virtual memory 3

last time

mmap: mapping files into program memory

- shared (modify file) or copy-on-write (do not modify file)

process memory as a list of regions setup with mmap

page cache: memory as cache for (files, program data)

- cache because everything has place on disk/SSD
- hit: managed by CPU with page table
- miss: managed by OS

minimizing page cache misses: Belady's MIN

practical algorithms from working set assumption: LRU-like

accessed and dirty bits

on exam regrades (1)

yes, I wish we proofread the exam more carefully

- and, yes, I'm still trying to get better at foreclosing unexpected (to me) interpretations of questions

in some cases we accepted additional answers that weren't on the key

- e.g. multiple system calls to read file

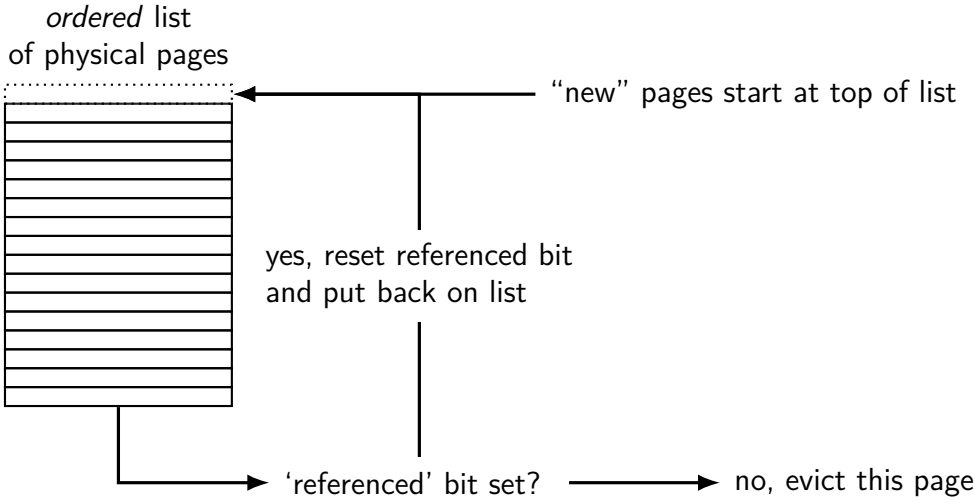
if you wrote comments regrading your interpretation of a question and think they weren't read or read carefully, please submit a regrade

on exam regrades (2)

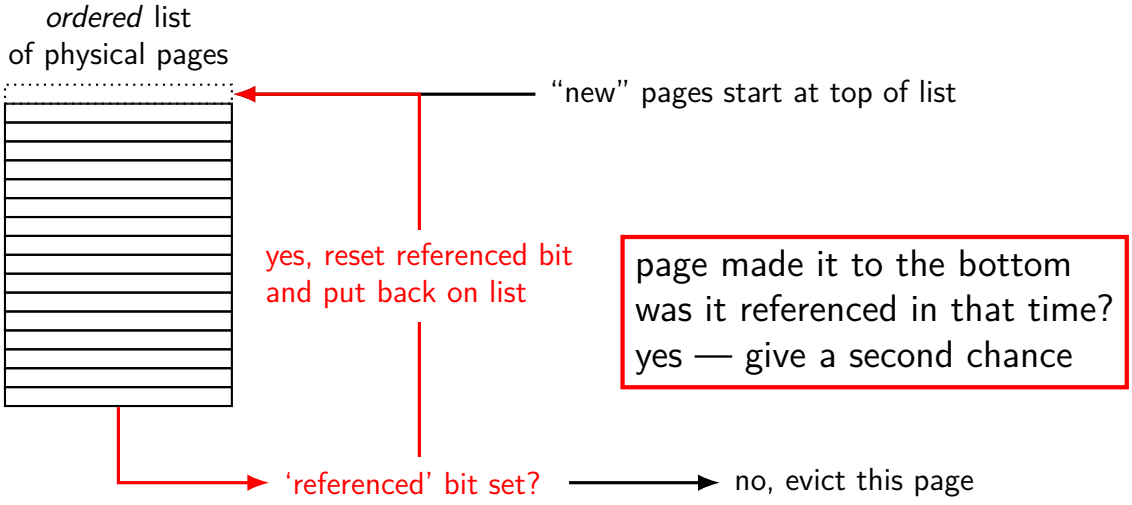
on the monitor question:

> it was pointed out to me that the key we graded with had a condition that caused pending non-videos to wait more often than necessary non-videos should have waited only when a pending video **and 4 non-videos running**
>
> ...can't just omit the pending condition: otherwise new non-video downloads can go in between when the video download is signalled and when it wakes up from the signal
>
> I believed I've scan through all the Q7s to find cases where we marked the fully correct answer wrong;

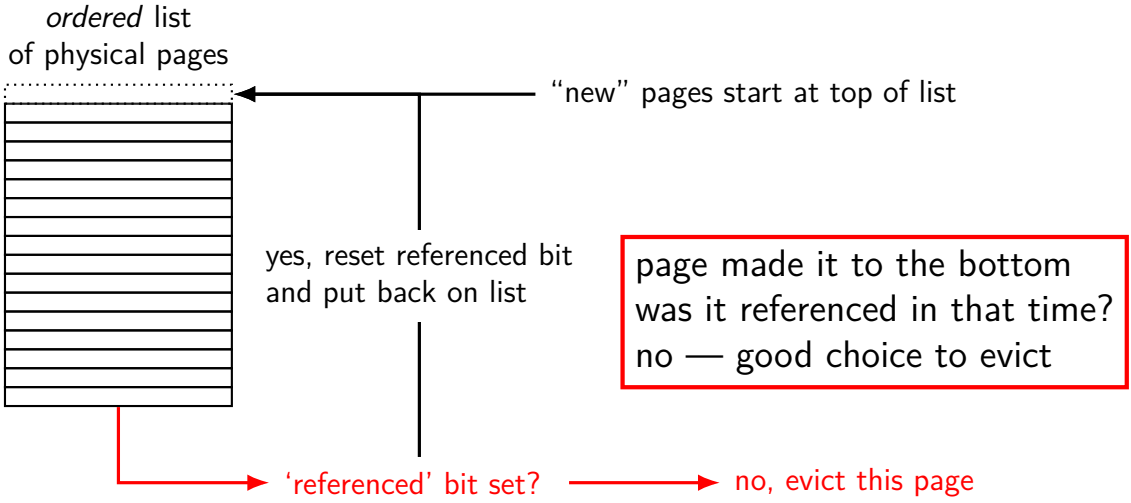
approximating LRU: second chance



approximating LRU: second chance



approximating LRU: second chance



second chance example (1)

	A	B	C	D	—	—	—	B
1	A						D	
2		B						
3			C			C		
page list								
last added	*1R	*2R	*3R	1NR	2NR	3NR	*1R	1R
—	3NR	1R	2R	3R	1NR	2NR	3NR	3NR
end of list	2NR	3NR	1R	2R	3R	1NR	2NR	*2R

second chance example (1)

page 2 was at bottom of list
is not referenced
okay to use

							—	B
1	A						D	
2		B						
3			C			C		
page list								
last added	*1R	*2R	*3R	1NR	2NR	3NR	*1R	1R
—	3NR	1R	2R	3R	1NR	2NR	3NR	3NR
end of list	2NR	3NR	1R	2R	3R	1NR	2NR	*2R

second chance example (1)

	A	B	C	D	—	—	—	B
1	A						D	
2		B						
3			C			C		
page list								
last added	*1R	*2R	*3R	1NR	2NR	3NR	*1R	1R
—	3NR	1R	2R	3R	1NR	2NR	3NR	3NR
end of list	2NR	3NR	1R	2R	3R	1NR	2NR	*2R

second chance example (1)

page 1 was at bottom of list
reference — give second chance
moves to top of list
clear referenced bit

							—	B
1	A						D	
2		B						
3			C			C		
page list								
last added	*1R	*2R	*3R	1NR	2NR	3NR	*1R	1R
—	3NR	1R	2R	3R	1NR	2NR	3NR	3NR
end of list	2NR	3NR	1R	2R	3R	1NR	2NR	*2R

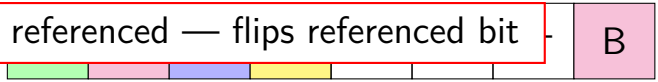
second chance example (1)

eventually page 1 gets to bottom of list again but now not referenced — use

1	A						D	
2		B						
3			C			C		
page list								
last added	*1R	*2R	*3R	1NR	2NR	3NR	*1R	1R
—	3NR	1R	2R	3R	1NR	2NR	3NR	3NR
end of list	2NR	3NR	1R	2R	3R	1NR	2NR	*2R

second chance example (1)

B referenced — flips referenced bit



1	A						D	
2		B						
3			C			C		
page list								
last added	*1R	*2R	*3R	1NR	2NR	3NR	*1R	1R
—	3NR	1R	2R	3R	1NR	2NR	3NR	3NR
end of list	2NR	3NR	1R	2R	3R	1NR	2NR	*2R

second chance example: exercise (1)

	A	B	C	D	—	—	—	B	A
1	A						D		
2		B							
3			C			C			
page list									
last added	*1R	*2R	*3R	1NR	2NR	3NR	*1R	1R	
—	3NR	1R	2R	3R	1NR	2NR	3NR	3NR	
end of list	2NR	3NR	1R	2R	3R	1NR	2NR	*2R	

exercise: What does this access to A replace? (D, B, or C?)
what is at end of list after? (PP 1, 2, or 3?)

second chance example: exercise (2)

	A	B	C	D	—	—	—	B	A	—	C
1	A						D				?
2		B									?
3			C			C				A	?
page list											
last added	*1R	*2R	*3R	1NR	2NR	3NR	*1R	1R	2NR	*3R	
—	3NR	1R	2R	3R	1NR	2NR	3NR	3NR	1R	2NR	
end of list	2NR	3NR	1R	2R	3R	1NR	2NR	*2R	3NR	1R	

second chance example: exercise (2)

	A	B	C	D	—	—	—	B	A	—	C
1	A						D				?
2		B									?
3			C			C				A	?
page list											
last added	*1R	*2R	*3R	1NR	2NR	3NR	*1R	1R	2NR	*3R	
—	3NR	1R	2R	3R	1NR	2NR	3NR	3NR	1R	2NR	
end of list	2NR	3NR	1R	2R	3R	1NR	2NR	*2R	3NR	1R	

exercise: What does this access to C replace? (D, B, or A?)
what is at end of list after? (PP 1, 2, or 3?)

second chance example (2)

	A	B	C	D	—	—	—	B	A	—	C	—
1	A						D					
2		B										C
3			C			C				A		
page list												
last added	*1R	*2R	*3R	1NR	2NR	3NR	*1R	1R	2NR	*3R	1NR	*2R
—	3NR	1R	2R	3R	1NR	2NR	3NR	3NR	1R	2NR	3R	1NR
end of list	2NR	3NR	1R	2R	3R	1NR	2NR	*2R	3NR	1R	2NR	3R

second chance cons

performs poorly with big memories...

may need to scan through lots of pages to find unaccessed

likely to count accesses from a long time ago

want some variation to tune its sensitivity

second chance cons

performs poorly with big memories...

