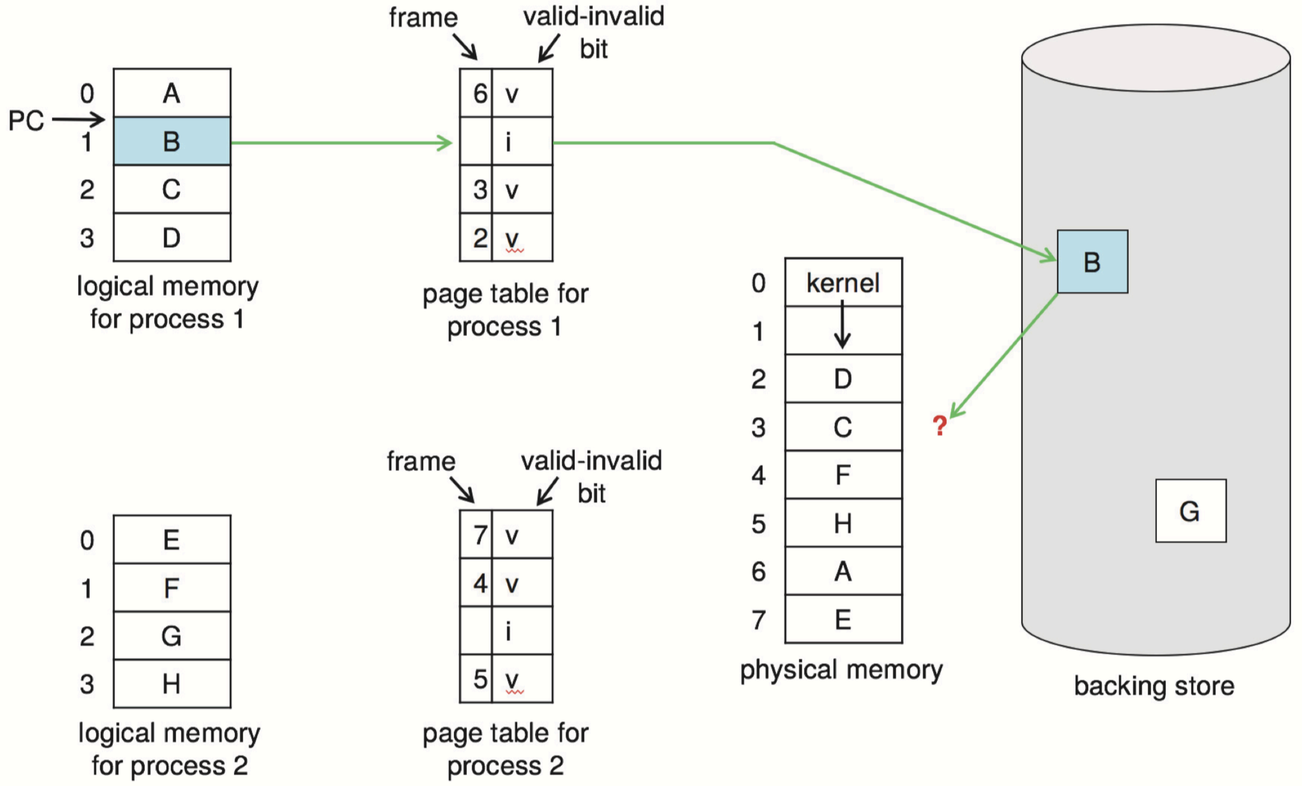
Virtual Memory (2)

Allocation of Frames

- Number of frames we allocate for a process is an important factor
- In this example, we allocate 3 frames for each process - this is fair
- Is it optimal? May be not
- P_1 is using all four pages, but P_2 is only using 2 pages
- In that case, 3, 3 allocation can lead to unnecessary page faults
- Minimum number of page frames needed by a process can also be dictated by the computer architecture



