CS 153
Design of Operating Systems

Fall 21

Lecture 13: Virtual Memory

Instructor: Chengyu Song

Some slides modified from originals by Dave O'hallaron

Core i7 Level 1-3 Page Table Entries

63	62–52	51–12	11–9	8	7	6	5	4	3	2	1	0
XD	Unused	Page table physical base address	Unused	G	PS		A	CD	WT	U/S	R/W	P=1

Available for OS (page table location on disk)	P=0

P: Child page table present in physical memory (1) or not (0).

R/W: Read-only or read-write access access permission for all reachable pages.

U/S: user or supervisor (kernel) mode access permission for all reachable pages.

WT: Write-through or write-back cache policy for the child page table.

CD: Caching disabled or enabled for the child page table.

A: Reference bit (set by MMU on reads and writes, cleared by software).

PS: Page size either 4 KB or 2 MB (defined for Level 1 PTEs only).

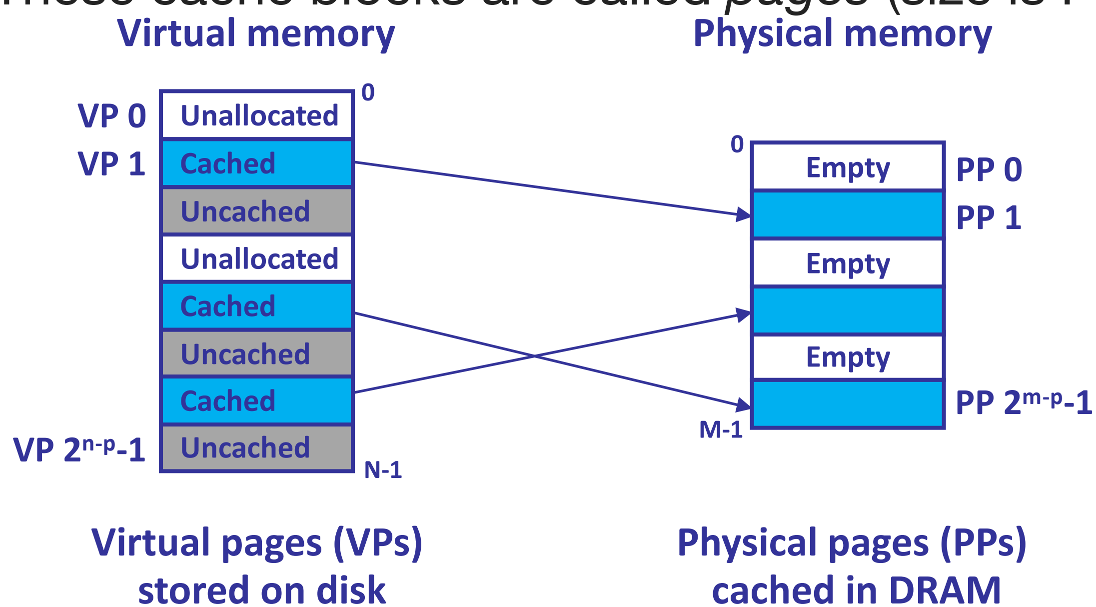
G: Global page (don’t evict from TLB on task switch)

Page table physical base address: 40 most significant bits of physical page table address (forces page tables to be 4KB aligned)

XD: Non-executable pages

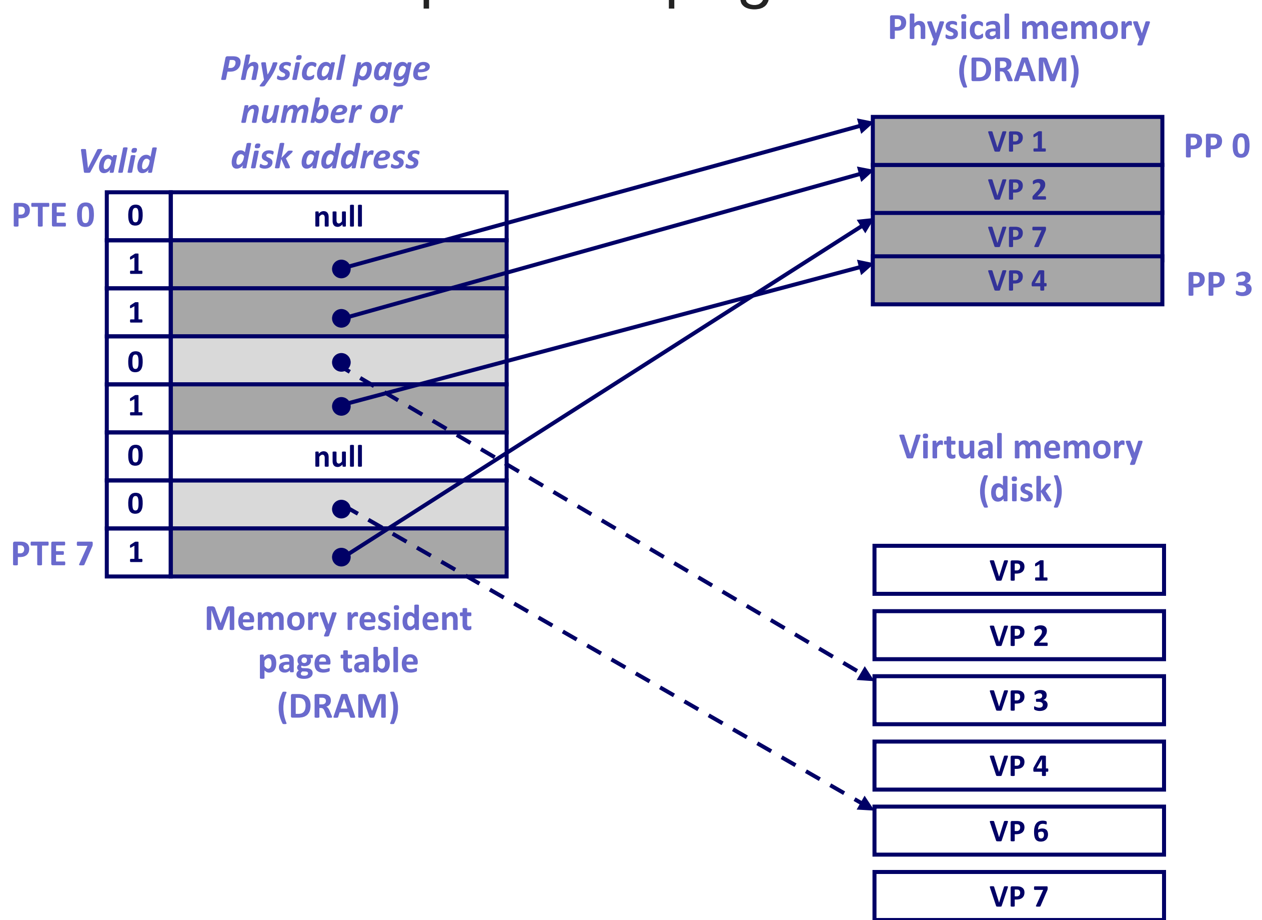
VM as a Tool for Caching

- Virtual memory is an array of N contiguous bytes stored on disk.
- The contents of the array on disk are cached in physical memory (DRAM cache)
  - These cache blocks are called *pages* (size is $P = 2^p$ bytes)



