Logical Vs Physical Address space

→ Address generated by the CPU is called a logical address.

→ Address seen by the memory unit is called Physical Address.

→ Binding at compile time and at load time generate same logical & physical address.

→ Binding at run time generates different logical & physical addresses.

→ In later case

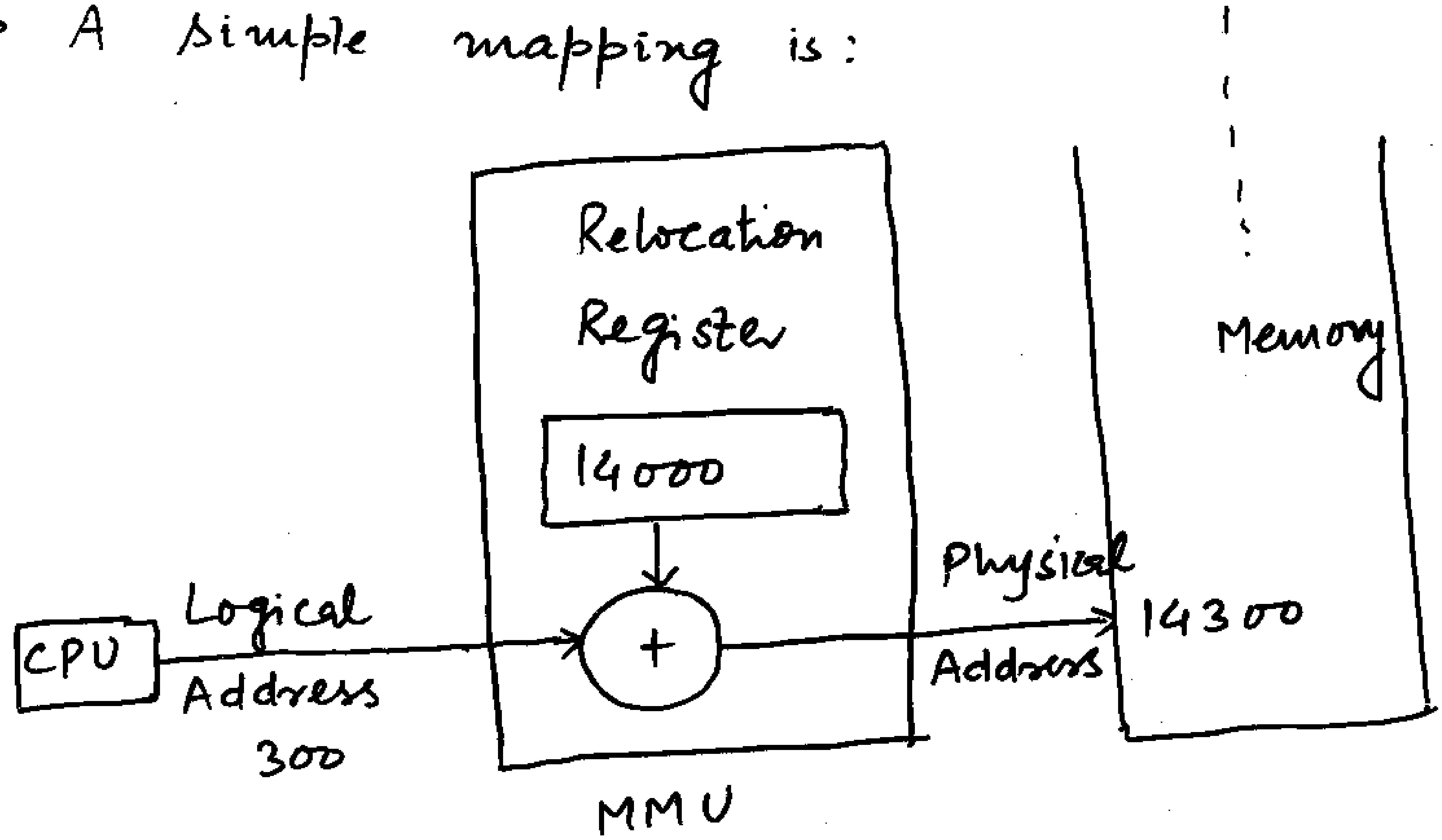
- logical address is called Virtual Address.
- P

→ Set of all logical addresses generated by the program is called Logical Address space.

→ Set of all physical addresses corresponding to logical addresses is called Physical Address space.

→ Run Time mapping from virtual to physical address is done by Memory Management Unit (MMU).

→ A simple mapping is:

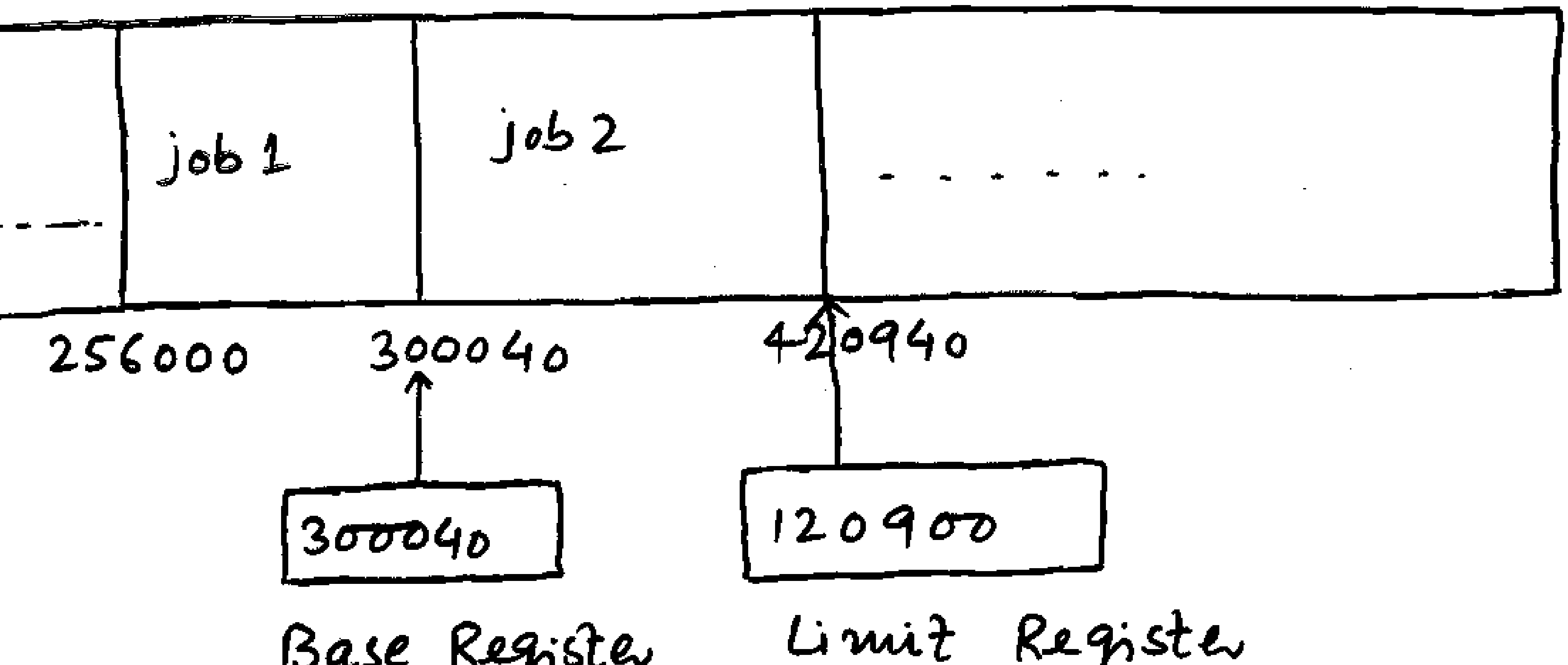


→ Content of relocation register is added to the address generated by CPU.

