mmap / page cache

last time (1)

kalloc/kfree: linked list of available pages

allocuvm: creating memory for program

handling page faults without crashing
- don't setup whole page table in advance
- on page fault: check if OS told program memory was okay
- if so, update page table for that memory

allocate-on-demand
- record somewhere what memory should be allocated
- only actually allocate it when the program tries to access it

last time (2)

copy-on-write

- on fork: don't copy pages; make them read-only instead
- record somewhere what memory should be read-only
- when process tries to access read-only page that "should be" writeable, make a copy

mmap: making files appear as pages

- Linux: treats proecss memory as list of mapped files regions
  - special case: region can be mapped to 'anonymous' file
- MAP_SHARED: modifications to memory modify file!
- MAP_PRIVATE: modifications to memory make private copy

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                                                      -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.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 001be                                              -2.19
7f60c7852000-7f60c7854000 rw-p 001c2                                              -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                                                 -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                                                              c-2.19
7f60c7852000-7                                                              c-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

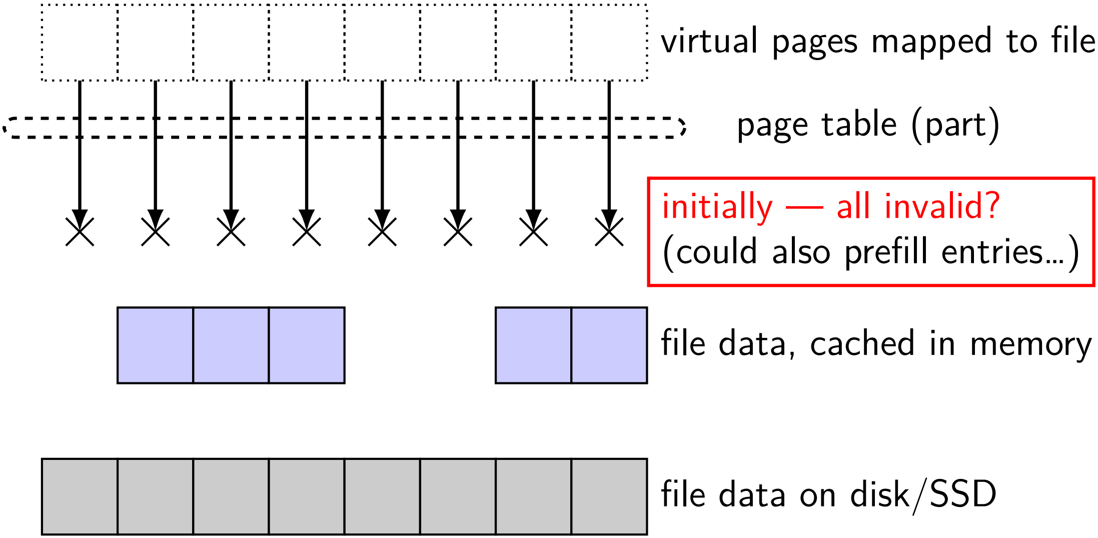
Linux maps

```
$ 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
7f60c7854000-7f60
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]
```

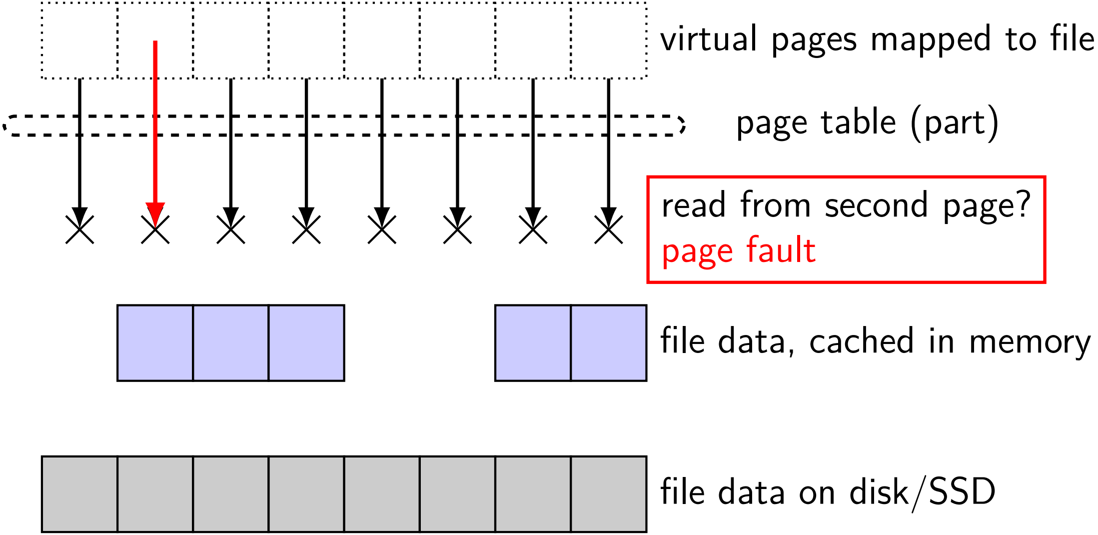
as if:

```c
int fd = open("/bin/cat", O_RDONLY);
mmap(0x400000, 0xb000, PROT_READ | PROT_EXEC,
     MAP_PRIVATE, fd, 0x0);
```

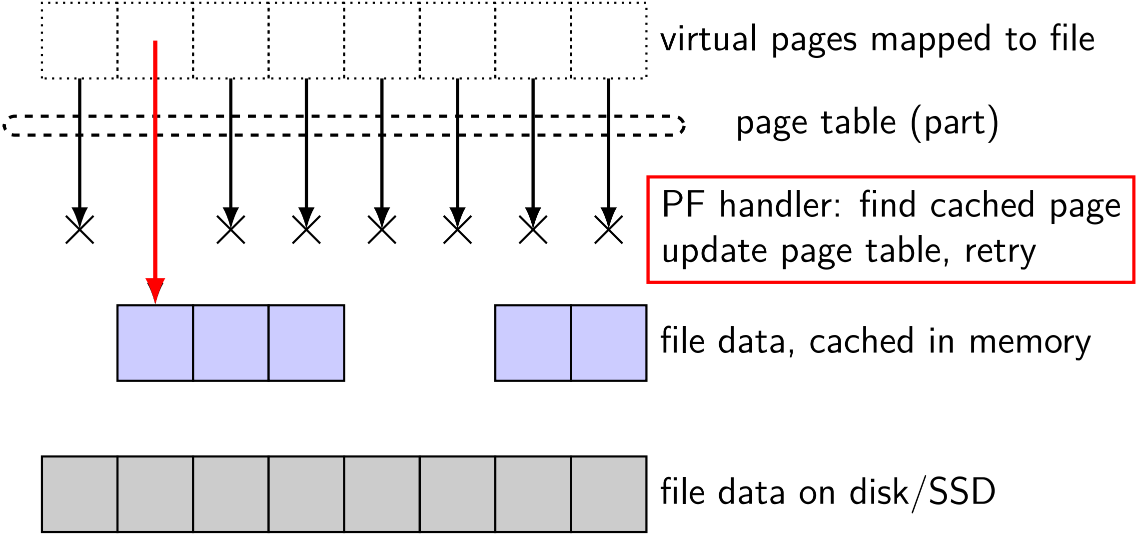
mapped pages (read-only)



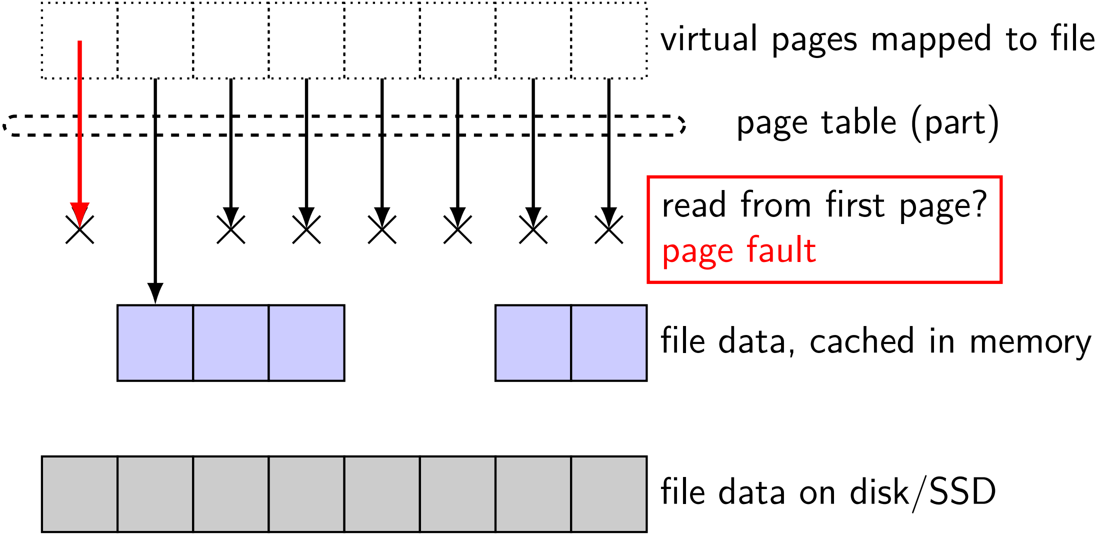
mapped pages (read-only)



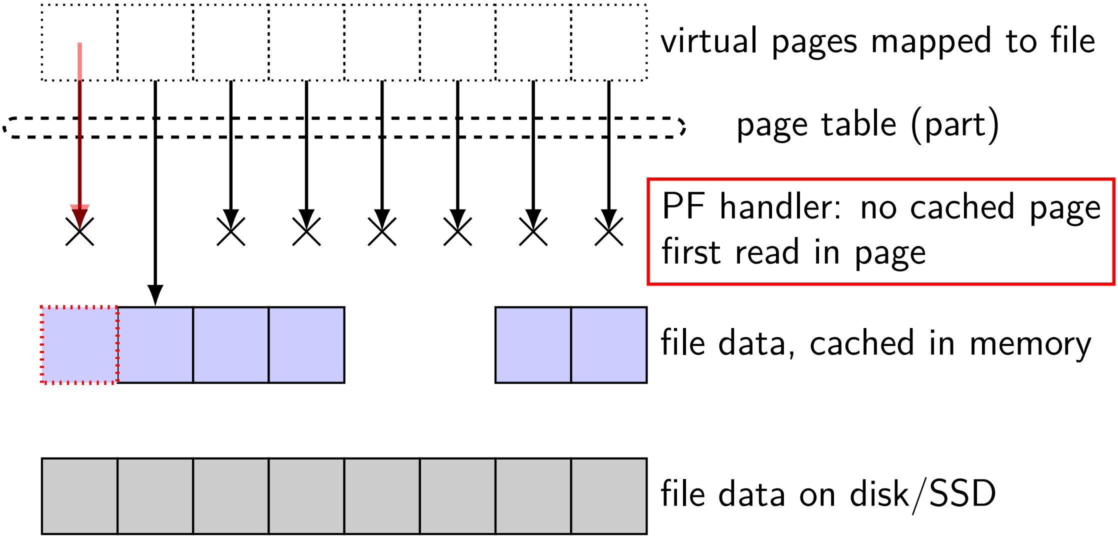
mapped pages (read-only)



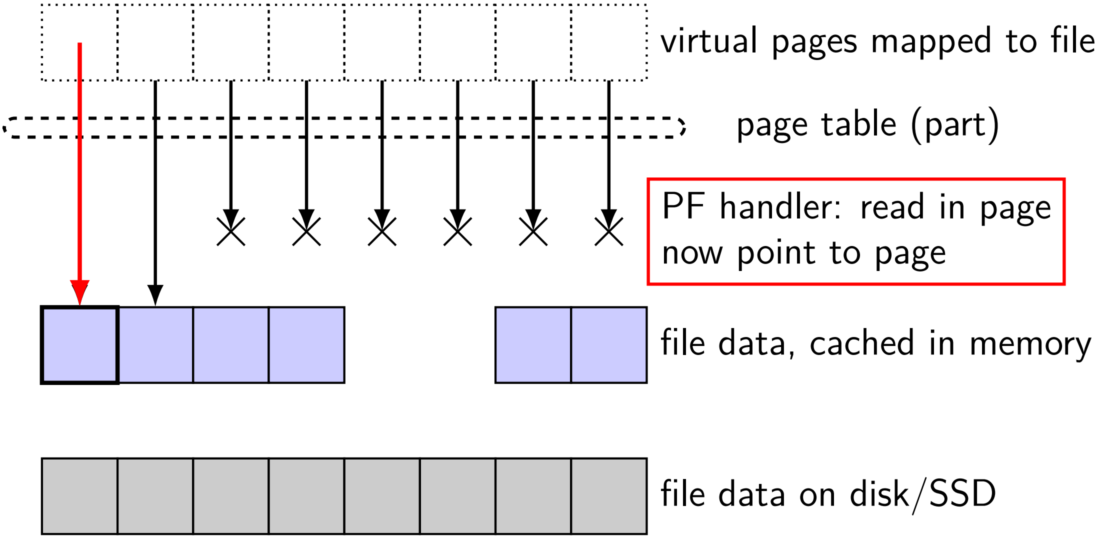
mapped pages (read-only)



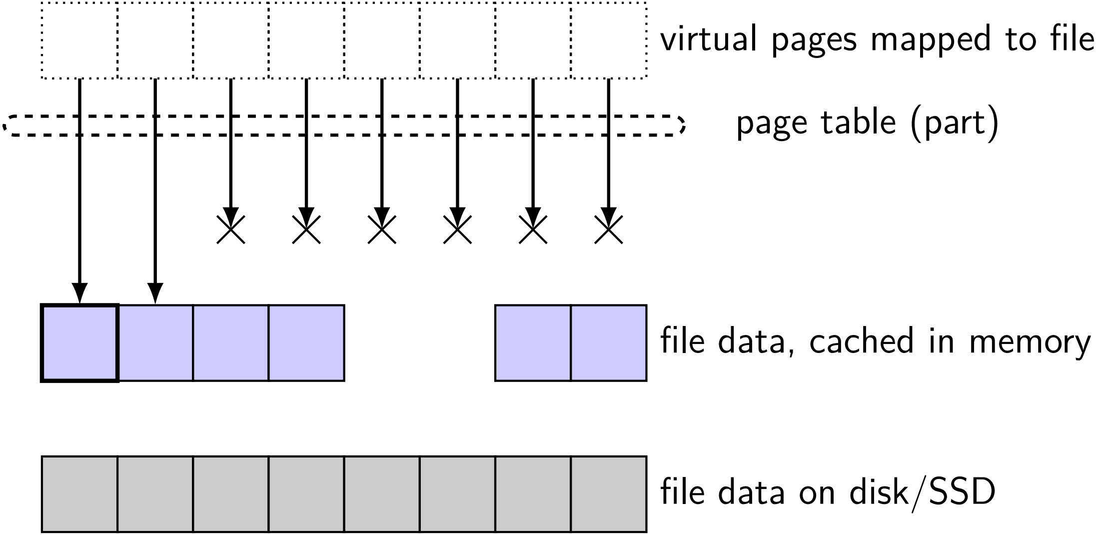
mapped pages (read-only)



mapped pages (read-only)



mapped pages (read-only)



shared mmap

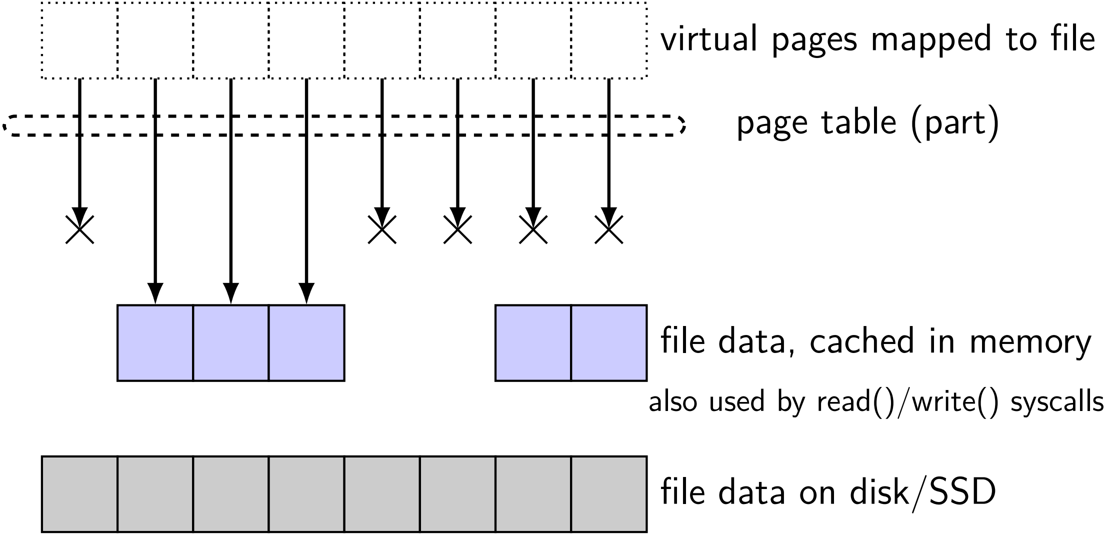
```
int fd  = open("/tmp/somefile.dat", O_RDWR);
mmap(0, 64 * 1024, PROT_READ | PROT_WRITE,
     MAP_SHARED, fd, 0);
```

