last time

mmap, shared:
- load from file on demand
- write out to file when freeing up space

mmap, private
- load from file on demand
- make copies on write

swapping/unbacked mappings
- make up location on disk to save data

page cache:
- virtual pages are really on disk (file, or temp location for swapping)
- physical pages “temporarily” cache copies
- challenge: cache managements

Belady’s MIN: minimum number of page replacements
- access furthest in the future

practically optimizing for hit-rate

recall?: locality assumption

temporal locality: things accessed now will be accessed again soon

(for now: not concerned about spatial locality)

more possible policies: least recently used or least frequently used

practically optimizing for hit-rate

recall?: locality assumption

temporal locality: things accessed now will be accessed again soon

(for now: not concerned about spatial locality)

more possible policies: least recently used or least frequently used

least recently used (the good case)

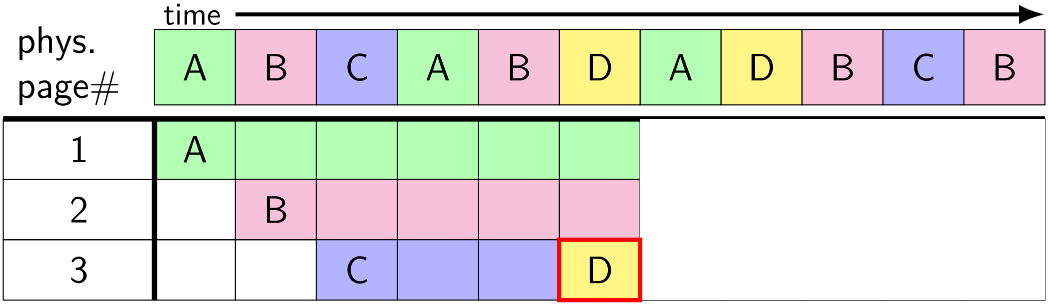
referenced (virtual) pages:

time →

phys. page#	A	B	C	A	B	D	A	D	B	C	B
1	A										
2		B									
3			C								

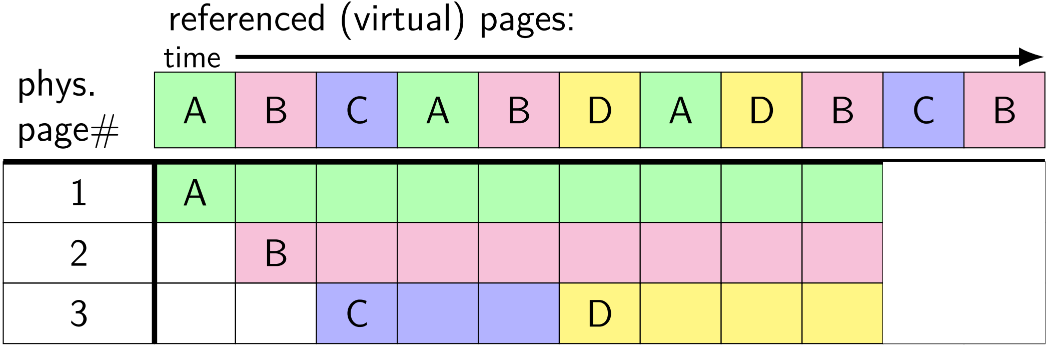
least recently used (the good case)

referenced (virtual) pages:

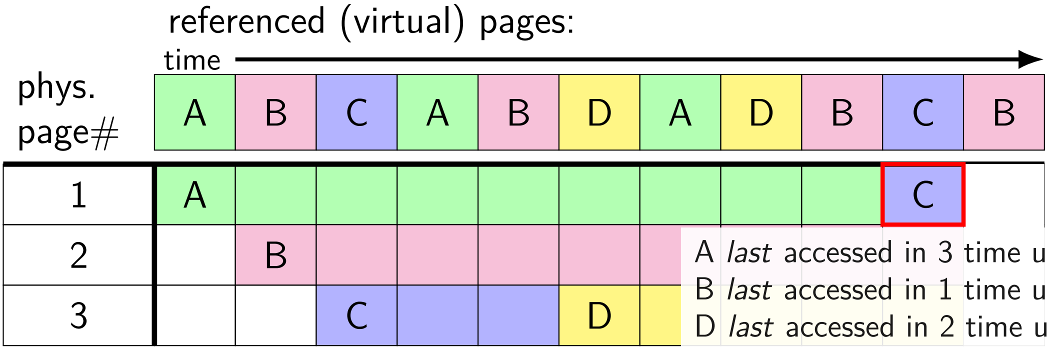


A *last* accessed 2 time units ago
B *last* accessed 1 time unit ago
C *last* accessed 3 time units ago
choose to replace C

least recently used (the good case)



least recently used (the good case)



A *last* accessed in 3 time units ago
B *last* accessed in 1 time unit ago
D *last* accessed in 2 time units ago
choose to replace A

least recently used (the good case)

referenced (virtual) pages:

time →

phys. page#	A	B	C	A	B	D	A	D	B	C	B
1	A									C	
2		B									
3			C			D					

least recently used (the worst case)

time →

phys. page#											
	A	B	C	D	A	B	C	D	A	B	C
1	A			D			C			B	
2		B			A			D			C
3			C			B			A		

least recently used (the worst case)

time →

phys. page#	A	B	C	D	A	B	C	D	A	B	C
1	A			D			C			B	
2		B			A			D			C
3			C			B			A		

8 replacements with LRU
versus 3 replacements with MIN:

phys. page#	A	B	C	D	A	B	C	D	A	B	C
1	A									B	
2		B					C				
3			C	D							

least recently used (exercise) [intro]

	A	B	A	D	C	B	D	B	C	D	A
1											
2											
3											

least recently used (exercise)

	A	B	A	D	C	B	D	B	C	D	A
1	A	A	A	A							
2		B	B	B							
3				D							

pure LRU implementation

implementing LRU in software

maintain doubly-linked list of all physical pages

whenever a page is accessed:

- remove page from linked list, then
- add page to head of list

whenever a page needs to replaced:

- remove a page from the tail of the linked list, then
- evict that page from all page tables (and anything else)
- and use that page for whatever needs to be loaded

pure LRU implementation

implementing LRU in software

maintain doubly-linked list of all physical pages

whenever a page is accessed:

- remove page from linked list, then
- add page

need to run code on every access
probably 100+x slowdown?

whenever a

- remove a page from the tail of the linked list, then
- evict that page from all page tables (and anything else)
- and use that page for whatever needs to be loaded

so, what's practical

probably won't implement LRU — too slow

what can we practically do?

tools for tracking accesses

approximating LRU = "was this accessed recently"?

don't need to detect all accesses, only one recent one

"was this accessed since we started looking a few seconds ago?"

tools for tracking accesses

approximating LRU = “was this accessed recently”?

don’t need to detect all accesses, only one recent one

- “was this accessed since we started looking a few seconds ago?”

ways to detect accesses AKA references:

- mark page invalid, if page fault happens make valid and record ‘accessed/referenced’
- ‘accessed’ or ‘referenced’ bit set by HW (on x86, but not everywhere)

tools for tracking accesses

approximating LRU = "was this accessed recently"?

don't need to detect all accesses, only one recent one

- "was this accessed since we started looking a few seconds ago?"

ways to detect accesses AKA references:

- mark page invalid, if page fault happens make valid and record 'accessed/referenced'
- 'accessed' or 'referenced' bit set by HW (on x86, but not everywhere)

tools for tracking accesses

approximating LRU = "was this accessed recently"?

don't need to detect all accesses, only one recent one

- "was this accessed since we started looking a few seconds ago?"

ways to detect accesses AKA references:

- mark page invalid, if page fault happens make valid and record 'accessed/referenced'
- 'accessed' or 'referenced' bit set by HW (on x86, but not everywhere)

tools for tracking accesses

approximating LRU = "was this accessed recently"?

don't need to detect all accesses, only one recent one

- "was this accessed since we started looking a few seconds ago?"

