



PROGRAMOWANIE WIELOWĄTKOWE – VIRTUAL THREADS

JDK 21 – Game Changer



ABOUT ME

Grzegorz Ząbek

Senior Software Consultant

Over 10 years experience in programming.

Mostly focused in Java but have some frontend experience (Angular and Vue).

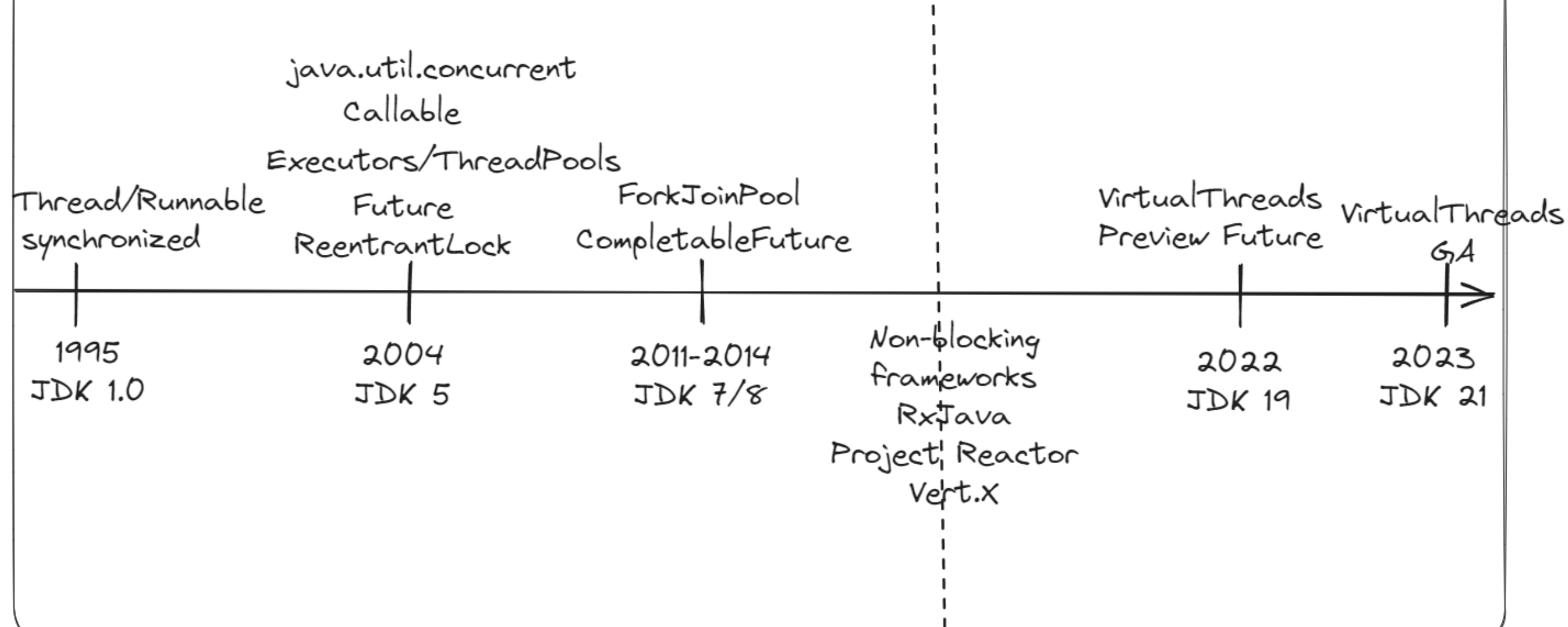
Seeking for new features and develop programming skills.

