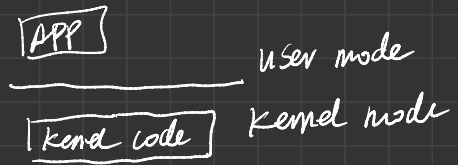
10/3/22 DME



Types of Transfers

① Exception
- → Synchronous
- → hw switch mode (kernel) save current state exception handler]

② System Calls
- → read, write
- → voluntary transfer to kernel (syscall, sysret)
- → resumes at PC+1
- → synchronous

③ interrupts
- → I/O completion, mouse or keyboard, network packet
- → asynchronous
- → timer interrupt

Kernel to User
- → upon syscall completion
- → new process starts
- → return from interrupts / exceptions.

Similarities btwn the 3 mode transfers

1. mode switch
2. save current state
3. run some kernel handler

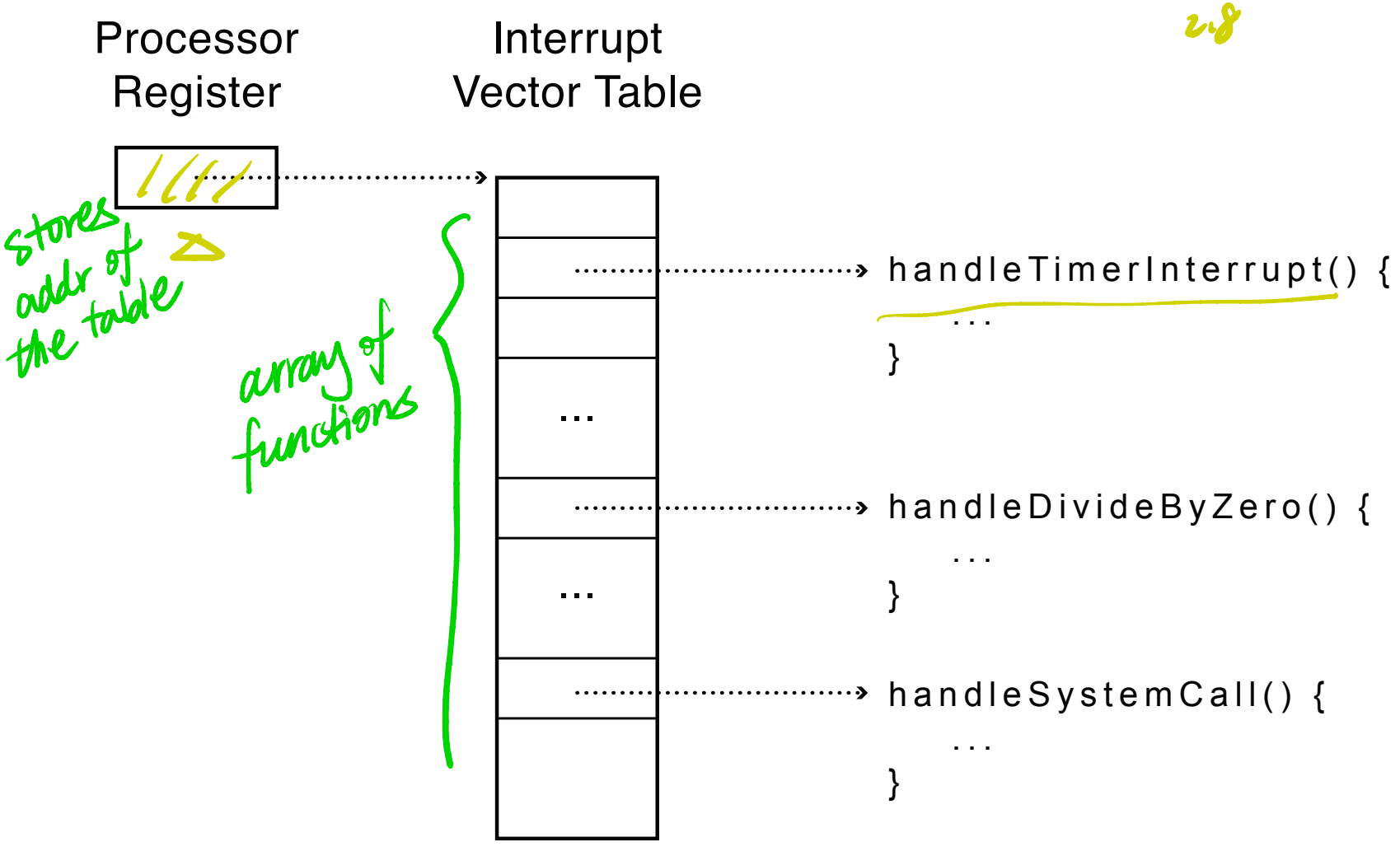
OSDev Wiki

How kernel supports this.

