Virtualizing Memory: Introduction

Operating Systems
Baochun Li
University of Toronto

OS: managing shared resources

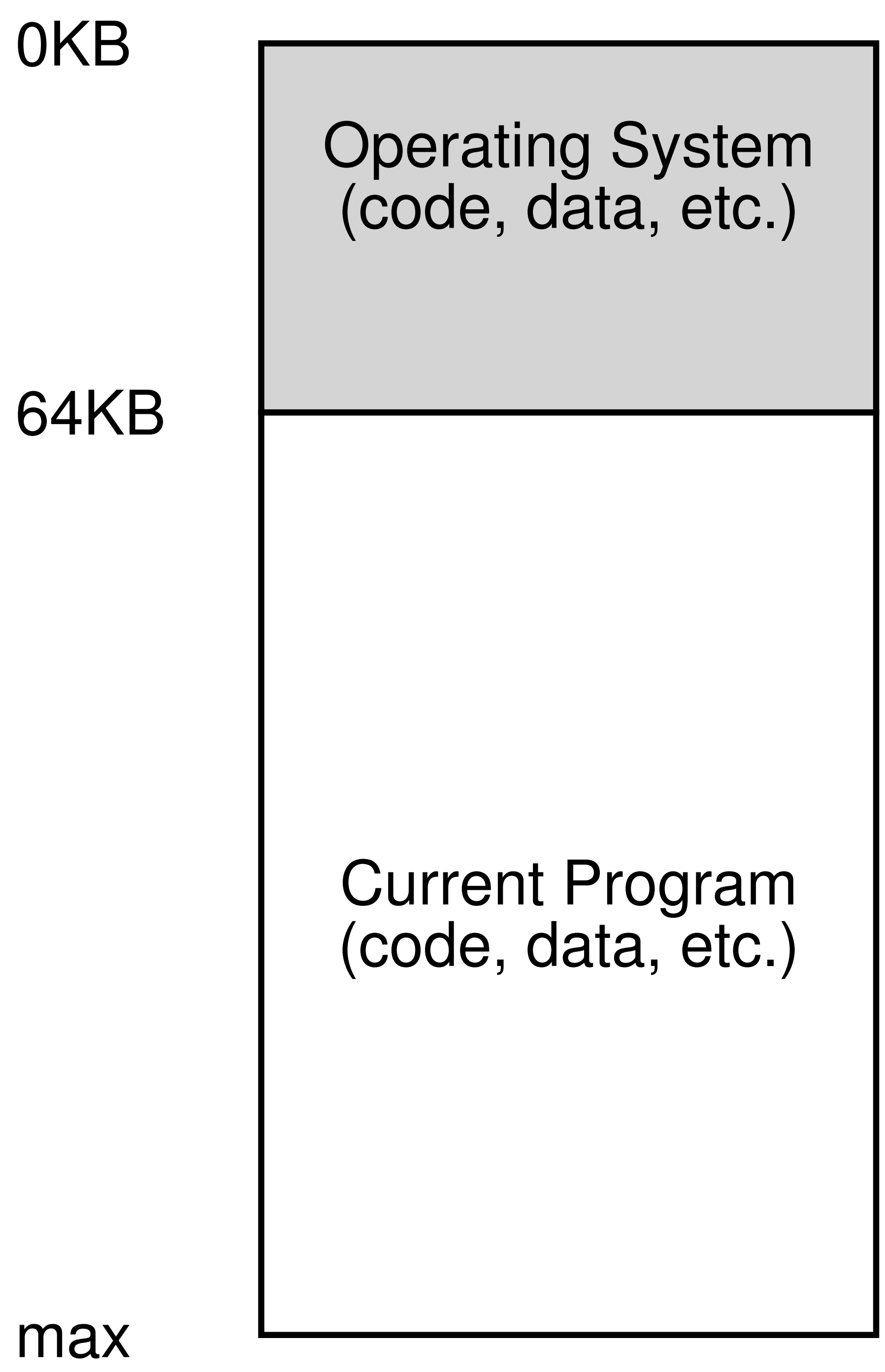
Sharing resources over time: Physical processors

- The main topic of past lectures

Sharing resources over space: Memory

- The main topic of upcoming lectures
- There is never enough memory
  - "640 KB ought to be enough for anyone."

