# CS343: Operating System

# Virtual Memory, Demand Paging, Page Replacement

Lect34 : 2nd Nov 2023

Dr. A. Sahu

Dept of Comp. Sc. & Engg.

Indian Institute of Technology Guwahati

# Outline

- Memory Management
  - Paging
  - Virtual memory
  - Demand Paging
  - **Page Replacement**
  - **Frame Allocation**

# First-In-First-Out (FIFO) Algorithm

- Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 frames (3 pages can be in memory at a time per process)

| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 7 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 7 | 7 |
|  | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
|  |  | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 1 |

15 page faults

# Optimal Algorithm

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

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

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 7 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 7 | 7 | 7 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

9 page faults

# LRU Algorithm

| 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 7 | 7 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 0 |
| | | 1 | 1 | 1 | 3 | 3 | 3 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 7 | 7 | 7 |

- 12 faults – better than FIFO (15) but worse than OPT (9)
- Generally good algorithm and frequently used
- But how to implement?

# LRU Algorithm Cont.

- 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 find smallest value
    - **Search through table needed : Bad**

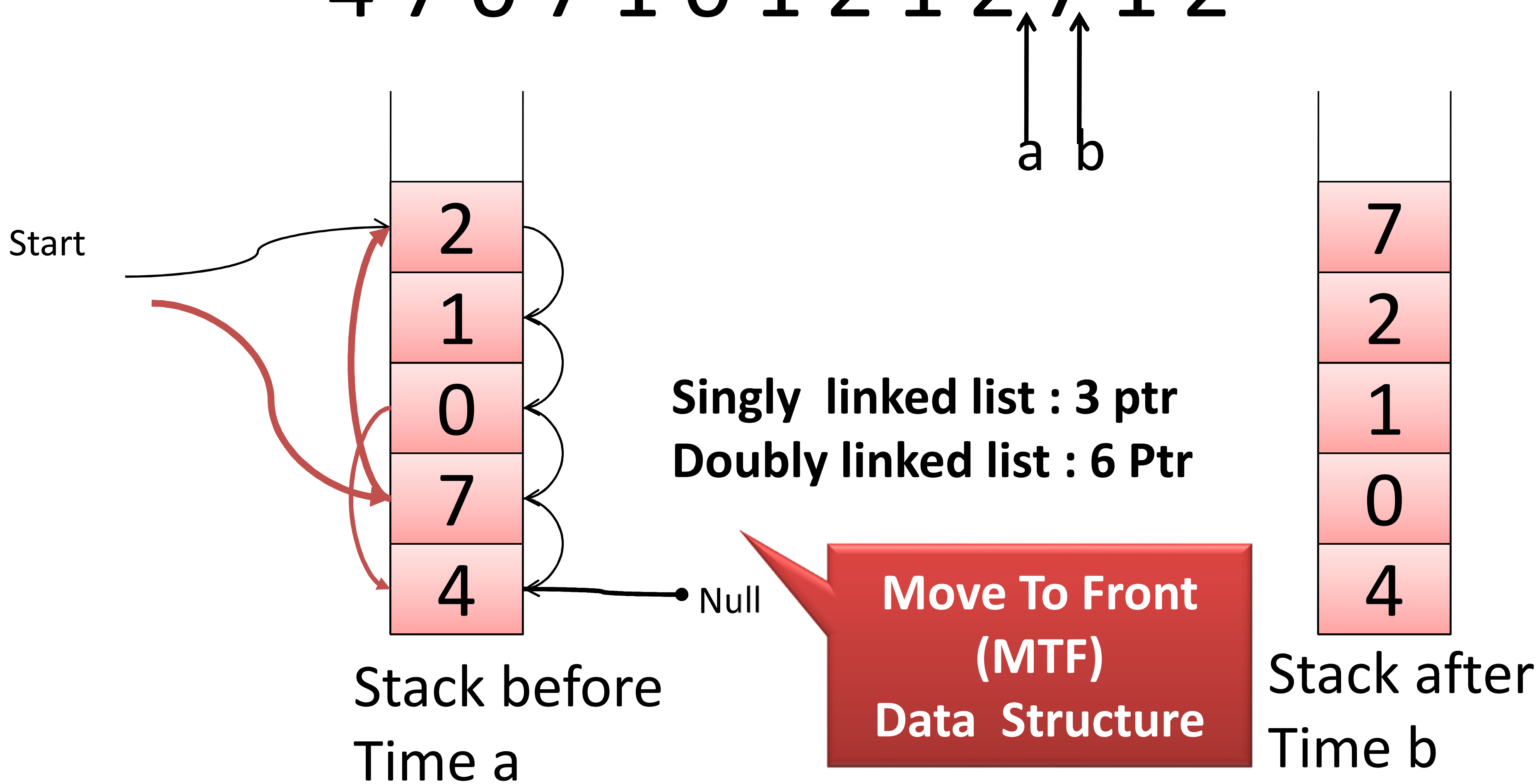
# LRU Algorithm Cont.

