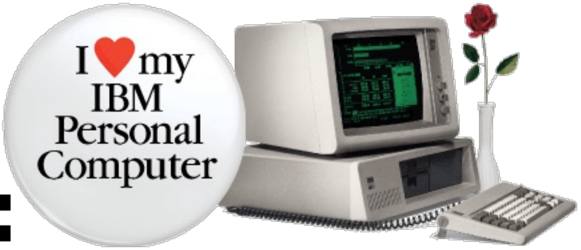
http://inst.eecs.berkeley.edu/~cs152

CS 152/252A Computer Architecture and Engineering

Lecture 9 – Address Translation

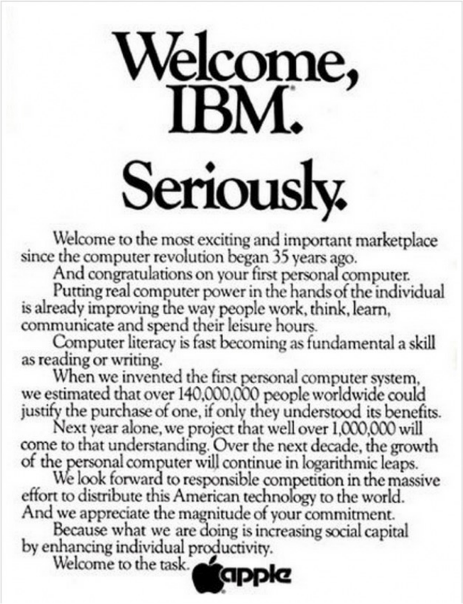
Sophia Shao

The PC and Wintel



♥ Wintel

It wasn't the first personal computer. Nor was it the most advanced. But shortly after the IBM ® Personal Computer arrived in 1981, it became the leading platform in the revolution that brought computing out of the glass house and into daily life.



Welcome, IBM. Seriously.

Welcome to the most exciting and important marketplace since the computer revolution began 35 years ago.

And congratulations on your first personal computer.

Putting real computer power in the hands of the individual is already improving the way people work, think, learn, communicate and spend their leisure hours.

Computer literacy is fast becoming as fundamental a skill as reading or writing.

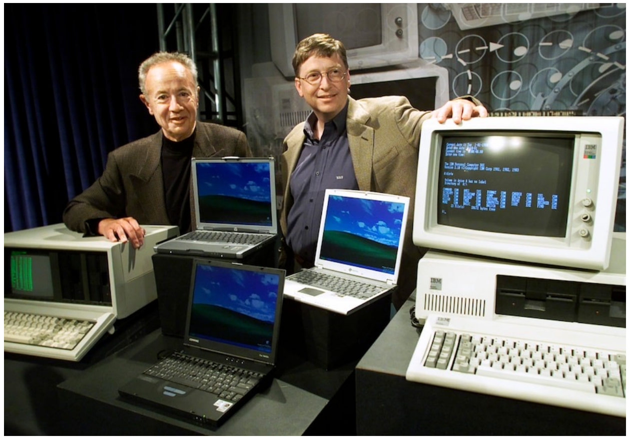
When we invented the first personal computer system, we estimated that over 140,000,000 people worldwide could justify the purchase of one, if only they understood its benefits.

Next year alone, we project that well over 1,000,000 will come to that understanding. Over the next decade, the growth of the personal computer will continue in logarithmic leaps.

We look forward to responsible competition in the massive effort to distribute this American technology to the world. And we appreciate the magnitude of your commitment.

Because what we are doing is increasing social capital by enhancing individual productivity.

Welcome to the task. apple

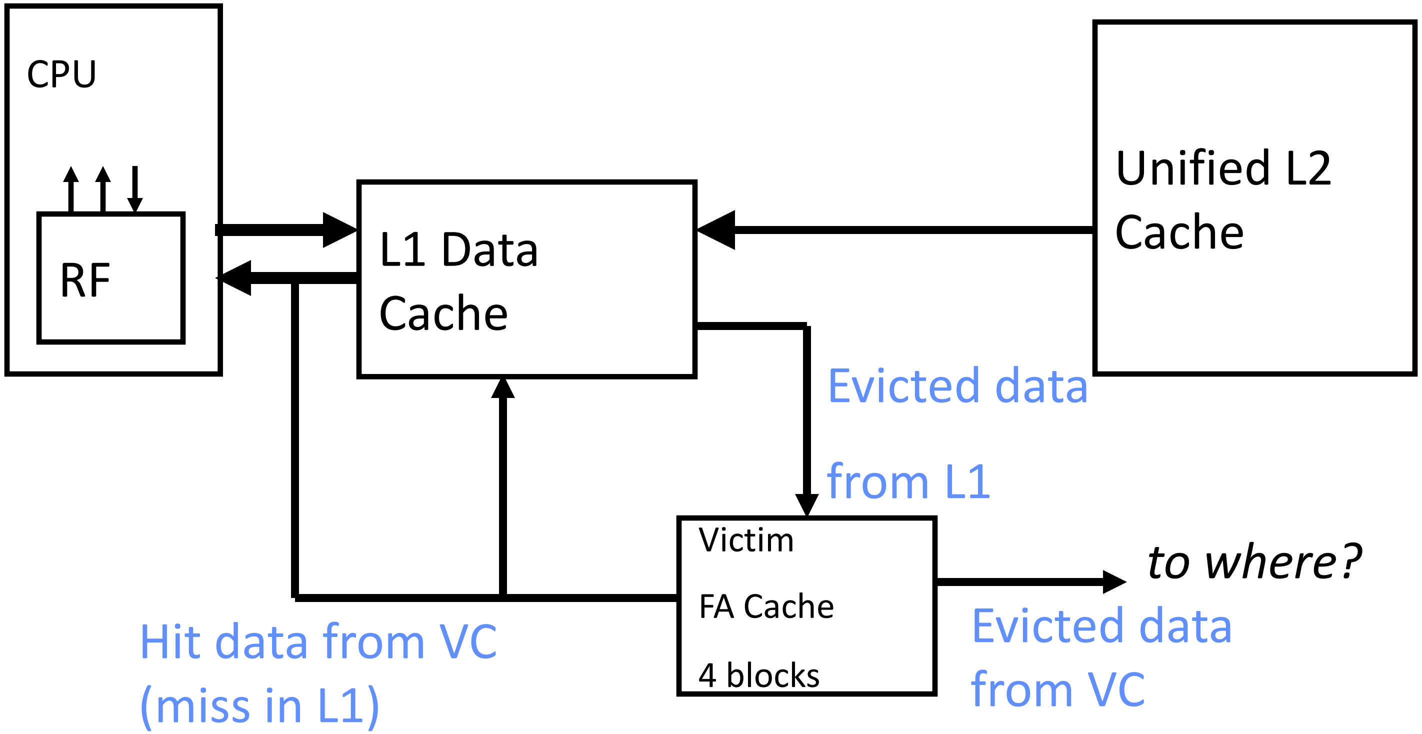


https://www.ibm.com/ibm/history/ibm100/us/en/icons/personalcomputer/

Last time in Lecture 8

- Prefetching, hardware or software
  - correctness, timeliness
  - instructions easier to prefetch than data
  - software difficult to use ideally

Victim Caches (HP 7200)



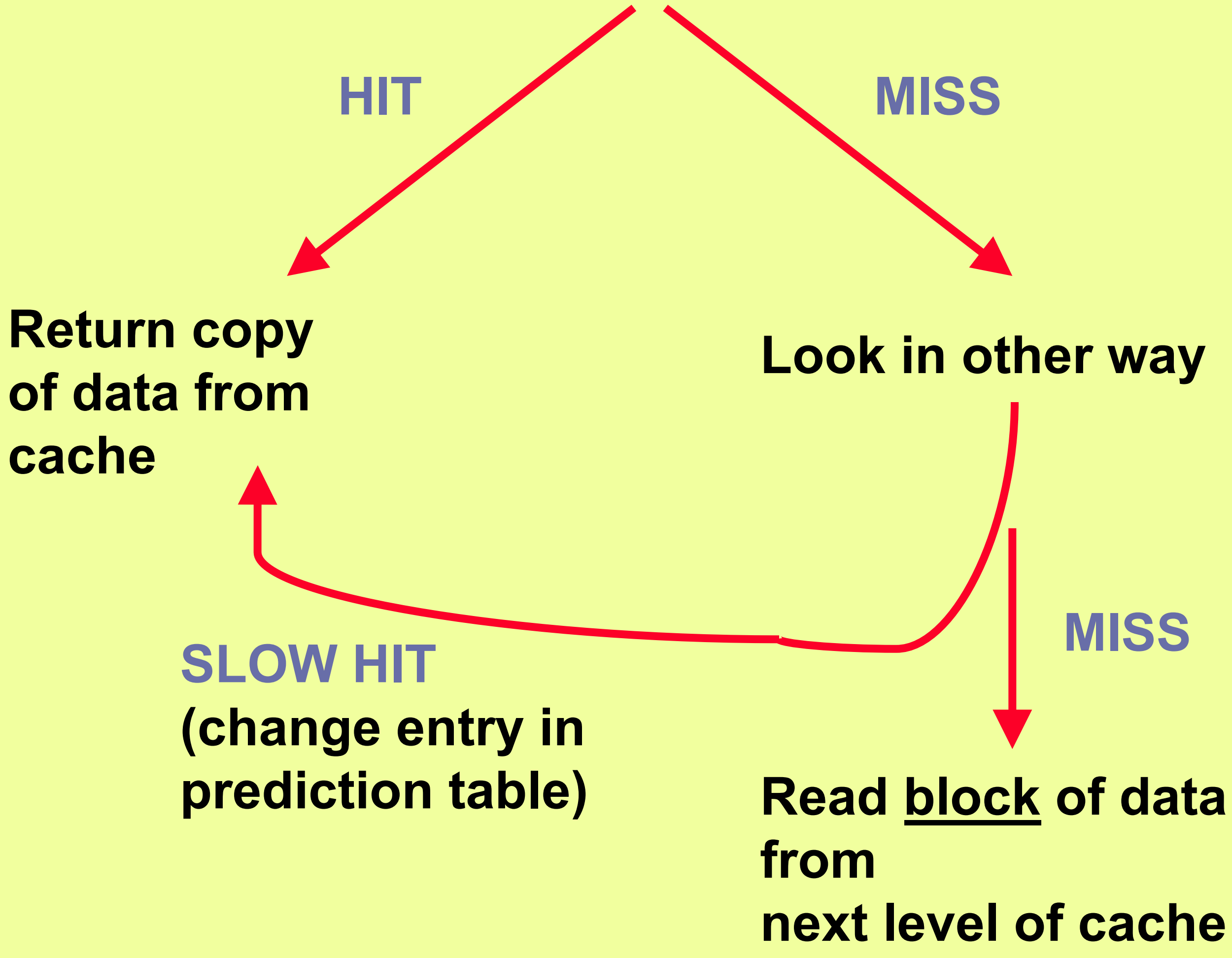
Victim cache is a small associative backup cache, added to a direct-mapped cache, which holds recently evicted lines

- First look up in direct-mapped cache
- If miss, look in victim cache
- If hit in victim cache, swap hit line with line now evicted from L1
- If miss in victim cache, L1 victim -> VC, VC victim->?

