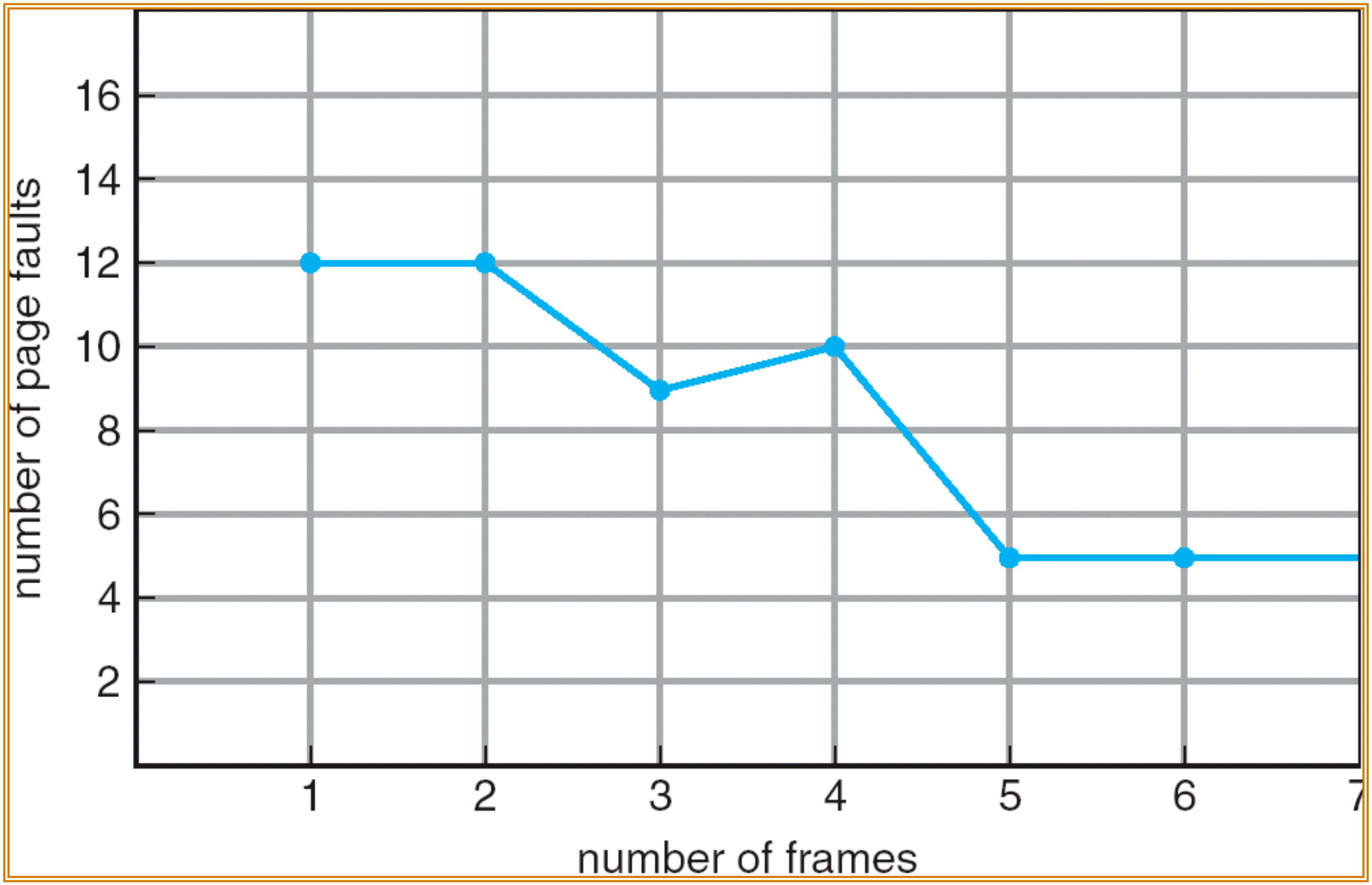
Page Replacement Algorithms

- 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
- In all our examples, the reference string is

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

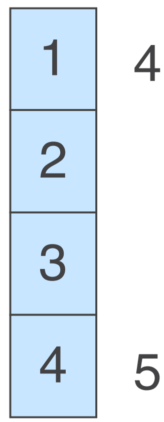
FIFO Illustrating Belady's Anomaly



Optimal Algorithm

- Replace page that will not be used for longest period of time
- 4 frames example