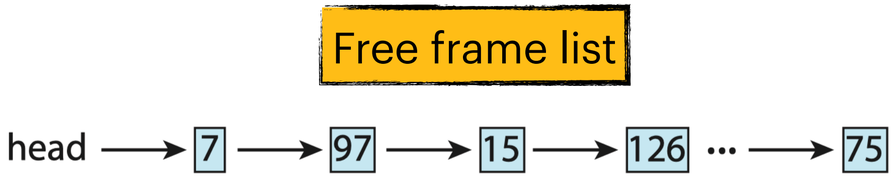
Allocation Algorithms

- Easiest allocation — equal allocation
- m frames are split among n processes equally — m/n frames per process
- In the above example, we need proportional allocation
- P_1 needs $4/6 * 6 = 4$ frames and P_2 needs $2/6 * 6 = 2$ frames, this way both processes have their needs satisfied in the optimal manner
- Equal and proportional allocation differs based on the multiprogramming level

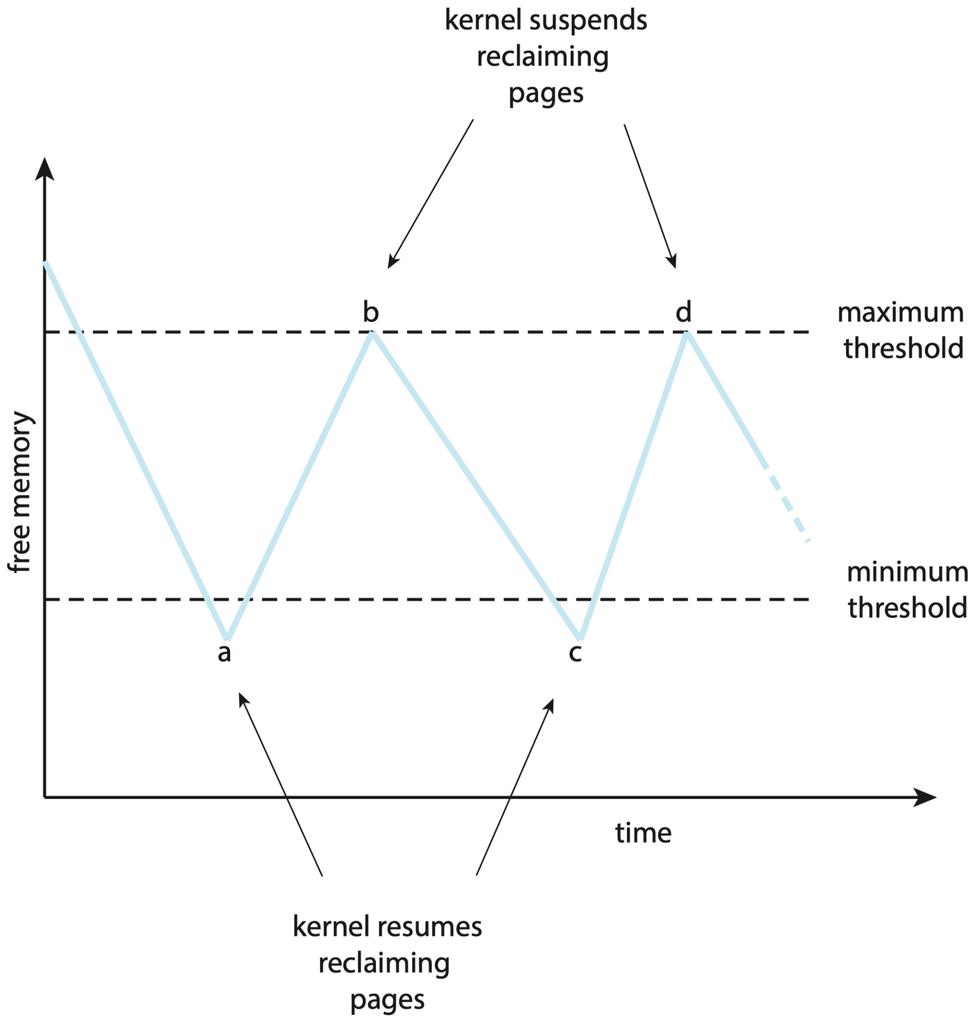
Global vs Local Allocation

- Frame allocation determines how many frames are allocated for a process
- There is another factor that determines the number of number of frames available for a process — how frames are being replaced
  - Local replacement — we replace a frame belonging to the process when we need to bring in a new page
  - Global replacement — we replace a frame belonging to any process when we need to bring in a new page
- With global replacement, number of frames allocated for a process can change
- A high priority process might use global replacement to get frames from low priority processes and increase its frame allocation

