

Java Platform Threads vs. Virtual Threads



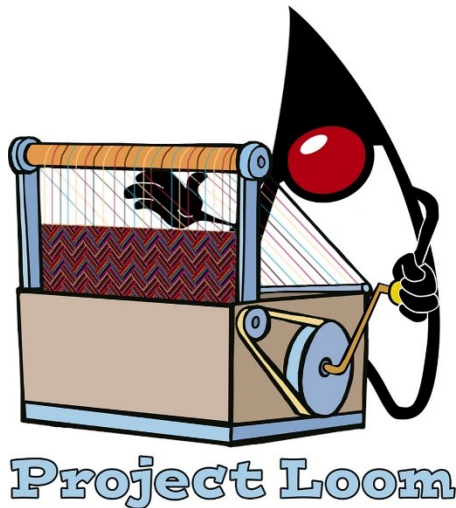
Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

**Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA**



Learning Objectives in this Part of the Lesson

- Understand how Java threads support concurrency
- Learn how our case study app works
- Know alternative ways of giving code to a thread
- Learn how to pass parameters to a Java thread
- Know the differences between Java platform & virtual threads



Platform threads

Thread supports the creation of *platform threads* that are typically mapped 1:1 to kernel threads scheduled by the operating system. Platform threads will usually have a large stack and other resources that are maintained by the operating system. Platform threads are suitable for executing all types of tasks but may be a limited resource.

Platform threads are designated *daemon* or *non-daemon* threads. When the Java virtual machine starts up, there is usually one non-daemon thread (the thread that typically calls the application's `main` method). The Java virtual machine terminates when all started non-daemon threads have terminated. Unstarted daemon threads do not prevent the Java virtual machine from terminating. The Java virtual machine can also be terminated by invoking the `Runtime.exit(int)` method, in which case it will terminate even if there are non-daemon threads still running.

In addition to the daemon status, platform threads have a *thread priority* and are members of a *thread group*.

Platform threads get an automatically generated thread name by default.

Virtual threads

Thread also supports the creation of *virtual threads*. Virtual threads are typically *user-mode threads* scheduled by the Java virtual machine rather than the operating system. Virtual threads will typically require few resources and a single Java virtual machine may support millions of virtual threads. Virtual threads are suitable for executing tasks that spend most of the time blocked, often waiting for I/O operations to complete. Virtual threads are not intended for long running CPU intensive operations.

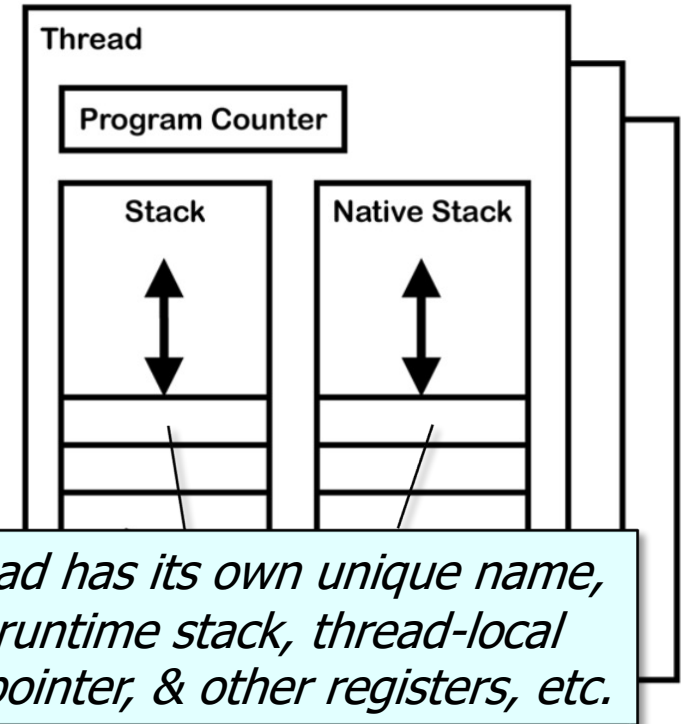
Virtual threads typically employ a small set of platform threads used as *carrier threads*. Locking and I/O operations are the *scheduling points* where a carrier thread is re-scheduled from one virtual thread to another. Code executing in a virtual thread will usually not be aware of the underlying carrier thread, and in particular, the `currentThread()` method, to obtain a reference to the *current thread*, will return the `Thread` object for the virtual thread, not the underlying carrier thread.

Virtual threads get a fixed name by default.

