Java Virtual Threads

07. Juli 2022

Jörg Hettel
Hochschule Kaiserslautern

Agenda

Hochschule Kaiserslautern
University of Applied Sciences

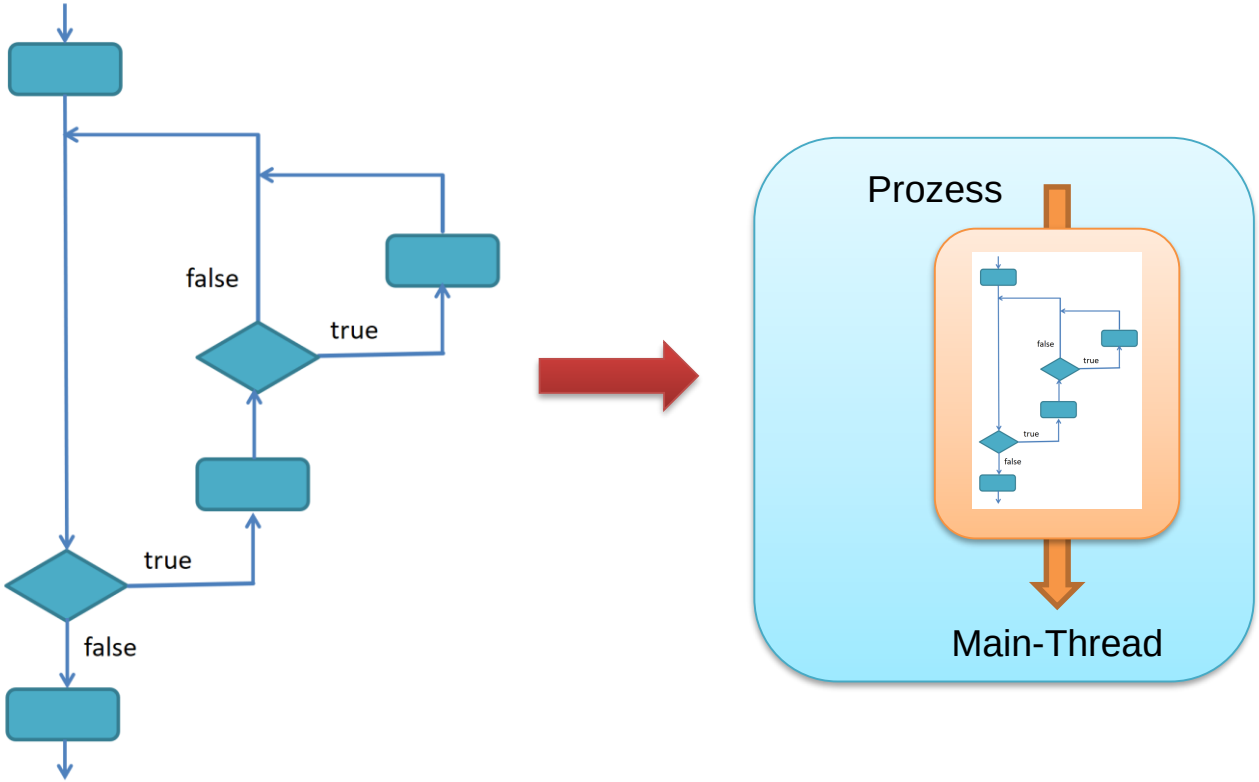
- Thread-Konzept (von Java)
- Konzept der Coroutine/Continuation
- Virtual Threads
- Ausblick

Begriffe

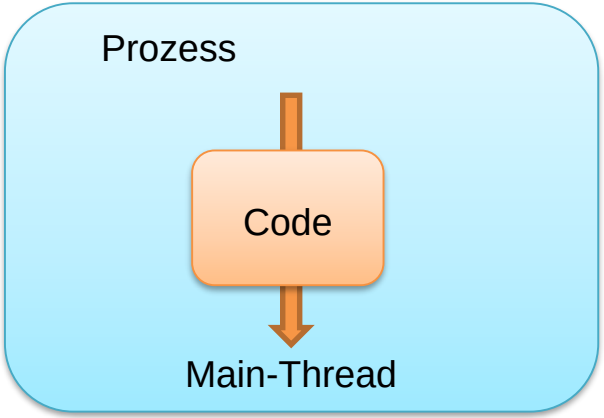
Hochschule Kaiserslautern
University of Applied Sciences

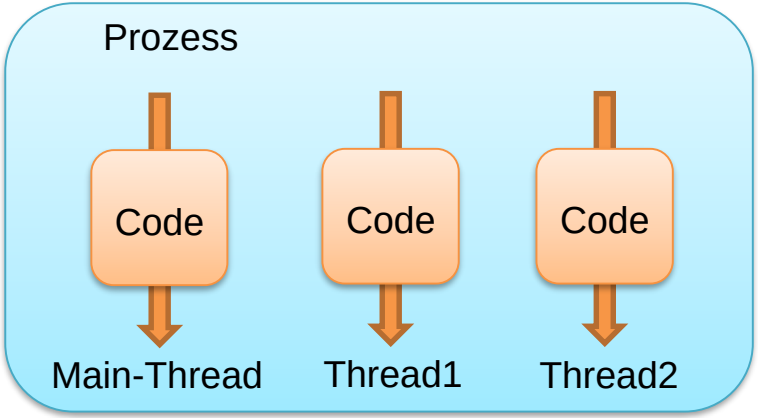
- **Parallel (kooperativ)**
  - Eine Aufgabe wird gemeinsam von vielen Helfern erledigt. Die Aufgabe wird geteilt und jeder Helfer macht das gleiche mit seinem Aufgabenteil.
    - Beispiel: Schälen von 50kg Kartoffeln mit vier Köchen.
    - Beispiel: Paralleles Suchen oder Filtern von Daten (parallel Streams)
- **Nebenläufig (konkurrierend)**
  - Es fallen verschiedene Aufgaben an, die gleichzeitig gelöst werden müssen. Aufgaben sind in der Regel unabhängig voneinander, benötigen aber Zugriff auf gemeinsam benutzte Ressourcen.
    - Beispiel: In eine Küche gibt es verschiedene Köche mit unterschiedlichen Aufgaben. Köche benutzen z.B. aber die selben „Werkzeuge“.
    - Beispiel: Web-Anwendung (Servlets oder REST-API)

