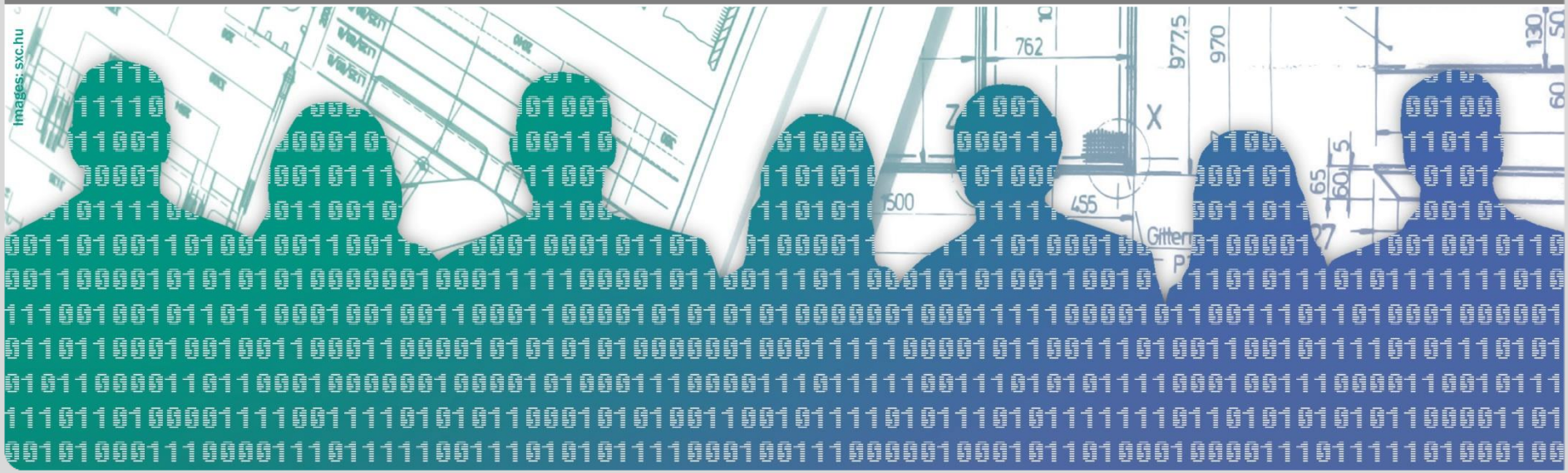
Karlsruher Institut für Technologie

Automatisierte Parallelisierung mit Auto-Futures

Studienarbeit von Jochen Huck
Betreuer: Frank Otto, Korbinian Molitorisz, Jochen Schimmel

IPD Tichy, Fakultät für Informatik

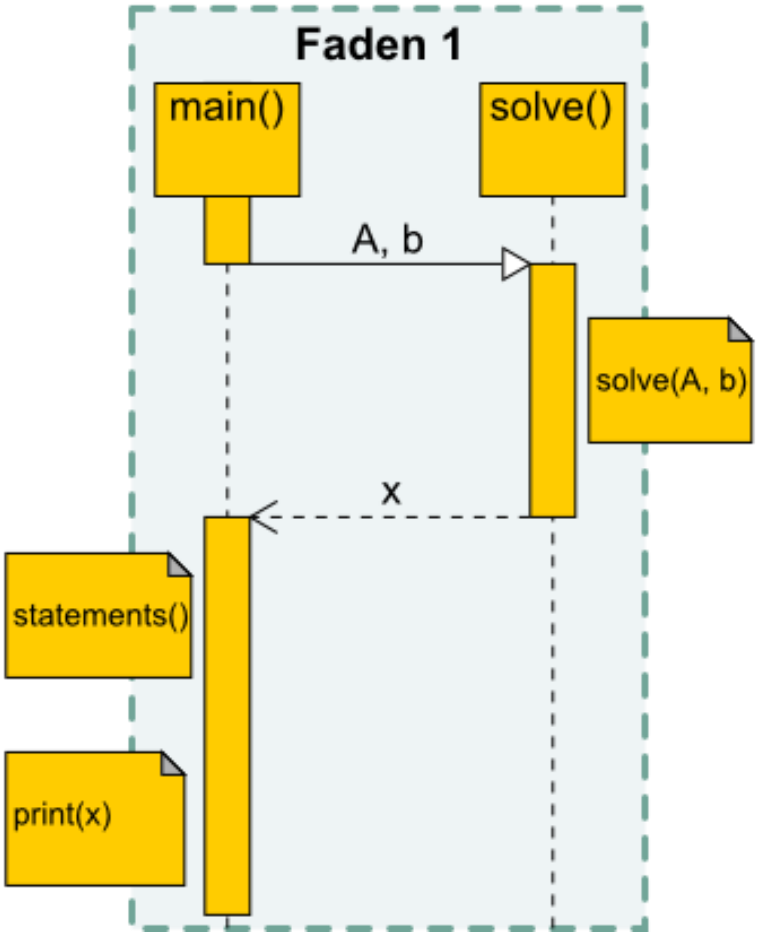


Parallelisierung mit Futures

```
main() {
   x = solve(A, b);
   statements();
   print(x);
}
```

