A tale of Multiple Threads

…AND TIMMY…

Timmy

Little ol' timmy, A developer was,
Just hired out of college three months back
He studiously learned all the frameworks indeed
Apache, glassfish, and Spring MVC

Then he started building all these wonderful apps
for a startup that really was about to start
with more than 3 queries to the DB at night
the site was complete, and this were allright

then the press release came, and the problems started
The site was live, but kept on crashing,
weird things call "deadlock" in the logs appeared
sun.misc.unsafe and more scary things.

Timmy

Oh timmy oh Timmy, how naive you have been

you knew threading is something you got a "C" in

....it doesn't go away ...if you don't think about it...

but today you will learn to be deadlock-free

Multithreading in Java

- Who am I?
  - Expedia Inc.
  - JUG Community Leader
  - Co-Author of Java 7 Recipes
  - Java Champion
  - Java Pub House / Java Off-Heap podcast
- Worked in Algo/High Frequency Trading

What's Timmy Learning Today?

- Why do I Need to Worry?
- What really happens behind the scenes
- Tools to Fix Multithreading issues
- Identifying Threading Issues

Why do I need to worry?

Why do I need to worry?

- Quirky Operational Failures

Why do I need t[illegible]orry?

- Quirky Operatio[illegible]

Why do I need to worry?

- Quirky Operational Failures

Why do I need to worry?

- Quirky Operational Failures

Why do I need to worry?

- Ok, where does it happen?

Why do I need to worry?

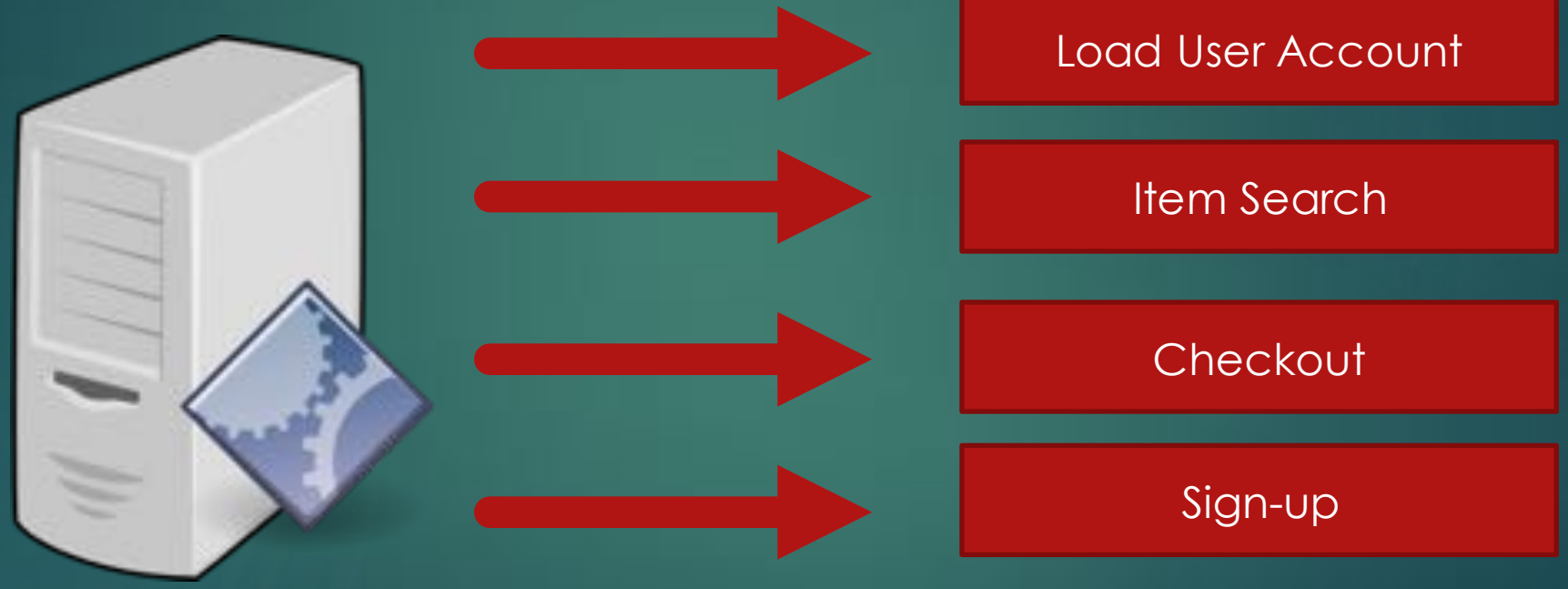
- Ok, where does it happen?

Why do I need to worry?

- Ok, where does it happen?

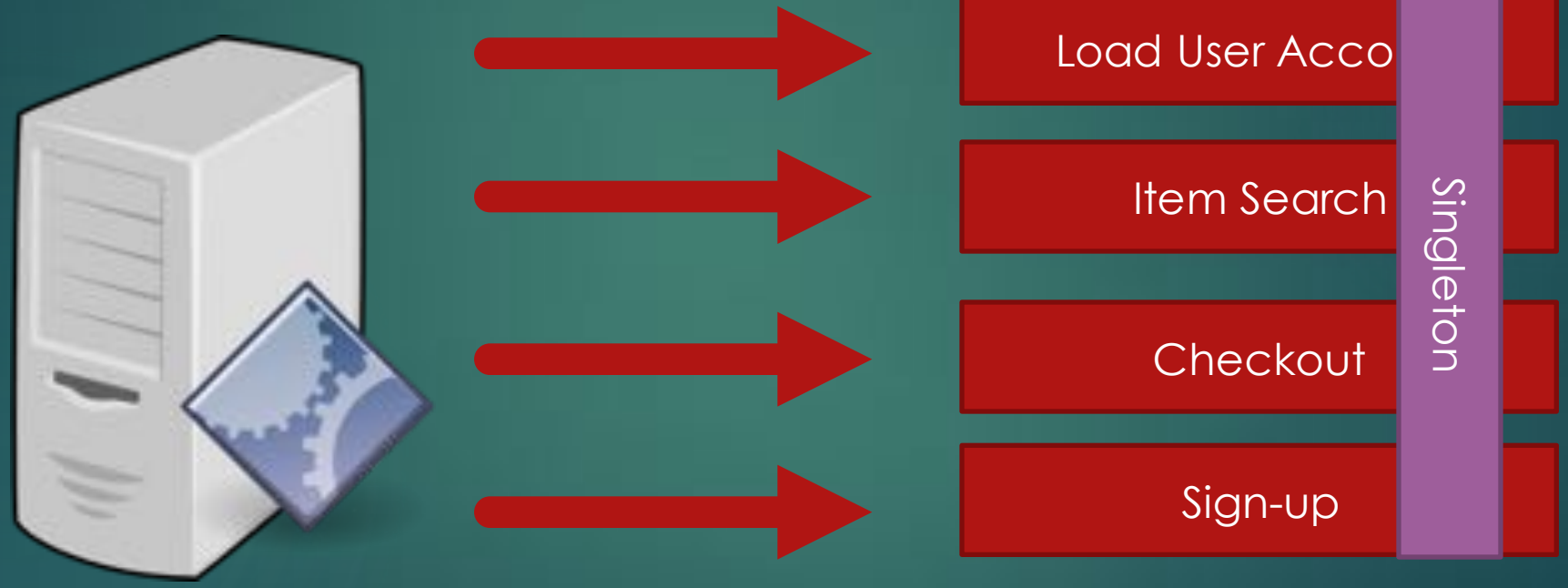
Why do I need to worry?

- Ok, where does it happen?



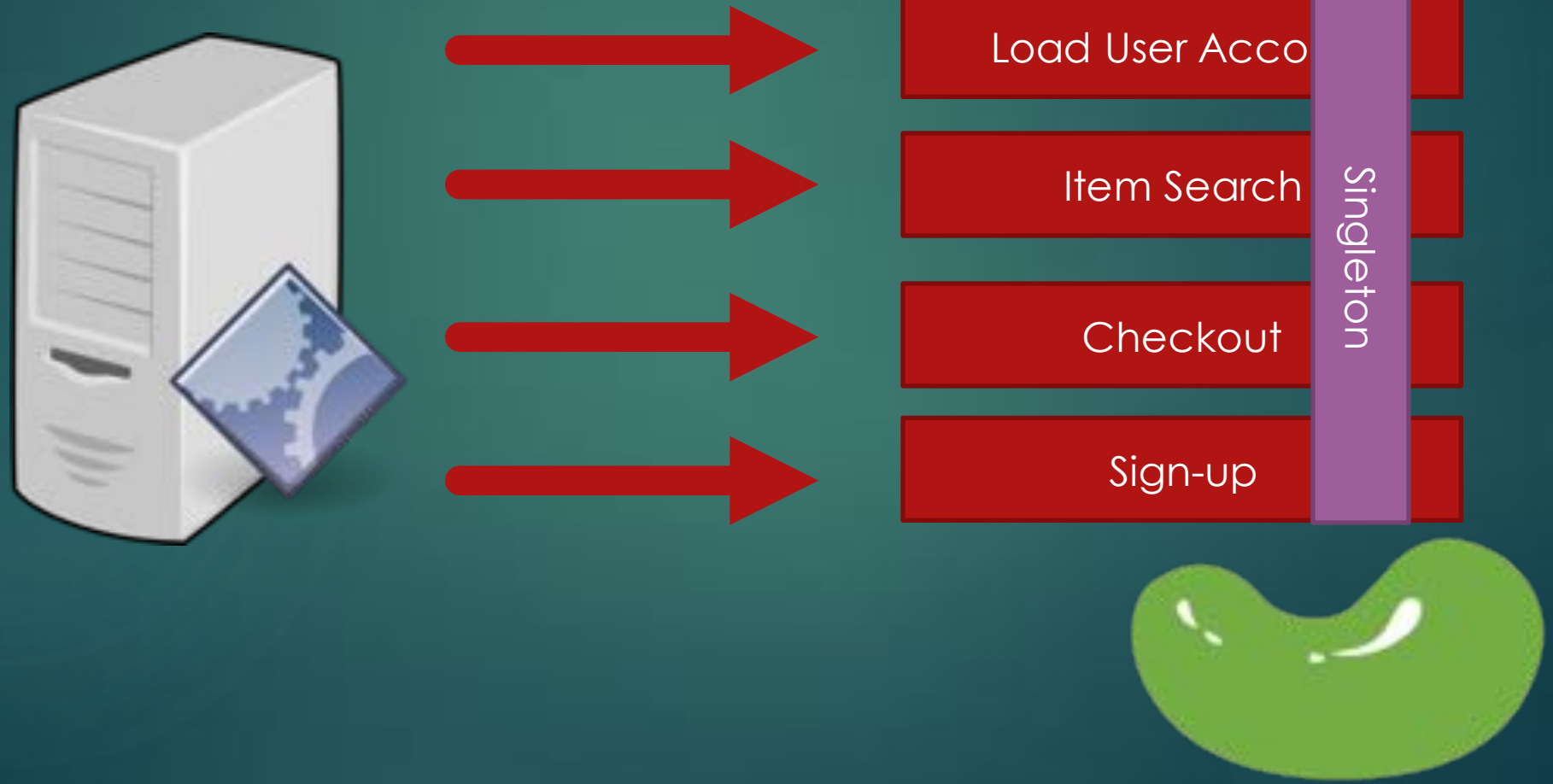
Why do I need to worry?

- Ok, where does it happen?



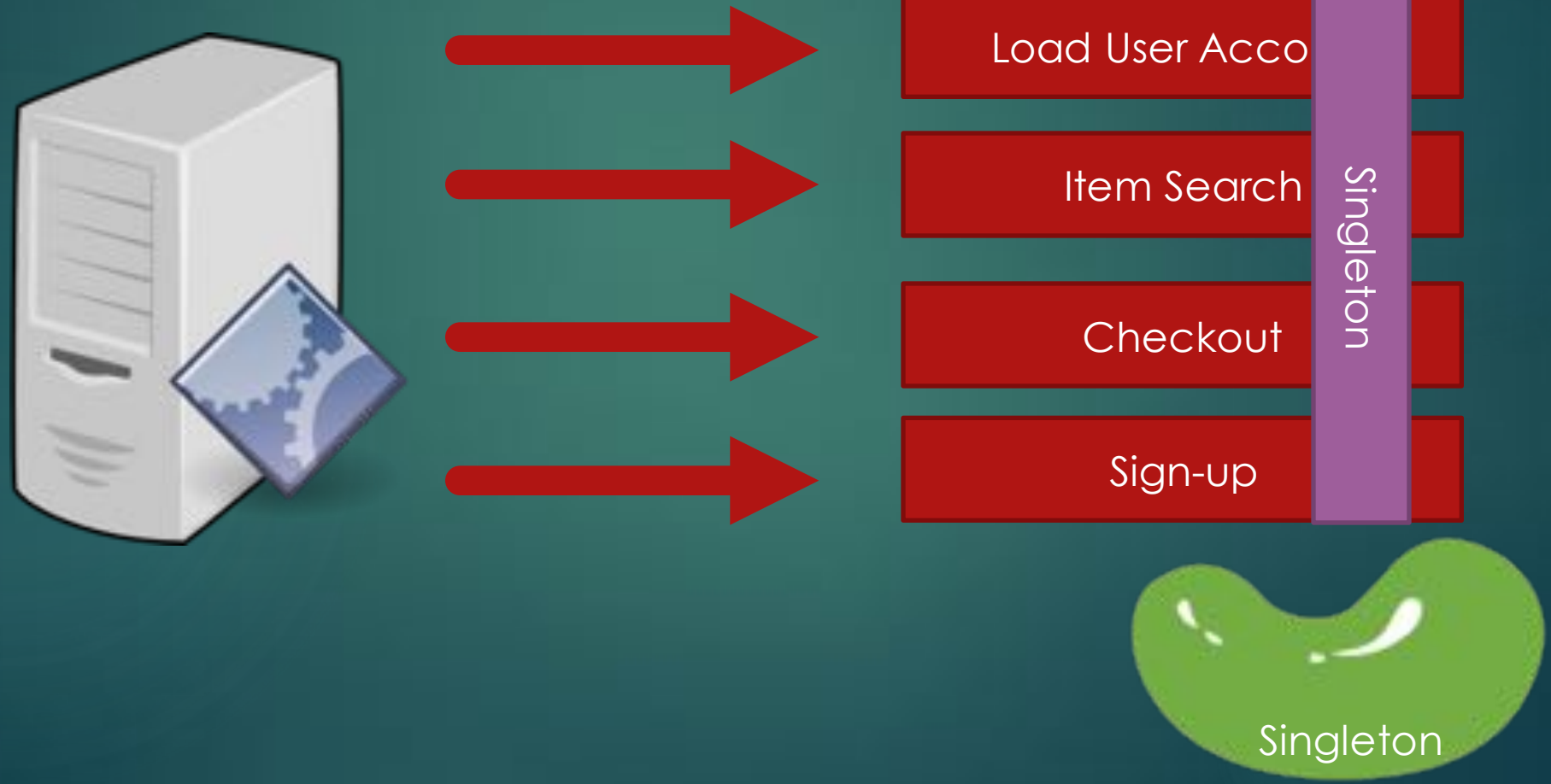
Why do I need to worry?

- Ok, where does it happen?



Why do I need to worry?

- Ok, where does it happen?

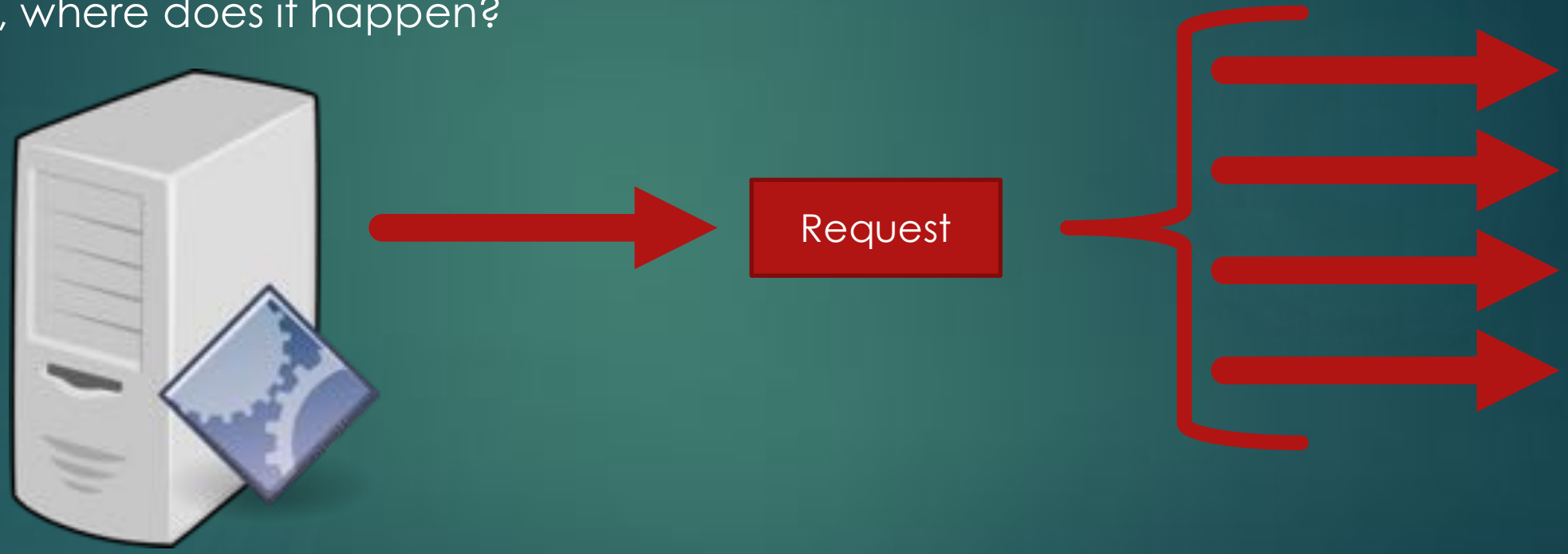


Why do I need to worry?

- Ok, where does it happen?

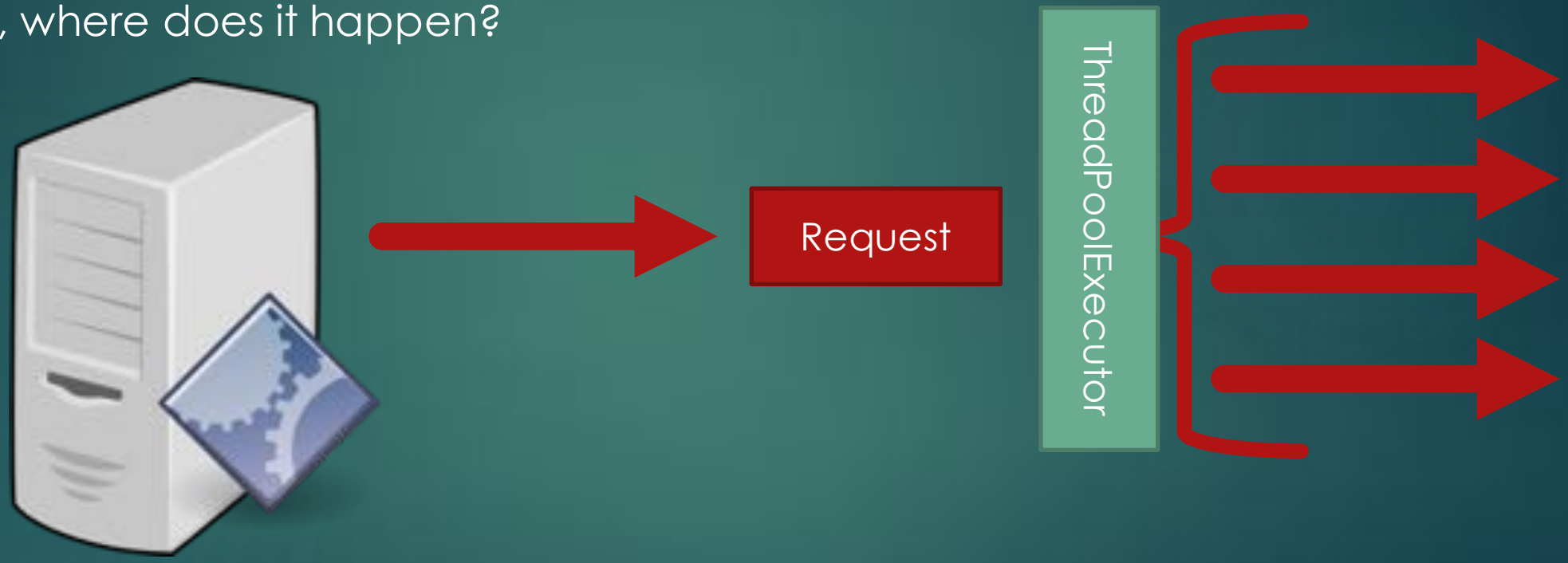
Why do I need to worry?

