virtual memory 4

Zoom logistics

recommend: exit full screen

open chat + participants window

participants window has non-verbal feedback features

I will try to monitor the chat window

I can take questions via raise hand + turn on your audio...

but probably text is usually easier/more reliable?

I intend to record these (both through Zoom and locally)

last time

mmap

- allow programs to place files in their memory
- multiple users of file: get same physical memory

page cache idea

- most of memory is cache for program + file data

page cache data structures

- hit: page table (HW), OS stuff for file locations
- miss: file location to disk mappping (filesystem)
- miss: program location to disk mapping (trick: in PTE?)

supporting page *replacement*

- out of space? evict used page + replace with new data
- reverse mappings to remove pointers to evicted page
  - from all page tables, etc.

page cache components [text]

mapping: virtual address or file+offset → physical page

- handle cache hits

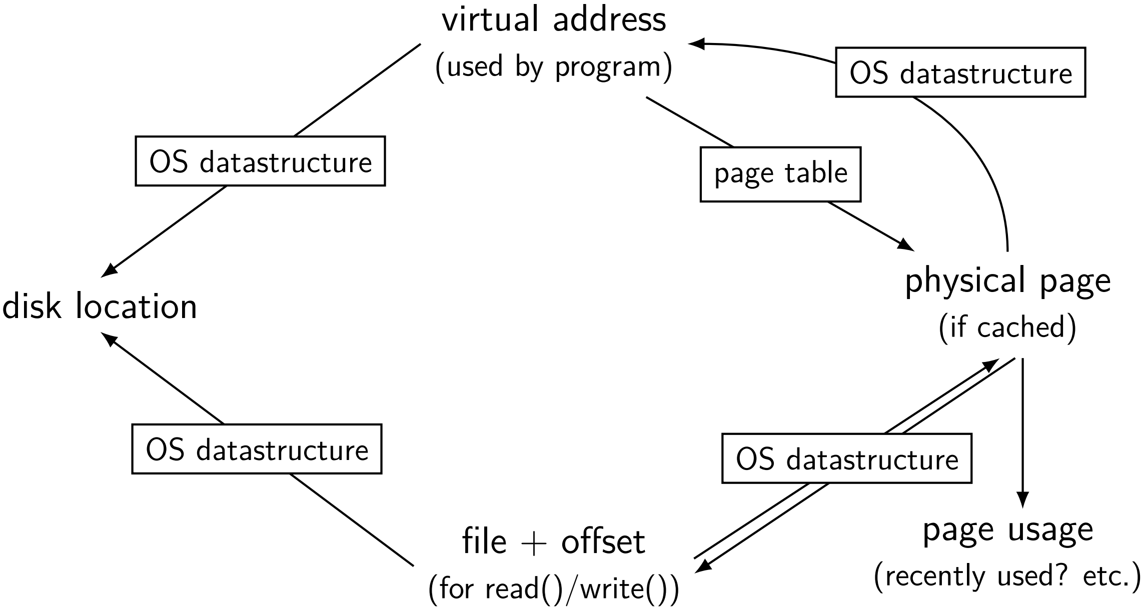
find backing location based on virtual address/file+offset

- handle cache misses

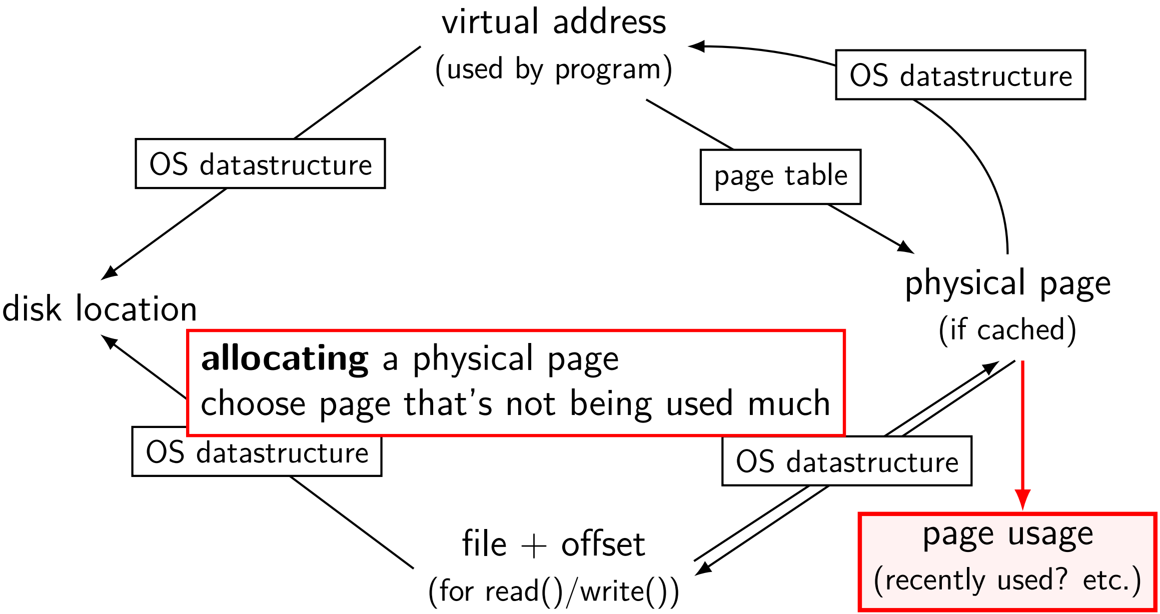
track information about each physical page

- handle page allocation
- handle cache eviction

page cache components



page cache components



allocating a physical page

choose page that's not being used much

tracking physical pages: finding free pages

Linux has list of “least recently used” pages:

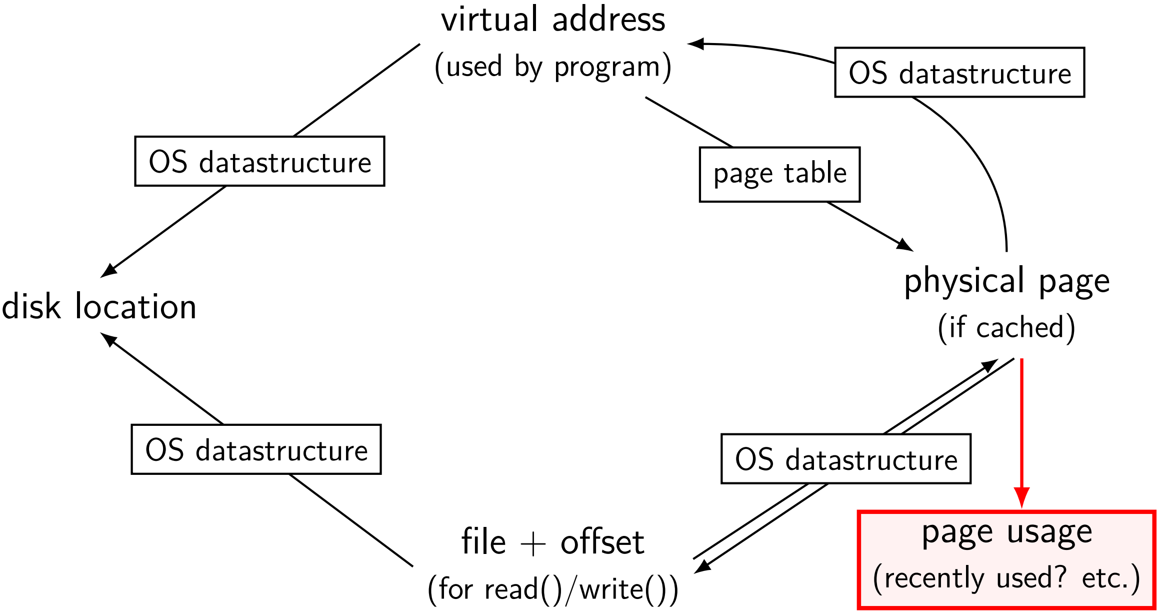
```
struct page {
    ...
    struct list_head lru;   /* list_head ~ next/prev pointer */
    ...
};
```

how we’re going to find a page to allocate

(and evict from something else)

later — what this list actually looks like (how many lists, ...)

page cache components



page replacement goals

hit rate: minimize number of misses

throughput: minimize overhead/maximize performance

fairness: every process/user gets its ‘share’ of memory

will start with optimizing hit rate

max hit rate $\approx$ max throughput

optimizing hit rate almost optimizes throughput, but...

max hit rate $\approx$ max throughput

optimizing hit rate almost optimizes throughput, but…

cache miss costs are variable

- creating zero page versus reading data from slow disk?
- write back dirty page before reading a new one or not?
- reading multiple pages at a time from disk (faster per page read)?
- …

being proactive?

can avoid misses by “reading ahead”

- guess what’s needed — read in ahead of time
- wrong guesses can have costs besides more cache misses

can save modified pages to disk in the background

we will get back to this later

for now — only access/evict on demand

optimizing for hit-rate

assuming:

- we only bring in pages on demand (no reading in advance)
- we only care about maximizing cache hits

best possible page replacement algorithm: Belady's MIN

replace the page in memory accessed furthest in the future

- (never accessed again = infinitely far in the future)

optimizing for hit-rate

assuming:

we only bring in pages on demand (no reading in advance)
we only care about maximizing cache hits

best possible page replacement algorithm: Belady's MIN

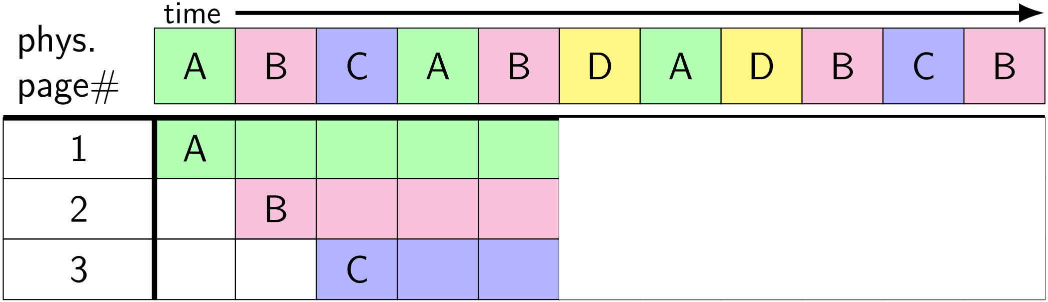
replace the page in memory accessed furthest in the future

(never accessed again = infinitely far in the future)

impossible to implement in practice, but...

