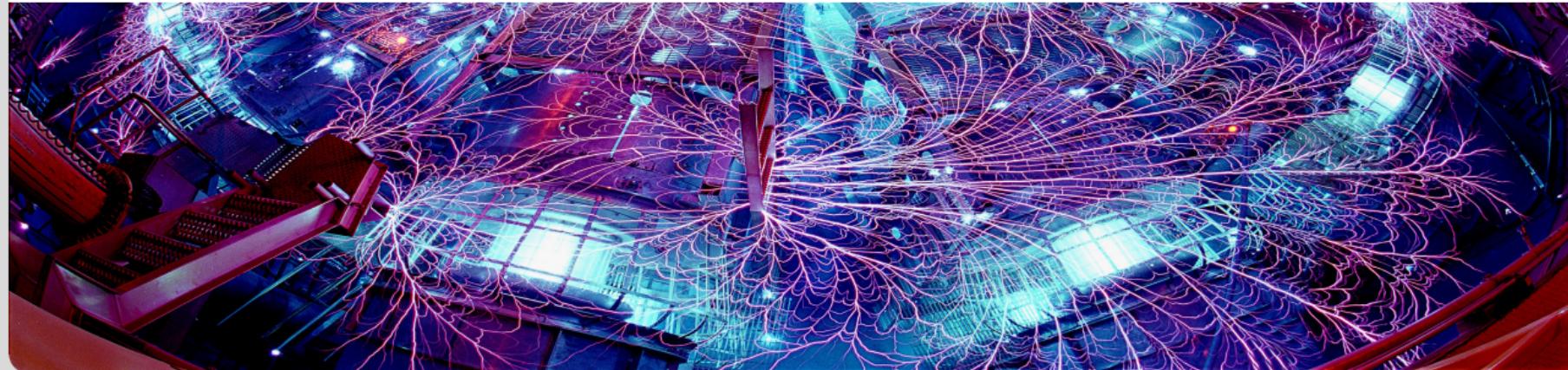


Parallele SCC-Erkennung

Miniseminar Parallele Algorithmen

Felix Karg, Konstantin Fickel | 03. Februar 2020

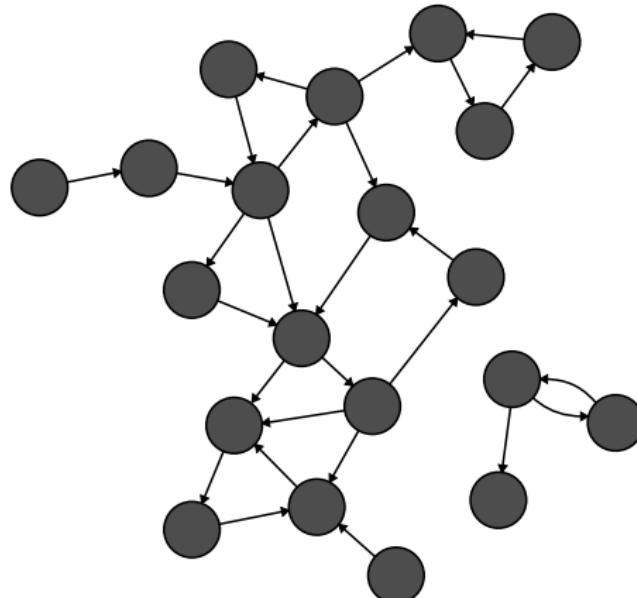
INSTITUT FÜR THEORETISCHE INFORMATIK



Starke Zusammenhangskomponenten

Gegenseitige Erreichbarkeit \sim_G

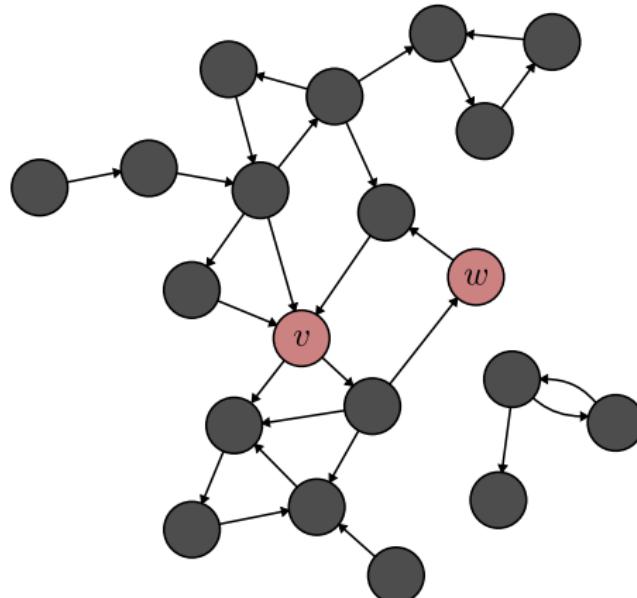
$$v \sim_G w \Leftrightarrow \exists \langle v, \dots, w \rangle \wedge \exists \langle w, \dots, v \rangle$$



Starke Zusammenhangskomponenten

Gegenseitige Erreichbarkeit \sim_G

$$v \sim_G w \Leftrightarrow \exists \langle v, \dots, w \rangle \wedge \exists \langle w, \dots, v \rangle$$



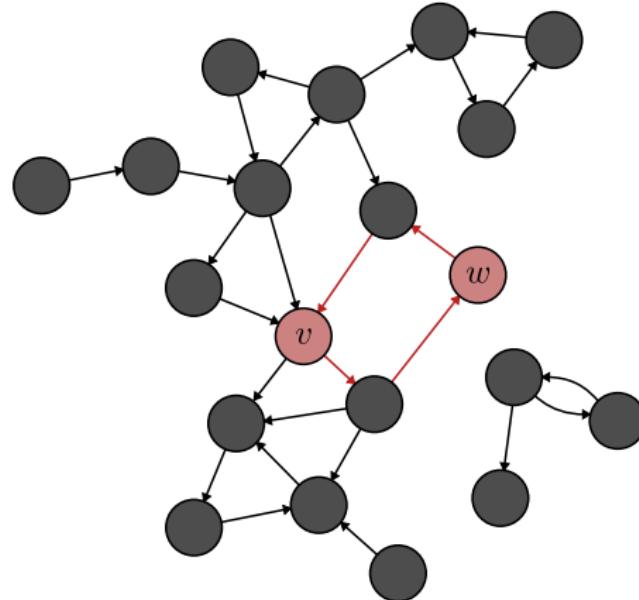
Starke Zusammenhangskomponenten

Gegenseitige Erreichbarkeit \sim_G

$$v \sim_G w \Leftrightarrow \exists \langle v, \dots, w \rangle \wedge \exists \langle w, \dots, v \rangle$$

Starke Zusammenhangskomponente

Äquivalenzklassen von \sim_G



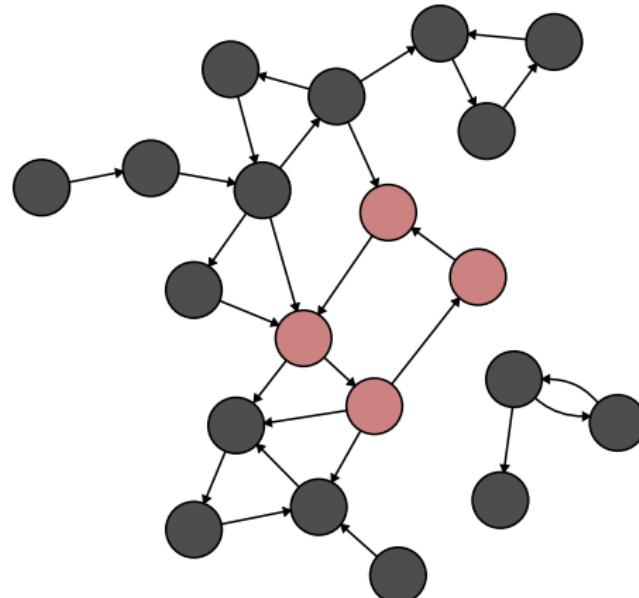
Starke Zusammenhangskomponenten

Gegenseitige Erreichbarkeit \sim_G

$$v \sim_G w \Leftrightarrow \exists \langle v, \dots, w \rangle \wedge \exists \langle w, \dots, v \rangle$$

Starke Zusammenhangskomponente

Äquivalenzklassen von \sim_G



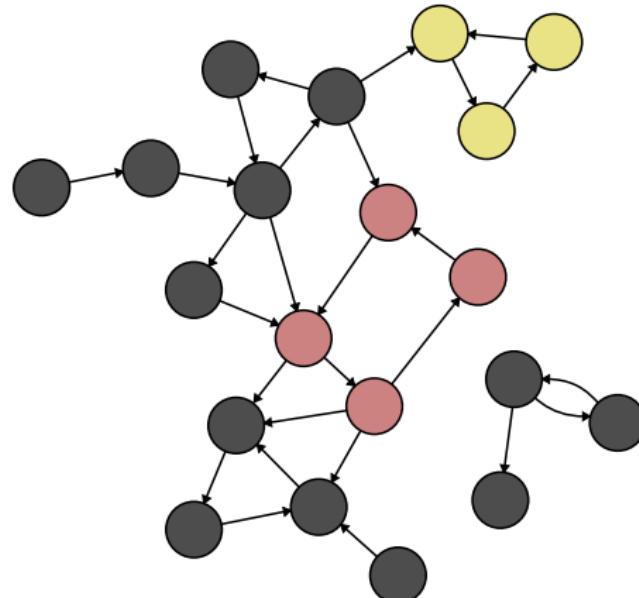
Starke Zusammenhangskomponenten

Gegenseitige Erreichbarkeit \sim_G

$$v \sim_G w \Leftrightarrow \exists \langle v, \dots, w \rangle \wedge \exists \langle w, \dots, v \rangle$$

Starke Zusammenhangskomponente

Äquivalenzklassen von \sim_G



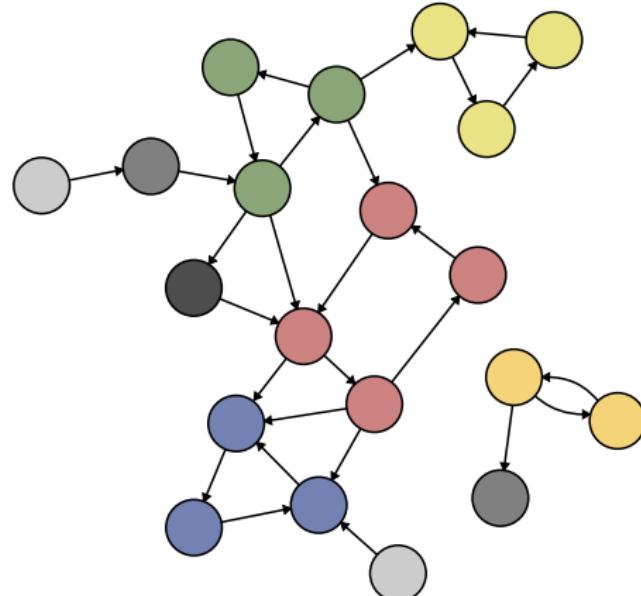
Starke Zusammenhangskomponenten

Gegenseitige Erreichbarkeit \sim_G

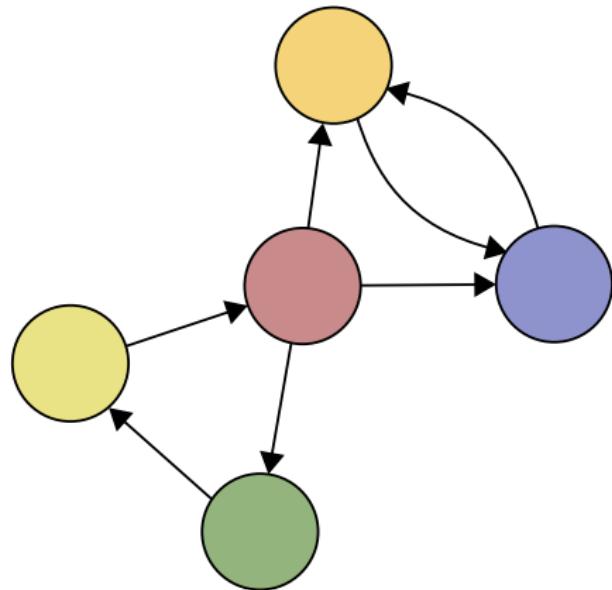
$$v \sim_G w \Leftrightarrow \exists \langle v, \dots, w \rangle \wedge \exists \langle w, \dots, v \rangle$$

Starke Zusammenhangskomponente

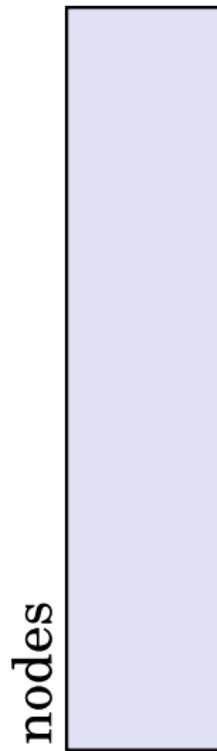
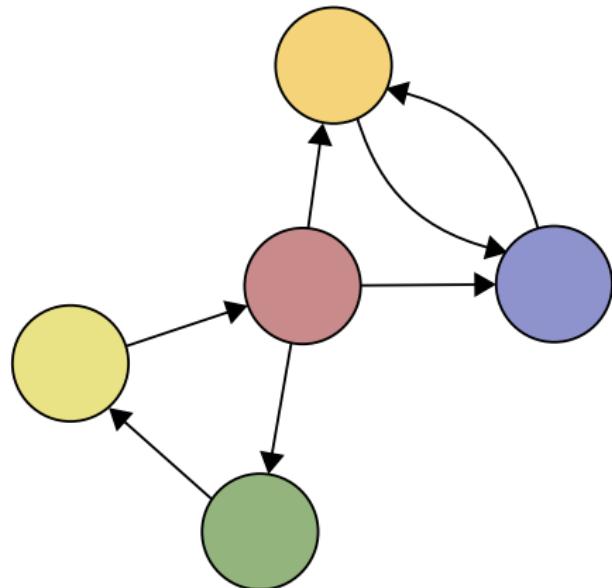
Äquivalenzklassen von \sim_G



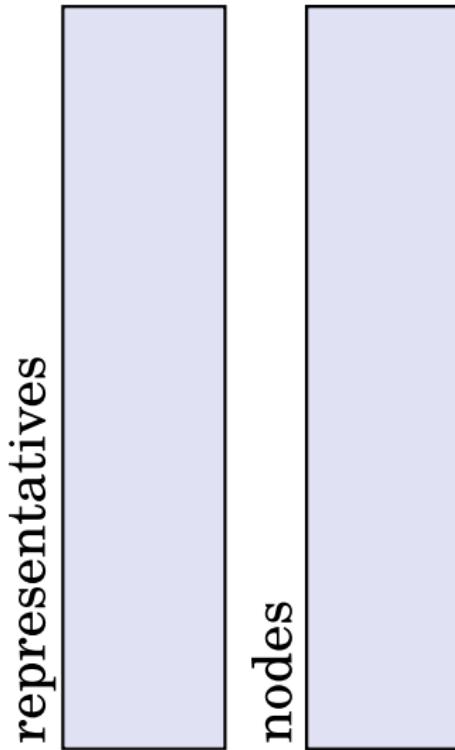
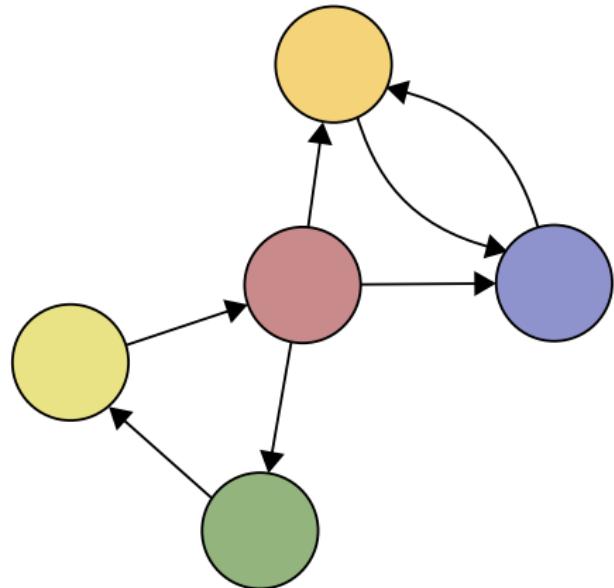
SCC-Erkennung (Cherian/Mehlhorn)



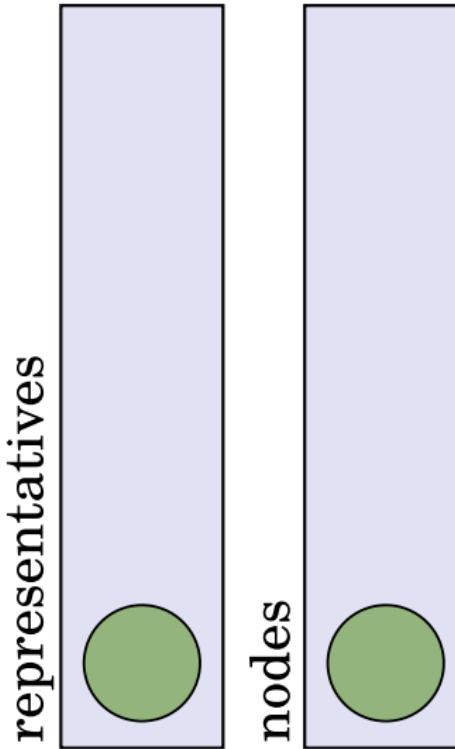
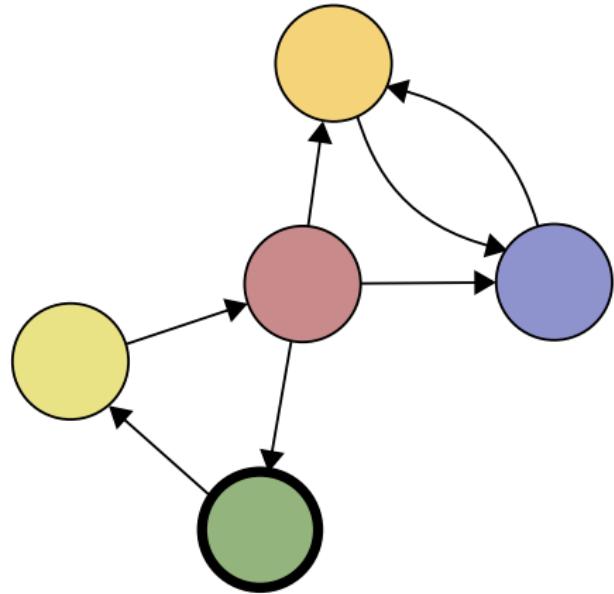
SCC-Erkennung (Cherian/Mehlhorn)



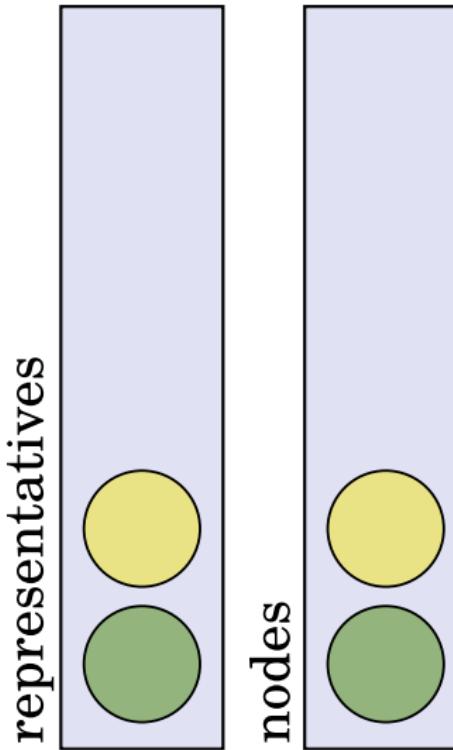
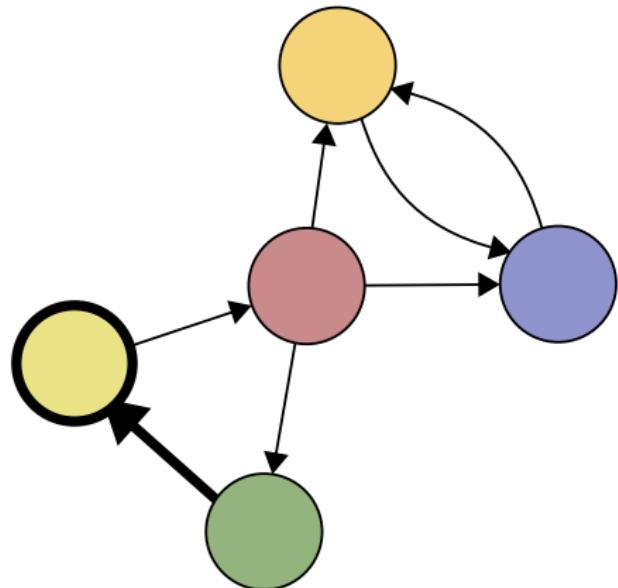
SCC-Erkennung (Cherian/Mehlhorn)



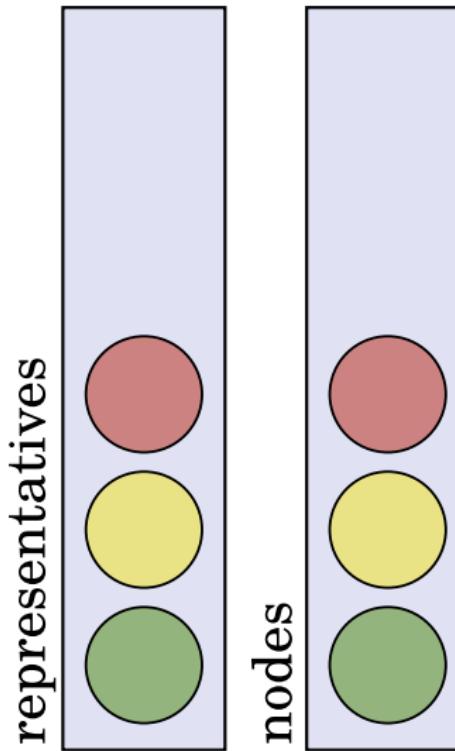
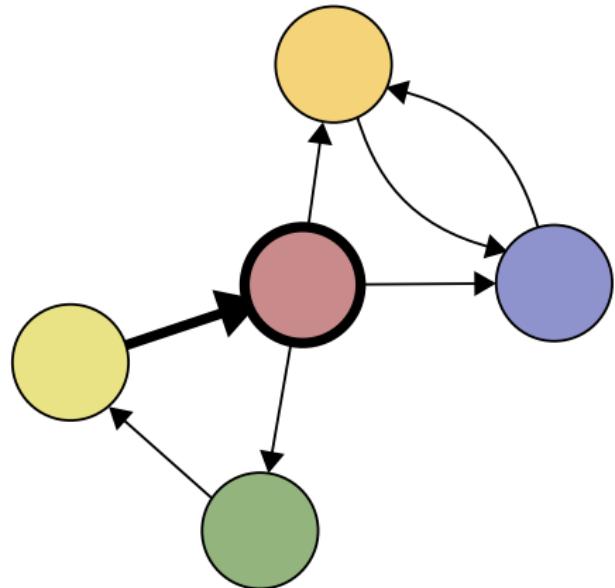
SCC-Erkennung (Cherian/Mehlhorn)



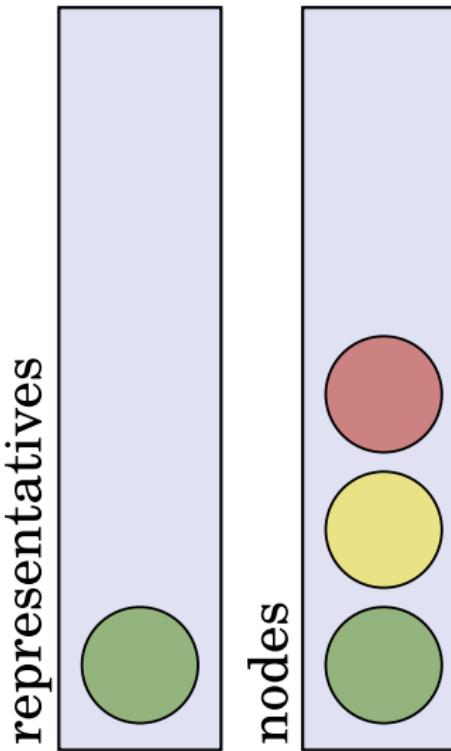
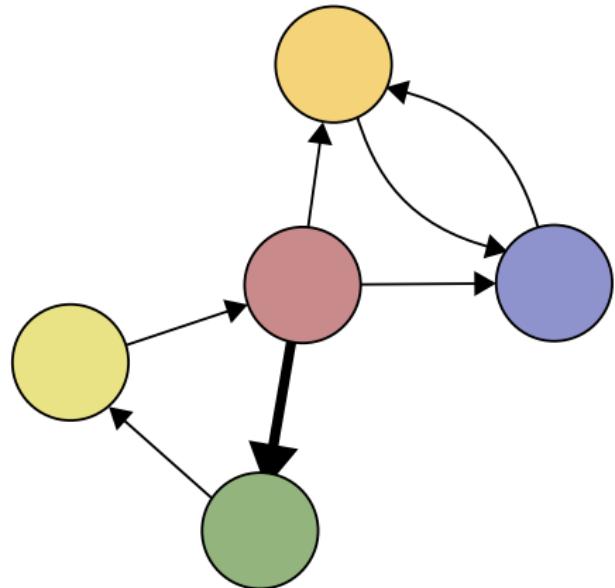
SCC-Erkennung (Cherian/Mehlhorn)



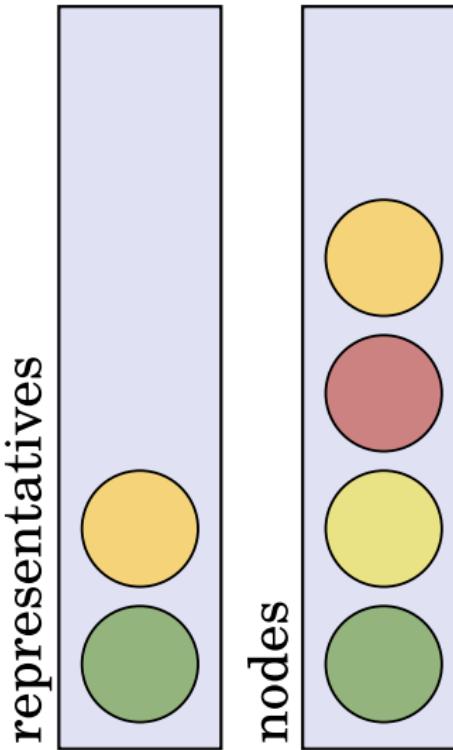
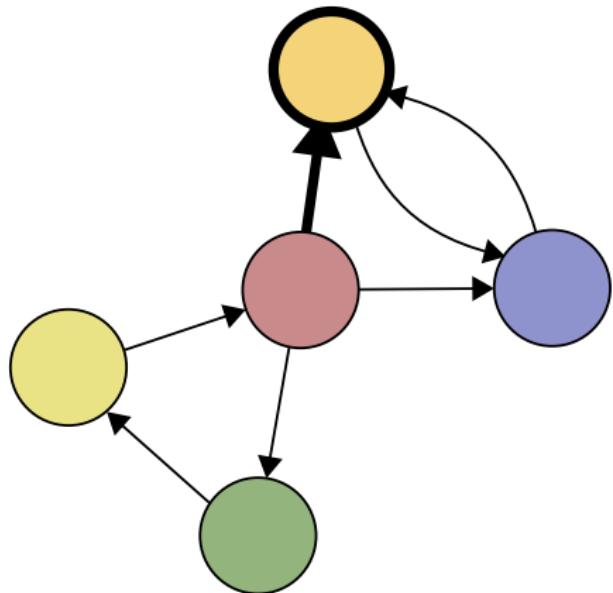
SCC-Erkennung (Cherian/Mehlhorn)



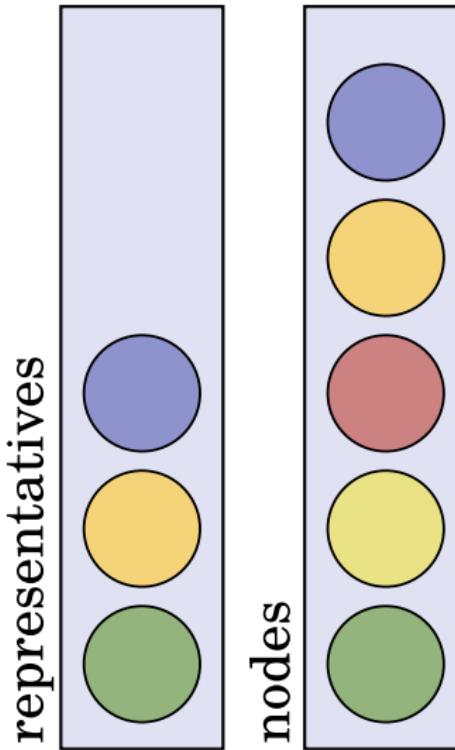
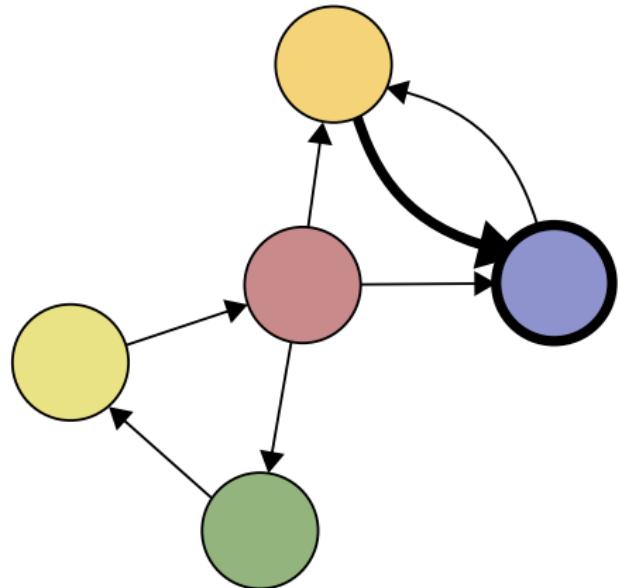
SCC-Erkennung (Cherian/Mehlhorn)



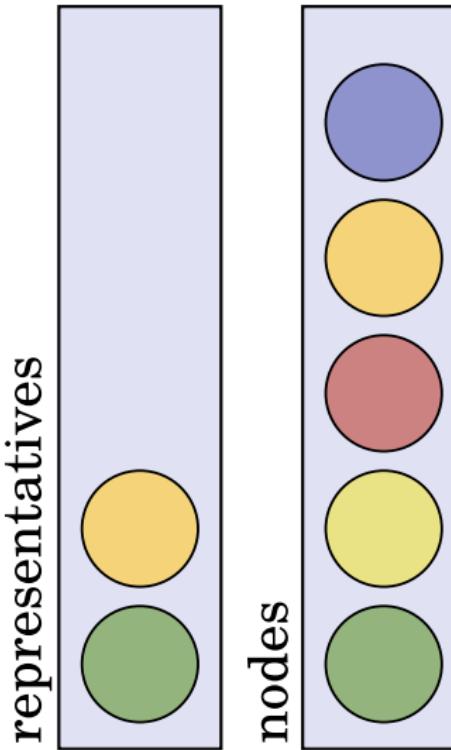
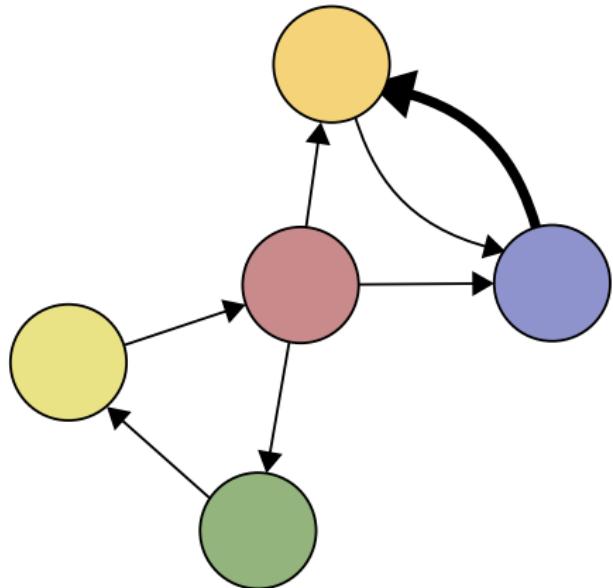
SCC-Erkennung (Cherian/Mehlhorn)



