The History of Concurrency Support in Java

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

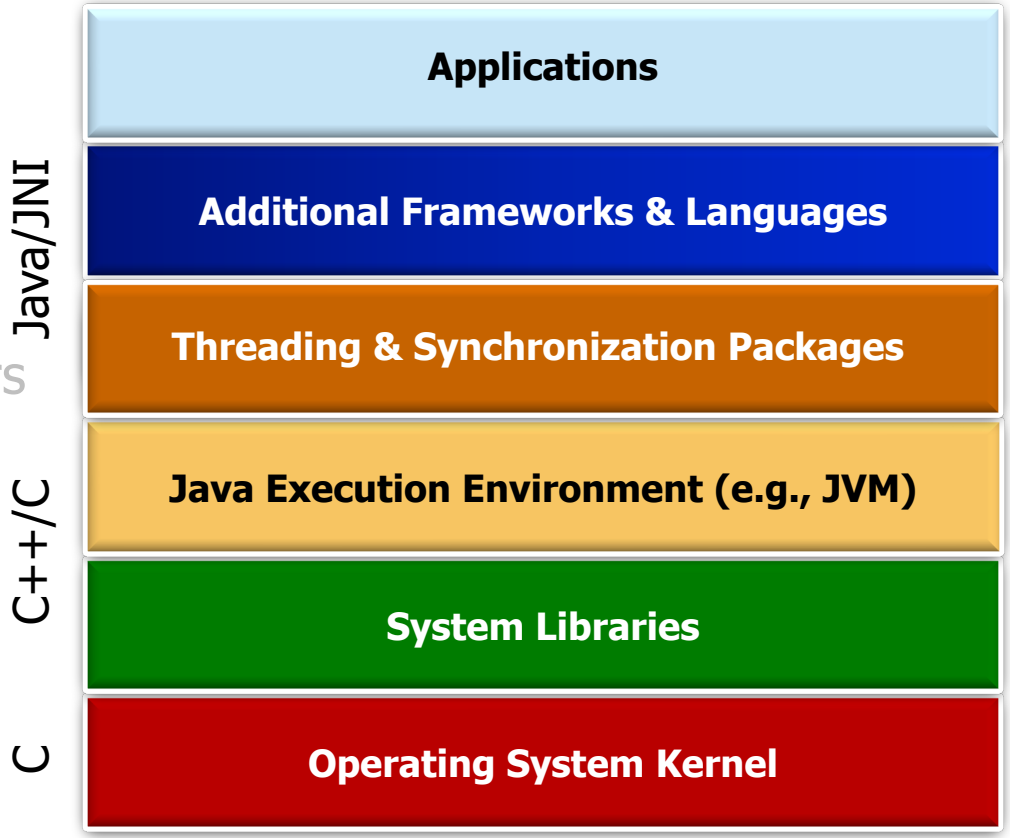
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

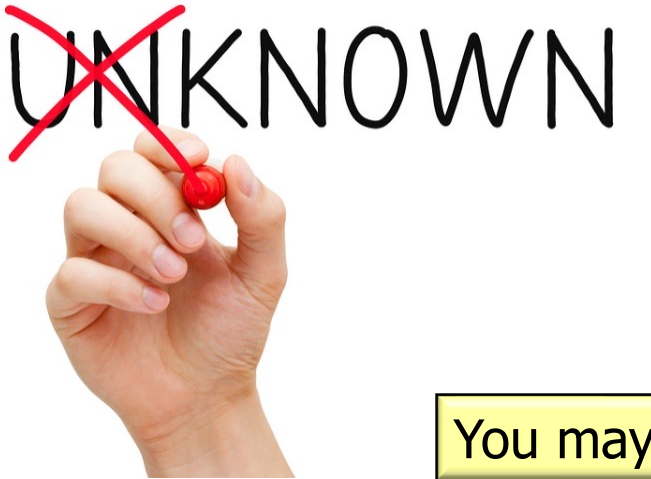
Learning Objectives in this Part of the Lesson

