Lec 3

Interrupts
+
System Calls

Warm up:

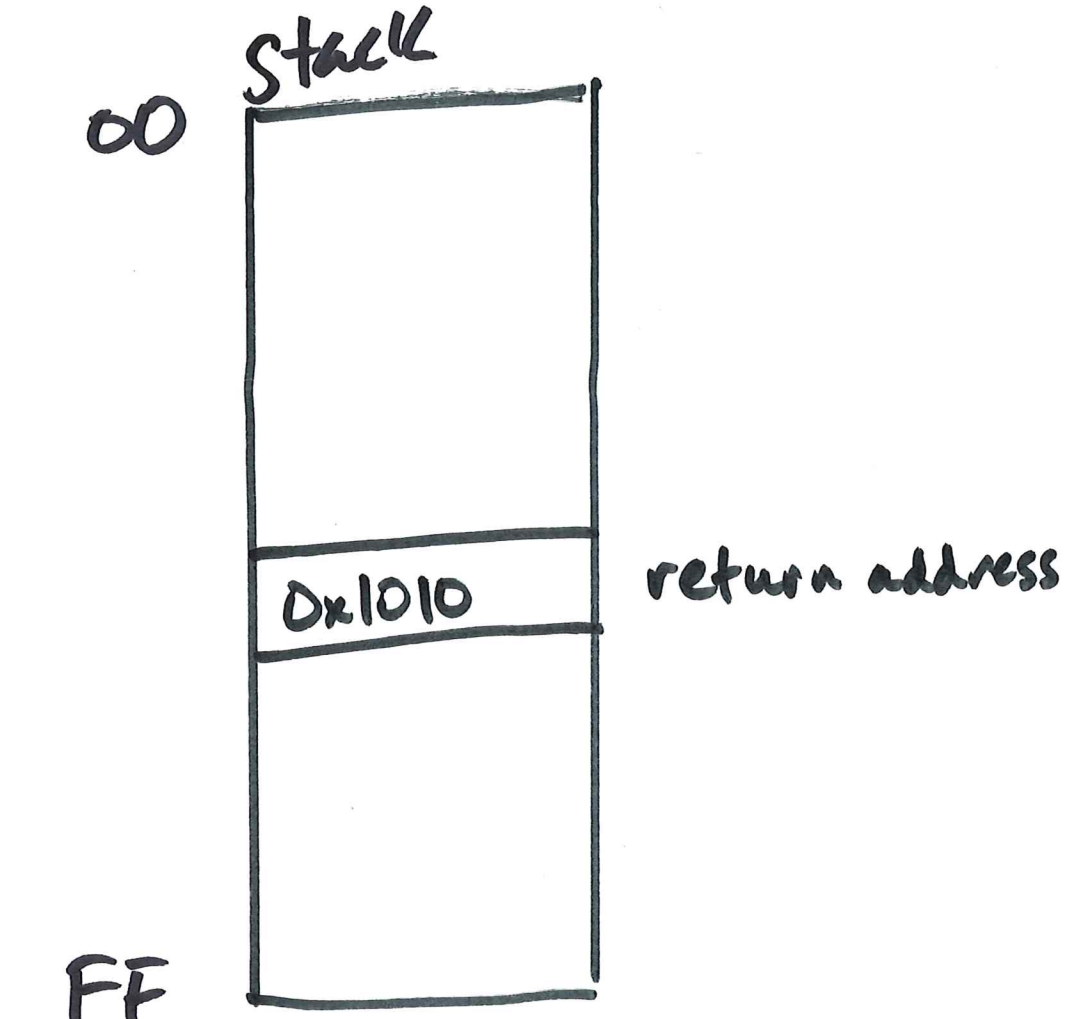
How do function calls work in assembly?

- args?
- return value?
- return address?
- registers?
- how to make the call? return?

