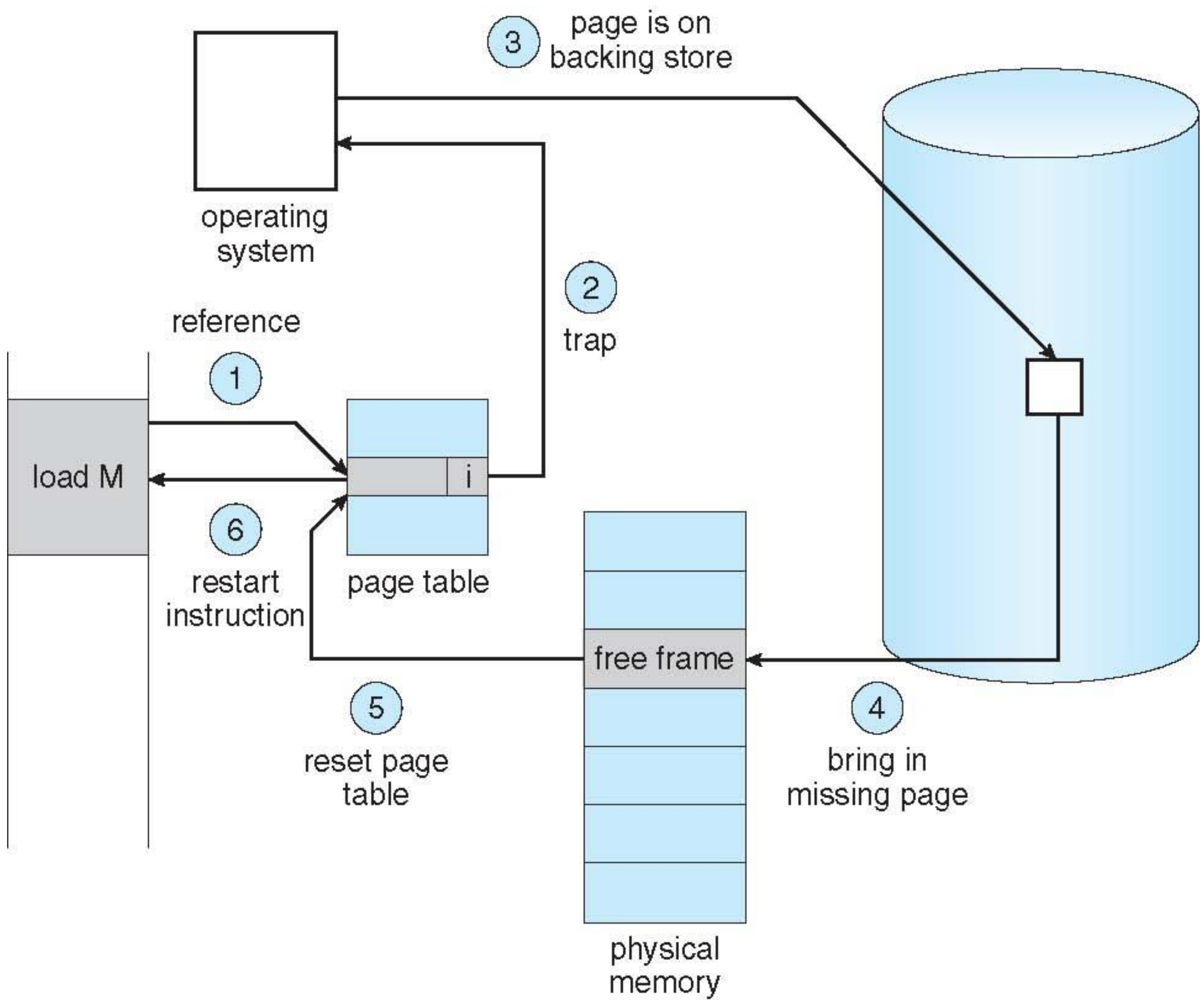
L27- FRAME ALLOCATION

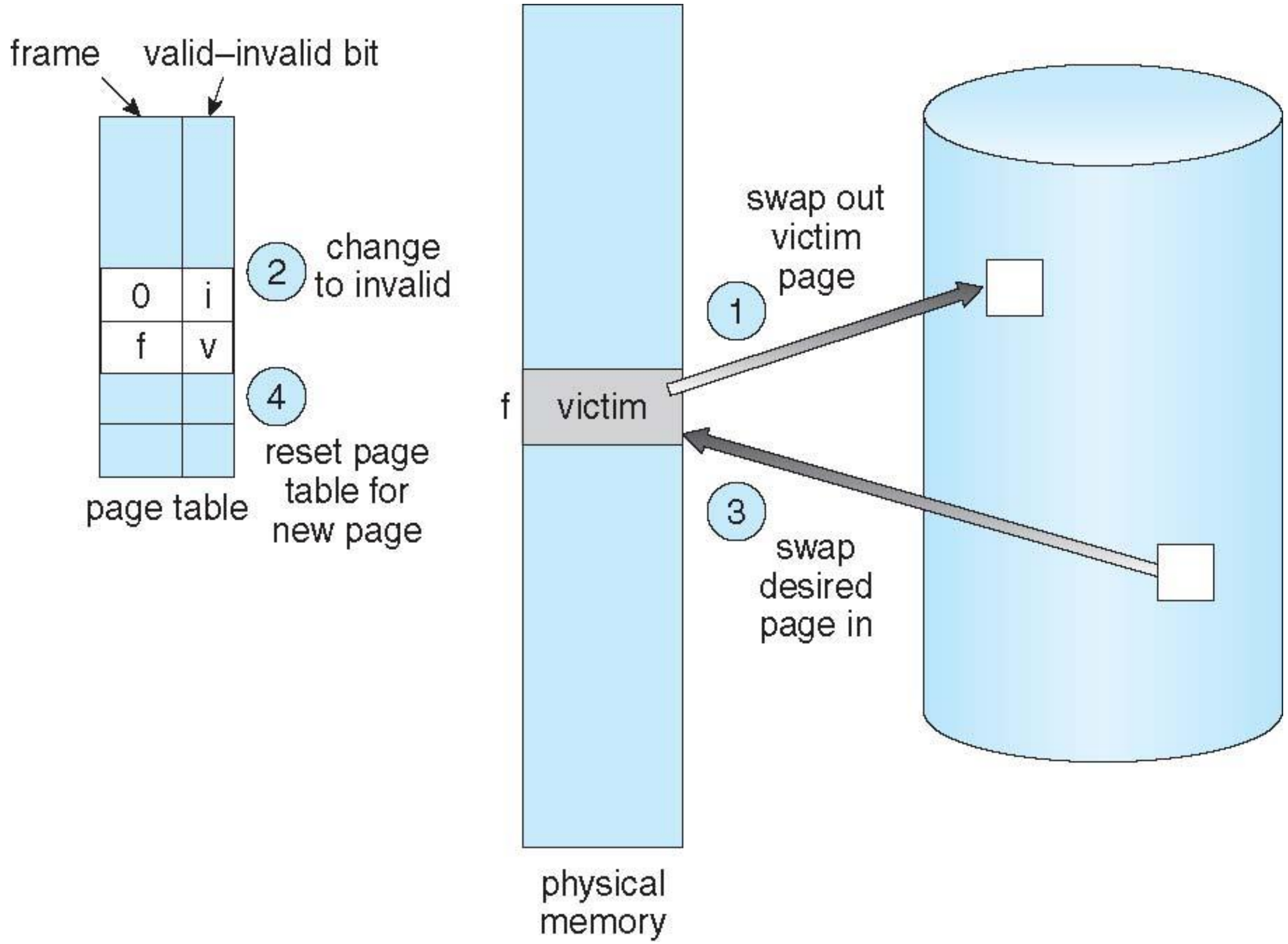
Overview of Memory Management

- Demand Paging
- Copy-on-Write
- Page Replacement
- Allocation of Frames
- Thrashing

