# Limited Direct Execution

Idea: To run as fast as possible, user code runs directly on the CPU.

But...

Some instructions are too

- powerful
- alter machine state beyond one process

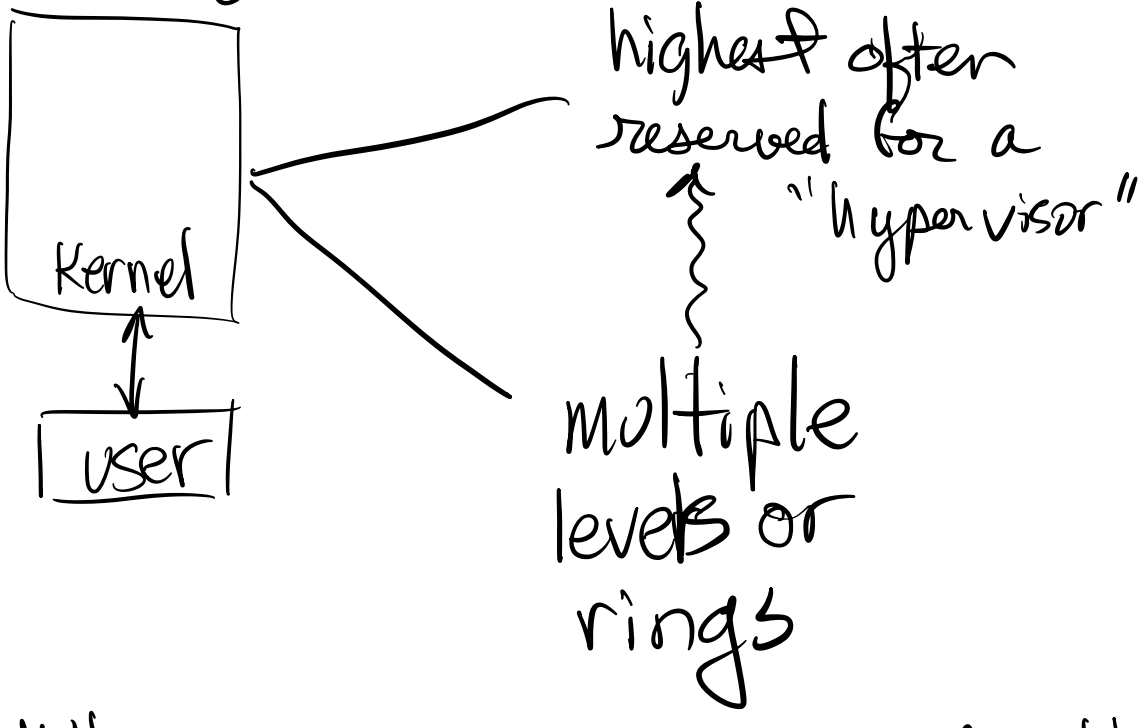
Therefor...

Some instructions, if attempted by user process, kill the process.

So how do these special instructions get executed?

AND... how to enter/exit the kernel state - for example - system calls.

