Changelog

Changes not seen in first lecture:

- 19 March 2020: move page usage slides later
- 19 March 2020: adjust PF counting exercise to specify addreses, not offsets
- 19 March 2020: Linux maps: correct shown mmap call for 0x400000

virtual memory 3

Zoom logistics

recommend: exit full screen

open chat + participants window

participants window has non-verbal feedback features

I will try to monitor the chat window

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

but probably text is usually easier/more reliable?

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

general logistics

lectures streamed via Zoom with questions

videos + audio-recordings + slides available

if you have trouble getting at anything, let us know

please use Piazza

office hours via Discord with queue

quizzes still happening

last time

virtual memory — two-level tables

page fault handling

- return from page fault normally → retry instruction
- trick: fix page table before returning

allocate-on-demand

- pretend to allocate right away
- actually allocate later (on use)

copy-on-write

- pretend to copy right away
- actually allocate later (on write)

xv6: adding space on demand

```
struct proc {
  uint sz;     // Size of process memory (bytes)
  ...
};
```

xv6 tracks "end of heap" (now just for `sbrk()`)

adding allocate on demand logic for the heap:

on sbrk(): don't change page table right away

on page fault

- case 1: if address $\geq$ sz: out of bounds: kill process
- case 2: otherwise, allocate page containing address, return from trap

versus more complicated OSes

typical desktop/server:
range of valid addresses is not just 0 to maximum

need some more complicated data structure to represent

copy-on write cases

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

- kill program instead of making it writable

fault from trying to write read-only page:

case 1: multiple process's page table entries refer to it

- copy the page
- replace read-only page table entry to point to copy

case 2: only one page table entry refers to it

- make it writeable

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`

(Linux extension: can omit fd with special value of `flags`)

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`

(Linux extension: can omit fd with special value of `flags`)

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`

(Linux extension: can omit fd with special value of `flags`)

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 one of:

MAP_SHARED — changing memory changes file and vice-versa

- multiple processes mmap same file: get same physical pages
- read()/write() must use same physical pages
- changes to memory (if writable) must be sent to disk eventually

MAP_PRIVATE — make a copy of data in file

- changes to memory do not change file
- almost as if copied during `mmap` call
- but probably actually copied using copy-on-write

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 one of:

MAP_SHARED — changing memory changes file and vice-versa

- multiple processes mmap same file: get same physical pages
- read()/write() must use same physical pages
- changes to memory (if writable) must be sent to disk eventually

MAP_PRIVATE — make a copy of data in file

- changes to memory do not change file
- almost as if copied during `mmap` call
- but probably actually copied using copy-on-write

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 one of:

MAP_SHARED — changing memory changes file and vice-versa

- multiple processes mmap same file: get same physical pages
- read()/write() must use same physical pages
- changes to memory (if writable) must be sent to disk eventually

MAP_PRIVATE — make a copy of data in file

- changes to memory do not change file
- almost as if copied during `mmap` call
- but probably actually copied using copy-on-write

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 one of:

MAP_SHARED — changing memory changes file and vice-versa

- multiple processes mmap same file: get same physical pages
- read()/write() must use same physical pages
- changes to memory (if writable) must be sent to disk eventually

MAP_PRIVATE — make a copy of data in file

- changes to memory do not change file
- almost as if copied during `mmap` call
- but probably actually copied using copy-on-write

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 one of:

MAP_SHARED — changing memory changes file and vice-versa

- multiple processes mmap same file: get same physical pages
- read()/write() must use same physical pages
- changes to memory (if writable) must be sent to disk eventually

MAP_PRIVATE — make a copy of data in file

- changes to memory do not change file
- almost as if copied during `mmap` call
- but probably actually copied using copy-on-write

mmap options (3)

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

`flags`, choose one of:

MAP_SHARED — changing memory changes file and vice-versa

MAP_PRIVATE — make a copy of data in file

...or'd with optional additonal flags

Linux: MAP_ANONYMOUS — ignore `fd`, allocate empty space

- trick: Linux tracks process's memory as list of mmap's
- ...'normal' memory heap, just special case w/o file

and more (see manual page)

mmap options (4)

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

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

- not mandatory unless MAP_FIXED is used (which is rare)
- can pass NULL — "choose for me"
- address chosen will be returned
- MAP_FAILED (constant) on failure

mmap exercise

suppose `hello.txt` initially contains "`foo`":

```
int fd = open("hello.txt", O_RDWR);
char *p1 = mmap(NULL, 3 /* size */,
                PROT_READ|PROT_WRITE,
                MAP_SHARED, fd, 0);
char *p2 = mmap(NULL, 3, PROT_READ|PROT_WRITE, MAP_PRIVATE, fd, 0);
char *p3 = mmap(NULL, 3, PROT_READ, MAP_SHARED, fd, 0);
p2[2] = 'b';
p1[2] = 'x'; p1[1] = 'i';
char buffer[3];
read(fd, buffer, 3);
printf("%3s/%3s/%3s\n", buffer, p2, p3);
```

What is the output? (Assume no failures.)

A. `foo/fob/foo`
B. `fix/fob/foo`
C. `fix/fix/fix`
D. `fix/fob/fix`
E. `fix/fob/fob`
F. something else

mmap exercise

suppose `hello.txt` initially contains "`foo`":

```
int fd = open("hello.txt", O_RDWR);
char *p1 = mmap(NULL, 3 /* size */,
                PROT_READ|PROT_WRITE,
                MAP_SHARED, fd, 0);
char *p2 = mmap(NULL, 3, PROT_READ|PROT_WRITE, MAP_PRIVATE, fd, 0);
char *p3 = mmap(NULL, 3, PROT_READ, MAP_SHARED, fd, 0);
p2[2] = 'b';
p1[2] = 'x'; p1[1] = 'i';
char buffer[3];
read(fd, buffer, 3);
printf("%3s/%3s/%3s\n", buffer, p2, p3);
```

What is the output? (Assume no failures.)

A. `foo/fob/foo`
B. `fix/fob/foo`
C. `fix/fix/fix`
D. `fix/fob/fix`
E. `fix/fob/fob`
F. something else

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 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]
```

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 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]
```

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.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]
```

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

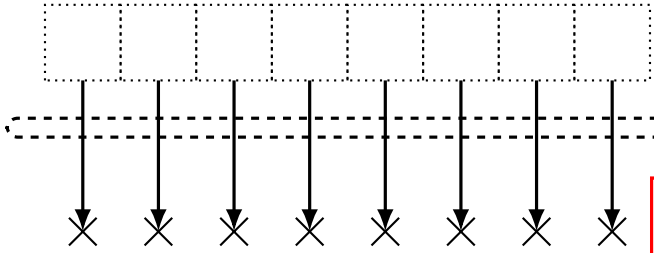
Linux maps

