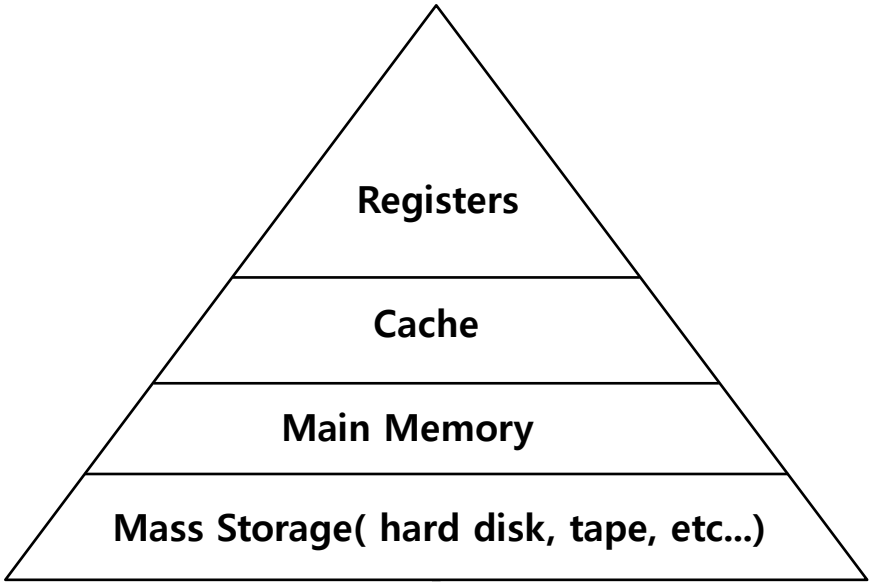
Operating Systems

Lecture 11

21. Swapping: Mechanisms

Beyond Physical Memory: Mechanisms

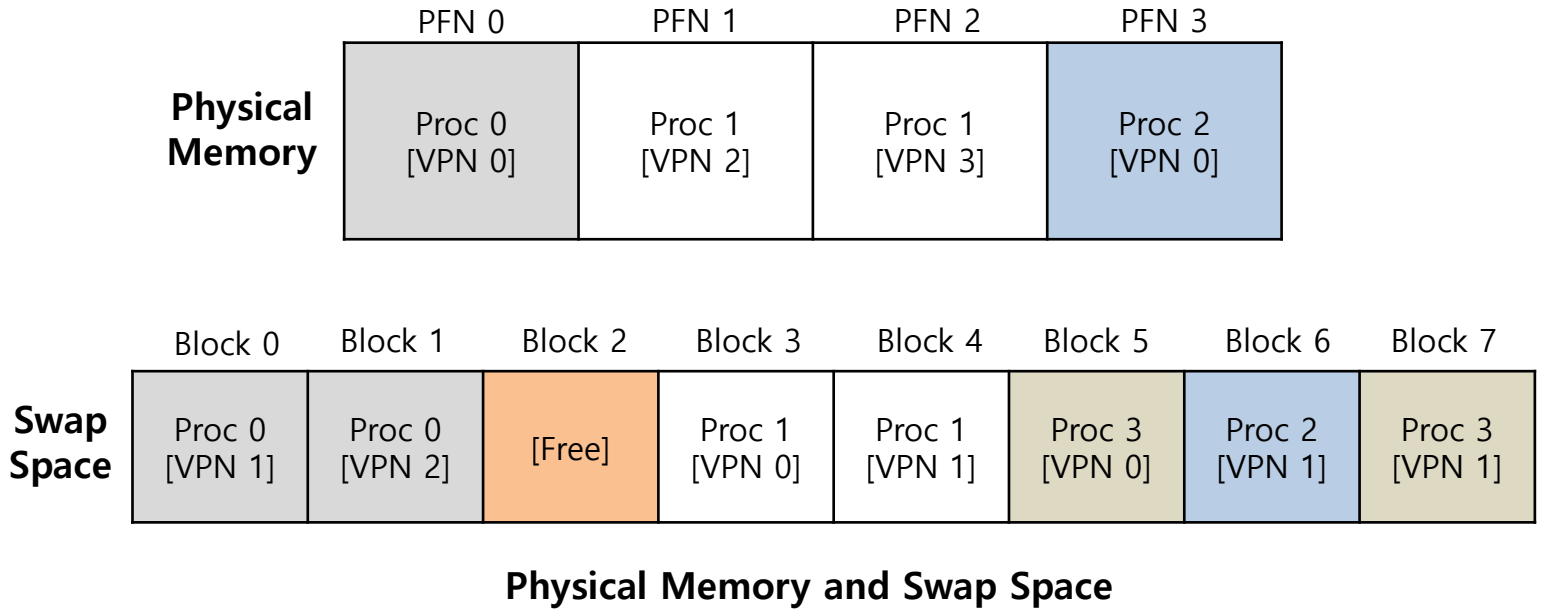
- Use part of disk as memory.
  - OS need a place to stash away portions of address space that currently aren't in great demand.
  - In modern systems, this role is usually served by a hard disk drive.



Memory Hierarchy in modern system

Swap Space

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



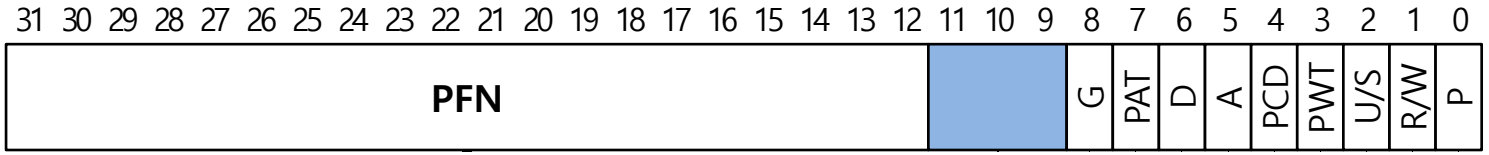
Physical Memory and Swap Space

Present Bit

- Add some machinery higher up in the system in order to support swapping the 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.

Example: x86 Page Table Entry



An x86 Page Table Entry(PTE)