<https://github.com/gzabek/>





Concurrency in JAVA





One thing stays the same

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

Either the task completes with a result

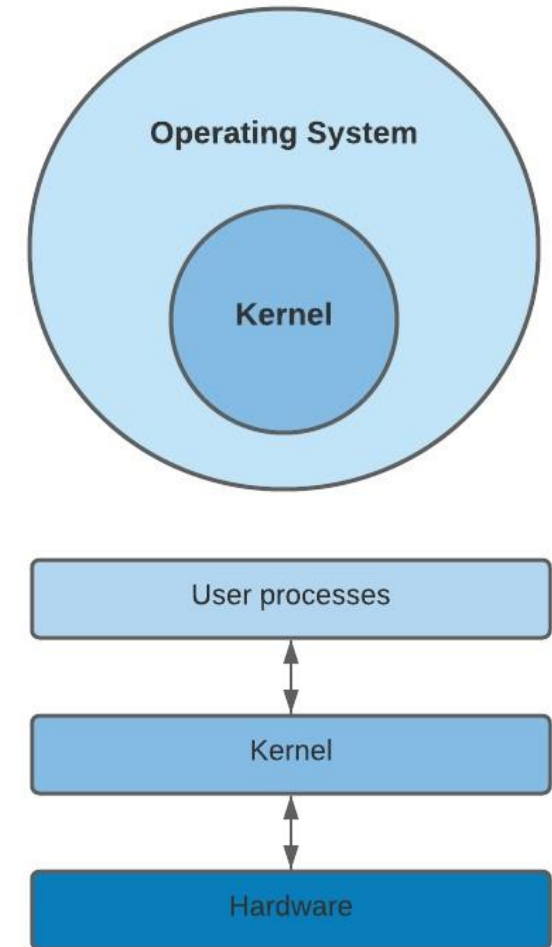
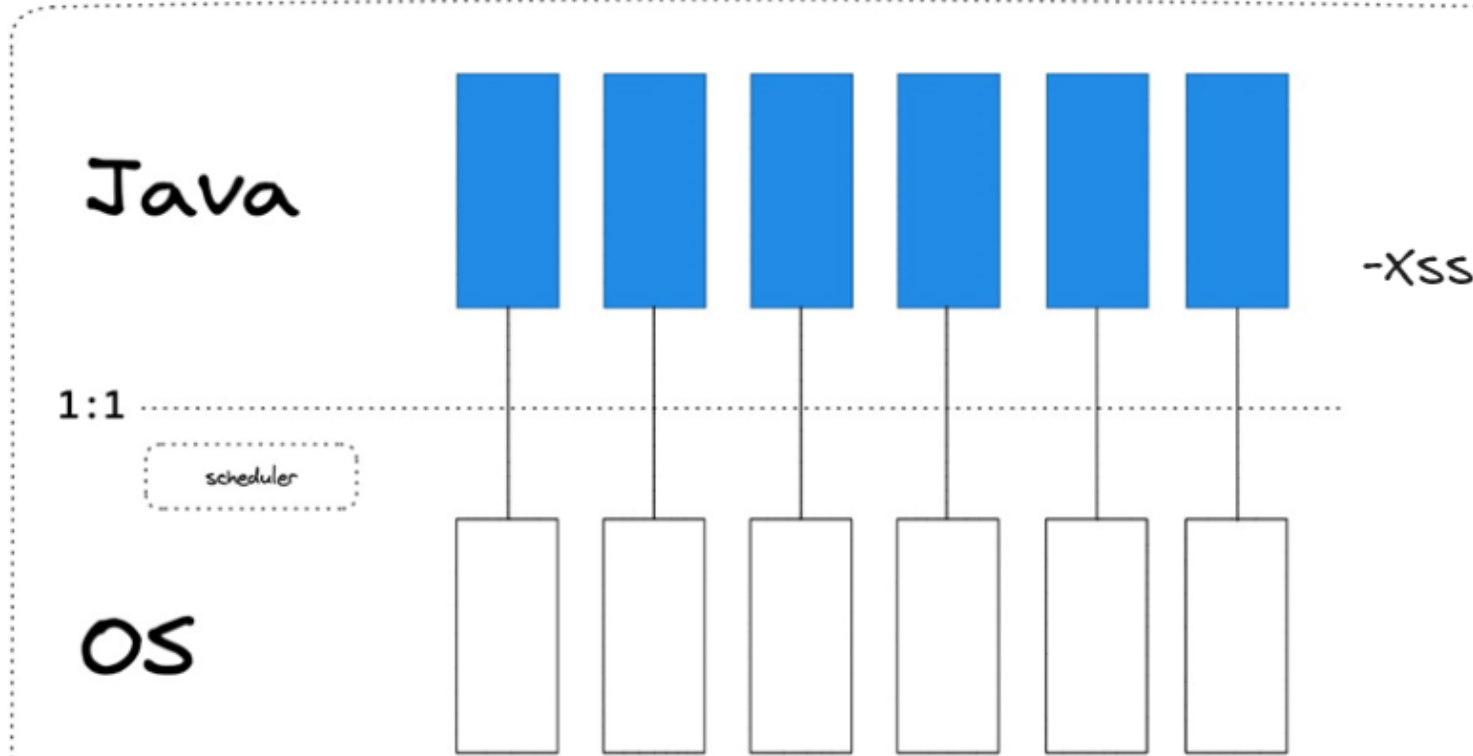
Or is completes with an exception