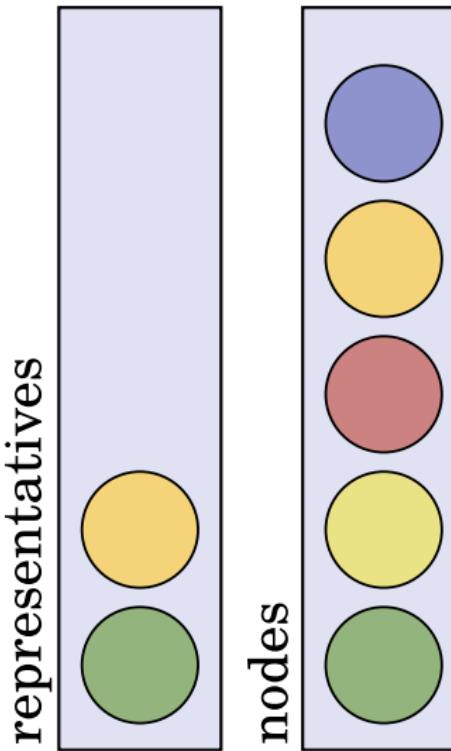
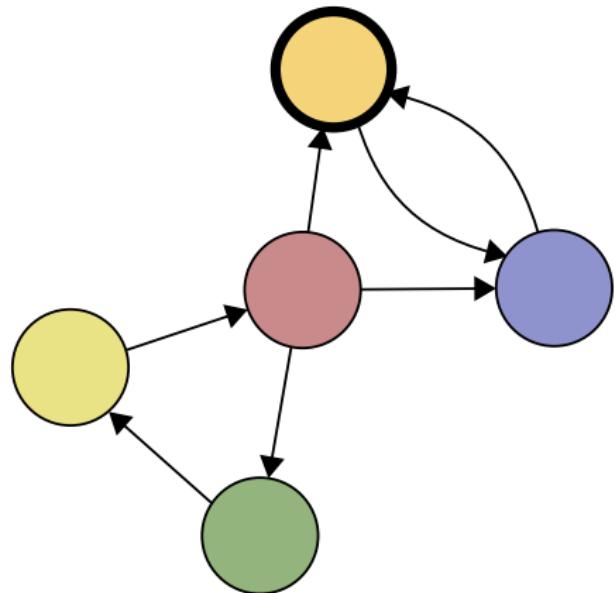
SCC-Erkennung (Cherian/Mehlhorn)



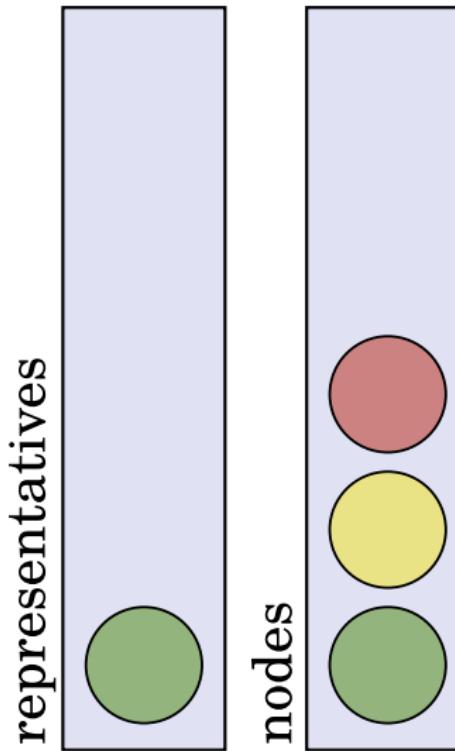
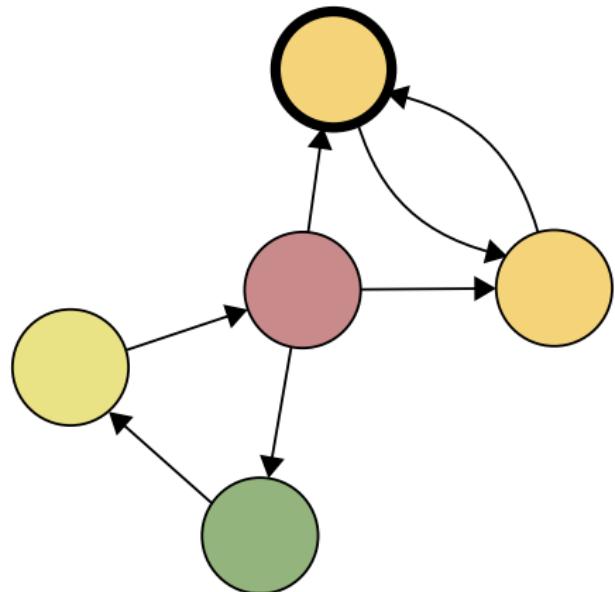
SCC-Erkennung (Cherian/Mehlhorn)



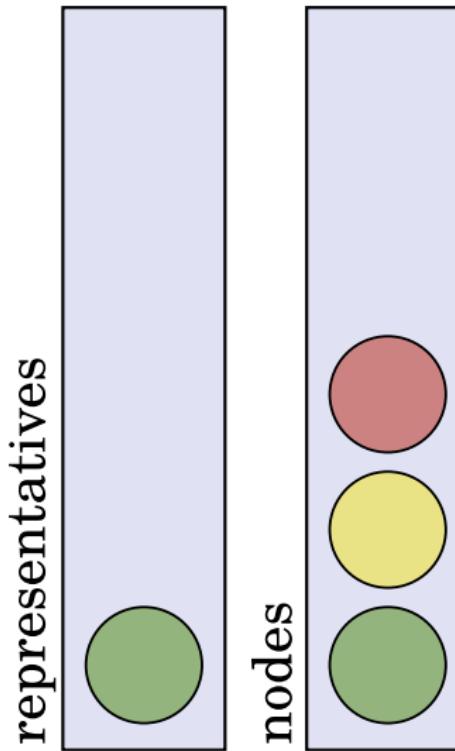
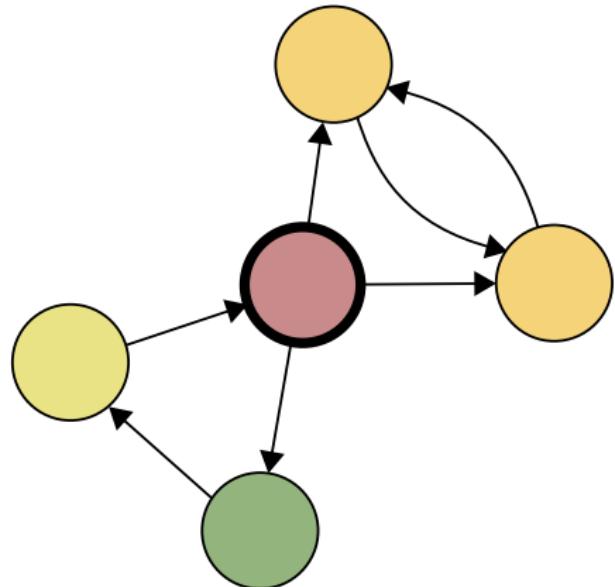
SCC-Erkennung (Cherian/Mehlhorn)



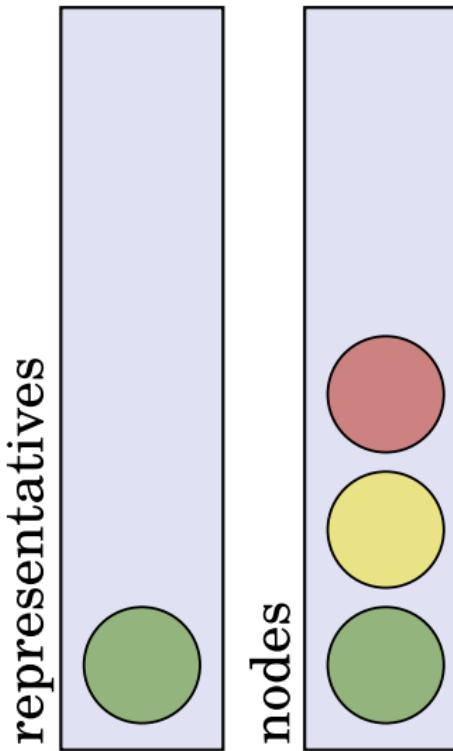
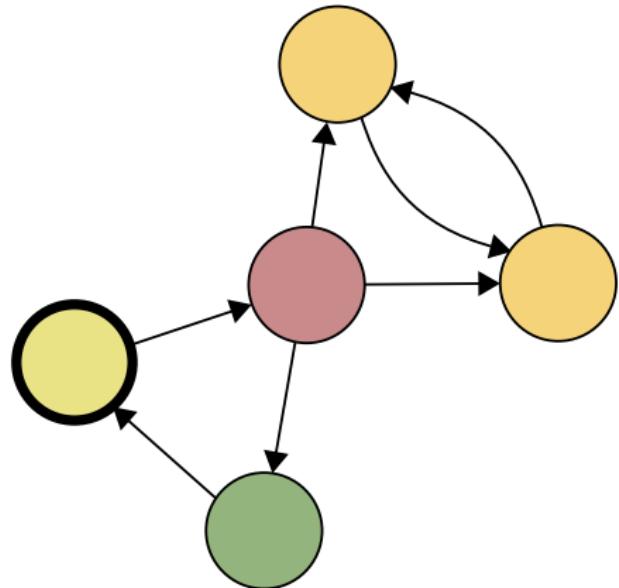
SCC-Erkennung (Cherian/Mehlhorn)



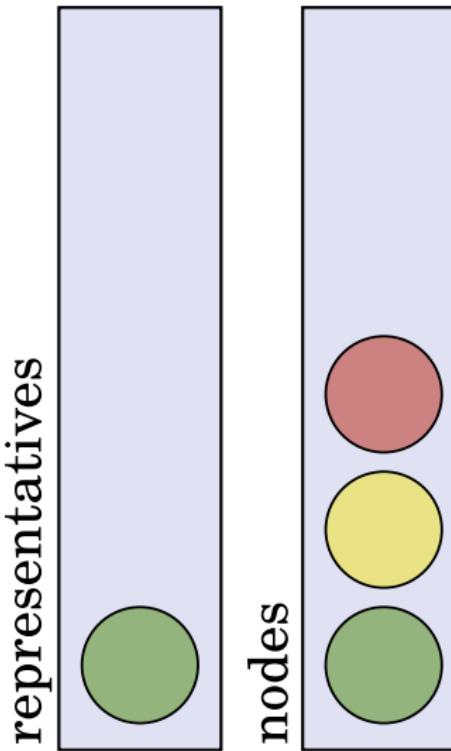
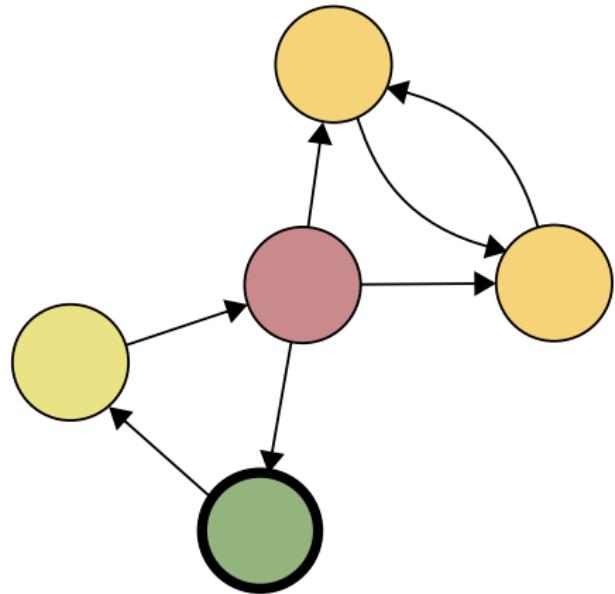
SCC-Erkennung (Cherian/Mehlhorn)



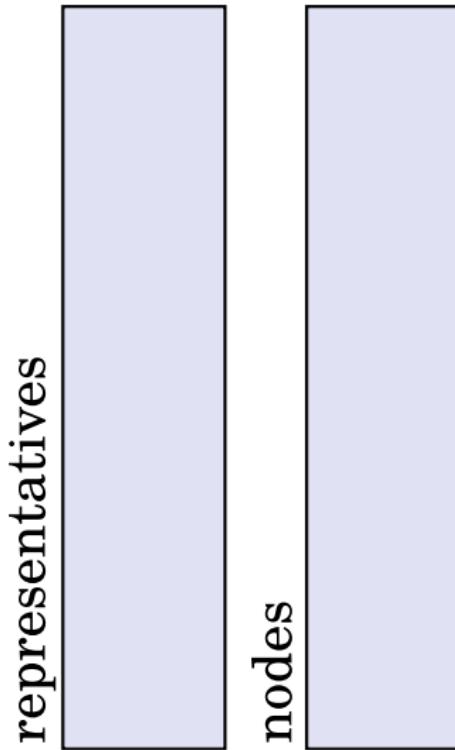
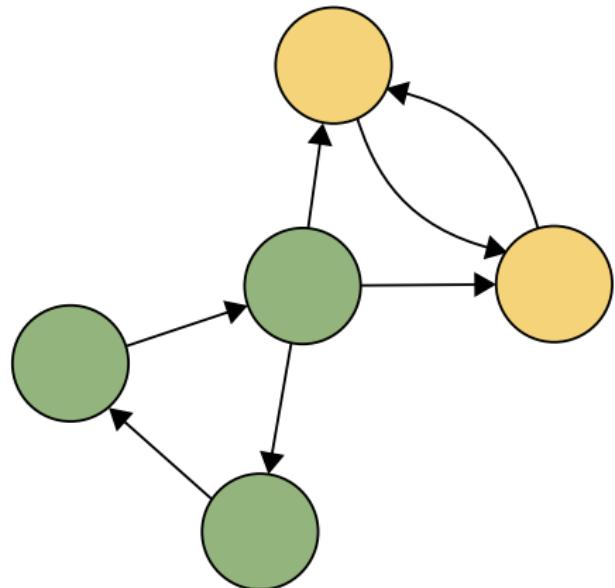
SCC-Erkennung (Cherian/Mehlhorn)



SCC-Erkennung (Cherian/Mehlhorn)



SCC-Erkennung (Cherian/Mehlhorn)

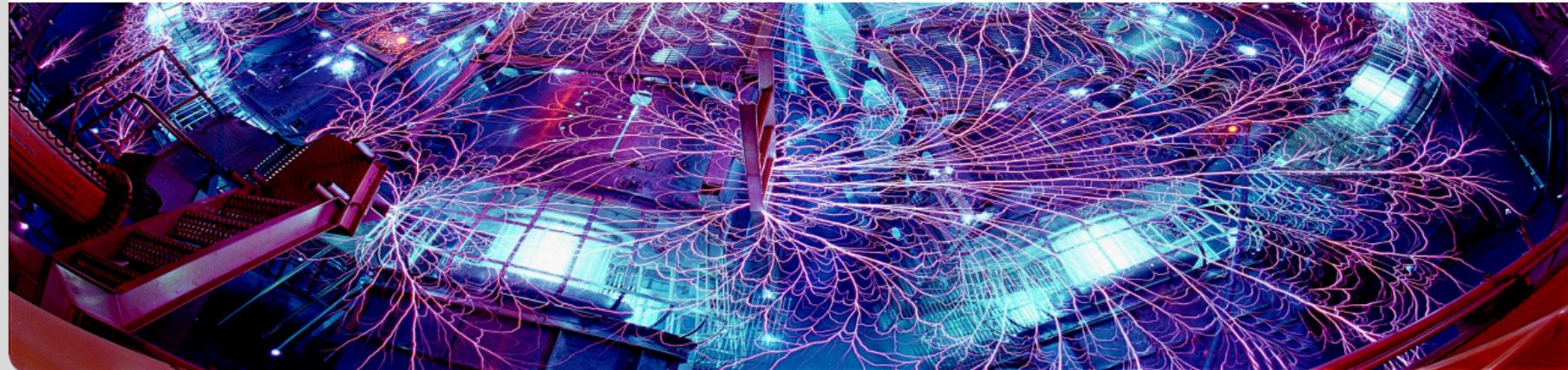


Parallele SCC-Erkennung

Miniseminar Parallele Algorithmen

Felix Karg, Konstantin Fickel | 03. Februar 2020

INSTITUT FÜR THEORETISCHE INFORMATIK



1 FWBW-Algorithmus

2 Distributed Memory

3 Pregel-artige Systeme

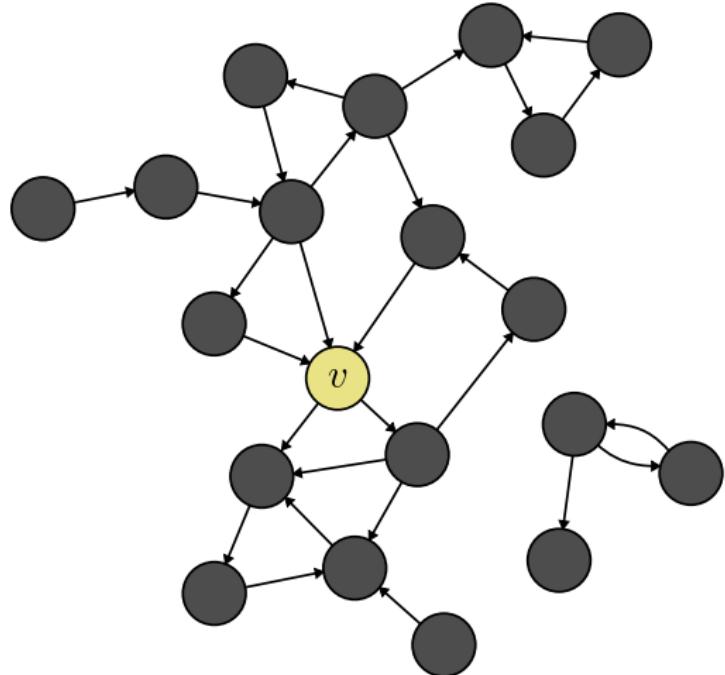
4 CUDA

5 Kleine-Welt-Graphen

Desc und Pred

Nachfolger $Desc_G(v)$

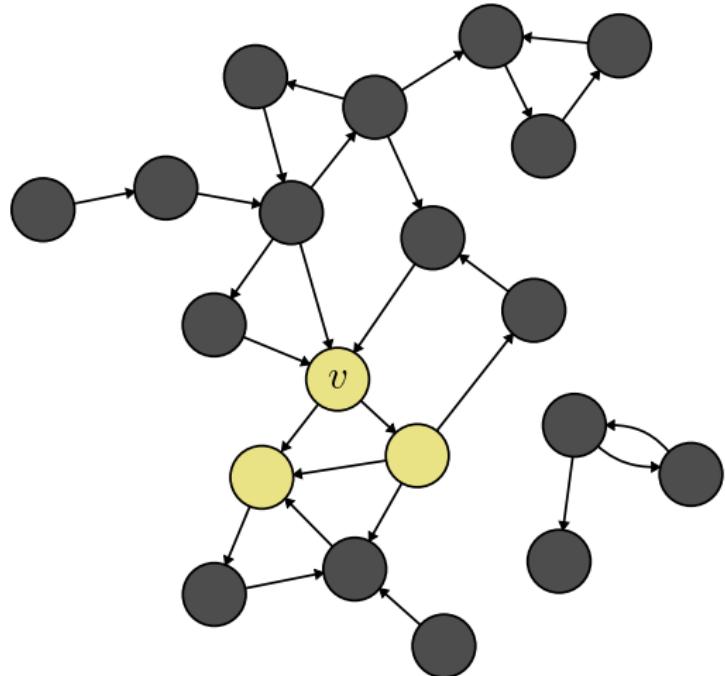
Knoten, die in G von v aus erreicht werden können



Desc und Pred

Nachfolger $Desc_G(v)$

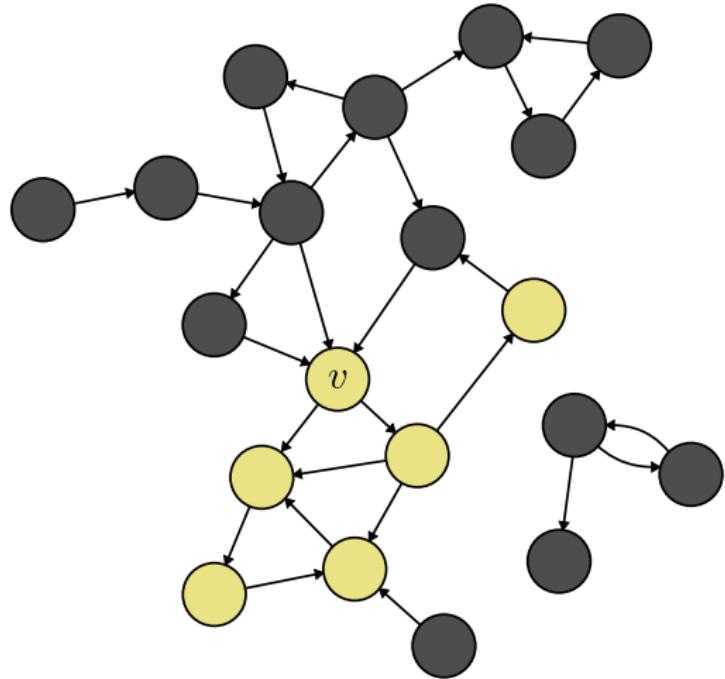
Knoten, die in G von v aus erreicht werden können



Desc und Pred

Nachfolger $Desc_G(v)$

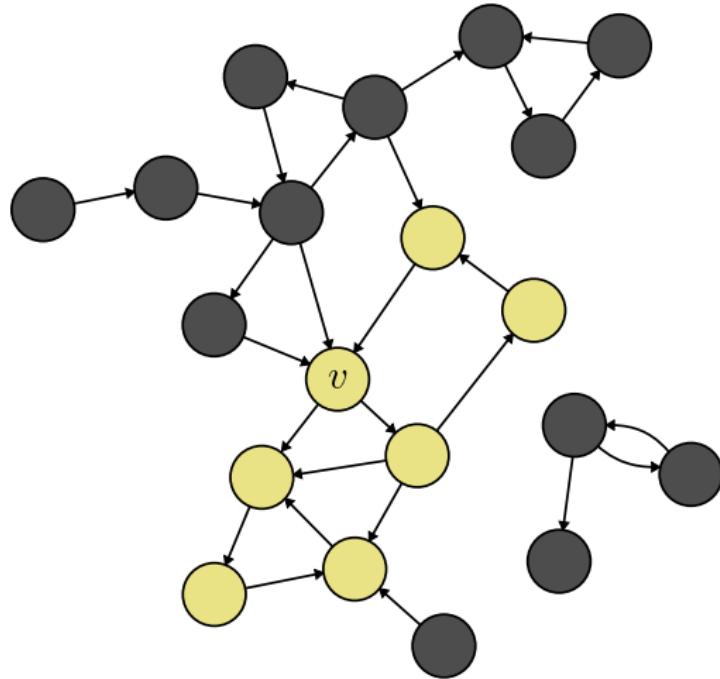
Knoten, die in G von v aus erreicht werden können



Desc und Pred

Nachfolger $Desc_G(v)$

Knoten, die in G von v aus erreicht werden können



Desc und Pred

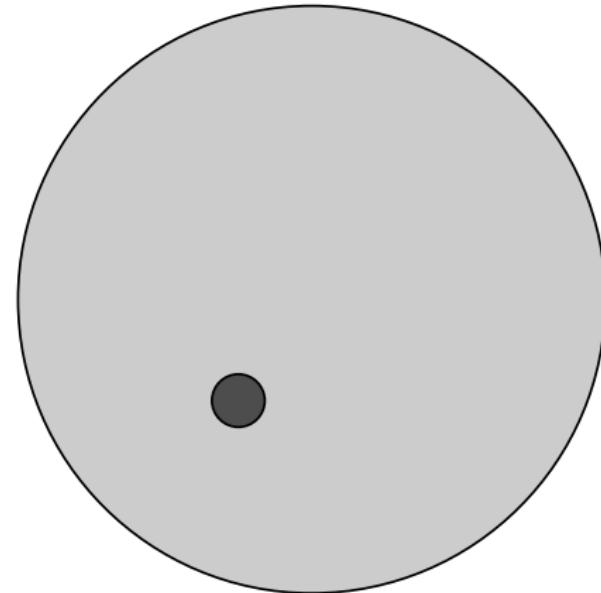
Nachfolger $Desc_G(v)$

Knoten, die in G von v aus erreicht werden können

Satz

Eine starke Zusammenhangskomponente $s \subset G$
ist entweder komplett enthalten in oder disjunkt zu
 $Desc_G(v)$

$s \subset G$ ist starke Zusammenhangskomponente \Rightarrow
 $(\forall x, y \in s : x \in Desc_G(v) \Leftrightarrow y \in Desc_G(v))$



Desc und Pred

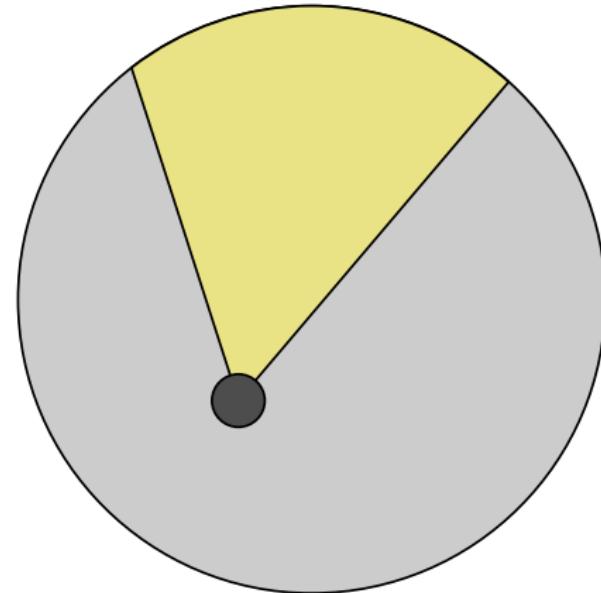
Nachfolger $Desc_G(v)$

Knoten, die in G von v aus erreicht werden können

Satz

Eine starke Zusammenhangskomponente $s \subset G$ ist entweder komplett enthalten in oder disjunkt zu $Desc_G(v)$

$s \subset G$ ist starke Zusammenhangskomponente \Rightarrow
 $(\forall x, y \in s : x \in Desc_G(v) \Leftrightarrow y \in Desc_G(v))$



Desc und Pred

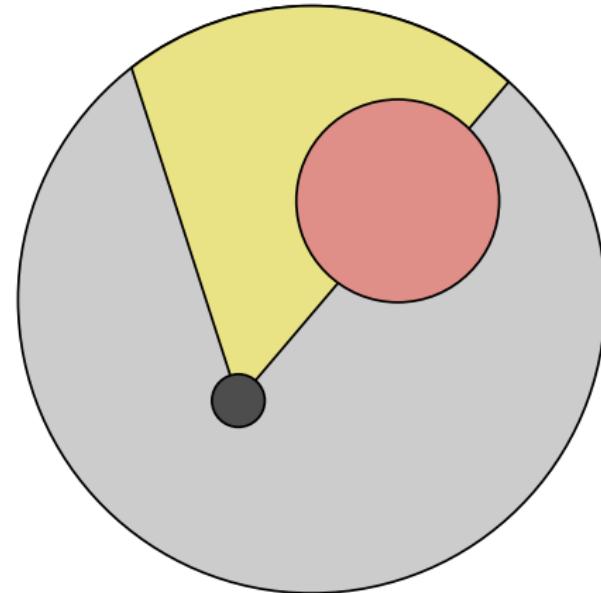
Nachfolger $Desc_G(v)$

Knoten, die in G von v aus erreicht werden können

Satz

Eine starke Zusammenhangskomponente $s \subset G$ ist entweder komplett enthalten in oder disjunkt zu $Desc_G(v)$

$s \subset G$ ist starke Zusammenhangskomponente \Rightarrow
 $(\forall x, y \in s : x \in Desc_G(v) \Leftrightarrow y \in Desc_G(v))$



Desc und Pred

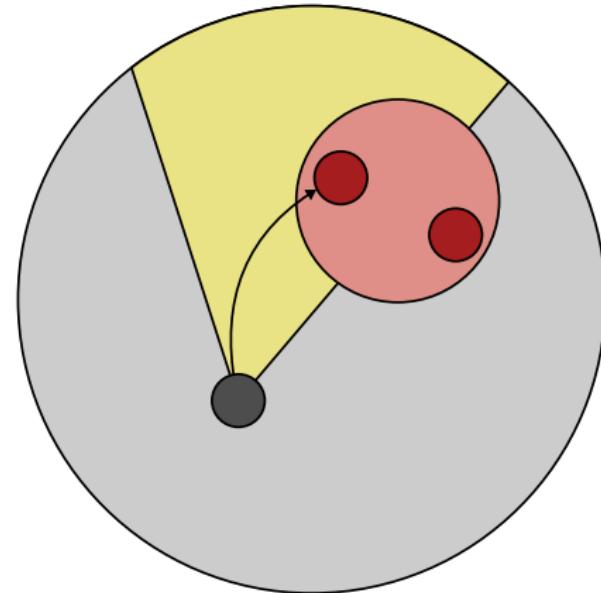
Nachfolger $Desc_G(v)$

Knoten, die in G von v aus erreicht werden können

Satz

Eine starke Zusammenhangskomponente $s \subset G$ ist entweder komplett enthalten in oder disjunkt zu $Desc_G(v)$

$s \subset G$ ist starke Zusammenhangskomponente \Rightarrow
 $(\forall x, y \in s : x \in Desc_G(v) \Leftrightarrow y \in Desc_G(v))$



Desc und Pred

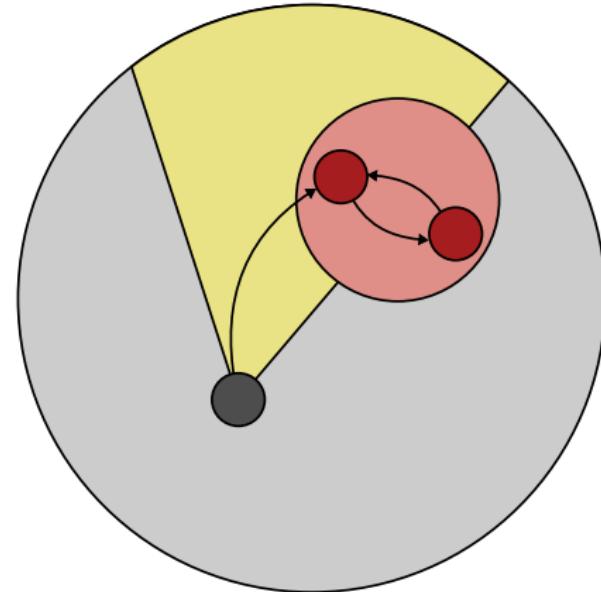
Nachfolger $Desc_G(v)$

Knoten, die in G von v aus erreicht werden können

Satz

Eine starke Zusammenhangskomponente $s \subset G$ ist entweder komplett enthalten in oder disjunkt zu $Desc_G(v)$

$s \subset G$ ist starke Zusammenhangskomponente \Rightarrow
 $(\forall x, y \in s : x \in Desc_G(v) \Leftrightarrow y \in Desc_G(v))$



Desc und Pred

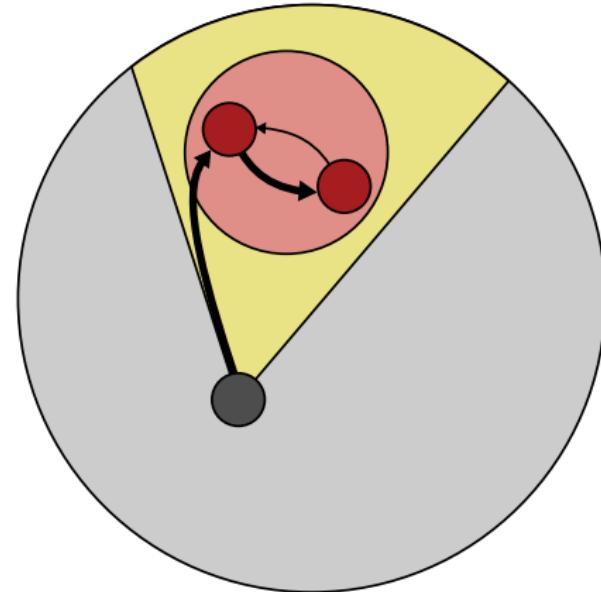
Nachfolger $Desc_G(v)$

Knoten, die in G von v aus erreicht werden können

Satz

Eine starke Zusammenhangskomponente $s \subset G$ ist entweder komplett enthalten in oder disjunkt zu $Desc_G(v)$

$s \subset G$ ist starke Zusammenhangskomponente \Rightarrow
 $(\forall x, y \in s : x \in Desc_G(v) \Leftrightarrow y \in Desc_G(v))$



Desc und Pred

Nachfolger $Desc_G(v)$

Knoten, die in G von v aus erreicht werden können

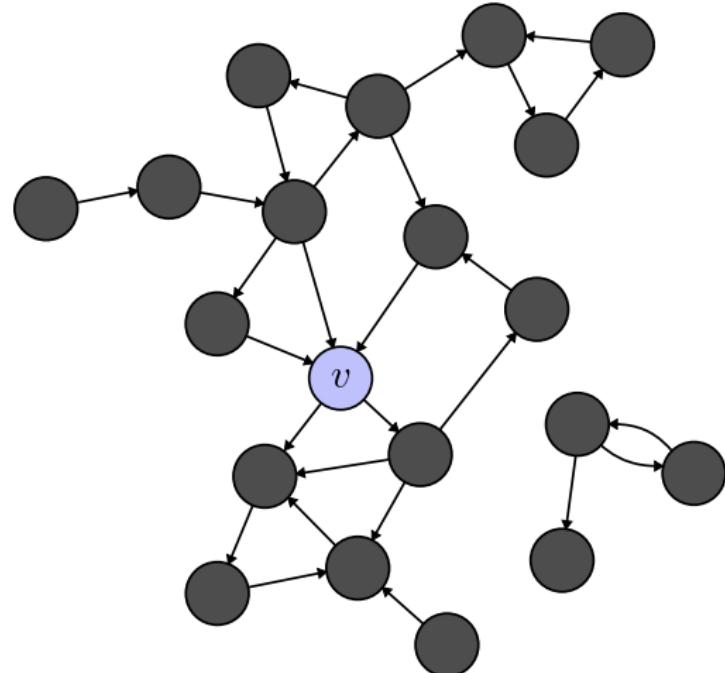
Satz

Eine starke Zusammenhangskomponente $s \subset G$ ist entweder komplett enthalten in oder disjunkt zu $Desc_G(v)$

