ECE 552 / CPS 550
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_ece252fall12.html

ECE552 Administrivia

19 October – Homework #3 Due

19 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.

23 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

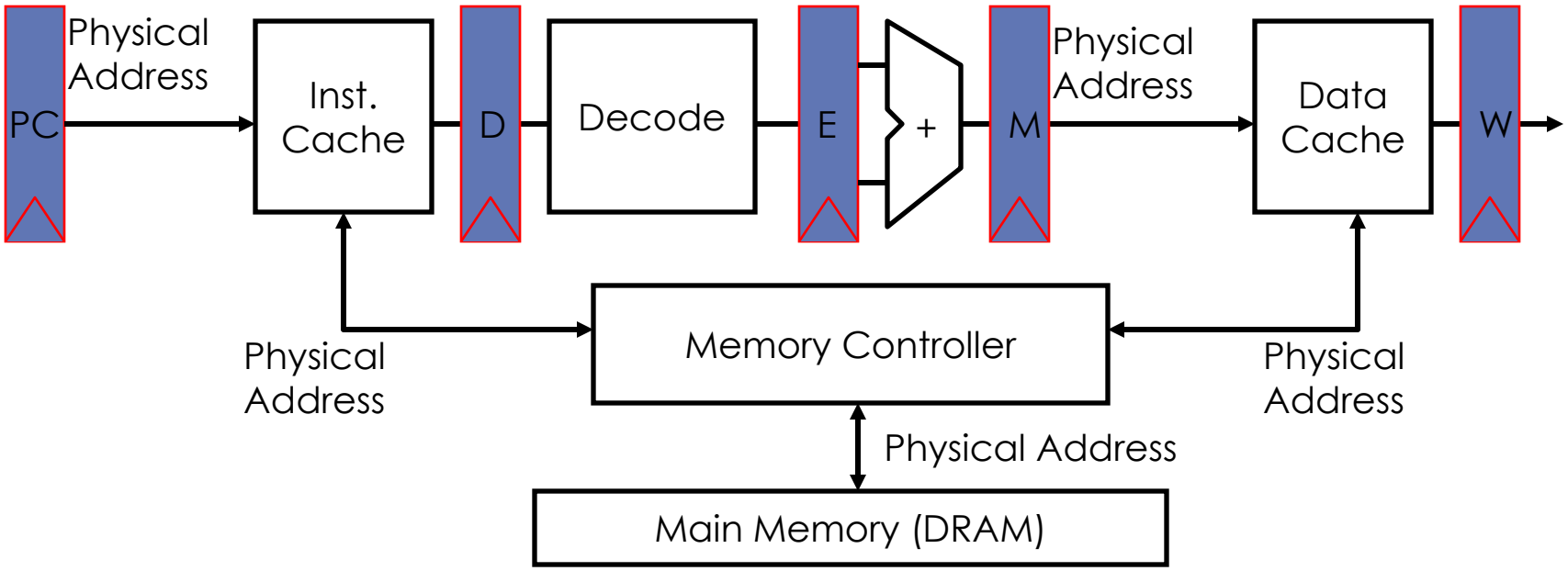
Physical addresses determined when program is loaded into memory.

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

- Location Independence – How do we write programs without knowing physical addresses

- For code, addresses determined by 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.

Translated Addresses

Motivation

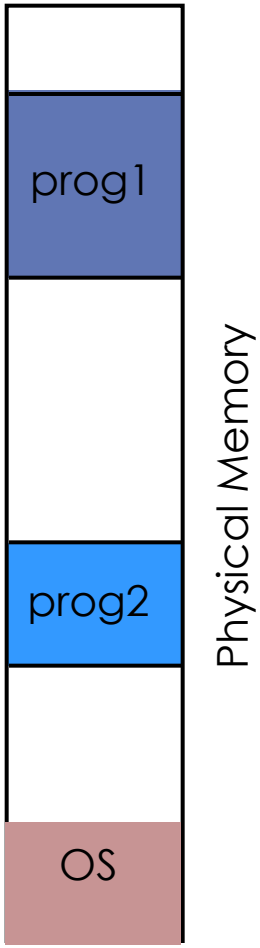
- In early machines, I/O is slow and requires processor support
- Slow I/O: Mitigate by over-lapping I/O from different programs
- Processor Support: Mitigate with DMAs

Direct Memory Accesses (DMAs)

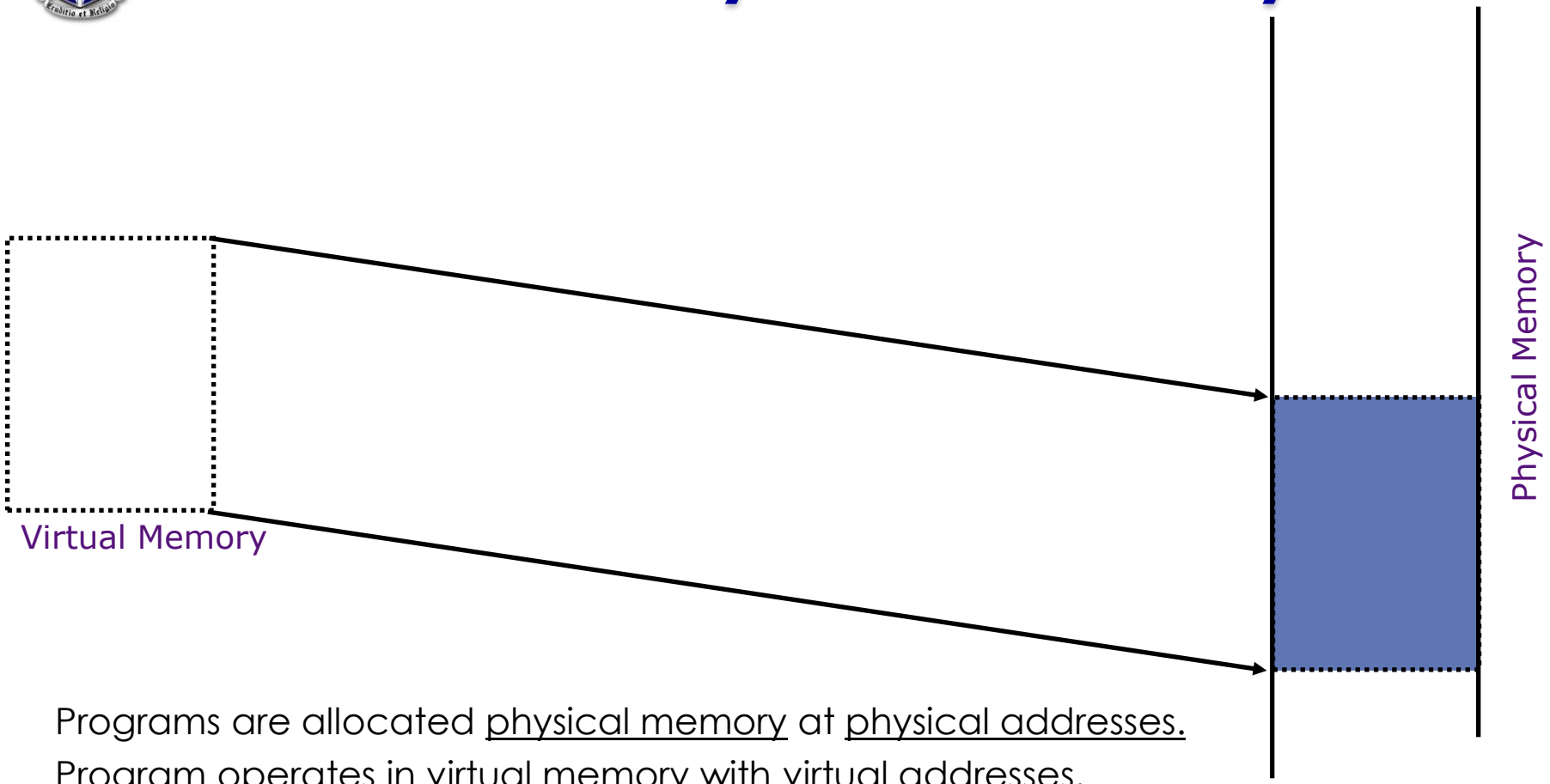
- Processor invokes DMA controller to perform I/O
- Processor does other operations during transfer
- Processor interrupted by DMA controller when transfer complete

