

ORACLE

# Loom: Virtual Threads and StructuredConcurrency and ScopedValues in the JDK 20

**José Paumard**

Java Developer Advocate

Java Platform Group





<https://twitter.com/JosePaumard>



<https://github.com/JosePaumard>

<https://www.youtube.com/user/java>

<https://www.youtube.com/user/JPaumard>



<https://www.youtube.com/c/coursenlignejava>

<https://www.youtube.com/hashtag/jepcafe>



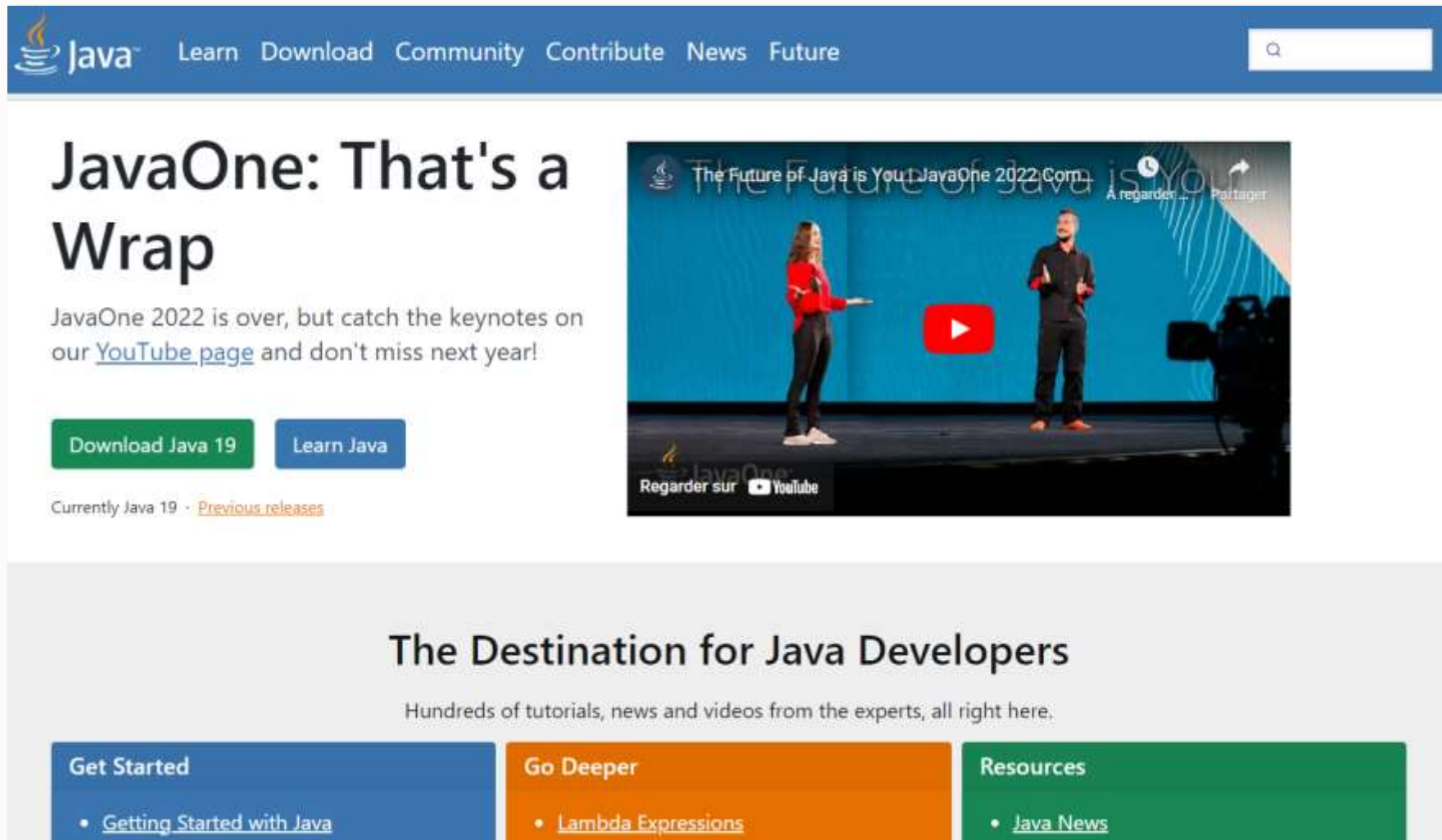
<https://fr.slideshare.net/jpaumard>



<https://www.pluralsight.com/authors/jose-paumard>



# <https://dev.java/>



The screenshot shows the Java developer website. At the top, there is a blue navigation bar with the Java logo and links for Learn, Download, Community, Contribute, News, and Future. A search bar is located on the right side of the navigation bar. The main content area features a large article titled "JavaOne: That's a Wrap" with a sub-headline "JavaOne 2022 is over, but catch the keynotes on our [YouTube page](#) and don't miss next year!". Below the article are two buttons: "Download Java 19" and "Learn Java". A video player is embedded on the right side of the article, showing a keynote presentation with two speakers on stage. The video player has a red play button in the center. Below the video player, there is a section titled "The Destination for Java Developers" with the subtitle "Hundreds of tutorials, news and videos from the experts, all right here.". This section contains three colored boxes: a blue box for "Get Started" with a link to "Getting Started with Java", an orange box for "Go Deeper" with a link to "Lambda Expressions", and a green box for "Resources" with a link to "Java News".

Java™ Learn Download Community Contribute News Future

## JavaOne: That's a Wrap

JavaOne 2022 is over, but catch the keynotes on our [YouTube page](#) and don't miss next year!

[Download Java 19](#) [Learn Java](#)

Currently Java 19 · [Previous releases](#)

The Future of Java is You | JavaOne 2022, Com... A regarder... Partager

Regarder sur YouTube

### The Destination for Java Developers

Hundreds of tutorials, news and videos from the experts, all right here.

- [Get Started](#)
  - [Getting Started with Java](#)
- [Go Deeper](#)
  - [Lambda Expressions](#)
- [Resources](#)
  - [Java News](#)