Fast hit time of direct mapped but with reduced conflict misses

Way-Predicting Caches (MIPS R10000 L2 cache)

- Use processor address to index into way-prediction table
- Look in predicted way at given index, then:



HIT

MISS

Return copy of data from cache

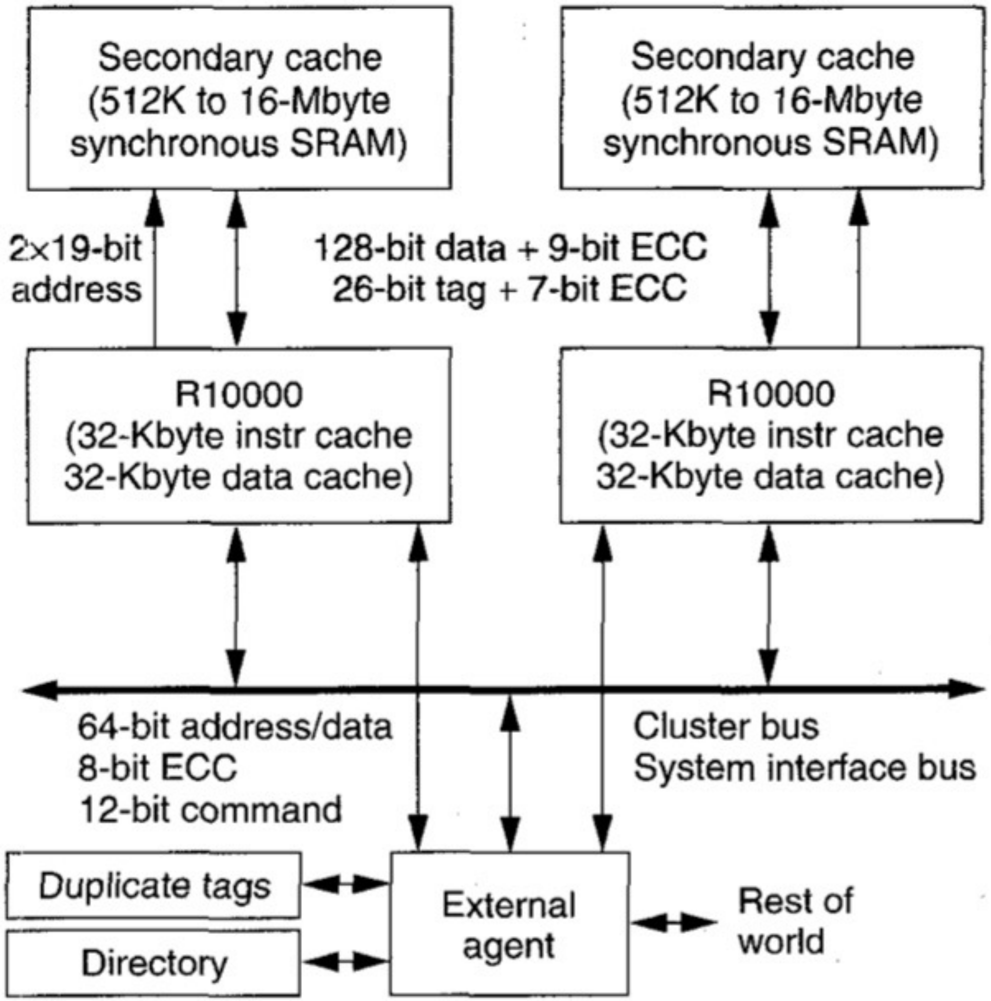
Look in other way

SLOW HIT (change entry in prediction table)

MISS

Read block of data from
next level of cache

MIPS R10000 Off-Chip L2 Cache (Yeager, IEEE Micro 1996)



Secondary cache (512K to 16-Mbyte synchronous SRAM)
Secondary cache (512K to 16-Mbyte synchronous SRAM)
2×19-bit address
128-bit data + 9-bit ECC
26-bit tag + 7-bit ECC
R10000 (32-Kbyte instr cache 32-Kbyte data cache)
R10000 (32-Kbyte instr cache 32-Kbyte data cache)
64-bit address/data
8-bit ECC
12-bit command
Cluster bus
System interface bus
Duplicate tags
Directory
External agent
Rest of world

Figure 1. System configuration. The cluster bus directly connects as many as four chips.

R10000 L2 Cache Timing Diagram

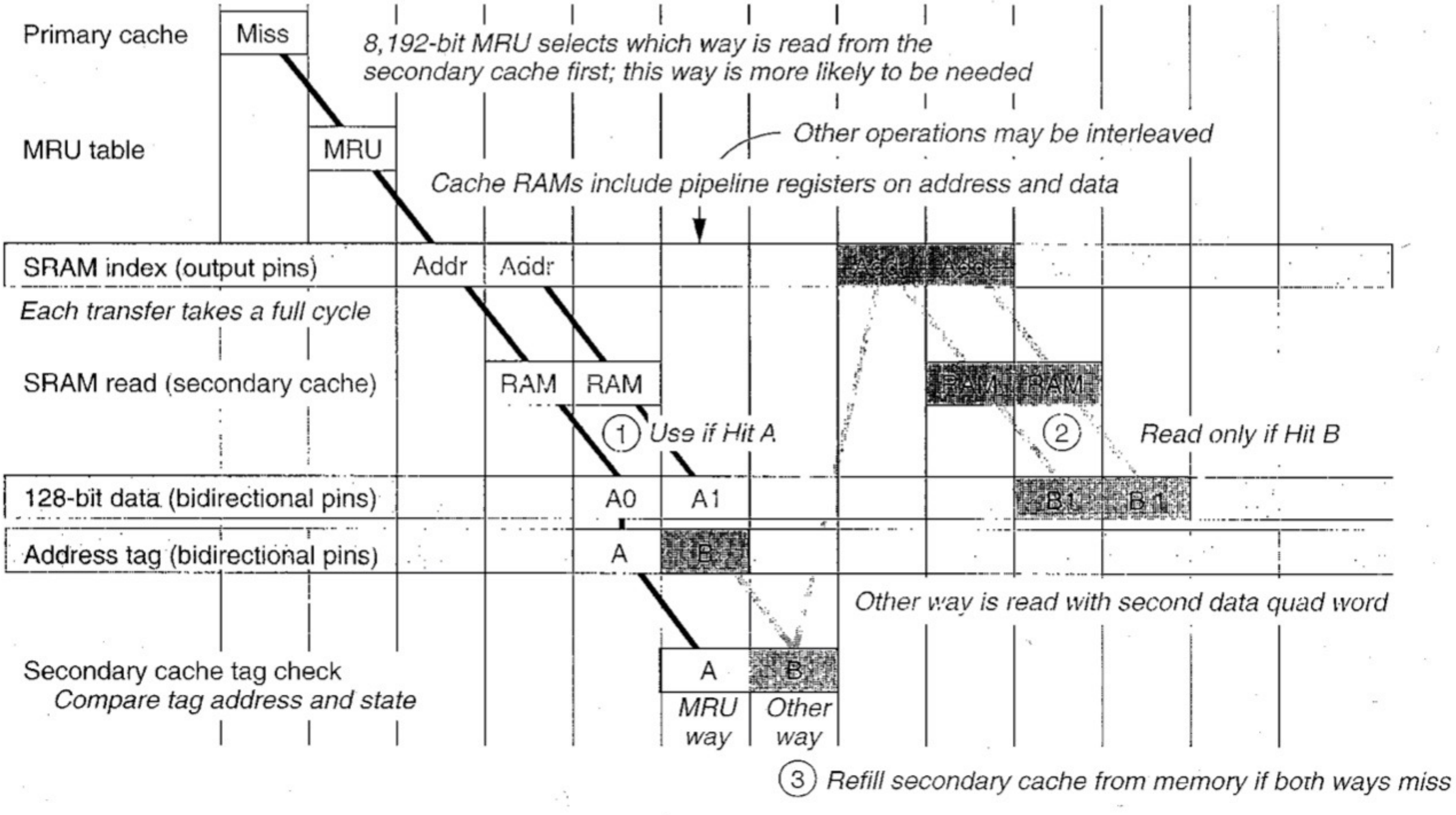

Primary cache
Miss
8,192-bit MRU selects which way is read from the secondary cache first; this way is more likely to be needed
MRU table
MRU
Other operations may be interleaved
Cache RAMs include pipeline registers on address and data
SRAM index (output pins)
Addr
Addr
Addr
Addr
Each transfer takes a full cycle
SRAM read (secondary cache)
RAM
RAM
RAM
RAM
1 Use if Hit A
2 Read only if Hit B
128-bit data (bidirectional pins)
A0
A1
B0
B1
Address tag (bidirectional pins)
A
B
Other way is read with second data quad word
Secondary cache tag check
Compare tag address and state
A
B
MRU way
Other way
3 Refill secondary cache from memory if both ways miss

Figure 12. Refill from the set-associative secondary cache. In this example, the secondary clock equals the processor's internal pipeline clock. It may be slower.

Way-Predicting Instruction Cache (Alpha 21264-like)

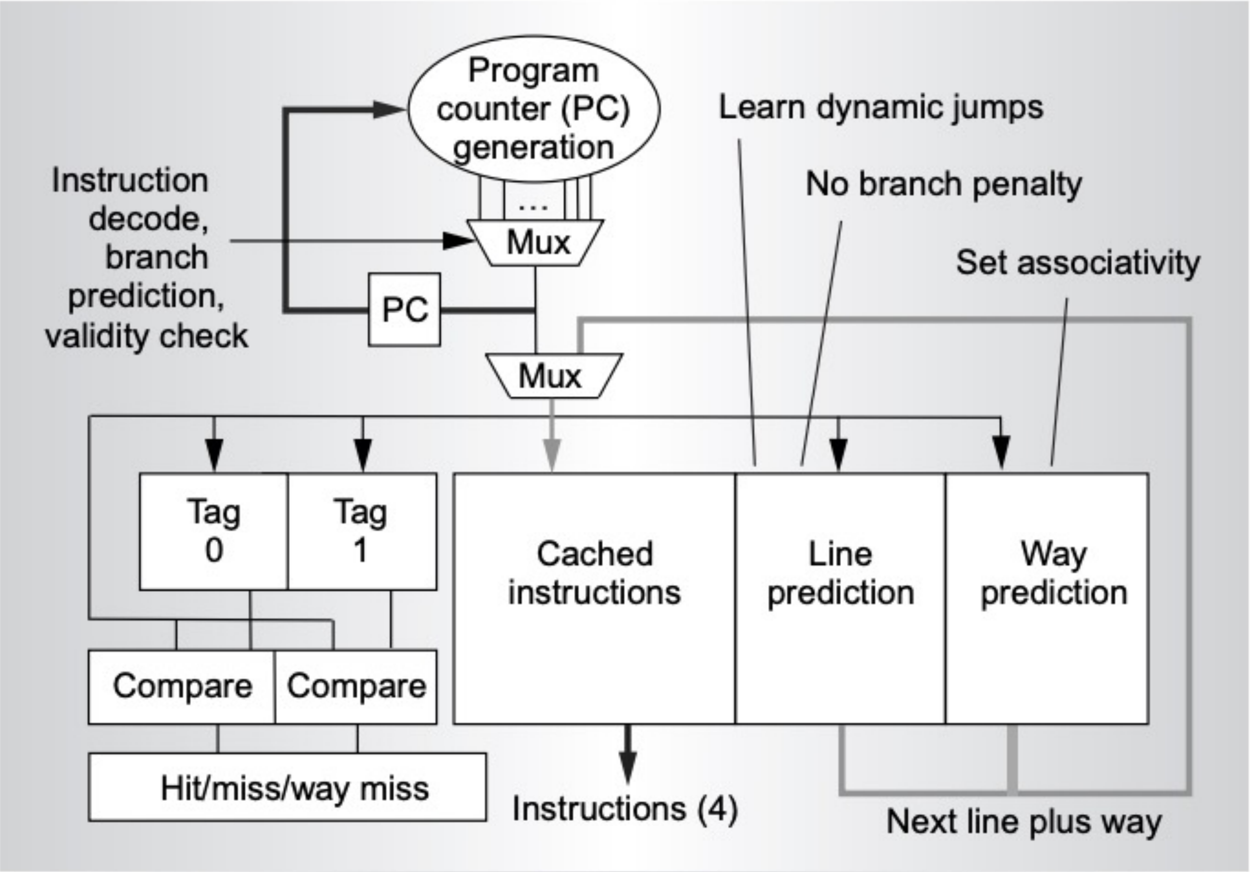
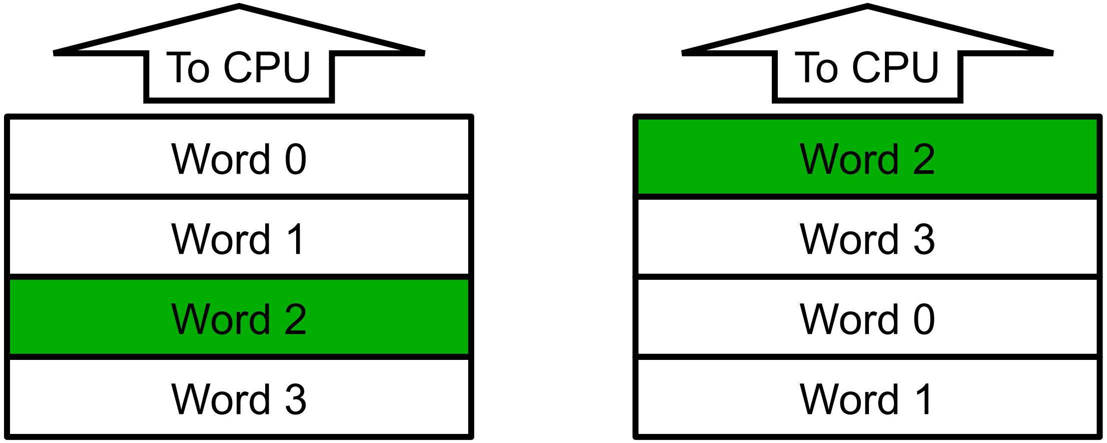


