Sistemas Distribuídos

José Orlando Pereira

Departamento de Informática
Universidade do Minho

2018/2019

Example: Game

Game state and operations

- State:
  - Map<String,Player> players;
  - 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 draw() {
  try { l.lock();
  players.values().forEach(p→Draw3D(p.x, p.y));
  } finally { l.unlock(); }
  }
- Problems:
  - Either drawing or moving
  - Drawing takes a long time
  - *“Lag”...*

Try/finally make it work with exceptions

Lengthy computation inside critical section

Immutable objects

- class Coord { final int x, y; }
- class Player {
  Coord xy;
  int shots, score;
  }
- void draw() {
  try { l.lock();
  c=players.values().stream()
  .map(p→p.xy).collect(toList());
  } finally { l.unlock(); }
  c.forEach(c→Draw3D(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 {
  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.u
  }

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
  - Threads acquiring the same lock are FIFO ordered (with j.u.c. Locks)
- So the first to arrive gets the lock, regardless of the lock used

Collection locking

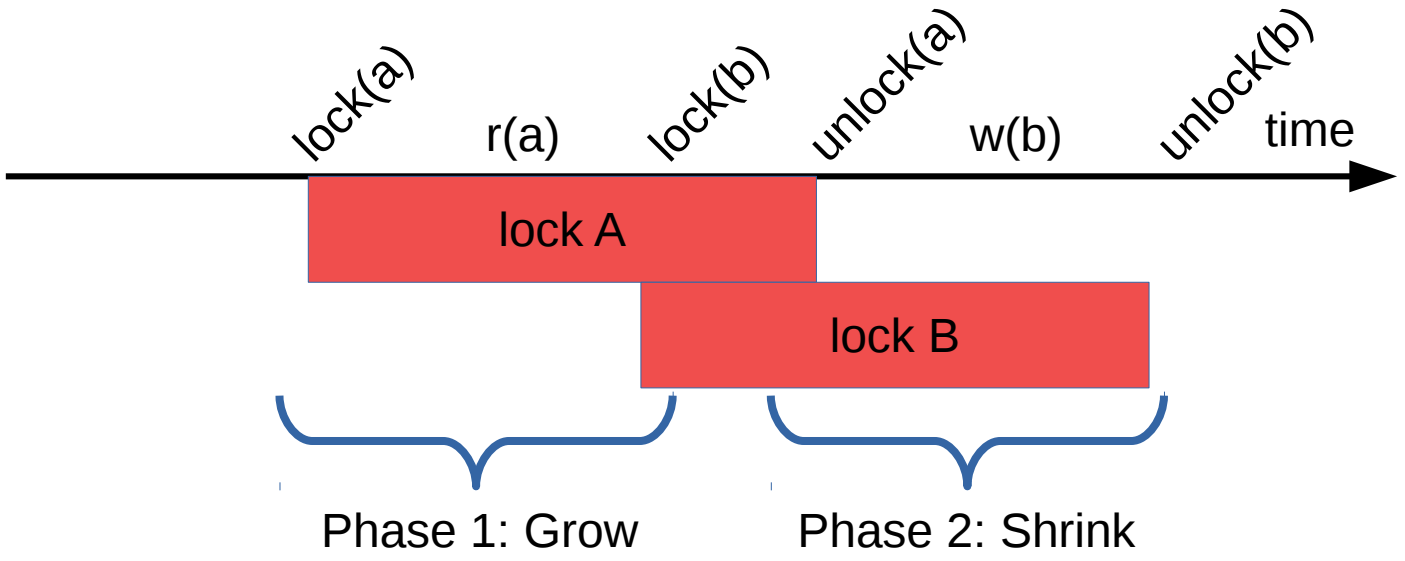
- What if the collection is not immutable?
  - “drop in the game”
  - remove when remaining life == 0
- Add back a global lock to game state...

Collection locking

```
• void shoot(String sn, String tn) {
      try { l.lock();
        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(); }
      } finally { l.unlock(); }
  }
```