Java Platform Threads vs. Virtual Threads

Java Platform Threads vs. Virtual Threads

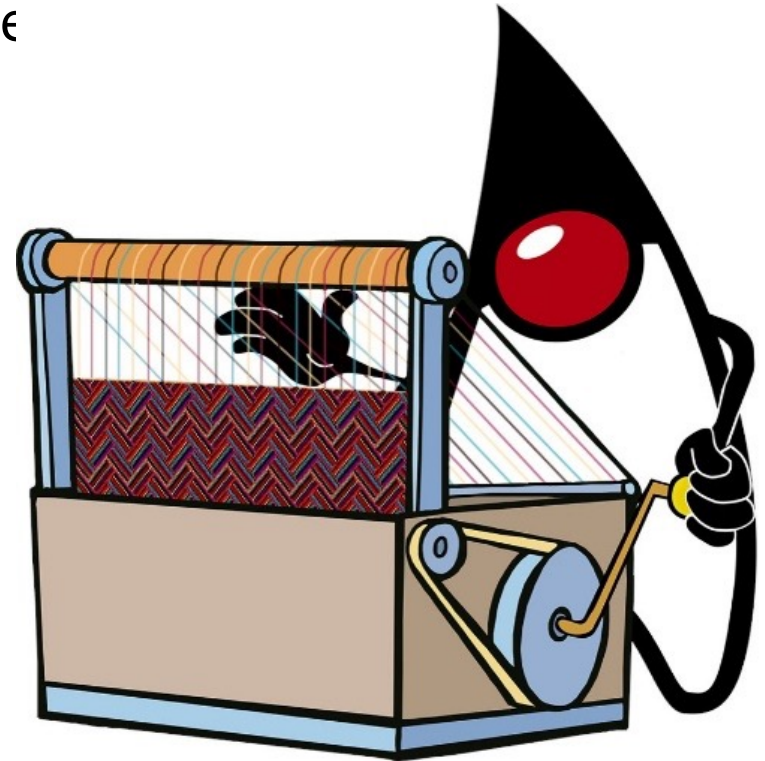
- A Java Thread has traditionally been an object containing various methods & fields that constitute its “state”



See blog.jamesdbloom.com/JVMInternals.html

Java Platform Threads vs. Virtual Threads

- A Java Thread has traditionally been an object containing various methods & fields that constitute its "state"
- Java 19 now refers to these types of Java threads as "platform threads"



Project Loom

See wiki.openjdk.java.net/display/loom/Main

Java Platform Threads vs. Virtual Threads

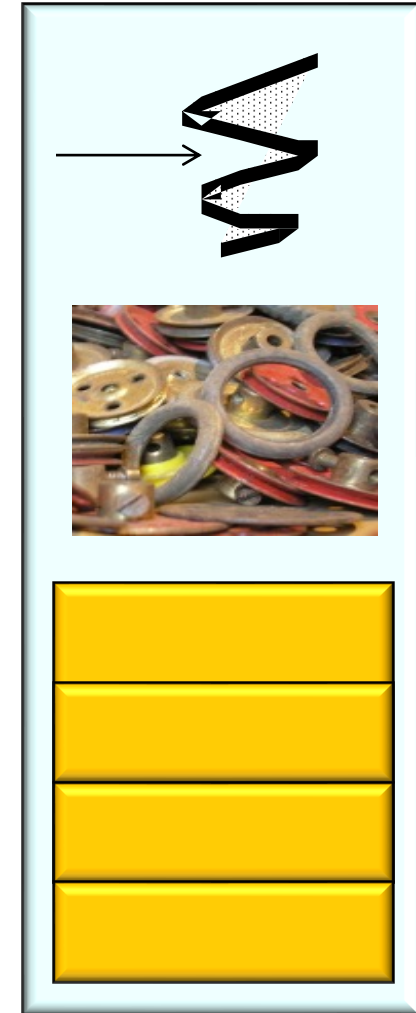
- Each Java platform thread is associated 1-to-1 with an OS kernel thread



See [en.wikipedia.org/wiki/Thread_\(computing\)#Kernel_threads](https://en.wikipedia.org/wiki/Thread_(computing)#Kernel_threads)

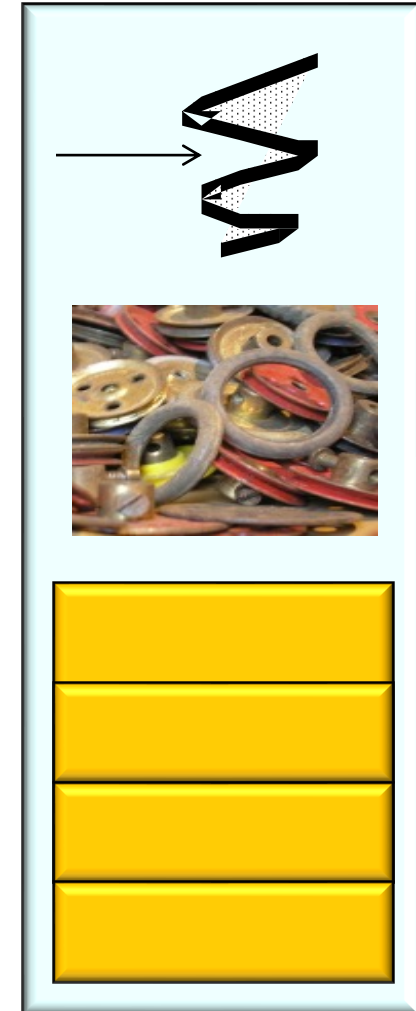
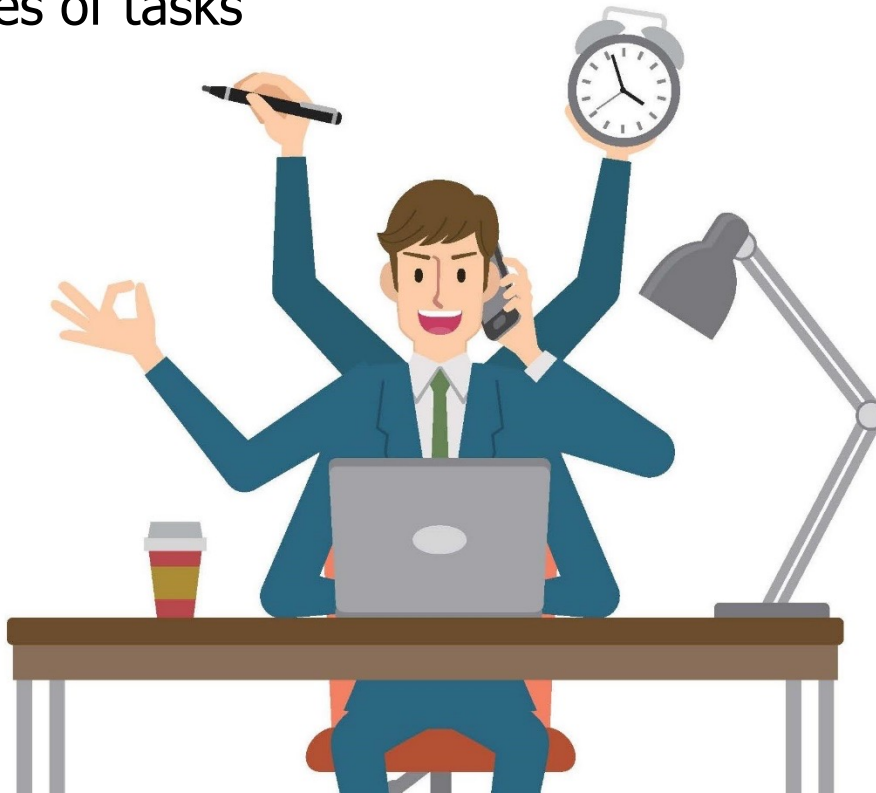
Java Platform Threads vs. Virtual Threads