```
$ 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, 0xb000, PROT_READ | PROT_EXEC,
     MAP_PRIVATE, fd, 0x0);
```

mapped pages (read-only)



virtual pages mapped to file

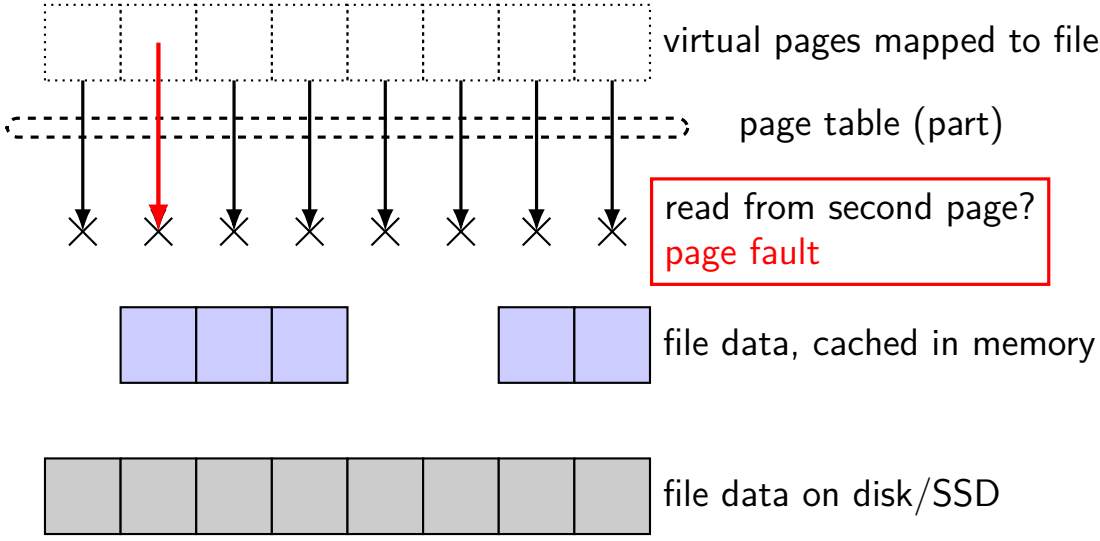
page table (part)

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

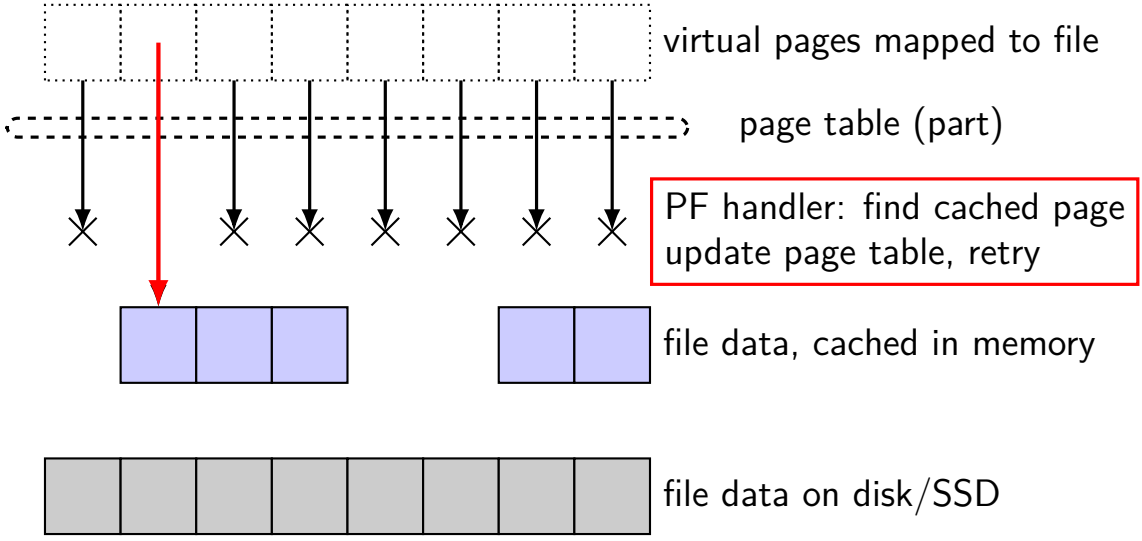
file data, cached in memory

file data on disk/SSD

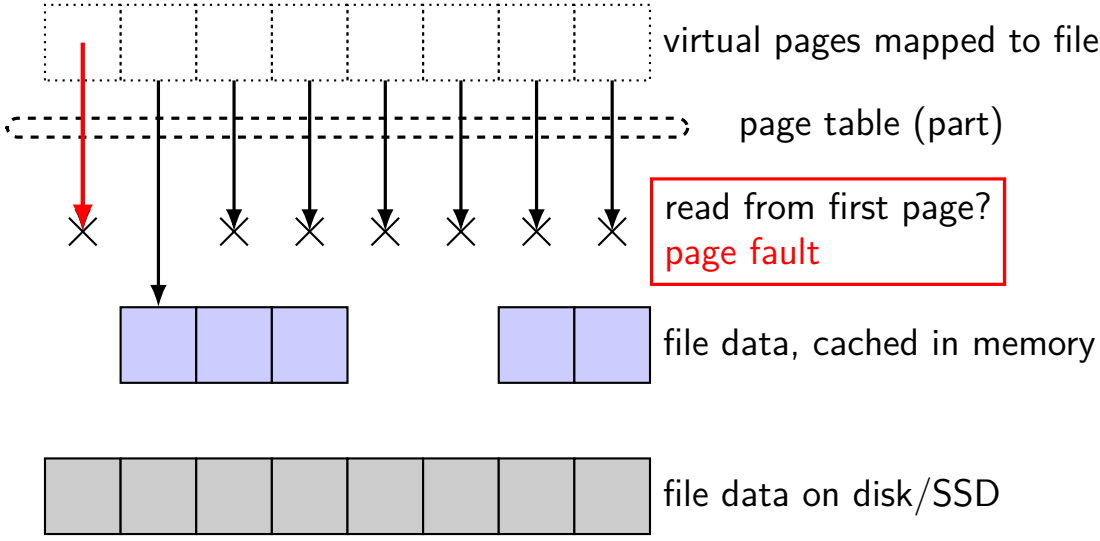
mapped pages (read-only)



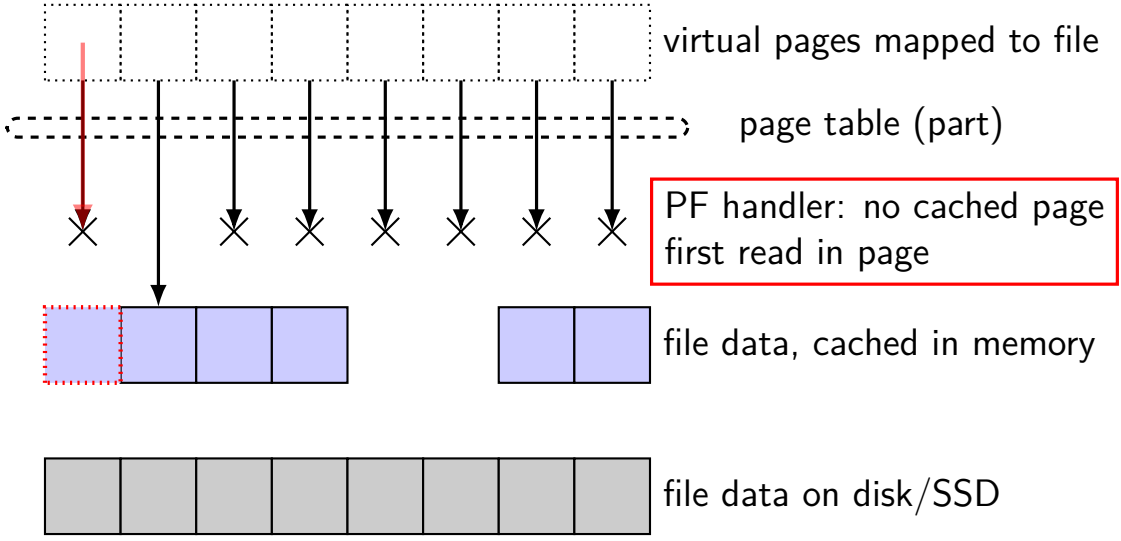
mapped pages (read-only)



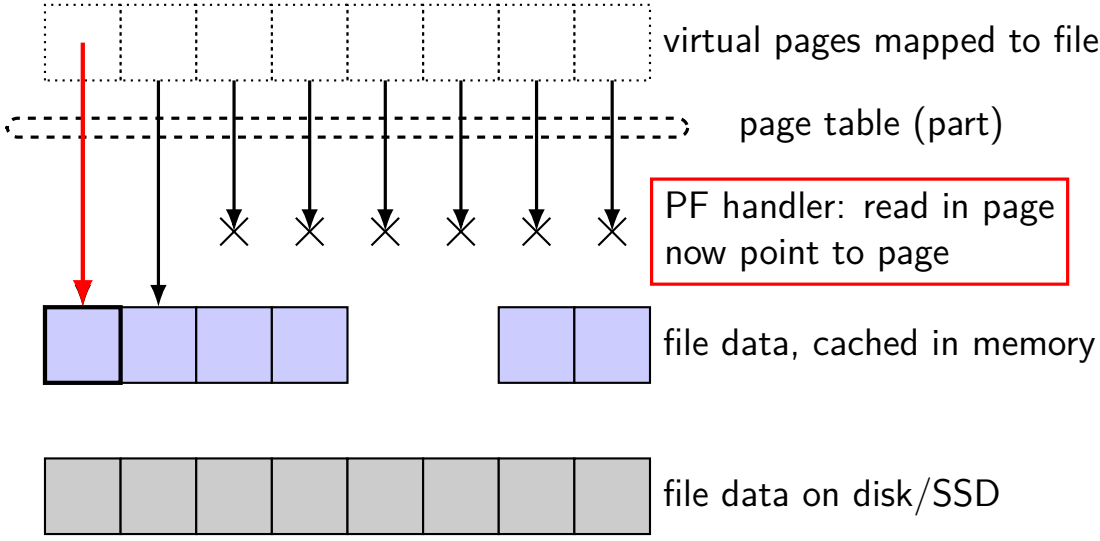
mapped pages (read-only)



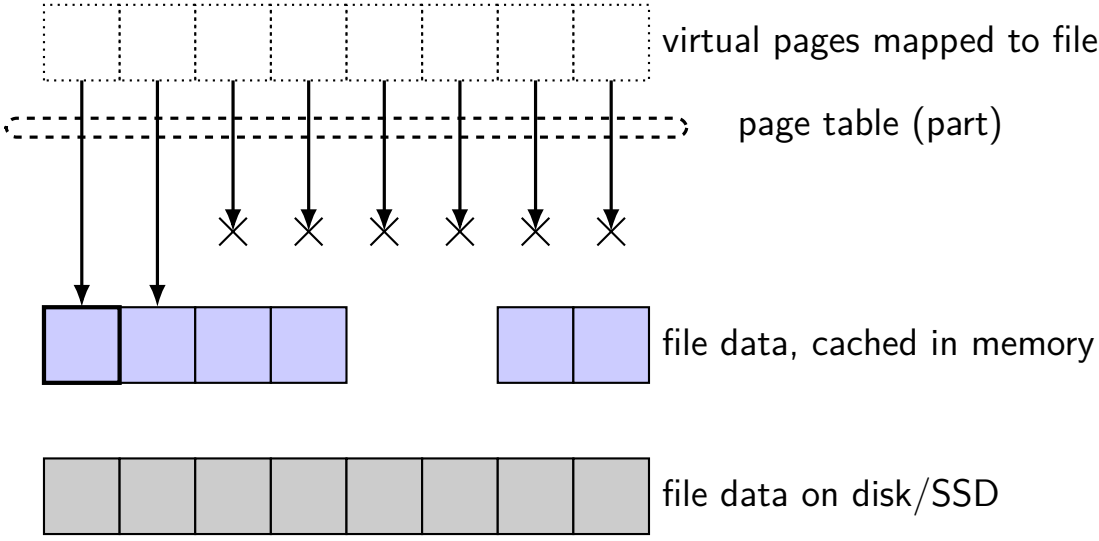
mapped pages (read-only)



mapped pages (read-only)



mapped pages (read-only)



shared mmap

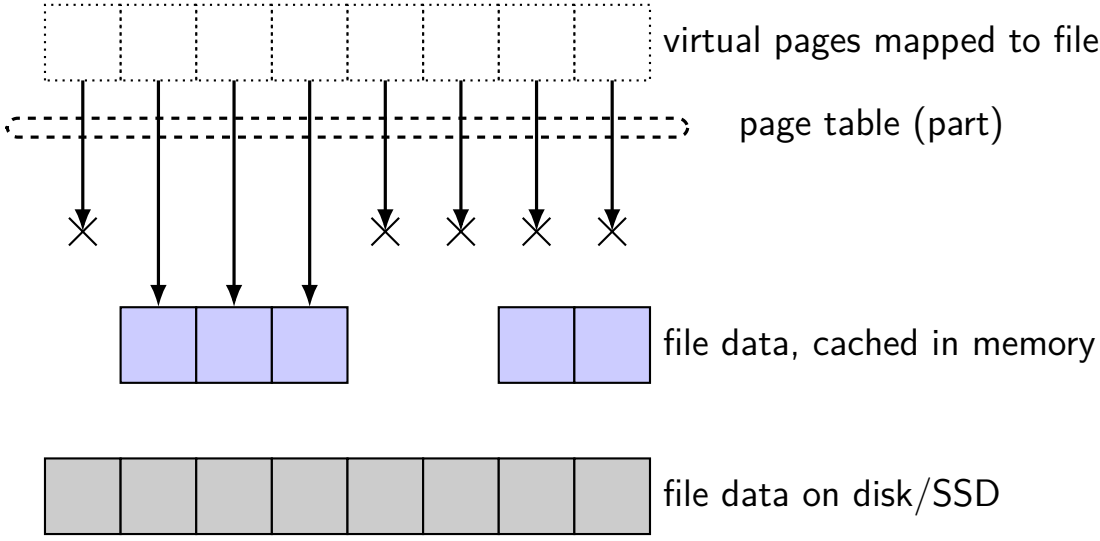
```
int fd  = open("/tmp/somefile.dat", O_RDWR);
mmap(0, 64 * 1024, PROT_READ | PROT_WRITE,
     MAP_SHARED, fd, 0);
```

---

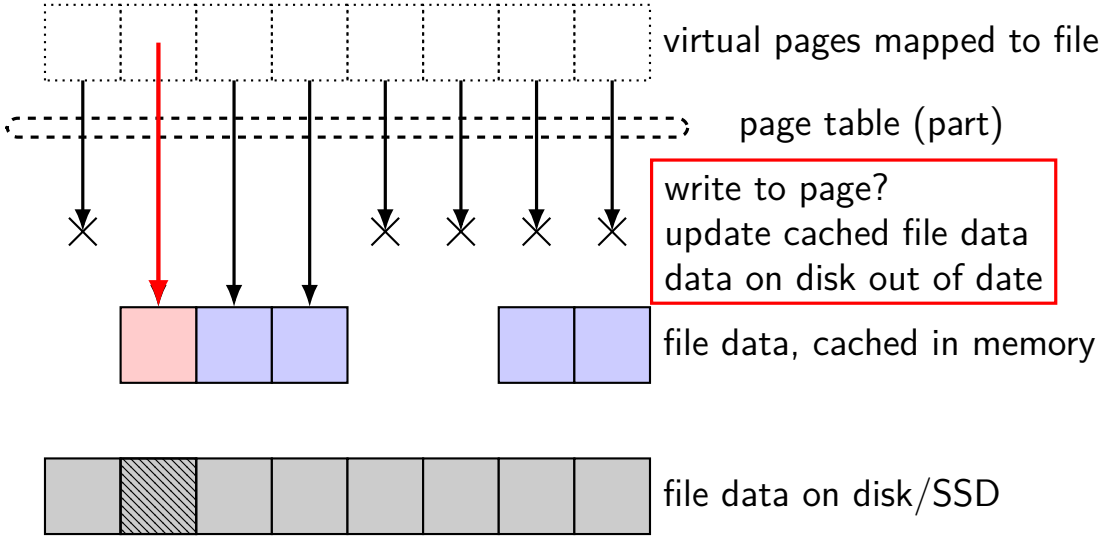
from /proc/PID/maps for this program:

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

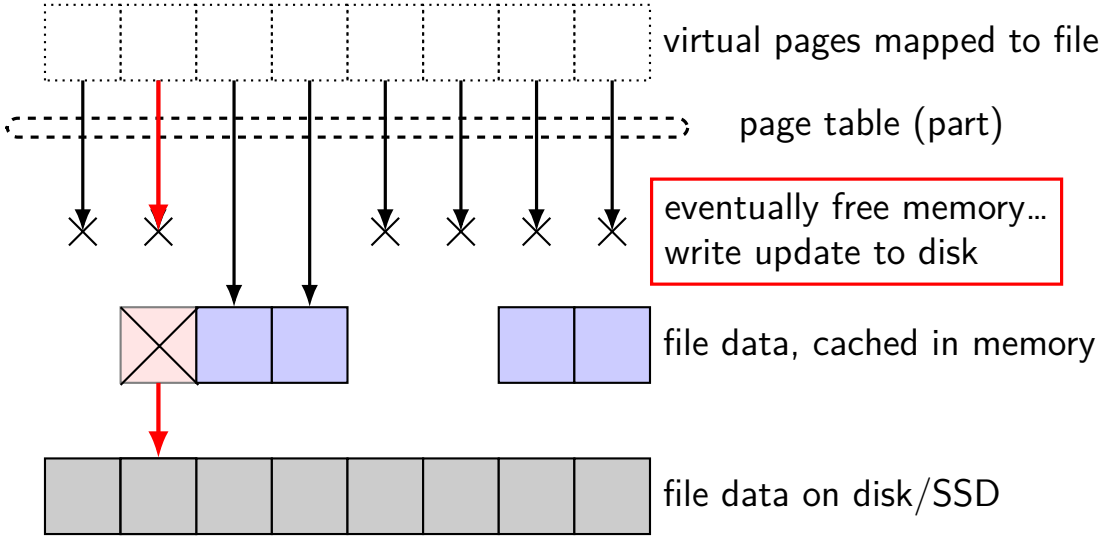
mapped pages (read/write, shared)



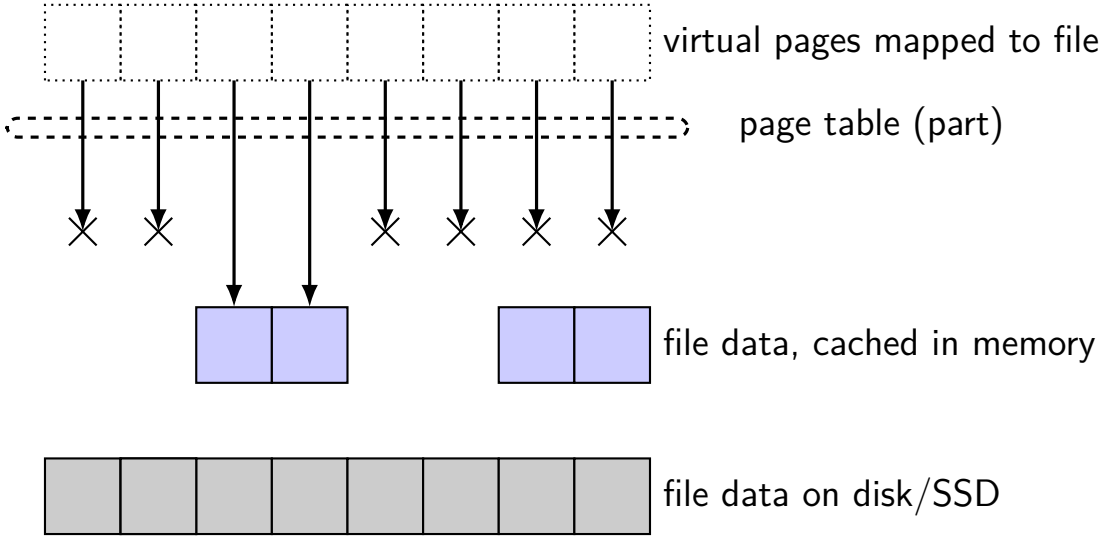
mapped pages (read/write, shared)



mapped pages (read/write, shared)



mapped pages (read/write, shared)



minor and major faults

minor page fault

- page is already in memory ("page cache")
- just fill in page table entry

major page fault

- page not already in memory ("page cache")
- need to allocate space
- possibly need to read data from disk/etc.

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 maps

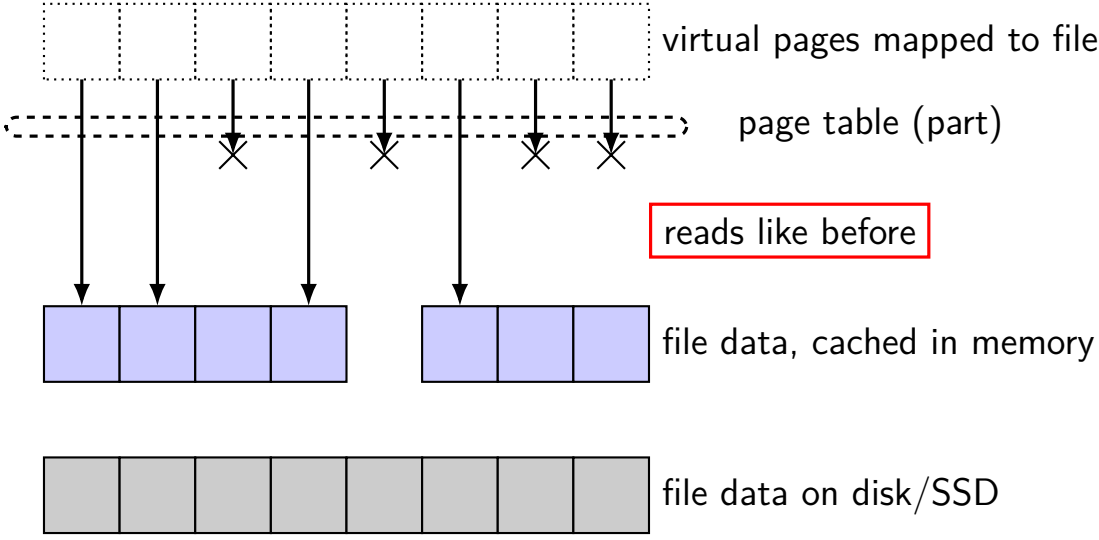
```
$ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 08:01 48328831                           /bin/cat
0060a000-0060b000 r--p 0000a000 08:01 48328831                           /bin/cat
0060b000-0060c000 rw-p 0000b000 08:01 48328831                           /bin/cat
01974000-01995000 rw-p 00000000 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                                                          -2.19
7f60c7852000-7f                                                          -2.19
7f60c7854000-7f
7f60c7859000-7f                                                          2.19.so
7f60c7a39000-7f
7f60c7a7a000-7f
7f60c7a7b000-7f                                                          2.19.so
7f60c7a7c000-7f                                                          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