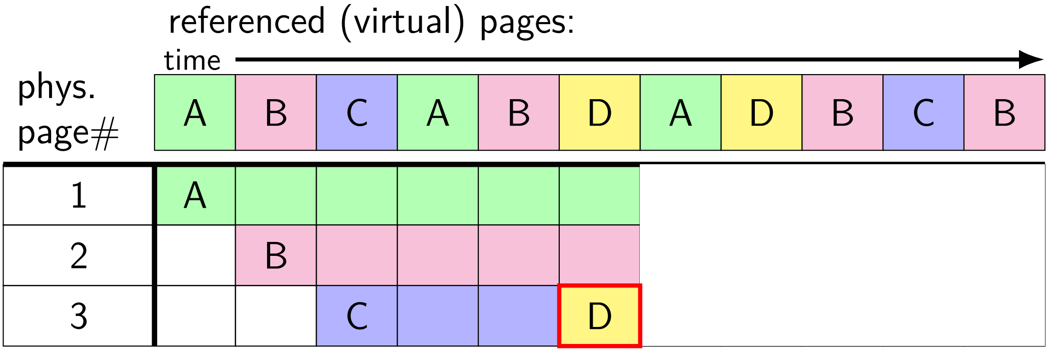
Belady's MIN

referenced (virtual) pages:



phys. page#	A	B	C	A	B	D	A	D	B	C	B
1	A	🟩	🟩	🟩	🟩						
2		B	🟥	🟥	🟥						
3			C	🟦	🟦						

Belady's MIN



referenced (virtual) pages:

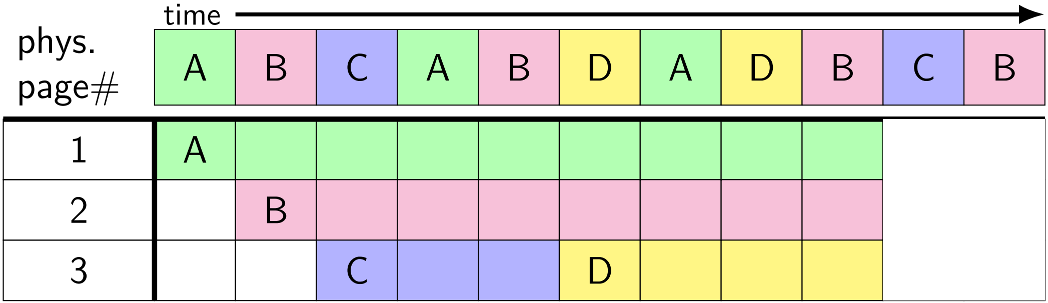
time →

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

A next accessed in 1 time unit
B next accessed in 3 time units
C next accessed in 4 time units
choose to replace C

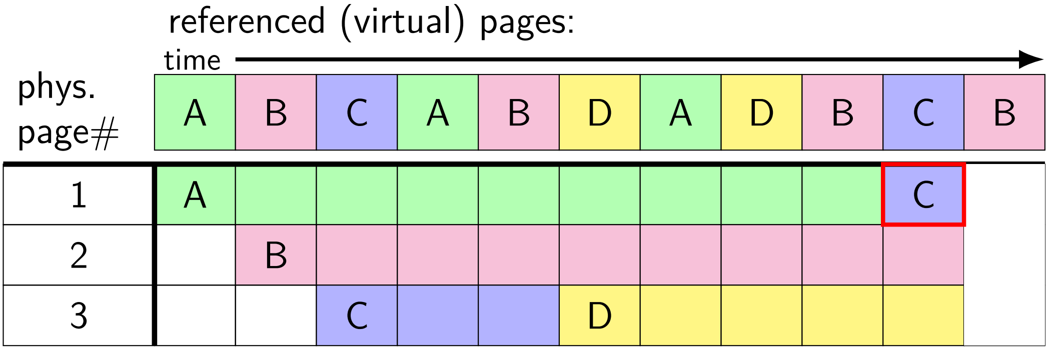
Belady's MIN

referenced (virtual) pages:



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

Belady's MIN



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					

A next accessed in ∞ time units
B next accessed in 1 time units
D next accessed in ∞ time units
choose to replace A or D (equally good)

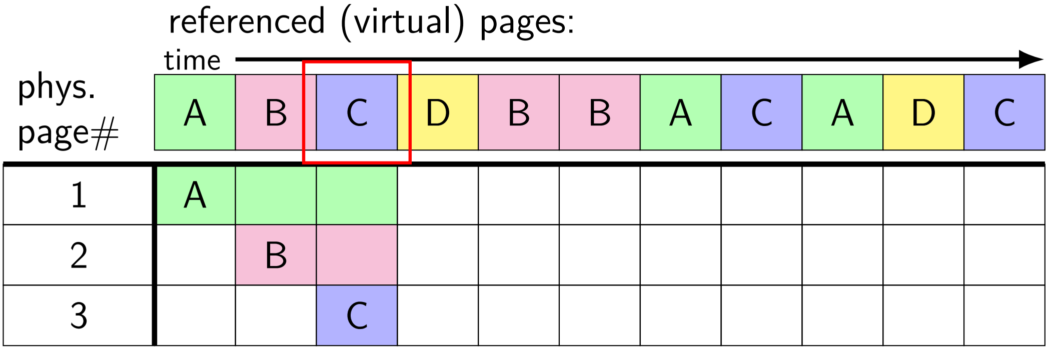
Belady's MIN

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					

Belady's MIN exercise



referenced (virtual) pages:

time

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

exercise: What does this access to D replace? (A, B, or C?)

predicting the future?

can't really...

look for common patterns

working set intuition

say we're executing a loop

what memory does this require?

code for the loop

code for functions called in the loop

and functions they call

data structures used by the loop and functions called in it, etc.

only uses a subset of the program's memory

the working set model

one common pattern: working sets

at any time, program is using a subset of its memory

...called its *working set*

rest of memory is inactive

...until program switches to different working set

working sets and running many programs

give each program its working set

…and, to run as much as possible, not much more

  inactive — won't be used

working sets and running many programs

give each program its working set

...and, to run as much as possible, not much more

inactive — won't be used

replacement policy: identify working sets ≈ recently used data

replace anything that's not in in it

cache size versus miss rate

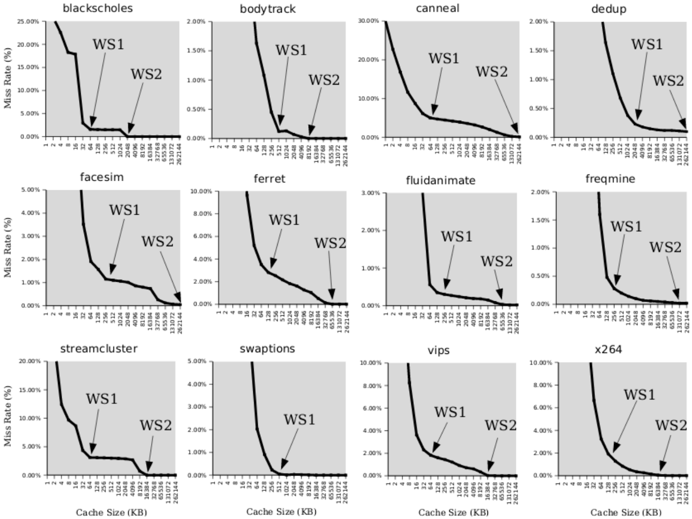


Figure 3: Miss rates versus cache size. Data assumes a shared 4-way associative cache with 64 byte lines. WS1 and WS2 refer to important working sets which we analyze in more detail in Table 2. Cache requirements of PARSEC benchmark programs can reach hundreds of megabytes.

Bienia et al, "The PARSEC Benchmark Suite: Characterization and Architectural Implications"

estimating working sets

working set ≈ what's been used recently

- except when program switching working sets

so, what a program recently used ≈ working set

can use this idea to estimate working set (from list of memory accesses)

estimating working sets

working set ≈ what's been used recently

except when program switching working sets

so, what a program recently used ≈ working set

can use this idea to estimate working set (from list of memory accesses)

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

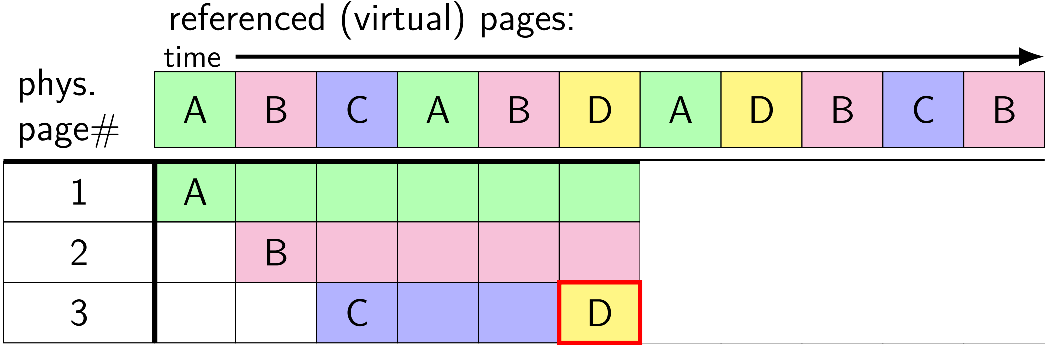
least recently used (the good case)

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:

time

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

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

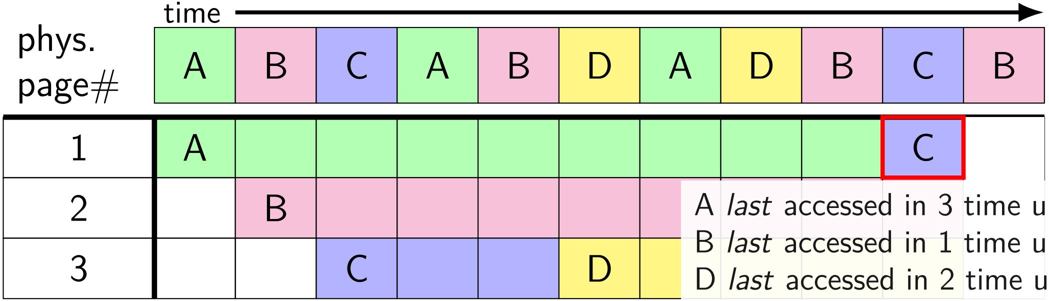
least recently used (the good case)

referenced (virtual) pages:

time →

phys. page#	A	B	C	A	B	D	A	D	B	C	B
1	A	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		
2		B	🟥	🟥	🟥	🟥	🟥	🟥	🟥		
3			C	🟦	🟦	D	🟨	🟨	🟨		

least recently used (the good case)



referenced (virtual) pages:

time

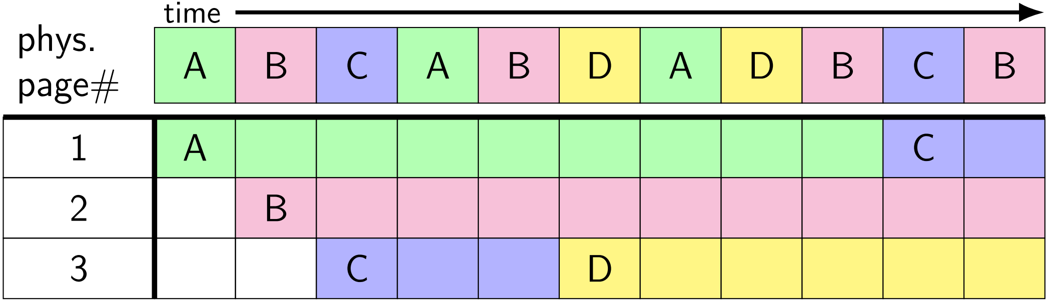
phys. page#	A	B	C	A	B	D	A	D	B	C	B
1	A									C	
2		B									
3			C			D					

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)

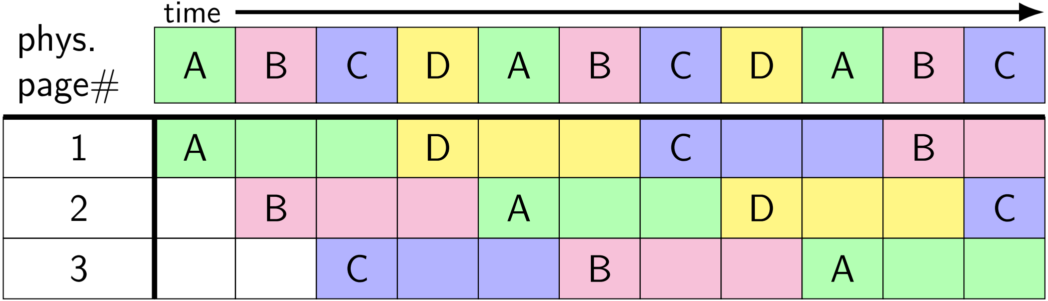
referenced (virtual) pages:

time →



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

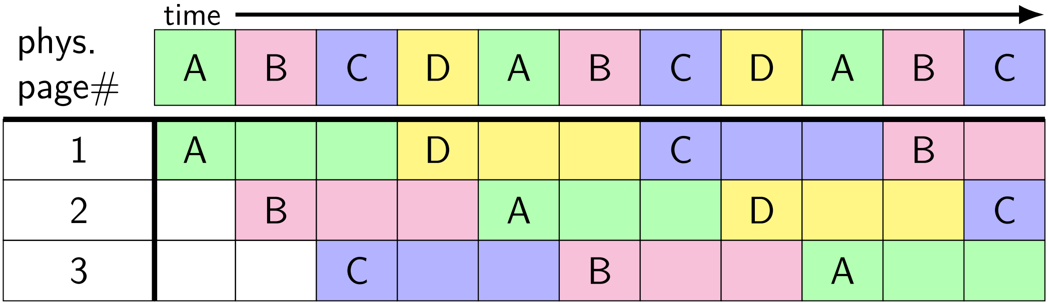
least recently used (the worst case)



time

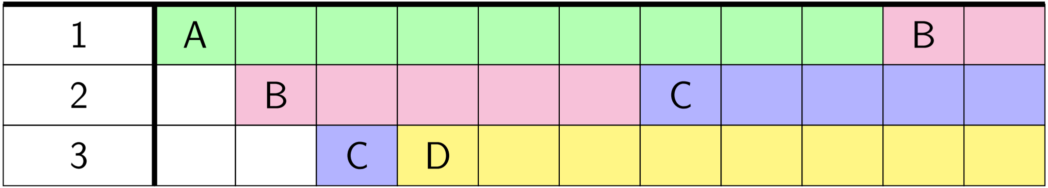
phys. page#	A	B	C	D	A	B	C	D	A	B	C
1	A			D			C			B	
2		B			A			D			C
3			C			B			A		

least recently used (the worst case)



