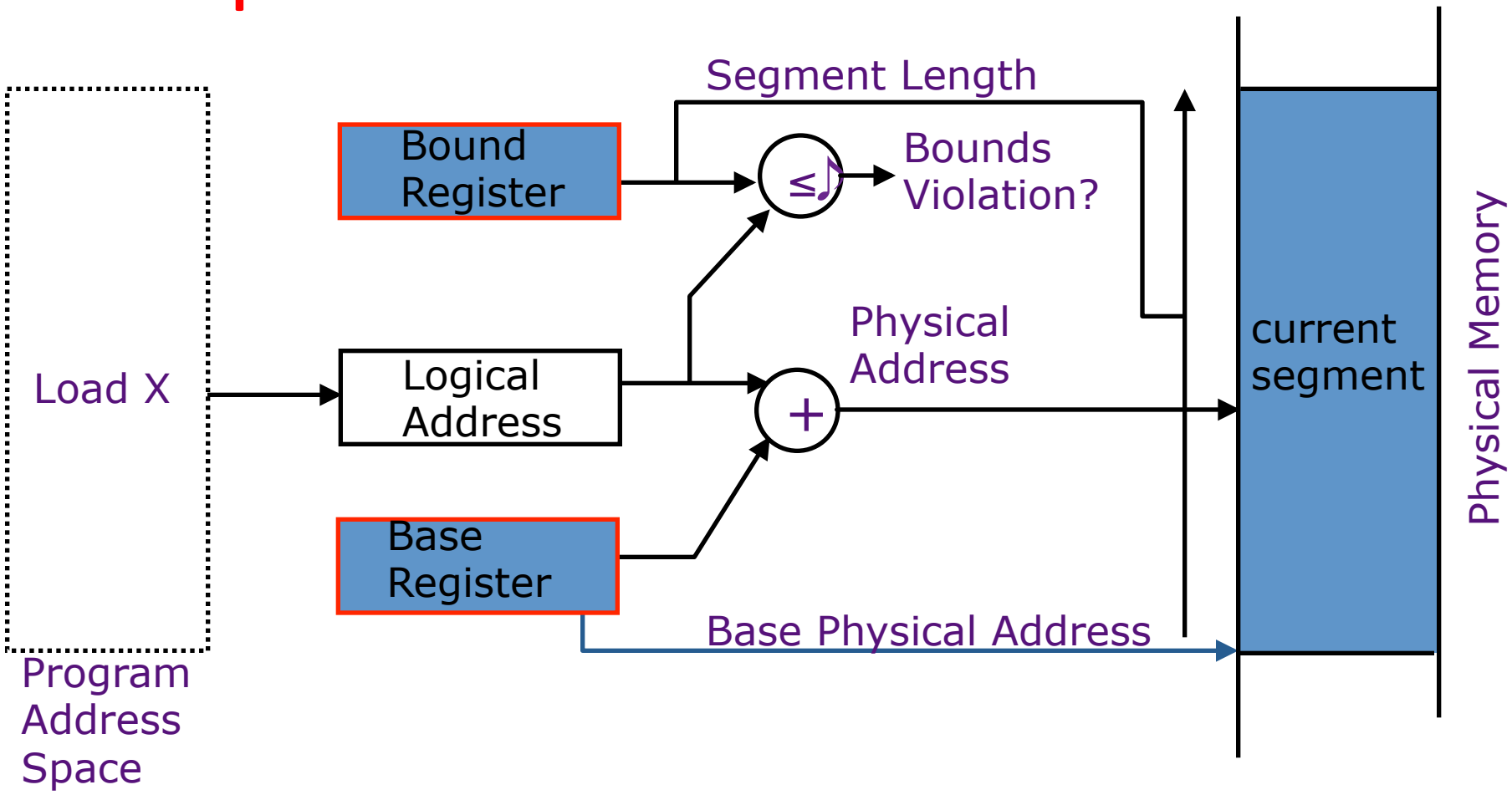
Protection

- Independent programs should not affect each other inadvertently
- ⇒ need for a *bound register*

Multiprogramming drives requirement for resident *supervisor (operating system)* software to manage context switches between multiple programs



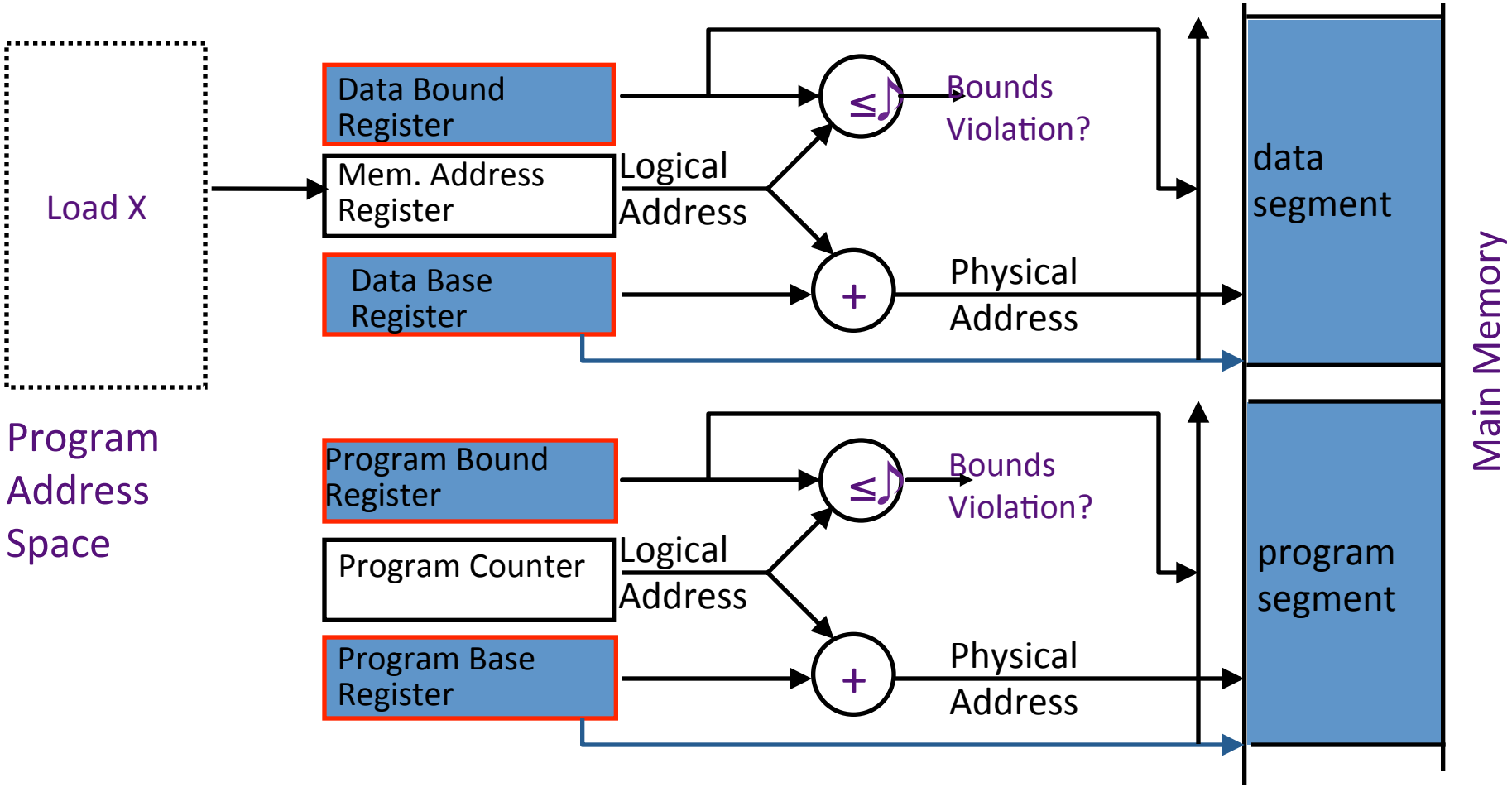
Simple Base and Bound Translation



Base and bounds registers are visible/accessible only when processor is running in *supervisor mode*