Brief reminder about function calls

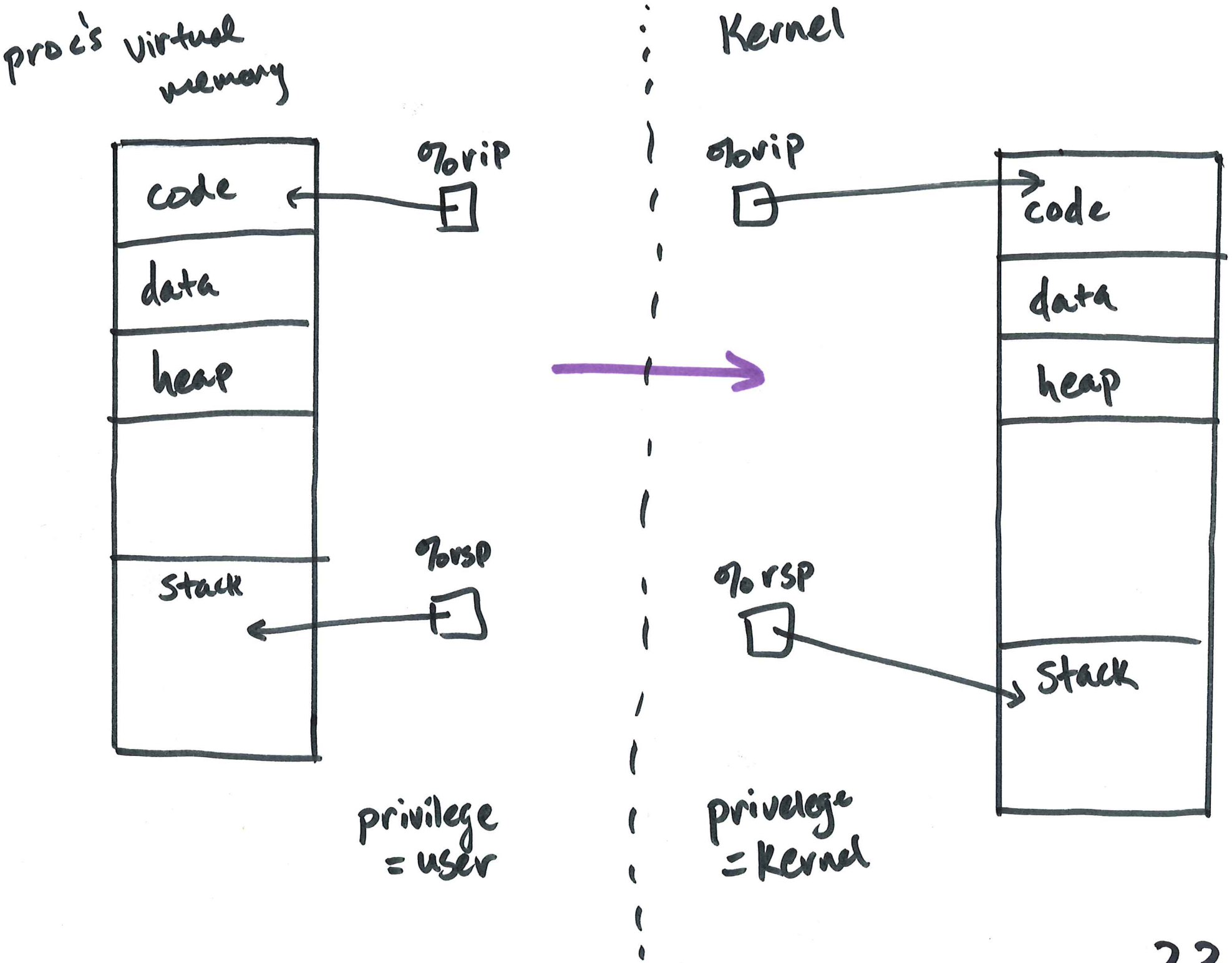
```
main:
        :
0x1008  callq   foo
0x1010  :

foo:
        :
        retq
```

