Memory Management

By

Lecturer: Ameen A.Noor

Memory Management Strategies

- These strategies are designed to obtain the best possible use of main memory. They are divided into:

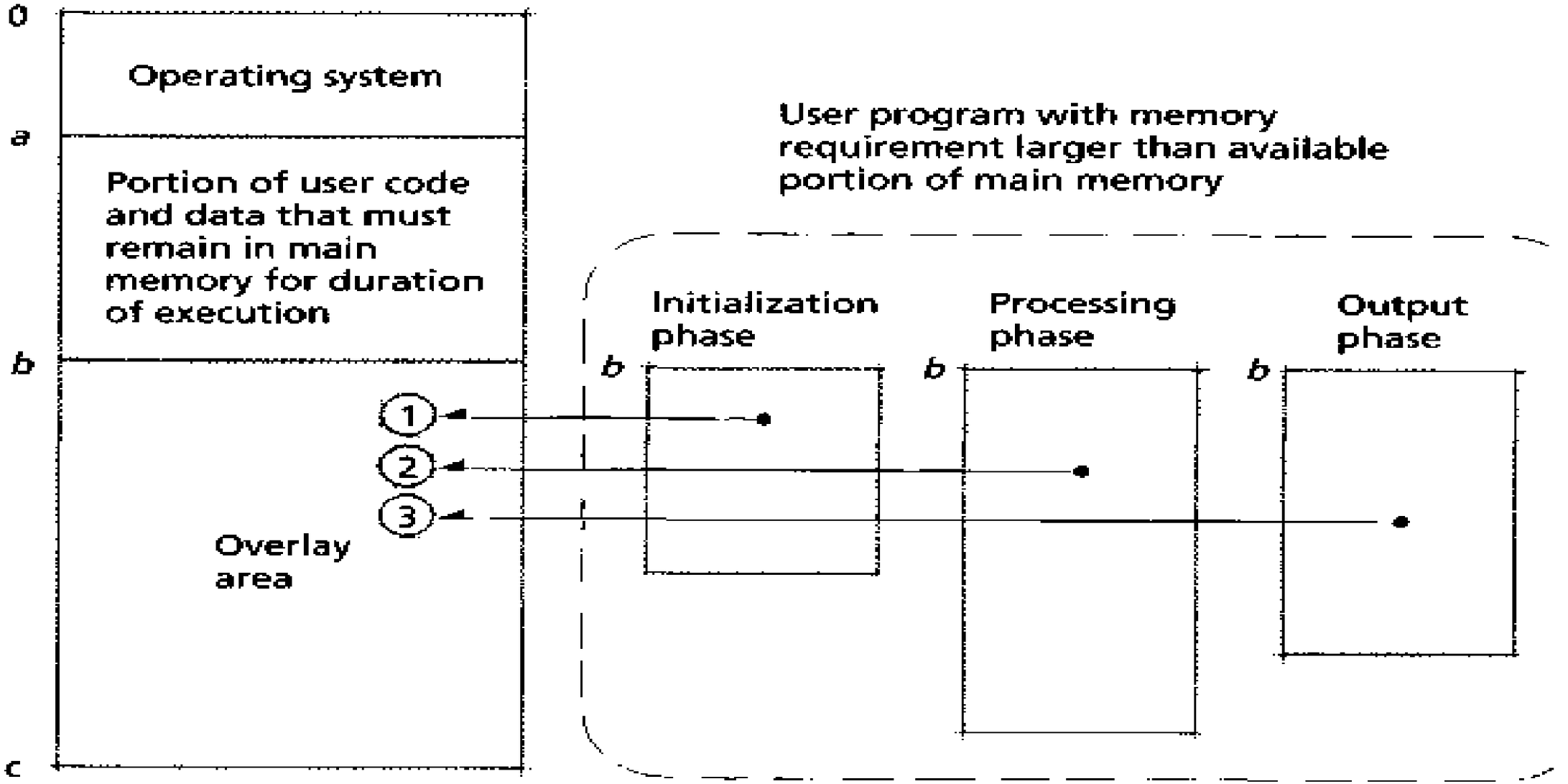
1. Fetch strategies: determine when to move the next piece of a program or data to main memory from secondary storage. We divide them into:
   - demand fetch strategy.
   - anticipatory fetch strategy.
2. Placement strategies: determine where in main memory the system should place incoming program or data pieces, first fit, best fit, and worst fit.
3. Replacement strategies: when memory is too full to accommodate a new program, the system must remove some (or all) of a program or data that currently resides in memory.

Contiguous Vs. Noncontiguous Memory Allocation

- **Contiguous Memory Allocation:** to execute a program in early computer systems, the system operator or the operating system had to find enough contiguous main memory to accommodate the entire program. If the program was larger than the available memory then the system could not execute it.
- **Non-Contiguous Memory Allocation:** a program is divided into blocks or segments that the system may place in non-adjacent slots in main memory. This allows making use of holes (unused gaps) in memory that would be too small to hold whole programs.

Overlays

- One way in which a software designer could overcome the memory limitations was to create overlays, which allowed the system to execute programs larger than main memory.

