KOÇ UNIVERSITY

Page Replacement

Didem Unat

Lecture 21

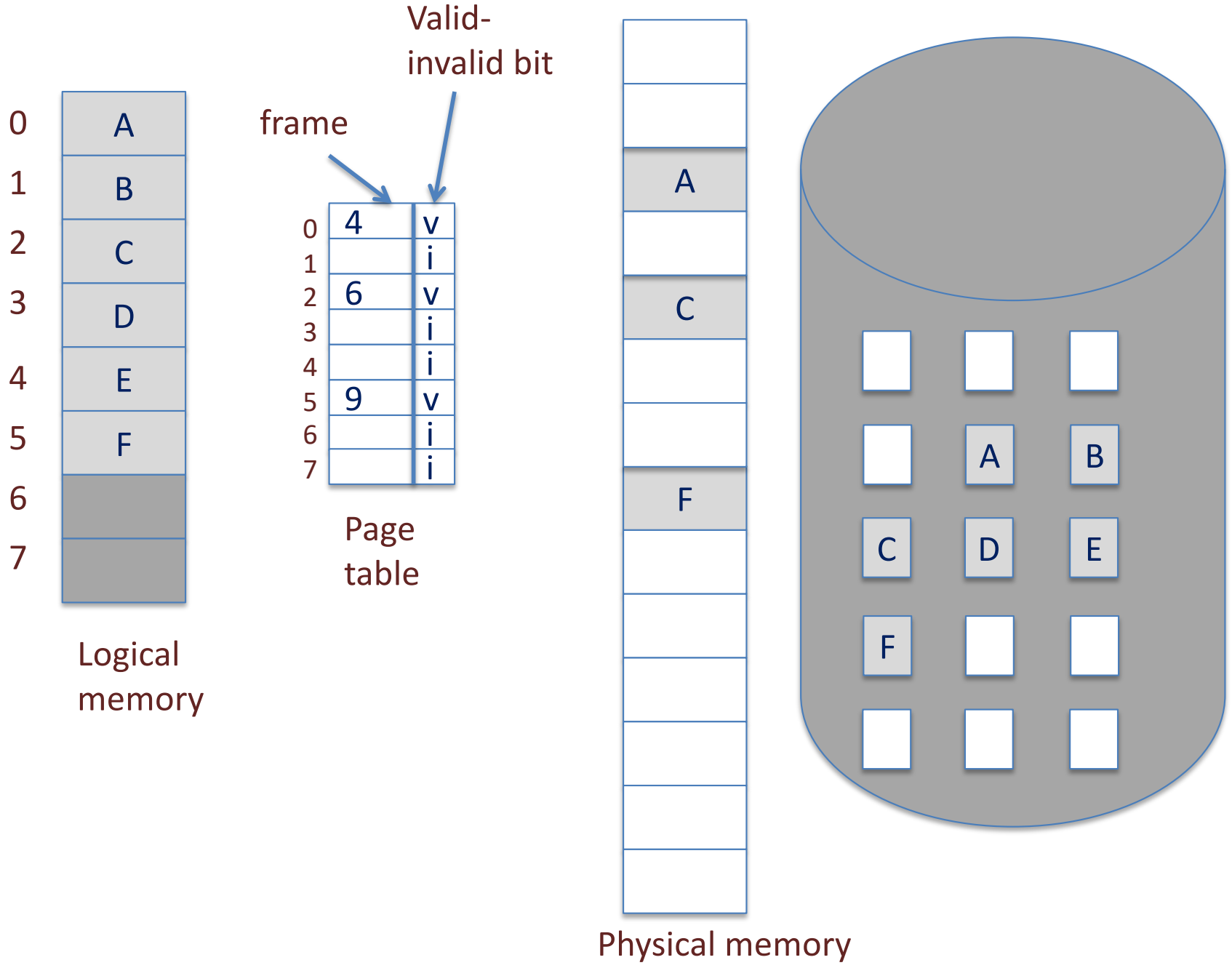
COMP304 - Operating Systems (OS)

Virtual Memory Management

- **Virtual memory**: separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution.
  - Logical address space can therefore be **much larger than** physical address space.
  - Allows address spaces to be shared by several processes.
  - Allows for more efficient process creation.

- Virtual memory can be implemented via:
  - **Demand paging**

Page Table when some pages are not in Main Memory



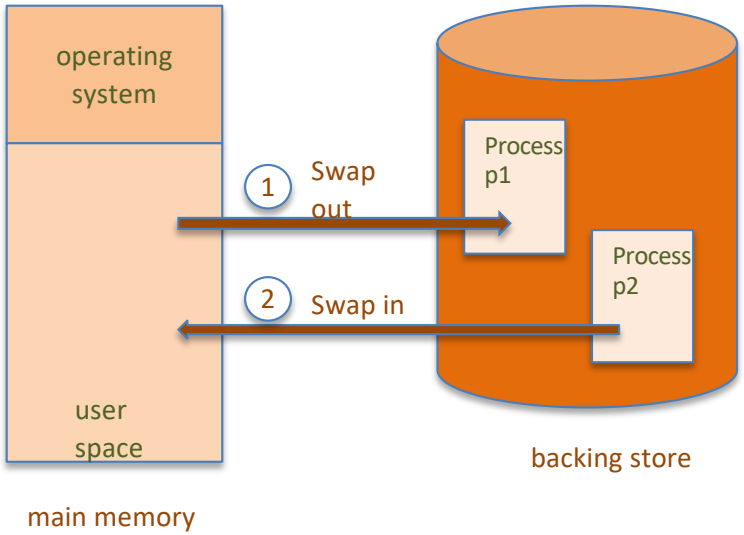
Page Fault

- If there is a reference to a page, first reference to that page will trap to operating system: **page fault**

