Computer Architecture

Memory Management

Ting-Jung Chang

NYCU CS

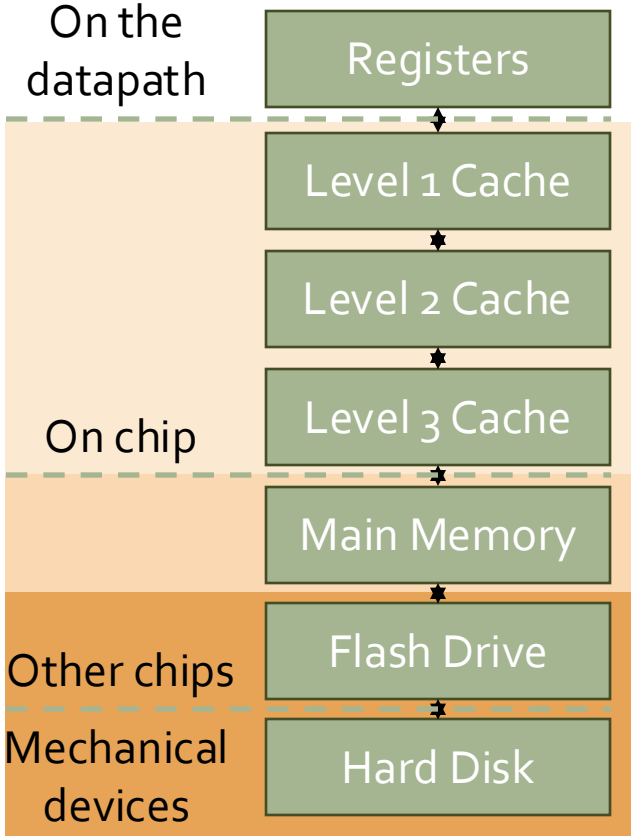
Course Administration

- Lab4 and PS4 released on e3

Recap: Cache Optimizations

Cache Optimization	Miss Rate	Miss Penalty	Hit Time	Bandwidth
Pipelined & banked caches			–	+
Writer Buffer		+		
Multilevel Cache	+	+		
Victim Cache	+	+		
Prefetching	+	+		
Compiler Optimization	+			
Non-blocking Cache		+		+
Critical Word First/Early Restart		+		

Memory Hierarchy



Access Time	Capacity	Managed by
1 cycle	1KB	Software/Compiler
Before: Hardware Caches		Hardware
		Hardware
		Hardware
Today: Virtual Memory		Software/OS
		Software/OS
		Software/OS

Extending the Memory Hierarchy

- Problem: DRAM vs disk has much more extreme differences than SRAM vs DRAM
  - Access latencies:
    - DRAM ~10-100x slower than SRAM
    - Disk ~100,000x slower than DRAM
  - Importance of sequential accesses
    - DRAM: Fetching successive words ~5x faster than first word
    - Disk: Fetching successive words ~100,000x faster than first word
- Result: Design decisions driven by enormous cost of misses
  - Associativity: High, minimize miss rate
  - Block size: Large, amortize cost of a miss over multiple words
  - Write policy: Write back, minimize number of writes

Memory Management

- From early absolute addressing schemes, to modern virtual memory systems with support for virtual machine monitors
- Can separate into orthogonal functions:
  - Translation (mapping of virtual address to physical address)
  - Protection (permission to access word in memory)
  - Virtual Memory (transparent extension of memory space using slower disk storage)
- But most modern systems provide support for all the above functions with a single page-based system

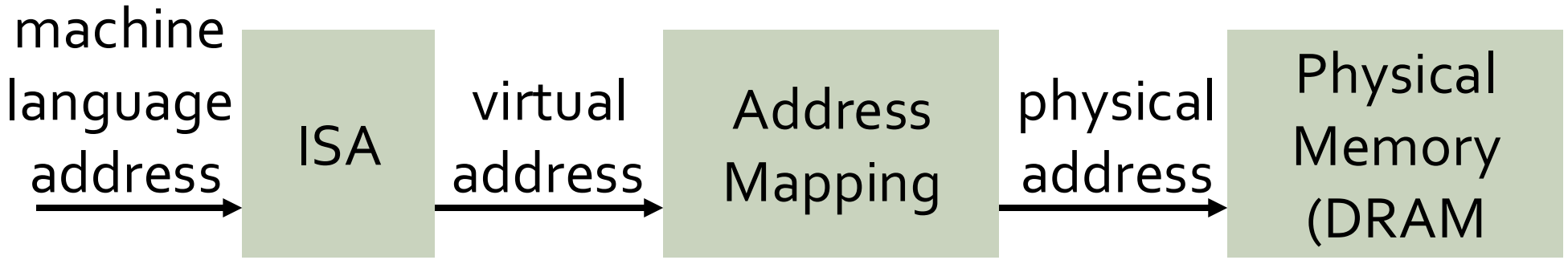
Agenda

- Evolution of Virtual Memory
- Modern VM Implementation
- TLB & Cache Organization
- Modern Usage

Memory Management

- The Fifties
  - Absolute Addresses
  - Dynamic address translation
- The Sixties
  - Atlas and Demand Paging
  - Paged memory systems and TLBs
- Modern Virtual Memory Systems

Names for Memory Locations



- Machine language address
  - as specified in machine code
- Virtual address
  - ISA specifies translation of machine code address into virtual address of program variable (sometimes called effective address)
- Physical address
  - Operating system specifies mapping of virtual address into name for a physical memory location

Absolute Addresses

virtual address = physical memory address

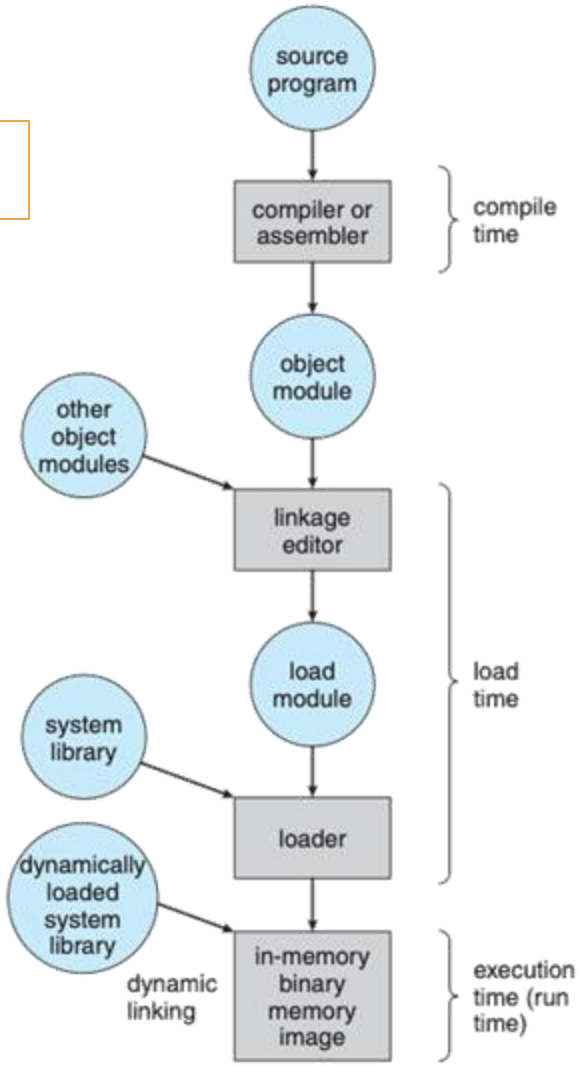
EDSAC, early 50's

- Only one program ran at a time, with unrestricted access to entire machine (RAM + I/O devices)
- Addresses in a program depended upon where the program was to be loaded in memory
- **But** it was more convenient for programmers to write location-independent subroutines
  - Different programs use different combination of routines

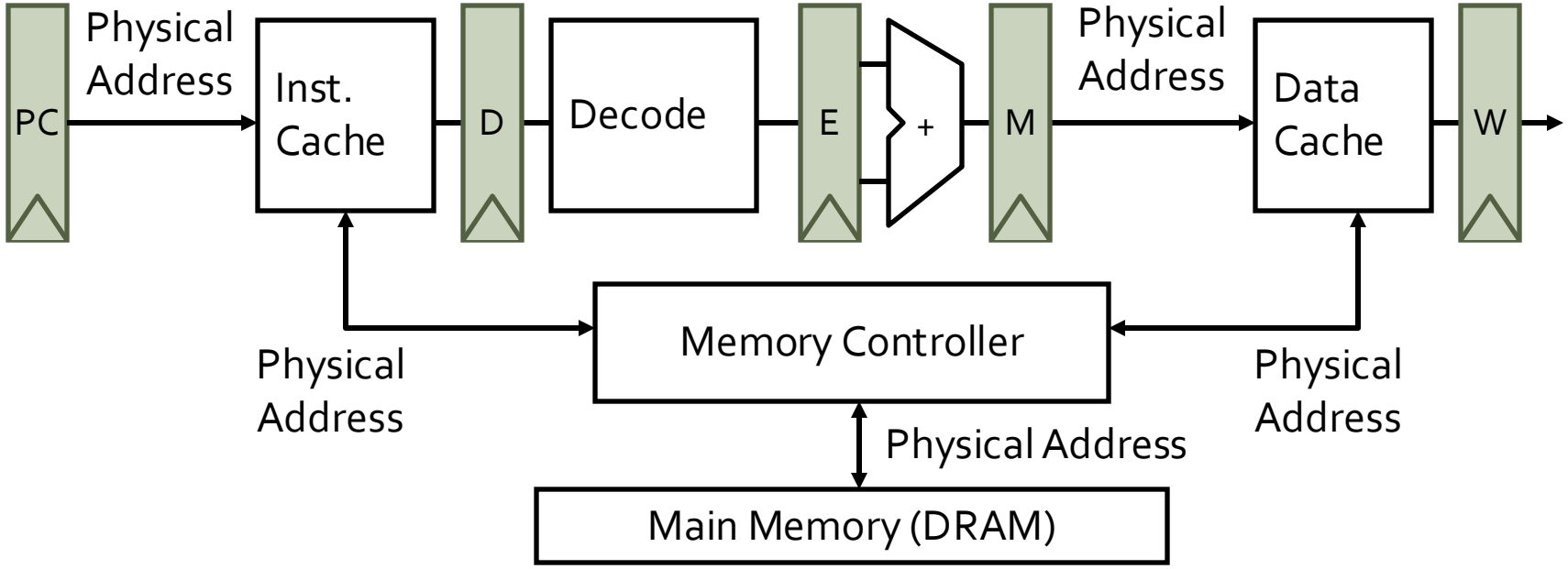
How could location independence be achieved?

Linker and/or loader modify addresses of subroutines and callers when building a program memory image

[Operating System Concepts, 9th ED]