→ Logical address space for a user program has bounds as 0 and max.

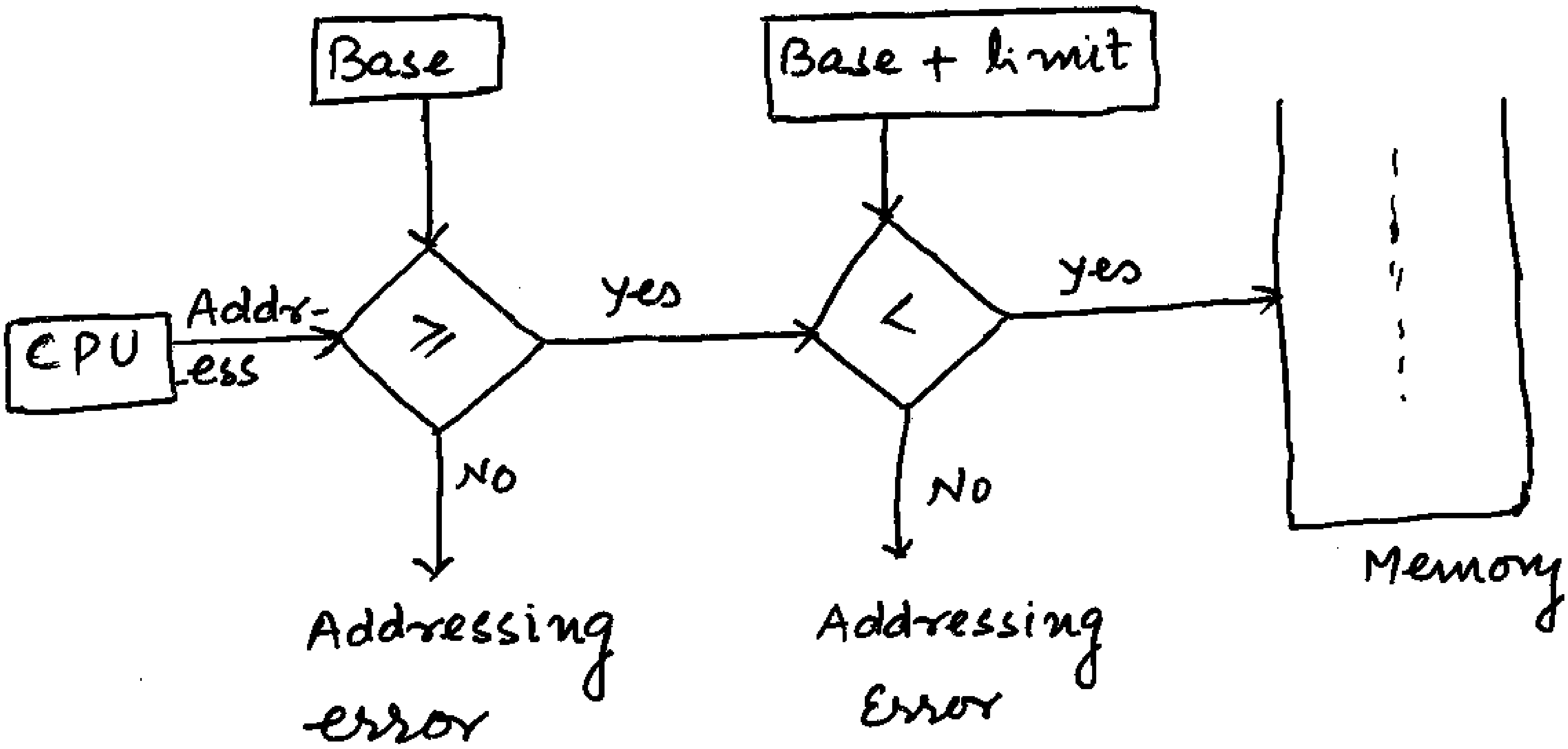
→ Physical address space for a user program has bound R+0 & R+ max.

R : Content of relocation / base register.



