Ordered Locking

CSE 334

Deadlock

- Happens when there’s a set of threads where
  - each thread waits for a resource held by another thread.

Deadlock

- Happens when there's a set of threads where
  - each thread waits for a resource held by another thread.

No-deadlock interleaving

```
// thread 1          // thread 2
a.lock();
b.lock();
drop(a);
drop(b);
                     b.lock();
                     a.lock();
                     drop(a);
                     drop(b);
```

Deadlock interleaving

```
// thread 1          // thread 2
a.lock();
                     b.lock();
b.lock();
                     a.lock();
drop(a);
drop(b);
                     drop(a);
                     drop(b);
```

Deadlock

- Happens when there's a set of threads where
  - each thread waits for a resource held by another thread.

No-deadlock interleaving

```
// thread 1          // thread 2
a.lock();
b.lock();
drop(a);
drop(b);
                     b.lock();
                     a.lock();
                     drop(a);
                     drop(b);
```

Deadlock interleaving

```
// thread 1          // thread 2
a.lock();
                     b.lock();
b.lock();
                     a.lock();
drop(a);
drop(b);
                     drop(a);
                     drop(b);
```

- Tricky to debug. What if it only deadlocks 1% of the time?

What code is deadlock-prone?

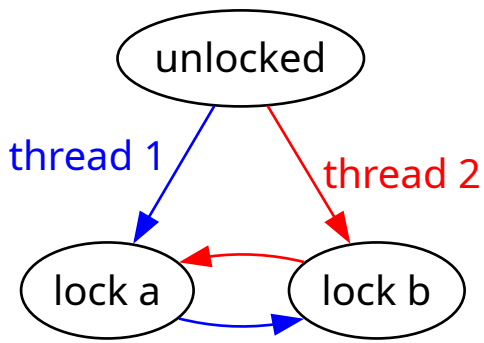
- Let’s draw a graph, where each path represents a possible order of locking.

What code is deadlock-prone?

- Let’s draw a graph, where each path represents a possible order of locking.

```
// thread 1
a.lock();
b.lock();
```

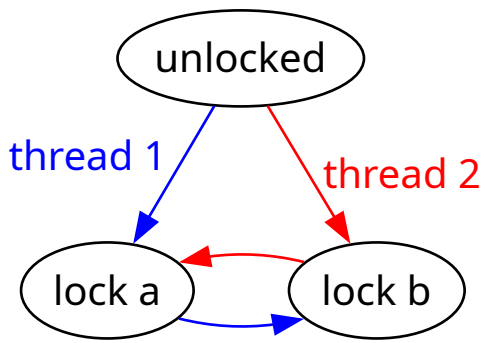
```
// thread 2
b.lock();
a.lock();
```



What code is deadlock-prone?

- Let’s draw a graph, where each path represents a possible order of locking.

```
// thread 1          // thread 2
a.lock();            b.lock();
b.lock();            a.lock();
```



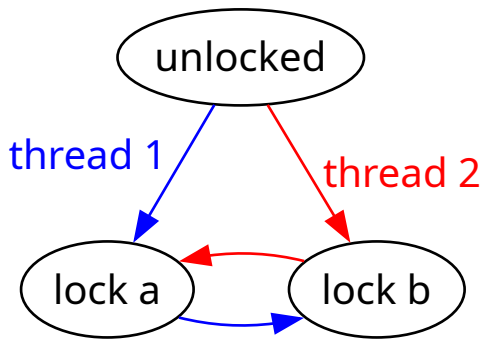
- What property of this graph tells us this can deadlock?
-
-

What code is deadlock-prone?

- Let’s draw a graph, where each path represents a possible order of locking.

```
// thread 1
a.lock();
b.lock();
```

```
// thread 2
b.lock();
a.lock();
```

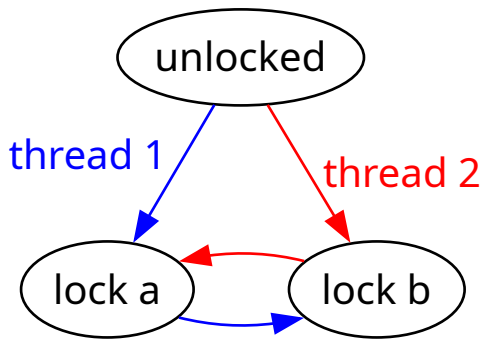


- What property of this graph tells us this can deadlock?
- The cycle!
-

What code is deadlock-prone?

- Let’s draw a graph, where each path represents a possible order of locking.

```
// thread 1          // thread 2
a.lock();            b.lock();
b.lock();            a.lock();
```

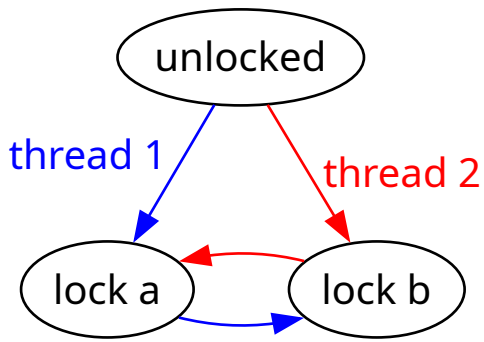


- What property of this graph tells us this can deadlock?
- The cycle!
- What rules can programmers follow to prevent cycles?

What code is deadlock-prone?

- Let’s draw a graph, where each path represents a possible order of locking.

```
// thread 1          // thread 2
a.lock();            b.lock();
b.lock();            a.lock();
```



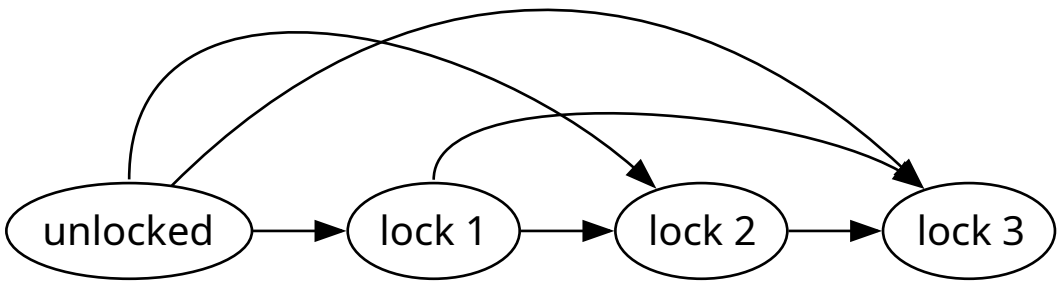
- What property of this graph tells us this can deadlock?
- The cycle!
- What rules can programmers follow to prevent cycles?

Lock Ordering

- We can assign the locks an order.

Lock Ordering

- We can assign the locks an order.



Hello

```
fn hi() {
  println!("SUP");
}
```