Lecture 26: Finishing Locks

COSC 273: Parallel and Distributed Computing

Spring 2023

This Week

- Homework 03 Due Friday
- Final Project group & topic selection, also due Friday
  - Option 1: computing prime numbers
  - Option 2: sorting
  - Option 3: choose your own adventure
- Week Plan
  - Today: finish locks
  - Wednesday: computing prime numbers
  - Friday: sorting

Last Time: `TASLock`

Test-and-set Lock

```java
import java.util.concurrent.atomic.AtomicBoolean;
public class TASLock implements SimpleLock {
    AtomicBoolean locked = new AtomicBoolean(false);
    public void lock () {
        while (locked.getAndSet(true)) {}
    }
    public void unlock () {
        locked.set(false);
    }
}
```

equiv. to
- boolean val = locked
- locked = true
- return val

- download `tas-locks.zip`

Progress Guarantees

Question. Is `TASLock` deadlock-free? Starvation-free?

→ Deadlock free:
- if all threads take steps, someone gets lock in finite # of steps

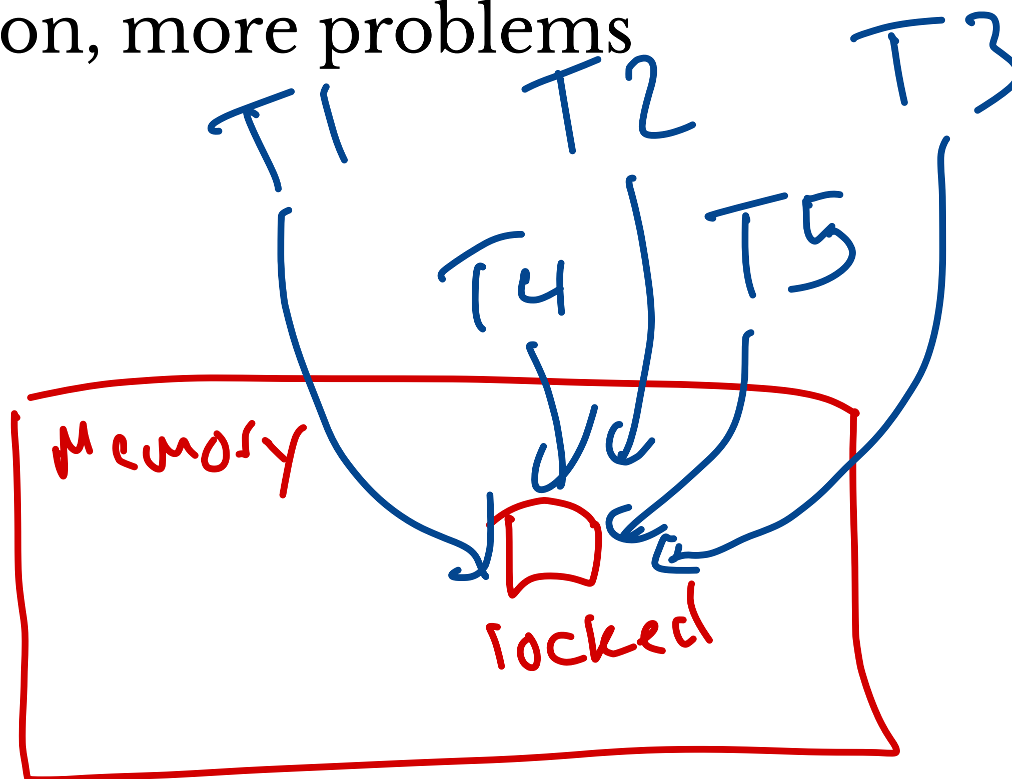
Yes! first thread to call locked.getAndSet(true) obtains lock, next after unlock gets lock next time...

∴ Starvation free? Each locker eventually gets lock

No! No guarantee about who obtains lock after release.

What About Performance?

- atomic operations are expensive
  - writing operations more-so
- more contention, more problems



Question: How does running time per lock/unlock depend on # threads?

Atomic Lock with Fewer Writes

Test-and-Test-and-Set Lock:

- check if locked
  - if not, attempt `getAndSet`
  - return if successful

TTASLock Implementation

```
public class TTASLock implements SimpleLock {
    AtomicBoolean locked = new AtomicBoolean(false);
    public void lock () {
        while (true) {
            while (locked.get()) {};
            if (!locked.getAndSet(true)) { return;}
        }
    }
    public void unlock() { locked.set(false);}
}
```

only read

Comparing Efficiency

- `tas-locks.zip`

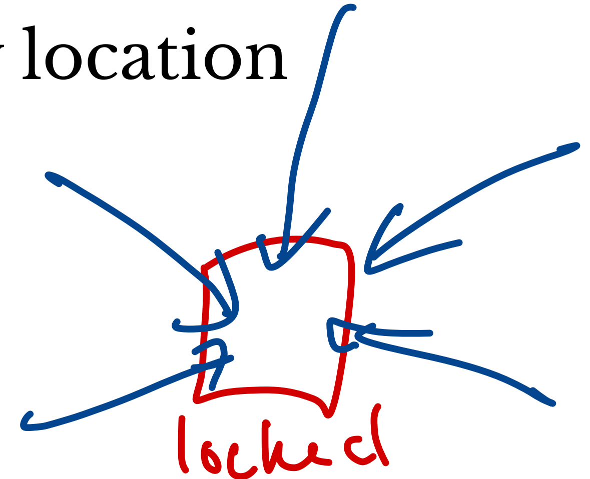
Two Issues

1. Locks are less efficient with more threads
    - more contention to single memory location
2. Locks are not starvation free
    - no notion of priority