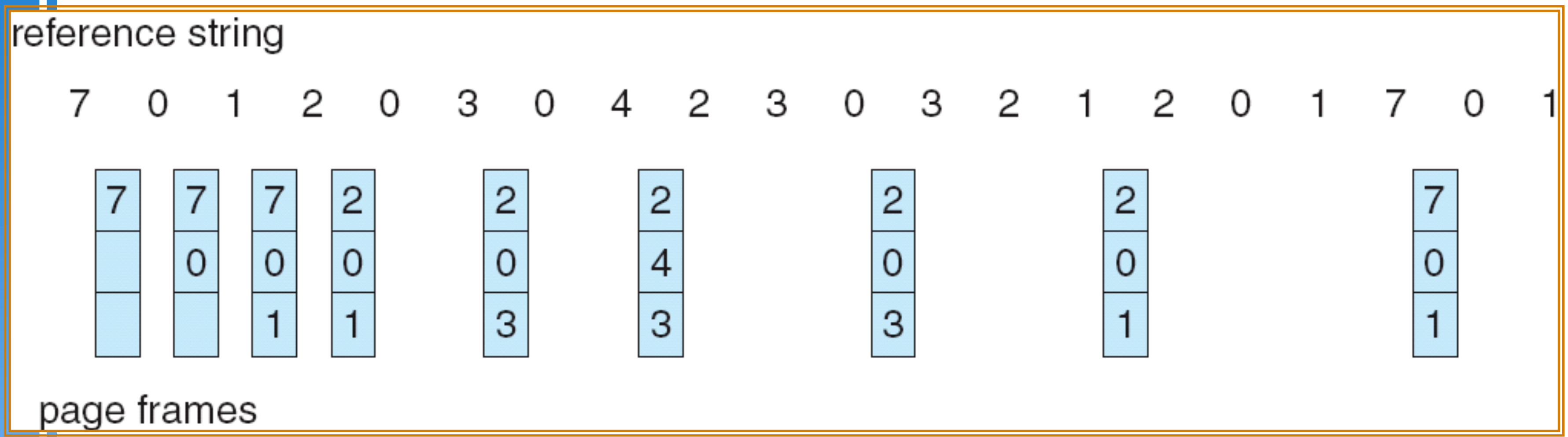
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



6 page faults

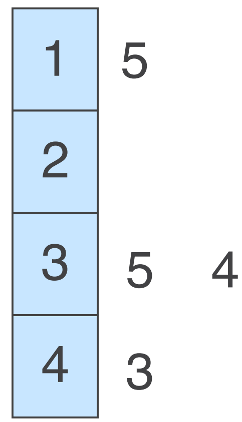
- How do you know this?
- Used for measuring how well your algorithm performs