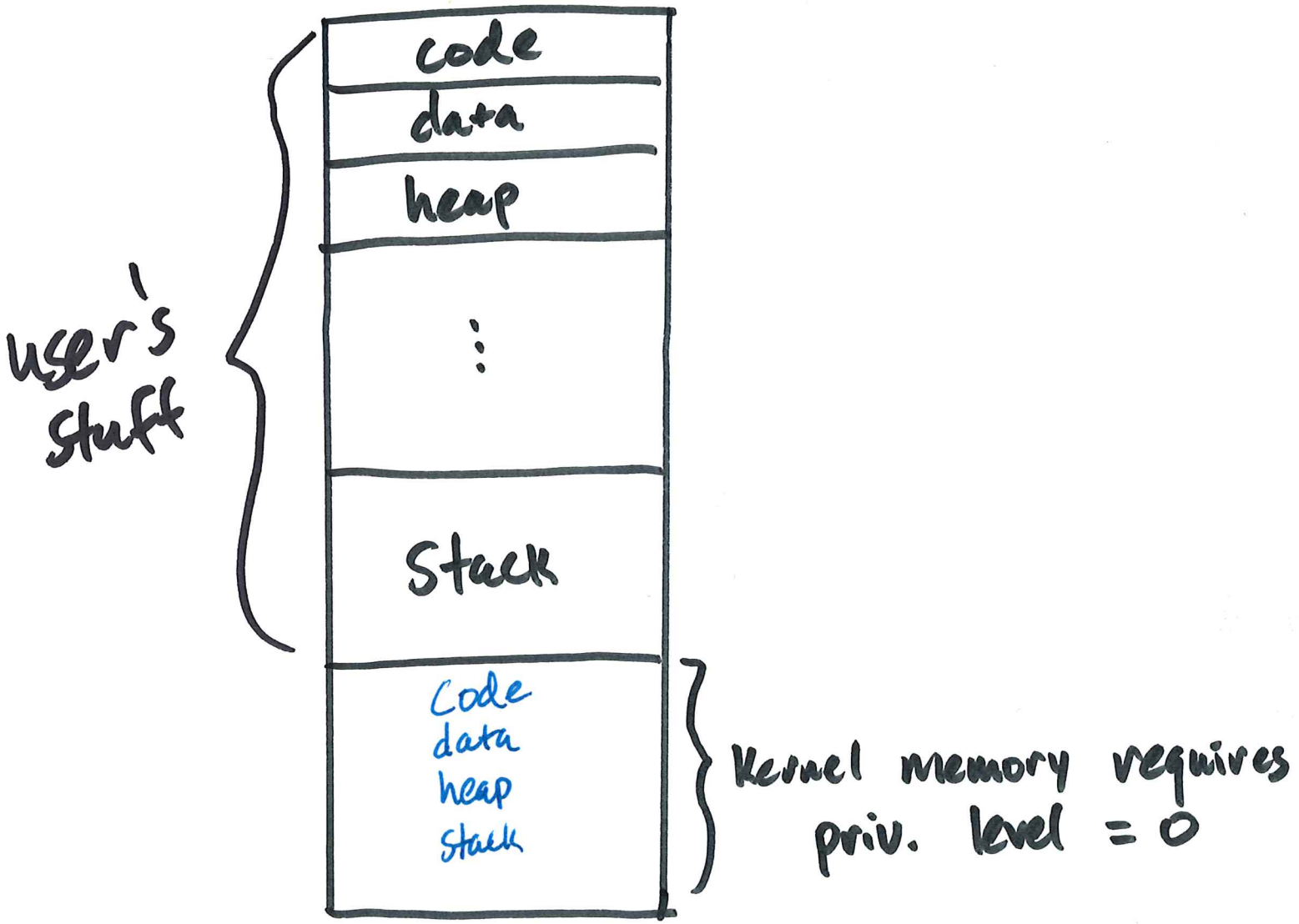


x86-64 Calling convention:

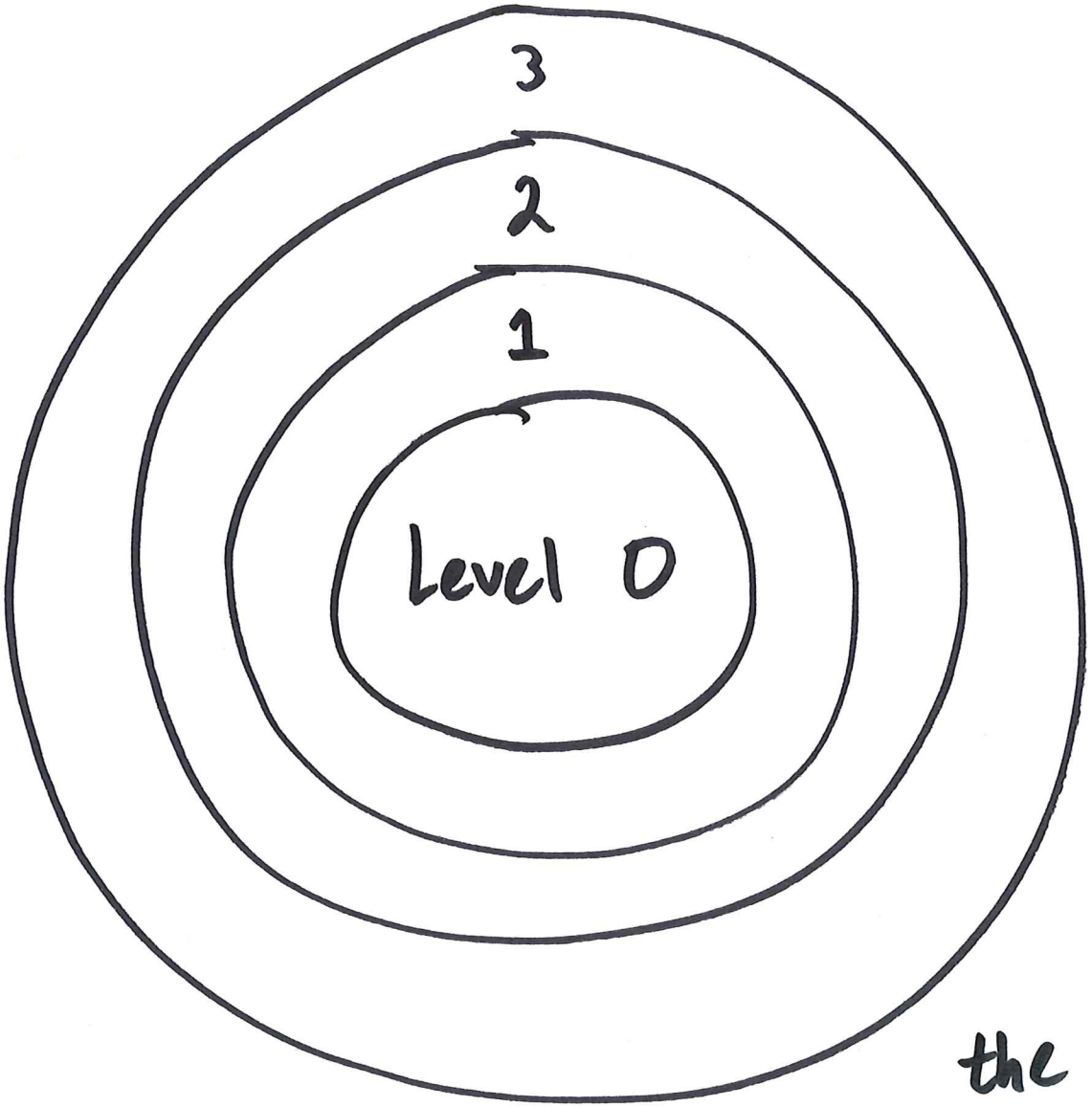
callee-saved: rbx, rsp, rbp, r12, r13, r14, r15


proc's virtual memory
Kernel
code
data
heap
stack
%rip
%rsp
%rip
%rsp
code
data
heap
stack
privilege = user
privilege = Kernel

Proc's virtual address space
also maps kernel data, etc.