It may be an `InterruptedException`



THREADS IN JAVA – CURRENT MODEL

Java is Made of Threads



StackOverflowError (the stack size is greater than the limit), increase the value

OutOfMemoryError: unable to create new native thread (too many threads, each thread has a large stack), decrease it.



Concurrency: Computations vs. I/O

Concurrency may be used in two different contexts:

- processing in-memory data in parallel, using all the CPU cores
- handling numerous blocking requests / responses

Computing in parallel:

- Each thread uses 100% of your CPU cores
- Threads are mostly not blocking

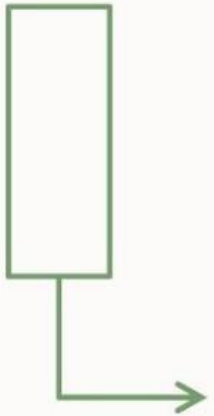
No need to have more threads than (physical) CPU cores



Concurrency for I/O

Processing I/O data:

- Each task waits for the data it needs to process



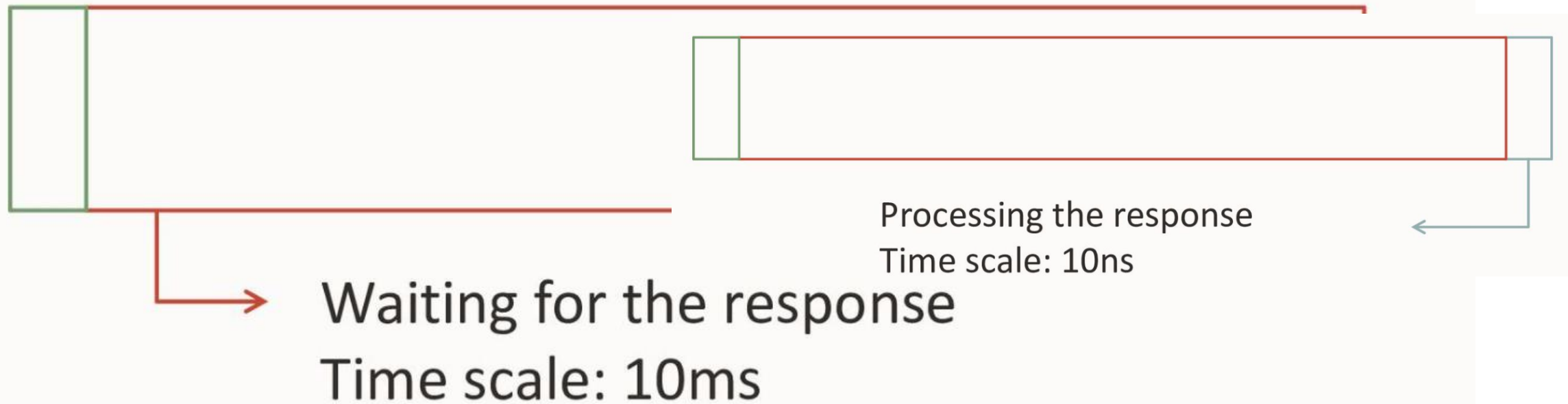
Preparing the request
Time scale: 10ns

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:

A Thread is idle 99.9999% of the time!



