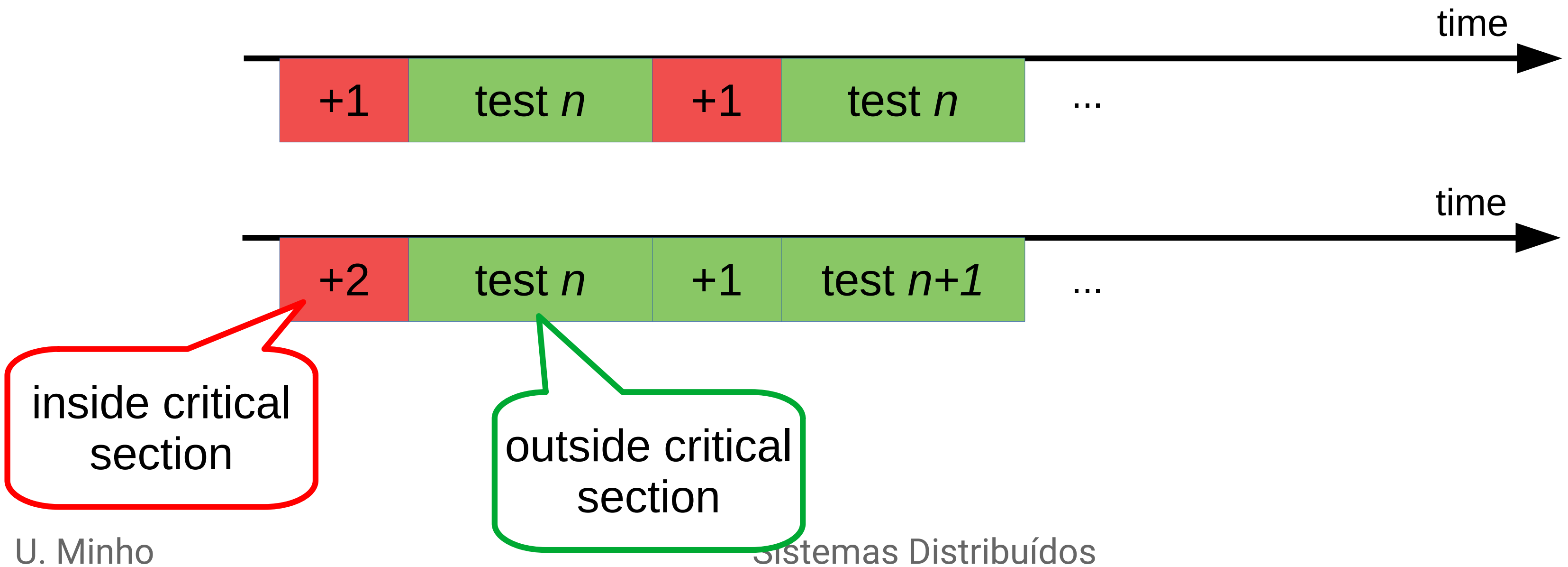
Sistemas Distribuídos

José Orlando Pereira

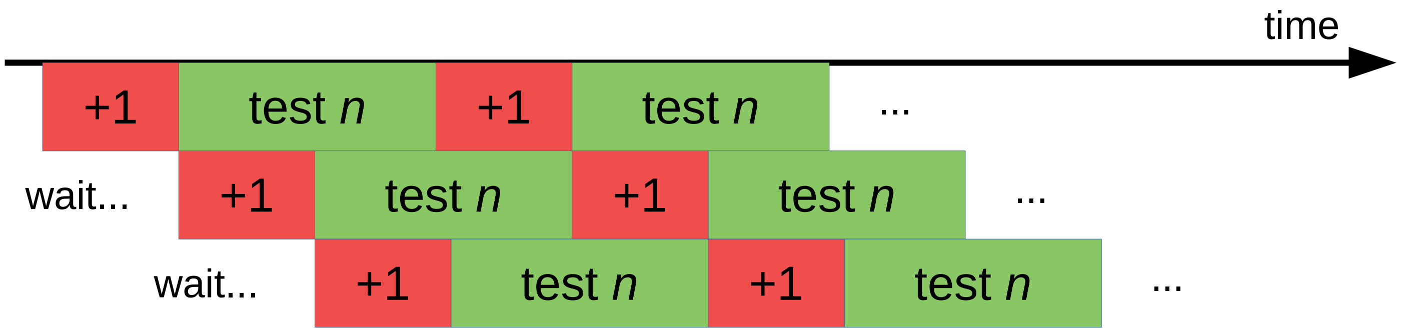
Departamento de Informática
Universidade do Minho

Motivation

- Consider two versions of the parallel primality testing code:
  - Increment +1 and get *n*, test *n*
  - Increment +2 and get *n*, test *n* and *n+1*

