Java Virtual Threads

13. April 2022

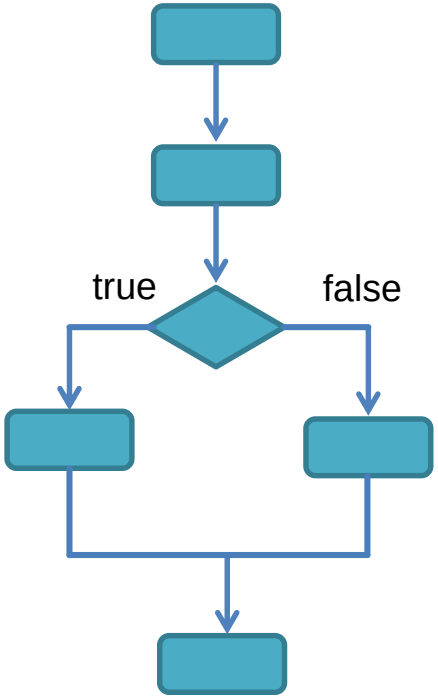
Jörg Hettel
Hochschule Kaiserslautern

Agenda

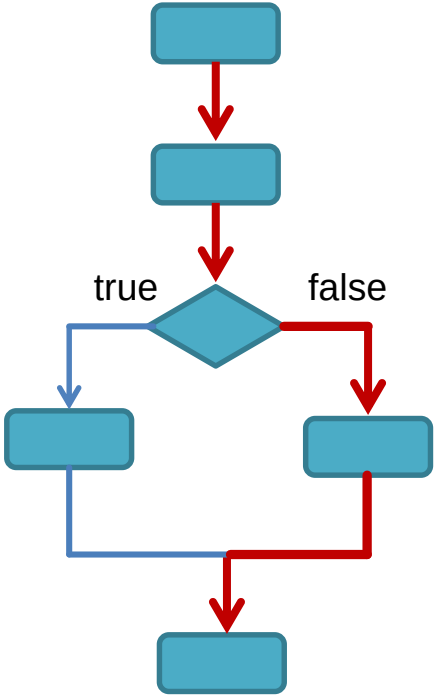
Hochschule Kaiserslautern
University of Applied Sciences