Thread-Konzept



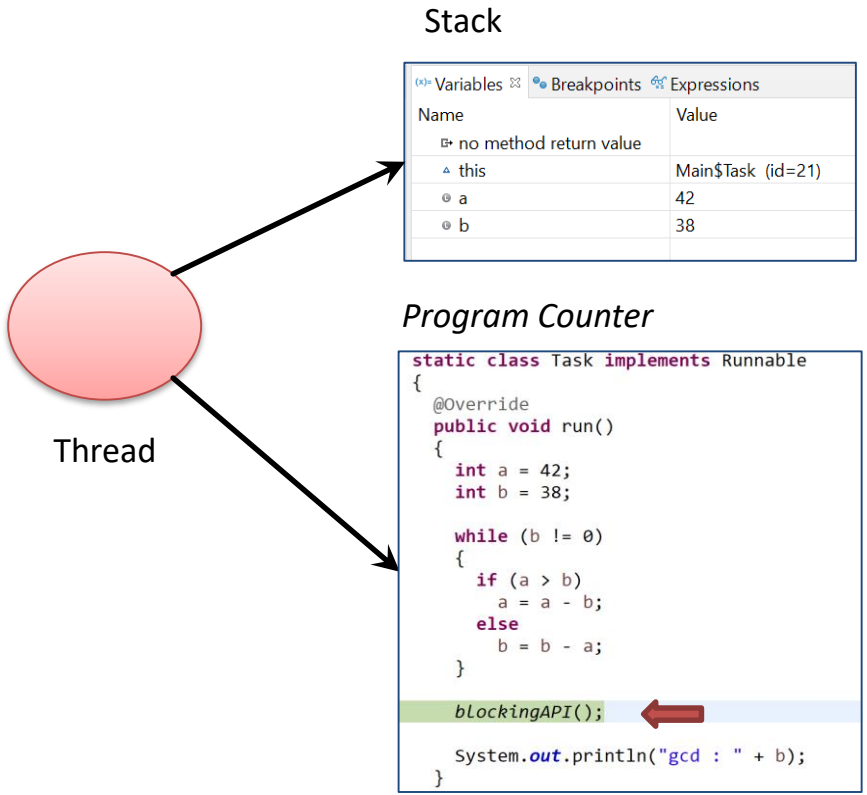
Idee: Multithreading



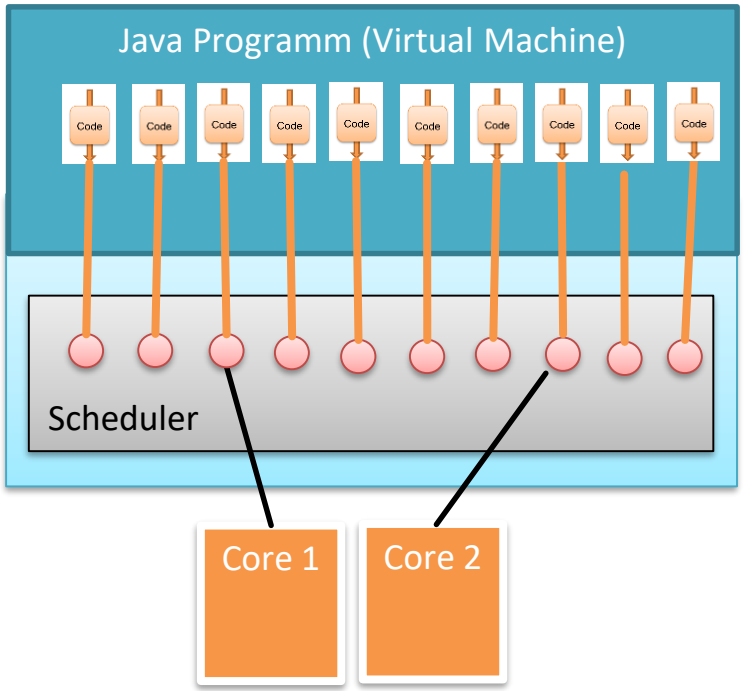


Daten eines Threads (Stack)

- Jeder Thread besitzt seinen eigenen Stack
  - Enthält lokale Daten
  - Sicht des Debuggers
- OS verwaltet Stacks
  - Default 1MB pro VM-Thread
  - Größe ist fix



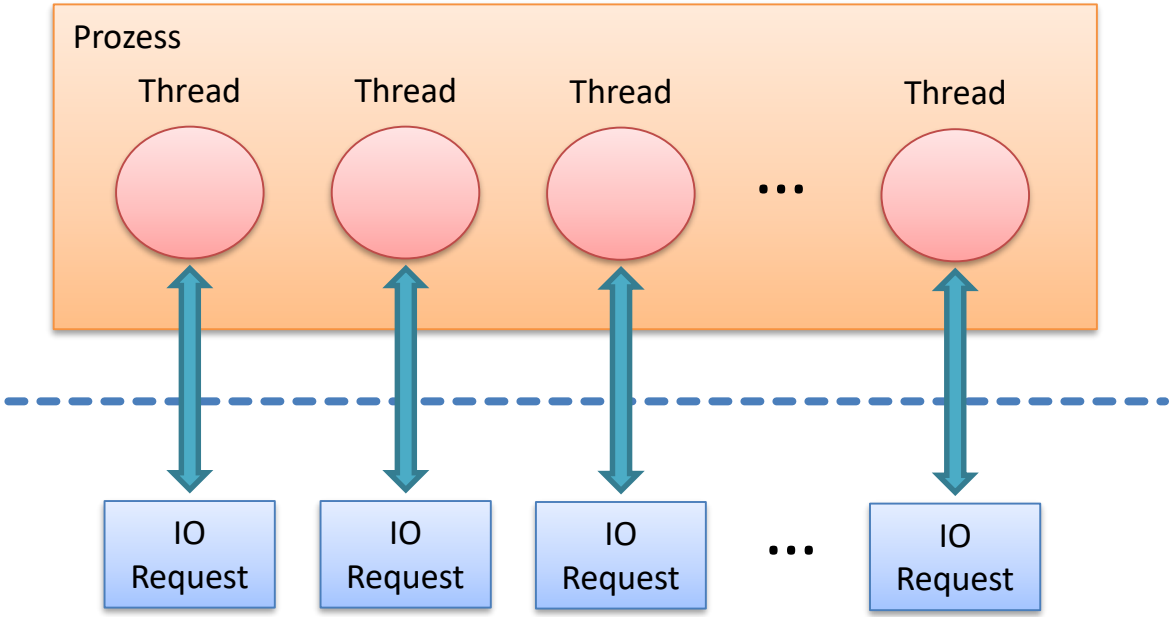
Technische Umsetzung (heute)



