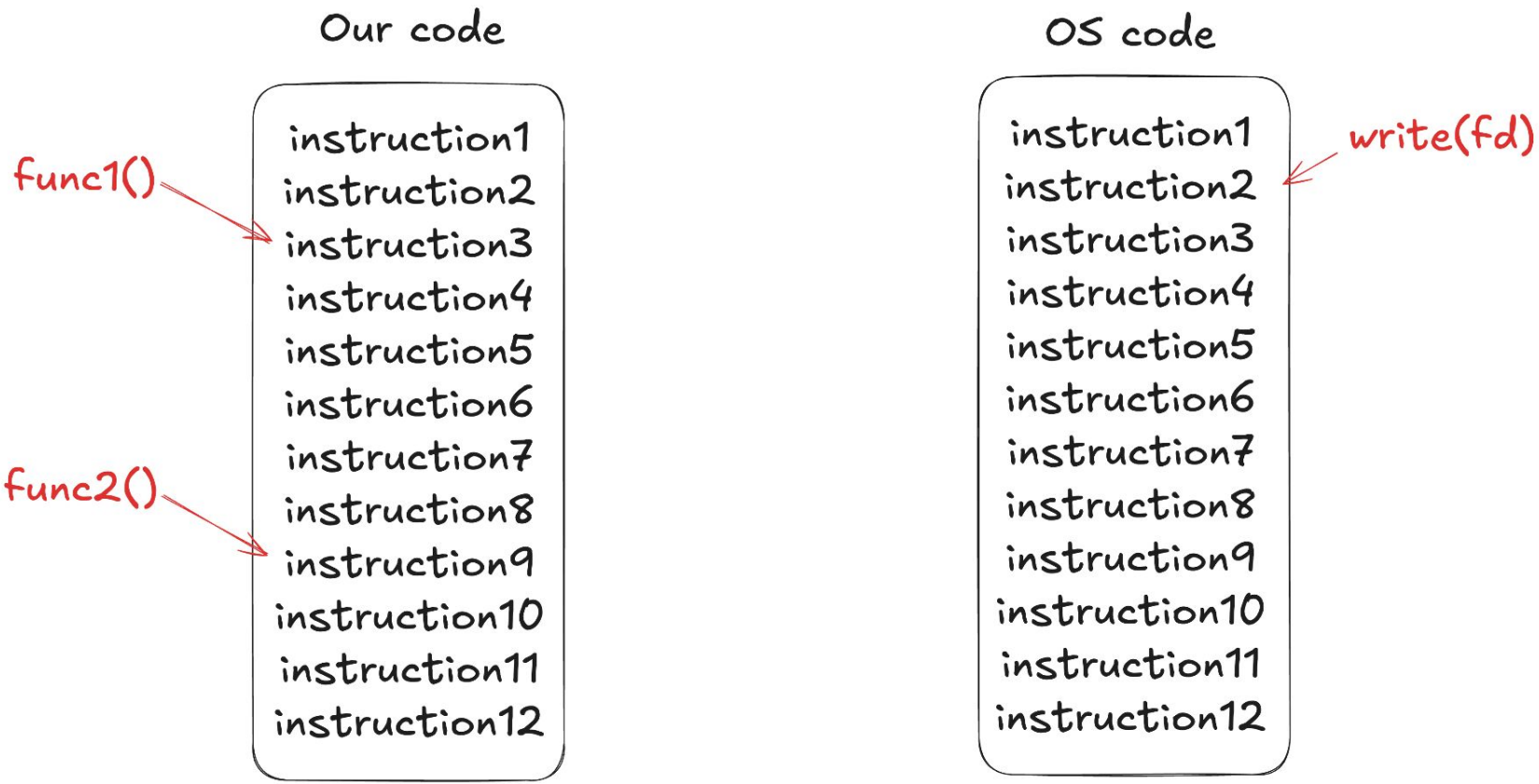