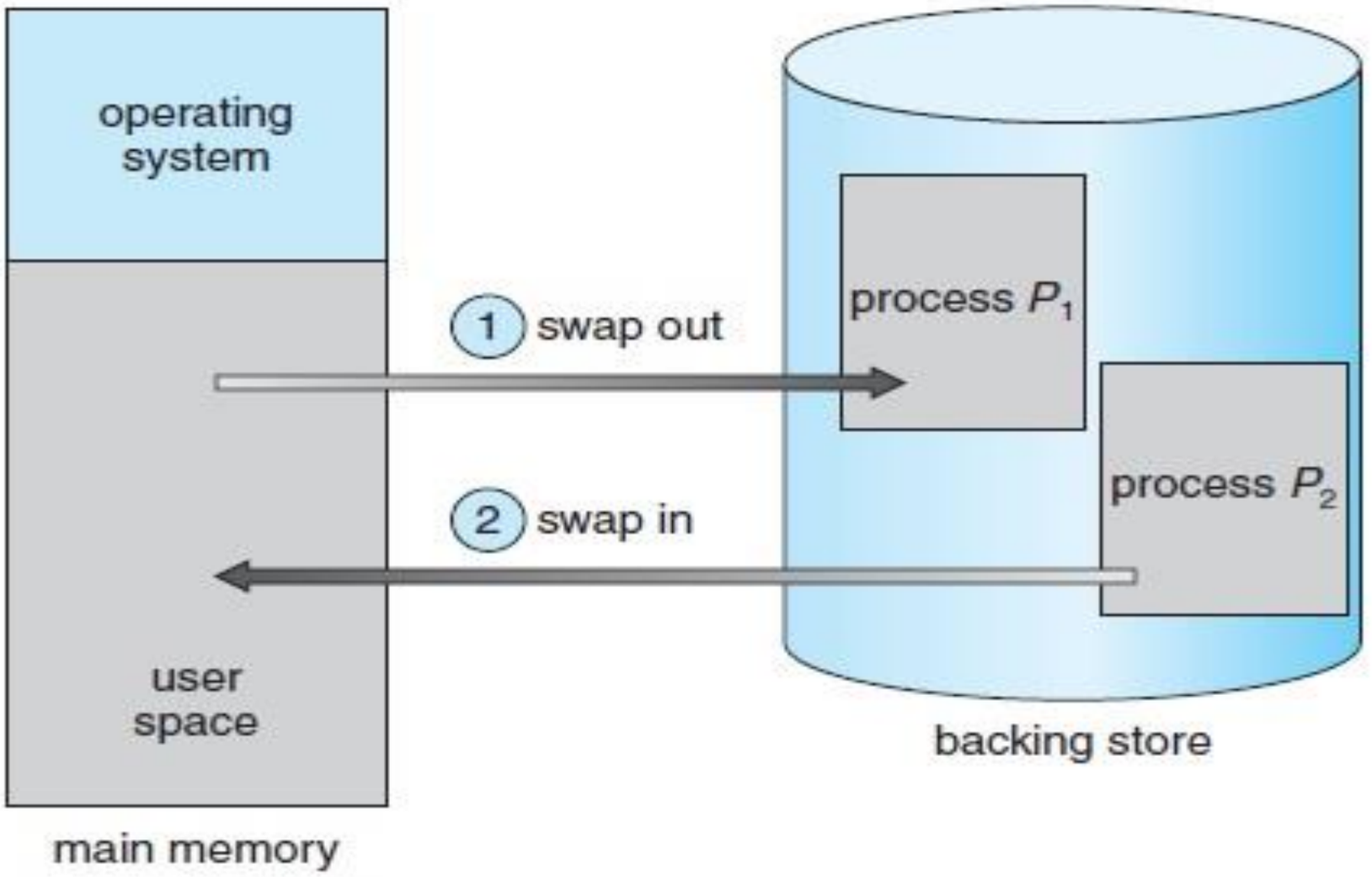


1. Load initialization phase at *b* and run.
2. Then load *processing phase* at *b* and run.
3. Then load output phase at *b* and run.

Swapping

- 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 these memory images
- **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 versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

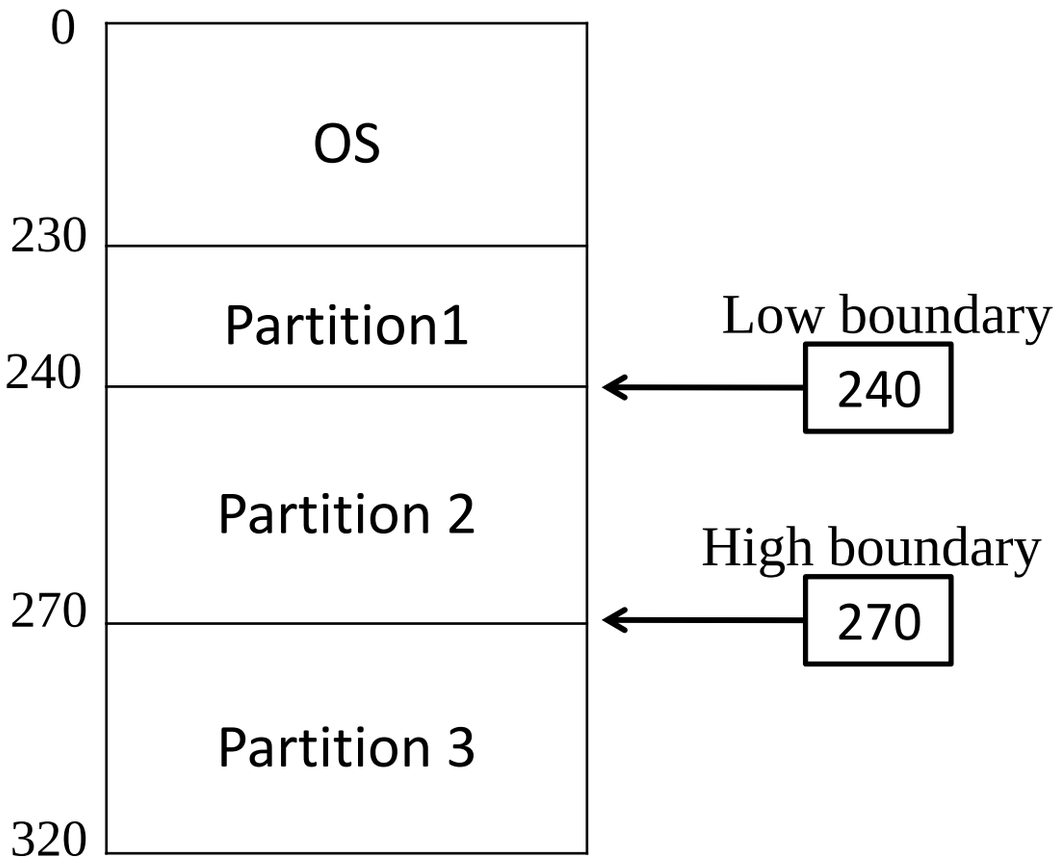
Swapping


operating system
user space
main memory
1 swap out
2 swap in
process P_1
process P_2
backing store

Protection in a Single-User System

- Without protection, the process may alter the operating system.
- The protection can be implemented with a single boundary register built into the processor.
- The boundary register contains the memory address at which the user's program begins.
- Each time a process references a memory address, the system determines if the request is for an address greater than or equal to that stored in the boundary register.
  - If so, the system services the request.
  - If not, then the program is trying to access the operating system.
- The system intercepts the request and terminates the process with an appropriate error message.