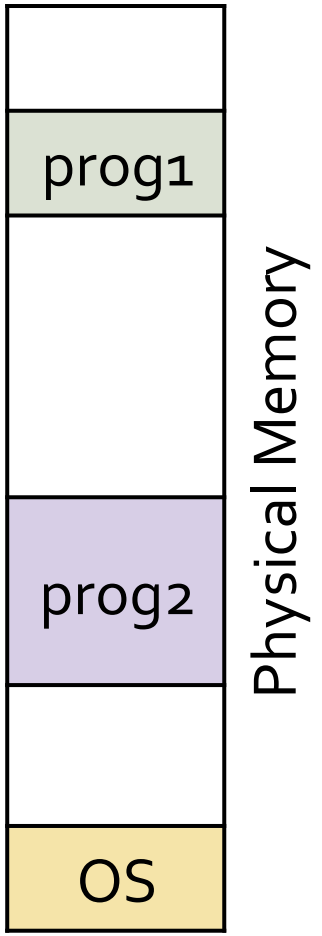
Bare Machine



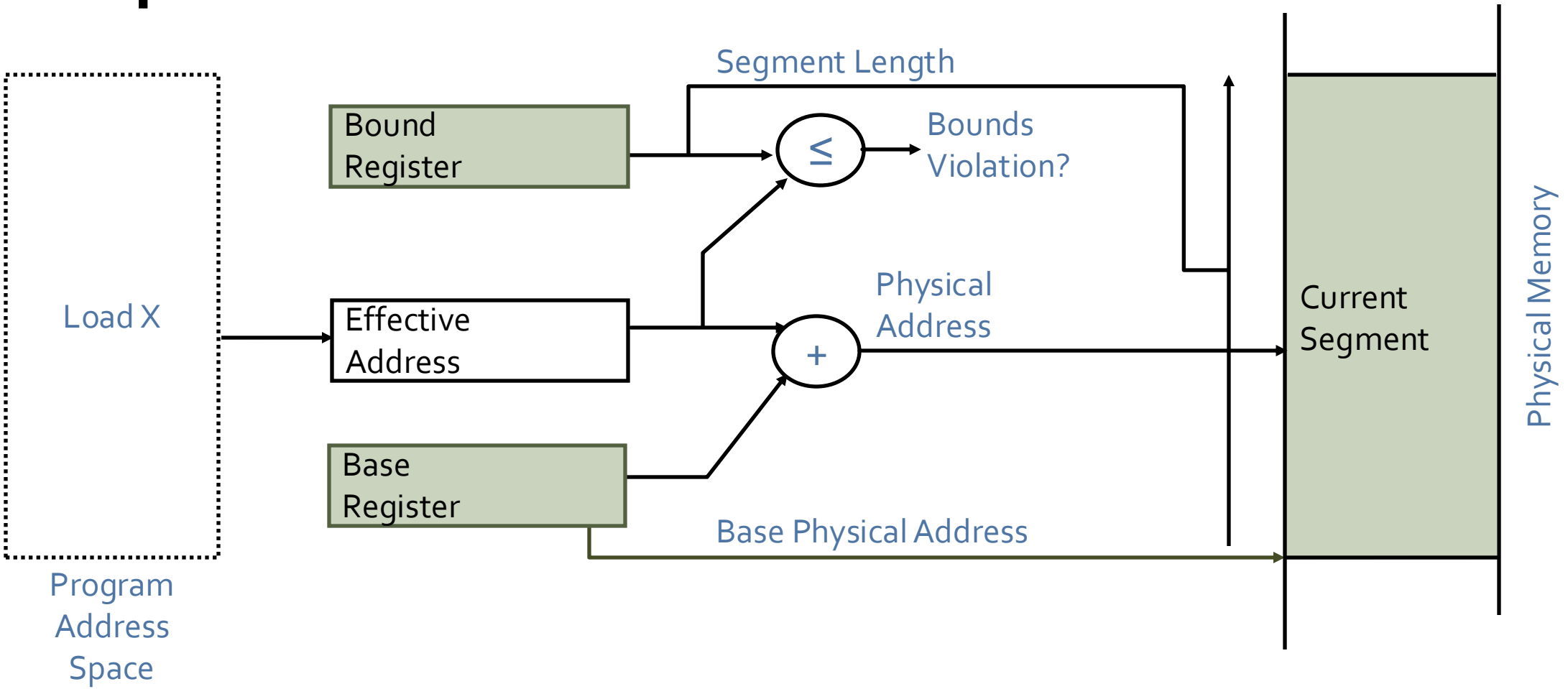
- In a bare machine, the only kind of address is a physical address

Dynamic Address Translation

- Motivation
  - In early machines, I/O was slow and each I/O transfer involved the CPU (programmed I/O)
  - Higher throughput possible 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** software to manage context switches between multiple programs

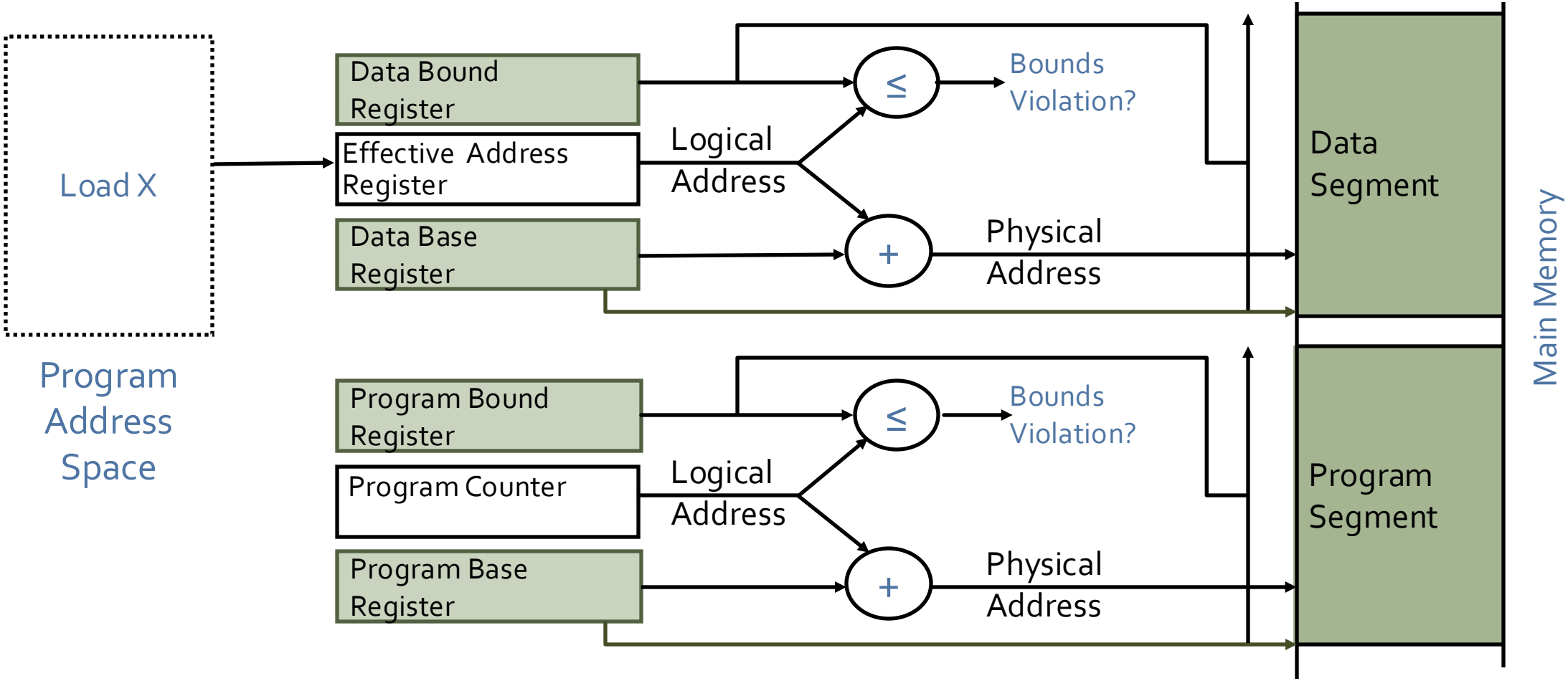


Simple Base and Bound Translation



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

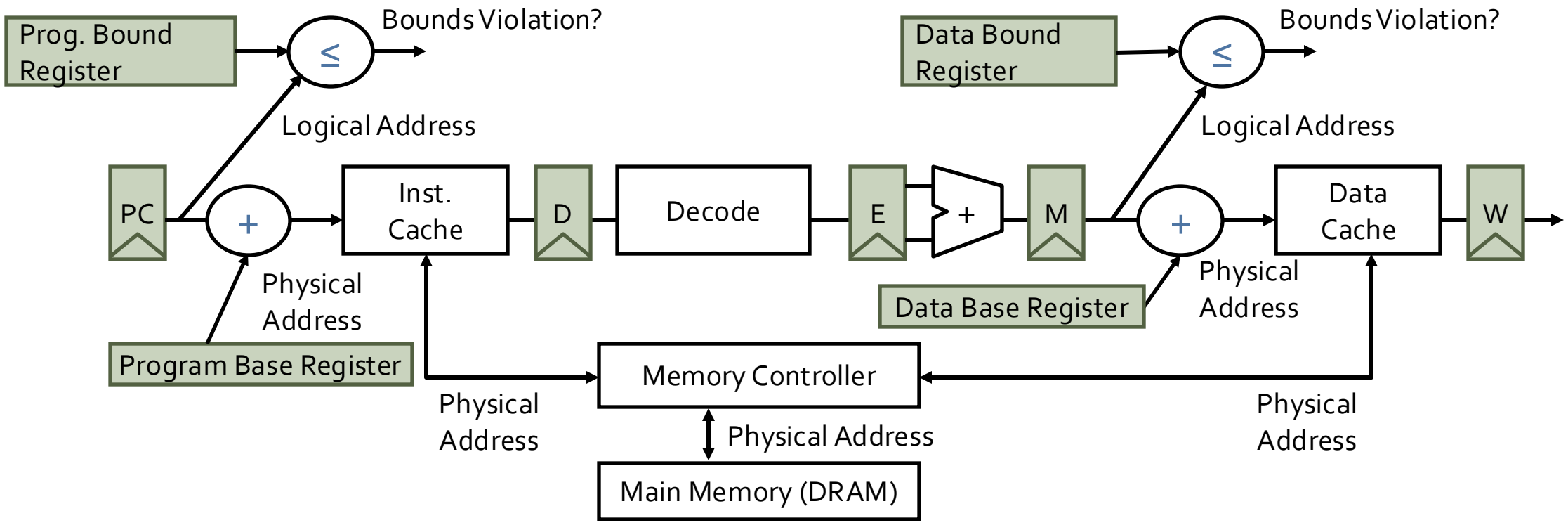
Separate Areas for Program and Data



What is an advantage of this separation?

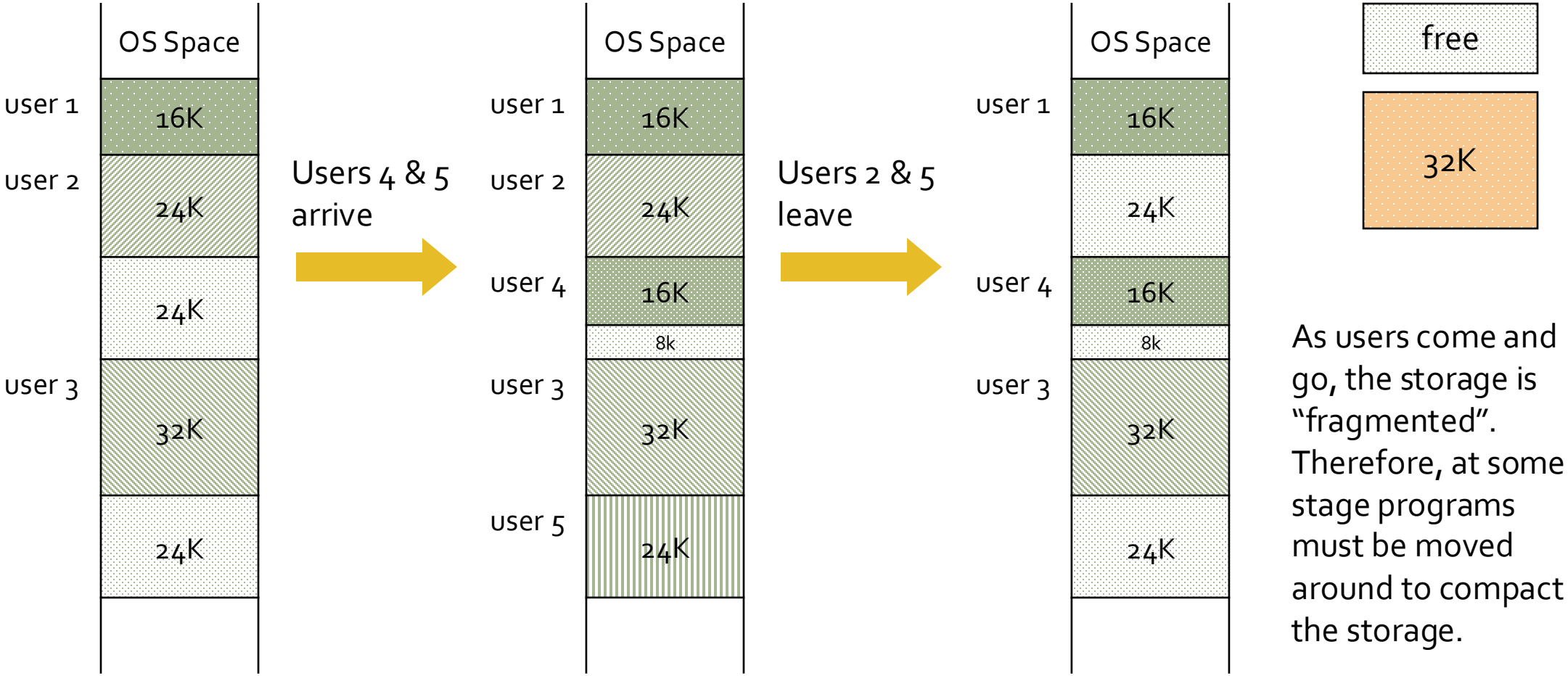
(Scheme used on all Cray vector supercomputers prior to X1, 2002)

