Week 5.b
CS6640
02/05 2026
https://naizhengtan.github.io/26spring/

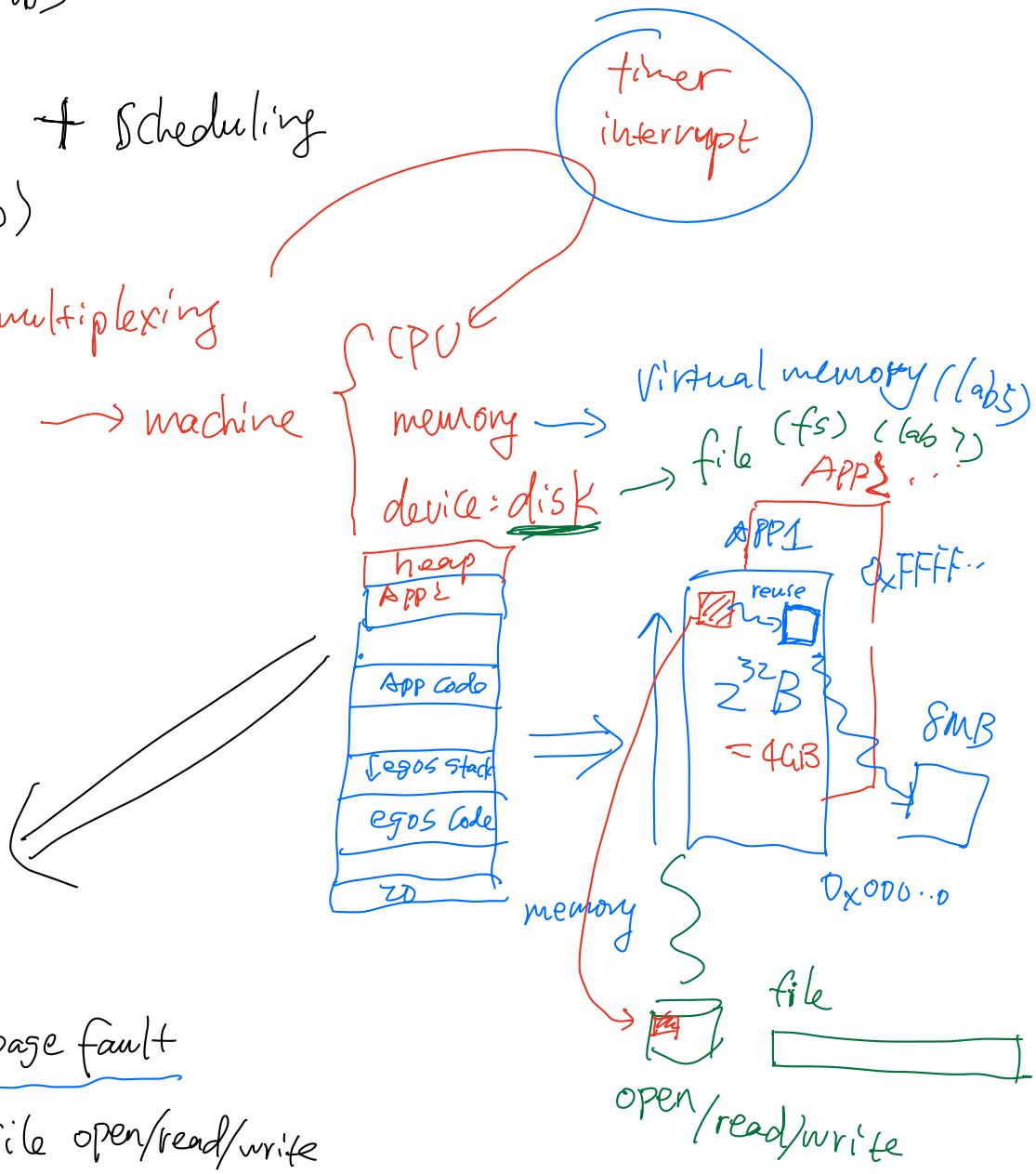
- [ ] 1. kernel ~= three handlers
- [ ] 2. egos syscall implementation
- [ ] 3. egos exception handling

Q: What are the two primary functions of an operating system?

1. managing res
2. providing abs

Last time: process + scheduling (PCB)

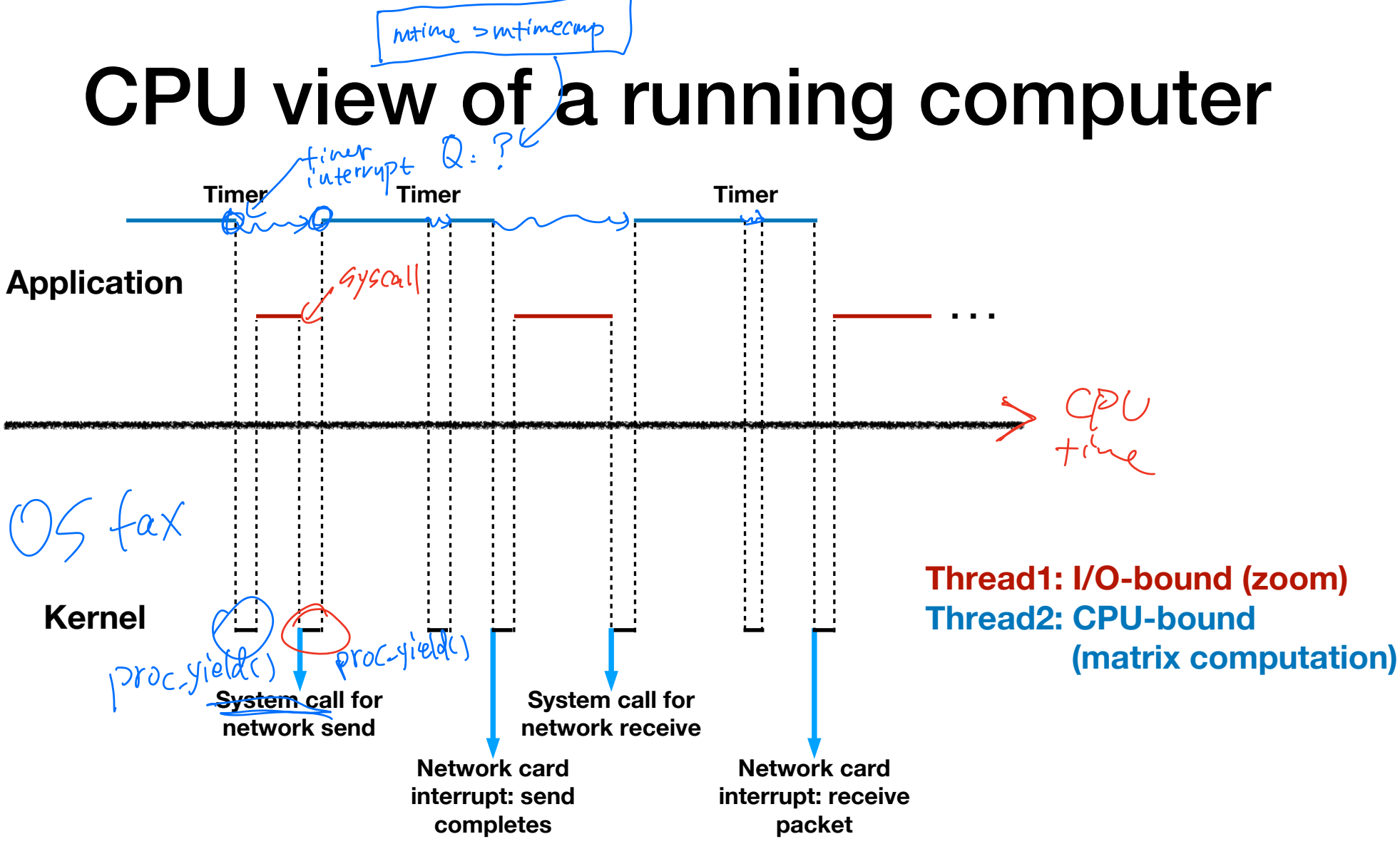
1. CPU multiplexing
2. process → machine



Kernel ~= 3 handlers

- interrupt
- exception: page fault
- syscall: file open/read/write
  - socket
  - ⋮
  - enable TI

CPU view of a running computer



a) interrupts

Q: How does a CPU know what to execute when an interrupt is triggered?

Code

CSR

mtvec

Q: How does the kernel know which interrupt has been triggered?

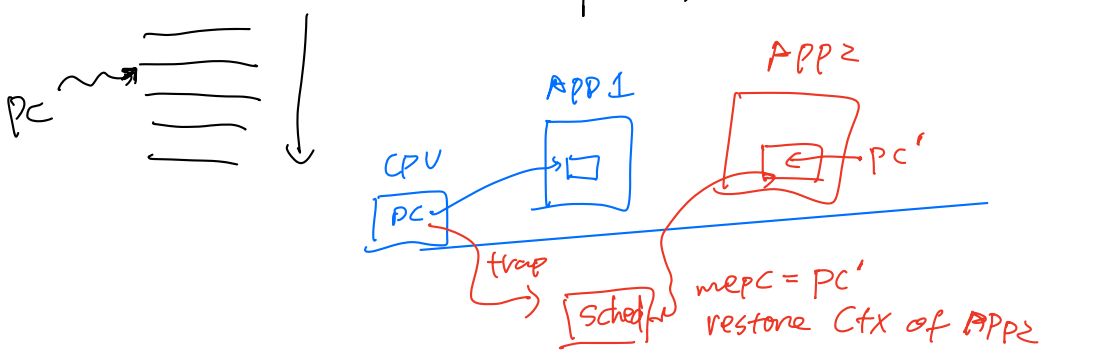
mCause

Q: After handling interrupts, where does the CPU jump to?

mret

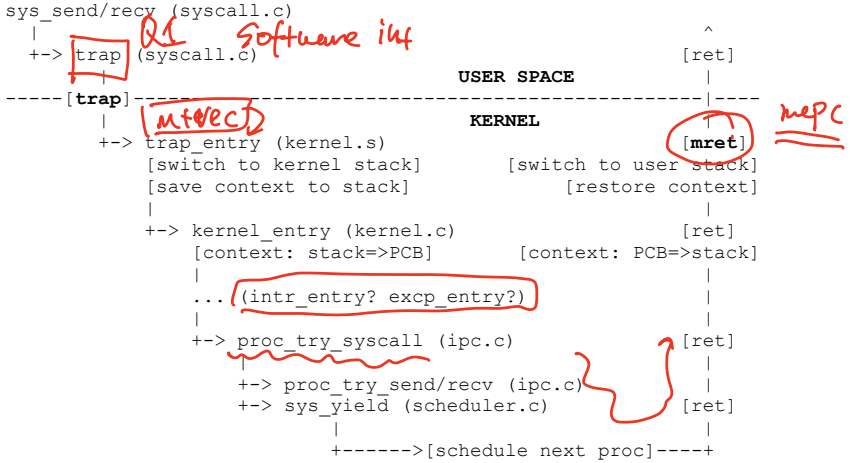
↳ pc = mepc (CSR)