Fixed-Partition

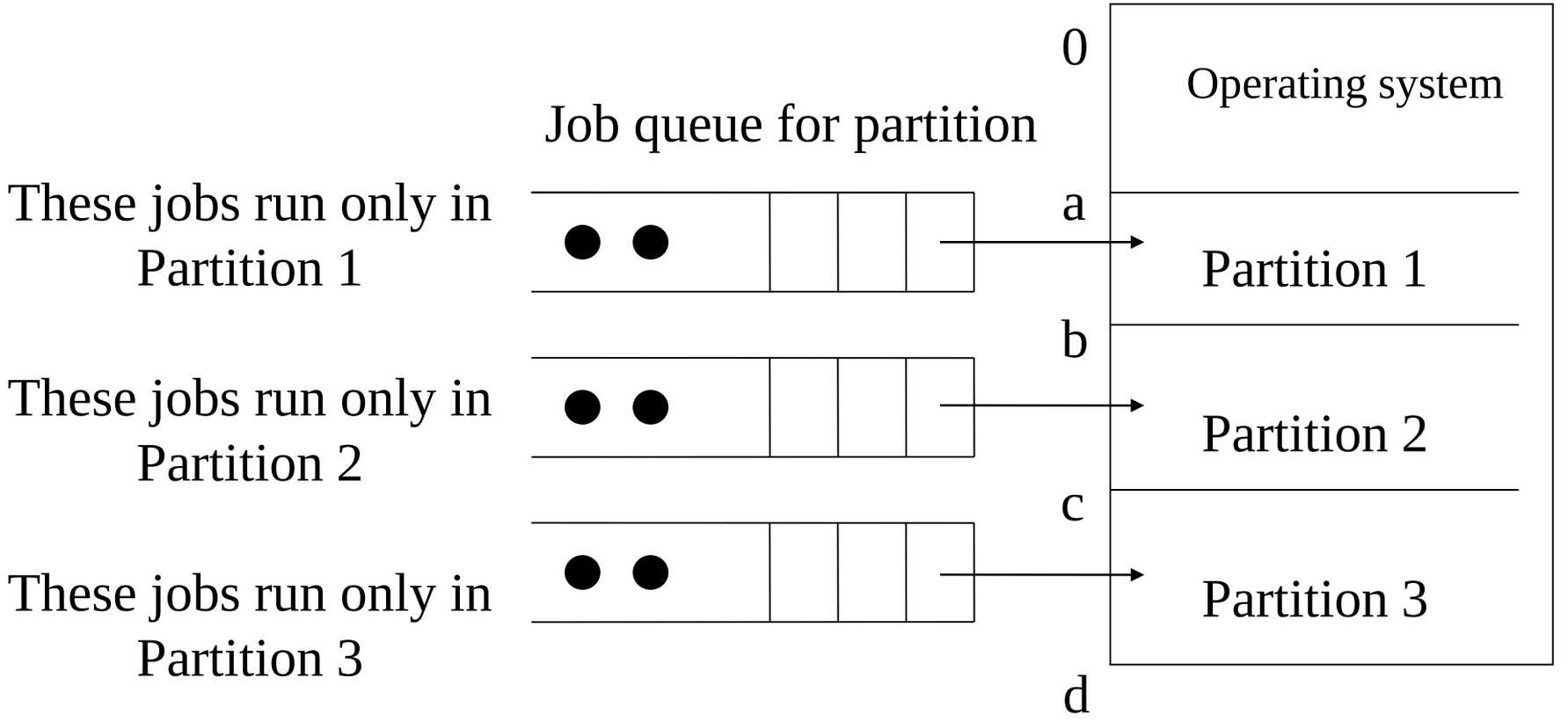


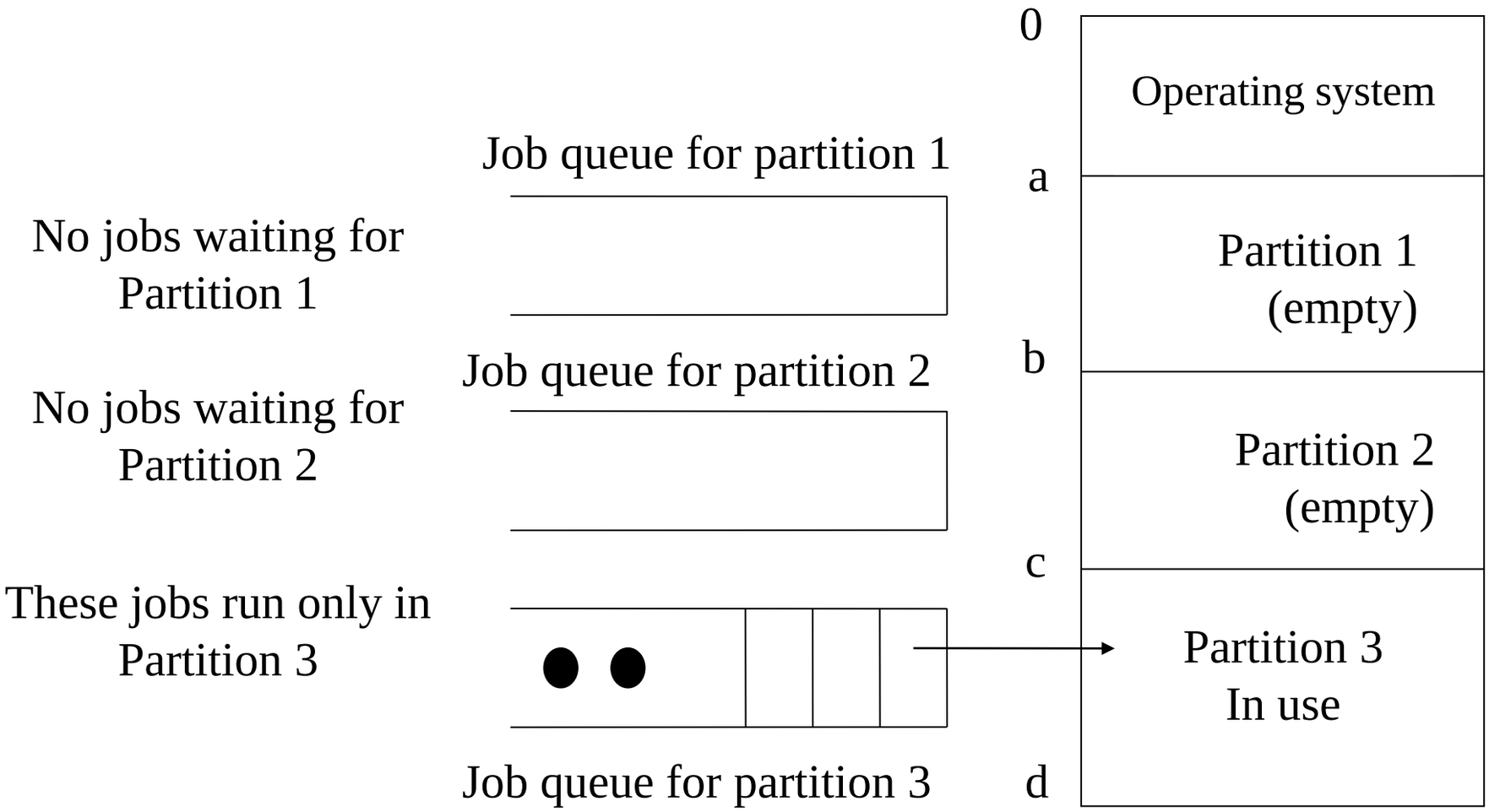
How the Memory Manager works

Partition	Size of Partition	Begin	Use
Partition 1	10	230	Empty
Partition 2	30	240	Empty
Partition 3	50	270	Busy

