

The Latest and Greatest in Java

Features, Bells and Whistles from Java 8 and beyond...

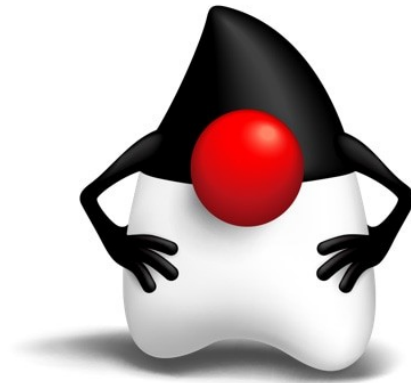
Who am I

- Johnny Egeland – Senior Developer
- Work at Storebrand – Doing the Taxes
- Have worked in Software Development for 21 years
- 10+ years of experience in C# and .Net
- ... and only 4+ years of experience in Java :-O



What will we look into

The most useful new Java features, introduced after Java 8... (and some future stuff as well)



Moving forward from Java 8

- What version of Java are you currently using?
- I assume Java 8 features is familiar stuff?
 - Lambda functions, Stream API, Optional, etc...
- XML processing and Old libs where migration pain points
 - All built in XML stuff is dropped after Java 8
 - Hard to find the correct replacements. But doable...
 - **Hint!** Jakarta and Glassfish has compatible API and Implementation
- Getting up to 11 was tedious... Moving on from there was simple.

New stuff Java 9 to 17 (current LTS)

- Interface default, static and private methods (8 - 9)
- Try-with-resources (9)
- Local Variable type inference (11)
- Files.readString and Files.writeString (11)
- Collectors.Teeing (12)
- Switch Expressions (14)
- Text Blocks (15)
- InstanceOf pattern matching, Records (16)
- Sealed Classes (17)



Beyond Java 17 ...

- Switch Pattern Matching
(Preview from 18)
- Project Loom
(Preview in Custom build of JDK 19)



Honorable mention...

- Project Jigsaw, aka. Java modules (9)



01. Interface default, static and private methods

- **default** and **static** methods introduced in Java 8
- Use of **private** keyword allowed since Java 9
- Place functionality related to the Interface, in the interface itself.
- Allows for code re-use without exposing internal details
- ... live demo ...

02. Try-with-resources

- Introduced in Java 9
- Automatic closing of opened resources
- Reduce verbosity and “ugly” code
- ... live demo ...

03. Local Variable Type inference

- Introduced in Java 11
- Reduced typing – less repetition of types
 - Useful for long class names or verbose generics
- Can only be used “locally” (for variables in methods)
- Frowned upon by some Java devs
 - Automatic -2 on reviews in Storebrand
- ... live heresy ...

04. Files.readString / .writeString

- Files API additions added in Java 11
- Old method was tedious and verbose
- Reduces code complexity for handling text IO
- ... live demo ...

05. Collectors.teeing()

- Introduced in Java 12
- Splits a stream and provides it to two Collectors
- ... and then you must **merge** the results at the end
- I believed I would use this a lot, but I really haven't
- ... live demo ...

06. Switch expressions

- Introduced in Java 14
- Simplifies switch expression that produce values
- Enables you to assign or return the result without a variable
- The basis for the upcoming switch pattern matching features
- ... live demo ...

07. Text blocks

- Introduced in Java 15
- Makes it much cleaner to embed multi-line text in Java code
- Removes the need for concatenation operators (+)
- Removes the need for newline chars in strings (\n)
- ... live demo ...
- AdHoc Bonus example: informative NullPointerException

08. InstanceOf pattern matching

- Introduced in Java 16
- Removes the need for type casts after type check
- Properties of objects can be used immediately
- ... live demo ...

09. Records

- Introduced in Java 16
- Create fully featured Data objects with very little code
- Are immutable and auto implements “getters”, equals() and hashCode()
- Provides default toString() that describes the contained data
- Can implement Interfaces, but not extend other classes
- Be careful with Records containing Mutable objects, Lists, Maps etc.
- ... live demo ...

10. Sealed Classes

- Introduced in Java 17
- Relates to **final** class; restrict extendability
- Specify who is allowed to extend or implement
- Sub-classes must be either Sealed or Final
- ... live demo ...

