☑ 1. Last time
☑ 2. Final exam
☑ 3. Your questions
☑ 4. Wrap-up

---

2. Final exam
   - 110 minute exam
   - May 17, 10am ET for ppl in East Asia
   - May 17, 12pm ET for everyone else
   - If you have another exam on 5/17 2PM ET and want to take at 10am ET, please send email to the course staff ↑ h. eod

Material
   - Readings
   - Labs
   - HWs
   - Classes → l15.txt
     [see midterm topic list]

Post midterm topics (not guaranteed to be necessary or sufficient)

mmap
I/O
   architecture
   how CPUs and devices interact

- mechanics
- polling vs. interrupts
- DMA vs. programmed I/O

- device drivers

Context switches

Synchronous vs. async I/O

User-level threading

Disks
- geometry
- performance
- interface
- scheduling (skipped in class, covered in book)

File systems
- basic objects: files, directories, metadata, links, inodes
- how does naming work?
- types of file layout
  - extents/contiguous, linked, index
  - classic Unix + FFS are variants of indexed
- analogy between inode and top-level page directory (aka L1 page table)
- tradeoffs
- performance

performance

Crash recovery
- ad hoc
- copy-on-write (COW)
- journaling (redo logging, undo logging, undo+redo)
  - WAL

RPC, client/server systems

Case study: NFS
- marquee user of RPC
- RPC: transparent or not?

protection and security
- stack smashing / buffer overflow
- Unix security model
  - access control, priviteges, setuid, attacks
- trusting trust

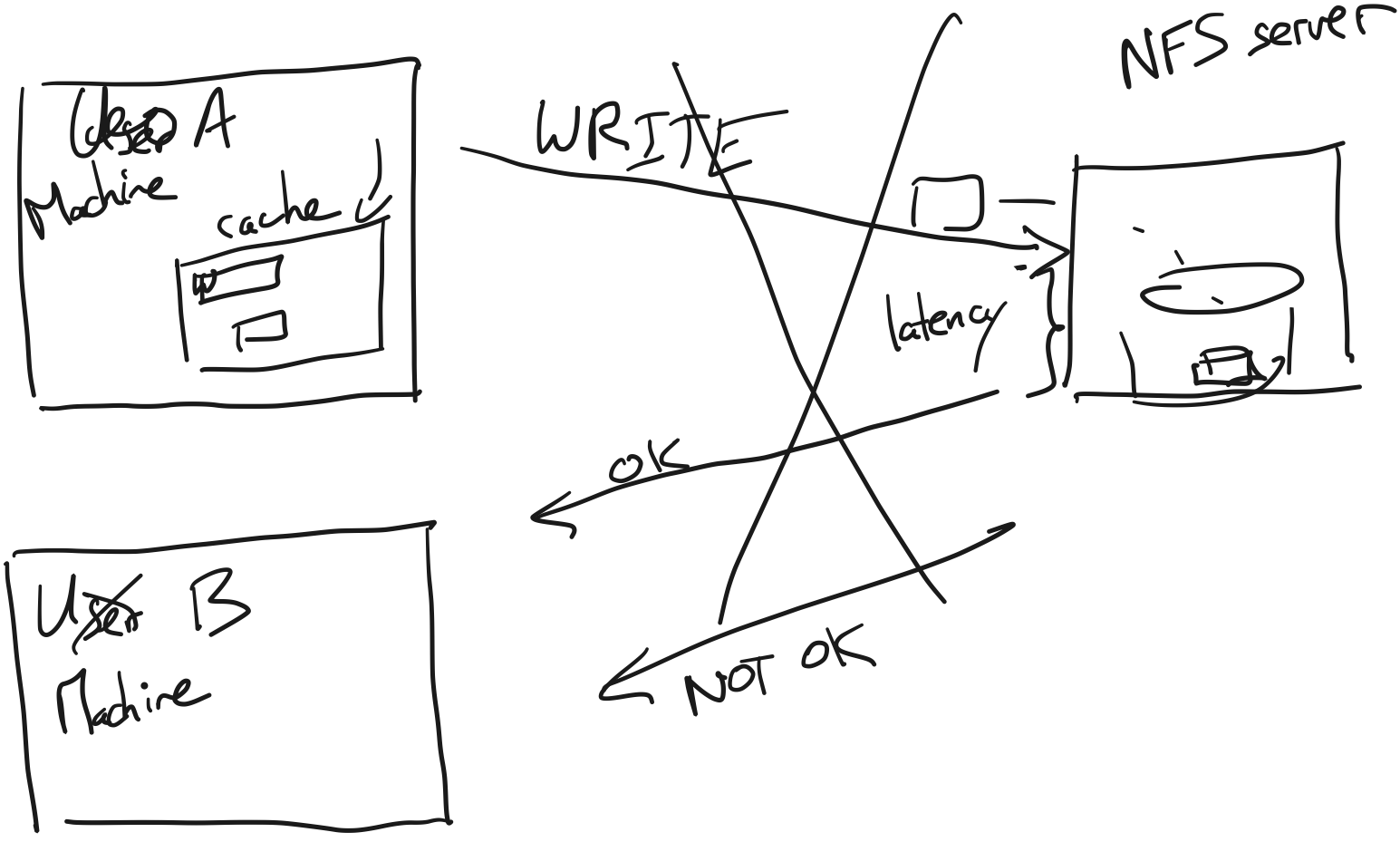
bootup, from power-on
- static linking + loading is a key tool
- bootstrap process
  - H/W copies firmware into read/write mem
  - firmware is mini OS
  - runs bootloader program, which ultimately begins kernel