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:



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							

least recently used (exercise) (2)

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

least recently used (exercise) (3)

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

least recently used (exercise) (4)

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

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

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

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

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

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

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)

…

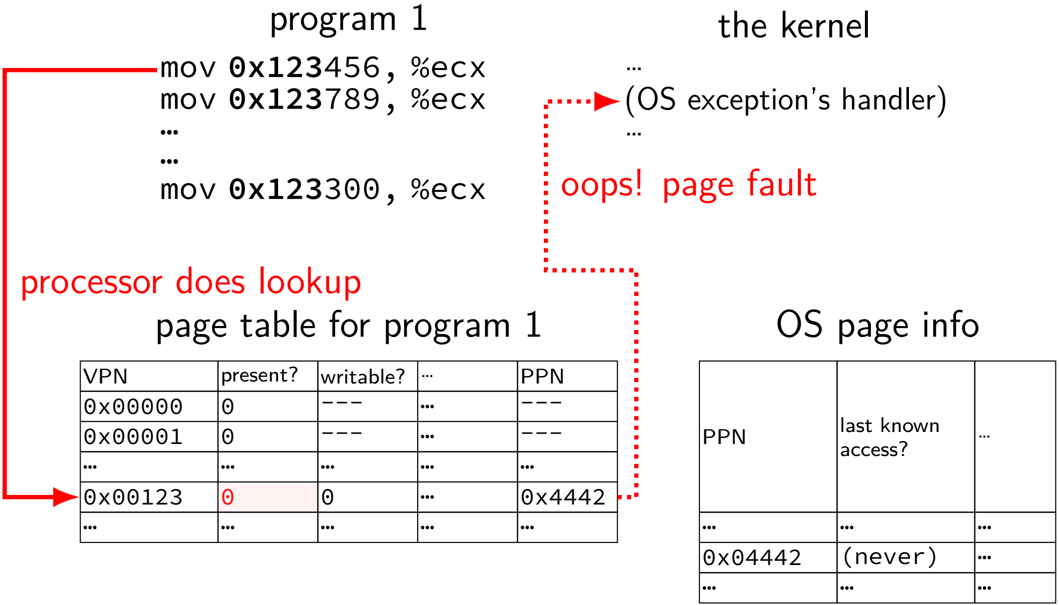
page table for program 1

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

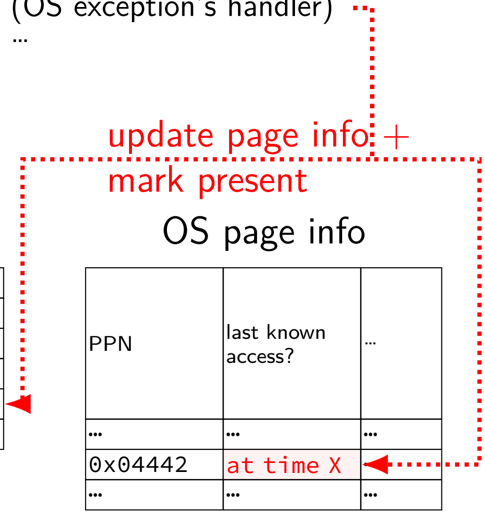
```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

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

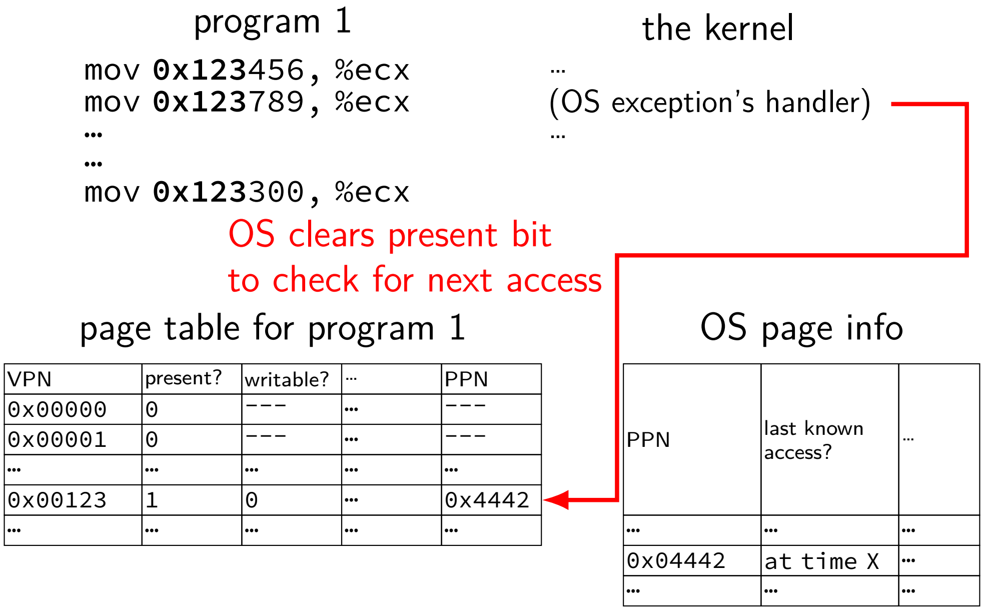
page table for program 1

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

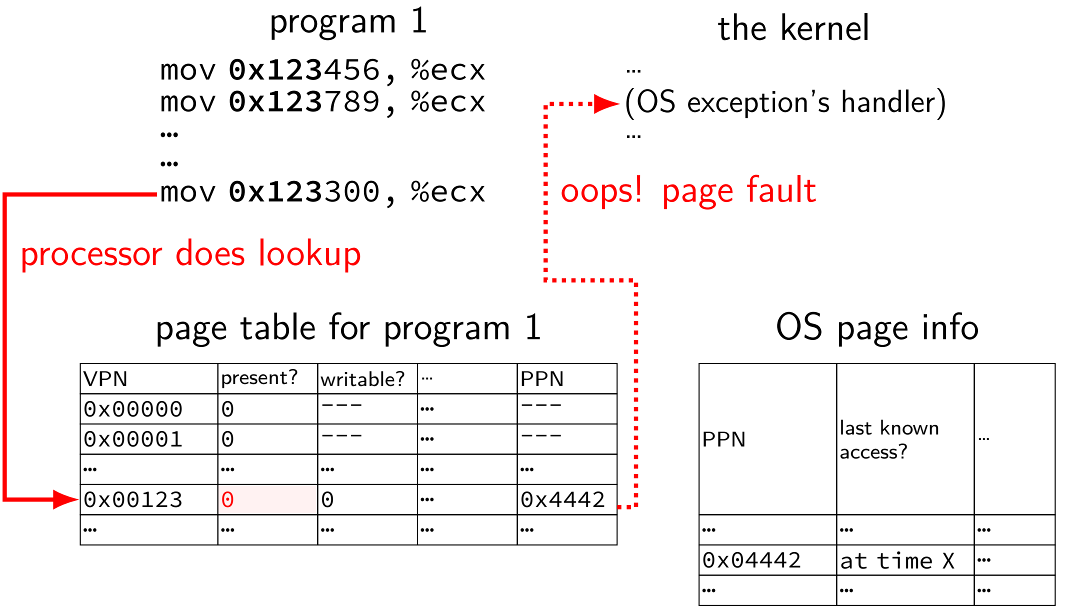
page table for program 1

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
```

the kernel

…
(OS exception's handler)
…

processor does lookup

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)

