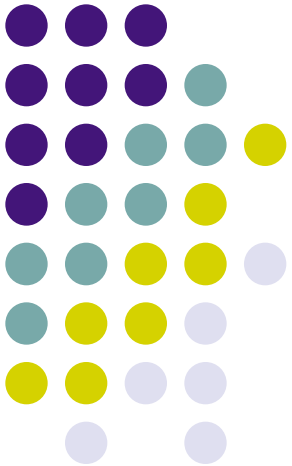
Working Set Model, Connecting the Dots, VM Review

ECE469, March 23

Yiying Zhang

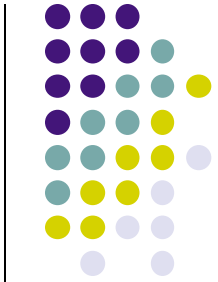


Reading

- Chapter 9

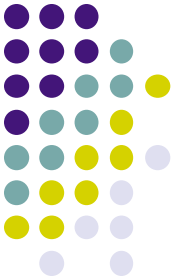
- Quiz 2 next Thur

Today



- Working Set Model
- Prefetching
- Memory Sharing
- VM Review
- NVM

Page replacement algorithms: Summary



- Optimal

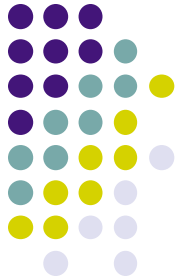
- FIFO
- Random

- Approximate LRU (NRU)

- FIFO with 2nd chance
- Clock: a simple FIFO with 2nd chance

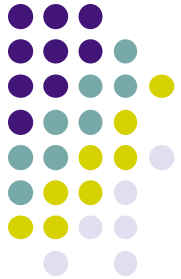
- Enhanced FIFO with 2nd chance

[lec18] Thrashing

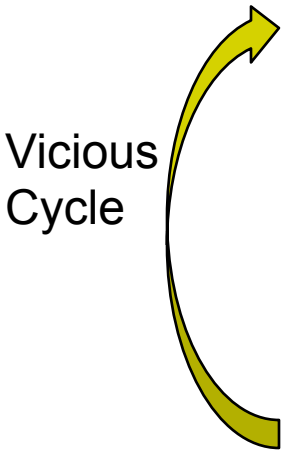


- **Thrashing** ≡ a process is busy swapping pages in and out

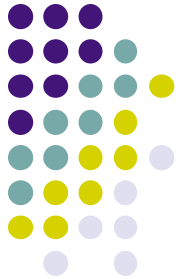
[lec18] Thrashing can lead to vicious cycle



- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - low CPU utilization
  - OS thinks that it needs to increase the degree of multiprogramming (actual behavior of early paging systems)
  - another process added to the system
  - page fault rate goes even higher



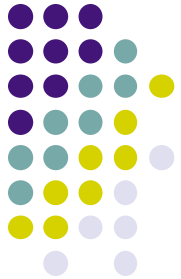
[lec18] Demand paging and thrashing



- Why does demand paging work?
  - Data reference exhibits locality

- Why does thrashing occur?
  - Σ size of locality > total memory size

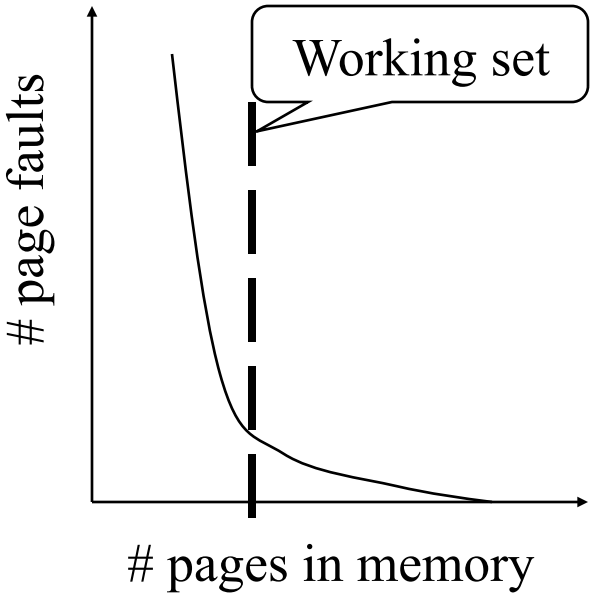
Intuitively, what to do about thrashing?



- If a single process's locality too large for memory, what can OS do?
  - e.g., pin most data (hotter data) in memory, sacrifice the rest
- If the problem arises from the sum of several processes?
  - Figure out how much memory each process needs – "locality"
  - What can we do?
    - Can limit effects of thrashing using local replacement
    - Or, bring a process' working set before running it
    - Or, wait till there is enough memory for a process's need

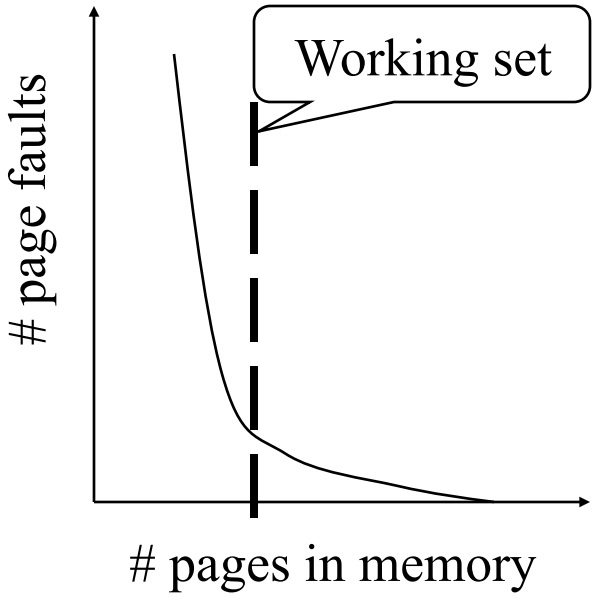
Key observation

- Locality in memory references
  - Spatial and temporal
- Want to keep a set of pages in memory that would avoid a lot of page faults
  - “Hot” pages
- Can we formalize it?



