USF UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

Synchronization

CS 272 Software Development

Department of Computer Science
https://www.cs.usfca.edu/

Motivation

#	Thread 1: **x++;**	Thread 2: **x--;**
	read value of **x**	read value of **x**
	calculate **x + 1**	calculate **x - 1**
	assign **x** to calculated result	assign **x** to calculated result

Motivation

#	Thread 1: **x++;**	Thread 2: **x--;**
1	read **x = 1**	
2	calculate **1 + 1 = 2**	
3	assign **x = 2**	
4		read **x = 2**
5		calculate **2 - 1 = 1**
6		assign **x = 1**
7		

Motivation

#	Thread 1: `x++;`	Thread 2: `x--;`
1	read `x = 1`	
2	calculate `1 + 1 = 2`	
3	assign `x = 2`	
4		read `x = 2`
5		calculate `2 - 1 = 1`
6		assign `x = 1`
7	final value `x = 1`	

Motivation

#	Thread 1: `x++;`	Thread 2: `x--;`
1	read `x = 1`	
2		read `x = 1`
3	calculate `1 + 1 = 2`	
4		calculate `1 − 1 = 0`
5	assign `x = 2`	
6		assign `x = 0`
7		

Motivation

#	Thread 1: **x++;**	Thread 2: **x--;**
1	read **x = 1**	
2		read **x = 1**
3	calculate **1 + 1 = 2**	
4		calculate **1 - 1 = 0**
5	assign **x = 2**	
6		assign **x = 0**
7	final value **x = 0**	

Motivation

#	Thread 1: **x++;**	Thread 2: **x--;**
1	read **x = 1**	
2		read **x = 1**
3	calculate **1 + 1 = 2**	
4		calculate **1 – 1 = 0**
5		assign **x = 0**
6	assign **x = 2**	
7	final value **x = 2**	

Problems

- Operators x++ and x-- are not **atomic** operations
    - Unable to divide operation(s)
    - Unable to interrupt when multithreading
    - All operations succeed or all fail (no partial results)
- Shared data is modified between read and use
    - Shared variable x is not **thread safe**

Thread Safety

- An object is **thread safe** if it maintains a valid or consistent state even when accessed concurrently
- Includes all constants and **immutable** objects
  - `String` or primitive types that are `final`
- Includes some **mutable** objects
  - `StringBuffer`, `java.util.concurrent.*`

Thread Safety

- Use **synchronization** is coordinate threads
  - Use to protect objects that are not thread safe
  - Use to provide atomic blocks of code
- Synchronization in Java
  - Use **`synchronized`** functions or blocks of code
  - Use **`volatile`** variables
  - Use specialized **lock** objects

Synchronized Blocks

- Must specify an object to use as a lock
  - Any calls to `wait()` or `notify()` within block must be called on lock object
- Exact behavior depends on type of object used
  - A class member versus an instance member versus an inner instance member all behave differently

Synchronized Blocks

- A thread entering block must attempt to **acquire** lock
  - Only one thread may hold lock object at once
  - Multiple blocks may use the same lock object
- The thread is **blocked** until able to obtain lock object
- The lock object is automatically **released** when a thread exits the `synchronized` block

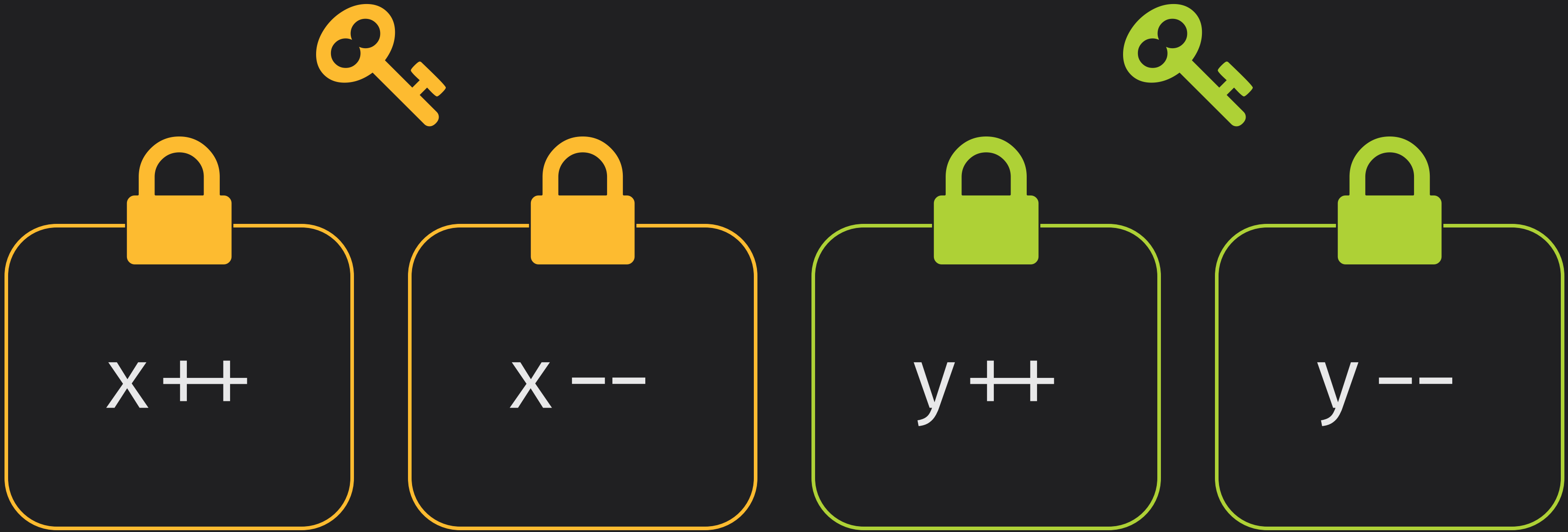
Synchronization Example

```
private Object lock;
private int a;
```

```
public void increment {
  synchronized (lock) {
    a++;
  }
}
```

```
public void decrement {
  synchronized (lock) {
    a--;
  }
}
```