- Thread-Konzept (von Java)
- Konzept der Coroutine
- Virtual Threads

Modell eines Programmablaufs

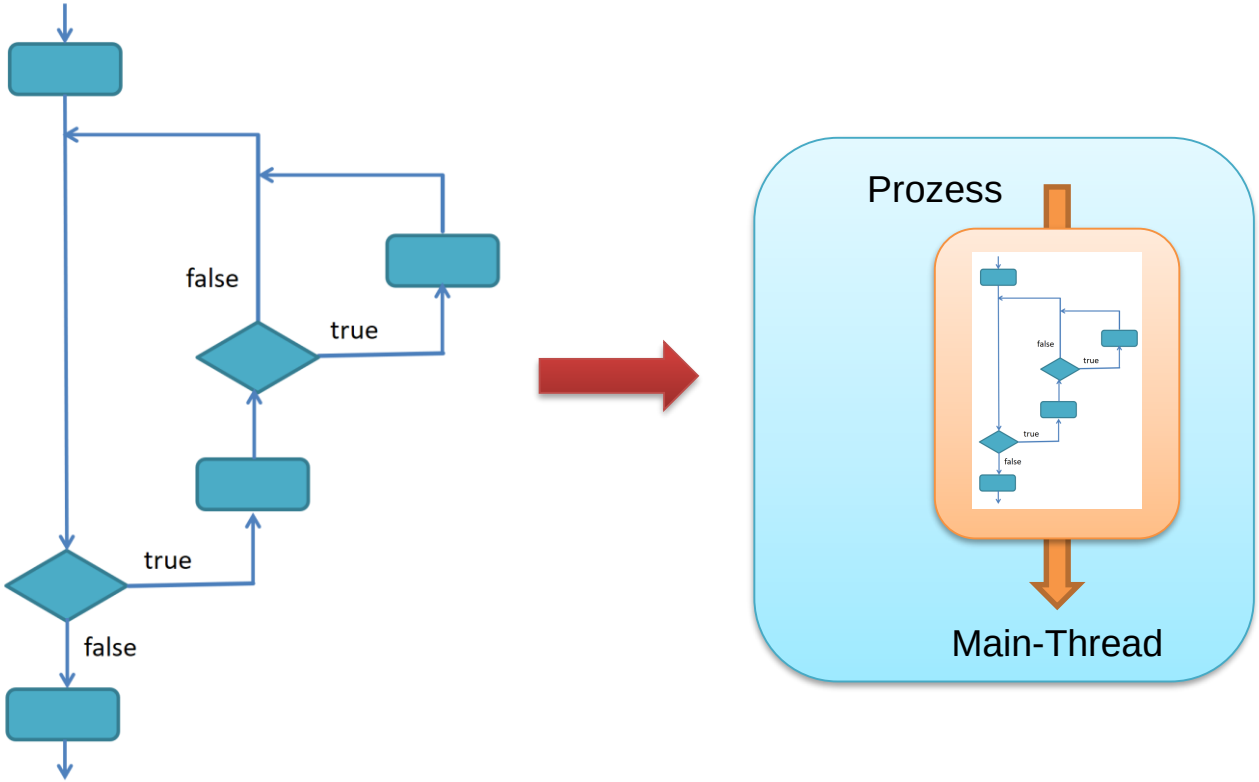


Kontrollflussgraph

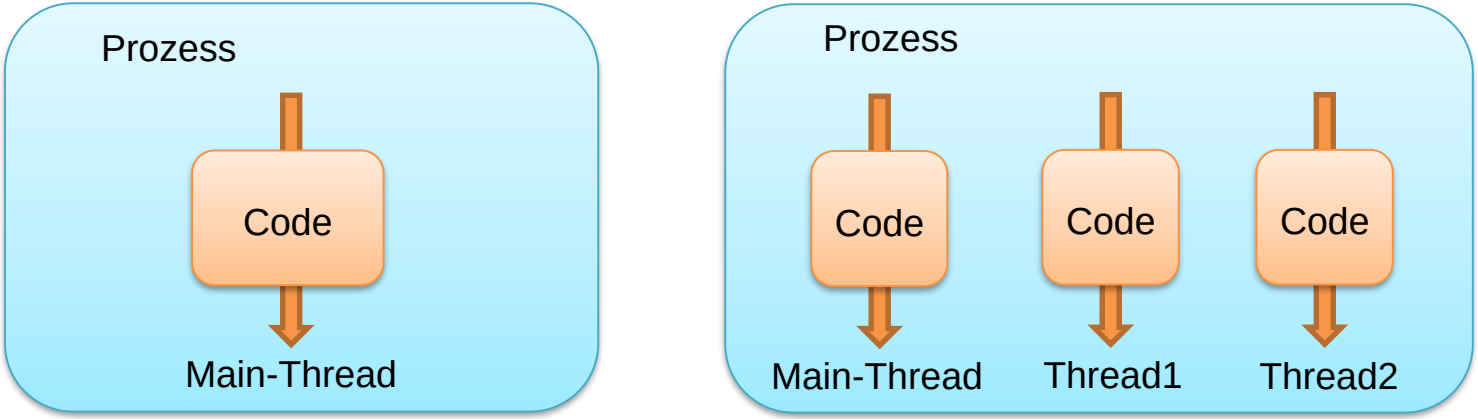


Ausführungspfad

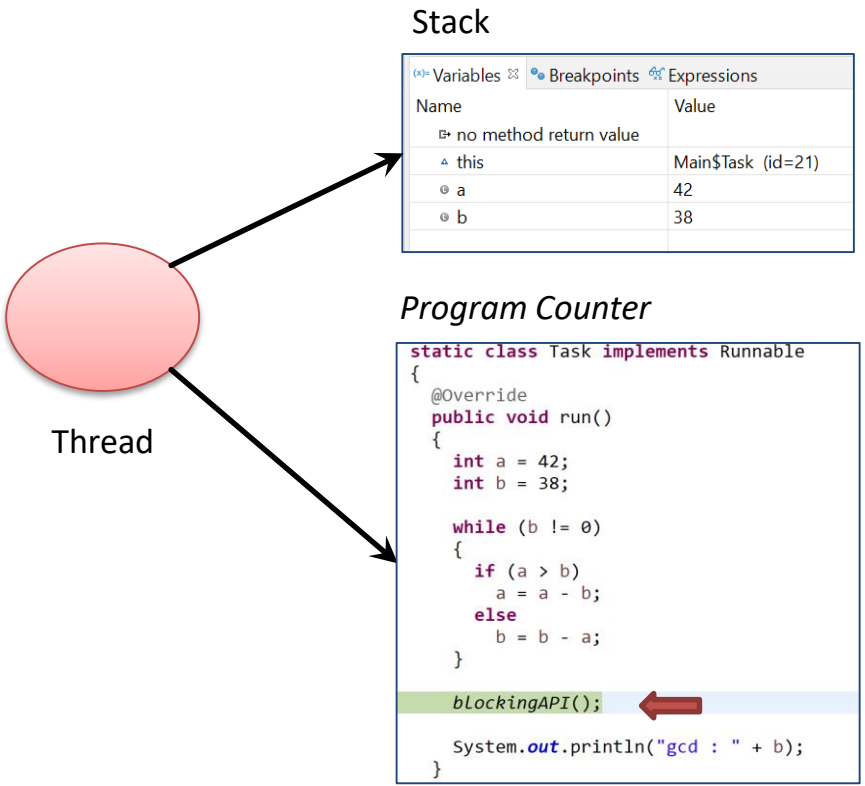
Thread-Konzept



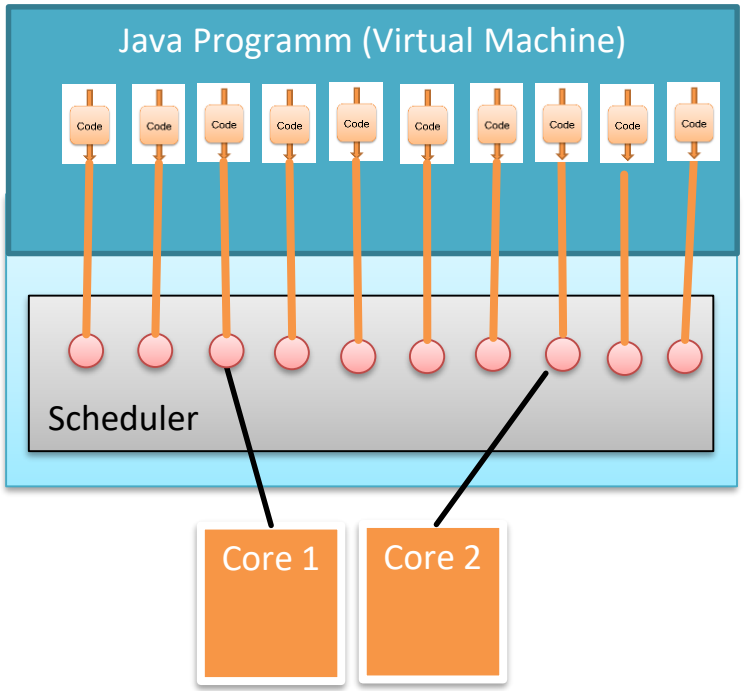
Idee: Multithreading



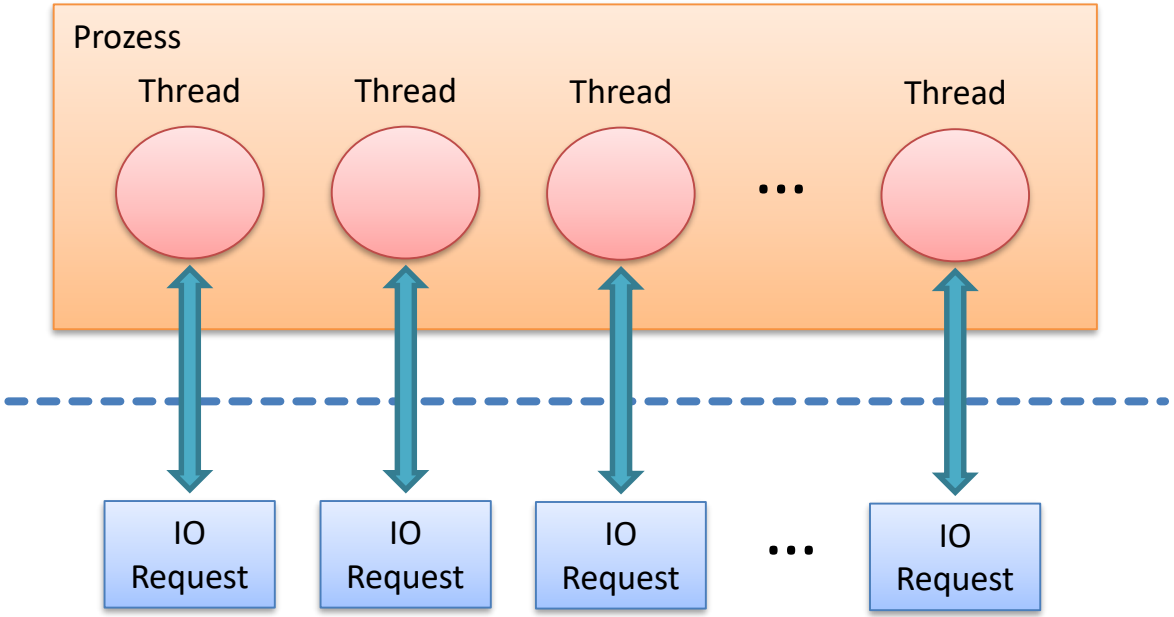
Daten eines Threads (Stack)



Technische Umsetzung (heute)



