Virtual Memory

Dr. Manmath N. Sahoo
Dept. of CSE, NIT Rourkela

Virtual Memory

- **Virtual Memory** is a technique that allows the execution of processes that are not completely in main memory.

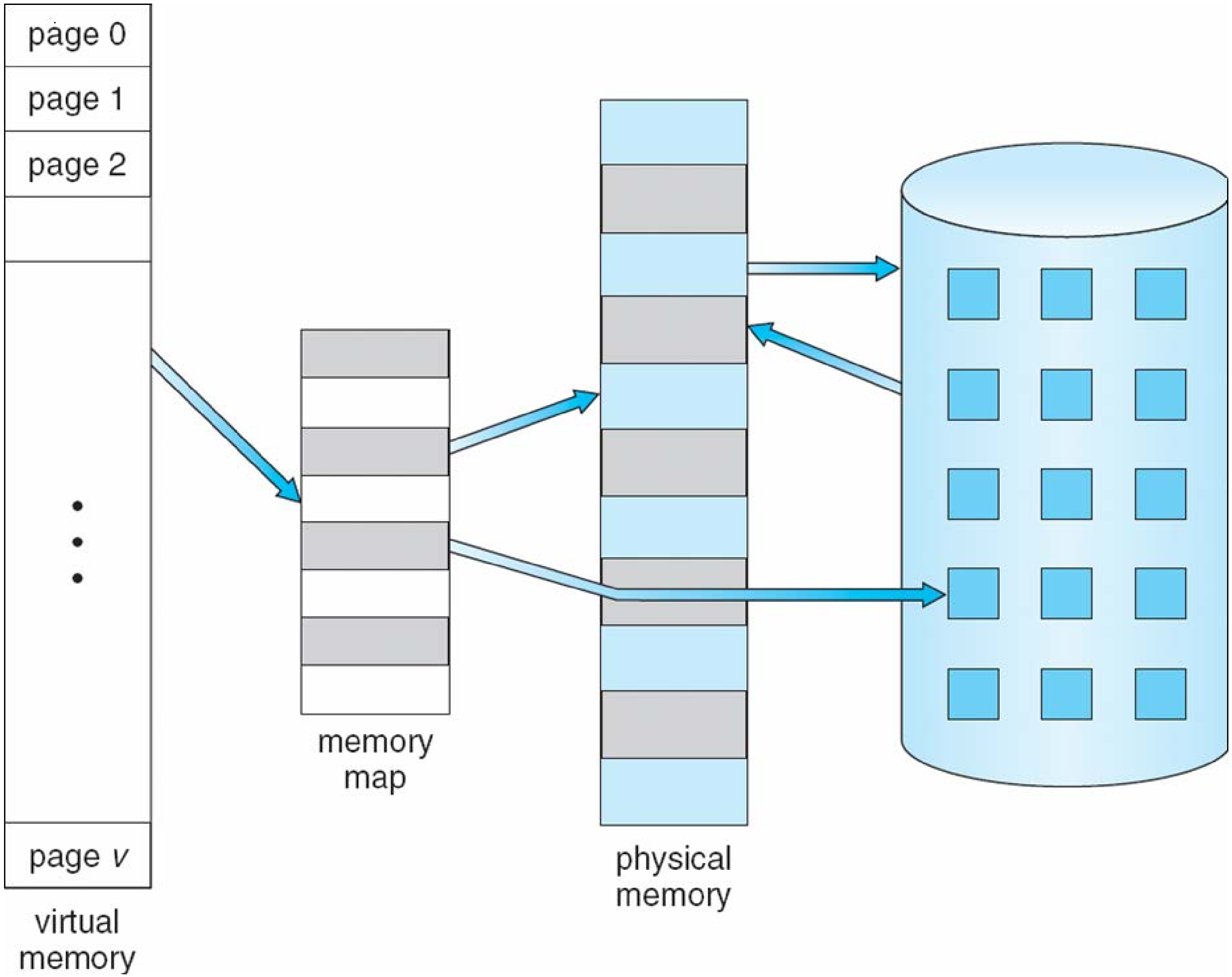
```
If(condition){
 ----
|    |
 ----
|    |
 ----
}
Else{
 ----
|    |
 ----
|    |
 ----
}
```

- **Motivation**
  - Conditional execution:
    - either 2 pages of **if** block will be loaded or 2 pages of **else** block.
  - Error handling codes in programs are seldom executed.
  - Even though all the pages are needed, they are not needed at the same time.

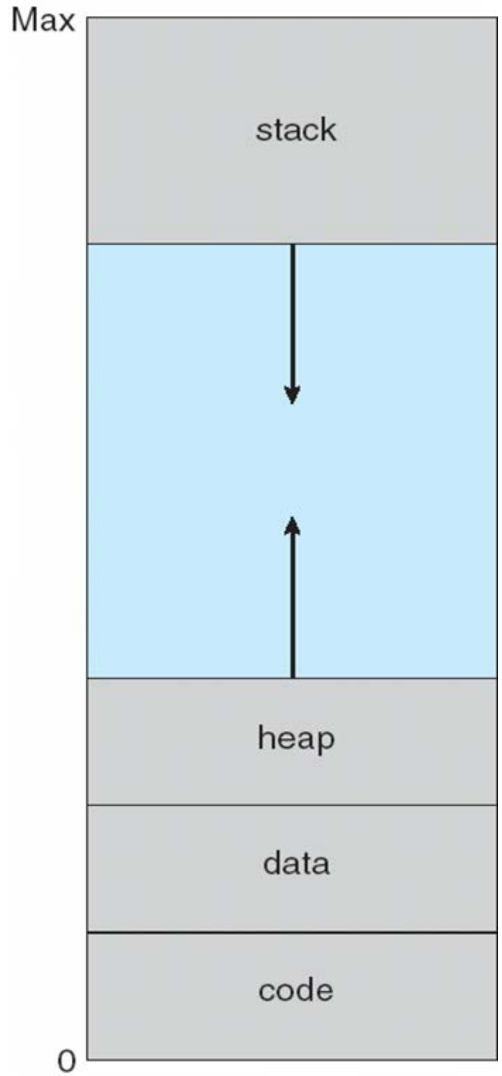
Virtual Memory

- Advantages:
  - A process would no longer be constrained by the amount of available physical memory.
  - Increases degree of multi-programming.
  - Better CPU utilization and throughput.
  - Less I/O overhead

