Virtual Memory III

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

Today's Topics

- **What if the physical memory becomes full?**
  - Page replacement algorithms
- **How to manage memory among competing processes?**
- **Advanced virtual memory techniques**
  - Shared memory
  - Copy on write
  - Memory-mapped files

Page Replacement (1)

- **Page replacement**
  - When a page fault occurs, the OS loads the faulted page from disk into a page frame of memory.
  - At some point, the process has used all of the page frames it is allowed to use.
  - When this happens, the OS must replace a page for each page faulted in.
    - It must evict a page to free up a page frame.
  - The page replacement algorithm determines how this is done.

Page Replacement (2)

- **Evicting the best page**
  - The goal of the replacement algorithm is to reduce the fault rate by selecting the best victim page to remove.
  - The best page to evict is the one never touched again.
    - as process will never again fault on it.
  - "Never" is a long time, so picking the page closest to "never" is the next best thing
    - Belady's proof: Evicting the page that won't be used for the longest period of time minimizes the number of page faults.

Belady's Algorithm

- **Optimal page replacement**
  - Replace the page that will not be used for the longest time in the future.
  - Has the lowest fault rate for any page reference stream.
  - Problem: have to predict the future
  - Why is Belady's useful? – Use it as a yardstick!
    - Compare other algorithms with the optimal to gauge room for improvement.
    - If optimal is not much better, then algorithm is pretty good, otherwise algorithm could use some work.
    - Lower bound depends on workload, but random replacement is pretty bad.

FIFO (1)

- **First-In First-Out**
  - Obvious and simple to implement
    - Maintain a list of pages in order they were paged in
    - On replacement, evict the one brought in longest time ago
  - Why might this be good?
    - Maybe the one brought in the longest ago is not being used.
  - Why might this be bad?
    - Maybe, it's not the case.
    - We don't have any information either way.
  - FIFO suffers from "Belady's Anomaly"
    - The fault rate might increase when the algorithm is given more memory.

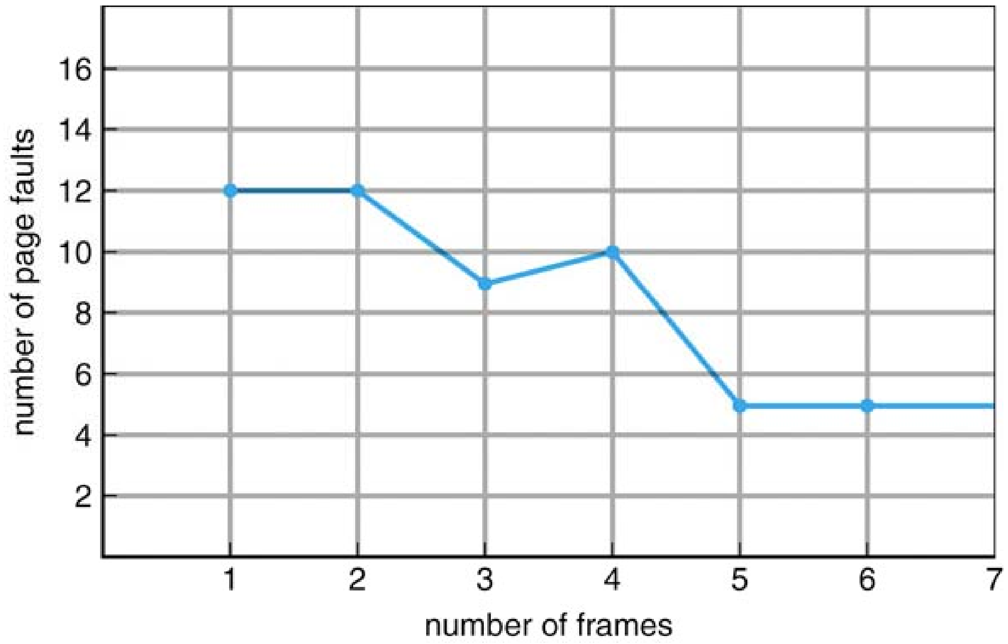
FIFO (2)

- **Example: Belady's anomaly**
  - Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
  - 3 frames: 9 faults

1	4	5
2	1	3
3	2	4

  - 4 frames: 10 faults

1	5	4
2	1	5
3	2	
4	3	



LRU (1)

- **Least Recently Used**
  - LRU uses reference information to make a more informed replacement decision.
    - Idea: past experience gives us a guess of future behavior.
    - On replacement, evict the page that has not been used for the longest time in the past.
    - LRU looks at the past, Belady's wants to look at future.
  - Implementation
    - Counter implementation: put a timestamp
    - Stack implementation: maintain a stack
  - Why do we need an approximation?

