Comp 310
Computer Systems and Organization

Lecture #18
Virtual Memory
(Issues & Techniques – Part 2)

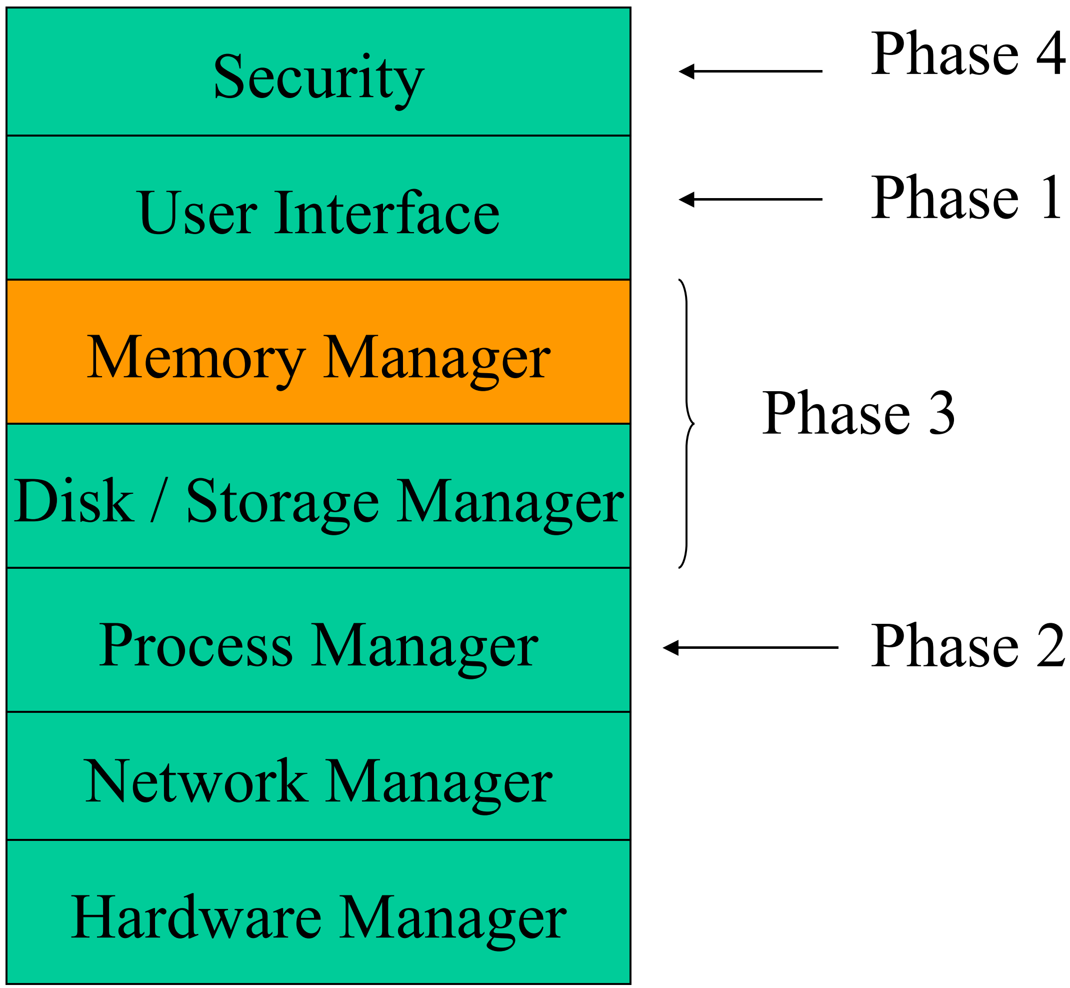
Prof. Joseph Vybihal

Announcements

- Course evaluation:
  - Minerva
  - Important to participate

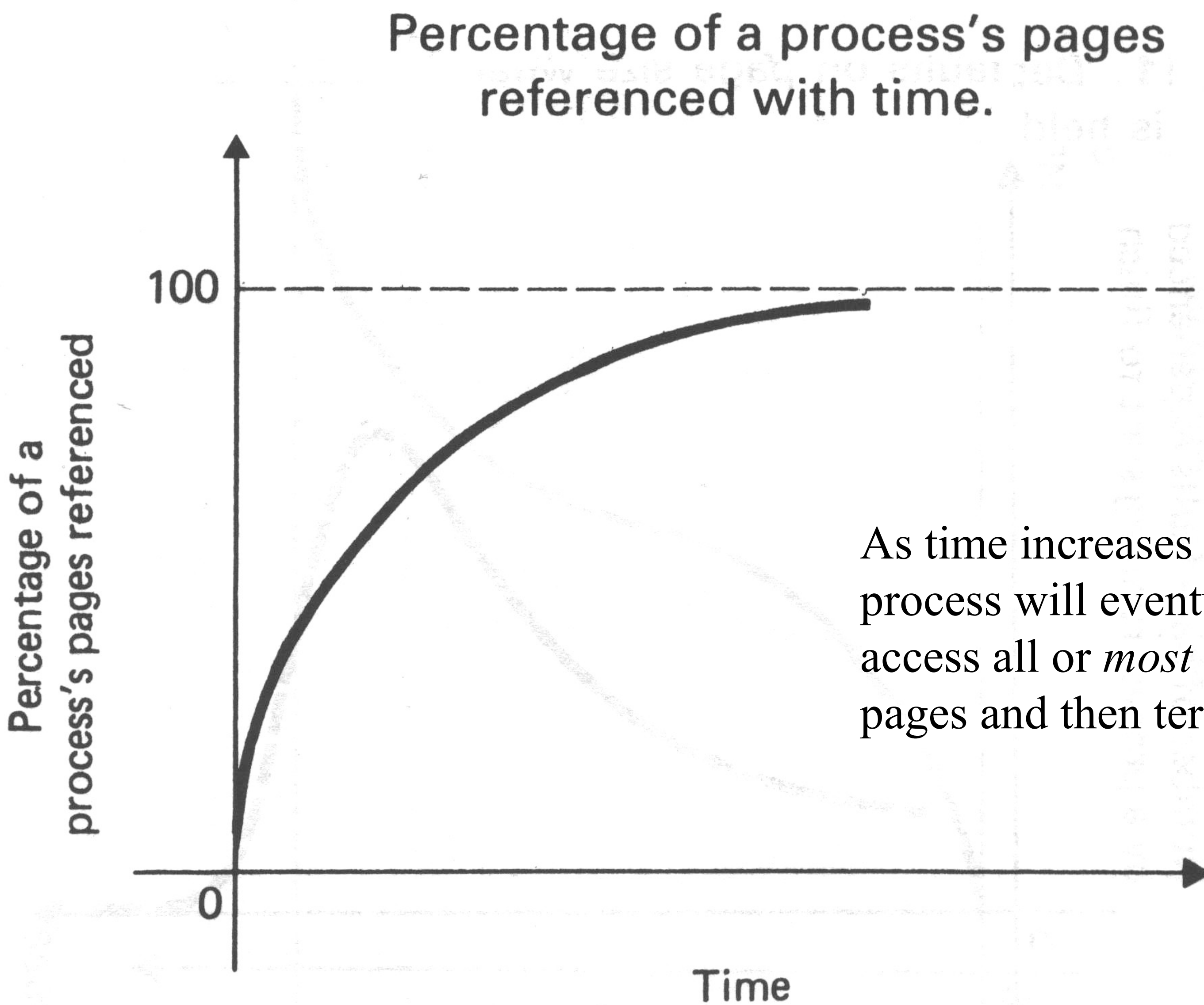
Basic OS Architecture

(Course Table of Contents)

