Parallele Programmierung

Prof. Dr. Luc Bläser

OST
Ostschweizer
Fachhochschule

Willkommen zu «ParProg»

- Das breite Spektrum der Parallelprogrammierung
  - Lokale Parallelisierung mit Multi-Threading
  - Verteilte Parallelisierung mit GPUs und auf Cluster
  - Fortgeschrittene Modelle
  - Verschiedene Technologien: Java, C#, C, C++, Python, JS

HPC Cluster

Java

.NET TPL

Actor Model

GPU-Parallelisierung

HPC

MPI

CUDA

async/await

JavaScript

OpenCL

Python

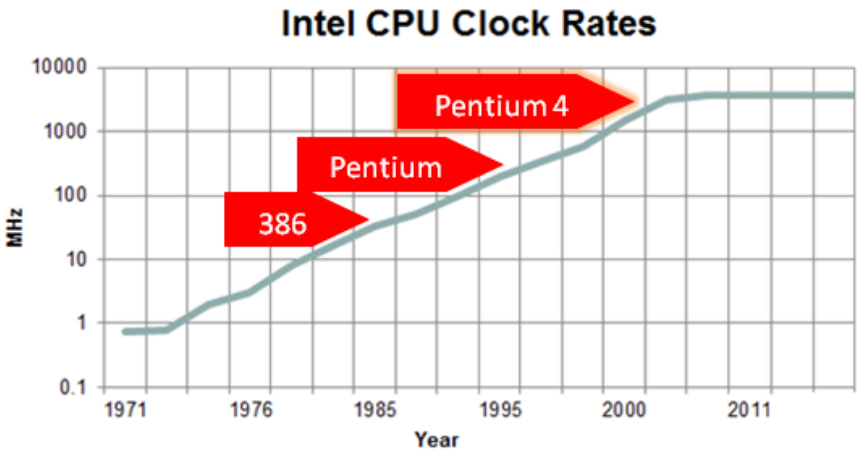
Wieso «Parallele Programmierung»?

- Performance
  - Mehrere Prozessoren nutzen
- Simplicity
  - Zeitlich unabhängige Aufgaben ausführen

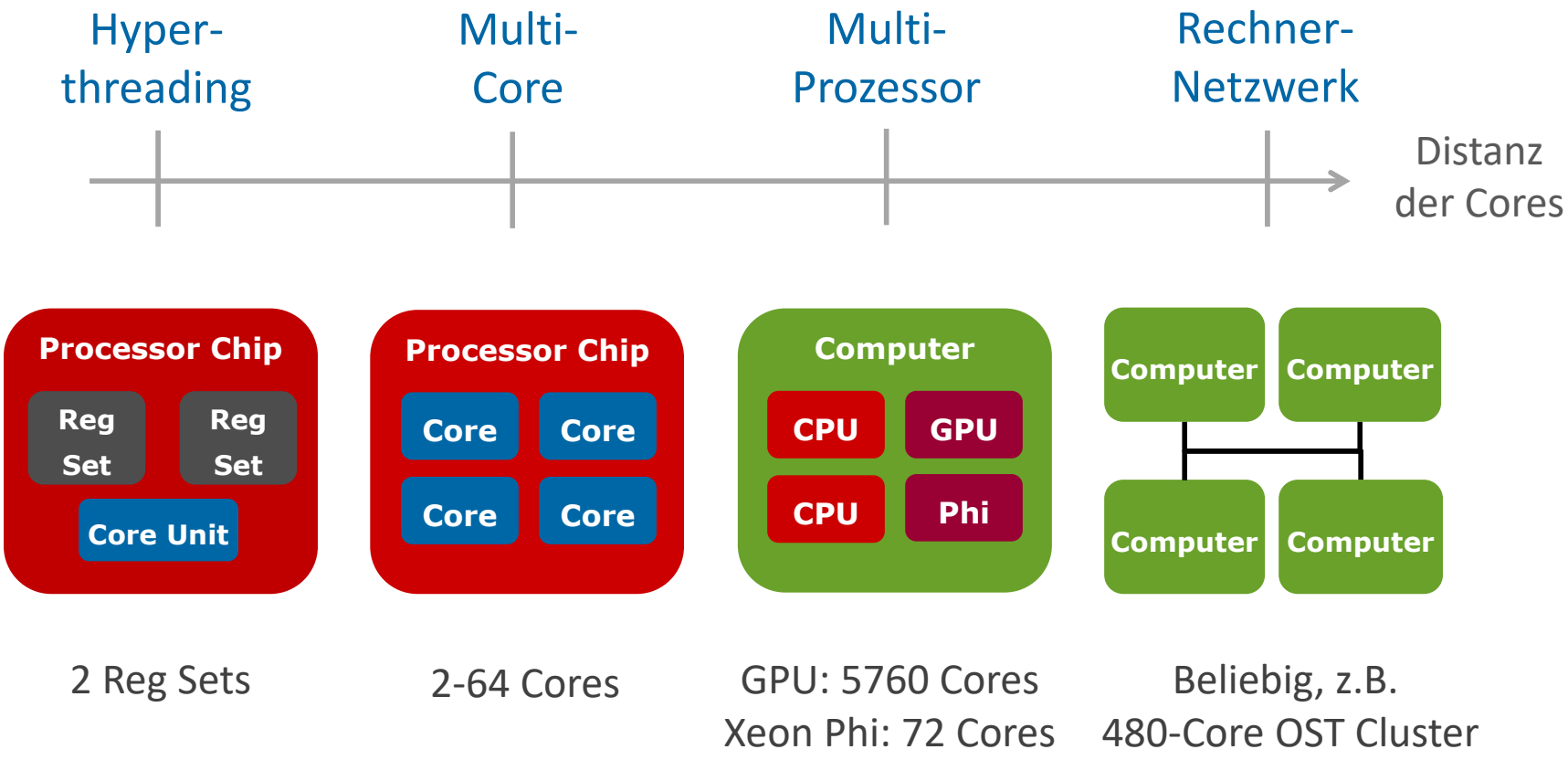
Die Multi-Core Ära

- Prozessoren werden schon lange nicht schneller
- Dafür kriegt man mehr Cores pro Chip und Geld

Zur Beschleunigung braucht es Parallelisierbarkeit - und die muss zuerst mal programmiert werden!