Next – To the future...

The following features
are Preview features
available from Java
18 and beyond



11. Extended switch expressions

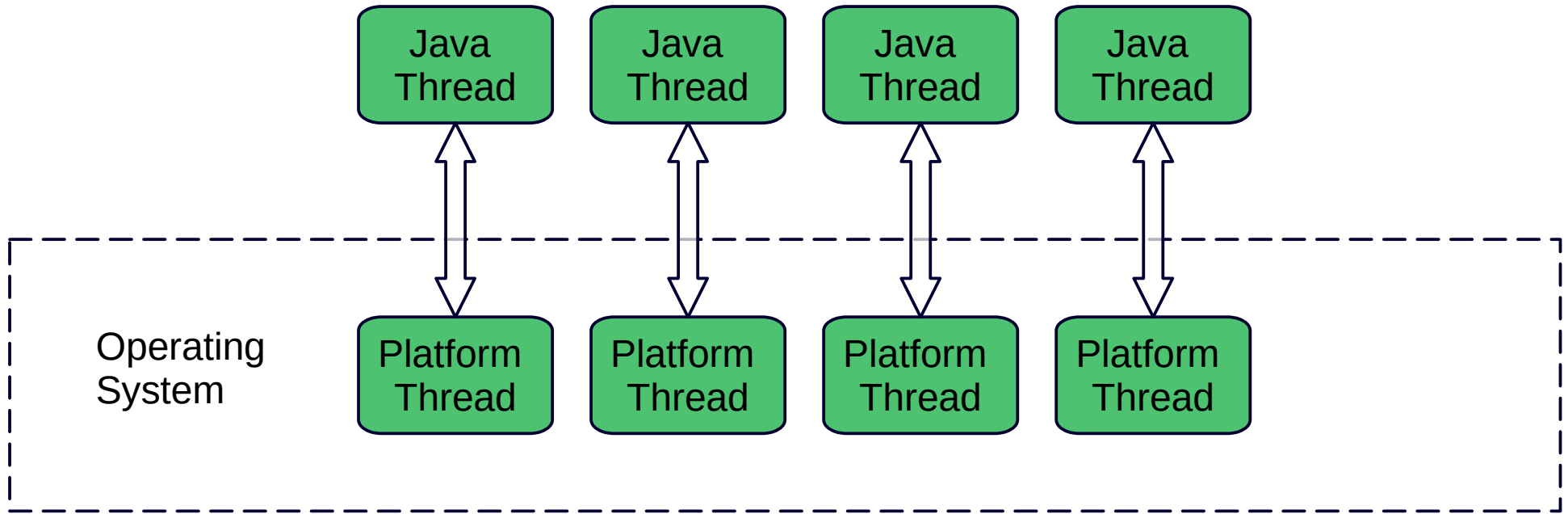
- In preview from Java 17
- But is in Third Preview in Java 19
- Makes it possible to match types in Switch
- Makes it possible to add expressions in Switch
- ... live demo ...

