Parallel Programming
Exercise Session 4

Spring 2025

Schedule

Post-Discussion Ex. 3

Theory

Pre-Discussion Ex. 4

Quiz

Post-Discussion Exercise 3

Counter

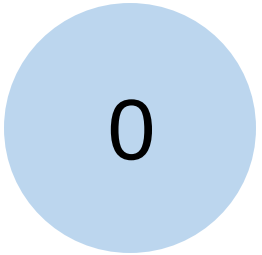
Let’s count number of times a given event occurs

```
public interface Counter {
  public void increment();
  public int value();
}
```

```
// background threads
for (int i = 0; i < numIterations; i++) {
  // perform some work

  counter.increment();
}

// progress thread
while (isWorking) {
  System.out.println(counter.value());
}
```

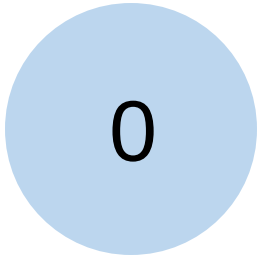
10 iterations each

Counter

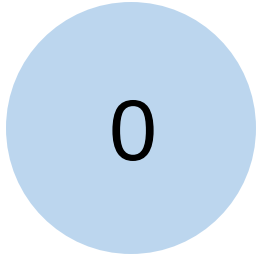


value of the
shared Counter

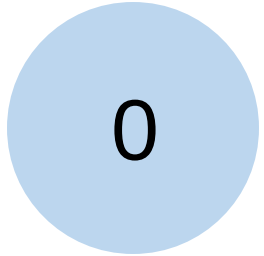
Thread 1



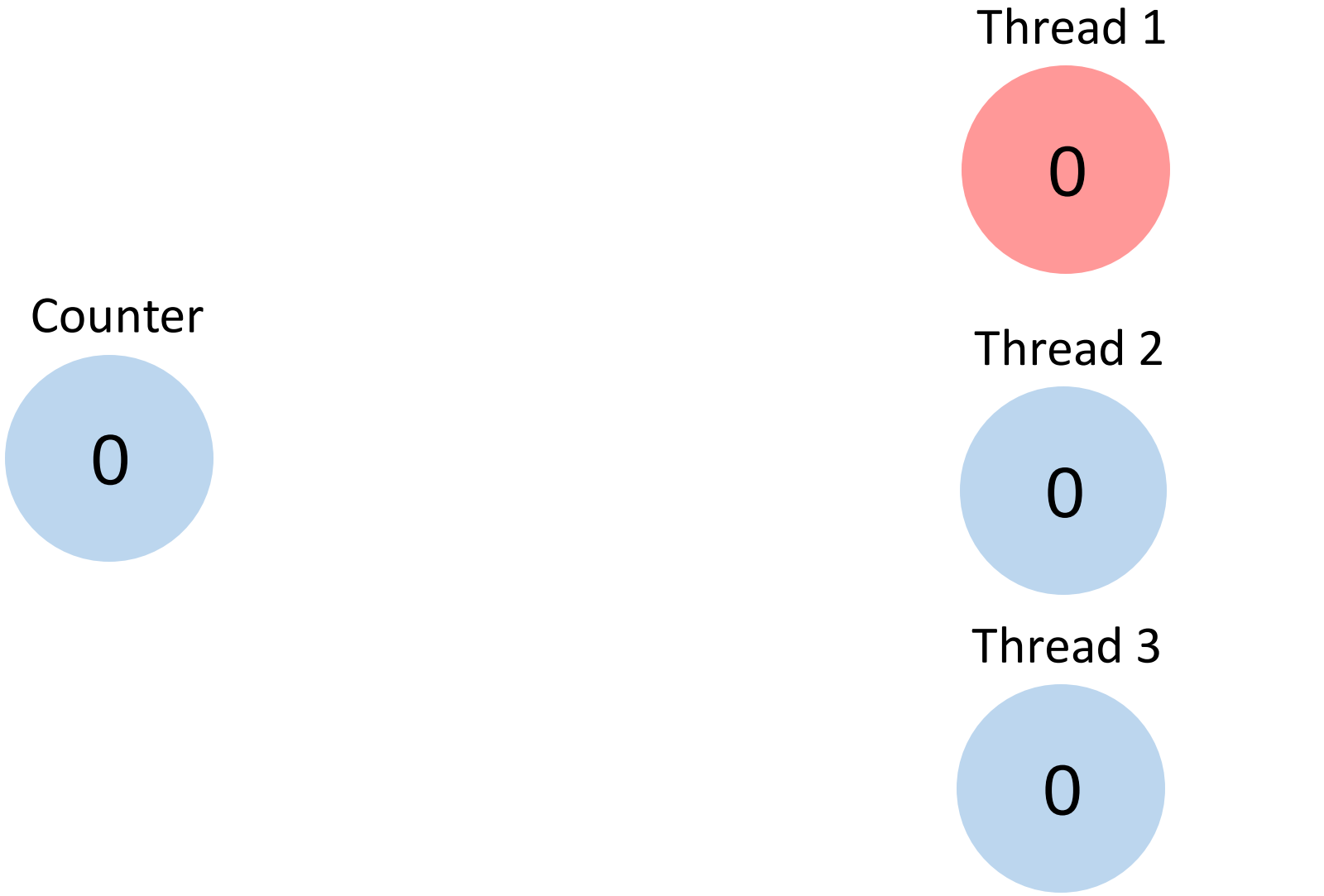
Thread 2



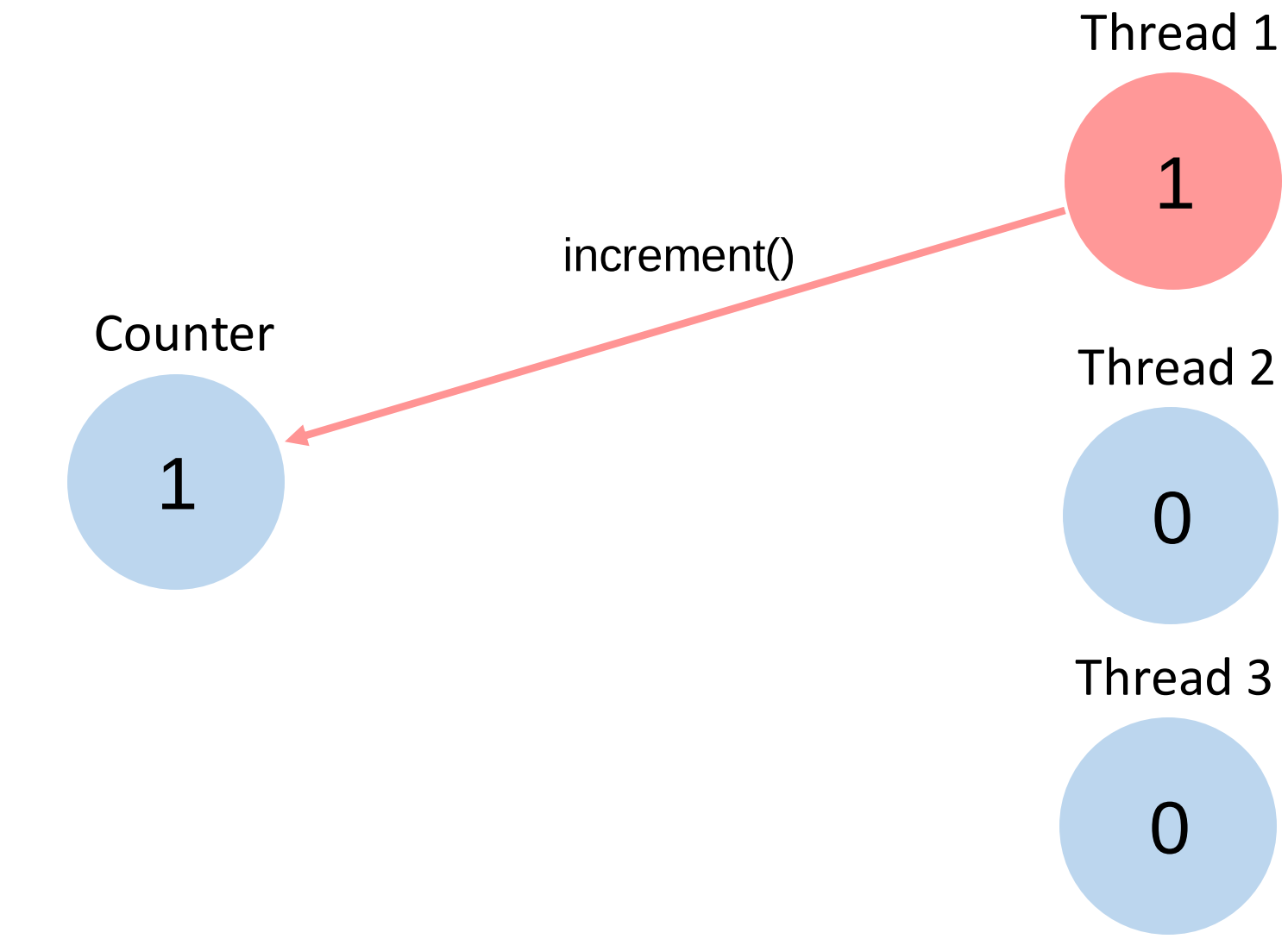
Thread 3



number of times
`increment()` is called

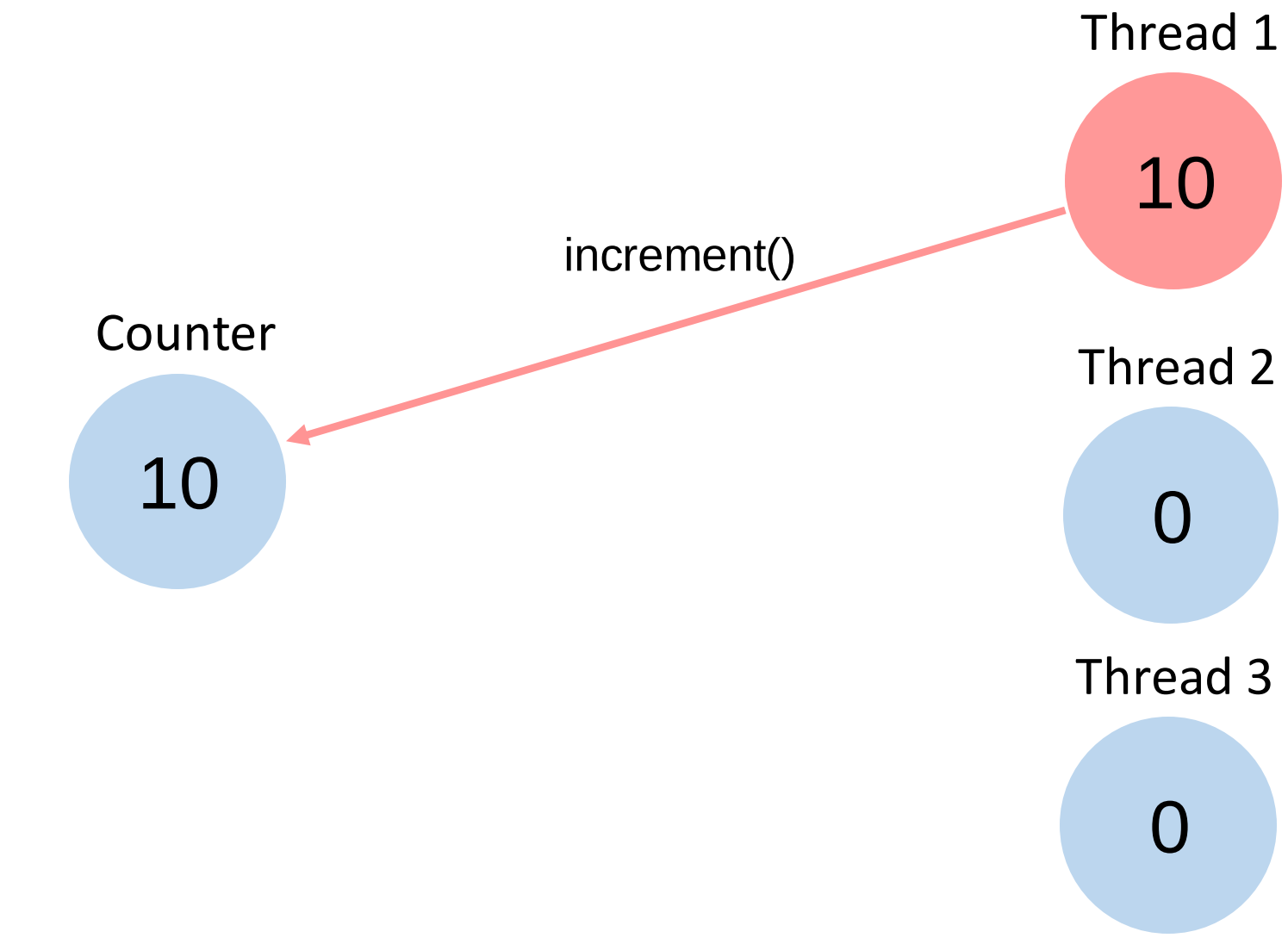

Thread 1
0
Counter
0
Thread 2
0
Thread 3
0
value of the
shared Counter
number of times
increment() is called


Thread 1
1
Counter
0
Thread 2
0
Thread 3
0
value of the
shared Counter
number of times
increment() is called



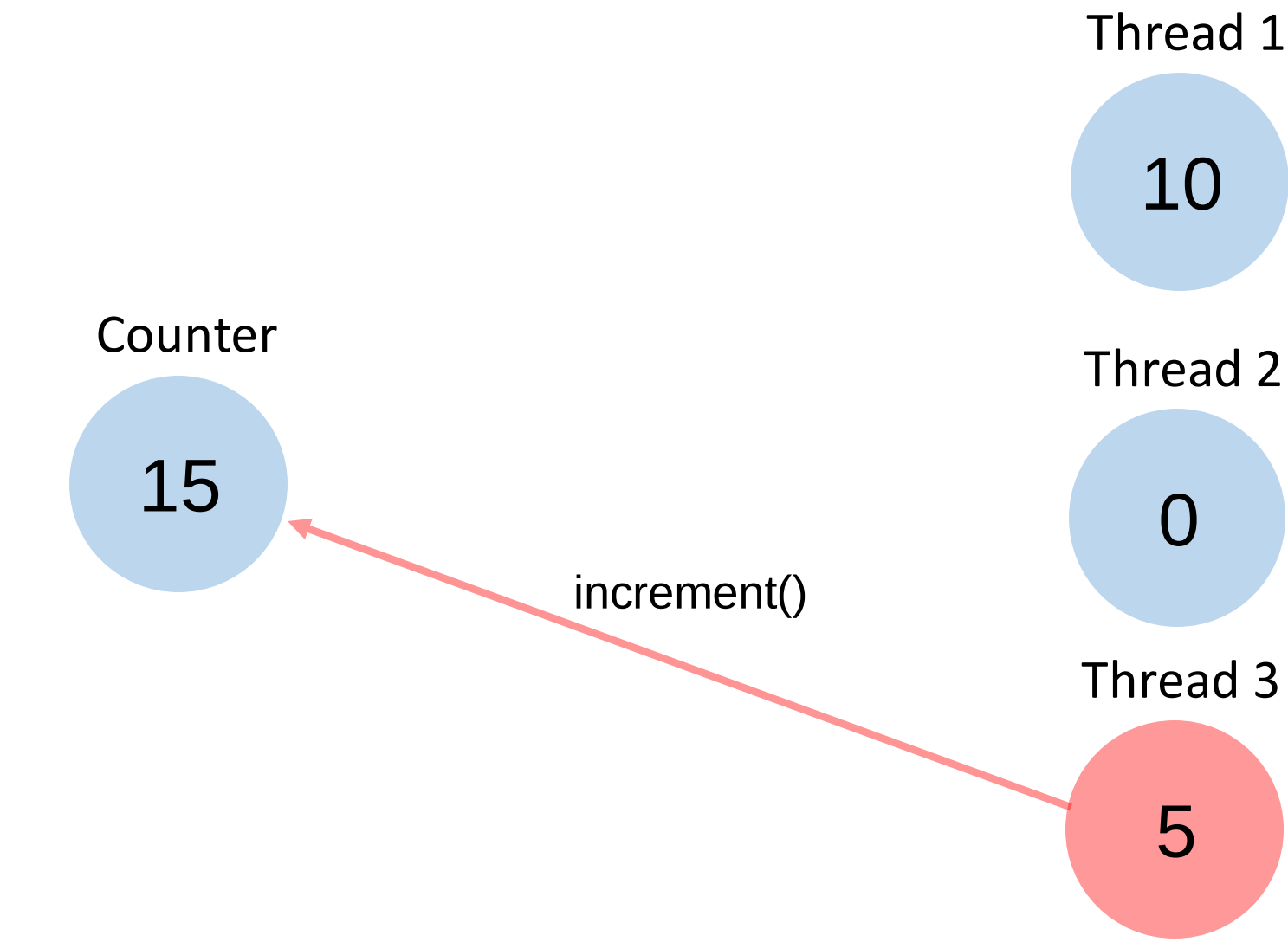
value of the
shared Counter

number of times
`increment()` is called



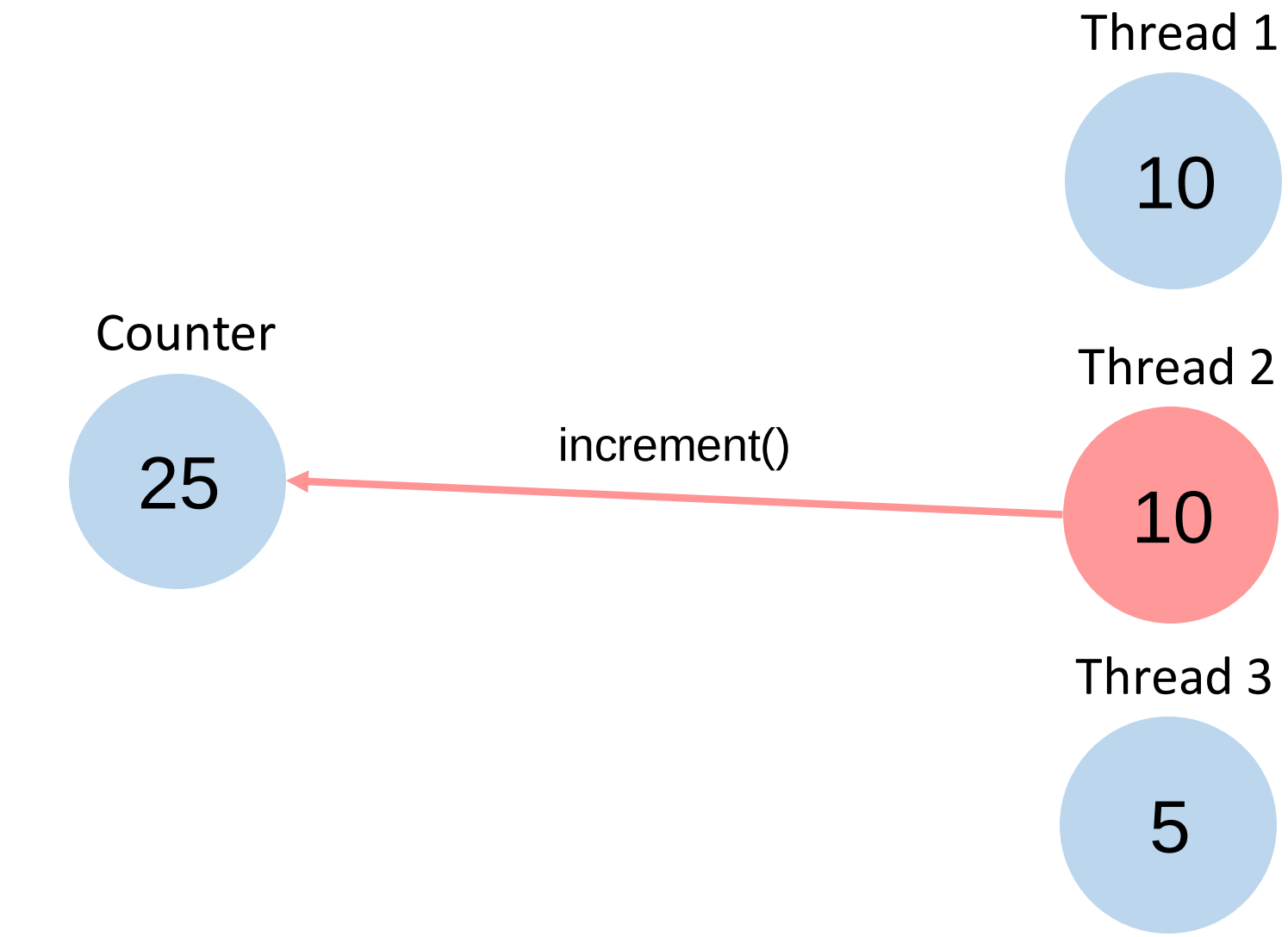
value of the
shared Counter

number of times
`increment()` is called



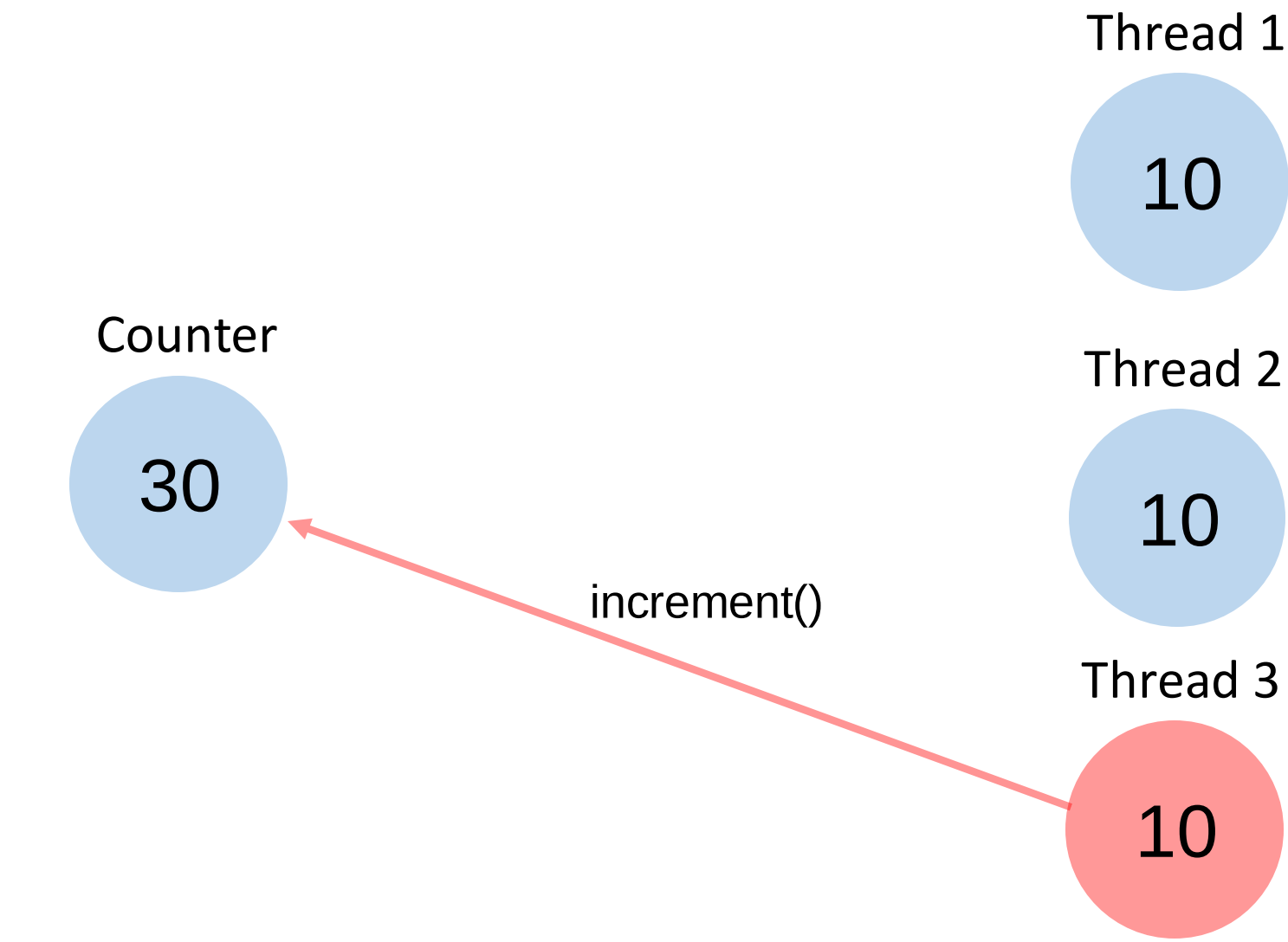
value of the
shared Counter

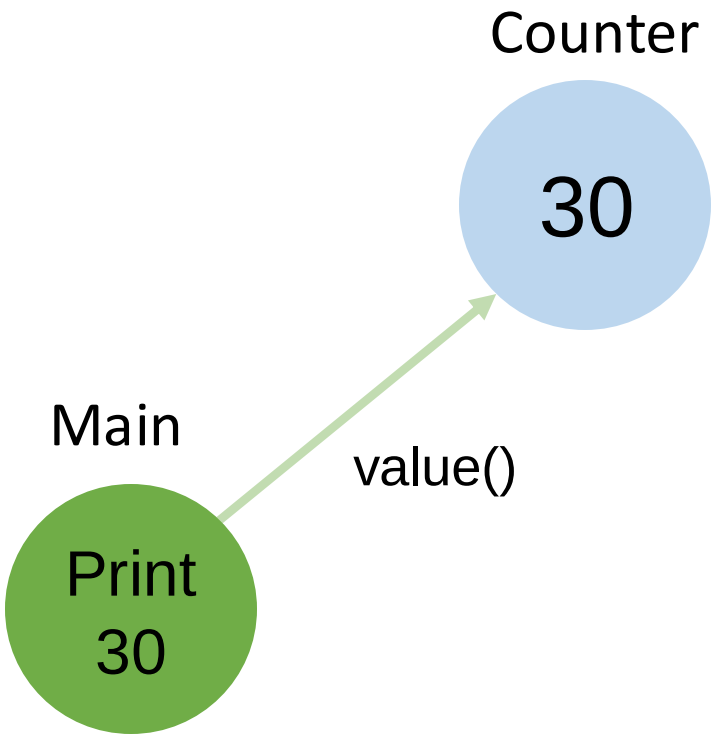
number of times
`increment()` is called



value of the
shared Counter

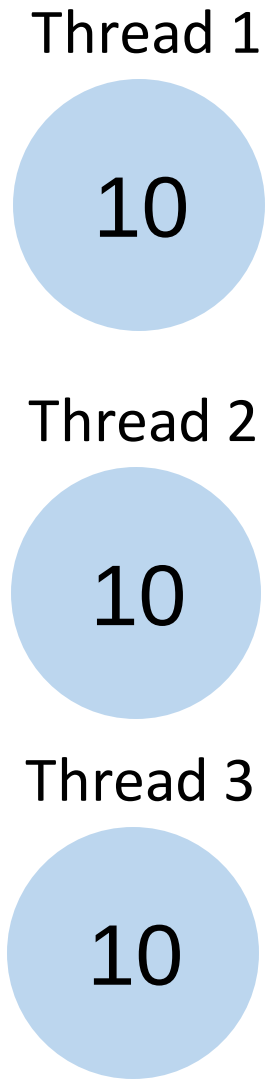
number of times
`increment()` is called


Thread 1
10
Counter
30
Thread 2
10
increment()
Thread 3
10
value of the
shared Counter
number of times
increment() is called