- Ok, where does it happen?



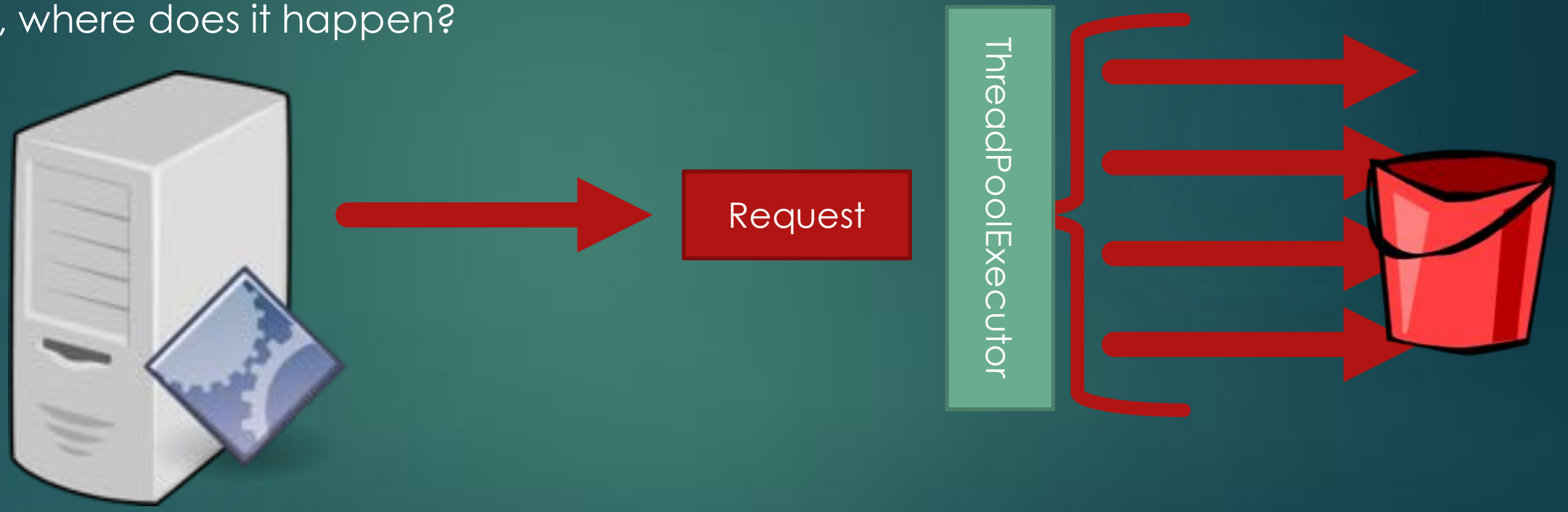
Why do I need to worry?

- Ok, where does it happen?



Why do I need to worry?

- Ok, where does it happen?



Why do I need to worry?

- Threading isn't hard

Why do I need to worry?

- Threading isn't hard
  - Once you get past the noise

Why do I need to worry?

- Threading isn't hard
  - Once you get past the noise
  - As long as you are not squeezing microseconds
    - Mechanical Sympathy
    - JVM Cache alignments, cache misses, cache locality, etc

What really happens behind the scenes

What really happens behind the scenes

What really happens behind the scenes

- Behaves as it looked.

What really happens behind the scenes

- Behaves as it looked.

```
long counter =0 ;
…
counter++;
```

What really happens behind the scenes

- Behaves as it looked.

```
long counter =0 ;
…
counter++;
```

What really happens behind the scenes

- Behaves as it looked.

```
long counter =0 ;
…
counter++;
```

- Moore's law broke down

What really happens behind the scenes

- Behaves as it looked.

```
long counter =0 ;
…
counter++;
```

- Moore's law broke down
- JSR-133: Java Memory Model

What really happens behind the scenes

- Behaves as it looked.

```
long counter =0 ;
…
counter++;
```

- Moore's law broke down
- JSR-133: Java Memory Model
  - Certain Guarantees

What really happens behind the scenes

- Behaves as it looked.

```
long counter =0 ;
…
counter++;
```

- Moore's law broke down
- JSR-133: Java Memory Model
  - Certain Guarantees
  - If not specified, left for the OS to decide

What really happens behind the scenes

- Instruction re-ordering

What really happens behind the scenes

- Instruction re-ordering
- Cache Locality

What really happens behind the scenes

- Instruction re-ordering
- Cache Locality
- Non-Atomic Memory Operations

What really happens behind the scenes

- It's all about "Safe Publication"

What really happens behind the scenes

- It's all about "Safe Publication"
- When a Thread writes

What really happens behind the scenes

- It's all about "Safe Publication"
- When a Thread writes
- Another Thread can read

What really happens behind the scenes

```
long counter =0 ;
…
counter++;
```

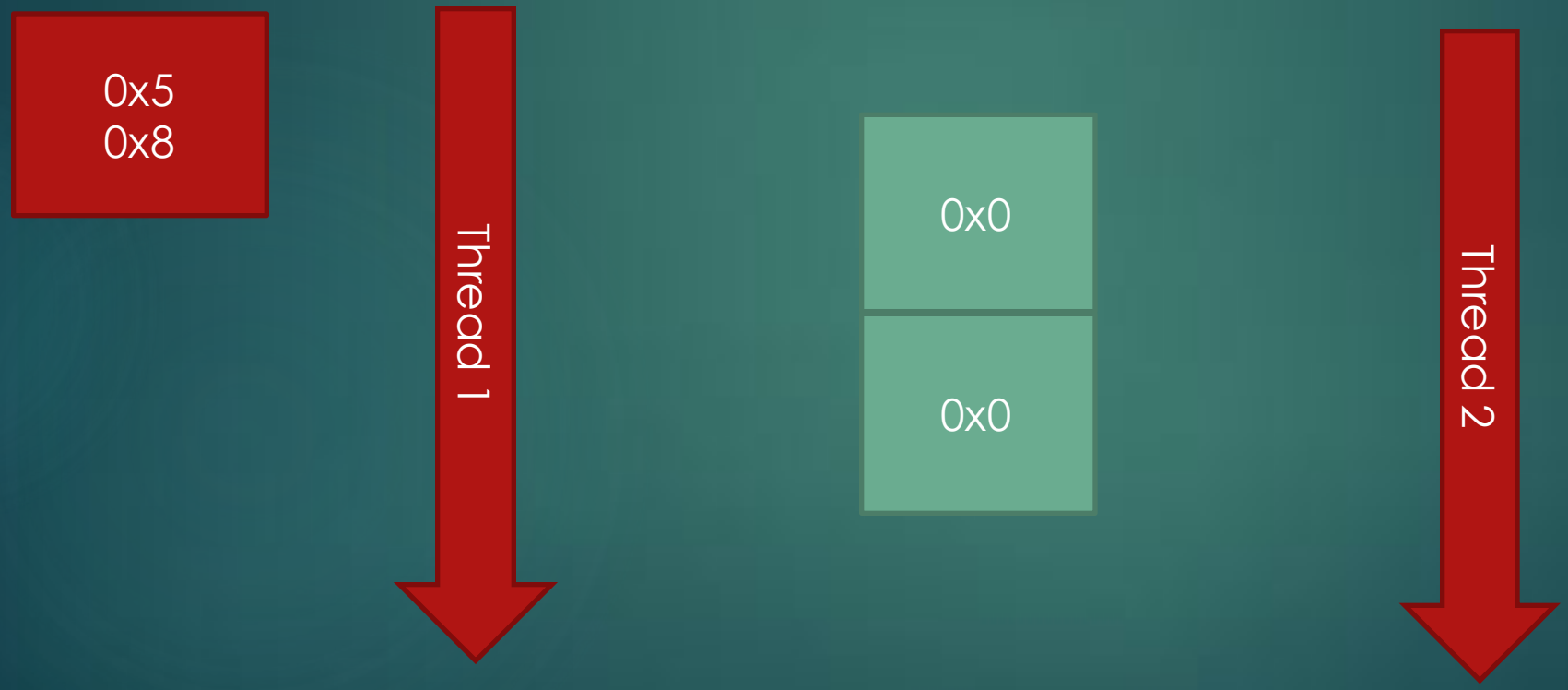
What really happens behind the scenes

```
long counter =0 ;
…
counter++;
```

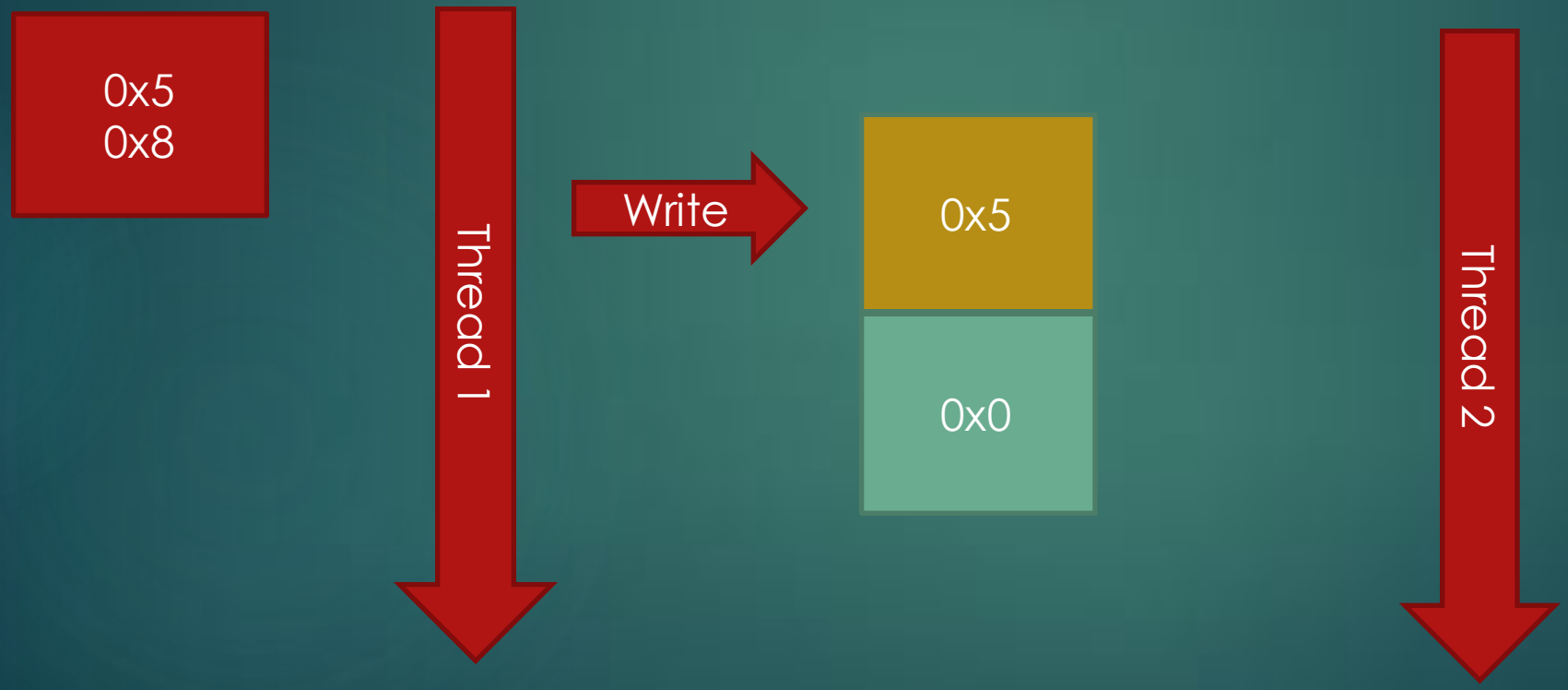
What really happens behind the scenes

```
long counter =0 ;
…
count
```

