Understanding Memory Management

L J Gokul Vasan

Contents

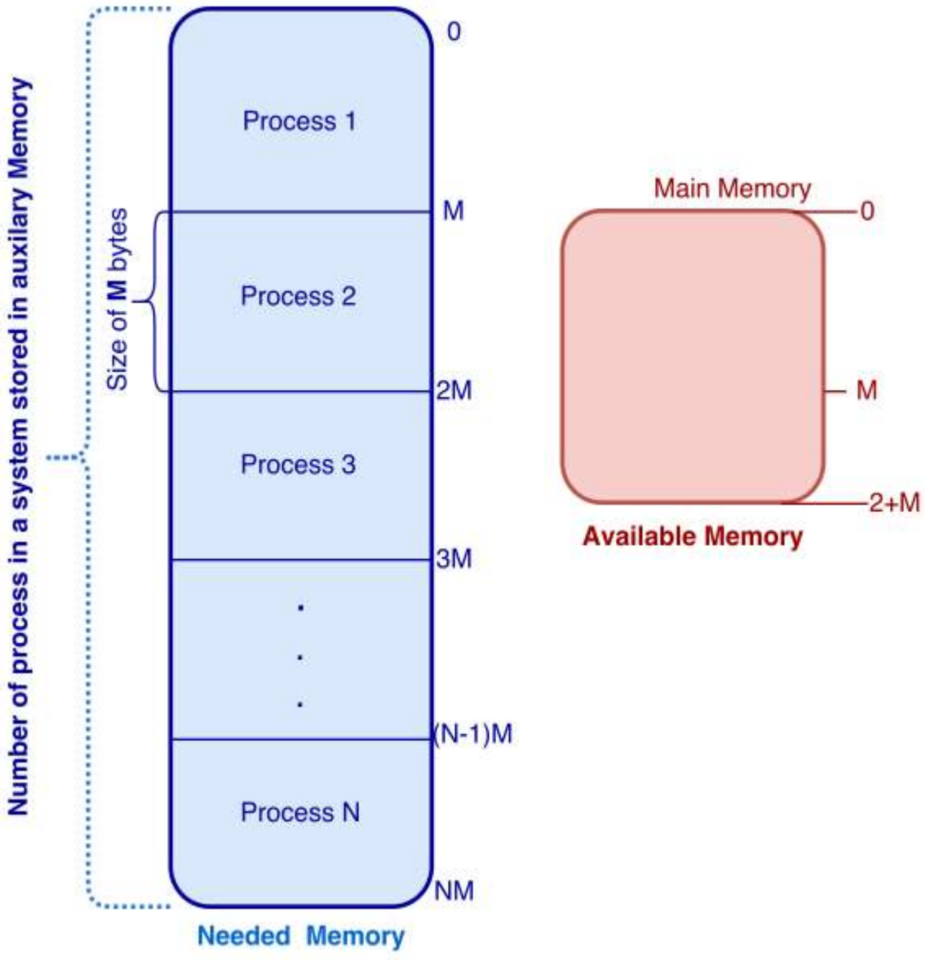
- Basics
  - Swapping
  - Paging
  - Virtual Memory

- Hardware in memory management
  - TLB
  - Translation of address

- Theory on memory management
  - Locality of reference
  - Thrashing Problem
  - Working set

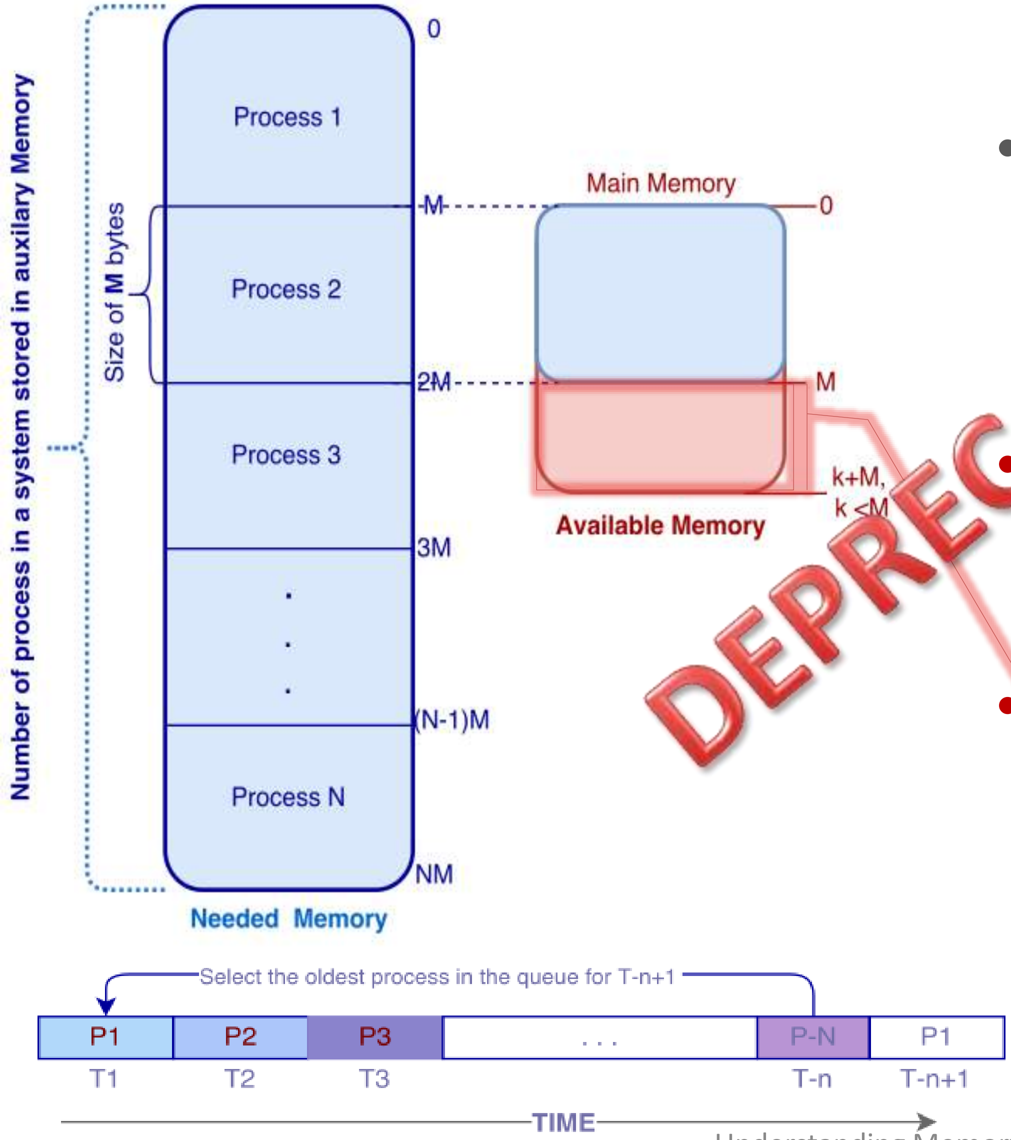
- Software in Memory management
  - Replacement policy
  - Placement policy
  - Scan rate policy
  - Fetch Policy

Process and Memory



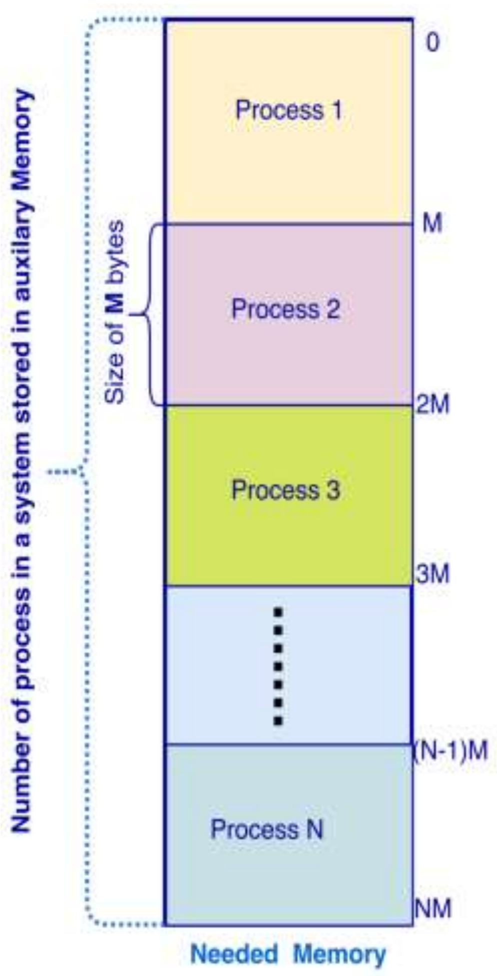
- In reality, main memory is smaller than required memory.

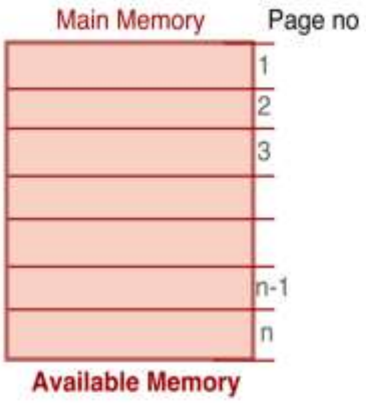
Swapping



- A process in main memory is swapped on time sharing or priority basis.
- Swapping: The whole process is brought in and out of main memory.
- Predicament:
  - External Fragmentation.
  - larger process can exist than main memory.
  - Time Consuming

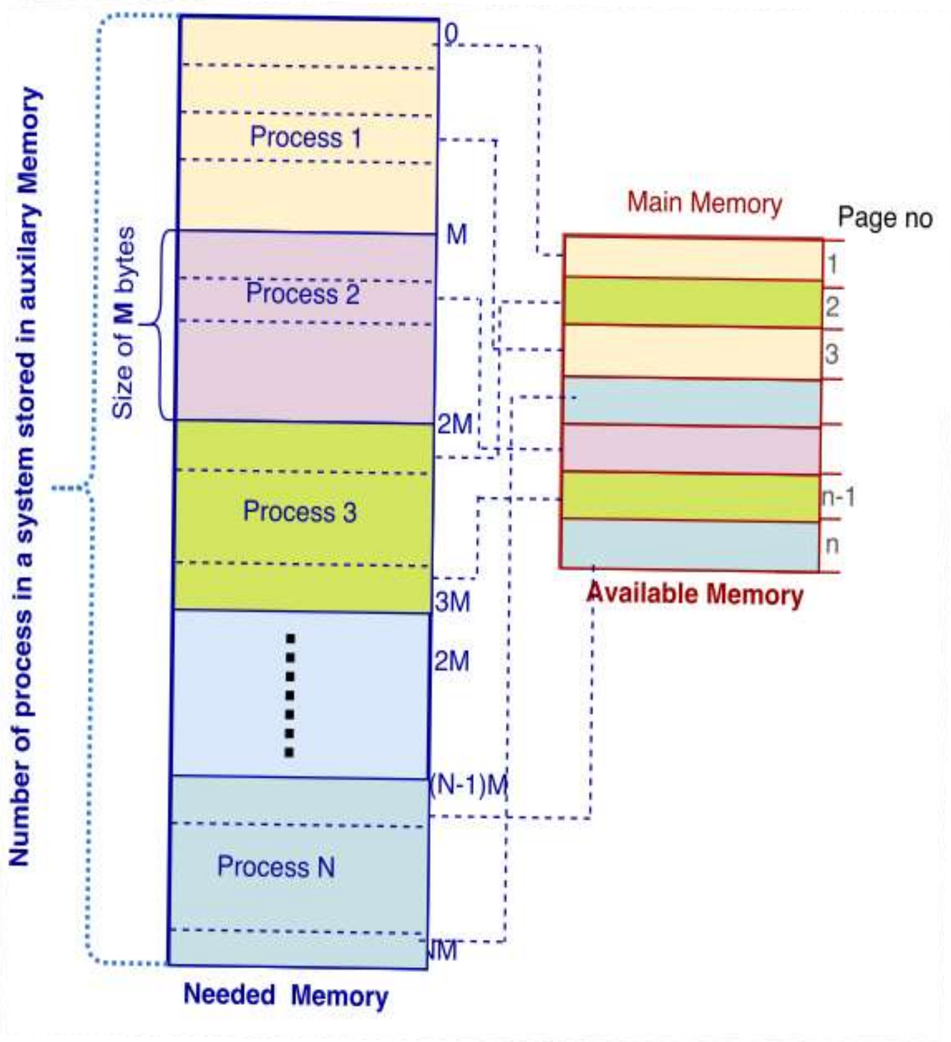
Paging





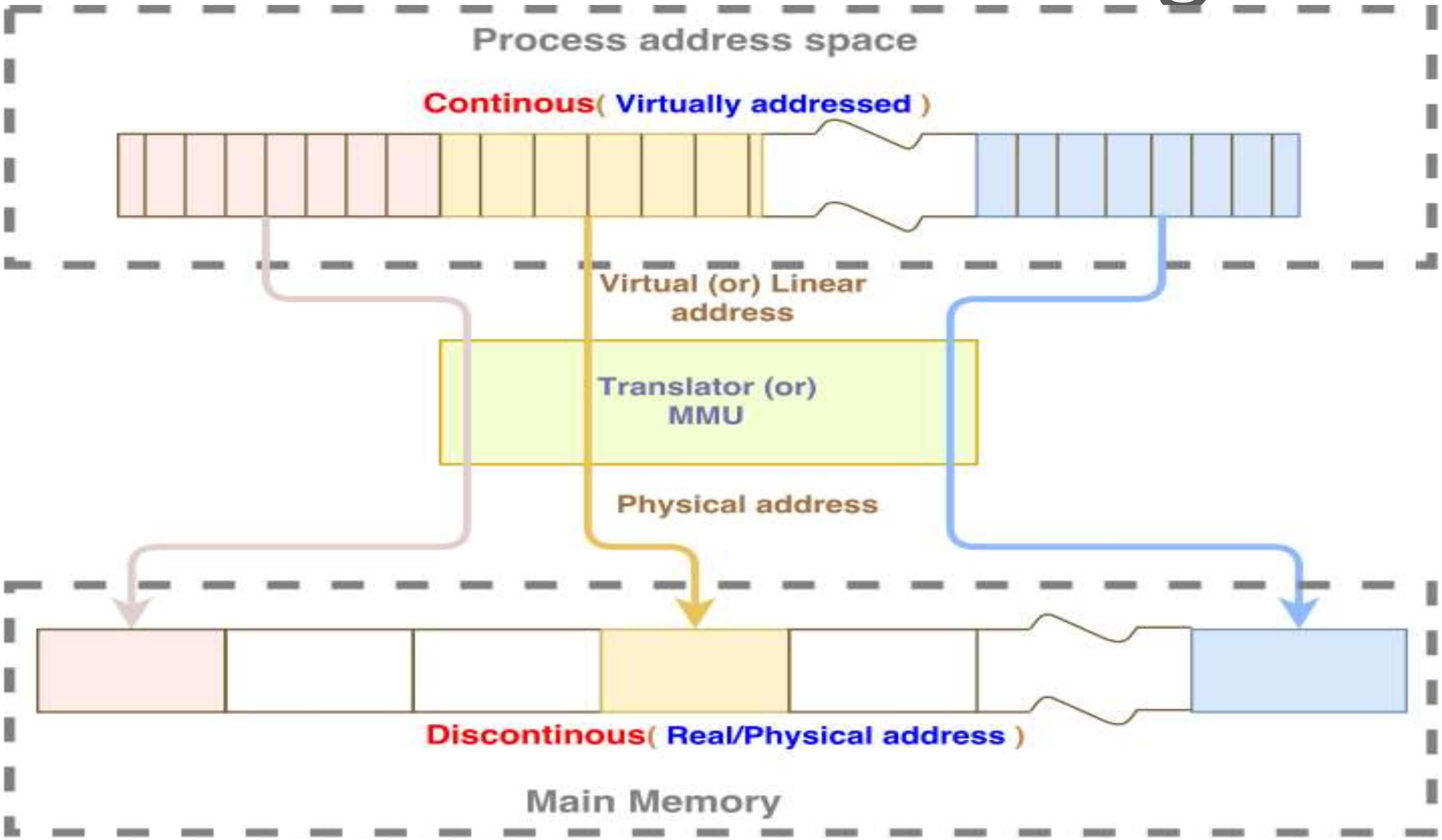
- Main memory is divided into equal chunks called pages.
- Only a piece of the process is loaded into the main memory.
- Paging: On Exhaustion of main memory only a page or a set of pages are evicted from main memory.