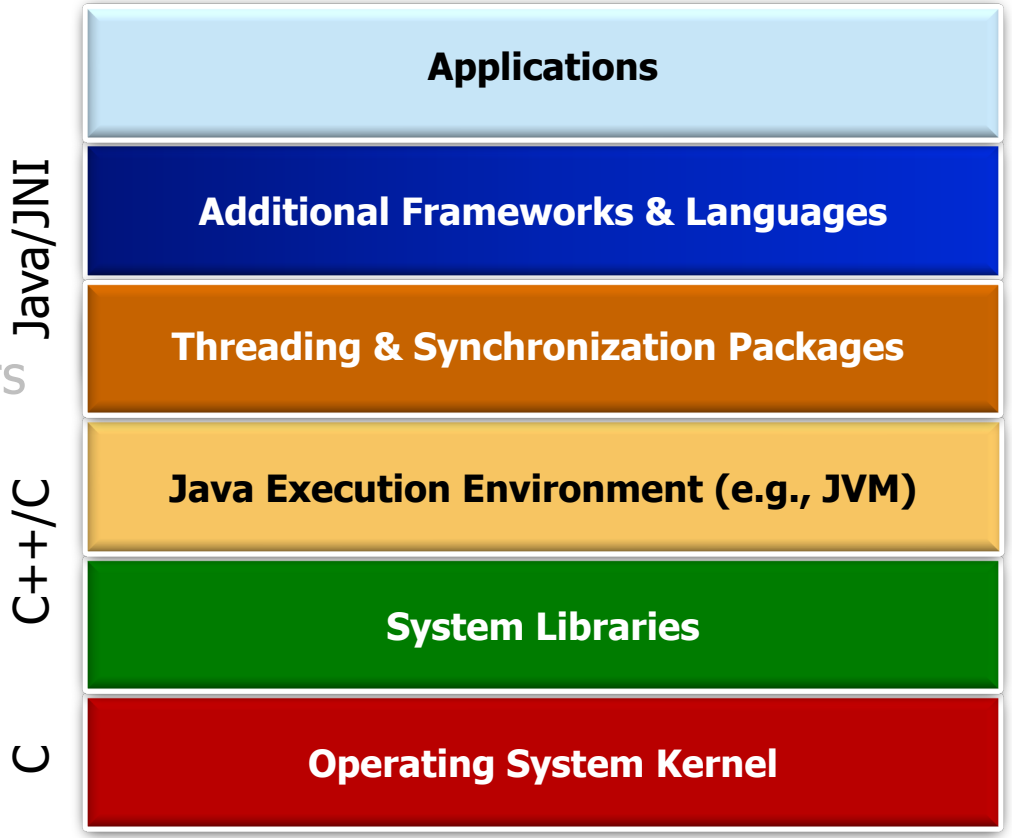
- Understand the meaning of key concurrent programming concepts
- Recognize how Java supports concurrent programming concepts
- Be aware of common concurrency hazards faced by Java programmers
- Learn Java concurrency history



Learning Objectives in this Part of the Lesson

- Understand the meaning of key concurrent programming concepts
- Recognize how Java supports concurrent programming concepts
- Be aware of common concurrency hazards faced by Java programmers
- Learn Java concurrency history



