Multi-Threading III

Andrew Hu

Administrivia

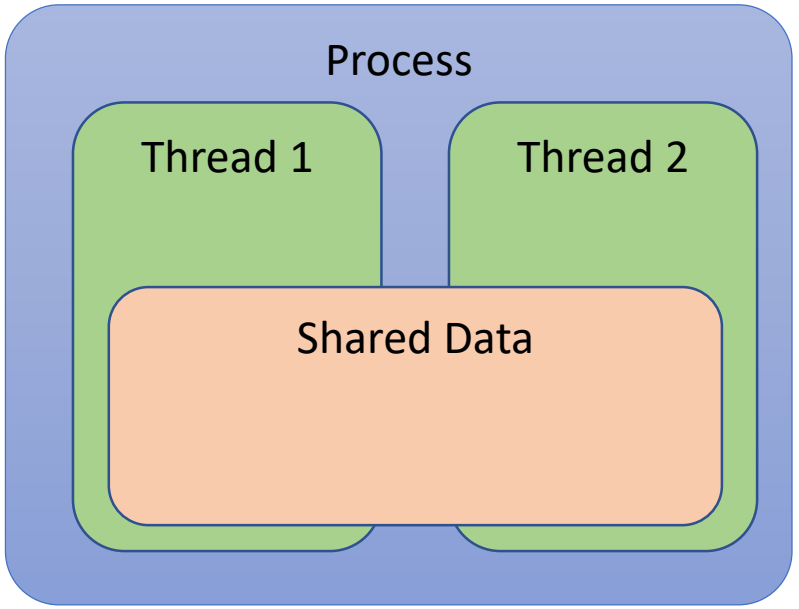
- Attendance link same as always
- http://tinyurl.com/uwigem/18sp/attendance/

Agenda

- Race conditions
- Java-isms in concurrency
- Peek under the hood

Review: Threads

- Processes can have multiple threads
- Threads can share data between each other
- Threads can write/read the same data at the "same time" (!)



Writing at the same time

```
void run() {
  this.data = 42;
  System.out.println("Set to 42");
}
```

```
void run() {
  this.data = 297;
 System.out.println("Set to 297");
}
```

```
static void main(String[] args) {
  Integer data = new Integer(0);
  //Start the two threads
  Child c1 = new Child(data);
  Child c2 = new Child(data);
  c1.start();
  c2.start();

  Thread.sleep(1000);
  System.out.println("Data: "+data);
}
```

```
Set to 297
Set to 42
Data: 42
```

```
Set to 42
Set to 297
Data: 297
```

Reading/Writing at the same time

```
void run() {
  if (this.data == 0) {
    this.data = 42;
    System.out.println("Set to 42");
  }
}
```

```
void run() {
  if (this.data == 0) {
    this.data = 297;
    System.out.println("Set to 297");
  }
}
```

```
static void main(String[] args) {
  Integer data = new Integer(0);
  //Start the two threads
  Child c1 = new Child(data);
  Child c2 = new Child(data);
  c1.start();
  c2.start();

  Thread.sleep(1000);
  System.out.println("Data: "+data);
}
```

```
Set to 297
Set to 42
Data: 42
```

```
Set to 42
Set to 297
Data: 297
```

```
Set to 42
Data: 42
```

```
Set to 297
Data: 297
```

Huh? Shouldn't this fix it?

Stack Example

```
class IntStack {
  private int[] array = new int[SIZE];
  private int index = -1;

  boolean isEmpty() {
    return index == -1;
  }

  void push(int val) {
    index++;
    array[index] = val;
  }

  int pop() {
    index--;
    return array[index+1];
  }
}
```

```
int peek(IntStack stack) {
  int res = stack.pop();
  stack.push(res);
  return res;
}
```