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

Concurrency for I/O



Because a thread cannot switch from one task to the other, the only solution is to have more threads

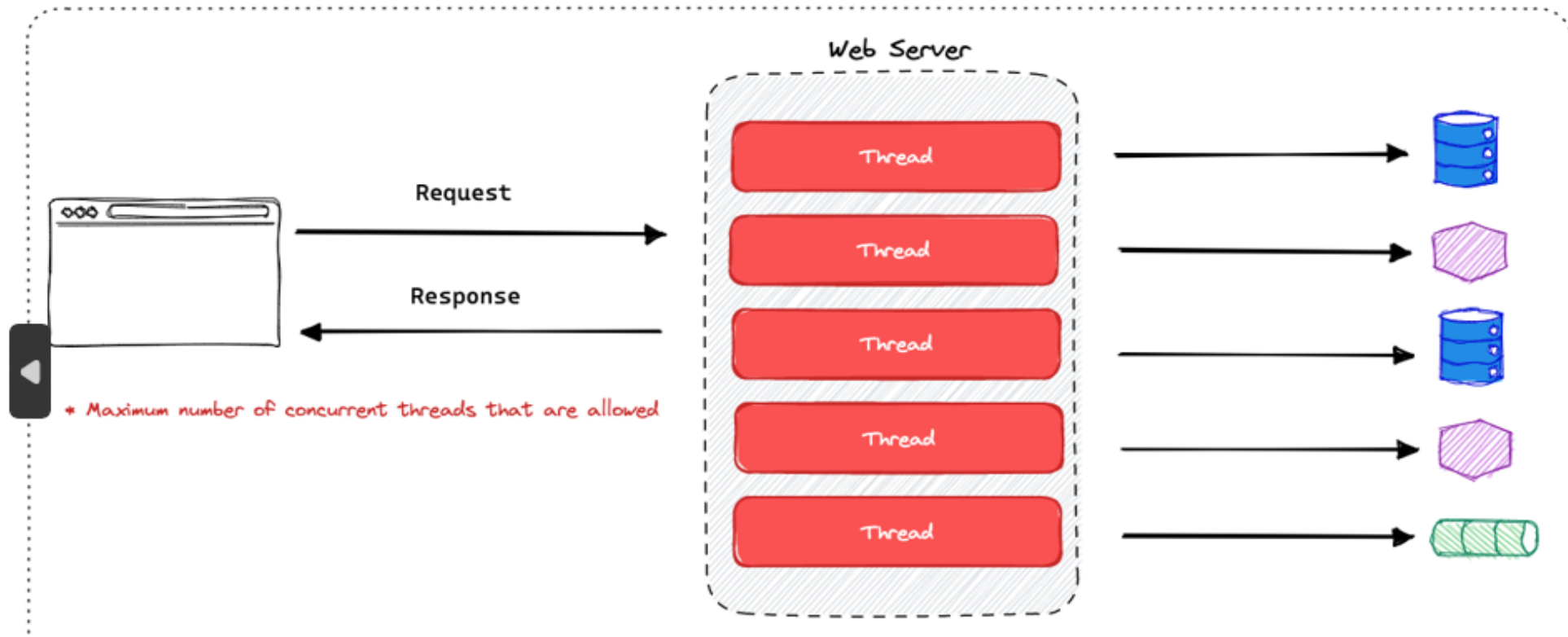
But a thread is not cheap!