1. Operating system looks at another table to decide:
   - Invalid reference $\Rightarrow$ abort
   - Just not in memory
2. Get empty frame
3. Swap page into frame
4. Reset tables
5. Set validation bit = **v**
6. Restart the instruction that caused the page fault

Swap Space

- *Swap space* is used when the amount of physical memory is full.
  - If the system needs more memory resources and the memory is full, inactive pages in memory are moved to the swap space.
- Swap space is located on hard drives, which have a slower access time than physical memory.
- Swap space is suggested to be equal 2x physical memory



Page and Frame Replacement Algorithms

- **Page-replacement algorithm**
  - Want lowest page-fault rate on both first access and re-access
- **Frame-allocation algorithm** determines
  - How many frames to give each process
  - Which frames to replace

- Evaluate an algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
  - String is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault
- In all our examples, the reference string is

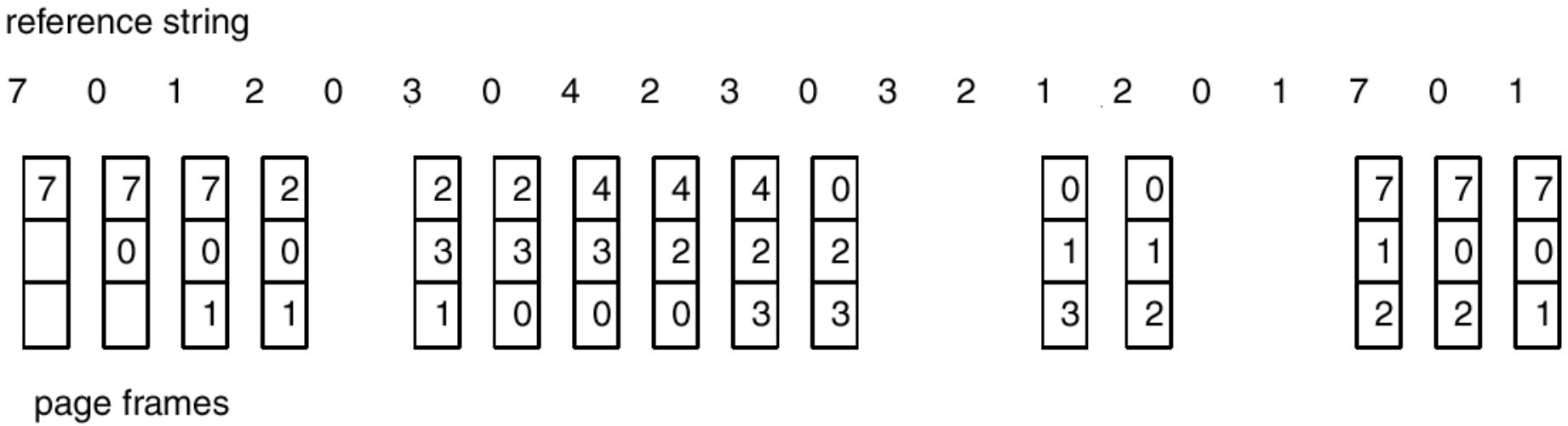
7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

1. FIFO Page Replacement

Each page is associated the time when it was brought into memory

When a page must be replaced, the oldest page is chosen

Example: 3 frames/process

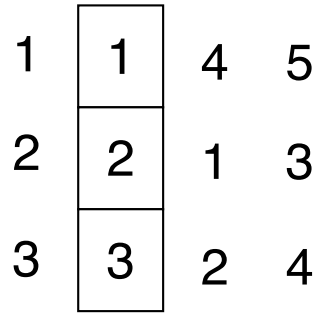


15 page faults

First-in-First-out (FIFO) Algorithm

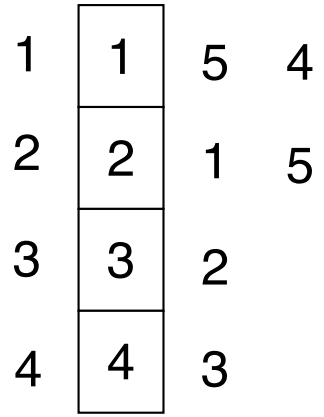
- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