Figure 3. Alpha 21264 instruction fetch. The line and way prediction (wrap-around path on the right side) provides a fast instruction fetch path that avoids common fetch stalls when the predictions are correct.

Reduce Miss Penalty of Long Blocks: Early Restart and Critical Word First

- Don't wait for full block before restarting CPU
- *Early restart*—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
- *Critical Word First*—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block
  - Long blocks more popular today ⇒ Critical Word 1st Widely used

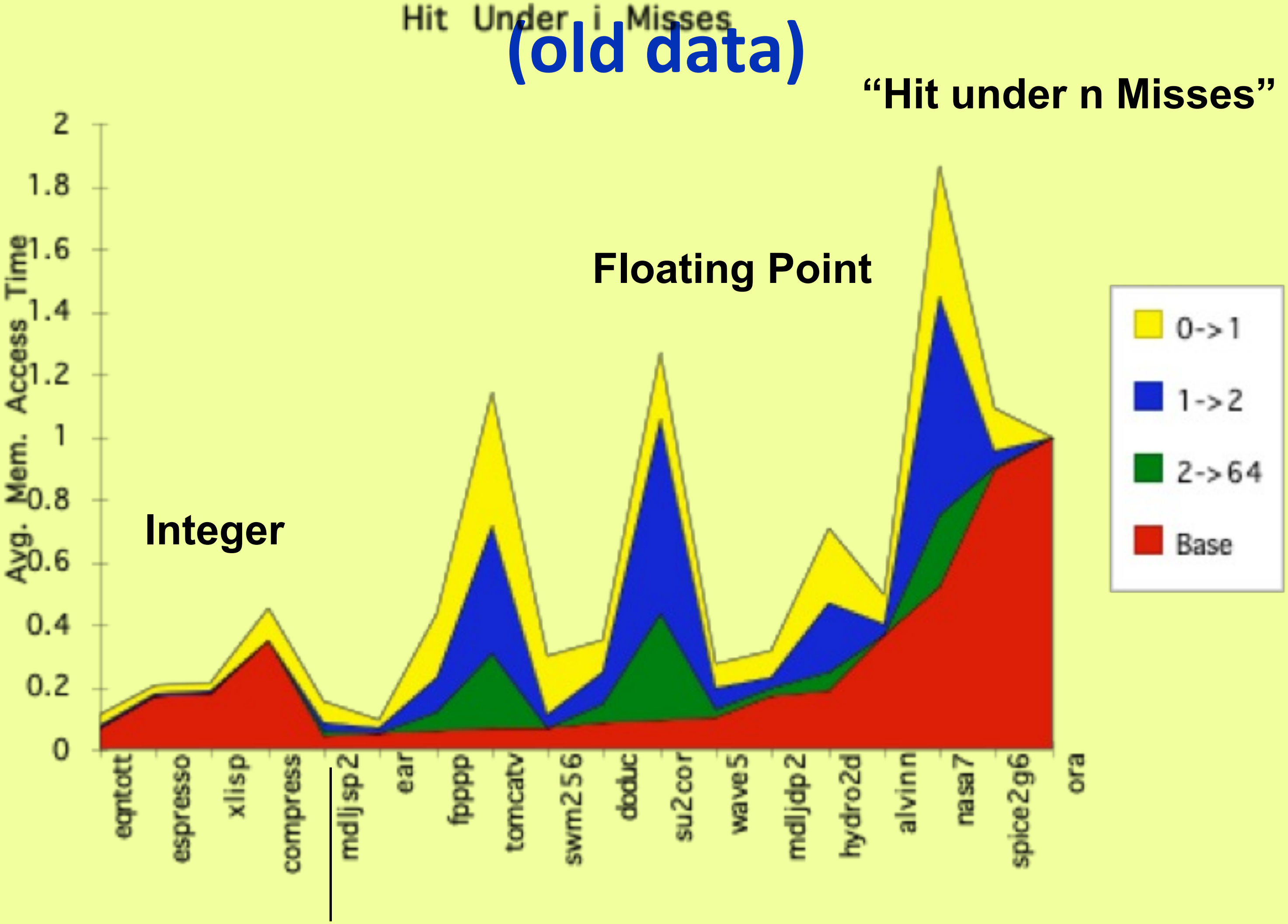


Rest of line filled in with wrap-around on cache line

Increasing Cache Bandwidth with Non-Blocking Caches

- *Non-blocking cache* or *lockup-free cache* allow data cache to continue to supply cache hits during a miss
  - requires Full/Empty bits on registers or out-of-order execution
- "*hit under miss*" reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- "*hit under multiple miss*" or "*miss under miss*" may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses, and can get miss to line with outstanding miss (secondary miss)
  - Requires pipelined or banked memory system (otherwise cannot support multiple misses)
  - Pentium Pro allows 4 outstanding memory misses
  - Cray X1E vector supercomputer allows 2,048 outstanding memory misses

Value of Hit Under Miss for SPEC (old data)



“Hit under n Misses”

Floating Point

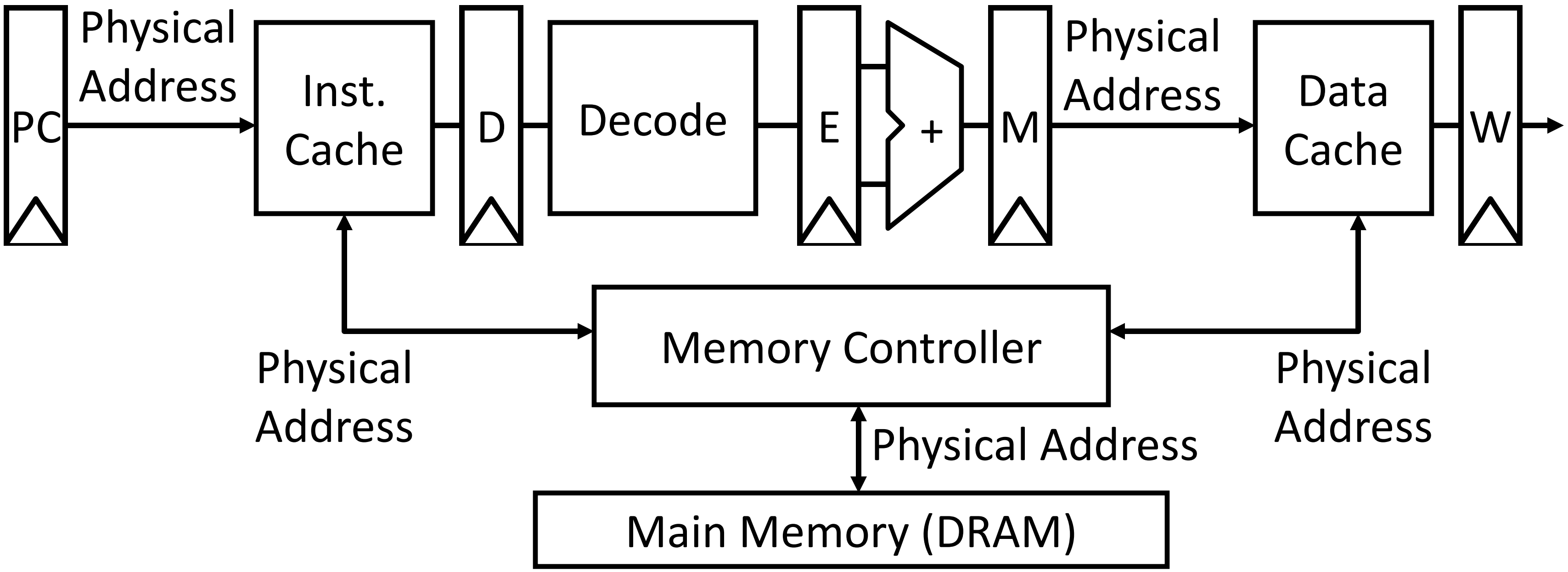
Integer

0->1
1->2
2->64
Base

- FP programs on average: AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26
- Int programs on average: AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss, SPEC 92

Address Translation

Bare Machine



In a bare machine, the only kind of address is a physical address, corresponding to address lines of actual hardware memory.

Managing Memory in Bare Machines

- Early machines only ran one program at a time, with this program having unrestricted access to all memory and all I/O devices
  - This simple memory management model was also used in turn by the first minicomputer and first microcomputer systems
- Subroutine libraries became popular, were written in location-independent form
  - Different programs use different combination of routines
- To run program on bare machines, use *linker* or *loader* program to relocate library modules to actual locations in physical memory

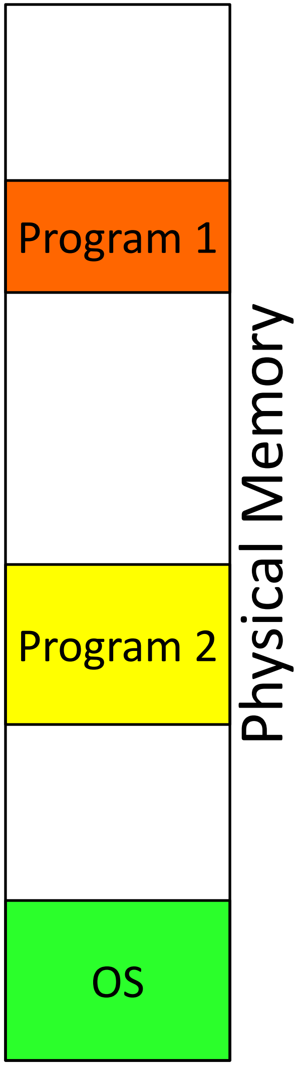
Dynamic Address Translation

- 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?

