virtual memory 2

last time

message passing

two-level page table structure

tricks with page fault handlers

- just-in-time correction of bad/missing page table entry
- return from handler — retry access

allocate-on-demand

- don't set page table entry when program thinks memory alloc'd
- actually alloc memory when first page fault for each page happens

copy-on-write

- mark each page as read-only instead of copying
- actually copy each page when page/protection fault for write happens

fast copies

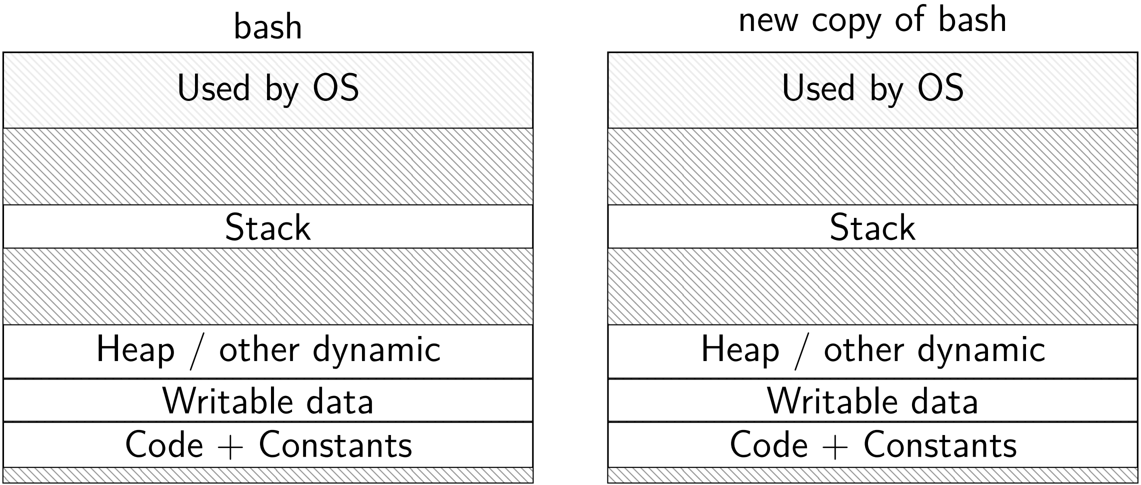
recall : `fork()`

creates a copy of an entire program!

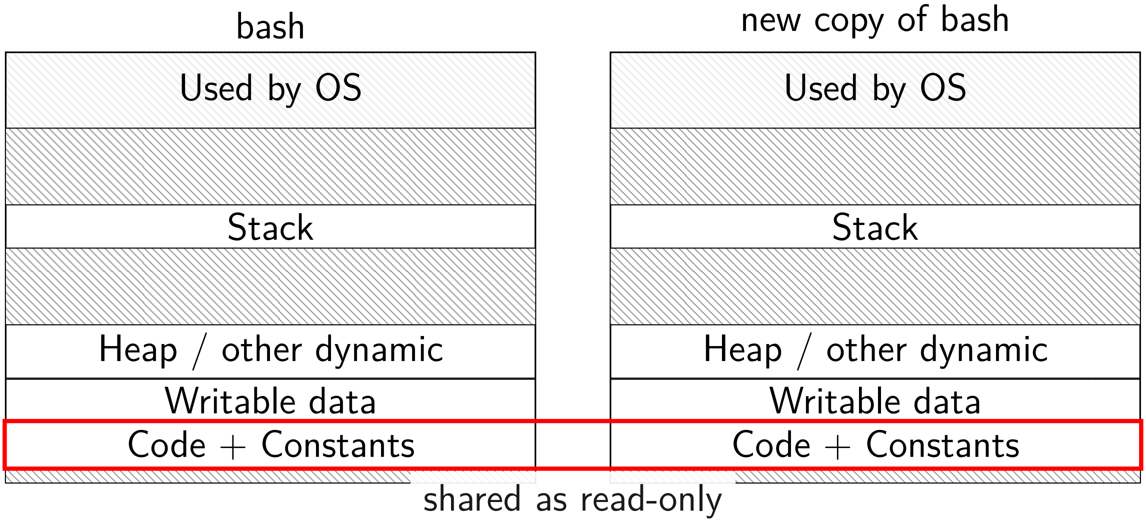
(usually, the copy then calls `execve` — replaces itself with another program)

how isn't this really slow?

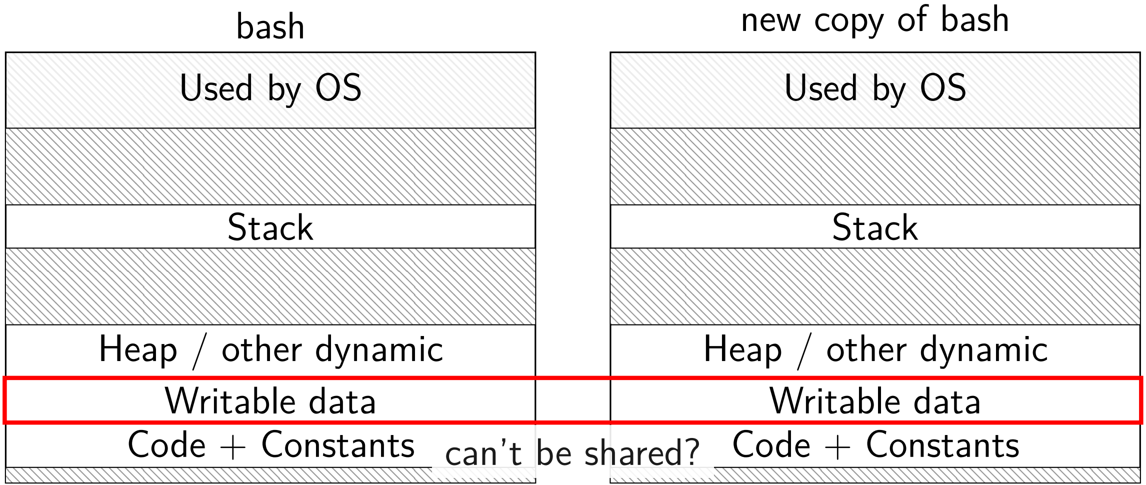
do we really need a complete copy?



do we really need a complete copy?



do we really need a complete copy?



trick for extra sharing

sharing writeable data is fine — until either process modifies the copy

can we detect modifications?

trick: tell CPU (via page table) shared part is read-only

processor will trigger a fault when it's written

copy-on-write and page tables

VPN	valid?	write?	physical page
…	…	…	…
0x00601	1	1	0x12345
0x00602	1	1	0x12347
0x00603	1	1	0x12340
0x00604	1	1	0x200DF
0x00605	1	1	0x200AF
…	…	…	…

copy-on-write and page tables

VPN	valid?	write?	physical page
…	…	…	…
0x00601	1	0	0x12345
0x00602	1	0	0x12347
0x00603	1	0	0x12340
0x00604	1	0	0x200DF
0x00605	1	0	0x200AF
…	…	…	…

VPN	valid?	write?	physical page
…	…	…	…
0x00601	1	0	0x12345
0x00602	1	0	0x12347
0x00603	1	0	0x12340
0x00604	1	0	0x200DF
0x00605	1	0	0x200AF
…	…	…	…

copy operation actually duplicates page table

both processes share all physical pages

but marks pages in both copies as read-only

copy-on-write and page tables

VPN	valid?	write?	physical page
...	...	...	...
0x00601	1	0	0x12345
0x00602	1	0	0x12347
0x00603	1	0	0x12340
0x00604	1	0	0x200DF
0x00605	1	0	0x200AF
...	...	...	...

VPN	valid?	write?	physical page
...	...	...	...
0x00601	1	0	0x12345
0x00602	1	0	0x12347
0x00603	1	0	0x12340
0x00604	1	0	0x200DF
0x00605	1	0	0x200AF
...	...	...	...

when either process tries to write read-only page triggers a fault — OS actually copies the page

copy-on-write and page tables

VPN	valid?	write?	physical page
…	…	…	…
0x00601	1	0	0x12345
0x00602	1	0	0x12347
0x00603	1	0	0x12340
0x00604	1	0	0x200DF
0x00605	1	0	0x200AF
…	…	…	…

VPN	valid?	write?	physical page
…	…	…	…
0x00601	1	0	0x12345
0x00602	1	0	0x12347
0x00603	1	0	0x12340
0x00604	1	0	0x200DF
0x00605	1	1	0x300FD
…	…	…	…

after allocating a copy, OS reruns the write instruction

copy-on write cases

trying to write forbidden page (e.g. kernel memory)

- kill program instead of making it writable

trying to write read-only page and…

only one page table entry refers to it

- make it writeable
- return from fault

multiple process's page table entries refer to it

- copy the page
- replace read-only page table entry to point to copy
- return from fault

mmap

Linux/Unix has a function to “map” a file to memory

```
int file = open("somefile.dat", O_RDWR);

    // data is region of memory that represents file
char *data = mmap(..., file, 0);

    // read byte 6 (zero-indexed) from somefile.dat
char seventh_char = data[6];

   // modifies byte 100 of somefile.dat
data[100] = 'x';
    // can continue to use 'data' like an array
```

mmap options (1)

```
#include <sys/mman.h>
void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
```

`length` bytes from open file `fd` starting at byte `offset`

protection flags `prot`, bitwise or together 1 or more of:

- PROT_READ
- PROT_WRITE
- PROT_EXEC
- PROT_NONE (for forcing segfaults)

mmap options (1)

```
#include <sys/mman.h>
void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
```

`length` bytes from open file `fd` starting at byte `offset`

protection flags `prot`, bitwise or together 1 or more of:

PROT_READ
PROT_WRITE
PROT_EXEC
PROT_NONE (for forcing segfaults)

mmap options (1)

```
#include <sys/mman.h>
void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
```

`length` bytes from open file `fd` starting at byte `offset`

protection flags `prot`, bitwise or together 1 or more of:

PROT_READ
PROT_WRITE
PROT_EXEC
PROT_NONE (for forcing segfaults)

mmap options (2)

```
#include <sys/mman.h>
void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
```

`flags`, choose at least

- MAP_SHARED — changing memory changes file and vice-versa
- MAP_PRIVATE — make a copy of data in file (using copy-on-write)

…along with additional flags:

- MAP_ANONYMOUS (not POSIX) — ignore `fd`, just allocate space
- … (and more not shown)

`addr`, suggestion about where to put mapping (may be ignored)

- can pass NULL — "choose for me"
- address chosen will be returned

mmap options (2)

```
#include <sys/mman.h>
void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
```

flags, choose at least

- MAP_SHARED — changing memory changes file and vice-versa
- MAP_PRIVATE — make a copy of data in file (using copy-on-write)

...along with additional flags:

- MAP_ANONYMOUS (not POSIX) — ignore fd, just allocate space
- ... (and more not shown)

addr, suggestion about where to put mapping (may be ignored)

- can pass NULL — “choose for me”
- address chosen will be returned

mmap options (2)

```
#include <sys/mman.h>
void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
```

`flags`, choose at least

- MAP_SHARED — changing memory changes file and vice-versa
- MAP_PRIVATE — make a copy of data in file (using copy-on-write)

...along with additional flags:

- MAP_ANONYMOUS (not POSIX) — ignore `fd`, just allocate space
- ... (and more not shown)

`addr`, suggestion about where to put mapping (may be ignored)

- can pass NULL — "choose for me"
- address chosen will be returned

mmap options (2)

```
#include <sys/mman.h>
void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
```

`flags`, choose at least

- MAP_SHARED — changing memory changes file and vice-versa
- MAP_PRIVATE — make a copy of data in file (using copy-on-write)

...along with additional flags:

- MAP_ANONYMOUS (not POSIX) — ignore `fd`, just allocate space
- ... (and more not shown)

`addr`, suggestion about where to put mapping (may be ignored)

- can pass NULL — "choose for me"
- address chosen will be returned

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.s
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.s
7f60c7a7c000-7f60c7a7d000 rw-p 00023000 08:01 96659109  /lib/x86_64-linux-gnu/ld-2.19.s
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 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                                                          -2.19
7f60c7852000-7f60c7854000 rw                                                         -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.s
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.s
7f60c7a7c000-7f60c7a7d000 rw-p 00023000 08:01 96659109   /lib/x86_64-linux-gnu/ld-2.19.s
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]
```

at virtual addresses `0x400000–0x40b000`

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 001be                                           c-2.19
7f60c7852000-7f60c7854000 rw-p 001c2                                           c-2.19
7f60c7854000-7f60c7859000 rw-p 00000
7f60c7859000-7f60c787c000 r-xp 00000                                            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]
```

read, not write, execute, private

private = copy-on-write (if writeable)

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                                                   -2.19
7f60c7852000-7f60c7854000 rw-p                                                   -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.s
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.s
7f60c7a7c000-7f60c7a7d000 rw-p 00023000 08:01 96659109  /lib/x86_64-linux-gnu/ld-2.19.s
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]
```

starting at offset 0 of the file /bin/cat

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
7f60c784e000-7                                                                    -2.19
7f60c7852000-7                                                                    -2.19
7f60c7854000-7
7f60c7859000-7                                                                    2.19.so
7f60c7a39000-7
7f60c7a7a000-7
7f60c7a7b000-7                                                                    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]
```

device major number 8

device minor number 1

inode 48328831

more on what this means when we talk about filesystems

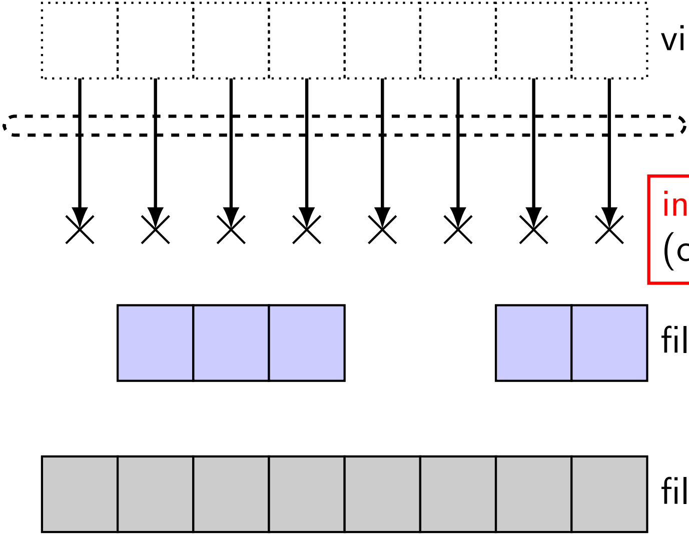
Linux maps

```
$ 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
7f60c784e000-7f6                                                              -2.19
7f60c7852000-7f6                                                              -2.19
7f60c7854000-7f6
7f60c7859000-7f6                                                              2.19.so
7f60c7a39000-7f6
7f60c7a7a000-7f6
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]
```

as if:

```
int fd = open("/bin/cat", O_RDONLY);
mmap(0x400000, 0x1000, PROT_READ | PROT_EXEC,
     MAP_PRIVATE, fd, 0xb000);
```

mapped pages (read-only)



virtual pages mapped to file

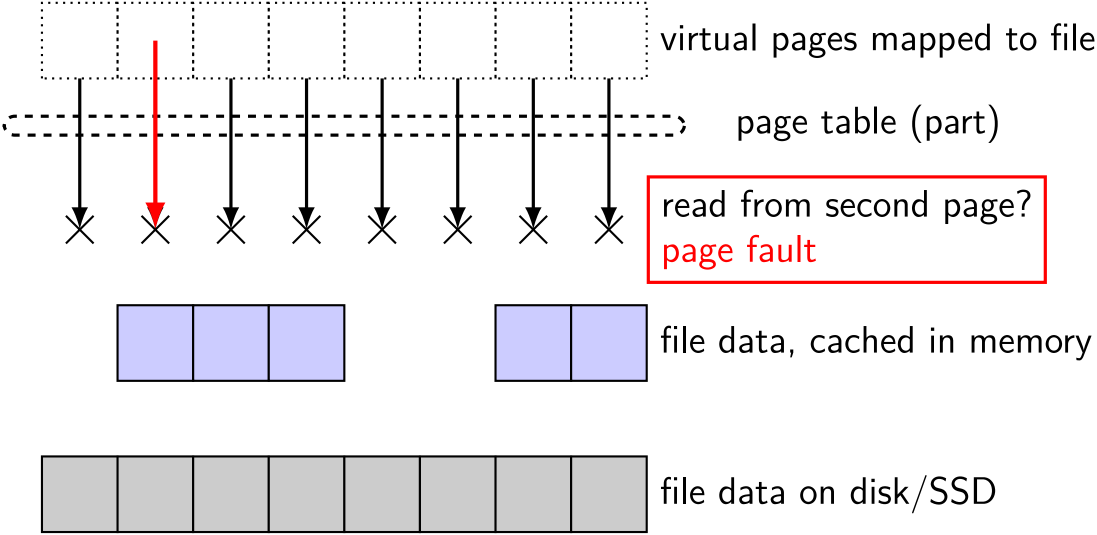
page table (part)

initially — all invalid?
(could also prefill entries...)

file data, cached in memory

file data on disk/SSD

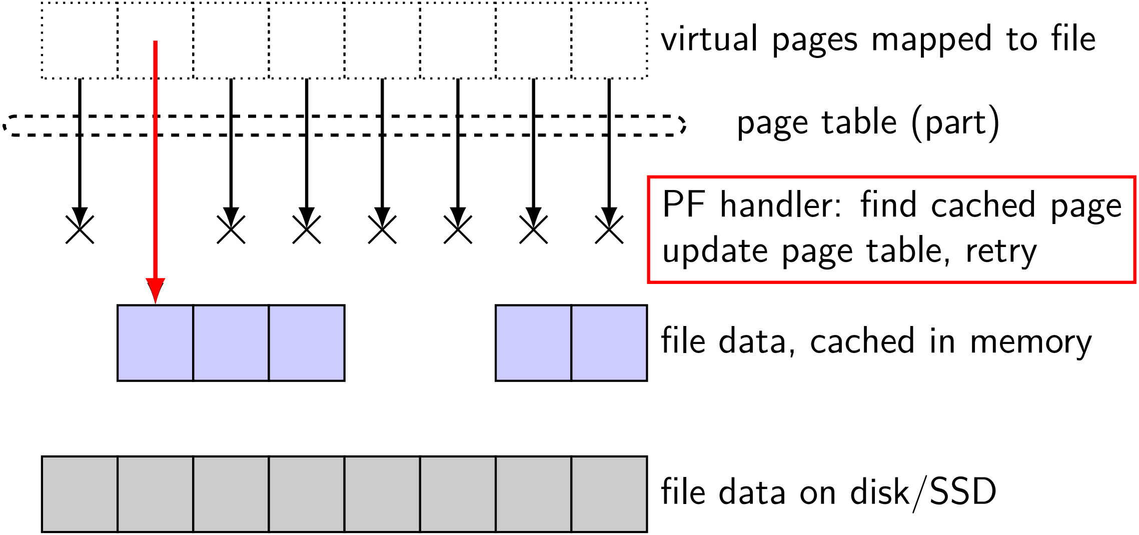
mapped pages (read-only)



read from second page?