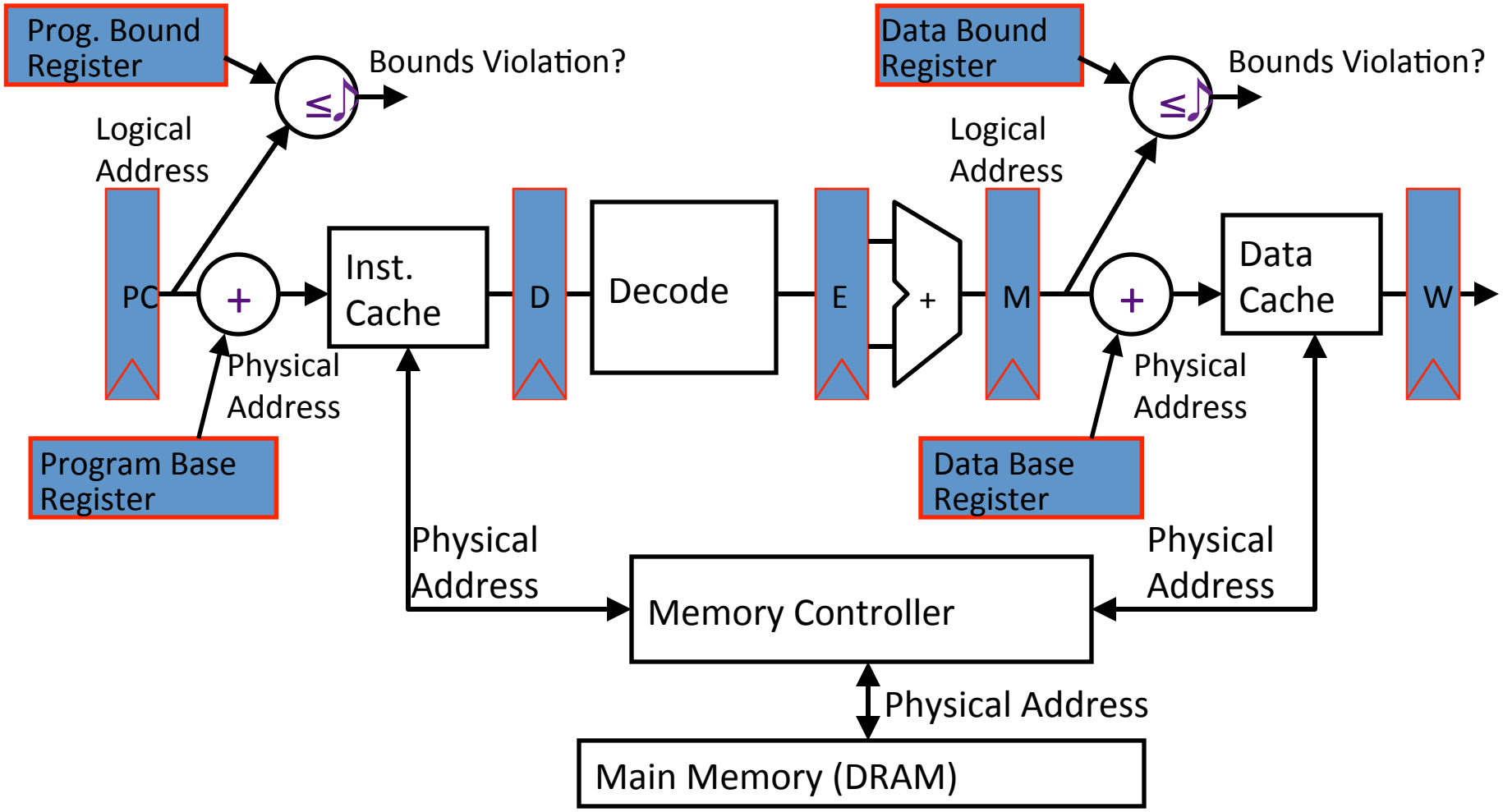
Separate Areas for Program and Data

(Scheme used on all Cray vector supercomputers prior to X1, 2002)



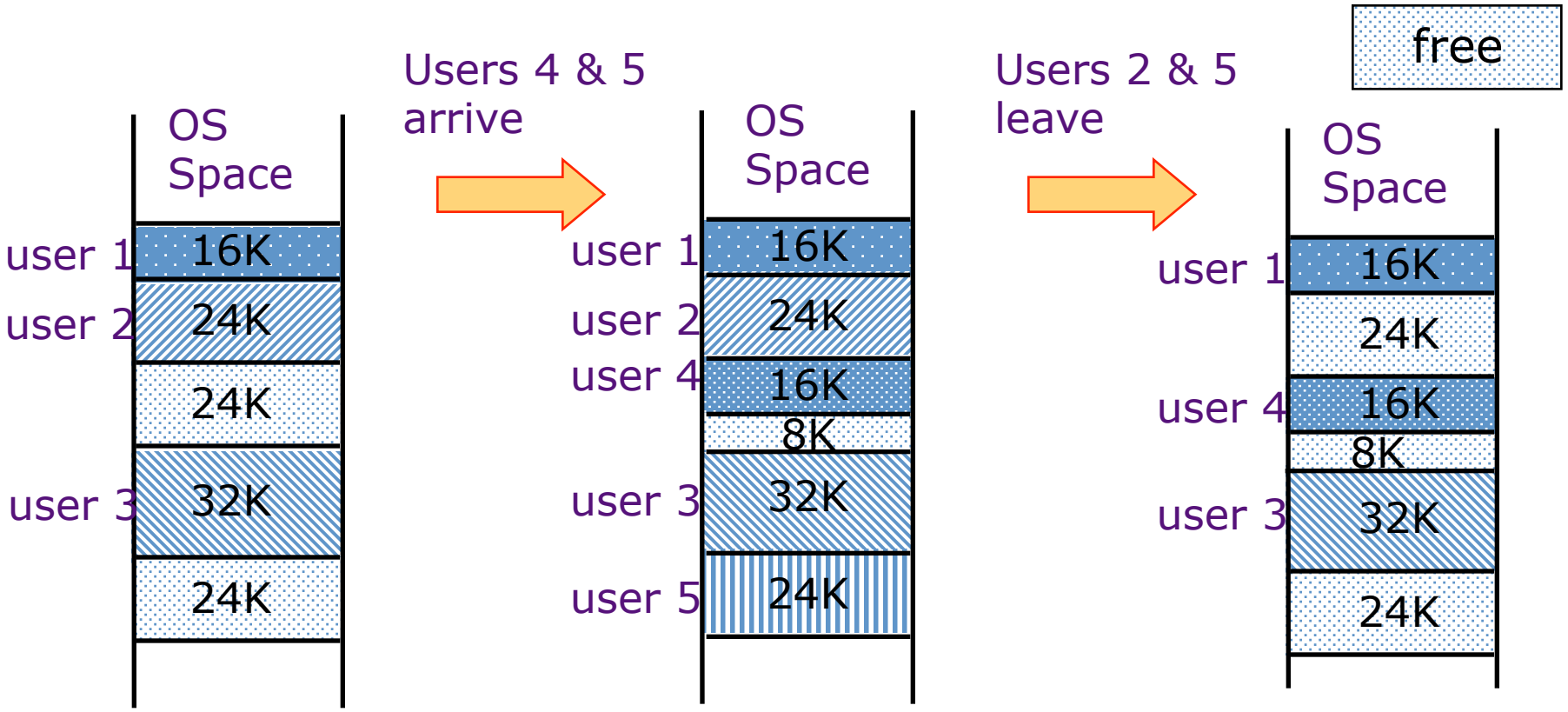
What is an advantage of this separation?

Base and Bound Machine



[Can fold addition of base register into (register+immediate) address calculation using a carry-save adder (sums three numbers with only a few gate delays more than adding two numbers)]

Memory Fragmentation



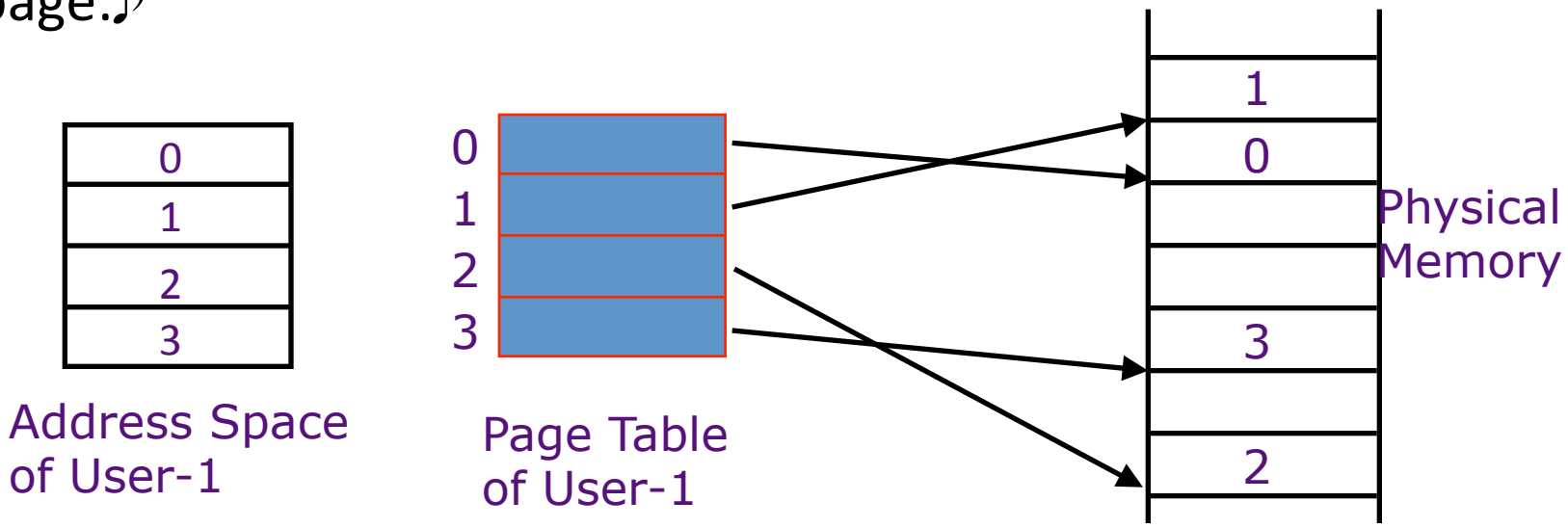
As users come and go, the storage is "fragmented". Therefore, at some stage programs have to be moved around to compact the storage.