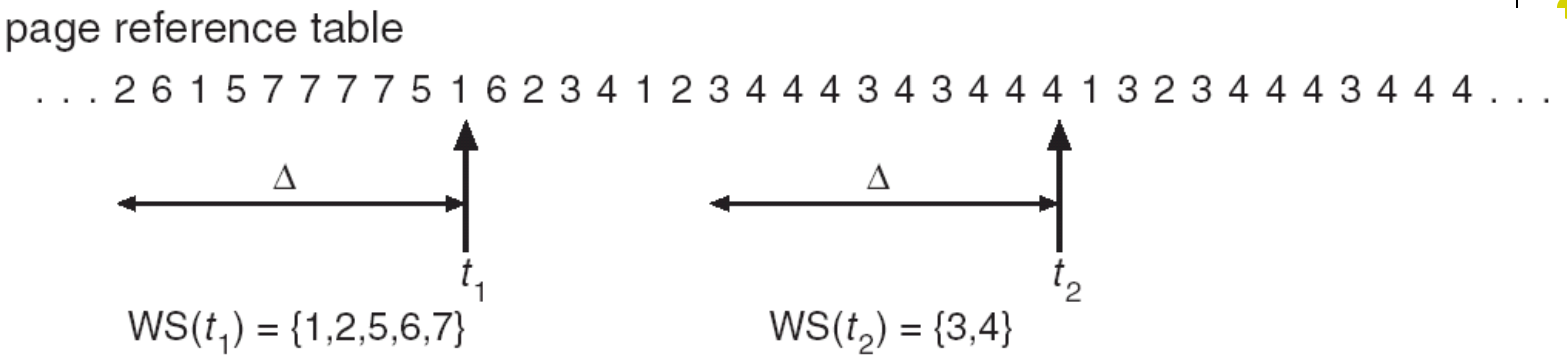
Working Set Model – by Peter Denning (Purdue CS head, 79-83)

- An informal definition:
  - Working set: The collection of pages that a process is working within a time interval, and which must thus be resident if the process is to avoid thrashing



- But how to turn the concept/theory into practical solutions?
  1. Capture the working set
  2. Influence the scheduler or replacement algorithm

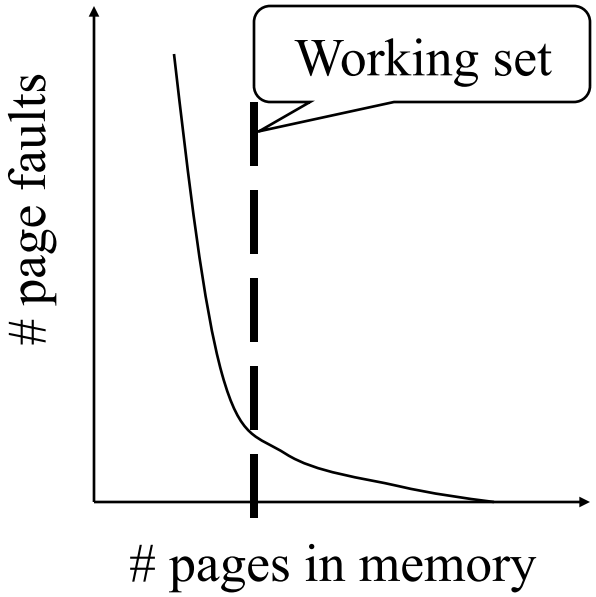
Working Sets



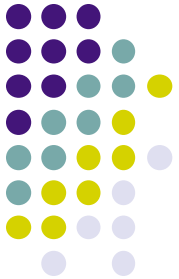
- The working set size is *num of* pages in the working set
  - the number of pages touched in the interval [t-Δ+1..t].
- The working set size changes with program locality.
  - during periods of poor locality, you reference more pages.
  - Within that period of time, you will have a larger working set size.
- Goal: keep WS for each process in memory.

Working Set Model

- Usage idea: use recent needs of a process to predict its future needs
  - Choose Δ, the WS parameter
  - At any given time, all pages referenced by a process in its last Δ seconds comprise its working set
  - Don't execute a process unless there is enough memory to fit its working set
- Needs a companion replacement algorithm



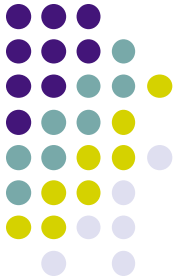
Working Set replacement algorithm



- Main idea
  - *Take advantage of reference bits*
  - *Variation of FIFO with 2nd chance*

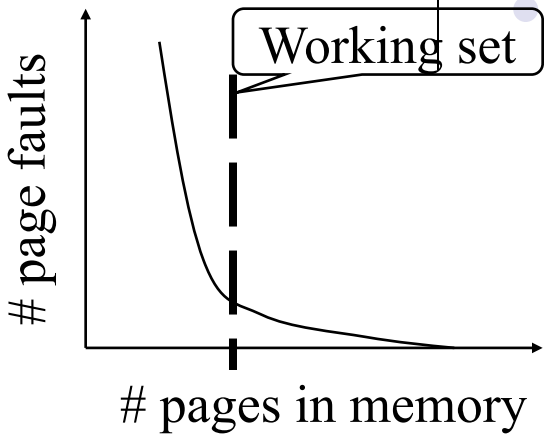
- An algorithm (assume reference bit)
  - On a page fault, scan through all pages of the process
  - If the reference bit is 1, clear the bit, record the current time for the page
  - If the reference bit is 0, check the "last use time"
    - If the page has not been used within Δ, replace the page
    - Otherwise, go to the next page

Working Set Clock Algorithm
(assume reference bit + modified bit)



- Upon page fault, follow the clock hand
- If the reference bit is 1, set reference bit to 0, set the current time for the page and go to the next
- If the reference bit is 0, check "last use time"
  - If page used within Δ, go to the next
  - If page not used within Δ and modify bit is 1
    - Schedule the page for page out (then reset modify bit) and go to the next
  - If page not used within Δ and modified bit is 0
    - Replace this page

