Operating Systems

Lecture 3
Physical Memory Management

IIIS & CS
Tsinghua University

Acknowledgement:
materials from Dr. Zhang Yong Guang in MSRA,
And from http://williamstallings.com/OS/OS5e.html , http://www.os-book.com

Review

- Dual Mode Operation
- What is an Interrupt/Exception/System Call?
- The difference of Interrupt/Exception/System Call
- X86 related
  - How to build IDT
  - The hardware processing when INT happens
  - The software processing when INT happens
  - The system call processing (non-privilege(user) mode /privilege(supervisor) mode)
  - The different stacks in different privilege mode

Review: Dual-mode operation

- Why do we have “user mode” and “kernel mode”?
- Problem: Would you trust any users to … read and write memory, manage resource, access I/O, …?
- Solution: dual mode operation
  - CPU has a “mode” when it is executing an instruction
  - “User Mode”: can only perform a restricted set of operation (applications)
  - “Kernel Mode”: can do anything (OS kernel)

From “User Mode” to “Kernel Mode”

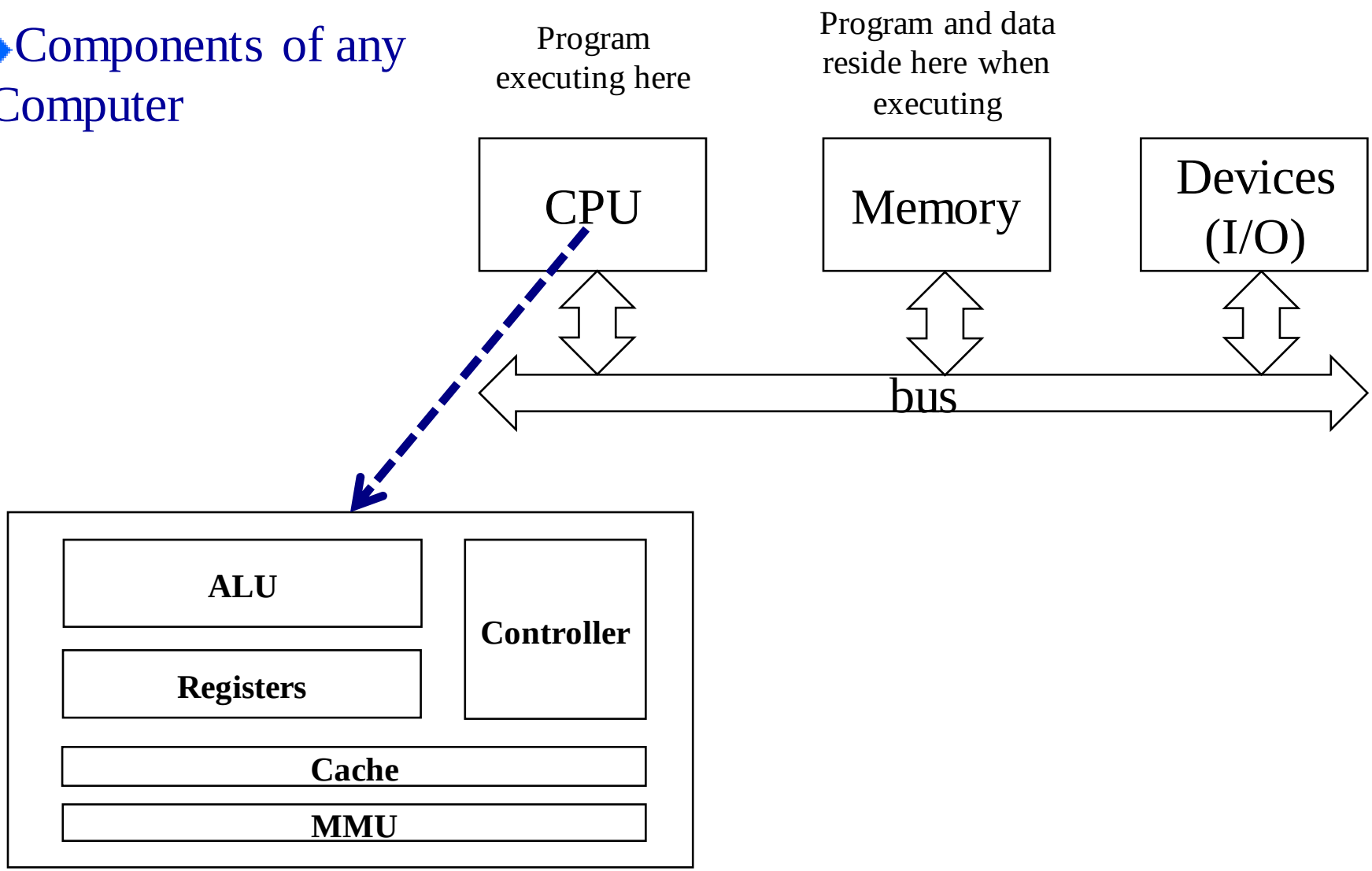
- Interrupt: hardware device requests OS service
  - CPU interrupts current execution and jumps to interrupt handler, and returns when done
  - None of this is visible to user program
- Exceptions: user program acts illegally
  - CPU executes exception handlers
  - May cause abnormal execution flow (such as terminated)
- System calls: user program requests OS service
  - User program execute a trap instruction
  - OS identifies the type of service and parameters, and executes the requested service
  - OS returns to user program when done
  - This appears as a function call to the user program

Outline

- Computer Arch/Memory Hierarchy
- Address Space & Address Generation
- Contiguous Memory Allocation
  - Dynamic Allocation of Partitions
- Non-Contiguous Memory Allocation
  - Segmentation
  - Paging
  - Page Table
    - Translation Look-aside Buffer (TLB)
    - Multi-Level Page Table
    - Inverted Page Table
  - Paged Segmentation Model

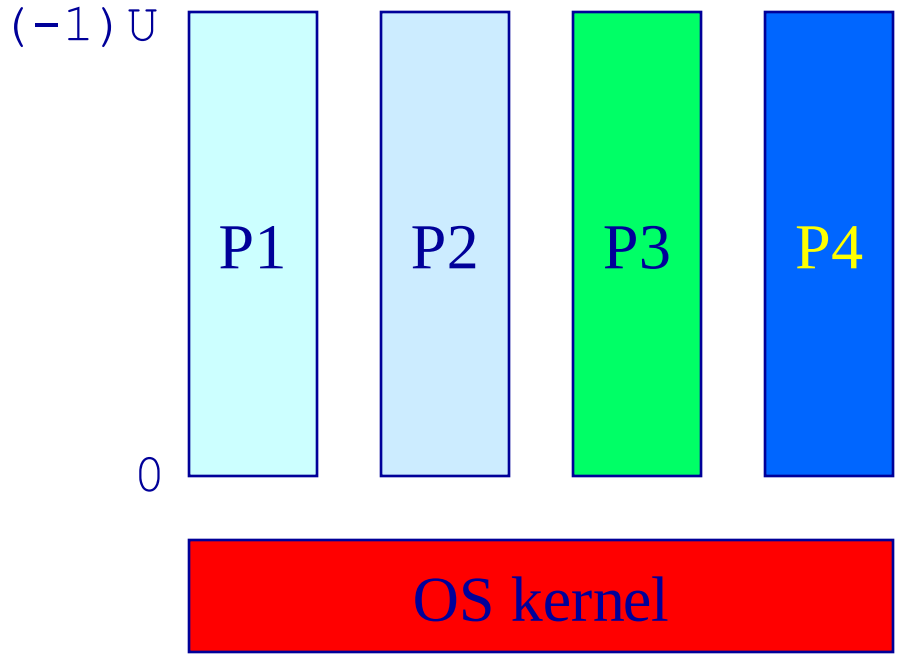
Brief Introduction to Computer Architecture

- Components of any Computer



Intel® 64 and IA-32 Architectures Software Developer Manuals

Modern Memory Management Paradigm



- Abstraction
  - Logical address space
- Protection
  - Isolation
- Programming models
  - Shared memory

Logical (virtual) space

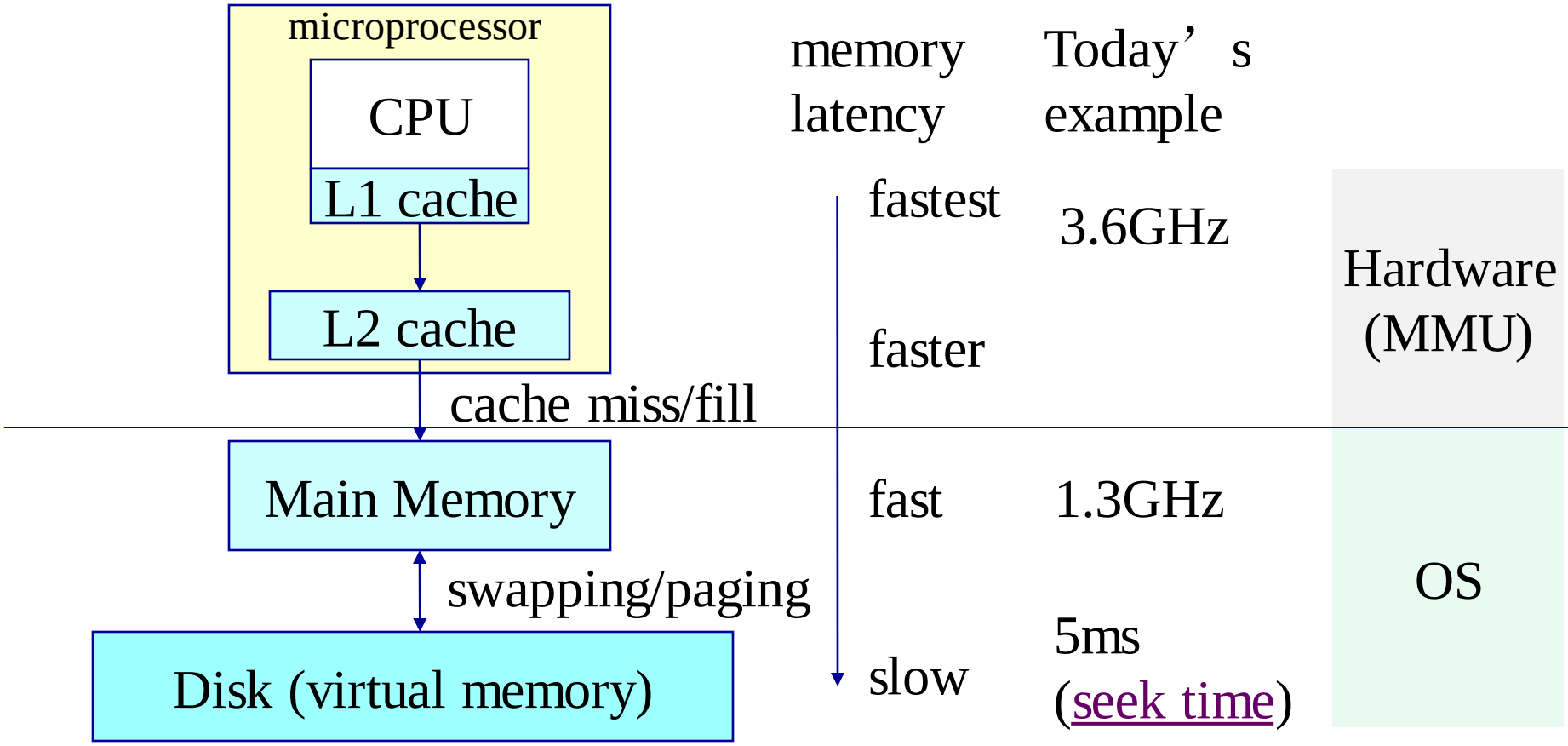
MMU

Physical space

Main memory

Disk (virtual memory)

Memory Hierarchy



OS Memory Models

- Different ways to manage memory in an OS
  - Program relocation
  - Segmentation
  - Paging
  - Virtual memory
  - Mostly (e.g., Linux): demand paging virtual memory
- Implementation highly hardware dependent
  - Must know memory architecture
  - MMU (Memory Management Unit): hardware components responsible for handling memory accesses requested by the CPU

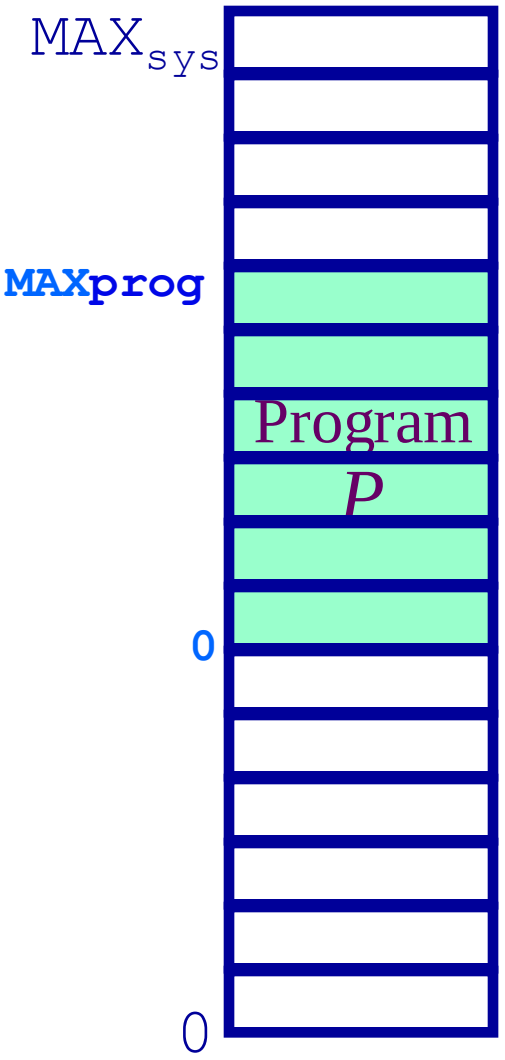
Outline

- Computer Arch/Memory Hierarchy
- Address Space & Address Generation
- Contiguous Memory Allocation
  - Dynamic Allocation of Partitions
- Non-Contiguous Memory Allocation
  - Segmentation
  - Paging
  - Page Table
    - Translation Look-aside Buffer (TLB)
    - Multi-Level Page Table
    - Inverted Page Table
  - Paged Segmentation Model

Address Space & Address Generation

address space

- *Physical address space* — The address space supported by the hardware
  - Starting at address 0, going to address **MAX_{sys}**
- *Logical address space* — A process's view of its own memory
  - Starting at address 0, going to address **MAX_{prog}**