- 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 (OK)**
  - But each update more expensive
  - No search for replacement
- LRU and OPT are cases of **stack algorithms** that don't have Belady's Anomaly

# Use Of A Stack to Record Most Recent Page References

Reference string

4 7 0 7 1 0 1 2 1 2 7 1 2



# LRU Approximation Algorithms

- LRU needs special hardware and still slow
- **Reference bit**
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Replace any with reference bit = 0 (if one exists)
    - We do not know the order, however

# LRU Approximation Algorithms

- Suppose every page associated with 8 bits reference bit (with a shift reg)
- When a reference made bit set to 1
- Every page reference bit shifted
- For last 8 references
  - Every page will get different value of shift reg
  - 00000000 in a Page : signifies no reference
  - 11111111 in a Page : signifies reference all the time
  - 10010101 is LRUed then 00101000
- **All page Shift reg need to shift : Bad Part**

# LRU Approximation Algorithms

- **Second-chance algorithm**
  - Generally FIFO, plus hardware-provided reference bit and
  - **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

# LRU Approximation Algorithms

- **Second-chance algorithm with modify bit**
- 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
- May be combined with modified bit
  - Ref bit (0) + Modified (0) ==> best guy to replace
  - 01 ➔ not recently used but modified ➔ not good
  - 10 => recently used but clean => probably will be used again
  - 11 ➔ used and modfied ➔ bad candidate

# Counting Algorithms

- Keep a counter of the number of references that have been made to each page
  - Not common
- **Lease Frequently Used** (**LFU**) **Algorithm**
  - Replaces page with smallest count
- **Most Frequently Used** (**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

- Keep a pool of free frames, always
  - Then frame available when needed, not found at fault time
  - Read page into free frame and select victim to evict and add to free pool
  - When convenient, evict victim

# Page-Buffering Algorithms

- Possibly, keep list of modified pages
  - When backing store idle, write pages there and set to non-dirty
- Possibly, keep free frame contents intact and note what is in them
  - If referenced again before reused, no need to load contents again from disk
  - Generally useful to reduce penalty if wrong victim frame selected
  - **Same as Victim Buffer**

# Applications and Page Replacement

- All of these algorithms have OS guessing about future page access
- Some applications have better knowledge – i.e. databases

# Applications and Page Replacement

- Memory intensive applications can cause double buffering
  - OS keeps copy of page in memory as I/O buffer
  - Application keeps page in memory for its own work
- OS can given direct access to the disk, getting out of the way of the applications
  - **Raw disk** mode
- Bypasses buffering, locking, etc