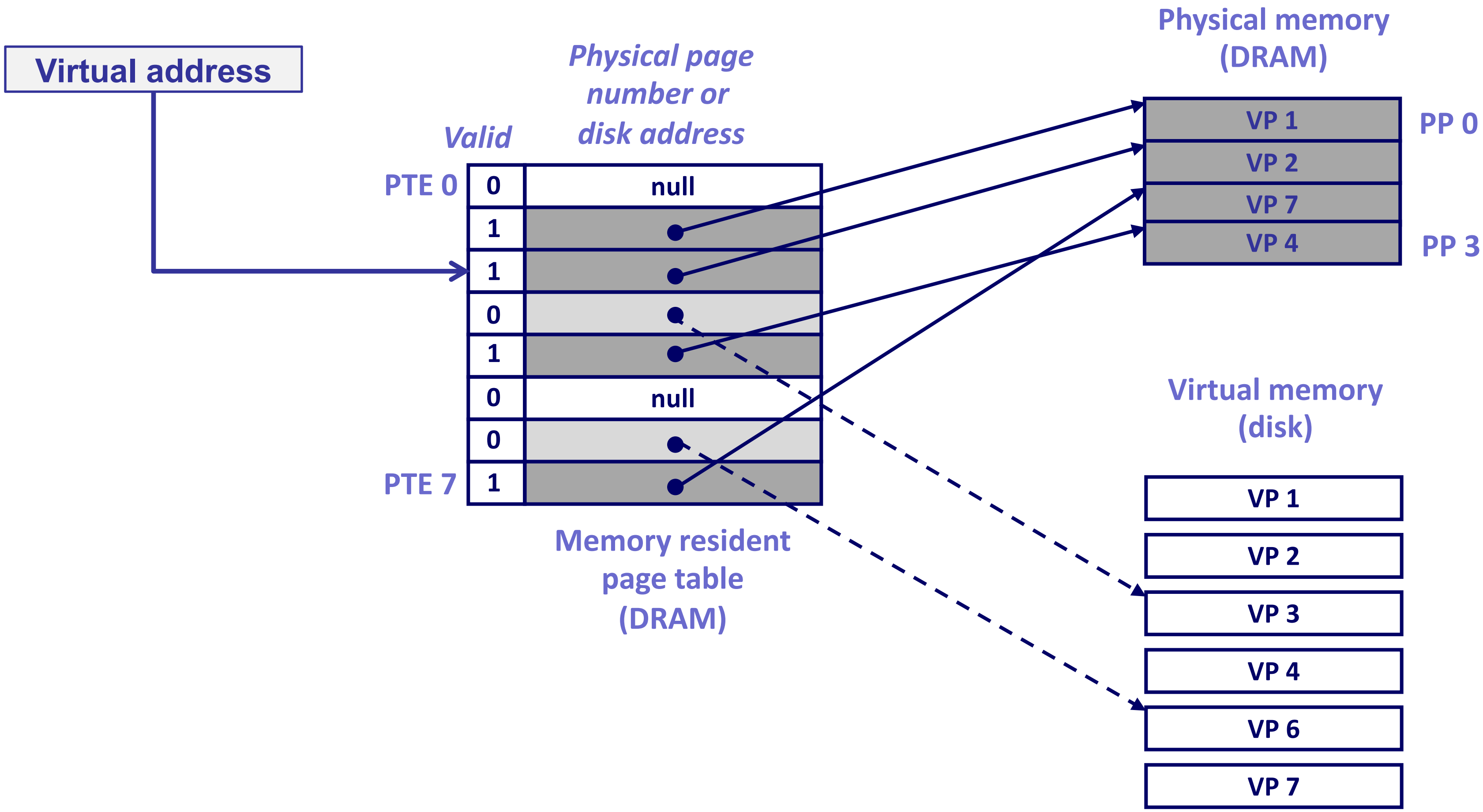
Page Table Setup

- Valid PTEs map virtual pages to physical pages.
- Invalid PTEs map virtual pages to disk blocks



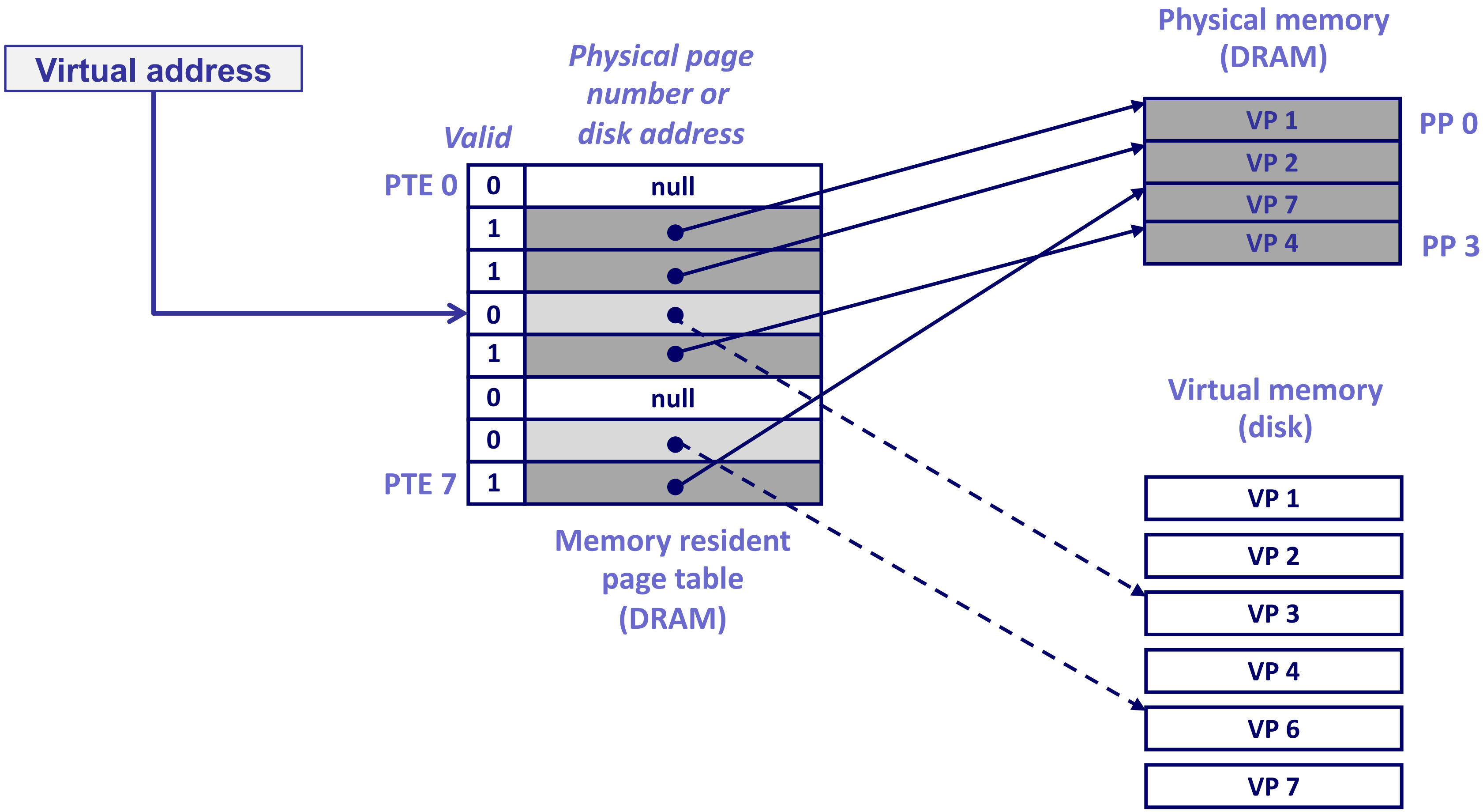
Page/Cache Hit

- Page hit: reference to VM word that is in physical memory (DRAM cache hit)