Memory in early operating systems



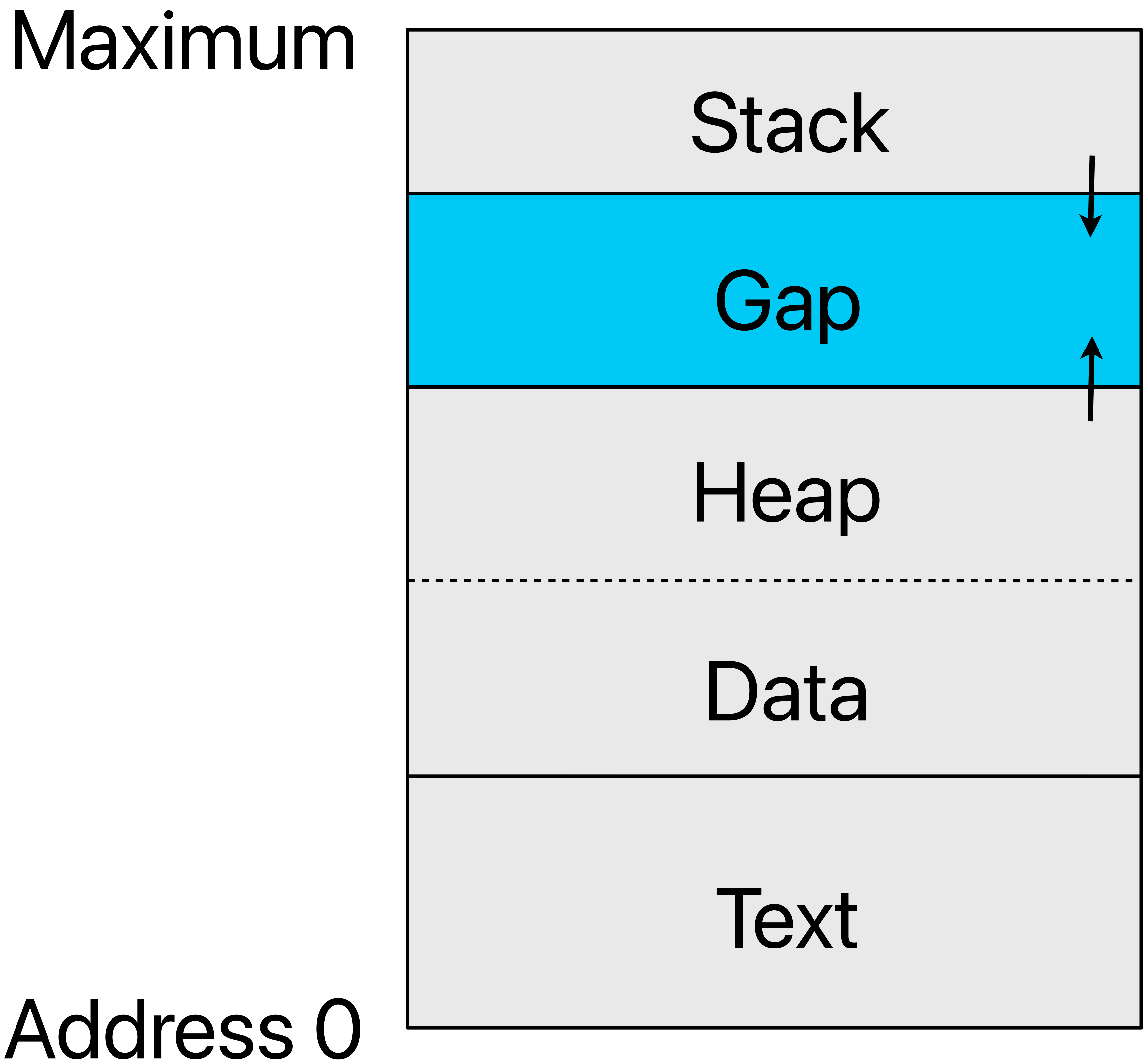
Multiprogramming and time sharing

We need to accommodate multiple processes

- Multiprogramming: multiple batch jobs run at the same time
- Time sharing: Multiple users using interactive processes at the same time

The important question is how?