$s \subset G$ ist starke Zusammenhangskomponente \Rightarrow
 $(\forall x, y \in s : x \in Desc_G(v) \Leftrightarrow y \in Desc_G(v))$

Vorfahren $Pred_G(v)$

Knoten, die in G^T von v aus erreicht werden können



Desc und Pred

Nachfolger $Desc_G(v)$

Knoten, die in G von v aus erreicht werden können

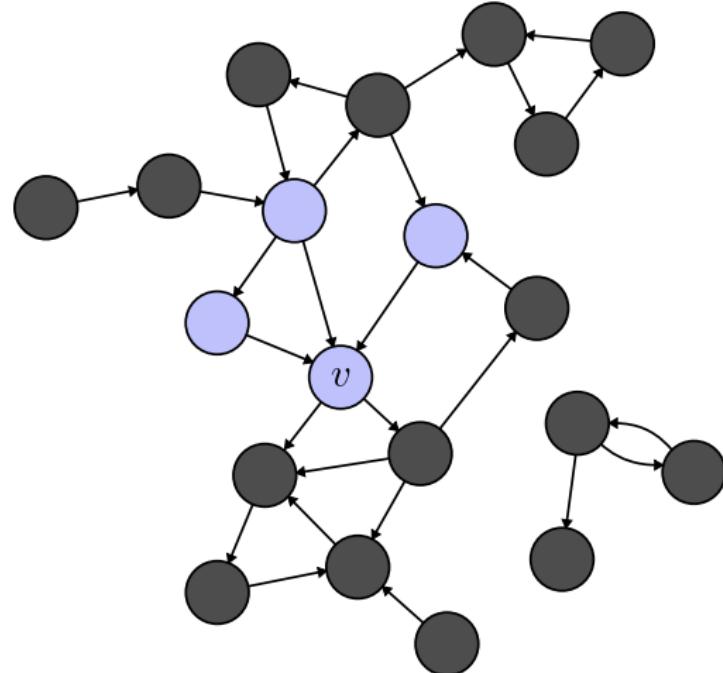
Satz

Eine starke Zusammenhangskomponente $s \subset G$ ist entweder komplett enthalten in oder disjunkt zu $Desc_G(v)$

$s \subset G$ ist starke Zusammenhangskomponente \Rightarrow
 $(\forall x, y \in s : x \in Desc_G(v) \Leftrightarrow y \in Desc_G(v))$

Vorfahren $Pred_G(v)$

Knoten, die in G^T von v aus erreicht werden können



Desc und Pred

Nachfolger $Desc_G(v)$

Knoten, die in G von v aus erreicht werden können

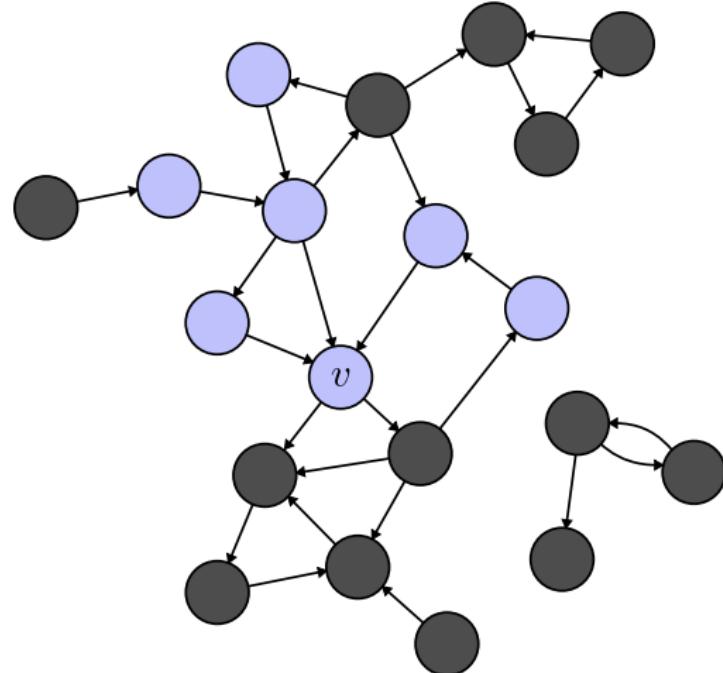
Satz

Eine starke Zusammenhangskomponente $s \subset G$ ist entweder komplett enthalten in oder disjunkt zu $Desc_G(v)$

$s \subset G$ ist starke Zusammenhangskomponente \Rightarrow
 $(\forall x, y \in s : x \in Desc_G(v) \Leftrightarrow y \in Desc_G(v))$

Vorfahren $Pred_G(v)$

Knoten, die in G^T von v aus erreicht werden können



Desc und Pred

Nachfolger $Desc_G(v)$

Knoten, die in G von v aus erreicht werden können

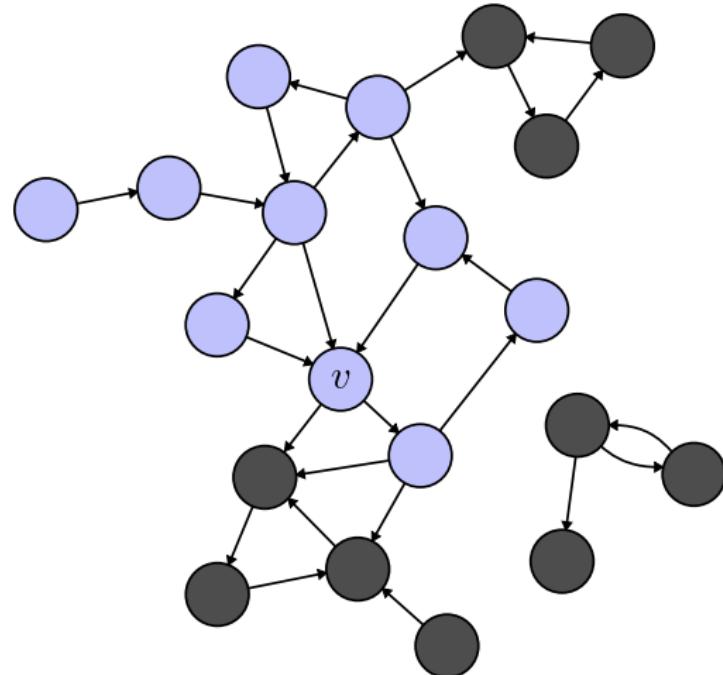
Satz

Eine starke Zusammenhangskomponente $s \subset G$ ist entweder komplett enthalten in oder disjunkt zu $Desc_G(v)$

$s \subset G$ ist starke Zusammenhangskomponente \Rightarrow
 $(\forall x, y \in s : x \in Desc_G(v) \Leftrightarrow y \in Desc_G(v))$

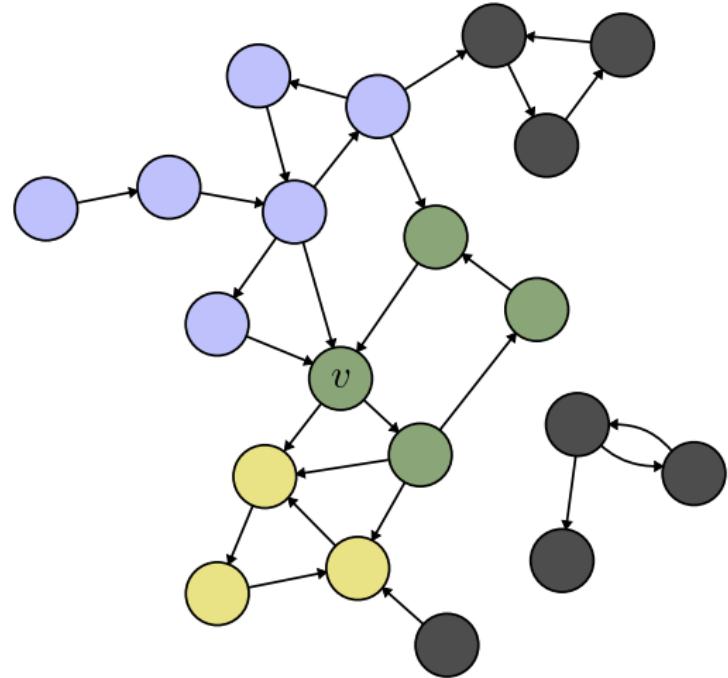
Vorfahren $Pred_G(v)$

Knoten, die in G^T von v aus erreicht werden können



Satz

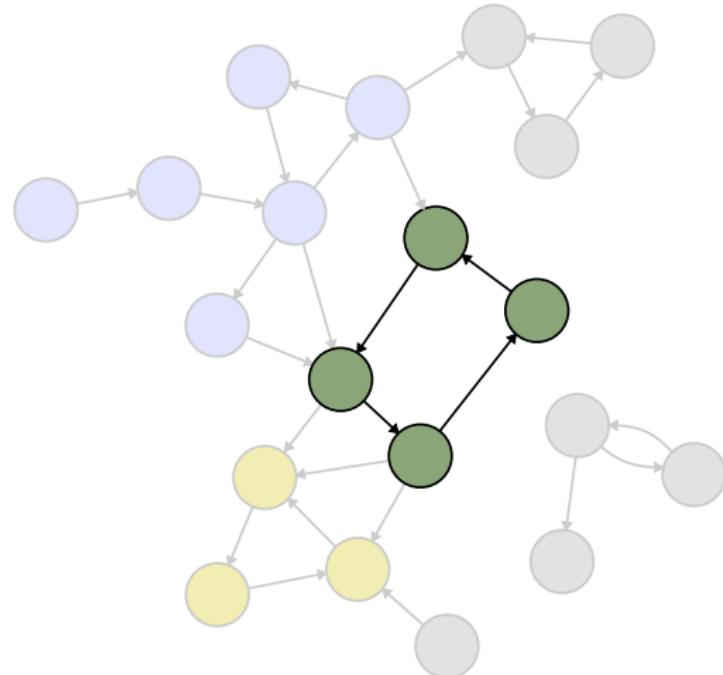
Sei s eine starke Zusammenhangskomponente von G . Dann gilt einer der folgenden Fälle:



Satz

Sei s eine starke Zusammenhangskomponente von G . Dann gilt einer der folgenden Fälle:

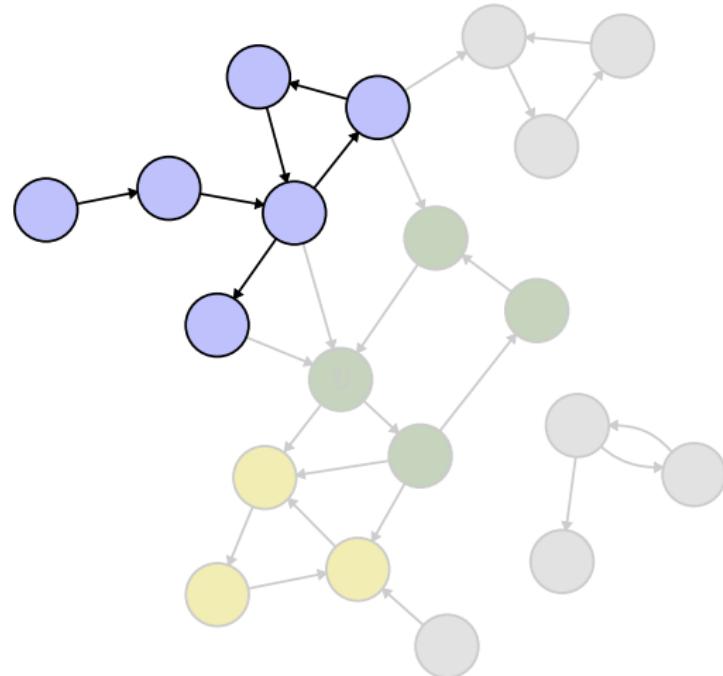
- 1 $s = \text{Pred}_G(v) \cap \text{Desc}_G(v)$



Satz

Sei s eine starke Zusammenhangskomponente von G . Dann gilt einer der folgenden Fälle:

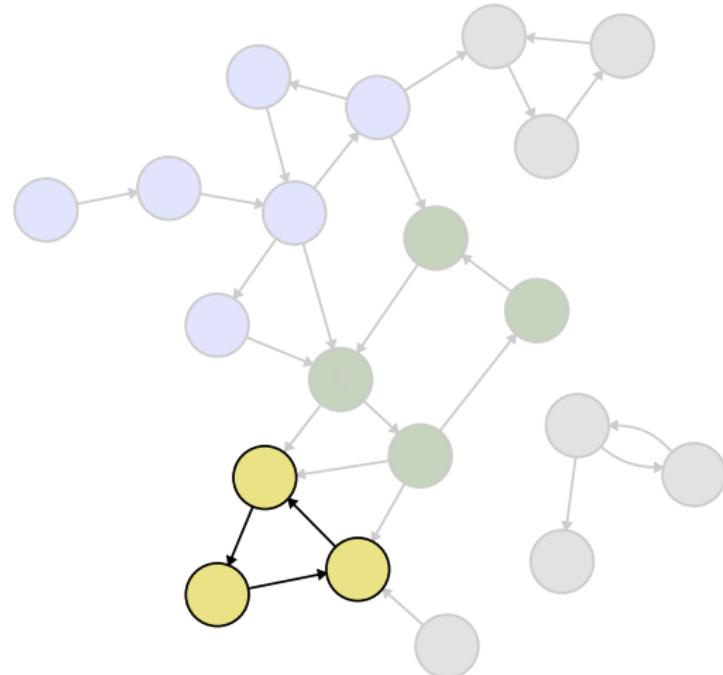
- ① $s = \text{Pred}_G(v) \cap \text{Desc}_G(v)$
- ② $s \subseteq \text{Pred}_G(v) \setminus \text{Desc}_G(v)$



Satz

Sei s eine starke Zusammenhangskomponente von G . Dann gilt einer der folgenden Fälle:

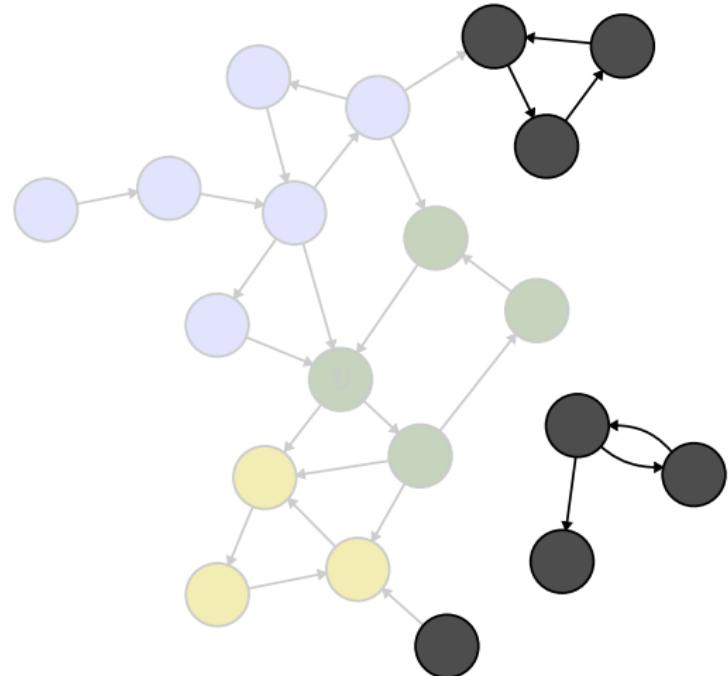
- ① $s = \text{Pred}_G(v) \cap \text{Desc}_G(v)$
- ② $s \subseteq \text{Pred}_G(v) \setminus \text{Desc}_G(v)$
- ③ $s \subseteq \text{Desc}_G(v) \setminus \text{Pred}_G(v)$



Satz

Sei s eine starke Zusammenhangskomponente von G . Dann gilt einer der folgenden Fälle:

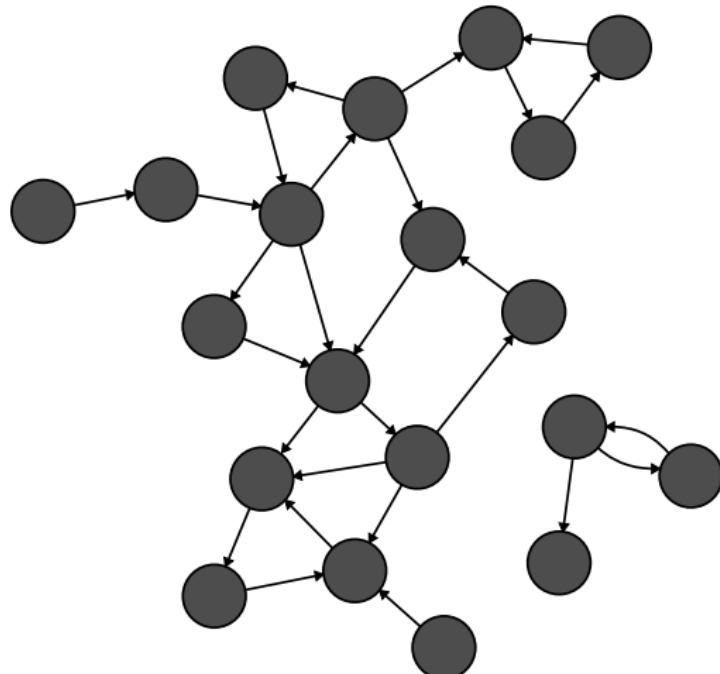
- ① $s = \text{Pred}_G(v) \cap \text{Desc}_G(v)$
- ② $s \subseteq \text{Pred}_G(v) \setminus \text{Desc}_G(v)$
- ③ $s \subseteq \text{Desc}_G(v) \setminus \text{Pred}_G(v)$
- ④ $s \subseteq \text{Rem}_G(v) = V \setminus (\text{Desc}_G(v) \cup \text{Pred}_G(v))$



FwBw-Algorithmus

Funktion FwBw-SEARCH($G = (V, E)$)

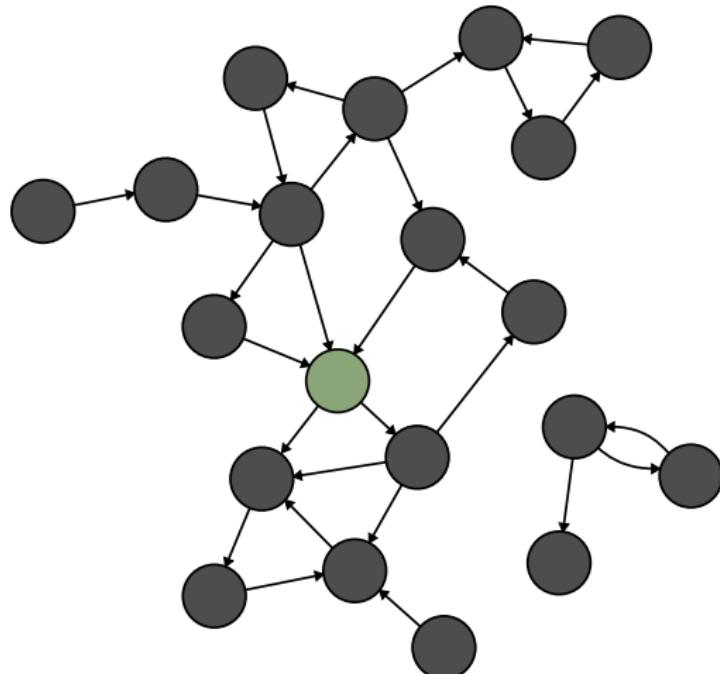
```
if  $G = \emptyset$  then
    ↳ return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```



FwBw-Algorithmus

Funktion FwBw-SEARCH($G = (V, E)$)

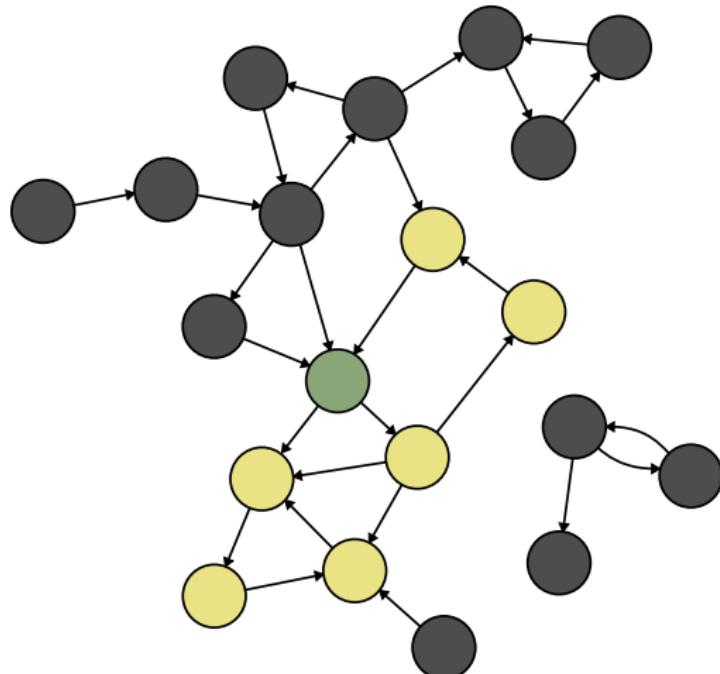
```
if  $G = \emptyset$  then
    ↘ return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```



FwBw-Algorithmus

Funktion FwBw-SEARCH($G = (V, E)$)

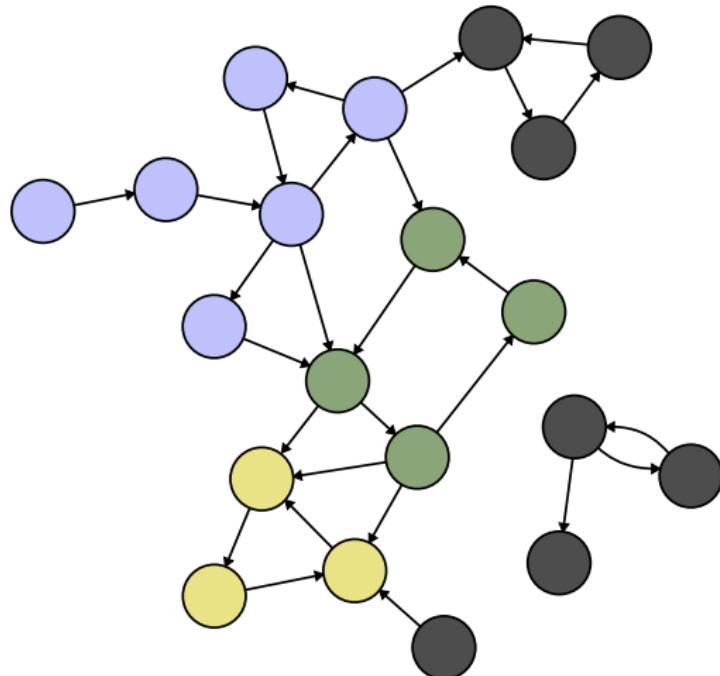
```
if  $G = \emptyset$  then
    ↳ return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```



FwBw-Algorithmus

Funktion FwBw-SEARCH($G = (V, E)$)

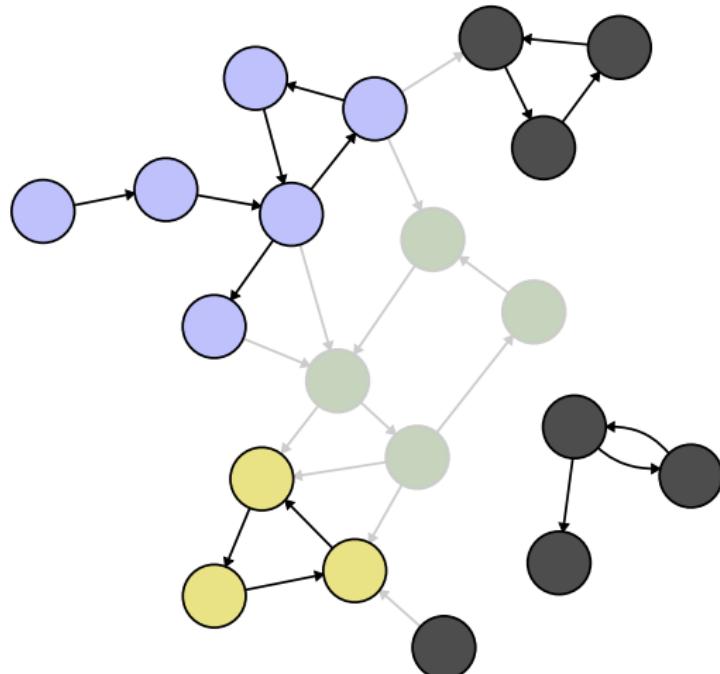
```
if  $G = \emptyset$  then
    ↳ return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```



FwBw-Algorithmus

Funktion FwBw-SEARCH($G = (V, E)$)

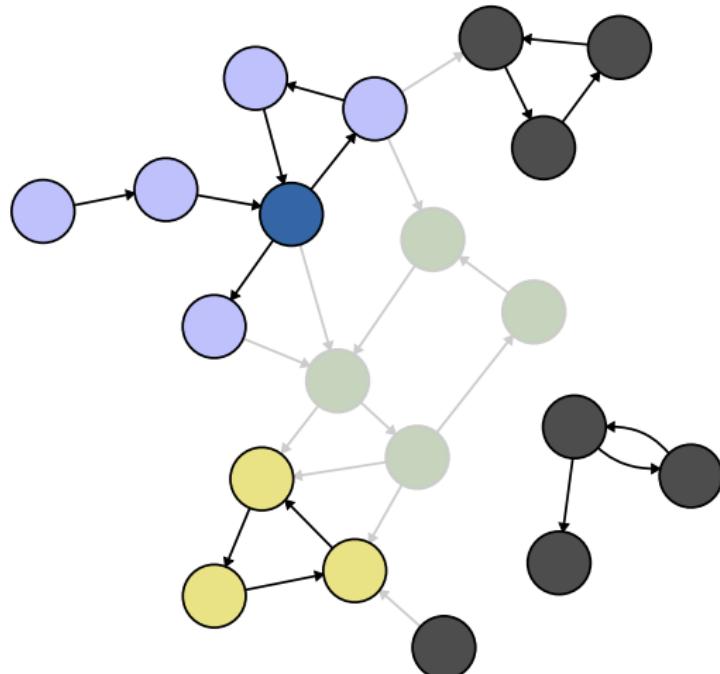
```
if  $G = \emptyset$  then
    ↳ return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```



FwBw-Algorithmus

Funktion FwBw-SEARCH($G = (V, E)$)

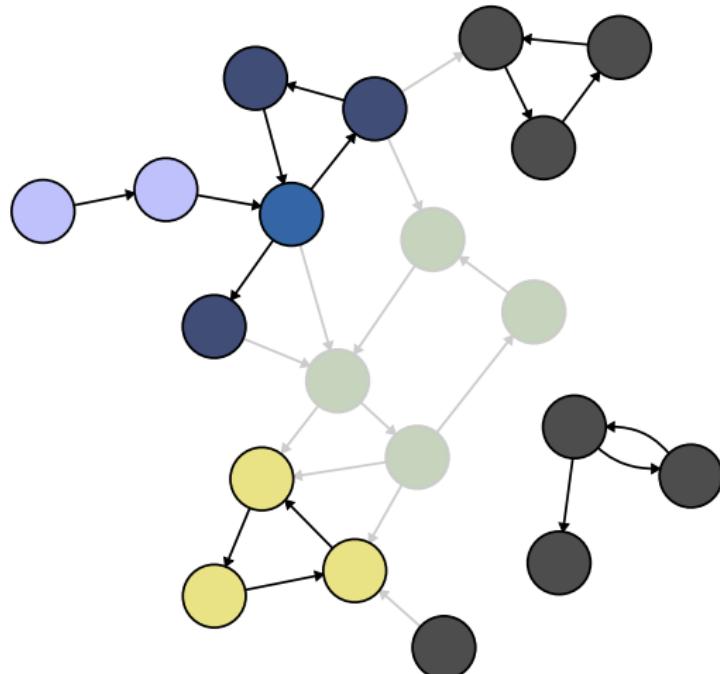
```
if  $G = \emptyset$  then
    ↳ return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```



FwBw-Algorithmus

Funktion FwBw-SEARCH($G = (V, E)$)

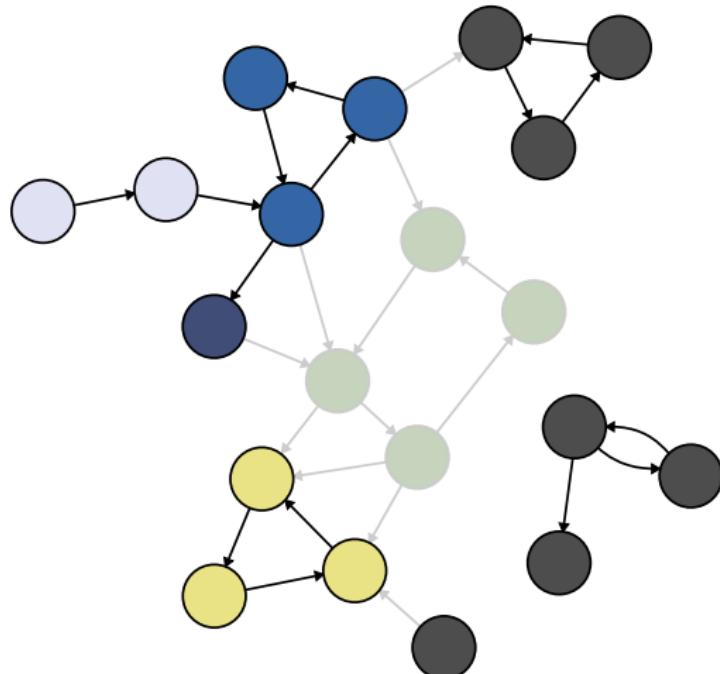
```
if  $G = \emptyset$  then
    ↳ return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```



FwBw-Algorithmus

Funktion FwBw-SEARCH($G = (V, E)$)

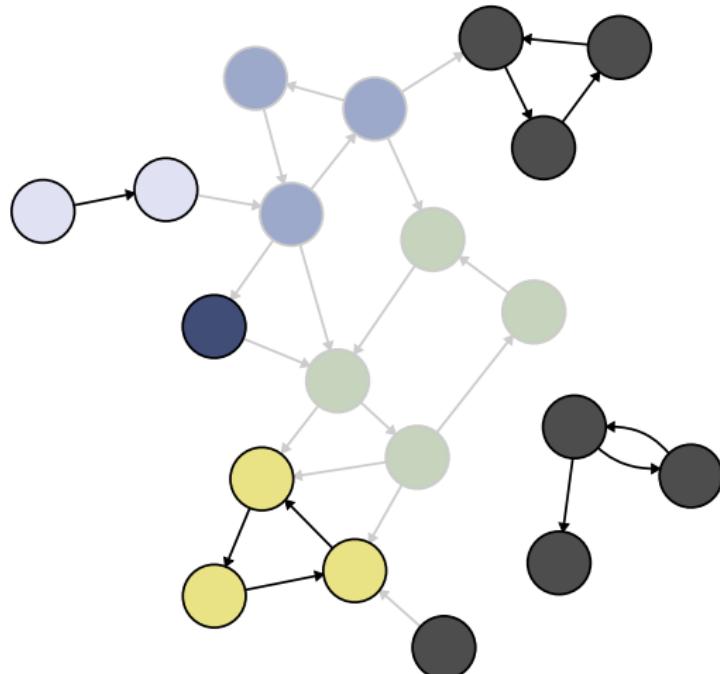
```
if  $G = \emptyset$  then
    ↳ return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```



FwBw-Algorithmus

Funktion FwBw-SEARCH($G = (V, E)$)

