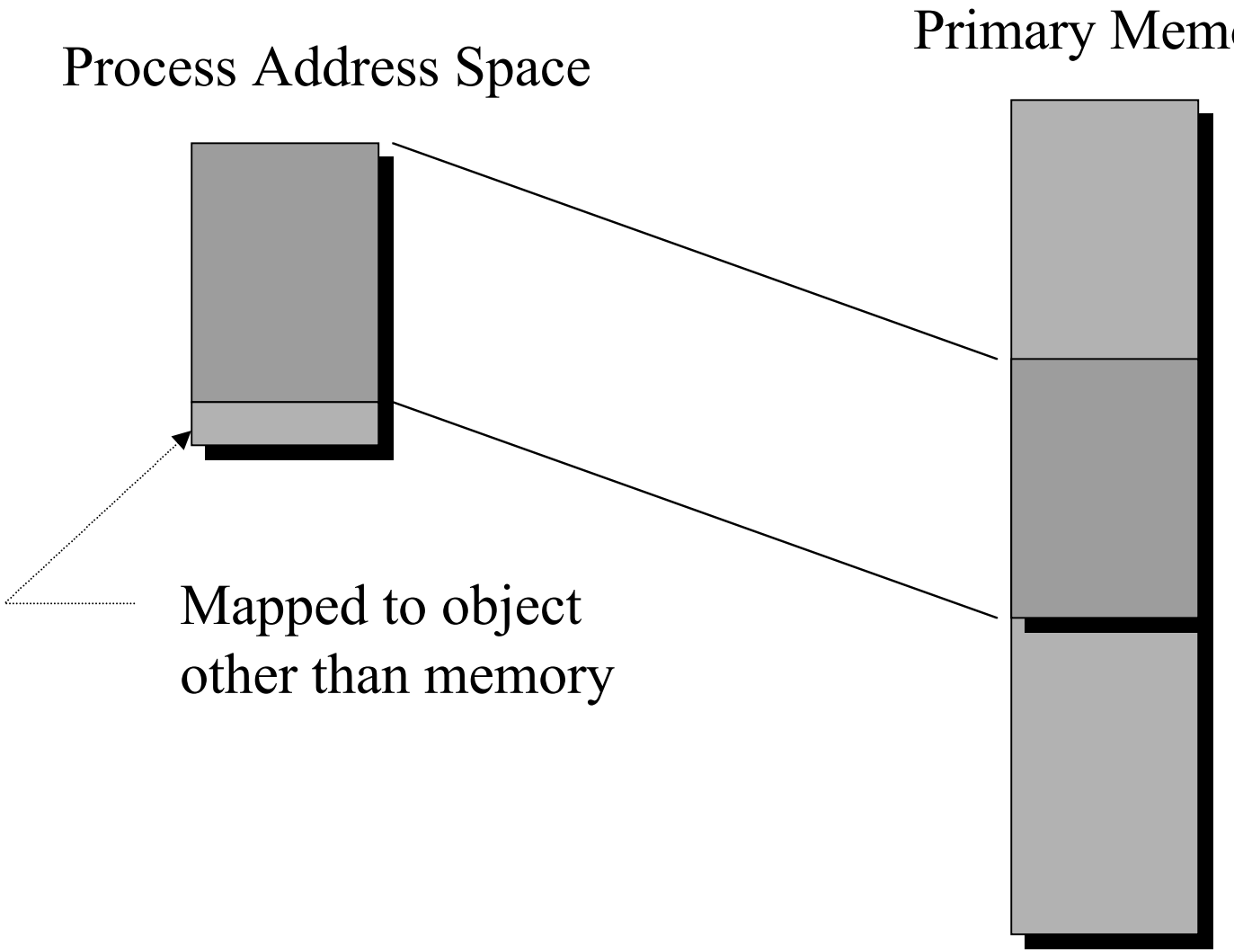
Memory Management

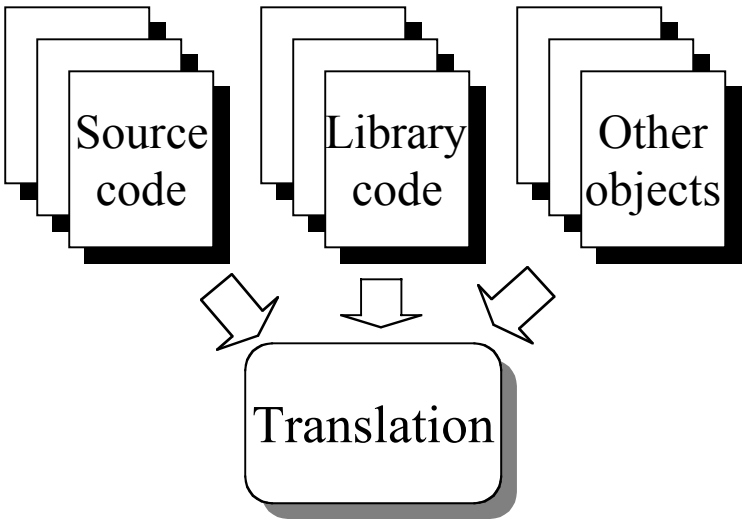
Memory Manager

- Requirements
  - Minimize primary memory access time
  - Maximize primary memory size
  - Primary memory must be cost-effective
- Today's memory manager:
  - Allocates primary memory to processes
  - Maps process address space to primary memory
  - Minimizes access time using cost-effective memory configuration