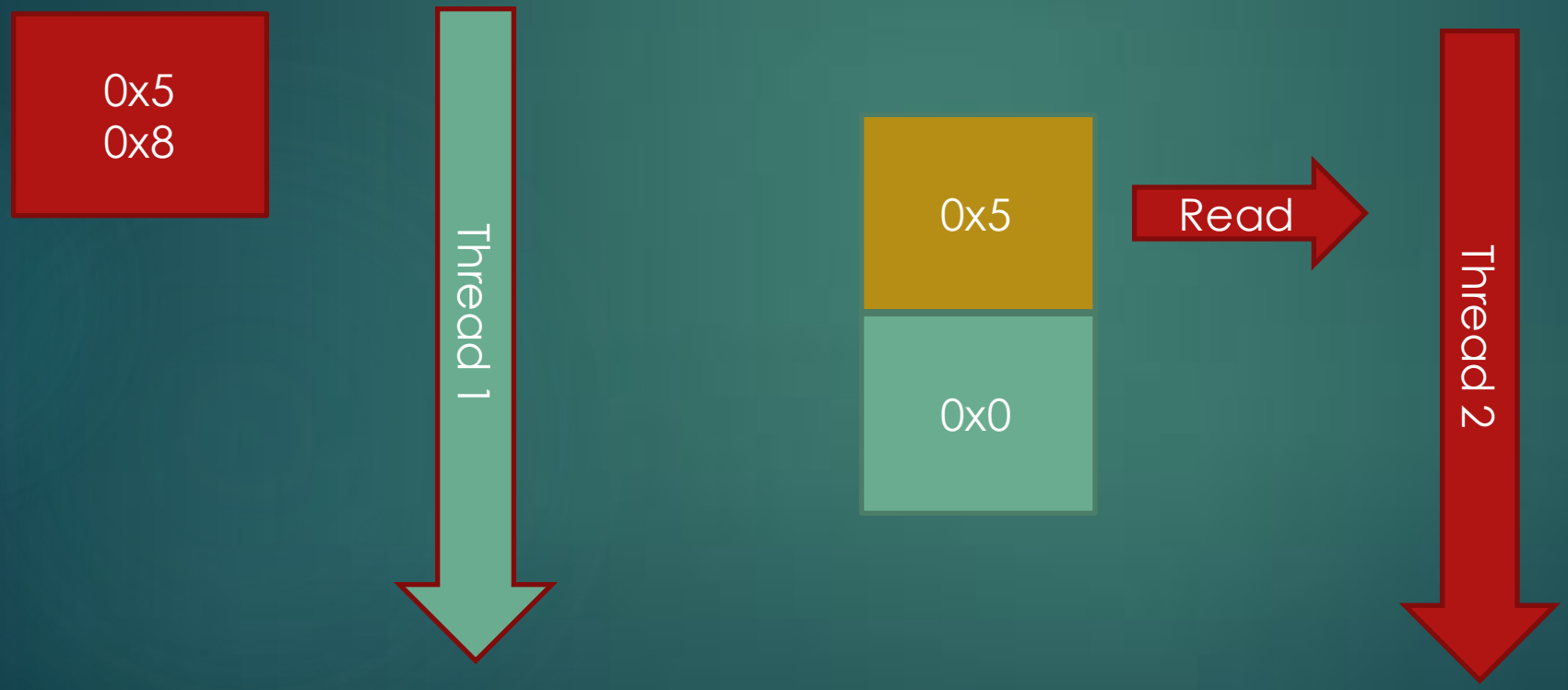
What really happens behind the scenes



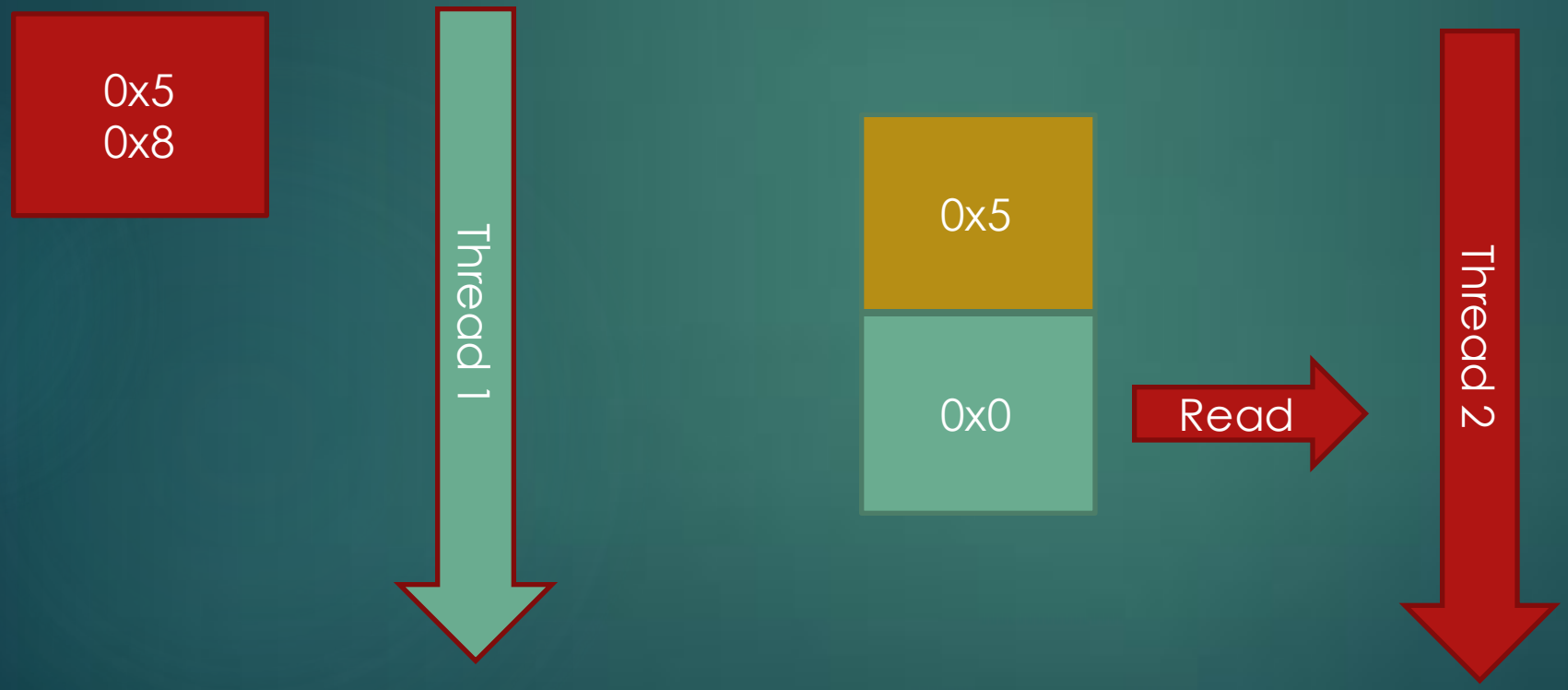
What really happens behind the scenes



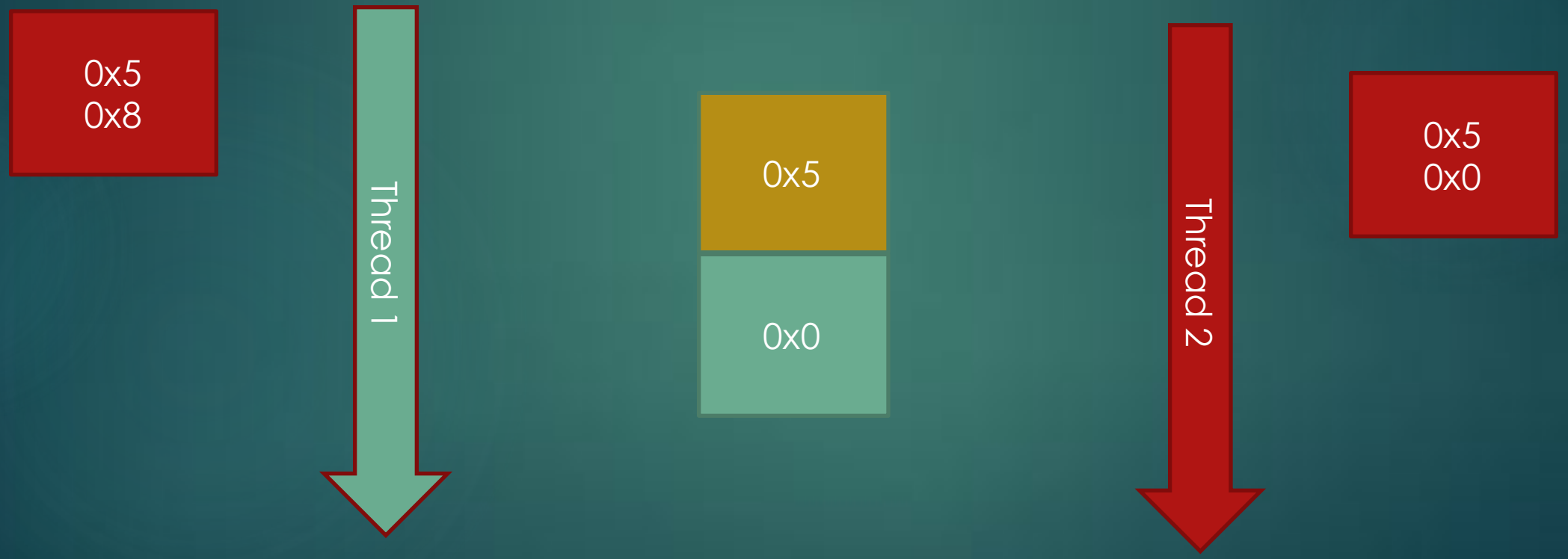
What really happens behind the scenes



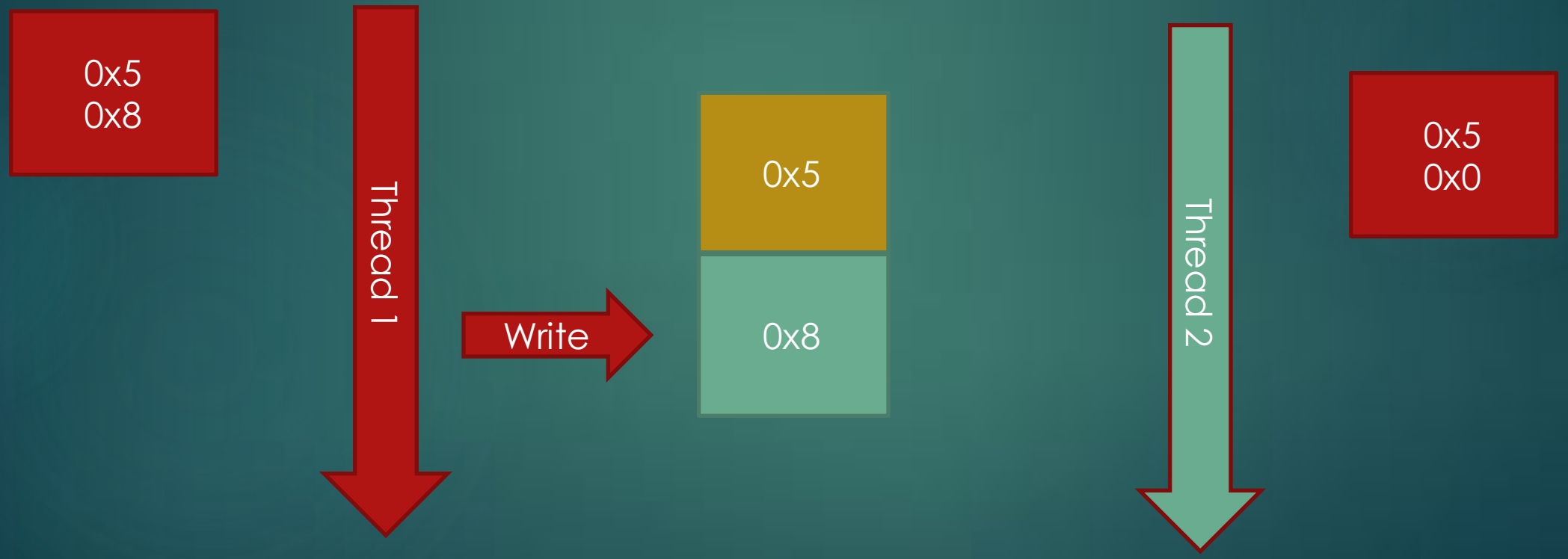
What really happens behind the scenes



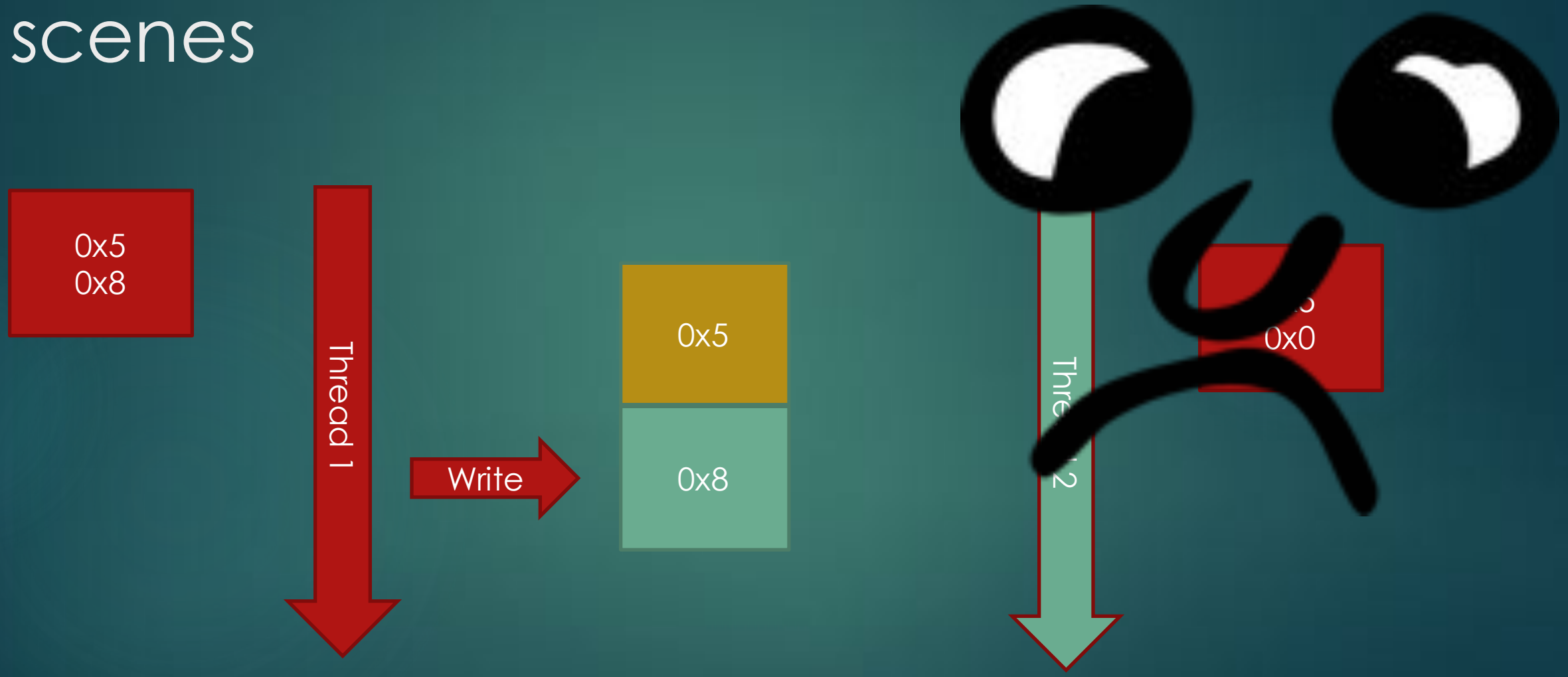
What really happens behind the scenes



What really happens behind the scenes



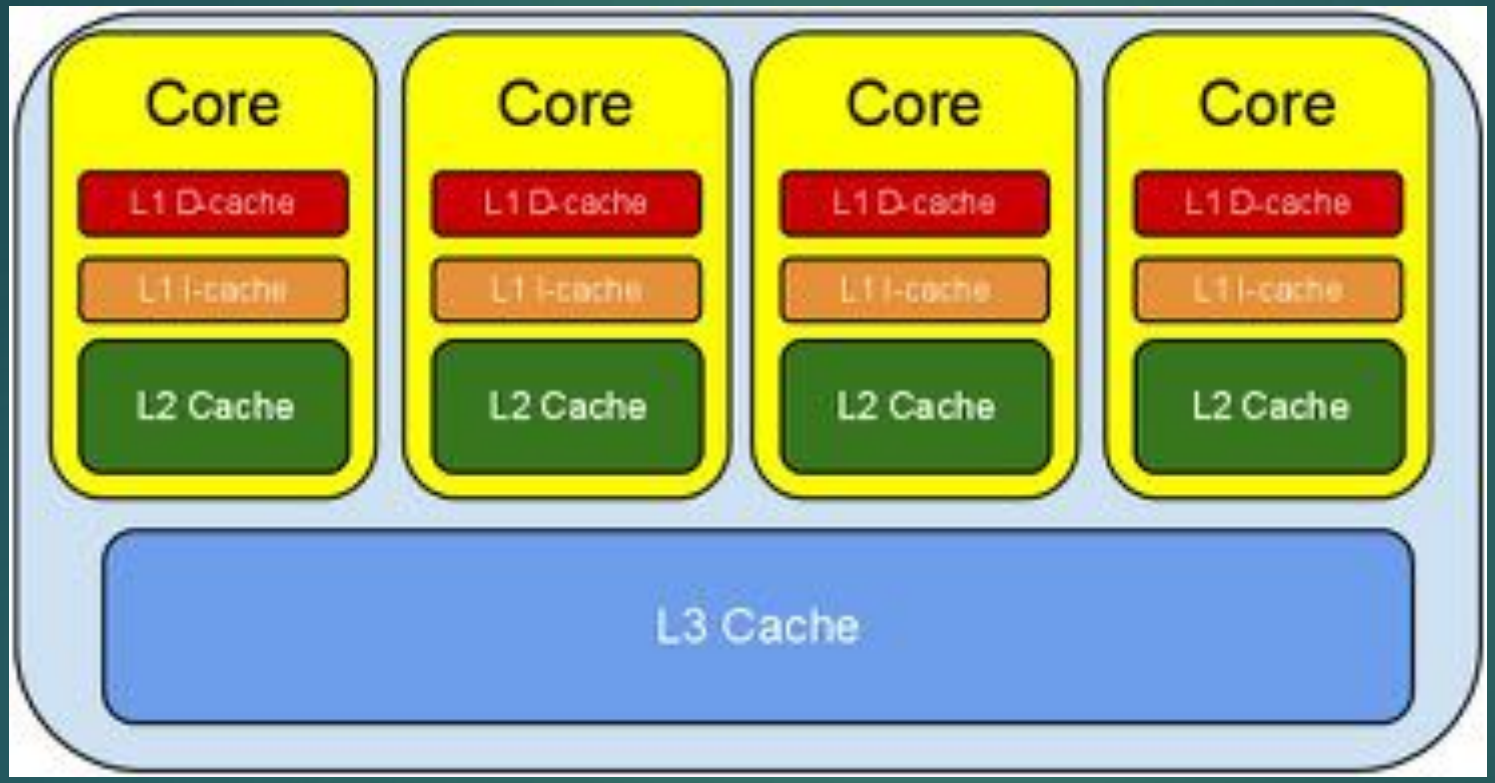
What really happens behind the scenes

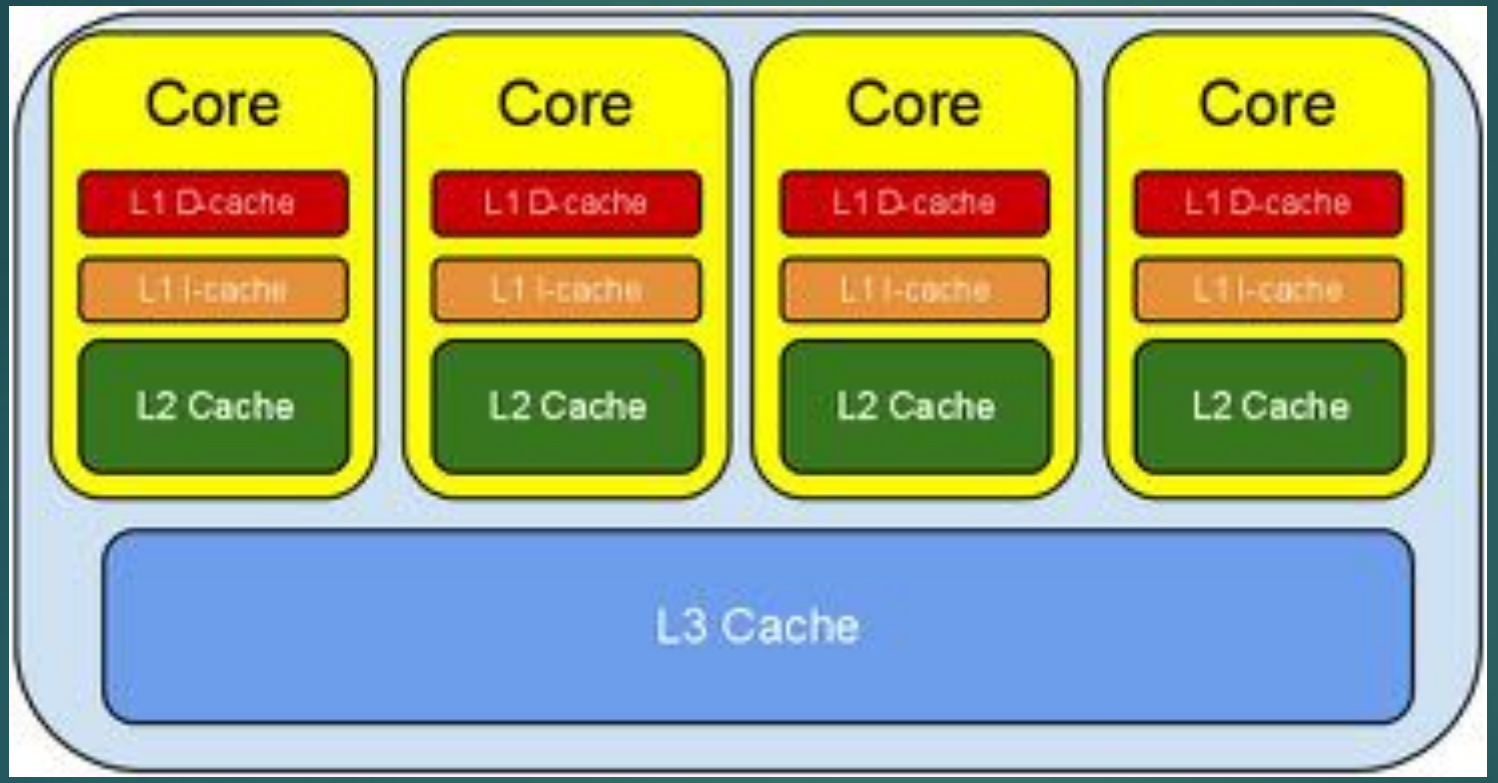


```
long counter =0 ;
…
counter++;
```

```
long counter =0 ;
…
counter++;
```

```
long counter =0 ;
…
counter++;
```


Core
L1 D-cache
L1 I-cache
L2 Cache
Core
L1 D-cache
L1 I-cache
L2 Cache
Core
L1 D-cache
L1 I-cache
L2 Cache
Core
L1 D-cache
L1 I-cache
L2 Cache
L3 Cache




```
L1 cache reference          0.5 ns
L2 cache reference            7   ns                  14x L1 cache
Main memory reference       100   ns                  20x L2 cache, 200x L1 cache
```

Thread 1

counter = 10

Thread 2

counter = 10

Thread 1

counter = 10

++

Thread 2

counter = 10

++

Thread 1

counter = 10

++

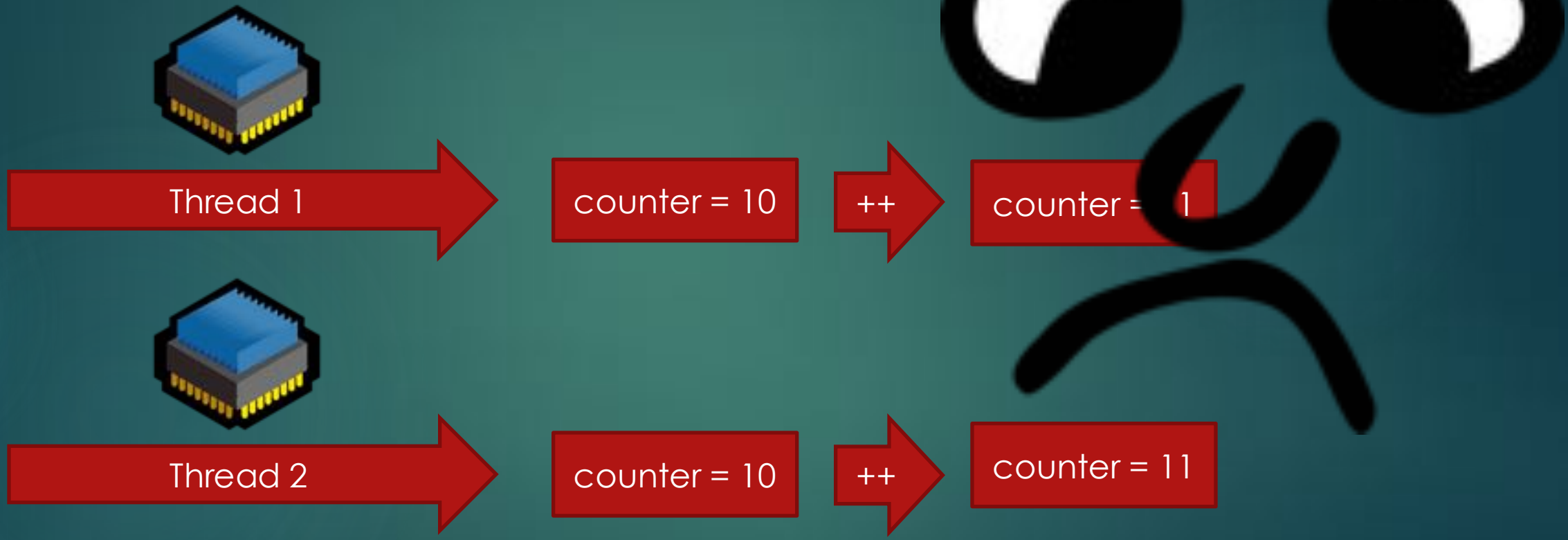
counter = 11

Thread 2

counter = 10

++

counter = 11


Thread 1
counter = 10
++
counter = 1
Thread 2
counter = 10
++
counter = 11

```
long counter =0 ;
…
counter++;
```

```
long counter =0 ;
…
counter++;
```

```
long counter =0 ;
…
counter++;
```

Thread 1

Thread 2

Thread 1

Thread 2

READ

count=10

count=10

Thread 1

Thread 2

READ

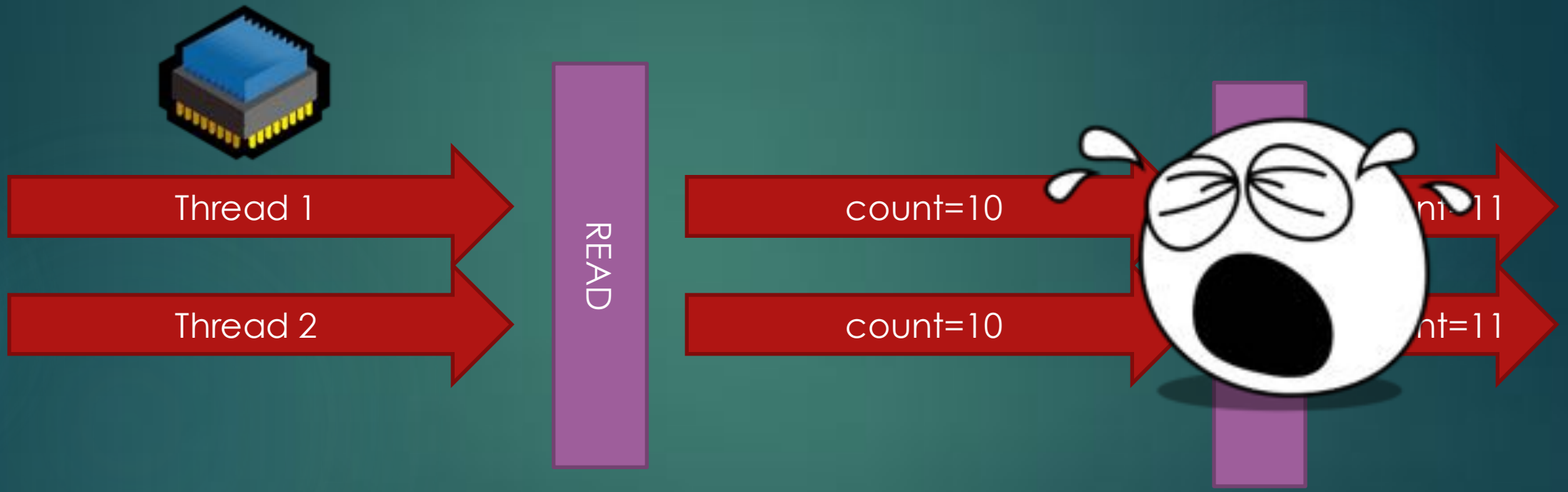
count=10

count=10

WRITE

count=11

count=11


Thread 1
Thread 2
READ
count=10
count=10

All is not lost!