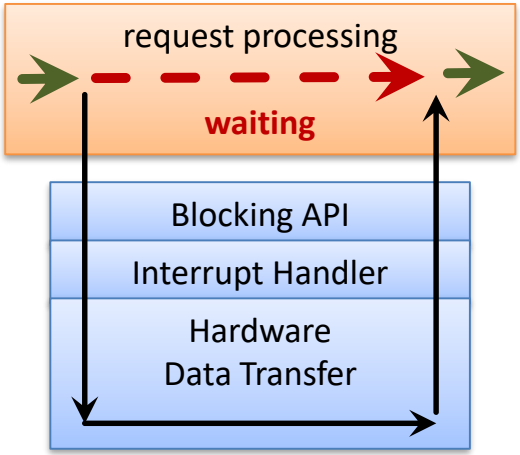
Limitierungen

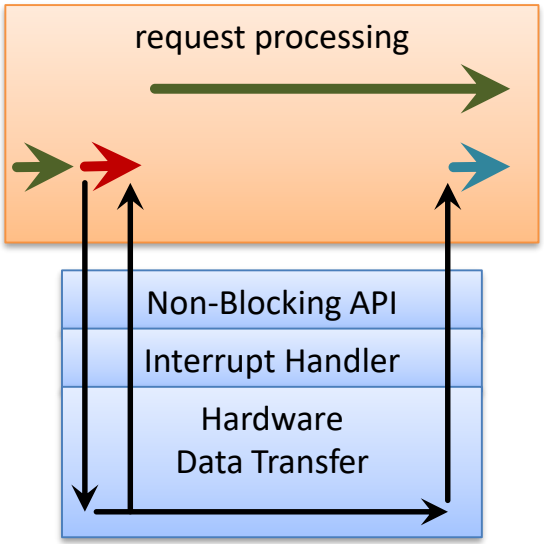
- **Thread ist eine "teure" Ressource**
  - Wird vom Betriebssystem koordiniert, Betriebssystem stellt Stack-Speicher zur Verfügung.
  - Betriebssystem hat „keine Ahnung“ von der Anwendung (generisches Scheduling).
- **Skalierungsbeschränkung**
  - Betriebssystem kann nicht beliebig viele Threads verwalten.
- **Performance-Bottleneck**
  - Stacks der Threads müssen bei einem Kontextwechsel verwaltet werden.

Problem: Wartende Threads



Asynchrone APIs: Funktionsweise

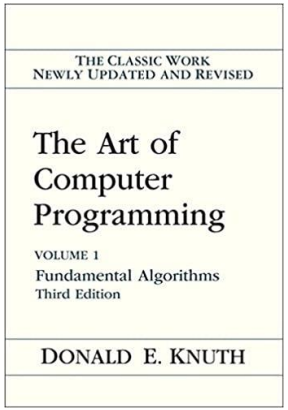




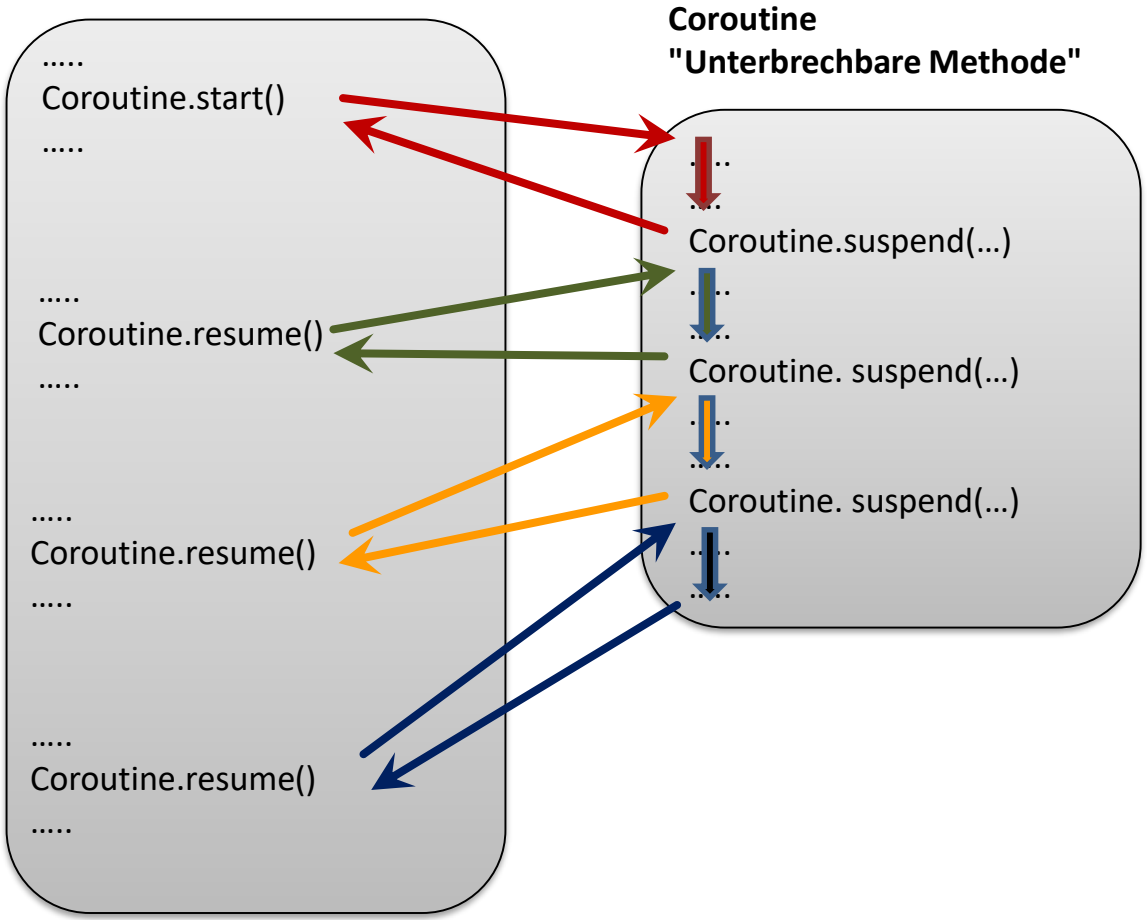
Paradigmen

- **Programmiermodell für synchronen blockierenden Code**
  - „Sequentiell“ ausgeführter Code, einfach zu verstehen
  - Aufruf- /Ablaufreihenfolge entspricht dem „Lesefluss“
    - Debugger-View
  - **Aber:** Blockierende Methoden sind „teuer bezüglich Ressourcenverbrauch (Threads)“
- **Programmiermodell für asynchronen nicht-blockierenden Code**
  - Verwendung von Callbacks
    - Async/await, fetch, CompletableFuture,…
    - Kontext (beim Debugging) geht verloren
  - Erleichterung durch „Reaktive Programmierung“