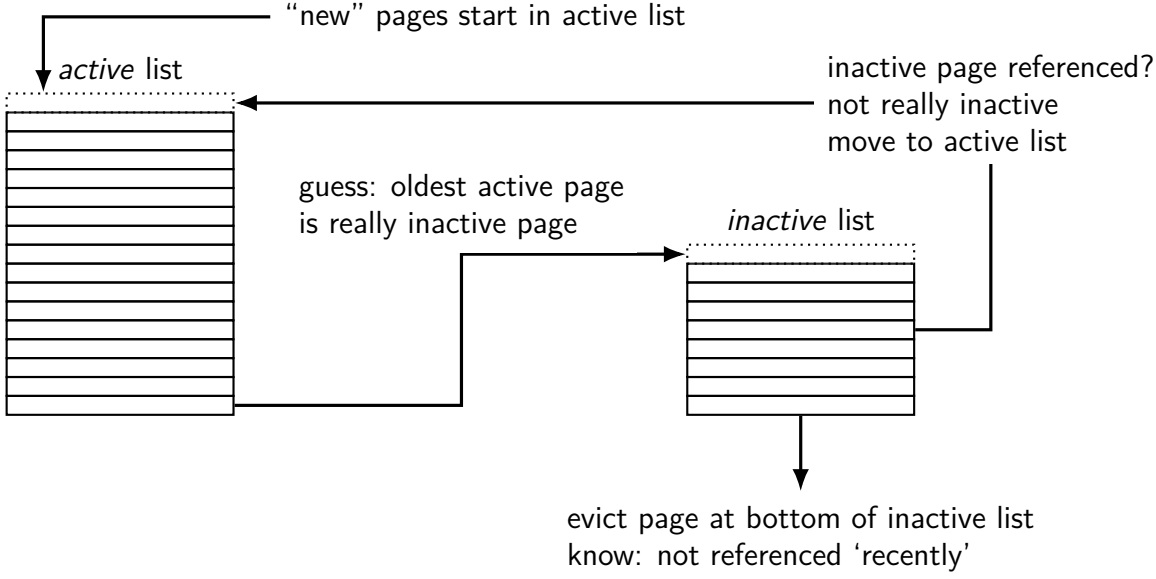
may need to scan through lots of pages to find unaccessed

likely to count accesses from a long time ago

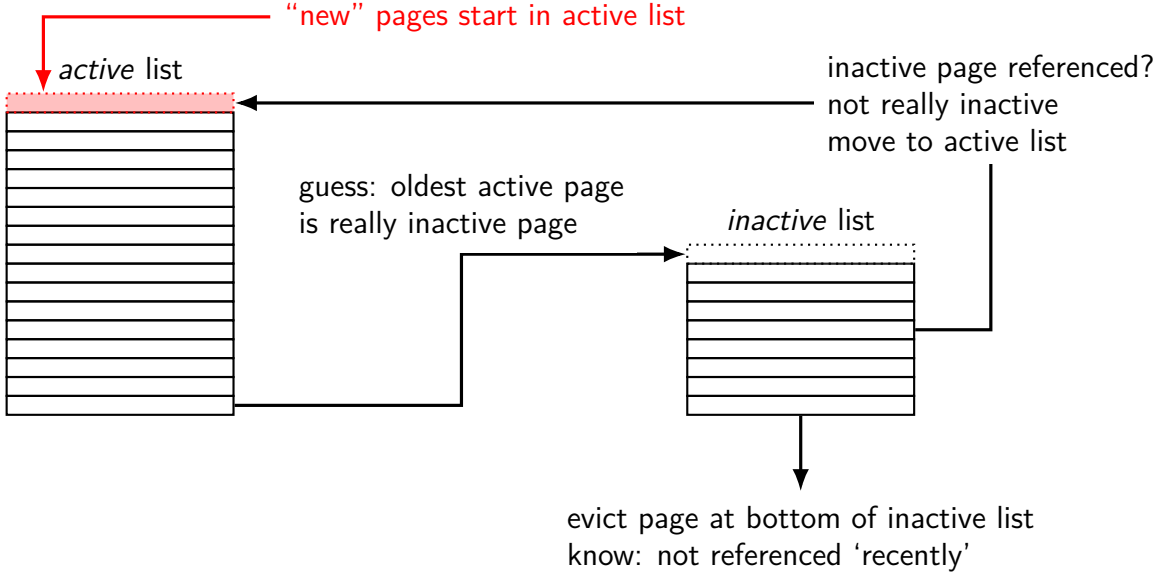
want some variation to tune its sensitivity