Multi-programming Support

- Overlapped I/O and DMAs require multi-programming support
- Supervisor schedules programs, manage context switches
- Address translation provides location-independent programs.
- Address translation provides isolated memory spaces.



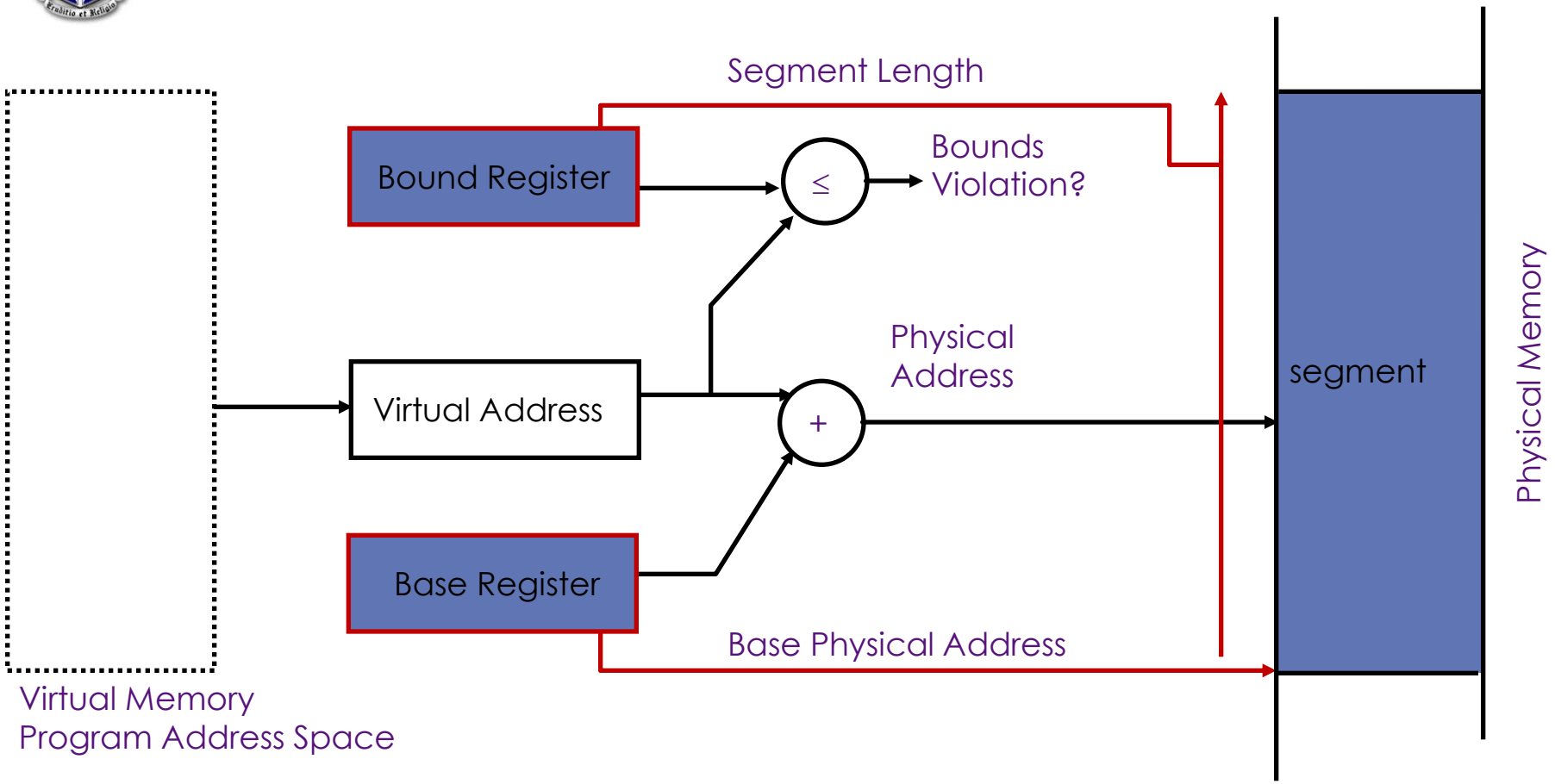
Virtual and Physical Memory



Programs are allocated physical memory at physical addresses.
Program operates in virtual memory with virtual addresses.