Optimal Page Replacement



Least Recently Used (LRU) Algorithm

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

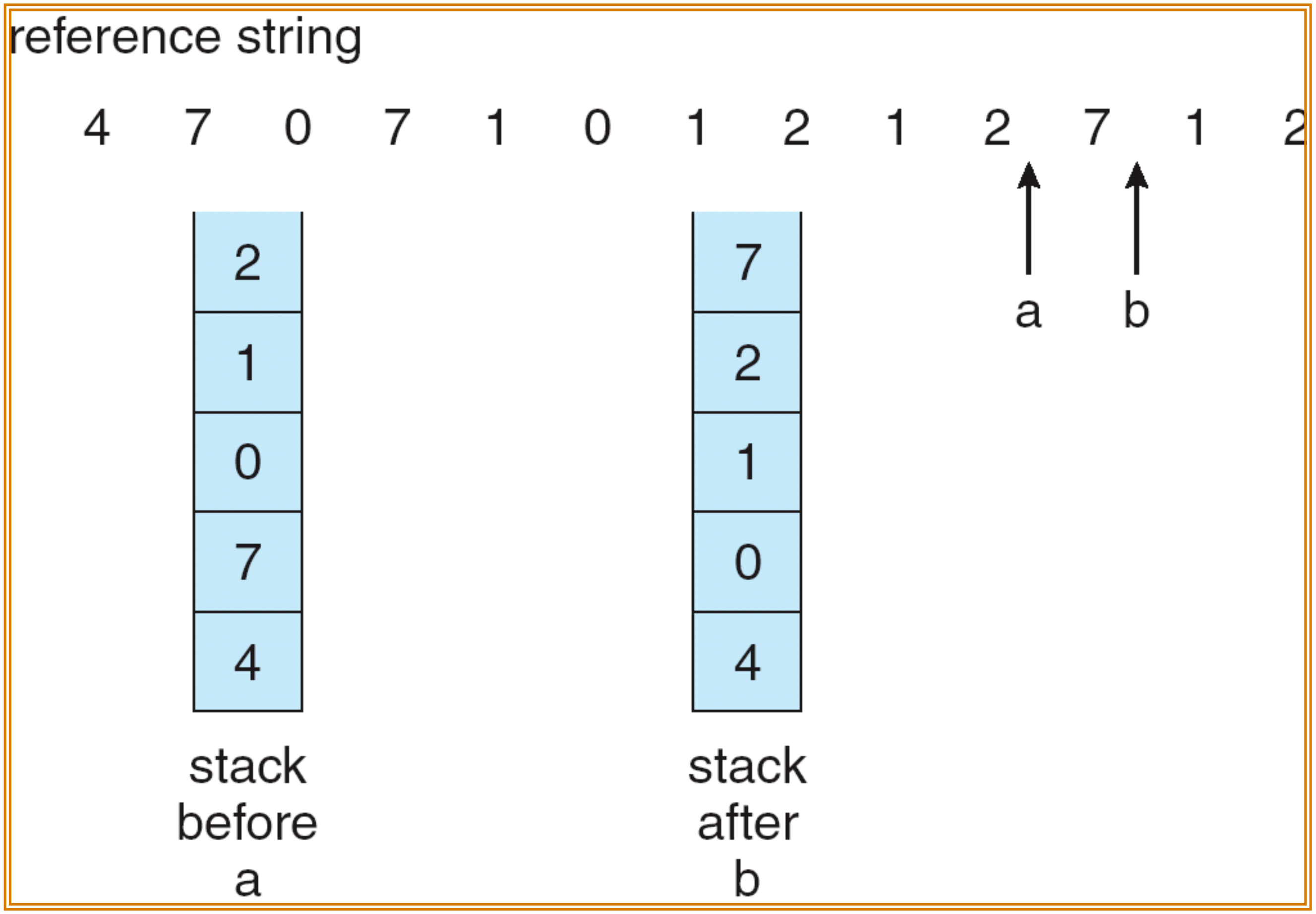


- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to determine which are to change

LRU Algorithm (Cont.)

- 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
  - Unlike counter based approach, does not search for replacement

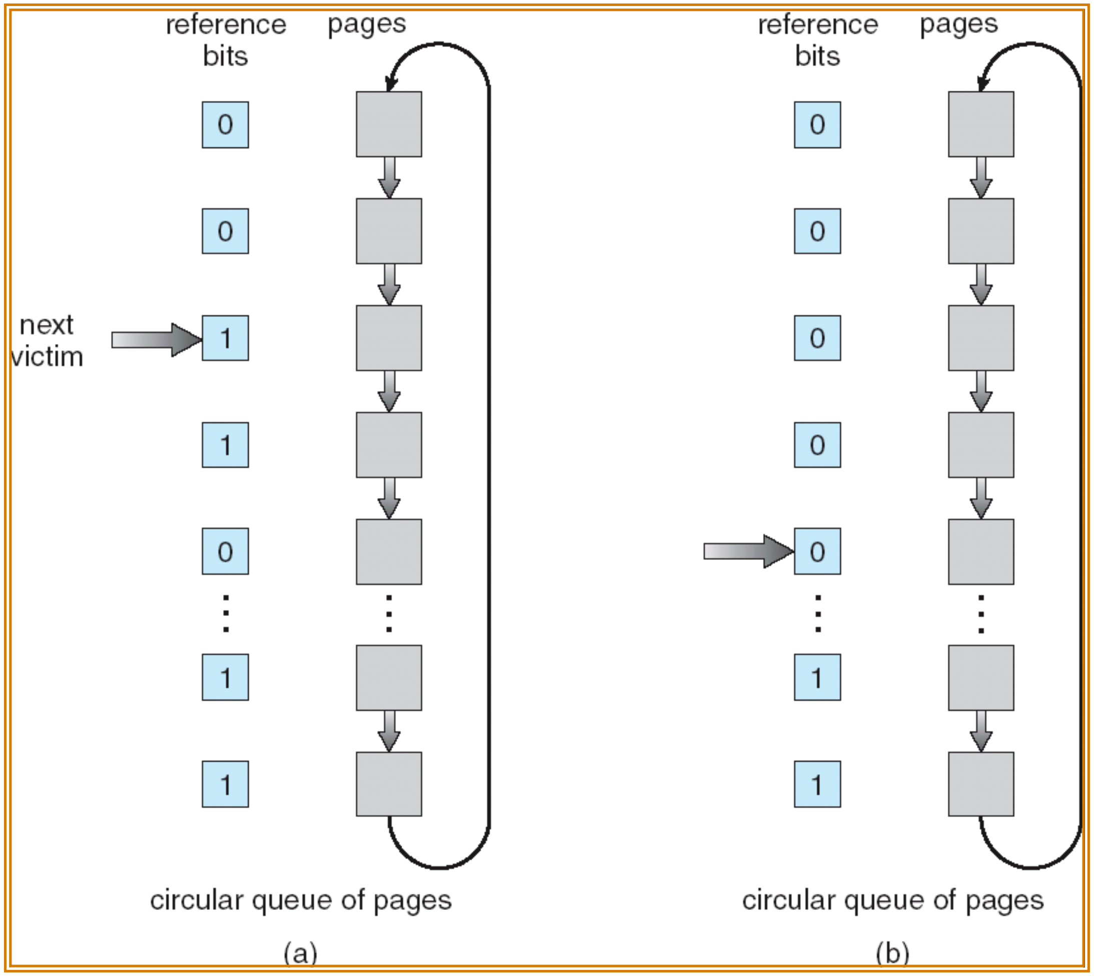
Use Of A Stack to Record The Most Recent Page References



