THE UNIVERSITY of EDINBURGH
informatics

Operating Systems (INFR10079)

2023/2024 Semester 2

Virtual Memory
(Working Set and More)

abarbala@inf.ed.ac.uk

Chapter 10

Frames Among Processes

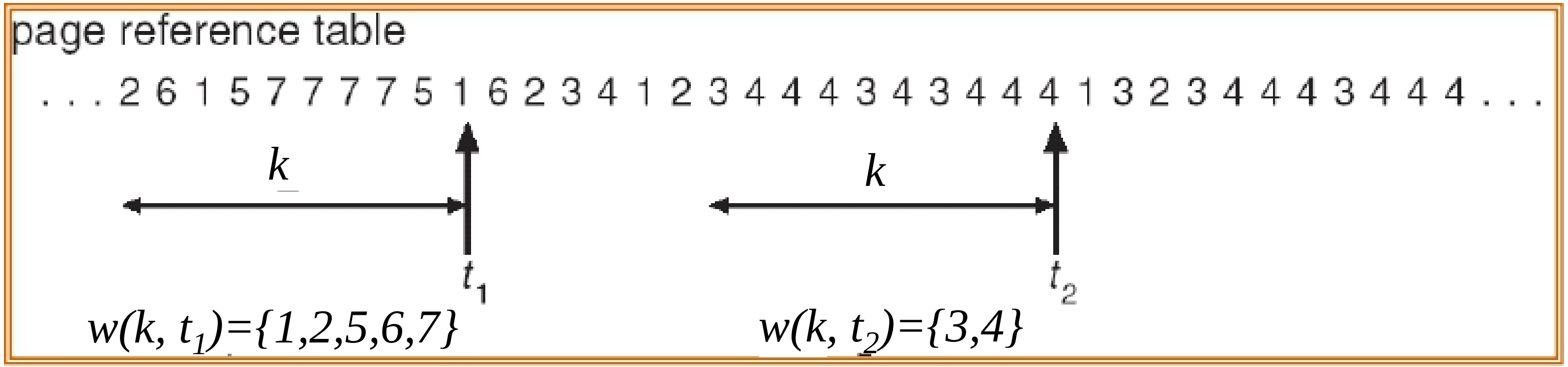
- Frame **Allocation**
  - **Equal:** an equal share each
  - **Proportional:** a share based on the program size
  - …
- Frame **Replacement**
  - **Local:** each process is given a limit of pages it can use
    - Process "pages against itself" (evicts its own pages)
      - Doesn't affect other processes
    - Poor utilization of (all) free page frames, and long access time
  - **Global:** the "victim" is chosen from among all page frames
    - Regardless of owner
    - Processes' page frame allocation can vary dynamically
    - Risk of global thrashing (see later)

How many pages a program really needs?

The *working set model of program behavior*

- Working set of a process is used to model the dynamic locality of its memory usage
  - working set = set of pages process currently “needs”
  - formally defined by Peter Denning in the 1960’s
- **Definition**
  - $w(k, t)$ = {pages referenced in the time interval $(t-k, t)$}
    - t: time
    - k: working set *window* (measured in page refs)
    - A page is in WS only if it was referenced in the last k references
- Working set varies over the life of the program
  - so does the working set size