Base and Bound Machine



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

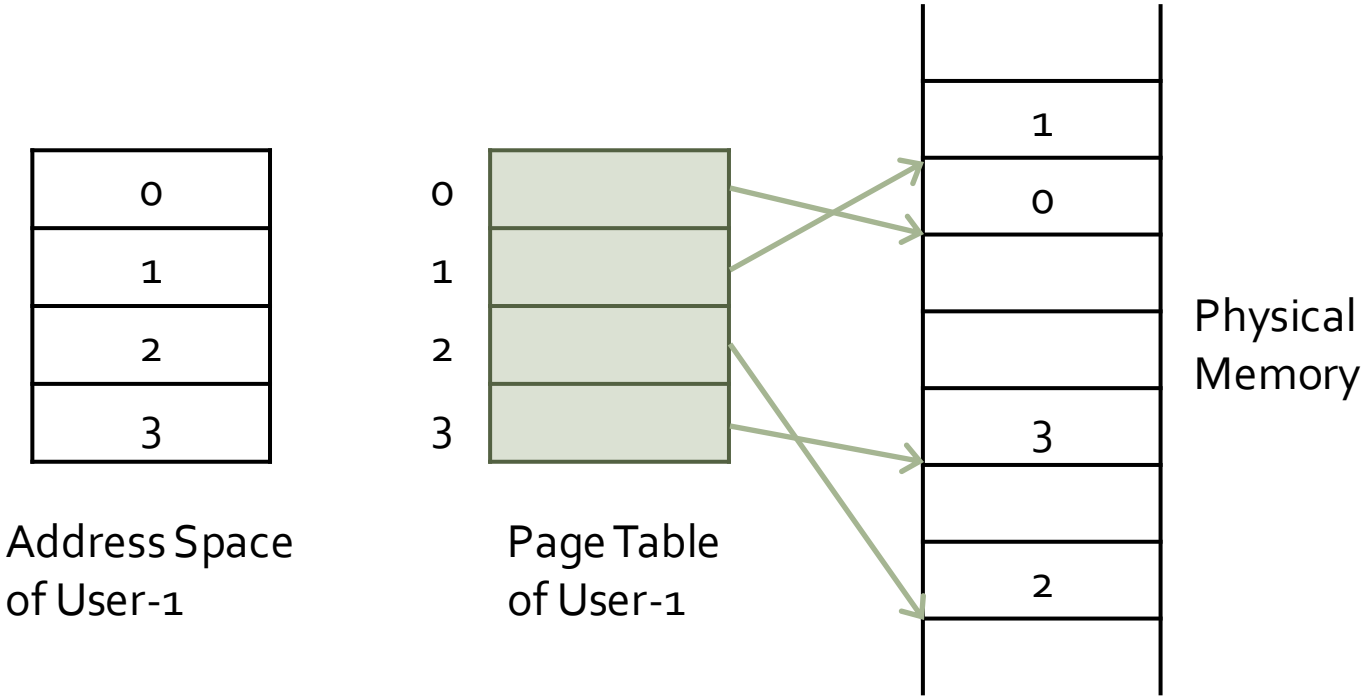
Memory Fragmentation



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

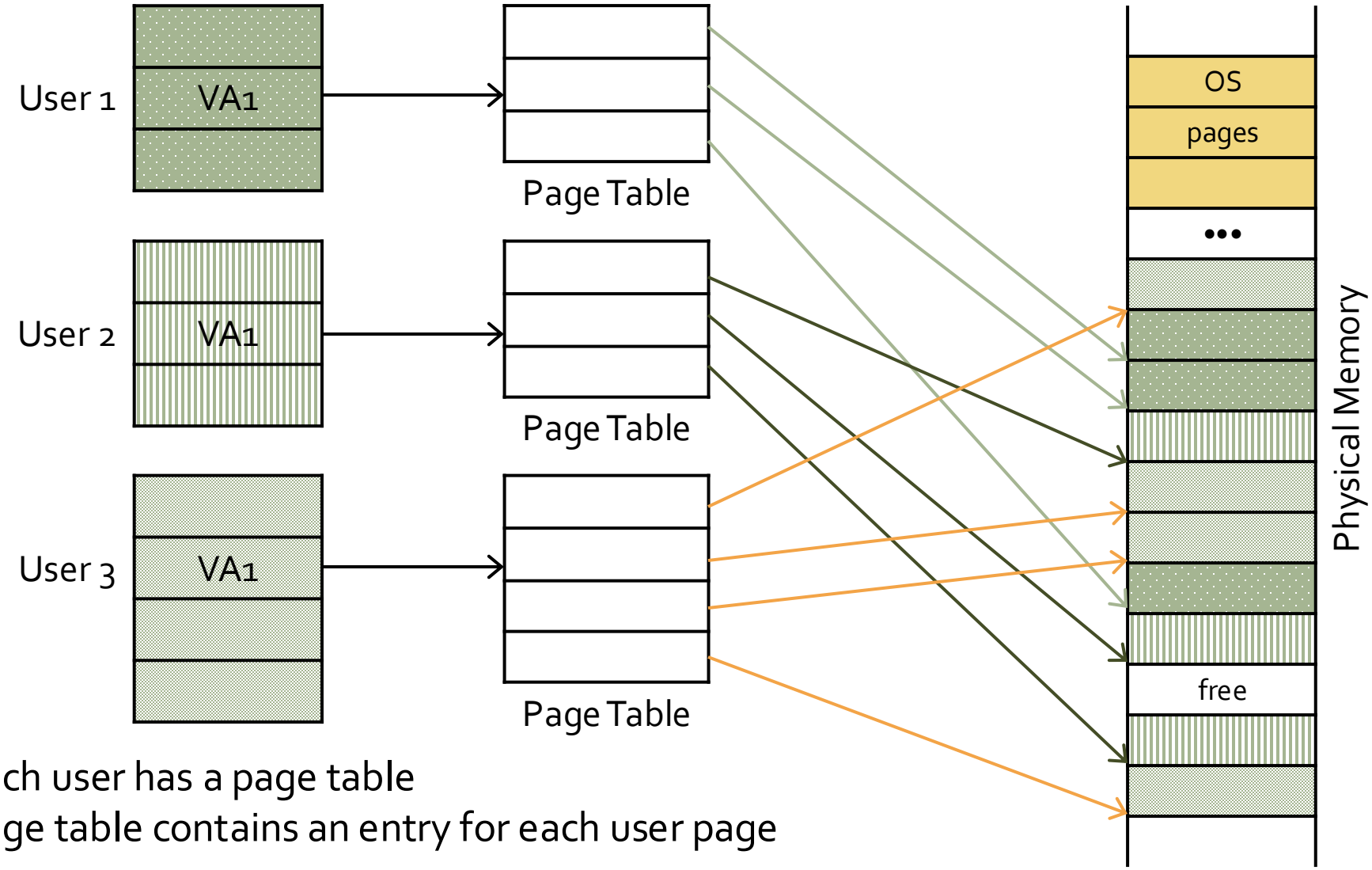
Paged Memory Systems

- Processor-generated address can be interpreted as a pair <page number, offset>: 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

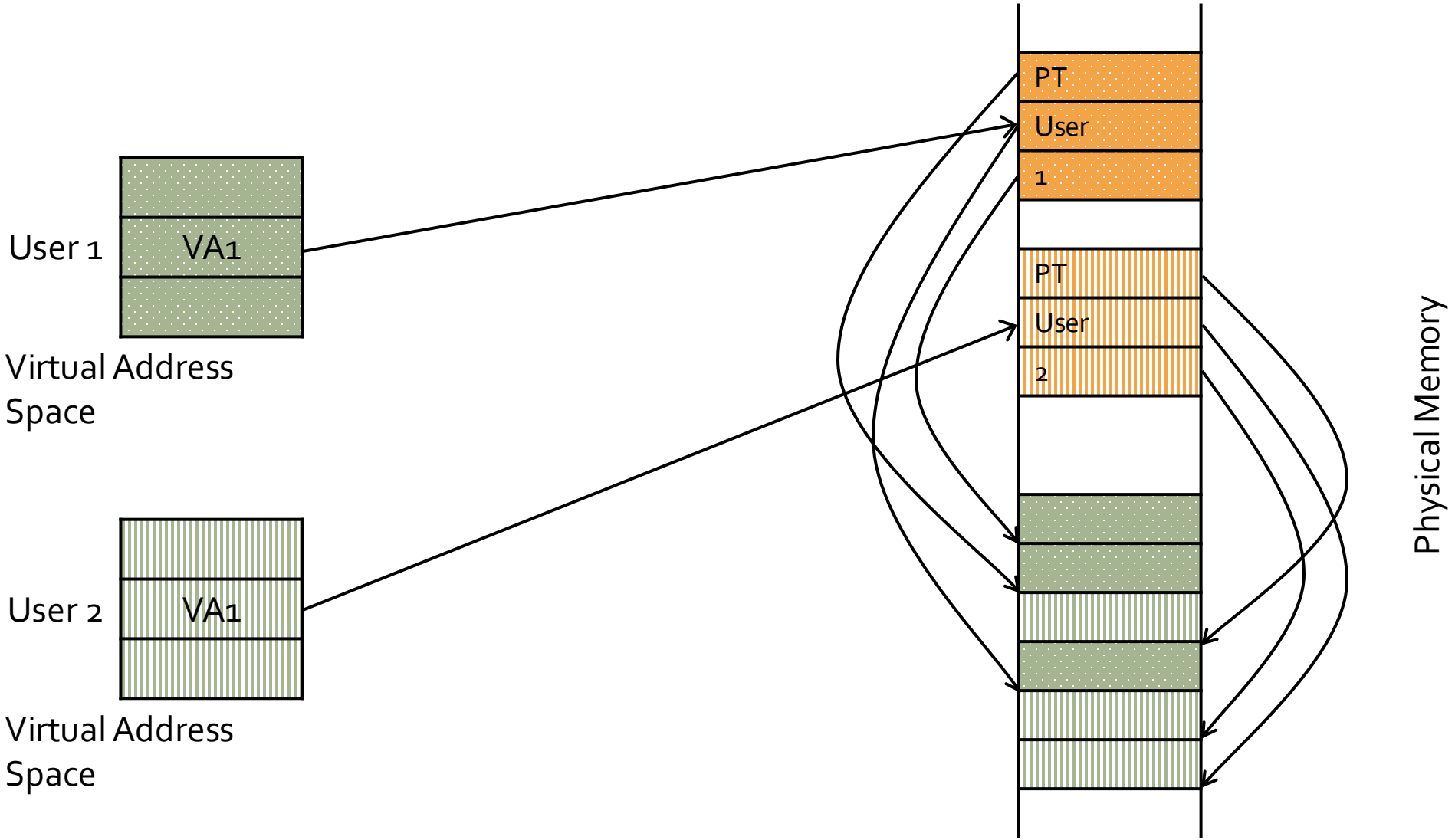
Paging Simplifies Allocation

- Fixed-size pages can be kept on OS free list and allocated as needed to any process
- Process memory usage can easily grow and shrink dynamically
- Paging suffers from **internal fragmentation** where not all bytes on a page are used
  - Much less of an issue than external fragmentation or compaction for common page sizes (4-8KB)
  - But one reason that many oppose move to larger page sizes

Where Should Page Tables Reside?

- Space required by the page tables (PT) is proportional to the address space, number of users, (inverse to) size of each page, …
  - Space requirement is large
  - Too expensive to keep in registers
- Idea: Keep PT of the current user in special registers
  - may not be feasible for large page tables
  - Increases the cost of context swap
- 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!**
  - Storage space to store PT grows with size of memory

Page Tables in Physical Memory

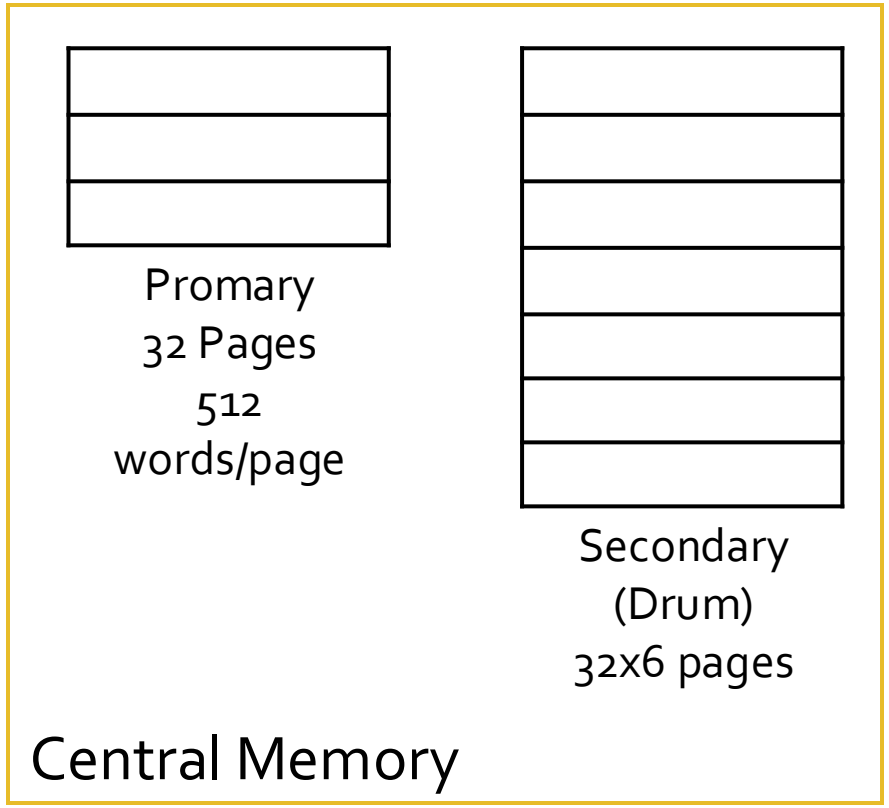


A Problem in Early Sixties

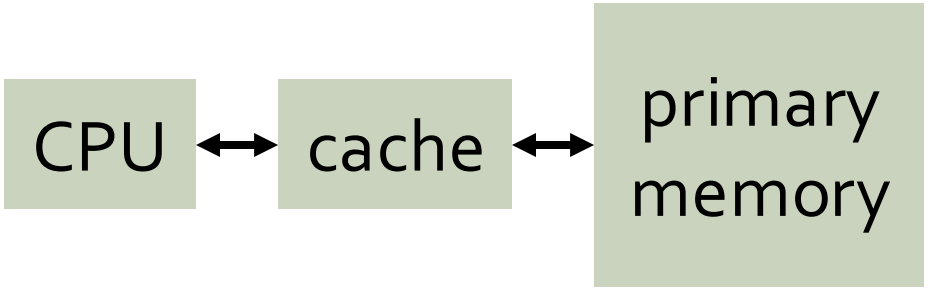
- There were many applications whose data could not fit in the main memory, e.g., payroll
- Paged memory system reduced fragmentation but still required the whole program to be resident in the main memory
- Programmers moved the data back and forth from the secondary store by overlaying it repeatedly on the primary store