Virtual Memory That is Larger Than Physical Memory



Virtual-address Space



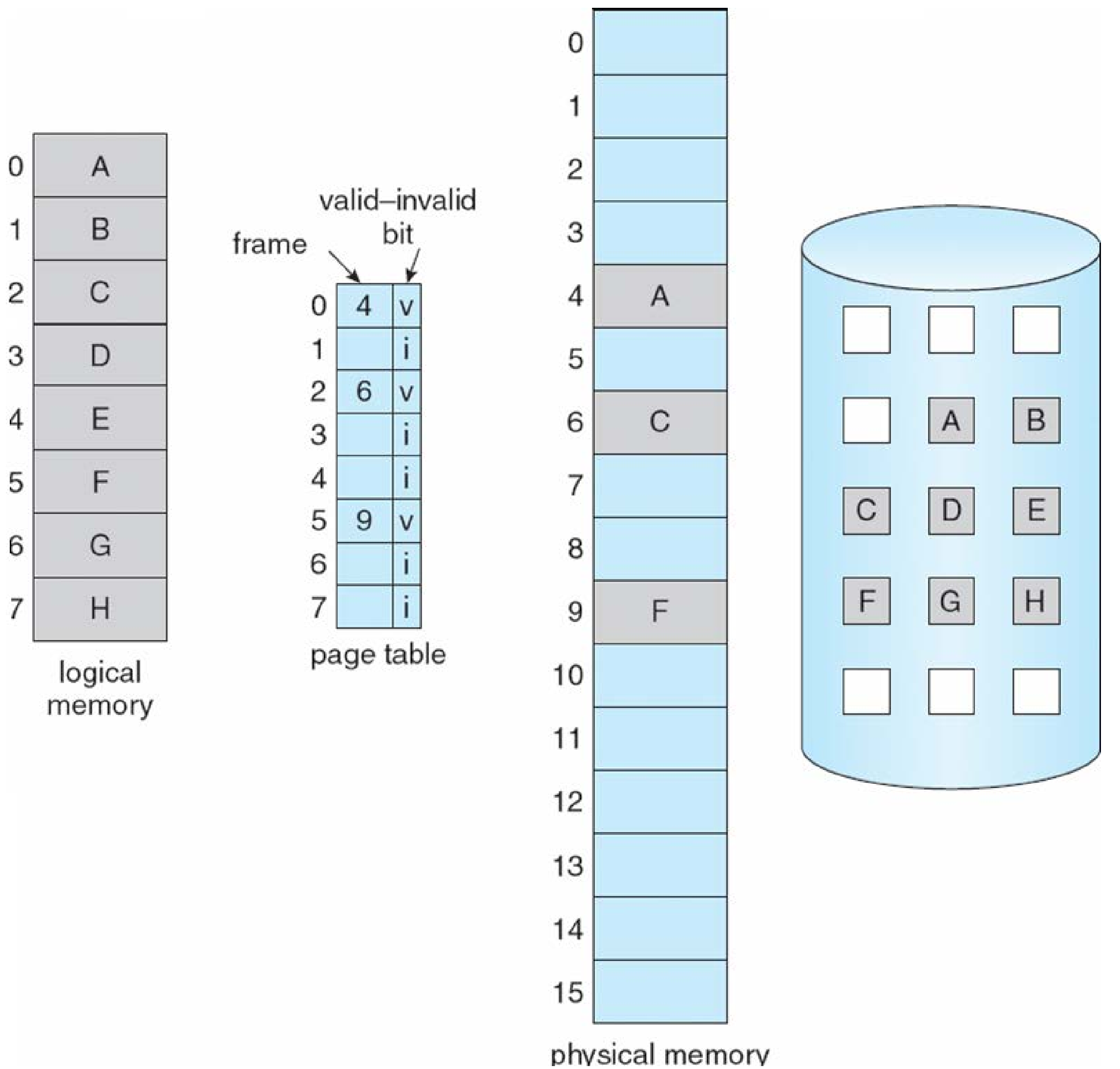
Demand Paging

- Bring a page into memory only when it is needed.
- Page is needed $\Rightarrow$ reference to it
  - invalid reference (not-in-memory) $\Rightarrow$ bring to memory