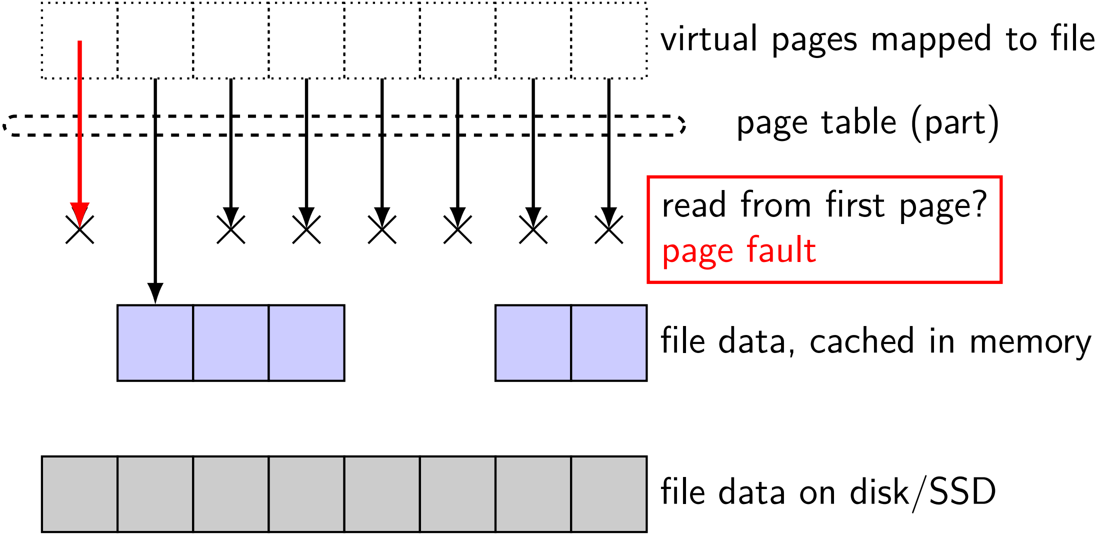
page fault

mapped pages (read-only)



PF handler: find cached page
update page table, retry

mapped pages (read-only)



virtual pages mapped to file

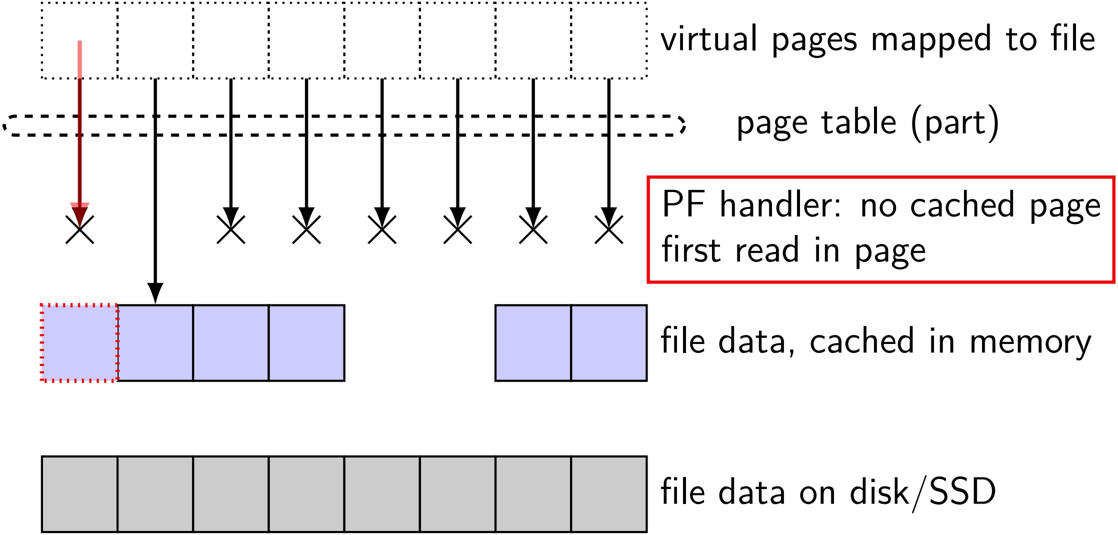
page table (part)

read from first page?
page fault

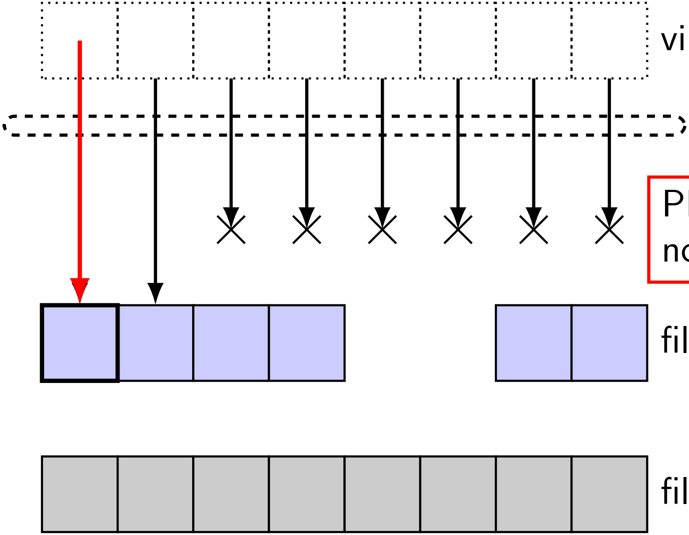
file data, cached in memory

file data on disk/SSD

mapped pages (read-only)


virtual pages mapped to file
page table (part)
PF handler: no cached page
first read in page
file data, cached in memory
file data on disk/SSD

mapped pages (read-only)



virtual pages mapped to file

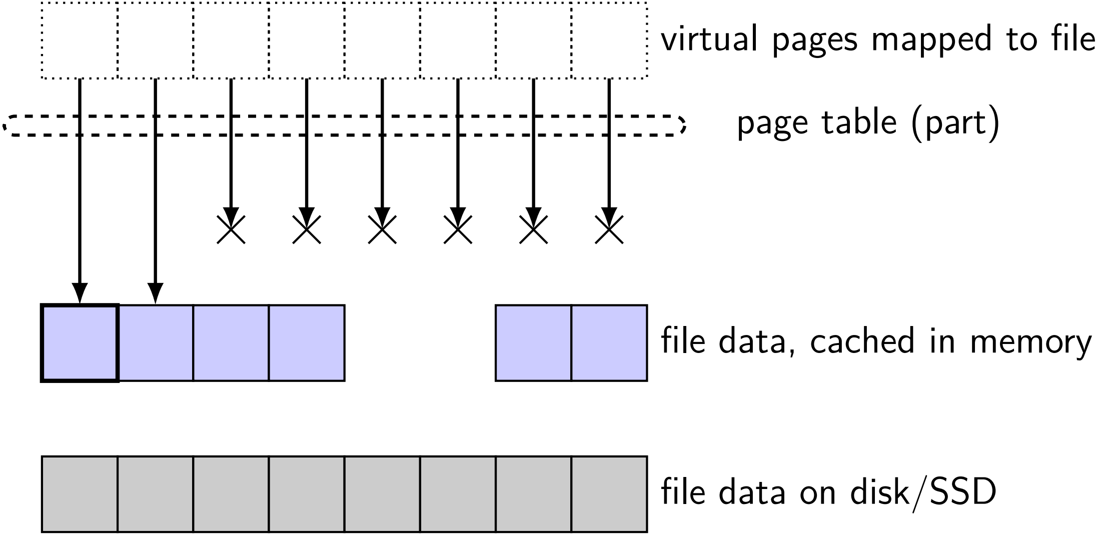
page table (part)

PF handler: read in page
now point to page

file data, cached in memory

file data on disk/SSD

mapped pages (read-only)



shared mmap

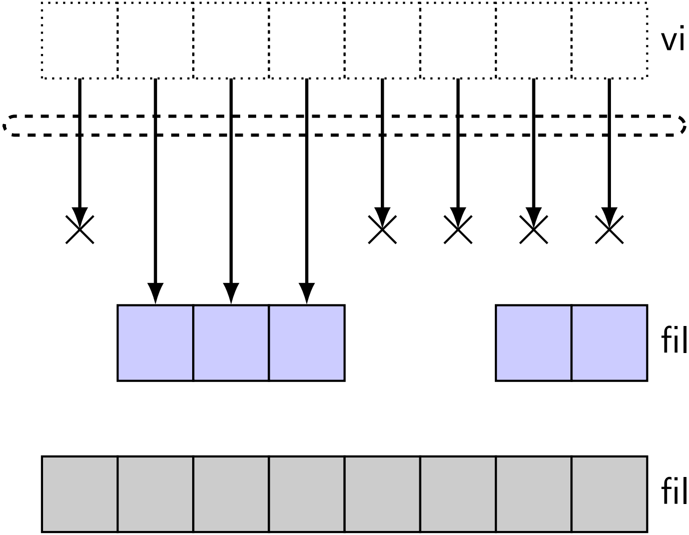
```
int fd  = open("/tmp/somefile.dat", O_RDWR);
mmap(0, 64 * 1024, PROT_READ | PROT_WRITE,
     MAP_SHARED, fd, 0);
```

---

from /proc/PID/maps for this program:

```
7f93ad877000-7f93ad887000 rw-s 00000000 08:01 1839758 /tmp/somefile.dat
```

mapped pages (read/write, shared)



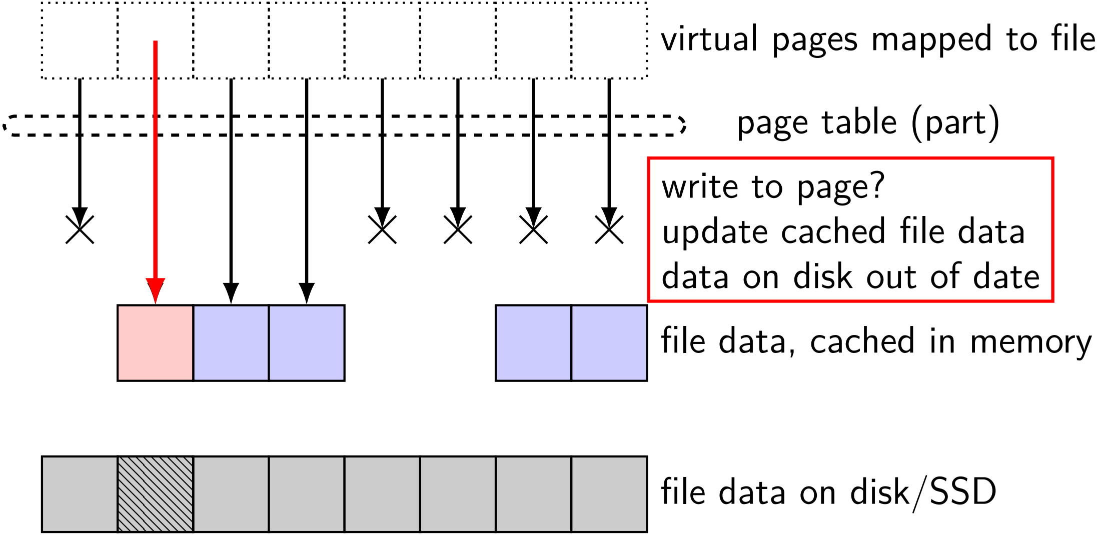
virtual pages mapped to file

page table (part)

file data, cached in memory

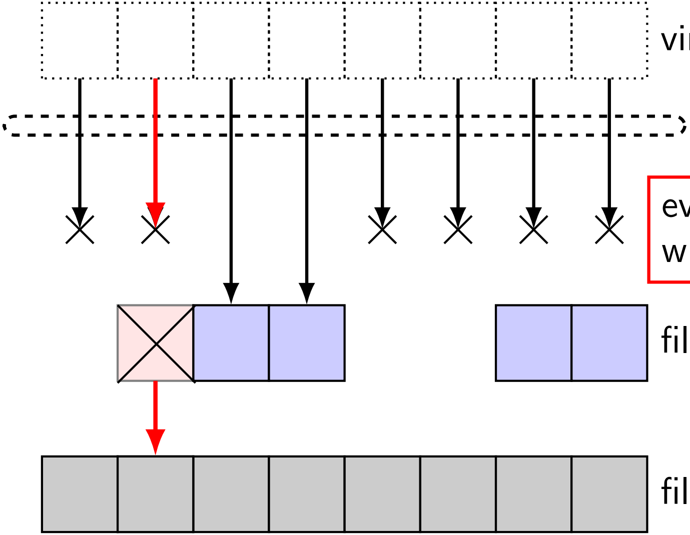
file data on disk/SSD

mapped pages (read/write, shared)



write to page?
update cached file data
data on disk out of date

mapped pages (read/write, shared)



virtual pages mapped to file

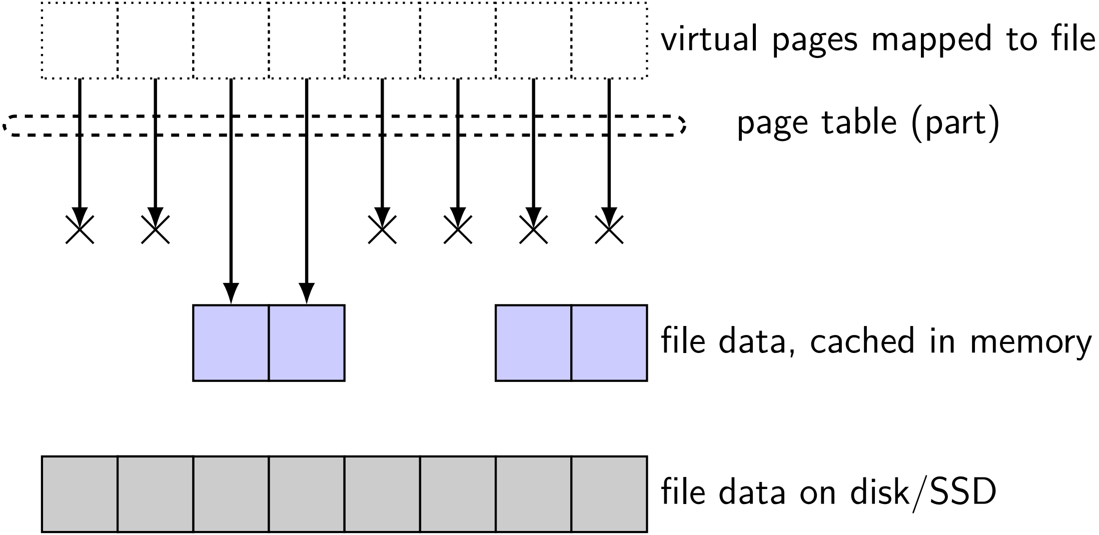
page table (part)

eventually free memory...
write update to disk

file data, cached in memory

file data on disk/SSD

mapped pages (read/write, shared)



