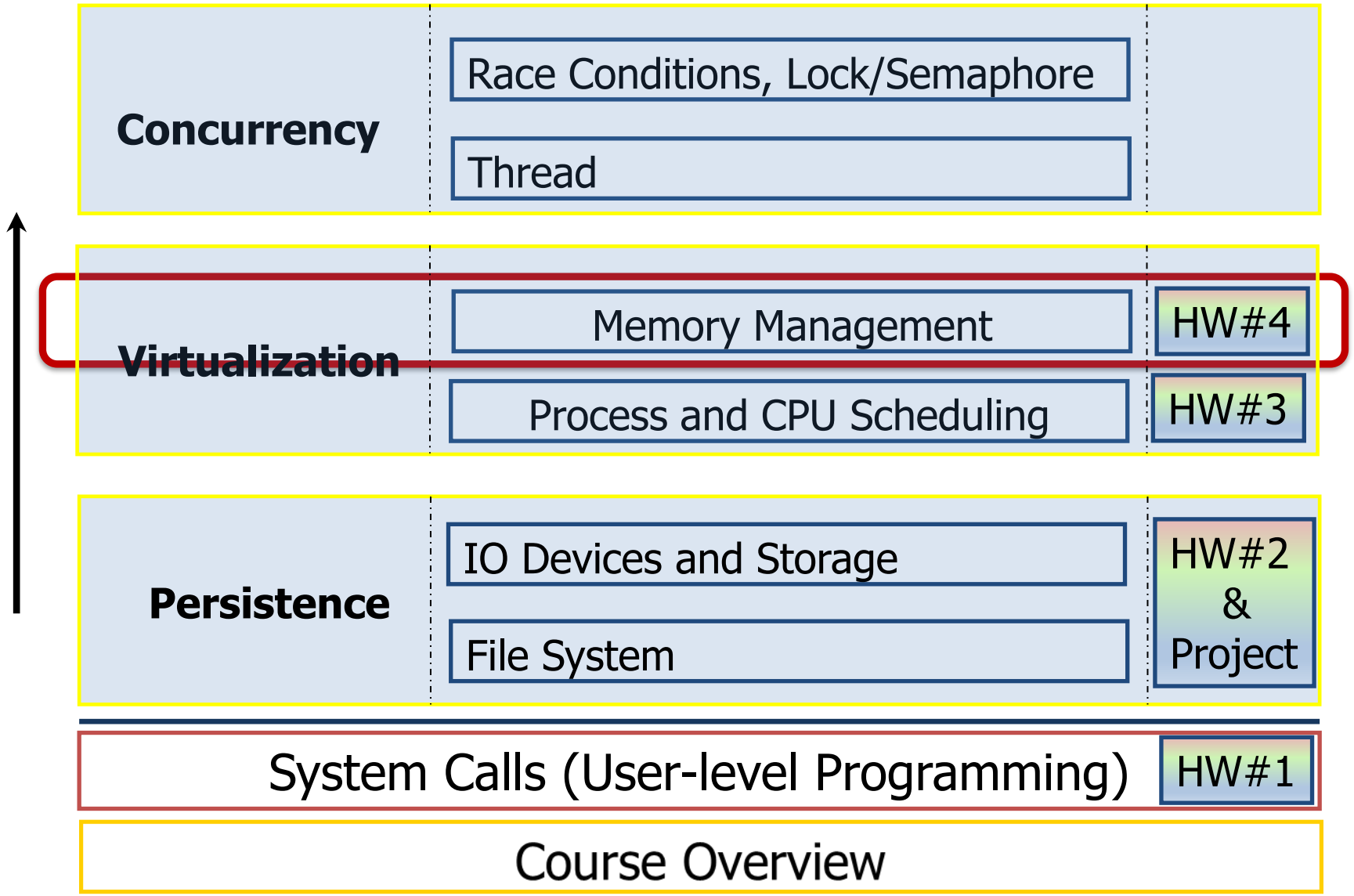
Lecture 12: Virtualizing Memory – Swapping