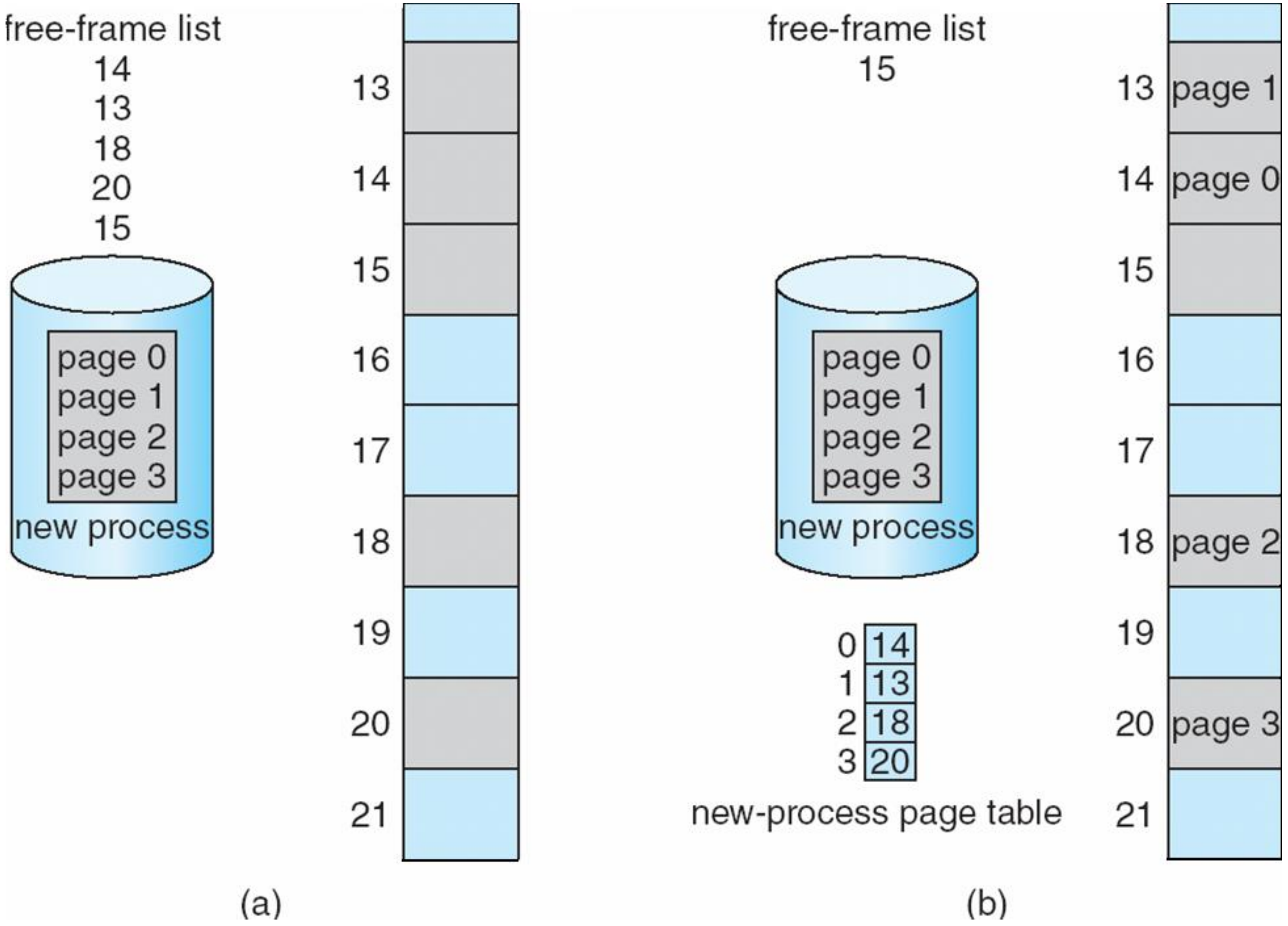
Steps in Handling a Page Fault



Page Replacement



Free Frames Allocation



Before allocation After allocation

Allocation of Frames

- **Frame-allocation algorithm** determines
  - How many frames to give each process?
  - Which frames to replace?
- Two major allocation schemes
  - fixed allocation
  - priority allocation