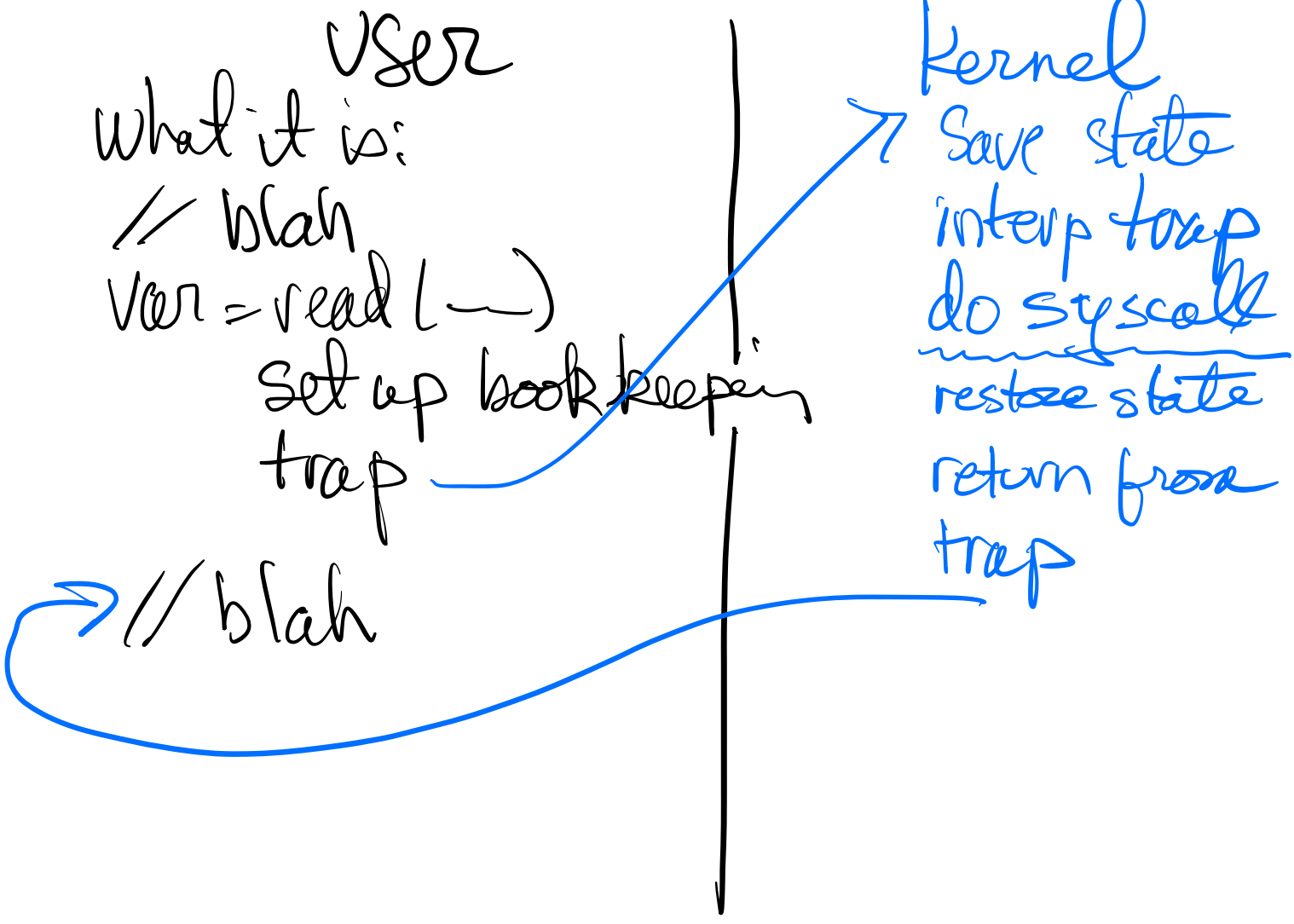
User processes cannot access kernel memory. Solution: processor states




When a user process needs the services of the kernel, it TRAPS to cause a state transition.

What it looks like:

```
// blah blah blah
   var = read(_____)
// blah blah blah
```




---

the open() system call

open(...) → put parameters in right place

put "I am a system call" in right place.

put "I am Open" in right place

trap.