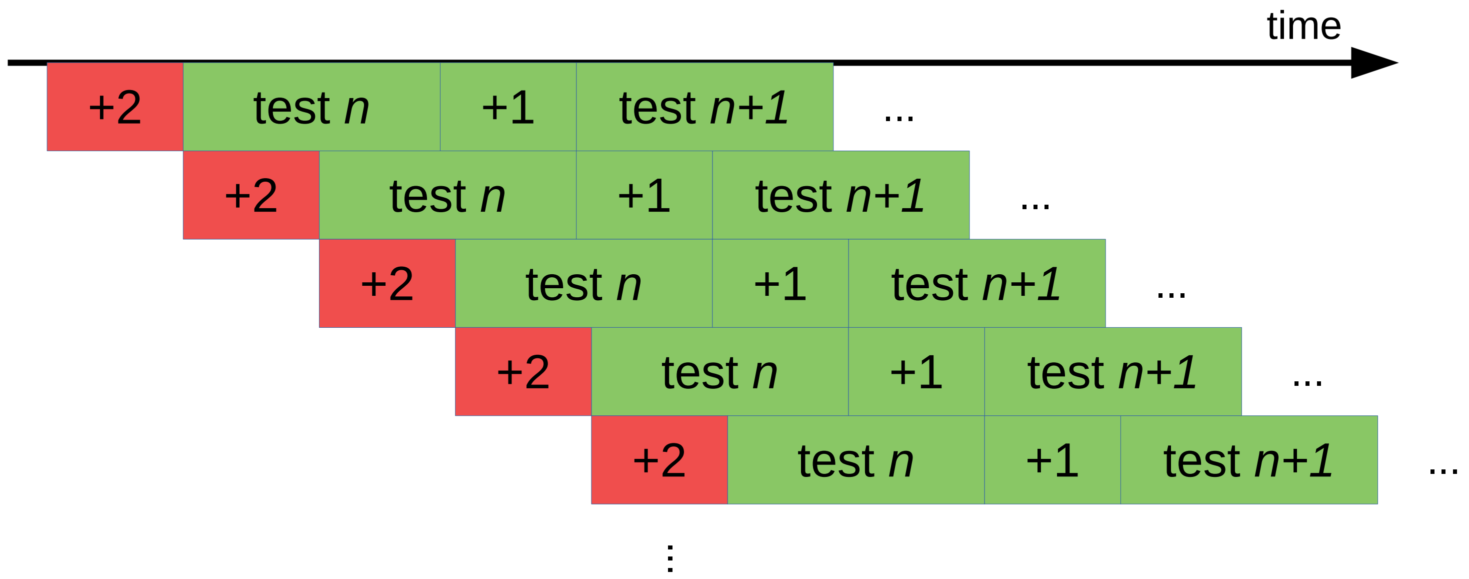


Motivation



- Eventually, at least one thread is blocked waiting for mutex...