# Page Replacement Algorithm

- Input : Reference string, number of frame
- Output: number of page fault

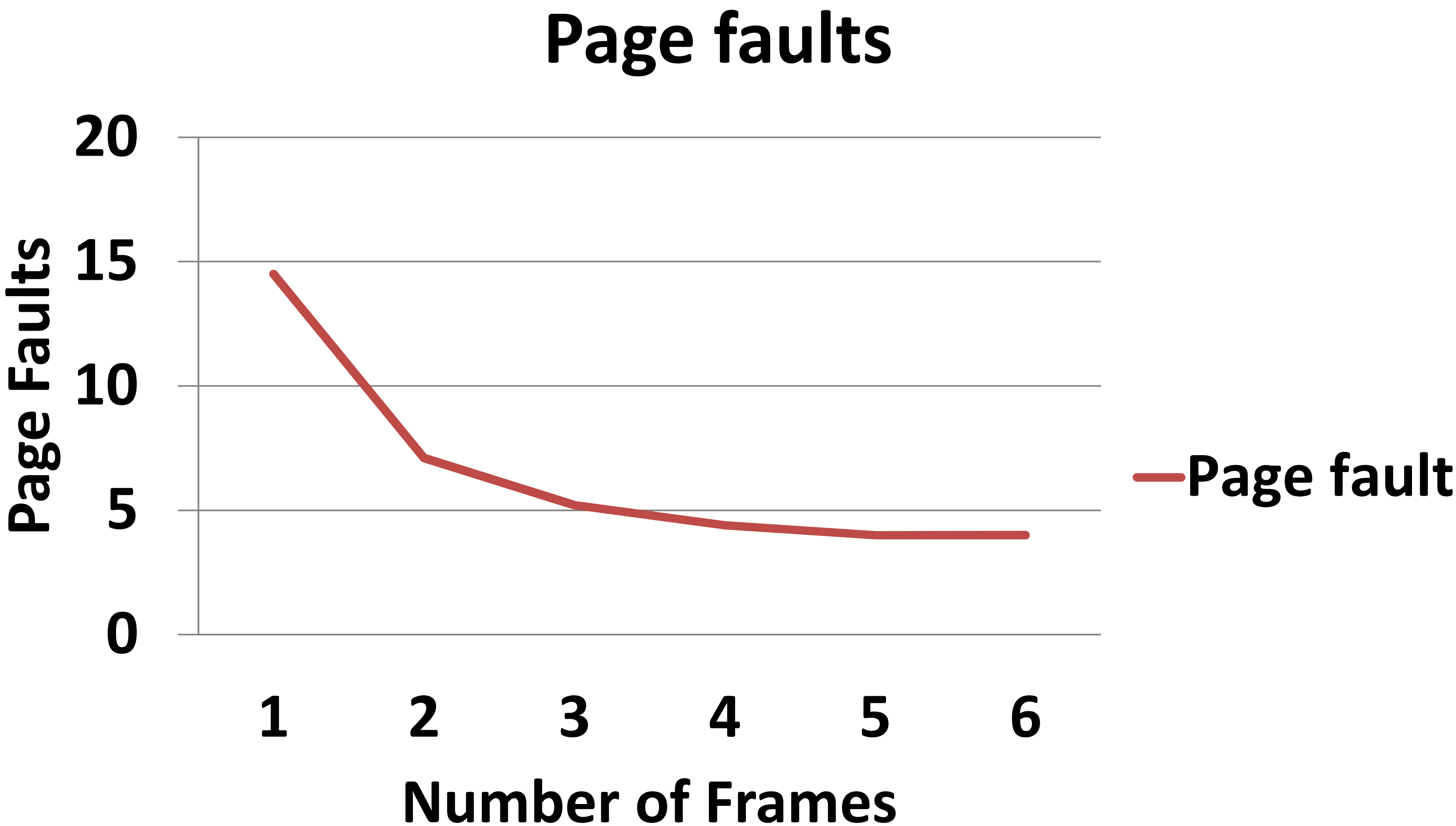
- **Timing information was missing in Reference string**
- **How to allocate the frame for process?**
  - **Static (fixed size for process life time)**
  - **Dynamic**

# Allocation of Frames

# Allocation of Frames

- Each process needs ***minimum*** number of frames
- 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*

# Page Faults Versus The Number of Frames



# Allocation of Frames

- ***Maximum*** of course is total frames in the system
- Two major allocation schemes
  - fixed allocation
  - priority allocation
- Many variations