Umsetzung: Coroutinen-Prinzip

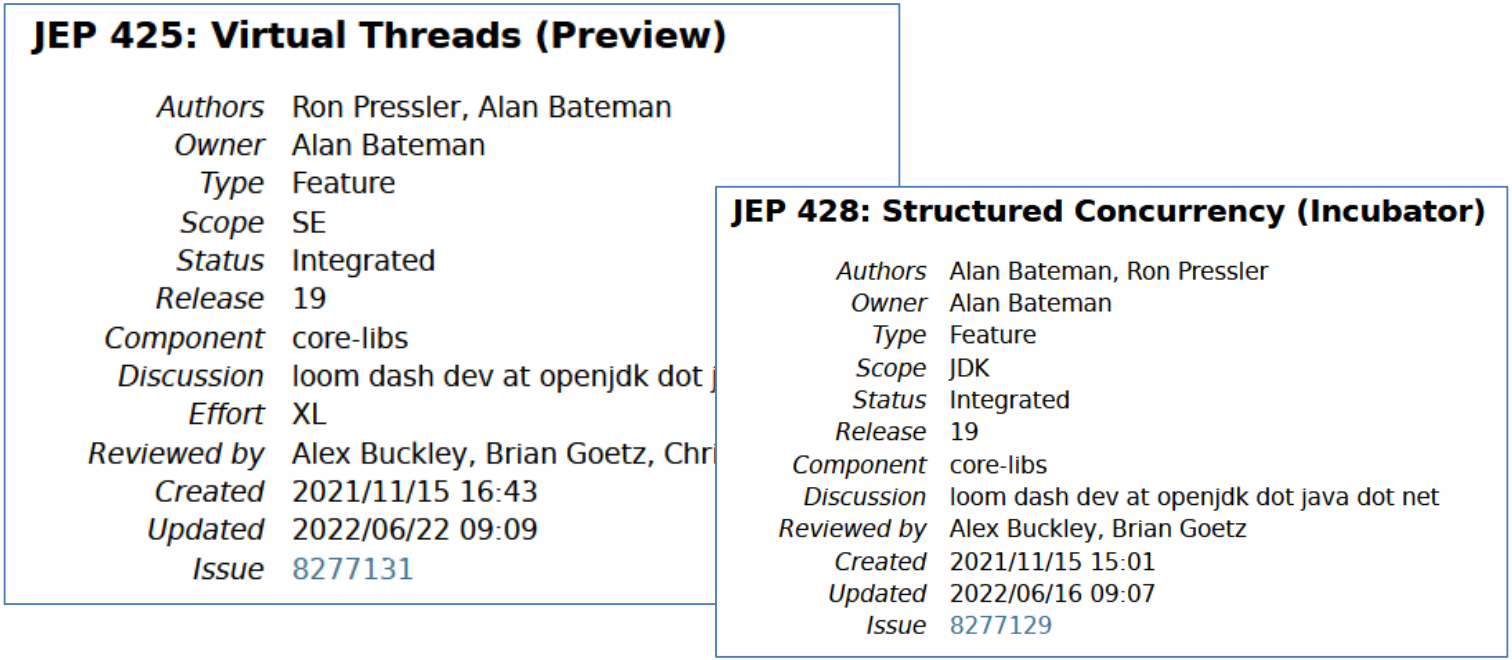


1.4. Some Fundamental Programming Techniques
1.4.1. Subroutines . . . . . .
1.4.2. Coroutines . . . . . .
1.4.3. Interpretive Routines . . . . . .



JEP 425: Virtual Threads (Preview)

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



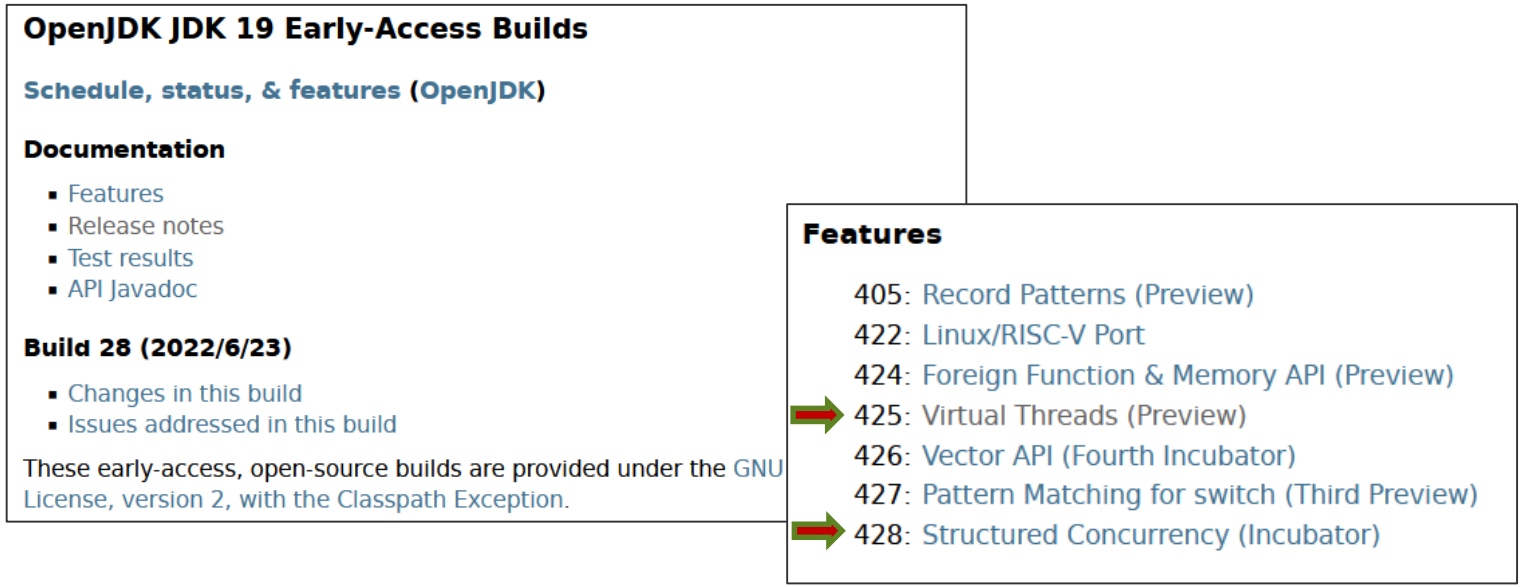
JEP 425: Virtual Threads (Preview)

Authors	Ron Pressler, Alan Bateman
Owner	Alan Bateman
Type	Feature
Scope	SE
Status	Integrated
Release	19
Component	core-libs
Discussion	loom dash dev at openjdk dot j
Effort	XL
Reviewed by	Alex Buckley, Brian Goetz, Chri
Created	2021/11/15 16:43
Updated	2022/06/22 09:09
Issue	8277131

JEP 428: Structured Concurrency (Incubator)