read the Counter value

value of the shared Counter



number of times `increment()` is called

Task A: SequentialCounter

```
public class SequentialCounter implements Counter {


    public void increment() {
        ??
    }

    public int value() {
        ??
    }
}
```

Task A: SequentialCounter

```
public class SequentialCounter implements Counter {
    private int c = 0;

    public void increment() {
        c++;
    }

    public int value() {
        return c;
    }
}
```

Task A: SequentialCounter

Thread 1

0

Counter

0

Thread 2

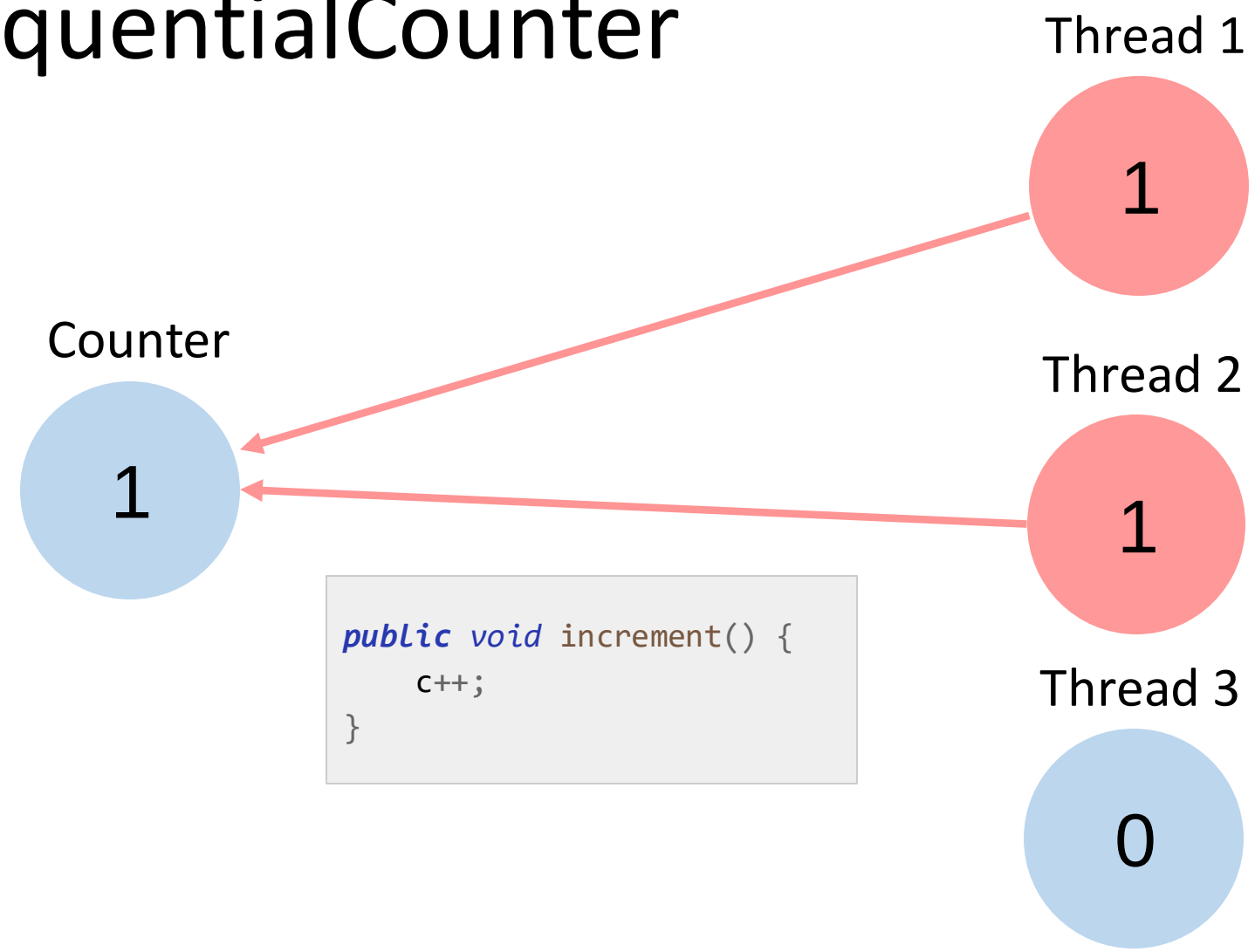
0

```
public void increment() {
    c++;
}
```

Thread 3

0

Task A: SequentialCounter

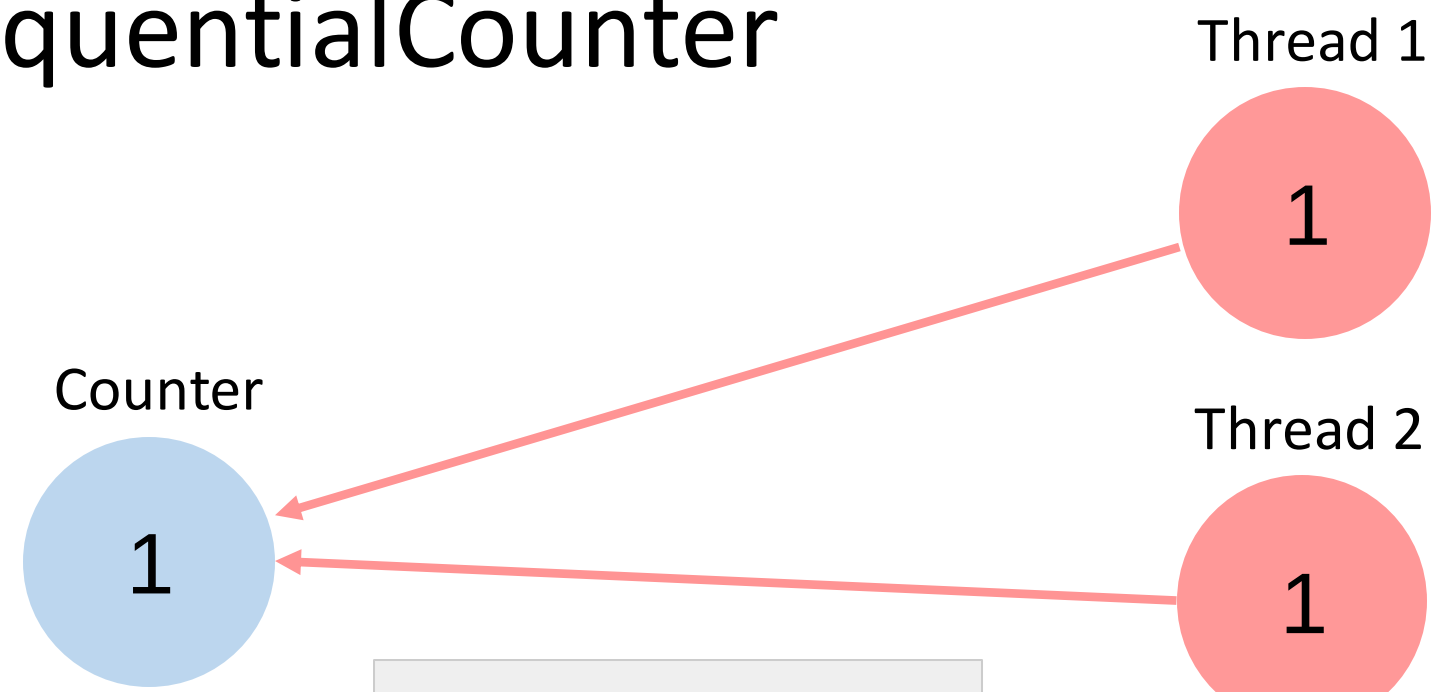


conflicting access!

How is this possible?

```
public void increment() {
    c++;
}
```

Task A: SequentialCounter



conflicting access!

How is this possible?

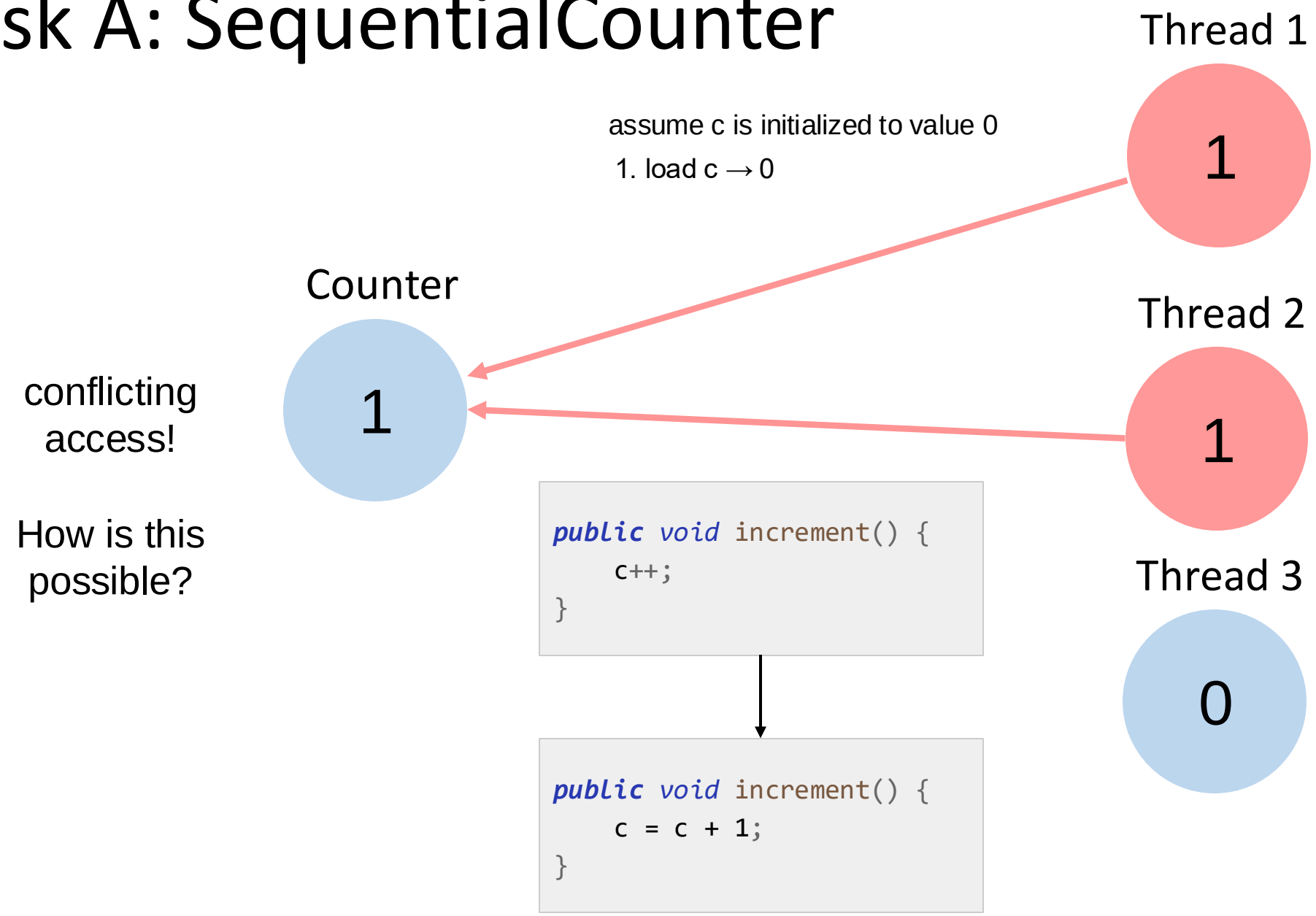
Thread 3

0

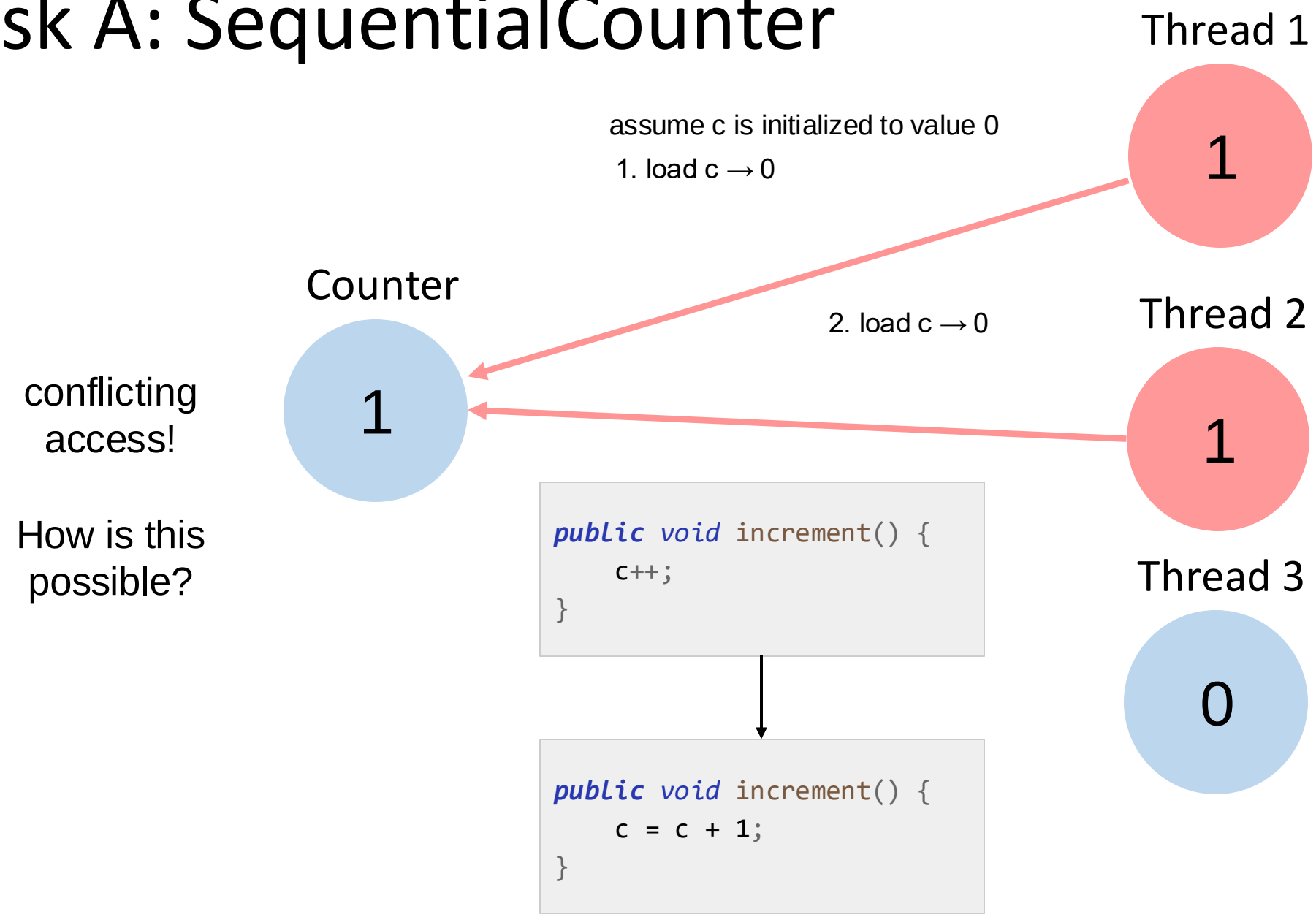
```
public void increment() {
    c++;
}
```

```
public void increment() {
    c = c + 1;
}
```

