Memory Management

Chapter 3: Section 3.4

Page Replacement

- What if we are out of frames?
  - Have to discard one
  - May have pages no longer used
- How to identify unneeded?

Working Set

- Can't assume all the pages we need will be in memory when we need them.
- Working set - pages a process is referencing in a short period, e.g the set of pages a process is currently using
- Sliding window - fixed interval over which the working set is measured.
- Page replacement - removing a page we may not need anymore.

Working Set

Page reference string:

1 2 1 5 7 1 6 3 7 1 6 4 2 7

We never reference page 5 again after step 4

Working set	Event Number
1	1
12	2
12	3
125	4
1257	5
1257	6
1567	7
1367	8
1367	9
1367	10
1367	11
1467	12
1246	13
2467	14

How about removing 5 after it falls out of the 4 step sliding window?

Working Set

Page reference string:

1 2 1 5 7 1 6 3 7 1 6 4 2 7

2 is not referenced between steps 2 and 13

Working set	Event Number
1	1
12	2
12	3
125	4
1257	5
1257	6
1567	7
1367	8
1367	9
1367	10
1367	11
1467	12
1246	13
2467	14

How about removing 5 after it falls out of the 4 step sliding window?

Basic Page Replacement

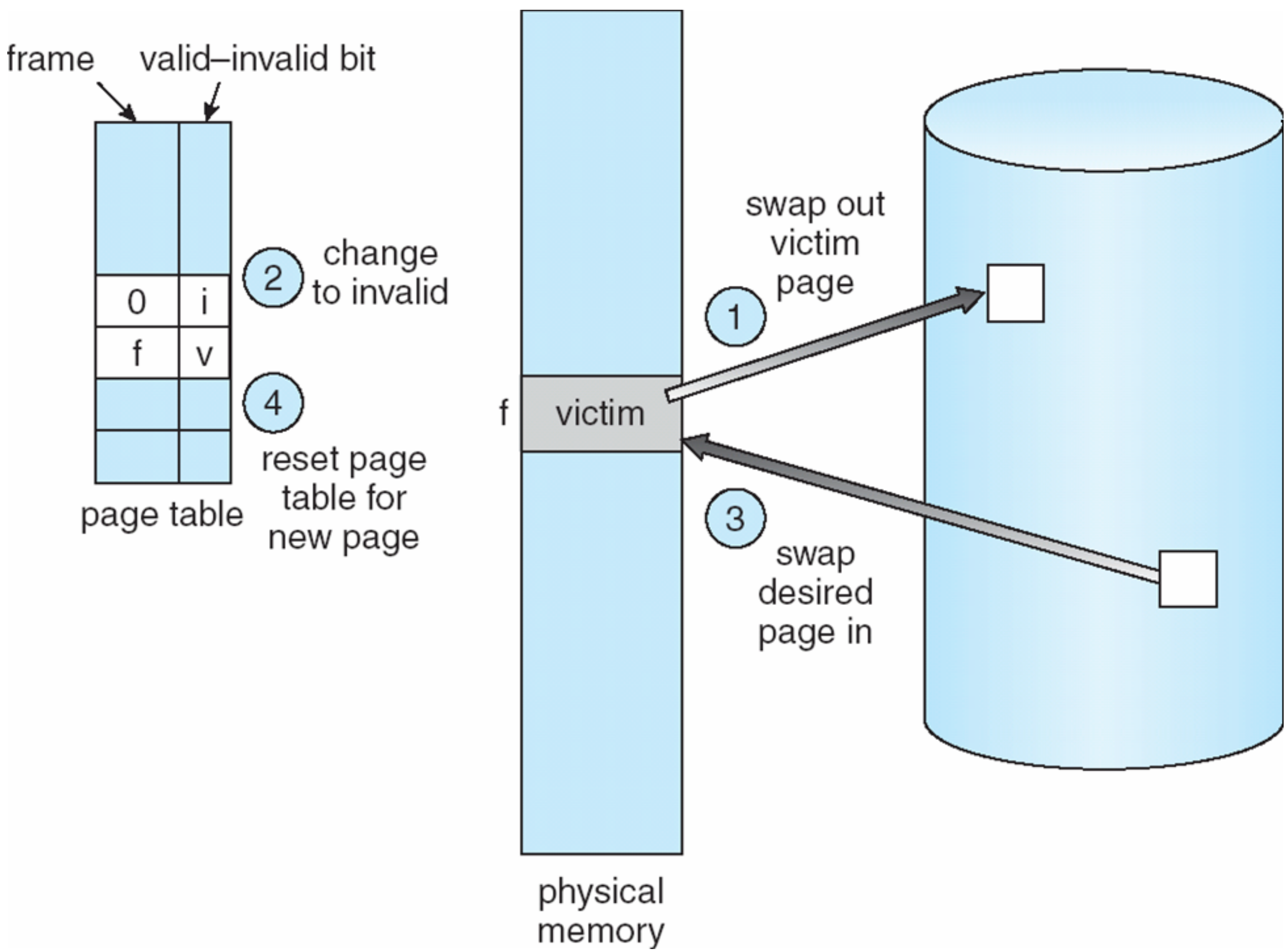
1. Find the location of the desired page on disk

2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement
   algorithm to select a **victim** frame

3. Bring the desired page into the (newly) free frame; update the page and frame tables

4. Restart the process

Page Replacement



Page Replacement Algorithms

- Want lowest page-fault rate
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string