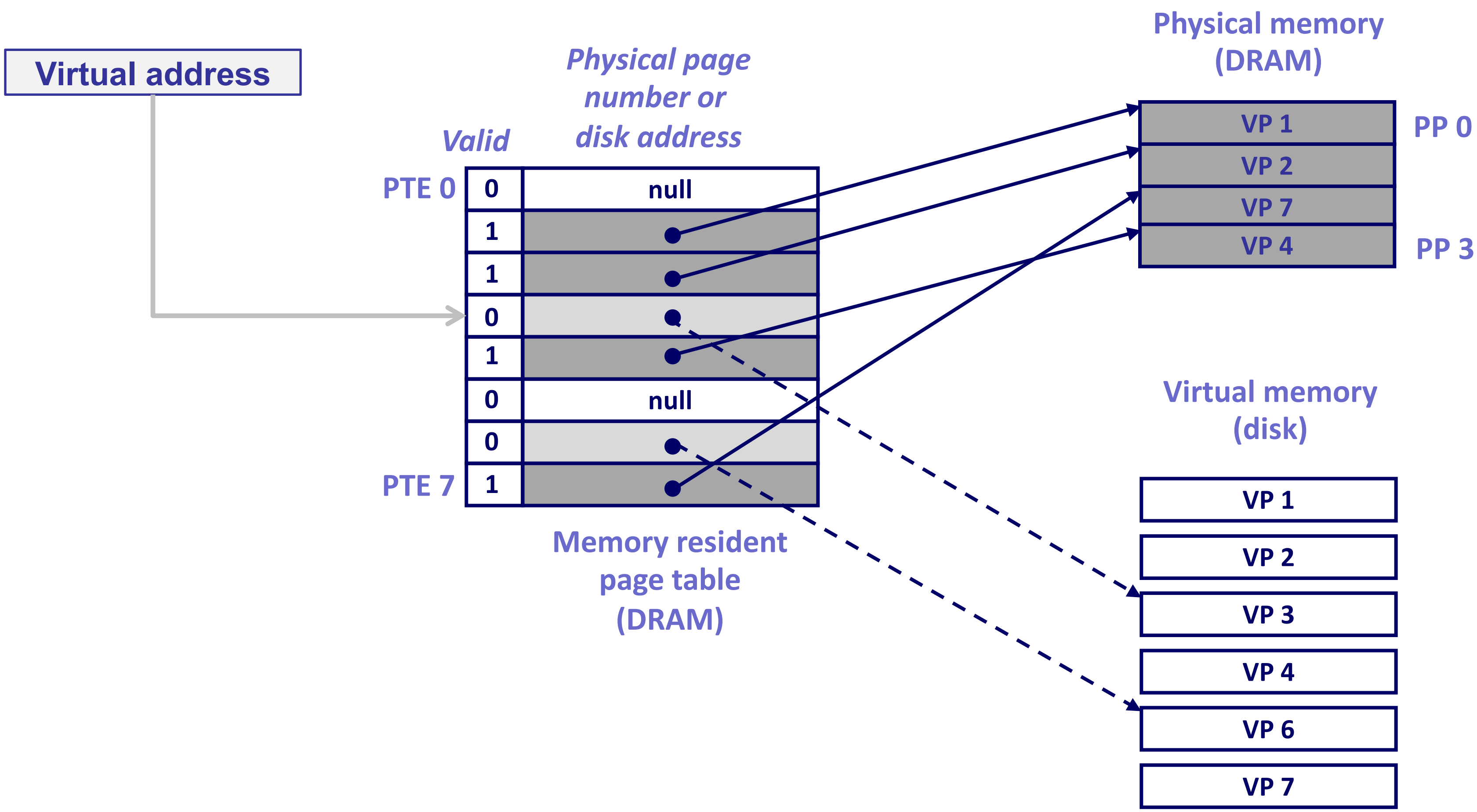
Page Fault (Cache miss)

- Page fault: reference to VM word that is not in physical memory (DRAM cache miss)



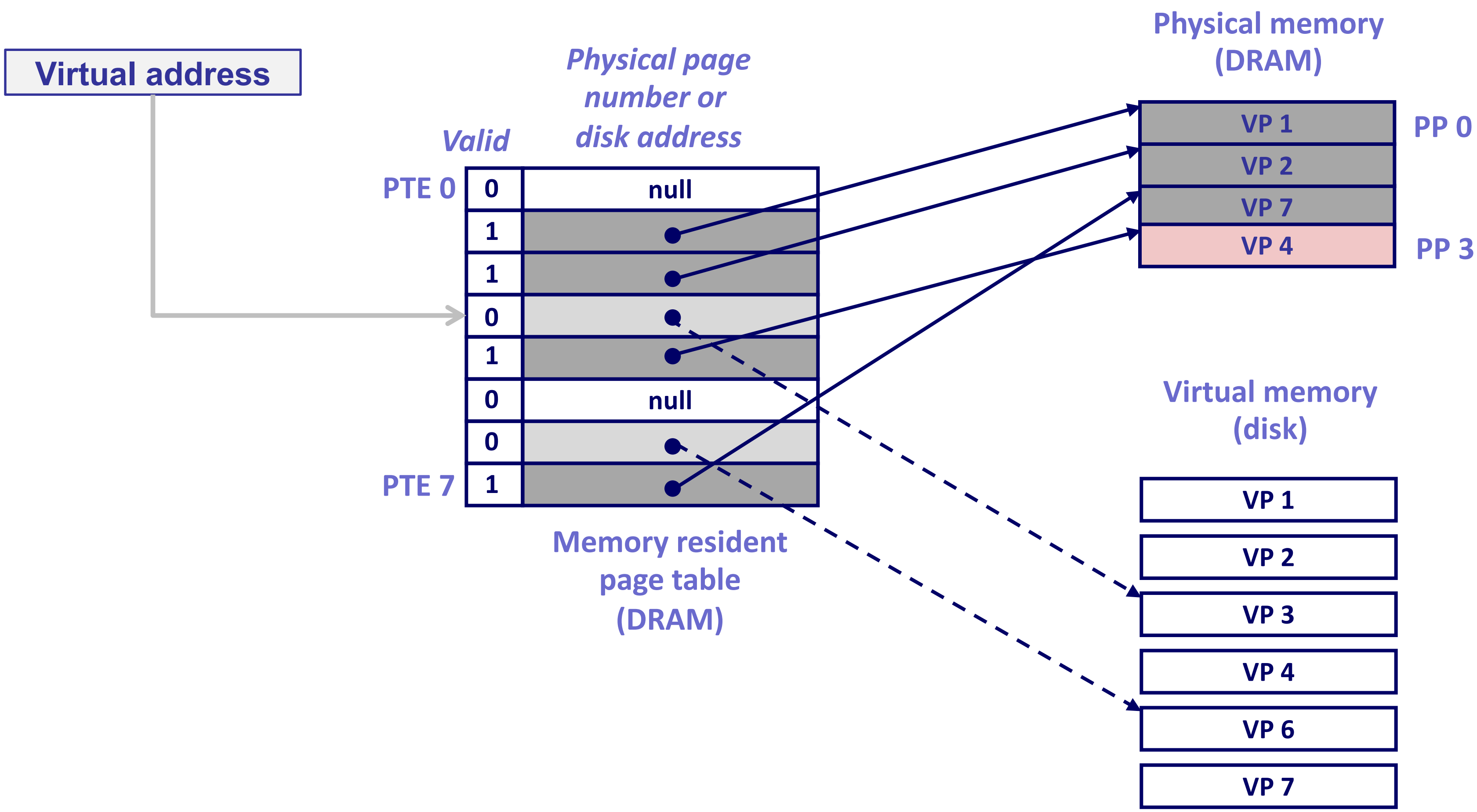
Handling Page Fault (1)

- Page miss causes page fault (an exception)



