Changelog

Changes made in this version not seen in first lecture:

- 19 March 2019: tmeporarily invalid PTE (software support): correct PPN in “OS page info” being a VPN instead

virtual memory 3: page cache / page replacement

last time

page table tricks

- allocate on demand
- copy on write

mapping files — `mmap`

- Linux: process memory is a list of maps
- maps may or may not correspond to file
- either private (copy on write) or shared (actually modify file)

page cache

- everything potentially in memory has location on disk
- for files: location is in the file
- for everything else: allocate disk space ("swap space")
- goal: manage memory as a cache of stuff on disk
- fully associative: all physical memory pages used for anything

the page cache

memory is a cache for disk

files, program memory has a place on disk

- running low on memory? always have room on disk
- assumption: disk space approximately infinite

physical memory pages: disk 'temporarily' kept in faster storage

- possibly being used by one or more processes?
- possibly part of a file on disk?
- possibly both

goal: manage this cache intelligently

the page cache

memory is a cache for disk

files, program memory has a place on disk
- running low on memory? always have room on disk
- assumption: disk space approximately infinite

physical memory pages: disk 'temporarily' kept in faster storage
- possibly being used by one or more processes?
- possibly part of a file on disk?
- possibly both

goal: manage this cache intelligently

memory as a cache for disk

"cache block" $\approx$ physical page

fully associative

- any virtual address/file part can be stored in any physical page

replacement is managed by the OS

normal cache hits happen without OS

- common case that needs to be fast

page cache components [text]

mapping: virtual address or file+offset → physical page

- handle cache hits

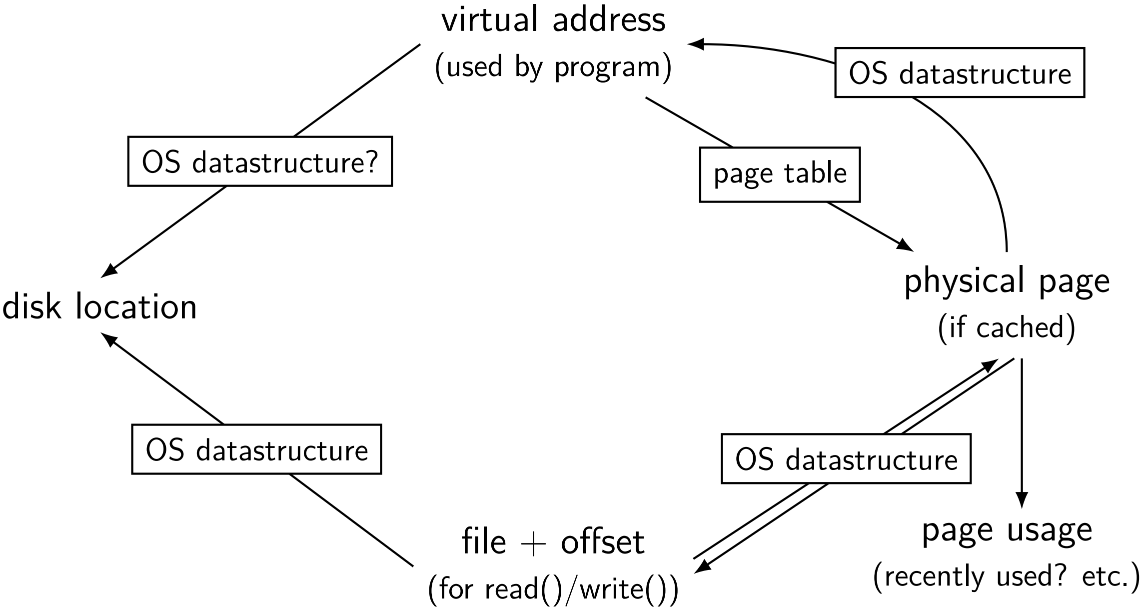
find backing location based on virtual address/file+offset

- handle cache misses

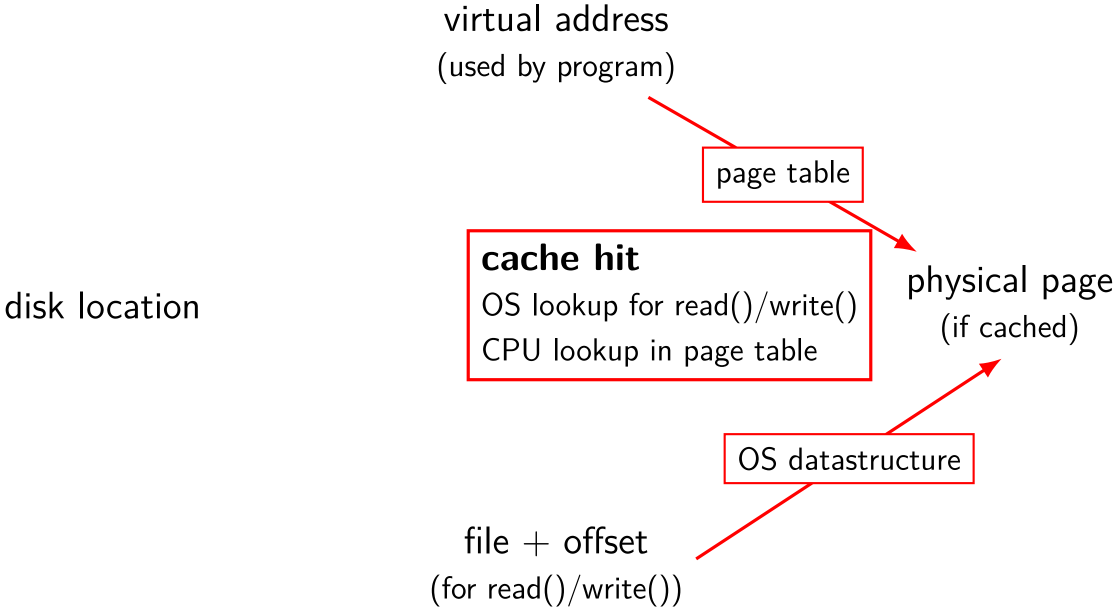
track information about each physical page