Address space in a process: revisited



Address space: an abstraction

The user process uses virtual addresses in its own address space

The virtual memory system in the OS is responsible for virtualizing physical memory and provides the abstraction of address spaces to user processes

But how can the OS build this abstraction of a private, potentially large address space for multiple processes on top of a single, physical memory?

Before we introduce more ideas, let's first think about our goals

Goals of virtualizing memory

Transparency

Efficiency

Protection

Address Translation

Translating addresses at run time

Transforms each memory address (instruction fetch, load, store)

- From the **virtual** address provided by the instruction to its corresponding **physical** address
- This is to be performed at every memory reference since we need **transparency**
- But we also need **efficiency**!

Hardware Support: Memory Management Unit (MMU) — as part of the CPU

What does the MMU do?

Virtual memory addresses in an address space

↓

Address translation
performed on the fly
during program execution

$\boxed{\textbf{Memory Management Unit (MMU)}}$

↓

Physical memory addresses in the physical memory

The OS has to get involved to set up the hardware

Three assumptions to get started

A user process's address space must be placed **contiguously** in physical memory

The size of the address space is **less than** the size of the physical memory

Each address space is exactly the **same size**

Dynamic Relocation

MMU has one base and one bounds (or limit) register

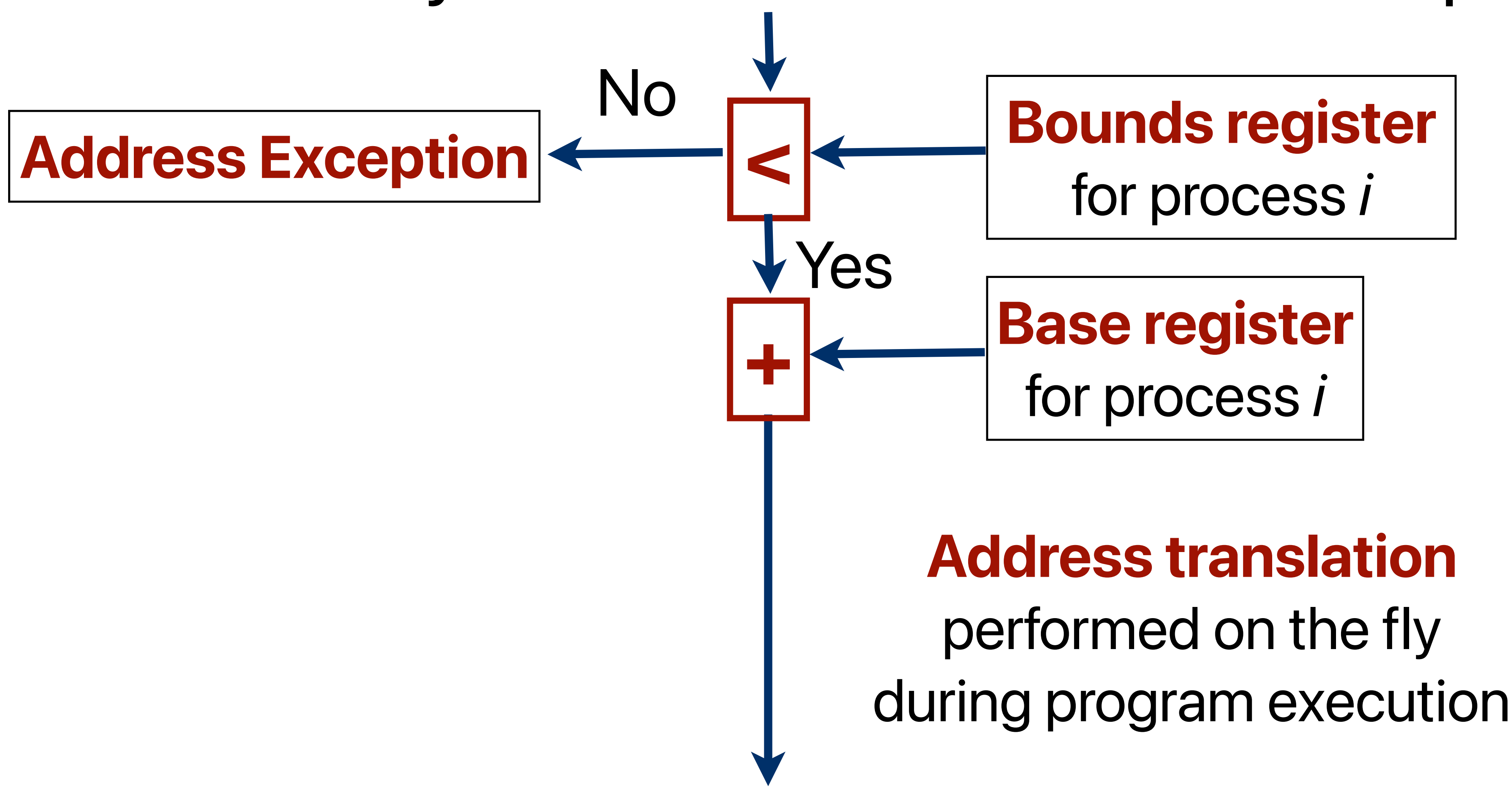
Base register converts each virtual address to a physical address by adding an offset — relocation

