Operating Systems (A)
(Honor Track)

Lecture 8: Memory Management

Yao Guo (郭耀)
Peking University
Fall 2021

Acknowledgements: Prof. Xiangqun Chen & Tao Wang at PKU and Prof. Yuanyuan Zhou at UCSD

Review: Threads

- The operating system as a large multithreaded program
  - Each process executes as a thread within the OS
- Multithreading is also very useful for applications
  - Efficient multithreading requires fast primitives
  - Processes are too heavyweight
- Solution is to separate threads from processes
  - Kernel-level threads much better, but still significant overhead
  - User-level threads even better, but not well integrated with OS

Review: Process vs. Thread

What are the main differences between processes and threads?

This Lecture

Memory Management

Overview

Memory Abstraction

Managing Free Memory

Buzz Words

Address space

Physical/virtual address

Relocation

Internal fragmentation

External fragmentation

Compaction

Swapping

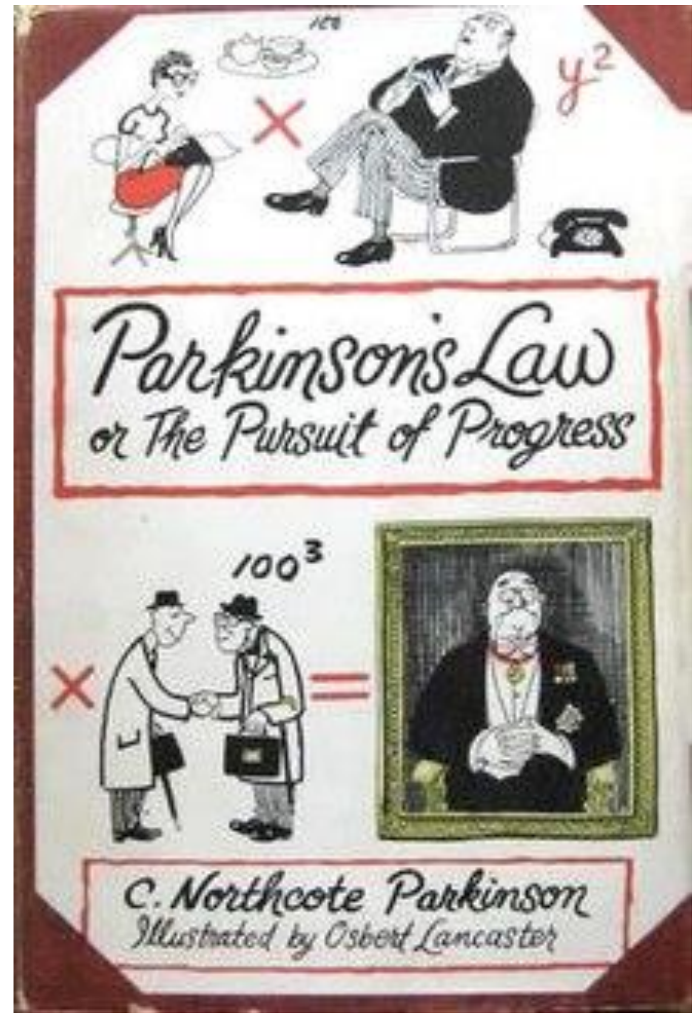
Memory is always limited

- Parkinson's Law:

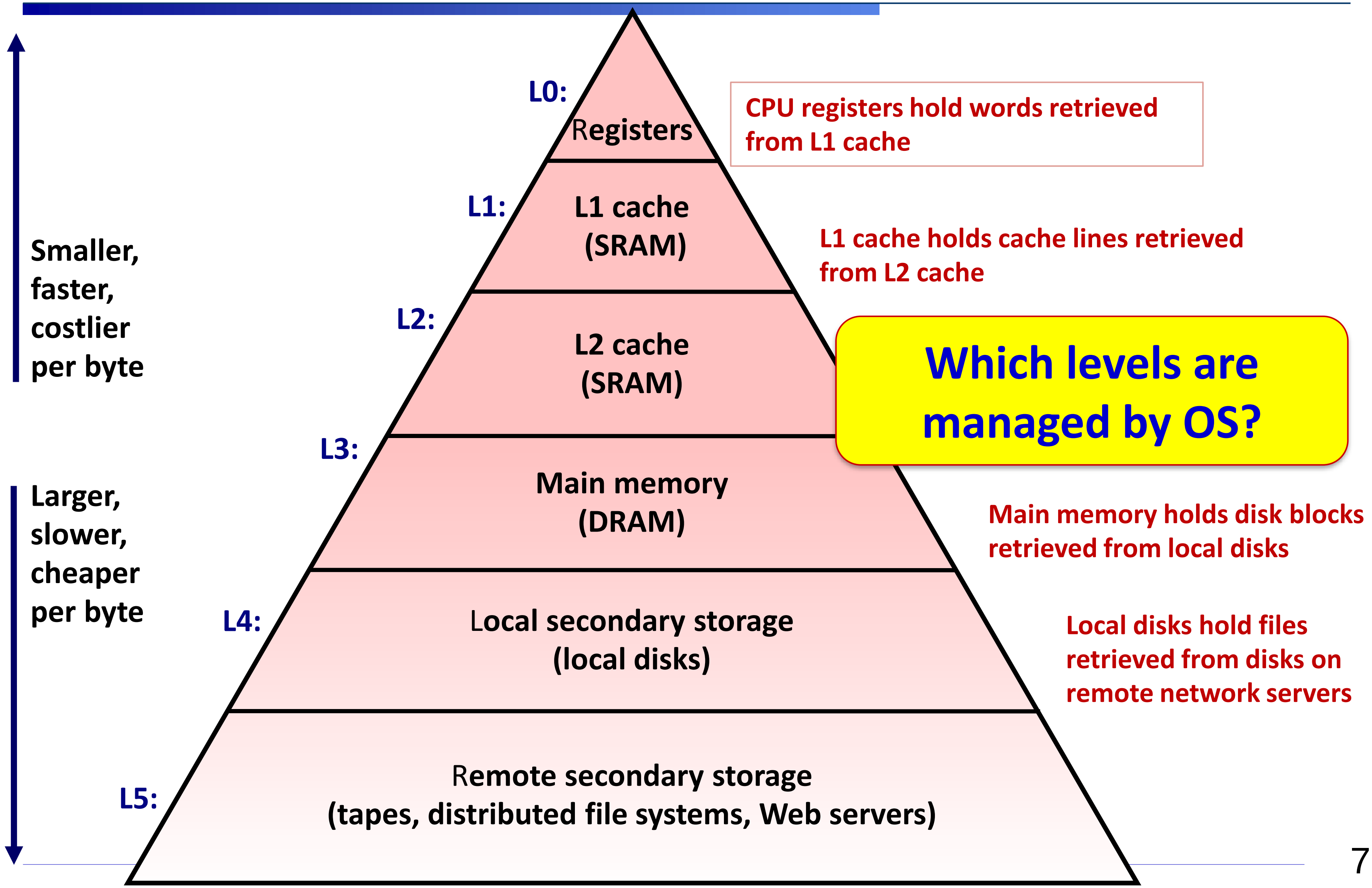
"Work expands so as to fill the time available for its completion"