when interrupted, mepc = PC for app



CS6640 Week 05b

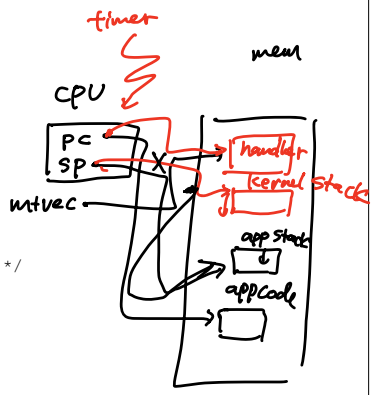
1. egos-2k+ syscall workflow



2. Trap to kernel (grass/kernel.s)
---

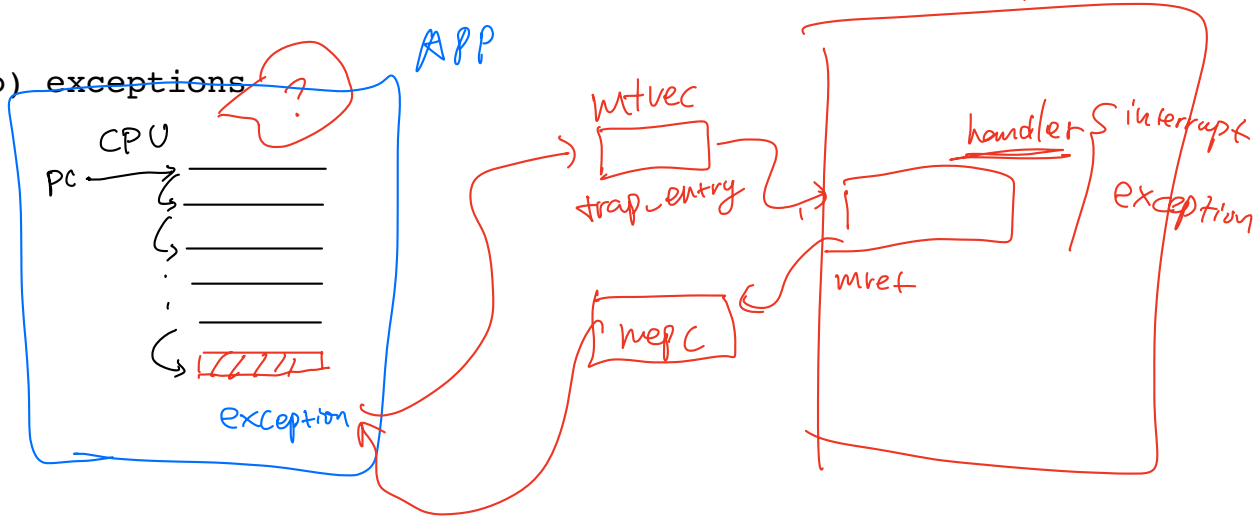
trap_entry:

```
/*
 * A. switch to the kernel stack.
 * B. save all the registers on the kernel stack.
 * C. call kernel_entry().
 * D. restore all the registers.
 * E. switch back to the process stack.
 * F. mret, returning to the process context. */

/* A */
csrw mscratch, sp
li sp, 0x80200000

/* B */
addi sp, sp, -128 /* now, sp == EGOS_STACK_TOP-32*4 */
sw a0,  0(sp)
...
sw a7,  28(sp)
sw t0,  32(sp)
...
sw t6,  56(sp)
sw s0,  60(sp)
...
sw s11, 104(sp)
sw ra,  108(sp)
sw gp,  112(sp)
sw tp,  116(sp)
csrr t0, mscratch /* Step-A has written sp to mscratch */
sw t0,  120(sp)   /* t0 holds the value of the old sp before trap_entry */

/* C */
call kernel_entry

/* D */
lw a0,  0(sp)
...
lw a7,  28(sp)
lw t0,  32(sp)
...
lw t6,  56(sp)
lw s0,  60(sp)
lw s1,  64(sp)
...
lw s11, 104(sp)
lw ra,  108(sp)
lw gp,  112(sp)
lw tp,  116(sp)

/* E */
lw sp,  120(sp)

/* F */
mret
```



Mem

b) ~~exceptions~~



Cheng Tan, OSI

3. Trap reason table
---

mCause (32bit)

Interrupt	Exception Code	Description
1	0	*Reserved*
1	1	Supervisor software interrupt
1	2	*Reserved*
1	3	Machine software interrupt
1	4	*Reserved*
1	5	Supervisor timer interrupt
1	6	*Reserved*
1	7	Machine timer interrupt
1	8	*Reserved*
1	9	Supervisor external interrupt
1	10	*Reserved*
1	11	Machine external interrupt
1	12–15	*Reserved*
1	≥16	*Designated for platform use*
0	0	Instruction address misaligned
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	*Reserved*
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	*Reserved*
0	15	Store/AMO page fault
0	16–23	*Reserved*
0	24–31	*Designated for custom use*
0	32–47	*Reserved*
0	48–63	*Designated for custom use*
0	≥64	*Reserved*

ill-formated

OR

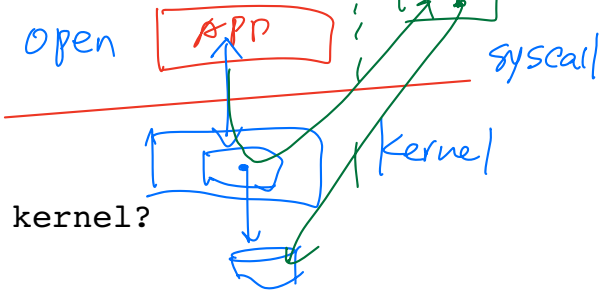
*ptr

0xdeadbeef

*ptr = 1

labs, VM

1 sys-fig



c) syscalls

Q1: How does an application trap into the kernel?

interrupt/exception

Q2: How does the kernel determine the operation requested by the application?
That is, what information is required to handle a system call?

send/recv

Q3: Where is this information stored?

- Registers
- stack
- memory (well-known)
  ⋮