Bounds (limit) register keeps memory references within bounds — protection

OS assigns each process a separate base and limit register value when a process is started

Dynamic relocation: Base and bounds registers

Virtual memory address in a virtual address space



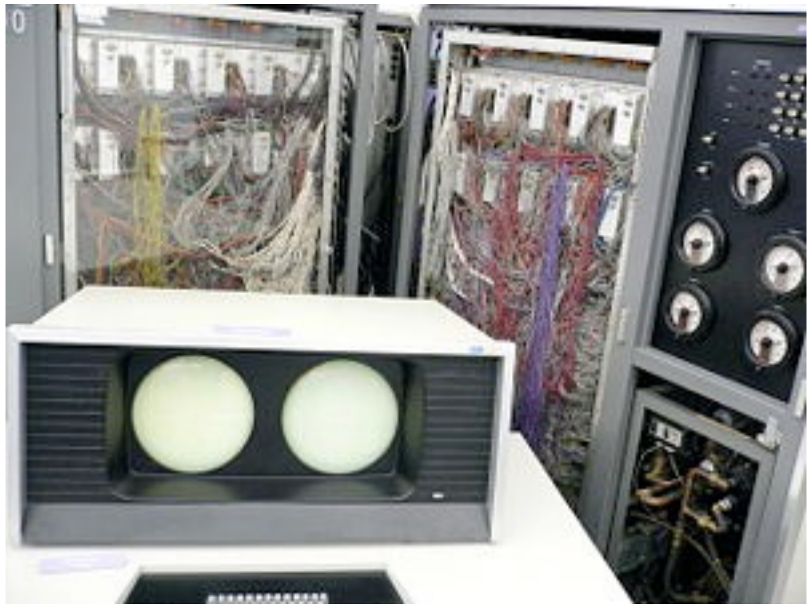
Physical memory addresses in the physical address space

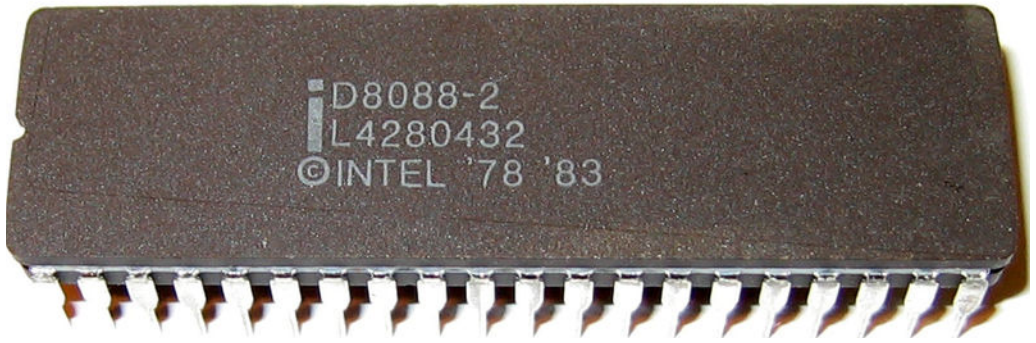
Base and Bounds Register MMUs

Used in these systems —

The first supercomputer: CDC 6600 (1964)

Intel 8088, original IBM PC (1980) (but with no Bounds Register)