Stufen der Parallelisierung



Parallelität und Nebenläufigkeit

- Parallelität (Parallelism)
  - Zerlegung eines Ablaufs in mehrere Teilabläufe, welche gleichzeitig auf mehreren Prozessoren laufen.
  - Ziel: Schnellere Programme
- Nebenläufigkeit (Concurrency)
  - Gleichzeitig oder verzahnt ausführbare Abläufe, welche auf gemeinsame Ressourcen zugreifen.
  - Ziel: Einfachere Programme

Dasselbe Prinzip = Mehrere Threads/Prozesse, die interagieren

Inhalt der Vorlesung (1)

- Multi-Threading
  - Grundlagen
  - Monitor Synchronisation
  - Gefahren und Korrektheitsbedingungen
  - Spezifische Synchronisationsprimitiven
  - Thread Pools und Asynchronität
  - Task Parallel Library in .NET
  - GUI und Threading
  - Speichermodelle

Inhalt der Vorlesung (2)

- Fortgeschrittene Themen
  - Actor Model
  - GPU Parallelisierung (CUDA)
  - HPC Cluster Parallelisierung
  - Concurrency in C#, Python, JavaScript

Lernziele der Vorlesung

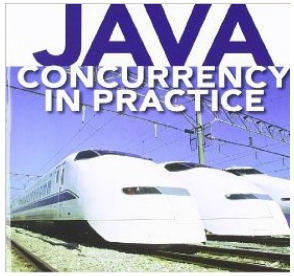
«Writing correct programs is hard – writing correct concurrent programs is harder» - Brian Goetz

- Die Konzepte der Nebenläufigkeit verstehen und in Programmen anwenden können
- Die Korrektheitskriterien der Nebenläufigkeit kennen, typische Gefahren erkennen und vermeiden
- Programme mittels Parallelisierung auf Multi-Cores, GPUs und Cluster beschleunigen können
- Verschiedene Programmiermodelle für Nebenläufigkeit kennen lernen

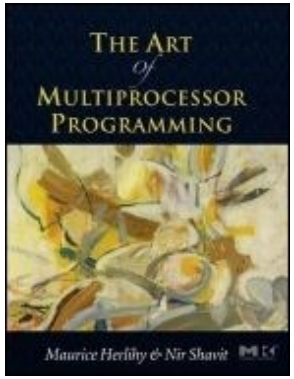
Textbücher zur Vertiefung (1)

Multi-Threading Java

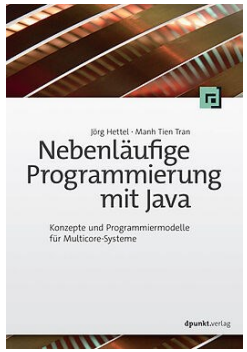
- B. Goetz et al. Java Concurrency in Practice, Addison-Wesley, 2006.
  (konzeptionell gut, aber älteres Java)



- M. Herlihy and N. Shavit. The Art of Multiprocessor Programming, Revised Reprint, Morgan Kaufmann, 2012.
  (Klassiker, konzeptionell stark)



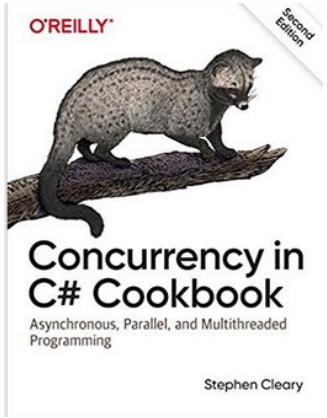
- J. Hettel and T. Tran. Nebenläufige Programmierung mit Java, dpunkt, 2016.
  (aktuelleres Java, deutsch)



Textbücher zur Vertiefung (2)

Multi-Threading .NET

- S. Clearly. Concurrency in C# Cookbook, 2nd Edition, O'Reilly, 2019
  (aktueller, wenig konzeptionell)

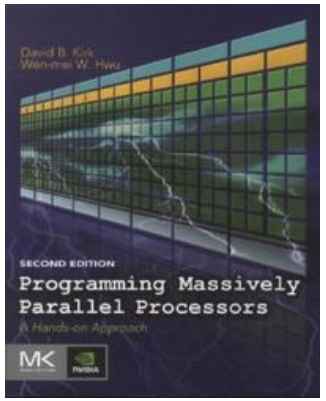


- M. A. Zhou. Parallel Computing in .NET. Entwickler Press, 2009.
  (konzeptioneller, aber älteres .NET)

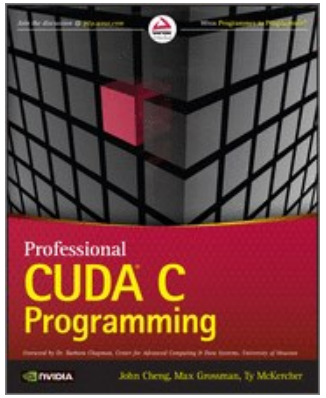
Textbücher zur Vertiefung (3)

GPU Parallelisierung

- D. B. Kirk and W. W. Hwu. Programming Massively Parallel Processors, 3rd Edition, Morgan Kaufmann, 2016.
  (konzeptionell gut, diverse Technologien)



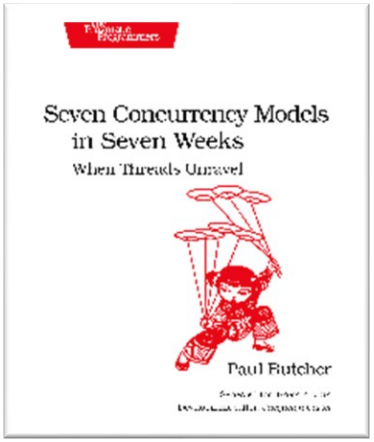
- J. Cheng et al. Professional CUDA C Programming, Wrox, 2014.
  (nur CUDA)