Fix # threads

→ some notion of priority

⟹ more space required



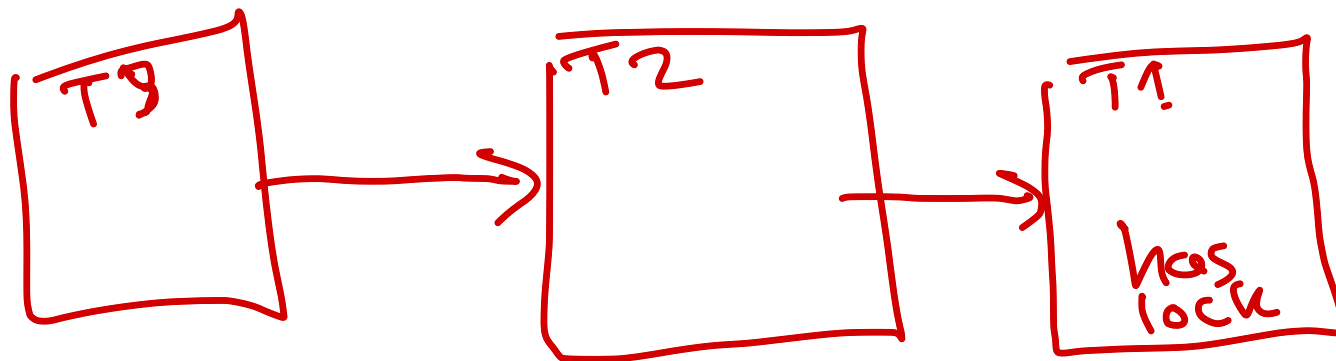
A Tradeoff

More memory, less contention

- each thread has its own field to lock/unlock

Incorporating priority

- each thread has predecessor it waits on
- like queue/Bakery algorithm



CLH Lock

Each thread has:

- `Node myNode` node "owned" by thread
- `Node myPred` node owned by predecessor thread

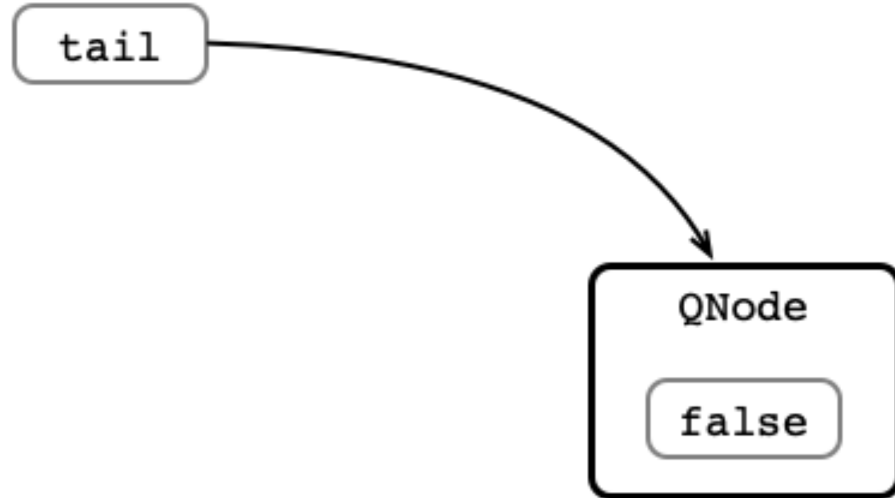
thread I am waiting on

Each `Node` has:

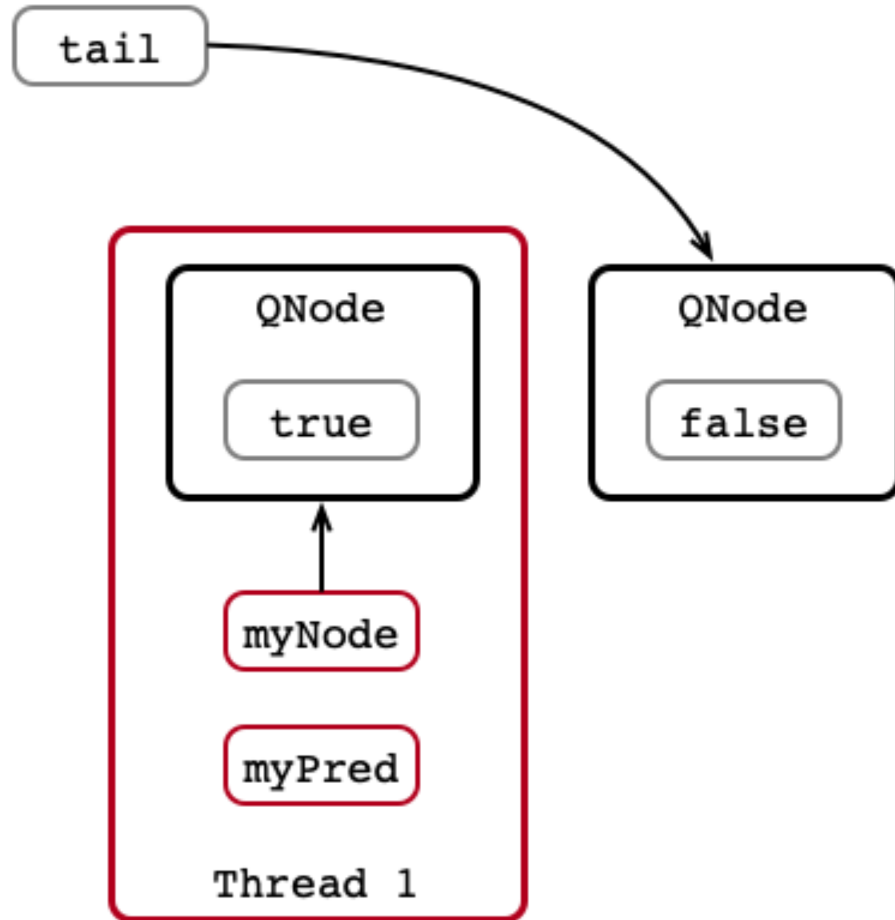
- `boolean locked`:
  - `myNode.locked = true` signals I want/have lock
  - `myNode.locked = false` signals I have released lock