Two phase locking

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

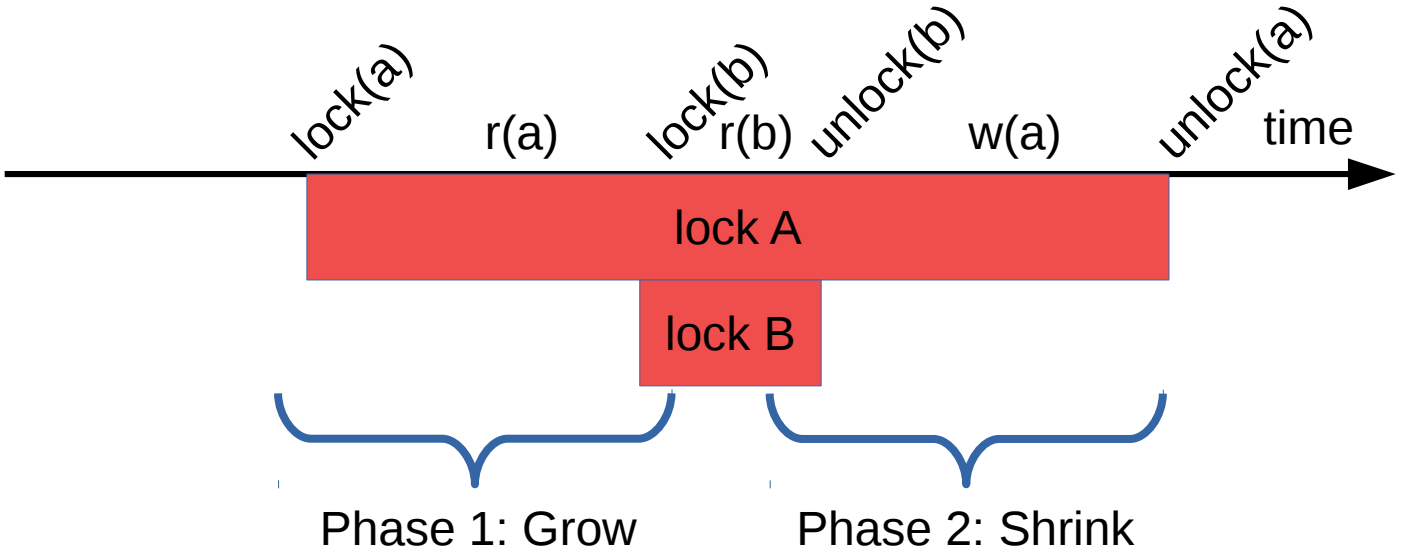


Locks vs Variables

- “Which lock corresponds to each data item?”
- Multiple threads accessing some data item concurrently must have acquired the same lock
- It is up to the program to ensure this!
- Typical solution:
  - Keep variables and the corresponding lock encapsulated within the same object
  - This is the solution encouraged by Monitors

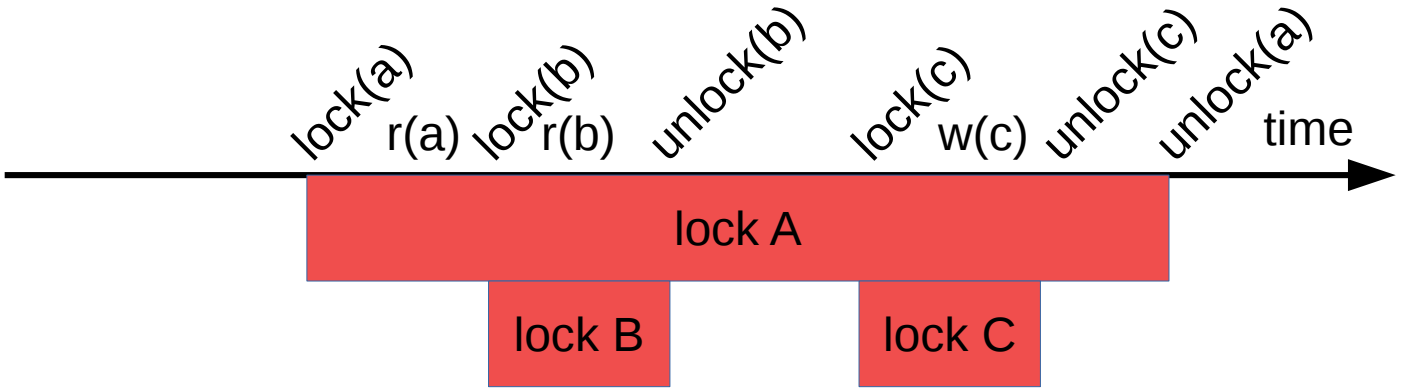
Two phase locking

- Another example:

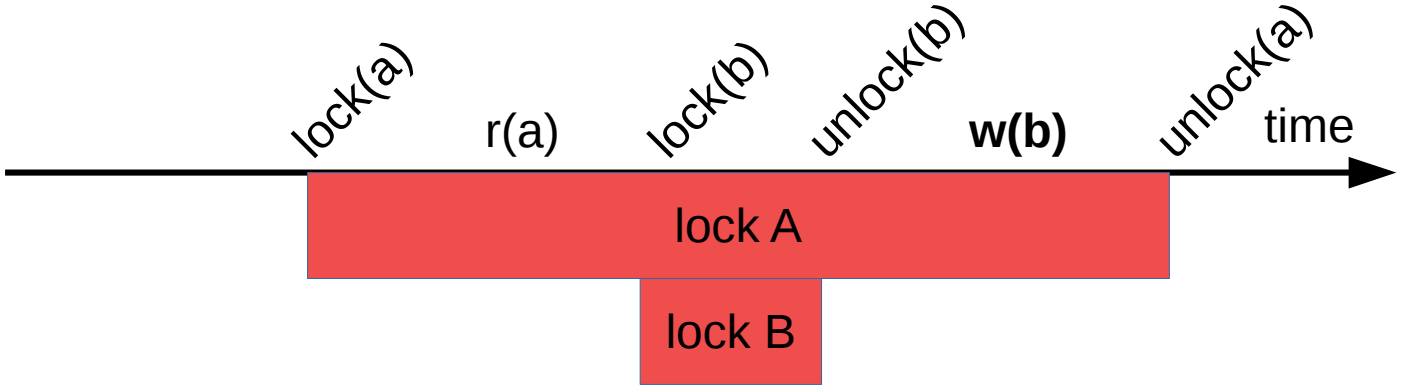


Two phase locking

- Fails Rule 1:

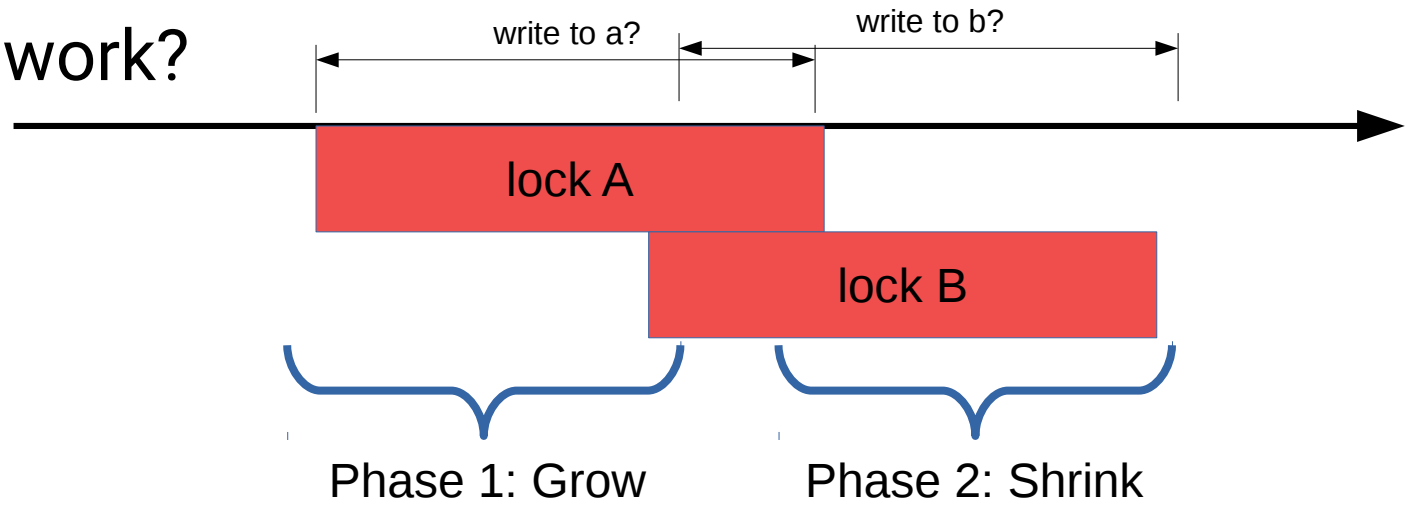


- Fails Rule 2:



Two phase locking

- Why does it work?