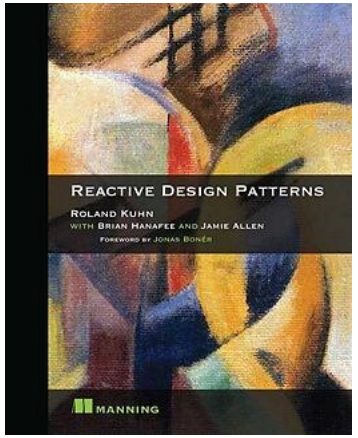


Textbücher zur Vertiefung (4)

Fortgeschrittene Themen

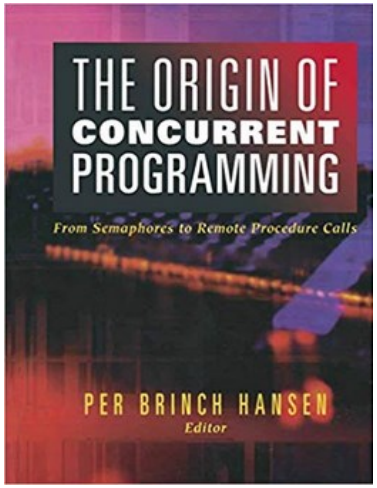
- P. Butcher. Seven Concurrency Models in Seven Weeks, Pragmatic Bookshelf, 2014.
- P. Kuhn et al. Reactive Design Patterns, Manning, 2017.

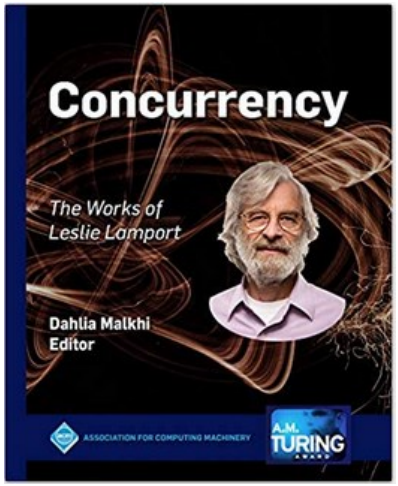




Historischer Überblick

- P. B. Hansen. The Origin of Concurrent Programming, 2013
- D. Malkhi. Concurrency: The Works of Leslie Lamport, 2019





Parallele Programmierung
Multi-Threading Grundlagen

Vorlesung 1

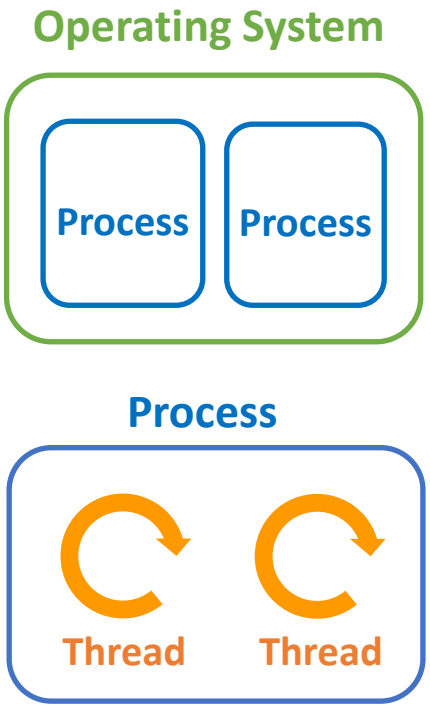
Prof. Dr. Luc Bläser

Inhalt Heute

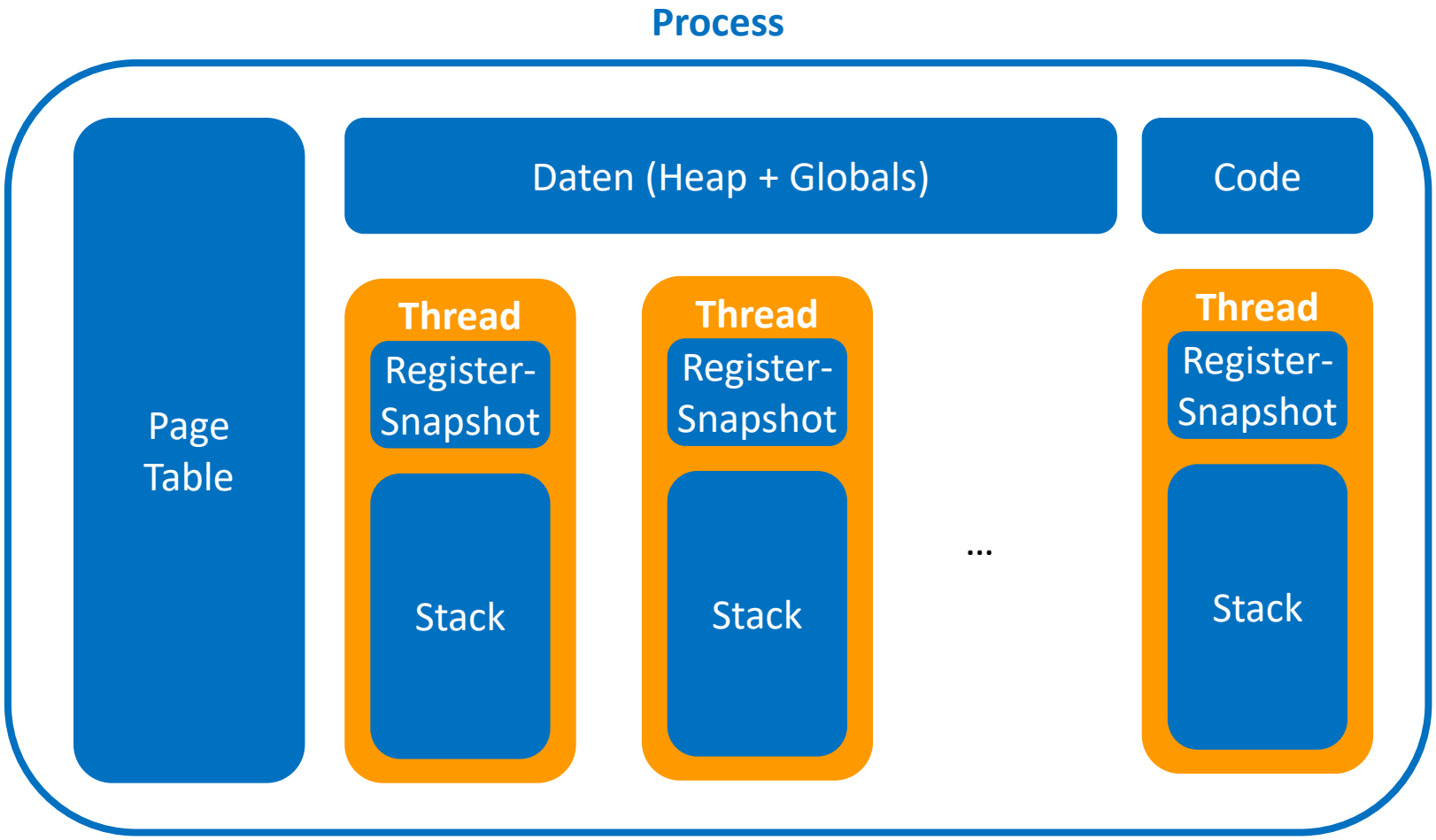
- Inhalt
  - System-Grundlagen Repetition
  - Multi-Thread Programmierung in Java
- Lernziele
  - Repetition der Betriebssystem-Konzepte von Prozess und Thread
  - Erlernen der Grundlagen der nebenläufigen Programmierung mit Java Threads