The Course Organization (Bottom-up)

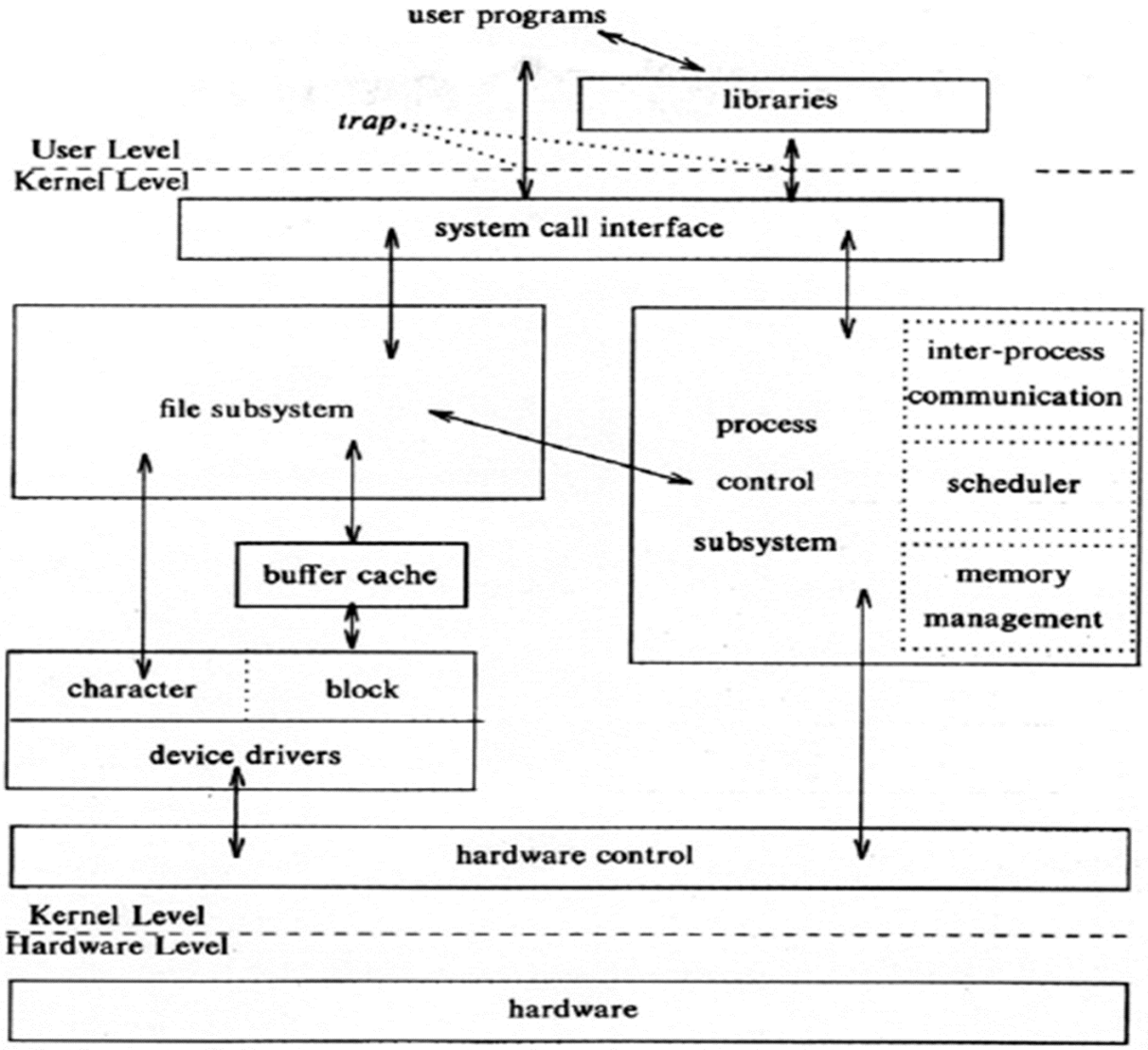


OS – Resource management via virtualization

OS provides services via **System Call** (typically a few hundred) to run **process**, access memory/devices/files, etc.