- Didn't program multi-threaded way

```
long counter =0 ;
…
counter++;
```

All is not lost!

- Didn't program multi-threaded way
- If single-threaded, above doesn't matter

```
long counter =0 ;
…
counter++;
```

All is not lost!

- Didn't program multi-threaded way
- If single-threaded, above doesn't matter
- Tell the JVM

```
long counter =0 ;
…
counter++;
```

All is not lost!

- Didn't program multi-threaded way
- If single-threaded, above doesn't matter
- Tell the JVM
  - Safe Publication!

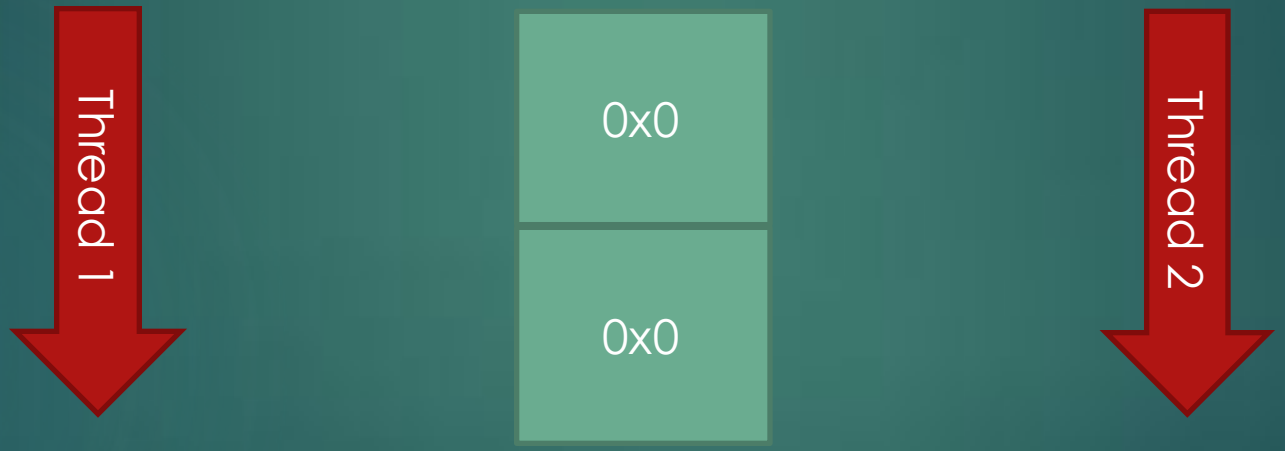
```
long counter =0 ;
…
counter++;
```

Safe Publication

- **Safe publication makes all the values written before the publication visible to all readers that observed the published object**

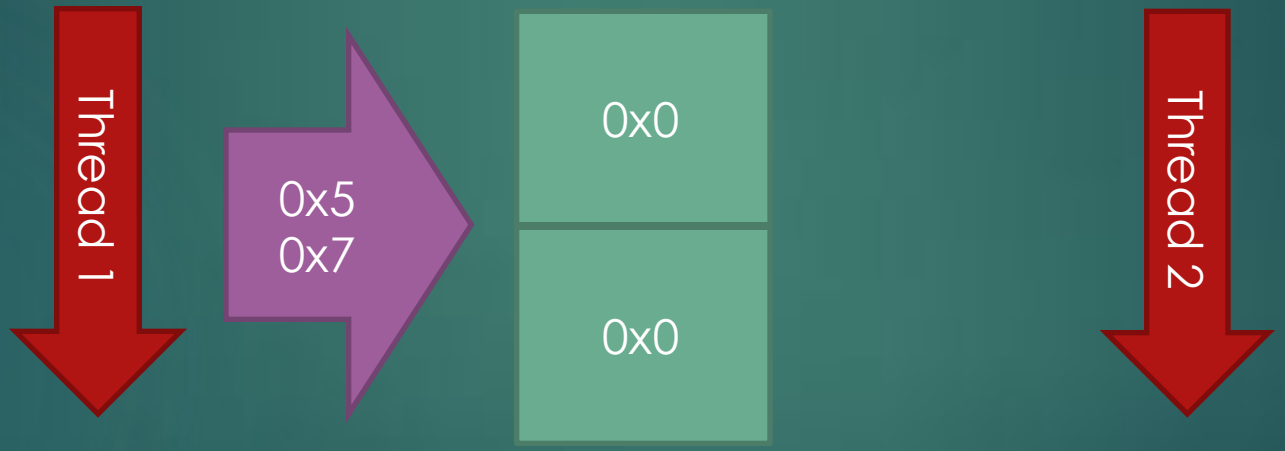
Safe Publication

- **Safe publication makes all the values written before the publication visible to all readers that observed the published object**



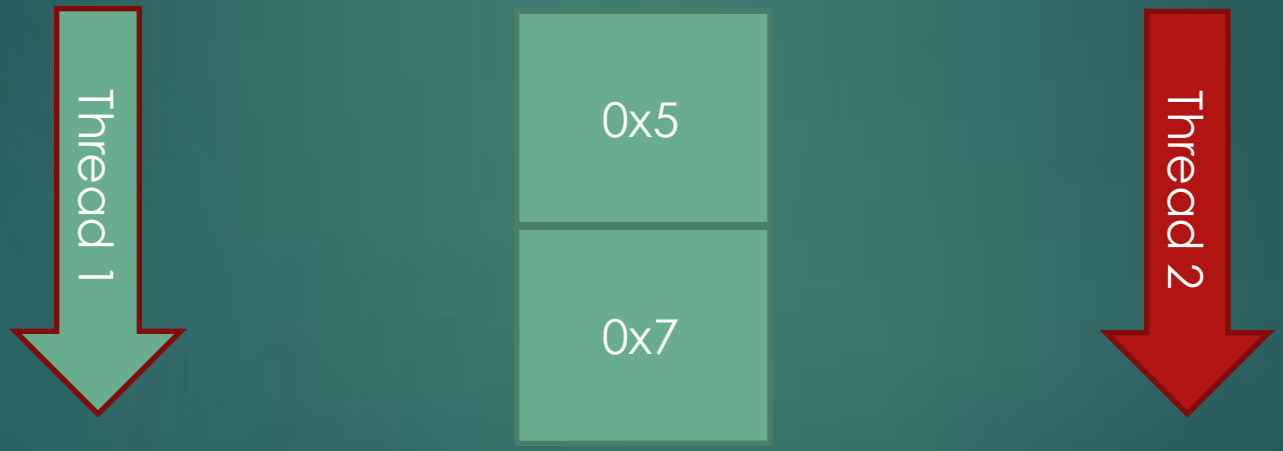
Safe Publication

- **Safe publication makes all the values written before the publication visible to all readers that observed the published object**



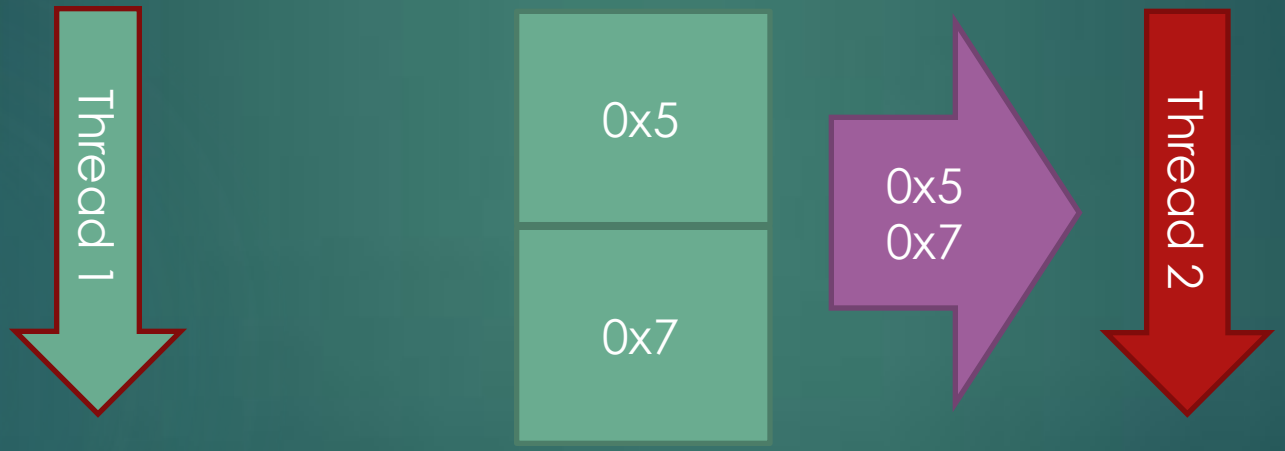
Safe Publication

- **Safe publication makes all the values written before the publication visible to all readers that observed the published object**



Safe Publication

- **Safe publication makes all the values written before the publication visible to all readers that observed the published object**



Tools to Fix Multithreading issues

Tools to Fix Multithreading issues

- Immutability

Tools to Fix Multithreading issues

- Immutability

Tools to Fix Multithreading issues

- Immutability
  - FINAL Fields

Tools to Fix Multithreading issues

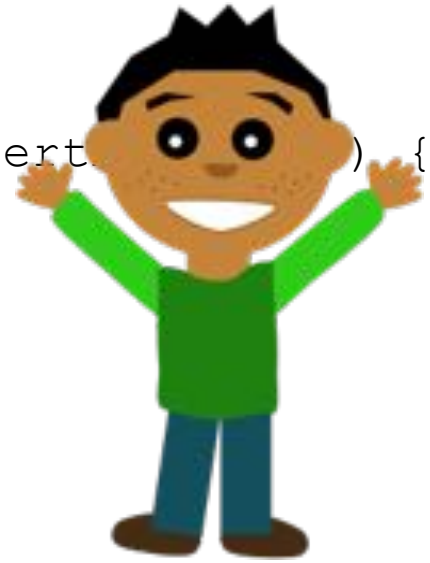
- Immutability
  - FINAL Fields
  - Initialized at Constructor

Tools to Fix Multithreading issues

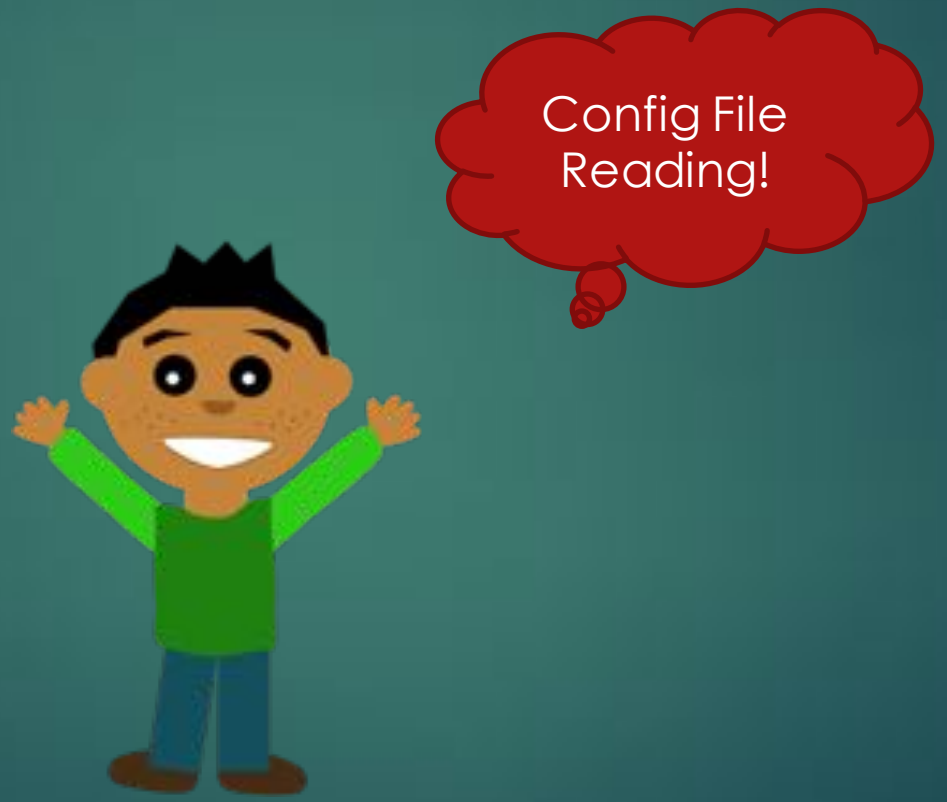
```
public class User {
    final String firstName;
    final String lastName;
    final Properties props;

    public User(String firstName, String lastName, Properties props) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.props = props;
    }
}
```

Tools to Fix Multithreading issues

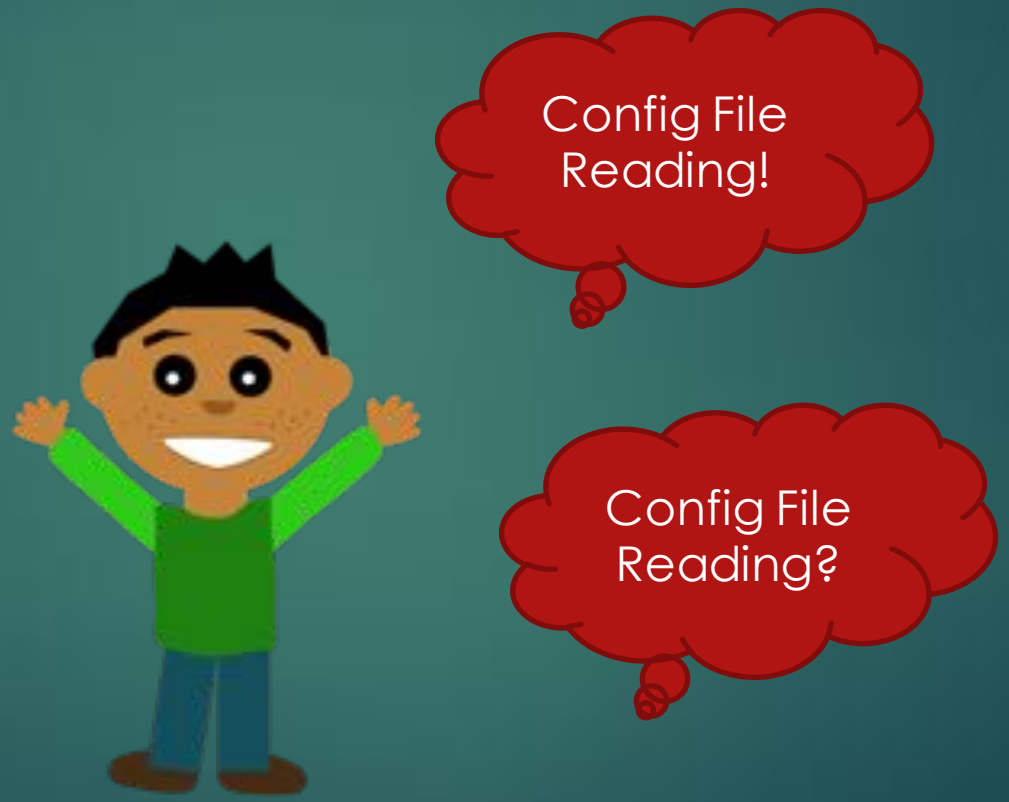
```
public class User {
    final String firstName;
    final String lastName;
    final Properties props;

    public User(String firstName, String lastName, Propert          ) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.props = props;
    }
}
```



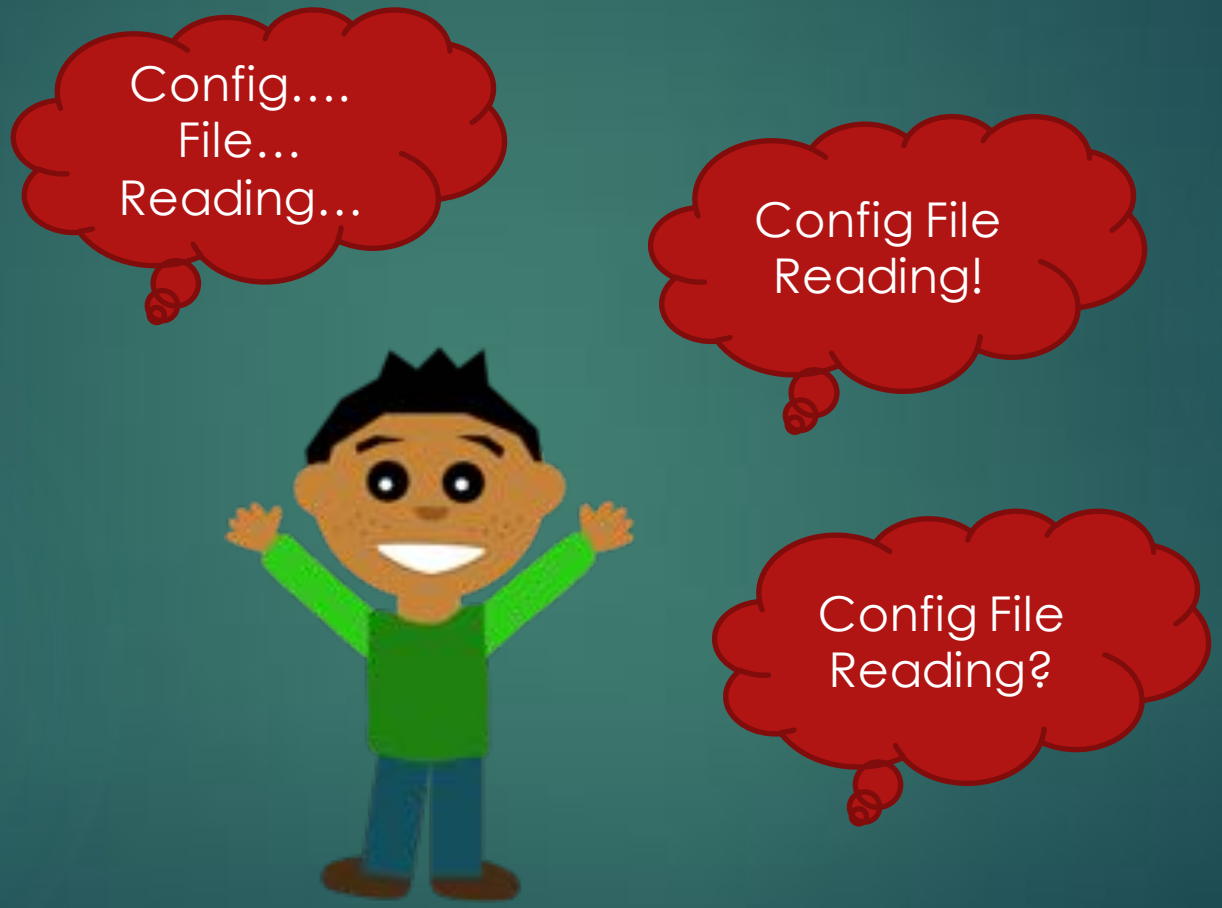
Tools to Fix Multithreading issues



Tools to Fix Multithreading issues



Tools to Fix Multithreading issues



Tools to Fix Multithreading issues

Tools to Fix Multithreading issues

```
public class User {
    final String firstName;
    final String lastName;
    final Properties props;

    public User(String firstName, String lastName, Properties props) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.props = props;
    }

    public static final class Builder {
        String firstName;
        String lastName;
        Properties props;

        public void setFirstName(String firstName) {
            this.firstName = firstName;
        }

        public void setLastName(String lastName) {
            this.lastName = lastName;
        }

        public void setProps(Properties props) {
            this.props = props;
        }

        public User build() {
           return new User(firstName, lastName, props);
        }
    }
}
```

