



# Basic Page Replacement

---

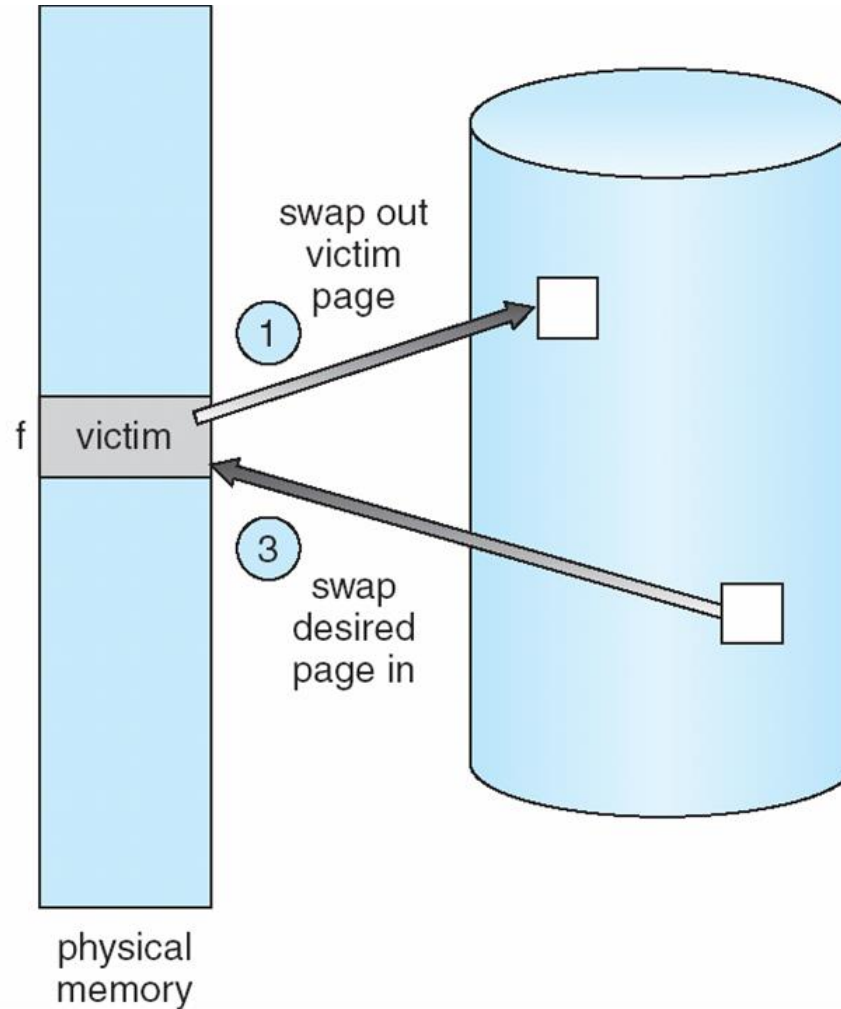
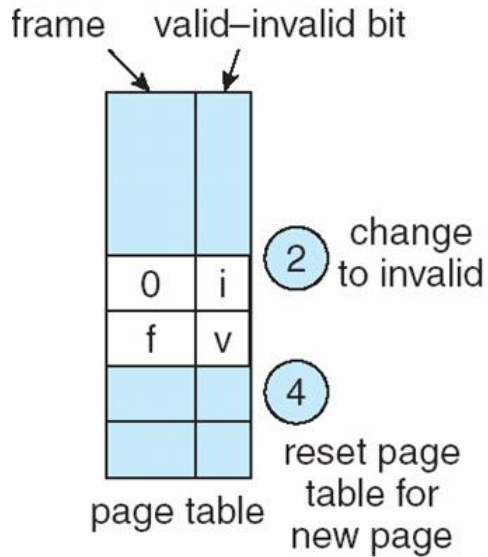
1. Find the location of the desired page on disk
2. Find a free frame:
  - If there is a free frame, use it
  - If there is no free frame, use a page replacement algorithm to select a **victim** frame
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Restart the process

Use **modify (dirty) bit** to reduce overhead of page transfers – only modified pages are written to disk