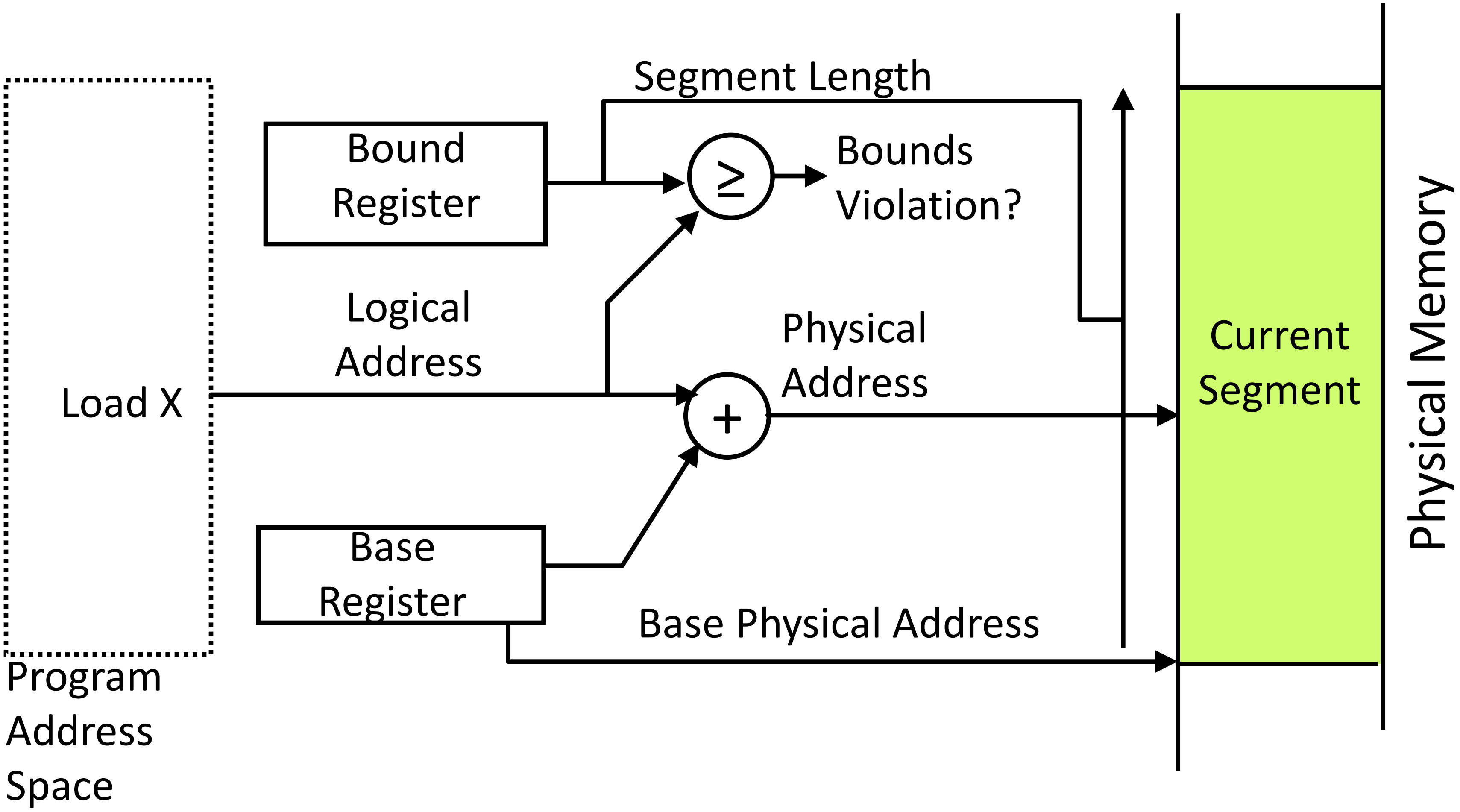
  → multiprogramming with DMA I/O devices, interrupts
- Location-independent programs
  - Programming and storage management ease

  → 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



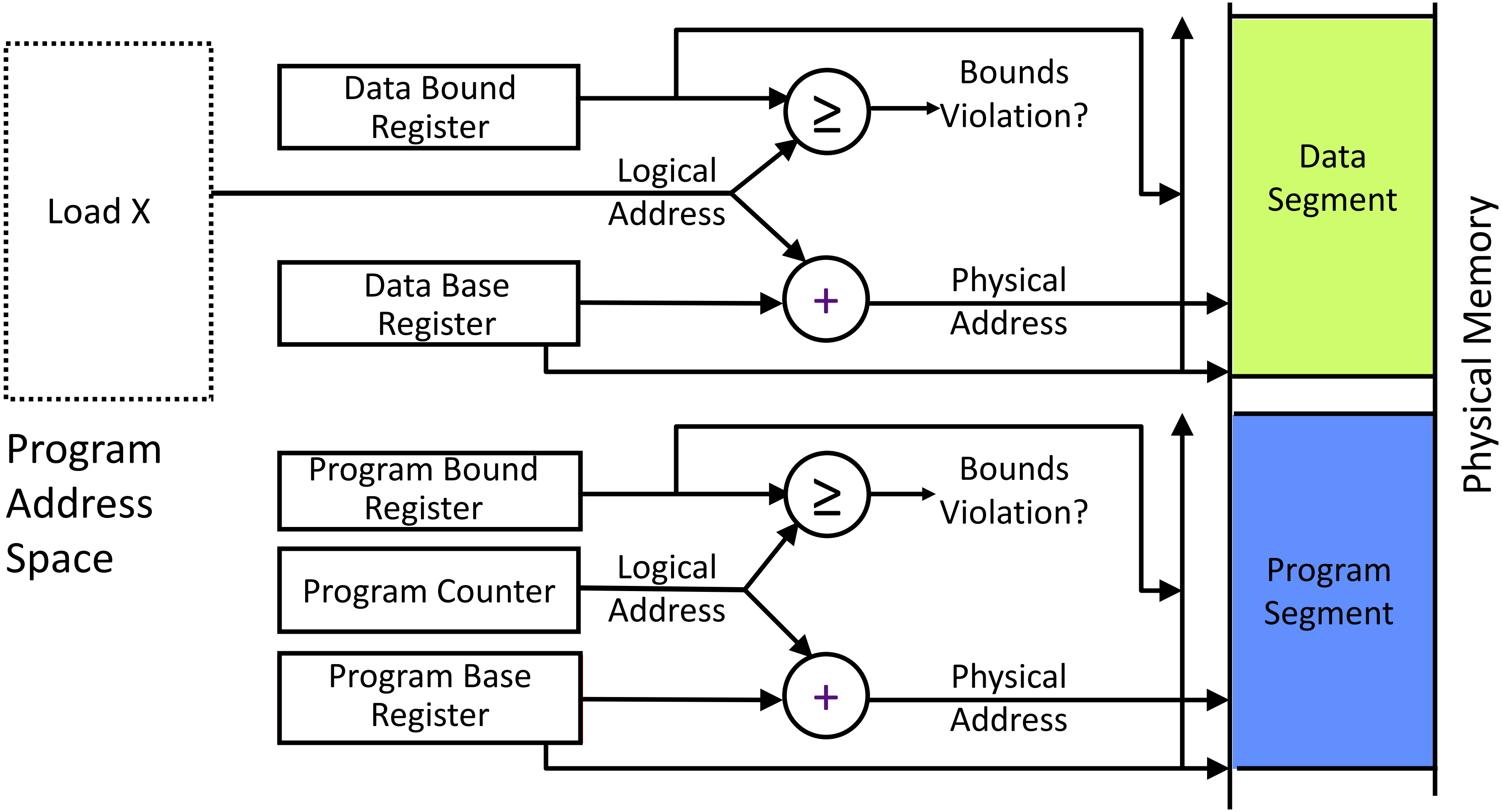
Simple Base and Bound Translation



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

Separate Areas for Program and Data

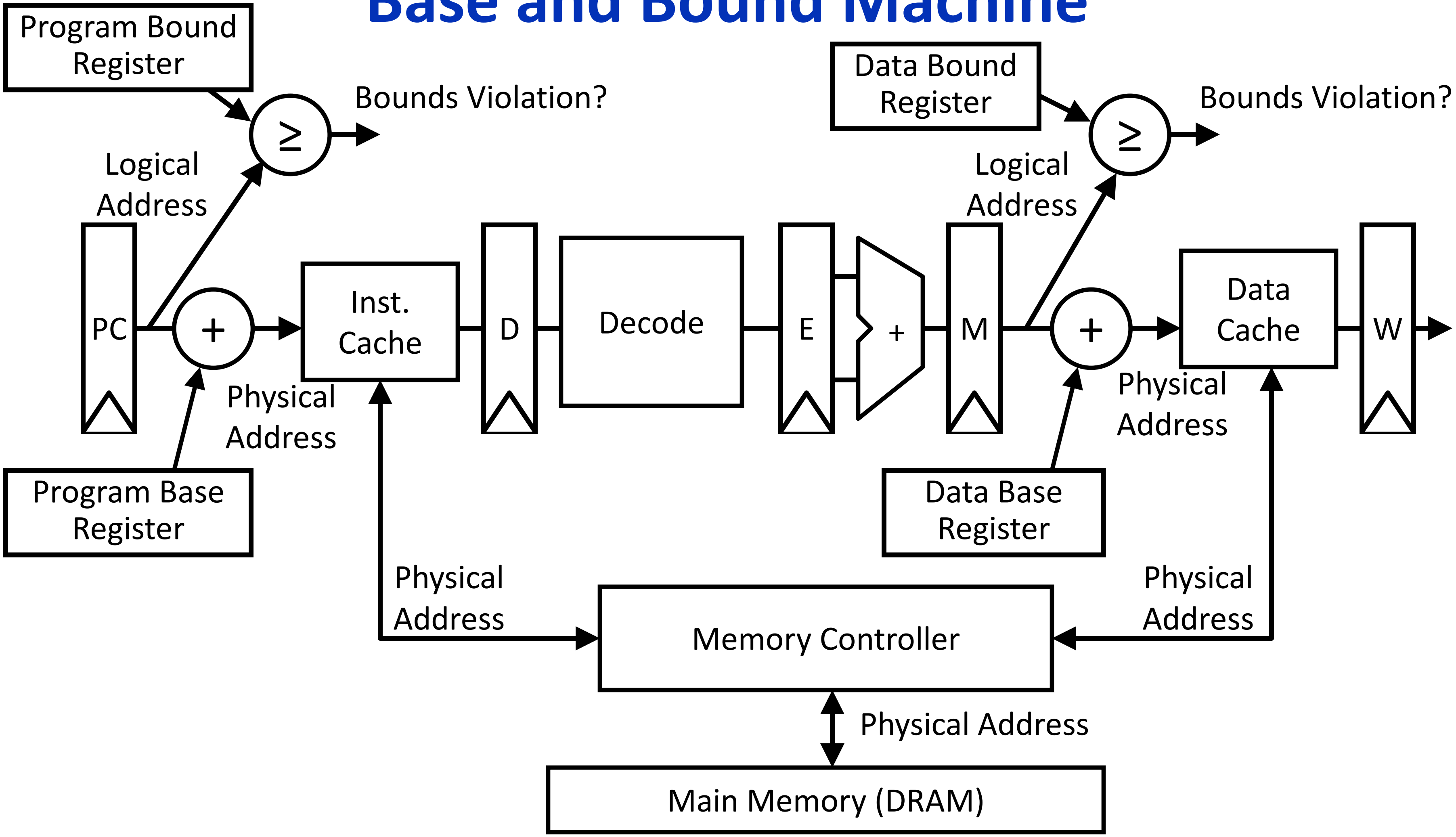
(Scheme used on all Cray vector supercomputers prior to X1, 2002)



What is an advantage of this separation?

