Operating Systems
CSCI 3150

Lecture 12: Memory Management
Part III: Swapping
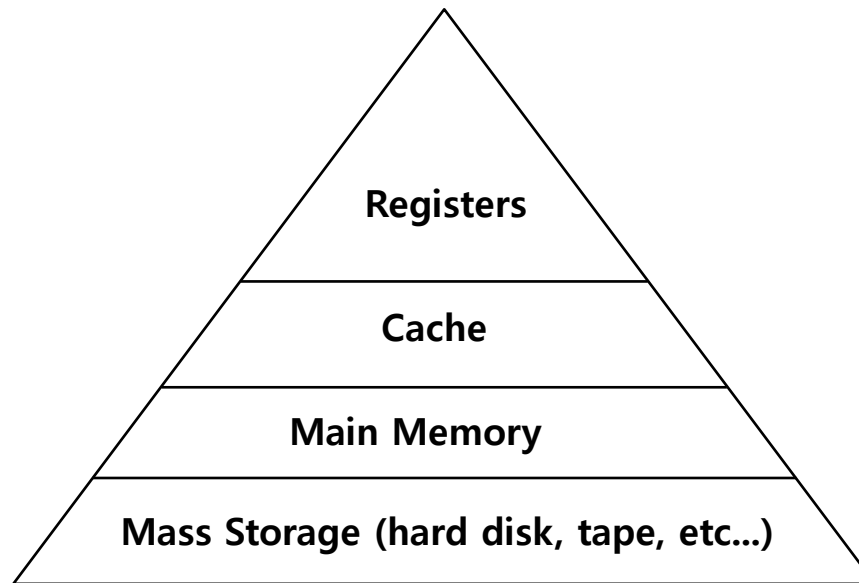
Hong Xu