# Tune in!



Inside Java Newscast



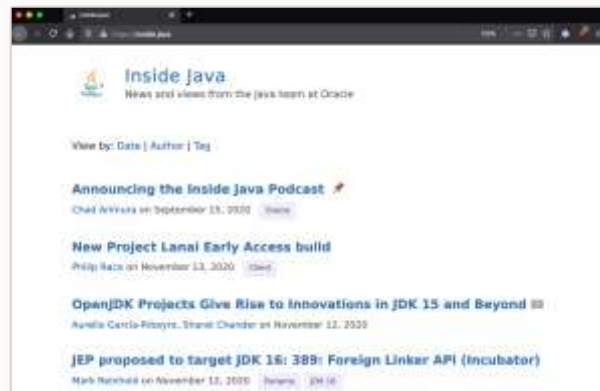
Dev.java



Inside Java Podcast



JEP Café



Inside.java



Sip of Java



# <https://dev.java/community/>



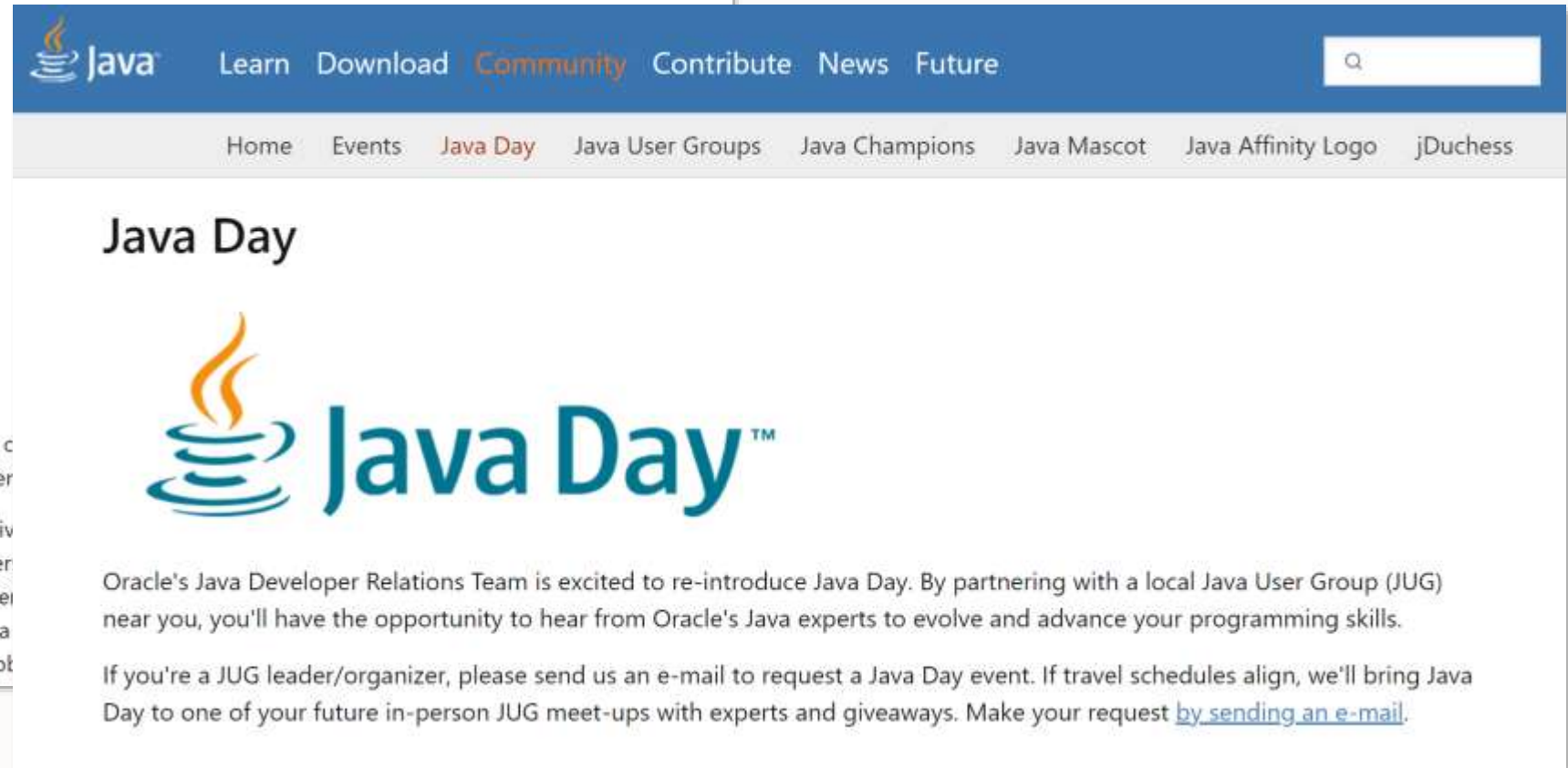
The screenshot shows the top navigation bar of the Java website. It features the Java logo on the left, followed by a horizontal menu with links for 'Learn', 'Download', 'Community' (highlighted in orange), 'Contribute', 'News', and 'Future'. A search box is located on the right side of the bar. Below the main navigation bar is a secondary menu with links for 'Home', 'Events', 'Java Day', 'Java User Groups' (highlighted in orange), 'Java Champions', 'Java Mascot', 'Java Affinity Logo', and 'jDuchess'.

## Java User Groups



Participating in a Java User Group is a great way to connect, c  
developer peers. JUGs can be found on almost every continer

Java User Groups (JUGs) are volunteer organizations that striv  
around the world. They provide a meeting place for Java user  
solutions, increase networking, expand Java Technology exper  
inclusive community. JUGs are the meeting point for the Java  
collaborate with developer peers. Explore the list of JUGs glot



This screenshot shows the 'Java Day' section of the Java website. It includes the same navigation bar as the previous screenshot, with 'Java Day' highlighted in the secondary menu. The main content area features the 'Java Day' title, the Java logo, and the 'Java Day™' logo. Below the logo is a paragraph of text: 'Oracle's Java Developer Relations Team is excited to re-introduce Java Day. By partnering with a local Java User Group (JUG) near you, you'll have the opportunity to hear from Oracle's Java experts to evolve and advance your programming skills.' This is followed by another paragraph: 'If you're a JUG leader/organizer, please send us an e-mail to request a Java Day event. If travel schedules align, we'll bring Java Day to one of your future in-person JUG meet-ups with experts and giveaways. Make your request [by sending an e-mail](#).'

# <https://inside.java/>



## Inside Java

News and views from members of the Java team at Oracle

Shows: [Podcast](#) | [Newscast](#) | [JEP Café](#) | [Sip Of Java](#)

[dev.java](#) | [About](#) | [Jobs](#)

Sort by: [Date](#) | [Author](#) | [Tag](#)

### [String Templates, JavaFX 19, and more at JavaOne](#)

Nicolai Parlog on August 23, 2022

[Amber](#)

[Client](#)

[Core Libraries](#)

### [airhacks.fm: Java 19 Millions of Threads in No Time](#)

Nicolai Parlog & Adam Bien (guest) on September 5, 2022

[JDK 19](#)

[Loom](#)

### [Job Opportunity: JavaFX Engineers](#)

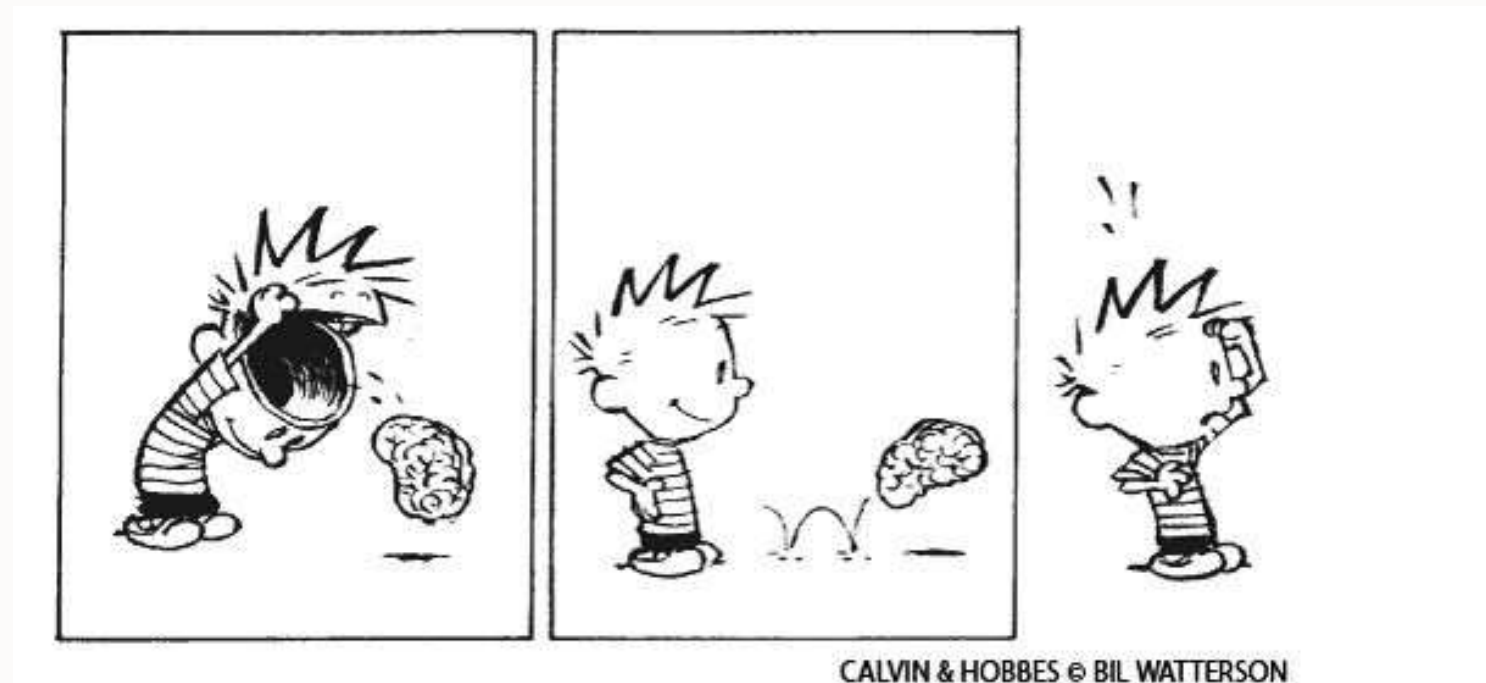
September 4, 2022

### [Java 8 to 18: Most important changes in the Java Platform](#)

Aurelio García-Ribeyro on August 29, 2022

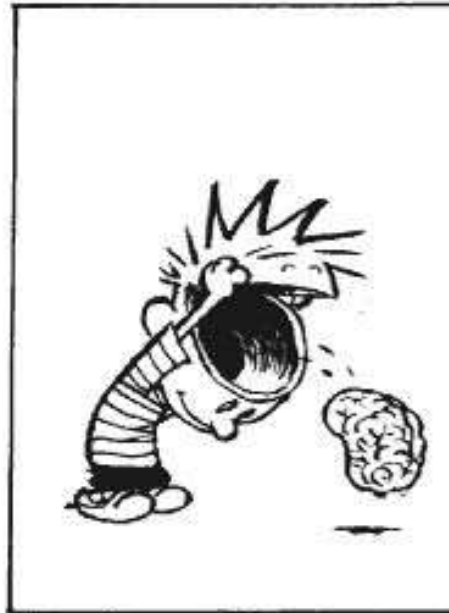
# Loom is a Work in Progress

Don't believe what we say!