Authors	Alan Bateman, Ron Pressler
Owner	Alan Bateman
Type	Feature
Scope	JDK
Status	Integrated
Release	19
Component	core-libs
Discussion	loom dash dev at openjdk dot java dot net
Reviewed by	Alex Buckley, Brian Goetz
Created	2021/11/15 15:01
Updated	2022/06/16 09:07
Issue	8277129

Early Access



OpenJDK JDK 19 Early-Access Builds

Schedule, status, & features (OpenJDK)

Documentation

- Features
- Release notes
- Test results
- API Javadoc

Build 28 (2022/6/23)

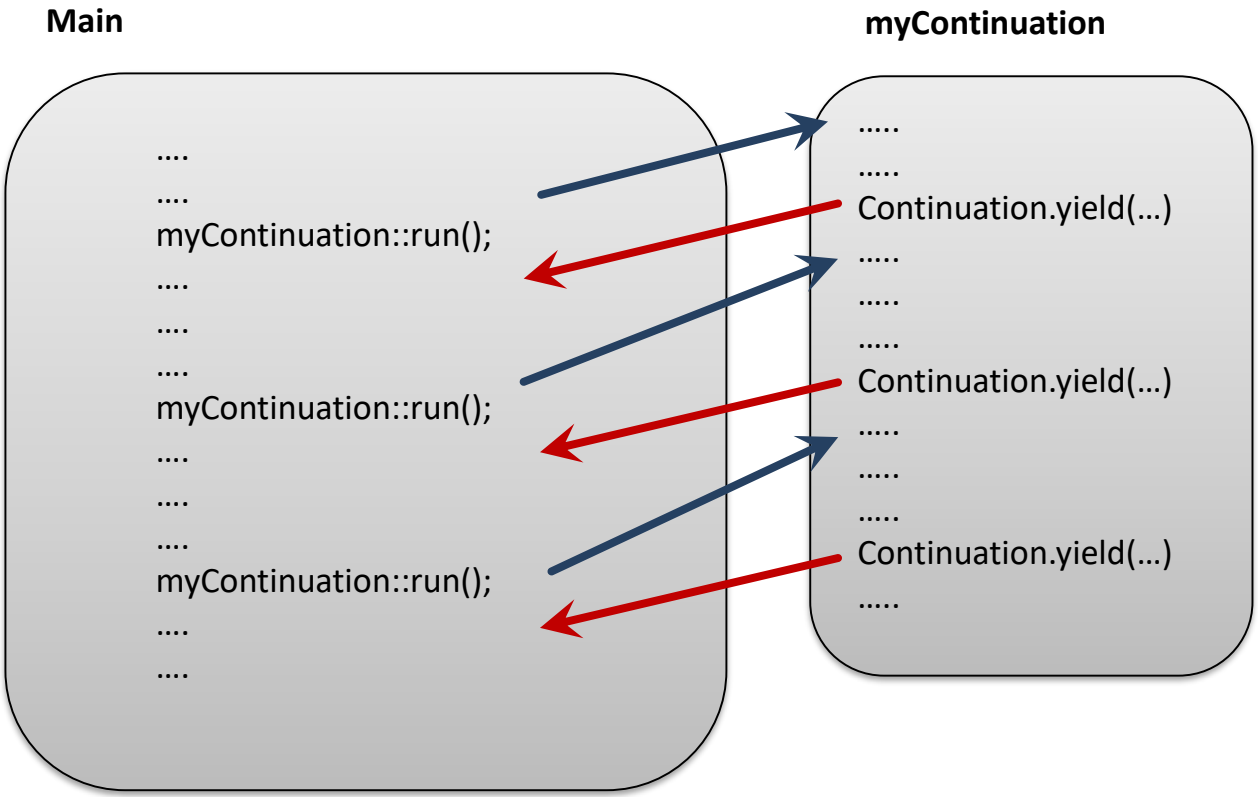
- Changes in this build
- Issues addressed in this build

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

Features

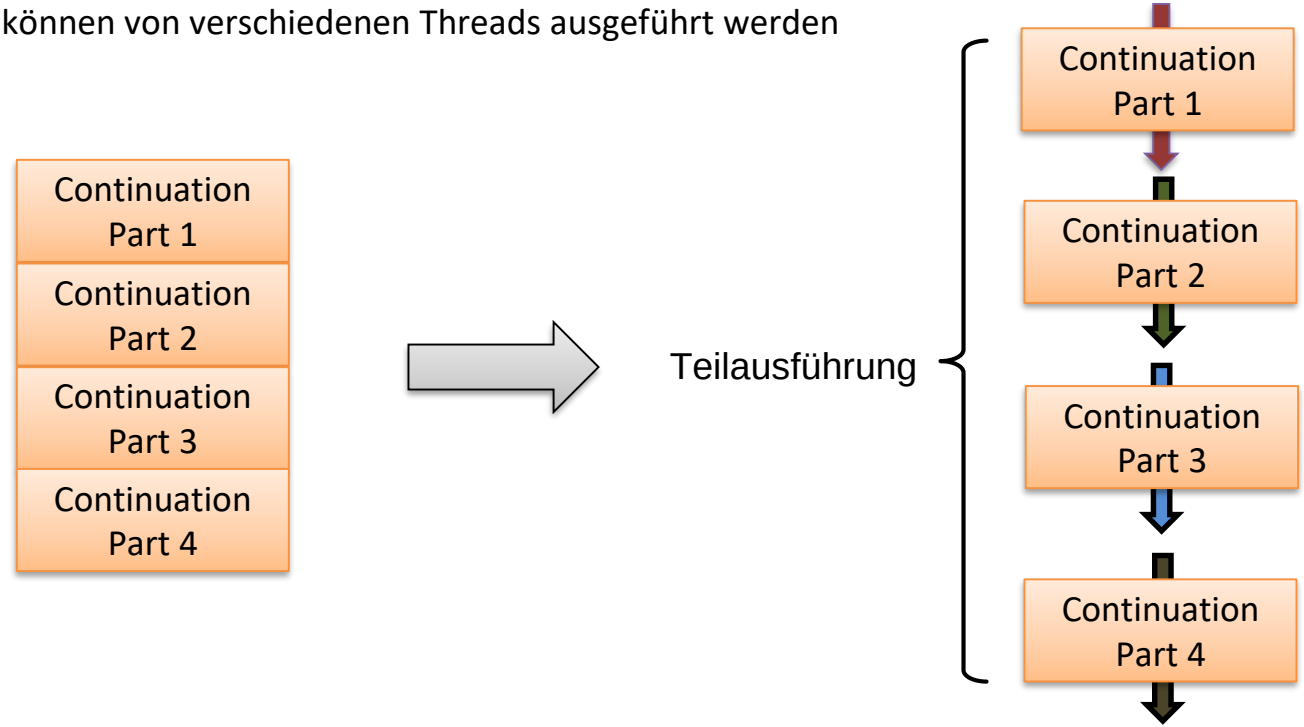
405: Record Patterns (Preview)
422: Linux/RISC-V Port
424: Foreign Function & Memory API (Preview)
425: Virtual Threads (Preview)
426: Vector API (Fourth Incubator)
427: Pattern Matching for switch (Third Preview)
428: Structured Concurrency (Incubator)