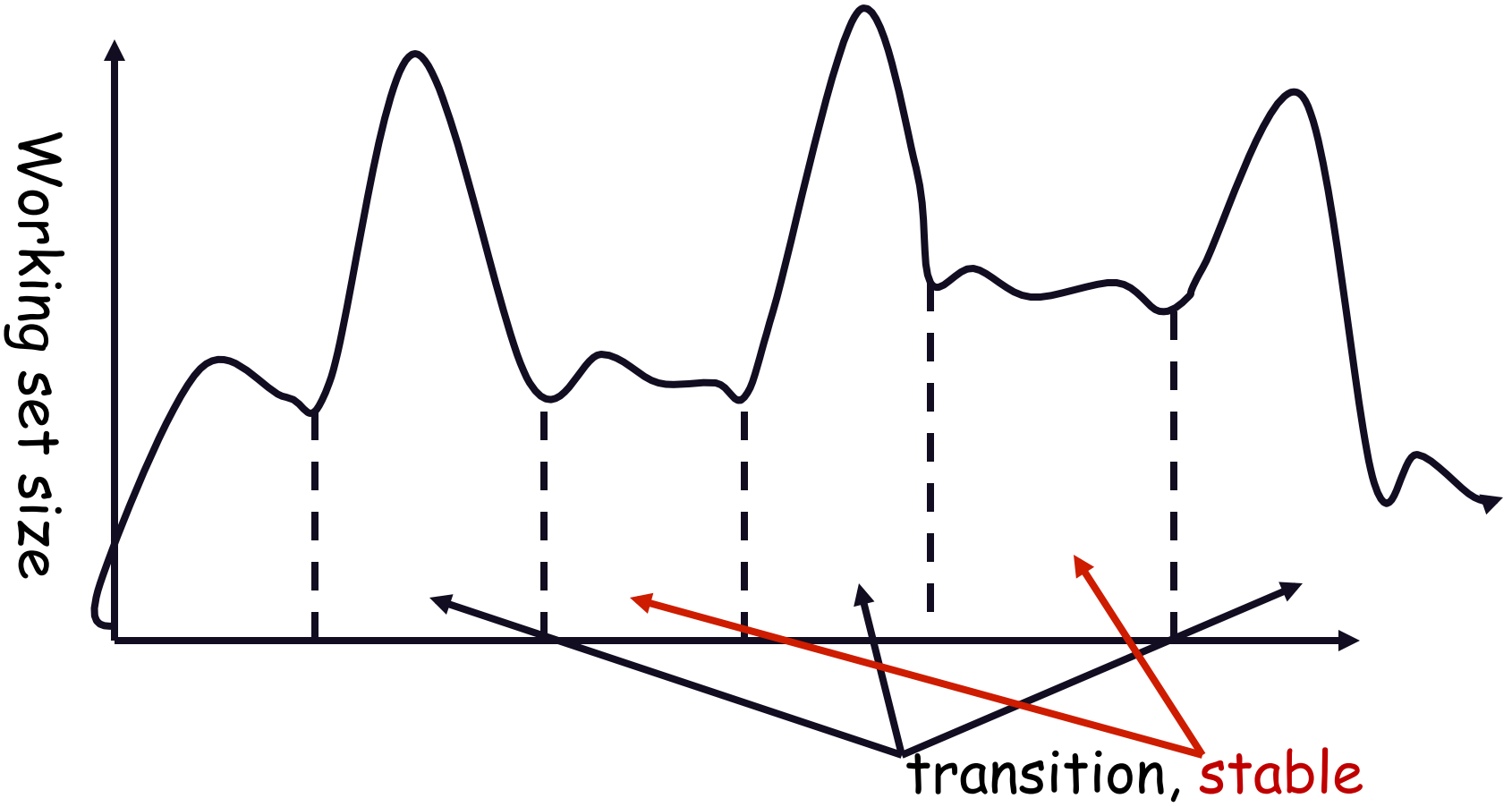
Challenges with WS algorithm implementation



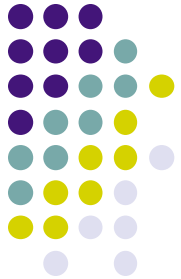
- What should Δ be?
  - What if it is too large?
  - What if it is too small?

- How many jobs need to be scheduled in order to keep CPU busy?
  - Too few → cannot keep CPU busy if all doing I/O
  - Too many → their WS may exceed memory

Working Sets in the Real World

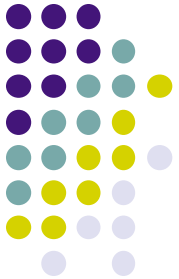


More Challenges with Capturing Working Set



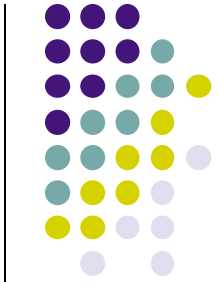
- Working set isn't static
- There often isn't a single "working set"
  - e.g., Multiple plateaus in previous curve (L1 $, L2 $, etc)
  - Program coding style affects working set
  - e.g., matrix multiply
- Working set is often hard to measure
  - What's the working set of an interactive program?
  - How to calculate WS if pages are shared?

Today



- Working Set Model
- Prefetching
- Memory Sharing
- VM Review
- NVM

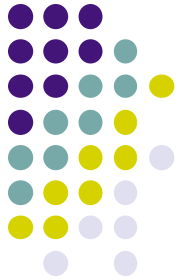
Locality



- So far, have discussed algorithms that utilize temporal locality

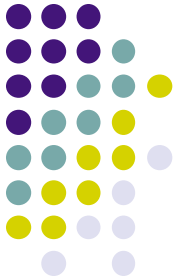
- What about spatial locality?

