



holisticon

JavaLand 2025

# Loom und Virtual Threads

War es das jetzt mit Reactive Programming?

Jan Weinschenker · 2. April 2025

# Agenda

- Einleitung
- Über Loom
- Threads und Nebenläufigkeit
- Blockierte Threads
- Loom im Detail
- Fragen

# Einleitung



# Transform today

Holisticon AG  
Hamburg

Holisticon ist eine Technologieberatung, die Strategie, Organisation und Technologie von Beginn an konsequent ganzheitlich denkt und steuert – und dadurch die Digitalisierung innovativer Geschäftsmodelle zukunftssicher ermöglicht.

17+

Jahre

70+

Berater\*Innen

250+

Kunden

850+

Projekte

200+

Conference-Talks



# Jan Weinschenker

Consultant, Developer, Architect, Nerd @ Holisticon

- Entwickelt Software seit fast 18 Jahren
- Java, Kotlin, Spring, ...
- Großer Fan von Reactive Frameworks \*



# 3 Dinge über Loom



# Project Loom

JDK Enhancement Proposal 425

*„Project Loom aims to drastically reduce the effort of writing, maintaining, and observing high-throughput concurrent applications that make the best use of available hardware.“*

Ron Pressler (Tech lead, Project Loom)

[https://cr.openjdk.org/~rpressler/loom/loom/sol1\\_part1.html](https://cr.openjdk.org/~rpressler/loom/loom/sol1_part1.html)

# Project Loom

JDK Enhancement Proposal 425

*„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.“*

<https://openjdk.org/jeps/425>

# Project Loom

JDK Enhancement Proposal 425

*„I think Project Loom is going to kill Reactive Programming.“*

*Brian Goetz (Java Language Architect at Oracle)*

<https://www.youtube.com/watch?v=9si7gK94gLo&t=1367s>

# 3 Dinge über Loom

- *Nebenläufigkeit soll einfacher werden.*
- *Virtual Threads*
- „*Kill reactive programming*“

*Loom wurde veröffentlicht mit Java 21 am 19. September 2023*

# 3 Dinge über Reactive Programming

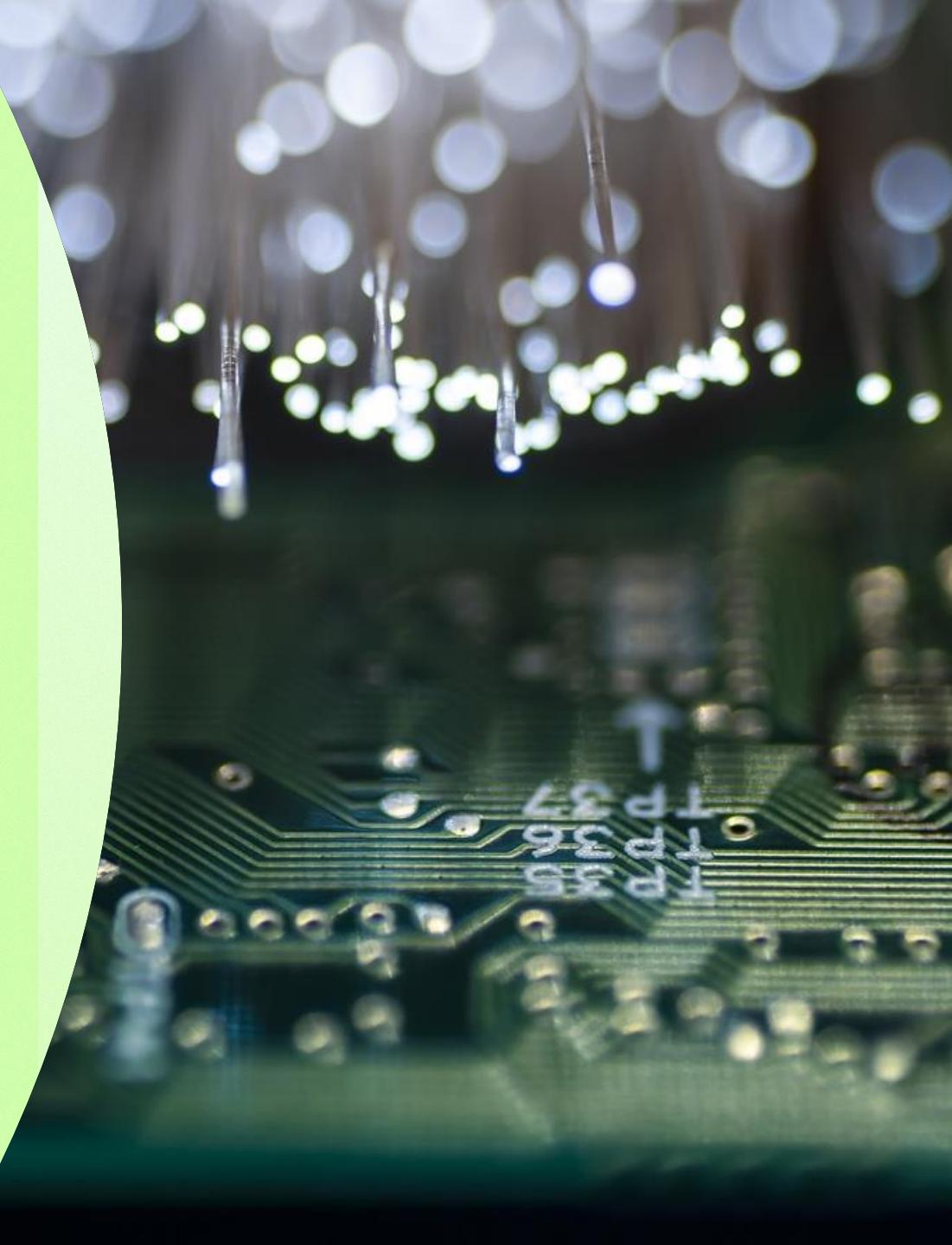
- *Sehr effiziente Ressourcennutzung*
- *Kann Nebenläufigkeit seit über 10 Jahren\**
- *Reakives Programmiermodell (Fluch und Segen)*

