ECE 252 / CPS 220
Advanced Computer Architecture I

Lecture 14
Virtual Memory

Benjamin Lee
Electrical and Computer Engineering
Duke University

www.duke.edu/~bcl15
www.duke.edu/~bcl15/class/class_ece252fall11.html

ECE252 Administrivia

20 October – Project Proposals Due

One page proposal

1. What question are you asking?
2. How are you going to answer that question?
3. Talk to me if you are looking for project ideas.

25 October – Homework #3 Due

25 October – Class Discussion

Roughly one reading per class. Do not wait until the day before!

1. Jouppi. "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers."
2. Kim et al. "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches."
3. Fromm et al. "The energy efficiency of IRAM architectures"
4. Lee et al. "Phase change memory architecture and the quest for scalability"

Last Time

Caches

- Quantify cache/memory hierarchy with AMAT
- Three types of cache misses: (1) compulsory, (2) capacity, (3) conflict
- Cache structure and data placement policies determine miss rates
- Write buffers improve performance

Prefetching

- Identify and exploit spatial locality
- Prefetchers can be implemented in hardware, software, both

Caches and Code

- Restructuring SW code can improve HW cache performance
- Data re-use can improve with code structure (e.g., matrix multiply)

Physical Addresses

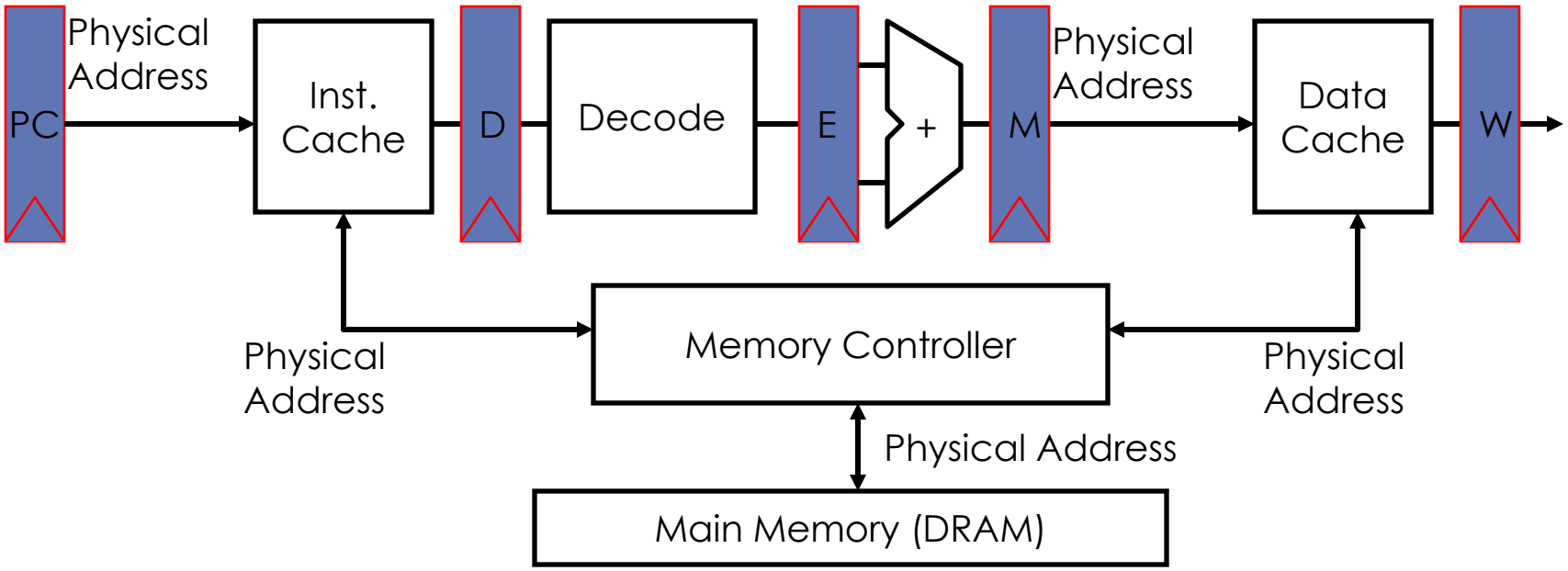
One program runs at a time. Program has unrestricted access to machine (RAM, I/O)

Program addresses depend on where program is loaded into memory

But programmers do not think about memory addresses when writing sub-routines…

- How is location independence achieved? Loader/Linker
- (1) loads sub-routines into memory, determines physical addresses,
- (2) links multiple sub-routines,
- (3) resolves physical addresses so jumps to sub-routines go to correct memory locations

Machine with Physical Addresses



With unrestricted access to a machine, program uses physical addresses.

Address Translation

Motivation

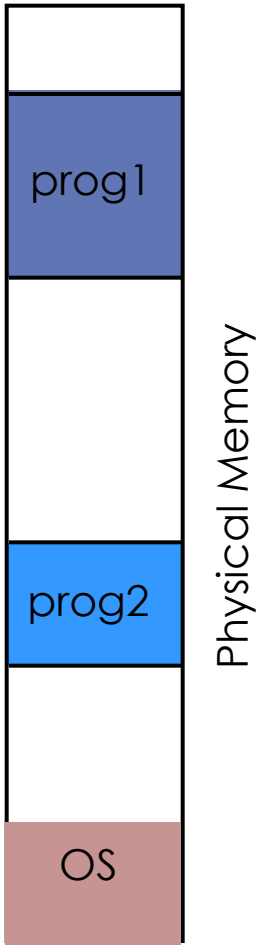
- In early machines, I/O is slow and requires processor support
- Increase throughput by over-lapping I/O for multiple programs
- Eliminate processor support with direct memory access (DMA)
- DMA: (1) Processor initiates I/O, (2) Processor does other operations during transfer, (3) Processor receives interrupt from DMA controller when transfer is complete