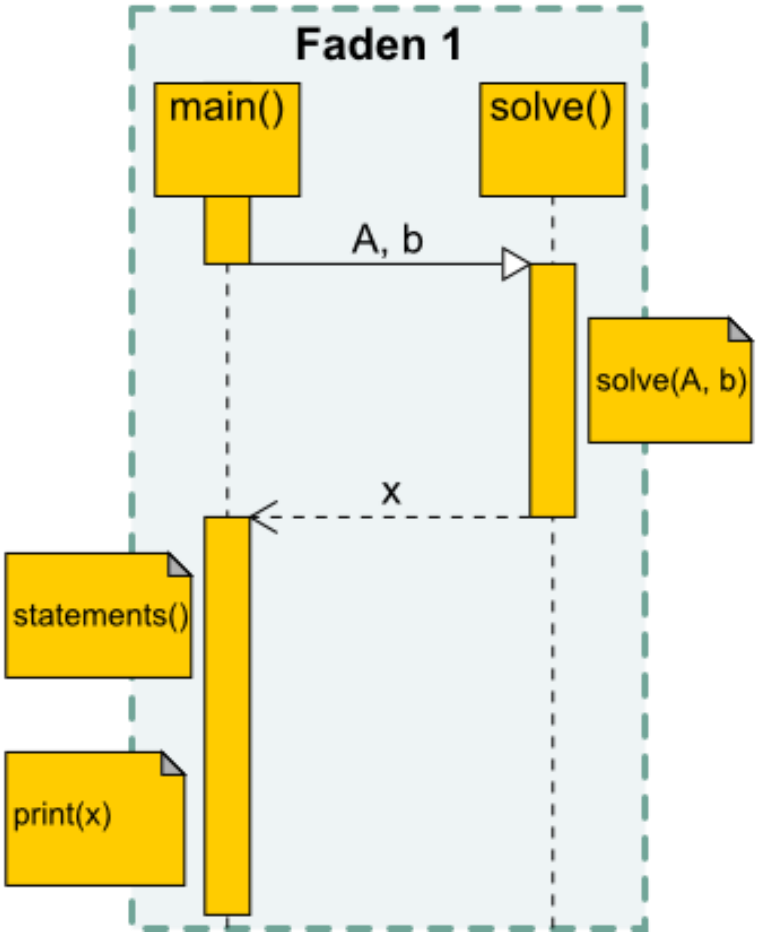
sequentiell

Parallelisierung mit Futures

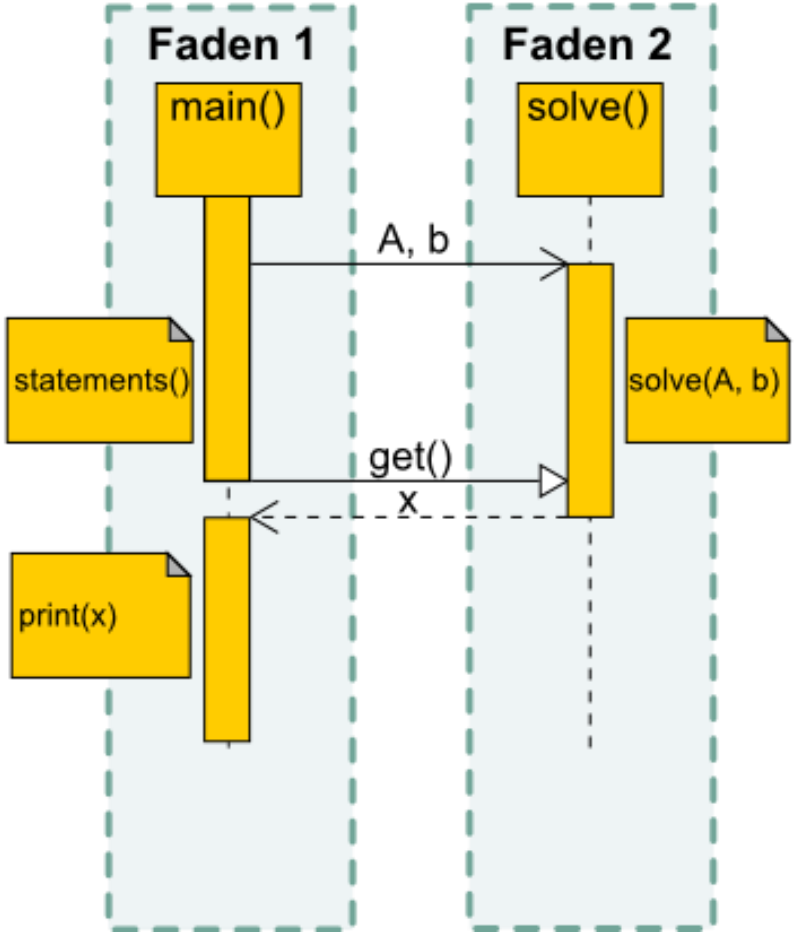


sequentiell

Parallelisierung mit Futures

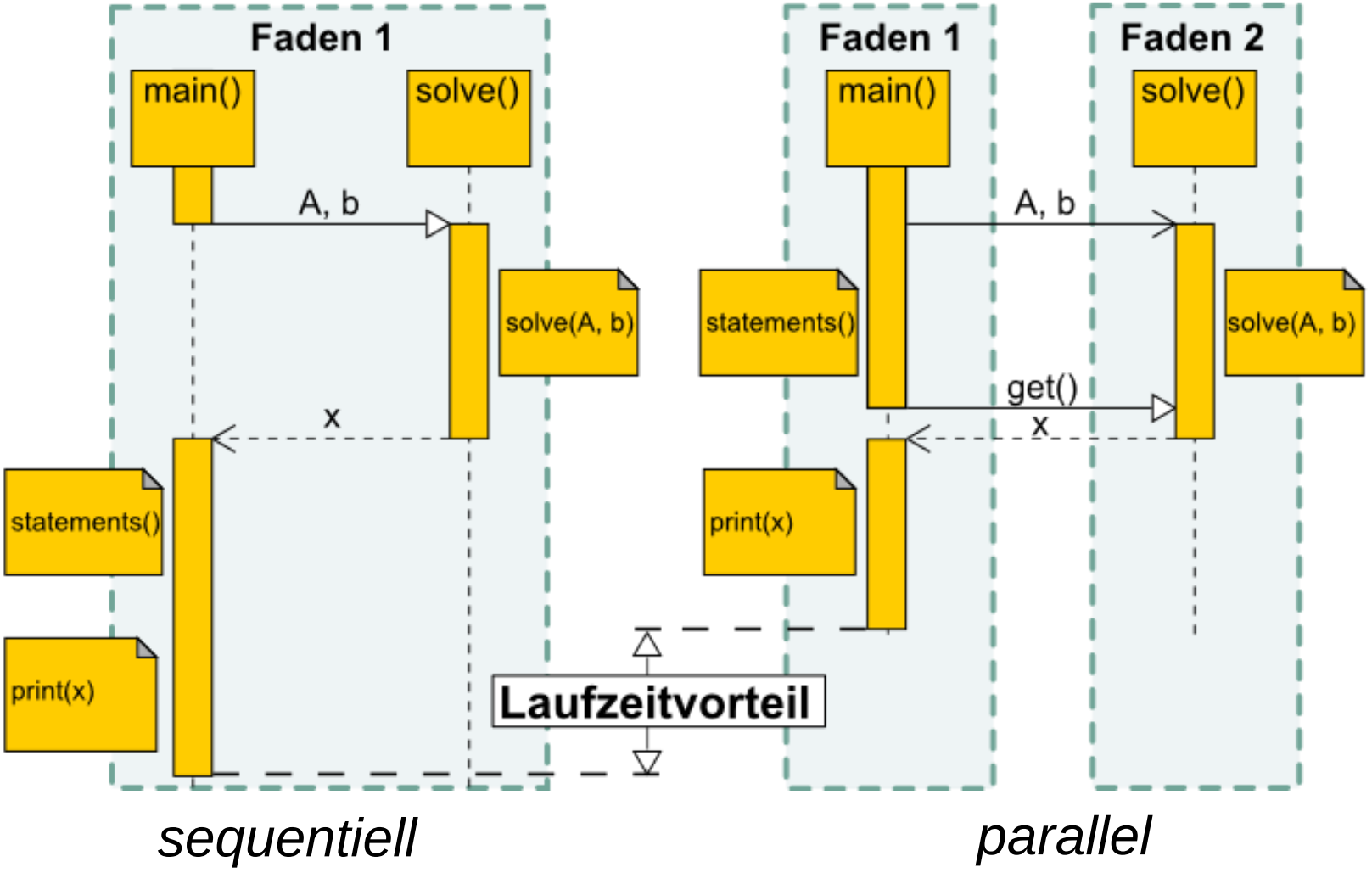


sequentiell



parallel

Parallelisierung mit Futures



Parallelisierung mit Futures

```
main() {
   x = solve(A, b);
   statements();
   print(x);
}
```

sequentiell

```
main() {
   f = async(solve(A, b));
   statements();
   x = get(f);
   print(x);
}
```

