UNIVERSITY OF OREGON

CIS 415: Operating Systems

VM Issues

Spring 2014

Prof. Kevin Butler

Administrative Notes

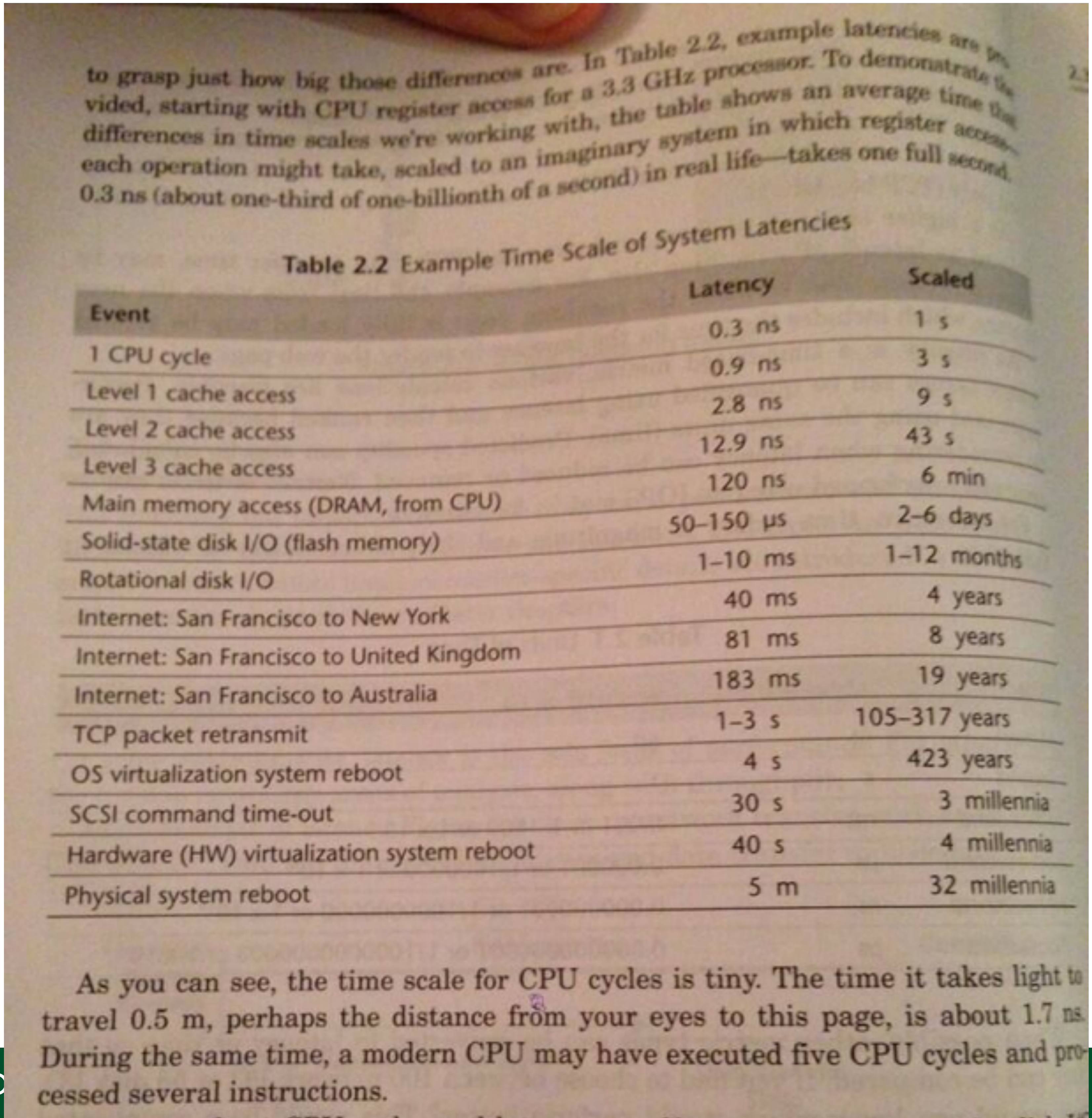
UNIVERSITY OF OREGON

- Project 2: due Friday night
- Project 3 and Assignment 3 out later today
- Final Exam on June 13 at 8 AM

- Good discussions on Piazza - make sure to read and use

System Performance

UNIVERSITY OF OREGON



to grasp just how big those differences are. In Table 2.2, example latencies are pro
vided, starting with CPU register access for a 3.3 GHz processor. To demonstrate the
differences in time scales we're working with, the table shows an average time that
each operation might take, scaled to an imaginary system in which register access
0.3 ns (about one-third of one-billionth of a second) in real life—takes one full second

Table 2.2 Example Time Scale of System Latencies

Event	Latency	Scaled
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access (DRAM, from CPU)	120 ns	6 min
Solid-state disk I/O (flash memory)	50–150 μs	2–6 days
Rotational disk I/O	1–10 ms	1–12 months
Internet: San Francisco to New York	40 ms	4 years
Internet: San Francisco to United Kingdom	81 ms	8 years
Internet: San Francisco to Australia	183 ms	19 years
TCP packet retransmit	1–3 s	105–317 years
OS virtualization system reboot	4 s	423 years
SCSI command time-out	30 s	3 millennia
Hardware (HW) virtualization system reboot	40 s	4 millennia
Physical system reboot	5 m	32 millennia

As you can see, the time scale for CPU cycles is tiny. The time it takes light to travel 0.5 m, perhaps the distance from your eyes to this page, is about 1.7 ns. During the same time, a modern CPU may have executed five CPU cycles and processed several instructions.