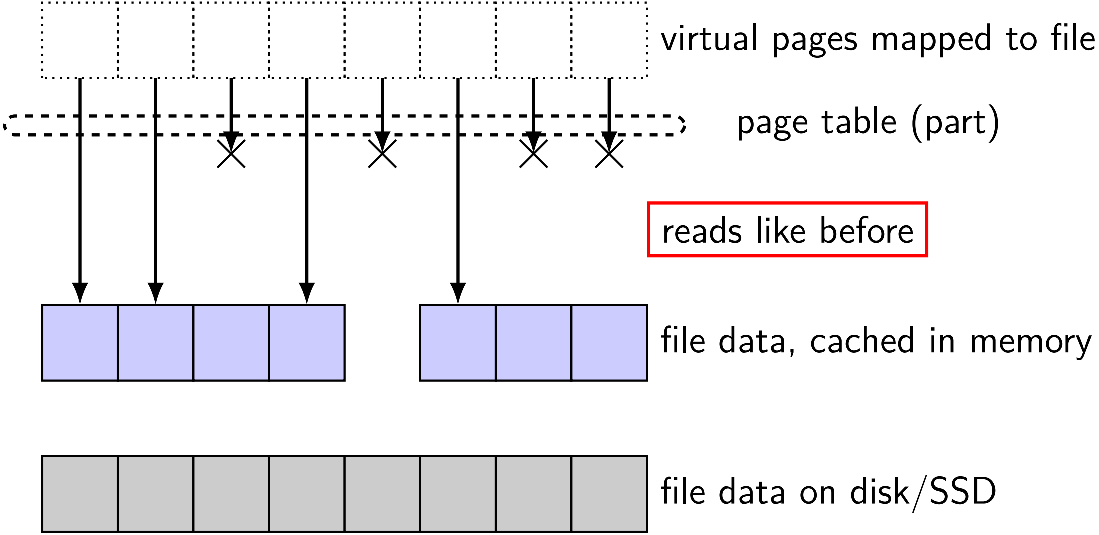
Linux maps

```
$ 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
7f60c784e000-7f                                                                  c-2.19
7f60c7852000-7f                                                                  c-2.19
7f60c7854000-7f
7f60c7859000-7f                                                                  2.19.so
7f60c7a39000-7f
7f60c7a7a000-7f
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]
```

read/write, copy-on-write (private) mapping

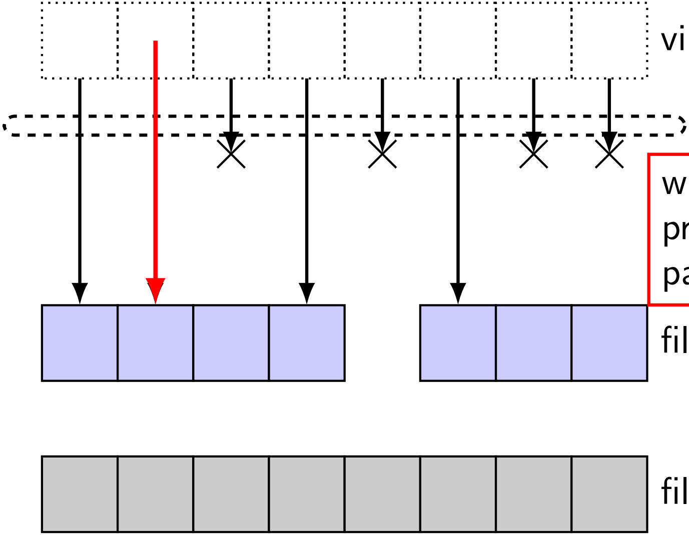
```
int fd = open("/bin/cat", O_RDONLY);
mmap(0x60b000, 0x1000, PROT_READ | PROT_WRITE,
     MAP_PRIVATE, fd, 0xb000);
```

mapped pages (copy-on-write)



reads like before

mapped pages (copy-on-write)



virtual pages mapped to file

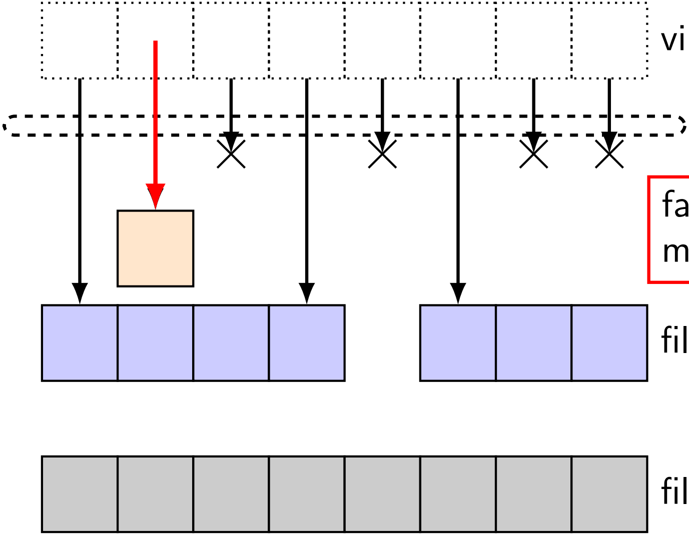
page table (part)

write to second page?
protection fault
page table entry says read-only

file data, cached in memory

file data on disk/SSD

mapped pages (copy-on-write)



virtual pages mapped to file

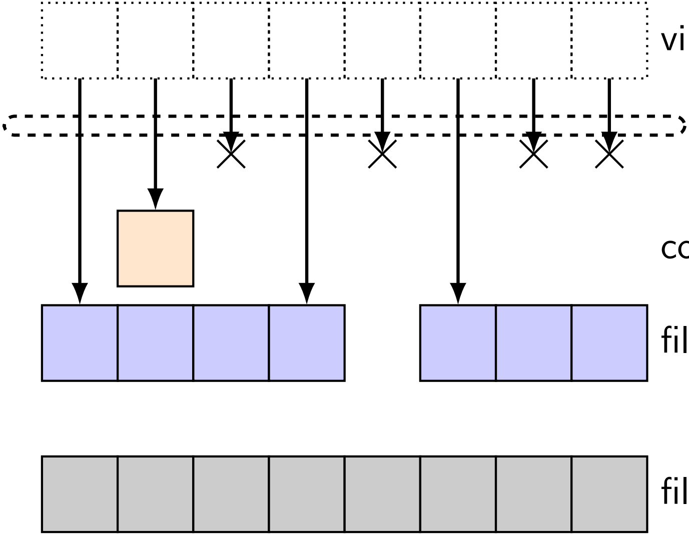
page table (part)

fault handler:
make copy, update page table

file data, cached in memory

file data on disk/SSD

mapped pages (copy-on-write)



virtual pages mapped to file

page table (part)

copies of file data, modified

file data, cached in memory

file data on disk/SSD

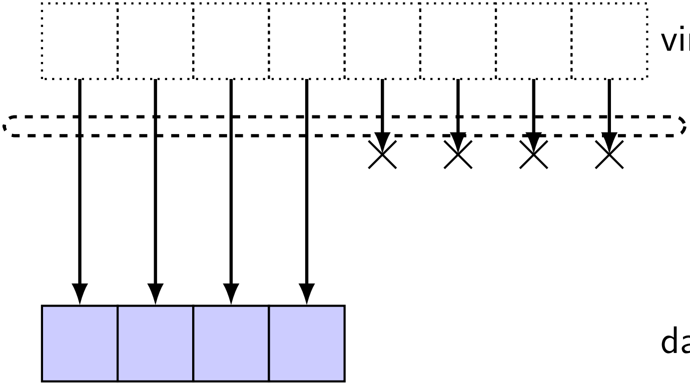
Linux maps

```
$ 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                                  c-2.19
7f60c7852000-7f60c7854000 rw-p 001c2000 08:01                                  c-2.19
7f60c7854000-7f60c7859000 rw-p 00000000 00:00
7f60c7859000-7f60c787c000 r-xp 00000000 08:01                                  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]
```

heap — no corresponding file

just read/write memory

mapped pages (no backing file)



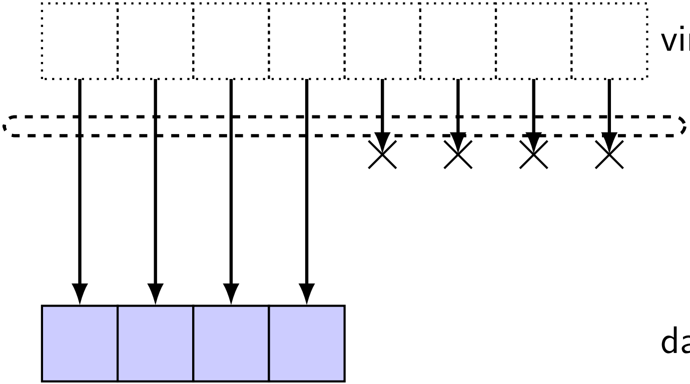
virtual pages w/o backing file

page table (part)

data in memory

swapped out data (if any)

mapped pages (no backing file)



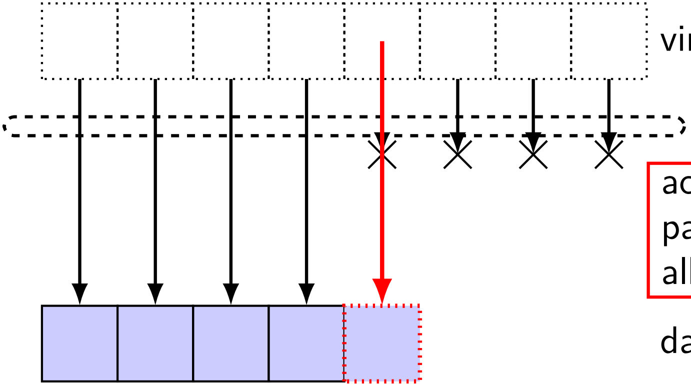
virtual pages w/o backing file

page table (part)

data in memory

swapped out data (if any)

mapped pages (no backing file)



virtual pages w/o backing file

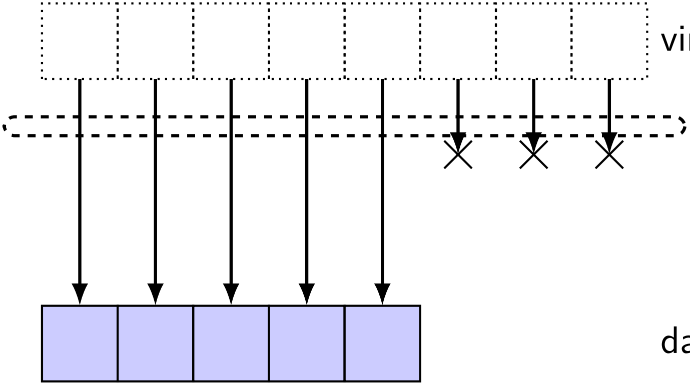
page table (part)

access new page
page fault handler
allocates on demand

data in memory

swapped out data (if any)

mapped pages (no backing file)



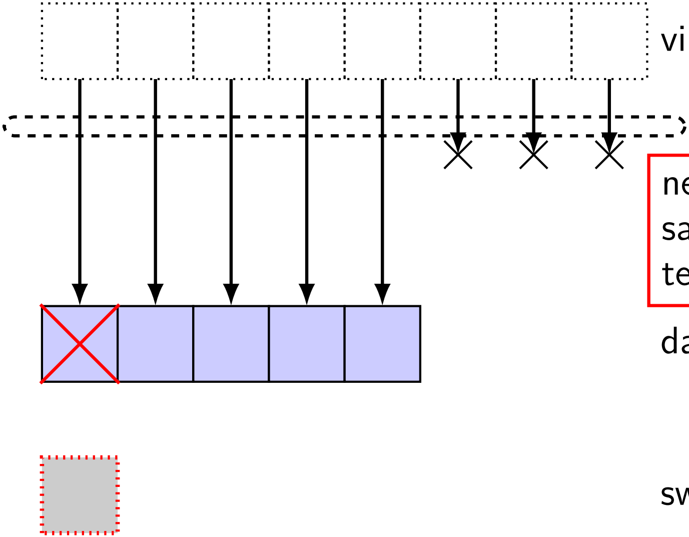
virtual pages w/o backing file

page table (part)

data in memory

swapped out data (if any)

mapped pages (no backing file)



virtual pages w/o backing file

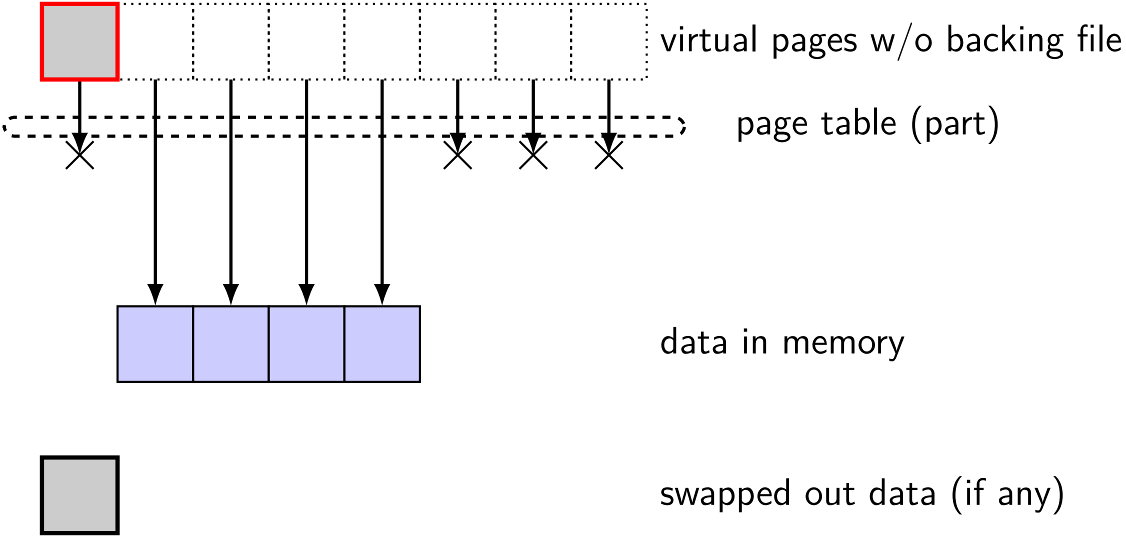
page table (part)

need more memory
save page to disk
temporarily

data in memory

swapped out data (if any)

mapped pages (no backing file)



virtual pages w/o backing file

page table (part)

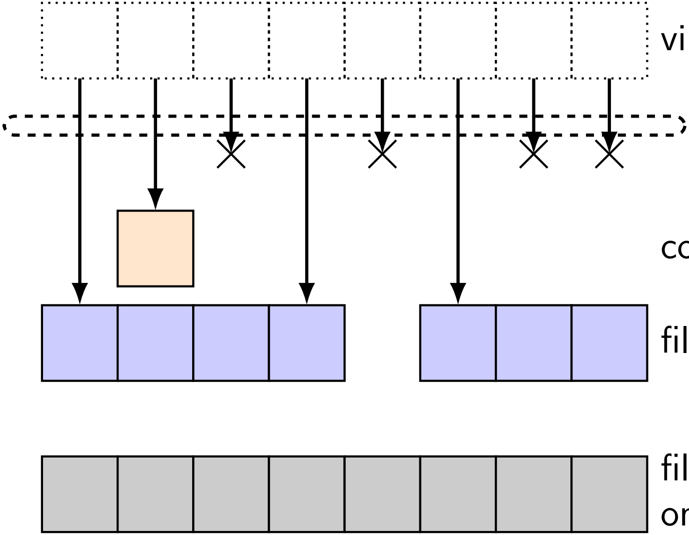
data in memory

swapped out data (if any)