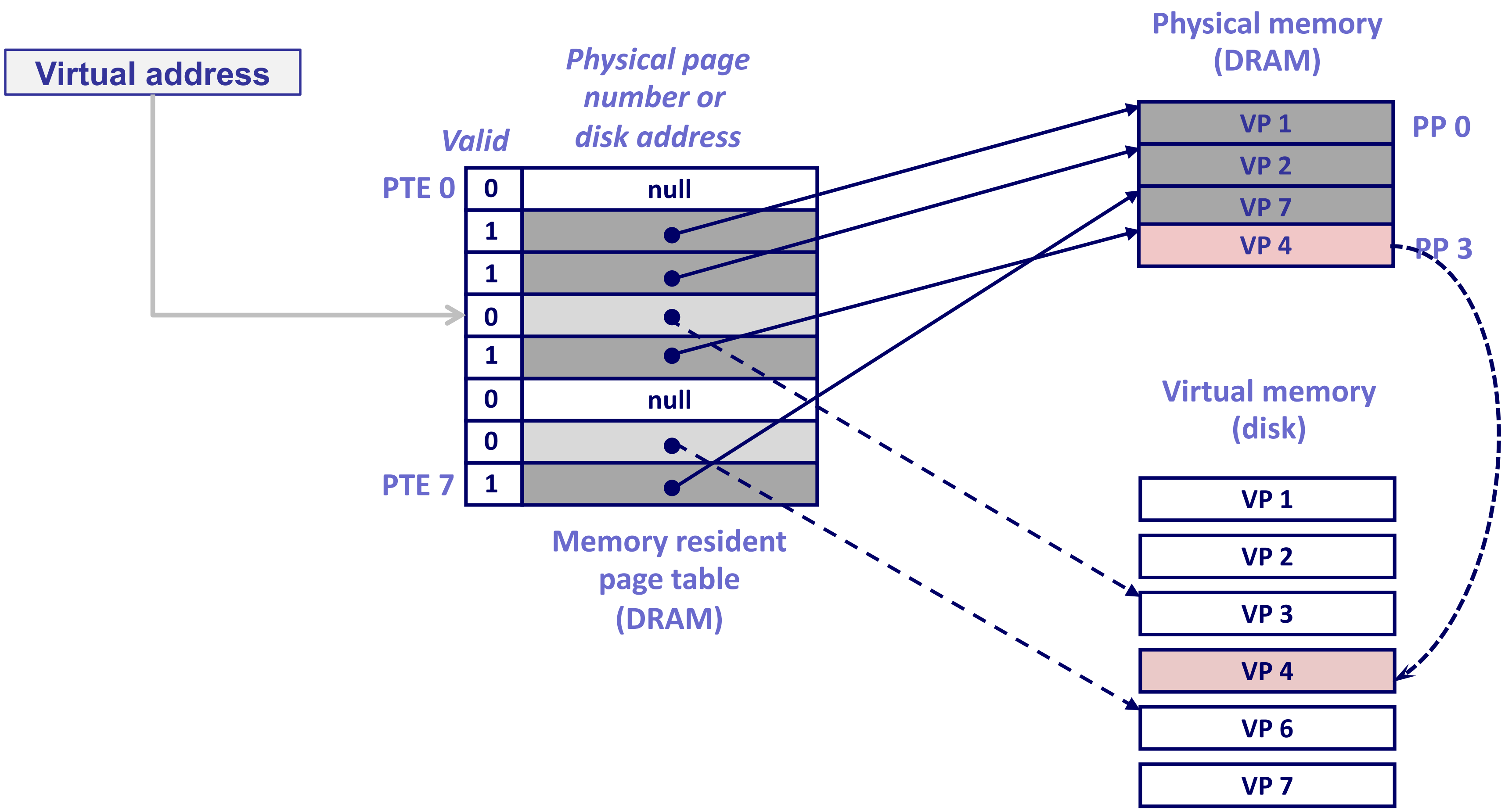
Handling Page Fault (2)

- Page fault handler selects a victim to be evicted (here VP 4)



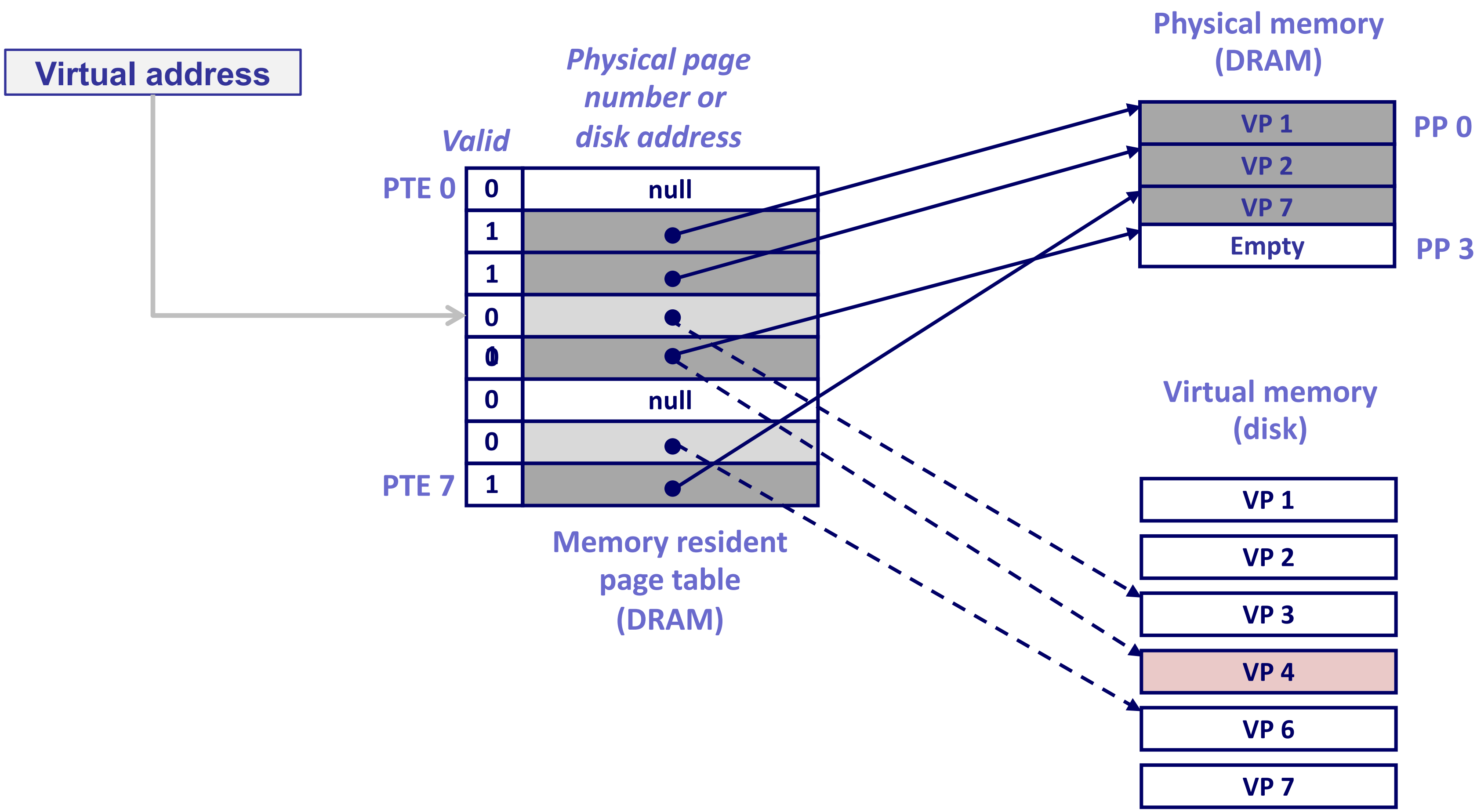
Handling Page Fault (3)