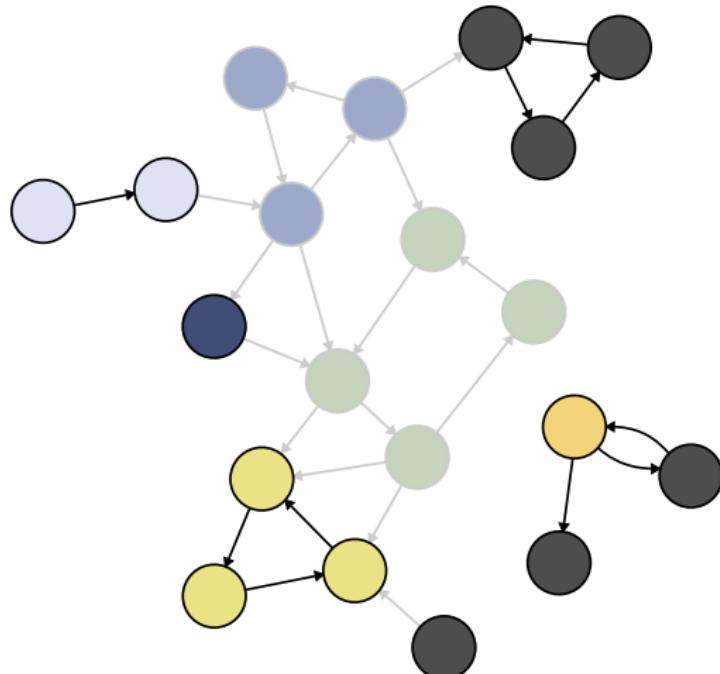
```
if  $G = \emptyset$  then
    ↳ return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```



FwBw-Algorithmus

Funktion FwBw-SEARCH($G = (V, E)$)

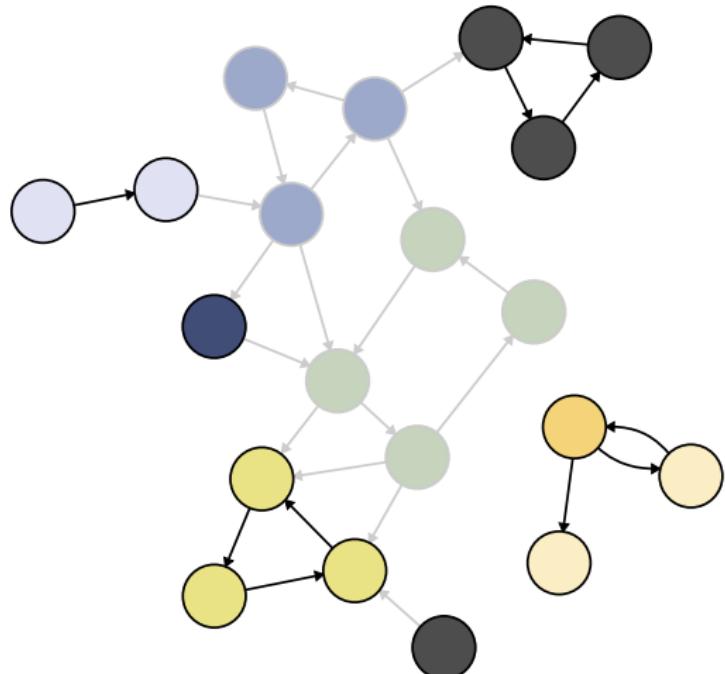
```
if  $G = \emptyset$  then
    ↳ return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```



FwBw-Algorithmus

Funktion FwBw-SEARCH($G = (V, E)$)

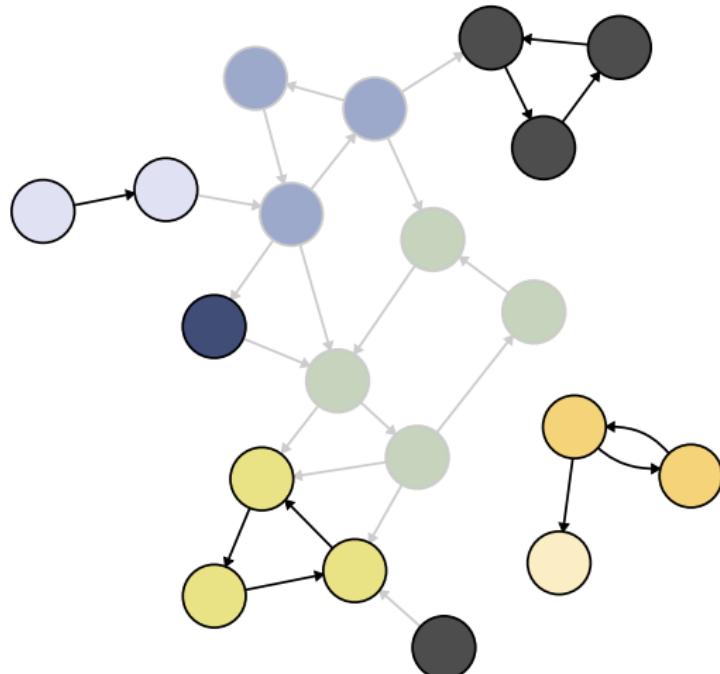
```
if  $G = \emptyset$  then
    ↳ return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```



FwBw-Algorithmus

Funktion FwBw-SEARCH($G = (V, E)$)

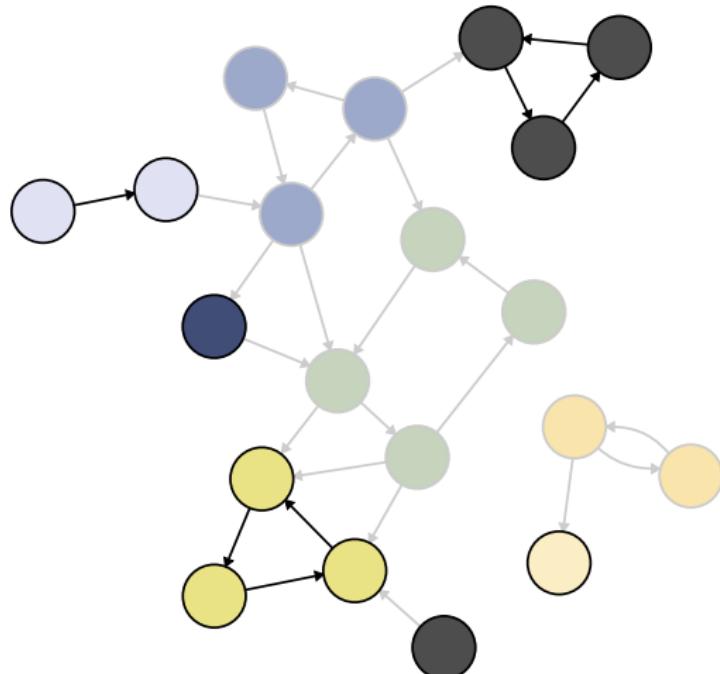
```
if  $G = \emptyset$  then
    ↳ return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```



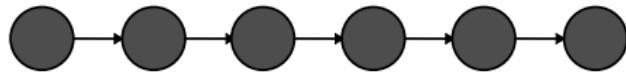
FwBw-Algorithmus

Funktion FwBw-SEARCH($G = (V, E)$)

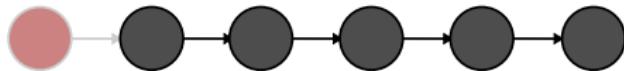
```
if  $G = \emptyset$  then
    ↳ return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```



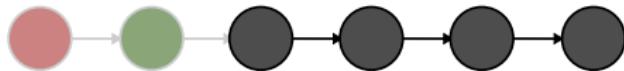
- Worst Case: $\mathcal{O}(n \cdot (n + m))$



- Worst Case: $\mathcal{O}(n \cdot (n + m))$



- Worst Case: $\mathcal{O}(n \cdot (n + m))$



- Worst Case: $\mathcal{O}(n \cdot (n + m))$



- Worst Case: $\mathcal{O}(n \cdot (n + m))$



- Worst Case: $\mathcal{O}(n \cdot (n + m))$
- Parallelisierung durch

- Worst Case: $\mathcal{O}(n \cdot (n + m))$
- Parallelisierung durch
 - Divide & Conquer

- Worst Case: $\mathcal{O}(n \cdot (n + m))$
- Parallelisierung durch
 - Divide & Conquer
 - BFS bei Nachfahren / Vorgängern

Erwartete Laufzeit

Funktion FwBw-SEARCH($G = (V, E)$)

```
if  $G = \emptyset$  then
    return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```

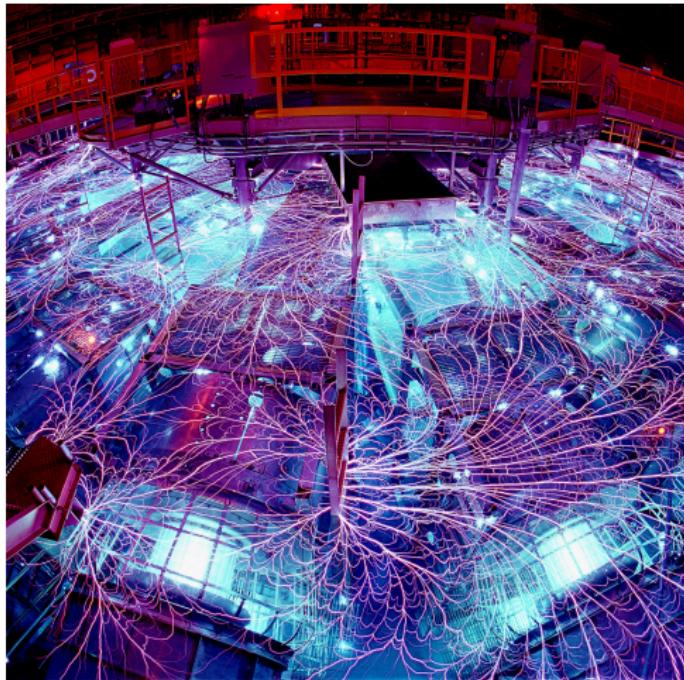
$$\begin{aligned}\mathbb{E}(T(n)) &= \frac{1}{n} \cdot \sum_{i=1}^n T(p_i) + T(d_i) \\ &\quad + T(r_i) + \Theta(n - r_i) \\ &\in \mathcal{O}(n \cdot \log(n))\end{aligned}$$

mit

$$\begin{aligned}p_i &= |Pred_G(v_i) \setminus Desc_G(v_i)| \\ d_i &= |Desc_G(v_i) \setminus Pred_G(v_i)| \\ r_i &= |Rem_G(v_i)| \\ V &= \{v_1, v_2, \dots, v_n\}\end{aligned}$$

Anwendung

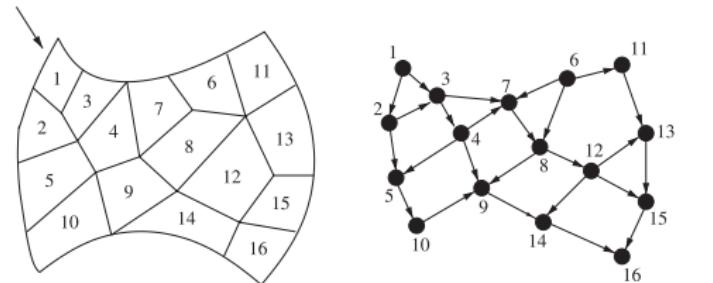
- Verwendung für
Strahlungstransportsimulation mit diskreten
Elementen



Quelle: Sandia National Laboratories

Anwendung

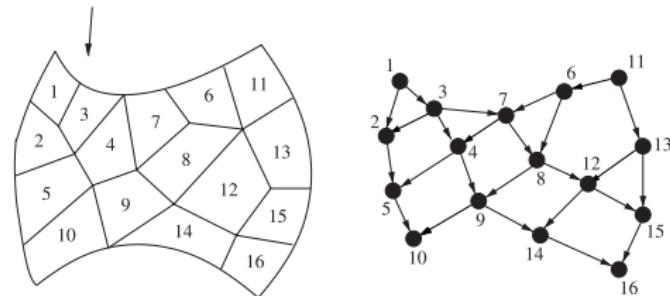
- Verwendung für
Strahlungstransportsimulation mit diskreten
Elementen



aus McLendon Iii et al. [2005]

Anwendung

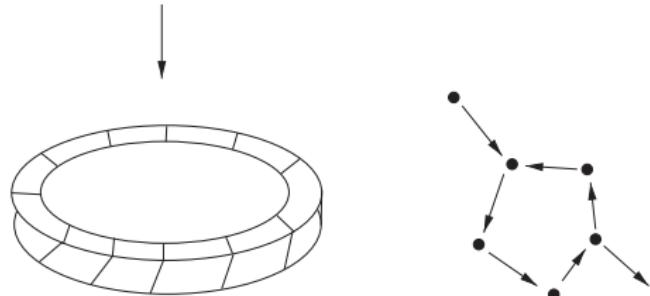
- Verwendung für Strahlungstransportsimulation mit diskreten Elementen
 - Verwendungen von mehreren verschiedenen Winkeln in der Simulation



angepasst aus McLendon III et al. [2005]

Anwendung

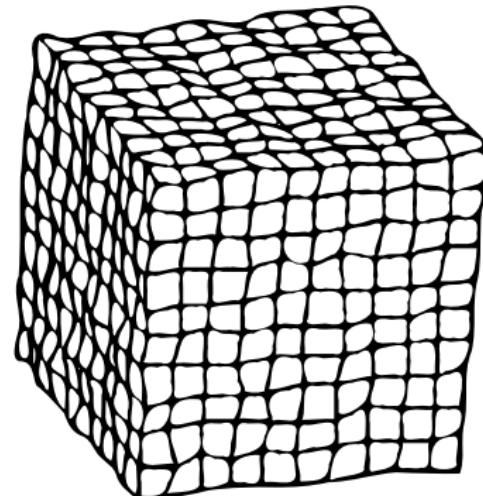
- Verwendung für
Strahlungstransportsimulation mit diskreten
Elementen
- Verwendungen von mehreren verschiedenen
Winkeln in der Simulation
- Algorithmen zur Berechnung des Flusses
scheitern, sobald Zyklen vorhanden sind ⇒
Erkennung von starken
Zusammenhangskomponenten wird benötigt



aus McLendon III et al. [2005]

Anwendung

- Verwendung für
Strahlungstransportsimulation mit diskreten
Elementen
- Verwendungen von mehreren verschiedenen
Winkeln in der Simulation
- Algorithmen zur Berechnung des Flusses
scheitern, sobald Zyklen vorhanden sind ⇒
Erkennung von starken
Zusammenhangskomponenten wird benötigt



aus McLendon III et al. [2005]

Anwendung

- Verwendung für Strahlungstransportsimulation mit diskreten Elementen
- Verwendungen von mehreren verschiedenen Winkeln in der Simulation
- Algorithmen zur Berechnung des Flusses scheitern, sobald Zyklen vorhanden sind ⇒ Erkennung von starken Zusammenhangskomponenten wird benötigt



Quelle: Sandia National Laboratories

Anwendung

- Verwendung für Strahlungstransportsimulation mit diskreten Elementen
- Verwendungen von mehreren verschiedenen Winkeln in der Simulation
- Algorithmen zur Berechnung des Flusses scheitern, sobald Zyklen vorhanden sind ⇒ Erkennung von starken Zusammenhangskomponenten wird benötigt
- Berechnung mit Distributed Memory
 - ASCI-Red (Intel TeraFLOPS Supercomputer in den Sandia National Laboratories)



Quelle: Sandia National Laboratories

Anwendung

- Verwendung für Strahlungstransportsimulation mit diskreten Elementen
- Verwendungen von mehreren verschiedenen Winkeln in der Simulation
- Algorithmen zur Berechnung des Flusses scheitern, sobald Zyklen vorhanden sind ⇒ Erkennung von starken Zusammenhangskomponenten wird benötigt
- Berechnung mit Distributed Memory
 - ASCI-Red (Intel TeraFLOPS Supercomputer in den Sandia National Laboratories)
 - 4640 PEs mit jeweils zwei Intel Pentium Pro 333 MHz-Prozessoren
 - 256 MB Speicher je PE

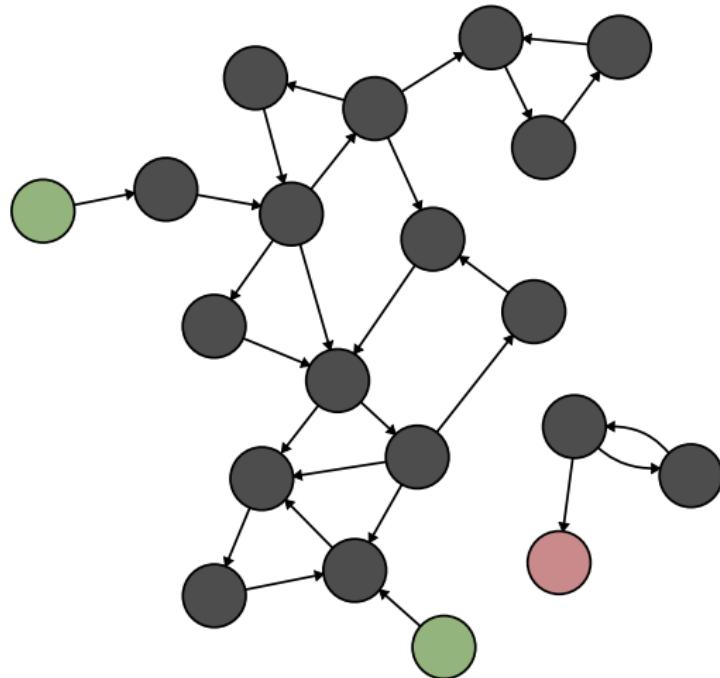


Quelle: Sandia National Laboratories

TRIM-Funktion

Funktion $\text{TRIM}(G = (V, E))$

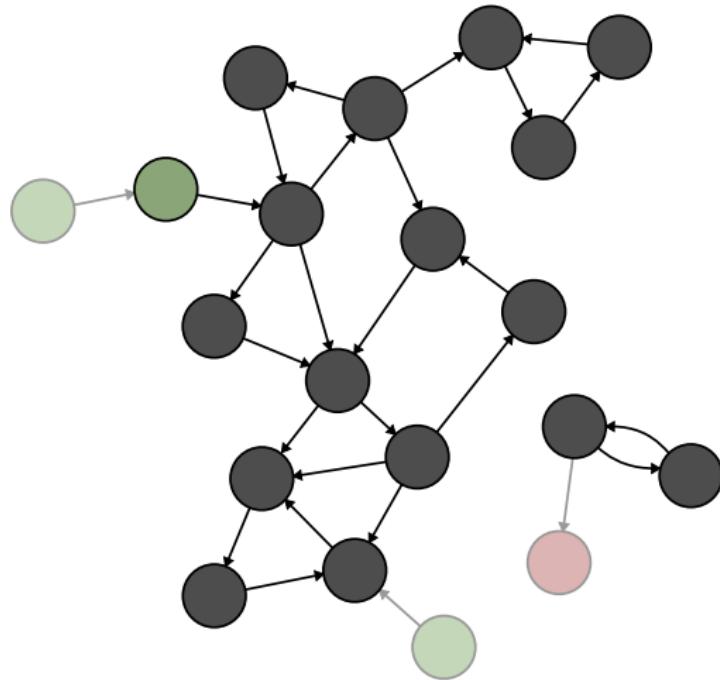
```
foreach  $v \in V$  with  $\text{indeg}_G(v) = 0$  or
 $\text{outdeg}_G(v) = 0$  do
    output  $\{v\}$ ;
     $G \leftarrow G \setminus \{v\}$ ;
```



TRIM-Funktion

Funktion $\text{TRIM}(G = (V, E))$

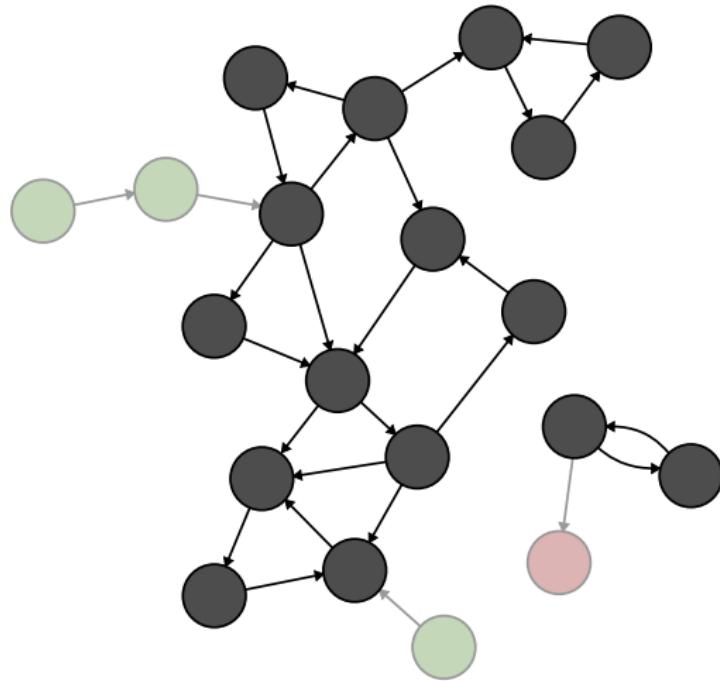
```
foreach  $v \in V$  with  $\text{indeg}_G(v) = 0$  or  
 $\text{outdeg}_G(v) = 0$  do  
    output  $\{v\}$ ;  
     $G \leftarrow G \setminus \{v\}$ ;
```



TRIM-Funktion

Funktion $\text{TRIM}(G = (V, E))$

```
foreach  $v \in V$  with  $\text{indeg}_G(v) = 0$  or  
 $\text{outdeg}_G(v) = 0$  do  
    output  $\{v\}$ ;  
     $G \leftarrow G \setminus \{v\}$ ;
```

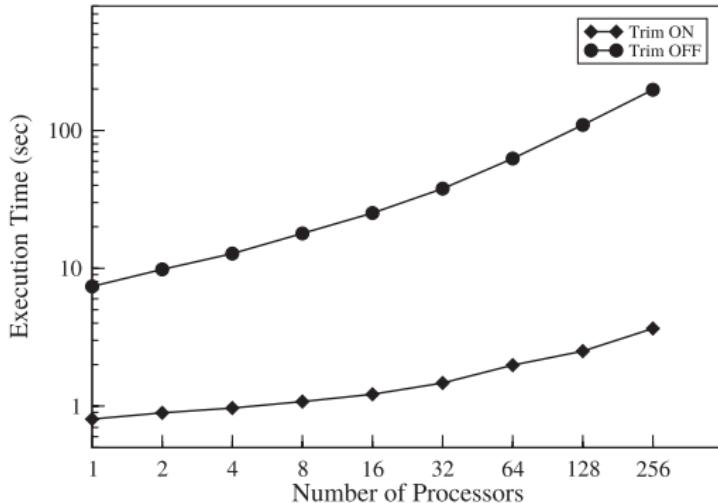


TRIM-Funktion

Funktion TRIM($G = (V, E)$)

```
foreach  $v \in V$  with  $\text{indeg}_G(v) = 0$  or
 $\text{outdeg}_G(v) = 0$  do
    output  $\{v\}$ ;
     $G \leftarrow G \setminus \{v\}$ ;
```

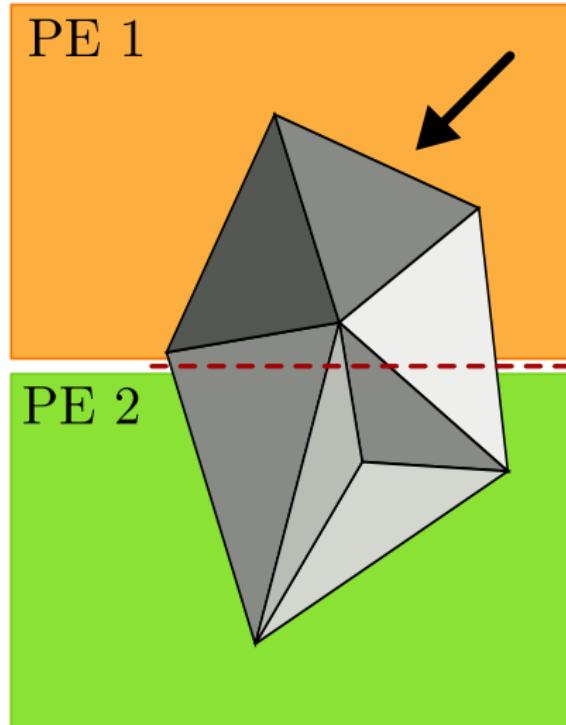
Effect of Trim on Execution Time
ASCI Red; Rectangular Mesh; 30% Deformation; 60 Angles



aus McLendon Iii et al. [2005]

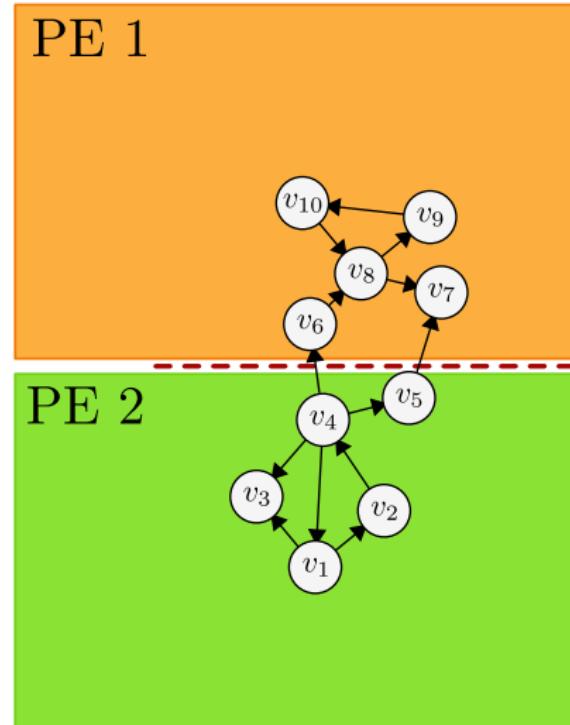
Aufteilung der Knoten

- Aufteilung des Meshs auf die einzelnen PEs



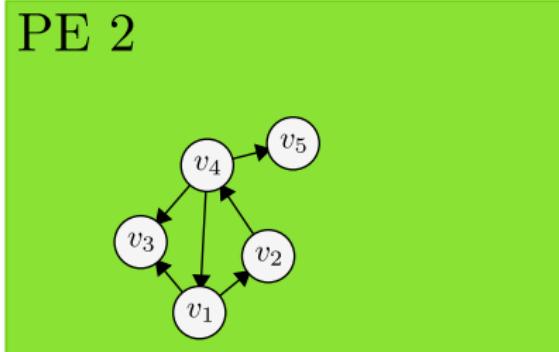
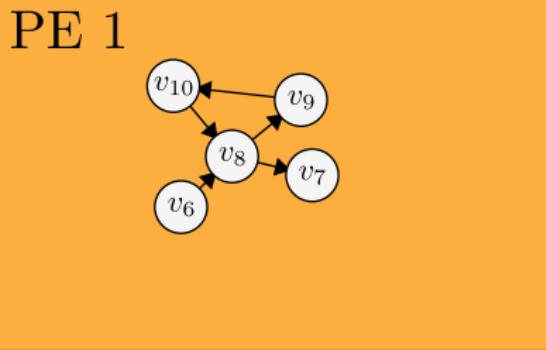
Aufteilung der Knoten

- Aufteilung des Meshs auf die einzelnen PEs
- Berechnung mehrerer *directed dependence graphs* (DDGs) für verschiedene Richtungen



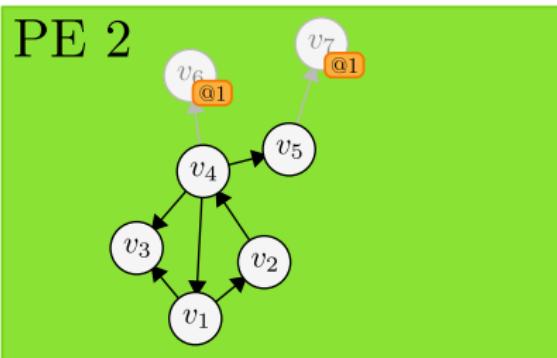
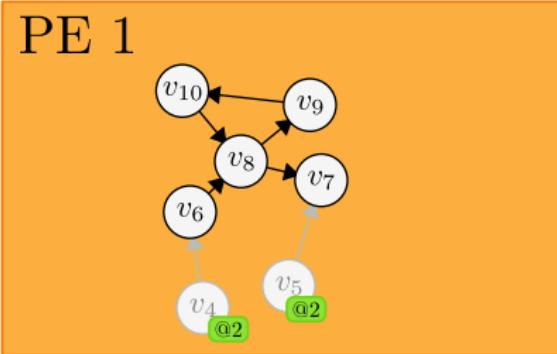
Aufteilung der Knoten

- Aufteilung des Meshs auf die einzelnen PEs
- Berechnung mehrerer *directed dependence graphs* (DDGs) für verschiedene Richtungen
- Geist-Knoten geben zuständige PE für Knoten am PE-Rand an



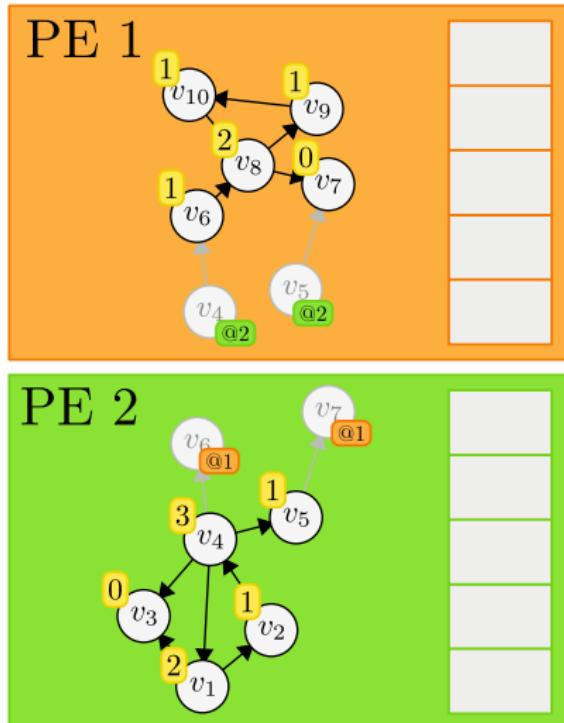
Aufteilung der Knoten

- Aufteilung des Meshs auf die einzelnen PEs
- Berechnung mehrerer *directed dependence graphs* (DDGs) für verschiedene Richtungen
- Geist-Knoten geben zuständige PE für Knoten am PE-Rand an



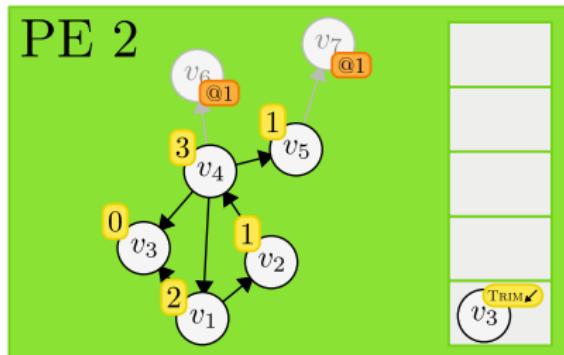
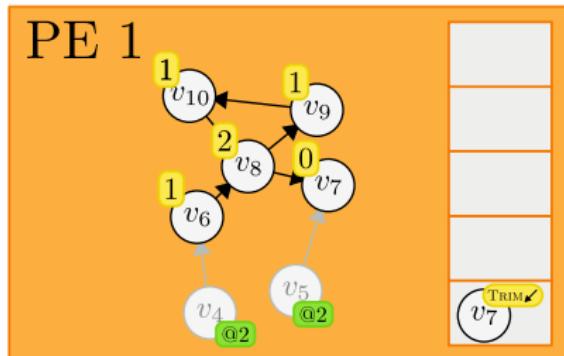
Parallelisierung

- Jede PE besitzt individuelle Task Queue



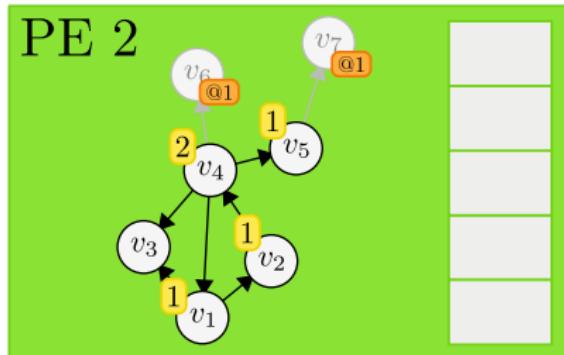
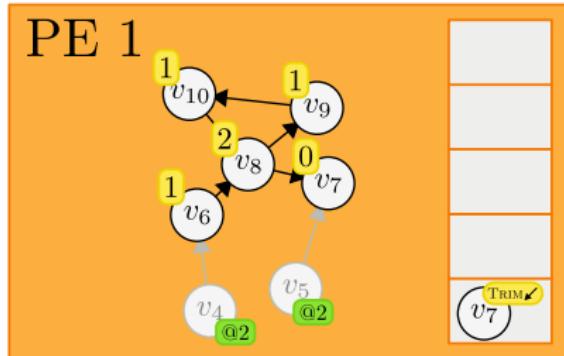
Parallelisierung

- Jede PE besitzt individuelle Task Queue
 - Vorwrtstrimmen: Jeder Knoten zhlt seine Vorgnger, sobald diese 0 werden, wird dieser in Task-Queue zum Entfernen eingefgt