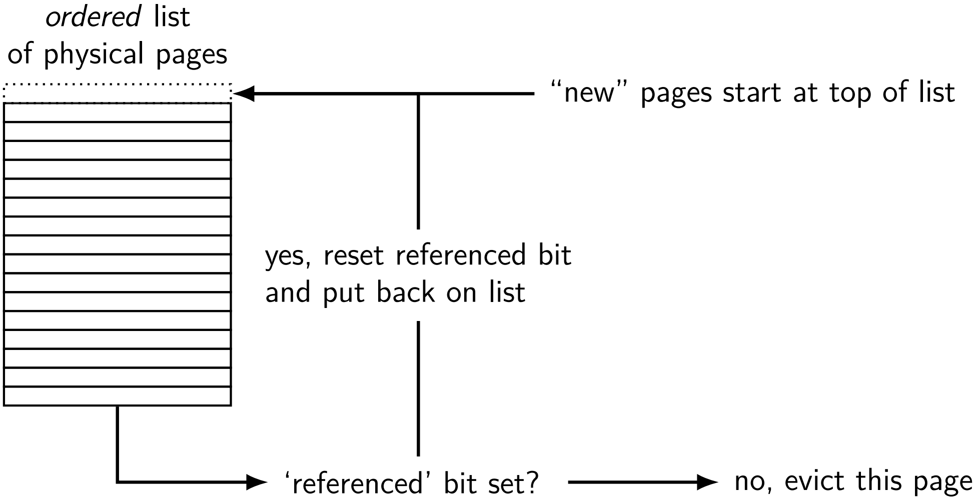
ways to detect accesses AKA references:

- mark page invalid, if page fault happens make valid and record 'accessed/referenced'
- 'accessed' or 'referenced' bit set by HW (on x86, but not everywhere)

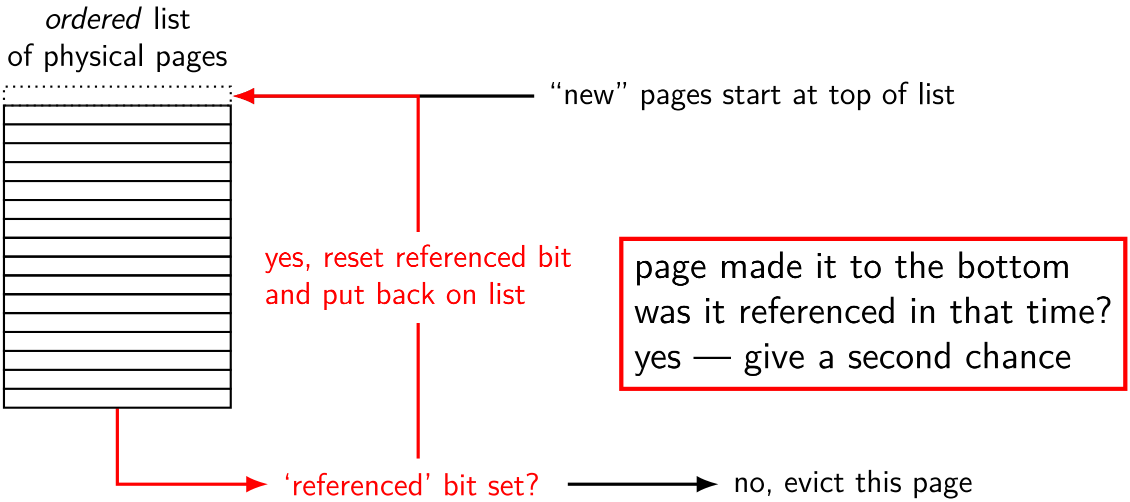
same idea applies for detecting writes

- to know whether replaced page needs to be saved to disk
- called "dirty" bit instead of accessed/referenced bit

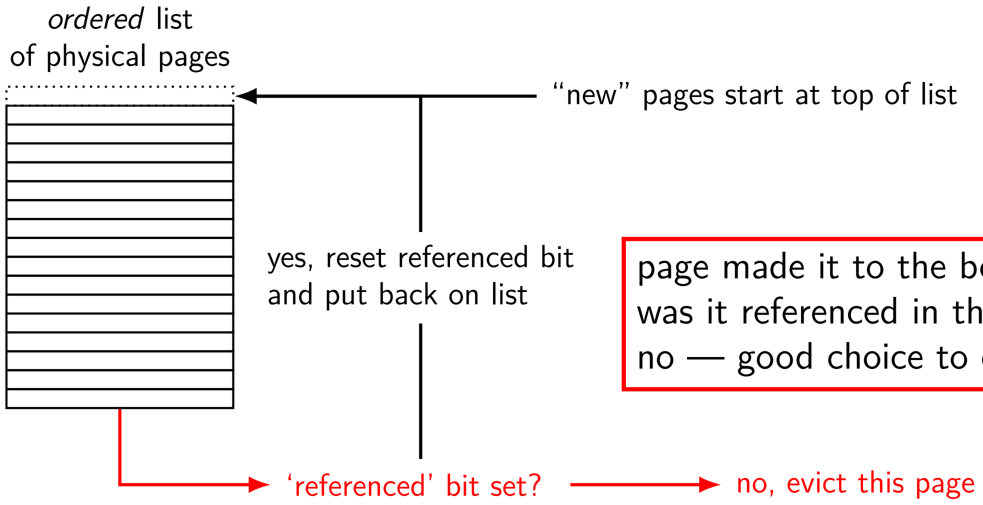
approximating LRU: second chance



approximating LRU: second chance



approximating LRU: second chance



page made it to the bottom
was it referenced in that time?
no — good choice to evict

second chance example (0)

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

second chance example (0)

	A		B			

place A in physical page 1
accessed right after → becomes referenced

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

second chance example (0)

place B in physical page 2
accessed right after $\rightarrow$ becomes referenced

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

second chance example (0)

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

future slides:
going to skip writing
these intermediate steps
(just for space)

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)

place A in page 1
not referenced on return from page fault handler
immediately referenced by program when page fault handler returns

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

								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: 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 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…

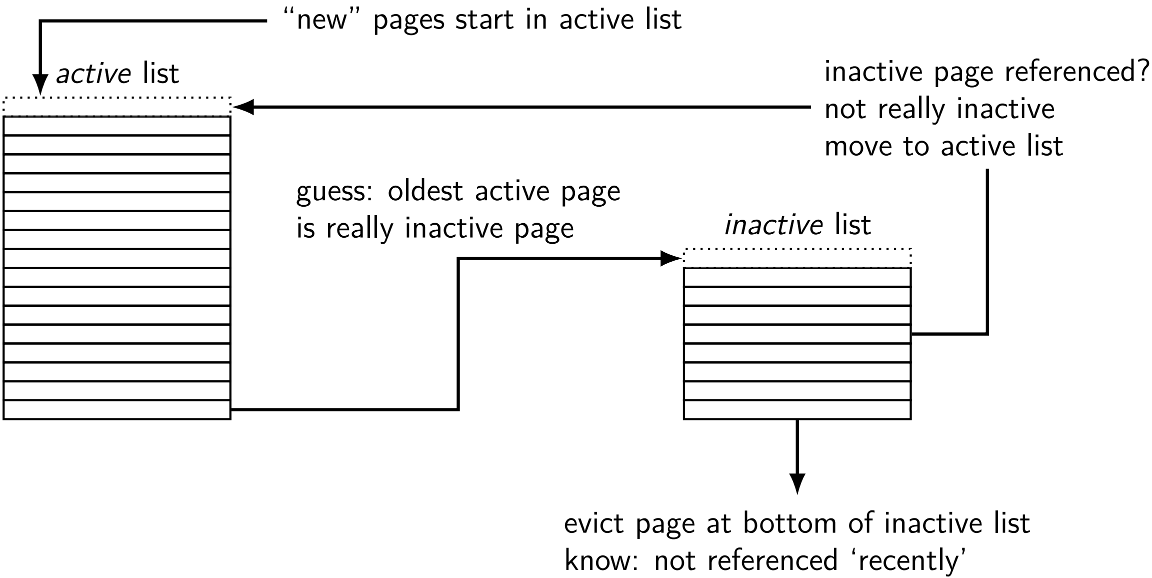
may need to scan through lots of pages to find unaccessed