Address Space vs Primary Memory

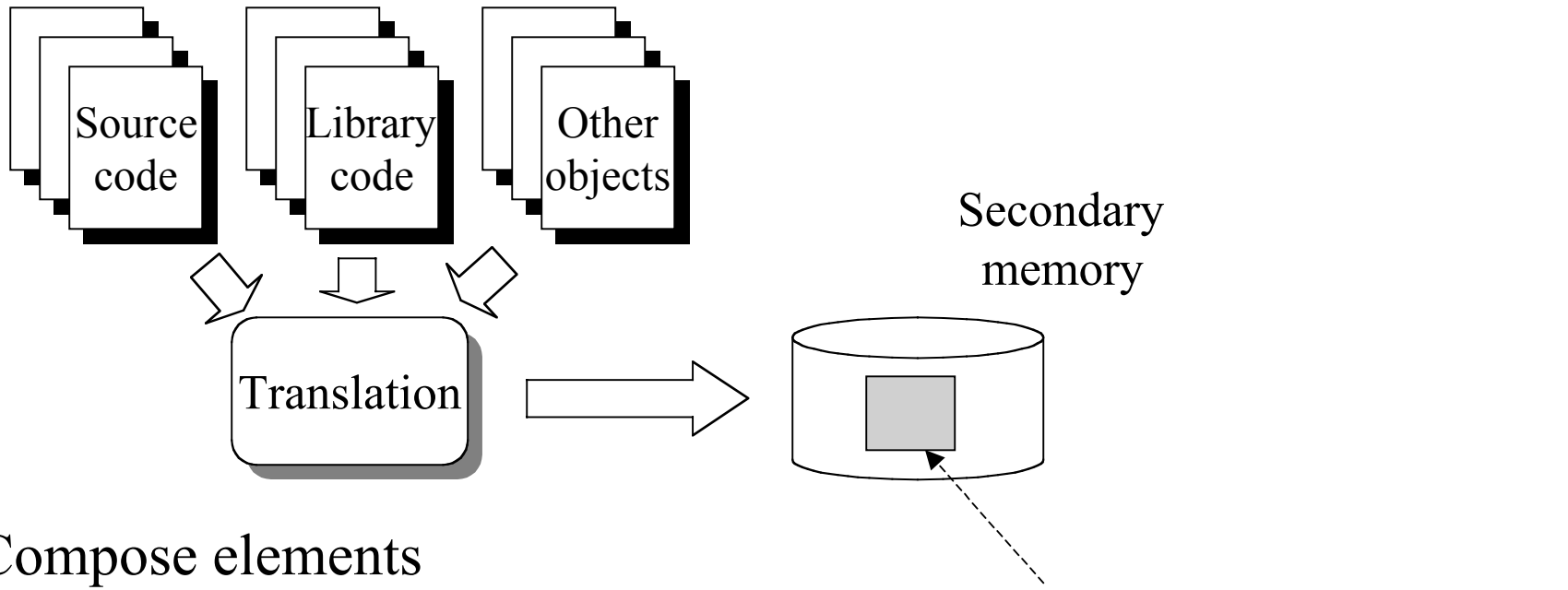


Building the Address Space



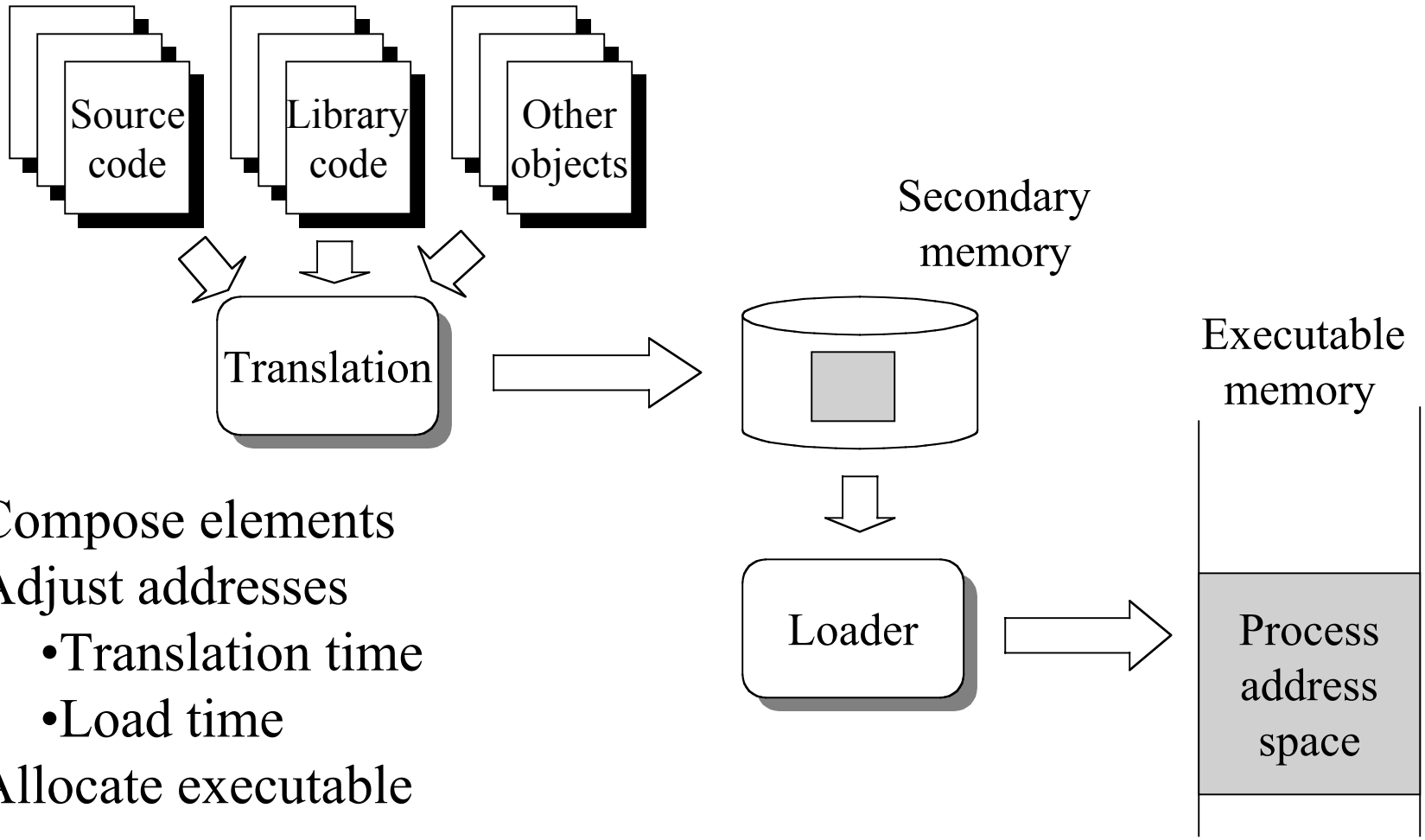
•Compose elements

Building the Address Space



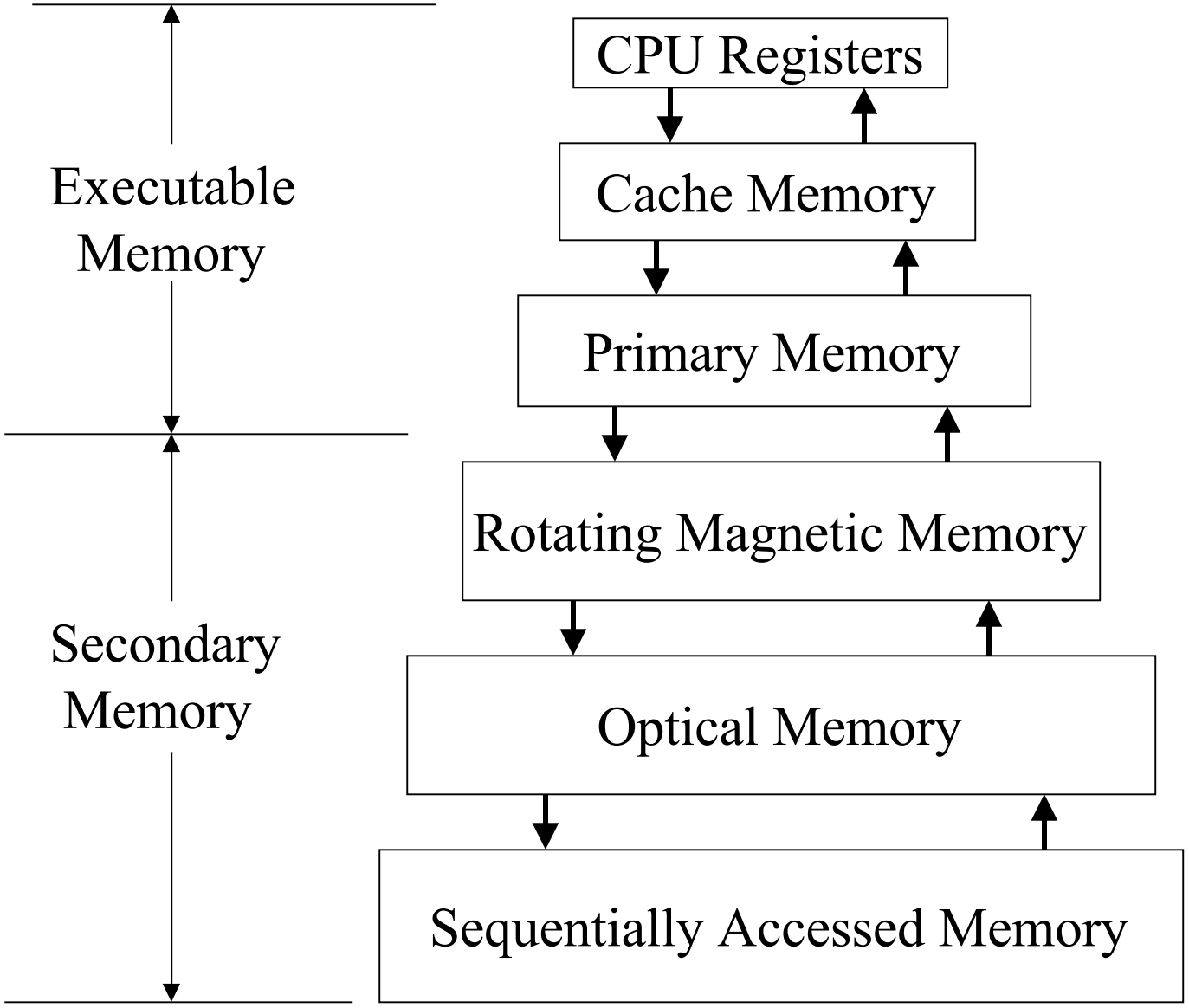