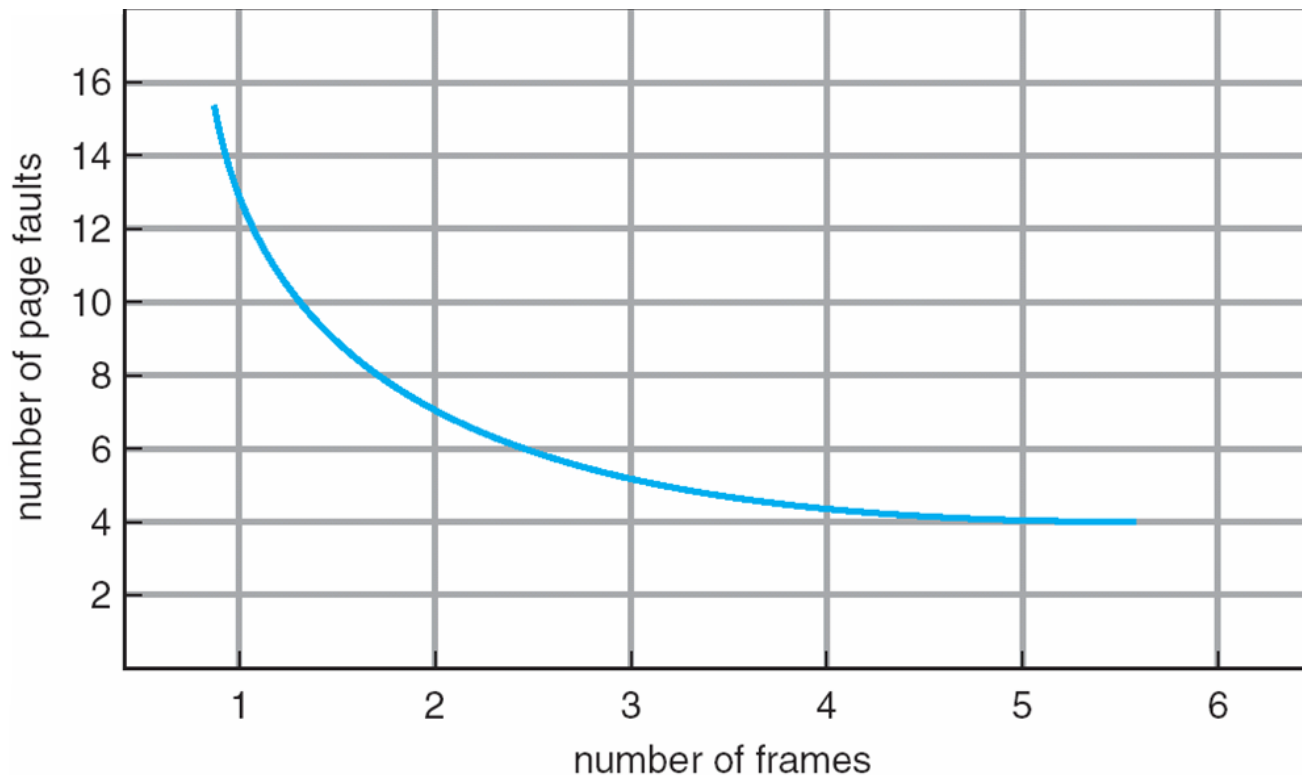
# Page Replacement





# 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

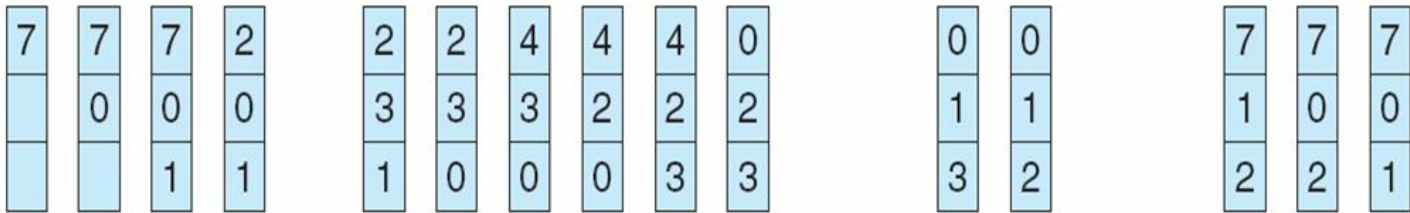




# First-In-First-Out Page Replacement

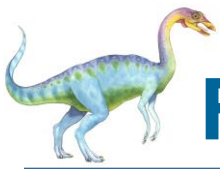
reference string