→ Base & limit registers define logical address space

Address protection with base & limit Registers

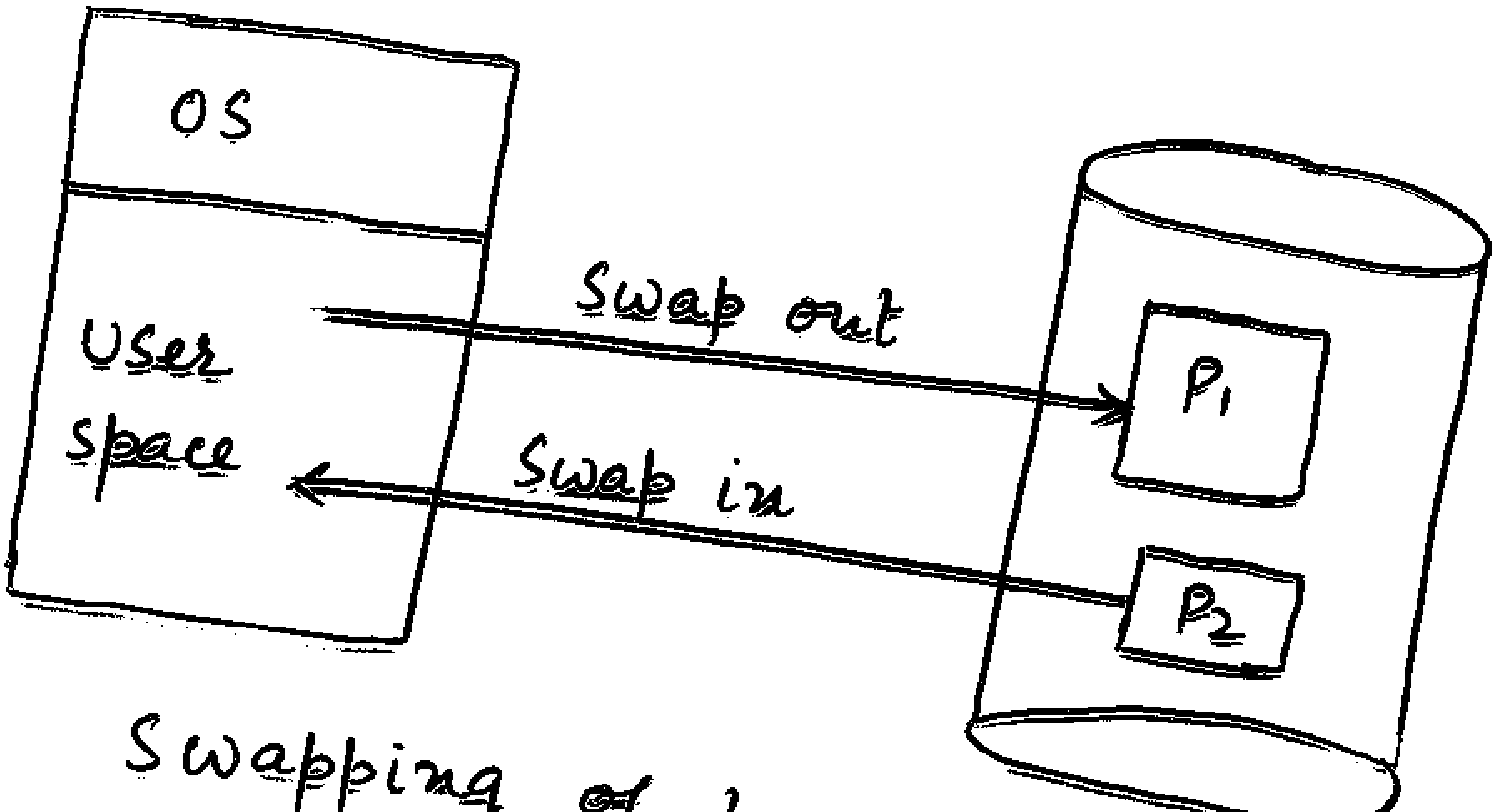


→ Any illegal access of address is prohibited.

Swapping

⇒ A process may be swapped temporarily out of memory to backing store, & then brought back into memory for continued execution.

⇒ Used in multiprogramming with round robin CPU scheduling.



Swapping of two processes

A process is swapped with another process if it finishes its quantum.

A variant of this swapping policy is priority based scheduling.

A higher priority job swaps out a lower priority job.

→ Context switch time in a swapping system is fairly high.

Assume a user process of 100 K.

Backing store is a standard hard disk with transfer rate of 1 M bytes/sec.

Actual transfer of 100 K process from or to memory takes :

$$100K / 1000K = 1/10 \text{ sec}$$

$$= 100 \text{ milli second.}$$

If average latency of 8 milli second, total swap time is 108 m sec.

Since both swap in & swap out are performed total time : 108 × 2 = 216 m sec.

→ So execution time should be fairly large than swap time for efficient utilization of CPU.

Contiguous Allocation

⇒ Main memory is usually divided into two partitions
- One for operating system
- One for user processes.

⇒ Operating system is placed in lower contiguous memory regions.

⇒ User processes execute in higher contiguous memory regions.

Partitioned Allocation

User space of main memory is chunked into fixed size blocks called partitions.

Each partition may exactly has one process.

Degree of multiprogramming is bounded by no. of partitions.

A variant of this technique further enhances degree of multiprogramming.

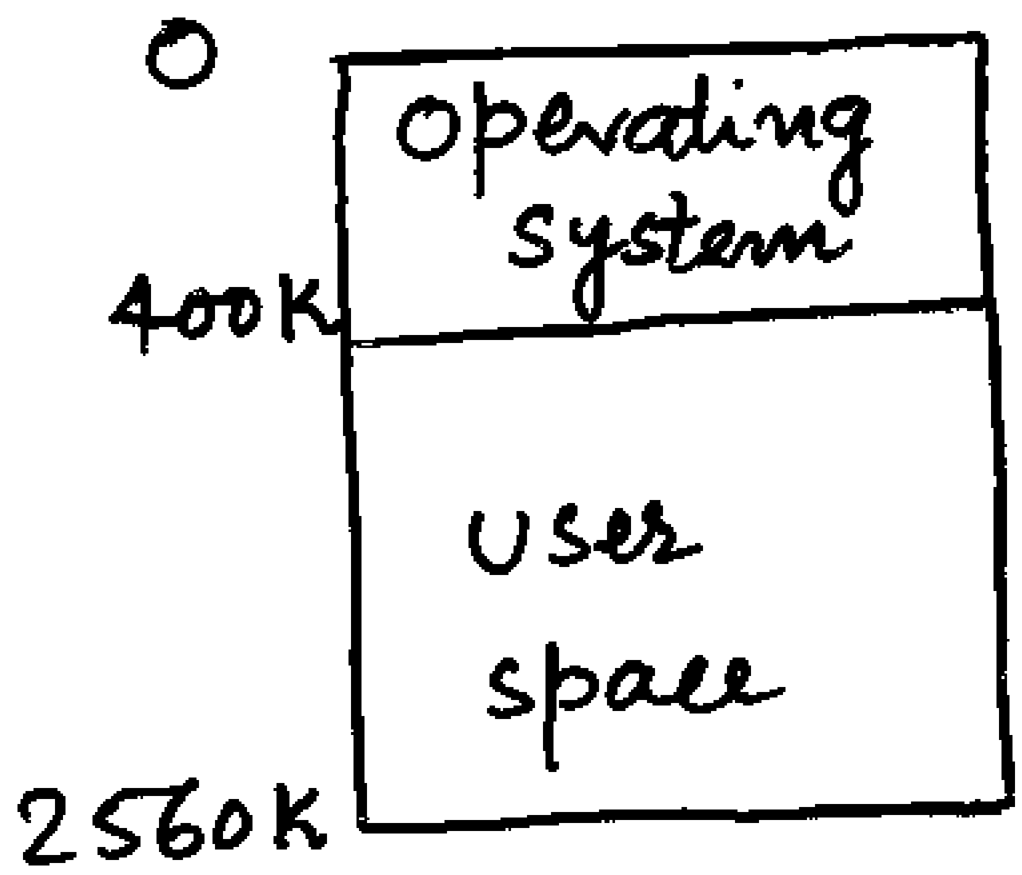
Variant Partition Allocation

→ Whole user space is considered as a large block of available memory, called a hole.

→ When a process arrives, we search for a hole large enough for this process, & allocate.

→ Rest is available to satisfy further requests.