Motivation

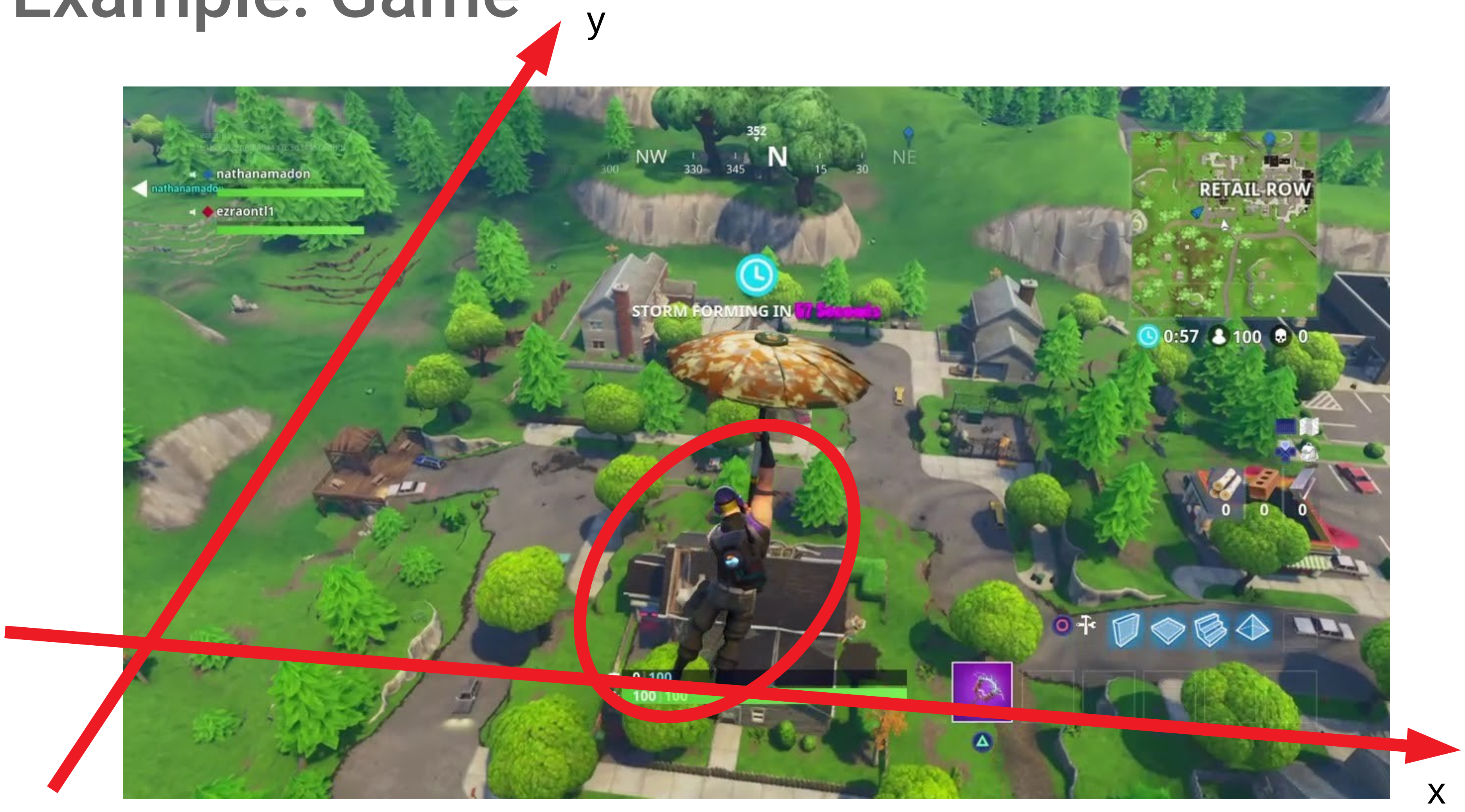


- Reducing the <u>contention</u> on critical sections lessens the performance impact of synchronization

Roadmap

- Use synchronization primitives to write correct concurrent code and avoid busy waiting
- Need to minimize **time** in critical sections
- Need to minimize **contention** in critical sections

Example: Game



Game state and operations

- State:
  - Map<String,Player> players;
  - class Player {
    ```
    class Player {
          int x,y;
          int life, score;
    }
    ```
- Operations:
  - drop in the game, move, and shoot
  - draw the game

First approach

- 1 thread for each player[*]
- 1 lock for the shared game state

[*] Later we make it distributed…

First approach

- void write(Output o) {

```
    try { l.lock();
    players.values().forEach(p→o.write(p.x, p.y));
    } finally { l.unlock(); }
}
```

Try/finally make it work with exceptions

Lengthy computation inside critical section

- Problems:
  - Either sending or moving
  - Writing may take a long time (blocking)
  - *“Lag”*...

Immutable objects

- class Coord { final int x, y; }
- class Player {
      Coord xy;
      int life, score;
  }
- void write(Output o) {

```
    try { l.lock();
    c=players.values().stream()
            .map(p→p.xy).collect(toList());
    } finally { l.unlock(); }
    c.forEach(c→o.write(c.x, c.y));
}
```

All fields final

Lengthy computation outside critical section

Multiple locks

- Can’t move two players concurrently
- Forget “drop in the game” for now…
- Use one lock for each player:
- class Player {

```
class Player {
    Lock l;
    Coord xy;
    int life, score;
}
```

Multiple locks

-
  ```
  void move(...) {
      try { l.lock();
      xy = new Coord(…);
      } finally { l.unlock(); }
  }
  ```

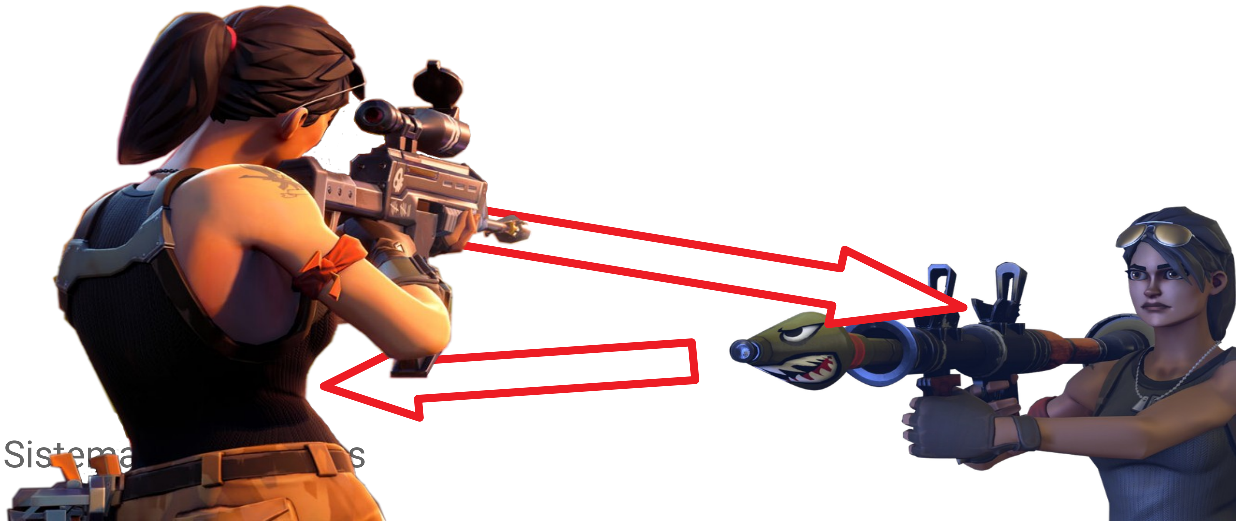
-
  ```
  Coord getLocation() {
      try { l.lock();
      return xy;
      } finally { l.unlock(); }
  }
  ```