parallel

Automatisierte Parallelisierung mit Auto-Futures

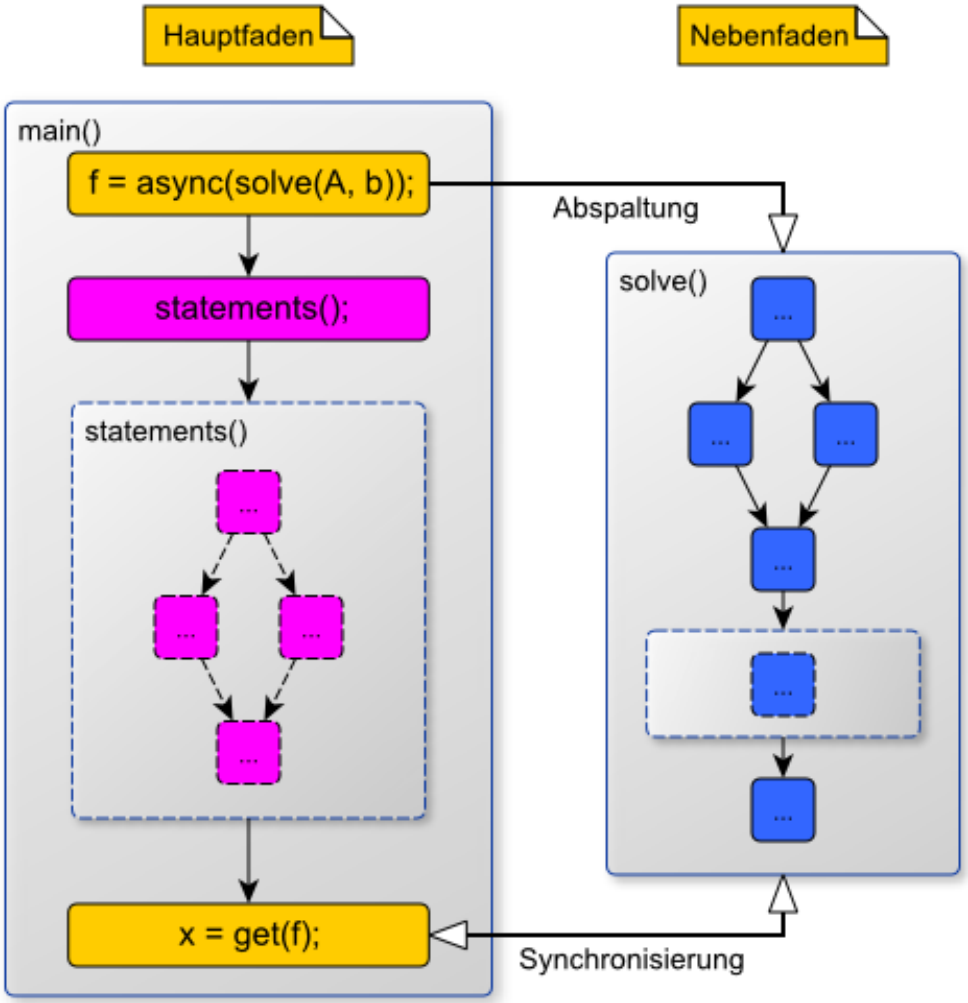
```
main() {
  x = solve(A, b);
  statements();
  print(x);
}
```

sequentiell

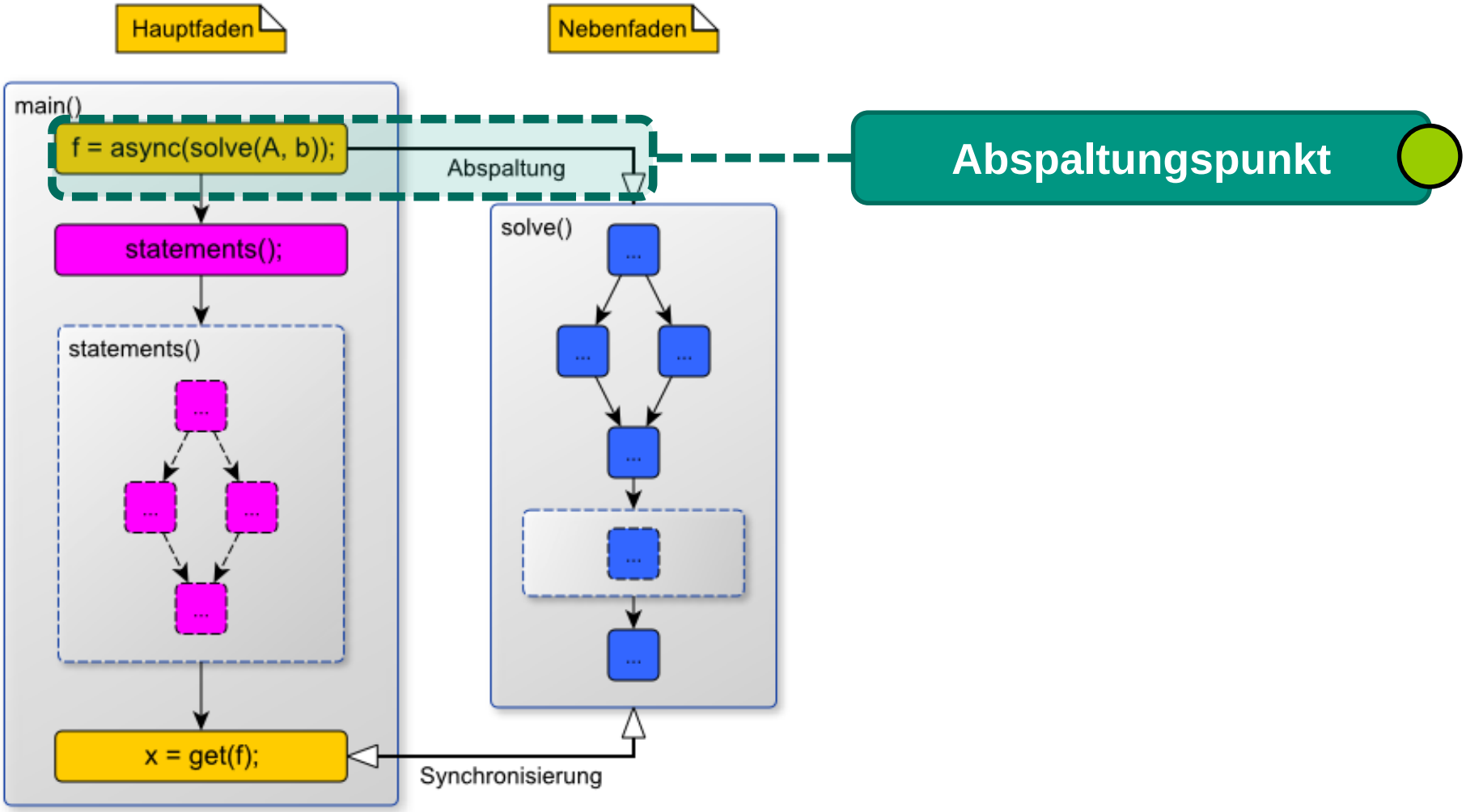
```
main() {
  f = async(solve(A, b));
  statements();
  x = get(f);
  print(x);
}
```