The OS **manages resources** such as *CPU*, *memory* and *disk* via **virtualization**.

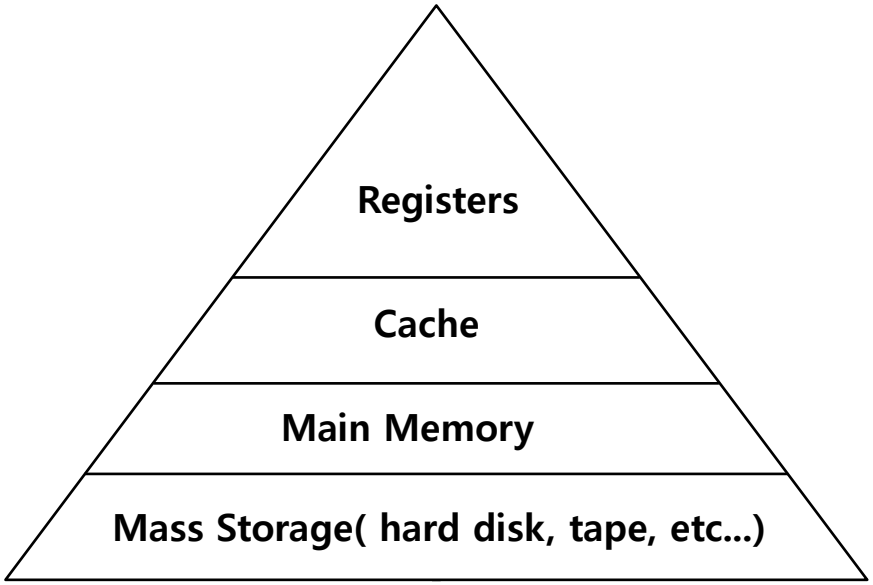
- many programs to run (processes) → Sharing the CPU
- many processes to *concurrently* access their own instructions and data → Sharing memory
- many processes to access devices → Sharing disks



The Design Of The Unix Operating System (Maurice Bach, 1986)

Beyond Physical Memory: Mechanisms

- Require an additional level in the memory hierarchy.
  - OS needs a place to stash away portions of address space that currently aren't in great demand.
  - In modern systems, this role is usually served by a hard disk drive