Paging



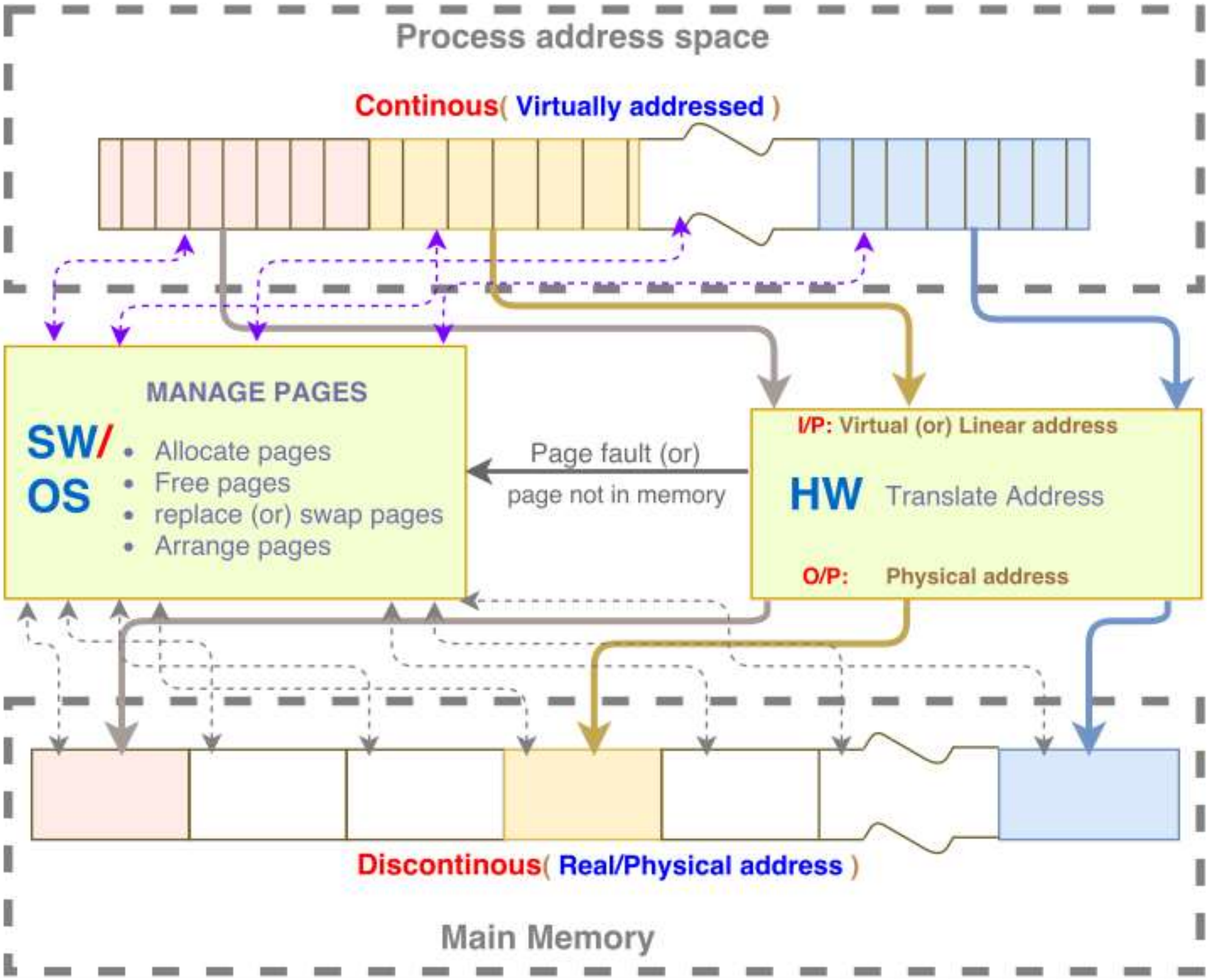
- Advantages:
  - Selective loading of a process.
  - process could load on discontinuous pages, avoiding external fragmentation.
  - Process can be larger than main memory.
- Problem:
  - A Process cannot be addressed continuously.
- Solution:
  - Virtual addressing.

Virtual Addressing


Process address space
Continous(Virtually addressed)
Virtual (or) Linear address
Translator (or) MMU
Physical address
Discontinous(Real/Physical address)
Main Memory

- Now, there are 2 addresses:
  - Virtual (or) Linear address (simulated to provide continuous perspective).
  - Physical address (represents real memory).
- Process is virtually addressed.
- Translator named Memory management unit(MMU) converts Virtual to Physical.

Role of OS and HW in Memory Management



Process address space

Continous(Virtually addressed)

MANAGE PAGES

SW/ OS

- Allocate pages
- Free pages
- replace (or) swap pages
- Arrange pages

Page fault (or) page not in memory

I/P: Virtual (or) Linear address

HW Translate Address

O/P: Physical address

Discontinous(Real/Physical address)

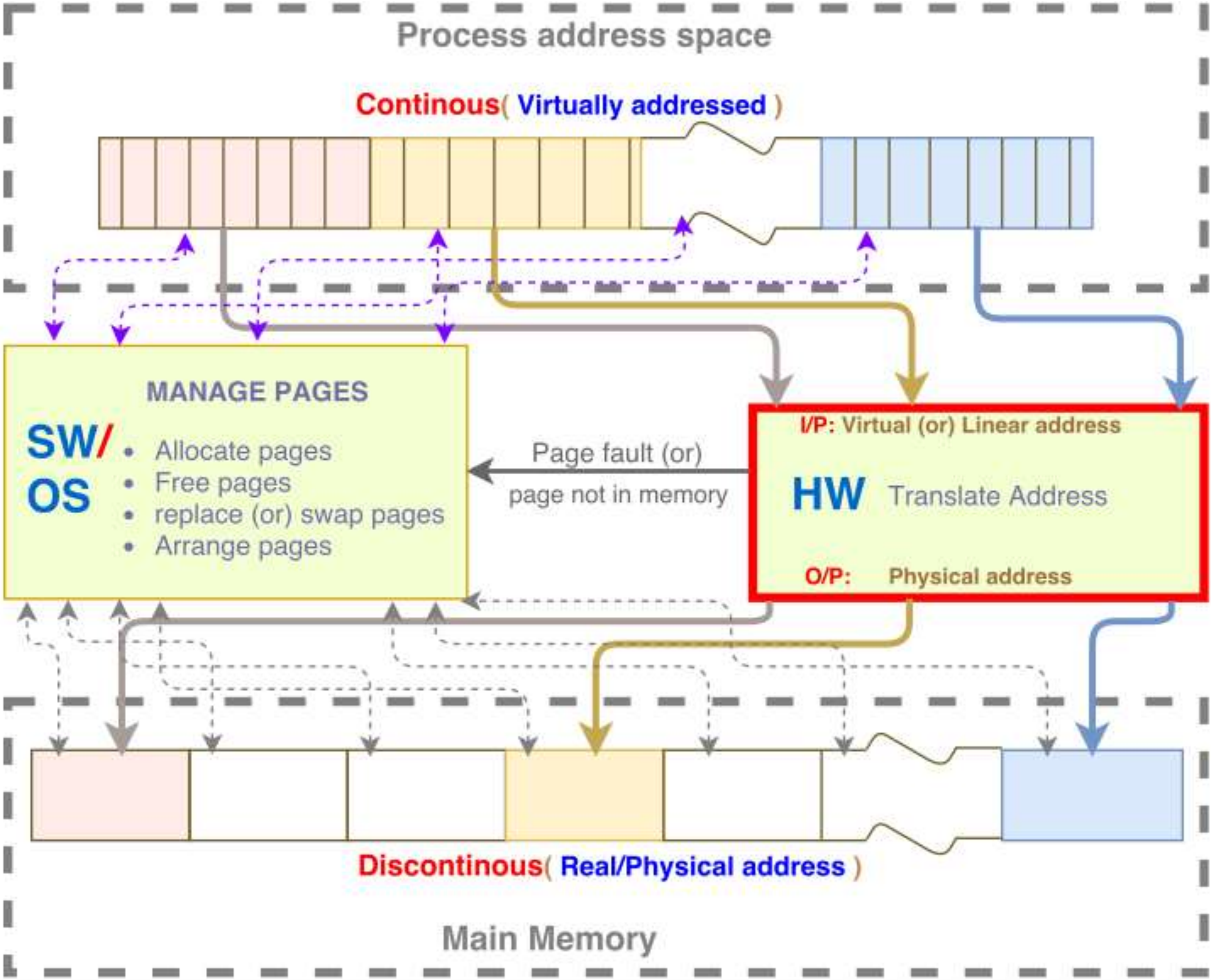
Main Memory

Hardware in Memory management

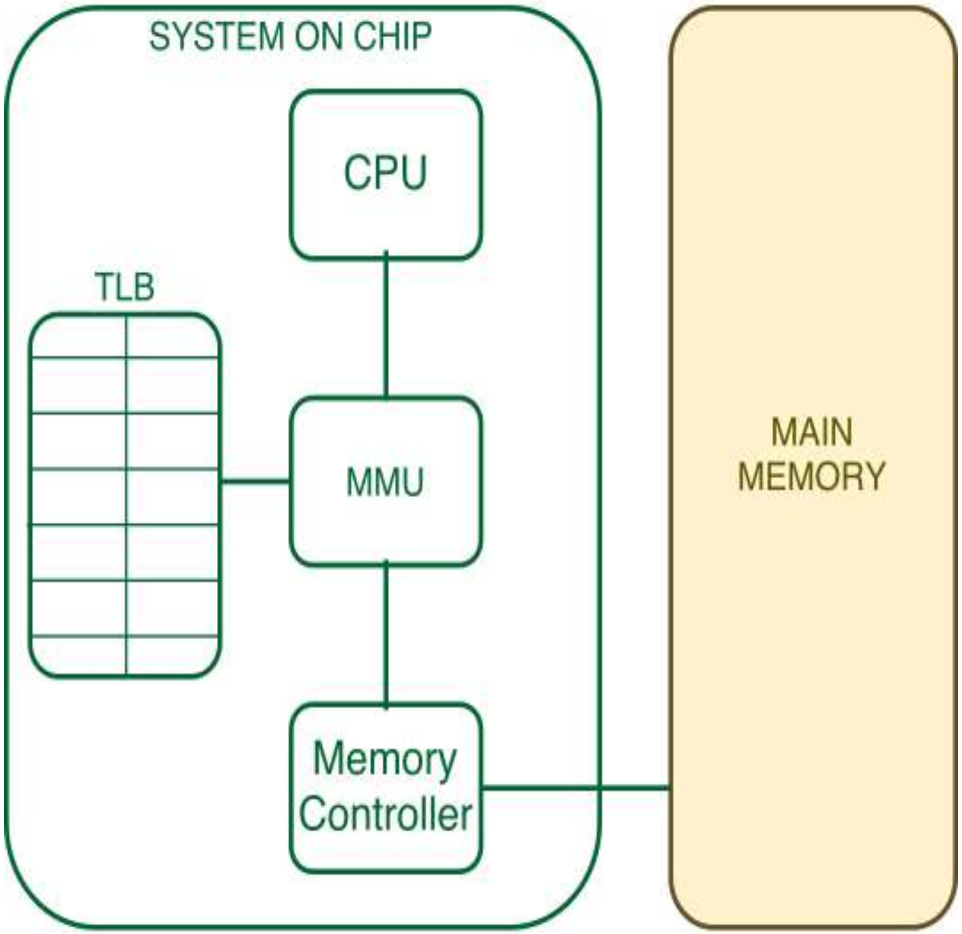
Address Translation

Virtual -> Physical

Hardware in Memory management

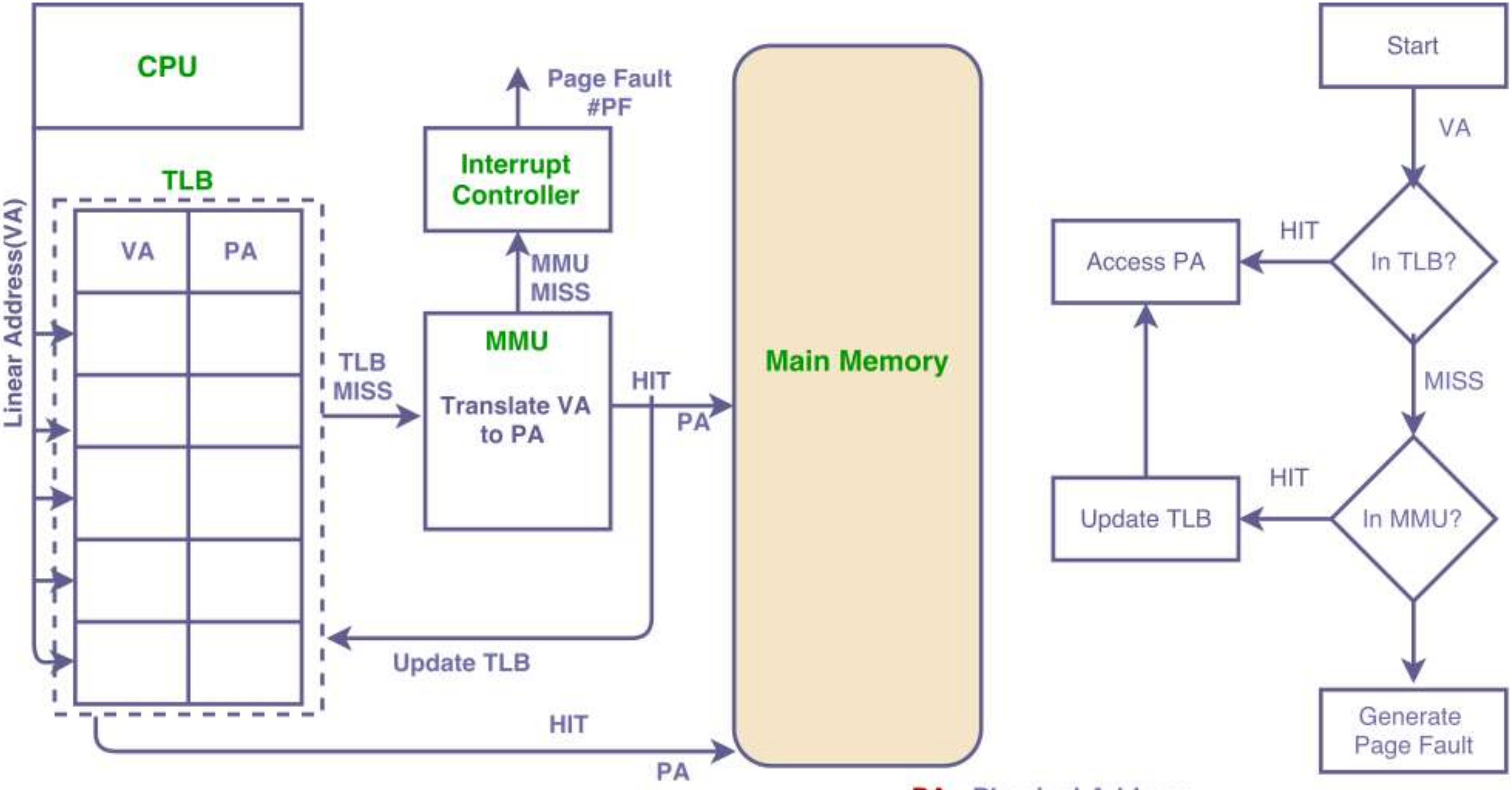


MMU position in HW



- Placed between CPU and memory controller.
- TLB(Translation Lookaside Buffer) would hold the last recent translated page addresses.
  - Reduces computation time of MMU translation.
- If not present in TLB, MMU translates and also updates TLB.

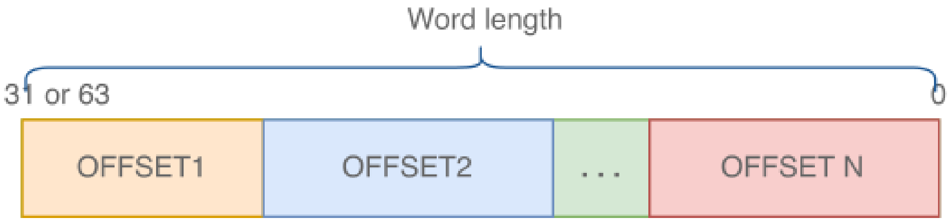
MMU and TLB data flow(Simplified)



PA : Physical Address
VA: Virtual Address

Virtual Address

- Virtual address is a sequence of offsets.



- Offsets: Complete address is divided into smaller contiguous bit chunks.