ARM64

syscall number → X8

parameters → X0 → X7

trap → SVC Ø

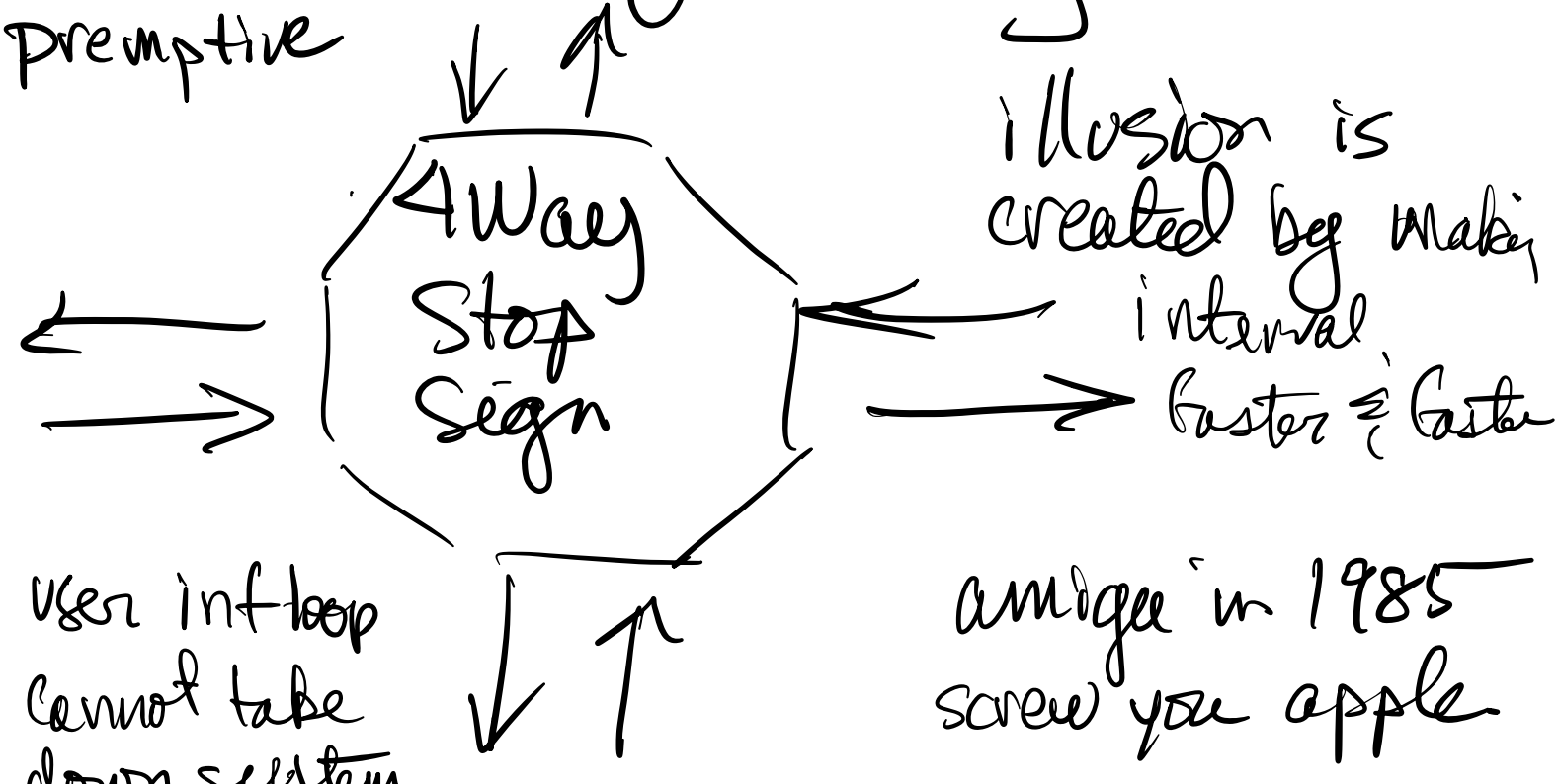
return value in X0

More details on RiscV lookat trampoline.S uservec

---

Methods of multitasking
— illusion of owning the CPU

preemptive



cooperative

yield() or sleep() etc
make a system call
in general.

Mac OS 1 → 9 till 2001

1: b 1b or in other words

labels: branch to label.

Any kind of infinite loop in user code causes tight lock up!!!

---

Context switch ~ ≤ 1 μs
equivalent to about 2000 instructions