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



BLOCKING MODEL

Thread Per Request





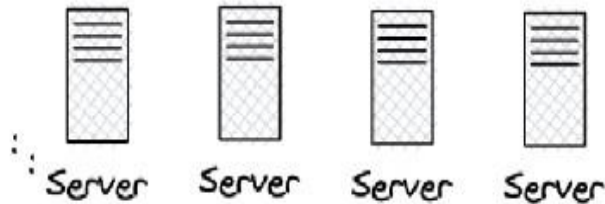
SCALABILITY SOLUTION

Hardware

Scaling Vertically



Scaling Horizontally



Asynchronous Programming

- CompletableFuture
- RxJava
- Project Reactor

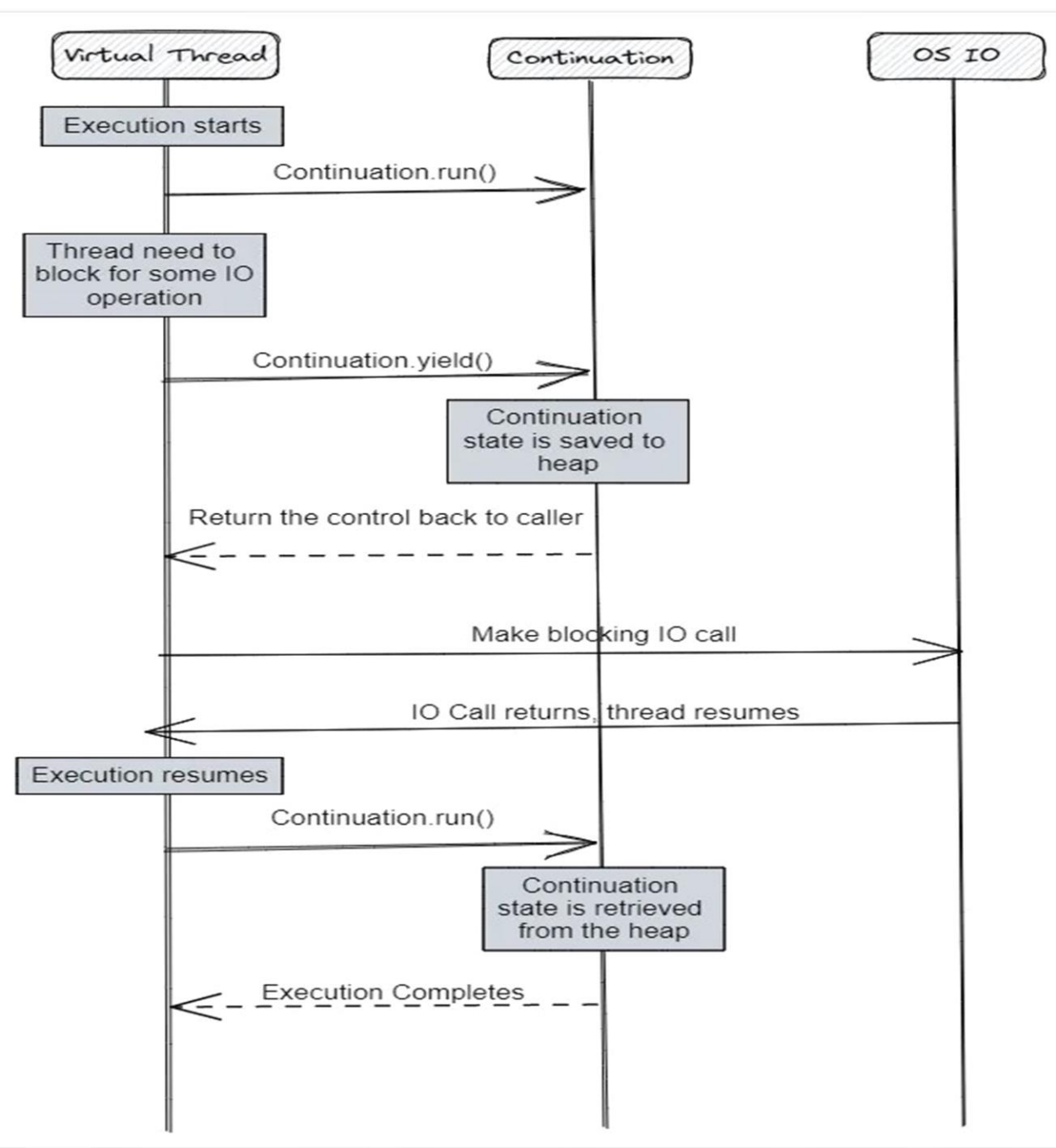
```
public DeferredResult<ResponseEntity<?>> createOrder(
    CreateOrderRequest createOrderRequest, Long sessionId, HttpServletRequest context) {
    DeferredResult<ResponseEntity<?>> deferredResult = new DeferredResult<>();

    Observable.just(createOrderRequest)
        .doOnNext(this::validateRequest)
        .flatMap(
            request ->
                sessionService
                    .getSessionContainer(request.getClientId(), sessionId)
                    .toObservable()
                    .map(ResponseEntity::getBody))
        .map(
            sessionContainer ->
                enrichCreateOrderRequest(createOrderRequest, sessionContainer, context))
        .flatMap(
            enrichedRequest ->
                orderPersistenceService.persistOrder(enrichedRequest).toObservable())
        .subscribeOn(Schedulers.io())
        .subscribe(
            success -> deferredResult.setResult(ResponseEntity.noContent()),
            error -> deferredResult.setErrorResult(error));

    return deferredResult;
}
```