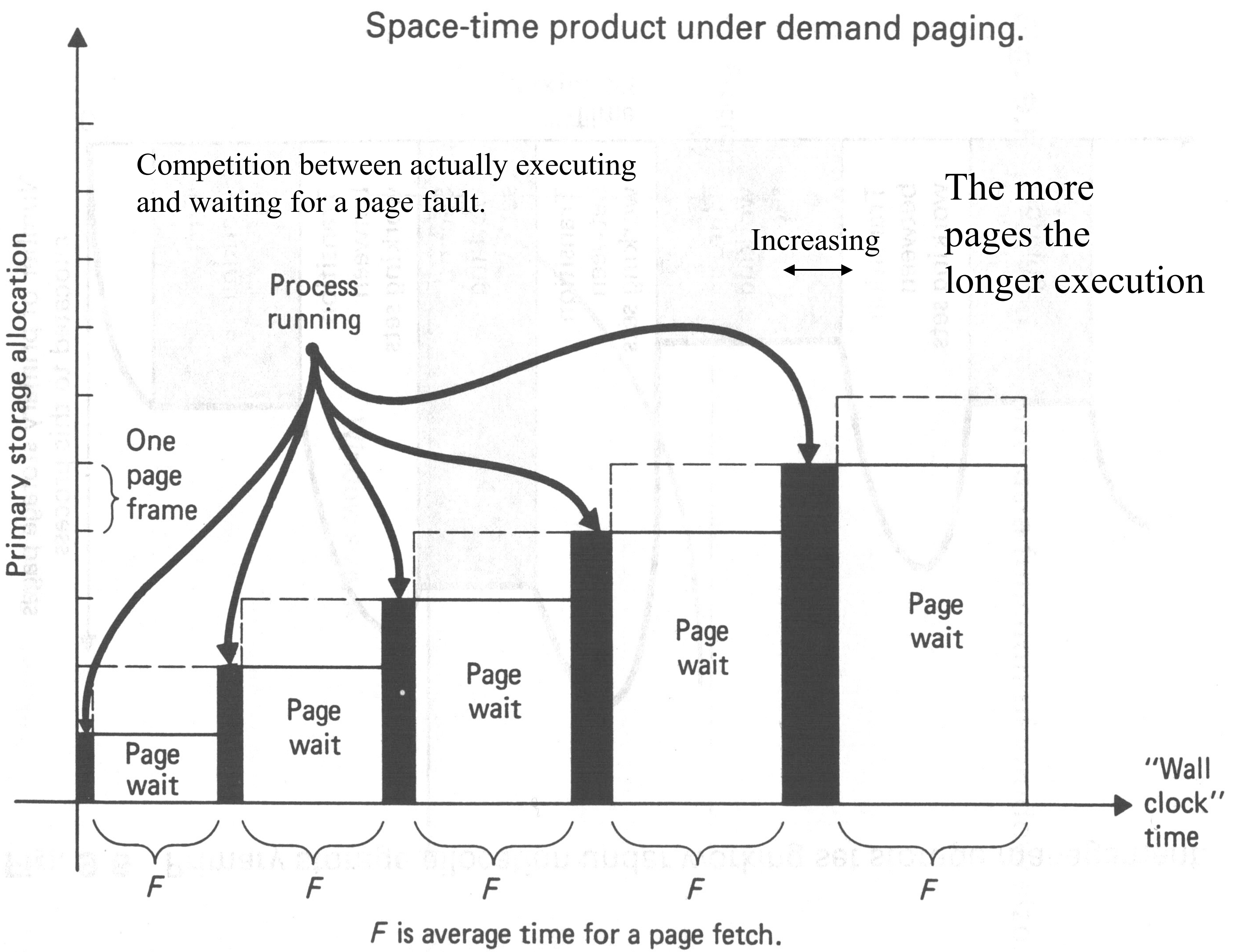


Part 1

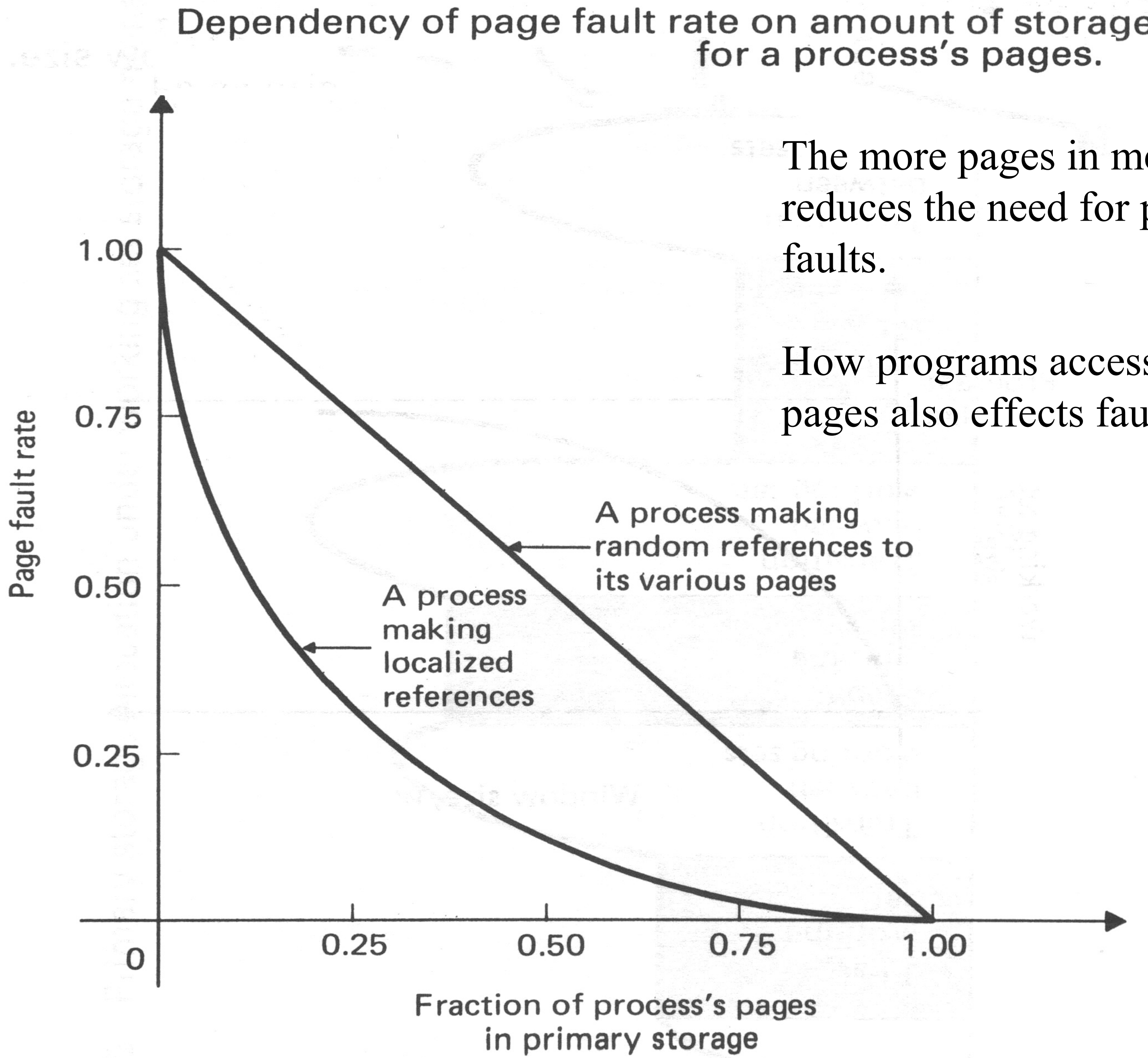
Page Replacement Issues



As time increases the process will eventually access all or *most* of its pages and then terminate



Space-time product under demand paging.
Competition between actually executing and waiting for a page fault.
Increasing
The more pages the longer execution
Primary storage allocation
Process running
One page frame
Page wait
Page wait
Page wait
Page wait
Page wait
"Wall clock" time
F
F
F
F
F
F is average time for a page fetch.