one idea: smaller list of pages to scan for accesses

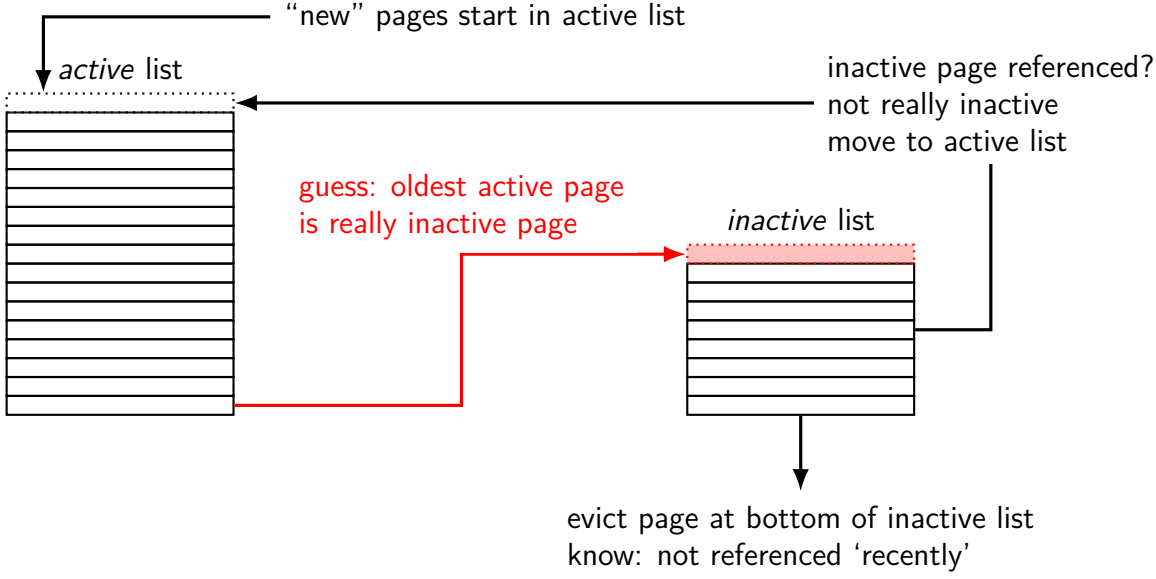
approximating LRU: SEQ



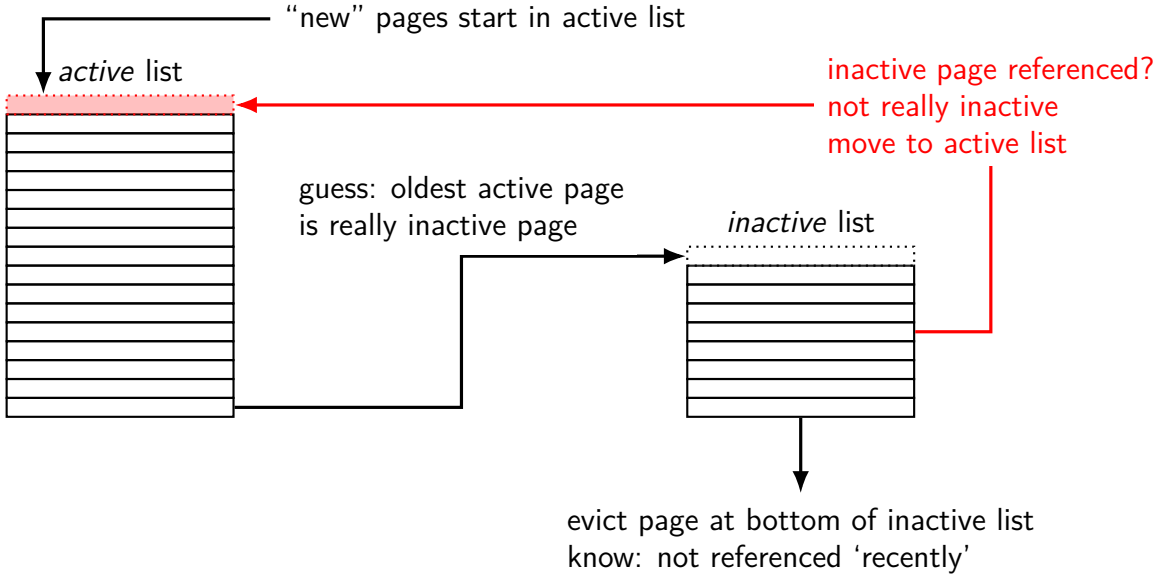
approximating LRU: SEQ



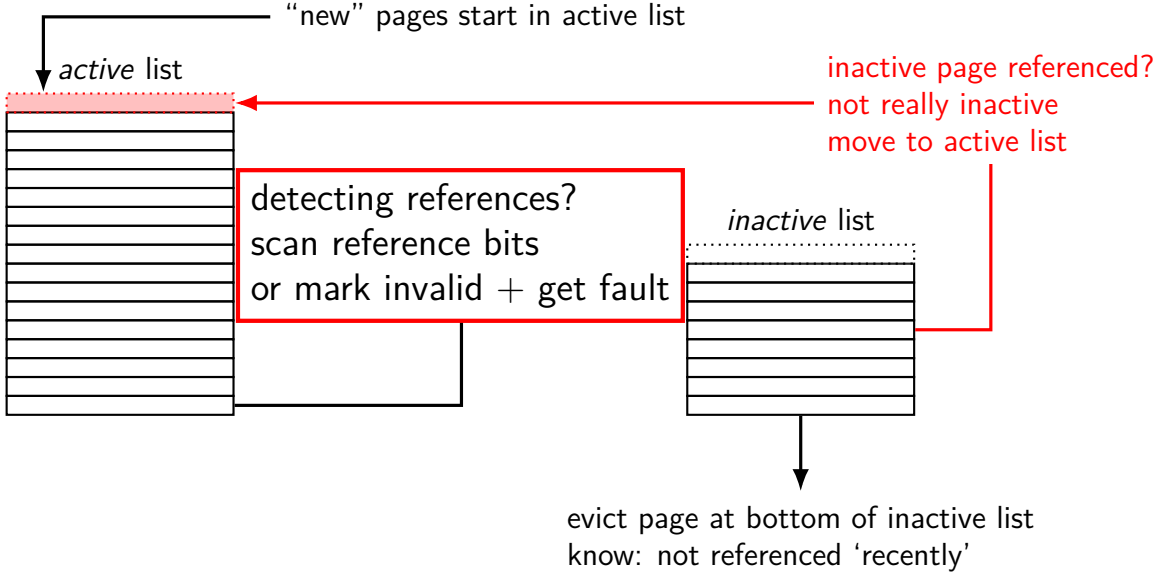
approximating LRU: SEQ



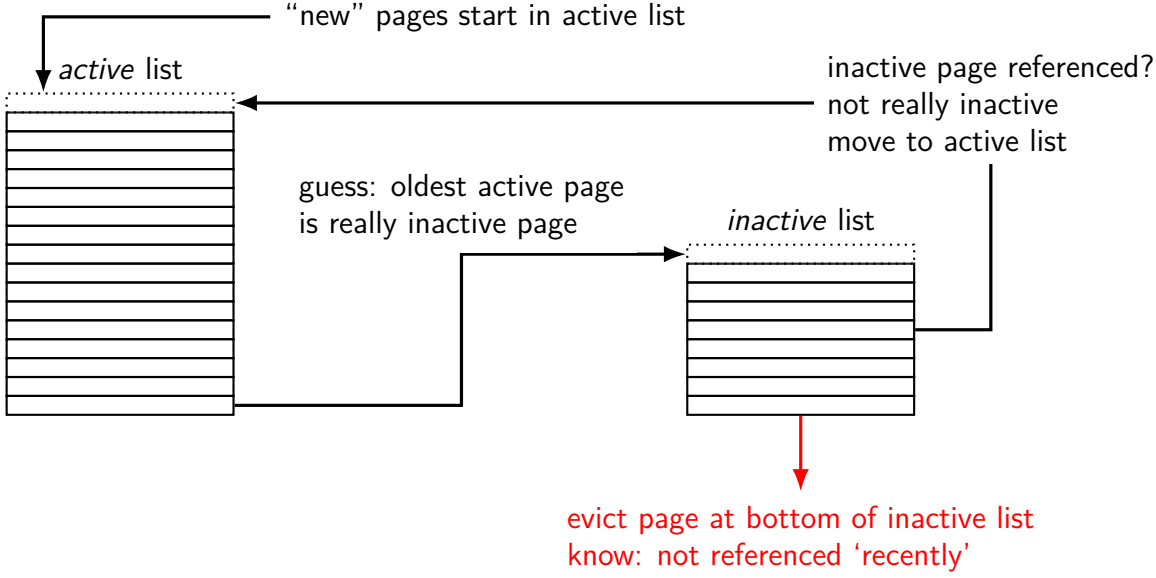
approximating LRU: SEQ



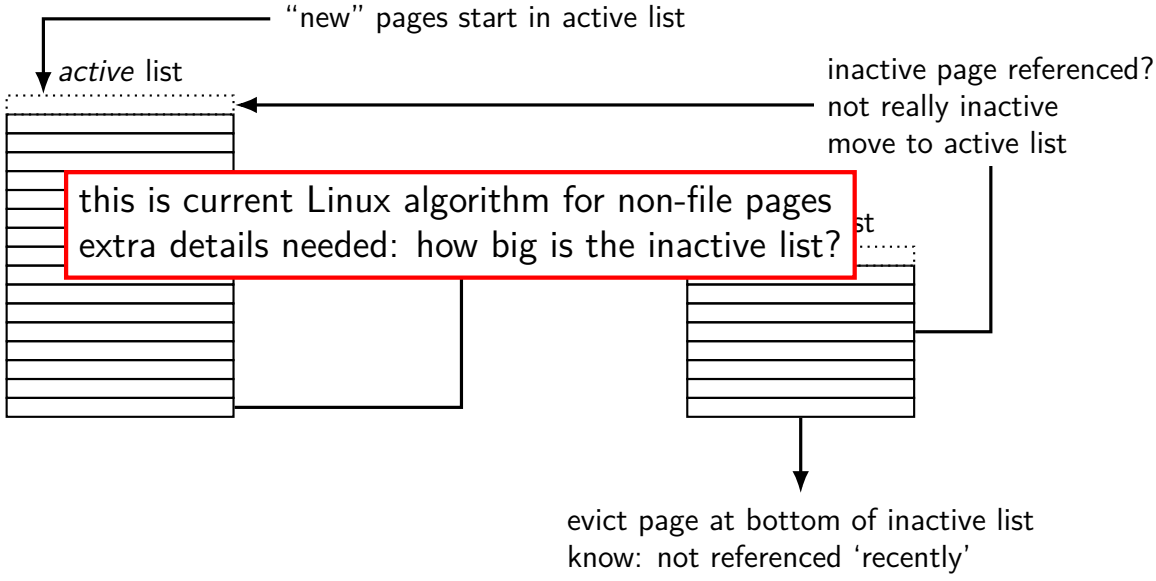
approximating LRU: SEQ



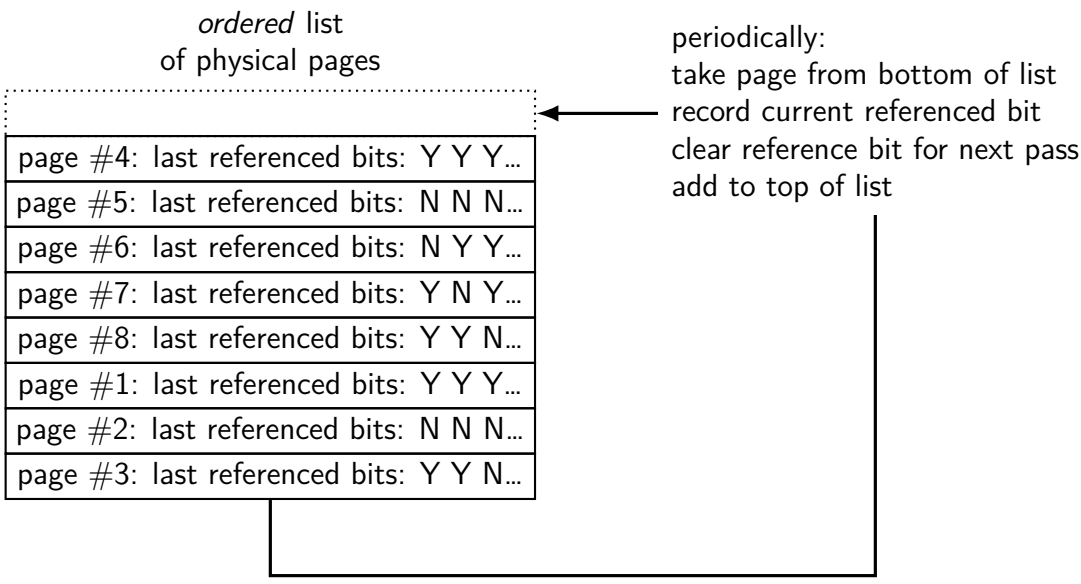
approximating LRU: SEQ



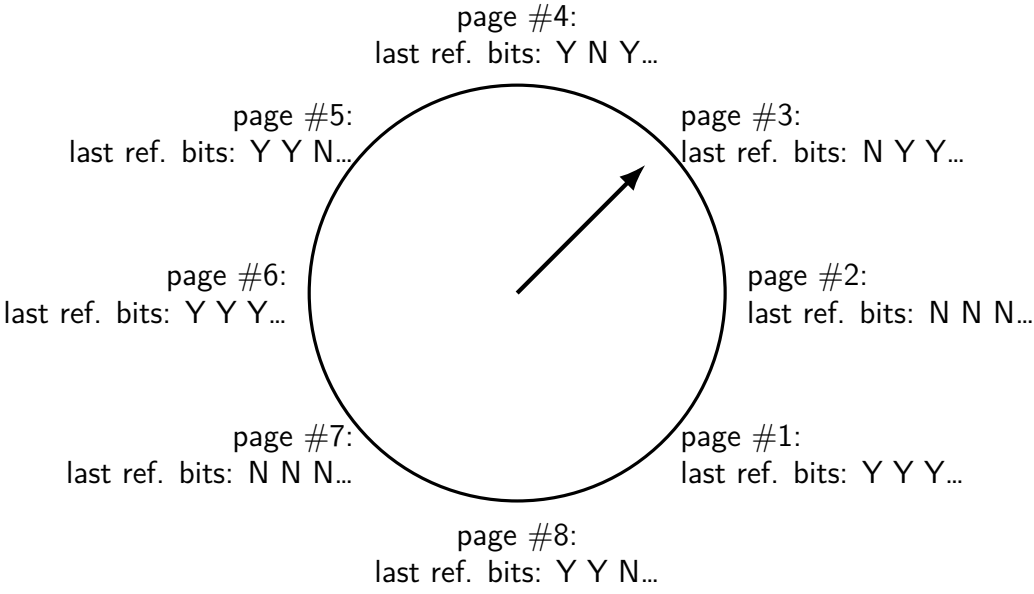
approximating LRU: SEQ



tracking usage: CLOCK (view 1)



tracking usage: CLOCK (view 2)



lazy replacement?

so far: don't do anything special until memory is full

only then is there a reason to writeback pages or evict pages

lazy replacement?

so far: don't do anything special until memory is full

only then is there a reason to writeback pages or evict pages

but real OSes are more proactive

non-lazy writeback

what happens when a computer loses power

how much data can you lose?

if we never run out of memory...all of it?