Location-Independent Programs

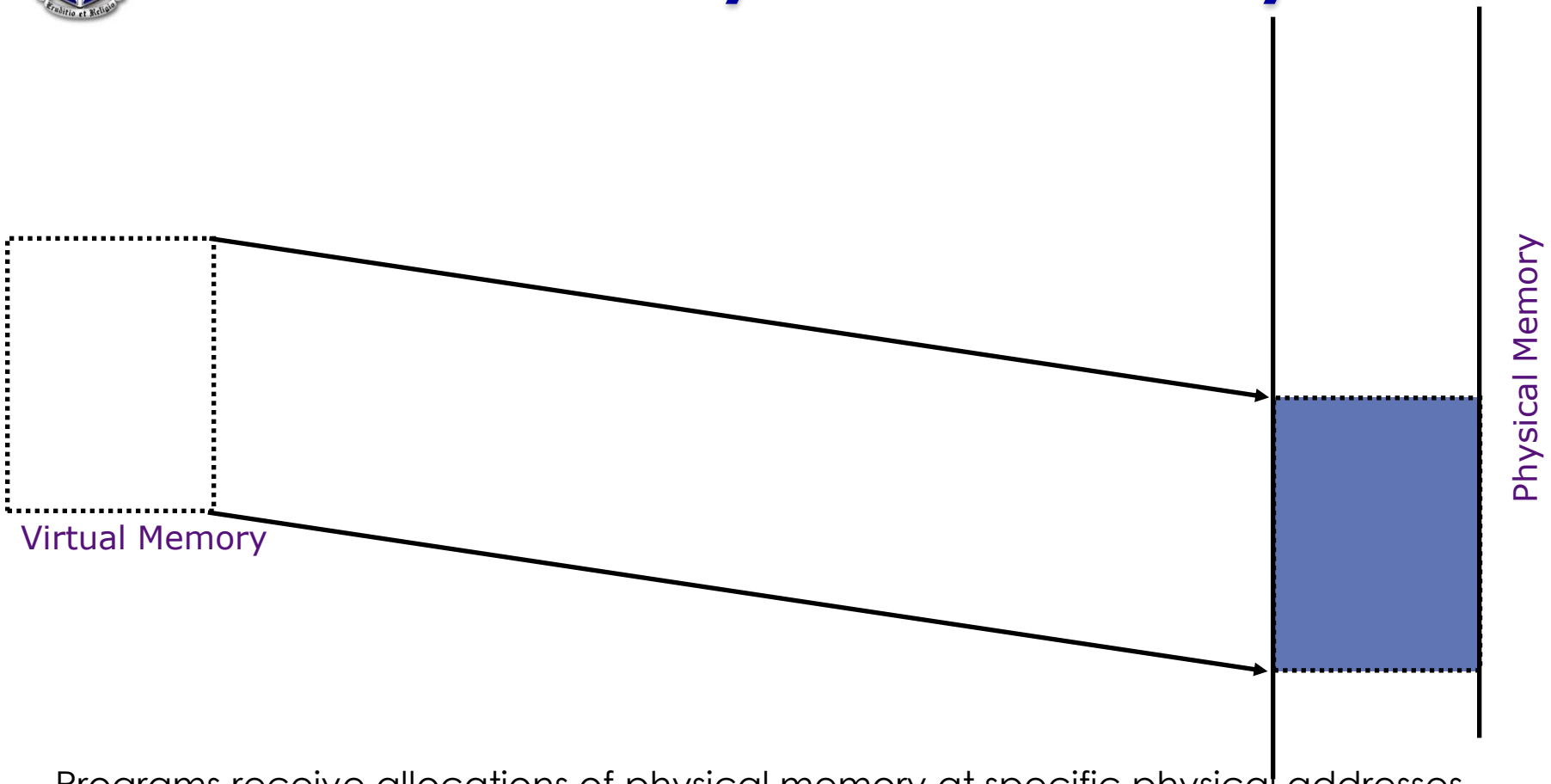
- Simplify program and storage management

Protection

- Independent programs should not affect others (isolated memory spaces)
- Multi-programming requires "supervisor" to schedule programs, manage context switches between programs