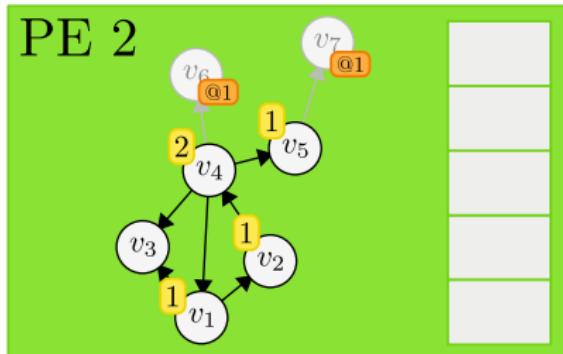
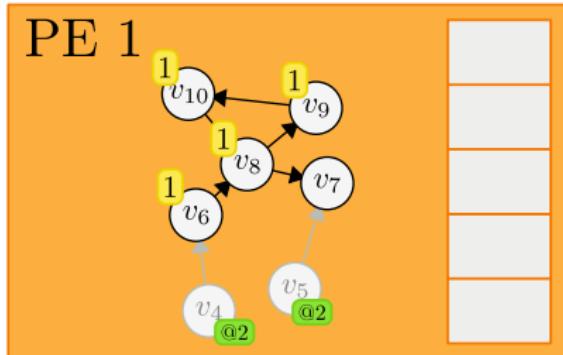
Parallelisierung

- Jede PE besitzt individuelle Task Queue
- Vorwärststrimmen: Jeder Knoten zählt seine Vorgänger, sobald diese 0 werden, wird dieser in Task-Queue zum Entfernen eingefügt



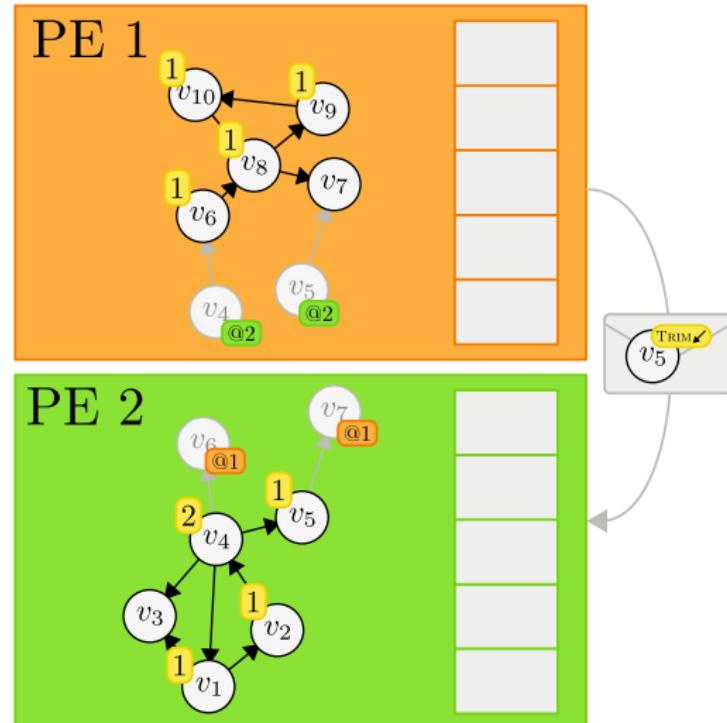
Parallelisierung

- Jede PE besitzt individuelle Task Queue
- Vorwrtstrimmnen: Jeder Knoten zahlt seine Vorgnger, sobald diese 0 werden, wird dieser in Task-Queue zum Entfernen eingefgt



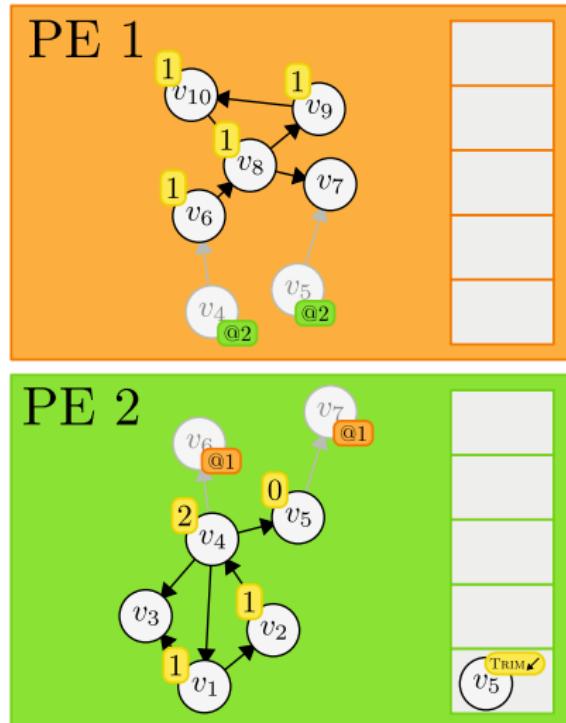
Parallelisierung

- Jede PE besitzt individuelle Task Queue
- Vorwärststrimmen: Jeder Knoten zählt seine Vorgänger, sobald diese 0 werden, wird dieser in Task-Queue zum Entfernen eingefügt
- Nachricht an entsprechenden Prozessor, falls PE-Grenze erreicht wird



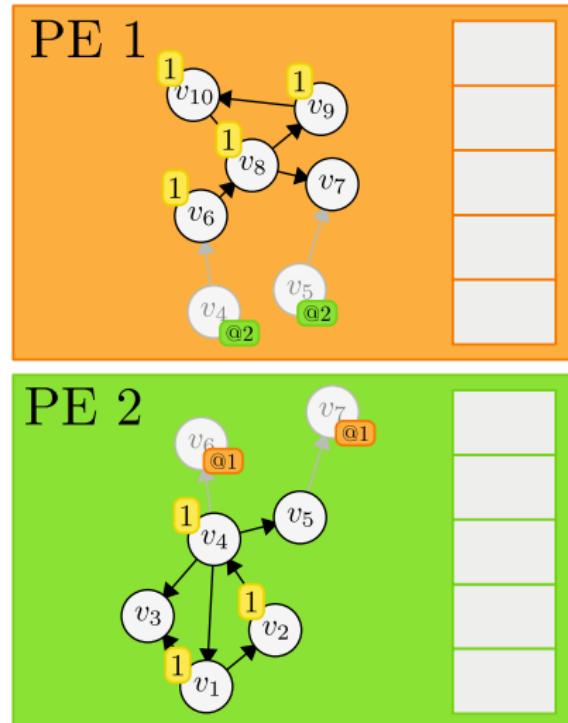
Parallelisierung

- Jede PE besitzt individuelle Task Queue
- Vorwärststrimmen: Jeder Knoten zählt seine Vorgänger, sobald diese 0 werden, wird dieser in Task-Queue zum Entfernen eingefügt
- Nachricht an entsprechenden Prozessor, falls PE-Grenze erreicht wird



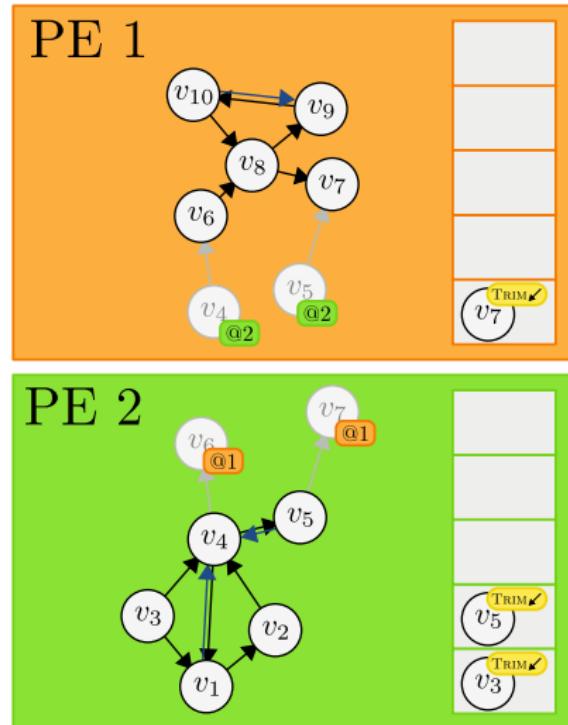
Parallelisierung

- Jede PE besitzt individuelle Task Queue
- Vorwärststrimmen: Jeder Knoten zählt seine Vorgänger, sobald diese 0 werden, wird dieser in Task-Queue zum Entfernen eingefügt
- Nachricht an entsprechenden Prozessor, falls PE-Grenze erreicht wird



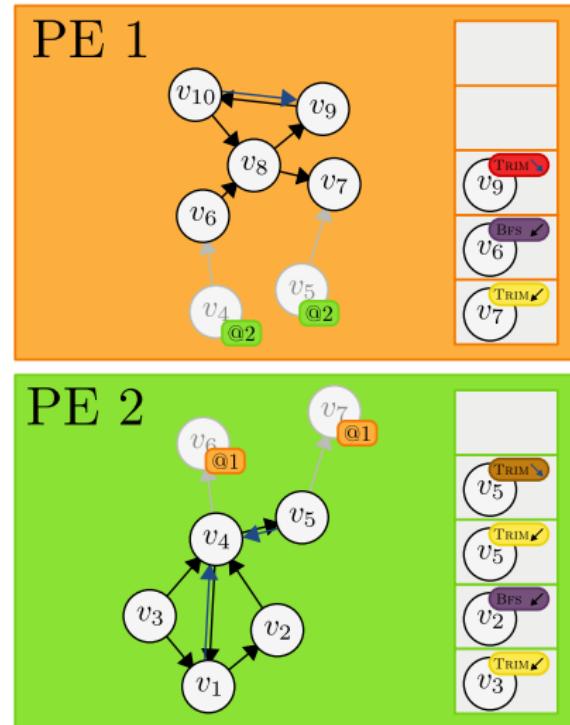
Parallelisierung

- Jede PE besitzt individuelle Task Queue
- Vorwärststrimmen: Jeder Knoten zählt seine Vorgänger, sobald diese 0 werden, wird dieser in Task-Queue zum Entfernen eingefügt
- Nachricht an entsprechenden Prozessor, falls PE-Grenze erreicht wird
- Parallelisierung durch gleichzeitige Berechnung...
 - ...mehrere Richtungs-DDGs



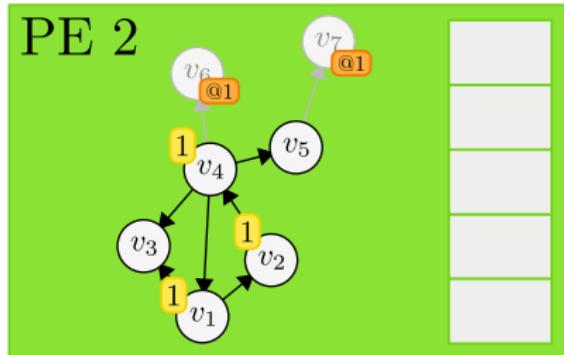
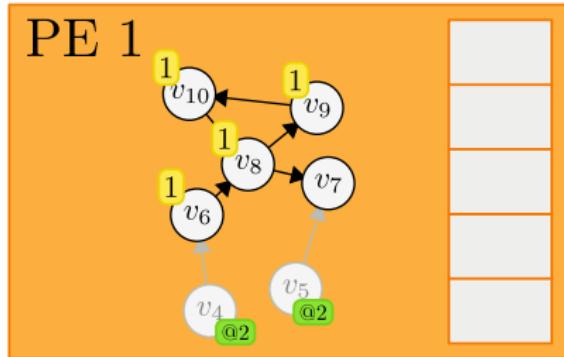
Parallelisierung

- Jede PE besitzt individuelle Task Queue
- Vorwärststrimmen: Jeder Knoten zählt seine Vorgänger, sobald diese 0 werden, wird dieser in Task-Queue zum Entfernen eingefügt
- Nachricht an entsprechenden Prozessor, falls PE-Grenze erreicht wird
- Parallelisierung durch gleichzeitige Berechnung...
 - ...mehrere Richtungs-DDGs
 - ...mehrere Operationen auf verschiedenen Teilgraphen



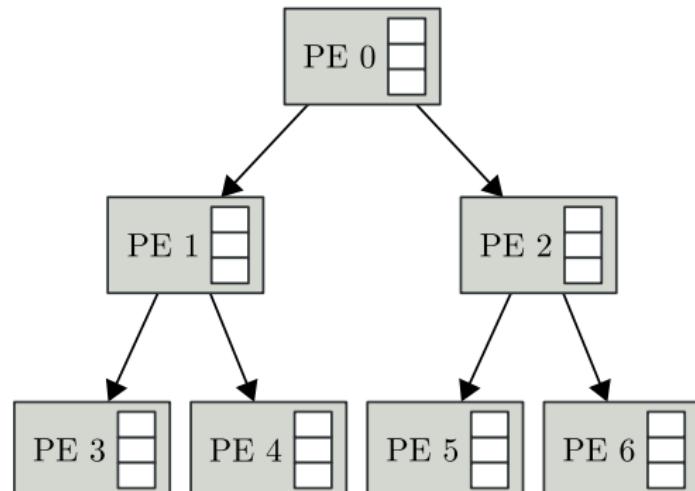
Verteilte Terminierungserkennung

- Einzelne Operationen ist abgeschlossen, sobald
 - alle lokalen Task Queues leer sind
 - keine Nachrichten mehr unterwegs sind



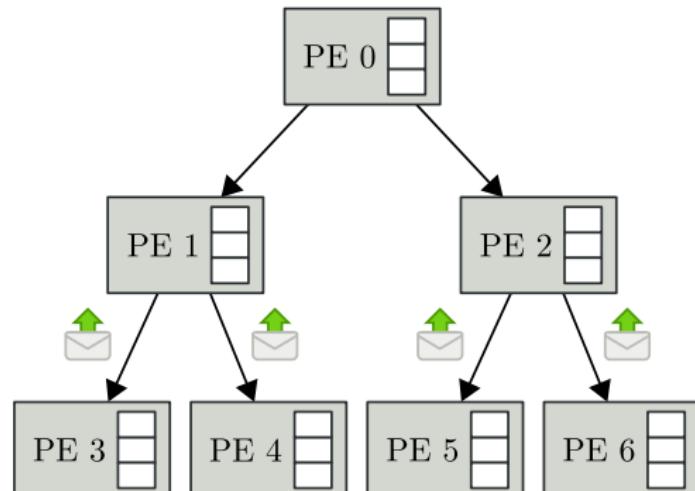
Verteilte Terminierungserkennung

- Einzelne Operationen ist abgeschlossen, sobald
 - alle lokalen Task Queues leer sind
 - keine Nachrichten mehr unterwegs sind
- Terminierungserkennung mithilfe von Binary-Tree-Reduktionen
 - Prozessoren werden in Binary-Tree-Topologie aufgefasst



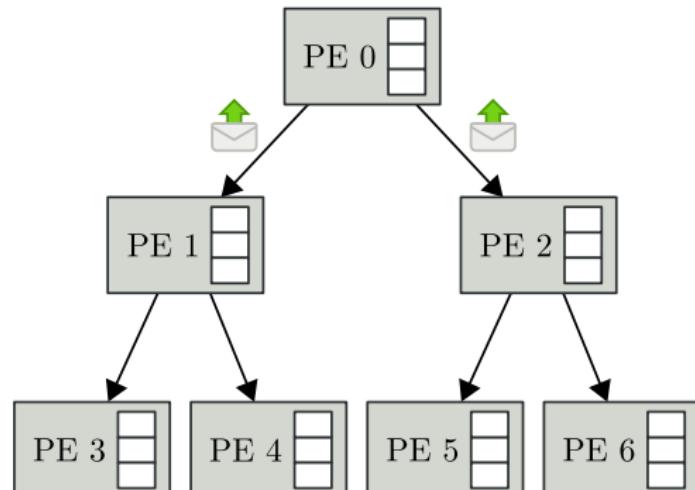
Verteilte Terminierungserkennung

- Einzelne Operationen ist abgeschlossen, sobald
 - alle lokalen Task Queues leer sind
 - keine Nachrichten mehr unterwegs sind
- Terminierungserkennung mithilfe von Binary-Tree-Reduktionen
 - Prozessoren werden in Binary-Tree-Topologie aufgefasst
 - Sammeln der getanen Arbeit und Anzahl der gesendeten / empfangenen Nachrichten



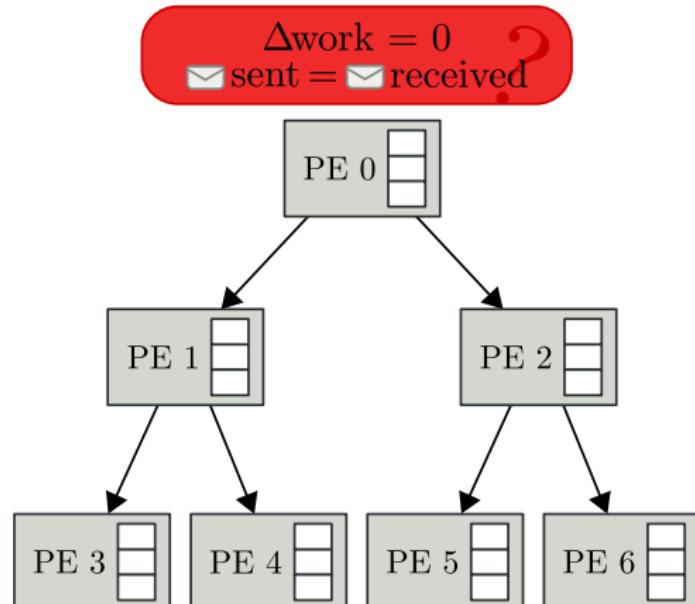
Verteilte Terminierungserkennung

- Einzelne Operationen ist abgeschlossen, sobald
 - alle lokalen Task Queues leer sind
 - keine Nachrichten mehr unterwegs sind
- Terminierungserkennung mithilfe von Binary-Tree-Reduktionen
 - Prozessoren werden in Binary-Tree-Topologie aufgefasst
 - Sammeln der getanen Arbeit und Anzahl der gesendeten / empfangenen Nachrichten



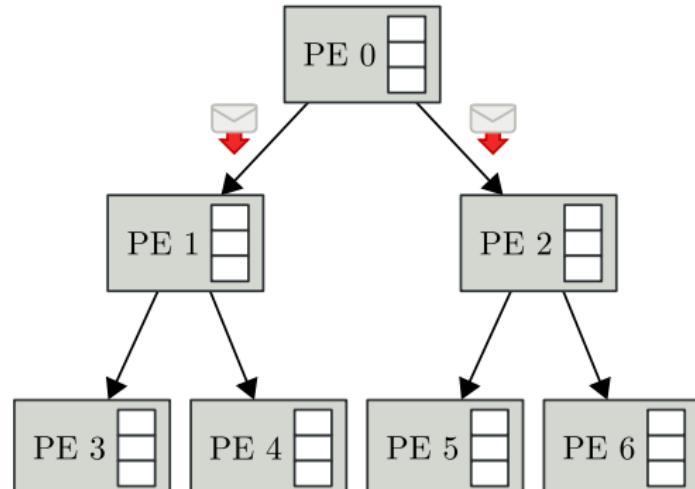
Verteilte Terminierungserkennung

- Einzelne Operationen ist abgeschlossen, sobald
 - alle lokalen Task Queues leer sind
 - keine Nachrichten mehr unterwegs sind
- Terminierungserkennung mithilfe von Binary-Tree-Reduktionen
 - Prozessoren werden in Binary-Tree-Topologie aufgefasst
 - Sammeln der getanen Arbeit und Anzahl der gesendeten / empfangenen Nachrichten
 - Broadcast, falls noch nicht fertig



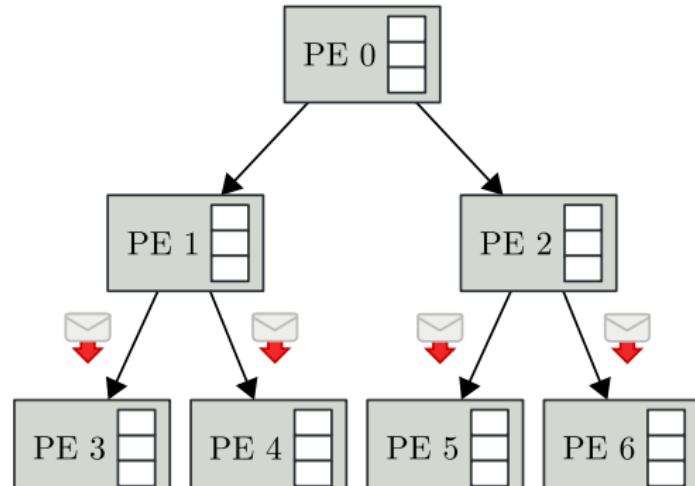
Verteilte Terminierungserkennung

- Einzelne Operationen ist abgeschlossen, sobald
 - alle lokalen Task Queues leer sind
 - keine Nachrichten mehr unterwegs sind
- Terminierungserkennung mithilfe von Binary-Tree-Reduktionen
 - Prozessoren werden in Binary-Tree-Topologie aufgefasst
 - Sammeln der getanen Arbeit und Anzahl der gesendeten / empfangenen Nachrichten
 - Broadcast, falls noch nicht fertig



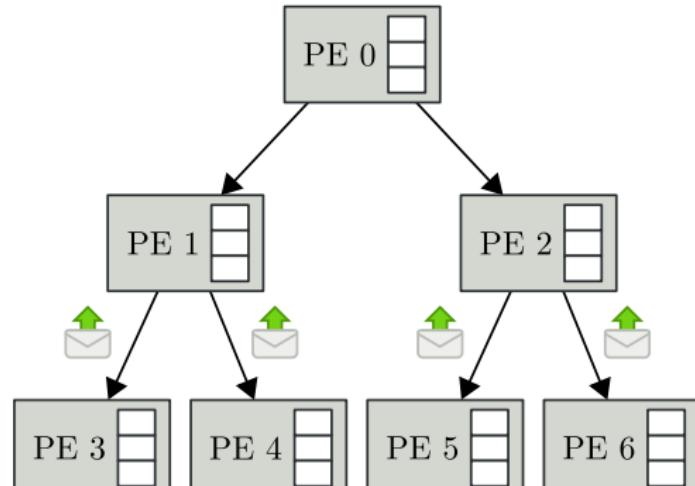
Verteilte Terminierungserkennung

- Einzelne Operationen ist abgeschlossen, sobald
 - alle lokalen Task Queues leer sind
 - keine Nachrichten mehr unterwegs sind
- Terminierungserkennung mithilfe von Binary-Tree-Reduktionen
 - Prozessoren werden in Binary-Tree-Topologie aufgefasst
 - Sammeln der getanen Arbeit und Anzahl der gesendeten / empfangenen Nachrichten
 - Broadcast, falls noch nicht fertig

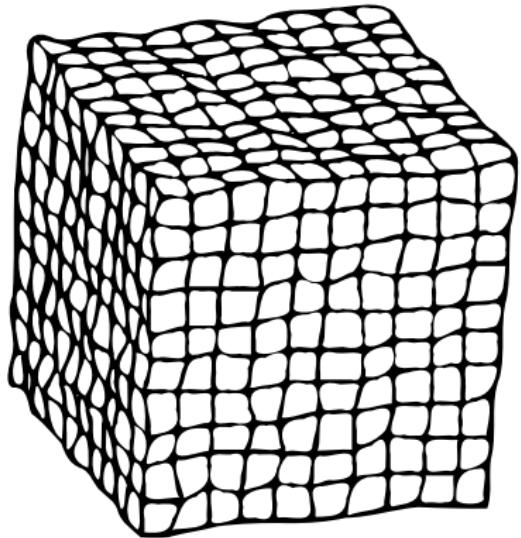


Verteilte Terminierungserkennung

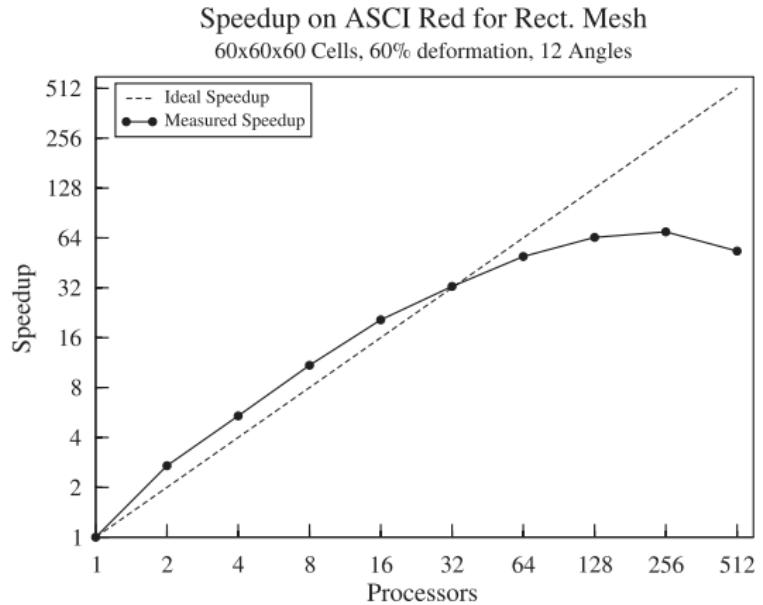
- Einzelne Operationen ist abgeschlossen, sobald
 - alle lokalen Task Queues leer sind
 - keine Nachrichten mehr unterwegs sind
- Terminierungserkennung mithilfe von Binary-Tree-Reduktionen
 - Prozessoren werden in Binary-Tree-Topologie aufgefasst
 - Sammeln der getanen Arbeit und Anzahl der gesendeten / empfangenen Nachrichten
 - Broadcast, falls noch nicht fertig



Benchmark



aus McLendon Iii et al. [2005]

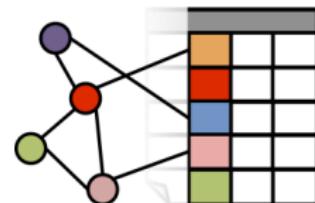


aus McLendon Iii et al. [2005]

Pregel-artige Systeme



Pregel



GraphX



FWBW-Algorithmus
○○○○○

Distributed Memory
○○○○○○○

Pregel-artige Systeme
●○○○○○○○

CUDA
○○○○○○○○○○

Kleine-Welt-Graphen
○○○○○○○○○○○

Literatur

Was ist Pregel?

Was ist Pregel?

- Große-Graphen-Massivparallelverarbeitungsframework mit Nachrichtensystem

Anwendung

- Berechnungen auf Graphen

Anwendung

- Berechnungen auf Graphen
- Die sehr viel größer sind als auf eine Maschine passt

- Berechnungen auf Graphen
- Die sehr viel größer sind als auf eine Maschine passt
- Wenn MapReduce nicht mehr reicht

- Berechnungen auf Graphen
- Die sehr viel größer sind als auf eine Maschine passt
- Wenn MapReduce nicht mehr reicht
- (MapReduce verlangt unabhängige Daten ... Pregel versickt Nachrichten)

Ablauf

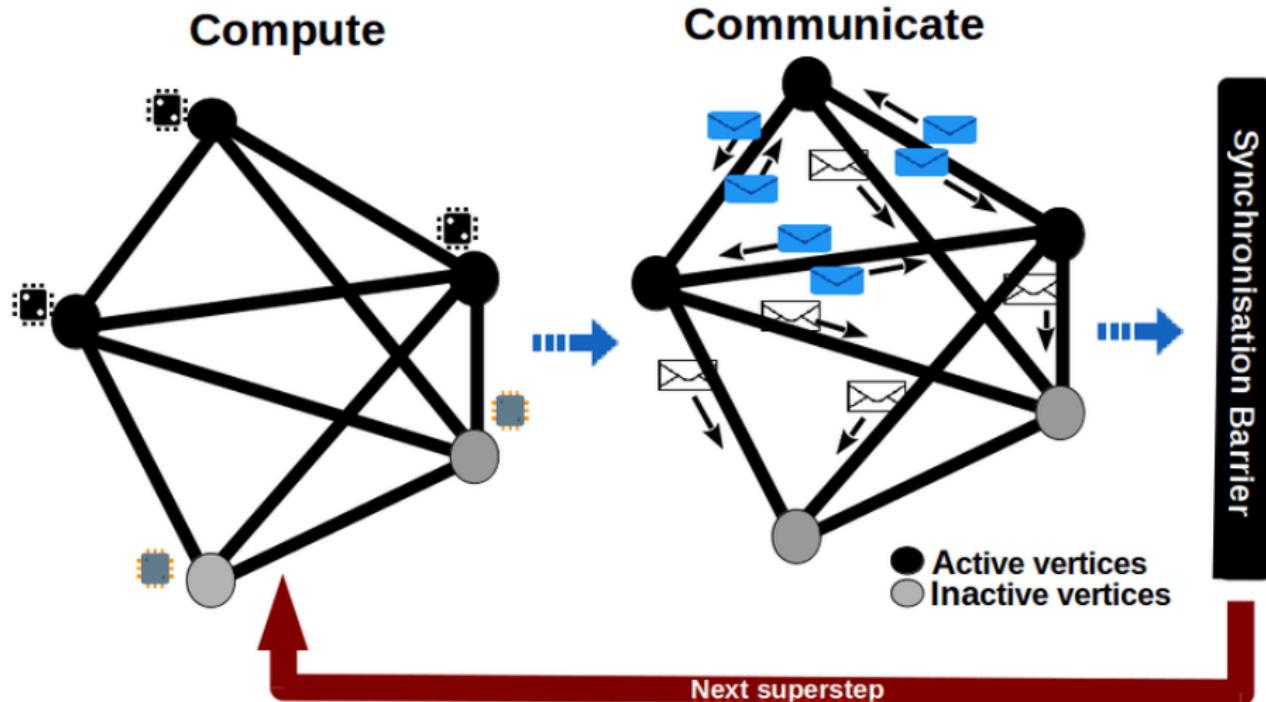


Bild aus Hassan and Bamha [2017], unverändert.

Ablauf

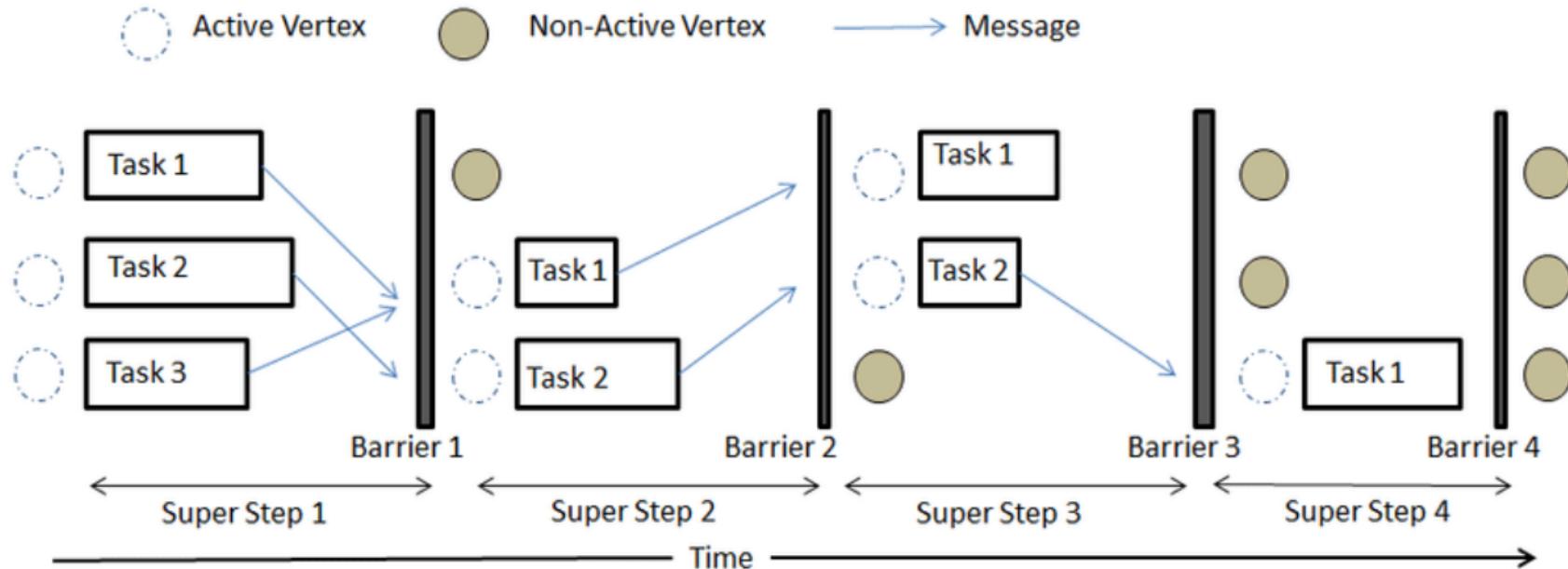
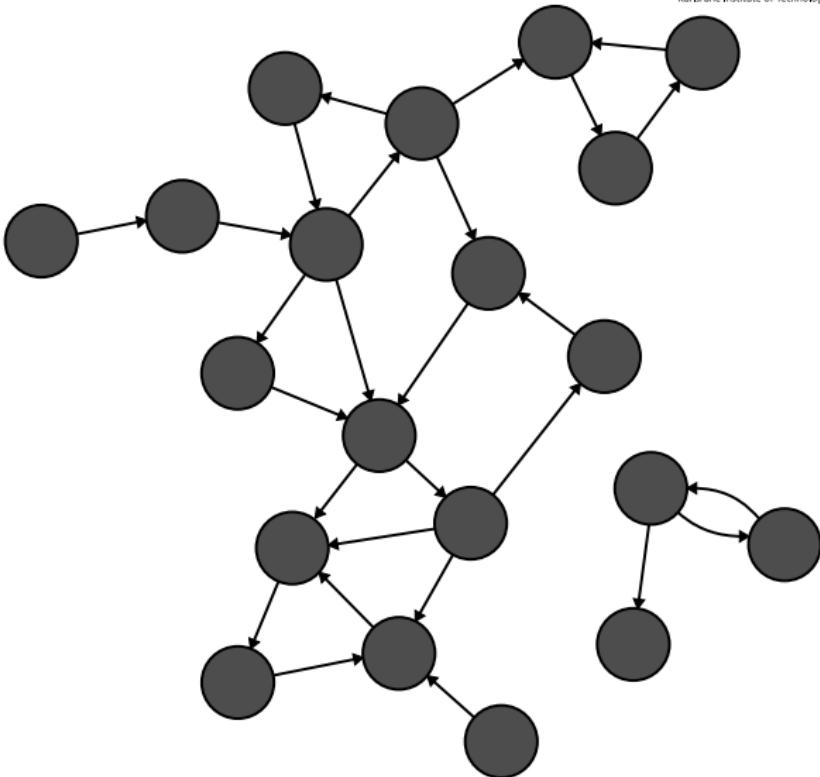


Bild aus Kumar et al. [2017], unverändert.