3 frames



9 page faults

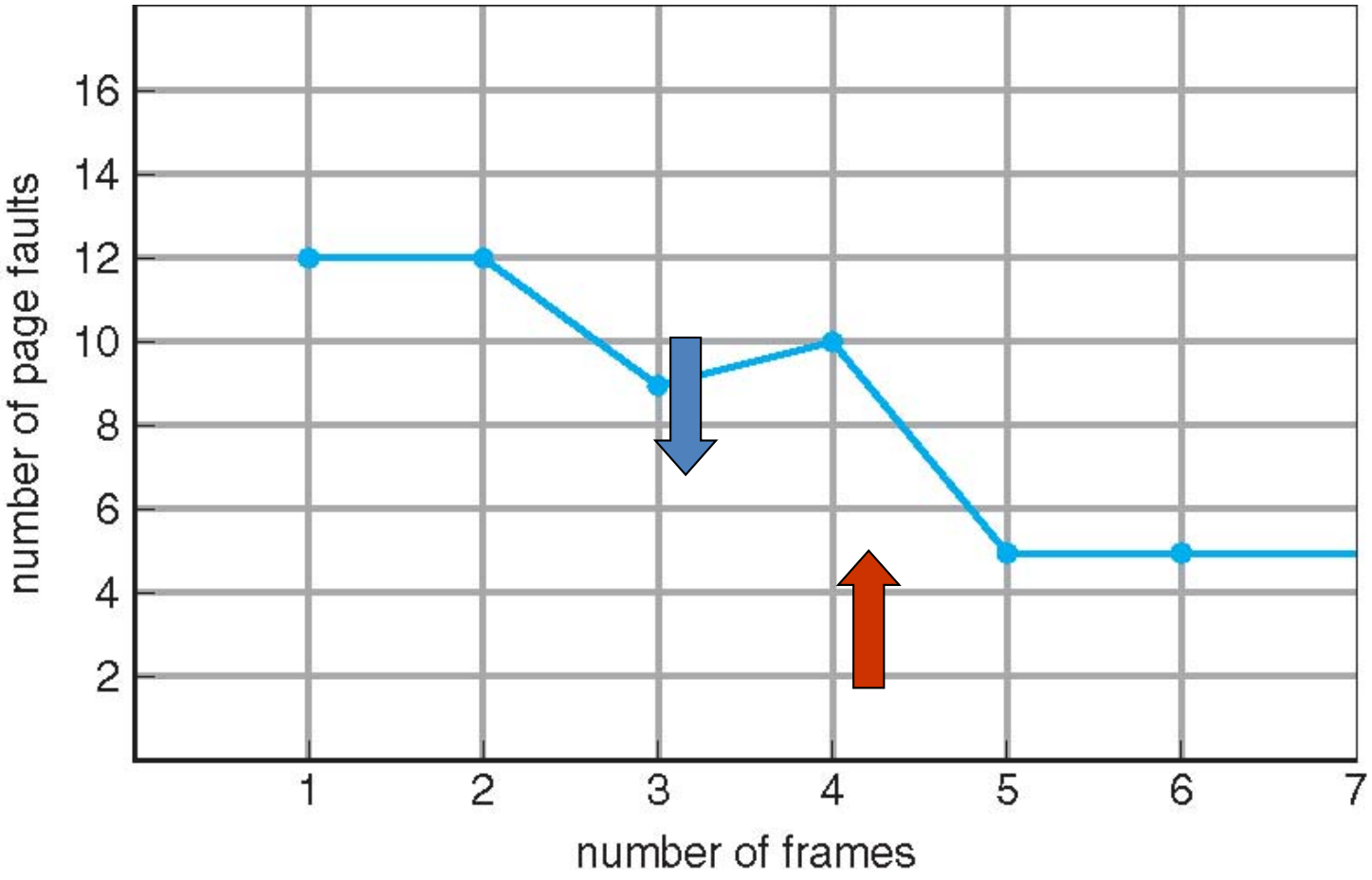
4 frames



10 page faults

- FIFO Replacement: Belady's Anomaly (page fault rate may *increase* as the number of allocated frames increases!)
- Ideal: more frames ⇒ less page faults

FIFO Illustrating Belady's Anomaly



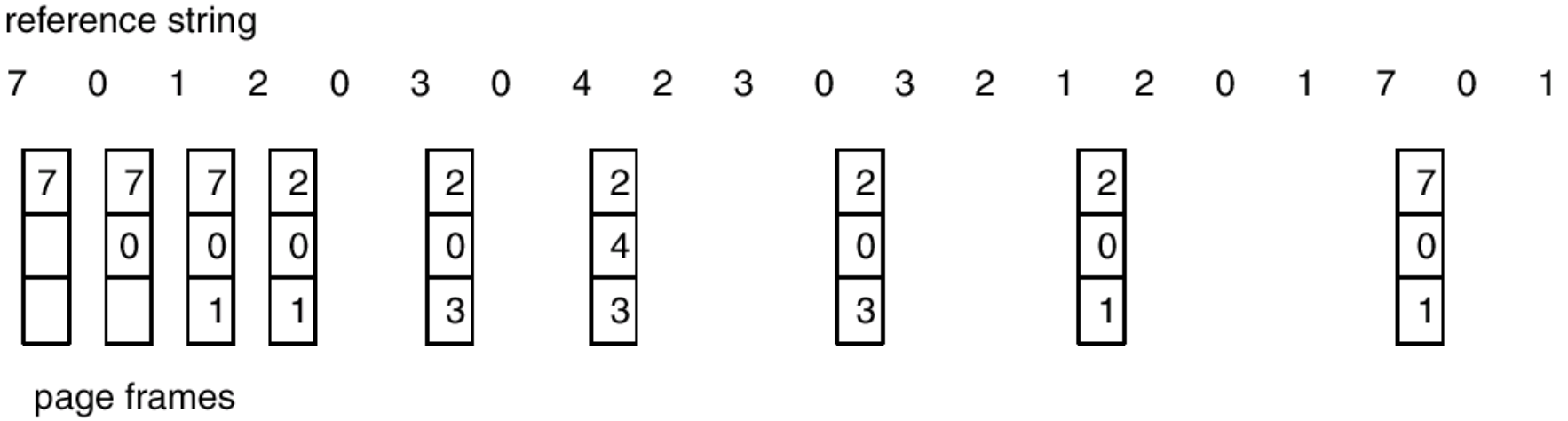
https://en.wikipedia.org/wiki/B%C3%A9l%C3%A1dy's_anomaly