…

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 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
```

the kernel

…

(OS exception's handler)

…

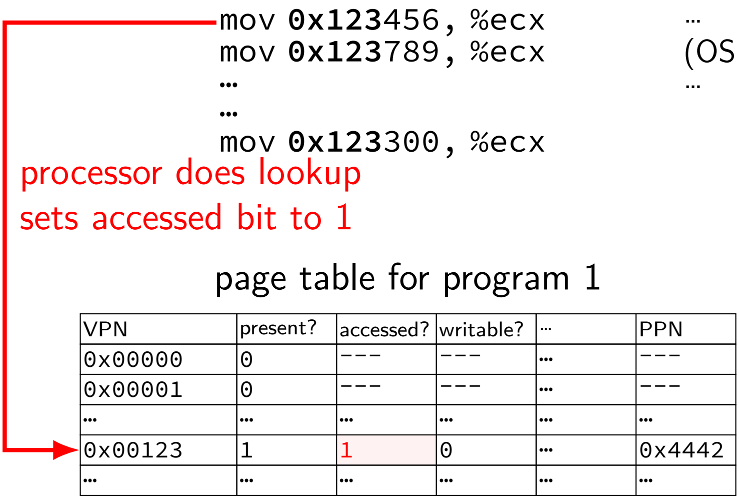
processor does lookup

sets accessed bit to 1

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

page table for program 1

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

the kernel

...

(OS exception's handler)

...

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?"

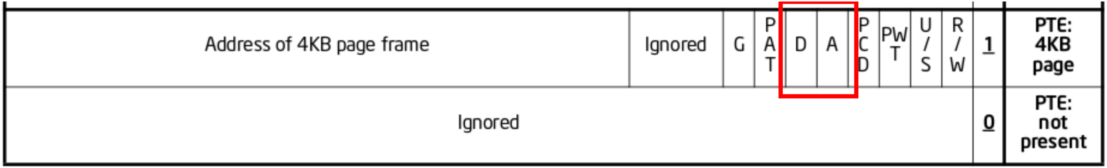
"is the old swapped copy still up to date?"

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	P A T	D	A	P C D	PW T	U / S	R / W	1	**PTE: 4KB page**
Ignored										0	**PTE: not present**

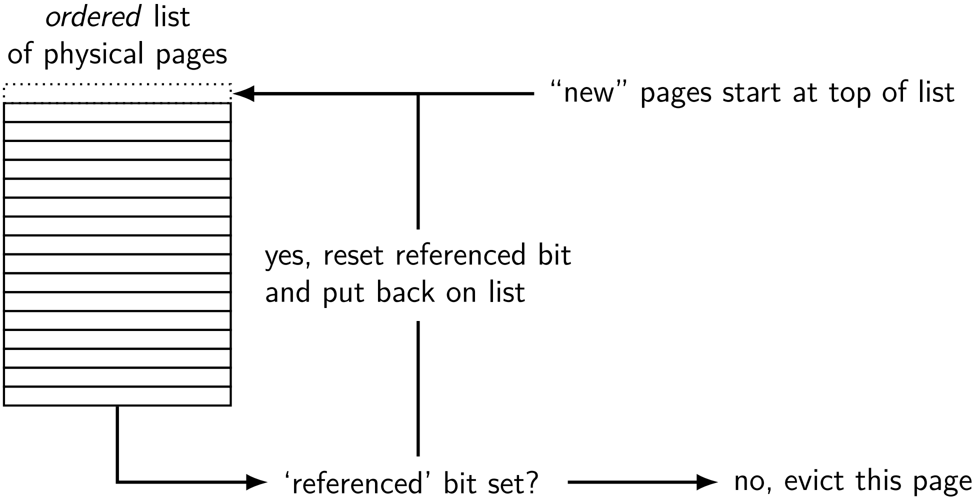
Figure 4-4. Formats of CR3 and Paging-Structure Entries with 32-Bit Paging

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

approximating LRU: second chance



ordered list
of physical pages

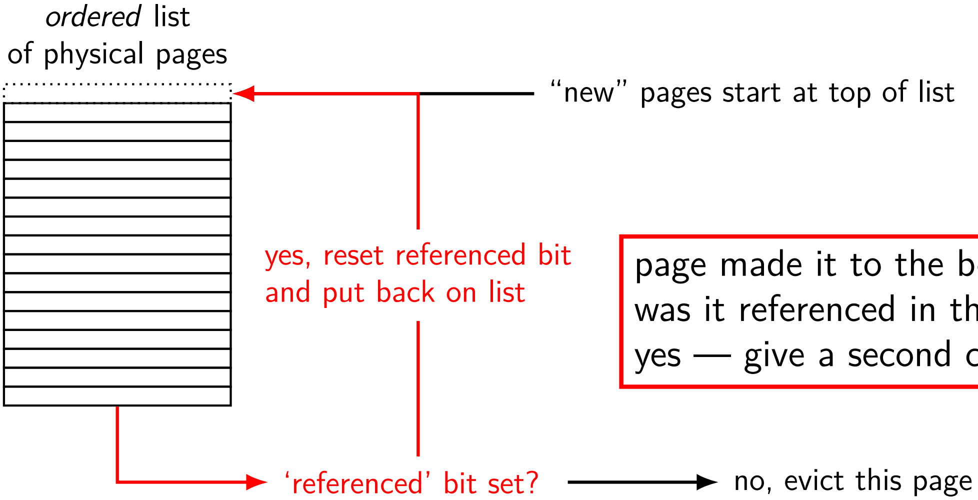
"new" pages start at top of list

yes, reset referenced bit
and put back on list

'referenced' bit set?

no, evict this page

approximating LRU: second chance

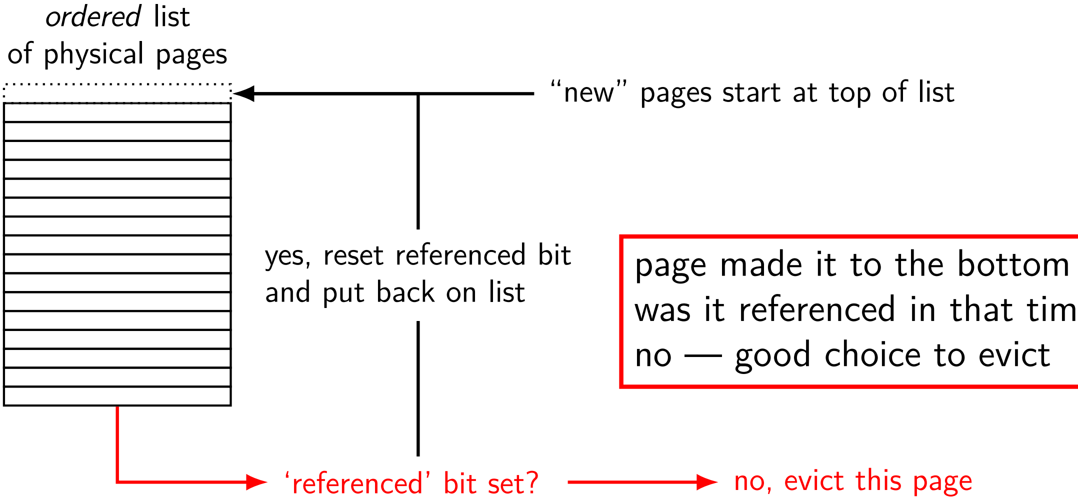


page made it to the bottom

was it referenced in that time?

yes — give a 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)

place A in physical page 1
accessed right after → 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			
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

place B in physical page 2
accessed right after → becomes referenced

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 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...

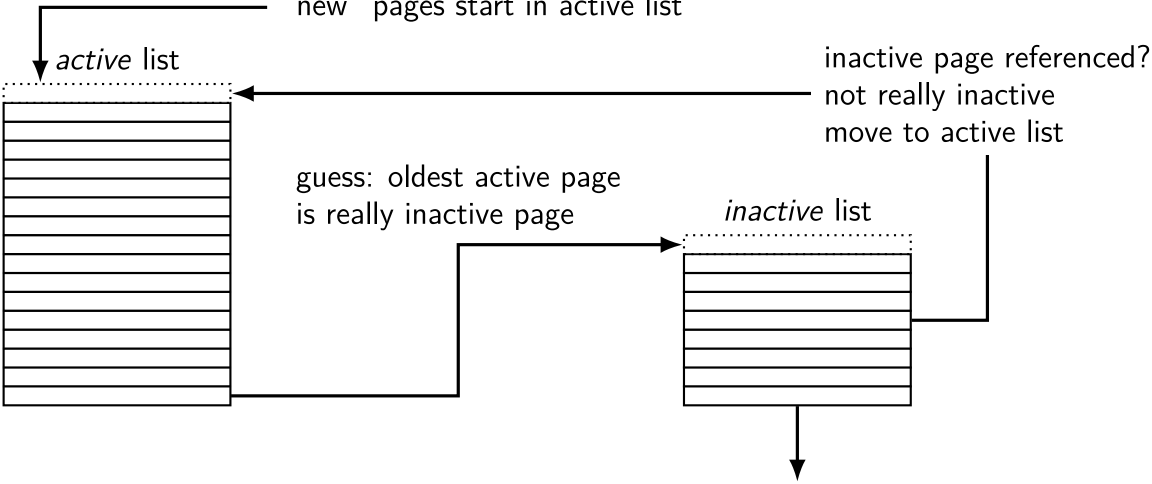
may need to scan through lots of pages to find unaccessed

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



“new” pages start in active list

active list

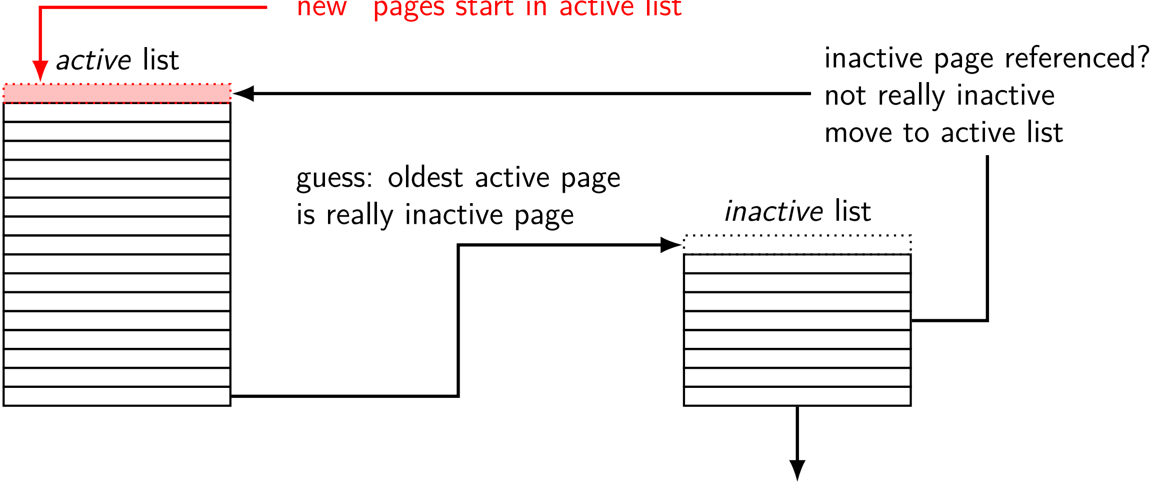
inactive page referenced?
not really inactive
move to active list

guess: oldest active page
is really inactive page

inactive list

evict page at bottom of inactive list
know: not referenced ‘recently’

approximating LRU: SEQ



“new” pages start in active list

active list

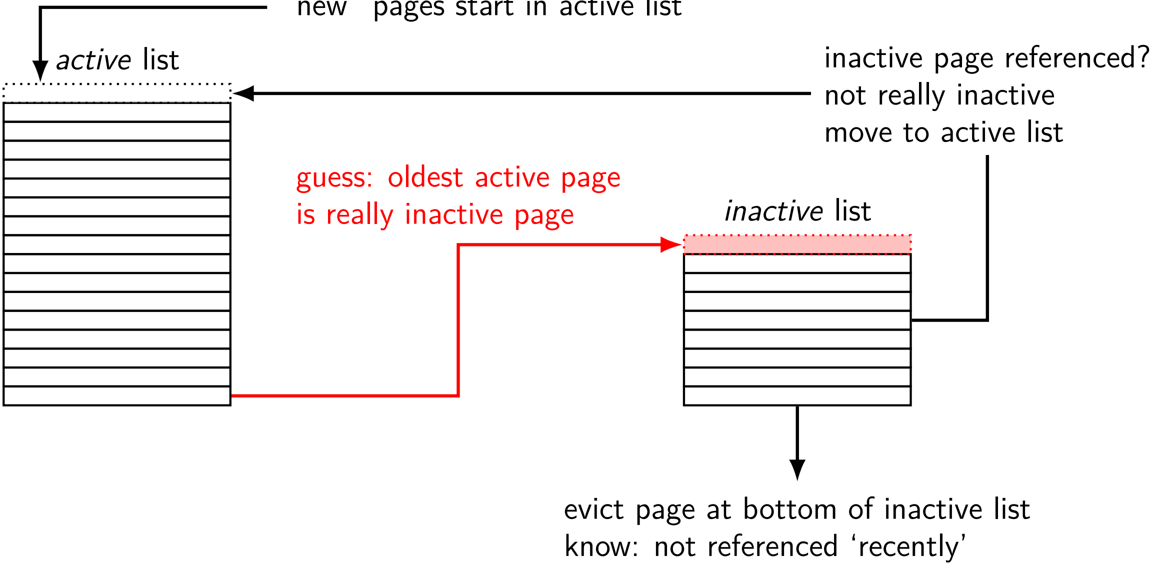
inactive page referenced?
not really inactive
move to active list

guess: oldest active page
is really inactive page

inactive list

evict page at bottom of inactive list
know: not referenced ‘recently’

approximating LRU: SEQ



“new” pages start in active list

active list

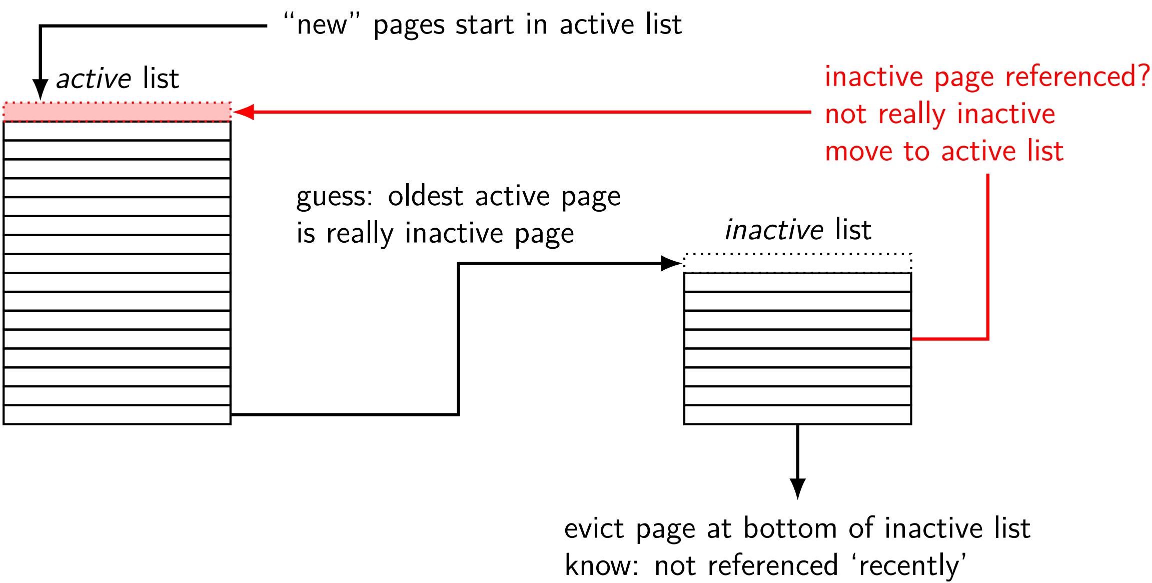
inactive page referenced?