x86 privilege levels

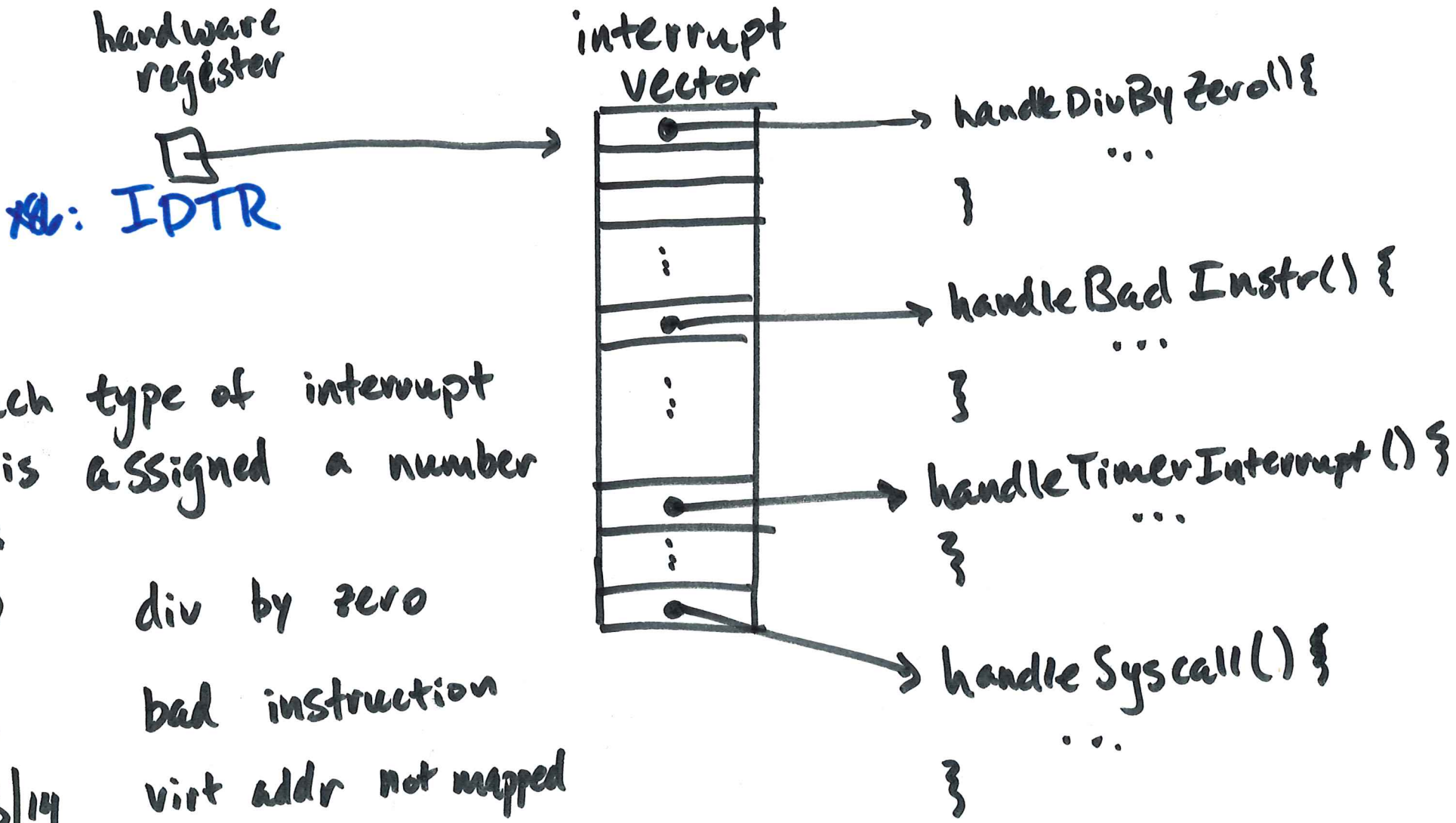


"ring 0"
= kernel mode

"ring 3"
= user mode

the current privelege level (CPL) is stored in low 2 bits of CS segment reg

Hardware Interrupts



- each type of interrupt is assigned a number

x86:

0	div by zero
6	bad instruction
13|14	virt addr not mapped

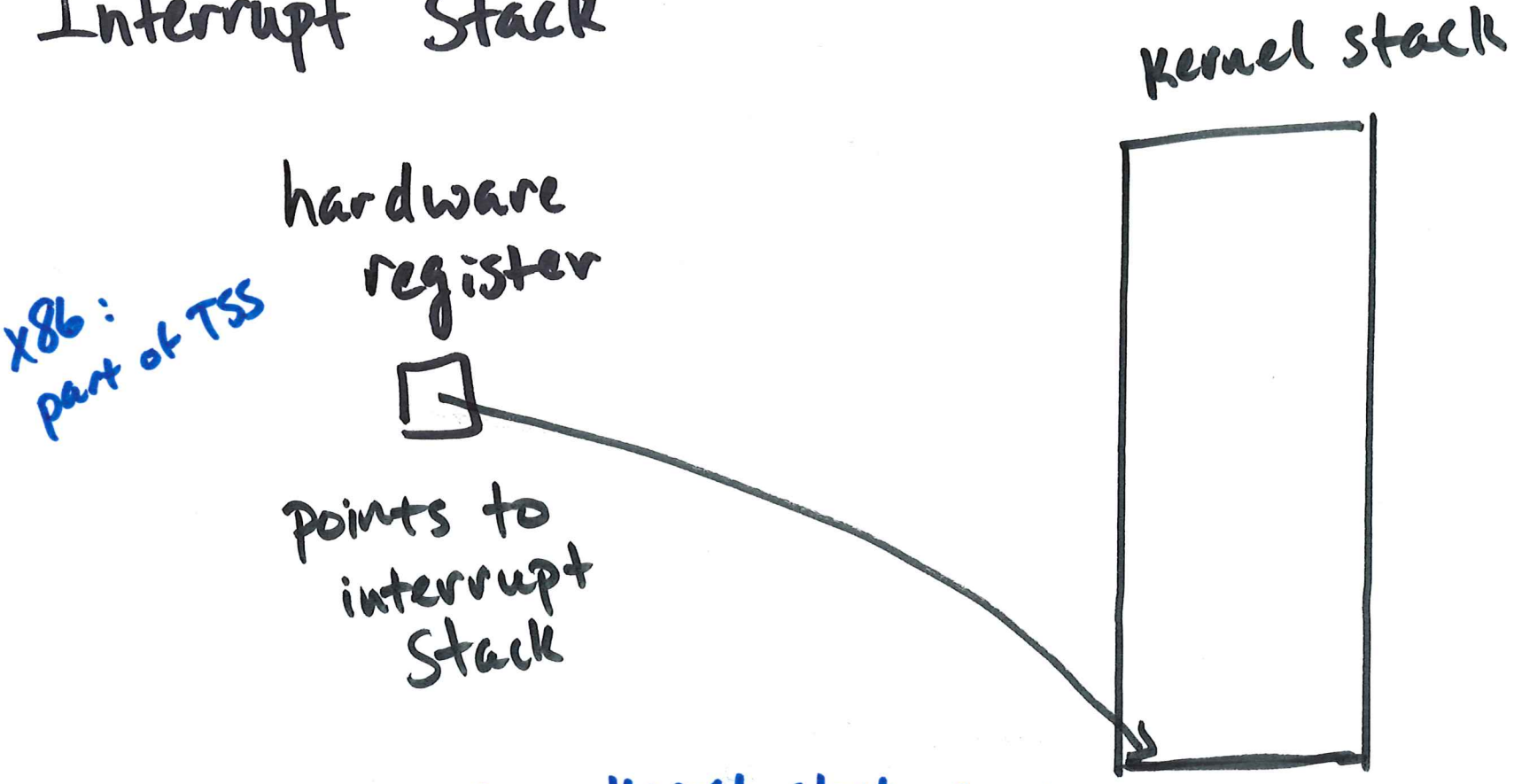
32-255 available

XK: syscall = 64

What must be true about mode transfer?

- Kernel must control entry points
  - malicious app must not jump to arbitrary Kernel code
- program counter, stack pointer, priv. level all change
  - what would happen if we used the proc's stack?
    - could be invalid
    - could point to Kernel memory
- old values of PC, SP, CPL must be saved
  - transparently restart process
  - mode transfers must happen at Known place in instruction stream

Interrupt Stack



- in user mode, Kernel stack empty
- on interrupt, set %rsp to this register
- convenient to allocate separate Kernel stack for each process

Hardware support for mode transfer (x86-64)

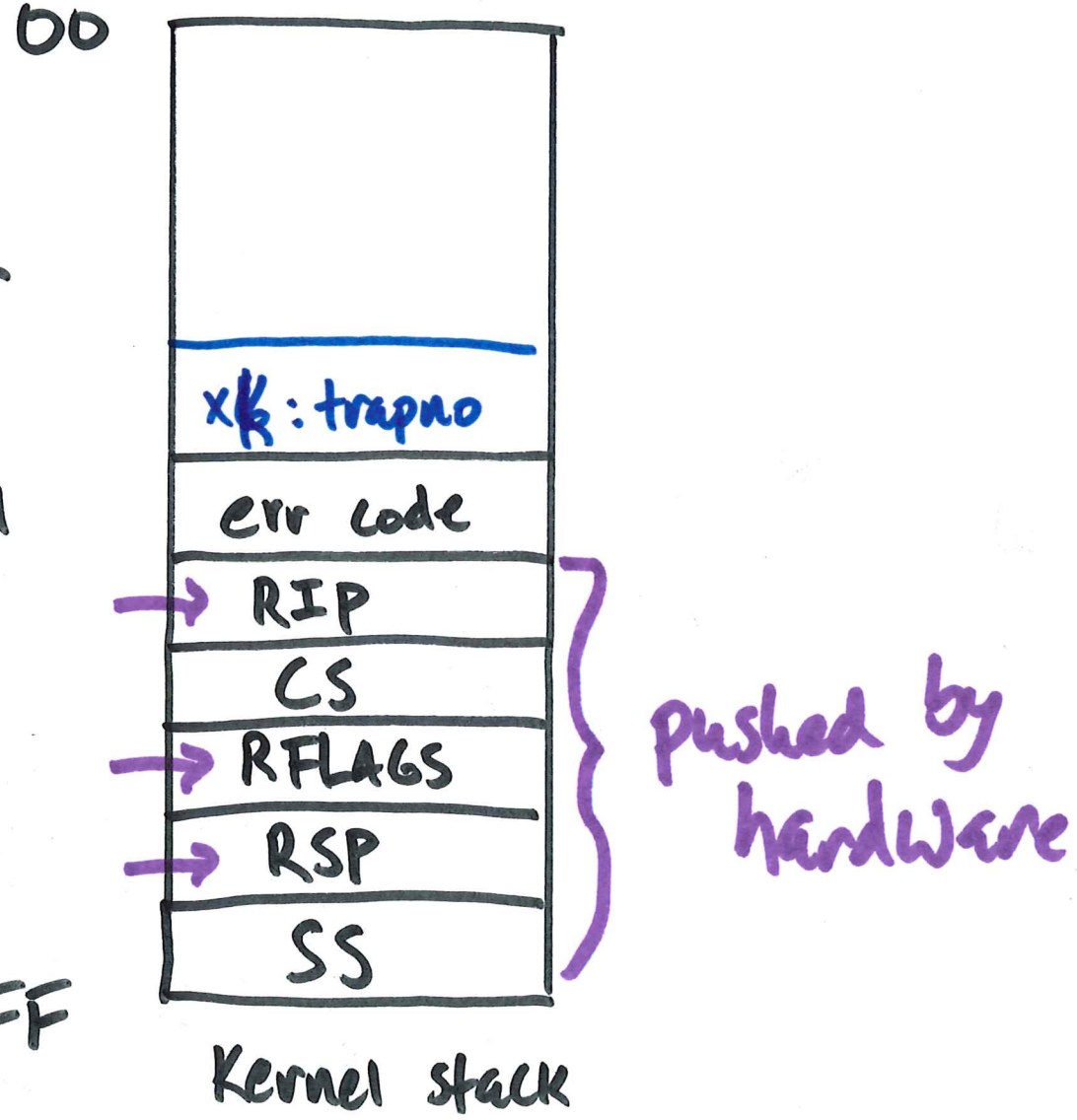
- before calling handler set up Kernel stack like this