Parallelität des Betriebssystems

- Prozess (Schwergewichtsprozess)
  - Parallel laufende Programm-Instanz im System
  - Eigener Adressraum pro Prozess
- Thread (Leichtgewichtsprozess)
  - Parallele Ablaufsequenz innerhalb eines Programms
  - Teilen gleichen Adressraum im Prozess



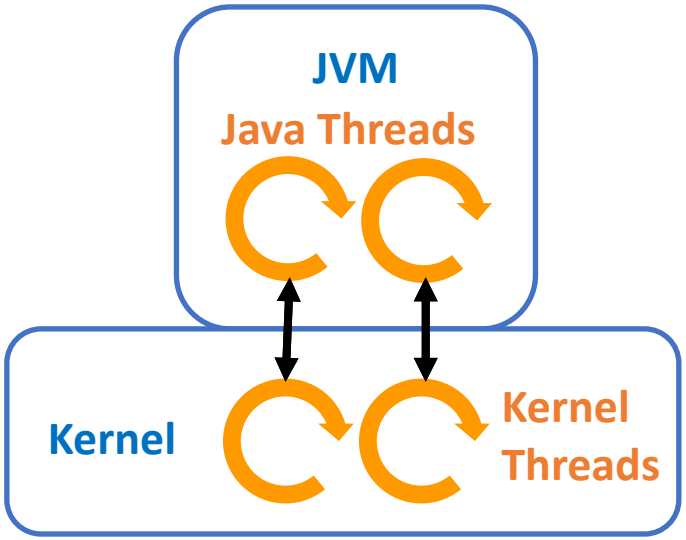
Speicherressourcen



Thread-Implementationen

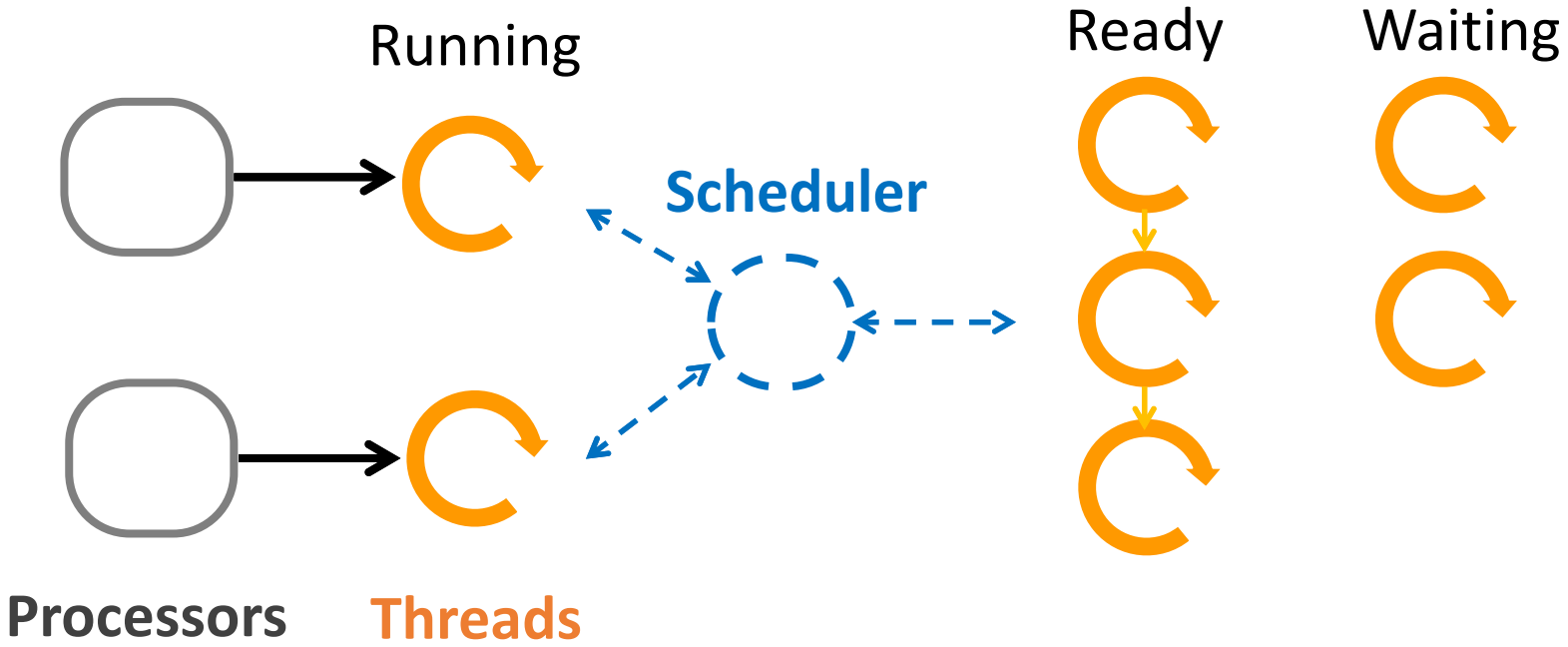
- User-Level Threads
  - Im Prozess implementiert (Programm-Laufzeitsystem)
  - Keine echte Parallelität durch mehrere Prozessoren
- Kernel-Level Threads
  - Im Kernel implementiert (Multi-Core Ausnutzung)
  - Kontextwechsel vom Prozess per SW-Interrupt (Kernel-Mode)