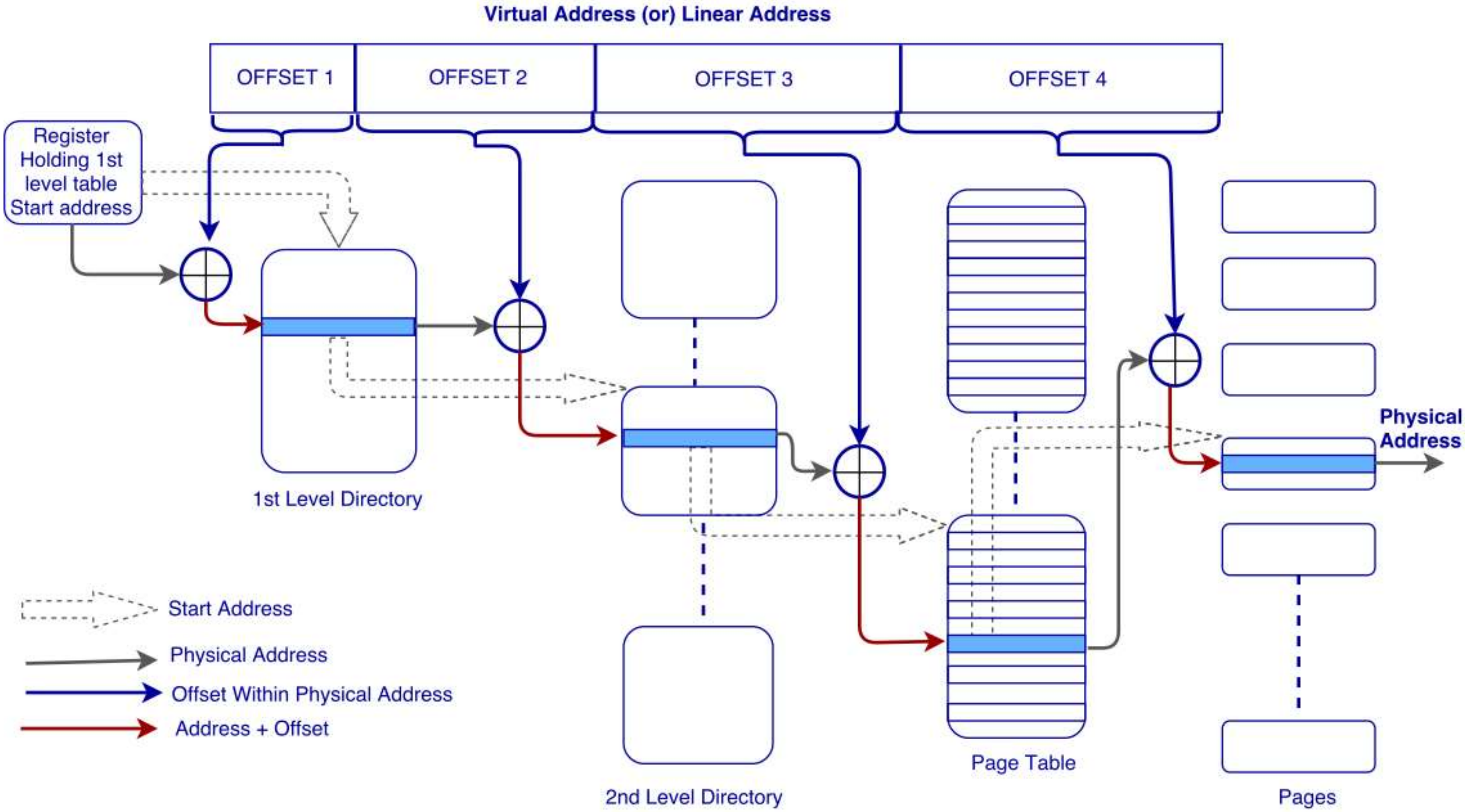
- Each bit chunk (Offset) holds a offset or index of a table.
  - Table will hold the starting address of the next table.
  - This address is added with the next bit chunk to derive the right location in next table or page.

- Tables are pages of memory, however, rather holding the data it is a list holding address of next table or page frame.

- On schedule of a processes, the OS loads the process's Directory start address (offset 1) into the corresponding memory management register.

MMU: Translation Procedure(Generalised)

- On TLB miss MMU does translation

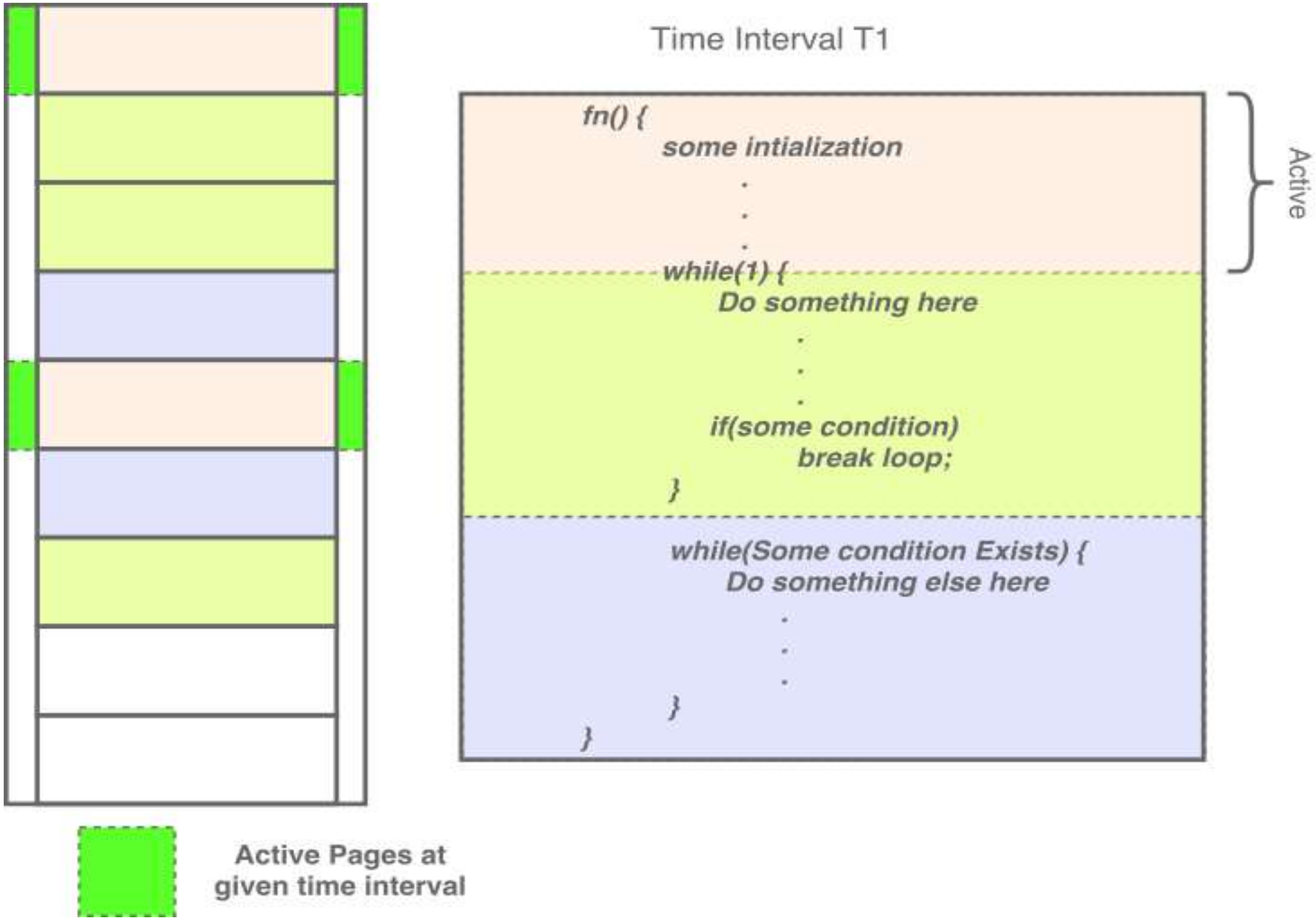


Why the concept of paging, TLB and caches work?

Locality of reference

Locality of reference(Exemplified)

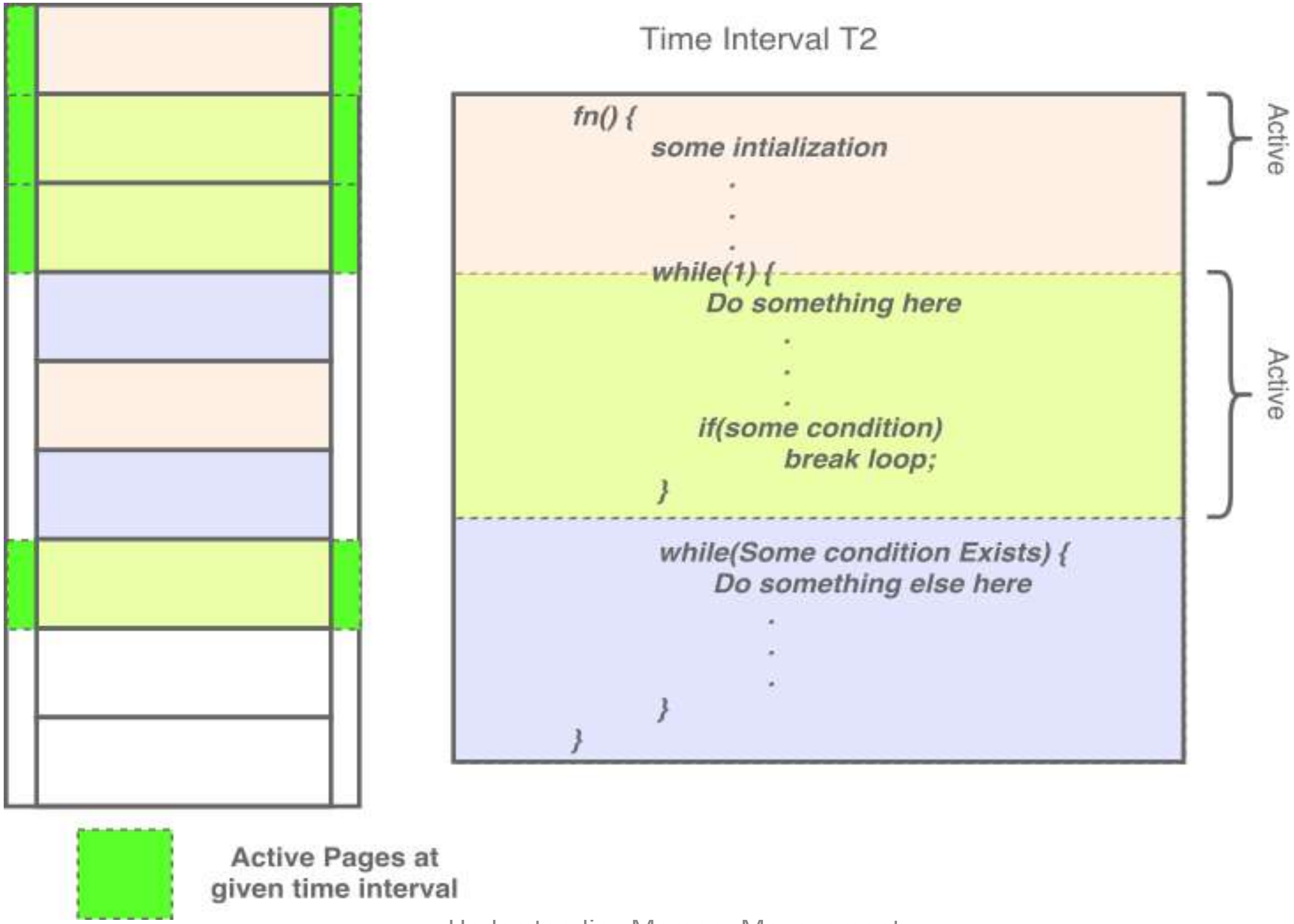
- Time interval **T1**



```
fn() {
        some intialization
                .
                .
                .
        while(1) {
                Do something here
                        .
                        .
                        .
                if(some condition)
                        break loop;
        }
        while(Some condition Exists) {
                Do something else here
                        .
                        .
                        .
        }
}
```

Locality of reference(Exemplified)

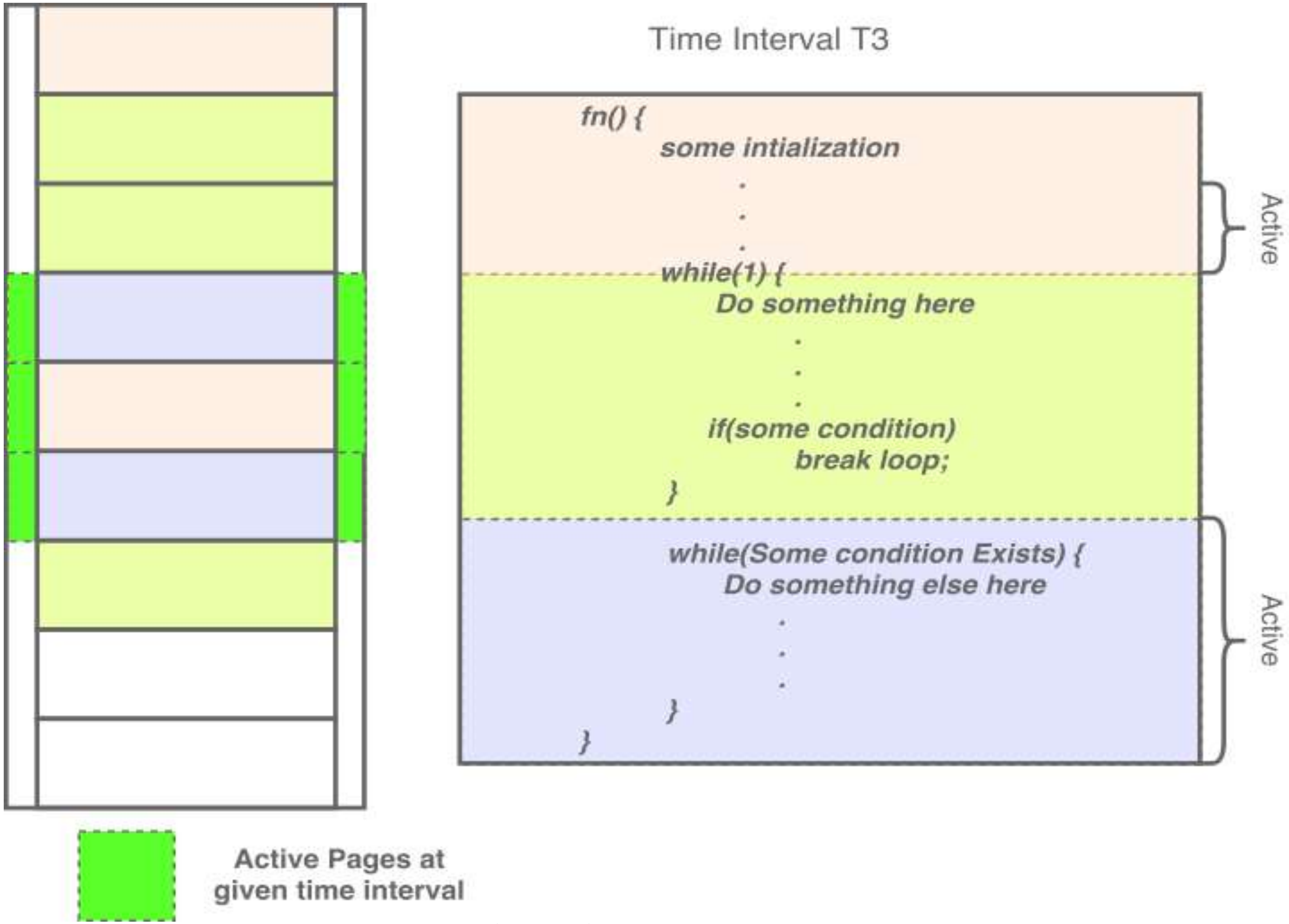
- Time interval **T2**



```
fn() {
        some intialization
                .
                .
                .
        while(1) {
                Do something here
                        .
                        .
                        .
                if(some condition)
                        break loop;
        }
        while(Some condition Exists) {
                Do something else here
                        .
                        .
                        .
        }
}
```

Locality of reference(Exemplified)

- Time interval **T3**



```
fn() {
        some intialization
                .
                .
                .
        while(1) {
                Do something here
                        .
                        .
                        .
                if(some condition)
                        break loop;
        }
        while(Some condition Exists) {
                Do something else here
                        .
                        .
                        .
        }
}
```

Locality of reference(In words)

1. Its the tendency of a programme to cluster references of pages to small set of their pages for extended intervals.

2. There exists a strong Relation between the near future and near past cluster of reference pages, i.e., The set tends to overlap.