parallel

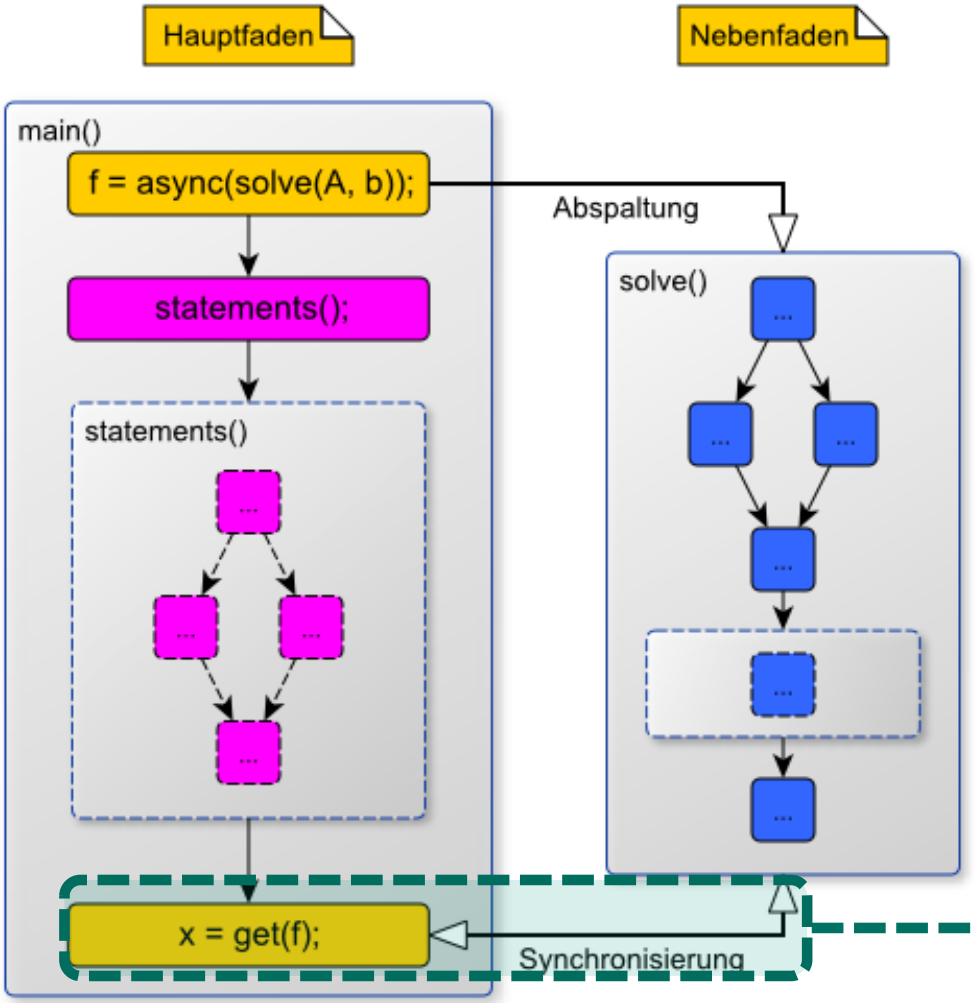
Definitionen



Definitionen



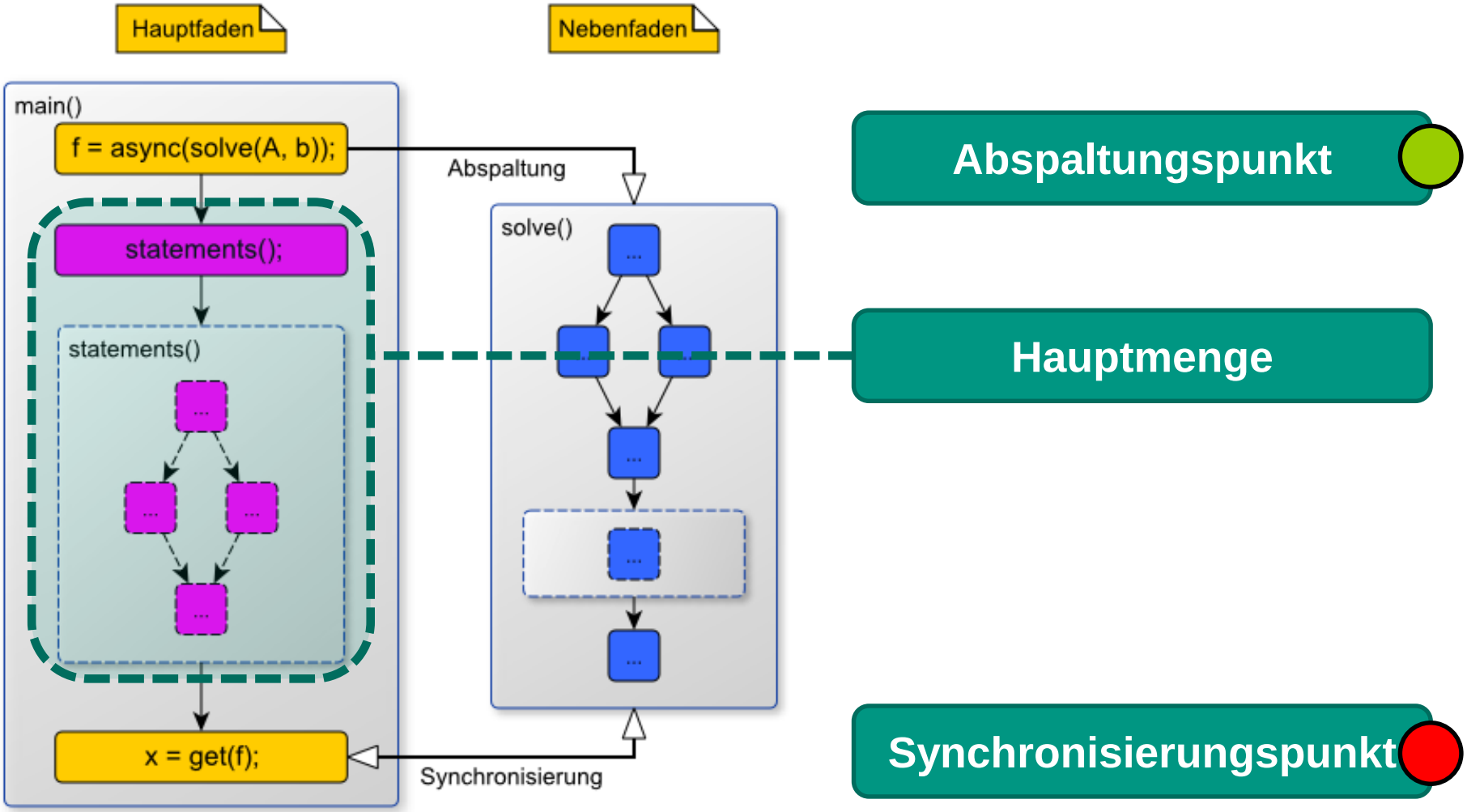
Definitionen



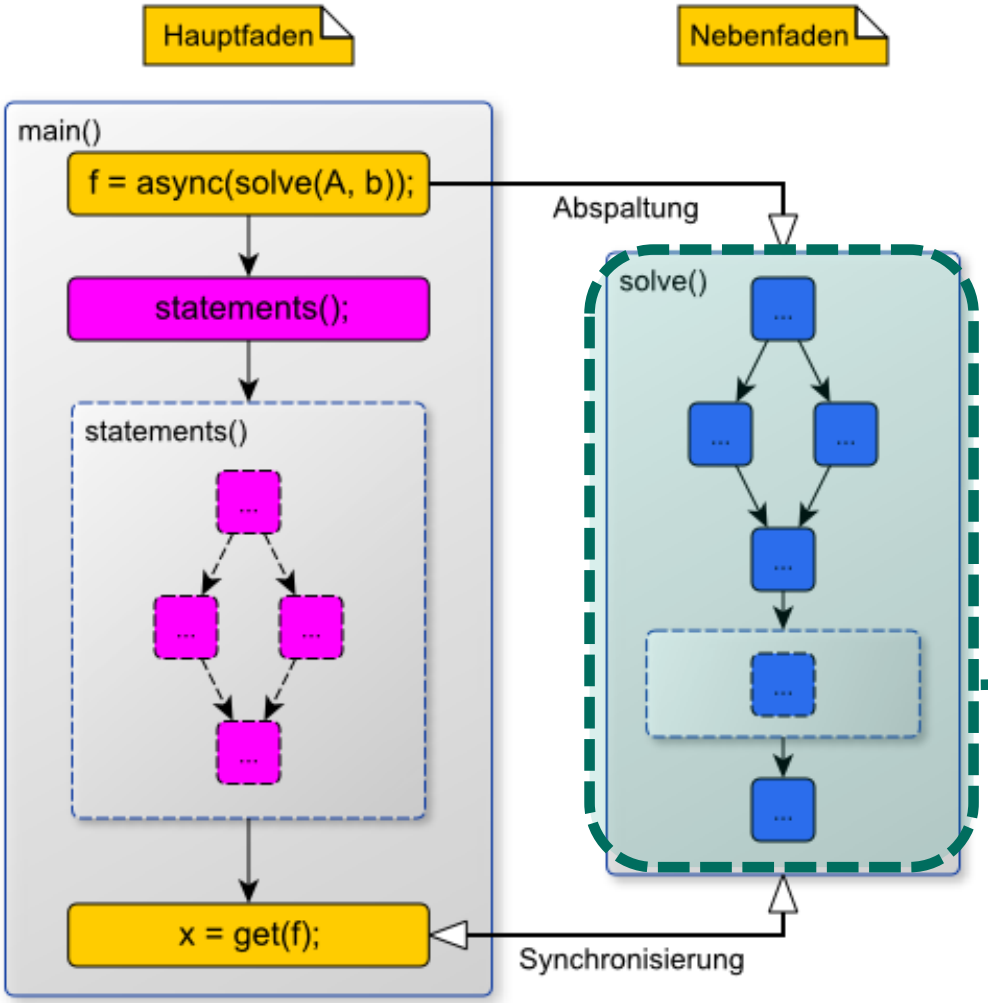
Abspaltungspunkt

Synchronisierungspunkt

Definitionen



Definitionen



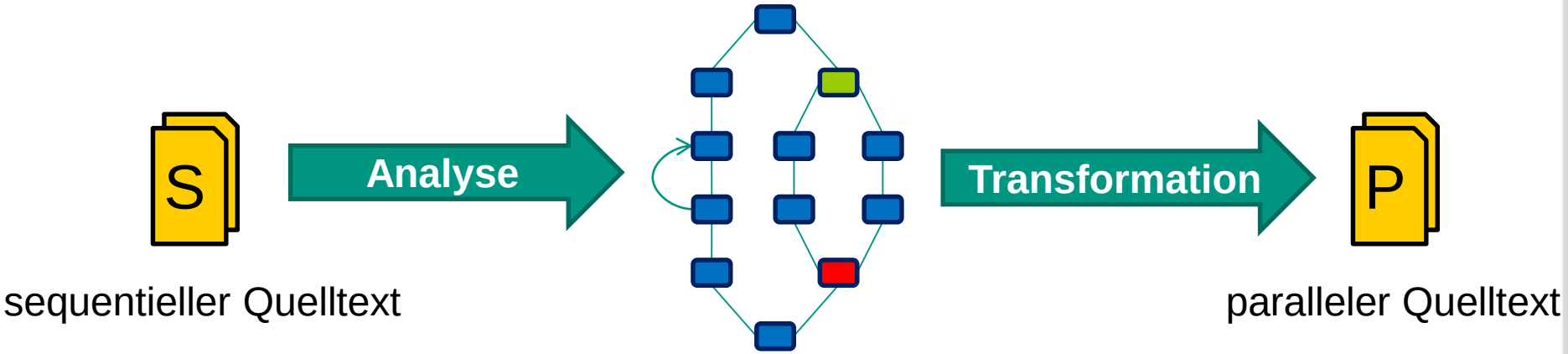
Abspaltungspunkt

Hauptmenge

Nebenmenge

Synchronisierungspunkt

Schematischer Überblick

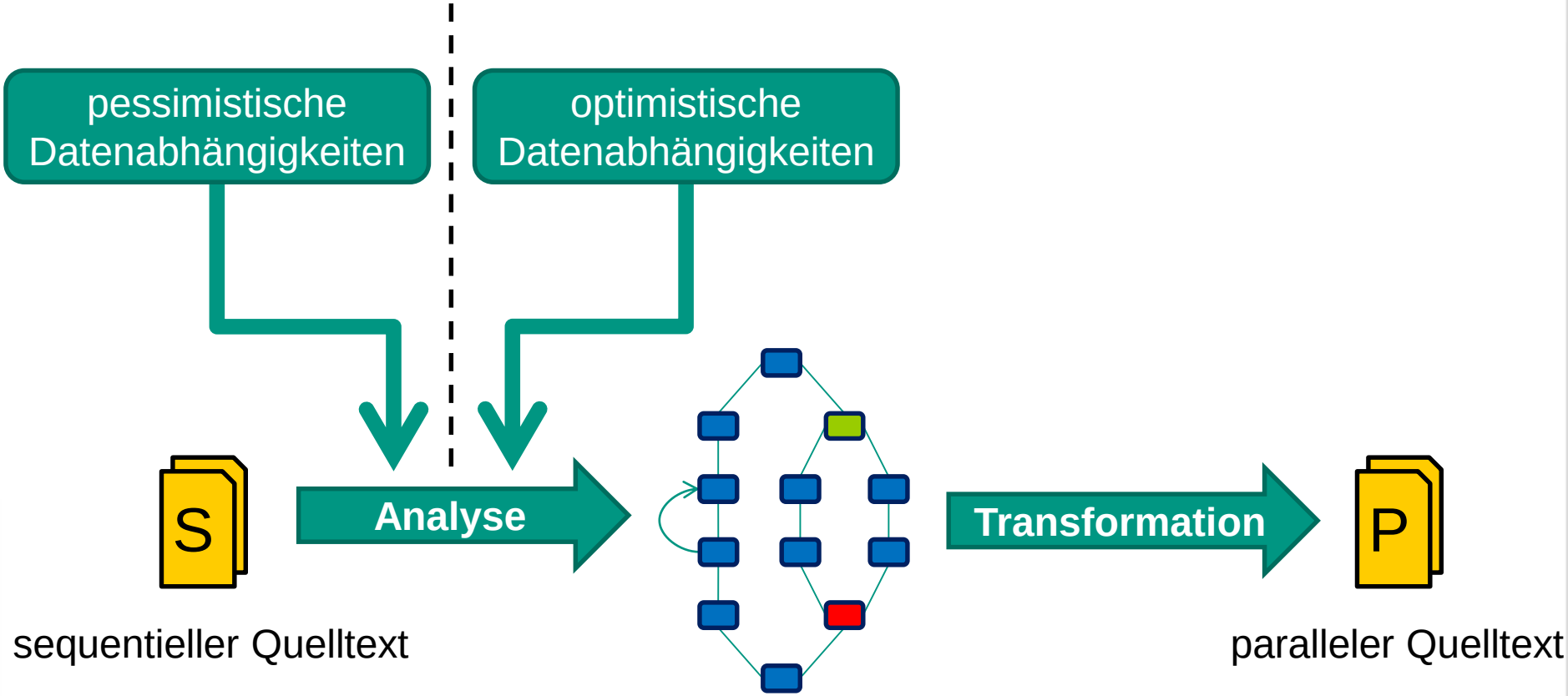


sequentieller Quelltext

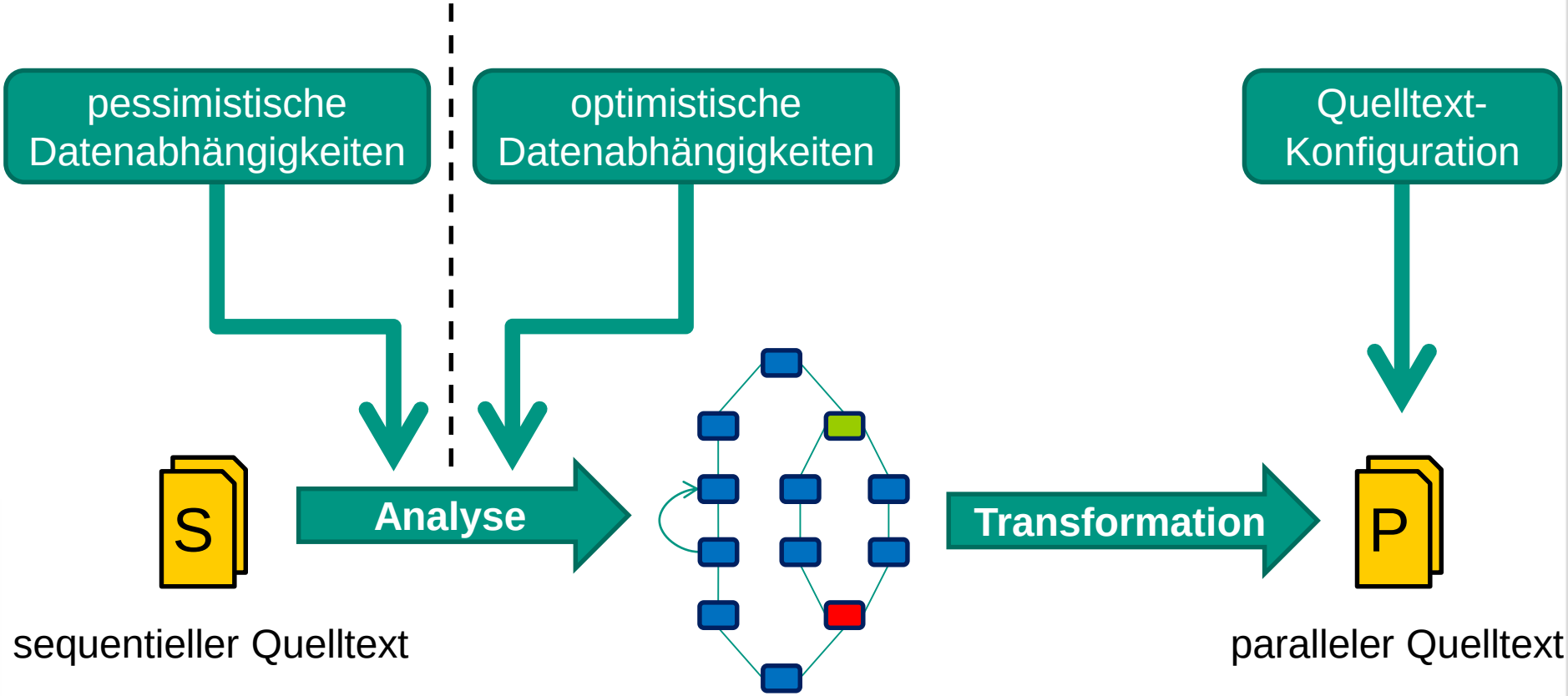
paralleler Quelltext

Zwischenrepräsentation
grün: Abspaltungspunkt
rot: Synchronisierungspunkt

Schematischer Überblick



Schematischer Überblick