Source: "System Performance: Enterprise and Cloud", Brendan Gregg

Summary of page replacement algorithms

- OPT, FIFO, NRU, second-chance/clock, LRU, approximate LRU

- In practice, OSes use second chance/clock or some variations of it.

Belady's Anomaly

UNIVERSITY OF OREGON

- Normally you expect number of page faults to decrease as you increase physical memory size.

- However, this may not be the case in certain replacement algorithms

Belady's Anomaly

- FIFO replacement Algorithm
- Reference string:
  - 1 2 3 4 1 2 5 1 2 3 4 5
- 3 physical frames
  - F F F F F F F - - F F -
  - # of faults = 9
- 4 physical frames
  - F F F F - - F F F F F F
  - # of faults = 10

- Algorithms which do NOT suffer from Belady's anomaly are called *stack algorithms*
- E.g. OPT, LRU.

Paging Issues

UNIVERSITY OF OREGON

- Keep the essentials of what you currently need *(working set)* in physical memory.
- When something you need is not in memory, bring it in from disk:
  - On demand (demand-paging)
  - Ahead of need (pre-paging)
- Programs need to exhibit good locality to avoid "thrashing" of pages in memory.
- This usually requires good programming skills!

Efficient Physical Memory

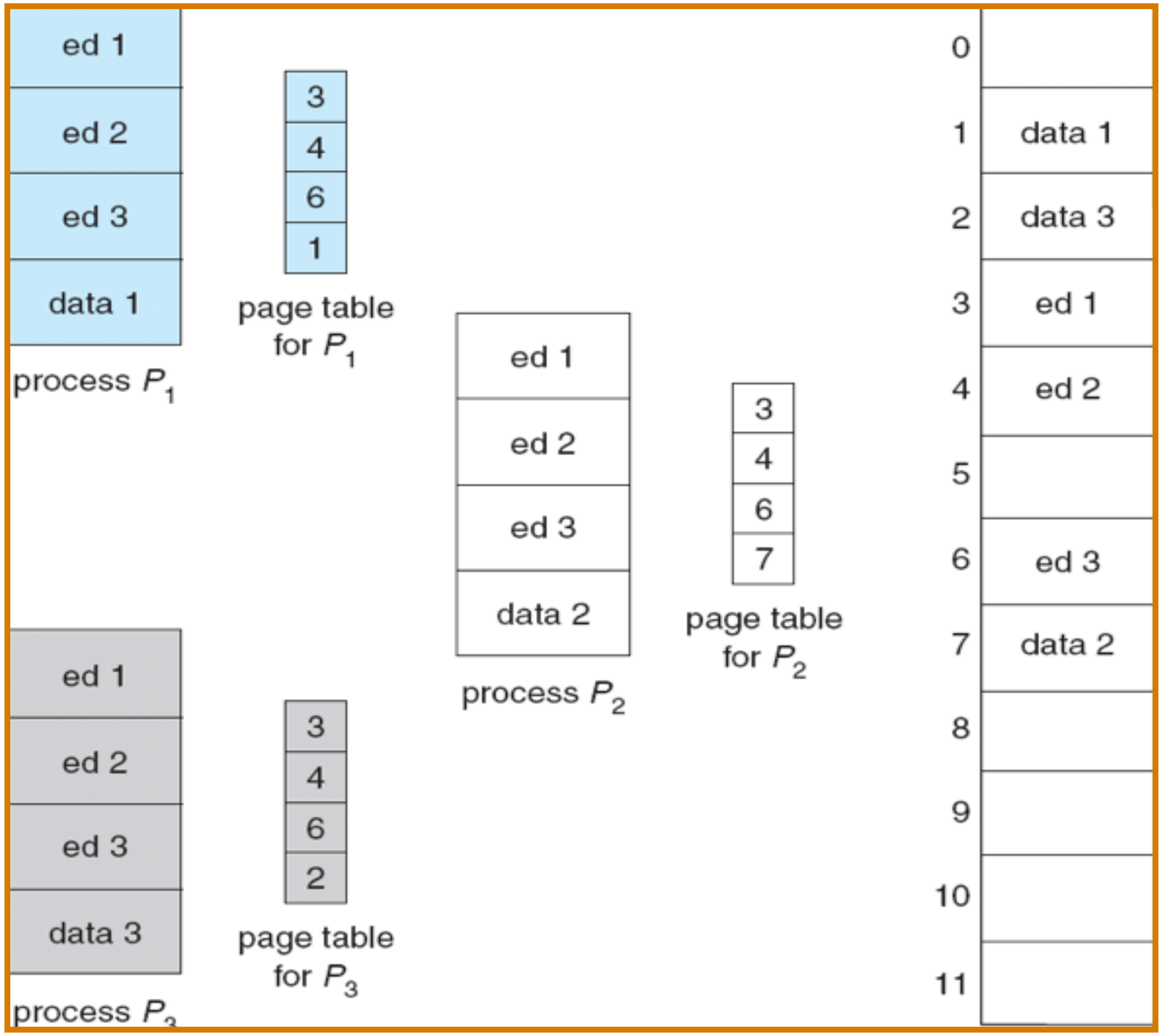
- Through virtual memory…
  - *N* 2^{32}-sized address spaces
  - All isolated by default
- Uses for memory
  - Make a new process
    - Address space
  - Make an IPC
    - Or a cross-address space call
- Challenges in memory use