Virtual and Physical Memory

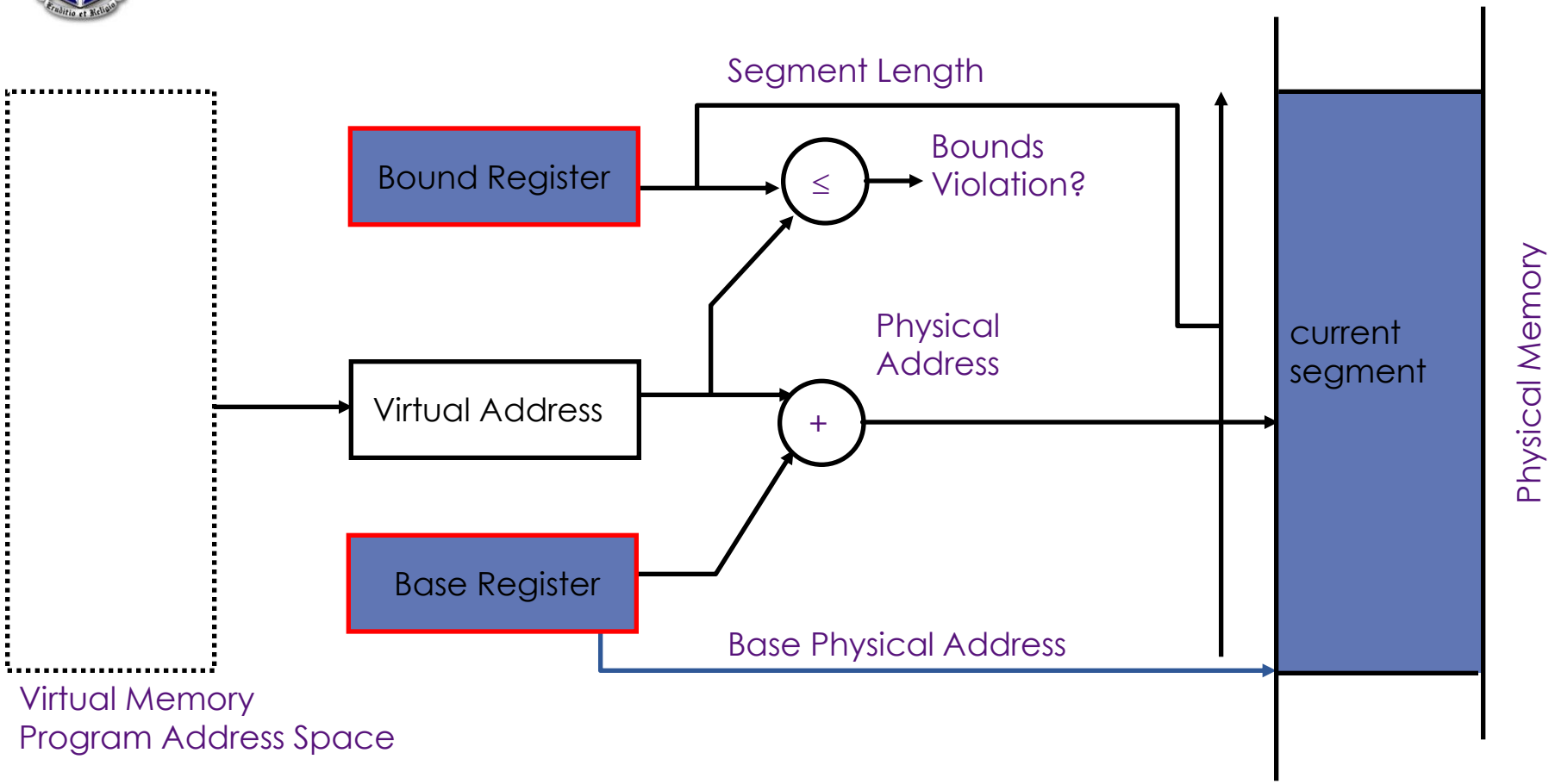


Programs receive allocations of physical memory at specific physical addresses.

Program operates in virtual memory with virtual addresses.

Memory Management Unit (MMU) – performs address translation to map virtual addresses to physical addresses.

1 - Base & Bound

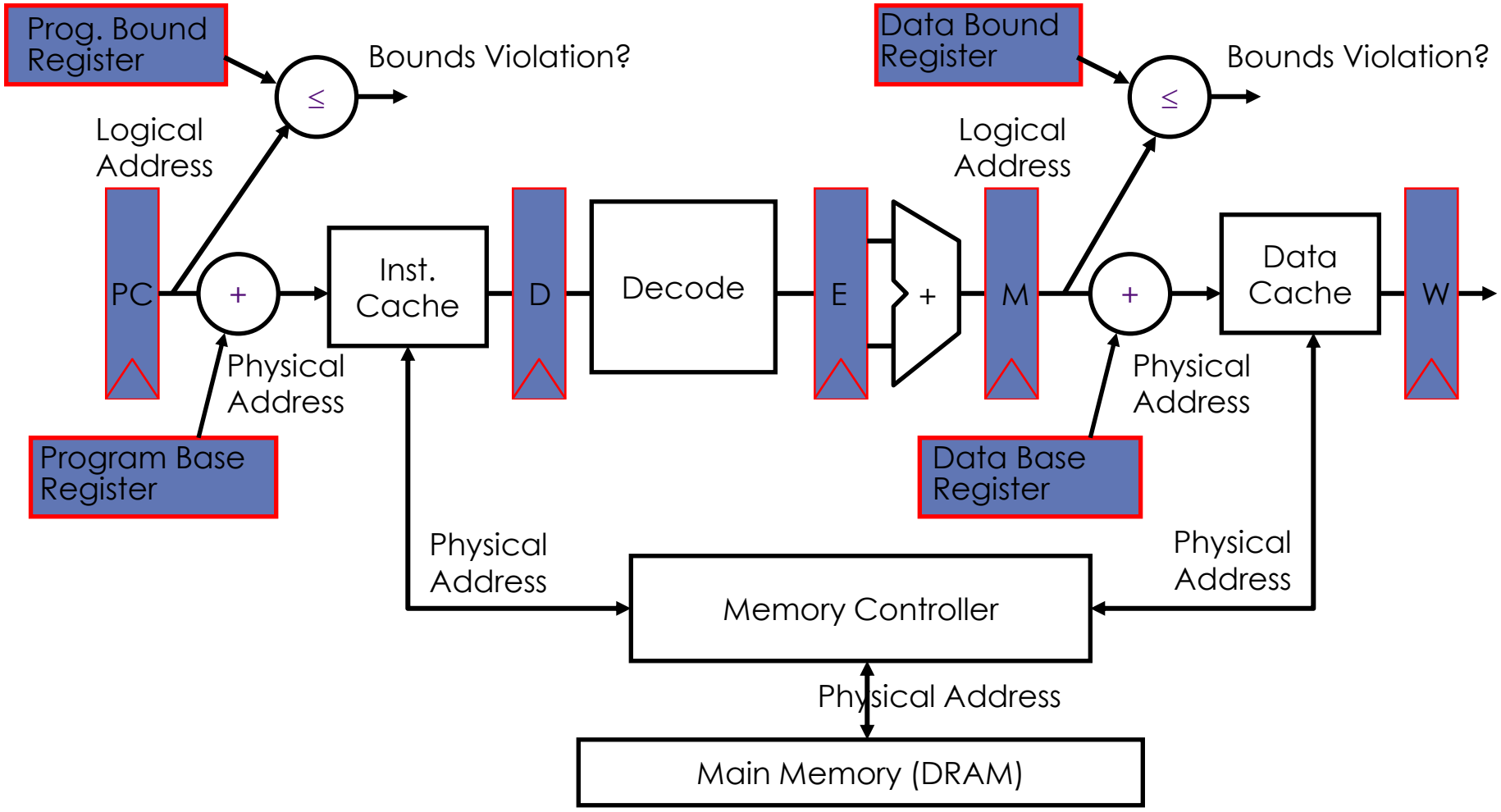


Base register – Identifies a program's allocation in physical memory.

Bound register – Provides protection and isolation between programs

Base, Bound registers visible only when processor is running in "supervisor" mode

Machine with Virtual Addresses



Every access to memory requires translating virtual addresses to physical addresses. Efficient adder implementations are possible for (base+offset).