Paged Memory Systems

- Processor-generated address can be split into:

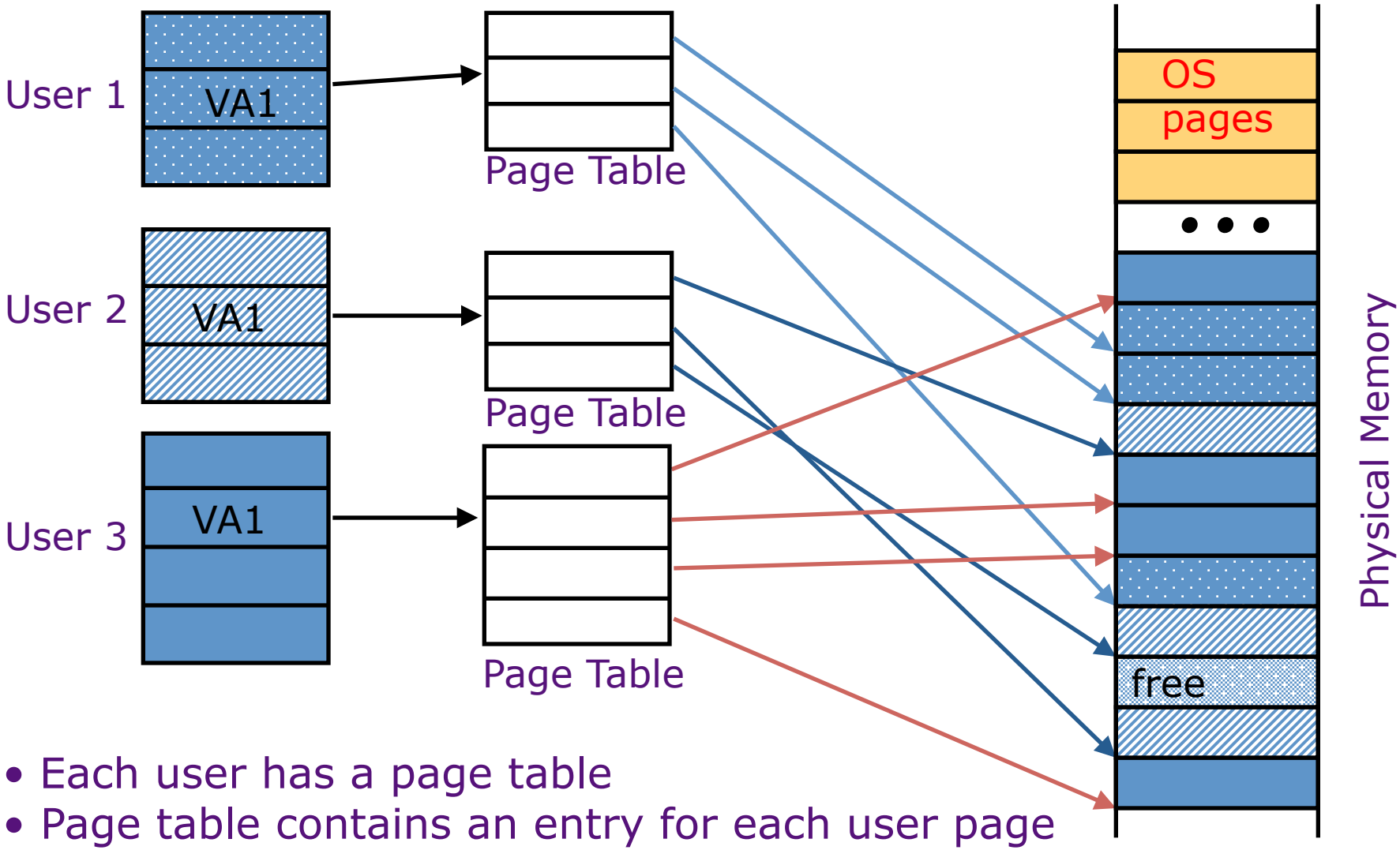
page number	offset

- 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 Address Space per User



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

Where Should Page Tables Reside?

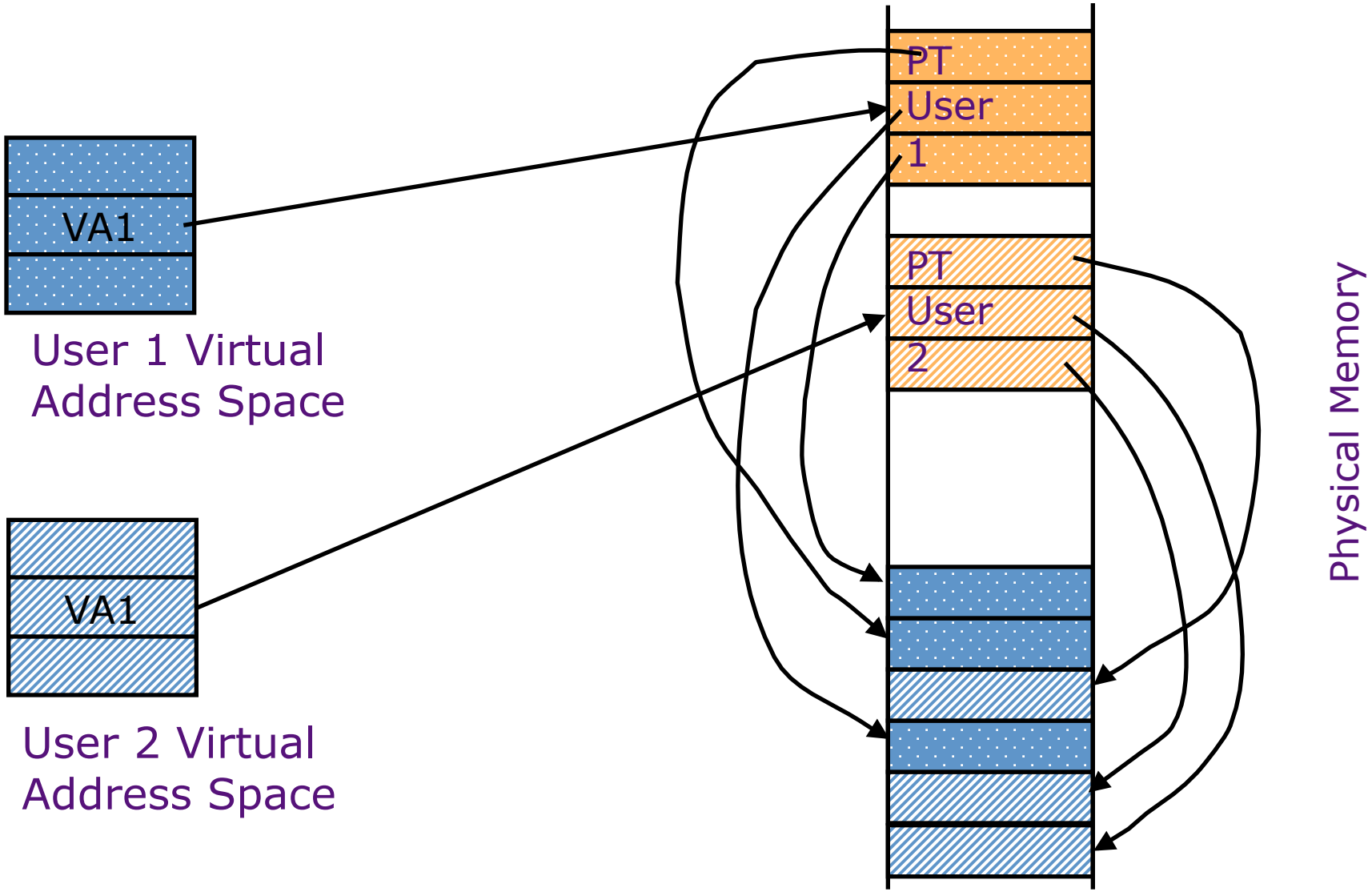
- Space required by the page tables (PT) is proportional to the address space, number of users, ...