---

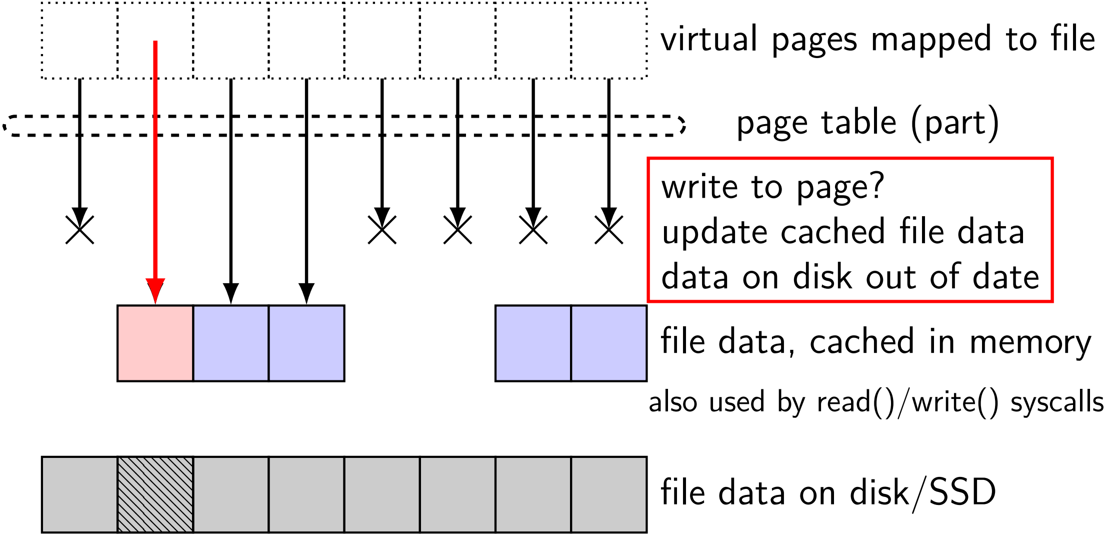
from /proc/PID/maps for this program:

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

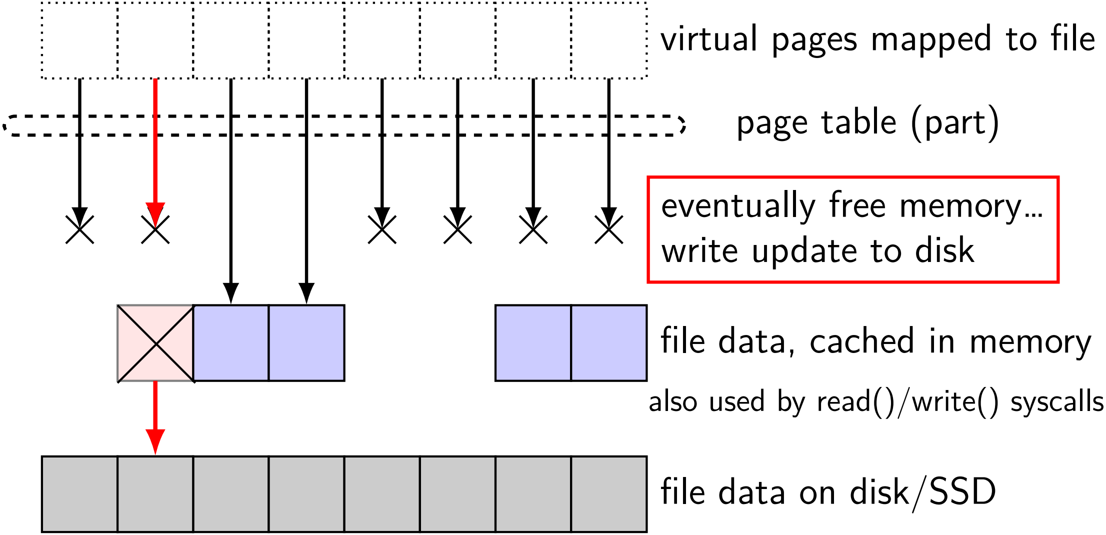
mapped pages (read/write, shared)



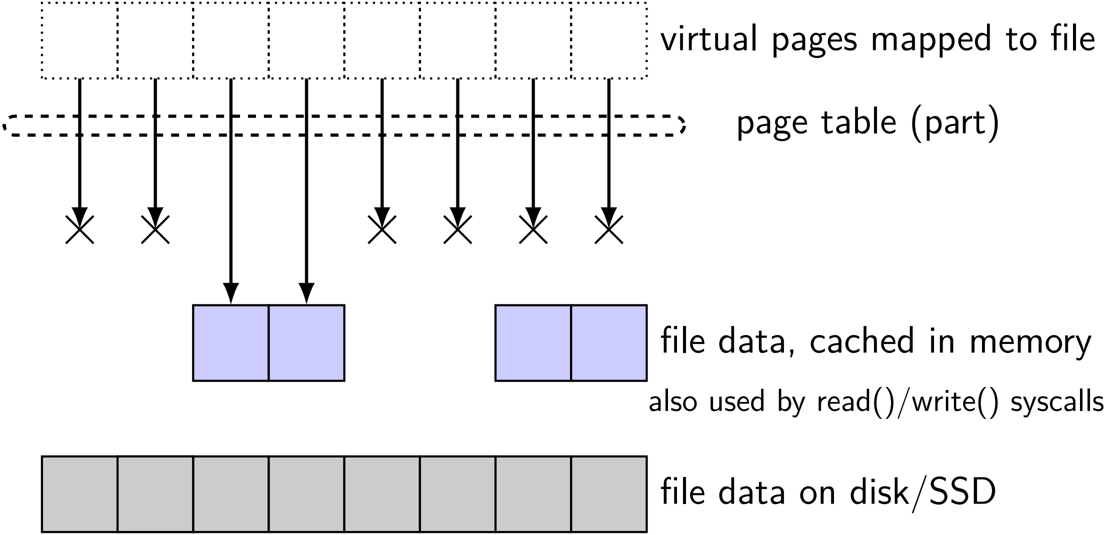
mapped pages (read/write, shared)



mapped pages (read/write, shared)



mapped pages (read/write, shared)



Linux maps

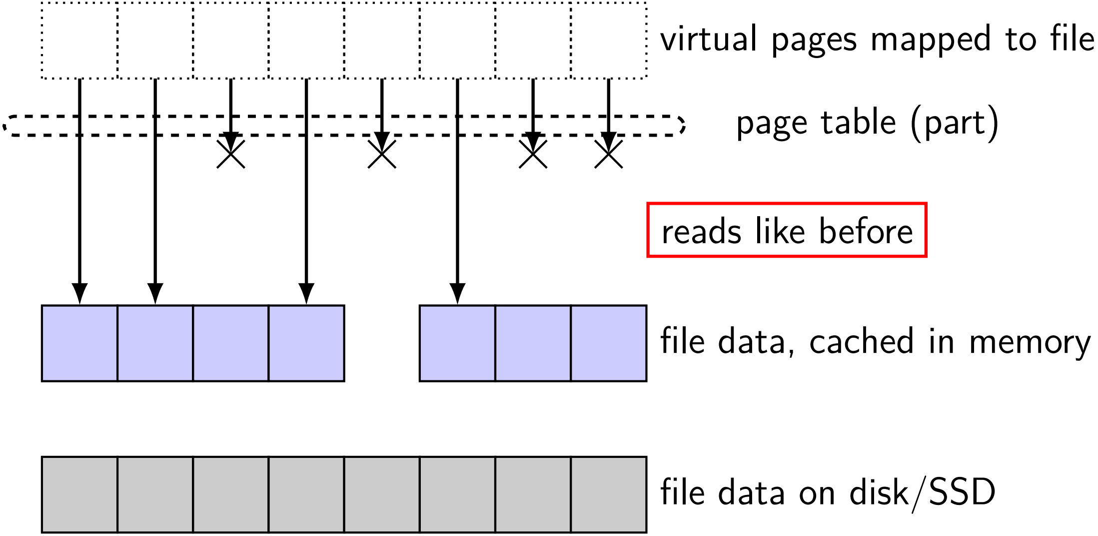
```
$ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 08:01 48328831          /bin/cat
0060a000-0060b000 r--p 0000a000 08:01 48328831          /bin/cat
0060b000-0060c000 rw-p 0000b000 08:01 48328831          /bin/cat
01974000-01995000 rw-p 00000000 00:00 0                 [heap]
7f60c718b000-7f60c7490000 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

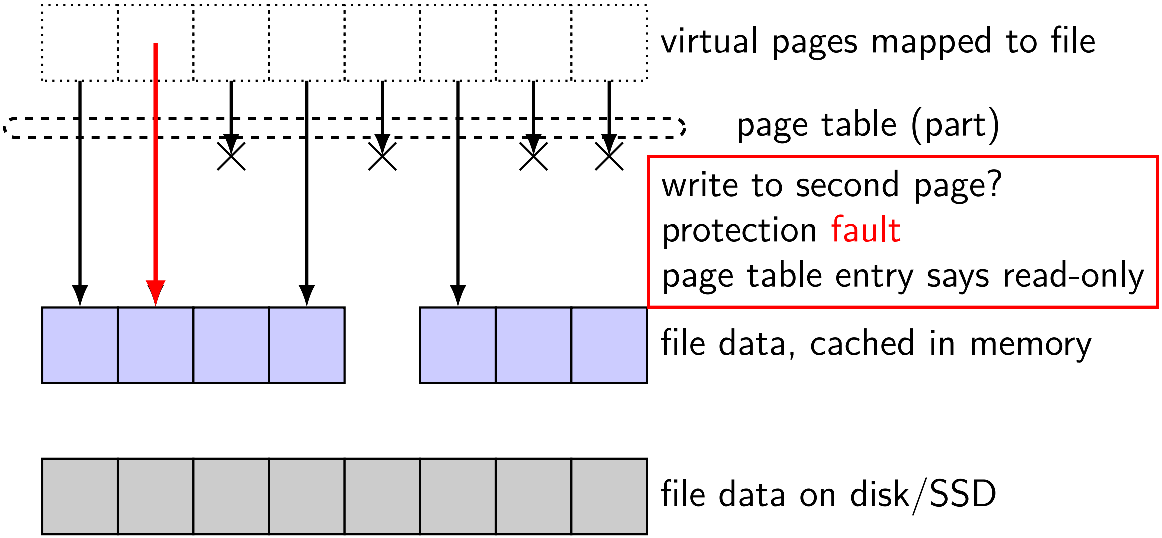
```
int fd = open("/bin/cat", O_RDONLY);
mmap(0x60b000, 0x1000, PROT_READ | PROT_WRITE,
     MAP_PRIVATE, fd, 0xb000);
```

(aside: probably used for global variables)

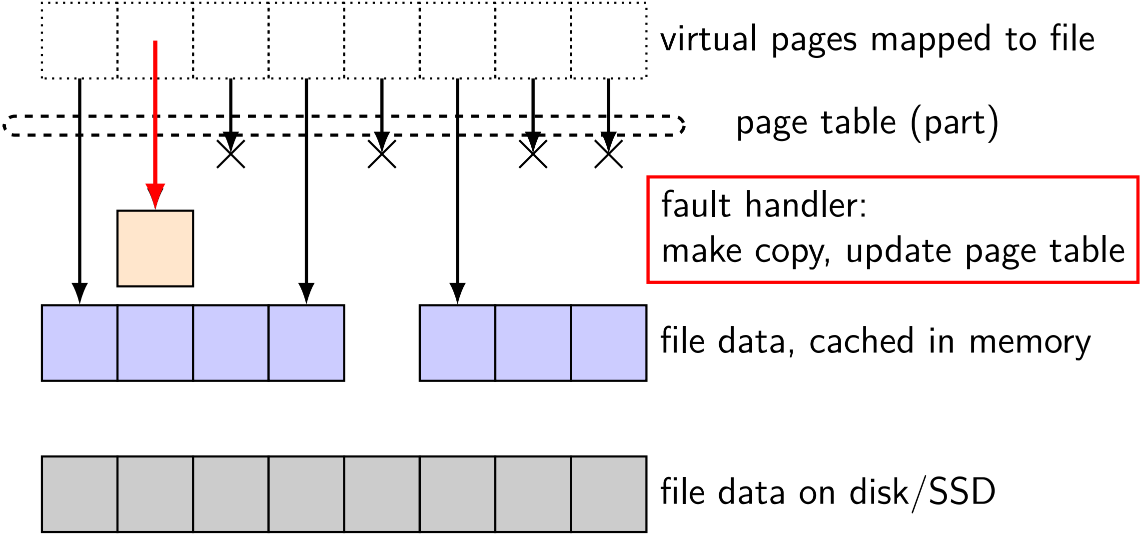
mapped pages (copy-on-write)



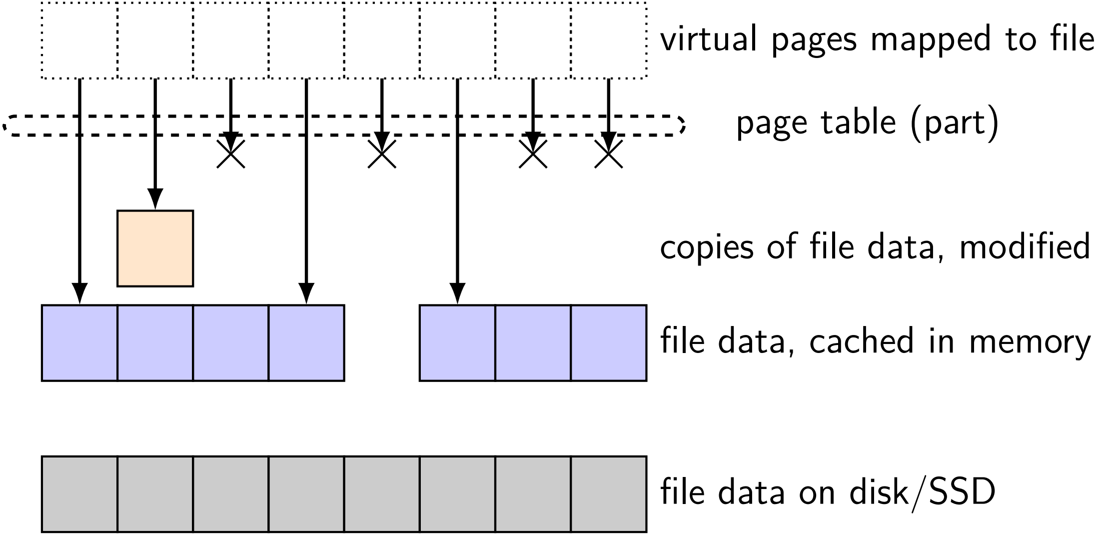
mapped pages (copy-on-write)



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 soln

virtual `0x10000–0x1FFFF` (64KB) $\rightarrow$ "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

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

4,5: copy for 0x4000-0x4FFF, 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

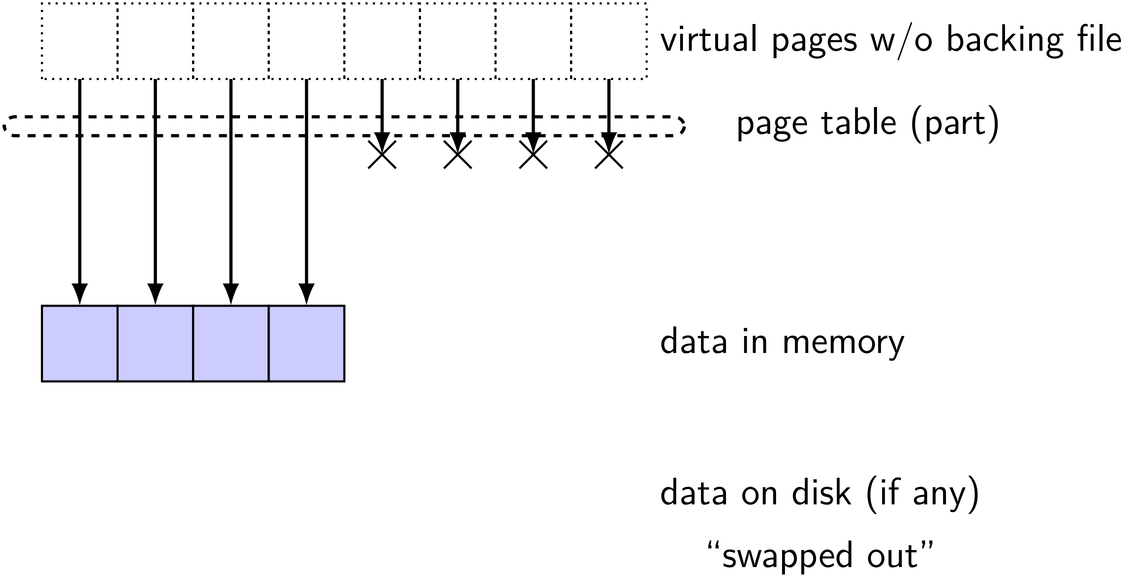
Linux maps