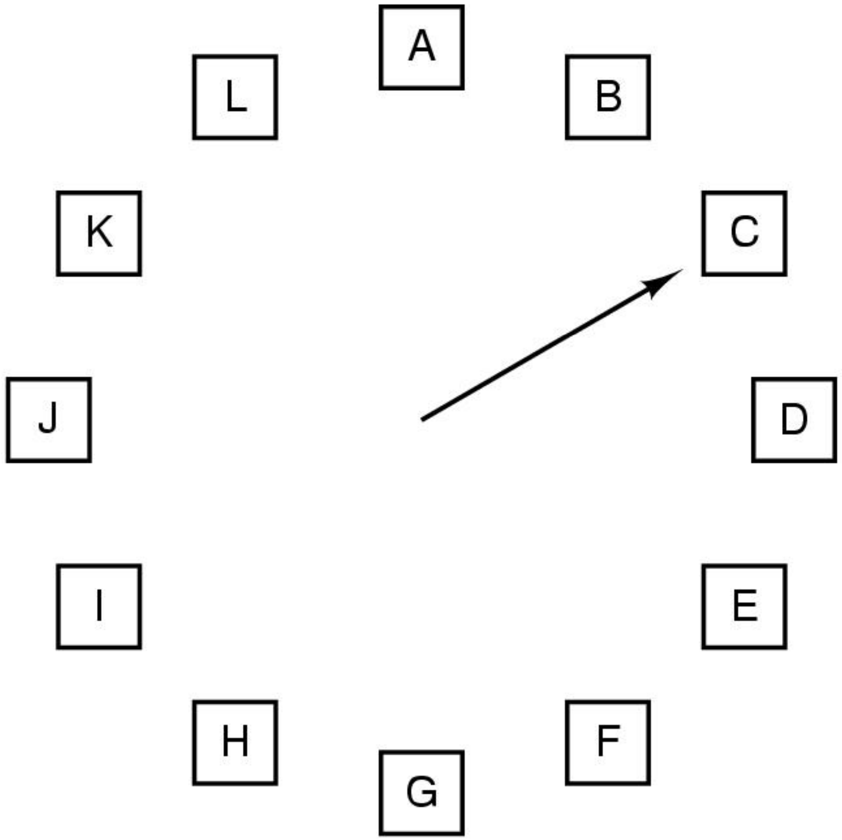
LRU (2)

- **Approximating LRU**
  - Many LRU approximations use the PTE reference (R) bit.
    - R bit is set whenever the page is referenced (read or written)
  - Counter-based approach
    - Keep a counter for each page.
    - At regular intervals, for every page, do:
      - If R = 0, increment the counter (hasn't been used)
      - If R = 1, zero the counter (has been used)
      - Zero the R bit
    - The counter will contain the number of intervals since the last reference to the page.
    - The page with the largest counter is the least recently used.
  - Some architectures don't have a reference bit.
    - Can simulate reference bit using the valid bit to induce faults.

Second Chance (1)

- **Second chance or LRU clock**
  - FIFO with giving a second chance to a recently referenced page.
  - Arrange all of physical page frames in a big circle (clock).
  - A clock hand is used to select a good LRU candidate.
    - Sweep through the pages in circular order like a clock
    - If the R bit is off, it hasn't been used recently and we have a victim.
    - If the R bit is on, turn it off and go to next page.
  - Arm moves quickly when pages are needed.
    - Low overhead if we have plenty of memory.
    - If memory is large, "accuracy" of information degrades.

