

Memory (Part I)

Professor Travis Peters
CSCI 460 Operating Systems
Fall 2019

Some slides & figures adapted from Stallings instructor resources.

*Some slides adapted from Adam Bates's F'18 CS423 course @ UIUC
<https://courses.engr.illinois.edu/cs423/sp2018/schedule.html>*

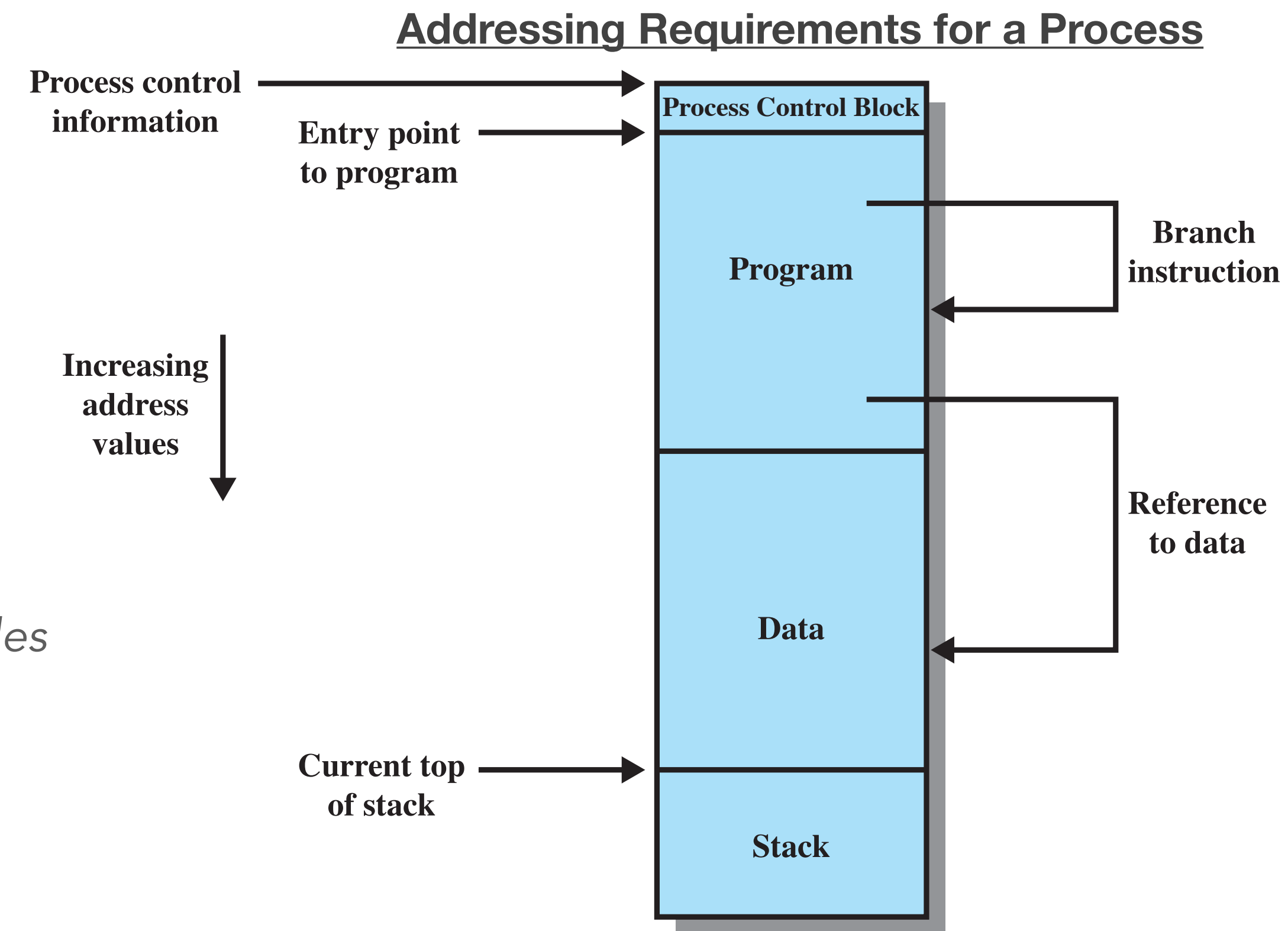
Goals for Today

Learning Objectives

- Understand basics of memory management, including
 - memory partitioning and common techniques
 - paging and segmentation — what they are, and their relative advantages and disadvantages
- Understand basics of loading and linking