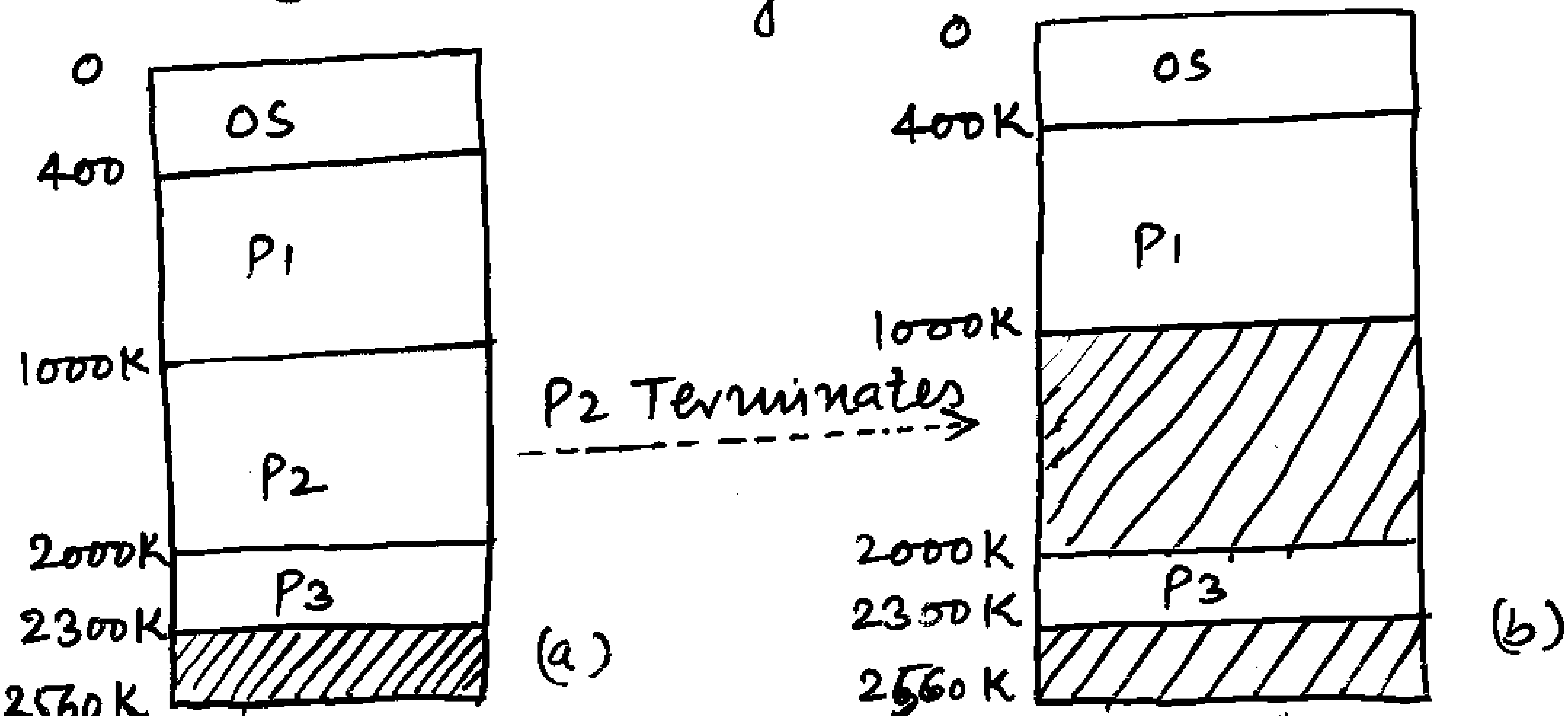
Example:

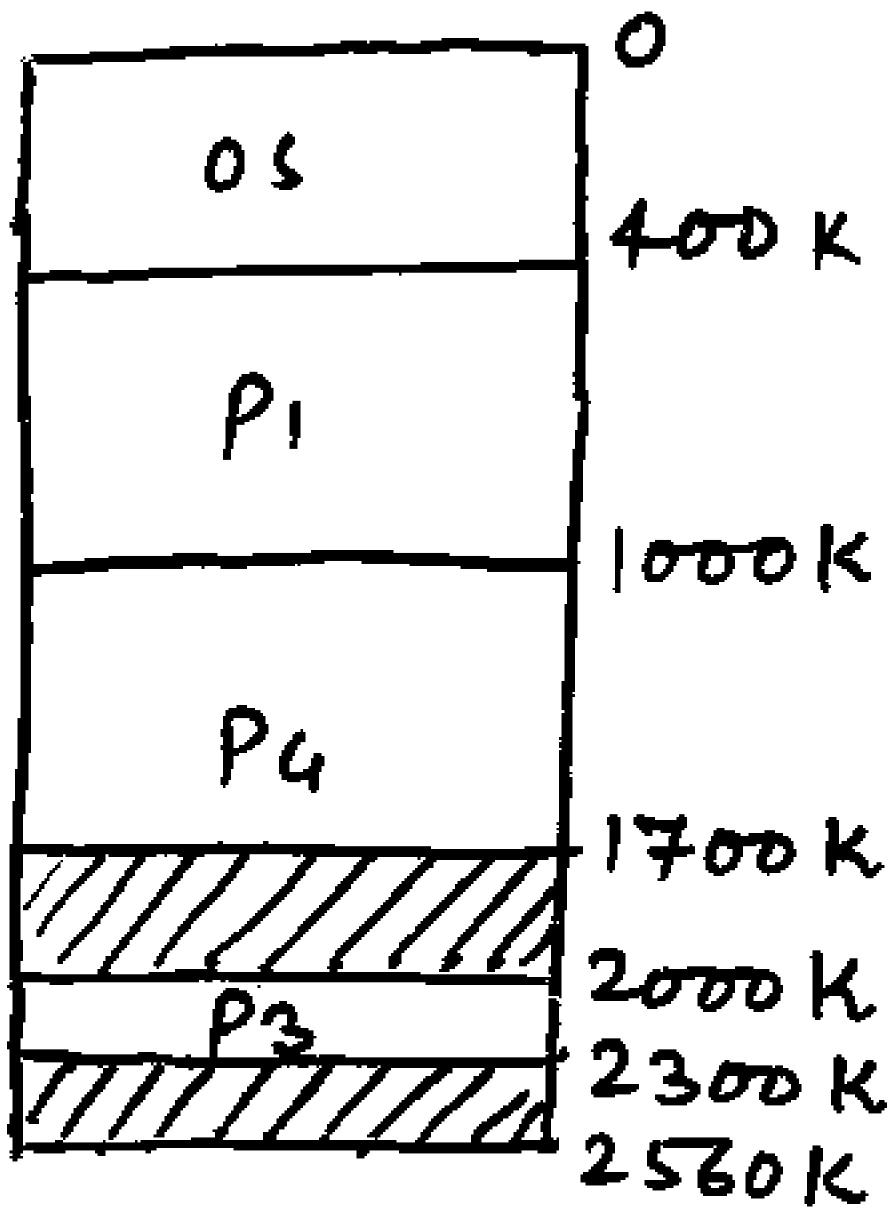


Job Queue

Process	Memory	Time
P_1	600 K	10
P_2	1000 K	5
P_3	300 K	20
P_4	700 K	8
~~P5~~	~~500 K~~	~~15~~