Optimal Algorithm

Replace page that will not be used for longest period of time

4 frames example

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

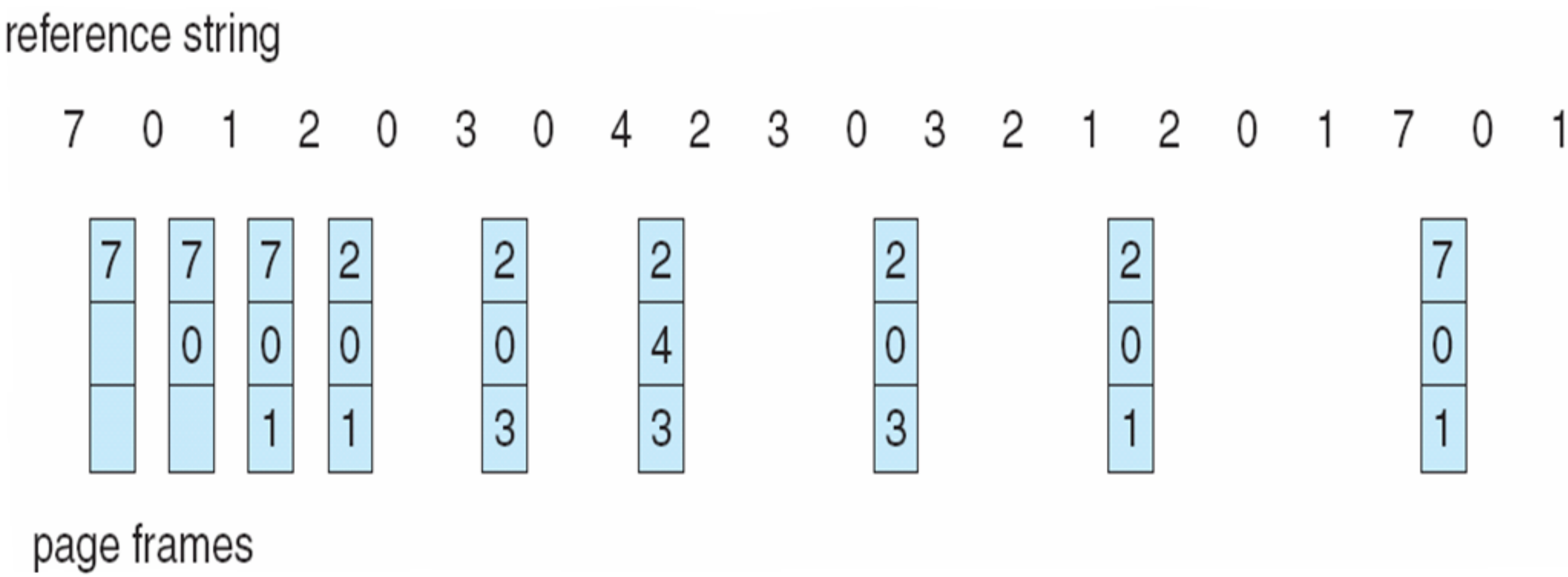
1		4
2		
3		
4	5	

6 page faults

How do you know this?

Used for measuring how well your algorithm performs against a best-case

Optimal Page Replacement



Optimal

- Unrealizable
- Have to be able to read the future
- Provides a best-case

Not Recently Used

- Each page has two status bit, R (read) and M (modified)
- On startup all bits are set to 0.
- Every page reference the R bit is set, every dirty page the M bit is set
- Every clock tick R bits are cleared

Not Recently Used

- On a page fault, OS inspects all pages and puts them into 4 categories
- Class 0: not referenced, not modified
- Class 1: not referenced, modified
- Class 2: referenced, not modified
- Class 3: referenced, modified
- NRU removes a page at random from the lowest class

