

# 22. Swaping: Policies

Operating System: Three Easy Pieces

---

# Beyond Physical Memory: Policies

- ▣ Memory pressure forces the OS to start **paging out** pages to make room for actively-used pages.
- ▣ Deciding which page to evict is encapsulated within the replacement policy of the OS.

# Cache Management

- Goal in picking a replacement policy for this cache is to minimize the number of cache misses.
- The number of cache hits and misses let us calculate 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

- ▣ Leads to the fewest number of misses overall
  - ◆ Replaces the page that will be accessed furthest in the future
  - ◆ Resulting 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  $\frac{\text{Hits}}{\text{Hits} + \text{Misses}} = 54.6\%$

**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  $\frac{\text{Hits}}{\text{Hits} + \text{Misses}} = 36.4\%$

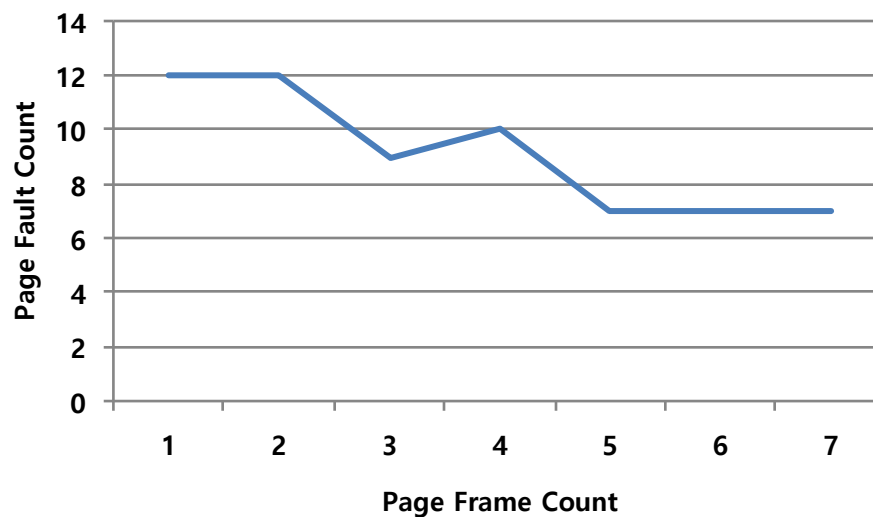
Even though page 0 had been accessed a number of times, **FIFO still kicks it out.**

# 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





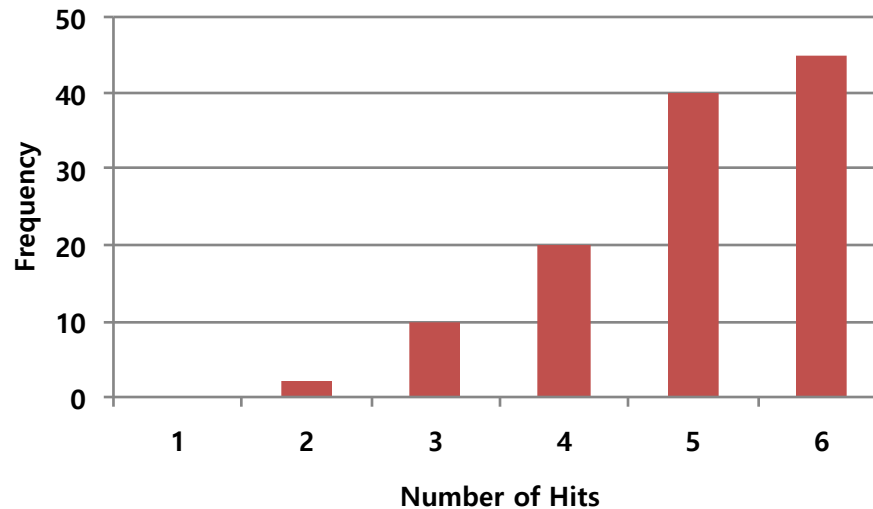
# Another Simple Policy: Random

- ▣ 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

- ▣ Lean on the past and use **history**.
  - ◆ Two type of historical information.