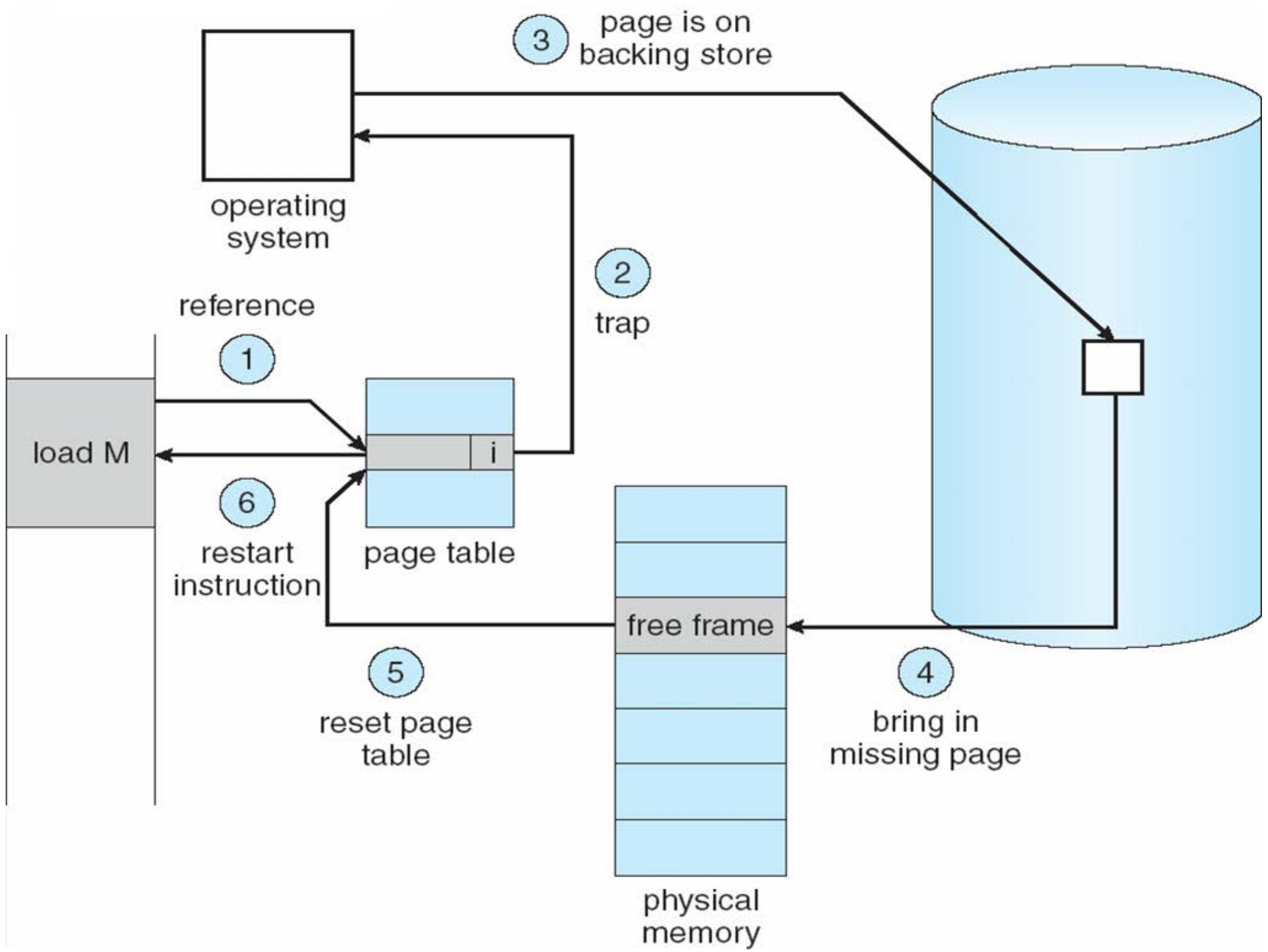
Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated (**1** $\Rightarrow$ in-memory, **0** $\Rightarrow$ not-in-memory)
- Initially valid–invalid bit is set to **0** on all entries
- During address translation, if valid–invalid bit in page table entry is **0** $\Rightarrow$ page fault

Page Table When Some Pages Are Not in Main Memory



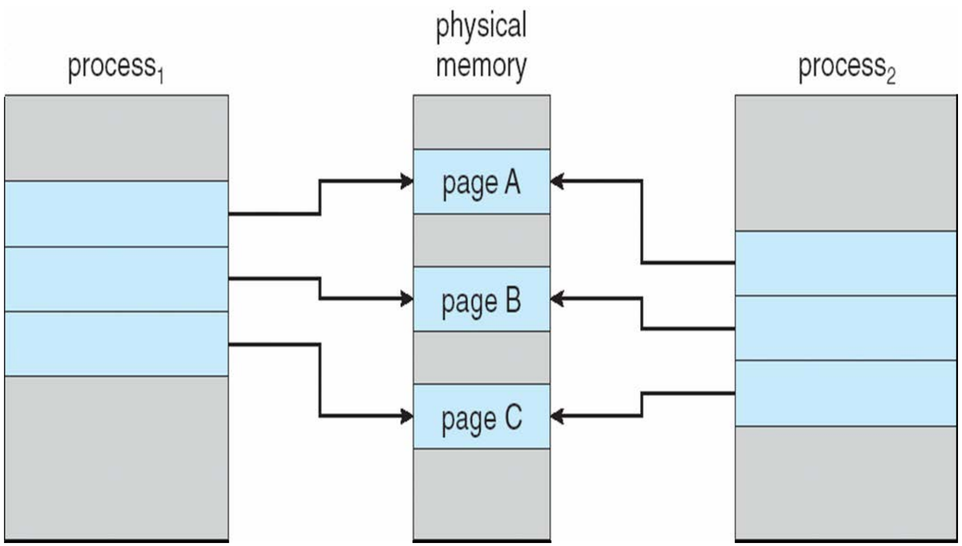
Steps in Handling a Page Fault



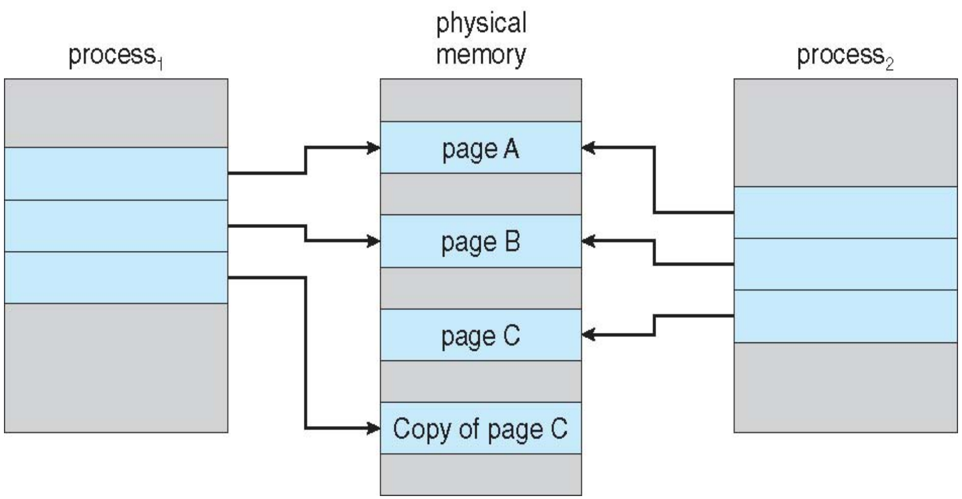
Copy-on-Write

- Copy-on-Write allows both parent and child processes to initially *share* the same pages in memory
- If either process modifies a shared page, only then is the page copied
- It allows more efficient process creation as only modified pages are copied

