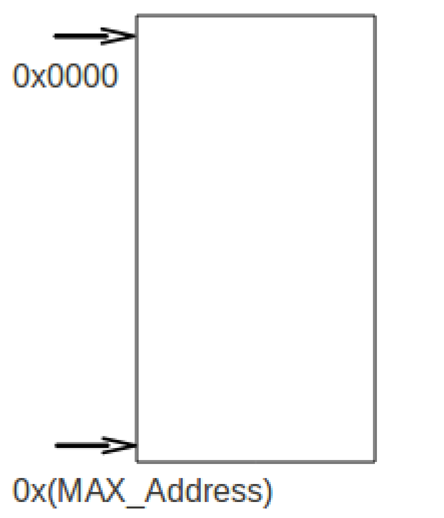
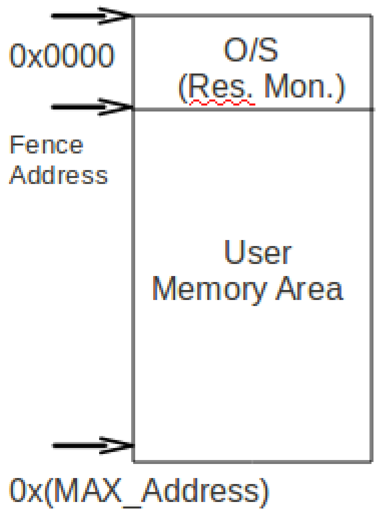
Memory Management (MFT)

2 November 2018
CS303
Autumn 2018

Memory Management



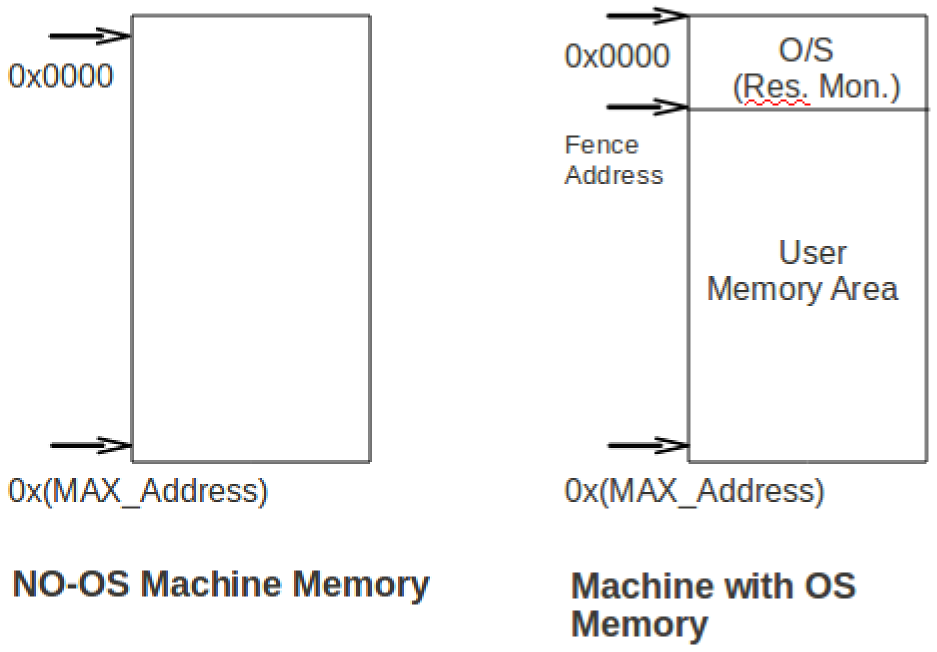
NO-OS Machine Memory



Machine with OS Memory

```
$:> pmap pid
```

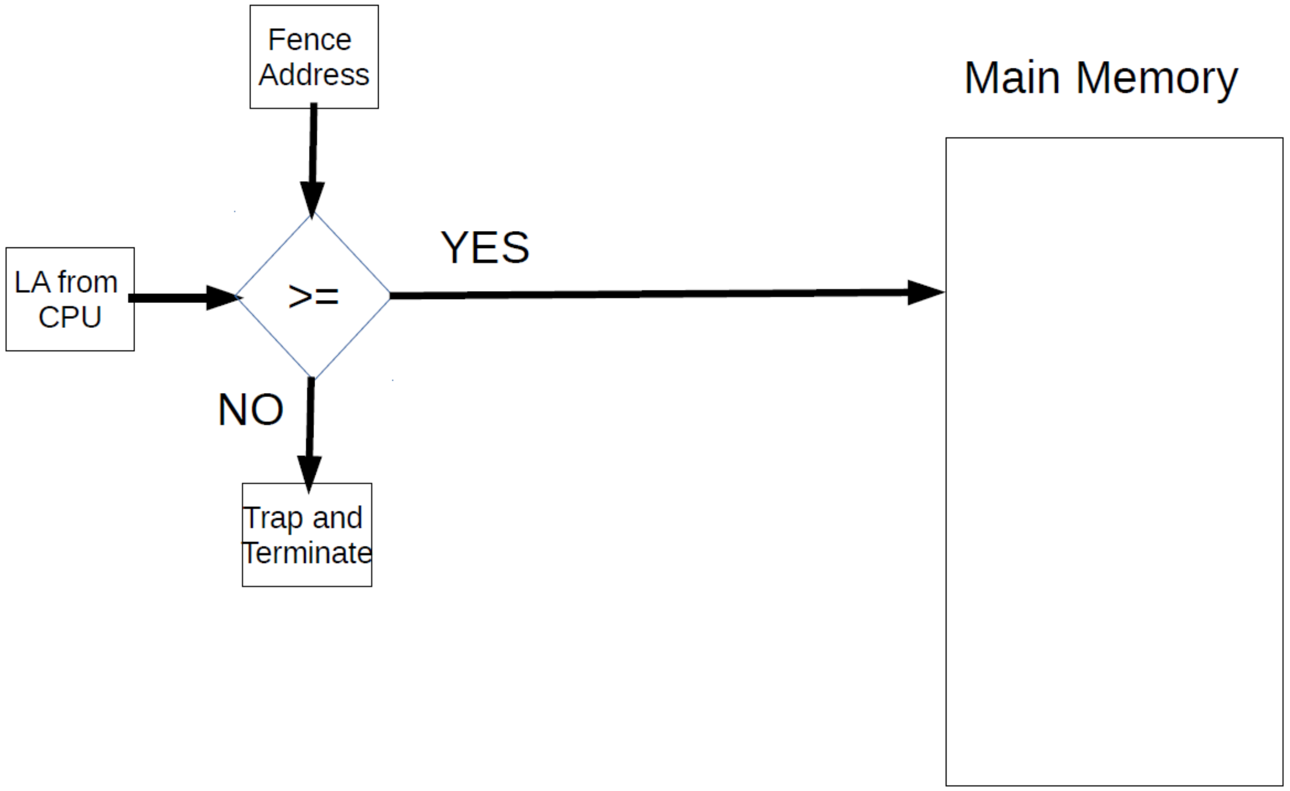
Memory Management



Fence Address (FA): stored in a register
Either Fixed in HW or Editable

1. On every address generated by CPU from a user process, it needs to be compared with the FA
    - this check can be implemented in
        - HW: hardware comparator 'Efficient'
        - SW: software instruction for comparing

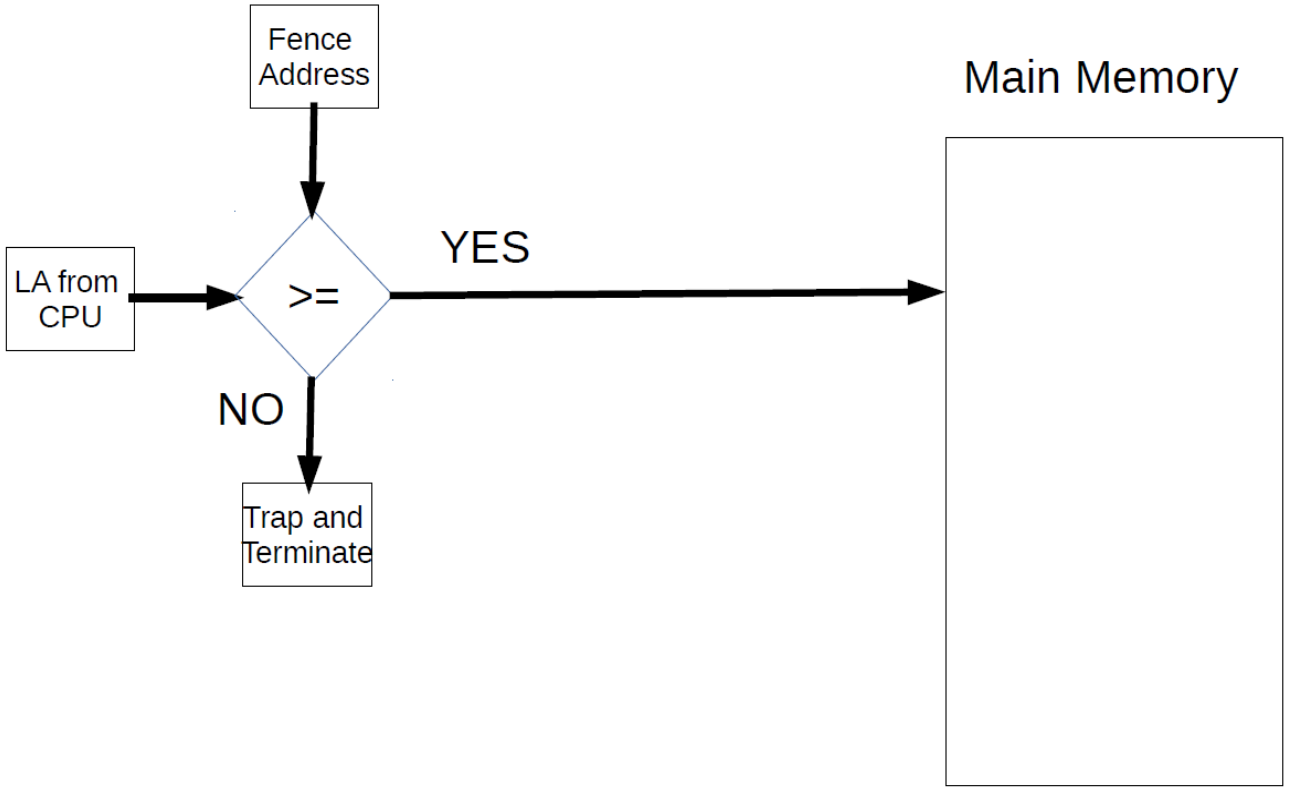
Memory Management



Fence Address (FA): stored in a register
Either Fixed in HW or Editable

1. On every address generated by CPU from a user process, it needs to be compared with the Fence Address
    - this check can be implemented in
        - HW: hardware comparator 'Efficient'
        - SW: software instruction for comparing