- Compose elements
- Adjust addresses
  - Translation time
  - Load time

Building the Address Space

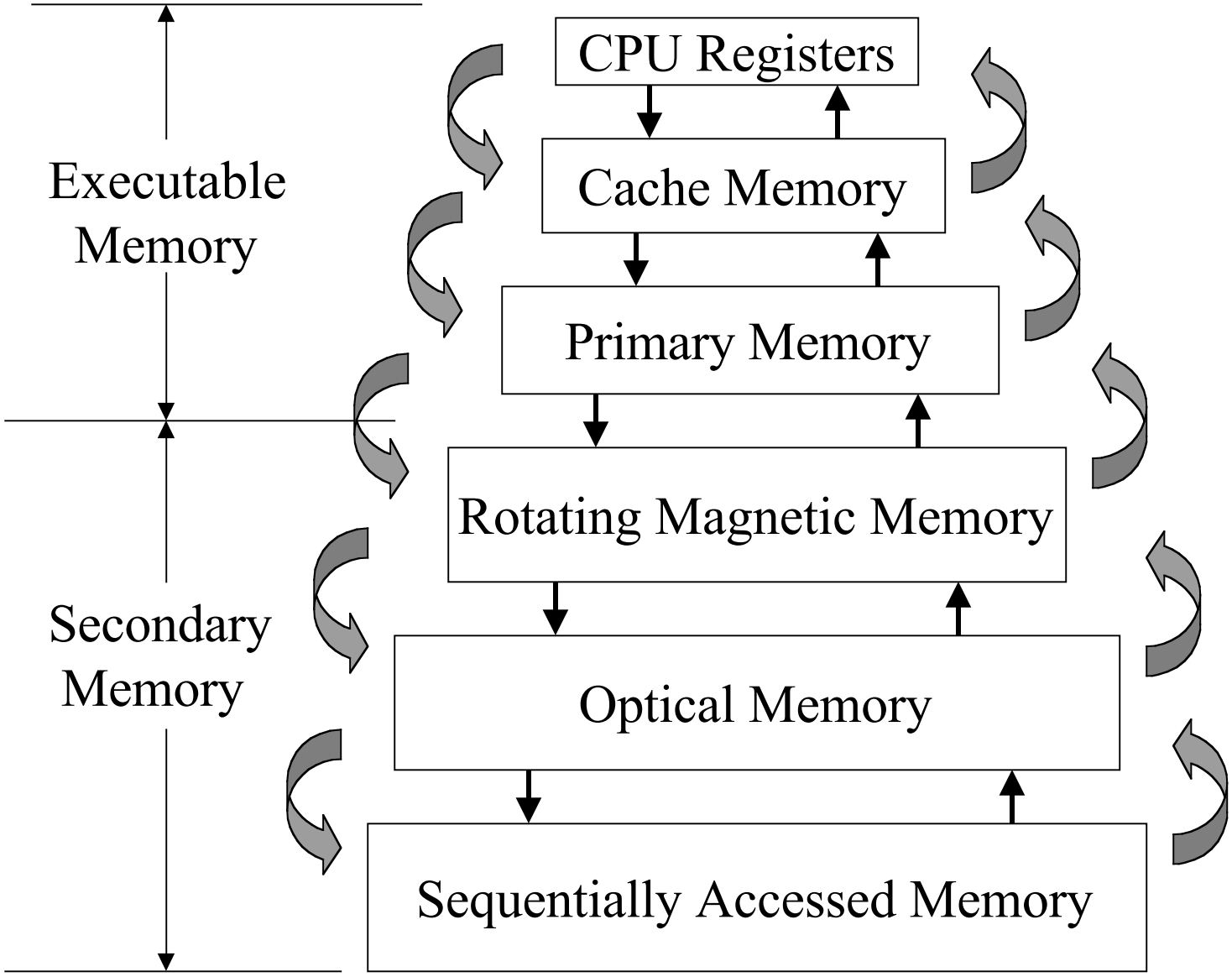


- Compose elements
- Adjust addresses
  - Translation time
  - Load time
- Allocate executable memory space

