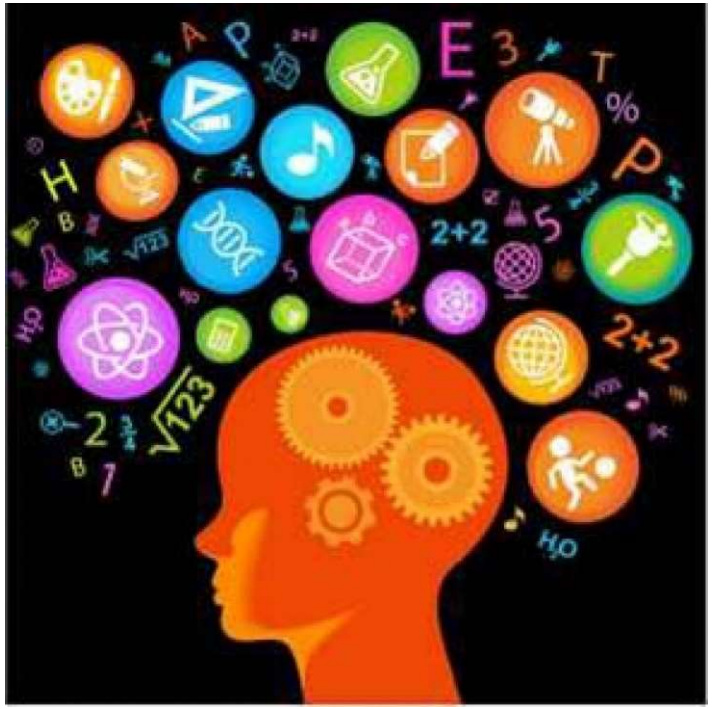
Memory Management
Part –1



Prachi Pandey

C-DAC Bangalore

prachip@cdac.in

Part 1 -Topics

- Basics of computer memory
- Need for Memory Management
- Concepts
  - Address, Address space
  - Address Translation
  - Address Binding
  - Loading, Linking
  - Swapping
- Memory Allocation
  - Continuous/Contiguous Allocation
    - Fixed/Static Partitioning
    - Internal and External Fragmentation, Compaction
    - Variable/Dynamic Partitioning
    - Memory Allocation Techniques
      - First Fit
      - Best Fit
      - Worst Fit

Basics of Memory

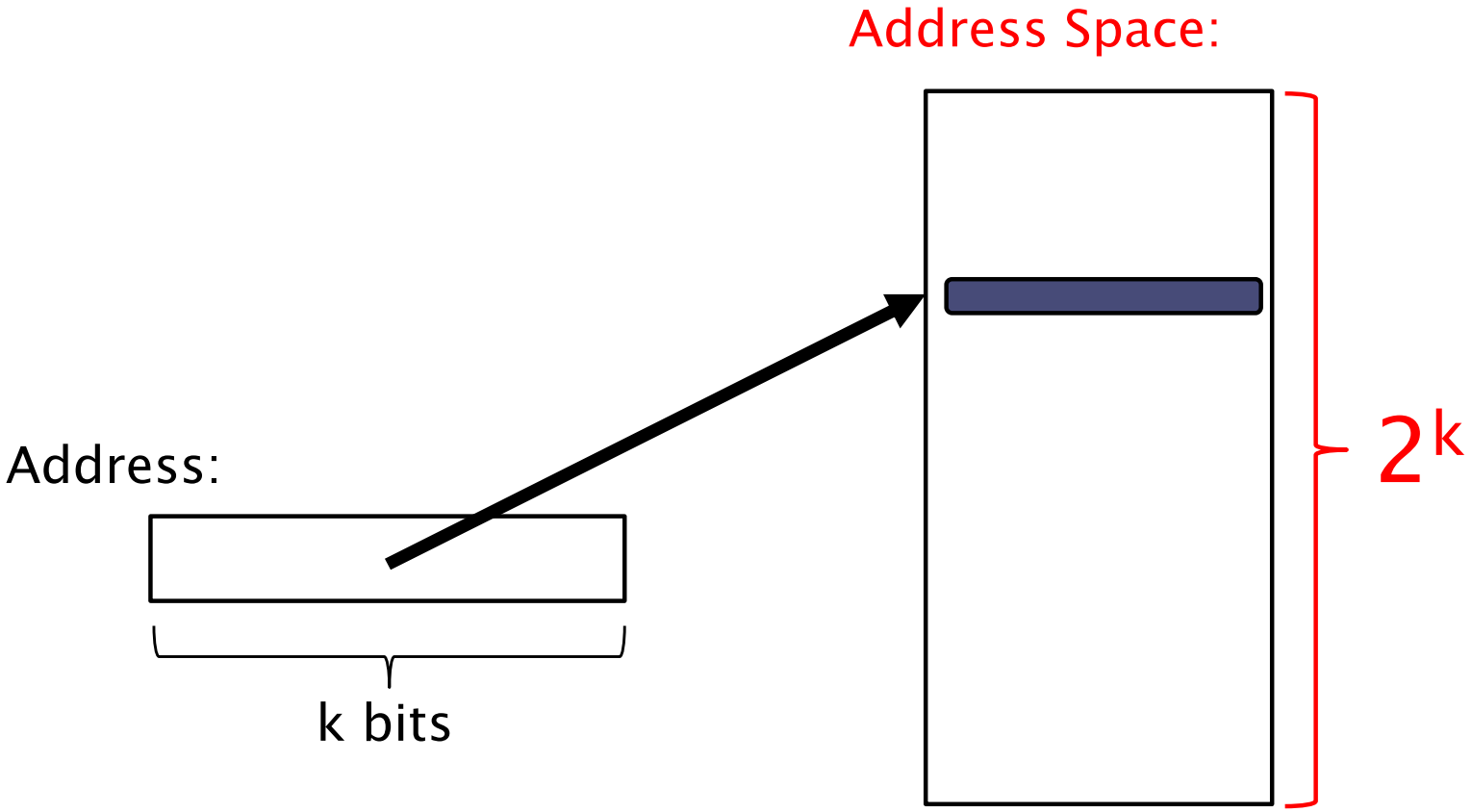
- Basic unit of memory is bit.
- A bit is a single unit of digital information and is represented either as 0 or a 1.
- Everything in a computer's memory takes the form of bits i.e. 0's and 1's.
- Example you are watching a movie, opening a text file etc , to the computer they are all 0's and 1's.
- Each of these are stored in a memory cell which can switch between two states 0 and 1.
- Files and programs consists of millions of these bits.
- These are all processed in the Central Processing Unit (CPU).
- CPU acts as the computer's brain.

Since a computer operates only on bits ie. 0s and 1s, hence the units are represented as power of 2s.

Multiples of Bits		
Unit (Symbol)	**Value (SI)**	**Value (Binary)**
Kilobit (Kb) (Kbit)	10^3	2^{10}
Megabit (Mb) (Mbit)	10^6	2^{20}
Gigabit (Gb) (Gbit)	10^9	2^{30}
Terabit (Tb) (Tbit)	10^{12}	2^{40}
Petabit (Pb) (Pbit)	10^{15}	2^{50}
Exabit (Eb) (Ebit)	10^{18}	2^{60}
Zettabit (Zb) (Zbit)	10^{21}	2^{70}
Yottabit (Yb) (Ybit)	10^{24}	2^{80}

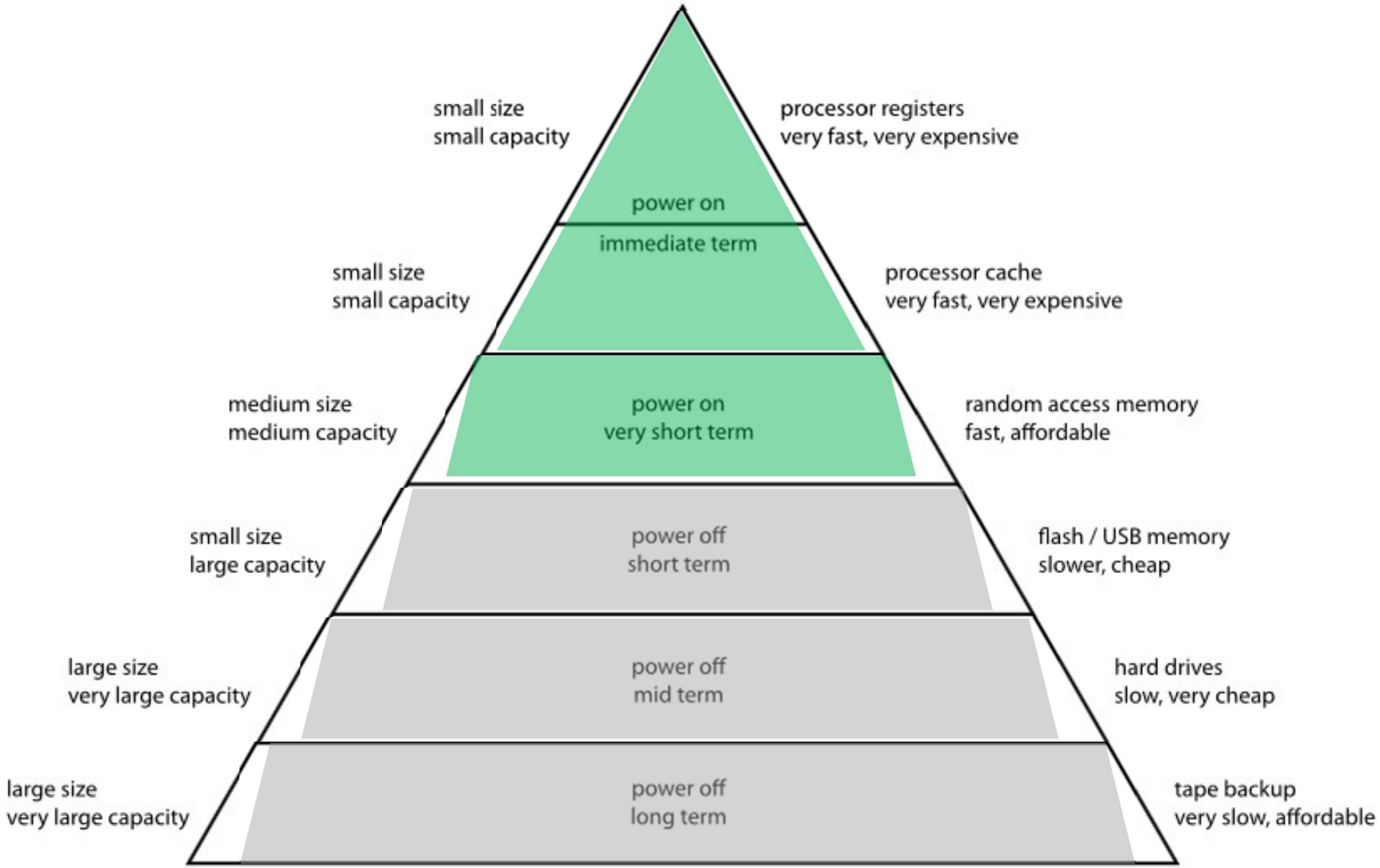
Multiples of Bytes		
Unit (Symbol)	**Value (SI)**	**Value (Binary)**
Kilobyte (kB)	10^3	2^{10}
Megabyte (MB)	10^6	2^{20}
Gigabyte (GB)	10^9	2^{30}
Terabyte (TB)	10^{12}	2^{40}
Petabyte (PB)	10^{15}	2^{50}
Exabyte (EB)	10^{18}	2^{60}
Zettabyte (ZB)	10^{21}	2^{70}
Yottabyte (YB)	10^{24}	2^{80}