Multiple locks

-
  ```
  void shoot(String sn, String tn) {
      Player s = players.get(sn);
      Player t = players.get(tn);
      try { s.l.lock(); t.l.lock();
          t.life--;
          s.score++;
      } finally { t.l.unlock(); s.l.unlock(); }
  }
  ```

Deadlock

- What if two players shoot at each other simultaneously (A → B and B → A) ?
- What if A → B, B → C and C → A?
- What if ...



Lock ordering

- What if two players A, B shoot at each other simultaneously?
  - A acquires A, B
  - B acquires A, B
- What if A → B, B → C and C → A?
  - A acquires A, B
  - B acquires B, C
  - C acquires A, C

Lock ordering

```
• void shoot(String sn, String tn) {
      Player s = players.get(sn);
      Player t = players.get(tn);
      try { Stream.of(sn,tn).sorted()
               .forEach(n→players.get(n).l.lock());
         t.life--;
         s.score++;
      } finally { t.l.unlock(); s.l.unlock(); }
  }
```

Acquire locks
in a fixed order

Release in
any order

Fairness

- “Doesn’t lock ordering mean that player A has an advantage?”
- No. It means that:
  - When A shoots some X and X shoots A, at the same time, the winner will be decided by lock of A
  - Any j.u.c.ReentrantLock is fair or, optionally, FIFO
- So they have the same chances regardless of the lock used

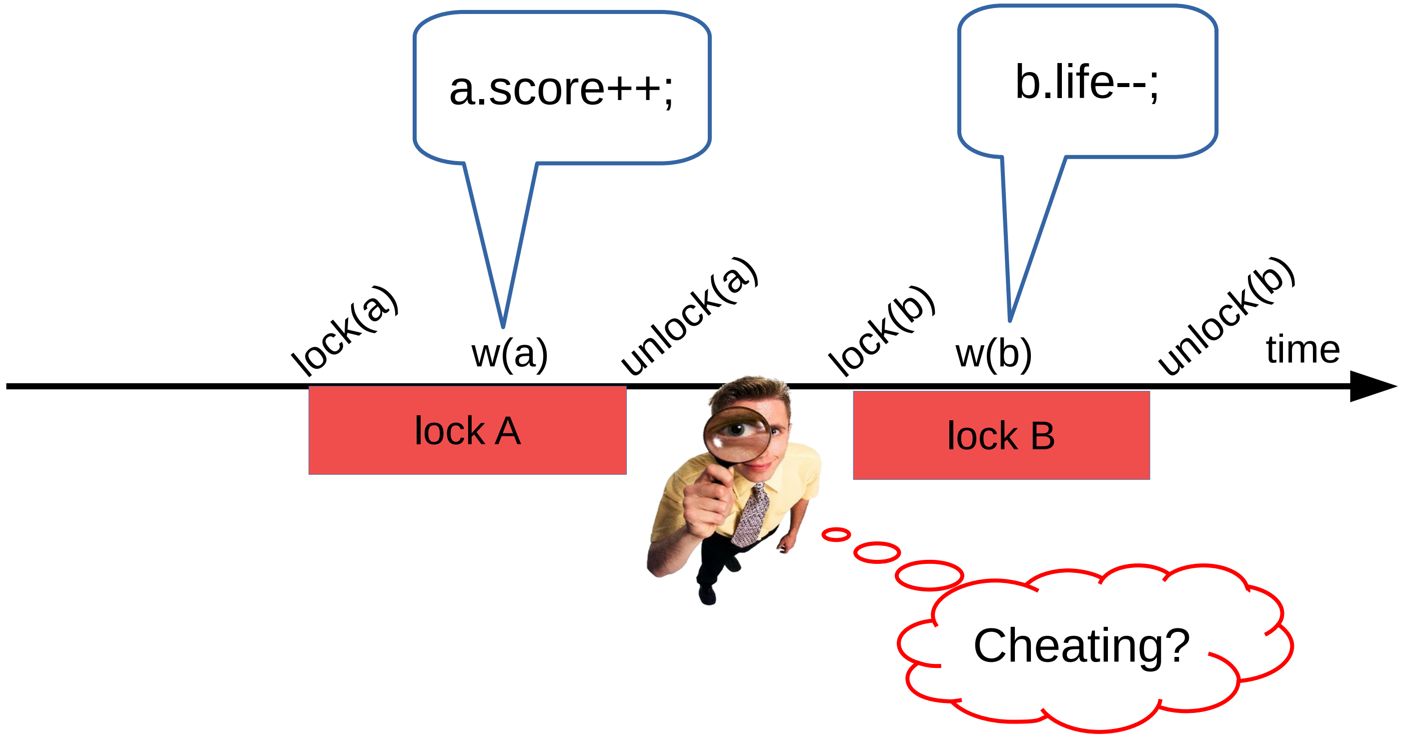
Multiple locks

- Acquiring all locks needed at the start and releasing them at the them of an operation works as well as single global lock

- What if we need to read some data before acquiring further locks?
- How to further reduce the time holding locks?