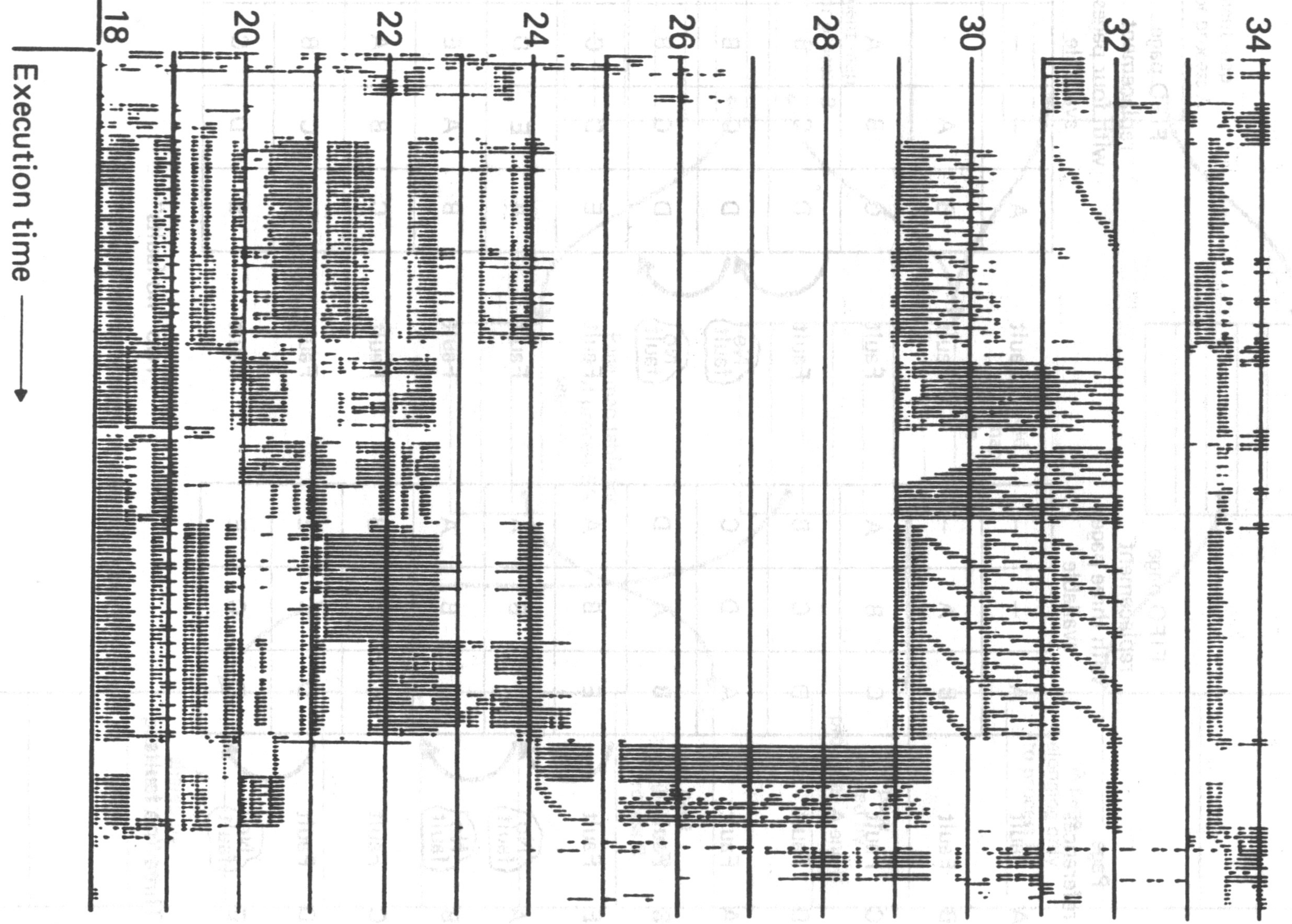
The more pages in memory reduces the need for page faults.

How programs access pages also effects fault rate.