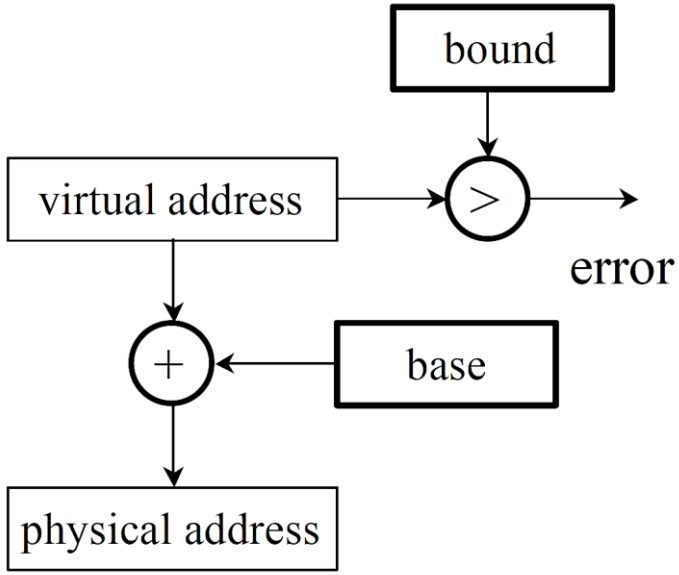
Base & Bounds Implementation

Hardware Cost

- Two registers, adder, comparator
- Fast logic

Context Switch

- Save/restore base, bound registers



2 - Segmentation

Motivation

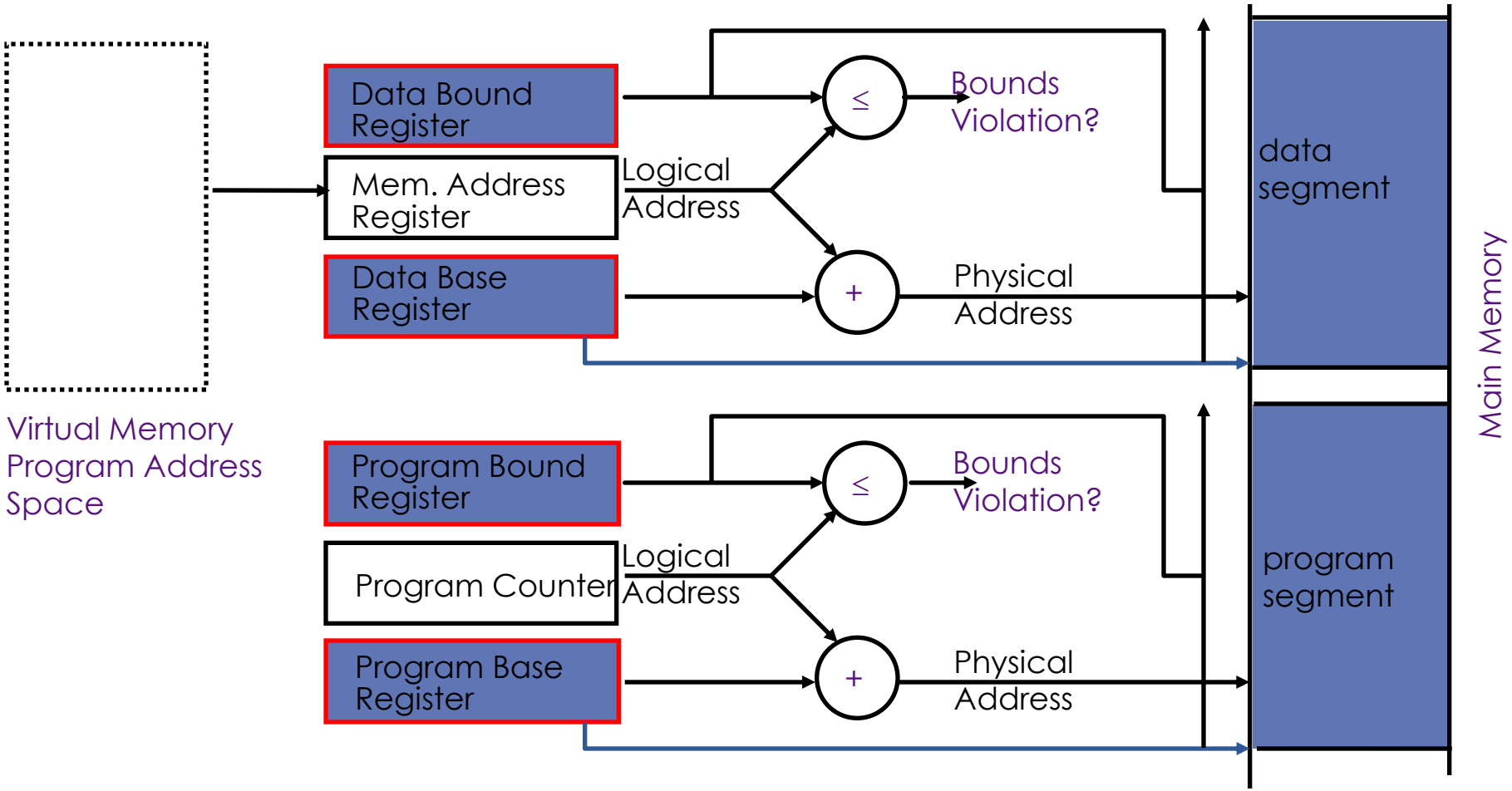
- Separate virtual address space into several segments.
- Each segment is contiguous but segments may be fragmented.
- Segmentation does not require fully contiguous address space (i.e., allow holes in address space)

Idea

- Generalize Base & Bounds
- Implement a table of base-bound pairs

Virtual Segment #		Physical Segment Base	Segment Size
Code	(00)	0x4000	0x700
Data	(01)	0x0000	0x500
-	(10)	0	0
Stack	(11)	0x2000	0x1000

Data & Program Segments



What is the advantage of this separation?