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

# Using History : LRU

- Replaces the least-recently-used page.

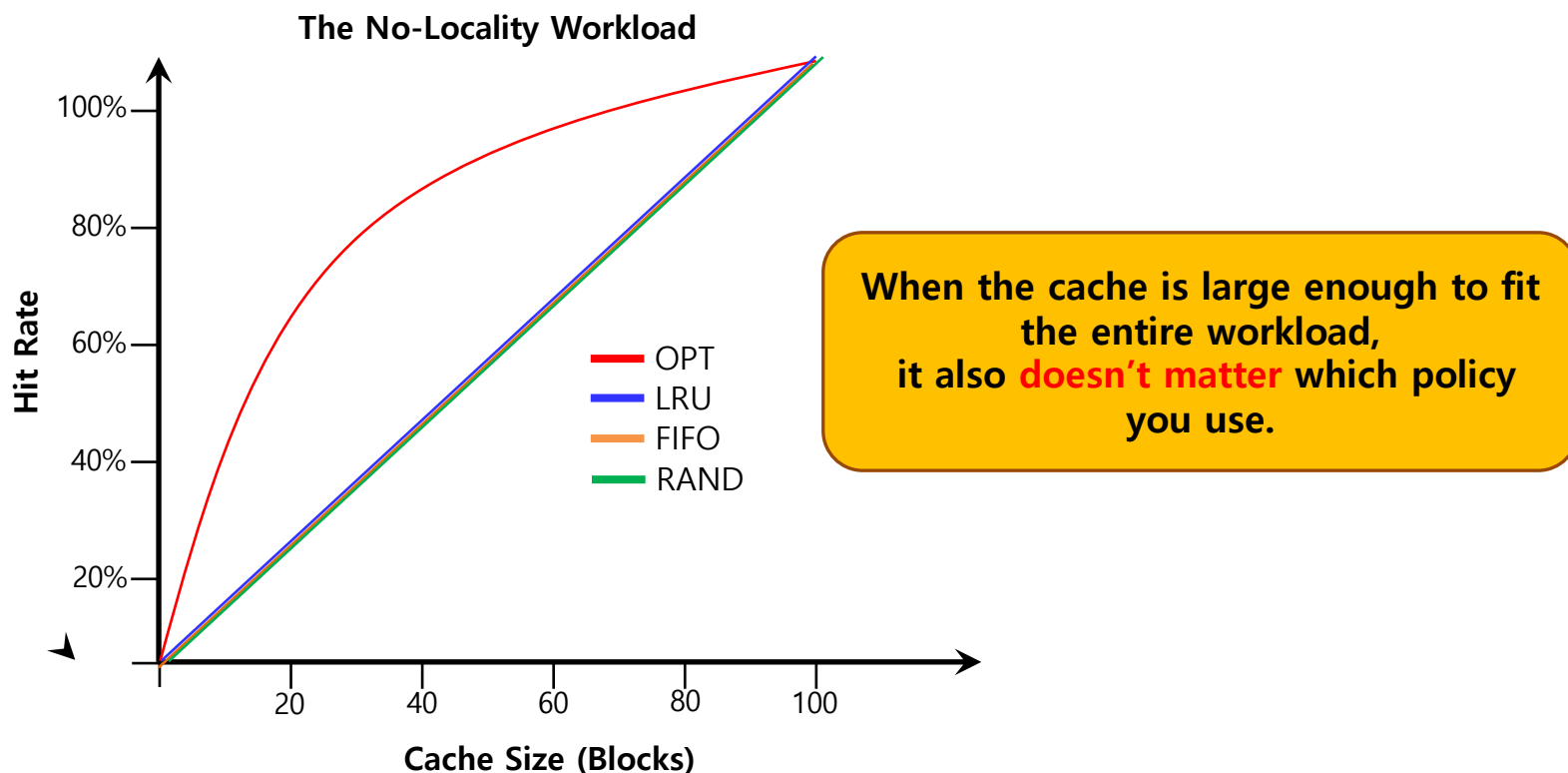