not really inactive
move to active list

guess: oldest active page
is really inactive page

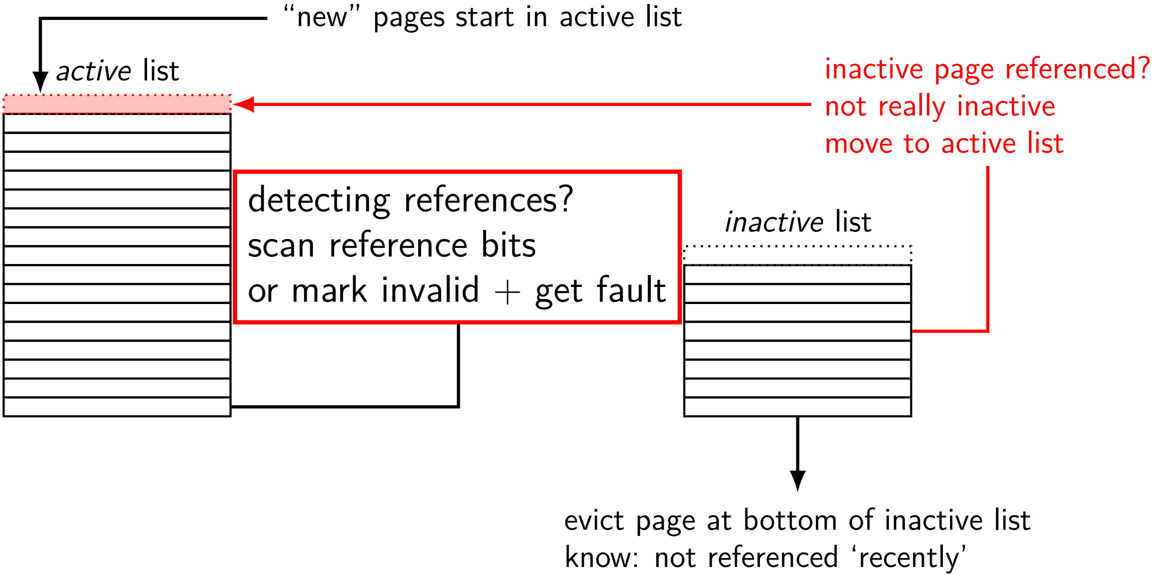
inactive list

evict page at bottom of inactive list
know: not referenced ‘recently’

approximating LRU: SEQ


"new" pages start in active list
active list
inactive page referenced?
not really inactive
move to active list
guess: oldest active page
is really inactive page
inactive list
evict page at bottom of inactive list
know: not referenced 'recently'

approximating LRU: SEQ



“new” pages start in active list

active list

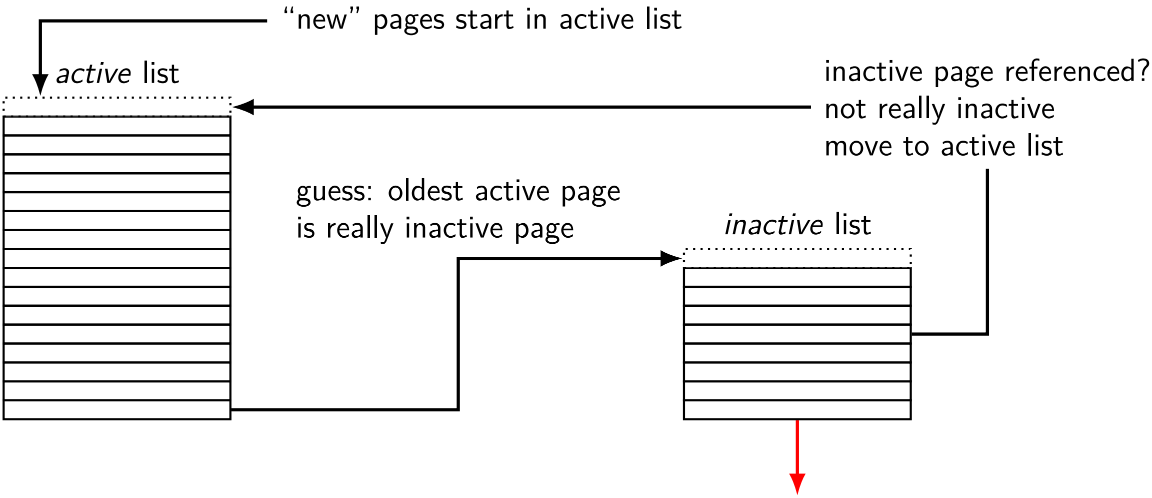
inactive page referenced?
not really inactive
move to active list

detecting references?
scan reference bits
or mark invalid + get fault

inactive list

evict page at bottom of inactive list
know: not referenced ‘recently’

approximating LRU: SEQ



“new” pages start in active list

active list

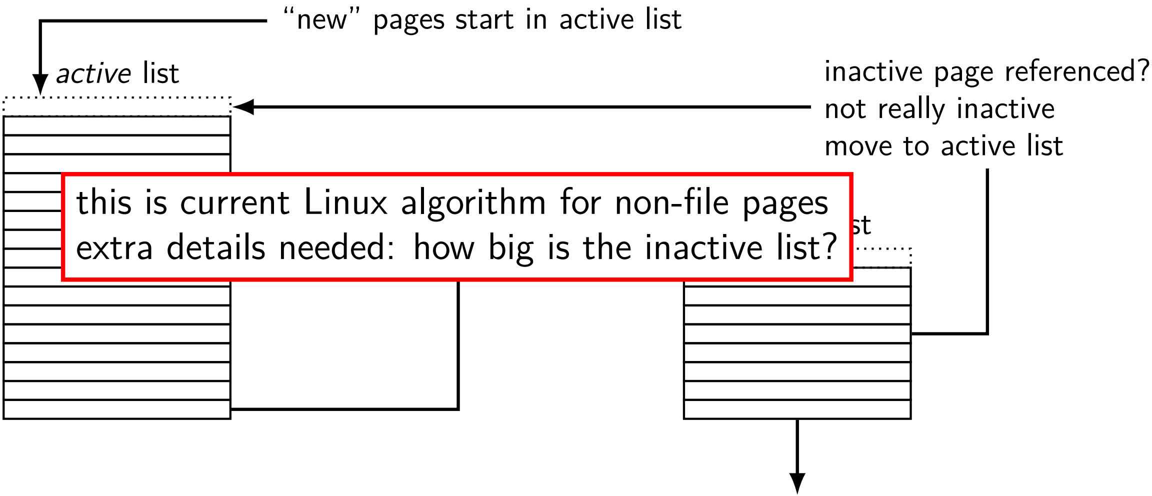
inactive page referenced?
not really inactive
move to active list

guess: oldest active page
is really inactive page

inactive list

evict page at bottom of inactive list
know: not referenced ‘recently’

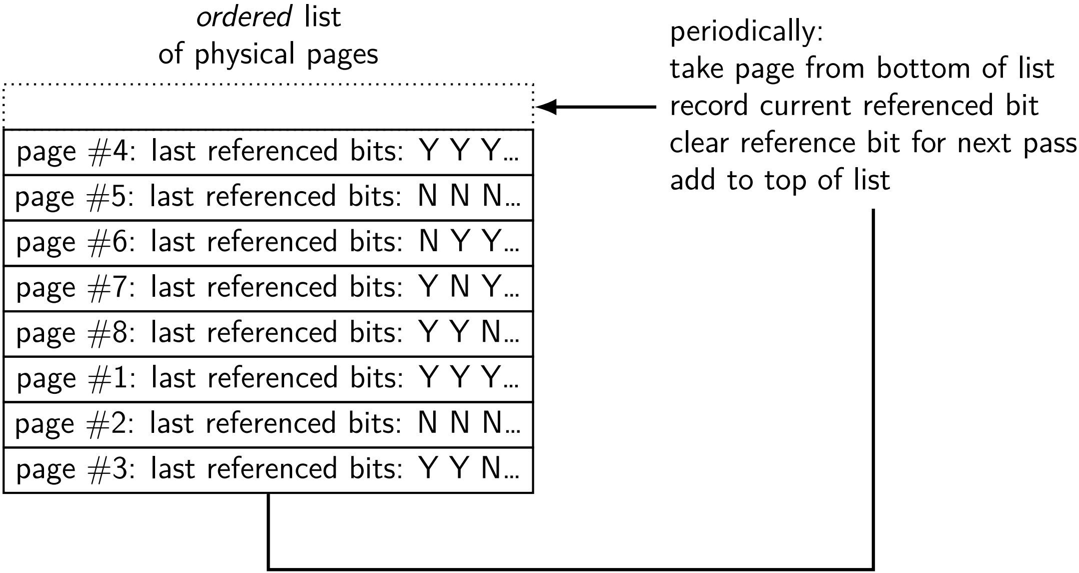
approximating LRU: SEQ



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

evict page at bottom of inactive list
know: not referenced ‘recently’

tracking usage: CLOCK (view 1)



ordered list
of physical pages

page #4: last referenced bits: Y Y Y...

page #5: last referenced bits: N N N...

page #6: last referenced bits: N Y Y...

page #7: last referenced bits: Y N Y...

page #8: last referenced bits: Y Y N...

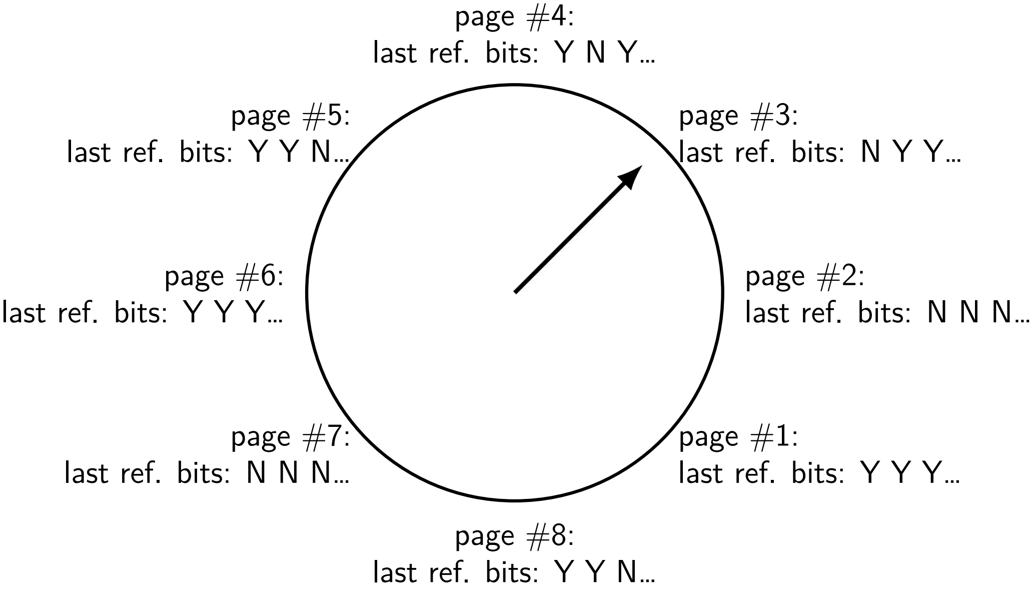
page #1: last referenced bits: Y Y Y...

page #2: last referenced bits: N N N...

page #3: last referenced bits: Y Y N...

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)



page #4:
last ref. bits: Y N Y...

page #5:
last ref. bits: Y Y N...

page #3:
last ref. bits: N Y Y...

page #6:
last ref. bits: Y Y Y...

page #2:
last ref. bits: N N N...

page #7:
last ref. bits: N N N...

page #1:
last ref. bits: Y Y Y...

page #8:
last ref. bits: Y Y N...

backup slides

detecting accesses

non-mmap file reads/writes — modify `read()`/`write()`

otherwise, two options:...

software-only: temporarily set page table entry invalid

- page fault handler record access + sets as valid

hardware assisted: hardware sets *accessed* bit in page table

- OS scans accessed bits later
- reverse mapping can help find page table entries to scan

detecting accesses

non-mmap file reads/writes — modify `read()`/`write()`