- For Memory:

"Programs expand to fill the memory available to hold them."



Memory Hierarchy (from CSAPP)



Memory Manager

- Memory manager: the part of the operating system that manages (part of) the memory hierarchy
- The job of a memory manager is to efficiently manage memory:
  - Keep track of which parts of memory are in use,
  - Allocate memory to processes when they need it, and
  - Deallocate it when they are done

Memory Management

- Goals of memory management
  - To provide a convenient abstraction for programming
  - To allocate scarce memory resources among competing processes to maximize performance with minimal overhead
- Mechanisms
  - Physical and virtual/logical addressing
  - Techniques: partitioning, paging, segmentation
  - Page table management, TLBs, VM tricks
- Policies
  - Page replacement algorithms

Virtual Memory

- The abstraction that the OS will provide for managing memory is virtual memory (VM)
  - Virtual memory enables a program to execute with less than its complete data in physical memory
    - A program can run on a machine with less memory than it "needs"
    - Can also run on a machine with "too much" physical memory
  - Many programs do not need all of their code and data at once (or ever) – no need to allocate memory for it
  - OS will adjust amount of memory allocated to a process based upon its behavior
  - VM requires hardware support and OS management algorithms to pull it off

This Lecture

Memory Management

Overview

Memory Abstraction

Managing Free Memory

When there were none...

- The simplest memory abstraction is to have no abstraction at all.
  - early mainframe computers (before 1960),
  - early minicomputers (before 1970), and
  - early personal computers (before 1980)
- Every program simply saw the physical memory.
- Can we run two programs at the same time?
  - How about two processes?
  - And two threads?

The Need for Memory Abstraction

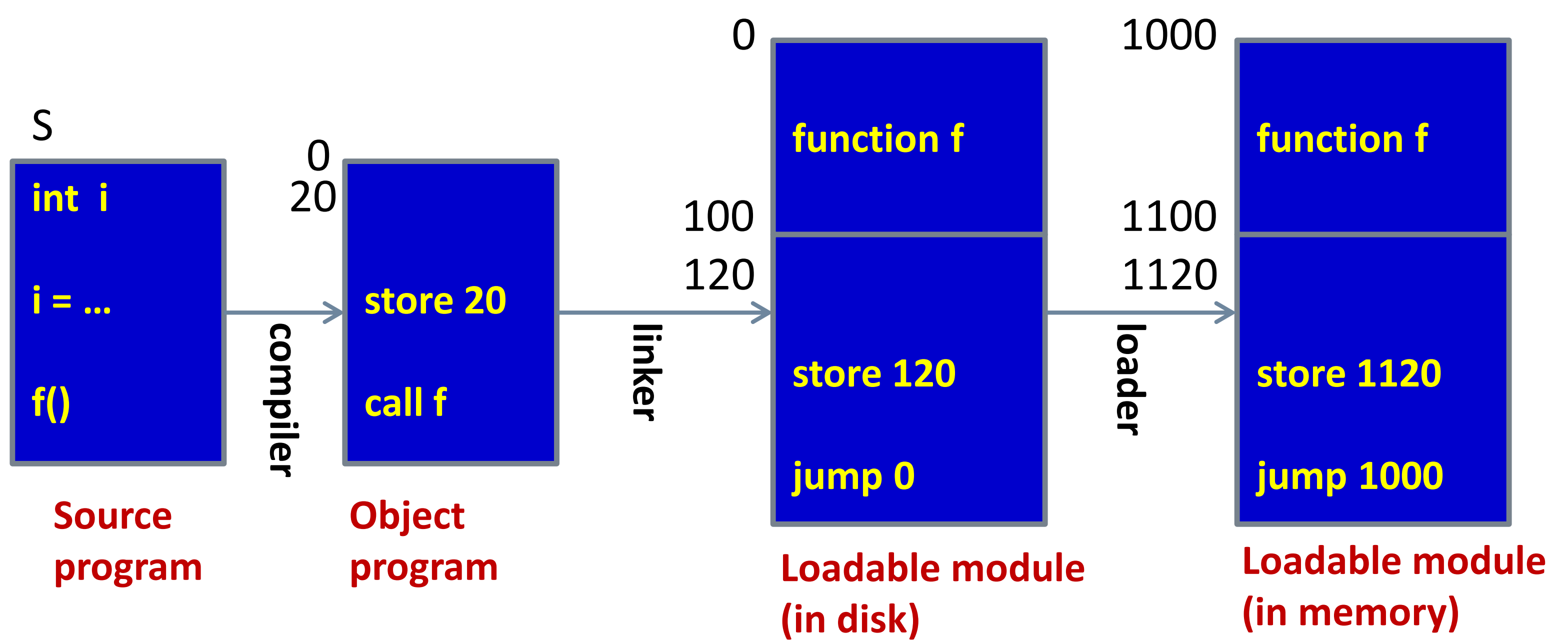
- Exposing physical memory to processes is bad
  - It can easily trash the operating system, even when there is only one process
- It is also difficult to support multiprogramming
  - Want multiple processes in memory at once
    - Overlap I/O and CPU of multiple jobs
  - Need protection – restrict which addresses jobs can use
  - Fast translation – lookups need to be fast
  - Fast change – updating memory hardware on context switch