- Each Java platform thread is associated 1-to-1 with an OS kernel thread
- It contains the same unique “state” as a traditional Java Thread object



Java Platform Threads vs. Virtual Threads

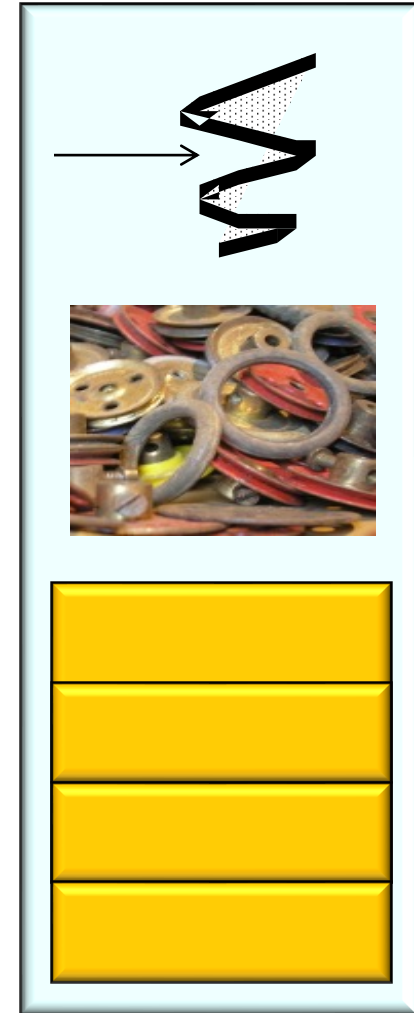
- Each Java platform thread is associated 1-to-1 with an OS kernel thread
 - It contains the same unique “state” as a traditional Java Thread object
- Platforms threads are suitable for executing all types of tasks



Java Platform Threads vs. Virtual Threads

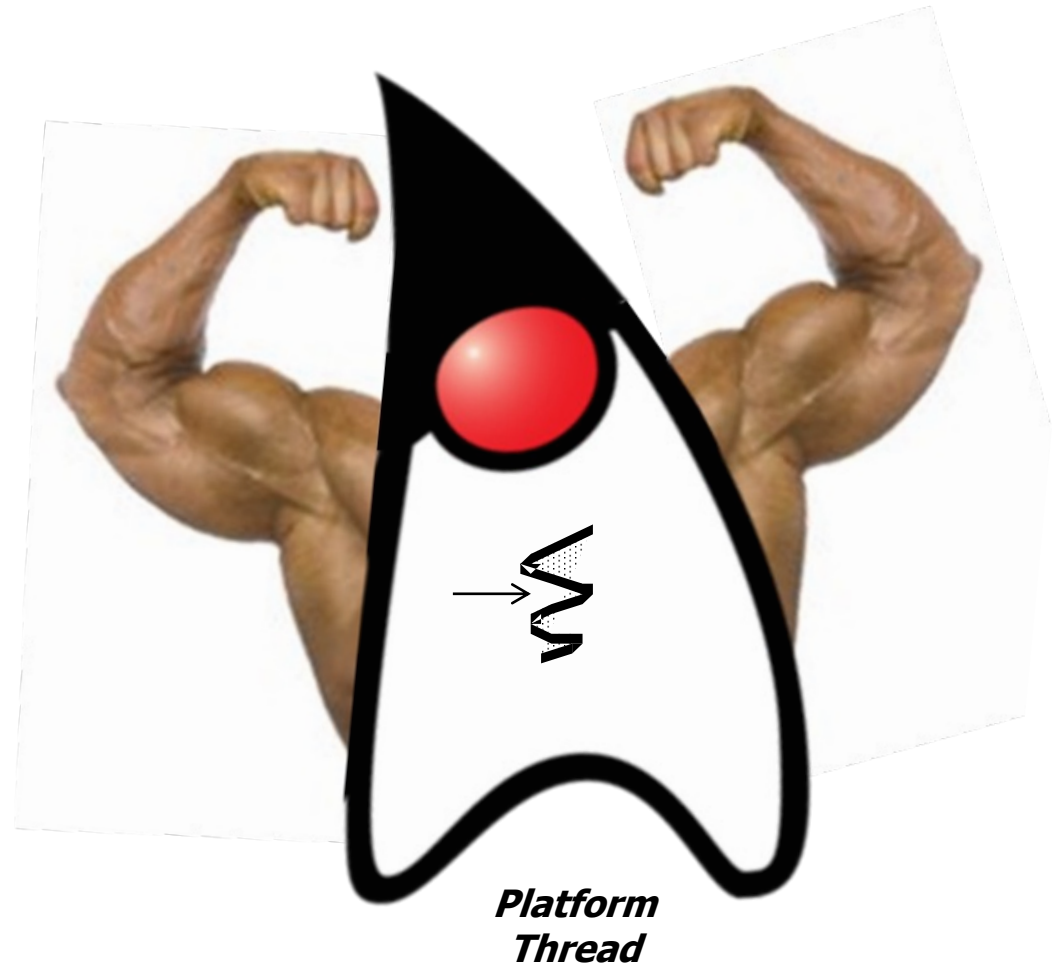
- Each Java platform thread is associated 1-to-1 with an OS kernel thread
 - It contains the same unique “state” as a traditional Java Thread object
- Platform threads are suitable for executing all types of tasks
 - However, they are a limited resource due to their non-trivial runtime stack size

