Universidade Estadual de Campinas
Instituto de Computação

MC504 Sistemas Operacionais

T09a Address Translation

Referências principais
Ch.15 of Operating Systems: Three Easy Pieces by Remzi and Andrea Arpaci-Dusseau (pages.cs.wisc.edu/~remzi/OSTEP/) and Ch.8 of Operating Systems Principles & Practice (Vol.III: Memory Management) by Thomas Anderson and Mike Dahlin.

Discutido em classe em 27 de agosto de 2018

Arthur João Catto, PhD

2º semestre de 2018

This module is intended to . . .

- Provide a detailed description of various ways of organizing memory hardware.
- Discuss various techniques for memory management, including paging and segmentation.

Main Points

- Address Translation Concept
  - How do we convert a virtual address to a physical address?
- Flexible Address Translation
  - Base and bound
  - Segmentation
  - Paging
  - Multilevel translation
- Efficient Address Translation
  - Translation Lookaside Buffers
  - Virtually and physically addressed caches

Memory Management is . . .

… subdividing main memory in order to accommodate multiple processes.

… allocating memory subdivisions to such processes to ensure a reasonable supply of ready processes to consume available processor time.

… especially dependent on the hardware design of the underlying computer system.

Background

Main memory and registers are the only storage that the CPU can access directly.

Register access can be made in at most one CPU clock cycle.

Main memory access can take many cycles.

To accommodate such difference, systems provide a high speed memory buffer called cache.

Background

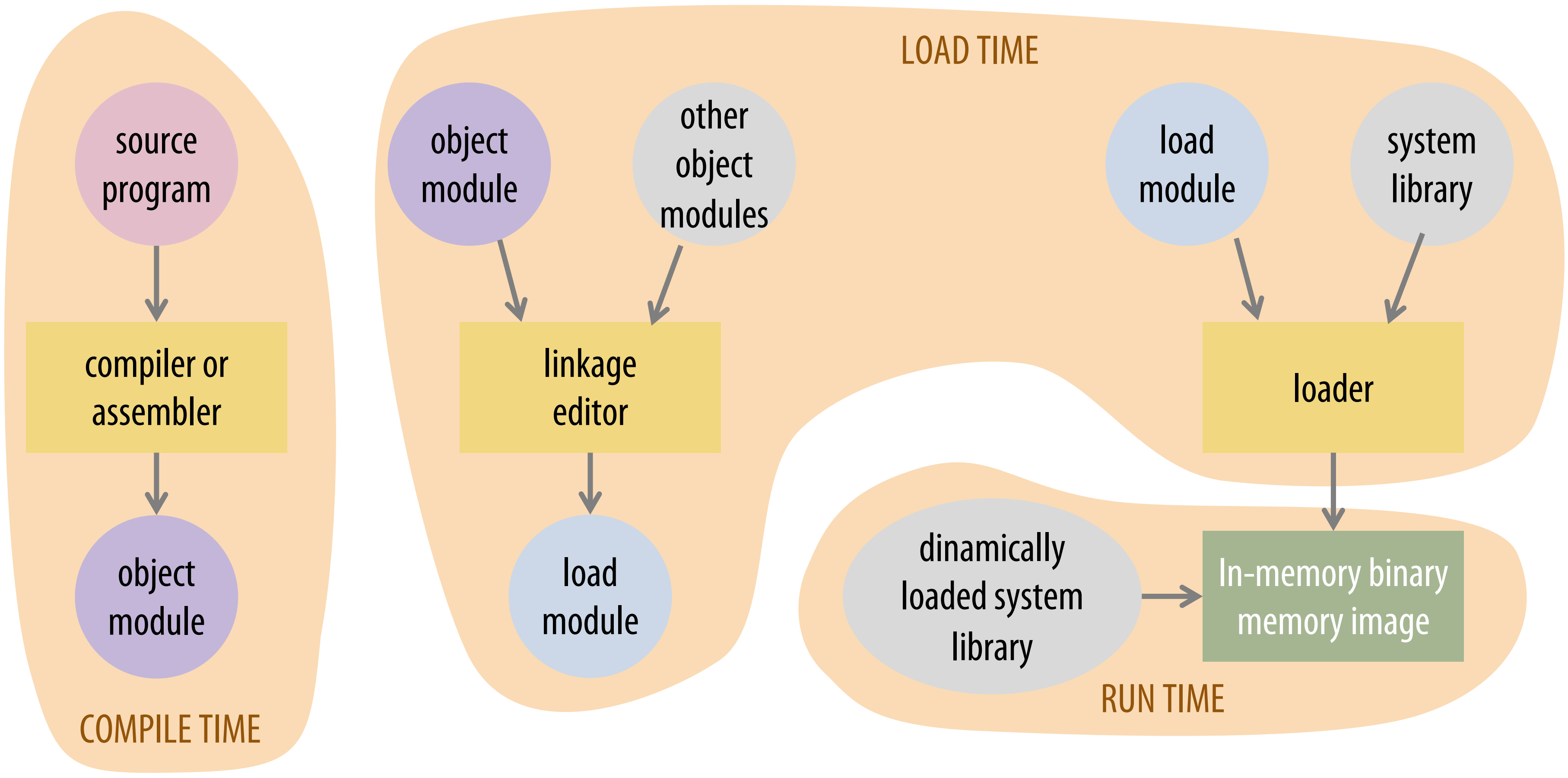
Programs must be brought into memory and placed within a process in order to be run.

Before being run, programs go through several steps.

Normally, there are always processes ready to run their programs.

Processes sharing memory, must be protected against unwanted mutual interference.

A user program has to go a long way before it runs. . .



Binding of instructions and data to memory can be performed at . . .

Compile time

- If memory location is known a priori.
- Code must be recompiled if starting location changes.

Load time

- If location will not change during execution.
- Code must be relocatable if memory location is not known at compile time.

Execution time

- If during execution the process may be moved from one place in memory to another.
- Needs hardware support for address maps (e.g., base and limit registers).

Dynamic Loading

- Routine is not loaded until it is called.
- Better memory-space utilization.
  - Any unused routines are never loaded.
- Useful when large amounts of code are needed to handle infrequently occurring cases.
- No special support from the OS is required.
  - Implemented through program design.

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.
- Particularly useful for libraries.

About address translation…

- Address translation is performed by the hardware to transform a **virtual address** into a **physical address**.
  - The desired data is actually stored in a physical address.
- The OS must be involved at key points to set up the hardware.
  - In order to do it properly, the OS must also be responsible for **memory management**.

Virtual vs. Physical Address Spaces

- The concept of a virtual address space that is bound to a separate physical address space is central to proper memory management.
  - Virtual address
    - Is generated by the CPU.
    - Is also referred to as a logical address.
  - Physical address
    - Is the address actually seen by the memory unit.