https://github.com/henryhxu/CSCI3150

Overview

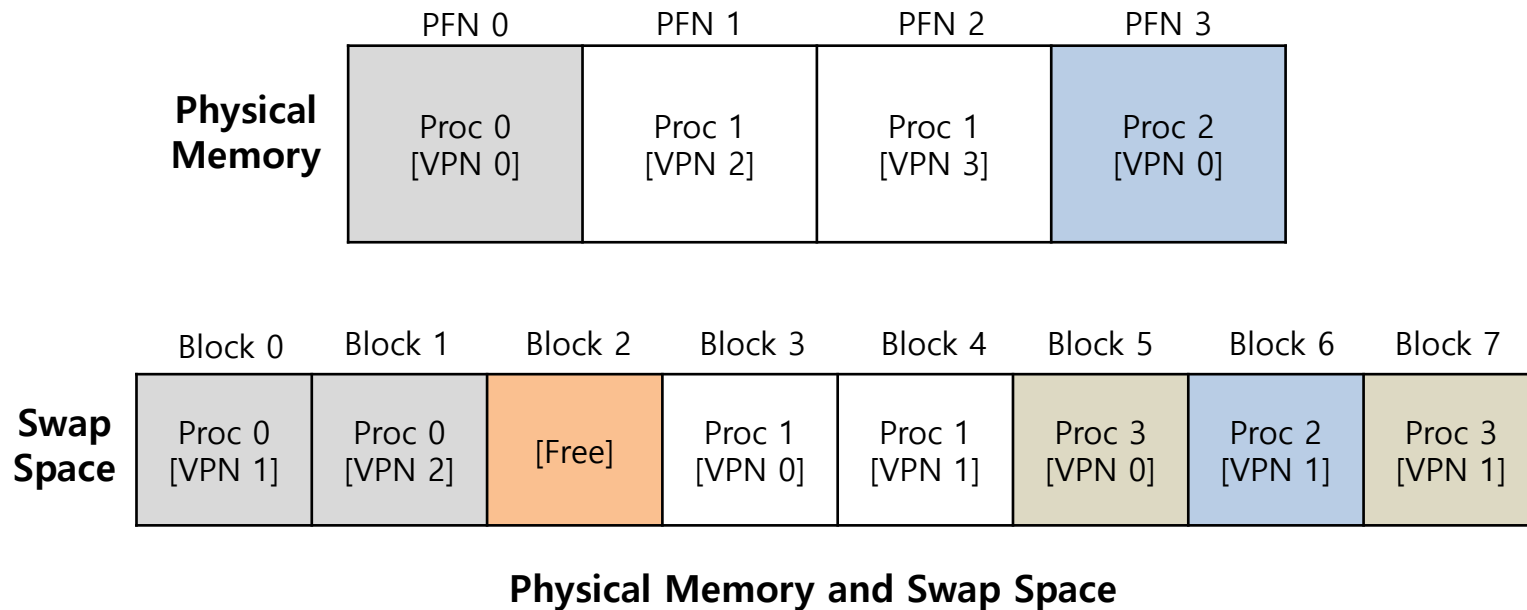- Mechanisms

- Policies

Beyond Physical Memory: Mechanisms

□ Use part of disk as memory

- ◆ 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.

```
                        /\
                       /  \
                      / Registers \
                     /------------\
                    /    Cache     \
                   /----------------\
                  /   Main Memory    \
                 /--------------------\
                / Mass Storage (hard   \
               /  disk, tape, etc...)   \
              /--------------------------\
```

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.