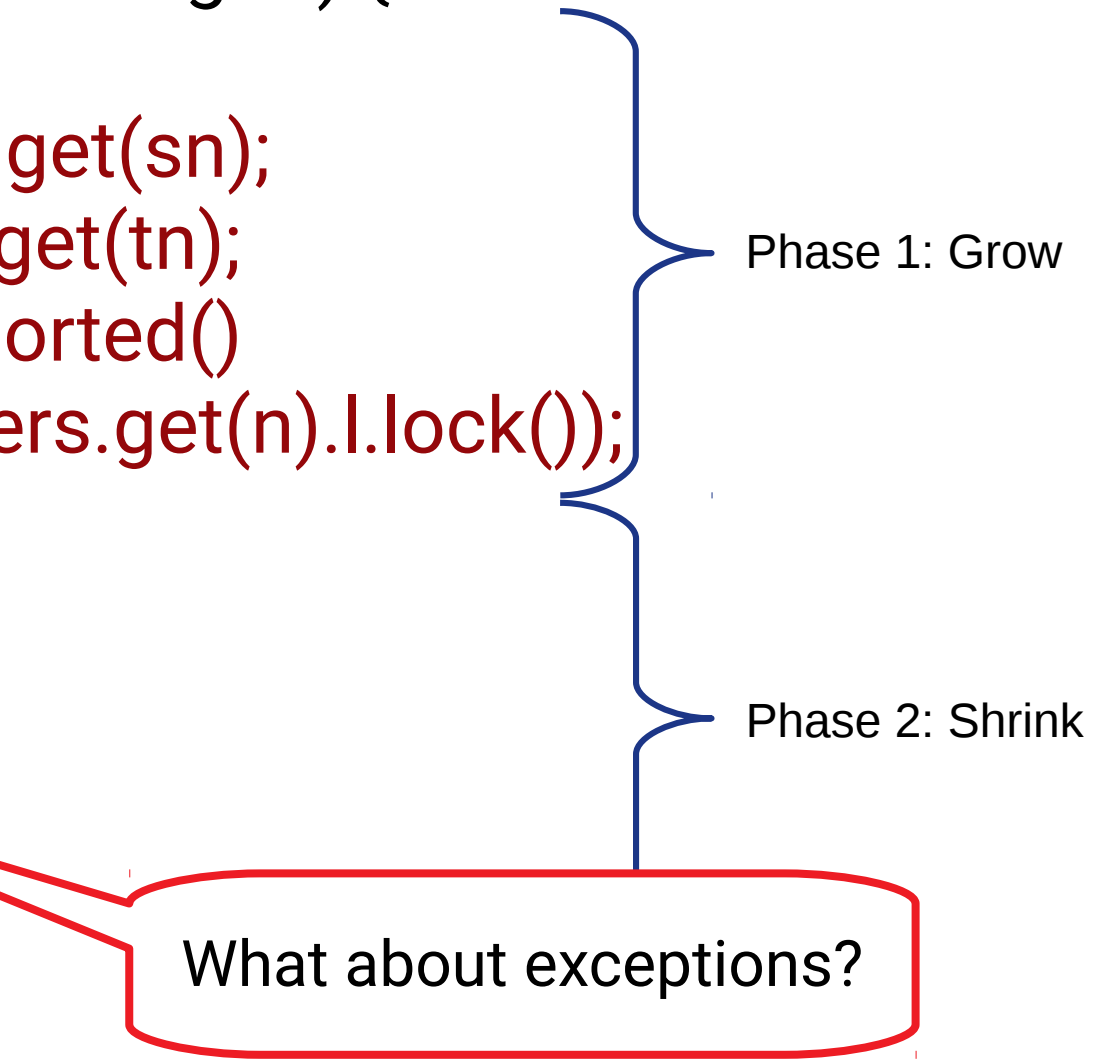


- An observer of a (resp. b) cannot tell where a (resp. b) is changed, as it is hidden by lock
  - Cannot deny that it happens in between phases
- Then, cannot tell the difference from all changes happening in the period with all locks acquired...
- … which is known to work!

Two phase 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());
    l.unlock();
    t.life--;
    t.l.unlock();
    s.score++;
    s.l.unlock();
}
```



Conclusions

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

j.u.c Locks vs Monitors

```
class C {
    private int i;

    synchronized public void m() {

        i++;

    }
}
```

There is a hidden “lock” in each object used by “synchronized”

```
class C {
    private int i;
    private Lock l =
        new ReentrantLock();

    public void m() {
        try { l.lock();
        i++;
        } finally { l.unlock(); }
    }
}
```

Equivalent code (aproximately...)

j.u.c. Locks vs Monitors

- Main differences, for now:
  - Synchronized blocks are always exited in LIFO order
    vs.
    j.u.c. Locks can be unlocked in any order
    - To take advantage of two phase locking
  - Threads waiting for a synchronized block enter in any order
    vs.
    Threads waiting for a j.u.c. Lock enter in FIFO order
- More later...