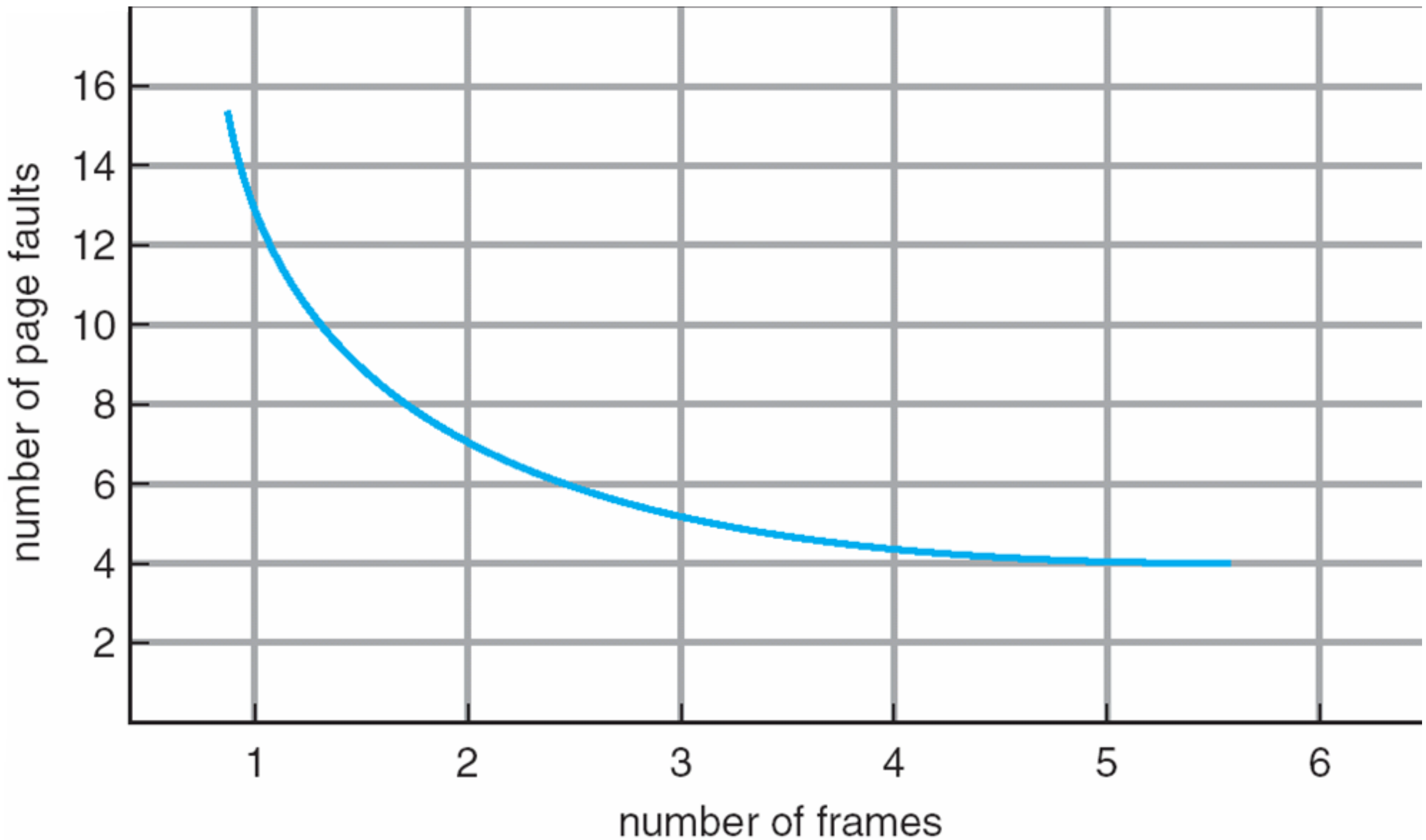
Not Recently Used

- Better to removed a modified page than a clean page frequently used.
- Easy to understand
- Moderately efficient to implement
- Not optimal, but adequate.

First In, First Out

- Eject the oldest page.
- Very low overhead
- Easy to implement
- Poor performance and erratic behavior.
  - Can end up evicting the most frequently used page
- Subject to Bélády’s anomaly

Graph of Page Faults Versus The Number of Frames



Bélády’s anomaly

- Increasing the number of page frames results in an increase in the number of page faults for a given memory access pattern.

Page Requests	3	2	1	0	3	2	4	3	2	1	0	4
Newest Page	3	2	1	0	3	2	4	4	4	1	0	0
		3	2	1	0	3	2	2	2	4	1	1
Oldest Page			3	2	1	0	3	3	3	2	4	4

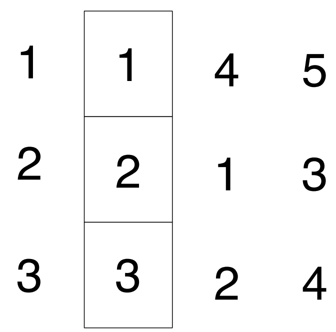
3 page frames
9 faults (red)

Page Requests	3	2	1	0	3	2	4	3	2	1	0	4
Newest Page	3	2	1	0	0	0	4	3	2	1	0	4
		3	2	1	1	1	0	4	3	2	1	0
			3	2	2	2	1	0	4	3	2	1
Oldest Page				3	3	3	2	1	0	4	3	2

4 page frames
10 faults (red)

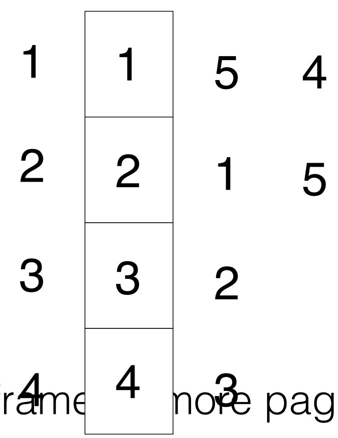
First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)