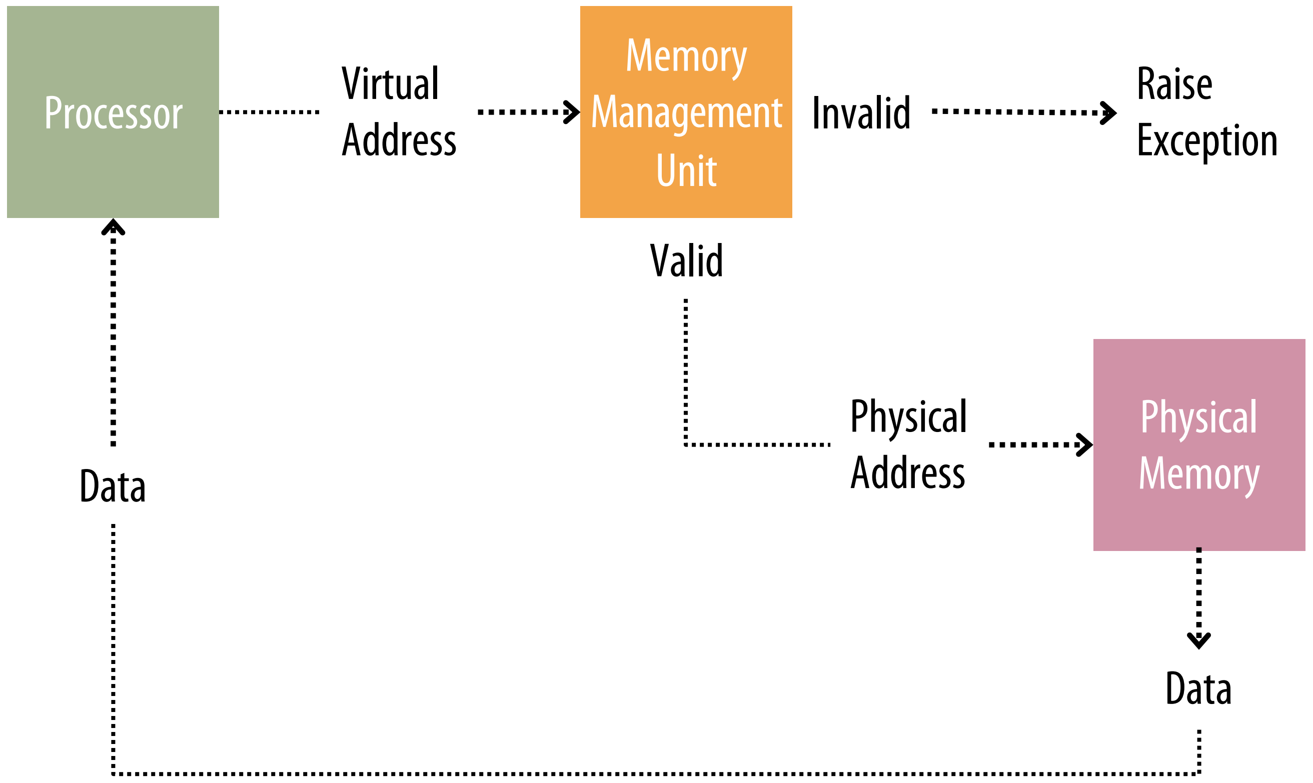
Virtual vs. Physical Address Spaces

- Virtual and physical addresses **are the same** in compile-time and load-time address-binding schemes.
- Logical and physical addresses **differ** in execution-time (or run-time) address-binding scheme.

Mapping virtual to physical addresses requires a dedicated device

- Memory-Management Unit (MMU)
  - Hardware device that maps virtual to physical addresses.
  - In an MMU scheme, the value in the relocation register is added to every address generated by a user process at the time such address is sent to memory.
  - Thus, the user program deals only with virtual addresses.
    - It never sees the real physical addresses.

Address Translation Concept



Address Translation Goals

Protection

Sharing

Sparse Addresses

Efficiency

Portability

Protection

- A process should not be able to reference memory locations in another process without permission.
- Since it is impossible to check absolute addresses at compile time, this must be done at run time.
- The memory protection requirement must be satisfied by the processor (hardware) rather than by the operating system (software).
  - The operating system cannot anticipate all the memory references a process will make.

Sharing

- Allow several processes to access the same portion of memory.
  - Shared libraries
  - Process intercommunication
- It is usually better to allow each child process to access the same copy of their parent's program rather than have their own separate copy.

Sparse addresses

- Programs are written in modules, involving multiple regions of dynamic allocation (heaps/stacks).
- Modules can be written and compiled independently.
- Different degrees of protection are given to modules.
  - Read-only, execute-only, etc.
- Modules can be shared among processes.

Efficiency

- Memory available for a program plus its data may be insufficient.
  - **Overlaying** allows various modules to be assigned the same region of memory, which is inefficient as we'll discuss later.
- Programmer does not know how much space will be available at run time.
  - Run time lookup takes time.
  - Translation tables take space.

Portability

- Programmer does not know where the program will be placed in memory when it is executed.
- It should be possible to relocate a program (i.e. return it to main memory at a different location), after having swapped it to disk.
- Thus, the translation of memory references in the code to actual physical memory addresses can only happen at run time.

Bonus Features

What can you do if you can (selectively) gain control whenever a program reads or writes a particular virtual memory location?

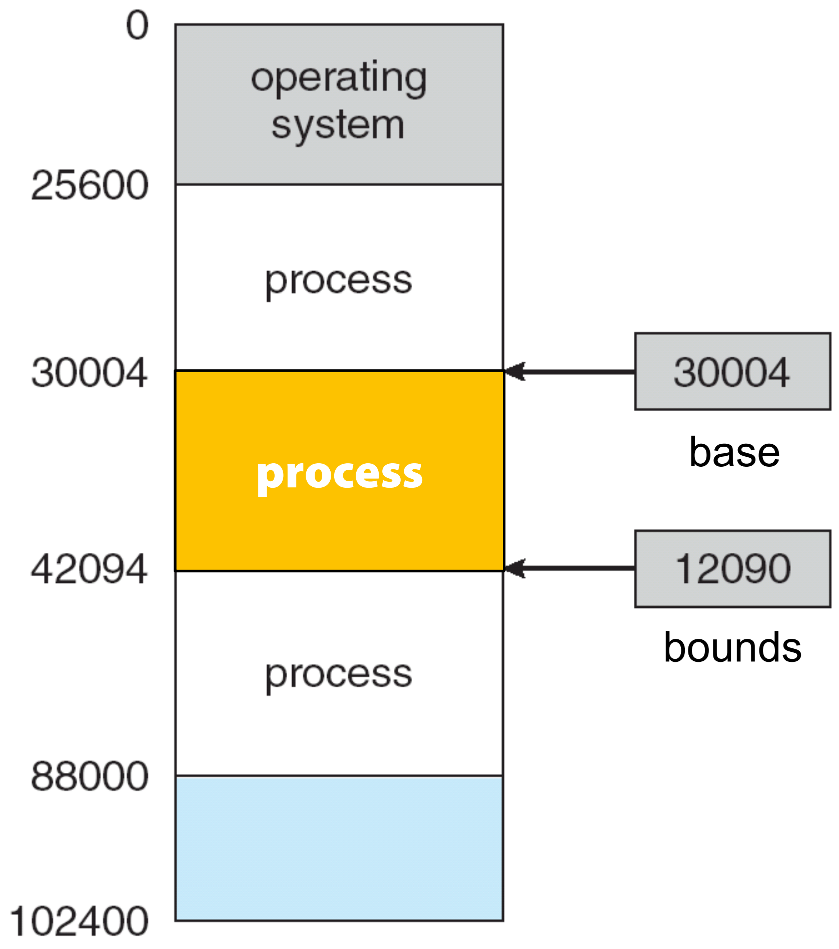
- Examples:
  - Copy-on-write
  - Zero-on-reference
  - Fill-on-demand
  - Demand paging
  - Memory mapped files
  - …