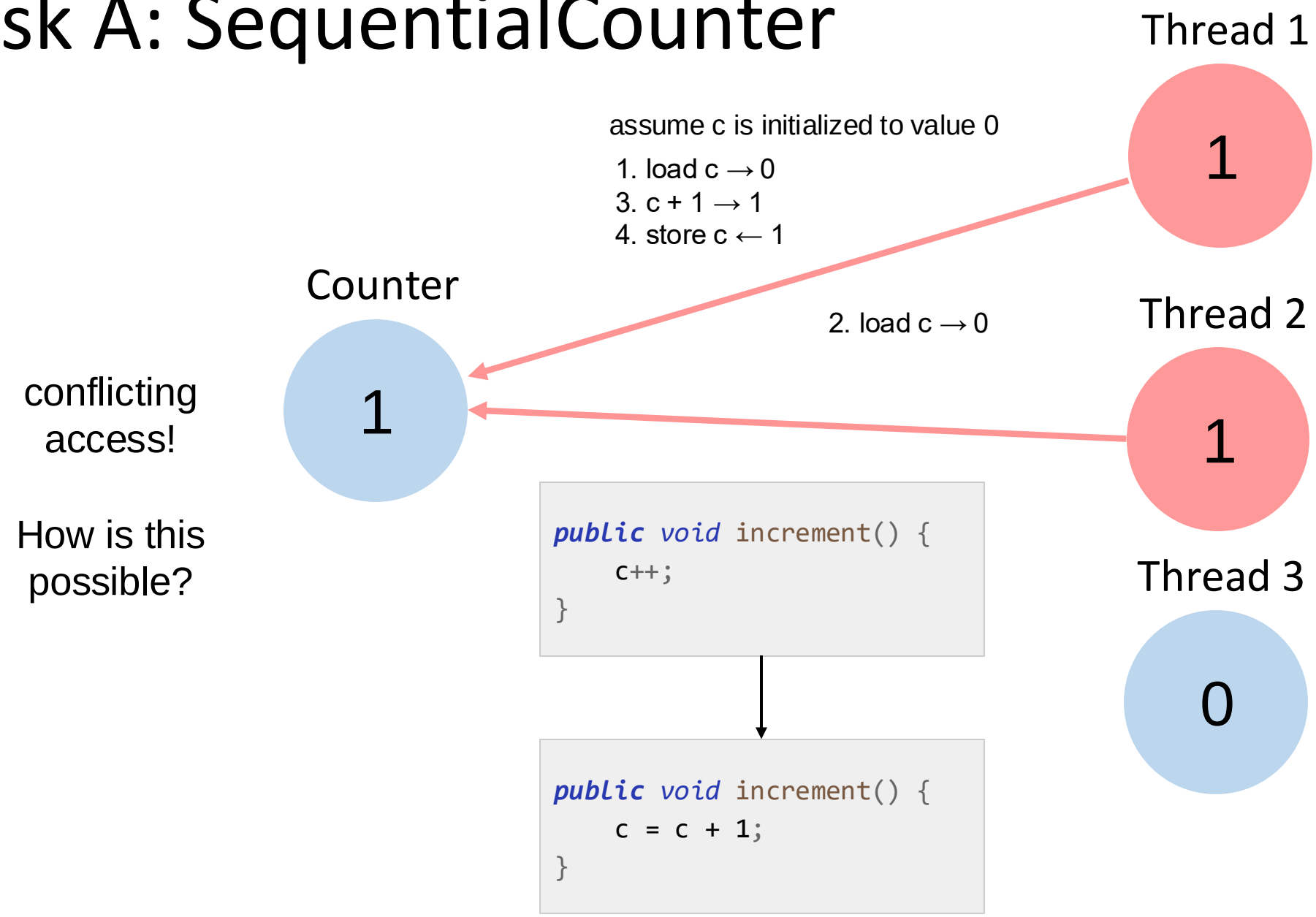
Task A: SequentialCounter



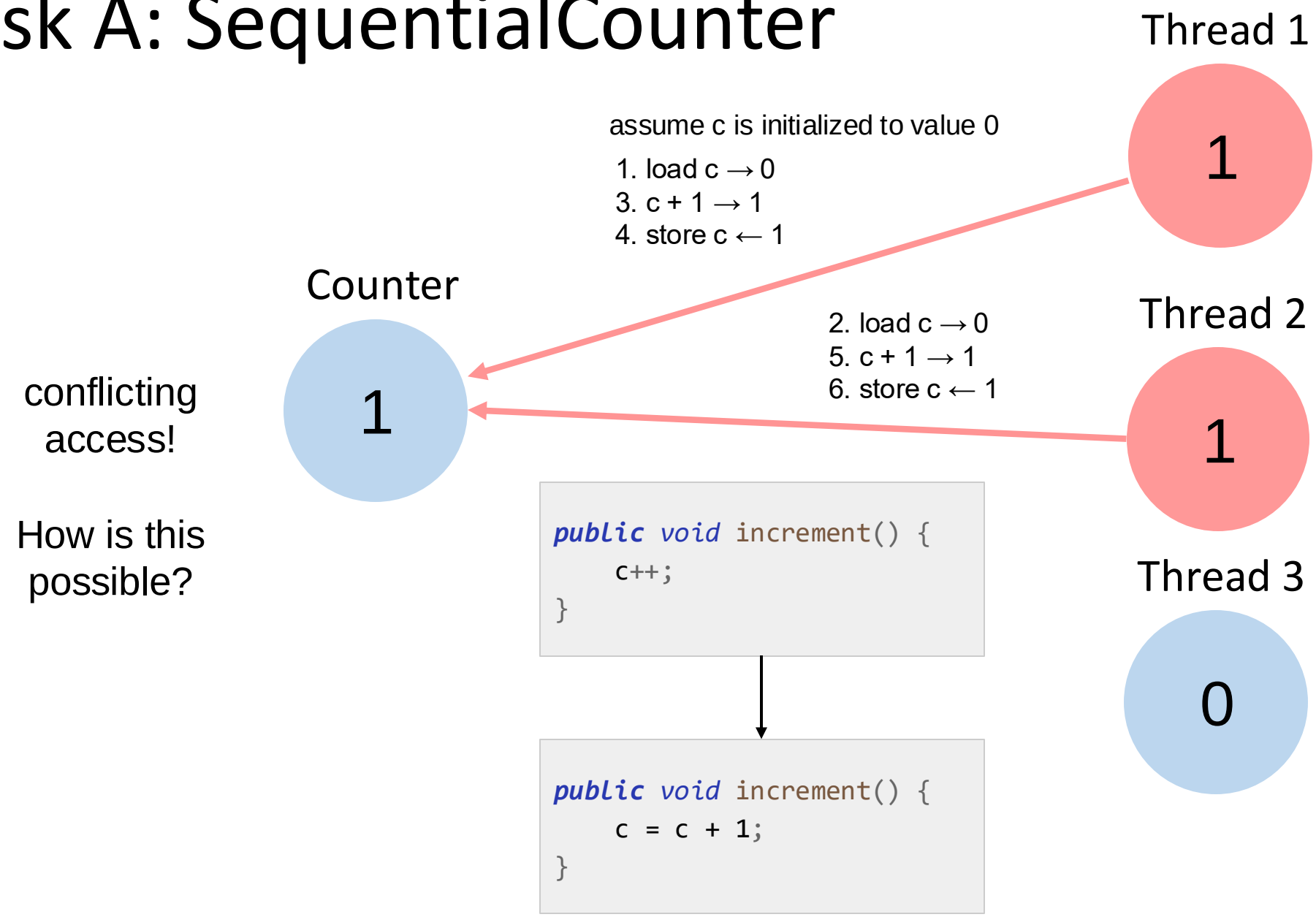
Task A: SequentialCounter



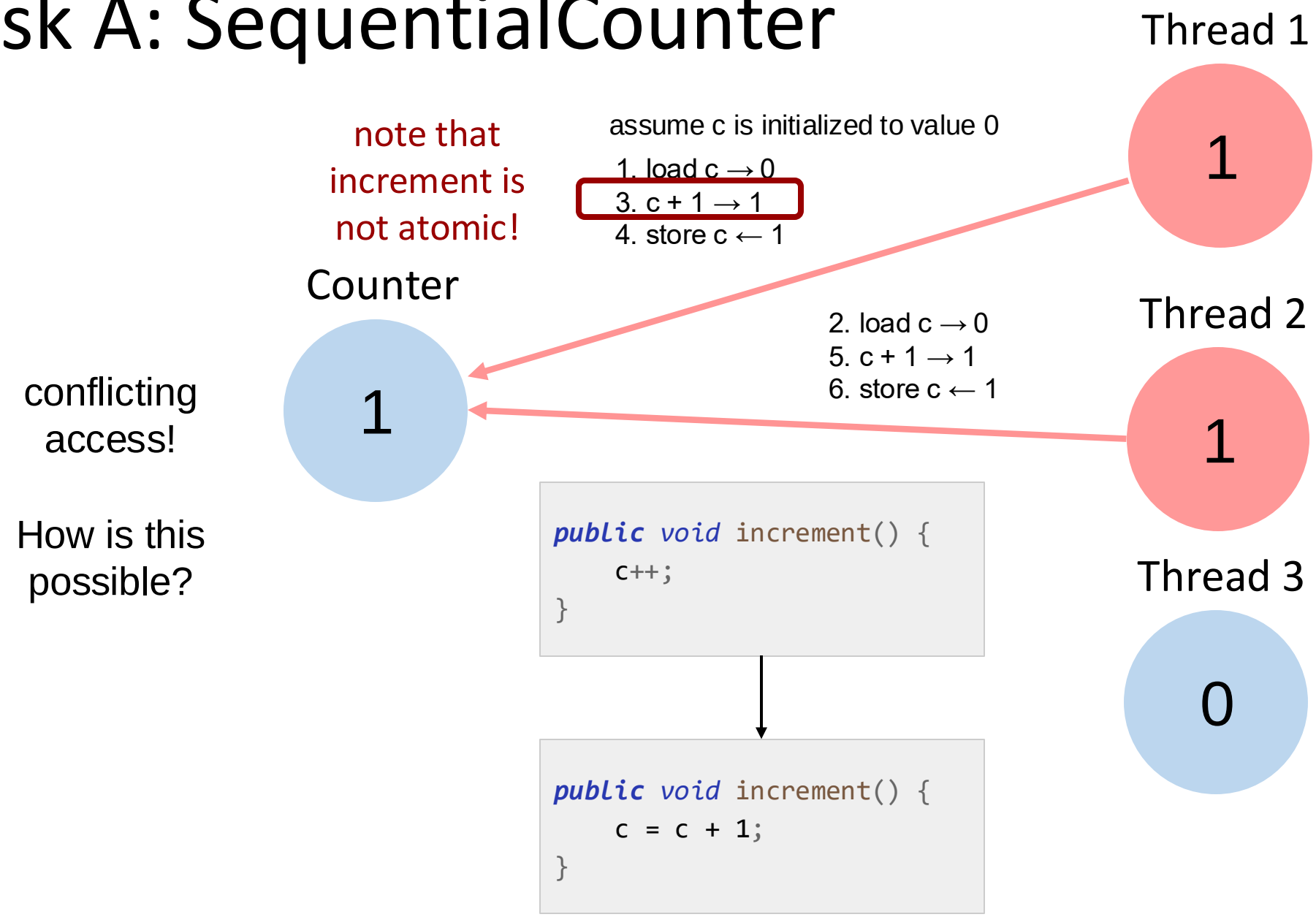
Task A: SequentialCounter



Task A: SequentialCounter



Task A: SequentialCounter



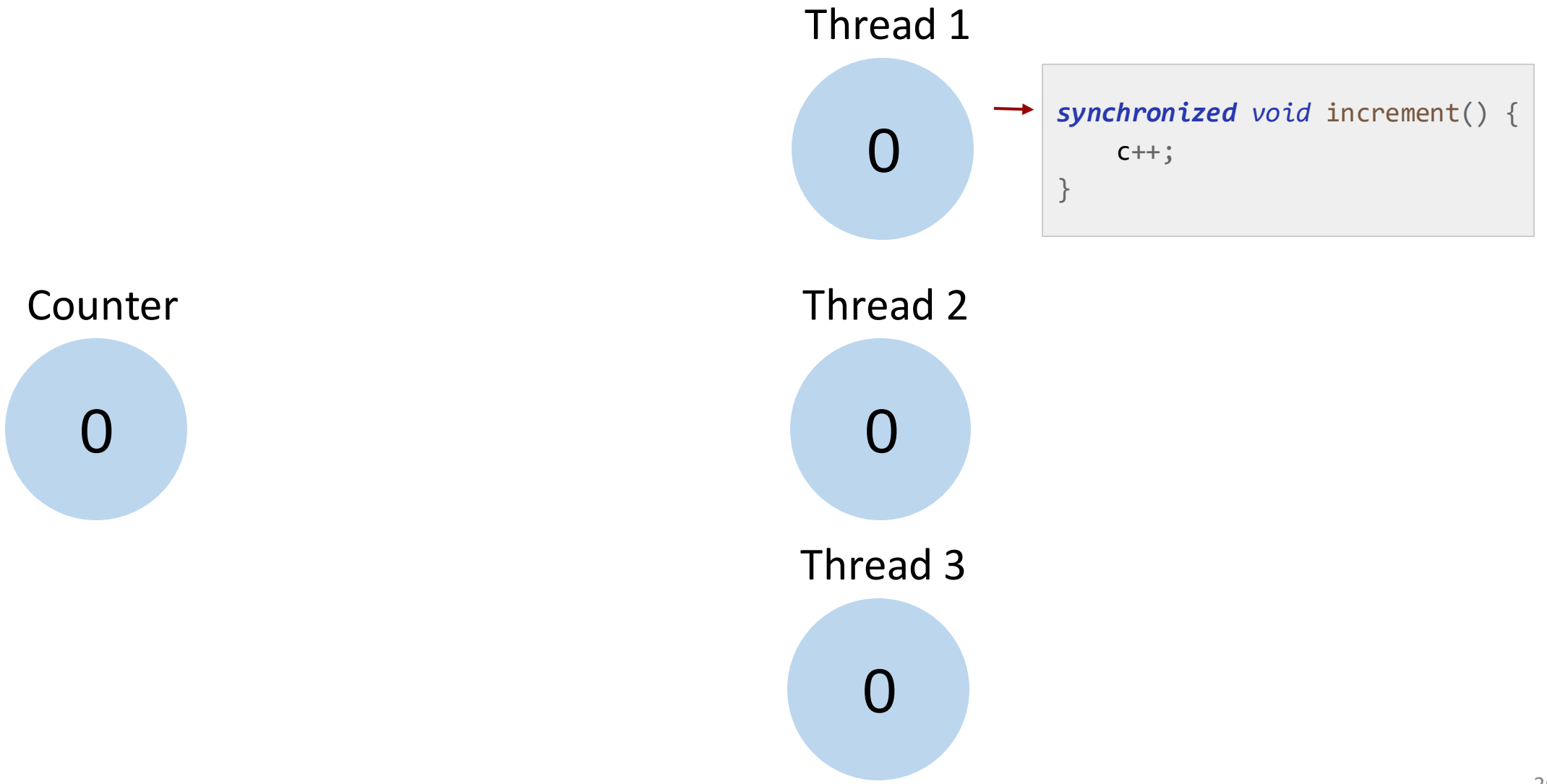
Task B: SynchronizedCounter

```
public class SynchronizedCounter implements Counter {


    public void increment() {
        ??
    }

    public int value() {
        ??
    }
}
```

Task B: SynchronizedCounter

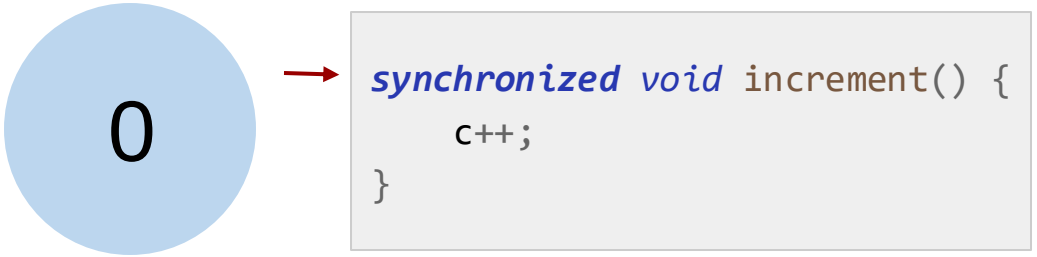
```
public class SynchronizedCounter implements Counter {
    private int c = 0;

    public synchronized void increment() {
        c++;
    }

    public synchronized int value() {
        return c;
    }
}
```

Task B: SynchronizedCounter



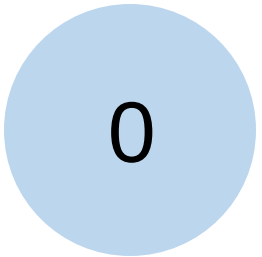
Task B: SynchronizedCounter

Thread 1



Thread 2

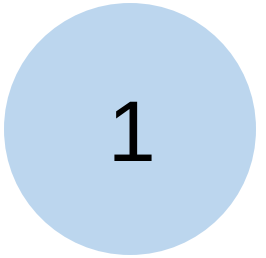
Thread 3



Thread 2 tries to acquire lock on counter. As the lock is already acquired by thread 1 the thread 2 suspends its execution.

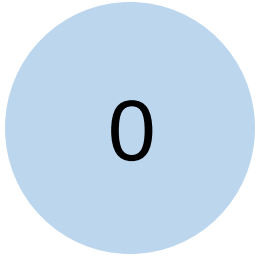
Task B: SynchronizedCounter

Thread 1



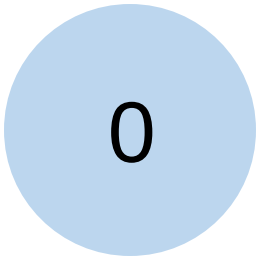

```
synchronized void increment() {
    c++;
}
```

Thread 2



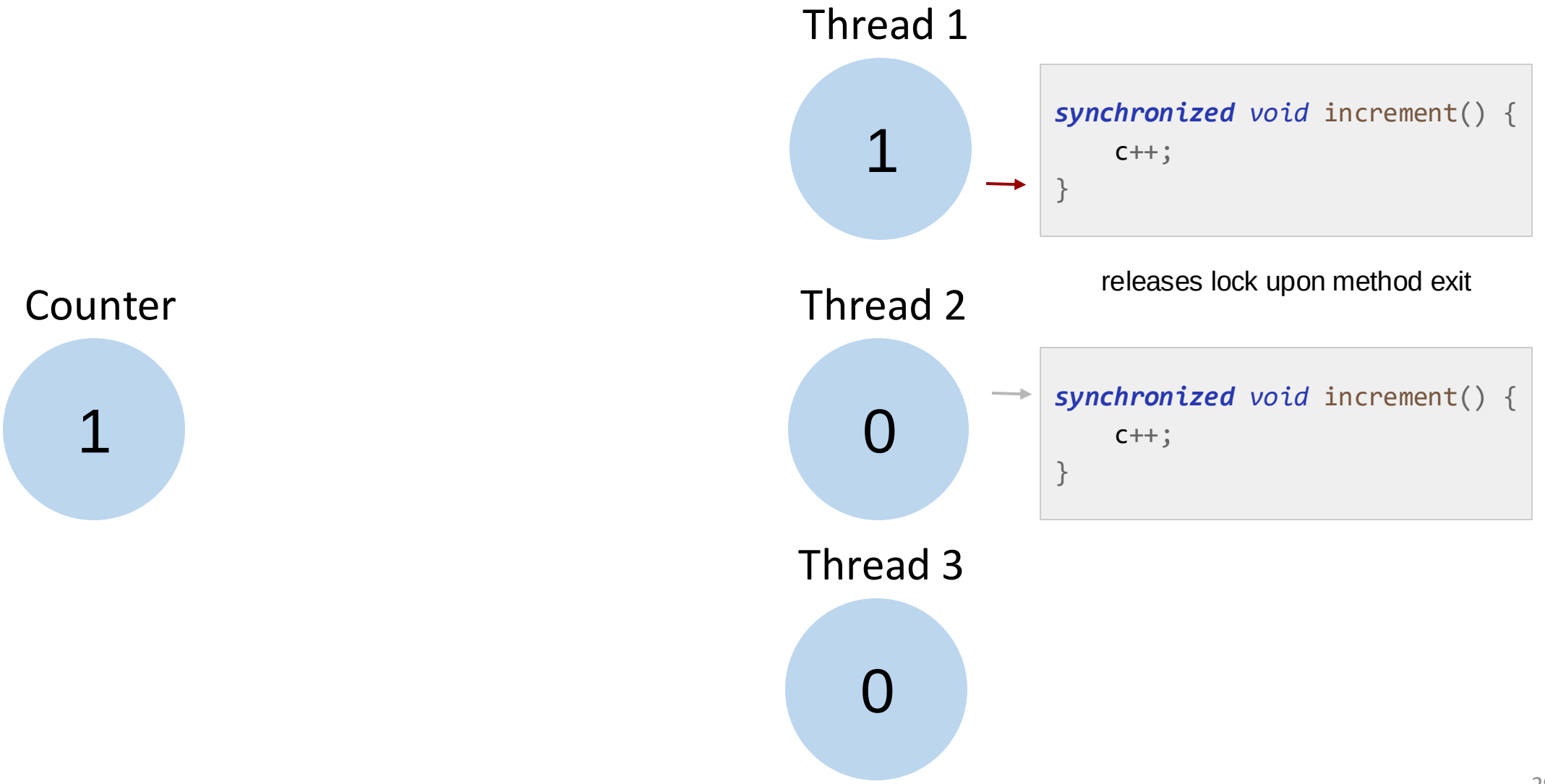

```
synchronized void increment() {
    c++;
}
```

Thread 3

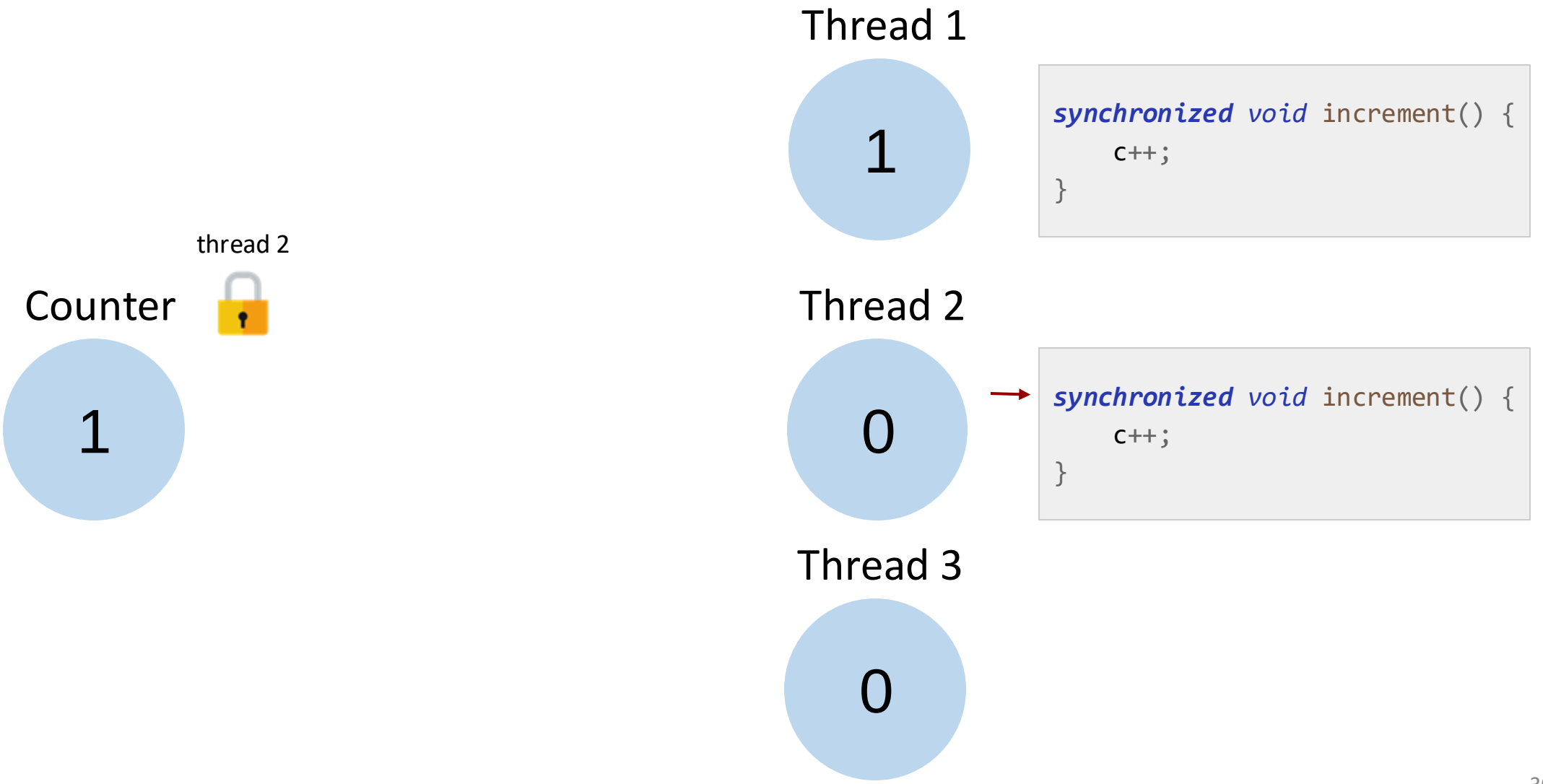


Thread 2 tries to acquire lock on counter. As the lock is already acquired by thread 1 the thread 2 suspends its execution.

Task B: SynchronizedCounter



Task B: SynchronizedCounter



Task C

Task: Print which thread performed the increment. Is there a pattern?

Solution: No, since Java Threads interleave non-deterministic

Task D

- Implement a FairThreadCounter that ensures that different threads increment the Counter in a round-robin fashion. That is, two threads with ids 1 and 2 would increment the value in the following order 1, 2, 1, 2, 1, 2, etc. You should implement the scheduling using the wait and notify methods.
- (Optional) Extend your implementation to work with arbitrary number of threads (instead of only 2) that increment the counter in round-robin fashion.

Wait and Notify Recap

Object (lock) provides `wait` and `notify` methods

(any object is a lock)

`wait`: Thread must own object's lock to call `wait`
thread releases lock and is added to "waiting list" for that object
thread waits until `notify` is called on the object

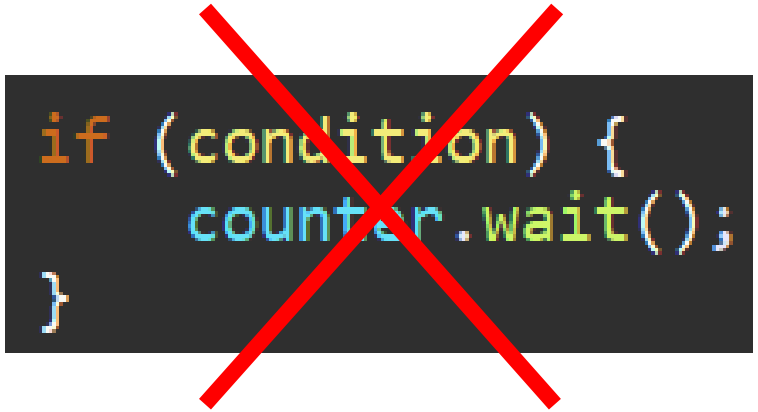
`notify`: Thread must own object's lock to call `notify`

`notify`: Wake one (arbitrary) thread from object's "waiting list"

`notifyAll`: Wake all threads

Wait and Notify Recap

```
while (condition) {
    counter.wait();
}
```



Spurious wake-ups and notifyAll()
→ `wait` has to be in a `while` loop

Spurious Wake-ups

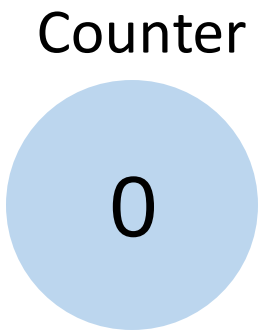
A waiting thread is woken up without being **explicitly** notified through `notify` or `notifyAll`.