Memory Hierarchy in modern system

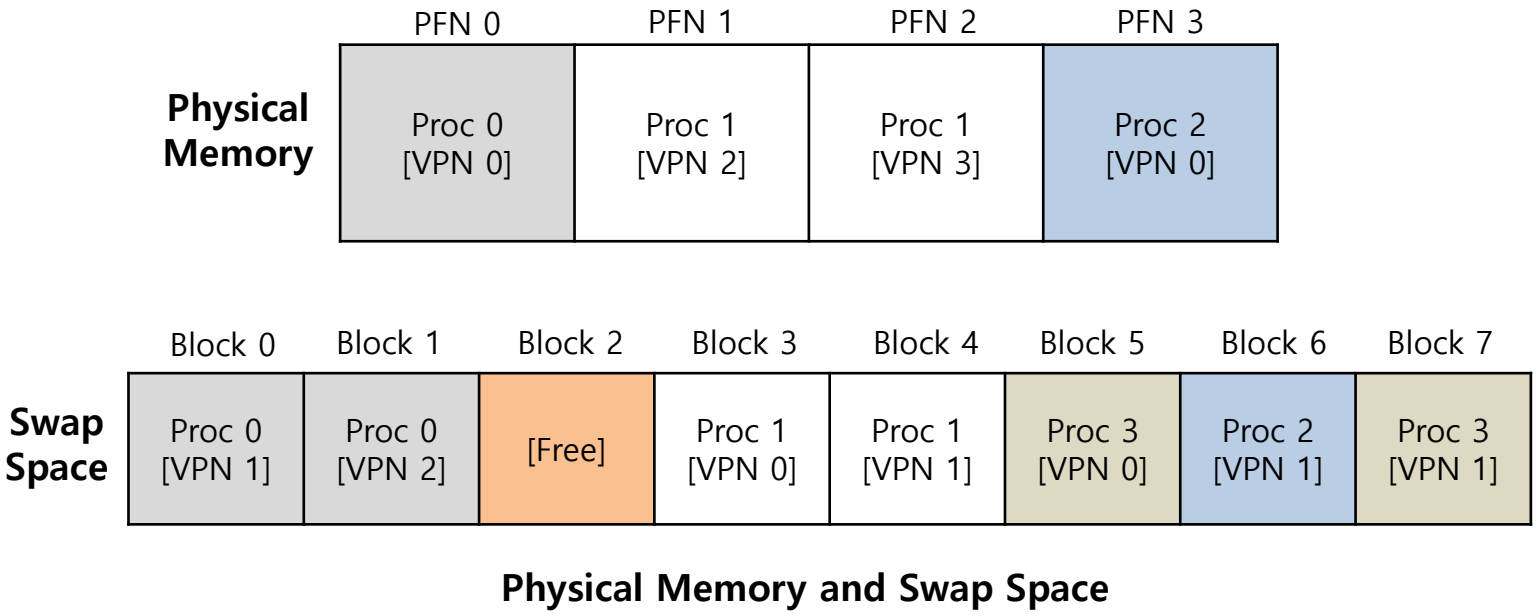
Single large address for a process

- Always need to first arrange for the code or data to be in memory before calling a function or accessing data.

- Multiple processes rather than a single process
  - The addition of swap space allows the OS to support the illusion of a large virtual memory for multiple concurrently-running processes

Swap Space

- Reserve some space on the disk for moving pages back and forth.
- OS needs to remember page addresses in the swap space, in page-sized unit



Physical Memory and Swap Space

What If Memory Is Full ?

- The OS moves out pages to make room for the new pages
  - The process of picking a page to kick out, or replace is known as page-replacement policy
- Add some machinery in the system to support swapping pages to and from the disk.
  - When the hardware looks in the PTE, it may find that the page is not present in physical memory.

Value	Meaning
1	page is present in physical memory
0	The page is not in memory but rather on disk.

