Java

Contents

[Parallelprogrammierung 2](#_Toc67312699)

[Amdahlsches Gesetz 2](#_Toc67312700)

[Happens-before-Beziehung 2](#_Toc67312701)

[Executor 2](#_Toc67312702)

[ExecutorService 3](#_Toc67312703)

[submit 3](#_Toc67312704)

[Futures 4](#_Toc67312705)

[Berechnung mit ExecutorService und Futures 4](#_Toc67312706)

[Vorgehensweise 4](#_Toc67312707)

[Complex Case 4](#_Toc67312708)

[Java Advanced 5](#_Toc67312709)

[Streams 5](#_Toc67312710)

[Optional 5](#_Toc67312711)

[Design by Contract 5](#_Toc67312712)

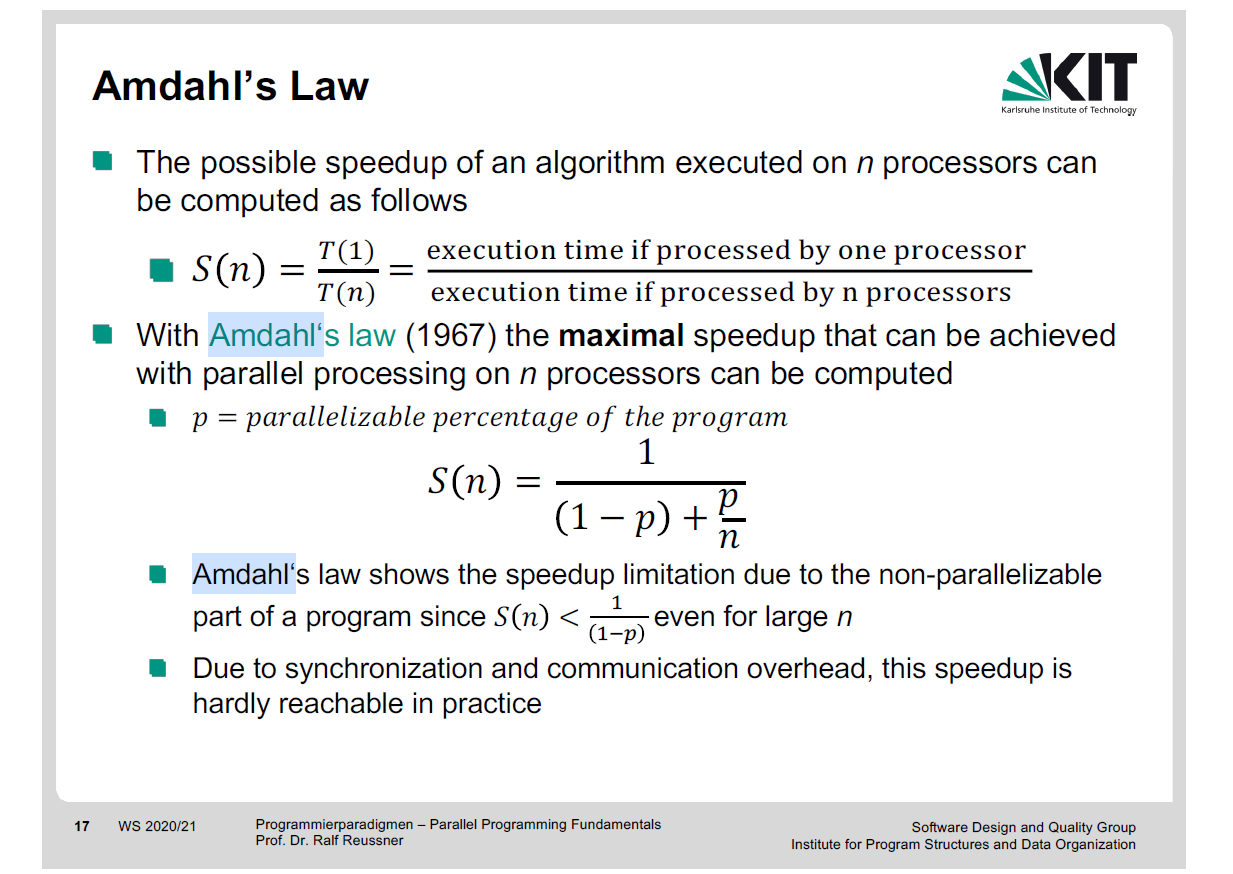
[Bespiele 5](#_Toc67312713)

[Amdahlsches Gesetz aus Ü9 5](#_Toc67312714)

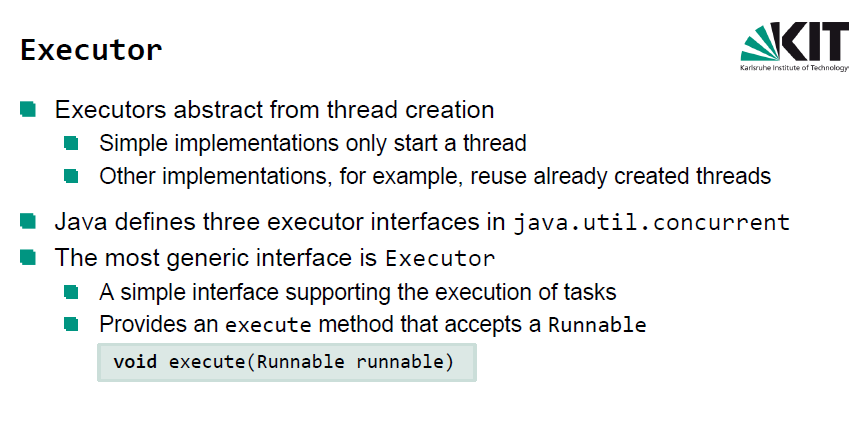
[ExecutorService Beispiel 5](#_Toc67312715)