otherwise, two options:...

software-only: temporarily set page table entry invalid

- page fault handler record access + sets as valid

hardware assisted: hardware sets *accessed* bit in page table

- OS scans accessed bits later
- reverse mapping can help find page table entries to scan

detecting accesses

non-mmap file reads/writes — modify `read()`/`write()`

otherwise, two options:...

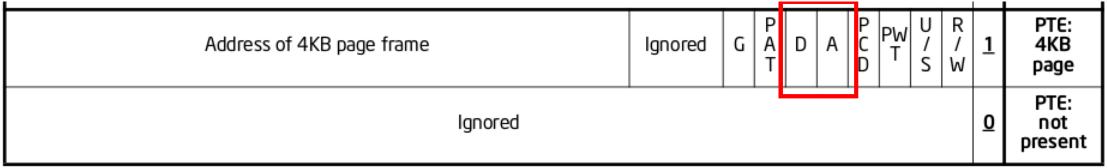
software-only: temporarily set page table entry invalid

- page fault handler record access + sets as valid

hardware assisted: hardware sets *accessed* bit in page table

- OS scans accessed bits later
- reverse mapping can help find page table entries to scan

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

multiple mappings?

page can have many page table entries

- file mmap'd in many processes (e.g. 10 instances of `emacs.exe`)
- copy-on-write pages after fork
- address in kernel memory + address in user memory?
- …

want to check all the accessed bits

aside: detecting write accesses

for updating mmap files/swap want to detect writes

same options as detect accesses in general:

software-only: temporarily set page table entry ***read-only***

- page fault handler records write + sets as writeable

hardware assisted: hardware sets ***dirty*** bit in page table

- OS scans dirty bits later

working set model and phases

what happens when a program changes what it's doing?

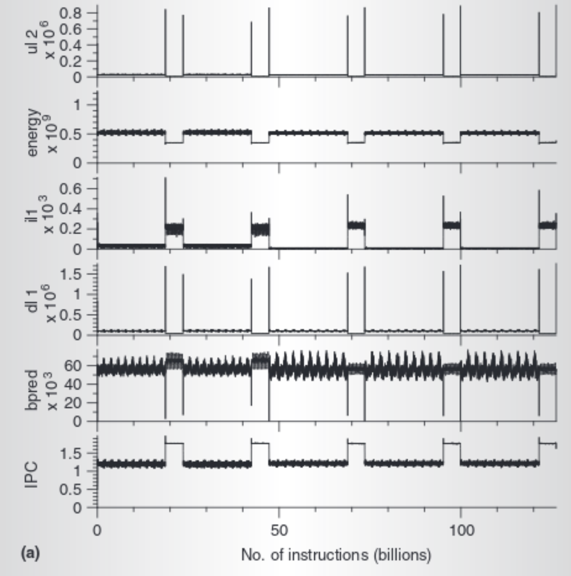
e.g. finish parsing input, now process it

phase change — discard one working set, gain another

phase changes likely to have spike of cache misses

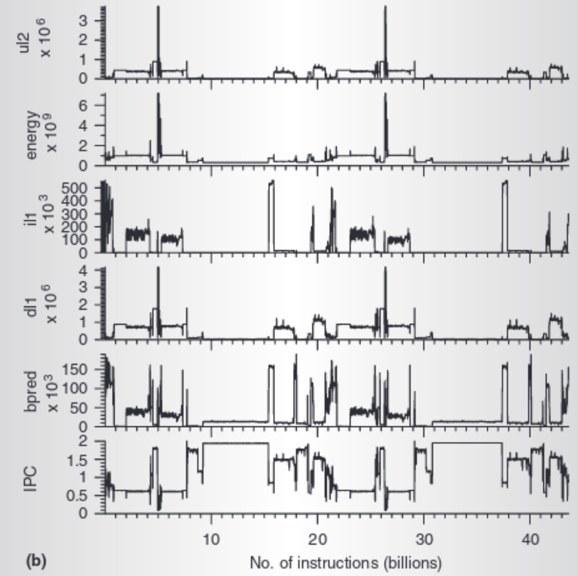
- whatever was cached, not what's being accessed anymore
- maybe along with change in kind of instructions being run

evidence of phases (gzip)



Sherwood et al, "Discovering and Exploiting Program Phases"

evidence of phases (gcc)



Sherwood et al, “Discovering and Exploiting Program Phases”

estimating working sets

working set ≈ what's been used recently

assuming not in phase change...

so, what a program recently used ≈ working set

can use this idea to estimate working set (from list of memory accesses)

using working set estimates

one idea: split memory into *part of working set* or *not*

using working set estimates

one idea: split memory into *part of working set* or *not*

not enough space for all working sets — stop whole program

- maybe a good idea, not done by common consumer/server OSes

using working set estimates

one idea: split memory into *part of working set* or *not*