- no changed data written back

solution: track or scan for dirty pages and writeback

example goals:

- lose no more than 90 seconds of data
- force writeback at file close
- ...

non-lazy eviction

so far — allocating memory involves evicting pages

hopefully pages that haven't been used a long time anyways

non-lazy eviction

so far — allocating memory involves evicting pages

hopefully pages that haven't been used a long time anyways

alternative: evict earlier "in the background"

- "free": probably have some idle processor time anyways

allocation = remove already evicted page from linked list

- (instead of changing page tables, file cache info, etc.)

problems with LRU

question: when does LRU perform poorly?

problems with LRU

question: when does LRU perform poorly?

only reading things once

repeated scans of large amounts of data

problems with LRU

question: when does LRU perform poorly?

only reading things once

repeated scans of large amounts of data

both common access patterns for files

CLOCK-Pro: special casing for one-use pages

by default, Linux tries to handle scanning of files

- one read of file data — e.g. play a video, load file into memory

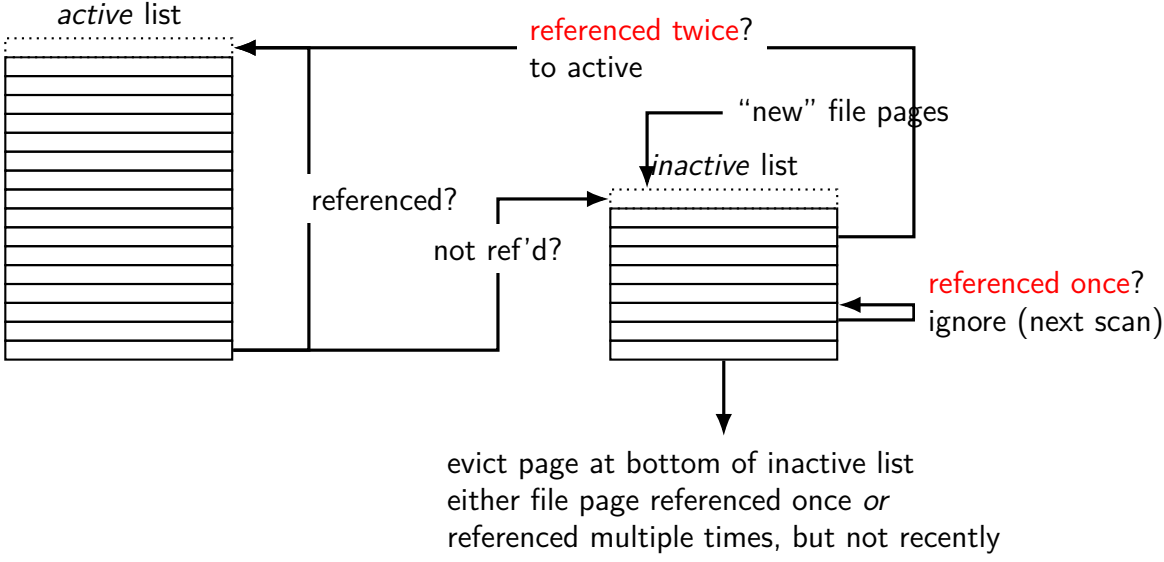
basic idea: don't consider pages active until the second access

single scans of file won't "pollute" cache

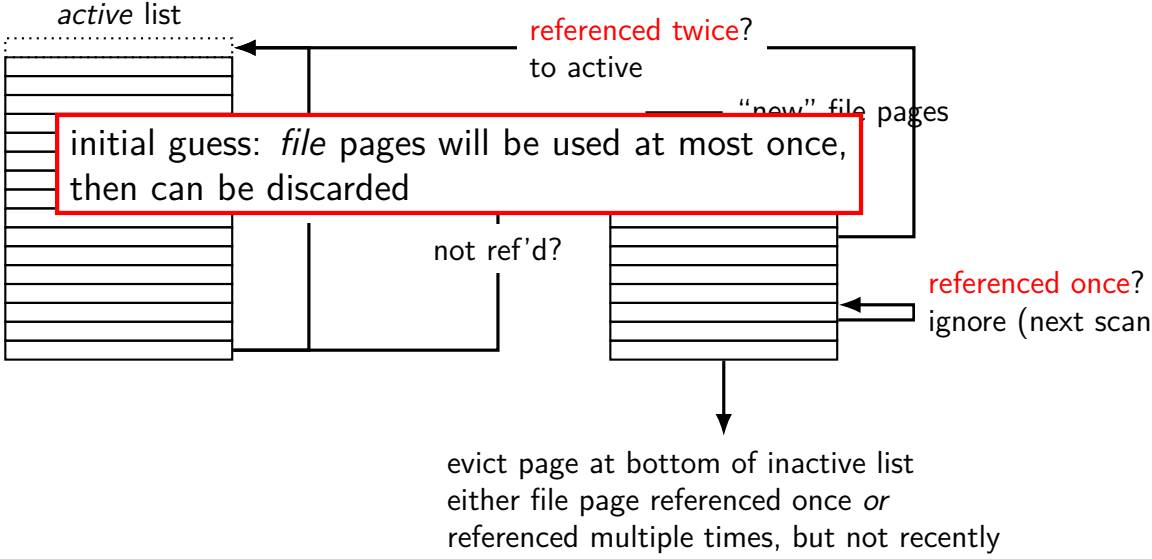
without this change: reading large files slows down other programs

- recently read part of large file steals space from active programs

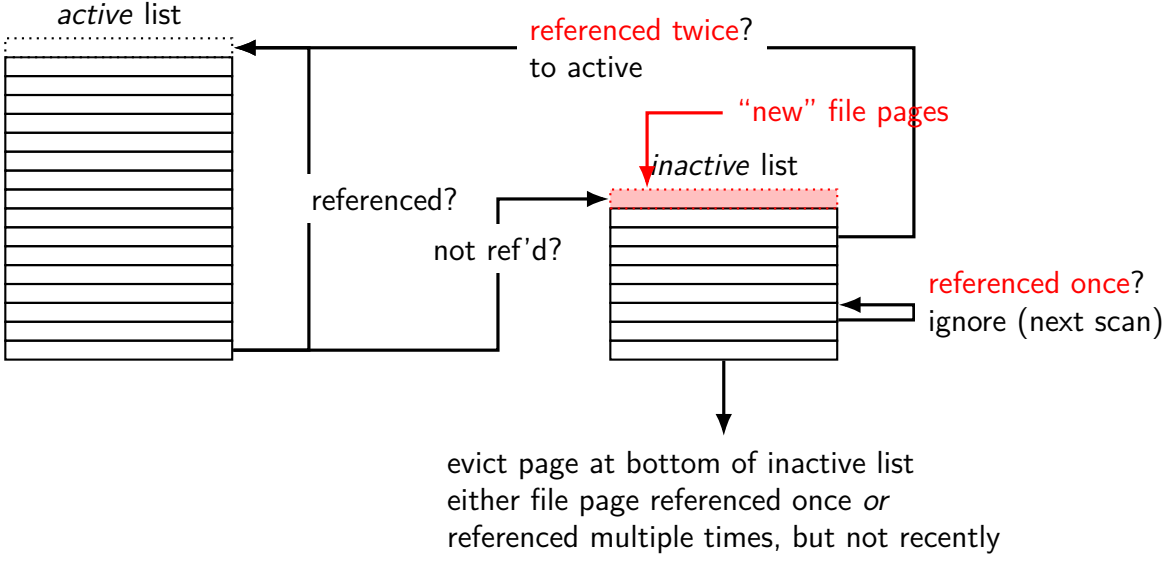
CLOCK-Pro: special casing for one-use pages



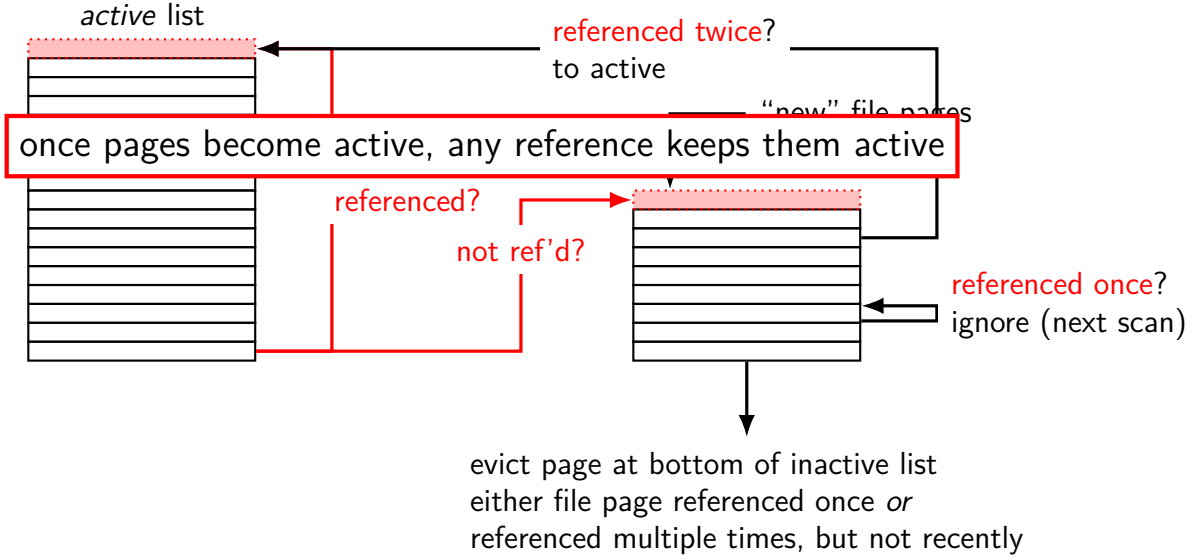
CLOCK-Pro: special casing for one-use pages



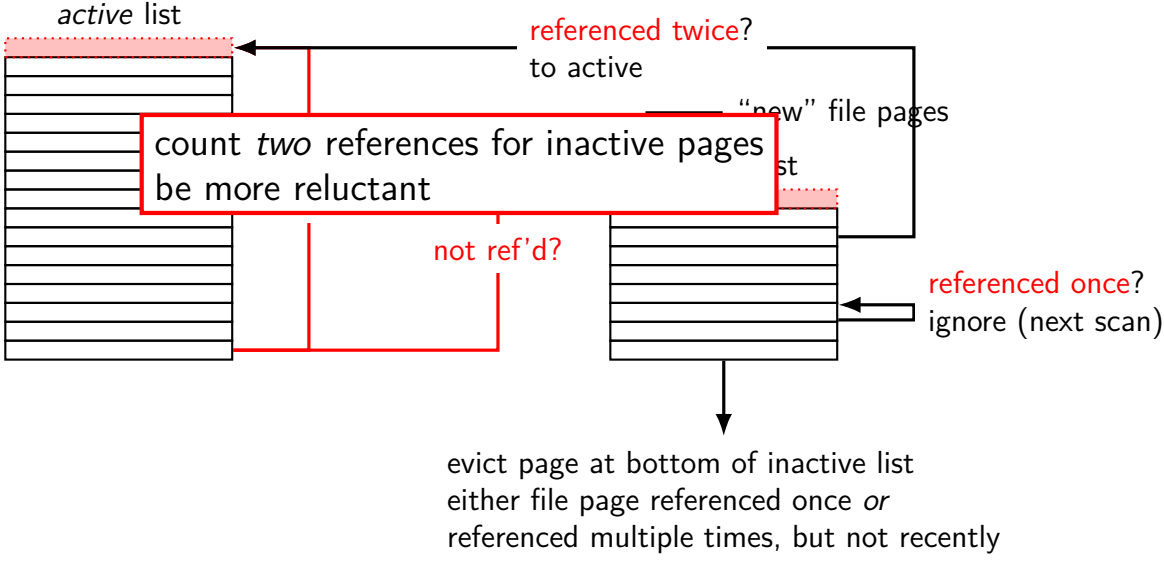
CLOCK-Pro: special casing for one-use pages