Copy-on-Write



Before Process 1 Modifies Page C

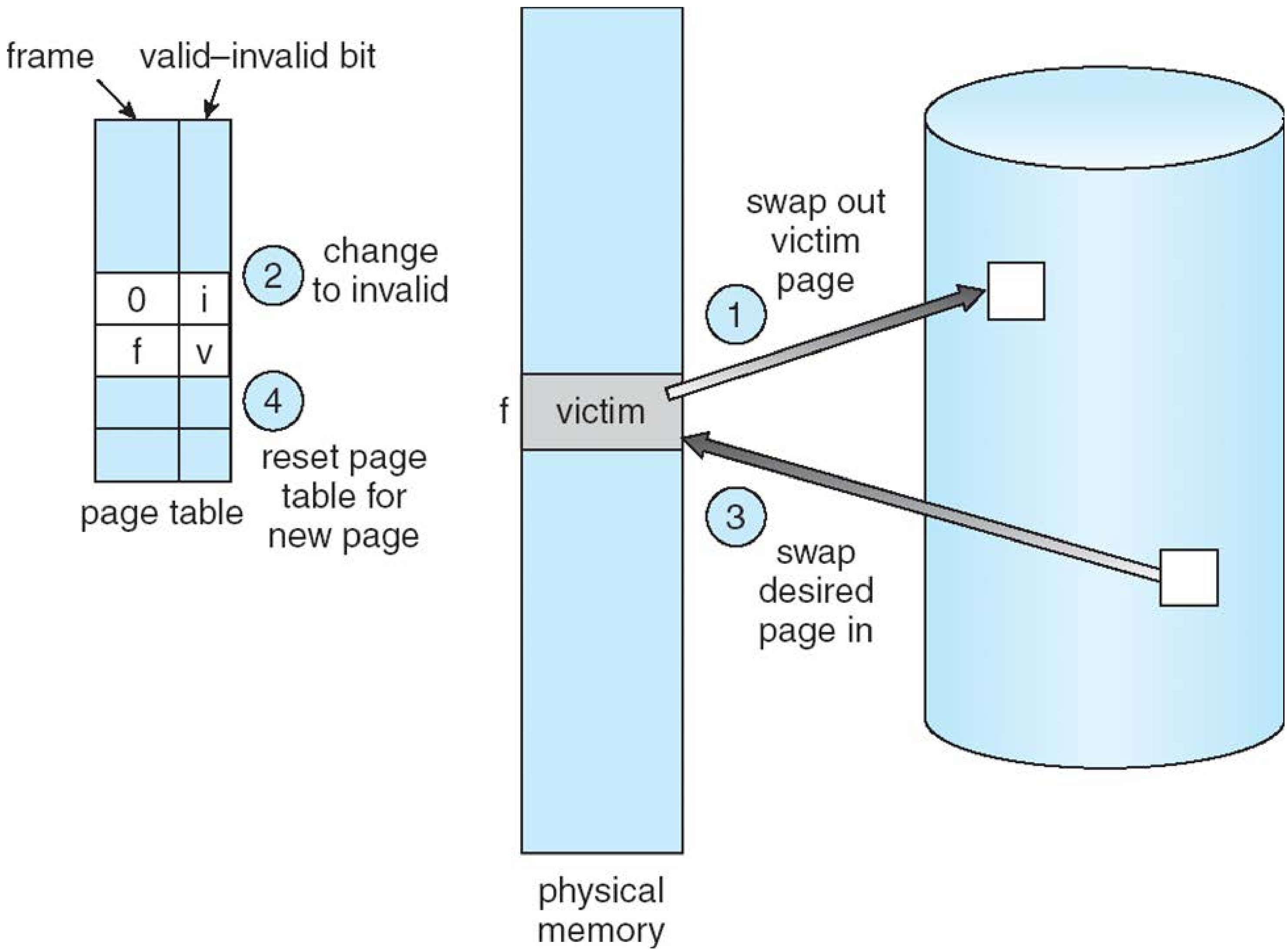


After Process 1 Modifies Page C

What happens if there is no free frame?

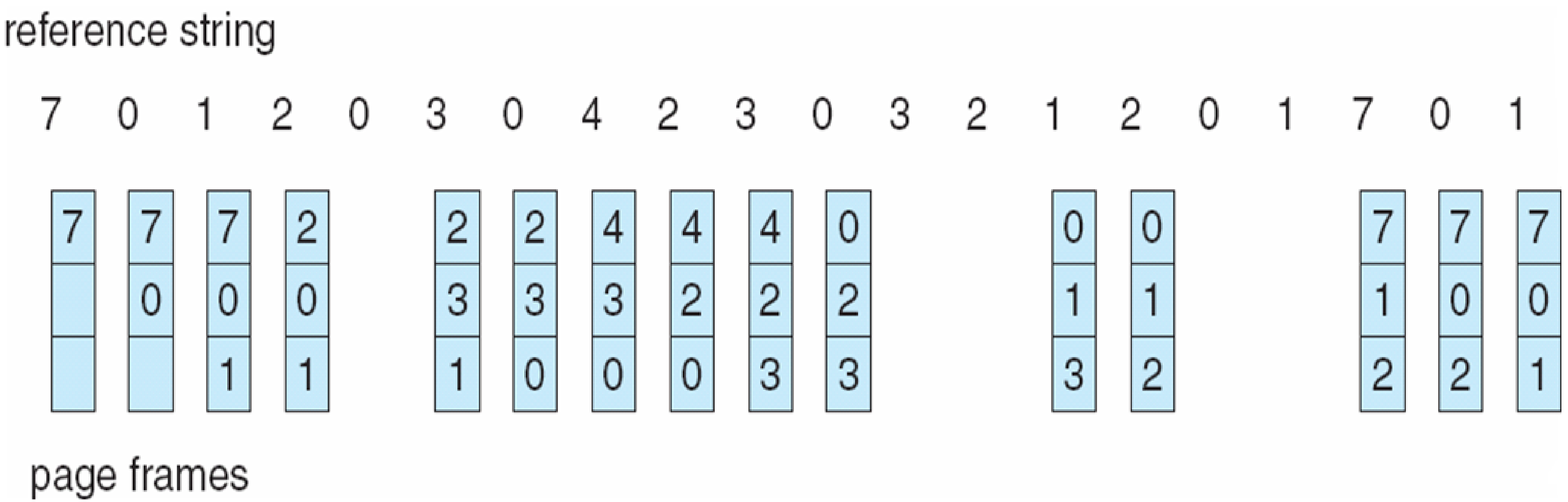
- Page replacement – find some page in memory, but not really in use, swap it out
  - algorithm
  - performance – want an algorithm which will result in minimum number of page faults
  - Use **modify (dirty) bit** to reduce overhead of page transfers – only modified pages are written to disk