likely to count accesses from a long time ago

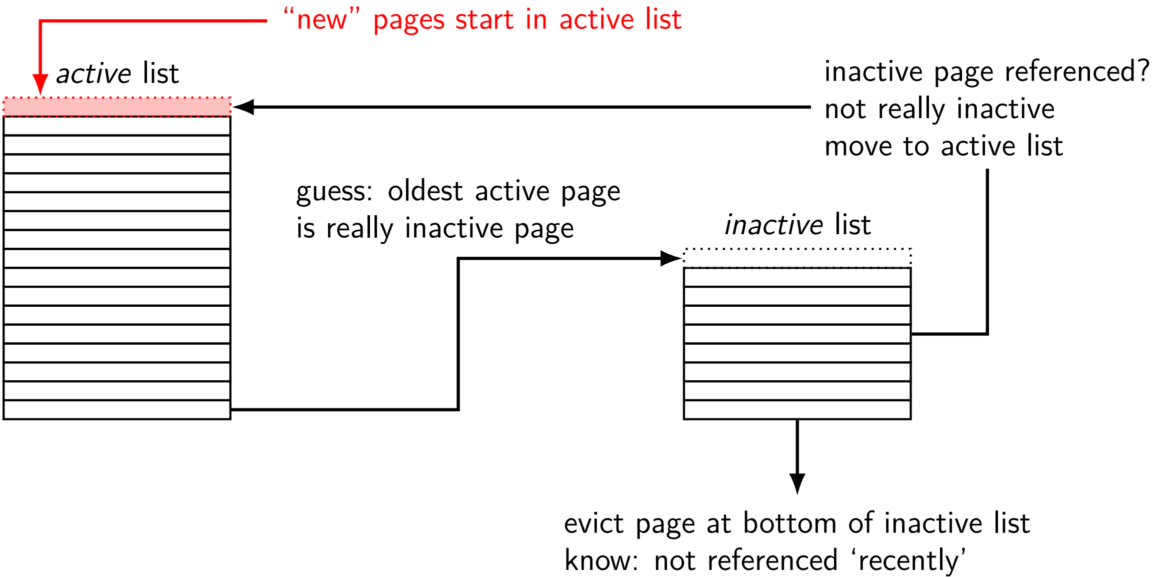
want some variation to tune its sensitivity