Thread acquires lock when `myPred.locked` is `false`

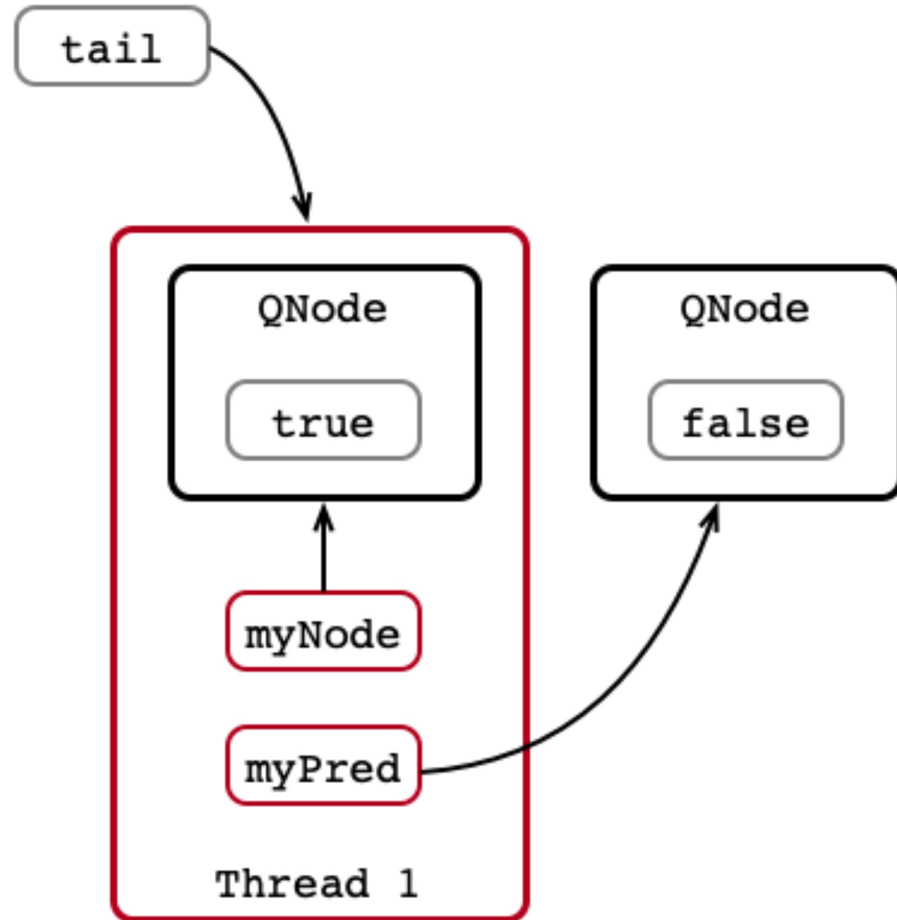
CLH Lock Initial State