2. Optimal Page Replacement

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

Lowest page-fault rate of all algorithms

Never suffers from Belady's anomaly

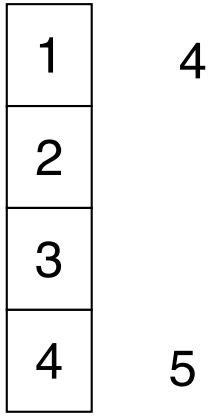


9 page faults

Optimal Algorithm

- 4 frames example

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



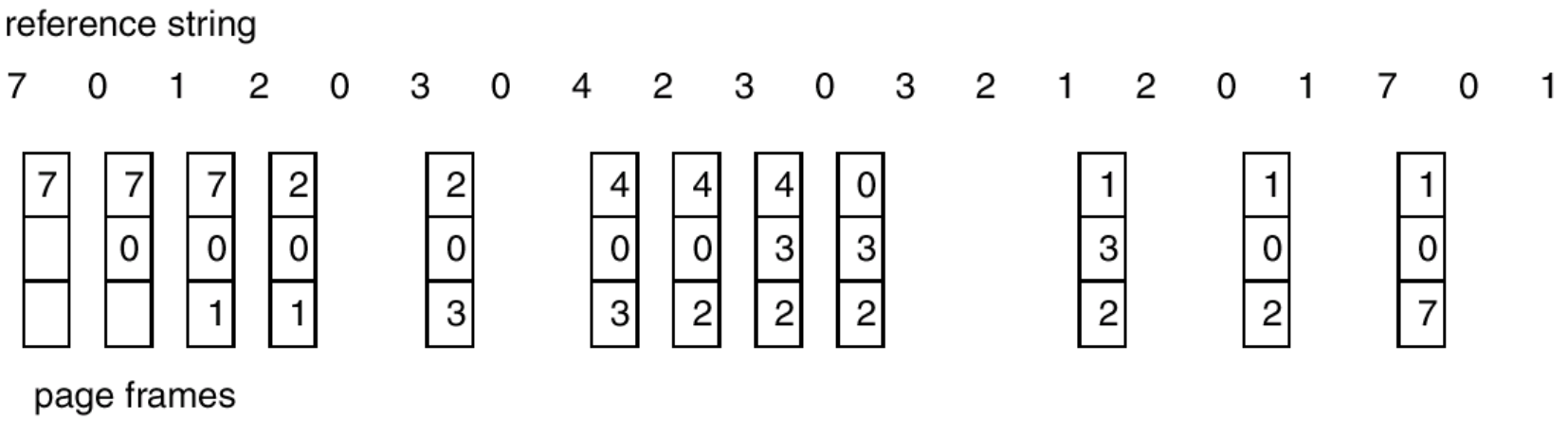
6 page faults

- Requires future knowledge of reference string , which is not possible!
- Used for measuring how well your placement algorithm performs because you cannot be better than optimal

3. Least Recently Used (LRU) Algorithm

Replace page that has not been used for longest period of time.

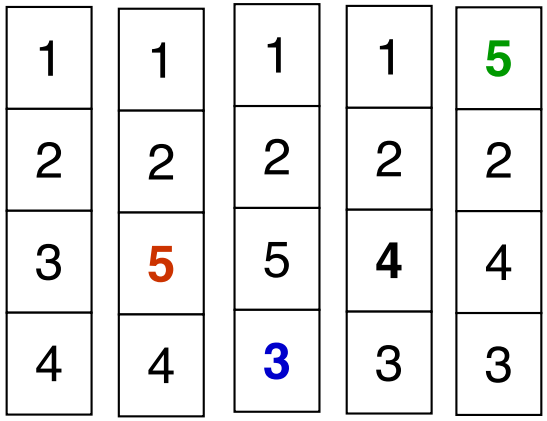
Does not suffer from Belady's anomaly



12 page faults

Least Recently Used (LRU) Algorithm

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



8 page faults

- Counter implementation of LRU
  - Every page entry has a counter; every time a page is referenced, copy the clock into the counter of the page
  - When a page needs to be replaced, look at the counters to determine which one to change (replace the page with the smallest counter) → least recently used