Basic Hardware Support

- Virtual address
  - Reference to a memory location independently of the actual assignment of data to memory.
  - Must be translated to a physical address.
- Relative address
  - Address expressed as a location relative to some known point.
- Physical address
  - Absolute address or actual location of a virtual address in main memory.

Basic Hardware Support

- Memory protection is provided via base and bounds (or base and limit) registers, which define the address space of a process.
  - **Base** holds the smallest legal address
  - **Bounds** specifies the size of the range.

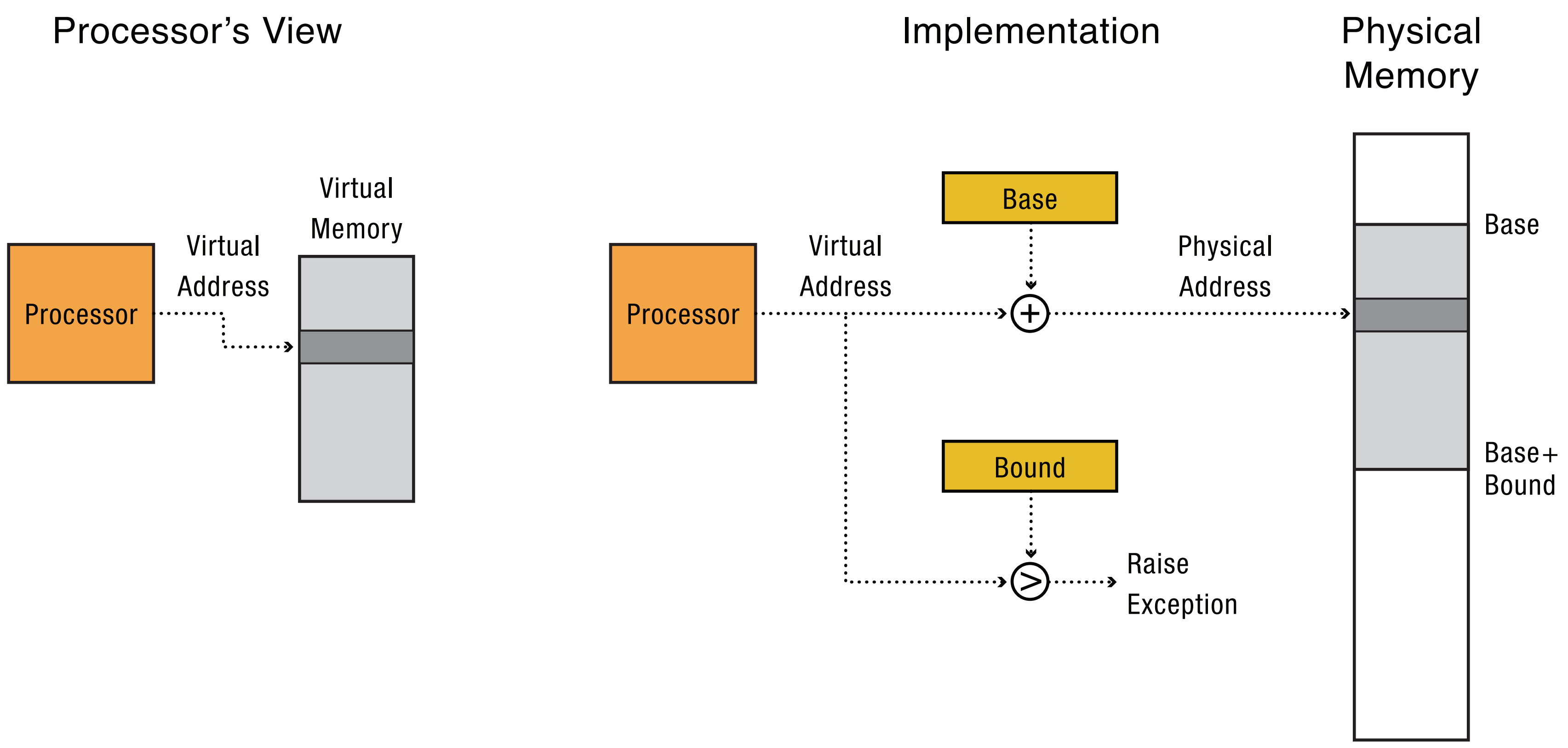


Virtually Addressed Base and Bounds

- Base register
  - Starting address for the process
- Bounds (or Limit) register
  - Ending location of the process or
  - Maximum allowed offset from the base address
- These values are set when the process is loaded or when the process is swapped in.

Virtually Addressed Base and Bounds

- The CPU hardware checks each address generated in user mode against the base-limit registers of the running process.



Question

With virtually addressed
base and bounds,
what is saved/restored on
a process context switch?

Virtually Addressed Base and Bounds

PROS?

- Simple
- Fast
  - 2 registers, adder, comparator
- Safe
- Can relocate in physical memory without changing process

CONS?

- Can't keep program from accidentally overwriting its own code
- Can't share code/data with other processes
- Can't grow stack/heap as needed