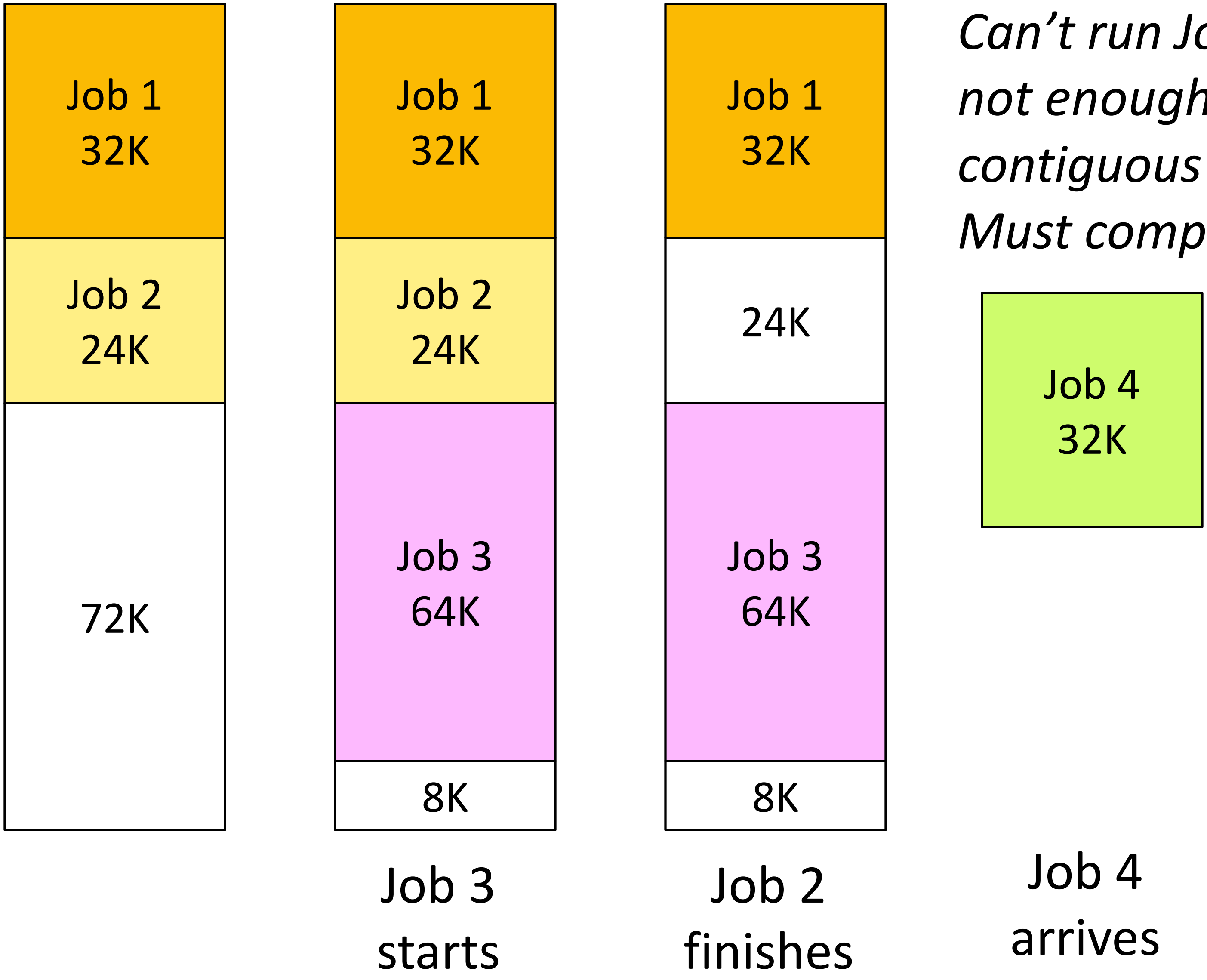
What about more base/bound pairs?

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)

External Fragmentation with Segments



Can't run Job 4, as not enough contiguous space. Must compact.

CS152 Administrivia

- HW2 out.
  - Due 2/21
- HW1 graded.
- Lab 2 out this week
  - Due 3/02
- Midterm 1 on Tuesday Feb 28
  - 7-9pm
  - 155 Dwinelle
  - Cover Lectures 1-10 (this week).

CS252 Administrivia

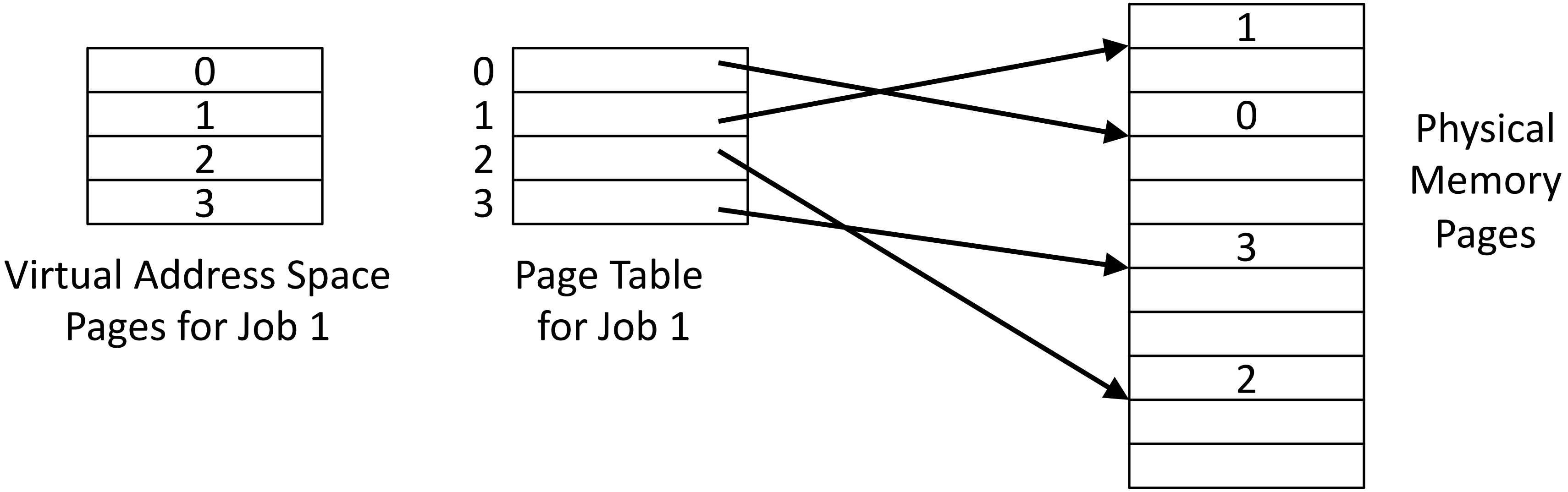
- Project Proposal due Wednesday 02/22.
- Proposal should be one page PDF including:
  - Title
  - Team member names
  - What are you trying to do?
  - How is it done today?
  - What is your idea for improvement and why do you think you'll be successful
  - What infrastructure are you going to use for your project?
  - Project timeline with milestones
- Submit through Gradescope.
- Give ~5-minute presentations in class in discussion section time on Wednesday 03/01 and 03/08

Paged Memory Systems

- Program-generated (*virtual* or *logical*) address split into:

Page Number	Offset

- Page Table contains physical address of start of each fixed-sized page in virtual address space



0
1
2
3

Virtual Address Space Pages for Job 1

0
1
2
3

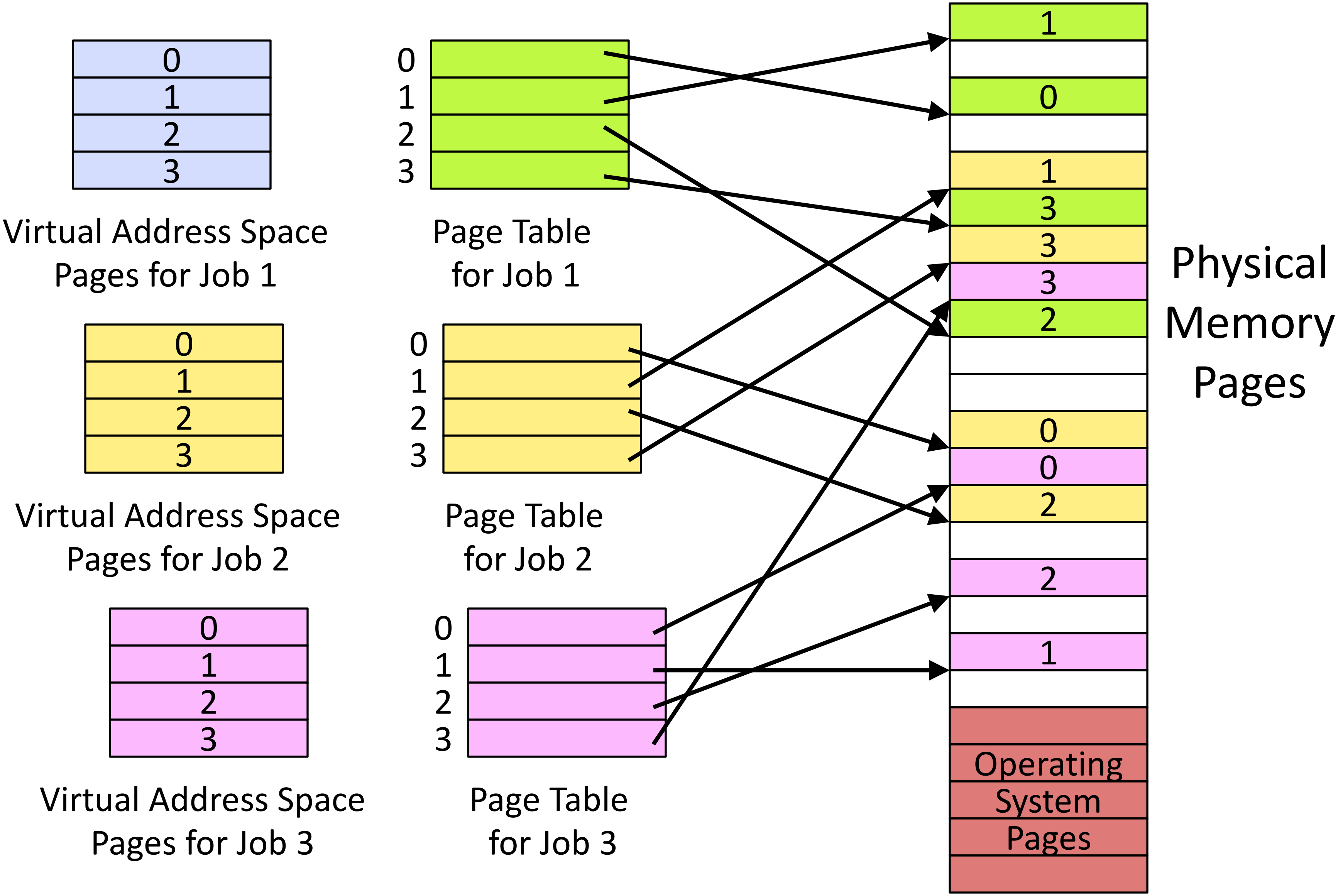
Page Table for Job 1

1
0
3
2

Physical Memory Pages

- Paging makes it possible to store a large contiguous virtual memory space using non-contiguous physical memory pages