We require address translation to map virtual addresses to physical addresses.
This lecture provides 4 increasingly sophisticated translation techniques

1 - Base & Bound

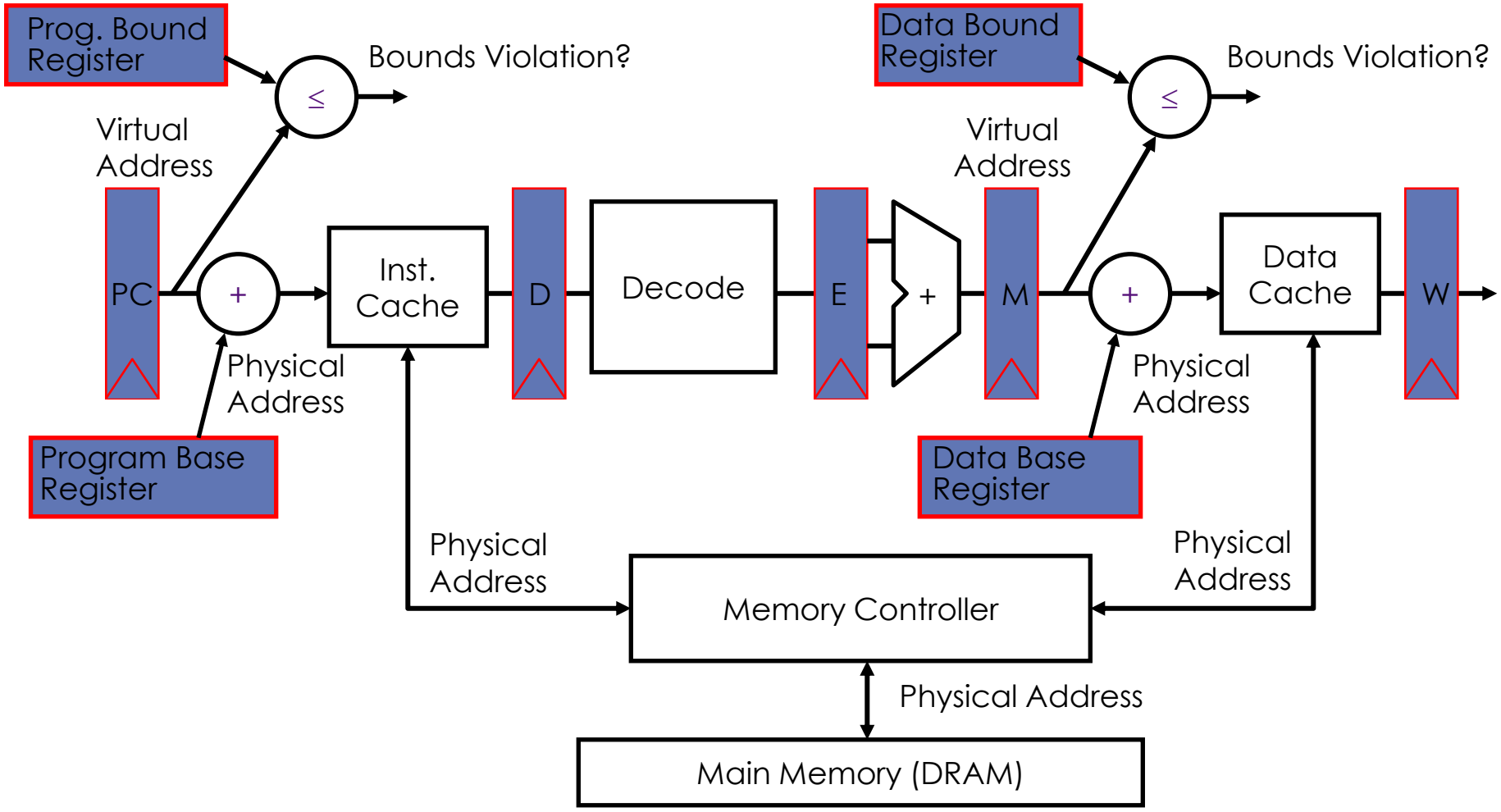


Base register – Identifies start of program's allocation in physical memory.