	PFN 0	PFN 1	PFN 2	PFN 3
Physical Memory	Proc 0 [VPN 0]	Proc 1 [VPN 2]	Proc 1 [VPN 3]	Proc 2 [VPN 0]

	Block 0	Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	Block 7
Swap Space	Proc 0 [VPN 1]	Proc 0 [VPN 2]	[Free]	Proc 1 [VPN 0]	Proc 1 [VPN 1]	Proc 3 [VPN 0]	Proc 2 [VPN 1]	Proc 3 [VPN 1]

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 <u>present</u> in physical memory.

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

Concepts

□ Page fault

- ◆ Accessing page that is not in physical memory.

- ◆ If a page is not present and has been swapped to disk, the OS needs to swap the page back into memory in order to service the page fault.
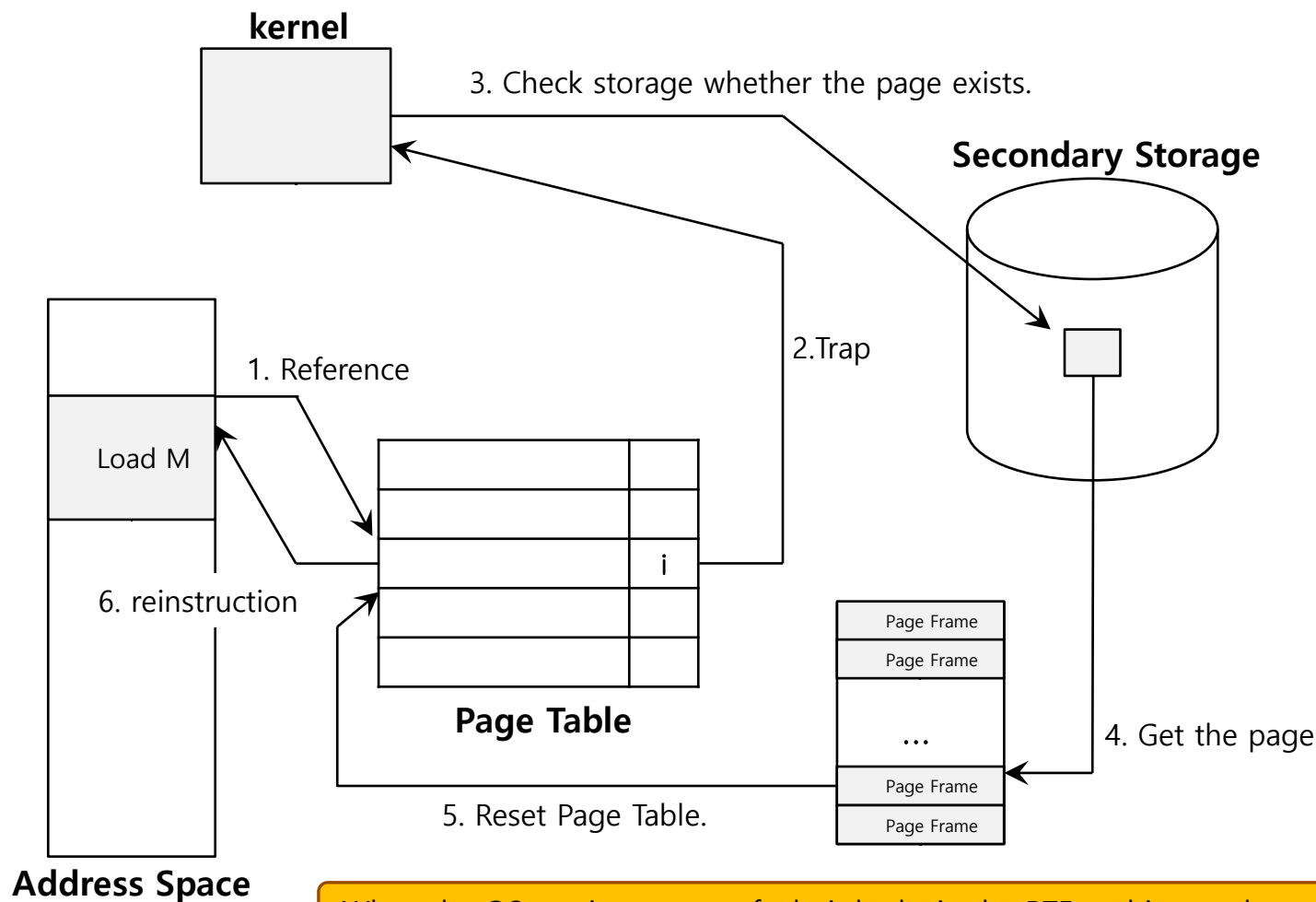
□ Page replacement