3. There exists a feeble or nearly no relation between distant future and distant past references of pages.

4. Pages are accessed in random exhibiting a stochastic behaviour.

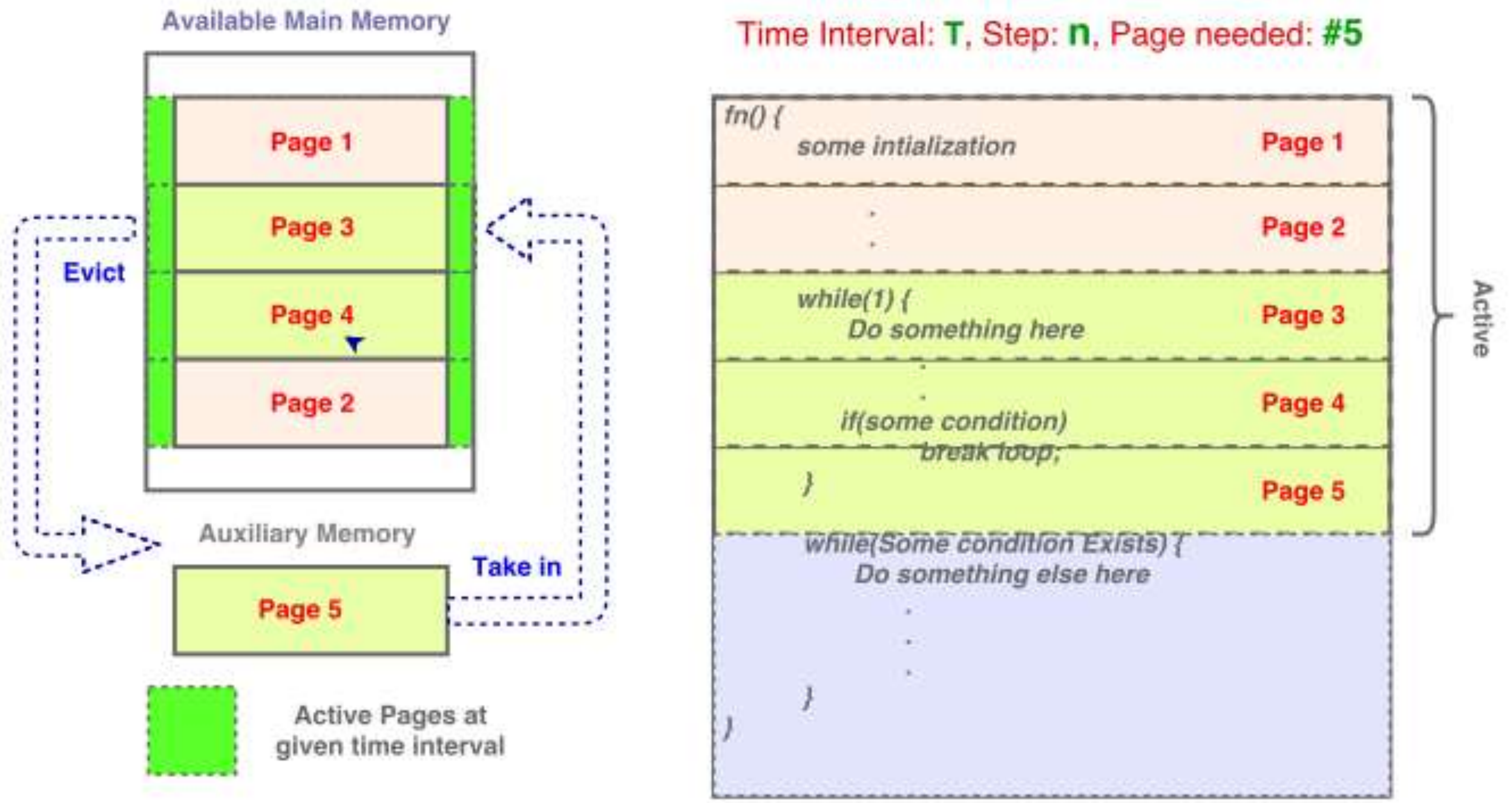
5. The cluster references tends to slowly move away from one active set to another, i.e., They exhibit a quasi stationary behaviour, manifesting a **time series model.**

Thrashing problem

- A process spends significant computation on paging rather than on its real computation.

Thrashing(Exemplified)

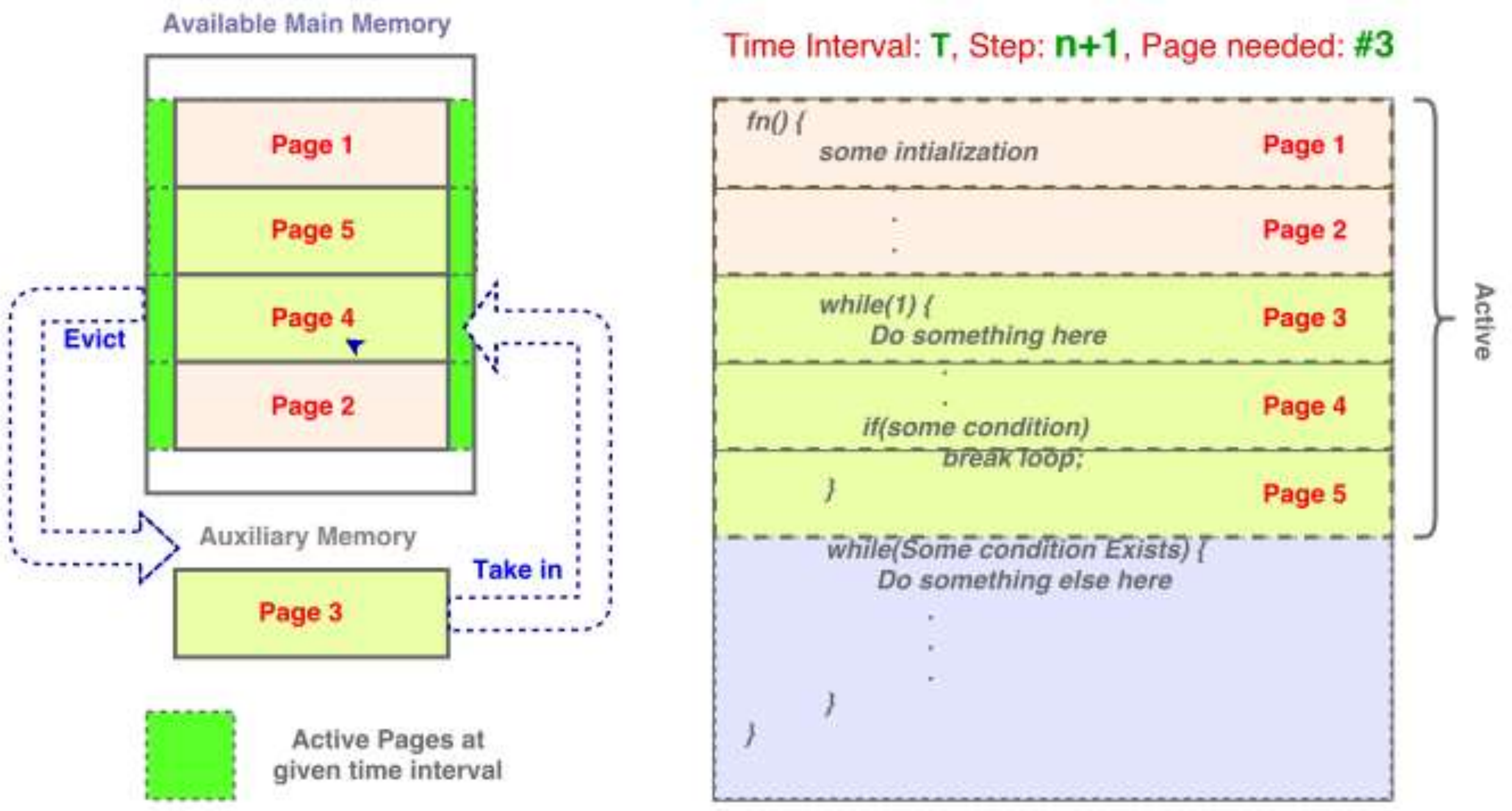
- Required page count at this instance is **5** , but available pages is **4**.



```
fn() {
        some intialization
                .
                .
        while(1) {
            Do something here
                .
                .
            if(some condition)
                break loop;
        }
        while(Some condition Exists) {
            Do something else here
                .
                .
                .
        }
}
```

Thrashing(Exemplified)

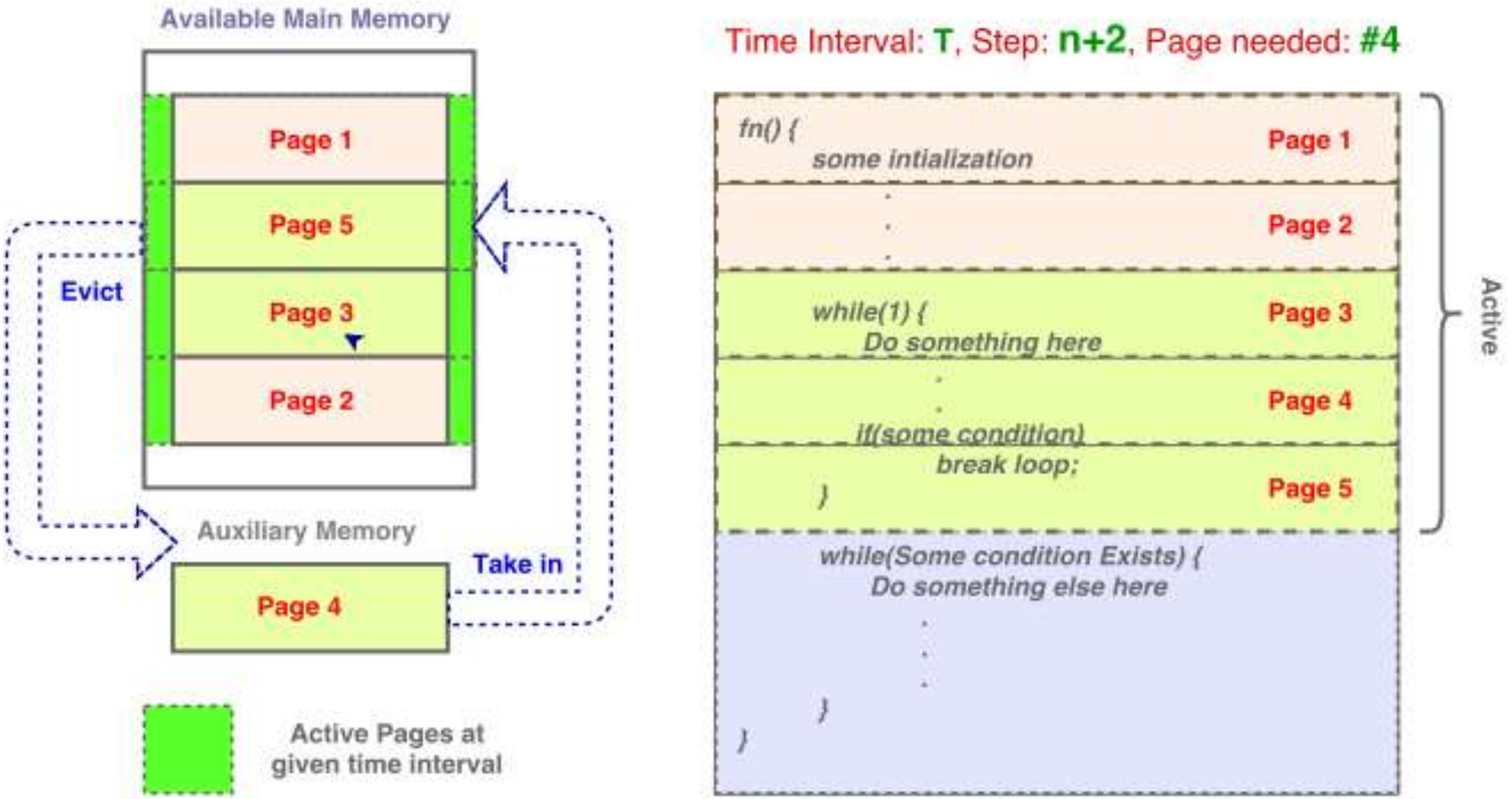
- Required page count at this instance is **5** , but available pages is **4**.



```
fn() {
        some intialization
            .
            .
        while(1) {
            Do something here
                .
                .
            if(some condition)
                break loop;
        }
        while(Some condition Exists) {
            Do something else here
                .
                .
                .
        }
}
```

Thrashing(Exemplified)

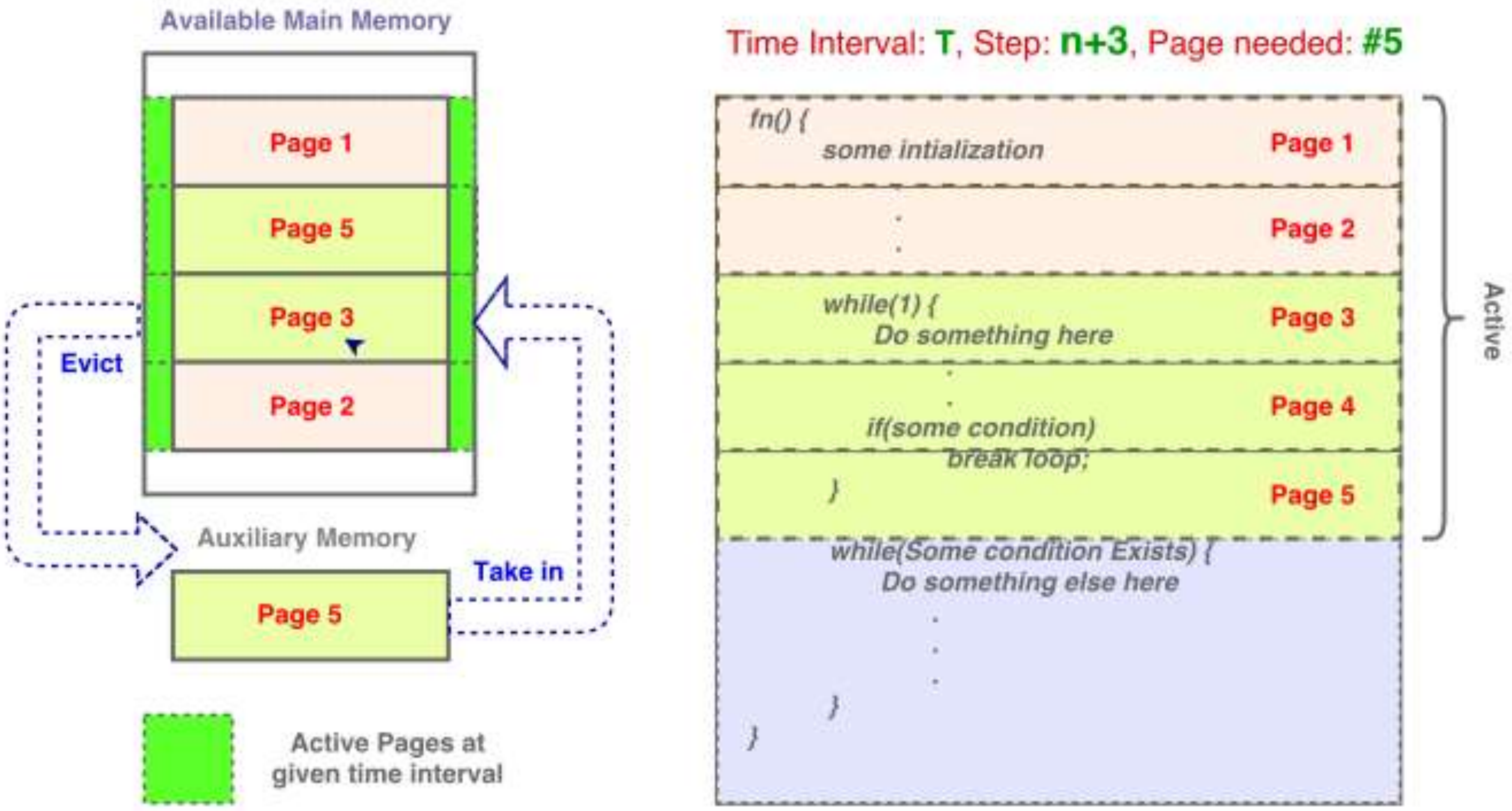
- Required page count at this instance is **5** , but available pages is **4**.



```
fn() {
    some intialization
        .
        .
        .
    while(1) {
        Do something here
            .
            .
        if(some condition)
            break loop;
    }
    while(Some condition Exists) {
        Do something else here
            .
            .
            .
    }
}
```

Thrashing(Exemplified)

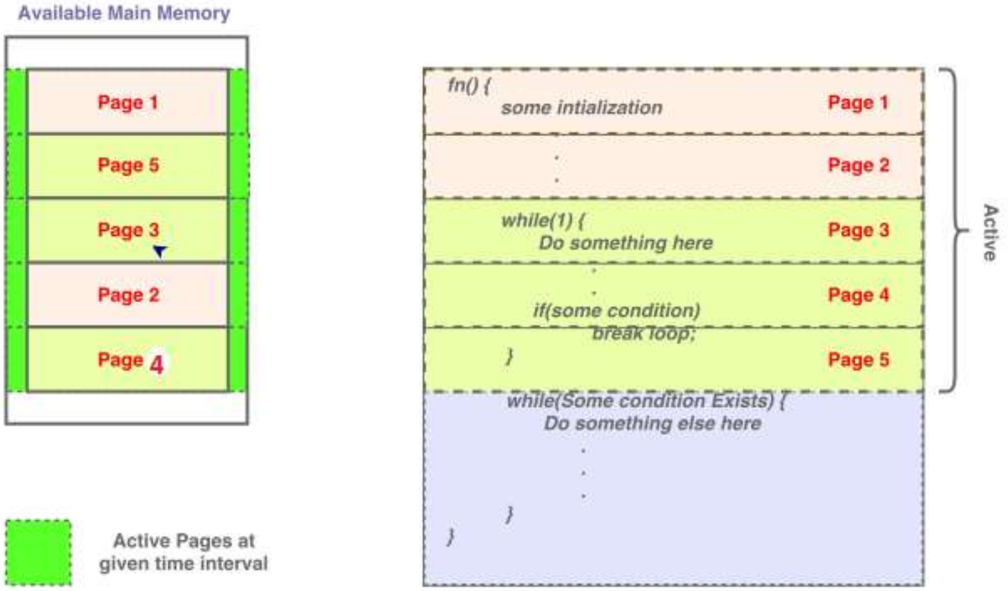
- Required page count at this instance is **5** , but available pages is **4**.



```
fn() {
    some intialization
        .
        .
    while(1) {
        Do something here
            .
            .
        if(some condition)
            break loop;
    }
    while(Some condition Exists) {
        Do something else here
            .
            .
            .
    }
}
```

Working set

- The working set model states that a process can be in main memory, iff all of the pages that it is currently using can be in main memory.