SCC-Implementierung

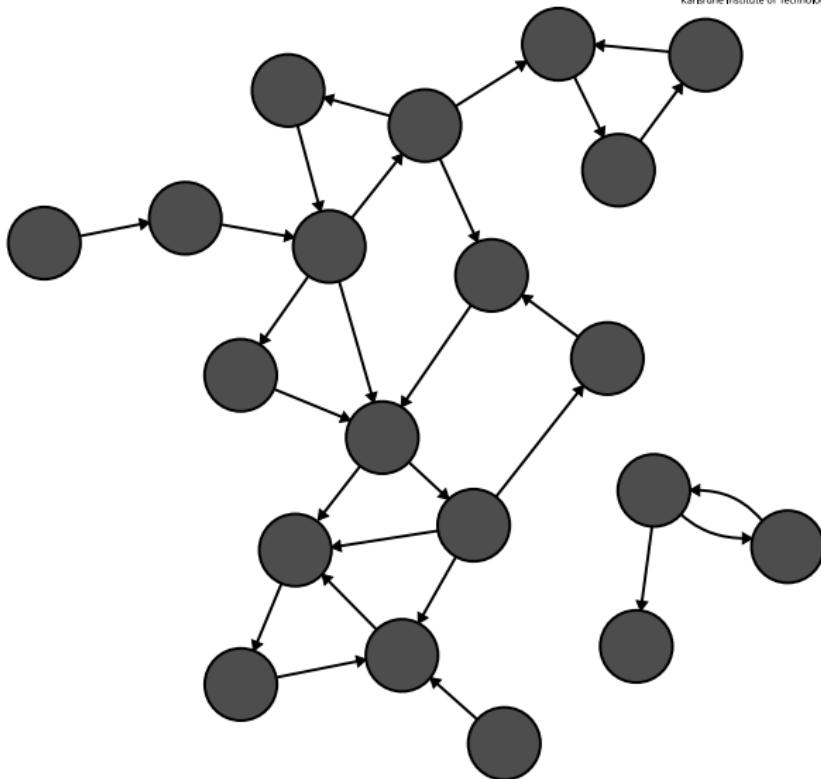
Schritte:



SCC-Implementierung

Schritte:

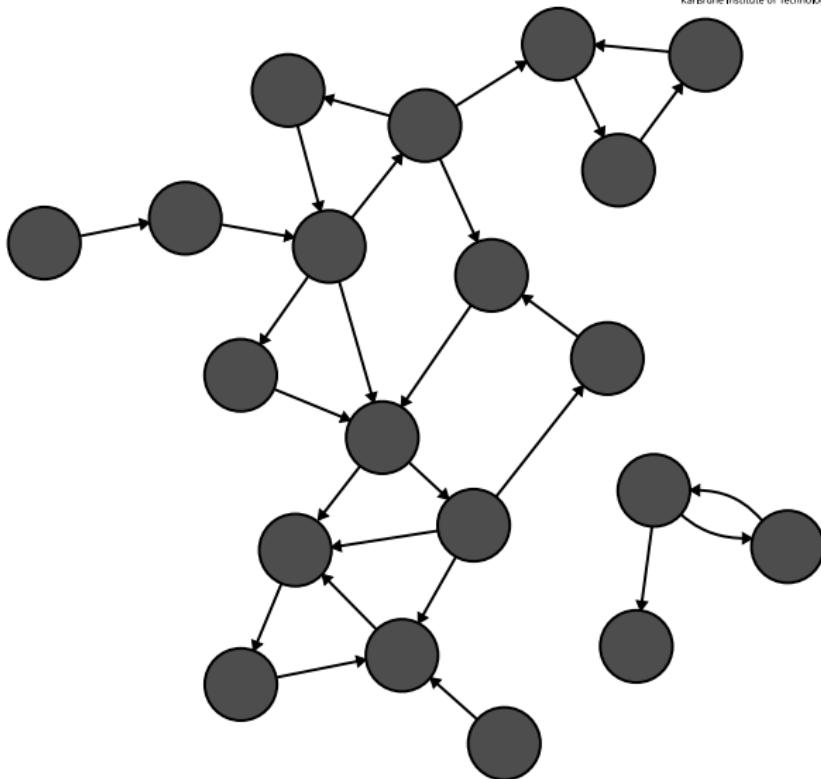
- ① Erstellen des Transponierten Graphen



SCC-Implementierung

Schritte:

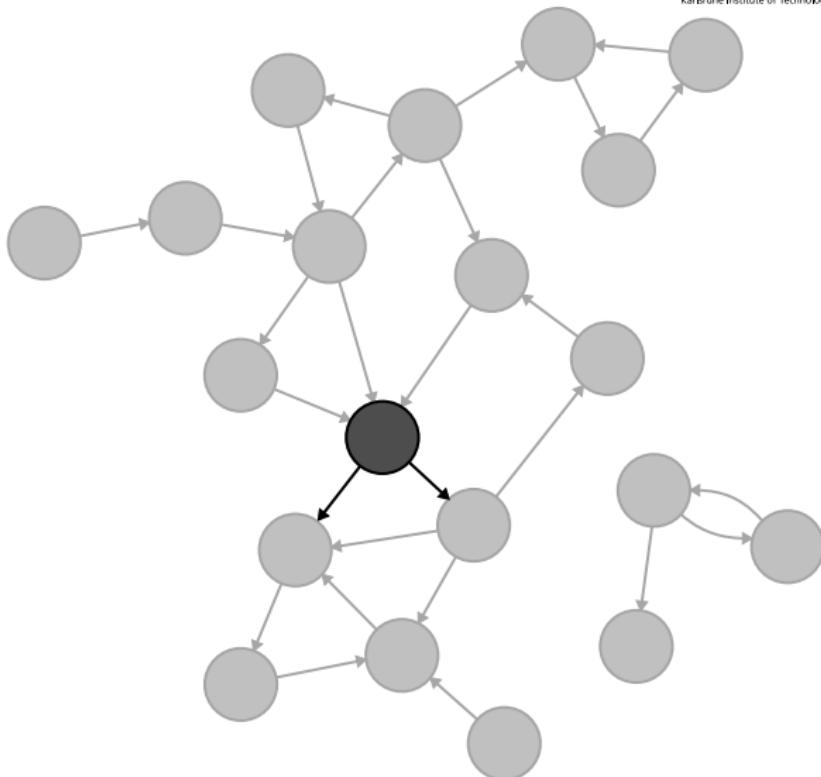
- ① Erstellen des Transponierten Graphen



SCC-Implementierung

Schritte:

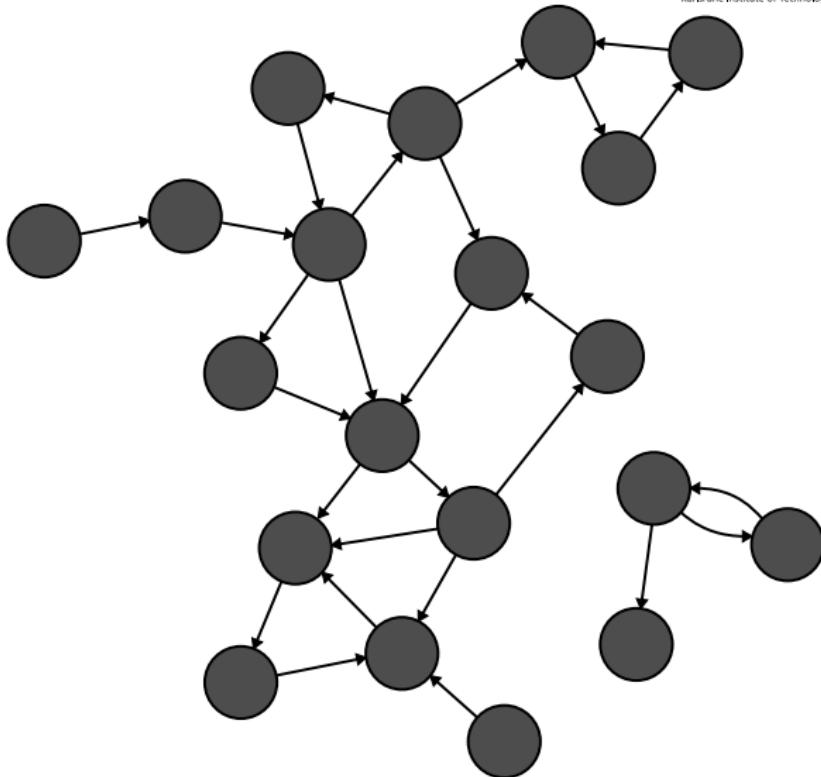
- ① Erstellen des Transponierten Graphen



SCC-Implementierung

Schritte:

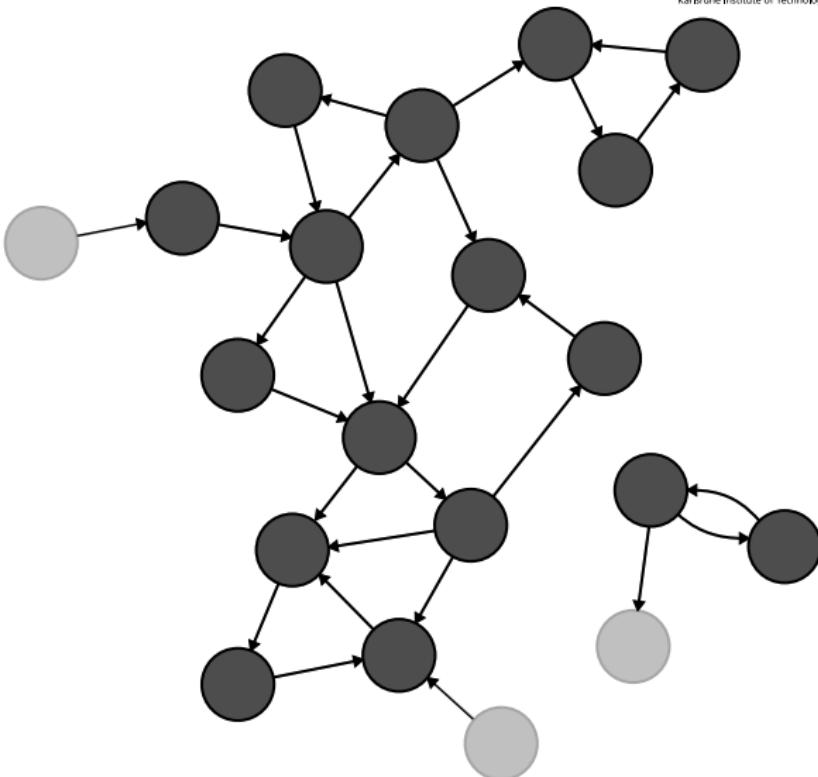
- ① Erstellen des Transponierten Graphen
- ② Trim: Löschen von trivialen SCCs



SCC-Implementierung

Schritte:

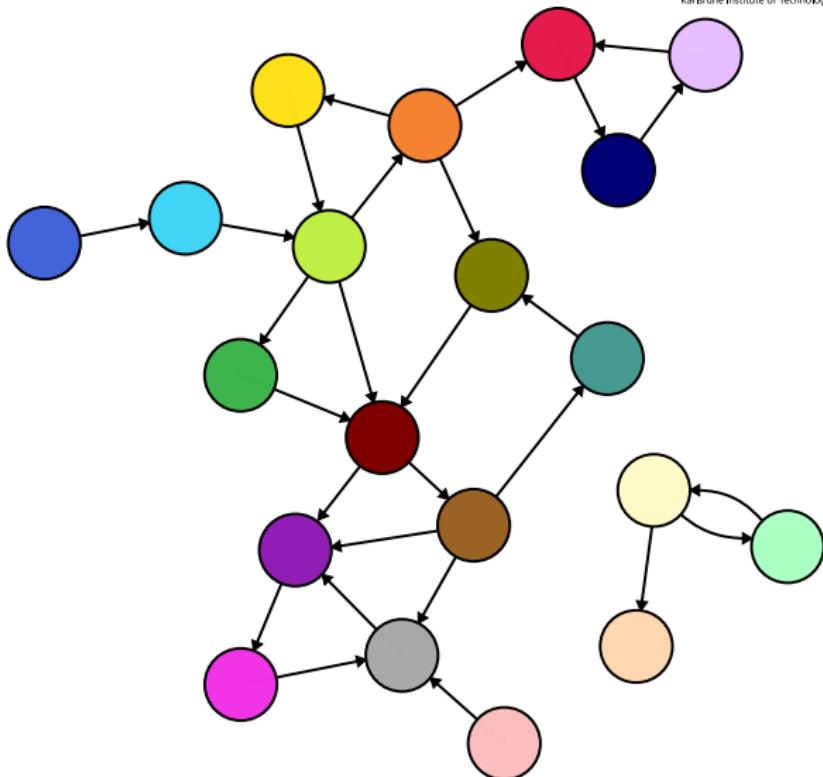
- ① Erstellen des Transponierten Graphen
- ② Trim: Löschen von trivialen SCCs



SCC-Implementierung

Schritte:

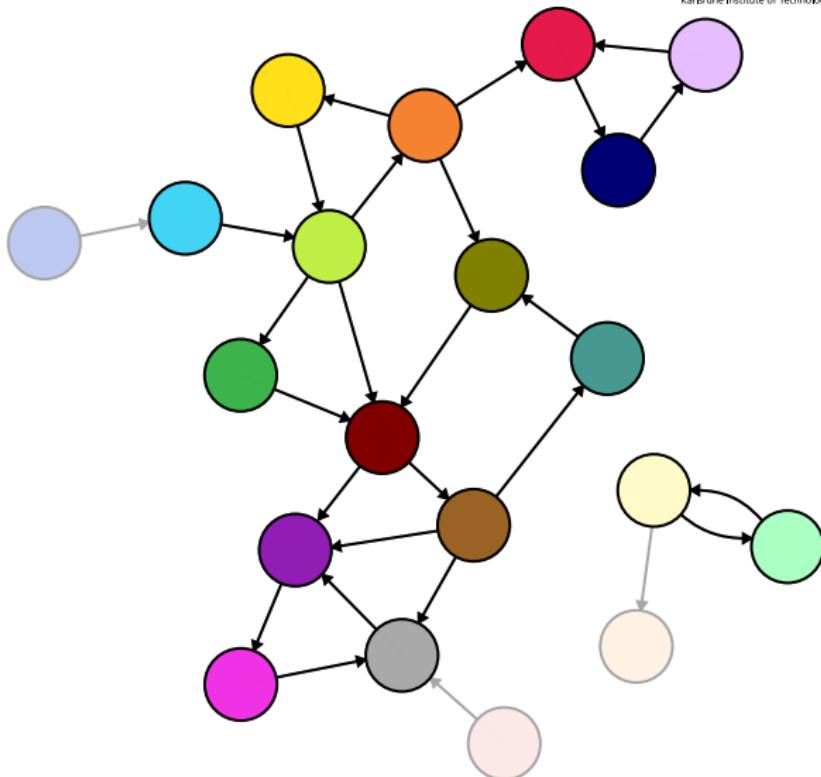
- 1 Erstellen des Transponierten Graphen
 - 2 Trim: Löschen von trivialen SCCs
 - 3 Vorwärts-maxcolor



SCC-Implementierung

Schritte:

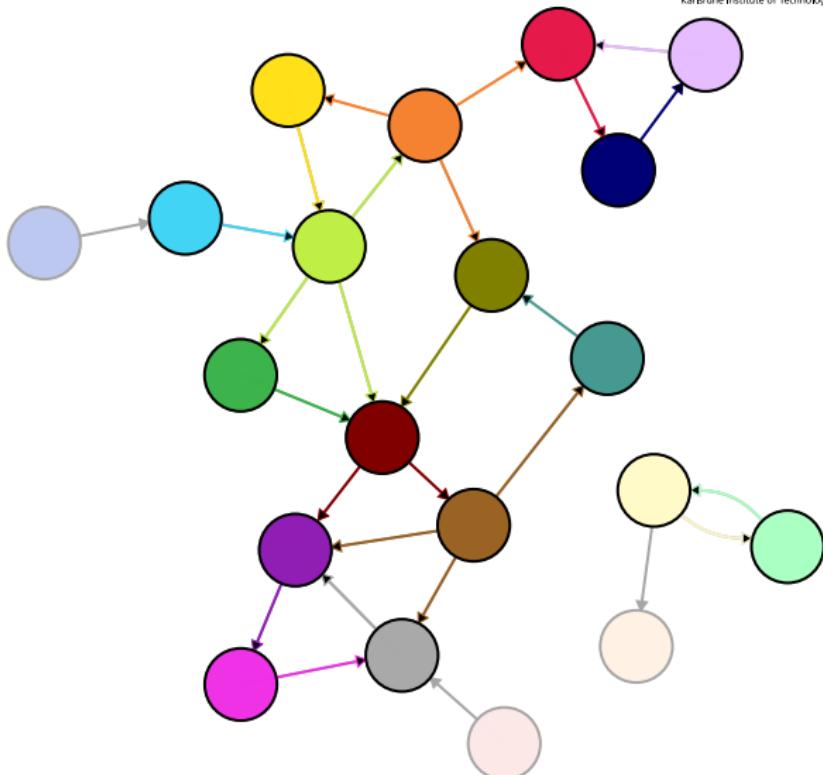
- ① Erstellen des Transponierten Graphen
- ② Trim: Löschen von trivialen SCCs
- ③ Vorwärts-maxcolor



SCC-Implementierung

Schritte:

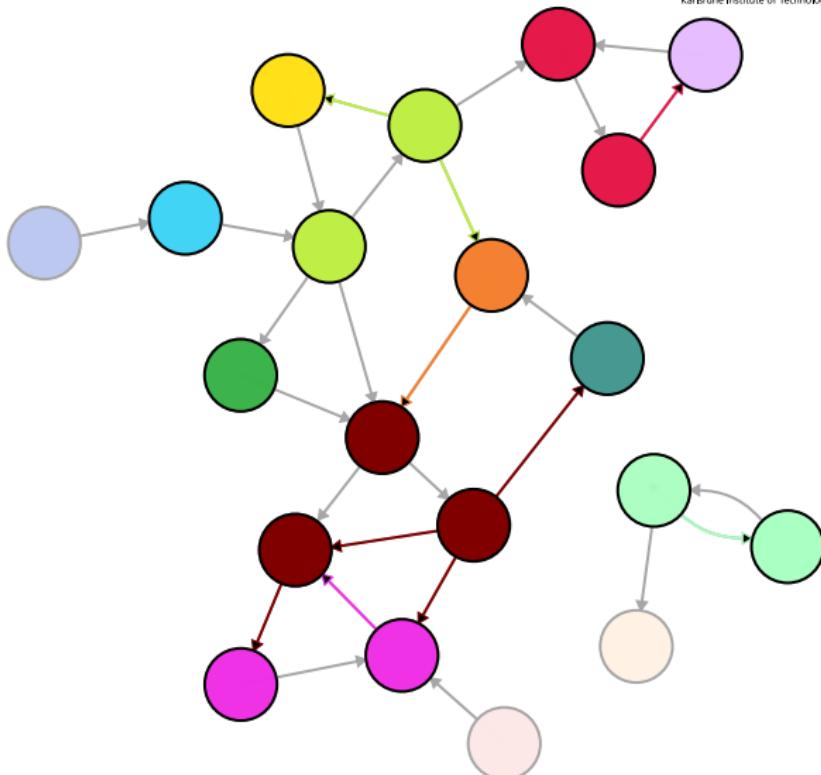
- ① Erstellen des Transponierten Graphen
- ② Trim: Löschen von trivialen SCCs
- ③ Vorwärts-maxcolor



SCC-Implementierung

Schritte:

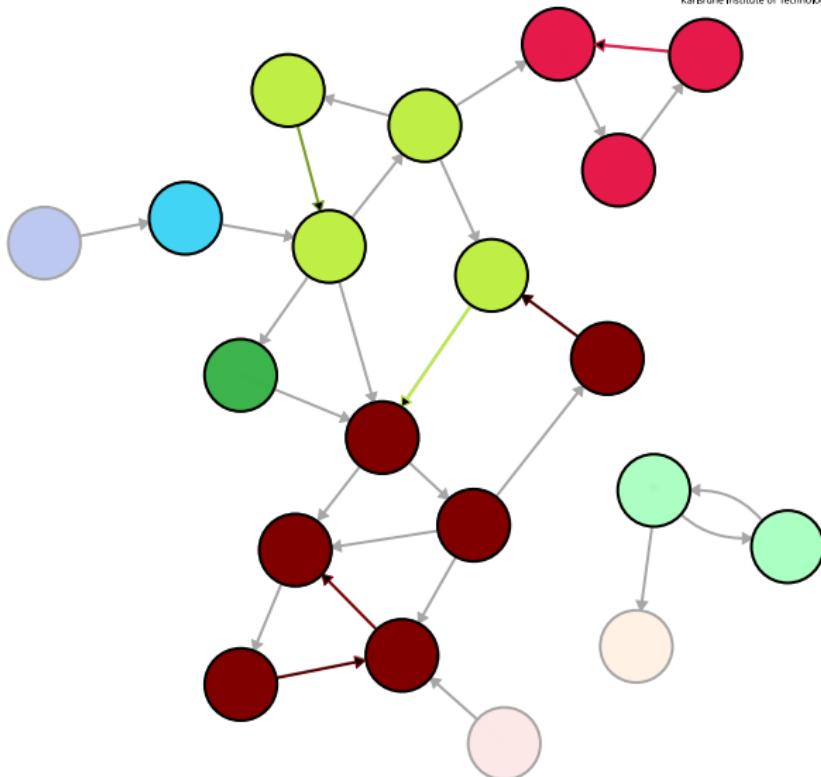
- ① Erstellen des Transponierten Graphen
- ② Trim: Löschen von trivialen SCCs
- ③ Vorwärts-maxcolor



SCC-Implementierung

Schritte:

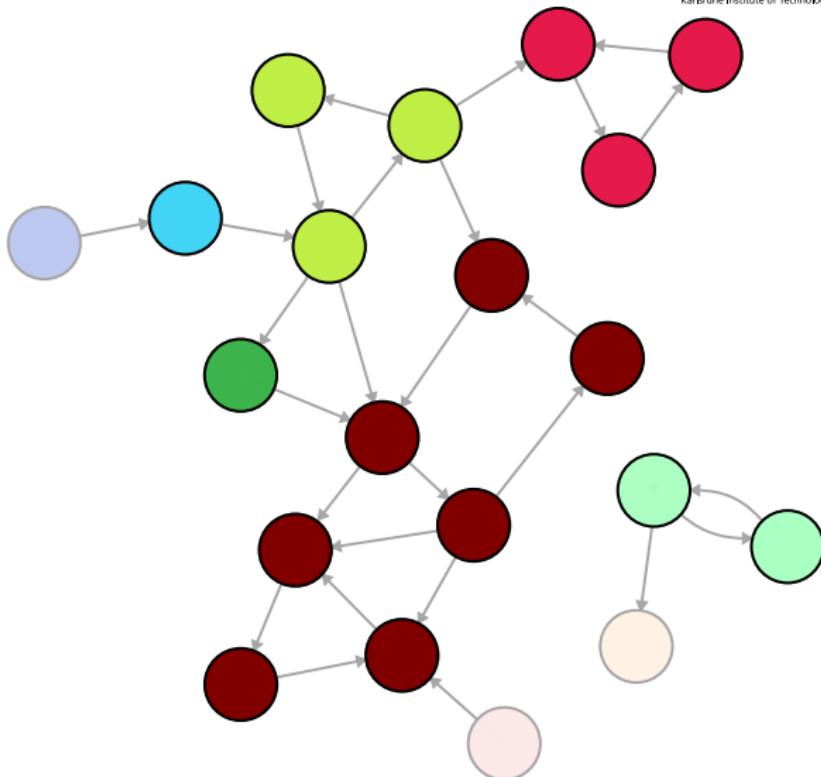
- ① Erstellen des Transponierten Graphen
- ② Trim: Löschen von trivialen SCCs
- ③ Vorwärts-maxcolor



SCC-Implementierung

Schritte:

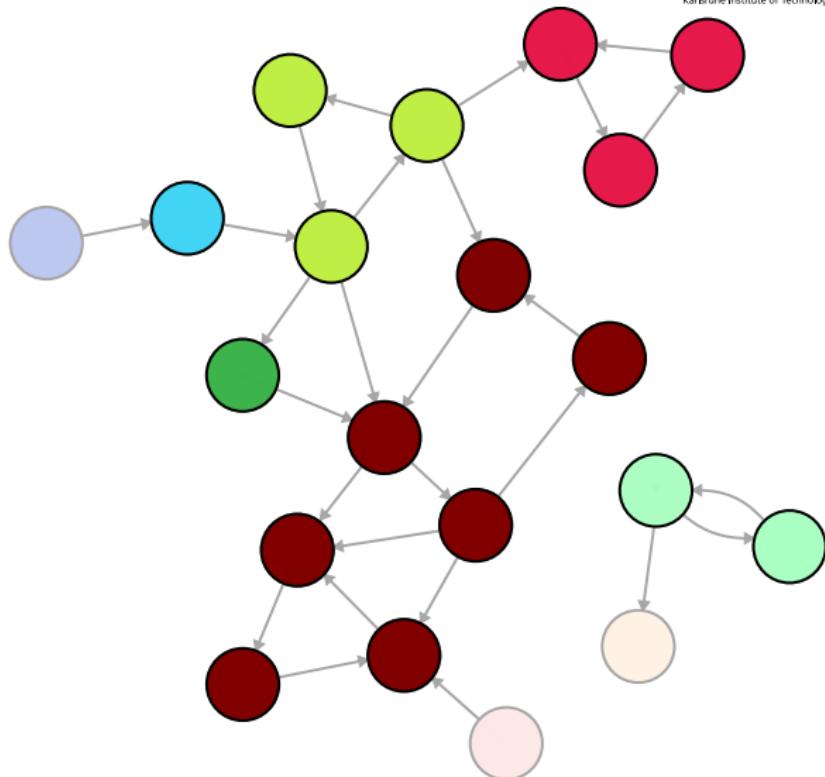
- ① Erstellen des Transponierten Graphen
- ② Trim: Löschen von trivialen SCCs
- ③ Vorwärts-maxcolor



SCC-Implementierung

Schritte:

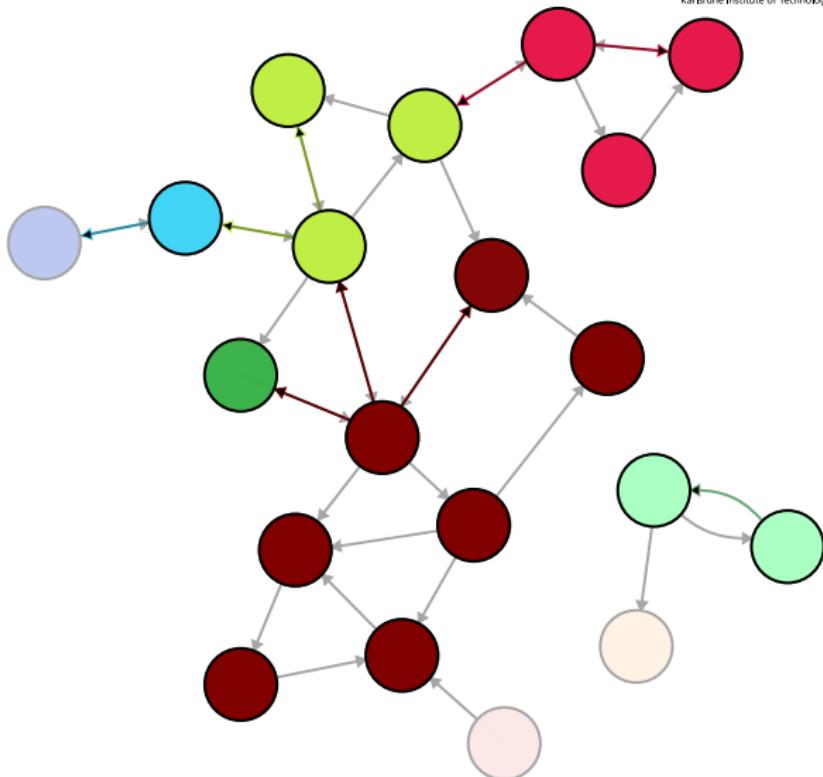
- ① Erstellen des Transponierten Graphen
- ② Trim: Löschen von trivialen SCCs
- ③ Vorwärts-maxcolor
- ④ Rückwärts-maxcolor



SCC-Implementierung

Schritte:

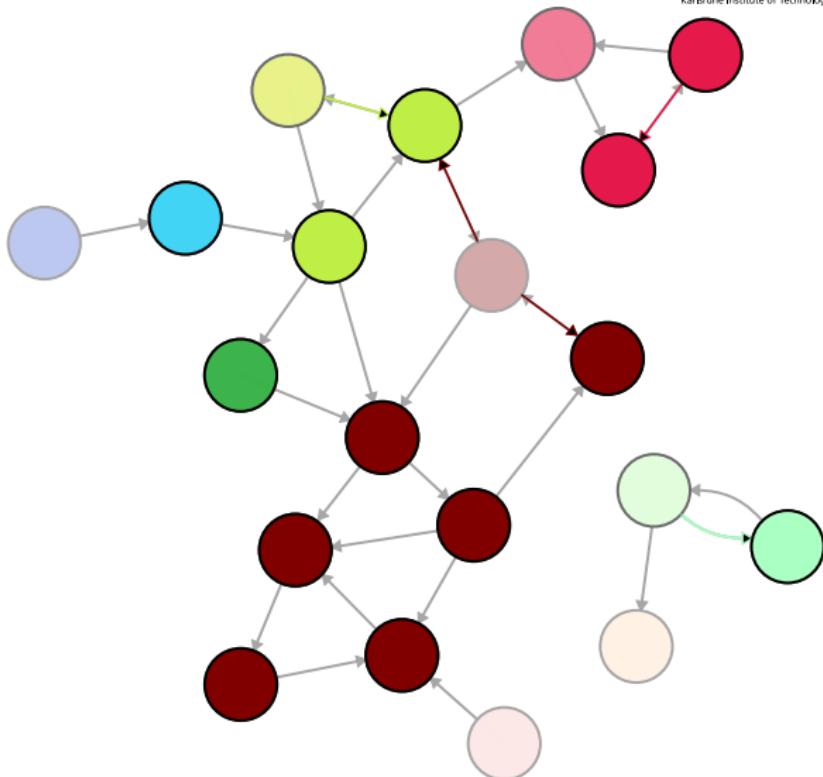
- ① Erstellen des Transponierten Graphen
- ② Trim: Löschen von trivialen SCCs
- ③ Vorwärts-maxcolor
- ④ Rückwärts-maxcolor



SCC-Implementierung

Schritte:

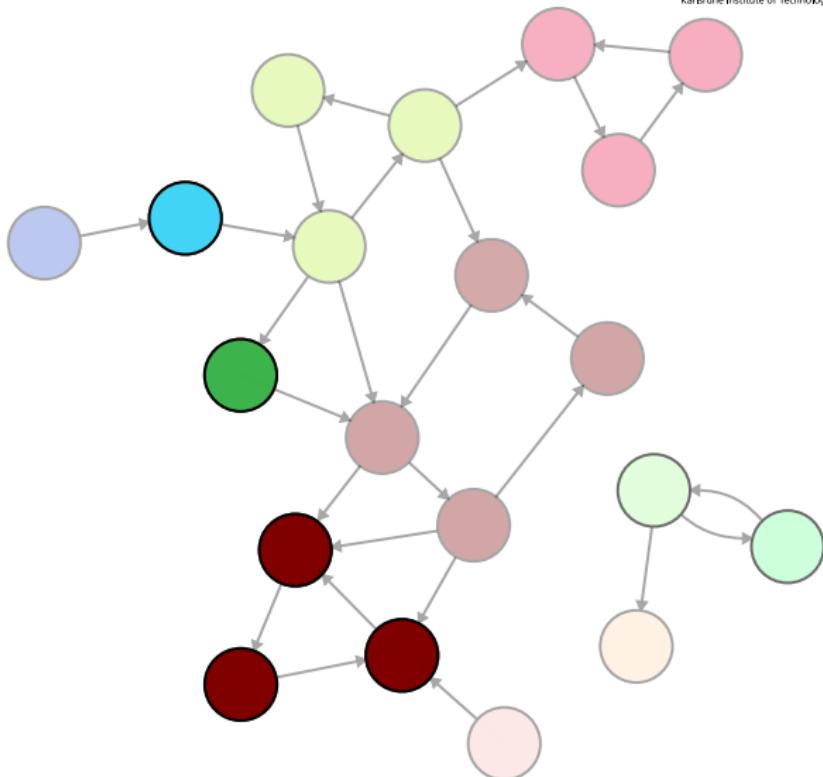
- ① Erstellen des Transponierten Graphen
- ② Trim: Löschen von trivialen SCCs
- ③ Vorwärts-maxcolor
- ④ Rückwärts-maxcolor



SCC-Implementierung

Schritte:

- ① Erstellen des Transponierten Graphen
- ② Trim: Löschen von trivialen SCCs
- ③ Vorwärts-maxcolor
- ④ Rückwärts-maxcolor



Was passiert wenn nur sehr wenige Vertices aktiv sind?