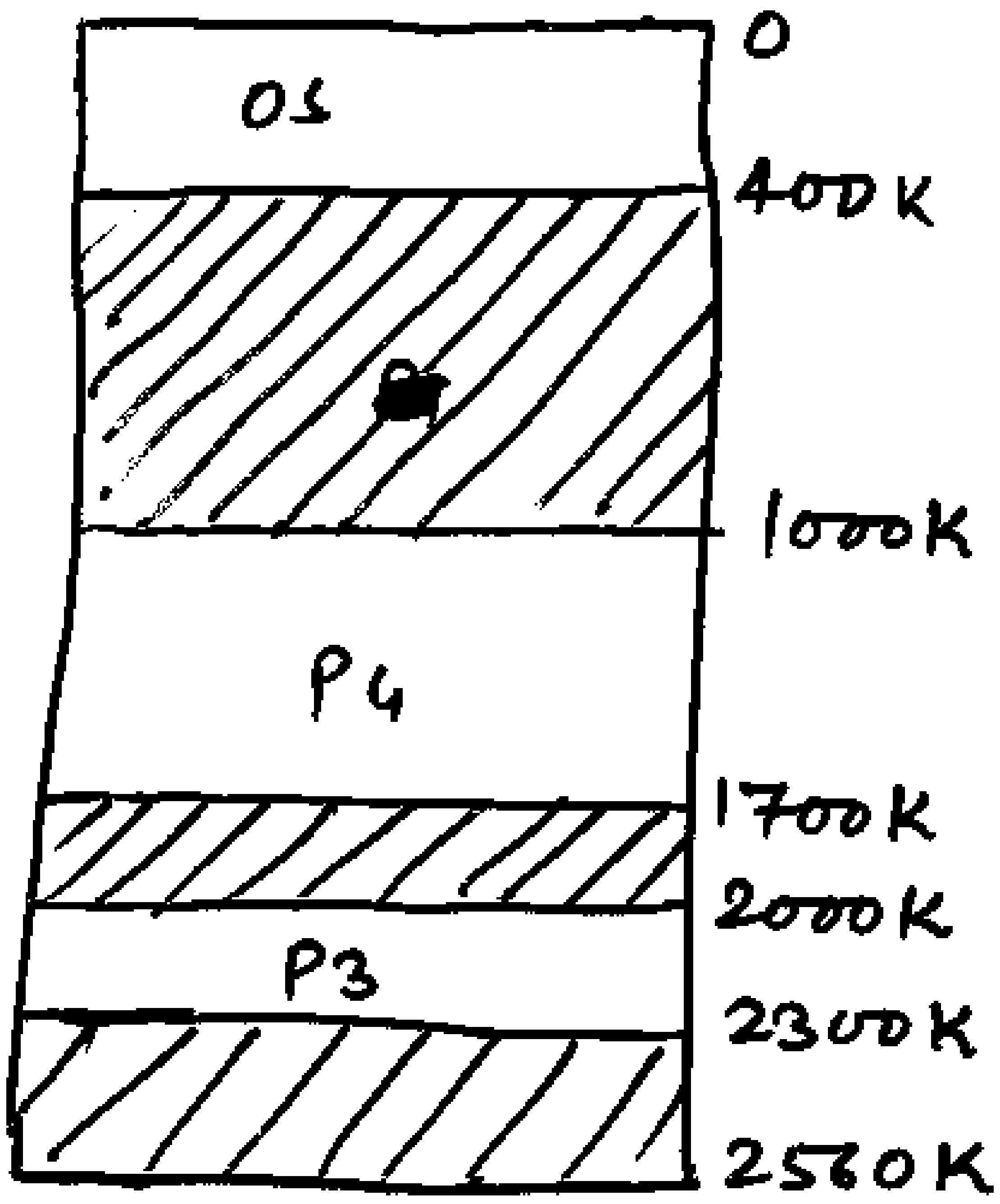
Used job scheduling is First Come First Serve.





(c)

P4 has been allocated



(d)

P1 terminates.

→ Procedure is a particular instance of general dynamic storage allocation problem which is:

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

→ Three strategies are commonly used to support dynamic storage allocation problem:

- First fit
- Best fit
- Worst fit

First Fit:

- Allocate the first hole that is big enough.
- Searching can start either at the beginning or where the previous first fit search ended.

Best fit:

- Allocat the smallest hole that is big enough.
- We must search the entire list, if list is not ordered by size.
- Allocates smallest leftover hole.

Worst fit:

- Largest possible hole is allocated.
- Search entire list unless sorted by size.
- strategy produces largest leftover hole.

Fragmentation

→ Two types of fragmentation:
- External fragmentation.
- Internal fragmentation.

External fragmentation

→ When enough total memory space exists but it is not contiguous to satisfy a request.

→ storage is fragmented into large no. of small holes.

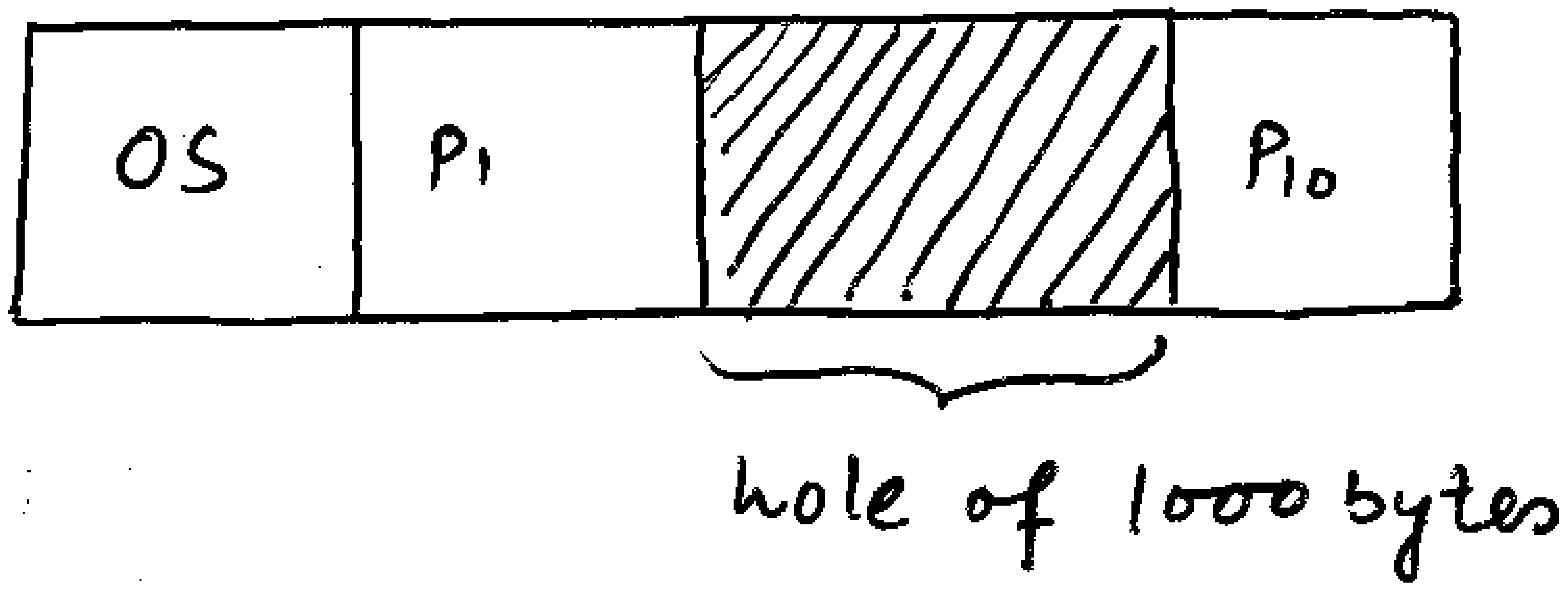
→ statistical analysis shows that given N blocks allocated, another 0.5 N blocks will be lost due to fragmentation.

→ That is, One third of memory may be unusable.

→ This property is called 50 percent rule.

Internal Fragmentation

→ Memory fragment that is internal to a partition but is not being used.



→ A process of 990 bytes may be allocated to free partition, causing a loss of 10 bytes.