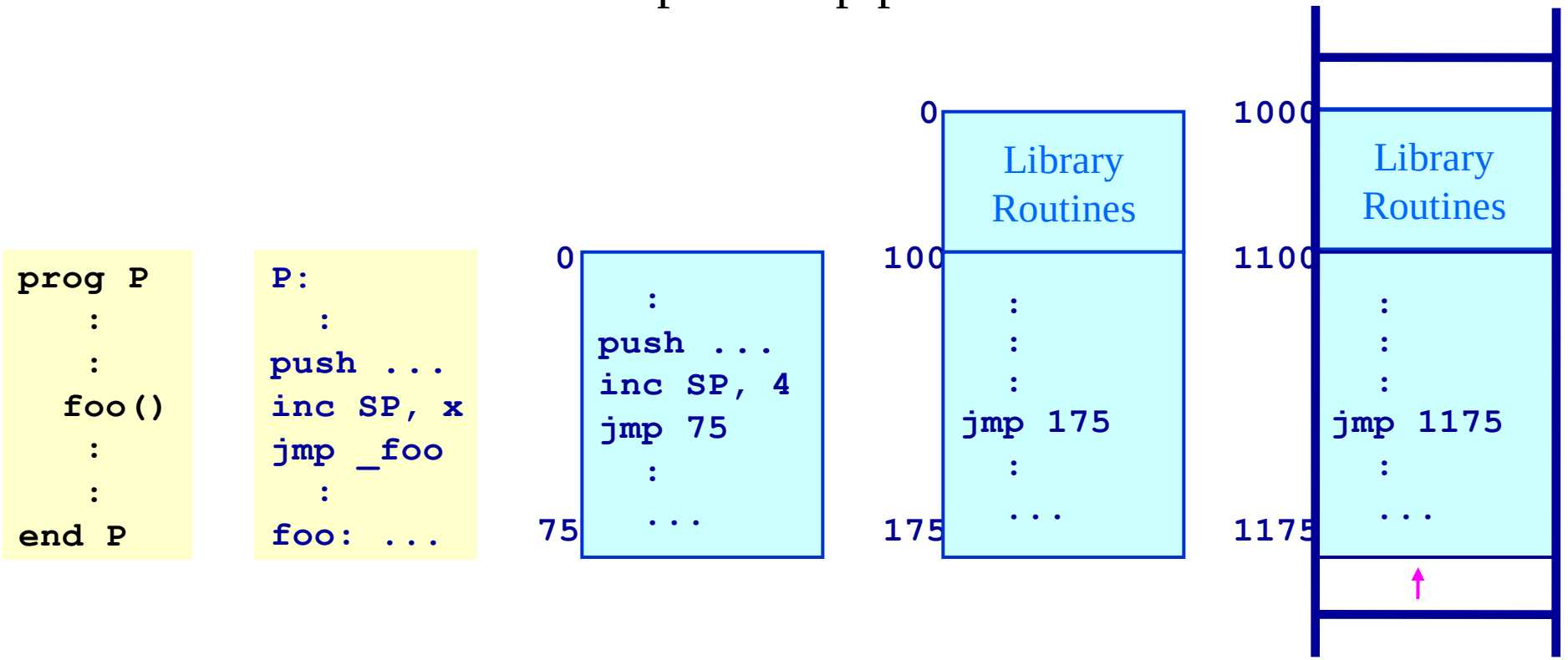
But where do addresses come from?

```
movl %eax, $0xfffa620e
```

Address Space & Address Generation

Address Generation

- The compilation pipeline



Compilation Assembly Linking Loading (program relocation)

Address Space & Address Generation

Address Generation Time

- Compile time
  - If memory location known a priori
  - Must recompile code if starting location changes
- Load time
  - Compiler must generate *relocatable* code if memory location is not known at compile time
  - Absolute addresses generated at load time
- Execution time
  - The process can be moved during its execution
  - Need hardware support for address translation

Outline

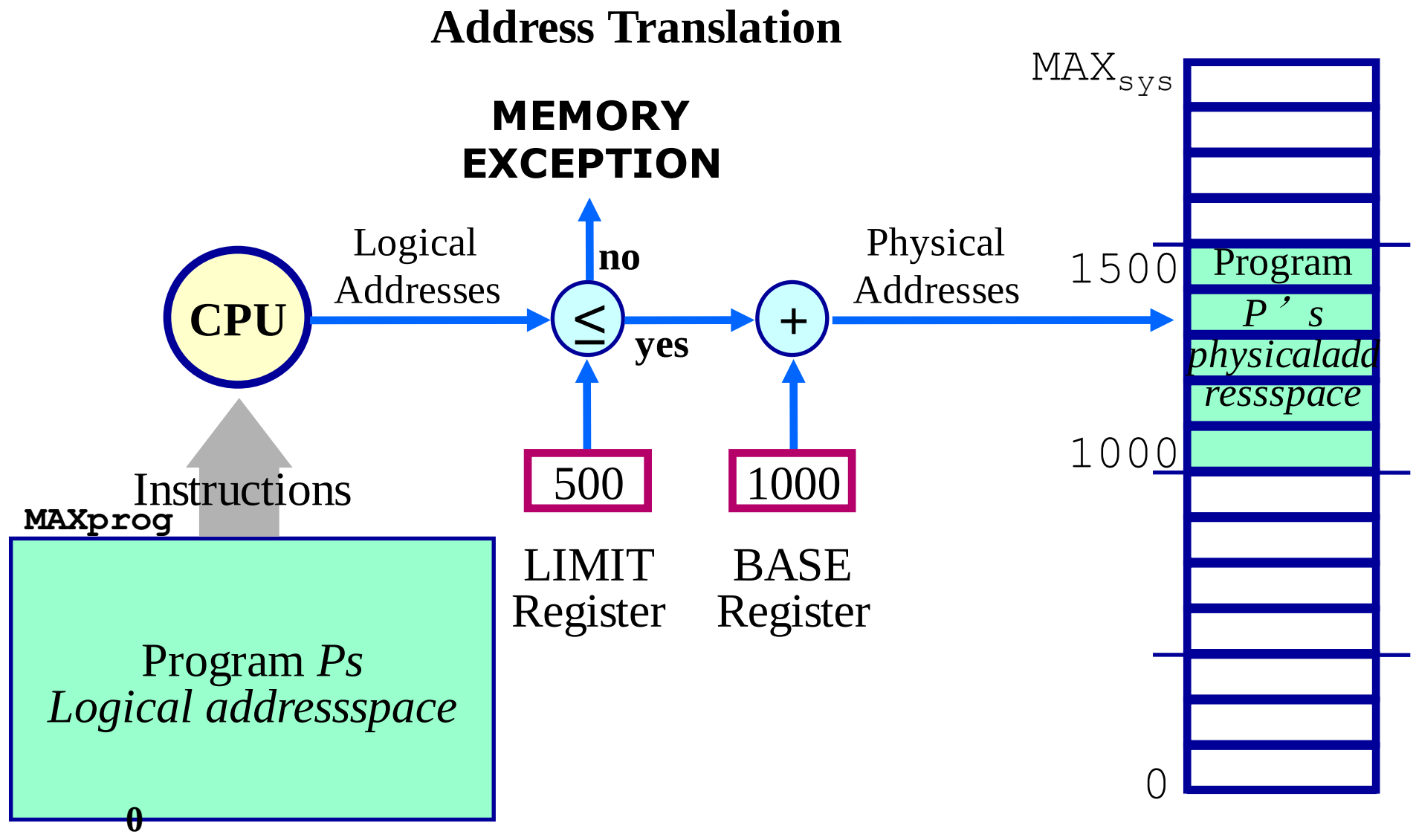
- Computer Arch/Memory Hierarchy
- Address Space & Address Generation
- Contiguous Memory Allocation
  - Dynamic Allocation of Partitions
- Non-Contiguous Memory Allocation
  - Segmentation
  - Paging
  - Page Table
    - Translation Look-aside Buffer (TLB)
    - Multi-Level Page Table
    - Inverted Page Table
  - Paged Segmentation Model

Address Space & Address Generation

Program Relocation

- Relocate logical addresses to physical at run time
  - While we are relocating, also bounds check addresses for safety.
- Require hardware support (MMU)
- Basic component
  - Address translation with two registers: BASE and LIMIT

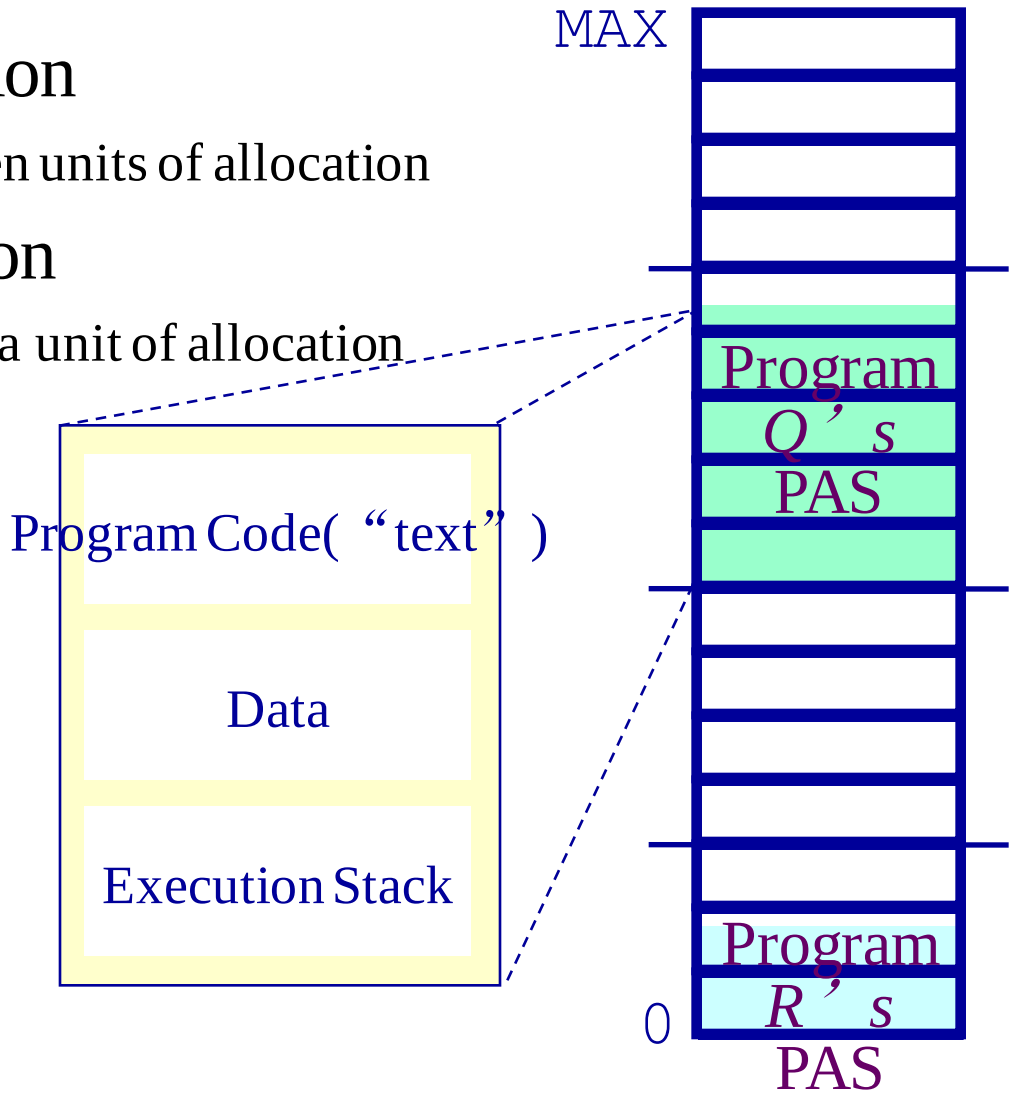
Contiguous Memory Allocation



Contiguous Memory Allocation

The Fragmentation Problem

- Free memory cannot be utilized
- External fragmentation
  - Unused memory between units of allocation
- Internal fragmentation
  - Unused memory within a unit of allocation



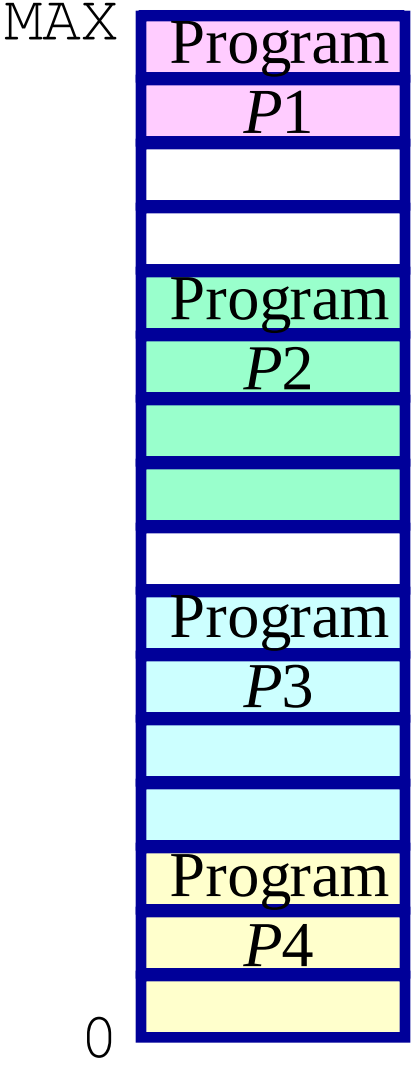
Contiguous Memory Allocation

Dynamic Allocation of Partitions

- Simple memory management approach:
  - Allocate a partition when a process is admitted into the system
  - Allocate a contiguous memory partition to the process

OS keeps track of...
Full-blocks
Empty-blocks (“holes”)

Allocation strategies
First-fit
Best-fit
Worst-fit

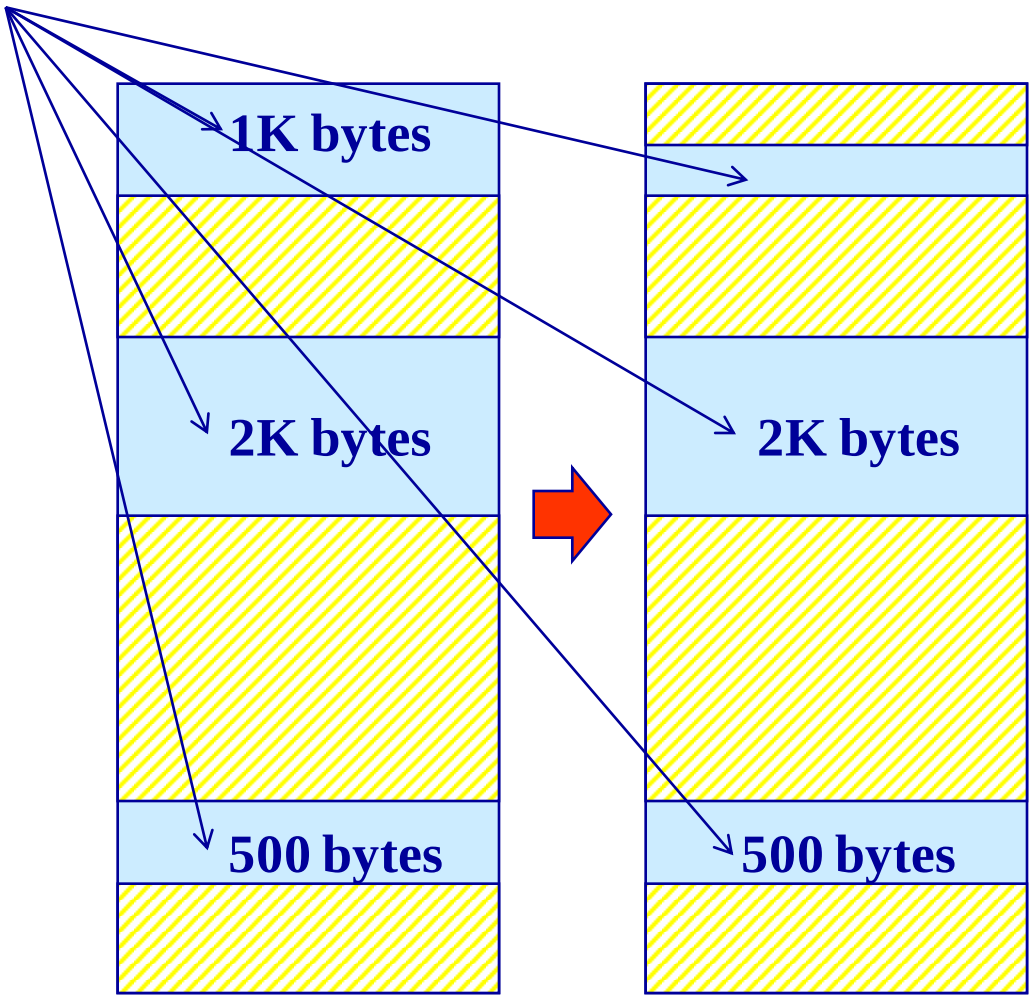


Contiguous Memory Allocation

First Fit Allocation

To allocate *n* bytes, use the *first* available free block such that the block size is larger than *n*.