kernel invokes init(1) / init(8)
init(1) invokes login(1)
login(1) lets you get a shell and begin executing programs

---

3. Your questions

NFS



close-to-open consistency

close():
generates a flush of the cache

```
close(fd);

if ((rc = close(fd)) < 0)
    handle_error();
```

---

mmap()

↳ manipulates page tables.

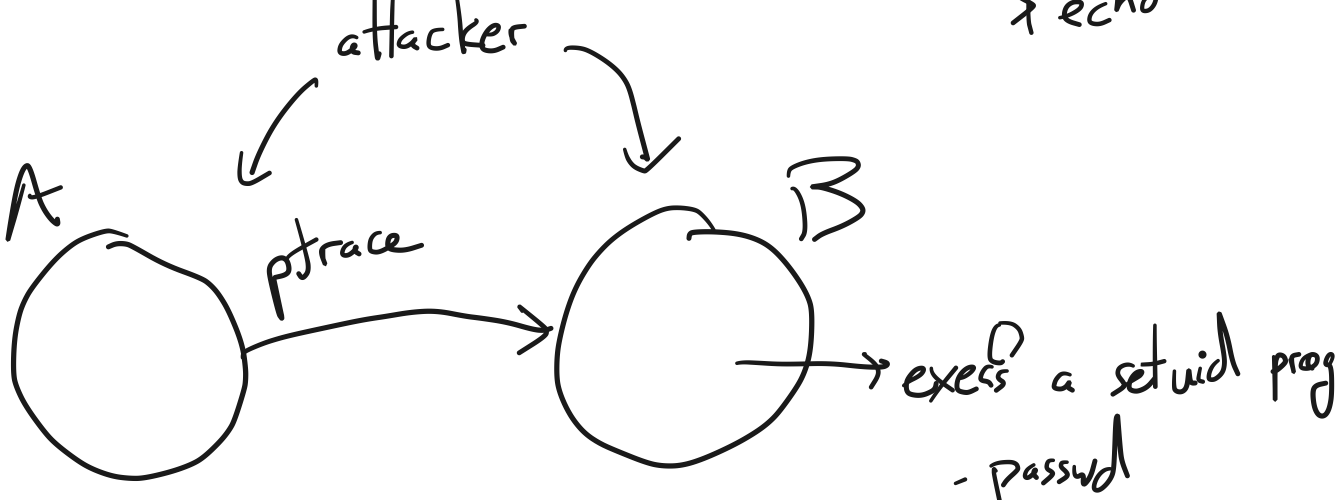
mprq

virtual_memory_map()

---

ptrace

$ gcc
$ echo



. su whatever

A: uses ptrace
to rewrite B's
memory to exec a shell,
and to connect the input/output to A.