not enough space for all working sets — stop whole program

- maybe a good idea, not done by common consumer/server OSes

allocating new memory: take from least recently used memory

- = not in a working set
- what most current OS try to do

page fault for every access?

want every access to page fault? make every page invalid

...but want access to happen eventually

...which requires marking page as valid

...which makes future accesses not fault

page fault for every access?

want every access to page fault? make every page invalid

…but want access to happen eventually

…which requires marking page as valid

…which makes future accesses not fault

one solution: use debugging support to run one instruction

- x86: “TF flag”

…then reset pages as invalid

page fault for every access?

want every access to page fault? make every page invalid

...but want access to happen eventually

...which requires marking page as valid

...which makes future accesses not fault

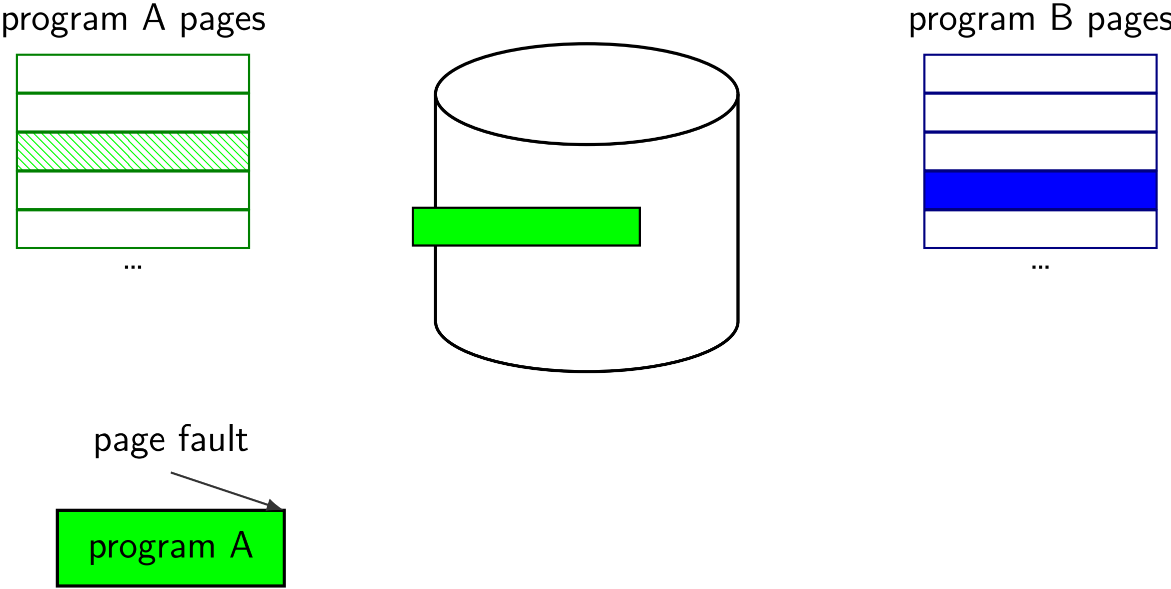
one solution: use debugging support to run one instruction

- x86: “TF flag”

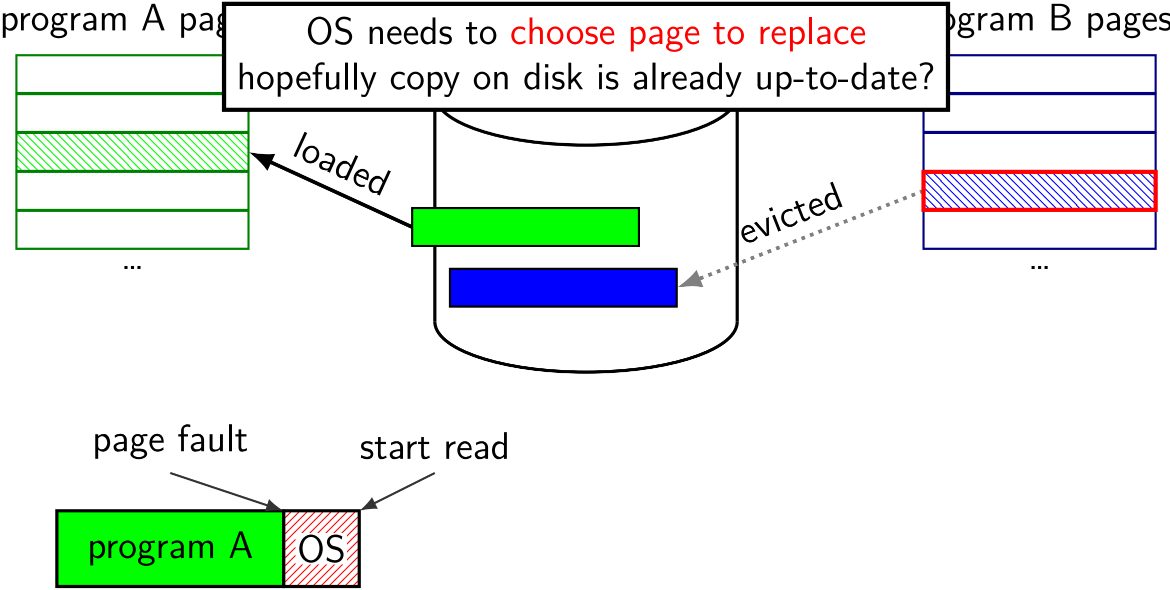
...then reset pages as invalid

okay, so I took something really slow and made it slower

swapping timeline



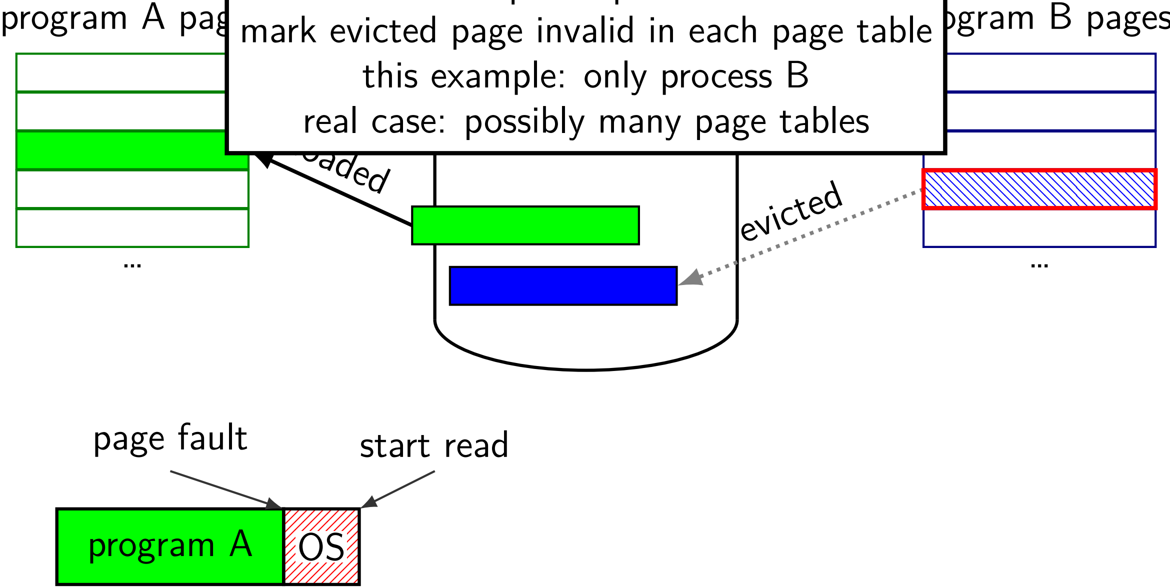
swapping timeline



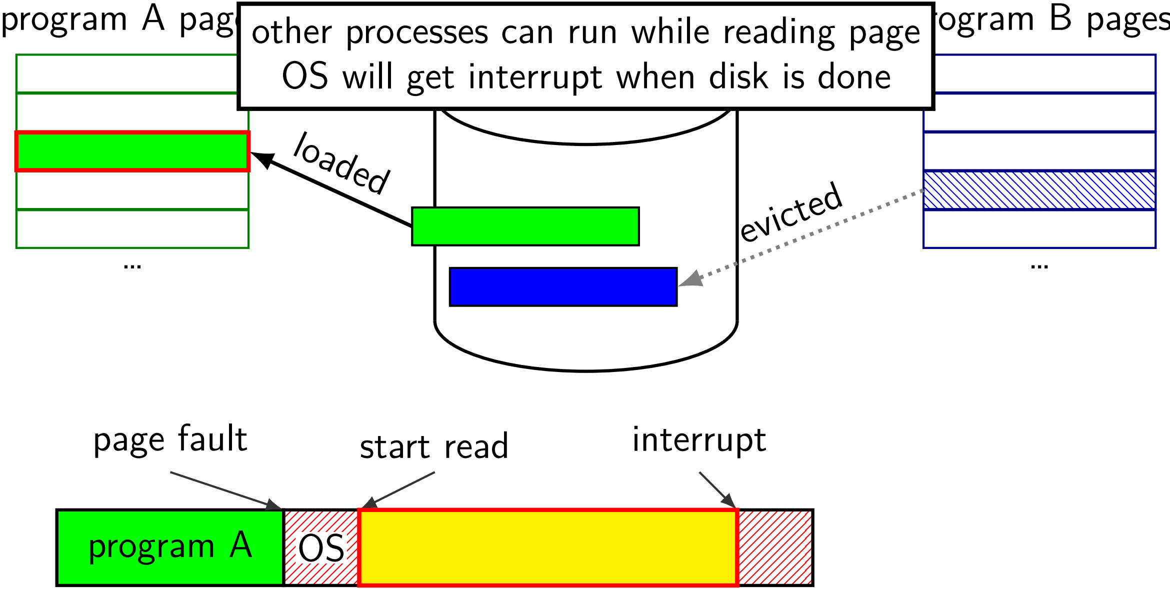
OS needs to choose page to replace
hopefully copy on disk is already up-to-date?

swapping timeline

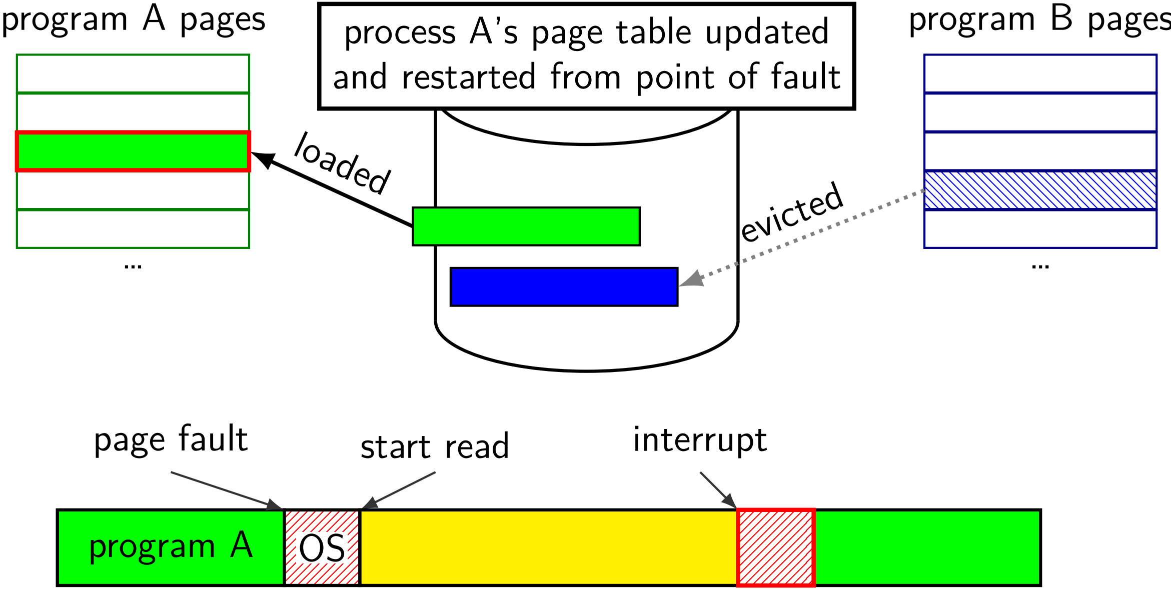
first step of replacement:
mark evicted page invalid in each page table
this example: only process B
real case: possibly many page tables



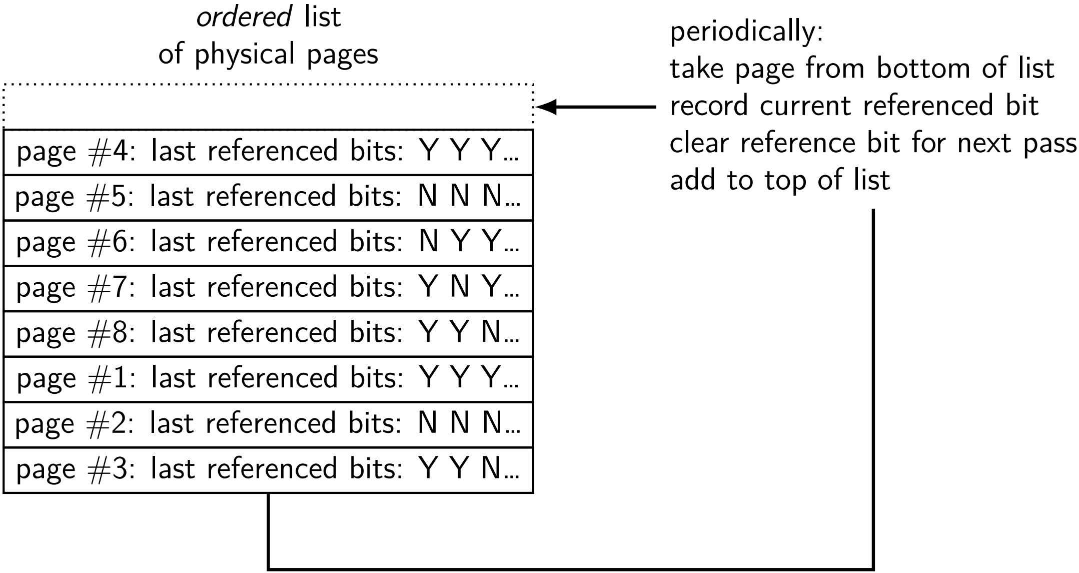
swapping timeline



swapping timeline



tracking usage: CLOCK (view 1)



ordered list
of physical pages

page #4: last referenced bits: Y Y Y...

page #5: last referenced bits: N N N...

page #6: last referenced bits: N Y Y...

page #7: last referenced bits: Y N Y...

page #8: last referenced bits: Y Y N...

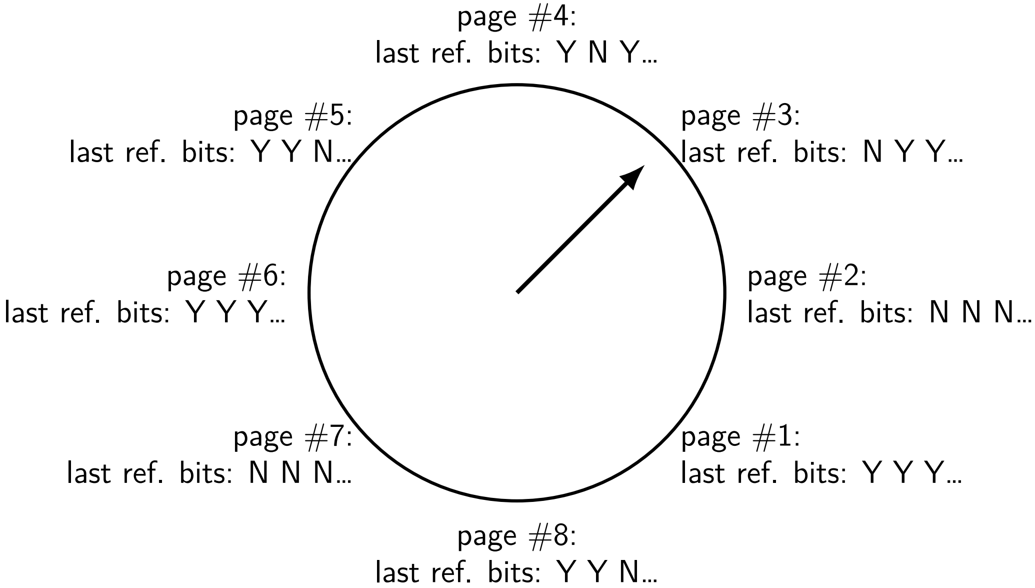
page #1: last referenced bits: Y Y Y...

page #2: last referenced bits: N N N...

page #3: last referenced bits: Y Y N...

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)



page #4:
last ref. bits: Y N Y...

page #5:
last ref. bits: Y Y N...

page #3:
last ref. bits: N Y Y...

page #6:
last ref. bits: Y Y Y...

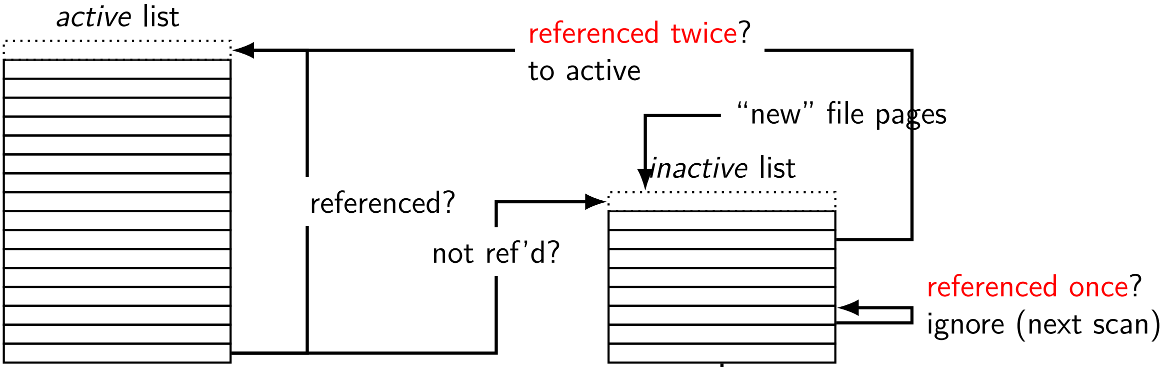
page #2:
last ref. bits: N N N...

page #7:
last ref. bits: N N N...

page #1:
last ref. bits: Y Y Y...

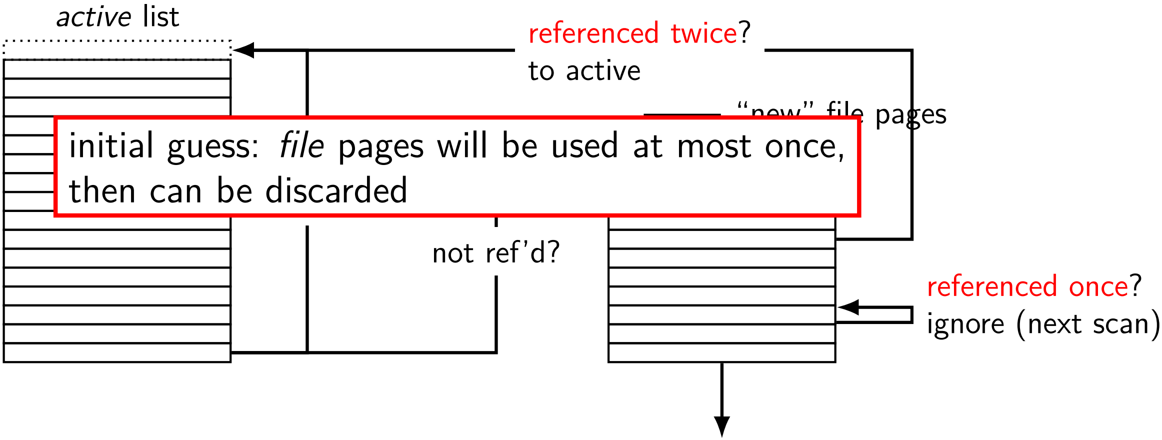
page #8:
last ref. bits: Y Y N...