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

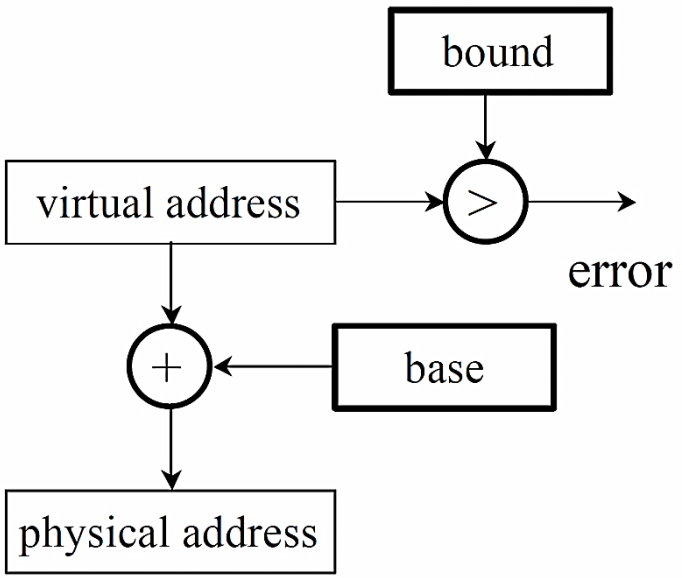
Base & Bounds Implementation

Hardware Cost

- Two registers, adder, comparator
- Fast logic

Context Switch

- Switch between programs
- Save/restore base, bound registers



2 - Segmentation

Motivation

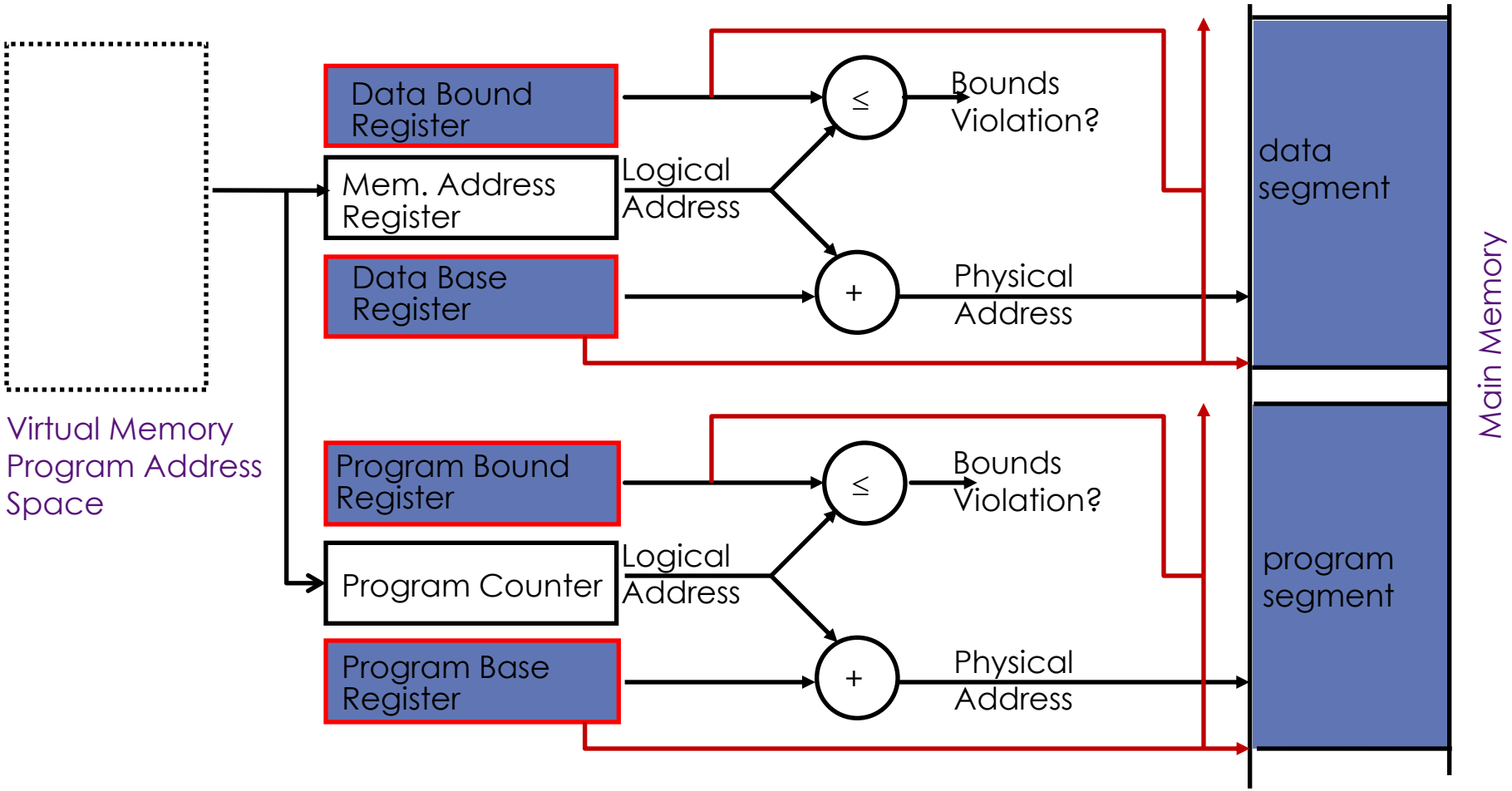
- Base&Bounds assumes one contiguous memory segment per program.
- Segmentation separates address space into several segments.
- Each segment is contiguous but multiple segments may be required.

Idea

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

Virtual Segment #		(Base) Physical Segment Base	(Bound) 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

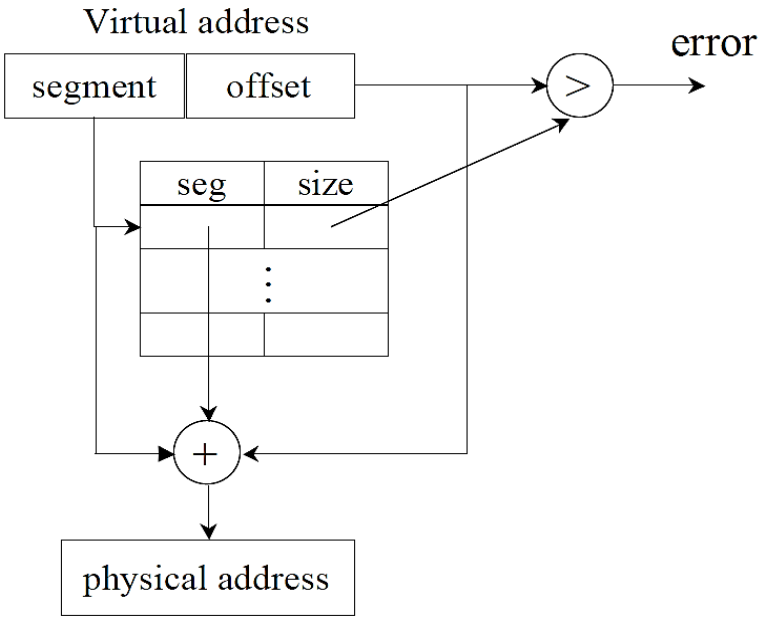
- Partition into segment and offset
- Segment – Specifies segment number, which indexes table
- Offset – Specifies offset within a segment

Segment Table

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

Translation

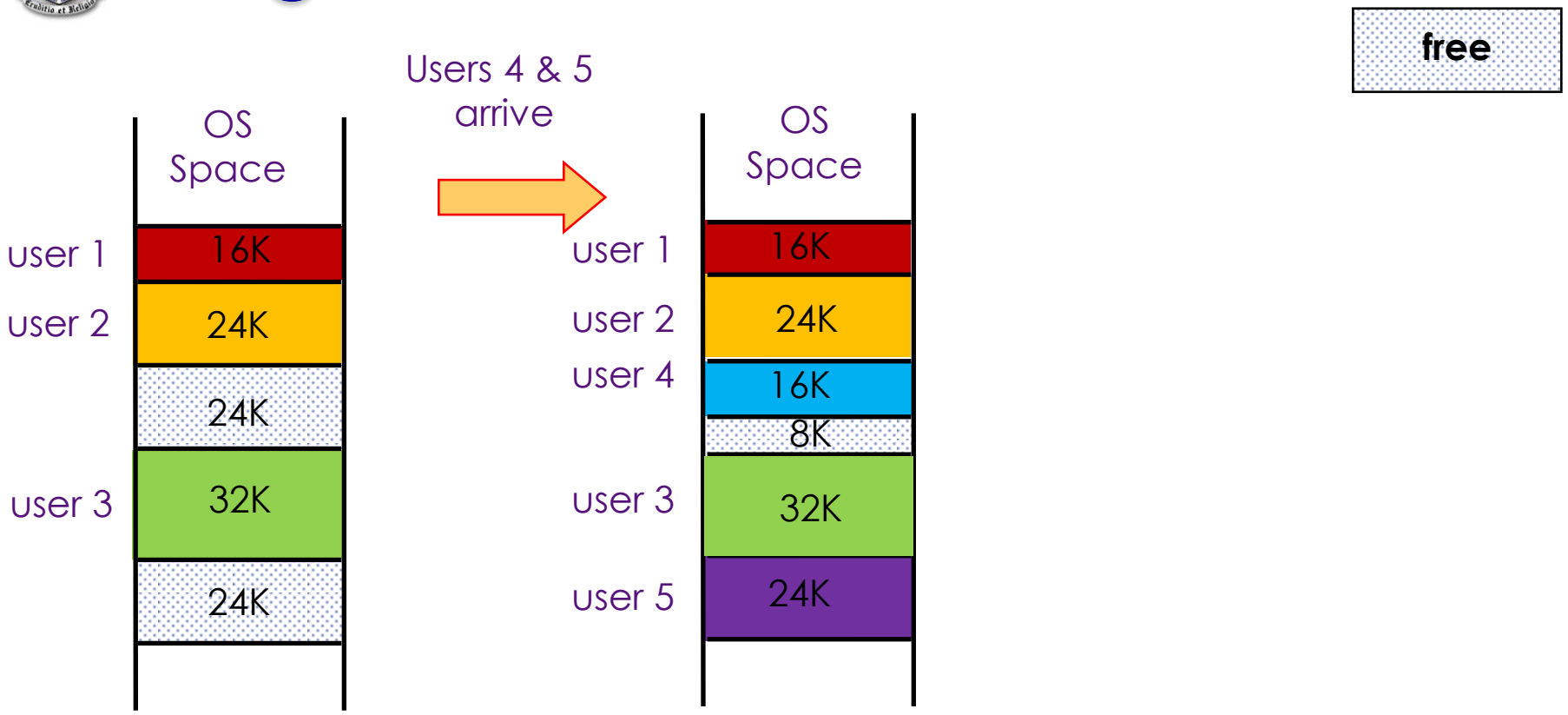
- Compute physical address from segment base, offset, and bound