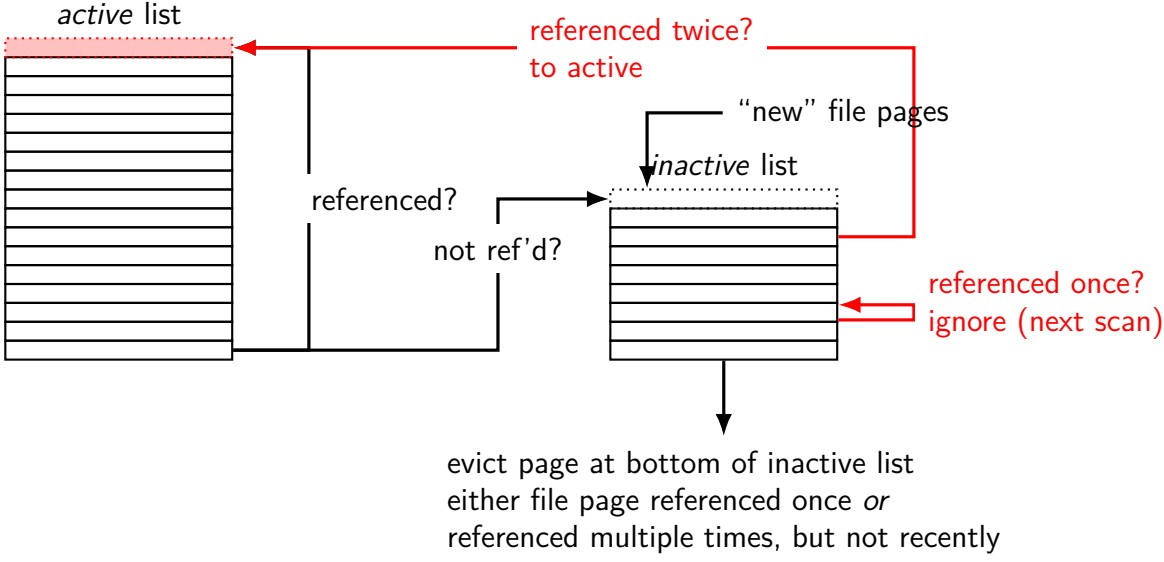
CLOCK-Pro: special casing for one-use pages



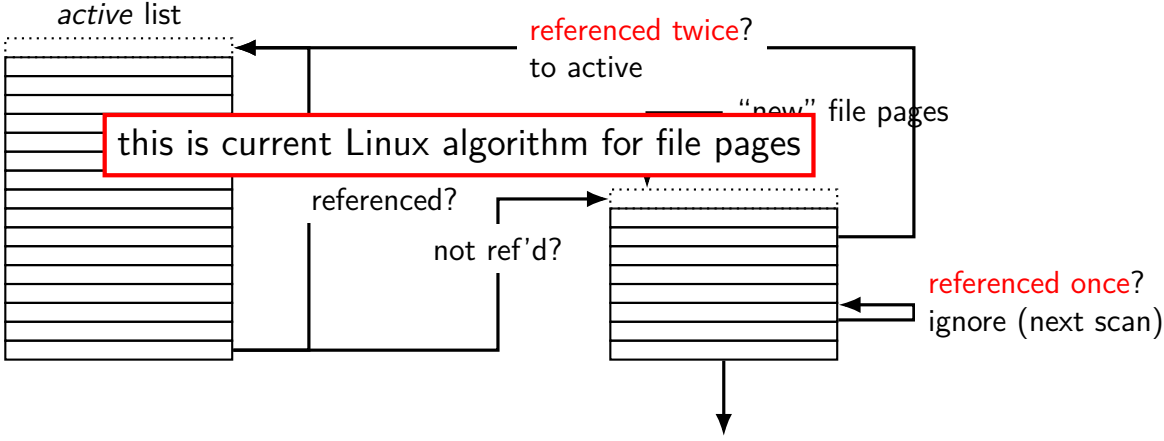
CLOCK-Pro: special casing for one-use pages



CLOCK-Pro: special casing for one-use pages



CLOCK-Pro: special casing for one-use pages



default Linux page replacement summary

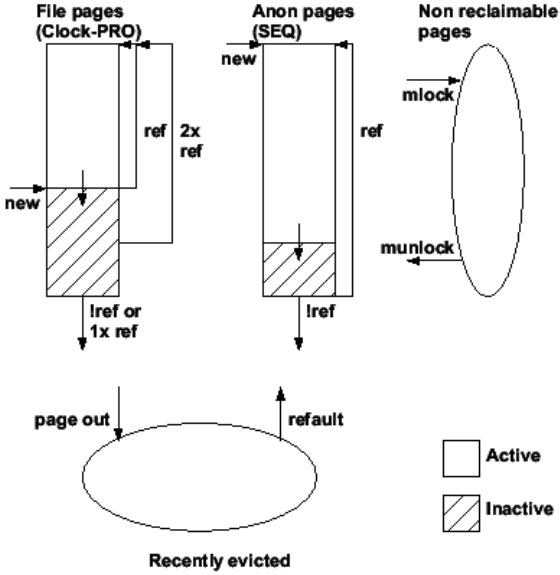


Figure: https://linux-mm.org/PageReplacementDesign

default Linux page replacement summary

identify *inactive* pages — guess: not going to be accessed soon

- file pages which haven't been accessed more than once, or
- any pages which haven't been accessed recently

some minimum threshold of inactive pages

- add to inactive list in background
- detecting references — scan referenced bits
- (I thought Linux marked as invalid — but wrong: not on x86)
- detect enough references — move to active

oldest inactive page still not used $\rightarrow$ evict that one

- otherwise: give it a second chance

being proactive

previous assumption: load on demand

why is something loaded?

- page fault
- maybe because application starts

can we do better?

readahead

program accesses page 4 of a file, page 5, page 6. What's next?

readahead

program accesses page 4 of a file, page 5, page 6. What's next?

page 7 — idea: guess this

on page fault, does it look like contiguous accesses?

called readahead

readahead heuristics

exercise: devise an algorithm to detect to do readahead.

- how to detect the reading pattern?
- when to start reads?
- how much to readahead?
- what state to keep?

Linux readahead heuristics — how much

how much to readahead?

Linux heuristic: count number of cached pages from before

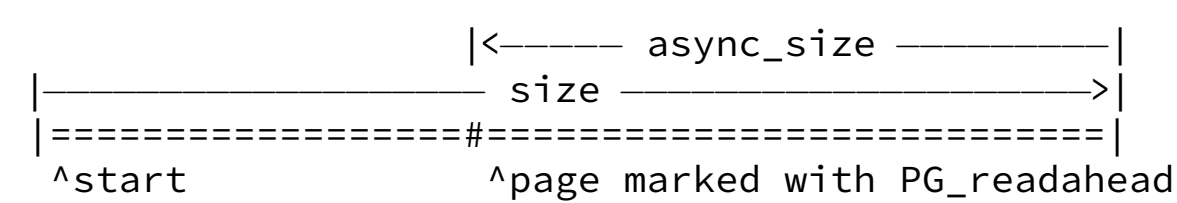
guess we should read about that many more
(plus minimum/maximum to avoid extremes)

goal: readahead more when applications are using file more

goal: don't readahead as much with low memory

Linux readahead heuristics — when

track “readahead windows” — pages read because of guess:



when `async_size` pages left, read next chunk

marked page = detect reads to this page

- one option: make page temporary invalid

idea: keep up with application, but not too far ahead

ASCII art figure: comments of Linux readahead code

thrashing

what if there's just not enough space?

- for program data, files currently being accessed

always reading things from disk

causes performance collapse — disk is really slow

known as thrashing

‘fair’ page replacement

so far: page replacement about least recently used

what about sharing fairly between users?

sharing fairly?

process A

4MB of stack+code, 16MB of heap
shared cached 24MB file X

process B

4MB of stack+code, 16MB of heap
shared cached 24MB file X

process C

4MB of stack+code, 4MB of heap
cached 32MB file Y

process D+E

4MB of stack+code (each), 70MB of heap (each)
but all heap + most of code is shared copy-on-write

accounting pages

shared pages make it difficult to count memory usage

Linux *cgroups* accounting (mostly): last touch

- count shared file pages for the process that last 'used' them
- ...as detected by page fault for page

Linux cgroup limits

Linux “control groups” of processes

can set memory limits for group of proceses:

low limit: don’t ‘steal’ pages when group uses less than this

- always take pages someone is using (unless no choice)

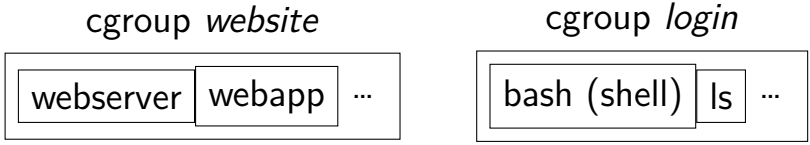
high limit: never let group use more than this

- replace pages from this group before anything else

...

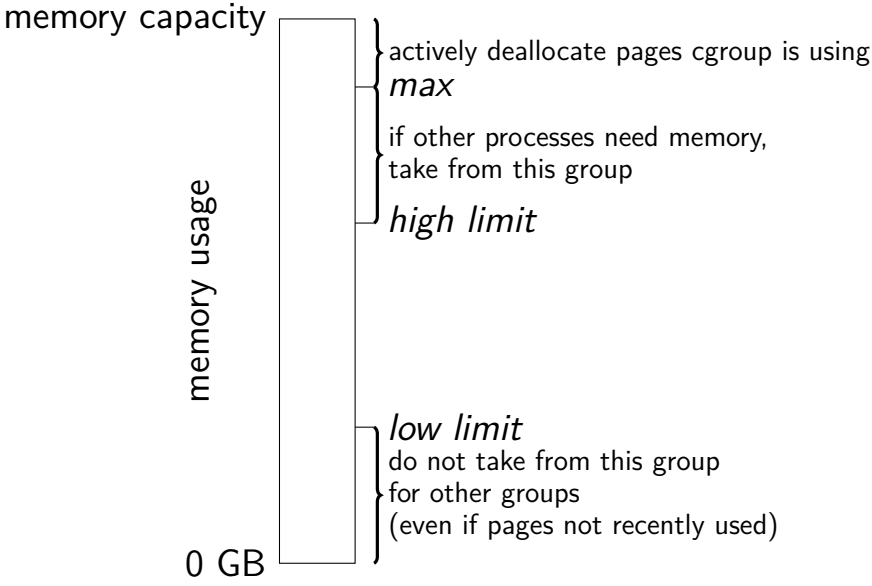
Linux cgroups

Linux mechanism: seperate processes into groups:



can set memory and CPU and ...shares for each group

Linux cgroup memory limits



page cache/replacement summary

program memory + files — swapped to disk, cached in memory

mostly, assume working set model

- keep (hopefully) small active set in memory
- least recently used variants

special cases for non-LRU-friendly patterns (e.g. scans)