Address Space

- An (virtual) address space is the set of addresses that a process can use to address memory.
- Each process has its own address space, independent of those belonging to other processes
  - With some exceptions, such as shared memory
- Relocation: logical address => physical address
  - Static relocation, or
  - Dynamic relocation

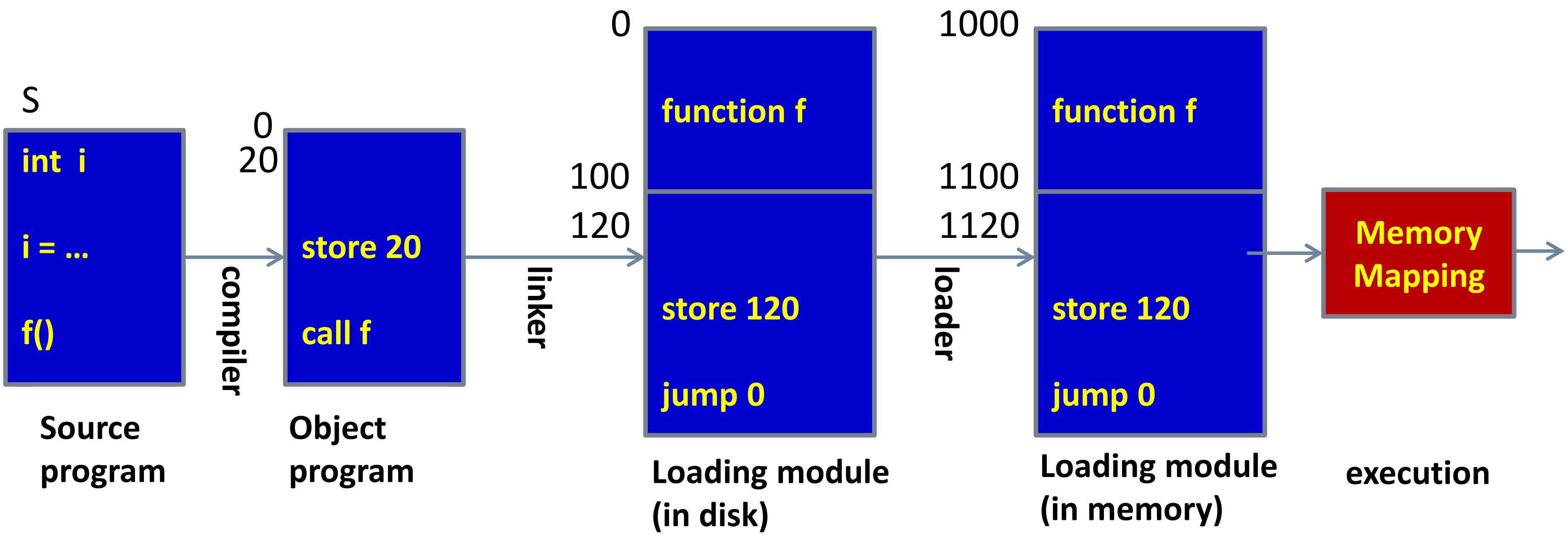
Static Relocation

- Static relocation: convert to physical addresses for all memory accesses during loading
  - Can be implemented in software
  - High overhead