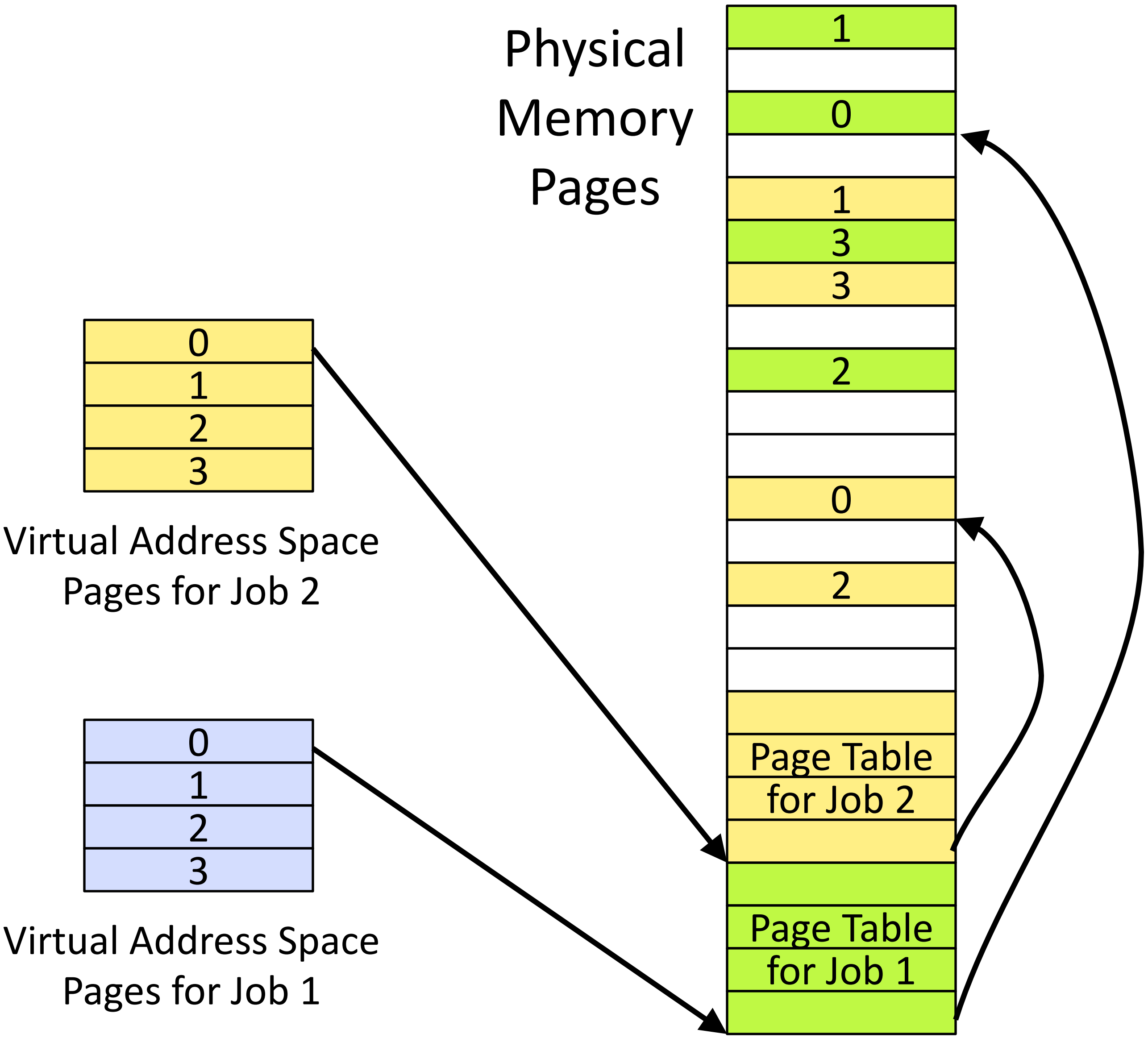
Private Address Space per User



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

Page Tables Live in Memory



Simple linear page tables are too large, so hierarchical page tables are commonly used (see later)

Common for modern OS to place page tables in kernel's virtual memory (page tables can be swapped to secondary storage)

Coping with Limited Primary Storage

- Paging reduces fragmentation, but still many problems would not fit into primary memory, have to copy data to and from secondary storage (drum, disk)
- Two early approaches:
  - **Manual overlays**, programmer explicitly copies code and data in and out of primary memory
    - Tedious coding, error-prone (jumping to non-resident code?)
  - **Software interpretive coding** (Brooker 1960). Dynamic interpreter detects variables that are swapped out to drum and brings them back in
    - Simple for programmer, but inefficient

Not just ancient black art, e.g., IBM Cell microprocessor using in Playstation-3 had explicitly managed local store!
Many new "deep learning" accelerators have similar structure.

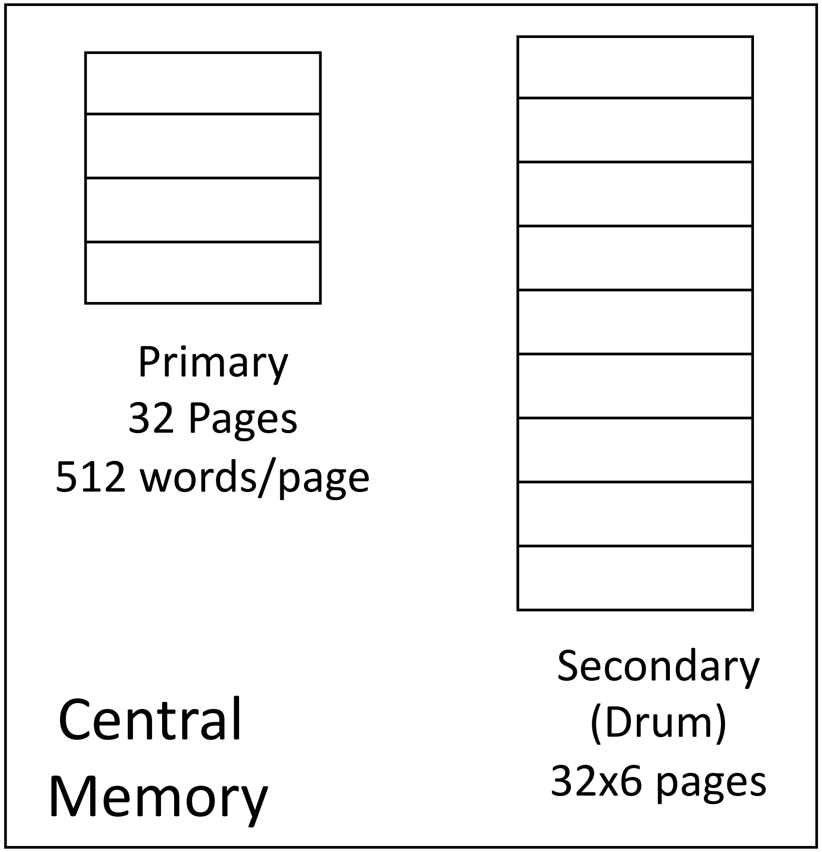
Demand Paging in Atlas (1962)

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

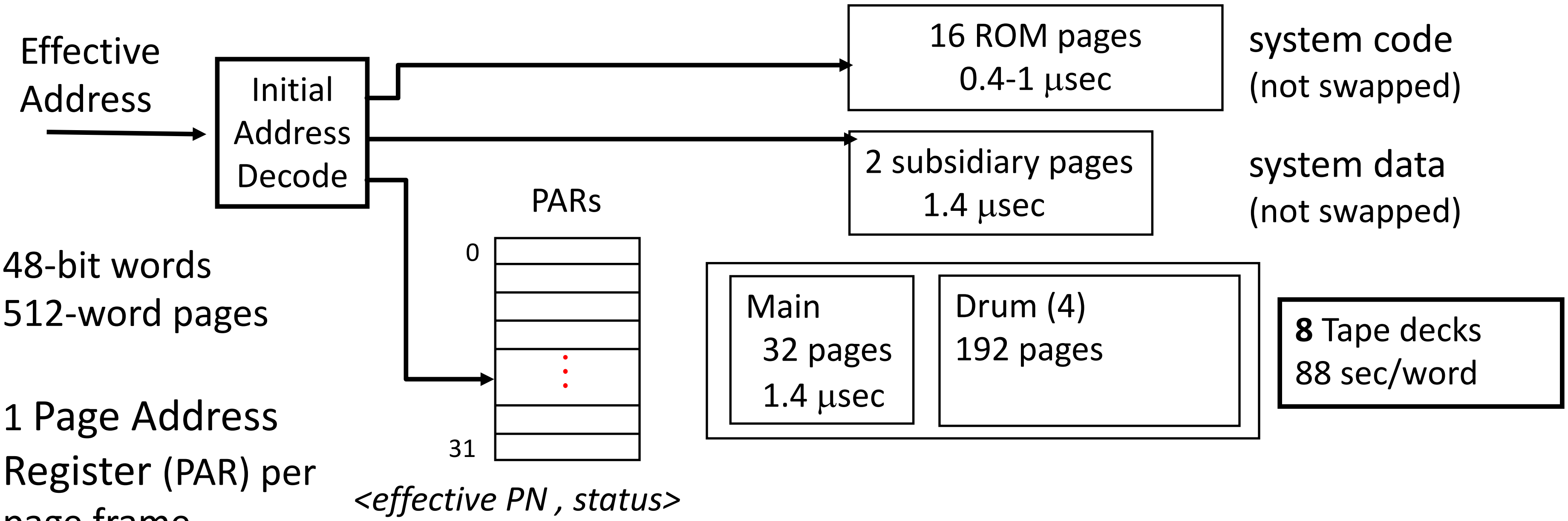
Tom Kilburn

Primary memory as a *cache* for secondary memory

User sees 32 x 6 x 512 words of storage



Hardware Organization of Atlas



Effective Address

Initial Address Decode

16 ROM pages
0.4-1 μsec

system code
(not swapped)

2 subsidiary pages
1.4 μsec

system data
(not swapped)

PARs

0

31

<effective PN , status>

Main
32 pages
1.4 μsec

Drum (4)
192 pages

8 Tape decks
88 sec/word

48-bit words
512-word pages

1 Page Address Register (PAR) per page frame

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:

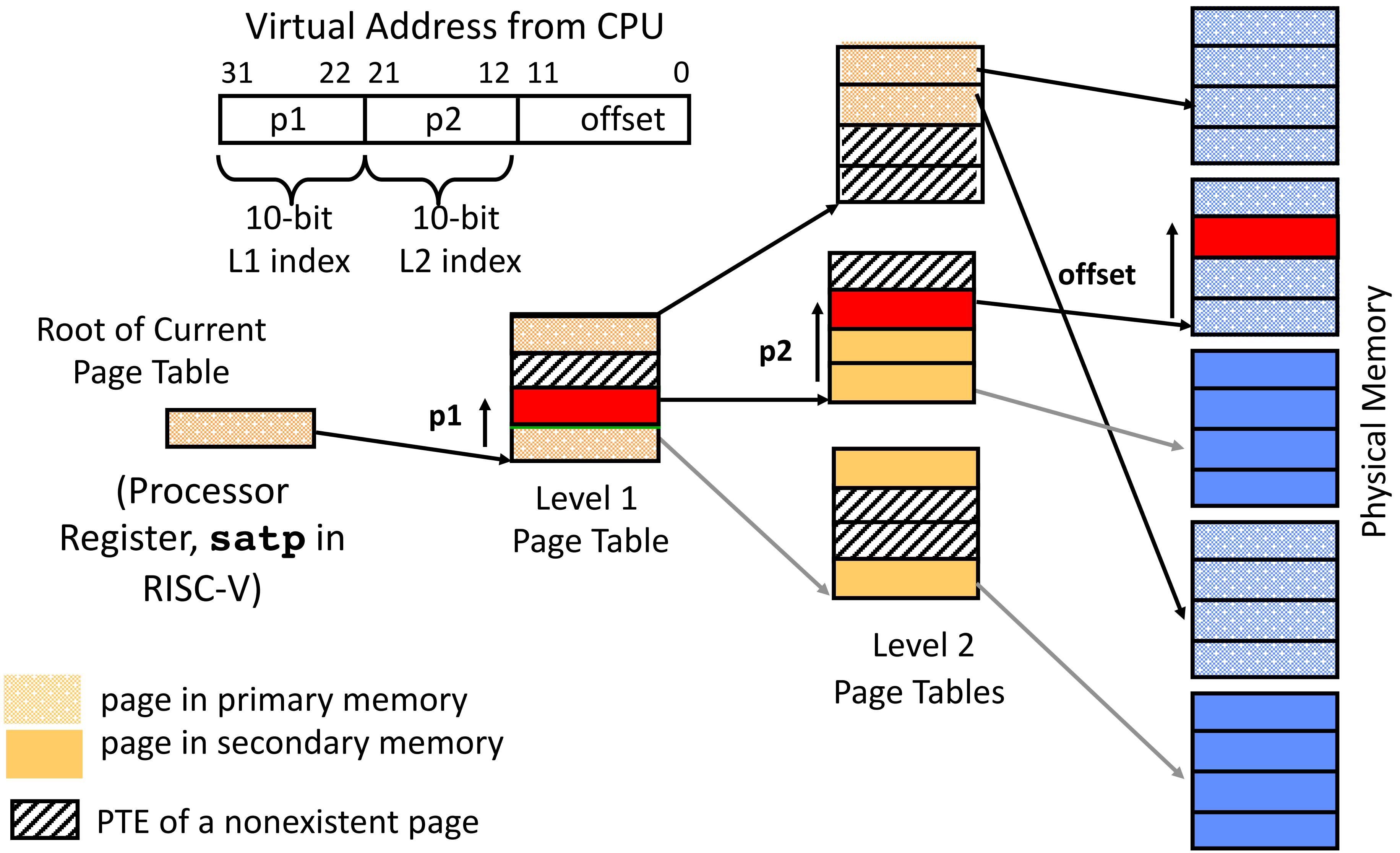
- Input transfer into a free page is initiated
- The Page Address Register (PAR) is updated
- If no free page is left, a page is selected to be replaced (based on usage)
- The replaced page is written on the drum
  - to minimize drum latency effect, the first empty page on the drum was selected
- The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

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



Data Pages

Virtual Address from CPU

31 22	21 12	11 0
p1	p2	offset

10-bit L1 index

10-bit L2 index

Root of Current Page Table

(Processor Register, **satp** in RISC-V)