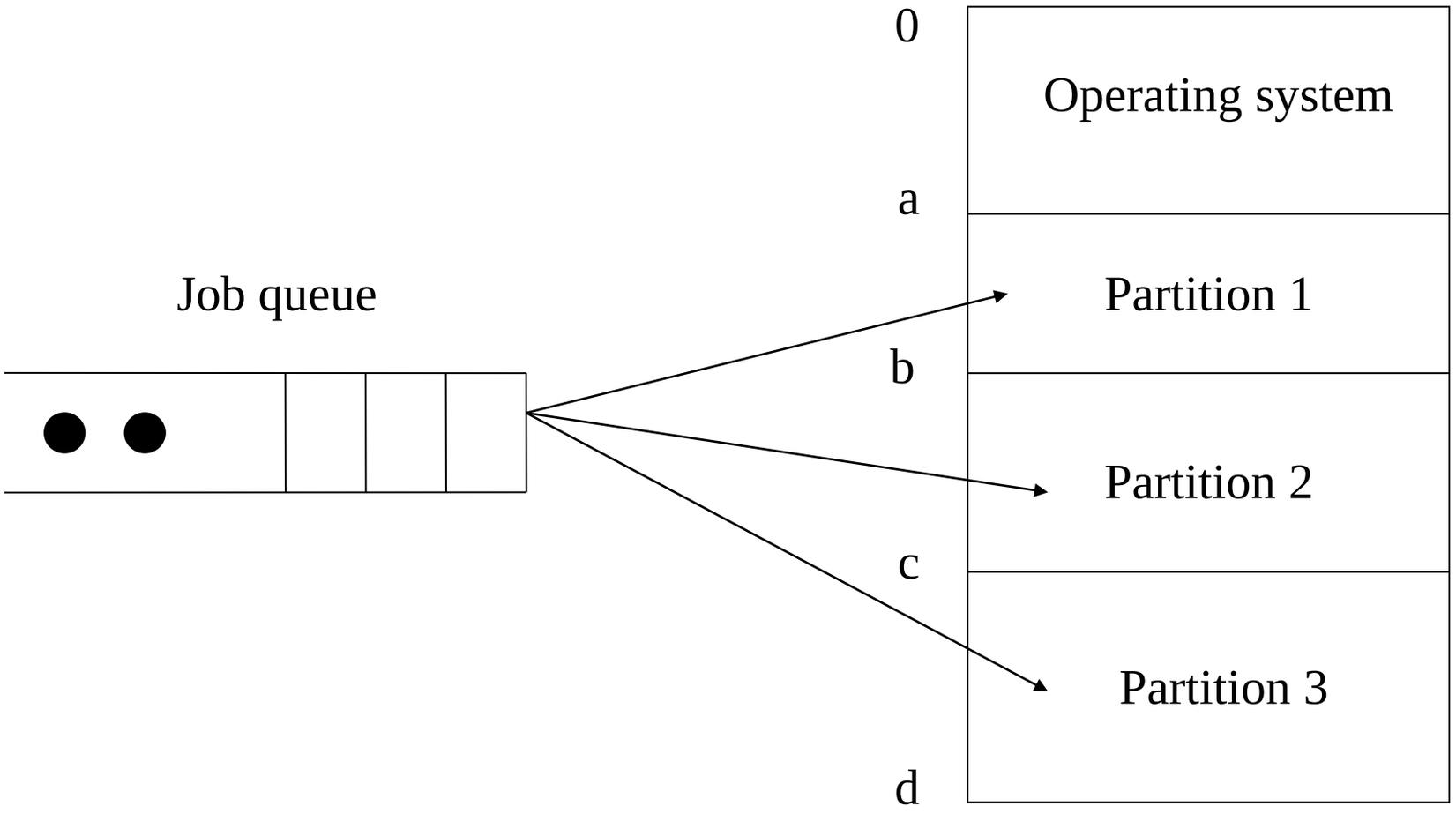
Fixed-Partition Multiprogramming

- The earliest multiprogramming systems used fixed partition multiprogramming.
- The system divides main memory into a number of fixed size partitions each partition holds a single job.
- In the earliest multiprogramming systems, the programmer translated a job using an absolute assembler or compiler.
- It meant that a job had its precise location in memory determined before it was launched and could run only in a specific partition.
- If the programs partition was occupied then that job had to wait even if other partitions were available.