Requirements for Memory Management

- Relocation
ability to move program memory around (e.g., due to swapping)
- Protection
*protection against unwanted interference by other processes (HW***)*
- Sharing
controlled access to shared areas of memory (e.g., multiple instances of the same program)
- Logical Organization
1-D/linear sequence of bytes; but most programs are organized as modules having different access permissions (R/W/X)
 - independent compilation
 - different degrees of protection
 - intuitive sharing mechanisms
- Physical Organization
2+ tiers of memory managed by the OS
 - main memory = faster, volatile
 - secondary memory = slower, non-volatile, larger (relative to MM)



Memory Partitioning: A Progression of Approaches

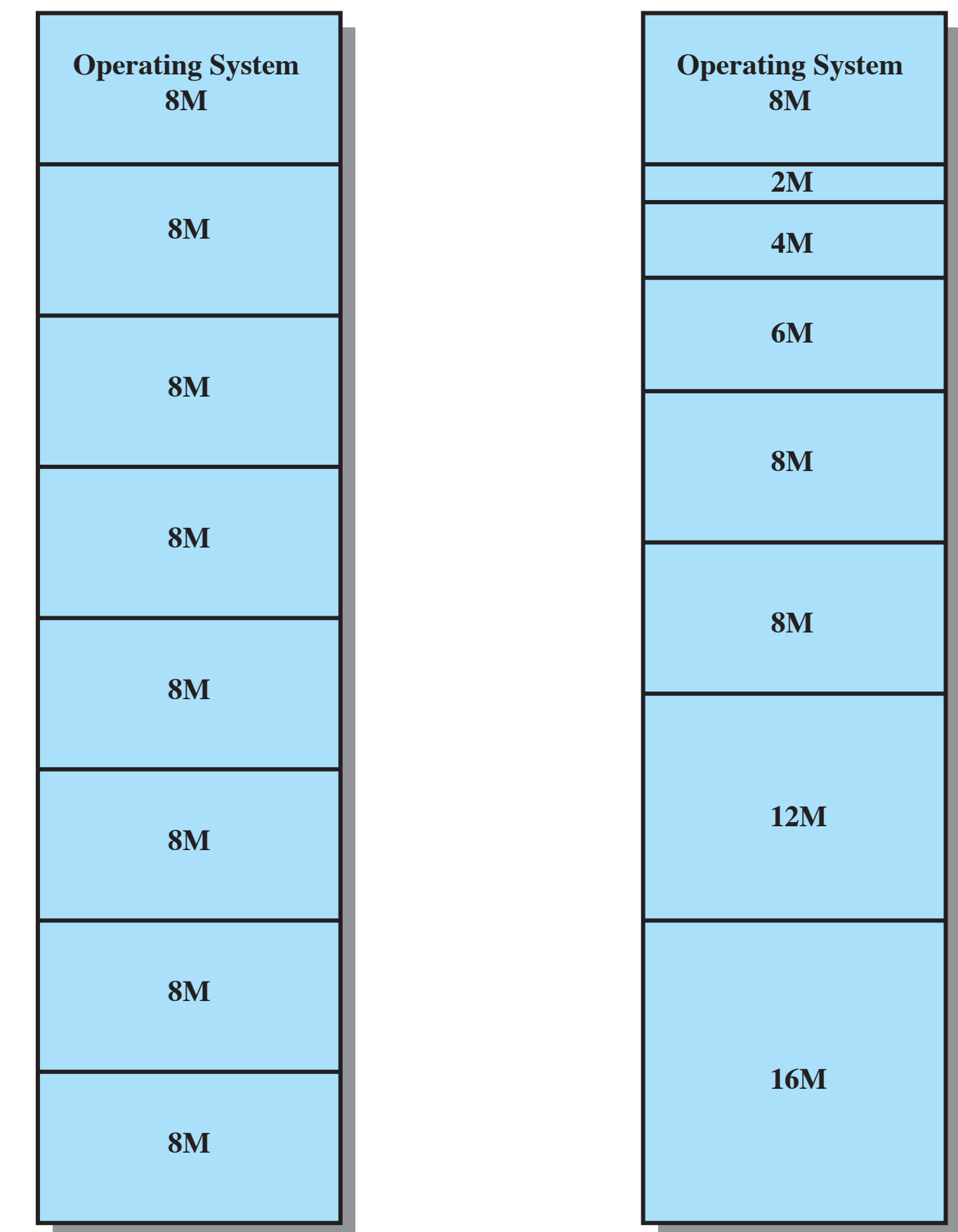
- Fixed Partitioning
- Dynamic Partitioning
- Paging
- Segmentation
- Paging **and** Segmentation