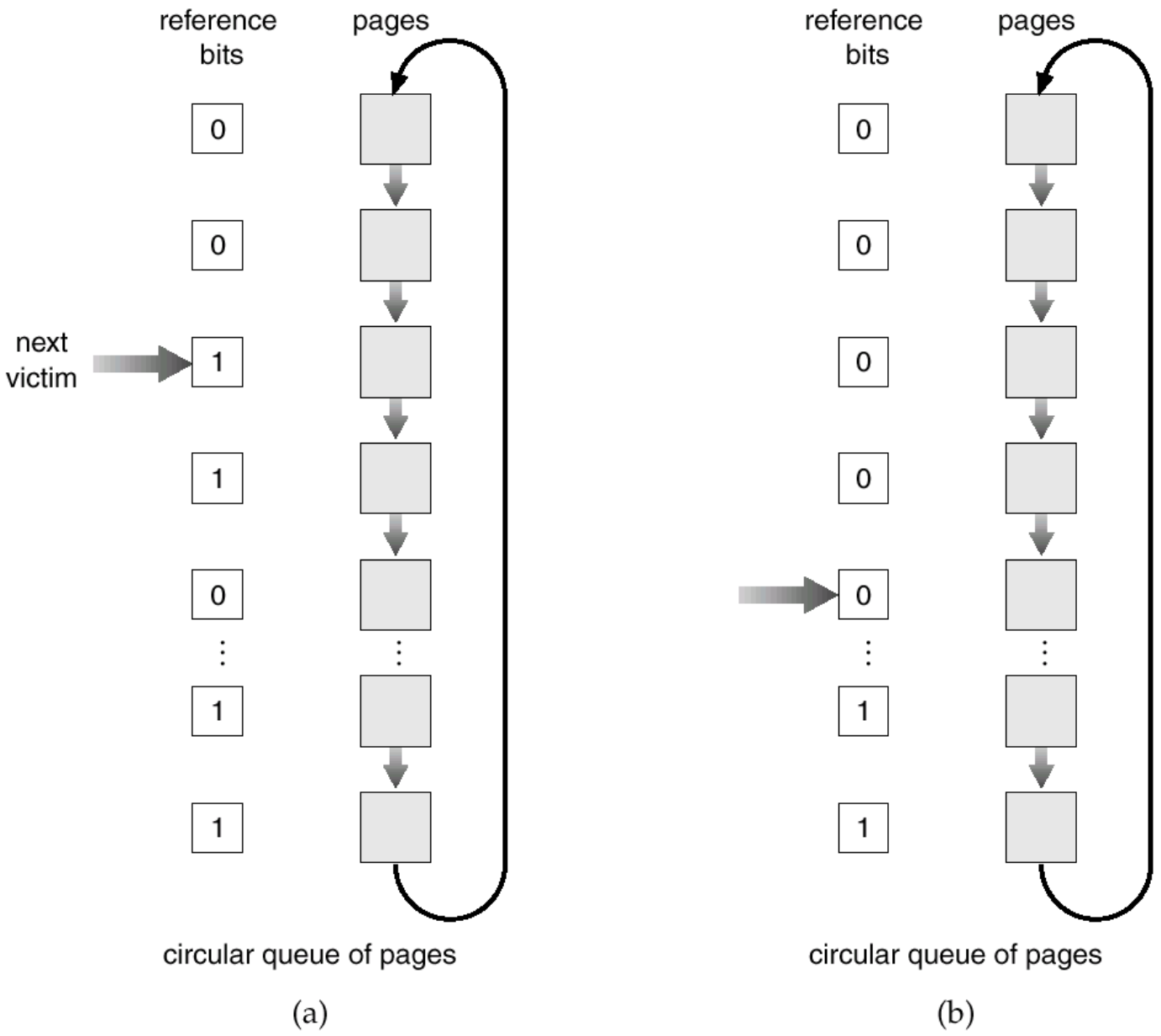
4. LRU Approximation Algorithms

- LRU needs hardware support
- **Reference bit**
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Periodically reset all the bits to zero
  - Replace any with reference bit = 0 (if one exists)
    - We do not know the order, however
- **Additional Reference bits**
  - 8 bit shift register
  - At regular intervals OS shifts the reference bit for each page into the high-order bit
  - 1100100 -> 0110010 if not used
  - 1100100 -> 1110010 if used

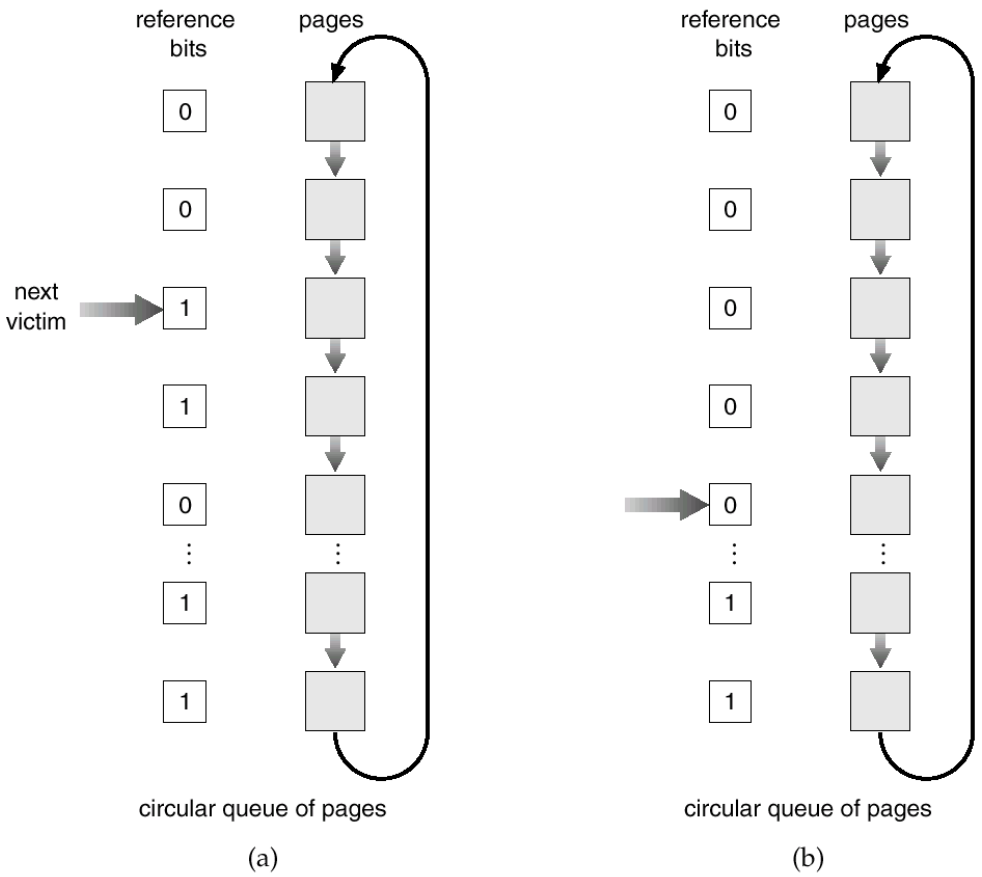
LRU Approximation Algorithms

- **Second-chance algorithm**
  - Generally FIFO, plus hardware-provided reference bit
  - Clock replacement
  - If page to be replaced has
    - Reference bit = 0 -> replace it
    - Reference bit = 1 then:
      - set reference bit 0, leave page in memory
      - replace next page, subject to same rules

Second-chance (clock) Page-Replacement Algorithm



Question?



- The pointer indicates the candidate page for replacement.
- What can you say about the system if you notice the following behavior?
  - Pointer is moving fast
  - Pointer is moving slow

5. Counting Algorithms

- Keep a counter of the number of references that have been made to each page.
  - LFU (least frequently used) Algorithm: replaces page with smallest count. Set the counter to zero when a page is moved into memory
  - MFU (most frequently used) Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Applications and Page Replacement

- All of these algorithms have OS guessing about future page access
- Some applications have better knowledge – i.e. databases
- Memory intensive applications can cause double buffering
  - OS keeps copy of a page in memory as I/O buffer
  - Application keeps page in memory for its own work

Allocation of Frames

- How many frames does each process get?
- Each process needs **minimum** number of pages
  - There must be enough frames to hold all the different pages that any single instruction can reference
- Two major allocation methods:
  - fixed allocation (equal, proportional)
  - priority allocation

Fixed Allocation

- Equal allocation:
  - e.g., if 100 frames and 5 processes, give each 20 pages.
- Proportional allocation:
  - Allocate according to the size of process.