Reale Datenabhängigkeiten

```
class Foo {
  int i;

  bar() {
    i++;
  }

  main() {
    Foo f1 = new Foo();
    Foo f2 = new Foo();
    f1.bar();
    f2.bar();
  }
}
```

R: {f1.i}, W: {f1.i}

R: {f1.i}, W: {f1.i}

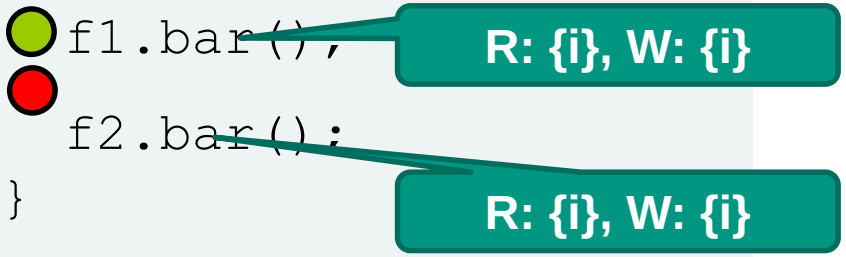
Pessimistische Datenabhängigkeiten

- Seiteneffektanalyse: kontext-insensitiv

```
class Foo {
  int i;

  bar() {
    i++;
  }

  main() {
    Foo f1 = new Foo();
    Foo f2 = new Foo();
    f1.bar();
    f2.bar();
  }
}
```



Pessimistische Datenabhängigkeiten

- Seiteneffektanalyse: kontext-insensitiv

```
class Foo {
  int i;

  bar() {
    i++;
  }

  main() {
    Foo f1 = new Foo();
    Foo f2 = new Foo();
    f1.bar();
    f2.bar();
  }
}
```

Triviale Parallelisierung

R: {i}, W: {i}

R: {i}, W: {i}

Optimistische Datenabhängigkeiten

- Seiteneffektanalyse & Heuristiken

```
class Foo {
  int i;

  bar() {
    i++;
  }

  main() {
    Foo f1 = new Foo();
    Foo f2 = new Foo();
    f1.bar();
    f2.bar();
  }
}
```

R: {i}, W: {i}

R: {i}, W: {i}

Heuristik:

- gleiche Methode
- keine Argumente
- unterschiedliche Objekte

Optimistische Datenabhängigkeiten

- Seiteneffektanalyse & Heuristiken

```
class Foo {
  int i;

  bar() {
    i++;
  }

  main() {
    Foo f1 = new Foo();
    Foo f2 = new Foo();
    f1.bar();
    f2.bar();
  }
}
```