Fixed Allocation

- Equal allocation – 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

- Proportional allocation – Allocate according to the size of process

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

$$S = \sum s_i$$

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

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

$$m = 64$$

$$s_1 = 10$$

$$s_2 = 127$$

$$a_1 = \frac{10}{137} \times 64 \approx 4$$

$$a_2 = \frac{127}{137} \times 64 \approx 57$$

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

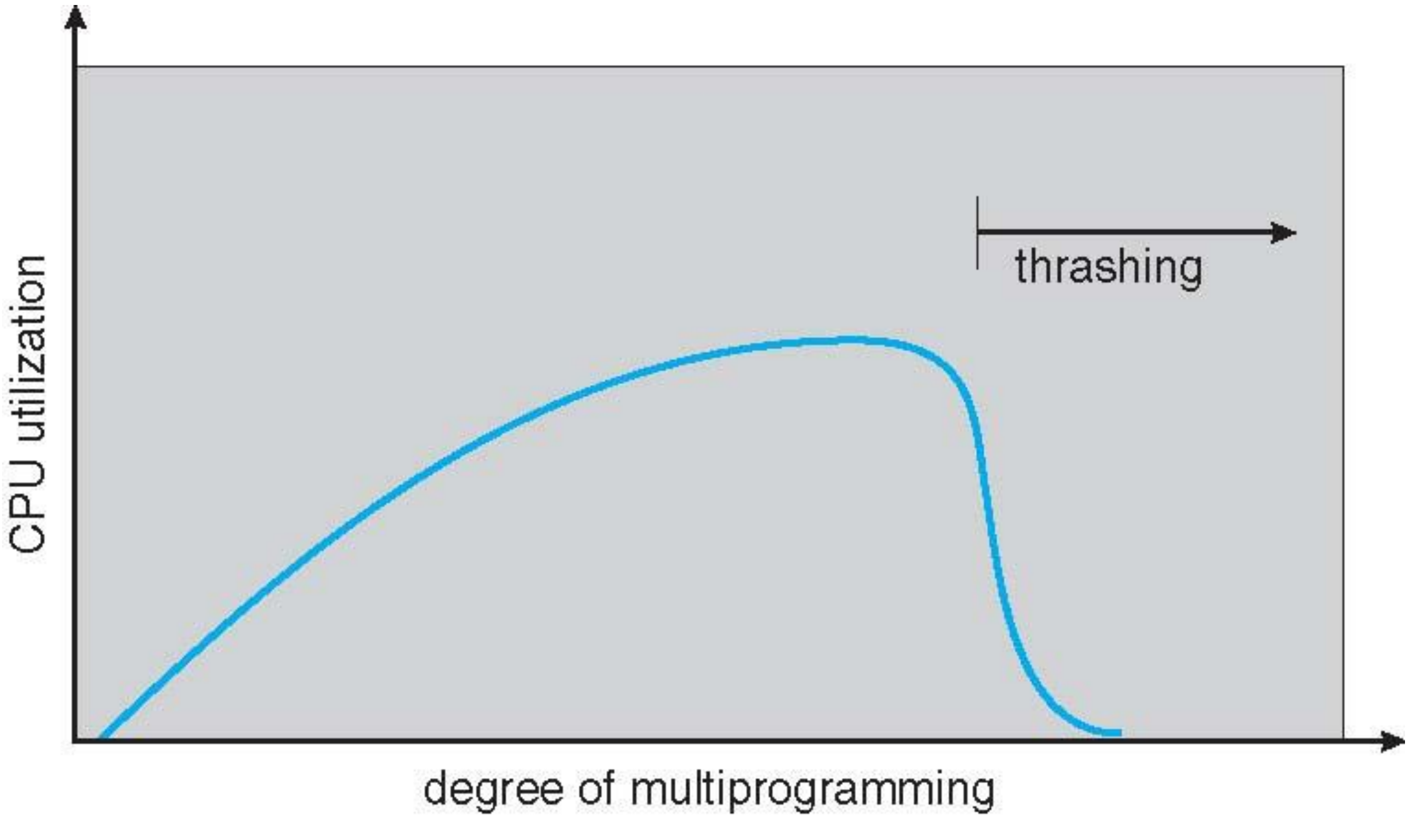
- **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another
  - But then process execution time can vary greatly
  - But greater throughput so more common