peek() code

```
int res = pop();



push(res);

return res;
```

user code

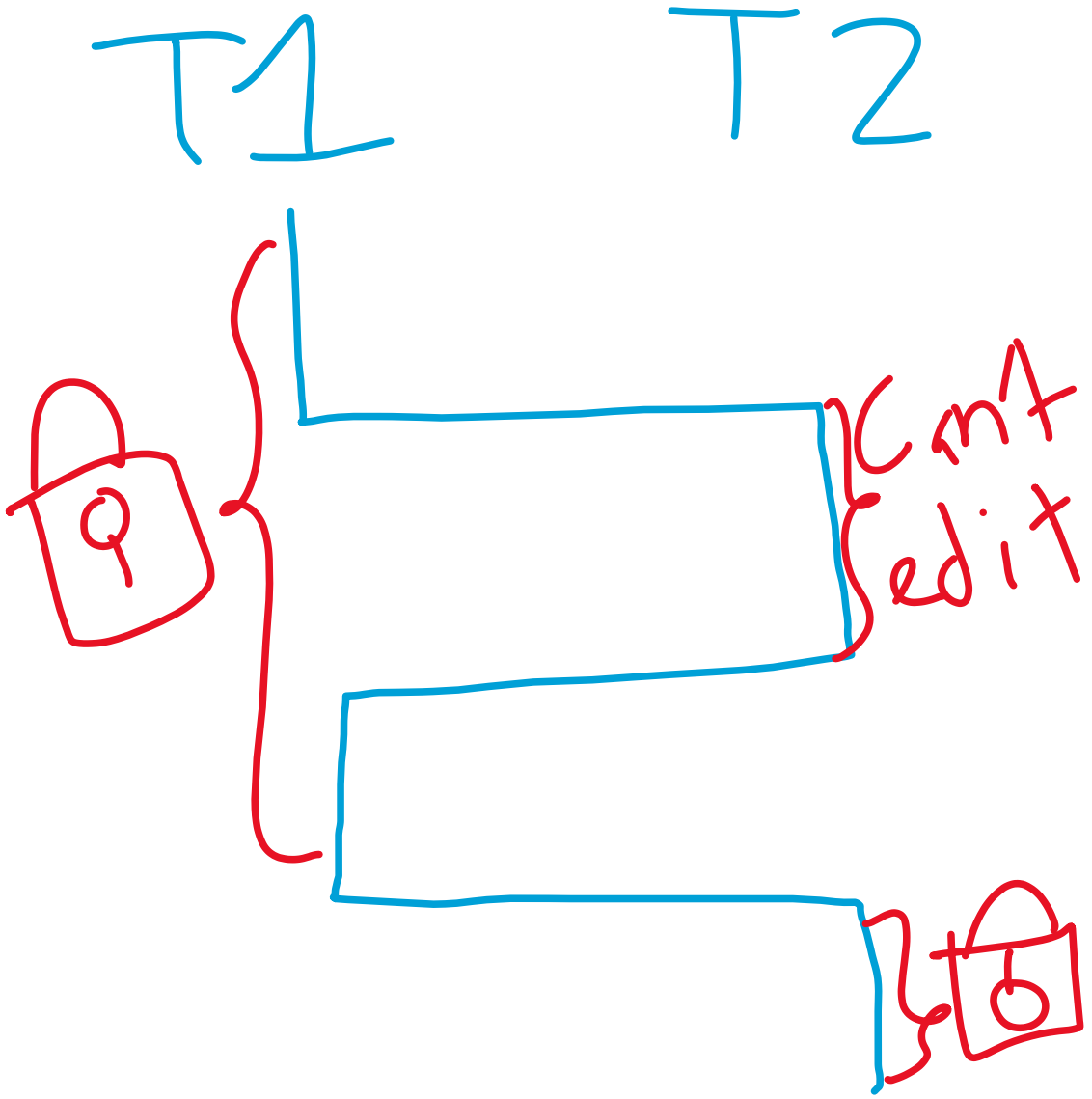
```
push(42);



isEmpty();
```

What is the problem?

- When two threads
  - Write to an object at the same time
  - Read an object, when another is writing
- This causes “nondeterministic behavior”
  - Basically, we don’t know what’ll happen

Solution: Locks

- Only give permission to one thread to modify the object, and don't let any else modify it *until they are done*
- Must be built-in to the language & OS



How this works in Java

```
class IntStack {
  private int[] array = new int[SIZE];
  private Integer index = -1;

  boolean isEmpty() {
    synchronized (index) {
      return index == -1;
  }}

  void push(int val) {
    synchronized (array) {
      index++;
      array[index] = val;
  }}

  int pop() {
    synchronized (array) {
      index--;
      return array[index+1];
  }}
}
```

```
int peek(IntStack stack) {
  synchronized (array) {
    int res = stack.pop();
    stack.push(res);
    return res;
}}
```