- handle page allocation
- handle cache eviction

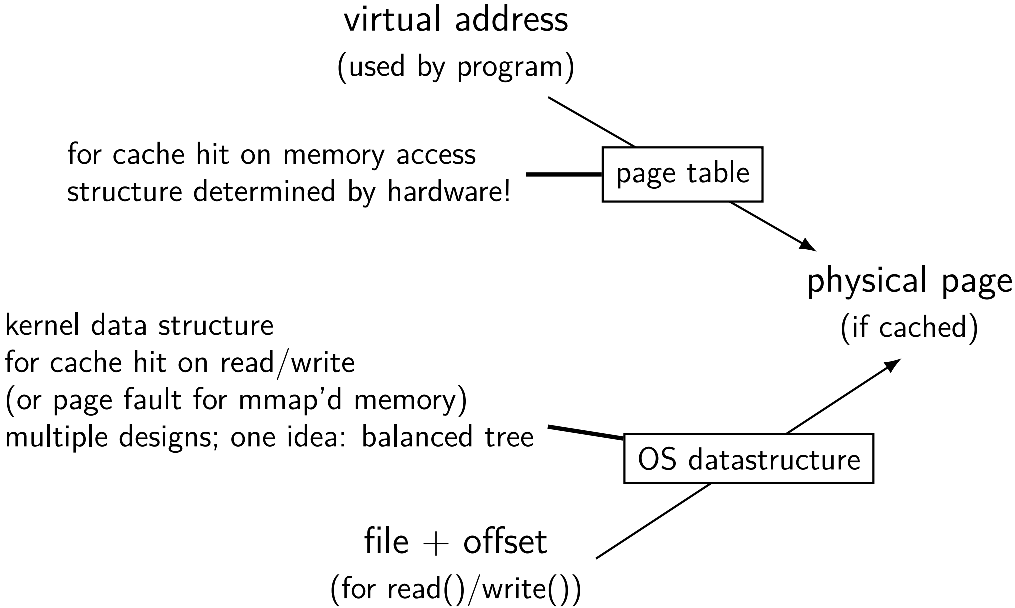
page cache components



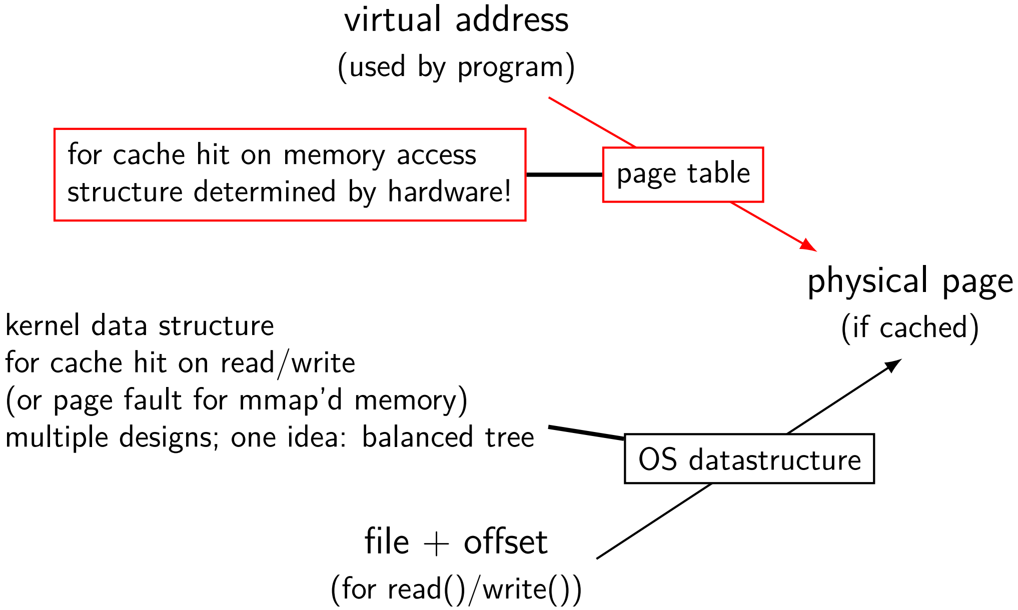
page cache components



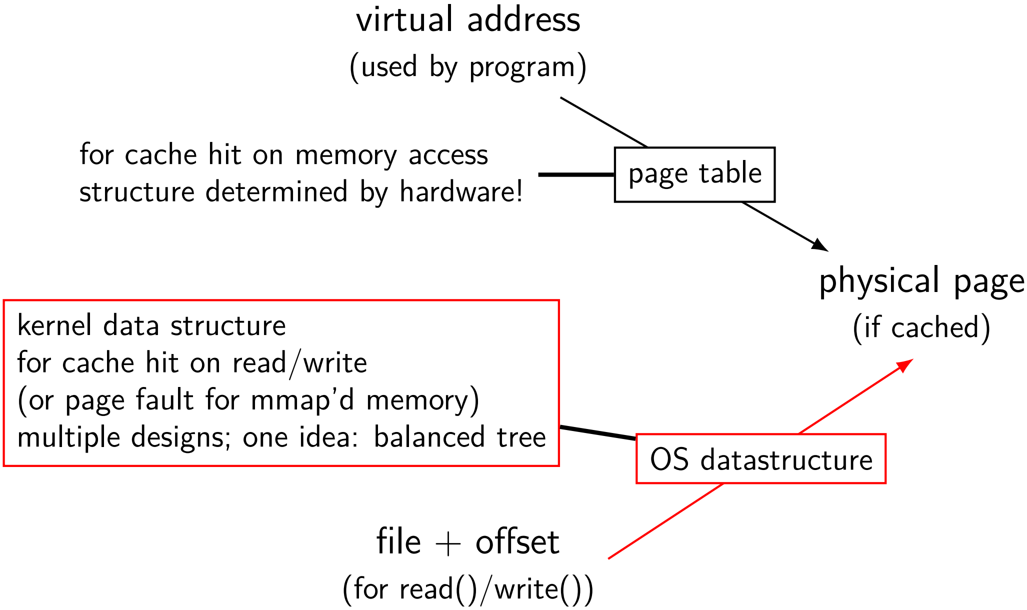
virtual addr/file offset to physical page



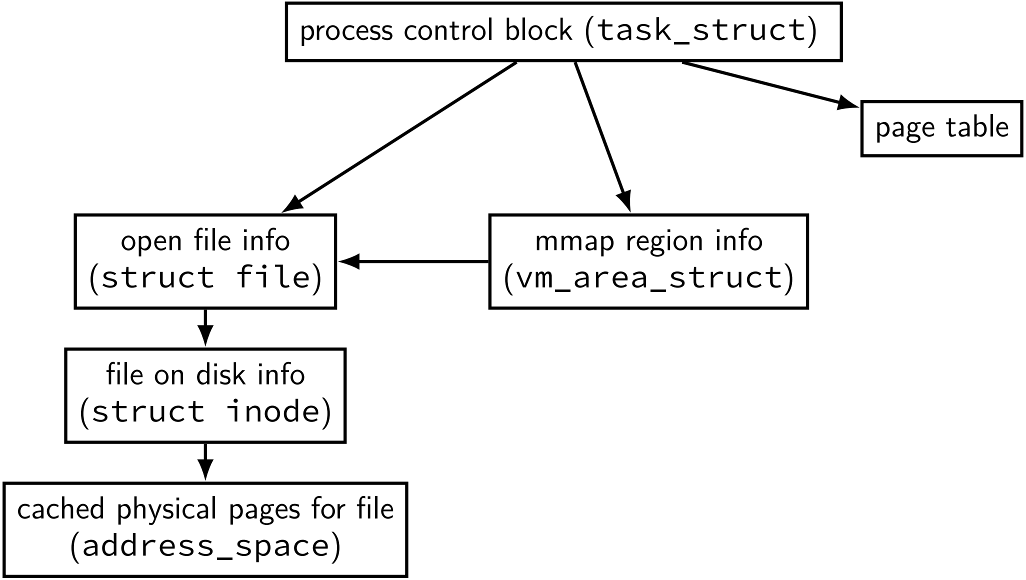
virtual addr/file offset to physical page



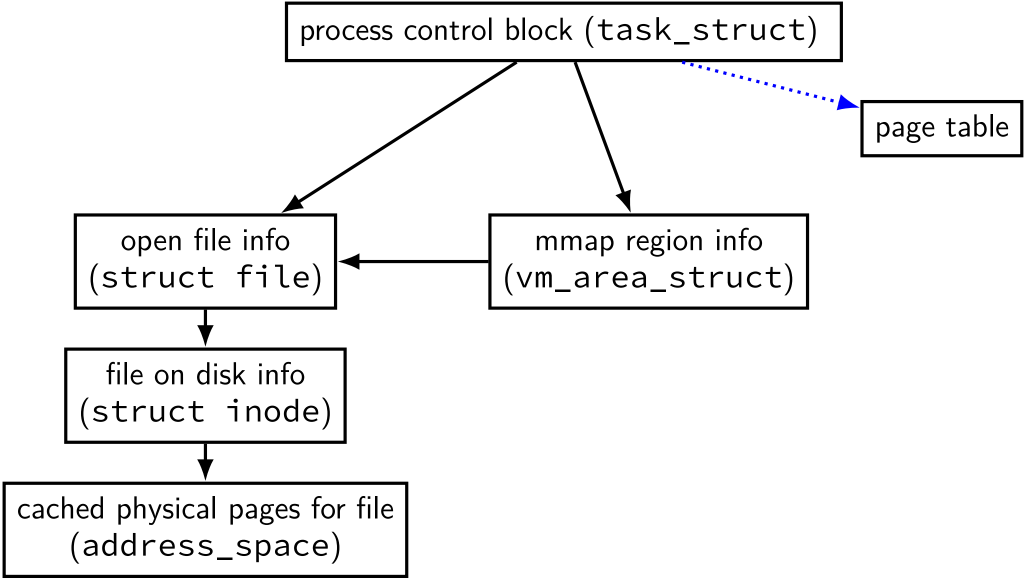
virtual addr/file offset to physical page