R: {i}, W: {i}

R: {i}, W: {i}

Heuristik:

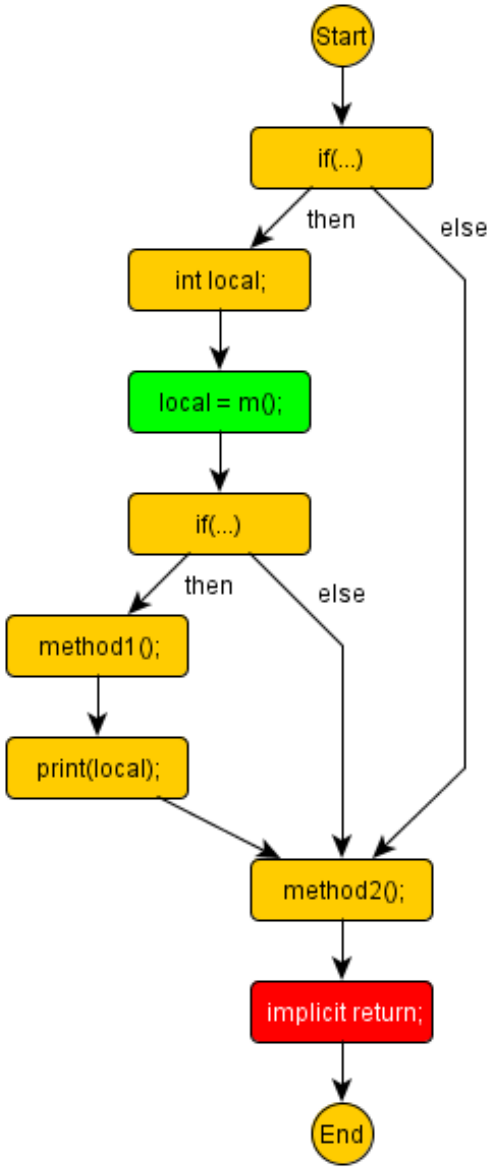
- gleiche Methode
- keine Argumente
- unterschiedliche Objekte

Analyse

1. return-Anweisungen

```
main() {
  if(…) {
    int local;
    local = m();
    if(…) {
      method1();
      print(local);
    }
  }
  method2();
  return;
}
```

sequentiell



Analyse

2. transitive Abhängigkeiten

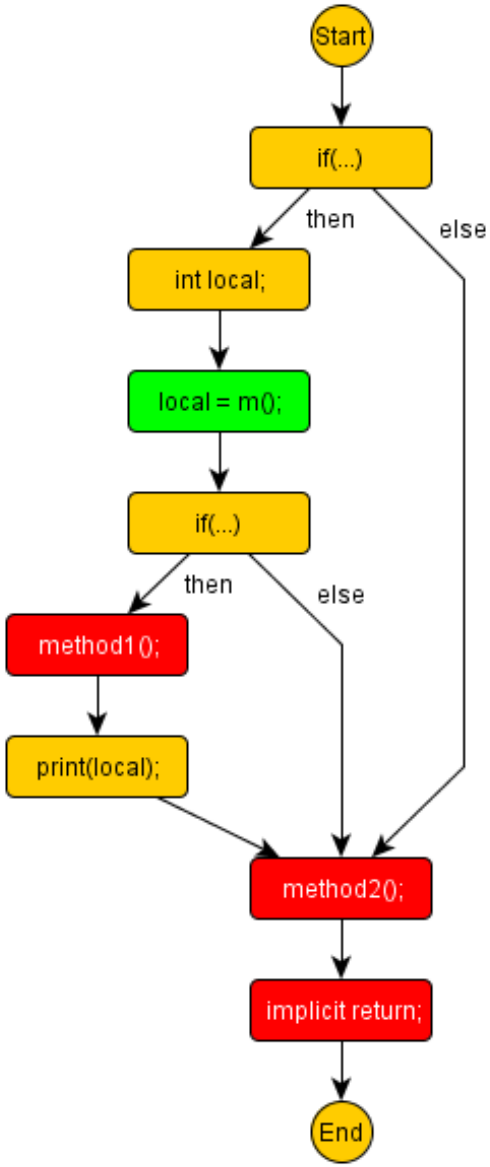
```
main() {
  if(…) {
    int local;
    local = m();
    if(…) {
      method1();
      print(local);
    }
  }
  method2();
  return;
}
```

R: {a, b}, W: {c}

R: {a}, W: {c}

R: {local}, W: {}

R: {b}, W: {a, c}



sequentiell

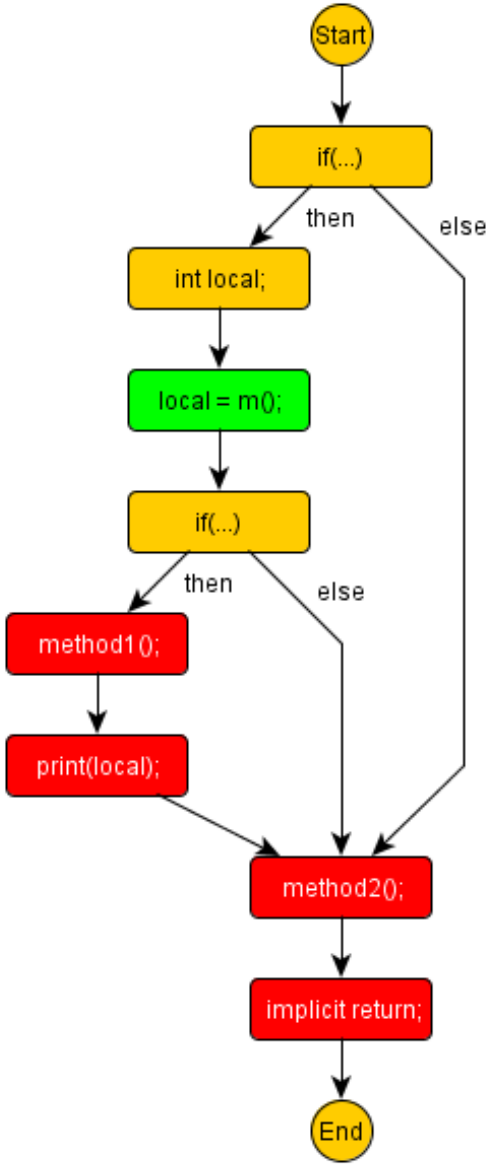
Analyse

3. lokale Abhängigkeiten

```
main() {
  if(…) {
    int local;
    local = m();
    if(…) {
      method1();
      print(local);
    }
  }
method2();
return;
}
```