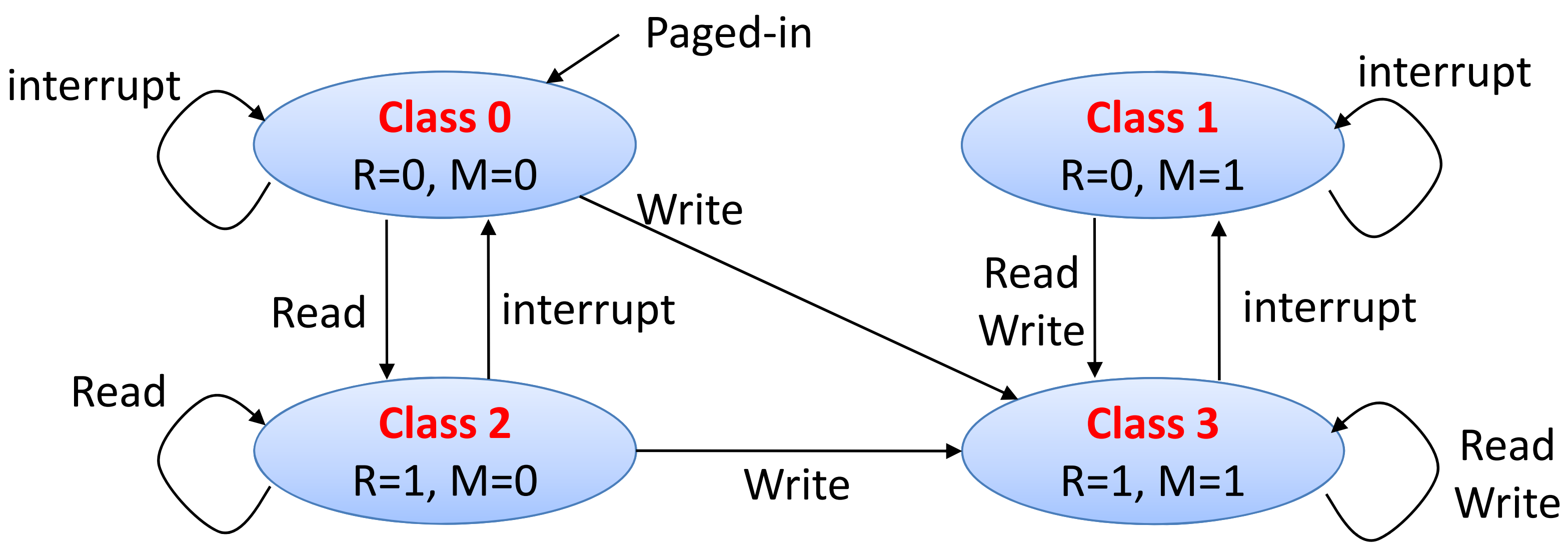
Second Chance (2)



When a page fault occurs,
the page the hand is
pointing to is inspected.
The action taken depends
on the R bit:
R = 0: Evict the page
R = 1: Clear R and advance hand

Not Recently Used (1)

- **NRU or enhanced second chance**
  - Use R (reference) and M (modify) bits
    - Periodically, (e.g., on each clock interrupt), R is cleared, to distinguish pages that have not been referenced recently from those that have been.



Not Recently Used (2)

- **Algorithm**
  - Removes a page at random from the lowest numbered nonempty class.
  - It is better to remove a modified page that has not been referenced in at least one clock tick than a clean page that is in heavy use.
  - Used in Macintosh.
- **Advantages**
  - Easy to understand.
  - Moderately efficient to implement.
  - Gives a performance that, while certainly not optimal, may be adequate.

LFU (1)

- **Counting-based page replacement**
  - A software counter is associated with each page.
  - At each clock interrupt, for each page, the R bit is added to the counter.
    - The counters denote how often each page has been referenced.
- **Least frequently used (LFU)**
  - The page with the smallest count will be replaced.
  - (cf.) Most frequently used (MFU) page replacement
    - The page with the largest count will be replaced
    - Based on the argument that the page with the smallest count was probably just brought in and has yet to be used.
  - It never forgets anything.
    - A page may be heavily used during the initial phase of a process, but then is never used again

LFU (2)

- **Aging**
  - The counters are shifted right by 1 bit before the R bit is added to the leftmost.