LIMITED



Java Platform Threads vs. Virtual Threads

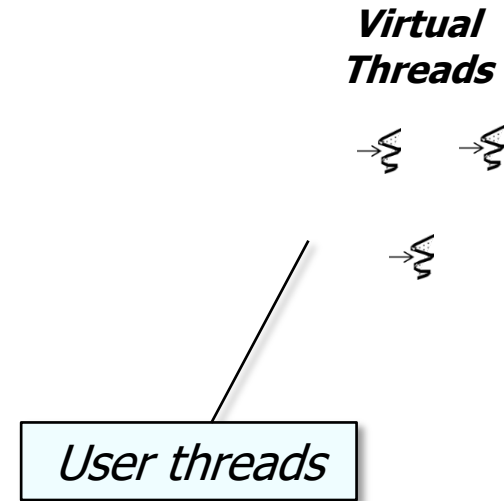
- In contrast, each Java virtual thread is a “lightweight” concurrency object



See www.infoq.com/articles/java-virtual-threads

Java Platform Threads vs. Virtual Threads

- In contrast, each Java virtual thread is a “lightweight” concurrency object
 - It is a user thread rather than a kernel thread

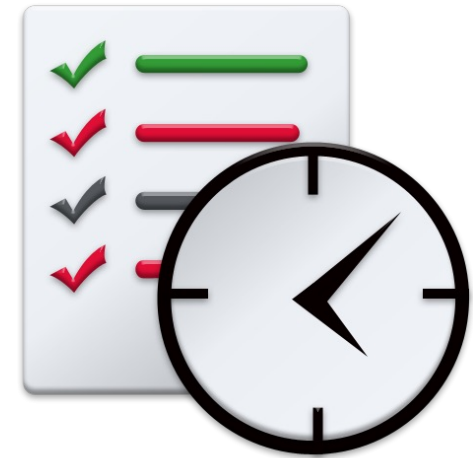
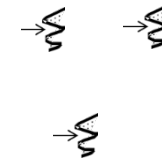


See [en.wikipedia.org/wiki/Thread_\(computing\)#User_threads](https://en.wikipedia.org/wiki/Thread_(computing)#User_threads)

Java Platform Threads vs. Virtual Threads

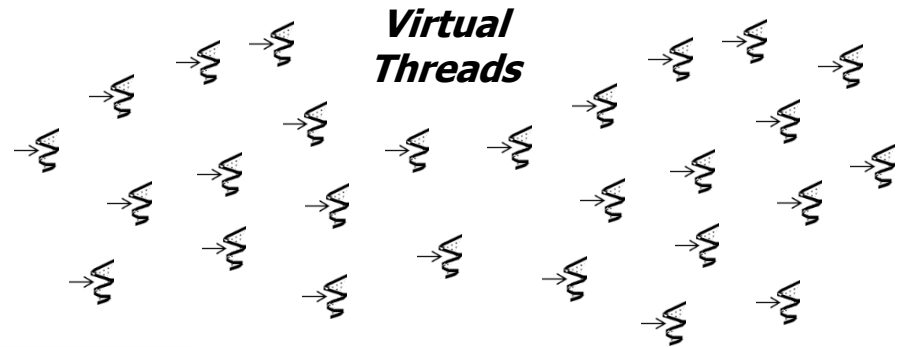
- In contrast, each Java virtual thread is a “lightweight” concurrency object
 - It is a user thread rather than a kernel thread
 - It is scheduled by the Java execution environment rather than the underlying OS

***Virtual
Threads***



Java Platform Threads vs. Virtual Threads

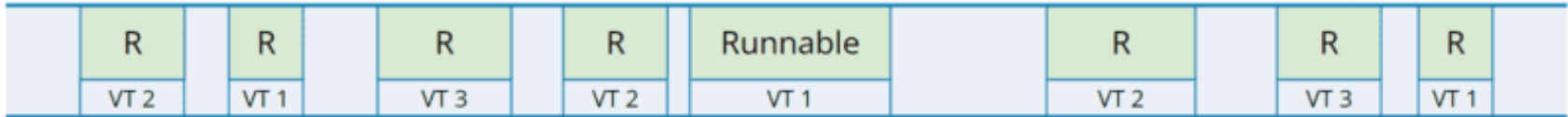
- In contrast, each Java virtual thread is a “lightweight” concurrency object
 - It is a user thread rather than a kernel thread
 - It is scheduled by the Java execution environment rather than the underlying OS
- A very large # of virtual threads can therefore be created