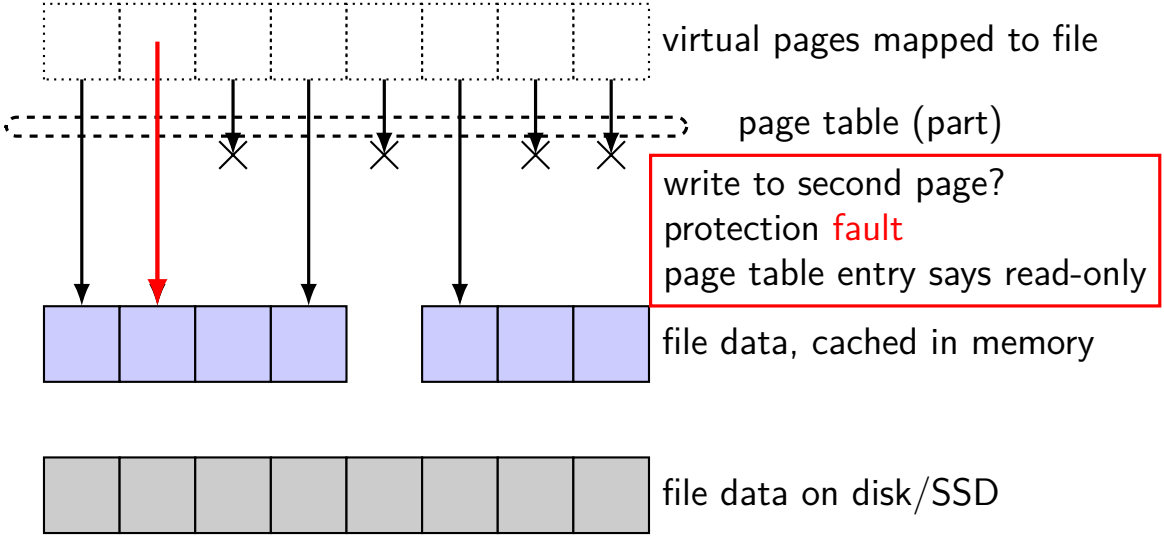
```
int fd = open("/bin/cat", O_RDONLY);
mmap(0x60b000, 0x1000, PROT_READ | PROT_WRITE,
     MAP_PRIVATE, fd, 0xb000);
```

(aside: probably used for global variables)

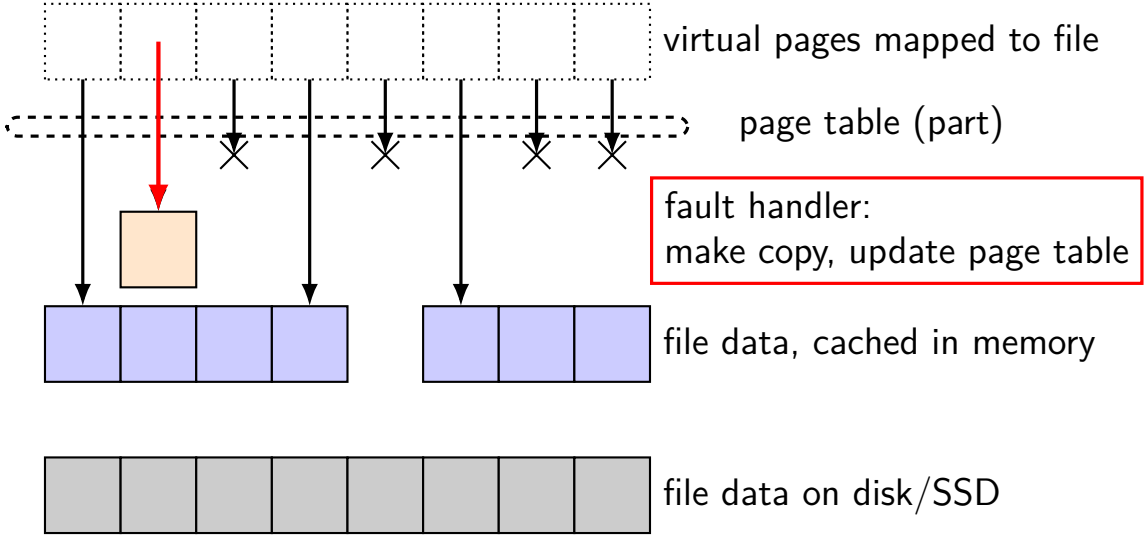
mapped pages (copy-on-write)



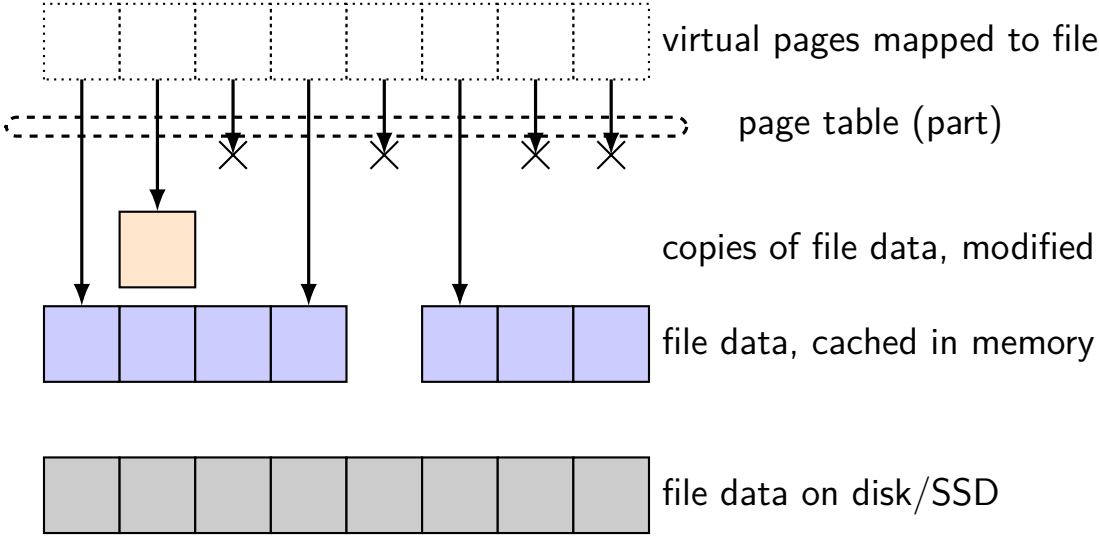
mapped pages (copy-on-write)



mapped pages (copy-on-write)



mapped pages (copy-on-write)



maps counting

4KB (`0x1000` byte) pages

virtual `0x10000–0x1FFFF` (64KB) → "foo.dat" bytes `0–0x0FFFF`

- map setup private (copy-on-write)
- bytes `0–0x3FFF` and `0x5000–0x6FFF` cached in memory

program reads addresses `0x13800–0x15800`

then, program overwrites addresses `0x14800–0x15100`

assume: program page table filled in on demand only

- smarter OS would probably proactively fill in multiple pages

maps counting

4KB (`0x1000` byte) pages

virtual `0x10000`–`0x1FFFF` (64KB) → "foo.dat" bytes `0`–`0x0FFFF`

- map setup private (copy-on-write)
- bytes `0`–`0x3FFF` and `0x5000`–`0x6FFF` cached in memory

program reads addresses `0x13800`–`0x15800`

then, program overwrites addresses `0x14800`–`0x15100`

assume: program page table filled in on demand only

- smarter OS would probably proactively fill in multiple pages

question: how much page/protection faults?

maps counting

4KB (`0x1000` byte) pages

virtual `0x10000`–`0x1FFFF` (64KB) → "foo.dat" bytes `0`–`0x0FFFF`

- map setup private (copy-on-write)
- bytes `0`–`0x3FFF` and `0x5000`–`0x6FFF` cached in memory

program reads addresses `0x13800`–`0x15800`

then, program overwrites addresses `0x14800`–`0x15100`

assume: program page table filled in on demand only

- smarter OS would probably proactively fill in multiple pages

question: how much page/protection faults?

- 1: set PTE for offset 0x3000-0x3FFF (use cached version)
- 2,3: read from disk + set PTE for 0x4000-0x4FFF; set PTE for 0x5000-0x5FFF

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
7f60c7859000-7f60c787c000 r-xp                                               2.19.so
7f60c7a39000-7f60c7a3b000 rw-p
7f60c7a7a000-7f60c7a7b000 rw-p
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
allocated using `sbrk()`
but can get same effect with `mmap()` call

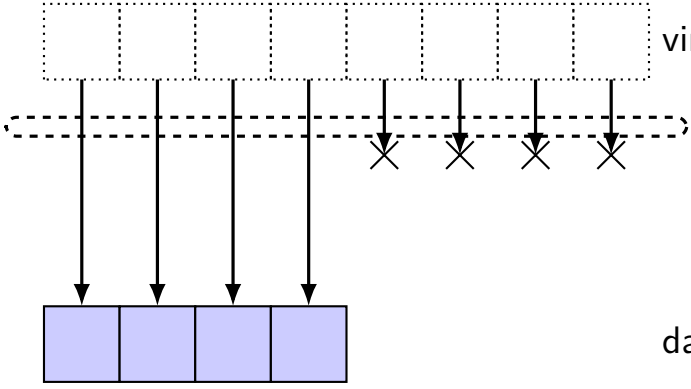
Linux maps

```
$ 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
7f                                                                                                  2.19.so
7f                                                                                                  2.19.so
7f
7f
7f
7ffc5d3b3000-7ffc5d3b5000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

as if:

```
mmap(..., 0x5000, PROT_READ | PROT_WRITE,
     MAP_PRIVATE | MAP_ANONYMOUS /* = no file */, ...);
```

mapped pages (no backing file)



virtual pages w/o backing file

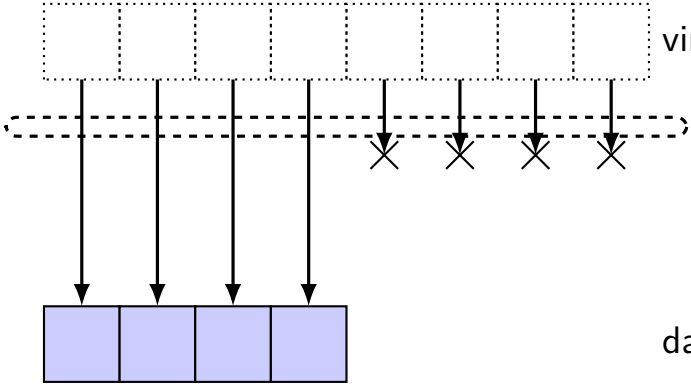
page table (part)

data in memory

data on disk (if any)

"swapped out"

mapped pages (no backing file)



virtual pages w/o backing file

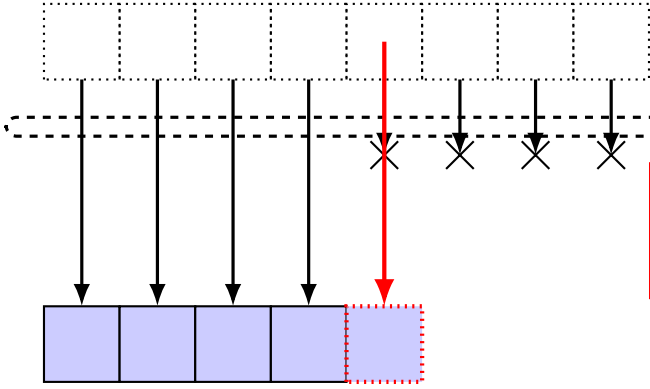
page table (part)

data in memory

data on disk (if any)

"swapped out"

mapped pages (no backing file)



virtual pages w/o backing file

page table (part)

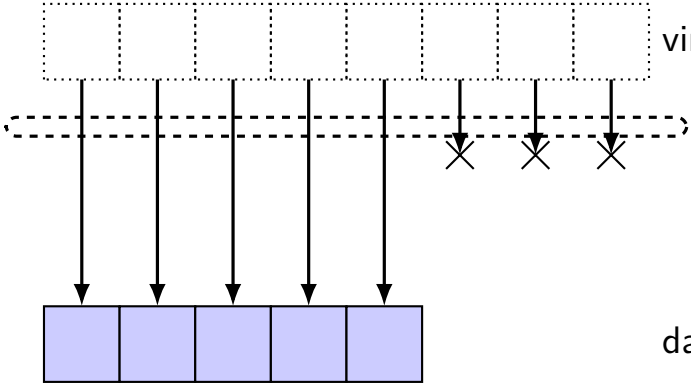
access new page
page fault handler
allocates on demand

data in memory

data on disk (if any)

"swapped out"

mapped pages (no backing file)



virtual pages w/o backing file

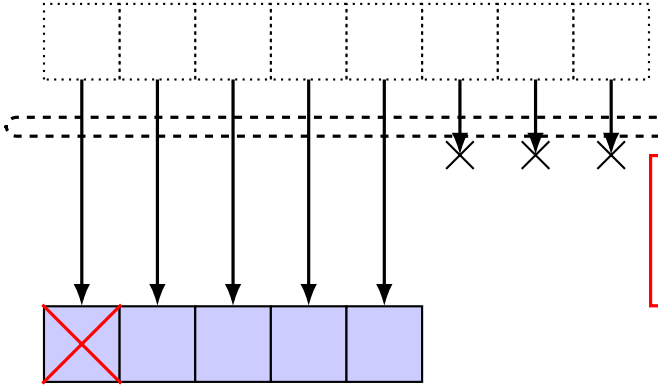
page table (part)

data in memory

data on disk (if any)

"swapped out"

mapped pages (no backing file)



virtual pages w/o backing file

page table (part)

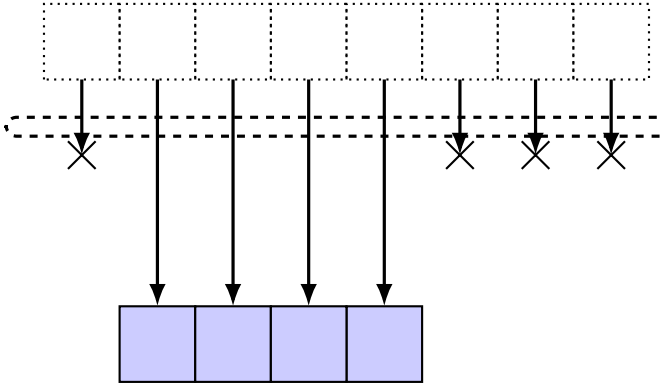
need more memory?
save page to disk
AKA "swap out"

data in memory

data on disk (if any)

"swapped out"

mapped pages (no backing file)



virtual pages w/o backing file

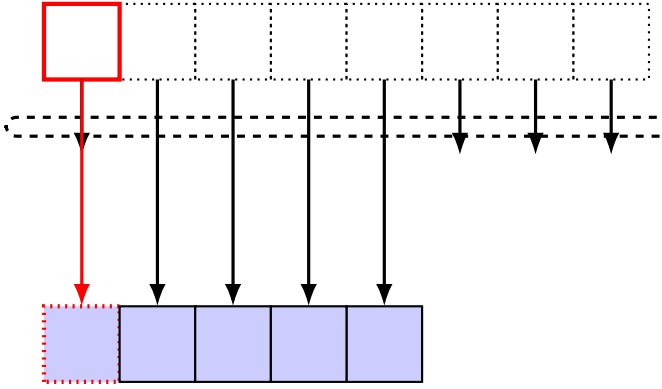
page table (part)

data in memory

data on disk (if any)

"swapped out"

mapped pages (no backing file)



virtual pages w/o backing file

page table (part)

data in memory

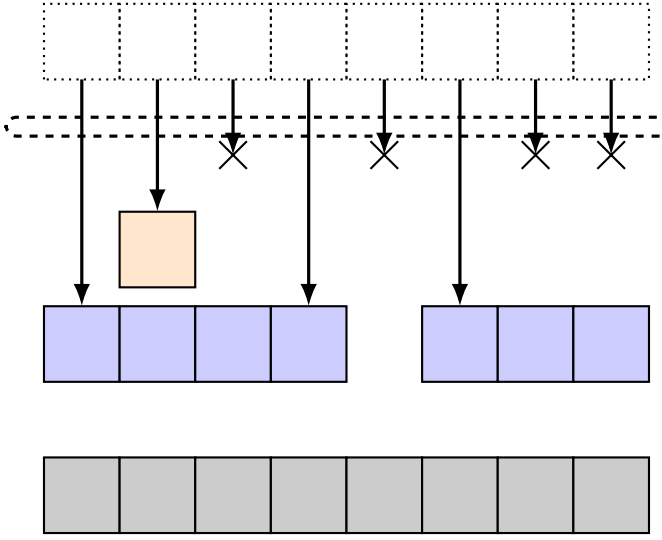
data on disk (if any)

"swapped out"

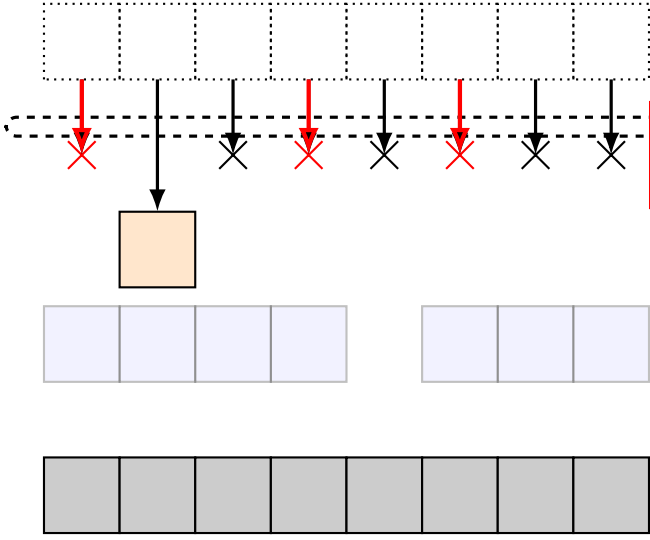
Linux maps

```
$ 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



swapping with copy-on-write



virtual pages mapped to file

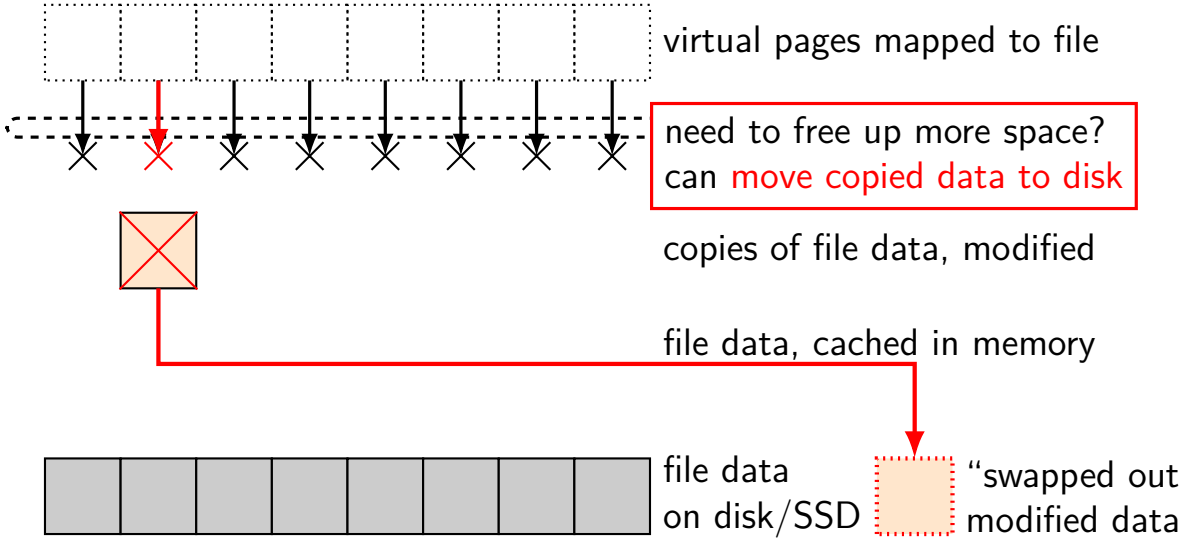
free up space by removing cached copies of file

copies of file data, modified

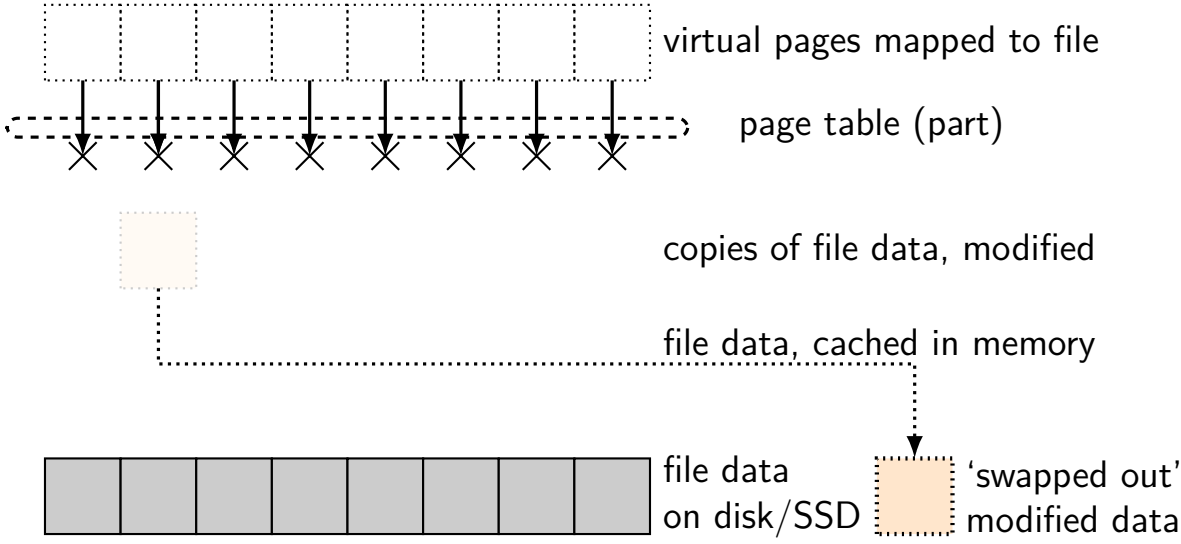
file data, cached in memory

file data on disk/SSD

swapping with copy-on-write



swapping with copy-on-write



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 and 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 being read/written?
- possibly both

goal: manage this cache intelligently

the page cache

memory is a cache for disk

files and 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 being read/written?
- possibly both

goal: manage this cache intelligently

the page cache

memory is a cache for disk

files and 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 being read/written?
- possibly both

goal: manage this cache intelligently

the page cache

memory is a cache for disk

files and 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 being read/written?
- possibly both

goal: manage this cache intelligently

page cache components [text]

mapping: virtual address or file+offset → physical page

- handle cache hits

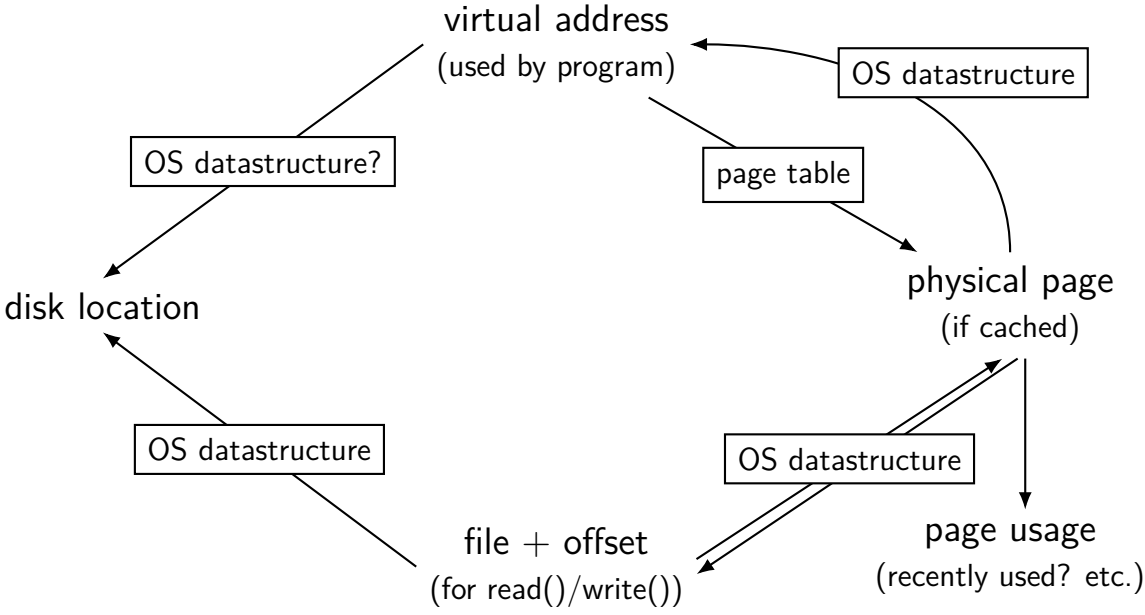
find backing location based on virtual address/file+offset