To allocate 400 bytes, we use the 1st free block available



Contiguous Memory Allocation

Rationale & Implementation

- Simplicity of implementation
- Requires:
  - Free block list sorted by address
  - Allocation requires a search for a suitable partition
  - De-allocation requires a check to see if the freed partition could be merged with adjacent free partitions (if any)

Advantages

- Simple
- Tends to produce larger free blocks toward the end of the address space

Disadvantages

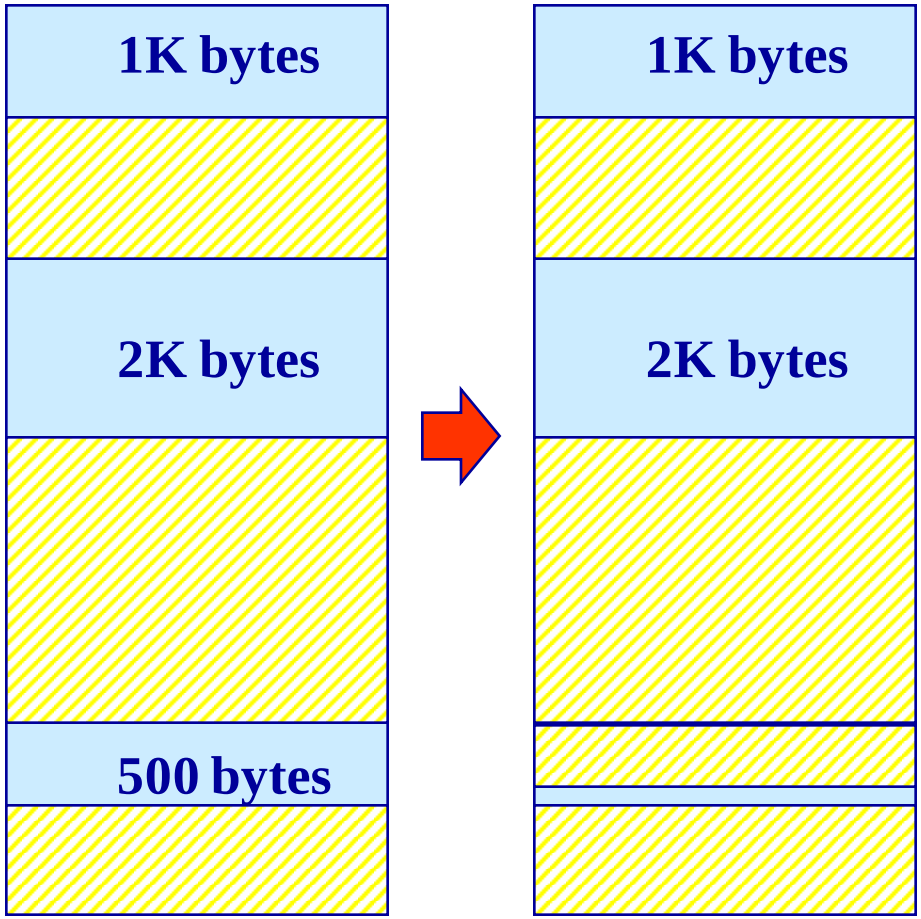
- External fragmentation
- Uncertainty

Contiguous Memory Allocation

Best Fit Allocation

To allocate *n* bytes, use the *smallest* available free block such that the block size is larger than *n*.

To allocate 400 bytes, we use the 3rd free block available (smallest)



Contiguous Memory Allocation

Rationale & Implementation

- To avoid fragmenting big free blocks
- To minimize the size of external fragments produced
- Requires:
  - Free block list sorted by size
  - Allocation requires search for a suitable partition
  - De-allocation requires search + merge with adjacent free partitions, if any

Advantages

- Works well when most allocations are of small size
- Relatively simple

Disadvantages

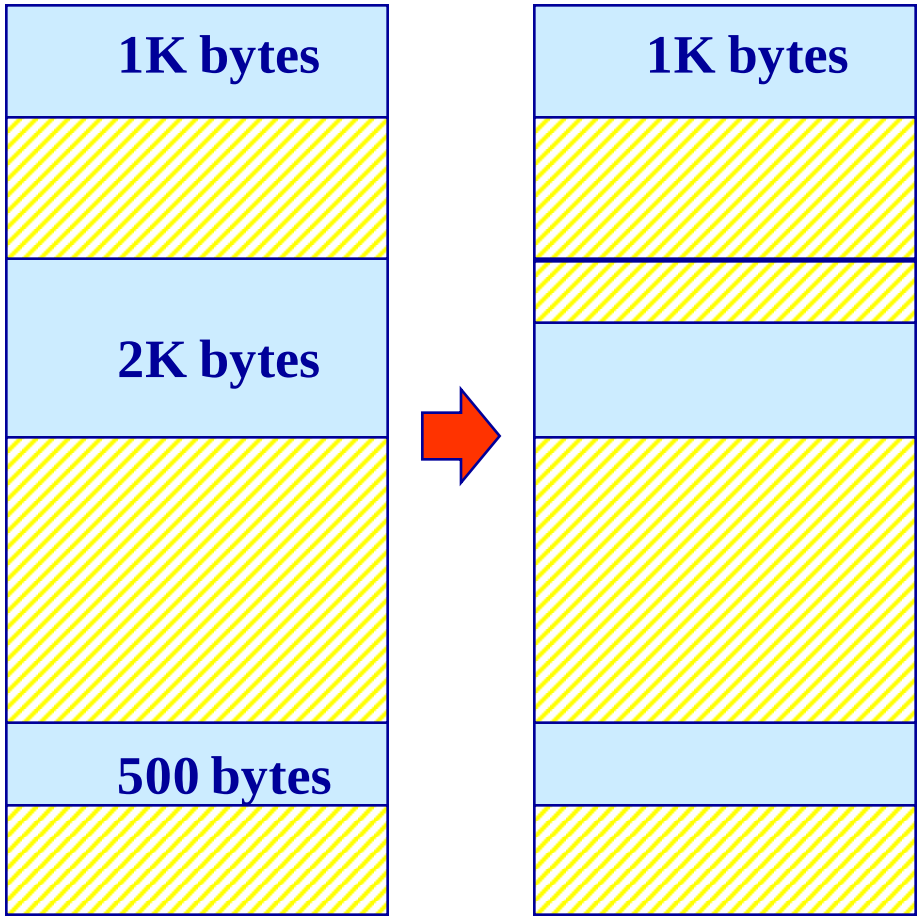
- External fragmentation
- Slow de-allocation
- Tends to produce many useless tiny fragments (not really great)

Contiguous Memory Allocation

Worst Fit Allocation

To allocate *n* bytes, use the *largest* available free block such that the block size is larger than *n*.

To allocate 400 bytes, we use the 2nd free block available (largest)



Contiguous Memory Allocation

Rationale & Implementation

- To avoid having too many tiny fragments
- Requires:
  - Free block list sorted by size
  - Allocation is fast (get the largest partition)
  - De-allocation requires merge with adjacent free partitions, if any, and then adjusting the free block list

Advantages

- Works best if allocations are of medium sizes

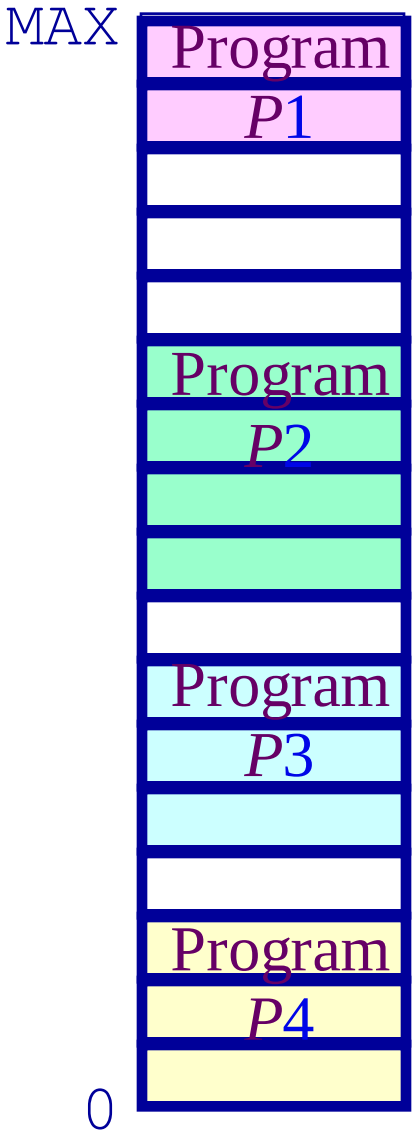
Disadvantages

- Slow de-allocation
- External fragmentation
- Tends to break large free blocks such that large partitions cannot be allocated

Contiguous Memory Allocation

De-fragmentation by Compaction

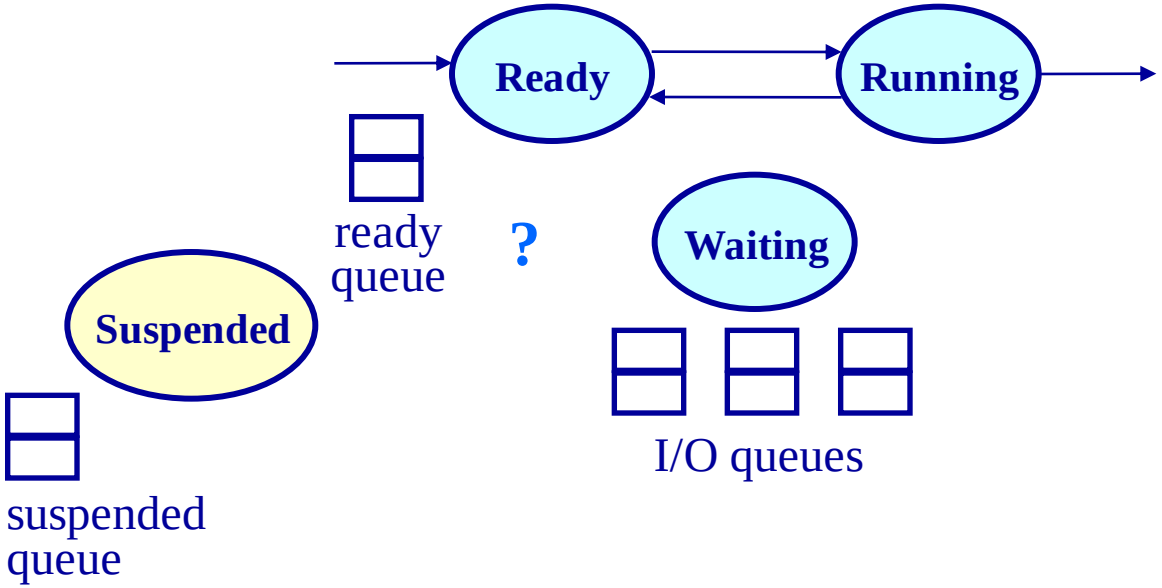
- Relocate programs to coalesce holes
- Require all programs to be dynamically relocatable
- Issues
  - When to relocate?
  - Overhead



Contiguous Memory Allocation

De-fragmentation by Swapping

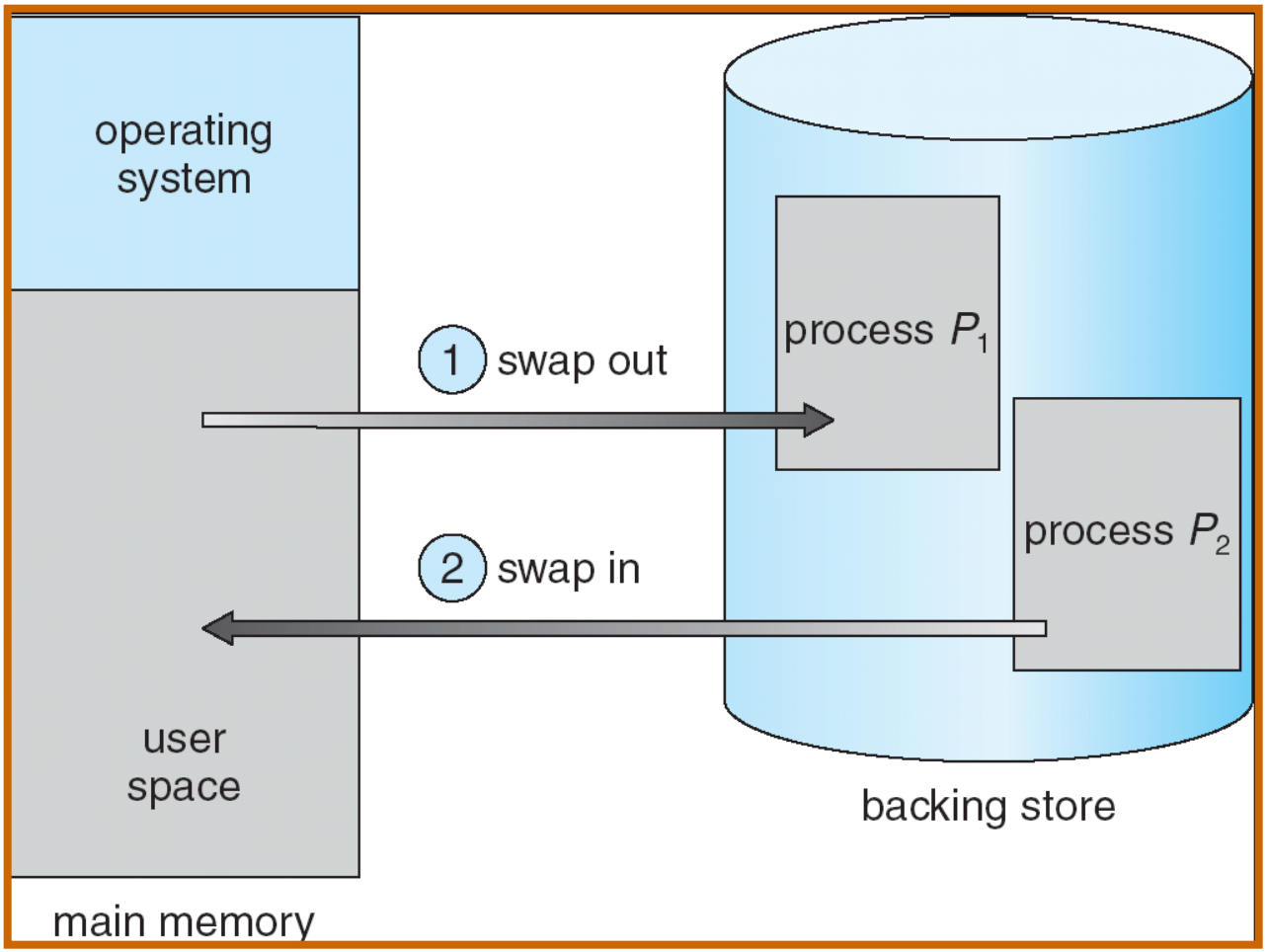
- Preempt processes & reclaim their memory



- Issue: which process(es) to swap?