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



page frames





# 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)

1	1	4	5	
2	2	1	3	9 page faults
3	3	2	4	

- 4 frames

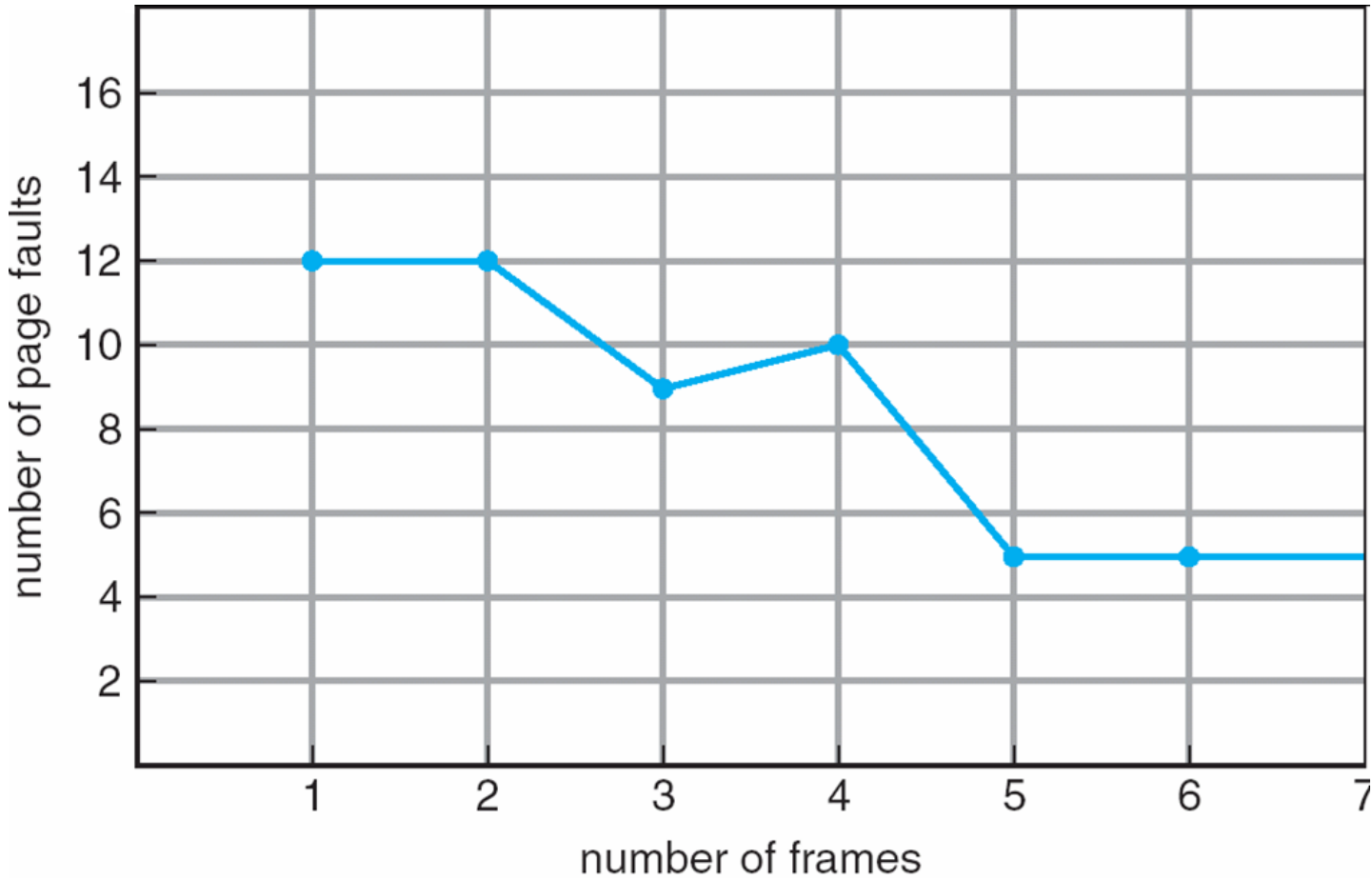
1	1	5	4	
2	2	1	5	10 page faults
3	3	2		
4	4	3		