Multiple locks

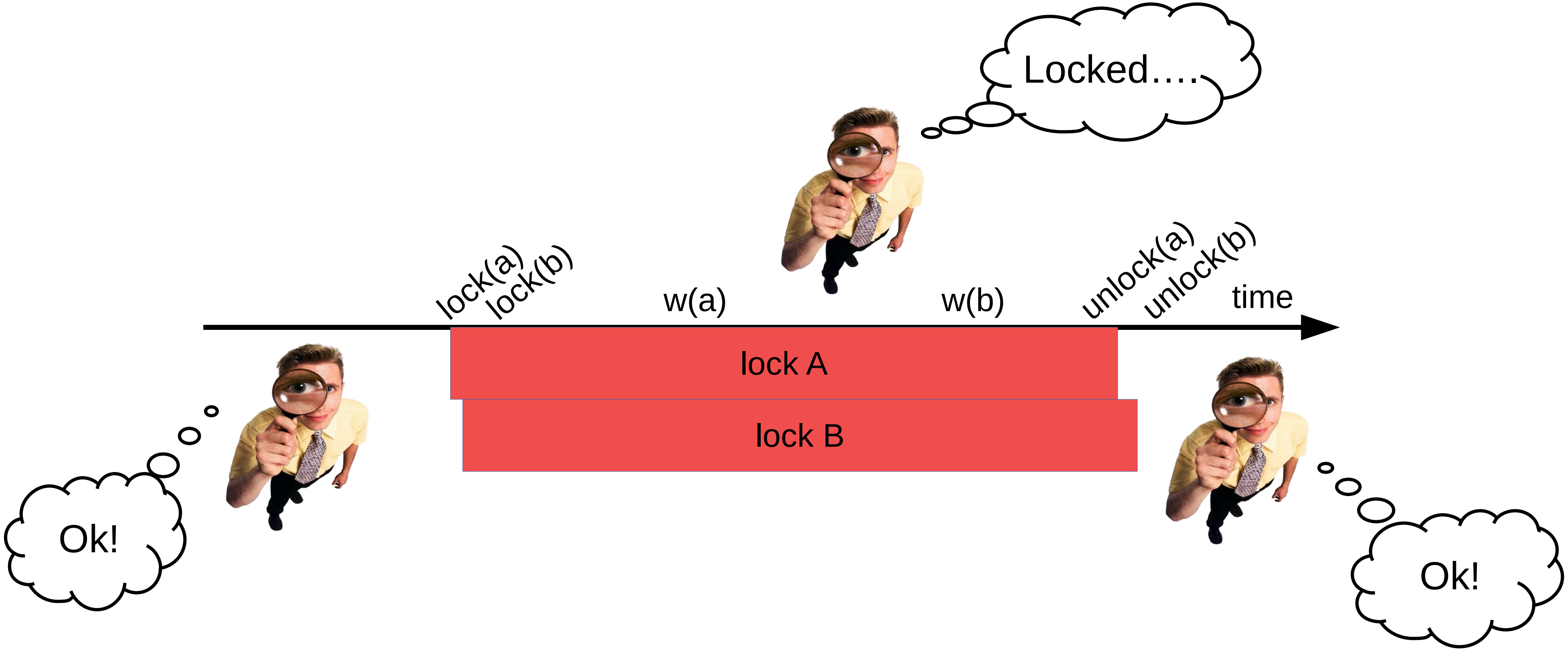
- Why acquire both locks simultaneously?
  - If we don’t….