Problem: Wartende Threads

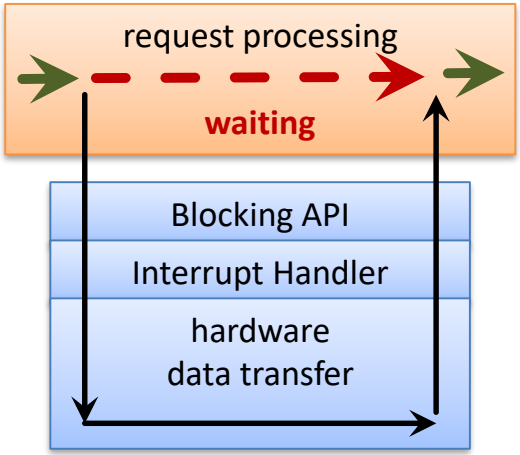


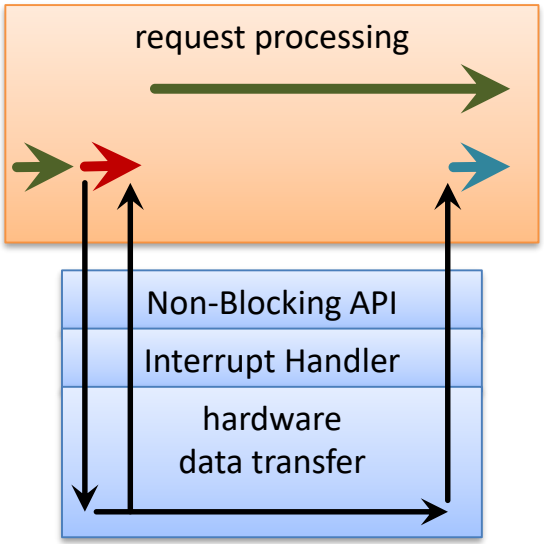
Limitierungen

Hochschule Kaiserslautern
University of Applied Sciences

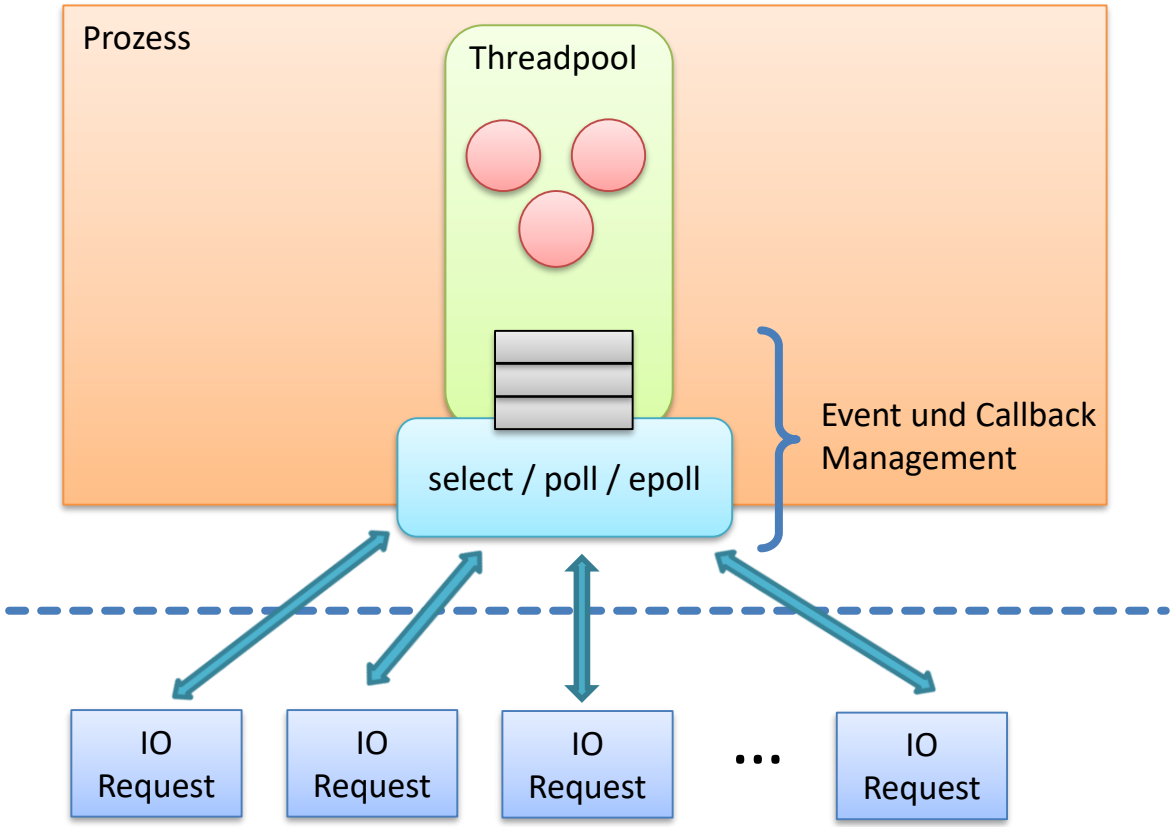
- Thread ist eine "teure" Ressource
- Skalierungsbeschränkung
- Performance-Bottleneck

Asynchrone APIs: Funktionsweise





Asynchrone APIs



Paradigmen

- Programmiermodell für synchronen blockierenden Code
- Programmiermodell für asynchronen nicht-blockierenden Code

JEP 425: Virtual Threads (Preview)

https://openjdk.java.net/jeps/425

JEP 425: Virtual Threads (Preview)

Authors	Ron Pressler, Alan Bateman
Owner	Alan Bateman
Type	Feature
Scope	SE
Status	Candidate
Component	core-libs
Discussion	loom dash dev at openjdk dot java dot net
Effort	XL
Reviewed by	Alex Buckley, Brian Goetz
Created	2021/11/15 16:43
Updated	2022/04/07 10:38
Issue	8277131

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.

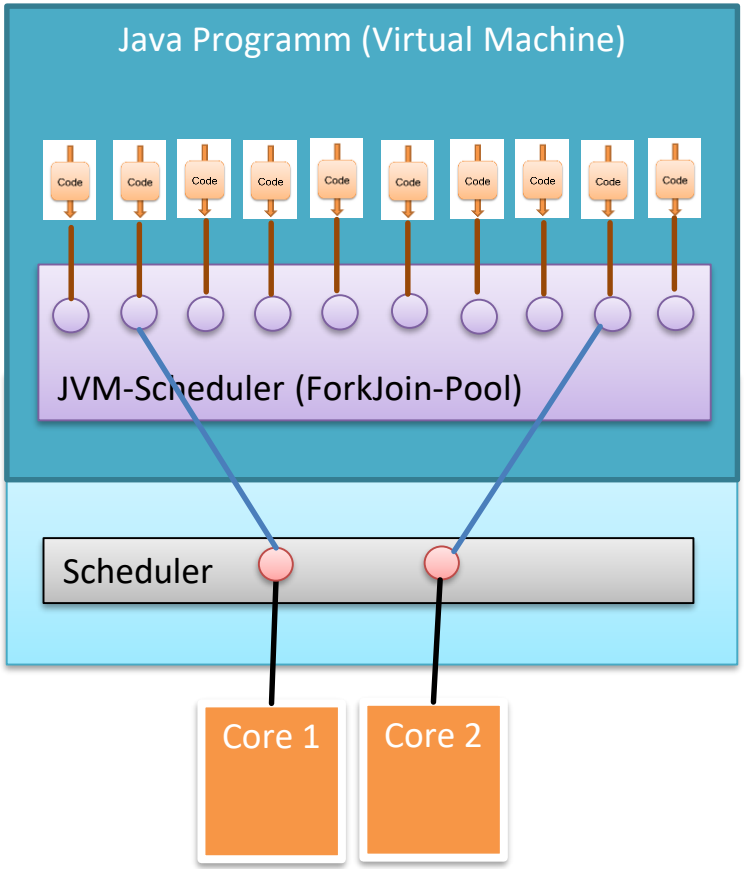
JEP draft: Structured Concurrency (Incubator)

Owner	Ron Pressler
Type	Feature
Scope	SE
Status	Draft
Component	core-libs
Created	2021/11/15 15:01
Updated	2022/04/02 08:27
Issue	8277129

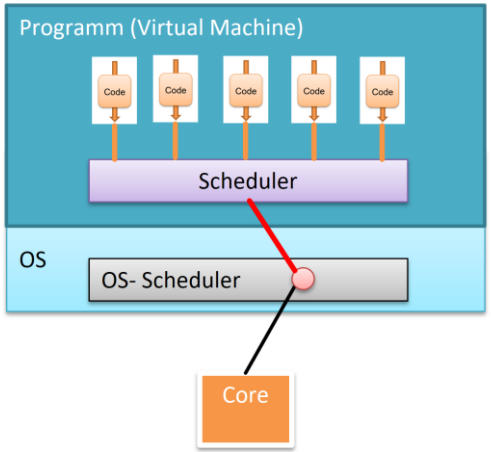
Summary

Simplify concurrent programs by adding elementary library support for structured concurrency, a simple principle that states that when the flow of execution splits into multiple concurrent flows, they rejoin in the same code block. That discipline makes dealing with failures, cancellation, and deadlines in concurrent code more manageable, as well as provides the runtime with useful information about the relationships among concurrent tasks that can be reflected by observability tools.