**Reference Row**

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

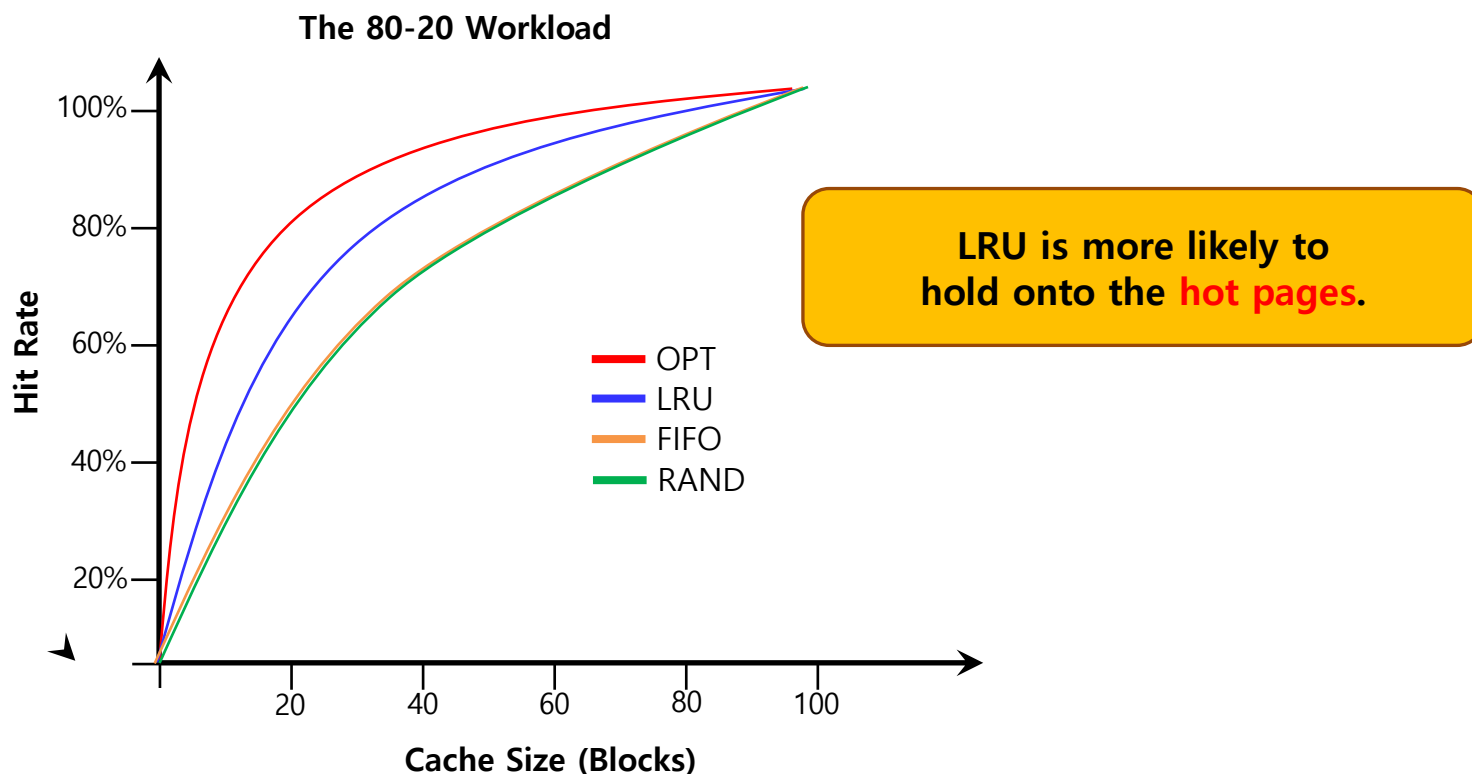
# Workload Example : The No-Locality Workload