Demand Paging in Atlas (1962)

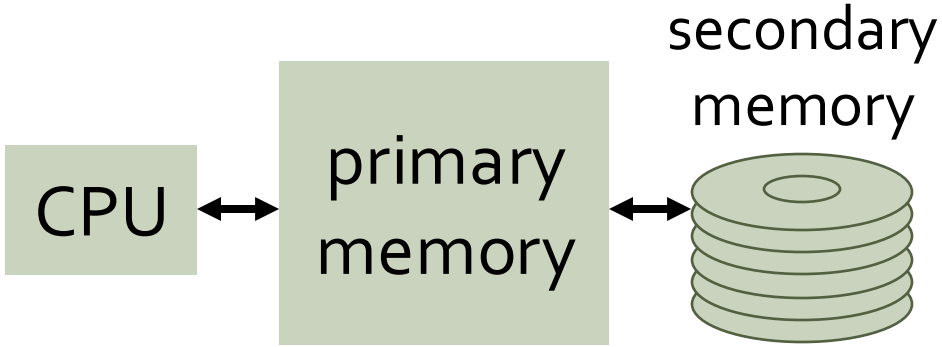
- A page from secondary storage is brought into the primary storage whenever it is (**implicitly**) demanded by the processor
- User sees the storage size of the secondary storage, since data transfer happens automatically
- Primary memory as a cache for secondary memory



Caching vs. Demand Paging



- Caching
- cache entry
- cache block (~32 bytes)
- cache miss rate (1% to 20%)
- cache hit (~1 cycle)
- cache miss (~100 cycles)
- a miss is handled in hardware



- Demand paging
- page frame
- page (~4K bytes)
- page miss rate (<0.001%)
- page hit (~100 cycles)
- page miss (~5M cycles)
- a miss is handled mostly in software

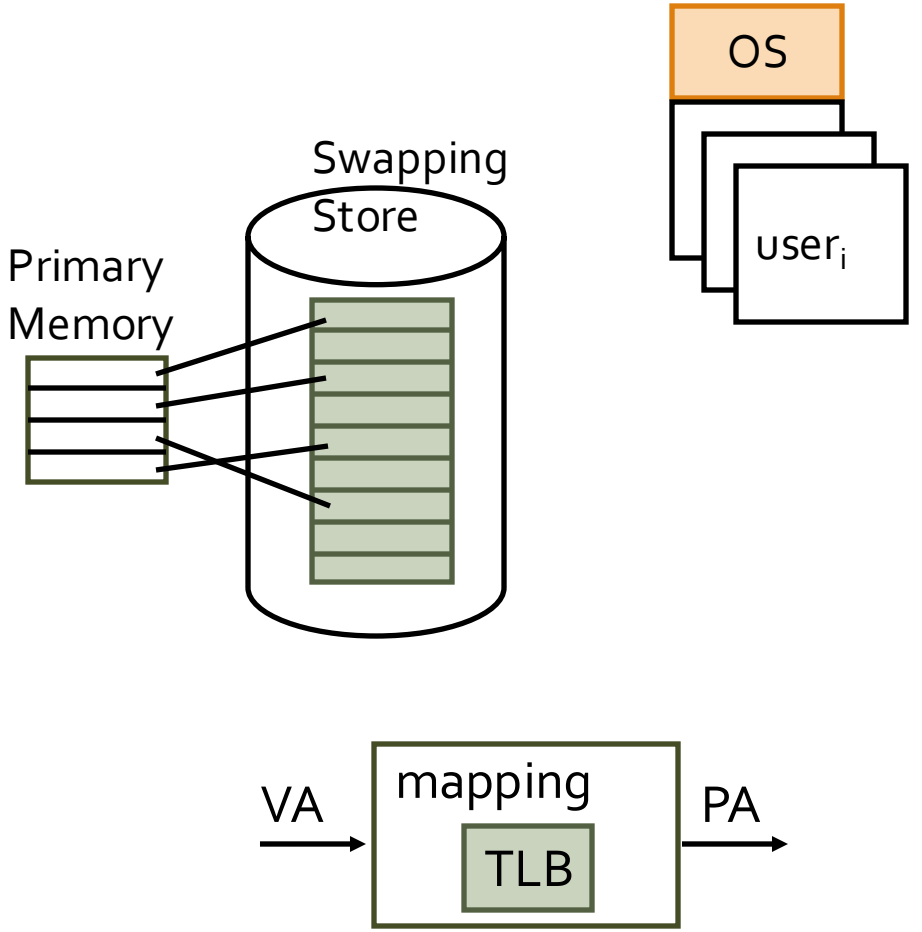
Modern Virtual Memory Systems

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

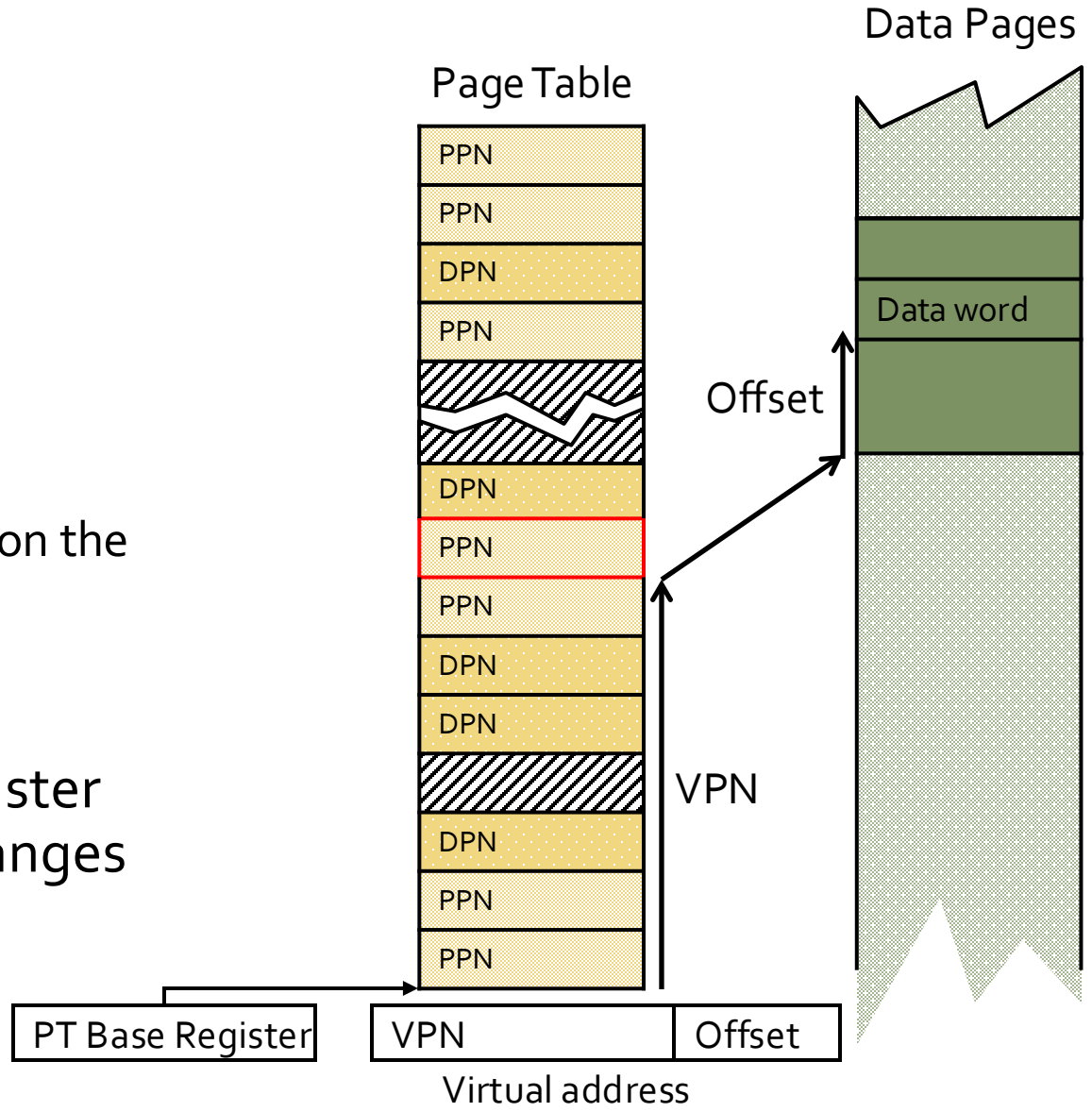


Agenda

- Evolution of Virtual Memory
- Modern VM Implementation
- TLB & Cache Organization
- Modern Usage

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:

- $\Rightarrow$ 2^{20} PTEs, i.e, 4 MB page table per user per process
- $\Rightarrow$ 4 GB of swap needed to back up full virtual address space

Larger pages?

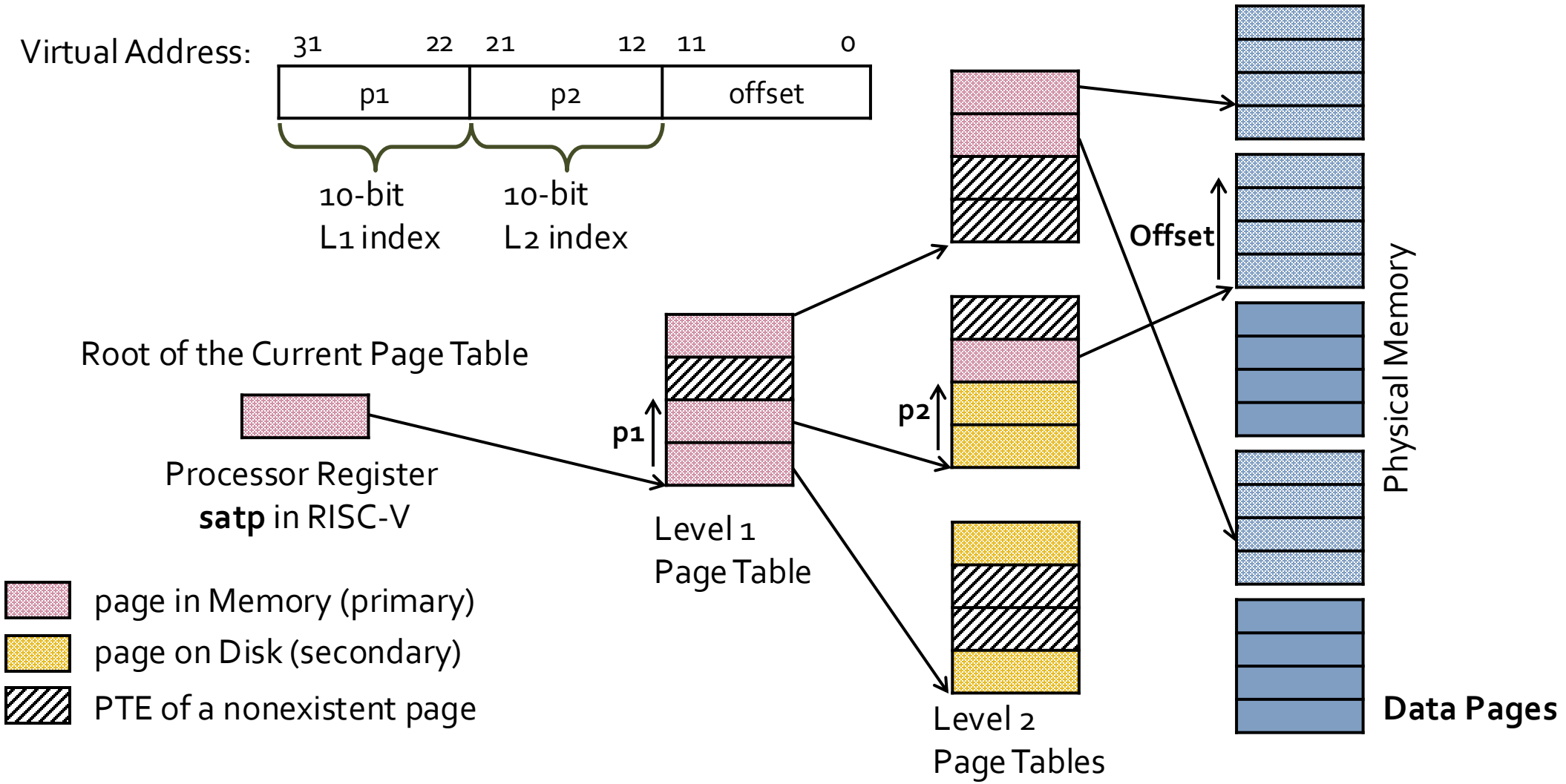
- Internal fragmentation (Not all memory in page is used)
- Larger page fault penalty (more time to read from disk)

What about 64-bit virtual address space???