Memory Hierarchies



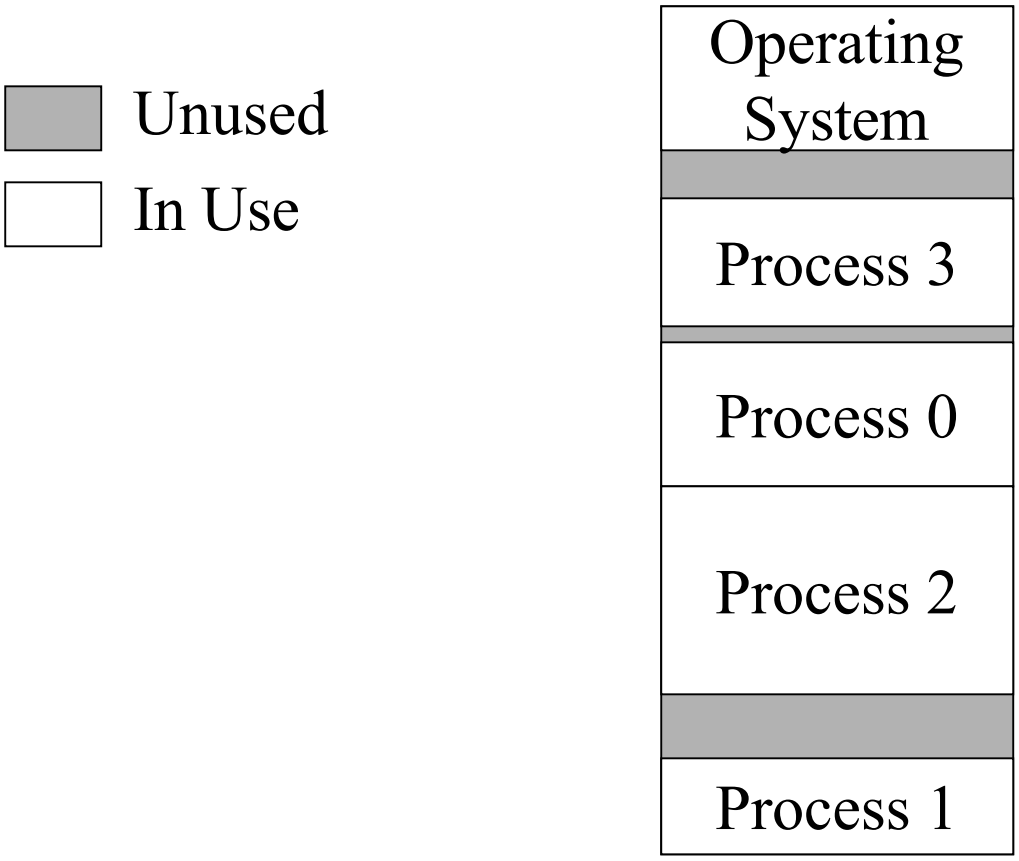
Memory Hierarchies



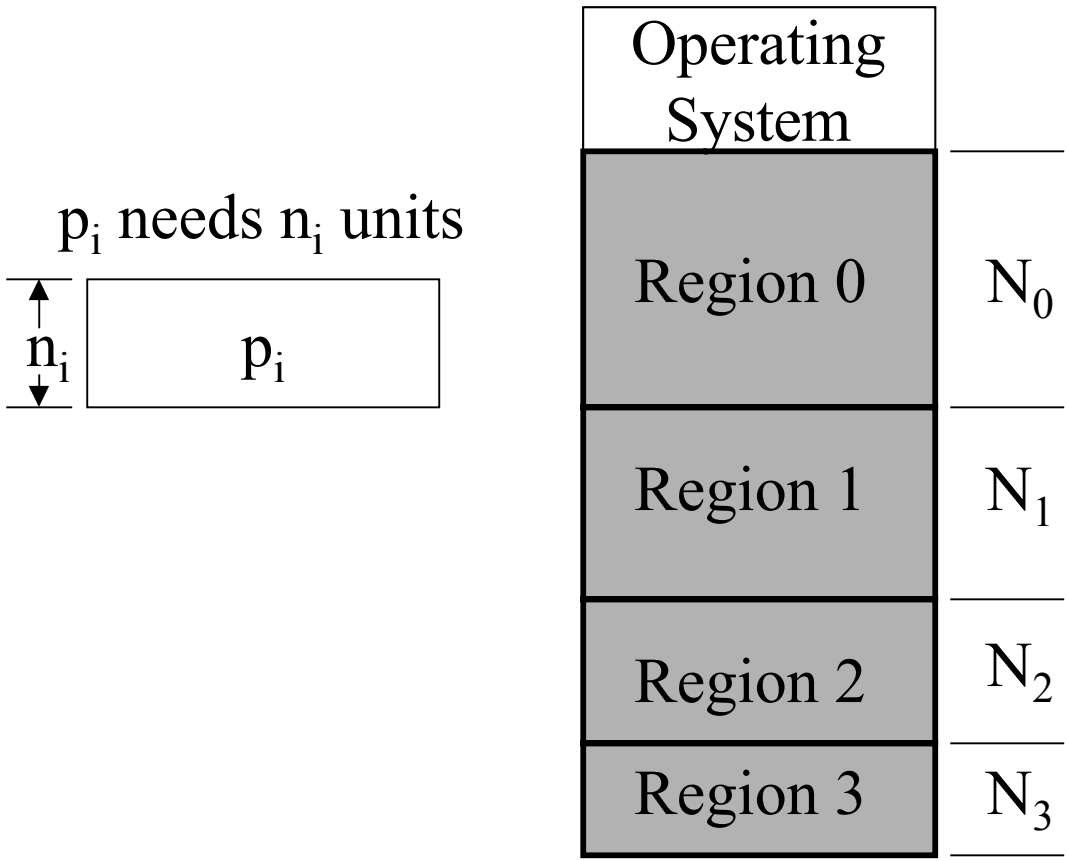
Managing the Hierarchy

- Move across executable-secondary memory boundary (or lower) requires I/O operation
- Upward moves are *copy* operations
  - Require allocation in upper memory
  - Image exists in both memories
- Updates are first applied to upper memory
- Downward move is (usually) *destructive*
  - Deallocate upper memory
  - Updates image in secondary memory