Shared Pages

UNIVERSITY OF OREGON

- Shared code
  - One copy of read-only (*reentrant*) code shared among processes (i.e., text editors, compilers, window systems).
- Private code and data
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Example



Create New Address Space

- Via `fork` or `clone`
  - Copy of the old address space
- Change completely
  - `Exec`
- Or use the copy independently

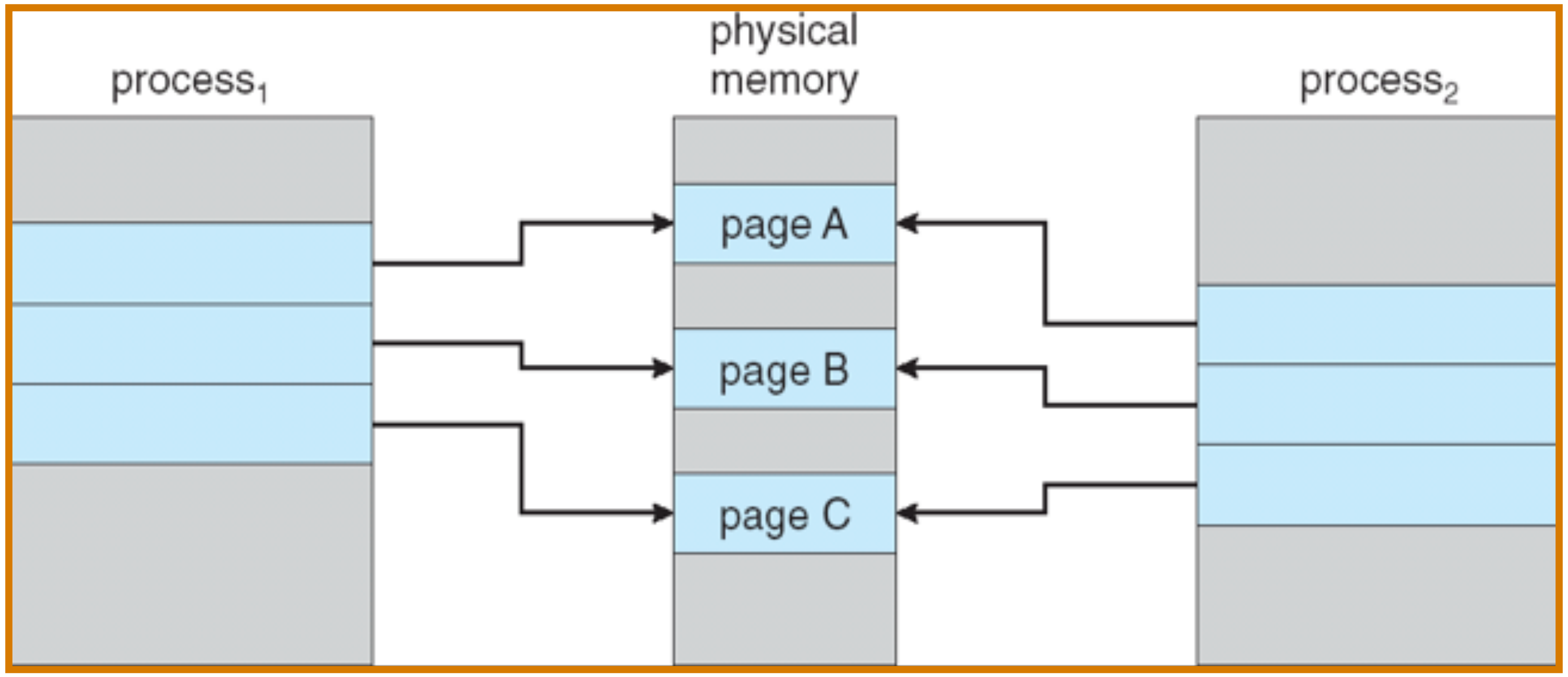
Copy-on-Write

- *Copy-on-Write* (COW) allows both parent and child processes to initially *share* the same pages in memory
  - If either process modifies a shared page, only then is the page copied
- COW allows more efficient process creation as only modified pages are copied
- Free pages are allocated from a **pool** of zeroed-out pages