Contiguous Memory Allocation

Schematic View of Swapping



Outline

- Computer Arch/Memory Hierarchy
- Address Space & Address Generation
- Contiguous Memory Allocation
  - Dynamic Allocation of Partitions
- Non-Contiguous Memory Allocation
  - Segmentation
  - Paging
  - Page Table
    - Translation Look-aside Buffer (TLB)
    - Multi-Level Page Table
    - Inverted Page Table
  - Paged Segmentation Model

Non-contiguous Allocation : Segmentation

- Previously,
  - Physical memory allocated to a process is contiguous
  - Poor memory utilization
  - Suffers from external fragmentation
- Noncontiguous allocation
  - Physical address space of a process is noncontiguous
  - Better memory utilization and management
  - Allow sharing of common blocks (code, data, library, etc.)
  - Support dynamic loading and dynamic linking
- Two schemes: segmentation and paging

Non-contiguous Allocation : Segmentation

Dynamic Loading

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- Useful when large amounts of code are needed to handle infrequently occurring cases
- Most OS allows user programs to do dynamic loading of components (relocatable object code)
- Some OS supports loadable kernel modules

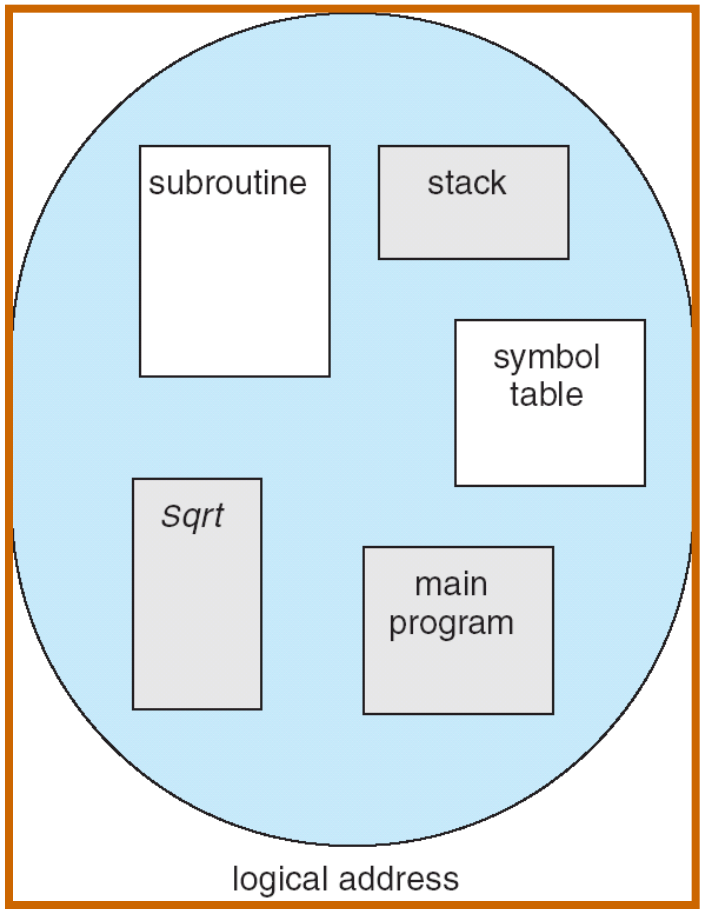
Non-contiguous Allocation : Segmentation

Dynamic Linking

- Linking postponed until execution time
  - Small piece of code, *stub*, used to locate the appropriate memory-resident library routine
  - Stub replaces itself with the address of the routine, and executes the routine
  - Operating system needed to check if routine is in processes' memory address
- Dynamic linking is particularly useful for libraries
  - Better known as *shared libraries*
- Dynamic linking in ucore

Non-contiguous Allocation : Segmentation

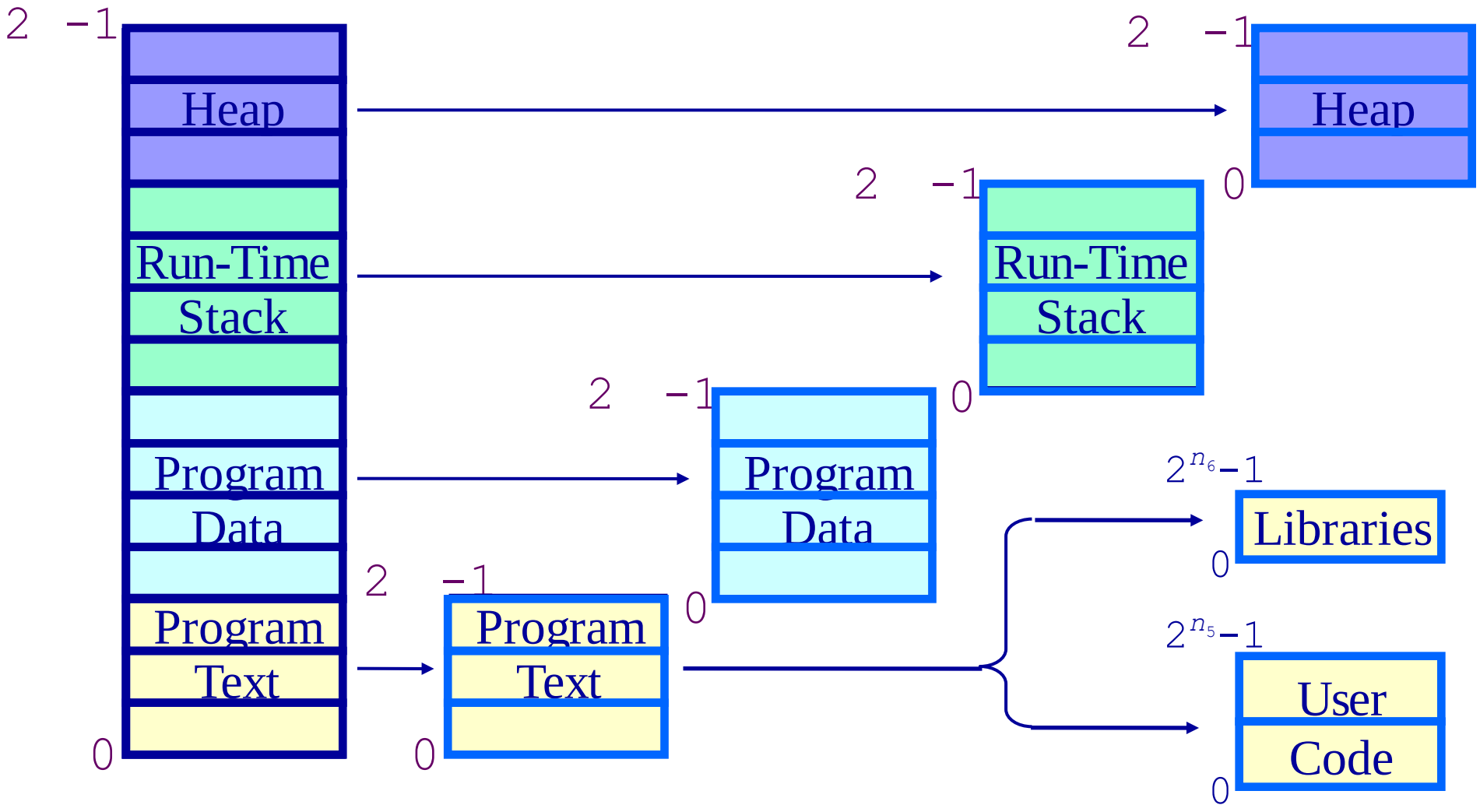
Segmentation



- A program is a collection of segments, such as
  - Main program
  - Subroutines
  - Stack
  - Symbols
  - Data
  - Common libraries
  - Common blocks
- Purpose: enable finer grain isolation and sharing

Non-contiguous Allocation : Segmentation

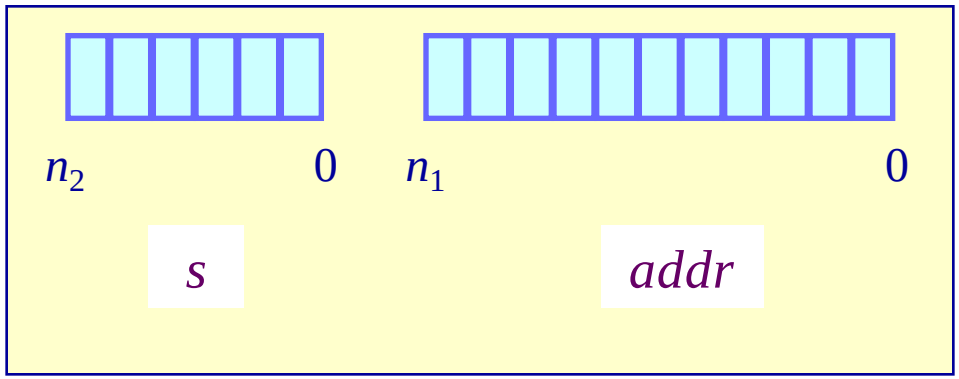
Separating into Multiple Address Spaces



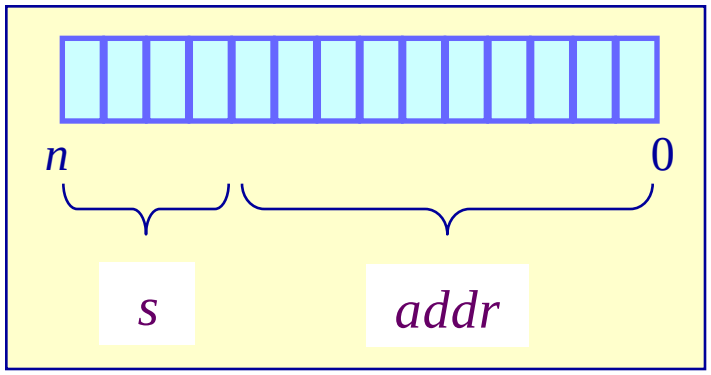
Non-contiguous Allocation : Segmentation

Segmentation Schemes

- New concept: A segment — a memory "object"
  - A logical address space
- A process now addresses objects —a pair (s, addr)
  - s — segment number
  - addr — an offset within an object