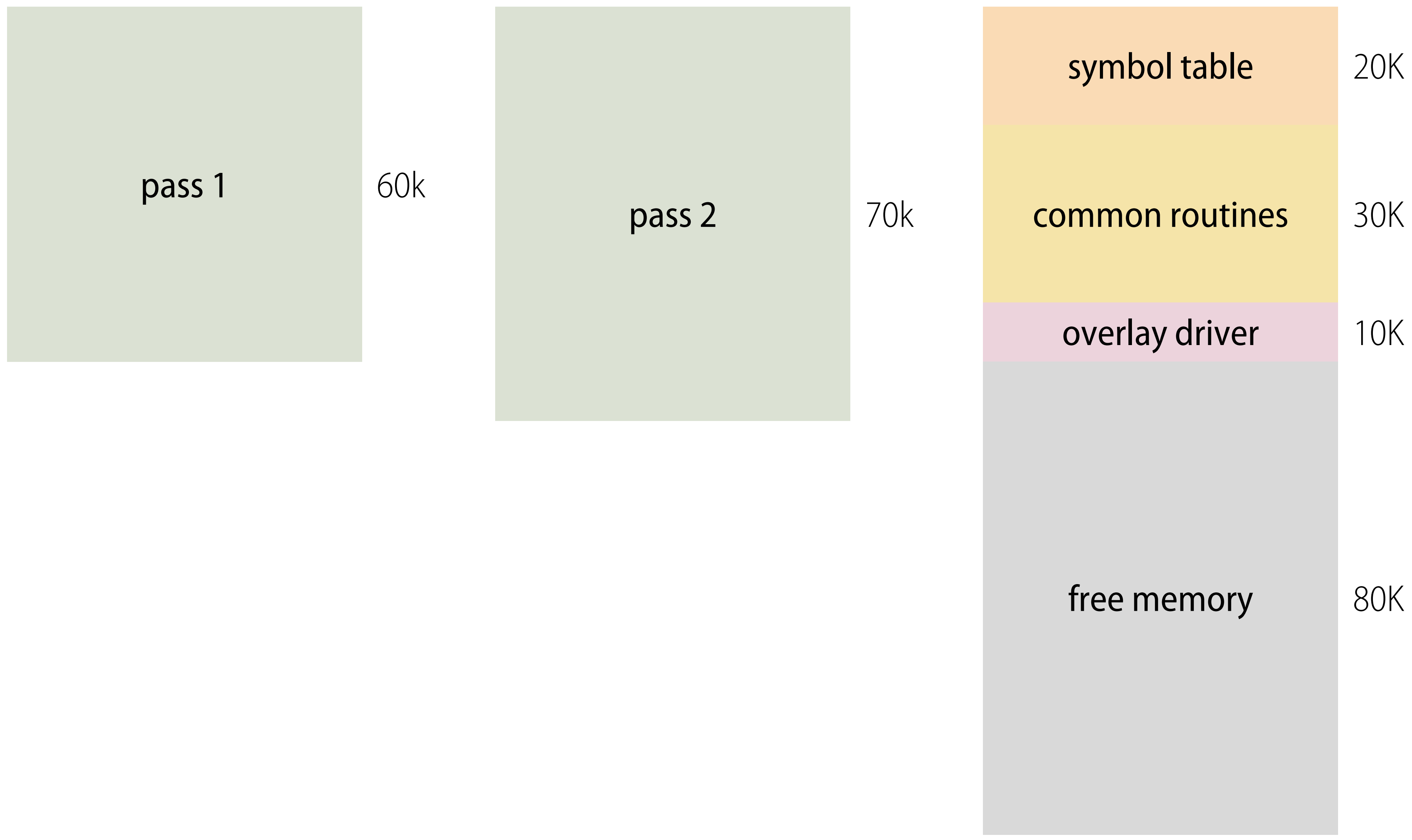
Address translation schemes

- Overlays
- Swapping
- Contiguous allocation
- Paging
- Segmentation
- Multi-level Translation

Overlays

- The earliest address translation scheme.
- Used when process is larger than the amount of memory allocated to it.
- Keep in memory only those instructions and data that are needed at any given time.
- Implemented by the user
  - No special support needed from operating system.
  - Programming design of overlay structure is complex.

Overlays for a Two-Pass Assembler



Swapping

- Used when the existing processes wouldn’t fit into the available memory at the same time.
- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.
- Backing store
  - Fast disk large enough to accommodate copies of all memory images for all users.
  - Must provide direct access to memory images.

Swapping

- Roll out, roll in
  - Swapping variant used for priority-based scheduling algorithms.
  - Lower-priority process is swapped out so higher-priority process can be loaded and executed.
- Major part of swap time is transfer time
  - Total transfer time is directly proportional to the amount of memory swapped.
- Modified (and more efficient) versions of swapping are found in most actual systems.

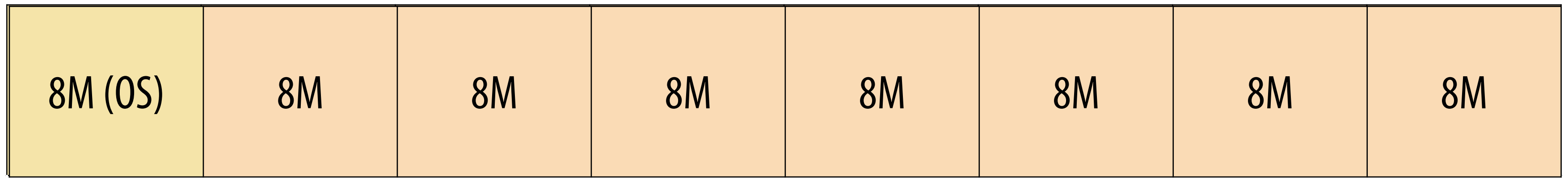
Contiguous Allocation

- Main memory broken into two regions:
  - Resident operating system (usually in low memory).
  - User processes (usually in high memory).
- Base-and-bounds register scheme used to protect user processes from each other, and from changing operating-system code and data.
  - Base register contains smallest physical address.
  - Bounds register contains range of virtual addresses.
  - Each virtual address must be less than the limit register.

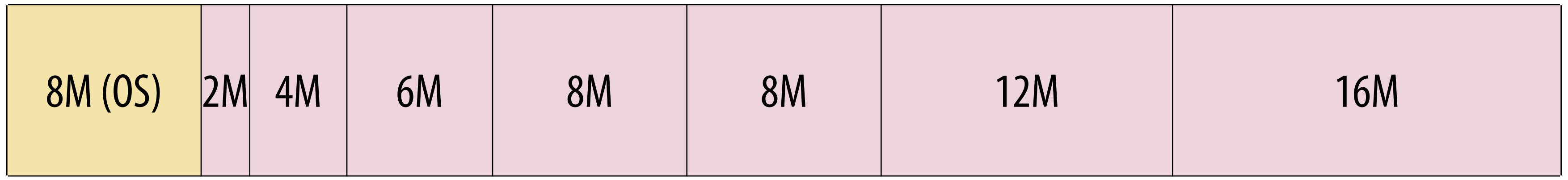
Partitioning techniques in continuous allocation

- User region further divided into either
  - Fixed-size partitions
    - Equal-size partitions
    - Unequal-size partitions
  - Variable-size partitions

Examples of fixed partitioning



Equal-size partitions



Unequal-size partitions

Fixed Partitioning

- Each partition may contain exactly one process.
  - Degree of multiprogramming bound by the number of partitions.
- When a partition is free one process is selected by the long-term scheduler from the input queue and loaded into it.
- When a process terminates, its partition becomes free and available to another process.