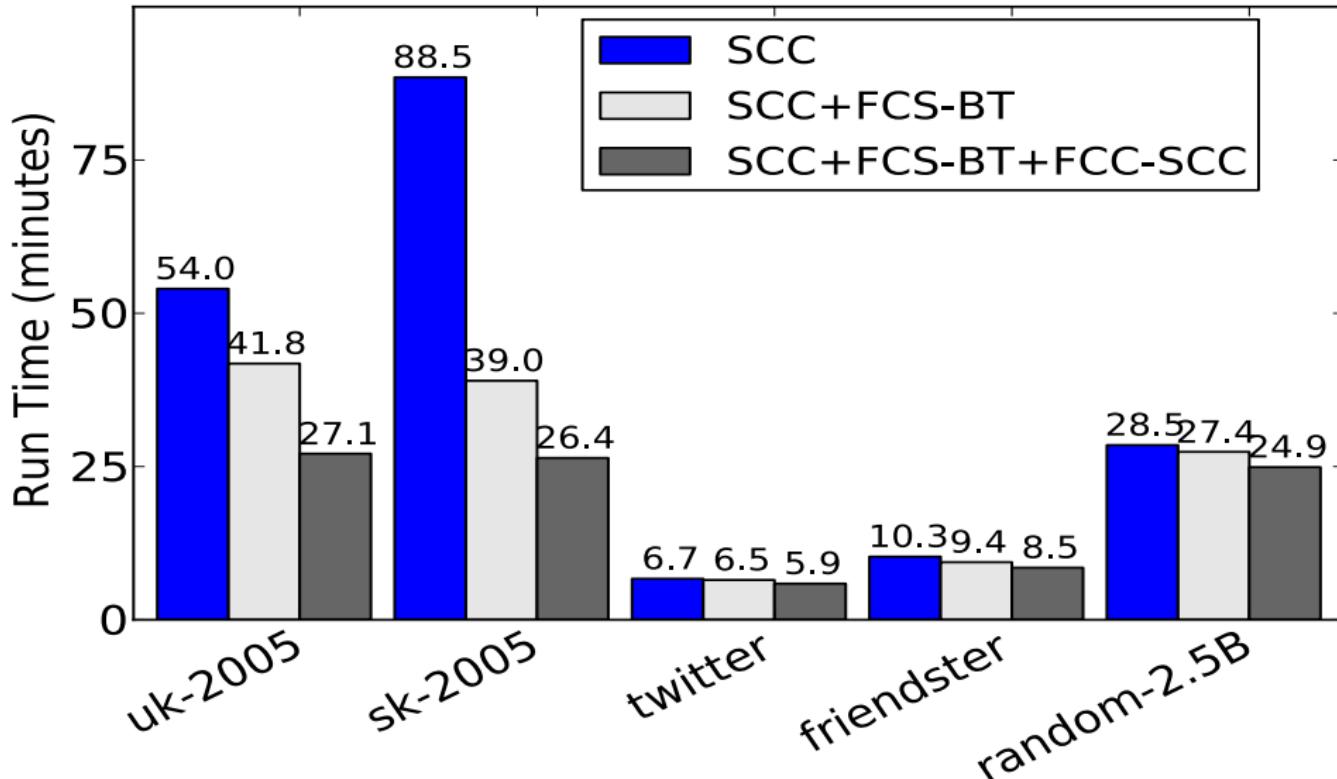
Was passiert wenn nur sehr wenige Vertices aktiv sind?

↪ Wenige Updates erst sequentiell ausführen! (Finishing Computations Serially, FCS)

Was passiert wenn nur sehr wenige Vertices aktiv sind?

→ Wenige Updates erst sequentiell ausführen! (Finishing Computations Serially, FCS)
Je nach Algorithmus 20-60% weniger Supersteps!

Vergleich mit und ohne FCS



Architekturübersicht

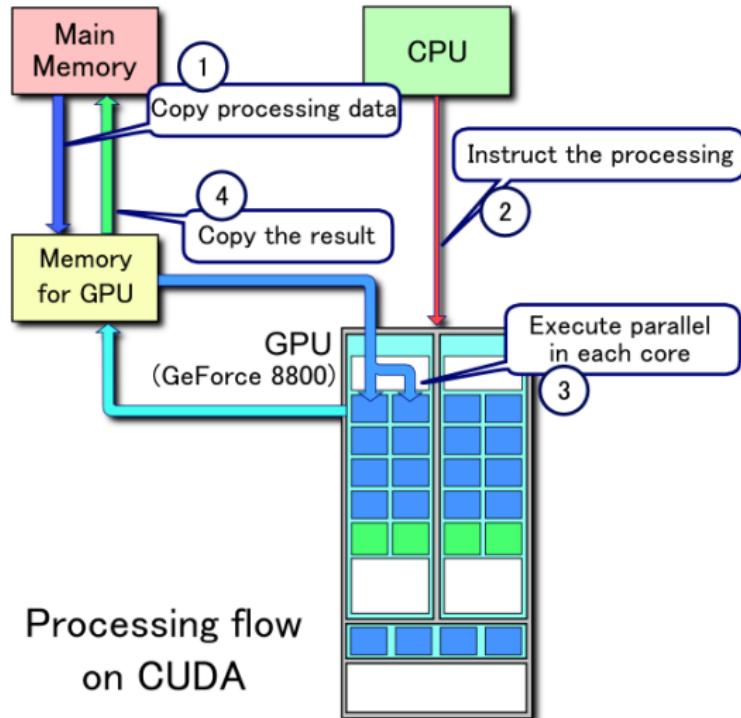


Bild aus Tosaka [2008], unverändert.

Architekturübersicht

- 1 Kopieren der notwendigen Daten zur GPU

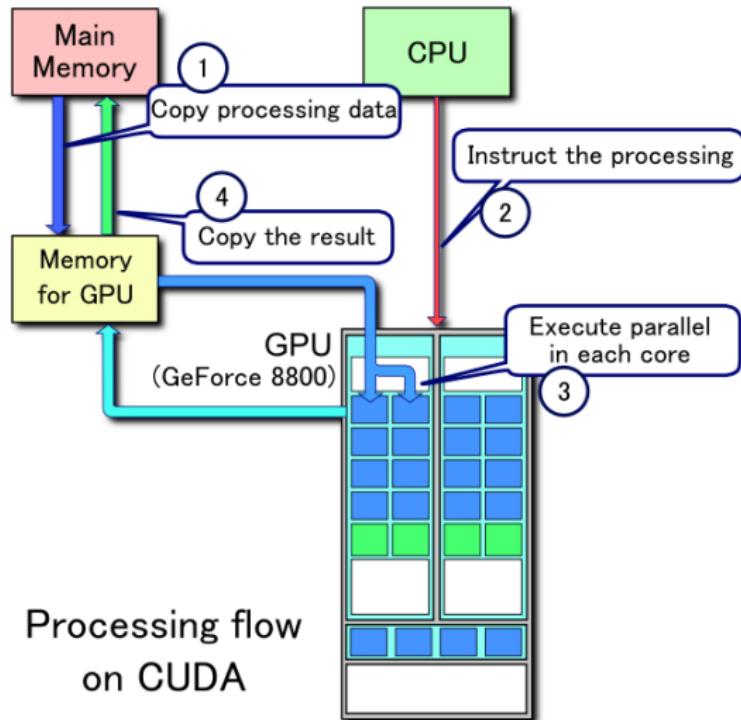


Bild aus Tosaka [2008], unverändert.

Architekturübersicht

- ① Kopieren der notwendigen Daten zur GPU
- ② Auslösen des Ausführens von 'Kerneln'

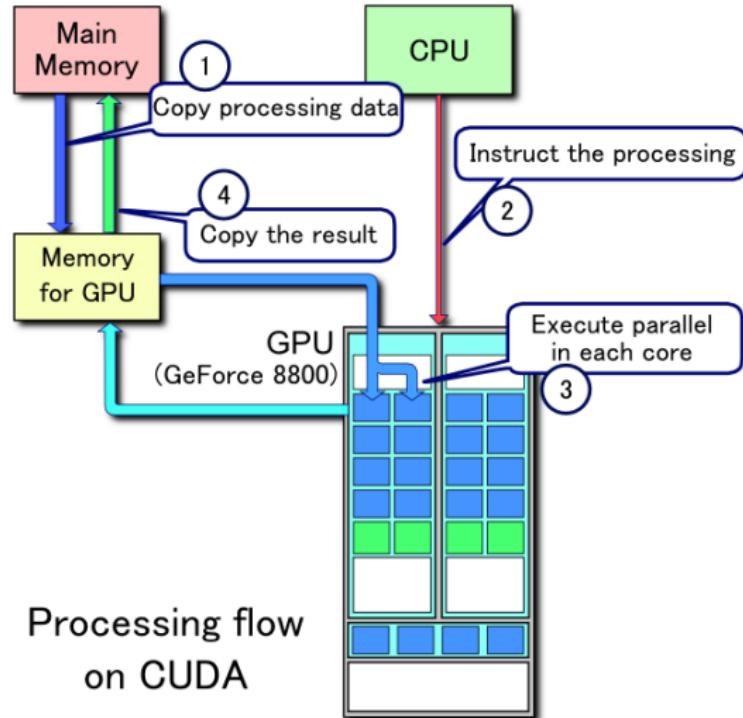


Bild aus Tosaka [2008], unverändert.

Architekturübersicht

- ① Kopieren der notwendigen Daten zur GPU
- ② Auslösen des Ausführens von 'Kerneln'
- ③ Massivparallelausführung

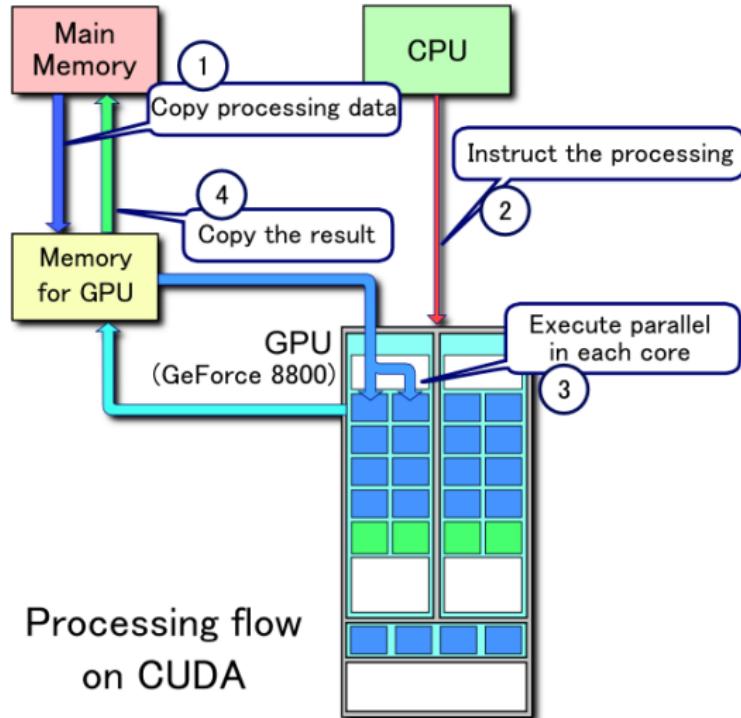


Bild aus Tosaka [2008], unverändert.

Architekturübersicht

- ① Kopieren der notwendigen Daten zur GPU
- ② Auslösen des Ausführens von 'Kerneln'
- ③ Massivparallelausführung
- ④ Zurückkopieren der Ergebnisse

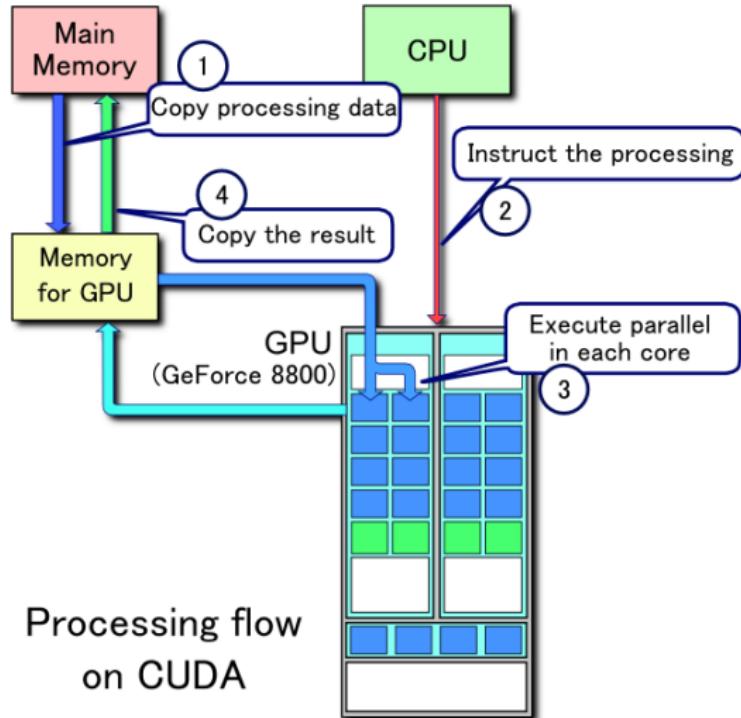


Bild aus Tosaka [2008], unverändert.

Programmablauf

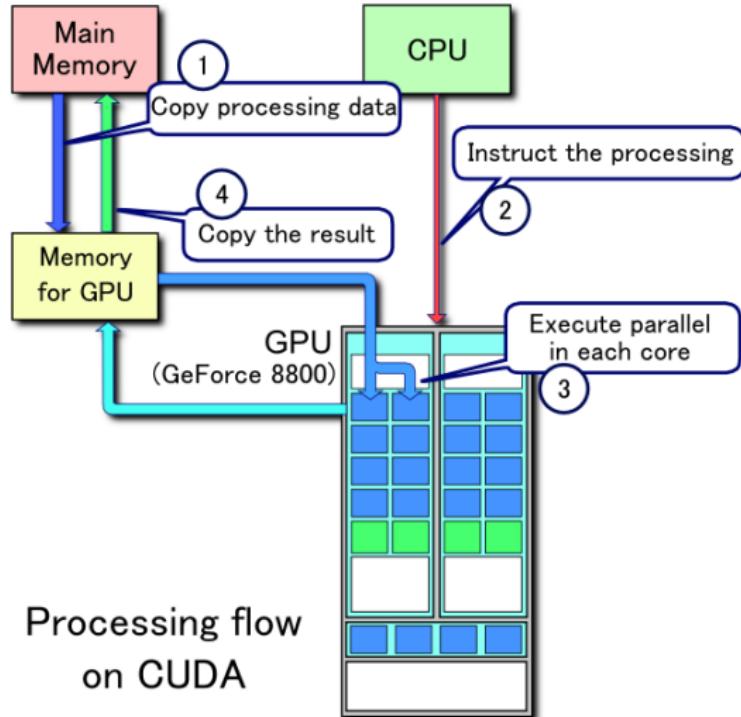


Bild aus Tosaka [2008], unverändert.

Programmablauf

- CUDA ist eine Erweiterung von C/C++

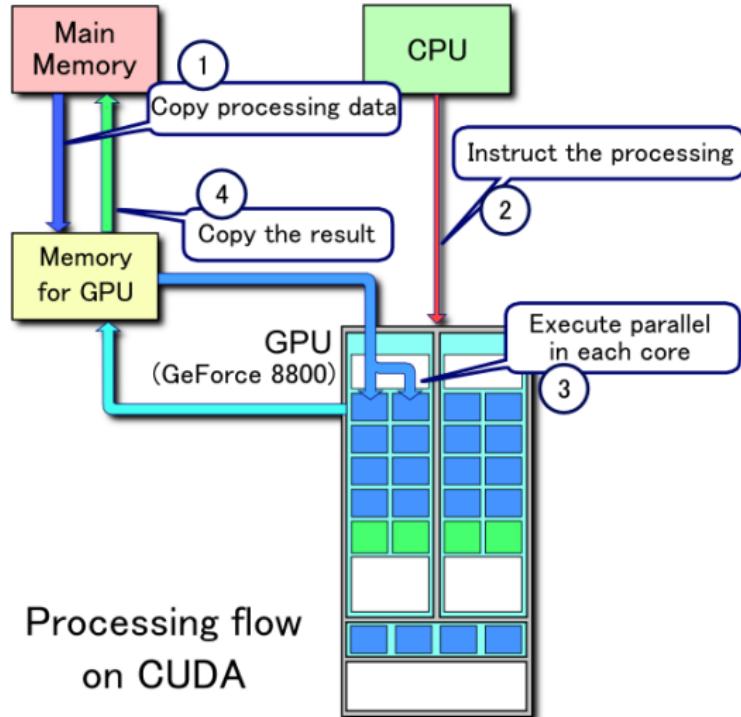


Bild aus Tosaka [2008], unverändert.

Programmablauf

- CUDA ist eine Erweiterung von C/C++
- Hostcode der auf der CPU läuft

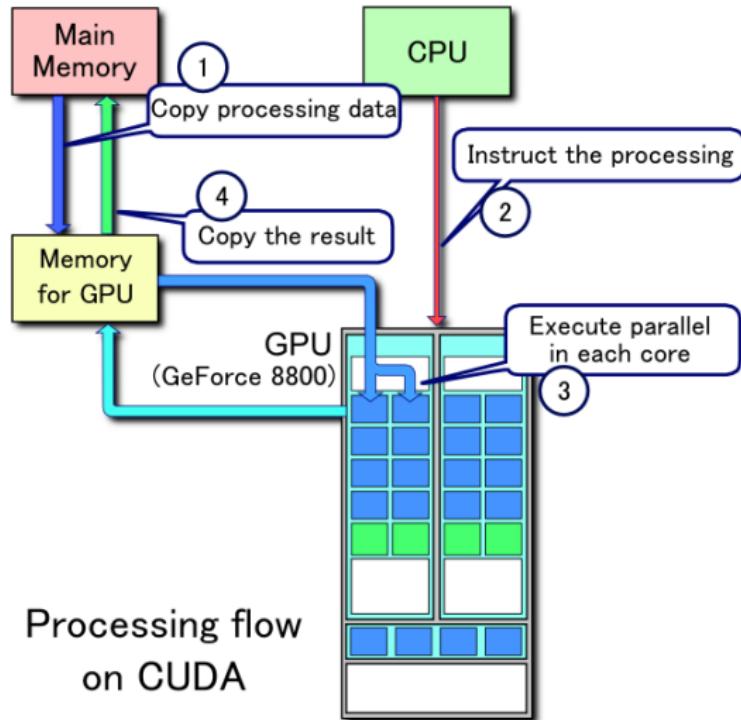


Bild aus Tosaka [2008], unverändert.

Programmablauf

- CUDA ist eine Erweiterung von C/C++
- Hostcode der auf der CPU läuft
- Gerätecode der auf der GPU läuft

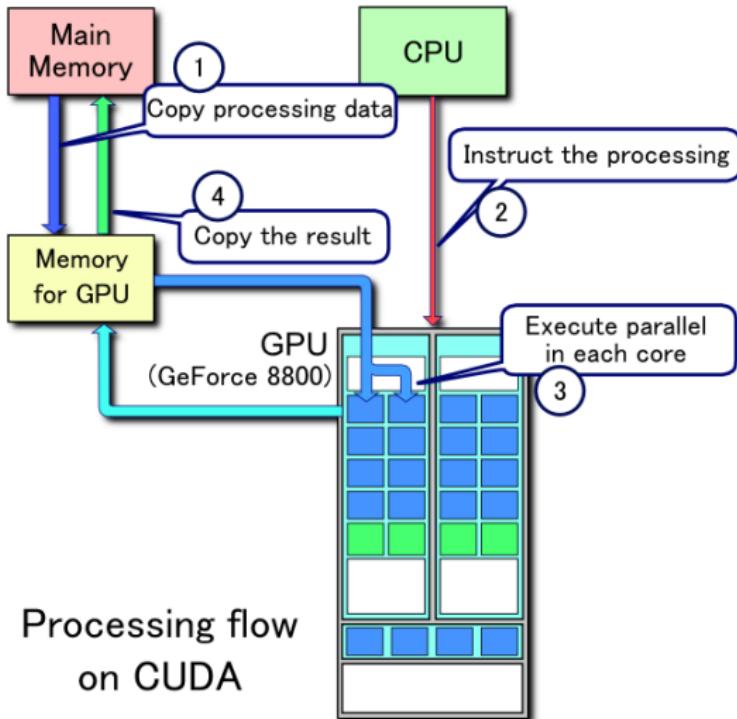


Bild aus Tosaka [2008], unverändert.

Programmablauf

- CUDA ist eine Erweiterung von C/C++
- Hostcode der auf der CPU läuft
- Gerätecode der auf der GPU läuft
- Gerätecode unterteilt in *Kernels*

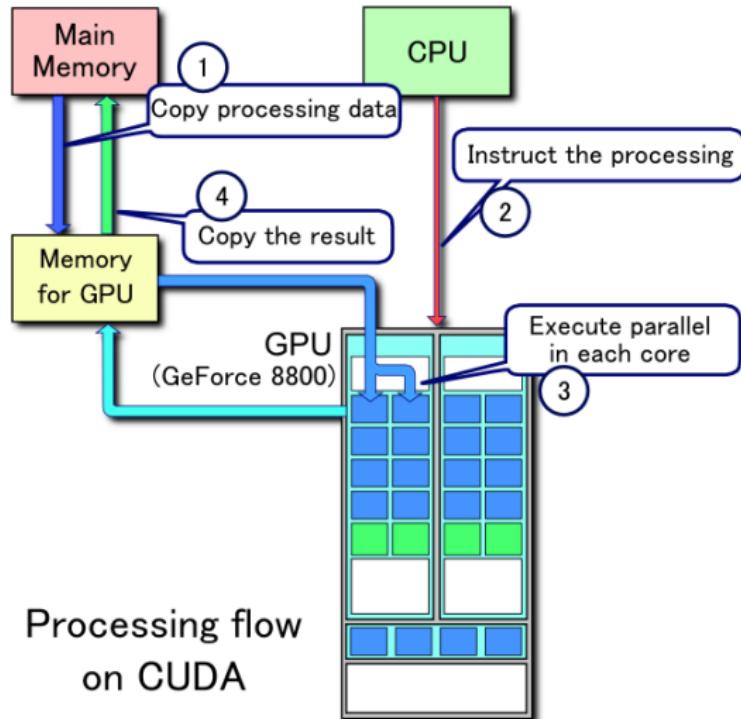


Bild aus Tosaka [2008], unverändert.

Programmablauf

- CUDA ist eine Erweiterung von C/C++
- Hostcode der auf der CPU läuft
- Gerätecode der auf der GPU läuft
- Gerätecode unterteilt in *Kernels*
- Kernel führen die *selben, Sequentiellen* Befehle auf *unabhängigen* Daten-parallelen Threads aus

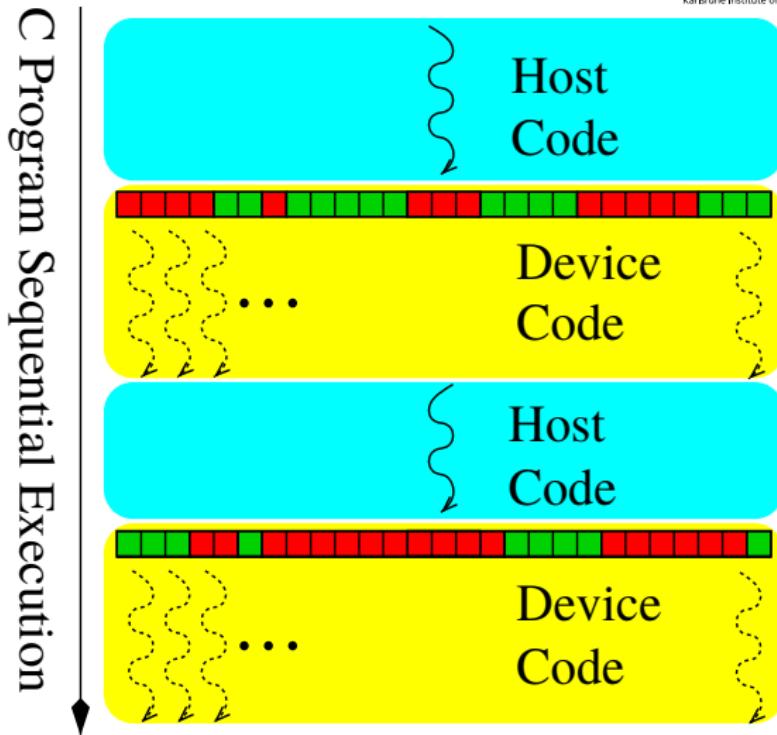


Bild aus Barnat et al. [2011], unverändert.

- Keine komplett neuen Algorithmen

- Keine komplett neuen Algorithmen
- Anpassen von Graphen-Repräsentation auf GPU

- Keine komplett neuen Algorithmen
- Anpassen von Graphen-Repräsentation auf GPU
- Anpassen von Graphen-Operationen auf GPU-Vektorisierung

- Keine komplett neuen Algorithmen
- Anpassen von Graphen-Repräsentation auf GPU
- Anpassen von Graphen-Operationen auf GPU-Vektorisierung
- Achtung: Datenstrukturen für GPUs sind sehr sensibel

- Keine komplett neuen Algorithmen
- Anpassen von Graphen-Repräsentation auf GPU
- Anpassen von Graphen-Operationen auf GPU-Vektorisierung
- Achtung: Datenstrukturen für GPUs sind sehr sensibel
- Potentielle Bottlenecks: Datenzugriffszeiten und Bandbreite

Verwendete Datenstruktur

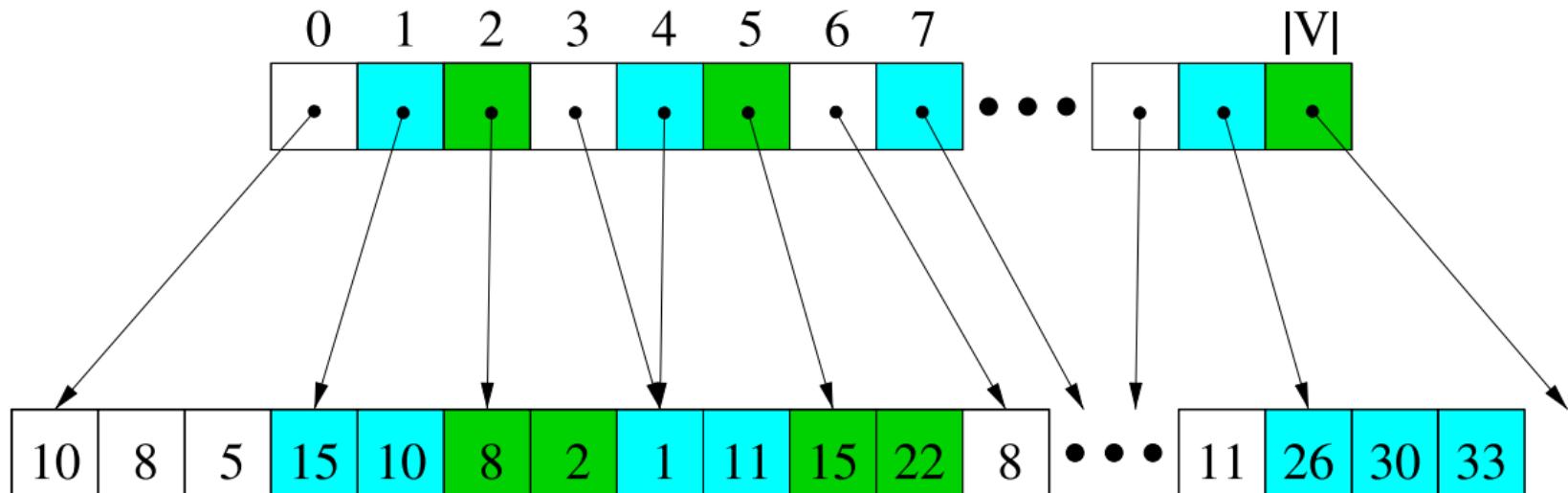


Bild aus Barnat et al. [2011], unverändert.