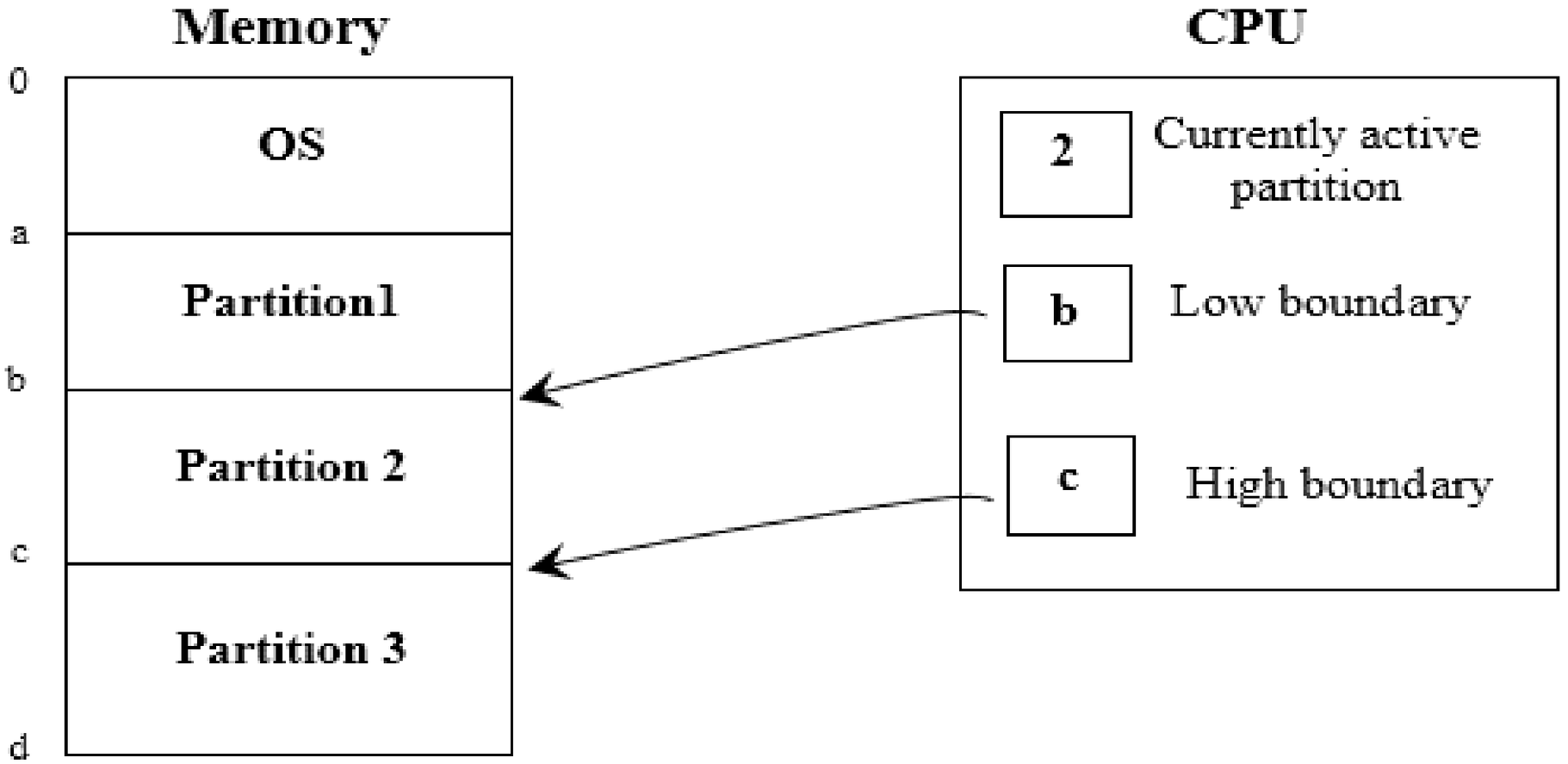

0
Operating system
Job queue for partition
These jobs run only in Partition 1
a
Partition 1
b
These jobs run only in Partition 2
Partition 2
c
These jobs run only in Partition 3
Partition 3
d


0
Operating system
Job queue for partition 1
a
No jobs waiting for Partition 1
Partition 1 (empty)
Job queue for partition 2
b
No jobs waiting for Partition 2
Partition 2 (empty)
c
These jobs run only in Partition 3
Partition 3 In use
Job queue for partition 3
d

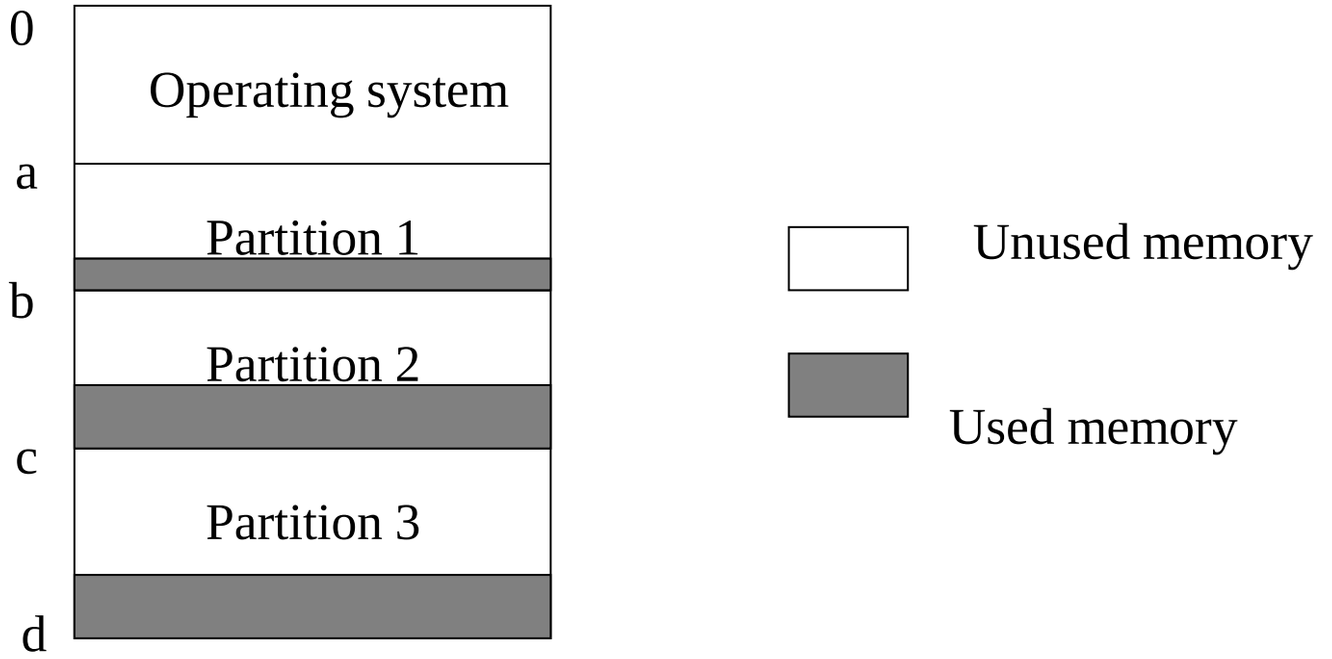
To overcome the problem, the developers created relocating compilers, assemblers and loaders. These tools produce a Relocatable program that can run in any available partition that is large enough to hold that program.



Protection often is implemented with multiple boundary register. The system can delimit each partition with two boundary registers low and high, also called base and limit registers.