- **Local replacement** – each process selects from only its own set of allocated frames
  - More consistent per-process performance
  - But possibly underutilized memory

Thrashing

- If a process does not have enough pages, the page-fault rate is 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** $\equiv$ a process is busy swapping pages in and out

Thrashing



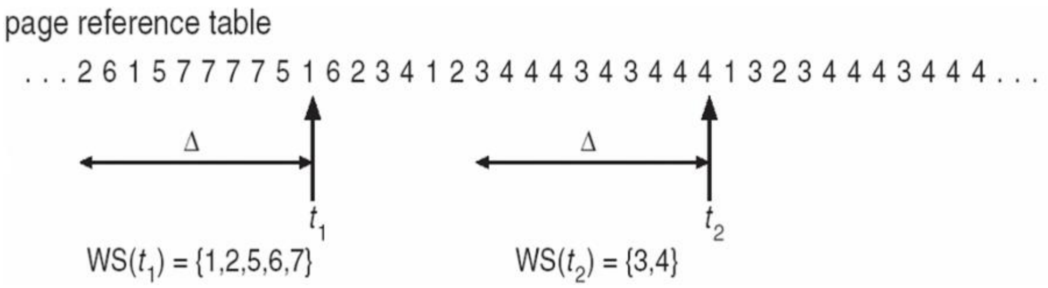
Demand Paging and Thrashing

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

- Why does thrashing occur?
  - Σ size of locality > total memory size
  - Limit effects by using local or priority page replacement

Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  Example: 10,000 instructions
- WS_i (working set of Process P_i) = Pages referenced in the most recent 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



- $D = \Sigma\, WS_i \equiv$ total demand frames (Approximation of locality)
- if $D > m \Rightarrow$ Thrashing; if $D > m$, then suspend/swap out processes

How to compute Working-Set?

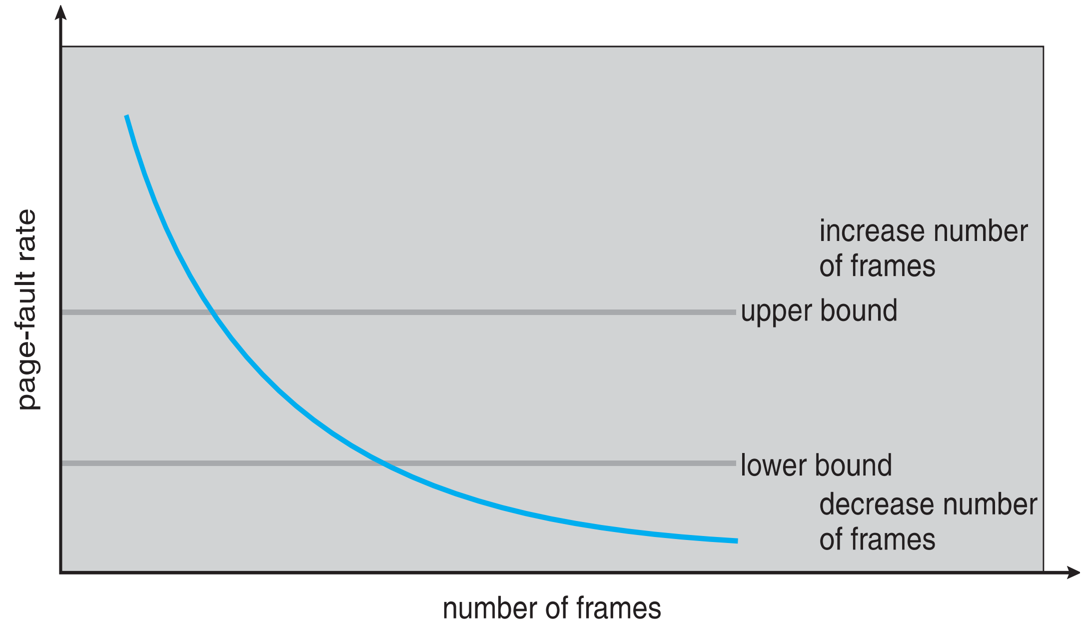
- Approximate with interval timer + a reference bit
- Example: Δ = 10,000: Timer interrupts after every 5000 time units
  - 2 history bits for each page is kept in memory
    - Whenever a timer interrupts, copy the reference bit to history bit.
    - Sets the values of all reference bits to 0
  - During page fault, if one of the history bits = 1 $\Rightarrow$ page in working set