\* Spring WebFlux Version 1.0 wurde im November 2013 veröffentlicht.

# War es das jetzt für Reactive Programming?

- Nebenläufigkeit?
- Ressourcennutzung?
- Programmiermodell?

# Threads und Nebenläufigkeit





# One Thread = One Request

JVM-Threads sind die Arbeitspferde in unseren Anwendungen.

- OS stellt Threads bereit
- Threads können nebenläufig arbeiten

# Thread Limit vs CPU Limit

- Betriebssystem stellt einige tausend Threads bereit.
- CPU Leistung im GHz-Bereich.
- Threads sind Performance Bottlenecks



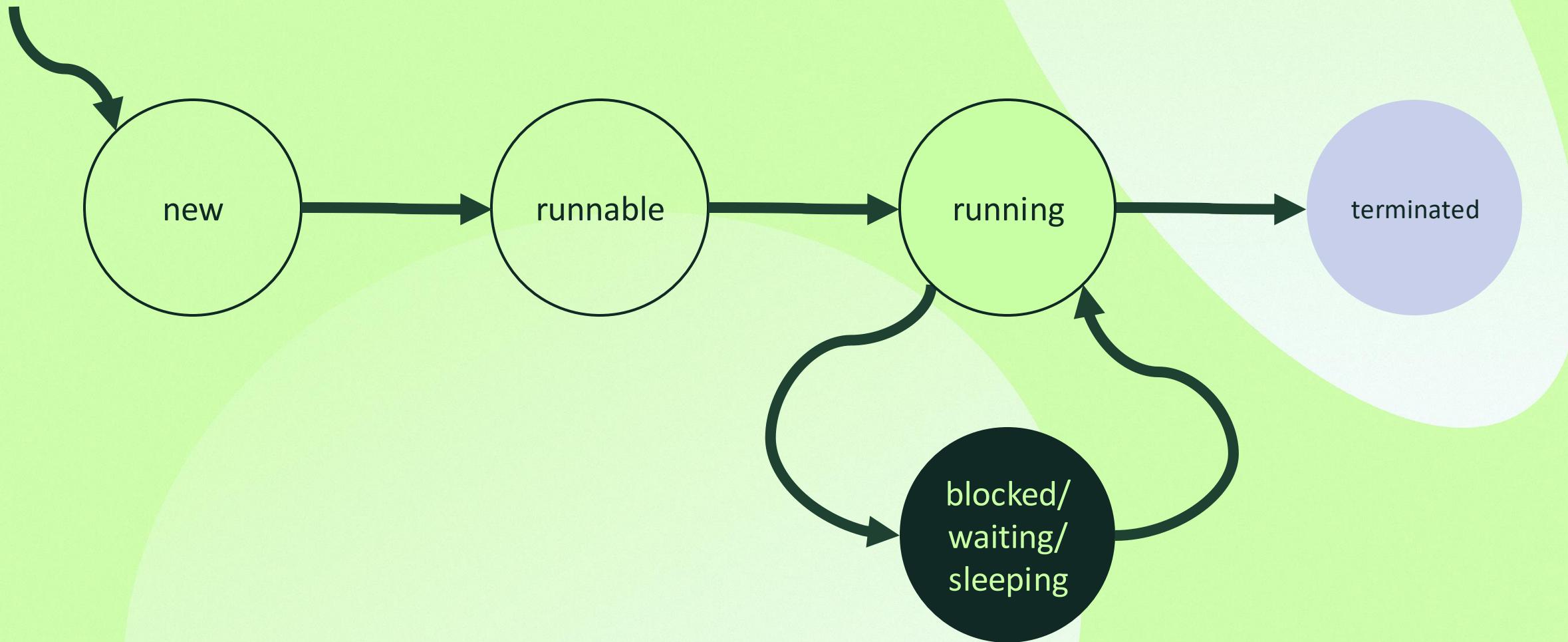
# One Thread = One Request

- Es gibt immer wartende oder blockierte Threads
- CPU wird nicht effektiv genutzt
- CPU wartet

