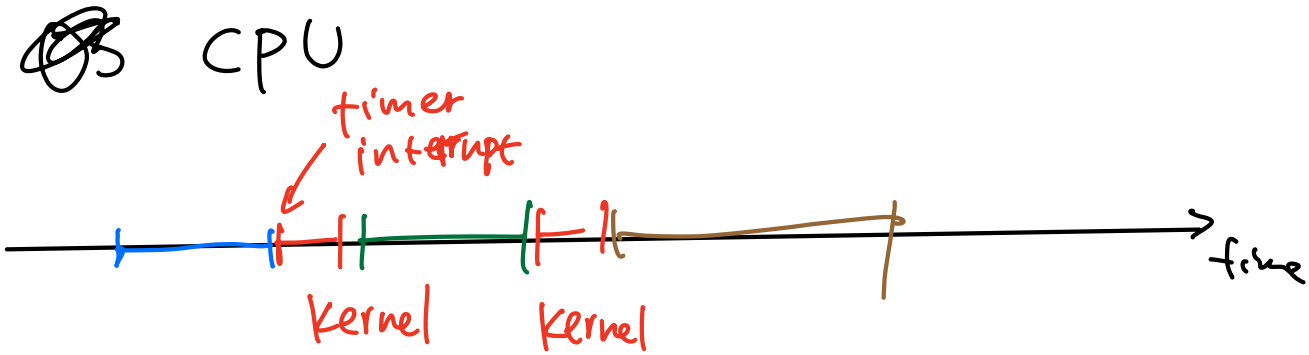
Week 4
CS4973/CS6640
01/27 2025
https://naizhengtan.github.io/25spring/

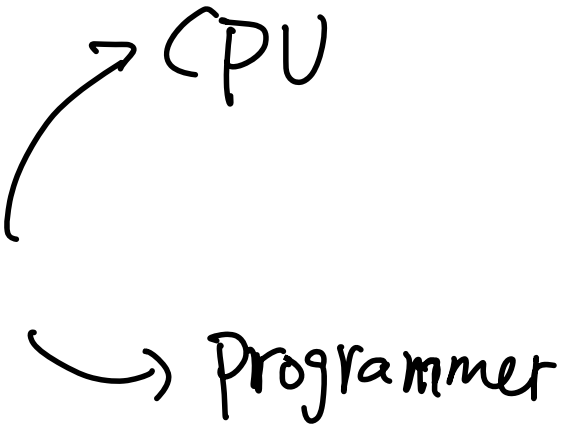
1. Timer interrupt
2. Review: OS scheduling (brief)
3. egos scheduler & lab3

---

lab2



Timer interrupts



[borrowed from Yunhao's
CS 4411/5411: Practicum in Operating Systems, 22fall]

CPU view: Core-local Interrupt (CLINT)

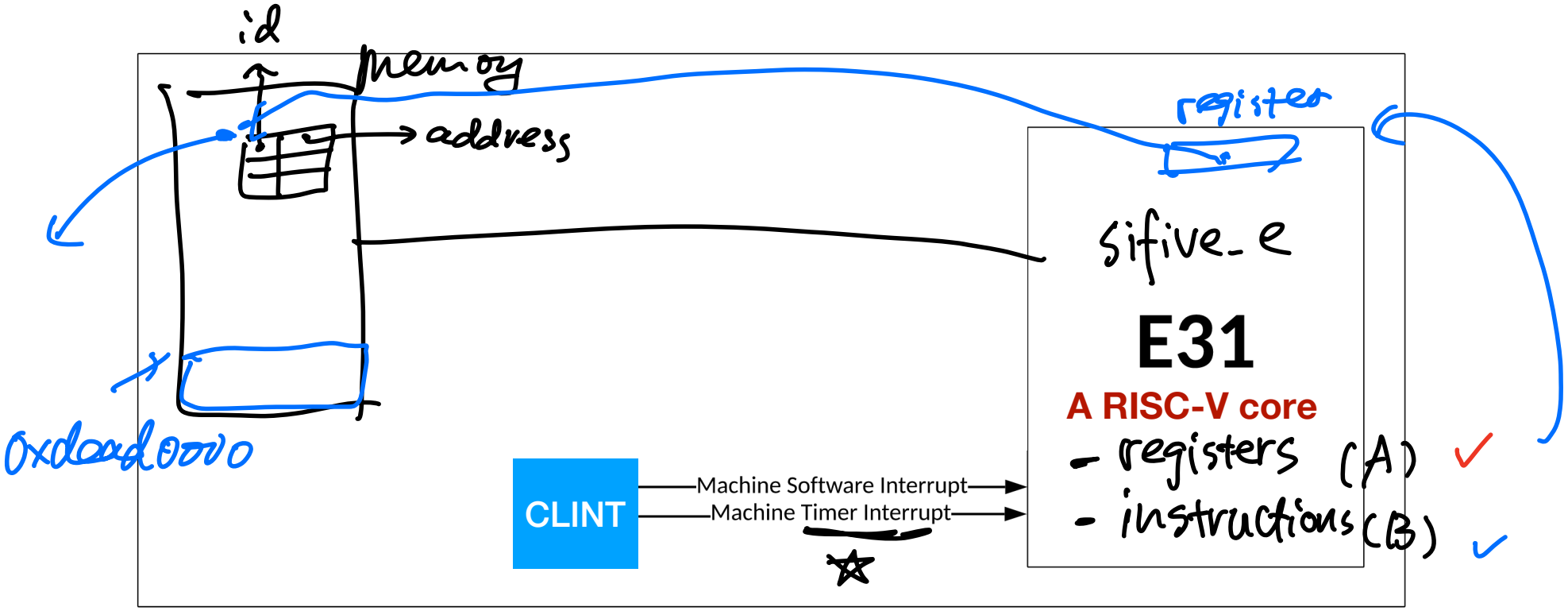


Figure 4 of Sifive FE310 manual

A programmer's view

```
void handler() {
    printf("Got a timer interrupt!");
    // (4) reset timer
}

int main() {
    // (1) register interrupt handler
    // (2) set a timer
    // (3) enable timer interrupt

    while(1);
}
```

Q: What if you were CPU designer

machine-mode

handler's address

trap

The mtvec CSR

Value	Name	Description
0	Direct	All exceptions set `pc` to BASE.
1	Vectored	Asynchronous interrupts set `pc` to BASE+4×cause.
≥2	—	*Reserved*

Table 3.5: Encoding of `mtvec` MODE field.

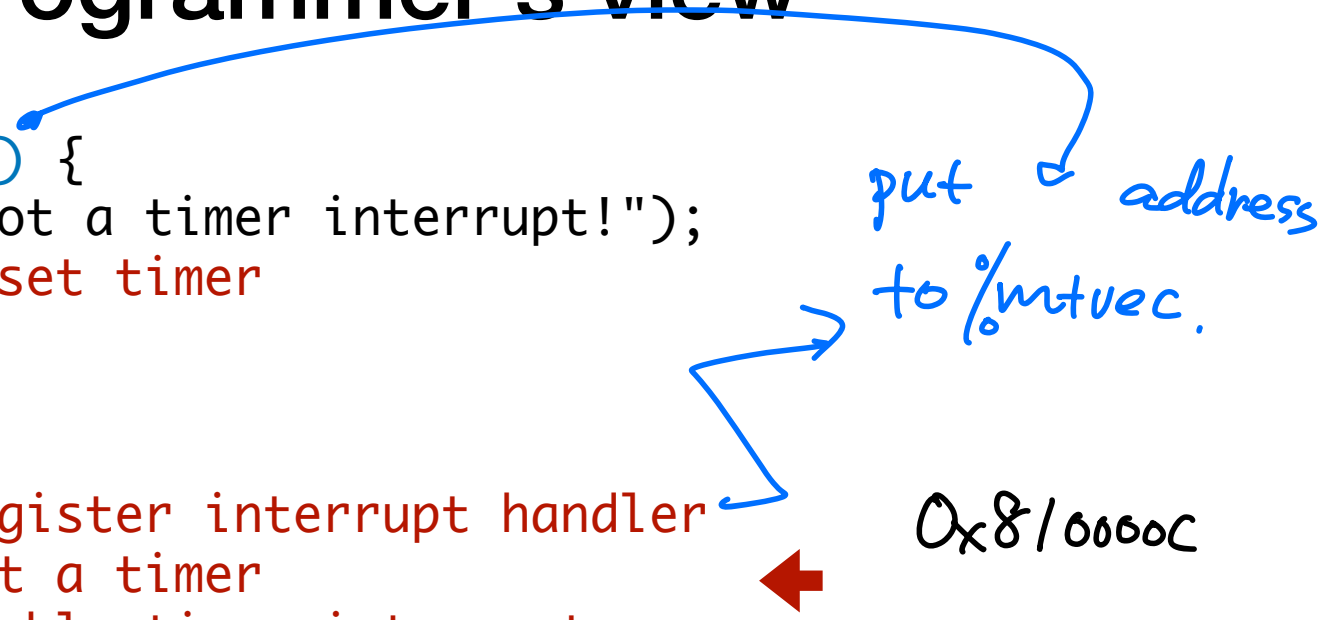
RISC-V privileged spec

A programmer's view

```
void handler() {
    printf("Got a timer interrupt!");
    // (4) reset timer
}

int main() {
    // (1) register interrupt handler
    // (2) set a timer
    // (3) enable timer interrupt

    while(1);
}
```



OSI Handout Week4

1. Timer interrupt handler

```
void handler() {
    CRITICAL("Got a timer interrupt!");
    // (4) reset timer

}

int main() {
    CRITICAL("This is a simple timer example");

    // (1) register handler() as interrupt handler


    // (2) set a timer


    // (3) enable timer interrupt


    while(1);
}
```


2. Background: RISC-V assembly II

Assembler instructions with C expression operands:

```
asm(Template : OutputOperands : InputOperands)
```

a) Template: a string that is the template for the assembler code.

```
asm("mret");
```

ret

b) OutputOperands: the C variables modified by the instructions in the Template.

```
void *sp;
asm("mv %0, sp" : "=r"(sp));
```

c) InputOperands: C expressions read by the instructions in the Template.

```
int mie;
asm("csrr %0, mie" : "=r"(mie));
asm("csrw mie, %0" ::"r"(mie | 0x80));
```

Q:

bit-wise OR

X | 0 → X

X | 1 → 1

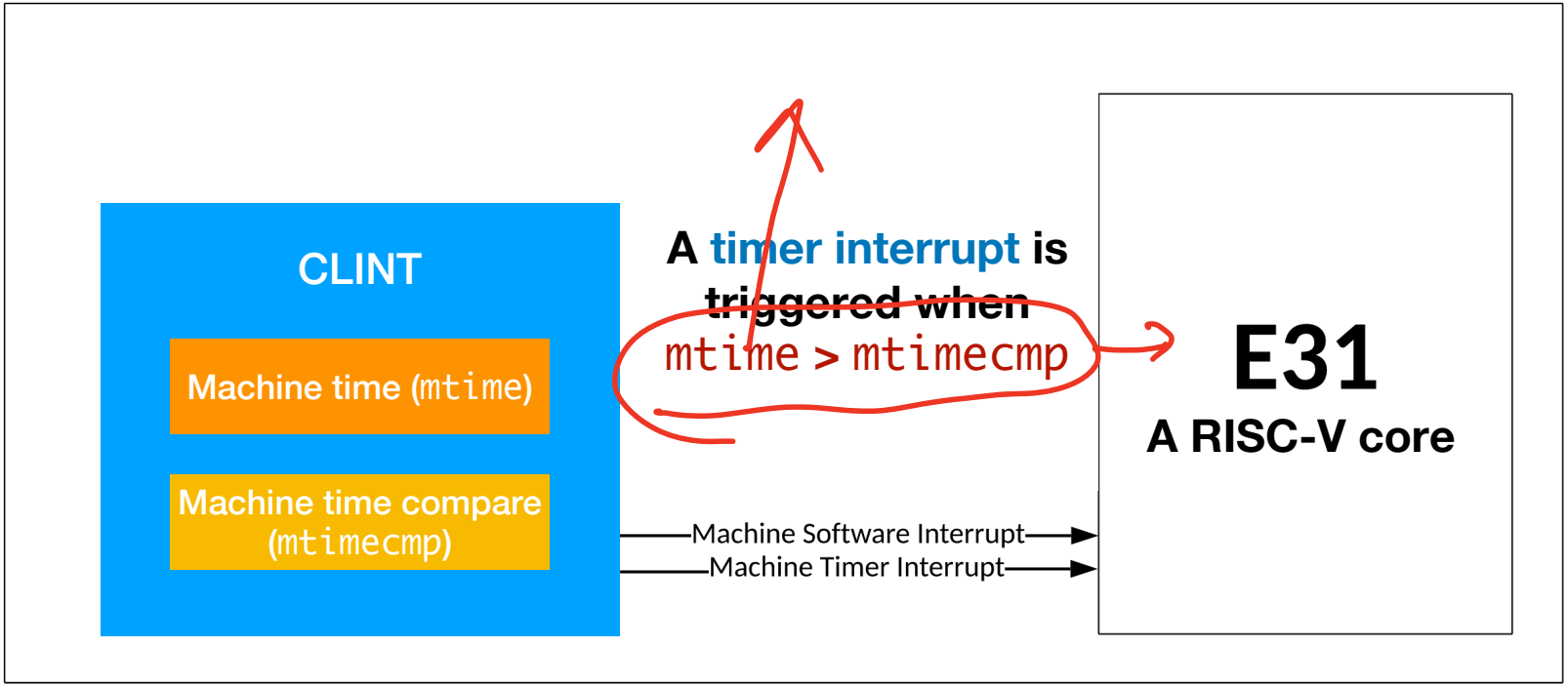
mie ← %mie

%mie ← set mie's ? bit 1

7 6 5 4 3 2 1 0

1 0 0 0 0 0 0 0

The `mtime` and `mtimecmp` CSRs



Sifive FE310 manual

CLINT CSRs are memory-mapped

`mtimecmp_set()` writes 8 bytes to

`mtime_get()` reads 8 bytes from

Address	Width	Attr.	Description
0x2000000	4B	RW	msip for hart 0
0x2004008			Reserved
...			
0x200bff7			
0x2004000	8B	RW	mtimecmp for hart 0
0x2004008			Reserved
...			
0x200bff7			
0x200bff8	8B	RW	mtime
0x200c000			Reserved

Sifive FE310 manual

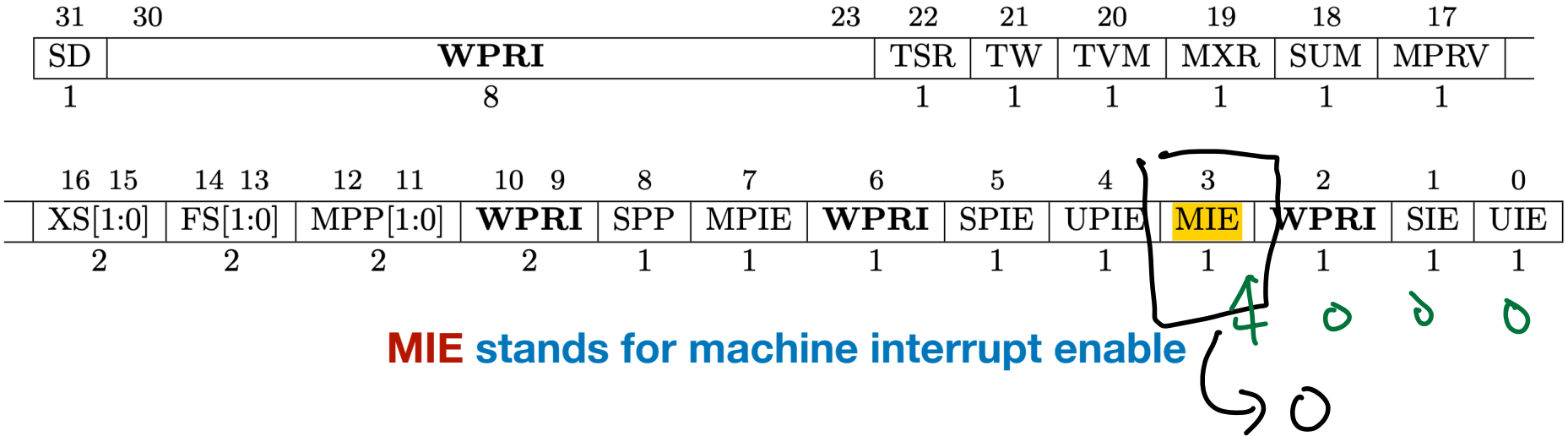
A programmer's view

```
void handler() {
    printf("Got a timer interrupt!");
    // (4) reset timer
}

int main() {
    // (1) register interrupt handler ✓
    // (2) set a timer ⌣
    // (3) enable timer interrupt

    while(1);
}
```

The mstatus CSR



RISC-V privileged spec

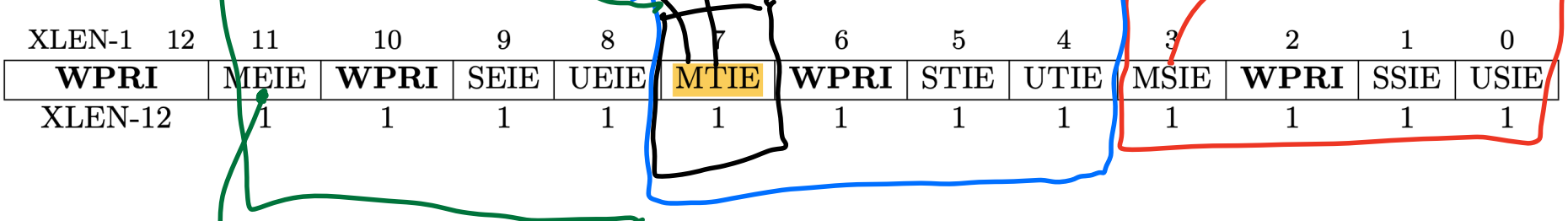
The `mie` CSR (not `mstatus.MIE`)

machine-mode timer

set 1

Software

External

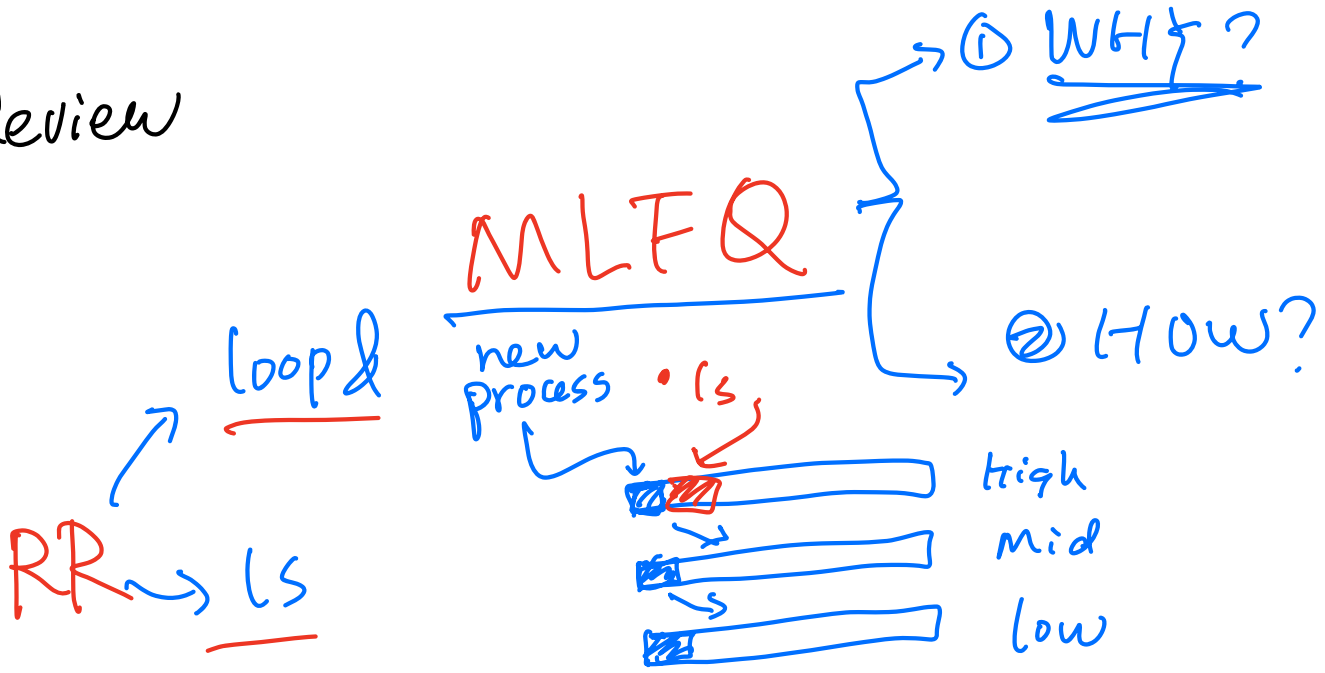


XLEN-1 12	11	10	9	8	7	6	5	4	3	2	1	0
WPRI	MEIE	**WPRI**	SEIE	UEIE	MTIE	**WPRI**	STIE	UTIE	MSIE	**WPRI**	SSIE	USIE
XLEN-12	1	1	1	1	1	1	1	1	1	1	1	1

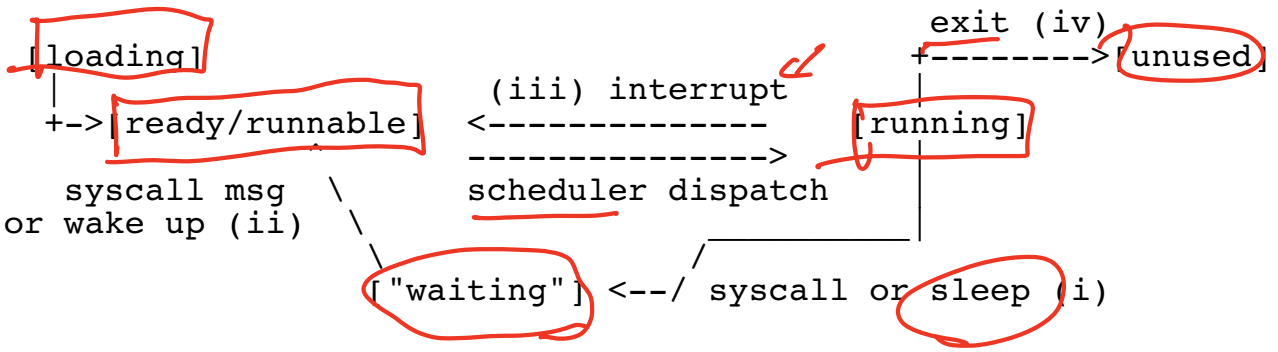
MTIE stands for machine timer interrupt enable

RISC-V privileged spec

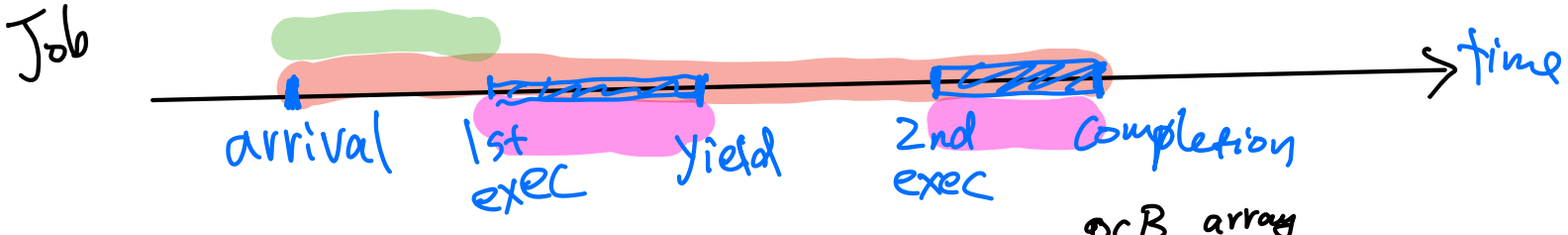
- Review



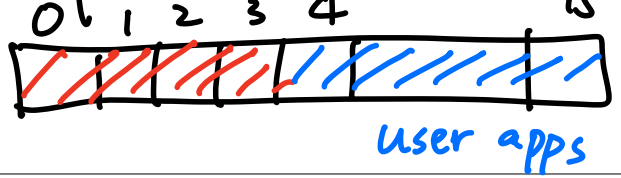
terminated



- metrics
  - turnaround time : (completion - arrival)
  - response time : (1st exec - arrival)
  - CPU time : CPU consumed time



PCB array



3. Machine-mode exception CSRs

a) **mstatus**
Machine Status, holds the global interrupt enable, along with a plethora of other state.

b) **mie**
Machine Interrupt Enable, lists which interrupts the processor can take and which it must ignore

c) **mcause**
Machine Exception Cause, indicates which exception occurred

d) **mtvec**
Machine Trap Vector, holds the address the processor jumps to when an exception occurs

e) **mepc**
Machine Exception PC, points to the instruction where the exception occurred

f) **mtval**
Machine Trap Value, holds additional trap information: the faulting address for address exceptions, the instruction itself for illegal instruction exceptions, and zero for other exceptions

g) **mip**
Machine Interrupt Pending, lists the interrupts currently pending

4. egos process management (sifive_e)

a) process control block (PCB)

[grass/process.h]

```
struct process {
    int pid;
    int status;
    int receiver_pid; /* used when waiting to send a message */
    void *sp, *mepc;  /* process context = stack pointer (sp)
                       * + machine exception program counter (mepc) */
    // scheduling attributes
    union {
        unsigned char      chars[64];
        unsigned int       ints[16];
        float              floats[16];
        unsigned long long longlongs[8];
        double             doubles[8];
    } schd_attr;
};
```

b) global process data structures

[grass/kernel.c]

```
int proc_curr_idx;
struct process proc_set[MAX_NPROCESS];
```

[grass/process.h]

```
#define curr_pid      proc_set[proc_curr_idx].pid
#define curr_status   proc_set[proc_curr_idx].status
```

c) process life cycles

[grass/scheduler.c]

life-cycle functions:

* **proc_on_arrive**(int pid): when creating pid
* **proc_yield**(): when deciding next running process
* **proc_on_stop**(int pid): when destroying pid

a process's life cycle:

- proc_on_arrive() ->
  - proc_yield() -> [other proc] -> [ctx_switch to this proc] ->
  - proc_yield() -> [other proc] -> [ctx_switch to this proc] ->
  - ...
- -> proc_on_stop()

Next…

- two bummers in setting timers
- interrupts beyond CLINT
- RISC-V fine-grained control of interrupts
- how to know which interrupt is triggered?

a bummer: mtime rollover

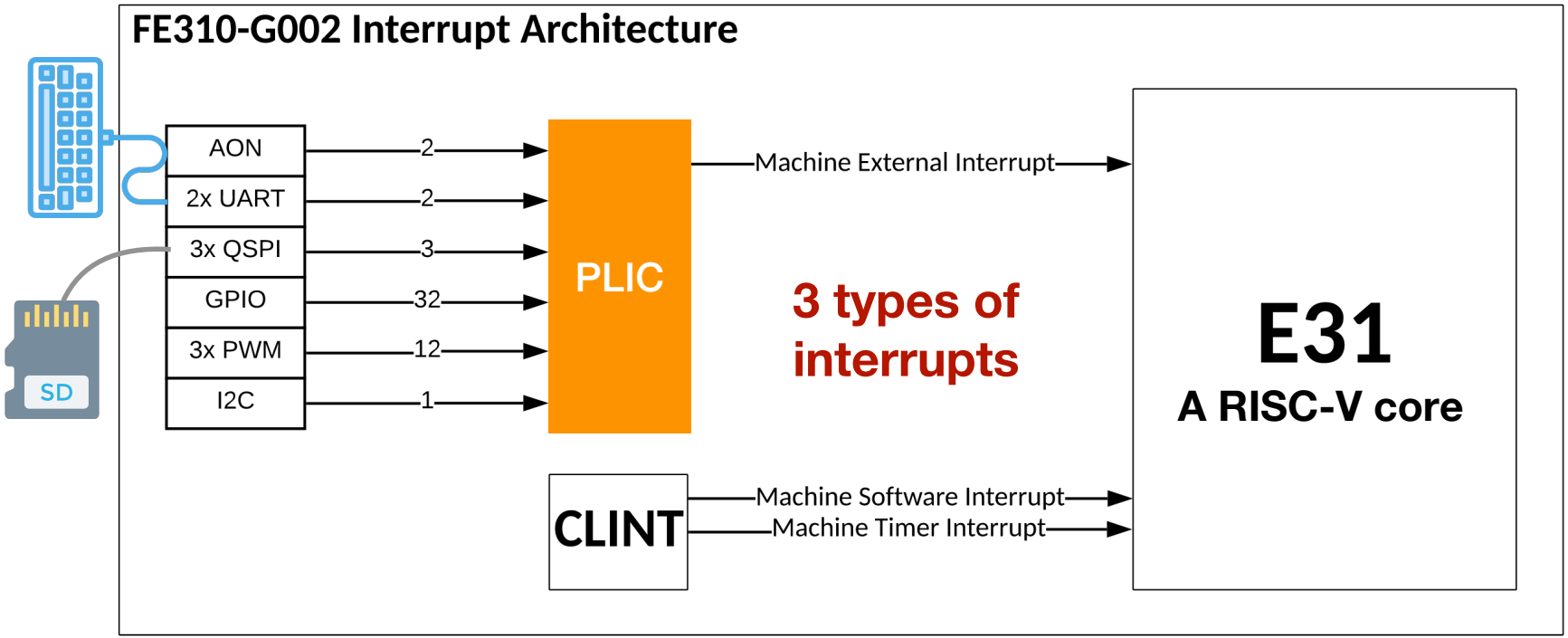
When reading the lower 4 bytes, mtime is
0x00000000 0xffffffff

When reading the higher 4 bytes, mtime is
0x00000001 0x00000000

Combine the two 0x00000001ffffffff is wrong!

There is another bummer: mtime overflow

Platform Interrupt Controller (PLIC)



Sifive FE310 manual

`mie` provides fine-grained control

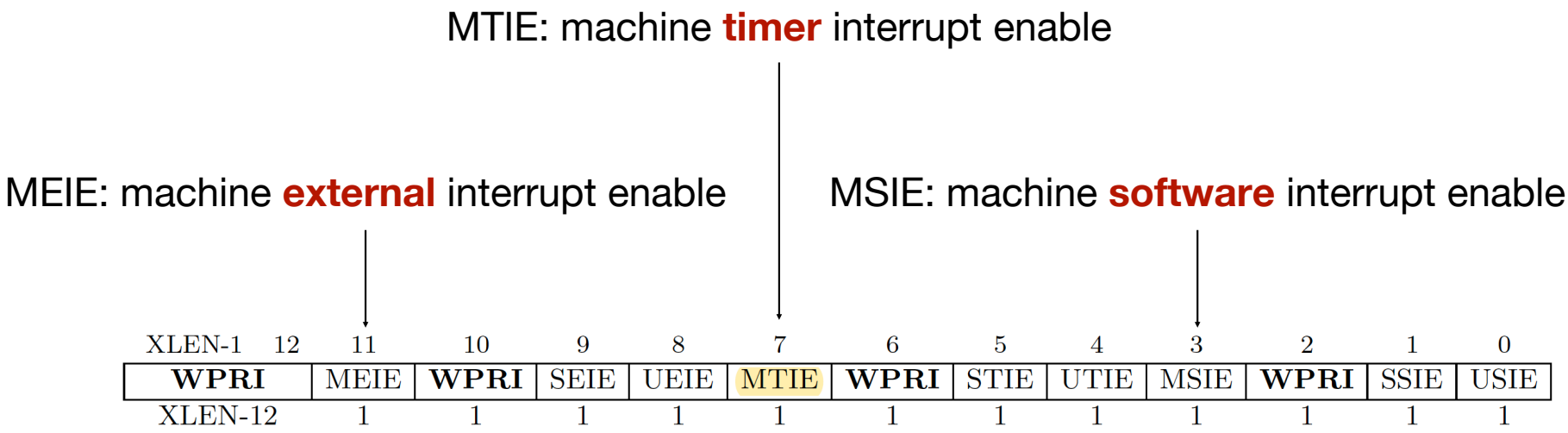


Figure 3.12: Machine interrupt-enable register (`mie`).

Timer is interrupt #7

Interrupts

Exceptions

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*

System calls?

Interrupts

Exceptions

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*