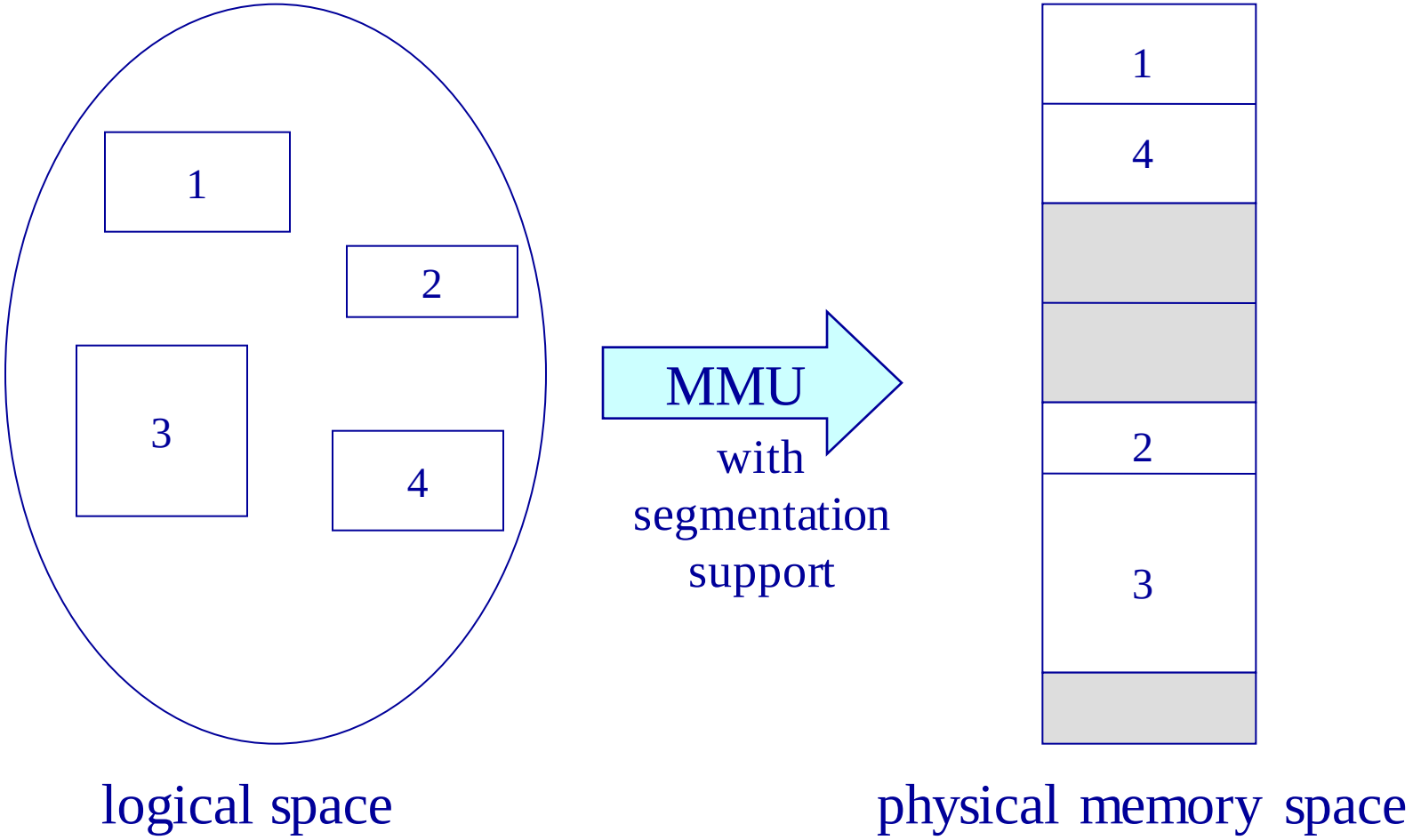
Segment + Address register scheme



Single address scheme

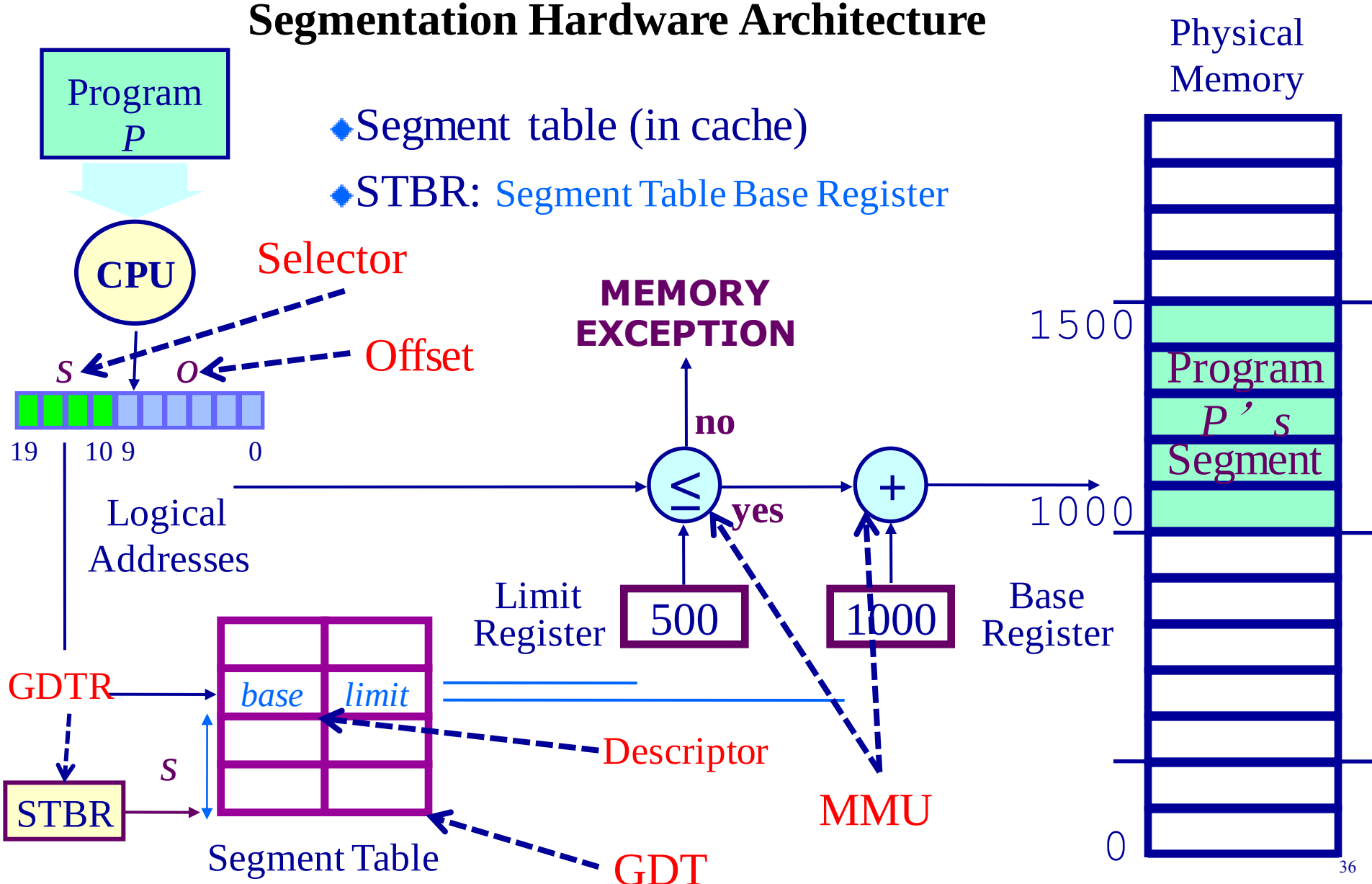
Non-contiguous Allocation: Segmentation

Logical View of Segmentation



Non-contiguous Allocation: Segmentation

Segmentation Hardware Architecture



Outline

- Computer Arch/Memory Hierarchy
- Address Space & Address Generation
- Contiguous Memory Allocation
  - Dynamic Allocation of Partitions
- Non-Contiguous Memory Allocation
  - Segmentation
  - Paging
  - Page Table
    - Translation Look-aside Buffer (TLB)
    - Multi-Level Page Table
    - Inverted Page Table
  - Paged Segmentation Model

Non-contiguous Allocation : Paging

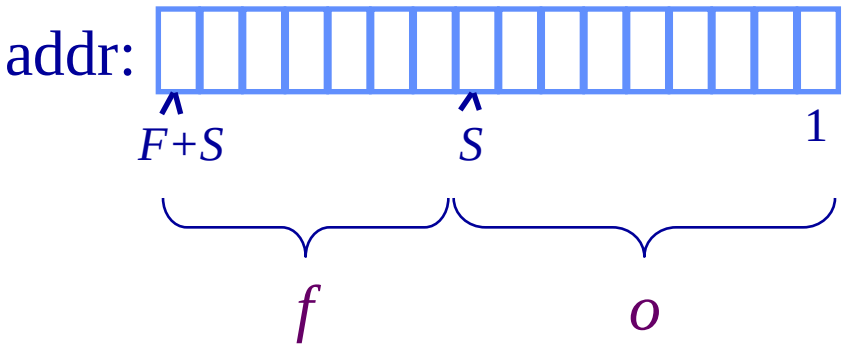
- Divide physical memory into fixed-sized frames
  - Size is power of 2, e.g., 512, 4096, 8192
- Divide logical address space into same size pages
- To run a program of size n pages, find n free frames and load program
- Set up a page table to translate logical to physical addresses (pages to frames)
- Frame/page: basic units of memory allocation
  - OS keep track of all free frames
  - Same-sized frame eliminates external fragmentation

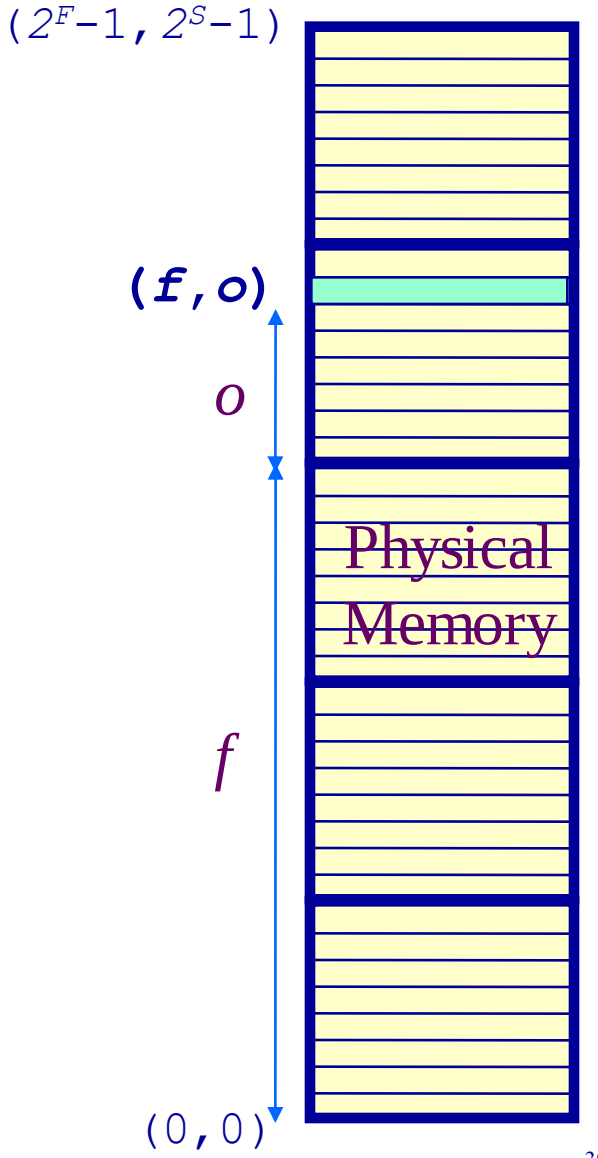
Non-contiguous Allocation : Paging

Frames

- Physical memory partitioned into equal sized *frames*

A memory address is a pair (*f*, *o*)
f — frame number (total 2^F frames)
o — frame offset (2^S bytes/frames)
Physical address = $2^S \times f + o$

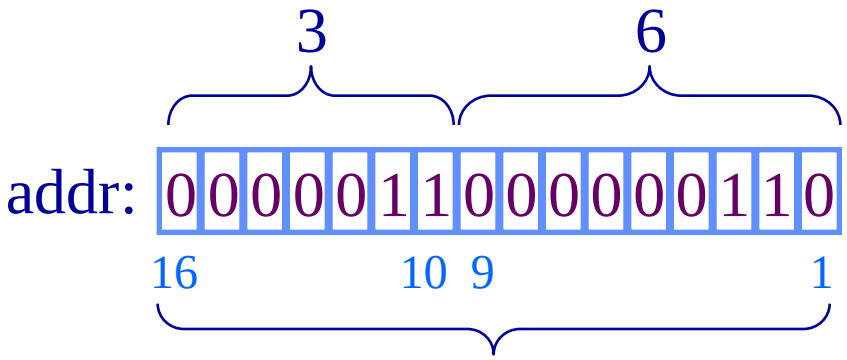




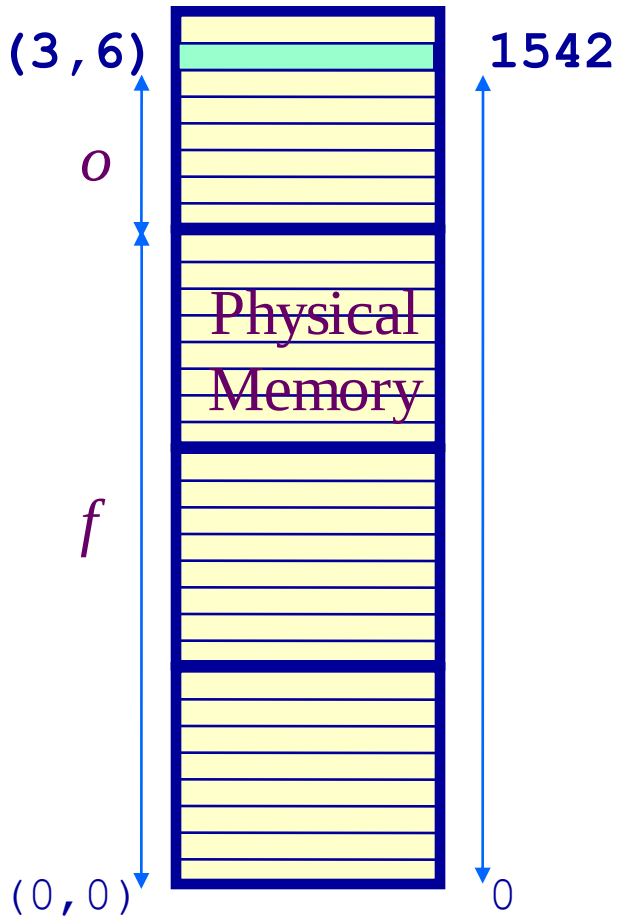
Non-contiguous Allocation : Paging

Frame Example

- Example: A 16-bit address space with 9-bit (512 byte) page frames
  - Addressing location (3, 6) = 1542



3*512+6=1542



Non-contiguous Allocation : Paging

Pages

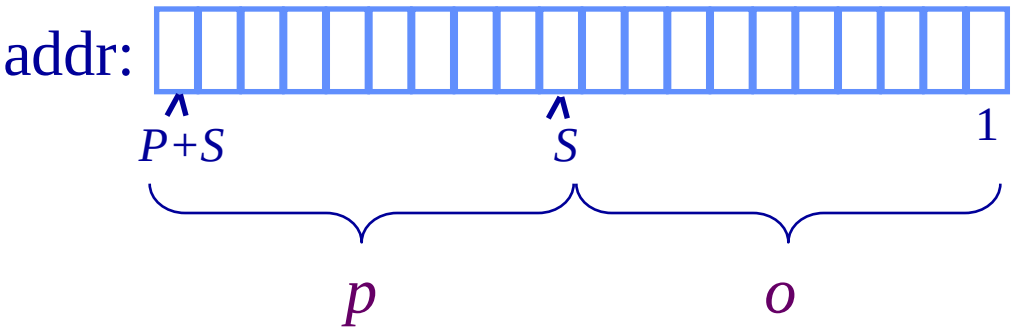
- A process's logical address space is partitioned into equal sized *pages*
  - $|page| = |frame|$