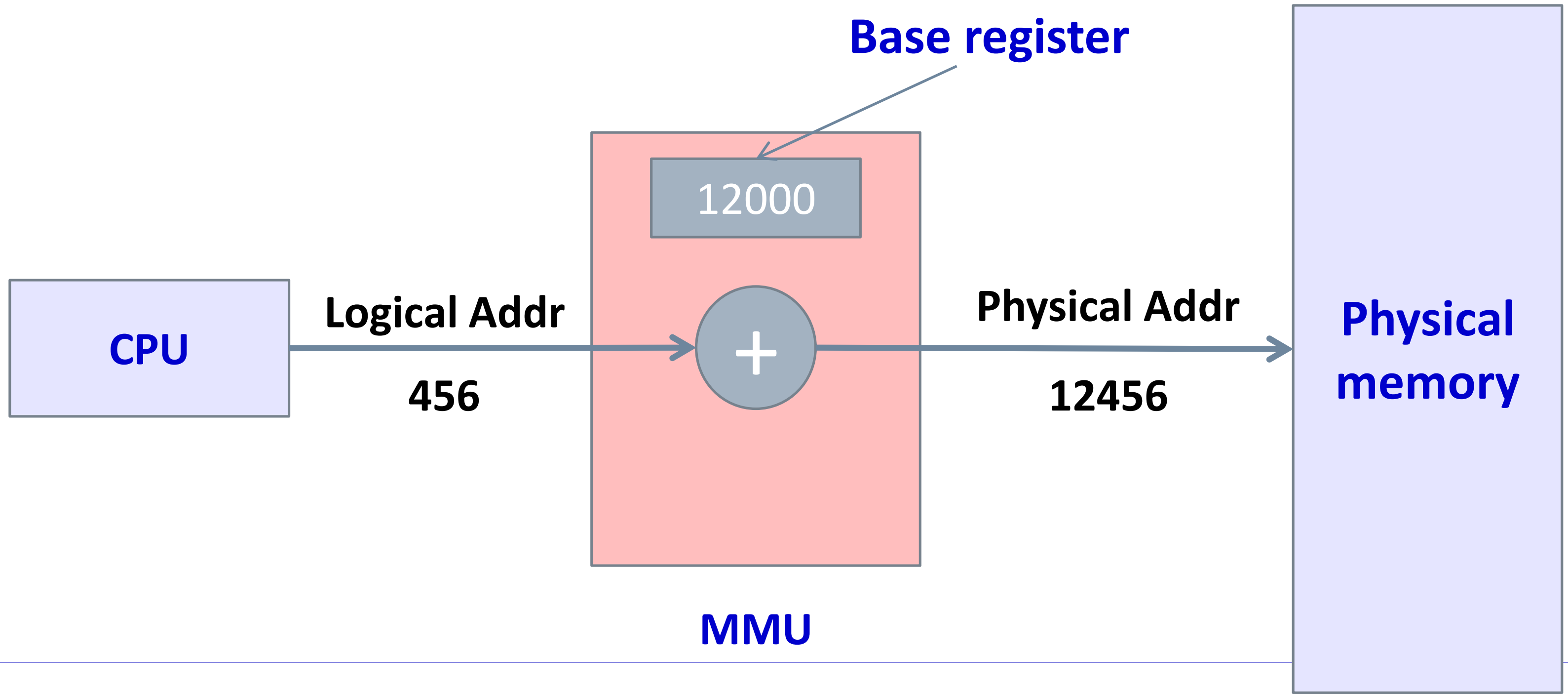
Dynamic Relocation

- **Dynamic relocation**: convert to physical addresses when an instruction is executed
  - Hardware support?



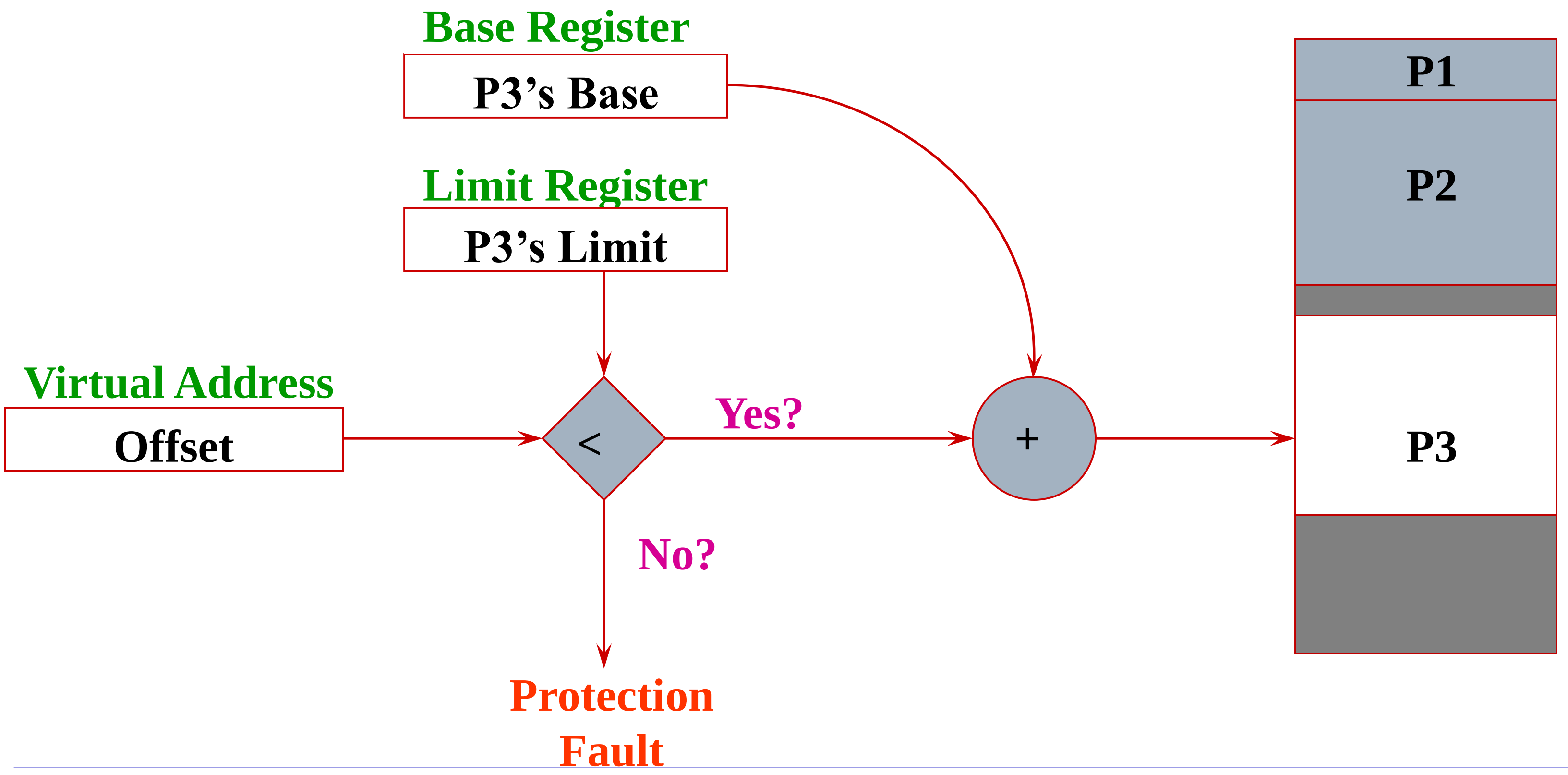
Dynamic Relocation Implementation

- MMU: Memory Management Unit
- Two special hardware registers: the base and limit registers.
  - Why do we need the limit register?