# Loom is a Work in Progress

Don't believe what we sa



## JEP 436: Virtual Threads (Second Preview)

*Authors* Ron Pressler, Alan Bateman  
*Owner* Alan Bateman  
*Type* Feature  
*Scope* SE  
*Status* Completed  
*Release* 20  
*Component* core-libs  
*Discussion* [loom dash dev at openjdk dot org](https://loom.dash.dev)  
*Relates to* [JEP 425: Virtual Threads \(Preview\)](#)  
*Reviewed by* Alex Buckley  
*Endorsed by* Brian Goetz  
*Created* 2022/10/23 15:18  
*Updated* 2023/01/18 21:51  
*Issue* [8295817](#)

### Summary

Introduce *virtual threads* to the Java Platform. Virtual threads are lightweight threads that dramatically reduce the effort of writing, maintaining, and observing high-throughput concurrent applications. This is a preview API.



# Loom is a Work in Progress

Don't believe

The image shows a screenshot of a tweet from David Delabassée (@delabassée) celebrating the implementation of Virtual Threads. The tweet text includes: 'Implementation of Virtual Threads (Preview) ✓🍷🎉', 'Stats: 99468 lines in 1133 files changed: 91198 ins; 3598 del; 4672 mod 😬', and hashtags #Java19 #OpenJDK #ProjectLoom. Below the tweet is a GitHub commit preview for 'openjdk/jdk' showing a commit of 8284161 lines with a bar chart indicating 104k lines changed (+95870, -8270). The commit was made by Alan Bateman on May 7, 2022. To the right of the tweet, a partial view of another tweet is visible, mentioning 'view)' and 'Threads are lightweight maintaining, and observing view API.'

David Delabassée  
@delabassée

Implementation of Virtual Threads (Preview) ✓🍷🎉  
Stats: 99468 lines in 1133 files changed: 91198 ins;  
3598 del; 4672 mod 😬  
#Java19 #OpenJDK #ProjectLoom  
Traduire le Tweet

openjdk/jdk

**8284161:**  
**Implementation of  
Virtual Threads...**

104k lines changed +95870 -8270

Alan Bateman committed May 7, 2022 9583e36

view)  
Threads are lightweight  
maintaining, and observing  
view API.

# Loom is a Work in Progress

Don't believe what we say



## JEP 437: Structured Concurrency (Second Incubator)

*Authors* Alan Bateman, Ron Pressler  
*Owner* Alan Bateman  
*Type* Feature  
*Scope* JDK  
*Status* Completed  
*Release* 20  
*Component* core-libs  
*Discussion* [loom dash dev at openjdk dot org](https://loom.dash.dev)  
*Reviewed by* Alex Buckley  
*Endorsed by* Brian Goetz  
*Created* 2022/10/28 12:41  
*Updated* 2023/01/13 17:18  
*Issue* 8296037

### Summary

Simplify multithreaded programming by introducing an API for *structured concurrency*. Structured concurrency treats multiple tasks running in different threads as a single unit of work, thereby streamlining error handling and cancellation, improving reliability, and enhancing observability. This is an incubating API.



# Loom is a Work in Progress

Don't believe what we s



## JEP 429: Scoped Values (Incubator)

*Authors* Andrew Haley, Andrew Dinn  
*Owner* Andrew Haley  
*Type* Feature  
*Scope* JDK  
*Status* Integrated  
*Release* 20  
*Component* core-libs  
*Discussion* loom dash dev at openjdk dot java dot net  
*Relates to* [8286666: JEP 429: Implementation of Scoped Values \(Incubator\)](#)  
*Reviewed by* Alan Bateman, Alex Buckley  
*Endorsed by* John Rose  
*Created* 2021/03/04 11:03  
*Updated* 2022/12/07 11:19  
*Issue* [8263012](#)

### Summary

Introduce *scoped values*, which enable the sharing of immutable data within and across threads. They are preferred to thread-local variables, especially when using large numbers of virtual threads. This is an [incubating API](#).

# Loom is a Work in Progress

Don't believe what we s



## JEP 429: Scoped Values (Incubator)

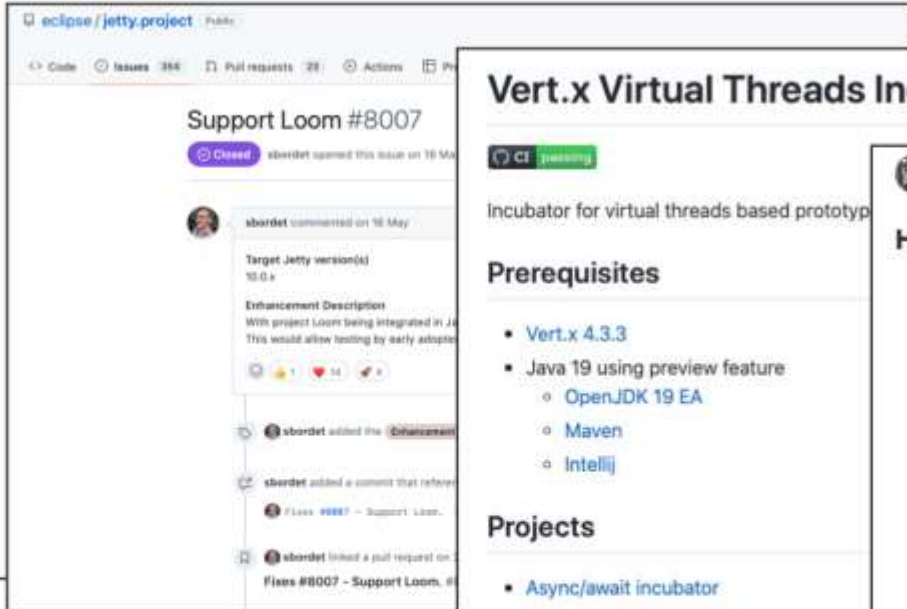
*Authors* Andrew Haley, Andrew Dinn  
*Owner* Andrew Haley  
*Type* Feature  
*Scope* JDK  
*Status* Integrated  
*Release* 20  
*Component* core-libs  
*Discussion* loom dash dev at openjdk dot java dot net  
*Relates to* [8286666: JEP 429: Implementation of Scoped Values \(Incubator\)](#)  
*Reviewed by* Alan Bateman, Alex Buckley  
*Endorsed by* John Rose  
*Created* 2021/03/04 11:03  
*Updated* 2022/12/07 11:19  
*Issue* [8263012](#)

### Summary

Introduce *scoped values*, which enable the sharing of immutable data within and

<http://jdk.java.net/loom/>

# Adoption ?



Support Loom #8007

ebardet opened this issue on 18 May

Target Jetty version(s)  
10.0.x

Enhancement Description  
With project Loom being integrated in Jetty, this would allow testing by early adopters.

ebardet added the Enhancement label

ebardet added a comment that references Files #8007 - Support Loom, #8007

ebardet linked a pull request on Files #8007 - Support Loom, #8007

## Vert.x Virtual Threads Incubator


Incubator for virtual threads based prototyp

Prerequisites

- Vert.x 4.3.3
- Java 19 using preview feature
  - OpenJDK 19 EA
  - Maven
  - IntelliJ

Projects

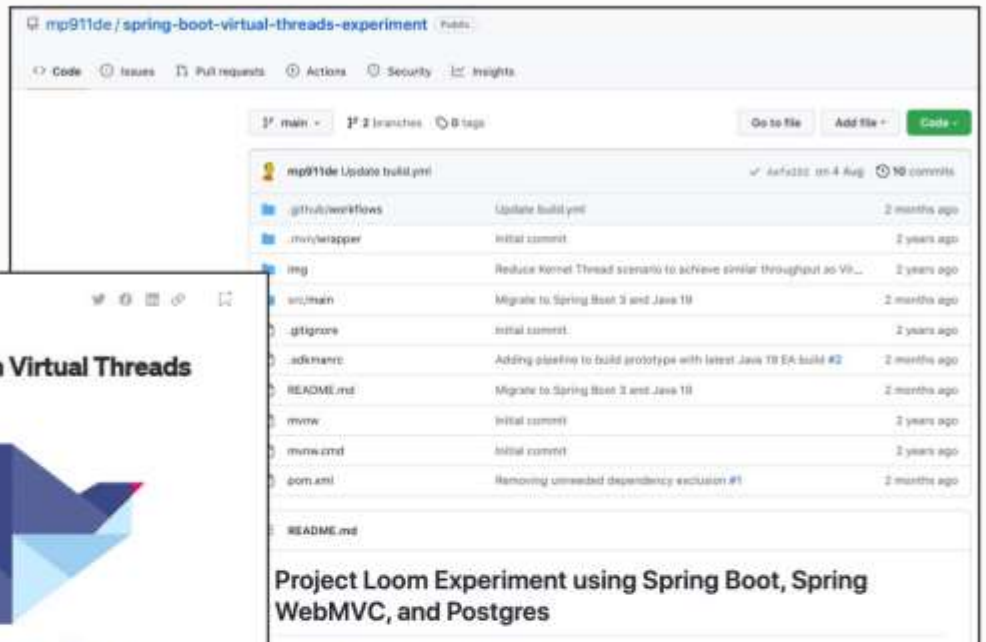
- Async/await incubator
- Execute blocking incubator
- Examples