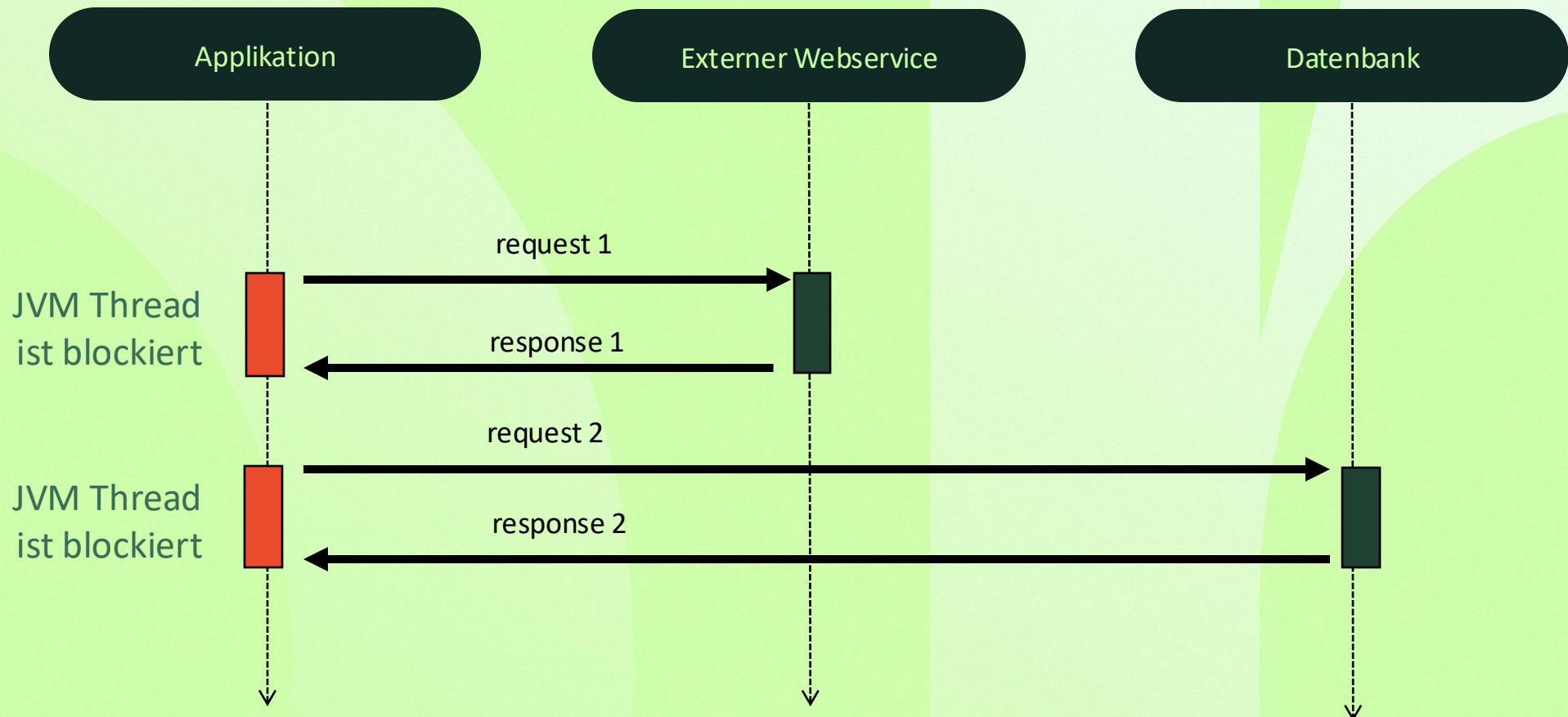
# Blockierte Threads überwinden



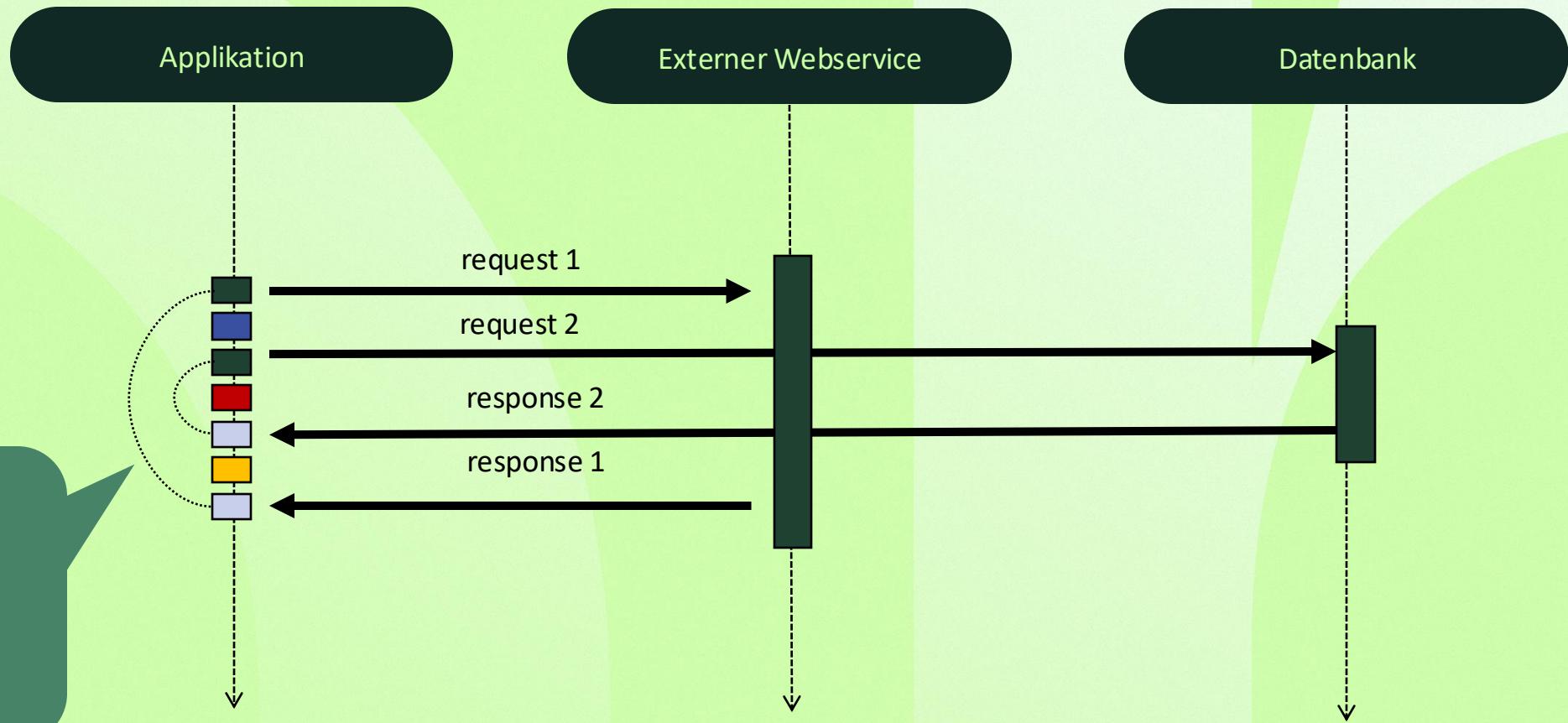
# Status von JVM Threads



# Blockierte Threads



# Reaktiver Ansatz



# Wie wird Software „Non-Blocking“?

- Idealerweise:

**Unlimited Threads – Jeder Task im eigenen Thread.**

- Aber Threads sind eine begrenzte Ressource!
- Wie verteilt man unendlich viel Arbeit auf endlich viele Threads?