Segmentation Implementation

Virtual Address

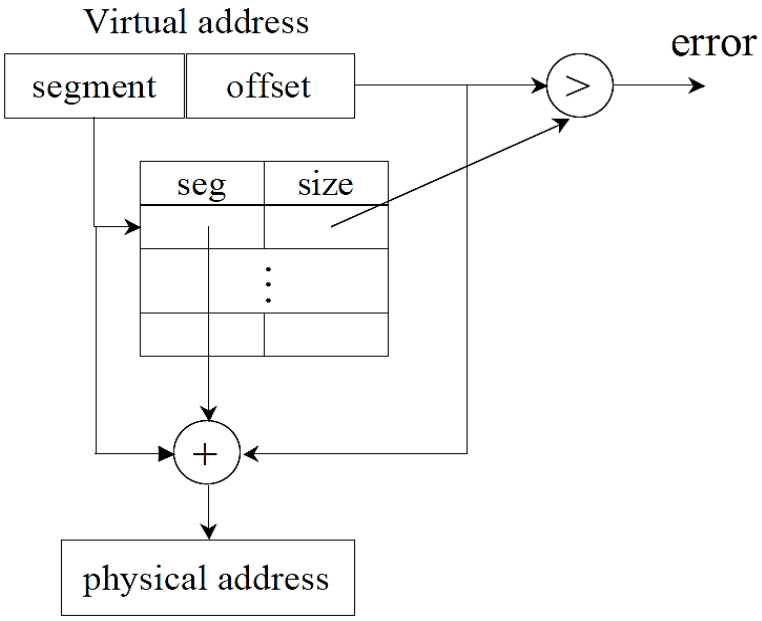
- Partition into segment and offset
- Segment – Specifies segment number, which indexes table
- Offset – Specifies offset within a segment

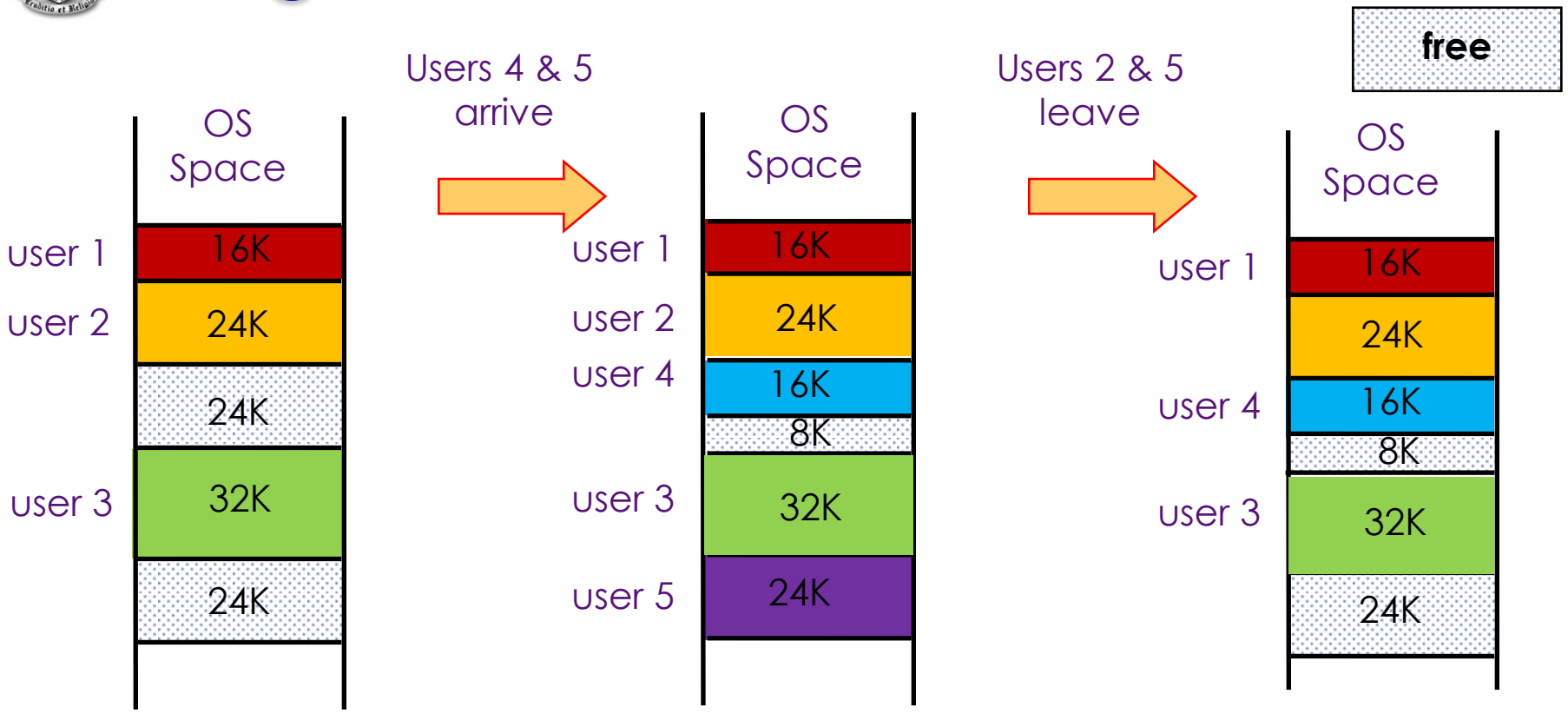
Segment Table

- Segment – Provides segment base
- Size – Provides segment bound

Translation

- Compute physical address from segment base, offset, and bound





Fragmentation

- As programs enter and leave the system, physical memory is fragmented.
- Fragmentation occurs because segments are variable size

3 - Paging

Motivation

- Branch&Bounds, Segmentation require fancy memory management
- Example: What mechanism coalesces free fragments?

Idea

- Constrain segmentation with fixed-size segments (e.g., pages)
- Paging simplifies memory management

- Example: free page management is a simple bitmap
- 00111111100000011100
- Each bit represents on page of physical memory
- 1 means allocated, 0 means free