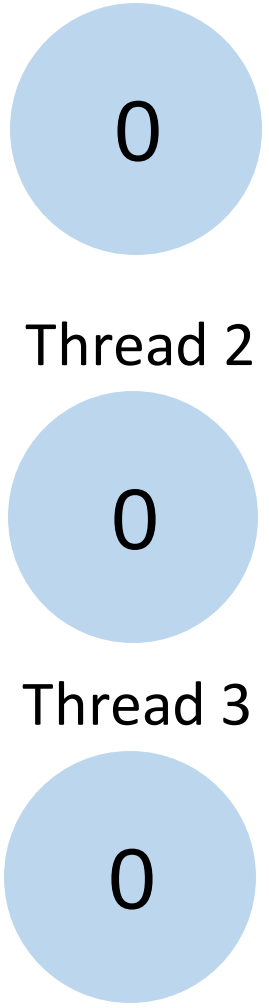
Reasons: **various**

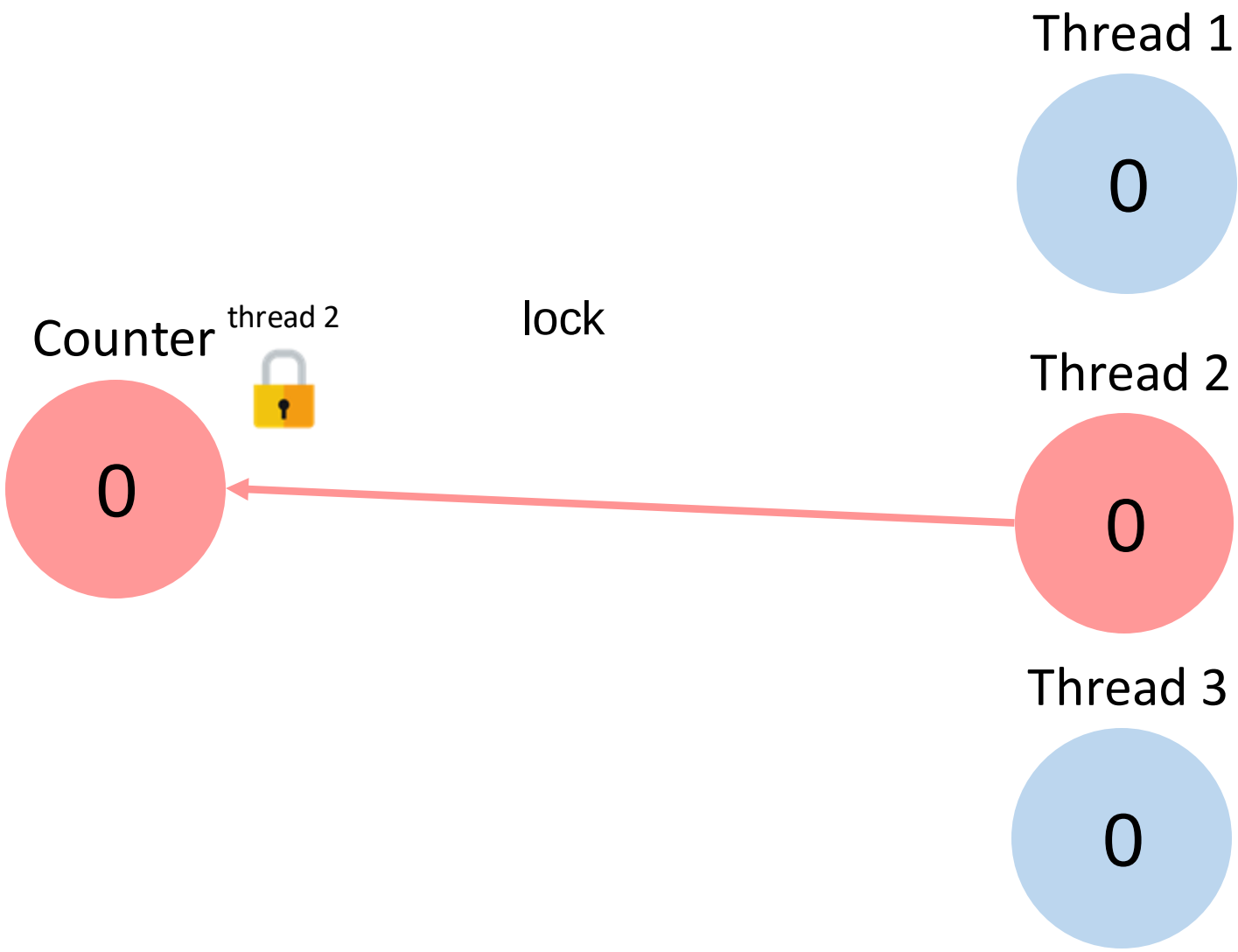
- JVM Implementation: one thread is “nudged for housekeeping”
- Performance (OS dependent): to avoid unnecessary context switches, to rebalance CPU load

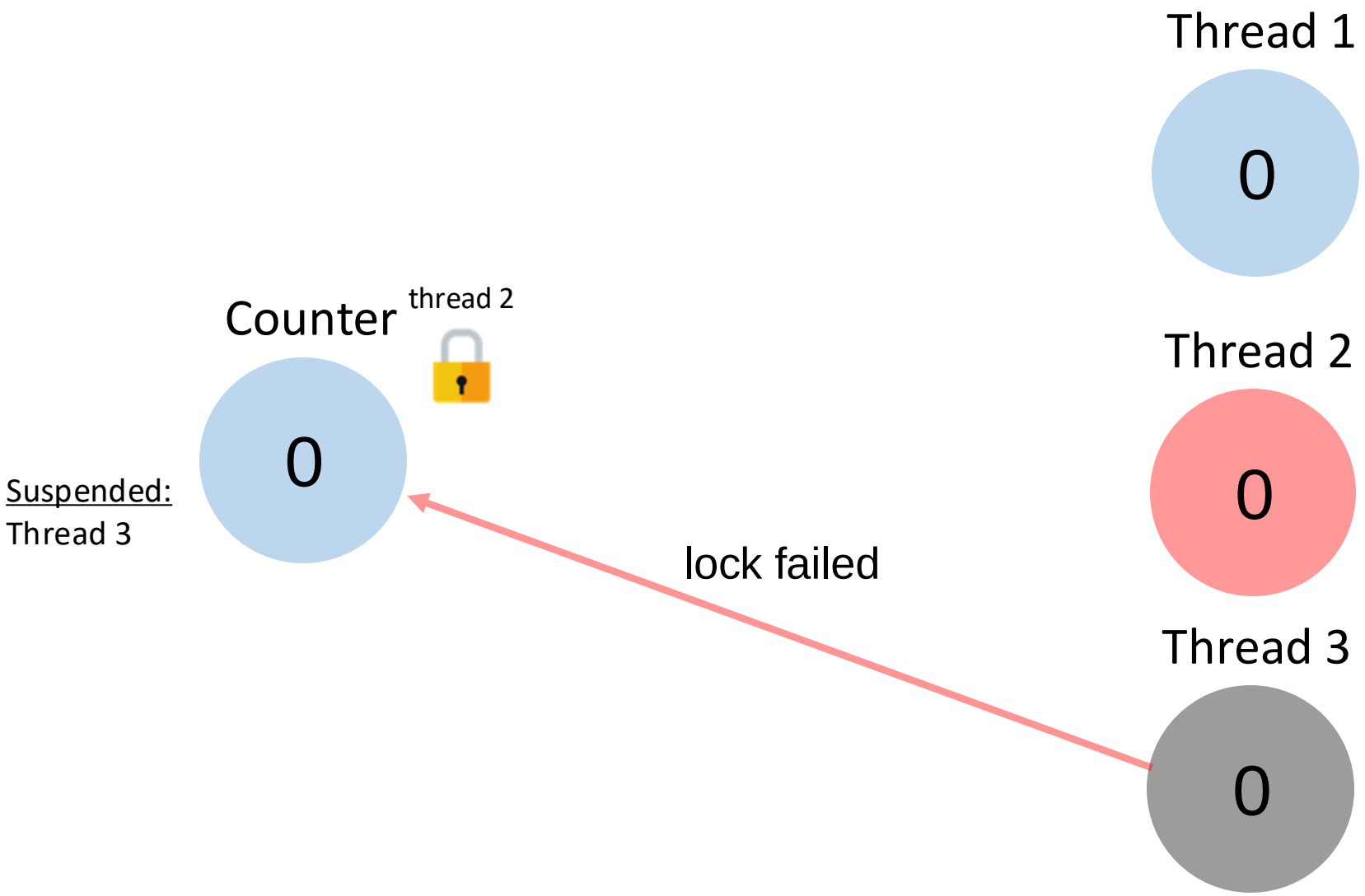
Java does not prevent them. It expects you to handle them.

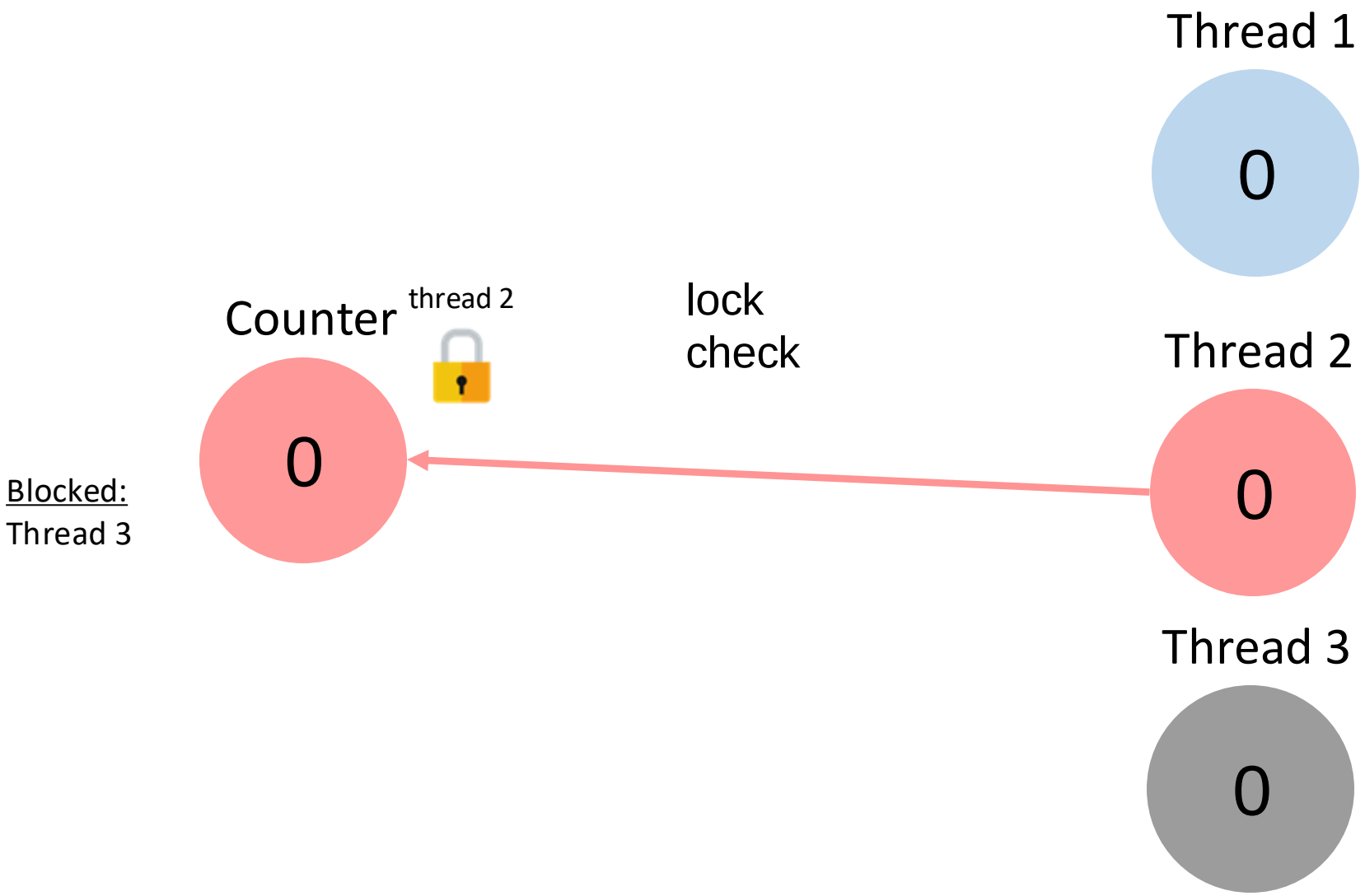
Thread 1 must increment first!

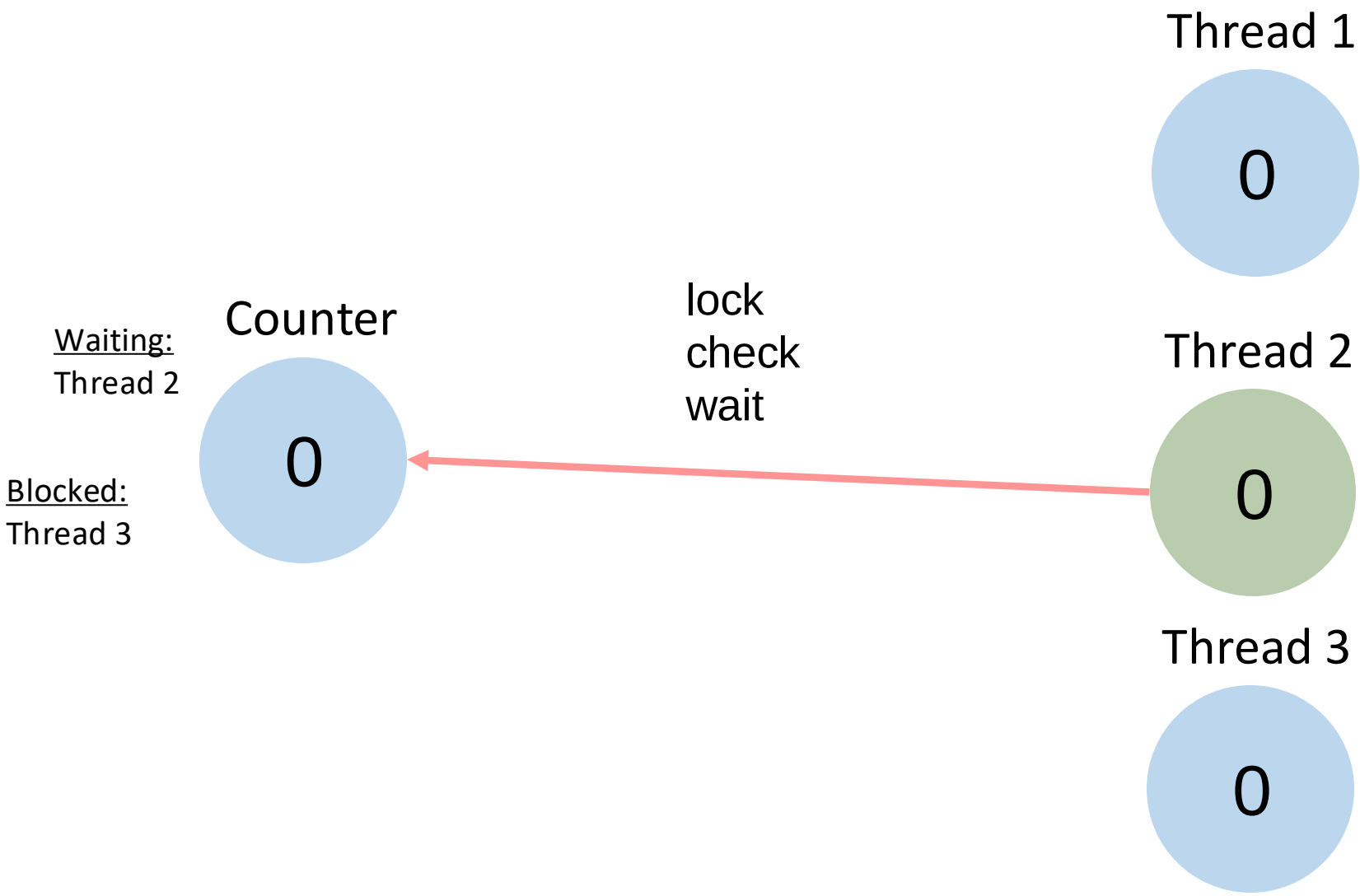


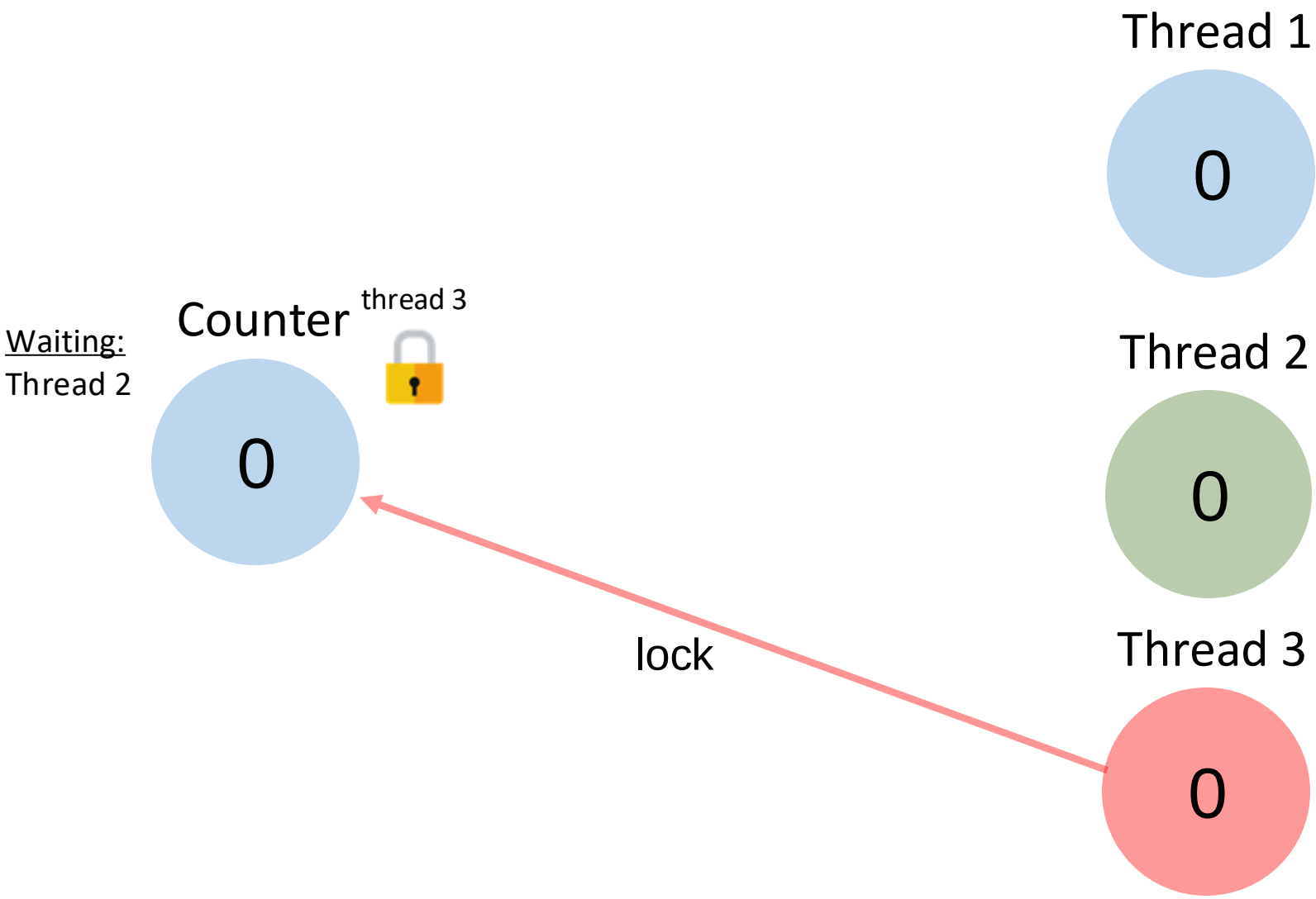


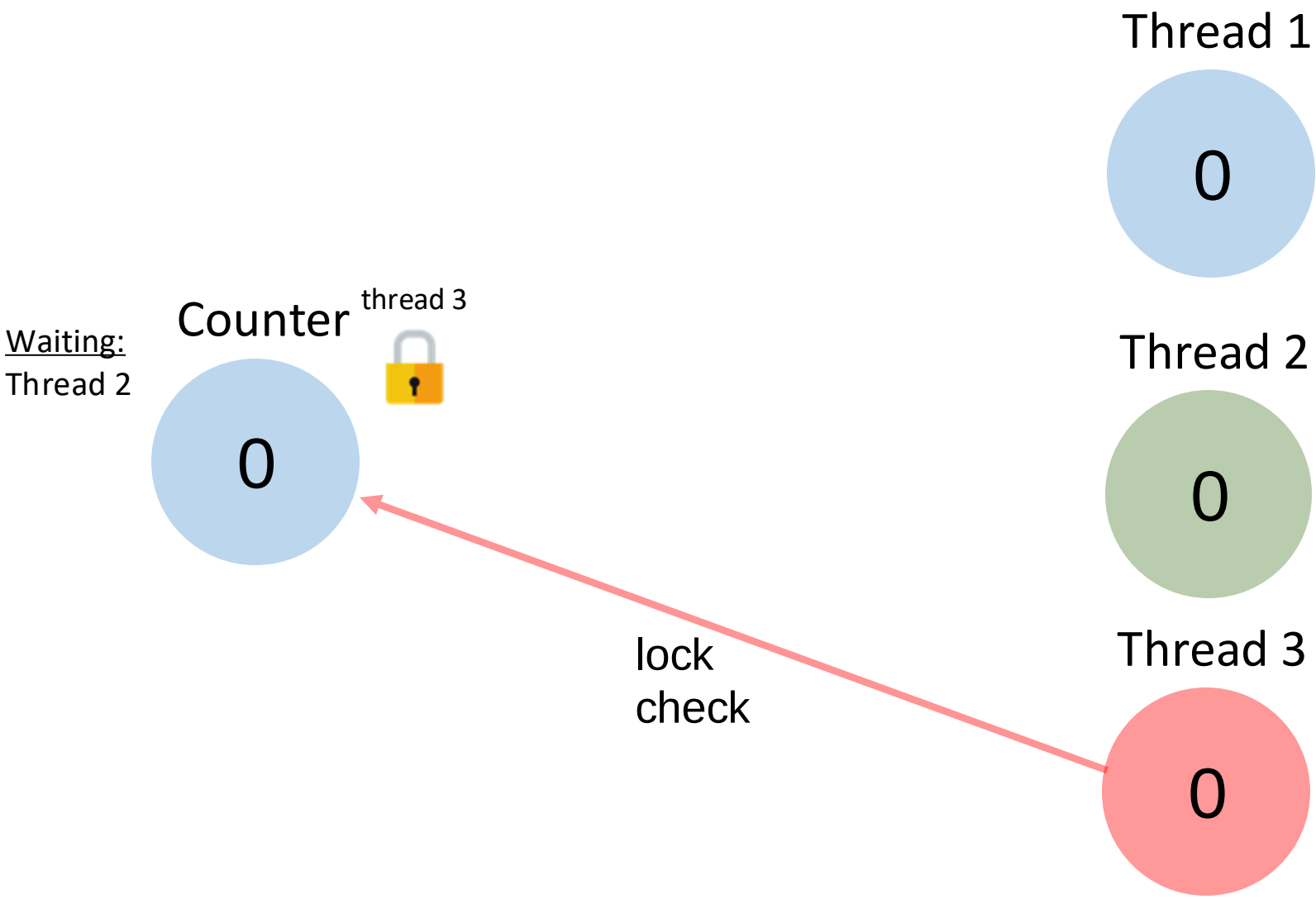

Thread 1
0
Counter
thread 2
lock
Thread 2
0
0
Thread 3
0

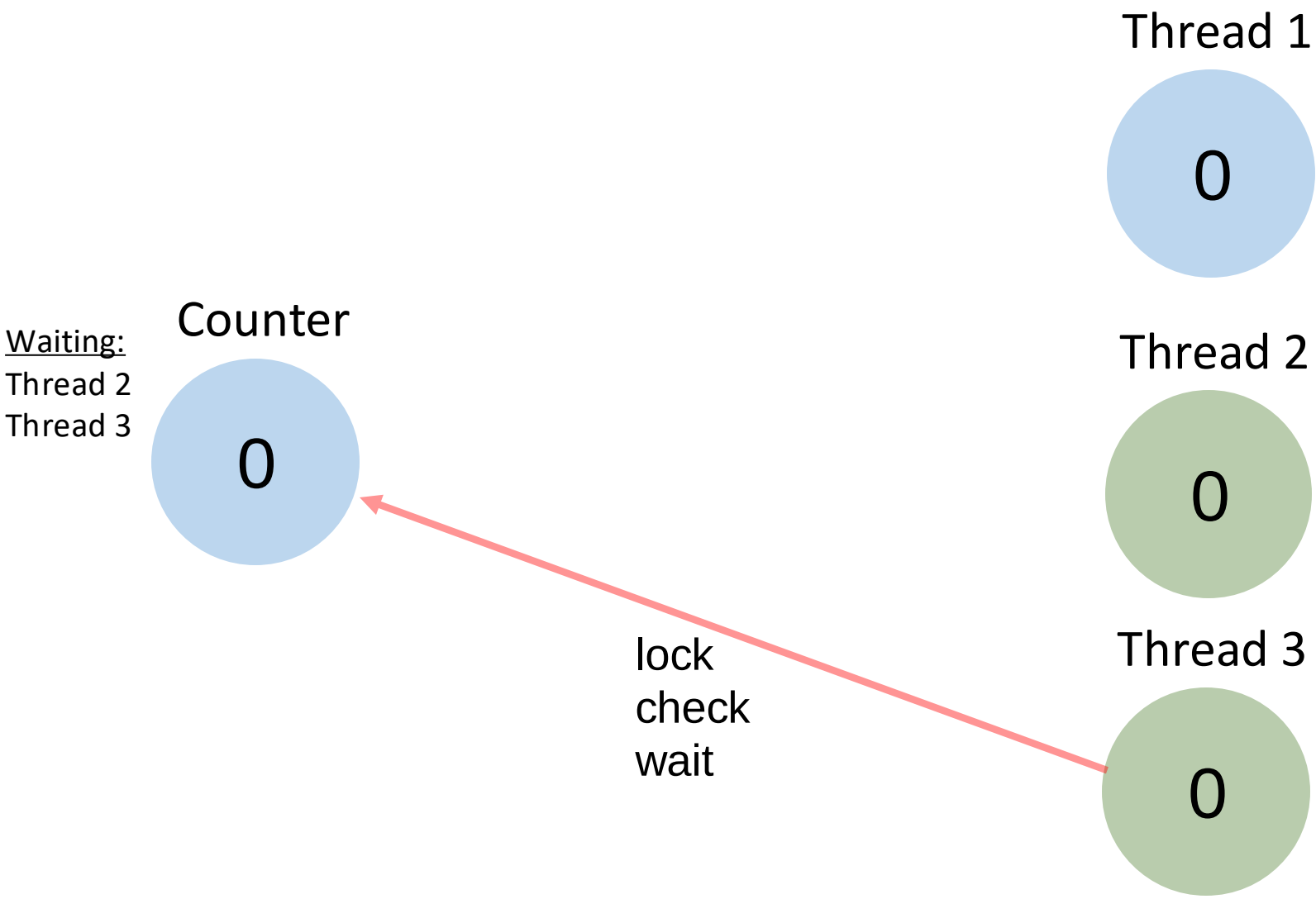

Thread 1
0
Counter
thread 2
Thread 2
0
0
Suspended:
Thread 3
lock failed
Thread 3
0