```
$ 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]
```

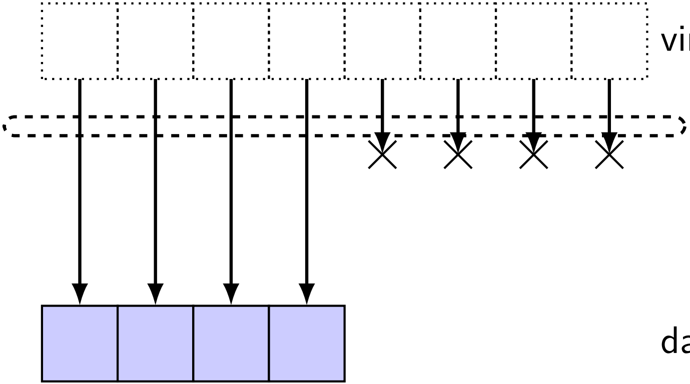
as if:

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

mapped pages (no backing file)



mapped pages (no backing file)



virtual pages w/o backing file

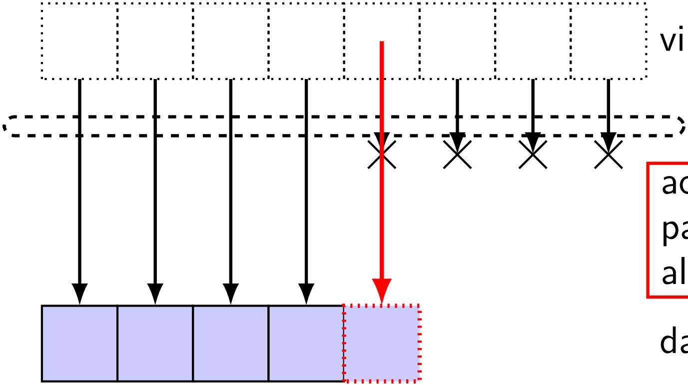
page table (part)

data in memory

data on disk (if any)

"swapped out"

mapped pages (no backing file)



virtual pages w/o backing file

page table (part)

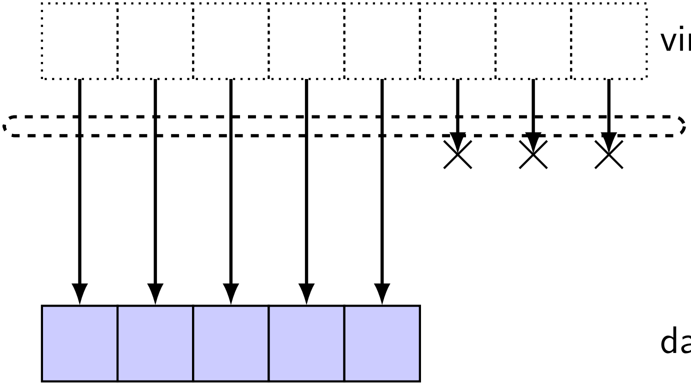
access new page
page fault handler
allocates on demand

data in memory

data on disk (if any)

"swapped out"

mapped pages (no backing file)



virtual pages w/o backing file

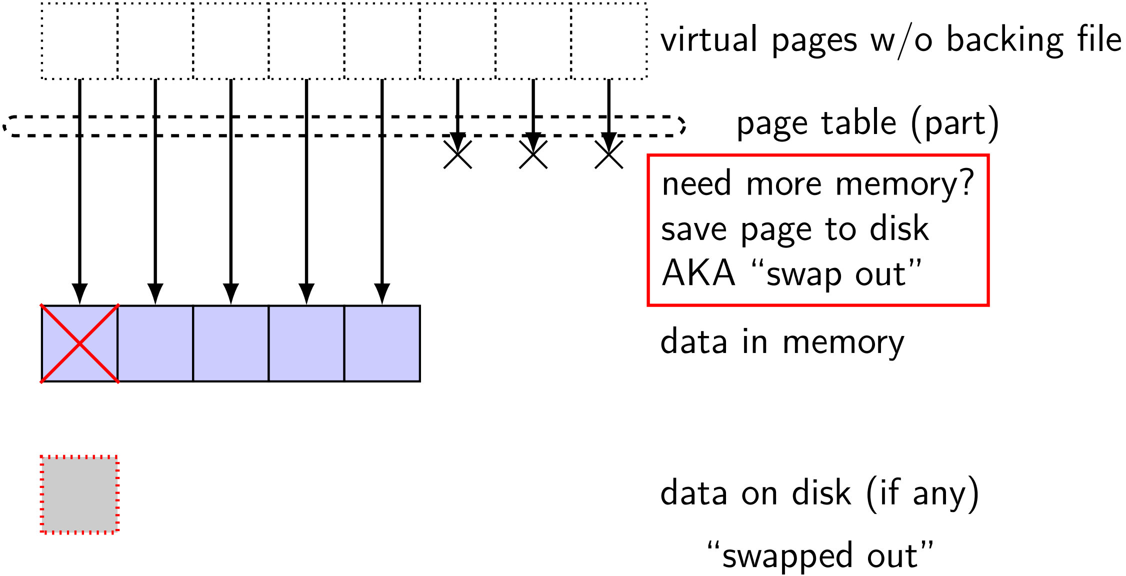
page table (part)

data in memory

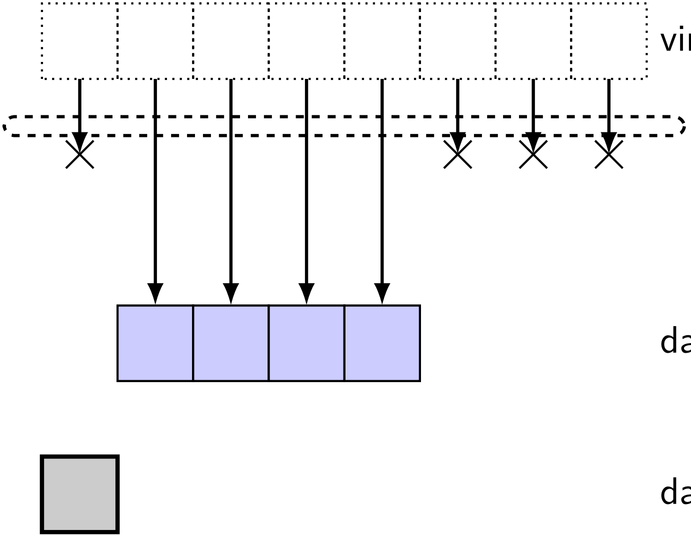
data on disk (if any)

“swapped out”

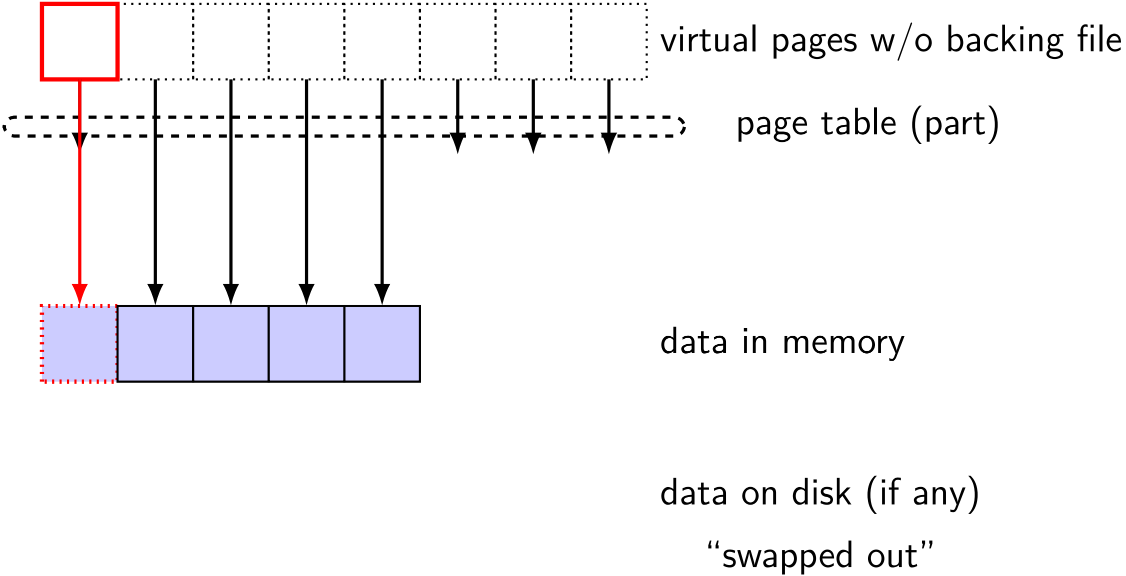
mapped pages (no backing file)



mapped pages (no backing file)



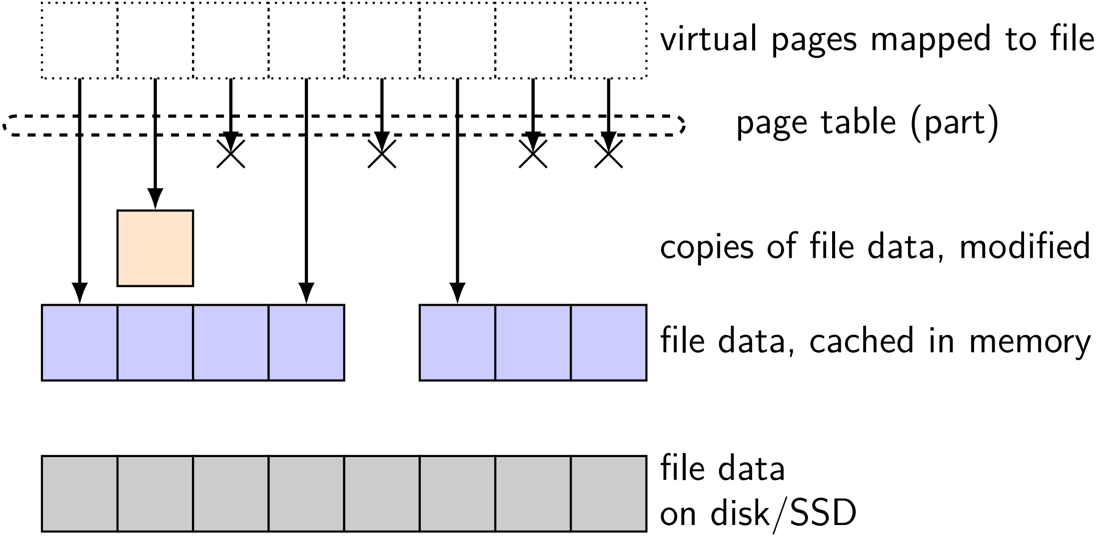
mapped pages (no backing file)



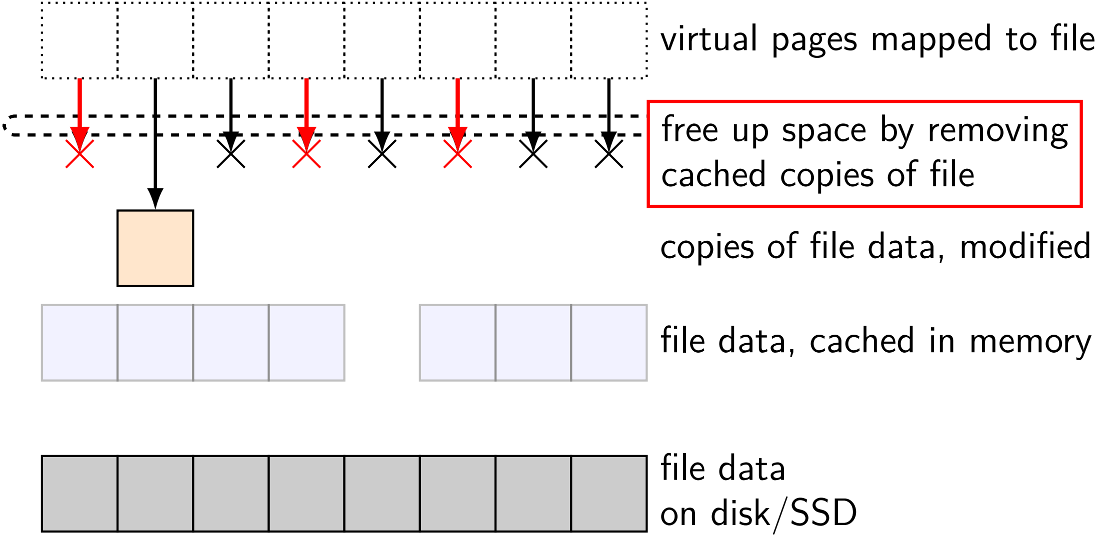
Linux maps