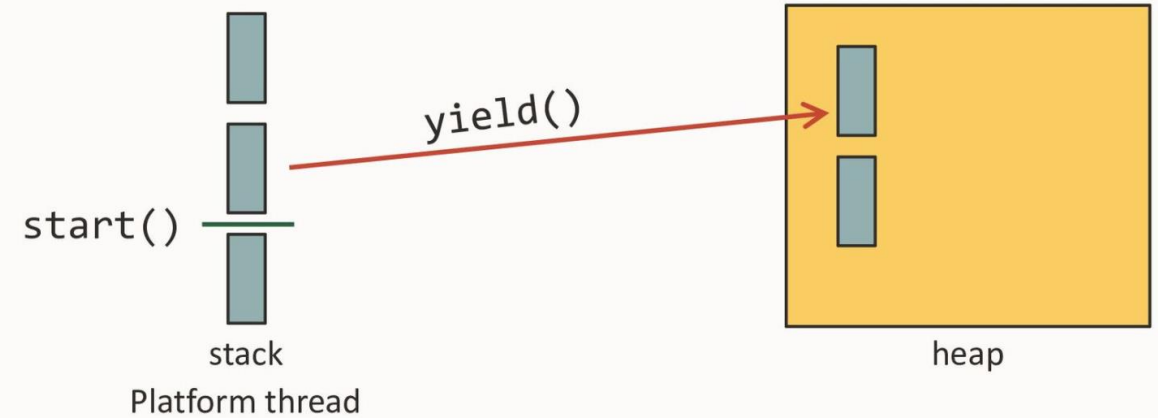

VIRTUAL THREADS





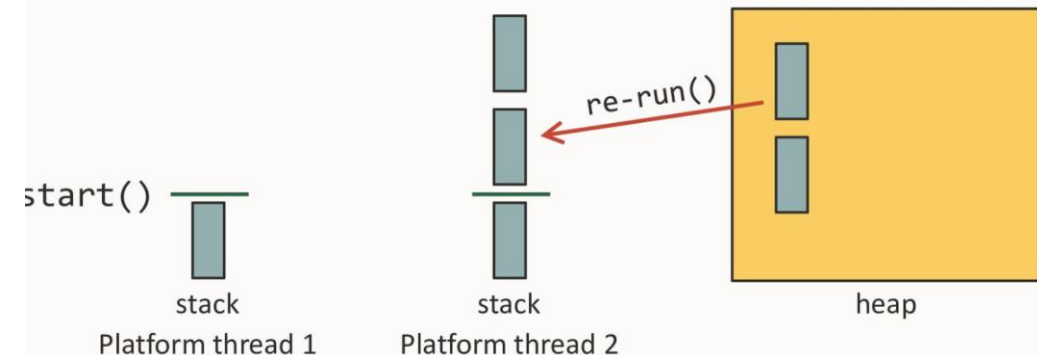
Continuation.yield()

yield() copies the stack to the heap



Continuation.run()