	(a)	(b)	(c)	(d)	(e)
R bits for pages 0-5	R bits for pages 0-5, clock tick 0	R bits for pages 0-5, clock tick 1	R bits for pages 0-5, clock tick 2	R bits for pages 0-5, clock tick 3	R bits for pages 0-5, clock tick 4
	1 0 1 0 1 1	1 1 0 0 1 0	1 1 0 1 0 1	1 0 0 0 1 0	0 1 1 0 0 0
Page 0	10000000	11000000	11100000	11110000	01111000
1	00000000	10000000	11000000	01100000	10110000
2	10000000	01000000	00100000	00100000	10001000
3	00000000	00000000	10000000	01000000	00100000
4	10000000	11000000	01100000	10110000	01011000
5	10000000	01000000	10100000	01010000	00101000

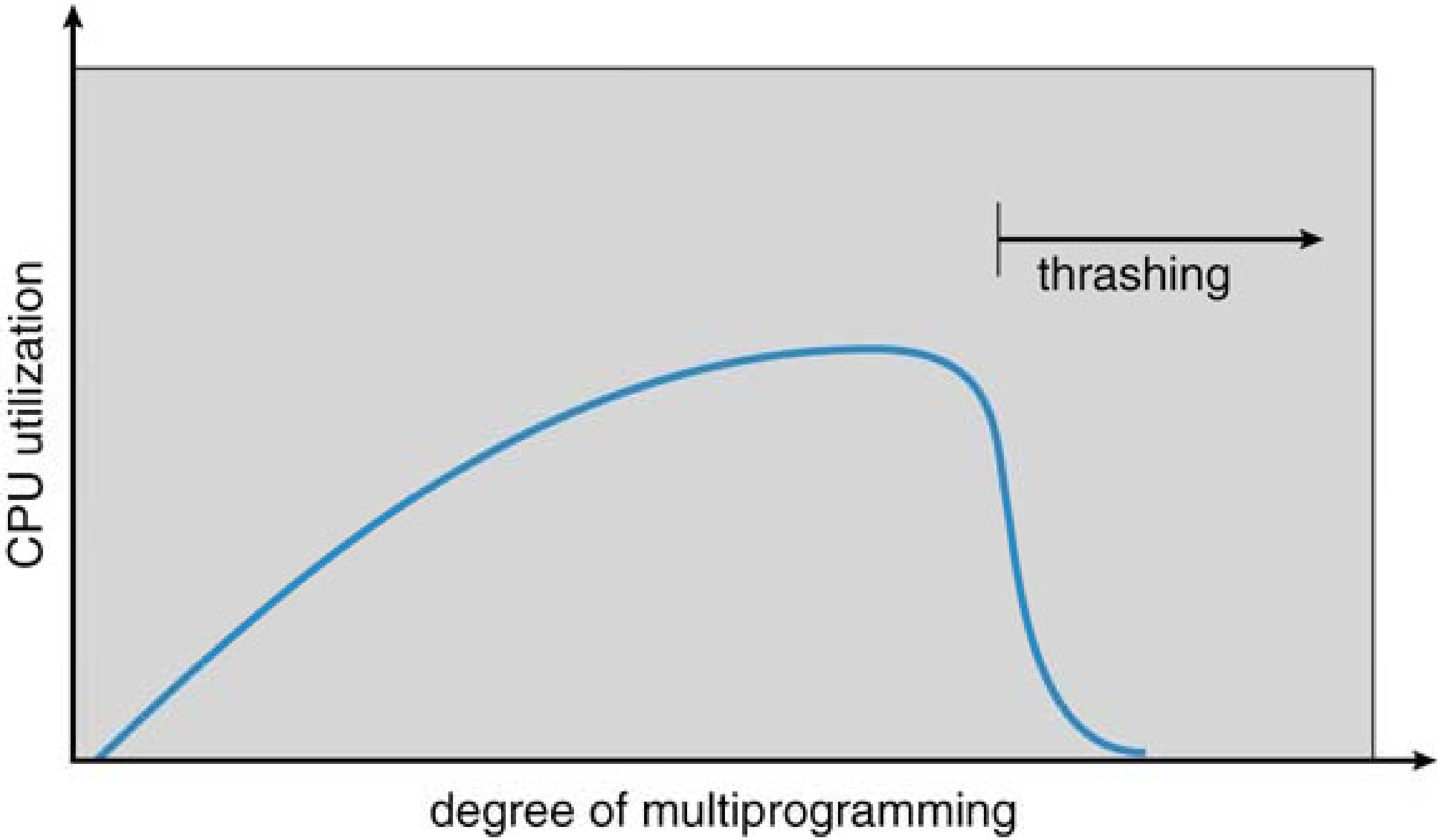
Allocation of Frames

- **Problem**
  - In a multiprogramming system, we need a way to allocate physical memory to competing processes.
    - What if a victim page belongs to another process?
    - How to determine how much memory to give to each process?
  - Fixed space algorithms
    - Each process is given a limit of pages it can use.
    - When it reaches its limit, it replaces from its own pages.
    - Local replacement: some process may do well, others suffer.
  - Variable space algorithms
    - Processes' set of pages grows and shrinks dynamically.