- 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



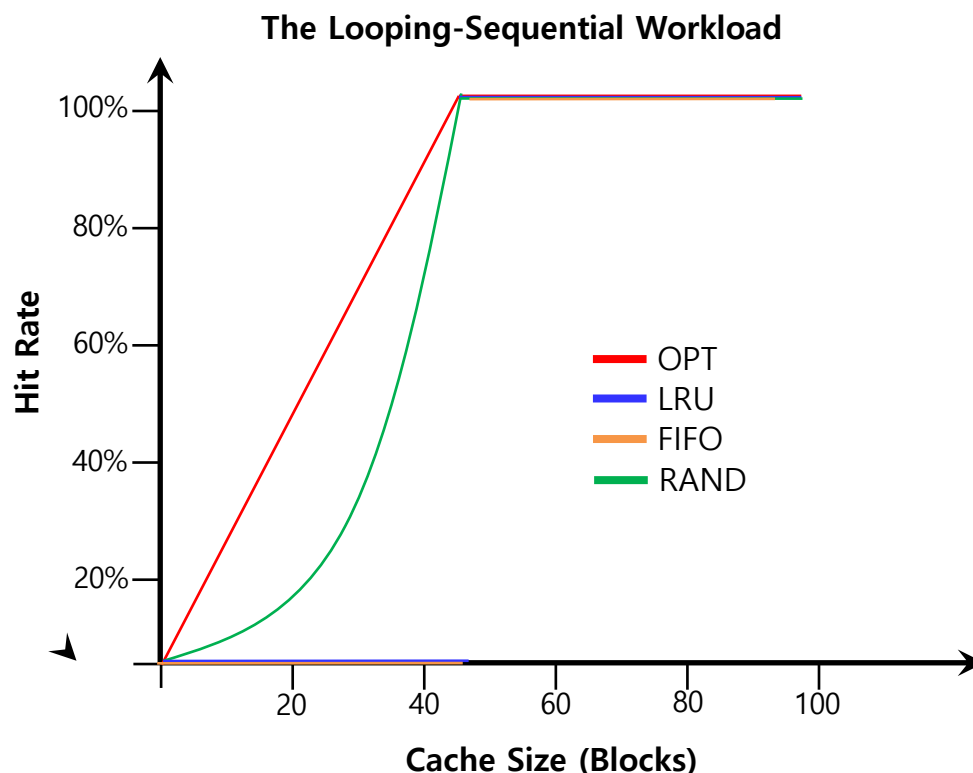
# Workload Example : The 80-20 Workload

- 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.



# Workload Example : The Looping Sequential

- 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.



# Implementing Historical Algorithms

- ▣ To keep track of which pages have been least-and-recently used, the system has to do some accounting work on **every memory reference**.
  - ◆ Add a little bit of hardware support.

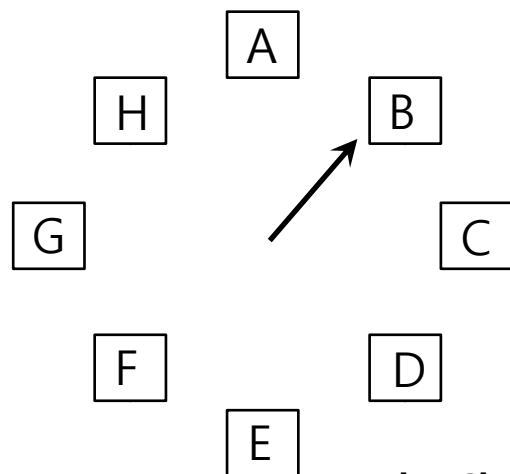


# Approximating LRU

- ▣ Require some hardware support, in the form of 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.