Overview of Memory Management Techniques

| Technique | Description | Strengths | Weaknesses |
|------------------------------------|--|--|---|
| Fixed Partitioning | Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size. | Simple to implement; little operating system overhead. | Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed. |
| Dynamic Partitioning | Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process. | No internal fragmentation; more efficient use of main memory. | Inefficient use of processor due to the need for compaction to counter external fragmentation. |
| Simple Paging | Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames. | No external fragmentation. | A small amount of internal fragmentation. |
| Simple Segmentation | Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous. | No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning. | External fragmentation. |
| Virtual Memory Paging | As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically. | No external fragmentation; higher degree of multiprogramming; large virtual address space. | Overhead of complex memory management. |
| Virtual Memory Segmentation | As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically. | No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support. | Overhead of complex memory management. |

Fixed Partitioning

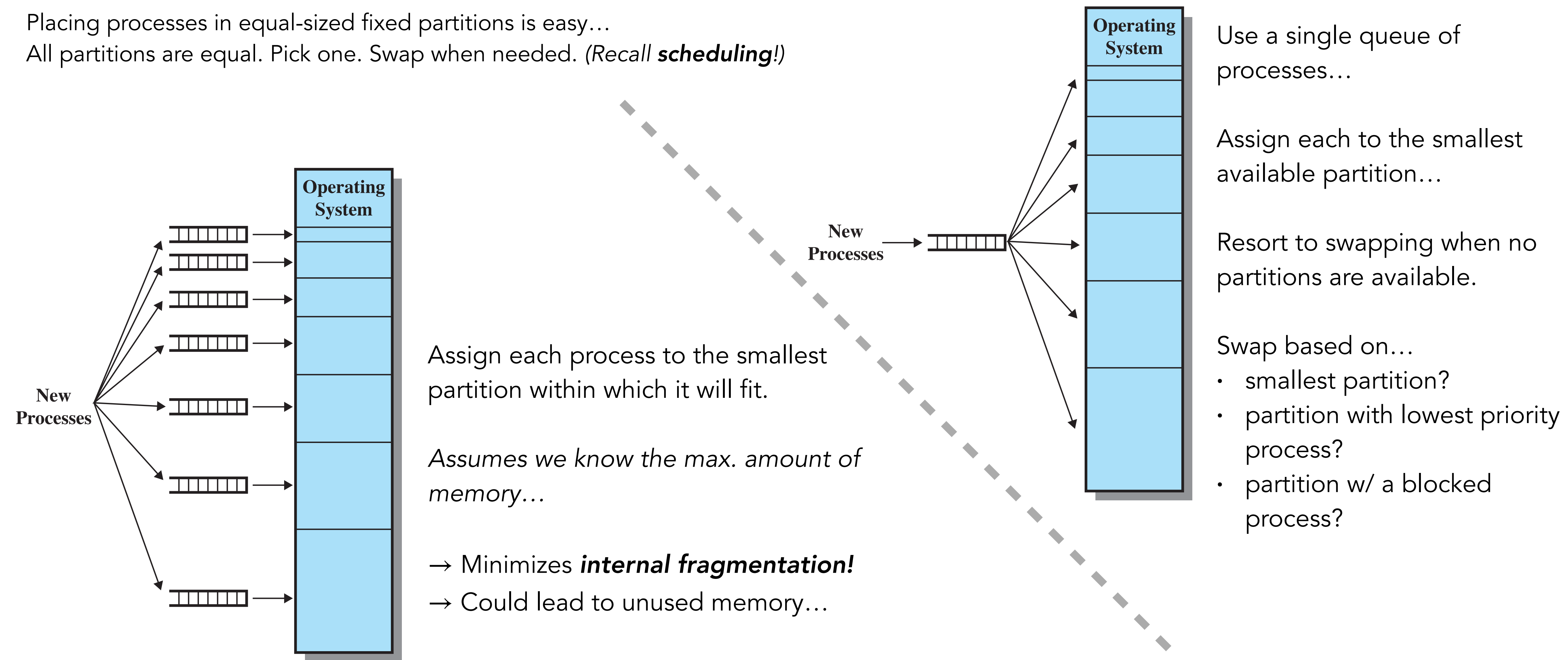
- OS occupies a fixed portion of main memory.
All other memory available for use by processes.
- **Simple Approach:**
Divide memory into fixed-size partitions
- **Example:** 64MB memory
 - What if a program doesn't fit into a partition?
→ **overlaying**
 - What if a program is much smaller than a partition?
→ **internal fragmentation**
 - How does program get put into a partition?
→ **loading function** (later...)
 - Can we do better?