Fixed Partitioning

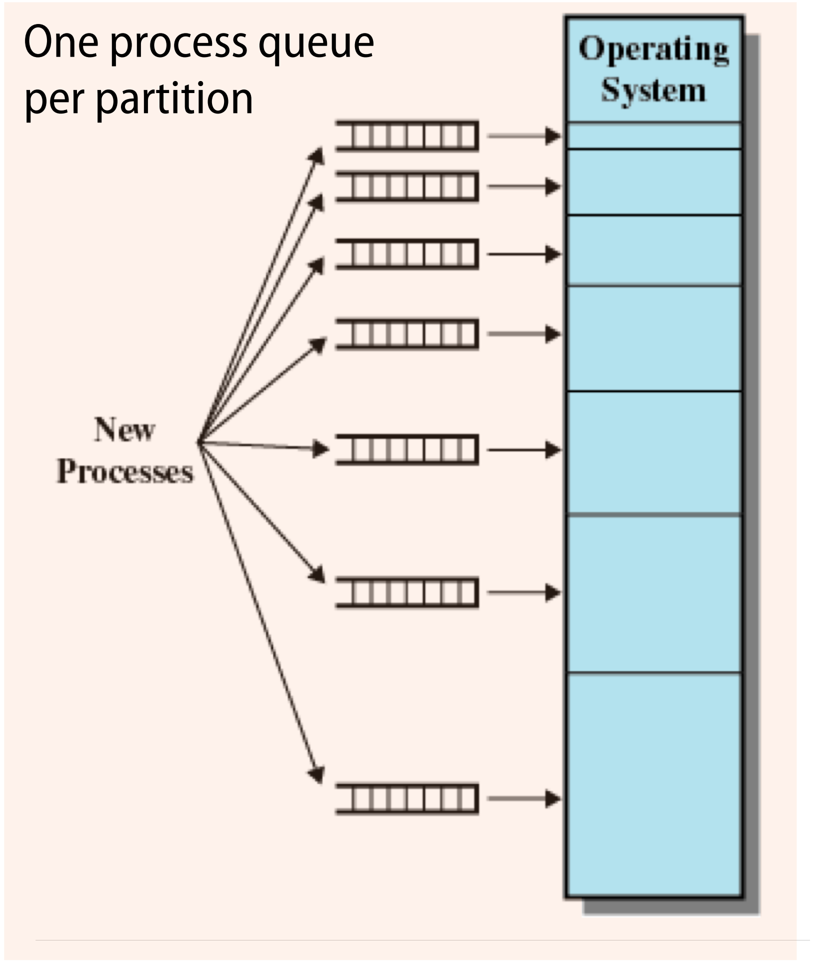
- Equal-size partitions
  - Any process whose size is less than or equal to the size of an available partition can be loaded into it.
  - If all partitions are full, the operating system can swap a process out of a partition.
  - A program may not fit in any partition.
    - The programmer must design the program with overlays.

Fixed Partitioning Placement Algorithms

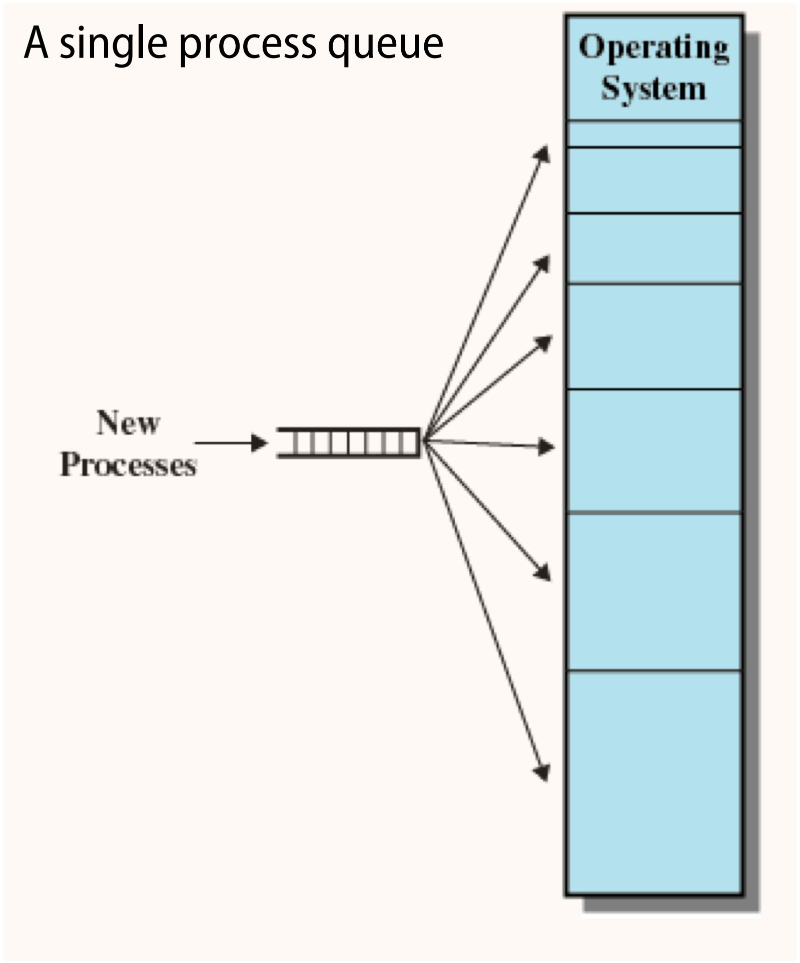
- Equal-size partitions
  - Since all partitions are of equal size, it does not matter which partition is used.
- Unequal-size partitions
  - Can assign each process to the smallest partition in which it will fit.
  - Queue for each partition.
  - Processes are assigned in such a way as to minimize wasted memory within a partition.

Memory Assigment in Unequal-size Fixed Partitioning

One process queue per partition



A single process queue



Fixed Partitioning Issues

Main memory use is inefficient.

Any program, no matter how small, occupies an entire partition.

- This sort of memory waste is called internal fragmentation.

Variable Partitioning

- Partitions are of variable length and number.
- Operating system maintains information about
  - Allocated partitions
  - Free partitions, i.e. blocks of available memory, called holes.
- When a process arrives, it is allocated exactly as much memory as required from a hole large enough to accommodate it.
  - The unused part of the selected hole becomes another (smaller) hole.