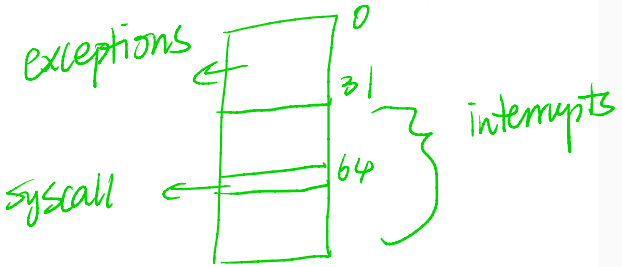
→ Interrupt Vector Table (stores into for all 3 mode transfers handler)

- what to invoke
- triggers the handler + saves the current state.

→ Isolate with kernel stack.


Processor Register
Interrupt Vector Table
stores addr of the table
array of functions
handleTimerInterrupt() {
...
}
handleDivideByZero() {
...
}
handleSystemCall() {
...
}

```c
// Interrupt descriptor table (shared by all CPUs).
struct gate_desc idt[256];
extern void *vectors[]; // in vectors.S: array of 256 entry pointers
struct spinlock tickslock;
uint ticks;

int num_page_faults = 0;

void tvinit(void) {
  int i;

  for (i = 0; i < 256; i++)
    set_gate_desc(&idt[i], 0, SEG_KCODE << 3, vectors[i], KERNEL_PL);
  set_gate_desc(&idt[TRAP_SYSCALL], 1, SEG_KCODE << 3, vectors[TRAP_SYSCALL],
                USER_PL);

  initlock(&tickslock, "time");
}

void idtinit(void) { lidt((void *)idt, sizeof(idt)); }
```



kernel handler

trap.c

vectors.S 20.31 KiB

```
.globl alltraps
.globl vector0
vector0:
  push $0
  push $0
  jmp alltraps
.globl vector1
vector1:
  push $0
  push $1
  jmp alltraps
.globl vector2
vector2:
  push $0
  push $2
  jmp alltraps
.globl vector3
vector3:
  push $0
  push $3
  jmp alltraps
.globl vector4
vector4:
  push $0
  push $4
  jmp alltraps
.globl vector5
vector5:
  push $0
  push $5
```

trap #

trapasm.S 447 bytes Ope

```
.globl alltraps
alltraps:
  push %r15
  push %r14
  push %r13
  push %r12
  push %r11
  push %r10
  push %r9
  push %r8
  push %rdi
  push %rsi
  push %rbp
  push %rdx
  push %rcx
  push %rbx
  push %rax

  mov %rsp, %rdi
  call trap

.globl trapret
trapret:
  pop %rax
  pop %rbx
  pop %rcx
  pop %rdx
  pop %rbp
  pop %rsi
  pop %rdi
  pop %r8
  pop %r9
  pop %r10
  pop %r11
  pop %r12
  pop %r13
  pop %r14
  pop %r15
  add $16, %rsp
  iretq
```

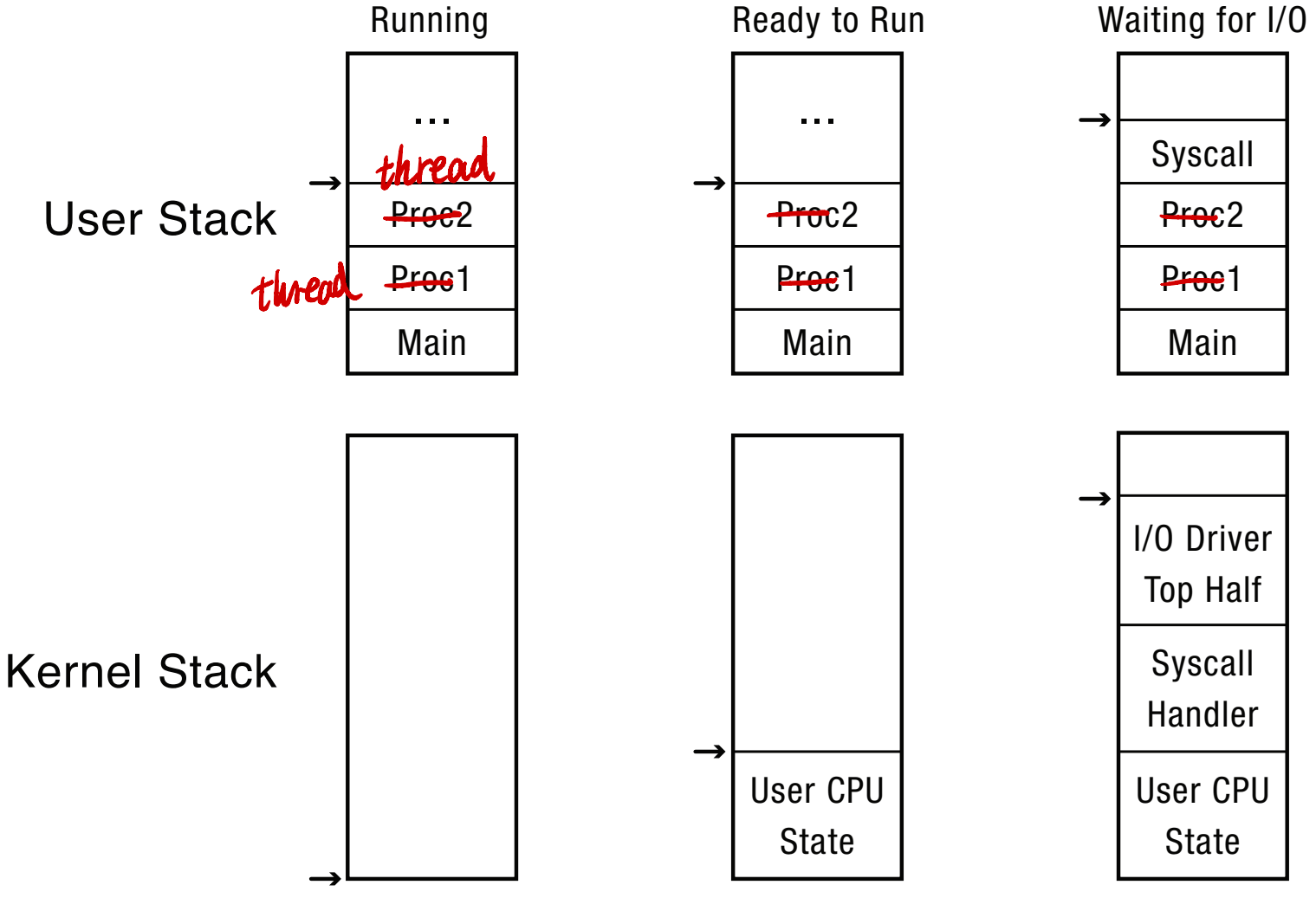
save state

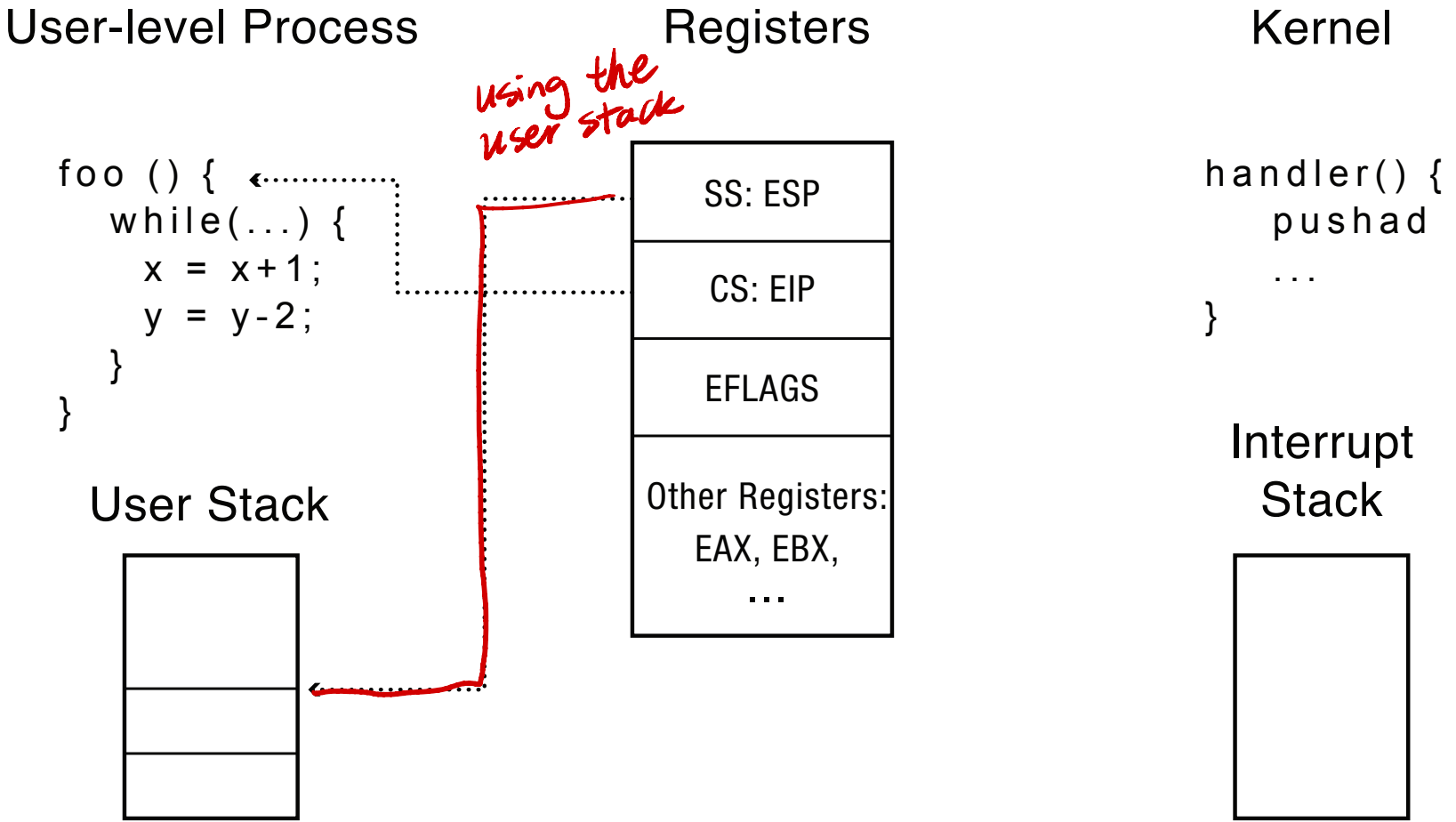
```
void trap(struct trap_frame *tf) {
  uint64_t addr;

  if (tf->trapno == TRAP_SYSCALL) {
    if (myproc()->killed)
      exit();
    myproc()->tf = tf;
    syscall();
    if (myproc()->killed)
      exit();
    return;
  }

  switch (tf->trapno) {
  case TRAP_IRQ0 + IRQ_TIMER:
    if (cpunum() == 0) {
      acquire(&tickslock);
      ticks++;
      wakeup(&ticks);
      release(&tickslock);
    }
    lapiceoi();
    break;
  case TRAP_IRQ0 + IRQ_IDE:
    ideintr();
    lapiceoi();
    break;
  case TRAP_IRQ0 + IRQ_IDE + 1:
    // Bochs generates spurious IDE1 interrupts.
    break;
  case TRAP_IRQ0 + IRQ_KBD:
    kbdintr();
    lapiceoi();
    break;
  case TRAP_IRQ0 + IRQ_COM1:
    uartintr();
    lapiceoi();
    break;
  case TRAP_IRQ0 + 7:
```