Tools to Fix Multithreading issues

```
public class User {
    final String firstName;
    final String lastName;
    final Properties props;

    public User(String firstName, String lastName, Properties props) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.props = props;
    }

    public static final class Builder {

        public void setProps(Properties props) {
            this.props = props;
        }

        public User build() {
            return new User(firstName, lastName, props);
        }
    }
}
```

BUILDER PATTERN

Tools to Fix Multithreading issues

- Immutability
  - FINAL Fields
  - Initialized at Constructor
  - “Longer Road” to Program

Tools to Fix Multithreading issues

- Immutability
  - FINAL Fields
  - Initialized at Constructor
  - “Longer Road” to Program
  - As Thread Safe as it can get

Tools to Fix Multithreading issues

- Language tools

Tools to Fix Multithreading issues

- Language tools
  - synchronized keyword

Tools to Fix Multithreading issues

- Language tools
  - synchronized keyword
    - Easiest to understand

Tools to Fix Multithreading issues

User

```
public void synchronized setPassword (String password) {

    ...some good magic here

}
```

Tools to Fix Multithreading issues

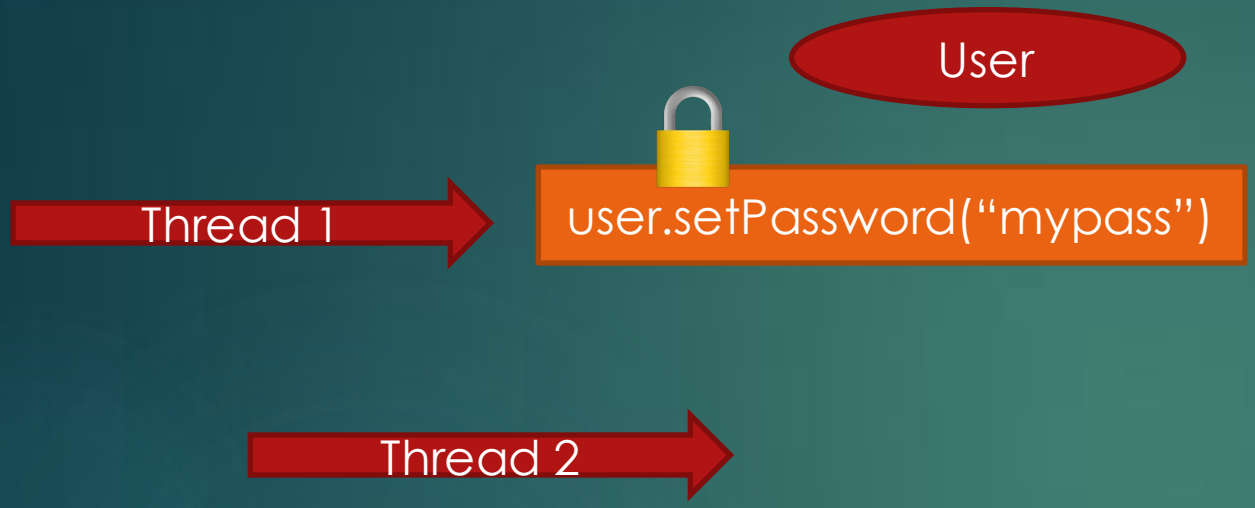
Tools to Fix Multithreading issues

Thread 1

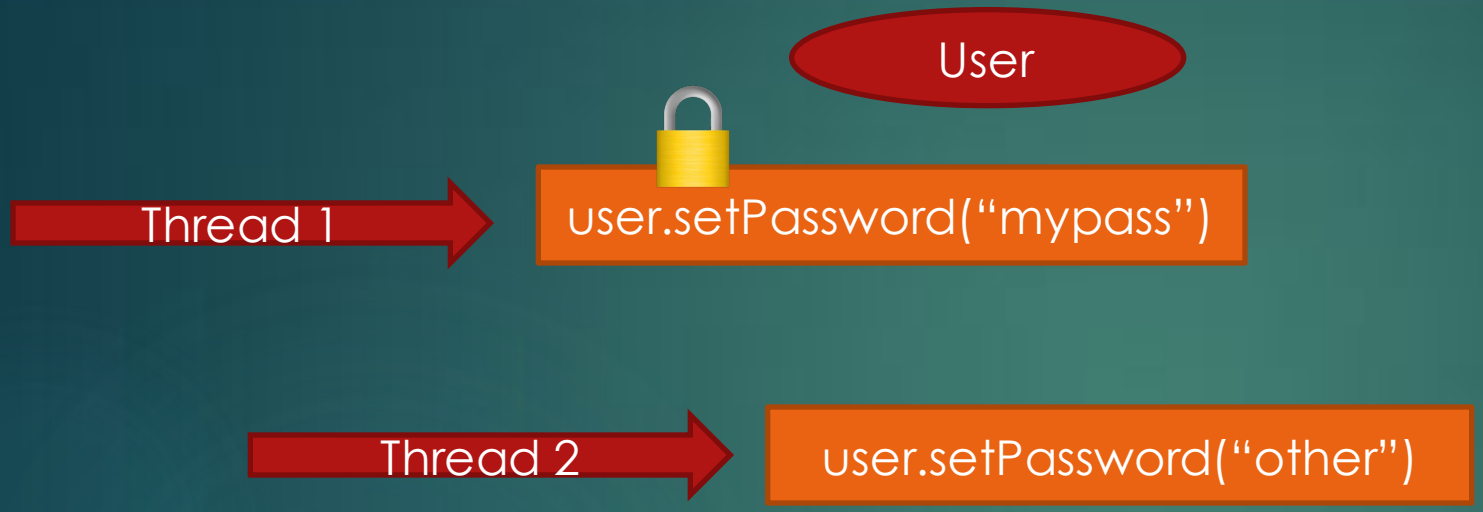
```
user.setPassword("mypass")
```

Tools to Fix Multithreading issues

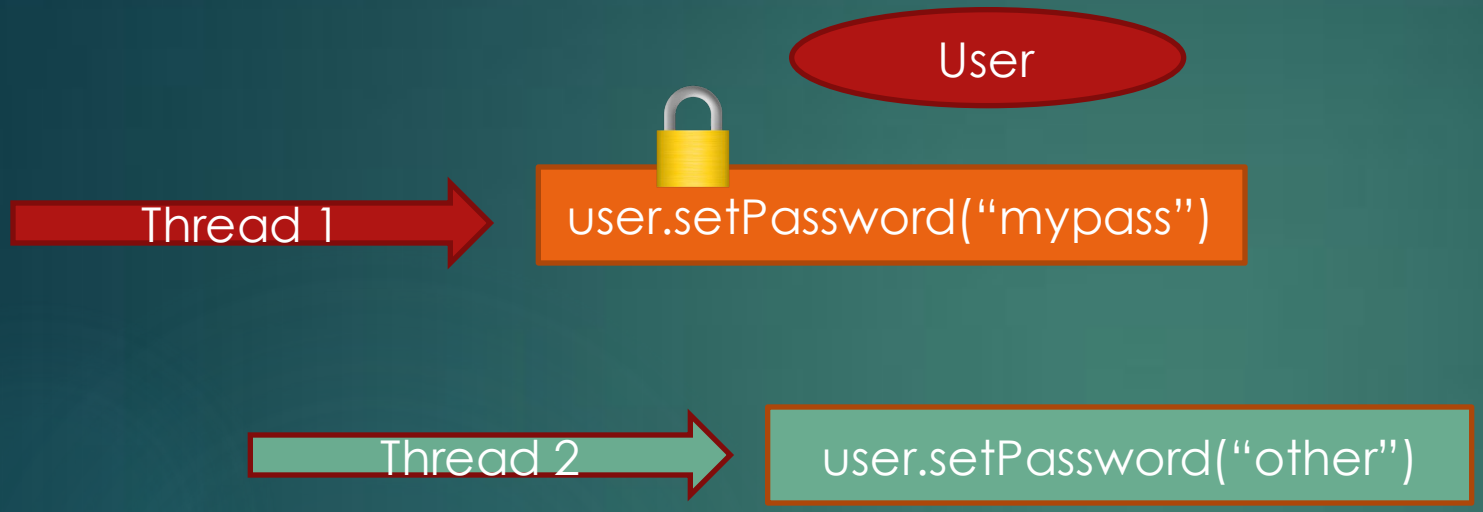
Tools to Fix Multithreading issues



Tools to Fix Multithreading issues



Tools to Fix Multithreading issues



Tools to Fix Multithreading issues

User

Thread 1

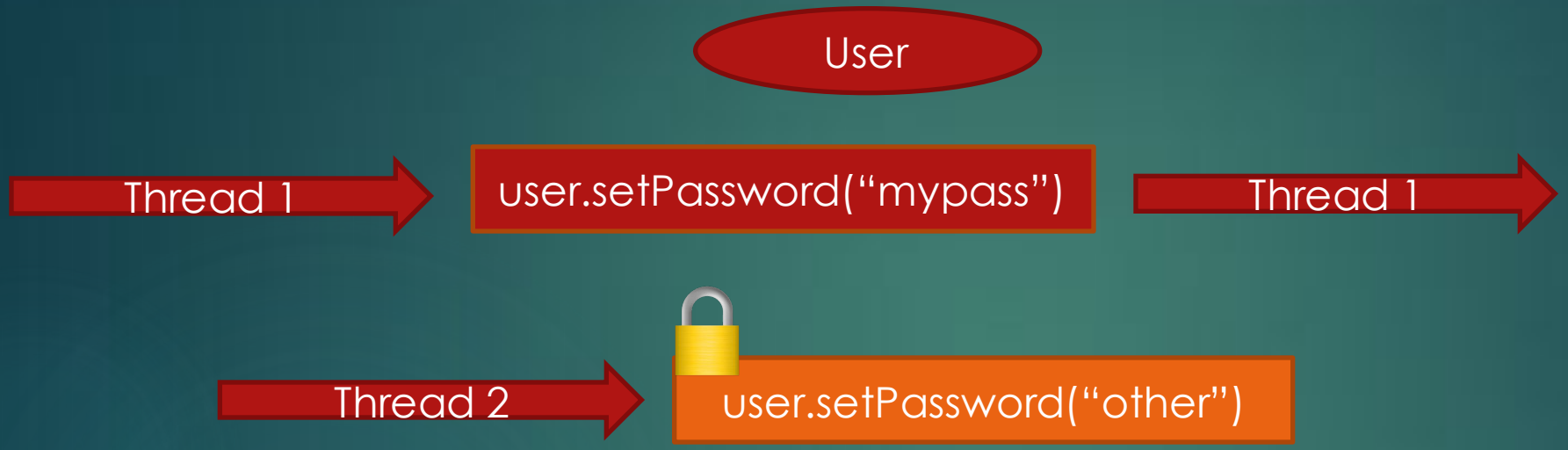
user.setPassword("mypass")

Thread 1

Thread 2

user.setPassword("other")

Tools to Fix Multithreading issues



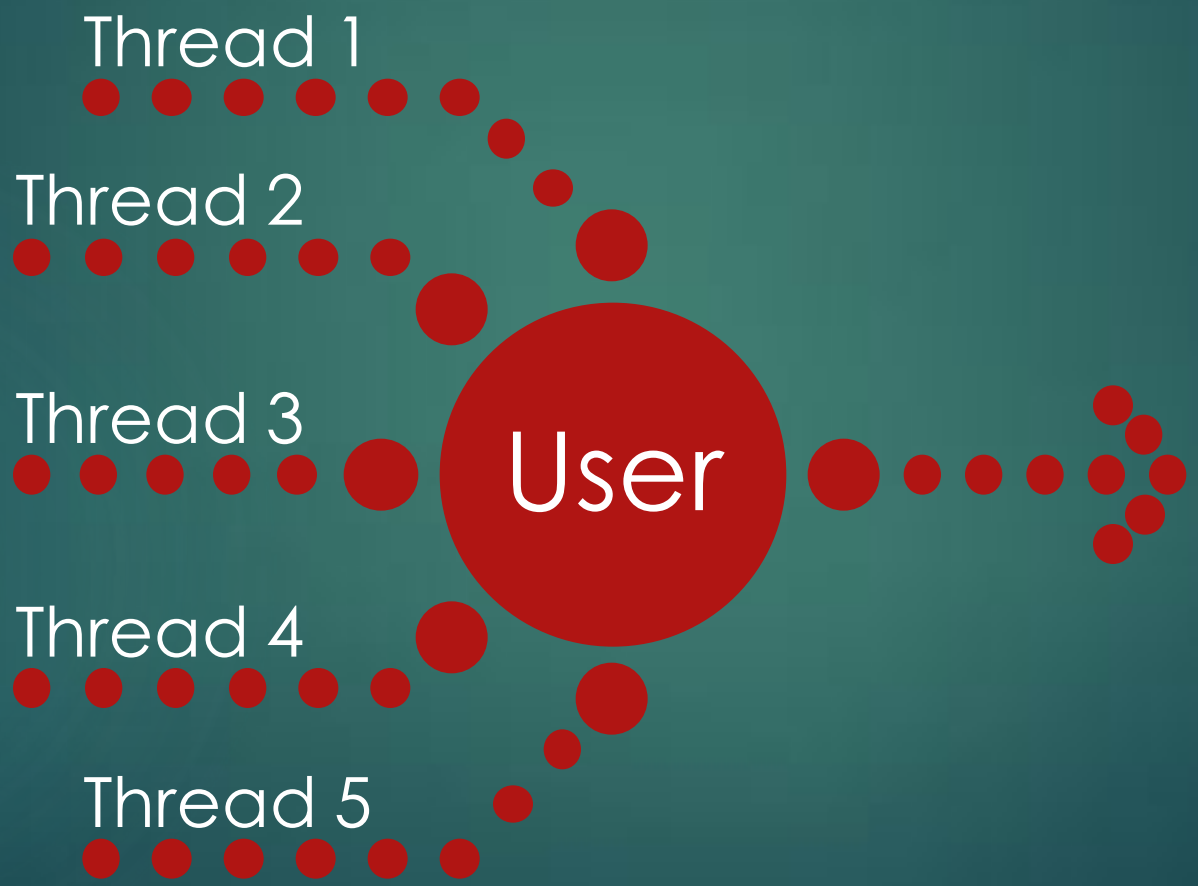
Tools to Fix Multithreading issues

Tools to Fix Multithreading issues

Thread 1 → user.setPassword("mypass")

Thread 2 → user.setAccessCnt(10)

Tools to Fix Multithreading issues



Tools to Fix Multithreading issues

- Language tools
  - synchronized keyword
    - Easiest to understand
    - Any changes made within synchronized block
      - Are safely published

Tools to Fix Multithreading issues

- Language tools
  - synchronized keyword
    - Easiest to understand
    - Any changes made within synchronized block
      - Are safely published
    - BUT, if not understood properly
      - Create a bottleneck

Tools to Fix Multithreading issues

- Language tools
  - synchronized keyword
    - Easiest to understand
    - Any changes made within synchronized block
      - Are safely published
    - BUT, if not understood properly
      - Create a bottleneck

Tools to Fix Multithreading issues

- Language tools
  - volatile keyword

Tools to Fix Multithreading issues

- Language tools
  - volatile keyword
  - makes it “uncacheable” by L1/L2 CPU

Tools to Fix Multithreading issues

- Language tools
  - volatile keyword
  - makes it “uncacheable” by L1/L2 CPU
  - It is not a lock! (counter++ still two actions)

Tools to Fix Multithreading issues

- Language tools
  - volatile keyword
  - makes it “uncacheable” by L1/L2 CPU
  - It is not a lock! (counter++ still two actions)
  - Has performance impact!

Tools to Fix Multithreading issues

- Language tools
  - volatile keyword
  - makes it “uncacheable” by L1/L2 CPU
  - It is not a lock! (counter++ still two actions)
  - Has performance impact!
    - Albeit less than other tools

Tools to Fix Multithreading issues

- Library Tools
  - Java.util.concurrent

Tools to Fix Multithreading issues

- Library Tools
  - Java.util.concurrent

Tools to Fix Multithreading issues

- Library Tools
  - Java.util.concurrent

CountdownLatch

Tools to Fix Multithreading issues

- Library Tools
  - Java.util.concurrent

CountdownLatch

BlockingQueue

Tools to Fix Multithreading issues

- Library Tools
  - Java.util.concurrent

ConcurrenHashMap

CountdownLatch

BlockingQueue

Tools to Fix Multithreading issues

- Library Tools
  - Java.util.concurrent

ConcurrenHashMap

CountdownLatch

AtomicLong

BlockingQueue

Tools to Fix Multithreading issues

- Library Tools
  - Java.util.concurrent
    - Specific Purpose

Tools to Fix Multithreading issues

- Library Tools
  - Java.util.concurrent
    - Specific Purpose
    - 99.999% use cases

Tools to Fix Multithreading issues

- Library Tools
  - Java.util.concurrent
    - Specific Purpose
    - 99.999% use cases
    - Documentation is GOLDEN

Tools to Fix Multithreading issues

```
/**
 * A synchronization aid that allows one or more threads to wait until
 * a set of operations being performed in other threads completes.
 *
 * <p>A {@code CountDownLatch} is initialized with a given <em>count</em>.
 * The {@link #await await} methods block until the current count reaches
 * zero due to invocations of the {@link #countDown} method, after which
 * all waiting threads are released and any subsequent invocations of
 * {@link #await await} return immediately.  This is a one-shot phenomenon
 * -- the count cannot be reset.  If you need a version that resets the
 * count, consider using a {@link CyclicBarrier}.
 *
 * <p>A {@code CountDownLatch} is a versatile synchronization tool
 * and can be used for a number of purposes.  A
 * {@code CountDownLatch} initialized with a count of one serves as a
 * simple on/off latch, or gate: all threads invoking {@link #await await}
 * wait at the gate until it is opened by a thread invoking {@link
 * #countDown}.  A {@code CountDownLatch} initialized to <em>N</em>
 * can be used to make one thread wait until <em>N</em> threads have
 * completed some action, or some action has been completed N times.
 *
 * <p>A useful property of a {@code CountDownLatch} is that it
 * doesn't require that threads calling {@code countDown} wait for
 * the count to reach zero before proceeding, it simply prevents any
```

Tools to Fix Multithreading issues

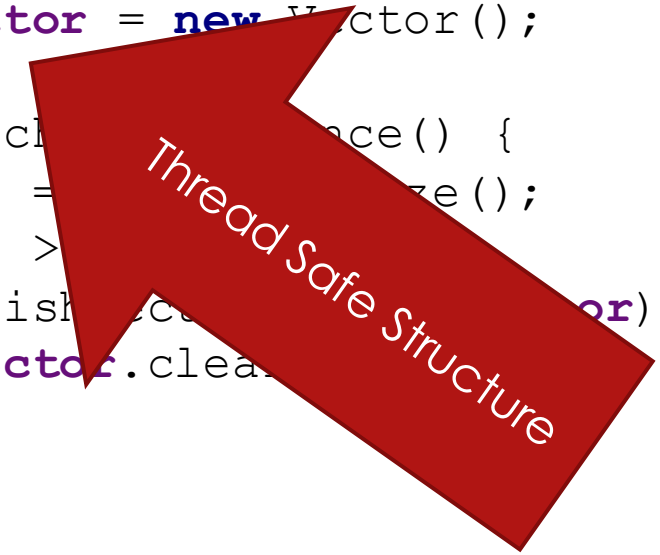
- Library Tools
  - Java.util.concurrent
    - Specific Purpose
    - 99.999% use cases
    - Documentation is GOLDEN
    - Tools are SHARP!
      - Do Understand
      - What they do