The Effect of Variable Partitioning

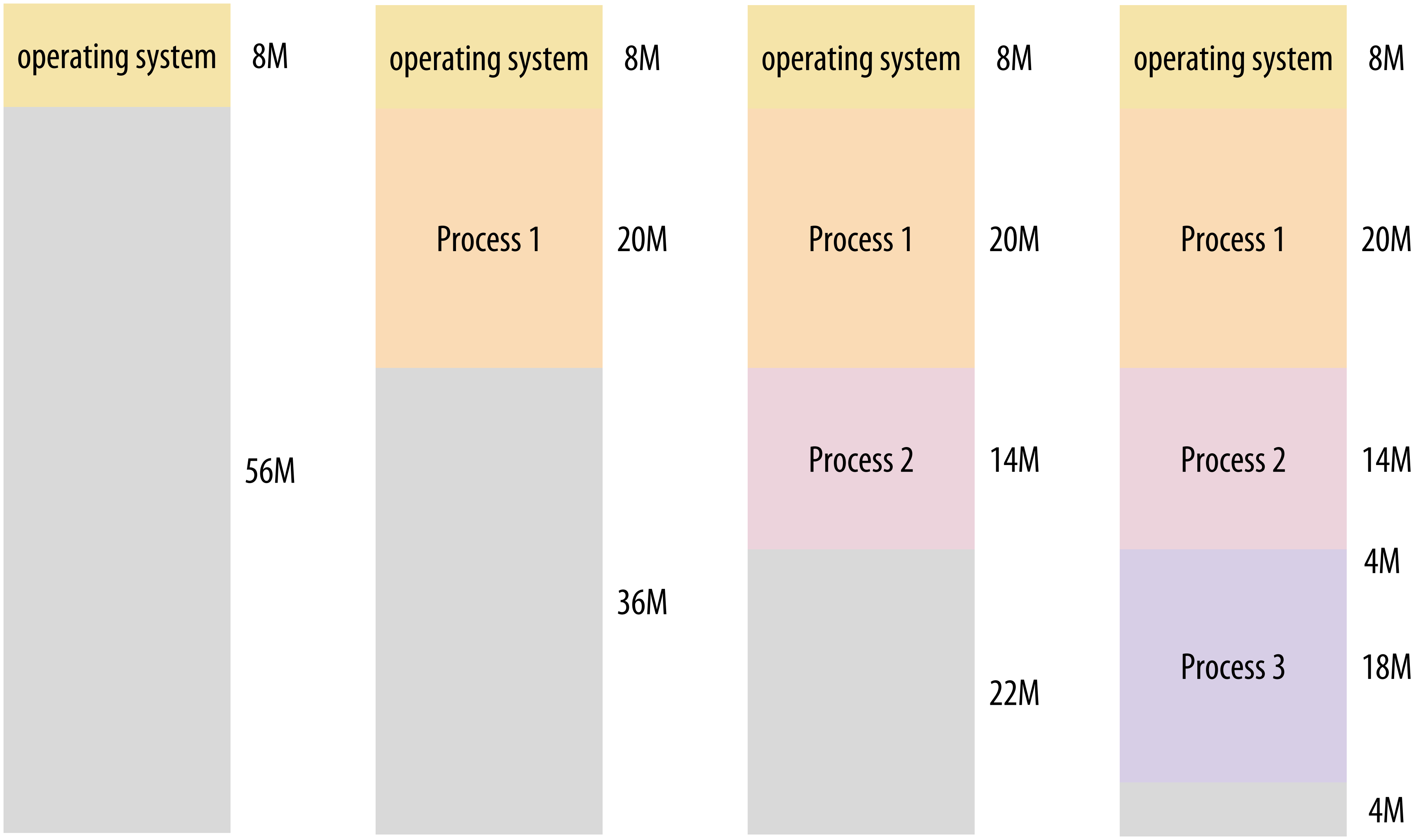
Due to the process allocation policy, holes of various sizes may appear and be scattered throughout memory.

This sort of memory waste is called external fragmentation.

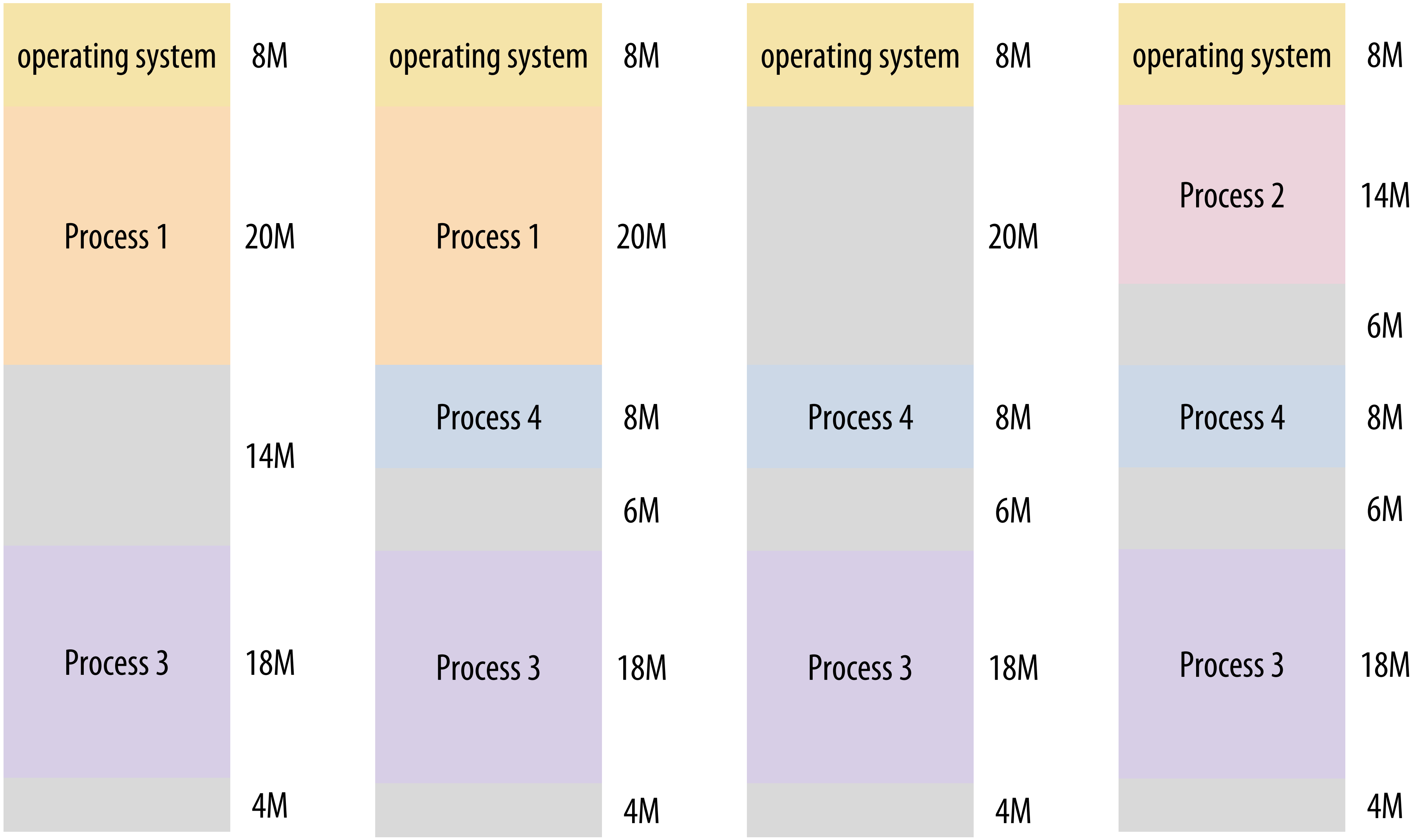
Compaction algorithm must be used to shift processes so they are contiguous and all free memory is in one block.

