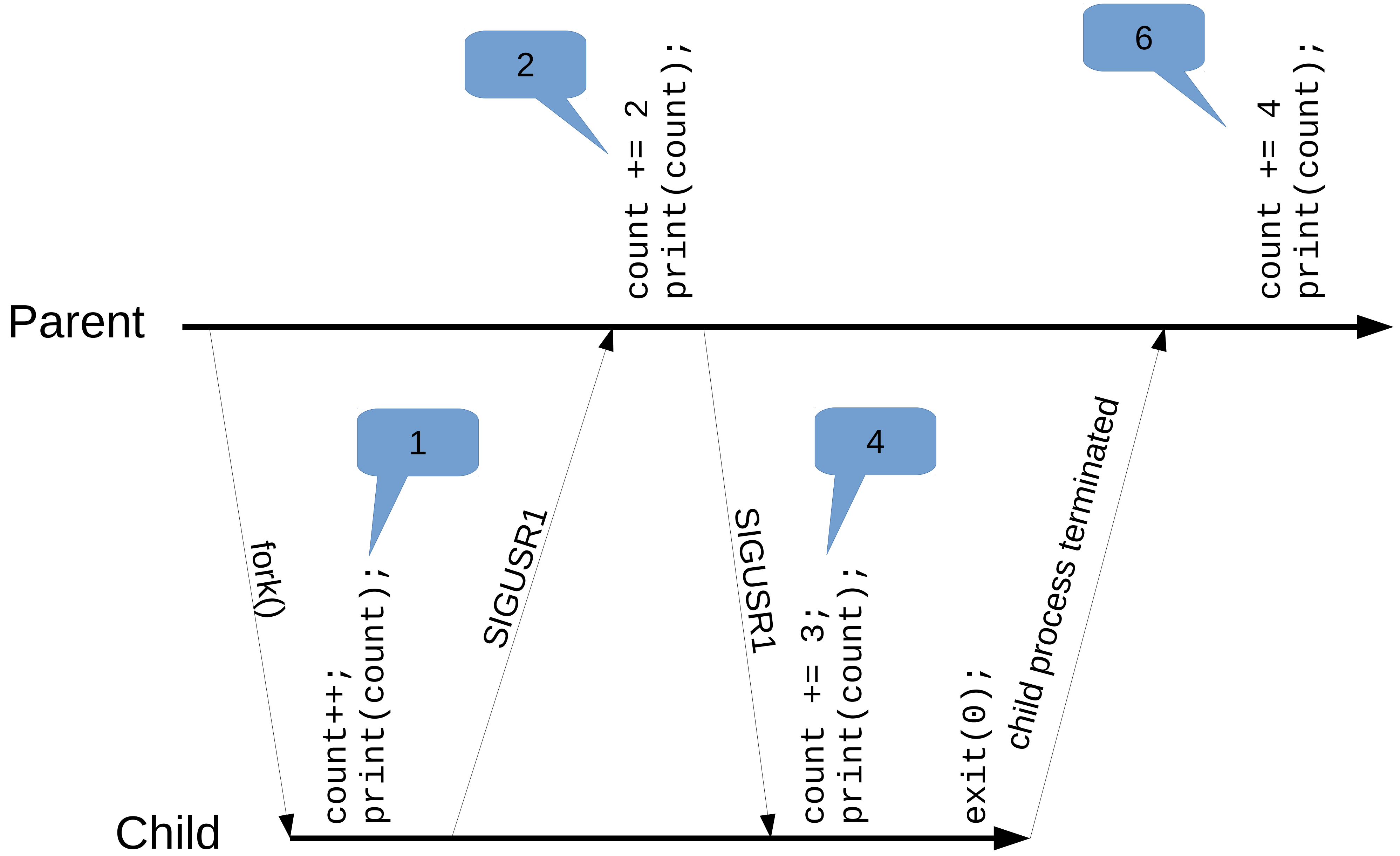
Recitation 4

Question 3:
Flying off the handle

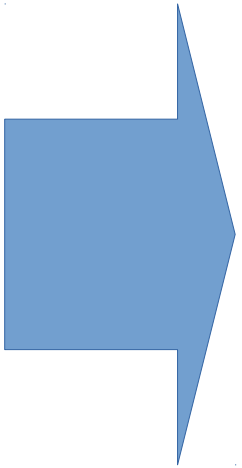


Parent
Child
fork()
SIGUSR1
SIGUSR1
child process terminated
1
2
4
6
count++;
print(count);
count += 2
print(count);
count += 3;
print(count);
exit(0);
count += 4
print(count);

Question 4:
Nice Threads

Atomic?

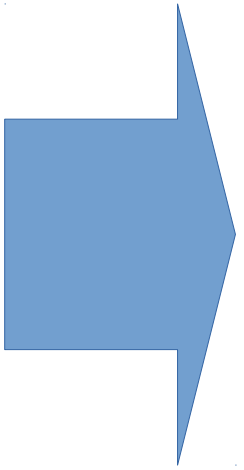
x++



```
reg = x;   //load()
reg = reg + 1; //update
x   = reg;  //store
```

Atomic?

```
x += k
```



```
reg = x;   //load()
reg = reg + k; //inc
x   = reg;  //store
```

Concurrency
What could possibly go wrong?!