[lec17] Virtual Memory Implementation



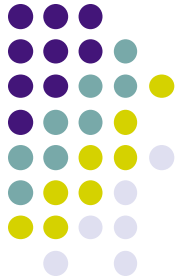
- Virtual memory is typically implemented via *demand paging*
- *demand paging:*
  - Load memory pages (from storage) "**on demand**"
  - paging with swapping, e.g., physical pages are swapped in and out of memory

[lec17] Page selection (when)



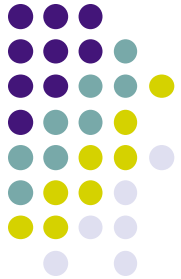
- Prepaging
  - Bring a page into memory before it is referenced
  - Hard to do without a “prophet"
- Request paging
  - Let user say which pages are needed when
  - Users don’t always know best
  - And aren’t always impartial
- Demand paging
  - Start up process with no pages loaded
  - Load a page when a page fault occurs, i.e., wait till it MUST be in memory
  - Almost all paging systems are demand paging

Prefetching (pre-paging)



- Pure demand paging relies only on faults to bring in pages
- What kind of locality are page replacement algorithms exploring?
- Problems?
  - Possibly lots of faults at startup
  - Ignores spatial locality
- Exploring spatial locality in replacement is hard
  - Why?
- What about prefetching (pre-paging)?
  - Loading groups of pages upon initial fault (what pages?)
  - Prefetching/preloading

break



Leslie Lamport

- Distributed Systems Theory
- Paxos, Lamport clock

Butler Lampson

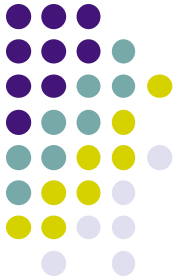
- Personal computer

- All problems in computer science can be solved by adding another level of indirection - usually attributed to Lampson who attributes it to David Wheeler

Vint Cerf and Bob Kahn

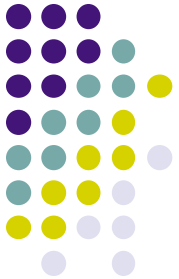
- Network
- Fathers of Internet
- TCP/IP