# Fixed Allocation

- Equal allocation
  - Allocate equal number of frame to each process
  - For example, if there are 100 frames (after allocating frames for the OS) and 5 processes, give each process 20 frames
  - Keep some as free frame buffer pool
- Proportional allocation
  - Allocate according to the size of process
  - Dynamic as degree of multiprogramming, process sizes change

# Fixed Allocation

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

$$S = \sum s_i$$

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

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

$$m = 64$$

$$s_1 = 10$$

$$s_2 = 127$$

$$a_1 = \frac{10}{137} \times 62 \approx 4$$

$$a_2 = \frac{127}{137} \times 62 \approx 57$$

# Priority Allocation Based on Fault Rate

- 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

**Cache Friendly and Cache Thrashing Apps**

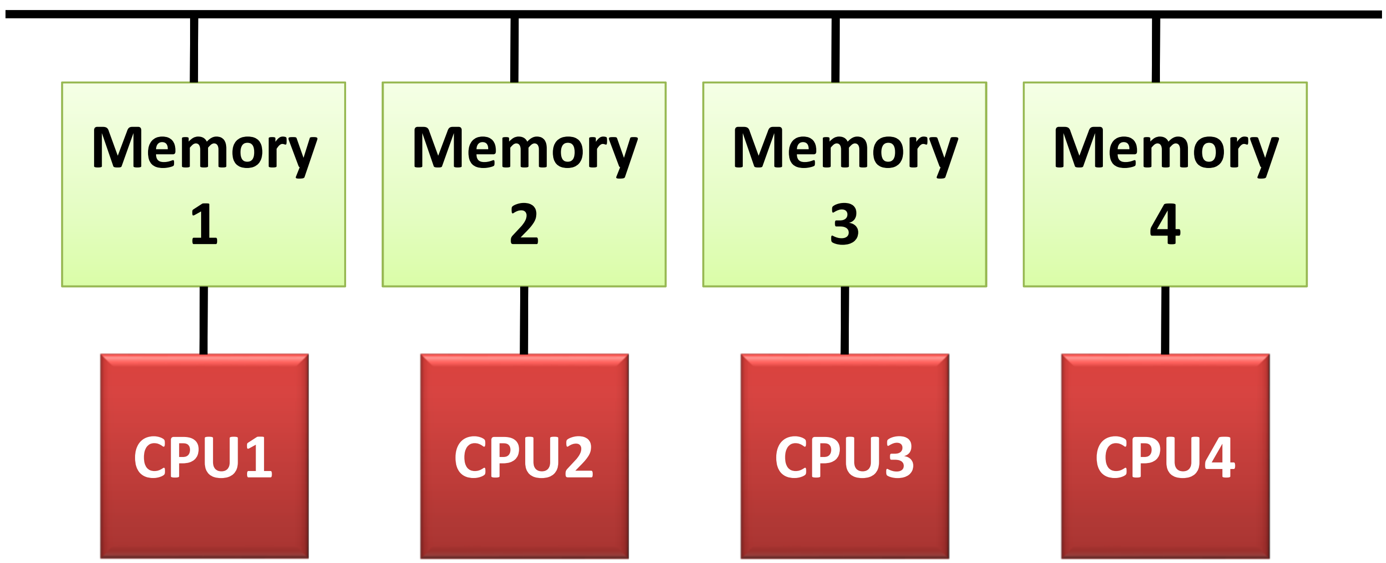
1. Reading a data with Temporal locality (same data reference many time)
2. Reading data in streaming manner, referred one time

# 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
  - But then process execution time can vary greatly
  - But greater throughput so more common
  - Interference : Interferer, Sufferer, Pinning/Quota
- **Local replacement** – each process selects from only its own set of allocated frames
  - More consistent per-process performance
  - But possibly underutilized memory

# Non-Uniform Memory Access

- So far all memory accessed equally
- Many systems are **NUMA** – speed of access to memory varies
  - Consider system boards containing CPUs and memory, interconnected over a system bus



