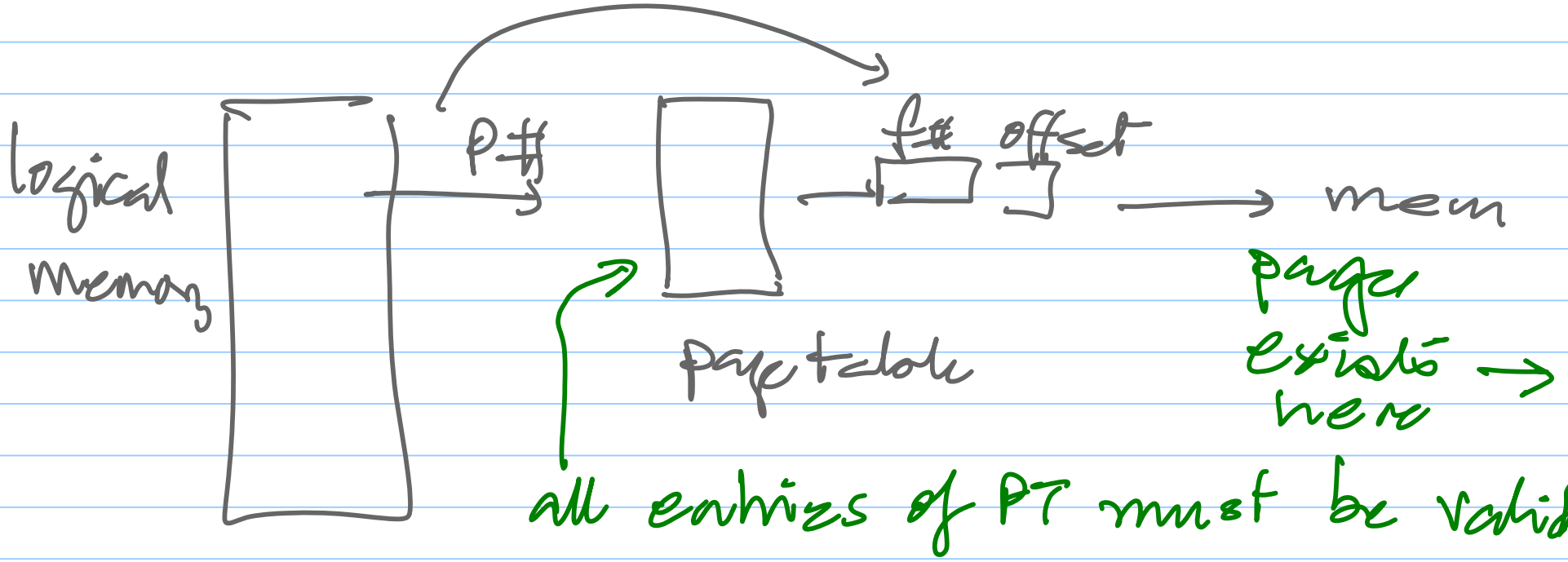
CSE 330: Operating Systems

Class: 19 **Date:** 10/27

Fall 2016

Note Title



→ All pages of a process must be in memory

→ up to 2GB / process

100's of processes

↳ can be too large

Virtual Memory

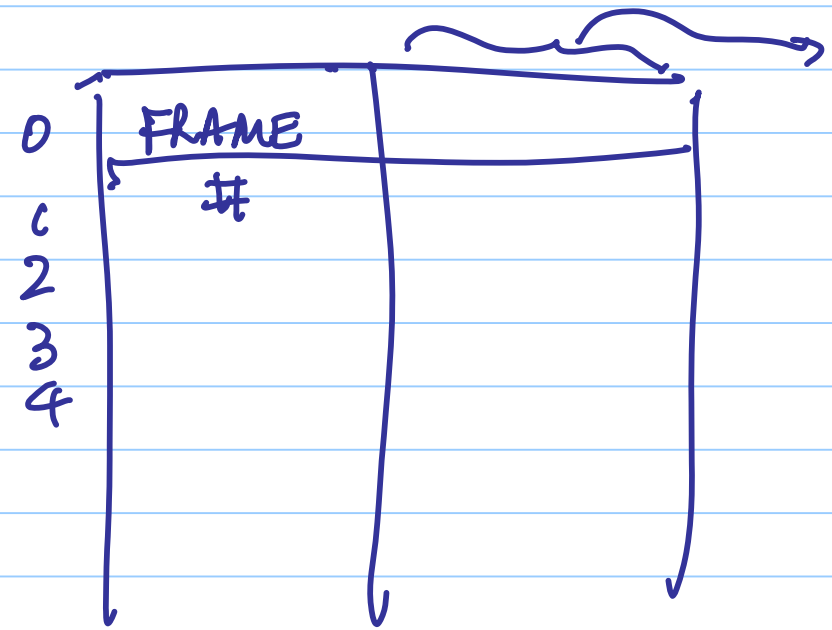
→ memory used by processes can be larger than physical memory

→

Virtual Memory → concept

Demand paging → method for implementing the concept

extra bits in the page table



V - valid
→ is this page in memory?