LRU Approximation Algorithms

- Reference bit
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Replace the one which is 0 (if one exists). We do not know the order, however.
- Additional reference bits
  - Hardware sets bit, OS periodically shifts bit
- Second chance
  - Need reference bit
  - Clock replacement
  - FIFO algorithm; if page to be replaced (in clock order) has reference bit = 1 then:
    - set reference bit 0
    - leave page in memory
    - replace next page (in clock order), subject to same rules

Second-Chance (clock) Page-Replacement Algorithm



- Enhanced second-chance (reference & modified bit)

Counting Algorithms

- Keep a counter of the number of references that have been made to each page
- LFU Algorithm: replaces page with smallest count. One problem is that pages that were active a long time back may survive. Can use a policy that shifts the counter periodically.
- MFU Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Page buffering algorithms

- Maintain a pool of free-frames
  - If page needs to be written to disk, allocate a page from free pool, and once the write completes return that page to the free pool
- List of modified files and when idle, write contents to disk and reset modified bit
- Move pages to free-list, but if process needs that page again, move it from free to active list

Allocation of Frames

- How should the OS distribute the frames among the various processes?
- Each process needs *minimum* number of pages - at least the minimum number of pages required for a single assembly instruction to complete
- 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.

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

- Proportional allocation – Allocate according to the size of process

$$m = 64$$
$$s_i = 10$$
$$s_2 = 127$$
$$a_1 = \frac{10}{137} \times 64 \approx 5$$
$$a_2 = \frac{127}{137} \times 64 \approx 59$$