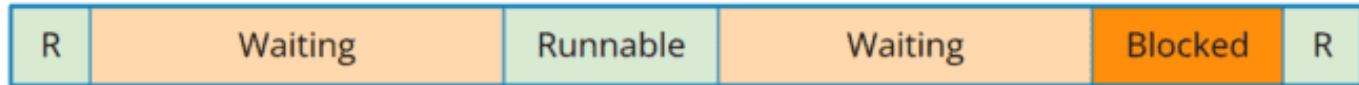
Java Platform Threads vs. Virtual Threads

- In contrast, each Java virtual thread is a “lightweight” concurrency object
 - It is a user thread rather than a kernel thread
- Virtual threads are multiplexed atop a pool of “carrier” threads

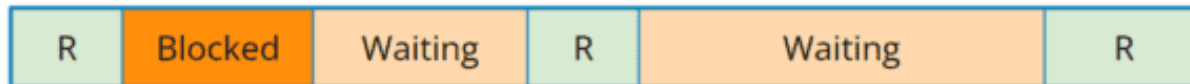
Carrier thread:



Virtual thread 1:



Virtual thread 2:



Virtual thread 3:

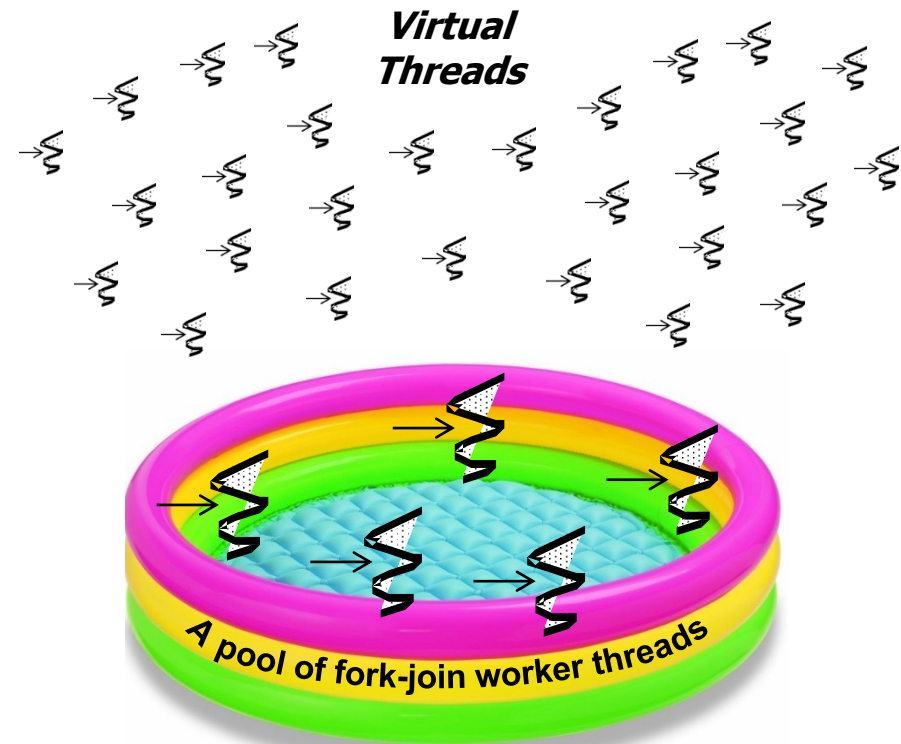


Blocking operations no longer block the executing thread, which enables the processing of a large # of requests in parallel with a small pool of carrier threads

See www.happycoders.eu/java/virtual-threads

Java Platform Threads vs. Virtual Threads

- In contrast, each Java virtual thread is a “lightweight” concurrency object
 - It is a user thread rather than a kernel thread
- Virtual threads are multiplexed atop a pool of “carrier” threads
 - The Java fork-join framework is currently used to implement the “carrier” threads

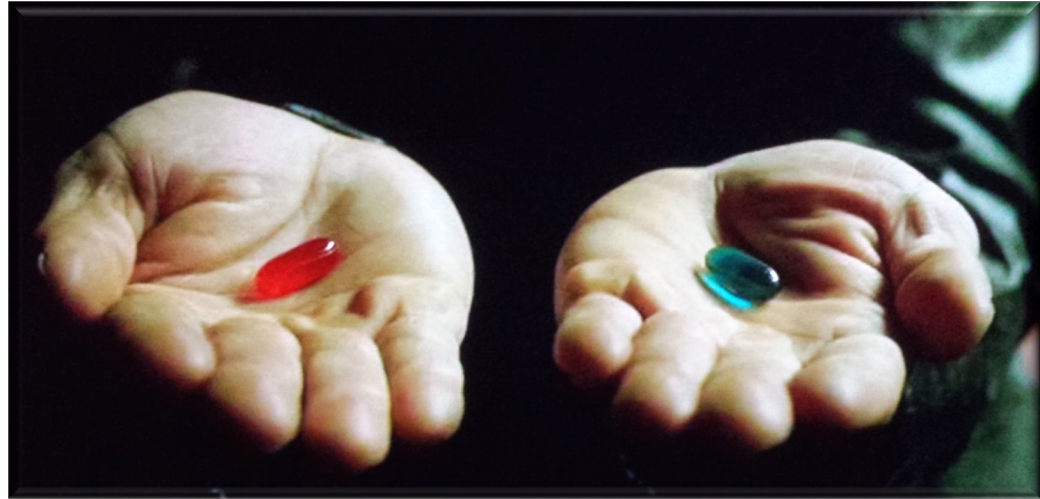


See theboreddev.com/understanding-java-virtual-threads

Creating Java Platform Threads vs. Virtual Threads

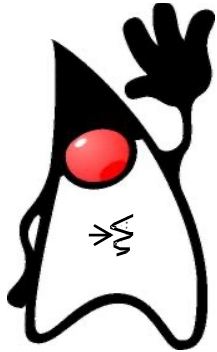
Creating Java Platform Threads vs. Virtual Threads