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]
```

swapping with copy-on-write



virtual pages mapped to file

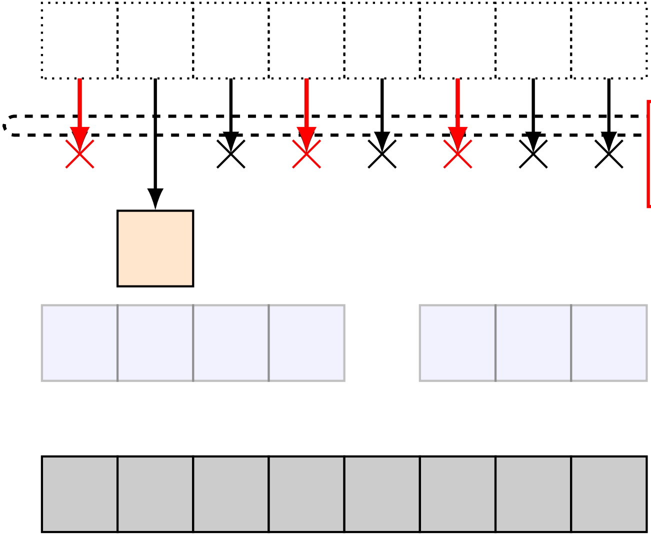
page table (part)

copies of file data, modified

file data, cached in memory

file data
on disk/SSD

swapping with copy-on-write



virtual pages mapped to file

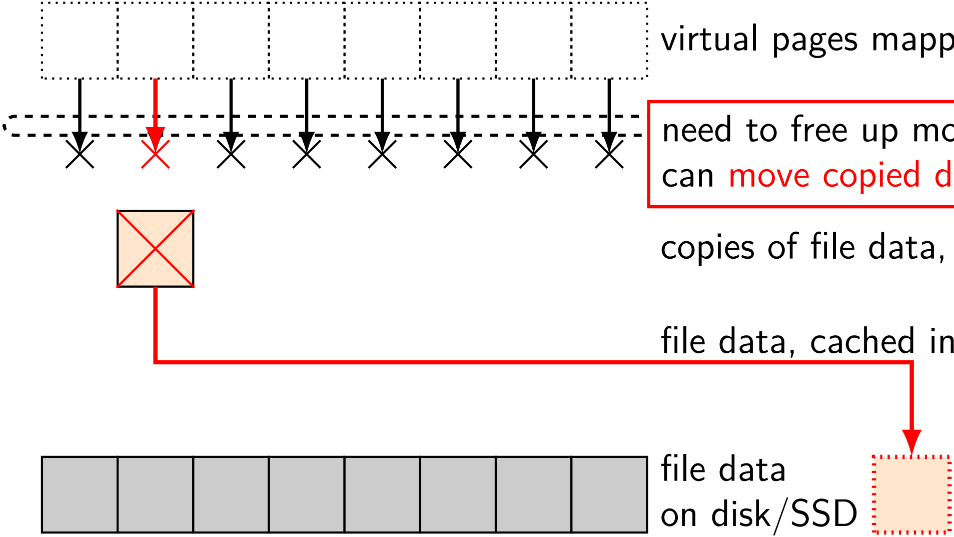
free up space by removing cached copies of file

copies of file data, modified

file data, cached in memory

file data
on disk/SSD

swapping with copy-on-write



virtual pages mapped to file

need to free up more space?
can move copied data to disk

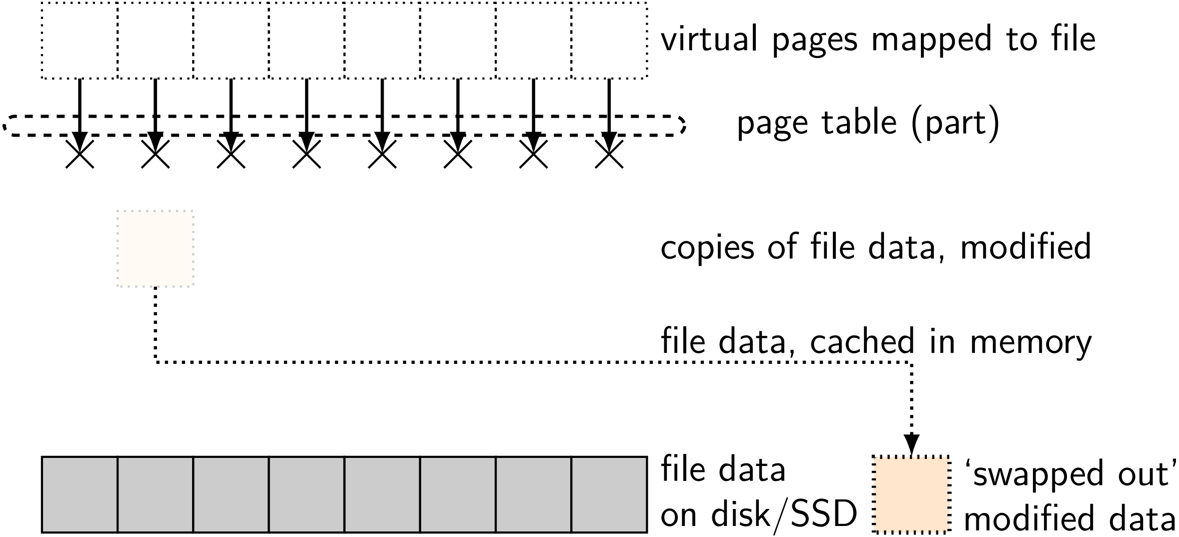
copies of file data, modified

file data, cached in memory

file data
on disk/SSD

“swapped out’
modified data

swapping with copy-on-write


virtual pages mapped to file
page table (part)
copies of file data, modified
file data, cached in memory
file data
on disk/SSD
'swapped out'
modified data

swapping

historical major use of virtual memory is supporting "swapping"

using disk (or SSD, ...) as the next level of the memory hierarchy

process is allocated space on disk/SSD

memory is a cache for disk/SSD

- only need keep 'currently active' pages in physical memory

swapping

historical major use of virtual memory is supporting “swapping”

using disk (or SSD, …) as the next level of the memory hierarchy

process is allocated space on disk/SSD

memory is a cache for disk/SSD

- only need keep ‘currently active’ pages in physical memory

swapping $\approx$ mmap with “default” files to use

HDD/SDDs are slow

HDD reads and writes: milliseconds to tens of milliseconds

- minimum size: 512 bytes
- writing tens of kilobytes basically as fast as writing 512 bytes

SSD writes and writes: hundreds of microseconds

- designed for writes/reads of kilobytes (not much smaller)

HDD/SDDs are slow

HDD reads and writes: milliseconds to tens of milliseconds

- minimum size: 512 bytes
- writing tens of kilobytes basically as fast as writing 512 bytes

SSD writes and writes: hundreds of microseconds

- designed for writes/reads of kilobytes (not much smaller)

HDD/SDDs are slow

HDD reads and writes: milliseconds to tens of milliseconds

- minimum size: 512 bytes
- writing tens of kilobytes basically as fast as writing 512 bytes

SSD writes and writes: hundreds of microseconds

- designed for writes/reads of kilobytes (not much smaller)

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" ≈ 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

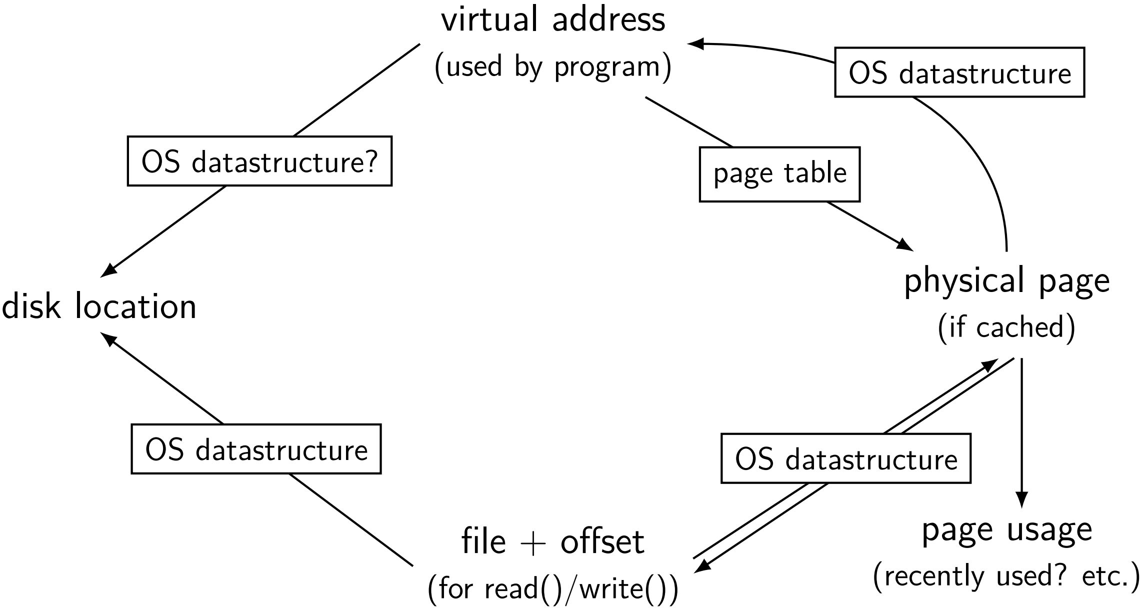
find backing location based on virtual address/file+offset

- handle cache misses

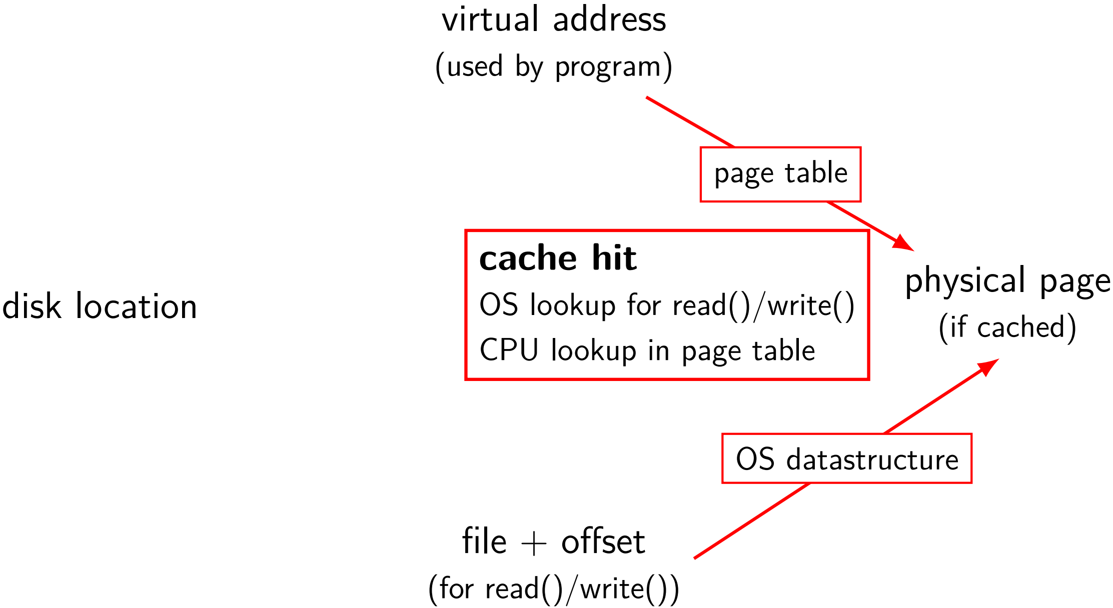
track information about each physical page

- handle page allocation
- handle cache eviction

page cache components



page cache components



virtual address

(used by program)

page table