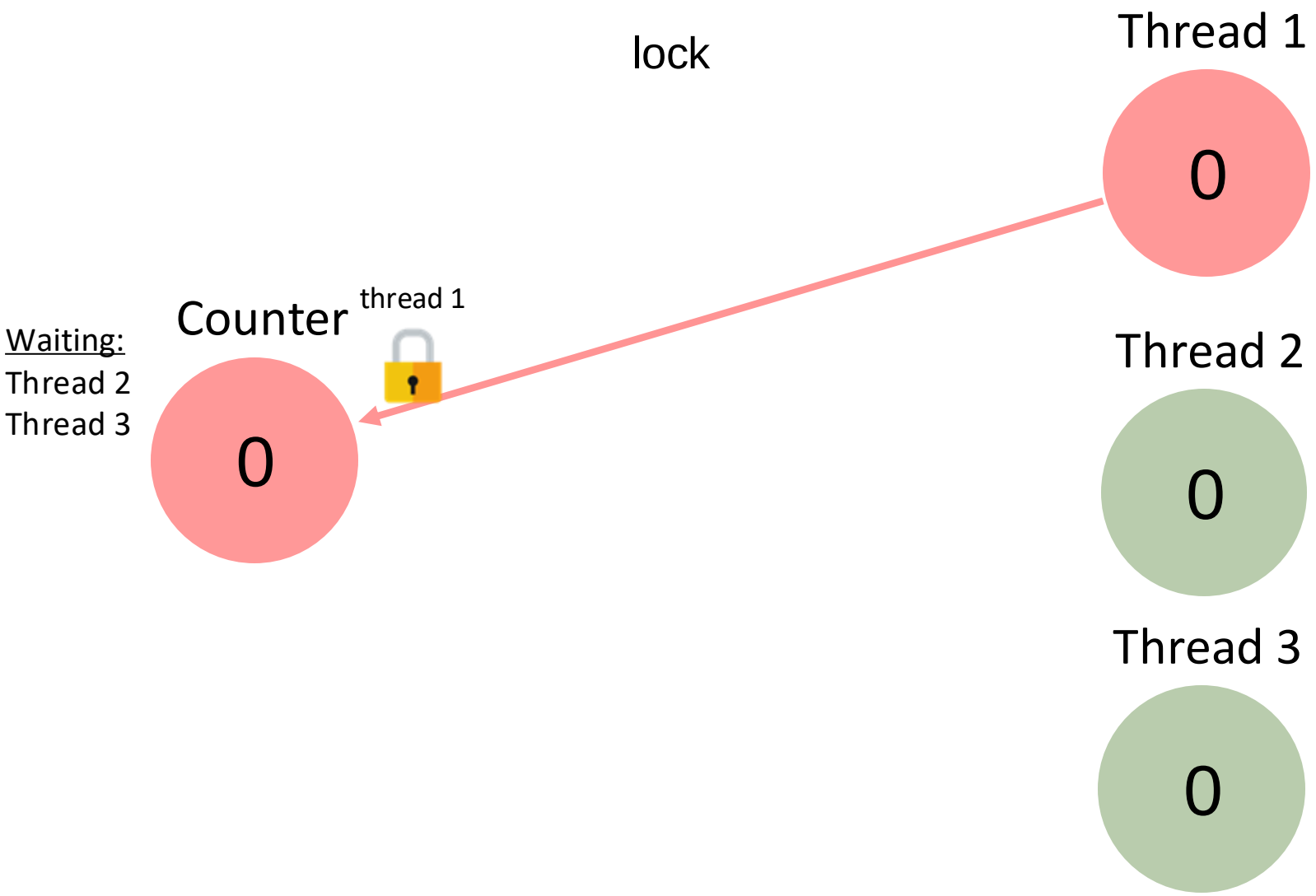

Thread 1
0
Counter
thread 2
lock
check
Thread 2
0
0
Blocked:
Thread 3
Thread 3
0

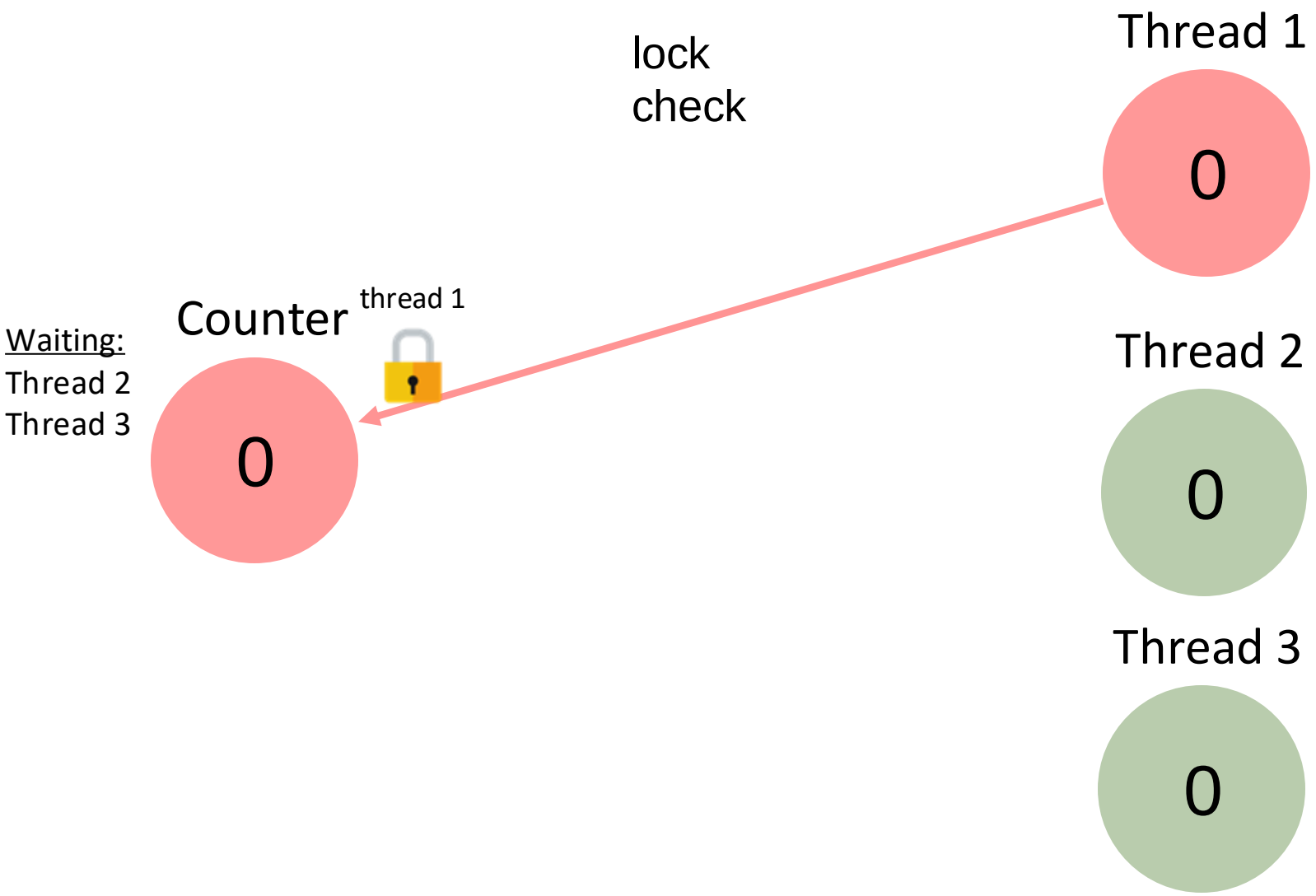

Thread 1
0
Counter
Waiting:
Thread 2
lock
check
wait
Thread 2
0
0
Blocked:
Thread 3
Thread 3
0

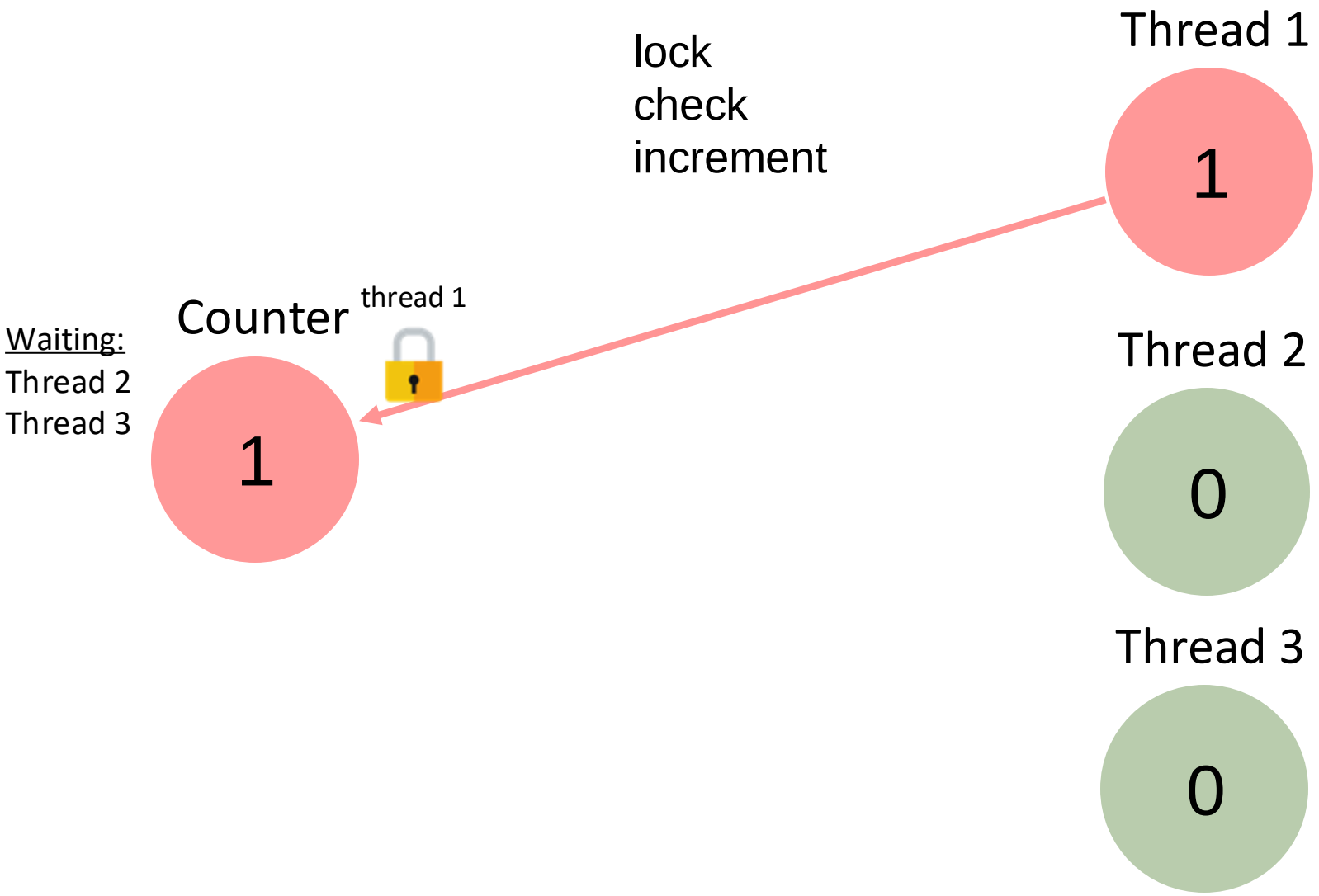

Thread 1
0
Waiting:
Thread 2
Counter
thread 3
0
Thread 2
0
lock
Thread 3
0

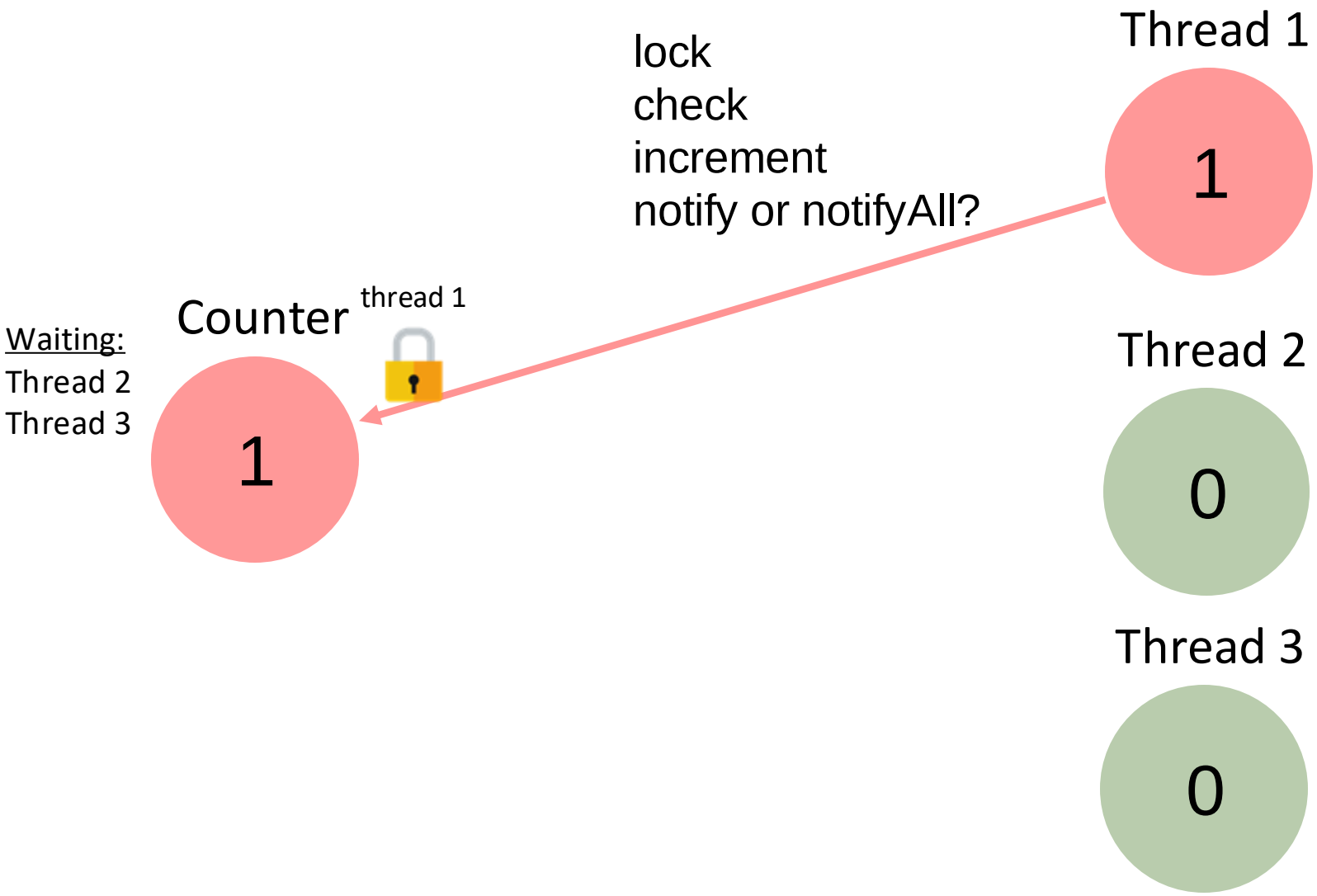

Thread 1
0
Waiting:
Thread 2
Counter
thread 3
0
Thread 2
0
lock
check
Thread 3
0

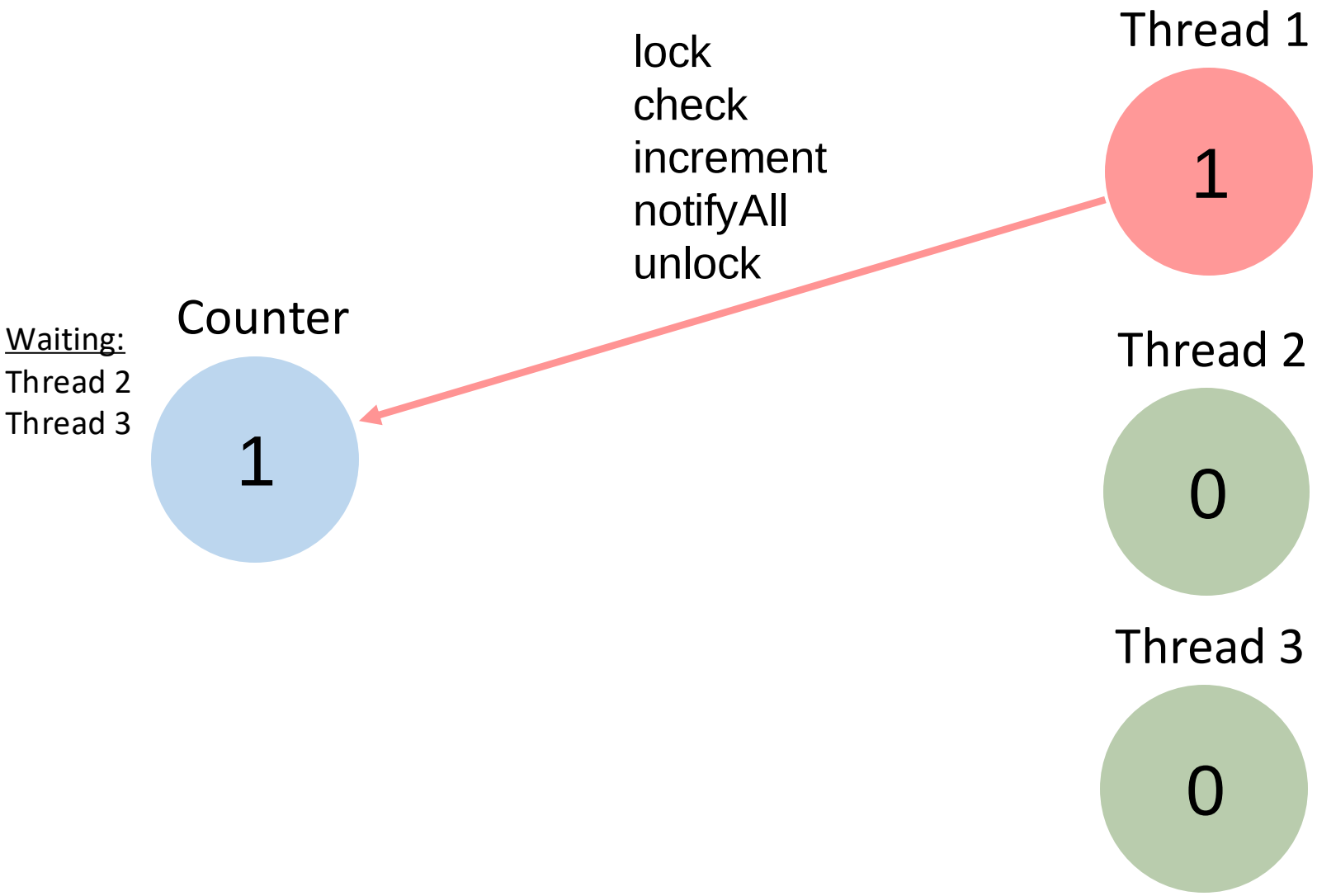

Thread 1
0
Counter
Waiting:
Thread 2
Thread 3
0
Thread 2
0
lock
check
wait
Thread 3
0

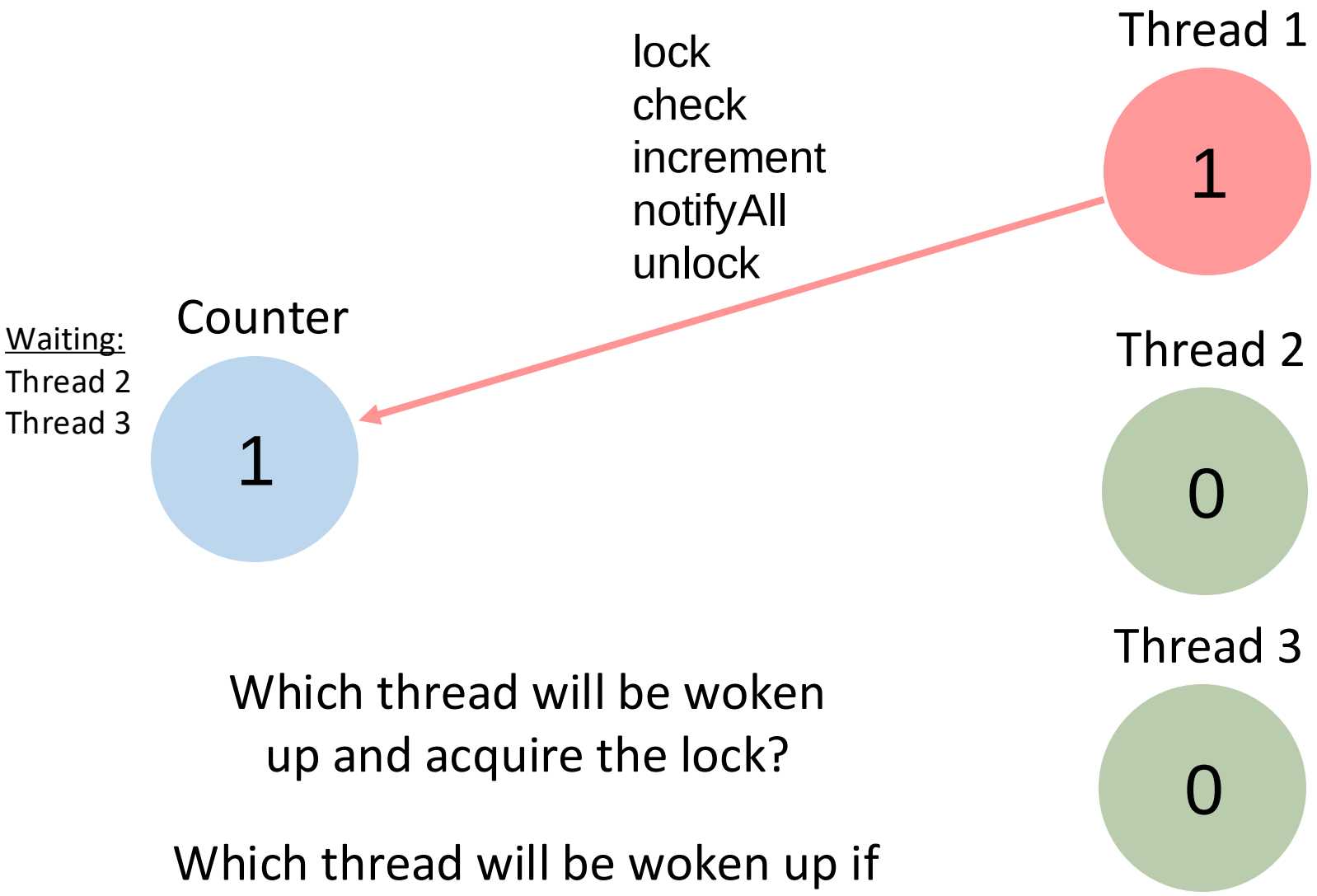

lock
Thread 1
0
Counter
thread 1
Waiting:
Thread 2
Thread 3
0
Thread 2
0
Thread 3
0


lock
check
Thread 1
0
Counter
thread 1
Waiting:
Thread 2
Thread 3
0
Thread 2
0
Thread 3
0


lock
check
increment
Thread 1
1
Counter
thread 1
Waiting:
Thread 2
Thread 3
1
Thread 2
0
Thread 3
0


Thread 1
lock
check
increment
notify or notifyAll?
1
Counter
thread 1
Waiting:
Thread 2
Thread 3
1
Thread 2
0
Thread 3
0


lock
check
increment
notifyAll
unlock
Thread 1
1
Counter
Waiting:
Thread 2
Thread 3
1
Thread 2
0
Thread 3
0



Which thread will be woken up and acquire the lock?