Heutzutage immer Kernel-Level Threading



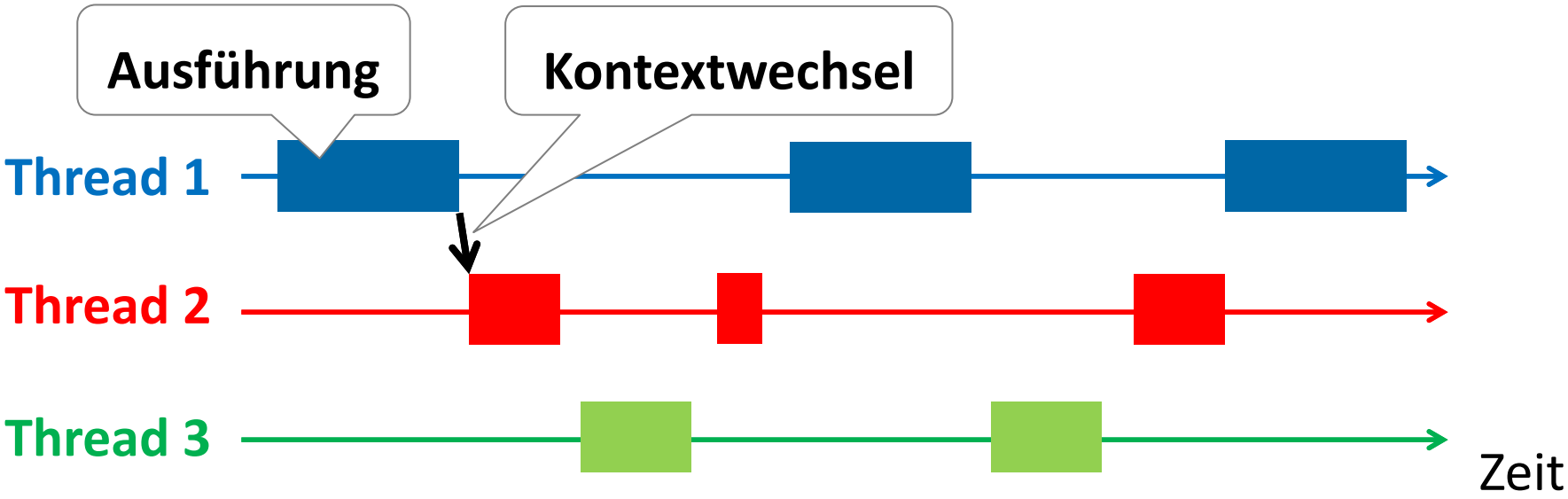
Thread Scheduling

- Processor Sharing
  - Mehr Threads als Prozessoren ausführen
  - Bei Wartebedingung: Prozessor an anderen bereiten Thread freigeben



Prozessor Multiplexing

- Verzahnte Ausführung
  - Prozessor führt Instruktionen von mehreren Threads abwechslungsweise in Teilsequenzen aus
  - Illusion der Parallelität mehrerer Threads auch bei nur einem Prozessor (Quasiparallelität)