COW in Action

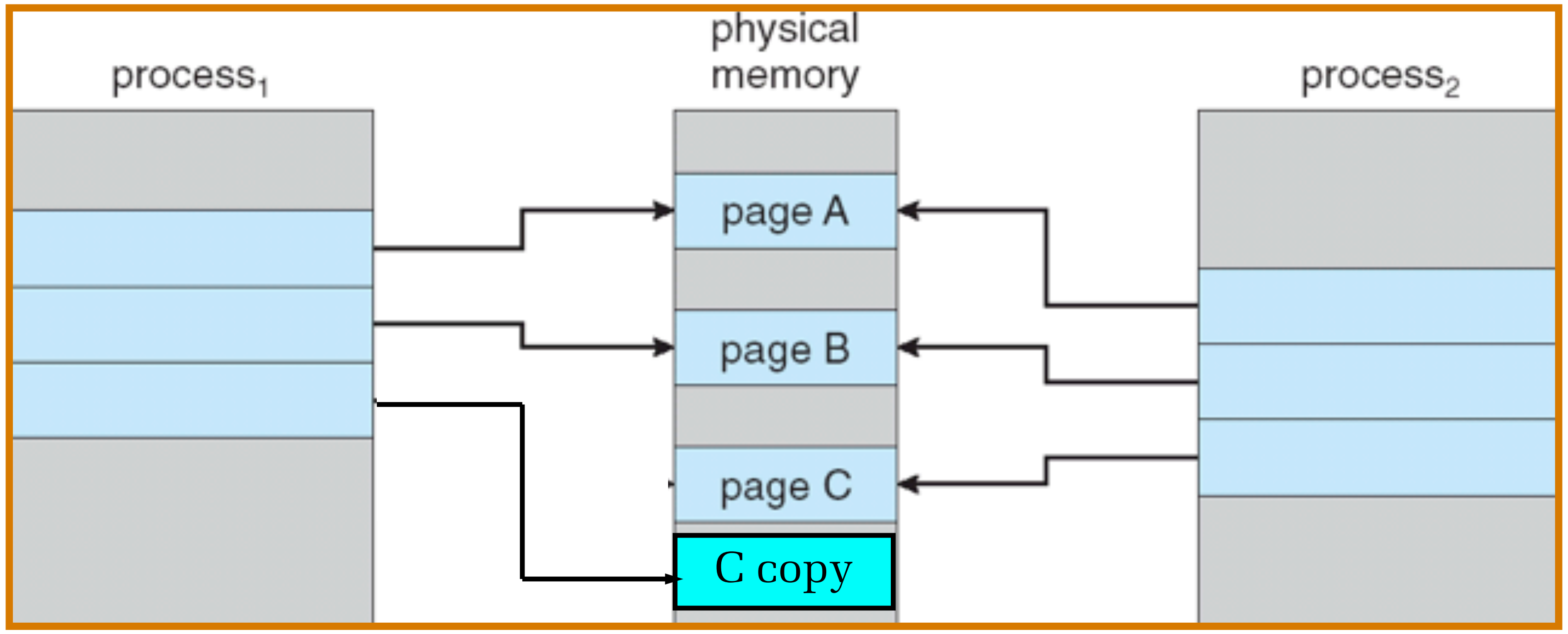
UNIVERSITY OF OREGON

Before Process 1 modifies Page C...



COW (ctd.)