- Belady's Anomaly: more frames  $\Rightarrow$  more page faults





# FIFO Illustrating Belady's Anomaly

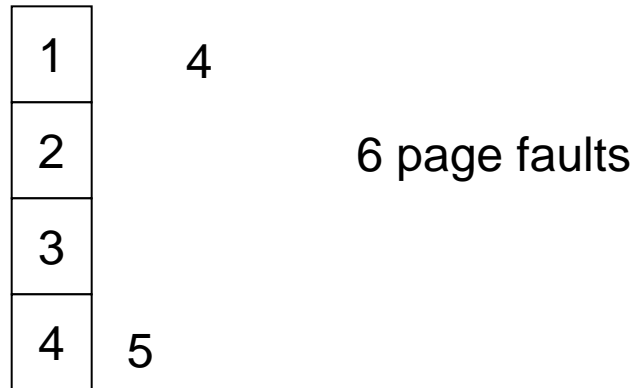




# Optimal Algorithm

- 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



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

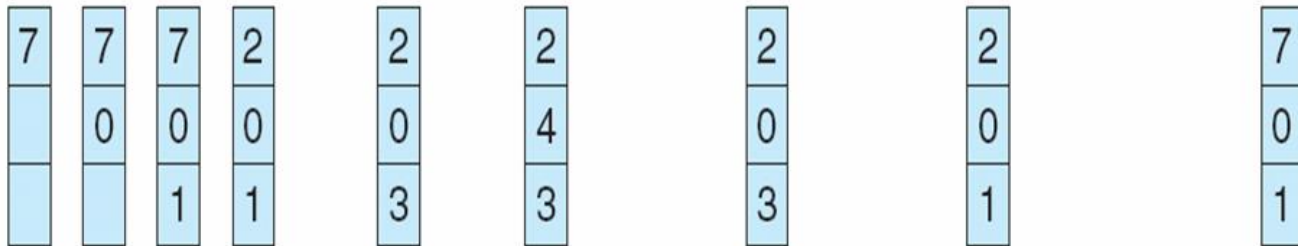




# Optimal Page Replacement

reference string

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



page frames







# 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

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

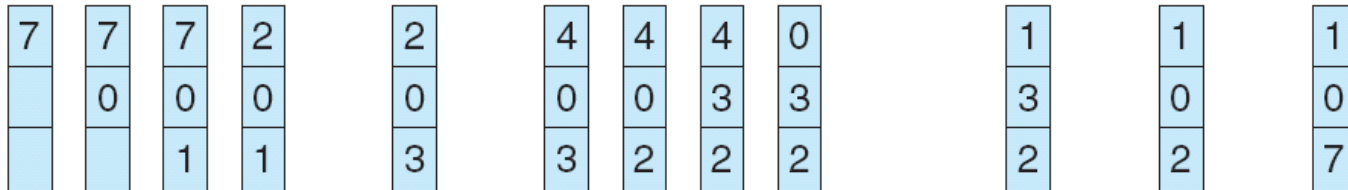




# LRU Page Replacement

reference string