→ Internal fragmentation is

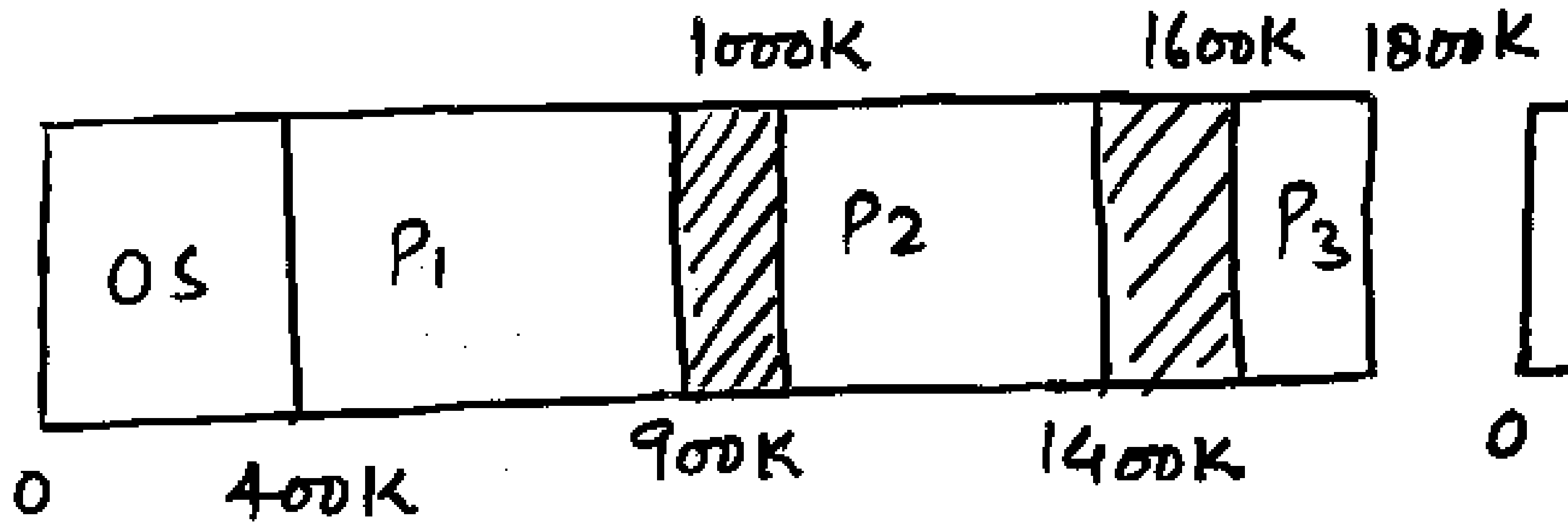
Difference between allocated memory and requested memory.

→ One solution to the problem of fragmentation is compaction.

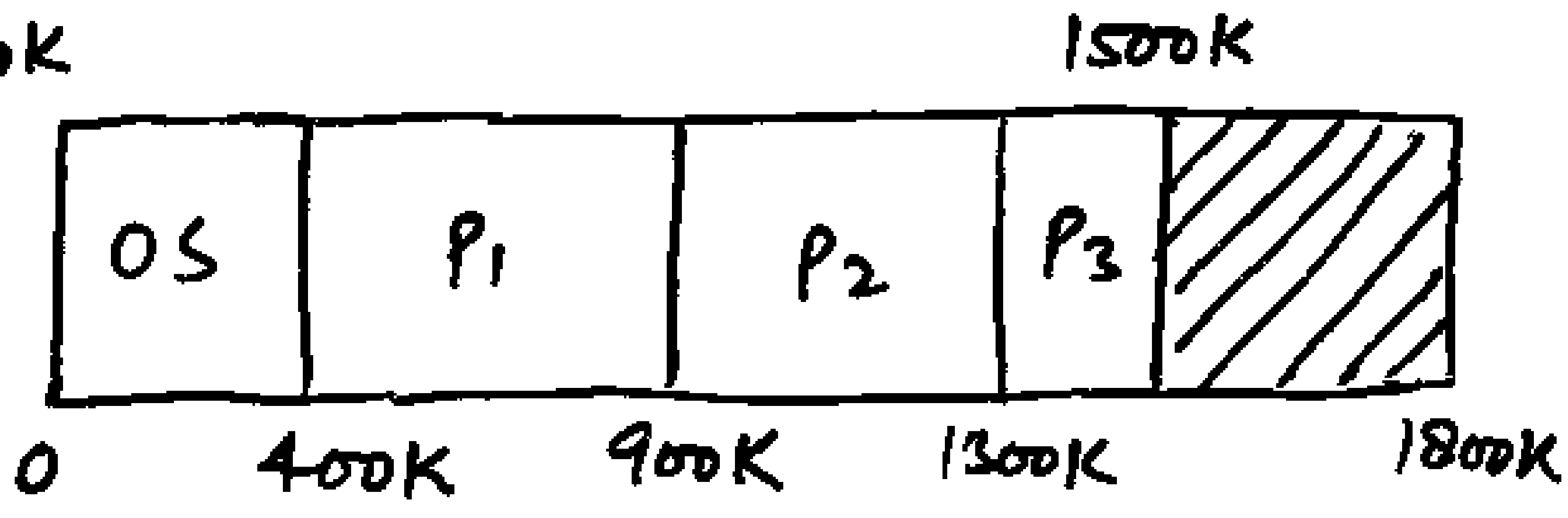
→ In compaction

- goal is to shuffle the memory contents to place all free memory together in one block.

Compaction contd..



Before compaction



After compaction.