Project Loom

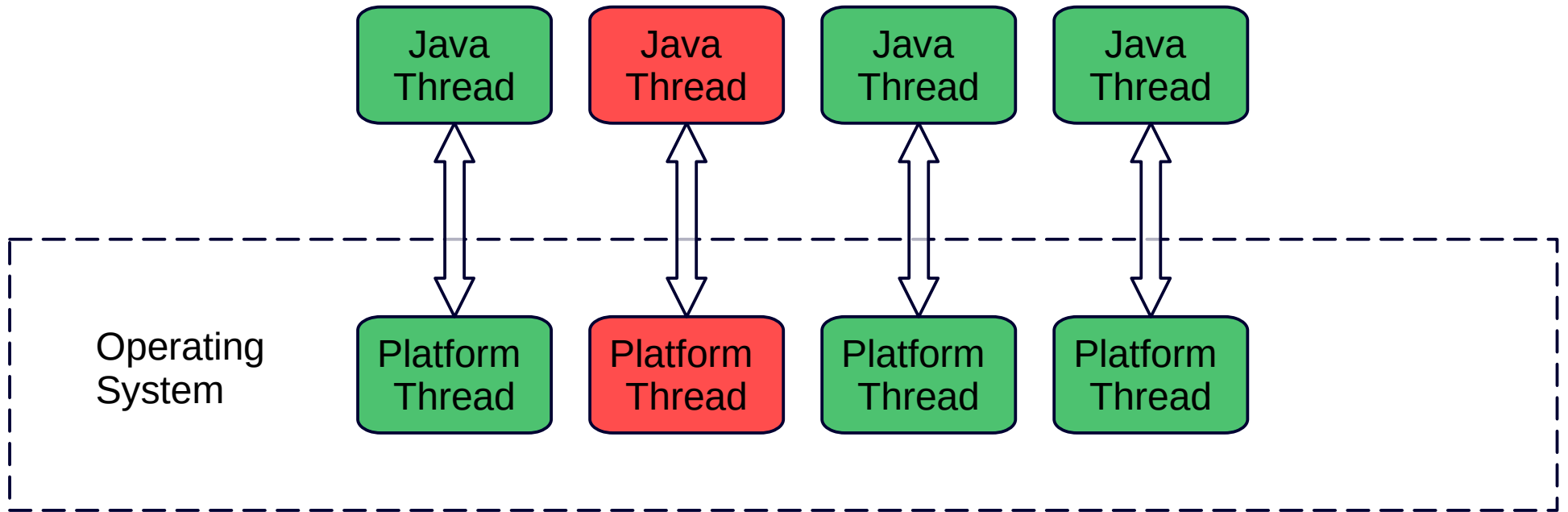


- Experimental release of Java 19
 - <https://jdk.java.net/loom/>
- In short: async/await (co-routines) for Java
- ... but much easier to use (transparent)
- ... and includes new concurrency features
- Avoid blocking of Platform threads, and instead re-purpose the thread for other tasks
- We will just scratch the surface...