one idea: smaller list of pages to scan for accesses

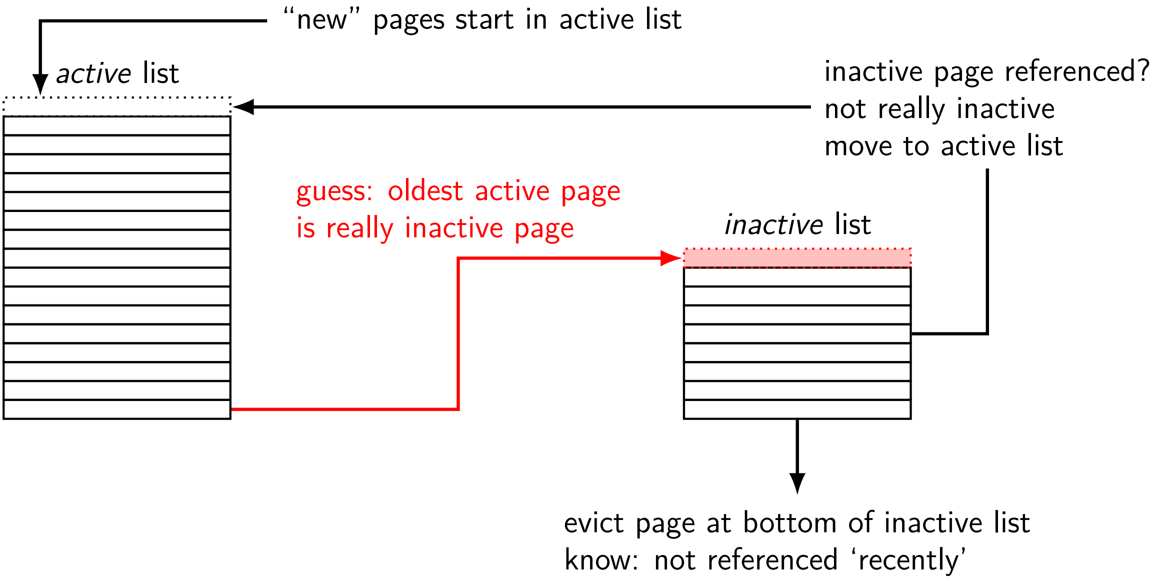
approximating LRU: SEQ



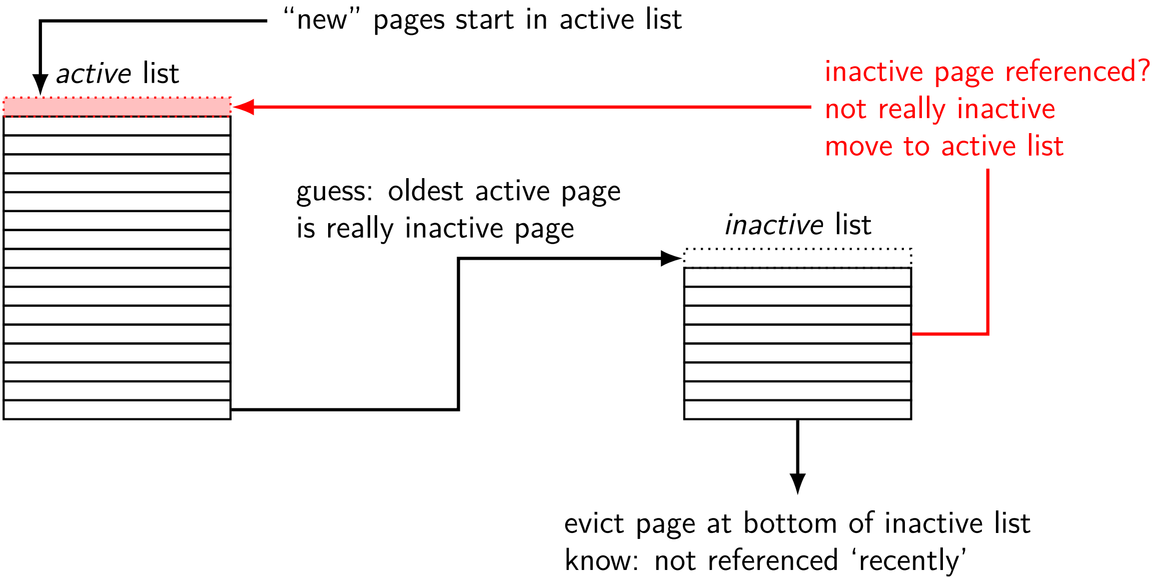
approximating LRU: SEQ



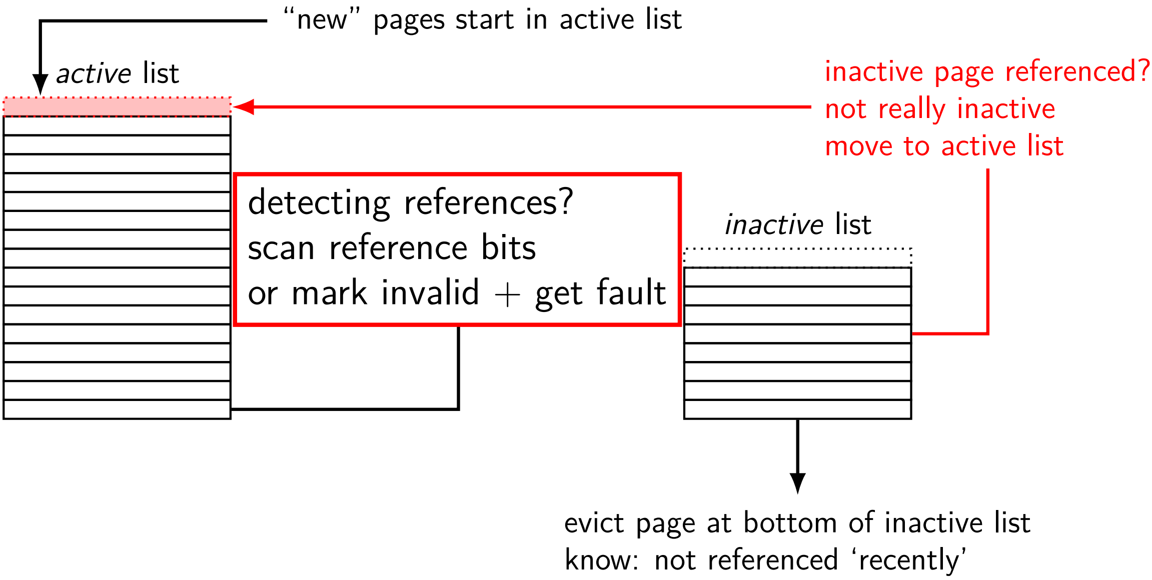
approximating LRU: SEQ



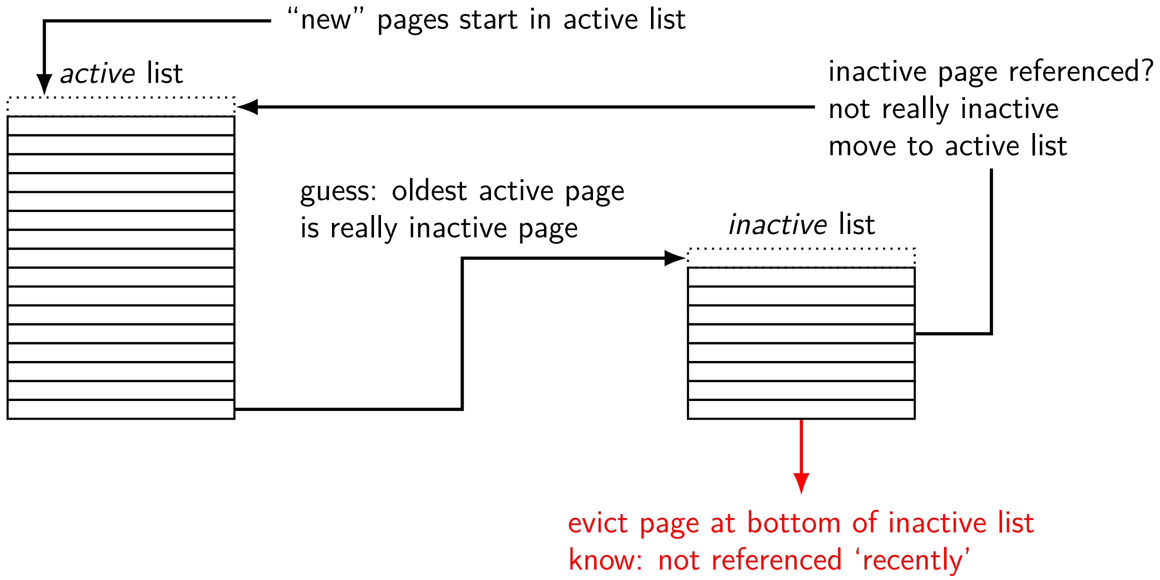
approximating LRU: SEQ



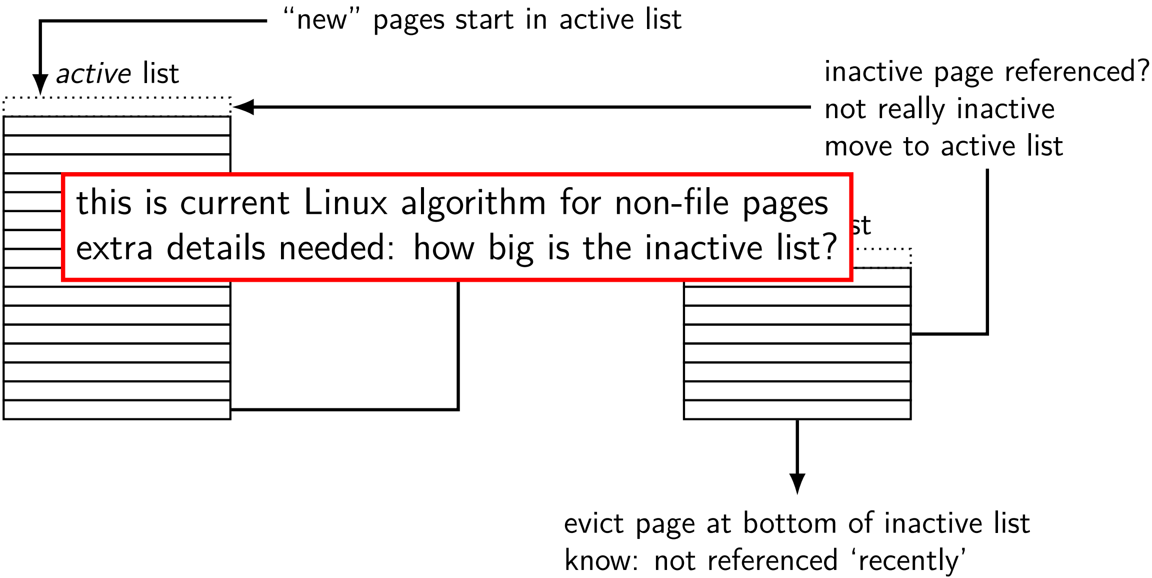
approximating LRU: SEQ



approximating LRU: SEQ



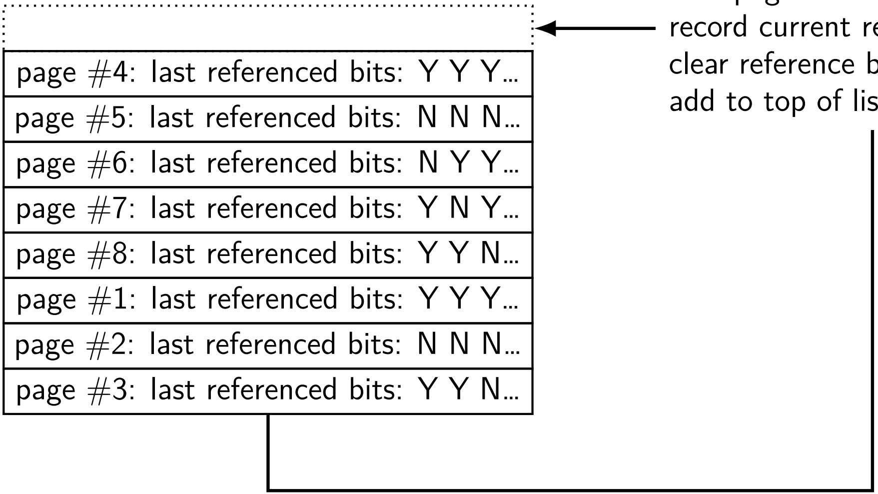
approximating LRU: SEQ



this is current Linux algorithm for non-file pages
extra details needed: how big is the inactive list?