- handle cache misses

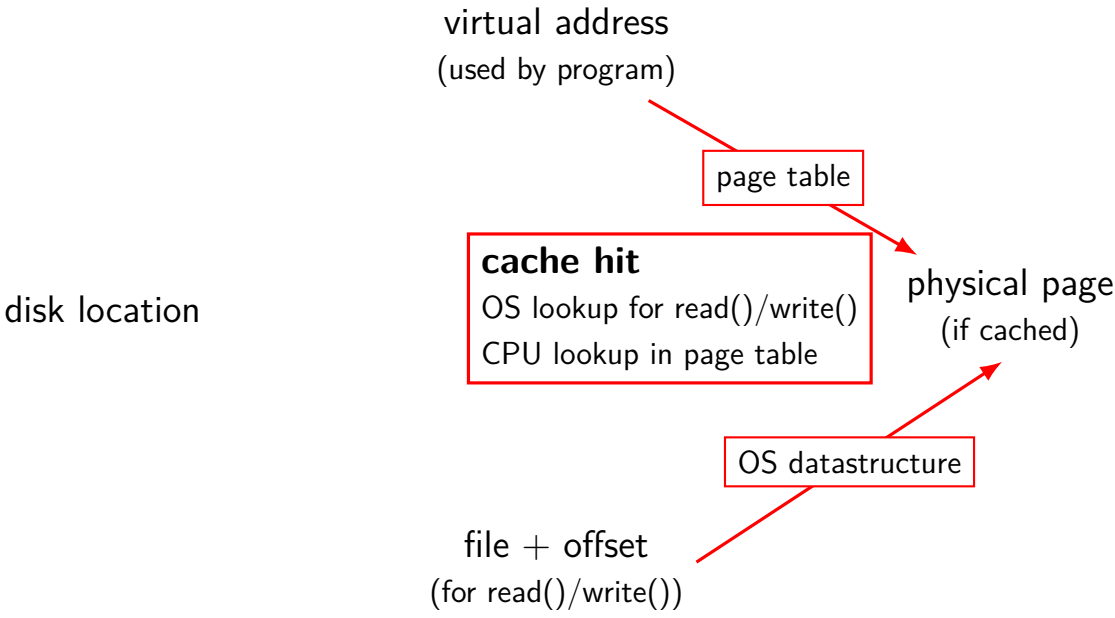
track information about each physical page

- handle page allocation
- handle cache eviction

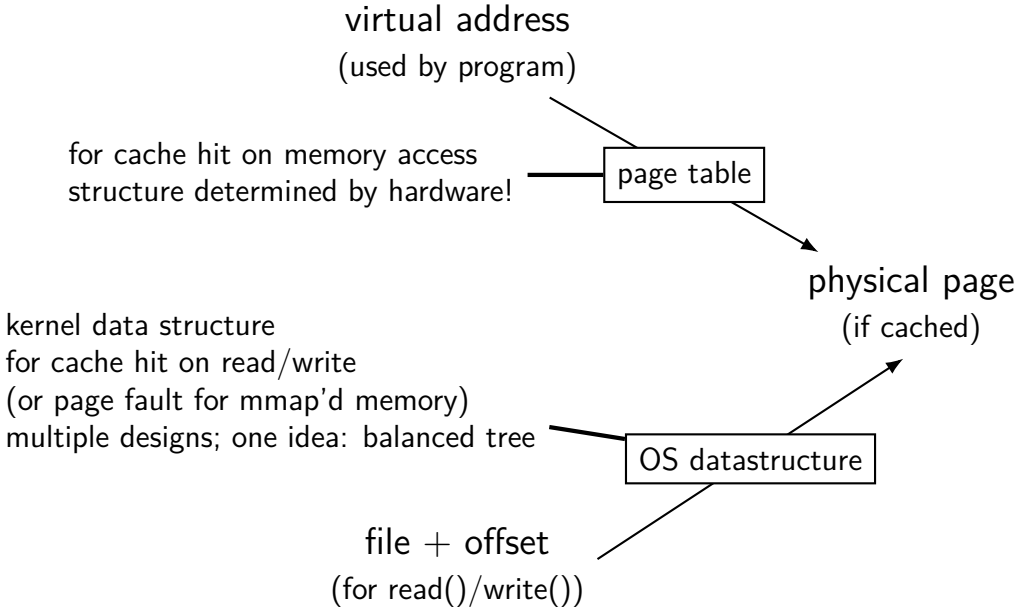
page cache components



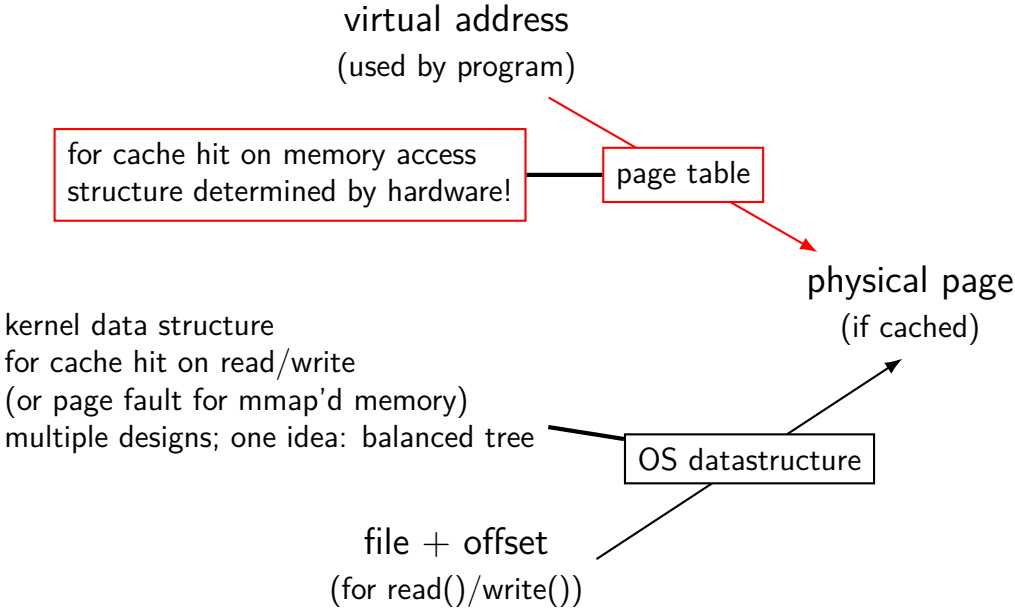
page cache components



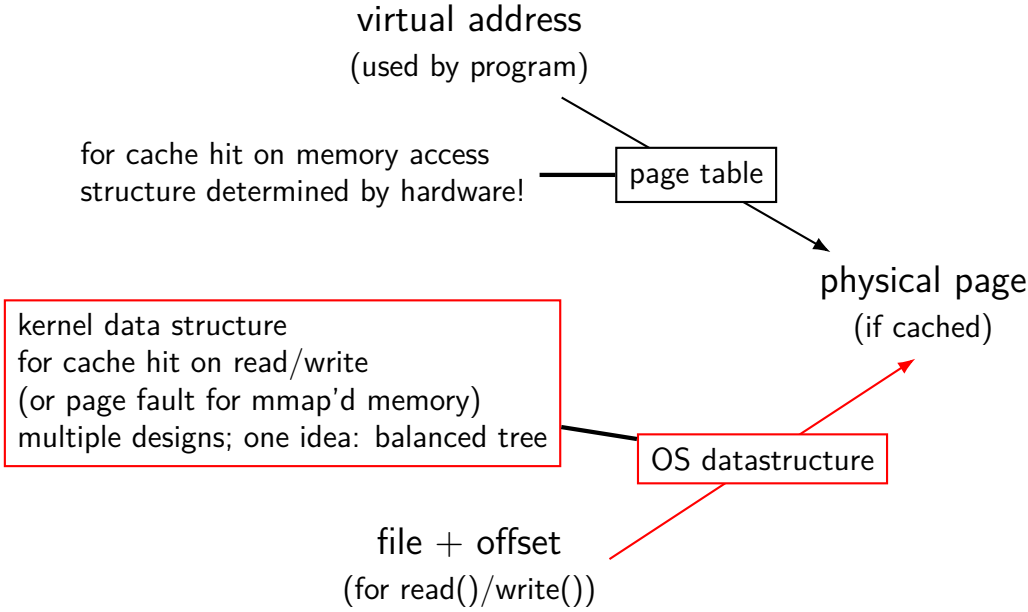
virtual addr/file offset to physical page



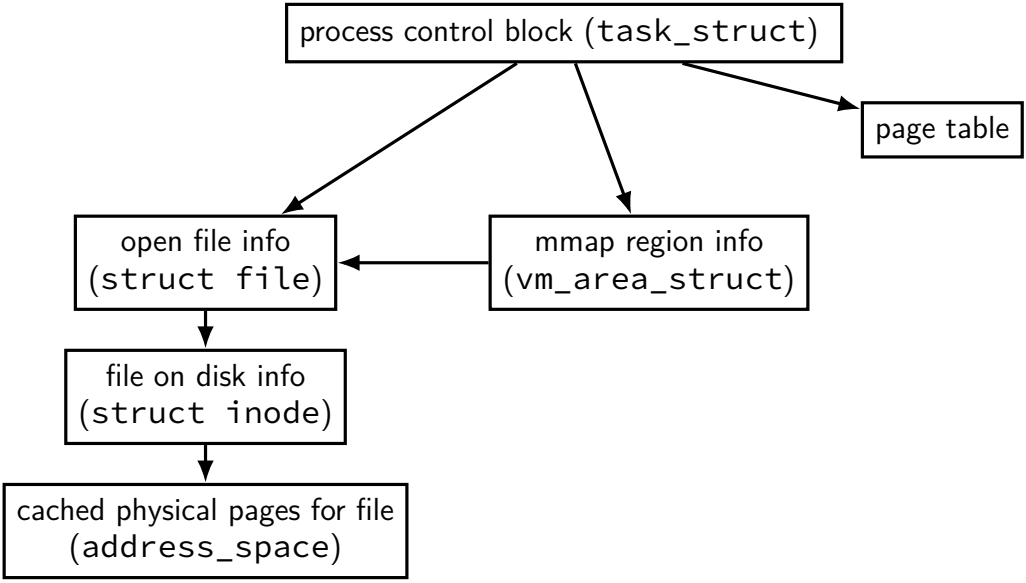
virtual addr/file offset to physical page



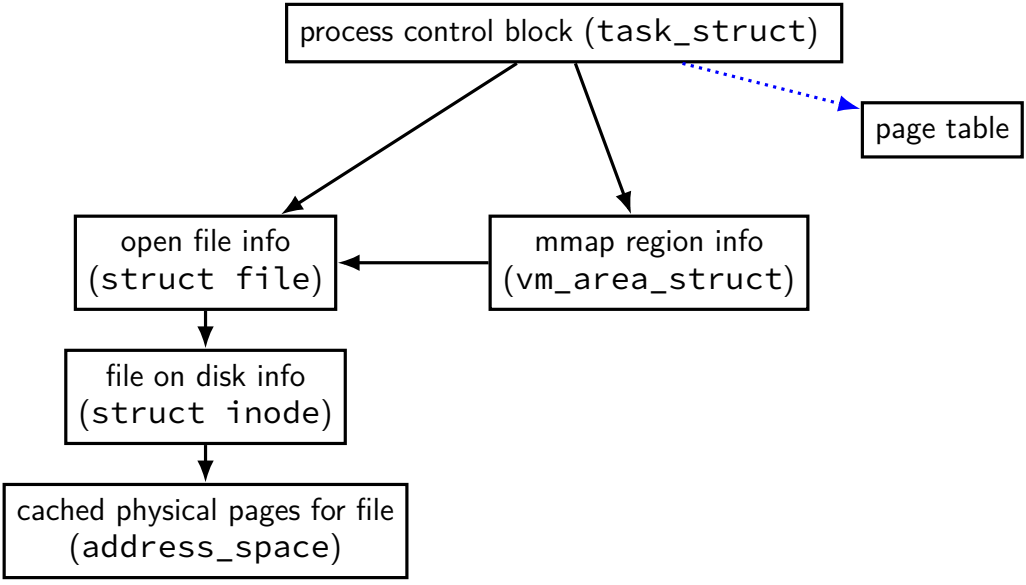
virtual addr/file offset to physical page