- handler should save all other registers!

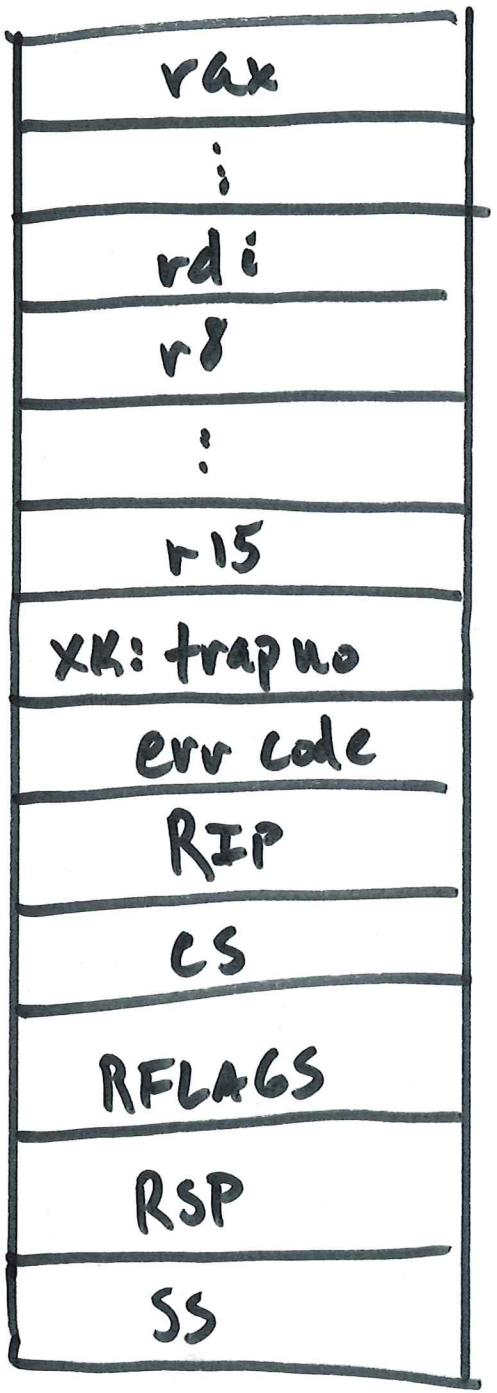
- to restore:

→ iretq

(after restoring all other regs + Popping trapno + err code) (see inc/trap.h)



After saving all registers, stack looks like this



Only <u>now</u> is it safe to call C functions in kernel. (why?)