    - Global replacement: one process can ruin it for the rest (Linux)

Thrashing (1)

- **Thrashing**
  - What the OS does if page replacement algorithms fail.
  - Most of the time is spent by an OS paging data back and forth from disk.
    - No time is spent doing useful work.
    - The system is overcommitted.
    - No idea which pages should be in memory to reduce faults.
    - Could be that there just isn't enough physical memory for all processes.
  - Possible solutions
    - Swapping – write out all pages of a process
    - Buy more memory.

Thrashing (2)



Working Set Model (1)

- **Working set**
  - A working set of a process is used to model the dynamic locality of its memory usage.
    - i.e., working set = set of pages process currently "needs"
    - Peter Denning, 1968.
  - Definition
    - WS(t,w) = {pages P such that P was referenced in the time interval (t, t-w)}
    - t: time, w: working set window size (measured in page references)
  - A page is in the working set only if it was referenced in the last w references.

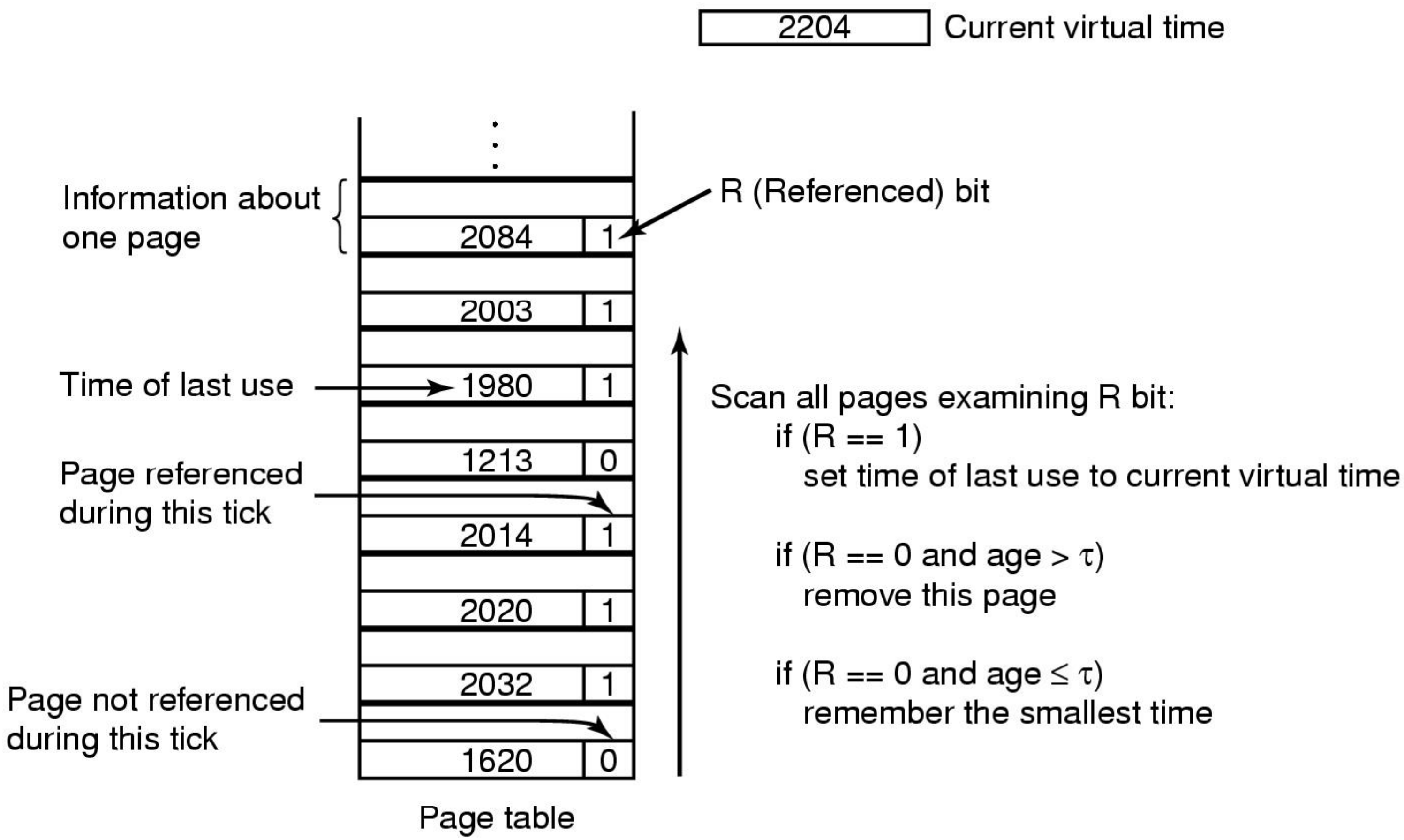
Working Set Model (2)