pop() code

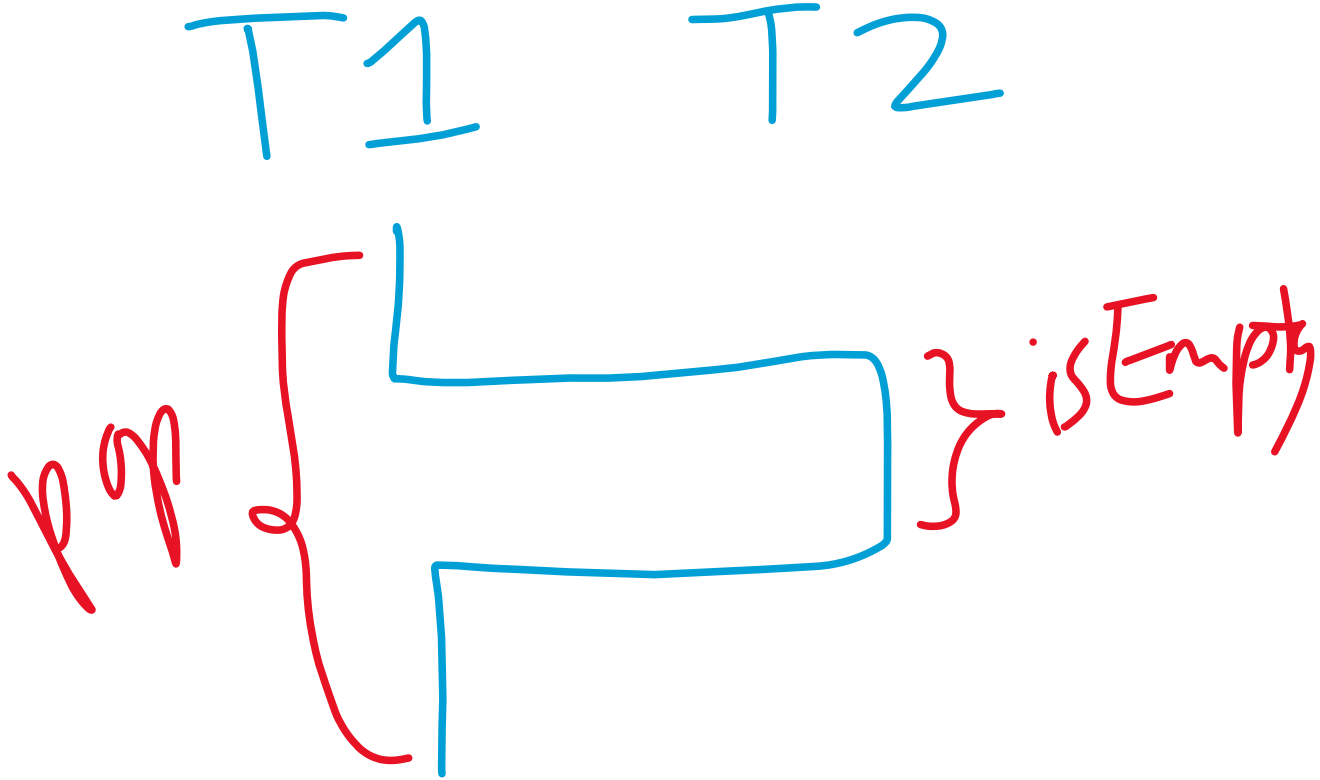
```
//acquire array lock
index--;


push(res);

return res;
```

user code

```
push(42);
//release array lock


//acquire index lock
isEmpty();
```

What was the problem?

- We were using different locks, and allowed two methods on this object to run “at the same time”



Solution: Locking the Object

```
class IntStack {
  private int[] array = new int[SIZE];
  private Integer index = -1;

  boolean isEmpty() {
    synchronized (array) {
      return index == -1;
  }}

  void push(int val) {
    synchronized (array) {
      index++;
      array[index] = val;
  }}

  int pop() {
    synchronized (array) {
      index--;
      return array[index+1];
  }}
}
```

```
int peek(IntStack stack) {
  synchronized (array) {
    int res = stack.pop();
    stack.push(res);
    return res;
}}
```

pop() code

```
//acquire array lock
index--;



push(res);
return res;
```

user code

```
push(42);
//release array lock


//do nothing, since
//the lock is taken
isEmpty();


//now isEmpty runs
```

Thread Overhead

- It takes a long time to create a thread
- It takes a short time to put a thread to sleep
- It takes a short time to wake up a thread
- What happens if we create a bunch of threads and put all but one to sleep?