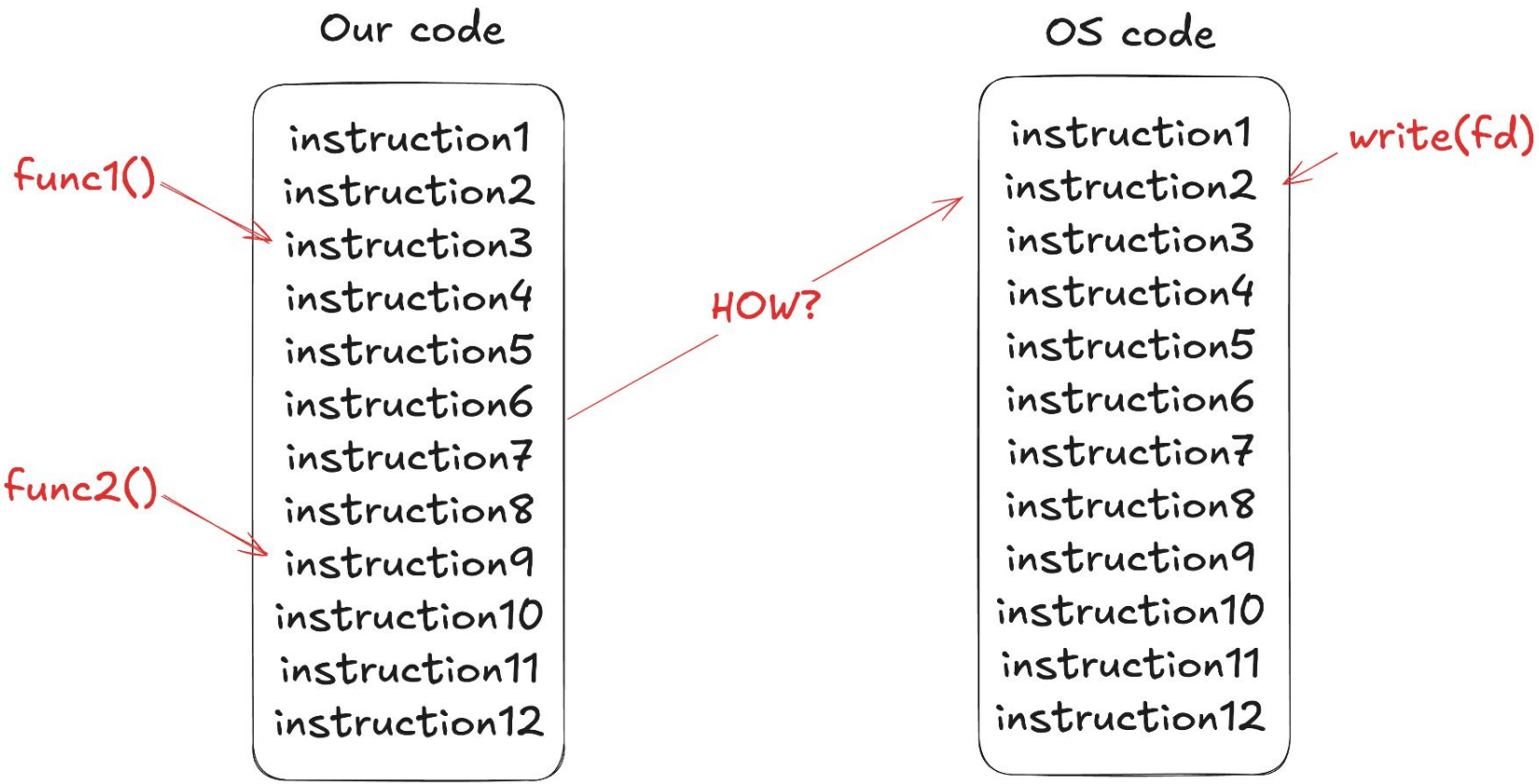
Processes, Threads, Thread Pools