9 page faults

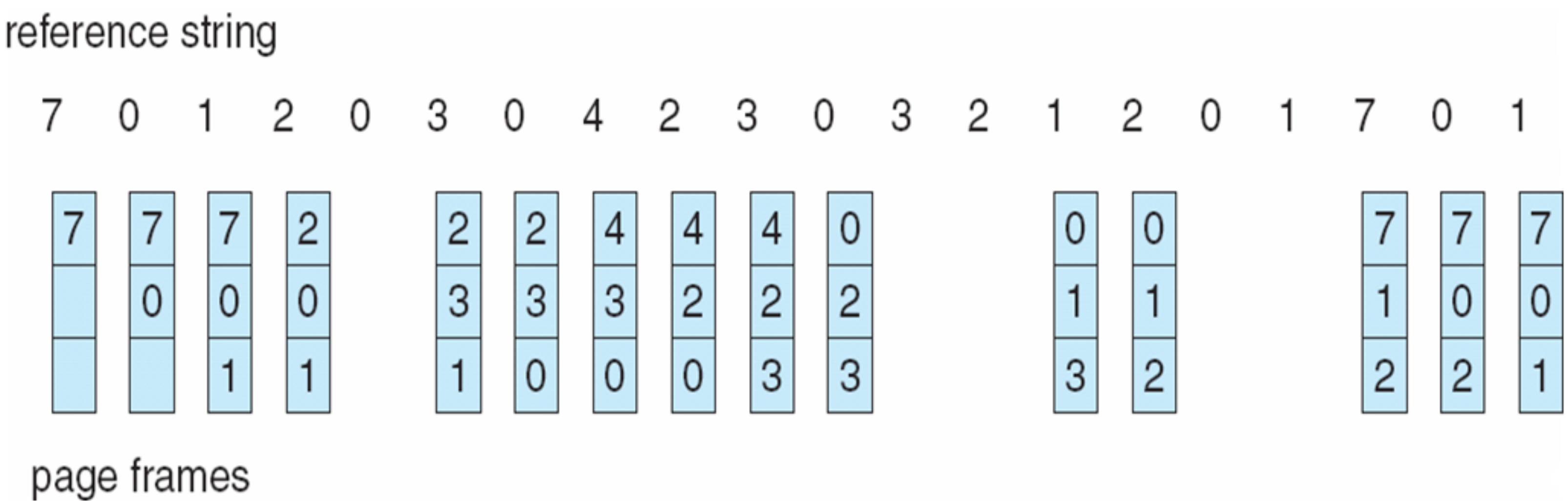
- 4 frames



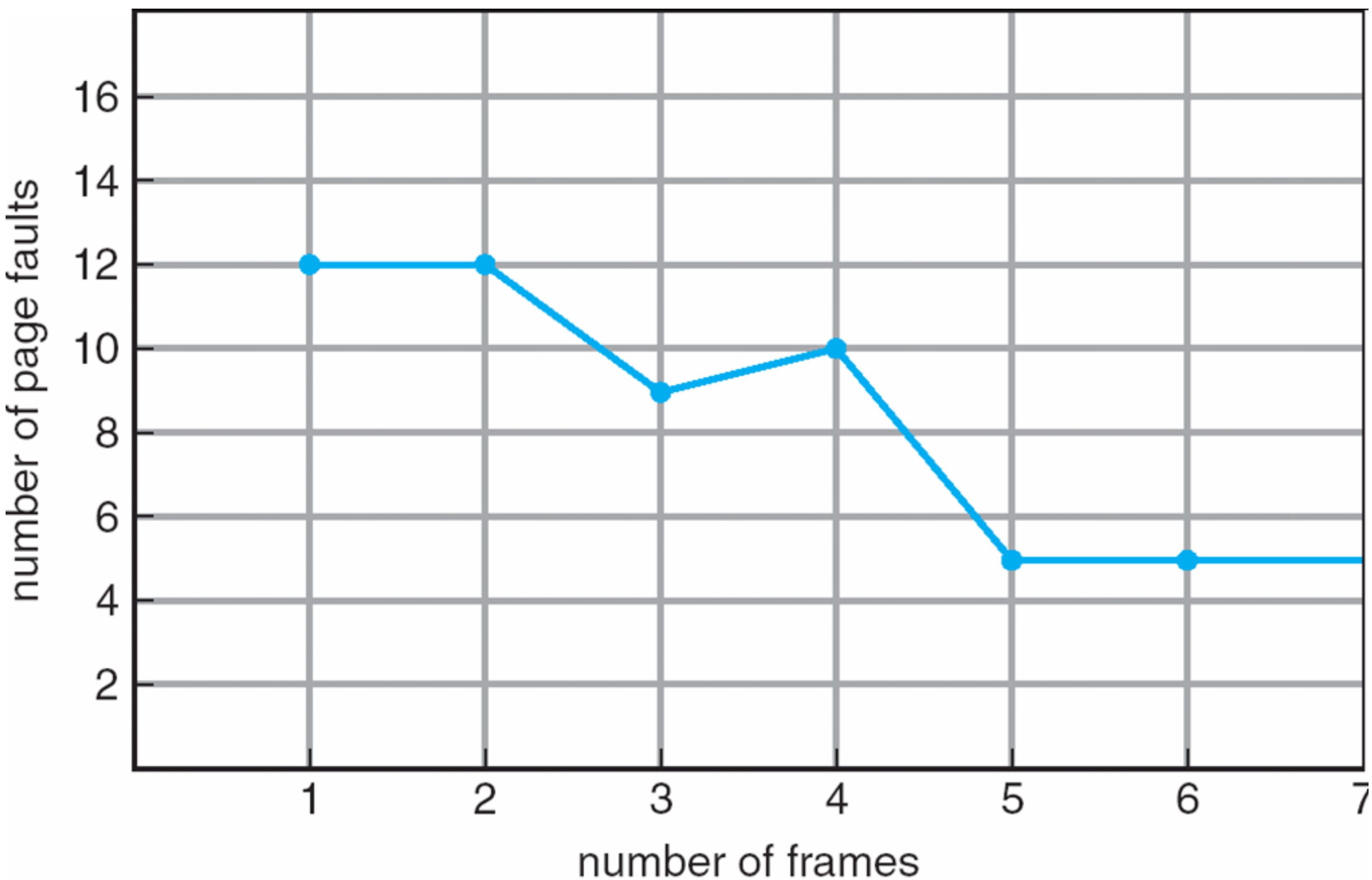
10 page faults

- Belady's Anomaly: more frames [illegible] more page faults

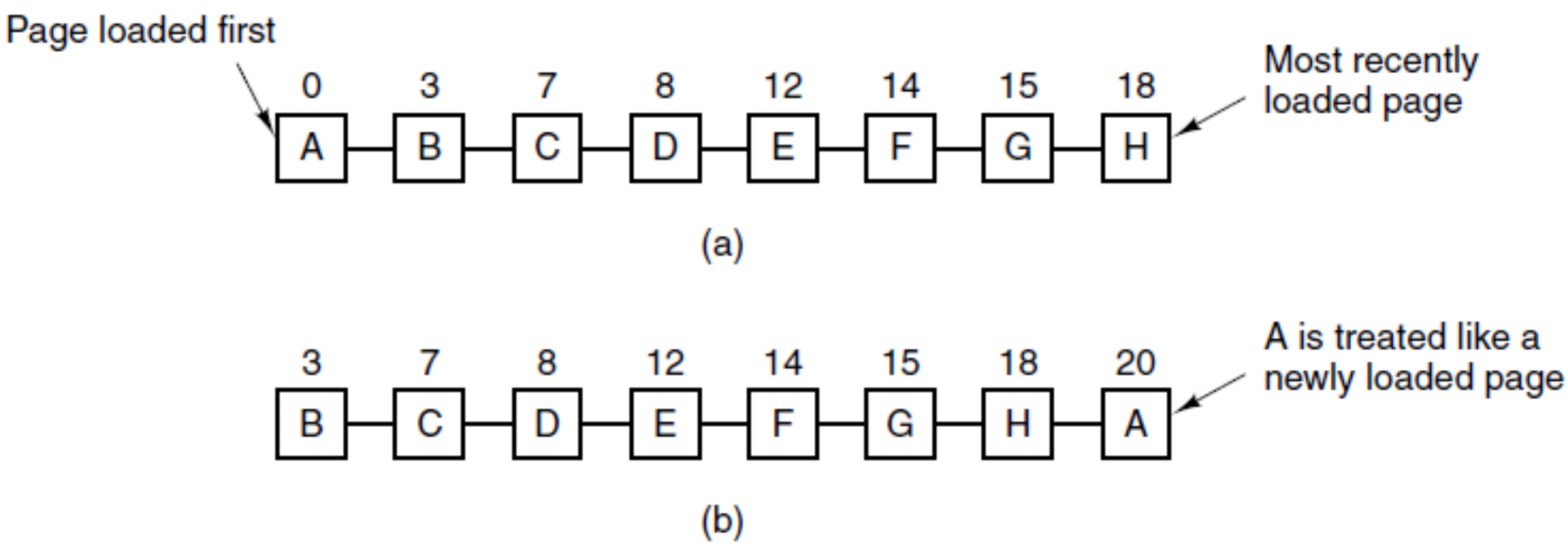
FIFO Page Replacement



FIFO Illustrating Belady's Anomaly