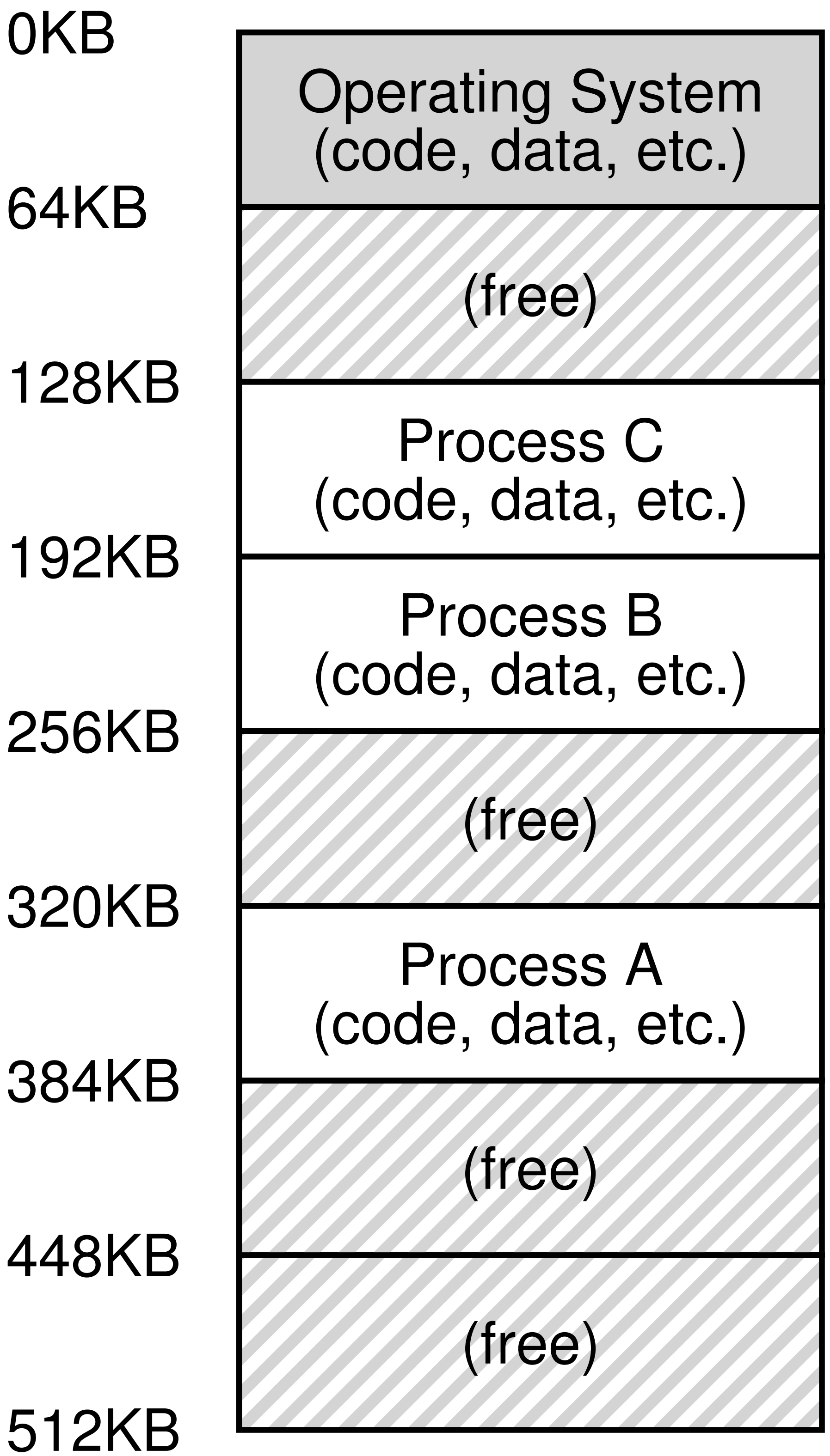




When a process is created, how can the OS find space in the physical memory for its new address space?