equal partitions vs. **unequal** partitions

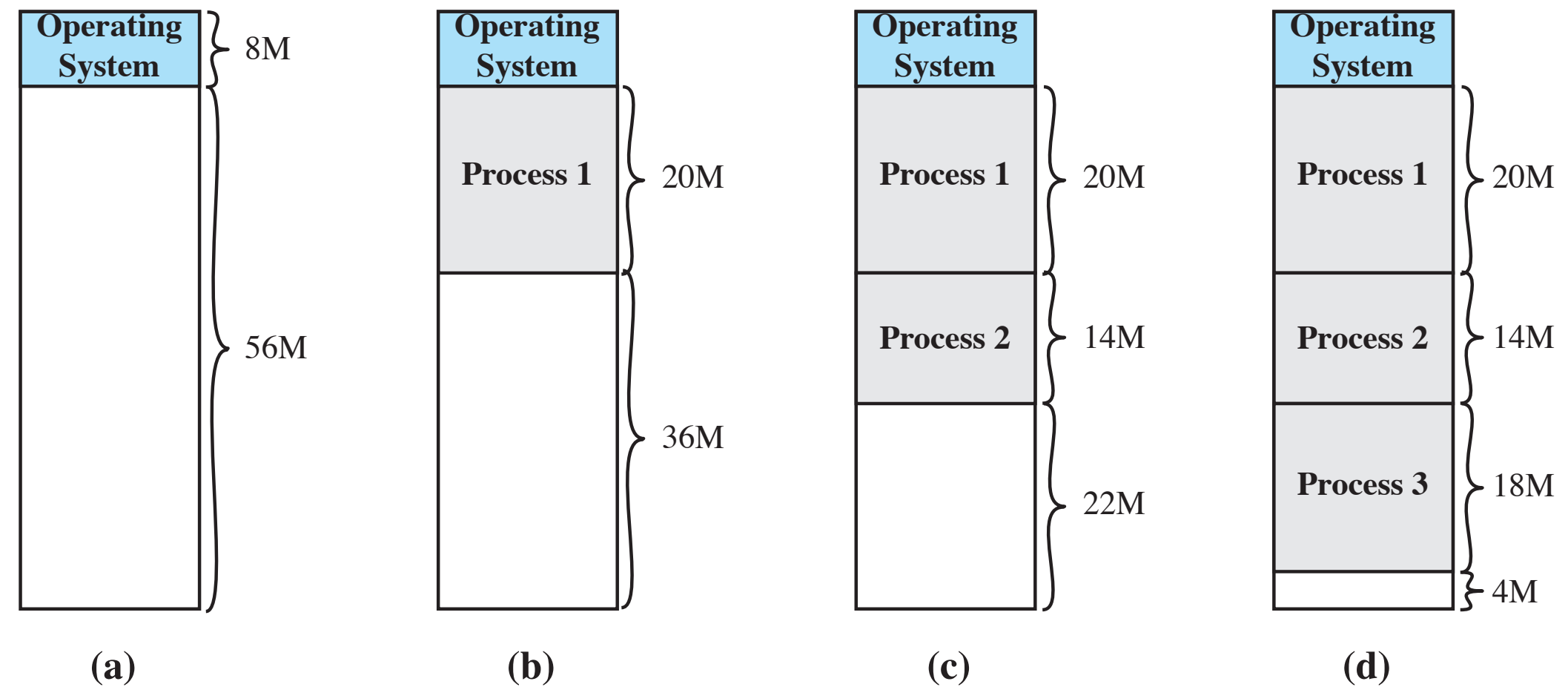
Placing Processes in Memory

Placing processes in equal-sized fixed partitions is easy...
All partitions are equal. Pick one. Swap when needed. (Recall **scheduling!**)