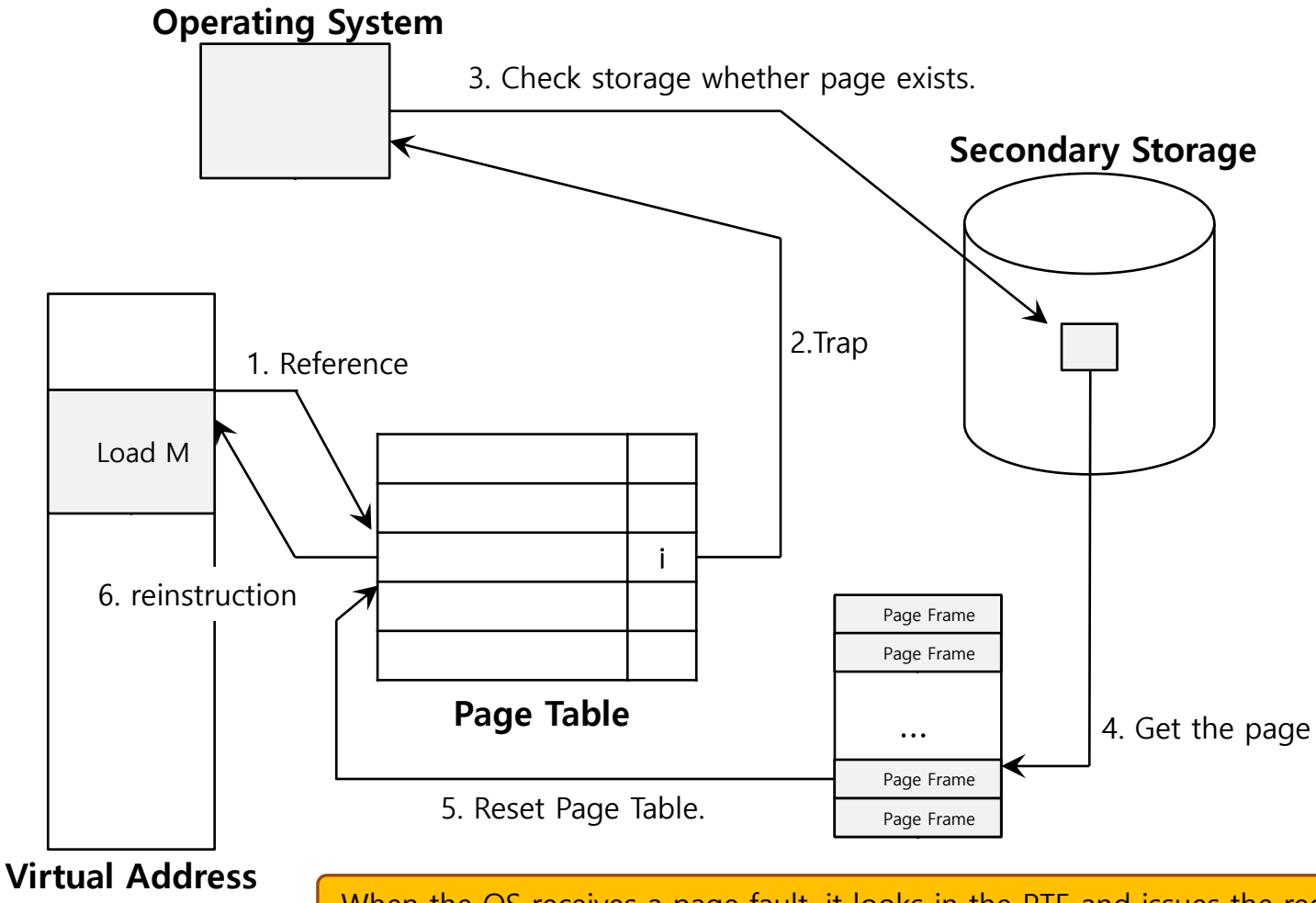
- P: present
- R/W: read/write bit
- U/S: supervisor
- A: accessed bit
- D: dirty bit
- PFN: the page frame number

concept

- 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 back into memory in order to service the page fault.
- Page replacement
  - The OS likes to page out pages to make room for the new pages the OS is about to bring in.
  - The process of picking a page to kick out, or replace is known as page-replacement policy.