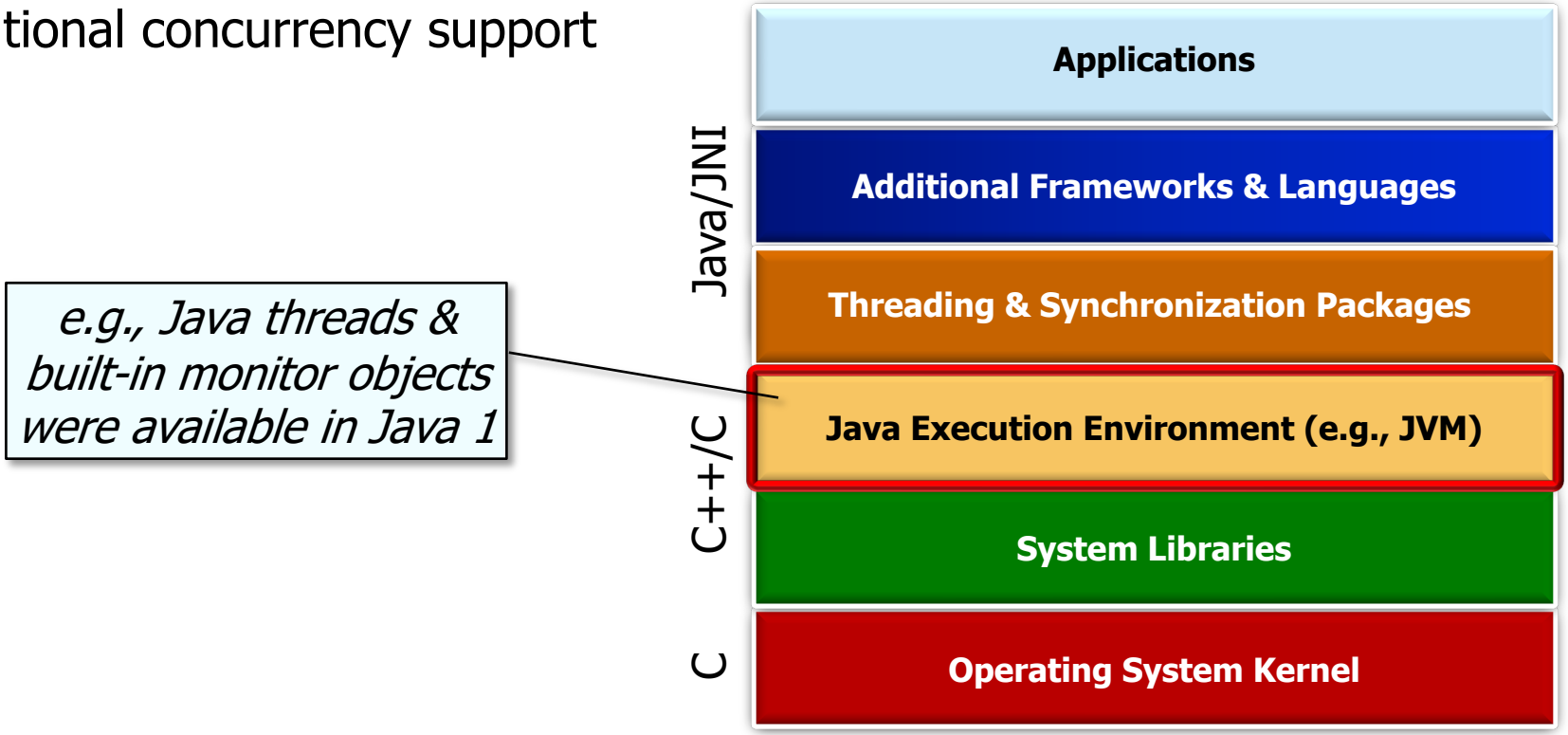


You may already know some of this!!!

A Brief History of Concurrency in Java

A Brief History of Concurrency in Java

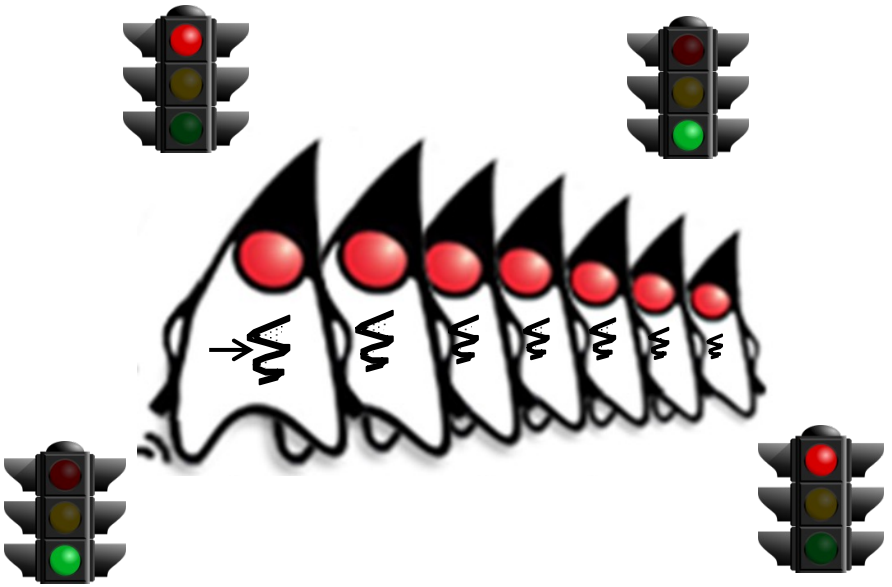
- Foundational concurrency support



See en.wikipedia.org/wiki/Java_version_history#JDK_1.0

A Brief History of Concurrency in Java

- Foundational concurrency support
  - Focus on basic multi-threading & synchronization primitives



See docs.oracle.com/javase/tutorial/essential/concurrency

A Brief History of Concurrency in Java

- Foundational concurrency support
  - Focus on basic multi-threading & synchronization primitives

Allow multiple threads to communicate & interact via a "bounded buffer"

```
SimpleBlockingBoundedQueue
<Integer> simpleQueue = new
    SimpleBlockingBoundedQueue<>();

Thread[] threads = new Thread[] {
  new Thread(new Producer<>
                   (simpleQueue)),
  new Thread(new Consumer<>
                   (simpleQueue))
};

for (Thread thread : threads)
  thread.start();

for (Thread thread : threads)
  thread.join();
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleBlockingQueue

A Brief History of Concurrency in Java

- Foundational concurrency support
  - Focus on basic multi-threading & synchronization primitives

Create two Thread objects that produce & consume messages via the bounded buffer

