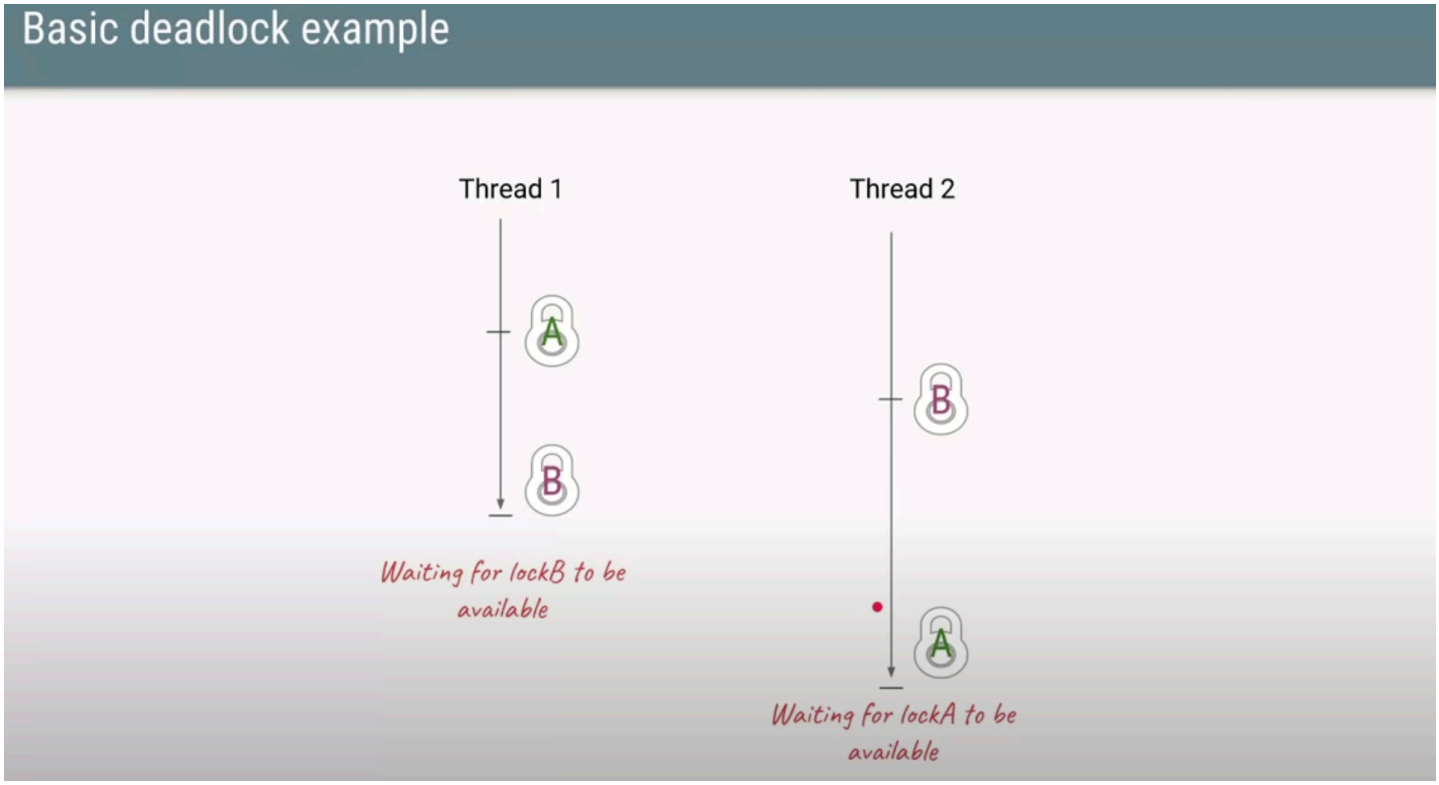