MMU Implementation

- Memory protection
  - If (physical address > base + limit) then exception fault



When Memory is Full…

- Swapping: swap in/out a process from/to disk

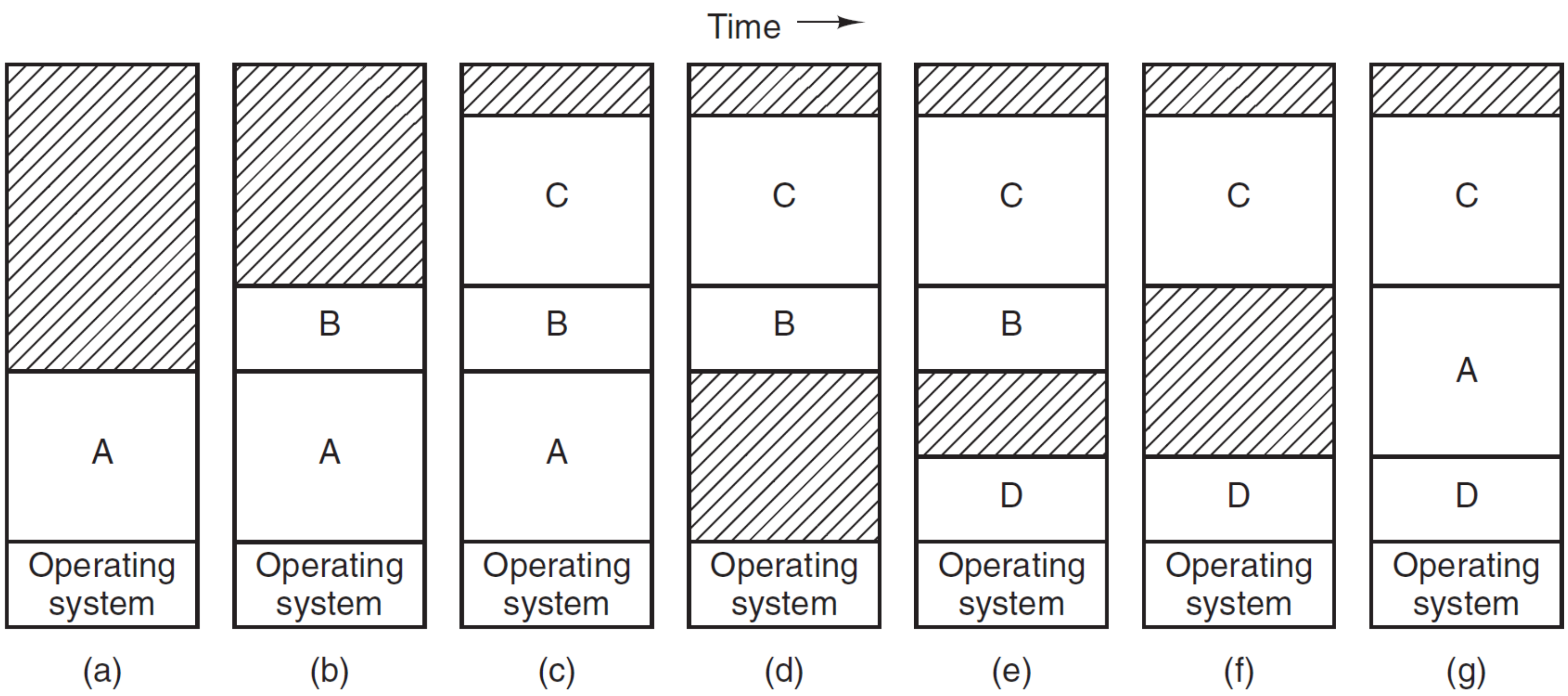


Figure 3-4. Memory allocation changes as processes come into memory and leave it. The shaded regions are unused memory.