- Java platform threads can be created in two different ways



Creating Java Platform Threads vs. Virtual Threads

- Java platform threads can be created in two different ways
 - The traditional way



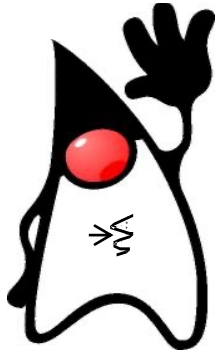
```
public class GCDThread
    extends Thread {
    public void run() {
        // code to run goes here
    }
}
```

```
Thread gcdThread = new GCDThread();
gcdThread.start();
```

*Create & start a thread using
GCDThread, which is a
named subclass of Thread*

Creating Java Platform Threads vs. Virtual Threads

- Java platform threads can be created in two different ways
 - The traditional way



Pass runnable to a new Thread object & start it

```
public class GCDThread
    extends Thread {
    public void run() {
        // code to run goes here
    }
}
```

```
Thread gcdThread = new GCDThread();
gcdThread.start();
```

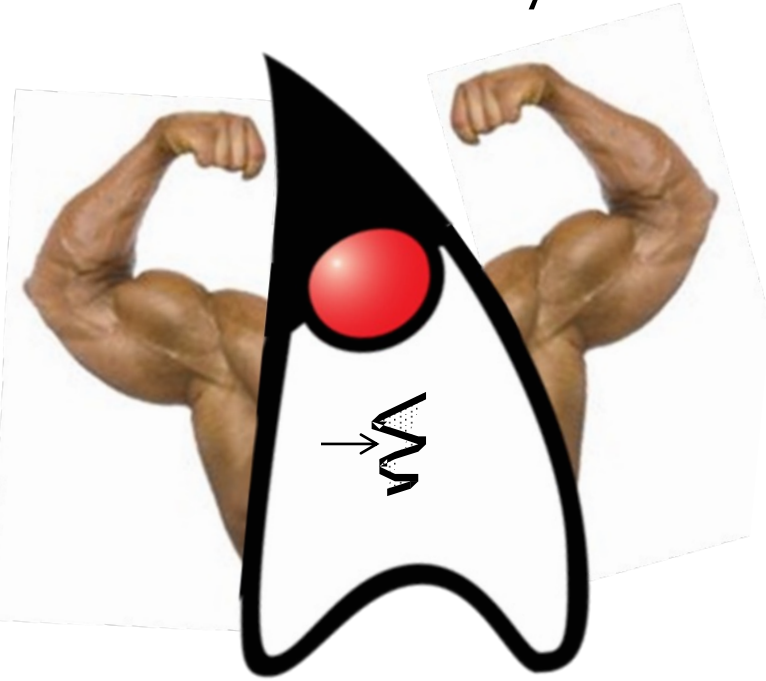
```
public class GCDRunnable
    implements Runnable {
    public void run() {
        // code to run goes here
    }
}
```

```
Runnable gcdRunnable =
    new GCDRunnable();
new Thread(gcdRunnable).start();
```

See [en.wikipedia.org/wiki/Thread_\(computing\)#User_threads](https://en.wikipedia.org/wiki/Thread_(computing)#User_threads)

Creating Java Platform Threads vs. Virtual Threads

- Java platform threads can be created in two different ways
 - The traditional way



```
public class GCDThread
    extends Thread {
    public void run() {
        // code to run goes here
    }
}
```

```
Thread gcdThread = new GCDThread();
gcdThread.start();
```

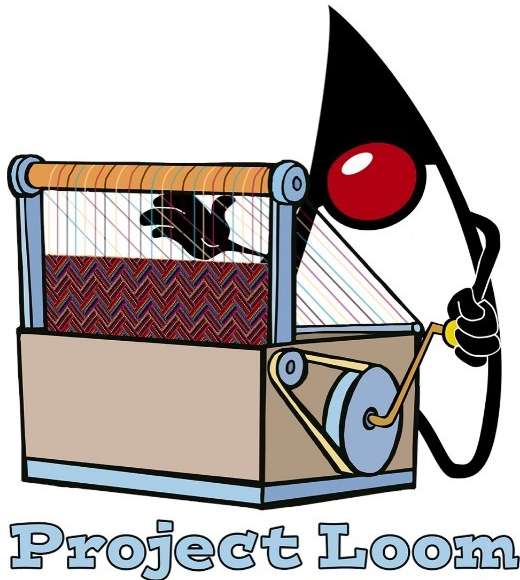
```
public class GCDRunnable
    implements Runnable {
    public void run() {
        // code to run goes here
    }
}
```

```
Runnable gcdRunnable =
    new GCDRunnable();
new Thread(gcdRunnable).start();
```

Java threads are relatively "heavyweight"

Creating Java Platform Threads vs. Virtual Threads

- Java platform threads can be created in two different ways
 - The traditional way
 - The Java 19 way



```
public class GCDRunnable  
    implements Runnable {  
    public void run() {  
        // code to run goes here  
    }  
}
```

```
Runnable gcdRunnable =  
    new GCDRunnable();
```