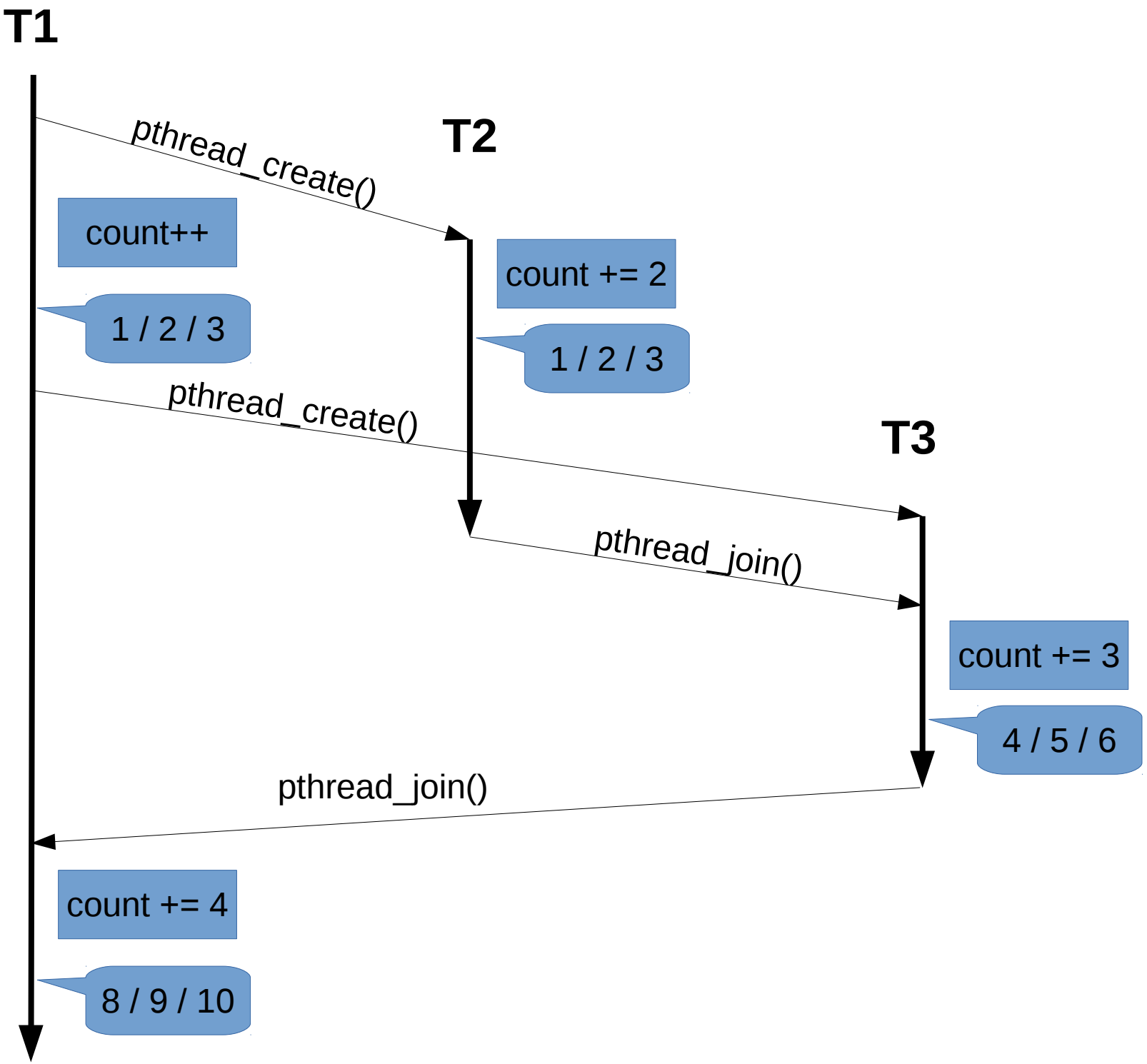
Thread 1

```
x += 1
```

Thread 2

```
x += 2
```

```
T1: reg_1 = load(x)

T2: reg_2 = load(x)

T1: reg_1 = reg_1 + 1

T2: reg_2 = reg_2 + 2

T2: store(x, reg_2)

T1: store(x, reg_1)
```


T1
T2
T3
pthread_create()
count++
1 / 2 / 3
count += 2
1 / 2 / 3
pthread_create()
pthread_join()
count += 3
4 / 5 / 6
pthread_join()
count += 4
8 / 9 / 10

Question 6:
The Semaphore that almost could

Binary Sempahores

```
void binary_P(s)
{
  if(s->count == 0)
  {
    // wait…
  }
  s->count = 0;
}
```

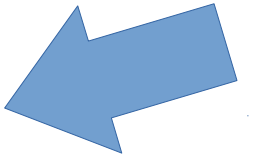
```
void binary_V(s)
{
  if(s->count == 0)
     s->count = 1;
//else
//   do nothing
}
```

```
binary_semaphore mutex = 1;
binary_semaphore delay = 0;
int C = {initvalue};

void counting_ P() {
     binary_P(mutex);
     C = C-1;
     if (C < 0) {
        binary_V(mutex);
        binary_P(delay);
     } else {
        binary_V(mutex);
     }
}
void counting_V() {
     binary_P(mutex);
     C = C+1;
     if (C <= 0)
        binary_V(delay);
     binary_V(mutex);
}
```