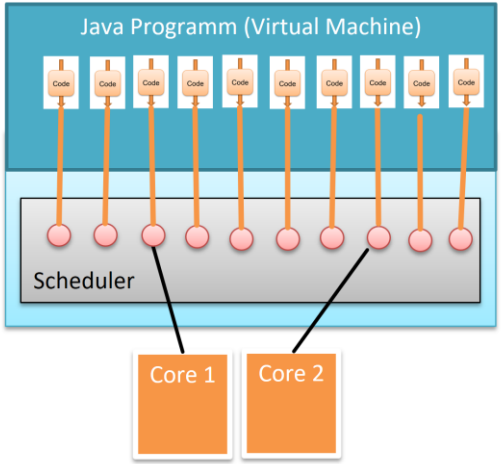
Virtual Thread Scheduling



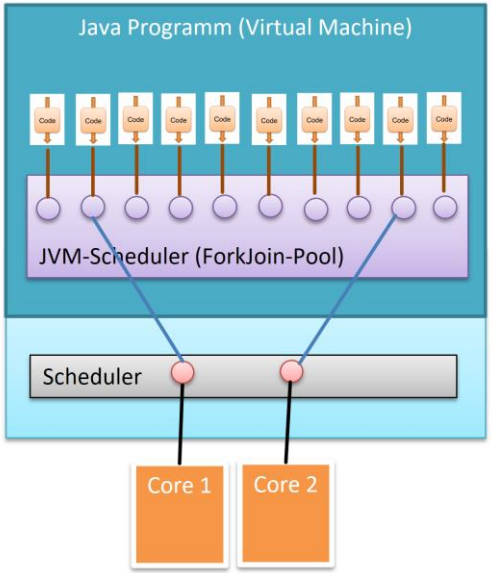
Technische Umsetzungen



Anfänge: Green Threads
Model: **n:1**

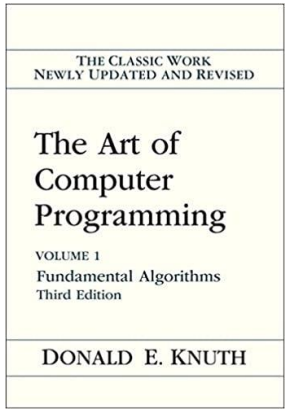


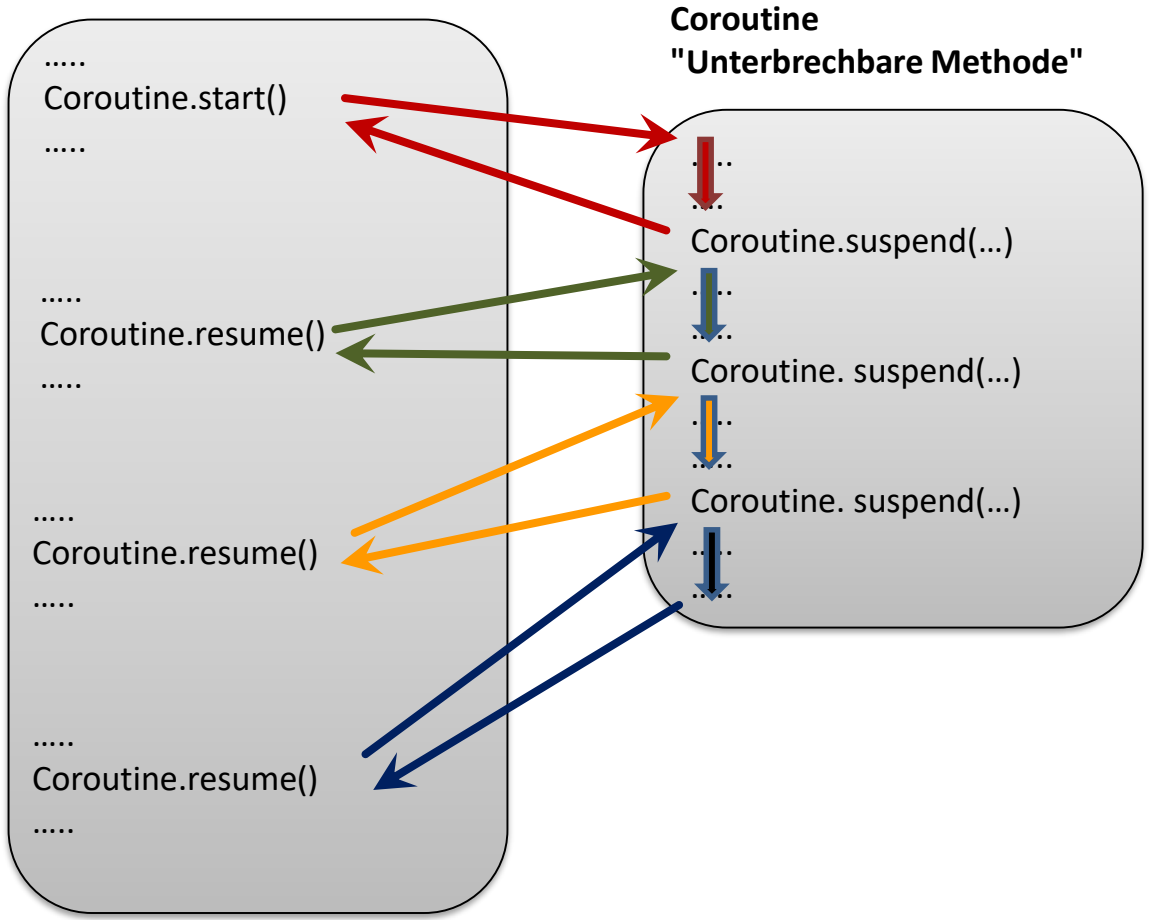
Aktuell: Plattform-Threads
Model: **1:1**