Address/Address Space



- What is 2^{10} bytes (where a byte is abbreviated as "B")?
  - 2^{10} B = 1024b = 1 KB (for memory, 1K = 1024, *not* 1000)
- How many bits to address each byte of 4KB page?
  - 4KB = 4×1KB = 4× 2^{10}= 2^{12}⇒ 12 bits (or $\log_2 4$)
- How much memory can be addressed with 20 bits? 32 bits? 64 bits?

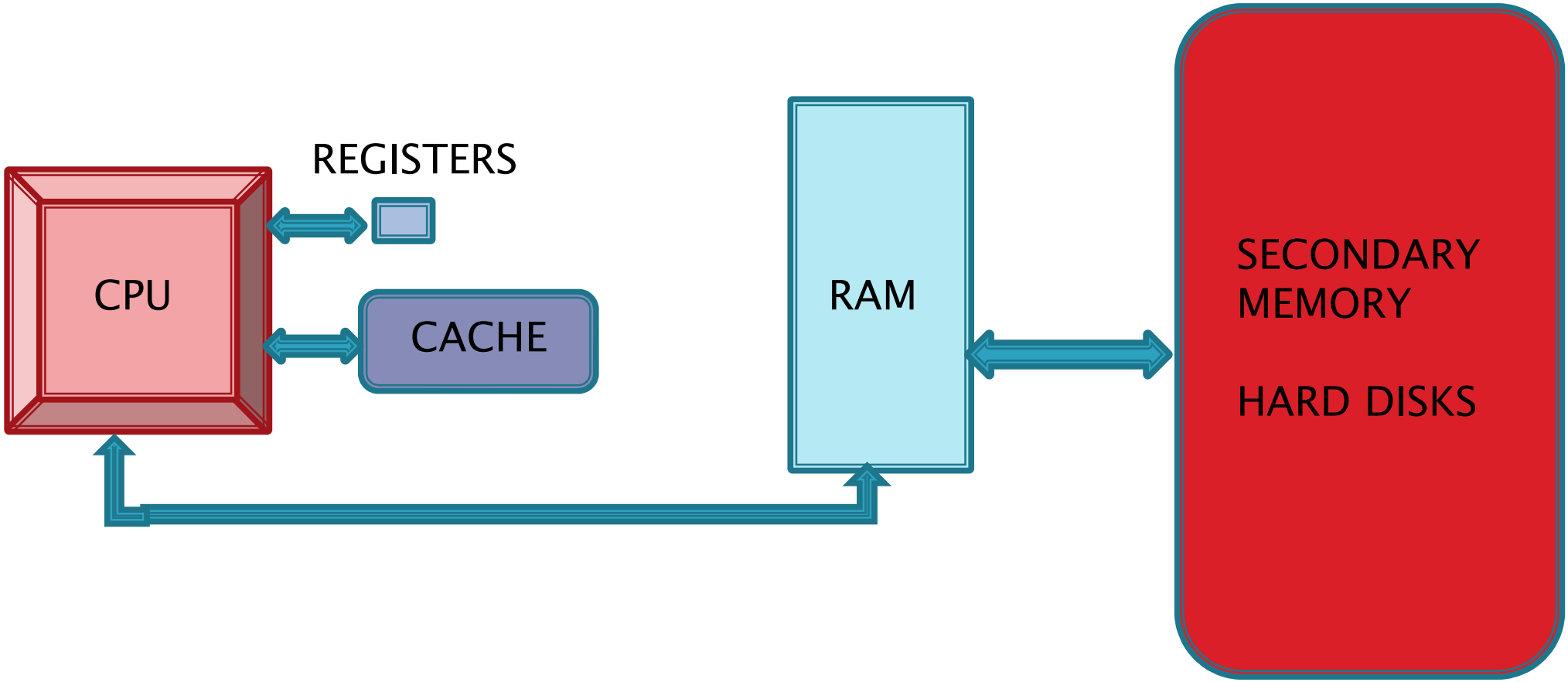
Computer Memory Hierarchy



Need for Memory Management

- As the number of bits to be processed grows exponentially, the need for memory management becomes essential.
- The following factors provide the basis of memory management:
  - Size – More the better
  - Cost – Less the better
  - Speed – More the better
- If we try to increase the size, the speed of access to data will decrease.
- We need to find an optimal solution which will provide the desired results.
- Memory management is one of the most important features of the operating system because it affects the execution time of process directly

Memory Hardware



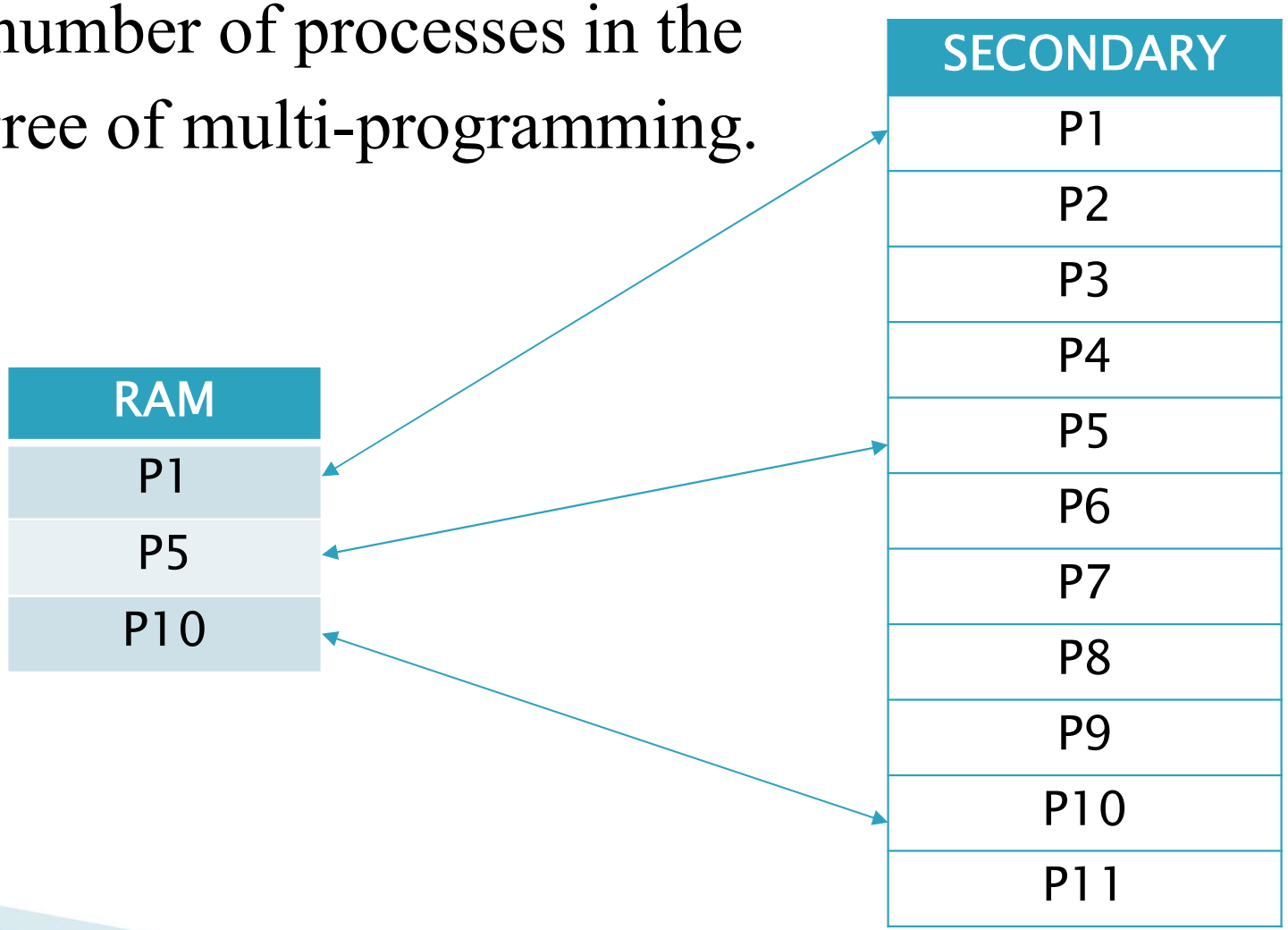
CPU does not have access to Secondary memory directly.