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



page frames





# 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*





# 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  $p_i = \frac{s_i}{S} \times m$

$$m = 64$$

$$s_i = 10$$

$$s_2 = 127$$

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

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





# Global vs. Local Allocation

---

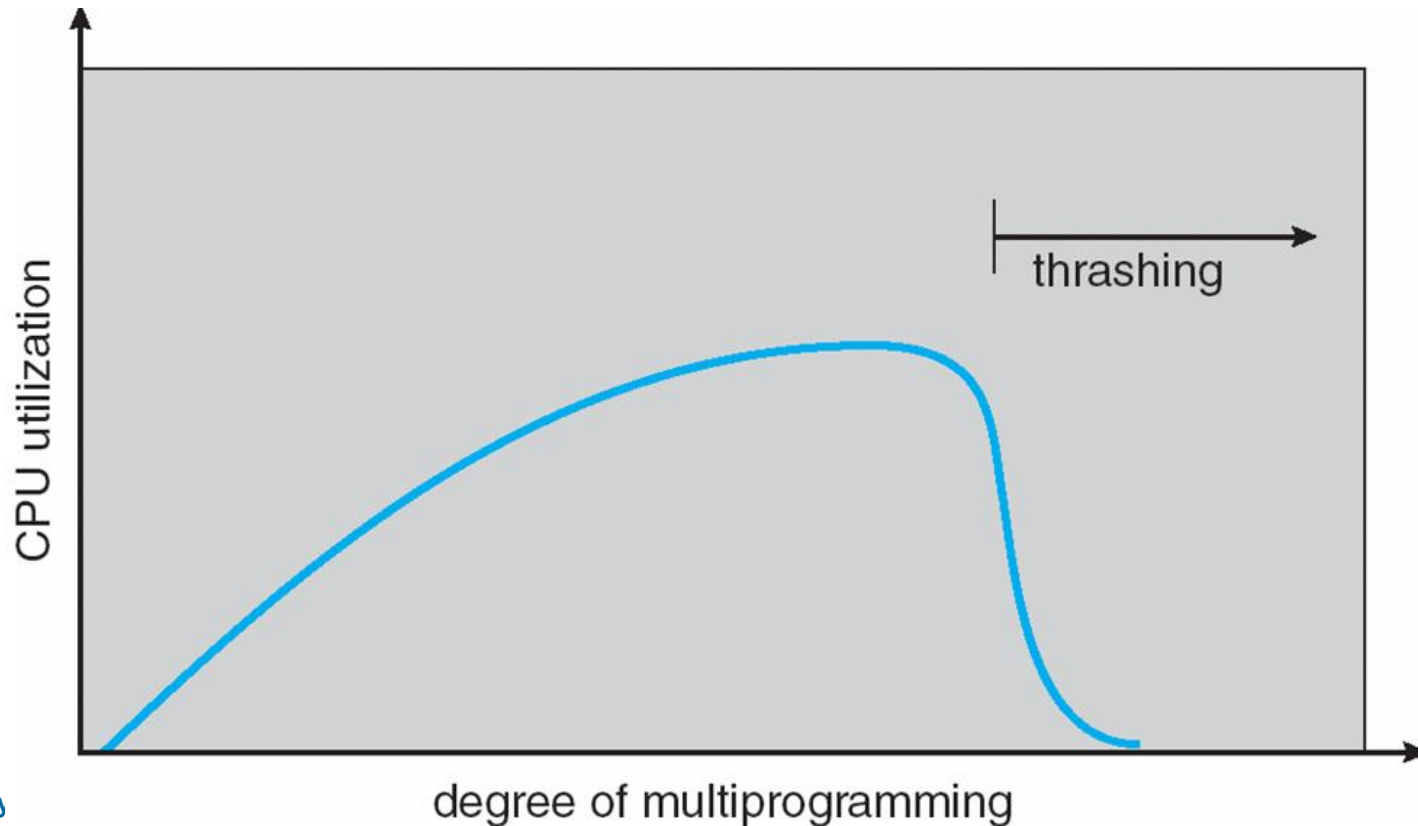
- **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another
- **Local replacement** – each process selects from only its own set of allocated frames