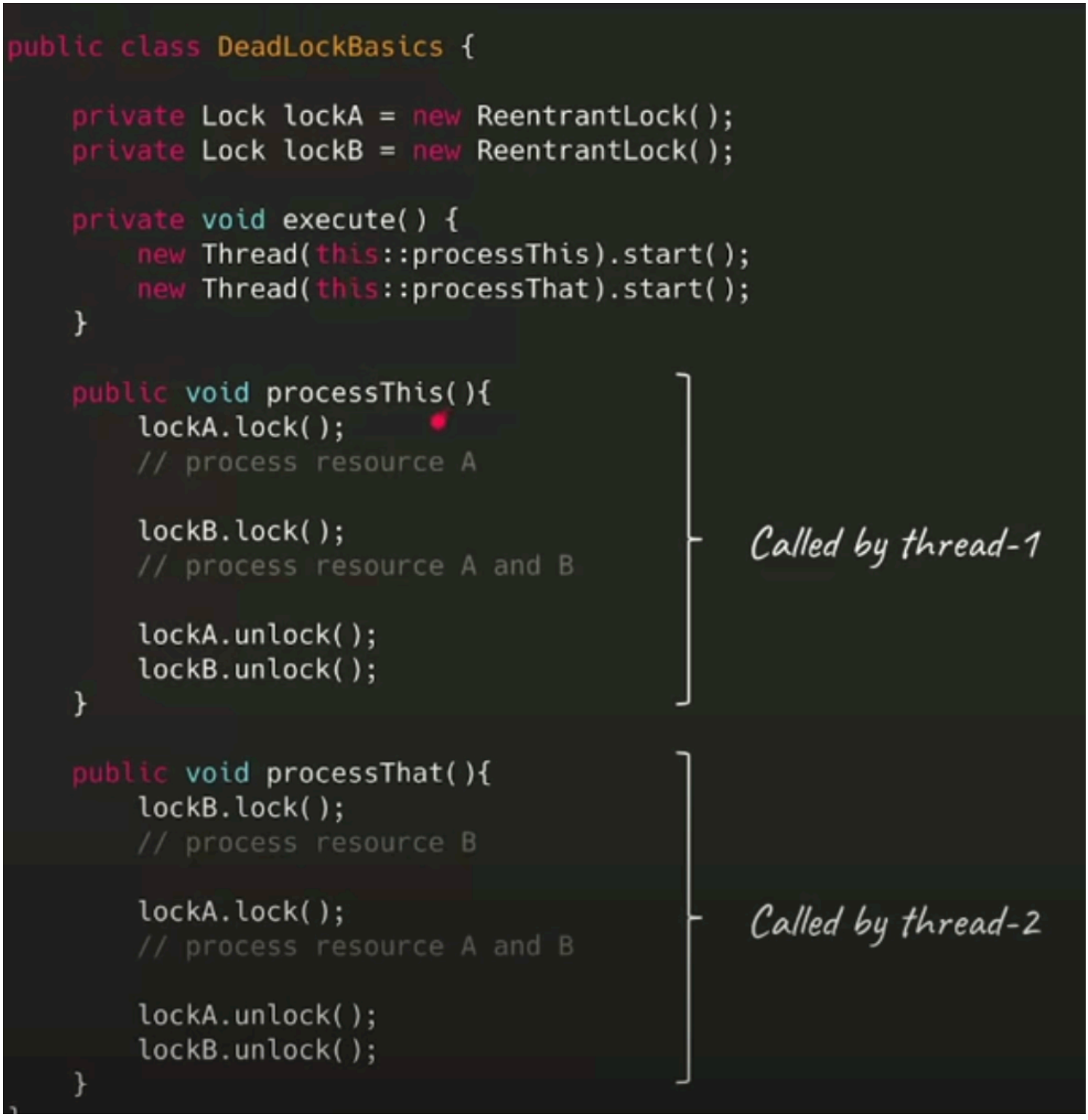
Multithreading 12: Dead Lock