- Evict the content of VP4



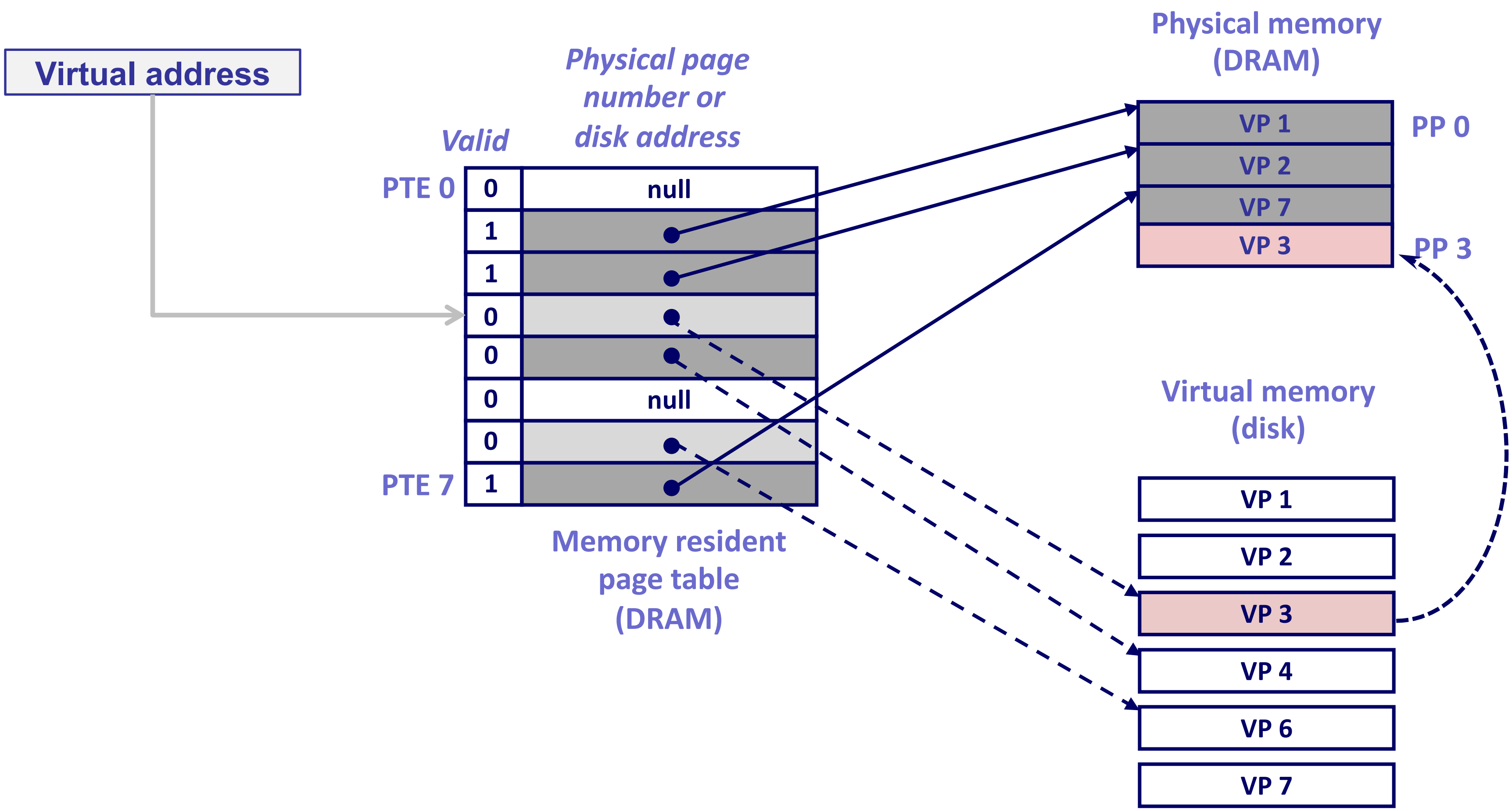
Handling Page Fault (4)

- Update page table



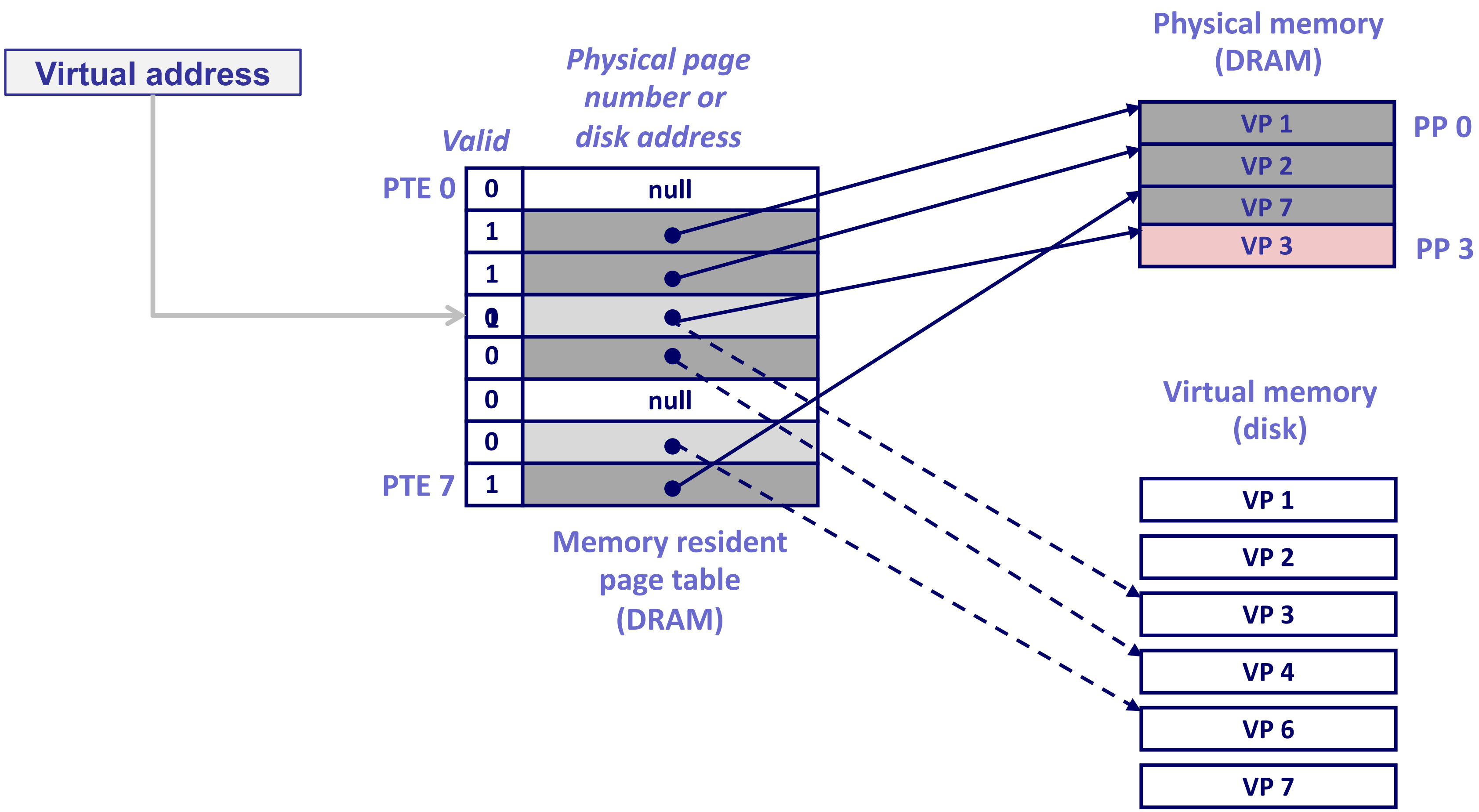
Handling Page Fault (5)