After Process 1 modifies Page C…



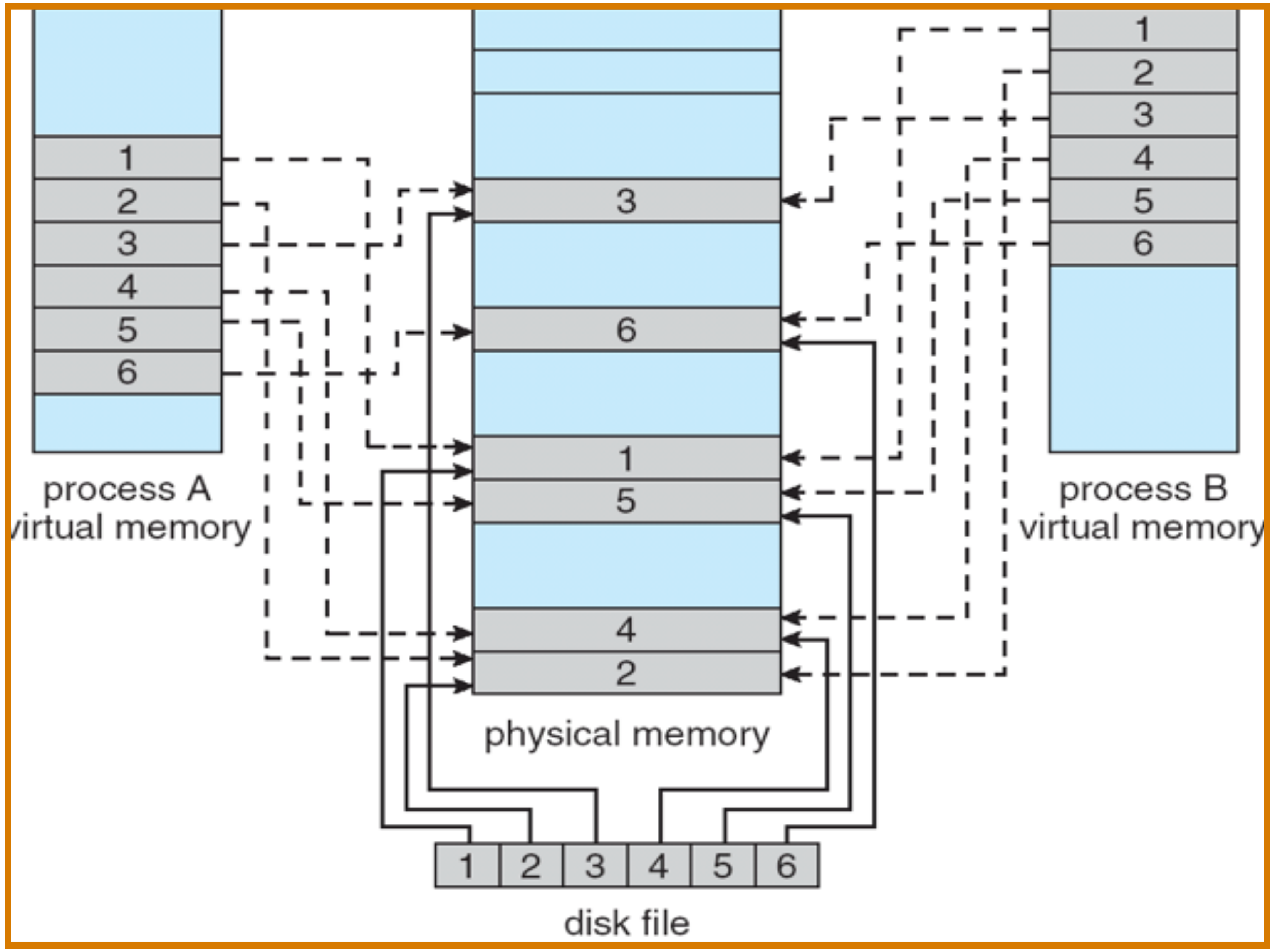
Memory-Mapped Files

- Memory-mapped file I/O allows file I/O to be treated as *routine memory access* by **mapping** a disk block to a page in memory
  - File is initially read using demand paging
  - Page-sized portion of the file is read from the file system into a physical page
  - Subsequent reads/writes to/from the file are treated as ordinary memory accesses.

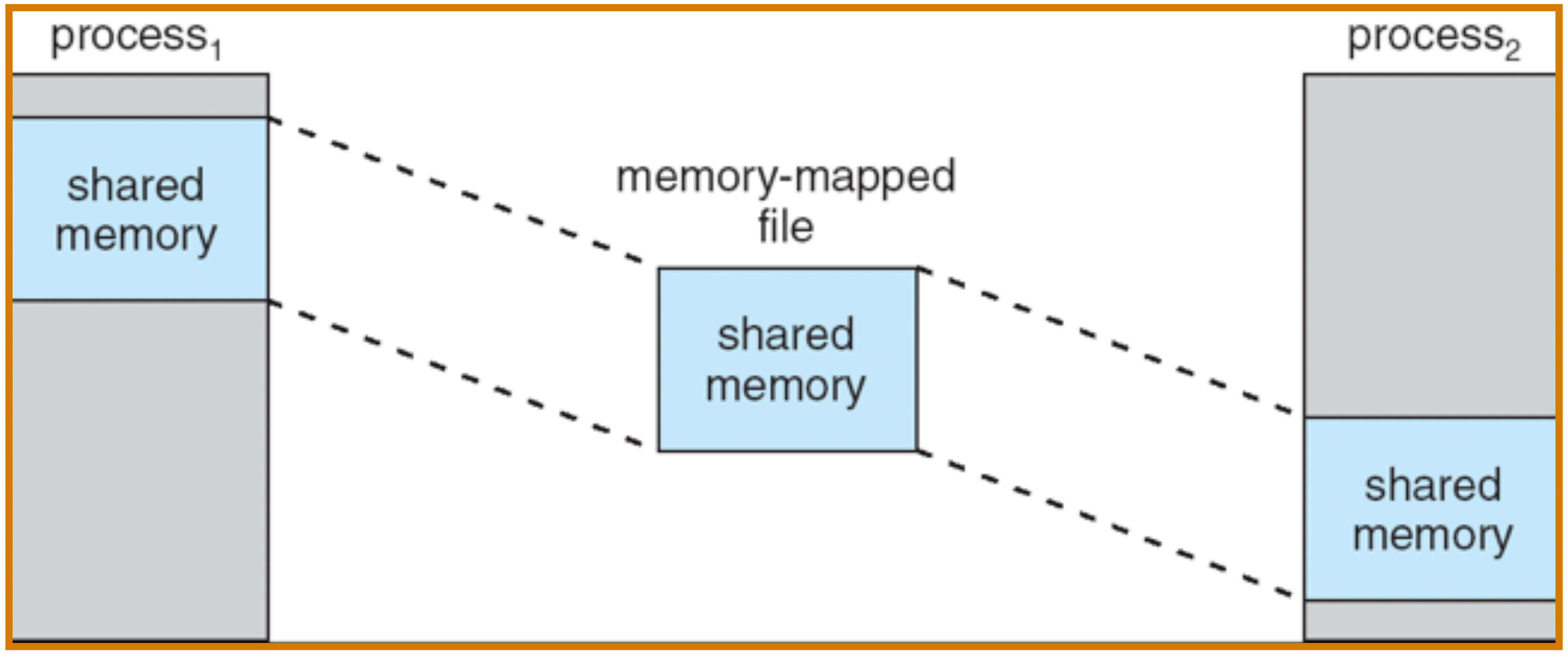
Memory Mapping Benefits

- Simplifies file access by treating file I/O through memory rather than `read()` or `write()` system calls
    - What is the benefit of doing this?
- Also allows several processes to map the same file allowing the pages in memory to be shared