- ◆ The OS likes to page out some pages to make room for the new ones it is about to bring in

- ◆ The process of picking a page to evict (or replace) is known as page-replacement policy.

When to Perform Page Replacement

- Lazy approach...

  - OS waits until memory is full to start replacing pages.

  - This is clearly unrealistic... Do not procrastinate!

- Swap Daemon, Page Daemon

  - When there are fewer than LW (low watermark) pages available, a background thread that is responsible for freeing memory runs.

  - The thread evicts pages until there are HW (high watermark) pages available.

Page Fault Control Flow

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

kernel

3. Check storage whether the page exists.

Secondary Storage

2. Trap

1. Reference

Load M

6. reinstruction

i

Page Table

Page Frame

Page Frame

...

4. Get the page

Page Frame

5. Reset Page Table.

Page Frame

Address Space

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

```
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

Summary

- Swapping: making the part of disk as memory

- Present bit required

22. Swapping: Policies

Goal of Cache Management

- To minimize cache misses

- To improve *average memory access time (AMAT)*

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

Notation	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 <u>furthest in the future.</u>

  - Result in the fewest possible cache misses.

- Serve only as a comparison point, to know how close we are to

  perfection.

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 $\dfrac{Hits}{Hits+Misses} = \mathbf{54.6\%}$

Future is unknown!

A Simple Policy: FIFO

❑ Pages were placed in a queue when they enter the system.