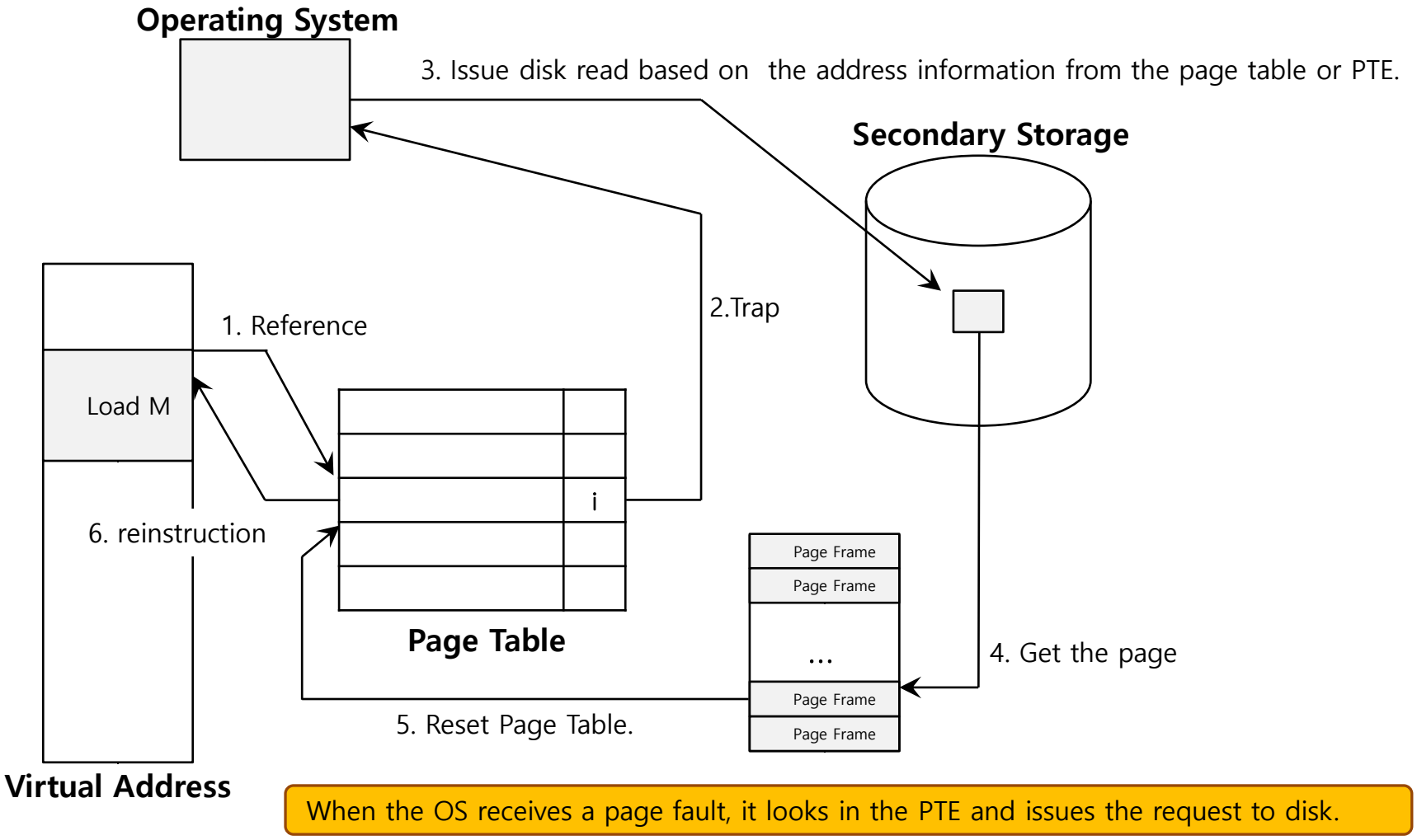
The Page Fault

- Accessing page that is not in physical memory.
  - If a page is not present and has been swapped disk, the OS needs to swap the page into memory in order to service the page fault.

Value	Meaning
1	page is present in physical memory
0	The page is not in memory but rather on disk.

Page Fault Control Flow

- PTE used for data such as the PFN of the page for a disk address.



When the OS receives a page fault, it looks in the PTE and issues the request to disk.

Page Fault Control Flow – Hardware

```
1:      VPN = (VirtualAddress & VPN_MASK) >> SHIFT
2:      (Success, TlbEntry) = TLB_Lookup(VPN)
3:      if (Success == True) // TLB Hit
4:         if (CanAccess(TlbEntry.ProtectBits) == True)
5:               Offset = VirtualAddress & OFFSET_MASK
6:               PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7:               Register = AccessMemory(PhysAddr)
8:         else RaiseException(PROTECTION_FAULT)
```

VPN-Virtual Page Number

PFN – Physical Frame Number

Page Fault Control Flow – Hardware

```
9:        else // TLB Miss
10:         PTEAddr = PTBR + (VPN * sizeof(PTE))
11:         PTE = AccessMemory(PTEAddr)
12:         if (PTE.Valid == False)
13:               RaiseException(SEGMENTATION_FAULT)
14:         else
15:            if (CanAccess(PTE.ProtectBits) == False)
16:               RaiseException(PROTECTION_FAULT)
17:            else if (PTE.Present == True)
18:            // assuming hardware-managed TLB
19:               TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
20:               RetryInstruction()
21:            else if (PTE.Present == False)
22:               RaiseException(PAGE_FAULT)
```