$s_i = \text{size of process } p_i$

$S = \sum s_i$

$m = \text{total number of frames}$

$a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m$

$m = 64$

$s_1 = 10$

$s_2 = 127$

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

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

Priority Allocation and Global Replacement

- Priority Allocation
  - Use a proportional allocation method 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.

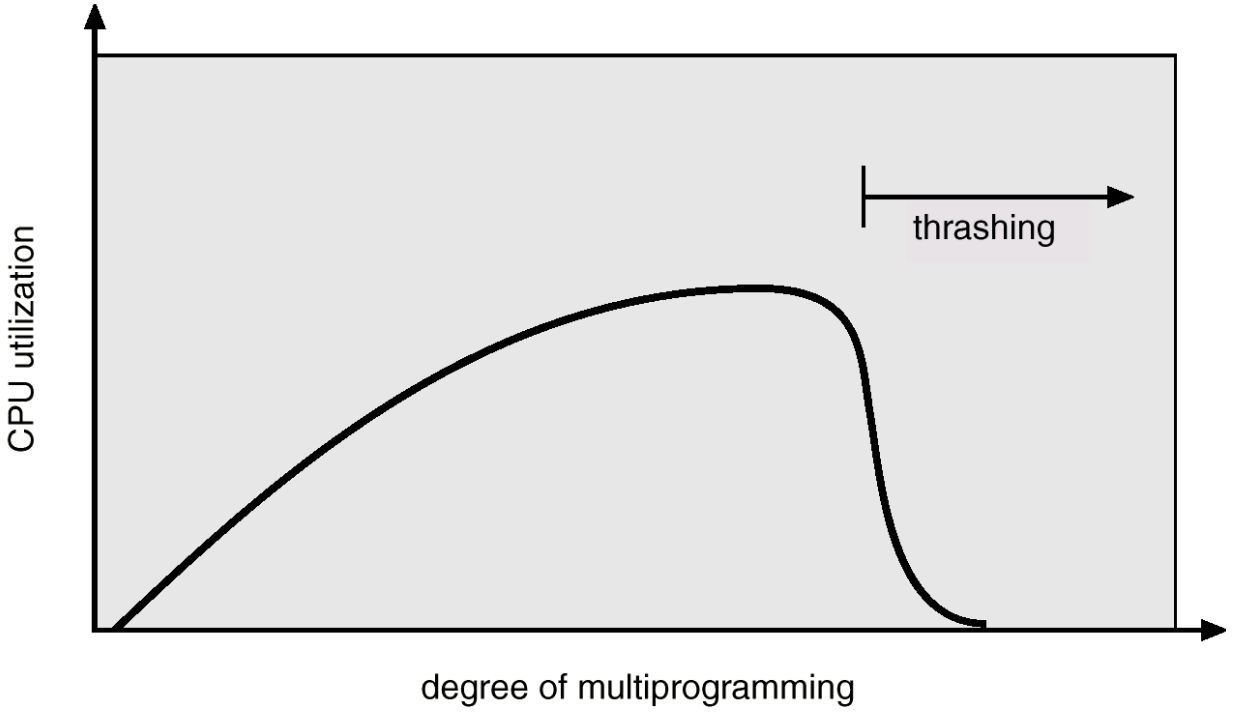
- **Global** replacement:
  - process selects a replacement frame from the set of all frames; one process can take a frame from another.
  - Generally results in greater system throughput
- **Local** replacement:
  - each process selects from only its own set of allocated frames.

Thrashing

- If a process does not have "enough" pages, the page-fault rate is very high, leading to
  - Low CPU utilization
  - Operating system thinks that it needs to increase the degree of multiprogramming
    - Why?
  - Another process is added to the system, makes it worse