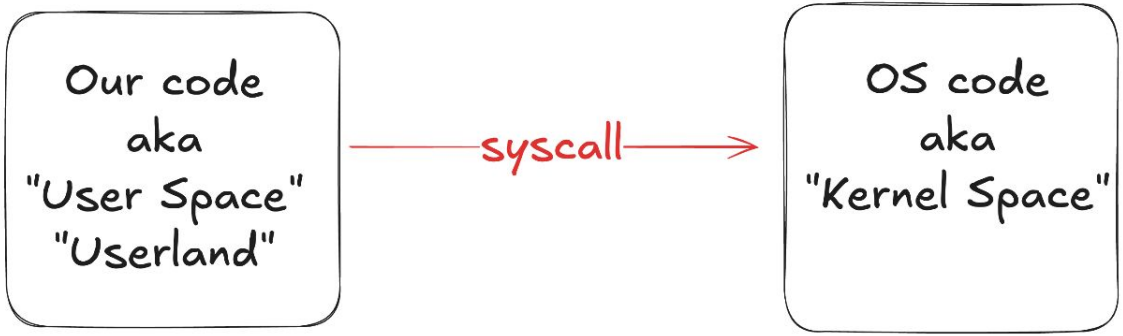
What's under the hood?

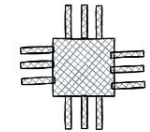
```
Thread t = new Thread(new MyRunnable());

t.start();
```


Our code
instruction1
instruction2
instruction3
instruction4
instruction5
instruction6
instruction7
instruction8
instruction9
instruction10
instruction11
instruction12
func1()
func2()
OS code
instruction1
instruction2
instruction3
instruction4
instruction5
instruction6
instruction7
instruction8
instruction9
instruction10
instruction11
instruction12
write(fd)


Our code
OS code
func1()
func2()
instruction1
instruction2
instruction3
instruction4
instruction5
instruction6
instruction7
instruction8
instruction9
instruction10
instruction11
instruction12
HOW?
instruction1
instruction2
instruction3
instruction4
instruction5
instruction6
instruction7
instruction8
instruction9
instruction10
instruction11
instruction12
write(fd)


Our code
aka
"User Space"
"Userland"
syscall
OS code
aka
"Kernel Space"



syscall

OS code

instruction1
instruction2
instruction3
instruction4
instruction5
instruction6
instruction7
instruction8
instruction9
instruction10
instruction11
instruction12

Userspace:

- put current "line number" into a CPU register
- put params into memory/CPU registers
- put operation number into a register
- `syscall` → (instruction at #10100101)

Linux:

- read operation number out of the register
- start the operation
- read params from memory or other registers
- write the result code into register
- get userspace "line number" and jump there

Process vs Thread

- Linux creates both similarly
- But Threads share memory with the parent
- Similar at OS level => similar capabilities (ps, top)

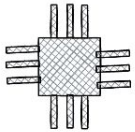
Troubleshooting threads in Linux

- If Alpine OS with limited BusyBox:
  - `apk update && apk add procps htop`
- `jcmd <pid> Thread.print`
- `ps -eTf`
- `top -p <pid>`, then press `H` (or `htop -p`)

Scheduling

How do we tell CPU to switch to another thread?

- During syscall
- Timer interrupts



Interrupts

1

2

...

[timer]

OS code

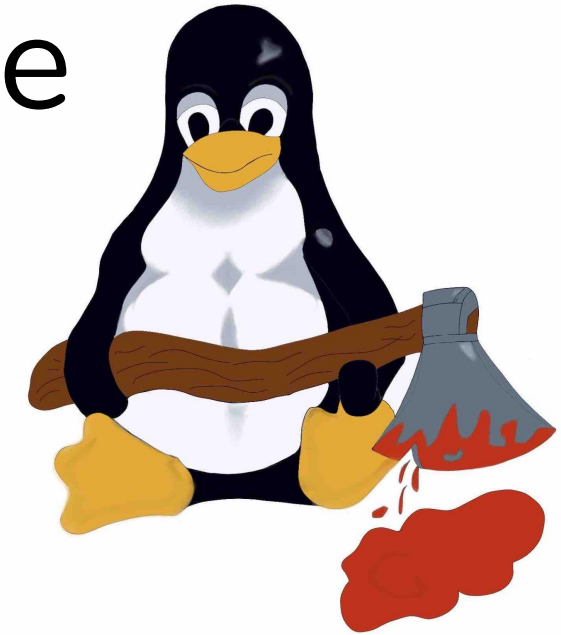
instruction1
instruction2
instruction3
instruction4
instruction5
instruction6
instruction7
instruction8
instruction9
instruction10
instruction11
instruction12

Stopping a thread

What happens when we `kill`?

- OS registers the request (Signal)
- And processes it during next scheduling

Now that we’re in Kernel, it’s kill time



Types of kill

- SIGKILL (`kill -9`) asks OS to interrupt threads in user-space and never start them again
- It's violent
- SIGTERM (`kill` or `kill -15`) asks OS to call a handler that the process itself registered

How threads are killed?

- We don’t want violence (like SIGKILL)
- So we ask nicely (analog of SIGTERM)

Inside Runnable