cache hit

OS lookup for read()/write()

CPU lookup in page table

disk location

physical page

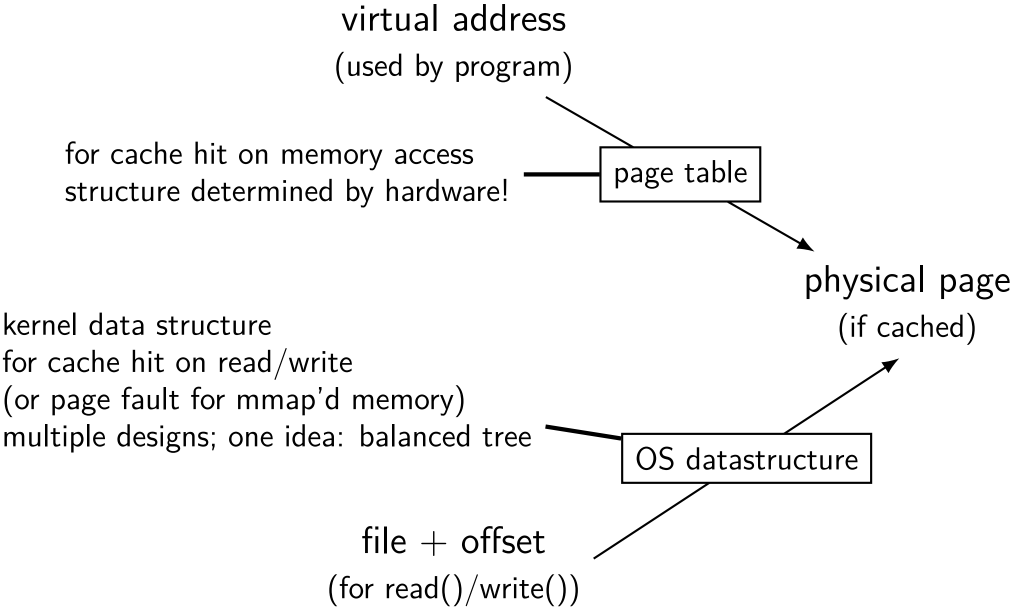
(if cached)

OS datastructure

file + offset

(for read()/write())

virtual addr/file offset to physical page



virtual address

(used by program)

for cache hit on memory access
structure determined by hardware!

page table

physical page

(if cached)

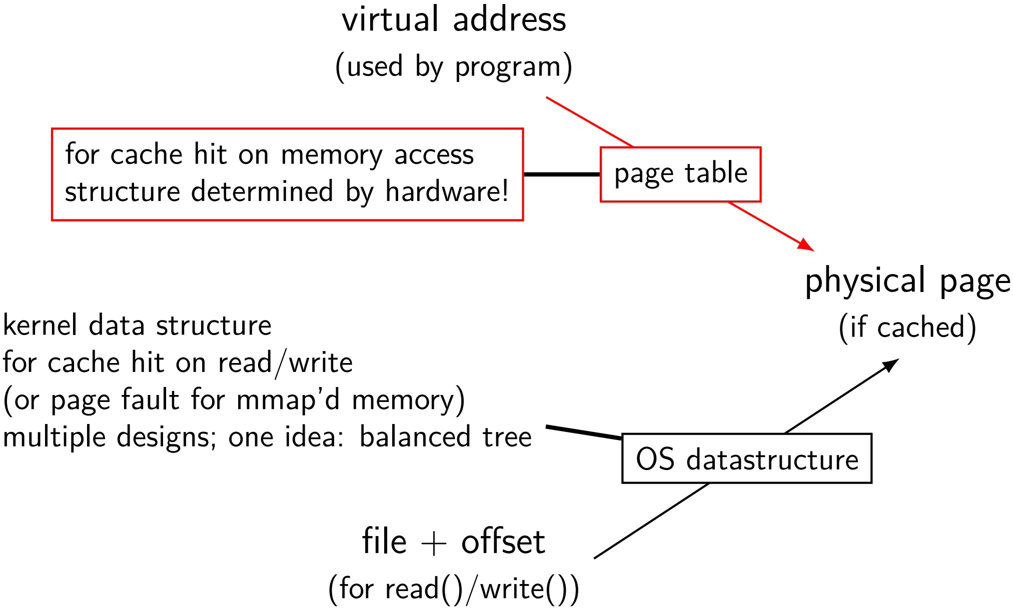
kernel data structure
for cache hit on read/write
(or page fault for mmap'd memory)
multiple designs; one idea: balanced tree

OS datastructure

file + offset

(for read()/write())

virtual addr/file offset to physical page



virtual address

(used by program)

page table

for cache hit on memory access
structure determined by hardware!

physical page

(if cached)

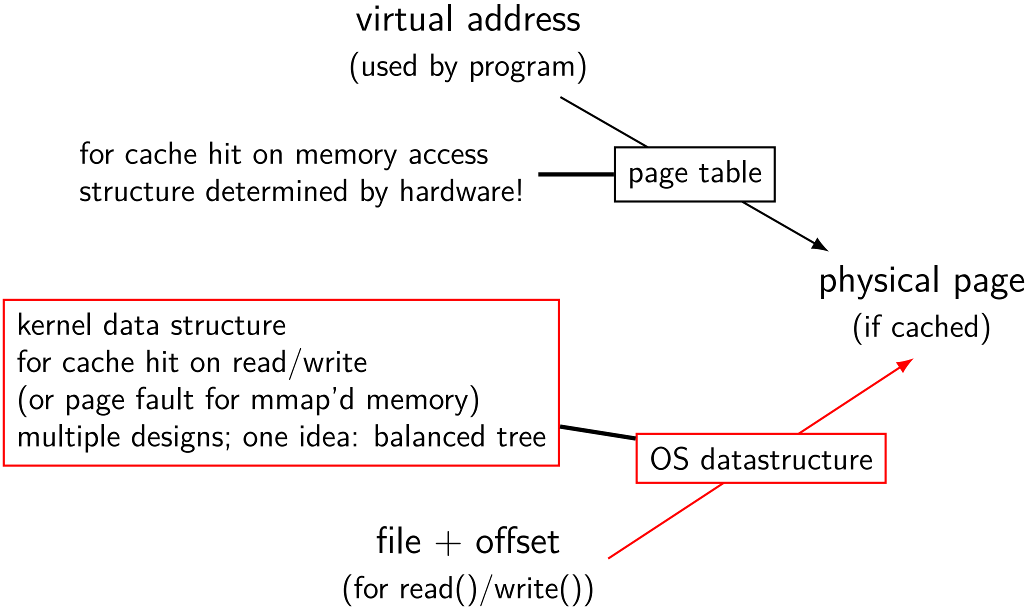
kernel data structure
for cache hit on read/write
(or page fault for mmap'd memory)
multiple designs; one idea: balanced tree

OS datastructure

file + offset

(for read()/write())

virtual addr/file offset to physical page



virtual address

(used by program)

page table

for cache hit on memory access
structure determined by hardware!

physical page

(if cached)

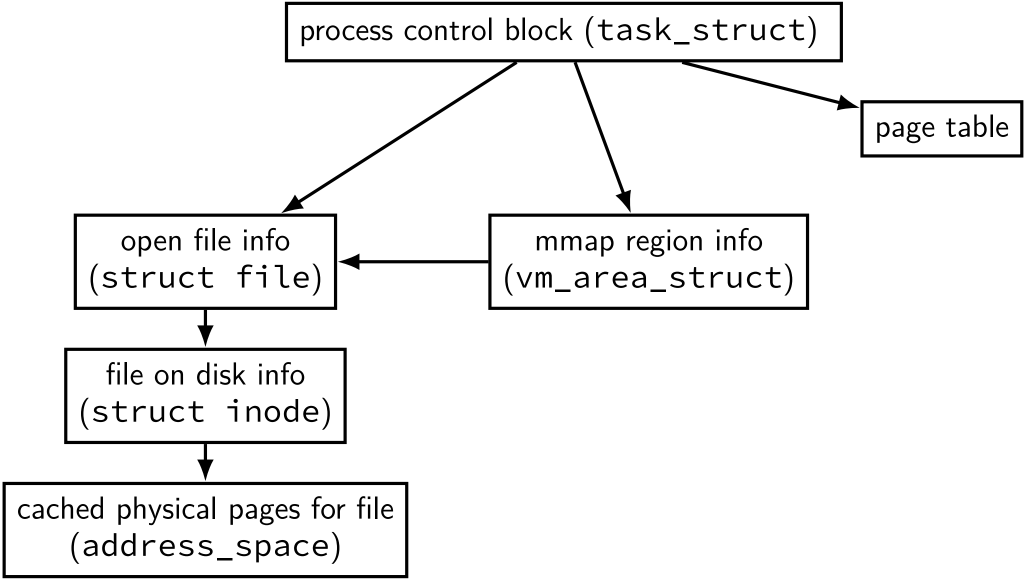
kernel data structure
for cache hit on read/write
(or page fault for mmap'd memory)
multiple designs; one idea: balanced tree

OS datastructure

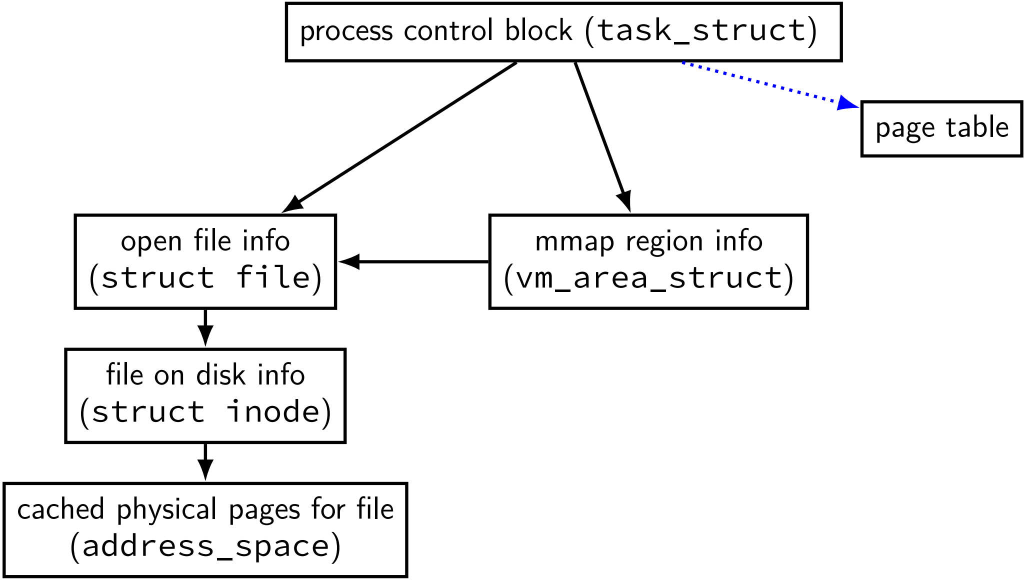
file + offset

(for read()/write())

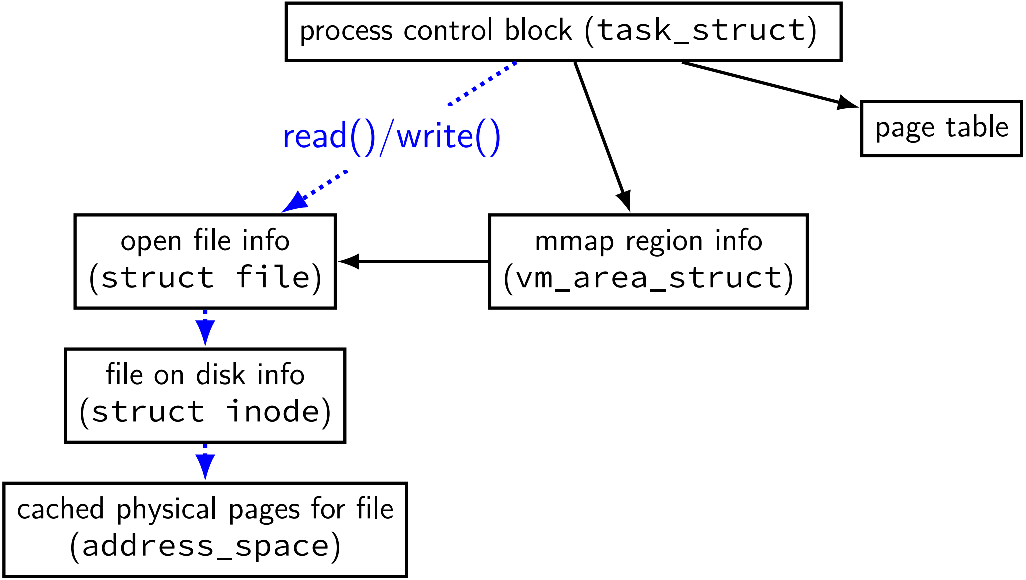
Linux: forward mapping


process control block (task_struct)
page table
open file info (struct file)
mmap region info (vm_area_struct)
file on disk info (struct inode)
cached physical pages for file (address_space)

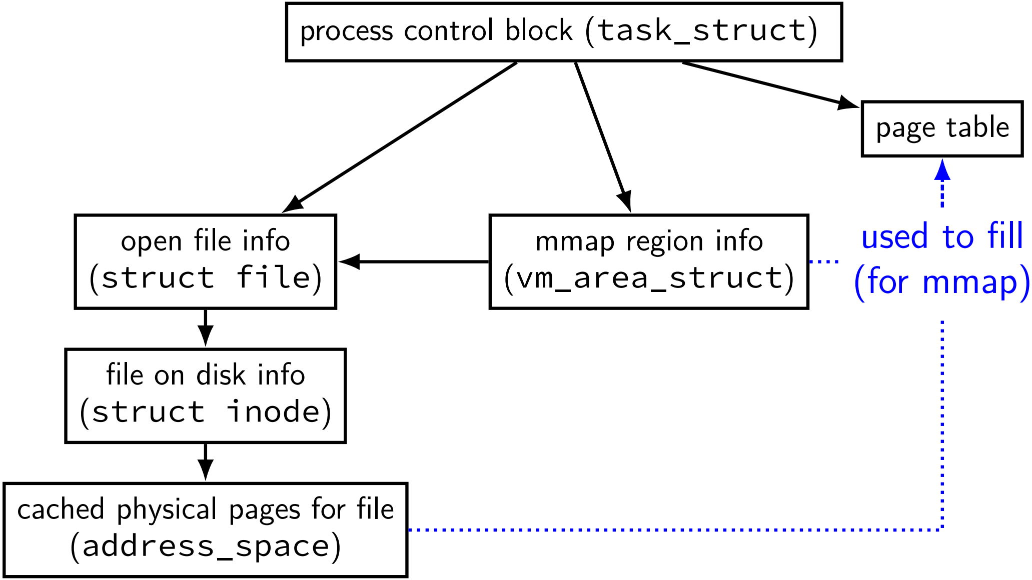
Linux: forward mapping


process control block (task_struct)
page table
open file info (struct file)
mmap region info (vm_area_struct)
file on disk info (struct inode)
cached physical pages for file (address_space)

Linux: forward mapping



Linux: forward mapping



process control block (task_struct)

page table

open file info (struct file)

mmap region info (vm_area_struct)

used to fill (for mmap)

file on disk info (struct inode)

cached physical pages for file (address_space)

minor and major faults

minor page fault

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

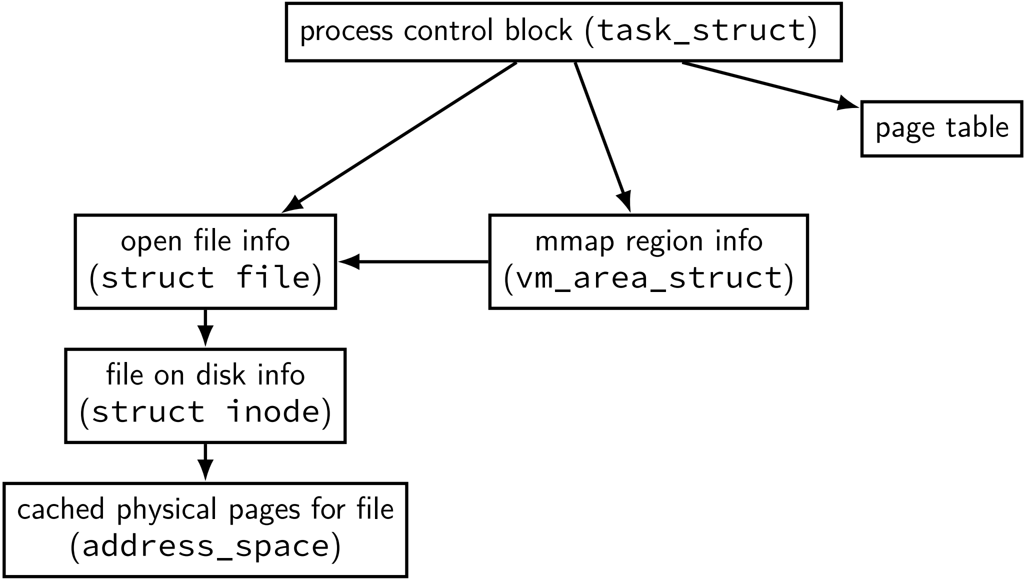
major page fault

- 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



process control block (`task_struct`)

page table

open file info (`struct file`)

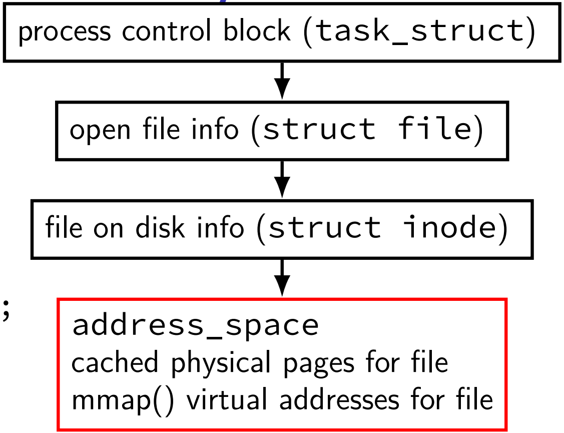
mmap region info (`vm_area_struct`)

file on disk info (`struct inode`)

cached physical pages for file (`address_space`)

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 mapp
    struct rb_root_cached   i_mmap;         /* tree of private and s
    ...
```