## Helidon Nima — Helidon on Virtual Threads

HELIDON NIMA

Helidon 4.0.0-ALPHA1 is now released with our brand new Helidon Nima, providing a virtual threads-based web server. This is an early access release for those of you interested in the latest Java technology, but it is not yet suitable for production use!



mp911de / spring-boot-virtual-threads-experiment

Code Issues Pull requests Actions Security Insights

main 2 branches 0 tags

mp911de Update build.yml ✓ 4a4200 05-4 Aug 10 commits

- github/workflows Update build.yml 2 months ago
- .mvn/wrapper Initial commit 2 years ago
- ing Reduce Kernel Thread scenario to achieve similar throughput as V... 2 years ago
- src/main Migrate to Spring Boot 3 and Java 19 2 months ago
- .gitignore Initial commit 2 years ago
- .adfmavrc Adding placeholder to build prototype with latest Java 19 EA build #2 2 months ago
- README.md Migrate to Spring Boot 3 and Java 19 2 months ago
- mvnw Initial commit 2 years ago
- mvnw.cmd Initial commit 2 years ago
- pom.xml Removing unneeded dependency exclusion #1 2 months ago

Project Loom Experiment using Spring Boot, Spring WebMVC, and Postgres



spring by VMware Tanzu

Why Spring

Spring Blog All Posts Engineering Releases

## Embracing Virtual Threads

June 22, 2022 #release

## Quarkus 2.10.0.Final released

### Preliminary work on Loom's virtual threads and various refinements all over the place

By Guillaume Smet

New month, new Quarkus feature release, you know the drill: Quarkus 2.10.0.Final has landed.

This version is a mix of exploratory work and refinements on existing extensions:


- Preliminary work on Loom's virtual threads

## Virtual Threads and Tomcat

Virtual threads are the ideal mechanism for running mostly blocking tasks, providing a high level of concurrency without requiring asynchronous abstractions from business logic programmers. I show that it is easy to configure Tomcat for virtual threads, provided one makes a small change to the Tomcat source code.

### Virtual Threads

Java 19 has virtual threads as a preview feature, described in JEP 425. Virtual threads are scheduled to run in platform threads. When a virtual thread blocks, it is parked and another virtual thread can run in its place. Large numbers of virtual threads can run concurrently, provided that they mostly block. This workload is typical in web applications where requests spend much of their time waiting for responses from database queries or other external services.



# It all Started with a Runnable...

# 1995: Threads and Runnable

## 1995: Thread, Runnable

```
Runnable task = new Runnable() {  
    void run() {  
        System.out.println("I am running in thread " +  
            Thread.currentThread().getName());  
    }  
};  
Thread thread = new Thread(task);  
thread.start();  
thread.join(); // blocks
```



# 1995: Threads and Runnable

## 1995: Thread, Runnable

```
Object key = new Object();  
  
synchronized(key) {  
    System.out.println("Only one thread can execute me!");  
}
```

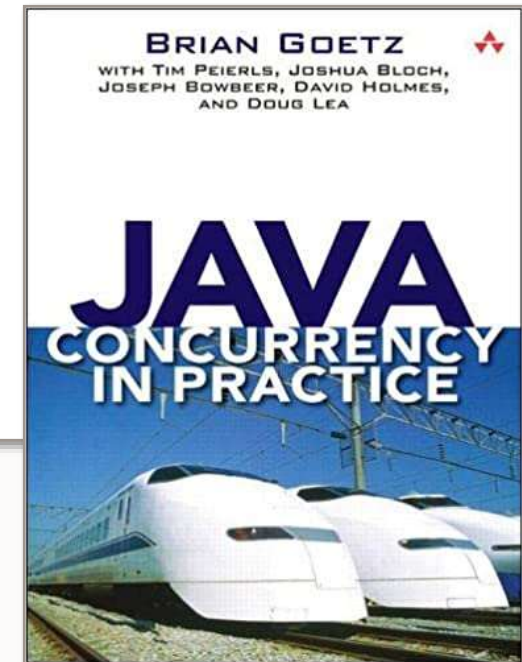




# 2004: Java Util Concurrent

2004: Java 5, `java.util.concurrent`

```
Callable<String> task = new Callable<String>() {  
  
    @Override  
    public String call() throws Exception {  
        return "I am running in thread " +  
            Thread.currentThread().getName();  
    }  
};
```



# 2004: Java Util Concurrent

2004: Java 5, `java.util.concurrent`

```
ExecutorService service =  
    Executors.newFixedThreadPool(4);  
  
Future<String> future = service.submit(task);
```

Wait lists inside!



# 2004: Java Util Concurrent

2004: Java 5, `java.util.concurrent`

```
String result = future.get(); // blocks  
  
String result = future.get(10, TimeUnit.MICROSECONDS);  
  
boolean cancelled = future.cancel(true);
```



# 2004: Java Util Concurrent

2004: Java 5, `java.util.concurrent`

```
Lock lock = new ReentrantLock();
lock.lock();
try {

    System.out.println("Only one thread can execute me!");

} finally {
    lock.unlock();
}
```



# 2004: Java Util Concurrent

2004: Java 5, `java.util.concurrent`

Plus many more concurrent classes:

- Lock, Semaphore, Barrier, CountdownLatch
- BlockingQueue, ConcurrentMap
- CopyOnWriteArrayList



# 2011: Fork / Join

2011 – 2014 (Java 7, Java 8):

- Fork / Join, parallel Stream

Allows to compute elements in parallel

Two phases:

- fork = splits a task in two sub-tasks
- join = merge the result of two sub-tasks

Uses work stealing to spread the tasks among threads



# 2014: CompletionStage

2011 – 2014 (Java 7, Java 8):

- CompletionStage, CompletableFuture

Subtype of Future

Asynchronous programming model

Allows to trigger tasks on the outcome of other tasks

User can control which thread executes what task

Exceptions handling



# One thing stays the same

Once a thread begins to process a task it cannot release it

Either the task completes with a result

Or is completes with an exception

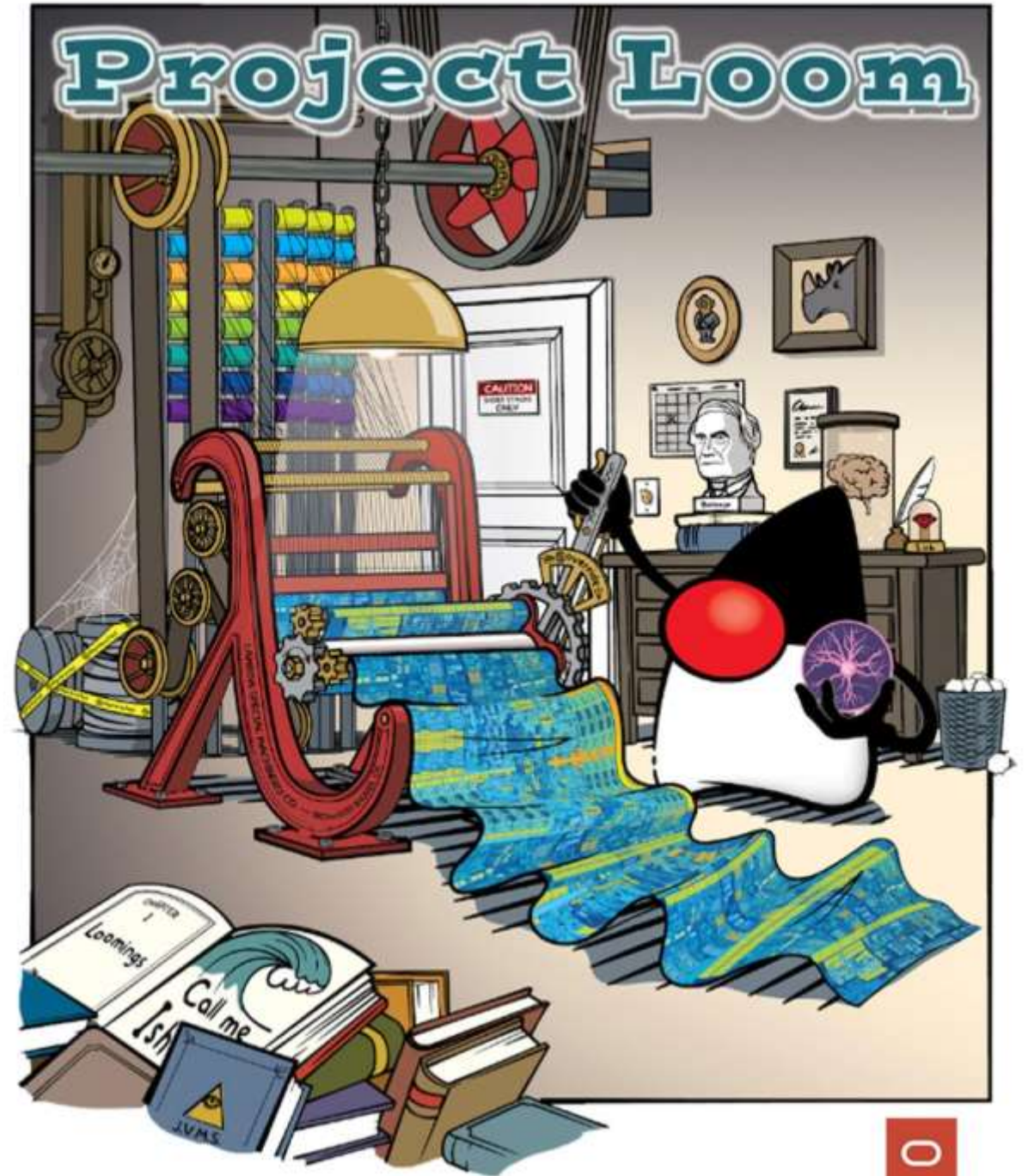
It may be an `InterruptedException`





# 2023?: Loom!

2022+ (prev. in Java 19)