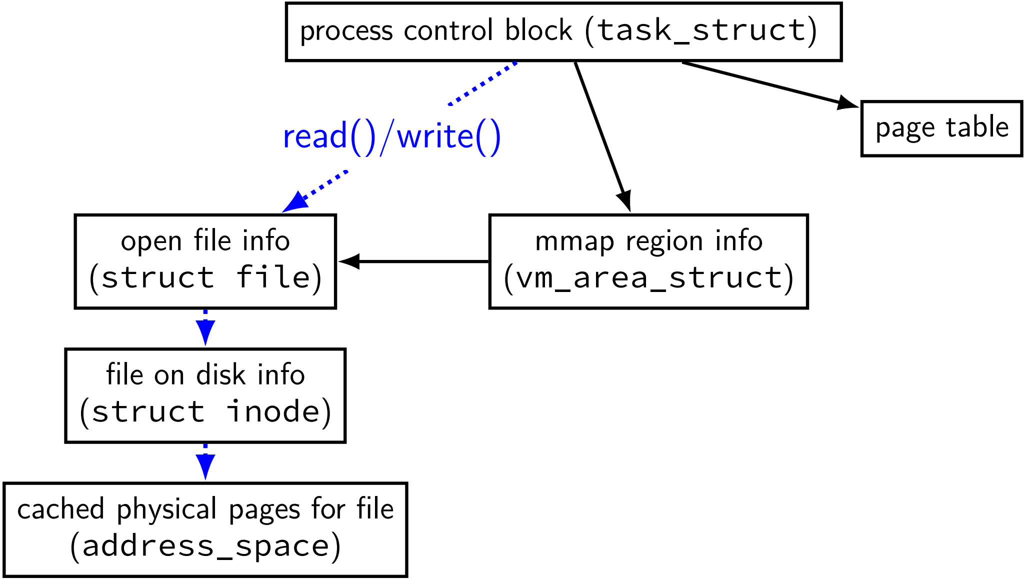
Linux: forward mapping



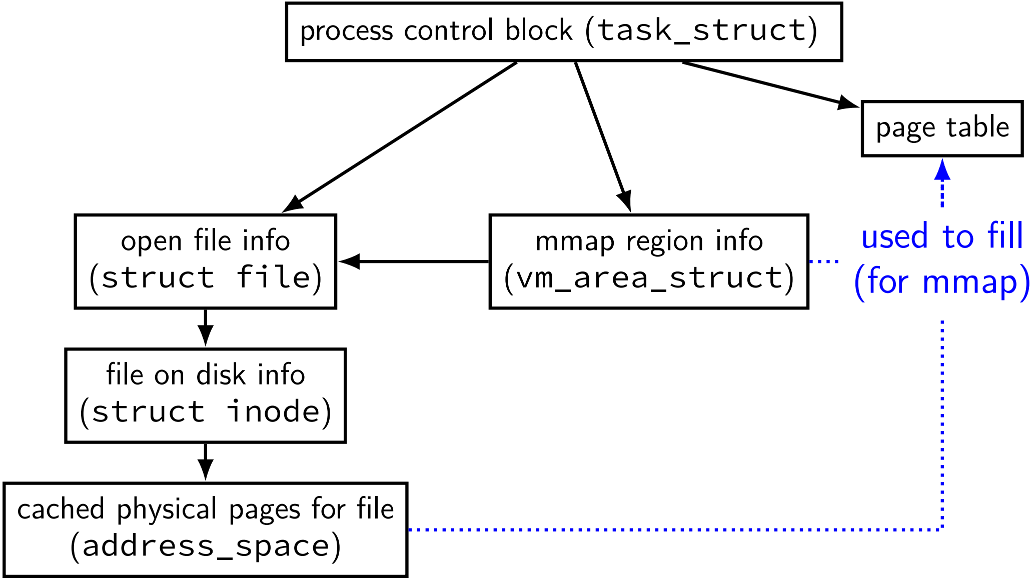
Linux: forward mapping



Linux: forward mapping



Linux: forward mapping



minor and major faults

minor page fault

- page is already in page cache
- just fill in page table entry

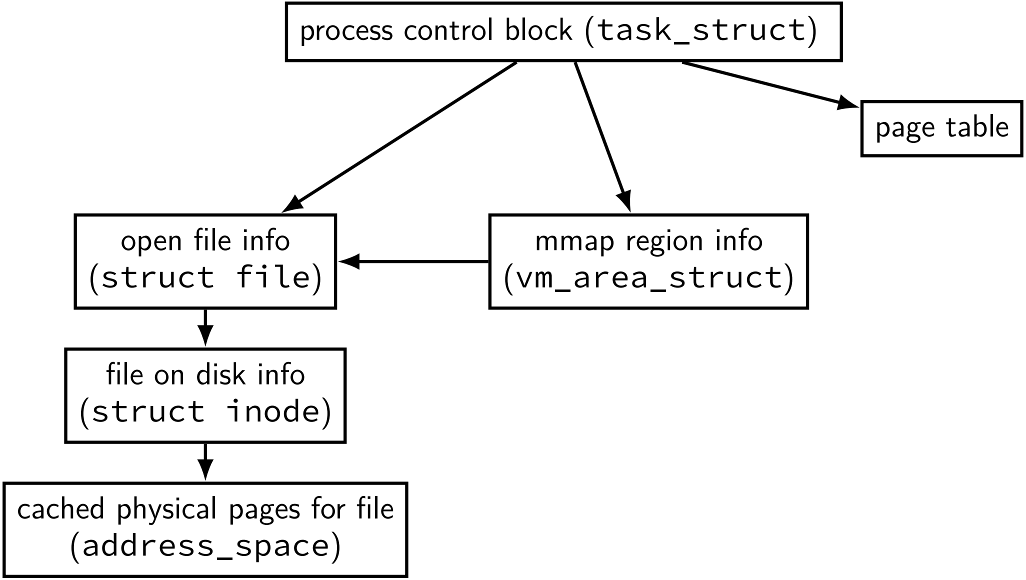
major page fault

- page not cached, need to allocate

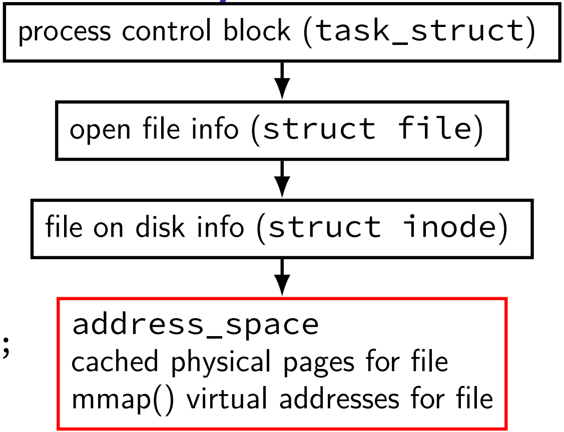
Linux: reporting minor/major faults

```
$ /usr/bin/time --verbose some-command
        Command being timed: "some-command"
        User time (seconds): 18.15
        System time (seconds): 0.35
        Percent of CPU this job got: 94%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:19.57
...
        Maximum resident set size (kbytes): 749820
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 230166
        Voluntary context switches: 1423
        Involuntary context switches: 53
        Swaps: 0
...
        Exit status: 0
```

Linux: forward mapping

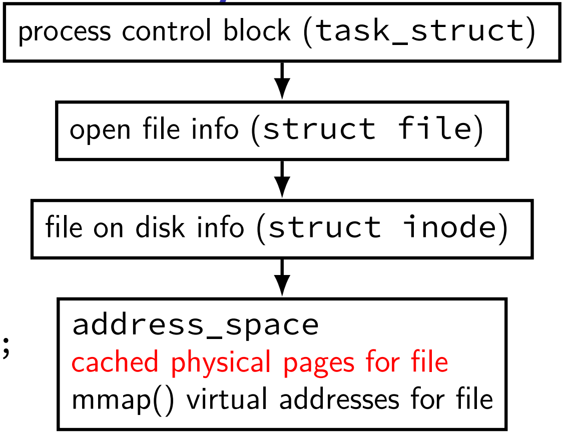


Linux: tracking files in memory



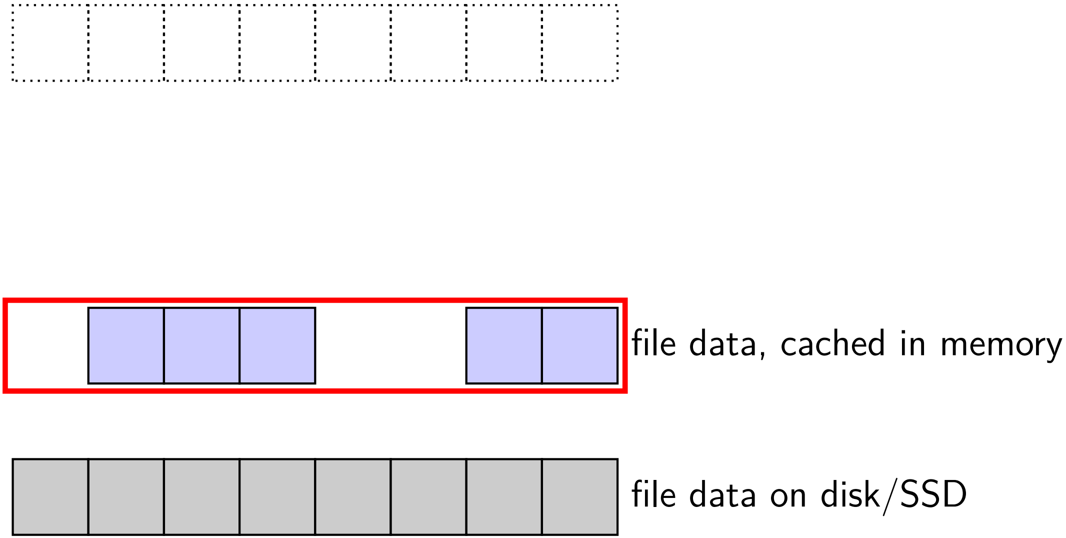

```
struct file {
    ...
    struct inode *f_inode;
    ...
};
...
struct inode {
    ...
    struct address_space i_data;
    ...
};
...
struct address_space {
    ...
    struct radix_tree_root  i_pages;          /* cached pages */
    atomic_t                i_mmap_writable;/* count VM_SHARED mappi
    struct rb_root_cached   i_mmap;           /* tree of private and s
    ...
```

Linux: tracking files in memory

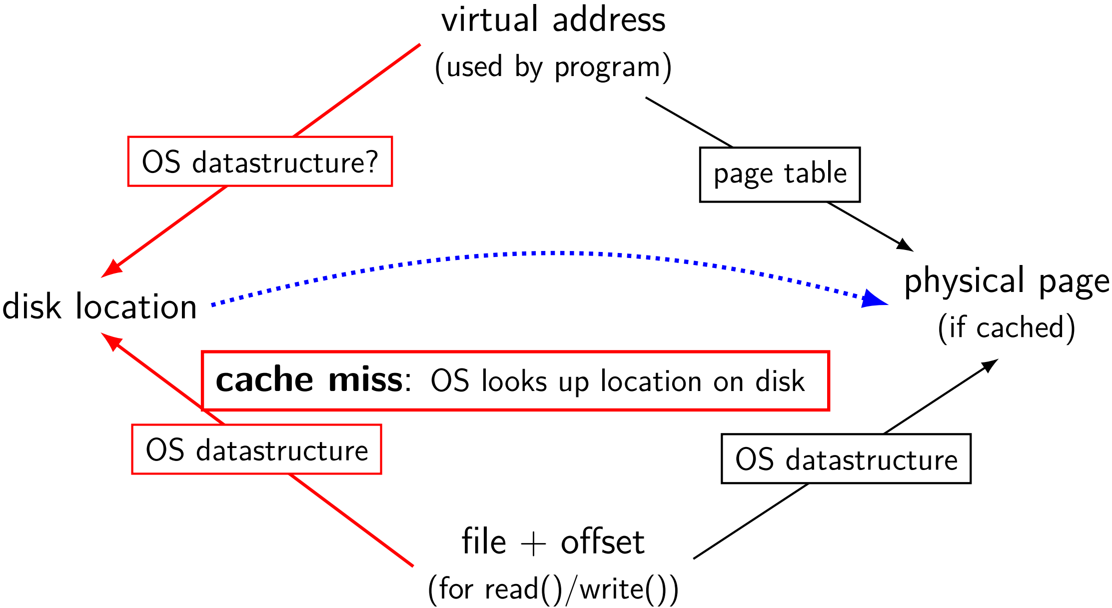



```
struct file {
    ...
    struct inode *f_inode;
    ...
};
...
struct inode {
    ...
    struct address_space i_data;
    ...
};
...
struct address_space {
    ...
    struct radix_tree_root  i_pages;        /* cached pages */
    atomic_t                i_mmap_writable;/* count VM_SHARED mappi
    struct rb_root_cached   i_mmap;         /* tree of private and s
    ...
```

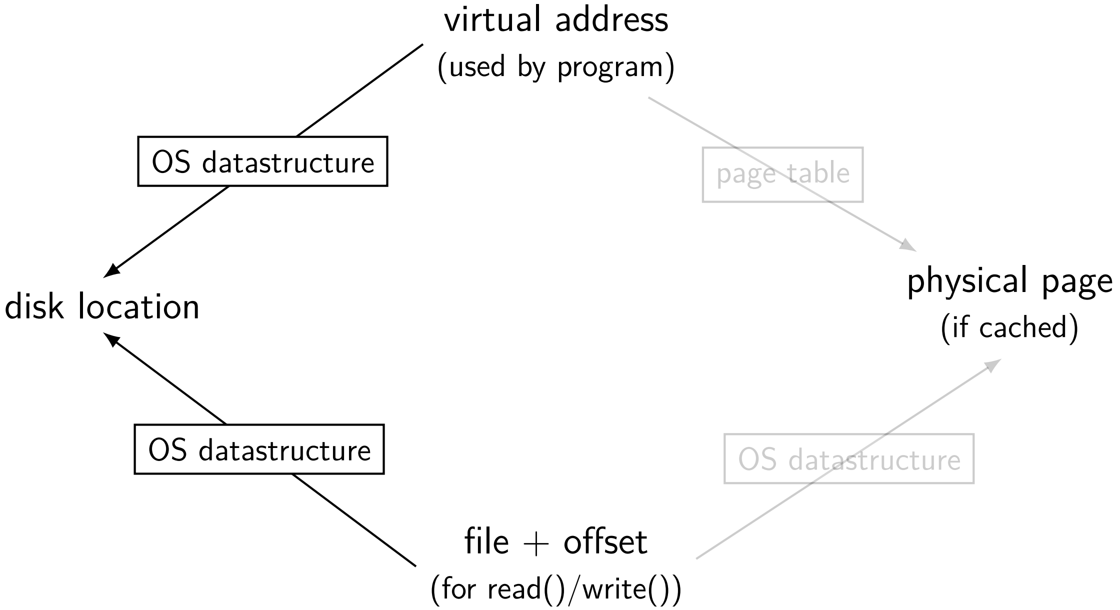
mapped pages (read/write, shared)