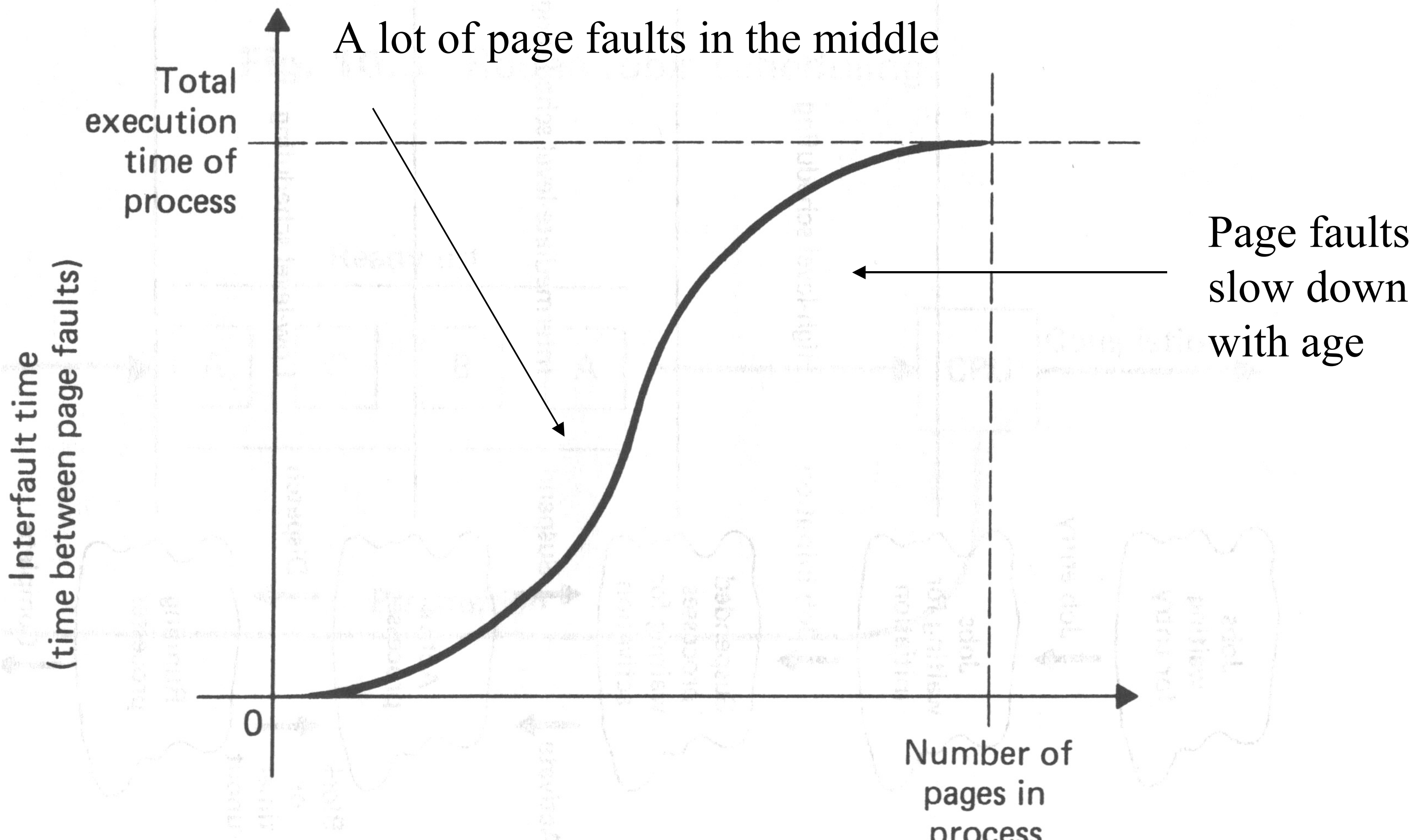
Storage address

Storage reference pattern exhibiting locality.

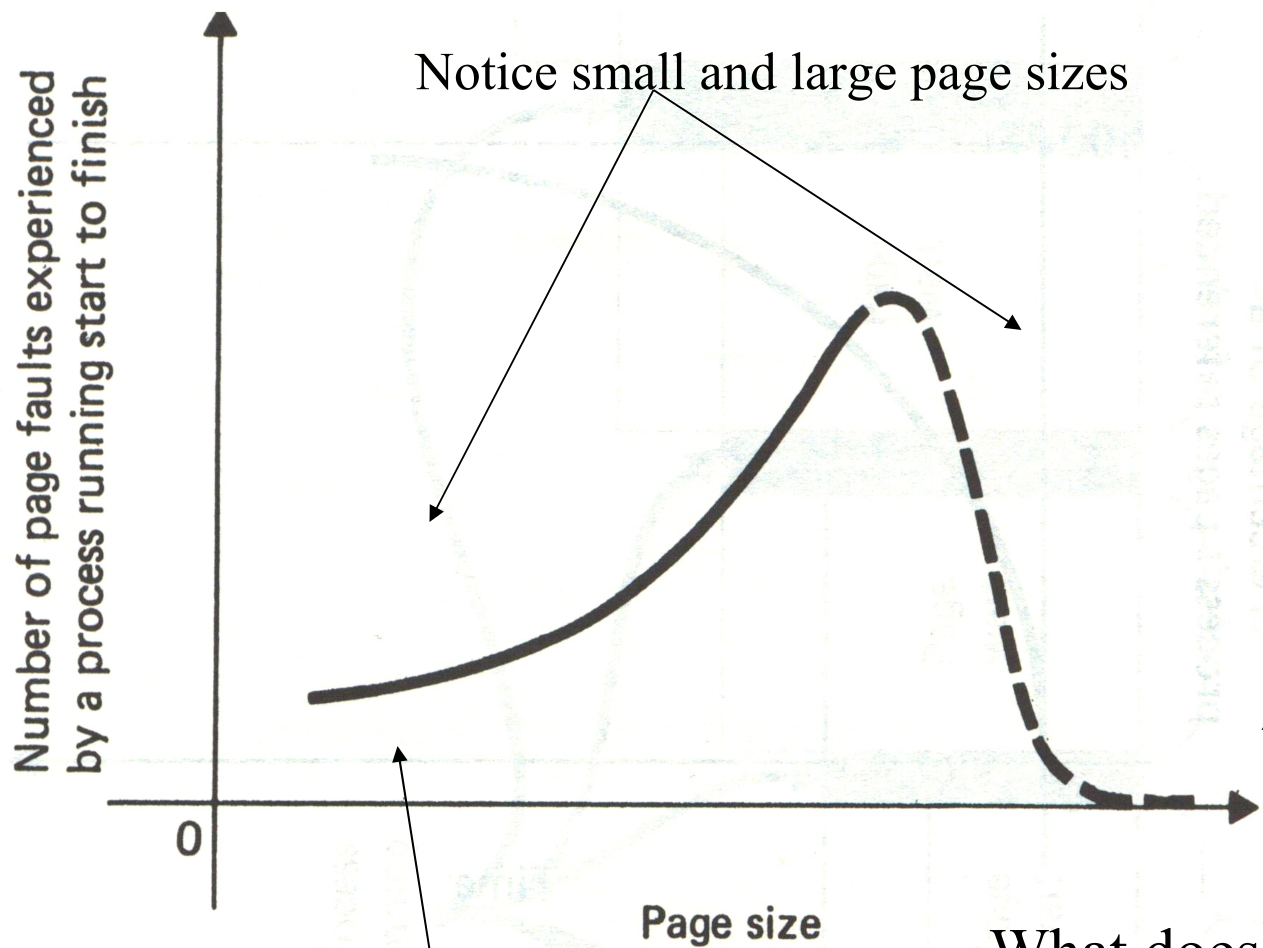
Page numbers



Dependency of interfault time on the number of page frames allocated to a proce



Dependency of page faults on page size when primary storage is held constant.



Notice small and large page sizes

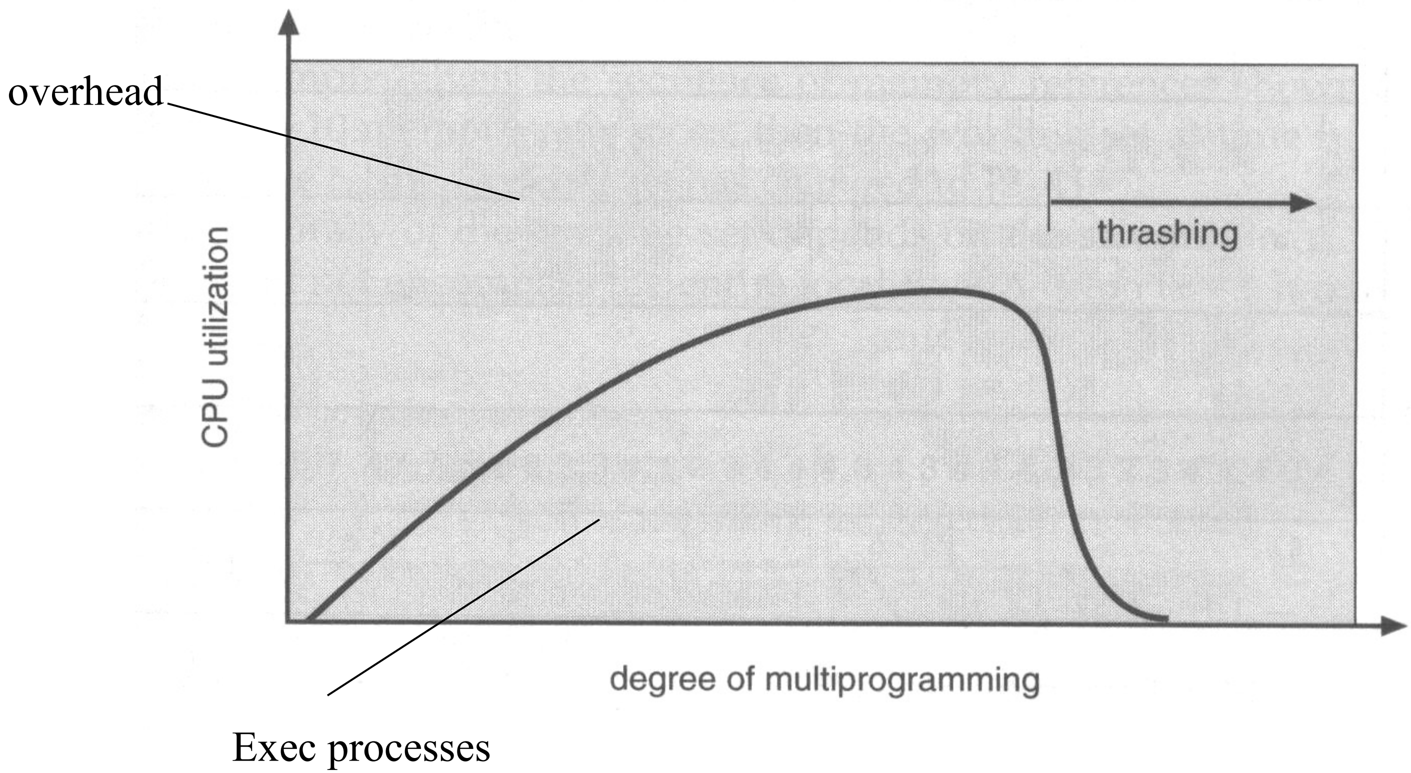
What does this mean?