---

user-level vs. kernel-level

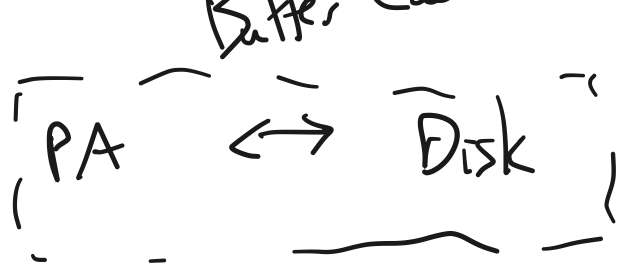
VM vs. file system

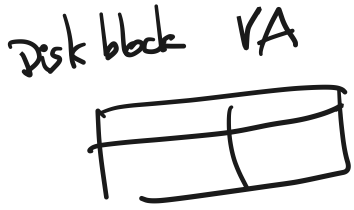
/

/swap

mmap(..., fd, ...)

VA → PA





Device driver

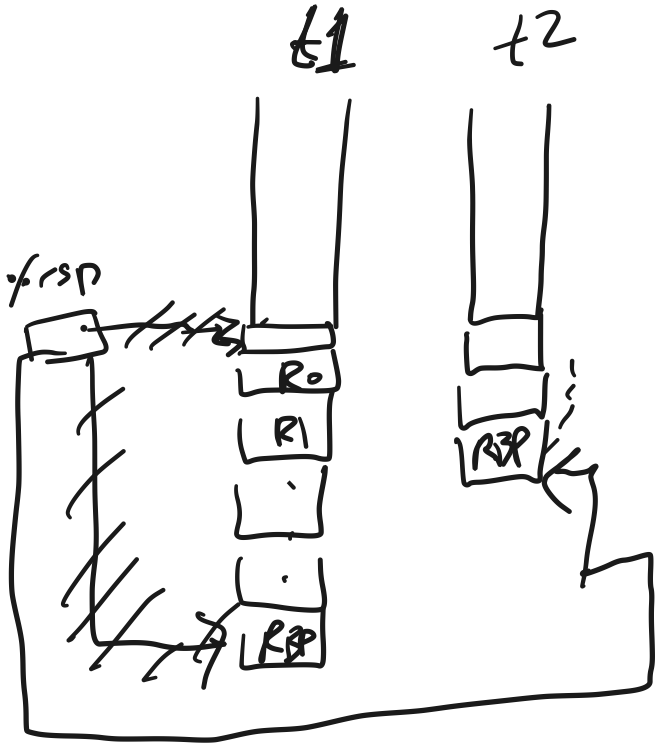
T/F? Buggy d.d. + non-buggy kernel $\overset{?}{\Rightarrow}$ crash

---

Context switch on user-level threading

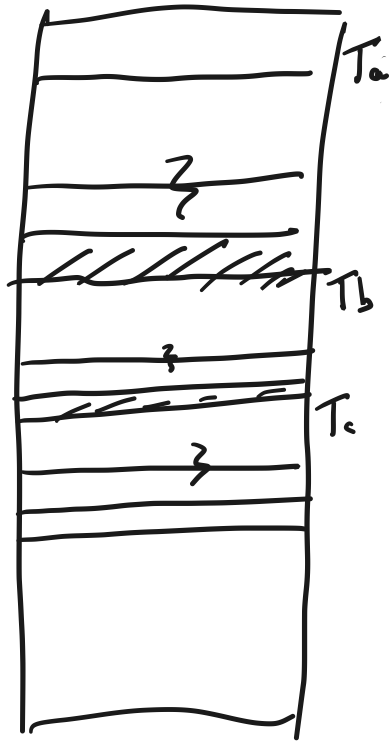
```
struct thread {
    int thr_id;
    uint64_t stack;
}
```