Memory Mapped Files



Memory-Mapped Shared Mem

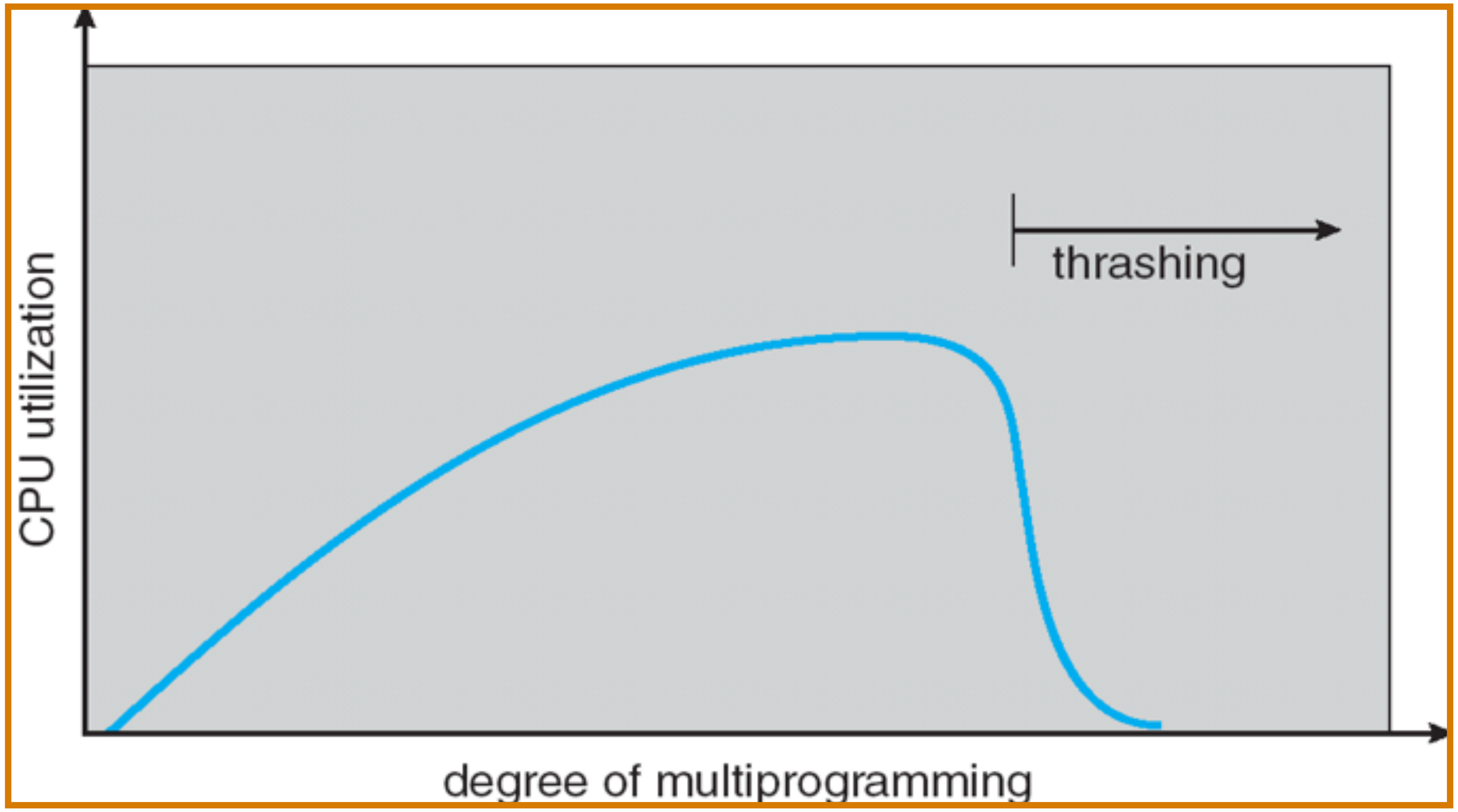


Thrashing

UNIVERSITY OF OREGON

- If a process does not have "enough" pages, the page-fault rate is very high. This leads to:
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
  - another process added to the system
- *Thrashing* ≡ a process is busy swapping pages in and out

