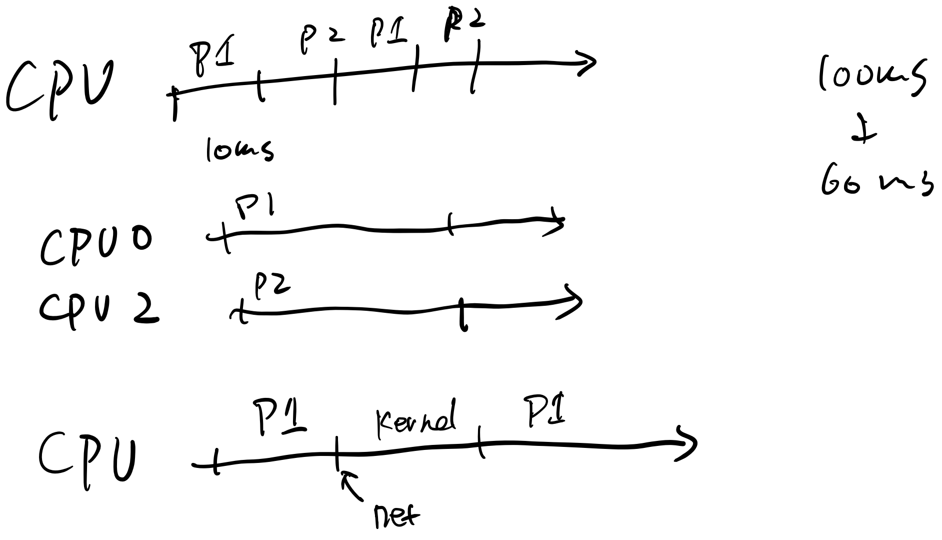
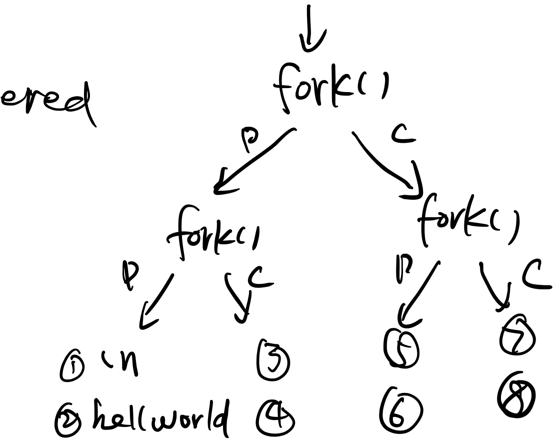
1. Last time
2. Intro to concurrency
3. Memory consistency model



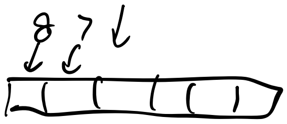
```
fork()
fork()
printf("\n helloworld")
```

line-buffered

"helloworld\n"



$$\# \text{schedule} = \frac{8!}{2^4} \leftarrow ① \rightarrow ②$$



1. Example to illustrate interleavings: say that thread A executes f() and thread B executes g(). (Here, we are using the term "thread" abstractly. This example applies to any of the approaches that fall under the word "thread".)

a. [this is pseudocode]

```
    int x;

    int main(int argc, char** argv) {

        tid tid1 = thread_create(f, NULL);
        tid tid2 = thread_create(g, NULL);

        thread_join(tid1);
        thread_join(tid2);

        printf("%d\n", x);

    }

    void f() {
        x = 1;
        thread_exit();
    }

    void g() {
        x = 2;
        thread_exit();
    }
```

?

f()
g() ⇒ 2

0 1 or 2 ✓

Undefined

What are possible values of x after A has executed f() and B has executed g()? In other words, what are possible outputs of the program above?

b. Same question as above, but f() and g() are now defined as follows

```
    int y = 12;

    f() { x = y + 1; }
    g() { y = y * 2; }
```

f: x=13
g: y=24

g: y=24
f: x=25

$\boxed{13 \text{ or } 25}$

24 0

What are the possible values of x?