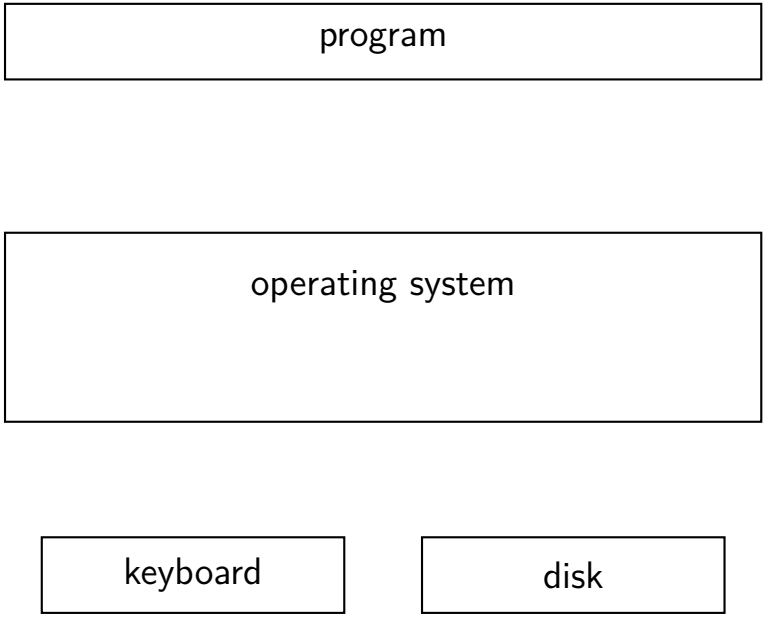
- maybe more we haven't discussed?

being proactive (writeback when idle, readahead, pool of pre-evicted pages)

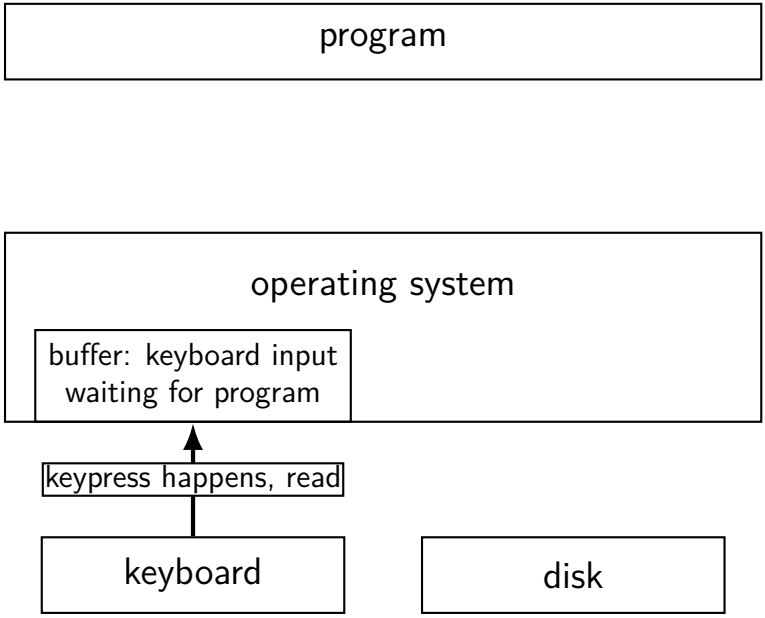
handling non-miss-rate goals

- fair replacement: limit active memory per user?
- probably more we haven't discussed here? optimizing throughput? fair throughput between users?

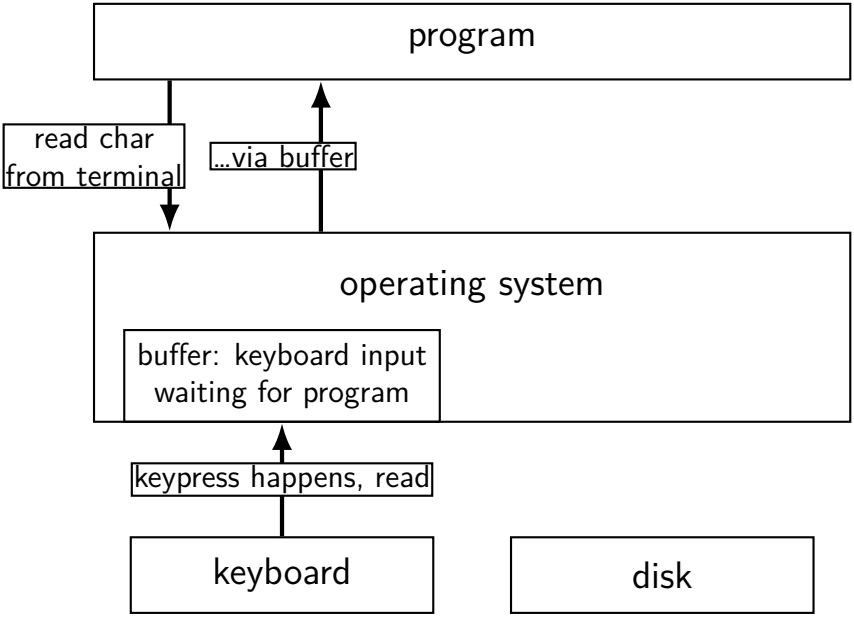
recall: kernel buffering (reads)



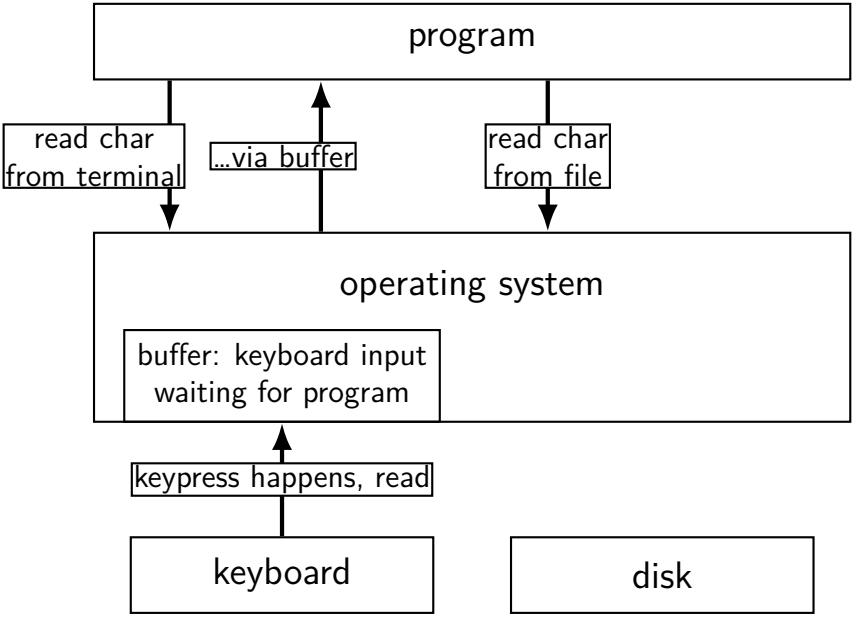
recall: kernel buffering (reads)



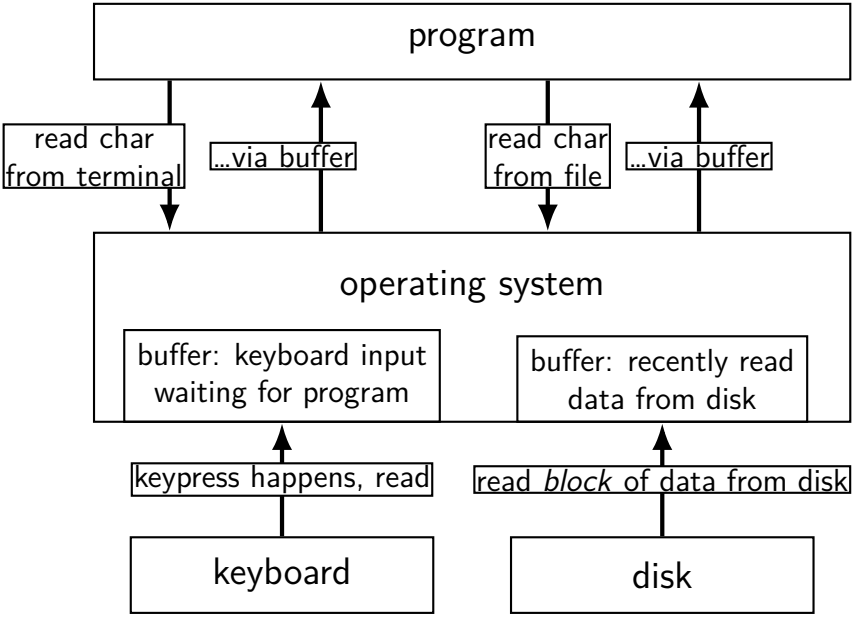
recall: kernel buffering (reads)



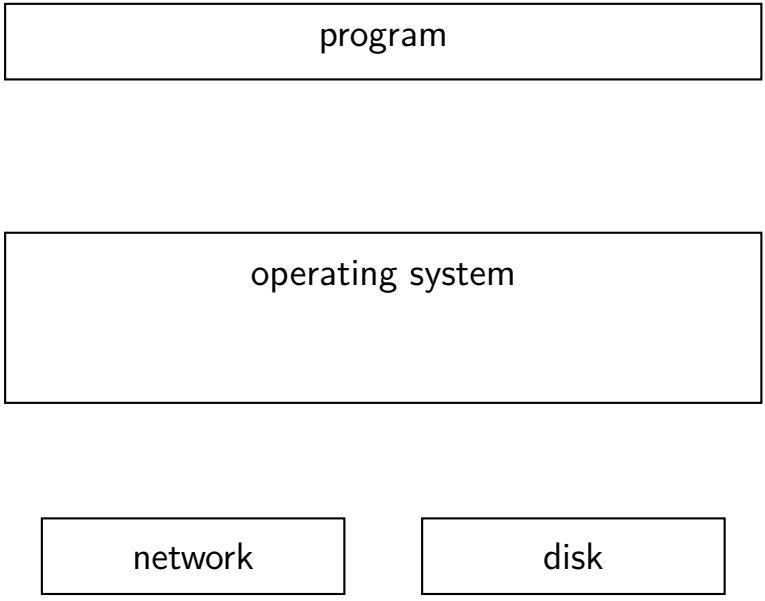
recall: kernel buffering (reads)