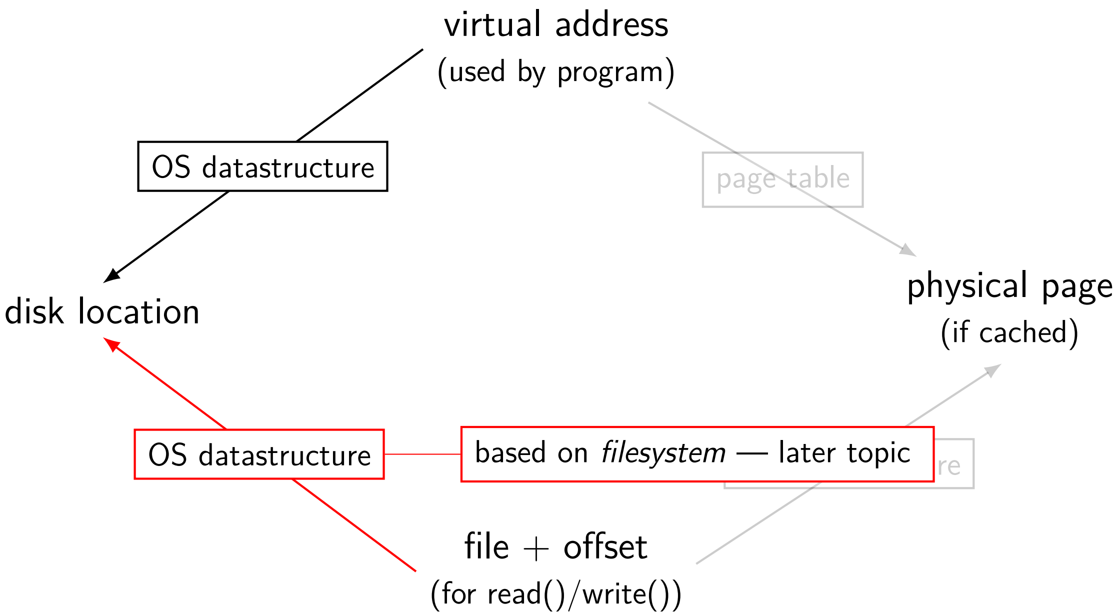
page cache components



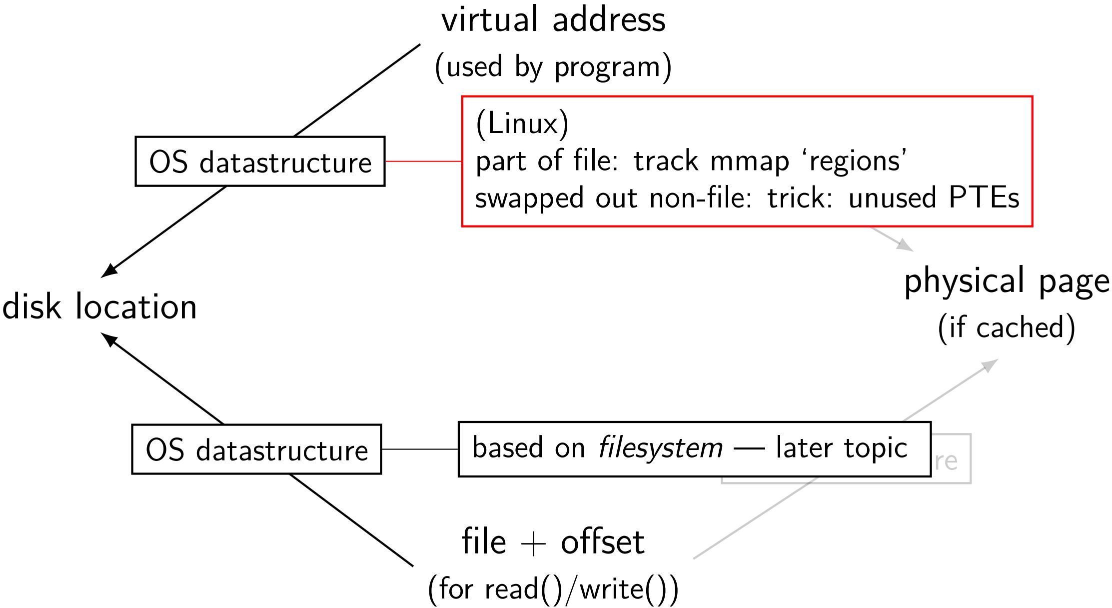
virtual address/file offset → location on disk



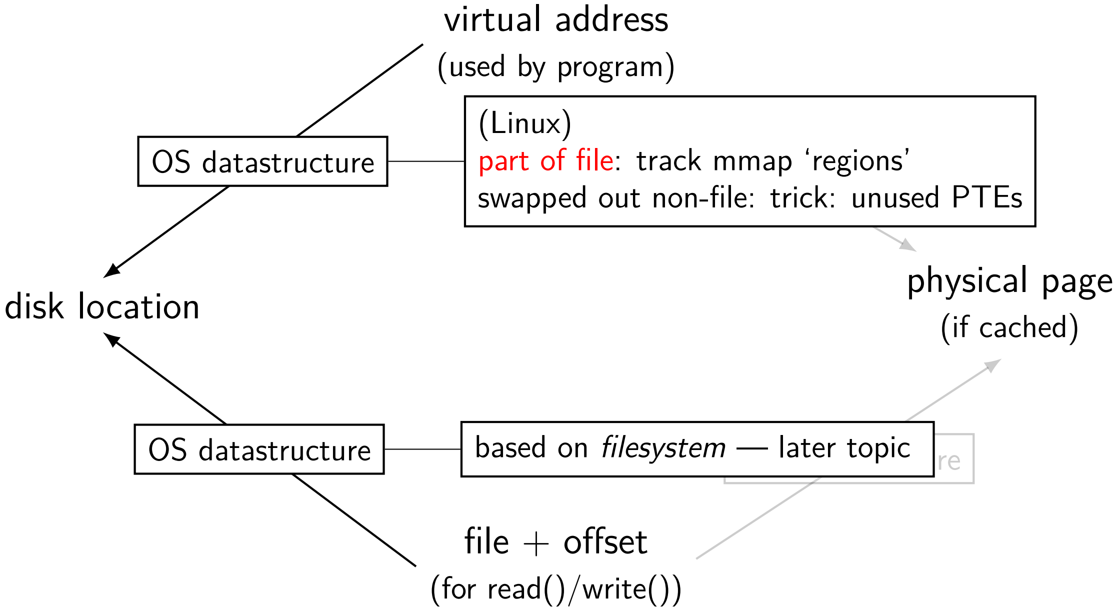
virtual address/file offset → location on disk



virtual address/file offset → location on disk



virtual address/file offset → location on disk



recall: Linux maps

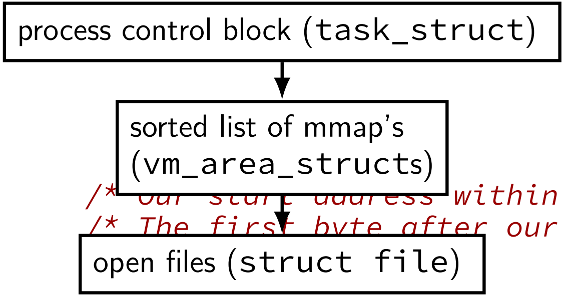
```
$ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 08:01 48328831          /bin/cat
0060a000-0060b000 r--p 0000a000 08:01 48328831          /bin/cat
0060b000-0060c000 rw-p 0000b000 08:01 48328831          /bin/cat
01974000-01995000 rw-p 00000000 00:00 0                 [heap]
7f60c718b000-7f60c7490000 r--p 00000000 08:01 77483660  /usr/lib/locale/locale-archive
7f60c7490000-7f60c764e000 r-xp 00000000 08:01 96659129  /lib/x86_64-linux-gnu/libc-2.19
7f60c764e000-7f60c784e000 ---p 001be000 08:01 96659129  /lib/x86_64-linux-gnu/libc-2.19
7f60c784e000-7f60c7852000 r--p 001be000 08:01 96659129  /lib/x86_64-linux-gnu/libc-2.19
7f60c7852000-7f60c7854000 rw-p 001c2000 08:01 96659129  /lib/x86_64-linux-gnu/libc-2.19
7f60c7854000-7f60c7859000 rw-p 00000000 00:00 0
7f60c7859000-7f60c787c000 r-xp 00000000 08:01 96659109  /lib/x86_64-linux-gnu/ld-2.19.so
7f60c7a39000-7f60c7a3b000 rw-p 00000000 00:00 0
7f60c7a7a000-7f60c7a7b000 rw-p 00000000 00:00 0
7f60c7a7b000-7f60c7a7c000 r--p 00022000 08:01 96659109  /lib/x86_64-linux-gnu/ld-2.19.so
7f60c7a7c000-7f60c7a7d000 rw-p 00023000 08:01 96659109  /lib/x86_64-linux-gnu/ld-2.19.so
7f60c7a7d000-7f60c7a7e000 rw-p 00000000 00:00 0
7ffc5d2b2000-7ffc5d2d3000 rw-p 00000000 00:00 0          [stack]
7ffc5d3b0000-7ffc5d3b3000 r--p 00000000 00:00 0          [vvar]
7ffc5d3b3000-7ffc5d3b5000 r-xp 00000000 00:00 0          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0  [vsyscall]
```