Tim Berners-Lee



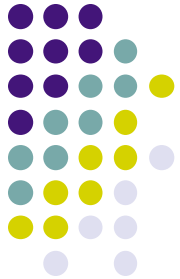
- Inventor of World Wide Web

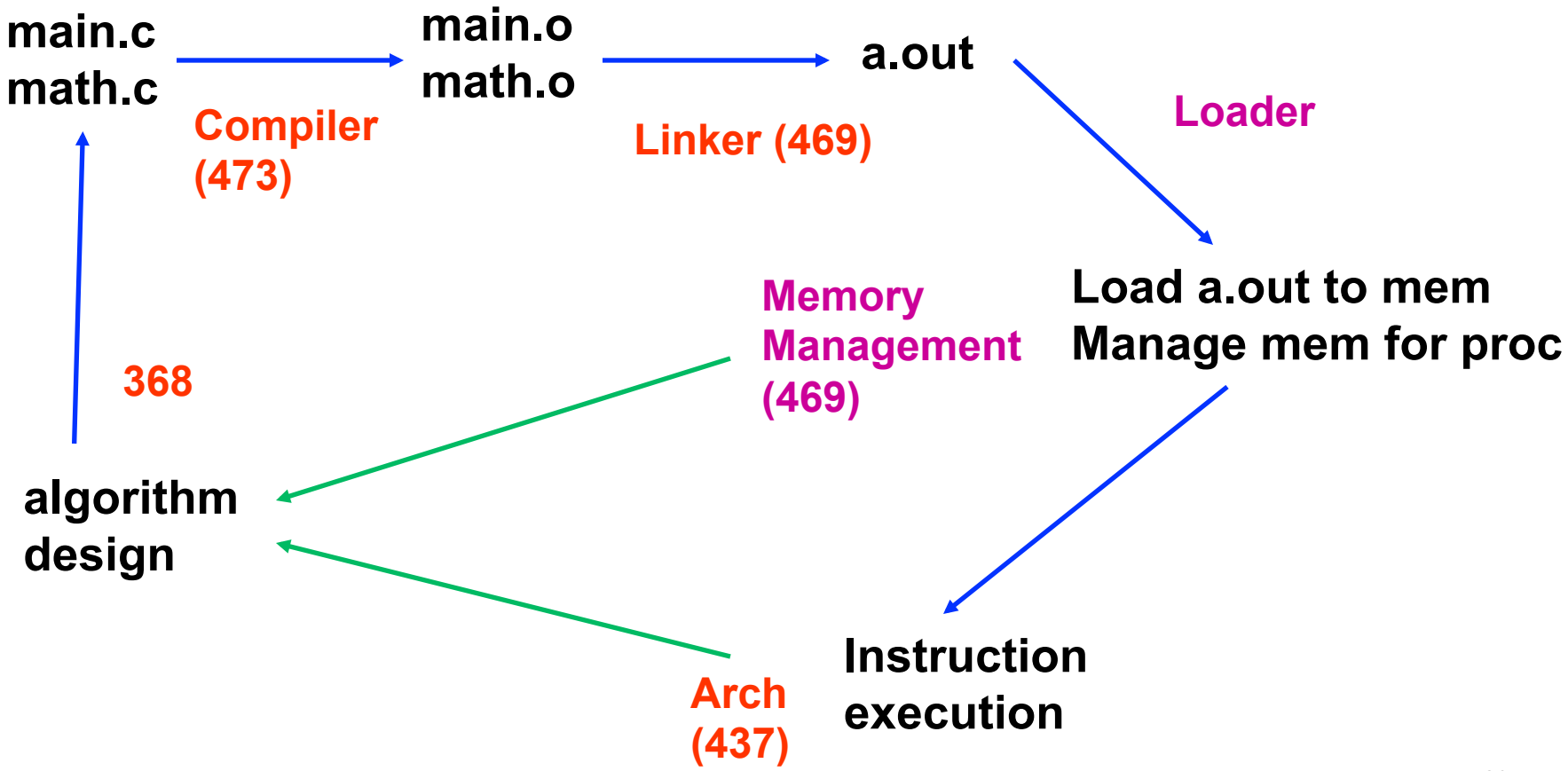
Today



- Working Set Model
- Prefetching
- Memory Sharing
- VM Review
- NVM

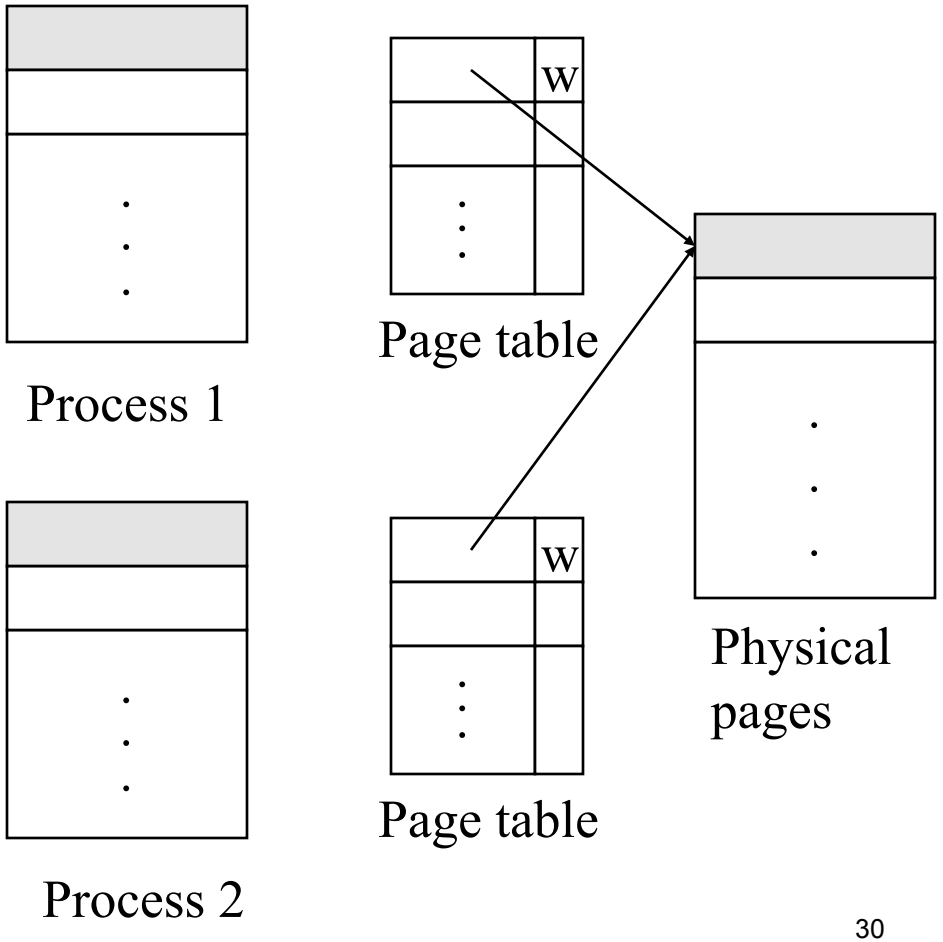
Connecting the dots … closing the loop





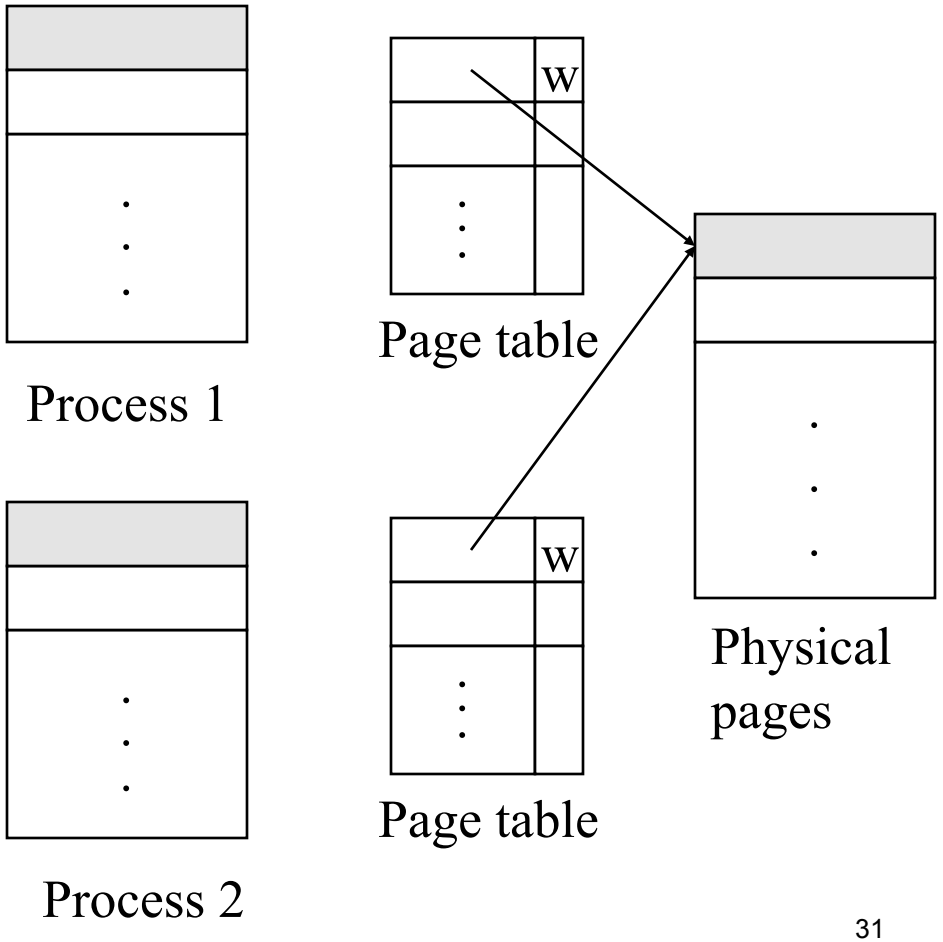
Shared Memory

- How do two processes share memory under paging?
  - PTEs pointing to same phys addr

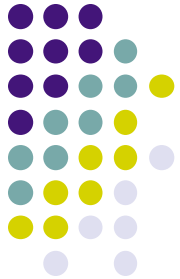


With Shared Memory

- How to destroy a virtual address space?
  - Reference count
- How to swap out/in?
  - Link all PTEs
  - Operation on all entries
- How to pin/unpin?
  - Link all PTEs
  - Reference count