- Load content of VP3 to DRAM



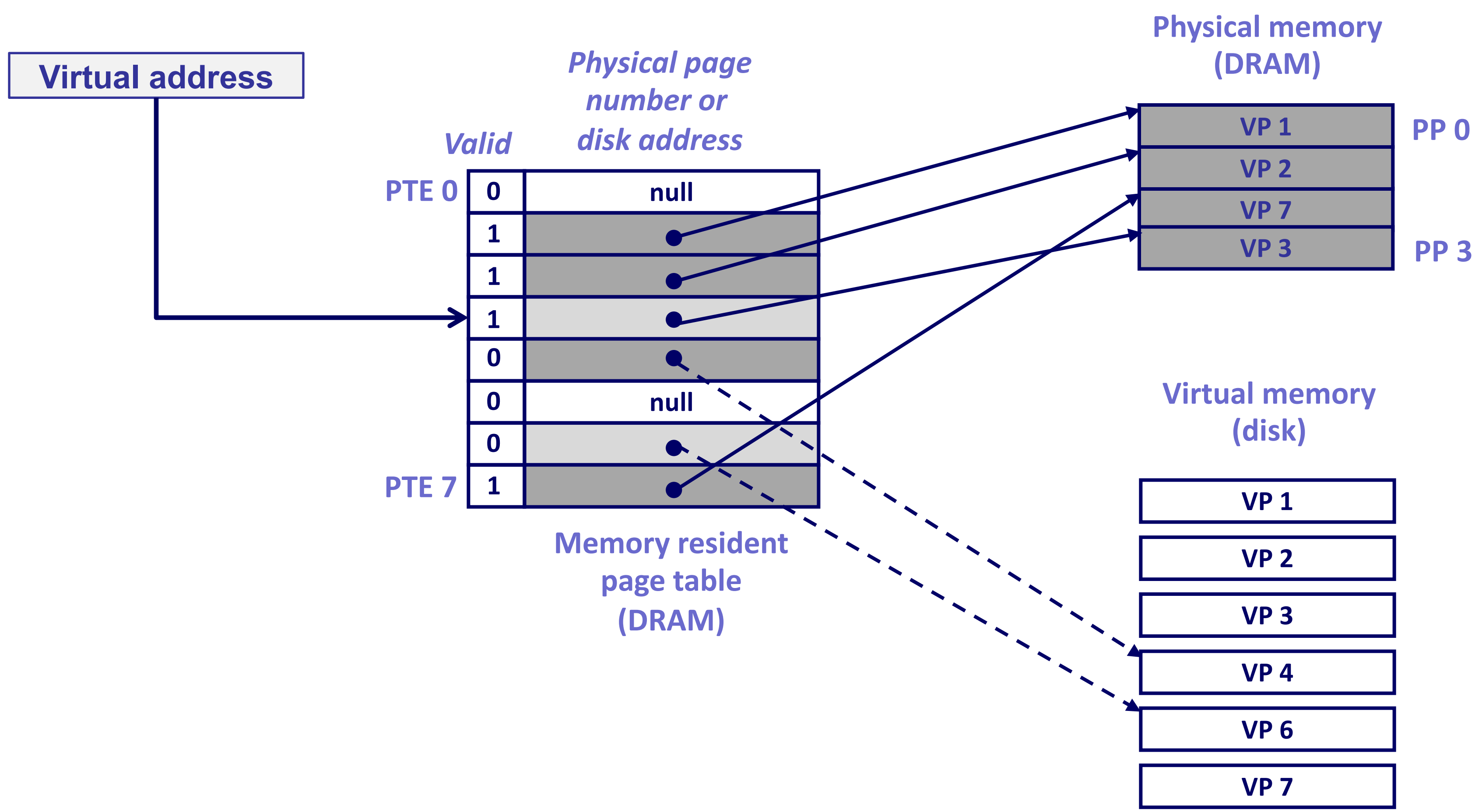
Handling Page Fault (6)

- Update page table



Handling Page Fault (7)

- Restart the instruction: page hit!



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
  - This is likely (much) less than all of available memory
- 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
  - Written back only if it has been modified (i.e., “dirty”)!

Page Replacement Policy

- Page replacement policy: determine which page to remove when we need a victim
- Does it matter?
  - Yes! Page faults are super expensive
  - Getting the number down, can improve the performance of the system significantly
- Silver lining
  - Virtual memory is "fully associative", we can pick any item
  - Because the fault time is so long, we can afford more complex algorithm

Locality to the Rescue

- Recall that cache works because of locality
  - Temporal and spatial
- All caching schemes depend on locality
  - What happens if a program does not have locality?
  - High cost of cache miss is acceptable, if infrequent
  - Processes usually reference data in localized patterns, making caching practical

Evicting the Best Data

- Goal is to reduce the cache/page miss rate
- The best data to evict is the one never touched again
  - Will never have a cache miss on it
- Never is a long time, so picking the data closest to "never" is the next best thing
  - Evicting the data that won't be used for the longest period of time minimizes the number of cache misses
  - Proved by Belady
- We'll survey various replacement algorithms, starting from Belady's

Belady’s Algorithm

- Belady’s algorithm
  - Idea: Replace the page that will not be used for the longest time in the future
  - Optimal? How would you show?
  - Problem: Have to predict the future
- Why is Belady’s useful then?
  - Use it as a yardstick/upper bound
  - Compare implementations of page replacement algorithms with the optimal to gauge room for improvement
    - » If optimal is not much better, then algorithm is pretty good
  - What’s a good lower bound?
    - » Random replacement is often the lower bound

First-In First-Out (FIFO)

- FIFO is an obvious algorithm and simple to implement
  - Maintain a list of pages in order in which 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?
  - Then again, maybe it’s not
  - We don’t have any info to say one way or the other
- FIFO suffers from “Belady’s Anomaly”
  - The miss rate might actually increase when the cache size grows (very bad)

Least Recently Used (LRU)

- LRU uses reference information to make a more informed replacement decision
  - Idea: We can't predict the future, but we can make a guess based upon past experience
  - On replacement, evict the page that has not been used for the longest time in the past (Belady's: future)
  - When does LRU do well? When does LRU do poorly?
- Implementation
  - To be perfect, need to time stamp every reference (or maintain a stack) – much too costly
  - So we need to approximate it