Page Replacement



First-In-First-Out (FIFO) Algorithm

Replace page that has come first.



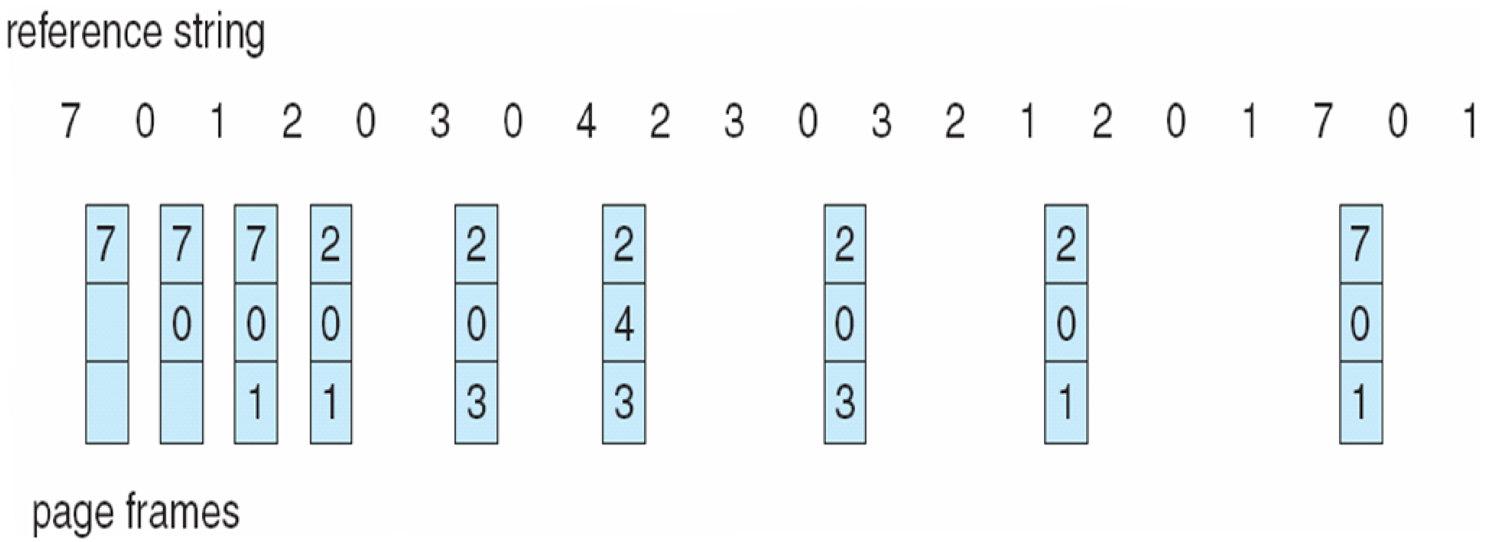
Total page faults = 15

FIFO Algorithm: Belady's Anomaly

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames – 9 page faults
- 4 frames – 10 page faults

Optimal Algorithm

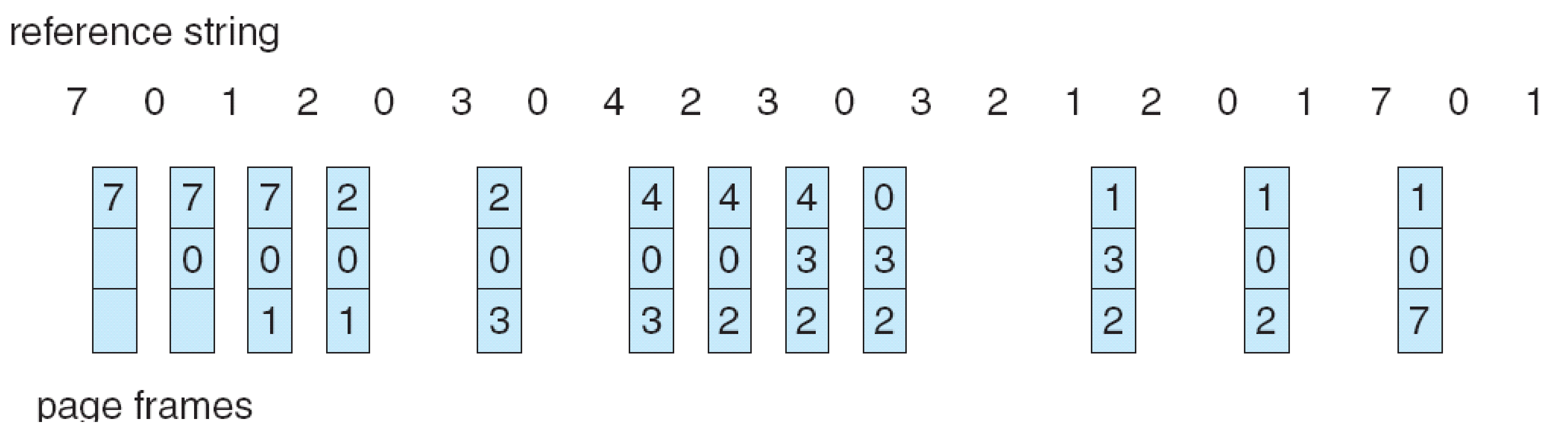
- Replace page that will not be used for longest period of time