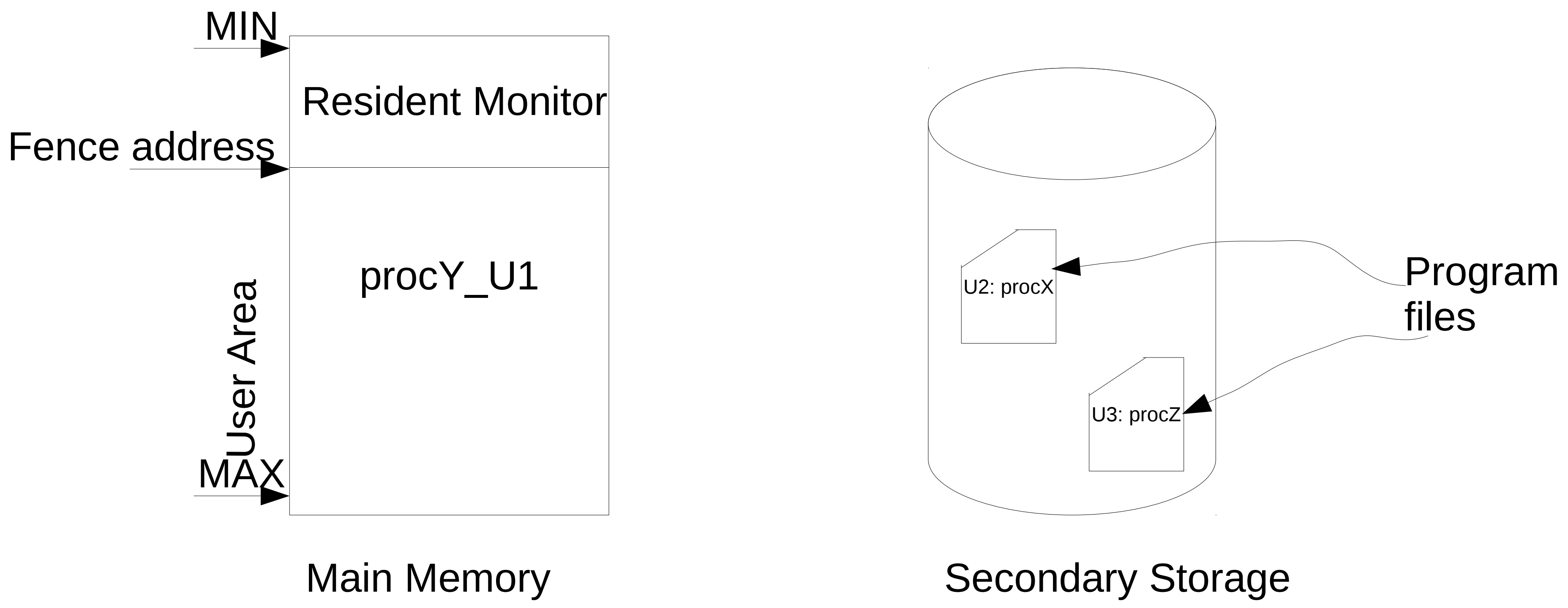
Memory Management



1. Fence Address (FA): stored in a register
   Either Fixed in HW or Editable

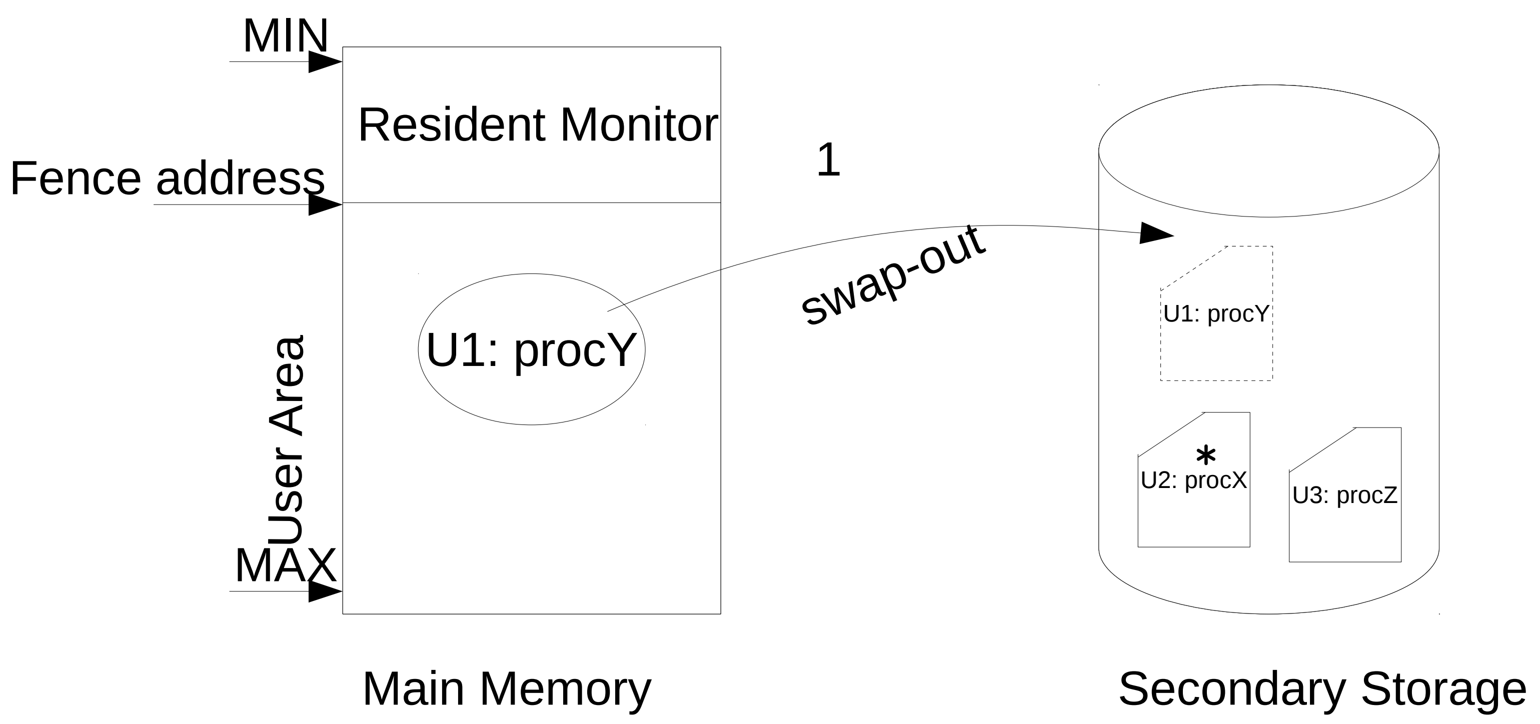
2. The checking requires a comparator, which could be implemented in:
   - HW: hardware comparator 'Efficient'
   - SW: software instruction for comparing

Simple Memory Model: Res. Montior + User Area



1. Single Process gets loaded into the MM from the SS/HD

Simple Memory Model: Res. Montior + Use Area

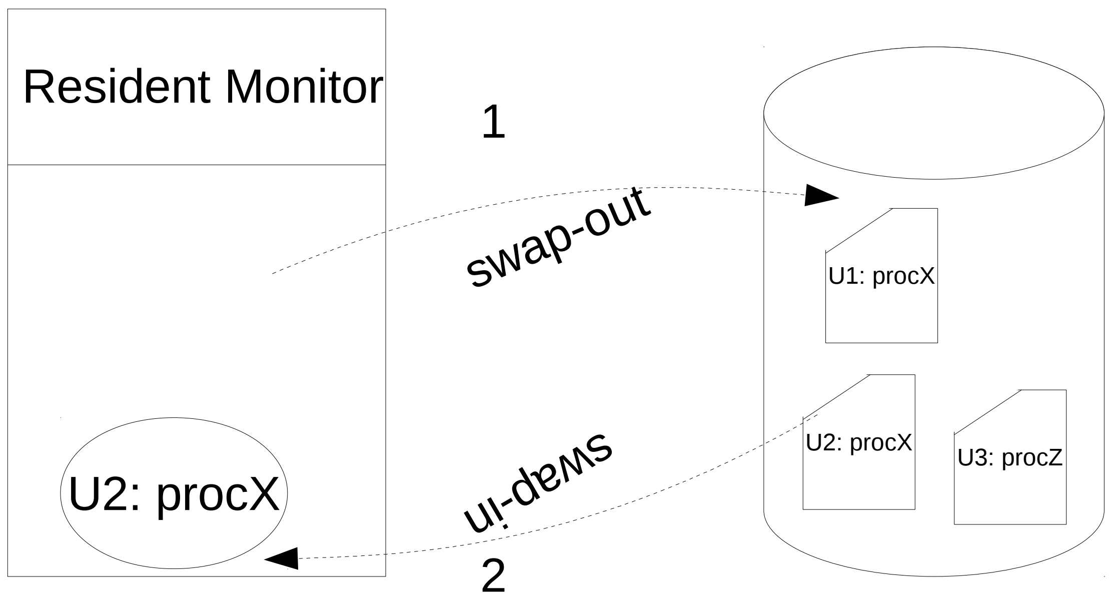


1. Single Process gets loaded into the MM from the SS/HD
2. When other process needs to run, first the current running process has to be moved-out onto the secondary storage

DRAWBACKS:

- The CPU would be idle during this IO operation, so inefficient
- Also, the degree of Multiprogramming is just 1.