Concepts

- Program must be brought (from disk) into memory for it to be executed
- Main memory and registers are the only storage which CPU can access directly
- Memory unit only sees a stream of:
  - addresses + read requests, or
  - address + data and write requests
- Register access is done in one CPU clock (or less)
- Main memory can take many cycles, causing a **stall**
- **Cache** sits between main memory and CPU registers
- Protection of memory is required to ensure correct operation

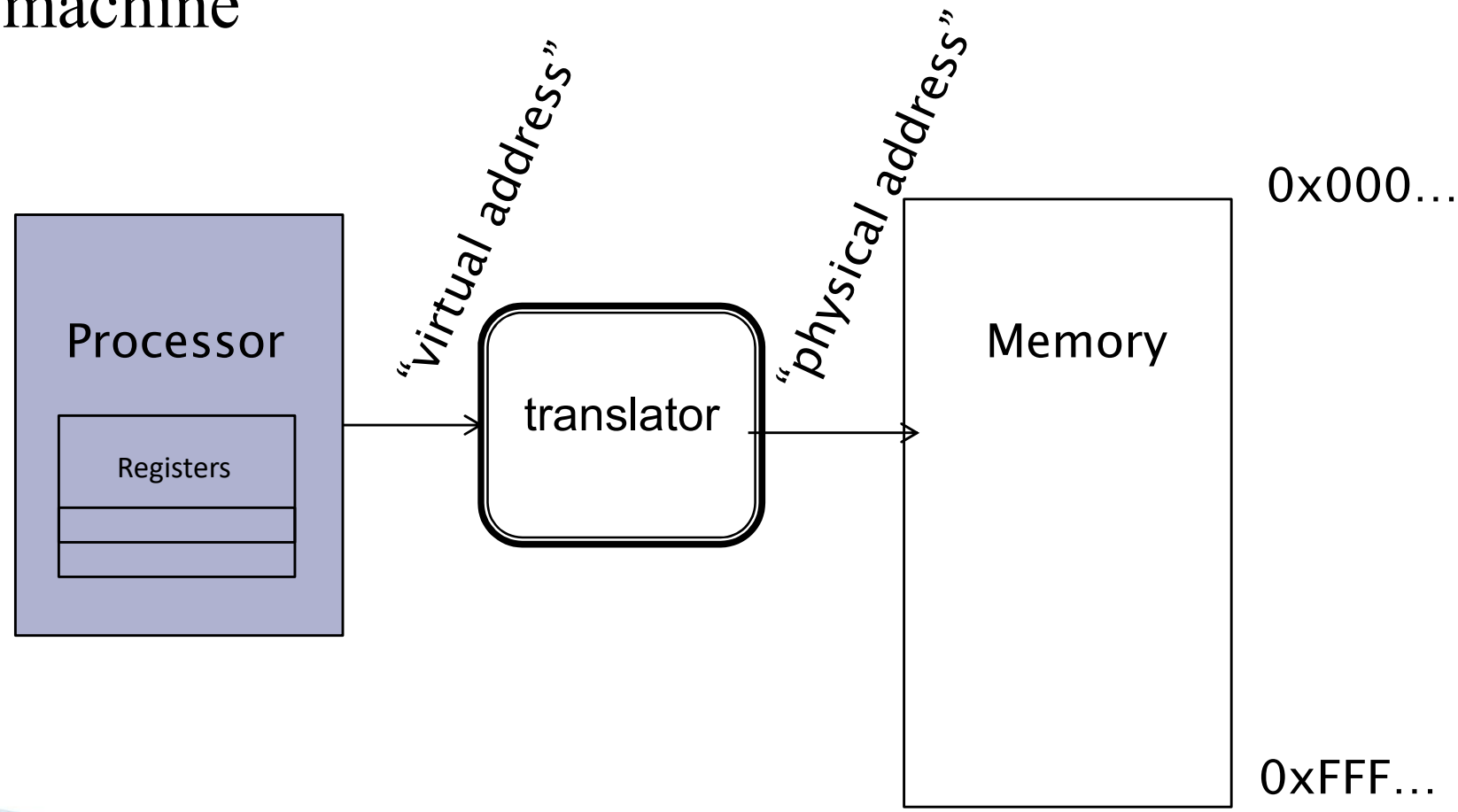
Degree of Multi-programming

- All programs are stored in secondary memory/storage.
- To execute the programs, we need to load them into RAM
- The maximum number of processes in the RAM is the degree of multi-programming.



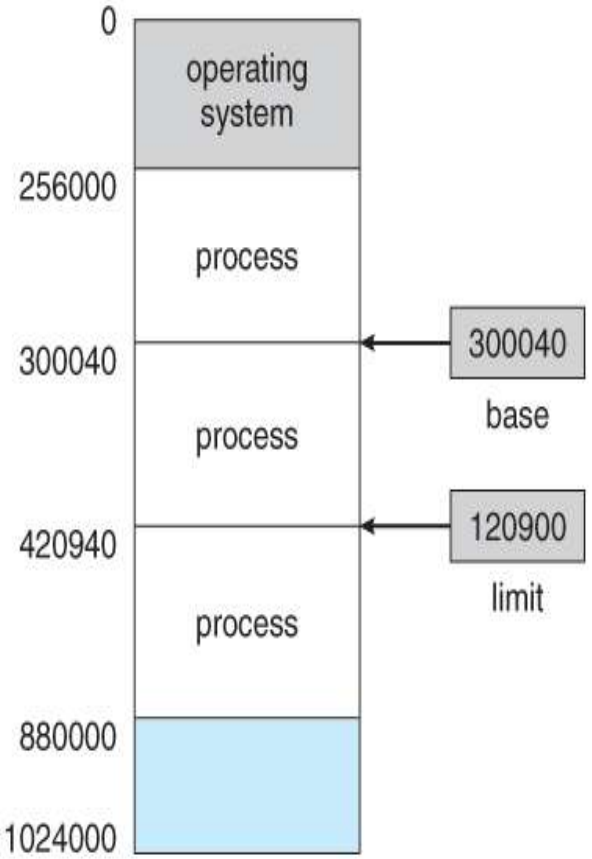
Address Translation

- Program operates in an address space that is distinct from the physical memory space of the machine

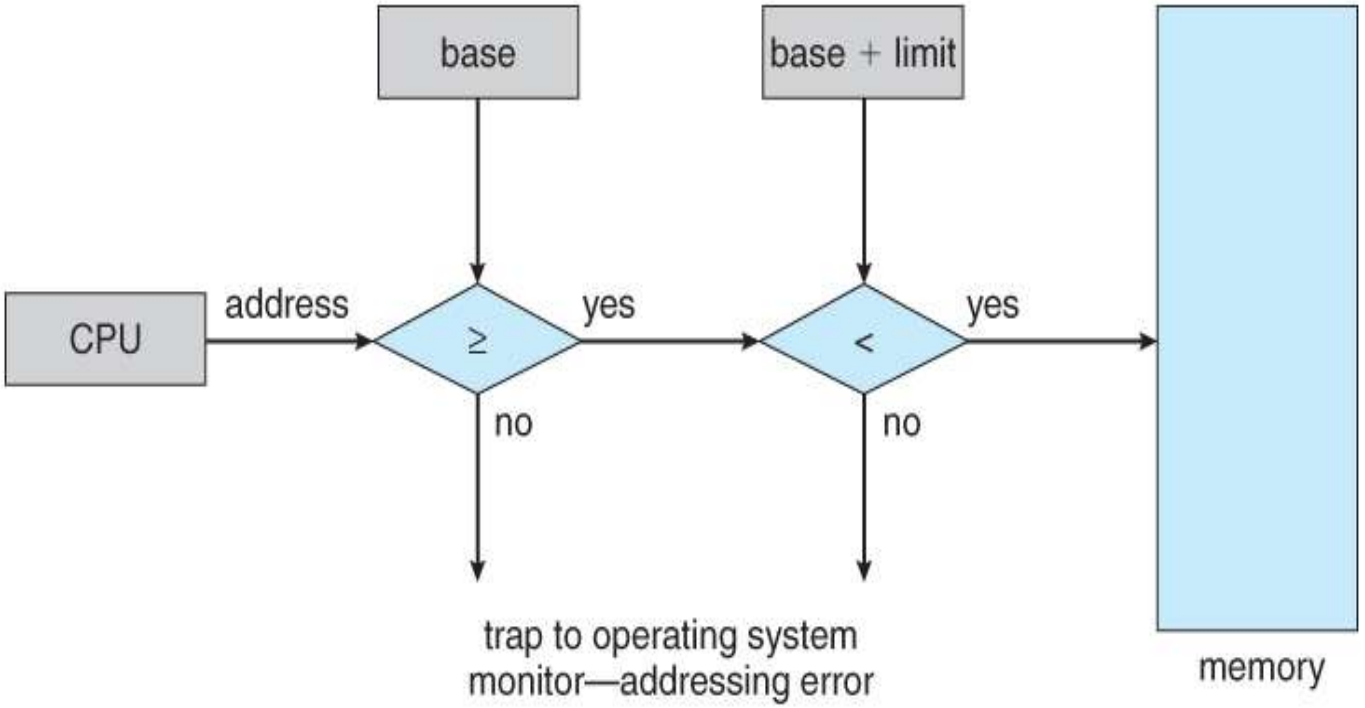


Address Translation

- The memory hardware doesn't know what a particular part of memory is being used for
- User processes must be restricted so that they only access memory locations that "belong" to that particular process.
- This is usually implemented using a **base register** and a **limit register** for **each process.**
- ***Every*** memory access made by a user process is checked against these two registers, and if a memory access is attempted outside the valid range, then a fatal error is generated.
- The OS has access to all existing memory locations, as this is necessary to swap users' code and data in and out of memory. It should also be obvious that changing the contents of the base and limit registers is a privileged activity, allowed only to the OS kernel.