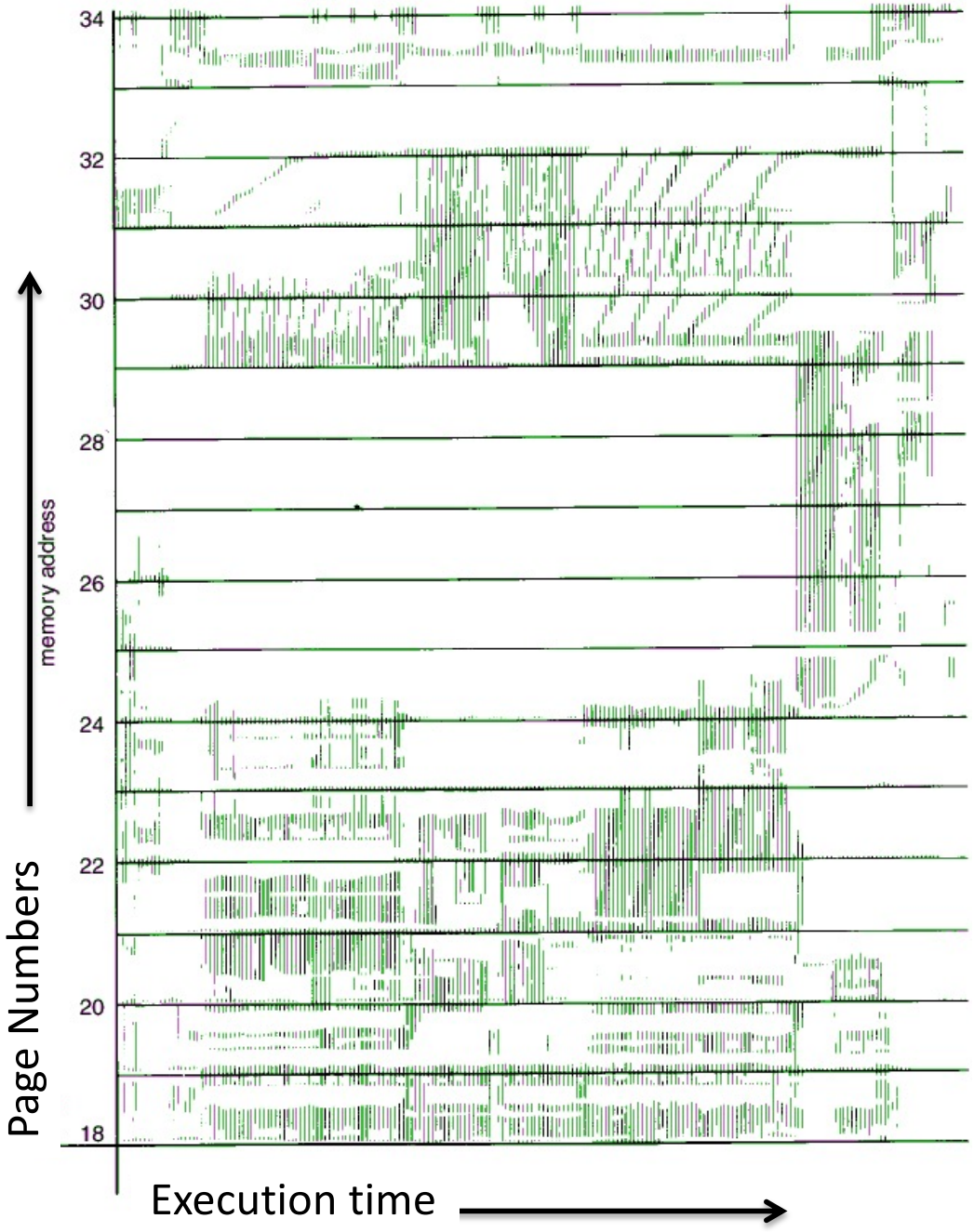
- **Thrashing** ≡ a process is busy swapping pages in and out
- A process is **thrashing** if it is spending more time for paging than executing.

Thrashing



- Why does demand paging work?
  Locality model
  - Process migrates from one locality to another as it executes.
  - Localities may overlap.
- Why does thrashing occur?
  Σ size of locality > total memory size

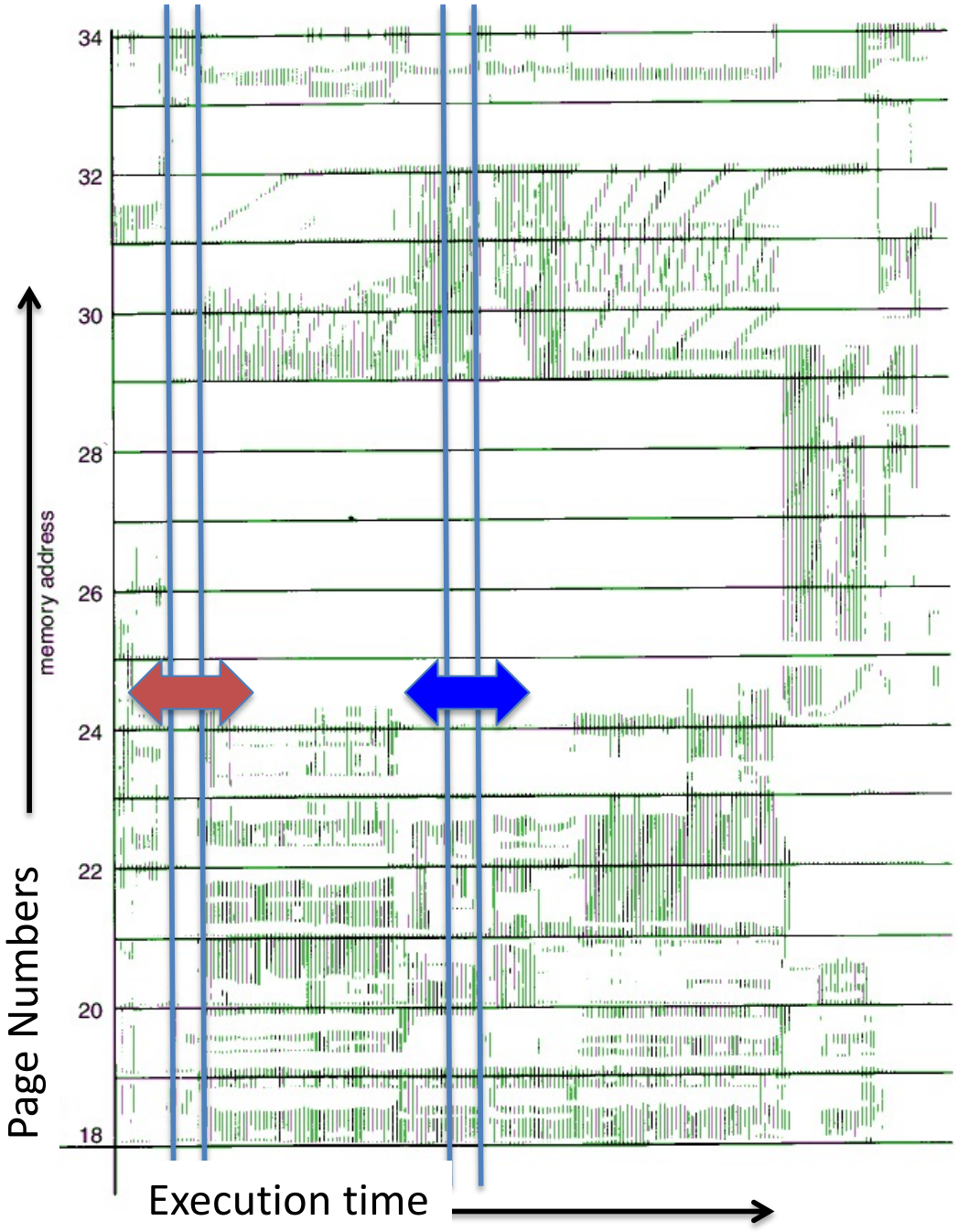
Locality in a Memory-Reference Pattern



Process needs to use only a subset of all the pages for execution at any given time.