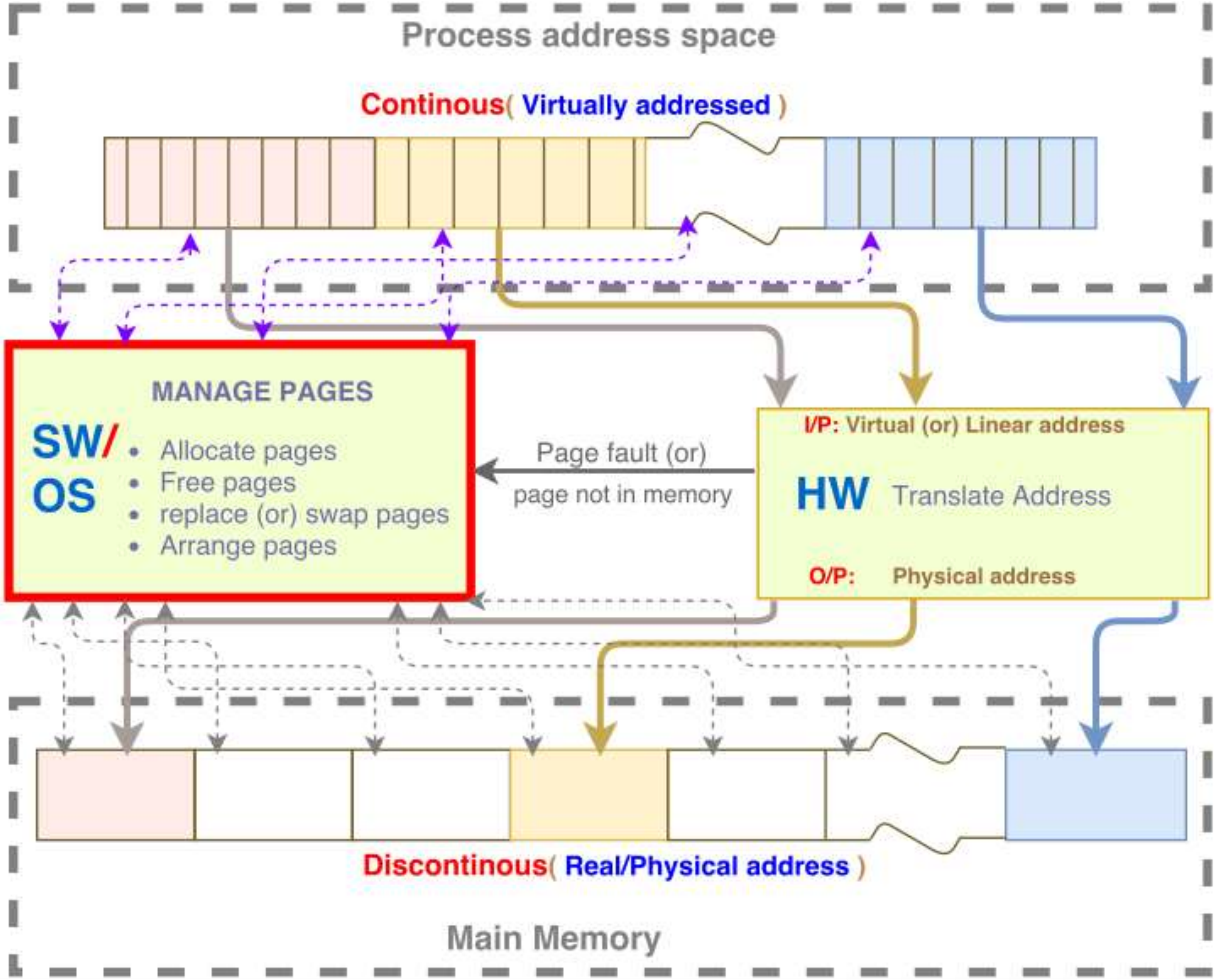
REQUIRED MEMORY: 5
AVAILABLE MEMORY: 5

Now, all pages needed by the programme is in memory, so no thrashing.

OS in Memory management

Policies (or) Rules

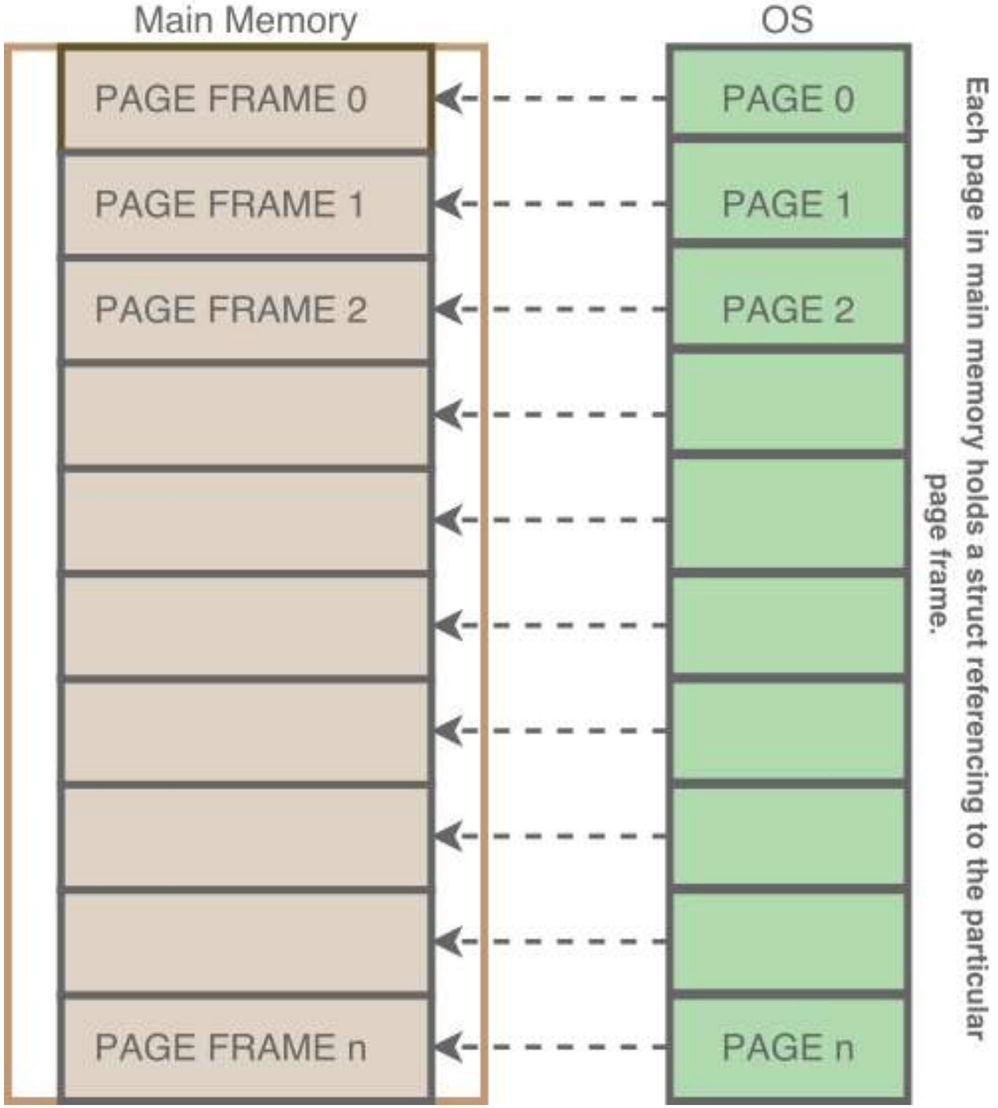
OS in Memory management



OS and MMU

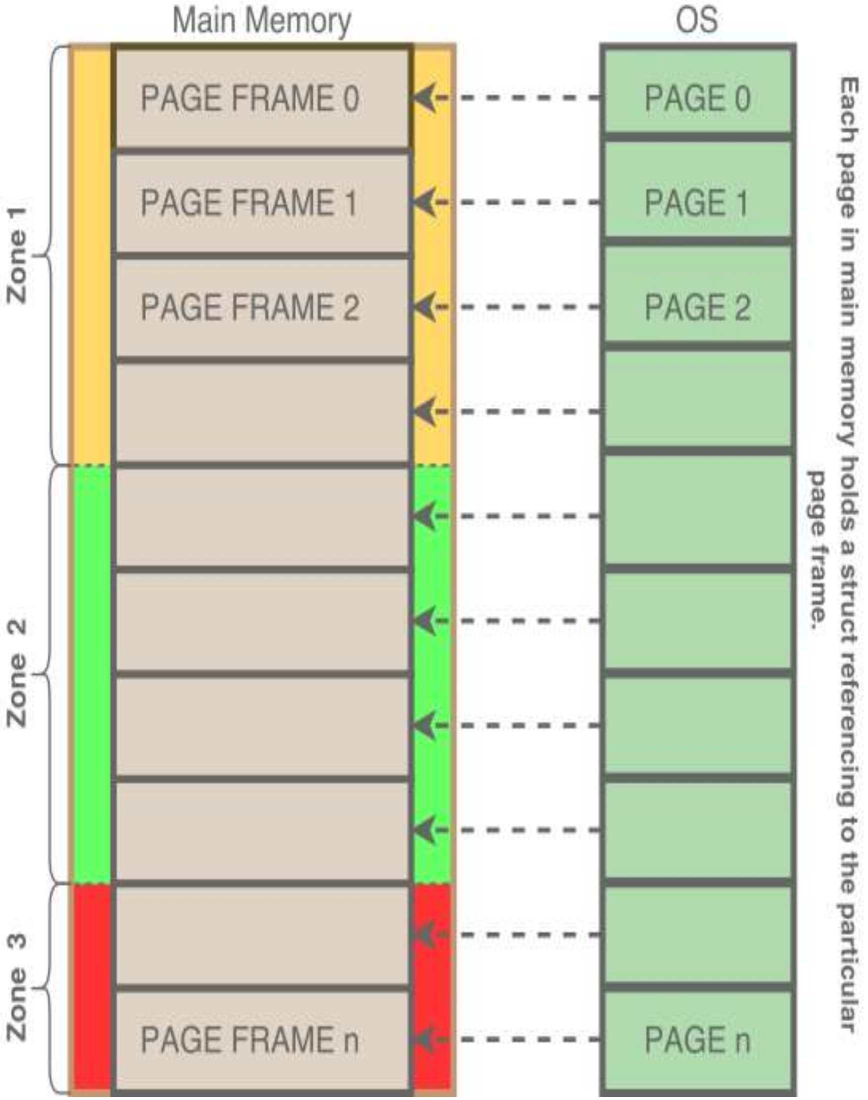
- Hardware facilitates the virtual memory management, however OS determines on optimal usage of this ability.
- Optimal usage: **Employ appropriate decision policies on**
  - Which page to choose on allocation request (Placement policy).
  - Which page to evict on memory exhaustion(Replacement policy).
  - When to take in a page(Fetch policy).
  - Sort the pages for replacement(Scan rate policy).
- To execute the policies, OS needs references to physical pages.

Pages and Page Frame



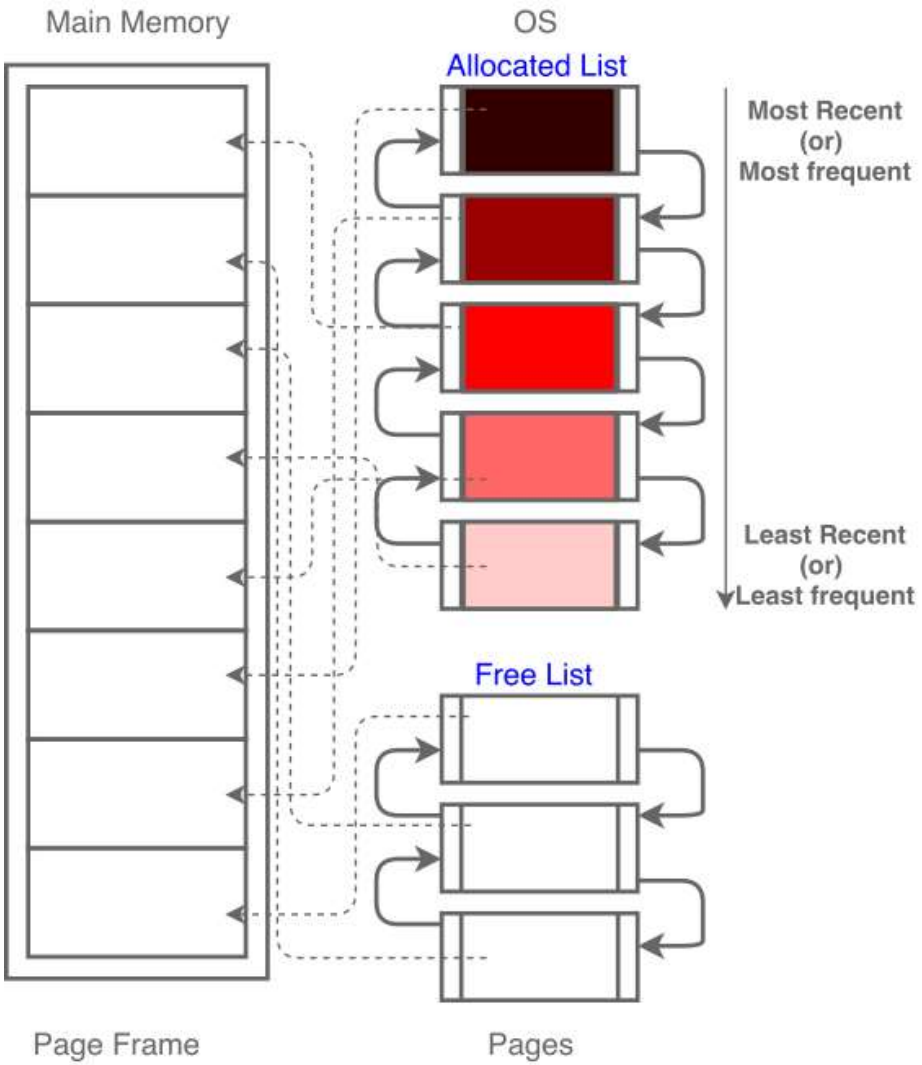
- Divided chunks in main memory is termed as Page frame.
- Each page frame has ***one-to-one*** record associated in OS termed as Page.
- Pages enable OS to entail policies on page management.

Pages, Page frame and Zones



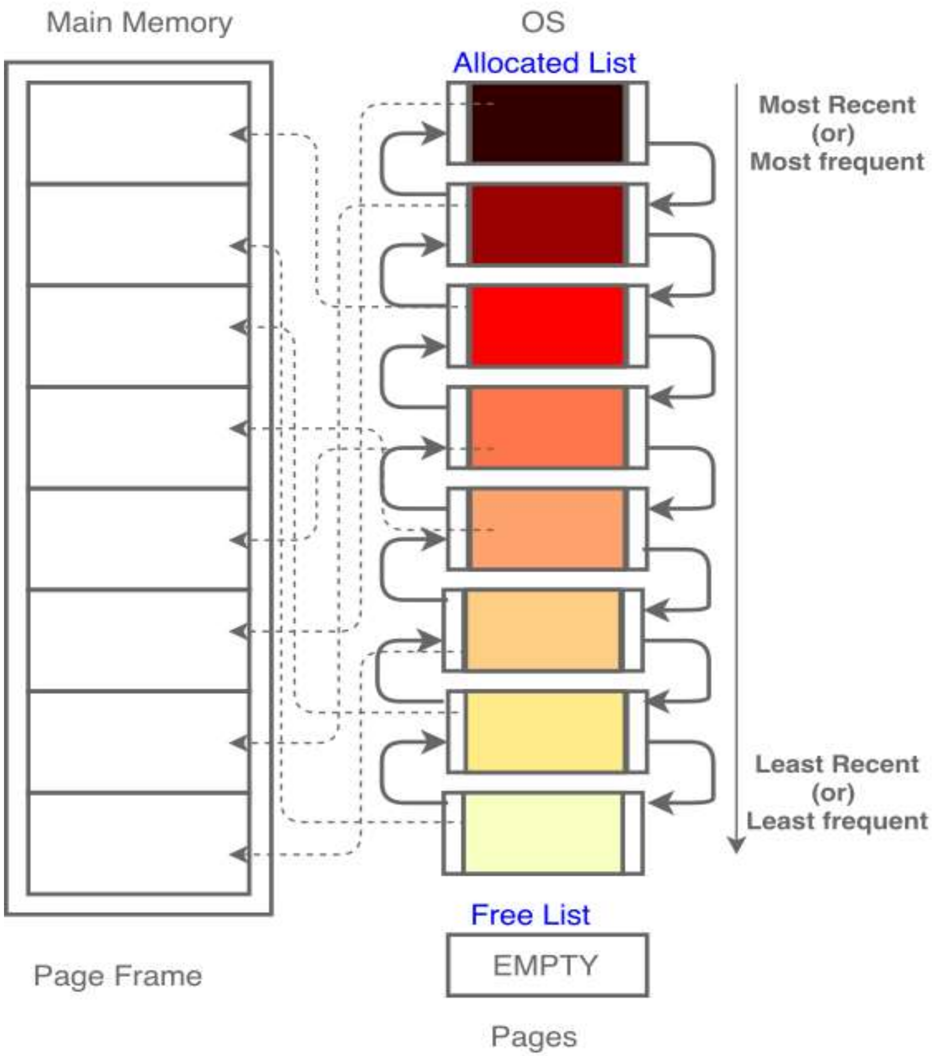
- Pages are further clustered into zones.
- zones are determined based on its location in physical memory and other criteria.
- Enables OS to apply different notions of allocation request.
- Example: Some DMA devices are 16bit can only access lower 65Kb, this would become a zone from which only 16bit DMA allocation request is fulfilled.