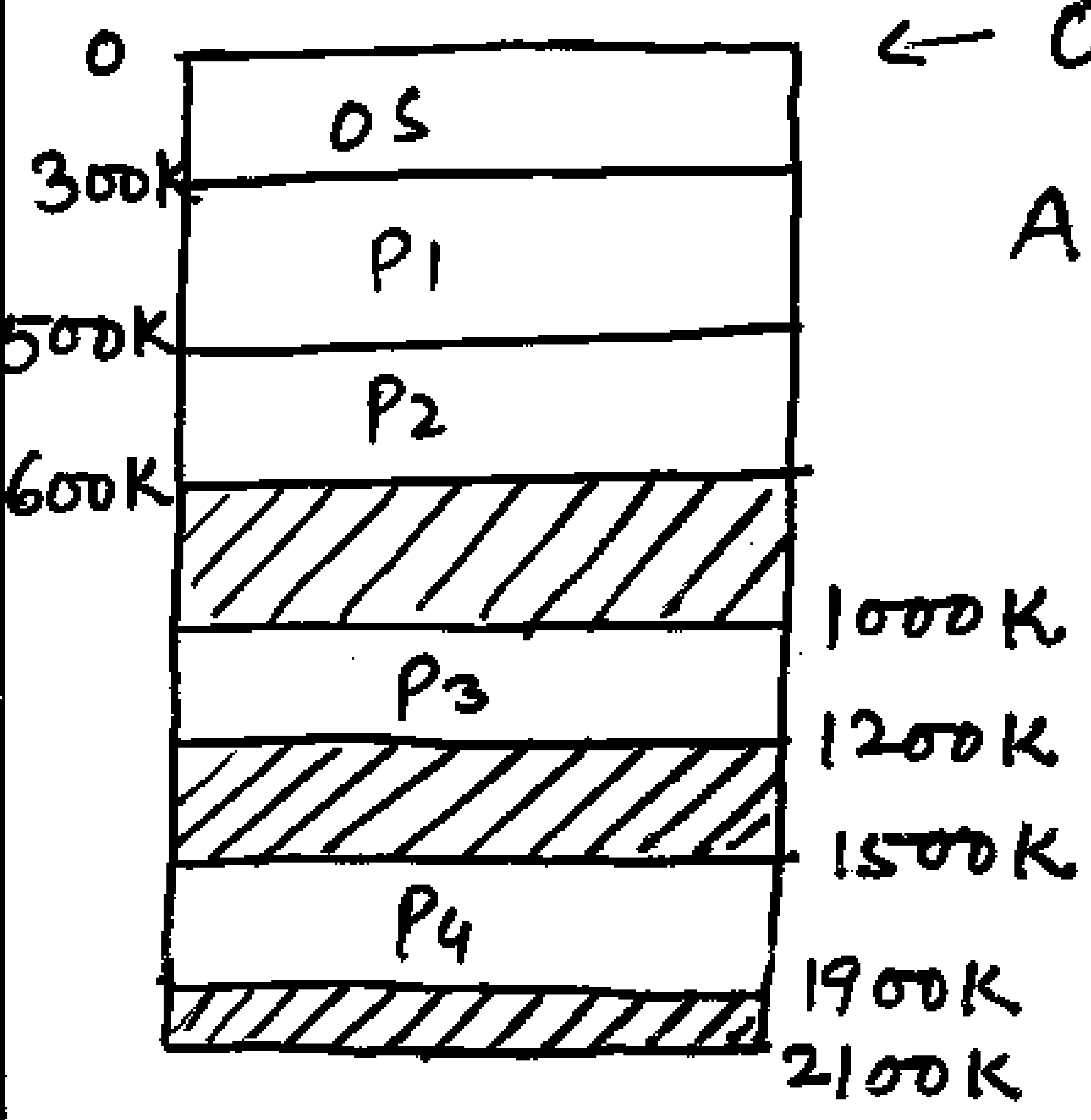
→ Compaction is not always possible:

- Possible, if address allocation is static as in compile/load time binding.

  Not possible, if binding is at execution time.

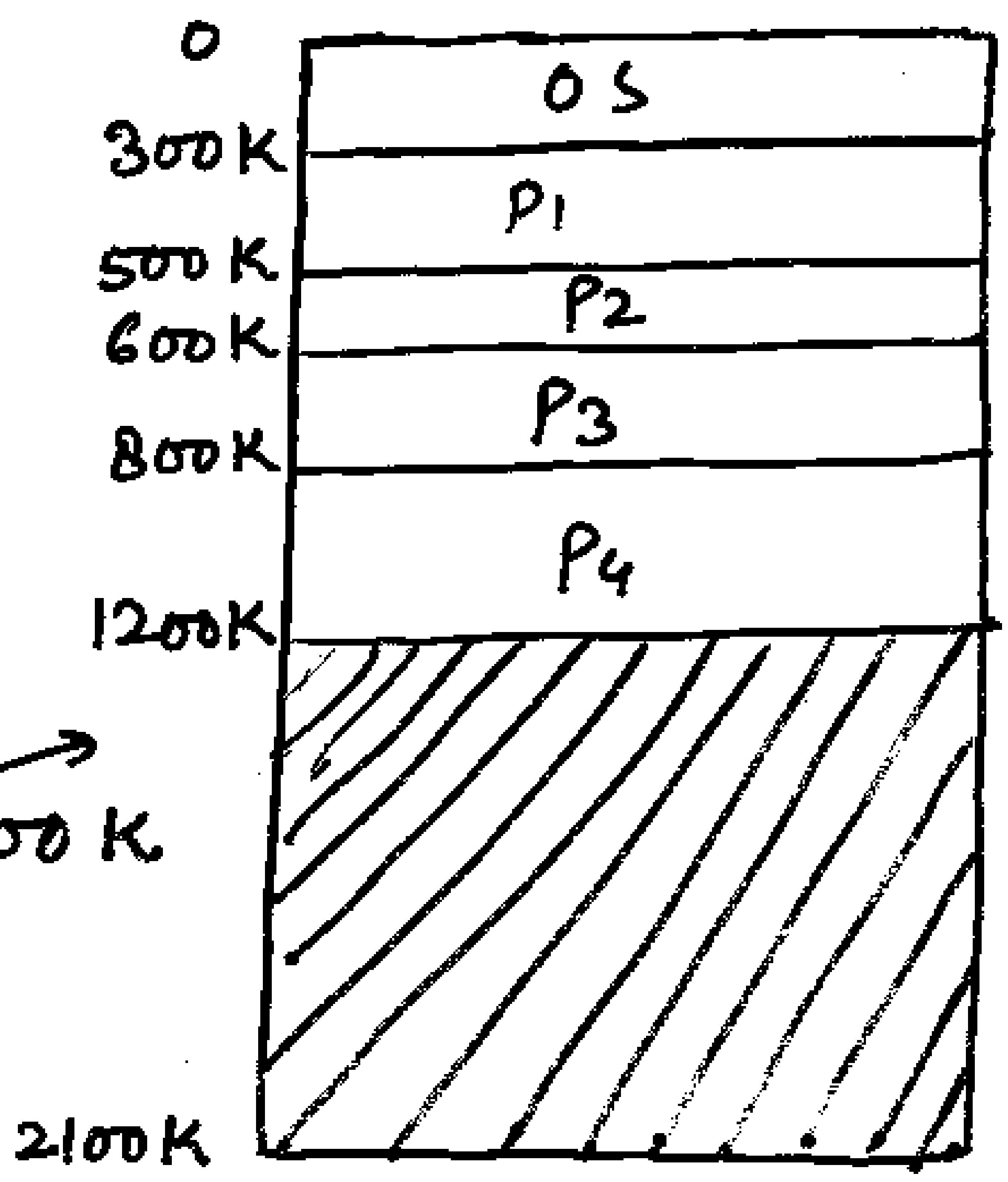
→ Swapping can also be combined with compaction.