This Lecture

Memory Management

Overview

Memory Abstraction

Managing Free Memory

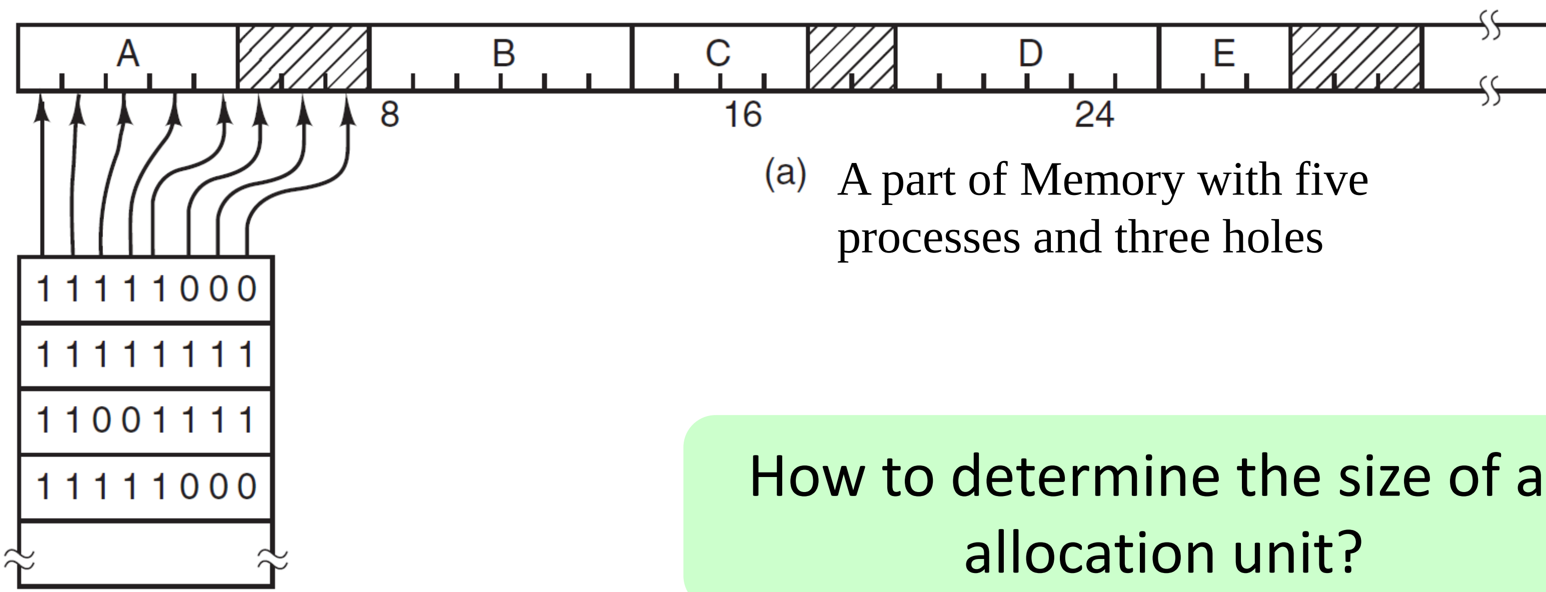
Managing Free Memory

- OS is responsible of managing dynamically allocated memory

- Methods:
  - Bitmaps
  - Free lists
- For Linux
  - Buddy system
  - Slab allocator

Memory Management with Bitmaps

- Each allocation unit is represented with 1 bit in the bitmap



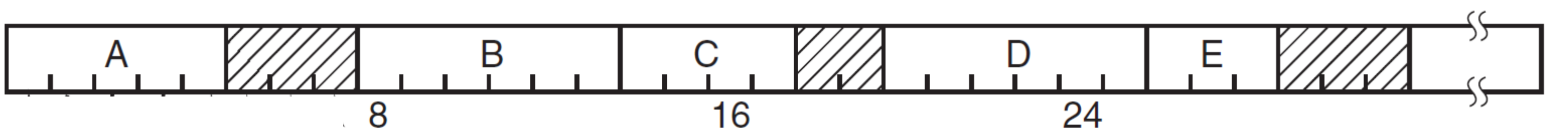
(a) A part of Memory with five processes and three holes

(b) The corresponding bitmap.

How to determine the size of an allocation unit?

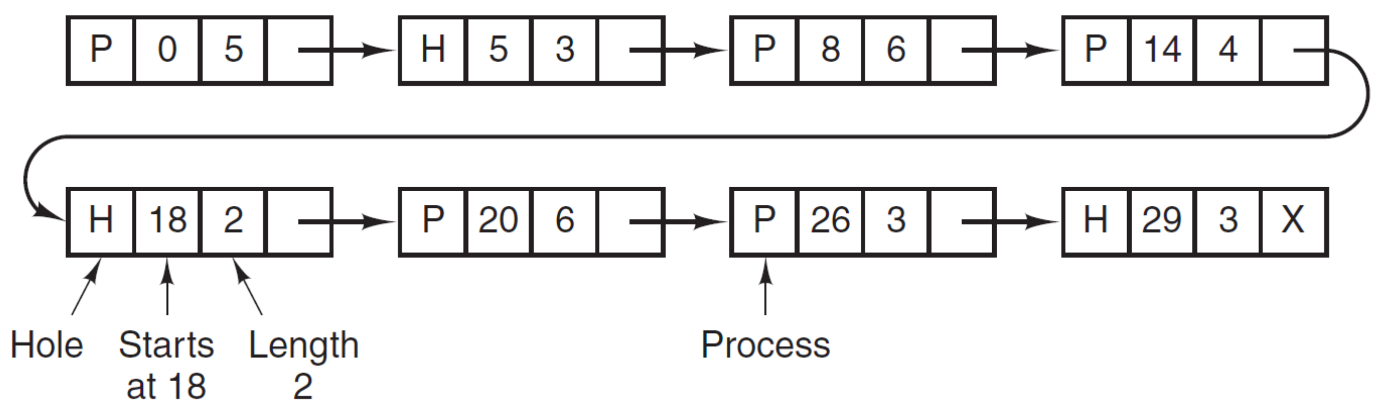
Memory Management with Linked Lists

- A linked list of allocated and free memory segments,
  - where a segment either contains a process or is an empty hole between two processes.



(a) A part of Memory

Linked lists with allocated and free units



(c)

Memory Allocation Algorithms

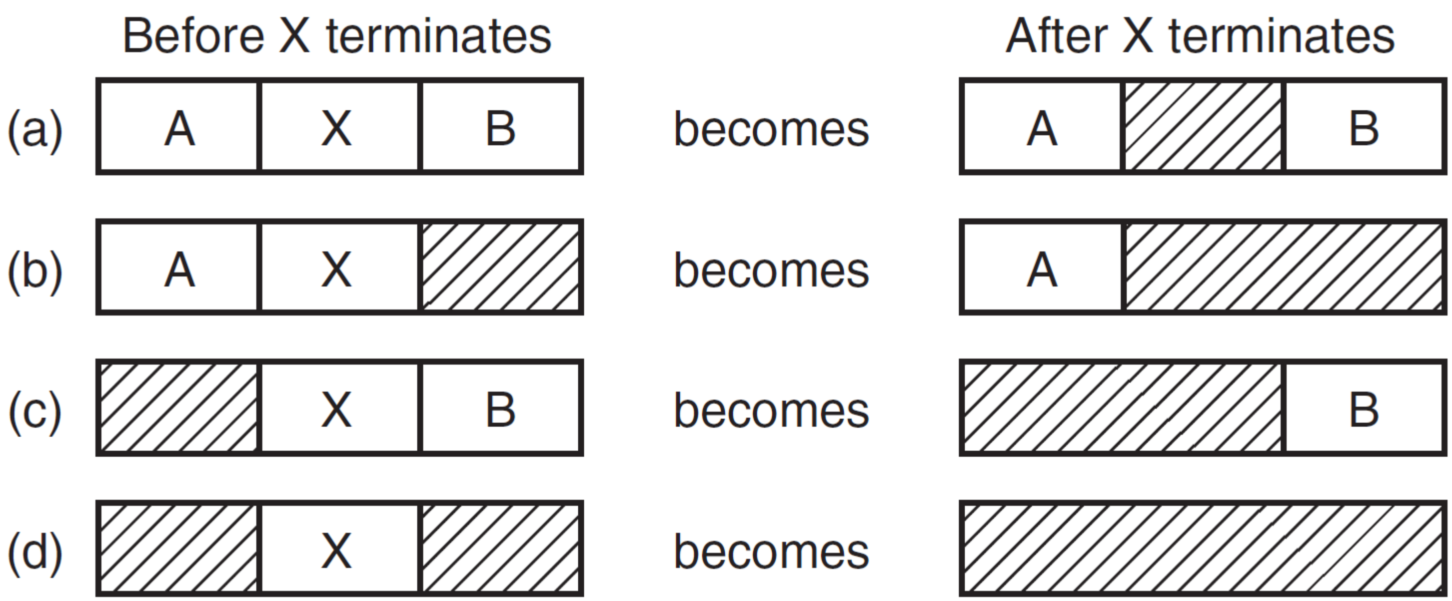
- How to allocate memory for a created process?
- First fit
  - Find the first hole that is big enough
  - Problem: creates average size holes
- Next fit
  - starts searching the list from the place where it left off last time
  - (Is it better than first fit?)
- Best fit
  - find the smallest hole that is adequate
  - Problem: creates small holes that can't be used