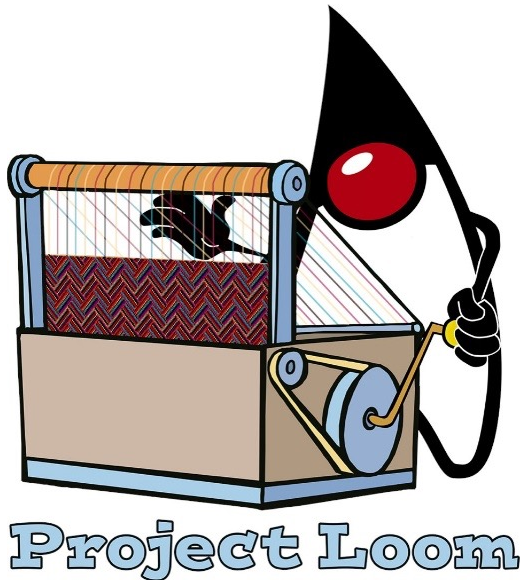
```
new Thread(gcdRunnable).start();
```

A familiar way to create & start a Java platform thread so it executes gcdRunnable

By default, a traditional Java Thread *is* a platform thread!

Creating Java Platform Threads vs. Virtual Threads

- Java platform threads can be created in two different ways
 - The traditional way
 - The Java 19 way



```
public class GCDRunnable  
    implements Runnable {  
    public void run() {  
        // code to run goes here  
    }  
}
```

```
Runnable gcdRunnable =  
    new GCDRunnable();
```

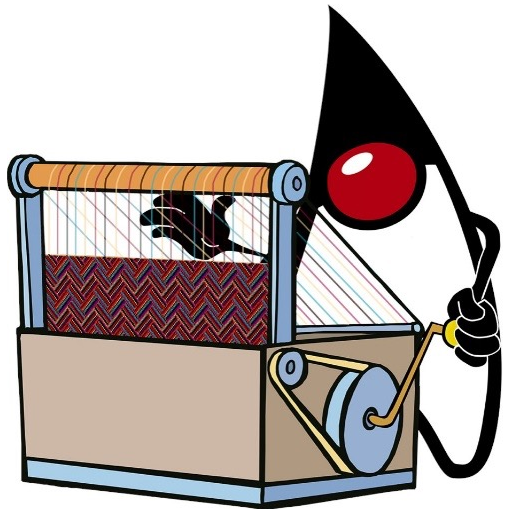
```
Thread.ofPlatform()  
    .start(gcdRunnable);
```

A more flexible way to create & start a platform thread so it executes gcdRunnable

See [docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Thread.html#ofPlatform\(\)](https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Thread.html#ofPlatform())

Creating Java Platform Threads vs. Virtual Threads

- Java platform threads can be created in two different ways
 - The traditional way
 - The Java 19 way



Project Loom

```
public class GCDRunnable
    implements Runnable {
    public void run() {
        // code to run goes here
    }
}
```

```
Runnable gcdRunnable =
    new GCDRunnable();
```

```
Thread thread = Thread
    .ofPlatform()
    .unstarted(gcdRunnable);

...
thread.start();
```

Create an "unstarted" platform thread & then start it so it executes gcdRunnable

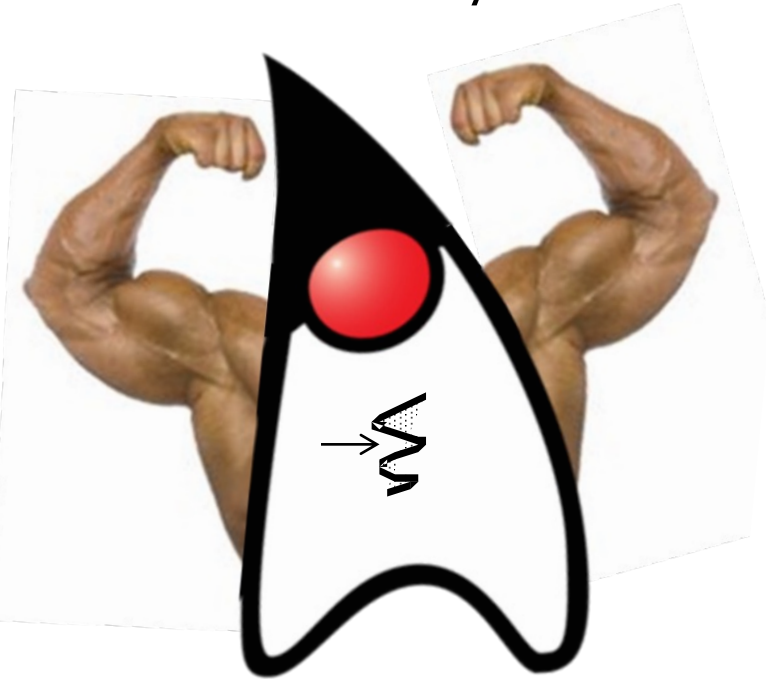
Creating Java Platform Threads vs. Virtual Threads

- Java platform threads can be created in two different ways
 - The traditional way
 - The Java 19 way

```
public class GCDRunnable  
    implements Runnable {  
    public void run() {  
        // code to run goes here  
    }  
}
```

```
Runnable gcdRunnable =  
    new GCDRunnable();
```