Working Set Model



Examples of working set for k = 10

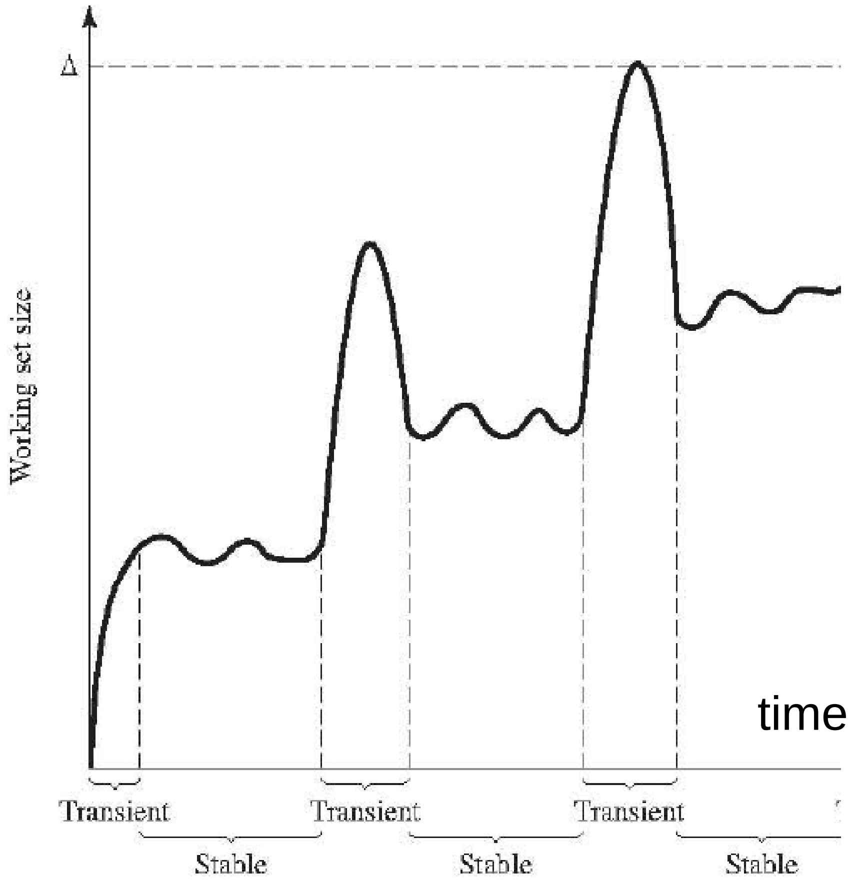
- Working set is the set of pages used by
  - the *k most recent m*emory references
- $|w(k, t)|$ is the size of the working set at time t

Working Set as Defined by Window Size

Sequence of Page References	Window Size, k: 2	3	4	5
24	24	24	24	24
15	24 15	24 15	24 15	24 15
18	15 18	24 15 18	24 15 18	24 15 18
23	18 23	15 18 23	24 15 18 23	24 15 18 23
24	23 24	18 23 24	•	•
17	24 17	23 24 17	18 23 24 17	15 18 23 24 17
18	17 18	24 17 18	•	18 23 24 17
24	18 24	•	24 17 18	•
18	•	18 24	•	24 17 18
17	18 17	24 18 17	•	•
17	17	18 17	•	•
15	17 15	17 15	18 17 15	24 18 17 15
24	15 24	17 15 24	17 15 24	•
17	24 17	•	•	17 15 24
24	•	24 17	•	•
18	24 18	17 24 18	17 24 18	15 17 24 18

Working Set Size

- Working set size, |*w(k, t)*|
  - Changes with program locality
- During periods of poor locality
  - More pages are referenced
  - Working set size is larger
- The working set must be all in memory
  - Otherwise, heavy faulting
  - Thrashing