Reclaiming Pages

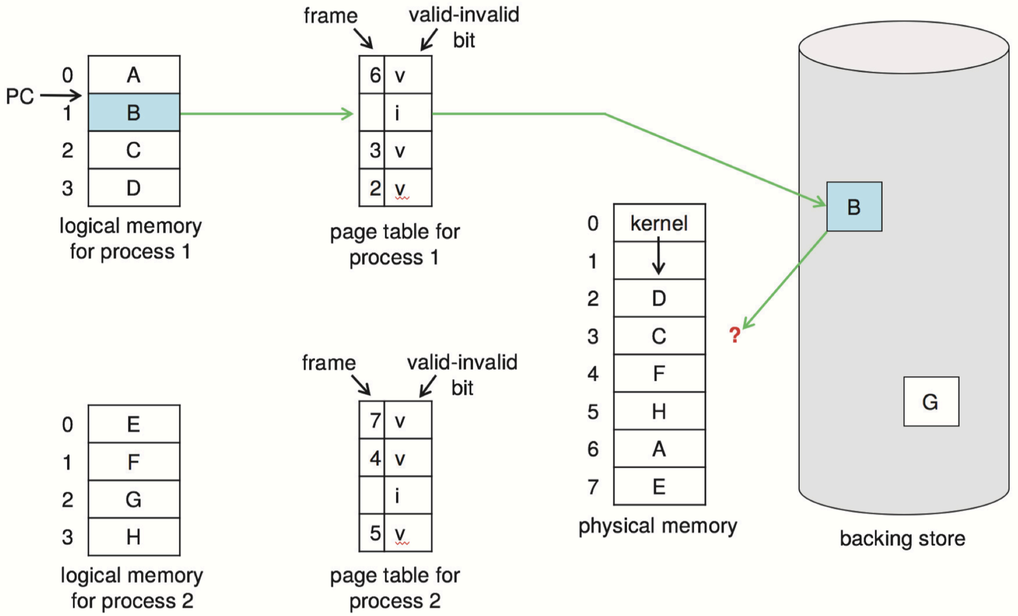


- Instead of waiting for free frame list to go to empty, we can periodically run a **reaper** process to reclaim memory
- Reaper can reclaim memory using LRU or any replacement strategies
- If it fails, Reaper can terminate processes (out-of-memory killer) when the free frame list is below a min threshold
- Reaper suspends when free frame list is above a threshold