# Clock Algorithm

- ▣ The algorithm continues until it finds a use bit that is set to 0.



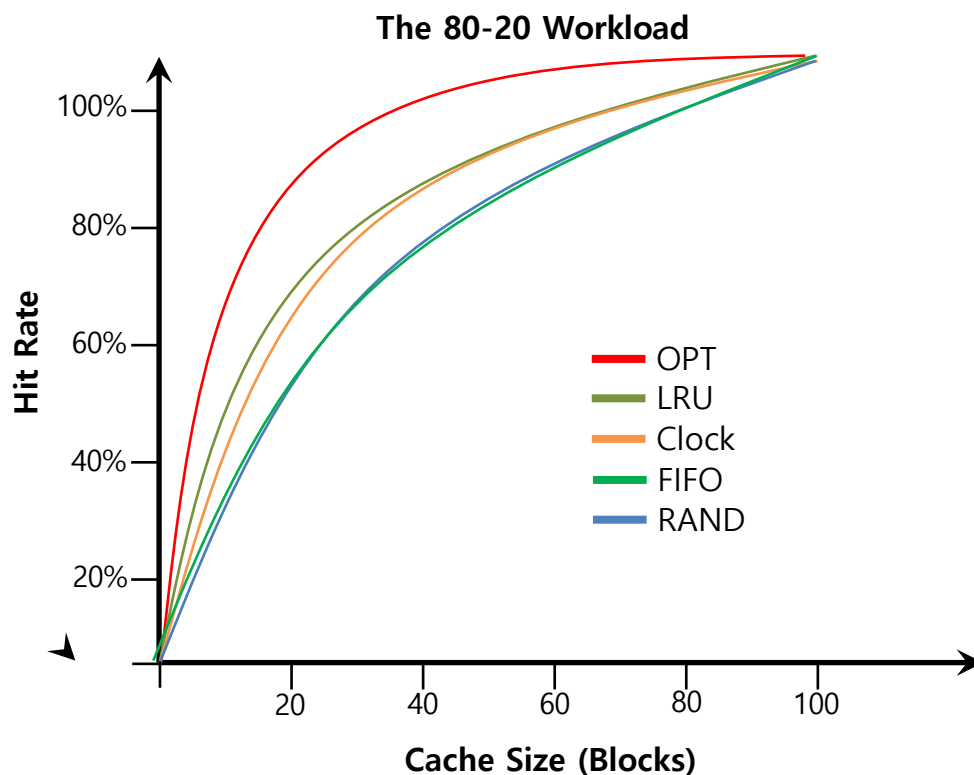
Use bit	Meaning
0	Evict the page
1	Clear <b>Use bit</b> and advance hand

The Clock page replacement algorithm

When a page fault occurs, the page the hand is pointing to is inspected.  
The action taken depends on the Use bit

# Workload with Clock Algorithm

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



# Considering Dirty Pages

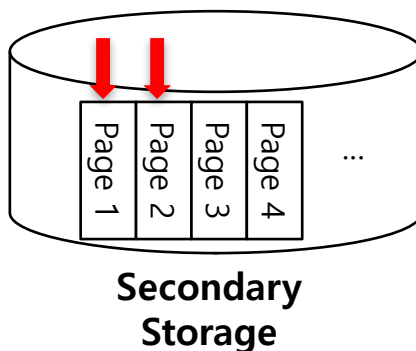
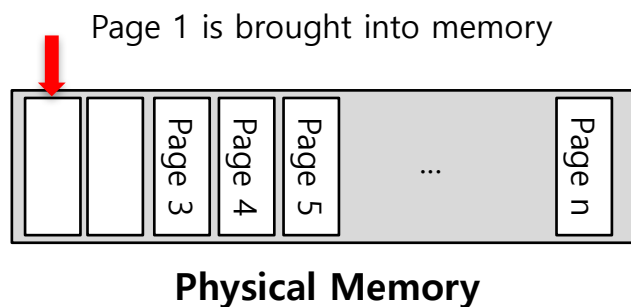
- ▣ The hardware include a **modified bit** (a.k.a **dirty bit**)
  - ◆ Page has been **modified** and is thus **dirty**, it must be written back to disk to evict it.
  - ◆ Page has not been modified, the eviction is free.