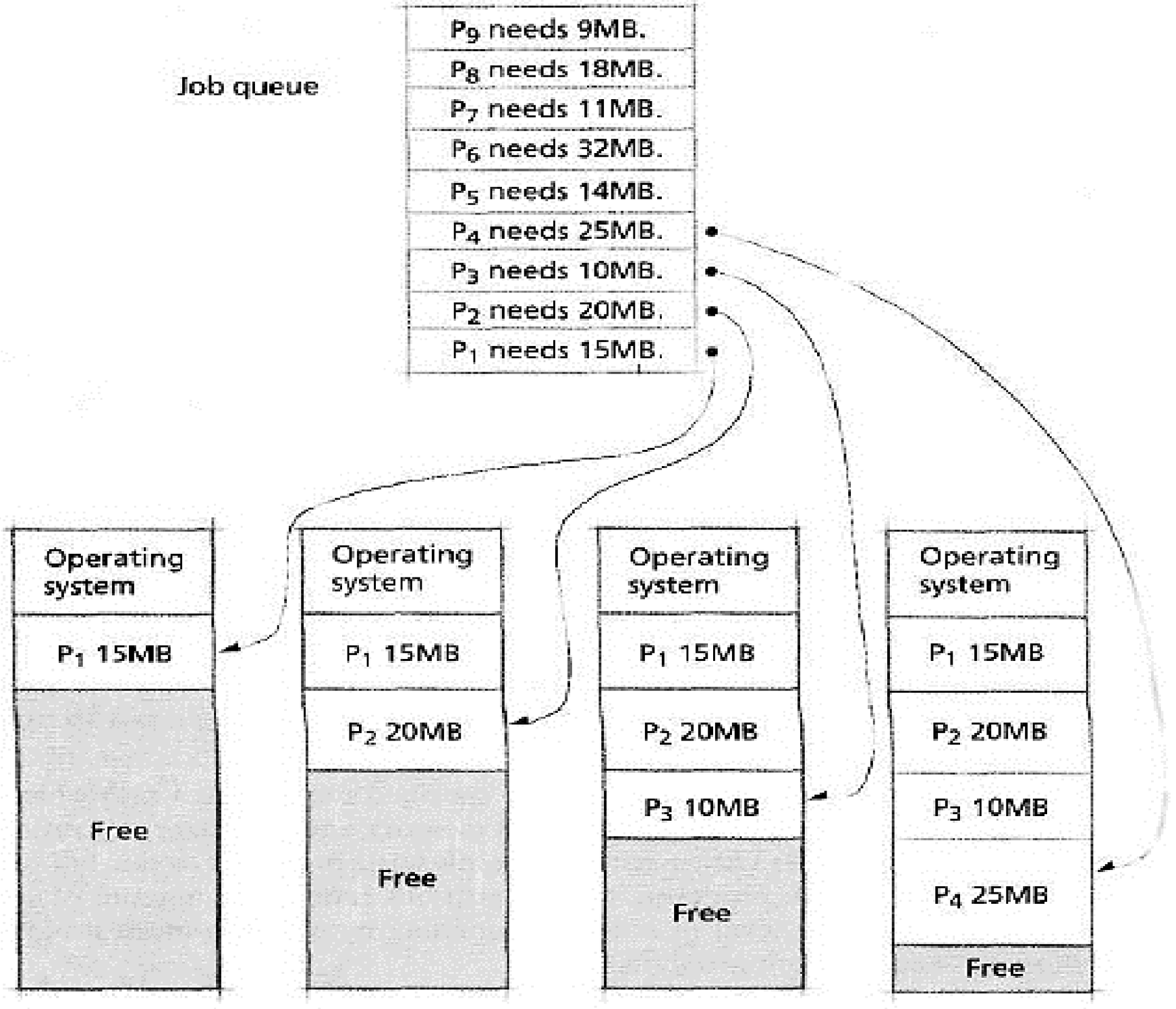


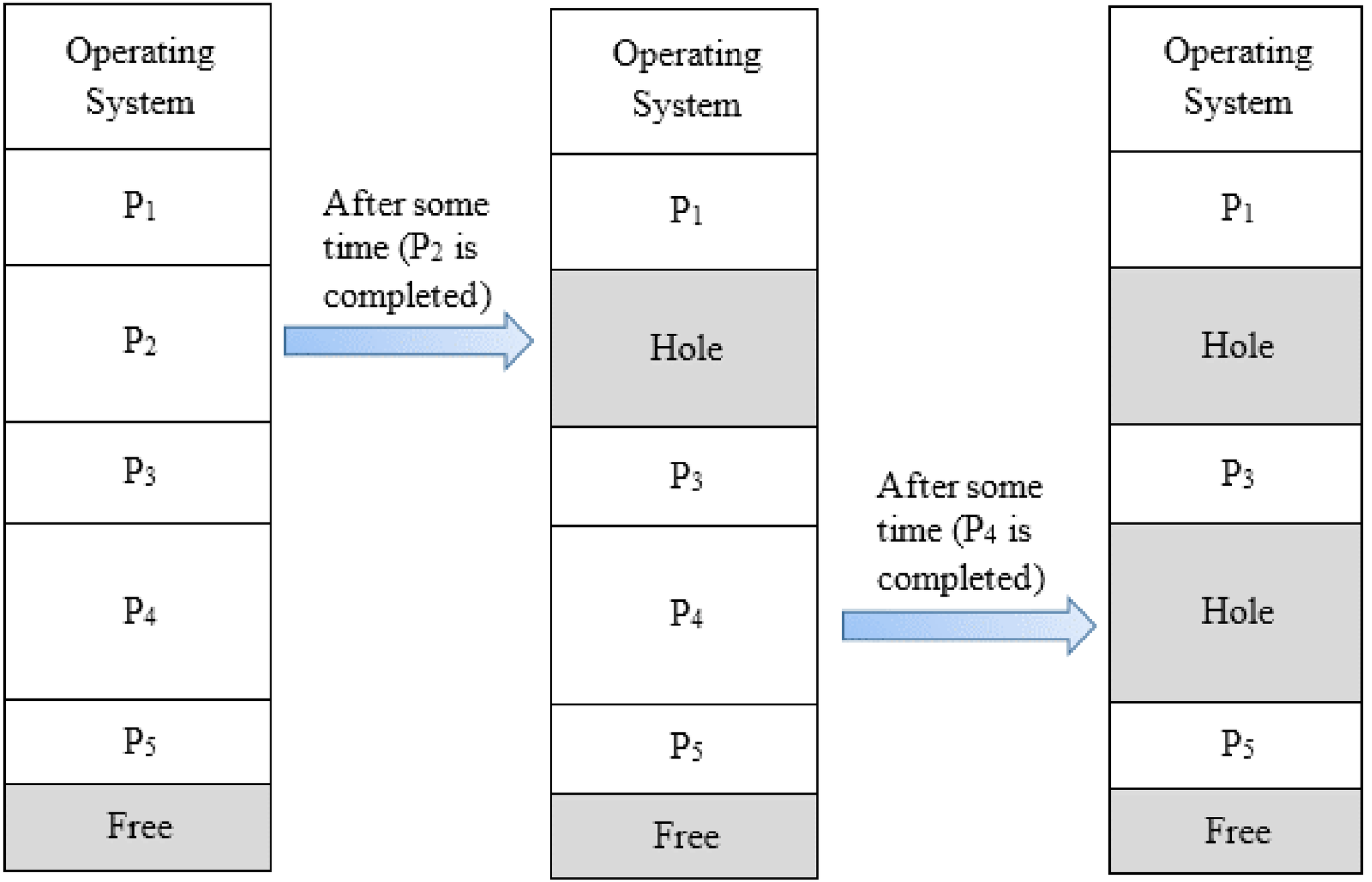
Fixed partition multiprogramming suffers from internal fragmentation, which occurs when the size of a process's memory and data is smaller than that of the partition in which the process executes.