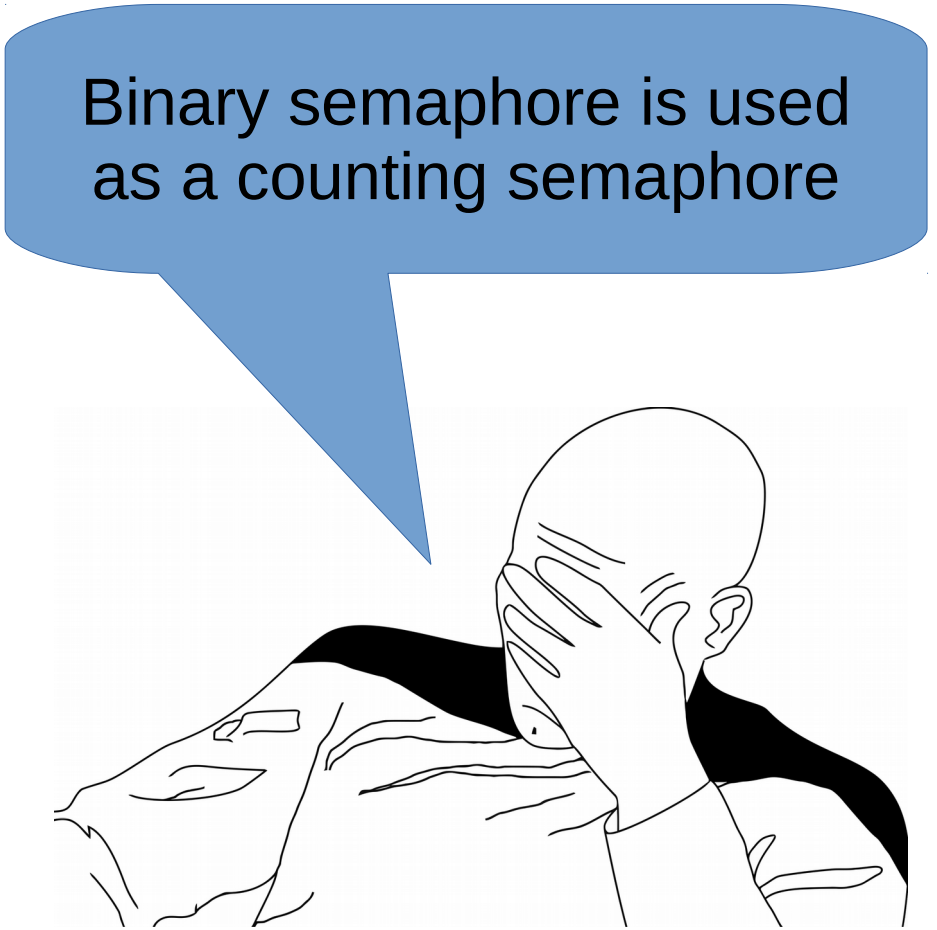
```
void consumer()
{
    counting_P();
}

void producer()
{
    counting_V();
    counting_V();
}

void main()
{
    start_thread(consumer);
    start_thread(consumer);
    start_thread(producer);
}
```

```
binary_semaphore mutex = 1;
binary_semaphore delay = 0;
int C = {initvalue};

void counting_ P() {
     binary_P(mutex);
     C = C-1;
     if (C < 0) {
        binary_V(mutex);
        binary_P(delay);
     } else {
        binary_V(mutex);
     }
}
void counting_V() {
     binary_P(mutex);
     C = C+1;
     if (C <= 0)
        binary_V(delay);
     binary_V(mutex);
}
```

```
T1: binary_P(mutex)
T1: C ← -1
T1: binary_V(mutex)
T1: binary_P(delay)
T2: binary_P(mutex)
T2: C ← -2
T2: binary_V(mutex)
T2: binary_P(delay)
T3: binary_P(mutex)
T3: C ← -1
T3: binary_V(delay)
T3: binary_V(mutex)
T3: binary_P(mutex)
T3: C ← 0
T3: binary_V(delay)
T3: binary_V(mutex)
```



```
binary_semaphore mutex = 1;
binary_semaphore delay = 0;
int C = {initvalue};

void counting_ P() {
     binary_P(mutex);
     C = C-1;
     if (C < 0) {
        binary_V(mutex);
        binary_P(delay);
     } else {
        binary_V(mutex);
     }
}
void counting_V() {
     binary_P(mutex);
     C = C+1;
     if (C <= 0)
        binary_V(delay);
     binary_V(mutex);
}
```



Jean-Luc Picard
– Senior Software Developer

One way to do it correctly

```
binary_semaphore mutex;
int count = {initvalue};
stack<thread> waiting;

void counting_P() {
   mutex.lock();
   if (waiting.size() > 0) {
      t = waiting.pop();
      t.start();
   } else {
      count += 1;
   }
   mutex.unlock();
}
```

```
void counting_V() {
     mutex.lock();
     if(count > 0) {
        count -= 1;
        mutex.unlock();
     } else {
        waiting.push(self());
        mutex.unlock();
        self.stop();
     }
}
```