# Page Selection Policy

- ▣ The OS has to decide when to bring a page into memory.
- ▣ Presents the OS with some different options.

# Prefetching

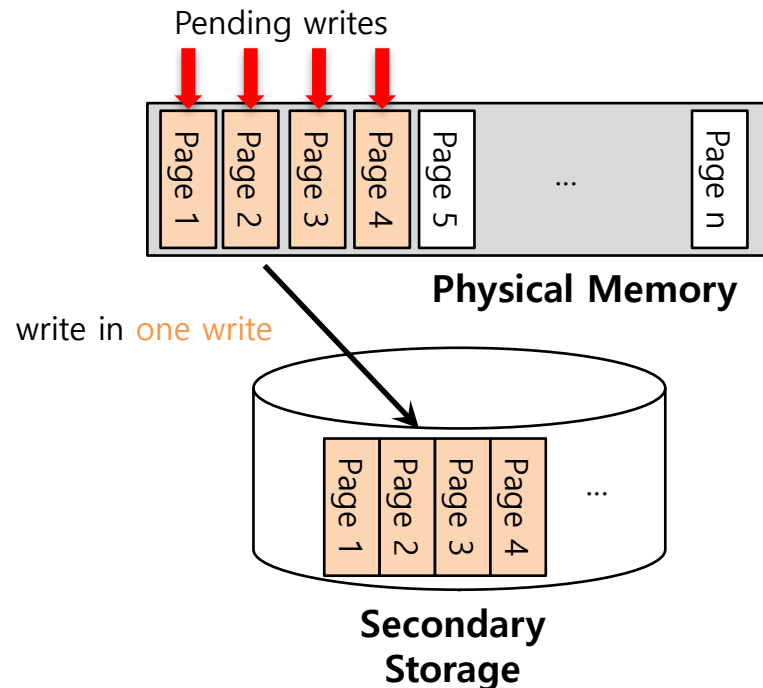
- The OS guess that a page is about to be used, and thus bring it in ahead of time.



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

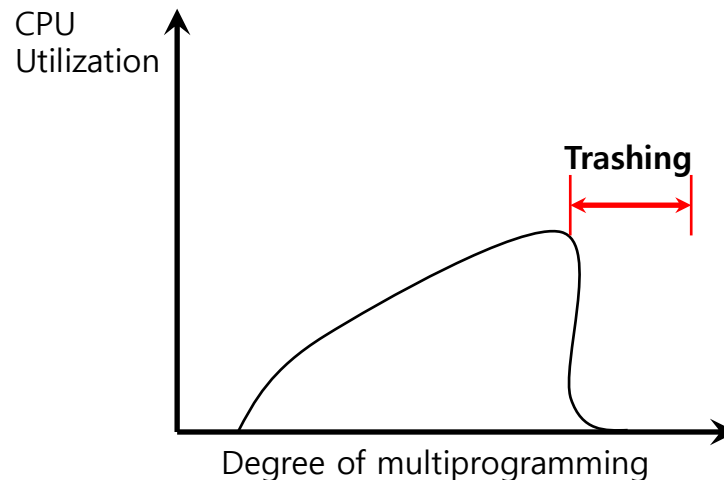
# Clustering, Grouping

- ❑ 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.





- Disclaimer: This lecture slide set was initially developed for Operating System course in Computer Science Dept. at Hanyang University. This lecture slide set is for OSTEP book written by Remzi and Andrea at University of Wisconsin.