- **Working set size (WSS)**
  - The number of pages in the working set
    = The number of pages referenced in the interval (t, t-w)
  - The working set size changes with program locality.
    – During periods of poor locality, more pages are referenced.
    – Within that period of time, the working set size is larger.
  - Intuitively, working set must be in memory to prevent heavy faulting (thrashing).
  - Controlling the degree of multiprogramming based on the working set:
    – Associate parameter "wss" with each process.
    – If the sum of "wss" exceeds the total number of frames, suspend a process.
    – Only allow a process to start if its "wss", when added to all other processes, still fits in memory.
    – Use a local replacement algorithm within each process.

Working Set Model (3)

- **Working set page replacement**
  - Maintaining the set of pages touched in the last k references is expensive.
  - Approximate the working set as the set of pages used during the past time interval.
    - Measured using the current virtual time: the amount of CPU time a process has actually used.
  - Find a page that is not in the working set and evict it.
    - Associate the "Time of last use (Tlast)" field in each PTE.
    - A periodic clock interrupt clears the R bit.
    - On every page fault, the page table is scanned to look for a suitable page to evict.
    - If R = 1, timestamp the current virtual time (Tlast ← Tcurrent).
    - If R = 0 and (Tcurrent – Tlast) > τ, evict the page.
    - Otherwise, remember the page with the greatest age.