Pages and Active list: Ideal scenario



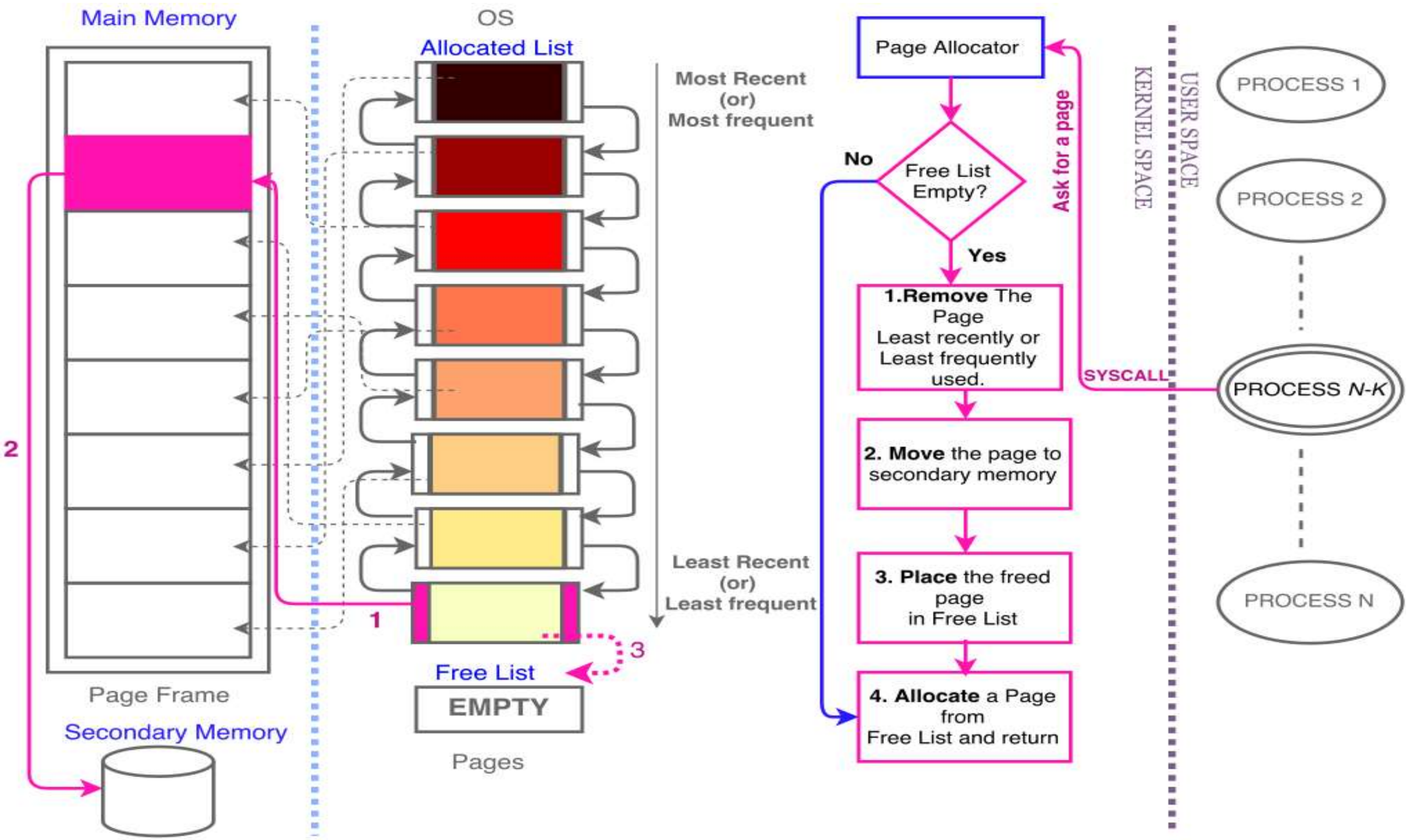
- Pages are kept in linked list.
- Initially, pages would be in free list.
- On request for memory, page is allocated from free list and moved to allocated or active list.
- quintessentially, pages in active list are sorted in descending order based on access or frequency.

Pages and Active list: Exhausted scenario



- Main memory could reach a point of exhaustion or near the point of exhaustion.
- After this point, if a process needs more memory, then system should not fail.
- Replacement Policy: OS should evict an appropriate page from main memory to auxiliary memory to pave way for new memory requirement.

Replacement Policy(Simplified)

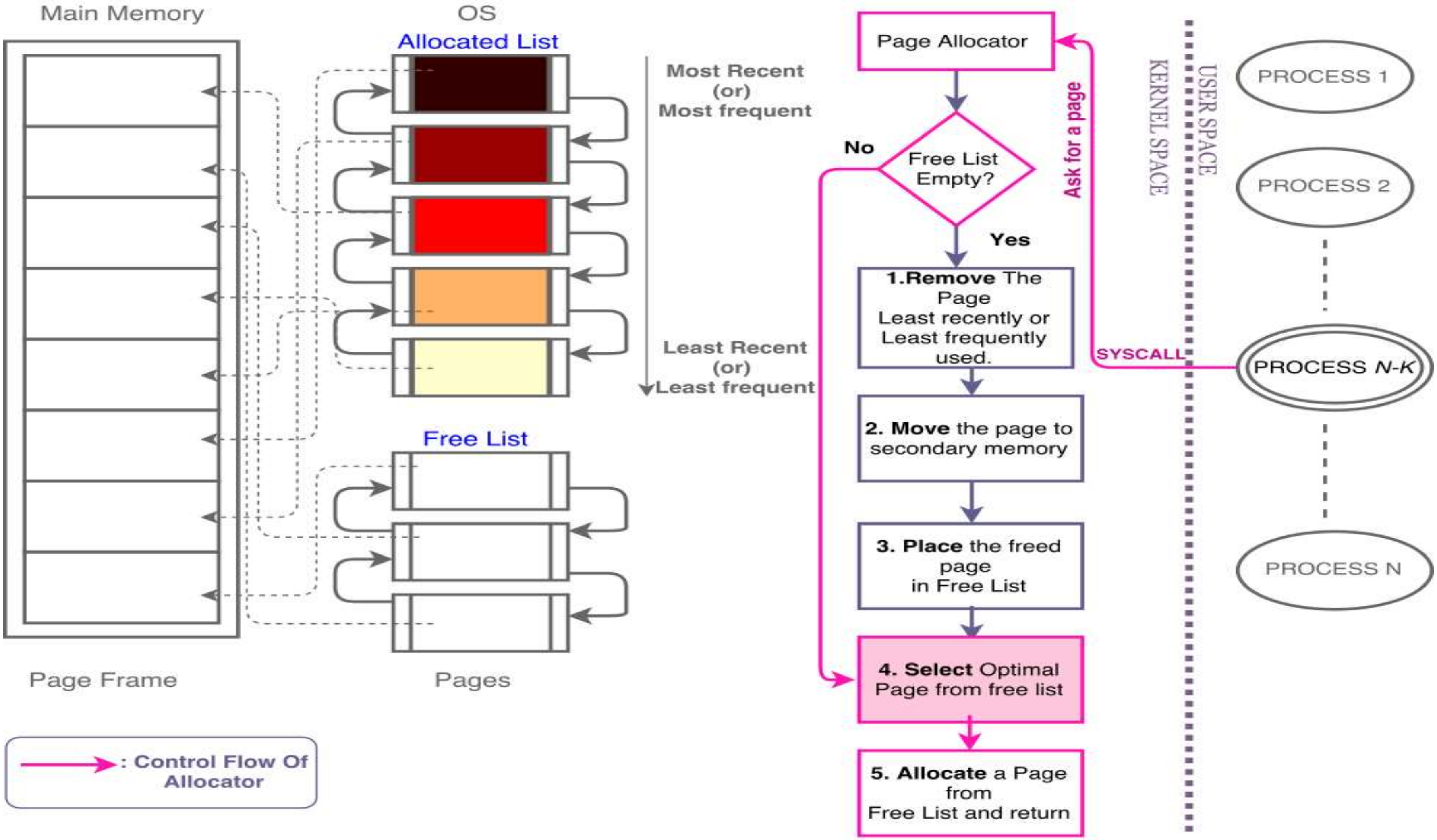


Placement Policy

- Tries to address following questions during allocation.
  - Are some physical pages preferable to others within a process?
  - Does allocating such a preferable page increases performance of the process?

- Addresses these query by selecting optimal page from the free list.

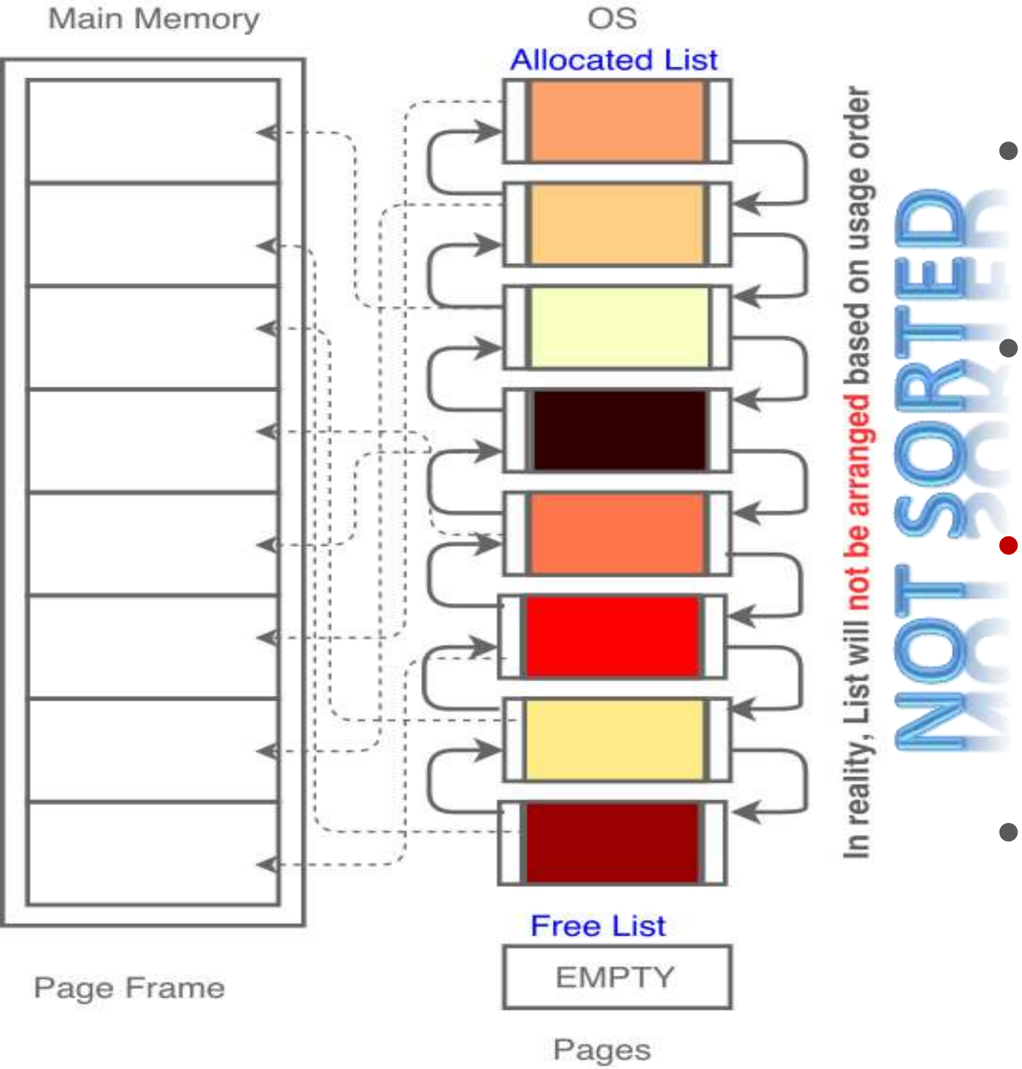
Placement Policy(Simplified)


Main Memory
OS
Allocated List
Most Recent
(or)
Most frequent
Least Recent
(or)
Least frequent
Free List
Page Frame
Pages
: Control Flow Of Allocator
Page Allocator
Free List Empty?
No
Yes
1.Remove The Page Least recently or Least frequently used.
2. Move the page to secondary memory
3. Place the freed page in Free List
4. Select Optimal Page from free list
5. Allocate a Page from Free List and return
Ask for a page
SYSCALL
KERNEL SPACE
USER SPACE
PROCESS 1
PROCESS 2
PROCESS N-K
PROCESS N

Placement Policy: Optimal Page

- Pick a right page from free list such that:
  - It reduces cache conflict or thrashing. (or/and)
  - It is near to the process's processor (NUMA). (or/and)
  - It saves energy. (or/and)
  - Increases parallelization.

Active List: Reality

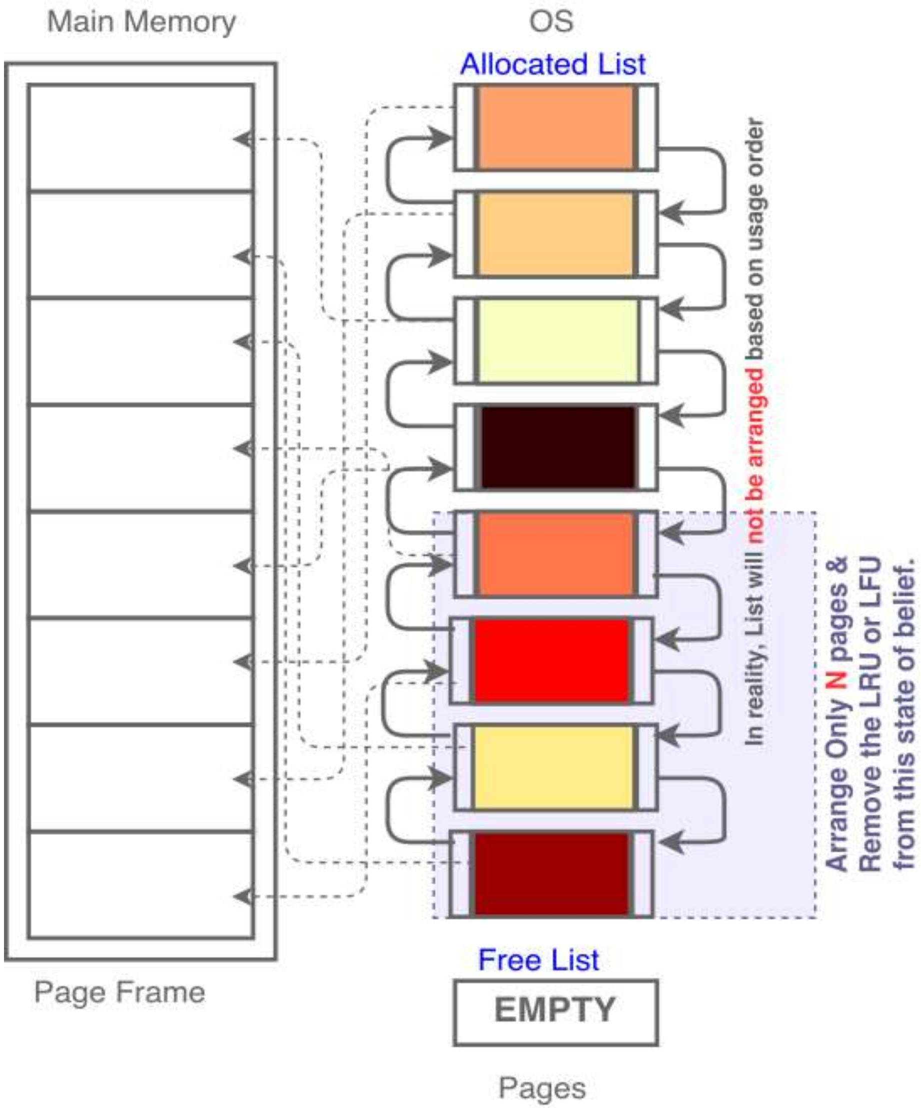


In reality, List will not be arranged based on usage order

NOT SORTED

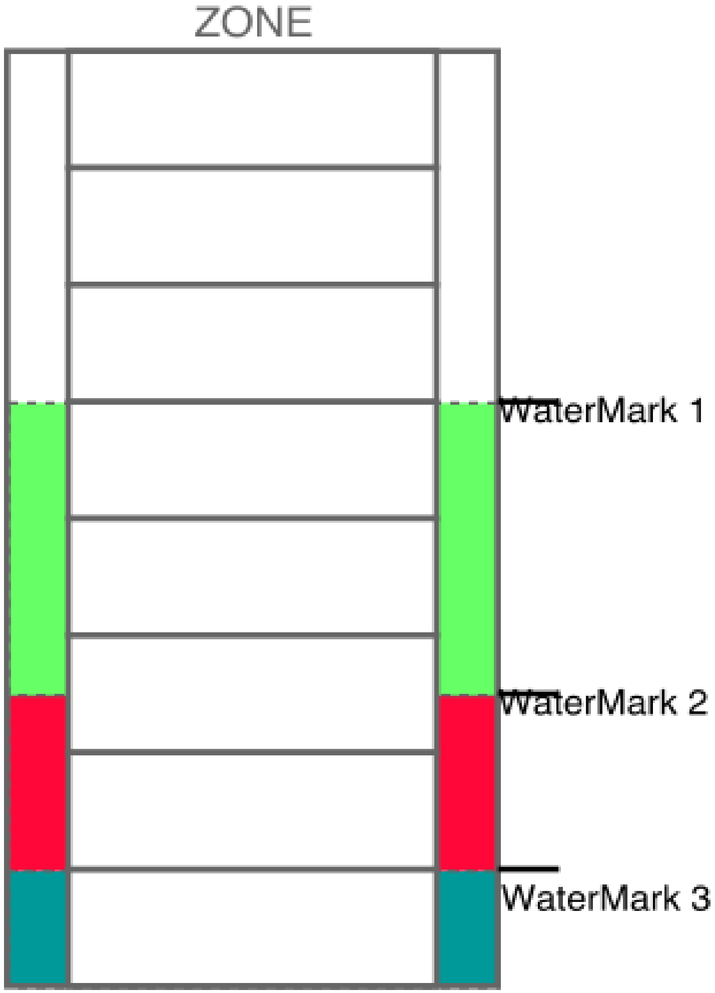
- In reality, pages in active list will not be sorted.
- In General, a system might have 1000s of active pages.
- Sorting the entire list periodically is a CPU intensive computation.
- Might reduce the system’s performance considerably.