R0 -- R3
RBP
RSP




```
swtch(thread* t1, thread* t2)
{
    push_reg(R0);
    push_reg(R1);
    push_reg(R2);
    push_reg(R3);
    push_reg(RBP);
    t1->stack = read_reg(RSP);
    write_reg(RSP, t2->stack);
    pop_reg(RBP)
    pop_reg(R3);

    pop_reg(R0);
    return;
}
```



t1: Tc
t2: Ta

Favor the published sol'n

)

`|cr3(_ __)` only kernel can mod cr3

---

setuid binary:

effective UID of the created process (from the binary) is that of the UID that owns the file

real UID is that of the invoking user

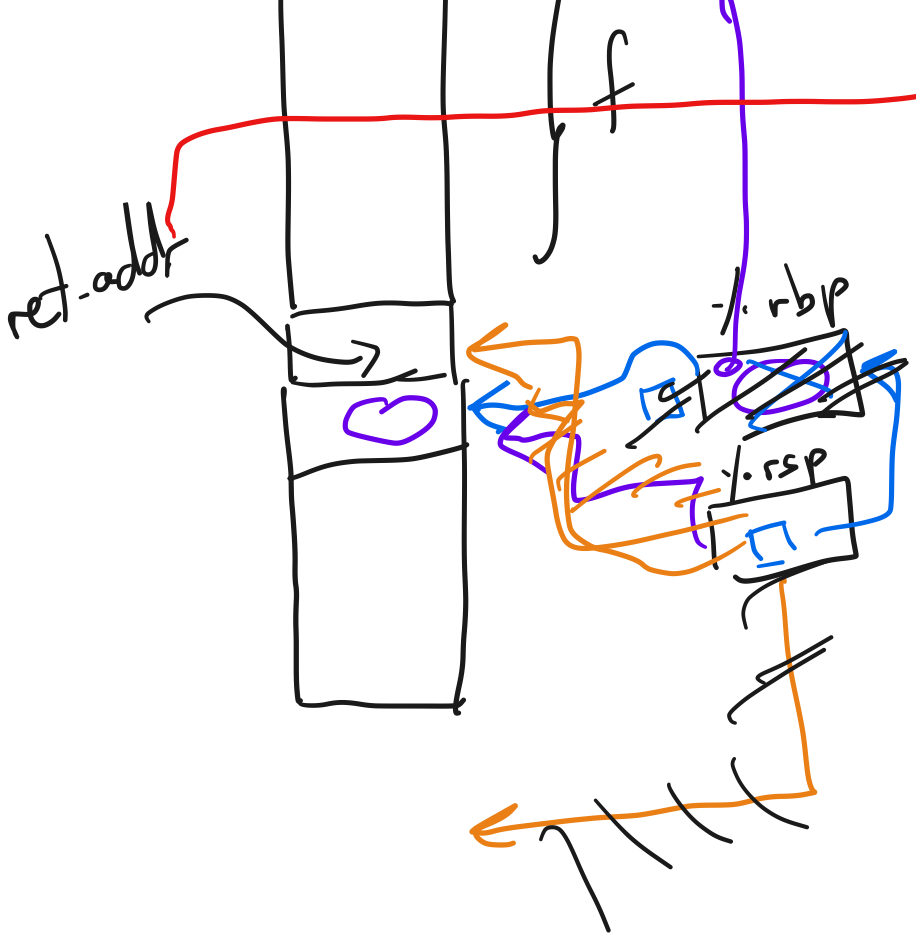
normally (not setuid binary)

effective UID = real UID: that of the invoking user

seteuid system call: drop privileges by setting effective user id

---

stack frames




```
g () {
  push %.rbp;
  mov %.rsp, %.rbp
  subq 128, %.rsp
  ...
  movq %.rbp, %.rsp
  popq %.rbp
  ret
}
```