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)
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)
```

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.

Summary

- Swapping: making the part of disk as memory
- Present bit required

22. Swapping: Policies

Goal of Cache Management

- to minimize the number of cache misses.
- the *average memory access time(AMAT)*.

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

Argument	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)

The Optimal Replacement Policy

- Lead to the fewest number of misses overall.
  - Replace the page that will be accessed furthest in the future.
  - Result in the fewest-possible cache misses.
- Serve only as a comparison point, to know how close we are to perfect.

Tracing the Optimal Policy

Reference Row

0 1 2 0 1 3 0 3 1 2 1

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		0,1,2
1	Hit		0,1,2
3	Miss	2	0,1,3
0	Hit		0,1,3
3	Hit		0,1,3
1	Hit		0,1,3
2	Miss	3	0,1,2
1	Hit		0,1,2

Hit rate is $\frac{Hits}{Hits+Misses} = \mathbf{54.6\%}$

Future is not known.

A Simple Policy: FIFO

- Pages were placed in a queue when they enter the system.
- When a replacement occurs, the page on the tail of the queue(the "**First-in**" pages) is evicted.
  - It is simple to implement, but can't determine the importance of blocks.

Tracing the FIFO Policy

Reference Row

0 1 2 0 1 3 0 3 1 2 1

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		0,1,2
1	Hit		0,1,2
3	Miss	0	1,2,3
0	Miss	1	2,3,0
3	Hit		2,3,0
1	Miss		3,0,1
2	Miss	3	0,1,2
1	Hit		0,1,2

Hit rate is $\frac{Hits}{Hits+Misses} = \mathbf{36.4\%}$

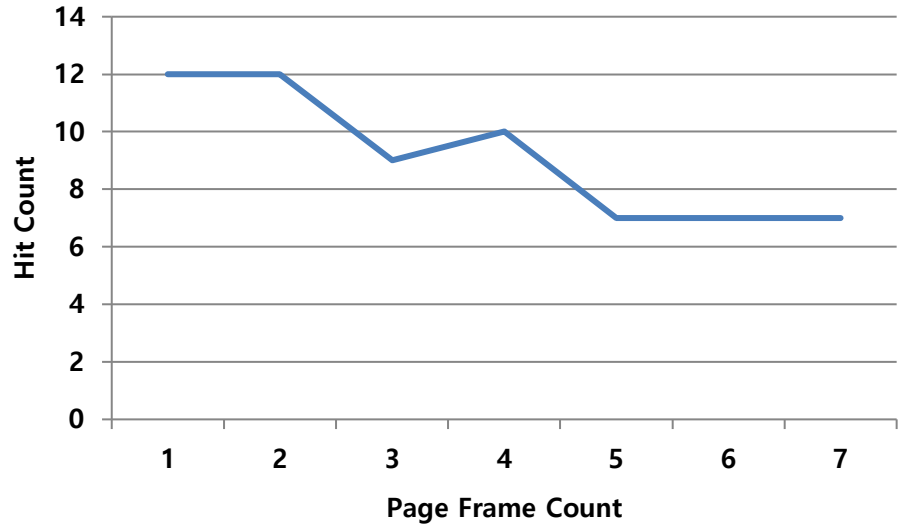
Even though page 0 had been accessed a number of times, FIFO still kicks it out.

BELADY'S ANOMALY