Paging Implementation

Virtual Address

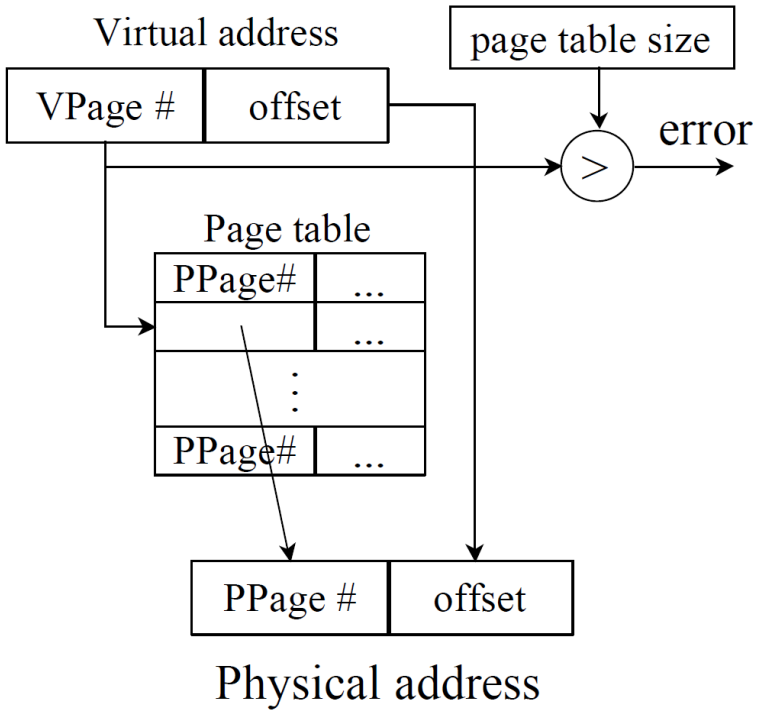
- Partition into page and offset
- Page – Specifies virtual page number, which indexes table
- Offset – Specifies offset within a page

Page Table

- Page – Provides a physical page number
- Size – No longer required, all pages equal size (e.g., 4KB)

Translation

- Compute physical address from physical page number, offset



4 – Segmentation & Paging

Motivation

- Assume 32-bit virtual addresses, 4KB page size
- 4KB = 4096 bytes = 2^{12} bytes per page
- 2^{12} bytes per page requires a 12-bit offset
- Remaining address bits used for page number, 20-bit page number

- Page tables can be very large.
- With 20-bit page number, 2^{20} pages
- Each program in multi-programmed machine requires its own page table
- Total size of page tables = [# of programs] x 2^{20}

Idea

- Combine segmentation with paging
- Adds indirection to reduce size of page table

Segmented Page Tables

Virtual Address

- Partition into segment, page, offset
- Segment –Specifies segment#, which indexes Table 1
- Page – Specifies virtual page # number, which indexes Table 2
- Offset – Specifies offset in page

Table 1

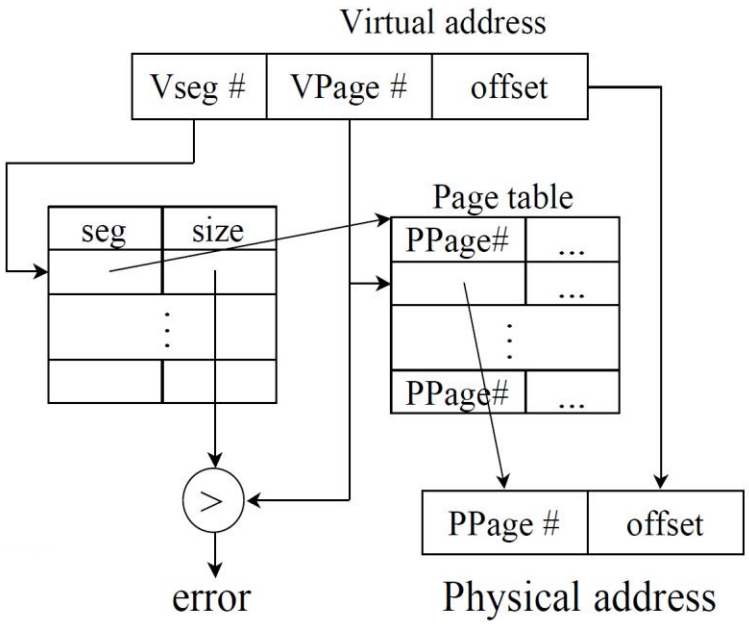
- Segment – Points to a page table
- Size – Specifies number of pages in segment

Table 2

- Page – Provides a physical page#

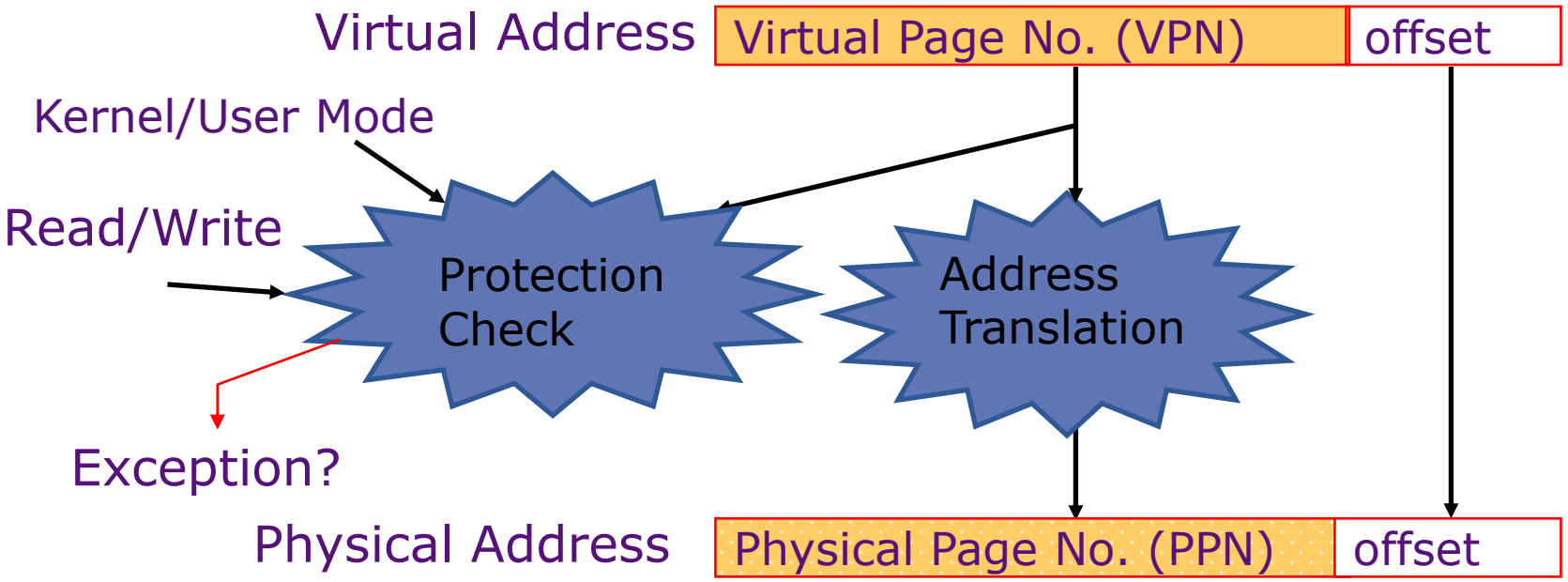
Translation

- Compute physical address from physical page number, offset



What about paged page tables?