process control block (task_struct)

open file info (struct file)

file on disk info (struct inode)

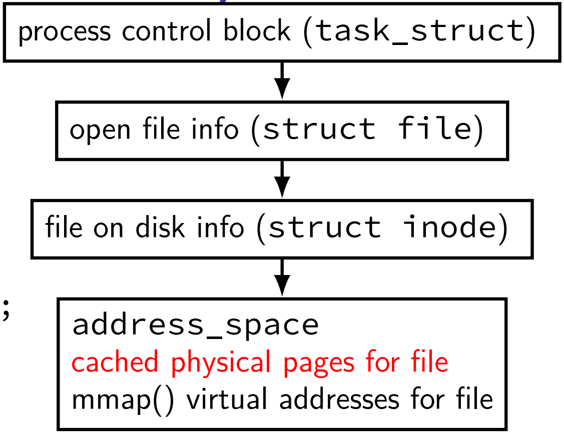
address_space

cached physical pages for file

mmap() virtual addresses for file

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 mapp
    struct rb_root_cached   i_mmap;         /* tree of private and s
    ...
```



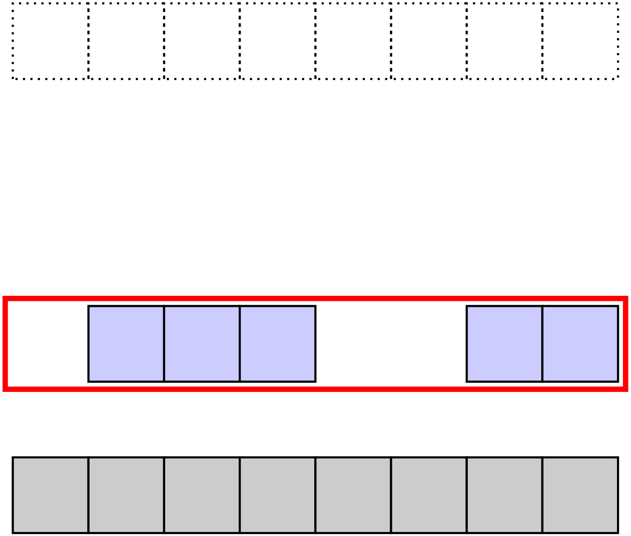
process control block (task_struct)

open file info (struct file)

file on disk info (struct inode)

address_space
cached physical pages for file
mmap() virtual addresses for file

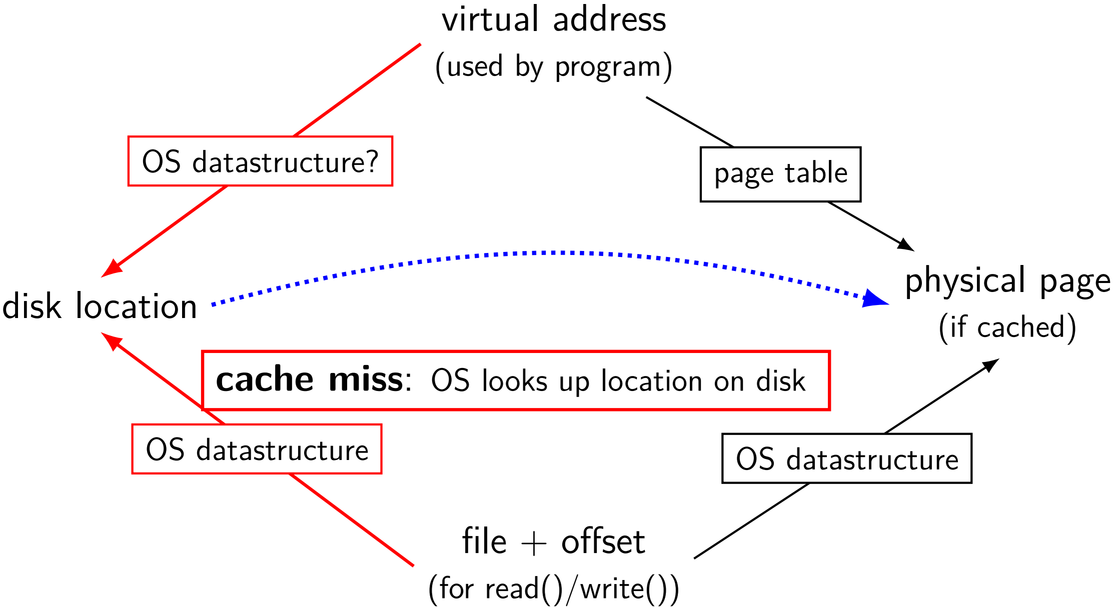
mapped pages (read/write, shared)



file data, cached in memory

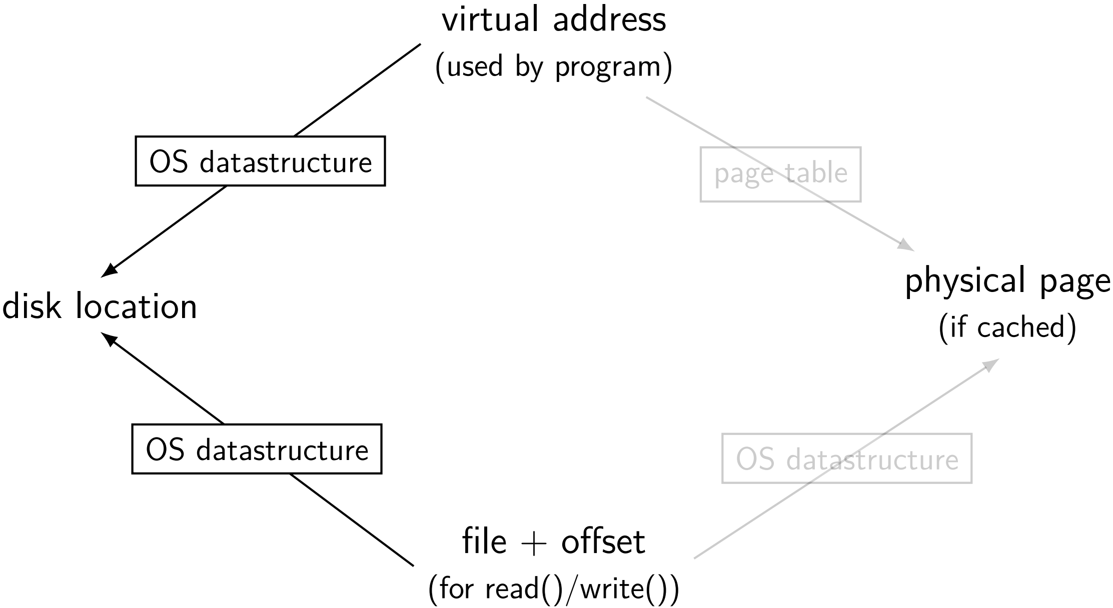
file data on disk/SSD

page cache components



cache miss: OS looks up location on disk

virtual address/file offset → location on disk



virtual address

(used by program)

OS datastructure

page table

disk location

physical page

(if cached)

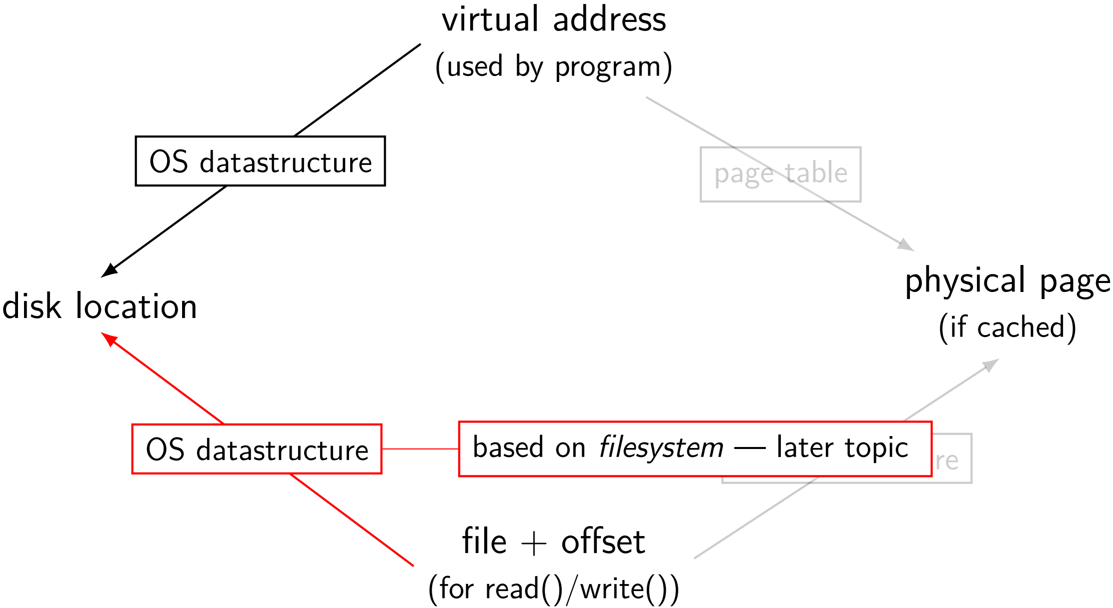
OS datastructure

OS datastructure

file + offset

(for read()/write())

virtual address/file offset → location on disk



virtual address

(used by program)

OS datastructure

page table

disk location

physical page

(if cached)

OS datastructure

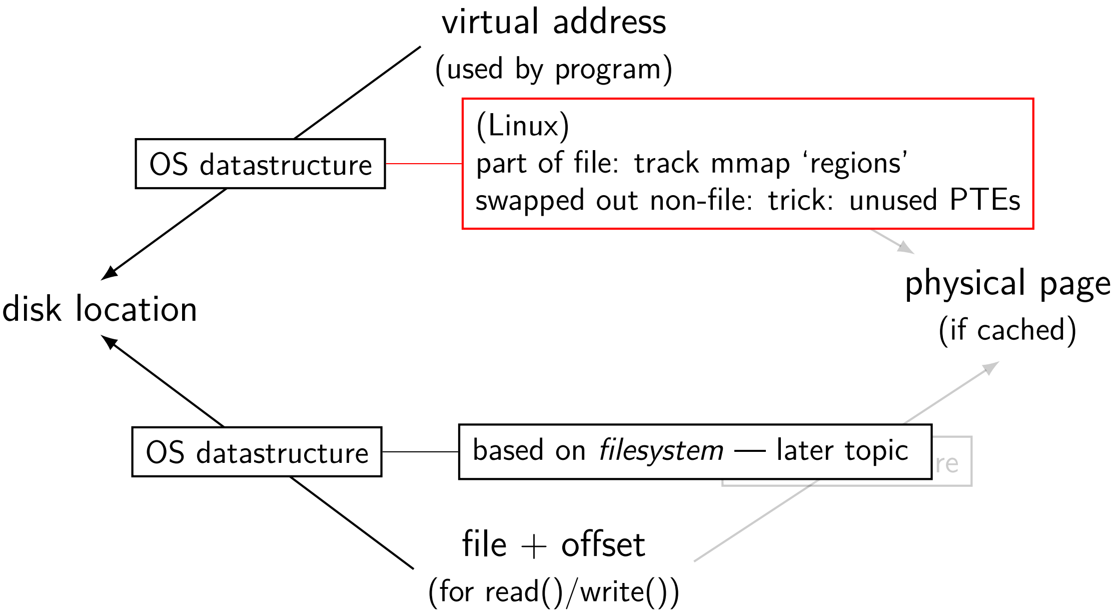
based on *filesystem* — later topic

re

file + offset

(for read()/write())

virtual address/file offset → location on disk



virtual address

(used by program)

OS datastructure

(Linux)
part of file: track mmap 'regions'
swapped out non-file: trick: unused PTEs

disk location

physical page

(if cached)

OS datastructure

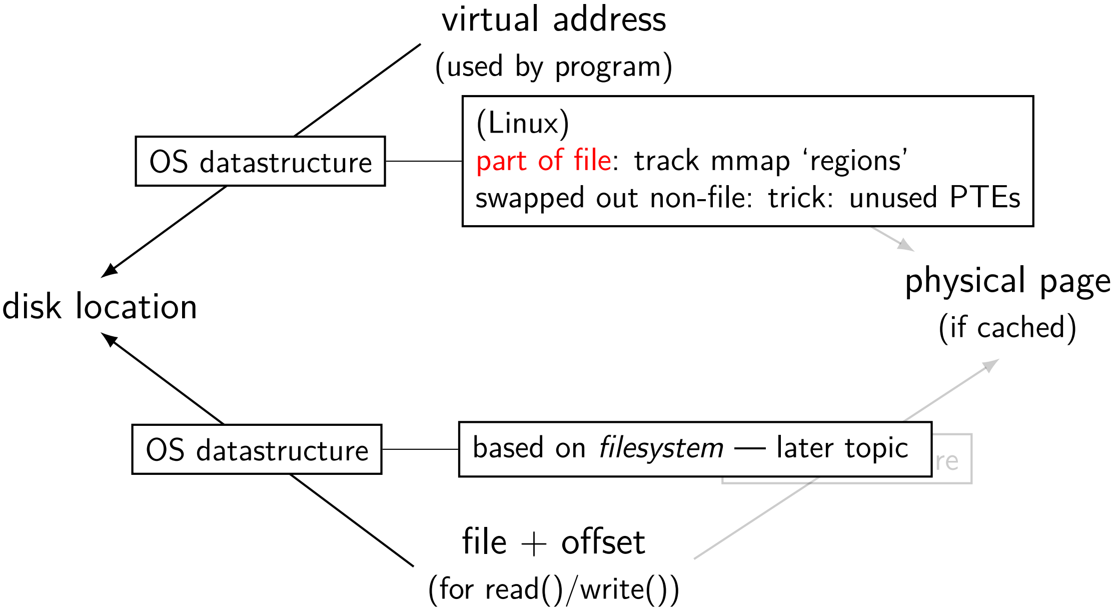
based on *filesystem* — later topic

re

file + offset

(for read()/write())

virtual address/file offset → location on disk



virtual address

(used by program)

OS datastructure

(Linux)

part of file: track mmap 'regions'

swapped out non-file: trick: unused PTEs

physical page

(if cached)

disk location

OS datastructure

based on *filesystem* — later topic

re

file + offset

(for read()/write())

recall: Linux maps

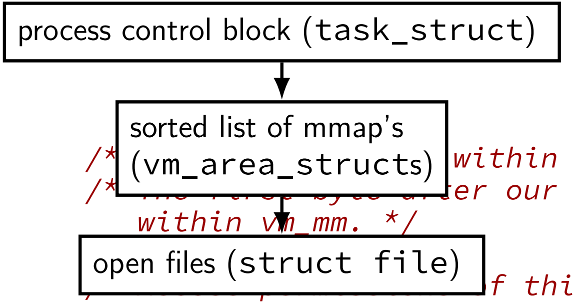
```
$ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 08:01 48328831          /bin/cat
0060a000-0060b000 r--p 0000a000 08:01 48328831          /bin/cat
0060b000-0060c000 rw-p 0000b000 08:01 48328831          /bin/cat
01974000-01995000 rw-p 00000000 00:00 0                 [heap]
7f60c718b000-7f60c7490000 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.s
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.s
7f60c7a7c000-7f60c7a7d000 rw-p 00023000 08:01 96659109  /lib/x86_64-linux-gnu/ld-2.19.s
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



process control block (task_struct)

sorted list of mmap's (vm_area_structs)

open files (struct file)

```
struct vm_area_struct { ...
    unsigned long vm_start;        /*           within vm_m
    unsigned long vm_end;          /*              er our end
                                      within vm_mm. */

    ...
    pgprot_t vm_page_prot;                           f 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

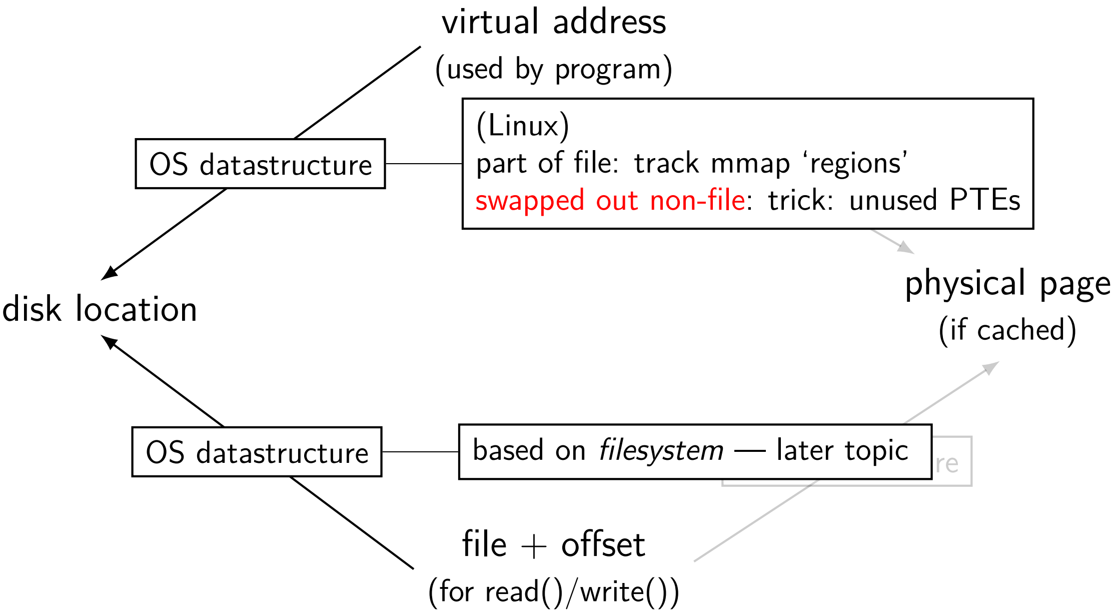
flags: private or shared? ...

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



virtual address
(used by program)

OS datastructure

(Linux)
part of file: track mmap 'regions'
swapped out non-file: trick: unused PTEs

physical page
(if cached)

disk location

OS datastructure

based on *filesystem* — later topic

re

file + offset
(for read()/write())

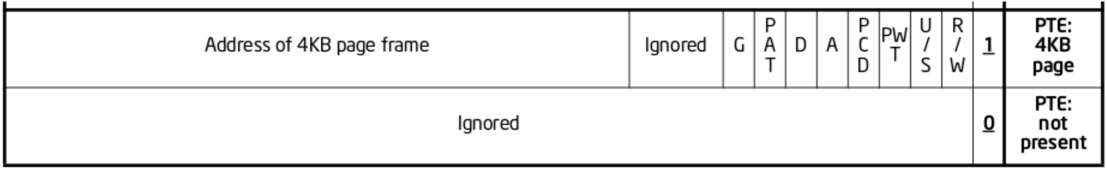
Linux: tracking swapped out pages

need to lookup location on disk

potentially one location for every virtual page

trick: store location in “ignored” part of page table entry

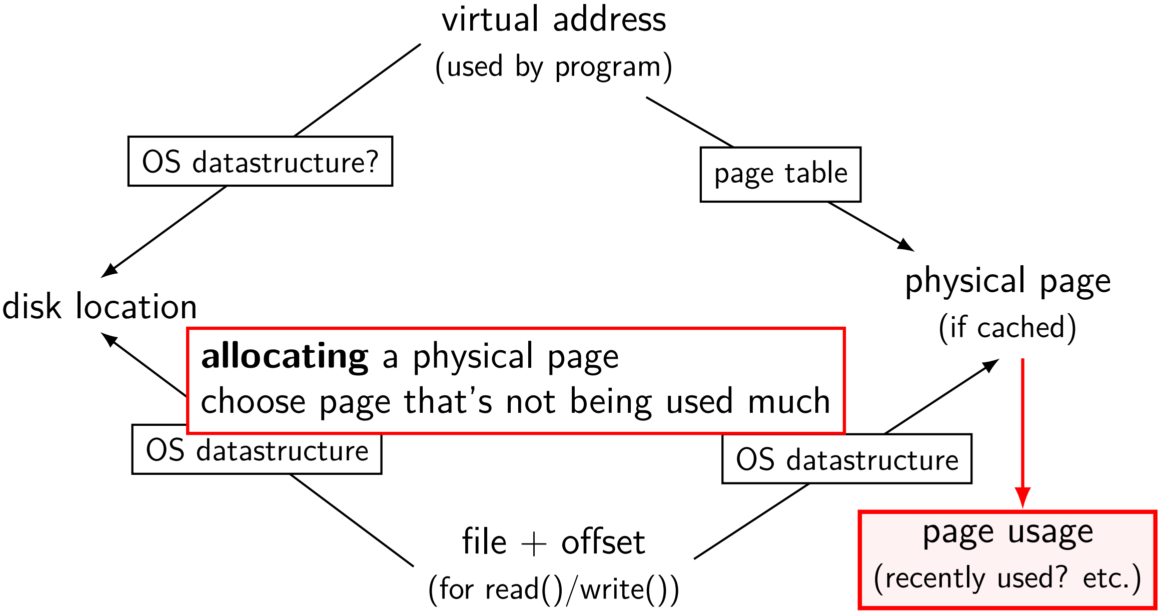
instead of physical page #, permission bits, etc., store offset on disk



Address of 4KB page frame	Ignored	G	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

page cache components



allocating a physical page

choose page that's not being used much

tracking physical pages: finding free pages

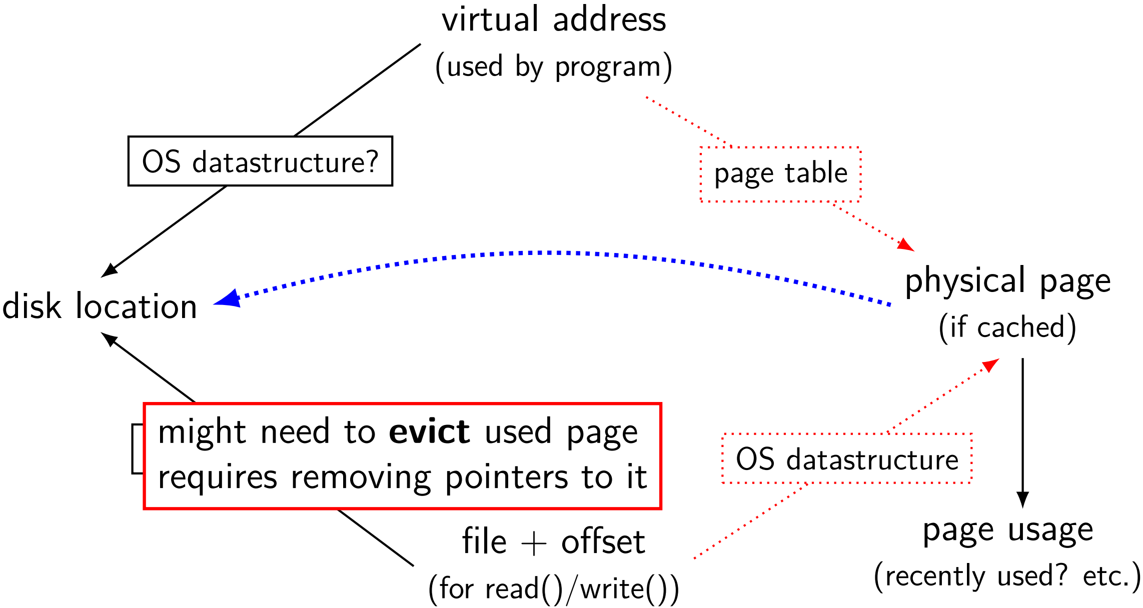
Linux has list of “least recently used” pages:

```
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



virtual address

(used by program)

OS datastructure?

page table

disk location

physical page

(if cached)

might need to **evict** used page

requires removing pointers to it

OS datastructure

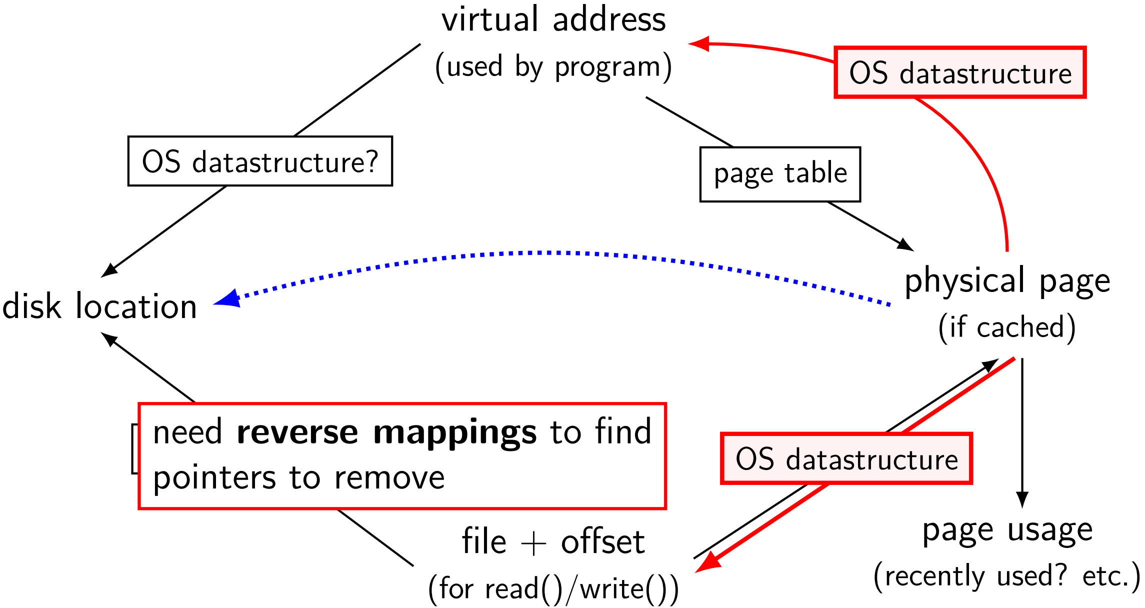
file + offset

(for read()/write())

page usage

(recently used? etc.)

page cache components



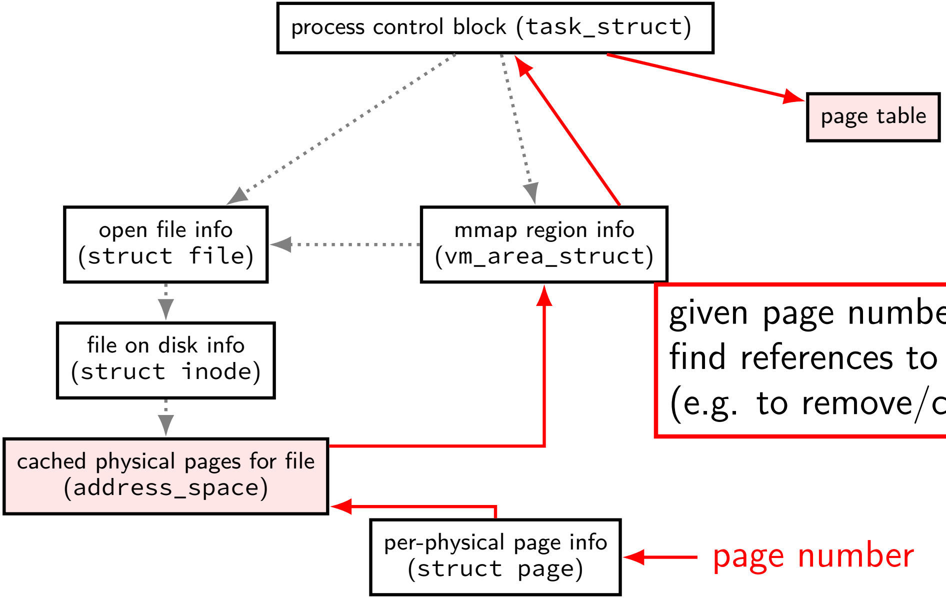
need **reverse mappings** to find pointers to remove

tracking physical pages: finding mappings

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

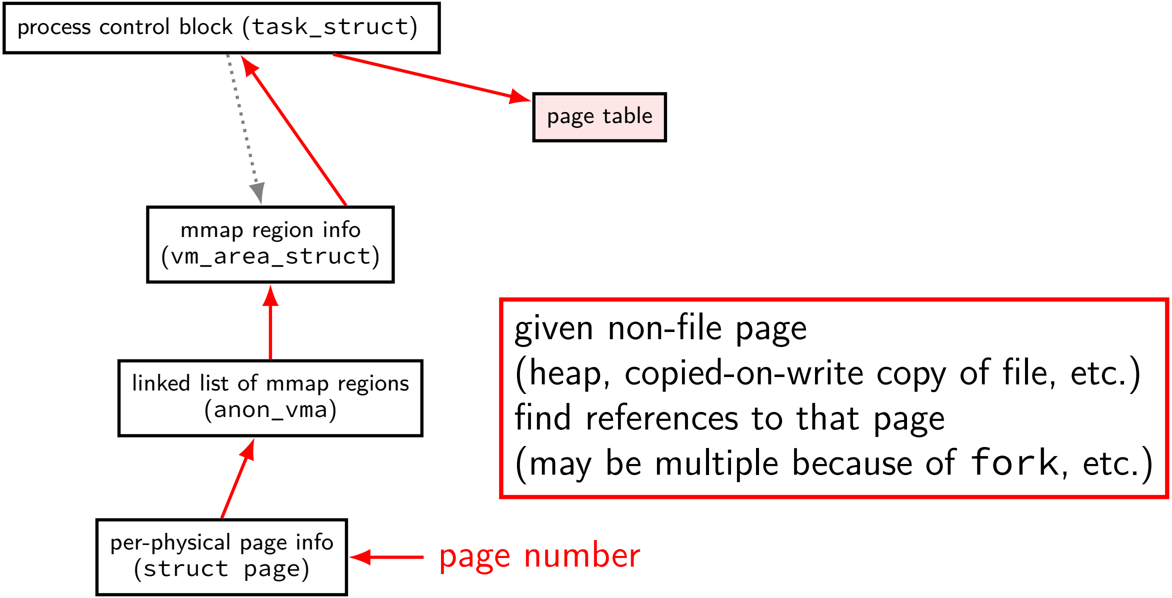
need to track where every page is used!

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

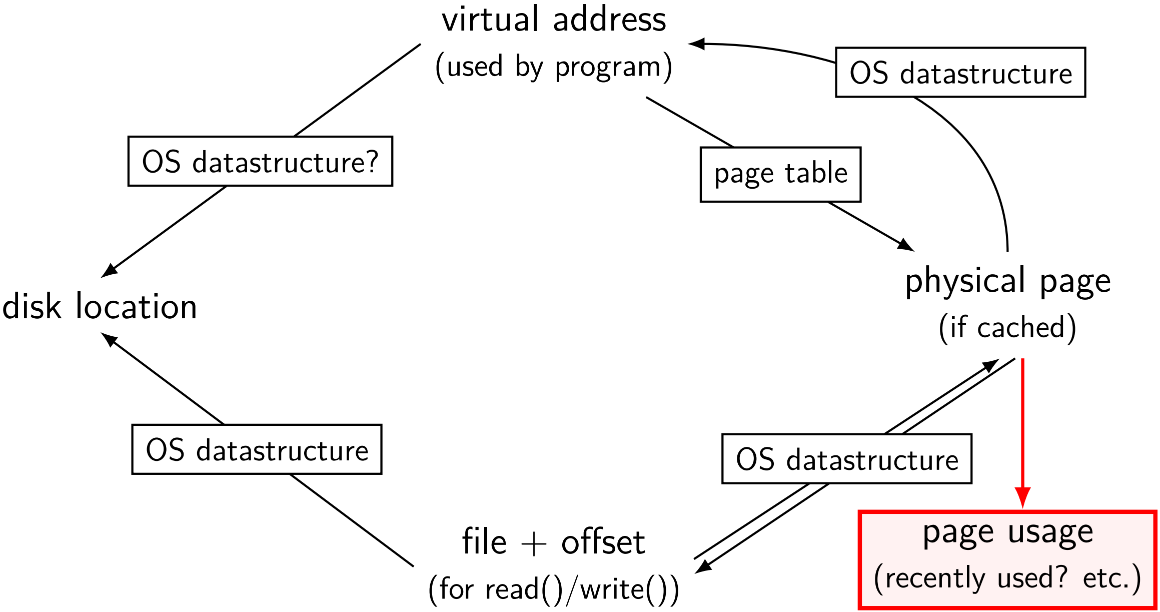
find a ‘victim’ page to evict

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 ≈ max throughput

optimizing hit rate almost optimizes throughput, but…

max hit rate ≈ 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

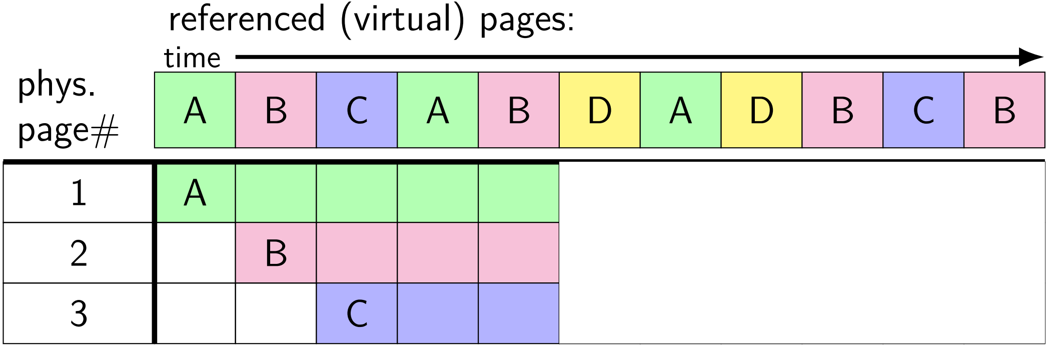
best possible page replacement algorithm: Belady's MIN

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

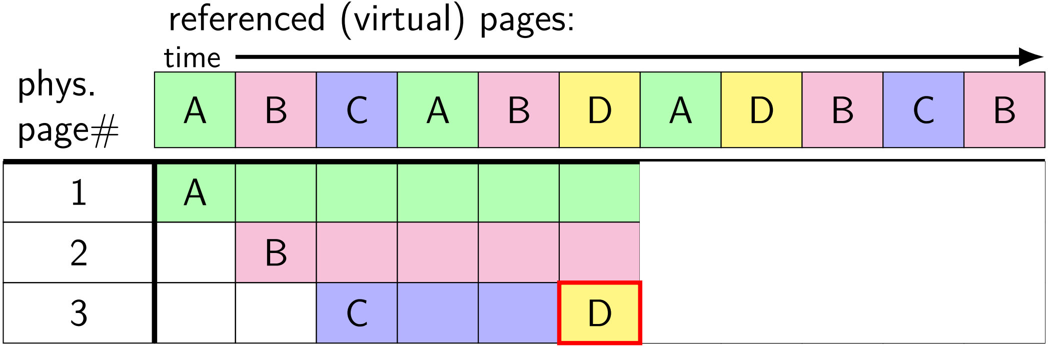


referenced (virtual) pages:

time →

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

Belady's MIN



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

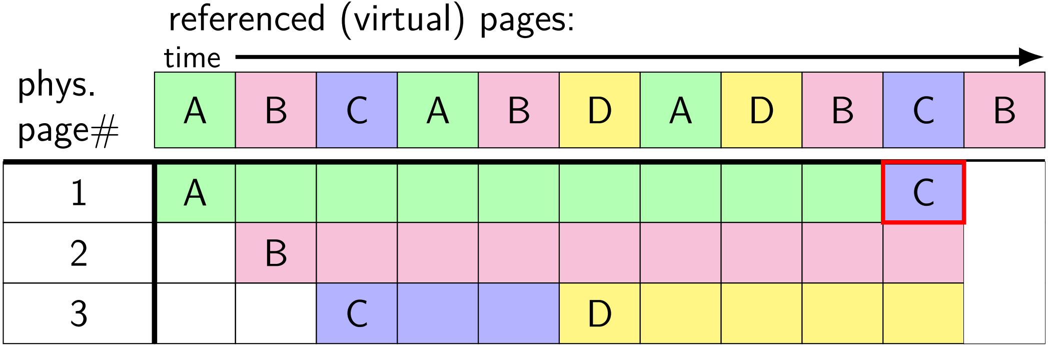
Belady's MIN

referenced (virtual) pages:

time →

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

Belady's MIN

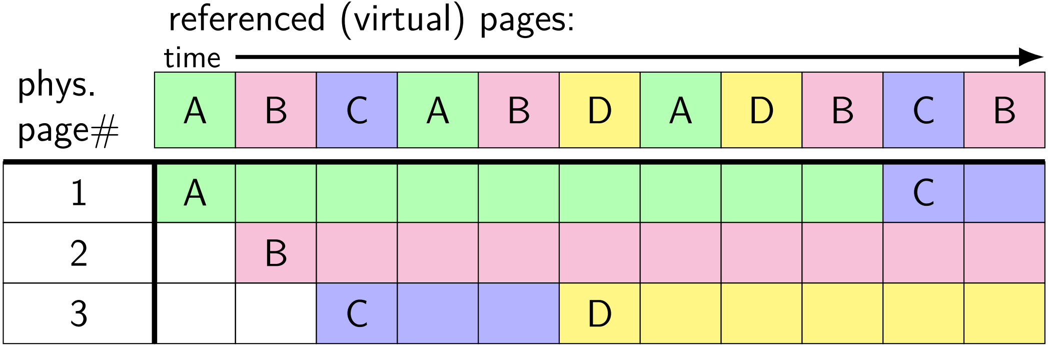


referenced (virtual) pages:

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

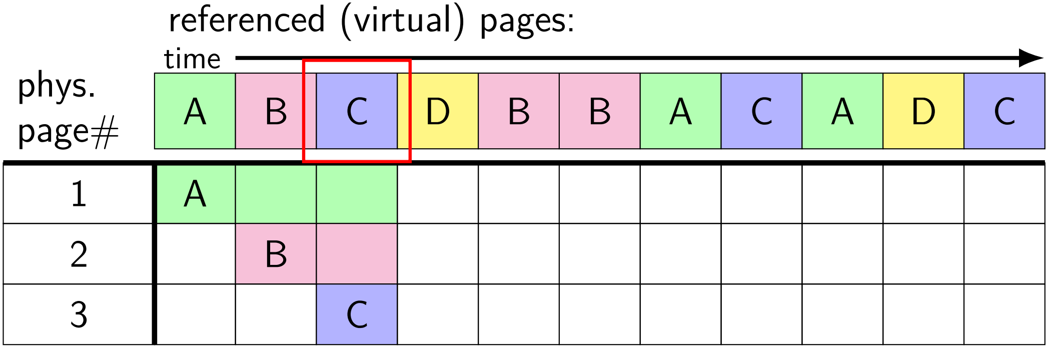


referenced (virtual) pages:

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



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

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

…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

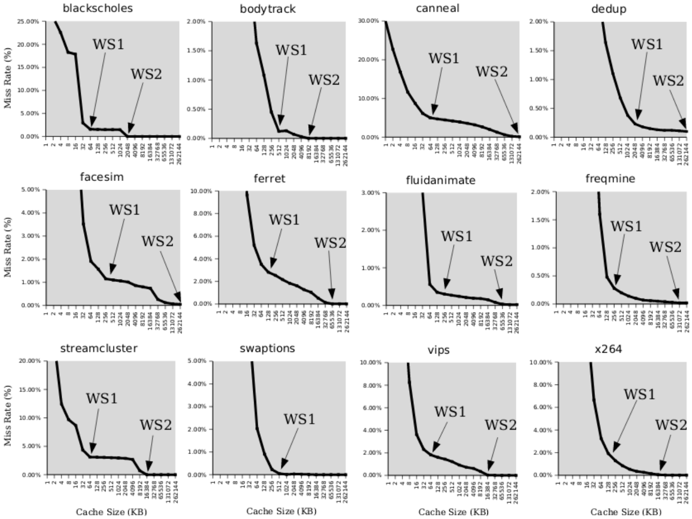


Figure 3: Miss rates versus cache size. Data assumes a shared 4-way associative cache with 64 byte lines. WS1 and WS2 refer to important working sets which we analyze in more detail in Table 2. Cache requirements of PARSEC benchmark programs can reach hundreds of megabytes.