Each of these subset of pages defines a locality

Locality in a Memory-Reference Pattern



First time interval, there are few pages that are referenced.

Second time interval, there are many more pages referenced.

Working-Set Model

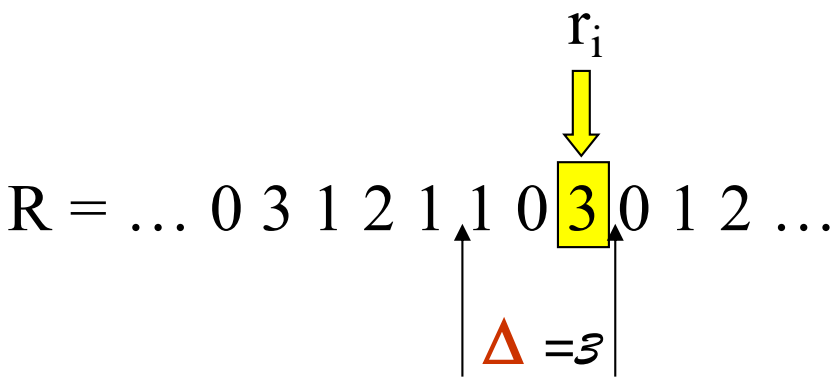
- Working-set model prevents thrashing while keeping the degree of multiprogramming as high as possible → optimizes CPU utilization
- To prevent thrashing, provide a process as many frames as it needs.
- Working set model defines the locality model of process execution.
  - An approximation of the set of pages that the process will access in the future

Working-Set Model

- Peter Denning in 1968 defined the working set model of a process
- Δ (working-set window): time interval from *t* to *t+i*
  - if Δ too small will not encompass entire locality.
  - if Δ too large will encompass several localities.
  - if $\Delta = \infty \Rightarrow$ will encompass entire program.
- WS_i : working set (pages) of process P_i
- WSS_i (working set size of process P_i): total number of pages referenced in the most recent Δ, (varies in time)
- $D = \Sigma\ WSS_i$: total demand of frames
- m : total number of frames (memory capacity)
- if $D > m \Rightarrow$ Thrashing
- Policy: if D > m, then suspend one of the processes.

Working-Set model

The "Window Size" = $\Delta = 3$



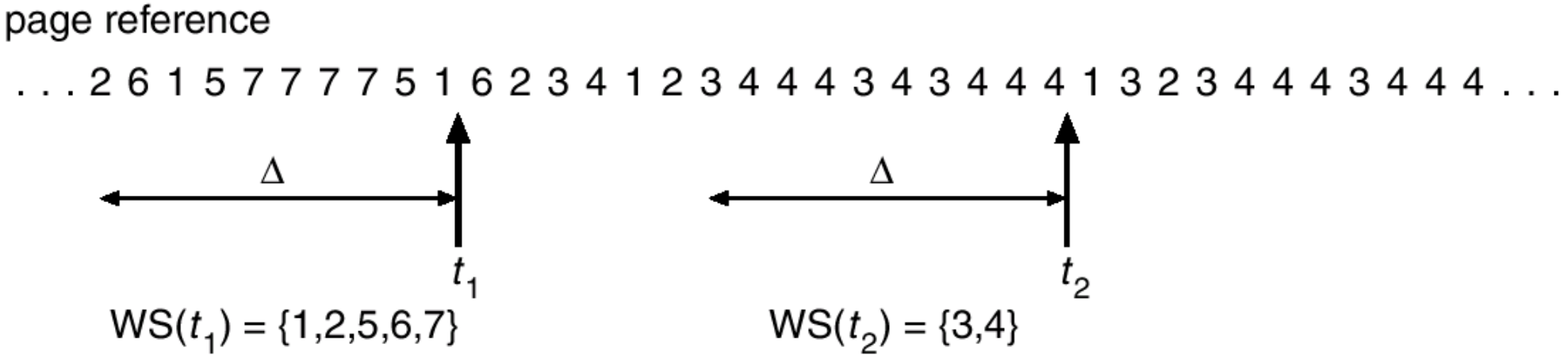
At virtual time i: working set = {0, 1, 3}

At virtual time i-1: working set = {0, 1}

What if $\Delta = 4$?

Working-Set model

$\Delta = 10$



Monitoring Working Set

- The operating system
  - monitors the working set of each process and
  - allocates to that working set enough frames to provide it with its working-set size.
  - If there are enough frames left, it can increase degree of multiprogramming
  - Otherwise, it will suspend some of the processes

Question?

- Is it possible for a process to have two working sets: one representing the data and another representing code?

- Yes, in fact some processors provide two TLBs for this very reason.
  - As an example, the code being accessed by a process may retain the same working set for a long period of time. However, the data the code accesses may change, thus reflecting a change in the working set for data accesses.

Questions – Program Structure

- Assume page size is 1024 words
- **int A[][] = new int[1024][1024];**
- Each row is stored in one page
- If the OS allocates less than 1024 frames for the program
- Program 1

```
for (j = 0; j < A.length; j++)
    for (i = 0; i < A.length; i++)
        A[i][j] = 0;
```

- Program 2

```
for (i = 0; i < A.length; i++)
    for (j = 0; j < A.length; j++)
        A[i][j] = 0;
```

How many page faults occur in each program?

Acknowledgments

- These slides are adapted from
  - Öznur Özkasap (Koç University)
  - Operating System and Concepts (9th edition) Wiley