# Thrashing

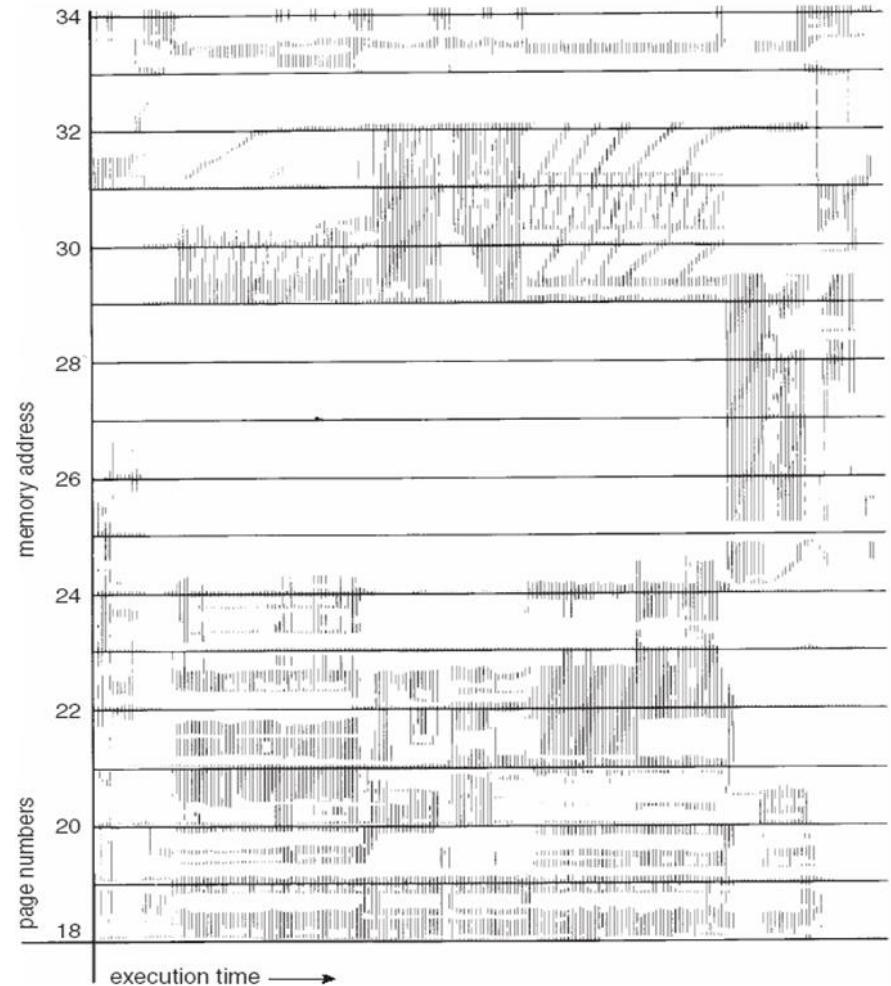
- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
- **Thrashing**  $\equiv$  a process is busy swapping pages in and out





# Demand Paging and Thrashing

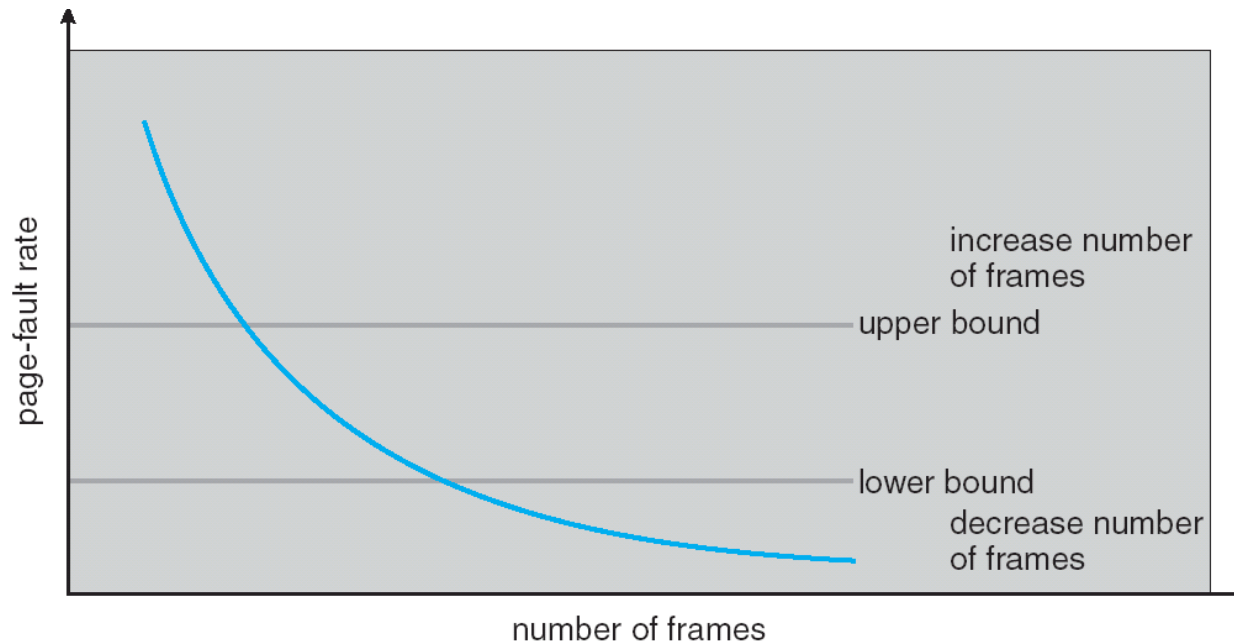
- Why does demand paging work?  
Locality model
  - Process migrates from one locality to another
  - Localities may overlap
- Why does thrashing occur?  
 $\Sigma$  size of locality > total memory size