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

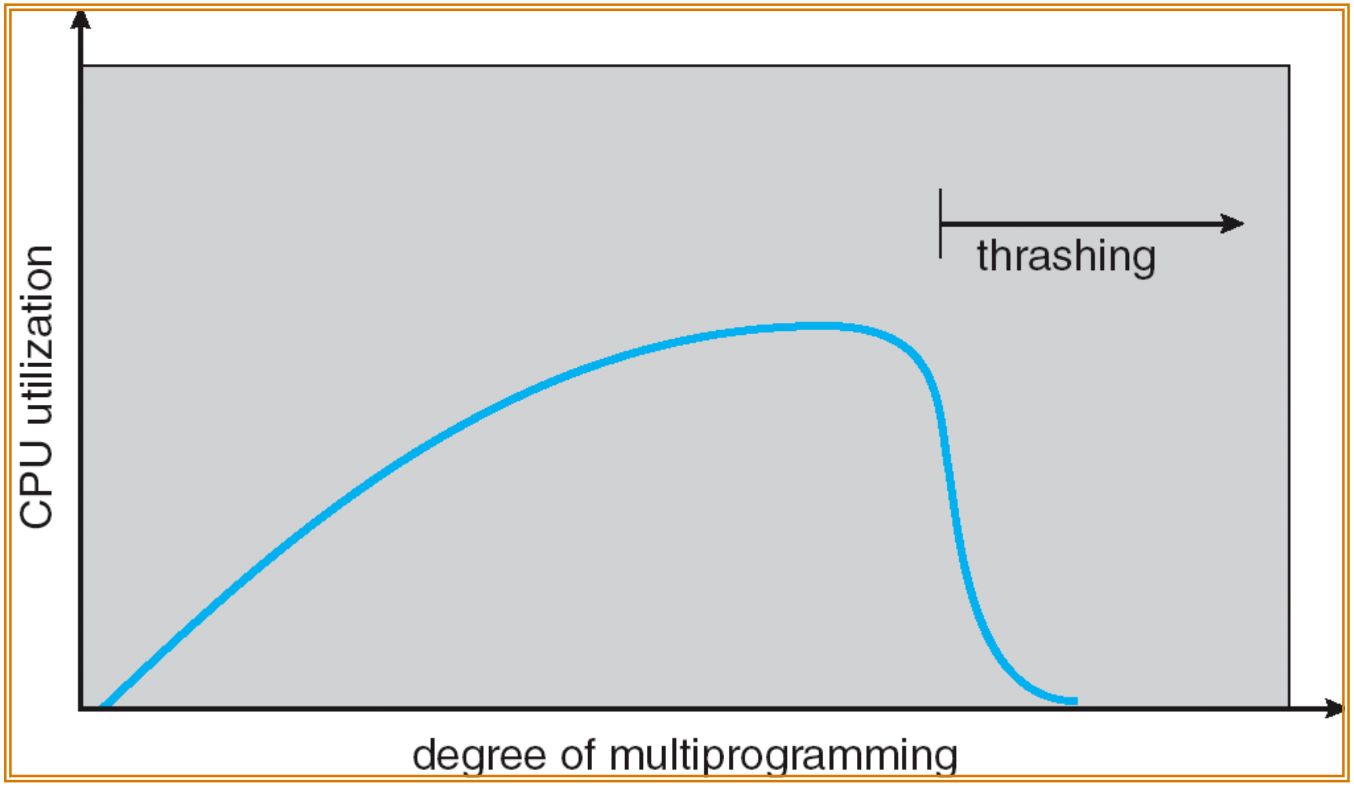
Global vs. Local Allocation

- Global replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another
  - It is possible for processes to suffer page faults through no fault of theirs
  - However, improves system throughput
- Local replacement – each process selects from only its own set of allocated frames
  - May not use free space in the system

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 because of low cpu utilization
  - another process added to the system
- Thrashing ≡ a process is busy swapping pages in and out

Thrashing (Cont.)