Memory Allocation Algorithms (cont.)

- How to allocate memory for a created process?
- Worst Fit
  - Use the largest available hole
  - Problem: gets rid of large holes making it difficult to run large programs
- Quick Fit
  - Maintains separate lists for some of the more common sizes requested
  - When a request comes for placement it finds the closest fit
  - Very fast, but merging is expensive

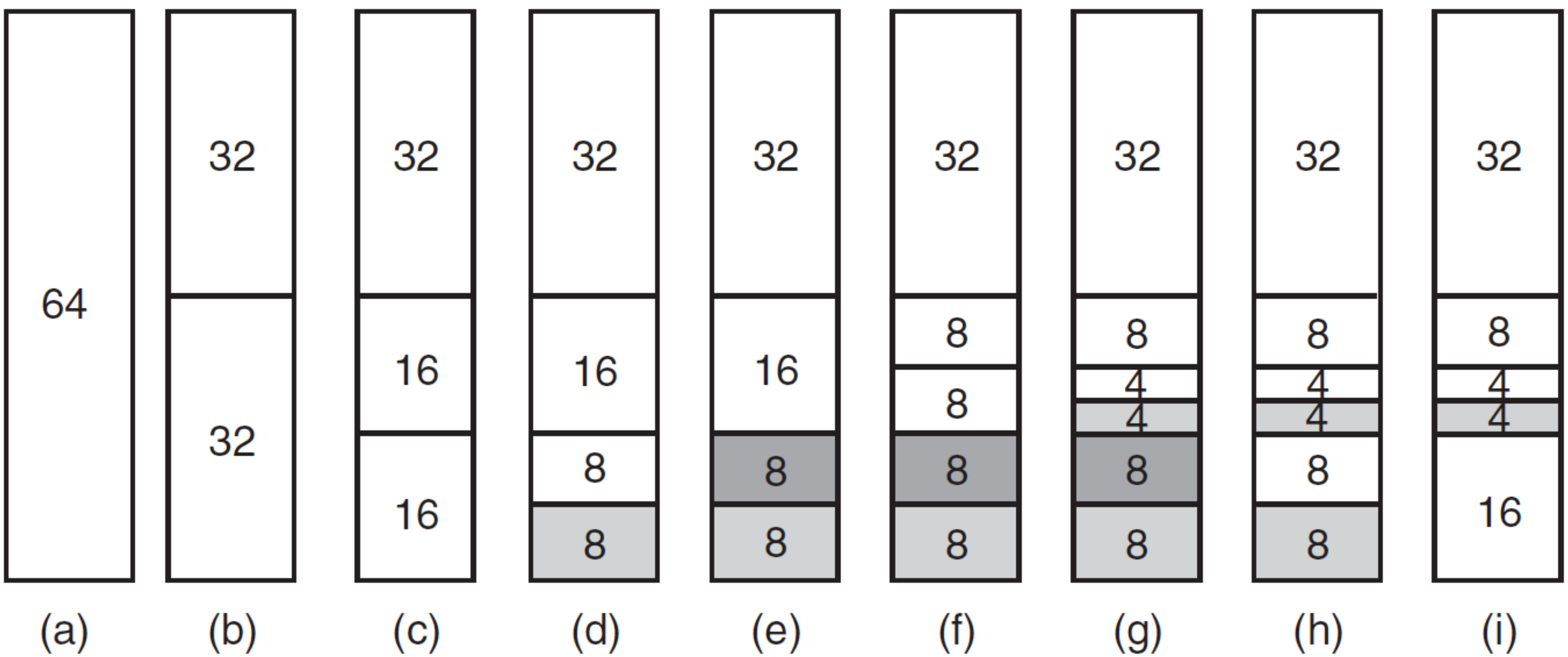
Memory Compaction

- A terminating process normally has two neighbors (except when it is at the very top or bottom of memory)
- Merge it with neighbors when it is terminated



Buddy System (Linux)

- It is also called the buddy algorithm
  - All free segments are in size of power of 2 (2^n)
  - Find the smallest hole larger than the requested size



……and other allocators

- Benefit: fast allocation possible
  - All blocks with the same size are put in a list
  - With an array pointing to each list with different sizes
- Problem: internal fragmentation is high, because the smallest unit is a page.
- Solution: Slab allocator
  - Managing small chunks (bytes within a page) separately
  - Used in kmalloc: allocates physically contiguous memory regions in the kernel address space
  - There is also vmalloc: allocates logically contiguous memory regions (with some performance degradation)