Virtuelle Threads
Model: **n:m**

Umsetzung: Coroutinen-Prinzip





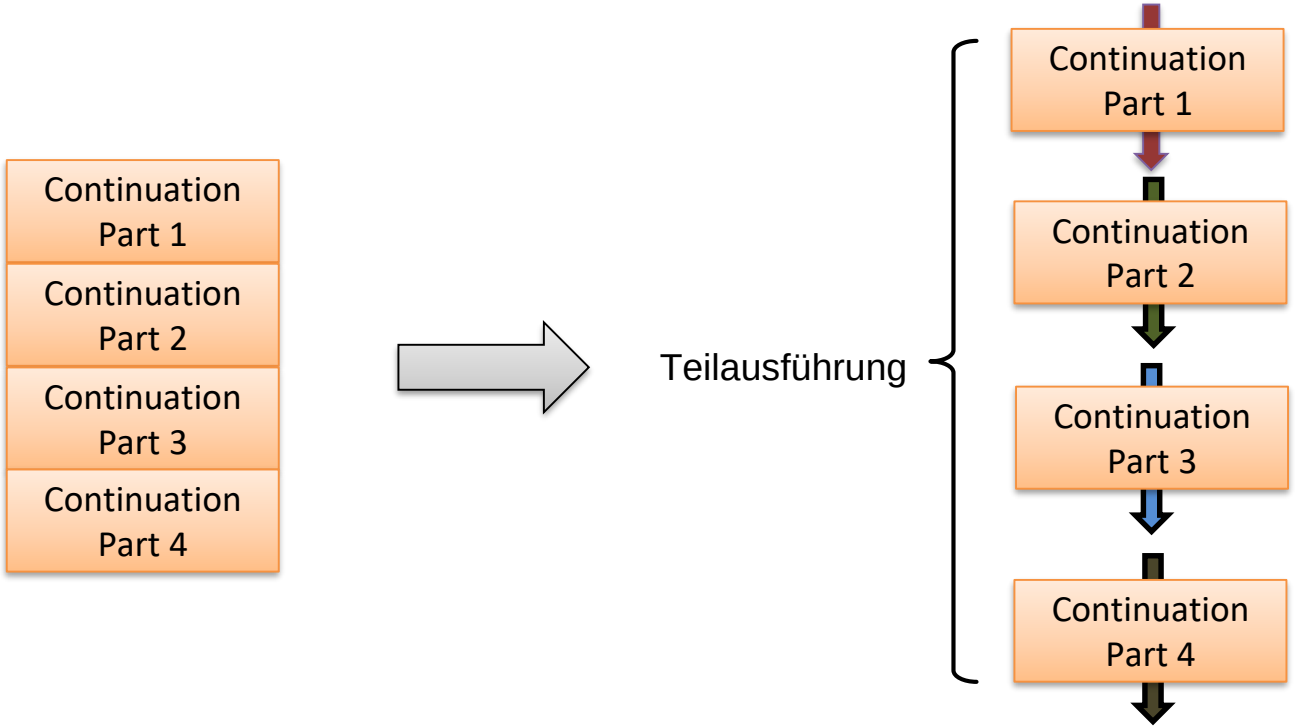
Code-Beispiel

Umsetzung bei Java: *Continuation*

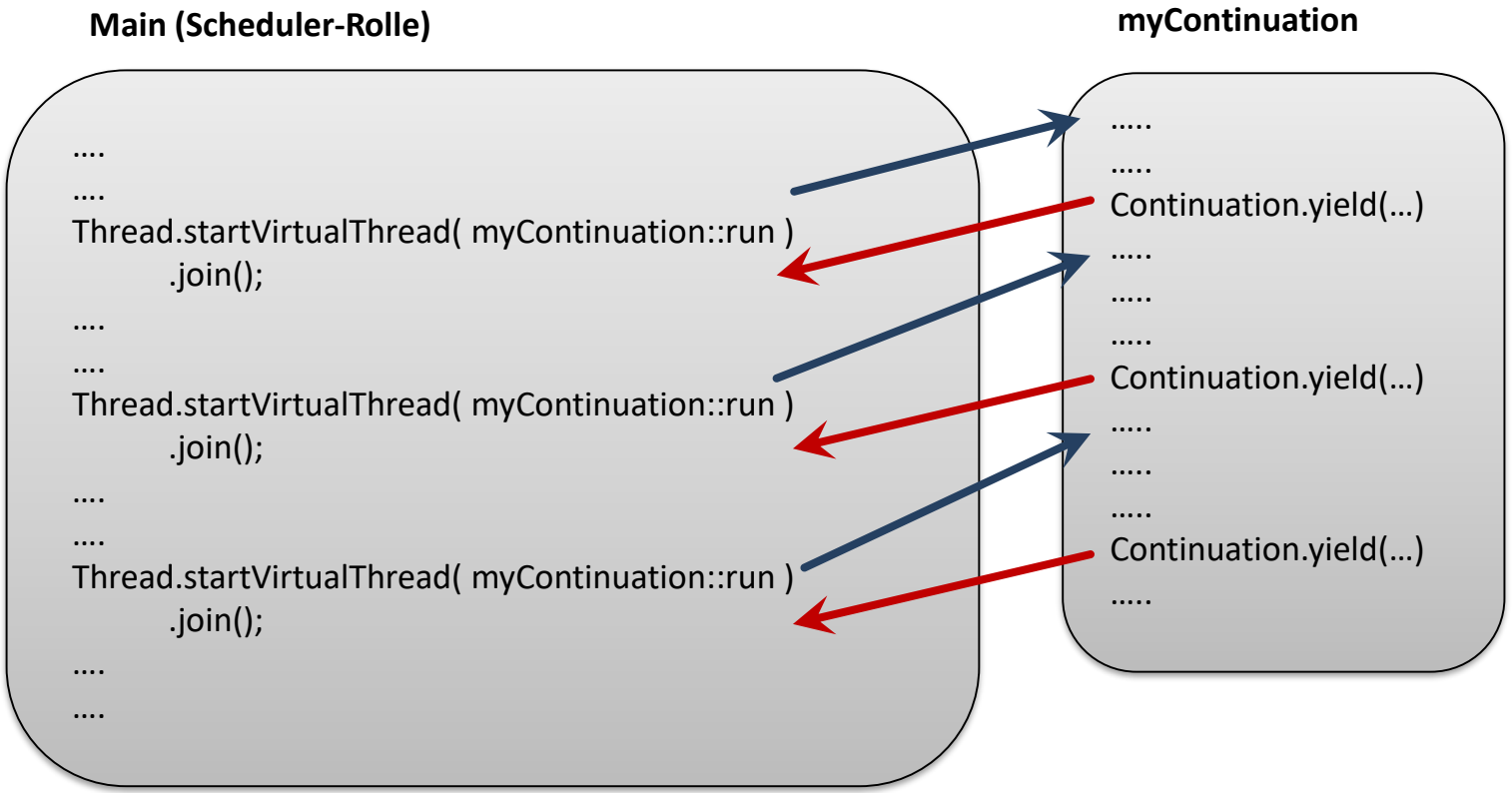
Nicht mehr Bestanteil des offiziellen Preview-APIs

Demo 1,2,3: "Ausführung" einer Continuation

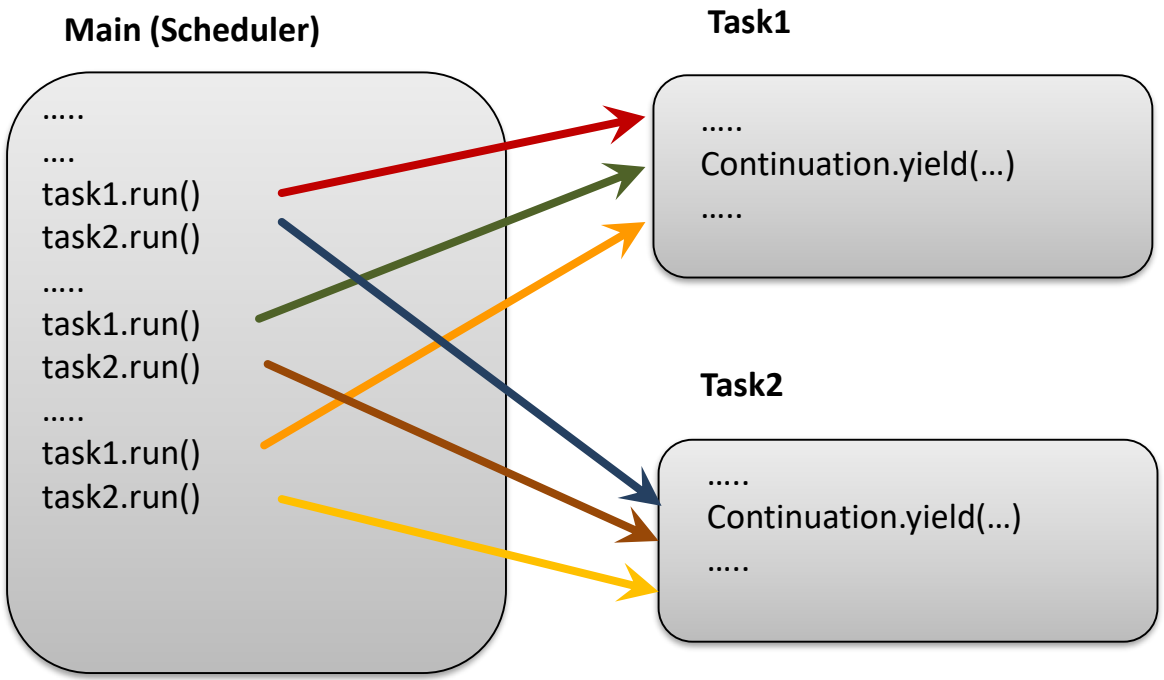
- **Continuation besitzt einen eigenen "Stackframe"**
  - **Wird auf dem Heap abgelegt**