Which thread will be woken up if we use notify instead of notifyAll?

How to find the difference between notify vs notifyAll?

notify

```
public final void notify()
```

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. A thread waits on an object's monitor by calling one of the `wait` methods.

notifyAll

```
public final void notifyAll()
```

Wakes up all threads that are waiting on this object's monitor. A thread waits on an object's monitor by calling one of the `wait` methods.

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Object.html

When notify instead of notifyAll?

Use it if you don‘t care which thread will be notified.

Generally: use notifyAll to avoid edge cases and deadlocks

Task D

```
public static void taskD() {
    Counter counter = new SequentialCounter();
    count(counter, numThreads:2, ThreadCounterType.FAIR, numInterations:100000);
    System.out.println("Task D, Counter: " + counter.value());
}
```

Task D

```java
public class FairThreadCounter extends ThreadCounter {

    public FairThreadCounter(Counter counter, int id, int numThreads, int numIterations) {
        super(counter, id, numThreads, numIterations);
    }

    @Override
    public void run() {
        for (int i = 0; i < numIterations; i++) {
            synchronized (counter) {
                while (counter.value() % numThreads != id) {
                    try {
                        counter.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }

                counter.increment();
                counter.notifyAll();
            }
        }
    }

}
```

Task D

The same thing without wait and notify?

- → Busy waiting: while loop constantly checking (less efficient)
- → Some other complex structure: probably also not more efficient

Task E: AtomicCounter

```
public class AtomicCounter implements Counter {


    public void increment() {
        ??
    }

    public int value() {
        ??
    }
}
```

Task E: AtomicCounter

```
public class AtomicCounter implements Counter {
    private AtomicInteger c = new AtomicInteger(0);

    public void increment() {
        c.incrementAndGet();
    }

    public int value() {
        return c.get();
    }
}
```

Task E: AtomicCounter

```
public class AtomicCounter implements Counter {
    private AtomicInteger c = new AtomicInteger(0);

    public void increment() {
        c.incrementAndGet();
    }

    public int value() {
        return c.get();
    }
}
```

What is the difference?

int	AtomicInteger
c++;	c.incrementAndGet();

Task E: AtomicCounter

```
public class AtomicCounter implements Counter {
    private AtomicInteger c = new AtomicInteger(0);

    public void increment() {
        c.incrementAndGet();
    }

    public int value() {
        return c.get();
    }
}
```

An operation is atomic if no other thread can see it partly executed. Atomic as in “appears indivisible”. However, does not mean it’s implemented as single instruction.

What is the difference?

int

1. load c → 0
2. c + 1 → 1
3. store c ← 1

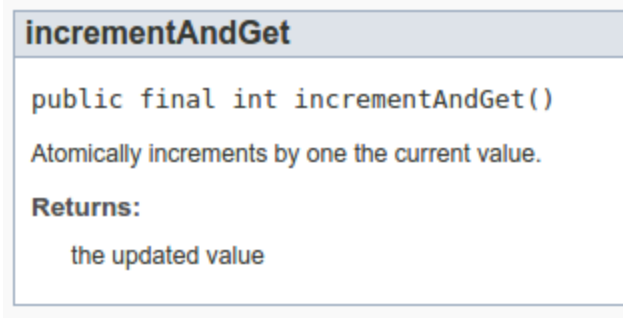
← `c++;`

not atomic

AtomicInteger

`c.getAndIncrement();` →

atomic



incrementAndGet

`public final int incrementAndGet()`

Atomically increments by one the current value.

Returns:

the updated value

Post- vs Pre-Increment

Post-Increment

```
int i = 0;
AtomicInteger c = new AtomicInteger(0);

System.out.println(i++);
System.out.println(c.getAndIncrement());
```

Pre-Increment

```
int i = 0;
AtomicInteger c = new AtomicInteger(0);

System.out.println(++i);
System.out.println(c.incrementAndGet());
```

Task F: Atomic Variables vs Synchronized

Better performance: atomic variables

Why?

Atomic variables are non-blocking

Task G

Task: Create a thread that observes the values of the Counter during the execution and prints them to the console. Make sure that the thread is properly terminated once all the work is done.

Task G

```java
public static void taskG() {
    Counter counter = new AtomicCounter();
    You, 22 seconds ago | 2 authors (juorel and one other)
    Thread progressThread = new Thread(new Runnable() {
        @Override
        public void run() {
            while (!Thread.currentThread().isInterrupted()) {
                System.out.println(counter.value());
            }
        }
    });
    progressThread.start();
    count(counter, numThreads:4, ThreadCounterType.NATIVE, numInterations:100000);
    progressThread.interrupt();
    try {
        progressThread.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("Counter: " + counter.value());
}
```

Theory Recap

Exploiting Parallelism on a Single Core

Pipelining

Instruction-Level Parallelism (ILP)

Vectorization

Pipelining: Main Concepts Recap

Latency

Throughput

Balanced/Unbalanced Pipeline

Pipelining: Main Concepts Recap

Latency

time needed to perform a given computation
(e.g., process a customer)

Throughput

Balanced/Unbalanced Pipeline

Pipelining: Main Concepts Recap

Latency

time needed to perform a given computation
(e.g., process a customer)

Throughput

amount of work that can be done by a system in a given period of time
(e.g., how many customers can be processed in one minute)

Balanced/Unbalanced Pipeline

Pipelining: Main Concepts Recap

Latency

time needed to perform a given computation
(e.g., process a customer)

Throughput

amount of work that can be done by a system in a given period of time
(e.g., how many customers can be processed in one minute)

Balanced/Unbalanced Pipeline

a pipeline is balanced if each stage takes the same length of time

Instance vs. Stage

Latency

Generally, you can take the total time of the first instance.

$$latency = total\_time(first\_instance) = sum(time(all\_stages))$$

If not constant, you can calculate it for the n-th instance.

$$latency = total\_time(first\_instance) + (\max(time\_stage) - time(first\_stage)) \cdot (n-1)$$

Definition 4.2.2. Throughput is the number of elements that exit the pipeline (at full capacity) per a given time unit. Throughput can be calculated as follows for *any* pipeline with one execution unit per stage:

$$Throughput = \frac{1}{max(computationtime(stages))}$$

Definition 4.2.3. Throughput under consideration of lead-in and lead-out time given n elements traverse the pipeline is the average time it takes to output an element. This throughput can be calculated as follows for *any* pipeline with one execution unit per stage:

$$\frac{n}{overall\ time\ for\ n\ elements}$$

$$= \frac{n}{n * max(computationtime(stages)) + sum(computationtime(all\ stages\ except\ longest))}$$

https://polybox.ethz.ch/index.php/s/KwIm2IqvWYP79UB?path=%2FSkript#pdfviewer

Library

At UZH the law students have been tasked with writing a legal essay about the philosophy of Swiss law. In order to write the essay, each student needs to read four different books on the subject, denoted as A, B, C and D (in this order).

This exercise is created by Lasse Meinen and is part of the unofficial VIS Prüfungsvorbereitungsworkshop Scripts available at:

https://vis.ethz.ch/de/services/pvw-scripts/

Every student takes the exact same amount of time to read a book, concretely:

1) Reading book **A** takes 80 minutes
2) Reading book **B** takes 40 minutes
3) Reading book **C** takes 120 minutes
4) Reading book **D** takes 40 minutes

Library

Over at UZH the law students have been tasked with writing a legal essay about the philosophy of Swiss law. In order to write the essay, each student needs to read four different books on the subject, denoted as A, B, C and D (in this order).

Question 1: Let's assume all law students are a bit too competitive and don't return any books before they're done reading all of them. How long will it take for 4 students until all of them have started writing their essays?

Every student takes the exact same amount of time to read a book, concretely:

1) Reading book **A** takes 80 minutes
2) Reading book **B** takes 40 minutes
3) Reading book **C** takes 120 minutes
4) Reading book **D** takes 40 minutes

Library

Total: 4 * 280 min

Latency: 280 min

Throughput: 1 per 280 min

Question 1: Let's assume all law students are a bit too competitive and don't return any books before they're done reading all of them. How long will it take for 4 students until all of them have started writing their essays?

Every student takes the exact same amount of time to read a book, concretely:

1) Reading book **A** takes 80 minutes
2) Reading book **B** takes 40 minutes
3) Reading book **C** takes 120 minutes
4) Reading book **D** takes 40 minutes

Library

Draw diagrams, as seen before

Question 2: The library introduces a ”one book at a time” policy, i.e., the students have to return a book before they can start on the next one. How long will it now take for 4 students until all of them have started writing their essays?

Every student takes the exact same amount of time to read a book, concretely:

1) Reading book **A** takes 80 minutes
2) Reading book **B** takes 40 minutes
3) Reading book **C** takes 120 minutes
4) Reading book **D** takes 40 minutes

Library

Latency?

Question 2: The library introduces a ”one book at a time” policy, i.e., the students have to return a book before they can start on the next one. How long will it now take for 4 students until all of them have started writing their essays?

Every student takes the exact same amount of time to read a book, concretely:

1) Reading book **A** takes 80 minutes
2) Reading book **B** takes 40 minutes
3) Reading book **C** takes 120 minutes
4) Reading book **D** takes 40 minutes

Library

Latency?

280 min

320 min

360 min

400 min

Question 2: The library introduces a ”one book at a time” policy, i.e., the students have to return a book before they can start on the next one. How long will it now take for 4 students until all of them have started writing their essays?

Every student takes the exact same amount of time to read a book, concretely:

1) Reading book **A** takes 80 minutes
2) Reading book **B** takes 40 minutes
3) Reading book **C** takes 120 minutes
4) Reading book **D** takes 40 minutes

Library

For this pipeline, latency makes sense only if asked for a particular student, not for the whole pipeline.

Latency?

280 min

320 min

360 min

400 min

Question 2: The library introduces a ”one book at a time” policy, i.e., the students have to return a book before they can start on the next one. How long will it now take for 4 students until all of them have started writing their essays?

Every student takes the exact same amount of time to read a book, concretely:

1) Reading book **A** takes 80 minutes
2) Reading book **B** takes 40 minutes
3) Reading book **C** takes 120 minutes
4) Reading book **D** takes 40 minutes

Library

Balanced? **Throughput?** **Latency?**

Question 2: The library introduces a ”one book at a time” policy, i.e., the students have to return a book before they can start on the next one. How long will it now take for 4 students until all of them have started writing their essays?

Every student takes the exact same amount of time to read a book, concretely:

1) Reading book **A** takes 80 minutes
2) Reading book **B** takes 40 minutes
3) Reading book **C** takes 120 minutes
4) Reading book **D** takes 40 minutes

Library

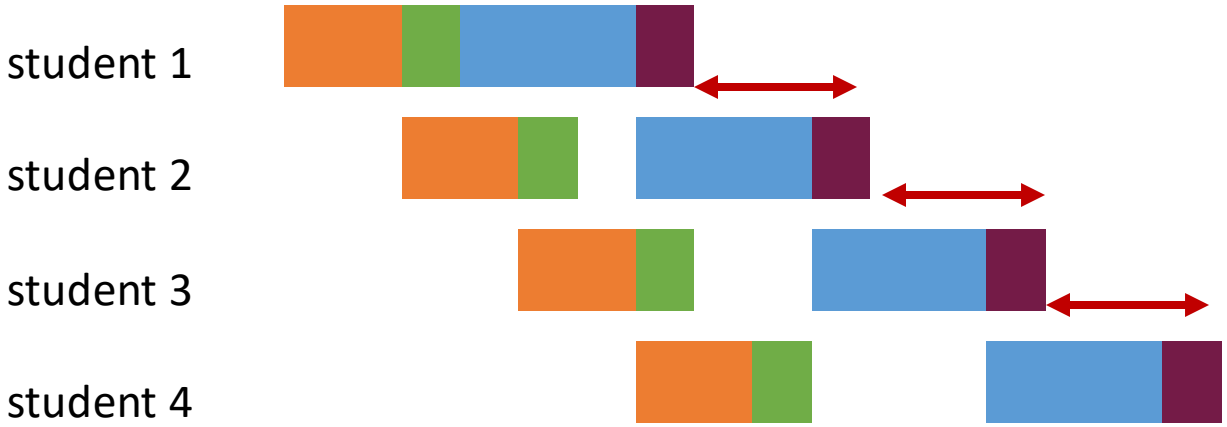
Balanced?	Throughput?	Latency?
	1 student	280 min
	per 160 minutes	320 min
	With lead in (fixed number of students)	360 min
		400 min

Question 2: The library introduces a ”one book at a time” policy, i.e., the students have to return a book before they can start on the next one. How long will it now take for 4 students until all of them have started writing their essays?

Every student takes the exact same amount of time to read a book, concretely:

1) Reading book **A** takes 80 minutes
2) Reading book **B** takes 40 minutes
3) Reading book **C** takes 120 minutes
4) Reading book **D** takes 40 minutes

Library



Balanced?	Throughput?	Latency?
	1 student	280 min
	per 120 minutes	320 min
	Without lead in (indefinite number of students)	360 min
		400 min

Question 2: The library introduces a ”one book at a time” policy, i.e., the students have to return a book before they can start on the next one. How long will it now take for 4 students until all of them have started writing their essays?

Every student takes the exact same amount of time to read a book, concretely:

1) Reading book **A** takes 80 minutes
2) Reading book **B** takes 40 minutes
3) Reading book **C** takes 120 minutes
4) Reading book **D** takes 40 minutes

Library

	Balanced?	Throughput?	Latency?
student 1	No	1 student per 160 minutes	280 min
student 2			320 min
student 3			360 min
student 4			400 min

The pipeline is not balanced since the stages have different length

Every student takes the exact same amount of time to read a book, concretely:

1) Reading book **A** takes 80 minutes
2) Reading book **B** takes 40 minutes
3) Reading book **C** takes 120 minutes
4) Reading book **D** takes 40 minutes

Library

Draw diagrams, as seen before

Question 3: At UZH the law students now need to read the books E, F, G, H, in this particular order. Let's assume we still have the "one book at a time" policy.

Every student takes the exact same amount of time to read a book, concretely:

1) Reading book **E** takes 120 minutes
2) Reading book **F** takes 80 minutes
3) Reading book **G** takes 120 minutes
4) Reading book **H** takes 40 minutes

Library

	Balanced?	Throughput?	Latency?
student 1	No	1 student per 180 minutes	360 min
student 2			360 min
student 3			360 min
student 4			360 min

The pipeline is not balanced since the stages have different length. The latency is constant.

Every student takes the exact same amount of time to read a book, concretely:

1) Reading book **E** takes 120 minutes
2) Reading book **F** takes 80 minutes
3) Reading book **G** takes 120 minutes
4) Reading book **H** takes 40 minutes

Library

Draw diagrams, as seen before

Question 4: A motivated student has written summaries for books E, F and G and shared them with the library. As our four students would like to enjoy some sun someday, they decided to read the summaries instead. Every summary takes exactly 40 minutes for each student.

Every student takes the exact same amount of time to read a summary, concretely:

1) Reading summary **E** takes 40 minutes
2) Reading summary **F** takes 40 minutes
3) Reading summary **G** takes 40 minutes
4) Reading summary **H** takes 40 minutes

Library

	Balanced?	Throughput?	Latency?
student 1	Yes	1 student per 70 minutes	160 min
student 2			160 min
student 3			160 min
student 4			160 min

The pipeline is balanced since the stages have the same length. The latency is constant

Every student takes the exact same amount of time to read a summary, concretely:

1) Reading summary **E** takes 40 minutes
2) Reading summary **F** takes 40 minutes
3) Reading summary **G** takes 40 minutes
4) Reading summary **H** takes 40 minutes

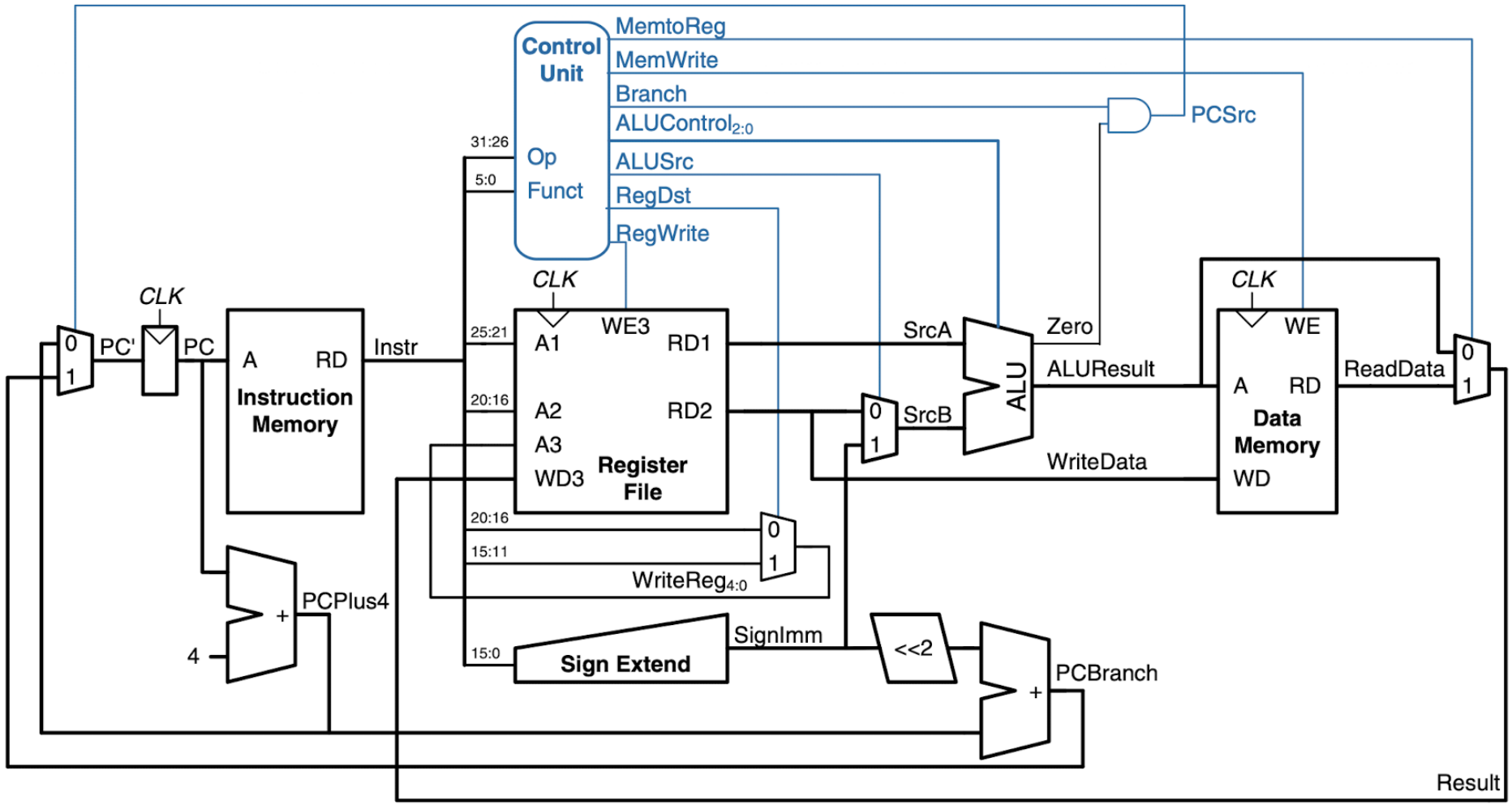
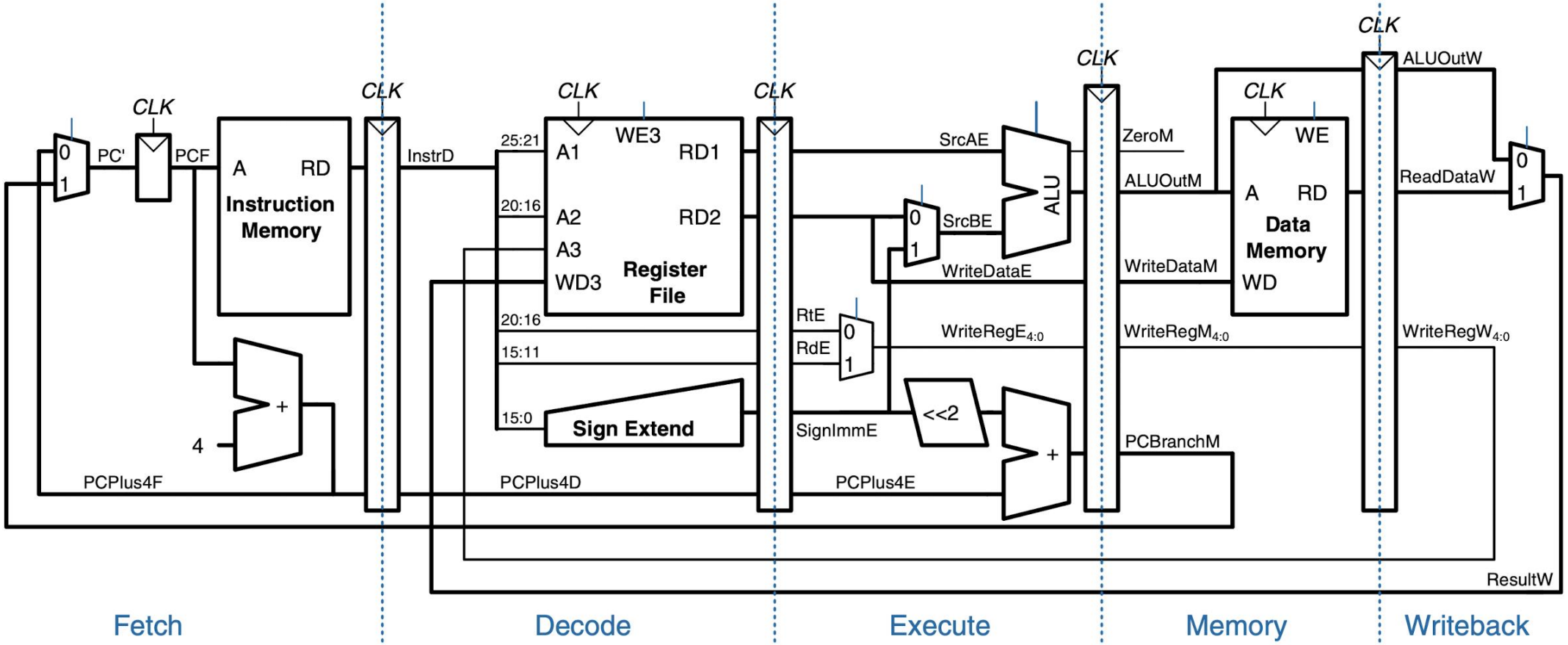


Figure 7.11 Complete single-cycle MIPS processor


CLK
PC'
PCF
A
RD
Instruction Memory
InstrD
25:21
20:16
15:11
15:0
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
Sign Extend
RtE
RdE
SrcAE
SrcBE
ALU
WriteDataE
WriteRegE4:0
SignImmE
<<2
PCBranchM
ZeroM
ALUOutM
WriteDataM
WriteRegM4:0
WE
WD
Data Memory
ALUOutW
ReadDataW
WriteRegW4:0
ResultW
4
PCPlus4F
PCPlus4D
PCPlus4E
Fetch
Decode
Execute
Memory
Writeback

Instruction-Level Parallelism

How do we get parallelism out of a program on a single core?