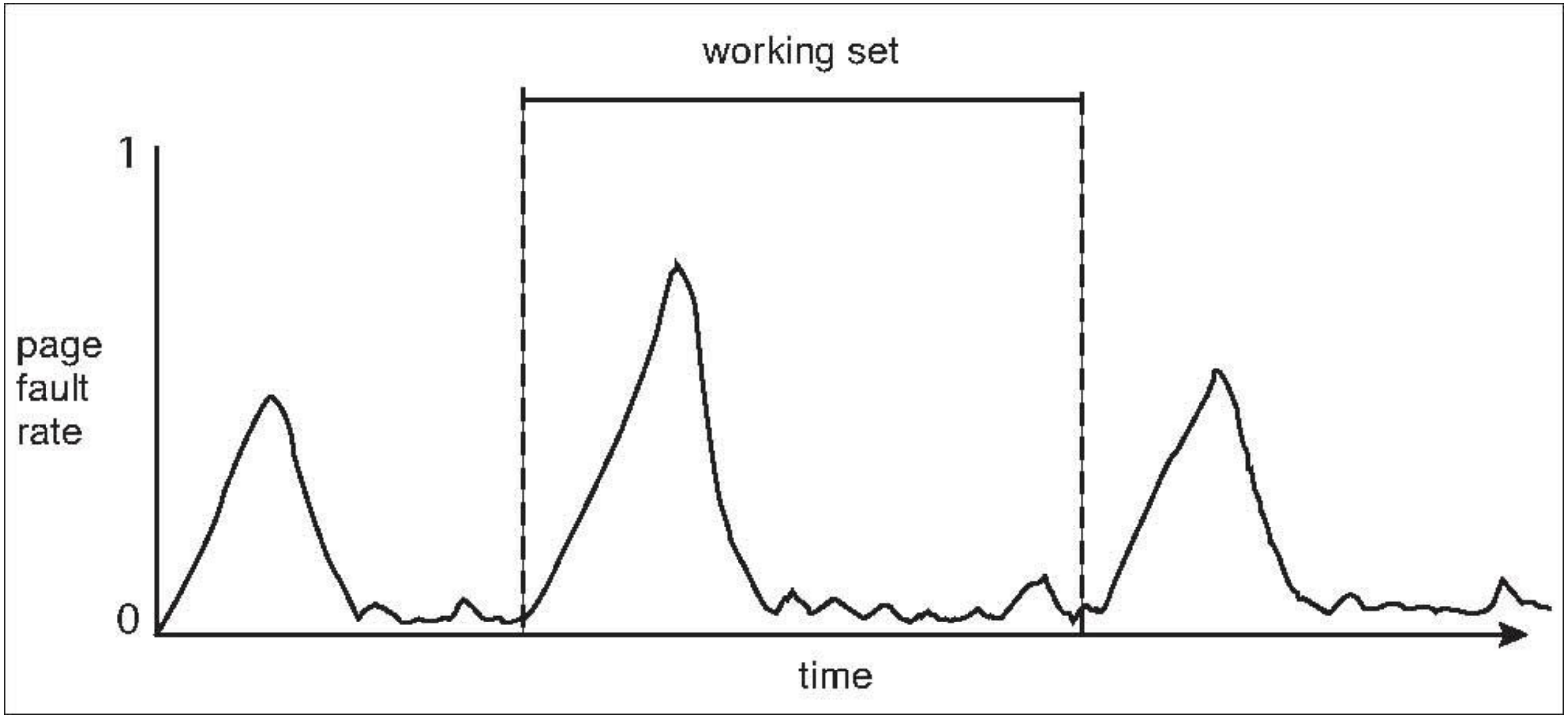
(Hypothetical) Working Set Allocation Algorithm

- Estimate |*w(k, 0)*| for a process
  - Allow process to start only if OS can provide that many frames
- Use a **local replacement algorithm**
  - Make sure that the working set are occupying the process's frames
- Track each process's working set size
  - Re-allocate page frames among processes dynamically

- How to keep track of processes' WSs?
  - Use reference bit with a fixed-interval timer interrupt