```
SimpleBlockingBoundedQueue
<Integer> simpleQueue = new
     SimpleBlockingBoundedQueue<>();

Thread[] threads = new Thread[] {
  new Thread(new Producer<>
                    (simpleQueue)),
  new Thread(new Consumer<>
                    (simpleQueue))
};

for (Thread thread : threads)
  thread.start();

for (Thread thread : threads)
  thread.join();
```

See docs.oracle.com/javase/8/docs/api/java/lang/Thread.html

A Brief History of Concurrency in Java

- Foundational concurrency support
  - Focus on basic multi-threading & synchronization primitives

```
SimpleBlockingBoundedQueue
<Integer> simpleQueue = new
     SimpleBlockingBoundedQueue<>();

Thread[] threads = new Thread[] {
  new Thread(new Producer<>
                    (simpleQueue)),
  new Thread(new Consumer<>
                    (simpleQueue))
};

for (Thread thread : threads)
  thread.start();

for (Thread thread : threads)
  thread.join();
```

Start the producer & consumer threads

See docs.oracle.com/javase/8/docs/api/java/lang/Thread.html#start

A Brief History of Concurrency in Java

- Foundational concurrency support
  - Focus on basic multi-threading & synchronization primitives

```
SimpleBlockingBoundedQueue
<Integer> simpleQueue = new
     SimpleBlockingBoundedQueue<>();

Thread[] threads = new Thread[] {
  new Thread(new Producer<>
                    (simpleQueue)),
  new Thread(new Consumer<>
                    (simpleQueue))
};

for (Thread thread : threads)
  thread.start();

for (Thread thread : threads)
  thread.join();
```

Barrier that waits for the producer & consumer threads to finish running

See docs.oracle.com/javase/8/docs/api/java/lang/Thread.html#join

A Brief History of Concurrency in Java

- Foundational concurrency support
  - Focus on basic multi-threading & synchronization primitives

```
class
SimpleBlockingBoundedQueue<E> {
  public E take() ...{
    synchronized(this) {
      while (mList.isEmpty())
        wait();

      notifyAll();

      return mList.poll();
    }
  }
```

Demonstrates Java's built-in monitor object mutual exclusion & coordination primitives

See github.com/douglascraigschmidt/LiveLessons/tree/master/SimpleBlockingQueue

A Brief History of Concurrency in Java

- Foundational concurrency support
  - Focus on basic multi-threading & synchronization primitives

Ensure mutually exclusive access to take()'s critical section via the intrinsic lock

```
class
SimpleBlockingBoundedQueue<E> {
  public E take() ...{
    synchronized(this) {
      while (mList.isEmpty())
        wait();

      notifyAll();

      return mList.poll();
    }
  }
```

See docs.oracle.com/javase/tutorial/essential/concurrency/locksync.html

A Brief History of Concurrency in Java

- Foundational concurrency support
  - Focus on basic multi-threading & synchronization primitives

Coordinate interactions between multiple producer & consumer threads

```
class
SimpleBlockingBoundedQueue<E> {
  public E take() ...{
    synchronized(this) {
      while (mList.isEmpty())
        wait();

      notifyAll();

      return mList.poll();
    }
  }
```

See docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html

A Brief History of Concurrency in Java

- Foundational concurrency support
  - Focus on basic multi-threading & synchronization primitives

```
class
SimpleBlockingBoundedQueue<E> {
  public E take() ...{
    synchronized(this) {
      while (mList.isEmpty())
        wait();

      notifyAll();