R - reference
has the page been referenced since the R bit was set to 0

permissions R, W, X

M → modify →

what happens if ---

V bit is 0, page is accessed → trap → page fault

error, interrupt generated.

Mod/Ref → no traps

R/W/X → generates traps → access violation

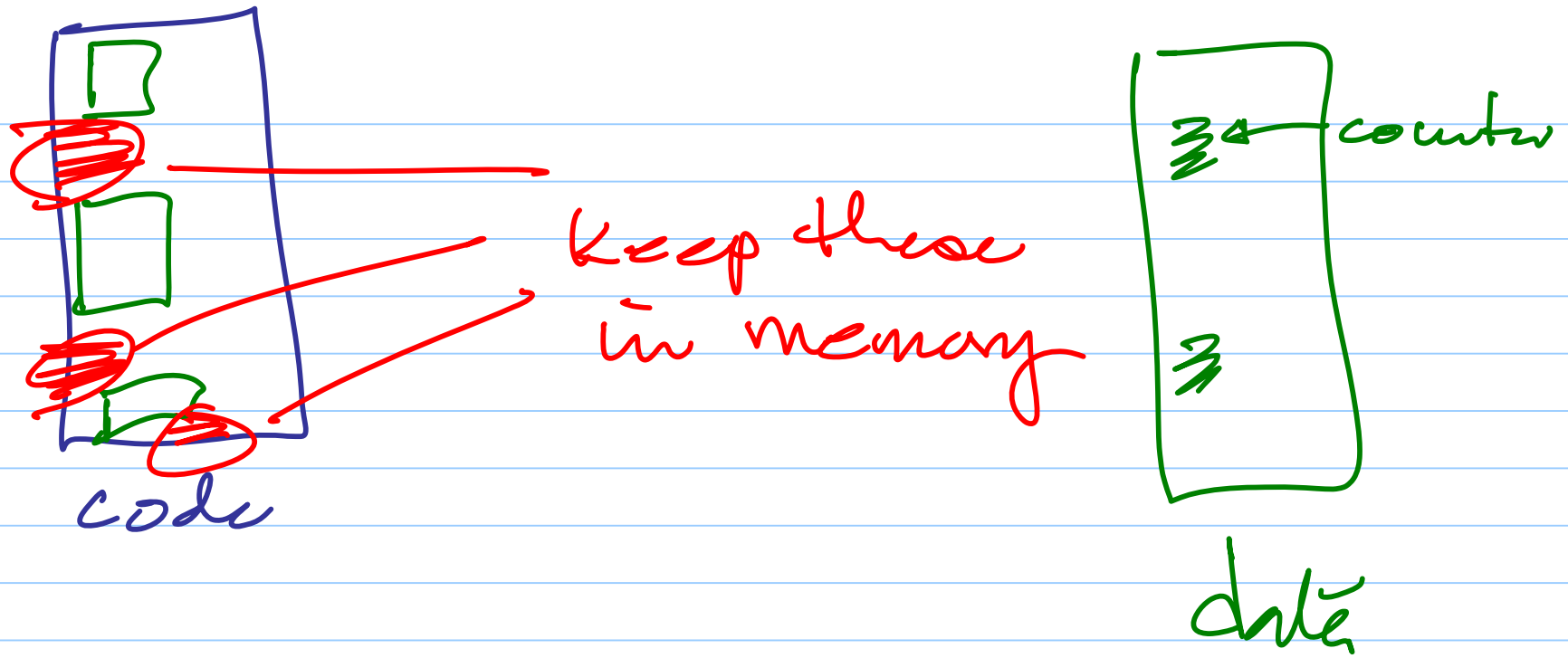
Demand Paging → justify
→ performance?