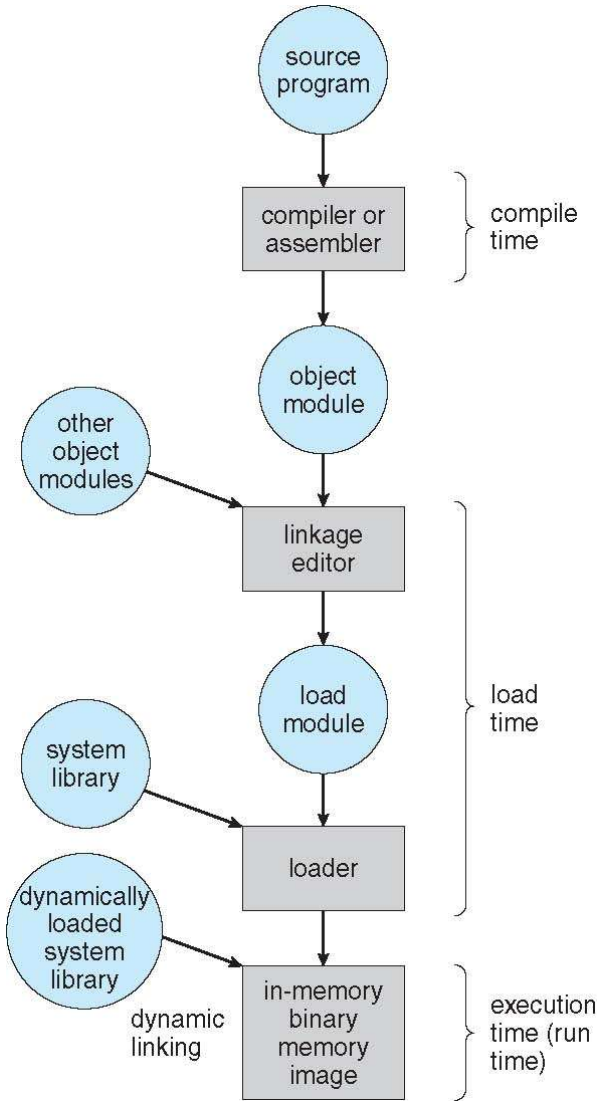
Address Translation Hardware



Address Binding

- **Compile Time** - If it is known at compile time where a program will reside in physical memory, then ***absolute code*** can be generated by the compiler, containing actual physical addresses. However if the load address changes at some later time, then the program will have to be recompiled. (DOS)
- **Load Time** - If the location at which a program will be loaded is not known at compile time, then the compiler must generate ***relocatable code***, which references addresses relative to the start of the program. If that starting address changes, then the program must be reloaded but not recompiled.
- **Execution Time** - If a program can be moved around in memory during the course of its execution, then binding must be delayed until execution time.

Multistep Processing of a User Program



Dynamic Loading

- The entire program does need to be in memory to execute
- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required
  - Implemented through program design
  - OS can help by providing libraries to implement dynamic loading

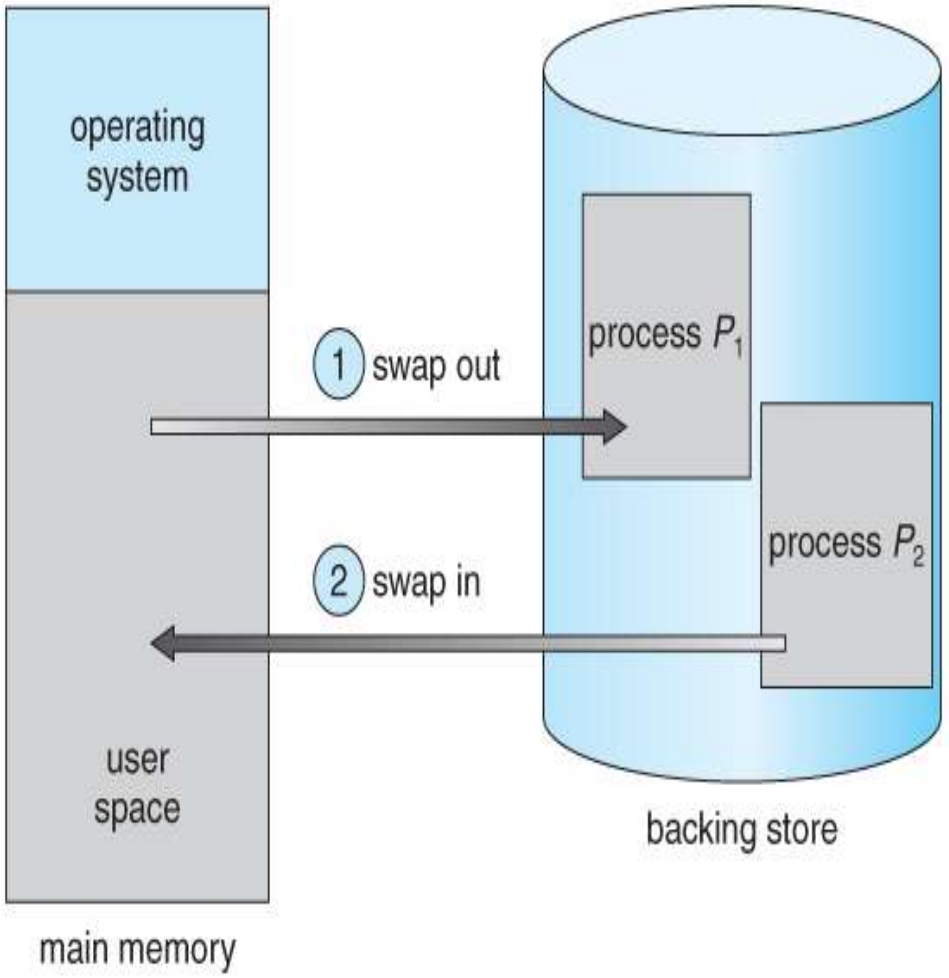
Dynamic Linking

- **Static linking** – system libraries and program code combined by the loader into the binary program image
- **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 checks if routine is in processes' memory address
- Dynamic linking is particularly useful for libraries
- System also known as **shared libraries**
- Consider applicability to patching system libraries
  - Versioning may be needed

Swapping

- A process must be loaded into memory in order to execute.

- If there is not enough memory available to keep all running processes at the same time, some processes who are not currently using the CPU may have their memory swapped out local disk called ***backing store.***
- If compile-time/ load-time address binding is used, then processes must be swapped back into the same memory location from which they were swapped out. If runtime binding is used, then processes can be swapped back into any available location.

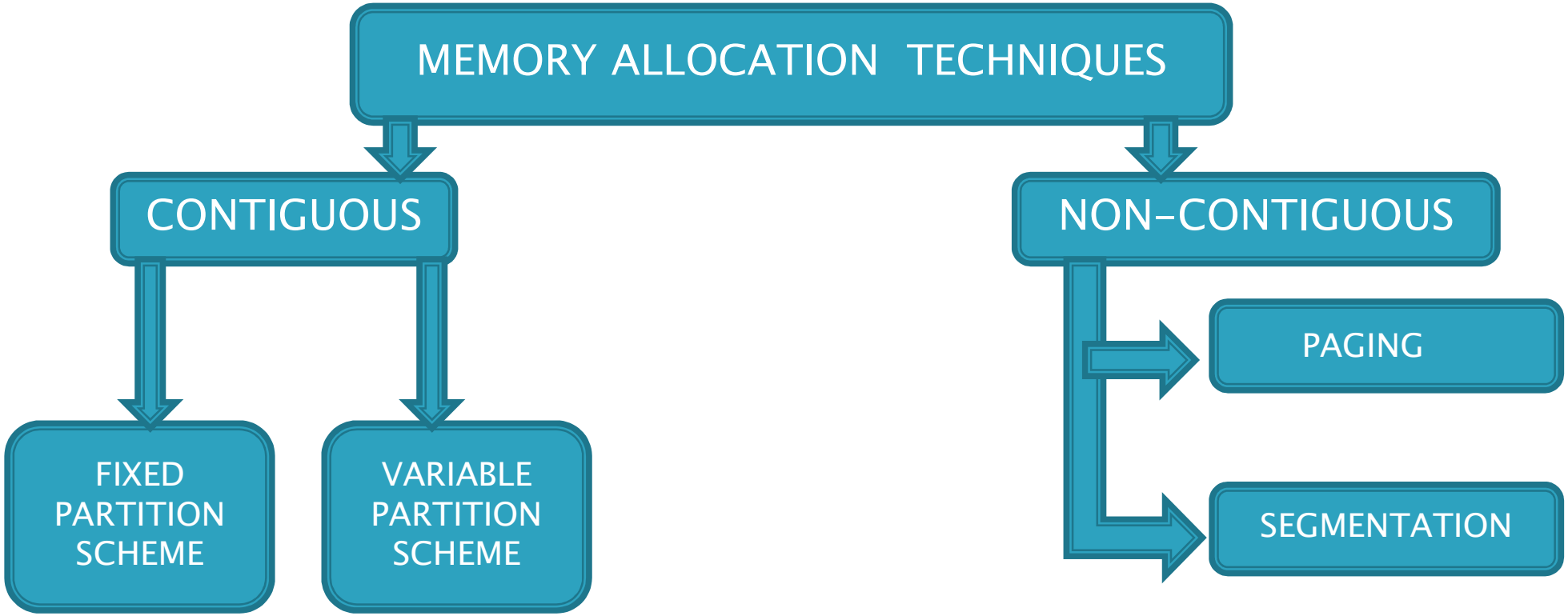


Memory Management functionalities

- The problem the OS has is how best to manage the memory resource, that is how and when to divide the memory into blocks and how and when to allocate the blocks to processes.