Demo 3: "Ausführung" einer Continuation

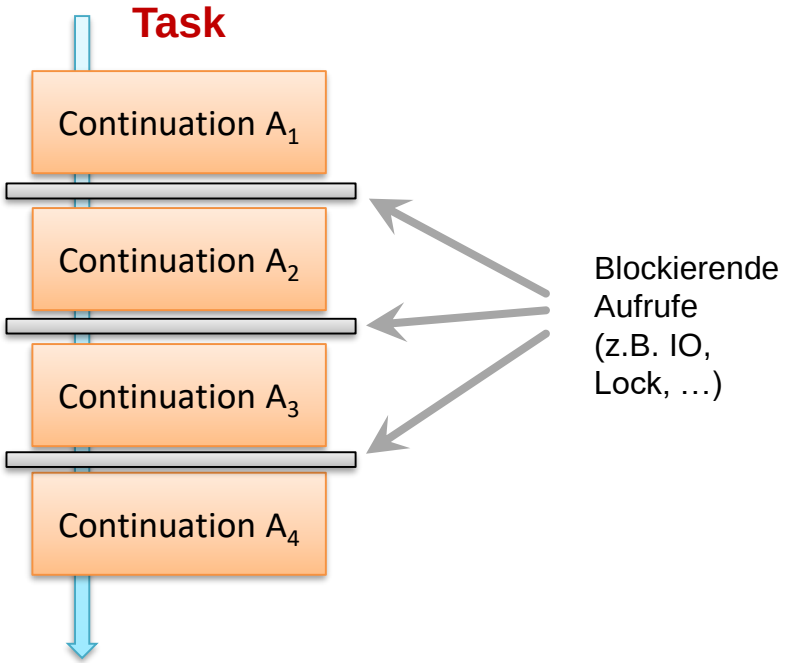


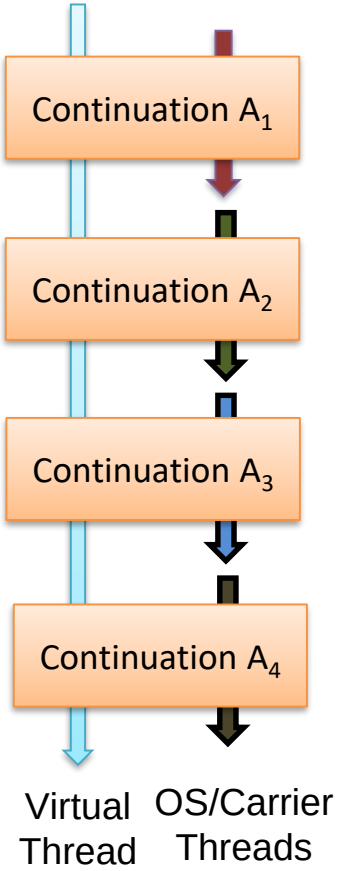
Demo 4: Umgang mit Tasks



Umgang mit blockierenden Aufrufen

- Mounting und Unmounting des virtuellen Threads auf bzw. von seinem Carrier-Thread





Interna

Beispiel Klasse **LockSupport**-Methode (Paket `java.util.concurrent.locks`)

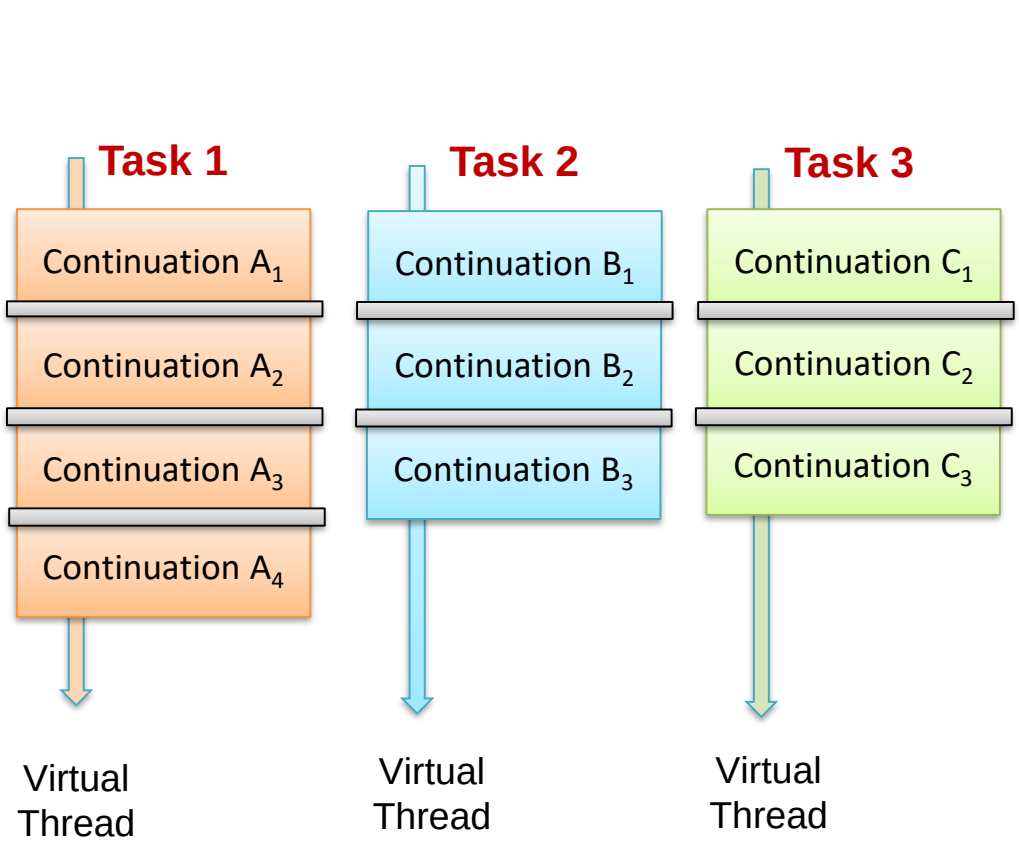
```
public static void park()
{
      if (Thread.currentThread().isVirtual())
      {
            VirtualThreads.park();
      }
     else
      {
            U.park(false, 0L);
      }
 }
```