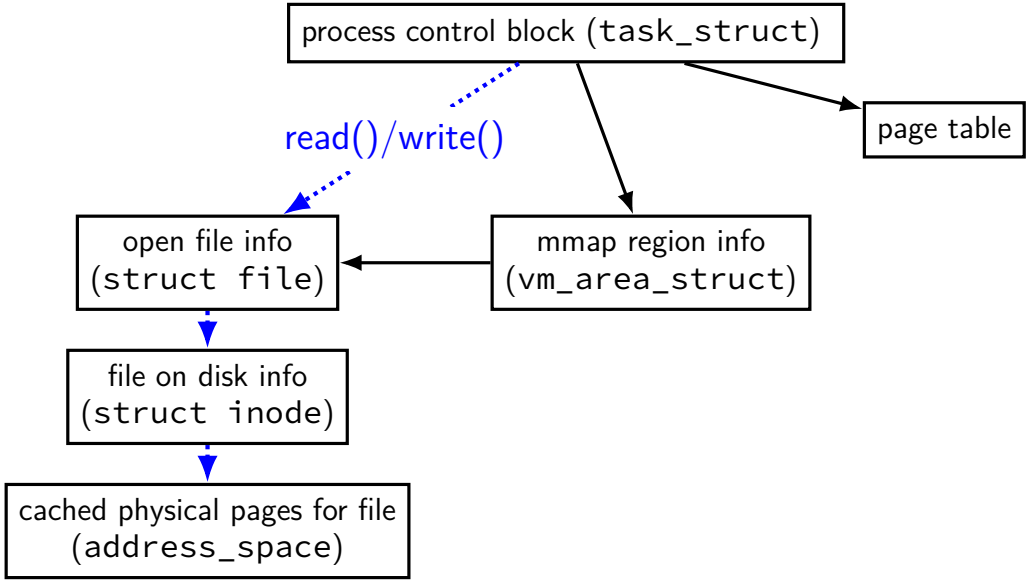
Linux: forward mapping



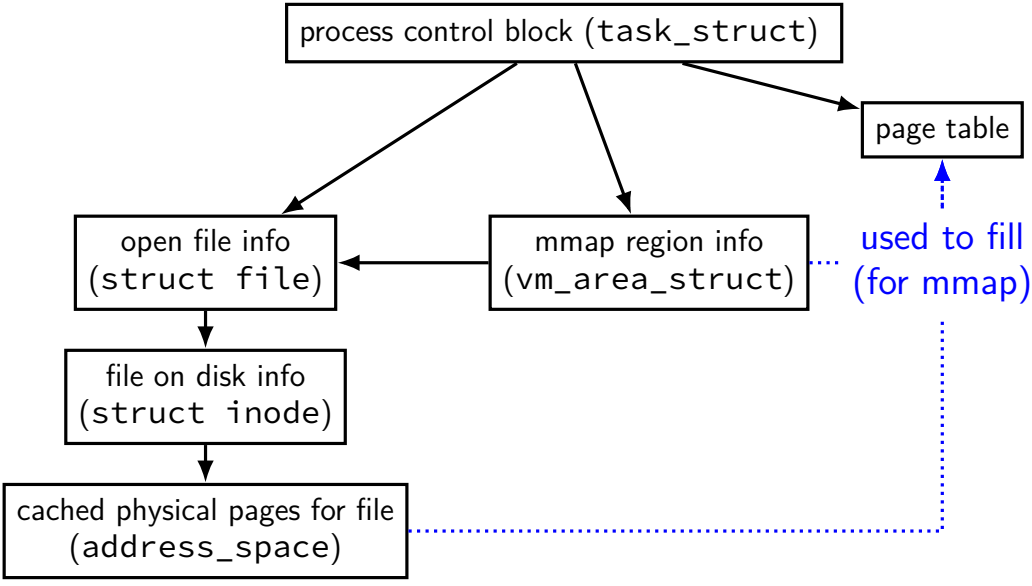
Linux: forward mapping



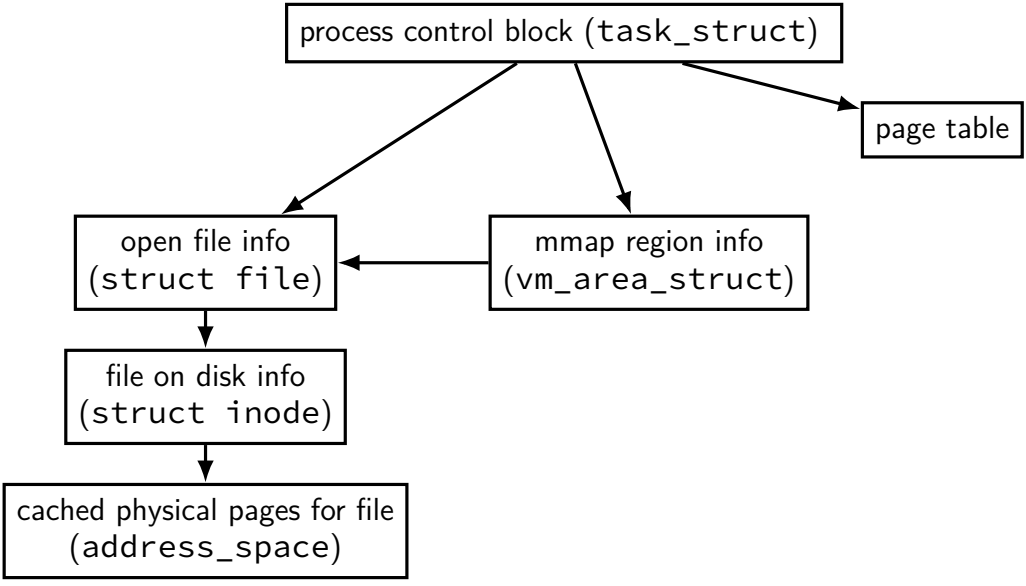
Linux: forward mapping



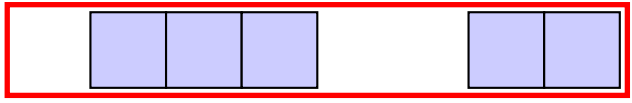
Linux: forward mapping



Linux: forward mapping



mapped pages (read/write, shared)



file data, cached in memory

file data on disk/SSD

page replacement

step 1: *evict* a page to free a physical page

case 1: there's an unused page, just use that (easy)

case 2: need to remove whatever what's in that page (more work)

step 2: load new, more important in its place

needs some way of knowing location of data

page replacement

step 1: *evict* a page to free a physical page

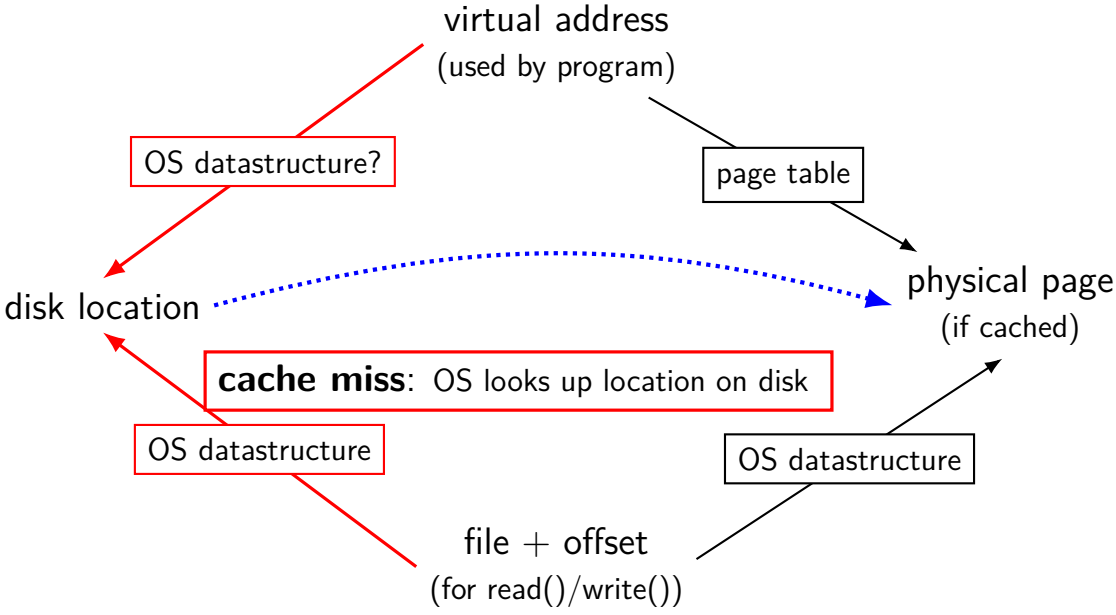
case 1: there's an unused page, just use that (easy)

case 2: need to remove whatever what's in that page (more work)

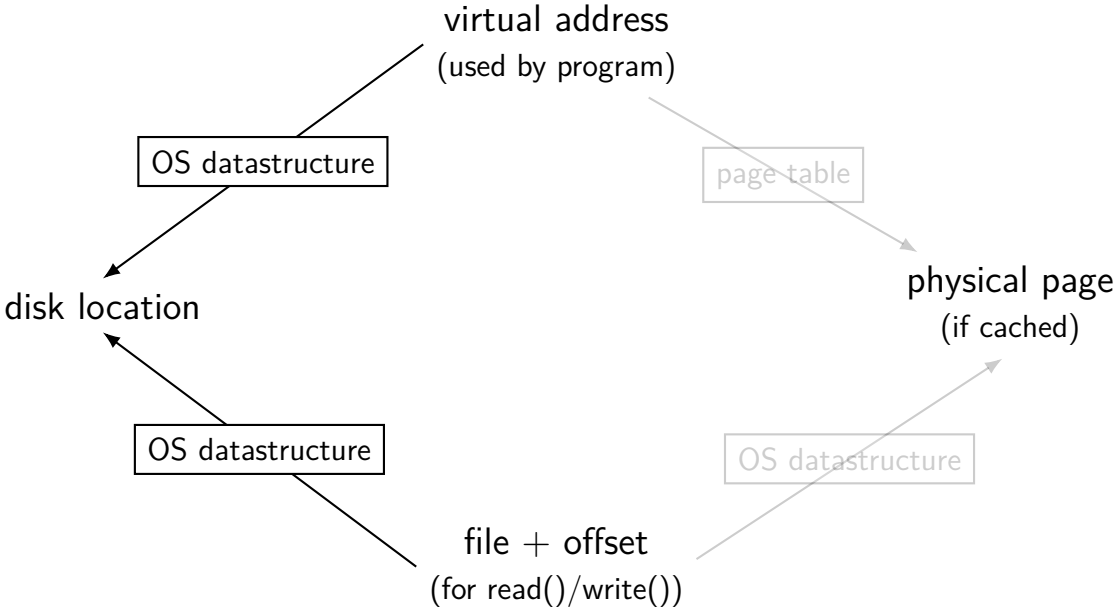
step 2: load new, more important in its place

needs some way of knowing location of data

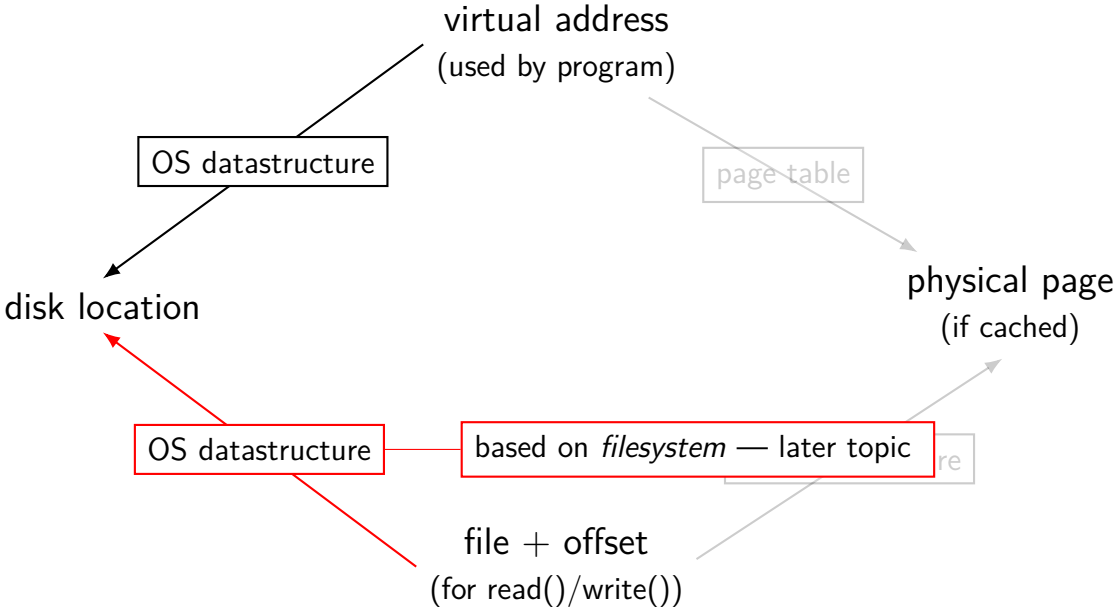
page cache components



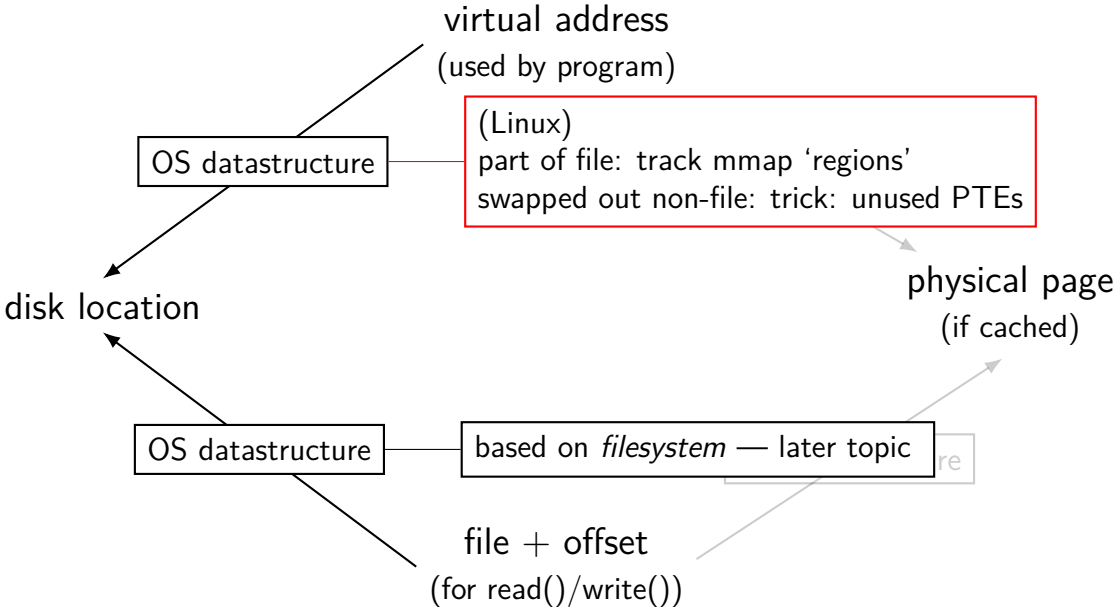
virtual address/file offset $\rightarrow$ location on disk