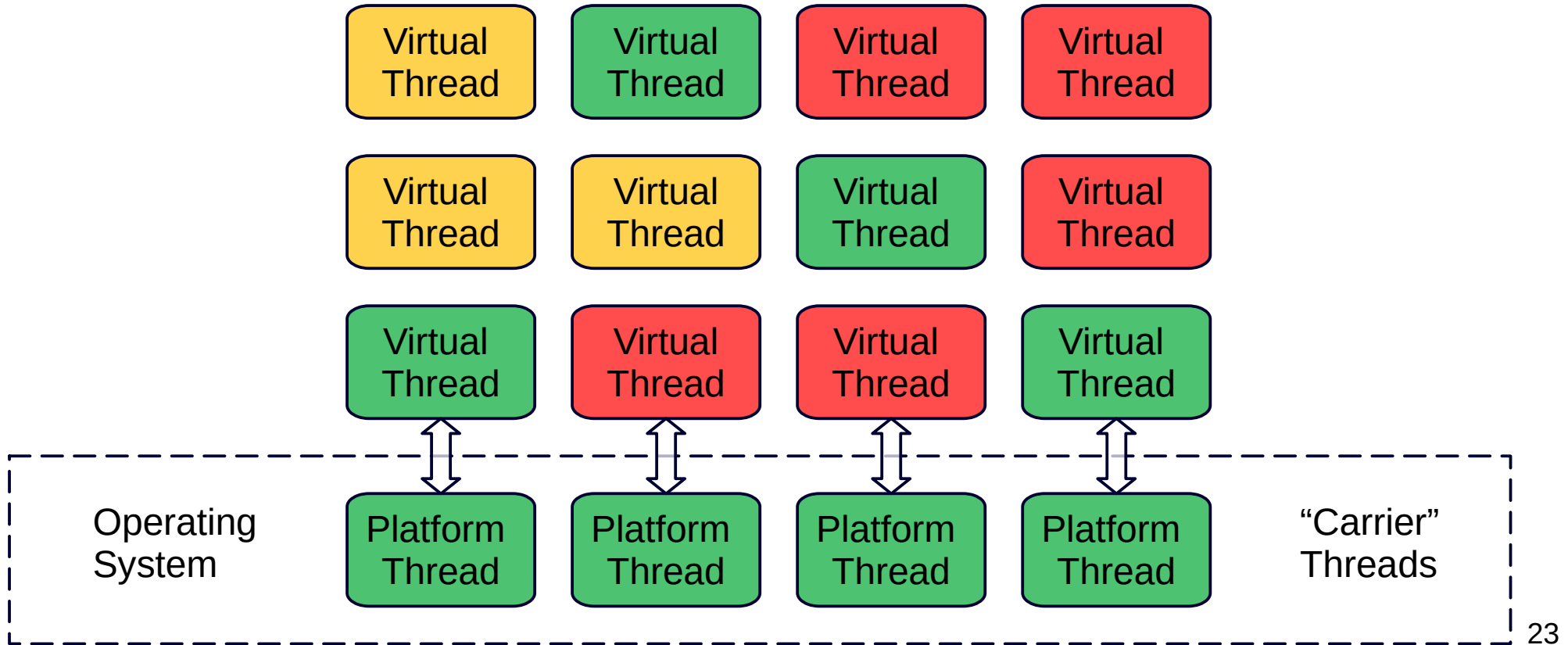
Traditional Java Thread model



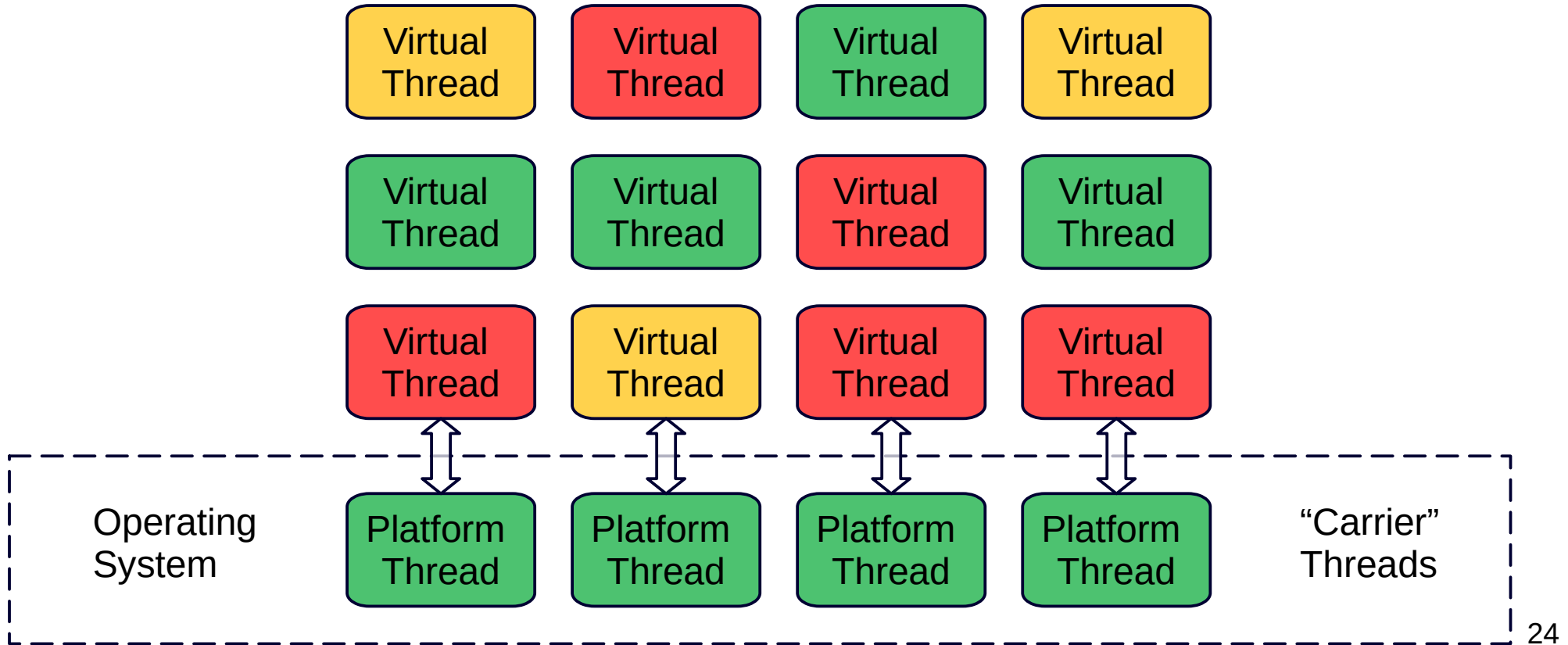
Traditional Java Thread model



Loom Virtual Threads

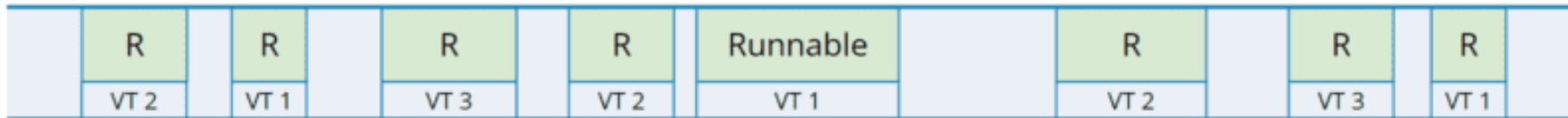


Loom Virtual Threads

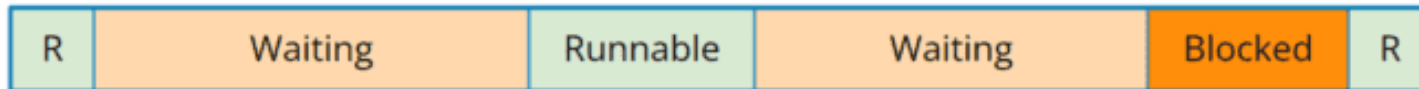


Loom Virtual Threads

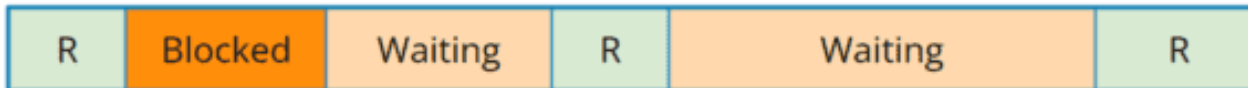
Carrier thread:



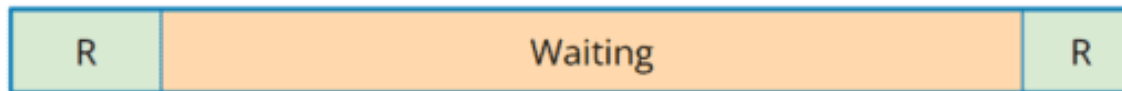
Virtual thread 1:



Virtual thread 2:



Virtual thread 3:



Loom 01 and 02: Virtual Threads

- Prevents Platform Threads from being blocked
- Light weight context switching at JVM level
- Java IO rewritten to take advantage automatically
- Go from Thousands of threads to Millions of threads
- ... let's scratch the surface ...

Loom 03: Async programming

- Sequential code is easy and readable. But...
- Blocking calls (I/O) blocks the thread down to OS
- Solvable using `CompletableFuture`. But...
- Async code is hard to write **and** to read. Bad stack traces.
- Virtual Threads makes sequential code behave as async.
 - No `async` / `await` keywords needed
 - Completely transparent (as long as API supports Virtual Threads)
- Sadly, hard to actually demonstrate effect of Virtual Thread blocking. But here is an example anyway...

Loom 03: Gotchas...

- There is no such thing as Magic!
- APIs must be “Virtual Thread Aware” and explicitly yield control instead of blocking
- Existing blocking code `_can_` still block OS threads, unless...
- ... they rely on `java.io` or other built in Java functionality, as...
- All built in Java APIs will be updated to be Virtual Thread aware (which should automatically make most libraries take advantage of Loom :-D)

Loom 04: Structured Concurrency

- Using `Thread.ofVirtual()` is low level and tedious
- New API for running Async Tasks
- Progression from the old Executor Service APIs
- Provide a simple way to solve async tasks that depends on output from other async tasks
- For instance a parent task can only complete once a set of sub-tasks are all complete
- ... one final example ...