Level 1 Page Table

Level 2 Page Tables

Physical Memory

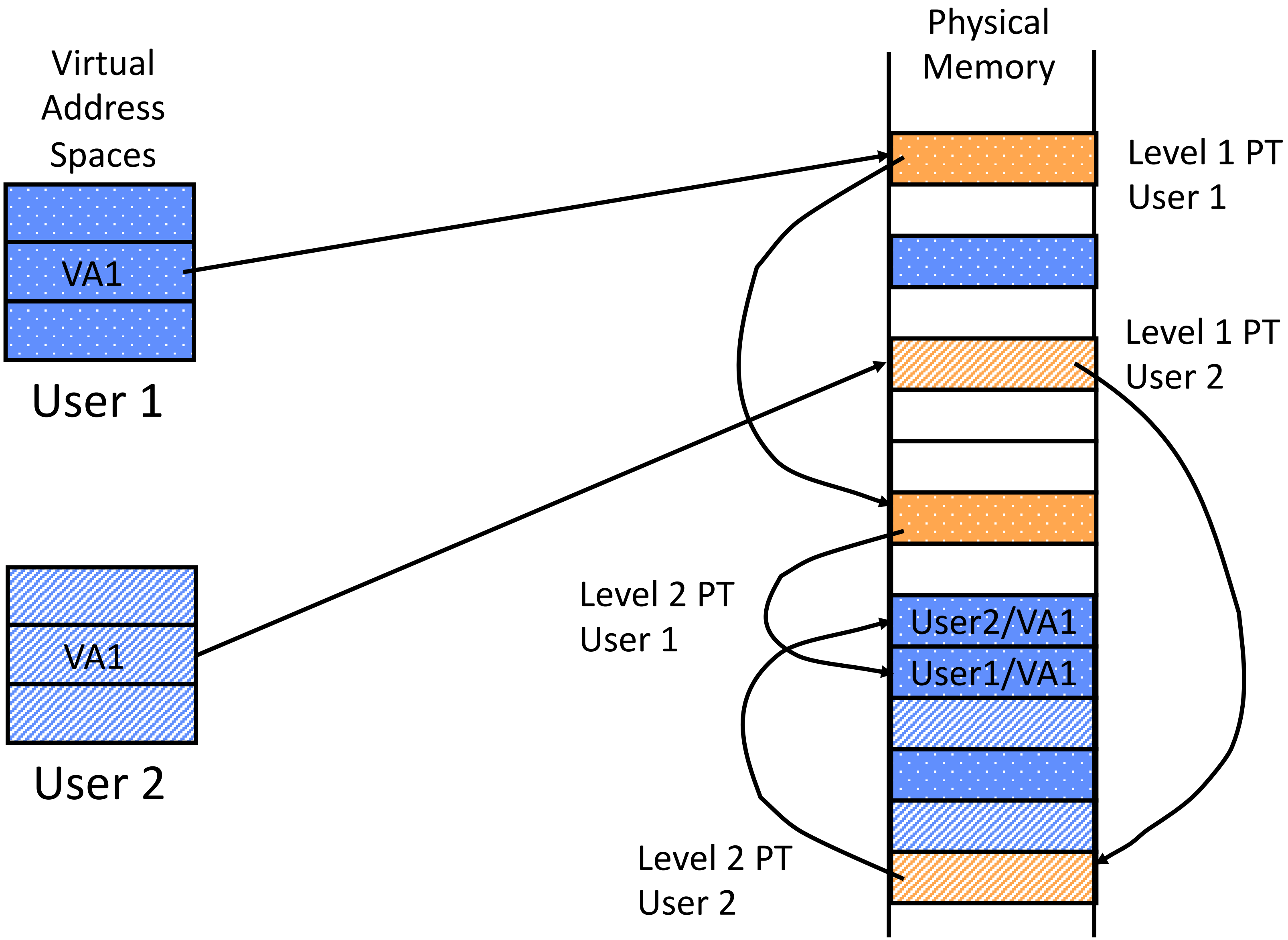
▨ page in primary memory

🟧 page in secondary memory

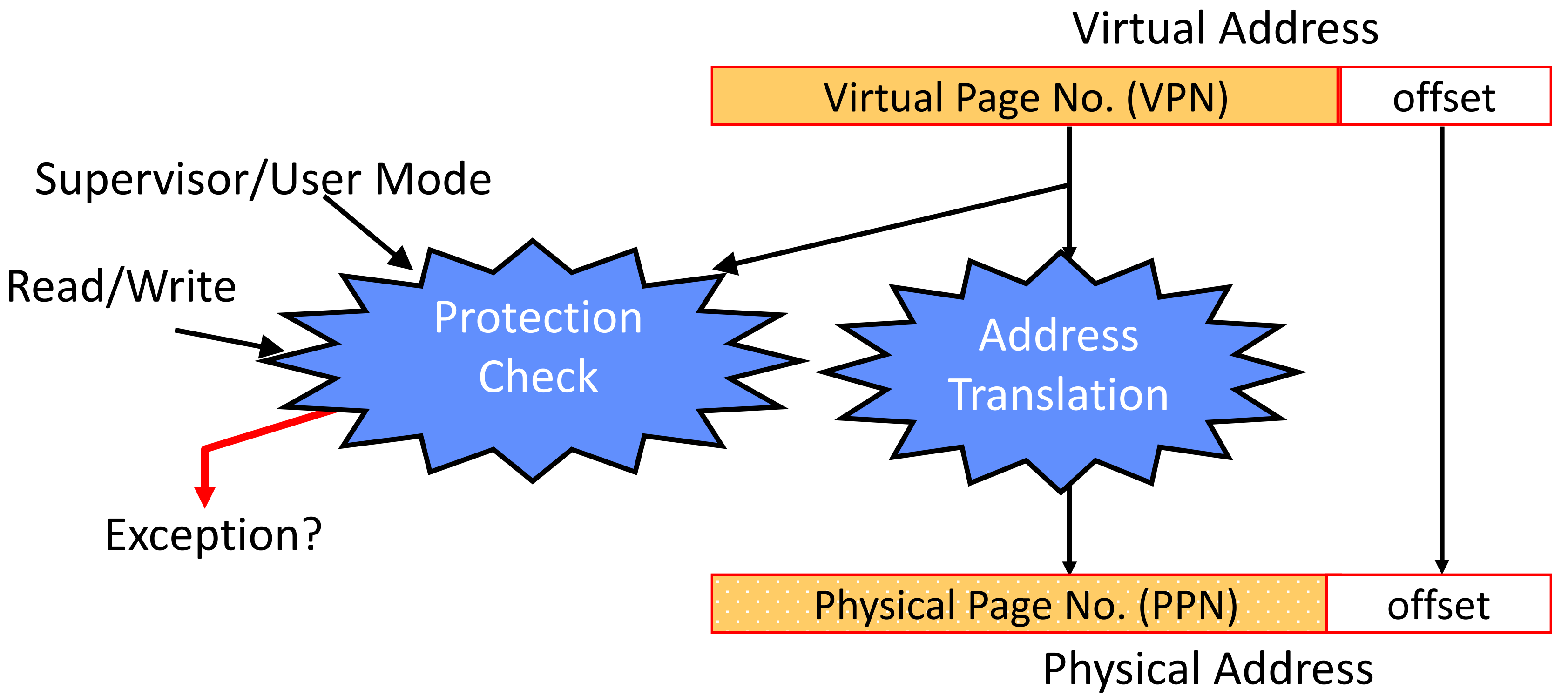
▨ PTE of a nonexistent page

RISC-V Sv32 Virtual Memory Scheme

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

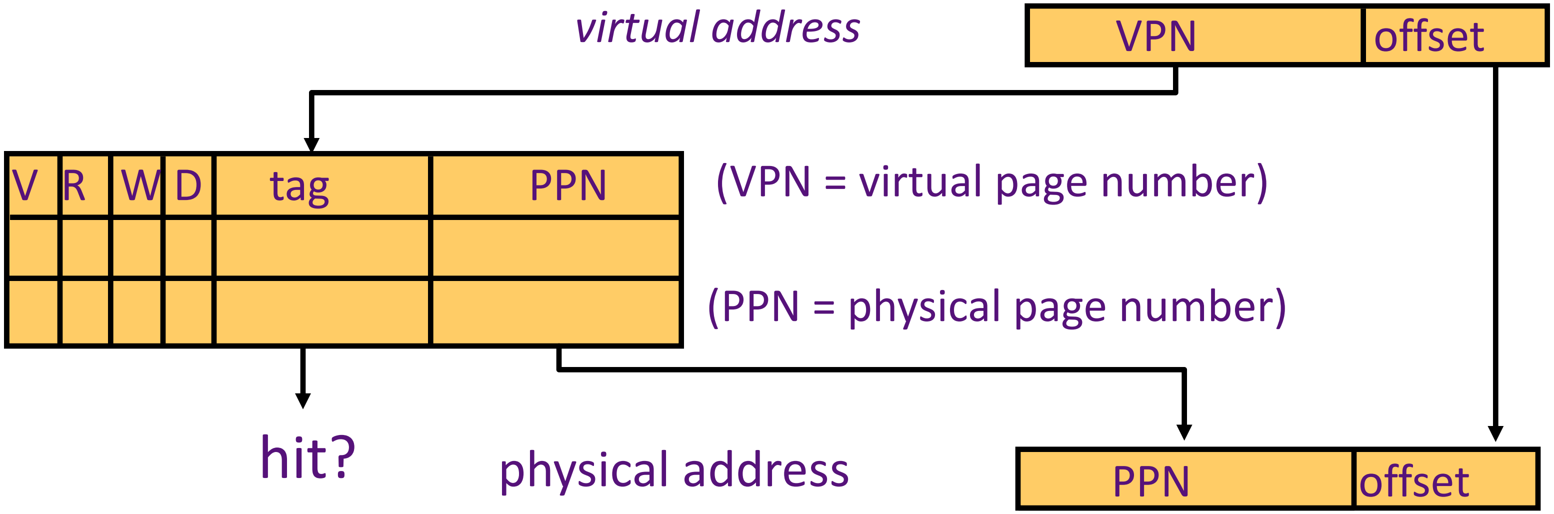
Translation-Lookaside Buffers (TLB)

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*



(VPN = virtual page number)

(PPN = physical page number)

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, 4KB pages, one page per entry
  - TLB Reach = ____?

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, 4KB pages, one page per entry
  - TLB Reach =

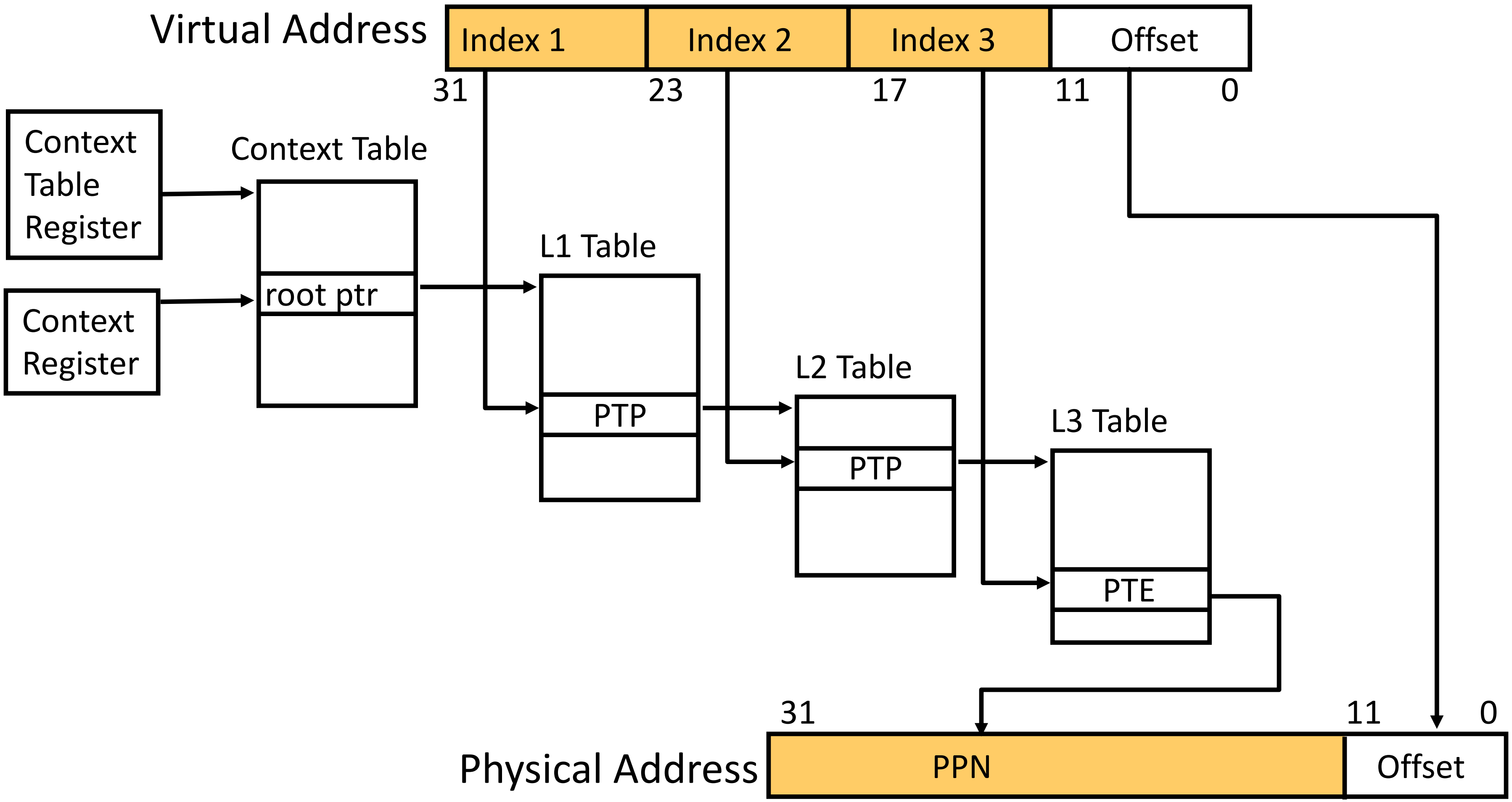
    *64 entries * 4 KB = 256 KB (if contiguous)*

    ______________________________________?

Handling a TLB Miss

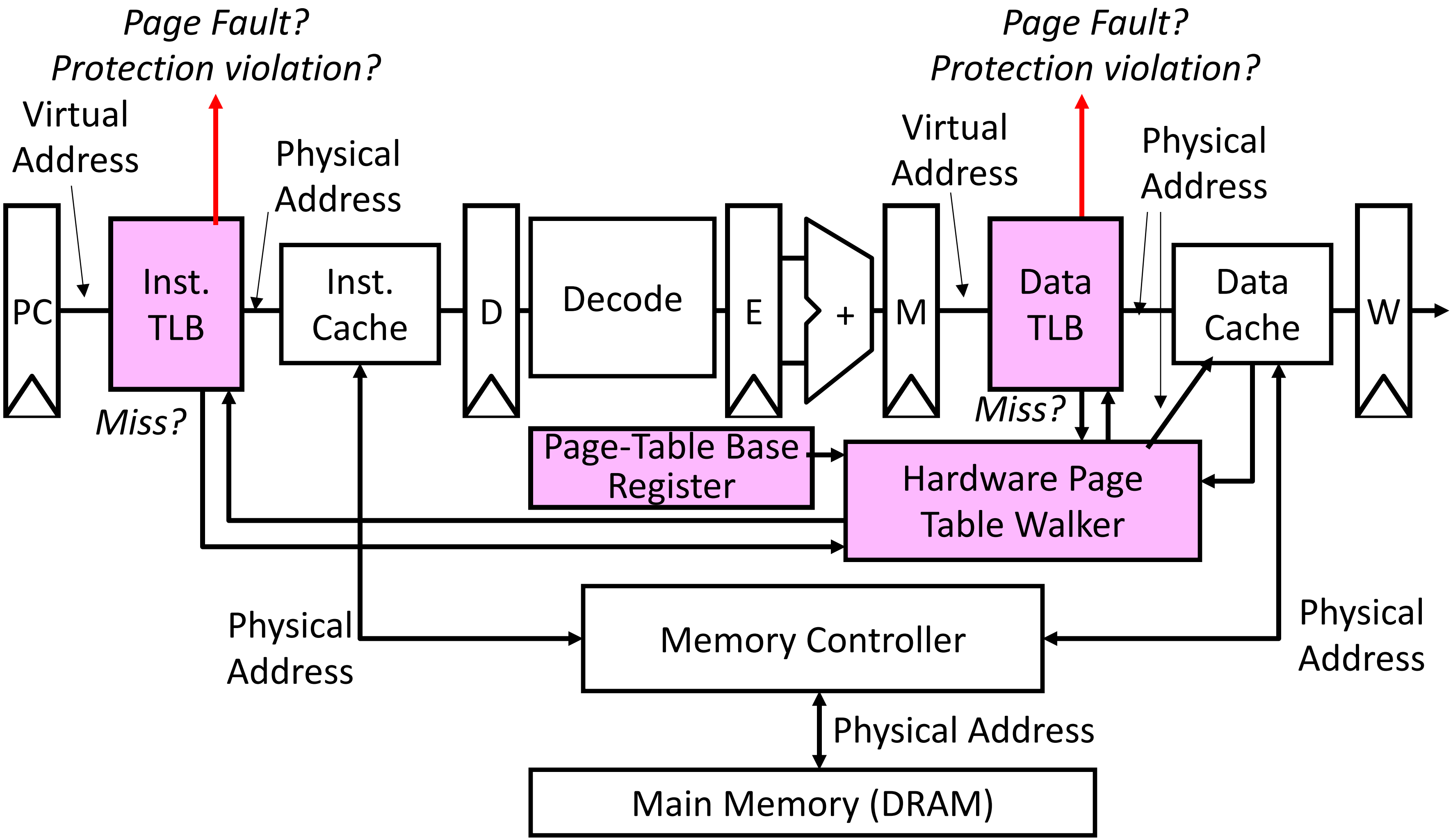
- Software (*MIPS, Alpha*)
  - TLB miss causes an exception and the operating system walks the page tables and reloads TLB. A privileged “untranslated” addressing mode used for walk.
  - Software TLB miss can be very expensive on out-of-order superscalar processor as requires a flush of pipeline to jump to trap handler.
- Hardware (*SPARC v8, x86, PowerPC, RISC-V*)
  - 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

Acknowledgements

- This course is partly inspired by previous MIT 6.823 and Berkeley CS252 computer architecture courses created by my collaborators and colleagues:
    - Arvind (MIT)
    - Krste Asanovic (MIT/UCB)
    - Joel Emer (Intel/MIT)
    - James Hoe (CMU)
    - John Kubiatowicz (UCB)
    - David Patterson (UCB)