Kontextwechsel

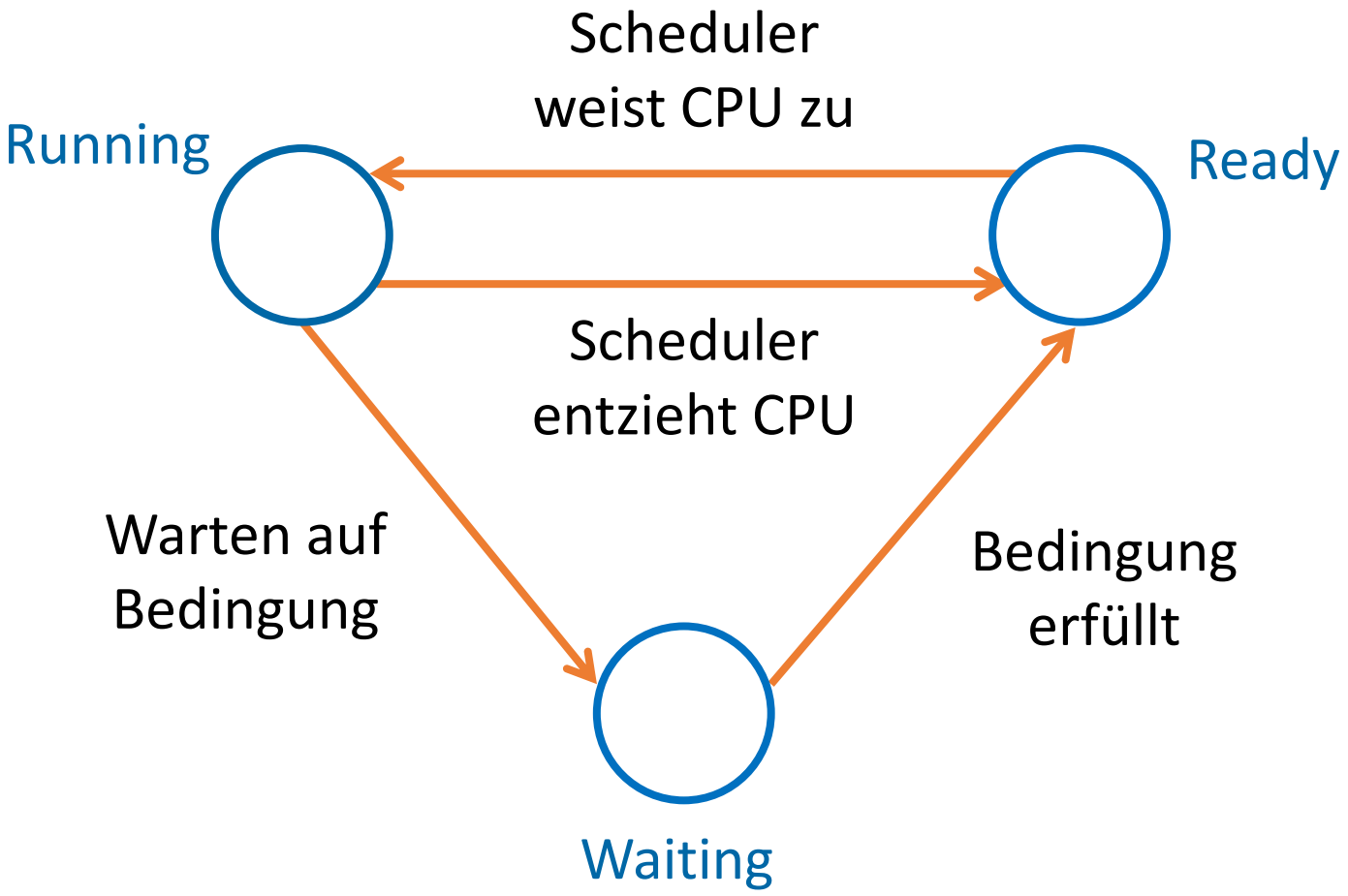
- Synchron: Warten auf Bedingung
  - Thread wartet auf Bedingung
  - Reiht sich als wartend ein und gibt Prozessor frei
- Asynchron: Zeitablauf
  - Nach gewisser Zeit soll der Thread den Prozessor freigeben
  - Verhindere, dass ein Thread dauerhaft Prozessor belegt

Multi-Tasking

- Kooperativ
  - Threads müssen explizit beim Scheduler in Abständen Kontextwechsel synchron initiieren
  - Scheduler kann laufenden Thread nicht unterbrechen
- Preemptiv
  - Scheduler kann per Timer-Interrupt den laufenden Thread asynchron unterbrechen
  - Time-Sliced Scheduling: Jeder Thread besitzt den Prozessor für einen maximalen Zeitintervall

Heutzutage immer preemptiv

Thread Zustände



Multi-Thread Programmierung

- Grundlagen der Java Threads
  - Ausführungsmodell
  - Implementierung
  - Start und Join
  - Passivierung

JVM Thread Modell

- Java ist ein Single Process System
  - Java Virtual Machine (JVM) ist ein Prozess im Betriebssystem
- Main-Thread
  - JVM erzeugt beim Aufstarten einen Thread, welcher die statische Methode `main()` aufruft
- Programmierer kann weitere Threads starten
  - Subsysteme / Laufzeitsystem starten auch eigene Threads (z.B. RMI, AWT, Garbage Collector)

JVM Terminierung

- Die JVM läuft, solange Threads laufen
  - Ausnahme: Threads können als Daemon markiert werden
  - JVM wartet nicht auf Daemon Threads
    - z.B. Garbage Collector
  - Daemon Threads brechen bei JVM Ende unkontrolliert ab
- Oder `System.exit()`/`Runtime.exit()`
  - Direkte Terminierung der JVM => unsauber
  - Unkontrolliertes Abbrechen aller Threads

Starten eines Threads

Thread implementieren

```
var myThread = new Thread(() -> {
  // thread behavior
});
```

```
myThread.start();
```

Nach Erzeugung starten

Thread Implementation

- Lambda oder Methodenreferenz implementiert Runnable Funktionsschnittstelle:

```
@FunctionalInterface
interface Runnable
  void run();
}
```

- Thread-Konstruktor nimmt dies entgegen:

```
Thread(Runnable behavior)
```

Thread Start und Ende

- Richtiger Thread wird erst bei `start()` erzeugt
  - Führt `run()`-Methode des `Runnable` Interface aus
- Thread endet beim Verlassen von `run()`
  - Ende der Methode
  - Return Statement
  - Unbehandelte Exception

Bei unbehandelter Exception: Andere Threads laufen weiter

Multi-Thread Beispiel

```
public class MultiThreadTest {
  public static void main(String[] args) {
    var a = new Thread(() -> multiPrint("A"));
    var b = new Thread(() -> multiPrint("B"));
    a.start();
    b.start();
    System.out.println("main finished");
  }

  static void multiPrint(String label) {
    for (int i = 0; i < 10; i++) {
      System.out.println(label + ": " + i);
    }
  }
}
```

Welche Ausgabe erwarten Sie?

Nicht-Determinismus

- Threads laufen ohne Vorkehrungen beliebig verzahnt oder parallel
  - Viele JVMs realisieren einzelne System Outputs ohne Verzahnung/Thread-Fehler - aber es ist nicht spezifiziert!