Fragmentation

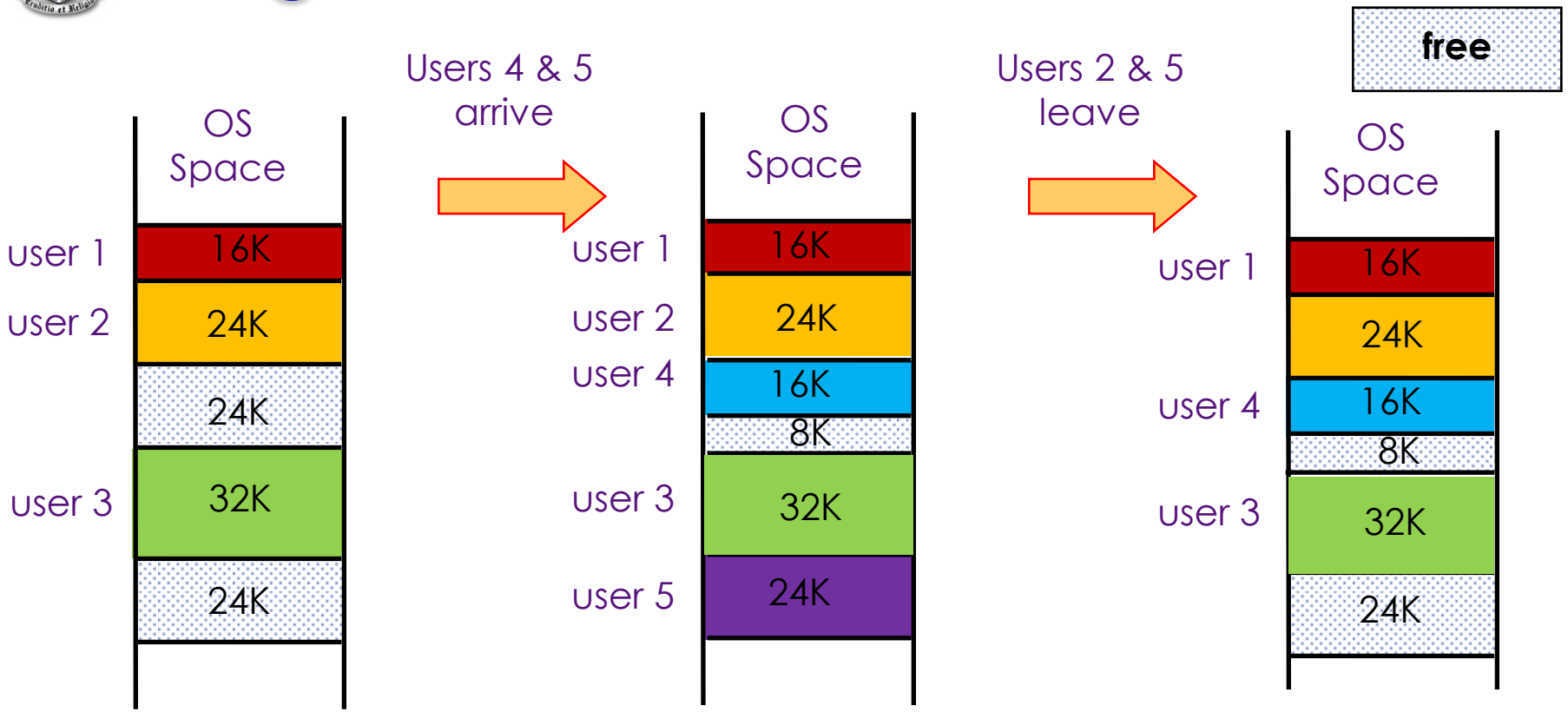
- As programs enter and leave the system, physical memory is fragmented.
- Fragmentation occurs because segments are variable size