Page Size Issues

- Equal Allocation
  - Everyone gets the same number of pages
  - All frames are the same size
- Proportional Allocation
  - frames = program size / VM size * total frames
- Categorized Replacement
  - Global Set (any page from any process)
  - Local Set (pages from only your process)
- On Demand vs. Intelligent Allocation

Thrashing

Question: How many frames should a program be allowed to have at any time? (equal? Incremental? Set no.?)

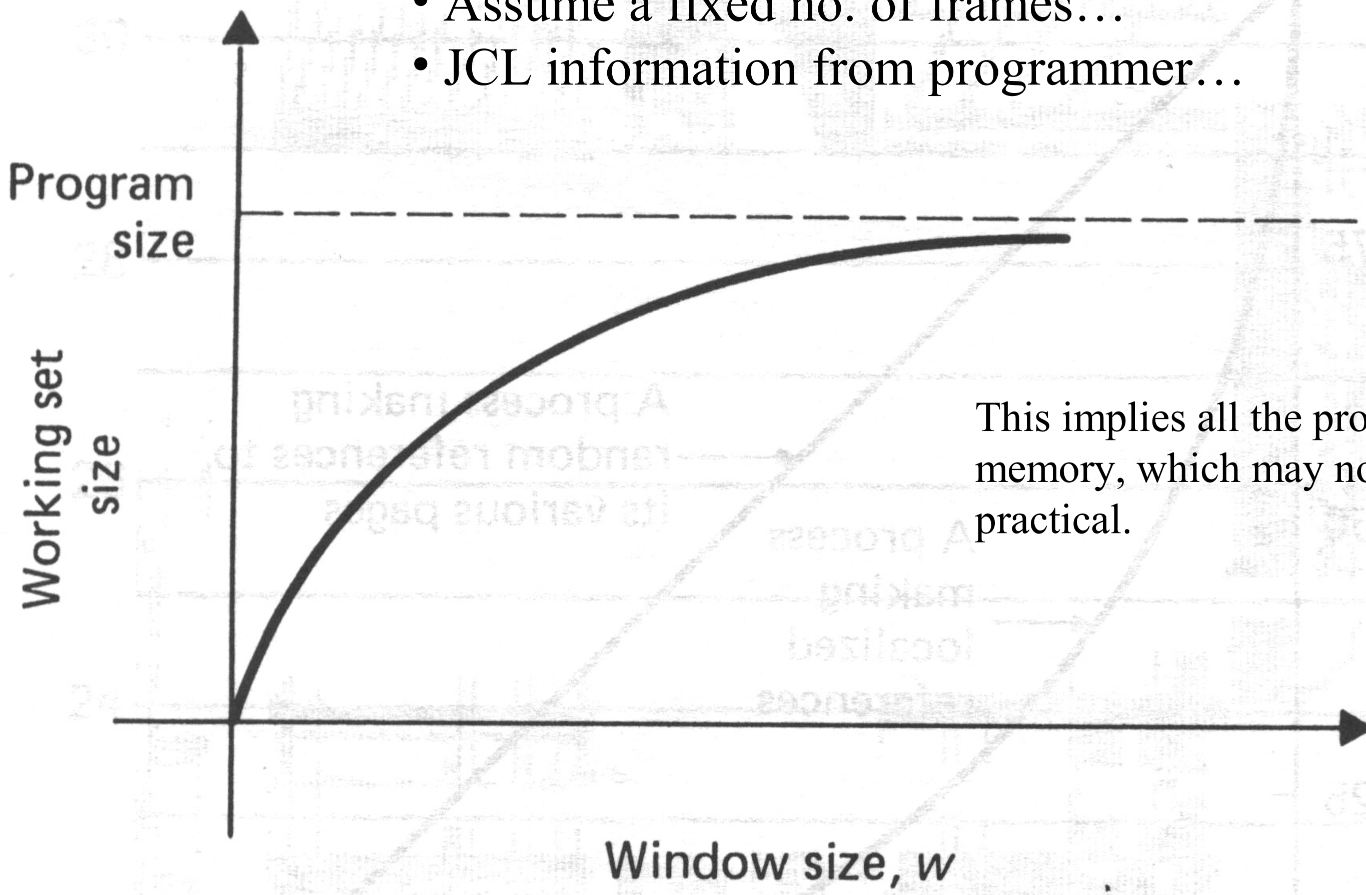


If we do not have enough pages to support the number of programs in memory, we get a situation where the CPU spends most of its time managing page faults

Working Set – A Solution to Thrashing?

How to determine a working set?