VPN-Virtual Page Number
PTE – Page Table Entry

PFN – Physical Frame Number
PTBR – Page Table Base Register

Page Fault Control Flow – Software

```
1:        PFN = FindFreePhysicalPage()
2:        if (PFN == -1) // no free page found
3:                PFN = EvictPage() // run replacement algorithm
4:        DiskRead(PTE.DiskAddr, pfn) // sleep (waiting for I/O)
5:        PTE.present = True // update page table with present
6:        PTE.PFN = PFN // bit and translation (PFN)
7:        RetryInstruction() // retry instruction
```

- The OS must find a physical frame for the soon-be-faulted-in page to reside within.
- If there is no such page, waiting for the replacement algorithm to run and kick some pages out of memory.

When Replacements Really Occur

- OS waits until memory is entirely full, and only then replaces a page to make room for some other page – Unrealistic
  - OS should proactively keep a small portion of memory free more proactively.

- Swap Daemon, Page Daemon
  - There are fewer than LW pages available, a background thread that is responsible for freeing memory runs.
  - The thread evicts pages until there are HW pages available.

    (LW – Low Watermark; HW – High Watermark)

Replacement Policy (Which pages to evict from memory)

- Goal in picking a replacement policy cache is to minimize the number of misses.
- The number of hits and misses let us calculate the *average memory access time(AMAT)*.

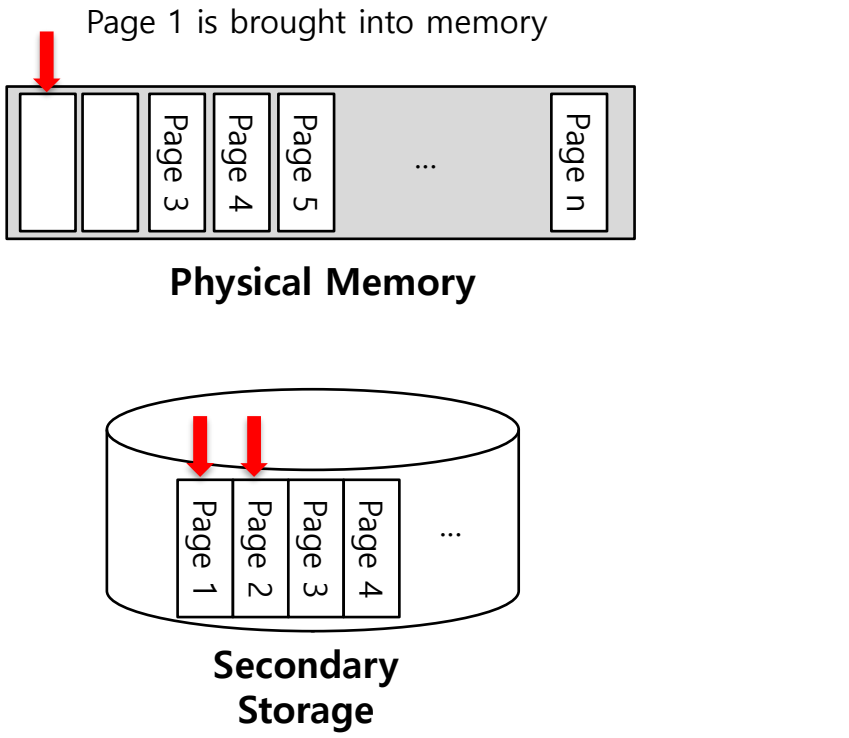
$$AMAT = T_M + (P_{Miss} * T_D)$$

Arguement	Meaning
T_M	The cost of accessing memory
T_D	The cost of accessing disk
P_{Hit}	The probability of finding the data item in the cache(a hit)
P_{Miss}	The probability of not finding the data in the cache(a miss)

- LRU (Least Recently Used) – A common approach (based on the history)
  - Replaces the least-recently-used page.