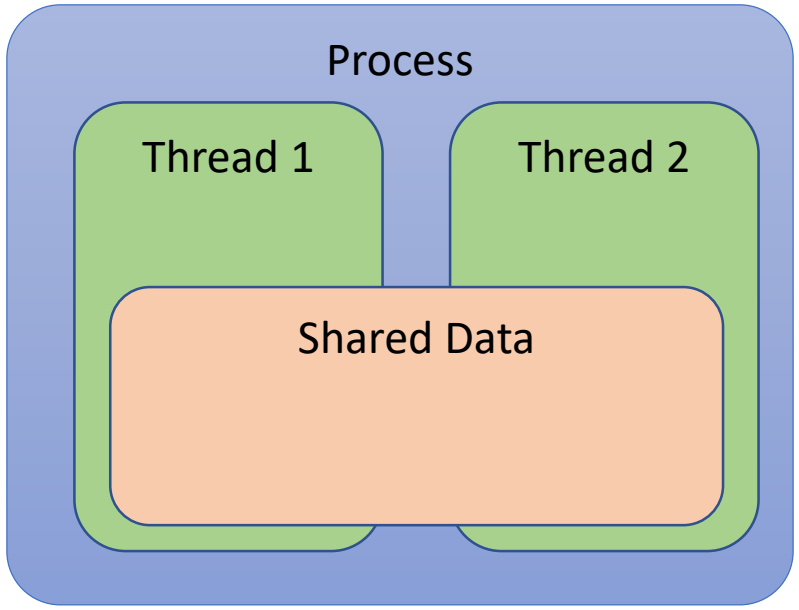
Thread Pools

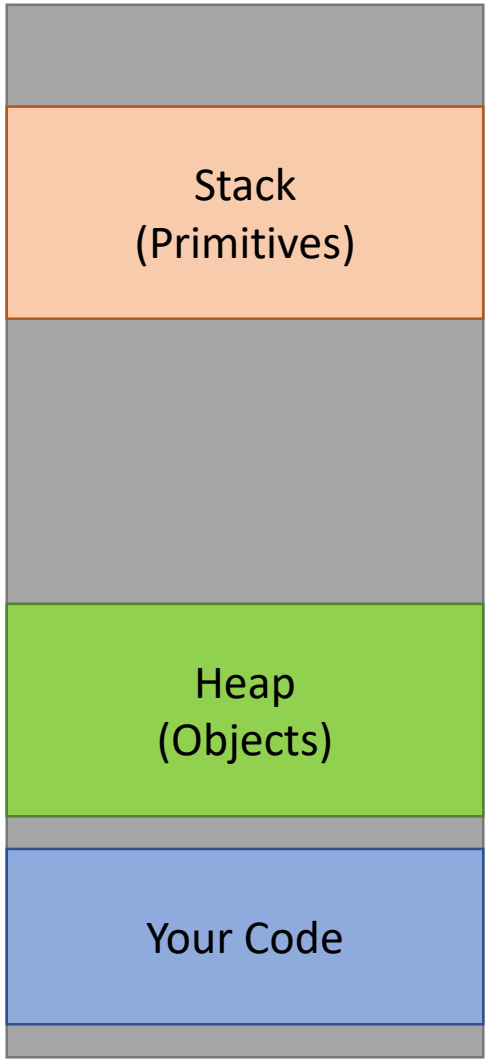
- Create a bunch of threads when you start the program
- Put them all to sleep, and only wake them up when you need them
- Turns out this is allows you to “create” threads almost instantly
- Trade-off between start up time and memory usage, or thread creation time

Java: Thread vs Runnable

- The Java thread object is a special language built in object
- When start() is called on it, it creates a new thread, and calls run()
- The Java Runnable interface describes a class that has the run() method
- It can be passed into various Java libraries that then run your code
  - e.g. some thread pools, or event-based/callback libraries