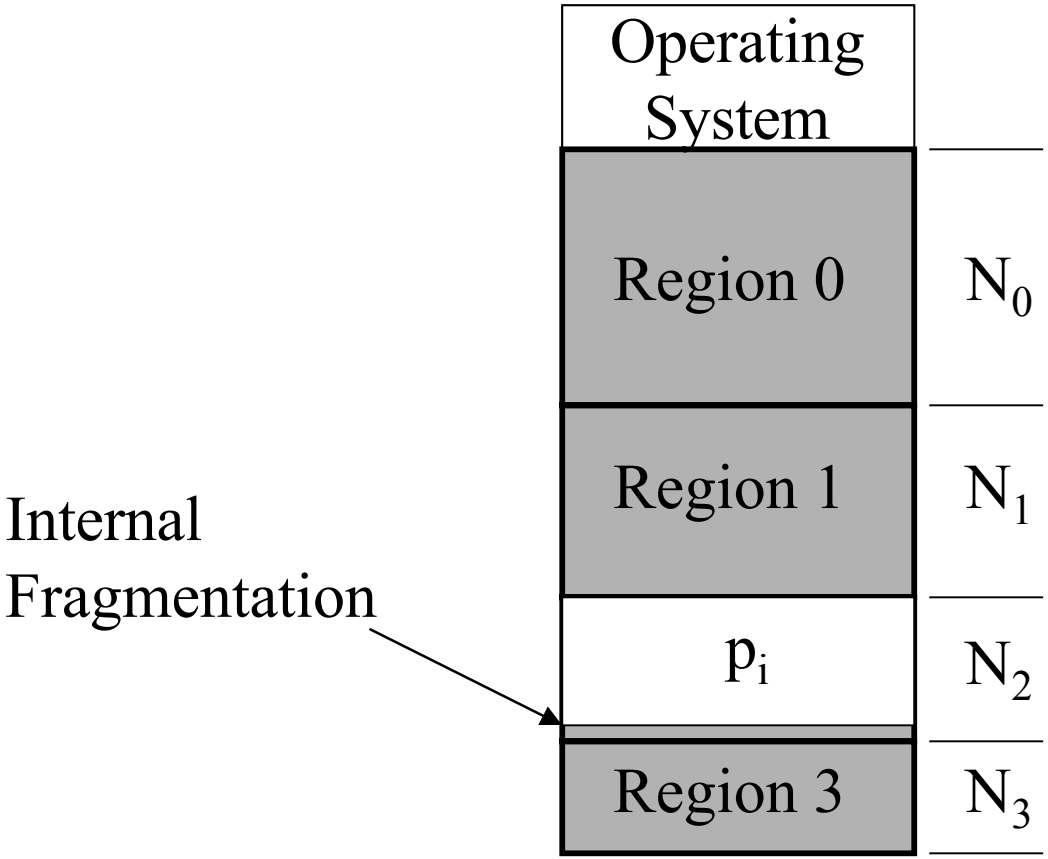
Memory Allocation



Fixed-Partition Memory

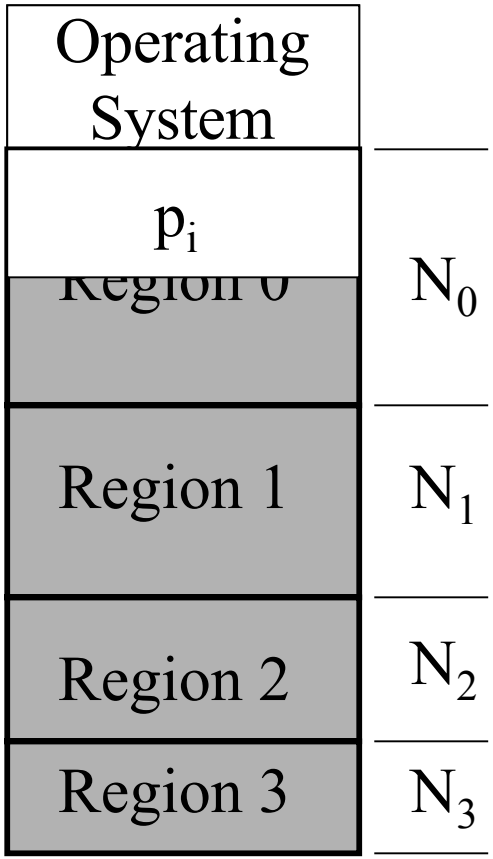


Fixed-Partition Memory -- Best-Fit

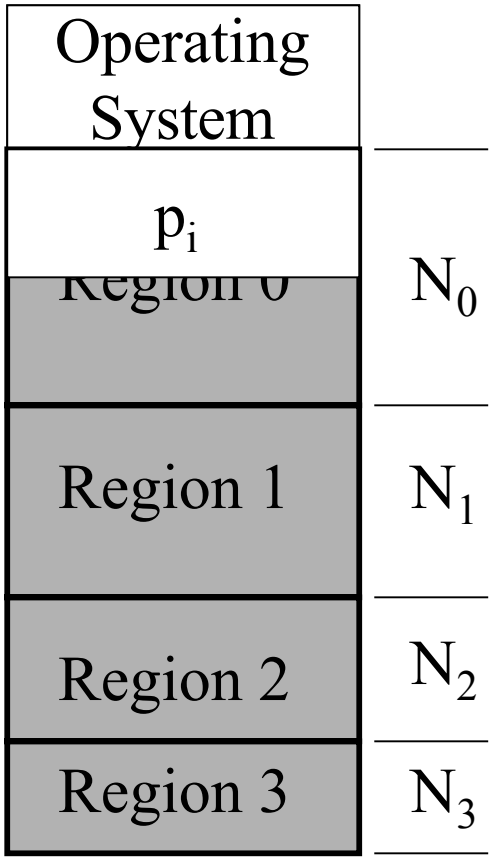


- Loader must adjust every address in the absolute module when placed in memory

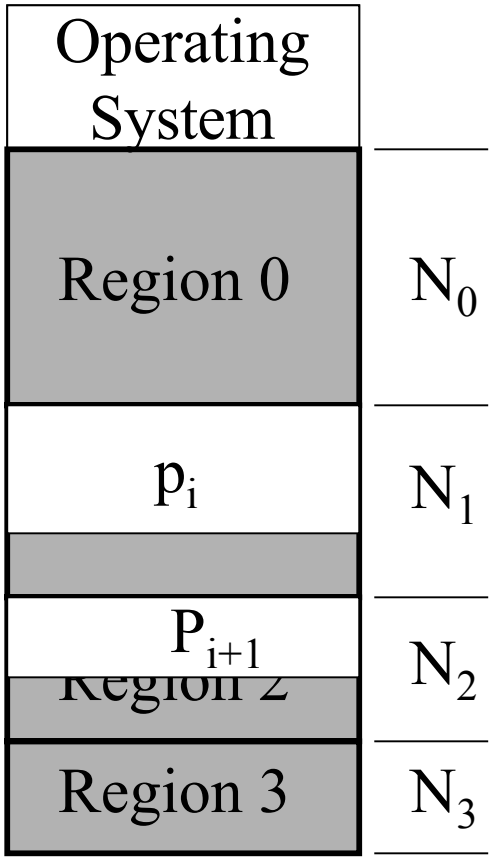
Fixed-Partition Memory -- Worst-Fit



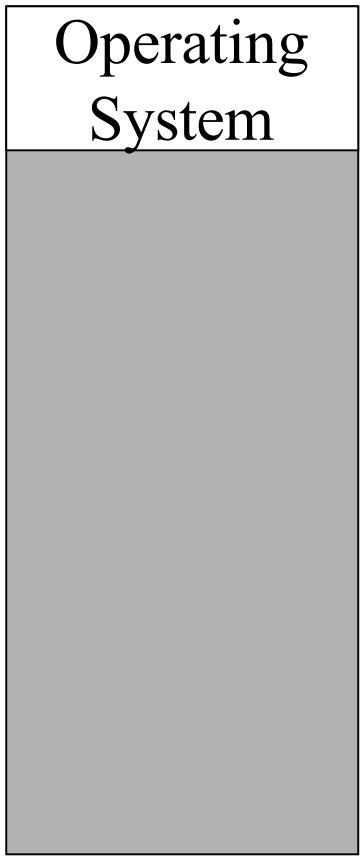
Fixed-Partition Memory -- First-Fit