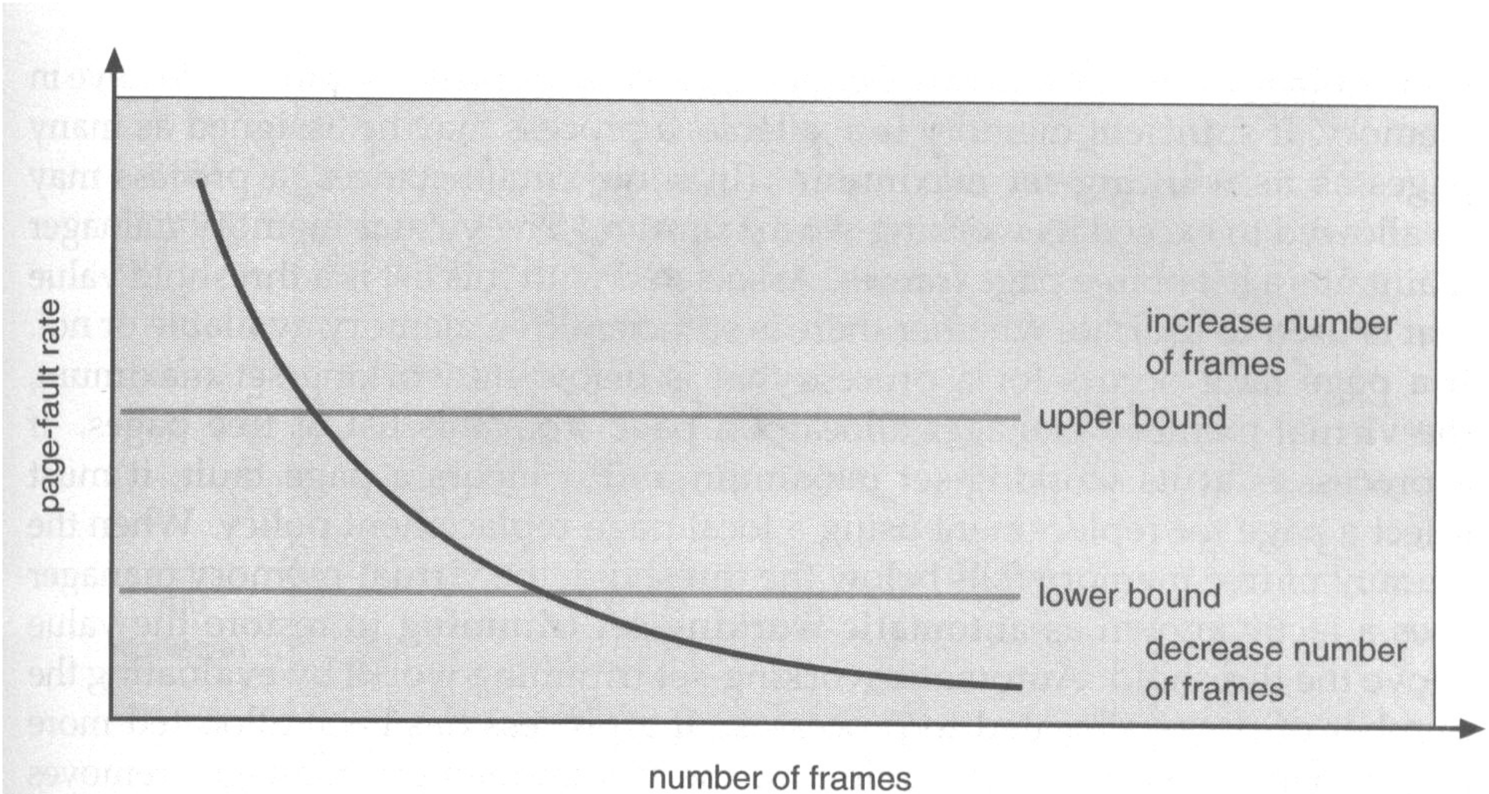
- Assume a fixed no. of frames…
- JCL information from programmer…



This implies all the program in memory, which may no be practical.

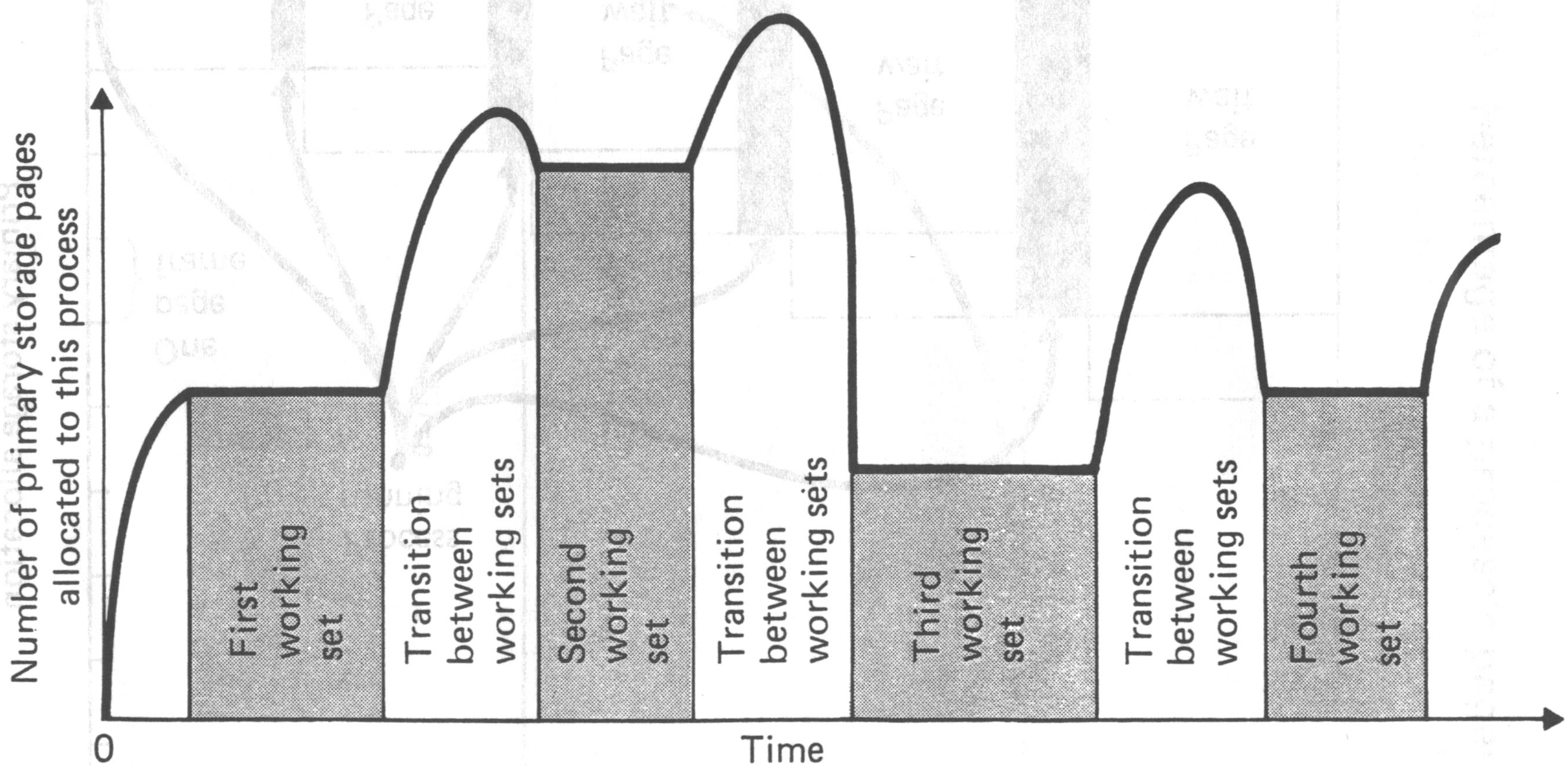
Increasing the working set is like increasing the window size

Thrashing Management



OS built with a lower bound and upper bound value that determines the number of frames a program should own

Primary storage allocation under working set storage management.