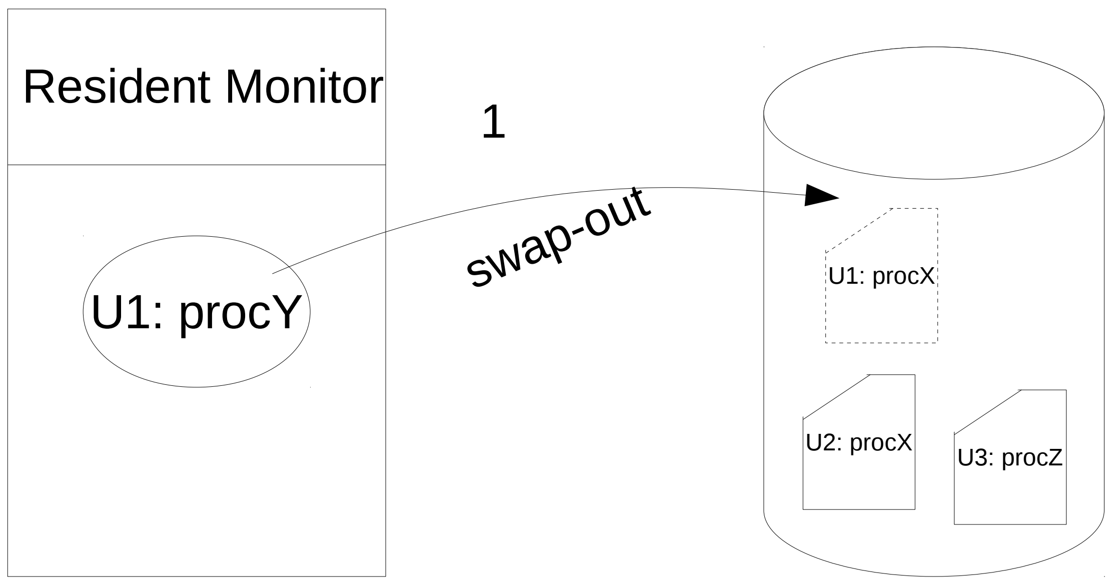
Multi-user System with OS



OBSERVE: It involves a kind of IO operation – as Secondary Storage is also a kind of IO device

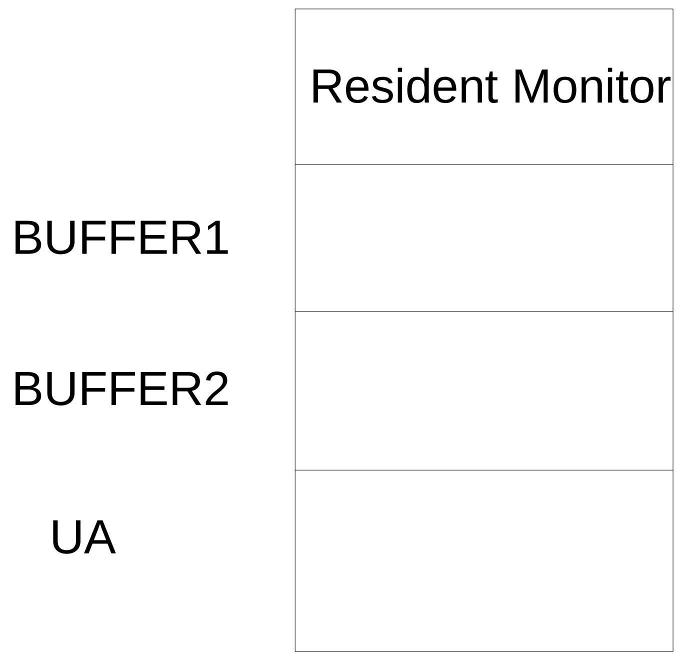
1. Sec. Storage is slower than MM access rate
2. The changes in var values etc. need to be changed in the SM as well

Multi-user System with OS



1. On demand, process is moved-out onto Sec. Storage
2. It involves IO operation
   - Sec. Storage is slower than MM access rate
3. The CPU would wait in this IO operation, so inefficient
4. Drawback back: Degree of Multiprogramming is just 1.

Partitioned Memory Model: Multiple Buffer Partitions + Active User Area



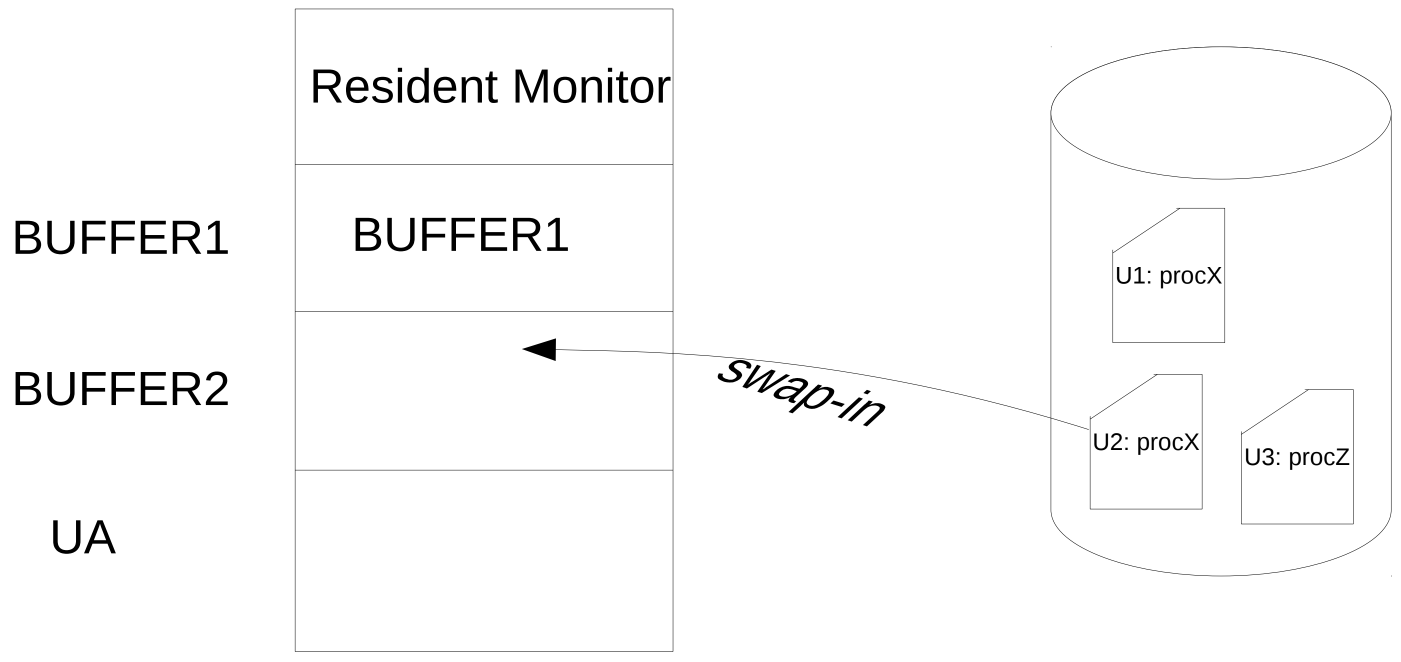
Paritions: Set of non-overlapping memory regions called partitions

I .Partition the UA of MM into
BUFFER1, BUFFER2, ..., User Area

II. This makes possible for:
1. Swap-in new process from SM into BUFFER2
2. **Move the new process from BUFFER2 to UA**
3. **Move Active process from UA to BUFFER1**
4. Swap-out process from BUFFER1 onto SM

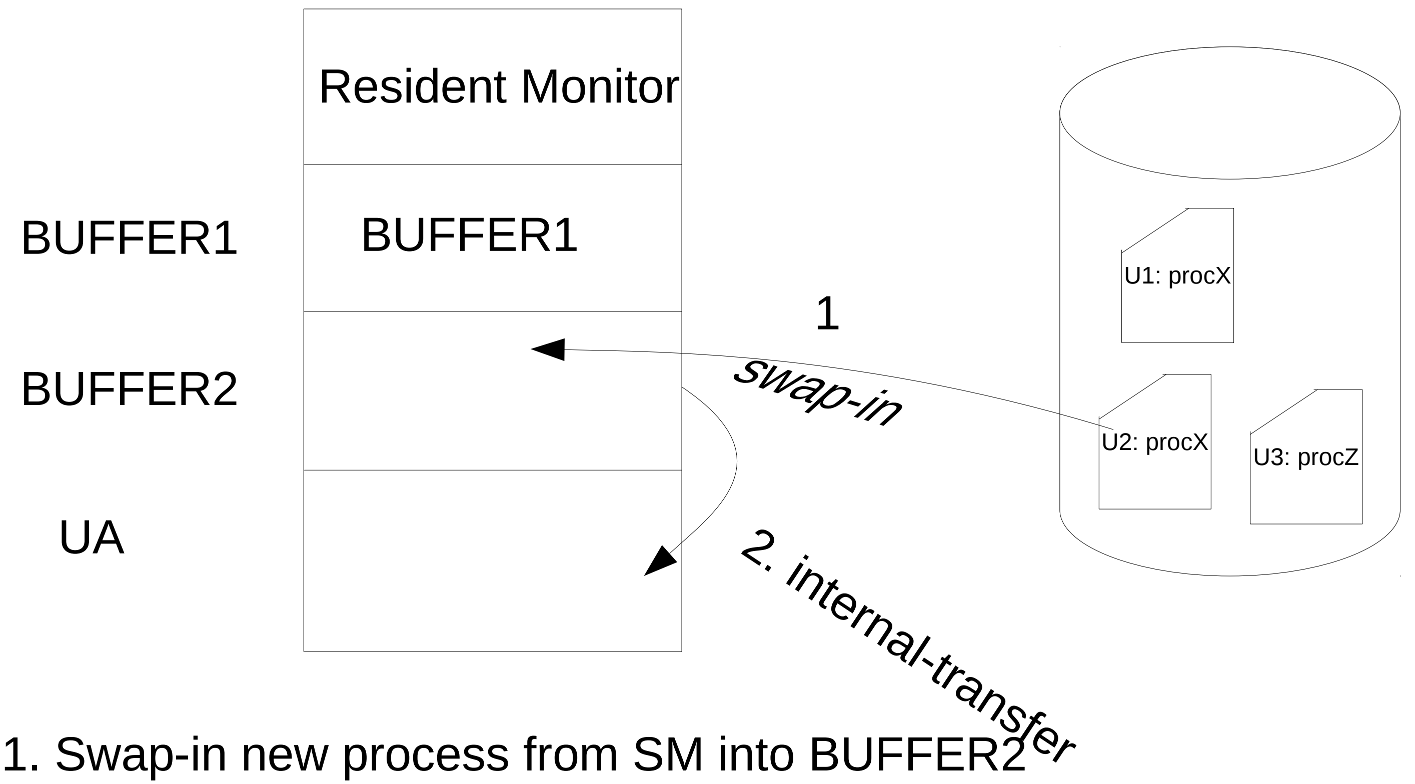
III. Multiple progs can be loaded into buffers simultaneously