# Wie wird Software „Non-Blocking“?

Lösungsansätze:

- Reactive programming (Seit 2013)
- Coroutines
- Seit 2023: Virtual Threads

# Reaktive Frameworks

- Keine blockierten Threads mehr
- Reaktives Programmiermodell
- Nicht einfach 😞



spring® WebFlux



QUARKUS

VERT.X

# Reaktive Java Frameworks (RF)

- Reactive Streams (2013)
- **WebFlux / Project Reactor (Nov 2013)**
- Vert.x (May 2013)
- RxJava (May 2013)
- Helidon (Sept 2018)
- Mutiny (Dec 2019)

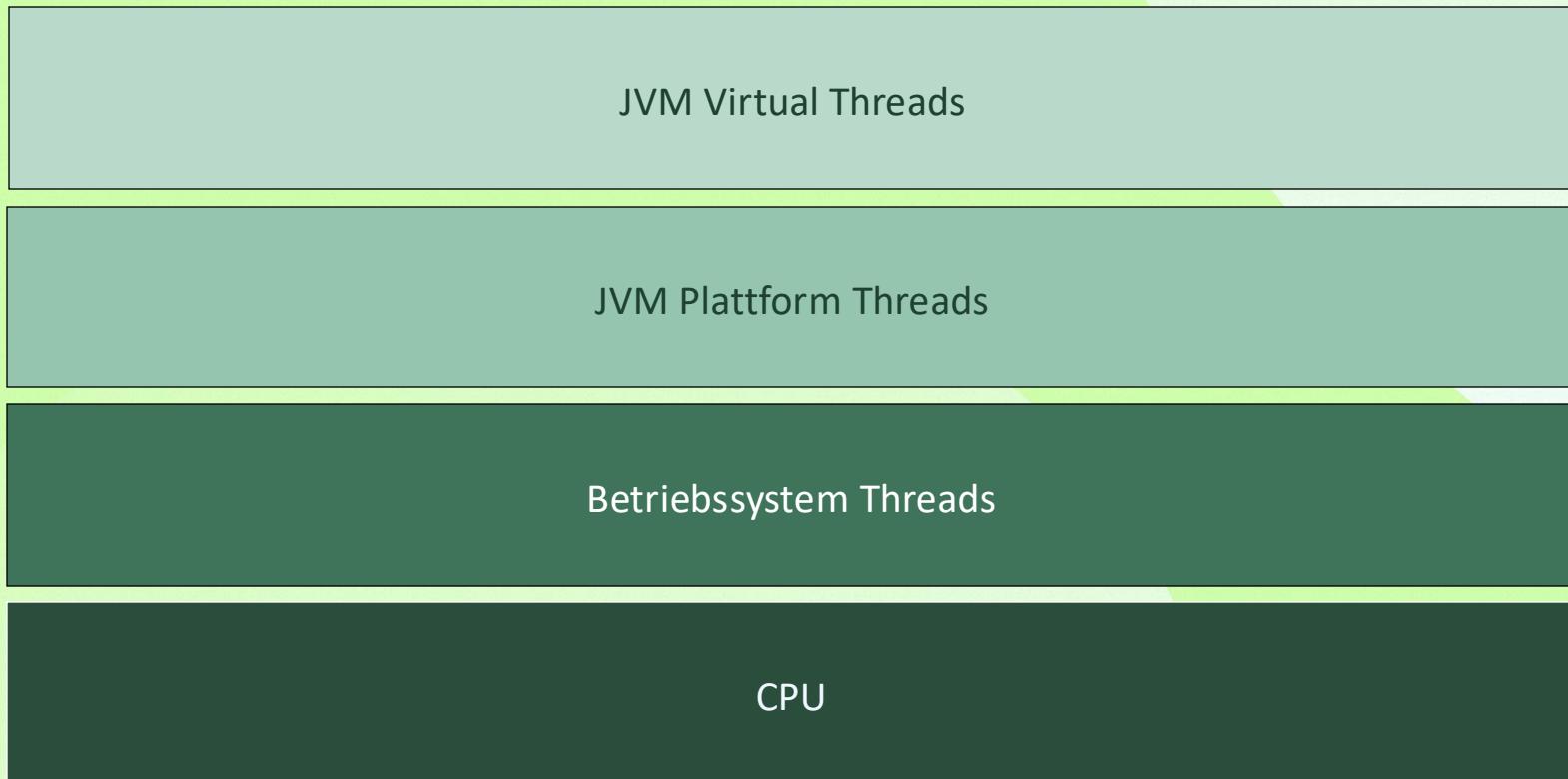
# Codebeispiele

- Zähle von 0 bis 1.000.000
- 10 Sekunden Verzögerung beim Zählen
- Verwende Nebenläufigkeit