Thrashing



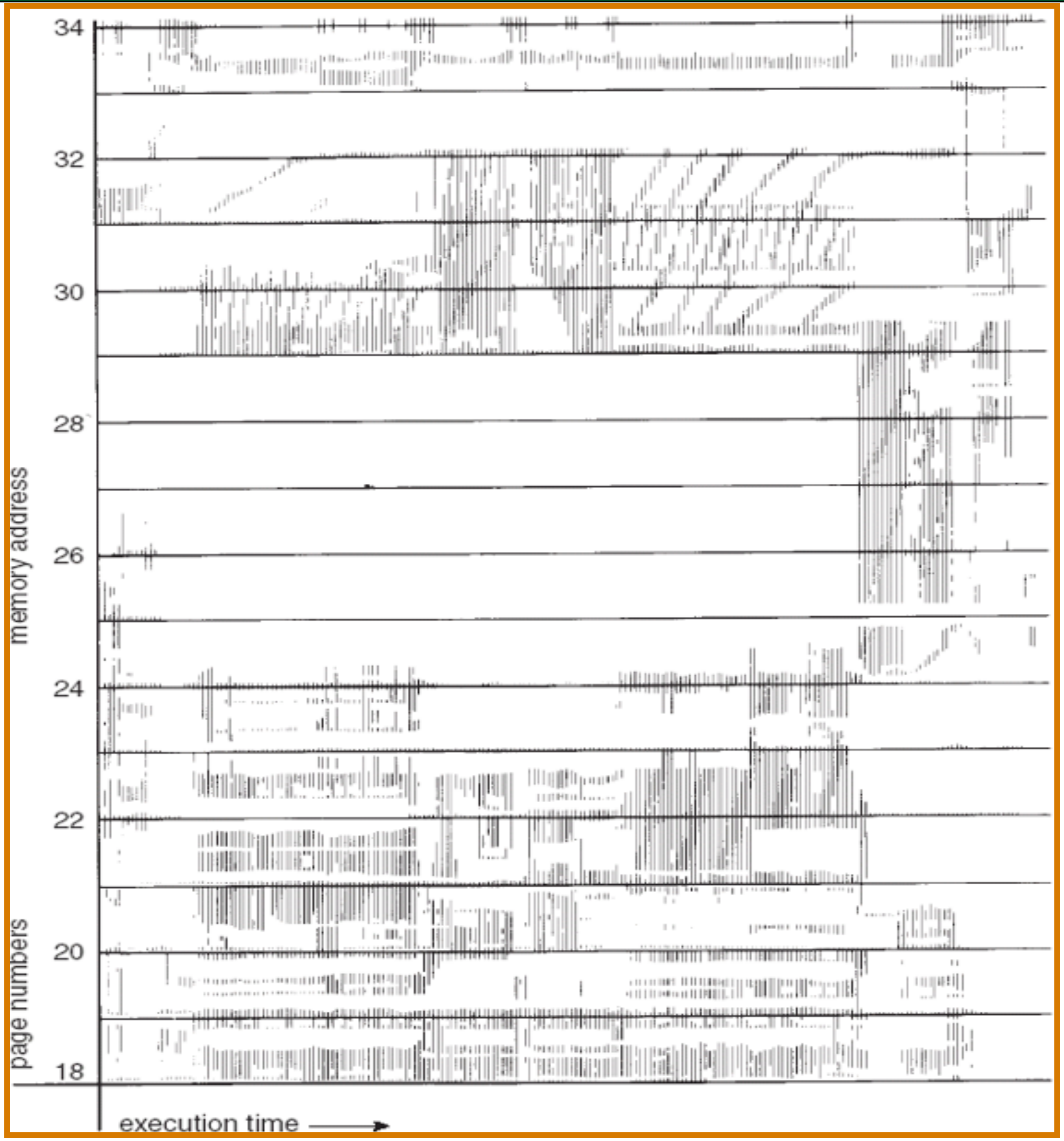
Demand Paging & Thrashing

UNIVERSITY OF OREGON

- Why does demand paging work?
  Locality model
  - Process migrates from one locality to another
  - Localities may overlap
- Why does thrashing occur?
  Σ size of locality > total memory size

Memory-Reference Locality

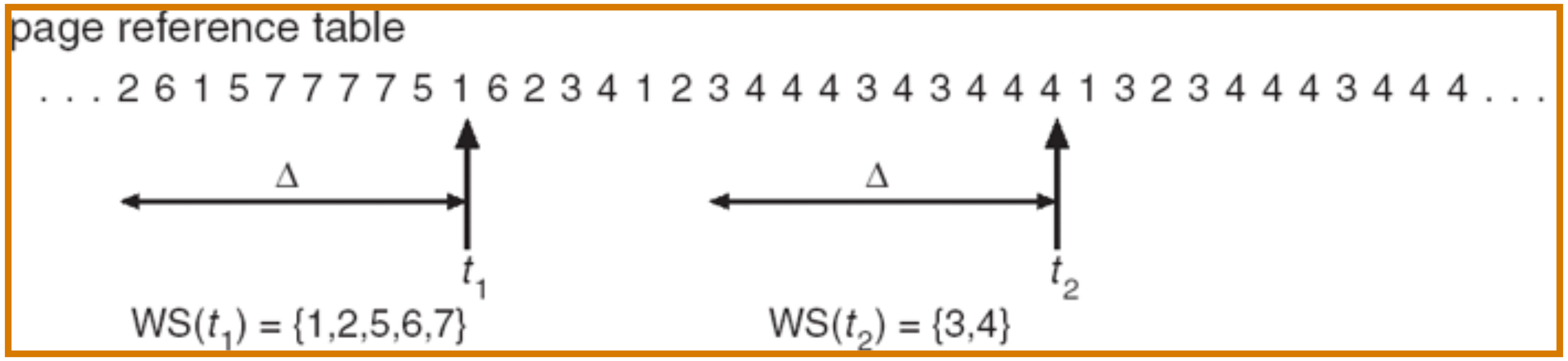
UNIVERSITY OF OREGON



Working-Set Model

- $\Delta \equiv$ *working-set window* $\equiv$ a fixed number of page references (e.g., 10,000 instructions)
- WSS_i (working set of Process P_i) = total number of pages referenced in the most recent Δ (varies in time)
  - if Δ too small, will not encompass entire locality
  - if Δ too large, will encompass several localities
  - if $\Delta = \infty \Rightarrow$ will encompass entire program
- $D = \Sigma\, WSS_i \equiv$ total demand frames
- if $D > m \Rightarrow$ Thrashing
- Policy: if $D > m$, suspend one of the processes