Contrary to internal fragmentation, external fragmentation can be treated.

The Effect of Variable Partitioning



The Effect of Variable Partitioning



Variable Partitioning Placement Strategies

- Operating system must decide which hole a process will be allocated to.
- Some common strategies for selecting a hole from the set of available holes are

Best-fit

First-fit

Next-fit

Variable Partitioning Placement Strategies

Best-fit

- Chooses the block that is closest in size to the request.
- Since smallest block is found for process,
  the smallest amount of fragmentation is left.
- Memory compaction must be done more often.
- Worst performer overall.

Variable Partitioning Placement Strategies

First-fit

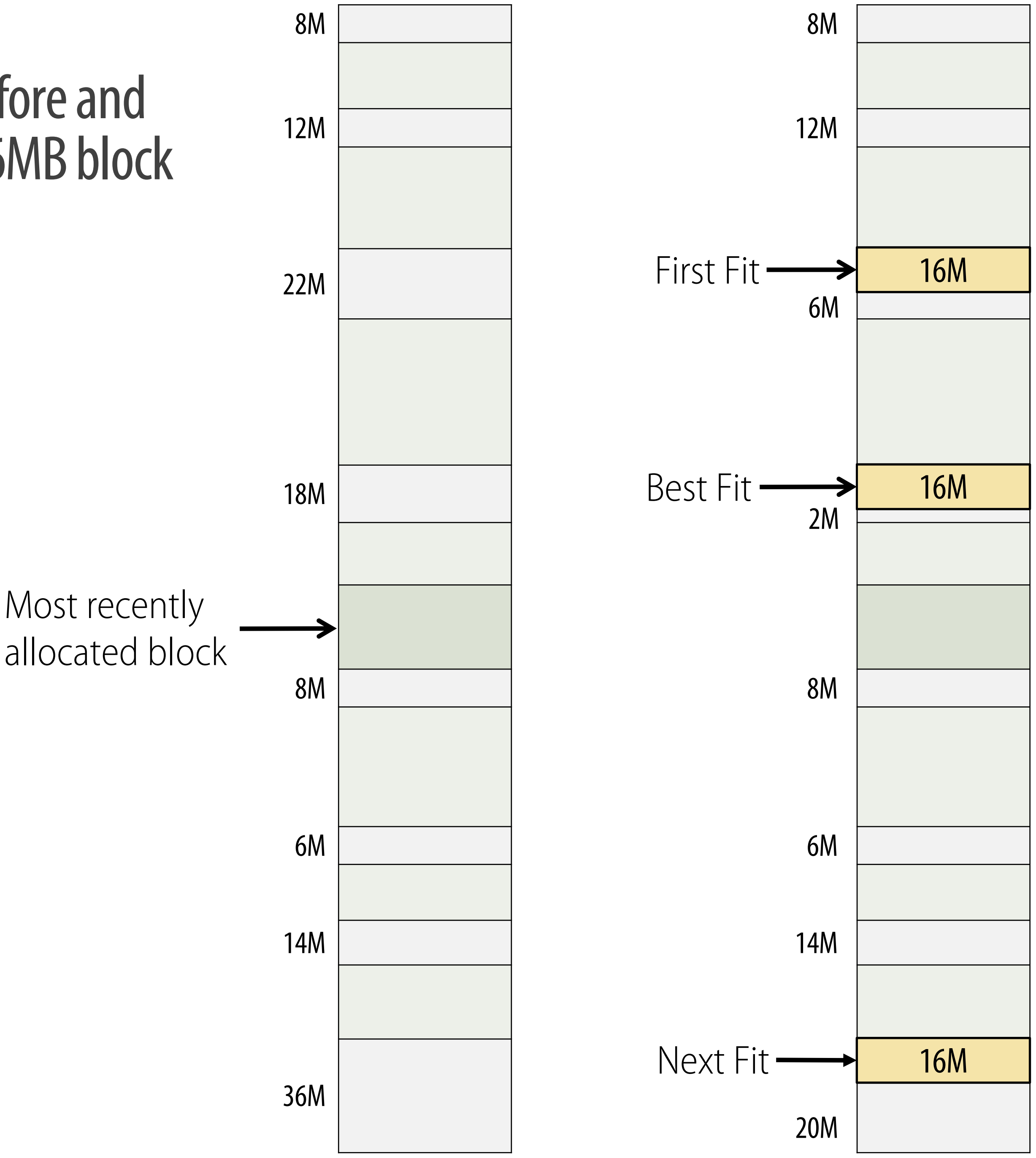
- Scans memory form the beginning and chooses the first available block that is large enough.
- May have many process loaded in the front end of memory that must be searched over when trying to find a free block.
- Fastest.

Variable Partitioning Placement Strategies

Next-fit

- Scans memory from the location of the last placement.
- More often allocates a block of memory at the end of memory where the largest block is found.
- The largest block of memory is broken up into smaller blocks.
- Compaction is required to obtain a large block at the end of memory.