```
$ 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]
```

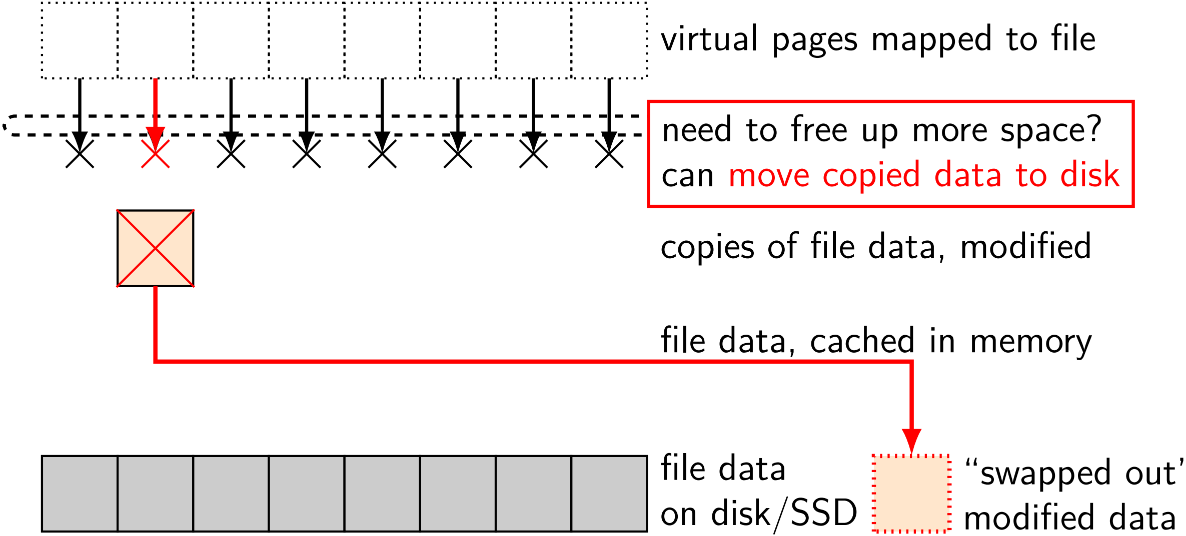
swapping with copy-on-write



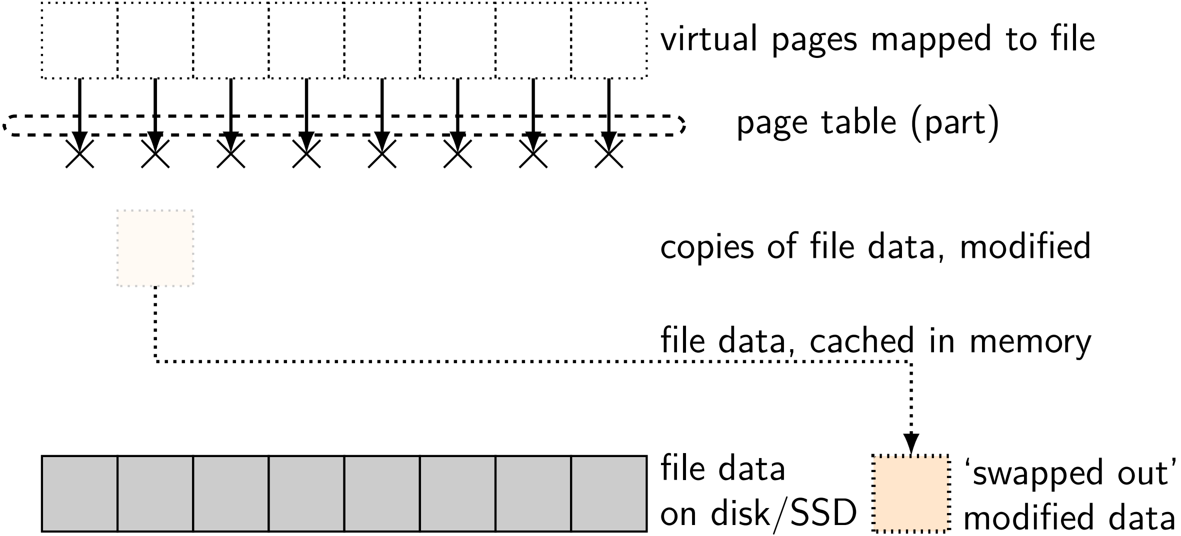
swapping with copy-on-write



swapping with copy-on-write



swapping with copy-on-write



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 $\rightarrow$ physical page

- handle cache hits

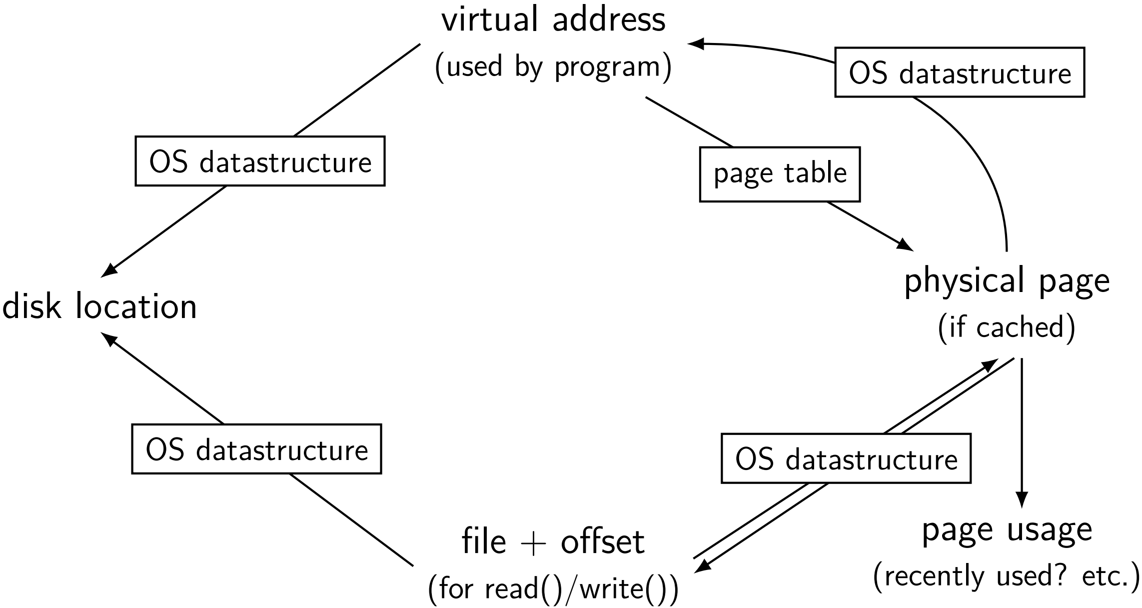
find backing location based on virtual address/file+offset

- handle cache misses

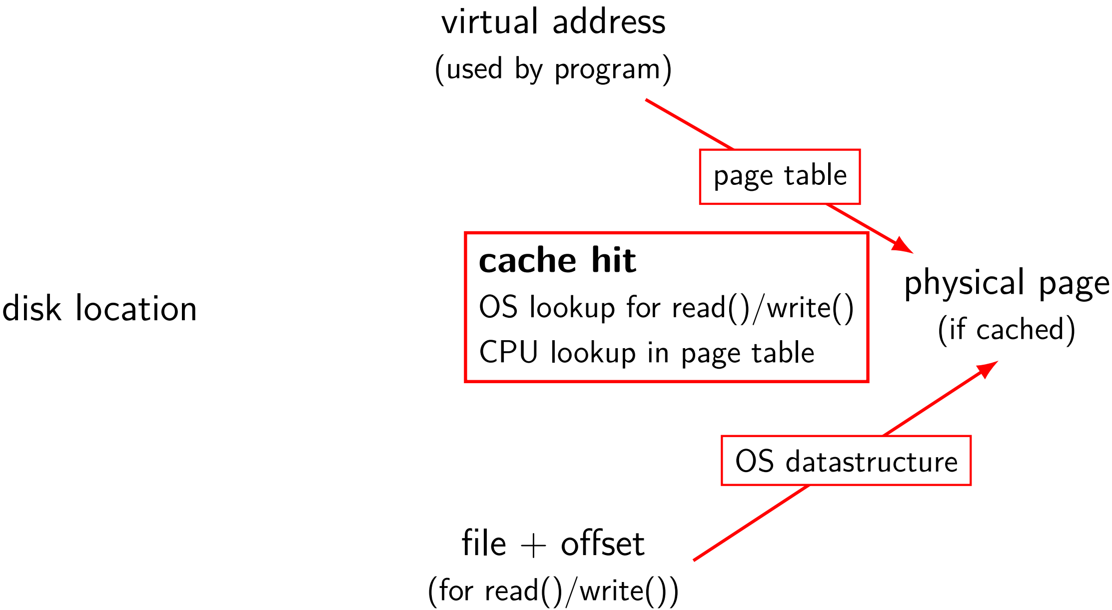
track information about each physical page

- handle page allocation
- handle cache eviction

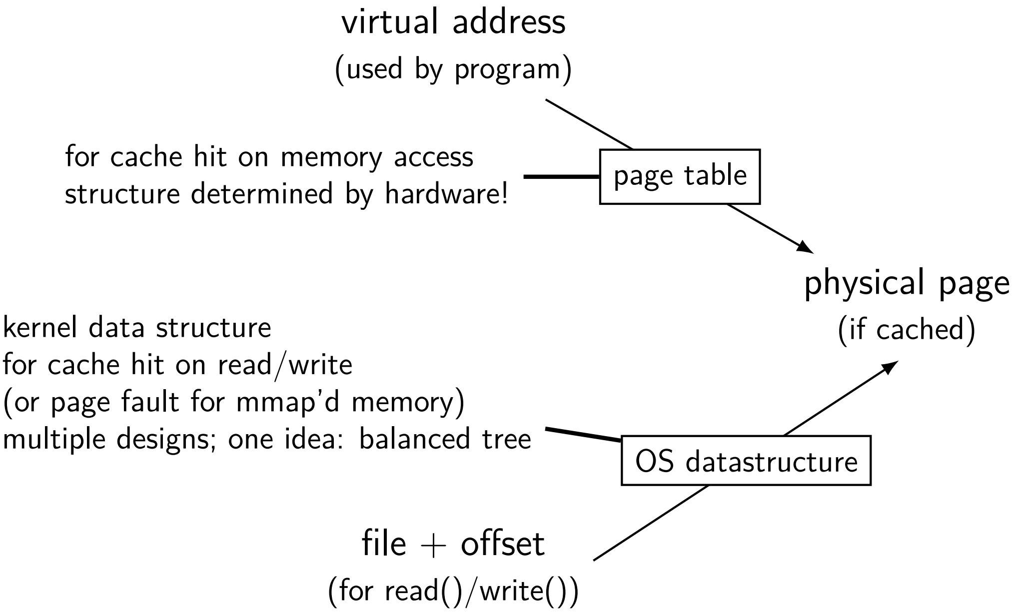
page cache components



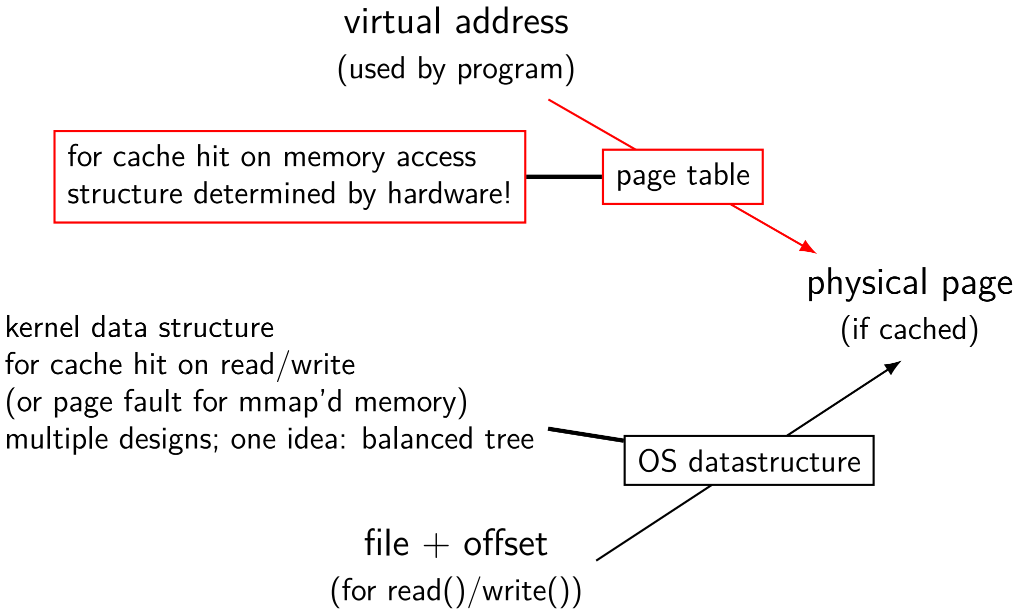
page cache components



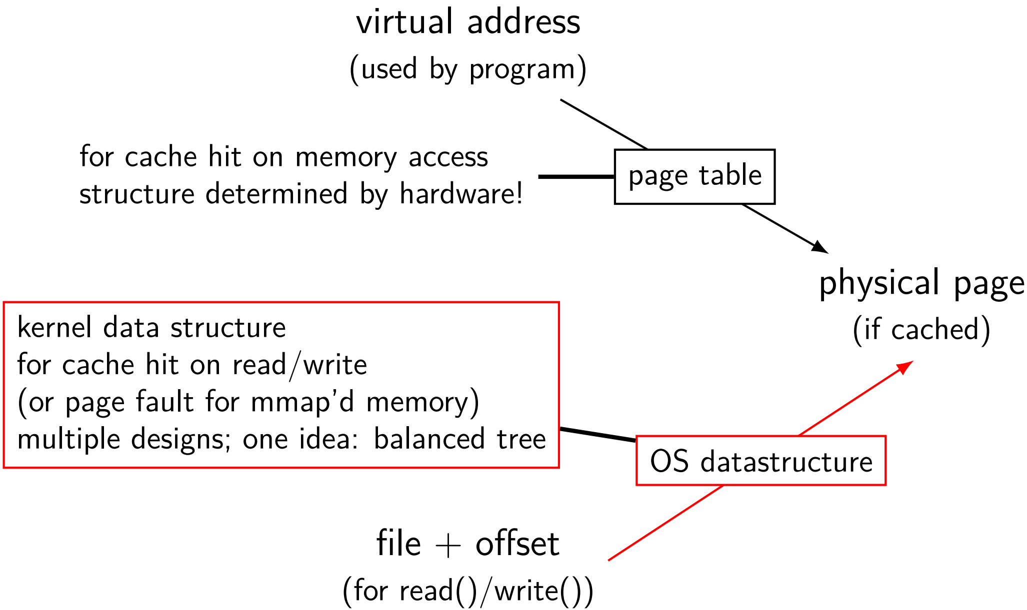
virtual addr/file offset to physical page



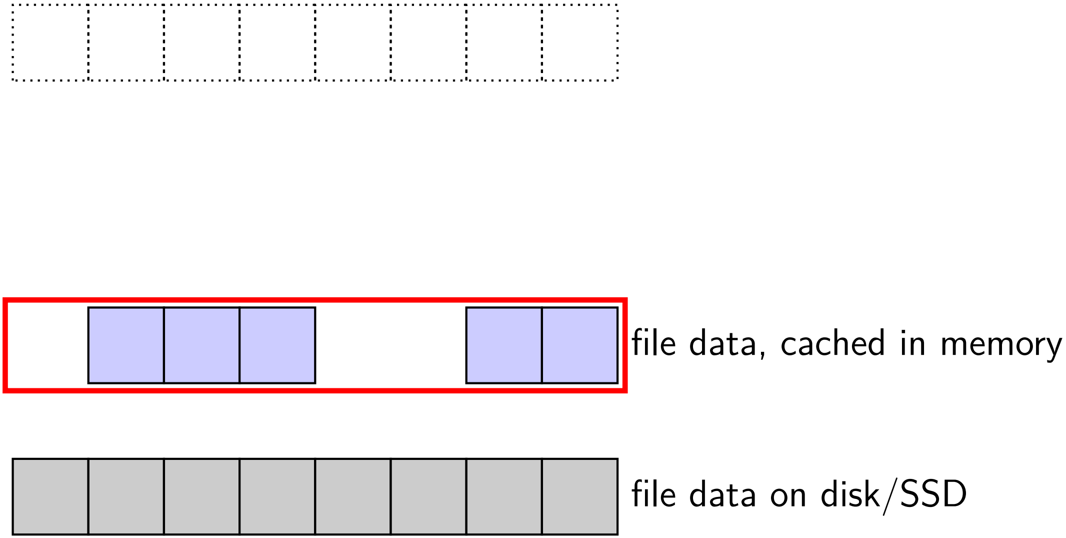
virtual addr/file offset to physical page



virtual addr/file offset to physical page



mapped pages (read/write, shared)



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

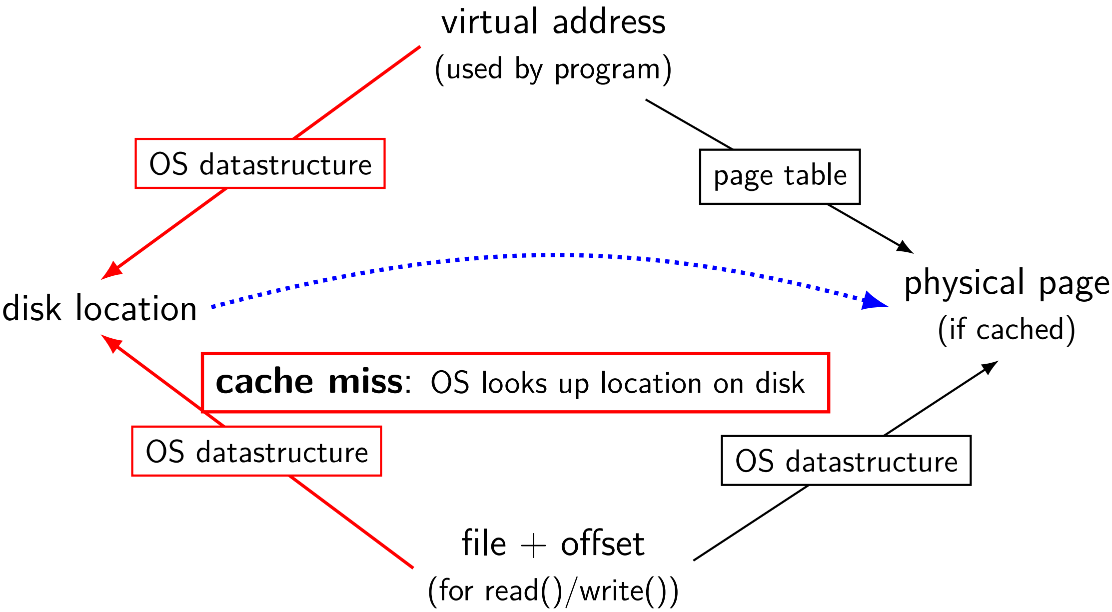
case 1: there's an unused page, just use that (easy)

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

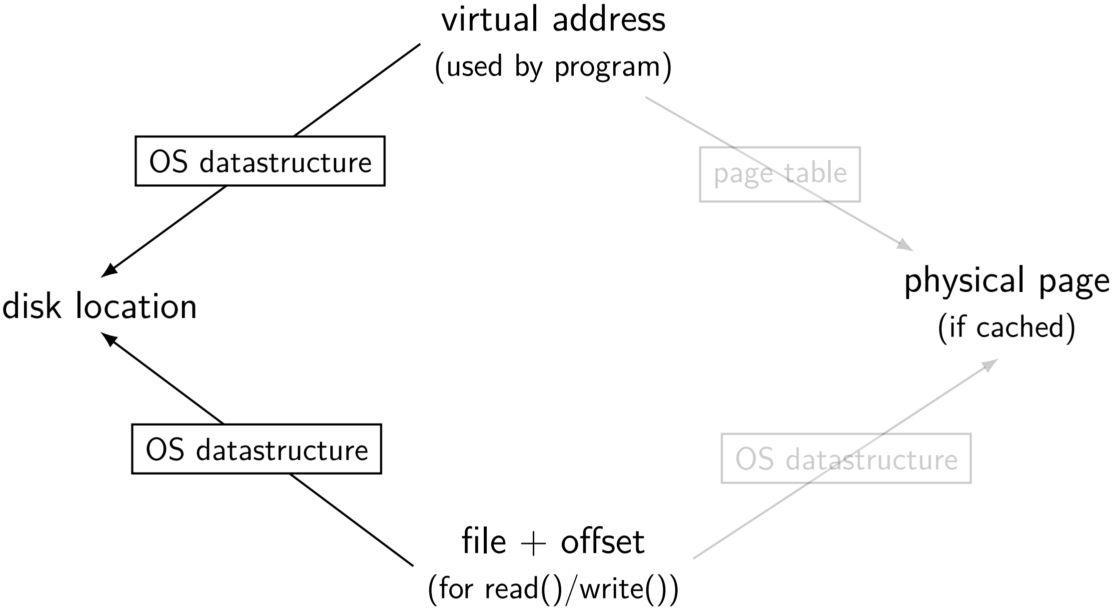
step 2: load new, more important in its place

needs some way of knowing location of data

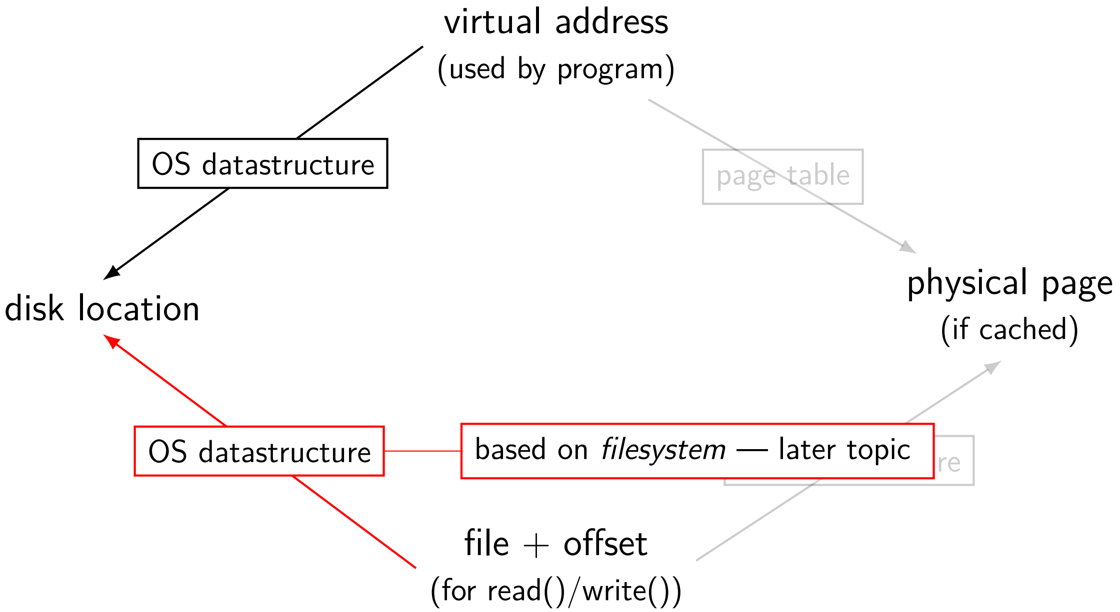
page cache components



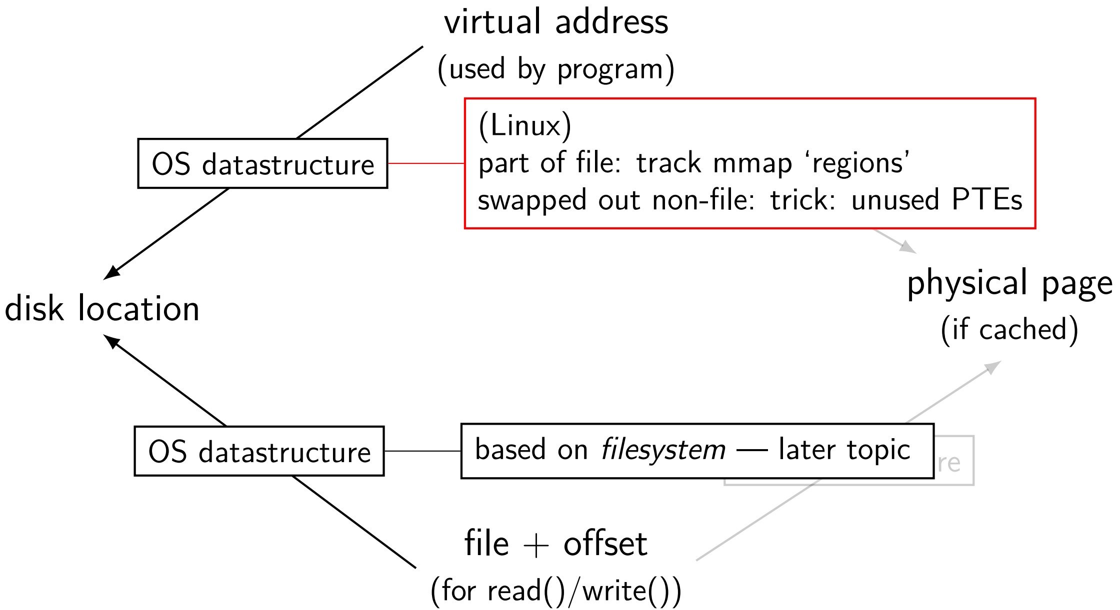
virtual address/file offset → location on disk



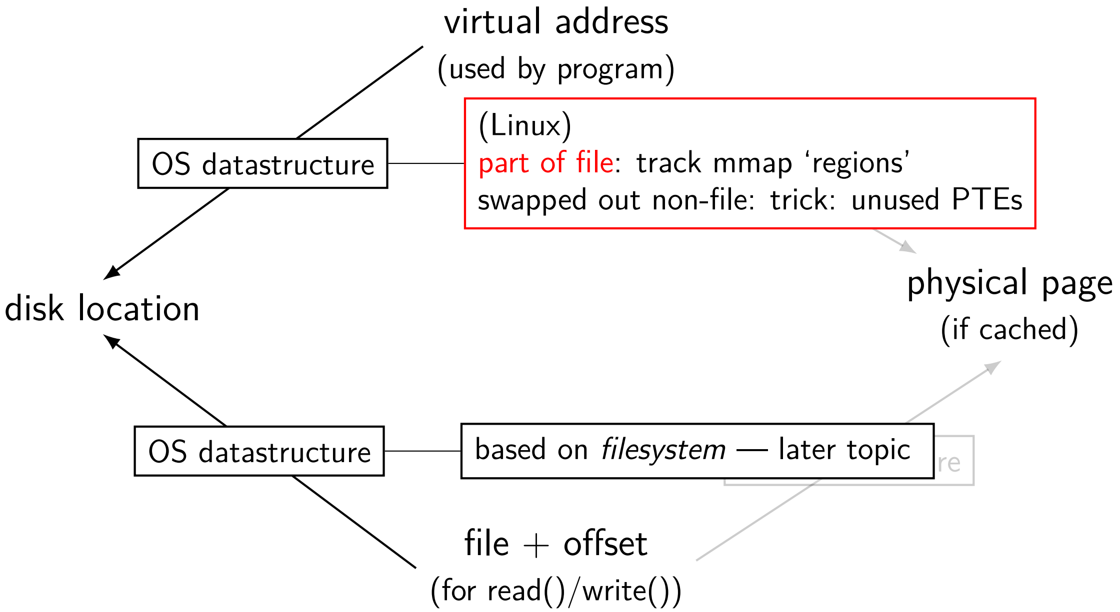
virtual address/file offset → location on disk



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 ... bc-2.19
7f60c ... bc-2.19
7f60c ... bc-2.19
7f60c ... bc-2.19
7f60c
7f60c ... -2.19.so
7f60c
7f60c
7f60c ... -2.19.so
7f60c ... -2.19.so
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) ...

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
- …

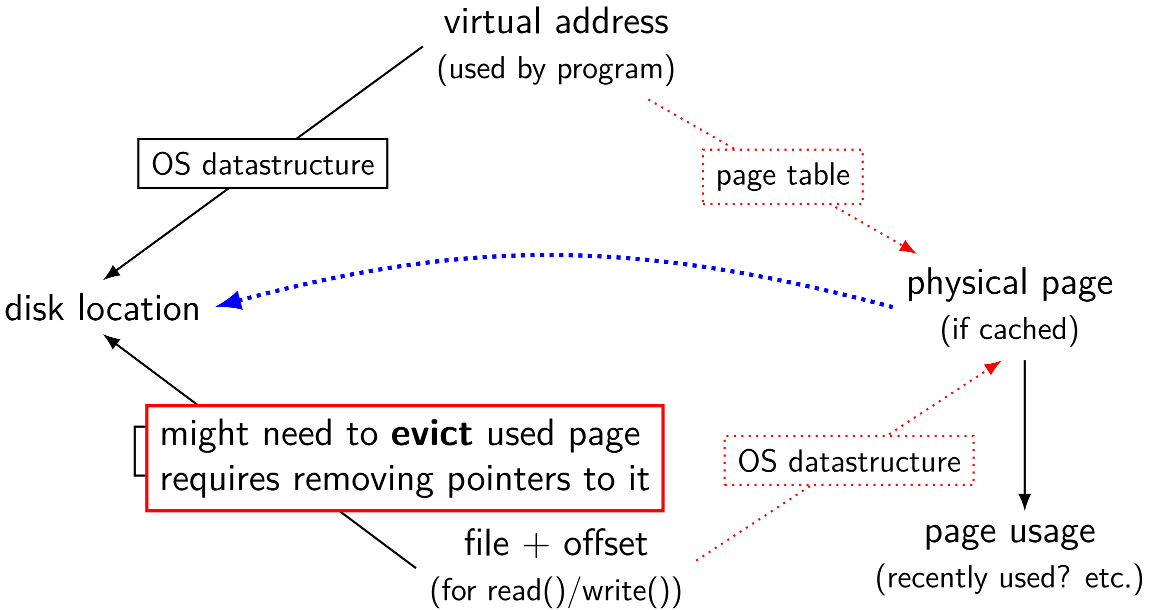
if needed, save victim page to disk

going to require:

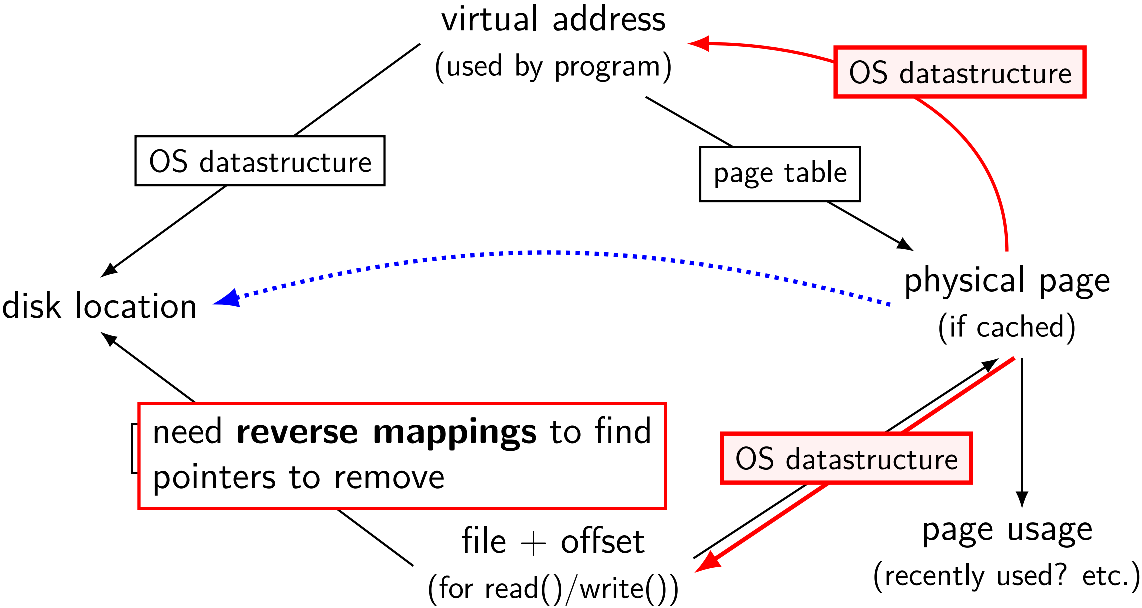
way to find page tables, etc. using page

way to detect whether it needs to be saved to disk

page cache components



page cache components



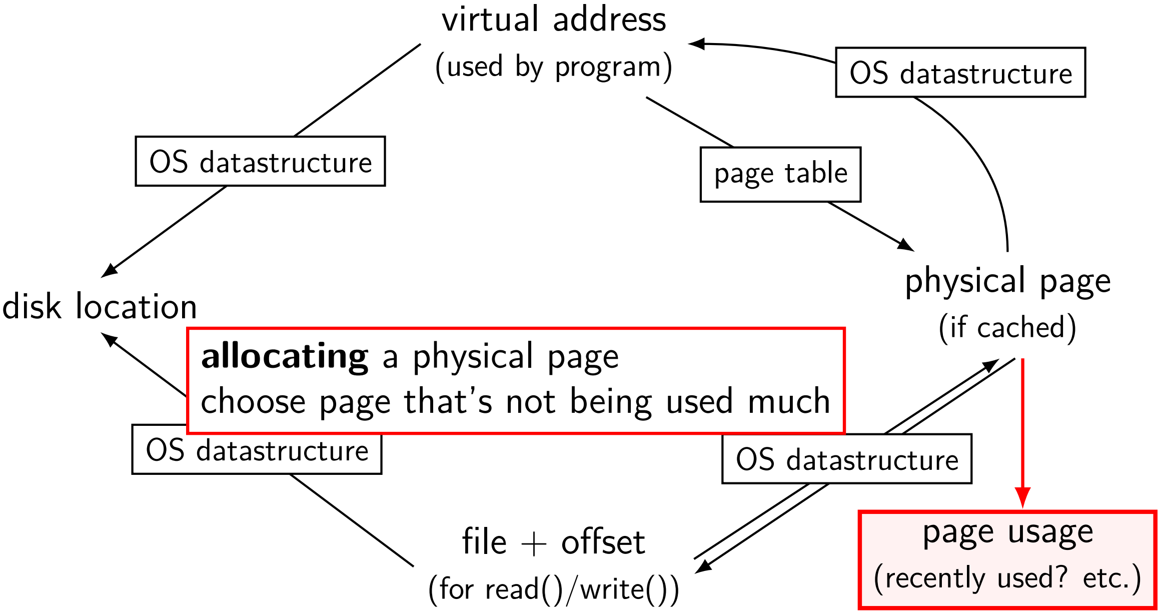
tracking physical pages: finding mappings

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

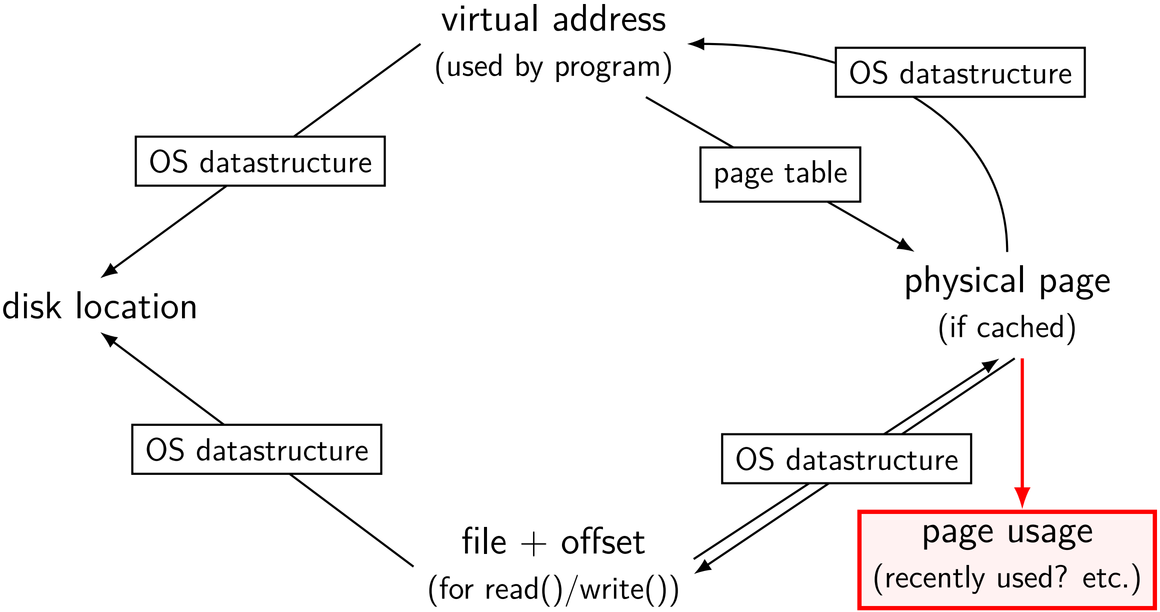
need to track where every page is used!

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

page cache components



page cache components



page replacement goals

hit rate: minimize number of misses

throughput: minimize overhead/maximize performance