Partitioned Memory Model: Multiple Buffer Partitions + Active User Area



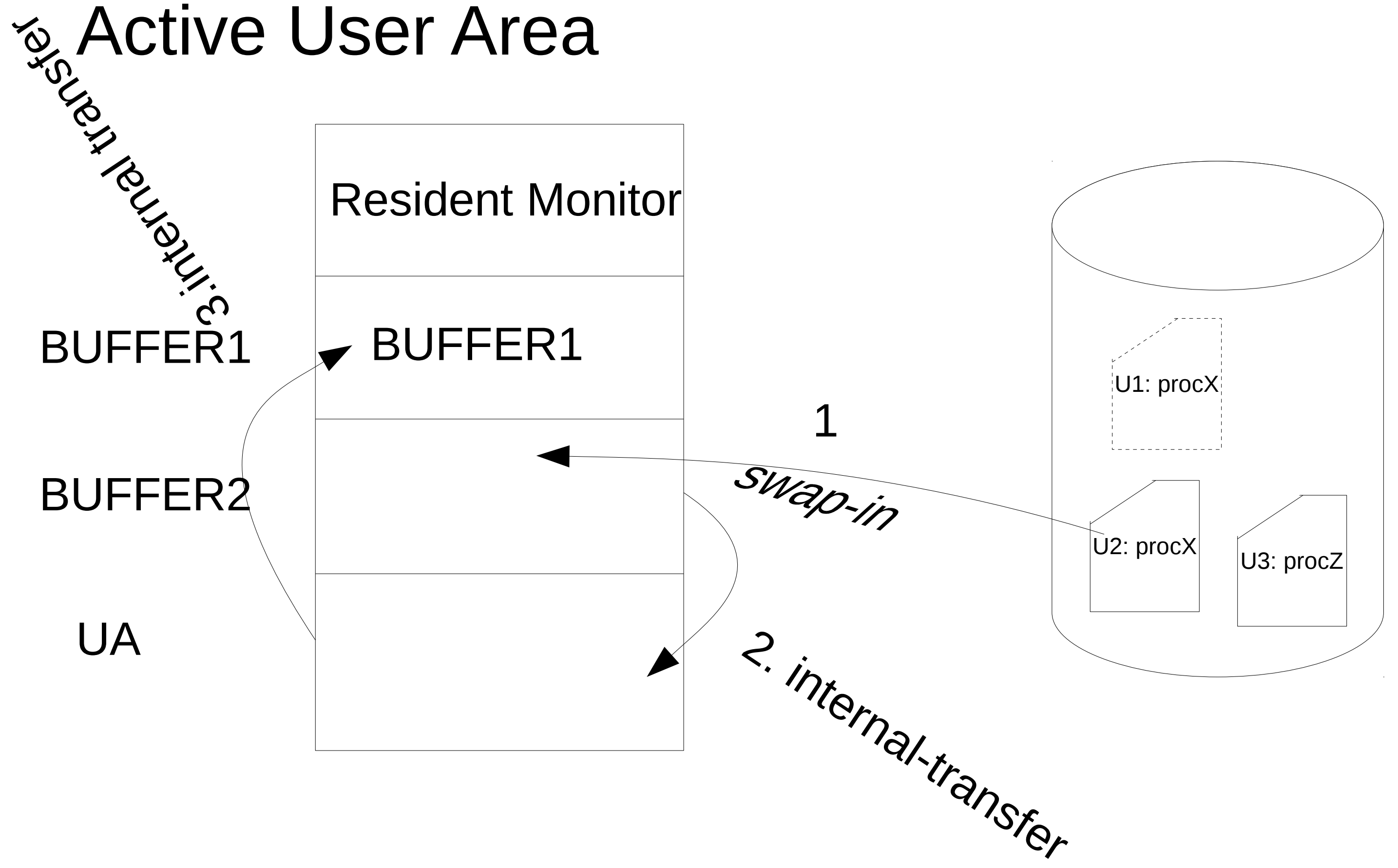
1. Swap-in new process from SM into BUFFER2

Partitioned Memory Model: Multiple Buffer Partitions + Active User Area



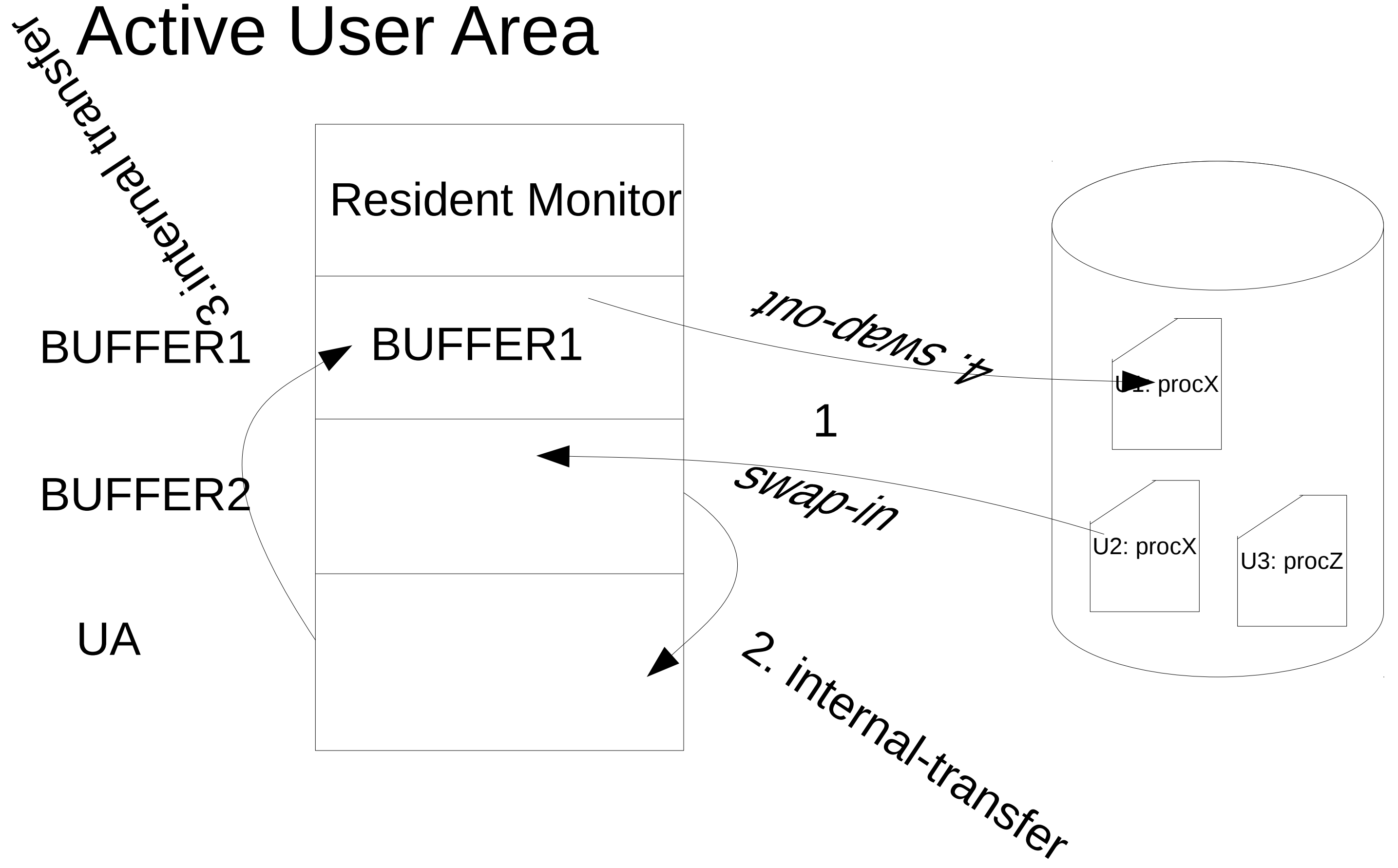
1. Swap-in new process from SM into BUFFER2
2. **Move the new process from BUFFER2 to UA**

Partitioned Memory Model: Multiple Buffer Partitions + Active User Area



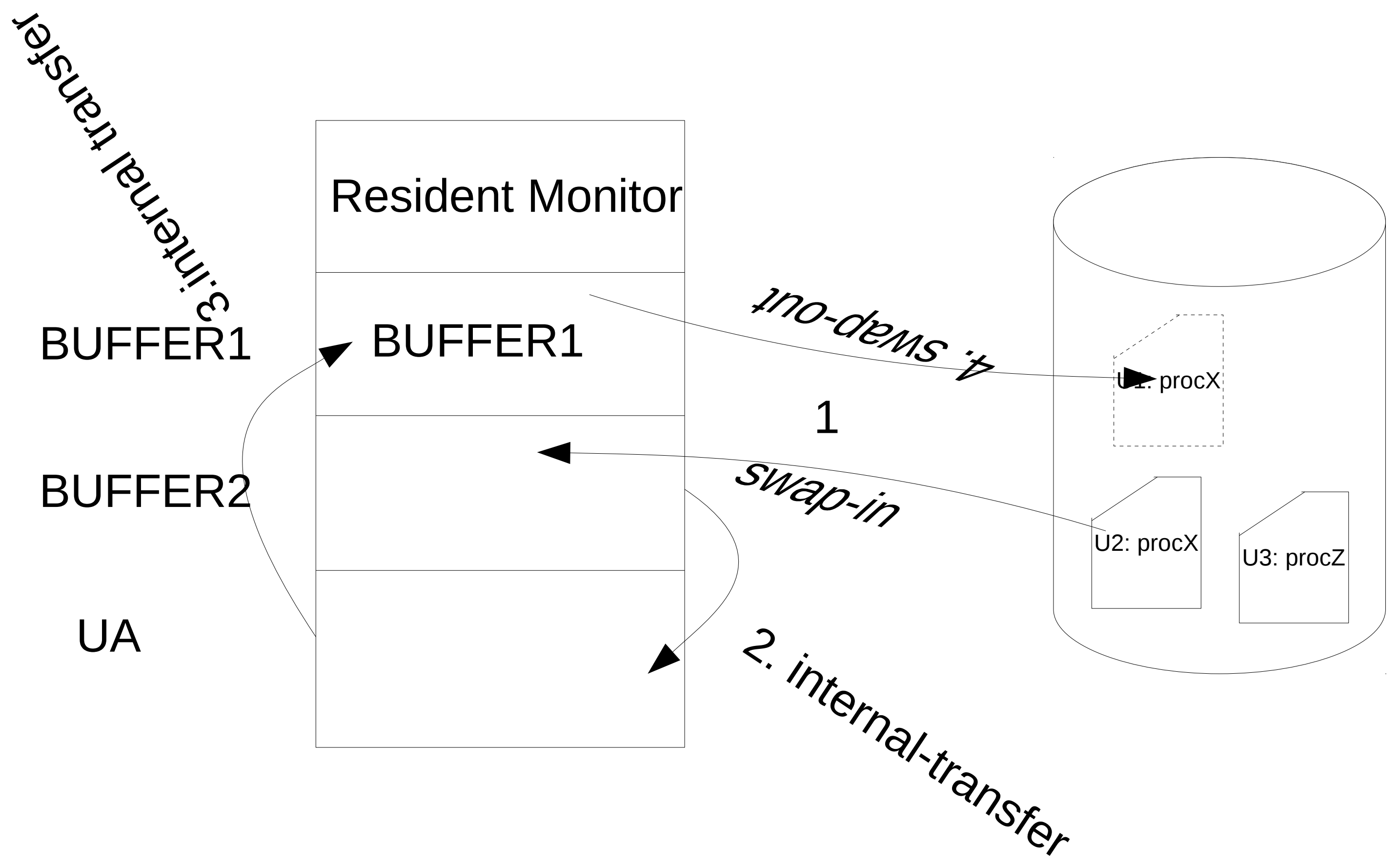
1. Swap-in new process from SM into BUFFER2
2. **Move the new process from BUFFER2 to UA**
3. **Move Active process from UA to BUFFER1**