Tools to Fix Multithreading issues

```
public class Example {
    Vector myVector = new Vector();

    public void checkTolerance() {
        int size = myVector.size();
        if (size > 50) {
            publishVectorInfo(myVector);
            myVector.clear();
        }
    }

    private void publishVectorInfo(Vector myVector) {

    }
}
```

Tools to Fix Multithreading issues

```
public class Example {
    Vector myVector = new Vector();

    public void cl      nce() {
        int size =      ze();
        if (size >
            publish      or);
            myVector.clea
        }
    }

    private void publishVectorInfo(Vector myVector) {

    }
}
```

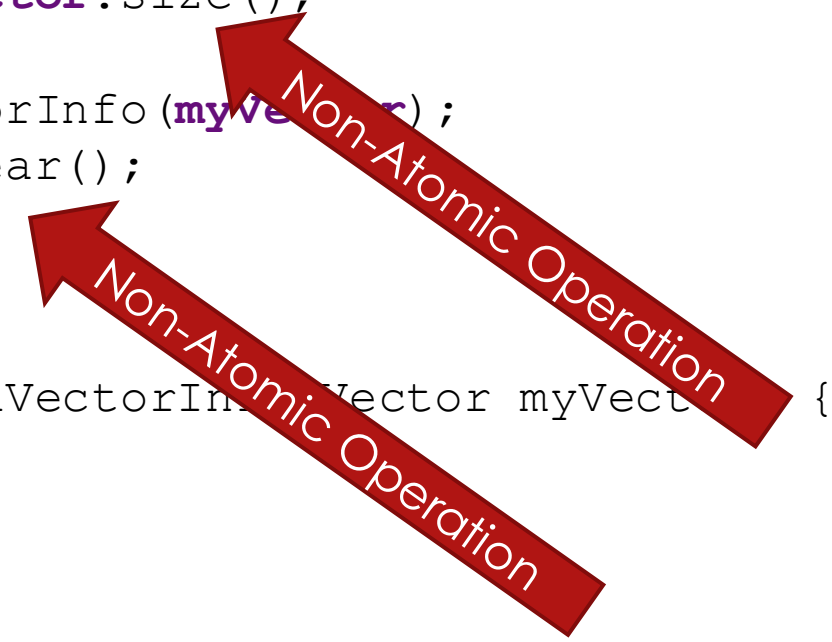


Tools to Fix Multithreading issues

```
public class Example {
    Vector myVector = new Vector();

    public void checkTolerance() {
        int size = myVector.size();
        if (size > 50) {
            publishVectorInfo(myVector);
            myVector.clear();
        }
    }

    private void publishVectorInfo(Vector myVector) {

    }
}
```

Tools to Fix Multithreading issues

```
public class Example {
    Vector myVector = new Vector();

    public void checkTolerance() {
        int size = myVector.size();
        if (size > 50) {
            publishVectorInfo(myVector);
            myVector.clear();
        }
    }

    private void publishVectorInfo(Vector myVect) {

    }
}
```



Tools to Fix Multithreading issues

```
public class Example {
    Vector myVector = new Vector();

    public void checkTolerance() {
        int size = myVector.size();
        if (size > 50) {
            publishVectorInfo(myVector);
            myVector.clear();
        }
    }

    private void publishVectorInfo(Vector myVect) {

    }
}
```

Non-Atomic Operation

Non-Atomic Operation

Tools to Fix Multithreading issues

```
public class Example {
    Vector myVector = new Vector();

    public synchronized void checkTolerance() {
        int size = myVector.size();
        if (size > 50) {
            publishVectorInfo(myVector);
            myVector.clear();
        }
    }

    private void publishVectorInfo(Vector myVector) {

    }
}
```

Tools to Fix Multithreading issues

```
public class Example {
    Vector myVector = new Vector();

    public synchronized void checkTolerance() {
        int size = myVector.size();
        if (size > 50) {
            publishVectorInfo(myVector);
            myVector.clear();
        }
    }

    private void publishVectorInfo(Vector myVector) {

    }
}
```

Tools to Fix Multithreading issues

```
public class Example {
    final Vector myVector = new Vector();

    public void checkTolerance() {
        synchronized (myVector) {
            int size = myVector.size();
            if (size > 50) {
                publishVectorInfo(myVector);
                myVector.clear();
            }
        }
    }

    private void publishVectorInfo(Vector myVector) {

    }
}
```

Tools to Fix Multithreading issues

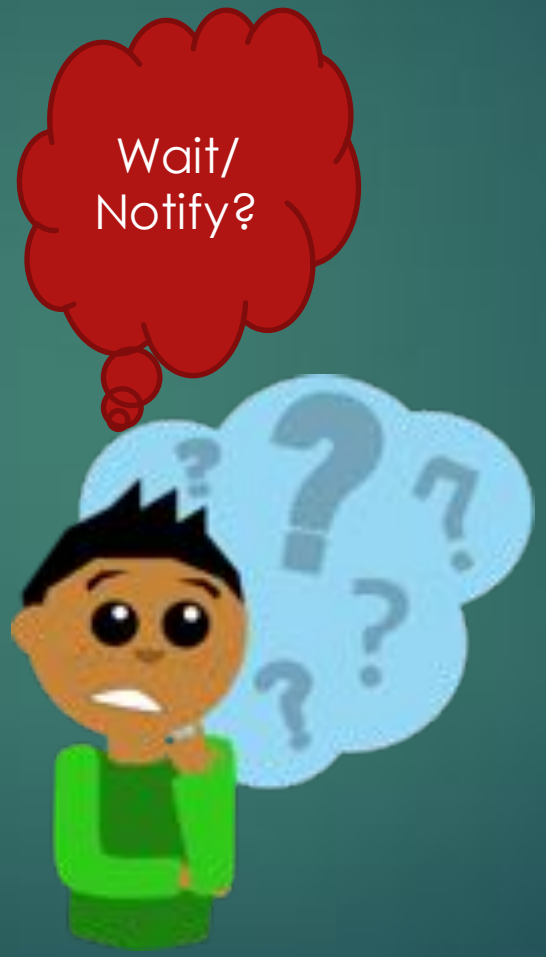
```
public class Example {
    final Vector myVector = new Vector();

    public void checkTolerance() {
        synchronized (myVector) {
            int size = myVector.size();
            if (size > 50) {
                publishVectorInfo(myVector);
                myVector.clear();
            }
        }
    }

    private void publishVectorInfo(Vector myVector) {

    }
}
```

Tools to Fix Multithreading issues

- Library Tools
  - Java.util.concurrent
    - Specific Purpose
    - 99.999% use cases
    - Documentation is GOLDEN

Tools to Fix Multithreading issues

- Library Tools
  - Java.util.concurrent
    - Specific Purpose
    - 99.999% use cases
    - Documentation is GOLDEN



Tools to Fix Multithreading issues

Tools to Fix Multithreading issues

Tools to Fix Multithreading issues

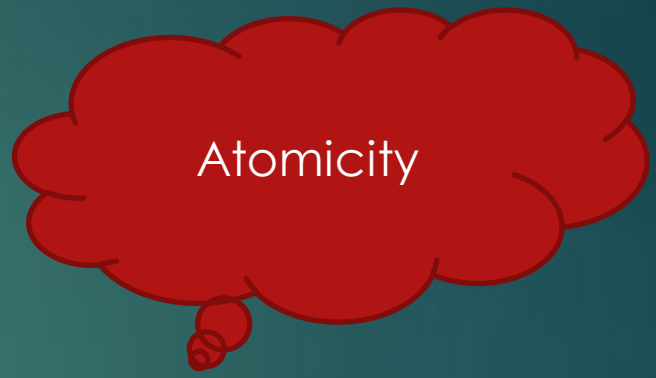
Identifying Threading Issues

Identifying Threading Issues

- How do I know there's a Threading issue?

Identifying Threading Issues

Identifying Threading Issues



Identifying Threading Issues

- Atomicity
  - 2 or more operations as One

Identifying Threading Issues

- Atomicity
  - 2 or more operations as One
  - Usually “Check-then-act” semantics

Identifying Threading Issues

- Atomicity
  - 2 or more operations as One
  - Usually “Check-then-act” semantics

```
If (map.size() > 25) {
…
   map.clear()
}
```

Identifying Threading Issues

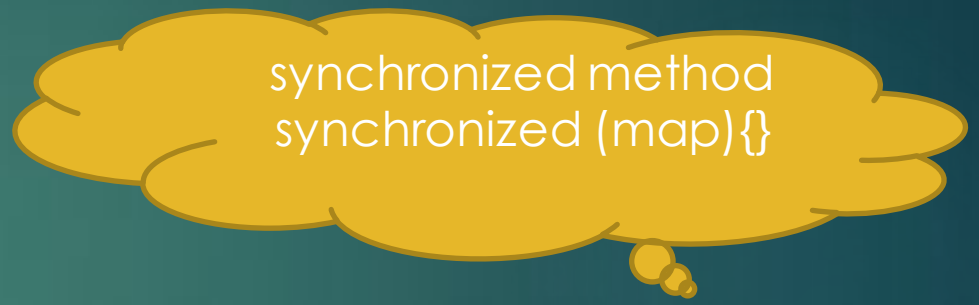
- Atomicity
  - 2 or more operations as One
  - Usually “Check-then-act” semantics

```
If (map.size() > 25) {
…
   map.clear()
}
```

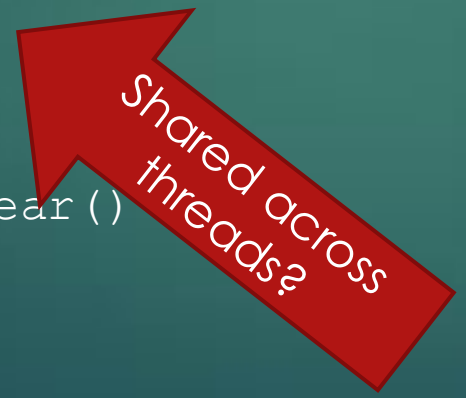
Shared across threads?

Identifying Threading Issues

- Atomicity
  - 2 or more operations as One
  - Usually “Check-then-act” semantics



```
If (map.size() > 25) {
…
   map.clear()
}
```



Identifying Threading Issues

- Atomicity
  - 2 or more operations as One
  - Usually “Check-then-act” semantics
- Mutability

Identifying Threading Issues

- Atomicity
  - 2 or more operations as One
  - Usually “Check-then-act” semantics
- Mutability
  - Other threads see values?

immutable variables
synchronized methods
volatile fields
Thread-safe collections

```
myUser.setSearchResult(result)
```

```
myUser.getSearchResult(result)
```

Identifying Threading Issues

- Atomicity
  - 2 or more operations as One
  - Usually “Check-then-act” semantics
- Mutability
  - Other threads see values?

```
myUser.setSearchResult(result)
```

Set as volatile

```
myUser.getSearchResult(result)
```

Identifying Threading Issues

- Atomicity
  - 2 or more operations as One
  - Usually “Check-then-act” semantics
- Mutability
  - Other threads see values?

```
myUser.setSearchResult(result)
```

Synchronized

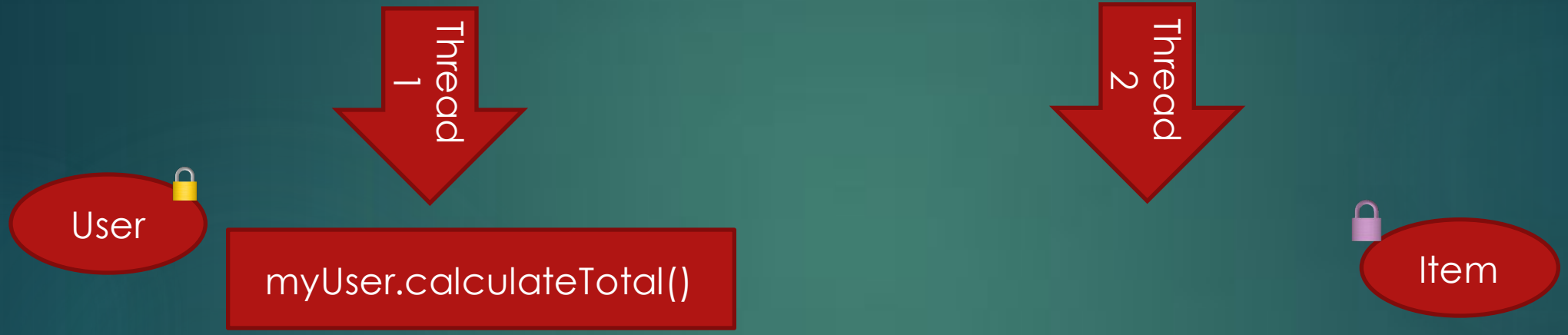
```
myUser.getSearchResult(result)
```

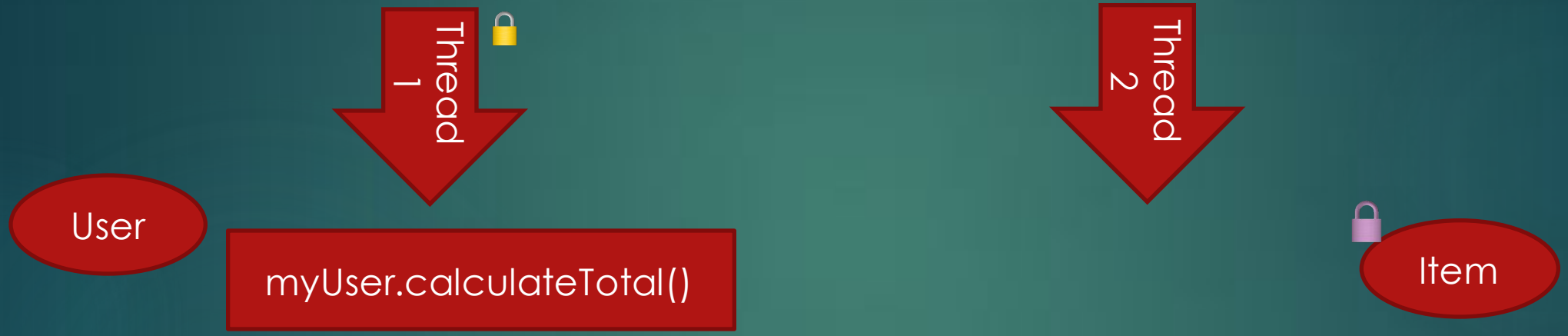
Identifying Threading Issues

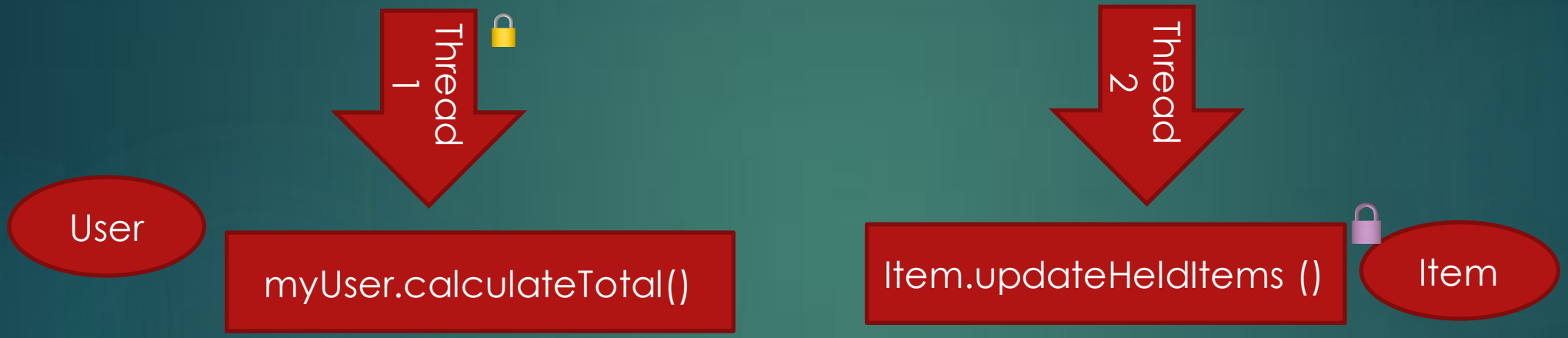
- Atomicity
  - 2 or more operations as One
  - Usually “Check-then-act” semantics
- Mutability
  - Other threads see values?
- Deadlock

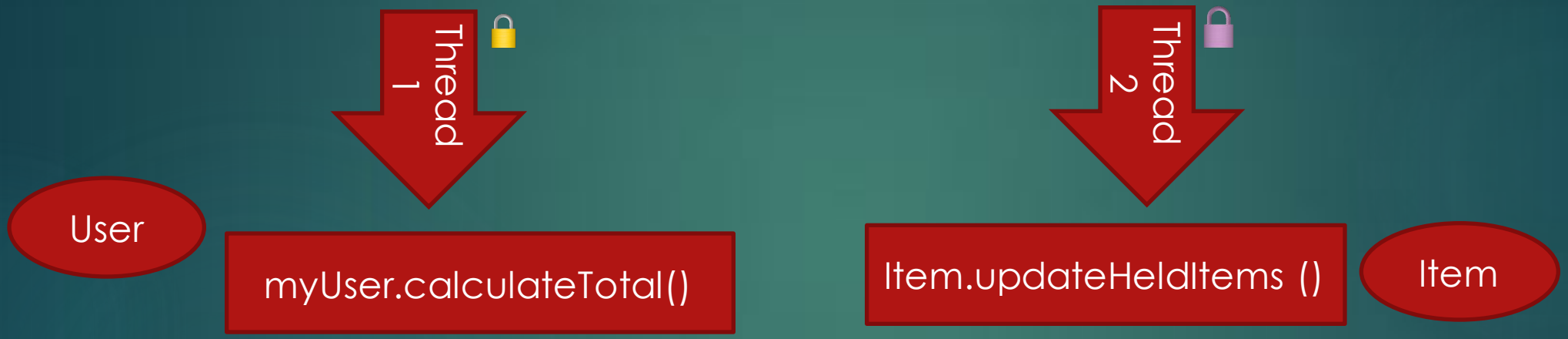
Identifying Threading Issues

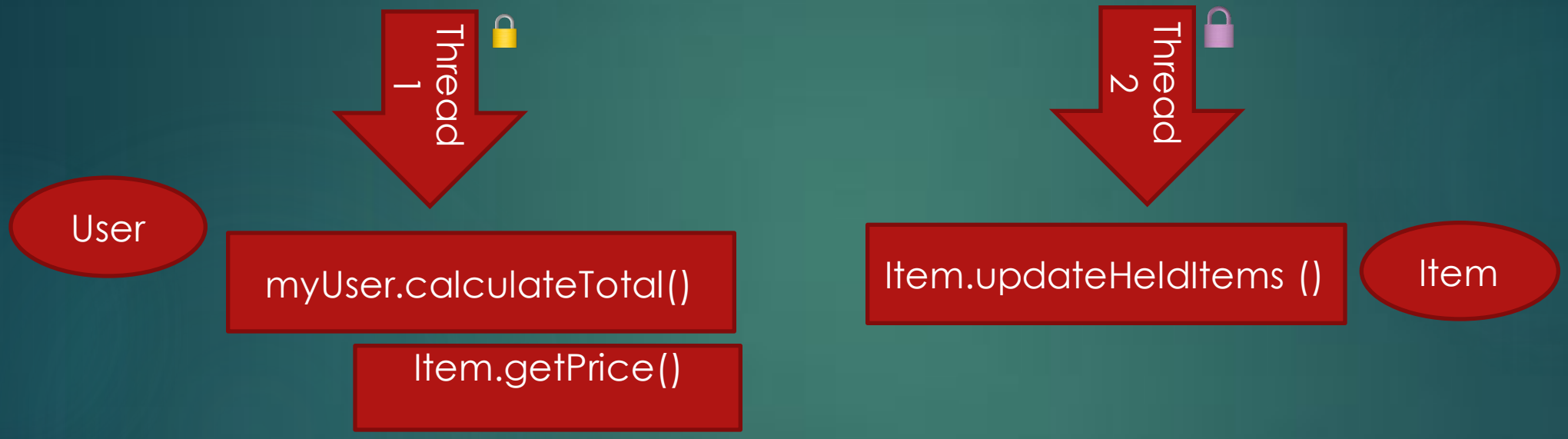
- Atomicity
  - 2 or more operations as One
  - Usually “Check-then-act” semantics
- Mutability
  - Other threads see values?
- Deadlock