Multiple locks

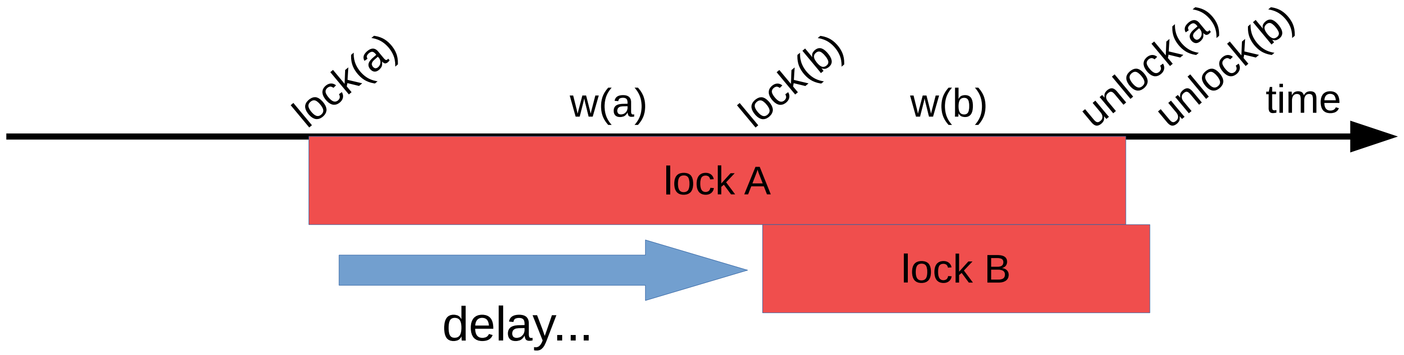
- Why acquire both locks simultaneously?

  (The observer will also lock A and B.)



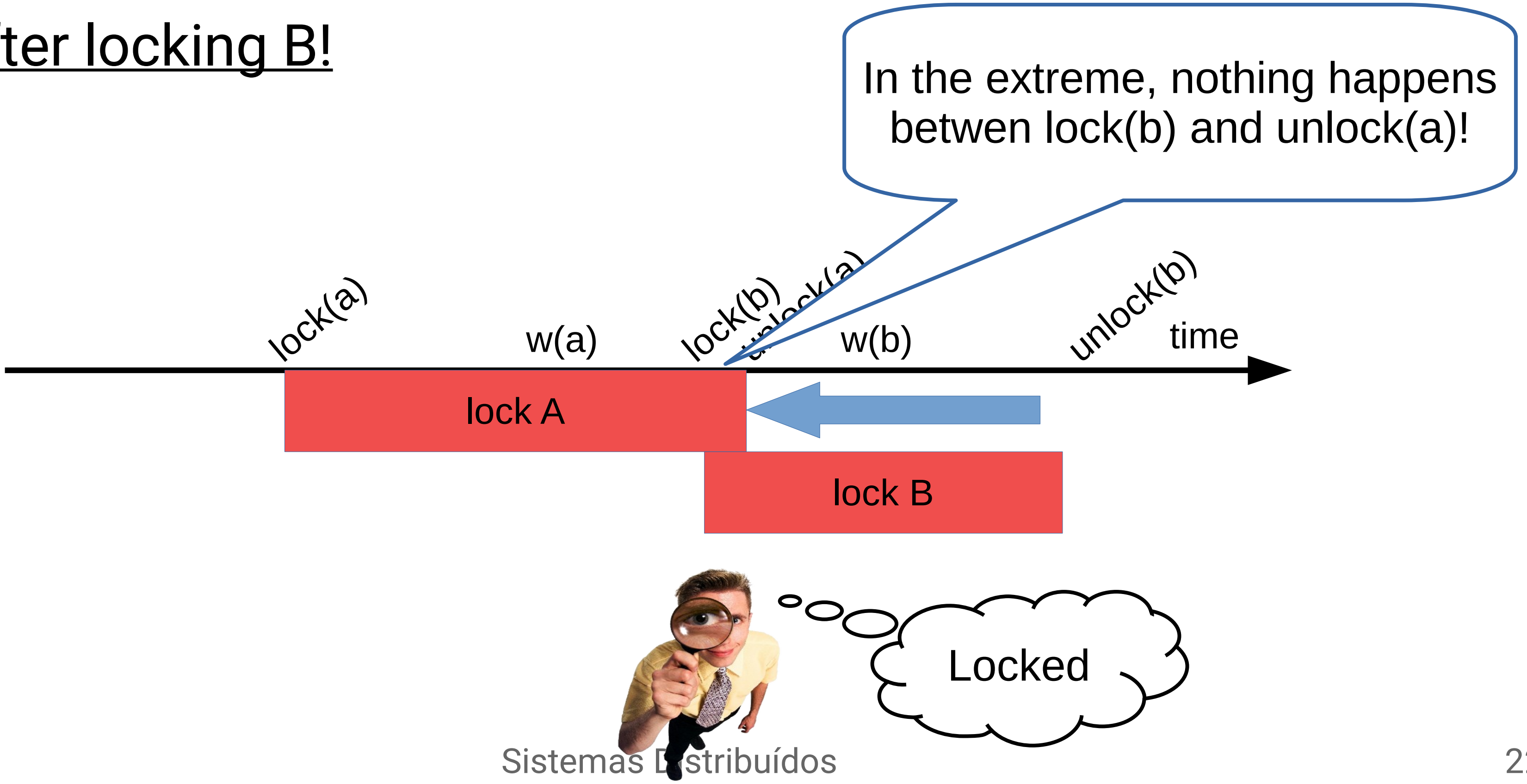
Lock later

- How much can we delay acquiring lock for B?
  - Until needed for modifying item b