CLOCK-Pro: special casing for one-use pages



evict page at bottom of inactive list
either file page referenced once *or*
referenced multiple times, but not recently

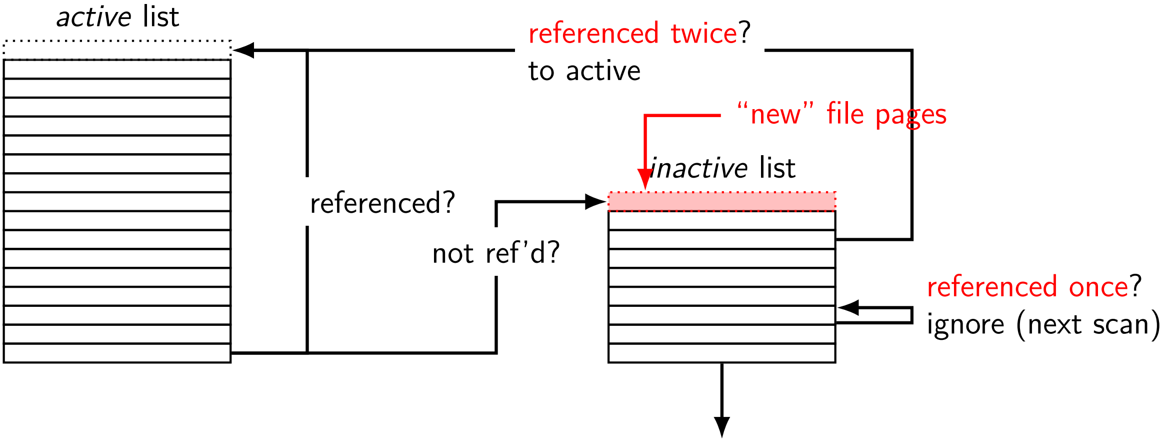
CLOCK-Pro: special casing for one-use pages



initial guess: *file* pages will be used at most once, then can be discarded

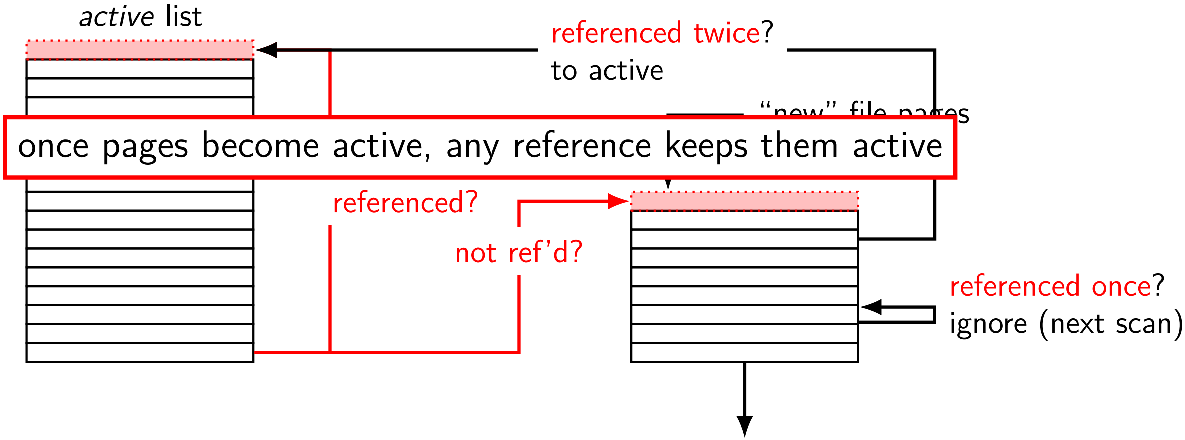
evict page at bottom of inactive list
either file page referenced once *or*
referenced multiple times, but not recently

CLOCK-Pro: special casing for one-use pages



evict page at bottom of inactive list
either file page referenced once *or*
referenced multiple times, but not recently

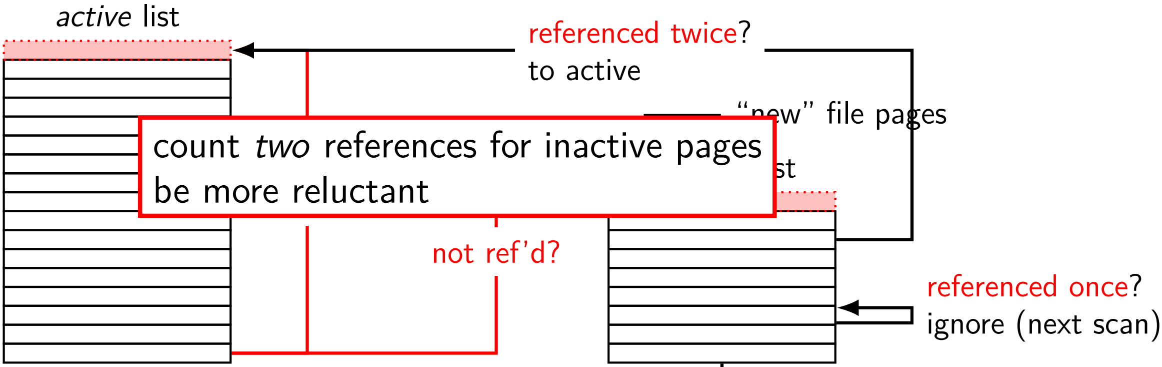
CLOCK-Pro: special casing for one-use pages



once pages become active, any reference keeps them active

evict page at bottom of inactive list
either file page referenced once *or*
referenced multiple times, but not recently

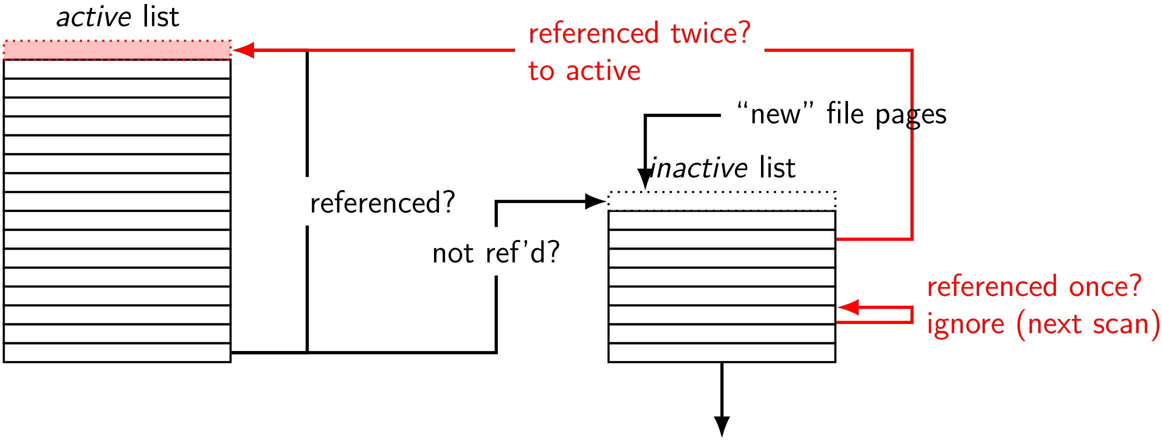
CLOCK-Pro: special casing for one-use pages



count *two* references for inactive pages
be more reluctant

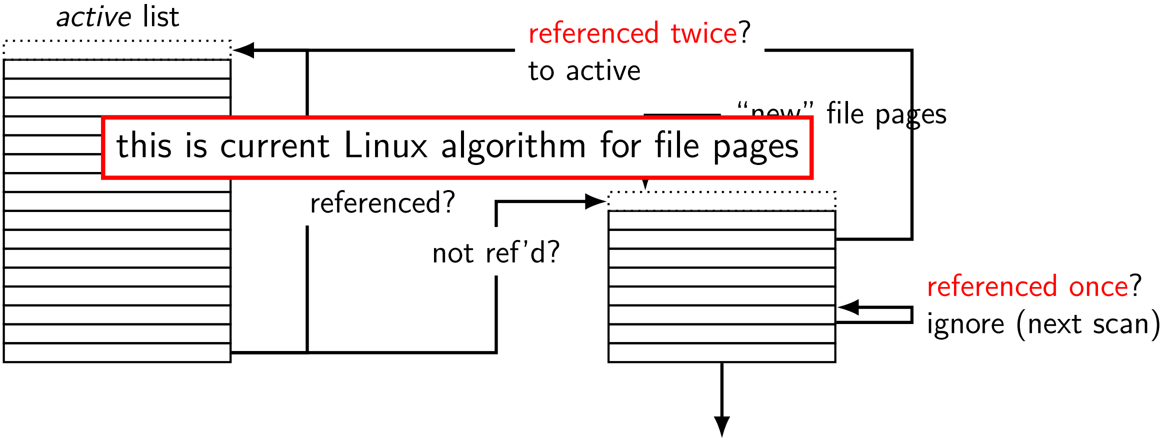
evict page at bottom of inactive list
either file page referenced once *or*
referenced multiple times, but not recently

CLOCK-Pro: special casing for one-use pages



evict page at bottom of inactive list
either file page referenced once *or*
referenced multiple times, but not recently

CLOCK-Pro: special casing for one-use pages



this is current Linux algorithm for file pages

evict page at bottom of inactive list
either file page referenced once *or*
referenced multiple times, but not recently

default Linux page replacement summary

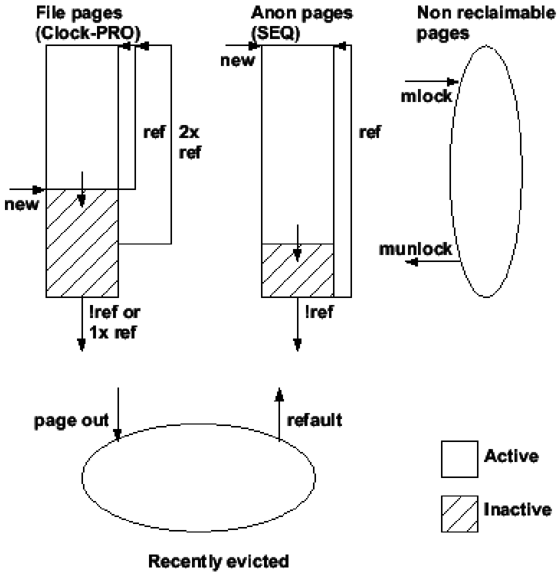


Figure: https://linux-mm.org/PageReplacementDesign

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 → evict that one

- otherwise: give it a second chance

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)

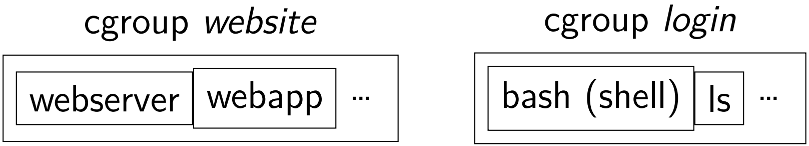
high limit: never let group use more than this

replace pages from this group before anything else

…

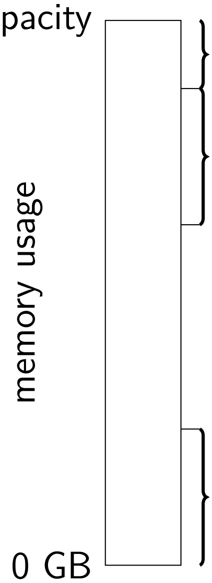
Linux cgroups

Linux mechanism: seperate processes into groups:



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

Linux cgroup memory limits



actively deallocate pages cgroup is using

max

if other processes need memory,
take from this group

high limit

low limit

do not take from this group
for other groups
(even if pages not recently used)

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

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