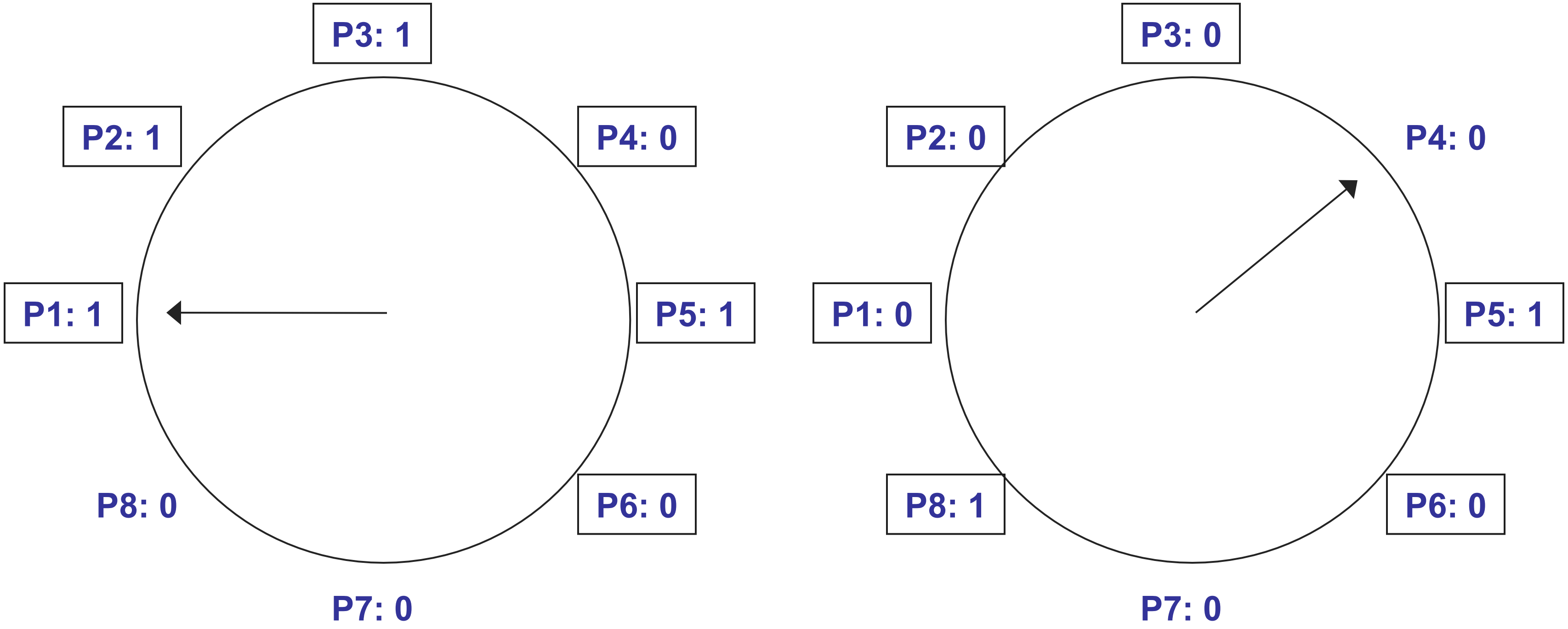
Approximating LRU

- LRU approximations by using a reference bit
  - Keep a counter for each cache block
  - At regular intervals, for every cache block do:
    - If ref bit = 0, increment counter
    - If ref bit = 1, zero the counter
    - Zero the reference bit
  - The counter will contain the number of intervals since the last reference to the page
  - The block with the largest counter is the least recently used
  - Finding the largest counter is still expensive!

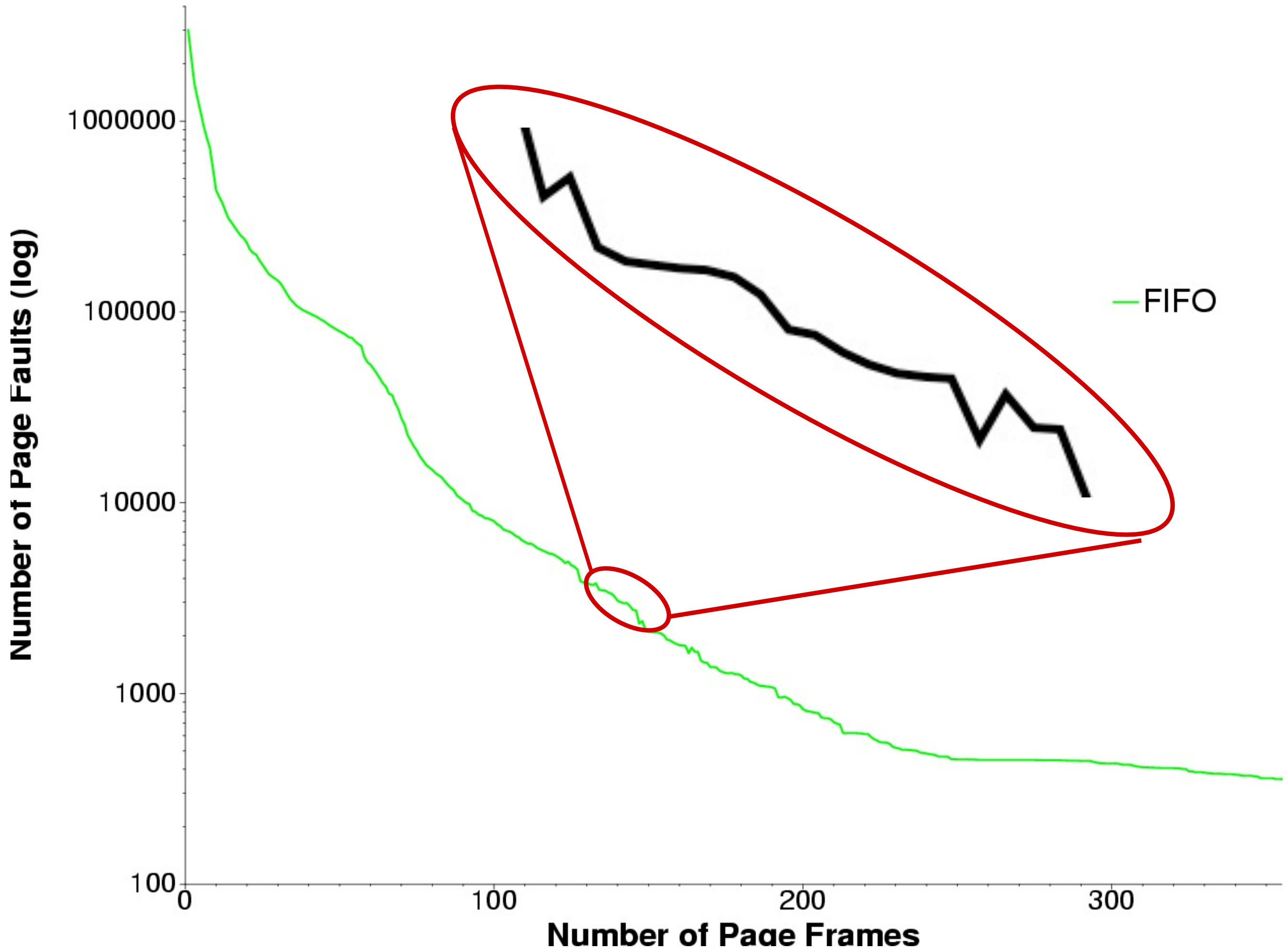
LRU Clock (Not Recently Used)

- Not Recently Used (NRU) – Used by Unix
  - Replace page that is "old enough"
  - Arrange all blocks in a big circle (clock)
  - A clock hand is used to select a good LRU candidate
    - » Sweep through the blocks in circular order like a clock
    - » If the ref bit is off, it hasn't been used recently
      - What is the minimum "age" if ref bit is off?
    - » If the ref bit is on, turn it off and go to next page
  - Arm moves quickly when blocks are needed
  - If number blocks is large, "accuracy" of information degrades
    - » What does it degrade to?

