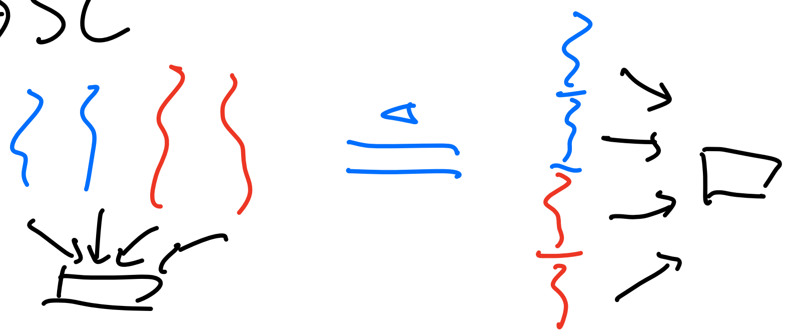
week 11a
CS6640
03/17 2026
https://naizhengtan.github.io/26spring/

☑ 1. on the difficulty of concurrency
☑ 2. challenges beyond interleaving
□ 3. consistency model

SC



OSI handout Week10

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();
    }
```

x= 1 OR 2

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; }
```

x= 13 OR 25

What are the possible values of x?

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; }
```

x=1 OR 2 OR3

What are the possible values of x?

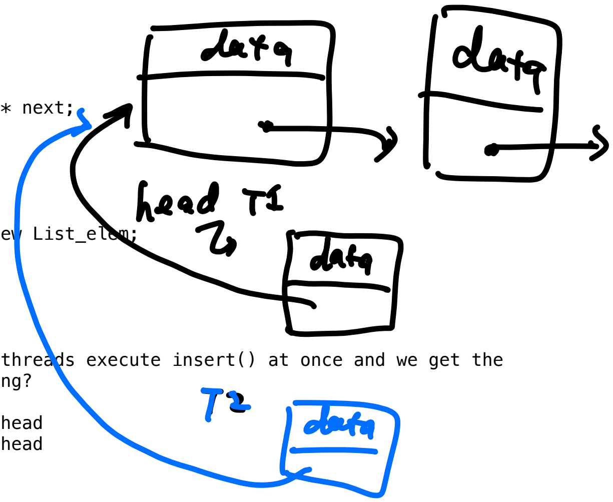
x=x+1 → { lw t0, 0x5000
addi t0, t0, 1
sw t0, 0x5000 }

&X = 0x5000

X=2
(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;
    }
```

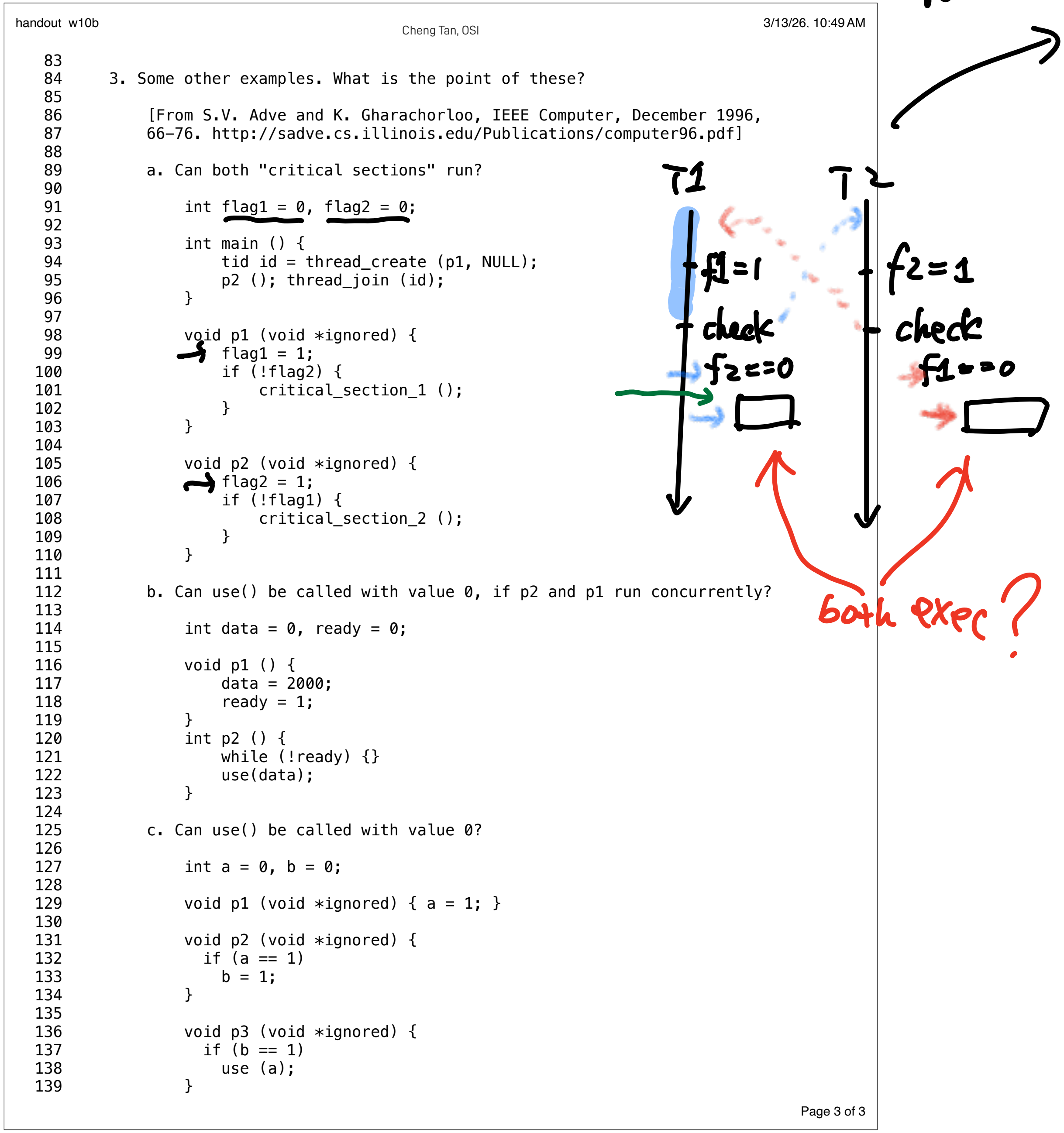


What happens if two threads execute insert() at once and we get the following interleaving?

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

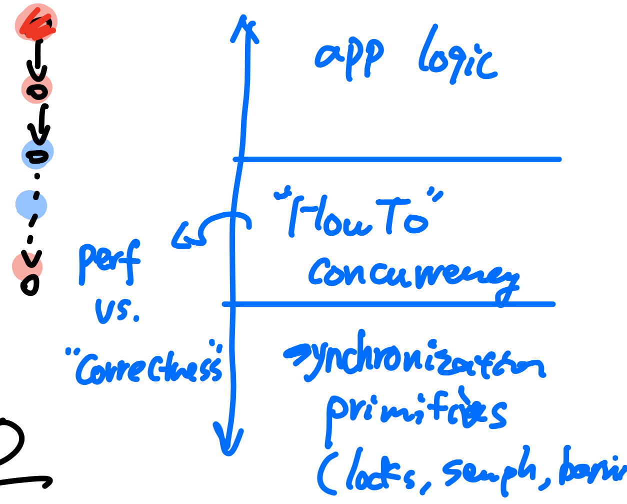
data race

x=1

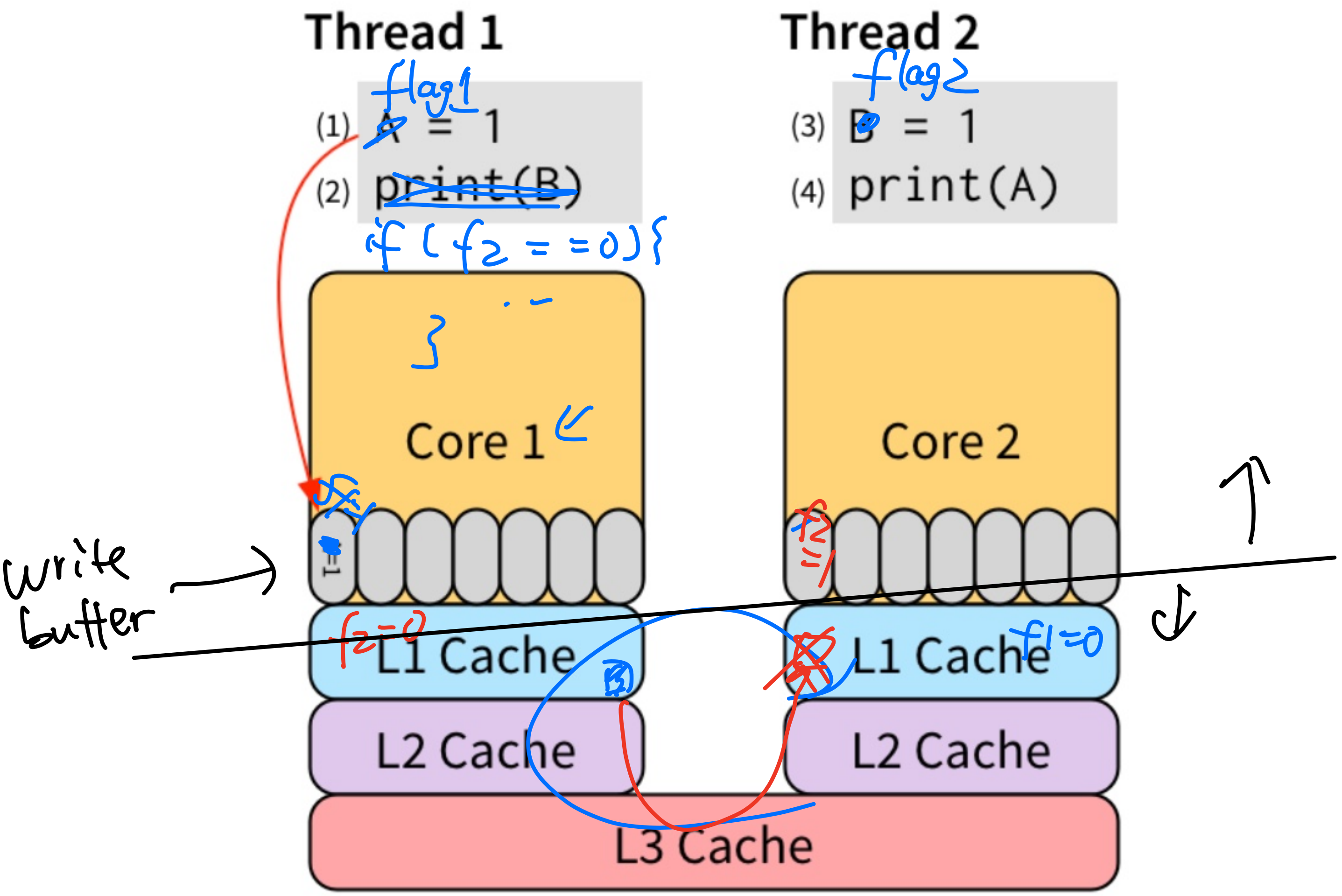


```
3. Some other examples. What is the point of these?

   [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 ();
           }
       }

   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);
       }

   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);
       }
```