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

What is the "saving grace" ?

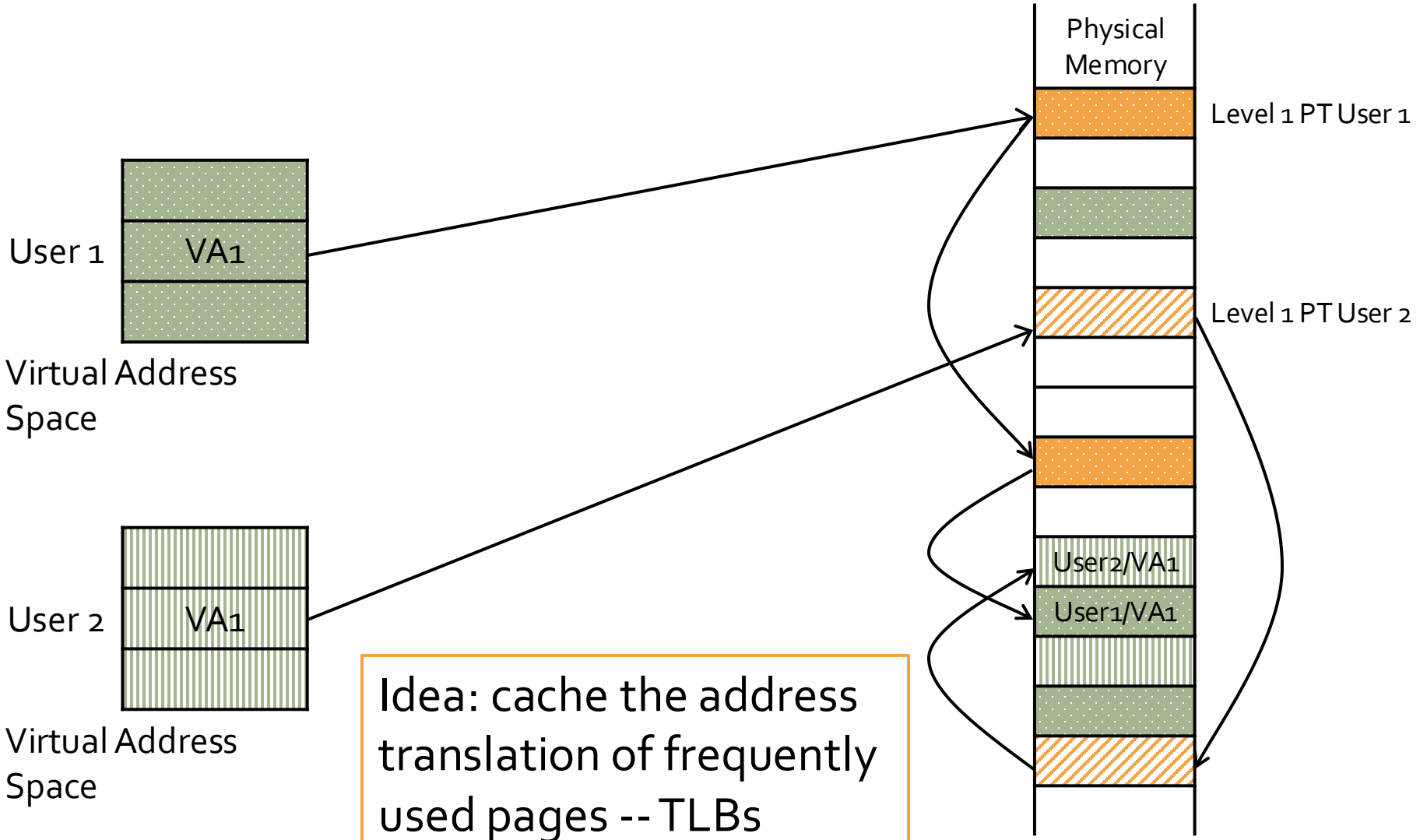
Hierarchical Page Table



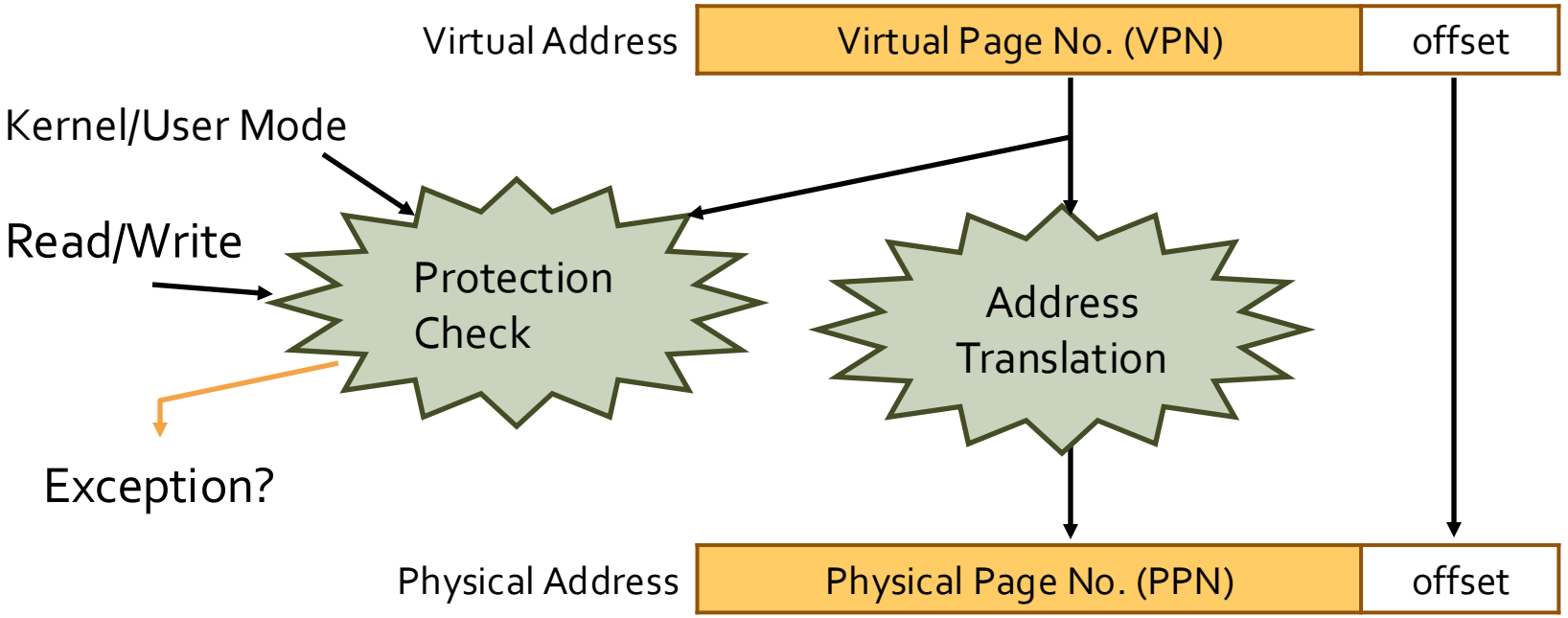
Real-world ISAs – RISC-V

- RISC-V defines three different virtual memory specifications
  - Sv32: Two level hierarchy, 32-bit virtual, 34-bit physical address
  - Sv39: Three level, 39-bit virtual, 56-bit physical address
  - Sv48: Four level, 48-bit virtual, 56-bit physical address
  - Up to chip designers to choose what to support
  - OS can learn which is implemented by reading the CSR "**satp**" (Supervisor Address Translation and Protection)
- Some specs have different PTE sizes
  - Sv32 has 32 bit PTEs
  - Sv39 and Sv48 has 64-bit PTEs

Two-Level Page Tables in Physical Memory



Address Translation & Protection



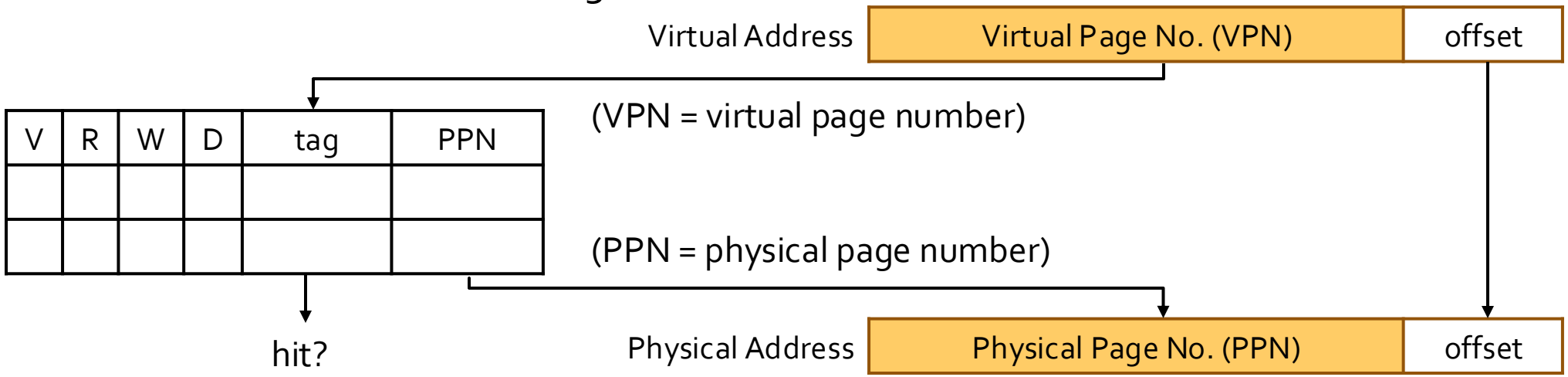
- Every instruction and data access needs address translation and protection checks
- A good Virtual Memory (VM) design needs to be fast (~ one cycle) and space efficient

Translation Lookaside Buffers (TLB)

- Problem: Address translation is very expensive!
  - In a hierarchical 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 (Clock Algorithm) or FIFO replacement policy
- Keep process information in TLB?
  - No process id ⟶ Must flush on context switch
  - Tag each entry with process id ⟶ No flush, but costlier
- 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

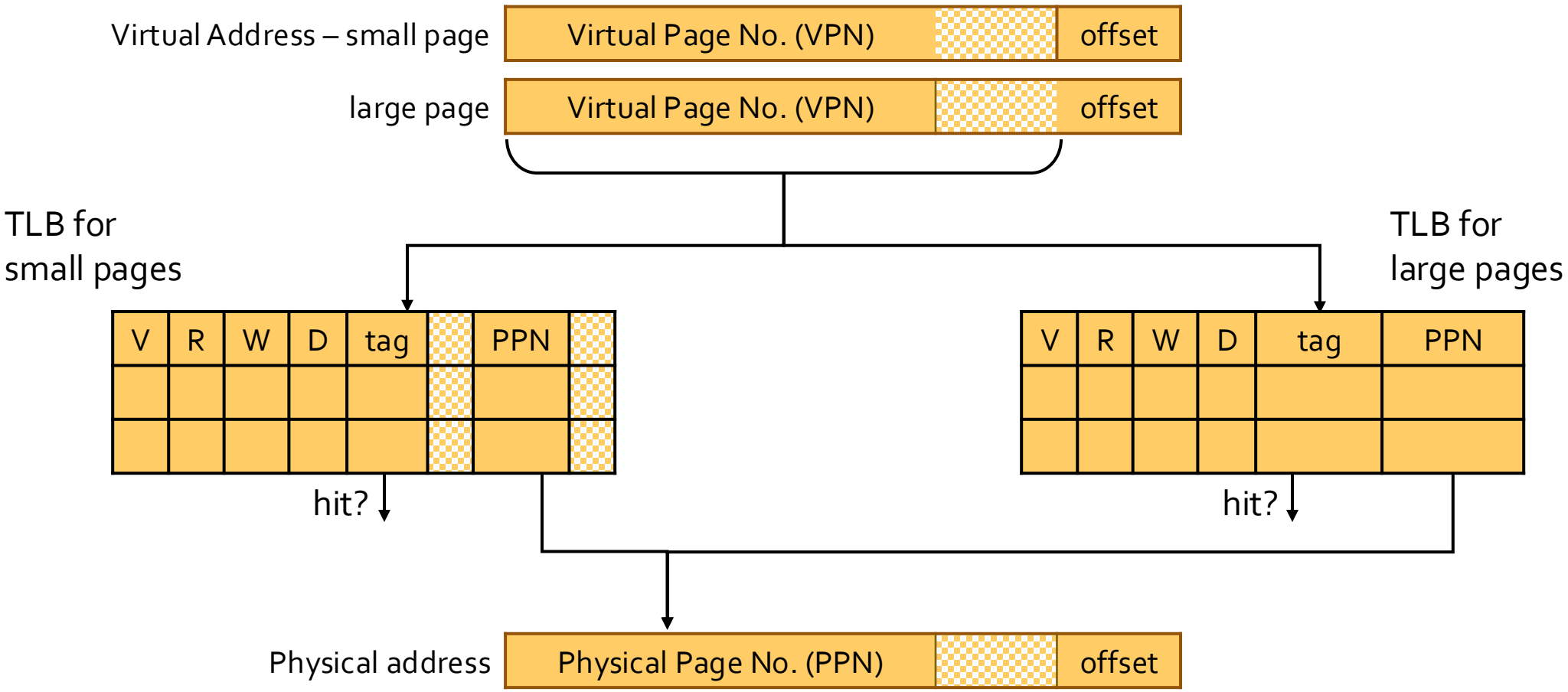
  TLB Reach = 64 entries * 4 KB = 256 KB (if contiguous) ?