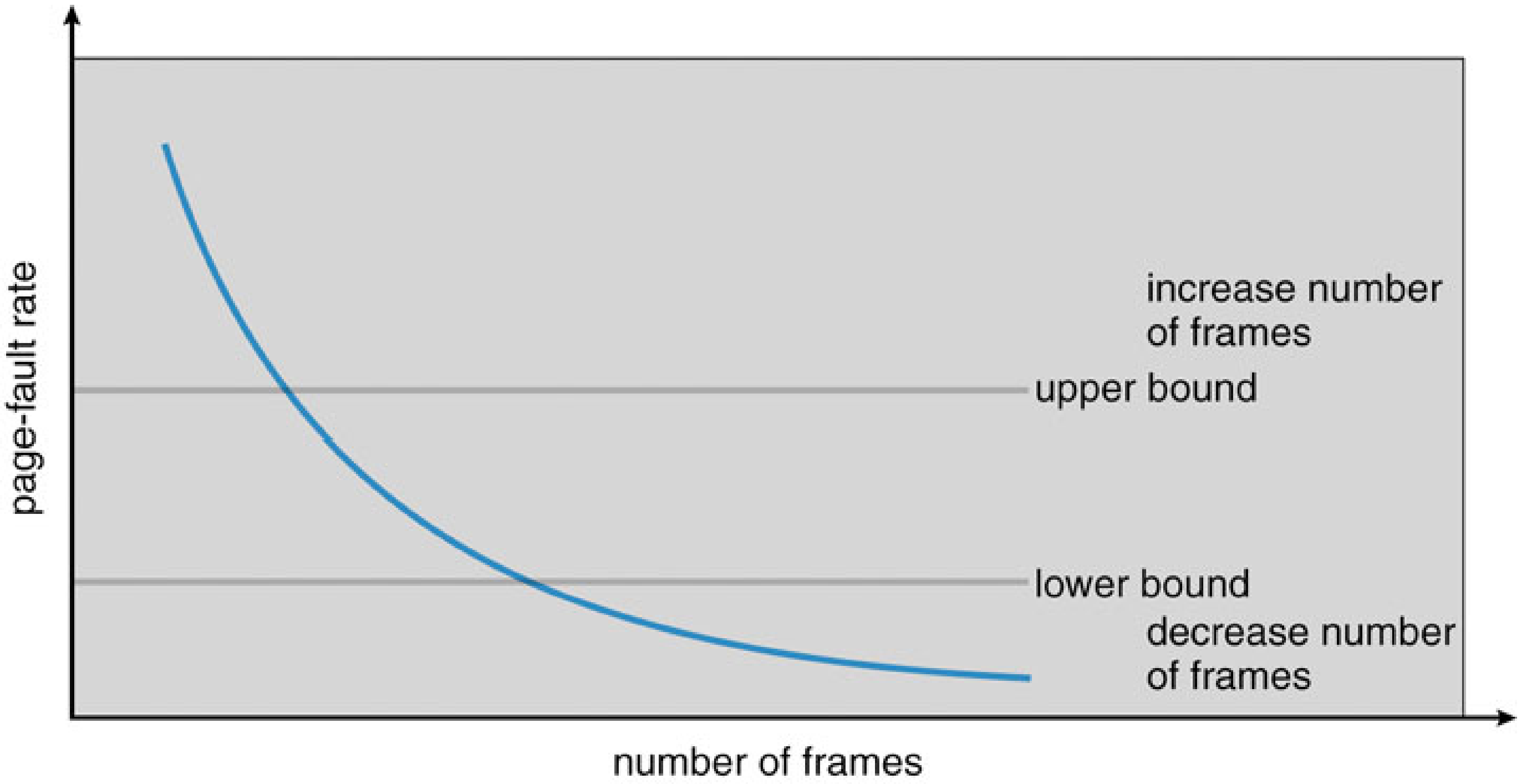
Working Set Model (4)



PFF (1)

- **Page Fault Frequency**
  - A variable space algorithm that uses a more ad-hoc approach.
    - Monitor the fault rate for each process.
    - If the fault rate is above a high threshold, give it more memory, so that it faults less (but not always – FIFO, Belady's anomaly).
    - If the fault rate is below a low threshold, take away memory (again, not always).
  - If the PFF increases and no free frames are available, we must select some process and suspend it.

PFF (2)



page-fault rate
increase number
of frames
upper bound
lower bound
decrease number
of frames
number of frames

Advanced VM Functionality

- **Virtual memory tricks**
  - Shared memory
  - Copy on write
  - Memory-mapped files

Shared Memory (1)

- **Shared memory**
  - Private virtual address spaces protect applications from each other.
  - But this makes it difficult to share data.
    - Parents and children in a forking Web server or proxy will want to share an in-memory cache without copying.
    - Read/Write (access to share data)
      Execute (shared libraries)
  - We can use shared memory to allow processes to share data using direct memory reference.
    - Both processes see updates to the shared memory segment.
    - How are we going to coordinate access to shared data?

Shared Memory (2)