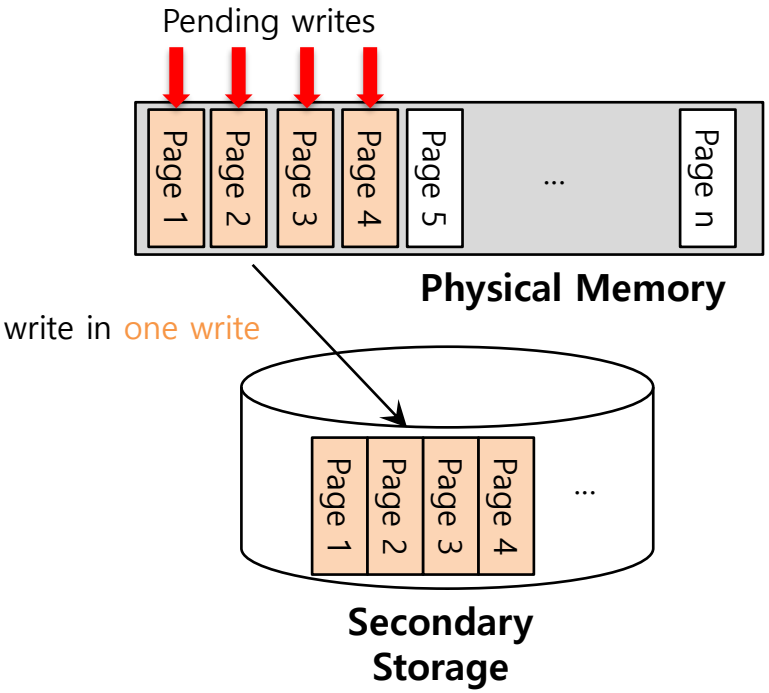
Prefetching

- The OS predicts that a page will be used, and thus brings it in beforehand.



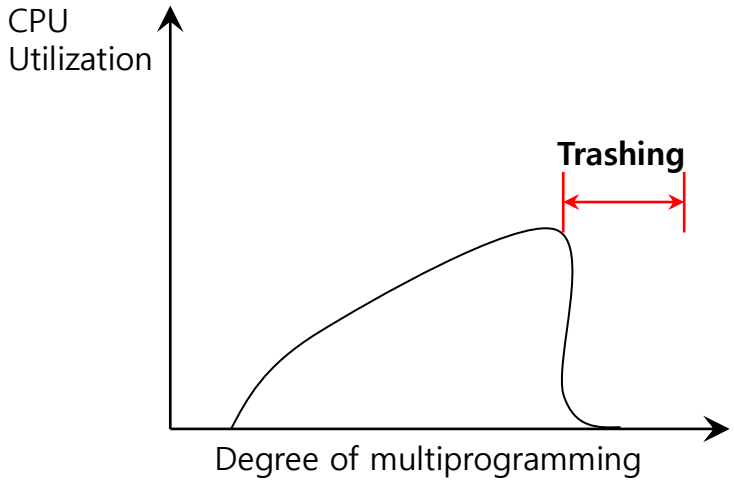
Clustering, Grouping

- Collect a number of pending writes together in memory and write them to disk in one write.
  - Perform a **single large write** more efficiently than **many small ones**.



Thrashing

- Memory is oversubscribed and the memory demand of the set of running processes exceeds the available physical memory.
  - Decide not to run a subset of processes.
  - Reduced set of processes working sets fit in memory.



Summary

- Swapping
  - Reserve some space on the disk for moving pages back and forth.
  - The OS moves out pages to make room for the new pages when there is not enough memory space
  - Page fault occurs when accessing a page not in memory but in the swap area
- Page Replacing
  - When to replace: LW/HW eviction
  - How to replace: LRU
- Next: Concurrency