Synchronization Example



Synchronization Example

```
private Object lock;
private int a;
```

```
public void increment {
  synchronized (lock) {
    a++;
  }
}
```

```
public void decrement {
  synchronized (lock) {
    a--;
  }
}
```

Synchronization Example

```java
// private Object lock;
private int a;

public void increment {
  synchronized (this) {
    a++;
  }
}
```

```java
public void decrement {
  synchronized (this) {
    a--;
  }
}
```

Synchronization Example

```
private int a;

public synchronized void increment {
  a++;
}

public synchronized void decrement {
  a--;
}
```

Synchronized Methods

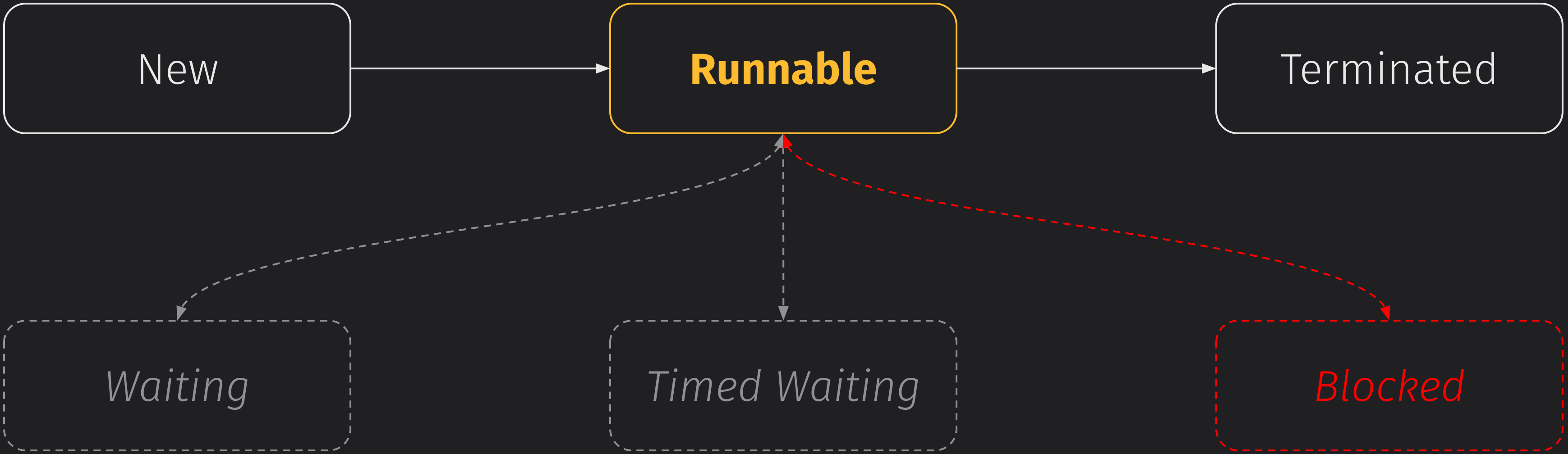
- Any method may be declared **`synchronized`**
  - `public synchronized void method()`
- Equivalent to placing all code within method in a **`synchronized (this)`** block
- All **`synchronized`** methods within a class use the same lock and may not run concurrently

** [Using “this” to handle synchronization can cause security issues...]() ***

Synchronization Issues

- Protects code blocks, **NOT** objects
  - Does not protect the lock or any objects within block
- Must be used consistently to provide **thread safety**
  - Objects accessed within a block may still be accessed concurrently elsewhere in code
- Causes **blocking**, which slows down code

Thread States



https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/lang/Thread.State.html

USF UNIVERSITY OF SAN FRANCISCO

CHANGE THE WORLD FROM HERE

Software Development
Department of Computer Science

Professor Sophie Engle
sjengle.cs.usfca.edu