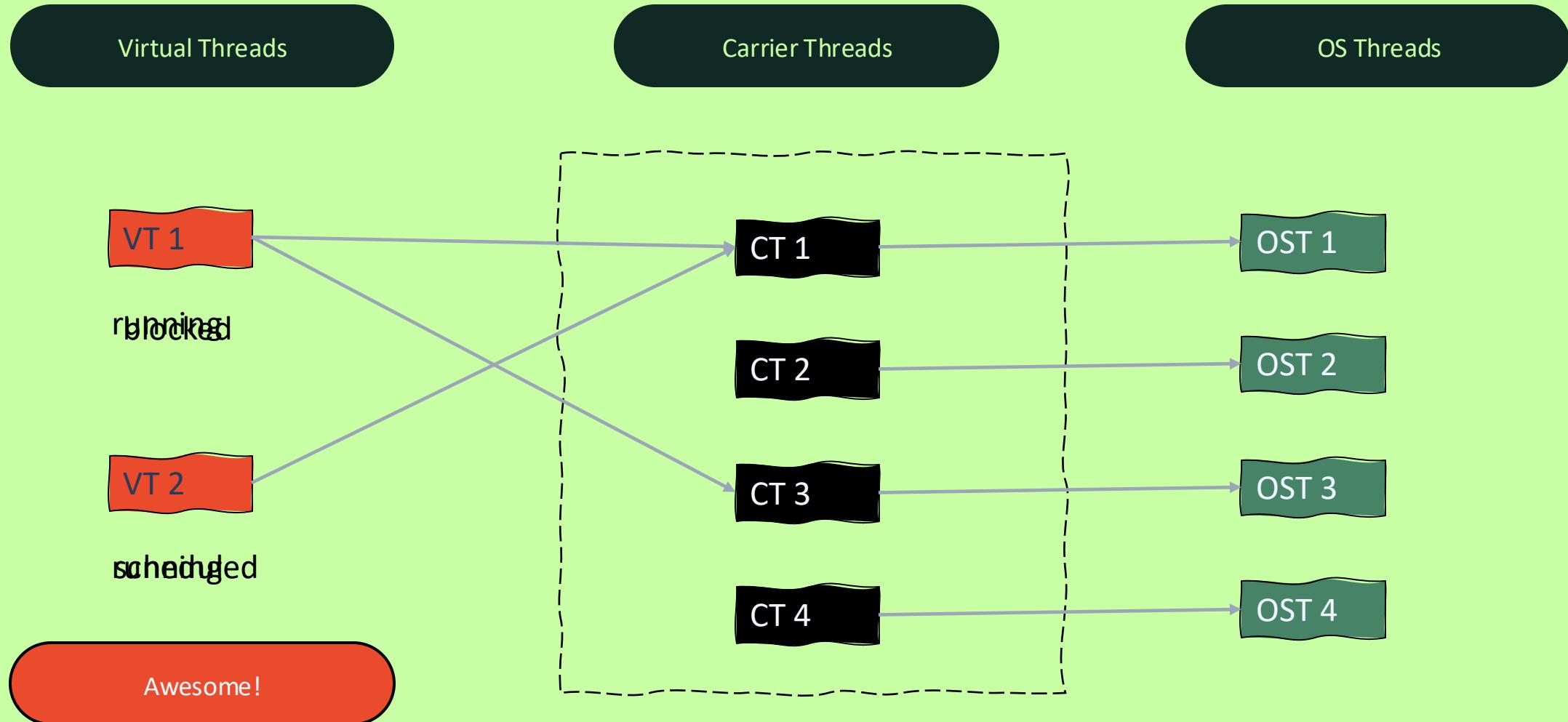


# Los geht's ...

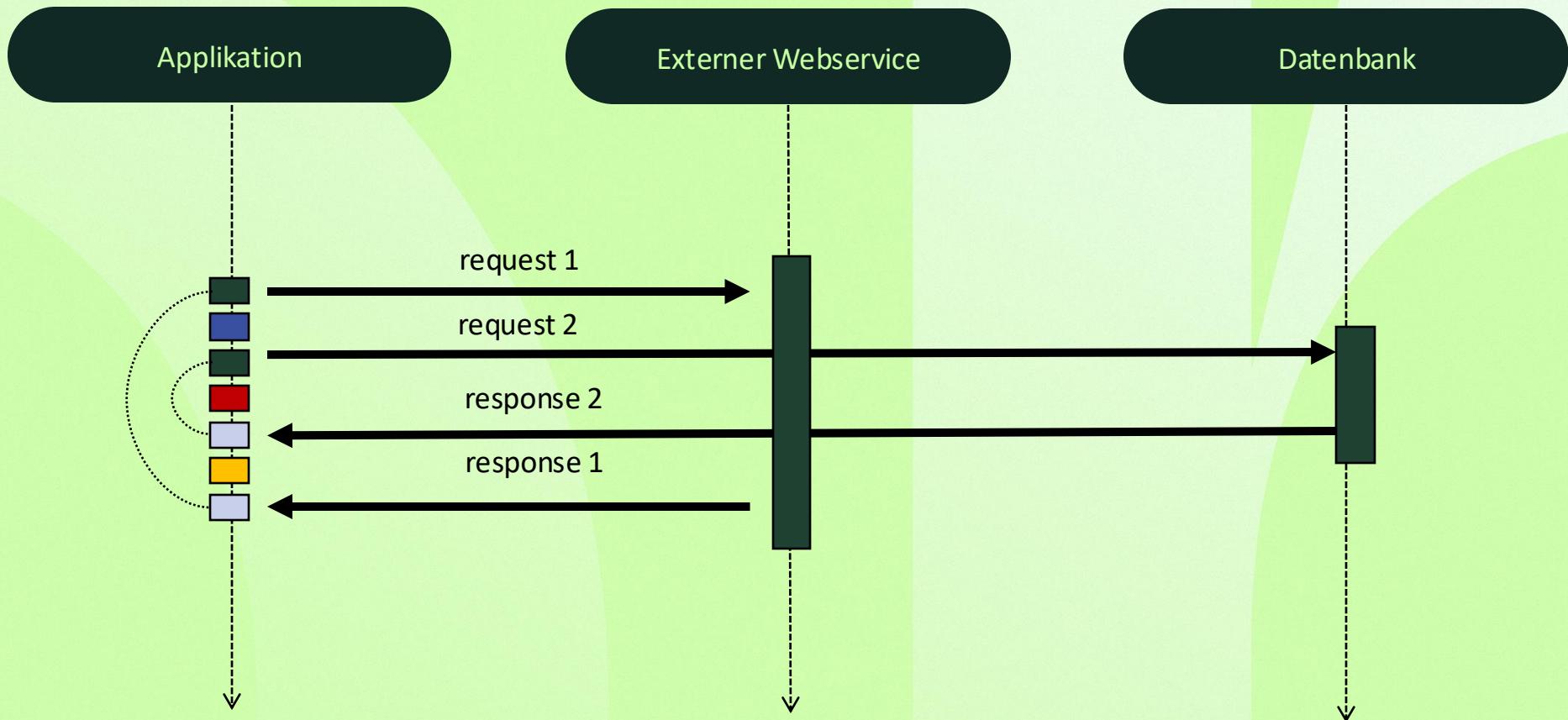
# Looms Virtual Threads



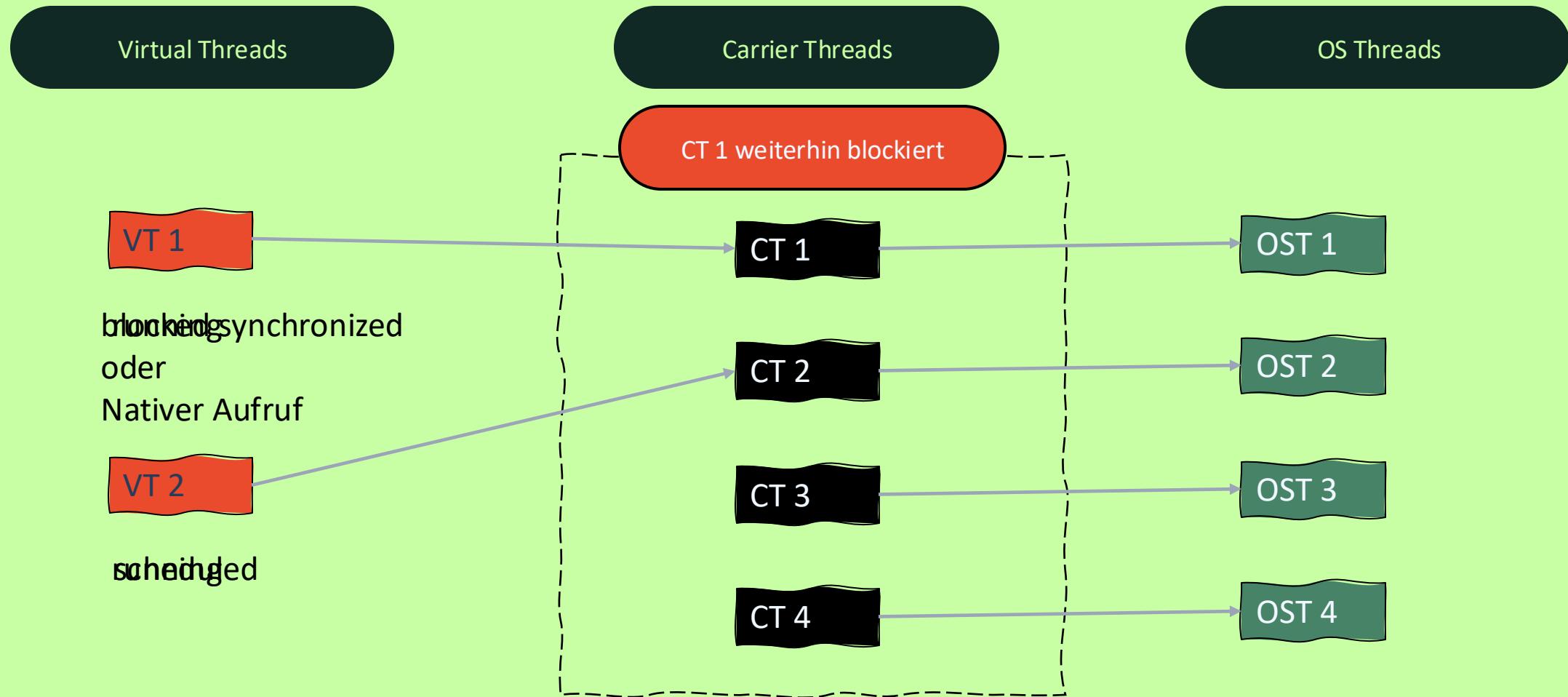
# Looms Virtual Threads



# Reaktiver Ansatz



# Looms Virtual Threads - Pinned



# Looms Virtual Threads - Pinned

- Virtual Threads können fest an Plattform Threads gebunden („Gepinnt“) werden, wenn
  - der VT einen `synchronized` Block ausführt (bis Java 23) \*
  - Java nativen Code aufruft.
  - blockierender Code in einem Class Initializer ausgeführt wird
- siehe: <https://openjdk.org/jeps/491>

# Looms Virtual Threads - Pinned

- Gepinnte Virtual Threads verhalten sich wie Plattform Threads.
- Zu viele gepinnte Threads sind ein Problem.