Fragmentation



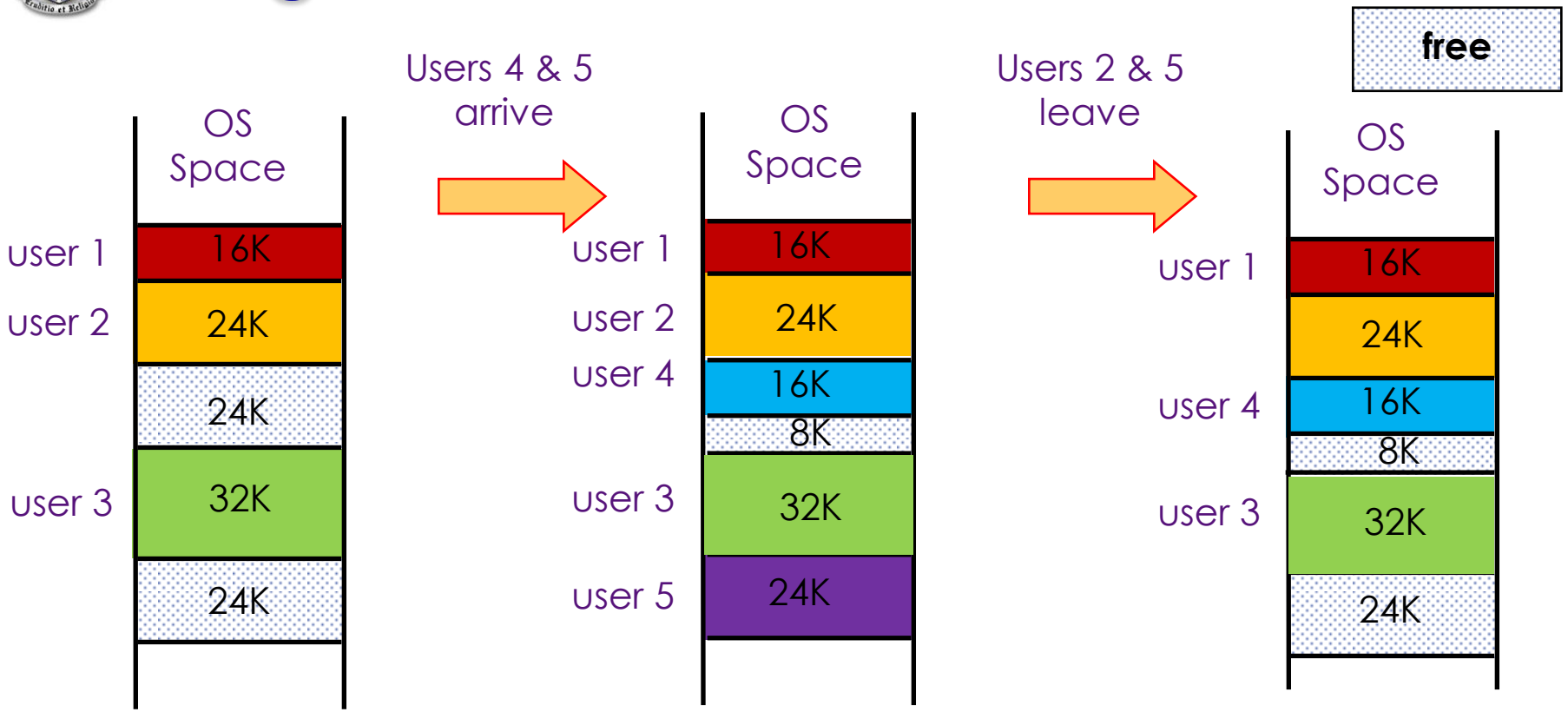
- As programs enter and leave the system, physical memory is fragmented.
- Fragmentation occurs because segments are variable size

Fragmentation



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

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 a page of physical memory
- 1 means allocated, 0 means free

Paging Implementation

Virtual Address

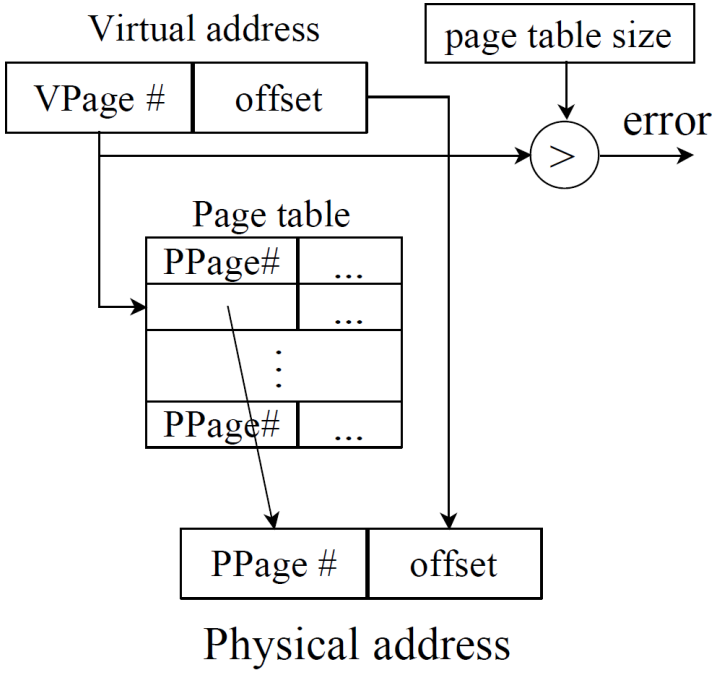
- Partition into page and offset
- Page – Specifies virtual page number, which indexes table
- Offset – Specifies offset within a page

Page Table

- Page – Provides a physical page number
- Size – Not required, pages equal size (e.g., 4KB)

Translation

- Compute physical address from physical page number, offset



4 – Segmentation & Paging

Motivation

- Page tables can be very large.

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

- Each page requires a page table entry
- With 20-bit page number, 2^{20} pages addressed
- Each program in multi-programmed machine requires its own page table
- Total size of page tables = [# of programs] x 2^{20} entries

Idea

- Page tables reside in memory
- Segmentation with paging and indirection to reduce page table size
- (Alternatively use indirect page tables, which hashes VPN to PPN)

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

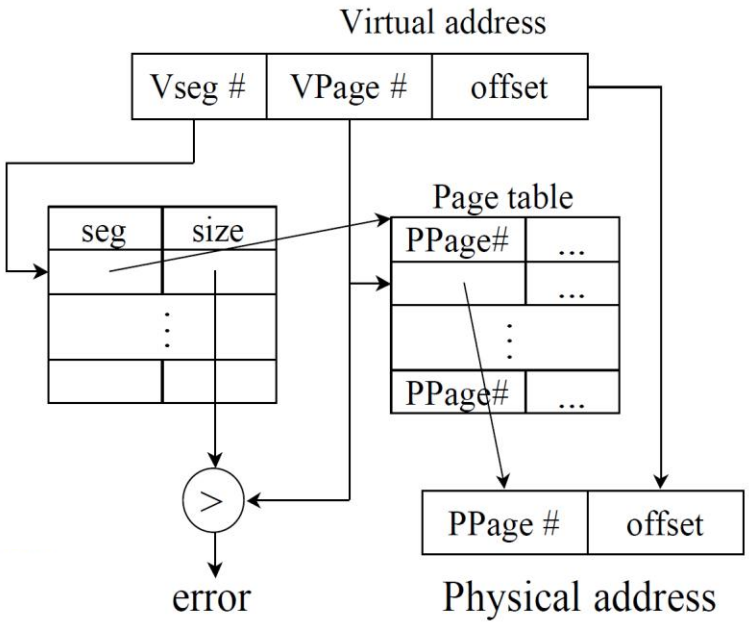
- Segment – Points to a page table
- Size – Specifies number of pages in segment