❑ When a replacement occurs, the page on the head of the queue (the "**first-in**" page) is evicted.

- ◆ Simple to implement

- ◆ Agnostic to the importance of pages

Tracing the FIFO Policy

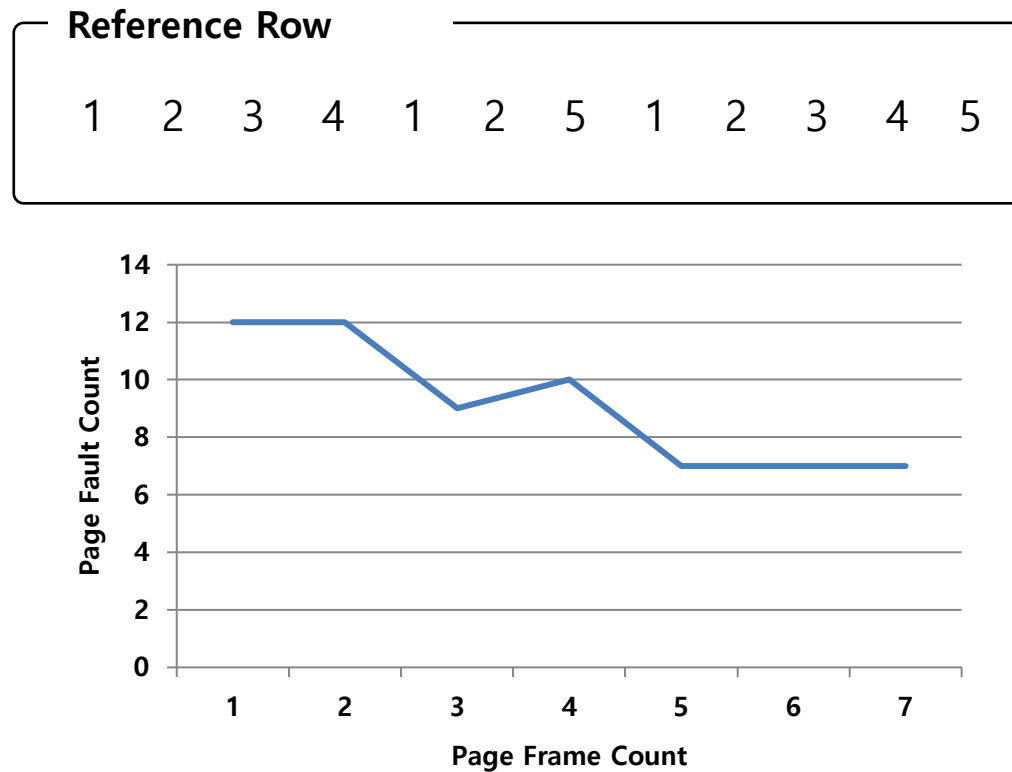**Reference Row**

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 page to evict

  ◆ Random does depends entirely upon how lucky <u>Random</u> 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

Using History

- Learn on the past and use **history**.

  - Two types of historical information.

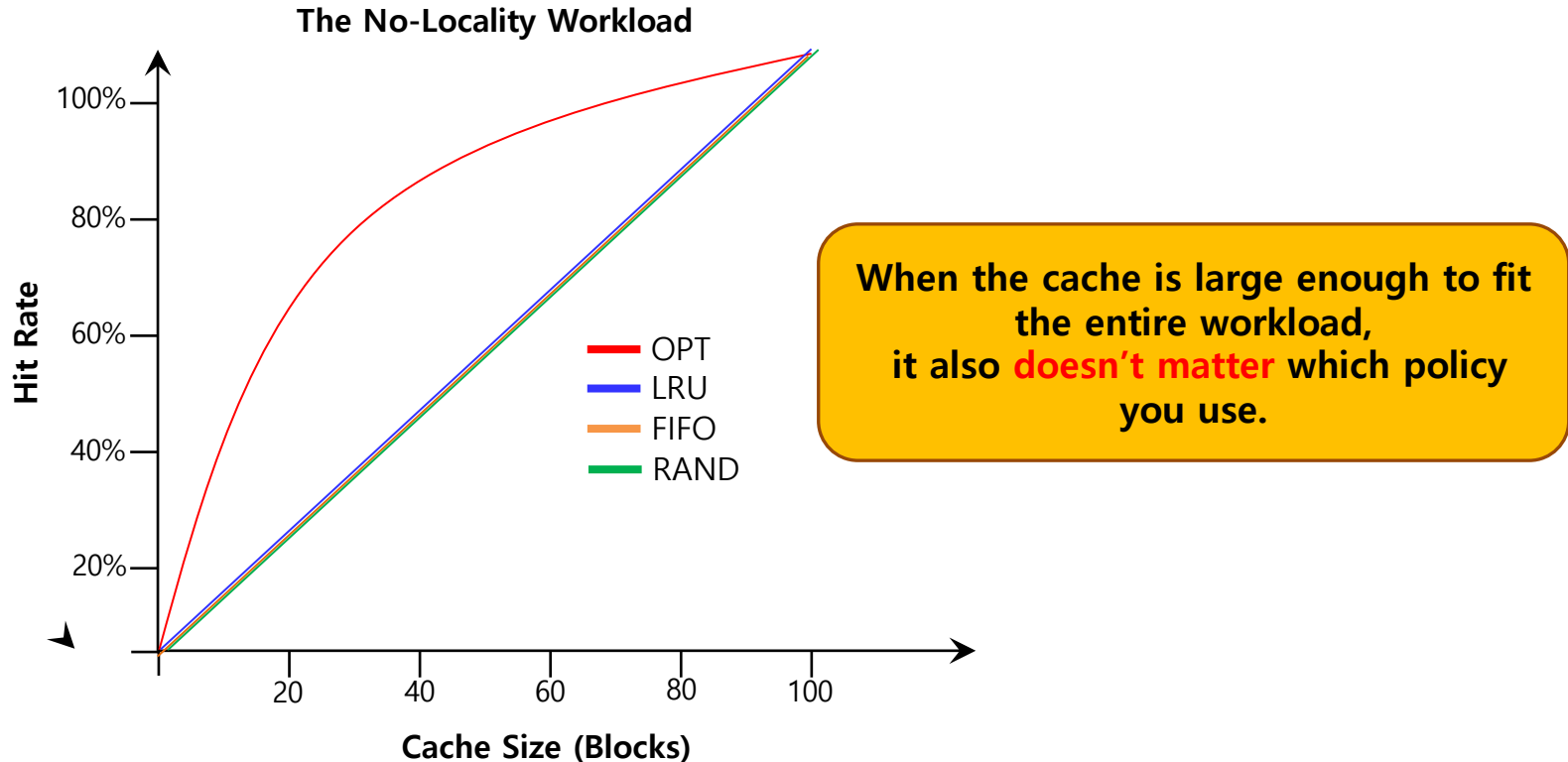| Historical Information	Meaning	Algorithms
Recency	The more recently a page has been accessed, the more likely it will be accessed again	LRU
Frequency	If a page has been accessed many times, It should not be replaced as it clearly has some value	LFU

◻ 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

- Each reference is to a random page in a set of 100 pages