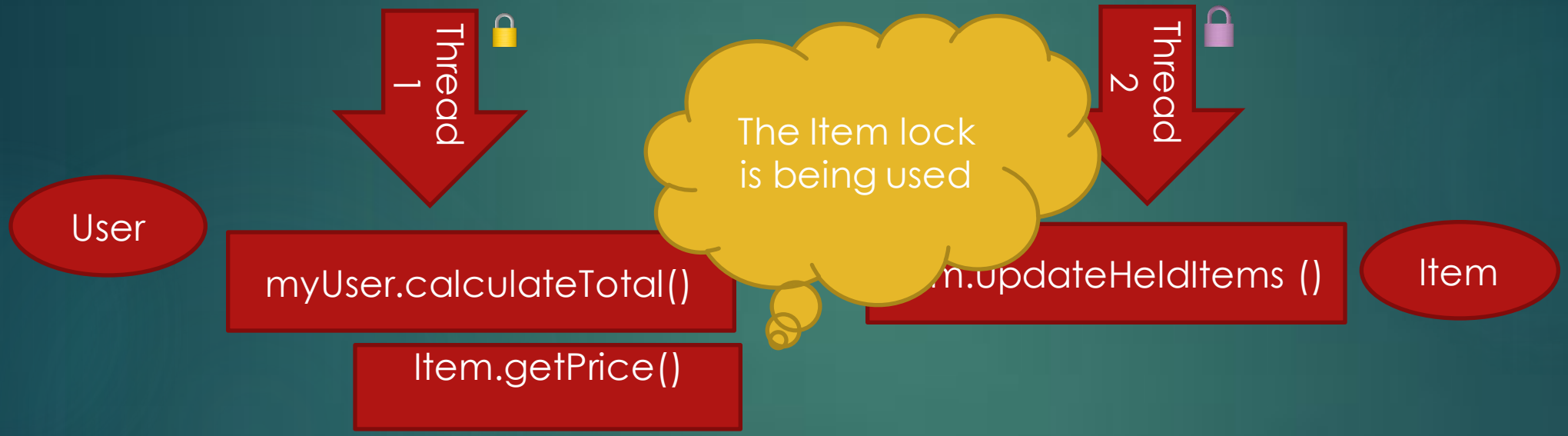

Thread 1
Thread 2
User
myUser.calculateTotal()
Item

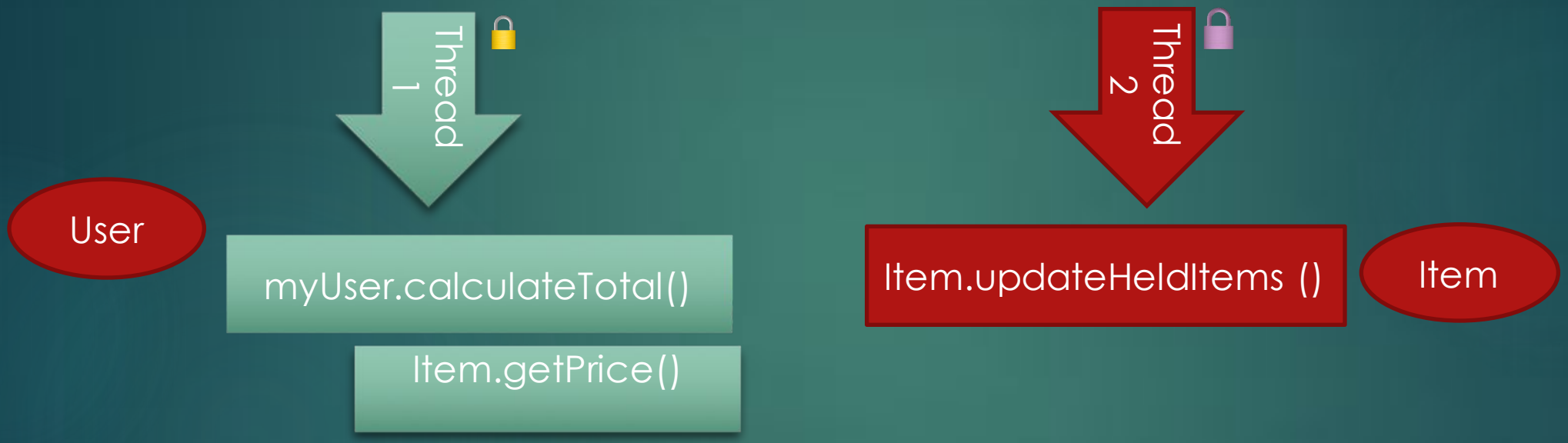

Thread 1
Thread 2
User
myUser.calculateTotal()
Item

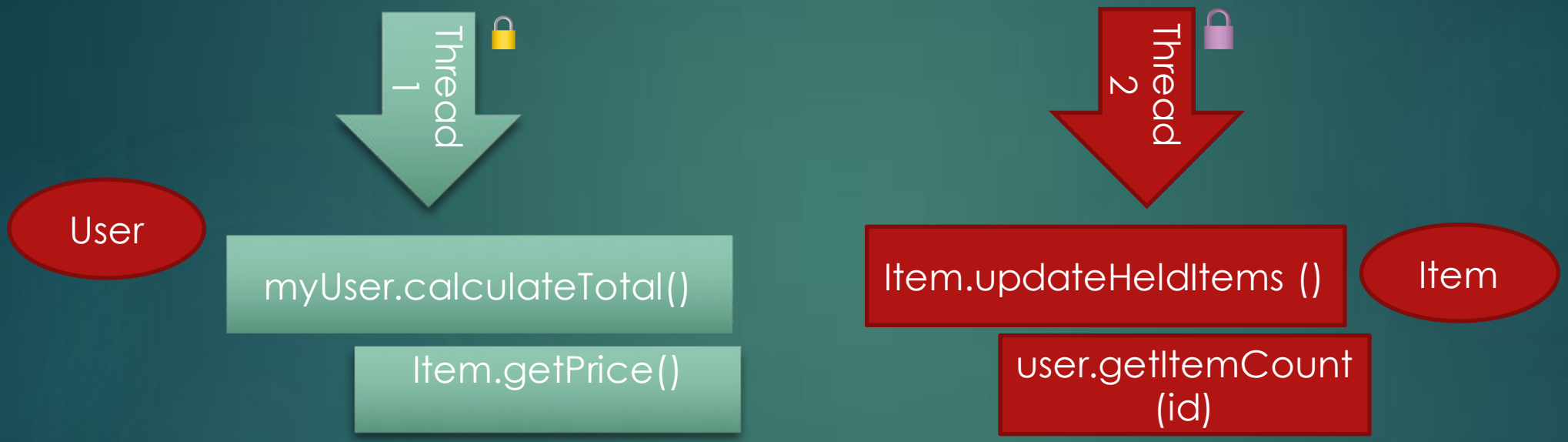

Thread 1
Thread 2
User
myUser.calculateTotal()
Item.updateHeldItems ()
Item

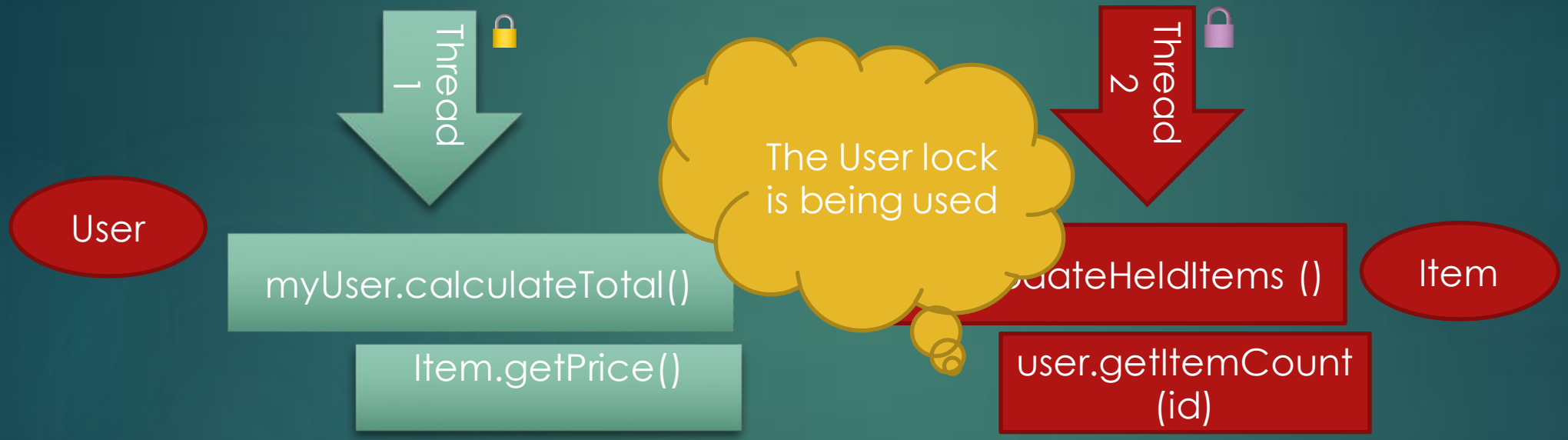

Thread 1
Thread 2
User
myUser.calculateTotal()
Item.updateHeldItems ()
Item

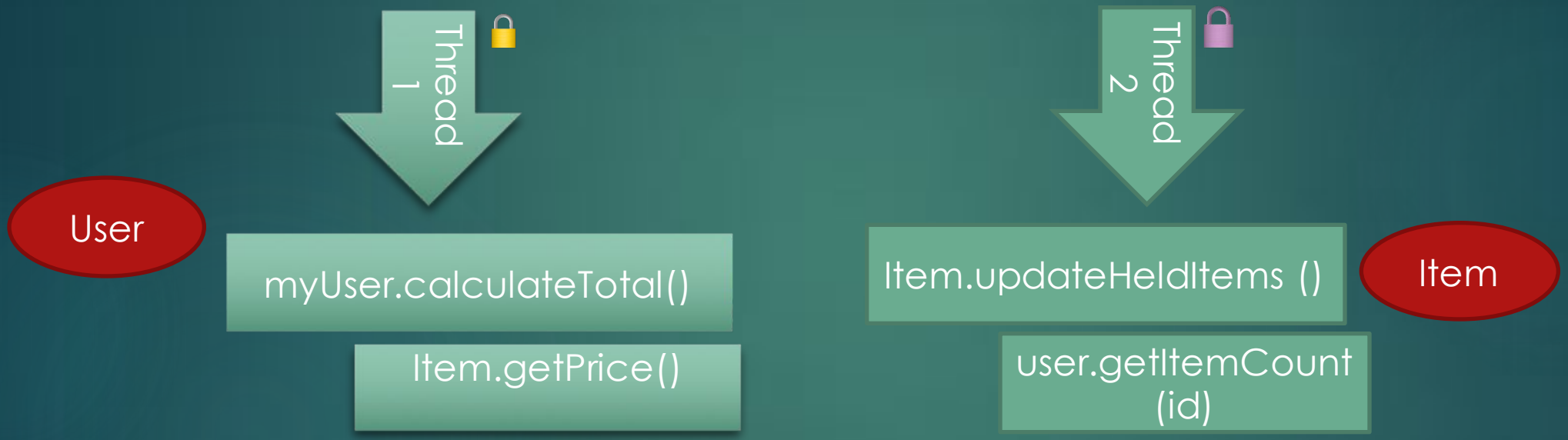

Thread 1
Thread 2
User
myUser.calculateTotal()
Item.getPrice()
Item.updateHeldItems ()
Item


Thread 1
Thread 2
The Item lock is being used
User
myUser.calculateTotal()
Item.getPrice()
m.updateHeldItems ()
Item


Thread 1
Thread 2
User
myUser.calculateTotal()
Item.getPrice()
Item.updateHeldItems ()
Item


Thread 1
Thread 2
User
myUser.calculateTotal()
Item.getPrice()
Item.updateHeldItems ()
Item
user.getItemCount (id)


Thread 1
Thread 2
The User lock is being used
User
myUser.calculateTotal()
Item.getPrice()
ateHeldItems ()
Item
user.getItemCount (id)


Thread 1
Thread 2
User
myUser.calculateTotal()
Item.getPrice()
Item.updateHeldItems ()
user.getItemCount (id)
Item


Thread 1
User
myUser.calculateTotal()
Item.getPrice()
em.updateHeldItems ()
Item
user.getItemCount (id)


Thread 1
User
myUser.calculateTotal()
Item.getPrice()
em.updateHeldItems ()
Item
user.getItemCount (id)

User

Thread 1

myUser.calculateTotal()

Item.getPrice()

Thread 2

Item.updateHeldItems ()

user.getItemCount (id)

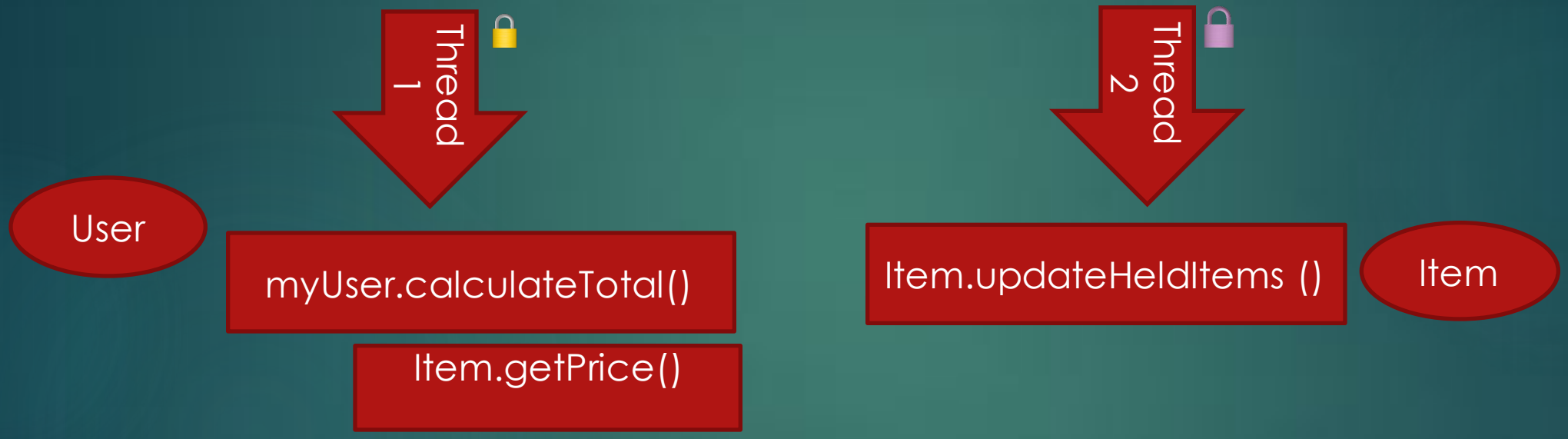
Item

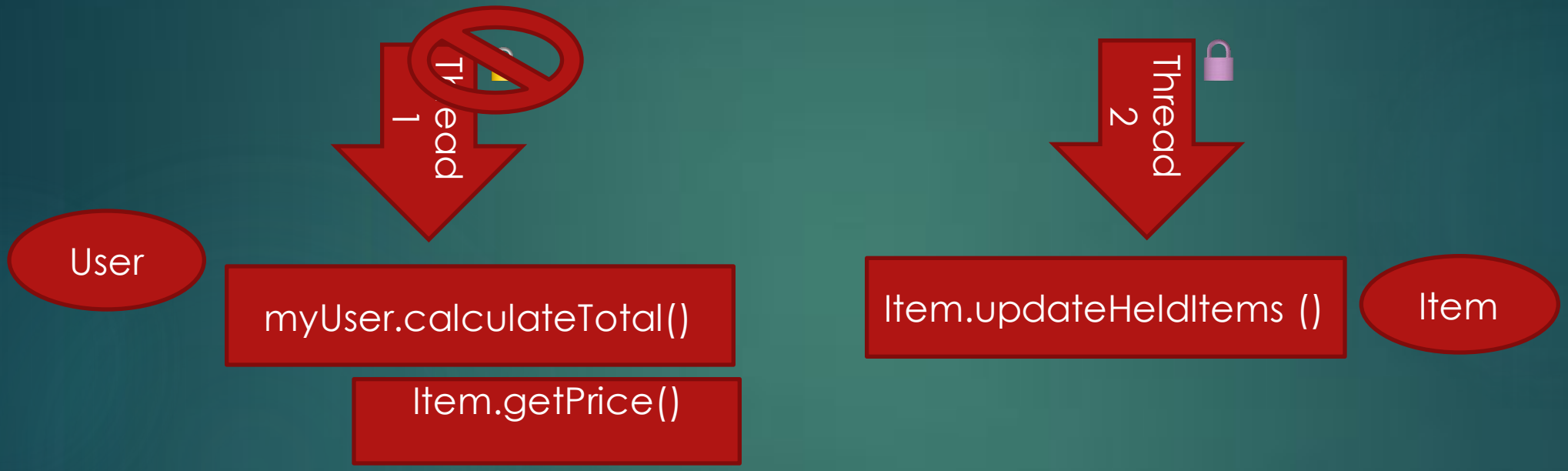
Identifying Threading Issues

- Atomicity
  - 2 or more operations as One
  - Usually “Check-then-act” semantics
- Mutability
  - Other threads see values?
- Deadlock

Identifying Threading Issues

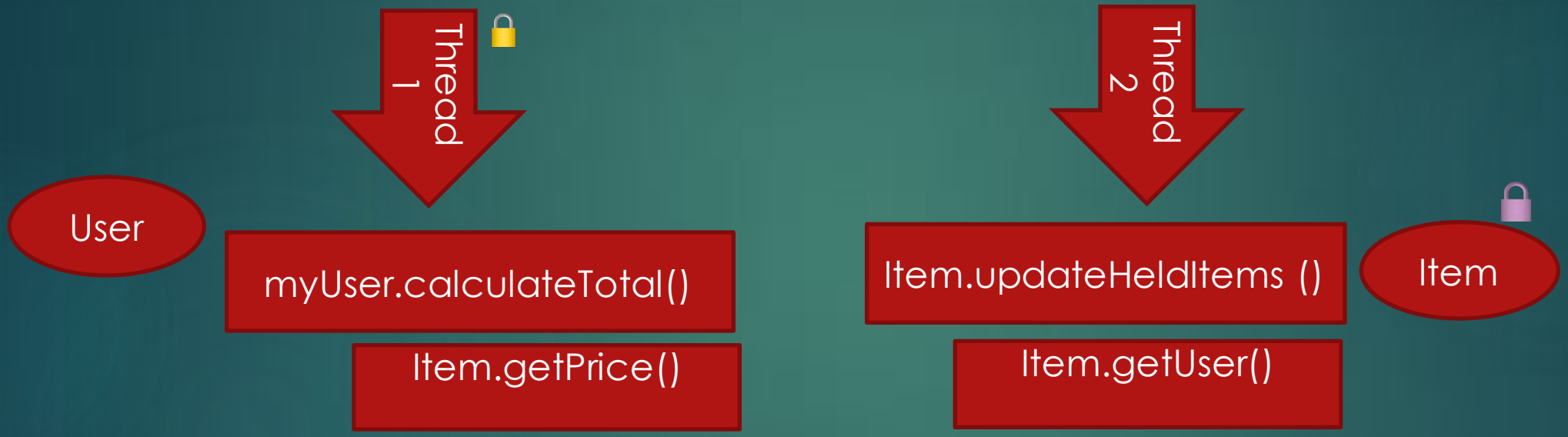
- Atomicity
  - 2 or more operations as One
  - Usually “Check-then-act” semantics
- Mutability
  - Other threads see values?
- Deadlock
  - Not letting “Locks Escape”