  ⇒ *Too large to keep in registers inside CPU*

- Idea: Keep PTs 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!*

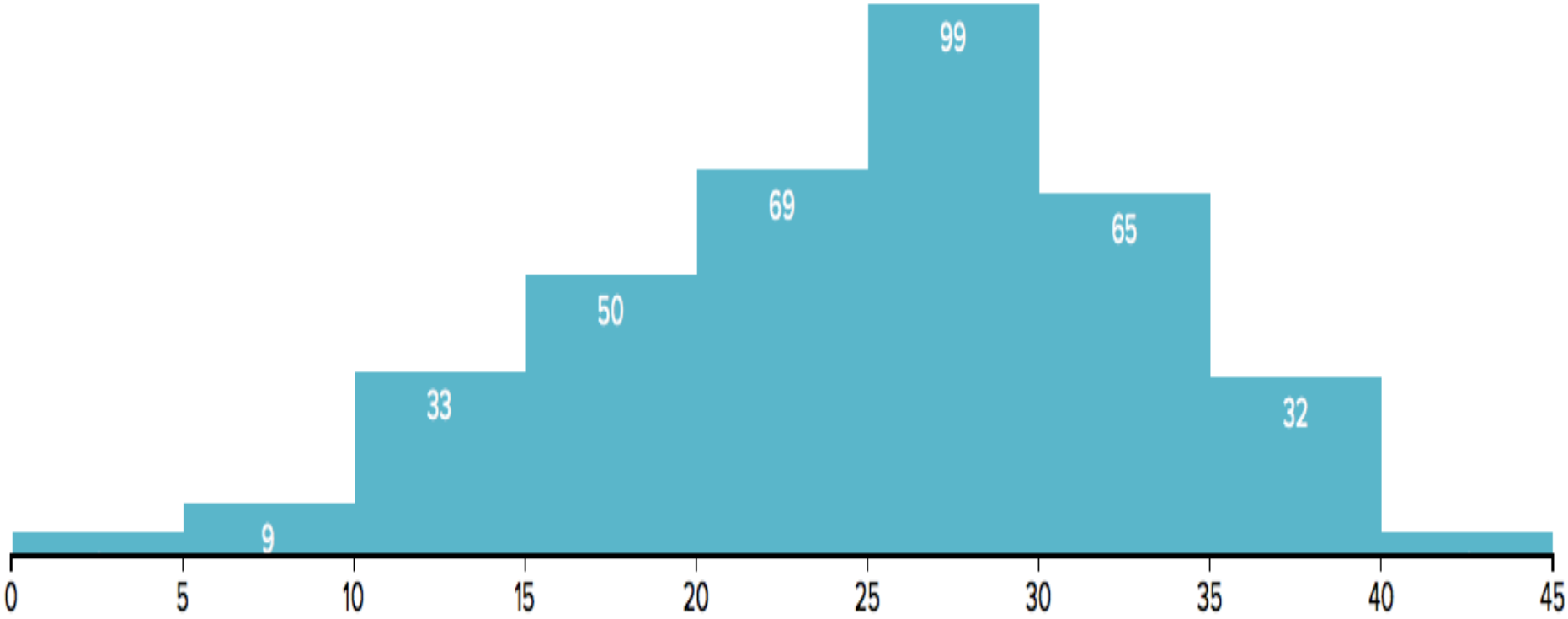
Page Tables in Physical Memory



Administrivia

- Midterm 2 scores up:
  - Regrade request deadline is 23:59:59 on Sunday April 26th
- Clobber Policy:
  - Final composed of MT1, MT2, post-MT2 sections
  - z-scores on MT1/MT2 sections of Final compared to MT1/MT2 grades, will replace if better
- Proj4-1 due date extended to Wed, April 29

Midterm 2 Distribution



MINIMUM	MEDIAN	MAXIMUM	MEAN	STD DEV
0.0	25.5	43.5	24.72	7.87

CS61C In the News:
"Moore's Law 50 Years Anniversary!"

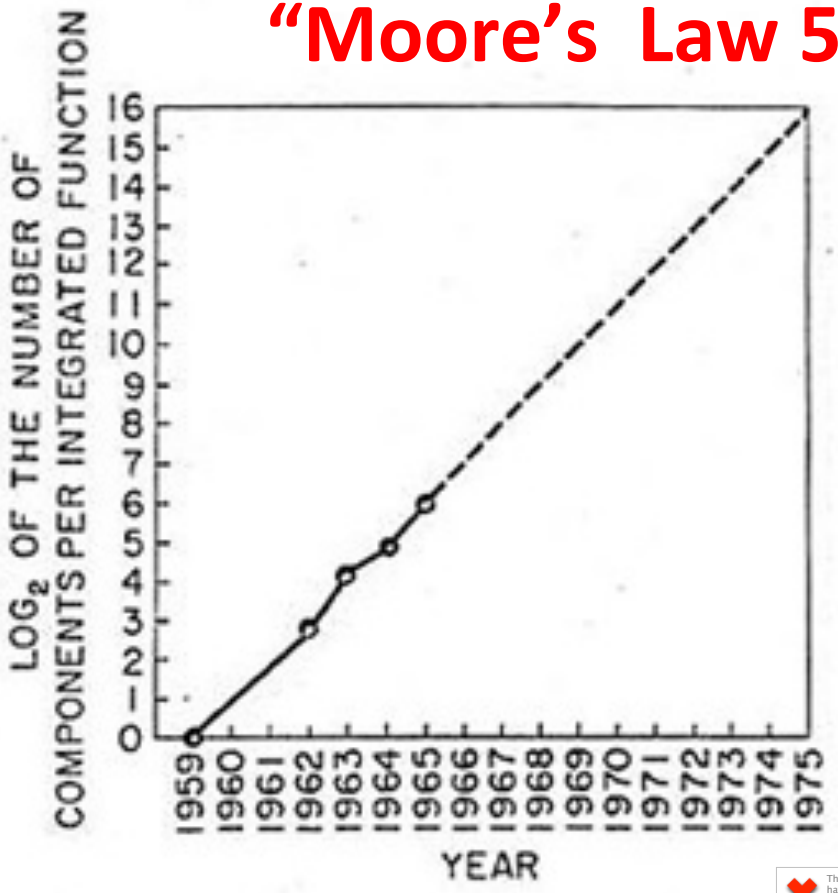


Fig. 2 Number of components per integrated function for minimum cost per component extrapolated vs time.

Gordon Moore's paper appeared in 19 April 1965 issue of *Electronics.*

"With unit cost falling as the number of components per circuit rises, by 1975 economics may dictate squeezing as many as 65,000 components on a single silicon chip."

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

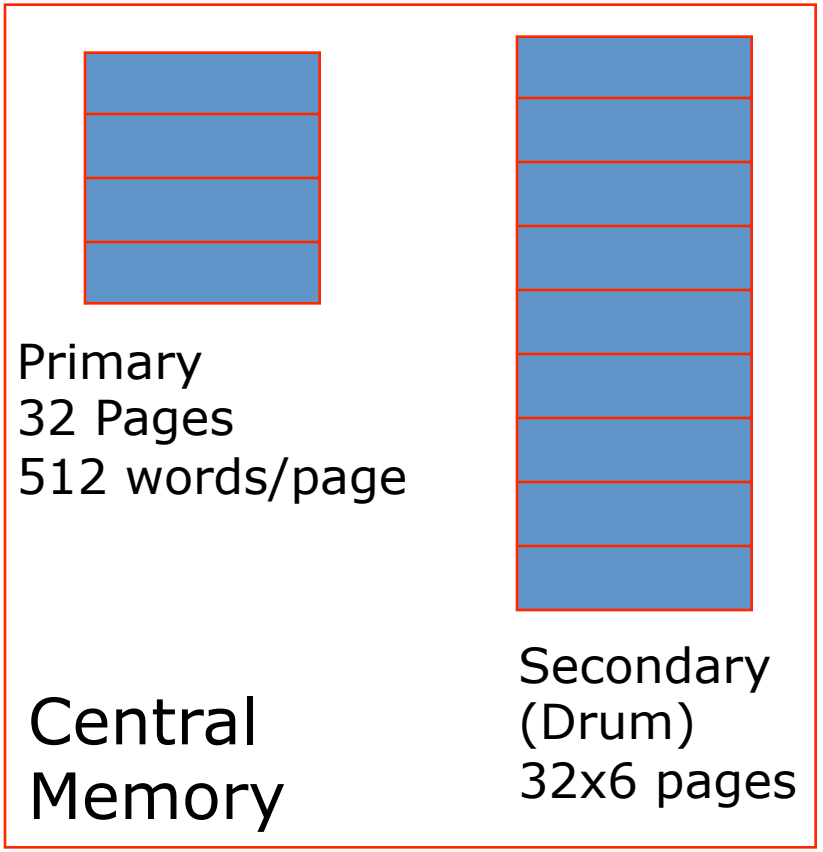
Demand Paging in Atlas (1962)

"A page from secondary storage is brought into the primary storage whenever it is (implicitly) demanded by the processor."