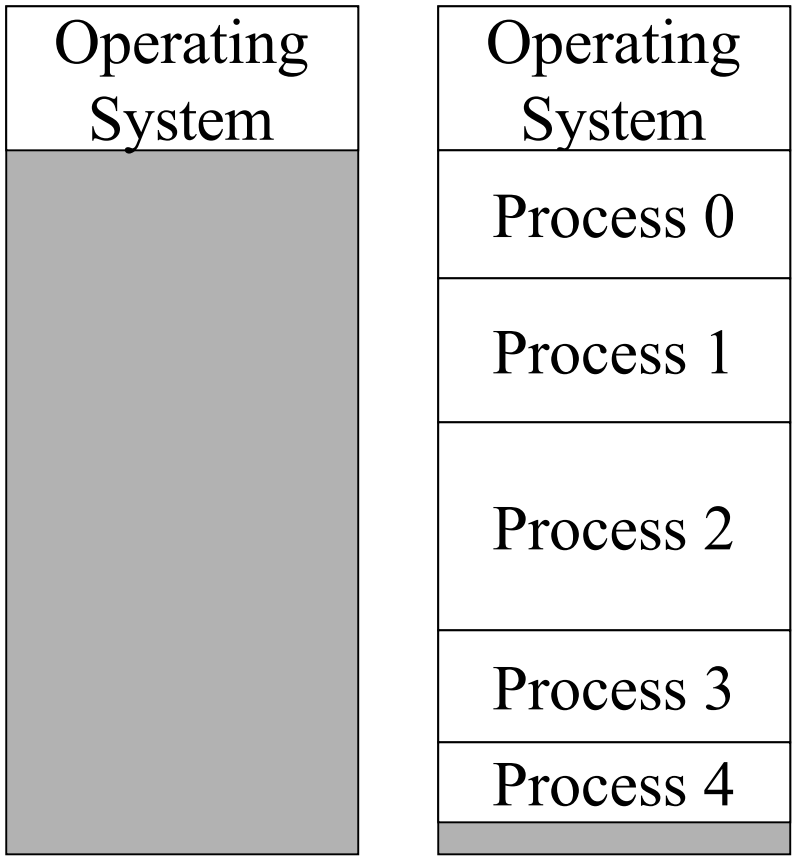
Fixed-Partition Memory -- Next-Fit



Variable Partition Memory

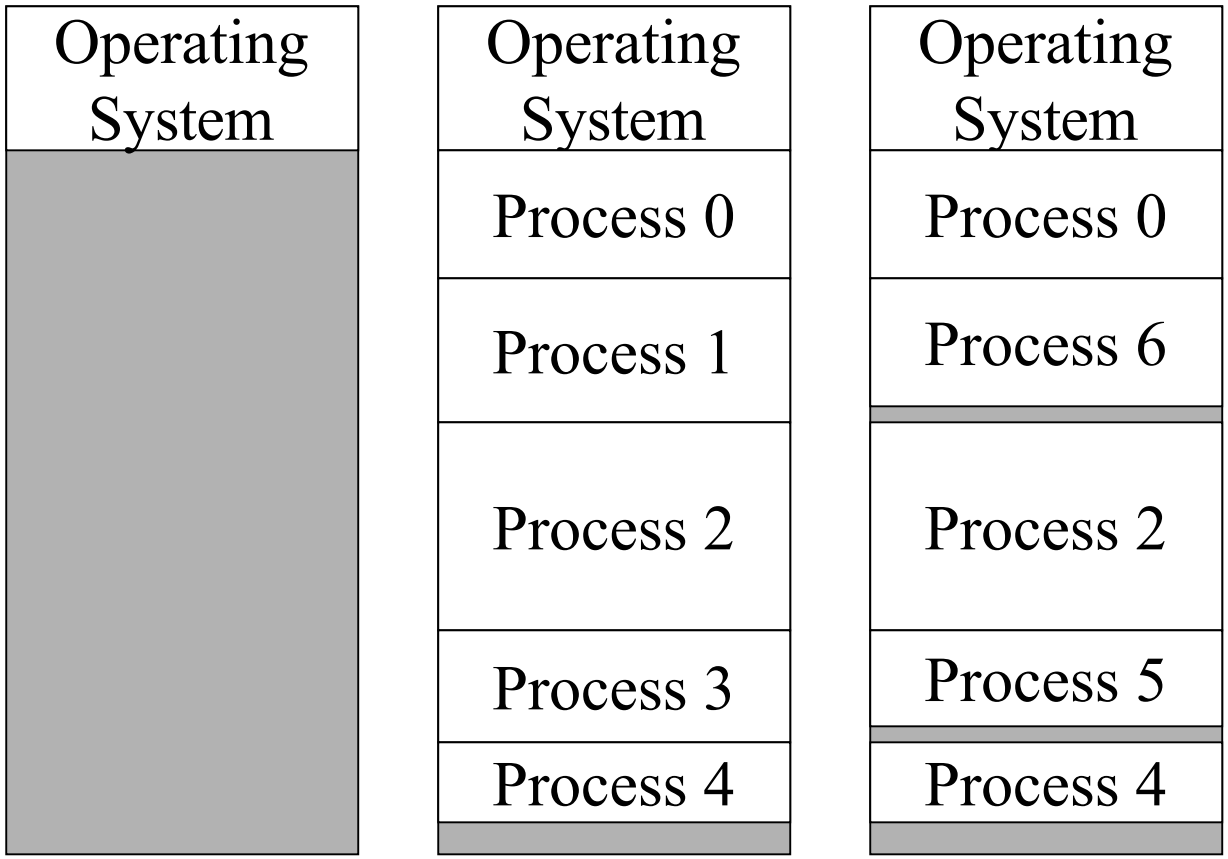


Variable Partition Memory



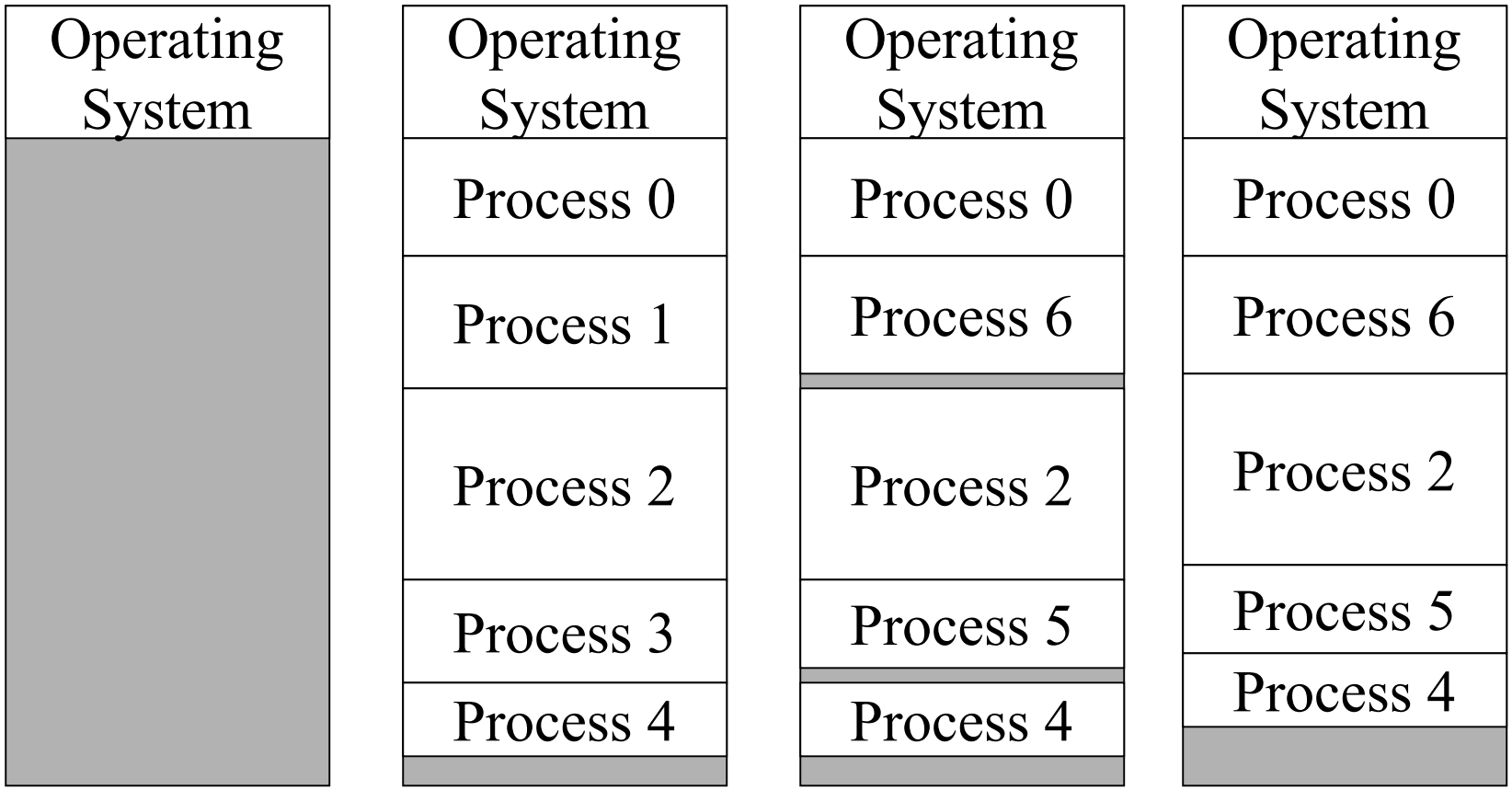
- Loader must adjust every address in every absolute module when placed in memory

Variable Partition Memory



- External fragmentation

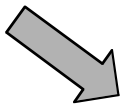
Variable Partition Memory



- ***Compaction*** moves program in memory

Cost of Moving Programs

```
load    R1, 0x02010
```

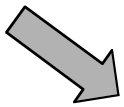


```
3F013010
```

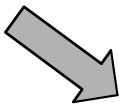
Program loaded at 0x01000

Cost of Moving Programs

```
load    R1, 0x02010
```



```
3F013010
```



Program loaded at 0x01000

```
3F016010
```

Program loaded at 0x04000

- Must run loader over program again!