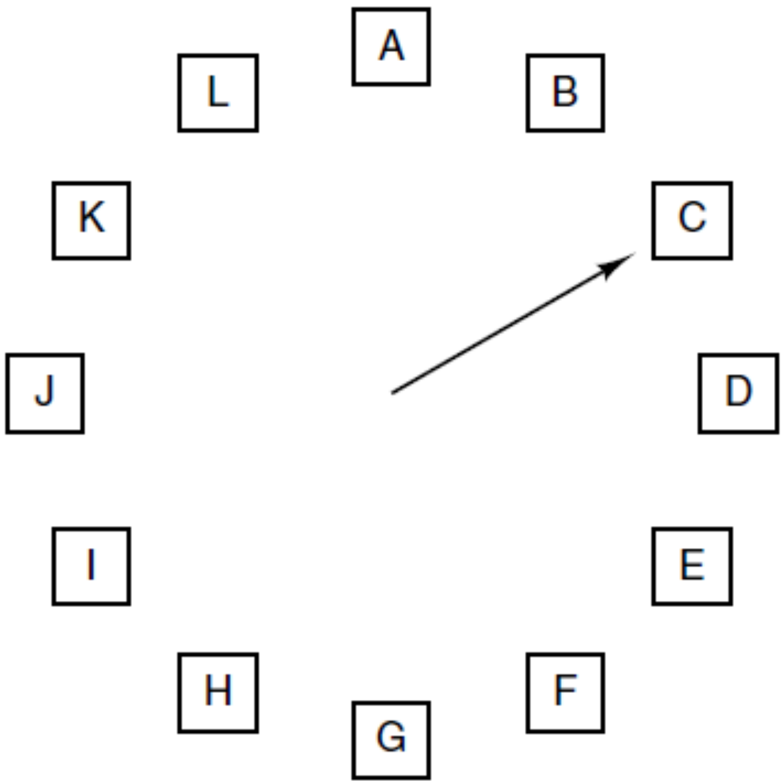


Second Chance Algorithm



- Sorted in reference time order.
- When page referenced, set R bit
- When faulted check oldest page, if R bit set then set page time to current time, clear R bit and move to the end.
- Repeat until page found with no R bit set.