Thread 1 Arrives

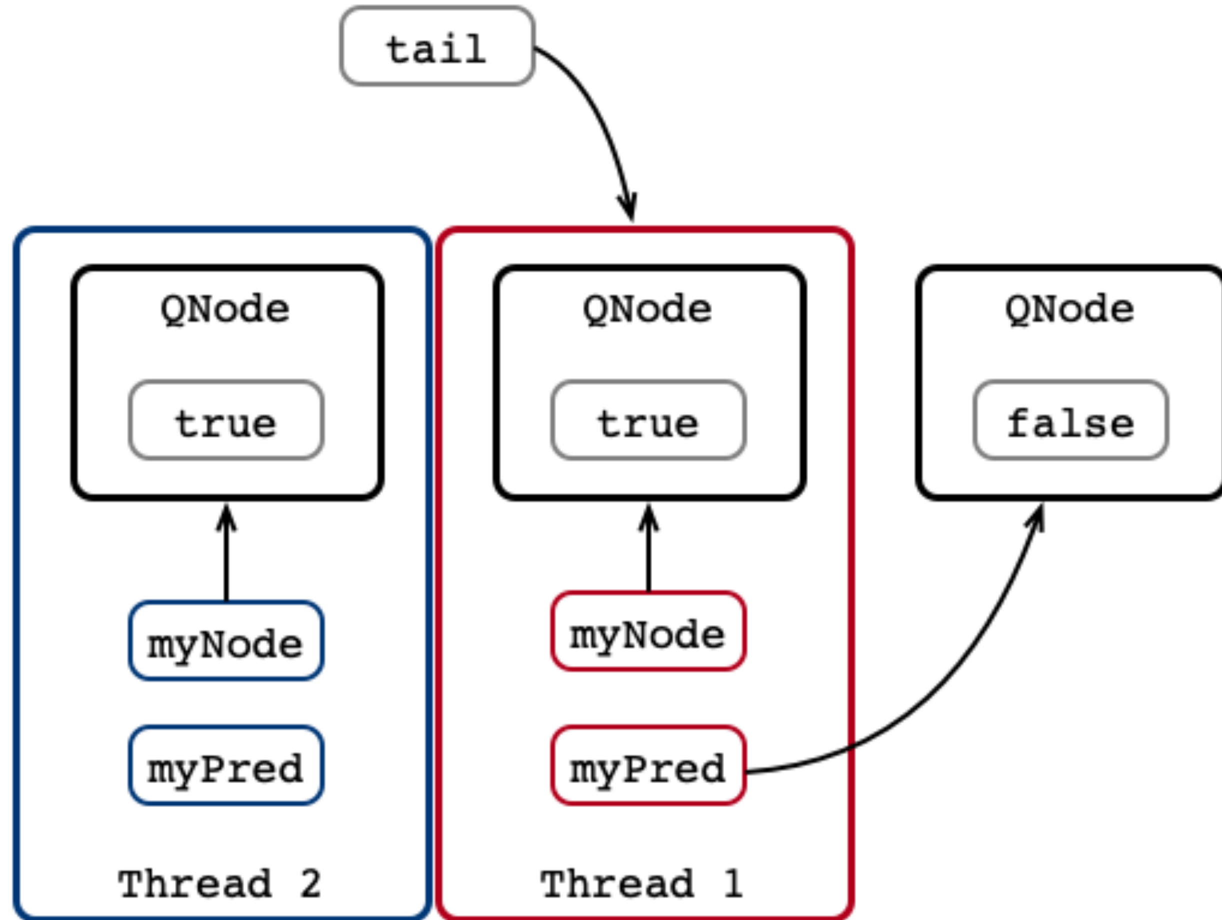


Thread 1 Acquires Lock

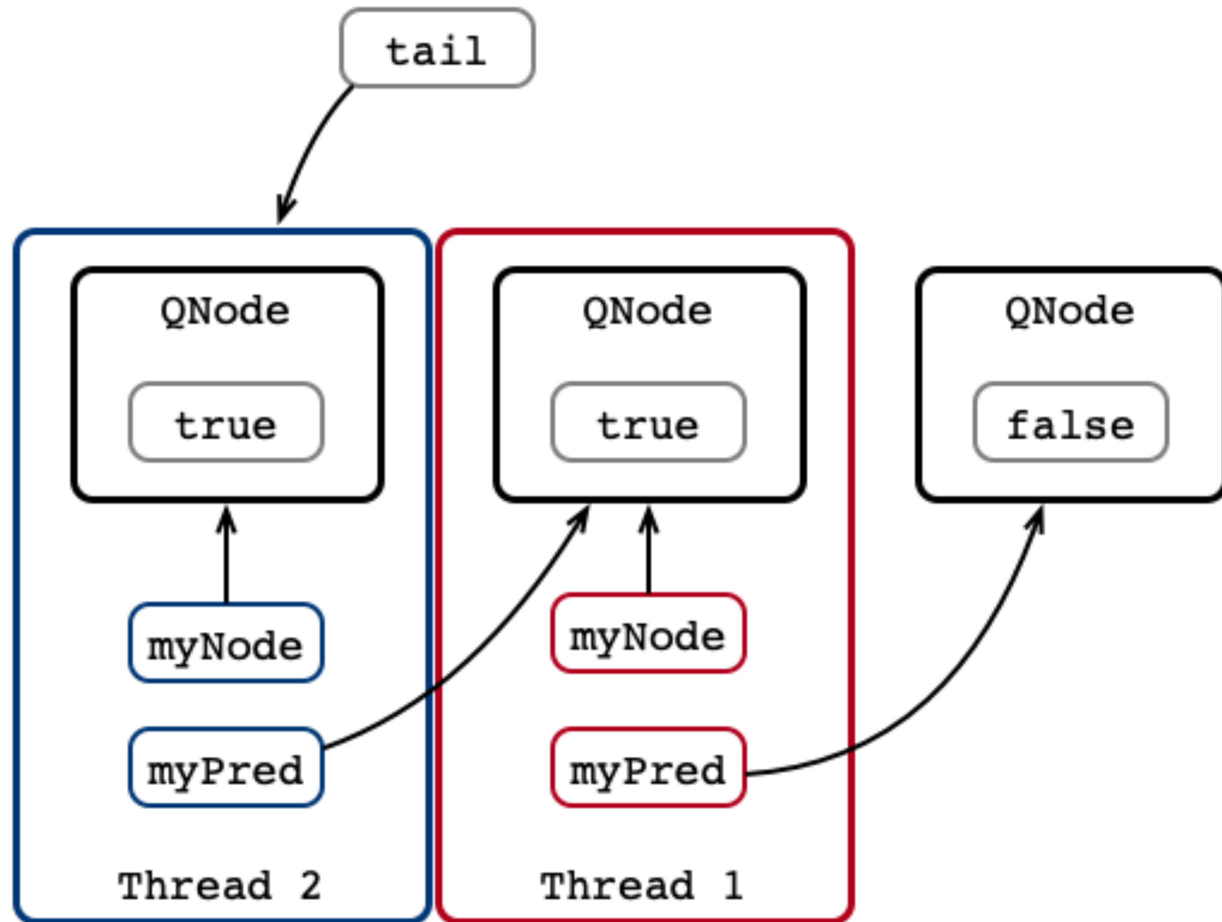