- We would expect the cache hit rate to increase when the cache gets larger. But in this case, with FIFO, it gets worse.

Reference Row

1 2 3 4 1 2 5 1 2 3 4 5



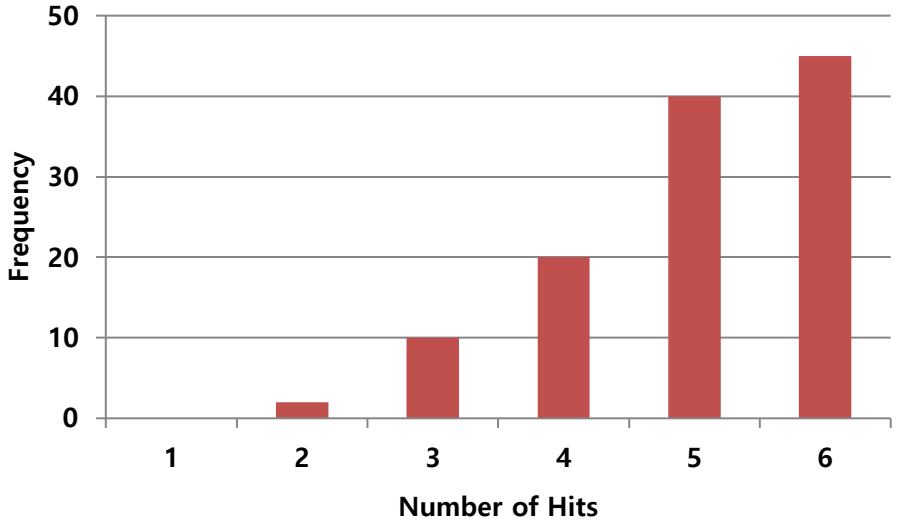
Another Simple Policy: Random

- Picks a random page to replace under memory pressure.
  - It doesn't really try to be too intelligent in picking which blocks to evict.
  - Random does depends entirely upon how lucky Random gets in its choice.

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		0,1,2
1	Hit		0,1,2
3	Miss	0	1,2,3
0	Miss	1	2,3,0
3	Hit		2,3,0
1	Miss	3	2,0,1
2	Hit		2,0,1
1	Hit		2,0,1

Random Performance

- Sometimes, Random is as good as optimal, achieving 6 hits on the example trace.



Random Performance over 10,000 Trials

Using History

- Learn on the past and use **history**.
  - Two type of historical information.

Historical Information	Meaning	Algorithms
recency	The more recently a page has been accessed, the more likely it will be accessed again	LRU
frequency	If a page has been accessed many times, It should not be replcaed as it clearly has some value	LFU

Using History : LRU

- Replace the least-recently-used page.

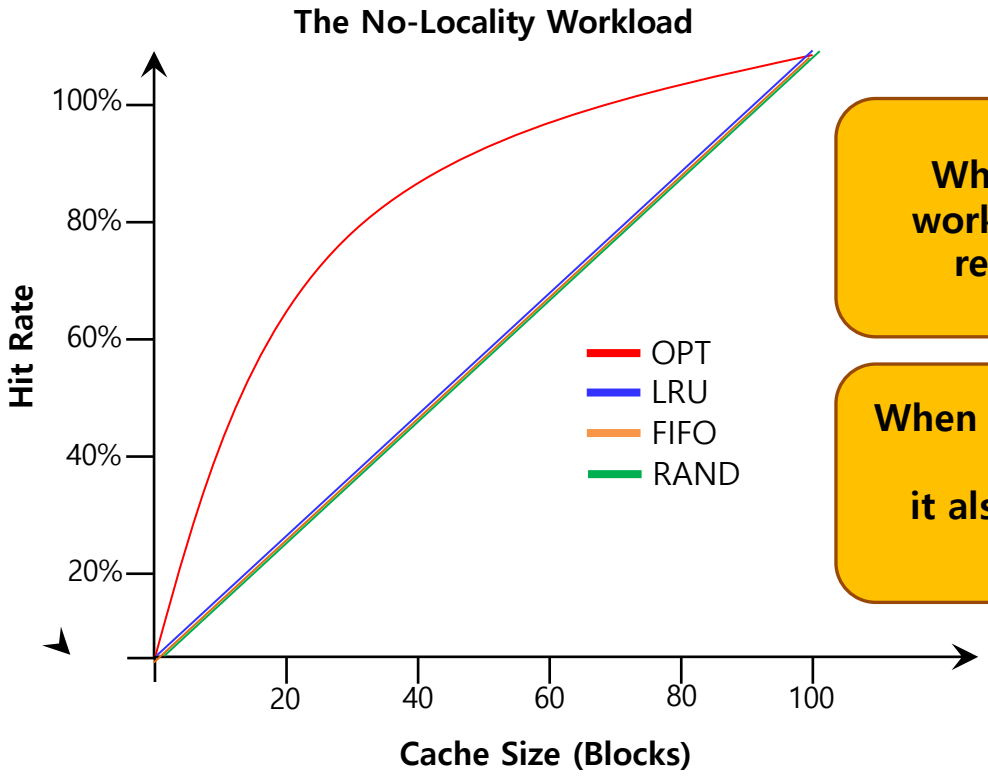
Reference Row

0 1 2 0 1 3 0 3 1 2 1

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		1,2,0
1	Hit		2,0,1
3	Miss	2	0,1,3
0	Hit		1,3,0
3	Hit		1,0,3
1	Hit		0,3,1
2	Miss	0	3,1,2
1	Hit		3,2,1

Workload Example : The No-Locality Workload

- Each reference is to a random page within the set of accessed pages.
  - Workload accesses 100 unique pages over time.
  - Choosing the next page to refer to at random

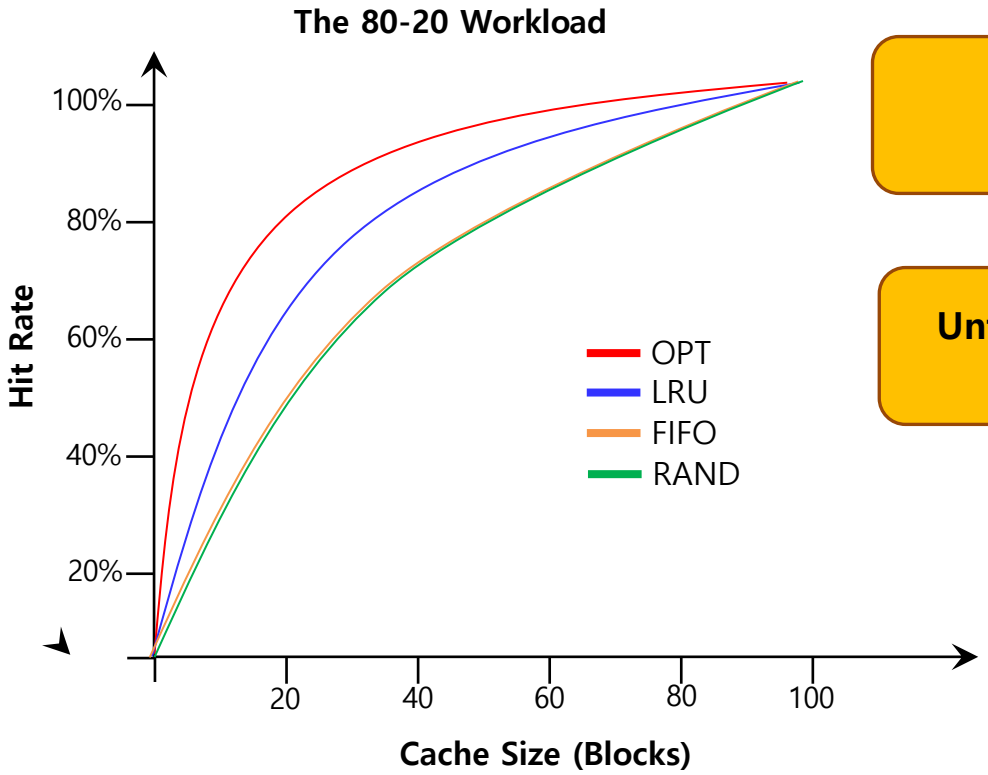


When there is no locality in the workload, it doesn't matter which realistic policy you are using.

When the cache is large enough to fit the entire workload, it also doesn't matter which policy you use.