→ Special hardware support

Idea: execute independent instructions in parallel

Independent instructions: result of one isn‘t input of the other

```
A1 <- a + b
A2 <- c + d
A3 <- A1 + A2
```

Vectorization

operations on vector-like data like arrays

Independent operations of the **same type**

Vectorization

Can easily parallelize the loop:

All interations are independent of each other

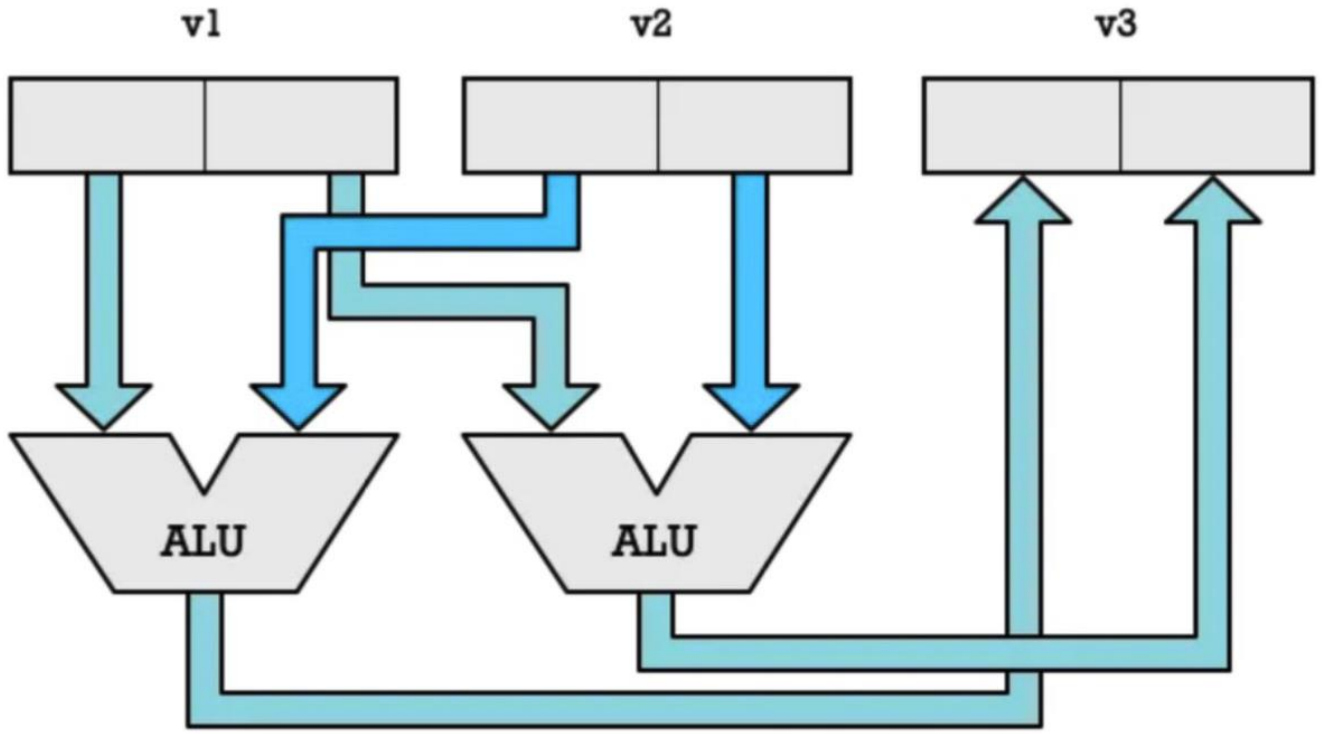
Each thread works on their part (like in assignment 2)

How about a single core approach (Vectorization)?
→ SIMD

```
double[] a = ...;
double[] b = ...; //same size;
double[] result= new double [SIZE];

final int SIZE = a.length;
for(int i=0;  i<SIZE; i++){
    result[i] = a[i] + b[i];
  }
return result;
}
```

Single Instruction Multiple Data (SIMD)

```
VLOAD V1, (a) // load next two doubles of 'a'
VLOAD V2, (b) // load next two doubles of 'b'
VMUL V3, V1, V2 // V3 <- V1 * V2
```



Vector-Instructions (Assembly)

```
vmovdqu   0x10(%r9, %rbx, 8), %ymm0
vmulpd    0x10(%rcx, %rbx, 8), %ymm0, %ymm0
vmovdqu   %ymm0, 0x10(%r13, %rbx, 8)
movslq    %ebx, %rsi
vmovdqu   0x30(%r9, %rsi, 8), %ymm0
vmulpd    0x30(%rcx, %rsi, 8), %ymm0, %ymm0
vmovdqu   %ymm0, 0x30(%r13, %rsi, 8)
```

Combine multiple instructions into a single vector-instruction

When does this happen?

Either compiler does it for us (auto-vectorization)

or

Programmer specifies it via vector-instructions

… given that the hardware supports it

(basically all modern hardware does)

More on those hardware optimizations

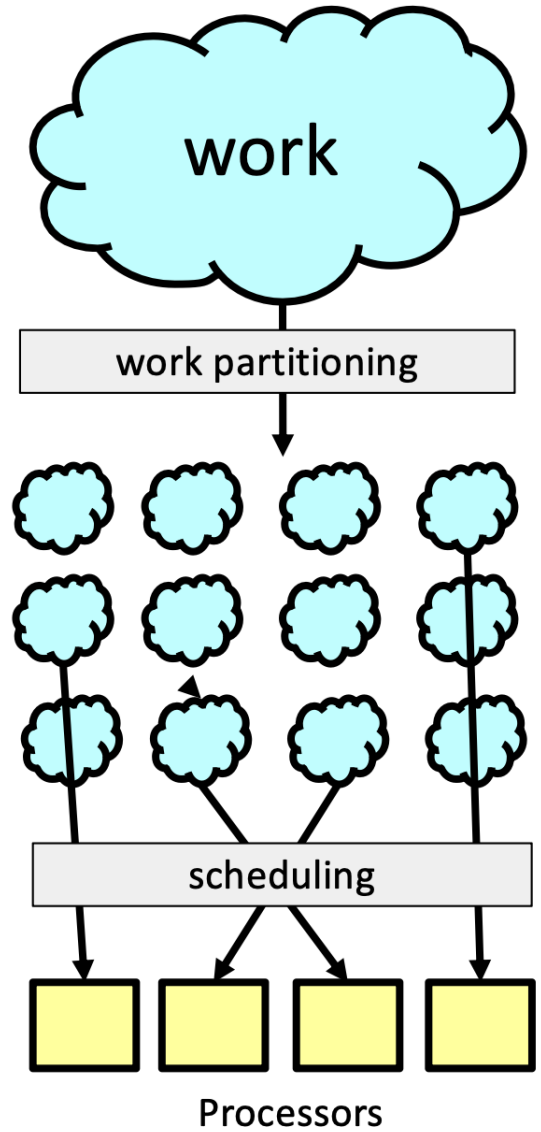
Digital Design and Computer Architecture

SPCA

Work Partitioning & Scheduling

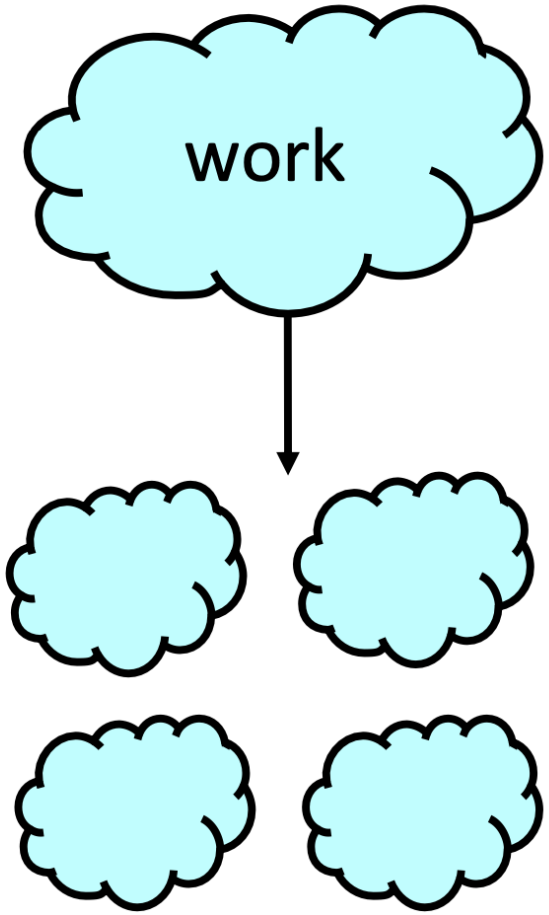
- **work partitioning**
  - **split up** work into **parallel tasks/threads**
  - (done by user)
  - A task is a unit of work
  - also called: **task/thread decomposition**
- **scheduling**
  - assign tasks to processors
  - (typically done by the system)
  - goal: full utilization

  (no processor is ever idle)

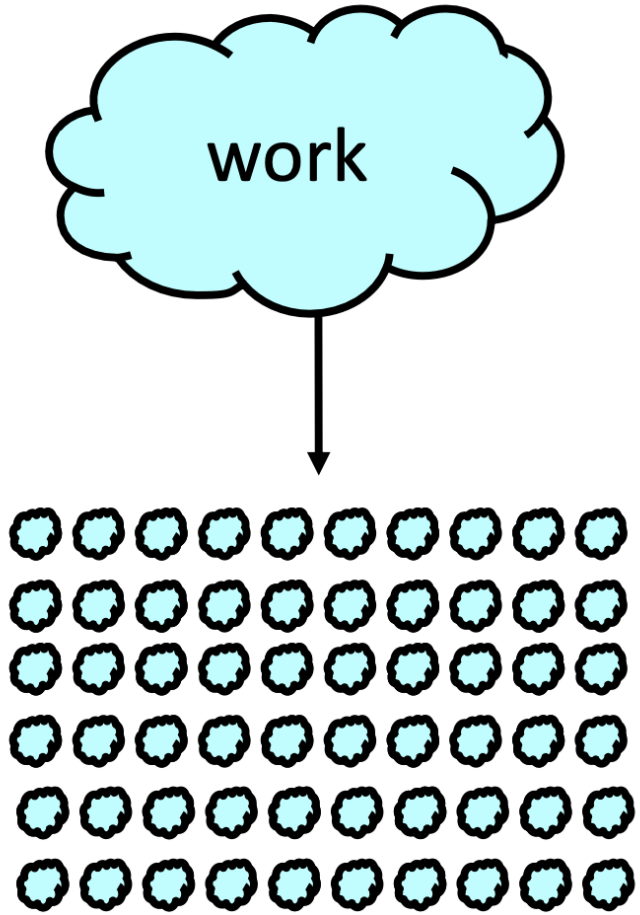


# of chunks should be larger than the # of processors

Task/Thread Granularity



Coarse granularity



Fine granularity

Coarse vs Fine granularity

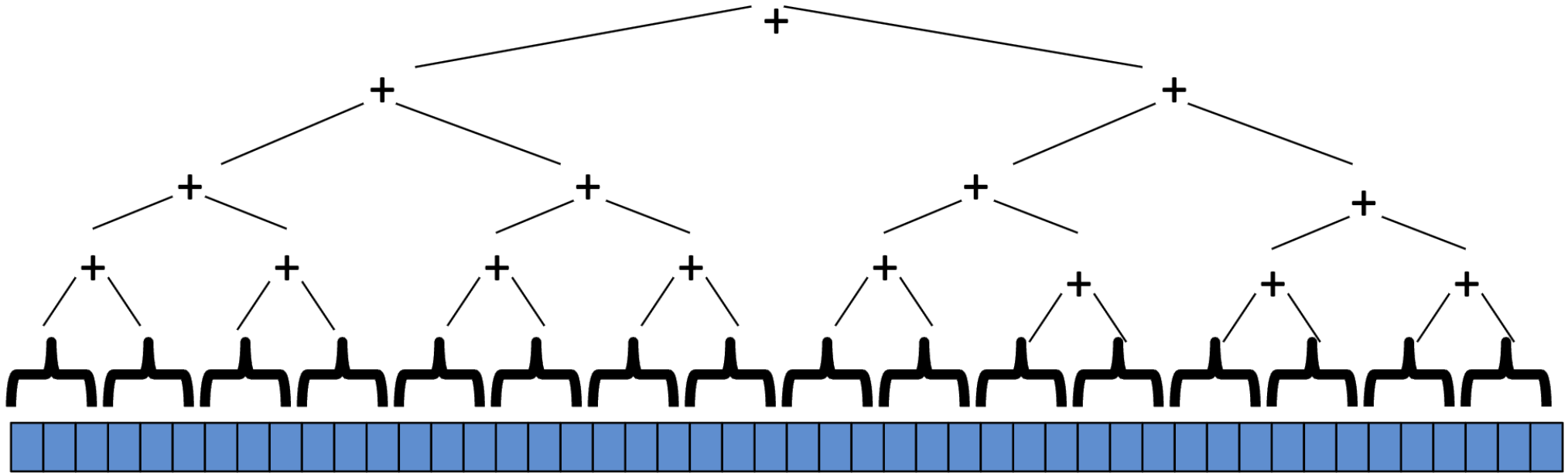
- **Fine granularity:**
  - more portable

  (can be executed in machines with more processors)
  - better for scheduling
  - but: if scheduling overhead is comparable to a single task → overhead dominates

Task granularity guidelines

- As small as possible
- but, significantly bigger than scheduling overhead
  - system designers strive to make overheads small

Divide and Conquer



With threads

```
public class SumThread extends Thread {
int[] xs;
int h, l;
int result;


public SumThread(int[] xs, int l, int h){
    super();
    this.xs = xs;
    this.h = h;
    this.l = l;
}


public void run(){
    /* Do computation and write to result */
    return;
}
```

```
public void run(){
    int size = h - l;
    if (size == 1) {
        result = xs[l];
        return;
    }
    int mid = size / 2;
    SumThread t1 = new SumThread(xs, l, l + mid);
    SumThread t2 = new SumThread(xs, l + mid, h);

    t1.start();
    t2.start();

    t1.join();
    t2.join();

    result = t1.result + t2.result;
    return;
}
```

Remark: This doesn't compile because join() can throw exceptions. We need a try-catch block here.

For large arrays: **Java.lang.OutOfMemoryError**: unable to create new native thread

With threads

```
public class SumThread extends Thread {
int[] xs;
int h, l;
int result;


public SumThread(int[] xs, int l, int h){
    super();
    this.xs = xs;
    this.h = h;
    this.l = l;
}


public void run(){
    /* Do computation and write to result */
    return;
}
```

Sequential cutoff: if (size <= cutoff) then calculate that part sequentially

```
public void run(){
    int size = h - l;
    if (size == 1) {
        result = xs[l];
        return;
    }
    int mid = size / 2;
    SumThread t1 = new SumThread(xs, l, l + mid);
    SumThread t2 = new SumThread(xs, l + mid, h);

    t1.start();
    t2.start();

    t1.join();
    t2.join();

    result = t1.result + t2.result;
    return;
}
```

Remark: This doesn't compile because join() can throw exceptions. We need a try-catch block here.

For large arrays: **Java.lang.OutOfMemoryError**: unable to create new native thread

With threads

```
public class SumThread extends Thread {
int[] xs;
int h, l;
int result;


public SumThread(int[] xs, int l, int h){
    super();
    this.xs = xs;
    this.h = h;
    this.l = l;
}


public void run(){
    /* Do computation and write to result */
    return;
}
```

Sequential cutoff: if (size <= cutoff) then calculate that part sequentially

```
public void run(){
    int size = h - l;
    if (size == 1) {
        result = xs[l];
        return;
    }
    int mid = size / 2;
    SumThread t1 = new SumThread(xs, l, l + mid);
    SumThread t2 = new SumThread(xs, l + mid, h);

    t1.start();
    t2.start();
    t2.run();
    t1.join();
    t2.join();

    result = t1.result + t2.result;
    return;
}
```

Remark: This doesn't compile because join() can throw exceptions. We need a try-catch block here.

For large arrays: **Java.lang.OutOfMemoryError**: unable to create new native thread

What do we want?

Partition the work into desirable sizes without running out of memory.

Solution: Create Tasks and let the scheduler deside when to map each task to which thread (with a fixed thread count)

Executor-Service

Can deadlock, if no threads to execute task

Executor-Service

ex.submit(Runnable task)

Submits a Runnable object for execution and returns a Future object representing that task.

ex.submit(Callable<T> task)

Submits a value-returning task for execution and returns a Future object representing the pending results of the task.

ex.shutdown()

Previously submitted tasks are executed, but no new tasks will be accepted.

Executor-Service

```
public static void main(String[] args) {
    /* submit 100 callables to the service and store the returned
     Future objects in a list */
    ExecutorService ex = Executors.newFixedThreadPool(4);
    List<Future<Integer>> futures = new ArrayList<>();
    for (int i = 0; i < 100; i++) {
        Future<Integer> future = ex.submit(new MyCallable());
        futures.add(future);
    }

    // sum up the results of all tasks
    int sum = 0;
    for (Future<Integer> future : futures) {
        try {
            sum += future.get();
        } catch (InterruptedException | ExecutionException e) {
            e.printStackTrace();
        }
    }

    System.out.println("Sum of all results: " + sum);
    ex.shutdown();
}
```

For what is Executor Service not suitable

Recursive problems where deep structures require waiting for partial results

Well-suited for **flat structures** or tasks that can run independently in parallel

Fork-Join

Fork/join framework solves the deadlock problem of ExecutorService

Designed for recursive tasks

Fork-Join

```
public int sumArray(int[] arr) {
    ForkJoinPool fj = new ForkJoinPool(4);
    SumRecCall sumTask = new SumRecCall(arr, 0, arr.length);
    int result = fj.invoke(sumTask);
    fj.shutdown();
    return result;
}
```

```
        int mid = size / 2;
        SumRecCall first = new SumRecCall(arr, startIdx, startIdx + mid);
        SumRecCall second = new SumRecCall(arr, startIdx + mid, endIdx);
        first.fork();
        second.fork();
        int firstSum = first.join();
        int secondSum = second.join();
        return firstSum + secondSum;
    }
}
```

Parallel Performance

Sequential execution time: T_1

Execution time T_p on **p** CPUs

- $T_p = T_1 / p$ (perfection)
- $T_p > T_1 / p$ (performance loss, what normally happens)
- $T_p < T_1 / p$ (sorcery!)

(parallel) Speedup

(parallel) speedup S_p on **p** CPUs:

$$S_p = T_1 / T_p$$

- $S_p = p$ → linear speedup (perfection)
- $S_p < p$ → sub-linear speedup (performance loss)
- $S_p > p$ → super-linear speedup (sorcery!)

- **Efficiency: S_p / p**

Amdahl's Law – Ingredients

Given P workers available to do parallelizable work, the times for sequential execution and parallel execution are:

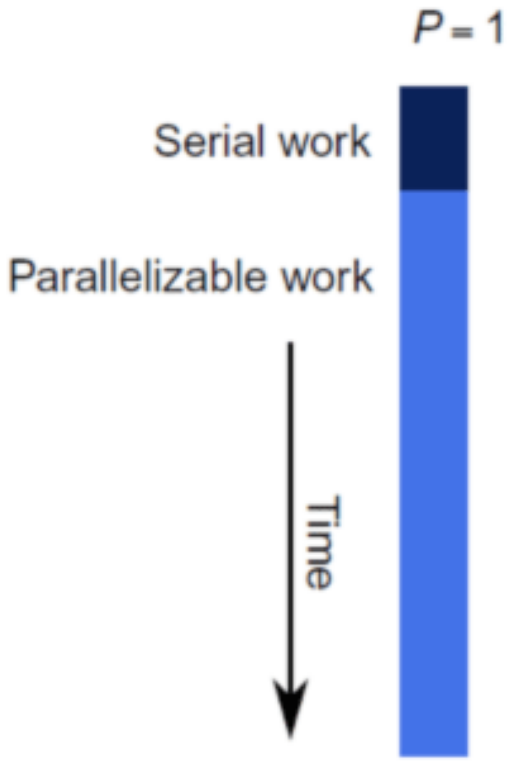
$$T_1 = W_{ser} + W_{par}$$

And this gives a bound on speed-up:

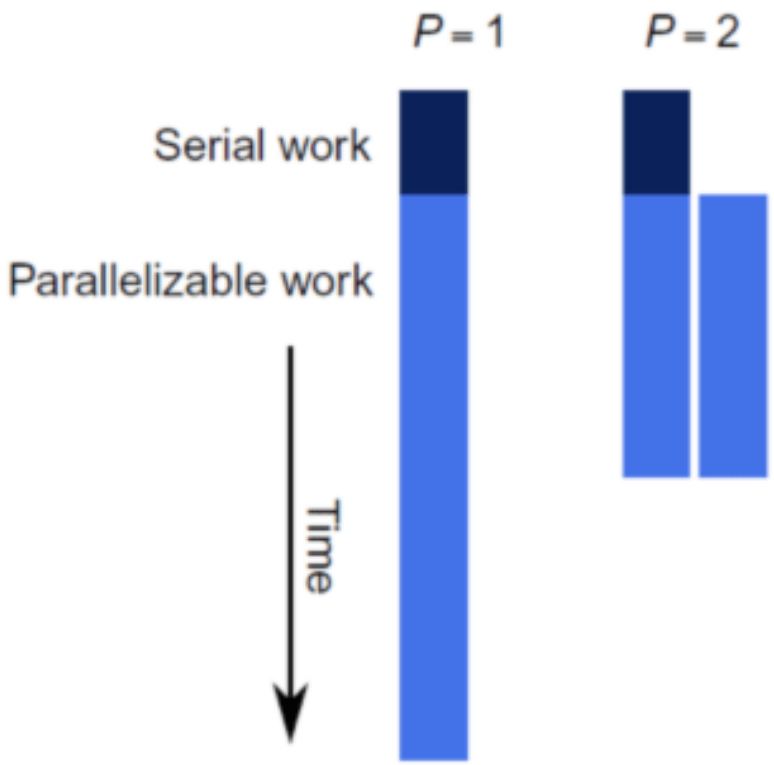
$$T_p \geq W_{ser} + \frac{W_{par}}{P}$$

$$S_\infty \leq \frac{1}{\mathbf{f}} \qquad S_p \leq \frac{W_{ser} + W_{par}}{W_{ser} + \frac{W_{par}}{P}} = \frac{1}{\mathbf{f} + \frac{1-\mathbf{f}}{P}}$$