How to detect and resolve deadlock:

What is deadlock:





It is not easy to detect deadlock.

Locks are everywhere

```
private void execute() throws InterruptedException {

    Lock lock = new ReentrantLock();
    lock.lock();

    BlockingQueue queue = new ArrayBlockingQueue(16);
    queue.take();

    Semaphore sem = new Semaphore(1);
    sem.acquire();
}

public synchronized void process() {
}

public static synchronized int count() {
}
```

Lock on 'this'

Lock on class

Threads are everywhere & they are non-deterministic

```java
private void execute() {

    Thread t1 = new Thread();
    t1.start();

    ExecutorService pool = Executors.newFixedThreadPool(10);
    pool.submit(() -> { /** task **/});

    ScheduledExecutorService schedulers = Executors.newScheduledThreadPool(1);
    schedulers.schedule(() -> { /** task **/}, 10, TimeUnit.SECONDS);
}

@RequestMapping("/user/32")
public void userDetails() {

}
```

CPU scheduling makes it difficult to know when an instruction is run

ThreadPools used by frameworks (hidden from developers)

HOw to detect deadlock at runtime?

1. Thread dump and similar tools
2. Java MX BEan

Thread Dumps (kill -3, or JStack)

```
Found one Java-level deadlock:
=============================
"Thread-1":
  waiting for ownable synchronizer 0x00000000d723cb28, (a java.util.concurrent.locks.ReentrantLock$NonfairSync),
  which is held by "Thread-0"
"Thread-0":
  waiting for ownable synchronizer 0x00000000d723cb58, (a java.util.concurrent.locks.ReentrantLock$NonfairSync),
  which is held by "Thread-1"

Java stack information for the threads listed above:
===================================================
"Thread-1":
        at sun.misc.Unsafe.park(Native Method)
        - parking to wait for  <0x00000000d723cb28> (a java.util.concurrent.locks.ReentrantLock$NonfairSync)
        at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)
        at java.util.concurrent.locks.AbstractQueuedSynchronizer.parkAndCheckInterrupt(AbstractQueuedSynchronizer.java:836)
        at java.util.concurrent.locks.AbstractQueuedSynchronizer.acquireQueued(AbstractQueuedSynchronizer.java:870)
        at java.util.concurrent.locks.AbstractQueuedSynchronizer.acquire(AbstractQueuedSynchronizer.java:1199)
        at java.util.concurrent.locks.ReentrantLock$NonfairSync.lock(ReentrantLock.java:209)
        at java.util.concurrent.locks.ReentrantLock.lock(ReentrantLock.java:285)
        at teaching.concurrency.deadlocks.DeadLockBasics.processThat(DeadLockBasics.java:54)
        at teaching.concurrency.deadlocks.DeadLockBasics$$Lambda$2/445051633.run(Unknown Source)
        at java.lang.Thread.run(Thread.java:748)
"Thread-0":
        at sun.misc.Unsafe.park(Native Method)
        - parking to wait for  <0x00000000d723cb58> (a java.util.concurrent.locks.ReentrantLock$NonfairSync)
        at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)
        at java.util.concurrent.locks.AbstractQueuedSynchronizer.parkAndCheckInterrupt(AbstractQueuedSynchronizer.java:836)
        at java.util.concurrent.locks.AbstractQueuedSynchronizer.acquireQueued(AbstractQueuedSynchronizer.java:870)
        at java.util.concurrent.locks.AbstractQueuedSynchronizer.acquire(AbstractQueuedSynchronizer.java:1199)
```

```
$ jstack 11475 > ./out.txt
```

```
$ kill -3 12704
```

Java MXBean

```
private static void detectDeadlock() {

  ThreadMXBean threadBean = ManagementFactory.getThreadMXBean();
  long[] threadIds = threadBean.findDeadlockedThreads();
  boolean deadLock = threadIds != null && threadIds.length > 0;
  System.out.println("Deadlocks found: " + deadLock);

}
```

HOw to resolve deadlock?

1. Add timeouts with each lock

Timeouts

```
private void execute() throws InterruptedException {

    Lock lock = new ReentrantLock();
    boolean acquired = lock.tryLock(2, TimeUnit.SECONDS);


    BlockingQueue queue = new ArrayBlockingQueue(16);
    queue.poll(2, TimeUnit.SECONDS);


    Semaphore sem = new Semaphore(1);
    sem.tryAcquire(2, TimeUnit.SECONDS);
}
```