Partitioned Memory Model: Multiple Buffer Partitions + Active User Area



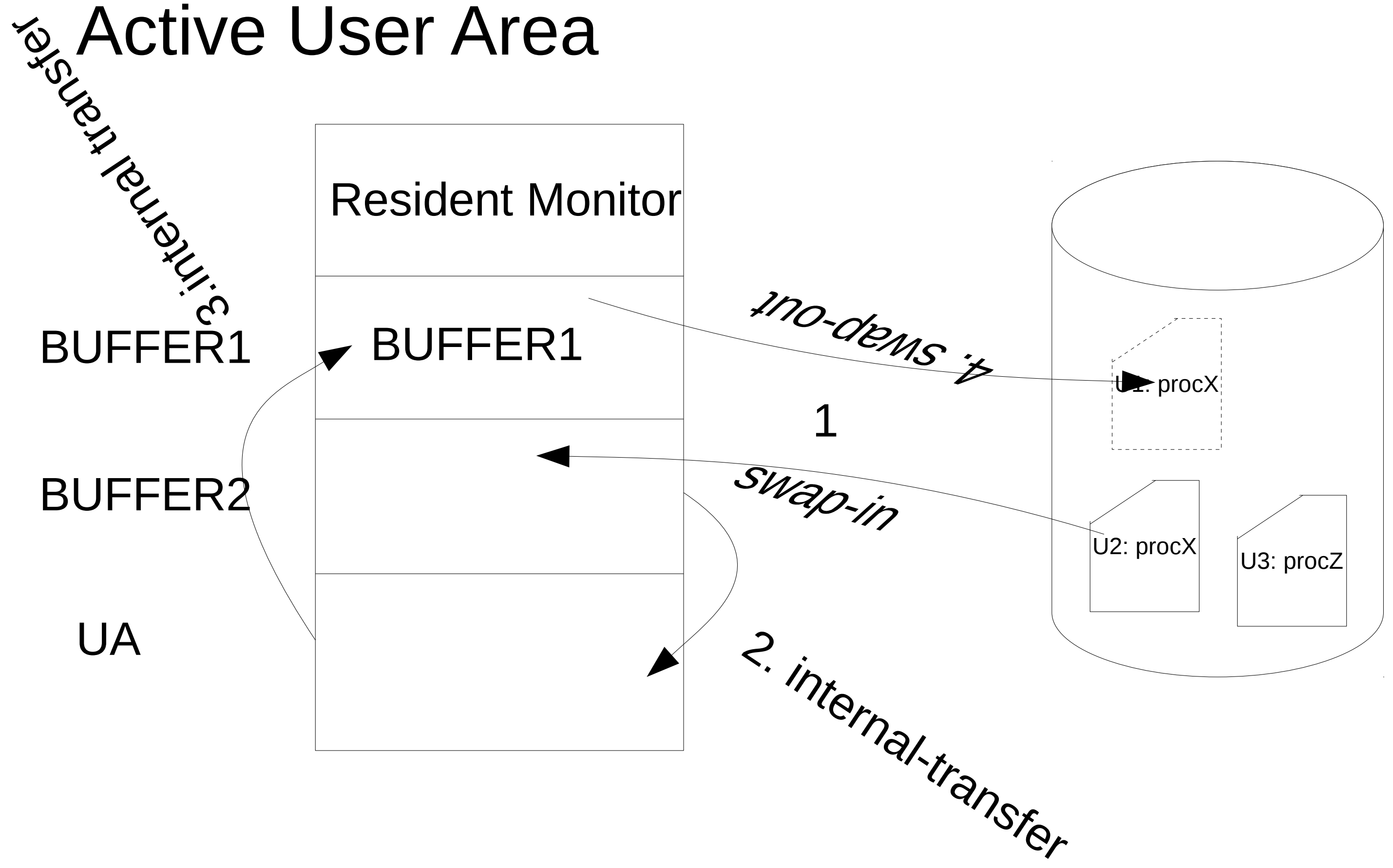
1. Swap-in new process from SM into BUFFER2
2. **Move the new process from BUFFER2 to UA**
3. **Move Active process from UA to BUFFER1**
4. Swap-out process from BUFFER1 onto SM

Multi-partitioned Memory Model



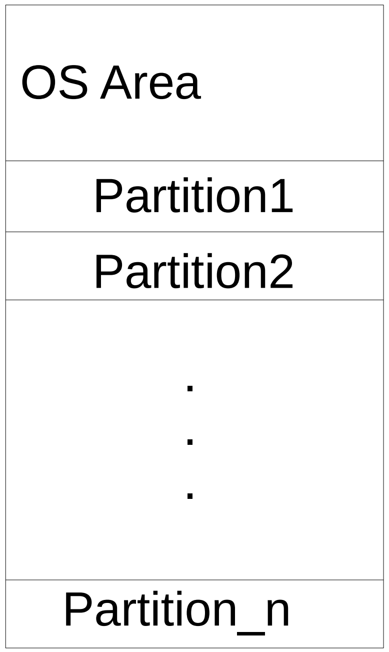
3. Multiple progs can be loaded into buffers simultaneously
  - degree of multiprogramming increases

Partitioned Memory Model: Multiple Buffer Partitions + Active User Area



Multiple progs can be loaded into buffers simultaneously
- degree of multiprogramming increases

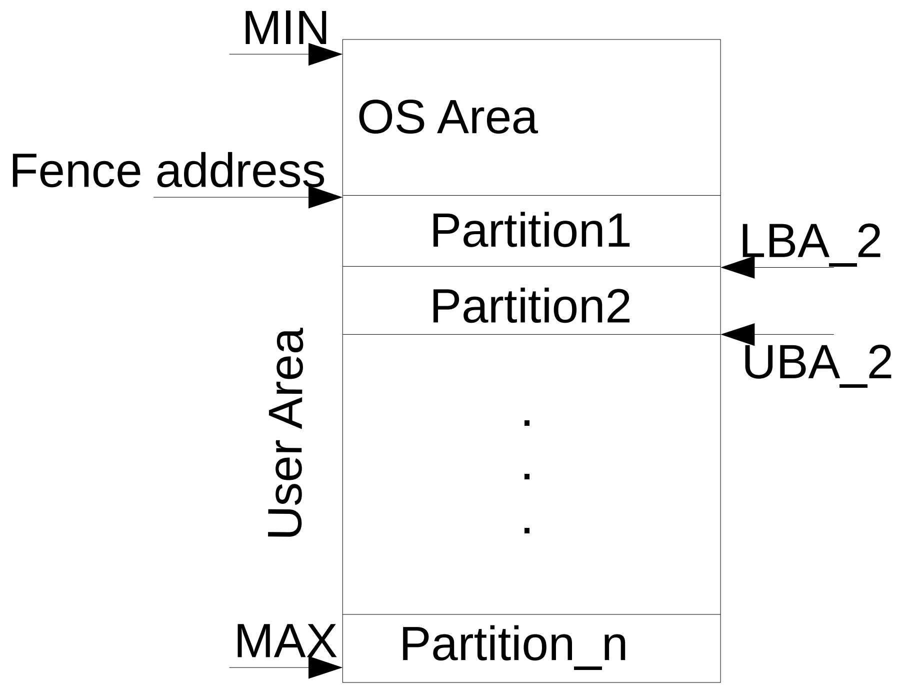
Multi-Partitioned Memory Model: Fixed number of fixed sized paritions



1 .Every partition is characterised by:
Base Address
Size/Limit

2. Process when loaded gets mapped to one partition

Multi-Partitioned Memory Model: Fixed number of paritions



1. In such multi-partitioned model, memory protection is more complex
    - need to make sure one process in a partition can not access address belonging to another partition

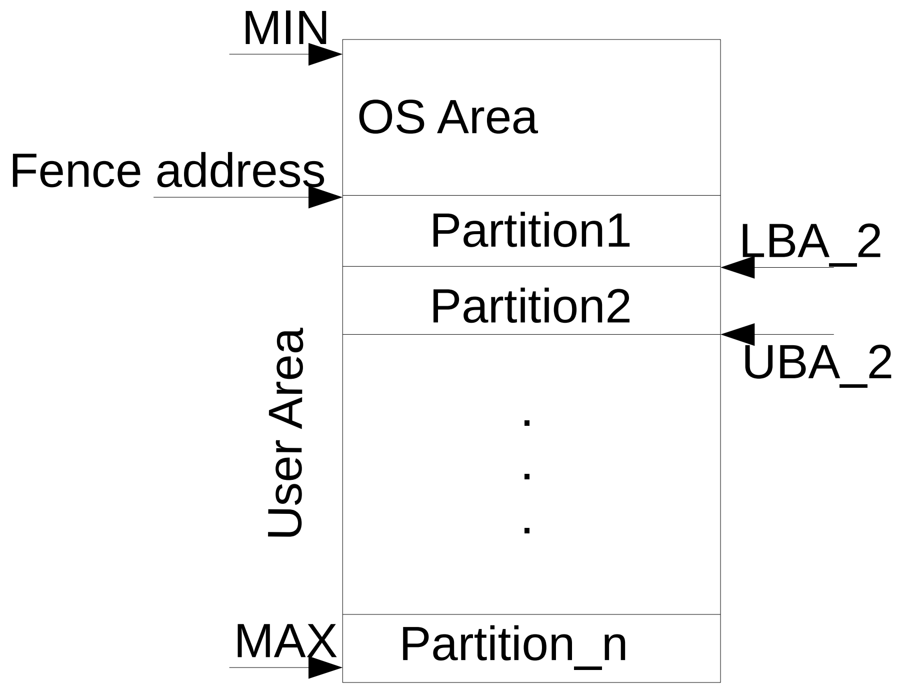
2. Each partition is identified by a contiguous memory addresses between two address bounds;
    -Lower Bound Address/Register (LBA)
    -Upper Bound Address/Register (UBA)