# 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







# Allocating Kernel Memory

---

- Treated differently from user memory
- Often allocated from a free-memory pool
  - Kernel requests memory for structures of varying sizes
  - Some kernel memory needs to be contiguous





# Buddy System

---

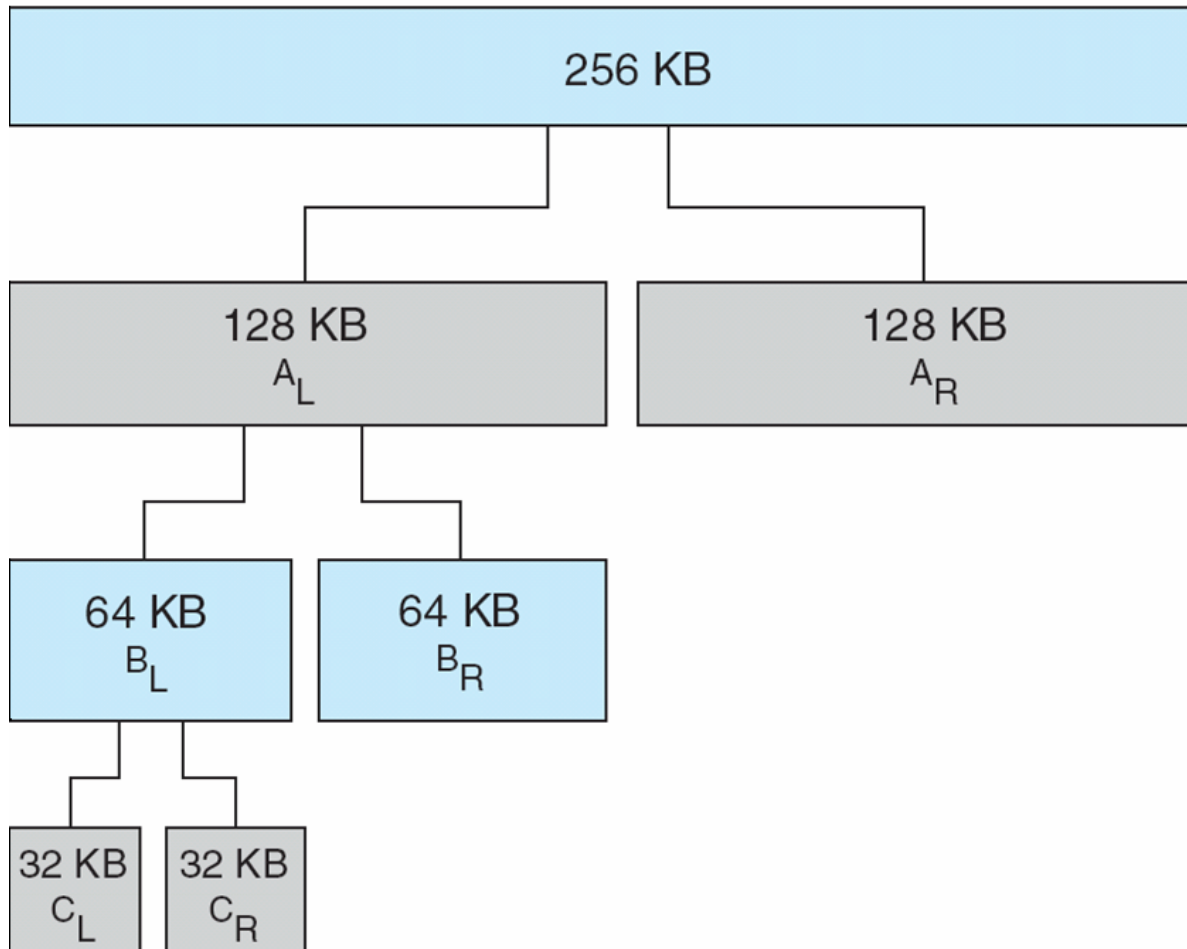
- Allocates memory from fixed-size segment consisting of physically-contiguous pages
- Memory allocated using **power-of-2 allocator**
  - Satisfies requests in units sized as power of 2
  - Request rounded up to next highest power of 2
  - When smaller allocation needed than is available, current chunk split into two buddies of next-lower power of 2
    - ▶ Continue until appropriate sized chunk available