Dynamic Partitioning

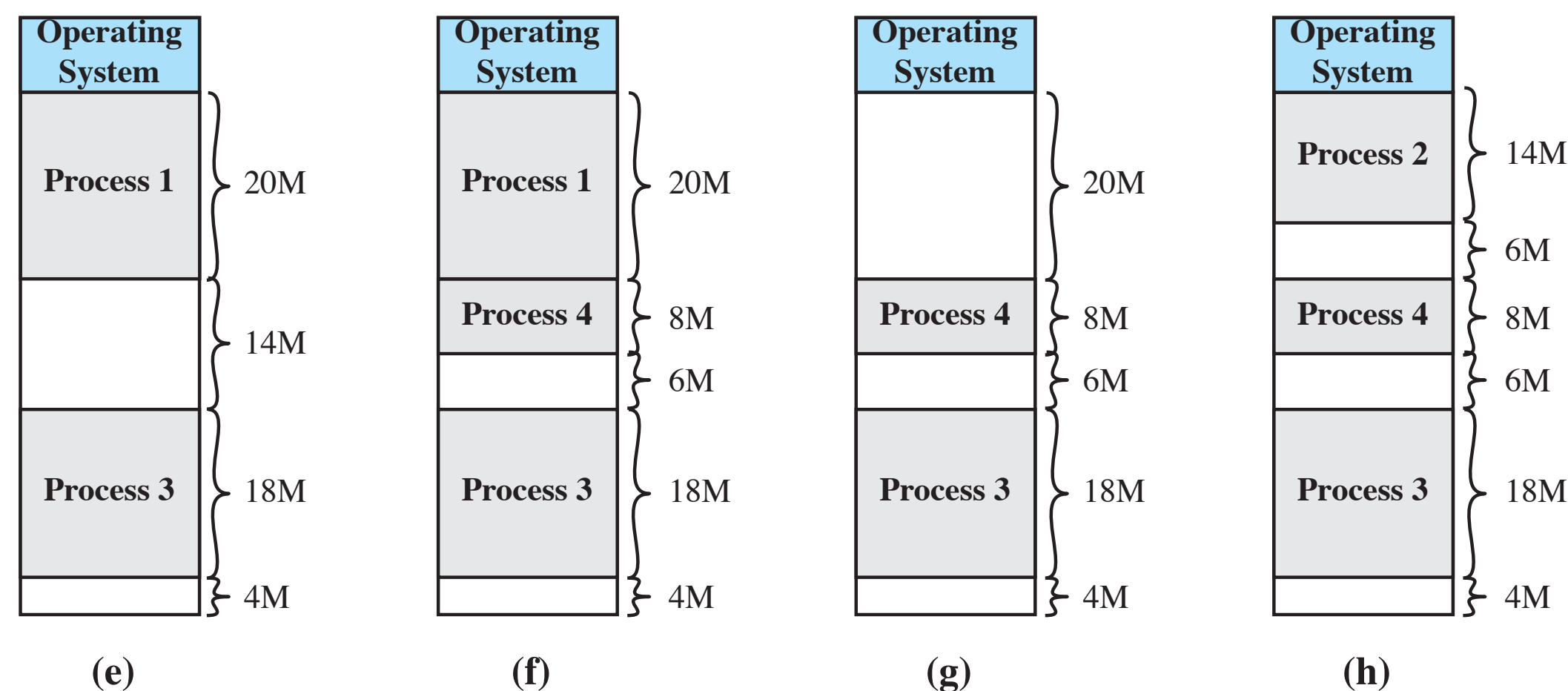
- partitions are of variable length & number
- each process allocated exactly as much memory as it needs



→ Quickly leads to **external fragmentation**!

→ One solution is **compaction**... *but this solution is time consuming and wasteful of processor time.*

→ How to best place processes in (available) memory?



→ How to best place (*fit*) processes in available memory?
(Recall: we'd like to avoid having to perform compaction often)

Placement Algorithms

Example: How to place a request for a 16MB block?

- **Best-Fit**

*scan all available memory blocks;
select block that is closest in size to the request*

Usually The Worst!

- **First-Fit**

*scan memory from start;
select the first available block that is large enough*

Simplest. Usually Best & Fastest.

- **Next-Fit**

*scan memory from location of the last placement;
select the next available block that is large enough*

Tends To Be Worse Than First-Fit...