- Challenges:
  - Memory Allocation
  - Memory Protection
  - Garbage Collection
  - IO Support
  - Swapping, fragmentation and compaction

Memory Allocation Techniques



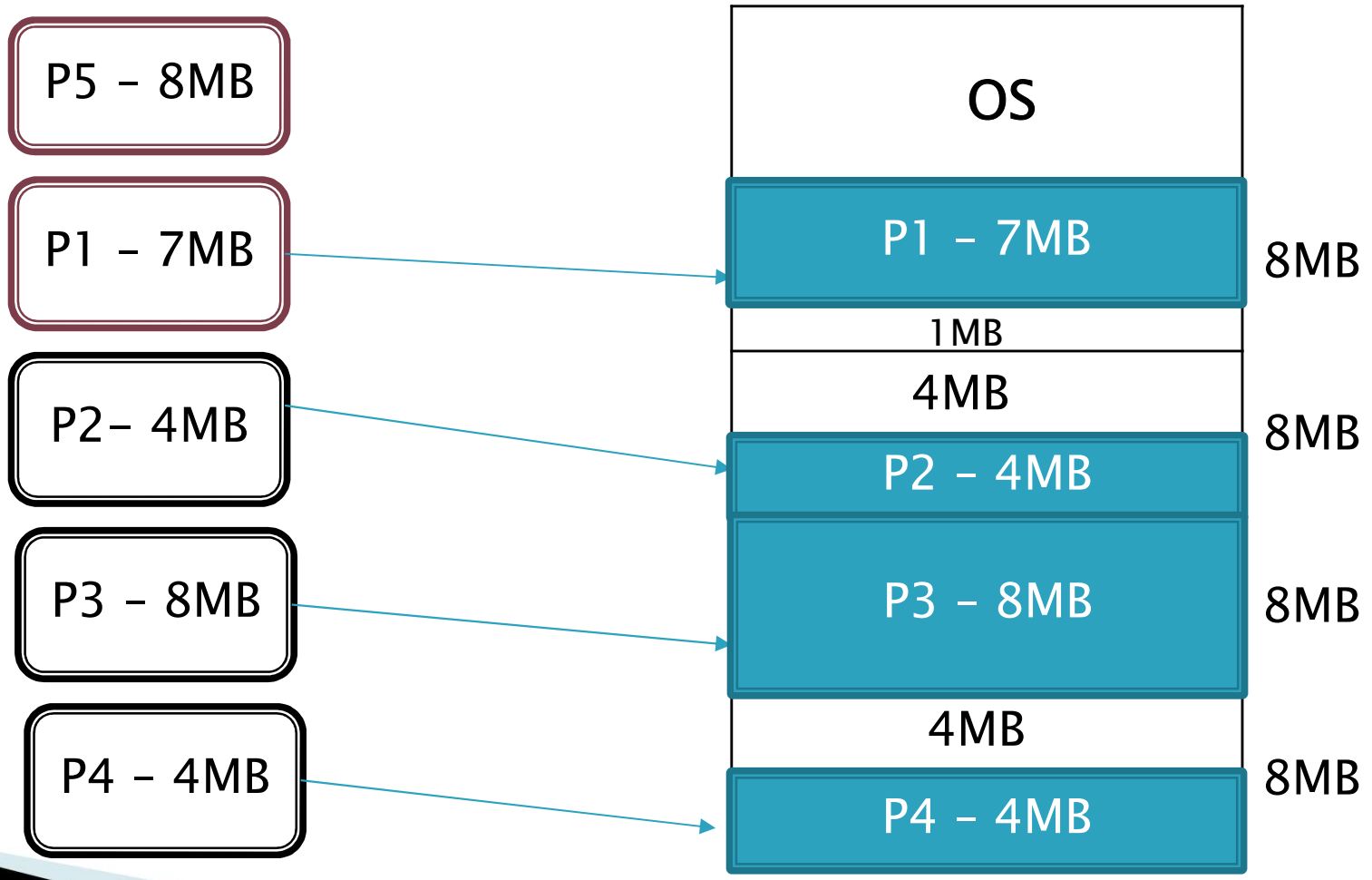
Fixed (static) Partitioning Scheme

- Number of partitions are fixed.
- Partitions are made before the execution or during system configure.
- Size of each partition may or may not be same.
- Hence, we have two types to fixed partitioning:
  - Equal size partitions
  - Unequal size partitions
- Contiguous allocation of memory is done and hence spanning is not allowed.

Fixed Partitioning – Equal size

External Fragmentation

Internal Fragmentation



Fixed Partitioning – Un-equal size

- Better memory utilization than equal size partition.
- Processes of different sizes can be accommodated.
- Here also we see that 16Mb blocks have 8MB unused space causing Internal Fragmentation
- We have 20 MB space available if we add all un-used spaces.
- A new process P6 with size 16MB cannot be allocated space causing External Fragmentation.

Fixed (static) Partitioning Scheme

- **Equal size partitions** - It divides the main memory into equal number of fixed sized partitions.
  - Memory use is inefficient, i.e., block of data loaded into memory may be smaller than the partition causing **internal fragmentation**.
  - Any process whose size is bigger than partition size cannot be loaded.

- **Un-equal Size Partitions** - It divides the main memory into un-equal sized partitions.
  - Efficient memory usage than equal size partitioning.
  - Still suffers from internal fragmentation.

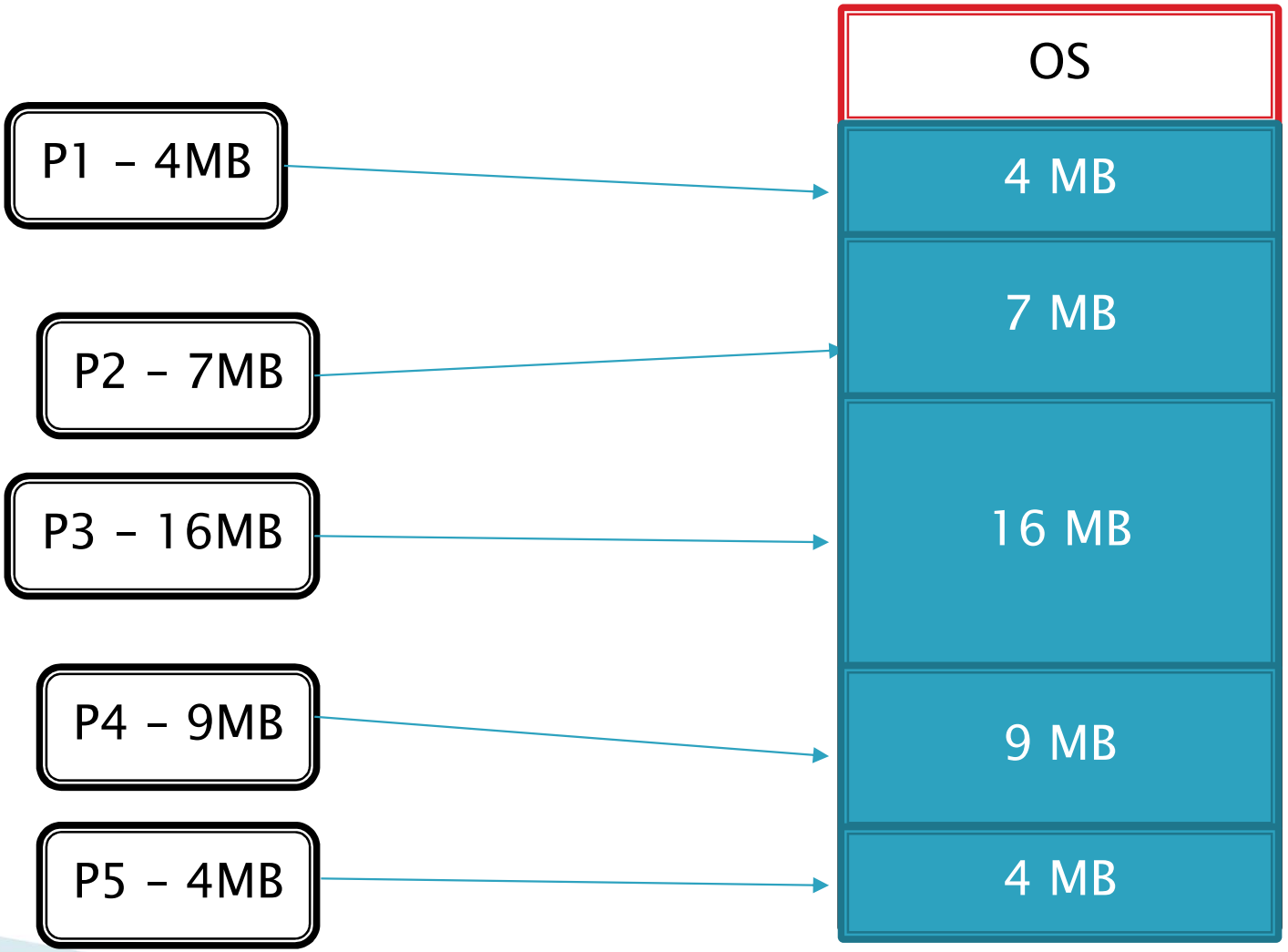
Disadvantages of Fixed Partitioning scheme

- Internal fragmentation - Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.
- Limit in process size as the memory blocks are of fixed sizes.
- Limitation on degree of multi-programming – number of partitions are fixed and hence there is hard limit to number processes that can be loaded into the memory.
- External Fragmentation - Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.