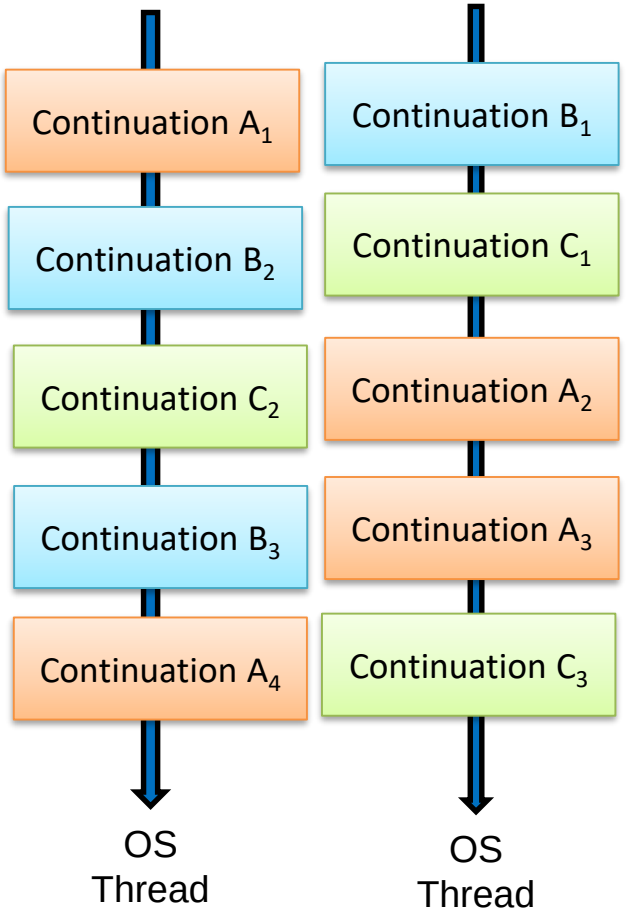
"Virtual Thread Friendly" Klassen

https://wiki.openjdk.java.net/display/loom/Blocking+Operations

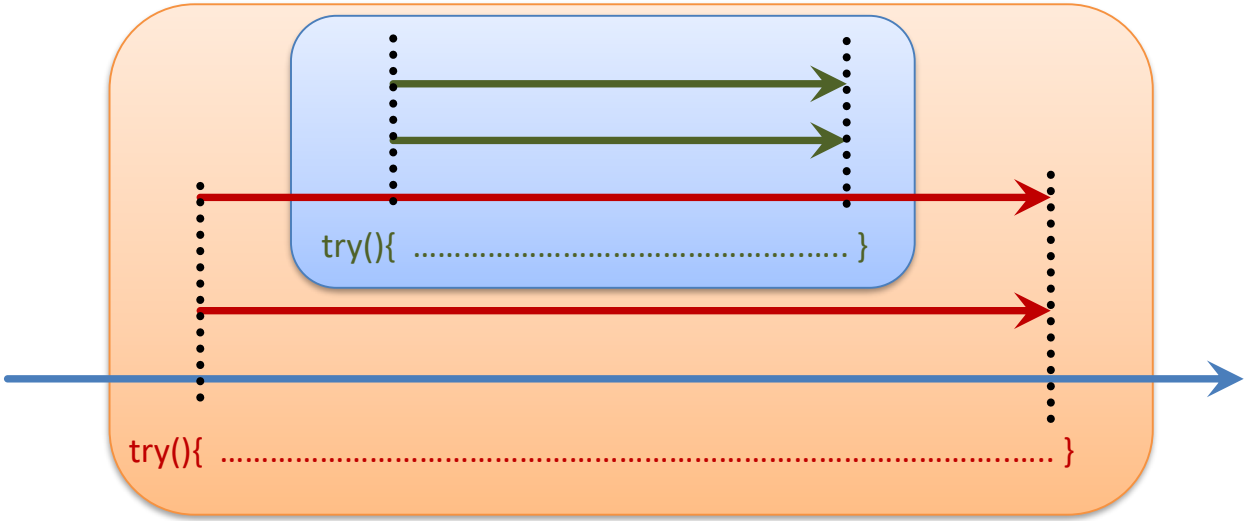
API	Method(s)	Notes
java.lang.Thread	sleep, join	join to wait for a virtual thread to terminate
java.lang.Process	waitFor	Linux/macOS only
java.util.concurrent	All blocking operations	
java.net.Socket	connect, read, write	Socket constructors with a host name parameter may need to do a lookup with InetAddress, see below
java.net.ServerSocket	accept	
java.net.DatagramSocket/MulticastSocket	receive	connect, disconnect and send do not block
java.nio.channels.SocketChannel	connect, read, write	
java.nio.channels.ServerSocketChannel	accept	
java.nio.channels.DatagramChannel	read, receive	connect, disconnect, send, and write do not block
java.nio.channels.Pipe.SourceChannel	read	
java.nio.channels.Pipe.SinkChannel	write	
Console streams (System.in, out, err)	read, write, printf	Linux/macOS only

Skalierung mit Continuations





Demo 5: Structured Concurrency



Demos

"Preview JDK 19"

- Thread-Erzeugung
- Kleinste gemeinsame Zahl
- Game of Life

Early Access

Aktueller Early Access: https://jdk.java.net/loom/

jdk.java.net

GA Releases
JDK 18
JDK 17
JMC 8

Early-Access Releases
JDK 19
Loom
Metropolis
Panama
Valhalla

Reference Implementations
Java SE 18
Java SE 17
Java SE 16
Java SE 15
Java SE 14
Java SE 13
Java SE 12
Java SE 11
Java SE 10
Java SE 9
Java SE 8
Java SE 7

Project Loom Early-Access Builds

These builds are intended for developers looking to "kick the tyres" and provide feedback on using the API or by sending bug reports.

Warning: This build is based on an incomplete version of JDK 19.

Build 19-loom+5-429 (2022/4/4)

These early-access builds are provided under the GNU General Public License, version 2, with the Classpath Exception.

Linux/AArch64	tar.gz (sha256)	192429496 bytes
Linux/x64	tar.gz (sha256)	193655045
macOS/AArch64	tar.gz (sha256)	188249612
macOS/x64	tar.gz (sha256)	190335614
Windows/x64	zip (sha256)	192675142

Documentation

- API Javadoc

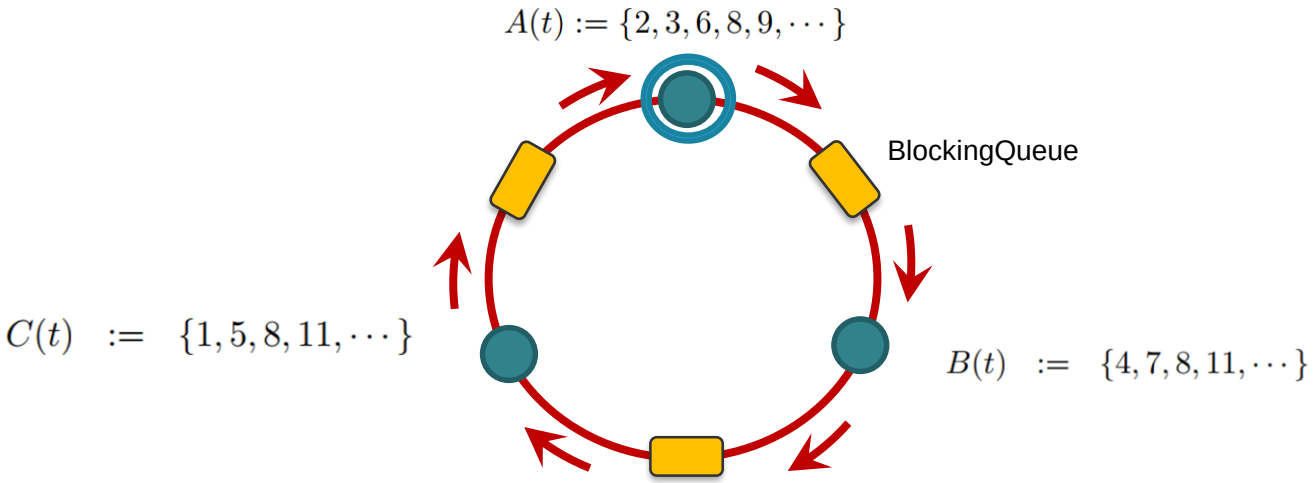
Einfluss auf das Anwendungsdesign

- **Bestimme den frühesten Zeitpunkt für ein Treffen, der für alle Mitglieder eines Teams passt.**
  - **Entspricht: Finden die kleinste gemeinsame Zahl in einer Anzahl von Listen**

$$
\begin{aligned}
A(t) &:= \{2, 3, 6, 8, 9, \cdots\} \\
B(t) &:= \{4, 7, 8, 11, \cdots\} \\
C(t) &:= \{1, 5, 8, 11, \cdots\}
\end{aligned}
$$

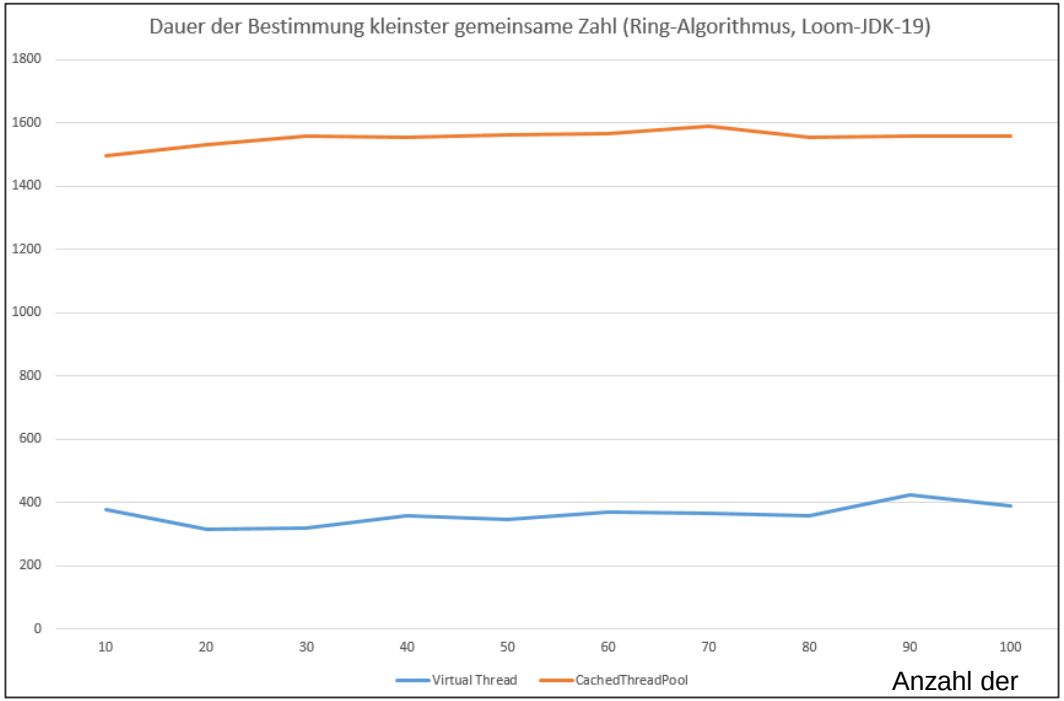
Einfluss auf das Anwendungsdesign

- **Eine mögliche Lösung (Ring-Token)**
  - **Jedes Teammitglied wird durch einen "Thread" repräsentiert**
  - **Es gibt einen Leader**