      return mList.poll();
    }
  }
```

The intrinsic lock is released after the next item on the list is removed/returned

See docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html#poll

A Brief History of Concurrency in Java

- Foundational concurrency support
  - Focus on basic multi-threading & synchronization primitives
  - Efficient, but low-level & very limited in capabilities

A Brief History of Concurrency in Java

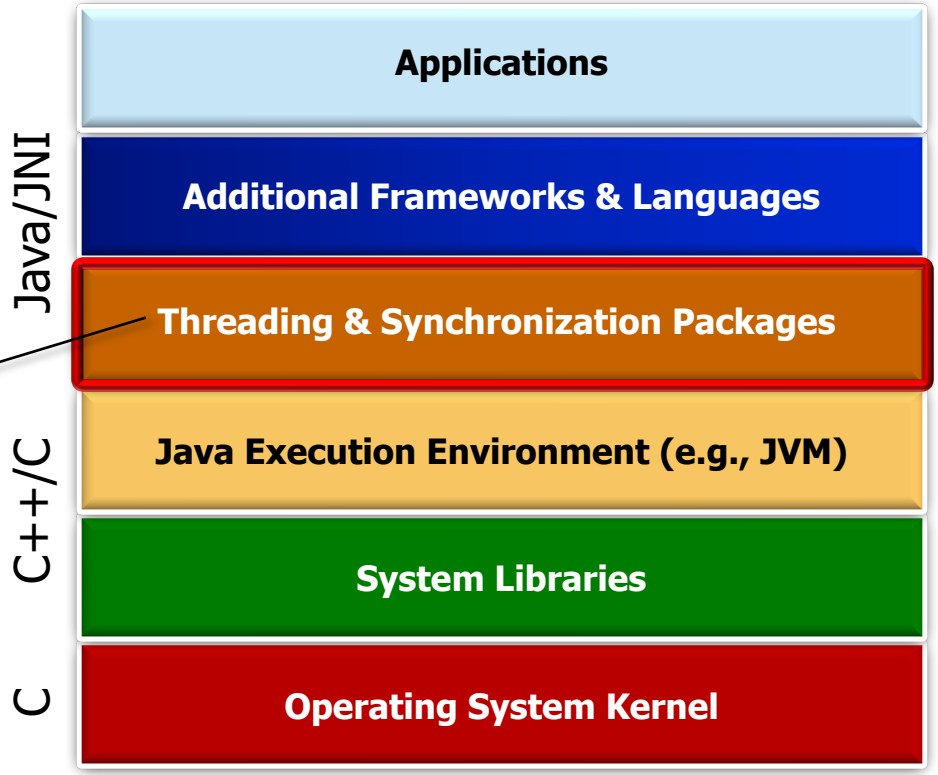
- Foundational concurrency support
  - Focus on basic multi-threading & synchronization primitives
  - Efficient, but low-level & very limited in capabilities
    - Many accidental complexities

Accidental complexities arise from limitations with software techniques, tools, & methods

See en.wikipedia.org/wiki/No_Silver_Bullet

A Brief History of Concurrency in Java

- Advanced concurrency support



e.g., Java executor framework, synchronizers, blocking queues, atomics, & concurrent collections all became available in Java 5+

See en.wikipedia.org/wiki/Java_version_history#J2SE_5.0

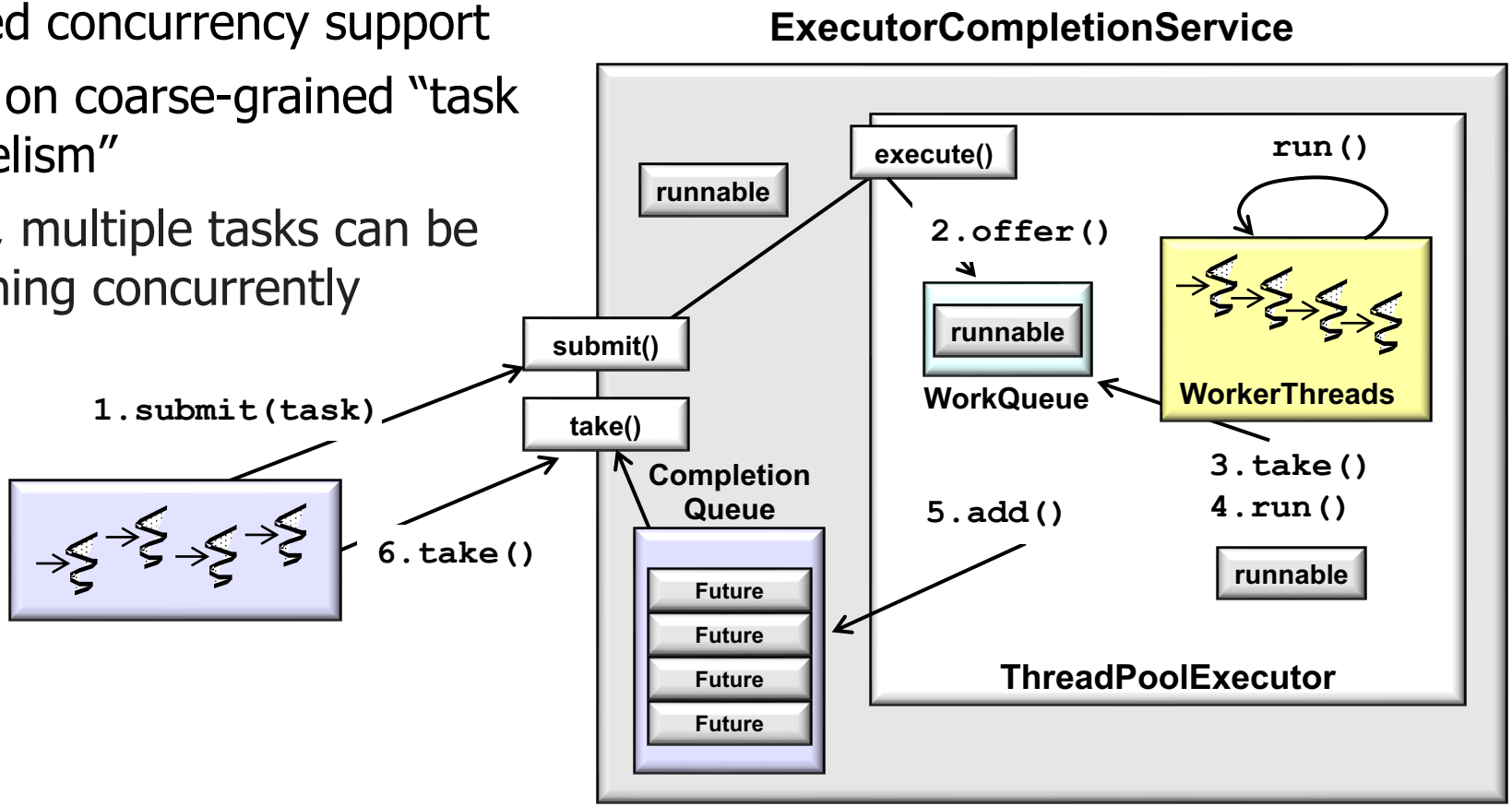
A Brief History of Concurrency in Java

- Advanced concurrency support
  - Focus on coarse-grained "task parallelism"

See en.wikipedia.org/wiki/Task_parallelism

A Brief History of Concurrency in Java

- Advanced concurrency support
  - Focus on coarse-grained "task parallelism"
    - e.g., multiple tasks can be running concurrently



The assumption then was there weren't many processor cores, e.g., 2 to 4

A Brief History of Concurrency in Java

- Advanced concurrency support
  - Focus on coarse-grained "task parallelism"
    - e.g., multiple tasks can be running concurrently

Create a fixed-sized thread pool & also coordinate the starting & stopping of multiple tasks that acquire/release shared resources