Thrashing Problem

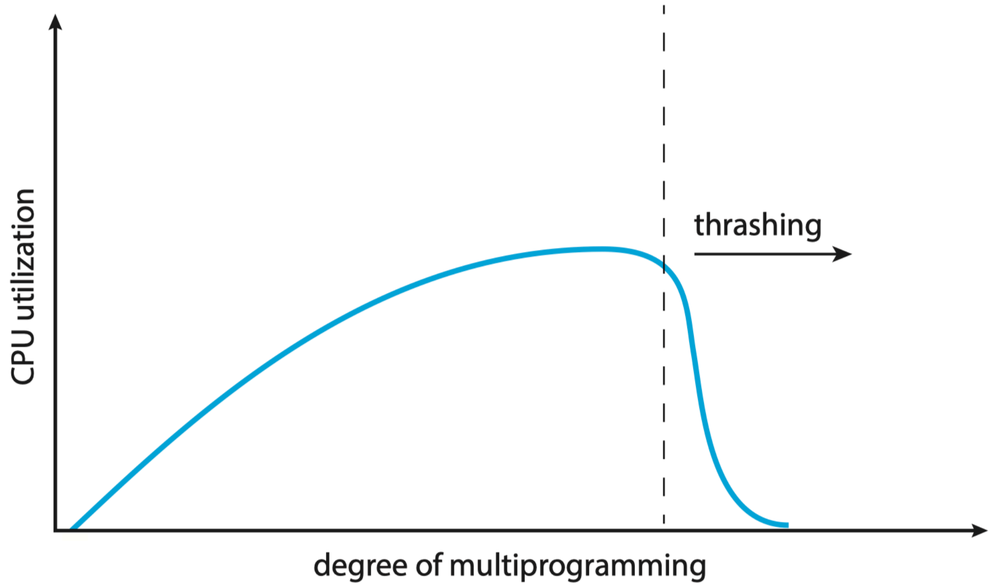
- Suppose P_1 wants to access pages A, B, C, D and then repeat it in a loop
- Suppose P_2 wants to access pages E, F, G, H and then repeat it in a loop
- We don't have enough frames to hold the pages ⇒ many faults happening
- To fix the problem:
  - Increase memory
  - Not run P_1 and P_2 at the same time
  - Run P_1 or P_2 and other processes that won't need as many pages to run



- Thrashing — CPU unable to run the program without waiting for pages to load
- Backing store (very slow), CPU is in constant state of wait for pages
- CPU utilization goes down