TLB Extensions

- Address Space Identifier (ASID)
  - Allow TLB Entries from multiple processes to be in TLB at same time. ID of address space (Process) is matched on.
  - Global Bit (G) can match on all ASIDs
- Variable Page Size (PS)
  - Can increase reach on a per page basis
- Kernel/User (K)

K	V	R	W	D	tag	PPN	PS	G	ASID

TLB Supporting Multiple Page Sizes



Pros/cons compared to unified TLB?

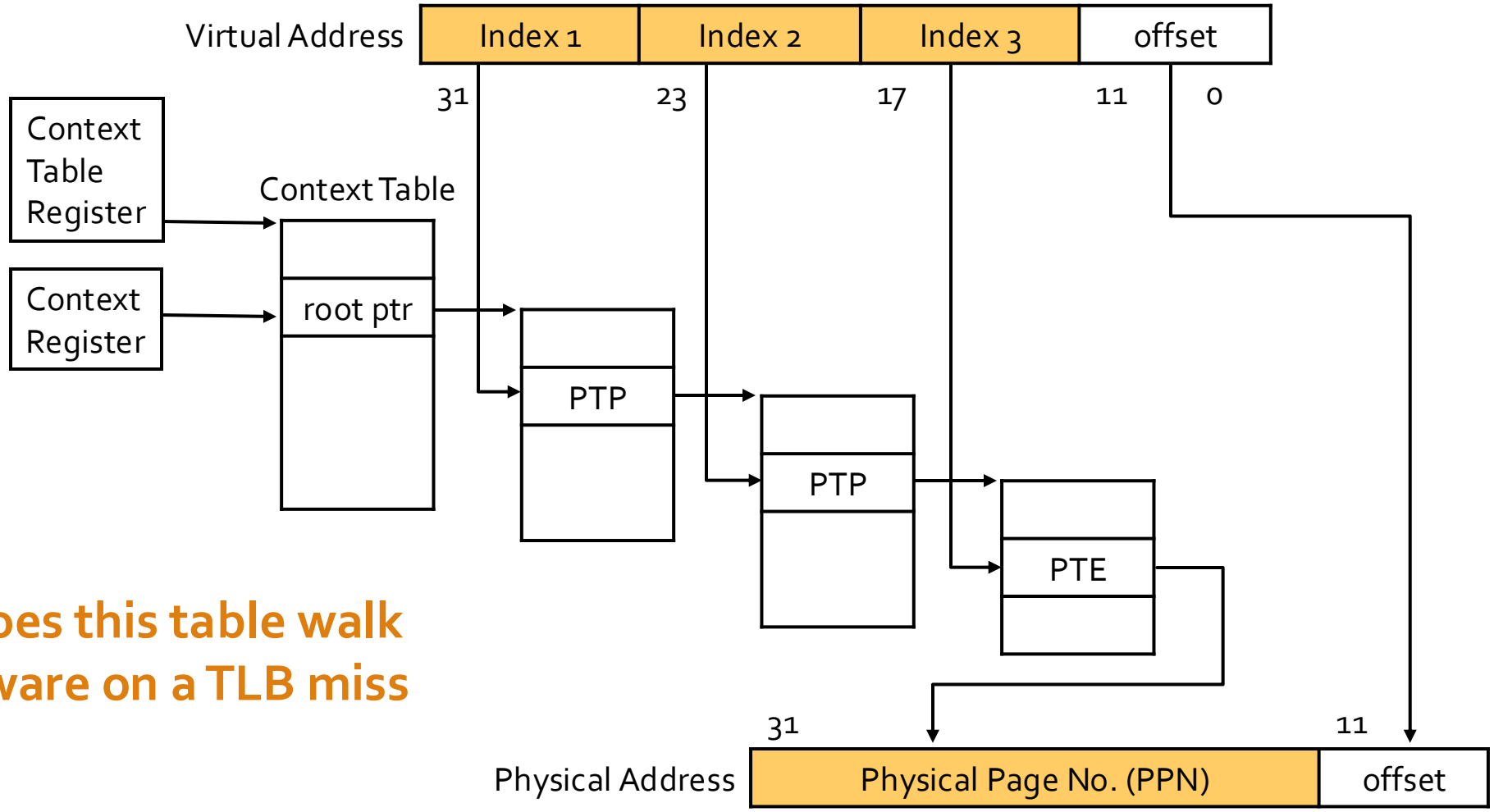
Handling a TLB Miss

- Software (MIPS, Alpha)
  - TLB miss causes an exception and the operating system walks the page tables and reloads TLB. A privileged "untranslated" addressing mode used for walk
  - Software TLB miss can be very expensive on out-of-order superscalar processor as requires a flush of pipeline to jump to trap handler

- Hardware (SPARC v8, x86, PowerPC)
  - A memory management unit (MMU) walks the page tables and reloads the TLB
  - If a missing (data or PT) page is encountered during the TLB reloading, MMU gives up and signals a Page-Fault exception for the original instruction

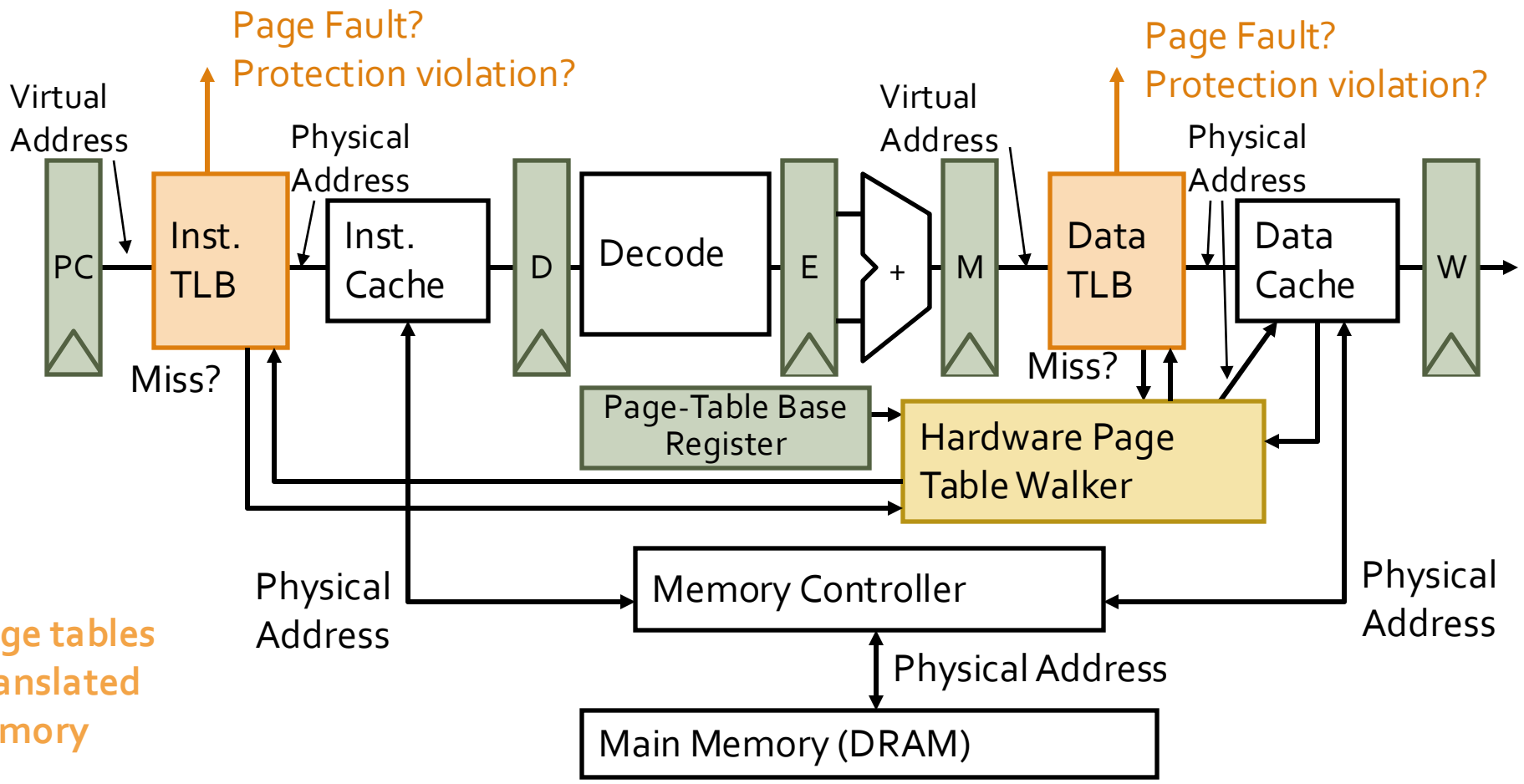
- NOTE: A given ISA can use either TLB miss strategy

Hierarchical Page Table Walk: SPARC v8



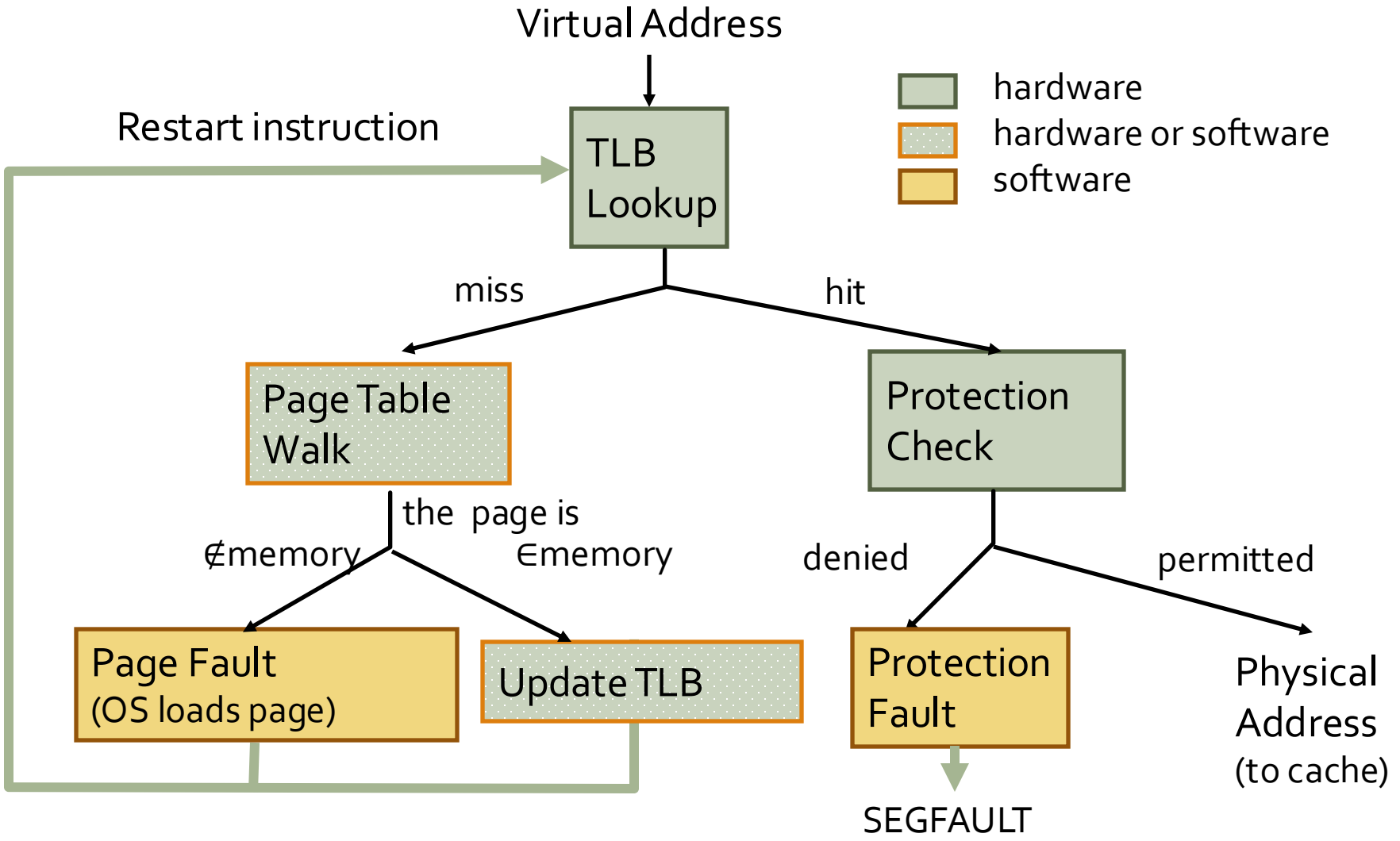
MMU does this table walk in hardware on a TLB miss