Clock Page Algorithm



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

R = 0: Evict the page

R = 1: Clear R and advance hand

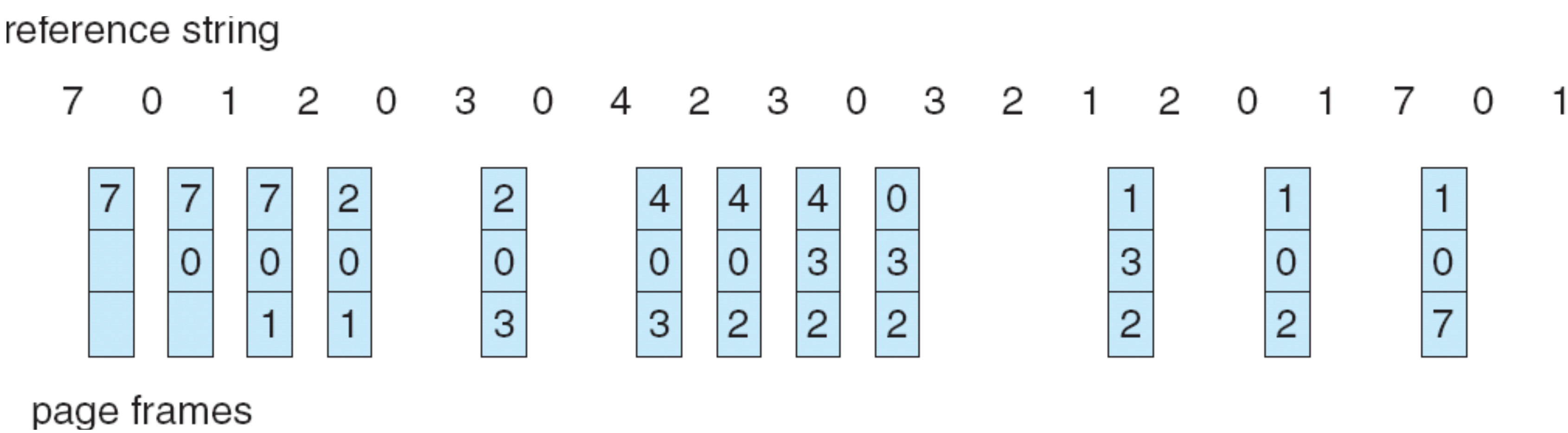
Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**

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

- Impractical to track.
  - Would need one additional memory access for each memory access or
  - Linked list of each page with most recent in front.
- Can approximate, similar to NRU
  - Need hardware assistance
  - Page reference bit in page table

LRU Page Replacement



LRU Algorithm

- Stack implementation – keep a stack of page numbers in a double link form
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - No search for replacement

LRU Algorithm

- Stack implementation – keep a stack of page numbers in a double link form

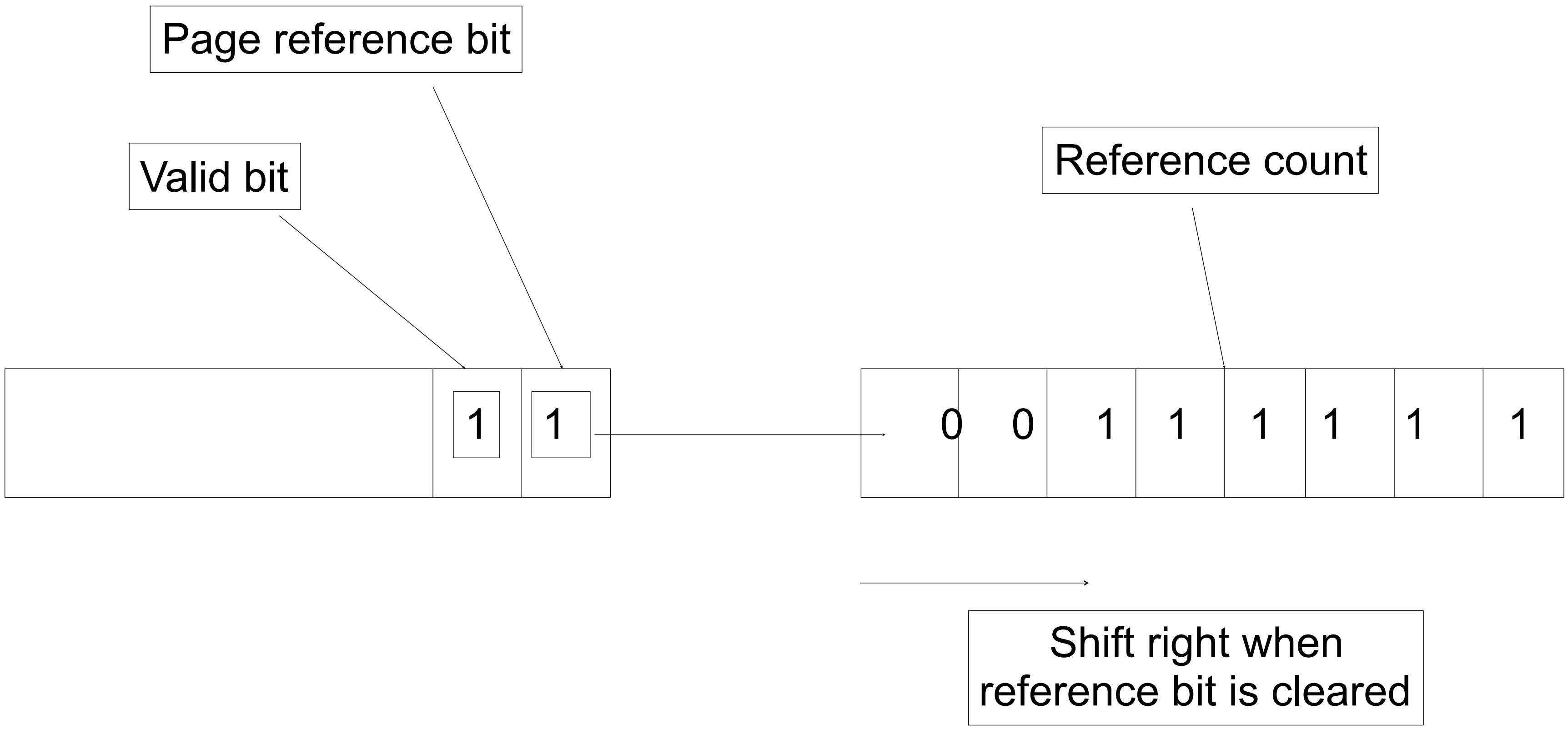
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed

  - No search for replacement

Reference Counter Variation

- Also known as Aging
- As reference bits are cleared, shift into a counter
  - one for each page
- Smallest counter referenced is least recently

Page Reference Bit & Counter



Page Reference Bit & Counter

Page	(a)	(b)	(c)	(d)	(e)
	R bits for pages 0-5, clock tick 0	R bits for pages 0-5, clock tick 1	R bits for pages 0-5, clock tick 2	R bits for pages 0-5, clock tick 3	R bits for pages 0-5, clock tick 4
	1 0 1 0 1 1	1 1 0 0 1 0	1 1 0 1 0 1	1 0 0 0 1 0	0 1 1 0 0 0
0	10000000	11000000	11100000	11110000	01111000
1	00000000	10000000	11000000	01100000	10110000
2	10000000	01000000	00100000	00010000	10010000
3	00000000	00000000	10000000	01000000	00100000
4	10000000	11000000	01100000	10110000	01011000
5	10000000	01000000	10100000	01010000	00101000

Counting Algorithms

- Keep a counter of the number of references that have been made to each page

- LFU Algorithm: replaces page with smallest count

- MFU Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Allocation of Frames

- Each process needs *minimum* number of pages
- Example: IBM 370 – 6 pages to handle SS MOVE instruction:
  - instruction is 6 bytes, might span 2 pages
  - 2 pages to handle *from*
  - 2 pages to handle *to*
- Two major allocation schemes
  - fixed allocation
  - priority allocation

Fixed Allocation

- Equal allocation – For example, if there are 100 frames and 5 processes, give each process 20 frames.
- Proportional allocation – Allocate according to the size of process

  - s_i = size of process p_i
  - $S = \sum s_i$
  - m = total number of frames
  - a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$$m = 64$$

$$s_i = 10$$

$$s_2 = 127$$

$$a_1 = \frac{10}{137} \times 64 \approx 5$$

$$a_2 = \frac{127}{137} \times 64 \approx 59$$

Priority Allocation

- Use a proportional allocation scheme using priorities rather than size

- If process P_i generates a page fault,
  - select for replacement one of its frames
  - select for replacement a frame from a process with lower priority number

Global vs. Local Allocation

- Global replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another

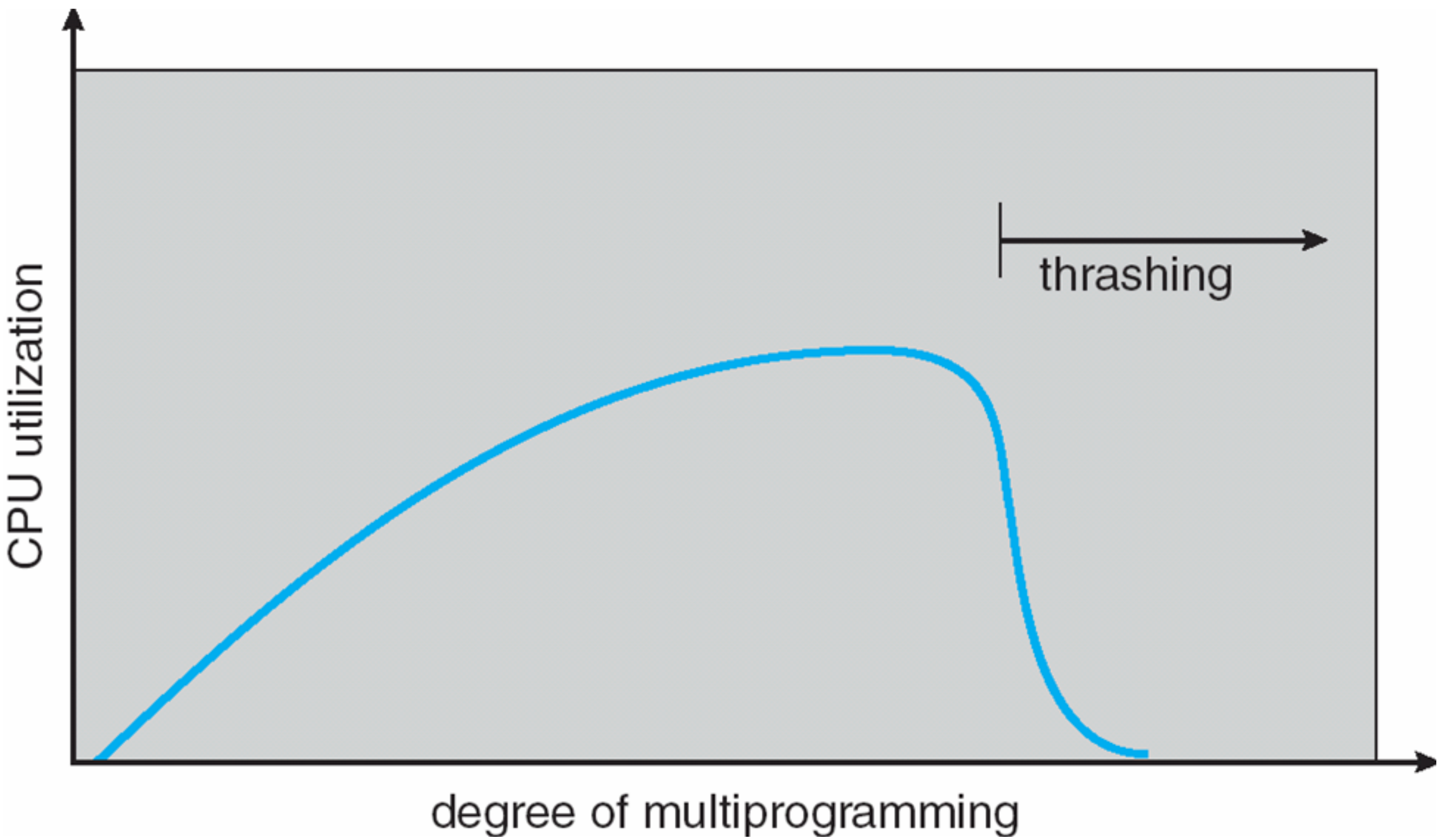
- Local replacement – each process selects from only its own set of allocated frames