fairness: every process/user gets its 'share' of memory

will start with optimizing hit rate

max hit rate $\approx$ max throughput

optimizing hit rate almost optimizes throughput, but…

max hit rate $\approx$ max throughput

optimizing hit rate almost optimizes throughput, but…

cache miss costs are variable

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

being proactive?

can avoid misses by “reading ahead”

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

can save modified pages to disk in the background

we will get back to this later

for now — only access/evict on demand

optimizing for hit-rate

assuming:

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

best possible page replacement algorithm: Belady's MIN

replace the page in memory accessed furthest in the future

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

optimizing for hit-rate

assuming:

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

best possible page replacement algorithm: Belady's MIN

replace the page in memory accessed furthest in the future

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

impossible to implement in practice, but...

Belady's MIN

referenced (virtual) pages:

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:

time →

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

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

Belady's MIN

referenced (virtual) pages:

time →

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

Belady's MIN exercise

referenced (virtual) pages:

time →

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

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

practically optimizing for hit-rate

recall?: locality assumption

temporal locality: things accessed now will be accessed again soon

(for now: not concerned about spatial locality)

more possible policies: least recently used or least frequently used

practically optimizing for hit-rate

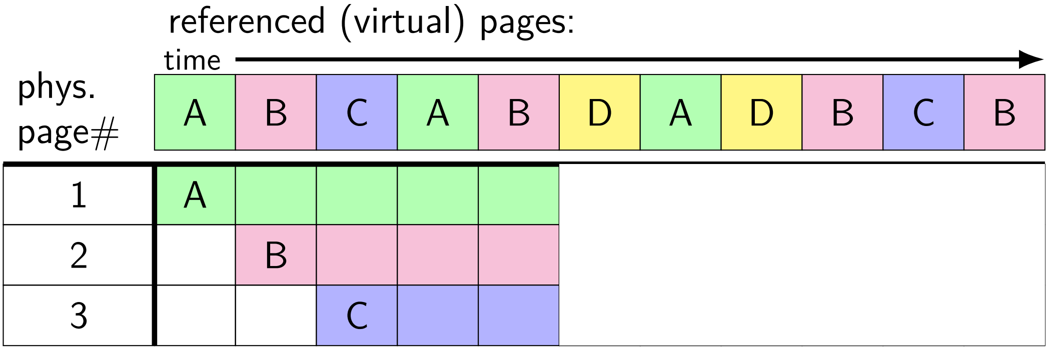
recall?: locality assumption

temporal locality: things accessed now will be accessed again soon

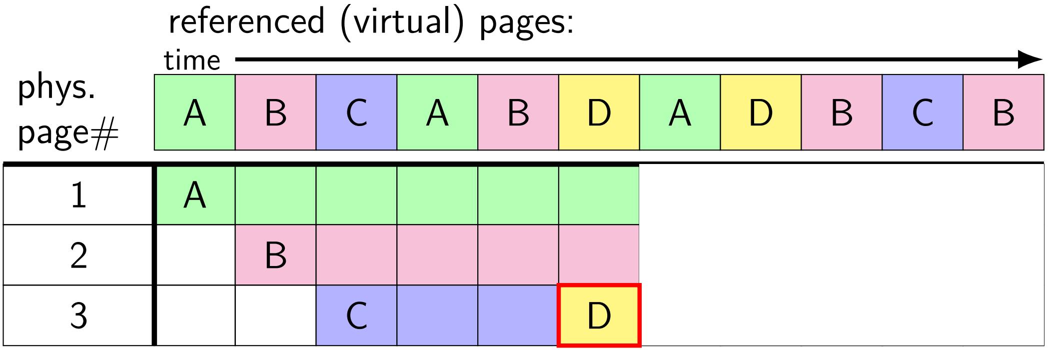
(for now: not concerned about spatial locality)

more possible policies: least recently used or least frequently used

least recently used (the good case)



least recently used (the good case)



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

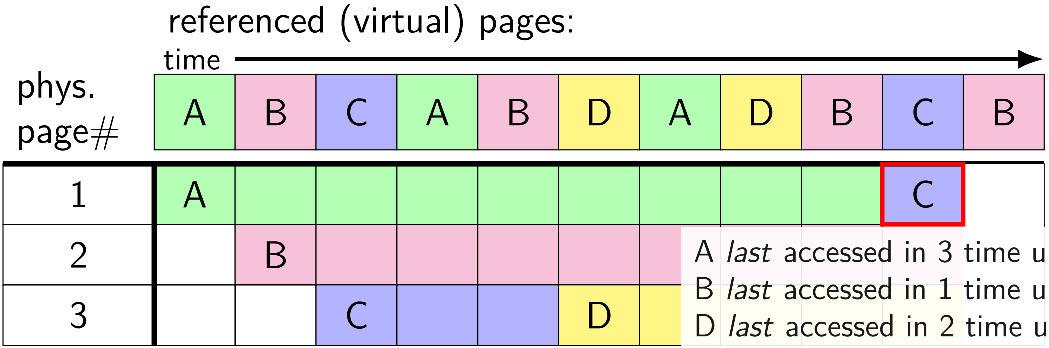
least recently used (the good case)

referenced (virtual) pages:

time →

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

least recently used (the good case)