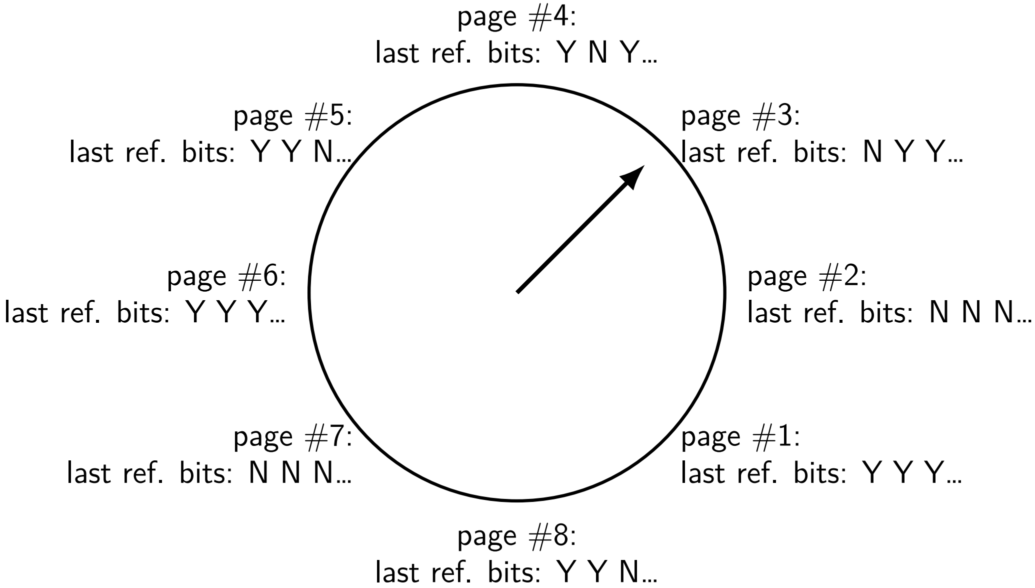
tracking usage: CLOCK (view 1)

ordered list
of physical pages



periodically:
take page from bottom of list
record current referenced bit
clear reference bit for next pass
add to top of list

tracking usage: CLOCK (view 2)



problems with LRU

question: when does LRU perform poorly?

exercise: which of these is LRU bad for?

code in a text editor for handling out-of-disk-space errors

initial values of the shell's global variales

on a desktop, long movies that are too big to fit in memory and played from beginning to end

on web server, long movies that are too big to fit in memory and frequently downloaded by clients

files that are parsed when loaded and overwritten when saved

on web server, frequently requested HTML files

solution for LRU being bad?

one idea that Linux uses:

for *file data*, use different replacement policy

tries to avoid keeping around file data accessed only once

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 implementation ideas?

which of these is probably best?

(a) when there's a page fault requring reading page X of a file from disk, read pages X and $X + 1$

(b) when there's a page fault requring reading page $X > 200$ of a file from disk, read the rest of the file

(c) when page fault occurs for page X of a file, read pages X through $X + 200$ and proactively add all to the current program's page table

(d) when page fault occurs for page X of a file, read pages X through $X + 200$ but don't place pages $X + 1$ through $X + 200$ in the page table yet

being less lazy elsewhere

showed OS: proactively reading in pages

can also proactively free pages (faster replacement)

and proactively write out pages 'dirty' pages

- save time writing later
- avoid data loss on power failure

page cache/replacement summary

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

mostly, assume temporal locality

- least recently used variants

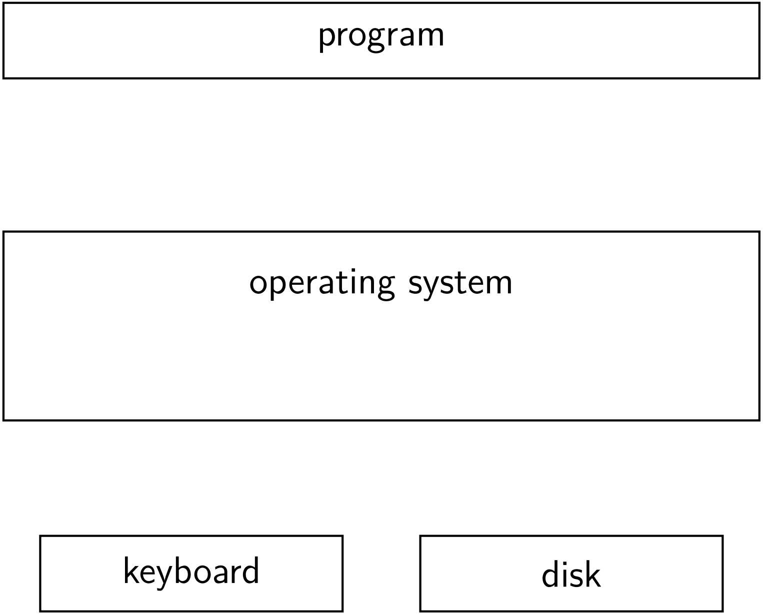
special cases for non-LRU-friendly patterns (e.g. scans)

- maybe more we haven't discussed?

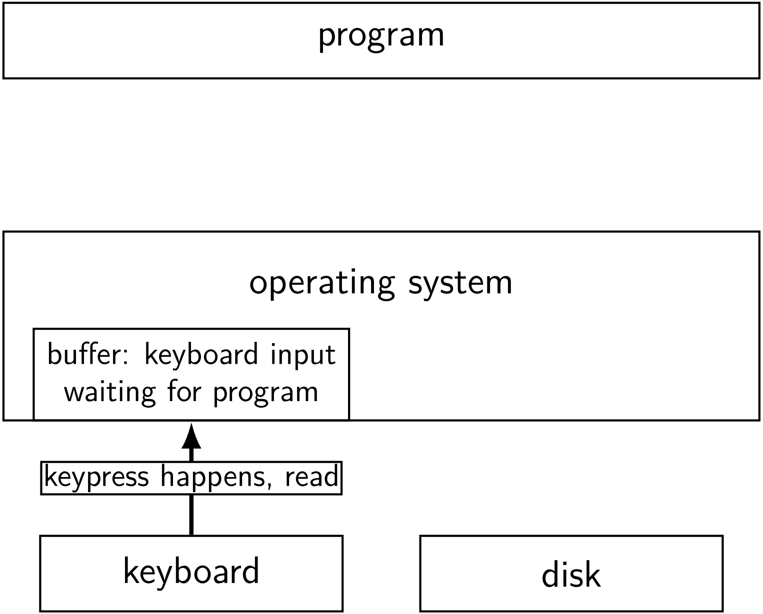
being proactive (writeback early, readahead, pre-evicted pages)

missing: handling non-miss-rate goals?

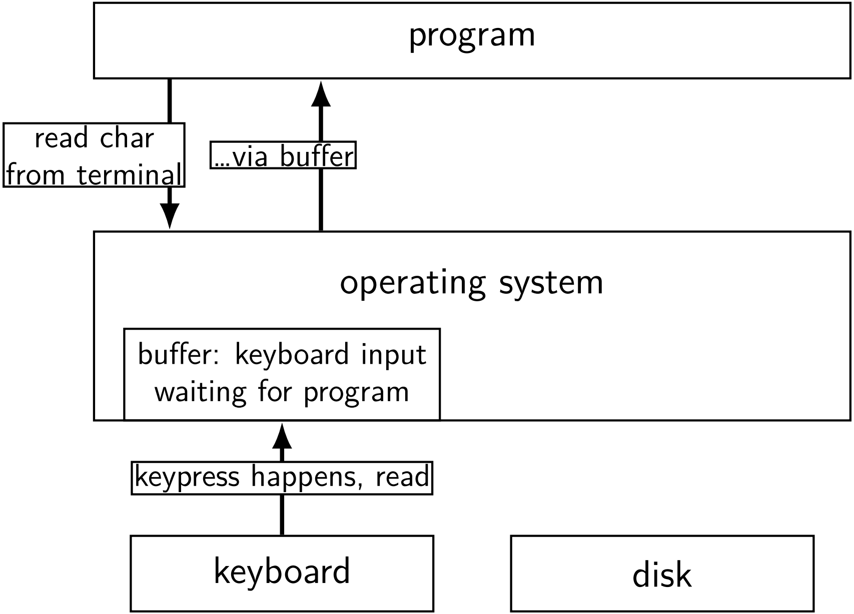
kernel buffering (reads)