How to compute Working-Set?

- Why counter, history and reference bits approach not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units

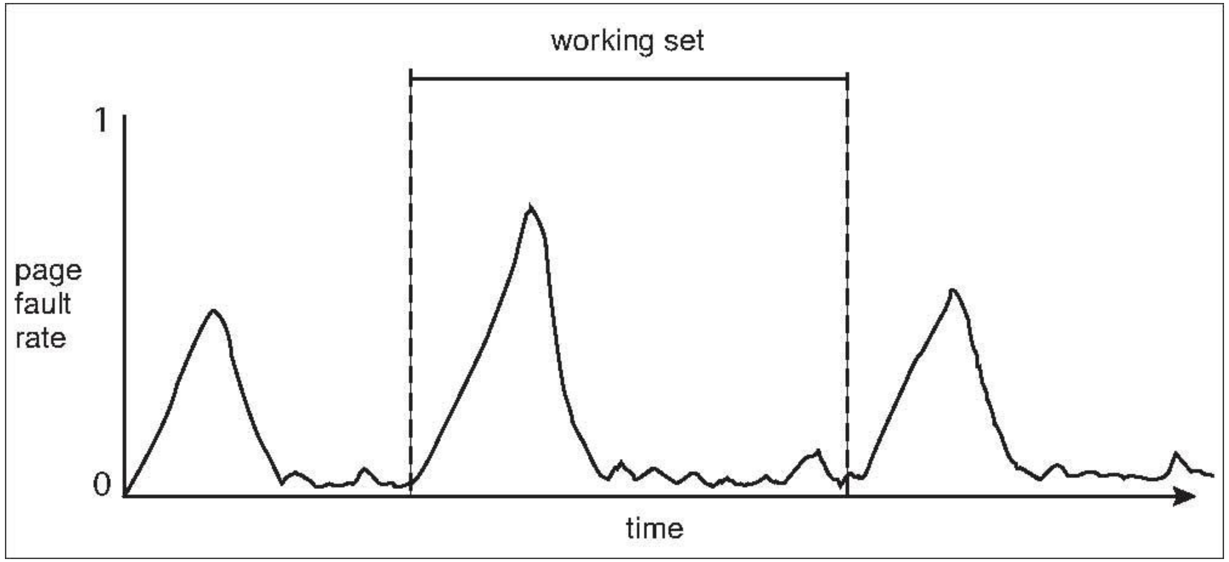
Page-Fault Frequency

- More direct approach than WSS
- 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