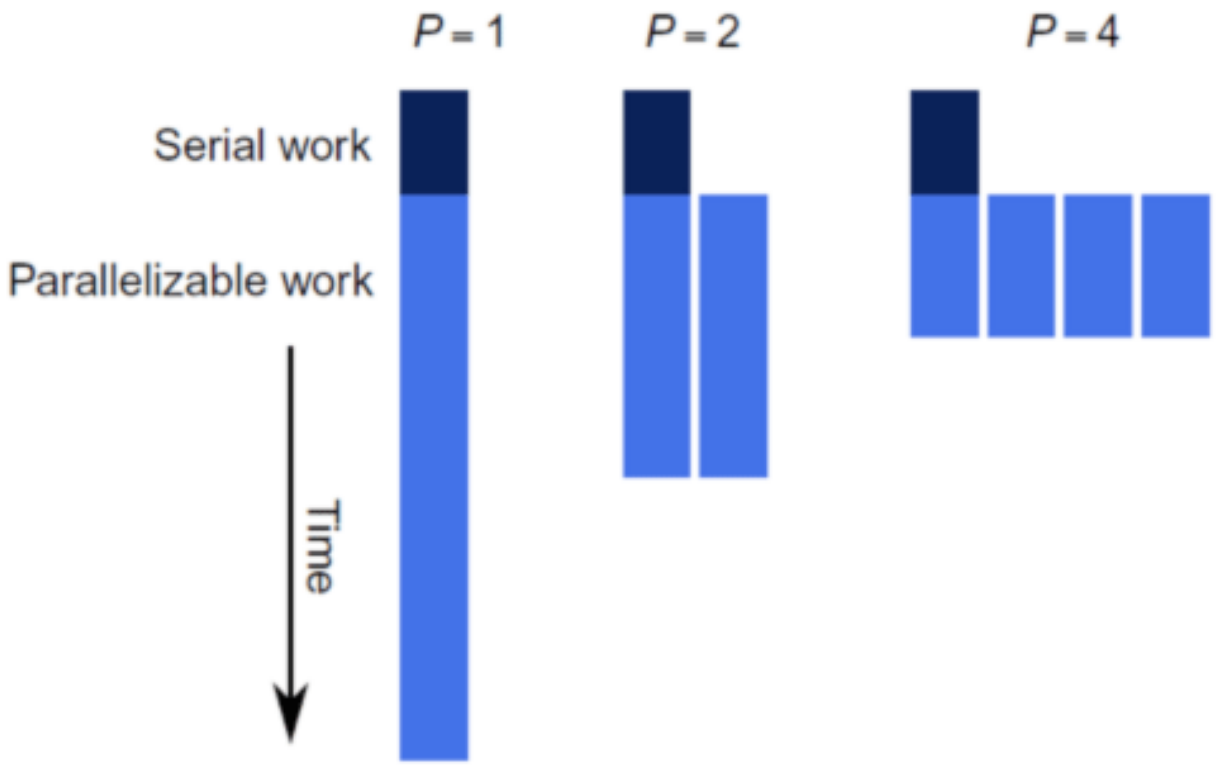
Amdahl's Law Illustrated



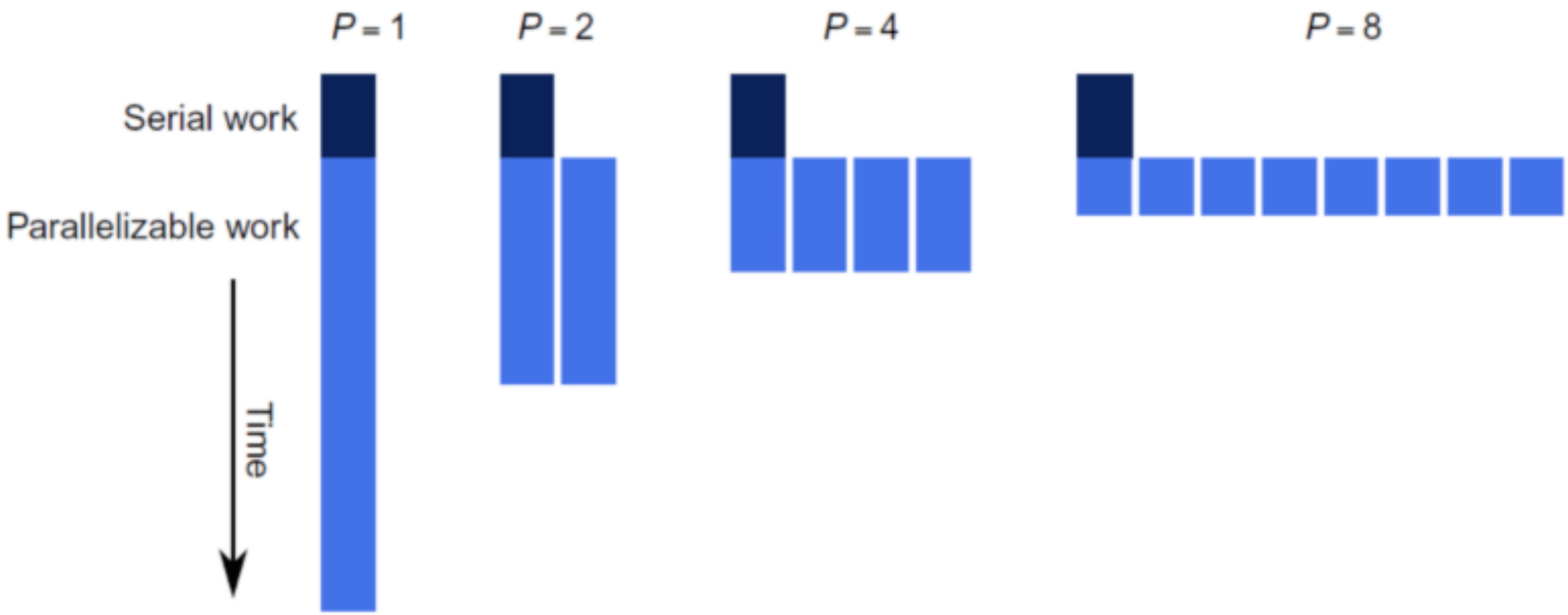
Amdahl's Law Illustrated



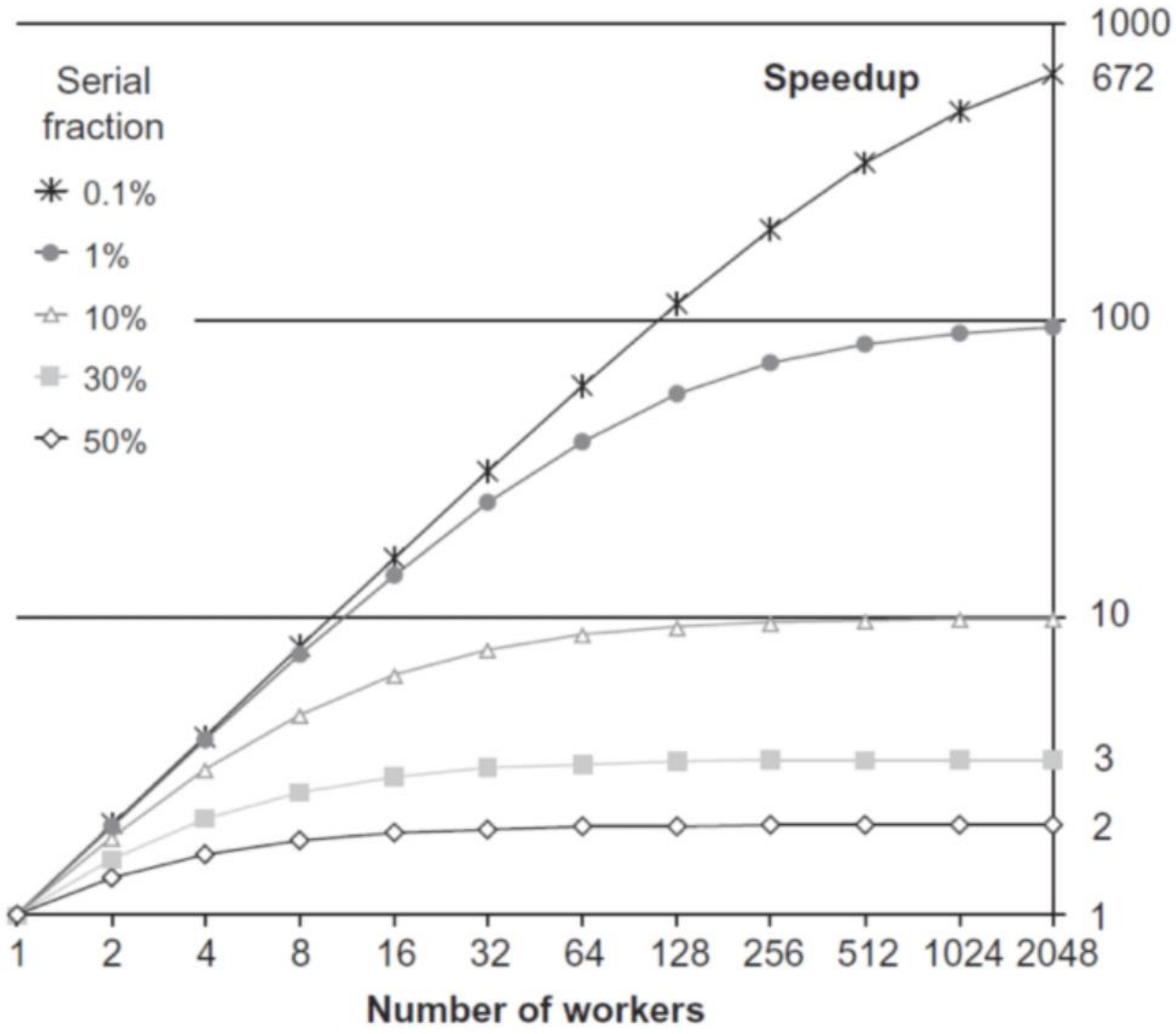
Amdahl's Law Illustrated



Amdahl's Law Illustrated



Speedup


Serial fraction
0.1%
1%
10%
30%
50%
Speedup
1000
672
100
10
3
2
1
1
2
4
8
16
32
64
128
256
512
1024
2048
Number of workers

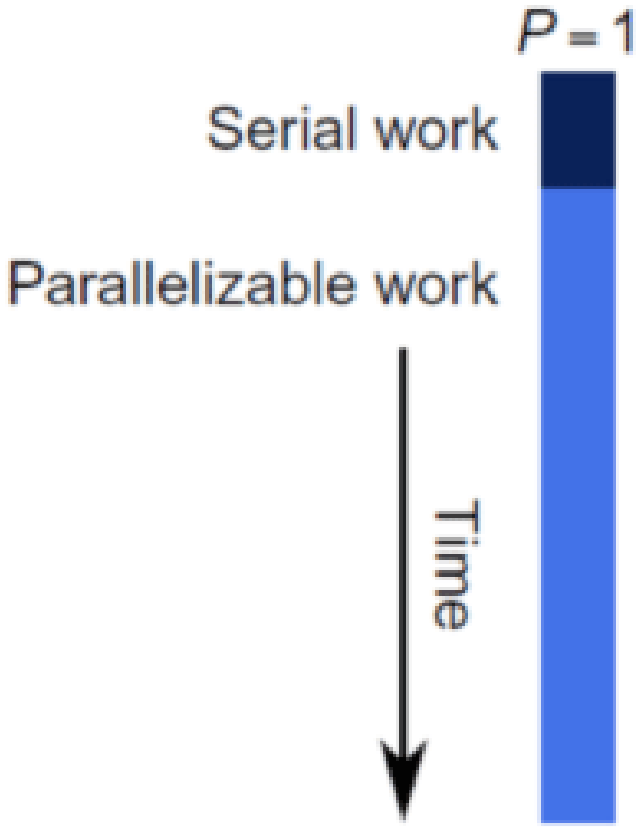
Gustafson‘s Law

$$W = \mathbf{f} * W + (1 - \mathbf{f}) * W$$

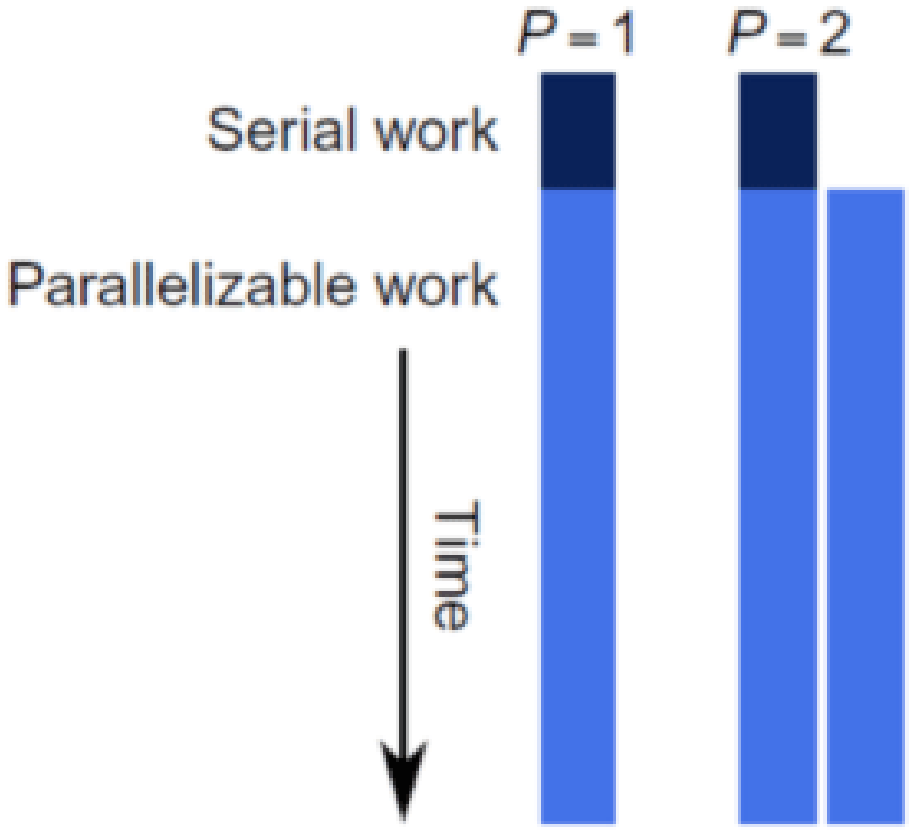
$$W_P = \mathbf{f} * W + P * (1 - \mathbf{f}) * W$$

$$\begin{aligned} S_P &= \mathbf{f} + P(1 - f) \\ &= P - \mathbf{f}(P - 1) \end{aligned}$$

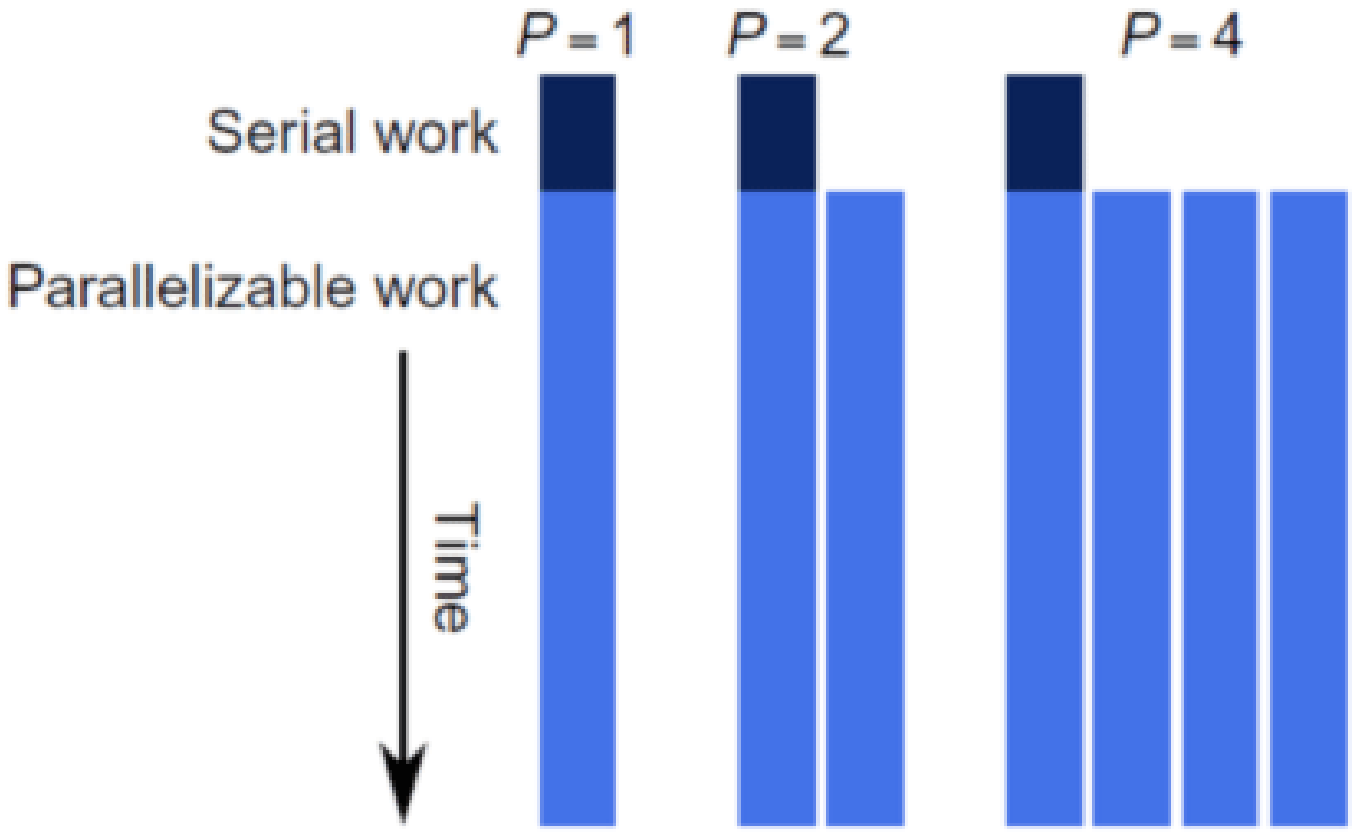
Gustafson's Law



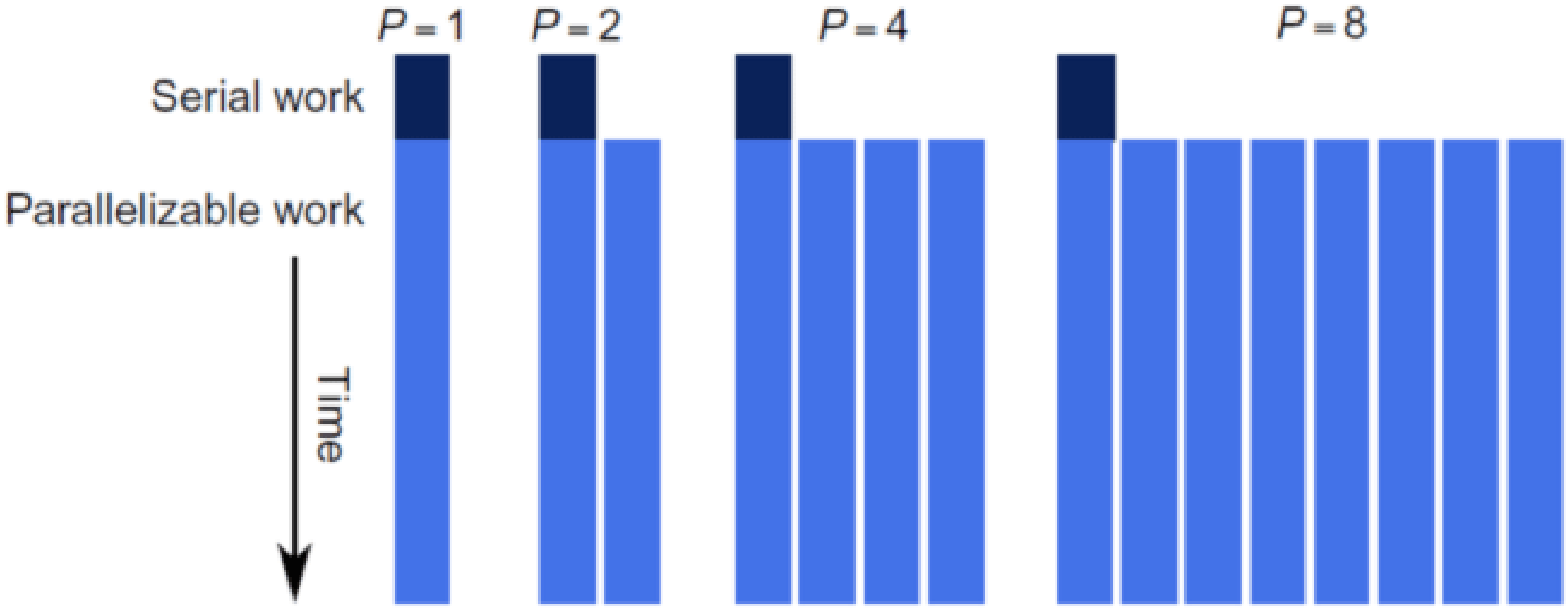
Gustafson's Law



Gustafson's Law



Gustafson's Law



Exercise 4

Task 1 - Pipelining

Bob, Mary, John and Alice

50 min

90 min

15 min

a) Laundry time using sequential order

b) Design a strategy with better laundry time

c) How would the laundry time improve if they bought a new dryer?

Task 2 - Pipelining II

Assume a processor that can each cycle issue either:

- one multiplication instruction with latency 6 cycles
- one addition instruction with latency 3 cycles

How many cycles are required to execute following loops?

```
for (int i = 0; i < data.length; i++) {
    data[i] = data[i] * data[i];
}
```

```
for (int i = 0; i < data.length; i += 2) {
    j = i + 1;
    data[i] = data[i] * data[i];
    data[j] = data[j] * data[j];
}
```

```
for (int i = 0; i < data.length; i += 4) {
    j = i + 1;
    k = i + 2;
    l = i + 3;
    data[i] = data[i] * data[i];
    data[j] = data[j] * data[j];
    data[k] = data[k] * data[k];
    data[l] = data[l] * data[l];
}
```

Task 3 - Identify Potential Parallelization

Can we parallelize following two loops using parallel for construct?

```
for (int i=1; i<size; i++) {    // for loop: i from 1 to (size-1)
    if (data[i-1] > 0)          // If the previous value is positive
        data[i] = (-1)*data[i]; // change the sign of this value
}                               // end for loop
```

```
for (int i=0; i<size; i++) {      // for loop: i from 0 to (size-1)
    data[i] = Math.sin(data[i]);  // calculate sin() of the value
}                                 // end for loop
```

Past Exam Task

Pipelining.
Sie leiten eine Produktionslinie in einer Fabrik, bei der die Produkte vier Montagephasen durchlaufen. Jede Phase hat eine spezifische Bearbeitungszeit:

(A) Material schneiden: 5 Minuten pro Produkt.

(B) Teile zusammenbauen: 10 Minuten pro Produkt.

(C) Lackieren: 15 Minuten pro Produkt.

(D) Qualitätsprüfung: 7 Minuten pro Produkt.

Pipelining.
You are managing a factory production line where products go through four stages of assembly. Each stage has a specific processing time:

(A) Cutting materials: 5 min per product.

(B) Assembling parts: 10 min per product.

(C) Painting: 15 min per product.

(D) Quality check: 7 min per product.

Exam, FS 2024

Past Exam Task

i. Was ist der **Throughput** der Produktionslinie? **Spezifizieren Sie die Einheiten.**

*What is the **throughput** of the production line? **Specify the units.*** (1)

ii. Was ist die **Latency** der Produktionslinie? **Spezifizieren Sie die Einheiten.**

*What is the **latency** of the production line? **Specify the units.*** (1)

Exam, FS 2024

Past Exam Task

i. Was ist der **Throughput** der Produktionslinie? **Spezifizieren Sie die Einheiten.** *What is the **throughput** of the production line? **Specify the units.*** (1)

$$\text{Throughput} = 60/15 = 4 \text{ products per hour}$$

or

$$\text{Throughput} = 1/15 = 1 \text{ product/15min}$$

ii. Was ist die **Latency** der Produktionslinie? **Spezifizieren Sie die Einheiten.** *What is the **latency** of the production line? **Specify the units.*** (1)

$$\text{Latency} = 5 + 10 + 15 + 7 = 37 \text{ minutes}$$

Exam, FS 2024


QUIZ