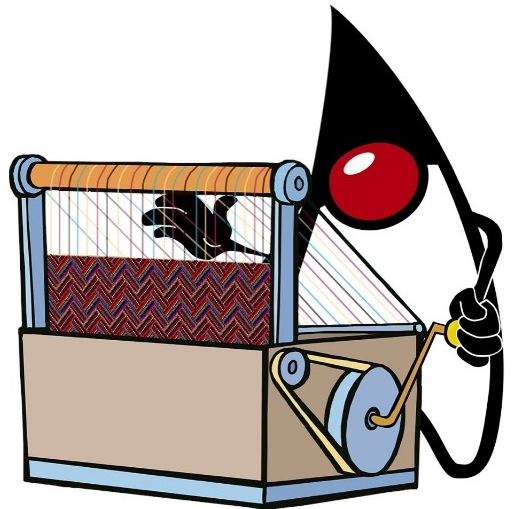
```
Thread thread = Thread  
    .ofPlatform()  
    .unstarted(gcdRunnable);  
...  
thread.start();
```



Java platform threads are also relatively “heavyweight”

Creating Java Platform Threads vs. Virtual Threads

- Java virtual threads can also be created in Java 19



Project Loom

```
public class GCDRunnable
    implements Runnable {
    public void run() {
        // code to run goes here
    }
}
```

```
Runnable gcdRunnable =
    new GCDRunnable();
```

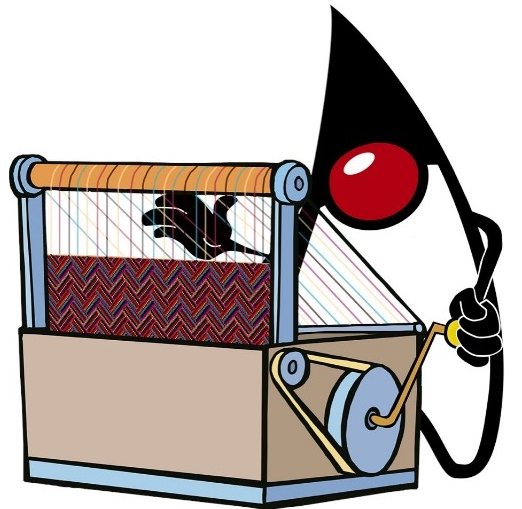
```
Thread.startVirtualThread
    (gcdRunnable);
```

A concise way to create & start a Java virtual thread so it executes gcdRunnable

See docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Thread.html#startVirtualThread

Creating Java Platform Threads vs. Virtual Threads

- Java virtual threads can also be created in Java 19



Project Loom

```
public class GCDSRunnable
    implements Runnable {
    public void run() {
        // code to run goes here
    }
}
```

```
Runnable gcdRunnable =
    new GCDSRunnable();
```

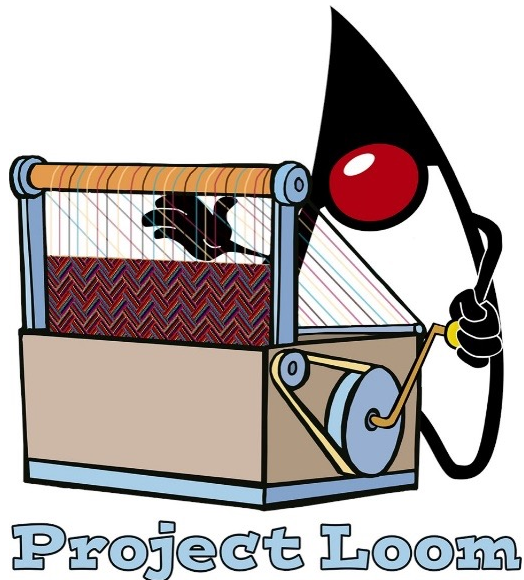
```
Thread.ofVirtual()
    .start(gcdRunnable);
```

A more flexible way to create & start a virtual thread so it executes gcdRunnable

See [docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Thread.html#ofVirtual\(\)](https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/lang/Thread.html#ofVirtual())

Creating Java Platform Threads vs. Virtual Threads

- Java virtual threads can also be created in Java 19



```
public class GCDRunnable  
    implements Runnable {  
    public void run() {  
        // code to run goes here  
    }  
}
```

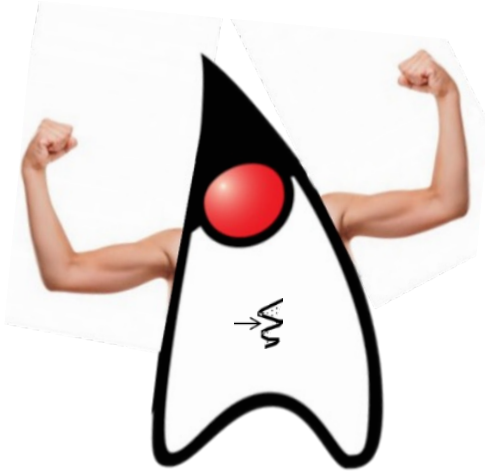
```
Runnable gcdRunnable =  
    new GCDRunnable();
```

```
Thread thread = Thread  
    .ofVirtual()  
    .unstarted(gcdRunnable);  
...  
thread.start();
```

Create an "unstarted" virtual thread & then start it so it executes gcdRunnable

Creating Java Platform Threads vs. Virtual Threads

- Java virtual threads can also be created in Java 19



```
public class GCDRunnable  
    implements Runnable {  
    public void run() {  
        // code to run goes here  
    }  
}
```

```
Runnable gcdRunnable =  
    new GCDRunnable();
```

```
Thread thread = Thread  
    .ofVirtual()  
    .unstarted(gcdRunnable);  
...  
thread.start();
```

Java virtual threads are relatively “lightweight”

End of Java Platform Threads vs. Virtual Threads