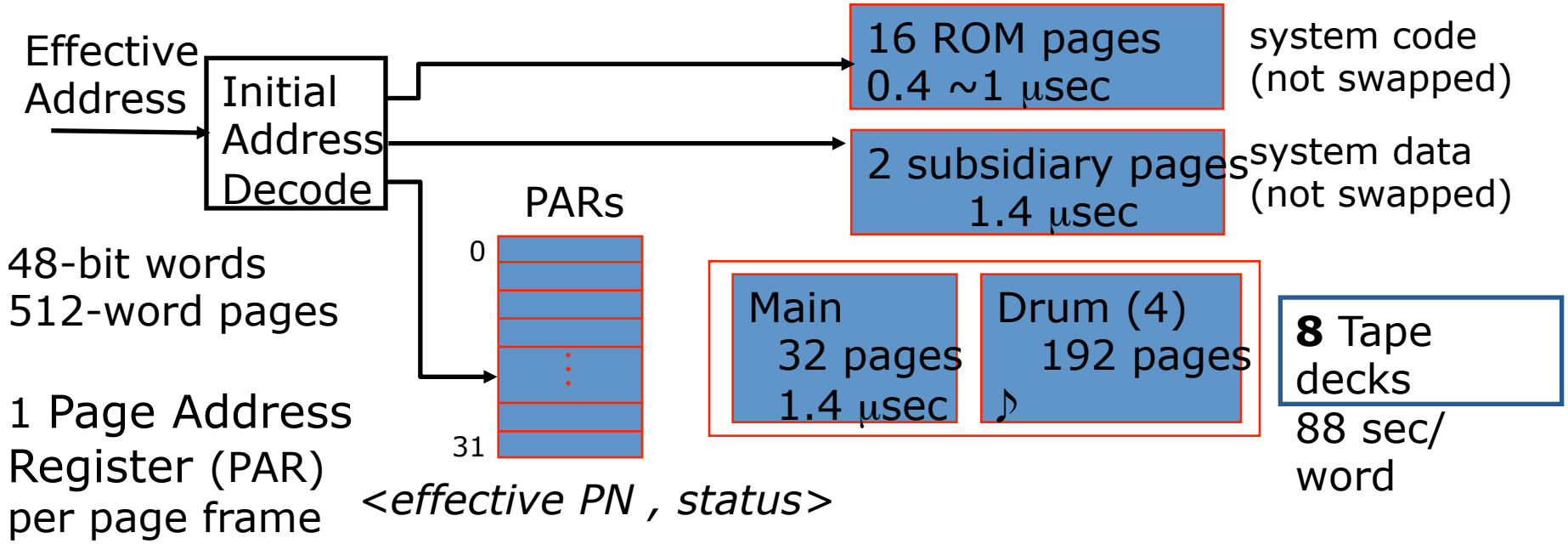
Tom Kilburn

Primary memory as a *cache* for secondary memory

User sees 32 x 6 x 512 words of storage



Hardware Organization of Atlas



Compare the effective page address against all 32 PARs

match ⇒ normal access

no match ⇒ *page fault*

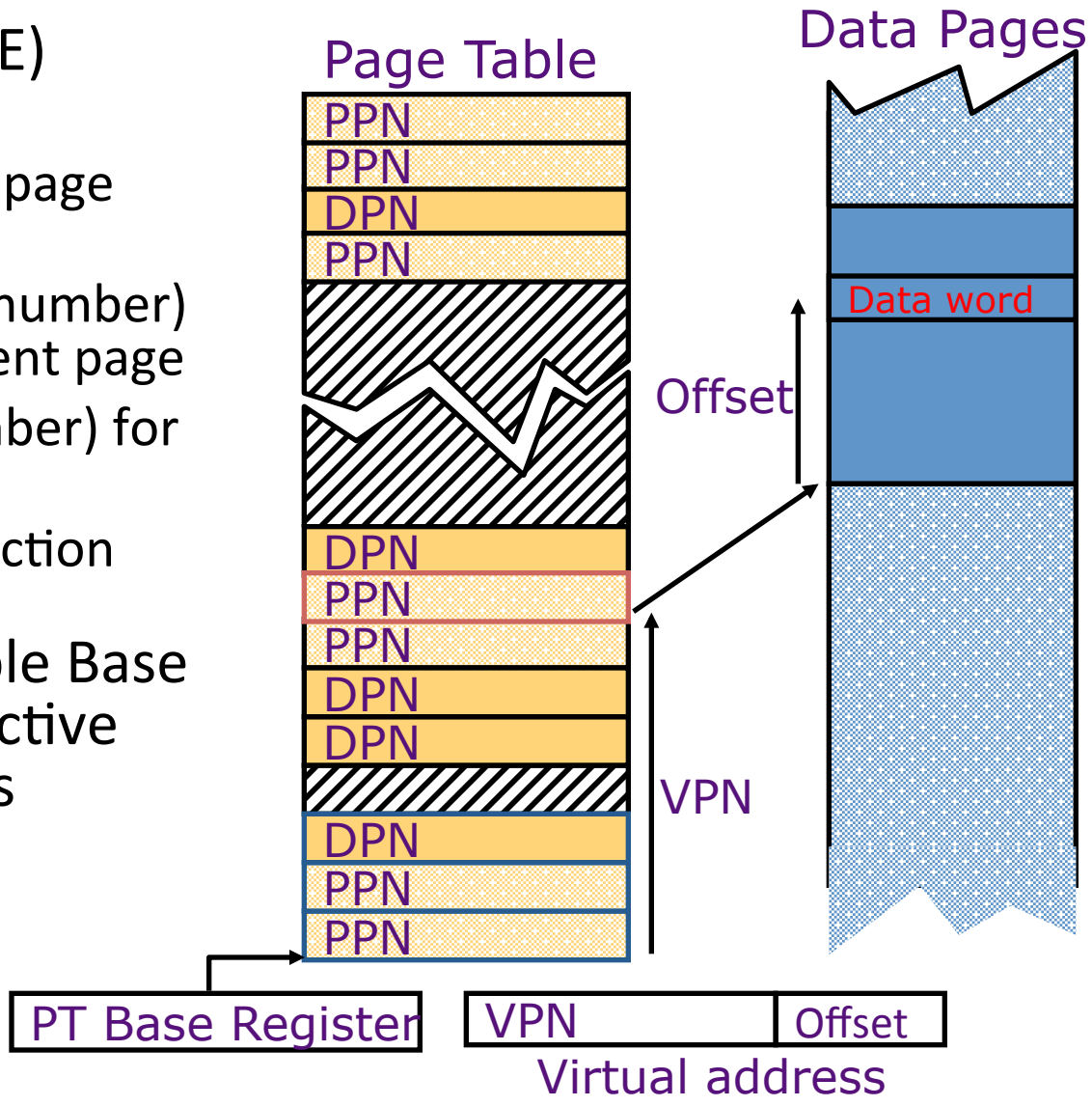
save the state of the partially executed instruction

Atlas Demand Paging Scheme

- On a page fault:
  - Input transfer into a free page is initiated
  - The Page Address Register (PAR) is updated
  - If no free page is left, a *page is selected to be replaced* (based on usage)
  - The replaced page is written on the drum
    - to minimize drum latency effect, the first empty page on the drum was selected
  - The *page table is updated* to point to the new location of the page on the drum

Linear Page Table

- Page Table Entry (PTE) contains:
  - A bit to indicate if a page exists
  - 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
- OS sets the Page Table Base Register whenever active user process changes



Size of Linear Page Table

With 32-bit addresses, 4-KB pages & 4-byte PTEs:

⇒ 2^{20} PTEs, i.e, 4 MB page table per user

⇒ 4 GB of swap needed to back up full virtual address space

Larger pages?