# Parallelprogrammierung

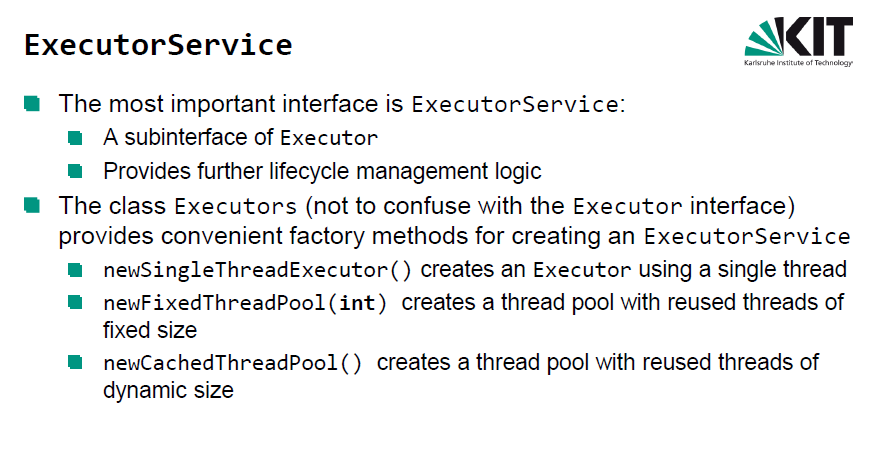
## Amdahlsches Gesetz



## Happens-before-Beziehung

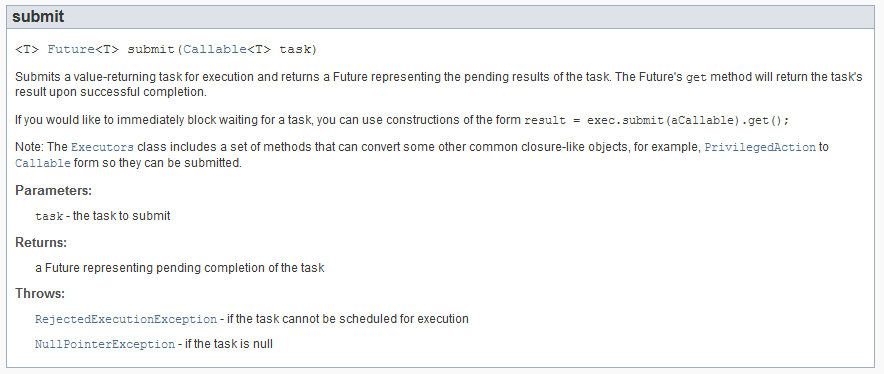
Executor

## ExecutorService



### submit

Future <T> submit(task):



task == lambda Funktion

Verwendung mit Lambda:

service.submit(() -> calculateX(elementIdx));

gibt Future<Integer> zurück, der später mit future.get() in einer Integer Variable geschrieben sein kann.

calculateX(elementIdx) ist eine Methode, die ein Integer zurückgibt.

Fututre<Integer> future = service.submit(() -> calculateX(elementIdx));

...

Integer result = future.get();

## Futures

## Berechnung mit ExecutorService und Futures

### Vorgehensweise

1. Zuerst mus man ein ExecutorService erstellen und Anzahl von Executors einstellen.

ExecutorService service = Executors.newFixedThreadPool(amountThreads);

2. Dann wird eine Liste von Futures erstellt

List<Future<Integer>> futures = new ArrayList<Future<Integer>>();

3. Iterire über Eingabedaten und fülle die Liste von Futures.

**Wichtig**: Anzahl der Iterationen (range von i) == Anzahl der Threads

for (int i = 0; i < input.size(); i += i) {

final int elemeintIdx = i;

futures.add(service.submit(() -> calculateX(elementIdx)));

}

4. Dann iteriere über liste von Futures und berechne das Programmergebnis:

int count = 0;

for(Future<Integer> future: futures) {

count += future.get();

}

### Complex Case

final int target = Integer.parseInt(args[0]);

final int amountThreads = Integer.parseInt(args[1]);

ExecutorService service = Executors.newFixedThreadPool(amountThreads);

for (int i = SEARCH\_BEGIN; i < target; i += BLOCK\_SIZE) {

final int from = i;

final int until = i + BLOCK\_SIZE;

futures.add(service.submit(() -> countPrimes(from, until)));

}

int count = 0;

for(Future<Integer> future: futures) {

count += future.get();

}

# Java Advanced

## Streams

## Optional

# Design by Contract

# Bespiele

## Amdahlsches Gesetz aus Ü9

Aufgabe: geg. Thread Pool. Jeder Leser und Schreiber werden repräsentiert durch einen Thread. 90% Threads sind Leser, 10% sind Schreiber.

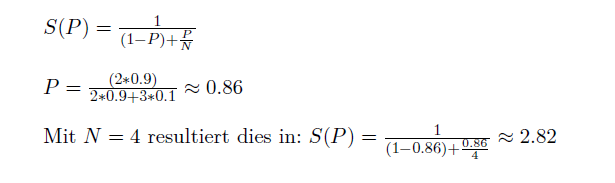
Leser sind nicht blockierend, benötigt 2 Sekunden

Schreiber ist blockierend für alle Schreiber und Lese, benötigt 3 Sekunden.

Lösung:

*P*: Anteil eines Programms, der parallelisiert werden kann

*N*: Anzahl der Prozessoren



P = P(Leser)/P(Gesamt)

## ExecutorService Beispiel