# Why Do We Need Concurrency?



# Concurrency: Computations vs. I/O

Concurrency may be used in two different contexts:

- 1) Processing in-memory data in parallel, using all the CPU cores
  - Each thread uses 100% of your CPU cores
  - Threads are mostly not blocking



# Concurrency: Computations vs. I/O

Concurrency may be used in two different contexts:

2) Handling numerous blocking requests / responses

HTTP Server → 1 request  $\Leftrightarrow$  1 thread

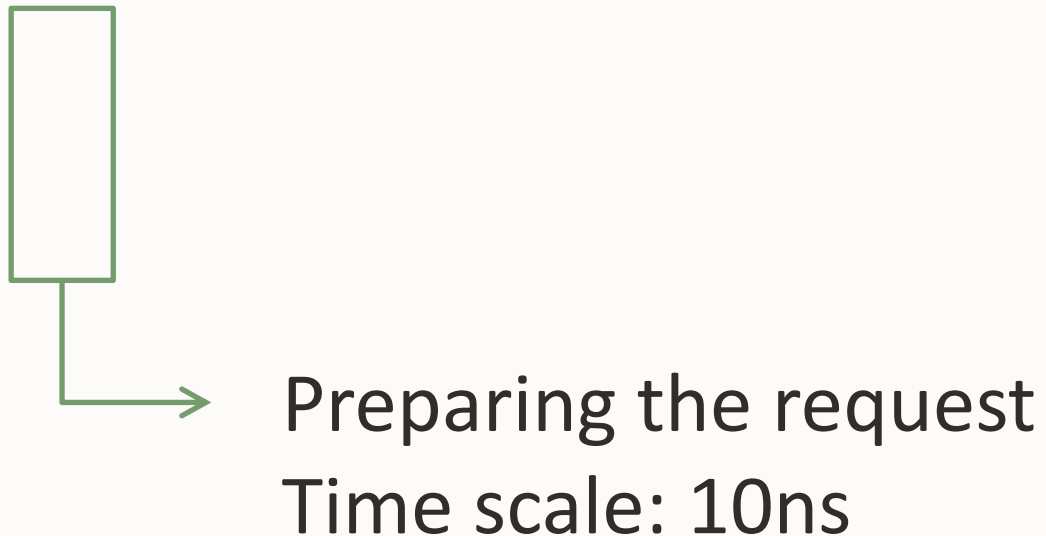
DB Server → 1 transaction  $\Leftrightarrow$  1 thread



# Concurrency for I/O

Processing I/O data:

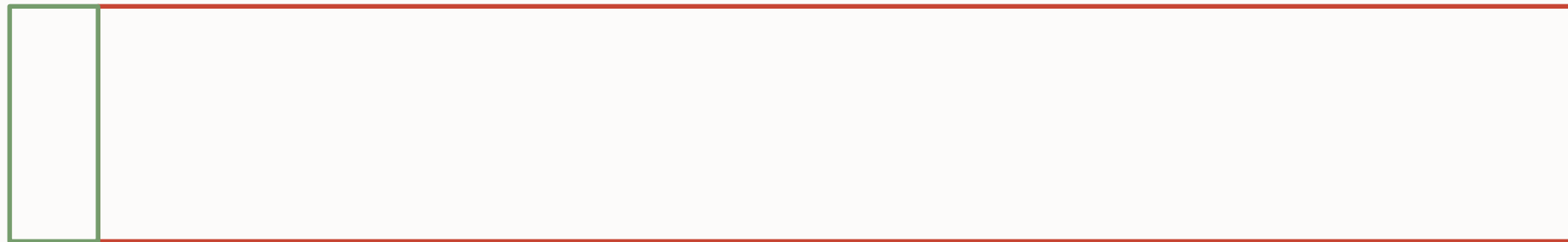
- Each task waits for the data it needs to process



# Concurrency for I/O

Processing I/O data:

- Each task waits for the data it needs to process



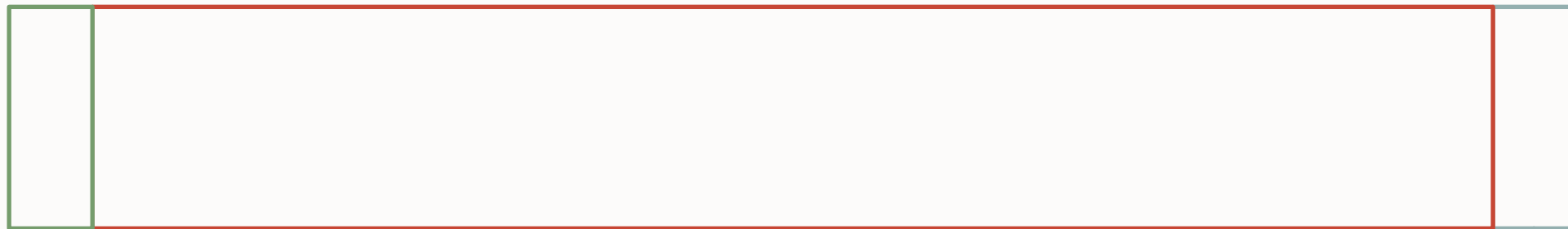
Waiting for the response  
Time scale: 10ms



# Concurrency for I/O

Processing I/O data:

- Each task waits for the data it needs to process



Processing the response

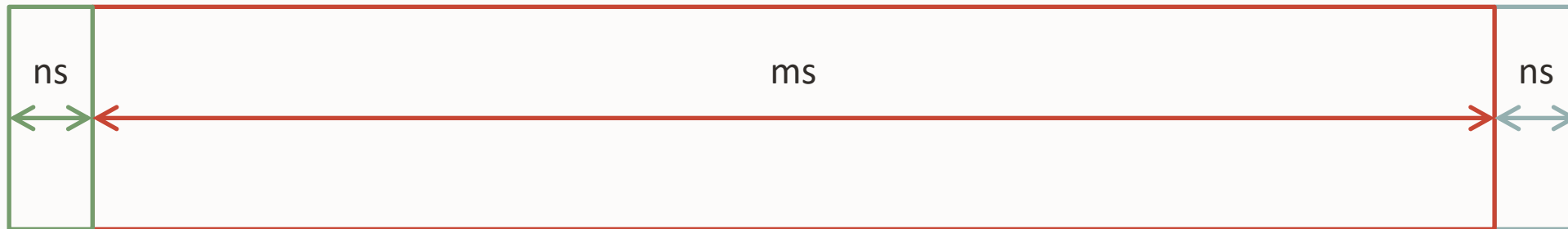
Time scale: 10ns



# Concurrency for I/O

Processing I/O data:

A Thread is idle 99.9999% of the time!



How many threads do you need to keep your CPU busy?





# Concurrency for I/O

A thread is not cheap!

- Thread startup time: ~1ms
- Thread memory consumption: 2MB of stack
- Context switching: ~100 $\mu$ s (depends on the OS)

Having 1 million platform threads is not possible!



# Solutions?

CompletionState / CompletableFuture

Asynchronous / Reactive programming

Async / Await (C# or Kotlin)

Mono / Multi (Spring)

Uni / Multi (Quarkus)



# Solutions?

Breaking down a request handling into small stages

Then compose them into a pipeline

The code becomes:

- hard to read and write (callback hell)
- hard to debug (call stack?)
- hard to test
- hard to profile



# Loom to the Rescue



# Virtual Thread!

```
// platform threads  
var pthread = new Thread(() -> {  
    System.out.println("platform " + Thread.currentThread());  
});  
pthread.start();  
pthread.join();
```



# Virtual Thread!

```
// virtual threads
var vthread = Thread.startVirtualThread(() -> {
    System.out.println("virtual " + Thread.currentThread());
});
vthread.join();
```

```
// platform threads
var pthread = Thread.ofPlatform(() -> {
    System.out.println("platform " + Thread.currentThread());
});
pthread.join();
```



# Virtual Thread!

```
// platform threads  
platform Thread[#14,Thread-0,5,main]  
  
// virtual threads  
virtual VirtualThread[#15]/runnable@ForkJoinPool-1-worker-1
```

A virtual thread runs on a carrier thread from a Fork-Join pool (not the common fork join pool)

This pool implements a FIFO queue (instead of a LIFO one)



# Thread Polymorphic Builder

```
// platform threads
var pthread = Thread.ofPlatform()
    .name("platform-", 0)
    .start(() -> {
        System.out.println("platform " + Thread.currentThread());
    });
pthread.join();

// virtual threads
var vthread = Thread.ofVirtual()
    .name("virtual-", 0)
    .start(() -> {
        System.out.println("virtual " + Thread.currentThread());
    });
vthread.join();
```



How many **virtual** threads can I run?