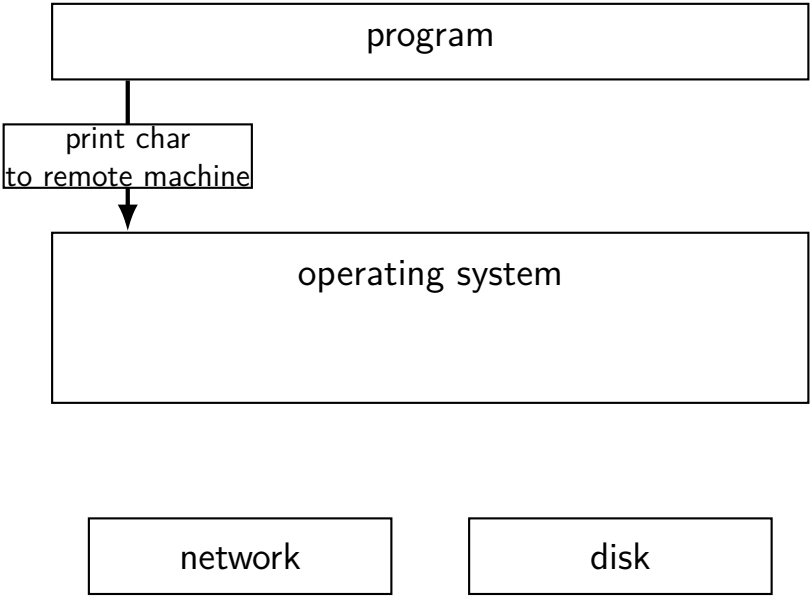
recall: kernel buffering (reads)



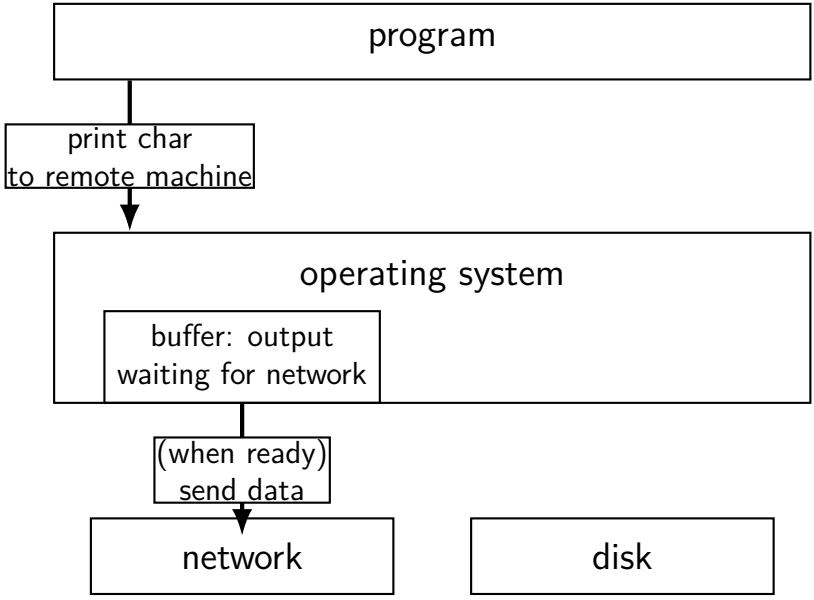
recall: kernel buffering (writes)



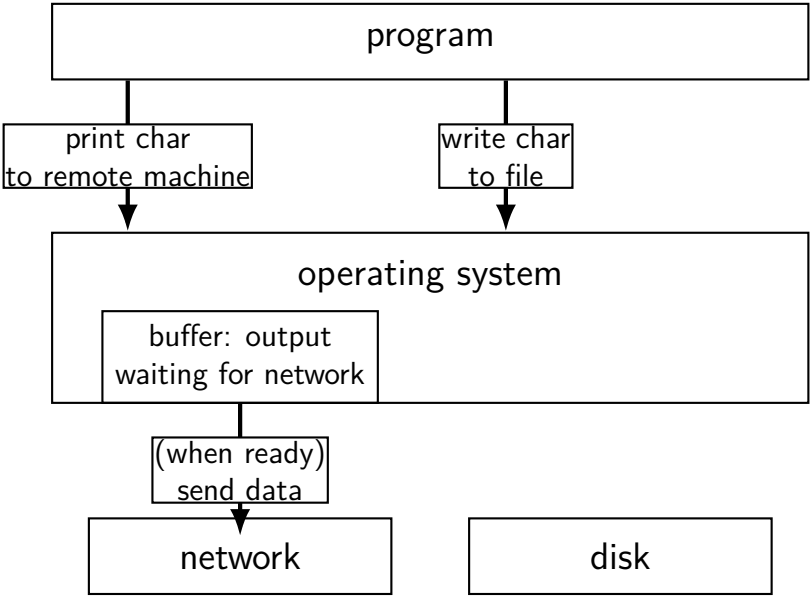
recall: kernel buffering (writes)



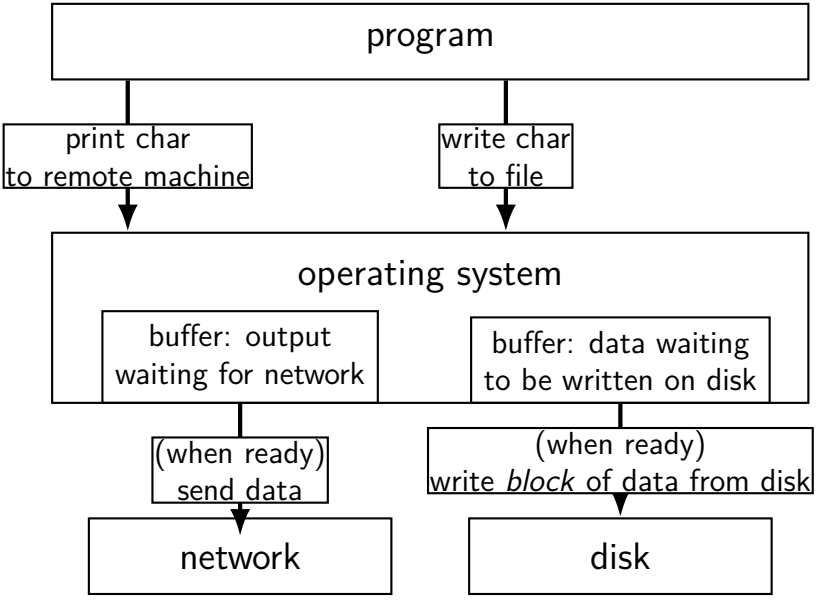
recall: kernel buffering (writes)



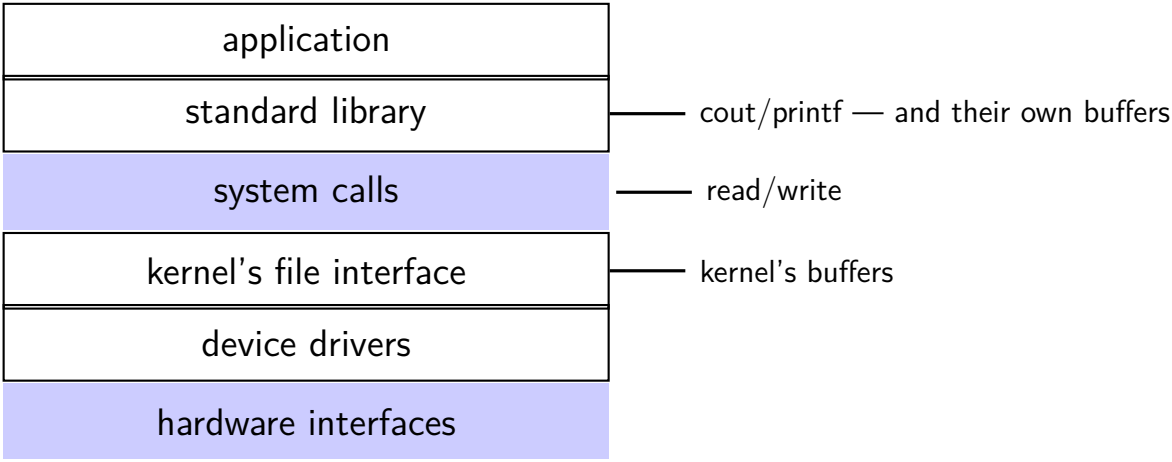
recall: kernel buffering (writes)



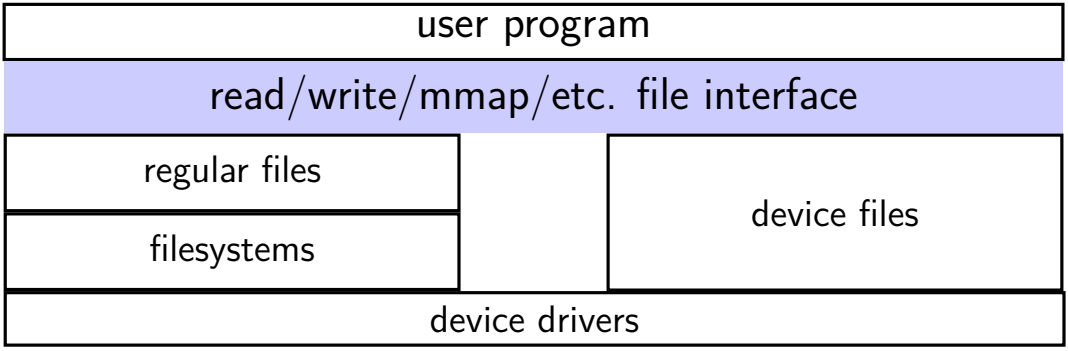
recall: kernel buffering (writes)



recall: layering



ways to talk to I/O devices



devices as files

talking to device? open/read/write/close

typically similar interface within the kernel

device driver implements the file interface

example device files from a Linux desktop

`/dev/snd/pcmC0D0p` — audio playback

configure, then write audio data

`/dev/sda`, `/dev/sdb` — SATA-based SSD and hard drive

usually access via filesystem, but can mmap/read/write directly

`/dev/input/event3`, `/dev/input/event10` — mouse and keyboard

can read list of keypress/mouse movement/etc. events

`/dev/dri/renderD128` — builtin graphics

DRI = direct rendering infrastructure

devices: extra operations?

read/write/mmap not enough?