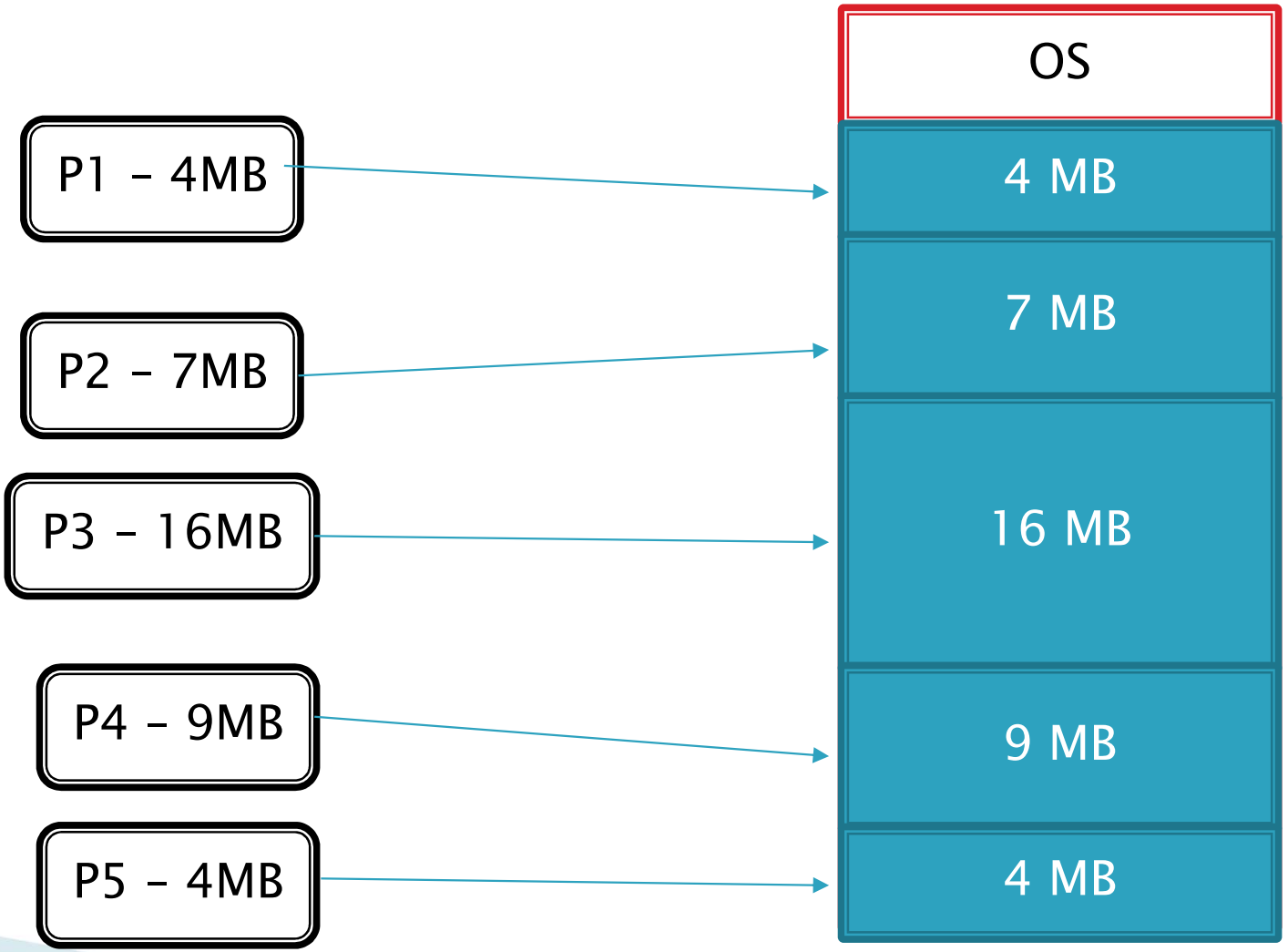
Variable (dynamic) Partitioning Scheme

- Partitions are not made before the execution or during system configure.
- The size of partition will be equal to incoming process.
- The partition size varies according to the need of the process so that the internal fragmentation can be avoided to ensure efficient utilization of RAM.
- Number of partitions in RAM is not fixed and depends on the number of incoming process and Main Memory's size.