Working-set model



Sliding window that
approximates program locality

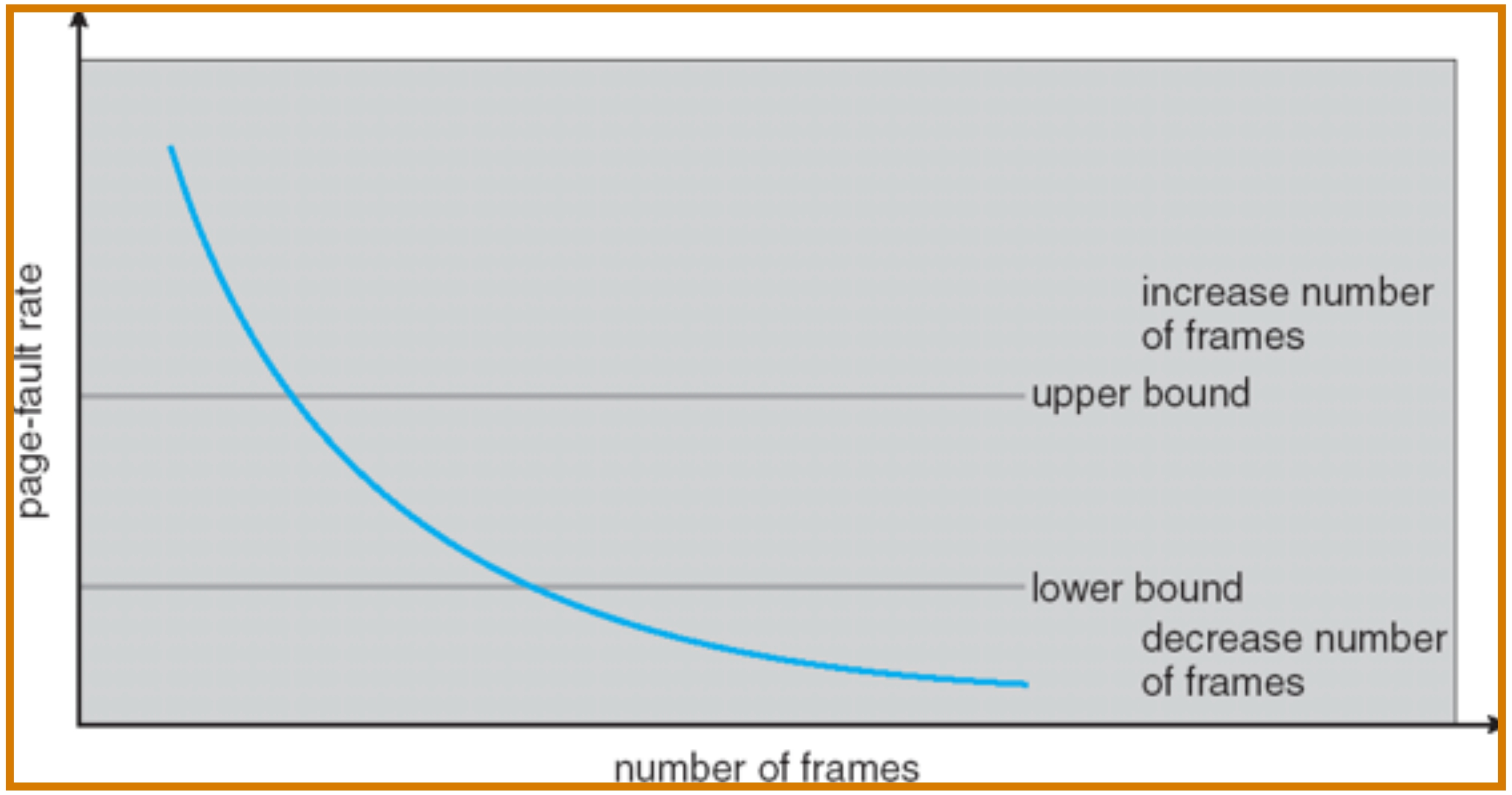
Tracking the Working Set

- Approximate with interval timer + reference bits
- Example: Δ = 10,000
  - Timer interrupts after every 5000 time units
  - Keep in memory 2 bits for each page
  - Whenever a timer interrupts copy and set the values of all reference bits to 0
  - If one of the bits in memory = 1 ⇒ page in working set
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units

Page-Fault Frequency

UNIVERSITY OF OREGON

- Establish "acceptable" page-fault rate
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame



Summary

- Uses
  - Shared Pages
  - Copy-on-write
  - Memory-mapped files
- Thrashing and the Working Set model

- Next time: Files