Workload Example : The 80-20 Workload

- Exhibits locality: 80% of the reference are made to 20% of the page
- The remaining 20% of the reference are made to the remaining 80% of the pages.

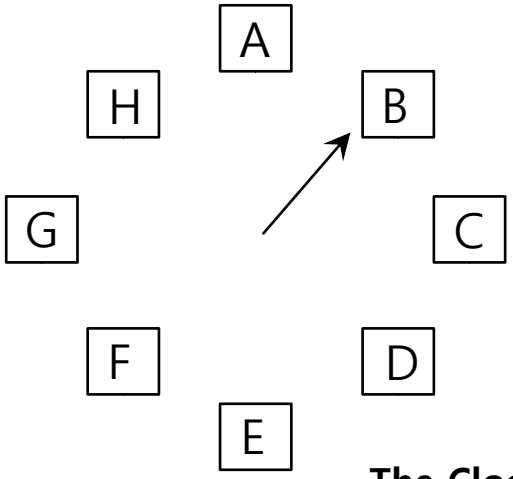


LRU is more likely to hold onto the hot pages.

Unfortunately, LRU algorithm is extremely expensive.

Approximating LRU: Clock Algorithm

- Require hardware support: a **use bit**
  - Whenever a page is referenced, the use bit is set by hardware to 1.
  - Hardware never clears the bit, though; that is the responsibility of the OS
- Clock Algorithm
  - All pages of the system arranges in a circular list.
  - A clock hand points to some particular page to begin with.
  - The algorithm continues until it finds a use bit that is set to 0.

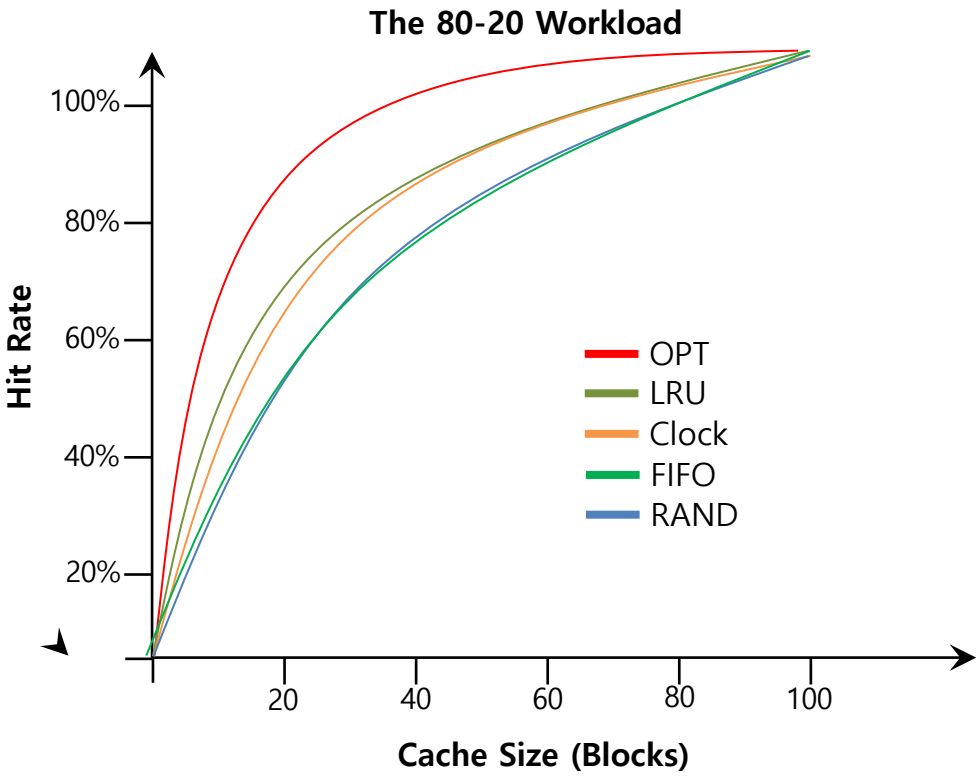


Use bit	Meaning
0	Evict the page
1	Clear **Use bit** and advance hand

The Clock page replacement algorithm

Workload with Clock Algorithm

- Clock algorithm doesn't do as well as perfect LRU, it does better then approach that don't consider history at all.

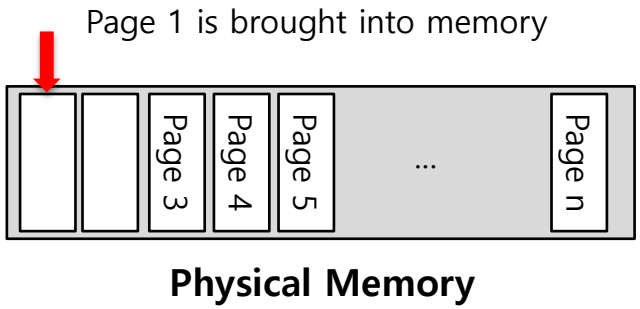


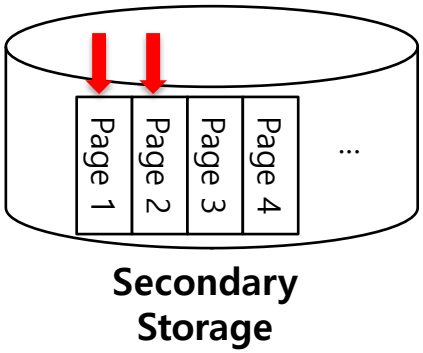
Considering Dirty Pages

- The hardware includes a **modified bit** (a.k.a **dirty bit**)
  - Page has been **modified** and is thus **dirty**, it must be written back to disk to evict it.
  - Page has not been modified, the eviction is free.

Prefetching

- The OS guesses that a page is about to be used, and thus bring it in ahead of time.

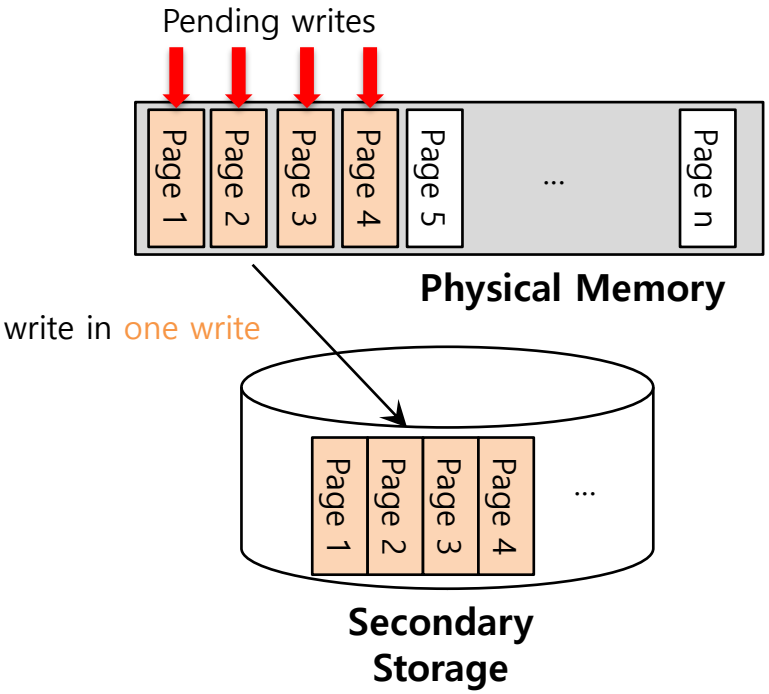




Page 2 likely soon be accessed and
thus should be brought into memory too

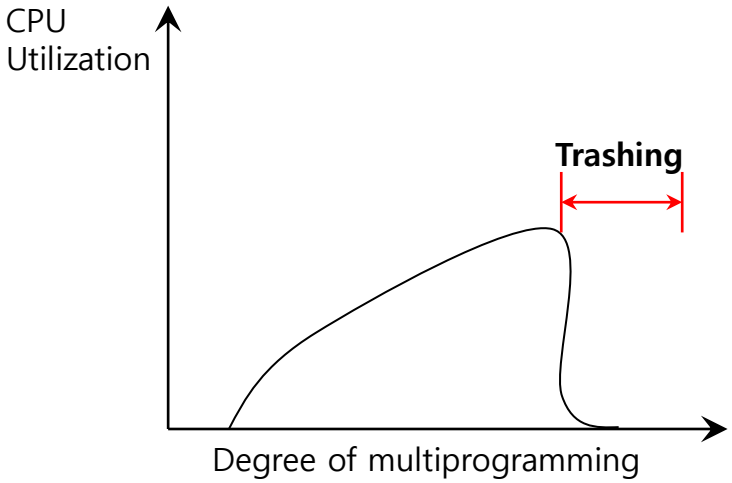
Clustering, Grouping