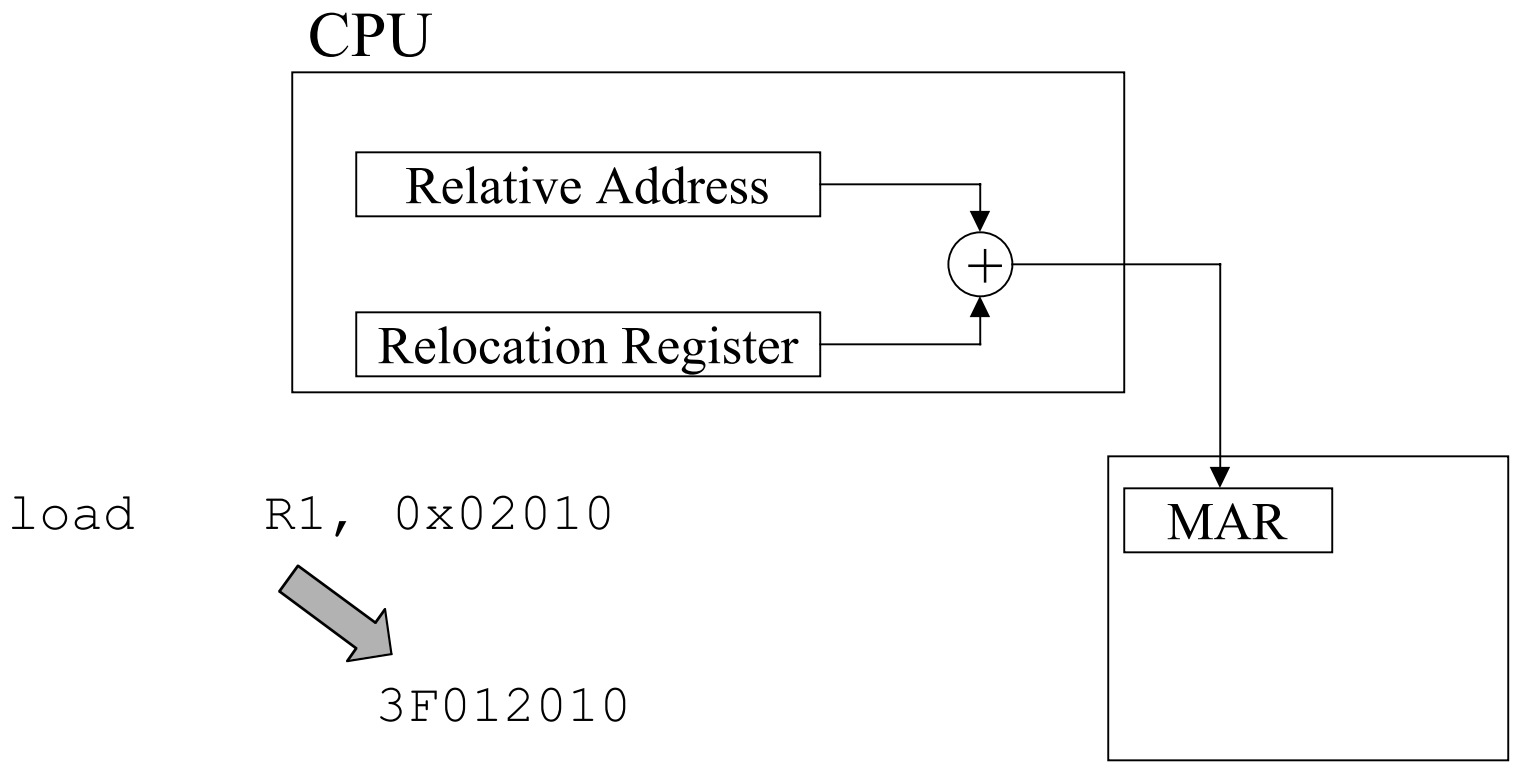
Dynamic Memory Allocation

- Common to use *dynamically allocated* memory
- Process wants to change the size of its address space
  - Smaller ⇒ Creates an external fragment
  - Larger ⇒ Have to move/relocate the program
- Allocate "holes" in memory according to
  - Best- /Worst- / First- /Next-fit

Swapping

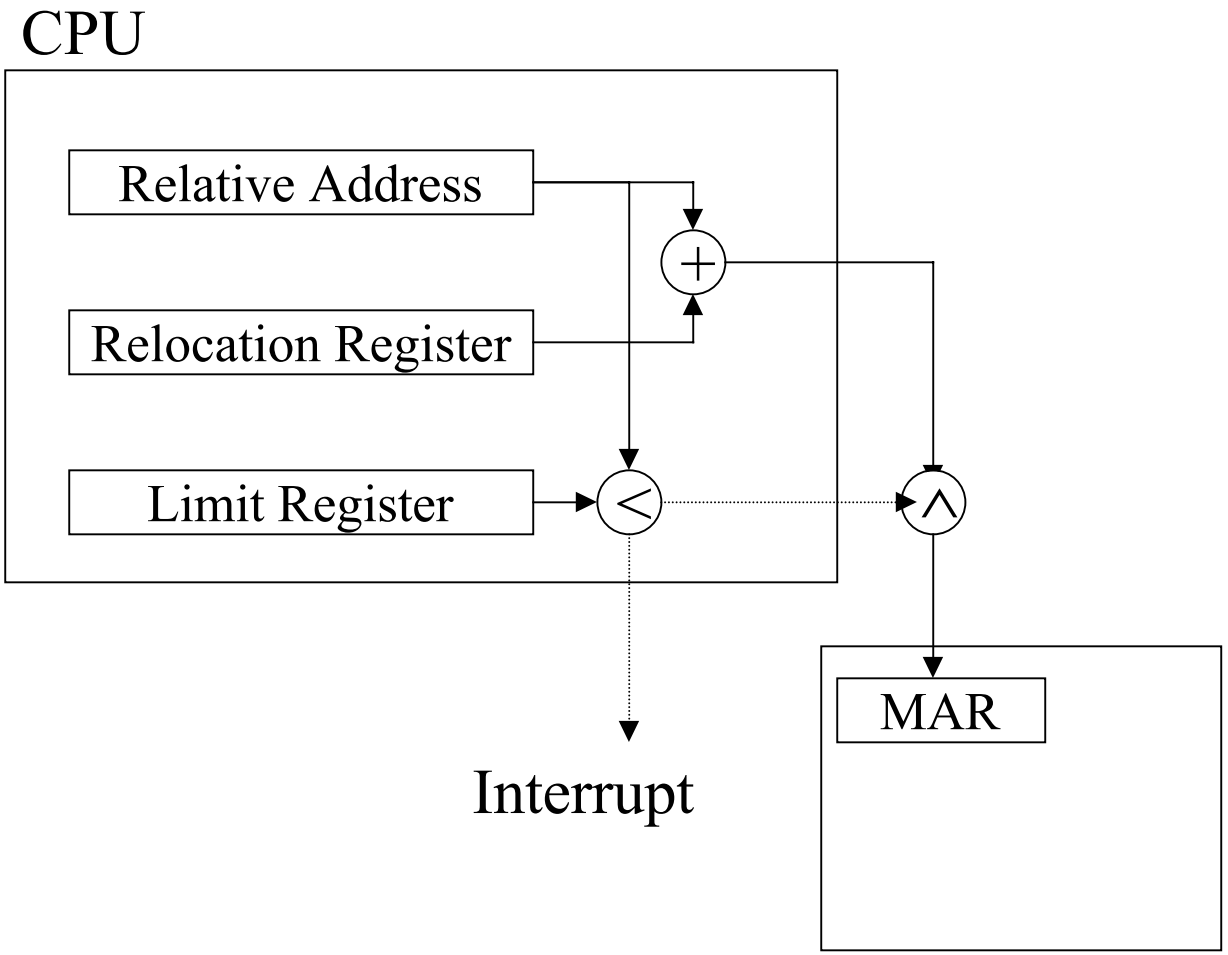
- Suppose there is high demand for executable memory
- Equitable policy might be to time-multiplex processes into the memory (also space-mux)
- Means that process can have its address space unloaded when it still needs memory
  - Usually only happens when it is blocked
- Have same problems as dynamic memory allocation

Dynamic Address Relocation



- Program loaded at 0x01000 ⇒ Relocation Register = 0x01000
- Program loaded at 0x04000 ⇒ Relocation Register = 0x04000

Runtime Bound Checking



Strategies

- Fixed-Partition used only in batch systems
- Variable-Partition used everywhere (except in virtual memory)
- Swapping systems
  - Popularized in timesharing
  - Relies on dynamic address relocation
  - Now dated
- Virtual Memory
  - Paging -- mainstream in contemporary systems
  - Segmentation -- the future

NT Memory-mapped Files

- Open the file
- Create a *section object* (that maps file)
- Identify point in address space to place the file