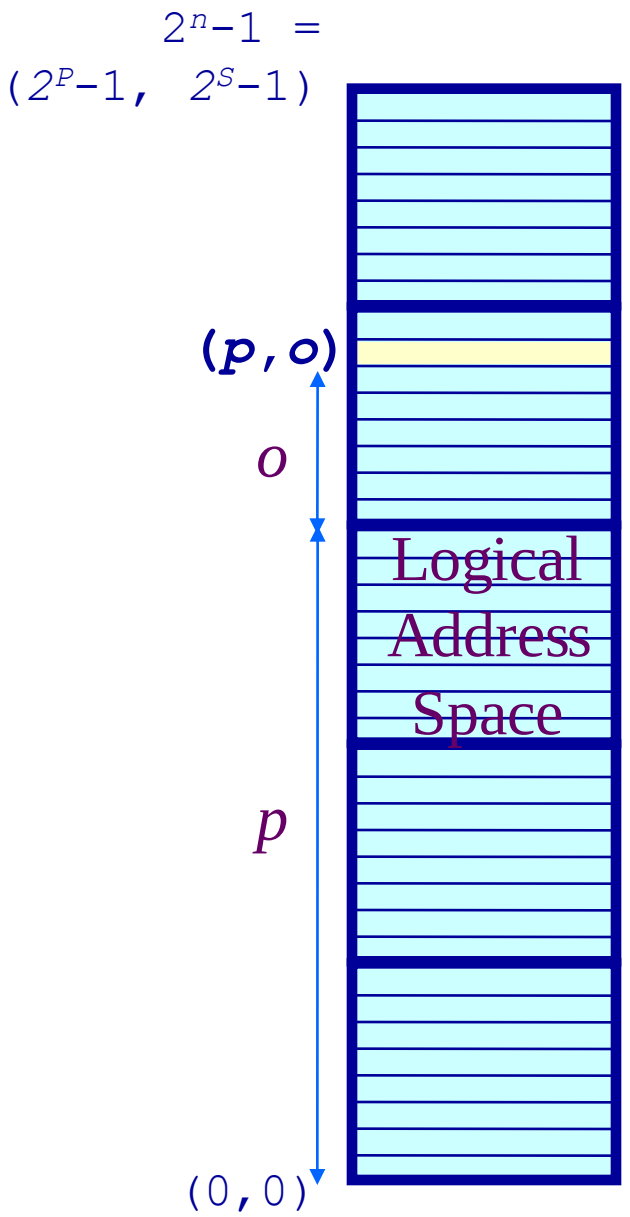
A logical address is a pair (p, o)

p — page number (2^P pages)

o — page offset (2^S bytes/pages)

Virtual address = $2^S \times p + o$

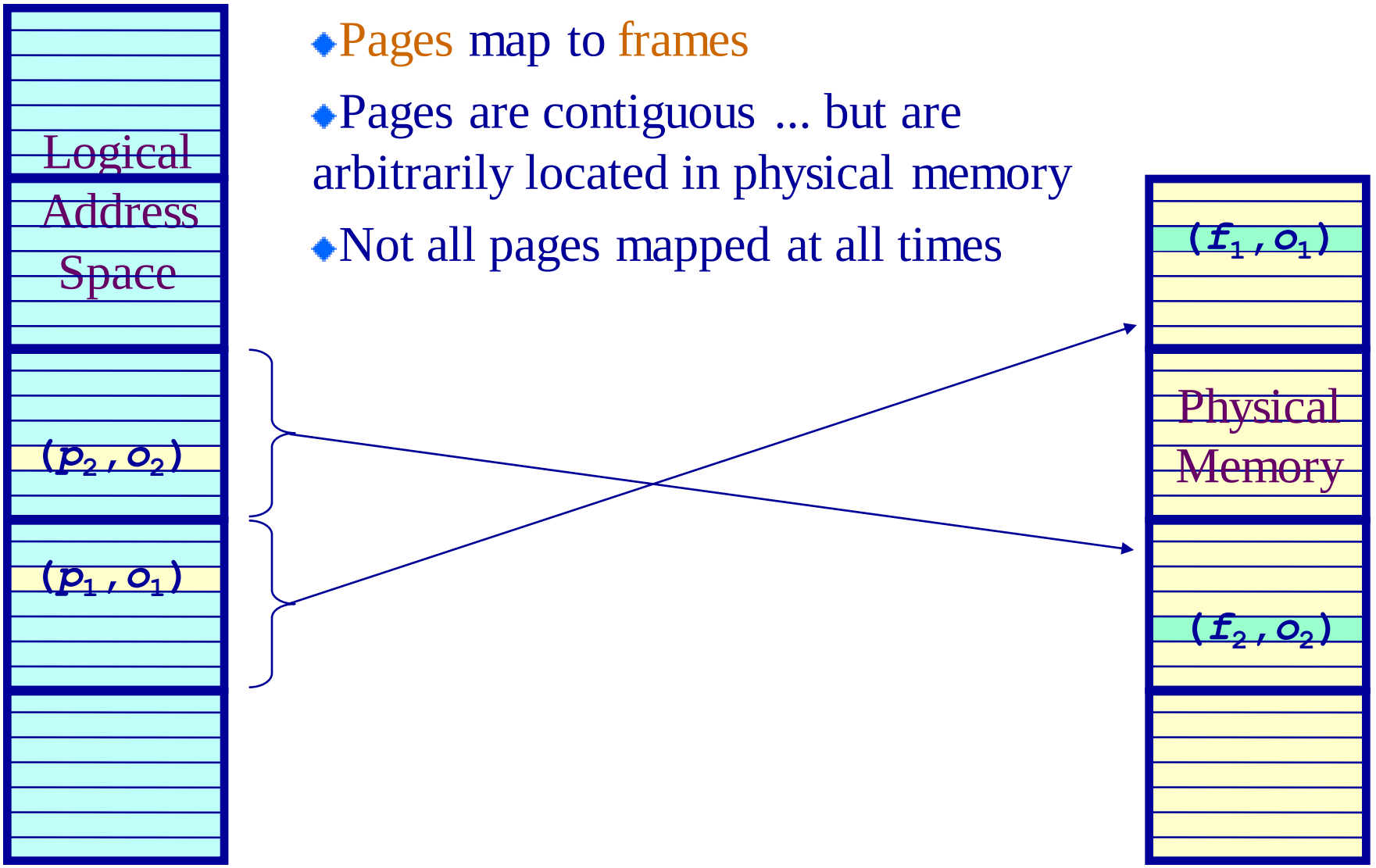




Non-contiguous Allocation : Paging

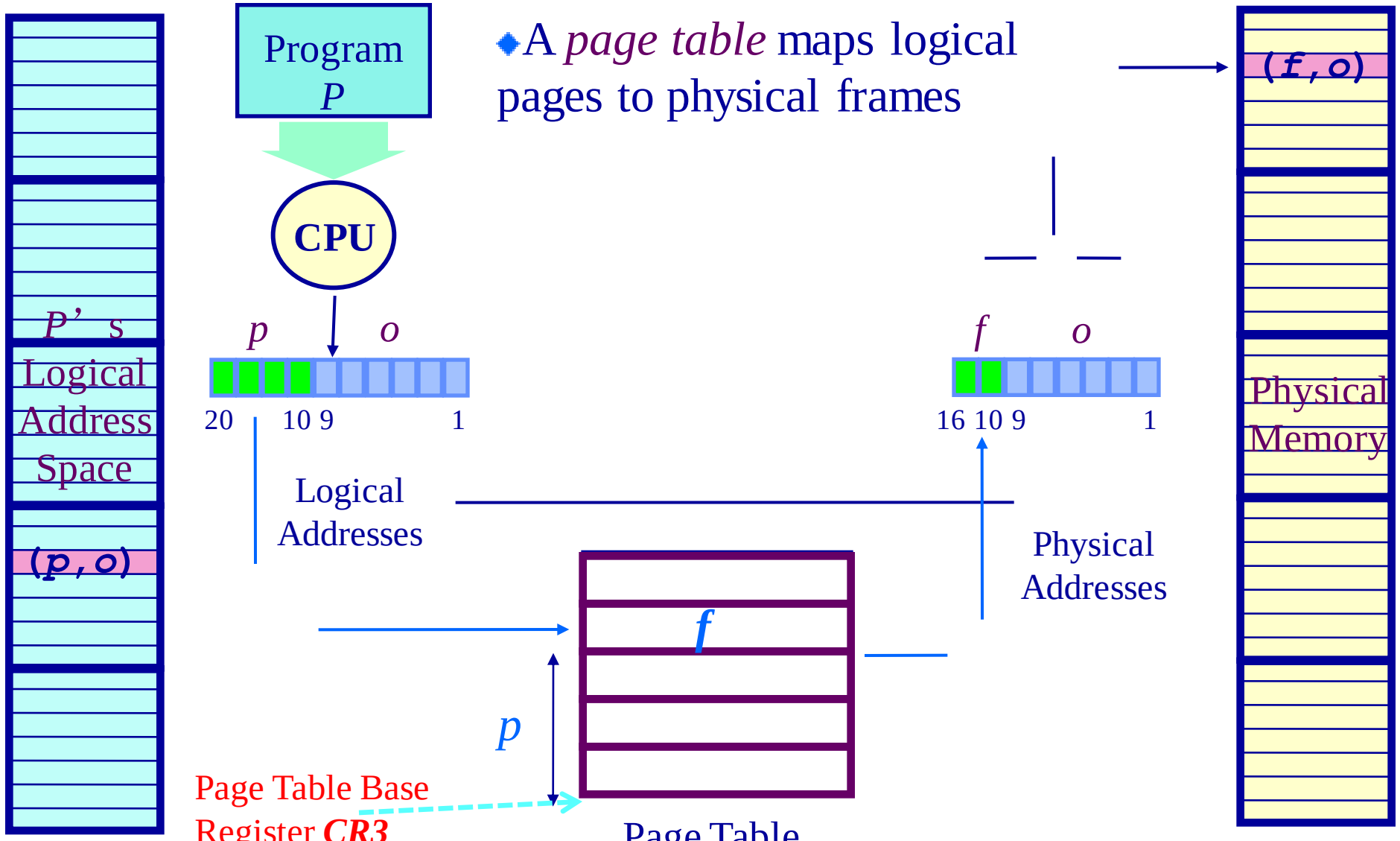
Paging Model

- Pages map to frames
- Pages are contiguous ... but are arbitrarily located in physical memory
- Not all pages mapped at all times



Non-contiguous Allocation : Paging

Paging Hardware Architecture



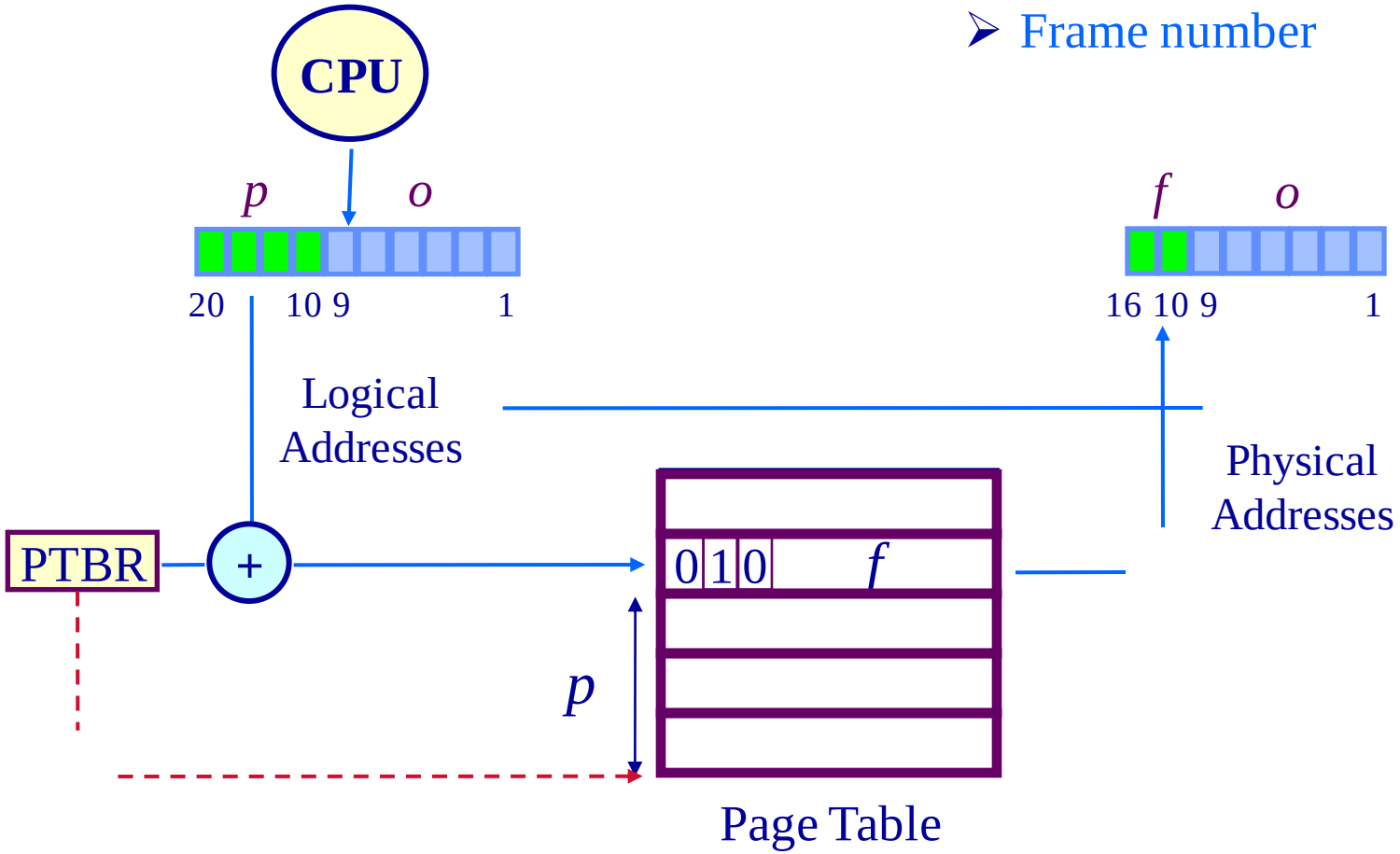
Outline

- Computer Arch/Memory Hierarchy
- Address Space & Address Generation
- Contiguous Memory Allocation
  - Dynamic Allocation of Partitions
- Non-Contiguous Memory Allocation
  - Segmentation
  - Paging
  - Page Table
    - Translation Look-aside Buffer (TLB)
    - Multi-Level Page Table
    - Inverted Page Table
  - Paged Segmentation Model