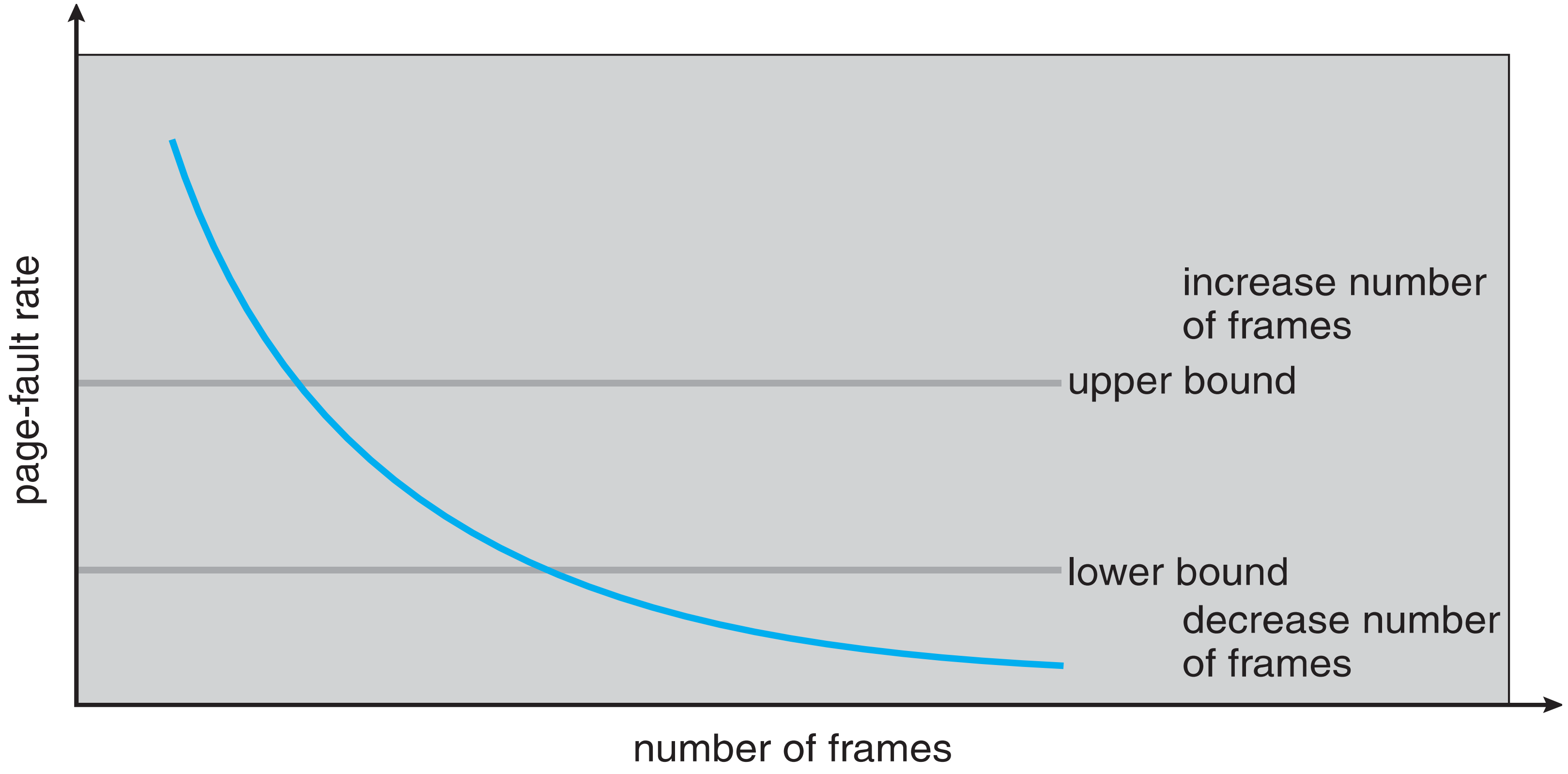
Working Sets and Page Fault Rates

- Relationship between **working set** and **page-fault rate** of a process
  - Working set changes over time
  - Page-fault rate peaks and then valley
- Can Page-fault rate/frequency be used to **steer allocations**?



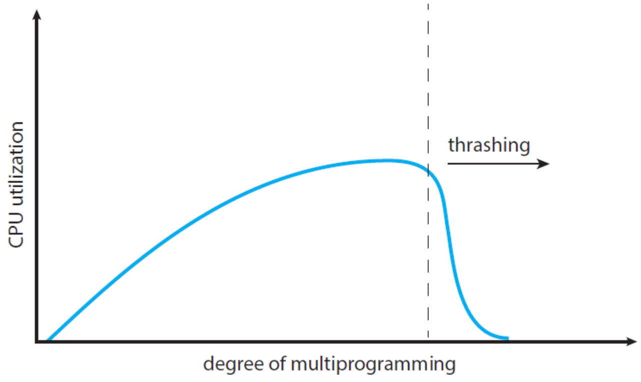
Page-Fault Frequency Allocation

- Establish “acceptable” **page-fault frequency** (**PFF**) rate
- Use a local **replacement algorithm**
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame



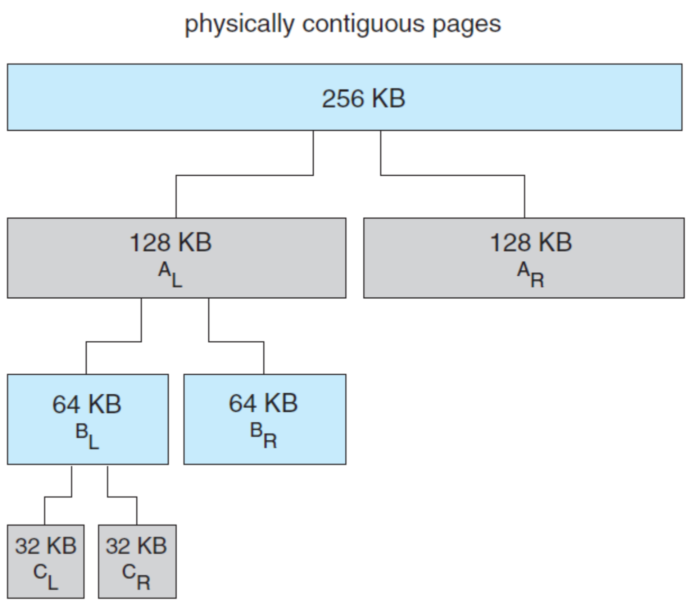
Thrashing

- System spends
  - Most of its time **servicing page faults**
  - Little time doing **useful work**
- Could be that there is **enough memory**
  - But a **poor replacement algorithm**
    - Incompatible with program behavior
- Could be that **memory is over-committed**
  - OS sees **CPU poorly utilized** and adds more processes
    - Many active processes, requesting memory



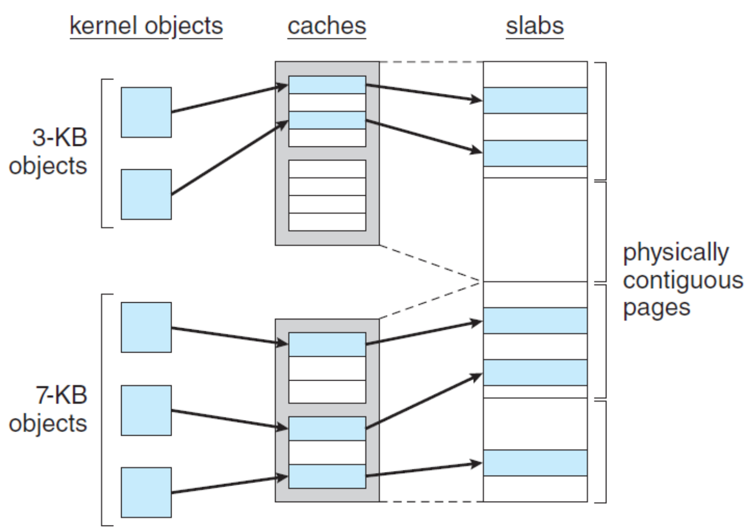
Kernel Memory Allocation: Buddy System

- Power of 2 allocator of **physically contiguous pages**
  - Satisfies requests in units sized as power of 2
  - Not power of 2 request size is rounded up
  - If request is smaller
    - Break down in 2 buddies that are also power of 2
  - Two equal size free buddies may be coalesced



Kernel Memory Allocation: Slab Allocation

- **Slab** is made up of one or more **physically contiguous pages**
- A **cache** consists of one or more slabs
- There is a cache for each **unique kernel data structure**
  - Each cache populated with objects
    - instantiations of the kernel data structure
- If there are **free slabs**, the allocation is immediate
  - No search for memory space
- SLOB and SLUB (more performant) variations in Linux



Summary

- Overlays
- Paged Virtual memory
- Page faults
- Demand paging
- Page replacement
  - FIFO, Optimal, LRU, Second Chance, Clock
  - local, global
- Locality
  - temporal, spatial
- Working set
- Thrashing
- Kernel Memory Allocation

CPU Cache vs. Virtual Memory “as a Cache”

- CPU Cache is a **hardware** component
  - It is **completely** transparent to the programmer
  - CPU cache **holds** data coming from memory
    - CPU cache fetches data from memory transparently
- Virtual memory “as a cache”, is a **hardware** component + **OS**
  - It is transparent to the application, but not to the OS
  - Virtual memory “as a cache” **holds** data coming from storage
    - The OS moves memory from storage to memory