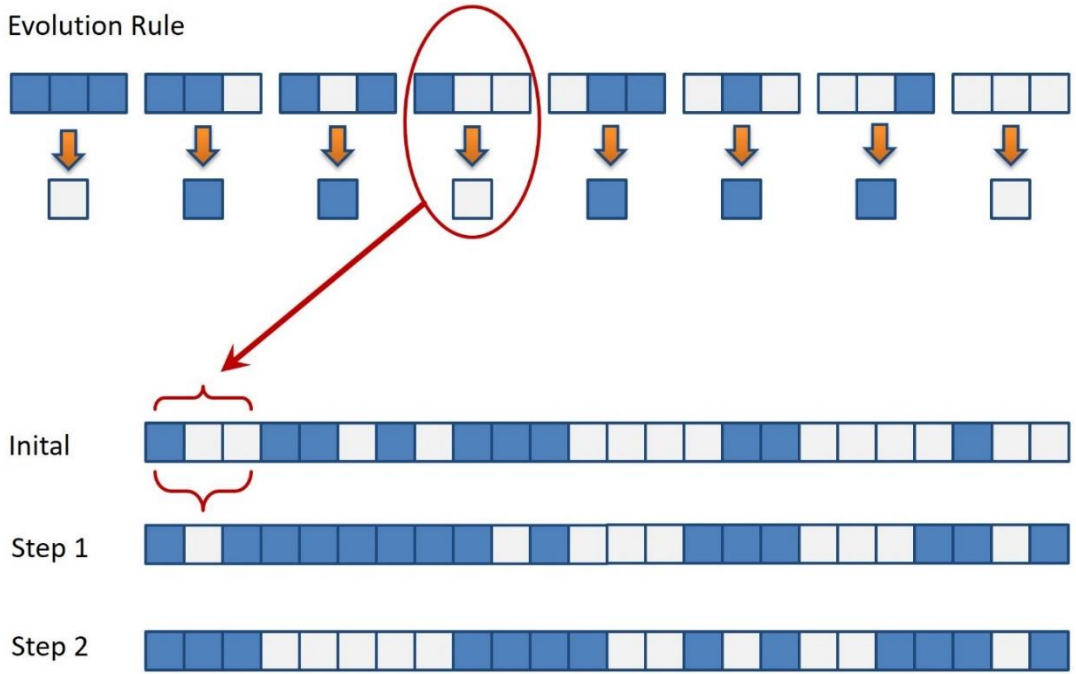
Einfluss auf das Anwendungsdesign

Laufzeitvergleich: Lösung durch Ring-Kommunikation



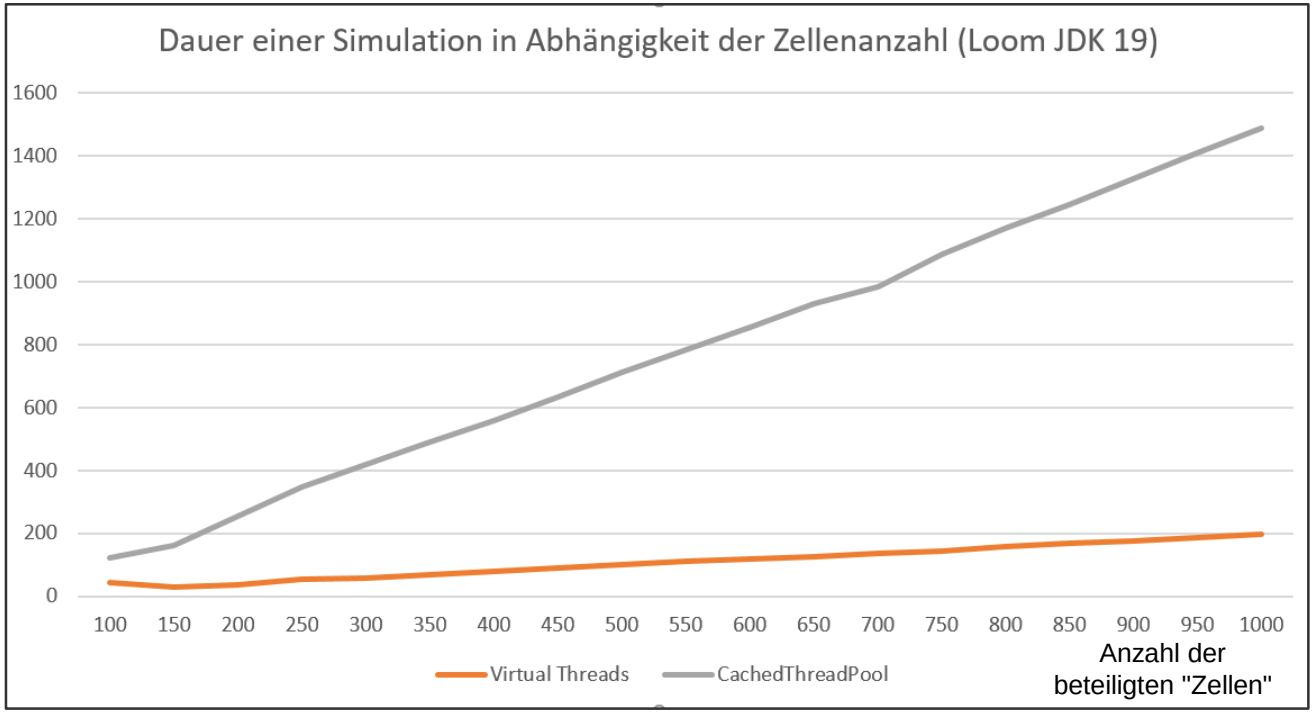
Einfluss auf das Anwendungsdesign

Game of Life



Einfluss auf das Anwendungsdesign

Game of Life: Skalierungsvergleich



Anwendungsfälle

Hochschule Kaiserslautern
University of Applied Sciences

- CPU-Bound
  - Ausnutzung von Rechenleistung durch Parallelisierung
  - "gewöhnliche Threads",
- IO-Bound
  - Viele blockierende Operationen (im Hintergrund)
  - Virtual Threads

Weitere Konzepte (Vorschau)

```
Carrier<String> carrier = new Carrier<>();

Thread producer = Thread.startVirtualThread(() -> {
  Carrier.Sink<String> sink = carrier.sink();
  sink.send("message1");
  sink.send("message2");
  sink.closeExceptionally(new InternalError());
});

Thread consumer = Thread.startVirtualThread(() -> {
  try (Carrier.Source<String> source = carrier.source()) {
    while (true) {
      String message = source.receive();
      System.out.println(message);
    }
  } catch (IllegalStateException e) {
    System.out.println("consumer: " + e + " cause: " + e.getCause());
  }
});

producer.join();
consumer.join();
```

Zusammenfassung und Ausblick

Hochschule Kaiserslautern
University of Applied Sciences

- Thread-Konzept (von Java)
  - (Internes) Konzept der Coroutine
- Virtual Thread Konzept für die asynchrone Programmierung ohne "Callbacks"
  - "Alter Code" bleibt kompatibel

- Noch anstehende Herausforderungen
  - Vernünftige Unterstützung von Debuggern und Profilern
    - Z.T. gelöst
  - Interaktion mit Build-In-Synchronization (Monitor-Konzept)
  - Aufruf von Native-Code

Vielen Dank und gibt es Fragen?

Kontakt:

Jörg Hettel

joerg.hettel@hs-kl.de

Hochschule Kaiserslautern
University of Applied Sciences