Non-contiguous Allocation : Page Table

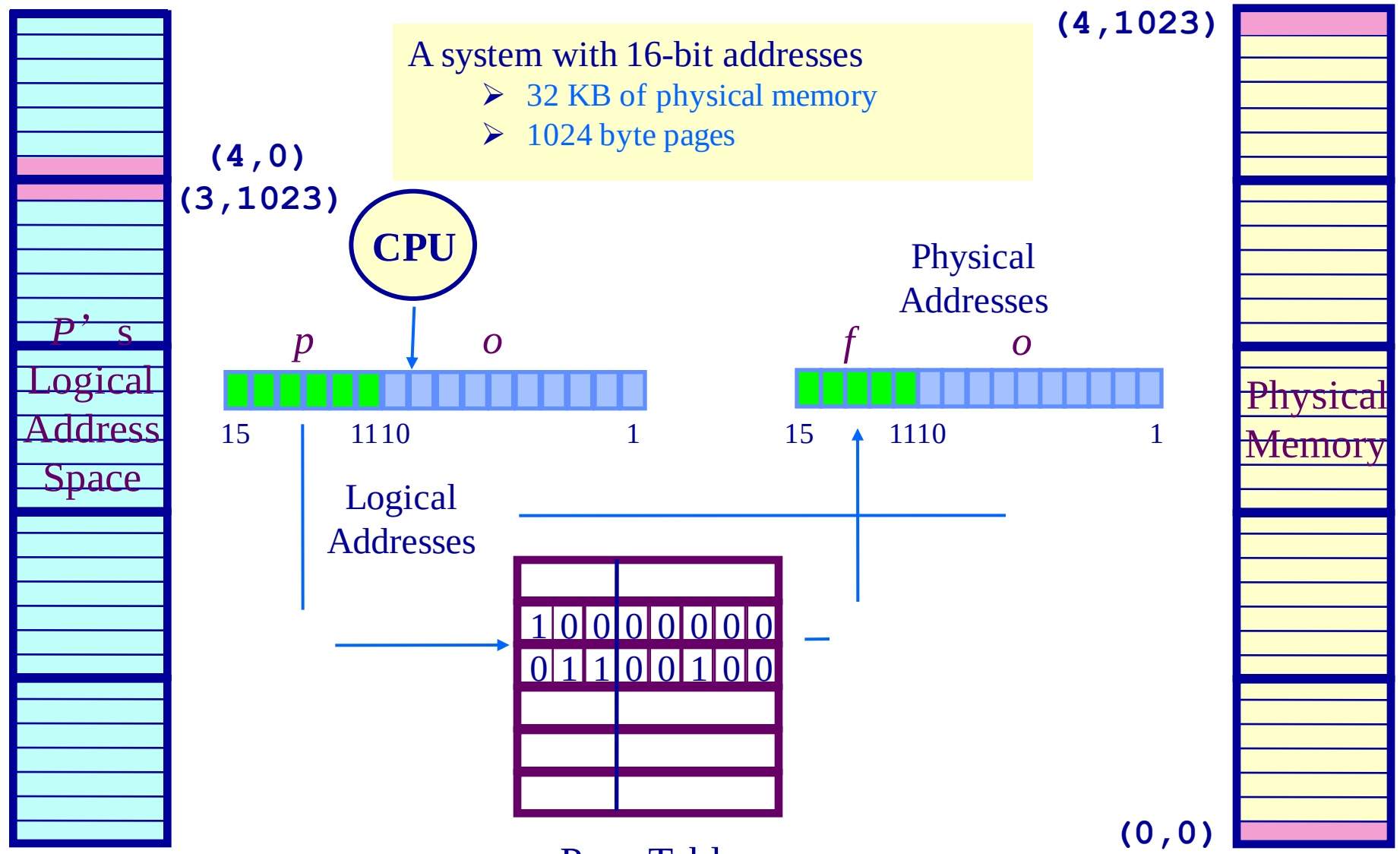
Page Table Structure

- One table per process
  - Part of process’ s state
  - PTBR: Page Table Base Register
- Contents:
  - Flags — dirty bit, resident bit, clock/reference bit
  - Frame number



Non-contiguous Allocation : Page Table

Example Address Translation



Non-contiguous Allocation : Page Table

Paging Performance Issue

- Problem — Requires 2 memory references!
  - One access to get the page table entry
  - One access to get the data
- Page table can be very large
  - For a machine with 64-bit addresses and 1024 byte pages, what is the size of a page table?
- What to do? Hint: most computing problems are solved by some form of…
  - Caching
  - Indirection

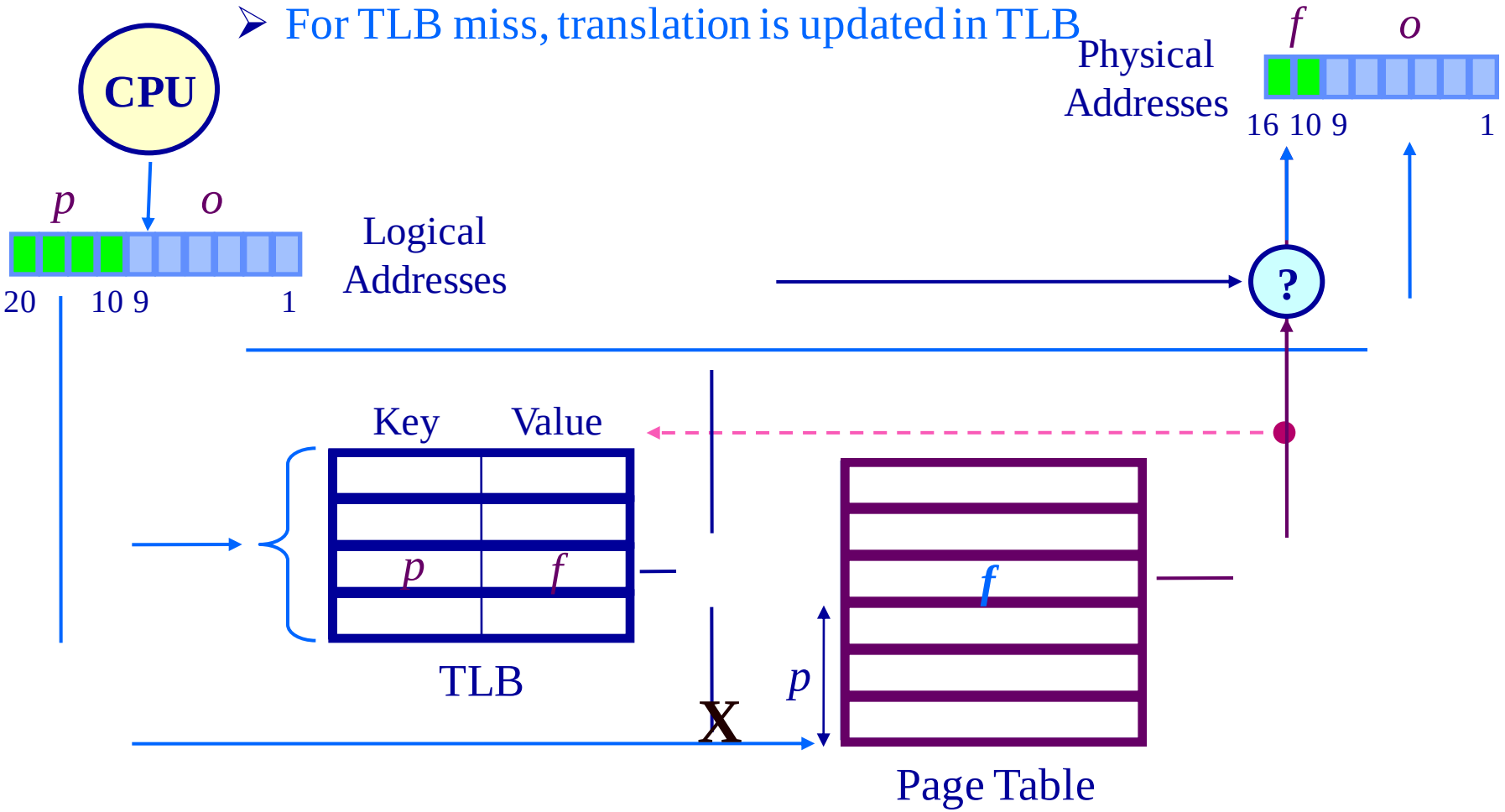
Outline

- Computer Arch/Memory Hierarchy
- Address Space & Address Generation
- Contiguous Memory Allocation
  - Dynamic Allocation of Partitions
- Non-Contiguous Memory Allocation
  - Segmentation
  - Paging
  - Page Table
    - Translation Look-aside Buffer (TLB)
    - Multi-Level Page Table
    - Inverted Page Table
  - Paged Segmentation Model

Non-contiguous Allocation : Page Table

Translation Look-aside Buffer (TLB)

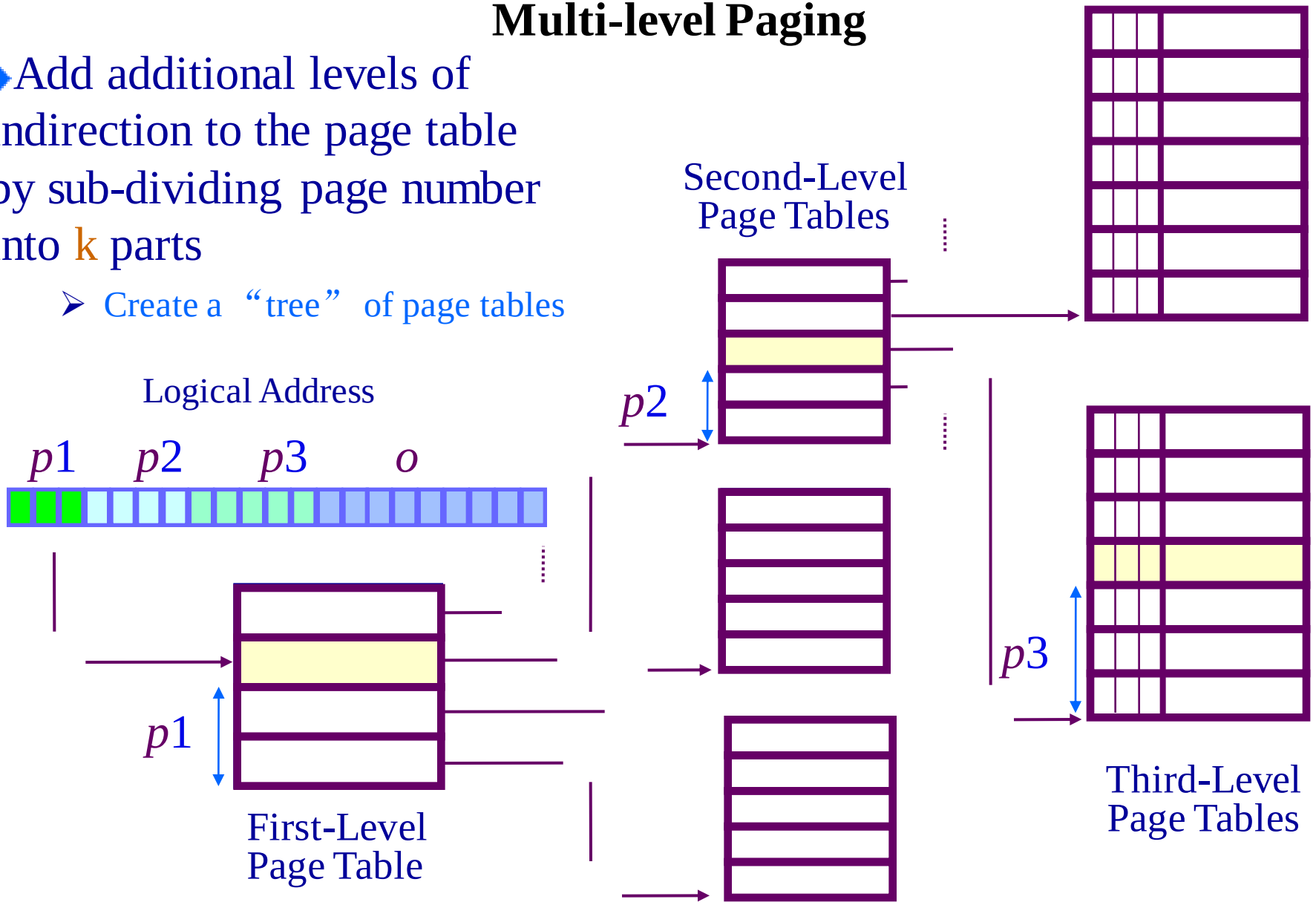
- Cache recently accessed page-to-frame translations
  - TLB implemented in associative memory for fast access
  - For TLB hit, physical page number obtained in 1 cycle
  - For TLB miss, translation is updated in TLB



Non-contiguous Allocation : Page Table

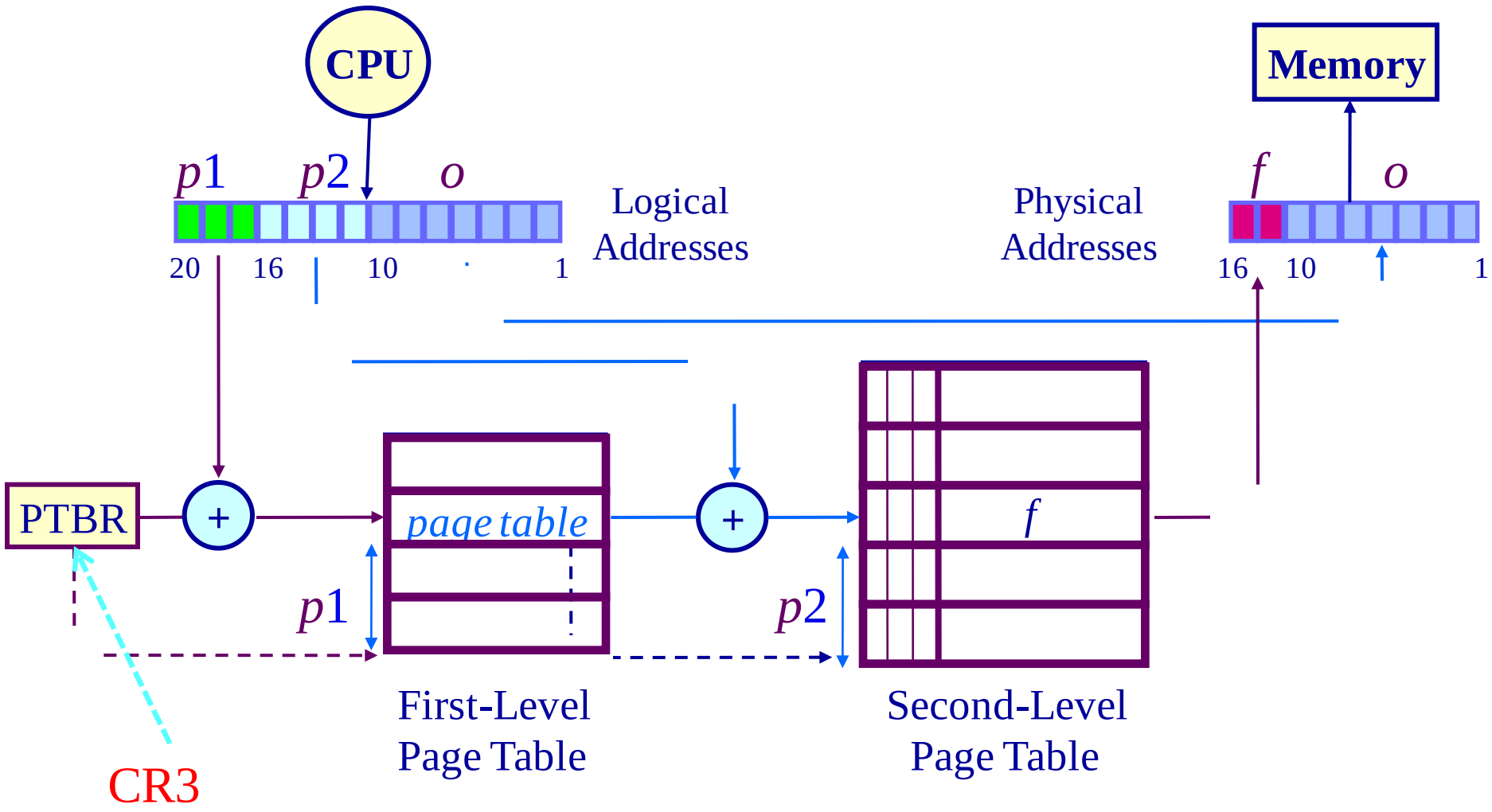
Multi-level Paging

- Add additional levels of indirection to the page table by sub-dividing page number into k parts
  - Create a "tree" of page tables



Non-contiguous Allocation : Page Table

Example: Two-level Paging



Non-contiguous Allocation : Page Table

The Problem of Large Address Spaces

- With large address spaces (64-bits) forward mapped page tables become cumbersome.
  - E.g. 5 levels of tables.