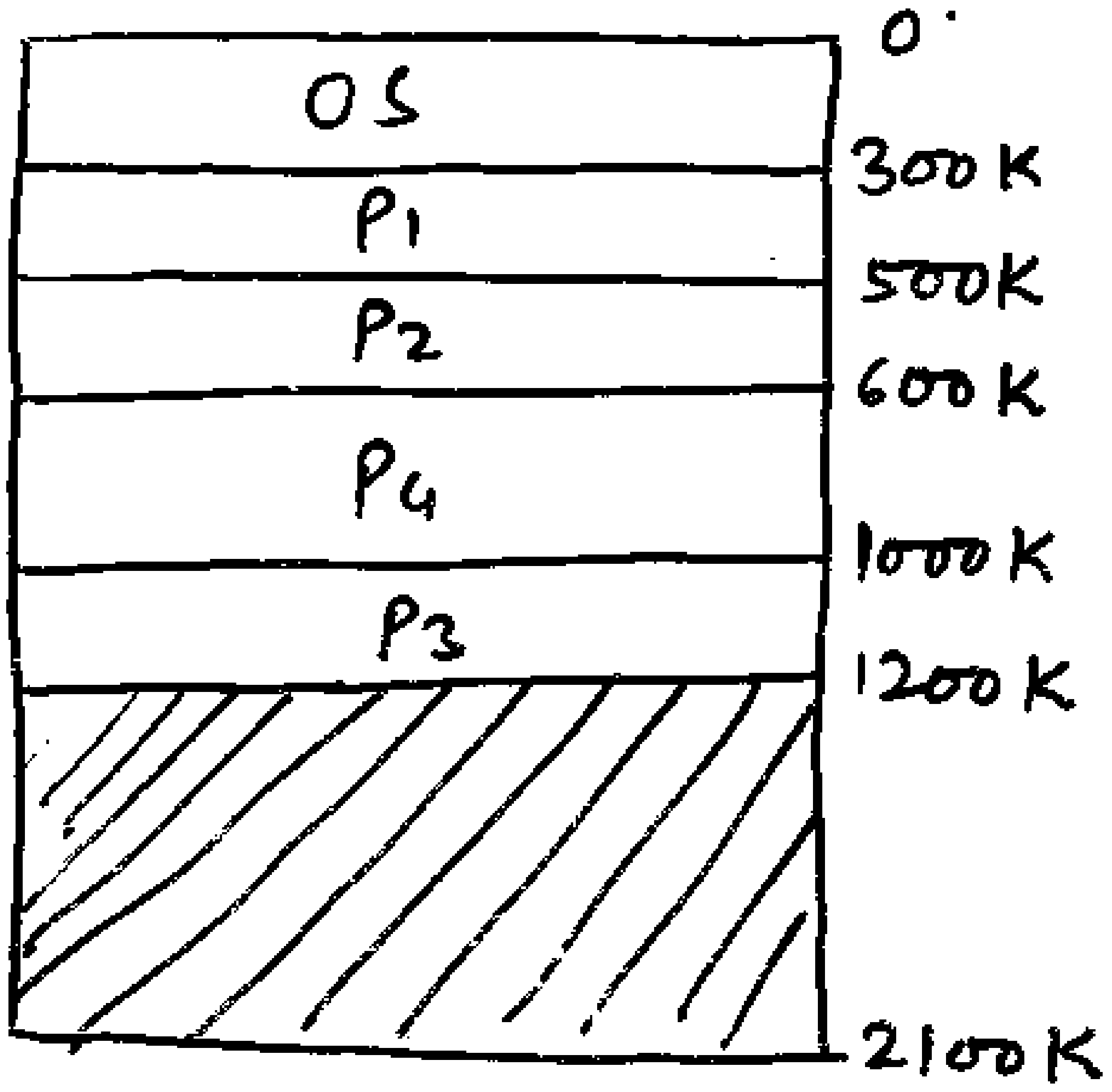
→ When compaction is possible, incurred cost is determined.



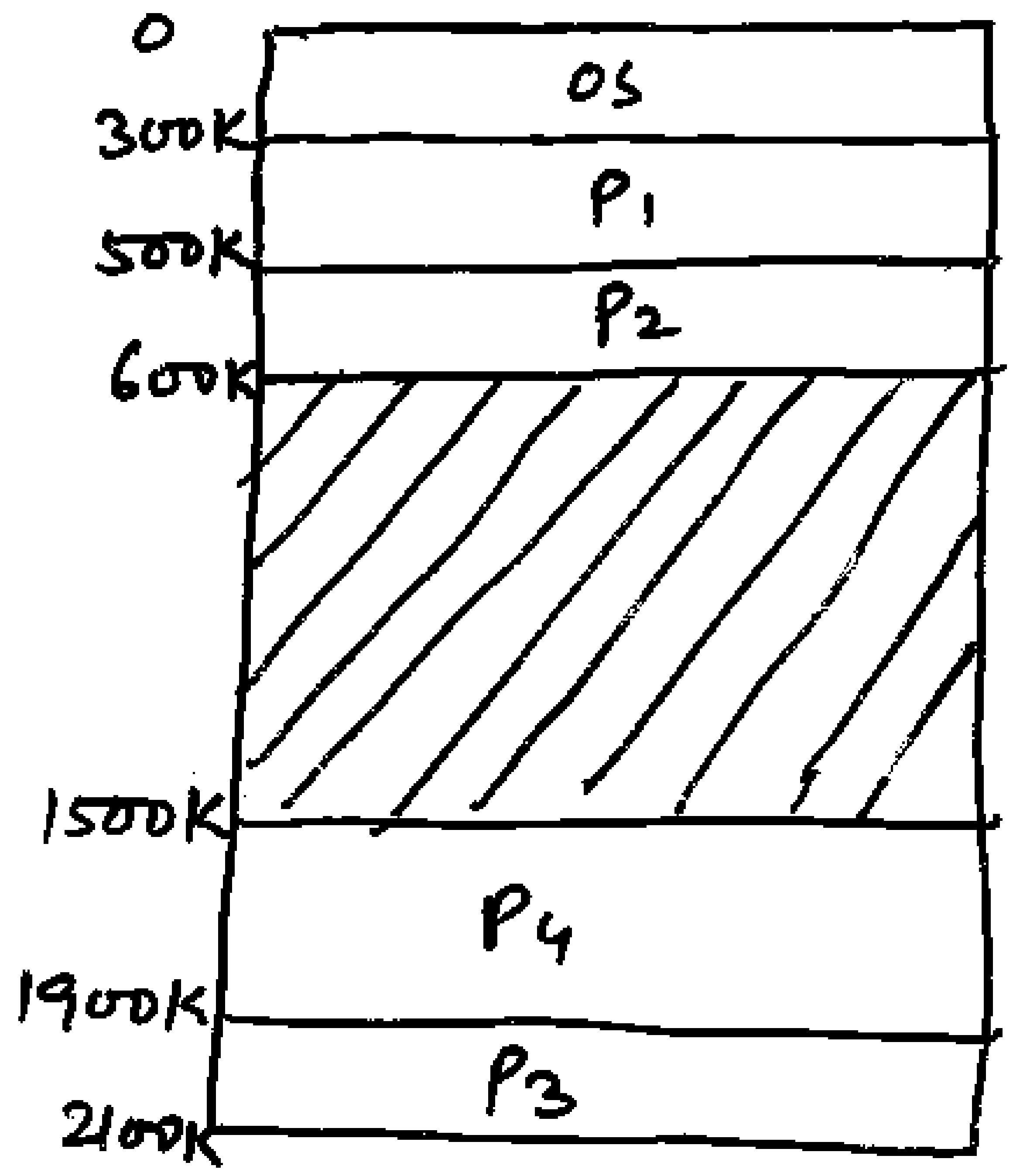
← Original Allocation

Moved 600K data





Moved 400K data



Moved 200K data

Paging

- Fragmentation problem can be solved if logical address space of a process becomes non-contiguous.
- Physical memory is broken into fixed-sized blocks called frames.
- logical memory is broken into blocks of the same size called Pages.
- when a process is executed, its pages are loaded into memory frames.