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

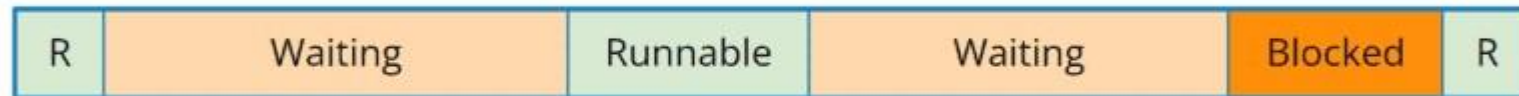




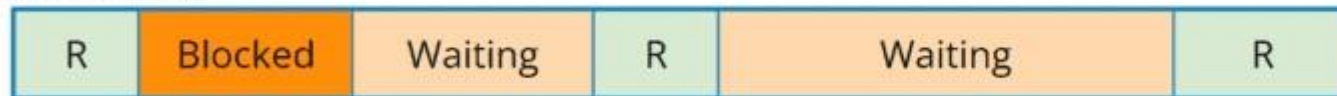
Carrier thread:



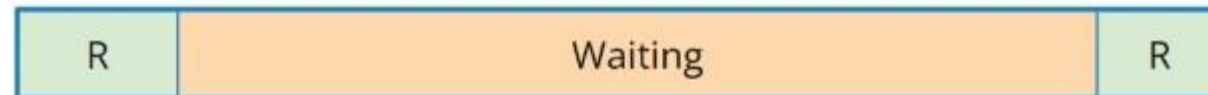
Virtual thread 1:



Virtual thread 2:



Virtual thread 3:





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)
 - When the handler is called, find a carrier thread and called `Continuation.start()`
 - Call `Continuation.yield()`



github.com/openjdk/jdk/pull/8166

openjdk / jdk

<> Code Pull requests 269 Security Insights

8284161: Implementation of Virtual Threads (Preview) #8166

Closed

AlanBateman wants to merge 23 commits into openjdk:master from AlanBateman:JDK-8284161

Conversation 230 Commits 23 Checks 12 Files changed 1,133 +95,870 -8,270

AlanBateman commented on Apr 8, 2022 • edited by openjdk bot

This is the implementation of JEP 425: Virtual Threads (Preview).
We will refresh this PR periodically to pick up changes and fixes from the loom repo.
Most of the new mechanisms in the HotSpot VM are disabled by default and require running with `--enable-preview` to enable.
The patch has support for x64 and aarch64 on the usual operating systems (Linux, macOS, and Windows). There are stubs (calling *Unimplemented*) for zero and some of the other ports. Additional ports can be contributed via PRs against the fibers branch in the loom repo.
There are changes in many areas. To reduce notifications/emails, the labels have been trimmed down for now to hotspot, serviceability and core-libs. We can add additional labels, if required, as the PR progresses.
The changes include a refresh of java.util.concurrent and ForkJoinPool from Doug Lea's CVS. These changes will probably be proposed and integrated in advance of this PR.
The changes include some non-exposed and low-level infrastructure to support the (in draft) JEPs for Structured Concurrency and Extent Locals. This is to make life a bit easier and avoid having to separate VM changes and juggle branches at this time.

Progress

- Change must not contain extraneous whitespace
- Commit message must refer to an issue
- Change requires a CSR request to be approved
- Change must be properly reviewed (1 review required, with at least 1 reviewer)

Issues

- JDK-8284161: Implementation of Virtual Threads (Preview)
- JDK-8284169: Implementation of Virtual Threads (Preview) (CSR)

Contributor

Reviewers

- Imesnik ✓
- egahlin ✓
- djelinski
- mgronlun
- magicus
- sspsysyn ✓
- ExE-Boss
- pron
- turbanoff
- theRealAph
- jalkiran
- shipilev
- bplb ✓
- alexmenkov ✓
- farhankhan04 ✓
- dfuch ✓
- dcubed-ojdk ✓
- mseledts ✓
- coleenp ✓
- tschatzl ✓
- fisk ✓
- stefank ✓



Post



David Delabassée

@delabassee

Implementation of Virtual Threads (Preview) ✓🚧🎉

Stats: 99468 lines in 1133 files changed: 91198 ins; 3598 del; 4672 mod



#Java19 #OpenJDK #ProjectLoom

github.com/openjdk/jdk/co...