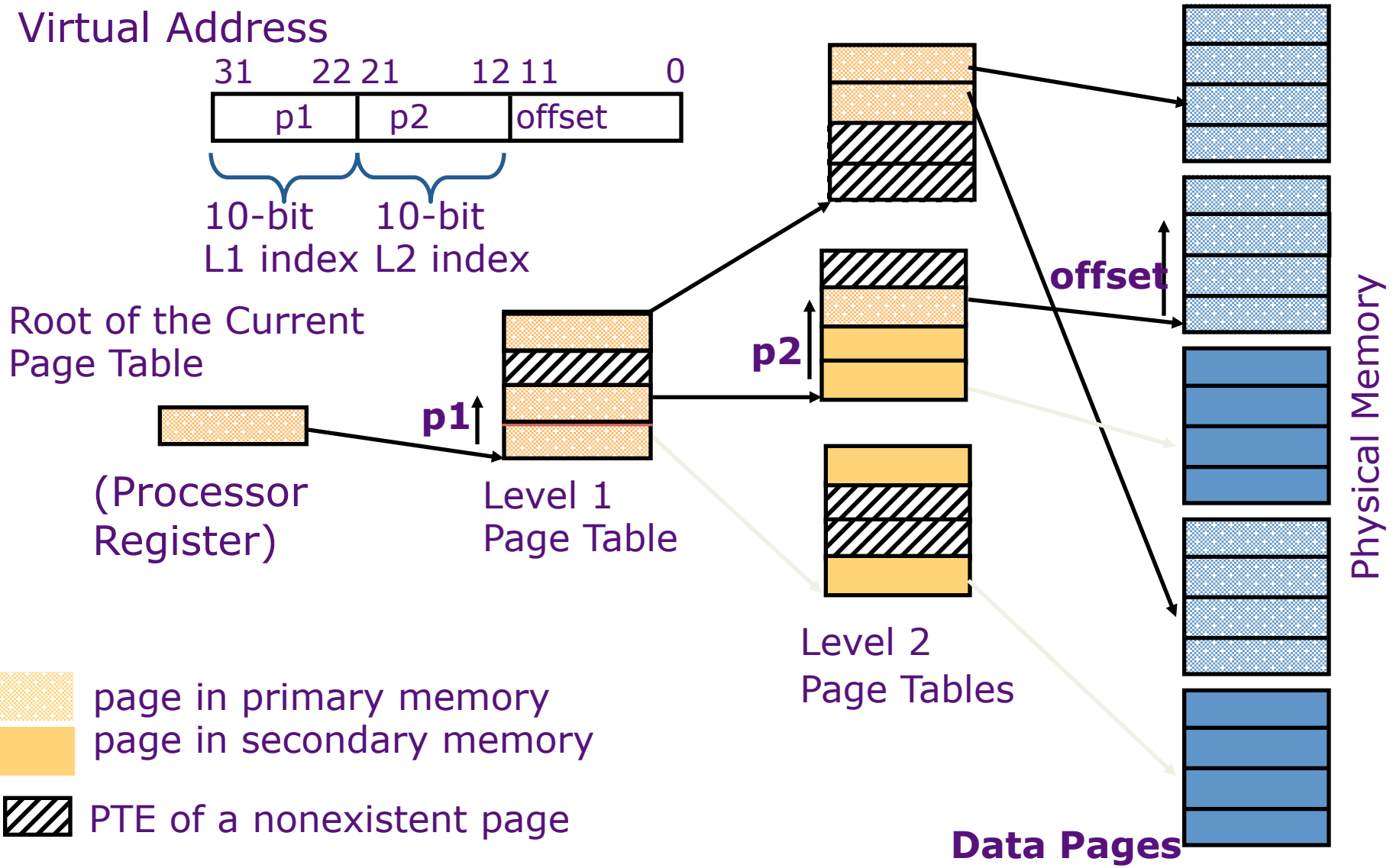
- Internal fragmentation (Not all memory in page is used)
- Larger page fault penalty (more time to read from disk)

What about 64-bit virtual address space???

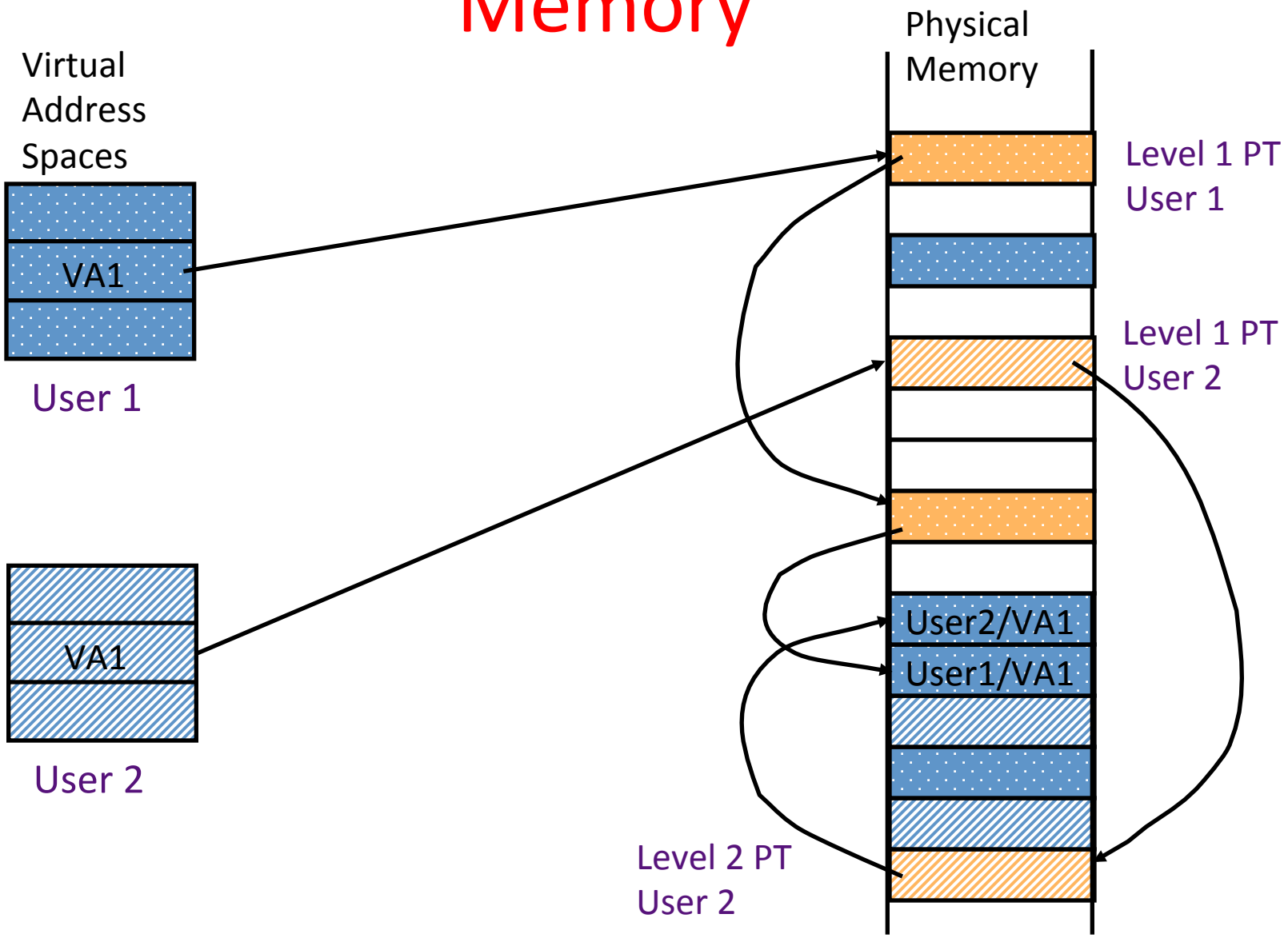
- Even 1MB pages would require 2^{44} 8-byte PTEs (35 TB!)

What is the "saving grace" ?

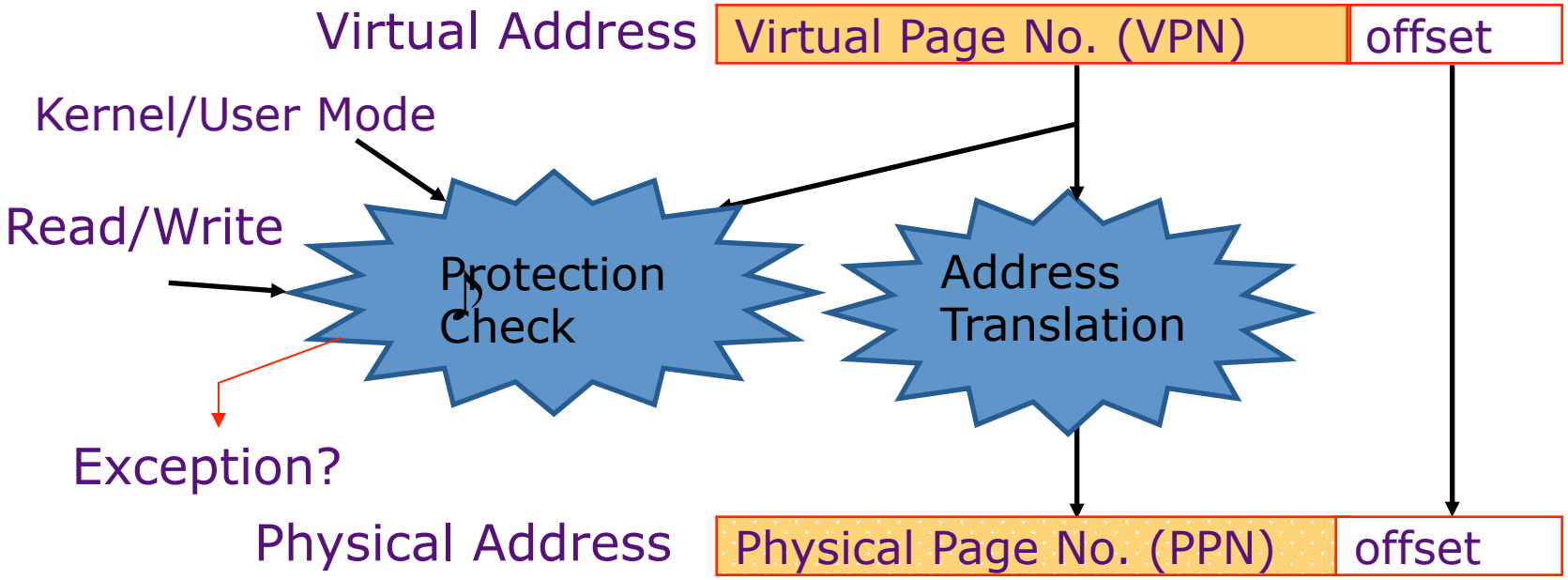
Hierarchical Page Table



Two-Level Page Tables in Physical Memory



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)