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

Least Recently Used (LRU) Algorithm

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

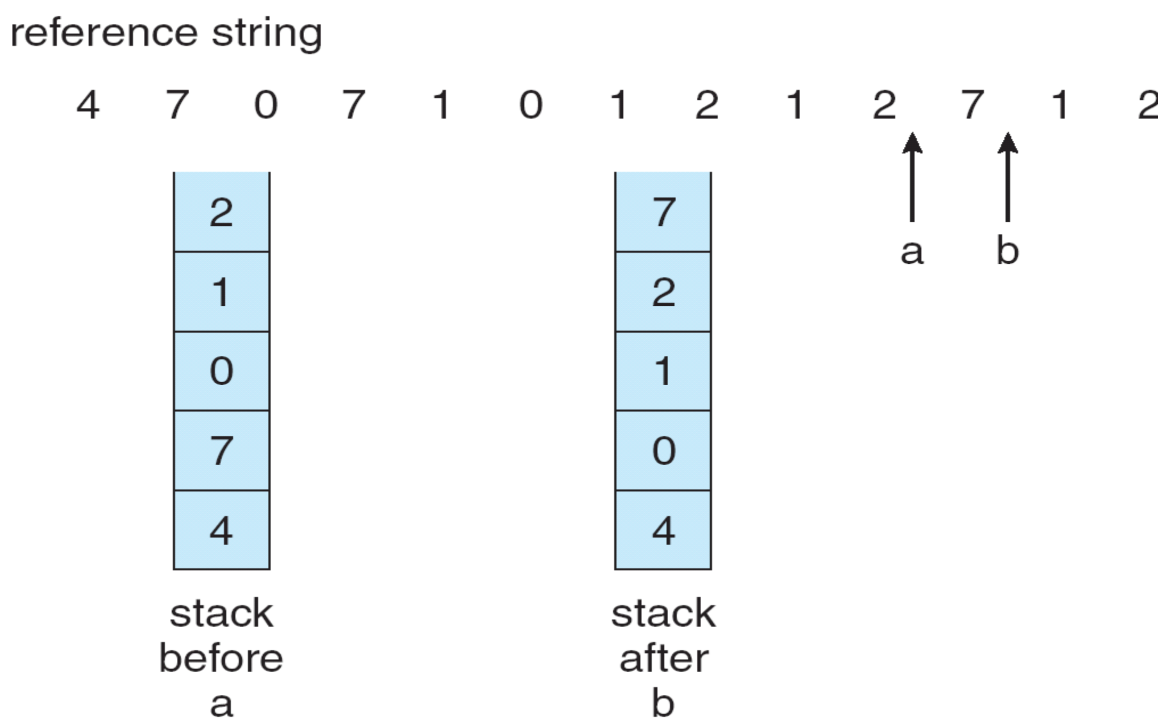


LRU Implementation

- Counter implementation
  - Every page entry has a counter; every time page is referenced, copy the clock into the counter
  - Logical counter ticks with every page reference
  - Replace a valid page with smallest counter

LRU Implementation

- Modified Stack implementation
  - Push at top only, Pop from anywhere
  - Page referenced $\Rightarrow$ on the memory $\Rightarrow$ move it to the top
  - LRU page will be at the bottom always



Additional Reference bits Algorithm

- Keep a reference/use bit and 7-bit reference history for each entry in PMT.
- For every page reference, update the reference bit
- At regular intervals (say 50ns)
  - Perform right shift to the history bits, discarding the LSB
  - Push reference bit value to MSB of history bits
  - Reset reference bit to 0
- Thus 1-byte represents the status of the page usage in 8 intervals.
  - 00000000 – The page has not been used for 8 intervals
  - 11111111 – The page is used at least once in each interval
  - 11000100 is more recently used than 01110111
  - So, lowest number is the LRU page

Additional Reference bits Algorithm

3, 2, 3, T, 8, 0, 3, T, 3, 0, 2, T, 6, 3, 4, 7

T marks the end of each time interval

Initial state

During 1st interval

P	U3	U2	U1	U0
3	1	0	0	0
2	1	0	0	0
-	0	0	0	0
-	0	0	0	0
-	0	0	0	0

At the end of 1st interval

P	U3	U2	U1	U0
3	0	1	0	0
2	0	1	0	0
-	0	0	0	0
-	0	0	0	0
-	0	0	0	0

Additional Reference bits Algorithm

During 2nd interval

P	U3	U2	U1	U0
3	1	1	0	0
2	0	1	0	0
8	1	0	0	0
0	1	0	0	0
-	0	0	0	0

At the end of 2nd interval

P	U3	U2	U1	U0
3	0	1	1	0
2	0	0	1	0
8	0	1	0	0
0	0	1	0	0
-	0	0	0	0

Additional Reference bits Algorithm

During 3rd interval

P	U3	U2	U1	U0
3	1	1	1	0
2	1	0	1	0
8	0	1	0	0
0	1	1	0	0
-	0	0	0	0

At the end of 3rd interval

P	U3	U2	U1	U0
3	0	1	1	1
2	0	1	0	1
8	0	0	1	0
0	0	1	1	0
-	0	0	0	0

Additional Reference bits Algorithm

During 4th interval, after 6, 3

P	U3	U2	U1	U0
3	1	1	1	1
2	0	1	0	1
8	0	0	1	0
0	0	1	1	0
6	1	0	0	0

During 4th interval, after 6, 3, 4

P	U3	U2	U1	U0
3	1	1	1	1
2	0	1	0	1
4	1	0	0	0
0	0	1	1	0
6	1	0	0	0

Page 4 replaced 8 with U bits 0010

Additional Reference bits Algorithm

During 4th interval, after 6, 3, 4, 7

P	U3	U2	U1	U0
3	1	1	1	1
7	1	0	0	0
4	1	0	0	0
0	0	1	1	0
6	1	0	0	0

If there are more victim pages with same U bits then chose FIFO

Page 7 replaced 2 with U bits 0101

Second Chance Algorithm

- Uses
  - a reference/used bit
  - NO history bits
  - FIFO

```
Search FIFO pages circularly
   If FIFO page's used bit is 0 then
         replace it
   Else
         give that FIFO page a second chance
         clear its used bit
         goto next FIFO page
```