Linux: tracking memory regions

```
struct vm_area_struct { ...
    unsigned long vm_start;        /* Our start address within vm_m
    unsigned long vm_end;          /* The first byte after our end
                                      within vm_mm. */

    ...
    pgprot_t vm_page_prot;         /* Access permissions of this VM
    unsigned long vm_flags;        /* Flags, see mm.h. */
    ...
    struct anon_vma *anon_vma;     /* Serialized by page_table_lock
    ...
    unsigned long vm_pgoff;        /* Offset (within vm_file) in PA
                                      units */
    struct file * vm_file;         /* File we map to (can be NULL).
    ...
}   randomize_layout;
```

Linux: tracking memory regions




```
struct vm_area_struct { ...
    unsigned long vm_start;        /* Our start address within vm_m
    unsigned long vm_end;          /* The first byte after our end

    ...
    pgprot_t vm_page_prot;         /* Access permissions of this VM
    unsigned long vm_flags;        /* Flags, see mm.h. */
    ...
    struct anon_vma *anon_vma;     /* Serialized by page_table_lock
    ...
    unsigned long vm_pgoff;        /* Offset (within vm_file) in PA
                                      units */
    struct file * vm_file;         /* File we map to (can be NULL).
    ...
} randomize_layout;
```

Linux: tracking memory regions

virtual addresses of mapping
mapping are part of sorted list/tree
to allow finding by start/end address

```
struct vm_area_struct { ...
    unsigned long vm_start;          /* Our start address within vm_m
    unsigned long vm_end;            /* The first byte after our end
                                        within vm_mm. */

    ...
    pgprot_t vm_page_prot;           /* Access permissions of this VM
    unsigned long vm_flags;          /* Flags, see mm.h. */
    ...
    struct anon_vma *anon_vma;       /* Serialized by page_table_lock
    ...
    unsigned long vm_pgoff;          /* Offset (within vm_file) in PA
                                        units */
    struct file * vm_file;           /* File we map to (can be NULL).
    ...
} randomize_layout;
```

Linux: tracking memory regions

permissions (read/write/execute)

```
struct vm_area_struct { ...
    unsigned long vm_start;          /* Our start address within vm_m
    unsigned long vm_end;            /* The first byte after our end
                                        within vm_mm. */
    ...
    pgprot_t vm_page_prot;           /* Access permissions of this VM
    unsigned long vm_flags;          /* Flags, see mm.h. */
    ...
    struct anon_vma *anon_vma;       /* Serialized by page_table_lock
    ...
    unsigned long vm_pgoff;          /* Offset (within vm_file) in PA
                                        units */
    struct file * vm_file;           /* File we map to (can be NULL).
    ...
} __randomize_layout;
```

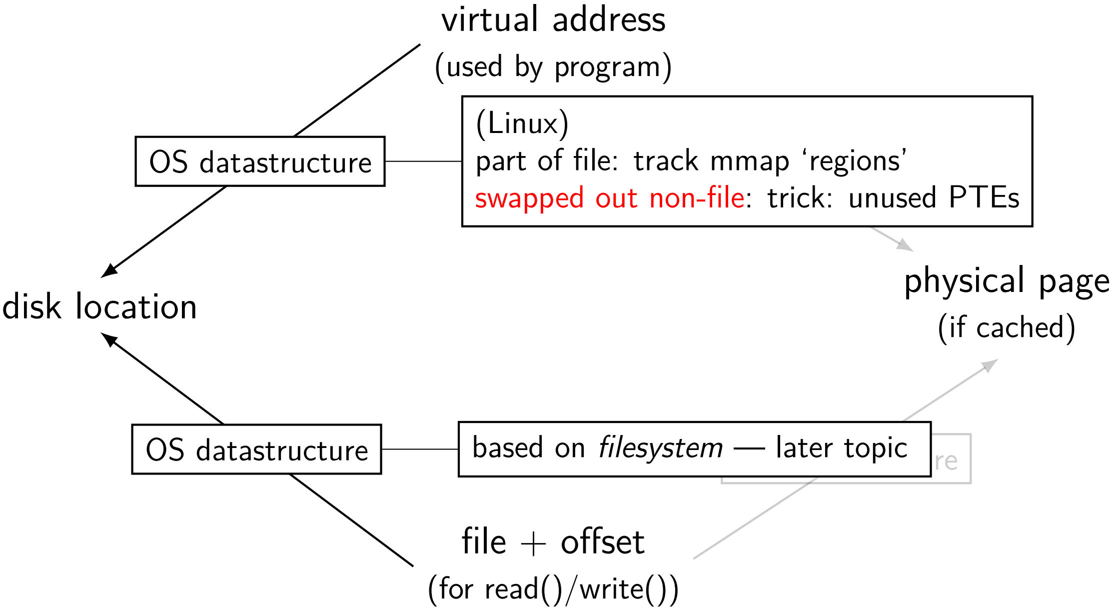
Linux: tracking memory regions

flags: private or shared? ...
private = copy-on-write
shared = make changes to underlying file

```
struct vm_area_struct { ...
    unsigned long vm_start;         /* Our start address within vm_m
    unsigned long vm_end;           /* The first byte after our end
                                       within vm_mm. */

    ...
    pgprot_t vm_page_prot;          /* Access permissions of this VM
    unsigned long vm_flags;         /* Flags, see mm.h. */
    ...
    struct anon_vma *anon_vma;      /* Serialized by page_table_lock
    ...
    unsigned long vm_pgoff;         /* Offset (within vm_file) in PA
                                       units */
    struct file * vm_file;          /* File we map to (can be NULL).
    ...
} randomize_layout;
```

virtual address/file offset → location on disk



Linux: tracking swapped out pages

need to lookup location on disk

potentially one location for every virtual page

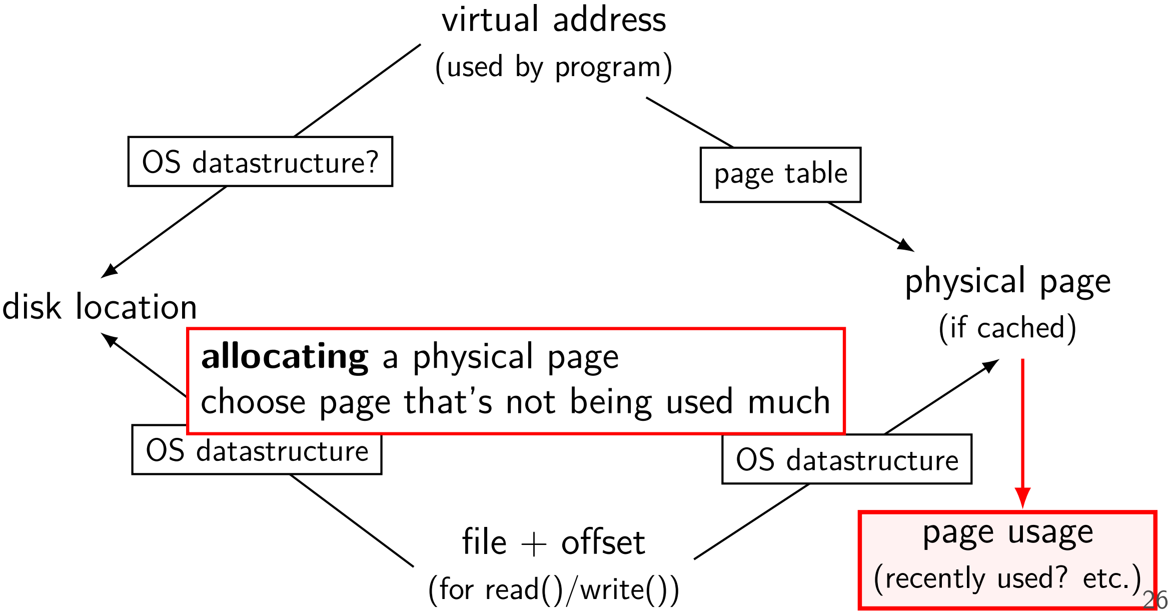
trick: store location in "ignored" part of page table entry

- instead of physical page #, permission bits, etc., store offset on disk

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

page cache components



tracking physical pages: finding free pages

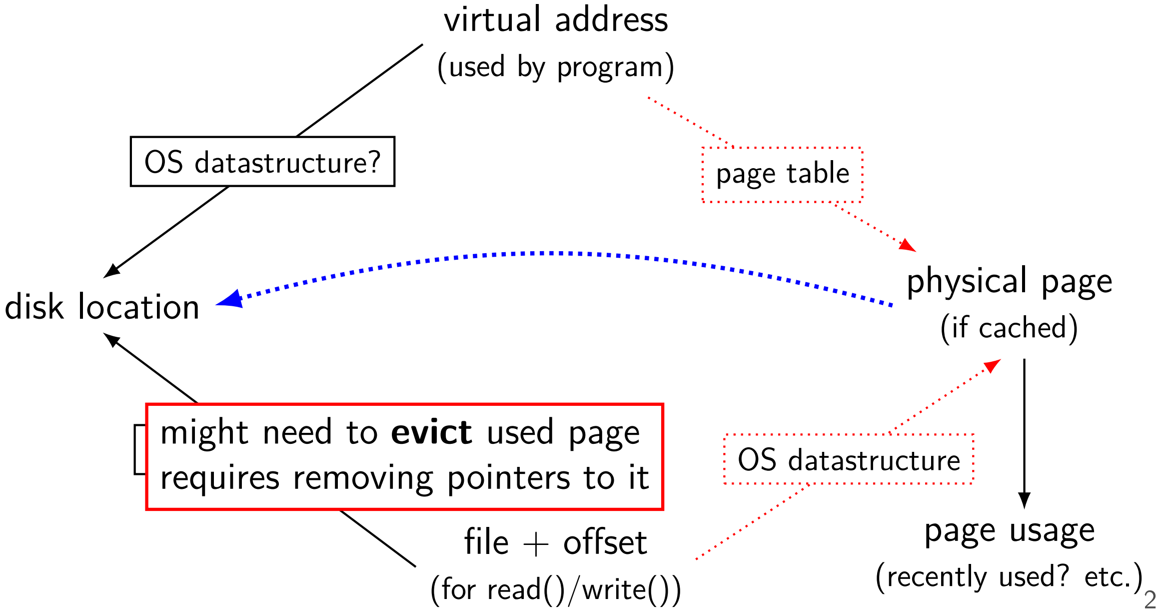
Linux has list of "least recently used" pages:

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

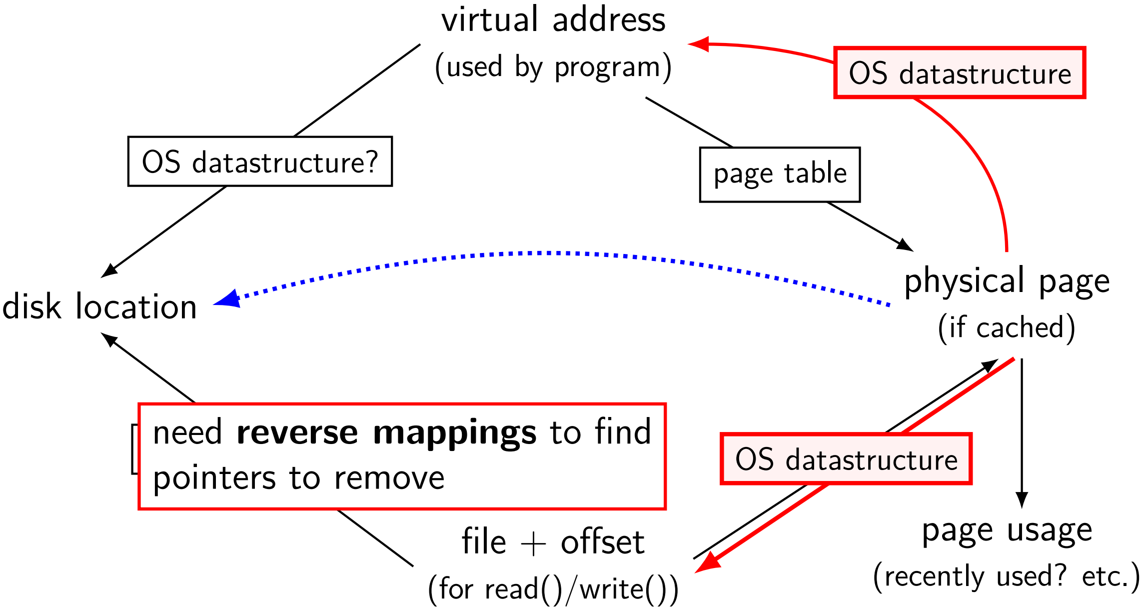
how we're going to find a page to allocate
(and evict from something else)

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

page cache components



page cache components

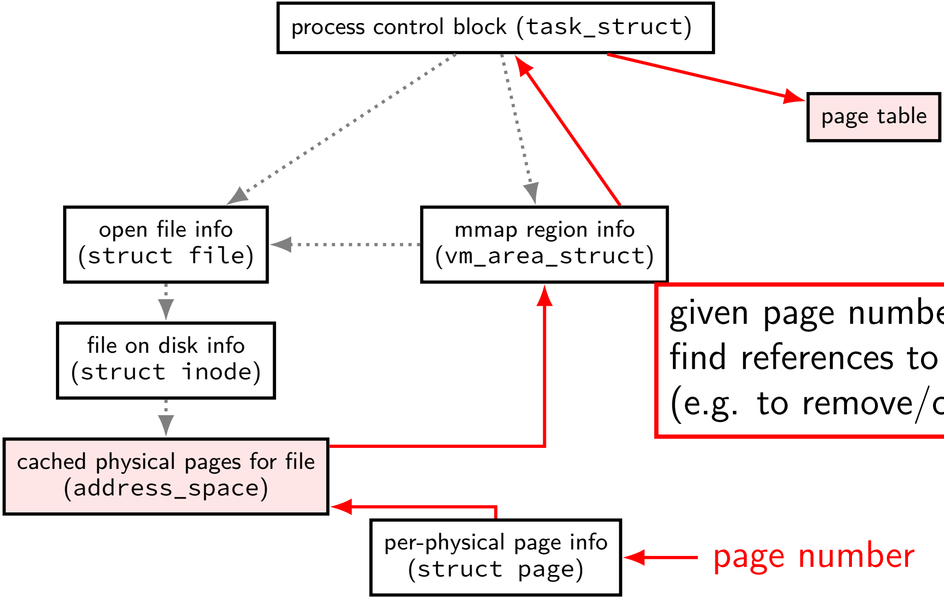


tracking physical pages: finding mappings

want to evict a page? remove from page tables, etc.

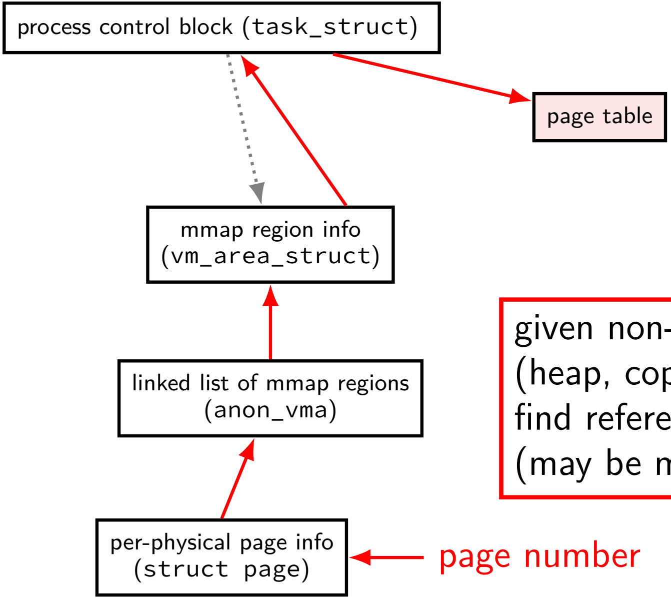
need to track where every page is used!

Linux: reverse mapping (file pages)



given page number
find references to that page
(e.g. to remove/change them)

Linux: reverse mapping (non-file pages)



given non-file page
(heap, copied-on-write copy of file, etc.)
find references to that page
(may be multiple because of `fork`, etc.)

list of allocations per page

naive solution: seperate list for each page?

- a lot of overhead (many tens of bytes per 4K page?)

but, trick: many pages 'copied' at the same time (e.g. fork)

idea: share list between all pages

- initially: list one of mmap region
- on fork: add to existing list; create a new one

Linux: tracking memory regions

for finding other
uses of non-file pages
e.g. two copies after fork

```
struct vm_area_struct { ...
    unsigned long vm_start;          /* Our start address within vm_m
    unsigned long vm_end;            /* The first byte after our end
                                        within vm_mm. */
    ...
    pgprot_t vm_page_prot;           /* Access permissions of this VM
    unsigned long vm_flags;          /* Flags, see mm.h. */
    ...
    struct anon_vma *anon_vma;       /* Serialized by page_table_lock
    ...
    unsigned long vm_pgoff;          /* Offset (within vm_file) in PA
                                        units */
    struct file * vm_file;           /* File we map to (can be NULL).
    ...
} __randomize_layout;
```

page replacement

step 1: evict a page to free a physical page

step 2: load new, more important in its place

evicting a page

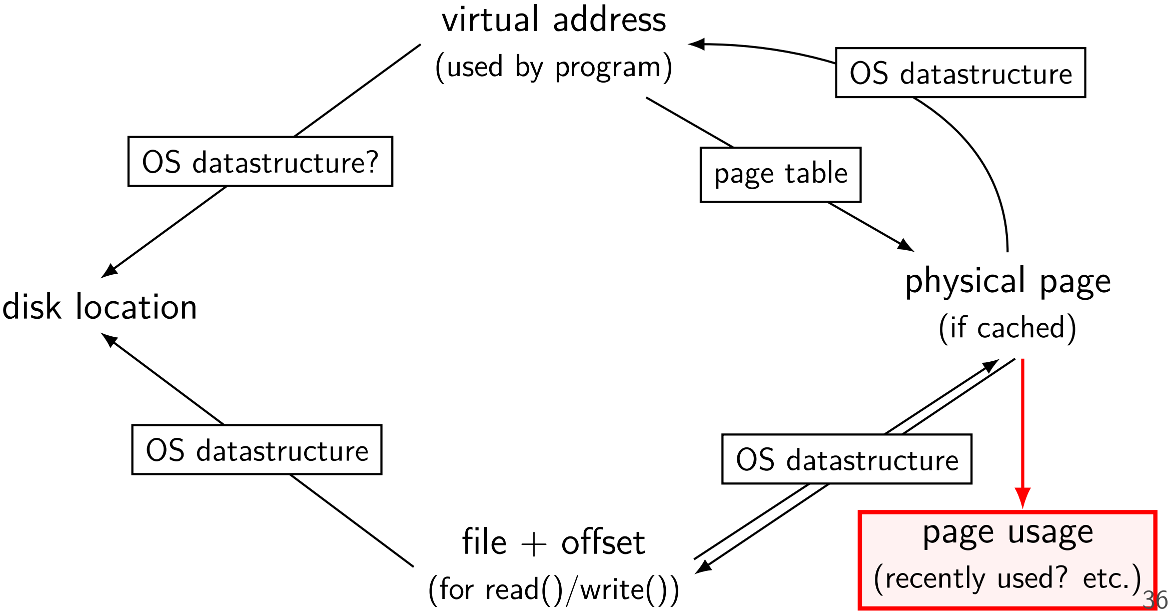
find a 'victim' page to evict

remove victim page from page table, etc.

- every page table it is referenced by
- every list of file pages
- ...

if needed, save victim page to disk

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

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...