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

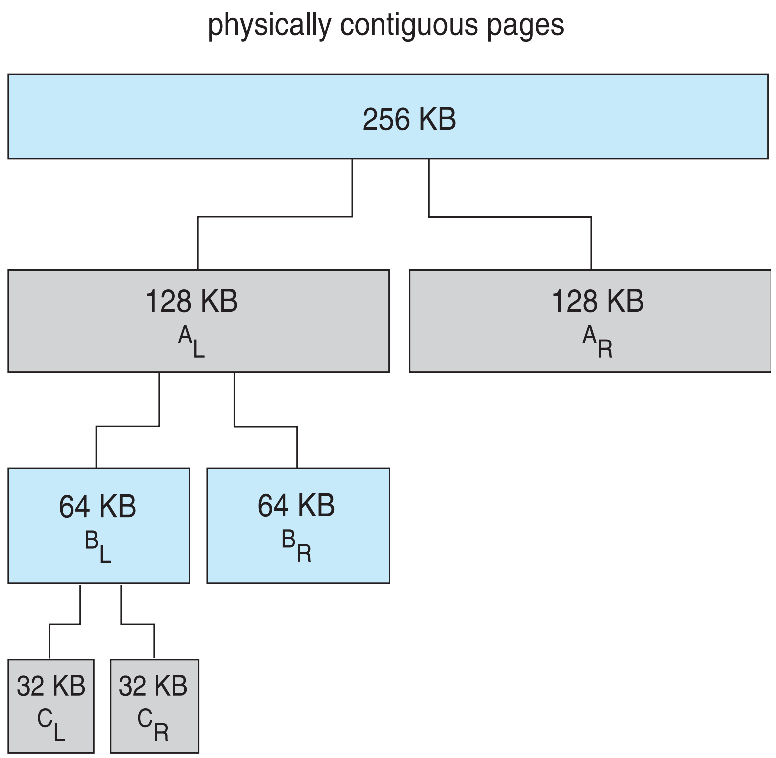


Buddy System

- 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
- Advantage – quickly coalesce unused chunks into larger chunk
- Disadvantage - fragmentation

Buddy System Allocator

- ❖ Assume 256KB chunk available
- ❖ Kernel requests 21KB
  - ❖ Split into A_L and A_R of 128KB each
  - ❖ One further divided into B_L and B_R of 64KB
  - ❖ One further into C_L and C_R of 32KB each
  - ❖ One used to satisfy request



Prepaging

- 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 prepaged pages are unused, I/O and memory was wasted
- Assume *s* pages are prepaged and *α* of the pages is used
- Cost of $\boldsymbol{s * \alpha}$ saved pages faults vs cost of prepaging $\boldsymbol{s * (1- \alpha)}$ unnecessary pages
- $\boldsymbol{\alpha}$ near zero $\Rightarrow$ prepaging loses

Page Size

- Sometimes OS designers have a choice on custom-built CPU
- Page size selection criteria:
  - Fragmentation and Resolution
  - Page table size
  - I/O overhead
  - Number of page faults
  - Locality
  - TLB size and effectiveness
- Always power of 2, usually in the range 2^{12} to 2^{22}

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 more time spend in resolving memory references in page table (delay).
- 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

Program Structure

- `int[128,128] data;` Each row is stored in one page
- A page can store 128 words

Program 1 [128 x 128 = 16,384 page faults]

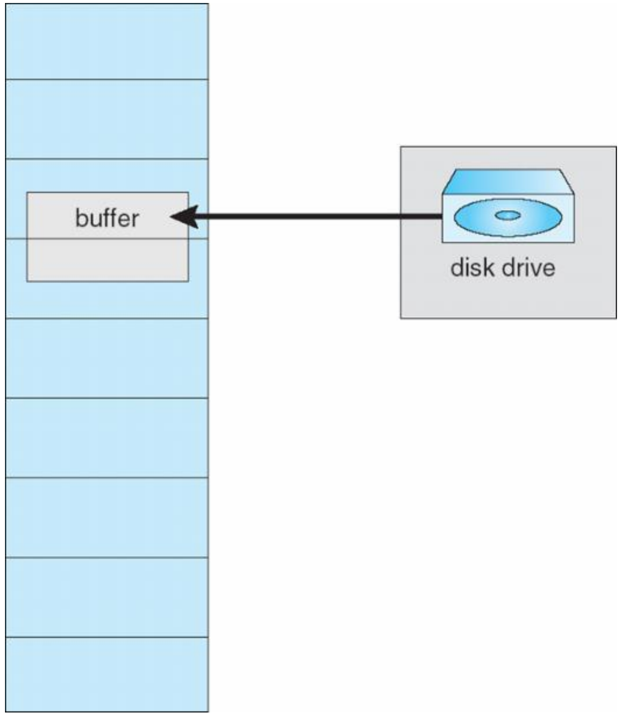
```
for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
        data[i,j] = 0;
```

Program 2 [128 page faults]

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

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
- When I/O is complete pages are unlocked



Thank You