6:56 PM • May 7, 2022



WHEN TO USE VIRTUAL THREADS

- Blocking code, many I/O operations. Virtual threads are not intended for long running CPU intensive operations.”
- Many requests that need to be processed asynchronously
- Synchronous code - We can write synchronous/blocking code, without having the decrease in performance or scalability. The cost of blocking a virtual thread is close to zero!
- With Project Loom we might no longer need reactive, non-blocking programming as it is known today.
- Just with usage of virtual thread CPU core become much more concurrent, the combination of virtual threads and multi core CPU with CompletableFutures to parallelized code is very powerful
- Easy to switch from current thread model to virtual threads, the same ExecutorService just different thread pool. Backward compatibility with threads.
- Some frameworks, libraries are already ready, like Spring or Tomcat.



CAUTIONS WHEN USING VIRTUAL THREADS

- When already all CPU is used then it won't speed up application, only make sense in blocking, waiting threads scenario,
- Long tasks in synchronized block can pin virtual thread to the carrier thread, use ReentrantLock instead,
- Native call that does upcall to Java can as well pin thread, problem with some external libraries using native codes: Hadoop, Spark.
- Using Thread Local variables is not recommended, wait for Scoped values – preview future in JDK 20
- It needs latest frameworks and libraries that suport VirtualThreads
- When ExecutorService are used to limit the number of threads that can concurrently access a shared resource, e.g. an I/O device, or a remote web service then it's recommended to use a Semaphore instead.



MORE TO COME

Structured Concurrency (Preview) in JDK 21
Scoped Values (Preview) in JDK 21

LIVE CODING



WORTH TO READ & WATCH

1

Project Loom & JDK spec

<https://www.baeldung.com/openjdk-project-loom>

<https://openjdk.org/projects/jdk/21/>

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

<https://inside.java/>

<https://dev.java/future/innovation/>

<https://openjdk.org/jeps/425> ---virtual threads preview

<https://openjdk.org/jeps/429> ---scoped values

<https://openjdk.org/jeps/428> ---structured concurrency

<https://openjdk.org/jeps/444> - virtual threads GA

2

VirtualThreads in Spring

<https://spring.io/blog/2022/10/11/embracing-virtual-threads>

<https://spring.io/blog/2023/02/27/web-applications-and-project-loom>

<https://github.com/danvega/loom>

3

YouTube

https://www.youtube.com/watch?v=9P9DZCZTq4E&ab_channel=InfoQ

https://www.youtube.com/watch?v=QxxG66eQoTc&ab_channel=IntelliJIDEAbyJetBrains

https://www.youtube.com/watch?v=Is5HXJhC3jE&ab_channel=DanVega





Q&A



**GET THE
FUTURE
YOU WANT**



About Capgemini

Capgemini is a global leader in partnering with companies to transform and manage their business by harnessing the power of technology. The Group is guided everyday by its purpose of unleashing human energy through technology for an inclusive and sustainable future. It is a responsible and diverse organization of over 360,000 team members more than 50 countries. With its strong 55-year heritage and deep industry expertise, Capgemini is trusted by its clients to address the entire breadth of their business needs, from strategy and design to operations, fueled by the fast evolving and innovative world of cloud, data, AI, connectivity, software, digital engineering and platforms. The Group reported in 2022 global revenues of €22 billion.

Get The Future You Want | www.capgemini.com



This presentation contains information that may be privileged or confidential and is the property of the Capgemini Group.

Copyright © 2023 Capgemini. All rights reserved.