Scan rate Policy


Main Memory
OS
Allocated List
In reality, List will not be arranged based on usage order
Arrange Only N pages & Remove the LRU or LFU from this state of belief.
Page Frame
Free List
EMPTY
Pages

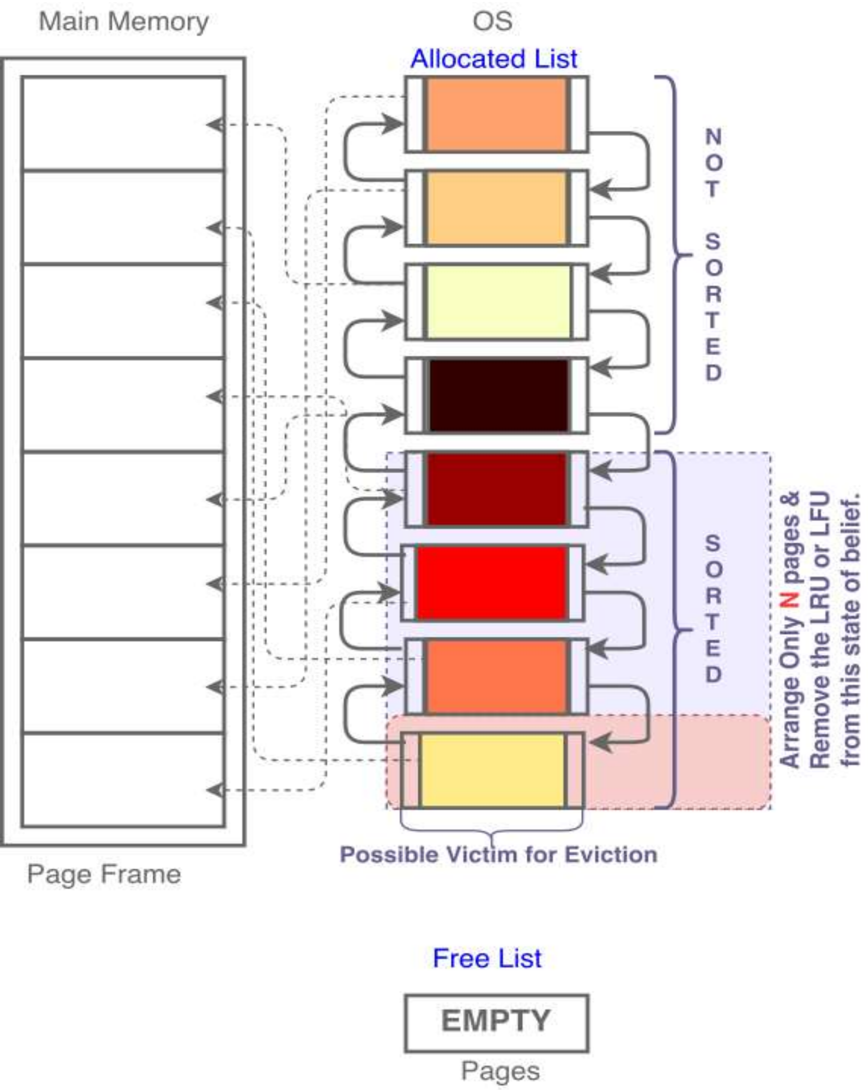
- Scanning: A process of arranging the pages in active list on accordance to replacement policy.
- Scan rate policy: Determines on scanning by deciding on 2 parameters
  - How much to scan.
  - When to scan.
- In general, scanning and replacement are done jointly.

Scan rate Policy: When to Scan



- Determines when to start and stop scanning.
- Determines different scan rate policy to apply based on watermark.
- Watermark: Zones have way points based on their distance from exhaustion.
- In general, watermark are static, determined by userspace variable (or) computed offline.
- Watermark also determines how much to scan.

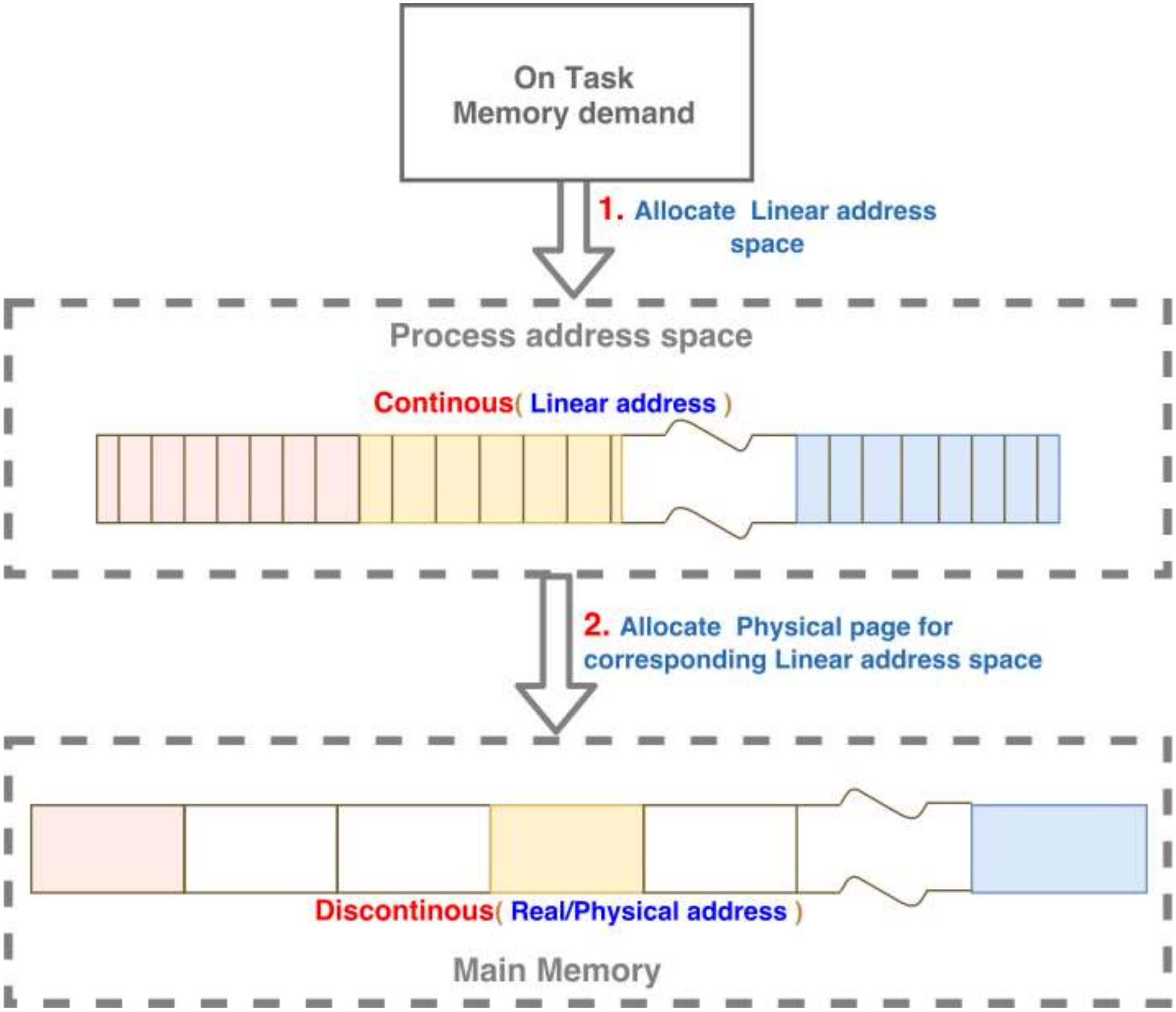
Scan rate Policy: How much to scan


Main Memory
OS
Allocated List
NOT SORTED
SORTED
Arrange Only N pages & Remove the LRU or LFU from this state of belief.
Possible Victim for Eviction
Page Frame
Free List
EMPTY
Pages

- Determines the measure on the range to scan within active list.
- Directly impacts the performance of the system.
- In general, nearer to exhaustion more the scanning, as more pages needs eviction.

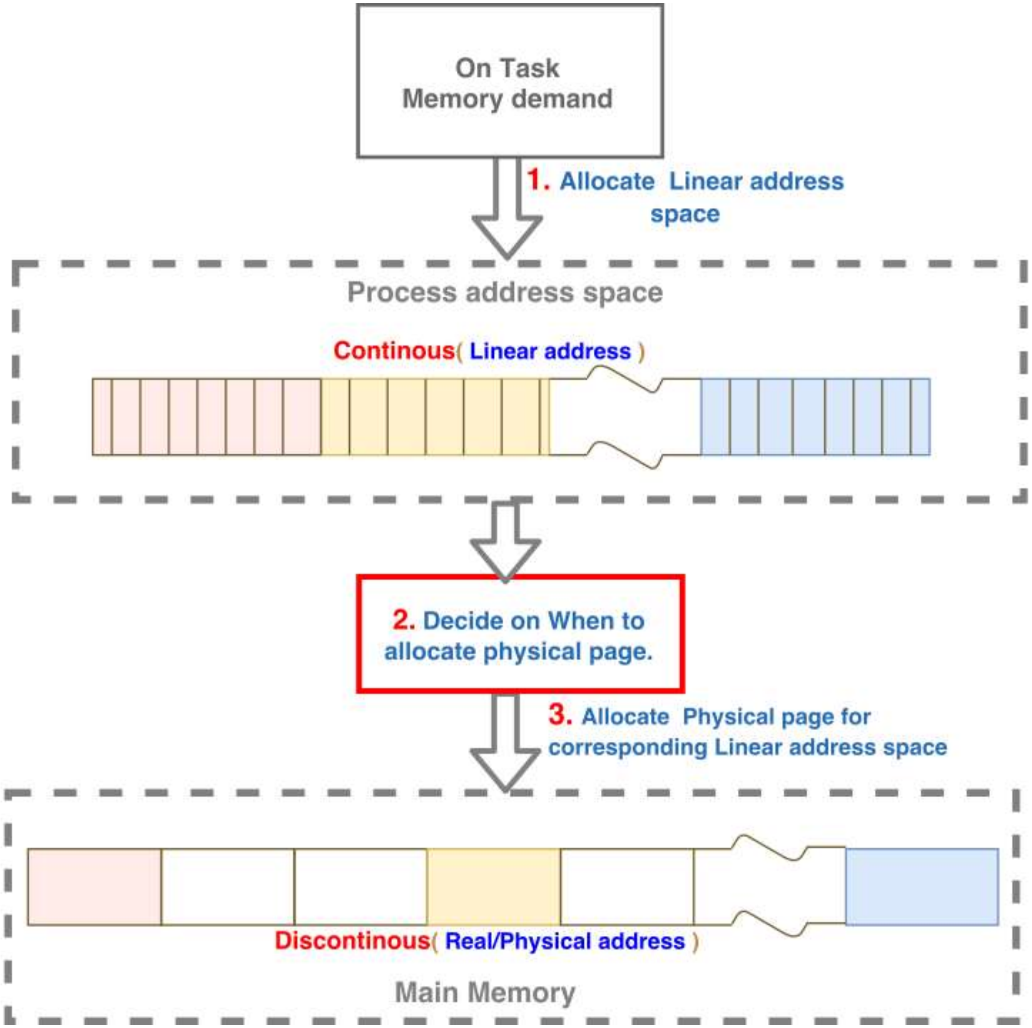
Address and Memory in OS

OS divides address and memory into **two** different entities.



On Task
Memory demand
1. Allocate Linear address space
Process address space
Continous(Linear address)
2. Allocate Physical page for corresponding Linear address space
Discontinous(Real/Physical address)
Main Memory

Fetch Policy



On Task Memory demand

1. Allocate Linear address space

Process address space

Continous(Linear address)

2. Decide on When to allocate physical page.

3. Allocate Physical page for corresponding Linear address space

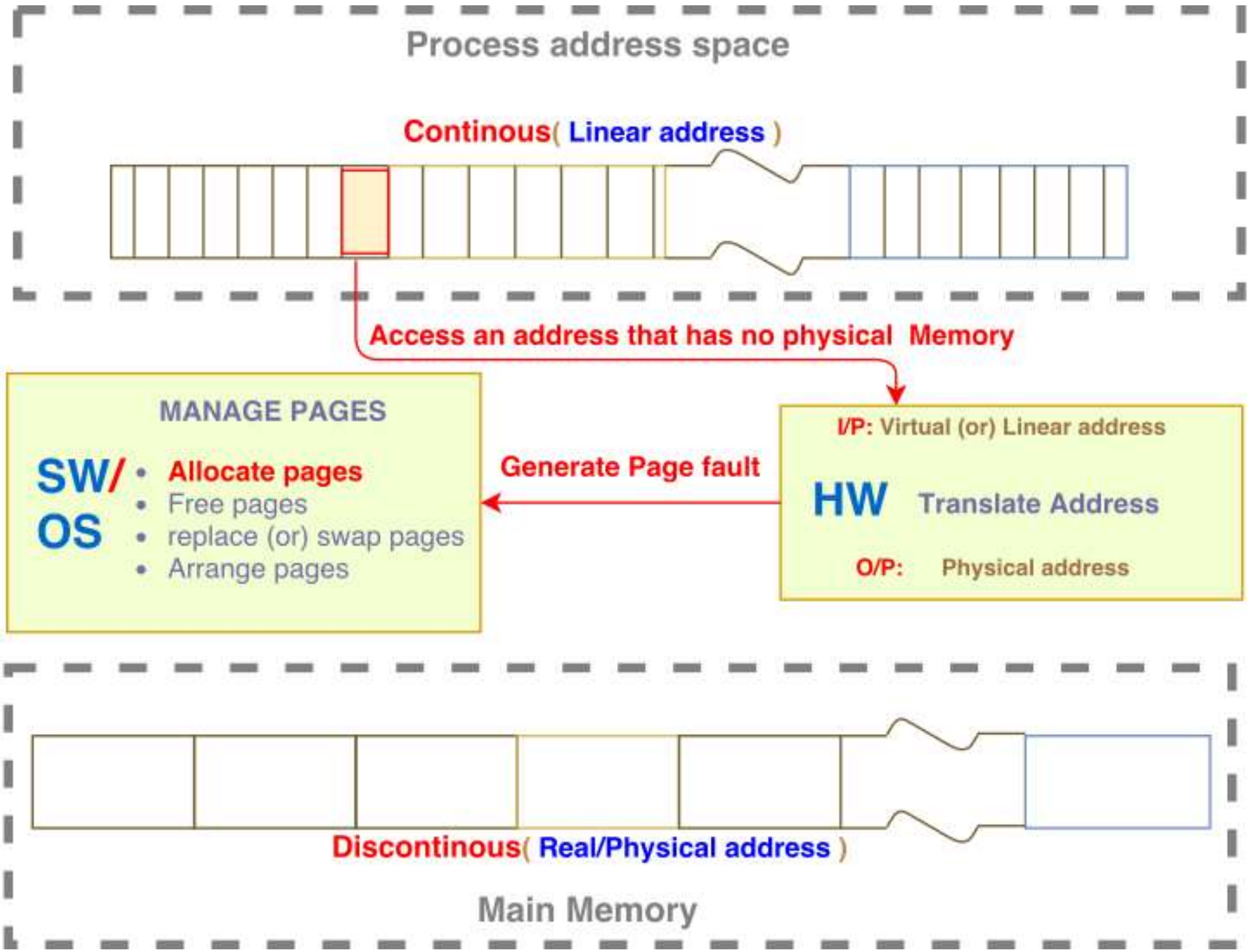
Discontinous(Real/Physical address)