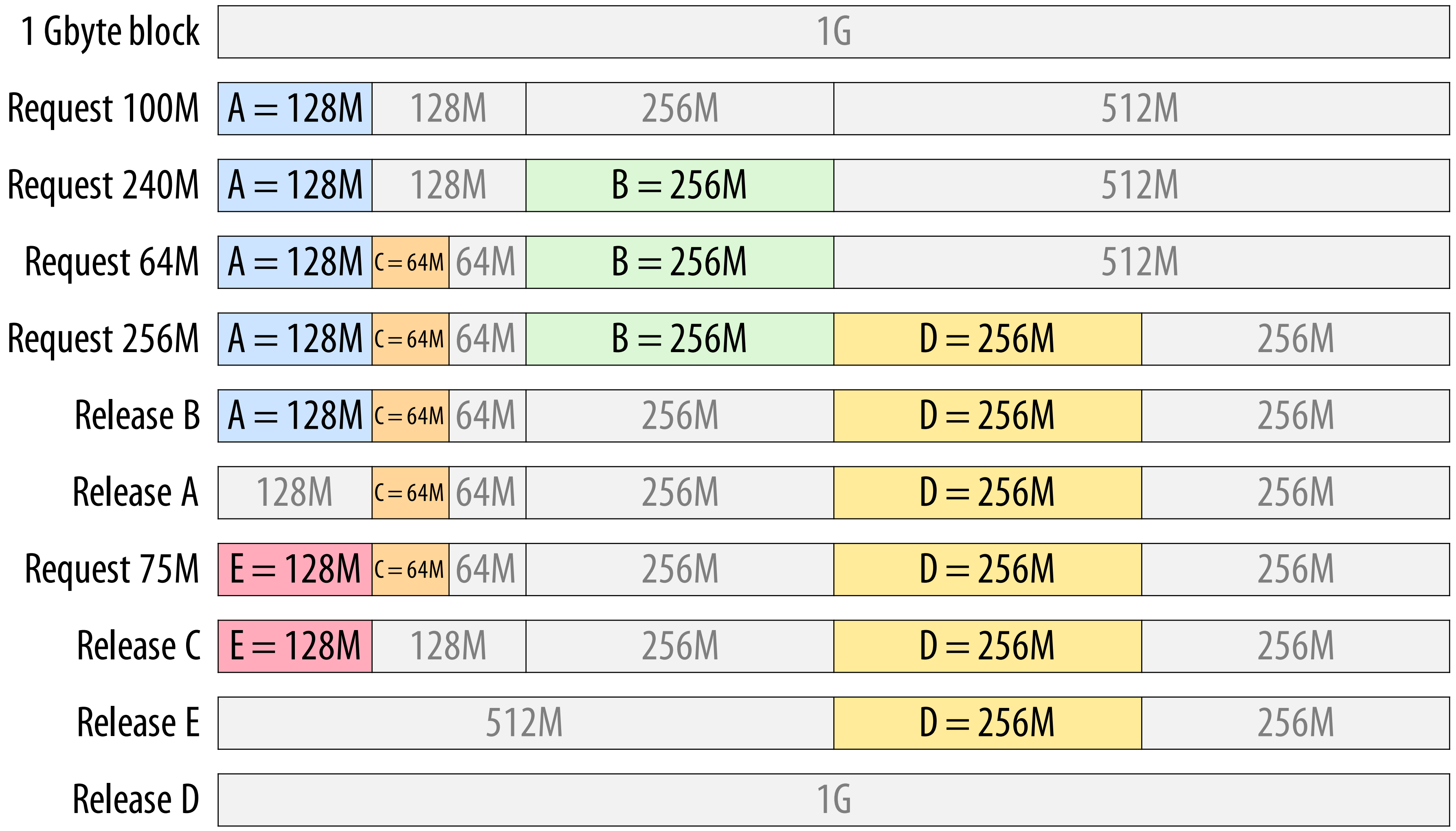
Memory configuration before and after the allocation of a 16MB block



Buddy System

- The Buddy system is a compromising partitioning strategy
- Entire space available is treated as a single block of size 2^U.
- When a request of size s is made …
  - If $2^{U-1} < s \leq 2^U$, the entire block is allocated.
  - Otherwise, block is split into two equal buddies.
  - Process continues on the lower-address buddy until smallest block with size at least equal to s is generated.

Buddy System: Example



Tree Representation of a Buddy System

1 Gbyte block

Request 100M

Request 240M

Request 64M

Request 256M

Release B

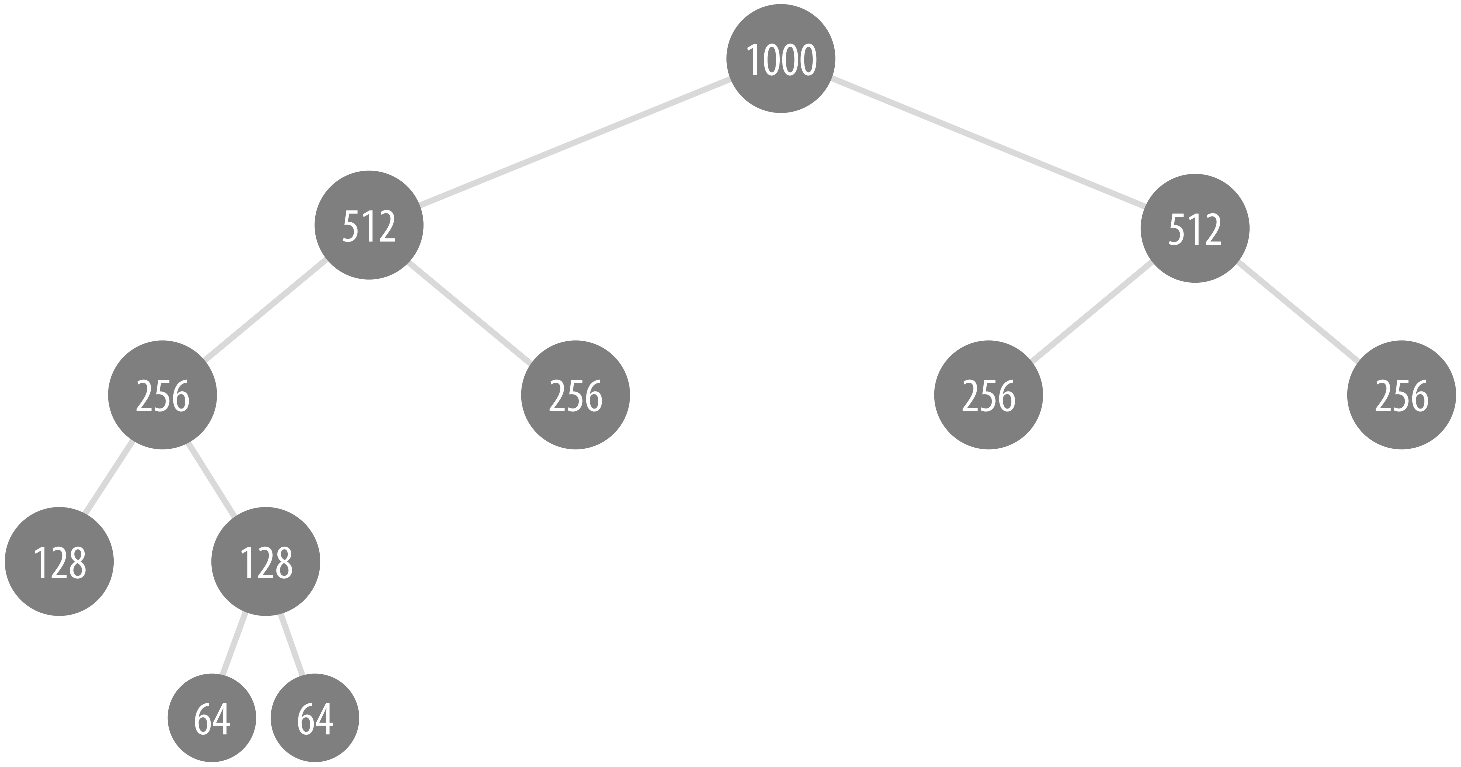
Release A

Request 75M

Release C

Release E

Release D



Memory Fragmentation Issues and (Possible) Solutions

External fragmentation

- Memory is enough to satisfy a request, but it is not contiguous.
- Can be reduced by compaction
  - Shuffle memory contents to put all free space in a single block.
  - Compaction is possible only if address translation is dynamic and done at execution time.
- To avoid the I/O problem
  - Latch job in memory while it is involved in I/O.
  - Do I/O only into OS buffers.

Internal fragmentation

- Allocated memory may be larger than requested memory.
- This excessive memory is internal to a partition and cannot be used.