24, 13, or 25, 26

c. Same question as above, but f() and g() are now defined as follows:

```
    int x = 0;

    f() { x = x + 1; }
    g() { x = x + 2; }
```

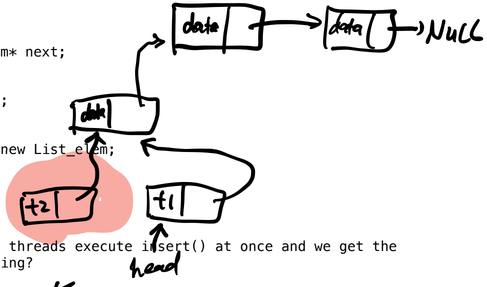
What are the possible values of x?

3, 1,2,or 3,

x: 0x5000

2. Linked list example

```
    struct List_elem {
        int data;
        struct List_elem* next;
    };

    List_elem* head = 0;

    insert(int data) {
        List_elem* l = new List_elem;
        l->data = data;
        l->next = head;
        head = l;
    }
```



What happens if two threads execute insert() at once and we get the following interleaving?

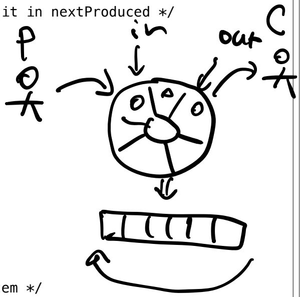
```
    thread 1: l->next = head
    thread 2: l->next = head
    thread 2: head = l;
    thread 1: head = l;
```

movq 0x5000, %rbx =0
addq $1, %rbx =1
movq %rbx, 0x5000 =1

movq 0x5000, %rbx =0
addq $2, %rbx
movq %rbx, 0x5000 = 2

3. Producer/consumer example:

```
    /*
        "buffer" stores BUFFER_SIZE items
        "count" is number of used slots. a variable that lives in memory
        "out" is next empty buffer slot to fill (if any)
        "in" is oldest filled slot to consume (if any)
    */

    void producer (void *ignored) {

        for (;;) {
            /* next line produces an item and puts it in nextProduced */
            nextProduced = means_of_production();
            while (count == BUFFER_SIZE)
                ; // do nothing
            buffer [in] = nextProduced;
            in = (in + 1) % BUFFER_SIZE;
            count++;
        }
    }

    void consumer (void *ignored) {
        for (;;) {
            while (count == 0)
                ; // do nothing
            nextConsumed = buffer[out];
            out = (out + 1) % BUFFER_SIZE;
            count--;
            /* next line abstractly consumes the item */
            consume_item(nextConsumed);
        }
    }
```



```
    /*
        what count++ probably compiles to:
            reg1 <-- count # load
            reg1 <-- reg1 + 1 # increment register
            count <-- reg1 # store

        what count-- could compile to:
            reg2 <-- count # load
            reg2 <-- reg2 - 1 # decrement register
            count <-- reg2 # store
    */
```

What happens if we get the following interleaving?

```
        reg1 <-- count
        reg1 <-- reg1 + 1
        reg2 <-- count
        reg2 <-- reg2 - 1
        count <-- reg1
        count <-- reg2
```

4. Some other examples. What is the point of these?

[From S.V. Adve and K. Gharachorloo, IEEE Computer, December 1996, 66-76. http://sadve.cs.illinois.edu/Publications/computer96.pdf]

a. Can both "critical sections" run?

```
        int flag1 = 0, flag2 = 0;

        int main () {
            tid id = thread_create (p1, NULL);
            p2 (); thread_join (id);
        }

        void p1 (void *ignored) {
            flag1 = 1;
            if (!flag2) {
                critical_section_1 ();
            }
        }

        void p2 (void *ignored) {
            flag2 = 1;
            if (!flag1) {
                critical_section_2 ();
            }
        }
```

NO

b. Can use() be called with value 0, if p2 and p1 run concurrently?

```
        int data = 0, ready = 0;

        void p1 () {
            data = 2000;
            ready = 1;
        }
        int p2 () {
            while (!ready) {}
            use(data);
        }
```

← 0?