Table 2

- Page – Provides a physical page#

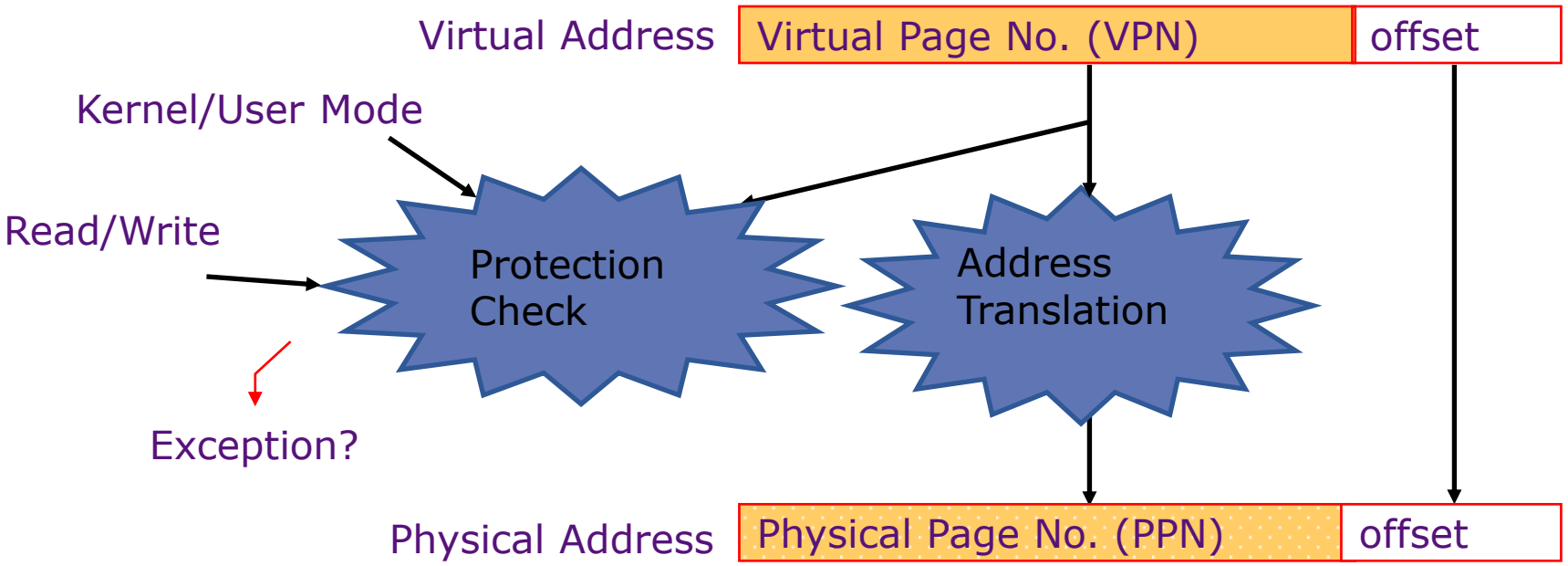
Translation

- Compute physical address from physical page number, offset



What about paged page tables?

Address Translation & Protection



- Every access to memory requires
  (1) address translation
  (2) protection check
- A good virtual memory system must be fast (e.g., 1 cycle), space efficient

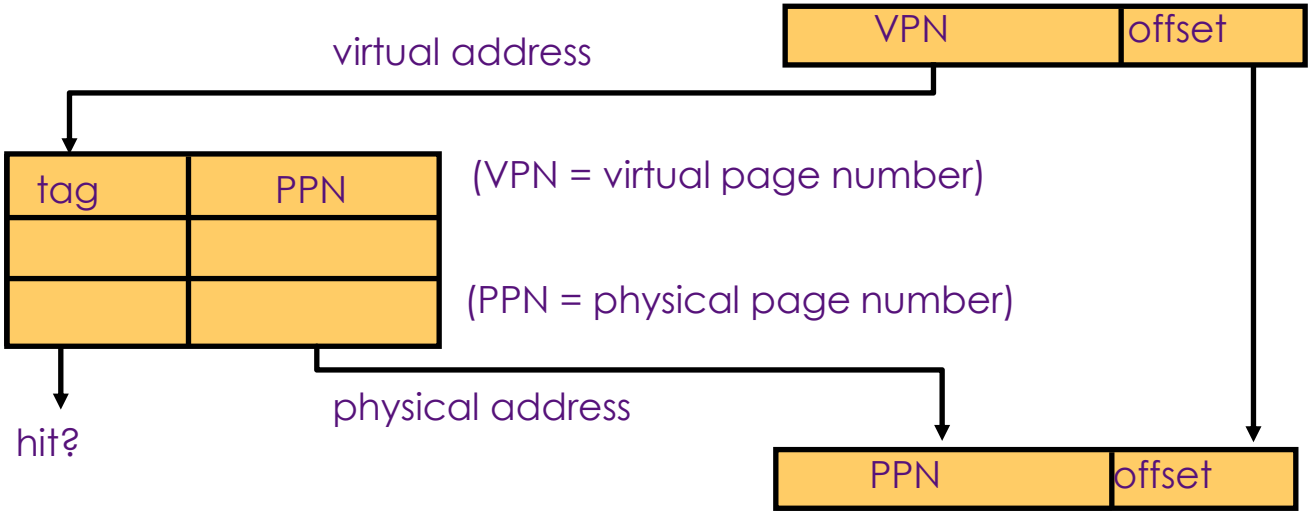
Translation Lookaside Buffer (TLB)

Address Translation is Expensive

- In a two-level page table, 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