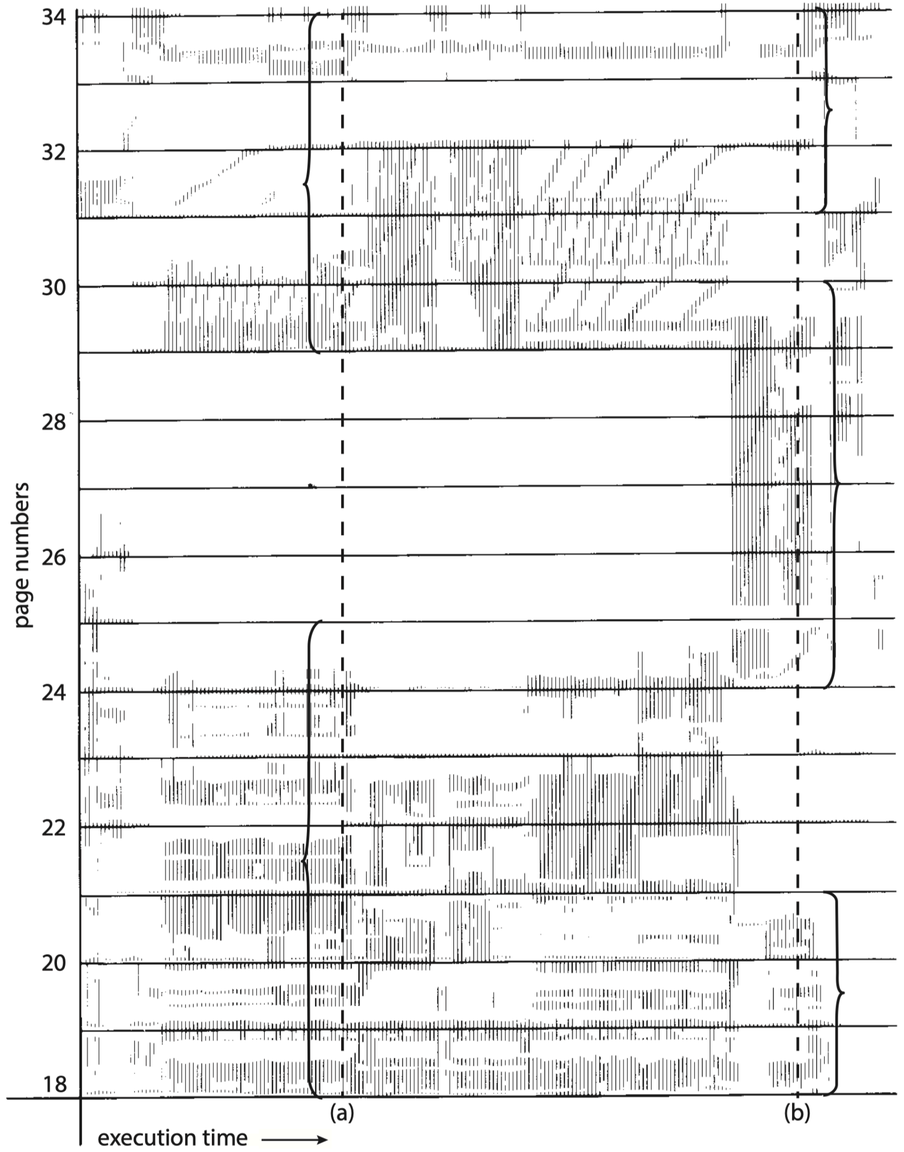
Thrashing

- Degree of multiprogramming (DoM) is the number of processes running concurrently in the system — long term scheduler decides this factor
- CPU utilization is low (not much work) — can increase utilization by increasing DoM — more work
- After a certain limit, not enough memory — lot of page faults — we start waiting on backing store
- Scheduler increase DoM to increase CPU utilization, more page faults — increase waiting times
- System enters into the state of **thrashing** — constant wait on backing store
- Fix: decrease DoM



Locality of Reference

- A program as it runs will access different portions of the memory
- At time (a), process is accessing pages {18, 19, 20, 21, 22, 23, 24, 29, 30, 33} (10 pages in the locality)
- At time (b), process is accessing pages {18, 19, 20, 24, 25, 26, 27, 28, 29, 31, 32, 33} (12 pages in the locality)
- Pages {18, 19, 20, 24, 29, 33} are in both localities
- If we allocate enough frames to hold the locality a process is in, faults happen until the pages are loaded, after that there won't be any faults
- Faults will resume after the program has moved to a different locality