- Instead of making tables proportional to size of logical address space, make them proportional to the size of physical address space.
  - Logical (virtual) address space is growing faster than physical.

Non-contiguous Allocation : Page Table

Using Page Registers (aka Inverted Page Tables)

- Each frame is associated with a register containing
  - Residence bit: whether or not the frame is occupied
  - Occupier: page number of the page occupying frame
  - Protection bits

- Page registers: an example
  - Physical memory size: 16 MB
  - Page size: 4096 bytes
  - Number of frames: 4096
  - Space used for page registers (assuming 8 bytes/register): 32 Kbytes
  - Percentage overhead introduced by page registers: 0.2%
  - Size of virtual memory: irrelevant

Non-contiguous Allocation : Page Table

Page Registers Tradeoffs

- Advantages:
  - Size of translation table occupies a very small fraction of physical memory
  - Size of translation table is independent of logical address space size

- Disadvantages:
  - We have reverse of the information that we need….
  - How do we perform translation ?
  - Search the translation table for the desired page number

Non-contiguous Allocation : Page Table

Searching for a Page in Inverted Page Tables

- If the number of frames is small, the page registers can be placed in an associative memory
- Logical page number looked up in associative memory
  - Hit: frame number is extracted
  - Miss: results in page fault
- Limitations:
  - Large associative memories are expensive
    - Difficult to make large and accessible in a single cycle.
    - They consume a lot of power

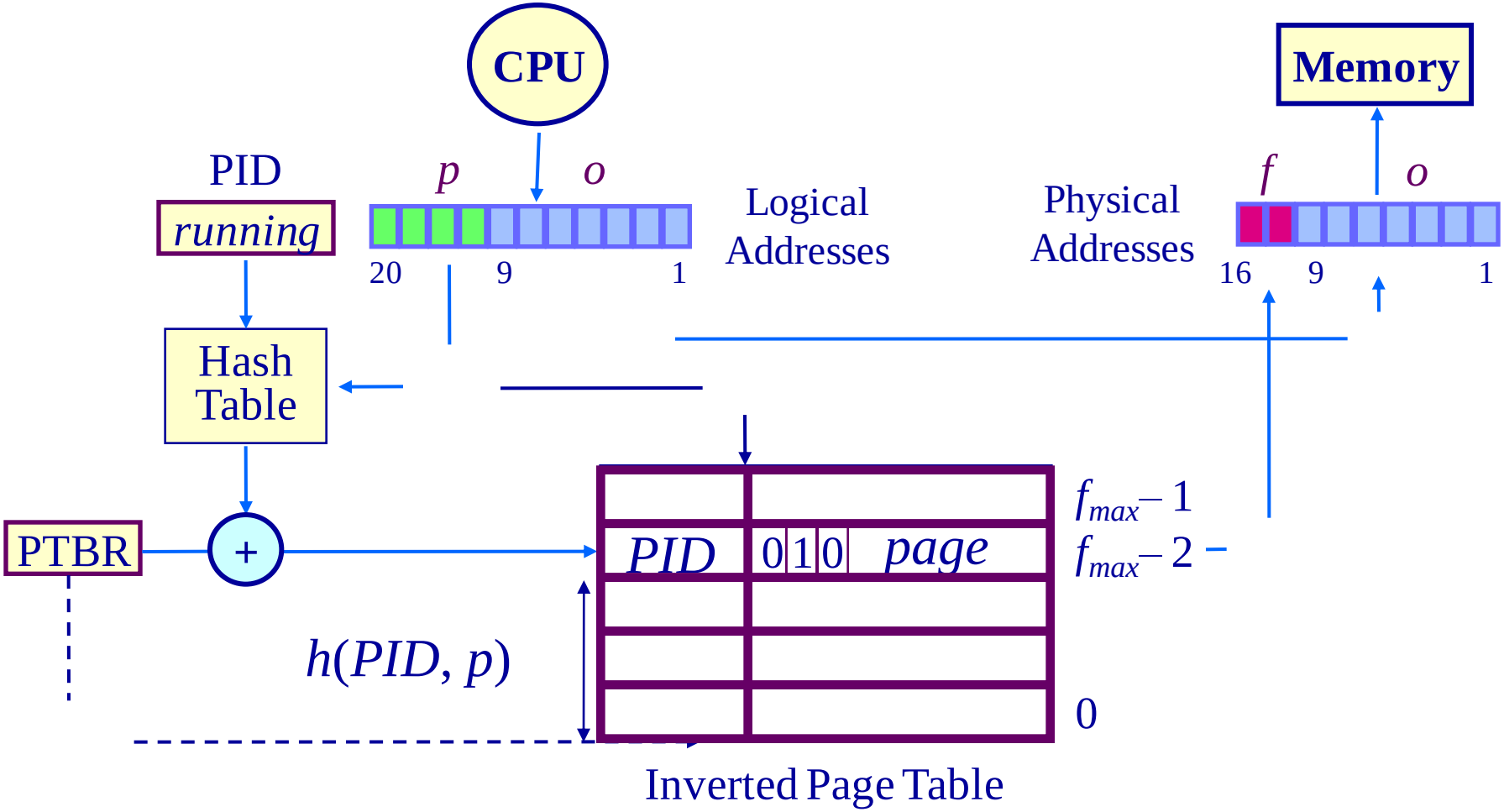
Non-contiguous Allocation : Page Table

Hashing Large Inverted Page Tables

- Hash page numbers to find corresponding frame numbers in a "frame" table with one entry per frame
- Page i is placed in slot f(i) where f is an agreed-upon hash function
- To lookup page i, perform the following:
  - Compute f(i) and use it as an index into the table of page registers
  - Extract the corresponding page register
  - Check if the register tag contains i, if so, we have a hit
  - Otherwise, we have a miss

Non-contiguous Allocation : Page Table

Hashed Inverted Page Table Architecture



Outline

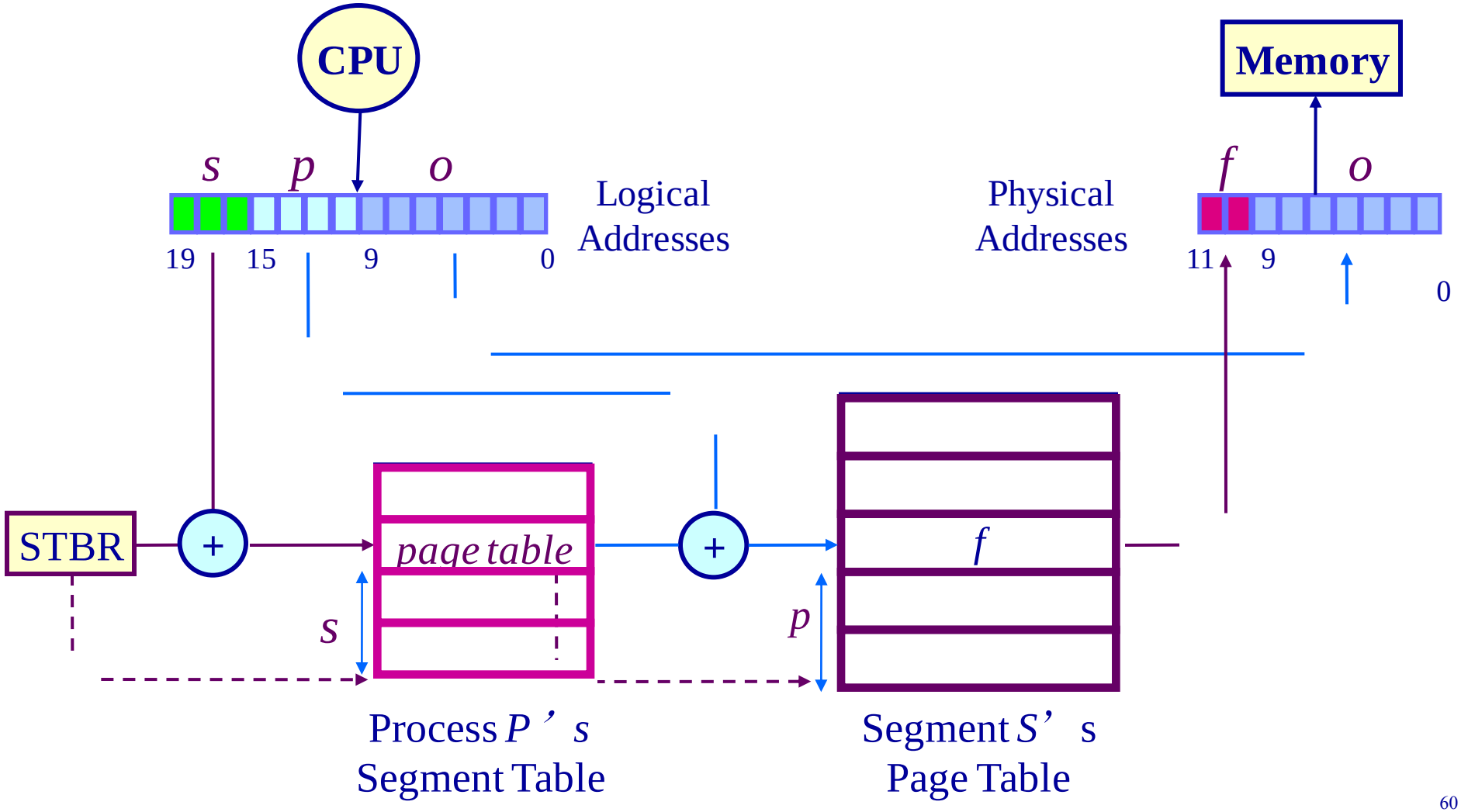
- Computer Arch/Memory Hierarchy
- Address Space & Address Generation
- Contiguous Memory Allocation
  - Dynamic Allocation of Partitions
- Non-Contiguous Memory Allocation
  - Segmentation
  - Paging
  - Page Table
    - Translation Look-aside Buffer (TLB)
    - Multi-Level Page Table
    - Inverted Page Table
  - Paged Segmentation Model

Paged Segmentation Model

- Segmentation has advantages for protection, paging has advantages for memory utilization and optimizing transfer to backing store.
- Can we combine segmentation and paging?

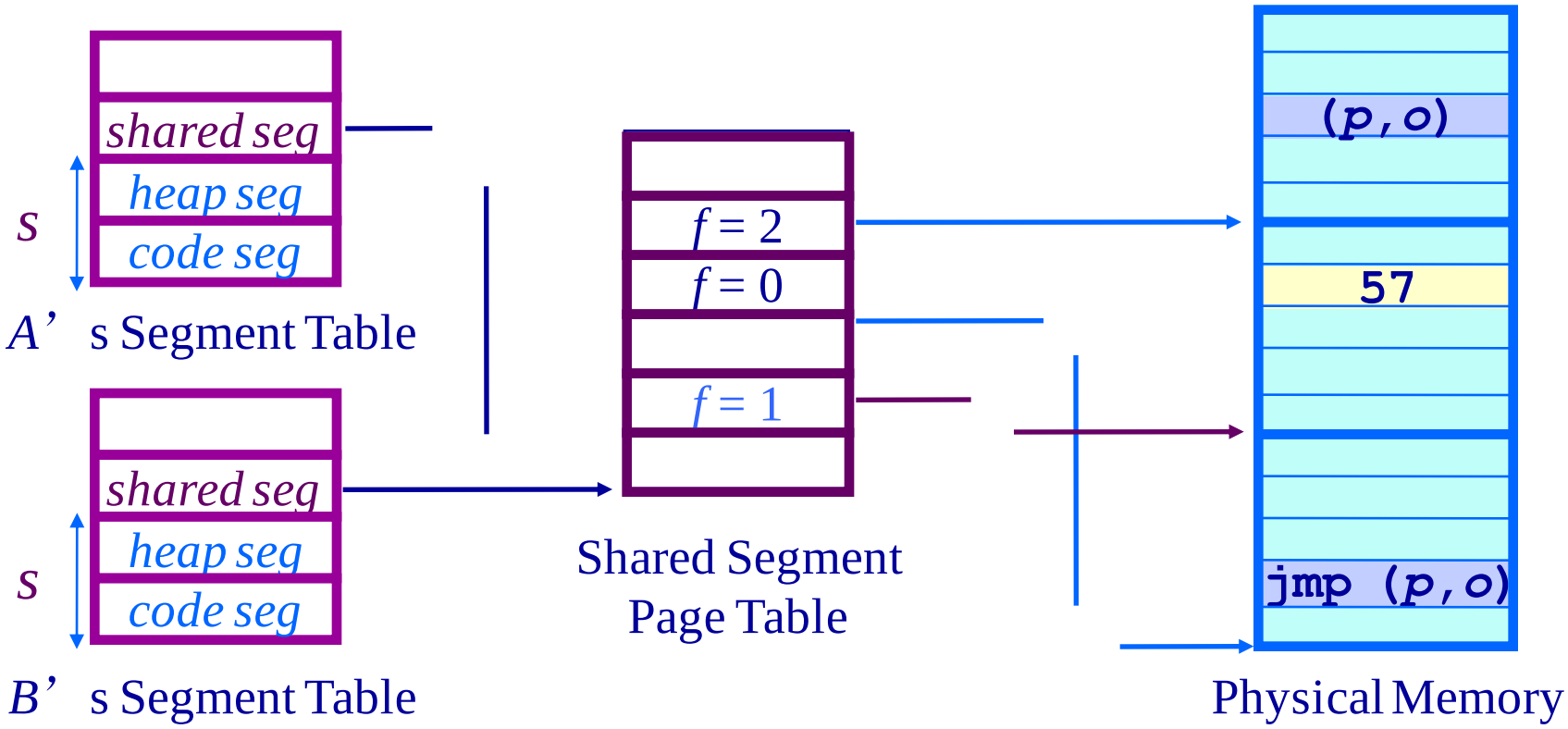
Paged Segmentation Hardware Architecture

- Add an additional level of indirection to page table



Sharing in Paged-Segmented Systems

- If segments are paged then page tables are automatically shared
  - Processes need only agree on a number for the shared segment



This week's Work

Lab1 should be finished!