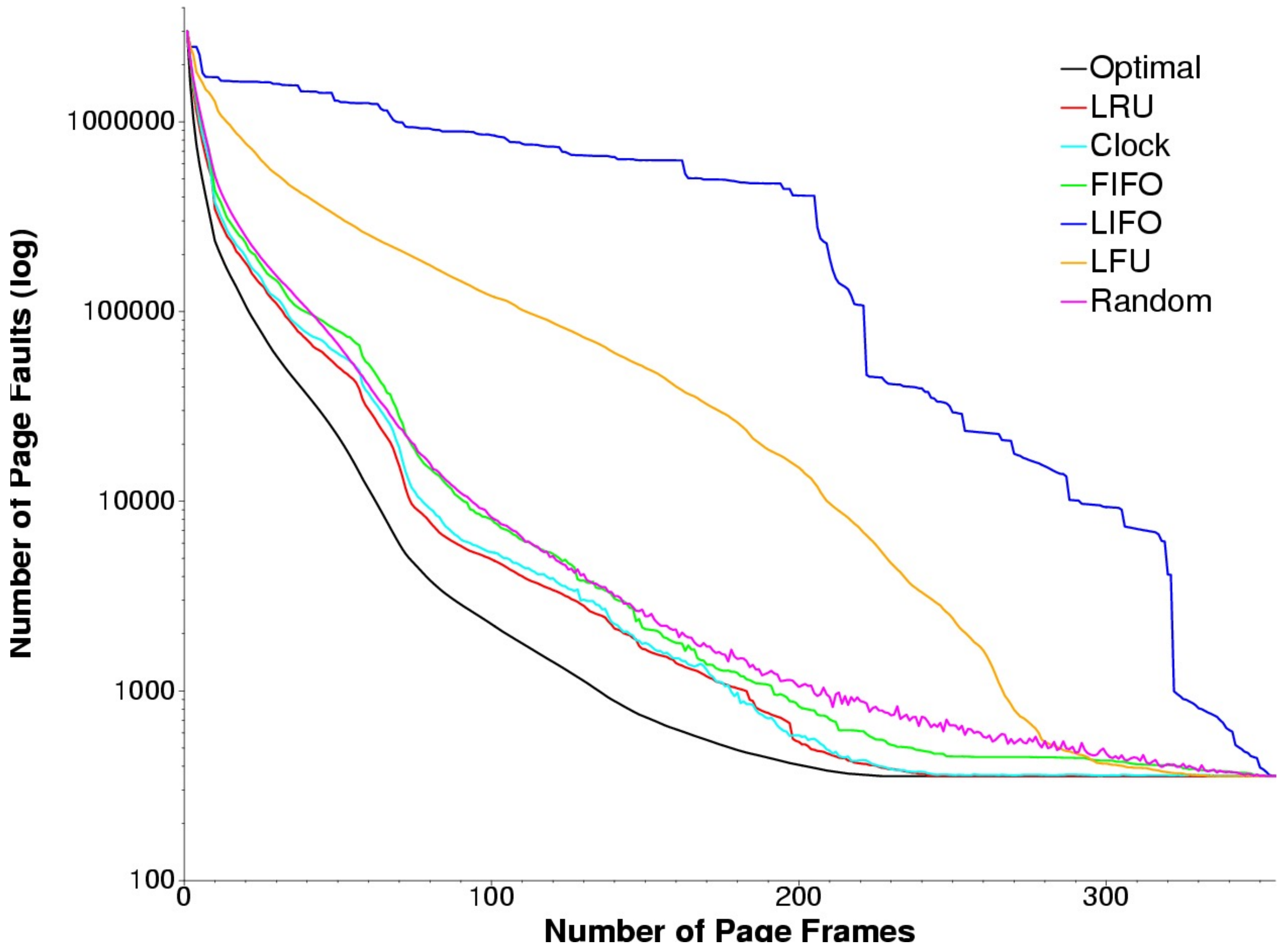
LRU Clock



Example: Belady’s Anomaly



Example: gcc Page Replace



Fixed vs. Variable Space

- In a multiprogramming system, we need a way to allocate memory to competing processes
- Problem: 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 the limit, it replaces from its own pages
    - Local replacement
      - Some processes may do well while others suffer
  - Variable space algorithms
    - Process' set of pages grows and shrinks dynamically
    - Global replacement
      - One process can ruin it for the rest

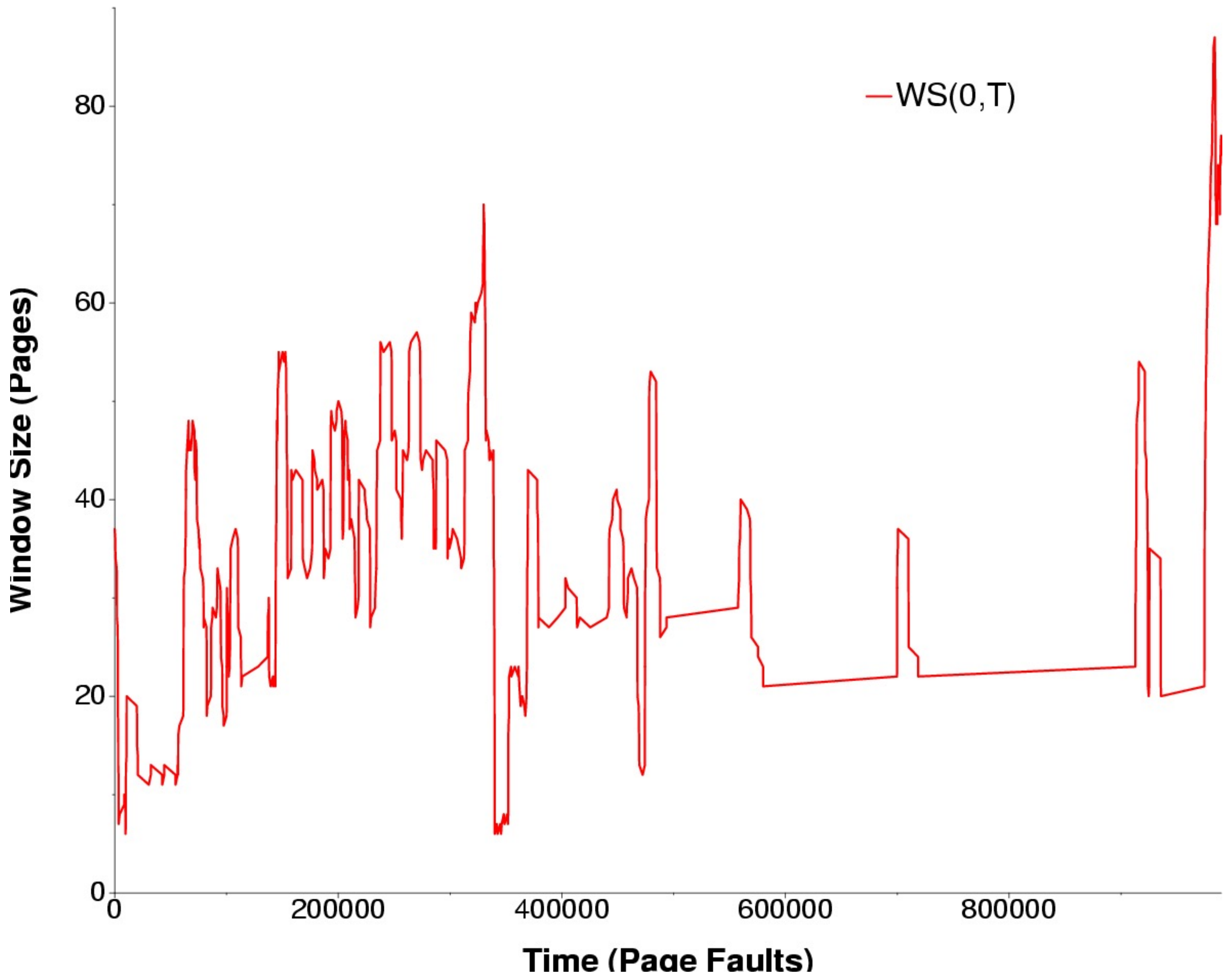
Working Set Model

- A working set of a process is used to model the dynamic locality of its memory usage
  - Defined by Peter Denning in 60s
- Definition
  - WS(t,w) = {set of pages P, such that every page in P was referenced in the time interval (t, t-w)}
  - t – time, w – working set window (measured in page refs)
- A page is in the working set (WS) only if it was referenced in the last w references

Working Set Size

- The working set size is 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, you reference more pages
  - Within that period of time, the working set size is larger
- Intuitively, want the working set to be the set of pages a process needs in memory to prevent heavy faulting
  - Each process has a parameter w that determines a working set with few faults
  - Denning: Don't run a process unless working set is in memory

Example: gcc Working Set



Working Set Problems

- Problems
  - How do we determine w?
  - How do we know when the working set changes?
- Too hard to answer
  - So, working set is not used in practice as a page replacement algorithm
- However, it is still used as an abstraction
  - The intuition is still valid
  - When people ask, "How much memory does Firefox need?", they are in effect asking for the size of Firefox's working set

Page Fault Frequency (PFF)

- Page Fault Frequency (PFF) is 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 (Belady's Anomaly)
  - If the fault rate is below a low threshold, take away memory
    - Should fault more
- Hard to use PFF to distinguish between changes in locality and changes in size of working set

Thrashing

- Page replacement algorithms avoid thrashing
  - When most of the time is spent by the OS in paging data back and forth from disk
  - No time spent doing useful work (making progress)
  - In this situation, the system is overcommitted
    - No idea which pages should be in memory to reduce faults
    - Could just be that there isn't enough physical memory for all of the processes in the system
    - Ex: Running Windows95 with 4 MB of memory…
  - Possible solutions
    - Swapping – write out all pages of a process
    - Buy more memory