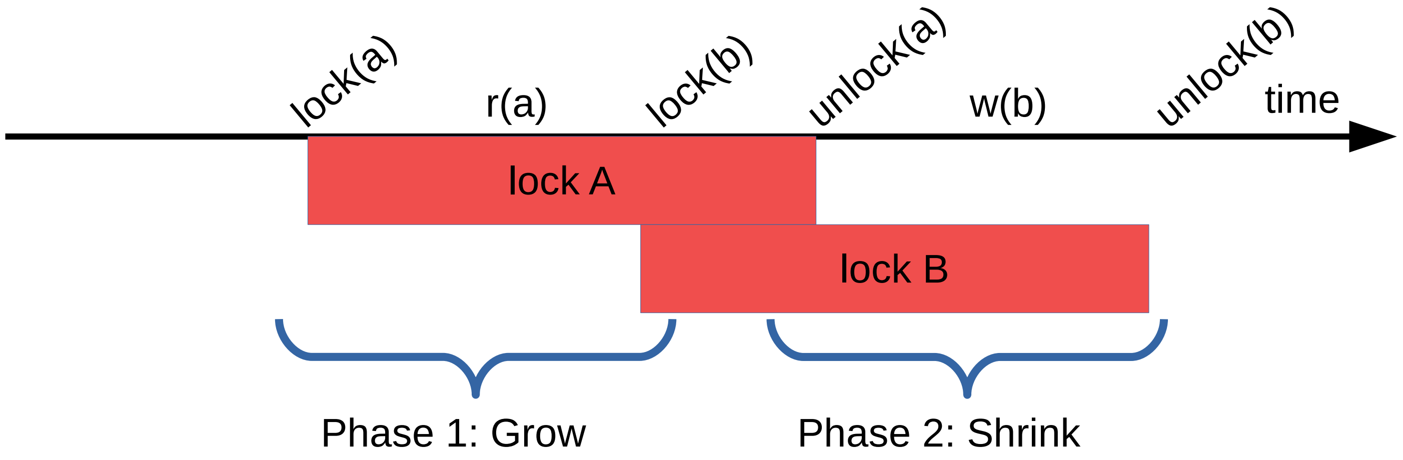
Unlock earlier

- How much can we anticipate releasing lock for A?
  - After modifying item a and…
  - after locking B!


In the extreme, nothing happens between lock(b) and unlock(a)!
lock(a)
w(a)
lock(b)
unlock(a)
w(b)
unlock(b)
time
lock A
lock B
Locked

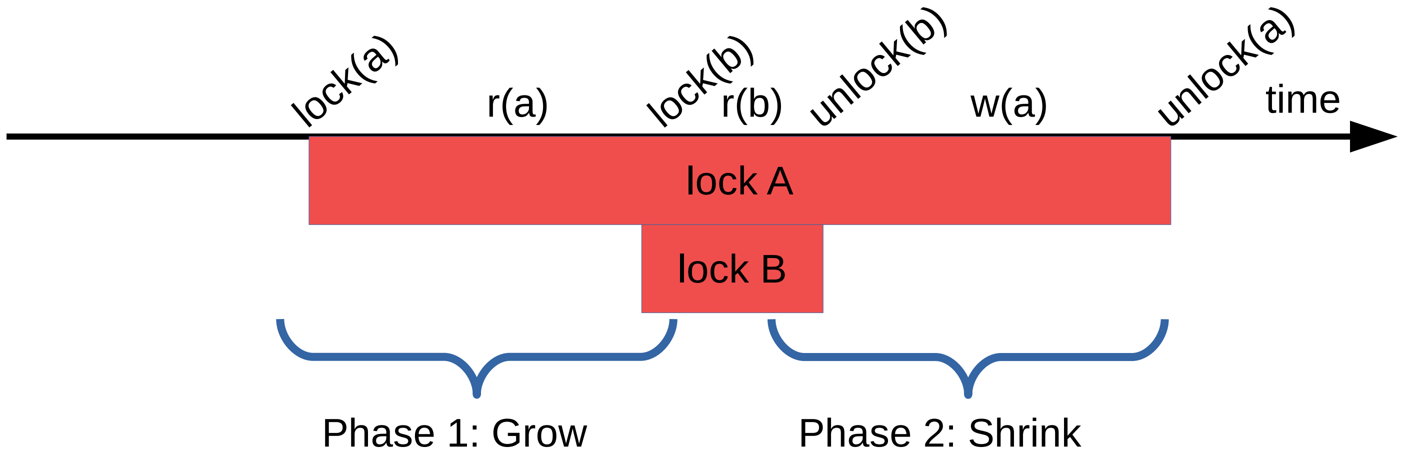
Two phase locking (2PL)

- **Rule 1**: All lock() precede all unlock()
- **Rule 2**: Each data item is read/written within the corresponding lock
  - Equivalent to holding all relevant locks, all the time