Alternative Thread-Implementationen

1. Explizite Runnable-Implementation
2. Sub-Klasse von Thread

Explizite Runnable-Implementation

```
class SimpleLogic implements Runnable {
  @Override
  public void run() {
    // thread behavior
  }
}

var myThread = new Thread(new SimpleLogic());

myThread.start();
```

Sub-Klasse von Thread

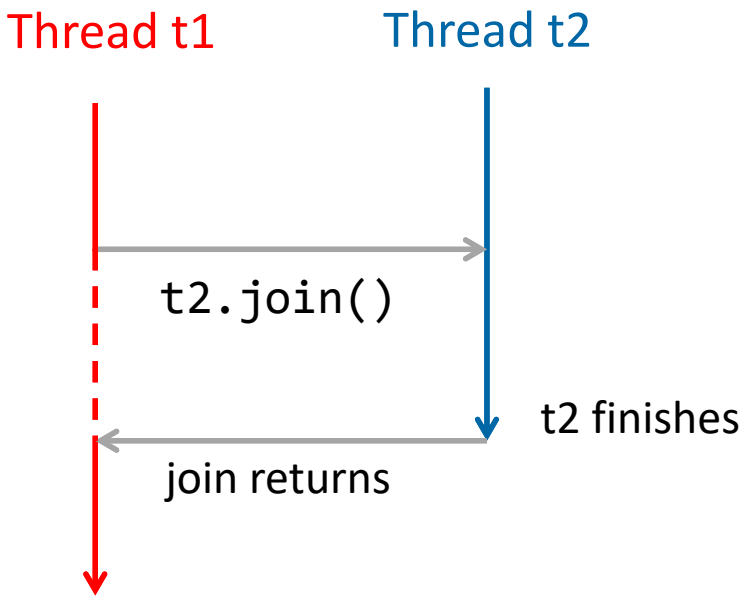
```
class SimpleThread extends Thread {
  @Override
  public void run() {
    // thread behavior
  }
}

var myThread = new SimpleThread();

myThread.start();
```

Thread Join

- Warten auf Beendigung eines Threads



`t2.join()` blockiert, solange `t2` läuft

Thread Join Beispiel

```
var a = new Thread(() -> multiPrint("A"));
var b = new Thread(() -> multiPrint("B"));
System.out.println("Threads start");
a.start();
b.start();
…
a.join();
b.join();
System.out.println("Threads joined");
```

Thread Passivierung

- Statische Methoden der Thread-Klasse
  - `Thread.sleep(milliseconds)`
    - Laufender Thread geht in Wartezustand
    - Nach Zeitablauf wird er wieder ready
  - `Thread.yield()`
    - Laufender Thread gibt Prozessor frei
    - Wird aber direkt wieder ready

Braucht man wirklich ein yield()?

InterruptedException

- Mögliche Exception bei blockierenden Aufrufen
  - `join()`, `sleep()` etc.
- Threads können von aussen unterbrochen werden
  - `myThread.interrupt()`
- Für kooperatives Canceling
  - Nur verwenden, wenn Cancel-Policy des Threads bekannt

Weitere Thread Methoden

- `static Thread currentThread()`
  - Gerade ausführende Thread-Instanz
- `void setDaemon(boolean on)`
  - Thread als «Daemon» markieren (default ist false)

Rückblick: Lernziele Heute

- ✓ Repetition der Betriebssystem-Konzepte von Prozess und Thread
- ✓ Erlernen der Grundlagen der nebenläufigen Programmierung mit Java Threads

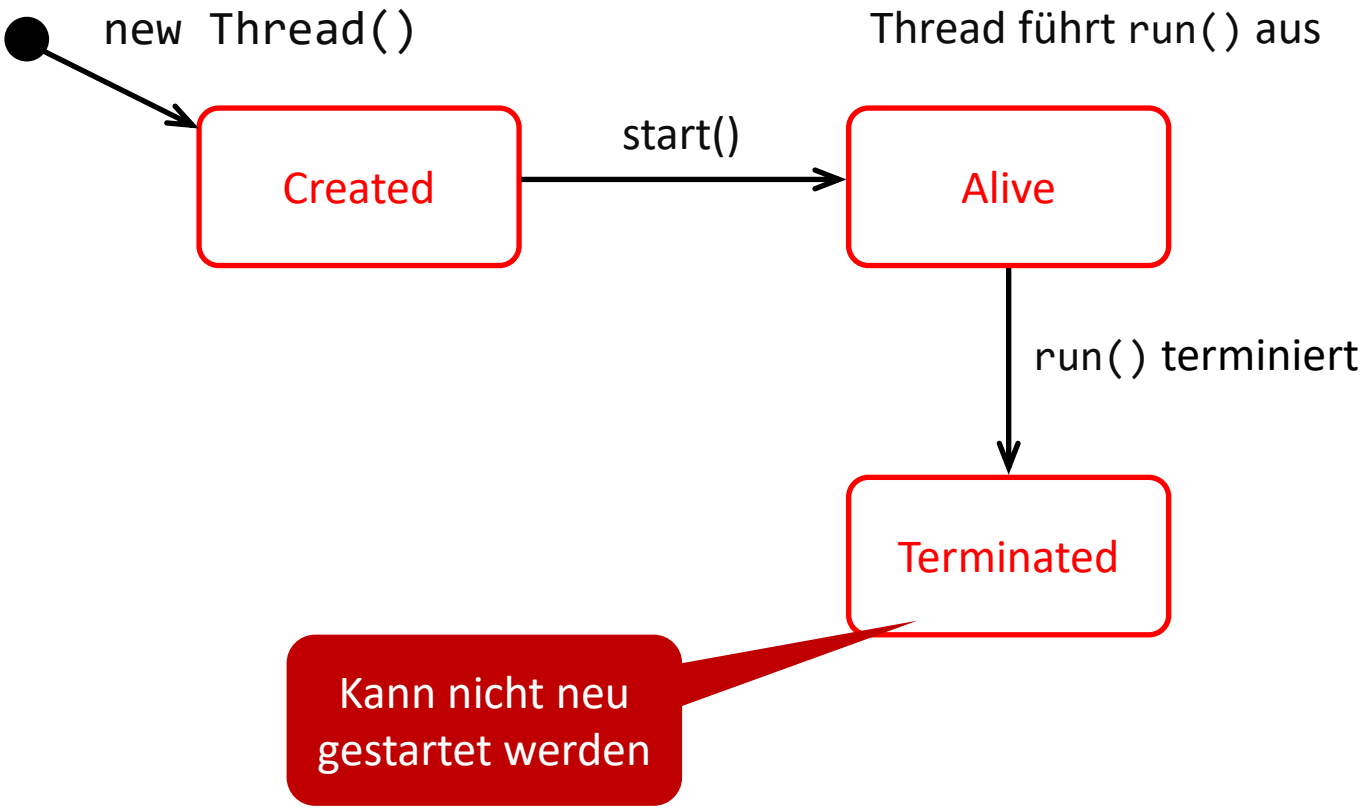
Selbststudium (nach Bedarf)