# Running a Thread

Platform/OS thread (starts in **ms**)

- Creates a 2MB stack upfront
- System call to ask the OS to schedule the thread

Virtual thread (starts in **μs**)

- Grow and shrink the stack dynamically
- Use a specific fork-join pool of platform threads (carrier threads)
- One platform thread per core



# How does it work under the hood?



# Continuation



# Where Does the Magic Come From?

Internal API

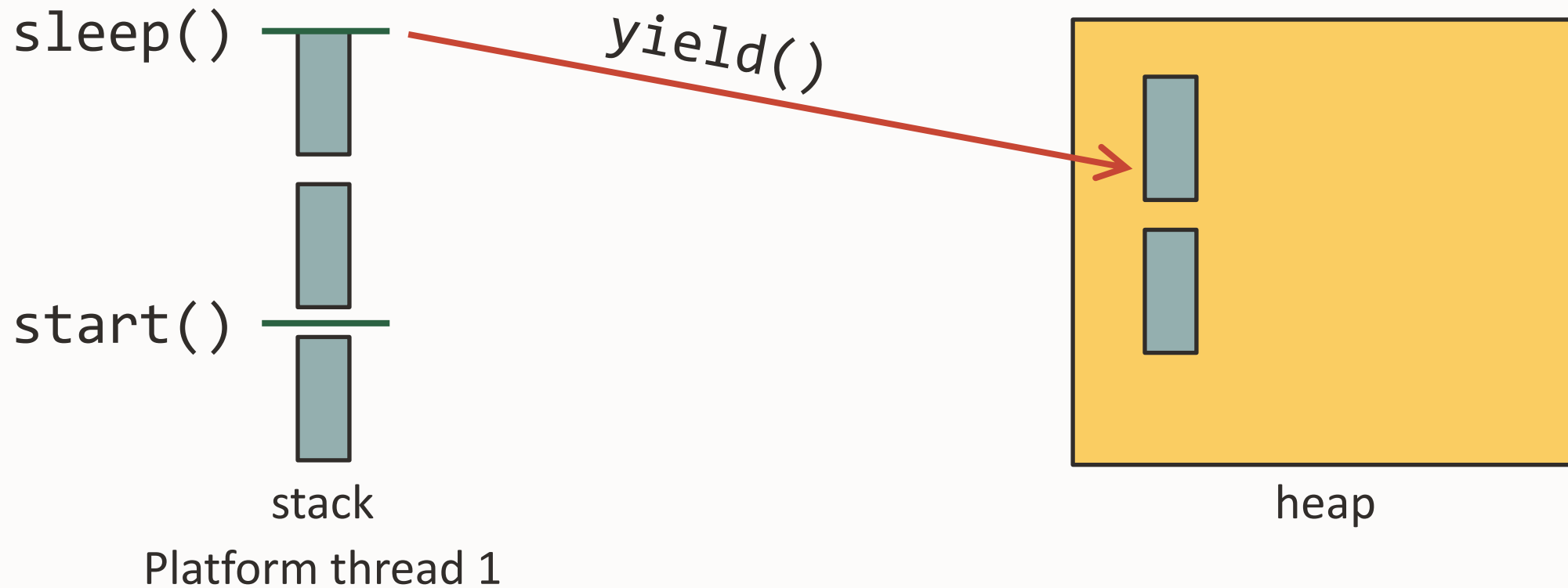
```
@ChangesCurrentThread
private boolean yieldContinuation() {
    boolean notifyJvmti = notifyJvmtiEvents;
    // unmount
    if (notifyJvmti) notifyJvmtiUnmountBegin(false);
    unmount();
    try {
        return Continuation.yield(VTHREAD_SCOPE);
    } finally {
        // re-mount
        mount();
        if (notifyJvmti) notifyJvmtiMountEnd(false);
    }
}
```



# Continuation.yield()

Internal API

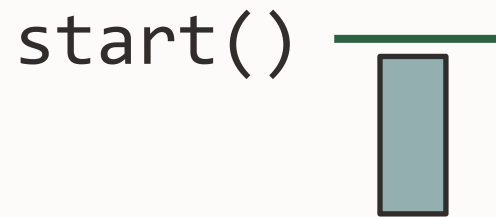
yield() copies the stack to the heap



# Continuation.yield()

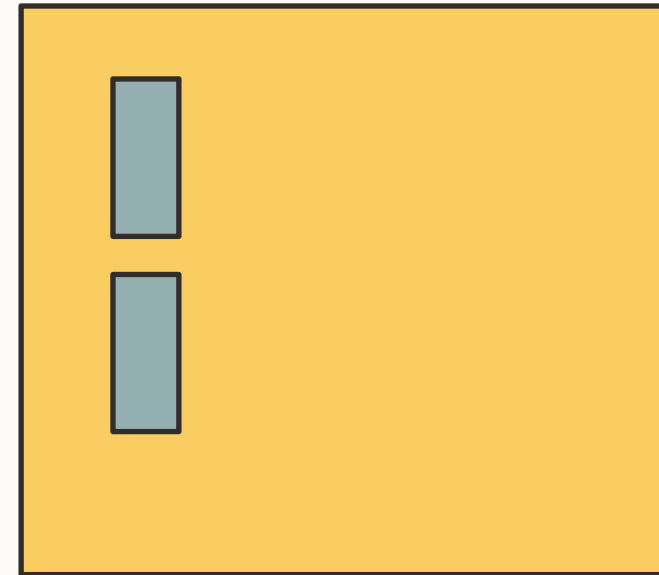
*Internal API*

`yield()` copies the stack to the heap



stack

Platform thread 1



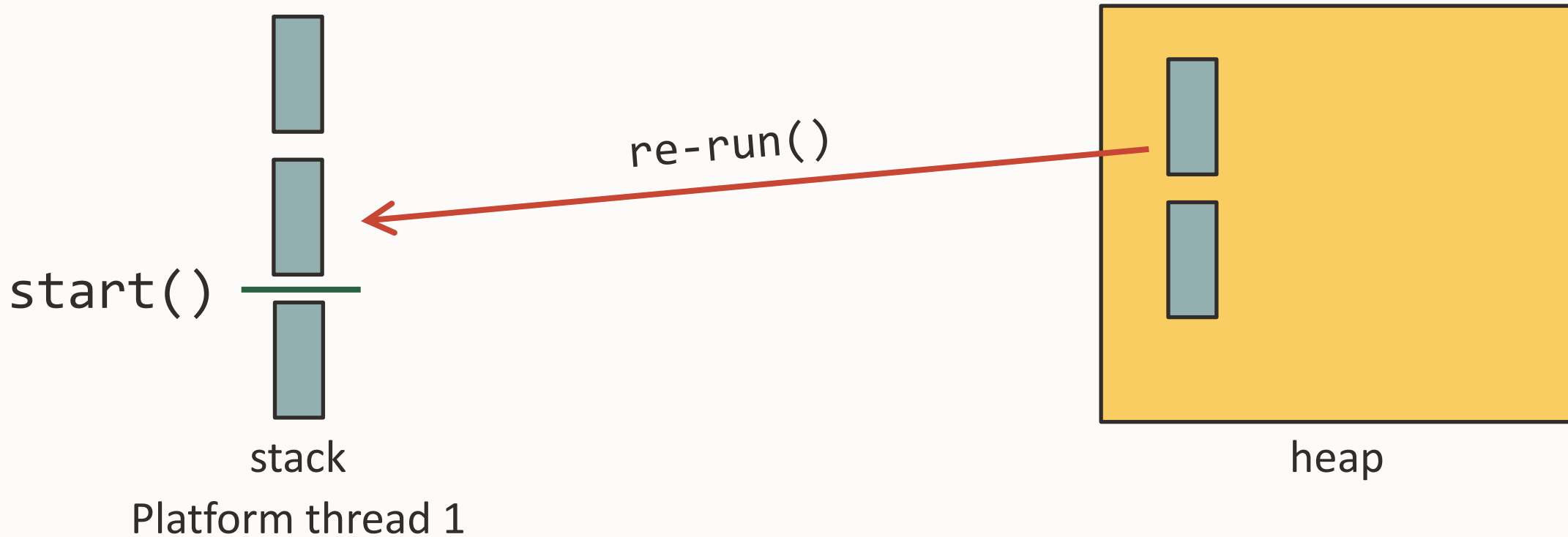
heap



# Continuation.run()

*Internal API*

run() copies from the heap to another stack  
(optimization: only copies the topmost stack frames)