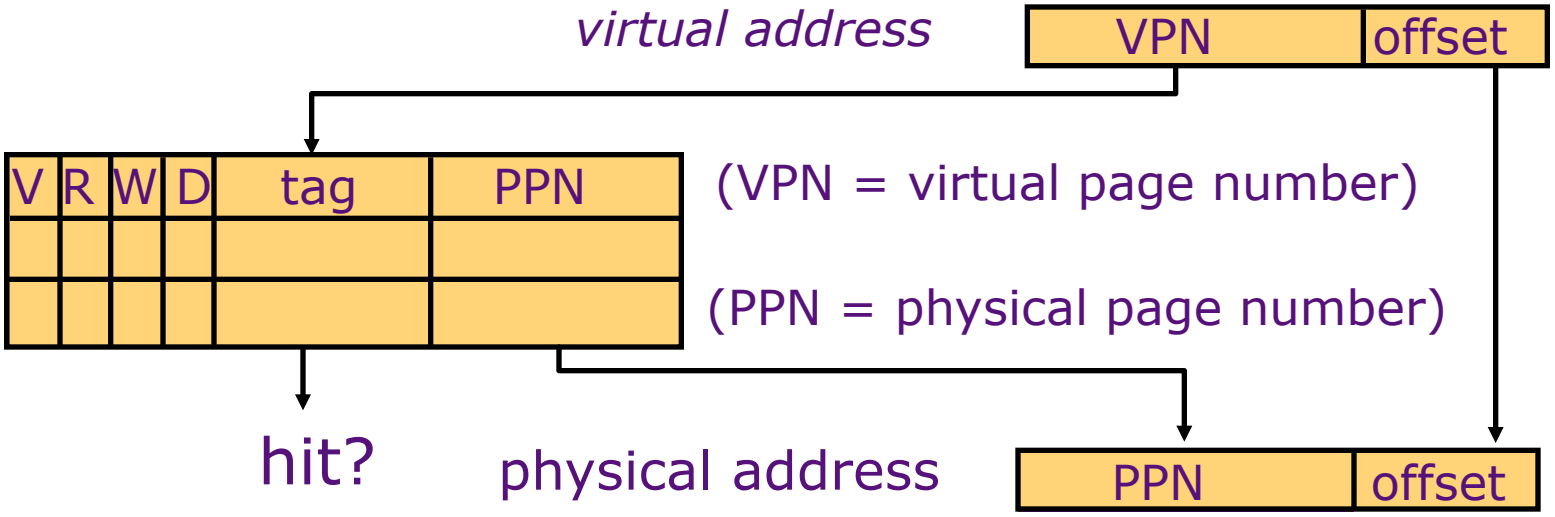
Address translation is very expensive!

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

Solution: *Cache translations in TLB*

TLB hit ⇒ *Single-Cycle Translation*

TLB miss ⇒ *Page-Table Walk to refill*



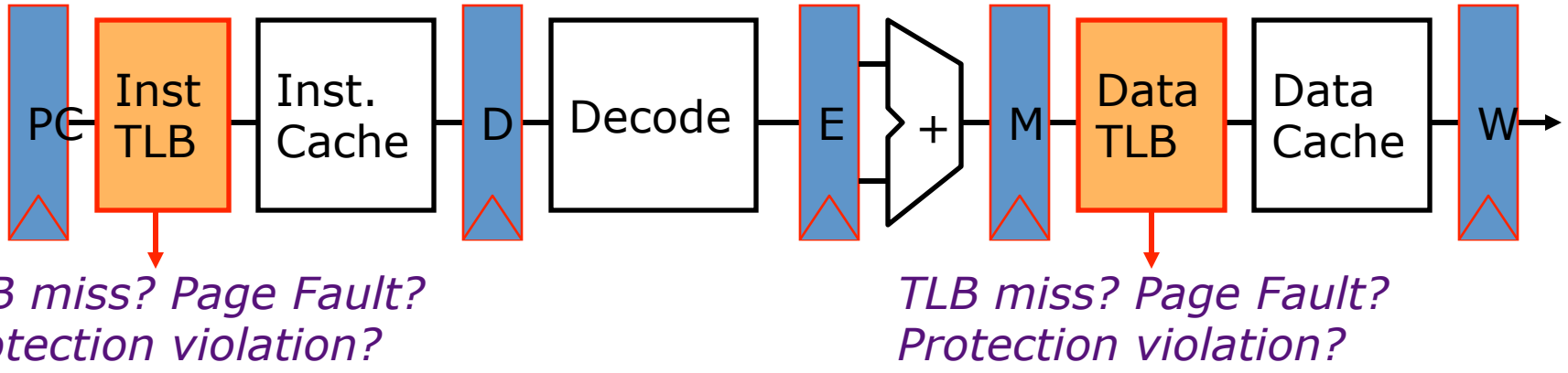
TLB Designs

- 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
- No process information in TLB*?*
- TLB Reach: Size of largest virtual address space that can be simultaneously mapped by TLB