Address Translation & Protection



- Every access to instruction/data memory requires (1) address translation and (2) protection checks
- A good virtual memory system must be fast (completing in approximately 1 cycle) and space efficient

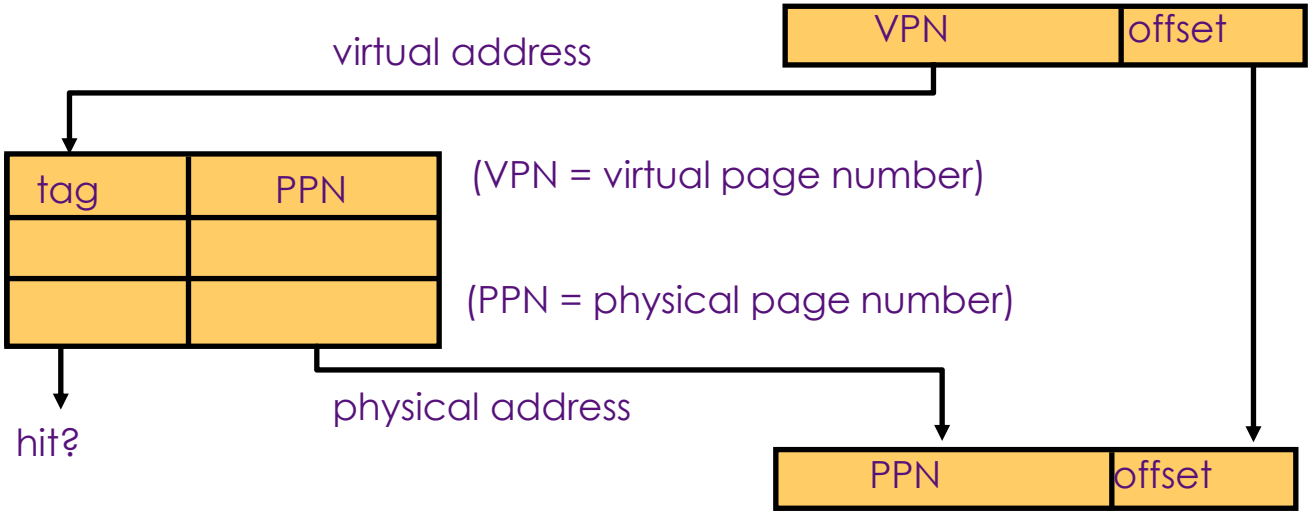
Translation Lookaside Buffer (TLB)

Address Translation is Expensive

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

Solution

- Cache translations. We use a data structure called a TLB.
- TLB Hit – single cycle translation
- TLB Miss – walk page table to translate, update TLB



TLB Design

32-128 entries, fully associative

- Each entry maps a large page. Little spatial locality across 4KB pages. Conflicts are more likely.
- Larger TLBs (e.g., 256-512 entries) may be 4-8 way set-associative
- Even larger systems may have multi-level TLBs

Random or FIFO replacement policies

Definition – TLB Reach

- Size of largest virtual address space that can be simultaneously mapped
- 64 entries, 4KB pages, 1 page per entry
- TLB Reach = [64 entries] x [4KB] = 256KB (if pages contiguous in memory)

TLB Misses

Software (MIPS, Alpha)

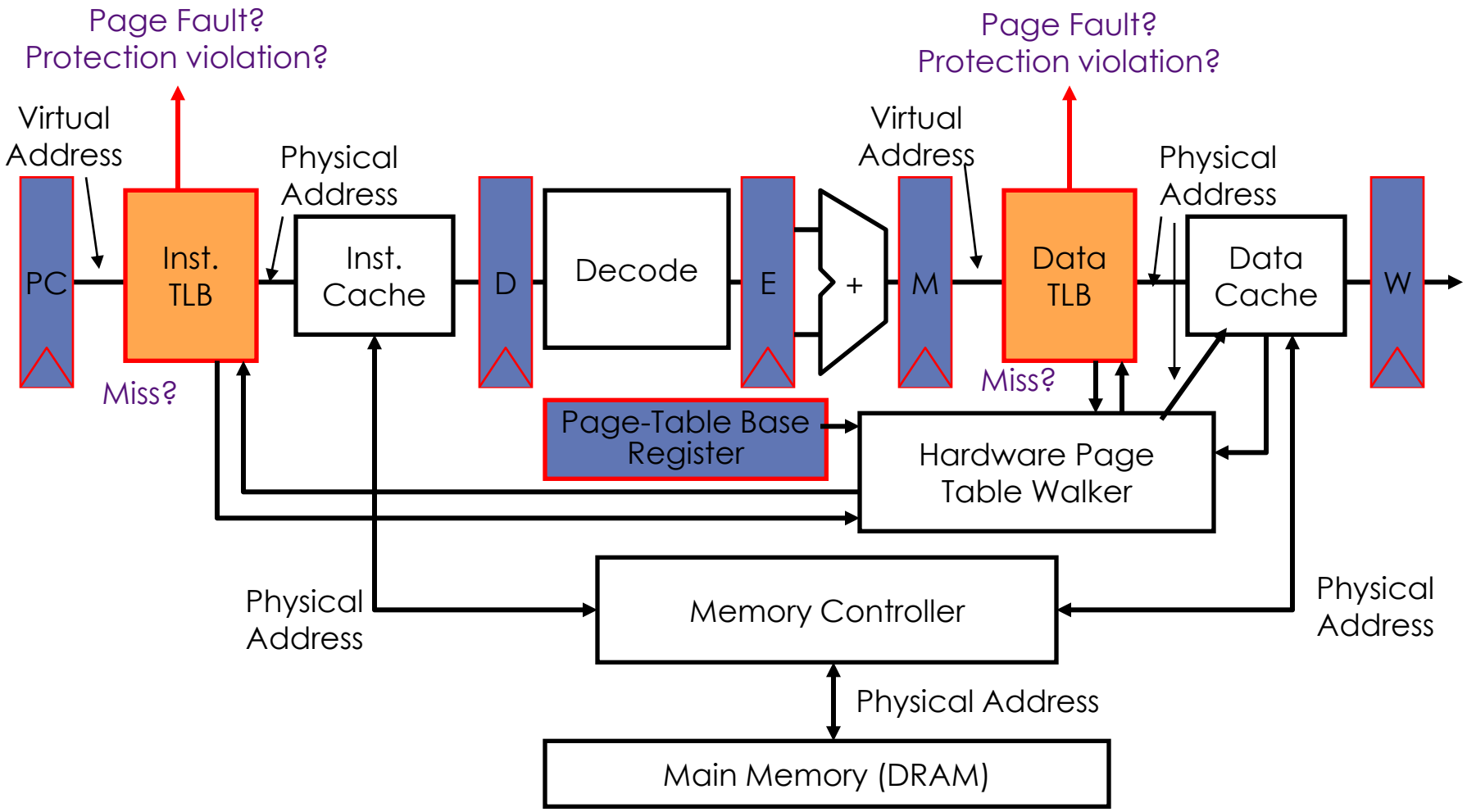
- TLB miss causes an exception
- Operating system walks page table and reloads TLB
- Requires privileged processor mode. Why?