```
ExecutorService executor =
  Executors.newFixedThreadPool
    (numOfBeings,
     mThreadFactory);
...
CyclicBarrier entryBarrier =
  new CyclicBarrier(numOfBeings+1);

CountDownLatch exitBarrier =
  new CountDownLatch(numOfBeings);

for (int i=0; i < beingCount; ++i)
  executor.execute
    (makeBeingRunnable(i,
                       entryBarrier,
                       exitBarrier));
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/PalantiriManagerApplication

A Brief History of Concurrency in Java

- Advanced concurrency support
  - Focus on coarse-grained "task parallelism"
    - e.g., multiple tasks can be running concurrently

Creates a thread pool that reuses a fixed # of threads operating off of a shared unbounded queue

```
ExecutorService executor =
  Executors.newFixedThreadPool
    (numOfBeings,
     mThreadFactory);
...
CyclicBarrier entryBarrier =
  new CyclicBarrier(numOfBeings+1);

CountDownLatch exitBarrier =
  new CountDownLatch(numOfBeings);

for (int i=0; i < beingCount; ++i)
  executor.execute
    (makeBeingRunnable(i,
                       entryBarrier,
                       exitBarrier));
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html#newFixedThreadPool

A Brief History of Concurrency in Java

- Advanced concurrency support
  - Focus on coarse-grained "task parallelism"
    - e.g., multiple tasks can be running concurrently

A synchronizer that allows a set of threads to all wait for each other to reach a common barrier point

```
ExecutorService executor =
  Executors.newFixedThreadPool
    (numOfBeings,
     mThreadFactory);
...
CyclicBarrier entryBarrier =
  new CyclicBarrier(numOfBeings+1);

CountDownLatch exitBarrier =
  new CountDownLatch(numOfBeings);

for (int i=0; i < beingCount; ++i)
  executor.execute
    (makeBeingRunnable(i,
                       entryBarrier,
                       exitBarrier));
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CyclicBarrier.html

A Brief History of Concurrency in Java

- Advanced concurrency support
  - Focus on coarse-grained "task parallelism"
    - e.g., multiple tasks can be running concurrently