# Buddy System Allocator

physically contiguous pages





# Other Issues -- Prepaging

---

- Prepaging
  - To reduce the large number of page faults that occurs at process startup
  - Prepage all or some of the pages a process will need, before they are referenced
  - But if prepagged pages are unused, I/O and memory was wasted





# Other Issues – Page Size

---

- Page size selection must take into consideration:
  - fragmentation
  - table size
  - I/O overhead
  - locality



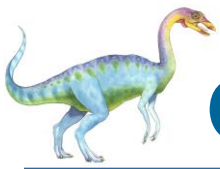


# Other Issues – TLB Reach

---

- TLB Reach - The amount of memory accessible from the TLB
- TLB Reach = (TLB Size) X (Page Size)
- Ideally, the working set of each process is stored in the TLB
  - Otherwise there is a high degree of page faults
- Increase the Page Size
  - This may lead to an increase in fragmentation as not all applications require a large page size
- Provide Multiple Page Sizes
  - This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation





# Other Issues – Program Structure

## ■ Program structure

- `Int[128,128] data;`
- Each row is stored in one page
- Program 1

```
for (j = 0; j < 128; j++)  
    for (i = 0; i < 128; i++)  
        data[i,j] = 0;
```

128 x 128 = 16,384 page faults

- Program 2

```
for (i = 0; i < 128; i++)  
    for (j = 0; j < 128; j++)  
        data[i,j] = 0;
```

128 page faults





# Other Issues – I/O interlock

---

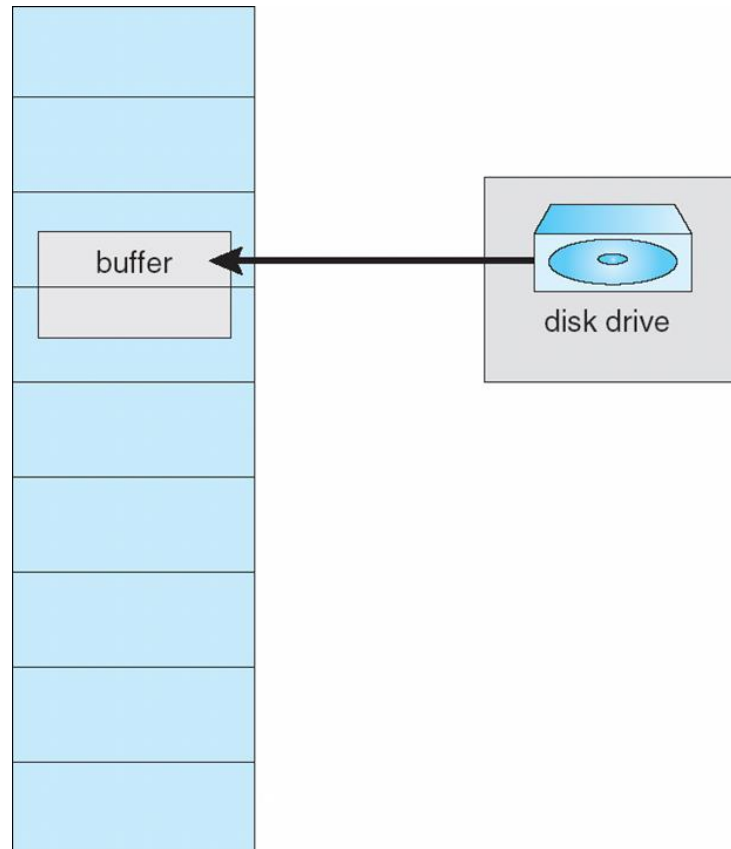
- **I/O Interlock** – Pages must sometimes be locked into memory
- Consider I/O - Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm







# Reason Why Frames Used For I/O Must Be In Memory



# End of Chapter 9

---