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

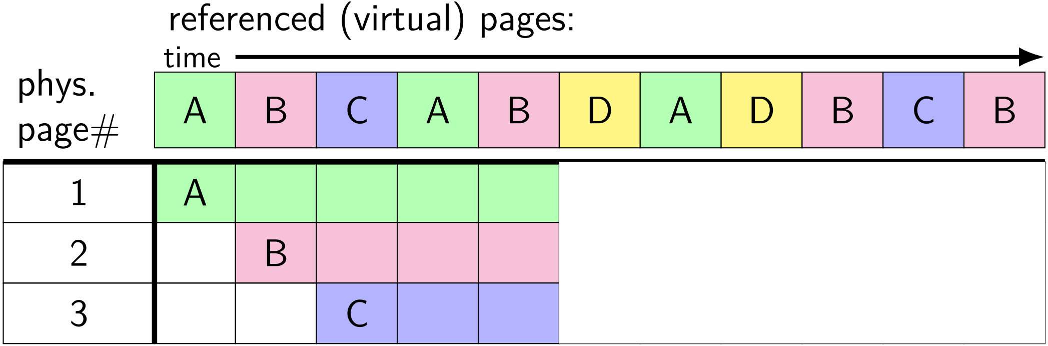
recall?: locality assumption

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)

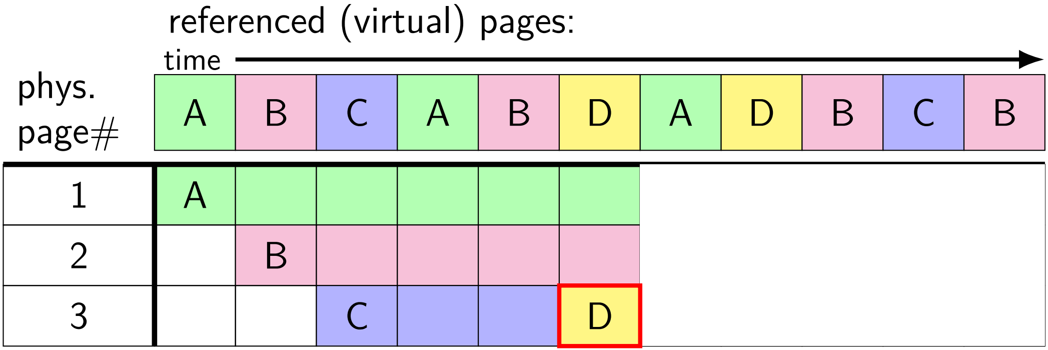


referenced (virtual) pages:

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:

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

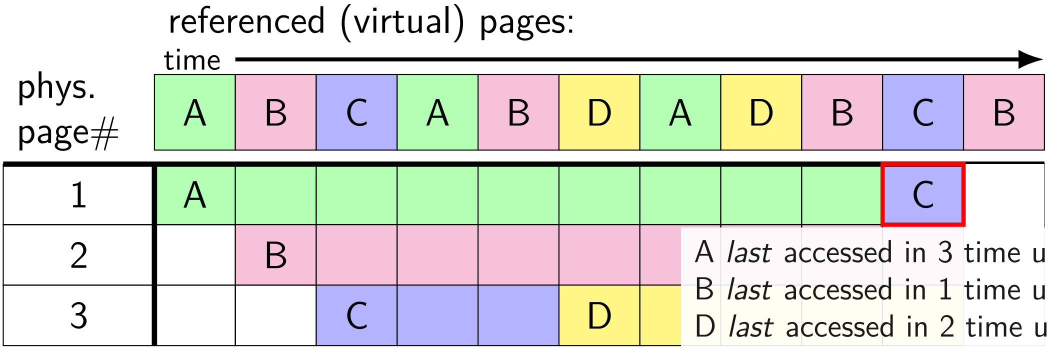
least recently used (the good case)

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					

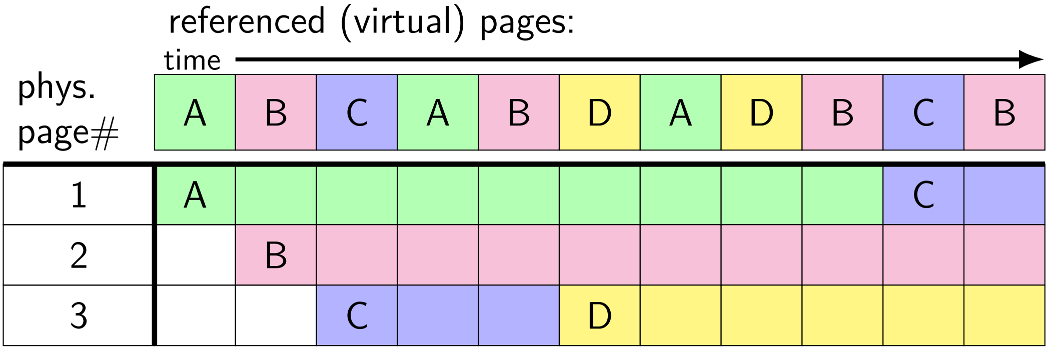
A *last* accessed in 3 time units ago

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)

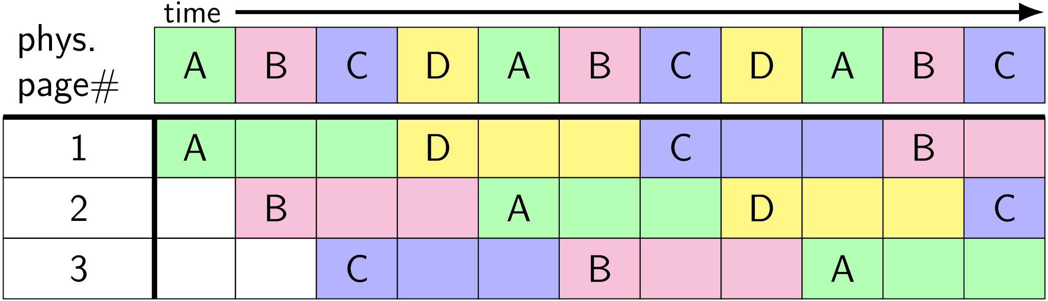


referenced (virtual) pages:

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)

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) (1)

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

least recently used (exercise) (2)

	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

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

wheneve

- 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

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

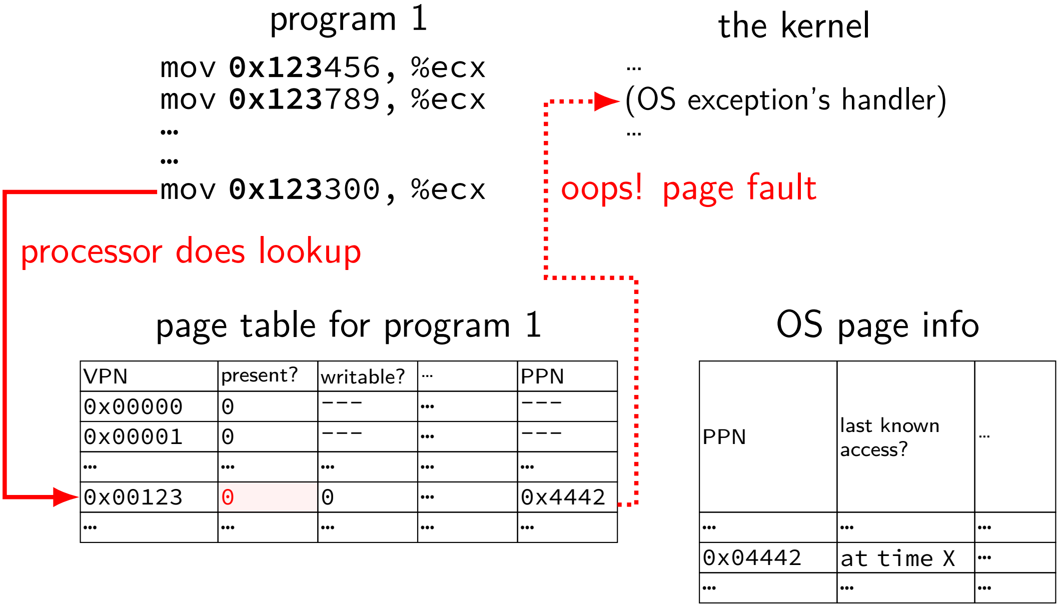
page table for program 1

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
```

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
```

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

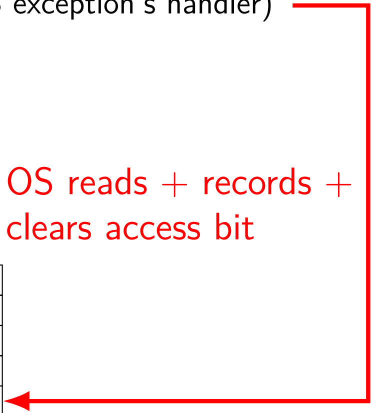
```
mov 0x123456, %ecx
mov 0x123789, %ecx
…
…
mov 0x123300, %ecx
```

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
```

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**

dirty bits

“was this part of the mmap’d file changed?”

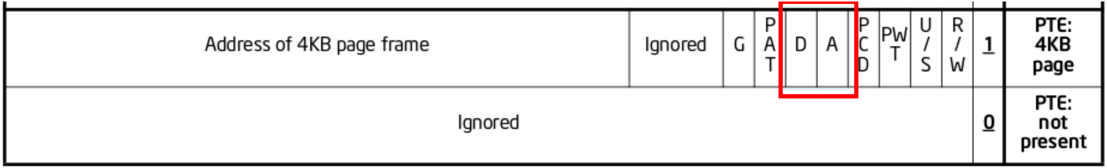
“is the old swapped copy still up to date?”

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

backup slides

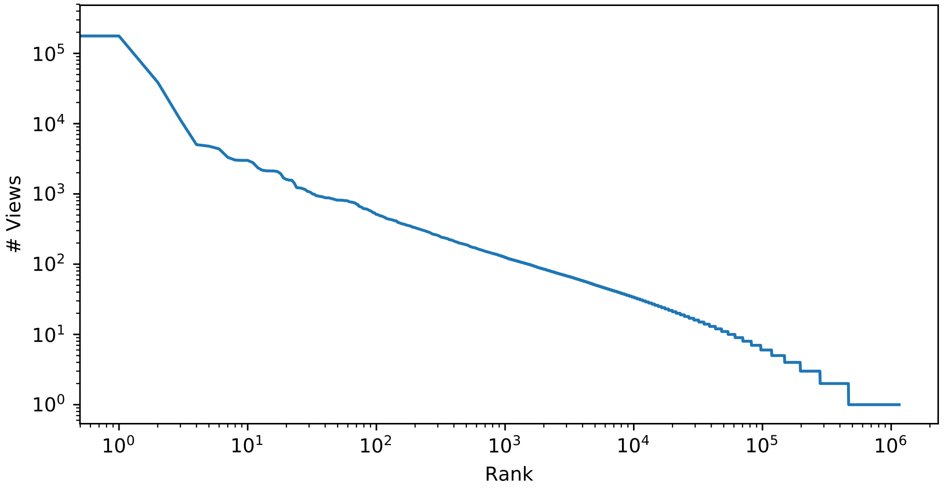
aside: Zipf model

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

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" ≈ 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

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:...

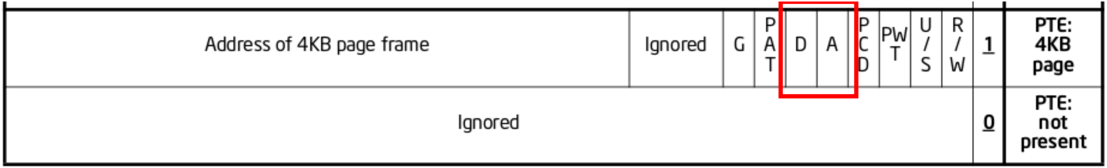
software-only: temporarily set page table entry invalid

- 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	<u>1</u>	**PTE: 4KB page**
Ignored										<u>0</u>	**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?

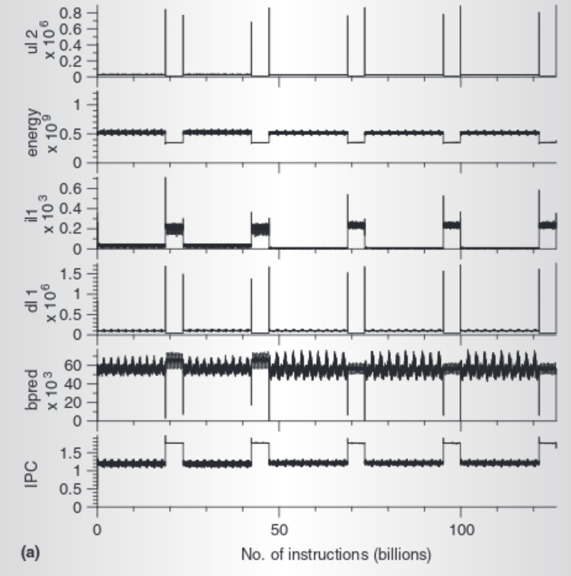
e.g. finish parsing input, now process it

phase change — discard one working set, gain another

phase changes likely to have spike of cache misses

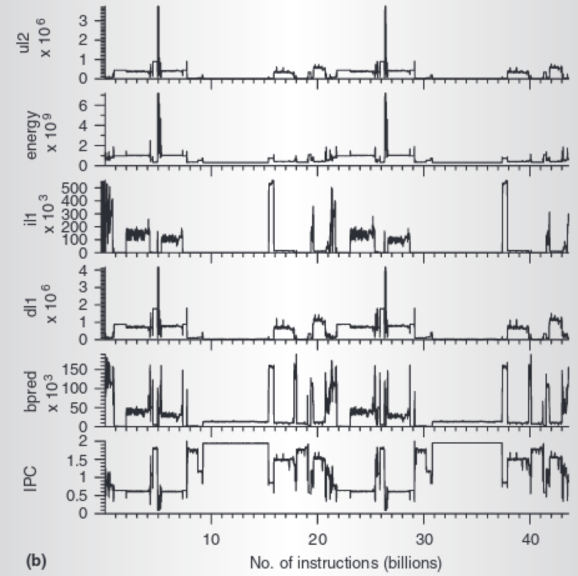
- whatever was cached, not what's being accessed anymore
- maybe along with change in kind of instructions being run

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