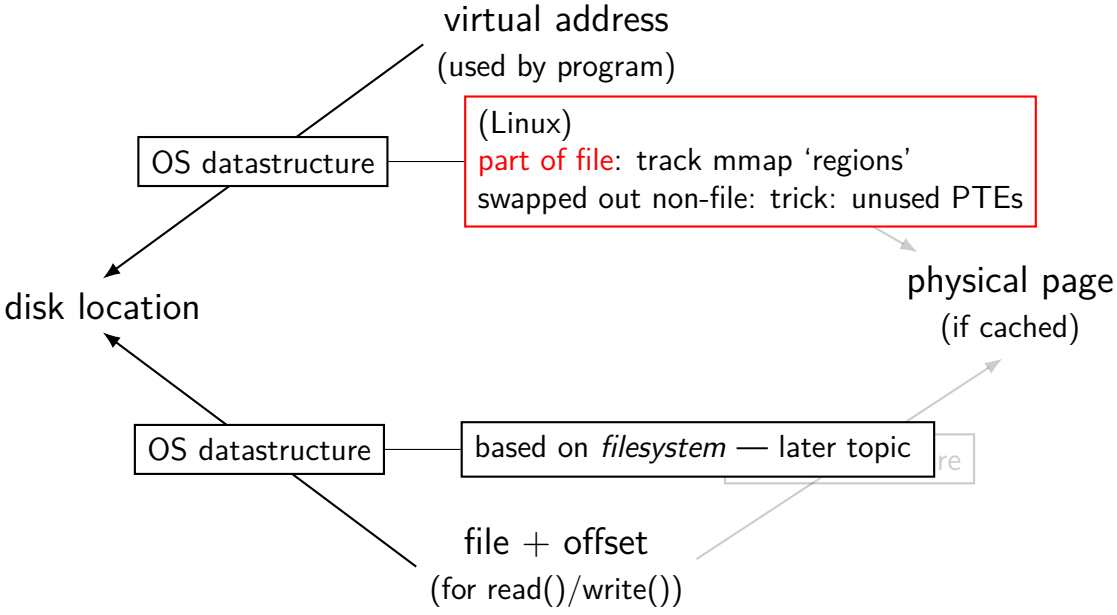
virtual address/file offset → location on disk



virtual address/file offset → location on disk



virtual address/file offset → location on disk



Linux maps: list of 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
7f60c
7f60c
7f60c
7f60c
7f60c
7f60c
7f60c
7f60c
7f60c
7f60c
7ffc5
7ffc5
7ffc5
fffff
```

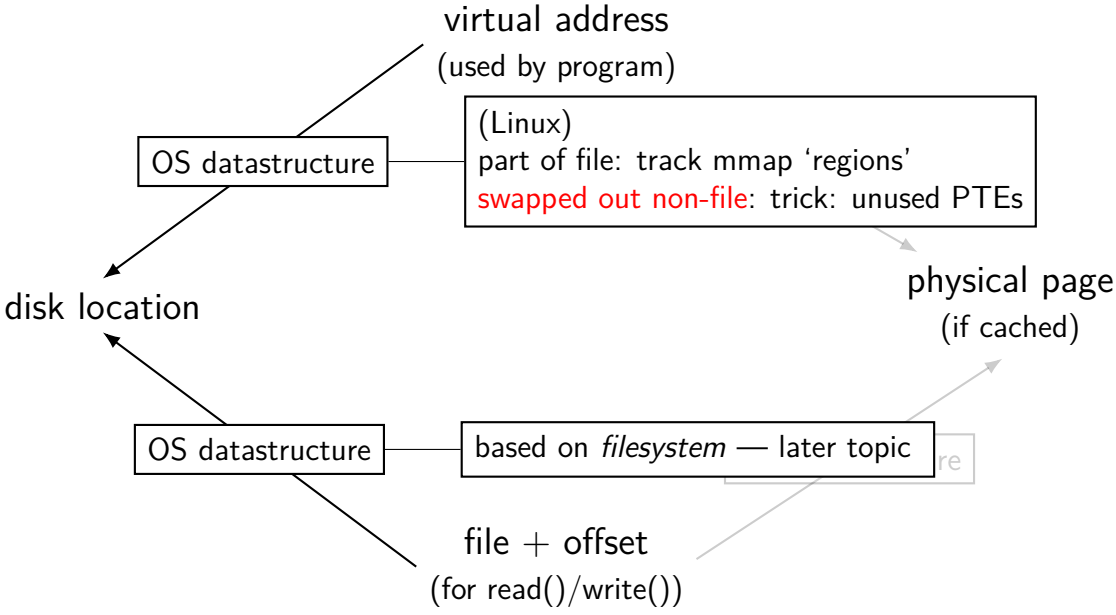
PCB contains list of `struct vm_area_struct` with:
(shown in this output):

- virtual address start, end
- permissions
- offset in backing file (if any)
- pointer to backing file (if any)

(not shown):

- info about sharing of non-file data (e.g. heap after fork) ...

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

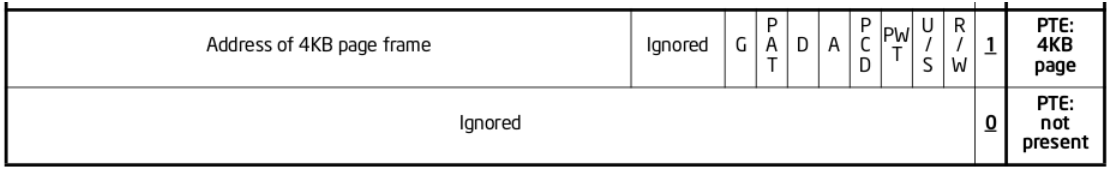


Figure 4-4. Formats of CR3 and Paging-Structure Entries with 32-Bit Paging

page replacement

step 1: *evict* a page to free a physical page

case 1: there's an unused page, just use that (easy)

case 2: need to remove whatever what's in that page (more work)

step 2: load new, more important in its place

needs some way of knowing location of data

evicting a page

remove victim page from page table, etc.

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

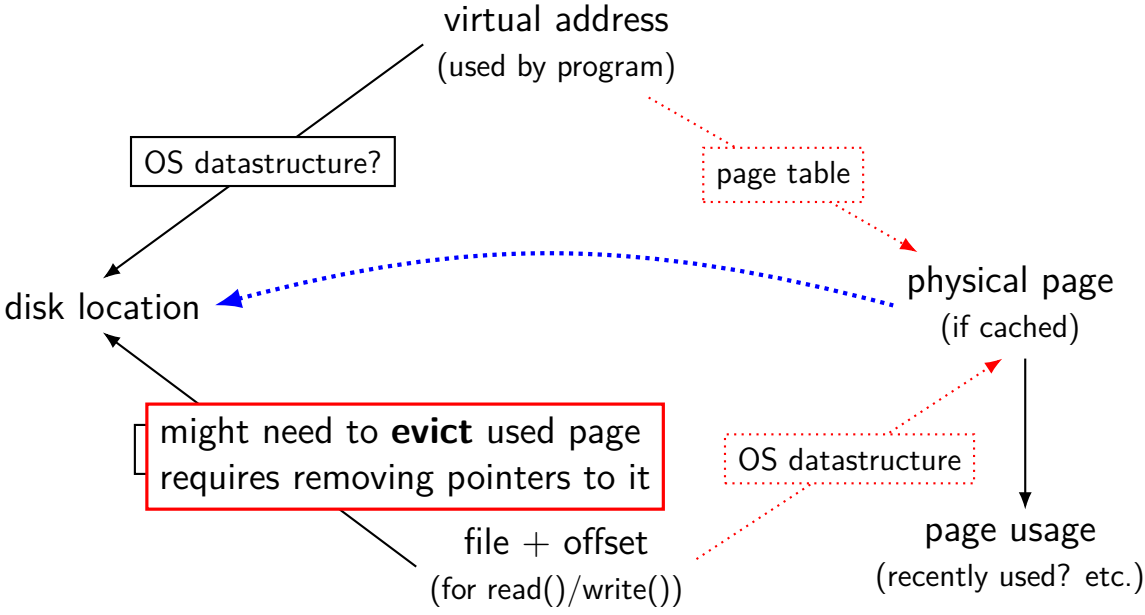
if needed, save victim page to disk

going to require:

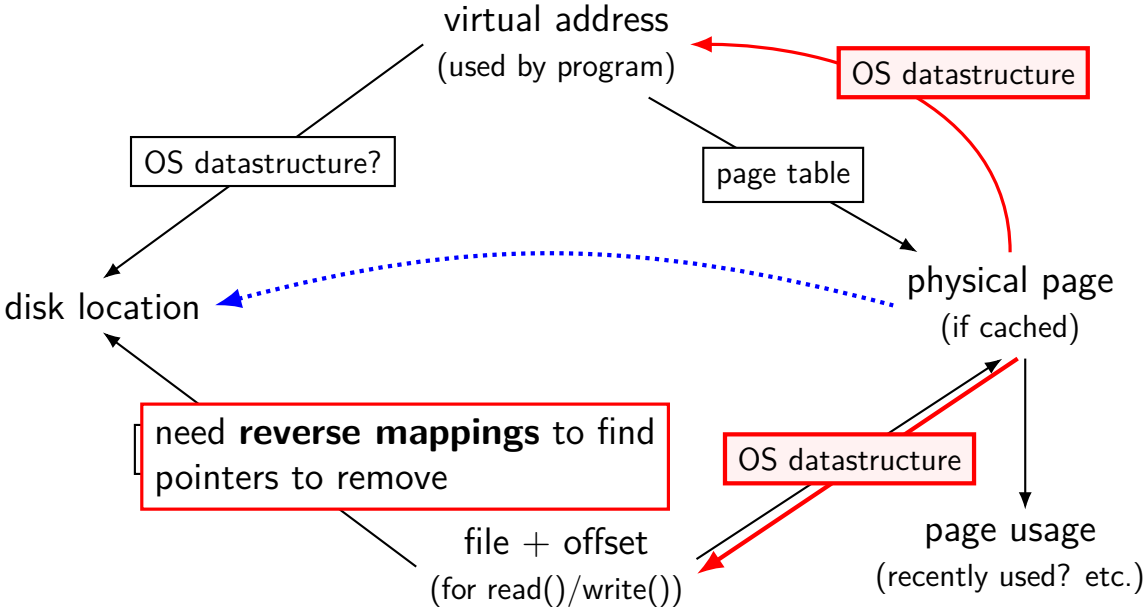
way to find page tables, etc. using page

way to detect whether it needs to be saved to disk

page cache components



page cache components



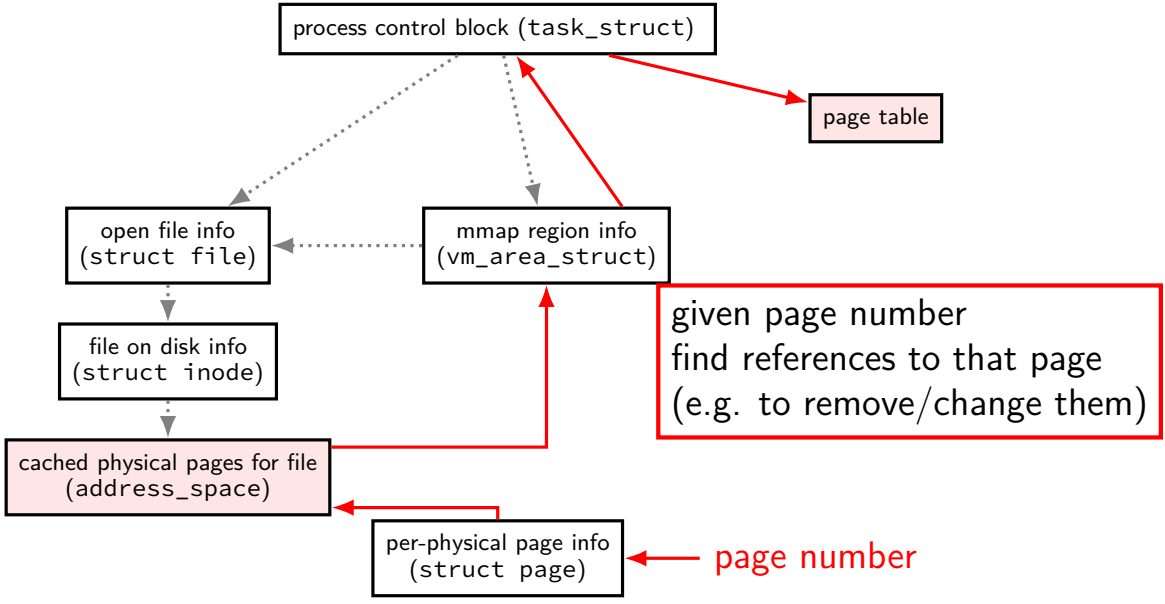
tracking physical pages: finding mappings

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

need to track where every page is used!

common solution: structure for every physical page with info about every cached file/page table using page

Linux: reverse mapping (file pages)



backup slides

fast copies

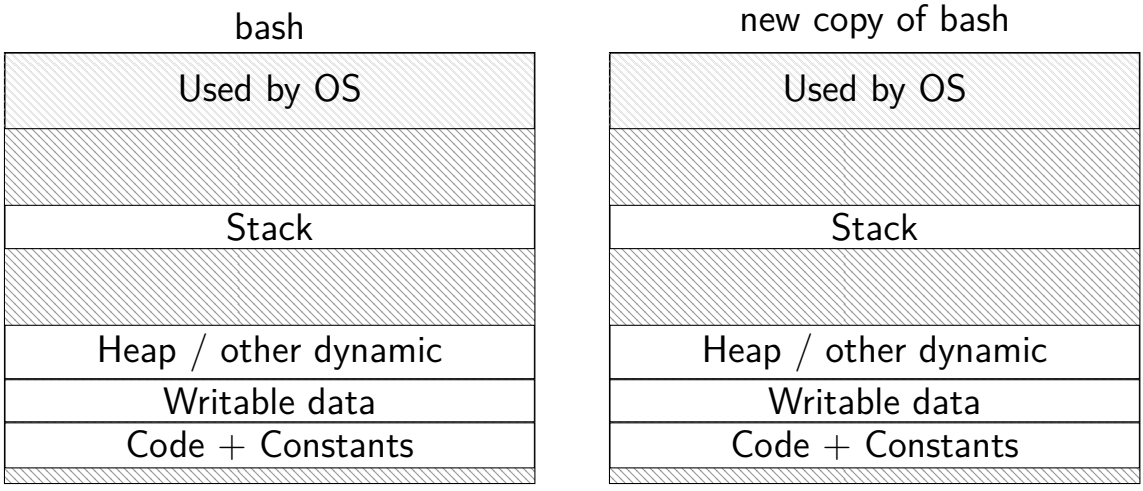
recall : `fork()`

creates a copy of an entire program!

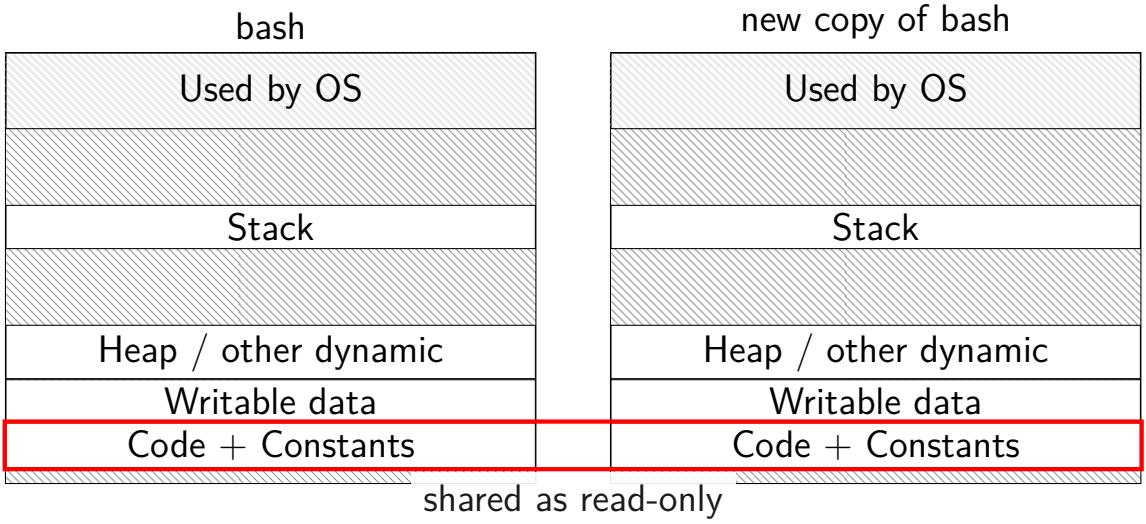
(usually, the copy then calls `execve` — replaces itself with another program)

how isn't this really slow?

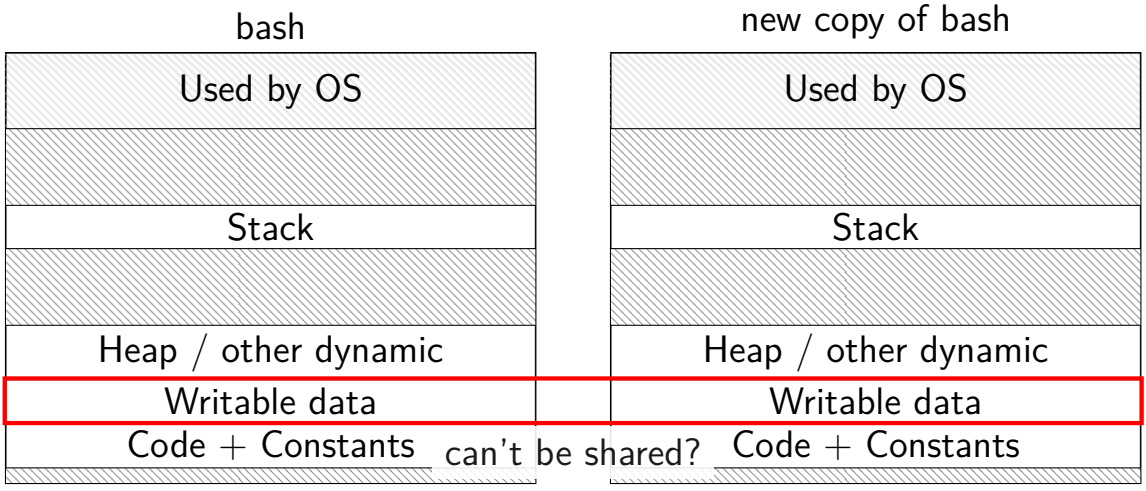
do we really need a complete copy?



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

sketch: implementing mmap

access mapped file for first time, read from disk

(like swapping when memory was swapped out)

write "mapped" memory, write to disk eventually

need to detect whether writes happened

usually hardware support: dirty bit

extra detail: other processes should see changes

all accesses to file use same physical memory

how? OS tracks copies of files in memory

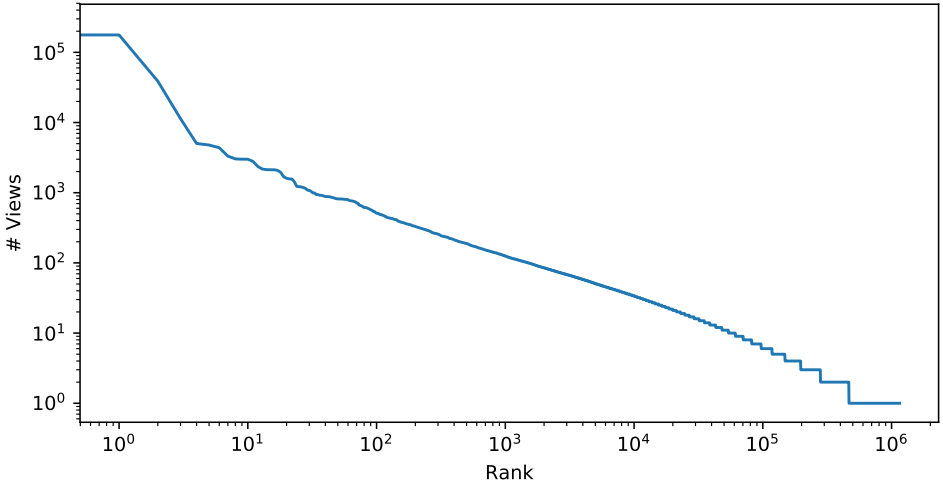
aside: Zipf model

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

page cache versus processor cache

unlike processor cache, page cache...

stores multi-kilobyte blocks

- add/remove whole 4+KB pages versus 64-128B blocks
- smaller page tables; better for hard drives/SSDs

handles misses (get value if not cached) in software

- OS data structures tack data on disk/SSDs
- hardware doesn't know/care about them
- hardware only knows how to invoke page fault handler

has no restrictions on where values are stored in cache

- any physical page can be used for any virtual page
- (processor caches have limited *associativity*)

page cache versus processor cache

unlike processor cache, page cache...

stores multi-kilobyte blocks

- add/remove whole 4+KB pages versus 64-128B blocks
- smaller page tables; better for hard drives/SSDs

handles misses (get value if not cached) in software

- OS data structures tack data on disk/SSDs
- hardware doesn't know/care about them
- hardware only knows how to invoke page fault handler

has no restrictions on where values are stored in cache

- any physical page can be used for any virtual page
- (processor caches have limited *associativity*)

page cache versus processor cache

unlike processor cache, page cache...

stores multi-kilobyte blocks

- add/remove whole 4+KB pages versus 64-128B blocks
- smaller page tables; better for hard drives/SSDs

handles misses (get value if not cached) in software

- OS data structures tack data on disk/SSDs
- hardware doesn't know/care about them
- hardware only knows how to invoke page fault handler

has no restrictions on where values are stored in cache

- any physical page can be used for any virtual page
- (processor caches have limited *associativity*)

page cache versus processor cache

unlike processor cache, page cache...

stores multi-kilobyte blocks

- add/remove whole 4+KB pages versus 64-128B blocks
- smaller page tables; better for hard drives/SSDs

handles misses (get value if not cached) in software

- OS data structures tack data on disk/SSDs
- hardware doesn't know/care about them
- hardware only knows how to invoke page fault handler

has no restrictions on where values are stored in cache

- any physical page can be used for any virtual page
- (processor caches have limited *associativity*)

page cache versus processor cache

unlike processor cache, page cache...

stores multi-kilobyte blocks

- add/remove whole 4+KB pages versus 64-128B blocks
- smaller page tables; better for hard drives/SSDs

handles misses (get value if not cached) in software

- OS data structures tack data on disk/SSDs
- hardware doesn't know/care about them
- hardware only knows how to invoke page fault handler

has no restrictions on where values are stored in cache

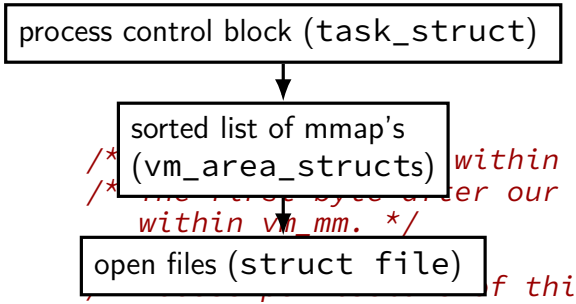
- any physical page can be used for any virtual page
- (processor caches have limited *associativity*)

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;          /*  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;
```