A synchronizer that allows one or more threads to wait until a set of operations being performed in other threads completes

```
ExecutorService executor =
  Executors.newFixedThreadPool
    (numOfBeings,
     mThreadFactory);
...
CyclicBarrier entryBarrier =
  new CyclicBarrier(numOfBeings+1);

CountDownLatch exitBarrier =
  new CountDownLatch(numOfBeings);

for (int i=0; i < beingCount; ++i)
  executor.execute
    (makeBeingRunnable(i,
                       entryBarrier,
                       exitBarrier));
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CountDownLatch.html

A Brief History of Concurrency in Java

- Advanced concurrency support
  - Focus on coarse-grained "task parallelism"
    - e.g., multiple tasks can be running concurrently

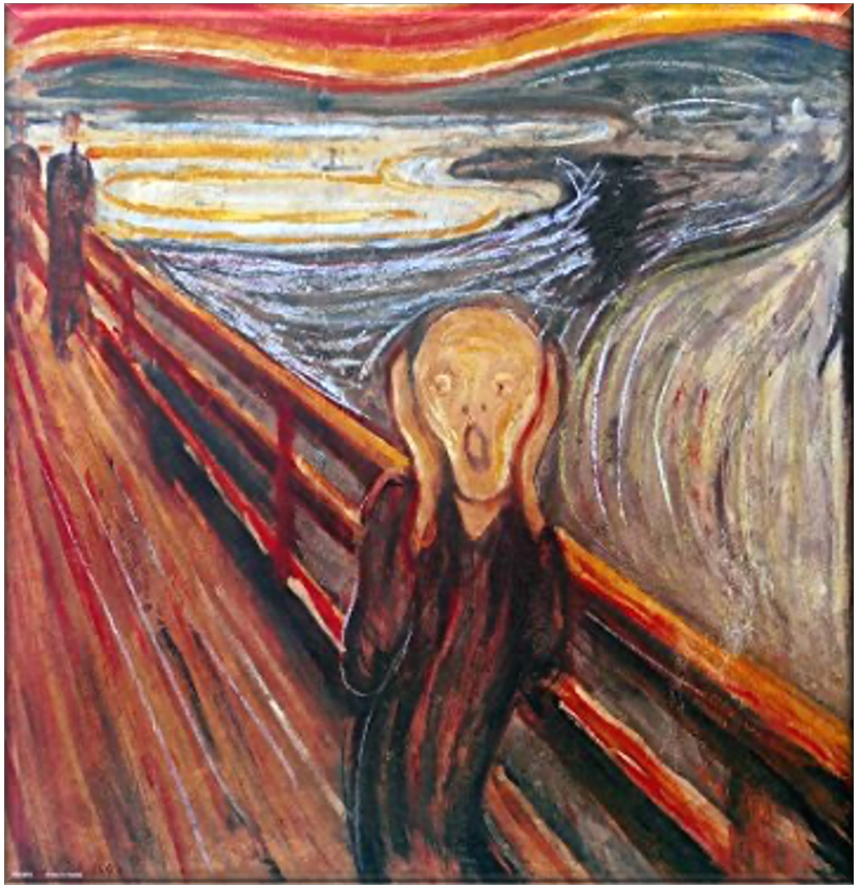
Executes the given command at some time in the future in the fixed-size pool of threads

```
ExecutorService executor =
  Executors.newFixedThreadPool
    (numOfBeings,
     mThreadFactory);
...
CyclicBarrier entryBarrier =
  new CyclicBarrier(numOfBeings+1);

CountDownLatch exitBarrier =
  new CountDownLatch(numOfBeings);

for (int i=0; i < beingCount; ++i)
  executor.execute
    (makeBeingRunnable(i,
                       entryBarrier,
                       exitBarrier));
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executor.html#execute