Second Chance Algorithm

```
  P    U    3     P    U    2    P    U    3
+----+----+     +----+----+    +----+----+
|    | 0  |*    | 3  | 1  |    | 3  | 1  |
+----+----+     +----+----+    +----+----+
|    | 0  |     |    | 0  |*   | 2  | 1  |
+----+----+     +----+----+    +----+----+
|    | 0  |     |    | 0  |    |    | 0  |*
+----+----+     +----+----+    +----+----+
|    | 0  |     |    | 0  |    |    | 0  |
+----+----+     +----+----+    +----+----+
```

Second Chance Algorithm

```
  P   U   0   P   U   8   P   U   4
+---+---+   +---+---+   +---+---+
| 3 | 1 |   | 3 | 1 |   | 3 | 1 |
+---+---+   +---+---+   +---+---+
| 2 | 1 |   | 2 | 1 |   | 2 | 1 |
+---+---+   +---+---+   +---+---+
|   | 0 | * | 0 | 1 |   | 0 | 1 |
+---+---+   +---+---+   +---+---+
|   | 0 |   |   | 0 | * | 8 | 1 |
+---+---+   +---+---+   +---+---+
|   | 0 |   |   | 0 |   |   | 0 | *
+---+---+   +---+---+   +---+---
```

Second Chance Algorithm

```
  P    U    2       P    U    5      P    U    0
+----+----+       +----+----+      +----+----+
| 3  | 1  |*      | 3  | 1  |*     | 5  | 1  |
+----+----+       +----+----+      +----+----+
| 2  | 1  |       | 2  | 1  |      | 2  | 0  |*
+----+----+       +----+----+      +----+----+
| 0  | 1  |       | 0  | 1  |      | 0  | 0  |
+----+----+       +----+----+      +----+----+
| 8  | 1  |       | 8  | 1  |      | 8  | 0  |
+----+----+       +----+----+      +----+----+
| 4  | 1  |       | 4  | 1  |      | 4  | 0  |
+----+----+       +----+----+      +----+----+
```

Second Chance Algorithm

```
    P     U      9      P     U      8      P     U    3
  +---+---+          +---+---+          +---+---+
  | 5 | 1 |          | 5 | 1 |          | 5 | 1 |
  +---+---+          +---+---+          +---+---+
  | 2 | 0 |*         | 9 | 1 |          | 9 | 1 |
  +---+---+          +---+---+          +---+---+
  | 0 | 1 |          | 0 | 1 |*         | 0 | 1 |*
  +---+---+          +---+---+          +---+---+
  | 8 | 0 |          | 8 | 0 |          | 8 | 1 |
  +---+---+          +---+---+          +---+---+
  | 4 | 0 |          | 4 | 0 |          | 4 | 0 |
  +---+---+          +---+---+          +---+---
```

Second Chance Algorithm

```
   P     U     2      P     U
+----+----+        +----+----+
| 5  | 1  |*       | 5  | 0  |
+----+----+        +----+----+
| 9  | 1  |        | 9  | 0  |
+----+----+        +----+----+
| 0  | 0  |        | 2  | 1  |
+----+----+        +----+----+
| 8  | 0  |        | 8  | 0  |*
+----+----+        +----+----+
| 3  | 1  |        | 3  | 1  |
+----+----+        +----+----+
```

Enhanced Second Chance Algorithm

- Considers reference/used bit and dirty/modified bit
- Four cases, based on (R,M), the pair:
  - (R,M) = (0,0) neither recently used nor modified -- best to replace
  - (R,M) = (0,1) not recently used but modified -- second choice
  - (R,M) = (1,0) recently used but not modified -- third choice
  - (R,M) = (1,1) recently used and modified -- fourth choice

Enhanced Second Chance Algorithm

- If class-1 page then use it
- If class-2 page then record first instance
- Else clear R bit and go to next FIFO page
- If scan is completed then
  - If class-2 page was recorded then use that recorded instance
  - Else repeat another scan

Counting based page replacements

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
  - Otherwise high page fault rate => slow execution
- Two major allocation schemes
  - Fixed allocation
  - Priority allocation

Fixed Allocation: Equal allocation

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

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

$$S = \sum s_i$$

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

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

$$m = 64$$

$$s_i = 10$$

$$s_2 = 127$$

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

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

Priority Allocation

- Use a proportional allocation scheme using priorities rather than size

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

- In local replacement, no. of allocated frames is not affected by other processes => page fault rate is not affected by other processes.