Thrashing

- If a process does not have "enough" pages, the page-fault rate is very high. This leads to:
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
  - another process added to the system

- **Thrashing** – a process is busy swapping pages in and out

Thrashing (Cont.)



Demand Paging and Thrashing

Why does demand paging work?

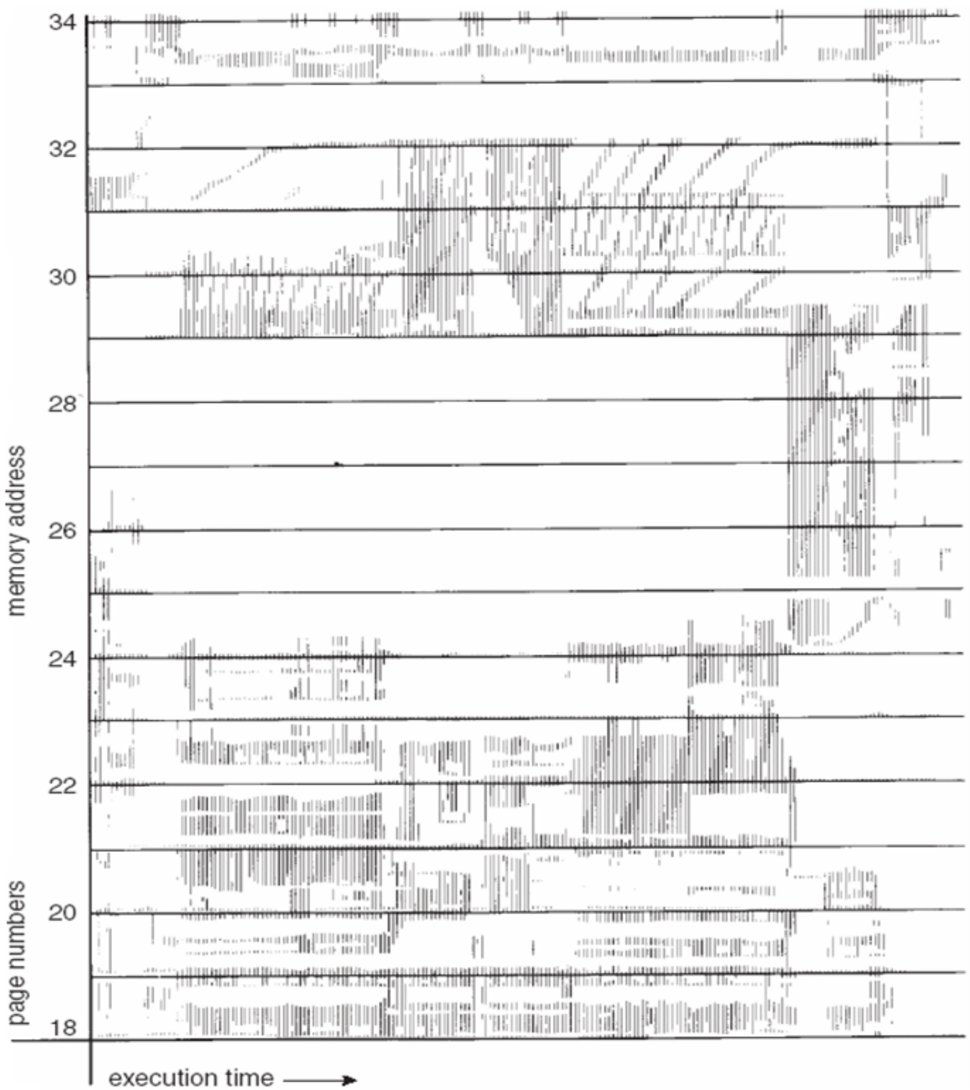
Locality model

- Process migrates from one locality to another
- Localities may overlap

Why does thrashing occur?

Σ size of locality > total memory size

Locality In A Memory-Reference Pattern



Page-Fault Frequency Scheme

- Establish “acceptable” page-fault rate
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame