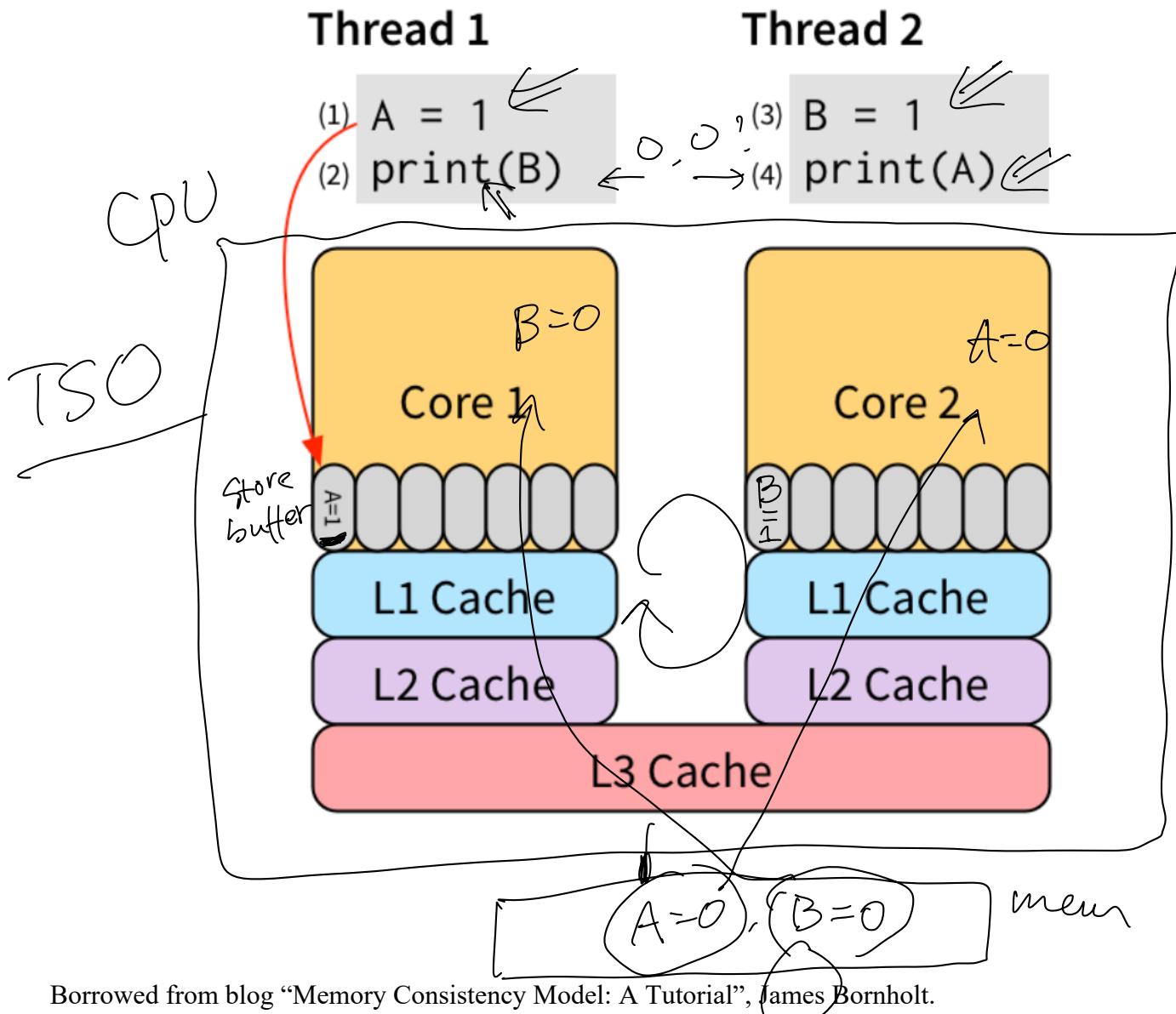


$A = 0; B = 0;$



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

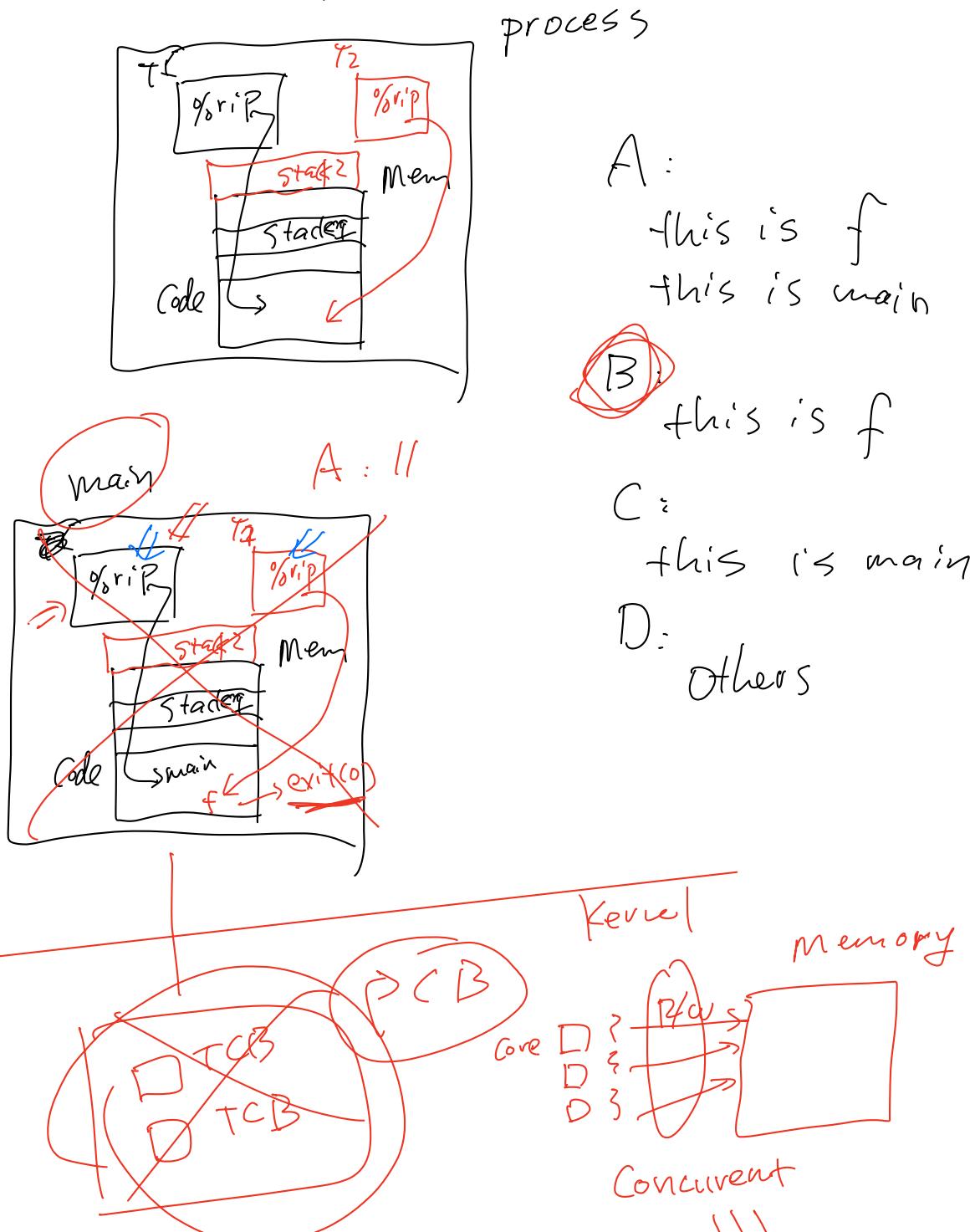
0. Processes vs. threads

1. Intro to concurrency

2. Managing concurrency

3. Mutexes

Q: What's the difference between processes and threads?



1. Intro to Concurrency

Sequential Consistency

handout w05b Cheng Tan, CS3650 2/24/11 11:27 AM

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; // global var
int main(int argc, char** argv) {
    tid tid1 = thread_create(f, NULL);
    tid tid2 = thread_create(g, NULL); ← A
    thread_join(tid1); ← B
    thread_join(tid2);
    printf("%d\n", x); ⇒ { 2 v OR f → g : 2
}
}

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

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

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 x;
int y = 12; ← // global
f() { x = y + 1; } X=13 X=24+1=25 13 or 25
g() { y = y * 2; } Y=24; f → g : 13
What are the possible values of x? g → f : 25

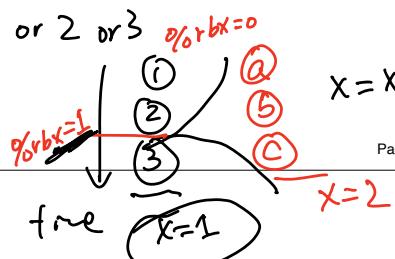
```

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

```

int x = 0; // global 3. X=X+1 ⇒
f() { x = x + 1; }
g() { x = x + 2; } ⇒ 1 or 2 or 3 % or b x = 0 @
What are the possible values of x? | (1) / (2) / (3) / (4) / (5) X=X+2 ⇒

```



handout w05b

Cheng Tan, CS360

2/7/24, 11:27 AM

```

58
59
60 2. Linked list example
61
62  {
63     struct List_elem {
64         int data;
65         struct List_elem* next;
66     };
67
68     List_elem* head = 0;
69
70     insert(int data) {
71         List_elem* l = new List_elem;
72         l->data = data;
73         l->next = head;
74         head = l;
75     }
76
77     What happens if two threads execute insert() at once and we get the
78     following interleaving?
79
80     ↗ thread 1: l->next = head
81     ↗ thread 2: l->next = head
82     ↗ thread 2: head = l;
83     ↗ thread 1: head = l;
84
85     T1
86
87     T2
88
89
90     Address of x
91
92     ① movq 0x5000, %rbx      = 0
93     ② addq $1, %rbx        = 1
94     ③ movq %rbx, 0x5000    = 1
95
96     ④ movq 0x5000, %rbx
97     ⑤ addq $2, %rbx
98     ⑥ movq %rbx, 0x5000

```

The diagram illustrates the state of a linked list during insertion. It shows three nodes: one with data and a blue arrow pointing to the next node, another with data and a blue arrow pointing to 'NULL', and a third with data and a blue arrow pointing back to the first node. A variable 'head' is shown with a blue arrow pointing to the first node. Below this, code for the 'insert' function is shown, with the assignment 'l->next = head' highlighted in red.

Further down, a scenario of interleaving two threads (T1 and T2) is depicted. Thread T1's modification 'head = l;' is shown in red. Thread T2's modifications 'l->next = head' and 'head = l;' are shown in green. The resulting state is labeled 'LOST DATA' with a red arrow pointing to the second node, which now has data but points to 'NULL' instead of the intended third node. A pink cloud surrounds the second node.

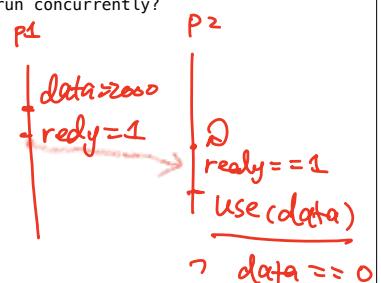
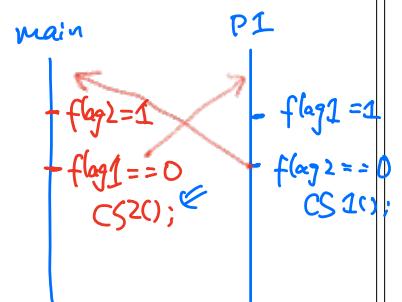
The diagram shows assembly code with annotations for register values. The first three lines (movq, addq, movq) are grouped under a circled '1'. The value '0' is written above the first line, and '1' is written below the third line, with arrows indicating the flow of the value. The next three lines (movq, addq, movq) are grouped under a circled '2'. The value '0' is written above the first line, and '2' is written below the third line, with arrows indicating the flow of the value. The final three lines (movq, addq, movq) are grouped under a circled '3'. The value '1' is written above the first line, and '1' is written below the third line, with arrows indicating the flow of the value.

1 CS5600 Week06.a
 2
 3. Some concurrent programs. What is the point of these?
 4
 [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?

```

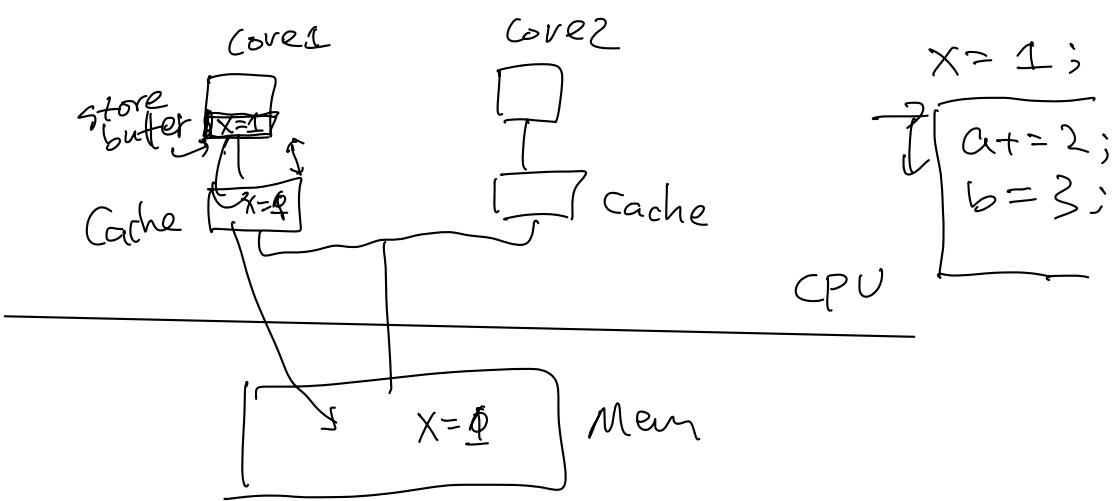
10 int flag1 = 0, flag2 = 0;
11
12 int main () {
13     tid id = thread_create (p1, NULL);
14     p2 (); thread_join (id);
15 }
16
17 void p1 (void *ignored) {
18     flag1 = 1; == 0: False
19     if (!flag2) {
20         critical_section_1 ();
21     }
22 }
23
24 void p2 (void *ignored) {
25     flag2 = 1;
26     if (!flag1) {
27         critical_section_2 ();
28     }
29 }
30
31 b. Can use() be called with value 0, if p2 and p1 run concurrently?
32
33 int data = 0, ready = 0;
34
35 void p1 () {
36     data = 2000;
37     ready = 1;
38 }
39 int p2 () {
40     while (!ready) {}
41     use(data);
42 }
43
44 c. Can use() be called with value 0?
45
46 int a = 0, b = 0;
47
48 void p1 (void *ignored) { a = 1; }
49
50 void p2 (void *ignored) {
51     if (a == 1)
52         b = 1;
53 }
54
55 void p3 (void *ignored) {
56     if (b == 1)
57         use (a);
58 }
59
60
  
```



61 2. Protecting the linked list...

```

62
63     Mutex list_mutex;
64
65     insert(int data) {
66         List_elem* l = new List_elem;
67         l->data = data;
68
69         acquire(&list_mutex);
70
71         l->next = head;
72         head = l;
73
74         release(&list_mutex);
75
76
77
  
```



Concurrency problem from Software

```

int x = 0; // a global variable
void foo() { // T1
    for (int i=0; i<100; i++) {
        x = 1;
        printf("%d", x);
    }
}
void bar() { // T2
    x = 0;
}
    
```