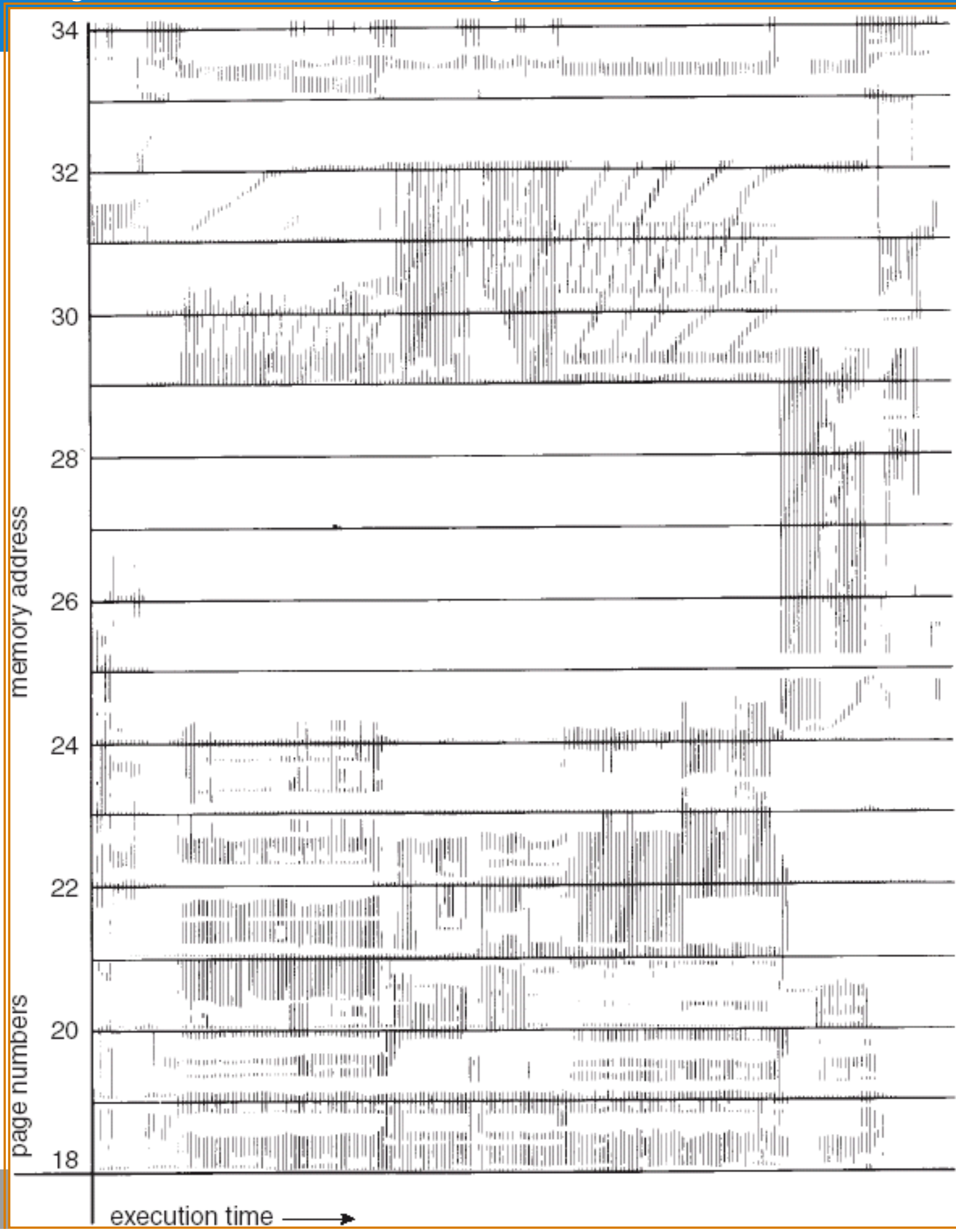
Demand Paging and Thrashing

- Why does demand paging work?
  Locality model
  - Process migrates from one locality to another
  - Localities may overlap
  - E.g.

```
for (……) {
    computations;
}
for (….. ) {
    computations;
}
```

- Why does thrashing occur?
  Σ size of locality > total memory size

Locality In A Memory-Reference Pattern

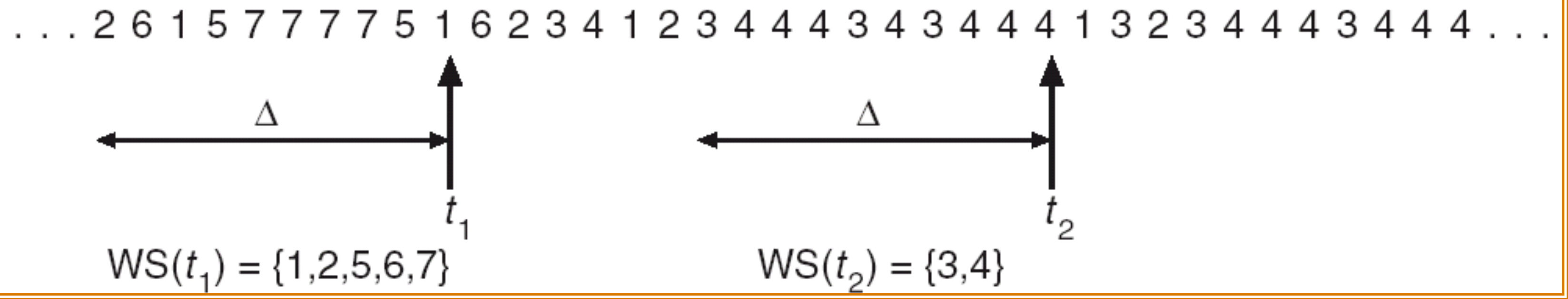


Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  Example: 10,000 instruction
- WSS_i (working set of Process P_i) = total number of pages referenced in the most recent Δ (varies in time)
  - 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
- $D = \Sigma\, WSS_i \equiv$ total demand frames
- if $D > m \Rightarrow$ Thrashing
- Policy if D > m, then suspend one of the processes

Working-set model

page reference table



Keeping Track of the Working Set

- Approximate with interval timer + a reference bit
- Example: Δ = 10,000
  - Timer interrupts after every 5000 time units
  - Keep in memory 2 bits for each page
  - Whenever a timer interrupts copy and sets the values of all reference bits to 0
  - If one of the bits in memory = 1 $\Rightarrow$ page in working set
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units

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