Main Memory

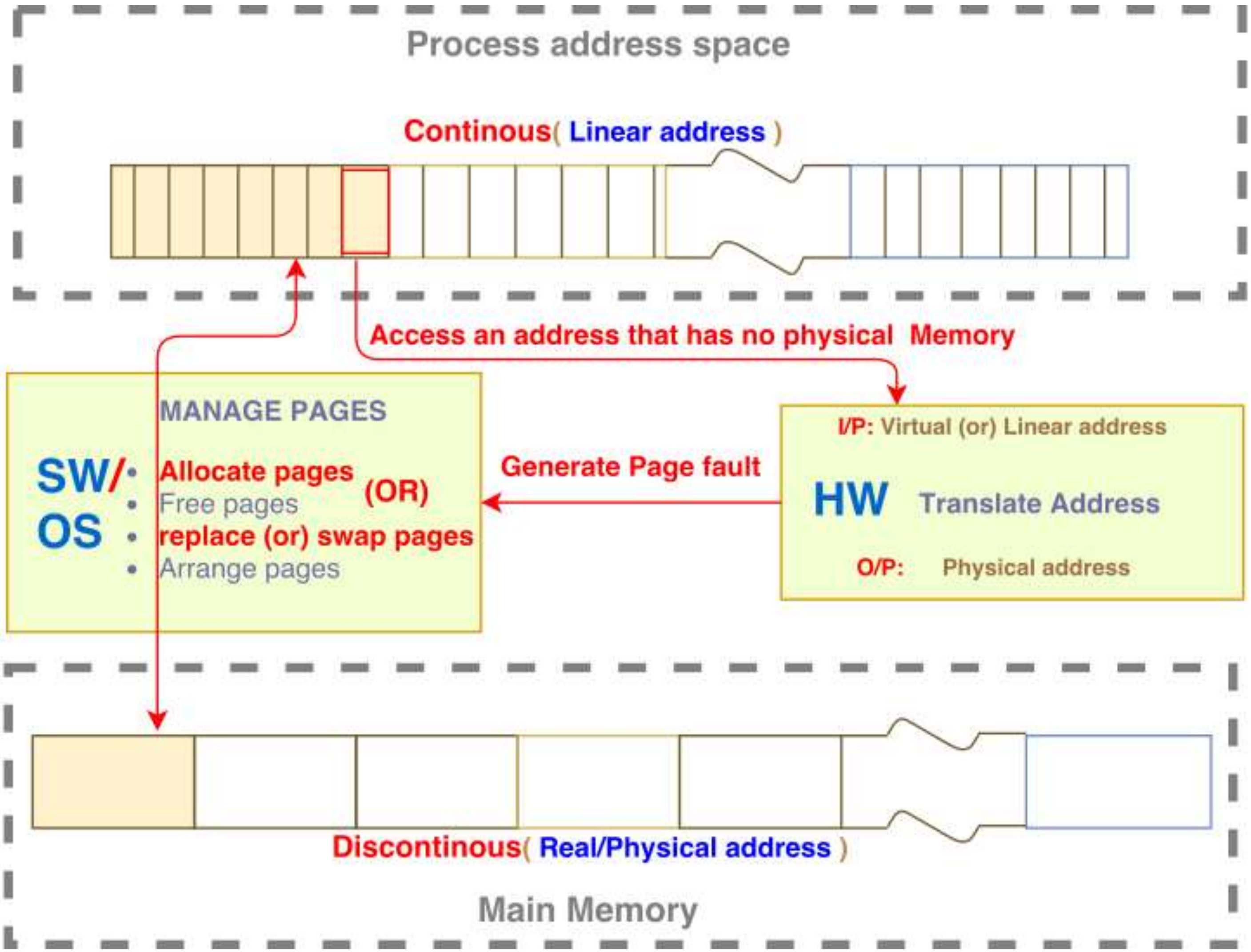
Fetch Policy

- When to fetch the page from auxiliary memory to main memory.
- 2 Approaches:
  - Demand Paging only brings pages into main memory when a reference is made to a location on the page.
  - Prepaging brings in pages whose use is anticipated.

Fetch Policy (Demand Paging)


Process address space
Continous(Linear address)
Access an address that has no physical Memory
MANAGE PAGES
SW/
OS
• Allocate pages
• Free pages
• replace (or) swap pages
• Arrange pages
Generate Page fault
I/P: Virtual (or) Linear address
HW
Translate Address
O/P: Physical address
Discontinous(Real/Physical address)
Main Memory

Fetch Policy (Demand Paging)



Process address space

Continous(Linear address)

Access an address that has no physical Memory

MANAGE PAGES

SW/ OS

- Allocate pages
- Free pages
- replace (or) swap pages
- Arrange pages

(OR)

Generate Page fault

I/P: Virtual (or) Linear address

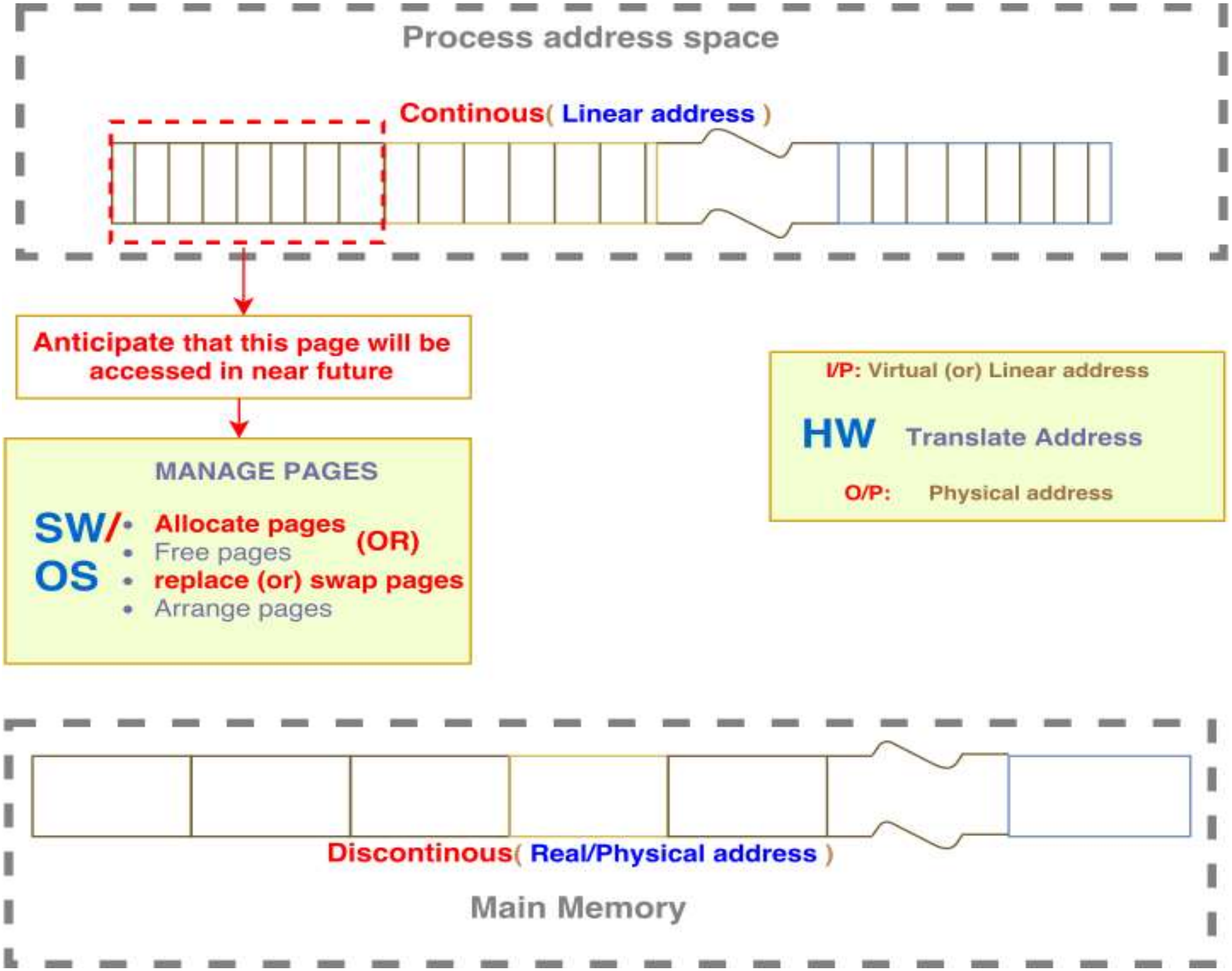
HW Translate Address

O/P: Physical address

Discontinous(Real/Physical address)

Main Memory

Fetch Policy (Prefetching)



Process address space

Continous(Linear address)

Anticipate that this page will be accessed in near future

MANAGE PAGES

SW/OS

- Allocate pages
- Free pages
- replace (or) swap pages
- Arrange pages

(OR)

I/P: Virtual (or) Linear address

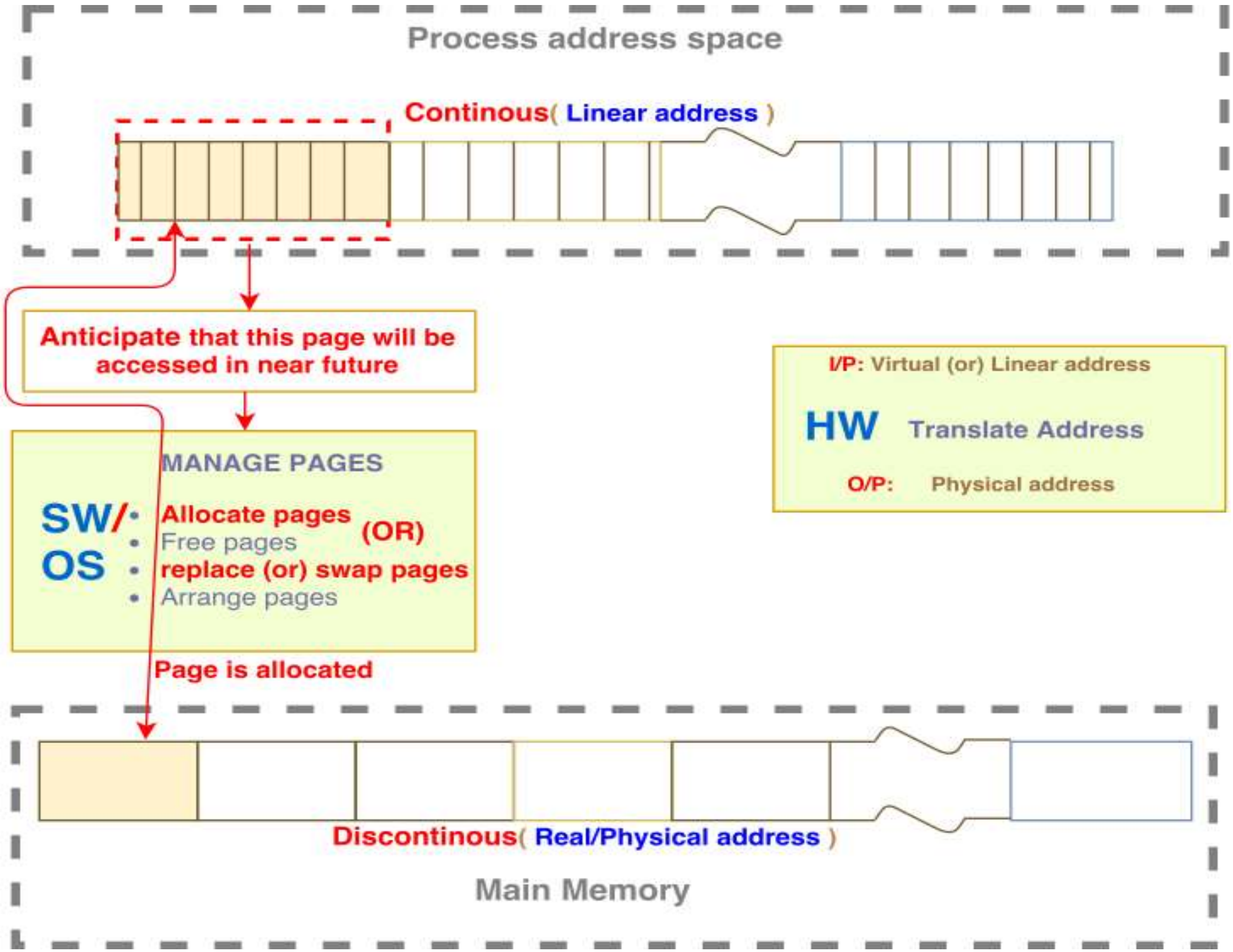
HW Translate Address

O/P: Physical address

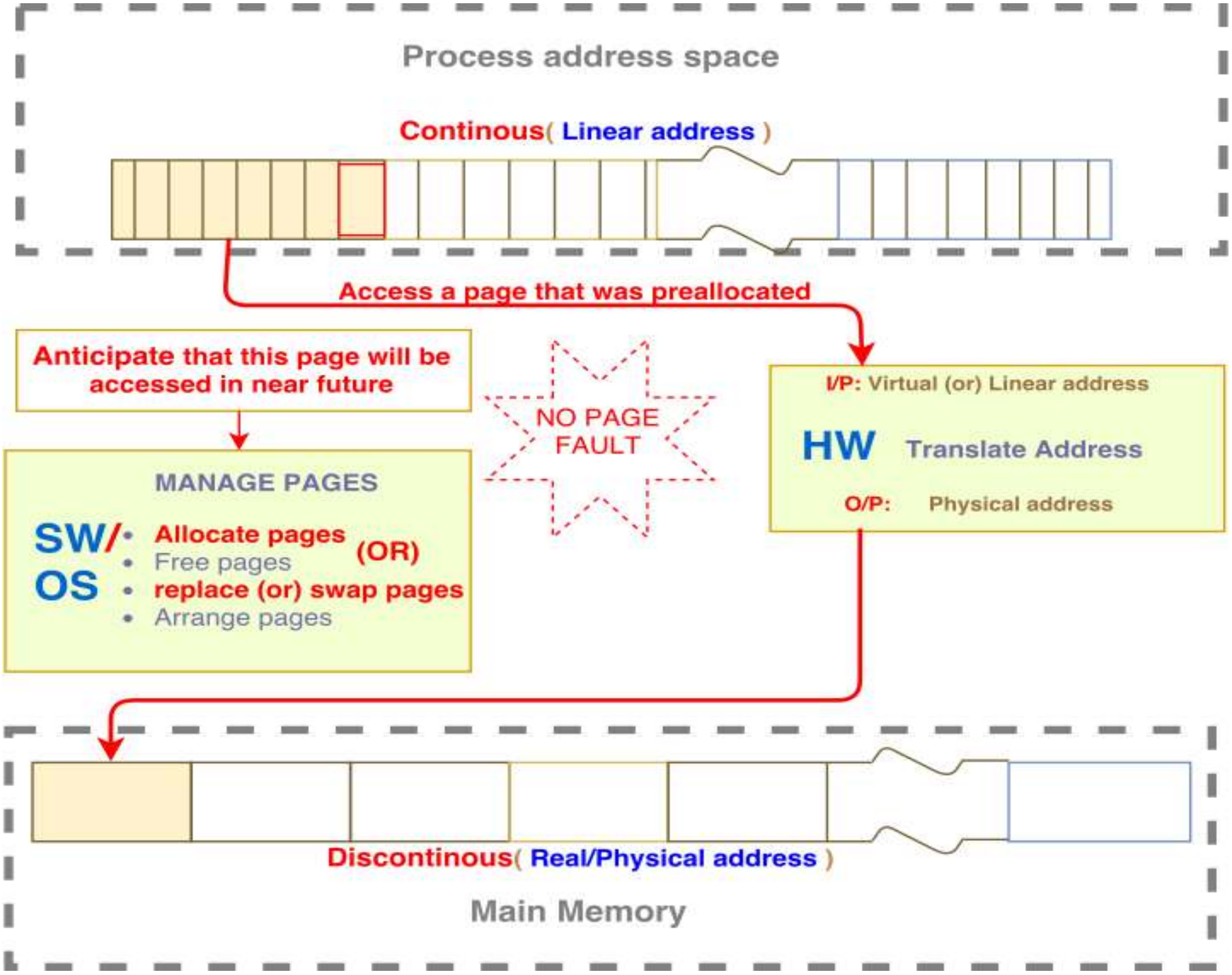
Discontinous(Real/Physical address)

Main Memory

Fetch Policy (Prefetching)



Fetch Policy (Prefetching)



Process address space

Continous(Linear address)

Access a page that was preallocated

Anticipate that this page will be accessed in near future

MANAGE PAGES

SW/OS

- Allocate pages
- Free pages
- replace (or) swap pages
- Arrange pages

(OR)

NO PAGE FAULT

I/P: Virtual (or) Linear address

HW Translate Address

O/P: Physical address

Discontinous(Real/Physical address)

Main Memory

Thank You