  - Workload has 100 accesses over time.

  - Choosing the next page to refer to at random

The No-Locality Workload

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.

The 80-20 Workload

LRU is more likely to
hold onto the **hot pages**.

Hit Rate

OPT
LRU
FIFO
RAND

20 40 60 80 100

Cache Size (Blocks)
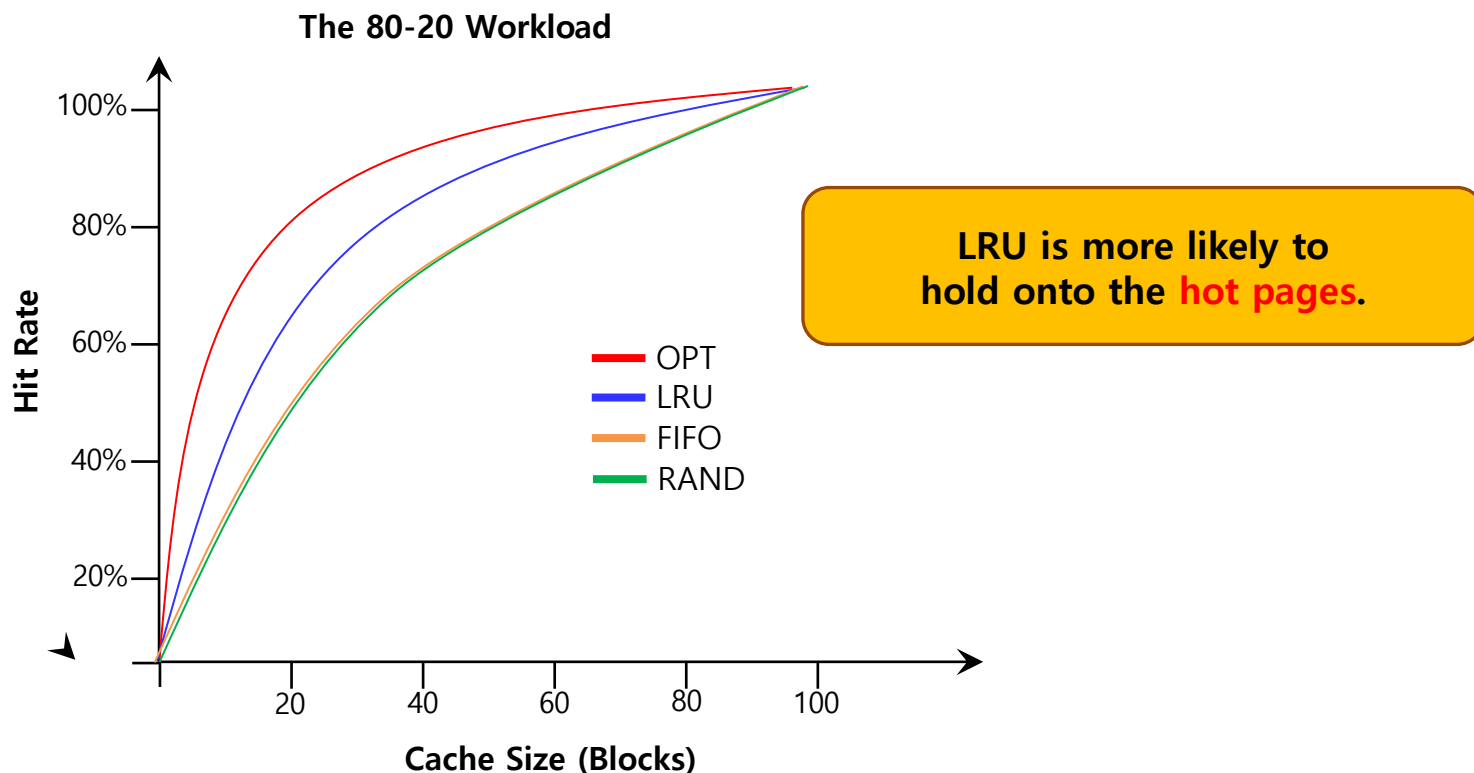
- Refer to 50 pages in sequence.

  - Starting at 0, then 1, ... up to page 49, and then we loop, repeating those accesses, for total of 10,000 accesses to 50 unique pages.

The Looping-Sequential Workload
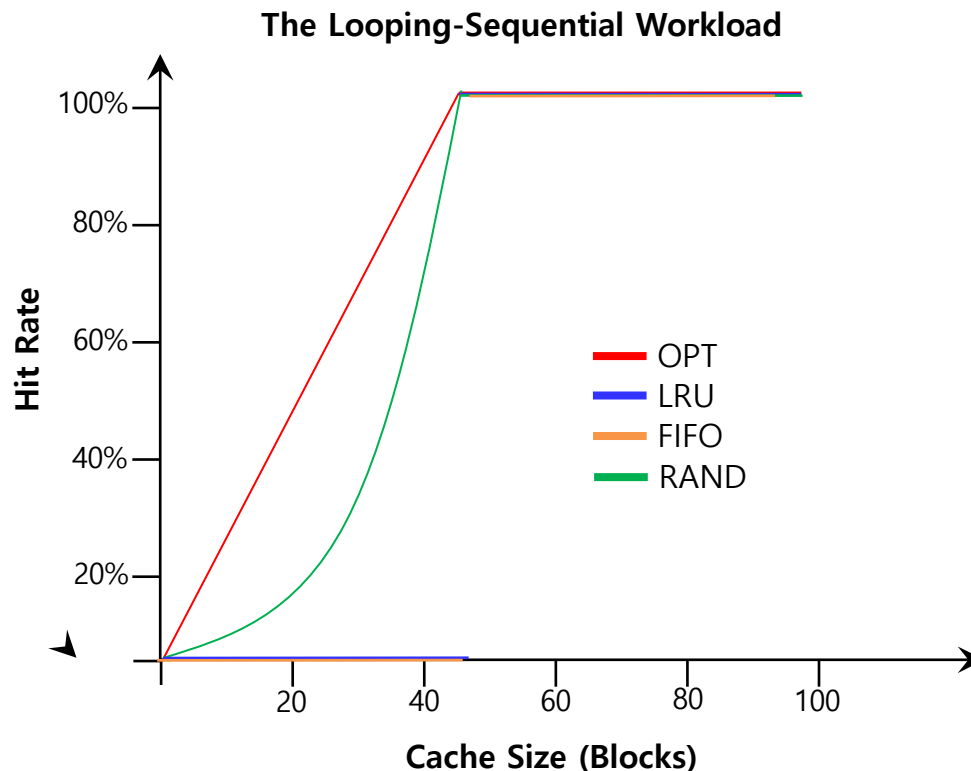
Approximating LRU: Clock Algorithm

□ 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.

A
H   B

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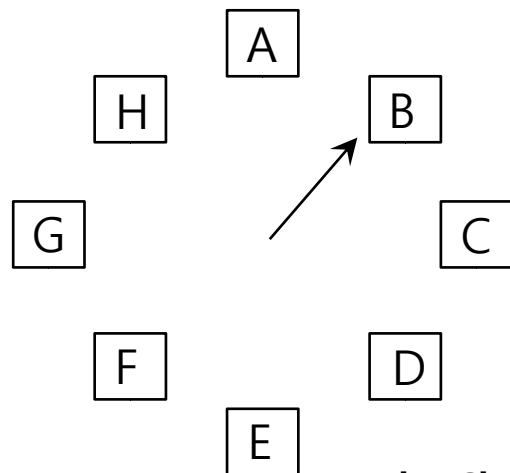
Use bit	Meaning
0	Evict the page
1	Clear **Use bit** and advance hand

F   D