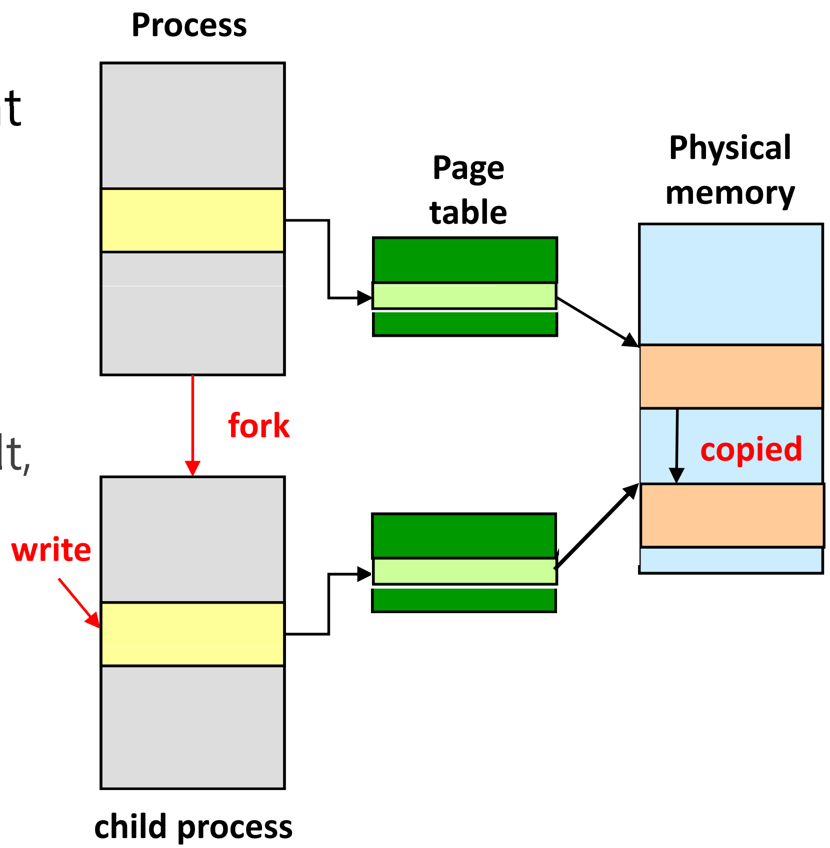
- **Implementation**
  - How can we implement shared memory using page tables?
    - Have PTEs in both tables map to the same physical frame.
    - Each PTE can have different protection values.
    - Must update both PTEs when page becomes invalid.
  - Can map shared memory at same or different virtual addresses in each process' address space
    - Different: Flexible (no address space conflicts), but pointers inside the shared memory segment are invalid.
    - Same: Less flexible, but shared pointers are valid.

Copy On Write (1)

- **Process creation**
  - requires copying the entire address space of the parent process to the child process.
  - Very slow and inefficient!
- **Solution 1: Use threads**
  - Sharing address space is free.
- **Solution 2: Use vfork() system call**
  - vfork() creates a process that shares the memory address space of its parent.
  - To prevent the parent from overwriting data needed by the child, the parent's execution is blocked until the child exits or executes a new program.
  - Any change by the child is visible to the parent once it resumes.
  - Useful when the child immediately executes exec().

Copy On Write (2)

- **Solution 3: Copy On Write (COW)**
  - Instead of copying all pages, create shared mappings of parent pages in child address space.
  - Shared pages are protected as read-only in child.
    - Reads happen as usual
    - Writes generate a protection fault, trap to OS, and OS copies the page, changes page mapping in client page table, restarts write instruction



Memory-Mapped Files (1)

- **Memory-mapped files**
  - Mapped files enable processes to do file I/O using memory references.
    - Instead of open(), read(), write(), close()
  - mmap(): bind a file to a virtual memory region
    - PTEs map virtual addresses to physical frames holding file data
    - <Virtual address base + N> refers to offset N in file
  - Initially, all pages in mapped region marked as invalid.
    - OS reads a page from file whenever invalid page is accessed.
    - OS writes a page to file when evicted from physical memory.
    - If page is not dirty, no write needed.

Memory-Mapped Files (2)

- **Note:**
  - File is essentially backing store for that region of the virtual address space (instead of using the swap file).
  - Virtual address space not backed by "real" files also called "anonymous VM".
- **Advantages**
  - Uniform access for files and memory (just use pointers)
  - Less copying
  - Several processes can map the same file allowing the pages in memory to be shared.
- **Drawbacks**
  - Process has less control over data movement.
  - Does not generalize to streamed I/O (pipes, sockets, etc.)

Summary (1)

- **VM mechanisms**
  - Physical and virtual addressing
  - Partitioning, Paging, Segmentation
  - Page table management, TLBs, etc.
- **VM policies**
  - Page replacement algorithms
  - Memory allocation policies
- **VM requires hardware and OS support**
  - MMU (Memory Management Unit)
  - TLB (Translation Lookaside Buffer)
  - Page tables, etc.

Summary (2)

- **VM optimizations**
  - Demand paging (space)
  - Managing page tables (space)
  - Efficient translation using TLBs (time)
  - Page replacement policy (time)

- **Advanced functionality**
  - Sharing memory
  - Copy on write
  - Mapped files