T1 has lock
b/c mPred.locked
== false

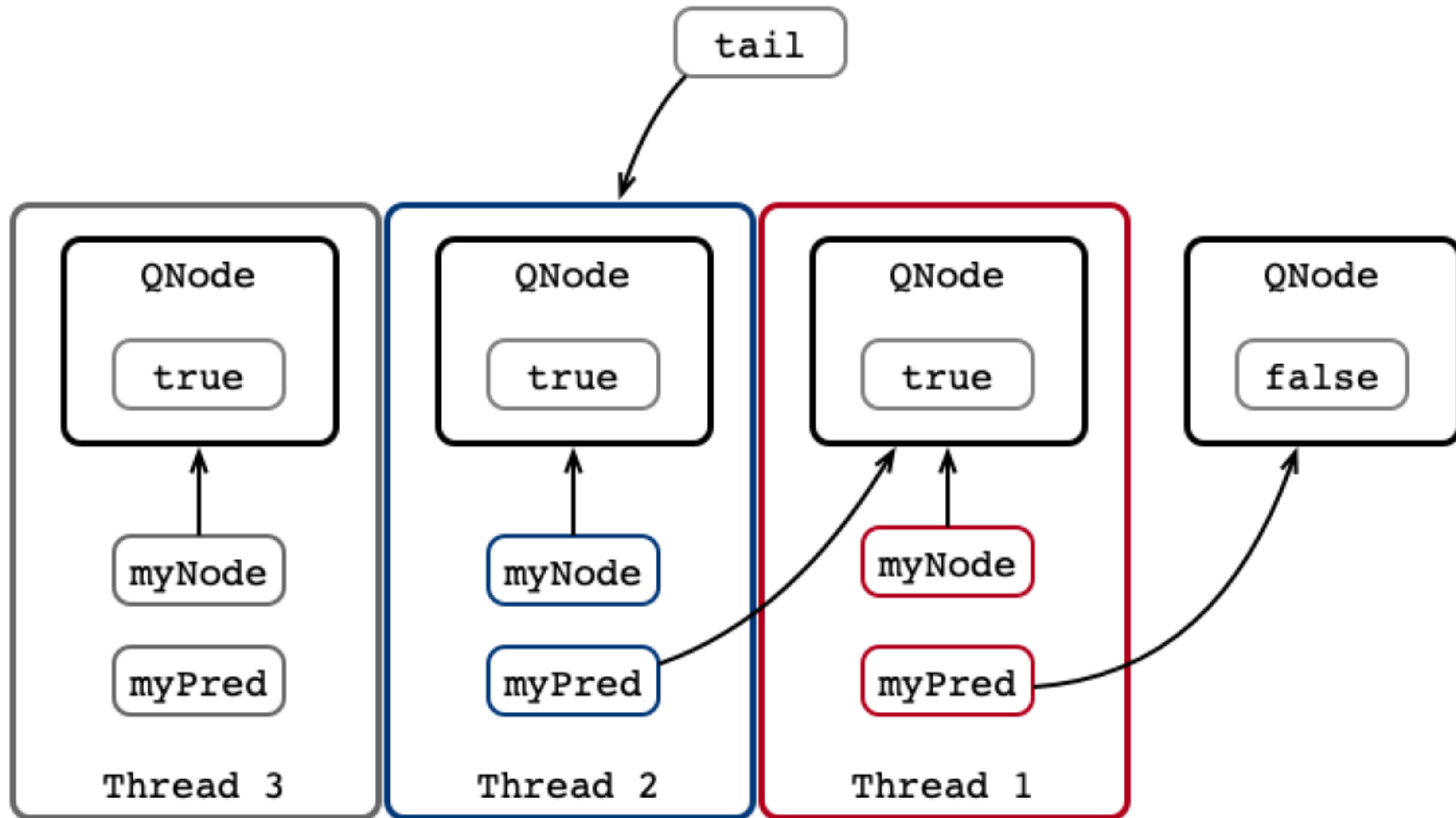
Thread 2 Arrives