Thread 1
Thread 2
User
myUser.calculateTotal()
Item.getPrice()
Item.updateHeldItems ()
Item

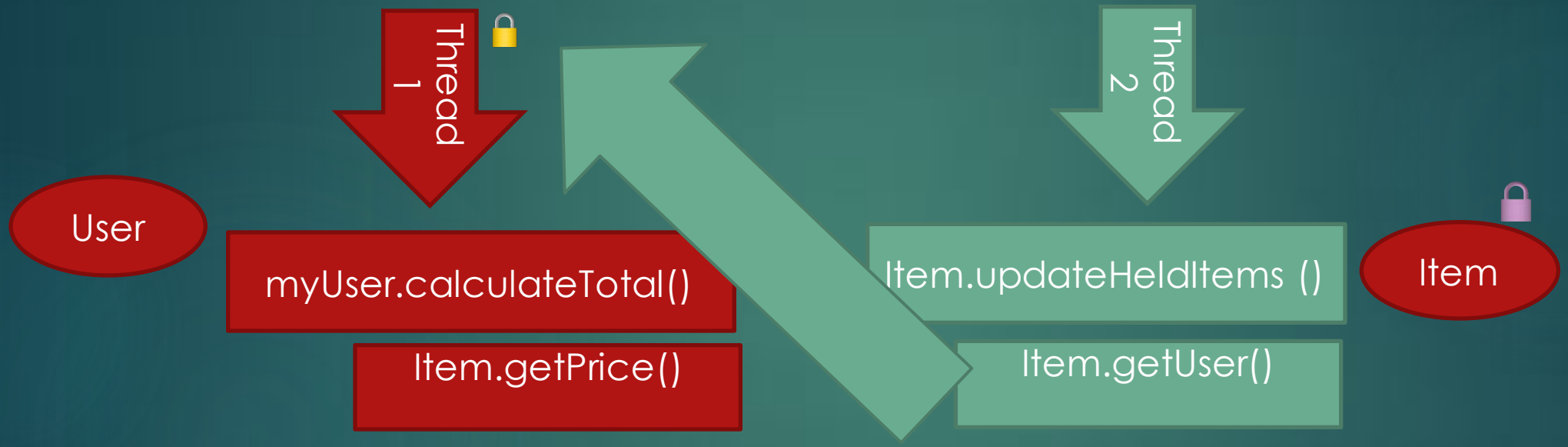

Thread 1
Thread 2
User
myUser.calculateTotal()
Item.getPrice()
Item.updateHeldItems ()
Item

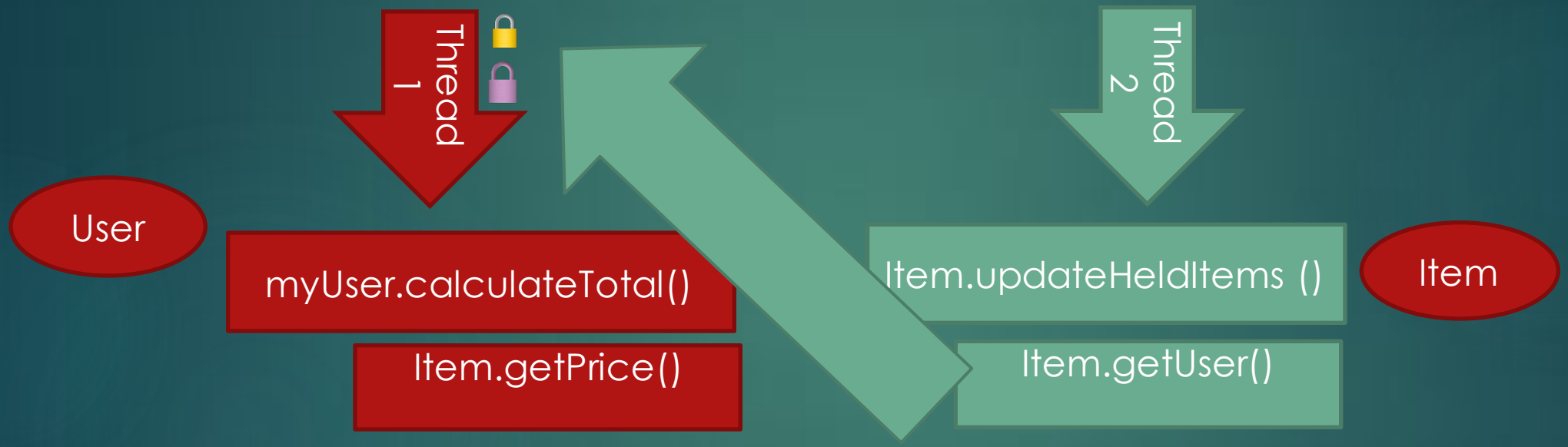
Identifying Threading Issues

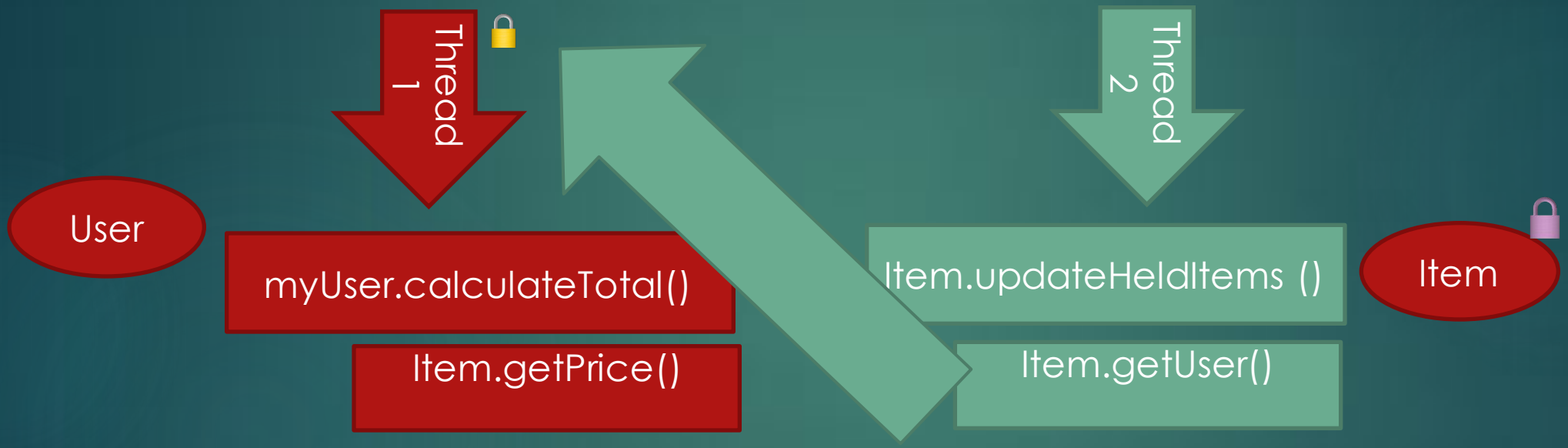
- Atomicity
  - 2 or more operations as One
  - Usually “Check-then-act” semantics
- Mutability
  - Other threads see values?
- Deadlock
  - Not letting “Locks Escape”
  - Always acquire on same order

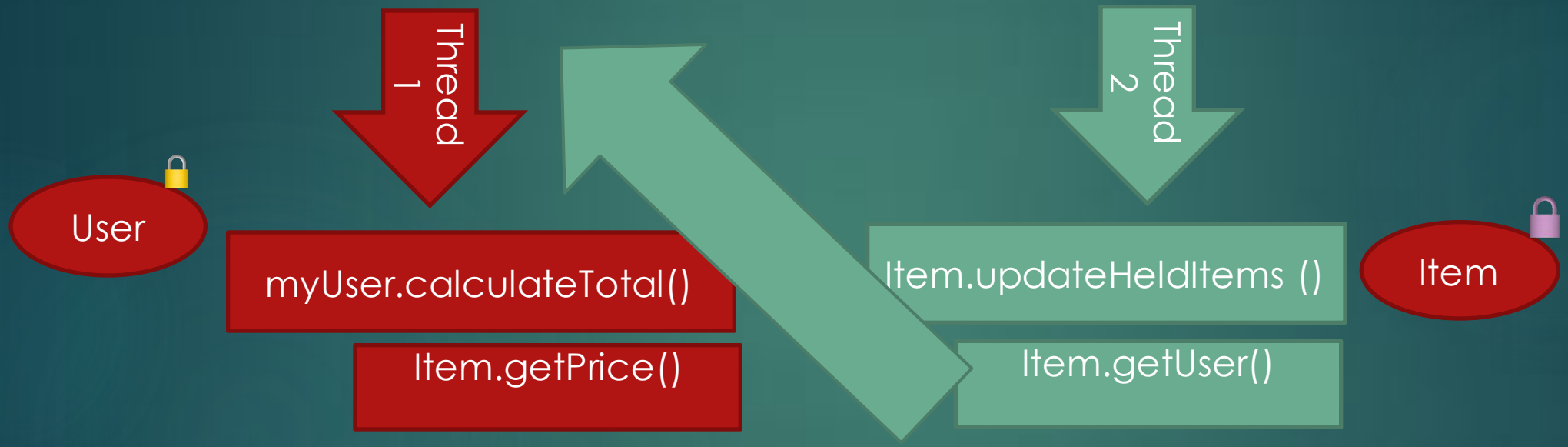

Thread 1
Thread 2
User
myUser.calculateTotal()
Item.getPrice()
Item.updateHeldItems ()
Item.getUser()
Item

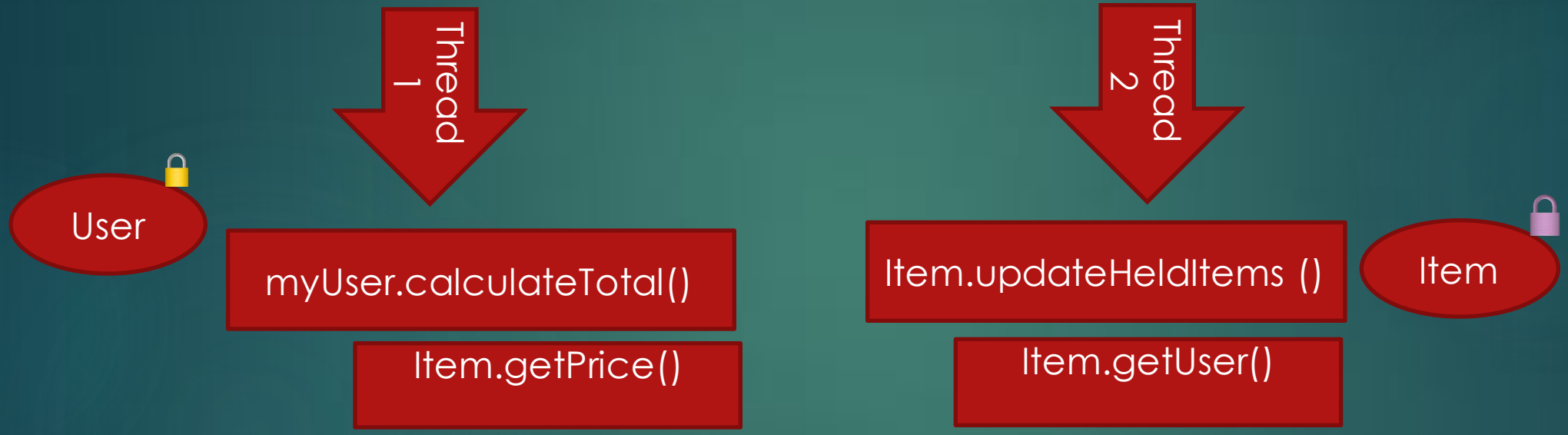

Thread 1
Thread 2
User
myUser.calculateTotal()
Item.getPrice()
Item.updateHeldItems ()
Item.getUser()
Item

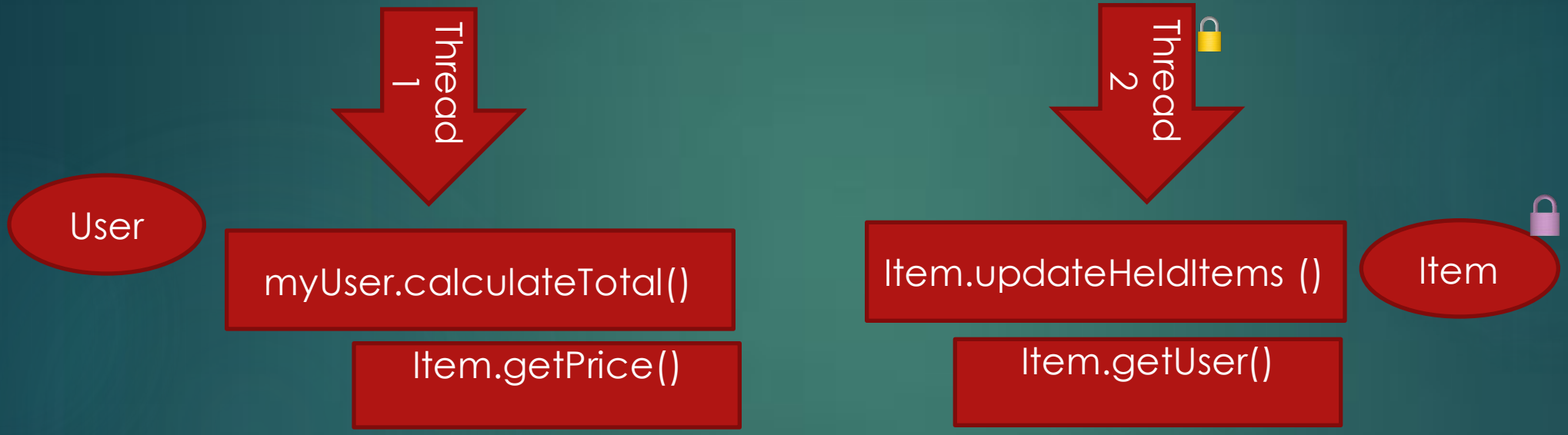

Thread 1
Thread 2
User
myUser.calculateTotal()
Item.getPrice()
Item.updateHeldItems ()
Item.getUser()
Item

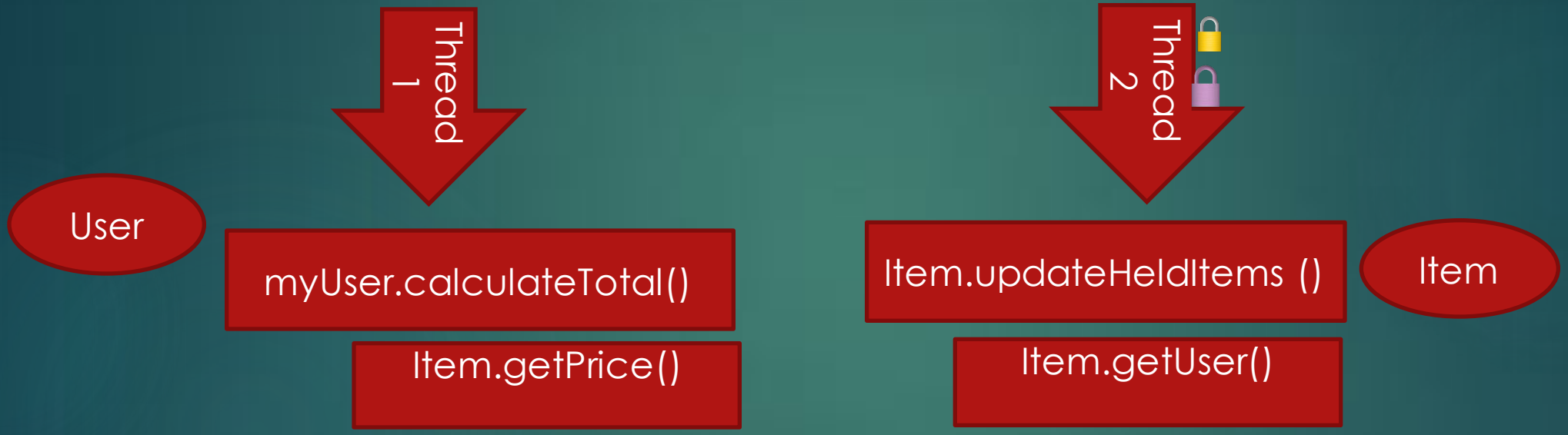

Thread 1
Thread 2
User
myUser.calculateTotal()
Item.getPrice()
Item.updateHeldItems ()
Item.getUser()
Item


Thread 1
Thread 2
User
myUser.calculateTotal()
Item.getPrice()
Item.updateHeldItems ()
Item.getUser()
Item


Thread 1
Thread 2
User
myUser.calculateTotal()
Item.getPrice()
Item.updateHeldItems ()
Item.getUser()
Item


Thread 1
Thread 2
User
myUser.calculateTotal()
Item.getPrice()
Item.updateHeldItems ()
Item.getUser()
Item

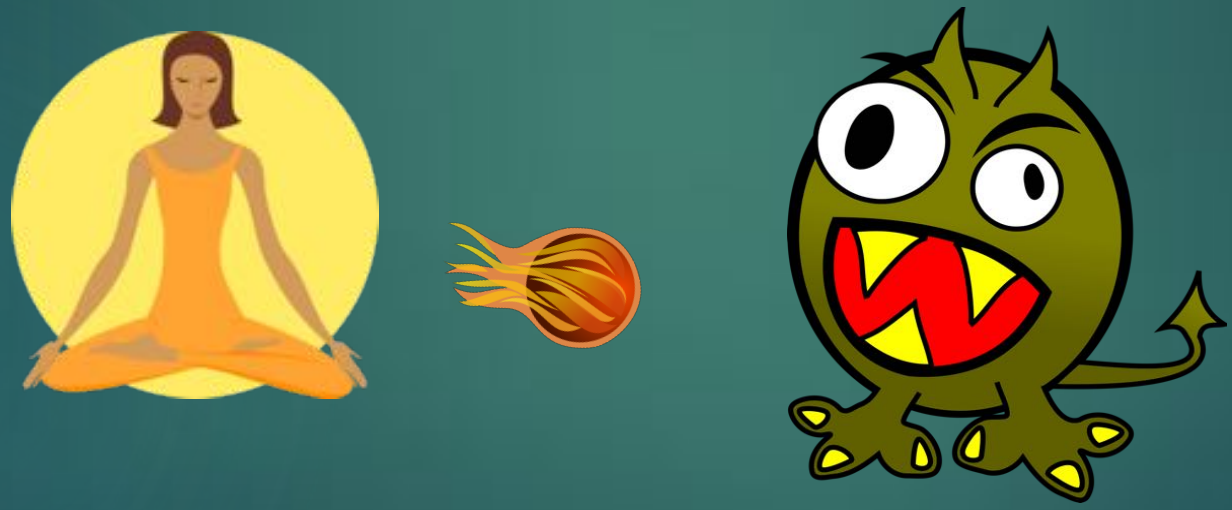

Thread 1
Thread 2
User
myUser.calculateTotal()
Item.getPrice()
Item.updateHeldItems ()
Item.getUser()
Item


Thread 1
Thread 2
User
myUser.calculateTotal()
Item.getPrice()
Item.updateHeldItems ()
Item
Item.getUser()

Identifying Threading Issues

- Atomicity
  - 2 or more operations as One
  - Usually “Check-then-act” semantics
- Mutability
  - Other threads see values?
- Deadlock
  - Not letting “Locks Escape”
  - Always acquire on same order

CODE?




BOOM!

Timmy

so we talked and we talked about synchronized blocks
we learned of atomicity, deadlocks and such
that immutability has two ems in it
but it's useful for thread safety

we also learn about concurrent collections
cpu caches and volatile fields
about synchronized blocks and object locks
how to code them and prevent a hang-up

He then became one of a kind of developers
the ones that get paid the big hourly rates
no problem was hard for him to fix up
he went and talked at conferences and Jug's

Timmy

oh Timmy, oh Timmy how much have you learned
that immutables are king for threading problems
....that you always check ...for the check-then-act
and you revise the singletons, in your code.

Timmy

Java Pub House

www.javapubhouse.com

OFFHEAP

www.javaoffheap.com