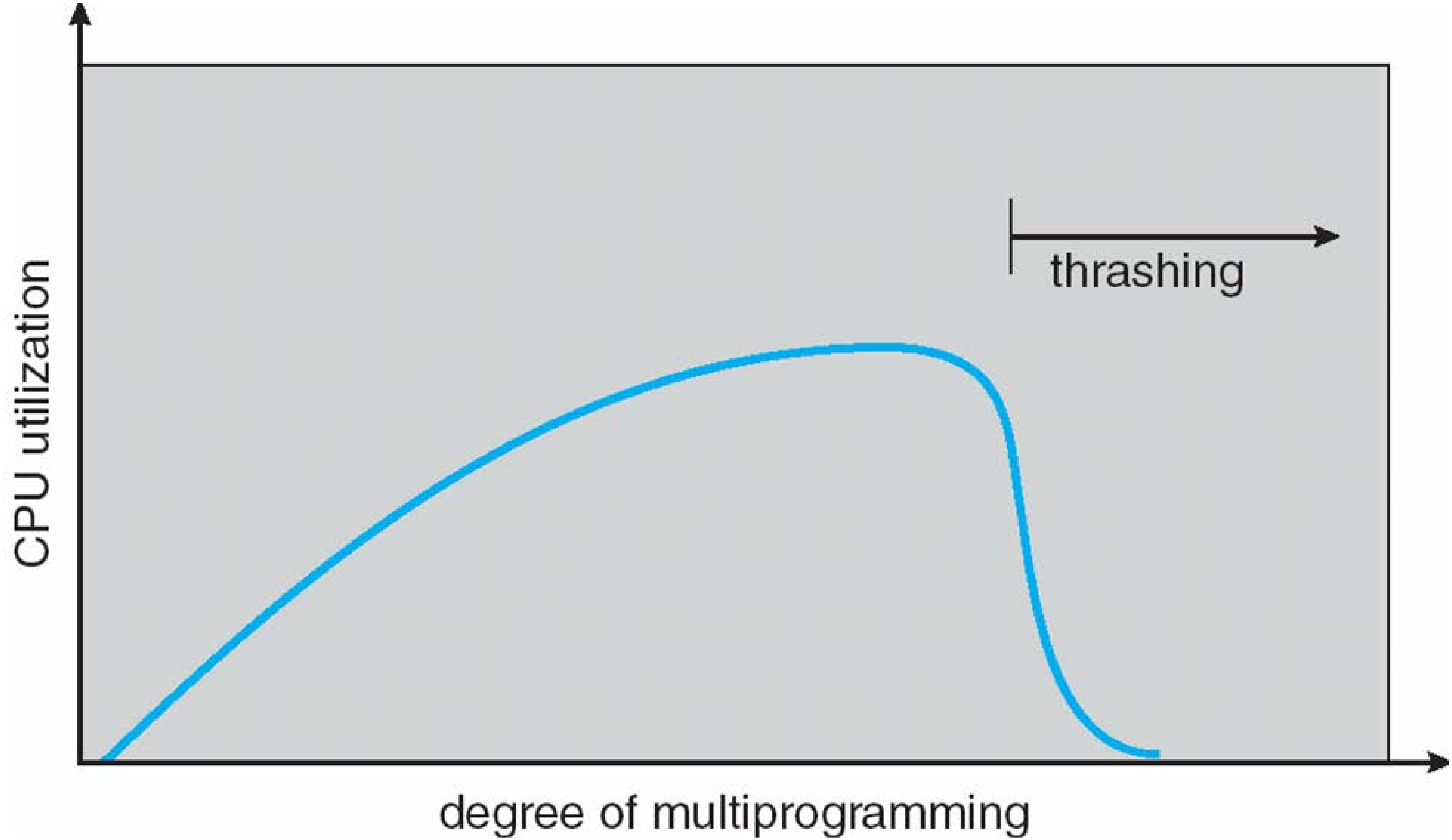
Thrashing

- P_{high} steals frames from P_{low}
- Page fault rate increases for P_{low}
- P_{hig}h steals more frames from P_{low}
- Page fault rate further increases for P_{low}

- If a process does not have "enough" frames, 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



Demand Paging and Thrashing

Why does demand paging work?

Locality model

- Process migrates from one locality to another
  - Currently executing **main** function (locality-1), may call **sqrt** function (locality-2)
- Localities may overlap
  - both the functions may use the same global pages

Why does thrashing occur?

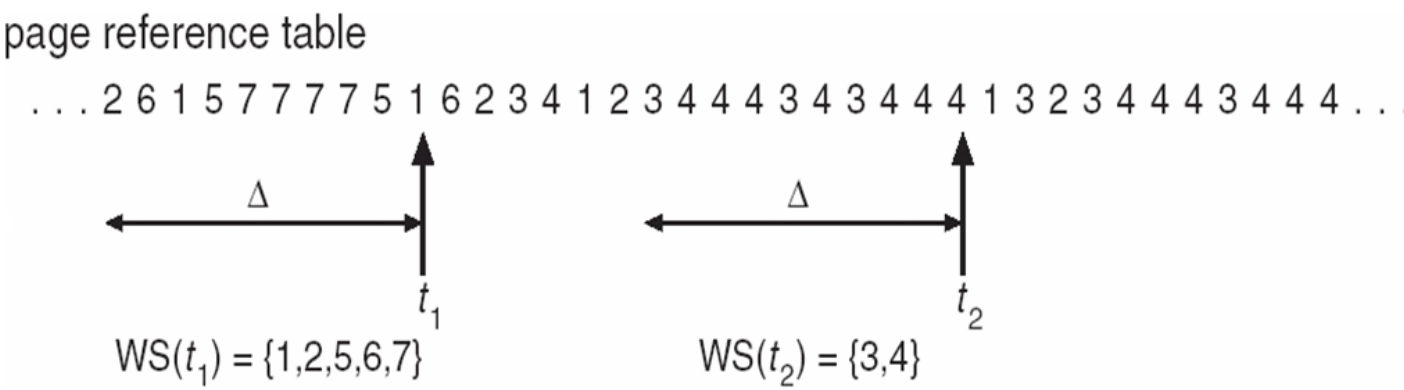
- $\sum$size of locality > total memory size

Solution to Thrashing

- Use local page replacement algorithm.
- Use working-set model.
- Page fault frequency scheme.

Working-set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references (say 10)
- **Working-Set** is the set of distinct pages in the working-set window.
- $WSS_i \equiv$ working set size of Process P_i



Working-set Model

- Performance depends on Δ
  - 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 prevents thrashing while keeping the degree of multiprogramming as high as possible.

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