Variable-Partition Multiprogramming

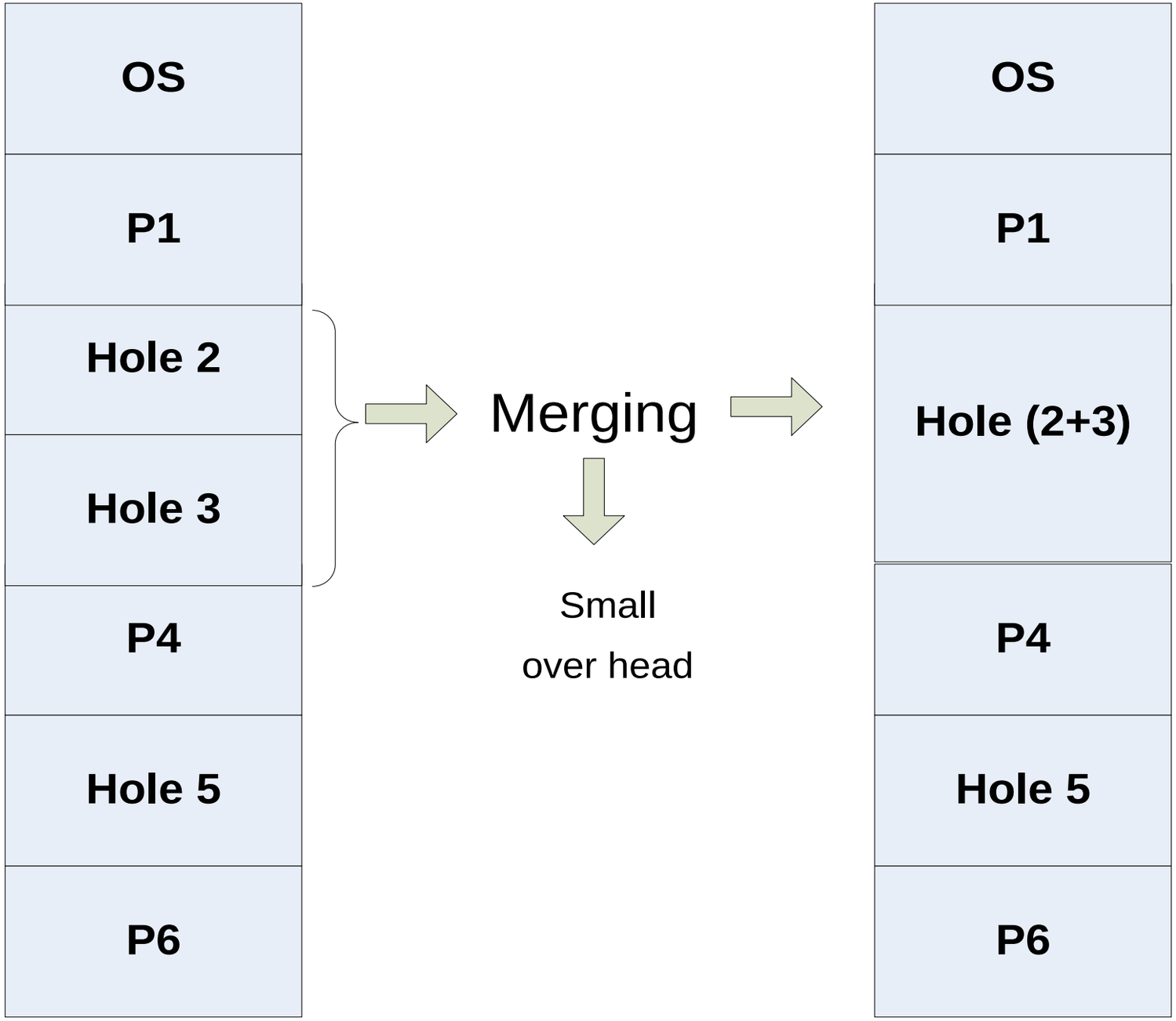
- Fixed-partition multiprogramming imposes restrictions on a system that result in inefficient resource use.
- For example, a partition may be too small to accommodate a waiting process, or so large that the system loses considerable resources to internal fragmentation.
- An obvious improvement, operating system designers decided, would be to allow a process to occupy only as much space as needed (up to the amount of available main memory). This scheme is called variable-partition multiprogramming.


Job queue
P9 needs 9MB.
P8 needs 18MB.
P7 needs 11MB.
P6 needs 32MB.
P5 needs 14MB.
P4 needs 25MB.
P3 needs 10MB.
P2 needs 20MB.
P1 needs 15MB.
Operating system
P1 15MB
Free
Operating system
P1 15MB
P2 20MB
Free
Operating system
P1 15MB
P2 20MB
P3 10MB
Free
Operating system
P1 15MB
P2 20MB
P3 10MB
P4 25MB
Free


Operating System
P1
P2
P3
P4
P5
Free
After some time (P2 is completed)
Operating System
P1
Hole
P3
P4
P5
Free
After some time (P4 is completed)
Operating System
P1
Hole
P3
Hole
P5
Free

Variable-Partition Multiprogramming

- The process of merging adjacent holes to form a single, larger hole is called **coalescing** (merge things). The system reclaims the largest possible contiguous blocks of memory.