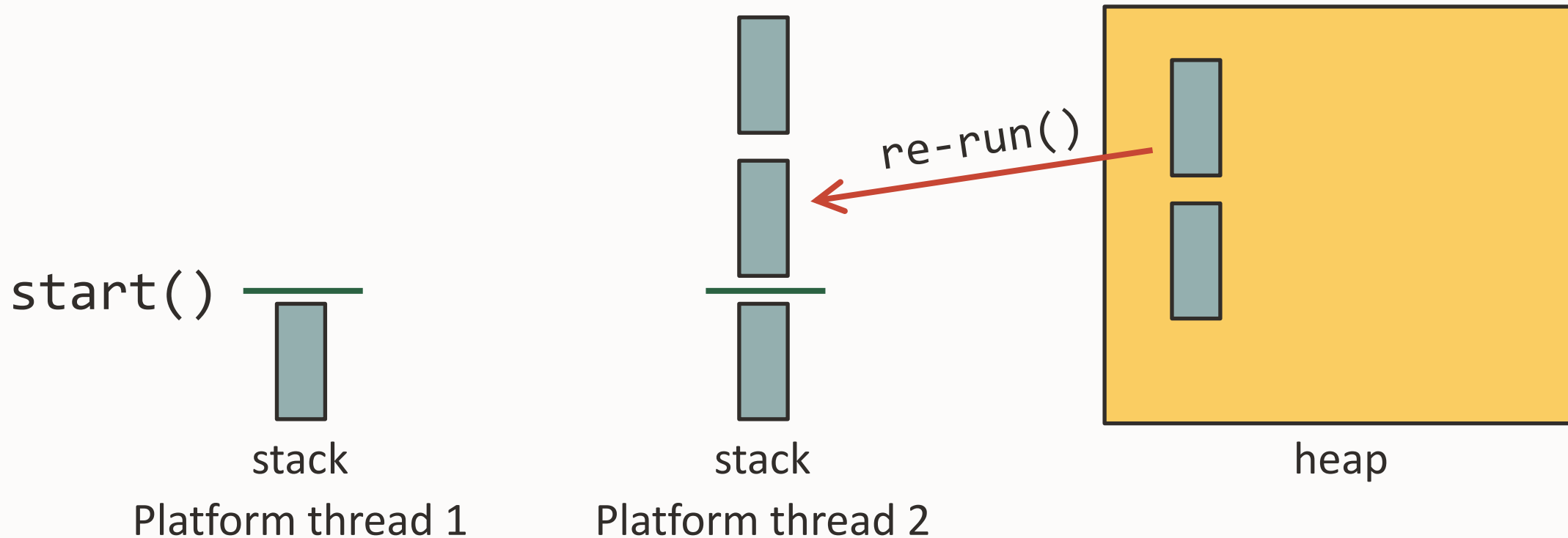




# Continuation.run()

*Internal API*

run() copies from the heap to another stack  
(optimization: only copies the topmost stack frames)



# jdk.internal.vm.Continuation

Internal API

```
var scope = new ContinuationScope("hello");
var continuation = new Continuation(scope, () -> {
    System.out.println("C1");
    Continuation.yield(scope);
    System.out.println("C2");
    Continuation.yield(scope);
    System.out.println("C3");
});
System.out.println("start");
continuation.run();
System.out.println("came back");
continuation.run();
System.out.println("back again");
continuation.run();
System.out.println("back again again");
```

Execution:

start

C1

came back

C2

back again

C3

back again again



# There Are Cases Where It Does Not Work

Sometimes virtual threads are pinned to their carrier thread

Native code that does an upcall to Java may use an address on stack

⇒ the stack frames can not be copied



# Running a Virtual Thread

A Platform Thread is a thin wrapper on an OS Thread

A Virtual Thread is not tied to a particular OS Thread

A Virtual Thread only consumes an OS Thread  
when it performs calculations on the CPU



Creating a virtual thread is cheap

Blocking a virtual thread is cheap

Pooling virtual threads is useless

# Loom is not Implemented « By the JVM »

Most of the code of the virtual threads scheduling is written in Java in the JDK (`jdk.internal.vm.Continuation`)

Written in C in the JVM:

- Copy of the stack frames back and forth
- GCs modified to find references in stack on heap



# In the JDK

All blocking codes are changed to

- Check if current thread is a virtual thread
  - If it is, instead of blocking:
    - Register a handler that will be called when the OS is ready (using NIO)
    - Call `Continuation.yield()`
    - When the handler is called, find a carrier thread and call `Continuation.start()`



# There Are Cases Where It Does Not Work

Sometimes virtual threads are pinned to their carrier thread

Synchronized blocks are written in assembly and use an address on the stack

⇒ the stack frames can not be copied

Prefer `ReentrantLock` over `synchronized()`





# Stealth Rewrite of the JDK for Loom

## Java 13

- JEP 353 Reimplement the Legacy Socket API

## Java 14

- JEP 373 Reimplement the Legacy Datagram Socket API
- JEP 374 Deprecate and Disable Biased Locking



# Stealth Rewrite of the JDK for Loom

## Java 18

- JEP 416 Reimplement Core Reflection with Method Handles
- JEP 418 (Internet-Address Resolution SPI) in JDK 18 defined a service-provider interface for host name and address lookup. This will allow third-party libraries to implement alternative `java.net.InetAddress` resolvers that do not pin threads during host lookup



# Loom Idea: Under the Hood

The JDK creates as many virtual threads as the user want

- Mount a virtual thread to an available carrier thread when starting
- If blocking, unmount the current virtual thread and mount another virtual thread





**Coffee (or whatever)  
break!**

# Structured Concurrency



# Why Do You Need Structured Concurrency?

Because thread dumps work well with several thousands of threads, not millions of threads

Not to talk about what can happen in your IDE...

You need to structure these threads



# Structured Task Scope

Welcome to Loom Scopes

- It's a pool of threads, that creates virtual threads on demand
- Once a task is done, the thread dies



# The Travel Agency Example

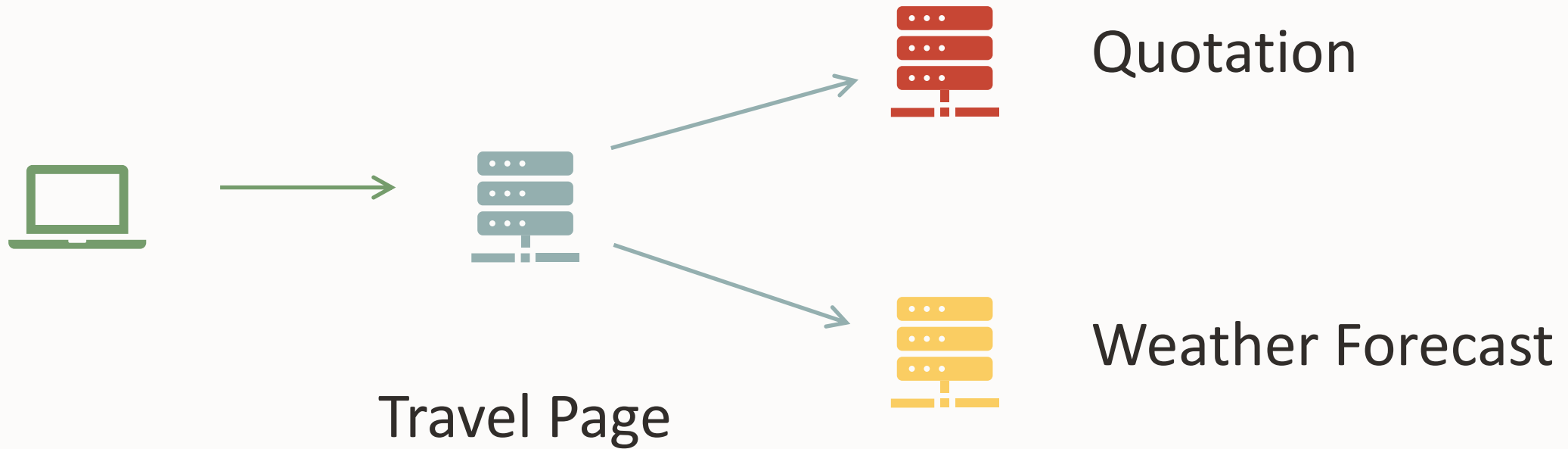
A travel agency sells travels. On the response page, it wants to display:

- the quotation
- the weather forecast for the destination





# The Travel Agency Example

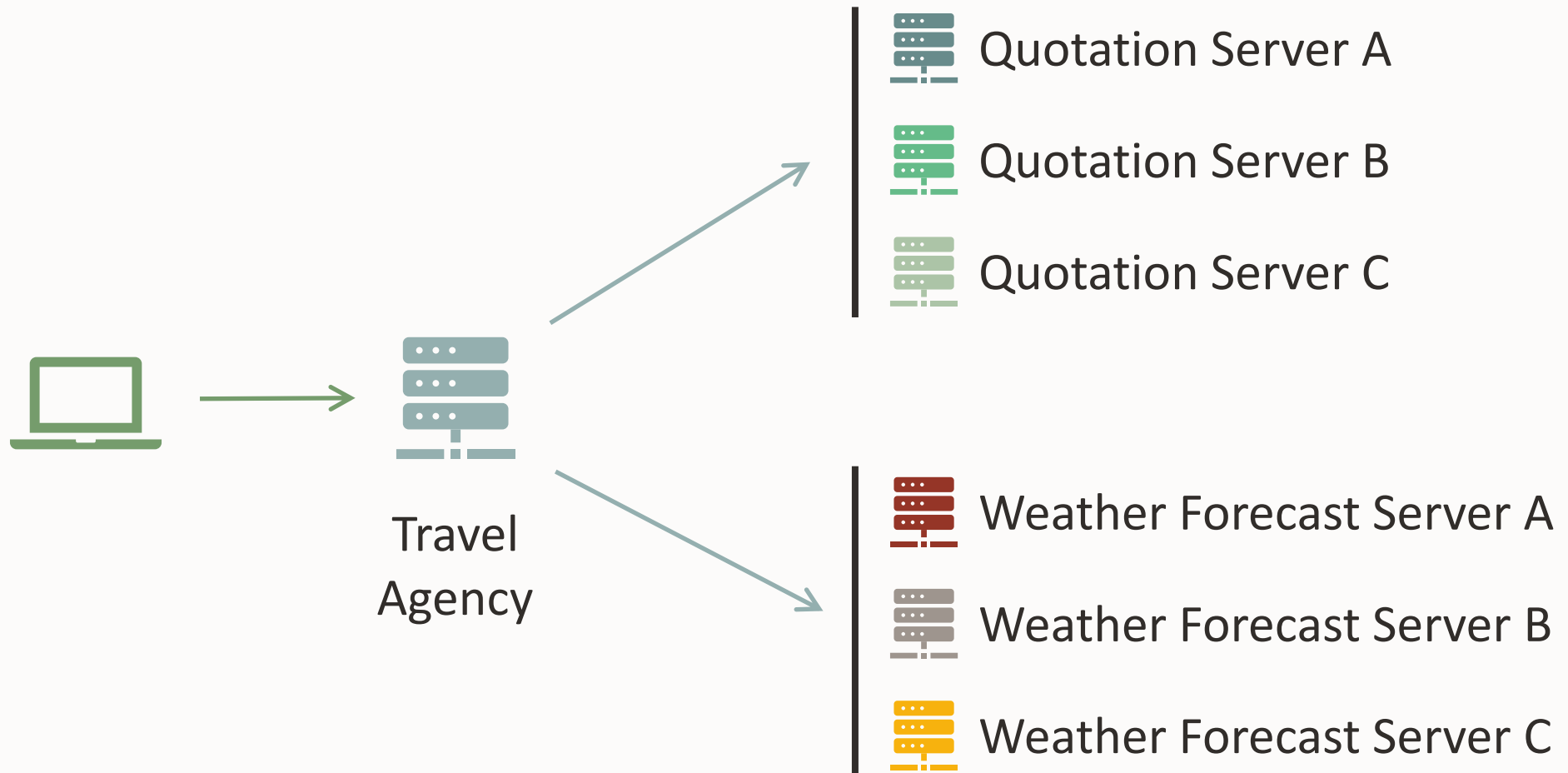


# CompletableFuture Based Travel Agency

```
var quotationCF =
    CompletableFuture.supplyAsync(() -> getQuotation());
var weatherCF =
    CompletableFuture.supplyAsync(() -> getWeather());

CompletableFuture<Page> travelPageCF =
    quotationCF
        .exceptionally(t -> {
            weatherCF.cancel(true);
            throw new RuntimeException(t);
        })
        .thenCompose(
            quotation -> weatherCF
                // .completeOnTimeout(Weather.UNKNOWN, 100, MILLISECONDS)
                .exceptionally(e -> Weather.UNKNOWN)
                .thenApply(
                    weather ->
                        buildPage(quotation, weather)));
```

# The Travel Agency Example



# Structured Scope

It needs to be closed (try with resources FTW!)

It creates virtual threads on demand

Pattern:

- Launch tasks
- Call `join()`
- Get the results



# StructuredTaskScope

A StructuredTaskScope object looks like an ExecutorService

- It takes tasks and run then
- And returns Future

But:

- An executor lives with your application
- A task scope lives with your tasks



# StructuredTaskScope

- ShutdownOnSuccess
- ShutdownOnFailure

Can be extended to implement specific needs



# Extending StructuredTaskScope

Allows you to implement your own logic and error handling

`handleComplete(Future<>)` is the method you need to override



# CompletableFuture Based Travel Agency

```
var quotationCF =
    CompletableFuture.supplyAsync(() -> getQuotation());
var weatherCF =
    CompletableFuture.supplyAsync(() -> getWeather());

CompletableFuture<Page> travelPageCF =
    quotationCF
        .exceptionally(t -> {
            weatherCF.cancel(true);
            throw new RuntimeException(t);
        })
        .thenCompose(
            quotation -> weatherCF
                // .completeOnTimeout(Weather.UNKNOWN, 100, MILLISECONDS)
                .exceptionally(e -> Weather.UNKNOWN)
                .thenApply(
                    weather ->
                        buildPage(quotation, weather)));
```



# Structured Concurrency Based Travel Agency

```
try (var scope = new WeatherScope()) {  
  
    scope.fork(() -> readWeatherFromA());  
    scope.fork(() -> readWeatherFromB());  
    scope.fork(() -> readWeatherFromC());  
  
    scope.join();  
  
    Weather firstWeather = scope.getFirstWeather();  
    return firstWeather;  
}
```

# Structured Concurrency Based Travel Agency

```
try (var scope = new QuotationScope()) {  
  
    scope.fork(() -> readQuotationFromA());  
    scope.fork(() -> readQuotationFromB());  
    scope.fork(() -> readQuotationFromC());  
  
    scope.join();  
  
    Quotation bestQuotation = scope.getBestQuotation();  
    return bestQuotation;  
}
```

# Structured Concurrency Based Travel Agency

```
try (var scope = new TravelPageScope()) {  
  
    scope.fork(() -> getFirstWeather());  
    scope.fork(() -> getBestQuotation());  
  
    scope.join();  
  
    TravelPage page = scope.buildTravelPage();  
    return page;  
}
```

# Structured Concurrency Based Travel Agency

```
protected void handleComplete(Future<Quotation> future) {  
  
    switch (future.state()) {  
        case RUNNING -> throw new IllegalStateException("Ooops");  
        case SUCCESS -> this.quotations.add(future.resultNow());  
        case FAILED -> this.exceptions.add(future.exceptionNow());  
        case CANCELLED -> { }  
    }  
}
```

# Structured Concurrency Based Travel Agency

```
public Quotation bestQuotation() {  
    return this.quotations.stream()  
        .min(Comparator.comparing(Quotation::quotation))  
        .orElseThrow(this::exceptions);  
}  
  
public QuotationException exceptions() {  
    QuotationException exception = new QuotationException();  
    this.exceptions.forEach(exception::addSuppressed);  
    return exception;  
}
```

```
try (var scope = new TravelPageScope()) {  
  
    scope.fork(() -> getFirstWeather());  
    scope.fork(() -> getBestQuotation());  
  
    scope.join();  
  
    TravelPage page = scope.buildTravelPage();  
    return page;  
}
```



# Stack Trace, ThreadDumps?

```
> jcmd <pid> Thread.dump_to_file -format-json <filename.json>
```



# ThreadLocal?

ThreadLocal are made to pass some information  
Without relying to method parameters!





# ThreadLocal?

ThreadLocal is a variable bound to a thread  
That can be read through this thread

```
ThreadLocal<String> threadLocal = new ThreadLocal<>();  
  
threadLocal.set("KEY_1");  
  
System.out.println(threadLocal.get()); // KEY_1  
  
new Thread(  
    () -> System.out.println(threadLocal.get())  
).start(); // null
```



# ThreadLocal under the hood

- 1) Thread local variables are stored in a map  
And are mutable!
- 2) Creating a new thread copies the map from the current thread
- 3) You know that there is a `remove()` method on `ThreadLocal`?



# Virtual Threads support ThreadLocal variables

but you can do better!

# Virtual Threads support ThreadLocal variables

but you can do better!

# Welcome to ScopedValue

ScopedValues are non-modifiable

They are not bound to a particular thread

```
ScopedValue<String> key = new ScopedValue.newInstance();

ScopedValue.where(key, "KEY_1")
    .run(() -> doSomethingSmart());

ScopedValue.where(key, "KEY_2")
    .run(() -> doSomethingSmart())
    .run(() -> soSomethingSmarter());
```





**Loom is Great!**