[lec4] C program Forking a new Process



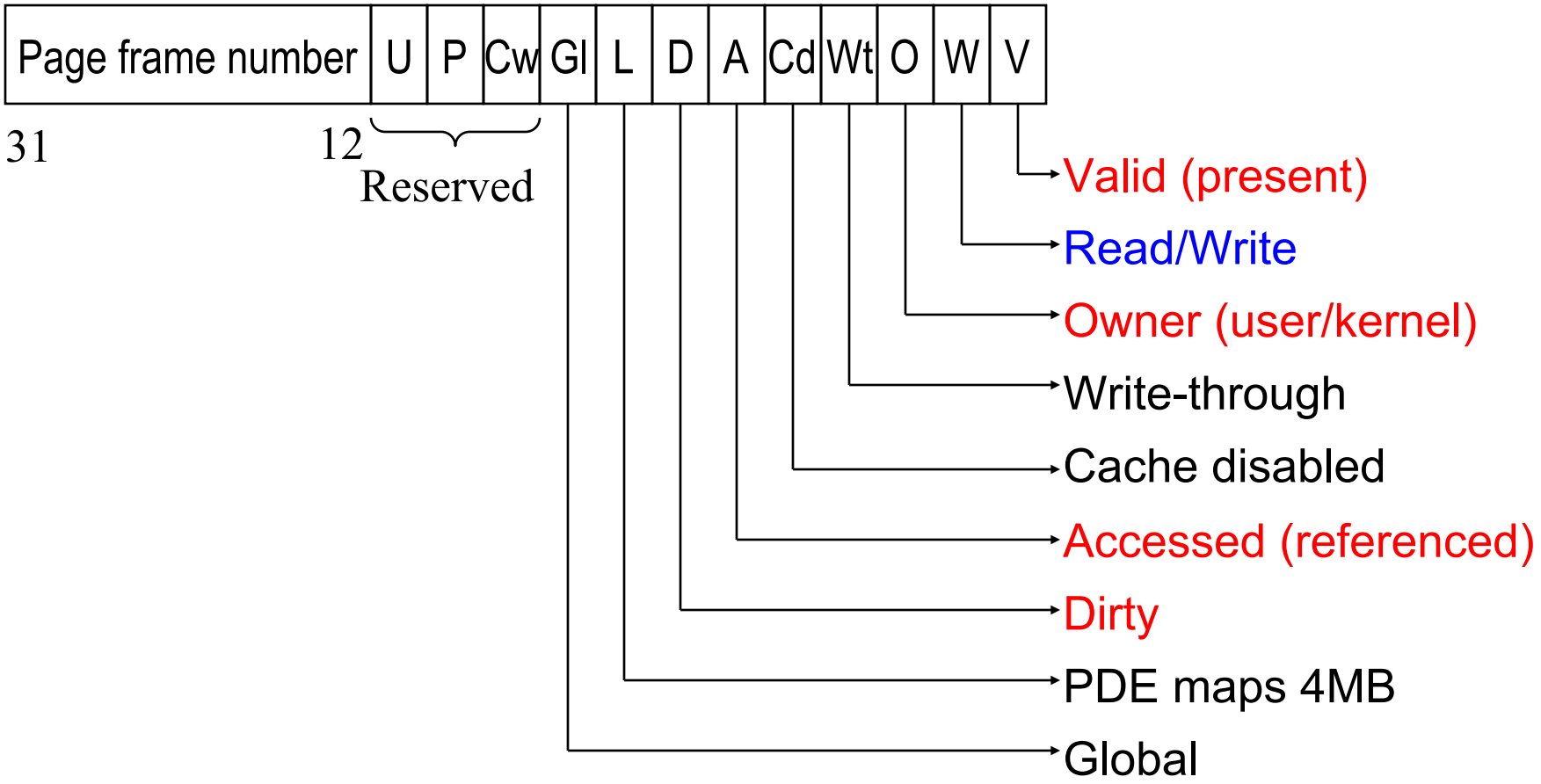
```
#include <stdio.h>

void main()
{
  int pid;        int was = 3;
  pid = fork(); /* fork another process */

  if (pid == 0) { /* child process */
     sleep(2); printf("was = %d", was);
     execlp("/bin/ls", "ls", NULL);}
  else { /* pid > 0; parent process */
     was = 4;
     printf("child process id %d was=%d ", pid, was);
     wait(NULL);     exit(0);
  }
}
```

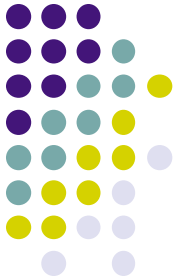
- How to efficiently implement fork()?