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

least recently used (the good case)

referenced (virtual) pages:

time →

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

least recently used (the worst case)

time →

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

least recently used (the worst case)

time →

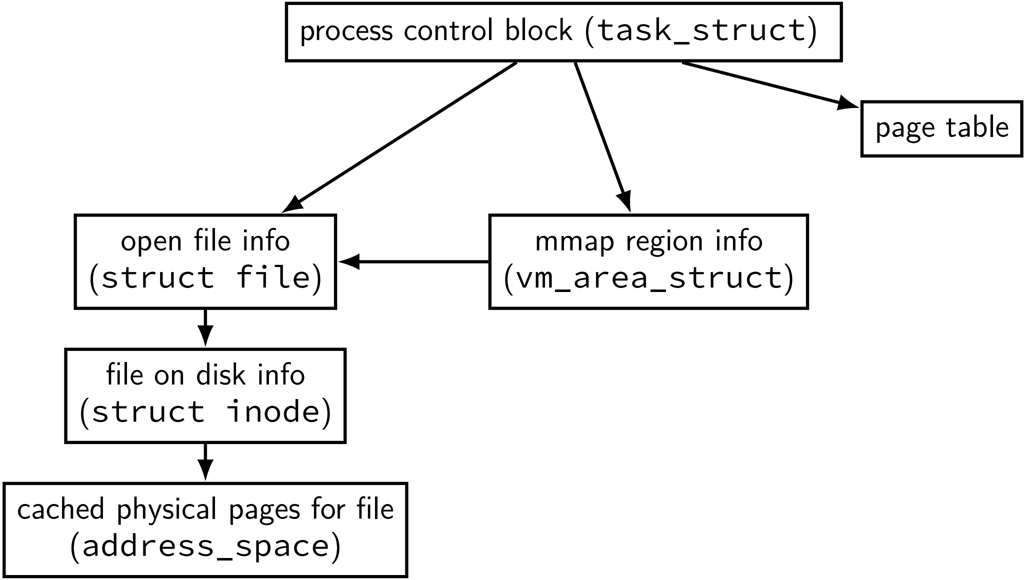
phys. page#	A	B	C	D	A	B	C	D	A	B	C
1	A			D			C			B	
2		B			A			D			C
3			C			B			A		

8 replacements with LRU
versus 3 replacements with MIN:

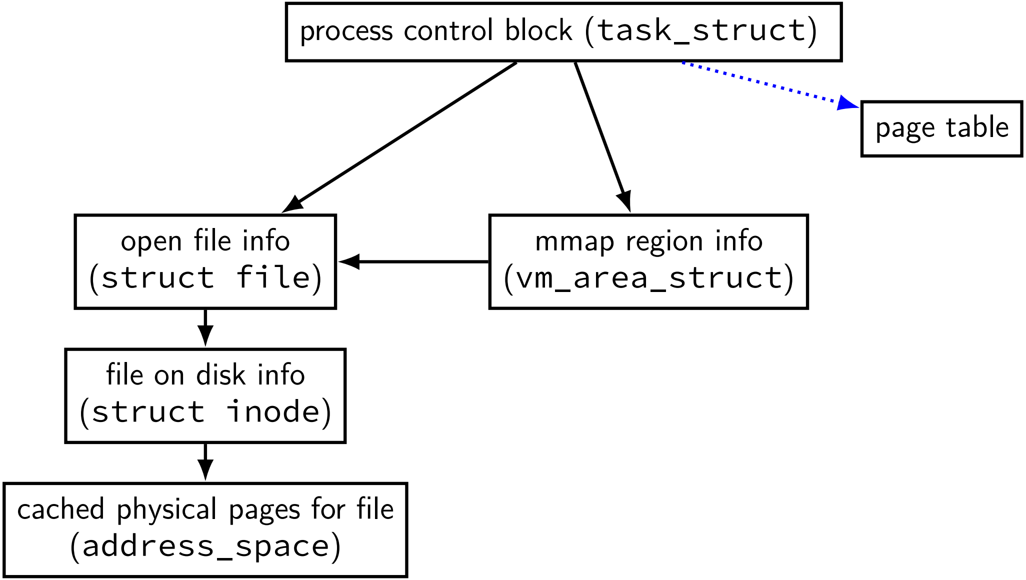
1	A									B	
2		B					C				
3			C	D							

backup slides

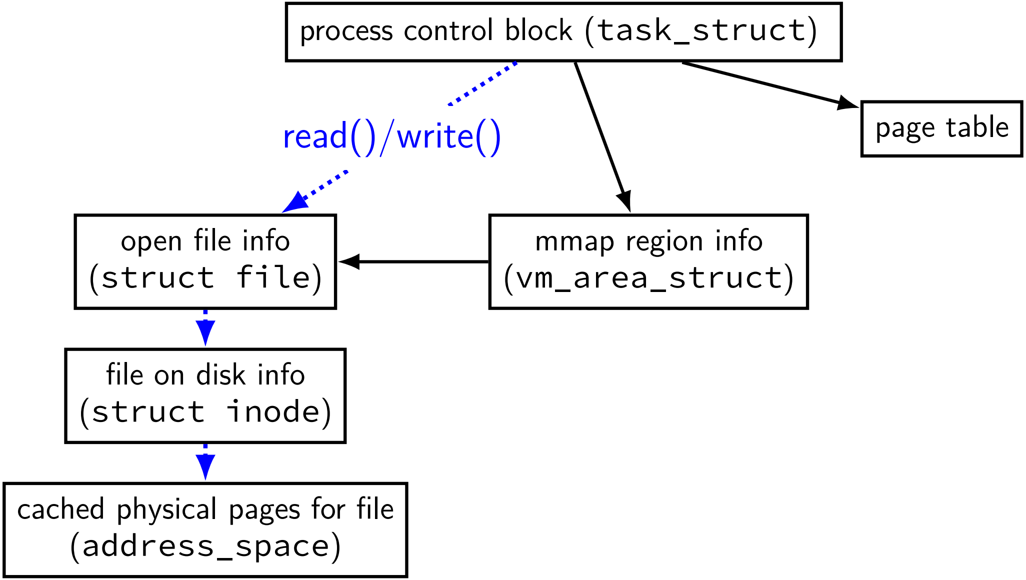
Linux: forward mapping



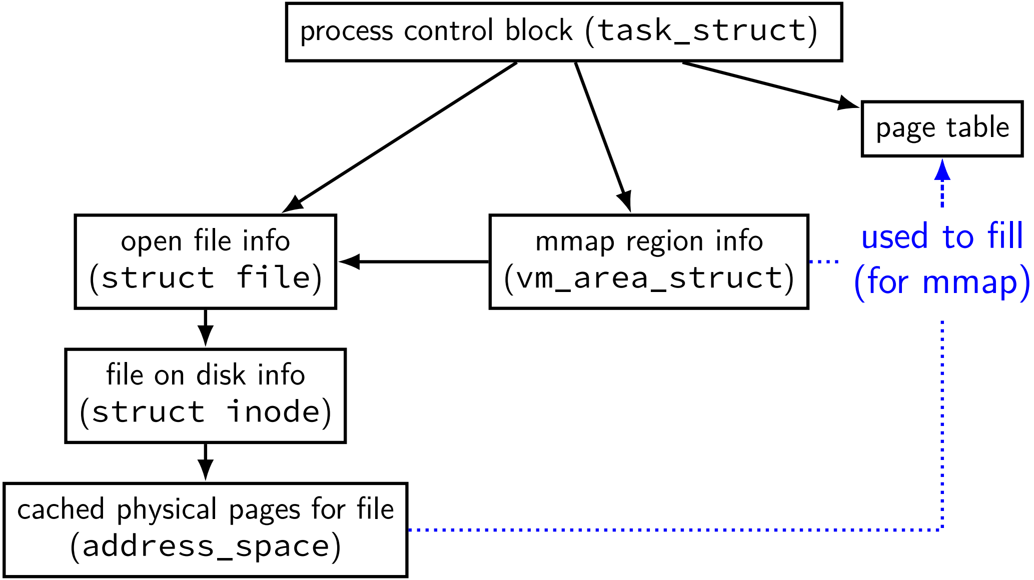
Linux: forward mapping



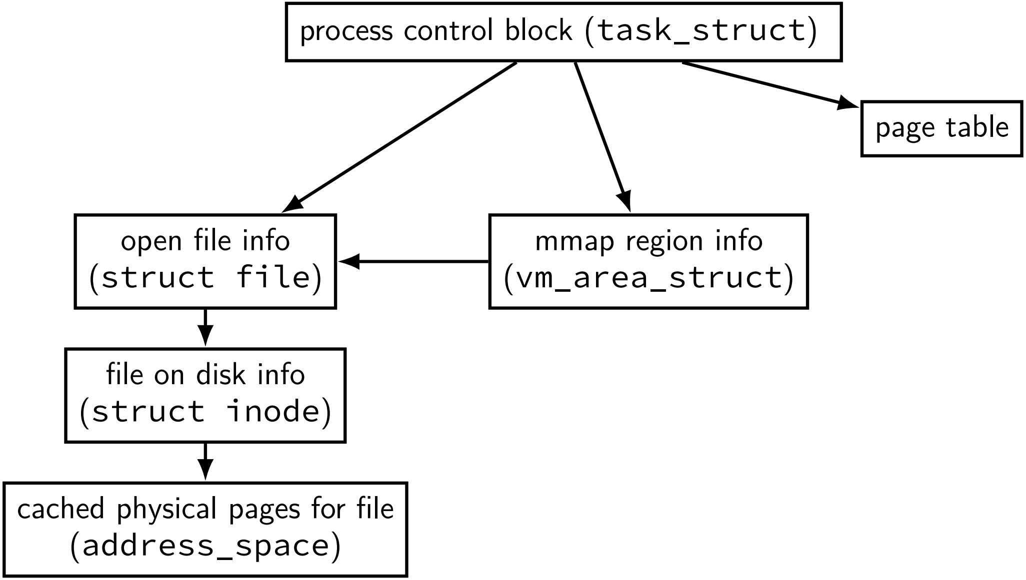
Linux: forward mapping



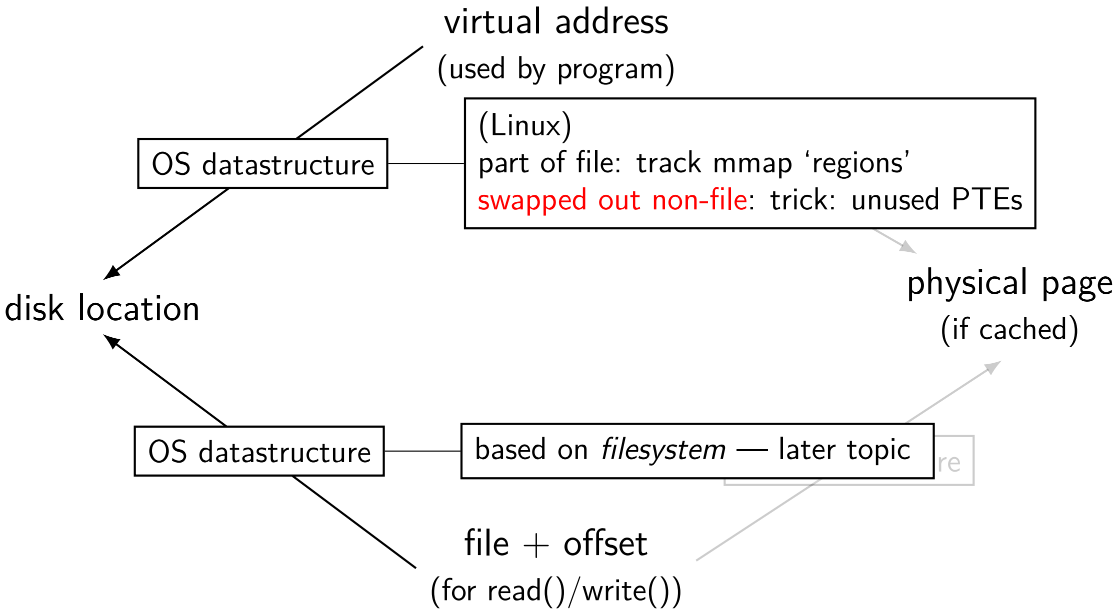
Linux: forward mapping



Linux: forward mapping



virtual address/file offset → location on disk



Linux: tracking swapped out pages

need to lookup location on disk

potentially one location for every virtual page

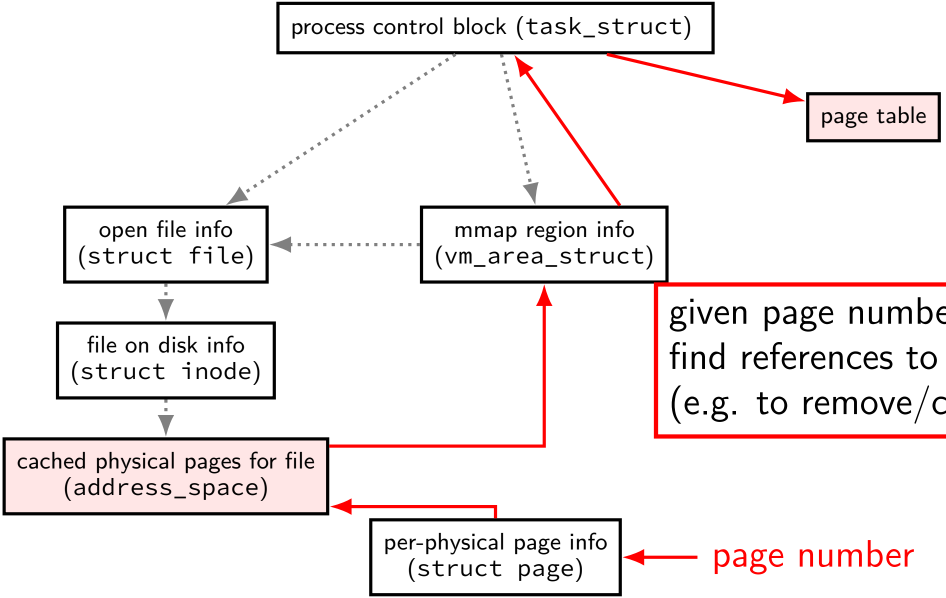
trick: store location in “ignored” part of page table entry

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

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

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

Linux: reverse mapping (file pages)



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

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

xv6: setting process page tables (`exec()`)

exec step 1: create new page table with kernel mappings

- done in `setupkvm()`, which calls `mappages()`

exec step 2a: allocate memory for executable pages

- `allocuvm()` in loop
- new physical pages chosen by `kalloc()`

exec step 2b: load from executable file

- copying from executable file implemented by `loaduvm()`

exec step 3: allocate pages for heap, stack (`allocuvm()` calls)

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, ...)

predicting the future?

can't really...

look for common patterns

working set intuition

say we're executing a loop

what memory does this require?

code for the loop

code for functions called in the loop

and functions they call

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

only uses a subset of the program's memory

the working set model

one common pattern: working sets

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

...called its *working set*

rest of memory is inactive

...until program switches to different working set

working sets and running many programs

give each program its working set