If size of partition is S_p, then
UBR_p = LBR_p + S_p

3. So every address from a process (belonging to a partition) shall respect these bounds for that partition.

4. LBA is stored in Base Register; while UBA is stored in Limit Register

Multi-Partitioned Memory Model: Fixed number of paritions



1. In such multi-partitioned model, memory protection is more complex
   - need to make sure one process in a partition can not access address belonging to another partition

2. Each partition is identified by a contiguous memory addresses between two address bounds;
   - -Lower Bound Address (LBA)
   - -Upper Bound Address (UBA)

If size of partition is S_p, then
UBA_p = LBA_p + S_p

3. So every address from a process (belonging to a partition) shall respect these bounds for that partition.

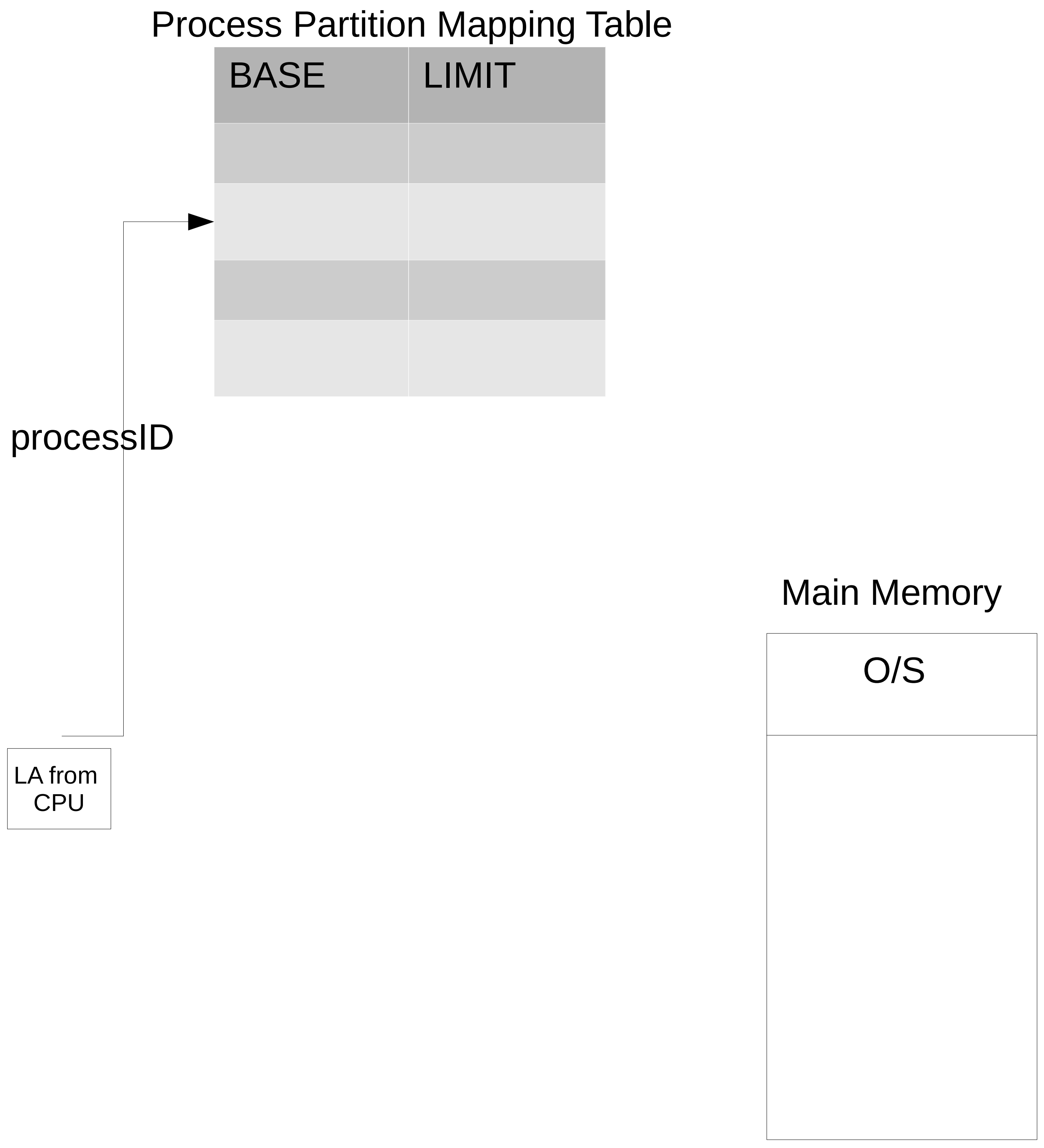
4. Every address shall be greater than o