When there is not enough memory

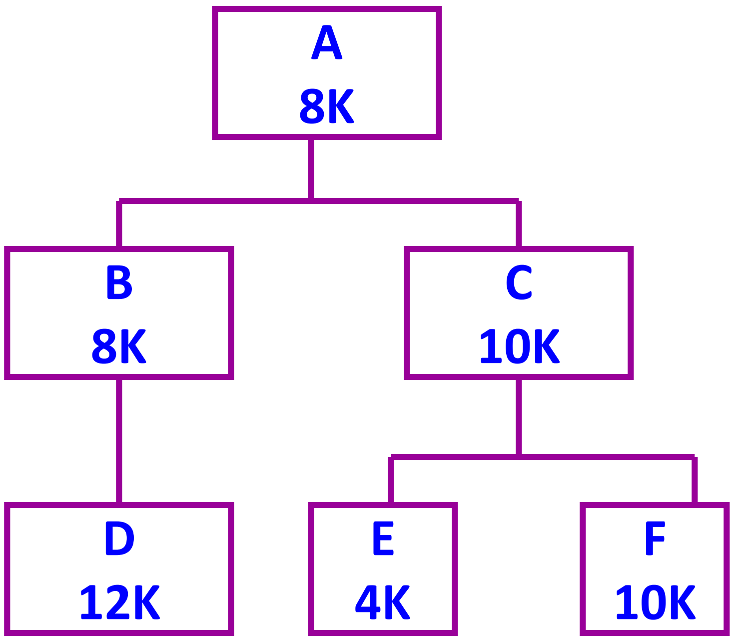
- What can an OS do when it does not have enough memory?
  - Memory compaction
  - Memory overlaying
  - Swapping
  - Virtual memory (next lecture)

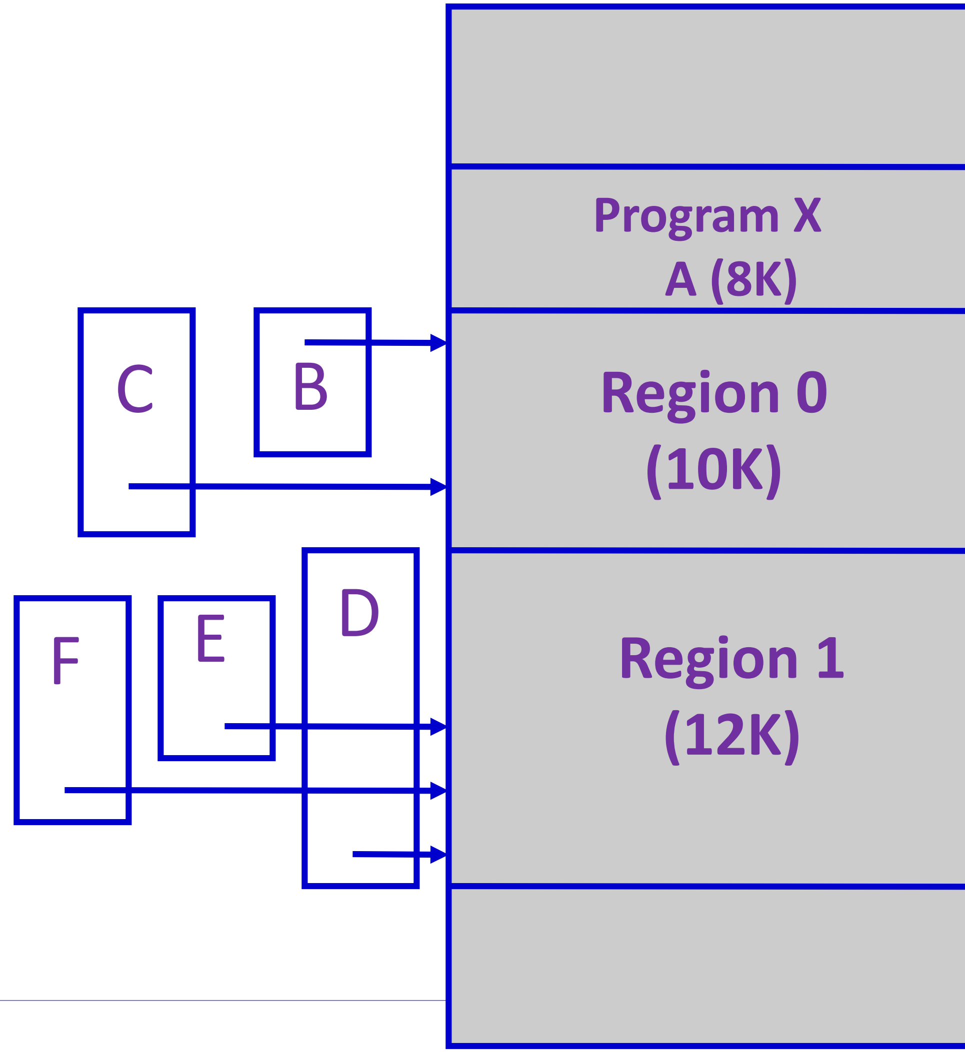
Memory Overlaying

- Static memory allocation
  - Different functions can reuse the same memory space if they are not dependent on each other
  - Load each overlaid module when it is called
  - Needs to know the calling relationship of all functions (subroutines)
- Used in early systems
  - Such as early FORTRAN

Example of Overlaying

Memory requirements:
30K





Calling graph of Program X:
Total requirements: 52K

Summary

- Overview of Memory Management
  - Memory abstraction: address space
  - OS + hardware translate virtual address into physical addresses
- Static and dynamic relocation
- Memory allocation algorithms
  - Fist fit, second fit, best fit, worst fit, quick fit
- Managing free memory
  - Bitmap, linked lists, buddy, slab, (and slub)

- Next lecture: Virtual Memory