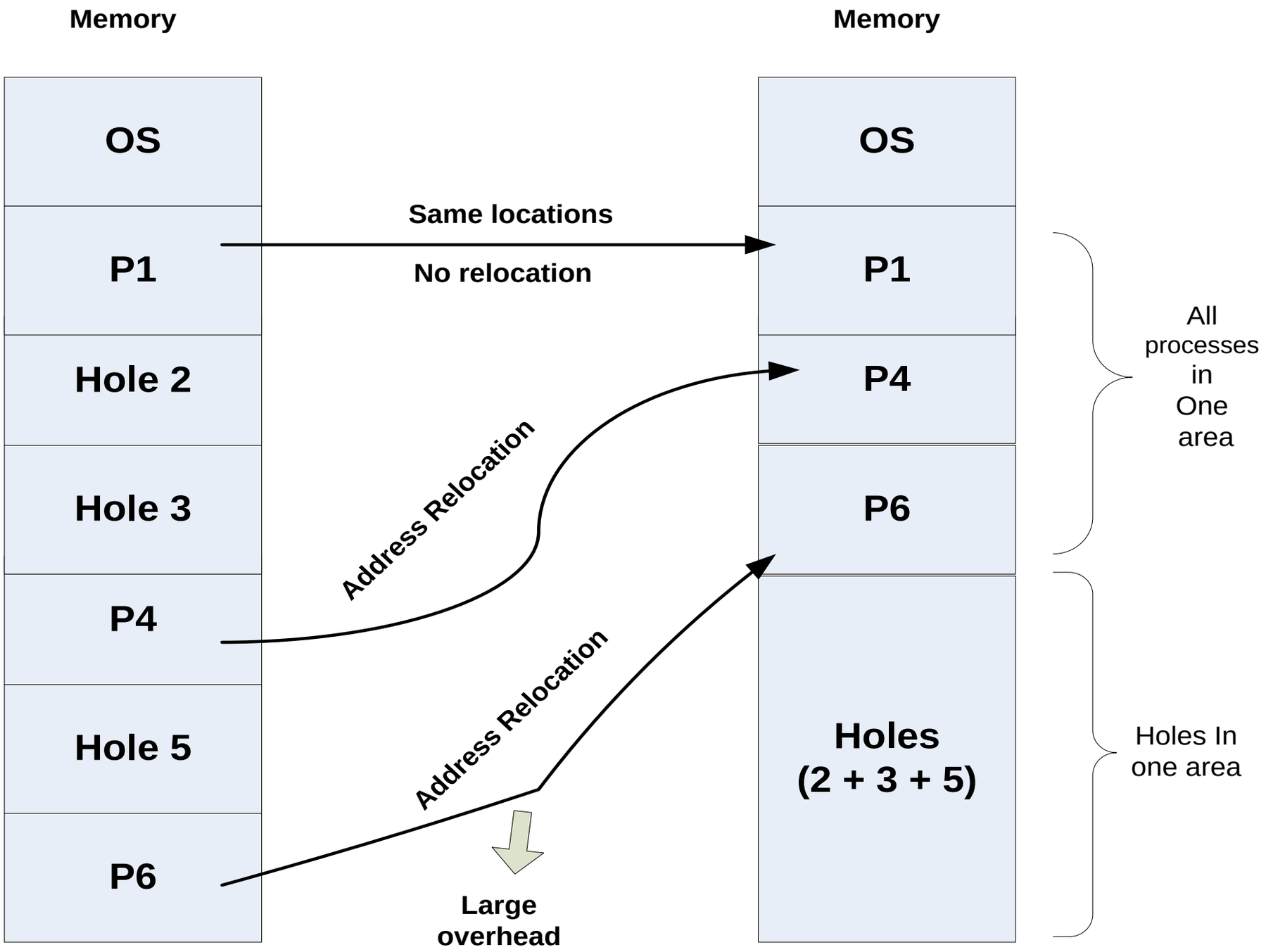

OS
P1
Hole 2
Hole 3
P4
Hole 5
P6
Merging
Small
over head
OS
P1
Hole (2+3)
P4
Hole 5
P6

Variable-Partition Multiprogramming

- Another technique for reducing external fragmentation is called **memory compaction (defragmentation)**.
- This relocates all occupied areas of memory to one end of main memory. Now all of the available free memory is contiguous.
- The drawbacks are:

  - Overhead consumes system resources.

  - The system must cease (stop something) all other computation during compaction which results in erratic response times for interactive users.


Memory
Memory
OS
P1
Hole 2
Hole 3
P4
Hole 5
P6
Same locations
No relocation
Address Relocation
Address Relocation
Large
overhead
OS
P1
P4
P6
Holes
(2 + 3 + 5)
All
processes
in
One
area
Holes In
one area

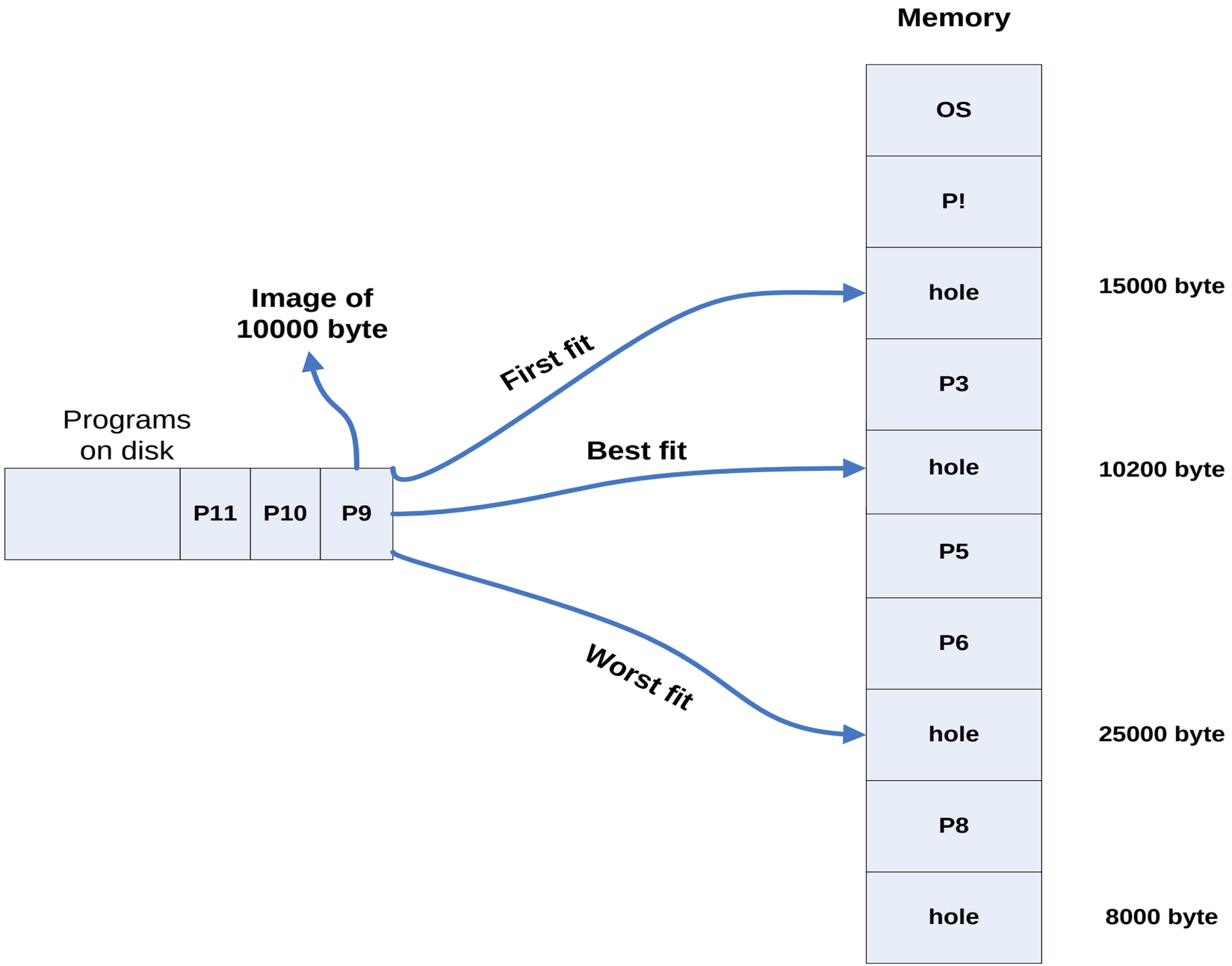
Memory Placement Strategies

- Determines where in main memory to place incoming programs and data. The main strategies are:

  - Best fit: place the job in the smallest possible hole. The disadvantage is that the rest of hole will not be enough for new job.

  - First fit: place the job in the first suitable hole. The advantage is low overhead i.e. small CPU wasted time in implementing the strategy.

  - Worst fit: place the job in the largest available hole. The rest of hole may be still enough for new job.


Memory
OS
P!
hole
15000 byte
P3
hole
10200 byte
P5
P6
hole
25000 byte
P8
hole
8000 byte
Image of
10000 byte
Programs
on disk
P11
P10
P9
First fit
Best fit
Worst fit

Fragmentation

- External Fragmentation: total memory space exists to satisfy a request, but it is not contiguous

- Reduce external fragmentation by compaction

  - shffle memory contents to place all free memory together in one large block

  - Compaction is possible only if relocation is dynamic, and is done at execution time.

  - I/O problem

    @ Latch job in memory while it is involved in I/O

    @ Do I/O only into OS buffers

Fragmentation

- Internal Fragmentation: allocated memory may be slightly larger than requestedmemory, this size difference is memory internal to a partition, but not being used

Equal size partitioning

unequal size partitioning