x86 Page Table Entry

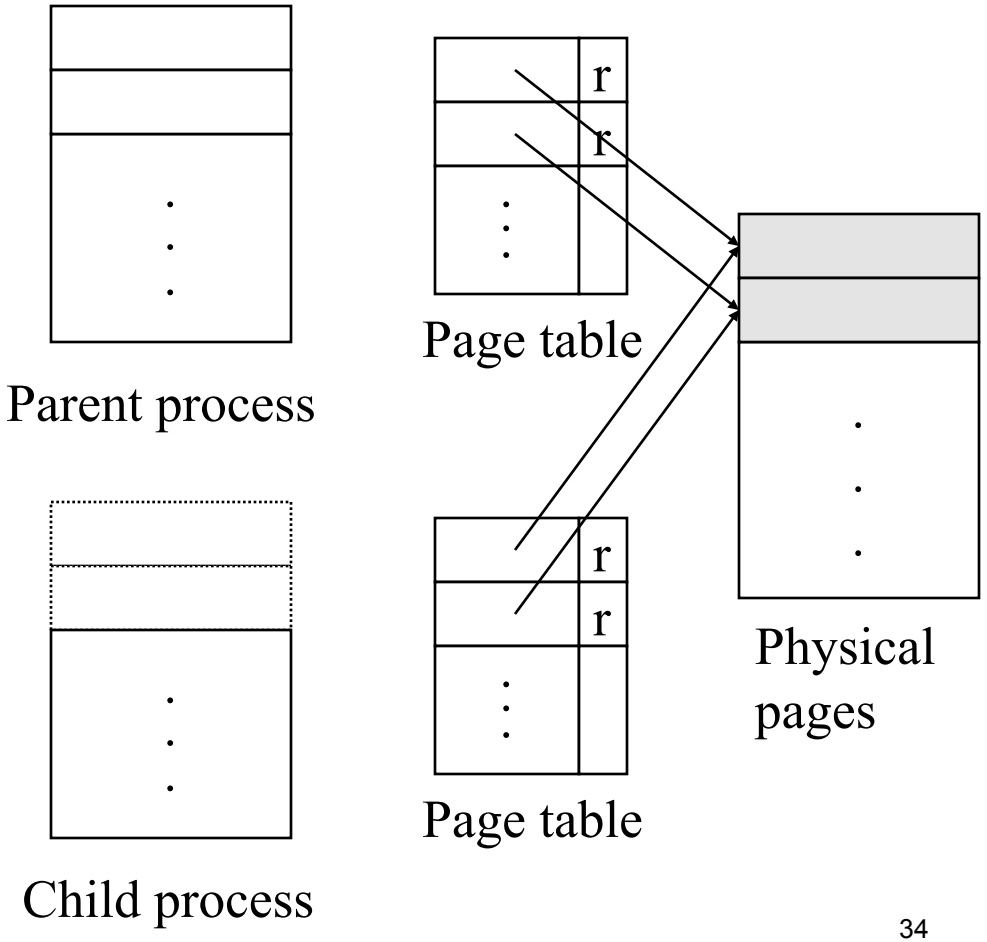


Copy-On-Write

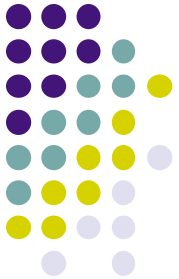
Used in Win2k,
Linux Solaris2
in duplicating processes



- Child's virtual address uses the same page mapping as parent's
  - Make all pages read-only (both parent and child)
- On a read, nothing happens
- On a write (either parent or child), generates an access fault
  - map to a new page frame
  - copy the page over
  - restart the instruction
  - the other process marks the page not shared (writeable)

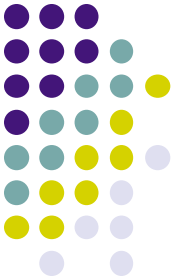


Today



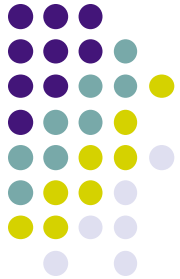
- Working Set Model
- Prefetching
- Memory Sharing
- VM Review
- NVM

Virtual Memory Review (1/3)



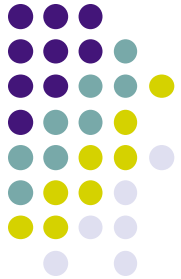
- Page fault handling (mechanism)
- Paging algorithms (policies)
  - Optimal
  - FIFO
  - FIFO with 2nd chance
  - Clock: a simple FIFO with 2nd chance
  - LRU
  - Approximate LRU
  - NFU

Virtual Memory Review (2/3)



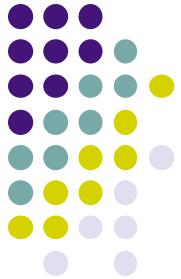
- Important questions
  - What is the use of optimal algo?
  - If future is unknown, what makes us think there is a chance for doing a good job?
  - Without additional hardware support, the best we can do?
  - What is the minimal hardware support under which we can do a decent job?
  - Why is it difficult to implement exact LRU? (exact anything)
  - For a fixed replacement algorithm, more page frames → less page faults?
  - How can we move page-out out of critical path?

Virtual Memory Review (3/3)

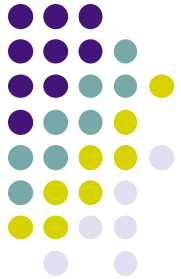


- Per-process vs. global page replacement
- Thrashing
- What causes thrashing?
- What to do about thrashing?
- What is working set?
- Working set replacement algorithms
- Memory sharing and copy-on−write

Backup Slides

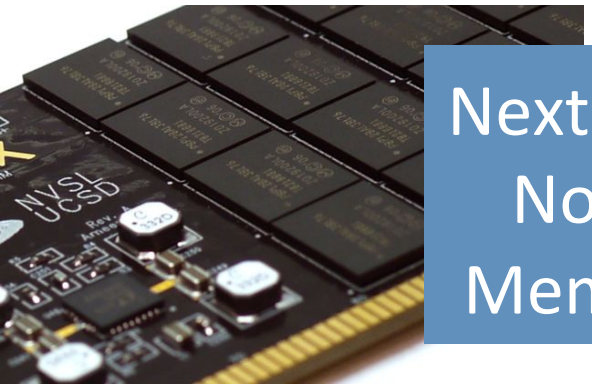


Next-Generation Non-Volatile Memory (NVM)



Memory

Fast
Volatile
In bytes



Next-Generation
Non-Volatile
Memory (NVM)