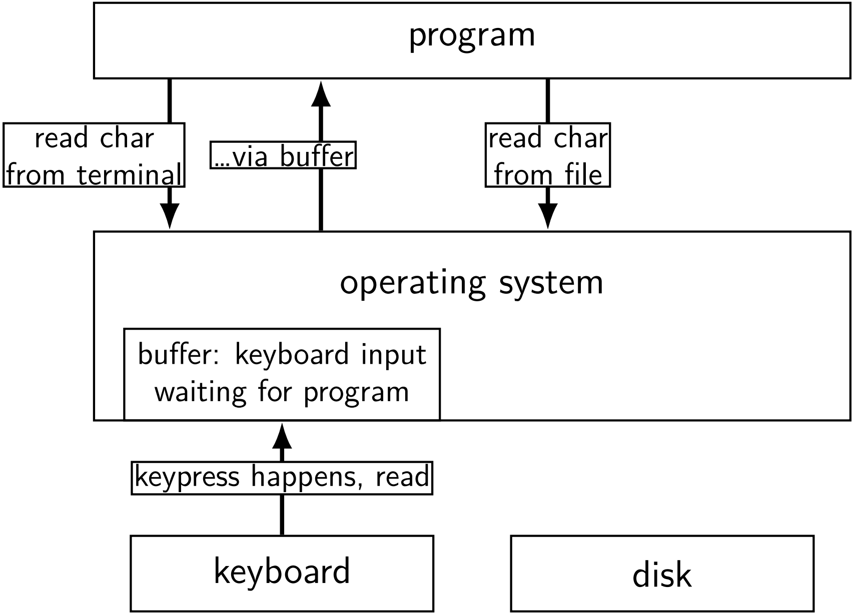
kernel buffering (reads)



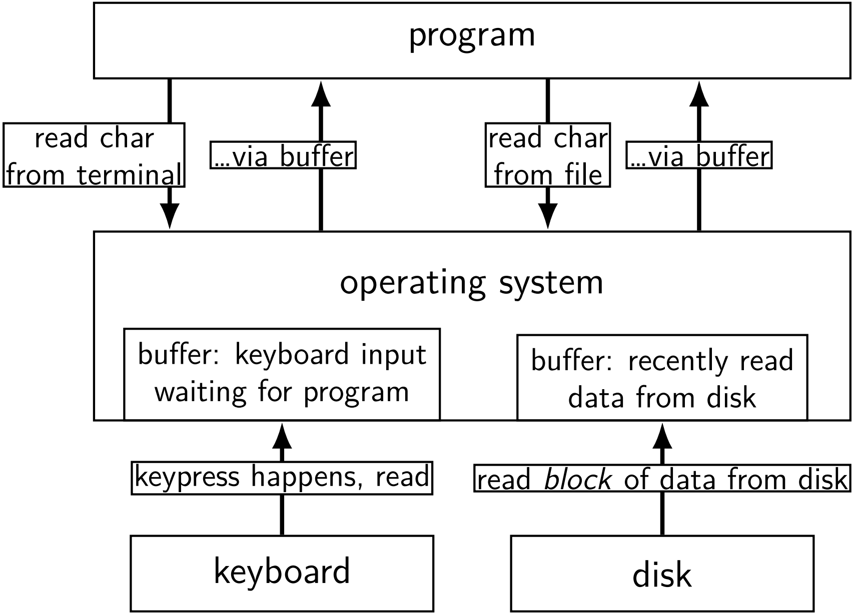
kernel buffering (reads)



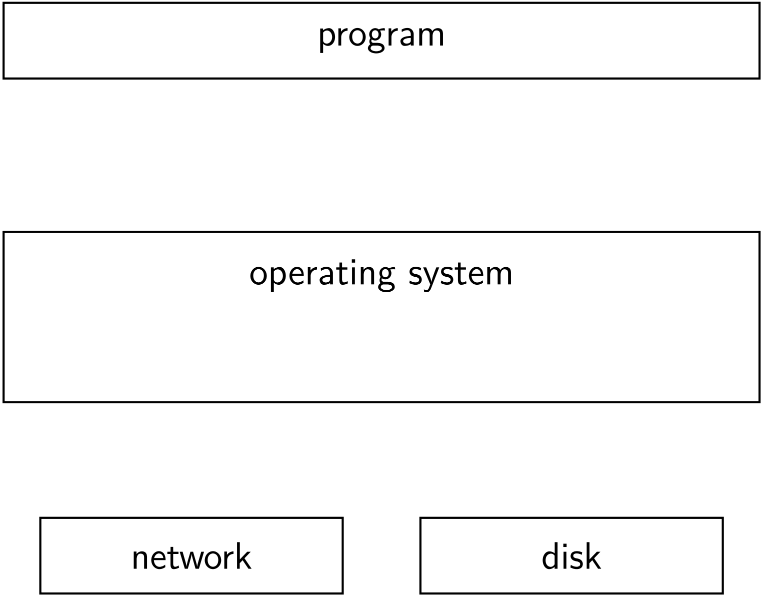
kernel buffering (reads)



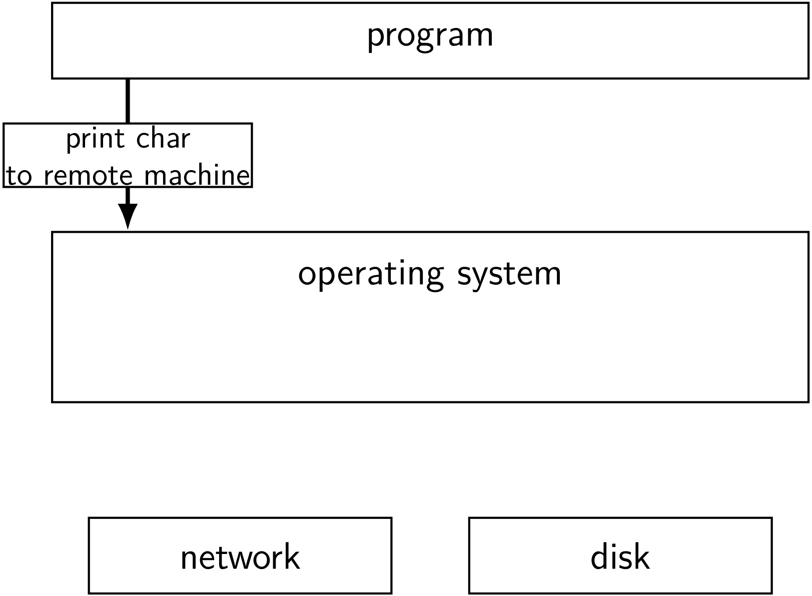
kernel buffering (reads)



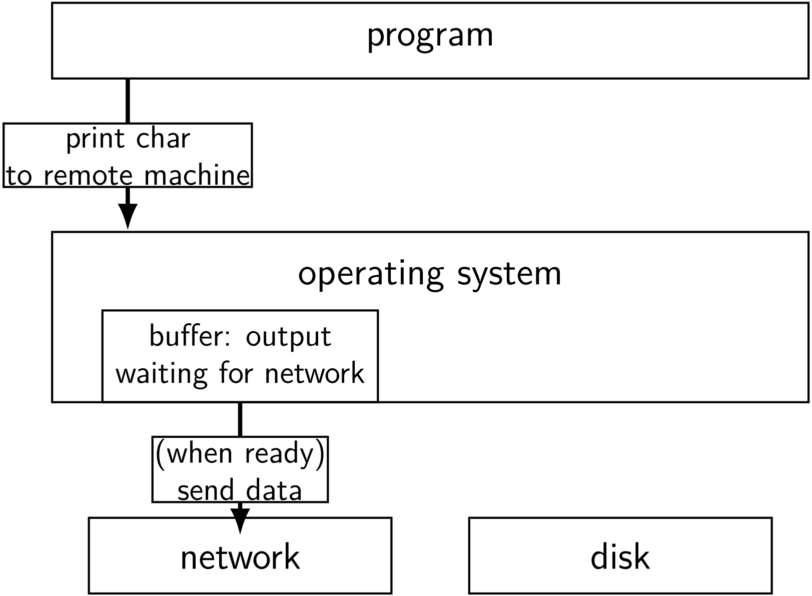
kernel buffering (writes)