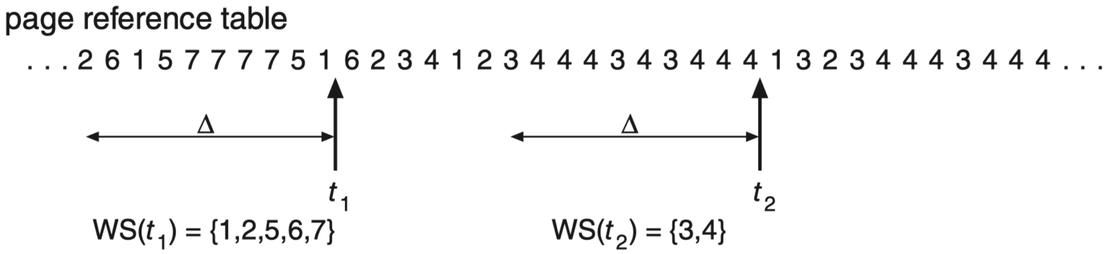


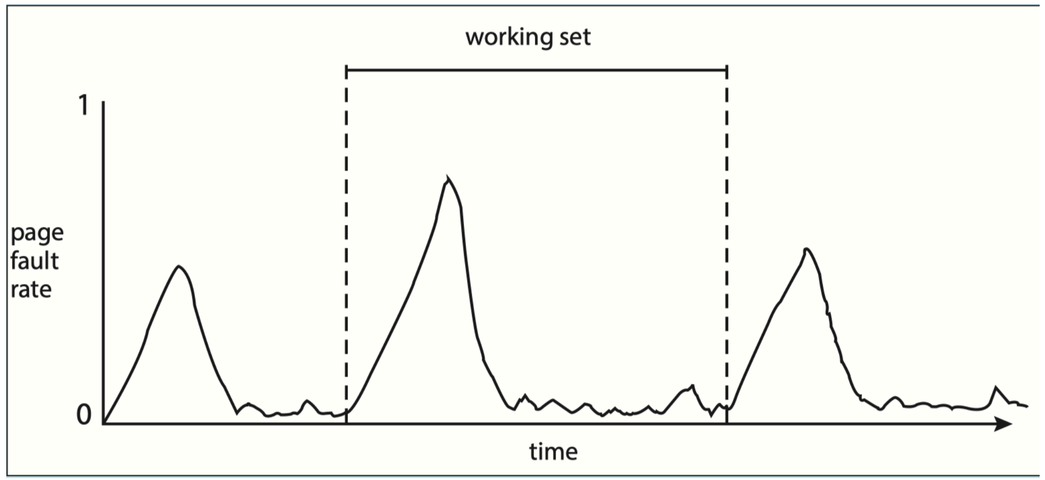
Locality of Reference and Page Faults

- If we have enough frames to hold all the pages in the current locality, we can minimize the page faults
- Page faults happen when we enter the locality — need to fault to load the pages
- While the program remains in that locality, page faults are not there or very minimal
- Challenge: how do we know the locality? Number of frames to allocate for a given locality and the pages making up the locality

Working Set Model

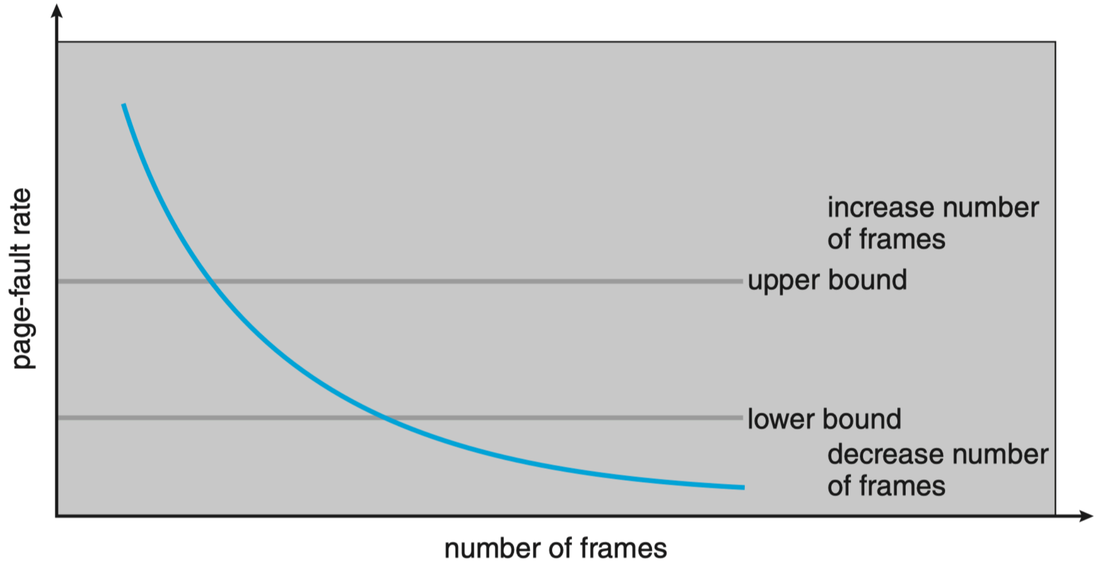
- Working-set model is based on the assumption of locality
- Working set window (Δ) worth of page references in the past are examined
- Working set for $\Delta = 10$ is shown on the right
- While the working set is being loaded - we can have page faults (demand paging)
- After all the pages are loaded, page faults should be minimal





Page-Fault Frequency

- Working set idea is great, but it is quite cumbersome to manage
- Page fault frequency of a process indicates that the process is happy with the number of frames allocated or not



Allocating Kernel Memory

- Buddy system allocator — allocate power of 2 blocks
- Segment is divided into two buddies
- Adjacent buddies are coalesced automatically at deallocation