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

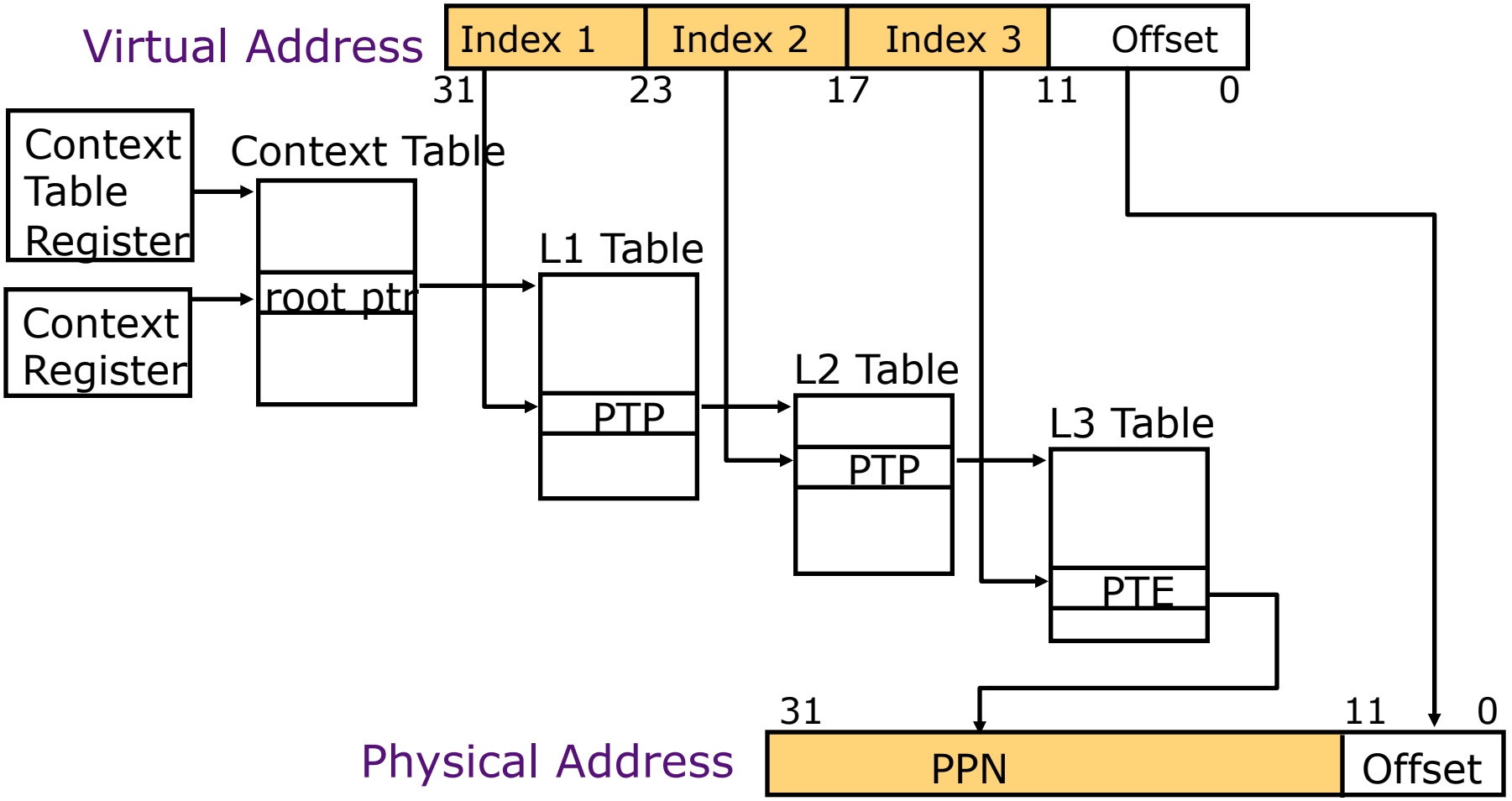
TLB Reach = _____*?*

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

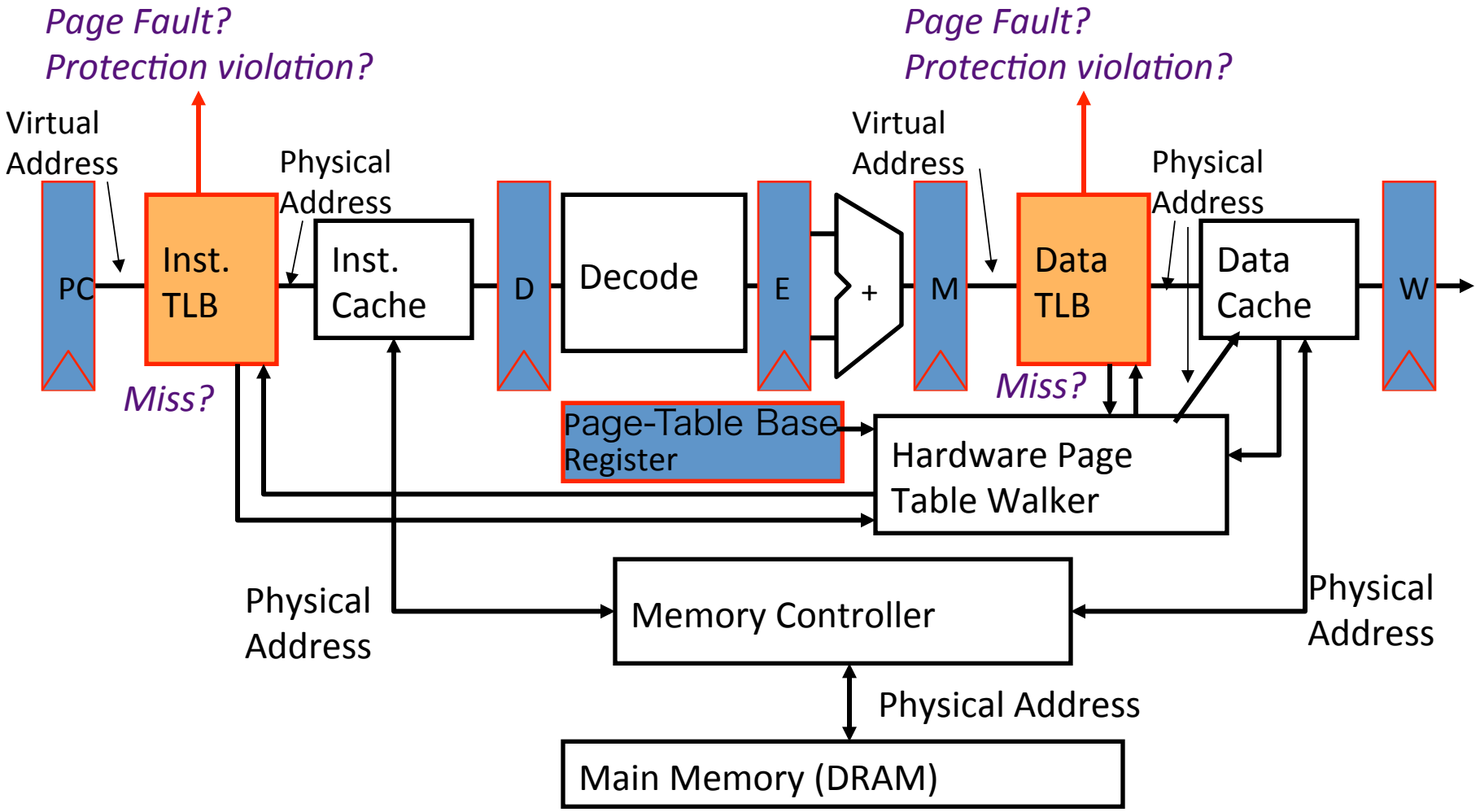
Hierarchical Page Table Walk: SPARC v8



MMU does this table walk in hardware on a TLB miss

Page-Based Virtual-Memory Machine

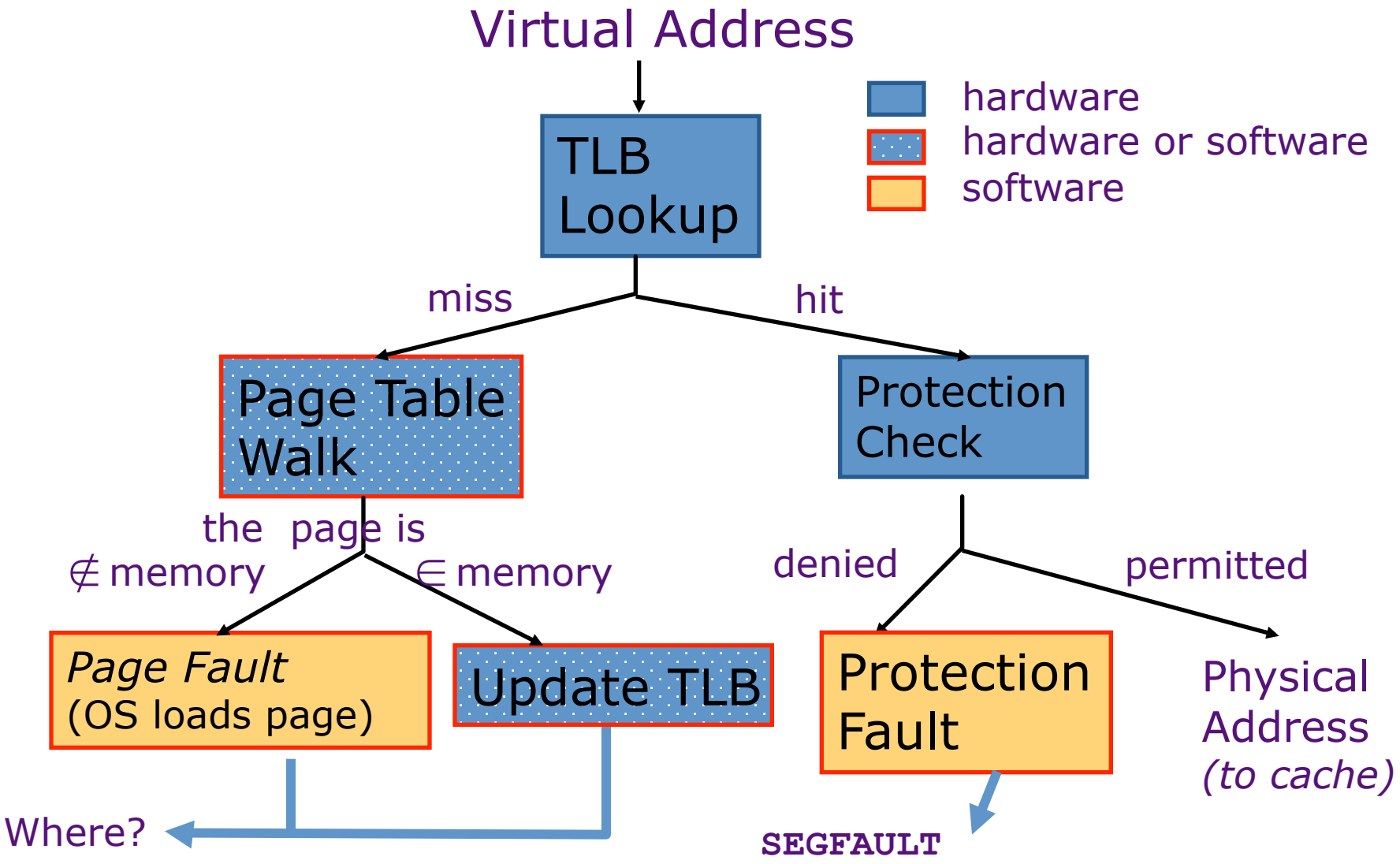
(Hardware Page-Table Walk)



- Assumes page tables held in untranslated physical memory

Address Translation:

putting it all together



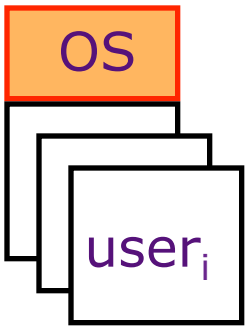
Modern Virtual Memory Systems

Illusion of a large, private, uniform store

Protection & Privacy

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

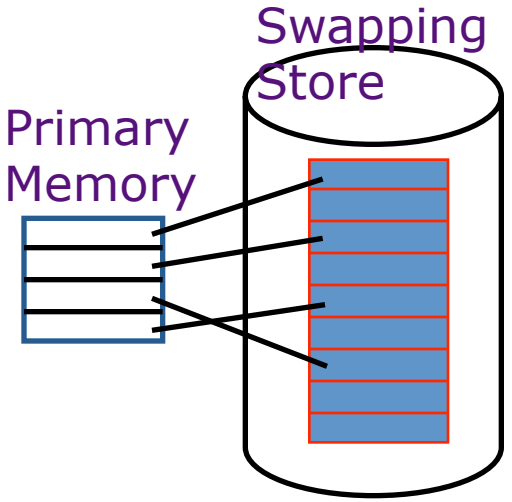
page table ≡ name space



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

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
- **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 for Project 4