E

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.
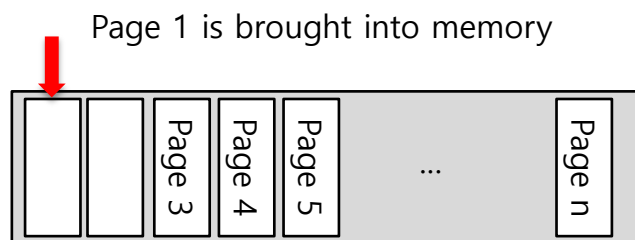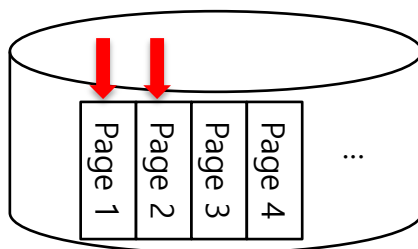

The 80-20 Workload
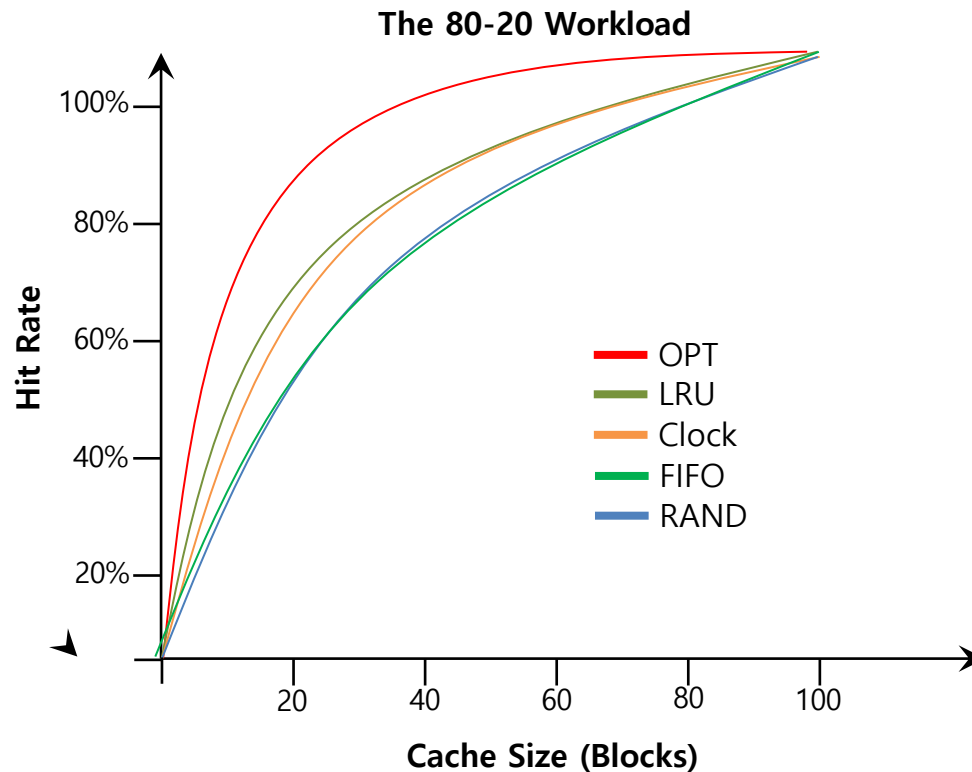
Considering Dirty Pages

- 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.

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

Page 1 is brought into memory

Page 3 | Page 4 | Page 5 | ... | Page n

Physical Memory

Page 1 | Page 2 | Page 3 | Page 4 | ...

Secondary Storage

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

- 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**.

Pending writes

Physical Memory

write in one write

Secondary
Storage
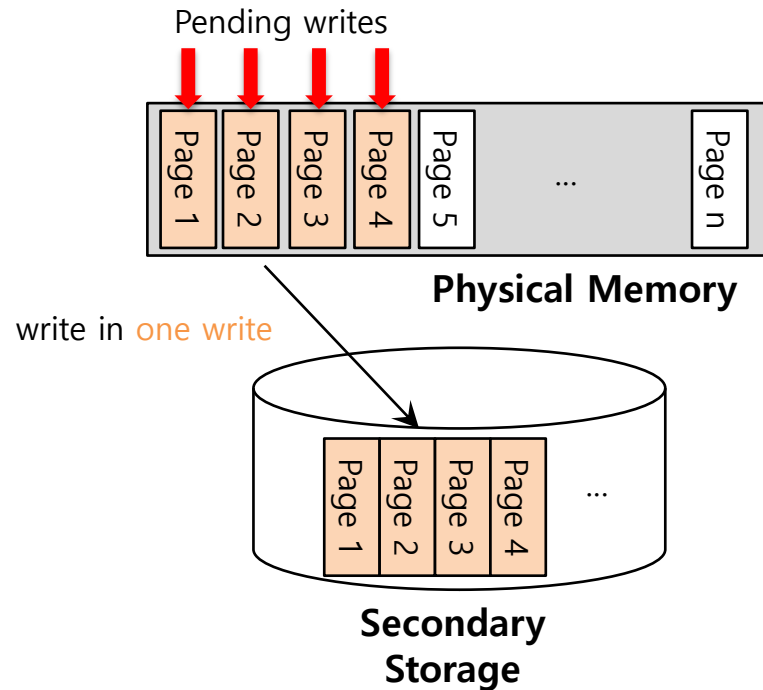
Thrashing

❑ 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.
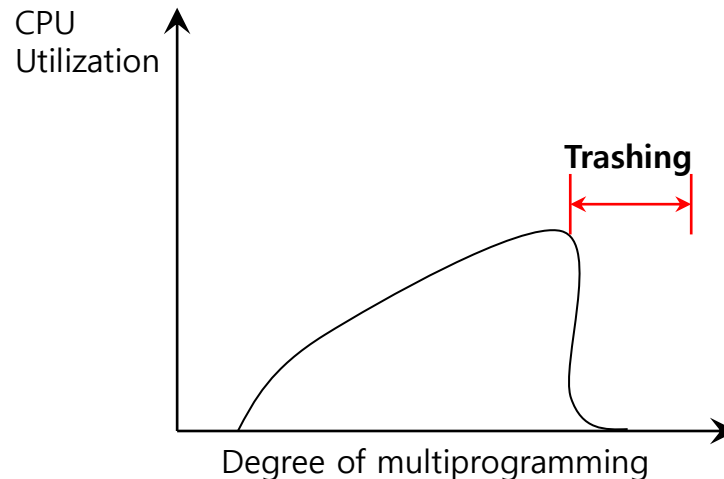
Summary

- 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