Linux: tracking memory regions

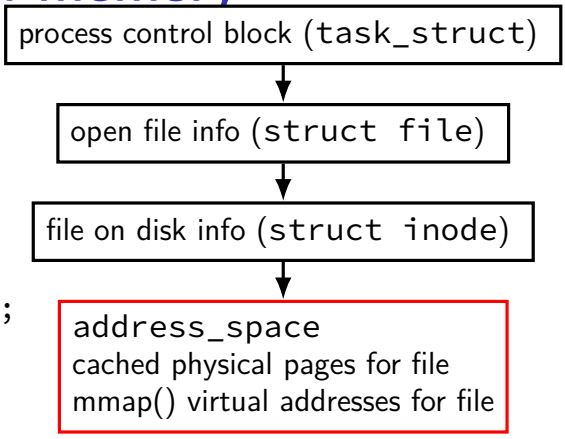
for finding other
uses of non-file pages
e.g. two copies after fork

```
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;
```

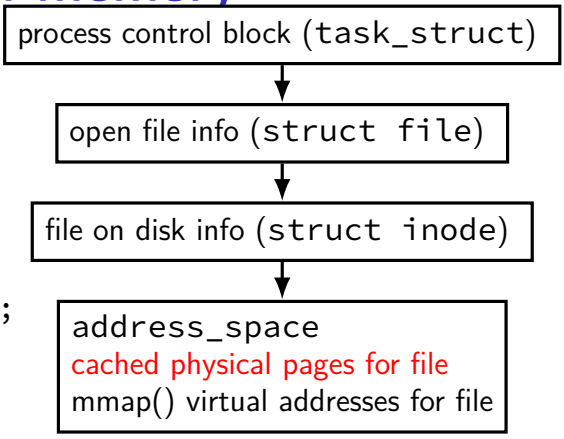
Linux: tracking files in memory

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

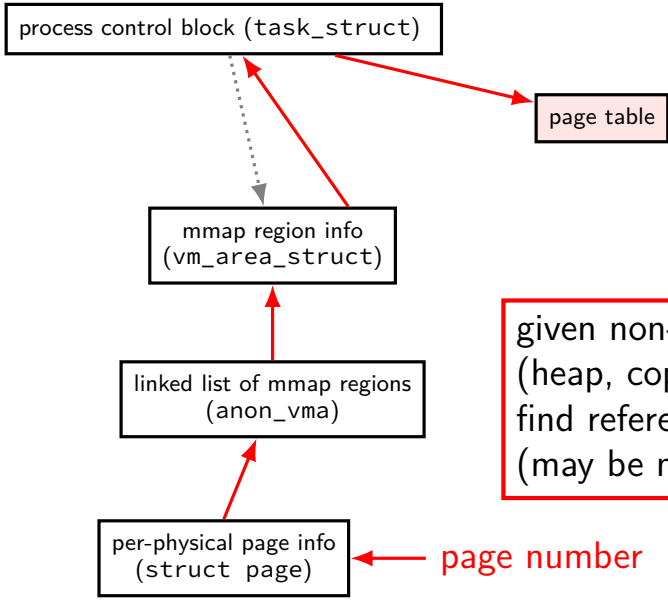


Linux: tracking files in memory

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



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: 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)