- this data structure is called a <u>trap frame</u>

struct trap_frame

Finishing interrupts

- timer goes off
- hardware calls handler
- handler saves all regs + calls C
- C decides what to do
  - run other process?
  - go back to same process?
- C returns
- handler restores regs + executes iretq

System Calls

- application wants to request kernel service
  - read a file, for example
- can do this via "software interrupts"
  - "int" instruction on x86
  - takes interrupt num. as arg
  - invokes that handler
- assign an interrupt number to system calls
  - xk: 64

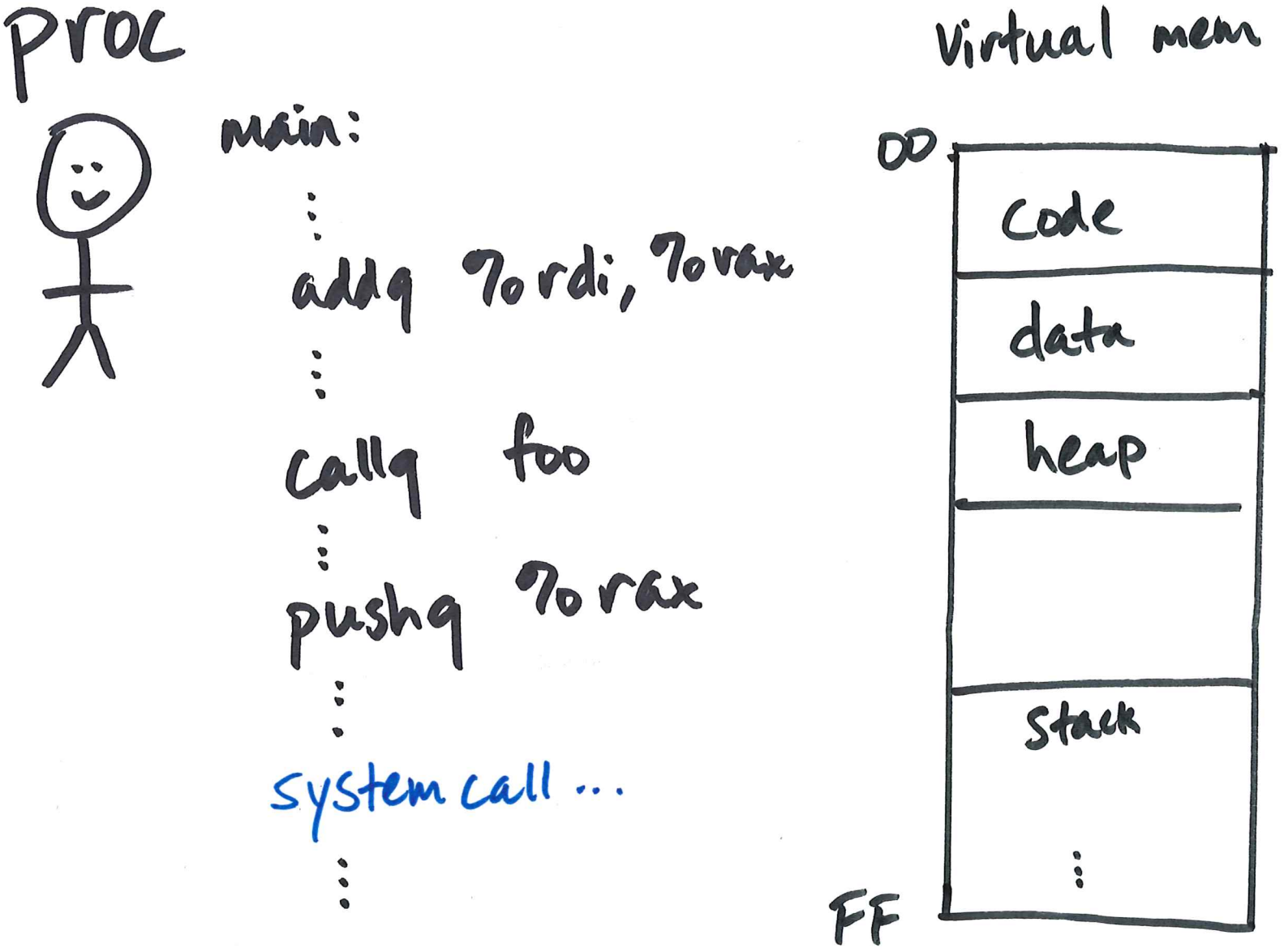
```
main() {
    file_open(arg1, arg2);
}
```



```
file_open(arg1, arg2) {
    movq $SYS_OPEN, %rax
    int $64
    retq
}
```

```
file_open_handler() {
    // checks...
    // copy args
    file_open(arg1, arg2)
    // copy ret val
    return;
}
```

```
file_open(arg1, arg2) {
    // do operation
}
```


proc
main:
addq %rdi, %rax
callq foo
pushq %rax
system call ...
Virtual mem
00
code
data
heap
stack
FF

3. extra