- B. Goetz. Java Concurrency in Action, Kapitel 1
- M. Herlihy, N. Shavit. The Art of Multiprocessor Programming, Appendix A.1, A.2.1 und A.2.3
- E. Glatz. Betriebssysteme, 4. Auflage, dpunkt Verlag, 2015, Kapitel 4
- A. Tannenbaum, H. Bos. Modern Operating Systems, 4th Edition, Pearson, 2014, Chapter 2
- A. Silberschatz, P. B. Galvin, G. Gagne. Operating System Concepts, Wiley, 2013, Part 2
- H. Sutter: The Free Lunch is Over, Dr. Dobbs, 30(3), March 2005

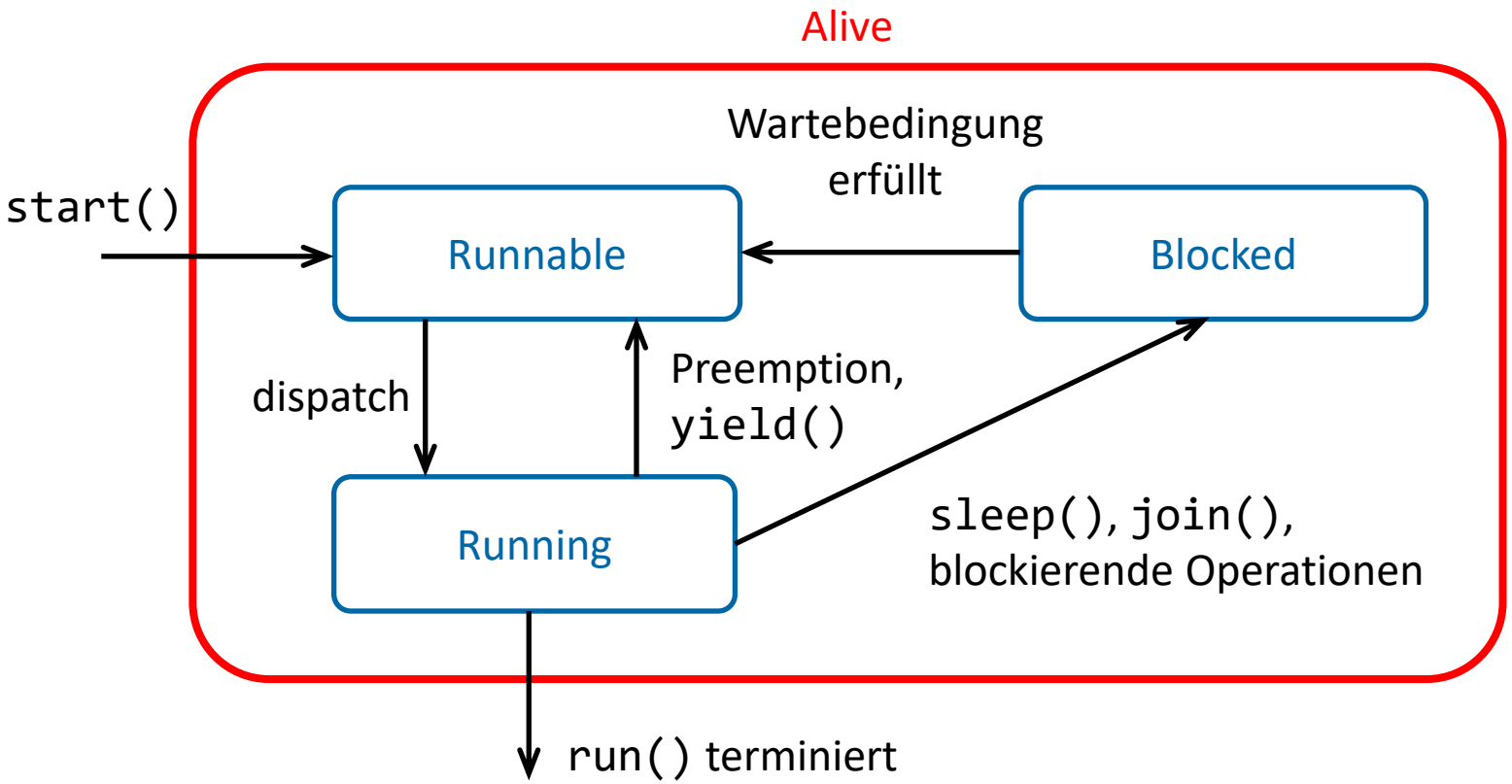
Anhang: Thread Zustände

Zum Selbststudium

Java Thread Lifecycle



Thread Alive Unterzustände



Frühere suspend/resume/stop Aufrufe sind veraltet (Designfehler)