Page-Based Virtual-Memory Machine
(Hardware Page-Table Walk)



Assumes page tables held in untranslated physical memory

Address Translation:
putting it all together



Page Fault Handler

- When the referenced page is not in DRAM:
  - The missing page is located (or created)
  - It is brought in from disk, and page table is updated
    - Another user job may run on CPU while first job waits for the requested page to be read from disk, provided system allows architectural context to be saved and restored
  - If no free pages are left, a page is swapped out
    - Pseudo-LRU replacement policy, implemented in software
    - A set of free pages can be maintained by OS as background activity
- Since it takes a long time to transfer a page (msecs), page faults are handled completely in software by the OS

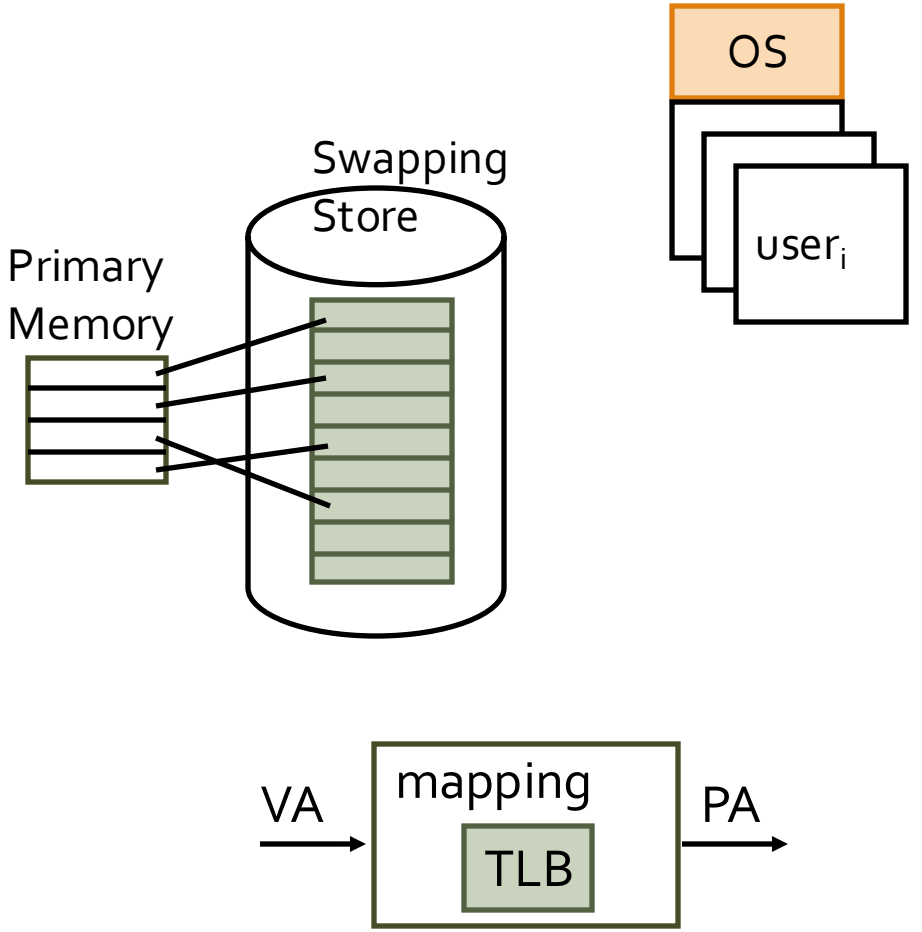
Modern Virtual Memory Systems

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

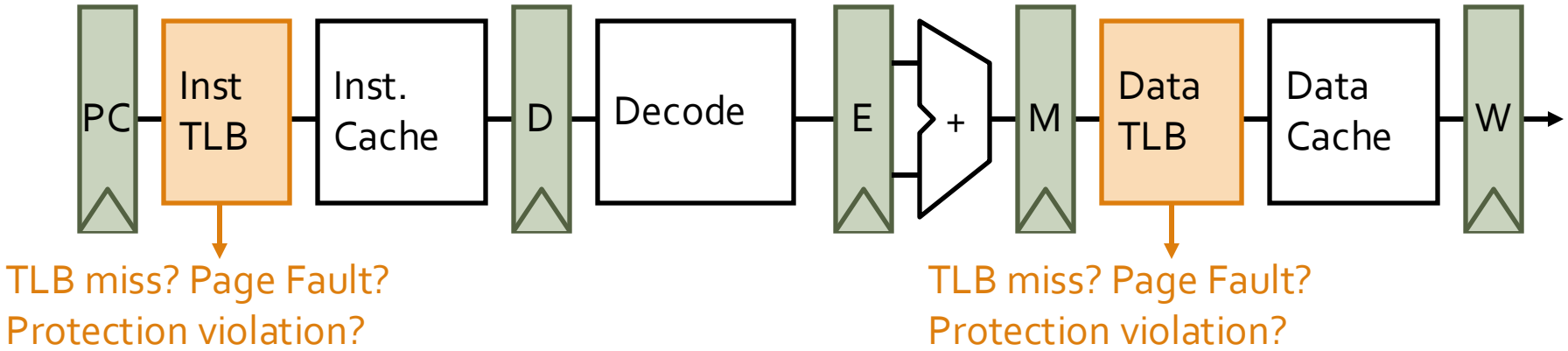
- Demand Paging
  - Provides the ability to run programs larger than the primary memory
  - Hides differences in machine configurations
- The price is address translation on each memory reference



Agenda

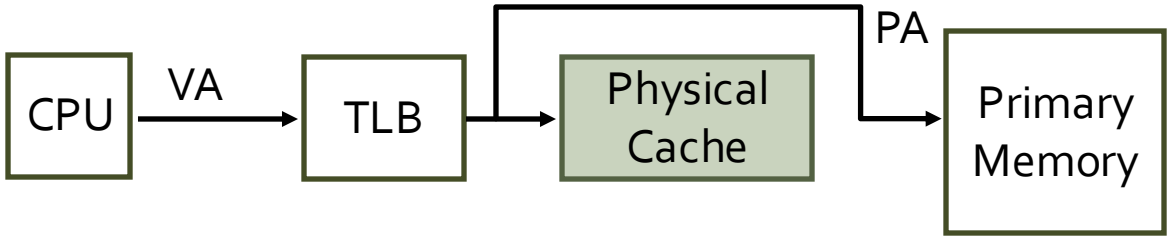
- Evolution of Virtual Memory
- Modern VM Implementation
- TLB & Cache Organization
- Modern Usage

Address Translation in CPU Pipeline

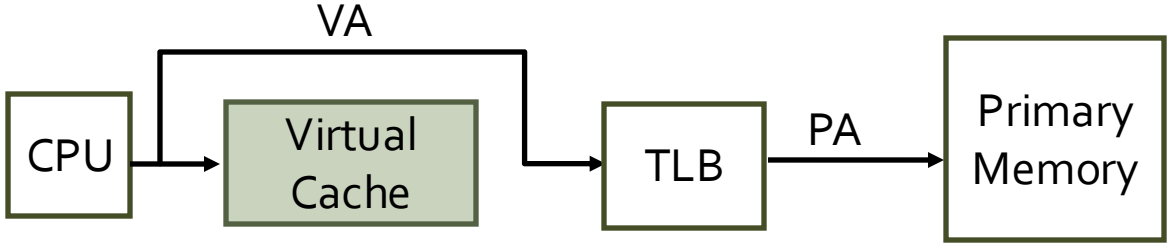


- Software handlers need **restartable** exception on TLB fault
- Handling a TLB miss needs a **hardware** or **software** mechanism to refill TLB
- Need to cope with additional latency of TLB:
  - slow down the clock?
  - pipeline the TLB and cache access?
  - virtual address caches
  - parallel TLB/cache access

Virtual-Address Caches



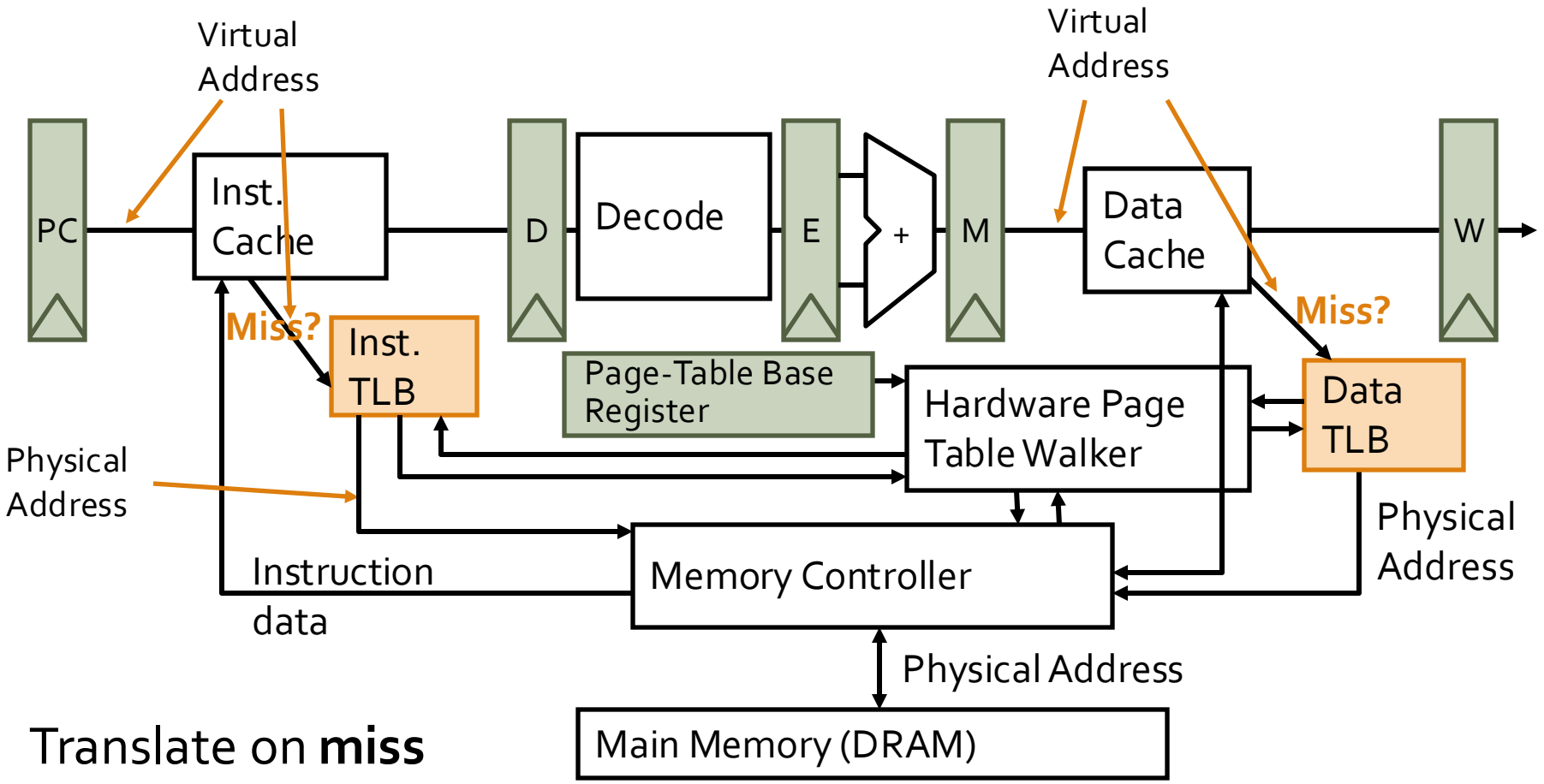
Alternative: place the cache before the TLB