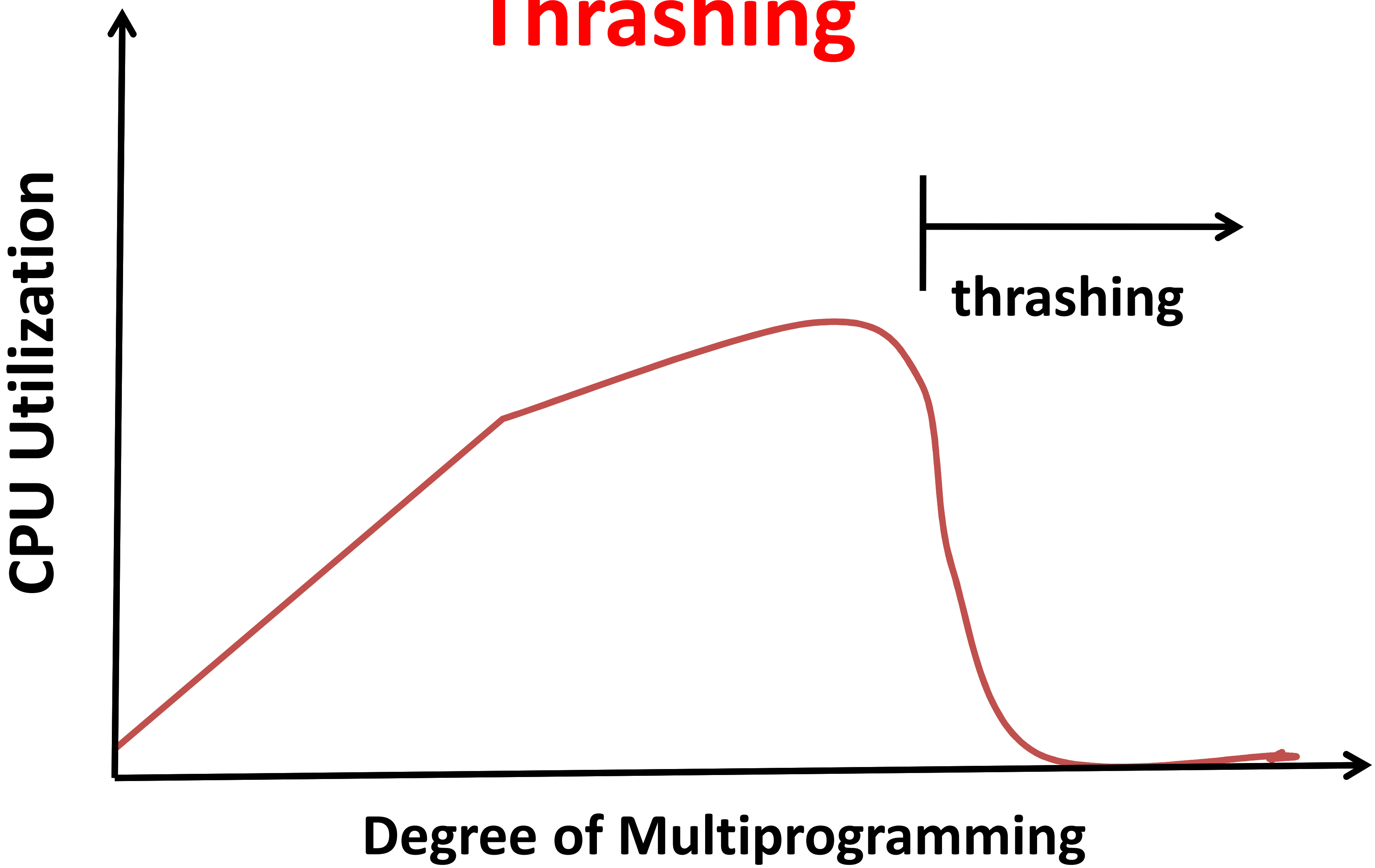
# Non-Uniform Memory Access

- Optimal performance comes
  - Allocating memory “close to” the CPU on which the thread is scheduled
- And modifying the scheduler to schedule the thread on the same system board when possible
- Solved by Solaris by creating **lgroups**
  - Structure to track CPU / Memory low latency groups
  - Used my schedule and pager
  - When possible schedule all threads of a process and allocate all memory for that process within the lgroup

# Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high
  - Page fault to get page
  - Replace existing frame
  - But quickly need replaced frame back
  - This leads to:
    - Low CPU utilization
    - Operating system thinking 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




(Number of Frame per process) is inversely proportional to (Degree of Multiprogramming )

# 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?

  **$\Sigma$ size of locality > total memory size**

  – Limit effects by using local or priority page replacement

  – Modeling locality by **Working Set Window**

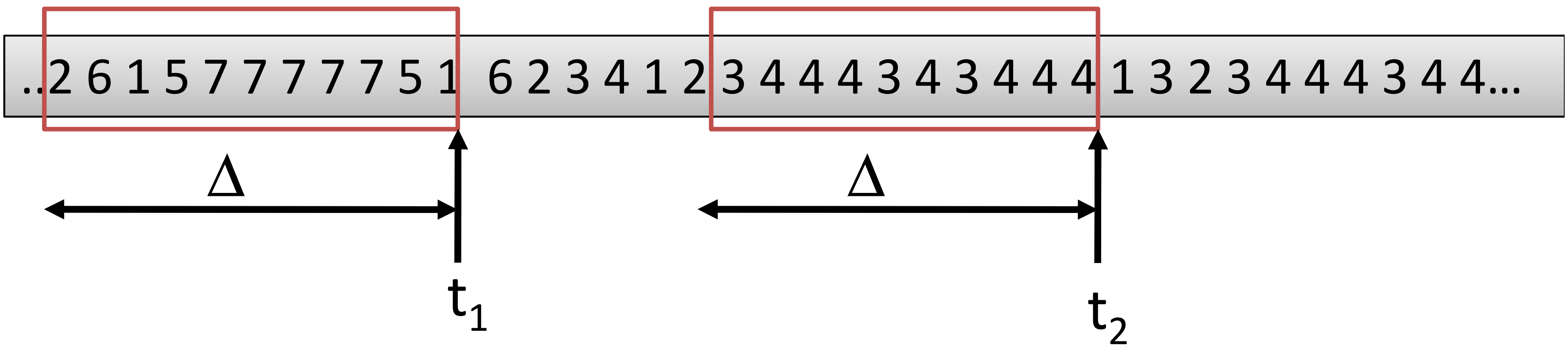
# Working-Set Model

- Working-set window ($\Delta$) : A fixed number of page references

  Or example: 10,000 instructions
- $WSS_i$ (working set of Process $P_i$) =
  total number of pages referenced in the most recent $\Delta$ (varies in time)
  - if $\Delta$ too small will not encompass entire locality
  - if $\Delta$ too large will encompass several localities
  - if $\Delta = \infty \Rightarrow$ will encompass entire program

# Working-Set Model

Page Reference String




$WS(t_1)=\{1,2,5,6,7\}$ $WS(t_2)=\{3,4\}$

# Working-Set Model

- $D = \Sigma\, WSS_i \equiv$ total demand frames
  - Approximation of locality
- Number of frames in memory : m
- if $D > m \Rightarrow$ **Thrashing**
- Policy if $D$ > m, then suspend or swap out one of the processes