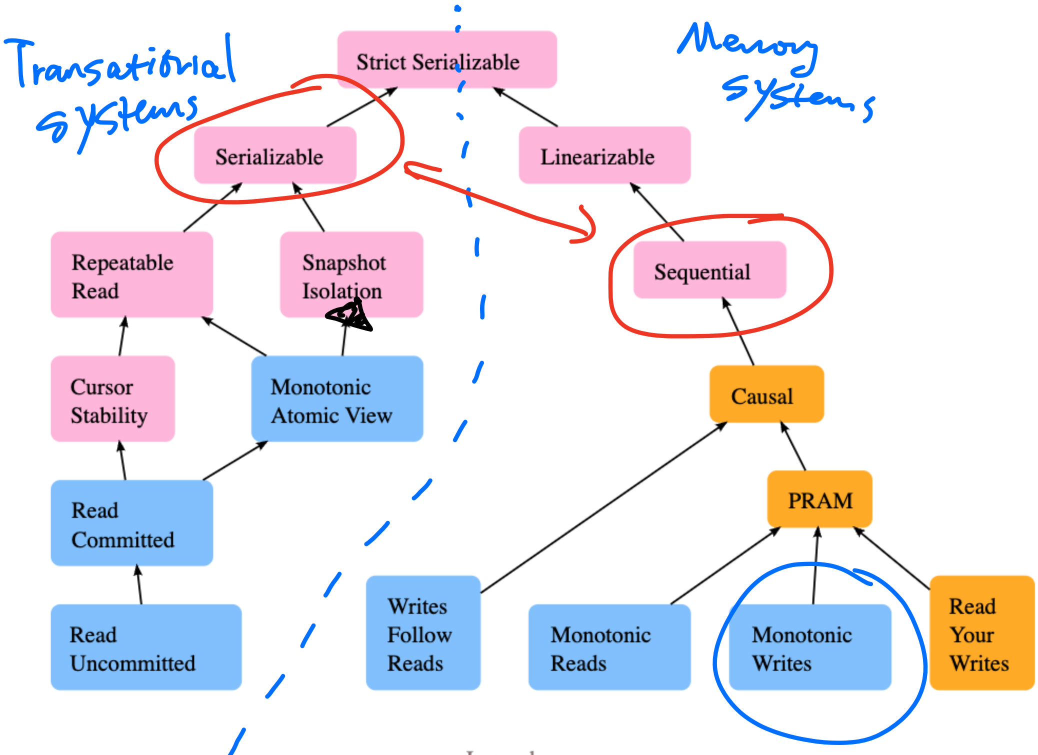
NO → Sequential consistency.



TSO



Borrowed from blog “Memory Consistency Model: A Tutorial”, James Bornholt.
https://www.cs.utexas.edu/~bornholt/post/memory-models.html



Legend

Unavailable	Not available during some types of network failures. Some or all nodes must pause operations in order to ensure safety.
Sticky Available	Available on every non-faulty node, so long as clients only talk to the same servers, instead of switching to new ones.
Total Available	Available on every non-faulty node, even when the network is completely down.