Part 2

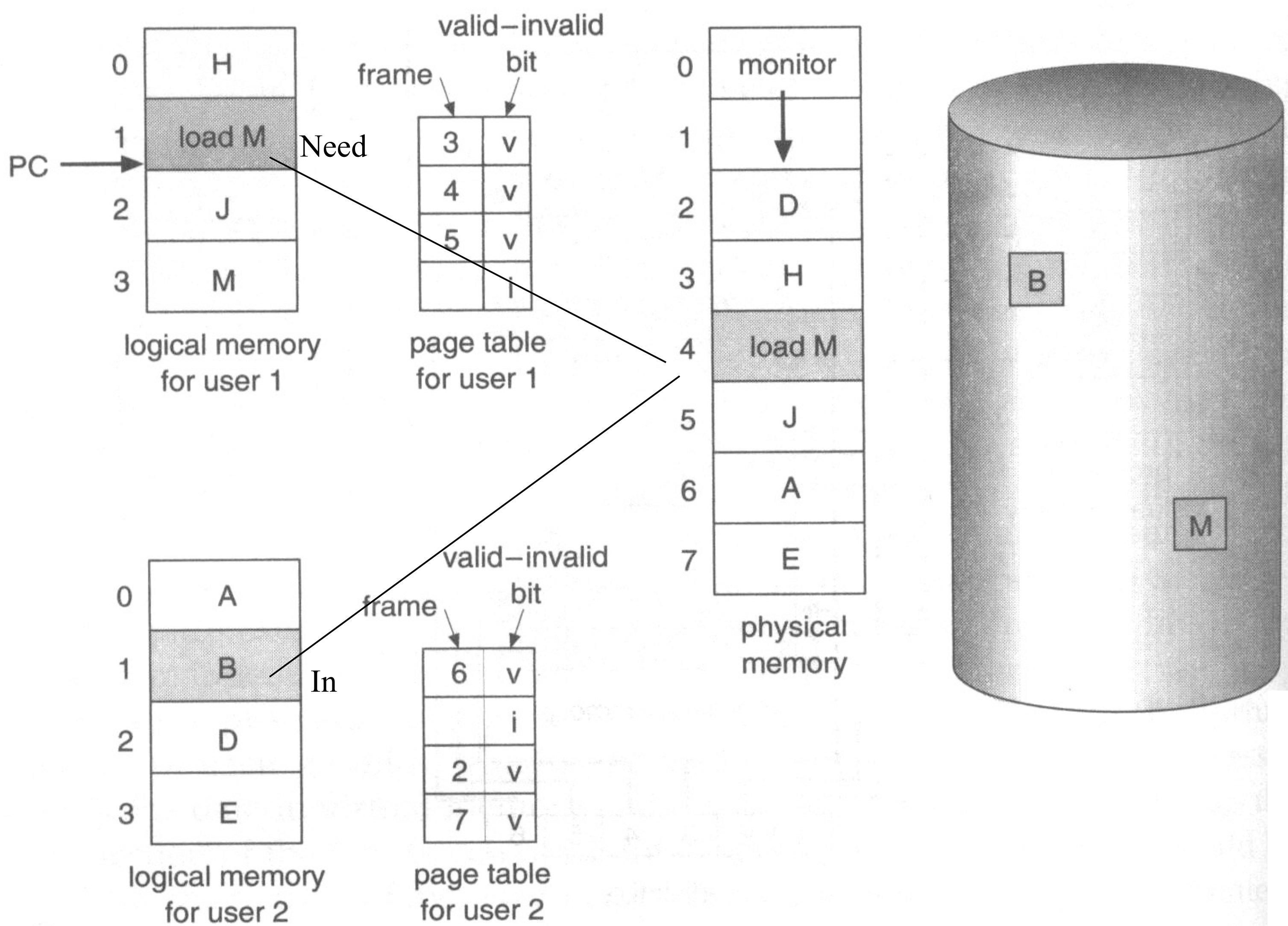
Page Replacement Techniques

Basic Algorithm

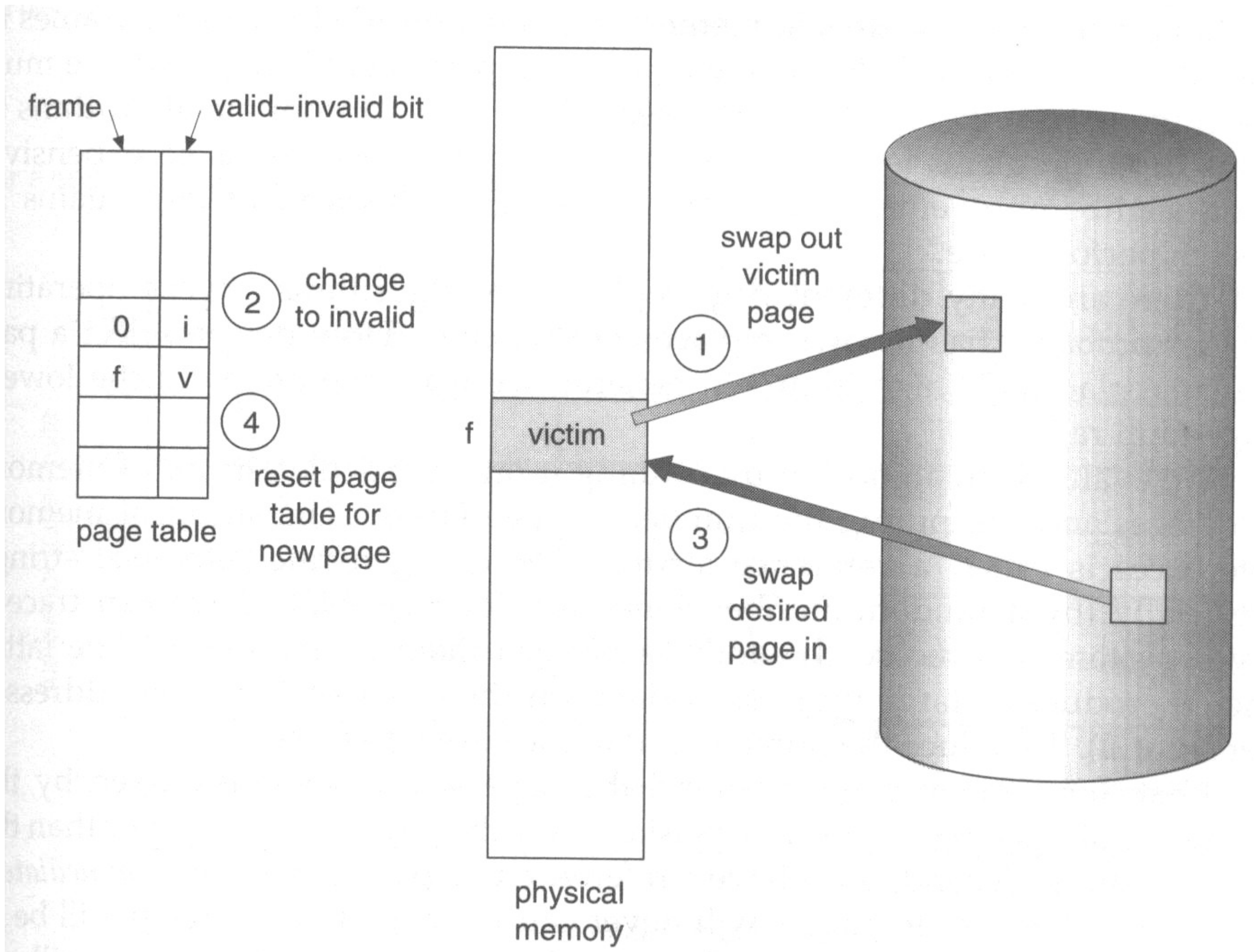
1. Process executes
2. Process generates a page fault, at this moment
3. Hardware traps to OS
   1. Fault = page fault or illegal memory access?
      - If illegal access then terminate process
   2. If page fault then
      1. Find page on disk
      2. Find free frame in RAM
         - If found then allocate page to frame
      3. If no free memory then….
         A) terminate the process asking for the page fault
         B) terminate another process on a queue (ready/wait)
         C) page replacement

Page Replacement (1/3)

1. Find page on disk
2. Find a free frame
    1. If free fame exists then use it
    2. If no free frame then select a victim frame
        - Write the victim's page to disk (modify flag/"dirty bit")
        - Adjust tables
3. Read the desired page into the frame
    - Update tables
4. Restart the user's process



valid–invalid bit
frame
0 H
1 load M
2 J
3 M
PC
Need
3 v
4 v
5 v
i
logical memory for user 1
page table for user 1
0 monitor
1
2 D
3 H
4 load M
5 J
6 A
7 E
physical memory
B
M
0 A
1 B
2 D
3 E
In
6 v
i
2 v
7 v
logical memory for user 2
page table for user 2


frame
valid–invalid bit
0
i
f
v
page table
2
change to invalid
4
reset page table for new page
f
victim
physical memory
1
swap out victim page
3
swap desired page in

But how do we select the victim?