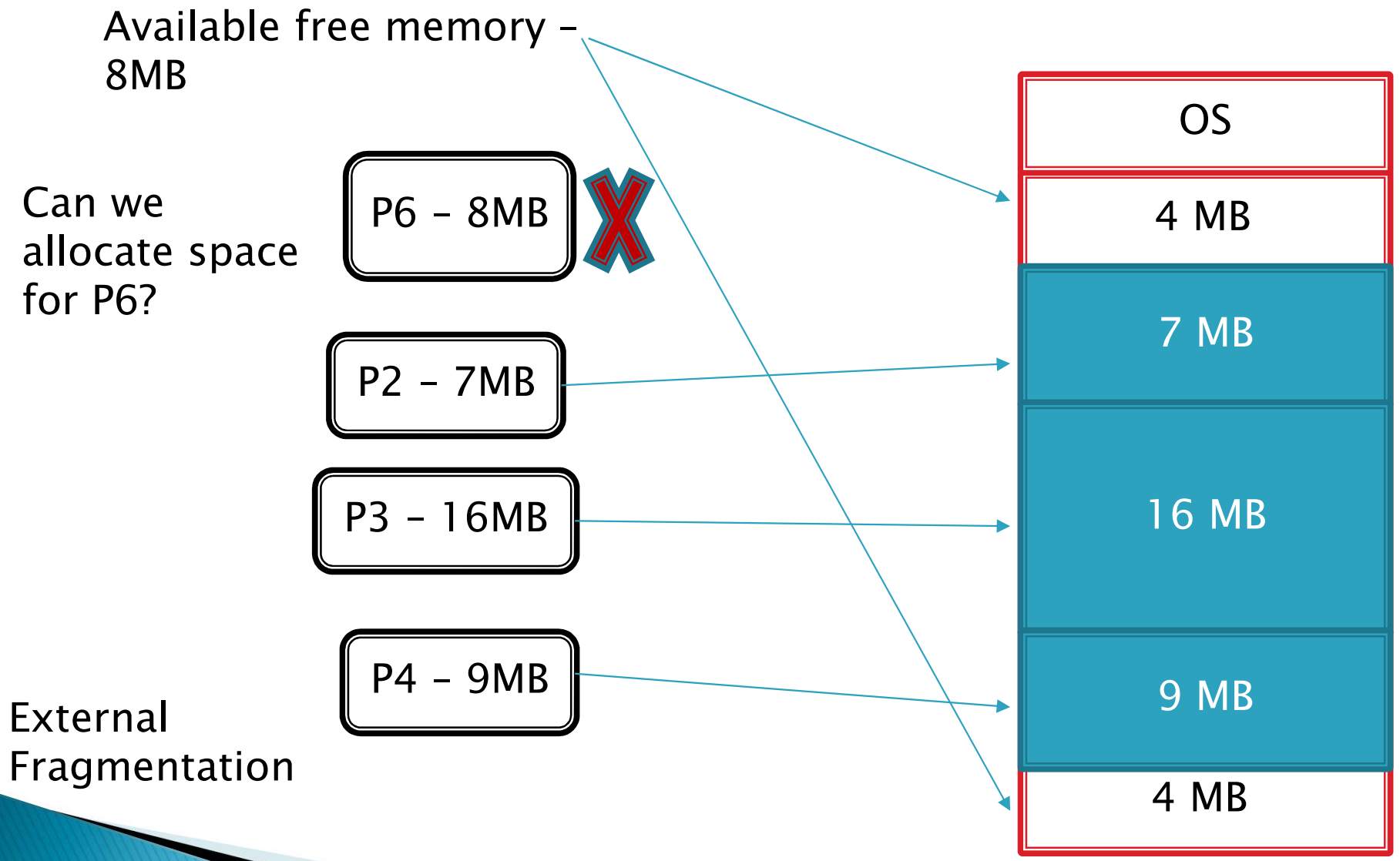
Variable Partitioning Scheme



Variable Partitioning Scheme



Variable Partitioning Scheme

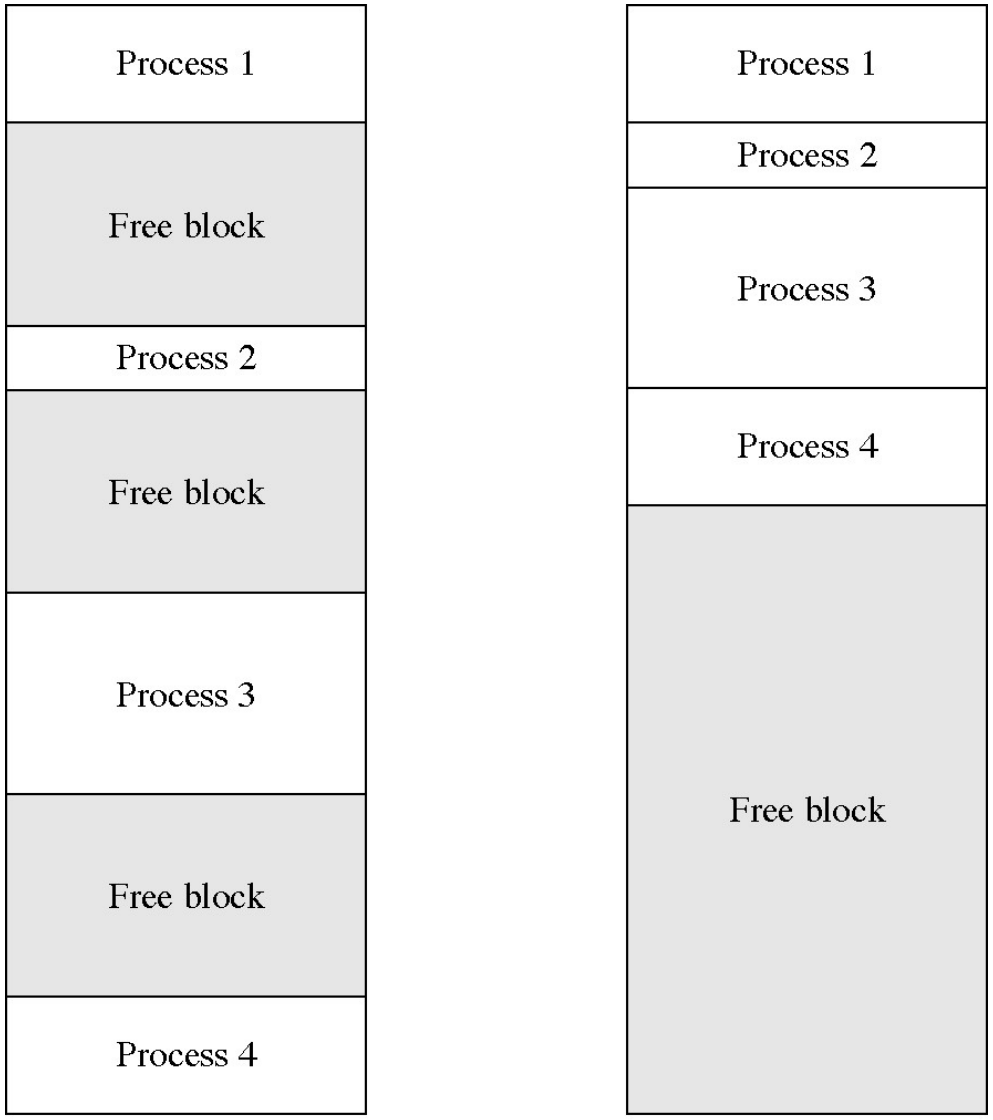


Variable Partitioning Scheme

- Advantages
  - No Internal Fragmentation
  - No restriction on Degree of Multiprogramming
  - No Limitation on the size of the process

- Disadvantages
  - Difficult Implementation: it involves allocation of memory during run-time rather than during system configure.
  - External Fragmentation: There will be external fragmentation inspite of absence of internal fragmentation.

Compacting memory



Before compaction

After compaction

Memory allocation strategies

How to satisfy a request of size *n* from a list of free holes

- First fit
- Best fit
- Worst fit

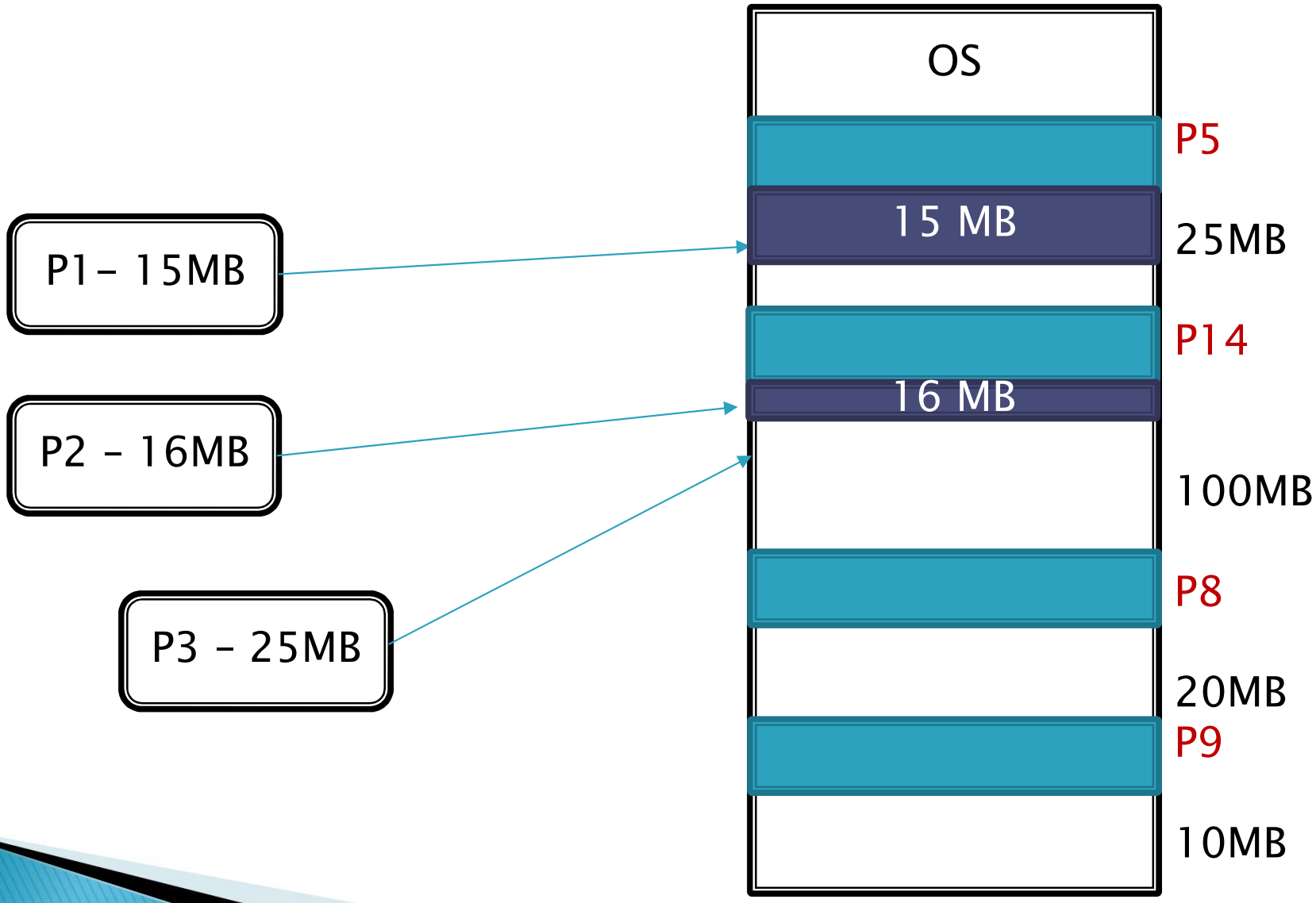
First Fit

First Fit Memory Allocation

- In this method, first job claims the first available memory with space more than or equal to it's size.
- The operating system doesn't search for appropriate partition but just allocate the job to the first memory partition available with sufficient size.

- Advantages
  - It is fast in processing.
- Disadvantages
  - Wastage of memory - smaller processes may be allocated larger spaces resulting in in-efficiency