- 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 demands 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

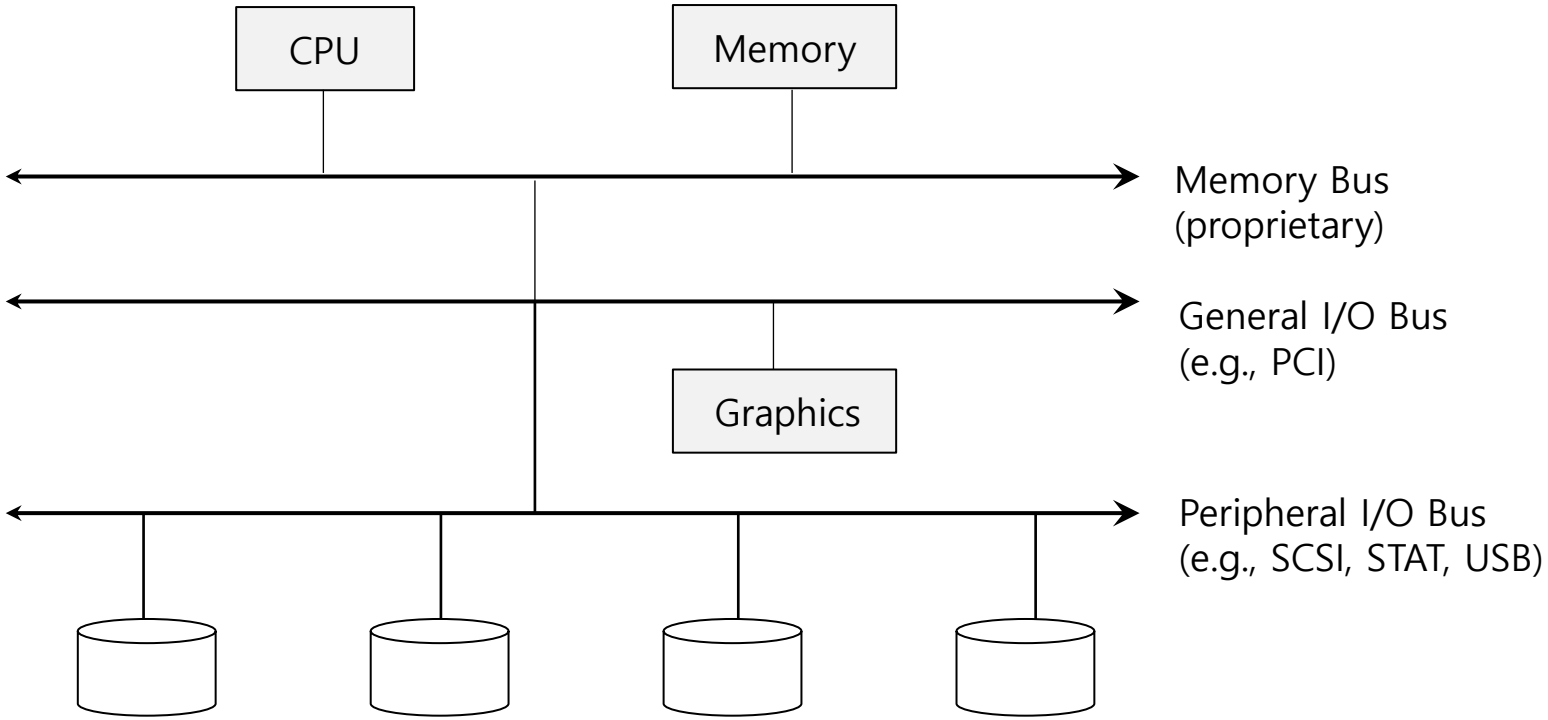
- Swapping: use part of disk as memory
- LRU, LFU, RANDOM, FIFO
- Approximation to LRU: Clock
- Making the disk IO in larger unit
  - Clustering
  - Grouping
  - prefetching

36. I/O Devices

I/O Devices

- I/O is **critical** to computer system to **interact with other systems.**
- Issue :
  - How should I/O be integrated into systems?
  - What are the general mechanisms?
  - How can we make the efficiently?

Structure of input/output (I/O) device

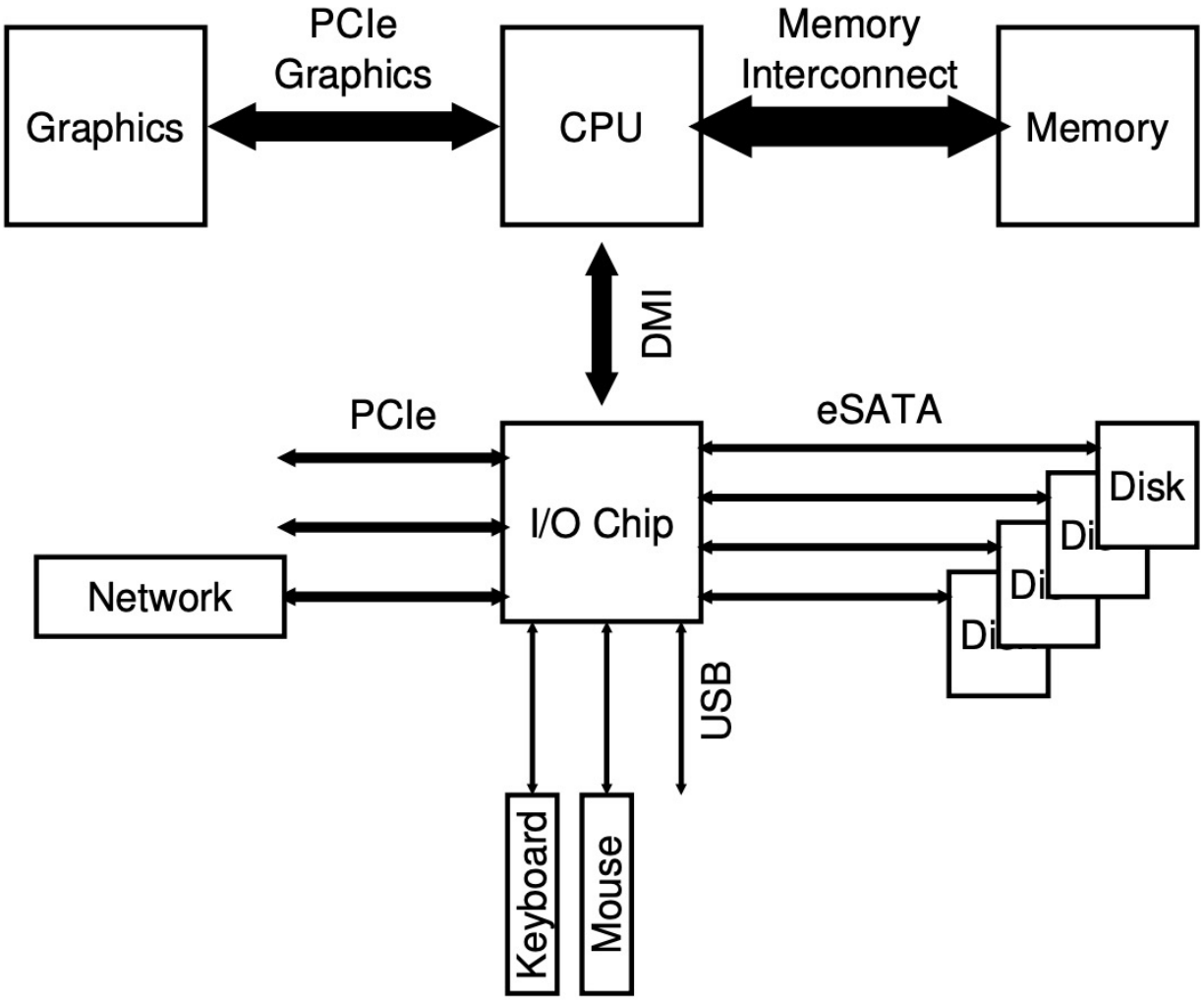


Prototypical System Architecture

CPU is attached to the main memory of the system via some kind of memory bus.

Some devices are connected to the system via a general I/O bus.

Modern System Architecture



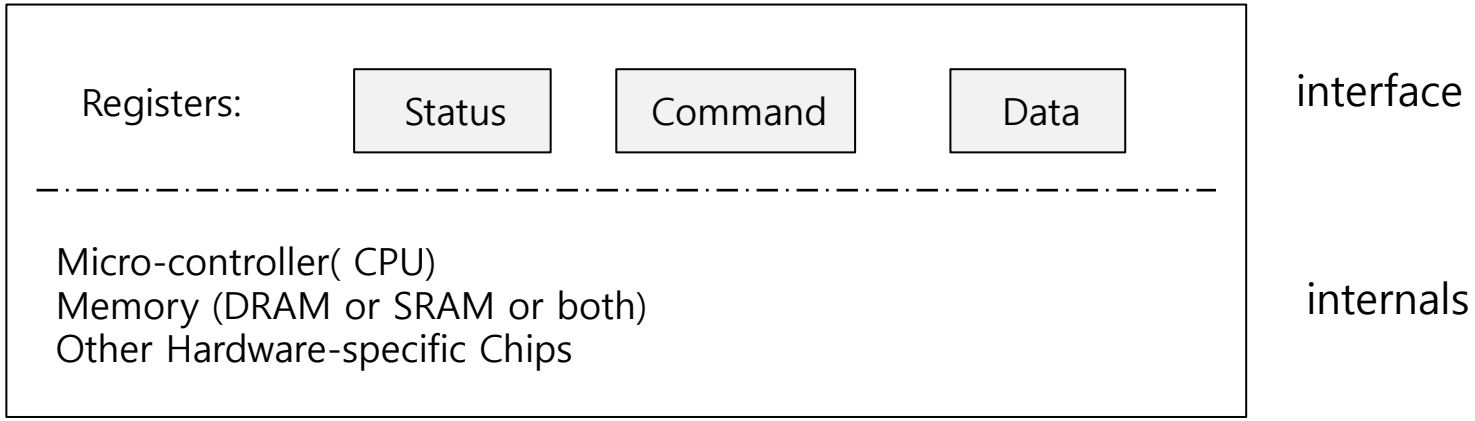
I/O Architecture

- Buses
  - Data paths that provided to enable information between CPU(s), RAM, and I/O devices.

- I/O bus
  - Data path that connects a CPU to an I/O device.
  - I/O bus is connected to I/O device by hardware components: I/O ports, interfaces and device controllers.

Canonical Device

- Canonical Devices has two important components.
  - **Hardware interface** allows the system software to control its operation.
  - **Internals** which is implementation specific.



Canonical Device

Hardware interface of Canonical Device

- **status register**
  - See the current status of the device
- **command register**
  - Tell the device to perform a certain task
- **data register**
  - Pass data to the device, or get data from the device

By reading and writing above three registers,
the operating system can control device behavior.

Hardware interface of Canonical Device (Cont.)

- Typical interaction example