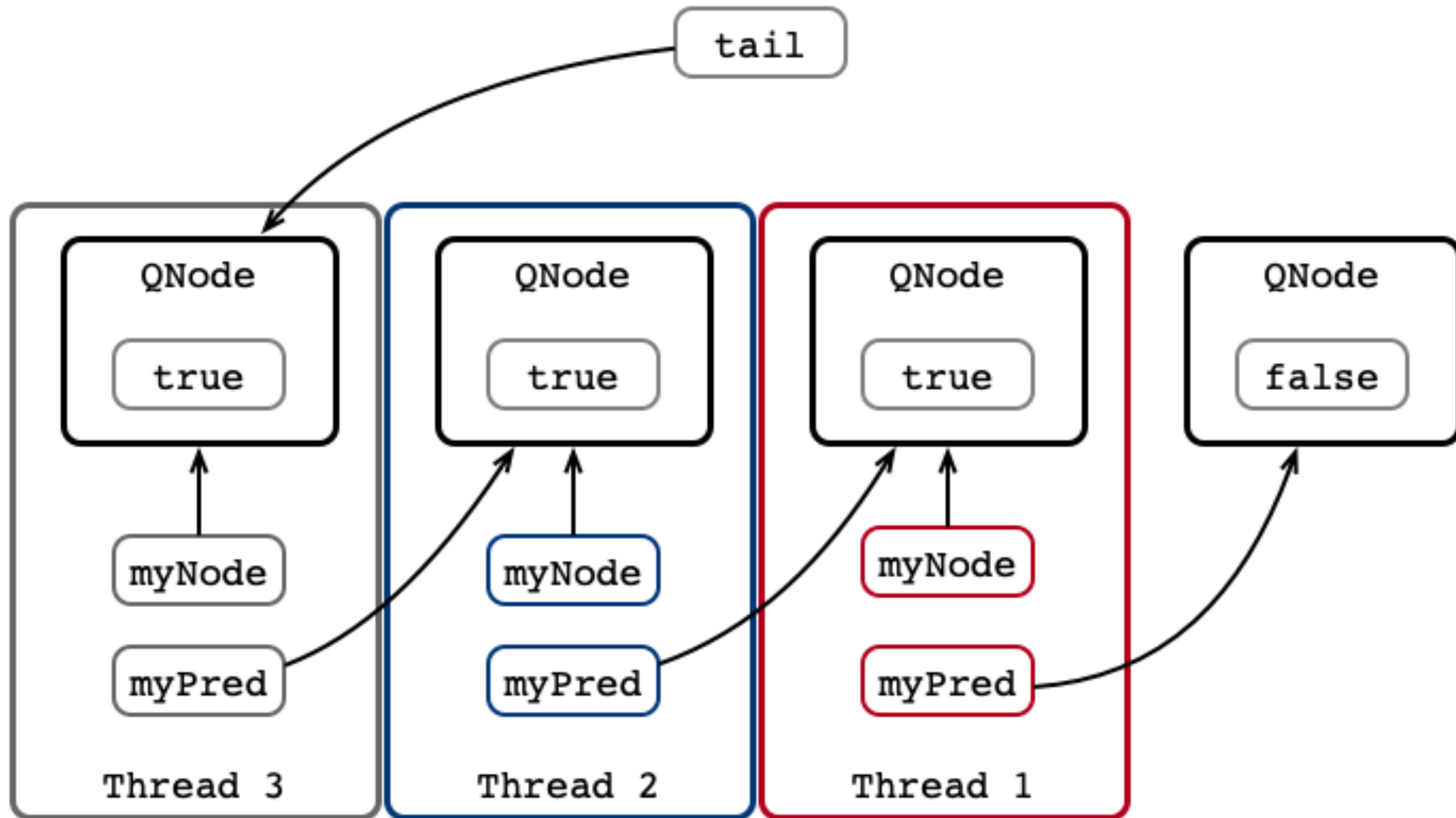
Thread 2 Locks



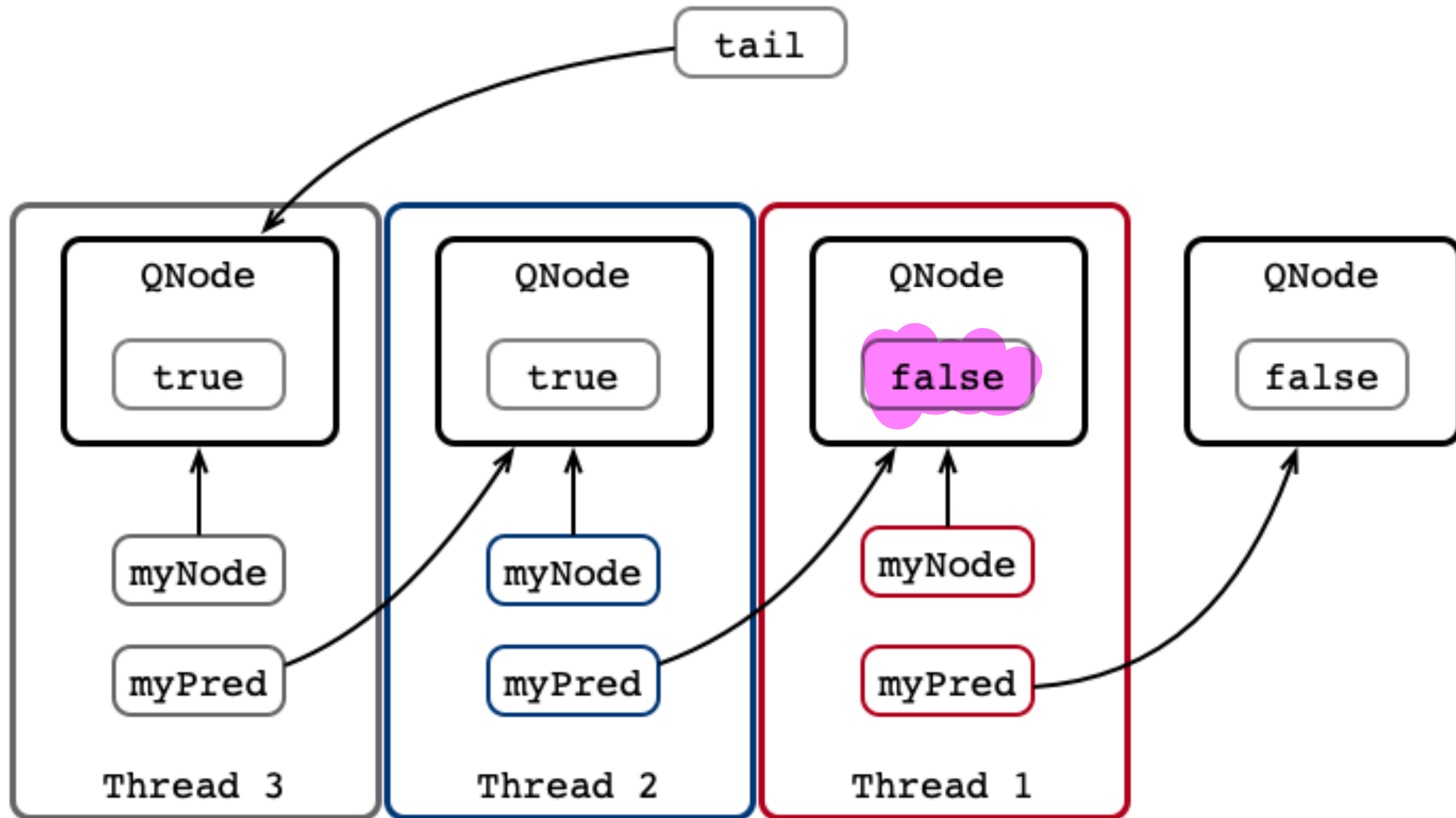
Thread 3 Arrives