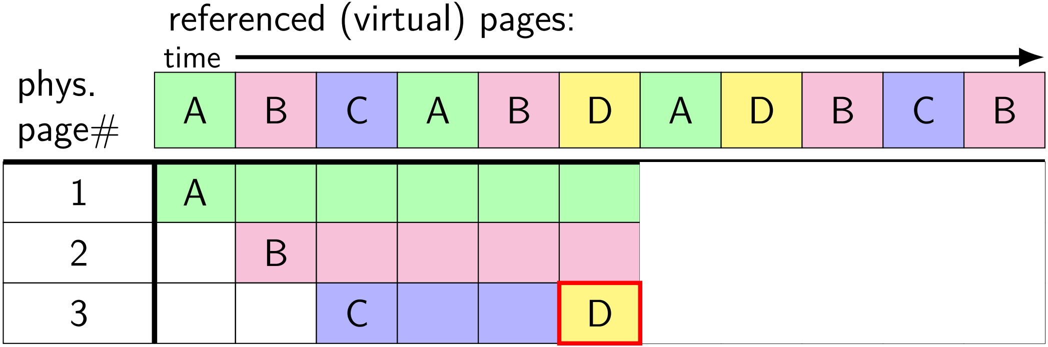
Belady's MIN

referenced (virtual) pages:

time →

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

Belady's MIN



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

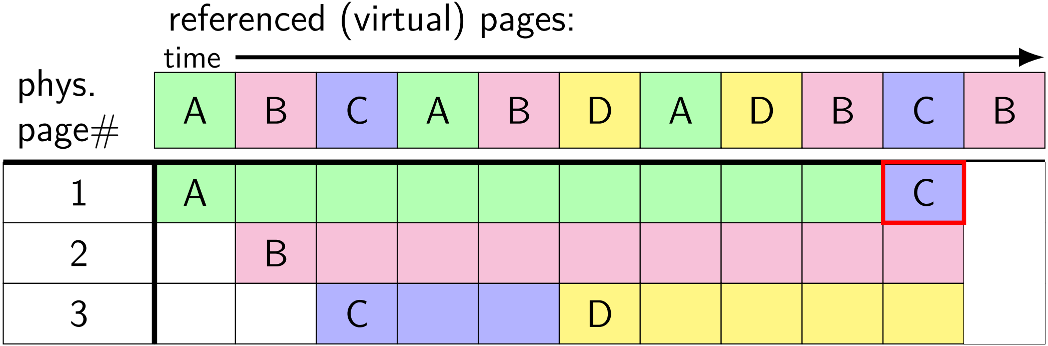
Belady's MIN

referenced (virtual) pages:

time →

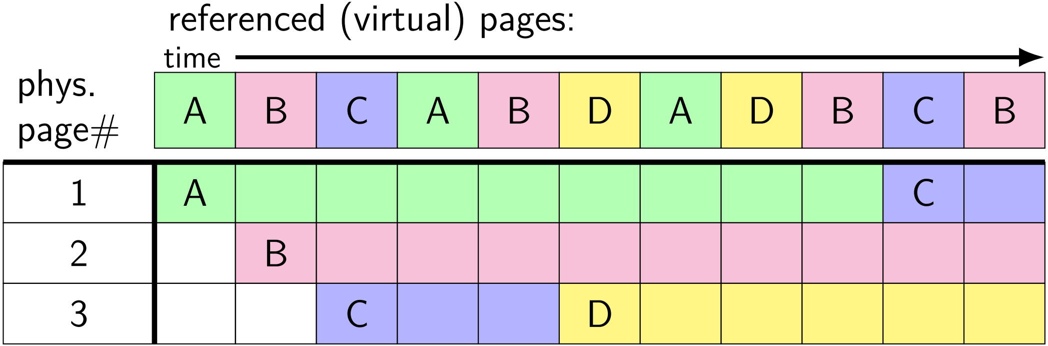
phys. page#	A	B	C	A	B	D	A	D	B	C	B
1	A										
2		B									
3			C			D					

Belady's MIN



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

predicting the future?

can't really...

look for common patterns

the working set model

one common pattern: working sets

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

- set of running functions
- their local variables, (parts of) global data structure

subset called its *working set*

rest of memory is inactive

cache size versus miss rate

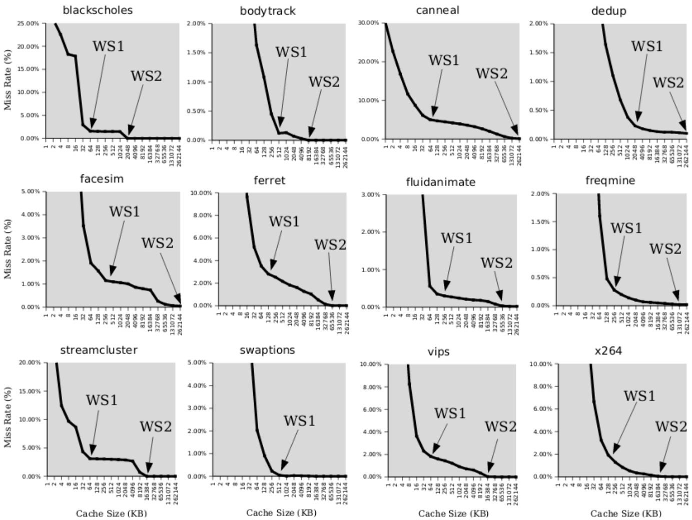


Figure 3: Miss rates versus cache size. Data assumes a shared 4-way associative cache with 64 byte lines. WS1 and WS2 refer to important working sets which we analyze in more detail in Table 2. Cache requirements of PARSEC benchmark programs can reach hundreds of megabytes.

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

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

replacemnet policy: identify working sets (how?)

replace anything that's not in in it

working set model and phases

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

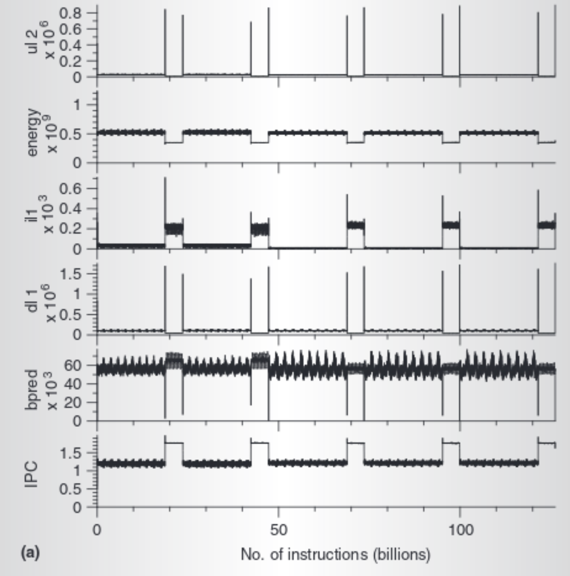
e.g. finish parsing input, now process it

phase change — discard one working set, give another

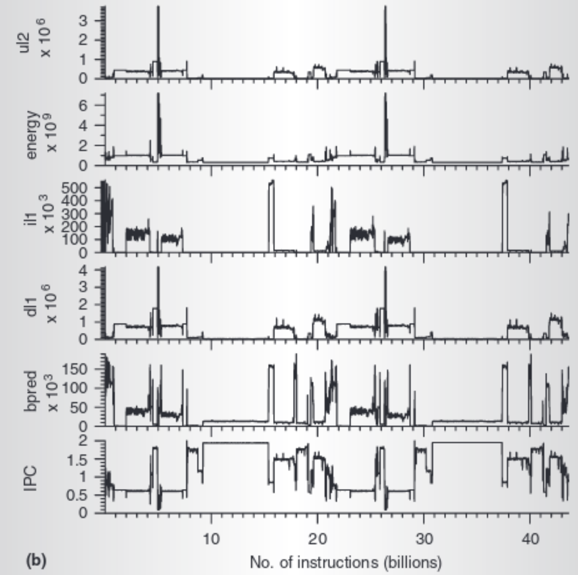
phase changes likely to have spike of cache misses

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

evidence of phases (gzip)



evidence of phases (gcc)



Sherwood et al, "Discovering and Exploiting Program Phases"

estimating working sets

working set $\approx$ what's been used recently

assuming not in phase change...

so, what a program recently used $\approx$ 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

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

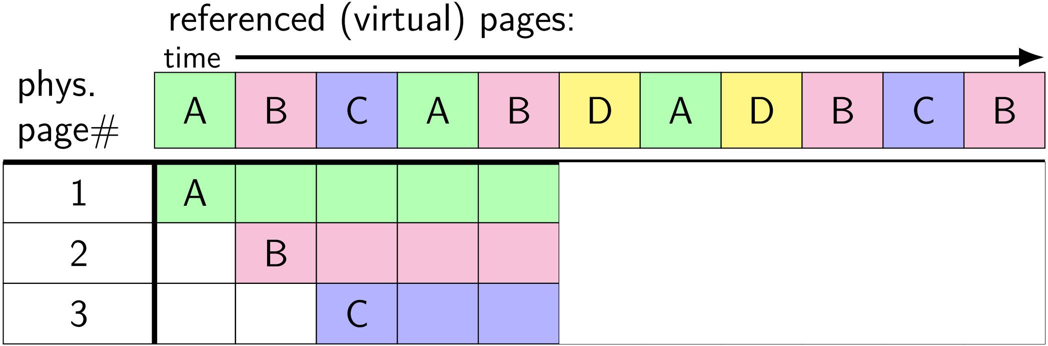
recall?: locality assumption

temporal locality: things accessed now will be accessed again soon

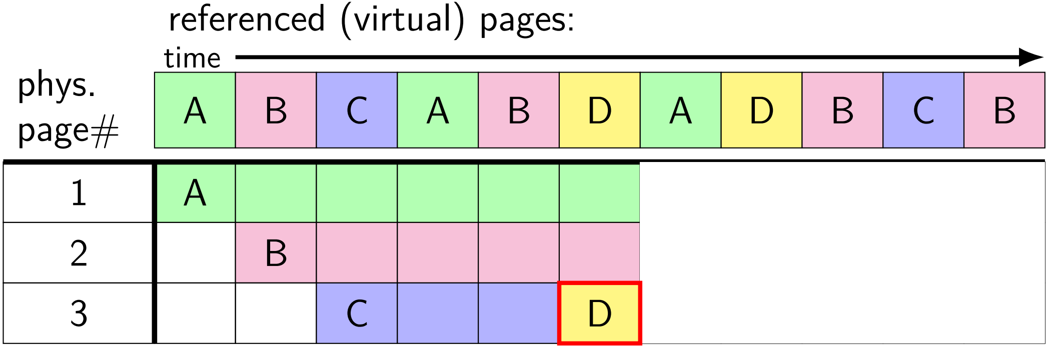
(for now: not concerned about spatial locality)

more possible policies: least recently used or least frequently used

least recently used (the good case)



least recently used (the good case)