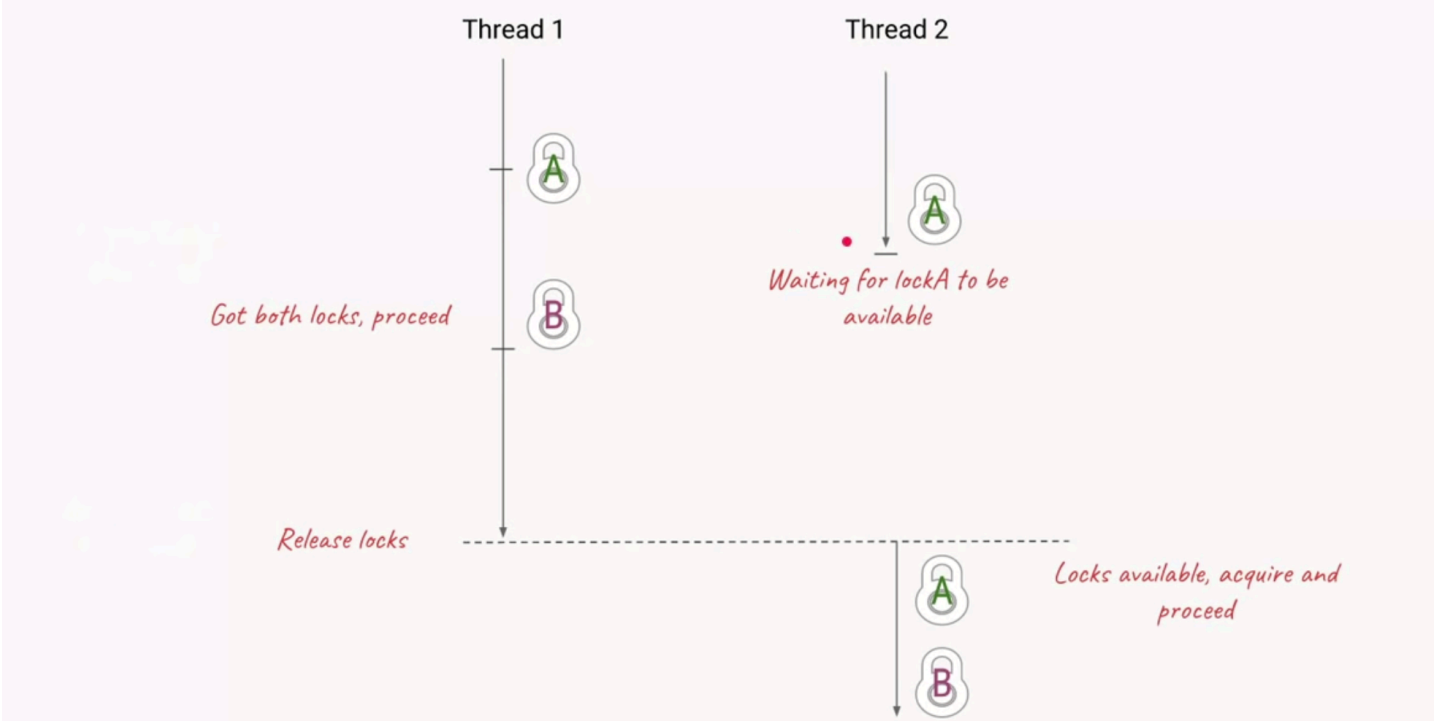
Global ordering

```
private void execute() {
    Thread t1 = new Thread(this::processThis);
    Thread t2 = new Thread(this::processThat);

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

public void processThis(){
    lockA.lock();
    // process resource A

    lockB.lock();
    // process resource A and B

    lockA.unlock();
    lockB.unlock();
}

public void processThat(){

    lockA.lock();
    lockB.lock();
    // process resource B
    // process resource A and B

    lockA.unlock();
    lockB.unlock();
}
```

Acquire LockA then LockB

Acquire LockA then LockB

Global ordering



Global ordering can be sometimes tricky

```
private void transfer(Account acc1, Account acc2, int amount) {

    synchronized (acc1) {
        synchronized (acc2) {
            acc1.deduct(amount);
            acc2.add(amount);
        }
    }
}

public void execute() {
    new Thread(this::transfer(john, marie, 100)).start();
    new Thread(this::transfer(marie, john, 300)).start();
}
```

acc1 | acc2

acc1 | acc2

Global ordering can be sometimes tricky

```
private void transfer(Account from, Account to, BigDecimal amount) {

    Account acc1 = getLarger(from, to);
    Account acc2 = getSmaller(from, to);

    synchronized (acc1) {
        synchronized (acc2) {
            from.deduct(amount);
            to.add(amount);
        }
    }
}
```

Summary

- DeadLocks occur when a thread is waiting for a lock held by other thread and vice versa
- Difficult to detect due to multiple lock types and thread sources
- Detect at runtime using thread dumps
- Consistent ordering of lock acquisition helps avoid deadlock
- Using timeouts for lock acquisition can also help