Hardware (SPARC v8, x86, PowerPC)

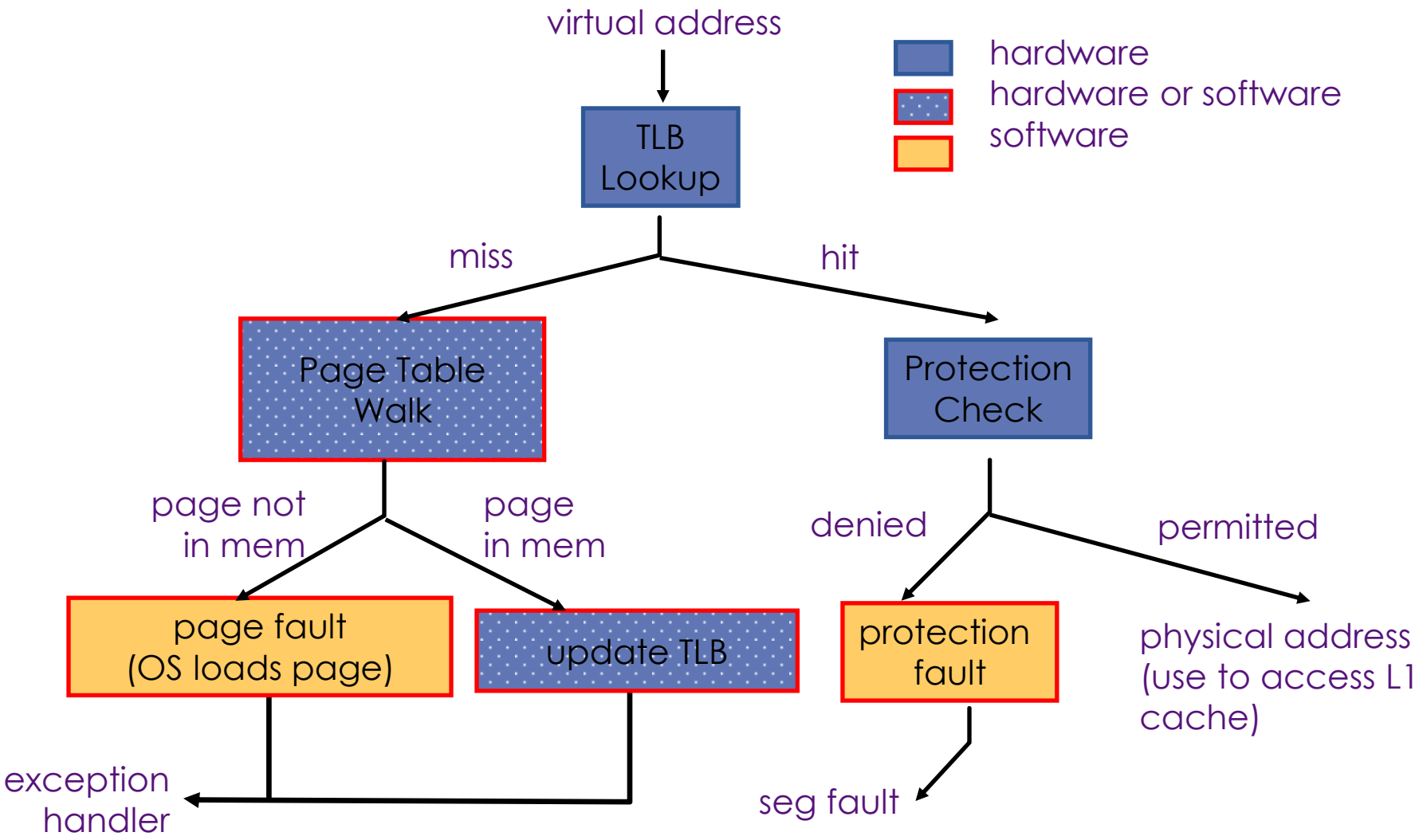
- Memory management unit (MMU) walks page table and reloads TLB
- If MMU encounters missing page, MMU causes an exception

- Operating system handles page fault, which requires transferring page from disk to memory.

Machine with Virtual Memory



Address Translation Summary



Summary

Virtual Memory

- Enables multi-programming
- Programs operate in virtual memory space
- Programs are protected from each other

Virtual to Physical Address Translation

- Base&Bound
- Segmentation
- Paging
- Multi-level Translation (segmented paging, paged paging)

Translation Lookaside Buffer

- Accelerates virtual memory, address translation

Acknowledgements

These slides contain material developed and copyright by

- Arvind (MIT)
- Krste Asanovic (MIT/UCB)
- Joel Emer (Intel/MIT)
- James Hoe (CMU)
- Arvind Krishnamurthy (U. Washington)
- John Kubiatowicz (UCB)
- Alvin Lebeck (Duke)
- David Patterson (UCB)
- Daniel Sorin (Duke)