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

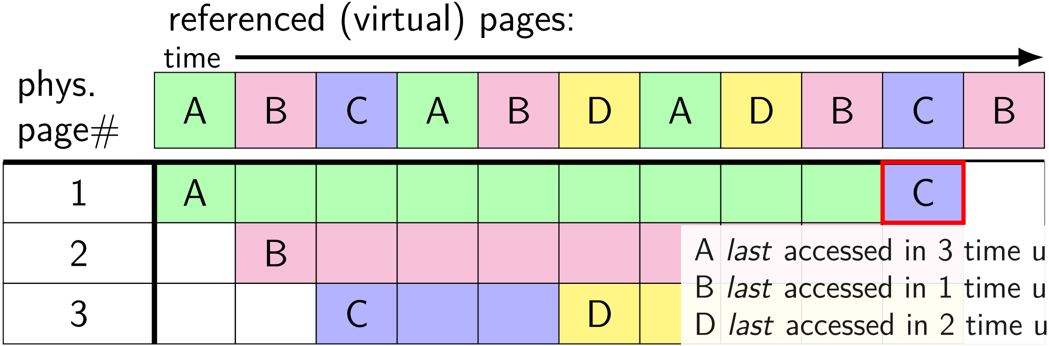
least recently used (the good case)

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)



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:

1	A									B	
2		B					C				
3			C	D							

least recently used (exercise)

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

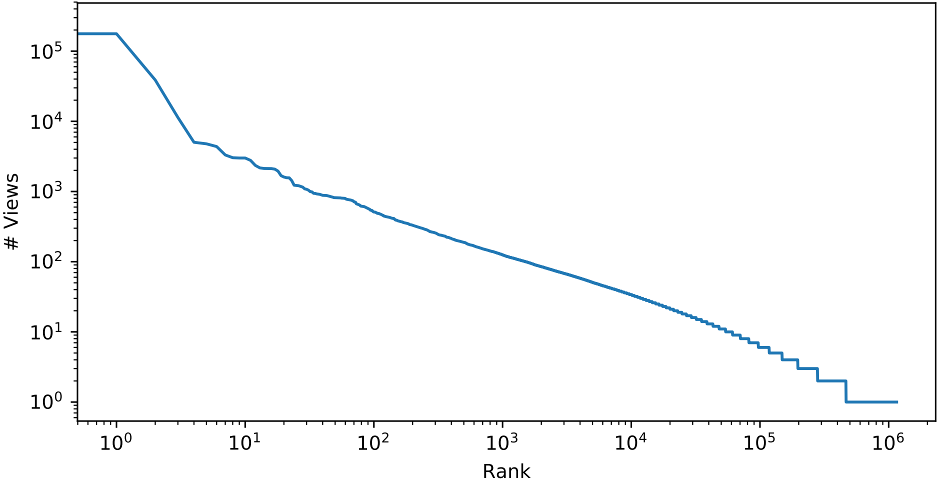
aside: Zipf model

working set model makes sense for programs

but not the only use of caches

example: Wikipedia — most popular articles

Wikipedia page views for 1 hour



NOTE: log-log-scale

Zipf distribution

Zipf distribution: straight line on log-log graph of rank v. count

a few items a much more popular than others

- most caching benefit here

long tail: lots of items accessed a very small number of times

- more cache less efficient — but does something
- not like working set model, where there's just not more

good caching strategy for Zipf

keep the most recently popular things

up till what you have room for

still benefit to caching things used 100 times/hour versus 1000

good caching strategy for Zipf

keep the most recently popular things

up till what you have room for

- still benefit to caching things used 100 times/hour versus 1000

LRU is okay — popular things always recently used

- seems to be what Wikipedia's caches do?

alternative policies for Zipf

least frequently used

- very simple policy
- if pure Zipf distribution — what you want
- practical problem: what about changes in popularity?

least frequently used + adjustments for ‘recentness’

more?

models of reuse

working set/locality

- active things are likely to be active soon
- what's popular changes over time
- want: something like least-recently used

Zipf distribution

- some things are just popular always
- want: something like least-frequently used

other models?

- when X is loaded, Y is always needed?
  - want: identify pairs of related values, load/discard together
- some things are only used once
  - want: identify these, do *not* cache

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

need to run code on every access
mechanism: make every access page fault
which will make everything really slow

whenever

- rem
- evict that page from all page tables (and anything else)
- and use that page for whatever needs to be loaded

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

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:

- mark page invalid, if page fault happens make valid and record 'accessed'
- '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:

- mark page invalid, if page fault happens make valid and record ‘accessed’
- ‘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:

- mark page invalid, if page fault happens make valid and record 'accessed'
- '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
```

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)

...

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
```

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
```

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	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
```

processor does lookup

sets accessed bit to 1 (again)

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 (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**

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

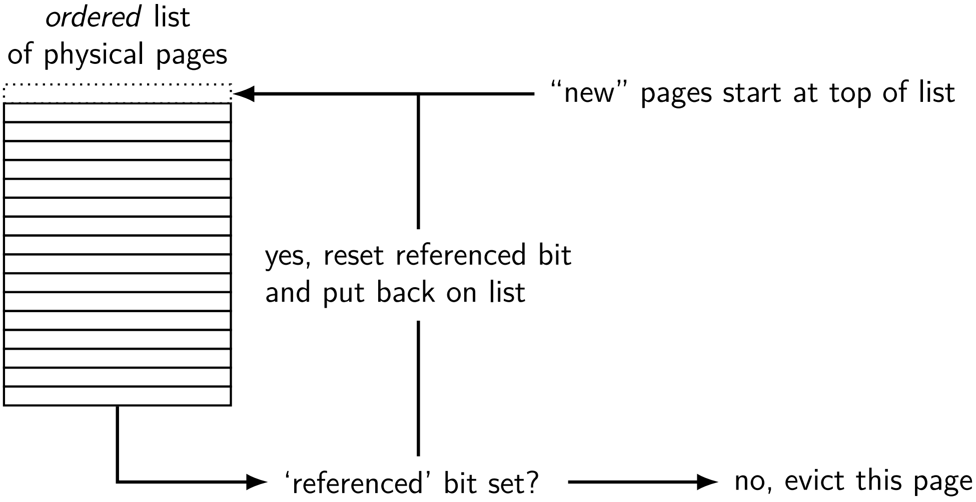
A: acccessed — processor sets to 1 when PTE used

used = for read or write or execute

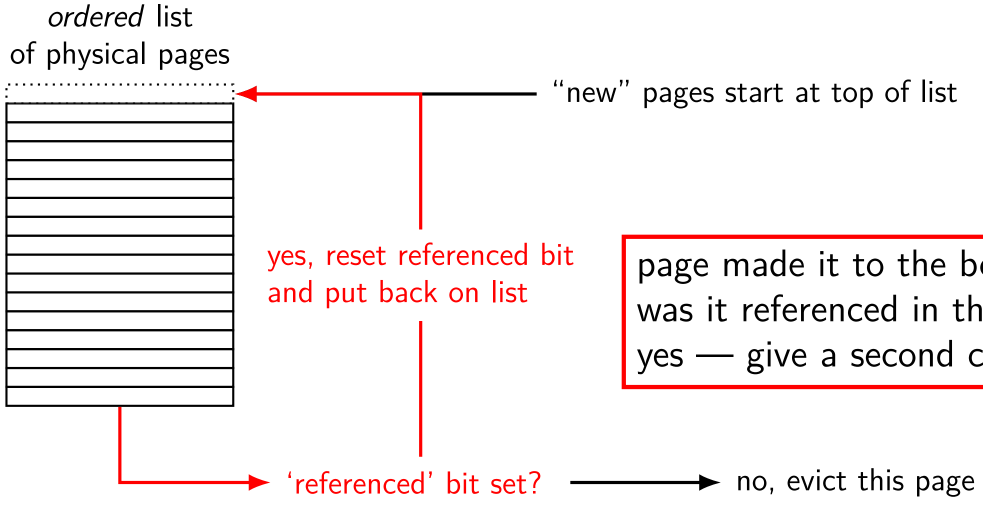
likely implementation: part of loading PTE into TLB

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

approximating LRU: second chance

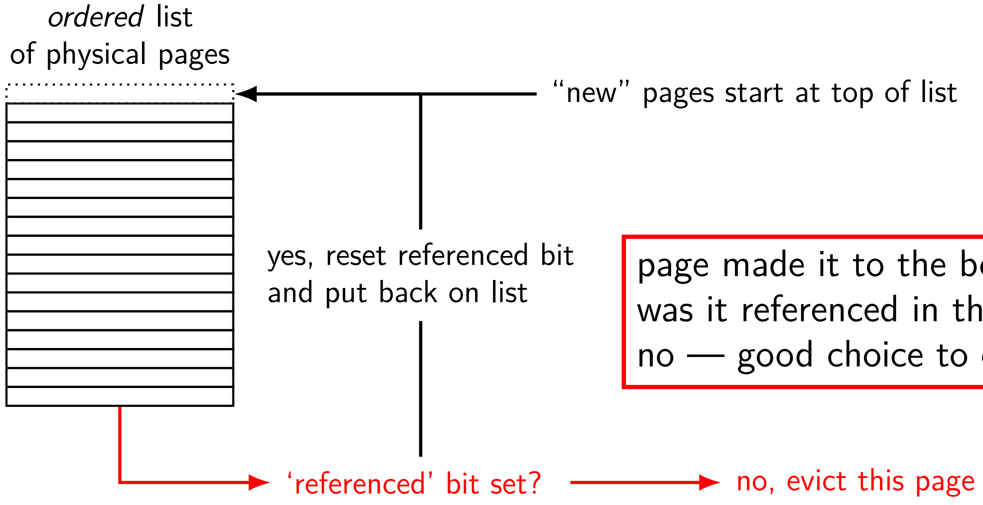


approximating LRU: second chance



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

	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 example

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

	A	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 example

	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 example

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

	A	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 example

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

	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 example

B referenced — flips referenced bit

	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 example

	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

backup slides

Linux: physical page → file → PTE

Linux tracking where file pages are in page tables:

```
struct page {
    ...
    struct address_space *mapping;
    pgoff_t index;              /* Our offset within mapping. */
    ...
};
struct address_space {
    ...
    struct rb_root_cached       i_mmap; /* tree of private and share
    ...
};
```

tree of mappings lets us find vm_area_structs and PTEs

rather complicated look up (but writing ot disk is already slow)

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