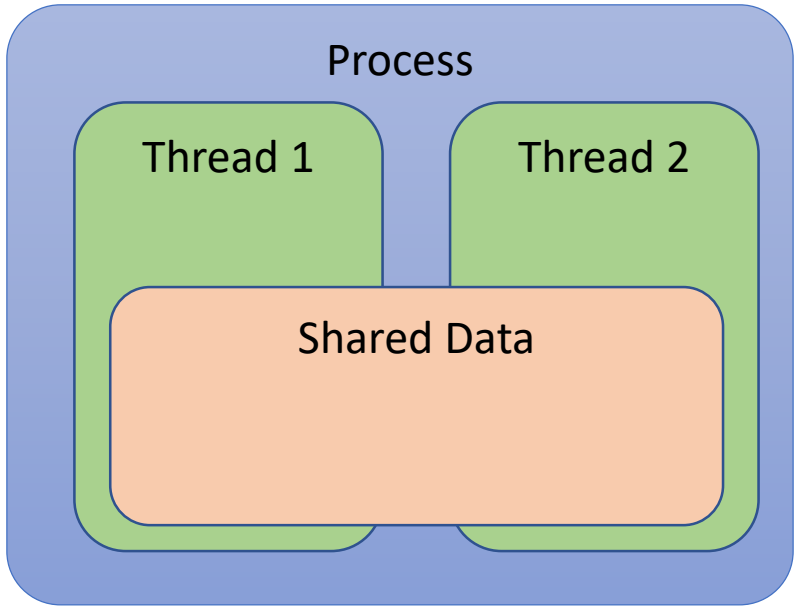
Peek Under the Hood



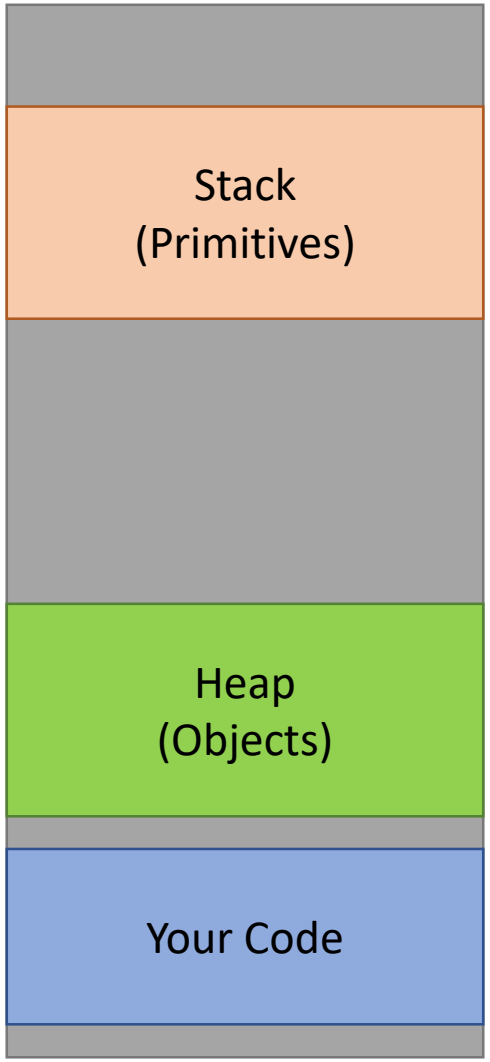


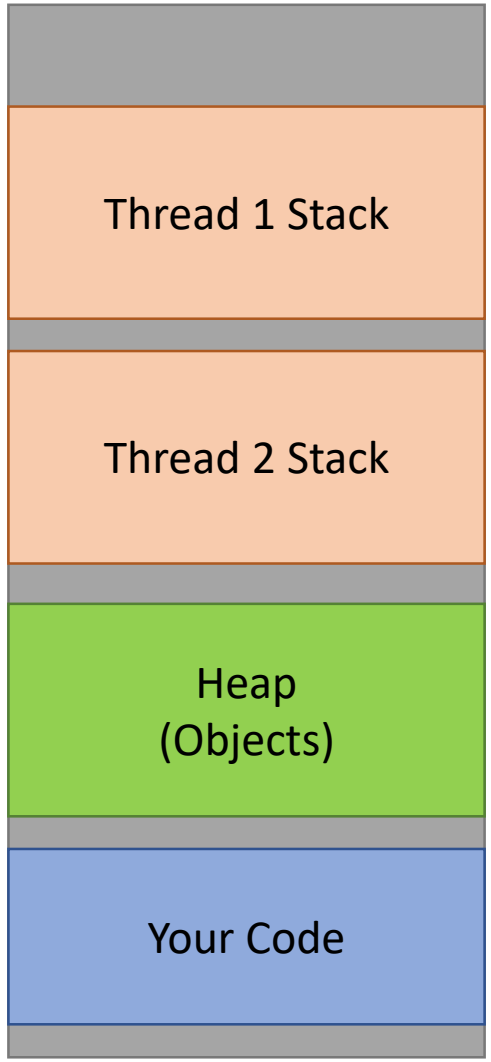
How is it possible for two threads to affect the same data?

Peek Under the Hood



How is it possible for two threads to affect the same data?





Simultaneous Multi-Threading

- Remember when we said you can't just take one program and run it on two cores? I kind of lied
- Requires special hardware, and the speed boost isn't incredible
- Also introduces some interesting security vulnerabilities
  - See the SPECTRE and Meltdown exploits of recent panic

Test

- Test

```
int main() {
        // Code goes here
}
```

Hello, there!

Hello, there!