Javas Coroutine: (Delimited) Continuation (non-public Klasse)



"Ausführung" einer Continuation

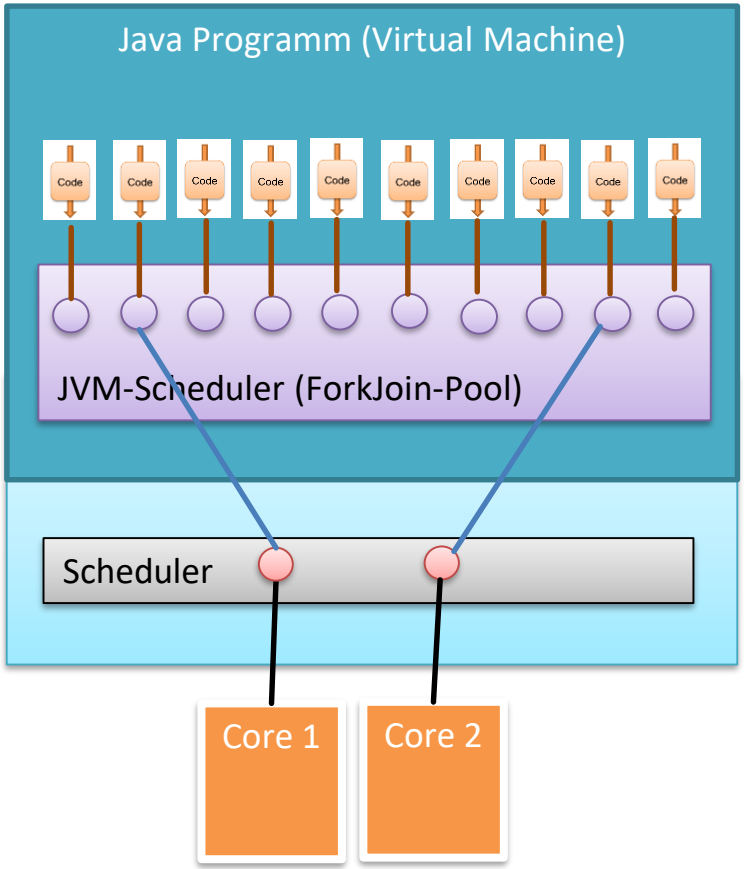
- **Eine Continuation besitzt einen eigenen "Stackframe"**
  - „Daten“ werden auf dem Heap abgelegt
  - Teile können von verschiedenen Threads ausgeführt werden



Code-Beispiele

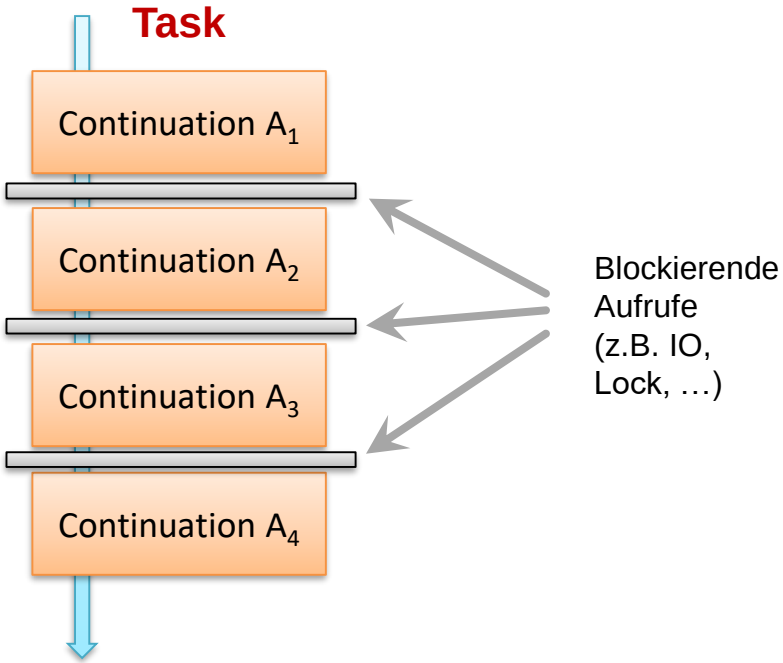
(`Continuation` ist non-public)

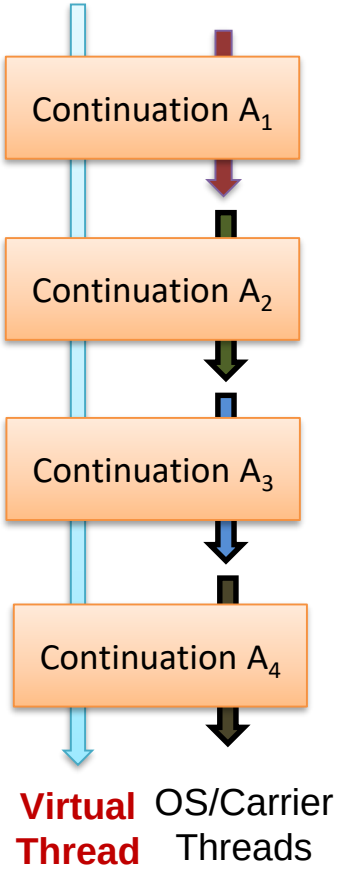
Virtual Thread Scheduling



Umgang mit blockierenden Aufrufen

- **Virtueller Thread als Ausführungsabstraktion**
- Mounting und Unmounting des virtuellen Threads auf bzw. von seinem "Carrier-Thread"