```
while ( STATUS == BUSY)
    ; //wait until device is not busy
write data to data register
write command to command register
    Doing so starts the device and executes the command
while ( STATUS == BUSY)
    ; //wait until device is done with your request
```

Polling

- Operating system waits until the device is ready by **repeatedly** reading the status register.
  - Positive aspect is simple and working.
  - **However, it wastes CPU time just waiting for the device**.
    - Switching to another ready process is better utilizing the CPU.

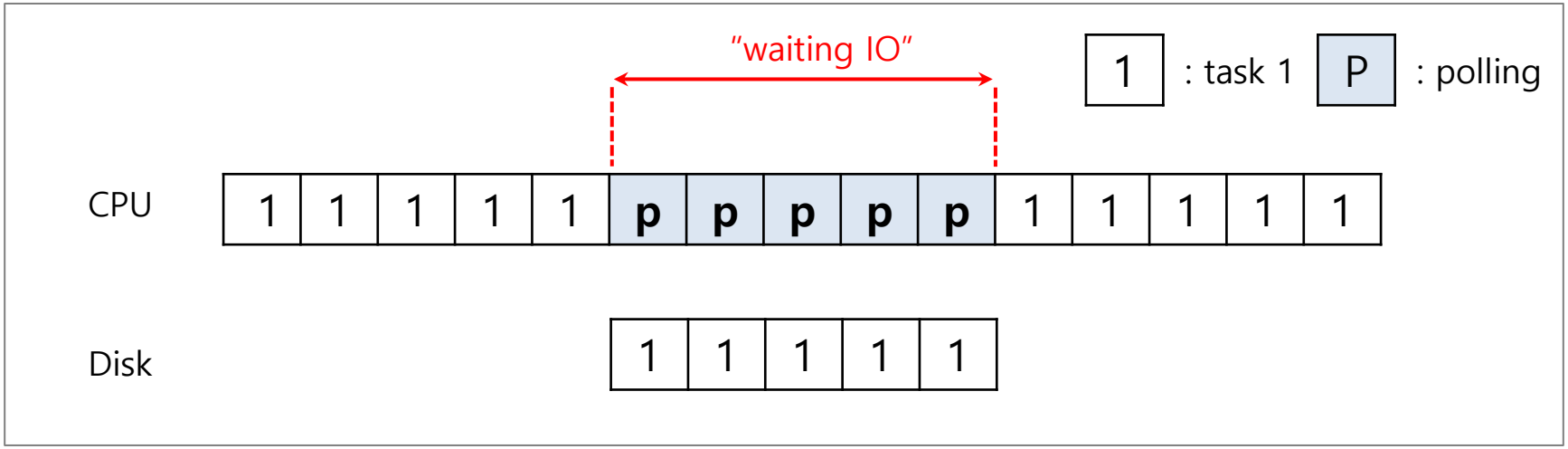


Diagram of CPU utilization by polling

Interrupts

- **Put the I/O request process to sleep** and context switch to another.
- When the device is finished, wake the process waiting for the I/O by **interrupt**.
  - Positive aspect is allow to **CPU and the disk are properly utilized.**

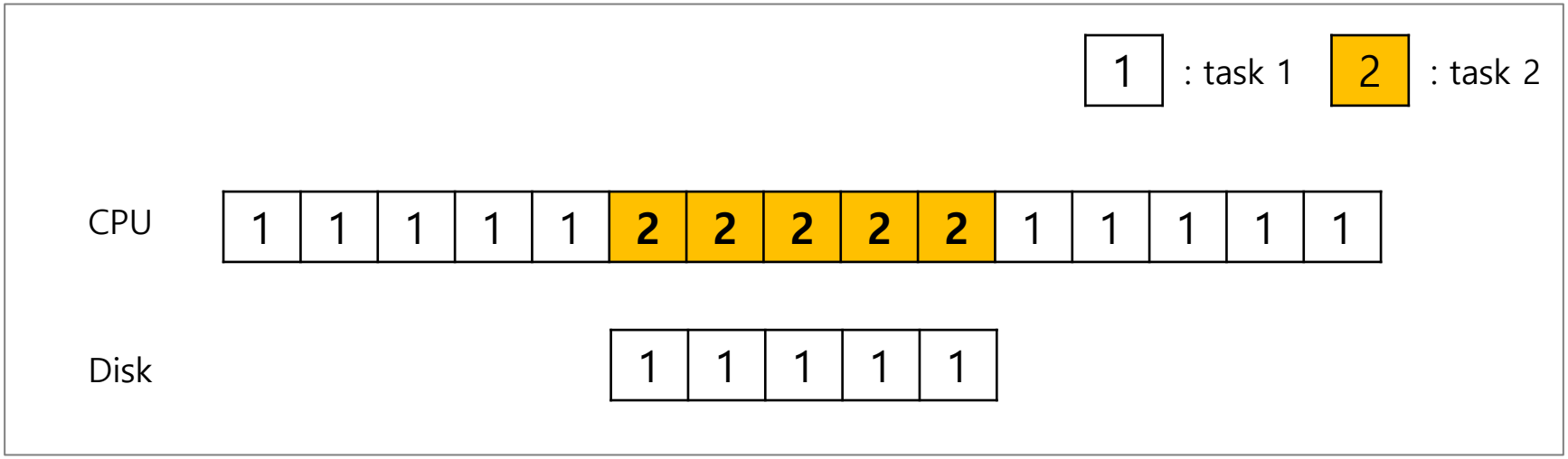


Diagram of CPU utilization by interrupt

Polling vs interrupts

- *However,* **"interrupts is not always the best solution"**
  - If, device performs very quickly, interrupt will "slow down" the system.
  - Because **context switch is expensive (switching to another process)**

If a device is fast → poll is best.
If it is slow → interrupts is better.

CPU is once again over-burdened

- CPU **wastes a lot of time** to copy the *a large chunk of data* from memory to the device.

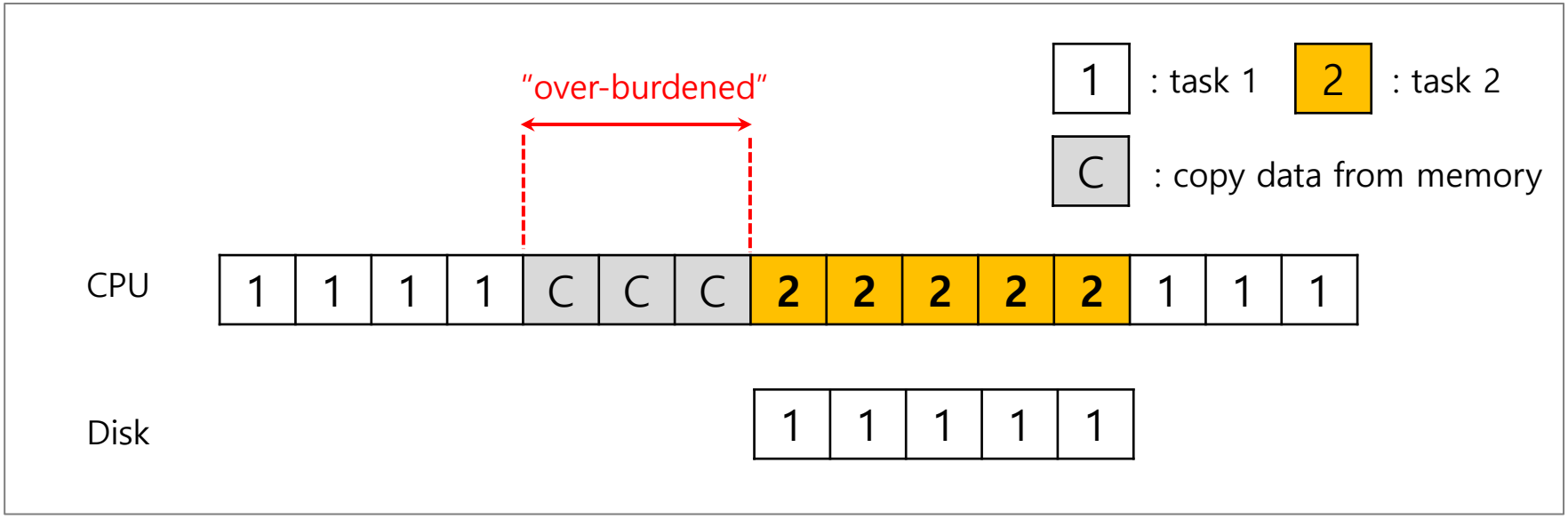