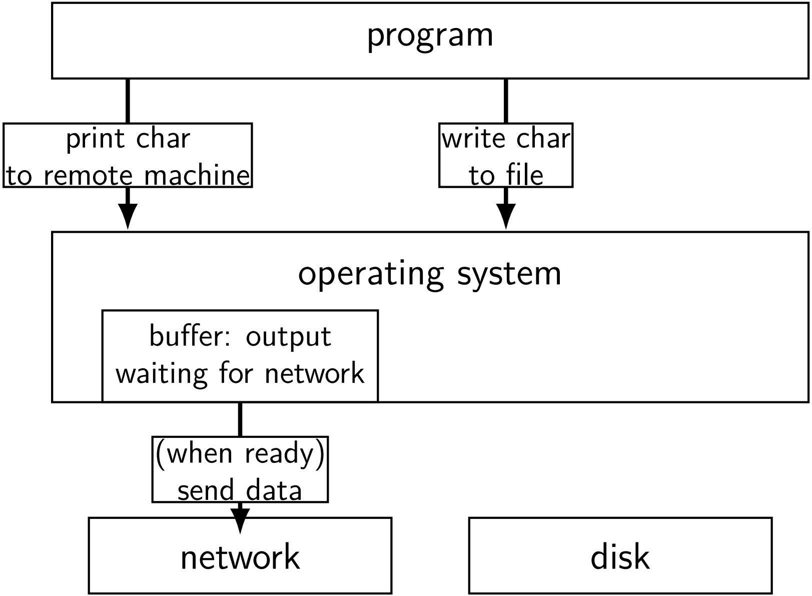
kernel buffering (writes)



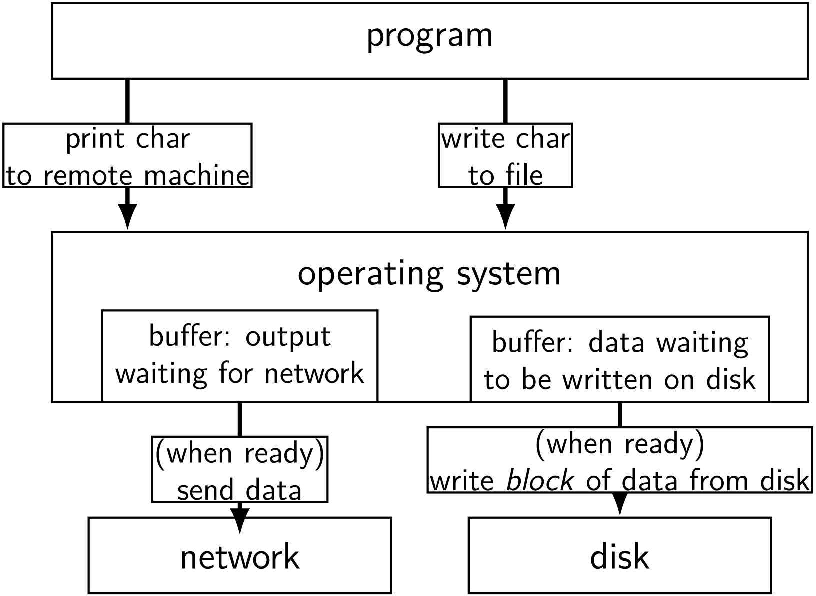
kernel buffering (writes)



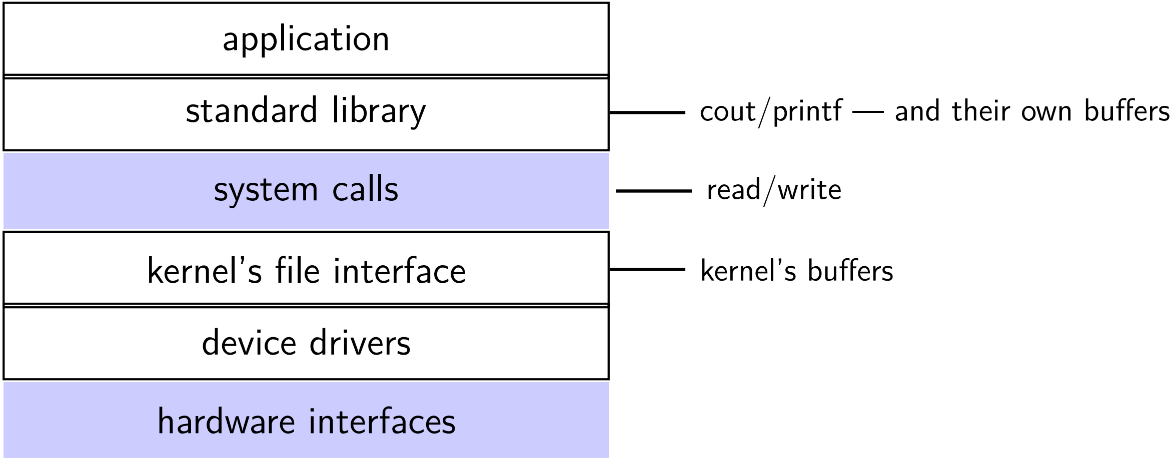
kernel buffering (writes)



kernel buffering (writes)



layering


application
standard library
system calls
kernel's file interface
device drivers
hardware interfaces
cout/printf — and their own buffers
read/write
kernel's buffers

backup slides

recording accesses

goal: "check is this physical page still being used?"

software support: temporarily mark page table invalid

- use resulting page fault to detect "yes"

hardware support: accessed bits in page tables

- hardware sets to 1 when accessed

temporarily invalid PTE (software support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

the kernel

…
(OS exception's handler)
…

page table for program 1

VPN	present?	writable?	…	PPN
0x00000	0	---	…	---
0x00001	0	---	…	---
…	…	…	…	…
0x00123	0	0	…	0x4442
…	…	…	…	…

OS page info

PPN	last known access?	…
…	…	…
0x04442	(never)	…
…	…	…

temporarily invalid PTE (software support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

the kernel

…
(OS exception's handler)
…

oops! page fault

processor does lookup

page table for program 1

VPN	present?	writable?	…	PPN
0x00000	0	---	…	---
0x00001	0	---	…	---
…	…	…	…	…
0x00123	0	0	…	0x4442
…	…	…	…	…

OS page info

PPN	last known access?	…
…	…	…
0x04442	(never)	…
…	…	…

temporarily invalid PTE (software support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
...
...
mov 0x123300, %ecx
```

the kernel

...
(OS exception's handler)
...

update page info +
mark present

page table for program 1

VPN	present?	writable?	...	PPN
0x00000	0	---	...	---
0x00001	0	---	...	---
...	...	...	...	...
0x00123	1	0	...	0x4442
...	...	...	...	...

OS page info

PPN	last known access?	...
...	...	...
0x04442	at time X	
...	...	...

temporarily invalid PTE (software support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

the kernel

…
(OS exception's handler)
…

processor does lookup

no page fault, not recorded in OS info

page table for program 1

VPN	present?	writable?	…	PPN
0x00000	0	---	…	---
0x00001	0	---	…	---
…	…	…	…	…
0x00123	1	0	…	0x4442
…	…	…	…	…

OS page info

PPN	last known access?	…
…	…	…
0x04442	at time X	…
…	…	…

temporarily invalid PTE (software support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

the kernel

…
(OS exception's handler)
…

processor does lookup

no page fault, not recorded in OS info

page table for program 1

VPN	present?	writable?	…	PPN
0x00000	0	---	…	---
0x00001	0	---	…	---
…	…	…	…	…
0x00123	1	0	…	0x4442
…	…	…	…	…

OS page info

PPN	last known access?	…
…	…	…
0x04442	at time X	…
…	…	…

temporarily invalid PTE (software support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

the kernel

…
(OS exception's handler)
…

OS clears present bit
to check for next access

page table for program 1

VPN	present?	writable?	…	PPN
0x00000	0	---	…	---
0x00001	0	---	…	---
…	…	…	…	…
0x00123	1	0	…	0x4442
…	…	…	…	…

OS page info

PPN	last known access?	…
…	…	…
0x04442	at time X	…
…	…	…

temporarily invalid PTE (software support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

the kernel

…
(OS exception's handler)
…

OS clears present bit
to check for next access

page table for program 1

VPN	present?	writable?	…	PPN
0x00000	0	---	…	---
0x00001	0	---	…	---
…	…	…	…	…
0x00123	0	0	…	0x4442
…	…	…	…	…

OS page info

PPN	last known access?	…
…	…	…
0x04442	at time X	…
…	…	…

temporarily invalid PTE (software support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
...
...
mov 0x123300, %ecx
```

processor does lookup

the kernel

...
(OS exception's handler)
...

oops! page fault

page table for program 1

VPN	present?	writable?	...	PPN
0x00000	0	---	...	---
0x00001	0	---	...	---
...	...	...	...	...
0x00123	0	0	...	0x4442
...	...	...	...	...