First Fit


OS
P5
15 MB
25MB
P1- 15MB
P14
16 MB
P2 - 16MB
100MB
P8
P3 - 25MB
20MB
P9
10MB

Best Fit

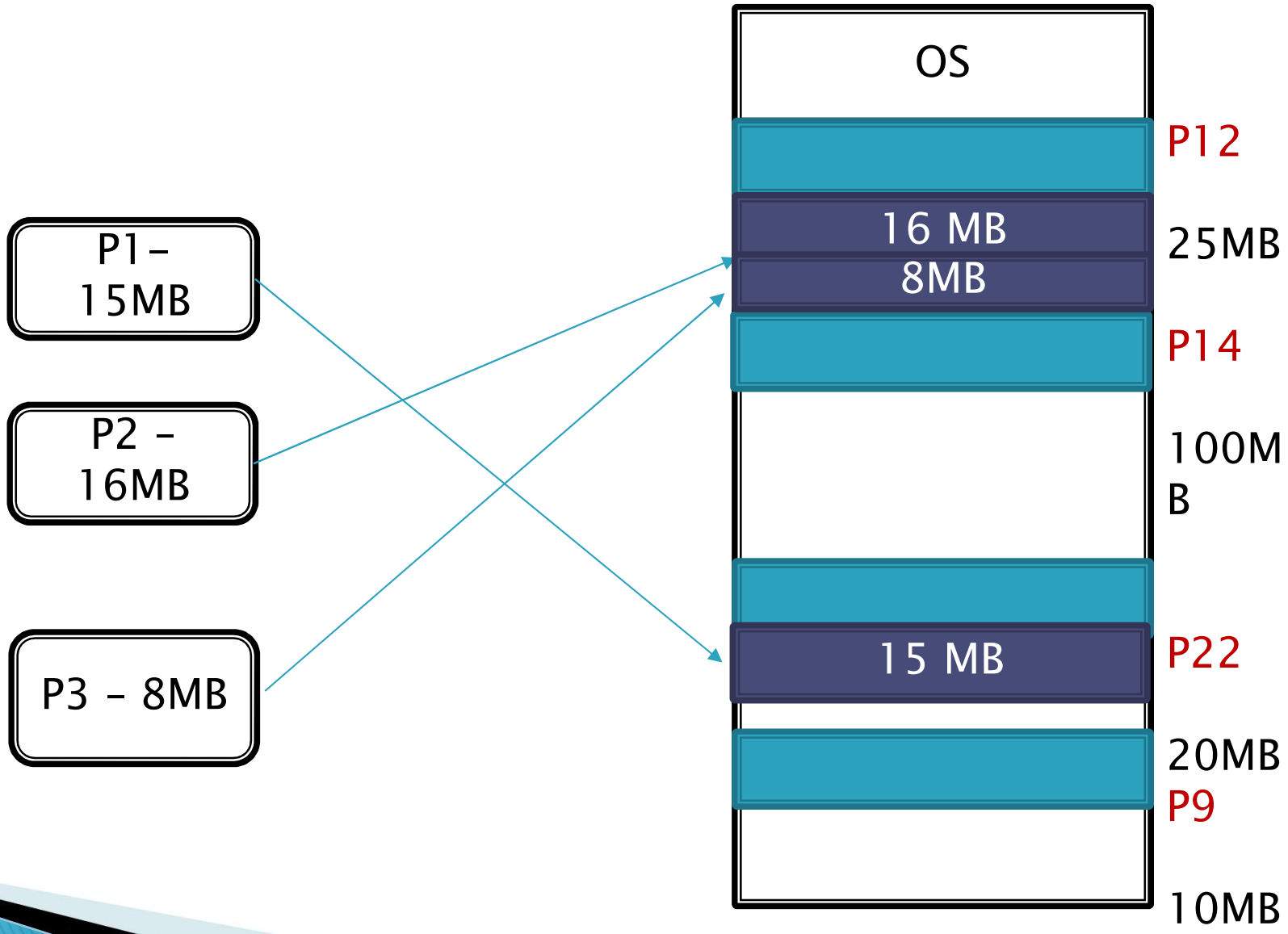
Best Fit Memory Allocation

- The operating system first searches the whole of the memory according to the size of the given job and allocates it to the closest-fitting free partition in the memory.

- Advantages
  - Memory efficient - allocates the process to the minimum possible space available in the memory
  - Minimal internal fragmentation

- Disadvantages
  - Slow Process - Checking the whole memory for each job makes the working of the operating system very slow.

Best Fit


P1 – 15MB
P2 – 16MB
P3 – 8MB
OS
16 MB
8MB
15 MB
P12
25MB
P14
100MB
P22
20MB
P9
10MB

Worst Fit

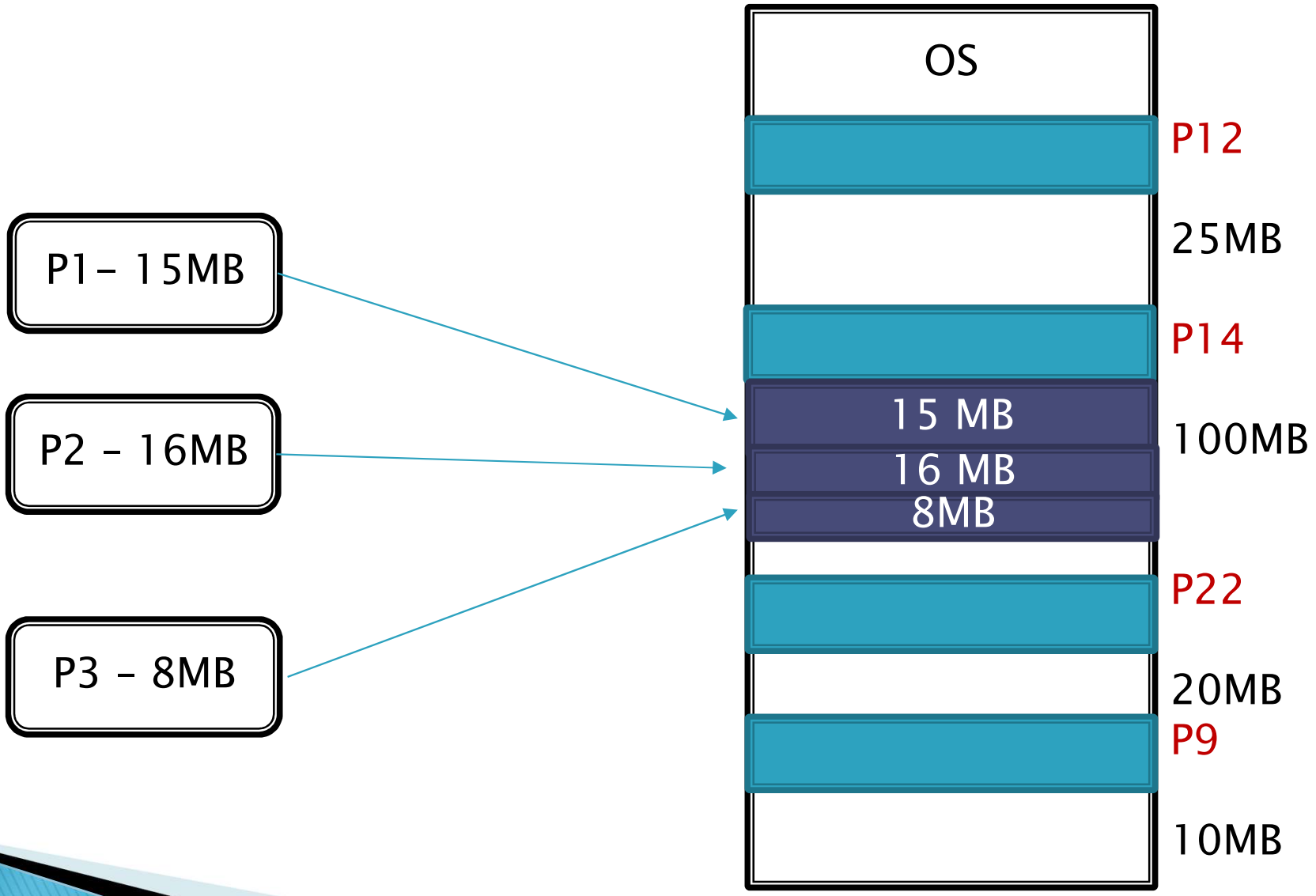
Worst Fit Memory Allocation

- The process traverses the whole memory and always search for the largest hole/partition, and then the process is placed in that hole/partition.

- Advantages
  - Since this process chooses the largest hole/partition, therefore there will be large internal fragmentation. This internal fragmentation is big enough for other small processes to be allocated memory.

- Disadvantages
  - Slow Process - Checking the whole memory for each job makes the working of the operating system very slow.

Worst Fit

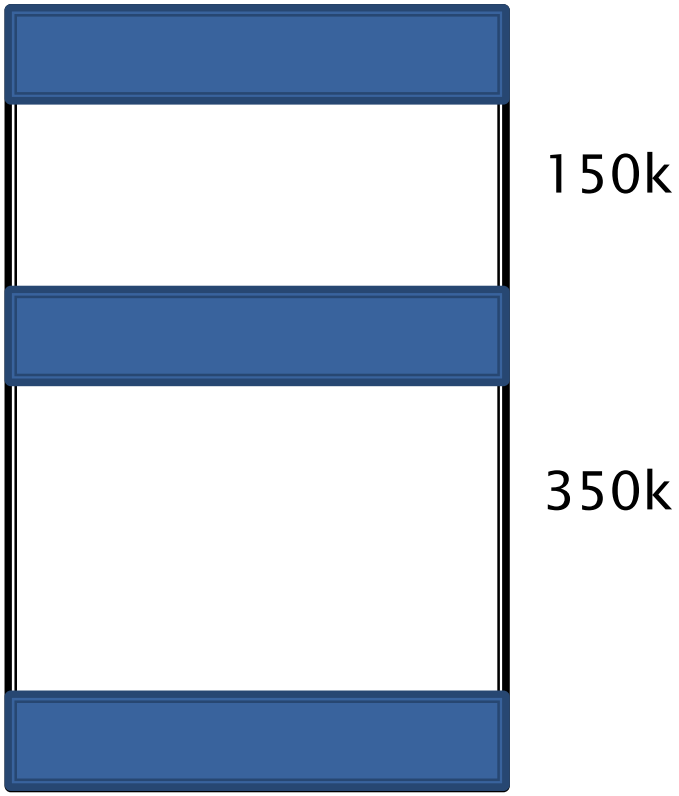

OS
P12
25MB
P14
15 MB
16 MB
8MB
100MB
P22
20MB
P9
10MB
P1- 15MB
P2 - 16MB
P3 - 8MB

Question?

Requests from processes are 300k, 25k, 125k, 50k respectively.

The requests can be satisfied with

A) Best fit but not first fit
B) First fit but not best fit
C) First fit and Worst fit
D) Both
E) None



Questions