(StrongARM)

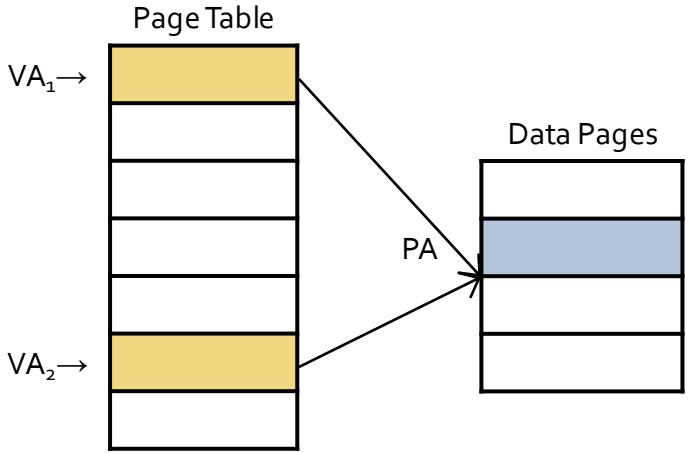
- one-step process in case of a hit (+)
- cache needs to be flushed on a context switch unless address space identifiers (ASIDs) included in tags (-)
- **aliasing problems** due to the sharing of pages (-)
- maintaining cache coherence (-) (see later in course)

Virtually Addressed Cache
(Virtual Index/Virtual Tag)



Translate on **miss**

Aliasing in Virtual-Address Caches



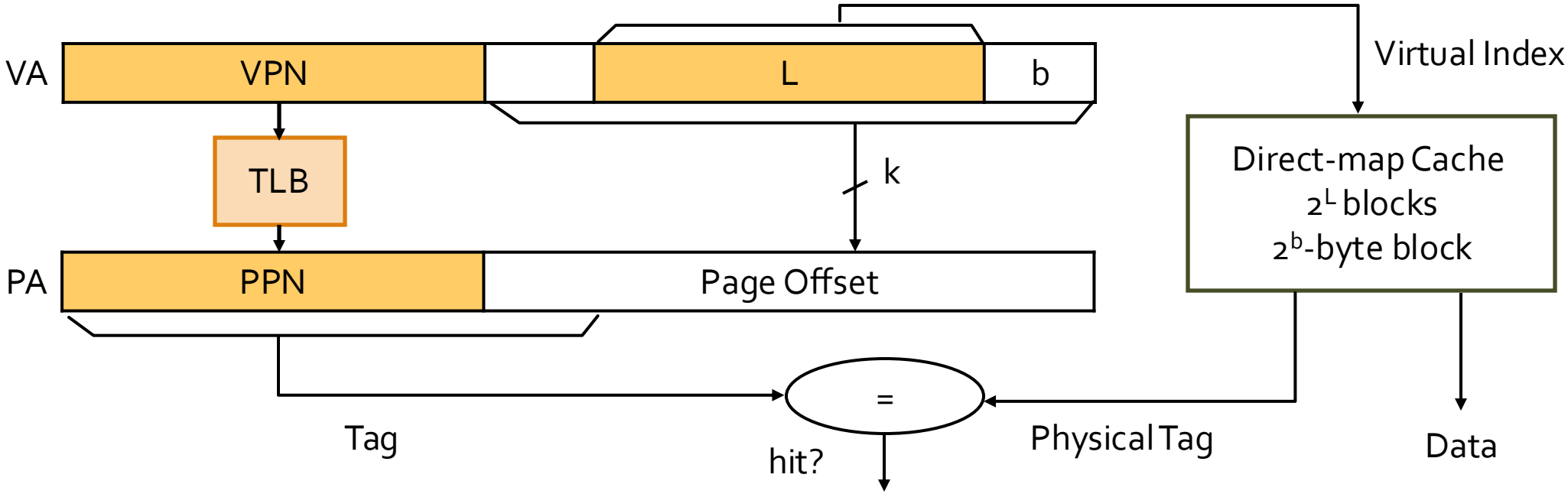
Two virtual pages share one physical page

Tag	Data
VA_1	1st Copy of Data at PA
VA_2	2nd Copy of Data at PA

Virtual cache can have two copies of same physical data. Writes to one copy not visible to reads of other!

- General Solution: Prevent aliases coexisting in cache
- Software (i.e., OS) solution for direct-mapped cache
  - VAs of shared pages must agree in cache index bits; this ensures all VAs accessing same PA will conflict in direct-mapped cache (early SPARCs)

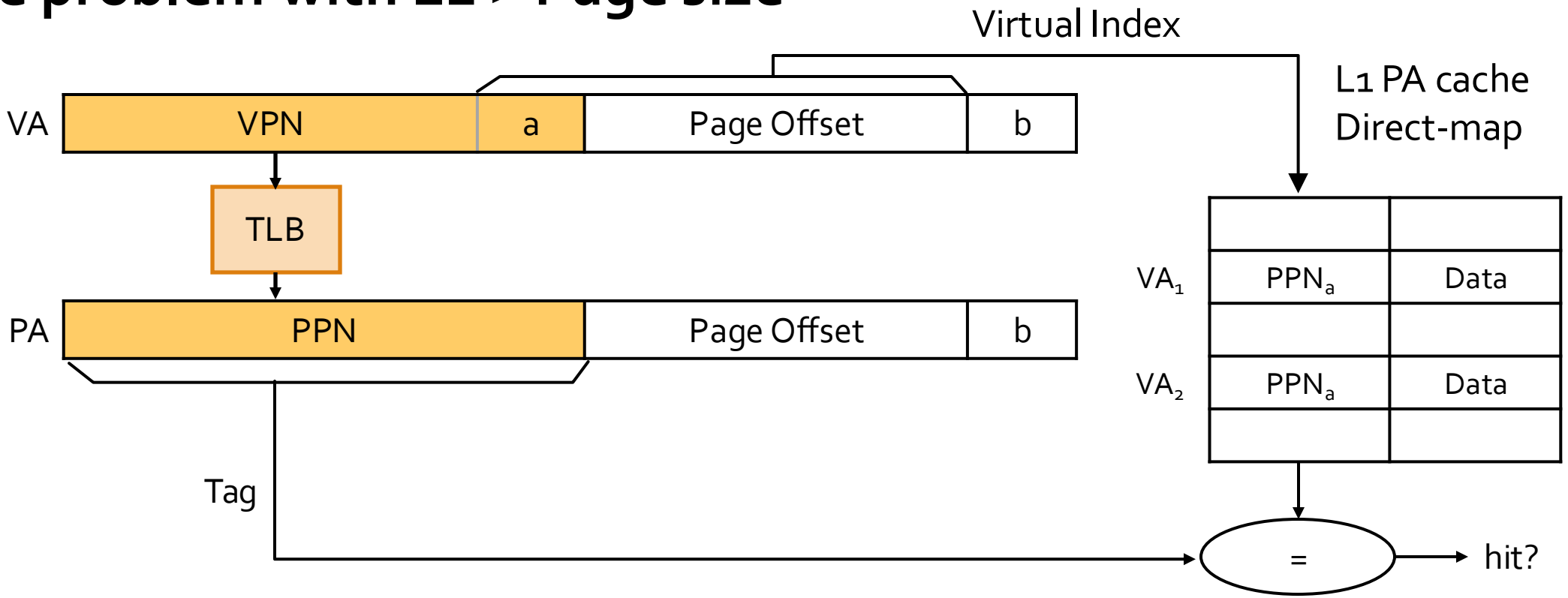
Concurrent Access to TLB & Cache
(Virtual Index/Physical Tag)



- Index L is available without consulting the TLB
  - cache and TLB accesses can begin simultaneously!
- Tag comparison is made after both accesses are completed
- Cases: L + b = k, L + b < k, ~~L + b > k~~ (cache can only be as big as page size!)

Concurrent Access to TLB & Large L1

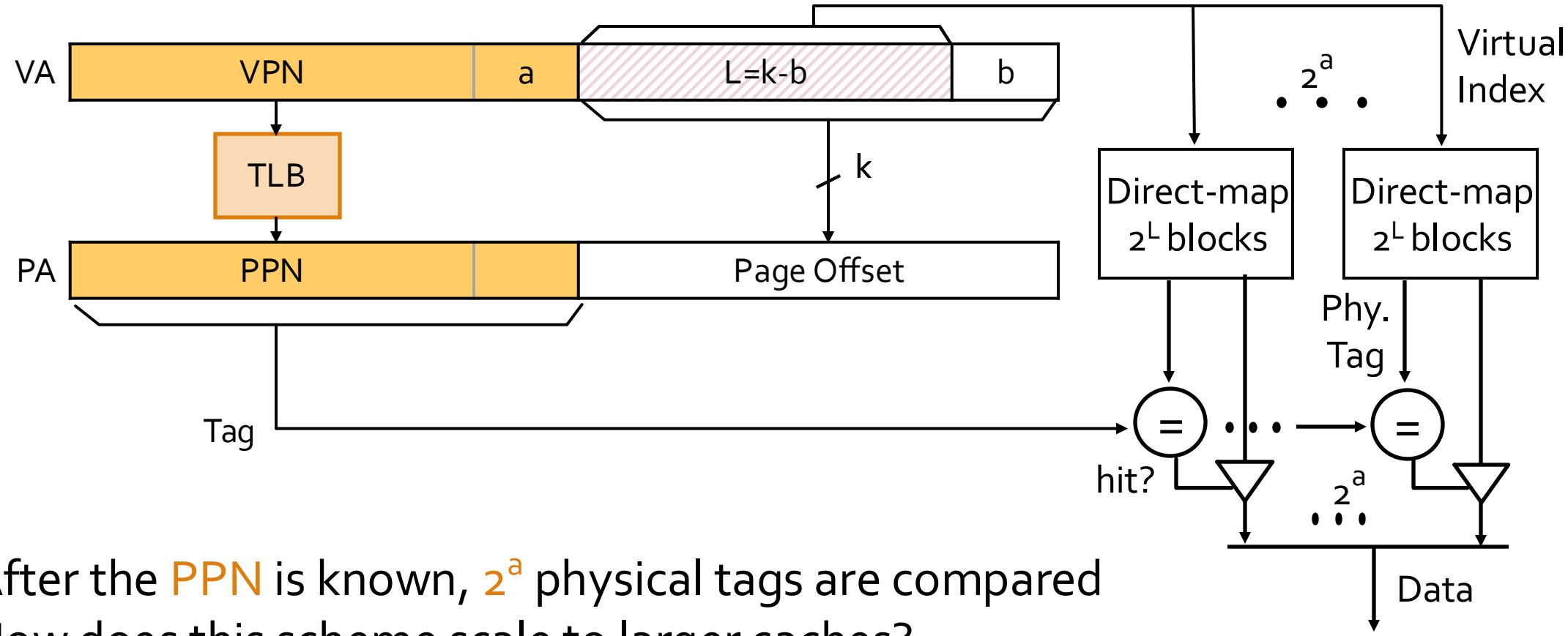
The problem with L1 > Page size



Can VA_1 and VA_2 both map to PA ?

Virtual-Index Physical-Tag Caches:

Use associativity to further increase cache capacity



- After the PPN is known, 2^a physical tags are compared
- How does this scheme scale to larger caches?
  - 4KB page size * 8-way (2^3) associative = 32KB cache

Cache-TLB Interactions

- Physically Indexed/Physically Tagged
- Virtually Indexed/Virtually Tagged
- Virtually Indexed/Physically Tagged
  - Concurrent cache access with TLB Translation
- Both Indexed/Physically Tagged
  - Small enough cache or highly associative cache will have fewer indexes than page size
  - Concurrent cache access with TLB Translation
- ~~Physically Indexed/Virtually Tagged~~

Agenda

- Evolution of Virtual Memory
- Modern VM Implementation
- TLB & Cache Organization
- Modern Usage

Virtual Memory Use Today - 1

- Servers/desktops/laptops/smartphones have full demand-paged virtual memory
  - Portability between machines with different memory sizes
  - Protection between multiple users or multiple tasks
  - Share small physical memory among active tasks
  - Simplifies implementation of some OS features
- Vector supercomputers have translation and protection but rarely complete demand-paging
  - (Older Crays: base&bound, Japanese & Cray X1/X2: pages)
  - Don't waste expensive CPU time thrashing to disk (make jobs fit in memory)
  - Mostly run in batch mode (run set of jobs that fits in memory)
  - Difficult to implement restartable vector instructions

Virtual Memory Use Today - 2

- Most embedded processors and DSPs provide physical addressing only
  - Can't afford area/speed/power budget for virtual memory support
  - Often there is no secondary storage to swap to!
  - Programs custom written for particular memory configuration in product
  - Difficult to implement restartable instructions for exposed architectures

Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)
  - Christopher Batten (Cornell)
  - David Wentzlaff (Princeton)
- MIT material derived from course 6.823
- UCB material derived from course CS252
- Cornell material derived from course ECE 4750
- Princeton material derived from course ECE 475