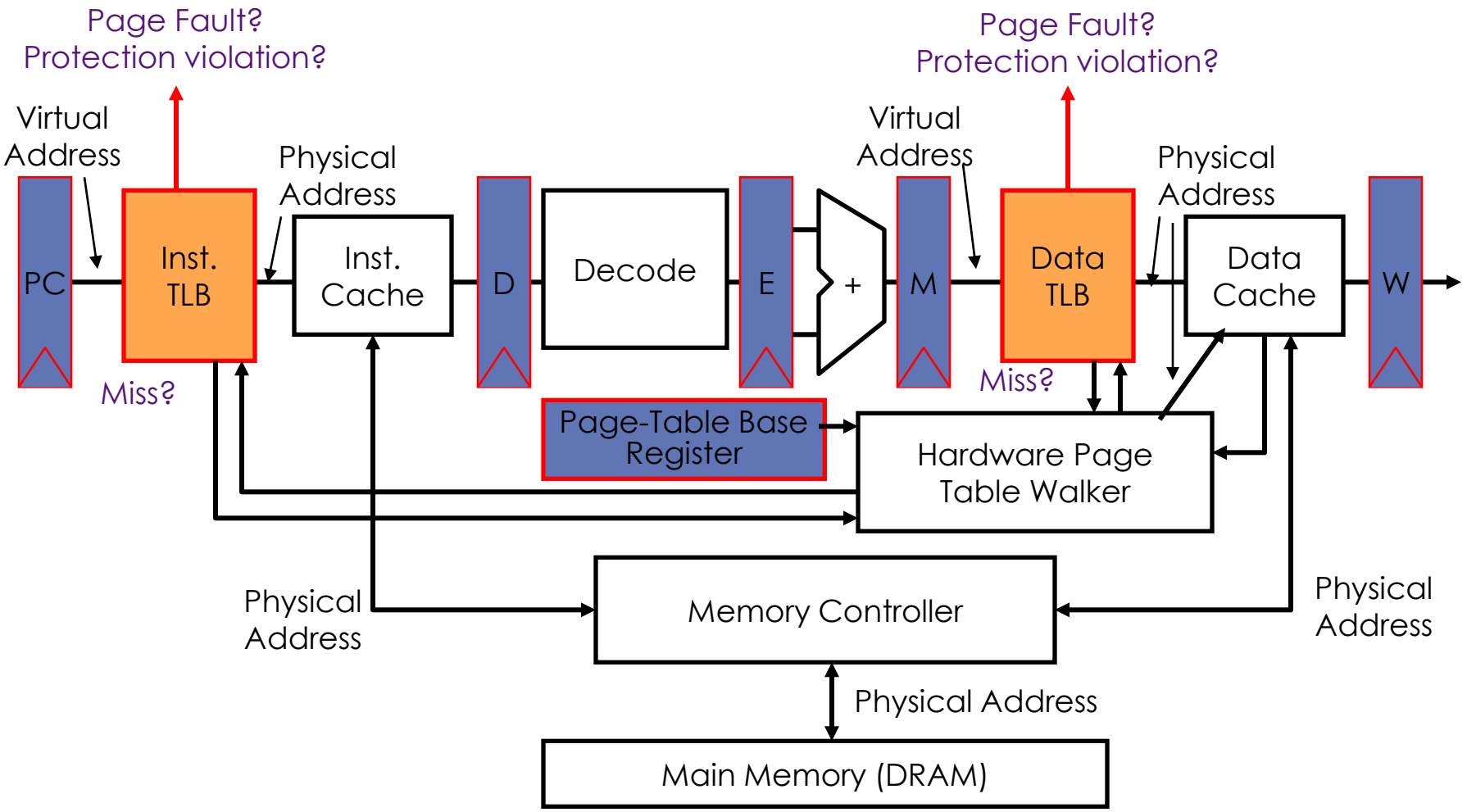
- Each entry maps a large page.
- Little spatial locality across 4KB pages so associative caches preferred.
- Larger TLBs (e.g., 256-512 entries) may be 4-8 way set-associative
- Even larger systems may have multi-level TLBs

Random or FIFO replacement policies

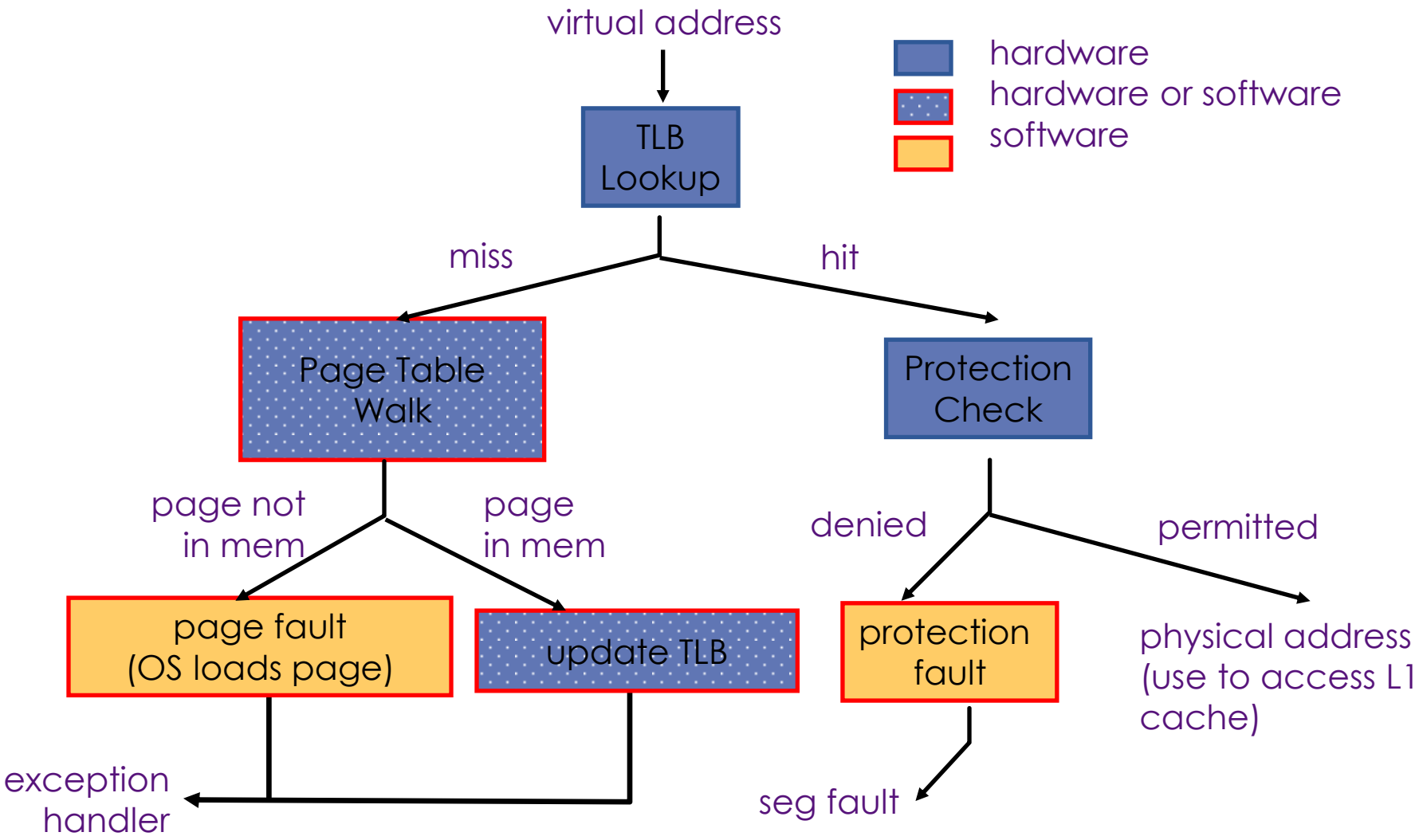
Definition – TLB Reach

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