Given our assumptions:
fixed size and less than physical memory

Simple idea: maintain a free list



The work that the OS must do

When a process is created

Find a free entry in the free list and mark it as used

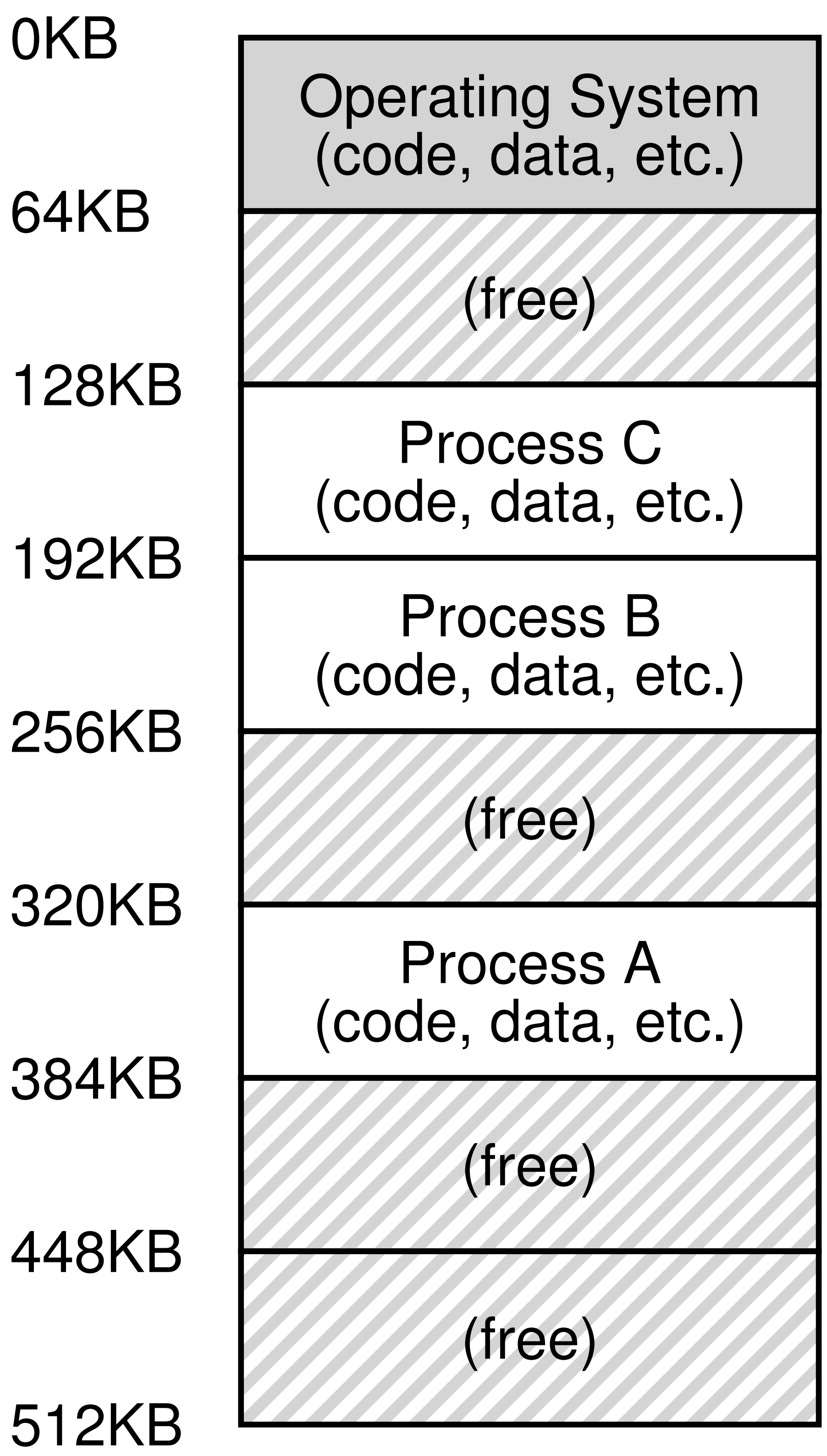
When a process is terminated (killed or exits gracefully)

Returns its memory back to the free list

During a context switch

Save and restore the base-and-bounds registers in the Process Control Block (PCB)

Any problems with base-and-bounds virtualization?



What we've covered so far

Three Easy Pieces

Chapter 13 (The Abstraction: Address Spaces), 15 (Mechanism: Address Translation)