Diagram of CPU utilization

DMA (Direct Memory Access)

- **Copy data** in memory by knowing “where the data lives in memory, how much data to copy”
- When completed, DMA raises an interrupt, I/O begins on Disk.

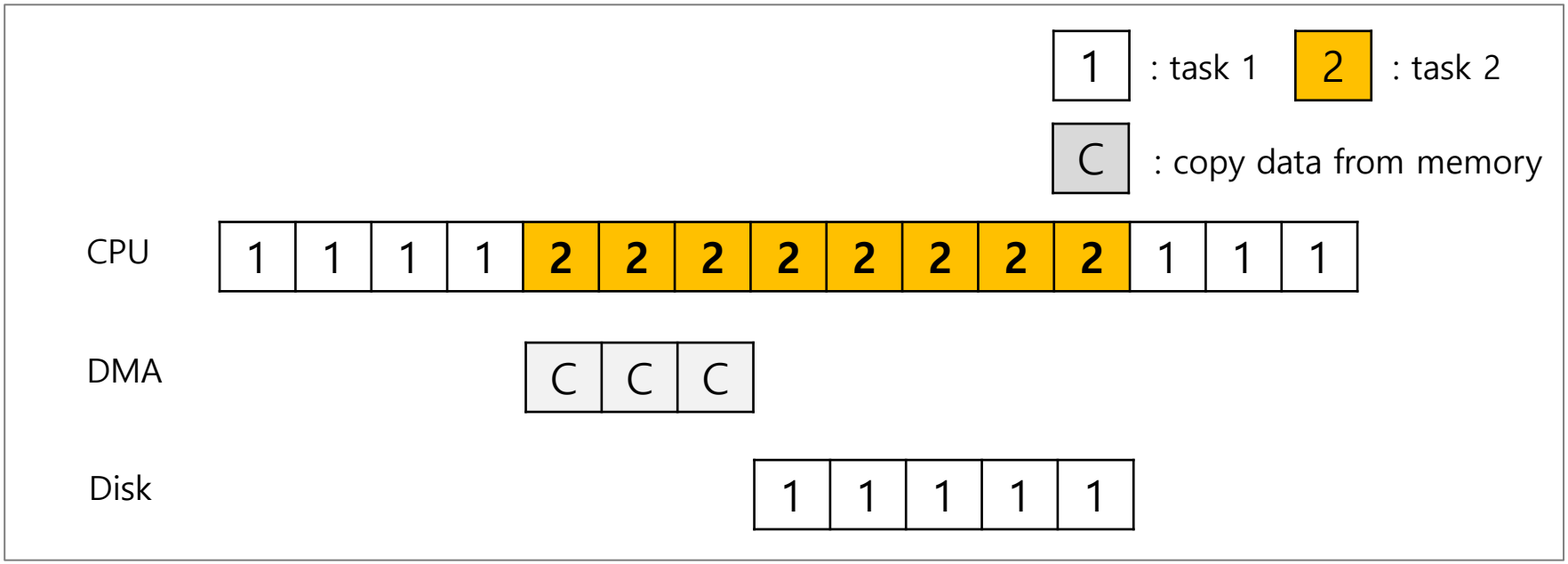


Diagram of CPU utilization by DMA

Device interaction

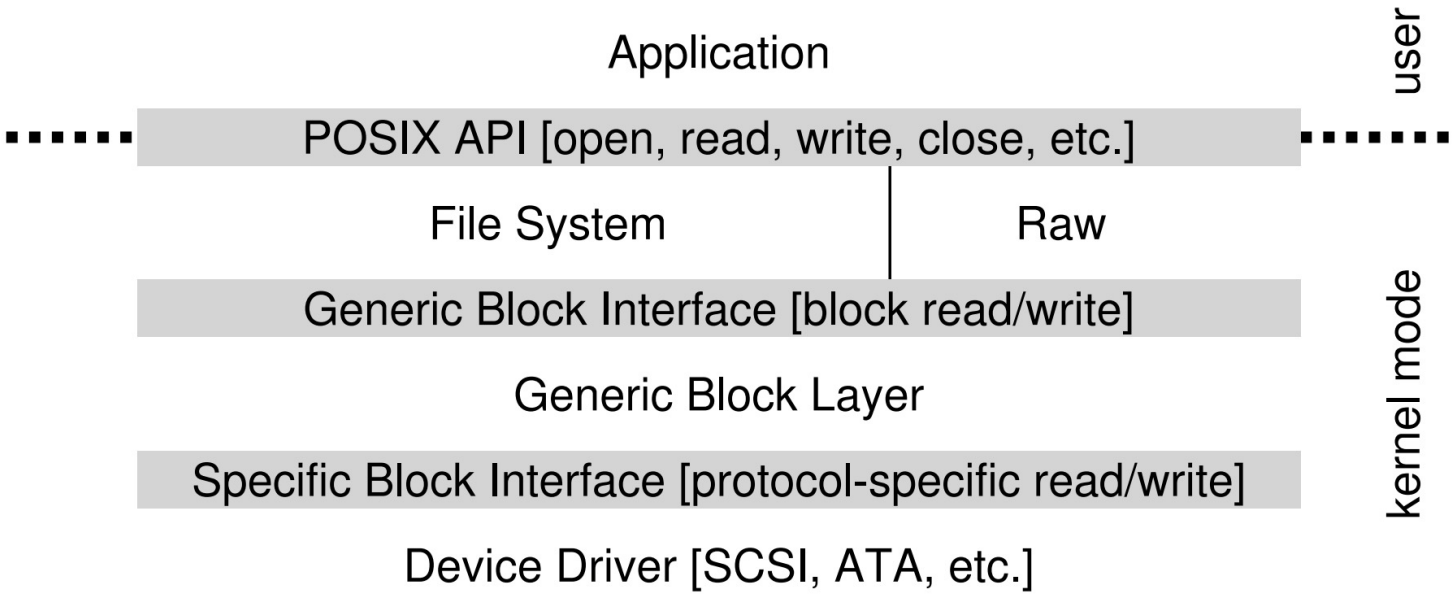
- How the OS communicates with the **device**?
- Solutions
  - I/O instructions: a way for the OS to send data to specific device registers.
    - Ex) `in` and `out` instructions on x86
  - memory-mapped I/O
    - Device registers available as if they were memory locations.
    - The OS `load` (to read) or `store` (to write) to the device instead of main memory.

Device interaction (Cont.)

- How the OS interact with **different types of interfaces**?
  - Ex) We'd like to build a file system that worked on top of SCSI disks, IDE disks, USB keychain drivers, and so on.
- Solutions: Abstraction
  - Abstraction encapsulate any specifics of device interaction.

Device driver

- A piece of software in the OS must know in detail how a device works.
- This piece of software is called **a device driver**, in which and any specifics of device interaction are encapsulated.



Linux File system stack

Summary

- To save the CPU cycles for IO
  - Use Interrupt
  - Use DMA
- To access the device registers
  - Memory mapped IO
  - Explicit IO instruction