OS page info

PPN	last known access?	...
...	...	...
0x04442	at time X	...
...	...	...

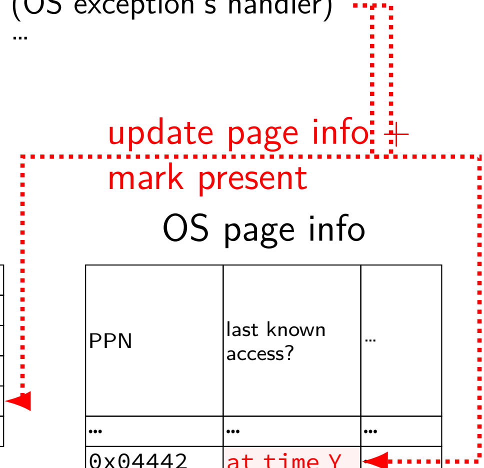
temporarily invalid PTE (software support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

the kernel

…
(OS exception's handler)
…



page table for program 1

VPN	present?	writable?	…	PPN
0x00000	0	---	…	---
0x00001	0	---	…	---
…	…	…	…	…
0x00123	1	0	…	0x4442
…	…	…	…	…

OS page info

PPN	last known access?	…
…	…	…
0x04442	at time Y	…
…	…	…

accessed bit usage (hardware support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

the kernel

…

(OS exception's handler)

…

page table for program 1

VPN	present?	accessed?	writable?	…	PPN
0x00000	0	---	---	…	---
0x00001	0	---	---	…	---
…	…	…	…	…	…
0x00123	1	0	0	…	0x4442
…	…	…	…	…	…

accessed bit usage (hardware support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

processor does lookup

sets accessed bit to 1

the kernel

…

(OS exception's handler)

…

page table for program 1

VPN	present?	accessed?	writable?	…	PPN
0x00000	0	---	---	…	---
0x00001	0	---	---	…	---
…	…	…	…	…	…
0x00123	1	0	0	…	0x4442
…	…	…	…	…	…

accessed bit usage (hardware support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

processor does lookup

sets accessed bit to 1

the kernel

…

(OS exception's handler)

…

page table for program 1

VPN	present?	accessed?	writable?	…	PPN
0x00000	0	---	---	…	---
0x00001	0	---	---	…	---
…	…	…	…	…	…
0x00123	1	1	0	…	0x4442
…	…	…	…	…	…

accessed bit usage (hardware support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

the kernel

…
(OS exception's handler)
…

processor does lookup

keeps access bit set to 1

page table for program 1

VPN	present?	accessed?	writable?	…	PPN
0x00000	0	---	---	…	---
0x00001	0	---	---	…	---
…	…	…	…	…	…
0x00123	1	1	0	…	0x4442
…	…	…	…	…	…

accessed bit usage (hardware support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

the kernel

…
(OS exception's handler)
…

processor does lookup

keeps access bit set to 1

page table for program 1

VPN	present?	accessed?	writable?	…	PPN
0x00000	0	---	---	…	---
0x00001	0	---	---	…	---
…	…	…	…	…	…
0x00123	1	1	0	…	0x4442
…	…	…	…	…	…

accessed bit usage (hardware support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

the kernel

…
(OS exception's handler)
…

OS reads + records + clears access bit

page table for program 1

VPN	present?	accessed?	writable?	…	PPN
0x00000	0	---	---	…	---
0x00001	0	---	---	…	---
…	…	…	…	…	…
0x00123	1	1	0	…	0x4442
…	…	…	…	…	…

accessed bit usage (hardware support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

the kernel

…
(OS exception's handler)
…

OS reads + records + clears access bit

page table for program 1

VPN	present?	accessed?	writable?	…	PPN
0x00000	0	---	---	…	---
0x00001	0	---	---	…	---
…	…	…	…	…	…
0x00123	1	0	0	…	0x4442
…	…	…	…	…	…

accessed bit usage (hardware support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

the kernel

…
(OS exception's handler)
…

processor does lookup

sets accessed bit to 1 (again)

page table for program 1

VPN	present?	accessed?	writable?	…	PPN
0x00000	0	---	---	…	---
0x00001	0	---	---	…	---
…	…	…	…	…	…
0x00123	1	0	0	…	0x4442
…	…	…	…	…	…

accessed bit usage (hardware support)

program 1

```
mov 0x123456, %ecx
mov 0x123789, %ecx
...
...
mov 0x123300, %ecx
```

the kernel

...
(OS exception's handler)
...

processor does lookup

sets accessed bit to 1 (again)

page table for program 1

VPN	present?	accessed?	writable?	...	PPN
0x00000	0	---	---	...	---
0x00001	0	---	---	...	---
...	...	...	...	...	...
0x00123	1	1	0	...	0x4442
...	...	...	...	...	...

accessed bits: multiple processes

page table for program 1

VPN	present?	accessed?	writable?	…	PPN
0x00000	0	---	---	…	---
0x00001	0	---	---	…	---
…	…	…	…	…	…
0x00123	1	0	0	…	0x4442
…	…	…	…	…	…

page table for program 2

VPN	present?	accessed?	writable?	…	PPN
0x00000	0	---	---	…	---
0x00001	0	---	---	…	---
…	…	…	…	…	…
0x00483	1	1	0	…	0x4442
…	…	…	…	…	…

OS needs to clear+check **all** accessed bits for the physical page

dirty bits

"was this part of the mmap'd file changed?"

"is the old swapped copy still up to date?"

software support: temporarily mark read-only

hardware support: ***dirty bit*** set by hardware

- same idea as accessed bit, but only changed on writes

x86-32 accessed and dirty bit

Address of 4KB page frame	Ignored	G	PAT	D	A	PCD	PWT	U/S	R/W	1	**PTE: 4KB page**
Ignored										0	**PTE: not present**

Figure 4-4. Formats of CR3 and Paging-Structure Entries with 32-Bit Paging

A: acccessed — processor sets to 1 when PTE used

- used = for read or write or execute
- likely implementation: part of loading PTE into TLB

D: dirty — processor sets to 1 when PTE is used for write

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.)

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: delay considering pages active until second access

- second access = second scan of accessed bits/etc.

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

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?

readahead heuristics

exercise: devise an algorithm to detect to do readahead.

how to detect the reading pattern?

- need to record subset of accesses to see sequential pattern
- not enough to look at misses!
- want to check when readahead pages are used — keep up with program

when to start reads?

how much to readahead?

readahead heuristics

exercise: devise an algorithm to detect to do readahead.

how to detect the reading pattern?

- need to record subset of accesses to see sequential pattern
- not enough to look at misses!
- want to check when readahead pages are used — keep up with program

when to start reads?

- takes some time to read in data — well before needed

how much to readahead?

readahead heuristics

exercise: devise an algorithm to detect to do readahead.

how to detect the reading pattern?

- need to record subset of accesses to see sequential pattern
- not enough to look at misses!
- want to check when readahead pages are used — keep up with program

when to start reads?

- takes some time to read in data — well before needed

how much to readahead?

- if too much: evict other stuff programs need
- if too little: won't keep up with program
- if too little: won't make efficient use of HDD/SSD/etc.