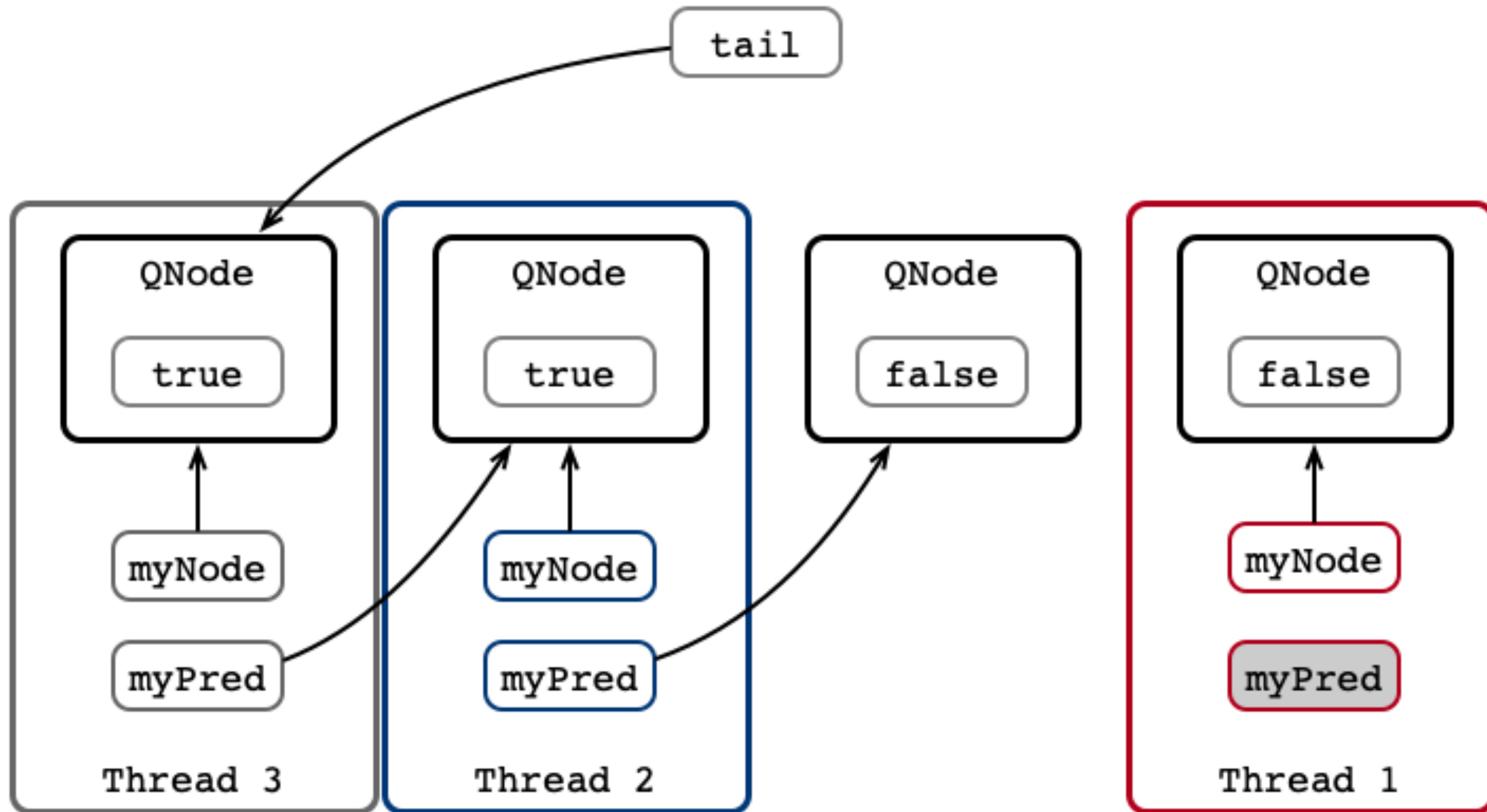
Thread 3 Locks



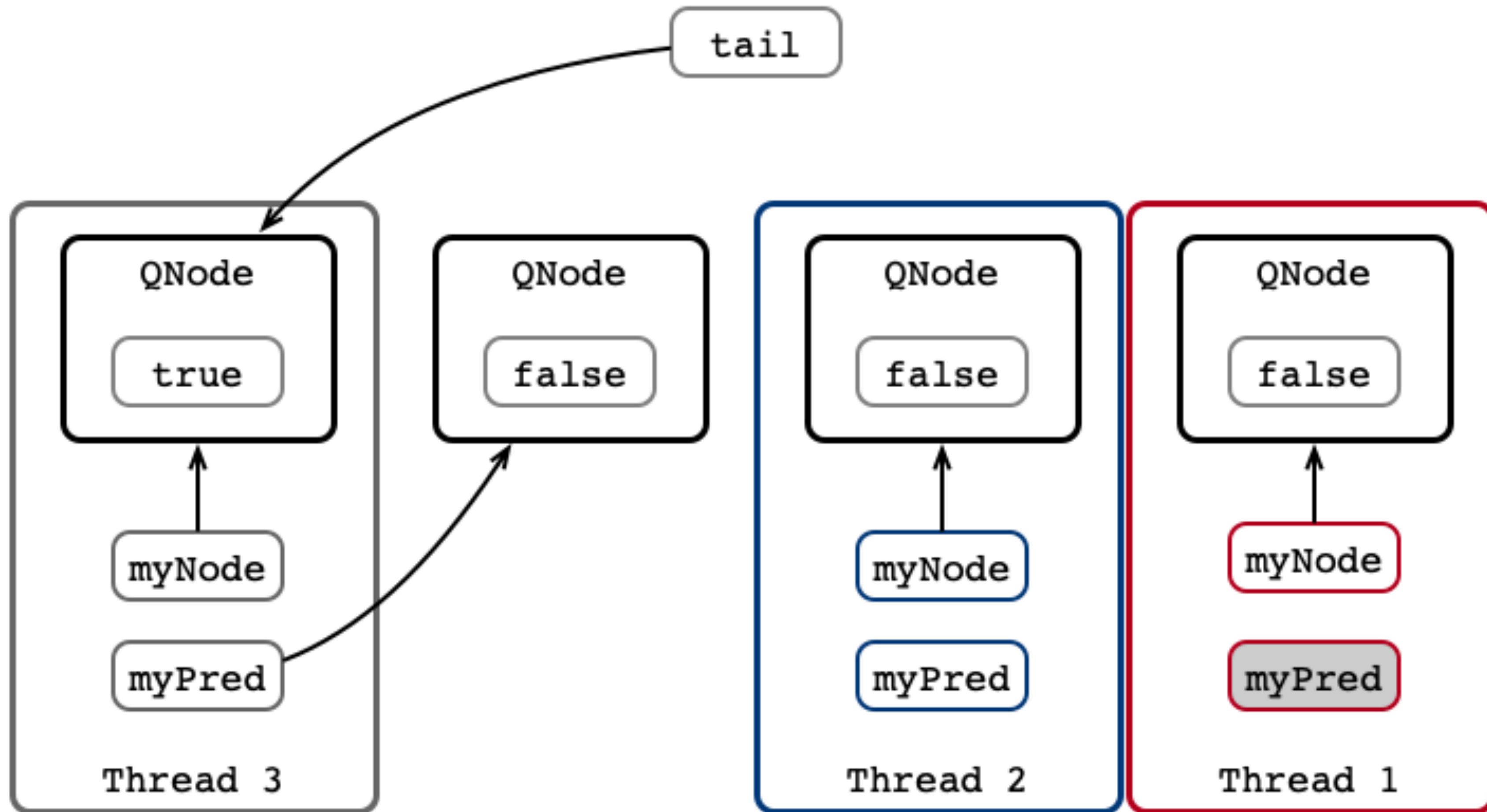
Thread 1 Unlocks (1)