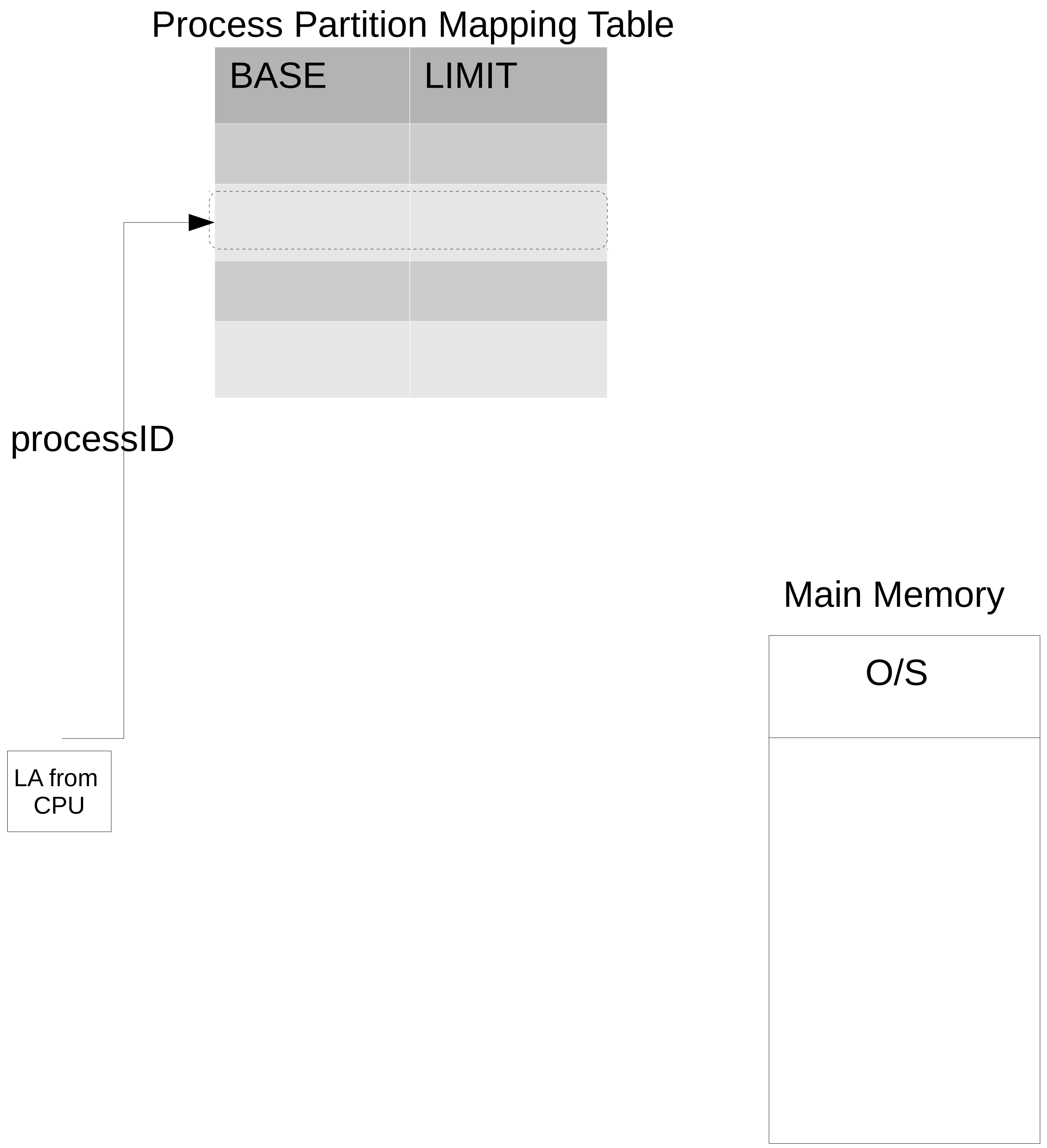
Process Partition Mapping Table

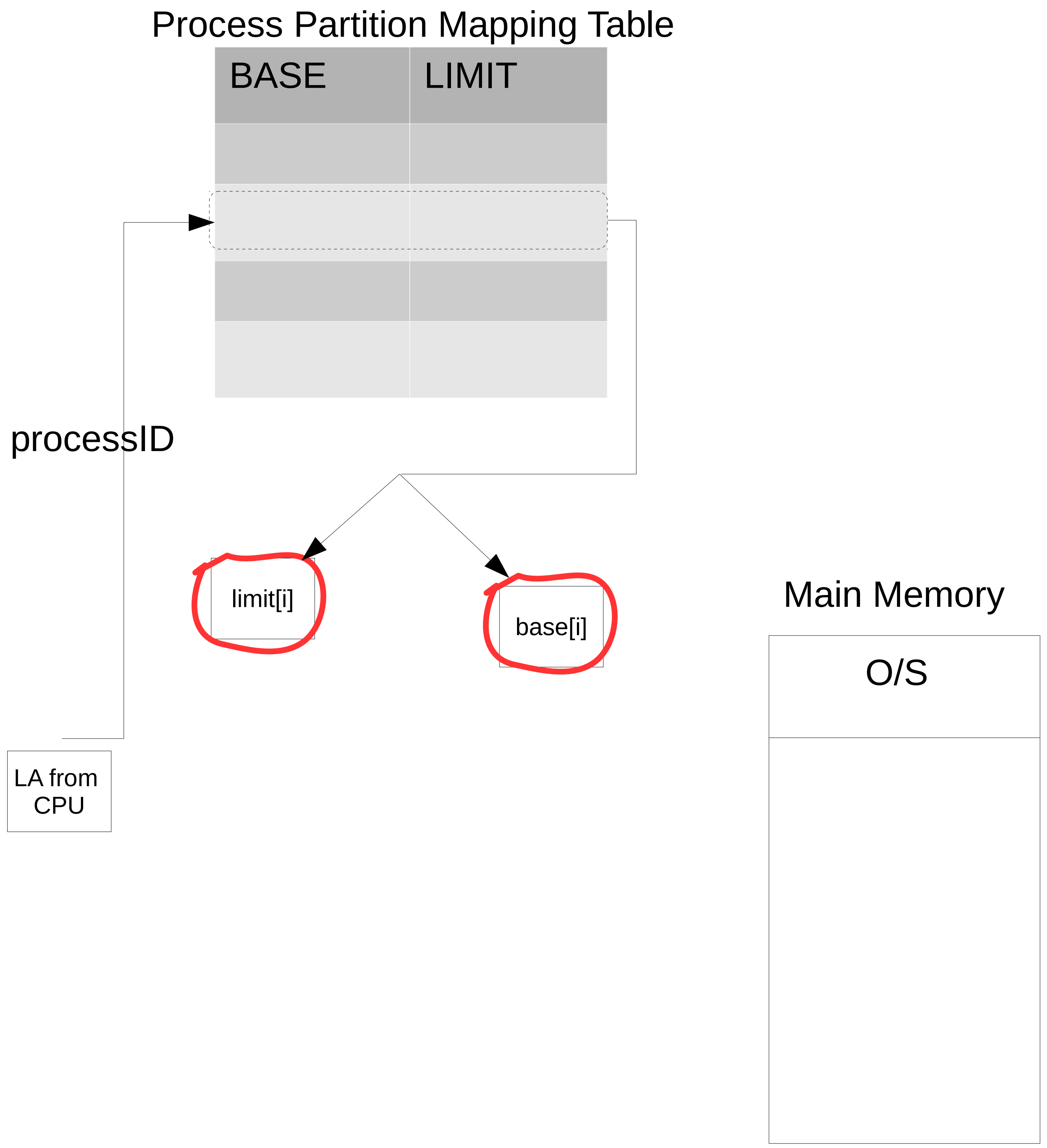
BASE	LIMIT

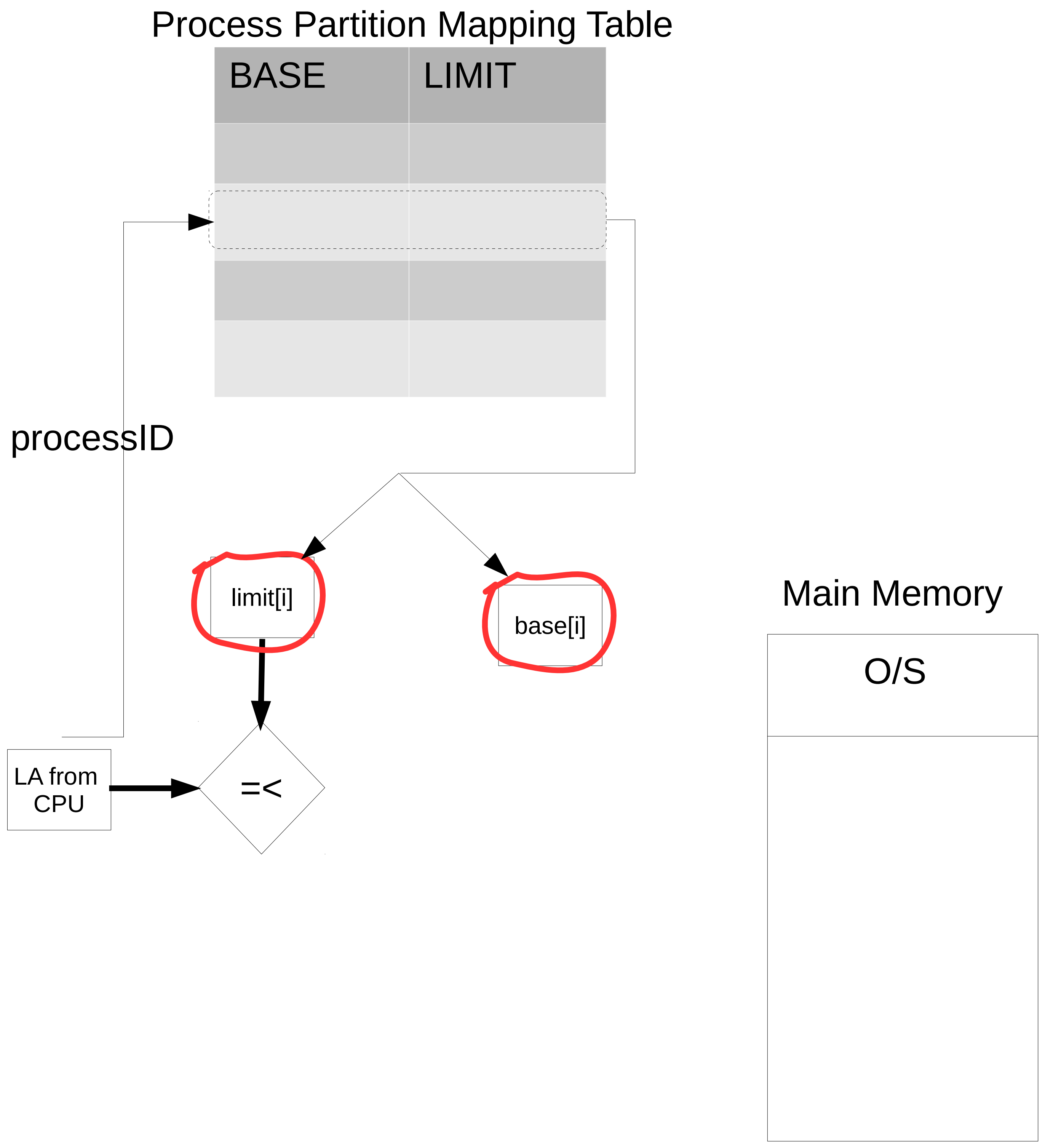
Main Memory

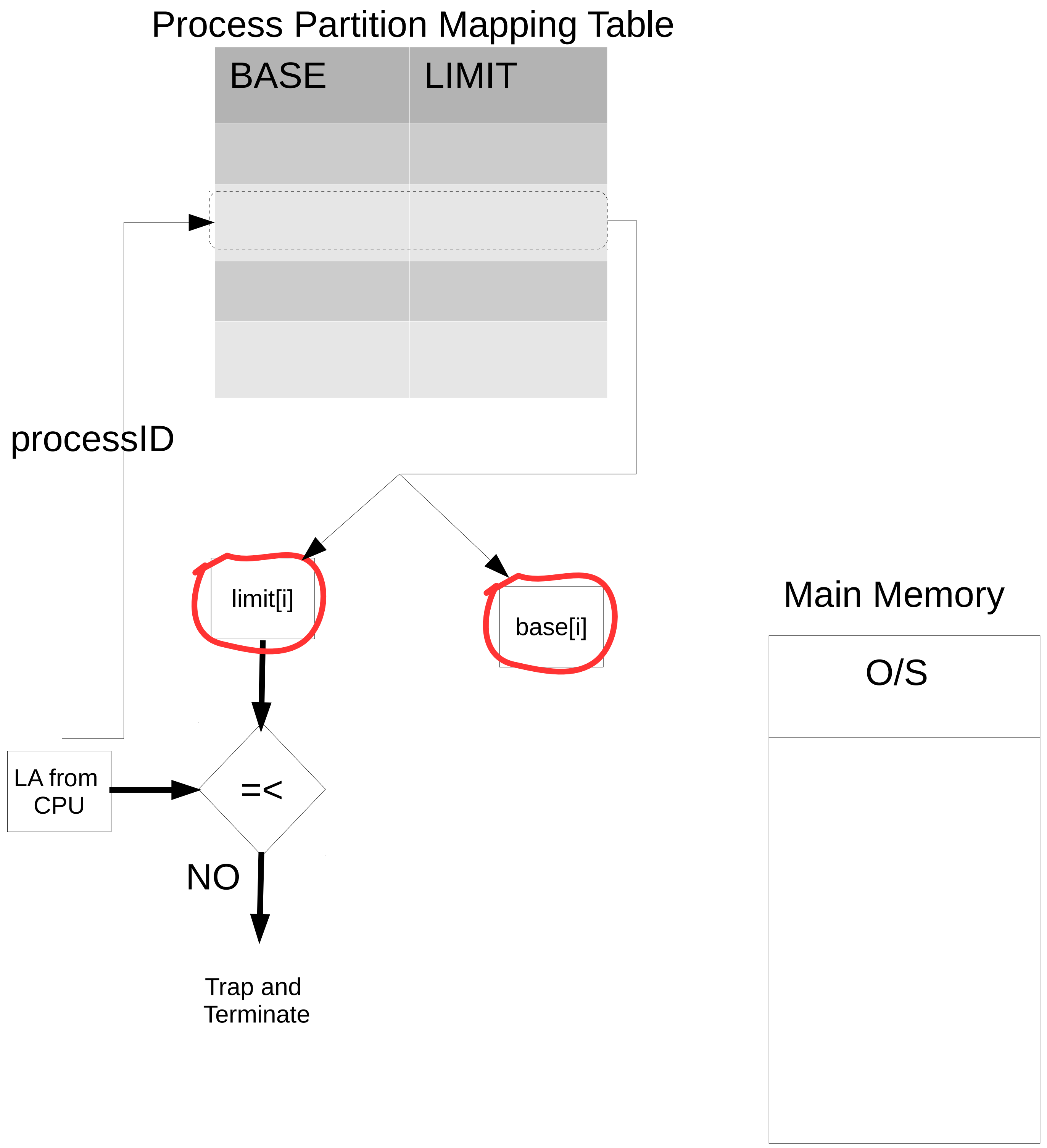
O/S
LA from CPU

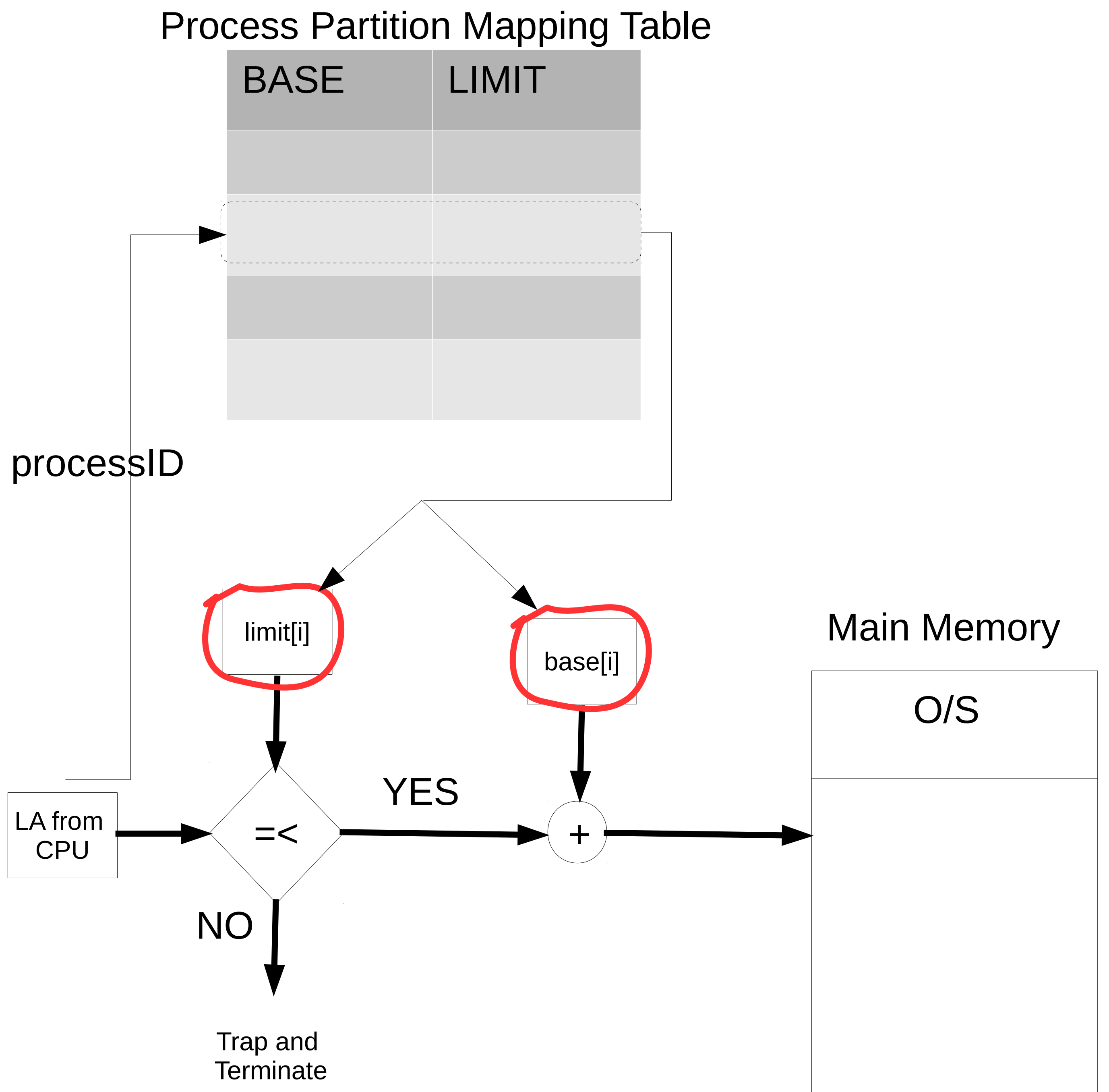

Process Partition Mapping Table
BASE
LIMIT
processID
LA from CPU
Main Memory
O/S

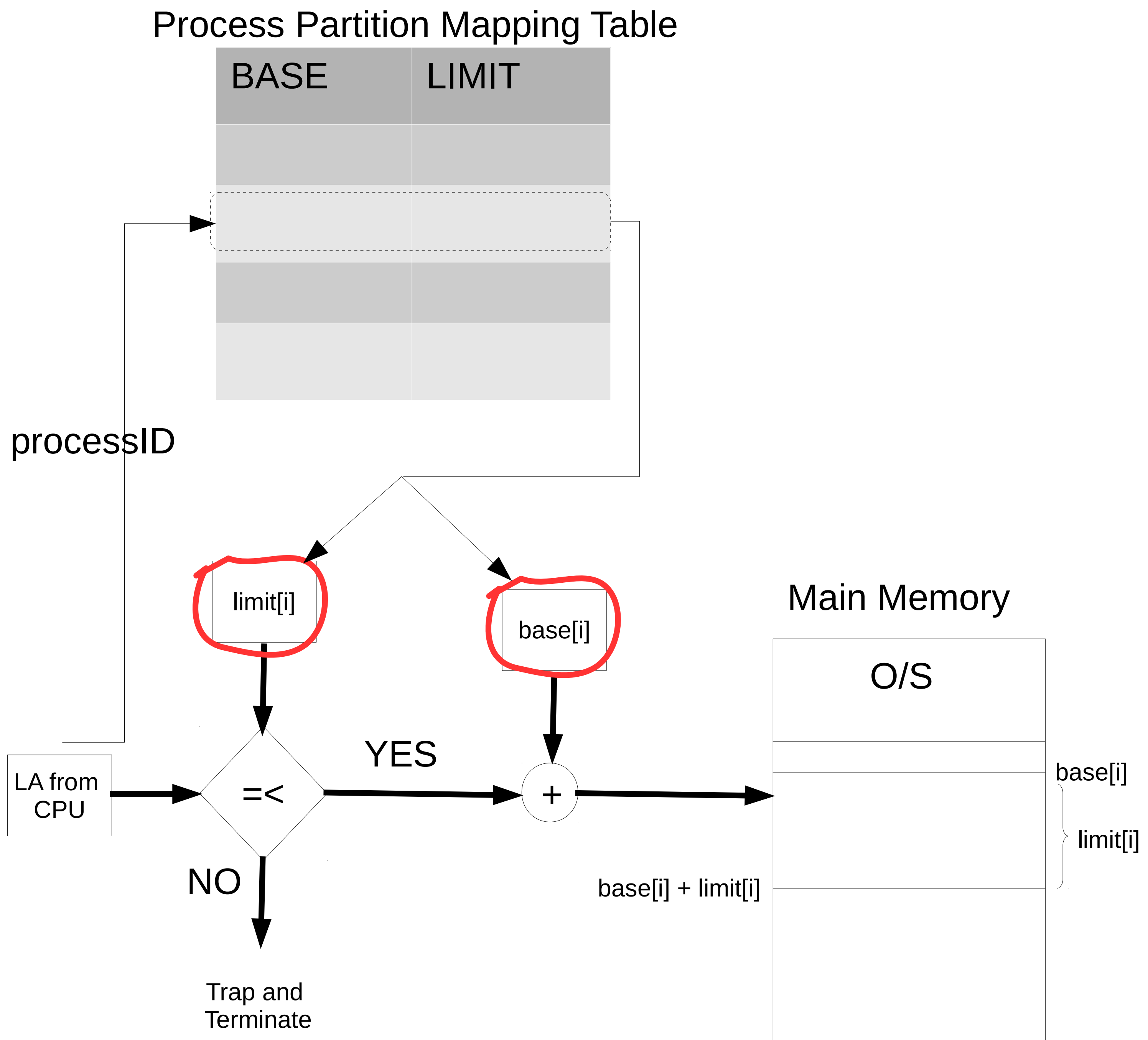

Process Partition Mapping Table
BASE
LIMIT
processID
LA from CPU
Main Memory
O/S


Process Partition Mapping Table
BASE
LIMIT
processID
limit[i]
base[i]
Main Memory
O/S
LA from CPU


Process Partition Mapping Table
BASE
LIMIT
processID
limit[i]
base[i]
LA from CPU
=<
Main Memory
O/S


Process Partition Mapping Table
BASE
LIMIT
processID
limit[i]
base[i]
LA from CPU
=<
NO
Trap and Terminate
Main Memory
O/S


Process Partition Mapping Table
BASE
LIMIT
processID
limit[i]
base[i]
Main Memory
O/S
LA from CPU
=<
YES
+
NO
Trap and Terminate


Process Partition Mapping Table
BASE
LIMIT
processID
limit[i]
base[i]
Main Memory
O/S
LA from CPU
=<
YES
+
NO
Trap and Terminate
base[i]
limit[i]
base[i] + limit[i]

Fragmentation in MM

- Memory fragmentation: breaking of memory into non-contiguous blocks that often lead to non-utilisation or wastage of memory
- In this Multi-partition with fixed number processes or fixed-sized paritions: (also called as <u>M</u>ultiprogramming with <u>F</u>ixed number of <u>T</u>asks (MFT))
  - the size of each partition is fixed