- audio output device — set format of audio? headphones plugged in?
- terminal — whether to echo back what user types?
- CD/DVD — open the disk tray? is a disk present?
- ...

extra POSIX file descriptor operations:

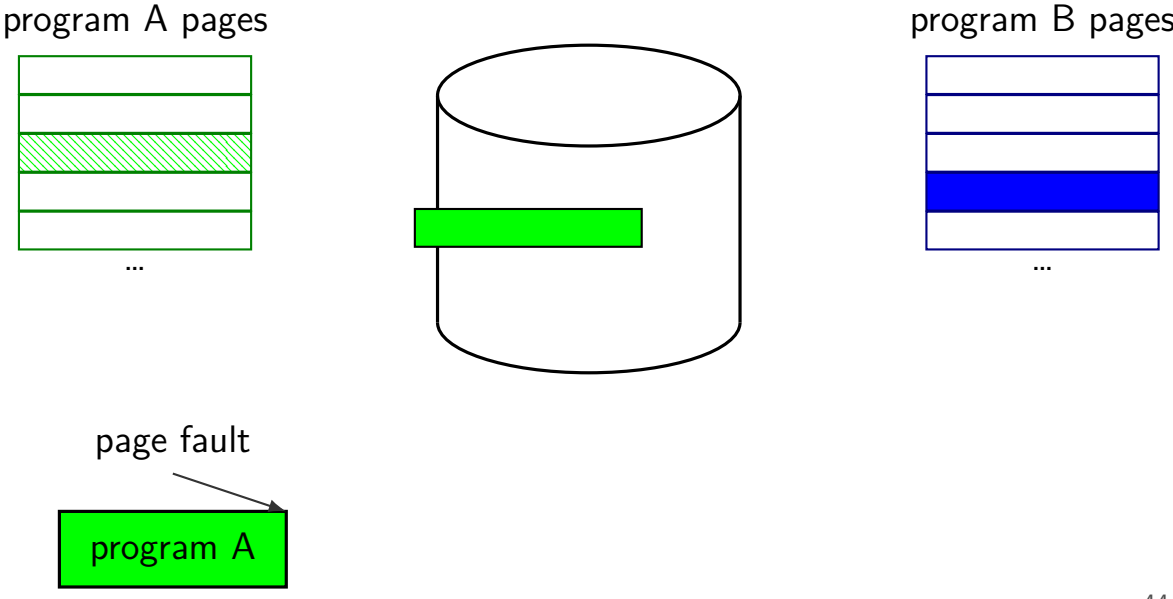
- ioctl (general I/O control) — device driver-specific interface
- tcsetattr (for terminal settings)
- fcntl
- ...

also possibly extra device files for same device:

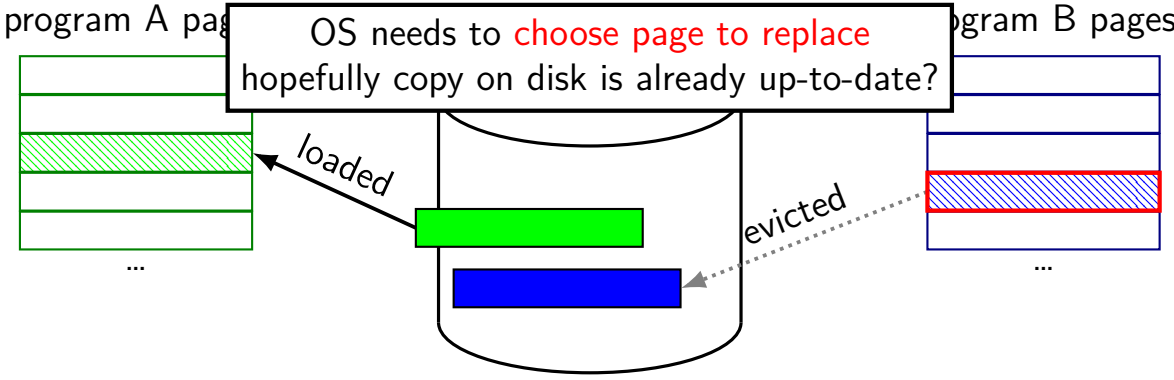
- `/dev/snd/controlC0` to configure audio settings for `/dev/snd/pcmC0D0p`, `/dev/snd/pcmC0D10p`, ...

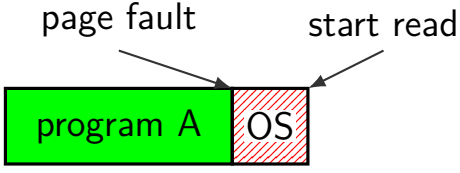
backup slides

swapping timeline

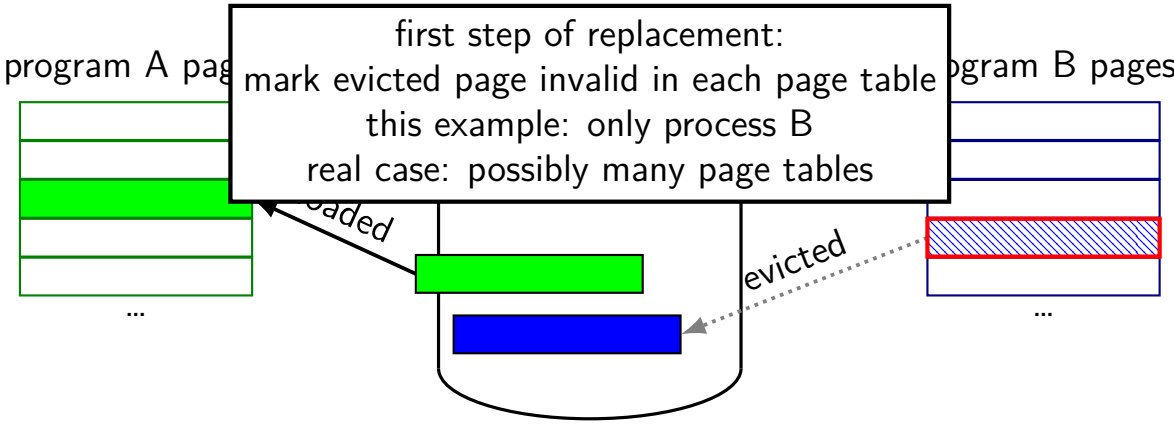


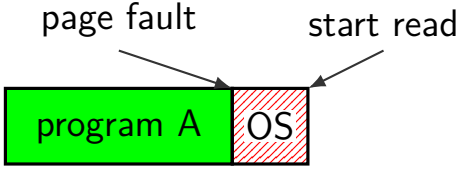
swapping timeline



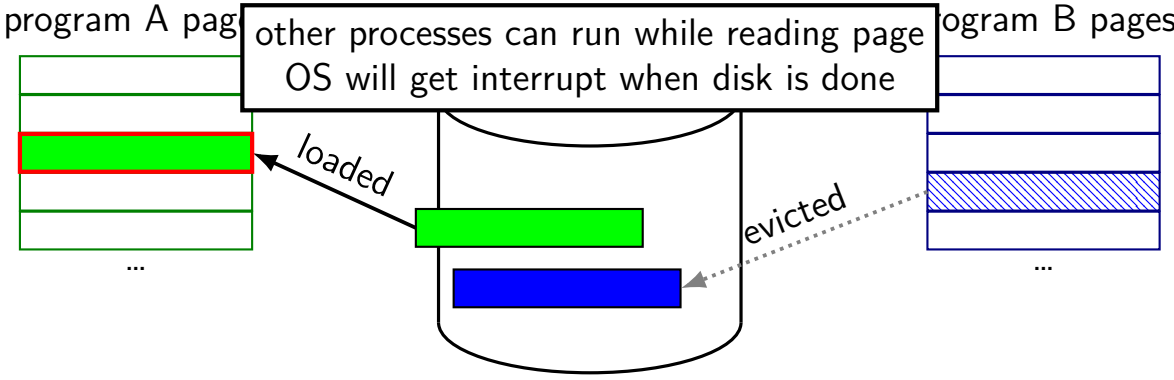


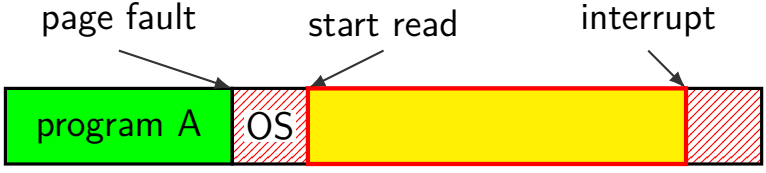
swapping timeline



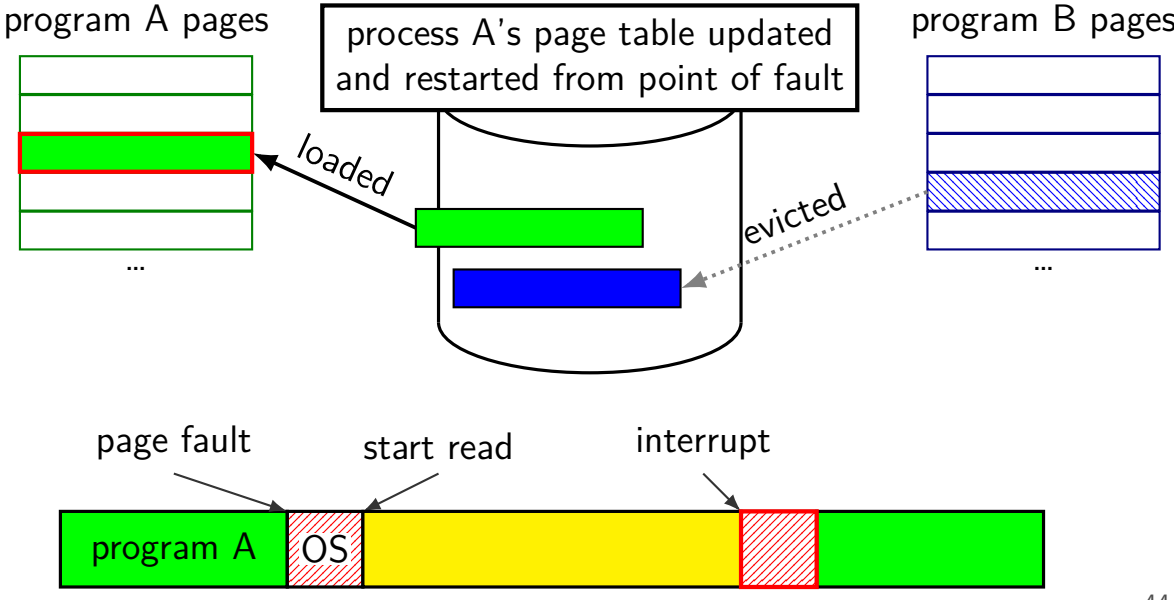


swapping timeline





swapping timeline



POSIX: everything is a file

the file: one interface for

- devices (terminals, printers, ...)
- regular files on disk
- networking (sockets)
- local interprocess communication (pipes, sockets)

basic operations: open(), read(), write(), close()

the file interface

open before use

- setup, access control happens here

byte-oriented

- real device isn't? operating system needs to hide that

explicit close

the file interface

open before use

- setup, access control happens here

byte-oriented

- real device isn't? operating system needs to hide that

explicit close