Interna

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

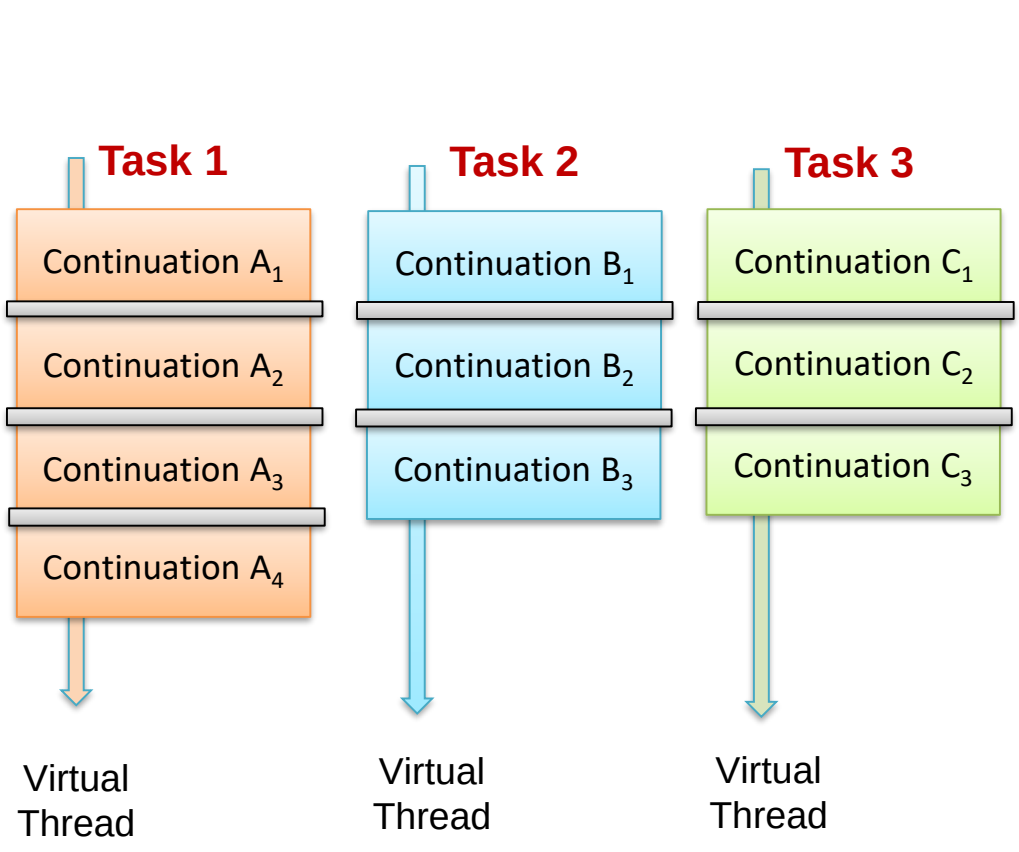
```
public static void park()
{
      if (Thread.currentThread().isVirtual())
      {
            VirtualThreads.park();
      }
     else
      {
            U.park(false, 0L);
      }
 }
```

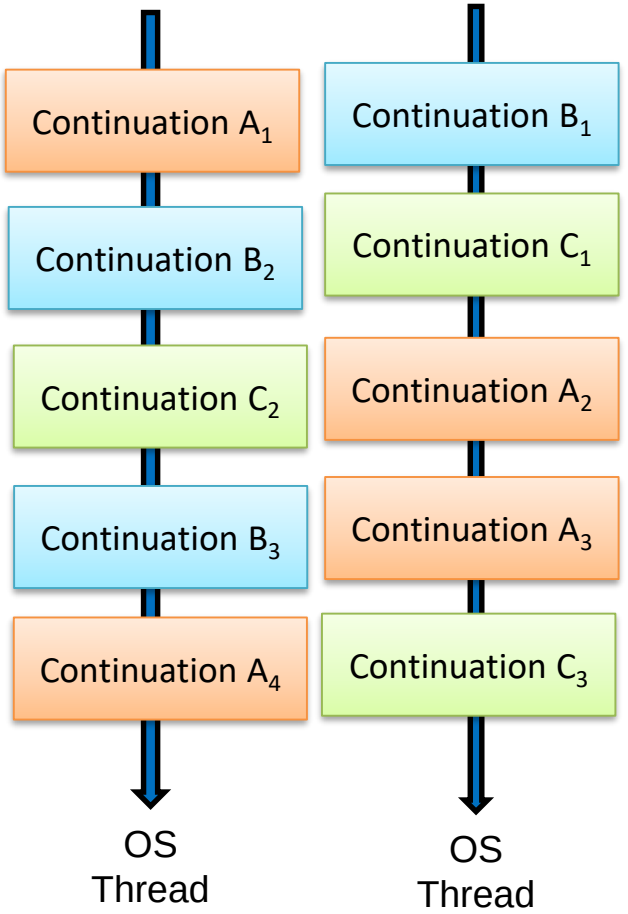
"Virtual Thread Friendly" Klassen

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

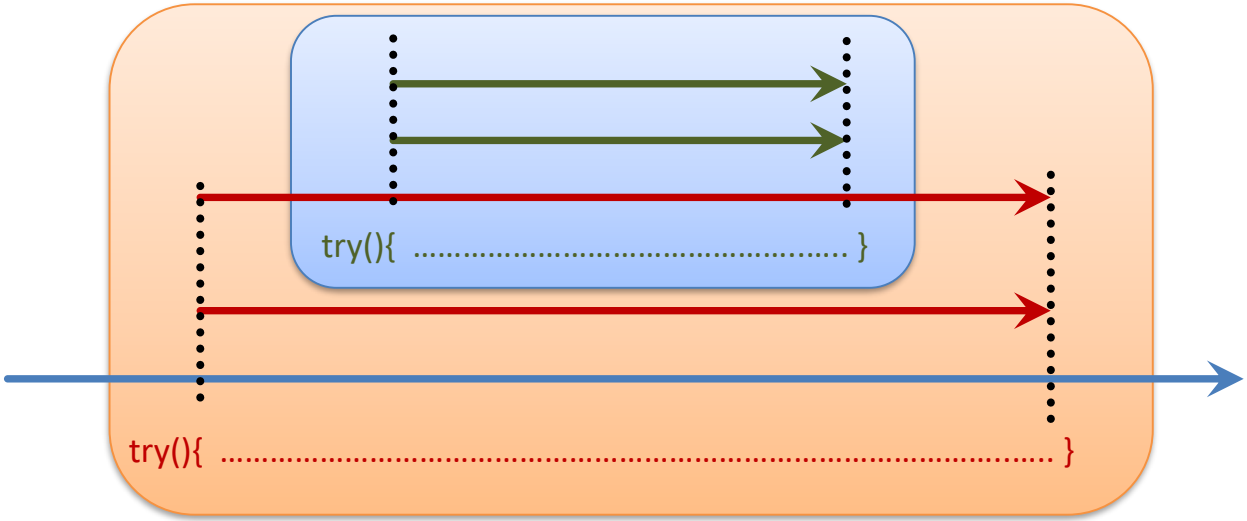
API	Method(s)	Notes
java.lang.Thread	sleep, join	join to wait for a virtual thread to terminate
java.lang.Process	waitFor	Linux/macOS only
java.util.concurrent	All blocking operations	
java.net.Socket	connect, read, write	Socket constructors with a host name parameter may need to do a lookup with InetAddress, 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