R: {}, W: {local}

R: {local}, W: {}



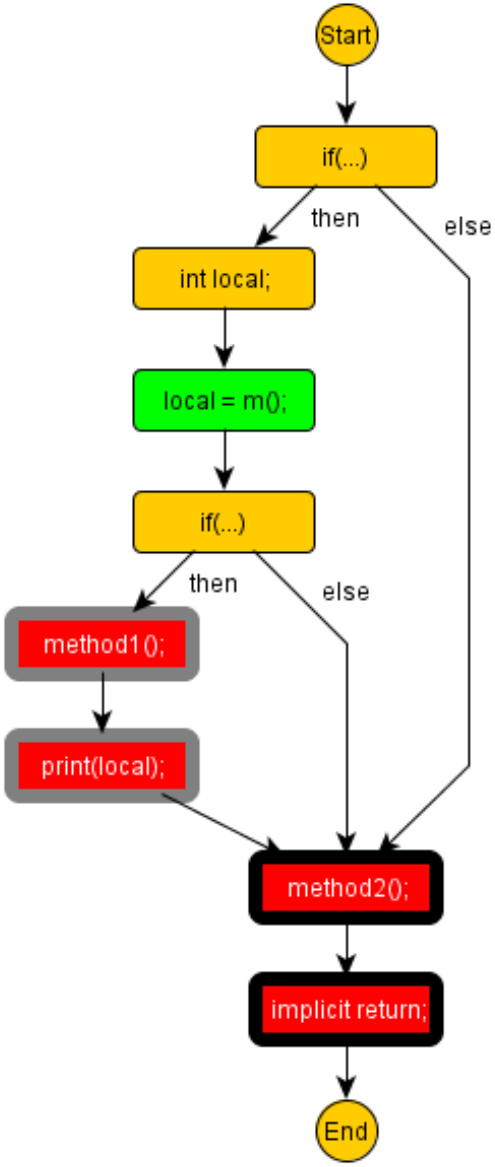
sequentiell

Analyse

4. Klassifizierung

```
main() {
  if(…) {
    int local;
    local = m();
    if(…) {
      method1();
      print(local);
      }
  }
method2();
return;
}
```

sequentiell

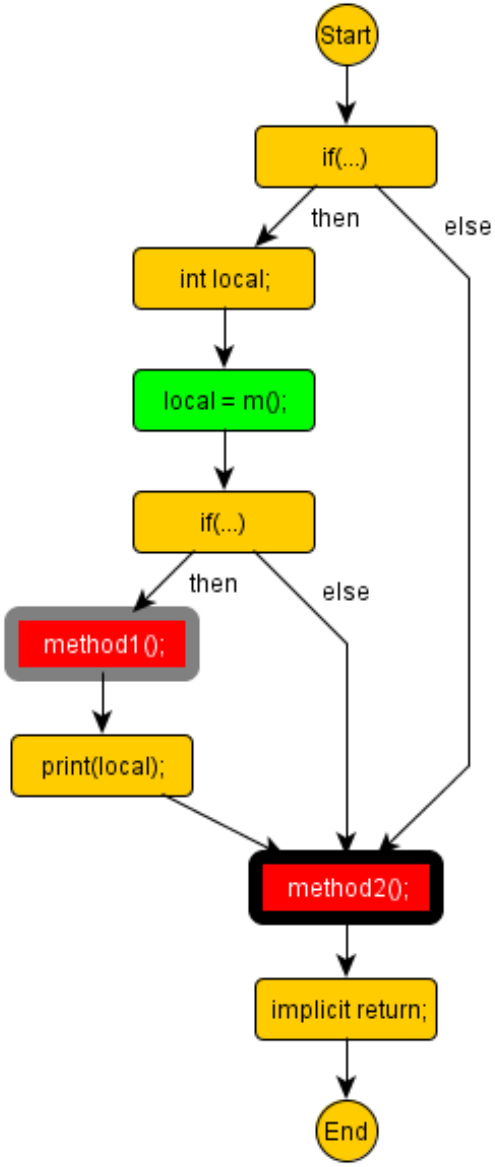


Analyse

5. Knoten eliminieren

```
main() {
  if(…) {
    int local;
    local = m();
    if(…) {
      method1();
      print(local);
    }
  }
  method2();
  return;
}
```

sequentiell



Transformation

```
main() {
  if(…) {
    int local;
    local = m();
    if(…) {
      method1();
      print(local);
    }
  }
  method2();
  return;
}
```

```
main() {
  if(…) {
    int local;
    f = async(m());
    if(…) {
      local = get(f);
      method1();
      print(local);
    }
  }
  get(f);
  method2();
  return;
}
```

sequentiell *parallel*

Evaluation

P: pessimistische Datenabhängigkeiten
O: optimistische Datenabhängigkeiten

	Mergesort	Matrix	PMD	ANTLR	ImageJ
LOC	34	81	44782	36733	93899
Transformationen	1	21	189	1043	3351
alle synchron	0,95	1,00	0,95	1,00	1,00
P, alle asynchron	0,024	0,99	0,01	0,004	0,98
P, opt. asynchron	0,95	1,00	0,95	1,00	1,00
O, opt. asynchron	**2,70**	**3,34**	0,95*	1,00*	**2,04**

Verwandte Arbeiten

[BC] Bryan Chan: *Run-Time Support for the Automatic Parallelization of Java Programs*

[RR] Radu Rugina und Martin Rinard: *Automatic parallelization of divide and conquer algorithms*

[TF] Georgios Tournavitis und Björn Franke: *Semi-automatic extraction and exploitation of hierarchical pipeline parallelism using profiling information*

Zusammenfassung

- Parallelisierung mit pessimistischen Datenabhängigkeiten
  - Bisher keine Beschleunigung
- Parallelisierung mit optimistischen Datenabhängigkeiten
  - Beschleunigungen von bis zu 3.3
- Verbesserungen:
  - Kontext-sensitive Seiteneffektanalyse
  - Erkennung trivialer Parallelisierung

Danke für die Aufmerksamkeit!

Gibt es Fragen?

Heuristiken

- Teile und Herrsche:

```
sort(int[] array) {
  …
  sort(left);
  sort(right);
  merge(left, right);
}
```

isCritical(`sort(left), sort(right)`***)*** = false

Heuristiken

- Simple Assign

```
main() {
  …
  left = sort();
  right = sort();
}
```

isCritical(`left = sort(), right = sort()`***)*** = false

Heuristiken

- Simple Invoke

```
main() {
   …
   left.sort();
   right.sort();
}
```

Falls `left` und `right` auf verschiedene Objekte zeigen:

isCritical(`left.sort(), right.sort()`***)*** = false

Objekt . Methode (Arg_1 , …, Arg_n);

```
class CallableKlasse implements Callable < Rückgabetyp > {
  private Typ0 Objekt ;
  private Typ1 Arg1 ;
    …
  private Typn Argn ;
  public CallableKlasse( Typ0 Objekt , Typ1 Arg1 , … , Typn Argn ) {
    this. Objekt = Objekt ;
    this. Arg1 = Arg1 ;
     …
    this. Argn = Argn ;
  }
  public Rückgabetyp call() {
    return Objekt . Methode ( Arg1 , … , Argn ) ;
  }
}
```

```
main() {
 Future<int[]> f1 = null;
 if(U.t1.runSeq()) {
  x = solve(A, b);
 } else {
  class C1 { … }
  f1 = U.p.submit(new C1(A, b));
  U.t1.active.incrementAndGet();
 }
  …
 if(f1 != null) {
  x = f1.get();
  U.t1.active.decrementAndGet()
 }
}
```

```
class U {
 ExecutorService p;
 TP t1 = new TP();

 class TP {
  boolean seq;
  AtomicInteger active;

  boolean runSeq() {
   if(seq||active.get()>max) {
    seq = true;
    return true;
   } else {
    return false;
   }
  }
 }
}
```