A Brief History of Concurrency in Java

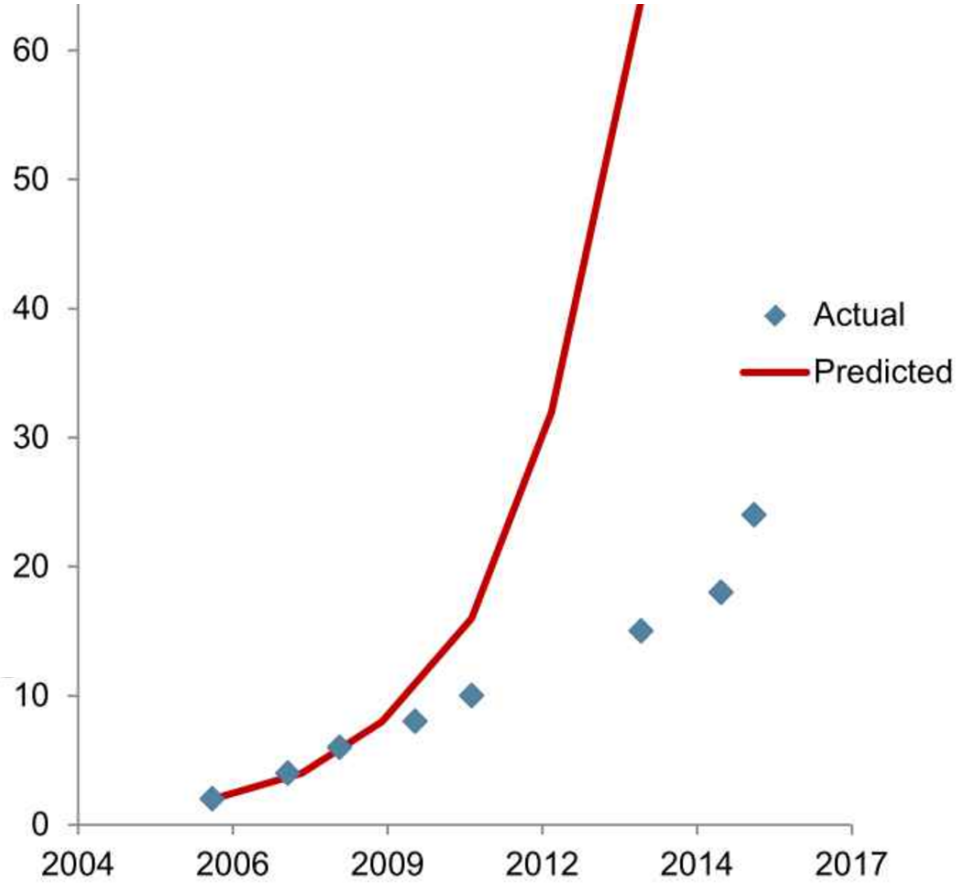
- Advanced concurrency support
  - Focus on coarse-grained "task parallelism"
  - Feature-rich & optimized, but also tedious & error-prone to program



See flylib.com/books/en/2.558.1/risks_of_threads.html

A Brief History of Concurrency in Java

- Advanced concurrency support
  - Focus on coarse-grained "task parallelism"
  - Feature-rich & optimized, but also tedious & error-prone to program
    - & scales poorly for modern multi-core processors



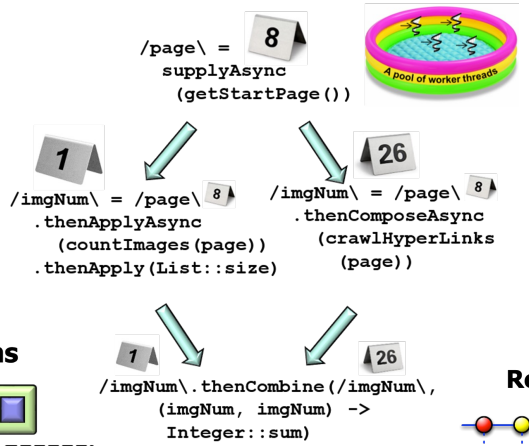
See www.infoq.com/presentations/parallel-java-se-8

A Brief History of Concurrency in Java

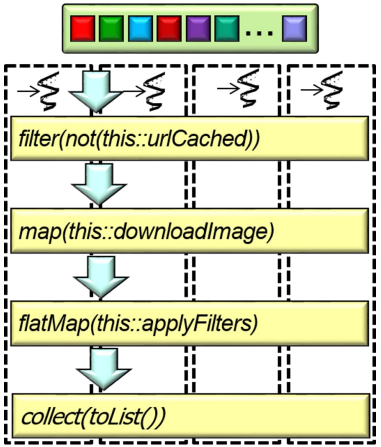
- Advanced concurrency support
  - Focus on coarse-grained "task parallelism"
  - Feature-rich & optimized, but also tedious & error-prone to program
    - & scales poorly for modern multi-core processors

Motivates Java's parallel, async, & reactive programming frameworks

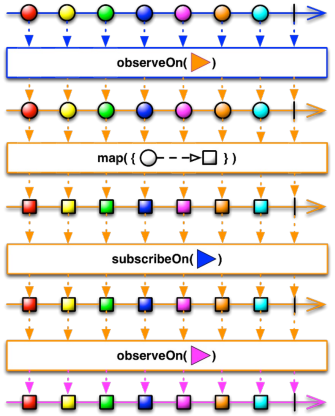
Completable Futures



Parallel Streams



Reactive Streams



See upcoming lesson on "*How Parallel Programs Are Developed in Java*"

End of the History of Concurrency Support in Java