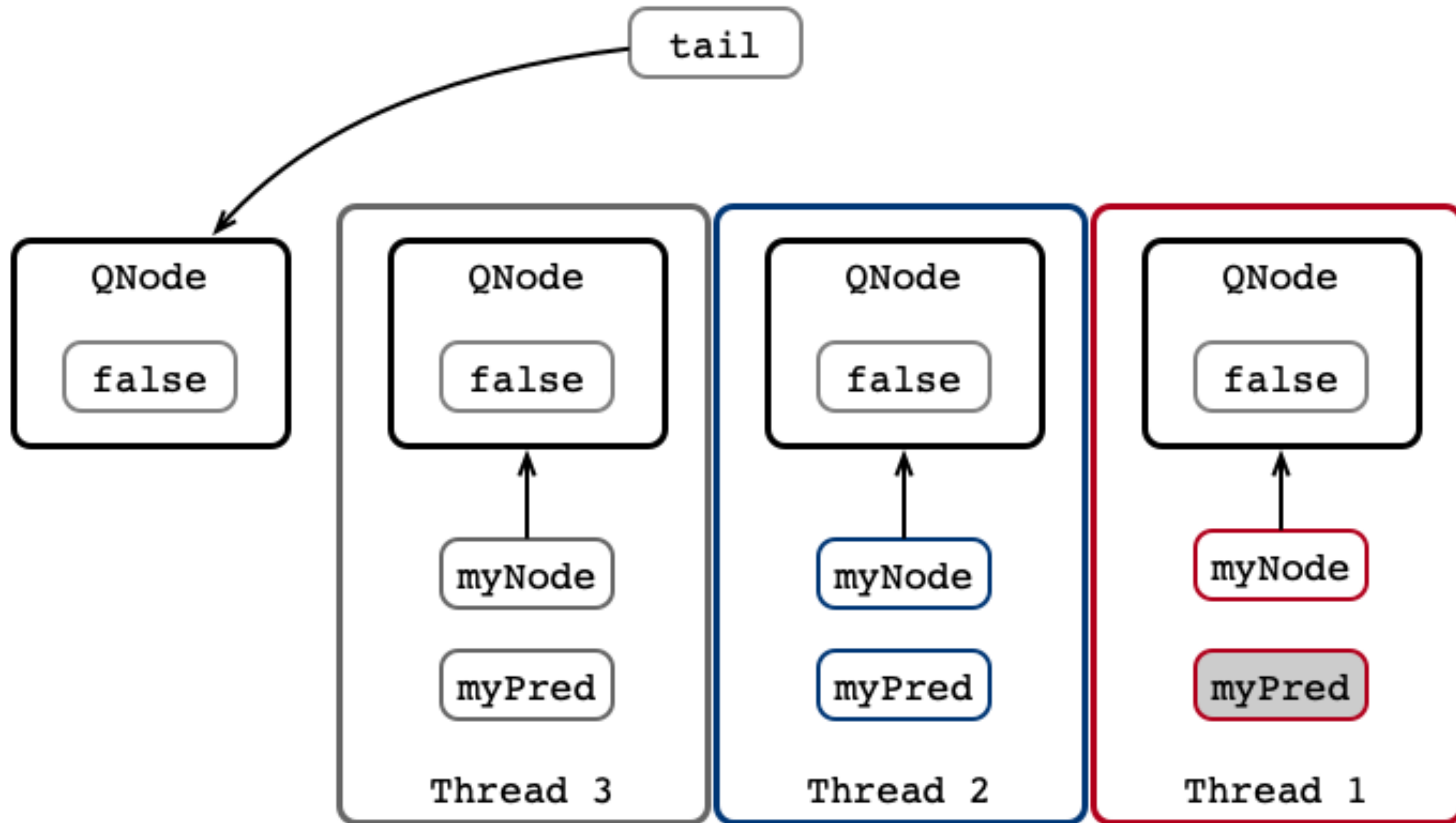
Thread 1 Unlocks (2)



Thread 2 Unlocks



Thread 3 Unlocks



CLHLock Implementation

```
import java.util.concurrent.atomic.AtomicReference;

public class CLHLock implements SimpleLock {
    AtomicReference<QNode> tail;
    ThreadLocal<QNode> myPred;
    ThreadLocal<QNode> myNode;

    private static class QNode {
        volatile boolean locked = false;
    }
}
```

each thread has own version

CLHLock Constructor

```java
public CLHLock () {
    tail = new AtomicReference<QNode>(new QNode());
    myNode = new ThreadLocal<QNode> () {
            @Override
            protected QNode initialValue() {
                return new QNode();
            }};
    myPred = new ThreadLocal<QNode>() {
            @Override
            protected QNode initialValue() {
                return null;
            }};}
```

create node for thread

init: pred is null

CLHLock Lock/Unlock

```
public void lock () {
    QNode qnode = myNode.get();
    qnode.locked = true;
    QNode pred = tail.getAndSet(qnode);
    myPred.set(pred);
    while (pred.locked) {};
}
```

```
public void unlock () {
    QNode qnode = myNode.get();
    qnode.locked = false;
    myNode.set(myPred.get());
}
```

Try It Yourself

- `clh-lock.zip`

Ugh

A Mystery. Why doesn’t the implementation work?

- deadlock for many threads?

Next Time

$105,097,565$ prime numbers!