- Siehe <https://netflixtechblog.com/java-21-virtual-threads-dude-wheres-my-lock-3052540e231d>

# ThreadLocal

- Virtual Threads unterstützen ThreadLocal Variablen.
- Achtung beim Speicherbedarf!
  - Virtual Threads sind billig und schnell zu erzeugen.
  - Speicher kann mit ThreadLocals „überschwemmt“ werden.

# Structured Concurrency

- Reaktive Frameworks haben eine dedizierte DSL für Nebenläufigkeit.
- Virtuelle Threads haben so etwas nicht.
- Preview Feature: Structured Concurrency

# Fazit

|                        | Loom  | Reactive                        |
|------------------------|---|---------------------------------|
| Blocking Code          | Handhabung durch JVM                        | In Verantwortung der Entwickler |
| Nebenläufigkeit        | Threads<br>Structured Concurrency (Preview) | Inherent im Programmiermodell   |
| ThreadLocal            | unterstützt                                 | Nicht unterstützt               |
| Programmier-<br>modell | Klassisch imperativ                         | Reaktiv                         |

## Mein Fazit (2)

- Loom: Non-blocking gibt es geschenkt
- „Try the easy stuff first before doing something weird.“  
(quoting Brian Goetz)
- Reactive Programming ist immer noch super für ereignisbasierte Anwendungsfälle.

Danke für Eure  
Aufmerksamkeit 😊



# Fragen?



Source Code

und

Folien

Auf GitHub