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

inactive — won't be used

working sets and running many programs

give each program its working set

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

- inactive — won't be used

replacement policy: identify working sets $\approx$ recently used data

replace anything that's not in in it

cache size versus miss rate

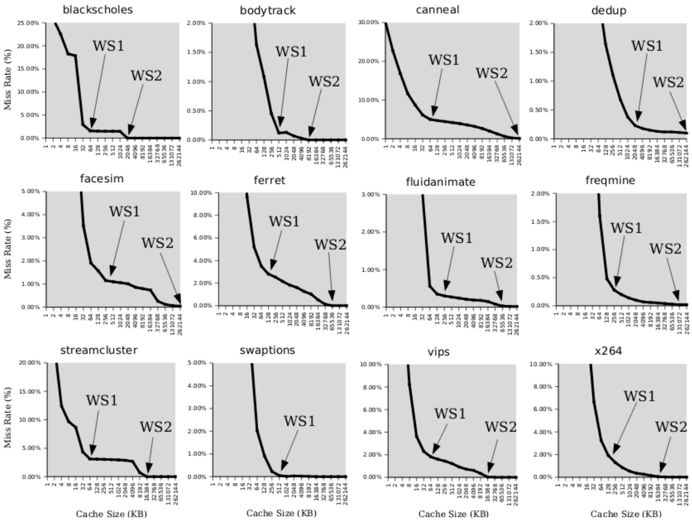


Figure 3: Miss rates versus cache size. Data assumes a shared 4-way associative cache with 64 byte lines. WS1 and WS2 refer to important working sets which we analyze in more detail in Table 2. Cache requirements of PARSEC benchmark programs can reach hundreds of megabytes.

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

estimating working sets

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

except when program switching working sets

so, what a program recently used $\approx$ working set

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

estimating working sets

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

except when program switching working sets

so, what a program recently used $\approx$ working set

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

second chance cons

performs poorly with big memories…

may need to scan through lots of pages to find unaccessed

likely to count accesses from a long time ago

want some variation to tune its sensitivity

second chance cons

performs poorly with big memories...

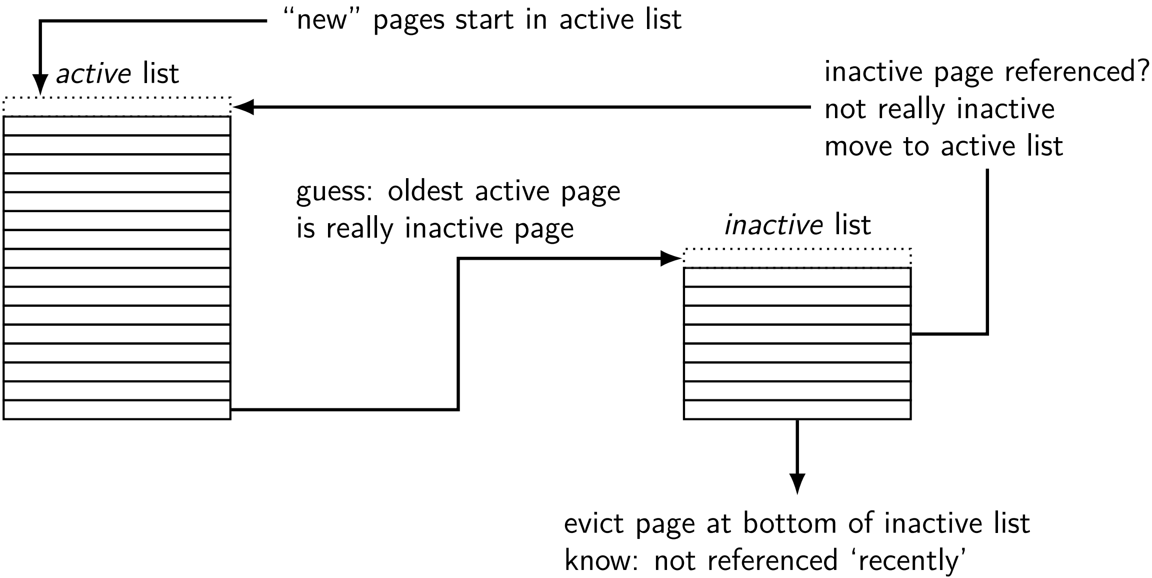
may need to scan through lots of pages to find unaccessed

likely to count accesses from a long time ago

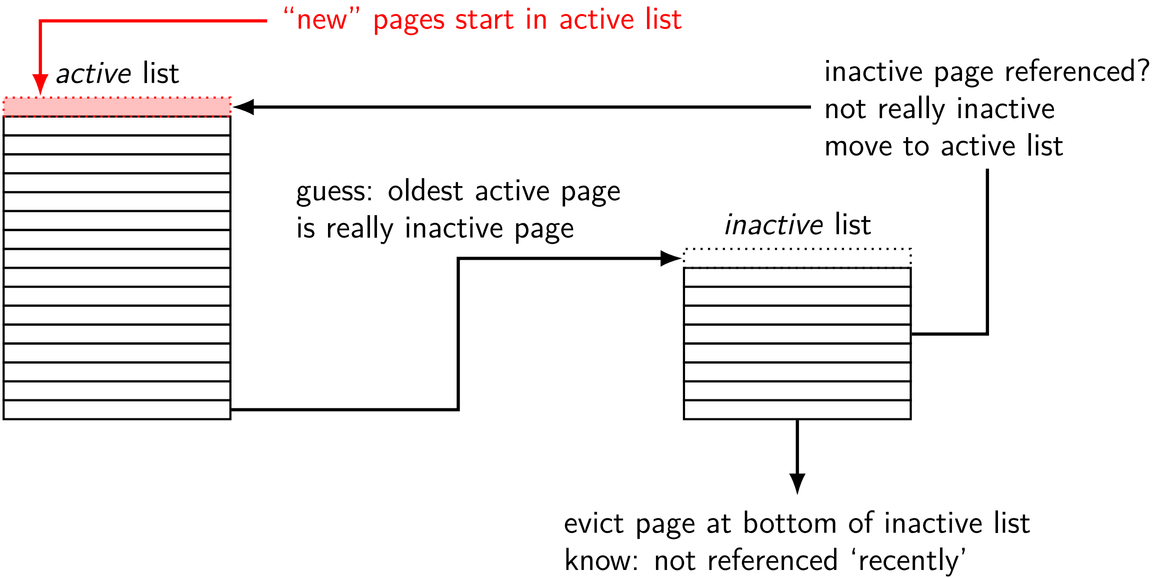
want some variation to tune its sensitivity

one idea: smaller list of pages to scan for accesses

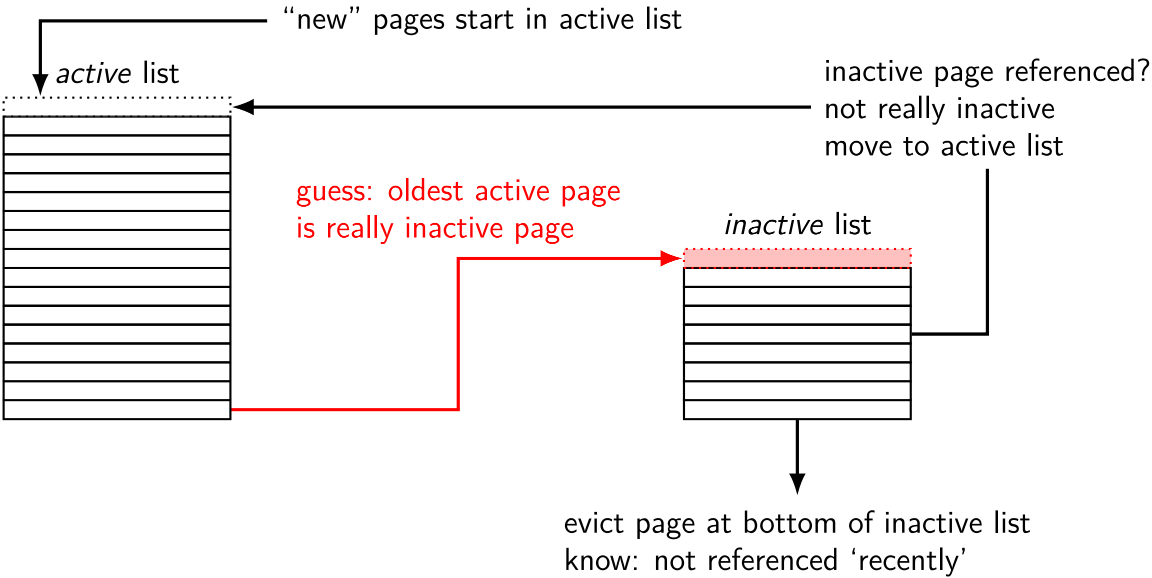
approximating LRU: SEQ



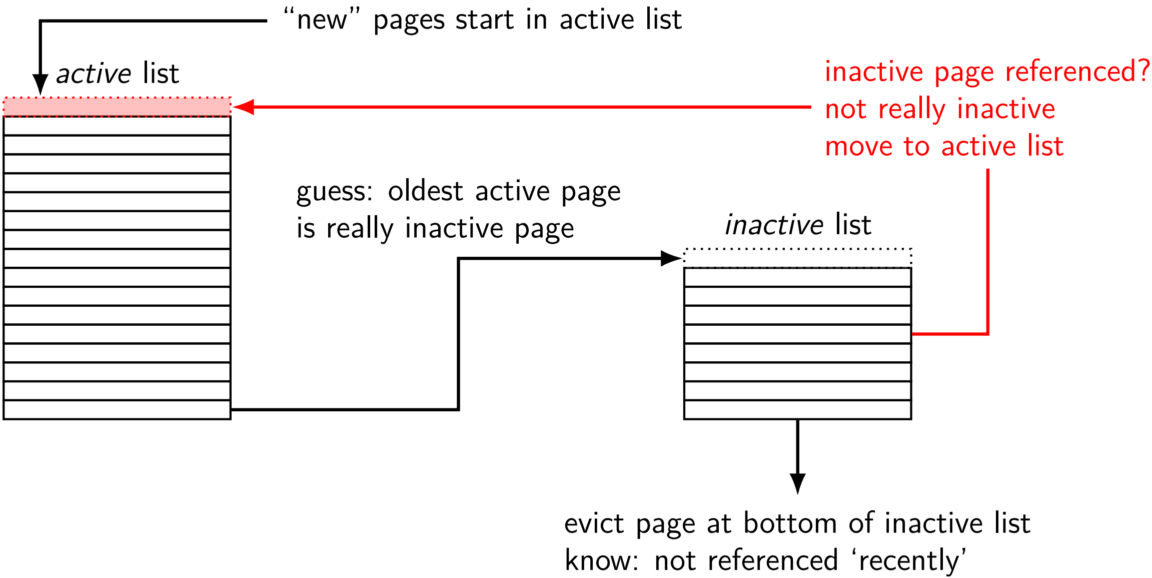
approximating LRU: SEQ



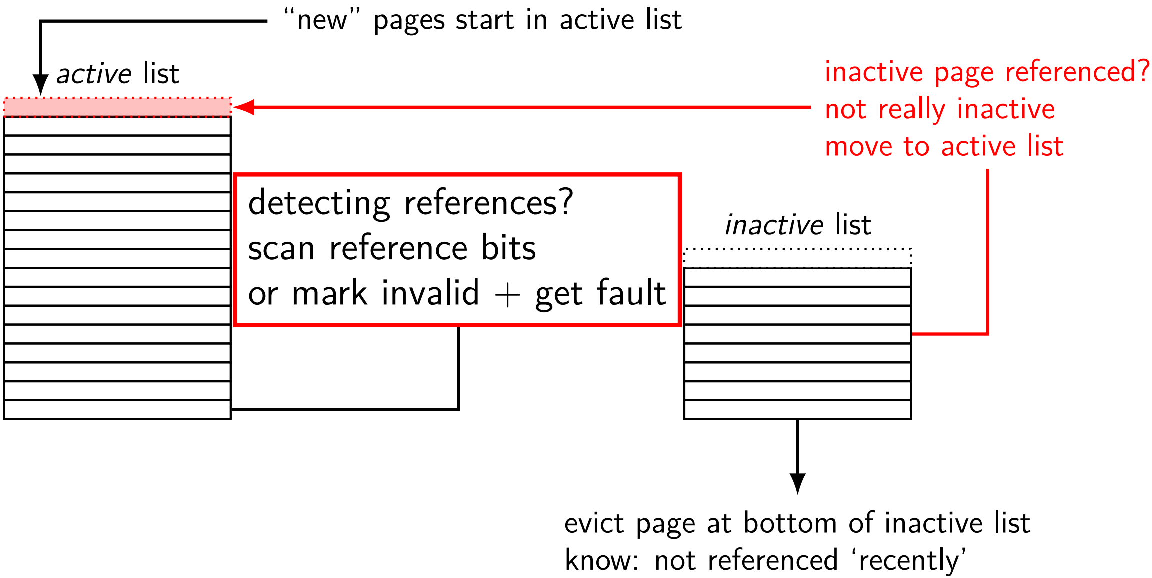
approximating LRU: SEQ



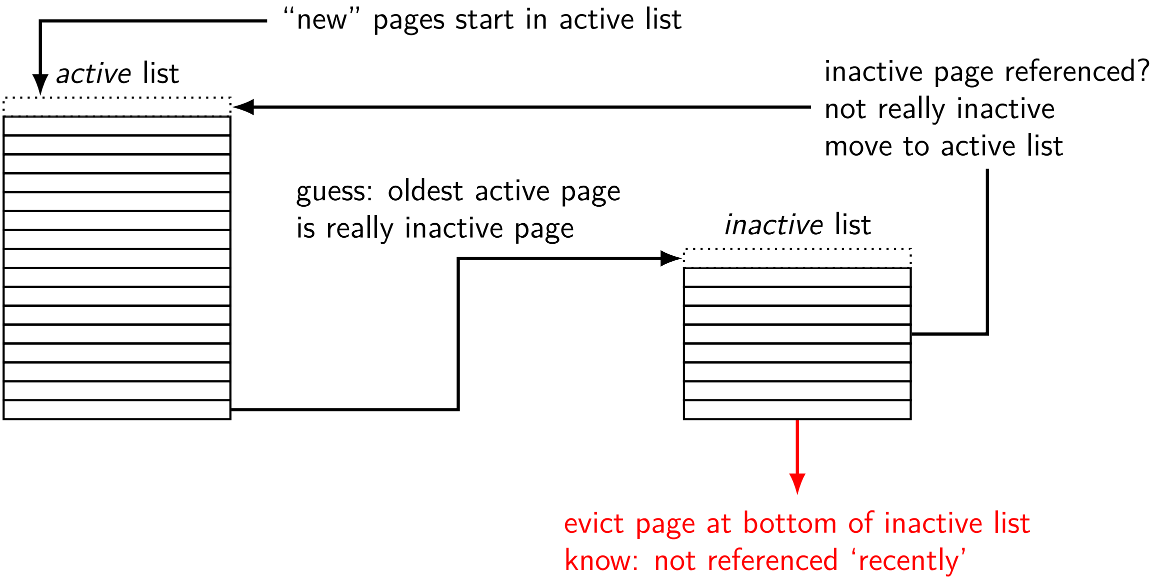
approximating LRU: SEQ



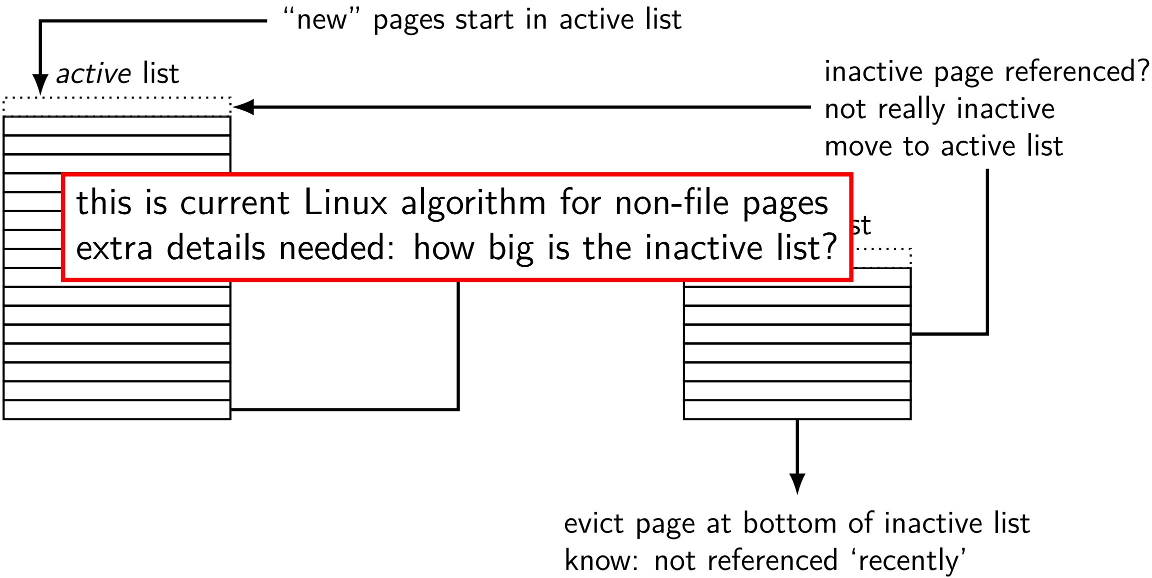
approximating LRU: SEQ



approximating LRU: SEQ



approximating LRU: SEQ



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

CLOCK-Pro: special casing for one-use pages

by default, Linux tries to handle scanning of files

- one read of file data — e.g. play a video, load file into memory

basic idea: delay considering pages active until second access

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

single scans of file won't "pollute" cache

without this change: reading large files slows down other programs