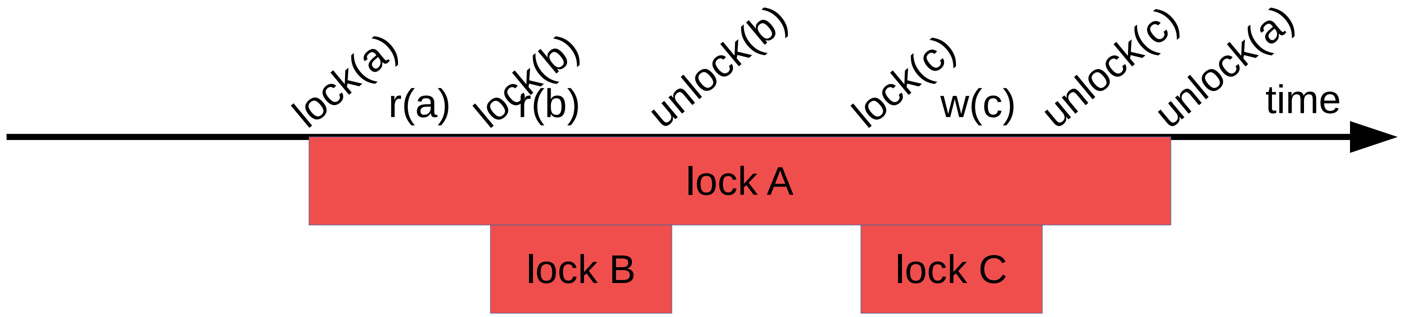
Two phase locking (2PL)

- Another example:

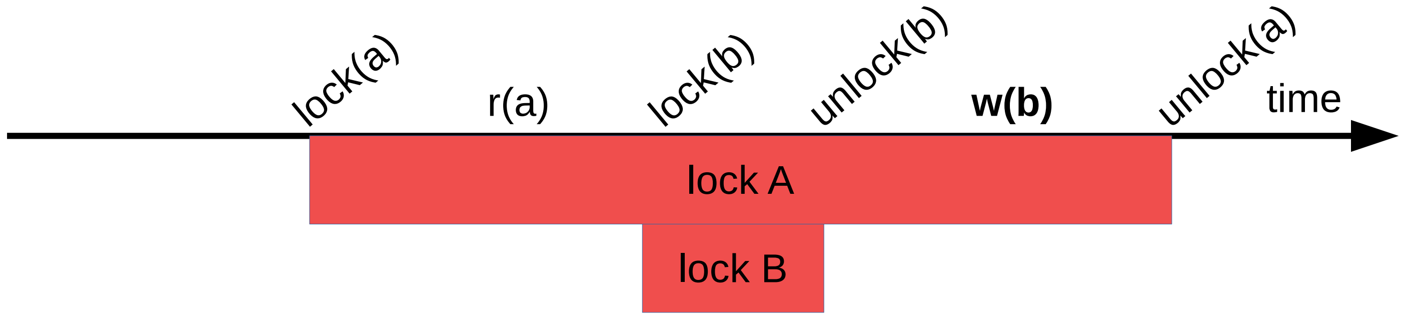


Two phase locking (2PL)

- Fails Rule 1:



- Fails Rule 2:



Two phase locking (2PL)

- void shoot(String sn, String tn) {

```
    Player s = players.get(sn);
    Player t = players.get(tn);
    Stream.of(sn,tn).sorted()
      .forEach(n→players.get(n).l.lock());
    t.life--;
    t.l.unlock();
    s.score++;
    s.l.unlock();
}
```

Phase 1: Grow

Phase 2: Shrink

Collection locking

- What if the collection is not immutable?
  - "drop in the game"
- Add back a global lock to game state...

Collection locking

```
• void shoot(String sn, String tn) {
      l.lock();
      Player s = players.get(sn);
      Player t = players.get(tn);
      Stream.of(sn,tn).sorted()
              .forEach(n→players.get(n).l.lock());
      t.life--;
      s.score++;
      t.l.unlock(); s.l.unlock();
      l.unlock();
  }
```

Collections with 2PL

- void shoot(String sn, String tn) {

```
    l.lock();
    Player s = players.get(sn);
    Player t = players.get(tn);
    Stream.of(sn,tn).sorted()
      .forEach(n→players.get(n).l.lock());
    l.unlock();
    t.life--;
    t.l.unlock();
    s.score++;
    s.l.unlock();
}
```

Is ordering needed?

Phase 1: Grow

Phase 2: Shrink

Collections with 2PL

- void shoot(String sn, String tn) {

```
    l.lock();
    Player s = players.get(sn);
    Player t = players.get(tn);
    s.l.lock();
    t.l.lock();
    l.unlock();
    t.life--;
    t.l.unlock();
    s.score++;
    s.l.unlock();
}
```

No, if these locks are always acquired in the context of the collection lock!

Conclusions

- Minimizing critical sections is key to performance and scale
- Strategies to reduce impact of critical sections:
  - Immutable objects
  - Granular locking
  - Two phase locking
    - Collections
- Avoid deadlocks by using a fixed locking order