- FIFO? (first-in first-out)
- Optimal? (won't be needed for longest time)
- LRU? (least-recently-used)
- Second chance? (FIFO w/ reference bit = 0)
- LFU? (least frequently used)
- MFU? (most frequently used)

FIFO

The FIFO Anomaly.

Page references		FIFO page replacement with three pages available				FIFO page replacement with four pages available			
A	Fault	A	—	—	Fault	A	—	—	—
B	Fault	B	A	—	Fault	B	A	—	—
C	Fault	C	B	A	Fault	C	B	A	—
D	Fault	D	C	B	Fault	D	C	B	A
A	Fault	A	D	C	No fault	D	C	B	A
B	Fault	B	A	D	No fault	D	C	B	A
E	Fault	E	B	A	Fault	E	D	C	B
A	No fault	E	B	A	Fault	A	E	D	C
B	No fault	E	B	A	Fault	B	A	E	D
C	Fault	C	E	B	Fault	C	B	A	E
D	Fault	D	C	E	Fault	D	C	B	A
E	No fault	D	C	E	Fault	E	D	C	B

Three "no faults"

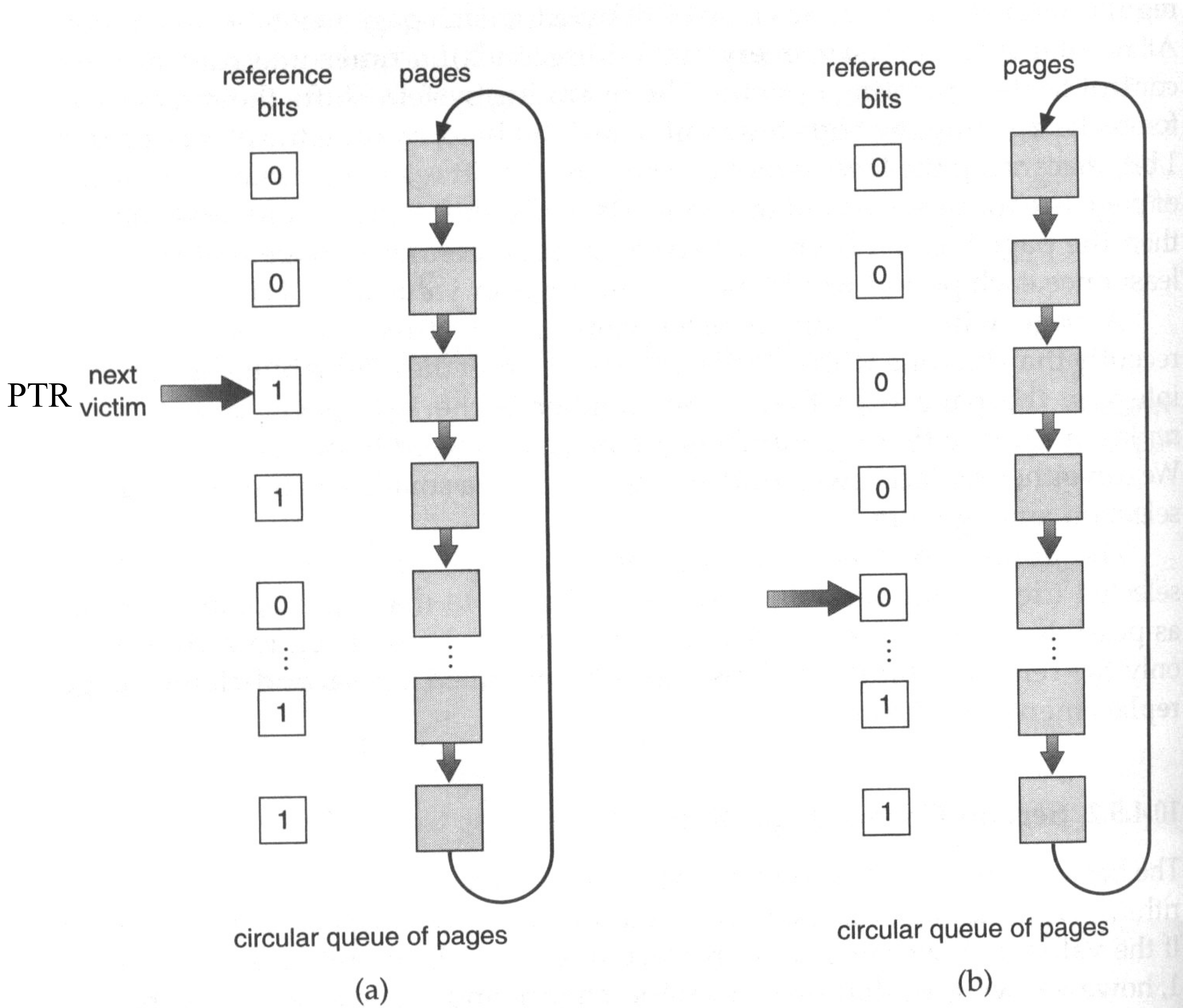
Two "no faults"

Optimal and LRU

- Problem: we can not see into the future…
- Rule of thumb:
  time stamp the last time it was used
  – Interpretation:
    - Was not used for a long time? So, remove …
    - It is due to be used soon? So, maybe keep…?

Second Chance

- Page frame table has an flag variable that is set to TRUE when the page is used.
- A timer is set to clear all bits to zero.
- Page faults are handled before the bit is set to zero.
  - Rule:
    - Victim if FIFO and flag == 0



Time-out

A Second chance implementation (circular queue)

LFU & MFU

- Page tables use an integer variable to count the number of time the page was referenced while in memory.
  - LFU: remove lowest integer
  - MFU: remove highest integer
- When page reloaded into RAM it is set to zero

Questions

- What data structures in C could we use to implement each of the techniques?
- What algorithms?

Part 3

At Home

Things to try out

1. Find the virtual memory page swap area in your OS. Change its size!!

2. Web Resources:
   - http://users.actcom.co.il/~choo/lupg/tutorials/unix-memory/unix-memory.html
   - http://developer.apple.com/documentation/Darwin/Conceptual/KernelProgramming/Mach/chapter_6_section_5.html
   - http://www.windowsitlibrary.com/Content/356/04/1.html
   - http://www.awprofessional.com/articles/article.asp?p=167857&rl=1
   - http://en.wikipedia.org/wiki/Memory_management_unit