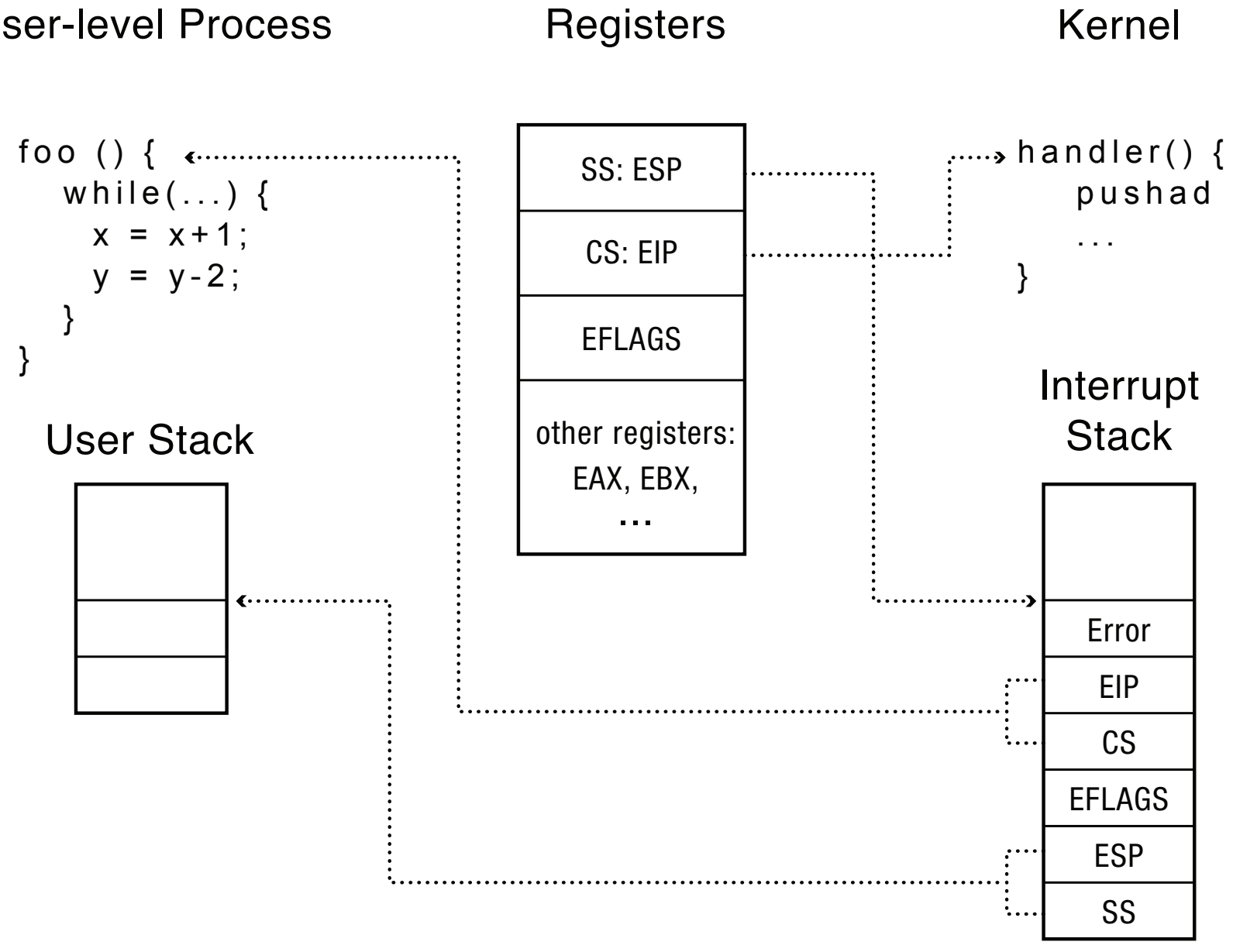
stack

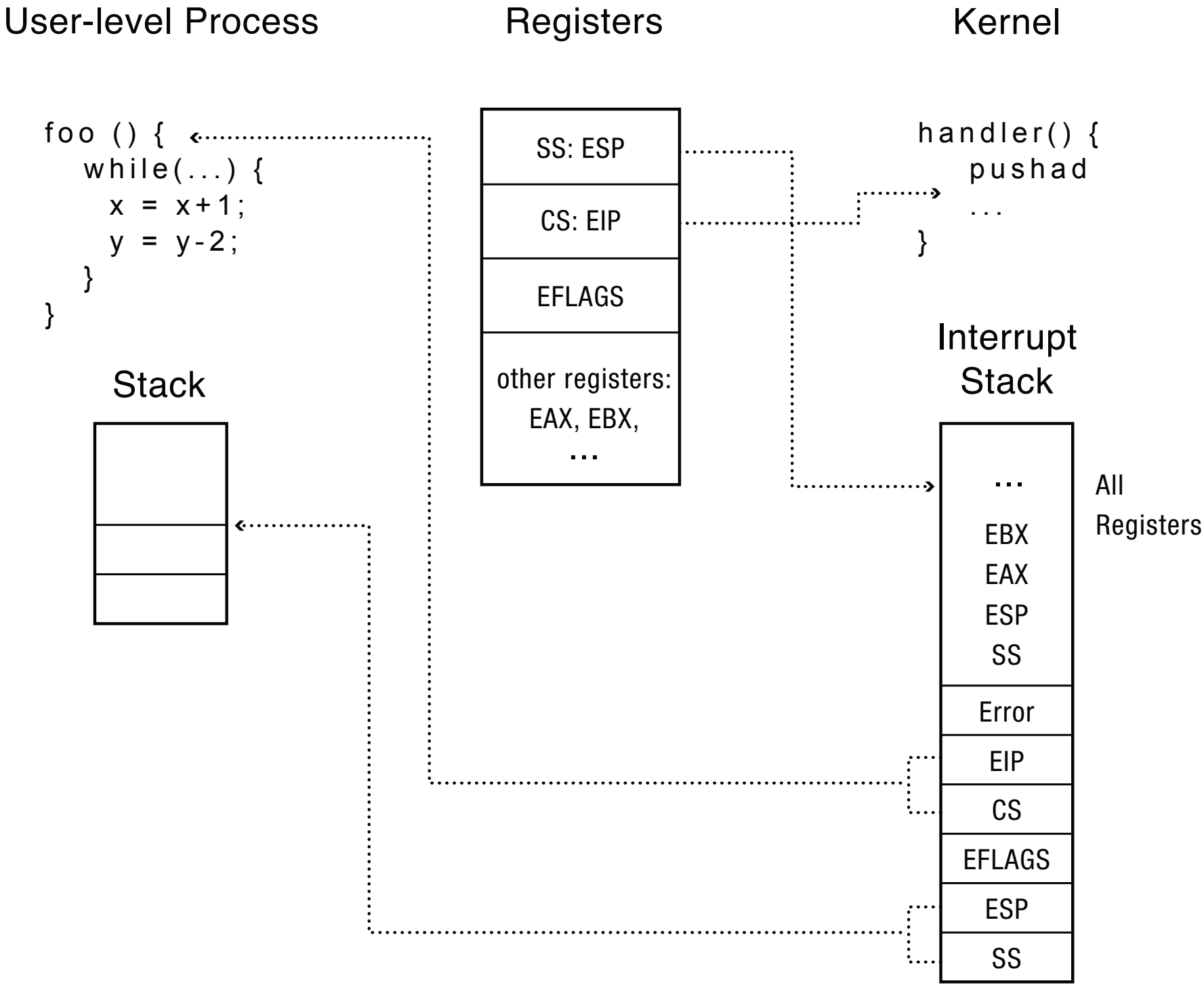
kernel mode

```
struct trap_frame {
  uint64_t rax; // rax
  uint64_t rbx;
  uint64_t rcx;
  uint64_t rdx;
  uint64_t rbp;
  uint64_t rsi;
  uint64_t rdi;
  uint64_t r8;
  uint64_t r9;
  uint64_t r10;
  uint64_t r11;
  uint64_t r12;
  uint64_t r13;
  uint64_t r14;
  uint64_t r15;
  uint64_t trapno;
  /* error code, pushed by hardware or 0 b
  uint64_t err;
  uint64_t rip;
  uint64_t cs;
  uint64_t rflags;
  /* ss:rsp is always pushed in long mode
  uint64_t rsp;
  uint64_t ss;
} __packed;
```


Running
Ready to Run
Waiting for I/O
User Stack
...
thread
Proc2
thread
Proc1
Main
...
Proc2
Proc1
Main
Syscall
Proc2
Proc1
Main
Kernel Stack
User CPU State
I/O Driver Top Half
Syscall Handler
User CPU State


User-level Process
Registers
Kernel
using the user stack
foo () {
while(...) {
x = x+1;
y = y-2;
}
}
SS: ESP
CS: EIP
EFLAGS
Other Registers:
EAX, EBX,
...
handler() {
pushad
...
}
User Stack
Interrupt Stack


User-level Process
Registers
Kernel
foo () {
while(...) {
x = x+1;
y = y-2;
}
}
SS: ESP
CS: EIP
EFLAGS
other registers:
EAX, EBX,
...
handler() {
pushad
...
}
User Stack
Interrupt
Stack
Error
EIP
CS
EFLAGS
ESP
SS


User-level Process
Registers
Kernel
foo () {
while(...) {
x = x+1;
y = y-2;
}
}
SS: ESP
CS: EIP
EFLAGS
other registers:
EAX, EBX,
...
handler() {
pushad
...
}
Stack
Interrupt
Stack
...
EBX
EAX
ESP
SS
All
Registers
Error
EIP
CS
EFLAGS
ESP
SS