NO

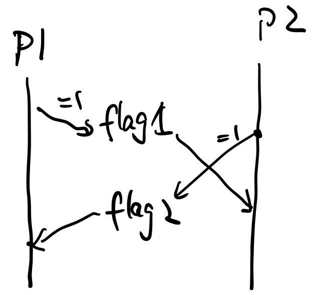
c. Can use() be called with value 0?

```
        int a = 0, b = 0;

        void p1 (void *ignored) { a = 1; }

        void p2 (void *ignored) {
          if (a == 1)
            b = 1;
        }

        void p3 (void *ignored) {
          if (b == 1)
            use (a);
        }
```

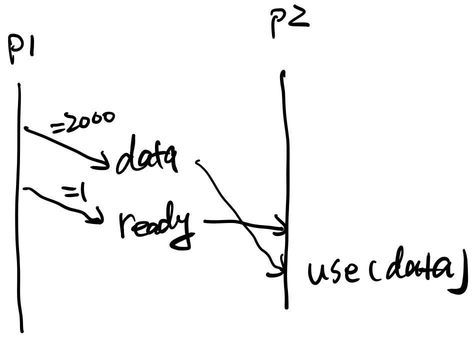
← 0?

4.a.

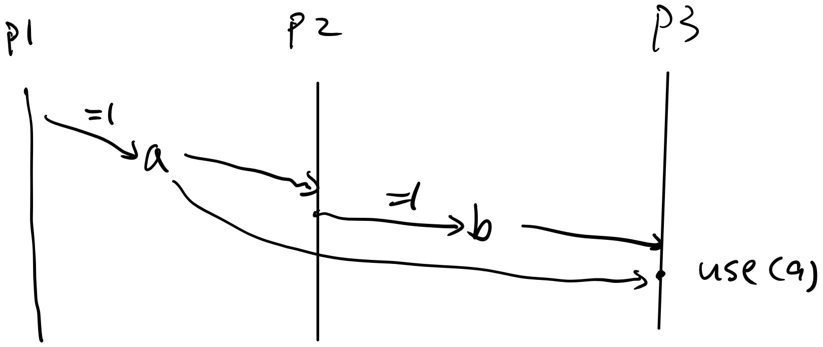
↓



4.b



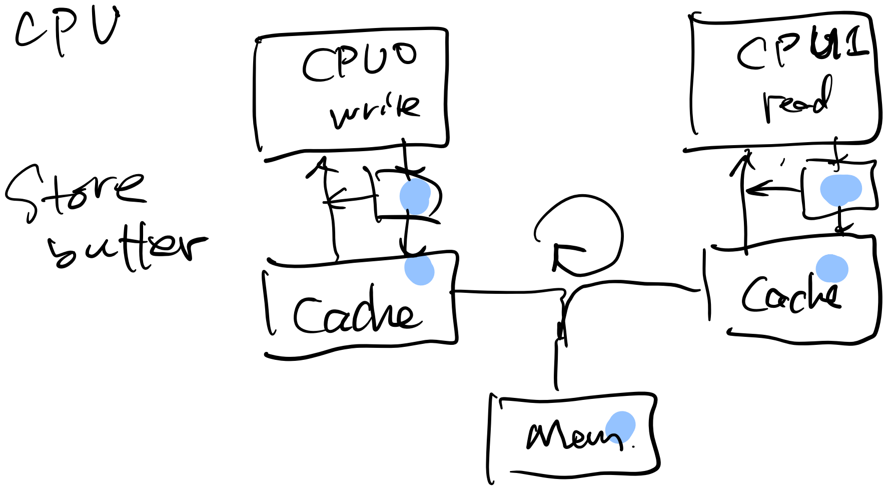
4.c

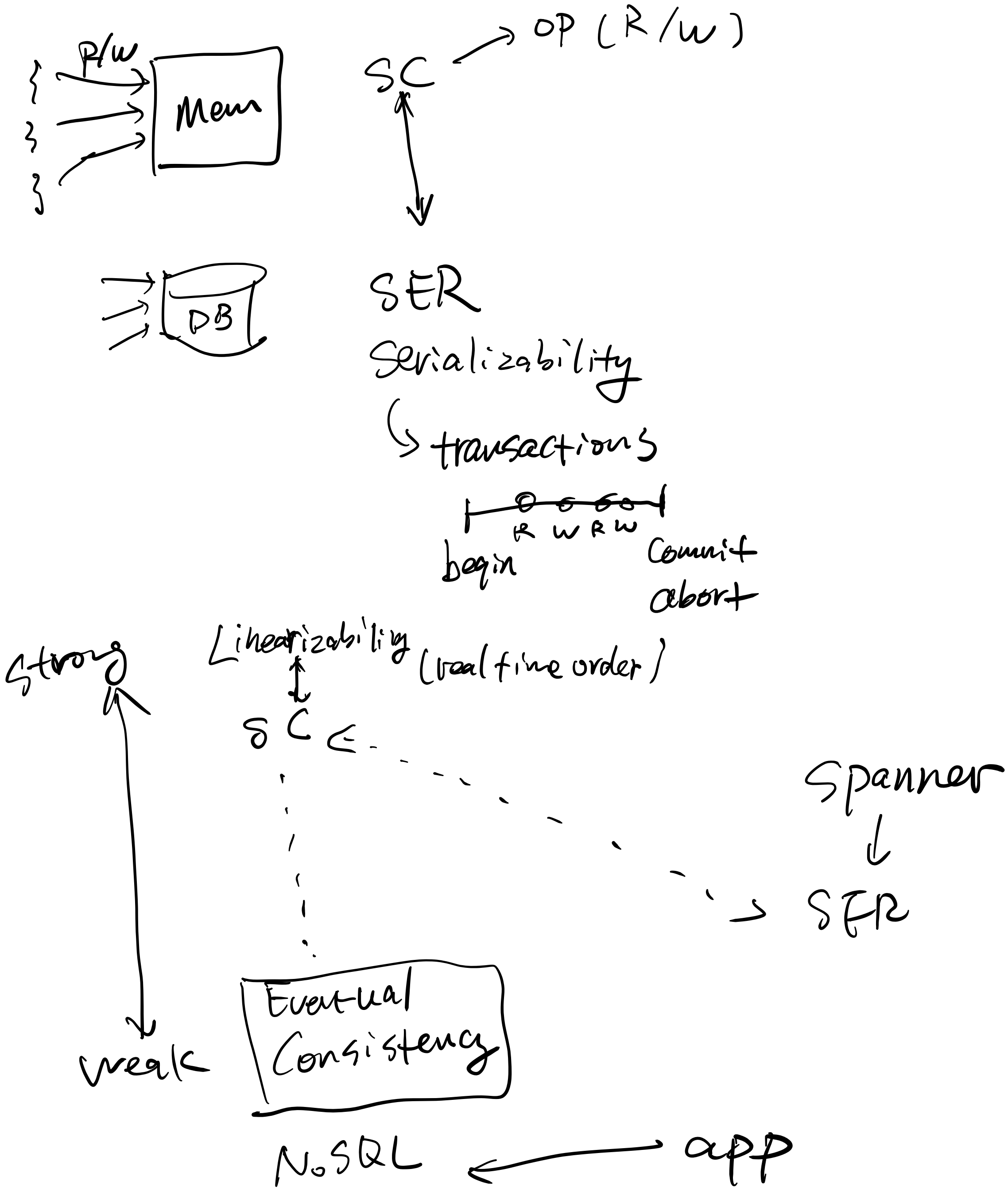


SC. memory model

↳ ≈ 1 CPU

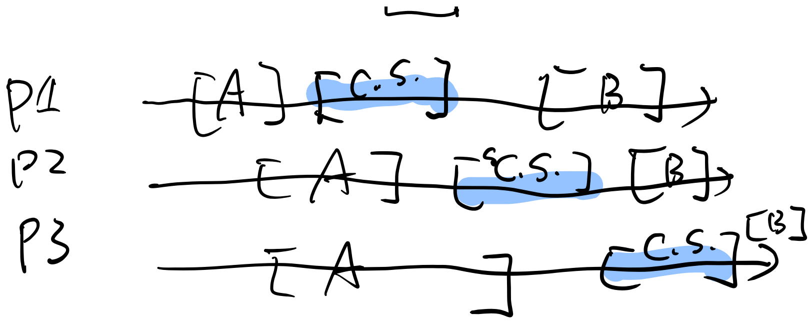
TSO

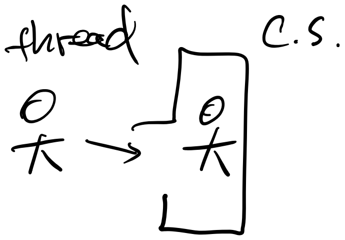




Critical section ⇒ managing Concurrency

program: $[\overrightarrow{A\ ;\ C.S.}\ ;\ B]$





① mutual exclusion

② progress

③ bounded waiting