  - Partitions could be of different sizes
  - With MFT we have fragmentation
    - Fragmentation: The inability in using memory even though it is empty!
  - Partition gets allocated to a process
    - Several algorithms exist for process-to-partition mapping
    - Different algorithms have different trade-offs particularly w.r.t fragementation perspective
    - Ofcourse, we need to have an associated data/record
      - Start address
      - Size
      - Allocation status

Partition allocation algorithms

- Two kinds:
  - First-fit algorithm
  - Best-fit algorithm
- First-fit algorithm:

  - Finds the first available partition with the size greater than or equal to the process memory needs

- Best-fit algorithm:

  - Finds the nearest sized partition meeting the process memory size of a process.

Fragmentation types

- Two type
  - Internal memory fragmentation
    - ILLUSTRATE
  - External memory fragmentation
    - ILLUSTRATE
- Best-fit minimises Int. mem. Frag
  - But complexity is high in comparison to FF

Fragmentation in MM

- Memory fragmentation: breaking of memory into non-contiguous blocks that often lead to non-utilisation or wastage of memory
- In this Multi-partition with fixed number of processes:
  - the size of each partition is fixed
  - Partitions could be of different sizes
  - With MFT we have fragmentation
  - Partition gets allocated to a process
    - This record is maintained in Partition Allocation Table