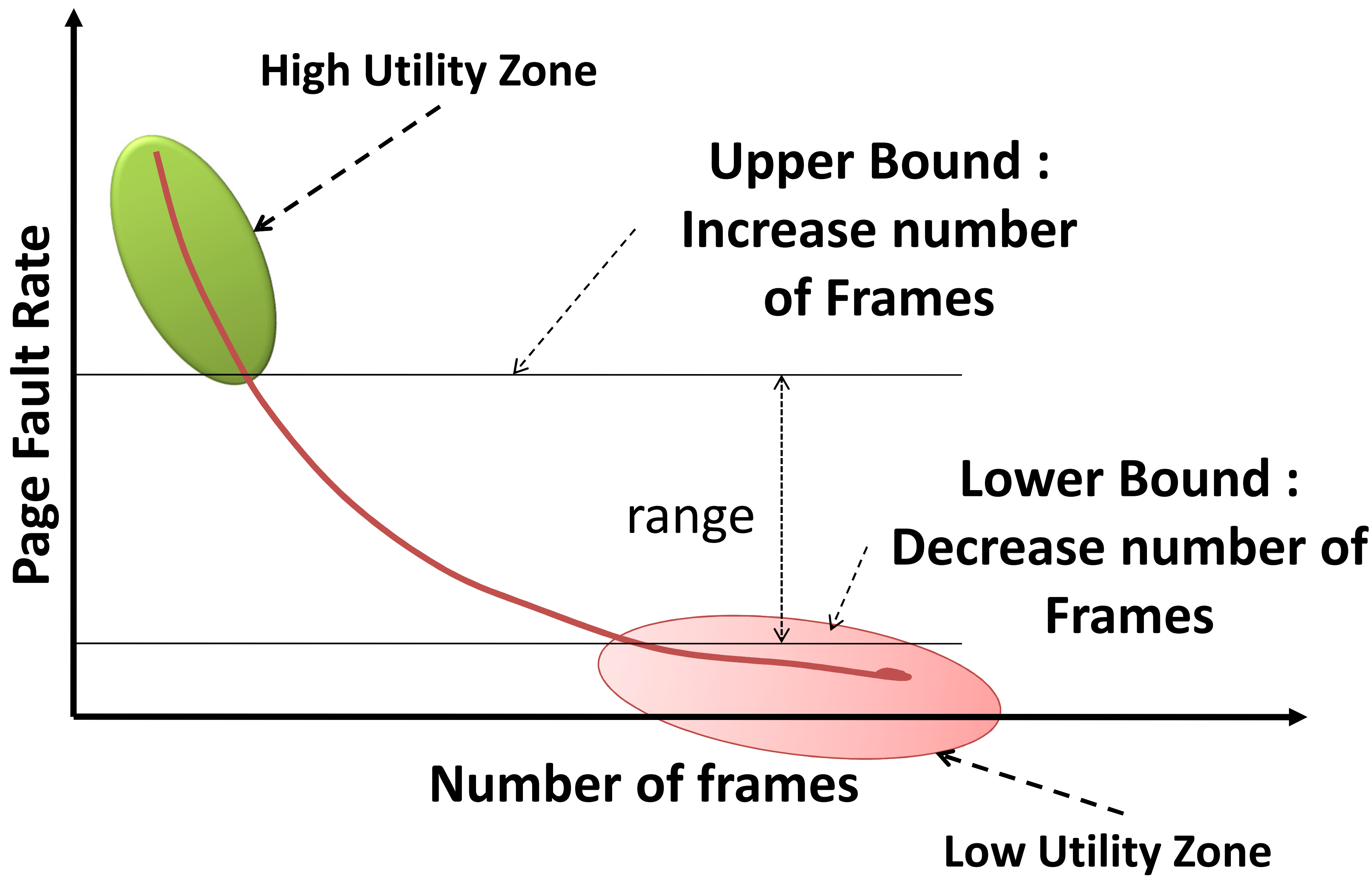
# Keeping Track of the Working Set

- Approximate with interval timer + a reference bit
- Example: $\Delta = 10{,}000$
  - 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

- More direct approach than WSS
- Utility based frame allocation
  - How much is getting utilized by allocating extra frames?
  - If Page fault is getting lower then beneficial
- Establish “acceptable” **page-fault frequency** (**PFF**) rate and use local replacement policy
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame

# Page-Fault Frequency



# Working Sets and Page Fault Rates

- Direct relationship between working set of a process and its page-fault rate
- Working set changes over time
- Peaks and valleys over time