Structured Concurrency



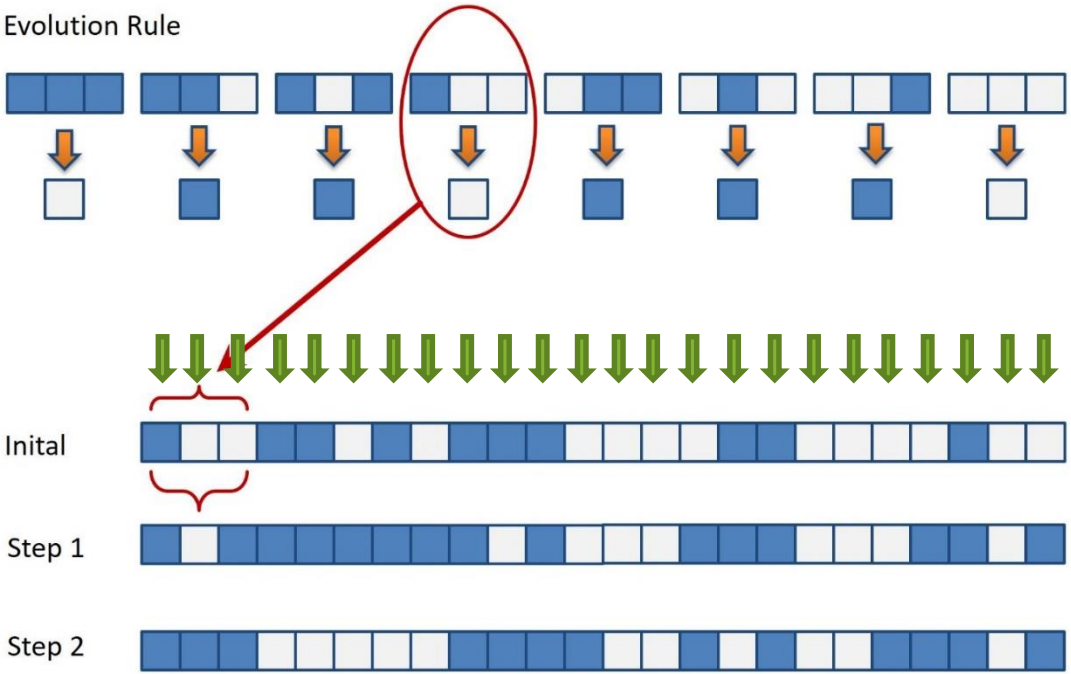
Code-Beispiele

(`StructuredTaskScope` ist noch non-public)

Beispiel: Game of Life

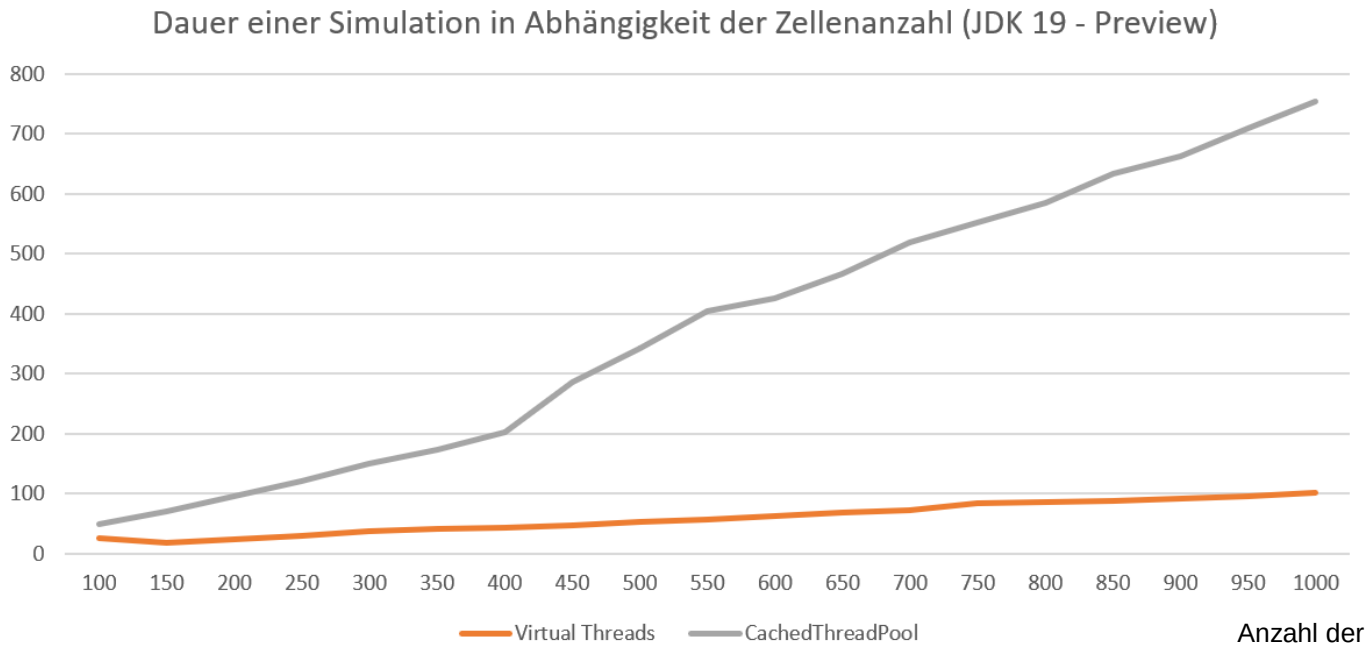
Einfluss auf das Anwendungsdesign

Game of Life



Einfluss auf das Anwendungsdesign

Game of Life: Skalierungsvergleich



Anwendungsfälle

- **CPU-Bound (parallele Verarbeitung)**
  - Ausnutzung von Rechenleistung durch Parallelisierung
  - "gewöhnliche Threads"
- **IO-Bound / Lock-Bound (nebenläufige Verarbeitung)**
  - Viele blockierende Operationen (im Hintergrund)
  - Virtual Threads

Zusammenfassung und Ausblick

Hochschule Kaiserslautern
University of Applied Sciences

- **„Gewöhnliche“ Threads**
- **Virtual Thread Konzept für die asynchrone Programmierung ohne "Callbacks"**
  - (Internes) Konzept der Continuation
  - "Alter Code" bleibt kompatibel
- **Noch anstehende Herausforderungen**
  - Vernünftige Unterstützung von Debuggern und Profilern
    - Z.T. gelöst
  - Interaktion mit Build-In-Synchronization (Monitor-Konzept)
  - Virtual Thread Deadlock
  - Aufruf von Native-Code

Vielen Dank und gibt es Fragen?

Kontakt:

Jörg Hettel

joerg.hettel@hs-kl.de

Hochschule Kaiserslautern
University of Applied Sciences