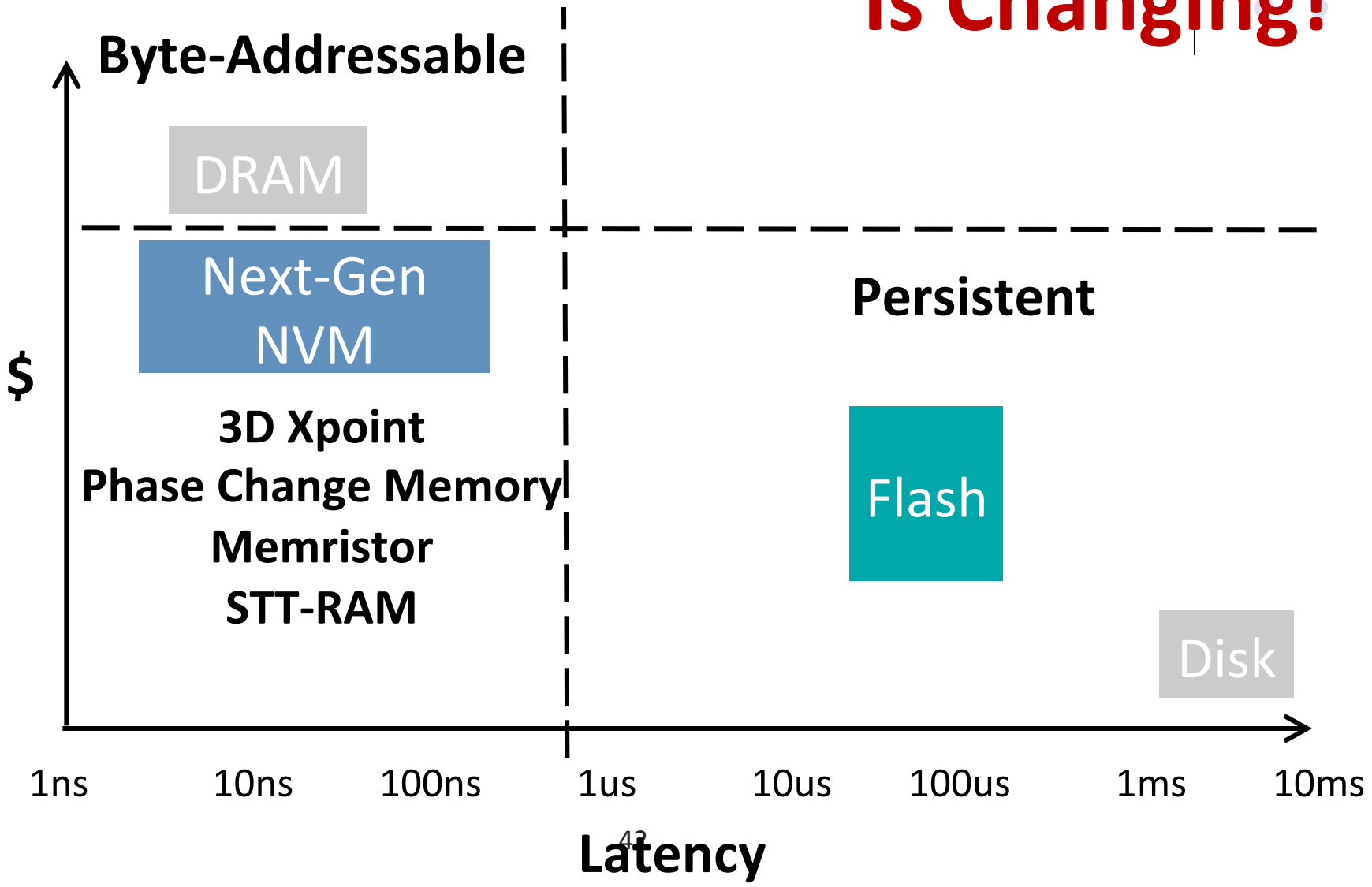
Storage

Slow
Persistent
In blocks

Flash-based
Solid State
Drive (SSDs)

2016

The Landscape of Memory and Storage Is Changing!



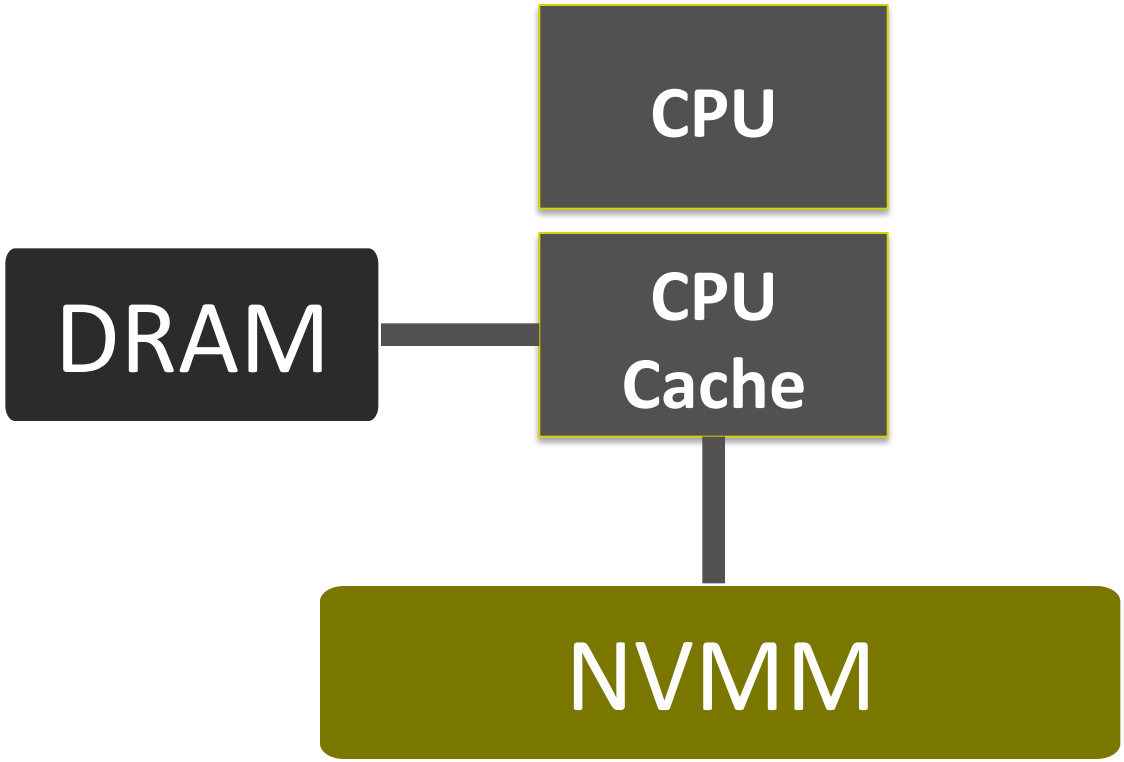
Next-Generation Non-Volatile Memory

Byte Addressable

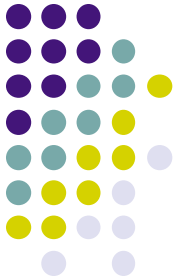
Low Latency

Persistence

Capacity



Design Issues of NVM



- Attach point and usage model
  - Main memory vs. storage
  - Used to store volatile data or persistent data?
  - Pointers?
- Access method
  - Memory load/store or I/O operations?
- Granularity
  - Byte vs. block
- Other layers
  - Do we still need DRAM? Hard disks/SSDs?
- Software overhead
- Remote access

Join WukLab!