Process have "locality"

Locality

→ processes access the same set of pages, frequently

↳ common functions
→ data
top of stack


keep these in memory
code
counter
data

Working set

↳ the set of pages that a process uses frequently (data, code, etc)

→ working ~~set~~ sets change over time

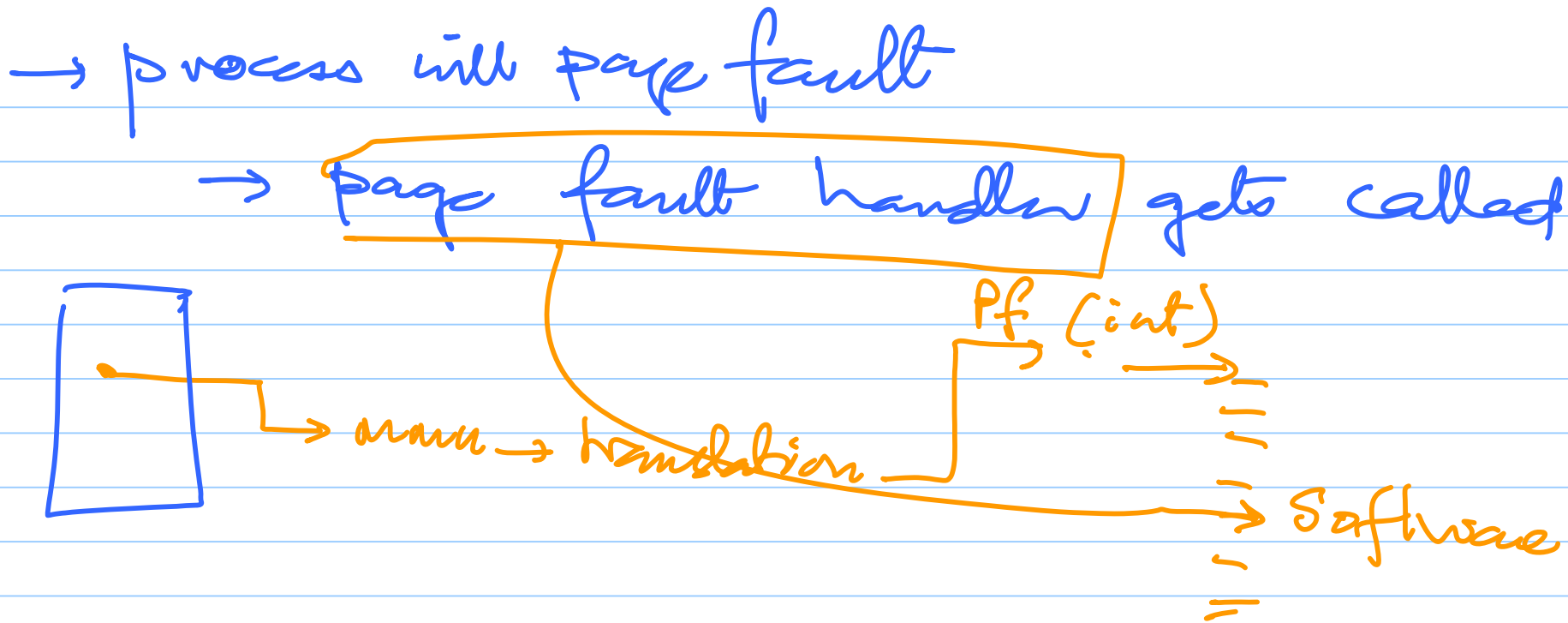
pure demand paging
(lazy)

Start a process

→ allocate page tables

→ do not allocate any memory

→ set all V bits to 0


→ process will page fault
→ page fault handler gets called
Pf (int)
mmu → translation
Software

page fault handler

{ locate the page on swap area

allocate one frame of memory

what if mem is full ?

copy page from disk to memory

set V bit to 1

set frame # of PTE to address of frame

}

- page faults always bring in pages on demand
- pages no longer needed → no indication!
- memory will/may fill up.

- one instruction may cause multiple page faults

e.g. a: mov x → y.

access memory locations a, x, y

→ set of microinstructions

= ← have to context save

page replacement algorithms

→ page replacement
  → when RAM is full
    some page has to
    be replaced. (on page fault)
  → which one

page fault
{
... allocate a frame (RAM is full)
- find a victim frame
- find the PTE pointing to frame
- set V bit to 0
- if m. bit is 1
  copy page to disk
}

(replacement contd)

copy frame to disk block x

put 'x' in frame # field of PTE

copy new frame from disk to mem

= finish page fault handling

V=1, R=0, M=0

Page replacement

→ demand for page A

→ select page B → victim

- page out B
- page in A

victim selection (how?)

FIFO page replacement

→ list of pages paged in (time order)

→ victim is oldest page

FIFO suffers from Belady's Anomaly

Page reference string ...

Process P → AAABCDDDABBC memory ref

ABCDABC page ref

time →

Ref string

→ 1 2 3 4 1 2 5 1 2 3 4 5

5 frame → 1 2 3 4 – – 5 – – – – – (5)

4 frame → 1 2 3 4 – – 5 1 2 3 4 5 (10)

3 frame → 1 2 3 4 1 2 5 – – 3 4 – (9)

Page replacement Algo

→ should be a stack policy

the pages in N frames should be a subset of pages in N+1 frame.

LRU least recently used.

→ choose the page that was not used for the longest time in the past