- recently read part of large file steals space from active programs

readahead heuristics

exercise: devise an algorithm to detect to do readahead.

how to detect the reading pattern?

when to start reads?

how much to readahead?

readahead heuristics

exercise: devise an algorithm to detect to do readahead.

how to detect the reading pattern?

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

when to start reads?

how much to readahead?

readahead heuristics

exercise: devise an algorithm to detect to do readahead.

how to detect the reading pattern?

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

when to start reads?

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

how much to readahead?

readahead heuristics

exercise: devise an algorithm to detect to do readahead.

how to detect the reading pattern?

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

when to start reads?

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

how much to readahead?

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

problems with LRU

question: when does LRU perform poorly?

exercise: which of these is LRU bad for?

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

initial values of the shell's global variales

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

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

files that are parsed when loaded and overwritten when saved

on web server, frequently requested HTML files

problems with LRU

question: when does LRU perform poorly?

only reading things once

repeated scans of large amounts of data

problems with LRU

question: when does LRU perform poorly?

only reading things once

repeated scans of large amounts of data

both common access patterns for files

solution for LRU being bad?

one idea that Linux uses:

for *file data*, use different replacement policy

"CLOCK-PRO"

tries to avoid keeping around file data accessed only once

being proactive

previous assumption: load on demand

why is something loaded?

- page fault
- maybe because application starts

can we do better?

readahead

program accesses page 4 of a file, page 5, page 6. What's next?

readahead

program accesses page 4 of a file, page 5, page 6. What's next?

page 7 — idea: guess this

on page fault, does it look like contiguous accesses?

called readahead

readahead implementation ideas?

which of these is probably best?

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

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

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

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