Funktion FwBw-SEARCH($G = (V, E)$)

```
if  $G = \emptyset$  then
    ↘ return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```

Benötigte Subroutinen:

FwBw Algorithmus

Funktion FwBw-SEARCH($G = (V, E)$)

```
if  $G = \emptyset$  then
    ↘ return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```

Benötigte Subroutinen:

- Vorwärtsabschluss / Rückwärtsabschluss
(Pred / Desc)

Funktion FwBw-SEARCH($G = (V, E)$)

```
if  $G = \emptyset$  then
    return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```

Benötigte Subroutinen:

- Vorwärtsabschluss / Rückwärtsabschluss (Pred / Desc)
- FwBw-Search-Funktion

Kernel FORWARD($G, \text{visited}, \text{terminate}$)

For all $v \in V$:

if ($\text{visited}[v] = \text{true}$) **then**

for all $u \in V$. $(v, u) \in E$ **do**

if $v \rightsquigarrow u \wedge \text{visited}[u] = \text{false}$ **then**

$\text{visited}[u], \text{terminate} \leftarrow \text{true}, \text{false}$

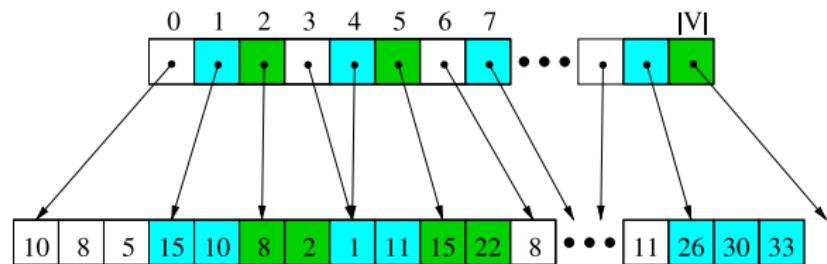


Bild aus Barnat et al. [2011], unverändert.

Kernel FORWARD($G, \text{visited}, \text{terminate}$)

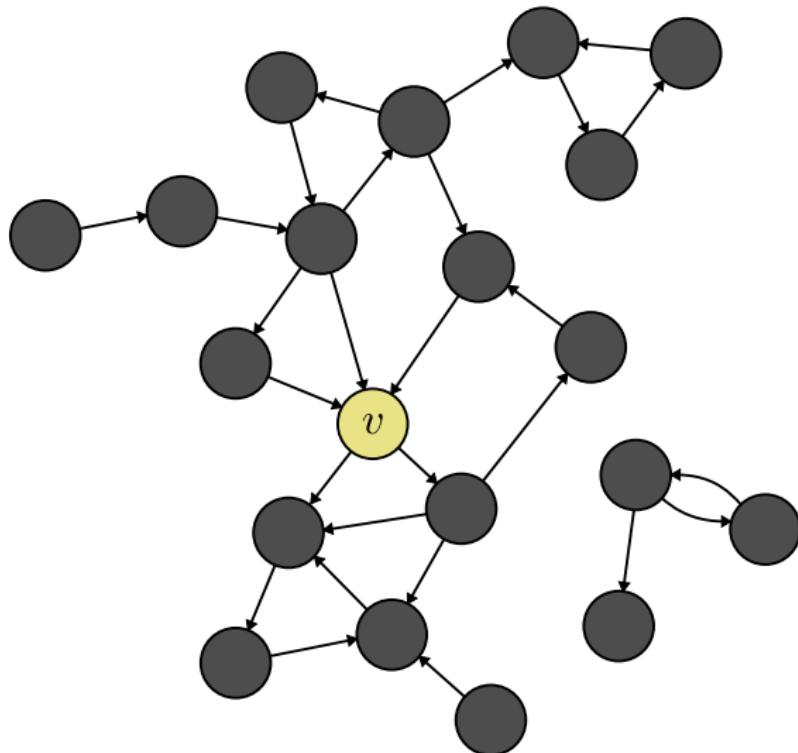
For all $v \in V$:

if ($\text{visited}[v] = \text{true}$) **then**

for all $u \in V. (v, u) \in E$ **do**

if $v \rightsquigarrow u \wedge \text{visited}[u] = \text{false}$ **then**

$\text{visited}[u], \text{terminate} \leftarrow \text{true}, \text{false}$



Kernel FORWARD($G, \text{visited}, \text{terminate}$)

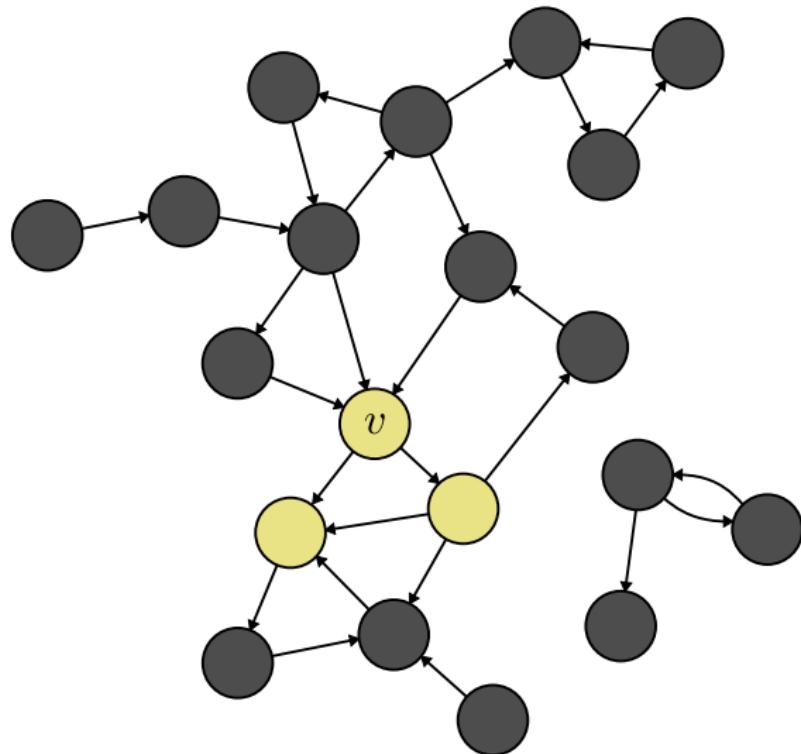
For all $v \in V$:

if ($\text{visited}[v] = \text{true}$) **then**

for all $u \in V. (v, u) \in E$ **do**

if $v \rightsquigarrow u \wedge \text{visited}[u] = \text{false}$ **then**

$\text{visited}[u], \text{terminate} \leftarrow \text{true}, \text{false}$



Kernel FORWARD($G, \text{visited}, \text{terminate}$)

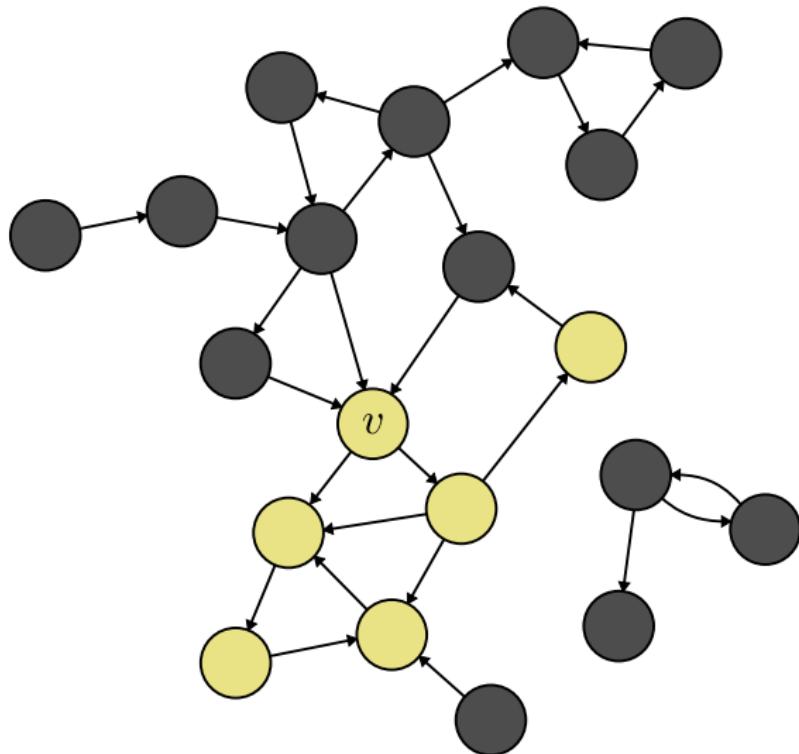
For all $v \in V$:

if ($\text{visited}[v] = \text{true}$) **then**

for all $u \in V. (v, u) \in E$ **do**

if $v \rightsquigarrow u \wedge \text{visited}[u] = \text{false}$ **then**

$\text{visited}[u], \text{terminate} \leftarrow \text{true}, \text{false}$



Kernel FORWARD($G, \text{visited}, \text{terminate}$)

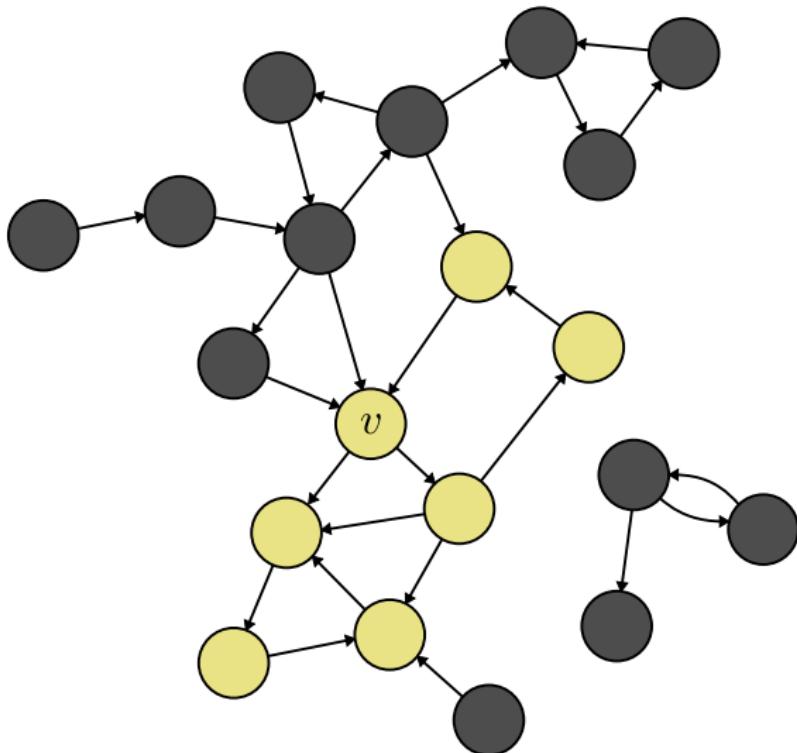
For all $v \in V$:

if ($\text{visited}[v] = \text{true}$) **then**

for all $u \in V. (v, u) \in E$ **do**

if $v \rightsquigarrow u \wedge \text{visited}[u] = \text{false}$ **then**

$\text{visited}[u], \text{terminate} \leftarrow \text{true}, \text{false}$



Program FWD-REACH($G = (V, E)$, $P \subseteq V$)

// Host Code

Out: $\forall v \in V : \text{visited}[v] = \text{true} \Leftrightarrow \exists u \in P : (u, v) \in E^*$

```

for all  $u \in P$  do
   $\text{visited}[u] \leftarrow \text{true}$ 
terminate  $\leftarrow \text{false}$ 
while  $\text{terminate} = \text{false}$  do
   $\text{terminate} \leftarrow \text{true}$ 
  FORWARD ( $V, \text{visited}, \text{terminate}$ )

```

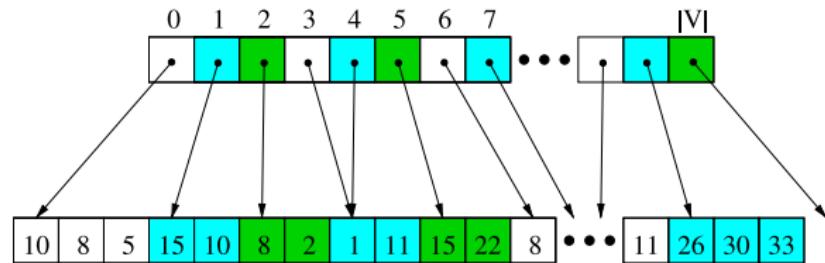


Bild aus Barnat et al. [2011], unverändert.

Rückwärtsabschluss

Möglichkeiten:

Möglichkeiten:

- Vorwärtsabschluss über den Transponierten Graph

Möglichkeiten:

- Vorwärtsabschluss über den Transponierten Graph
- Eigener, neuer Kernel

Möglichkeiten:

- Vorwärtsabschluss über den Transponierten Graph
- Eigener, neuer Kernel

Ergebnis: Eigener Kernel platzeffizienter, aber Vorwärtsabschluss um einiges schneller (nach Barnat et al. [2011]).

Verbesserungen von FWBW

■ Anwenden von Trim-Kernel

Kernel TRIM($G, eliminated, terminate$)

For all $v \in V$:

```
if ( $eliminated[v] = false$ ) then
     $elim \leftarrow true$ 
    if  $\exists u \in V. (u, v) \in E \wedge v \rightsquigarrow u$  then
         $elim \leftarrow false$ 
    if  $elim = true$  then
         $eliminated[v], terminate \leftarrow true, false$ 
```

Verbesserungen von FWBW

- Anwenden von Trim-Kernel
- Verbesserte Pivot Auswahl

Funktion FwBw-SEARCH($G = (V, E)$)

```
if  $G = \emptyset$  then
    ↘ return
     $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```

- Anwenden von Trim-Kernel
- Verbesserte Pivot Auswahl
- Üblicherweise Pseudorandom-Zufallszahlen

Verbesserungen von FWBW

- Anwenden von Trim-Kernel
- Verbesserte Pivot Auswahl

- Üblicherweise Pseudorandom-Zufallszahlen
- Tendenziell keine gute Verteilungen

- Anwenden von Trim-Kernel
- Verbesserte Pivot Auswahl

- Üblicherweise Pseudorandom-Zufallszahlen
- Tendenziell keine gute Verteilungen
- Vorschlag: Jeder Vertex im Subgraph schreibt sich in einen Vektor

- Anwenden von Trim-Kernel
- Verbesserte Pivot Auswahl

- Üblicherweise Pseudorandom-Zufallszahlen
- Tendenziell keine gute Verteilungen
- Vorschlag: Jeder Vertex im Subgraph schreibt sich in einen Vektor
- Letzter Vertex ist neuer Pivot!

- Anwenden von Trim-Kernel
- Verbesserte Pivot Auswahl

- Üblicherweise Pseudorandom-Zufallszahlen
- Tendenziell keine gute Verteilungen
- Vorschlag: Jeder Vertex im Subgraph schreibt sich in einen Vektor
- Letzter Vertex ist neuer Pivot!
- Nichttrivial: Wo dieser Vektor sein sollte

Geschwindigkeit

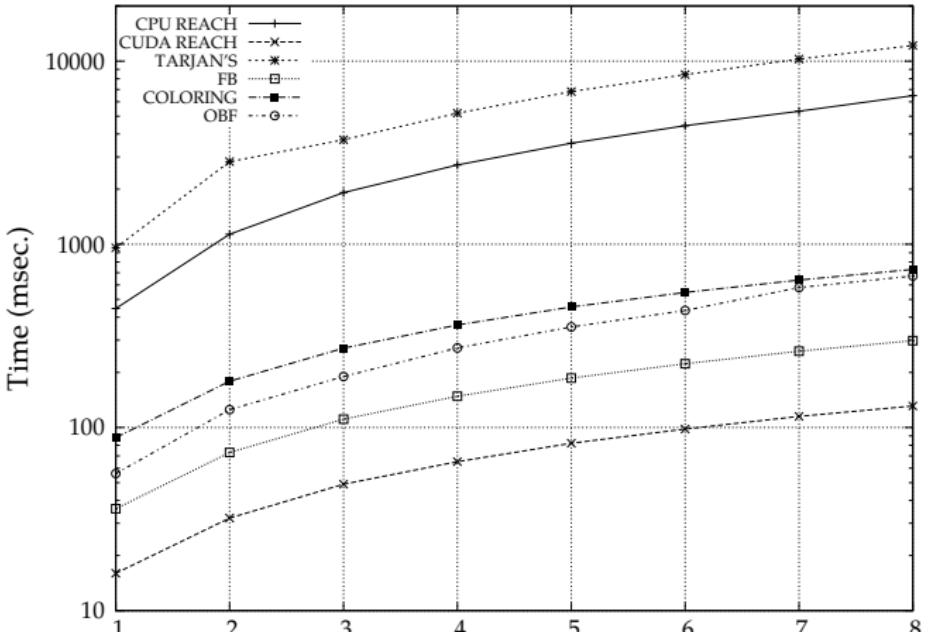


Figure 3. Run-times for Random graphs in milliseconds.

Bild aus Barnat et al. [2011], unverändert.

- CPU-REACH
- CUDA-REACH
- TARJANS
- FB: CUDA-basiert, mit Trim
- Coloring: CUDA-basiert
- OBF: CUDA-basiert mit Coloring und Trim

Die Zahlen bestimmen verschiedene Graphen. Durchschnittliche Anzahl an Verbindungen eines Knoten: 12

Geschwindigkeit

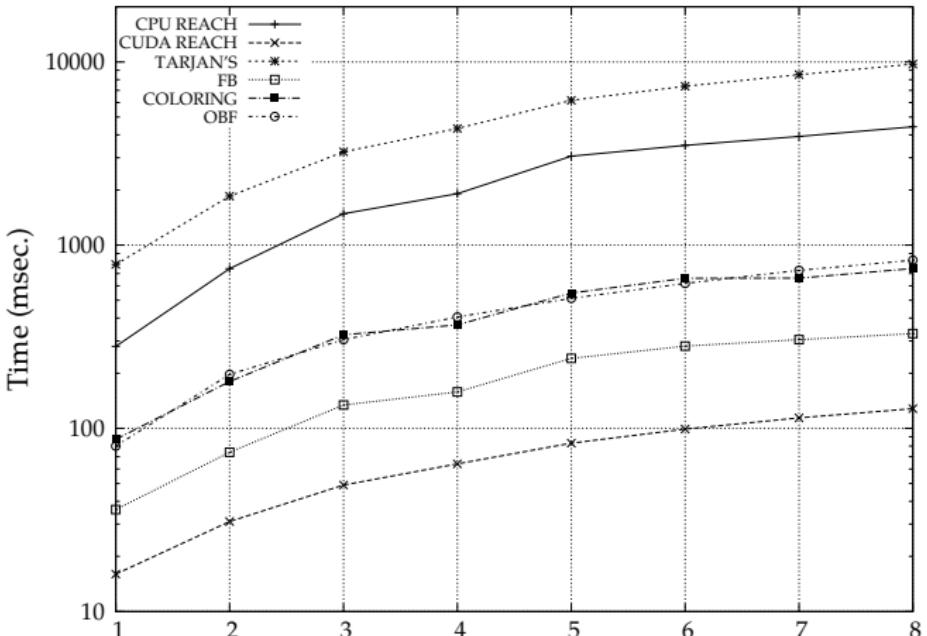


Figure 4. Run-times for R-MAT graphs in milliseconds.
 Bild aus Barnat et al. [2011], unverändert.

- CPU-REACH
- CUDA-REACH
- TARJANS
- FB: CUDA-basiert, mit Trim
- Coloring: CUDA-basiert
- OBF: CUDA-basiert mit Coloring und Trim

Die Zahlen bestimmen verschiedene Graphen. Durchschnittliche Anzahl an Verbindungen eines Knoten: 12

Geschwindigkeit

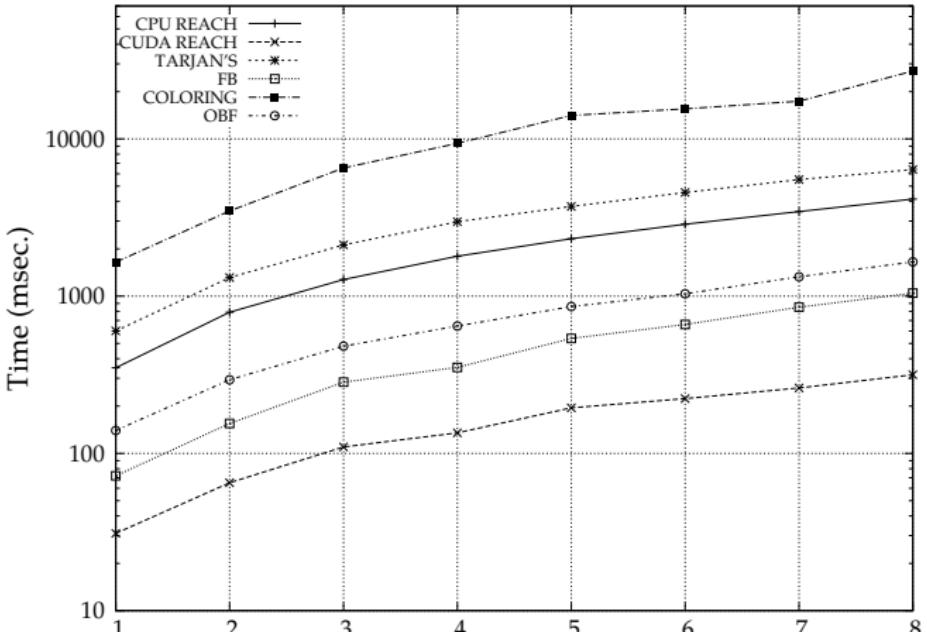


Figure 5. Run-times for SCCA #2 graphs in milliseconds.

Bild aus Barnat et al. [2011], unverändert.

- CPU-REACH
- CUDA-REACH
- TARJANS
- FB: CUDA-basiert, mit Trim
- Coloring: CUDA-basiert
- OBF: CUDA-basiert mit Coloring und Trim

Die Zahlen bestimmen verschiedene Graphen. Durchschnittliche Anzahl an Verbindungen eines Knoten: 12

Bisheriger Ansatz: Ausnutzen der Architektur

Bisheriger Ansatz: Ausnutzen der Architektur
Was wenn wir etwas über die Struktur des Graphen wissen?

Was sind das für Graphen

Beispiele:

- Links innerhalb von Live-Journal (Web)
- Verbindungen zwischen Flickr-Nutzern (Sozial)
- Links innerhalb der Baidu Enzyklopädie (Web)
- Links innerhalb des Englischen Wikipedia (Web)
- Verbindungen zwischen Friendster-Benutzern (Sozial)
- Verbindungen zwischen Twitternutzern (Sozial)
- Verbindungen zwischen Orkut-Benutzern (Sozial)
- Zitierungen zwischen US-Patenten
- Kalifornisches Straßennetz

Graphen in der Echten Welt

Eigenschaften:

Eigenschaften:

- Eine große Komponente und viele kleine

Graphen in der Echten Welt

Eigenschaften:

- Eine große Komponente und viele kleine

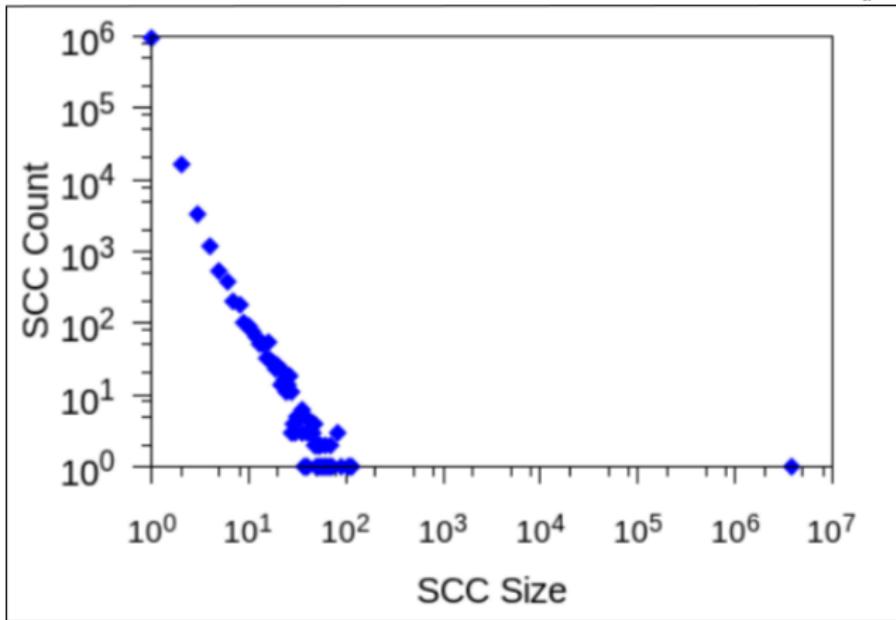


Bild aus Hong et al. [2013], unverändert.

Graphen in der Echten Welt

Eigenschaften:

- Eine große Komponente und viele kleine
- Durchmesser sind extrem klein (Avg: ≤ 6)

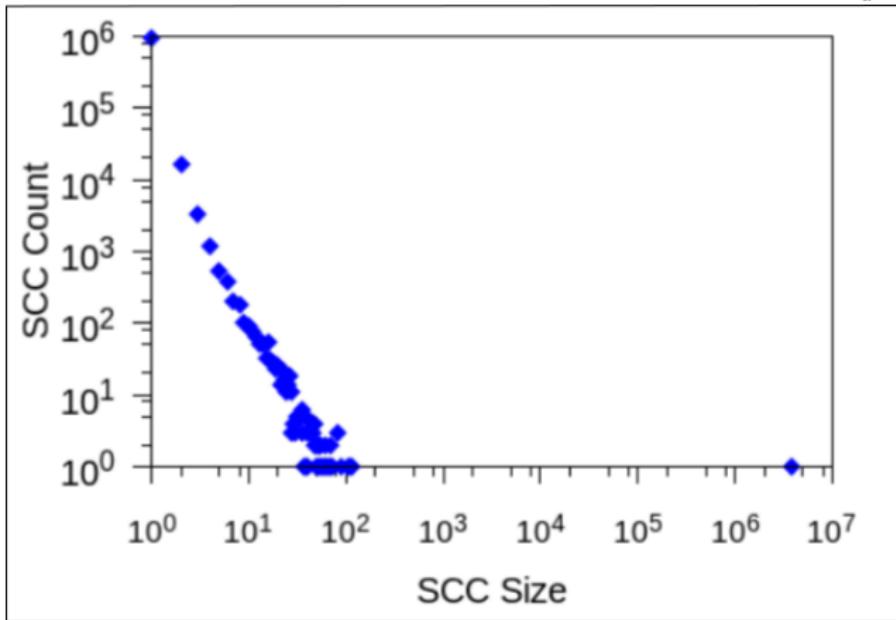


Bild aus Hong et al. [2013], unverändert.

Graphen in der Echten Welt

Eigenschaften:

- Eine große Komponente und viele kleine
- Durchmesser sind extrem klein (Avg: ≤ 6)
 - Facebook: Avg 4.57 im Februar 2016

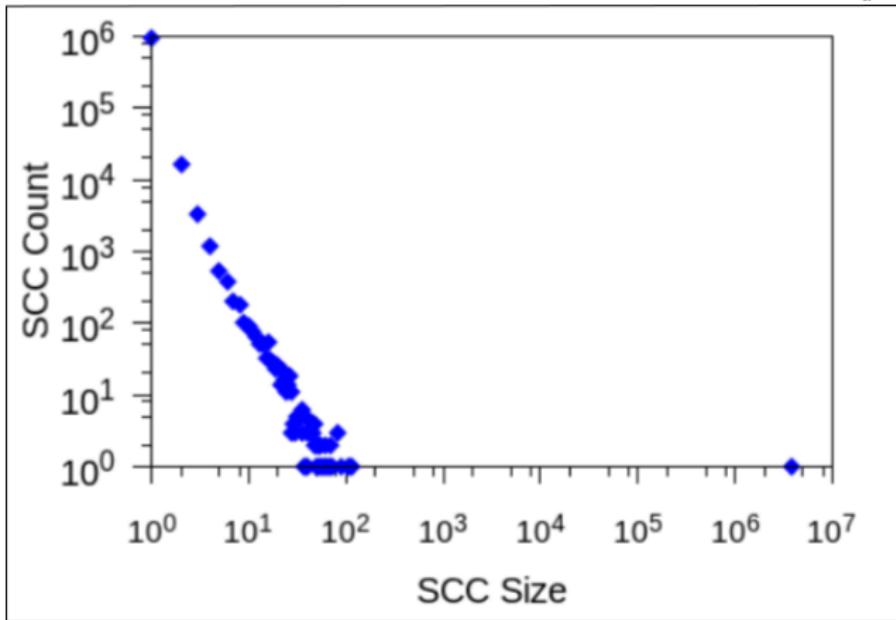


Bild aus Hong et al. [2013], unverändert.

Graphen in der Echten Welt

Eigenschaften:

- Eine große Komponente und viele kleine
- Durchmesser sind extrem klein (Avg: ≤ 6)
 - Facebook: Avg 4.57 im Februar 2016
 - Wikipedia: Avg 3.019

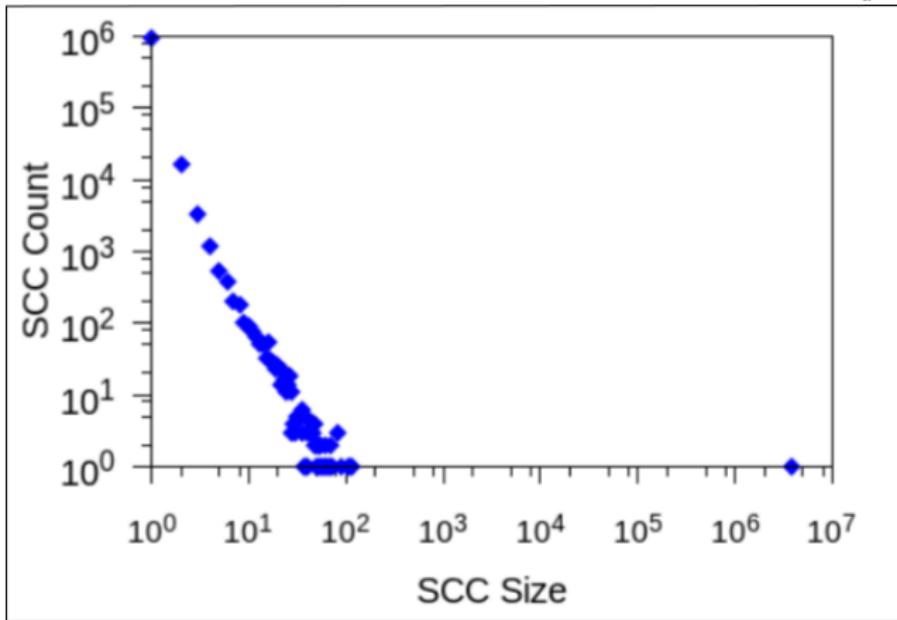


Bild aus Hong et al. [2013], unverändert.

Graphen in der Echten Welt

Eigenschaften:

- Eine große Komponente und viele kleine
- Durchmesser sind extrem klein (Avg: ≤ 6)
 - Facebook: Avg 4.57 im Februar 2016
 - Wikipedia: Avg 3.019
 - Twitter: Avg 3.435

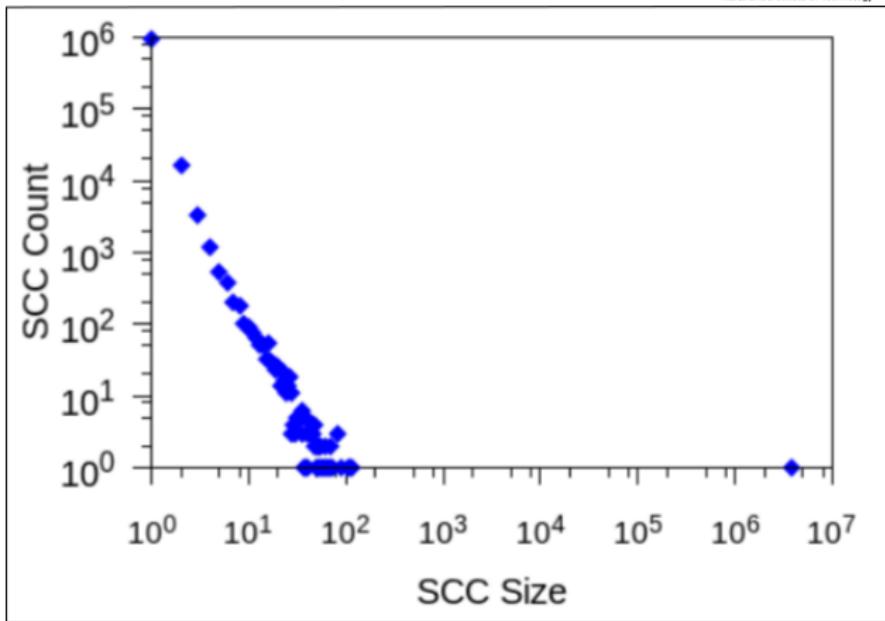


Bild aus Hong et al. [2013], unverändert.

Graphen in der Echten Welt

Eigenschaften:

- Eine große Komponente und viele kleine
- Durchmesser sind extrem klein (Avg: ≤ 6)
 - Facebook: Avg 4.57 im Februar 2016
 - Wikipedia: Avg 3.019
 - Twitter: Avg 3.435
- Großer SCC mit $O(N)$ Knoten

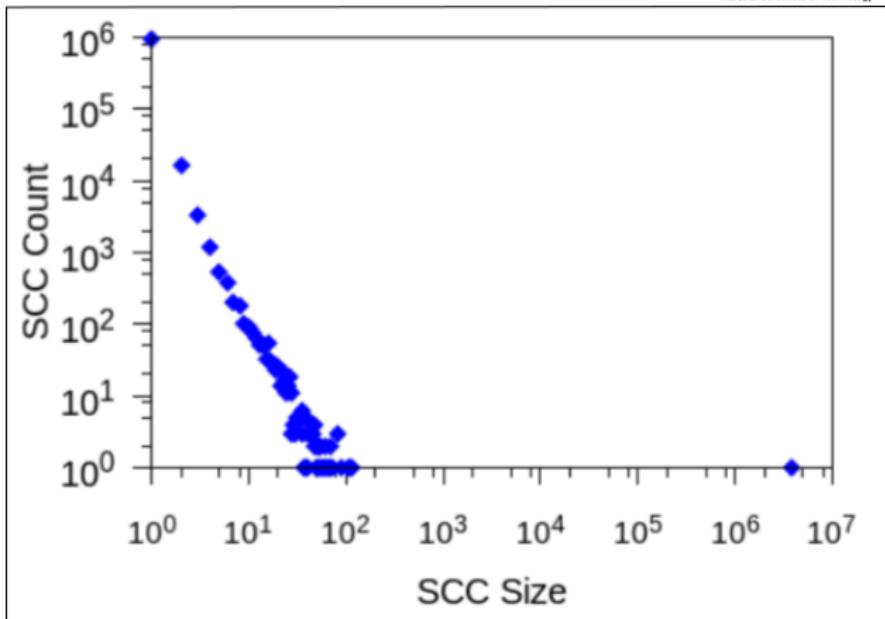


Bild aus Hong et al. [2013], unverändert.

Graphen in der Echten Welt

Eigenschaften:

- Eine große Komponente und viele kleine
- Durchmesser sind extrem klein (Avg: ≤ 6)
 - Facebook: Avg 4.57 im Februar 2016
 - Wikipedia: Avg 3.019
 - Twitter: Avg 3.435
- Großer SCC mit $O(N)$ Knoten
- $O(N)$ viele SCCs der Größe 1

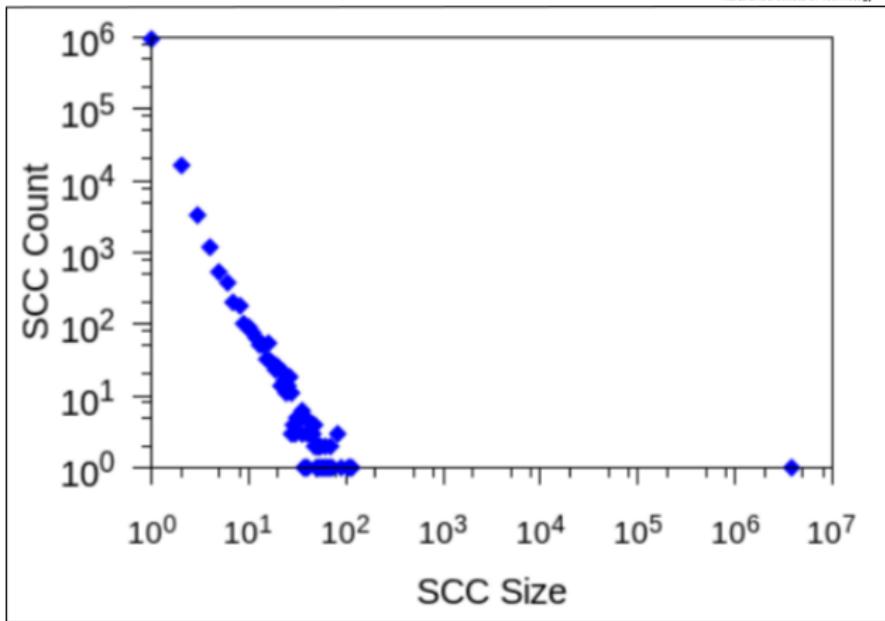


Bild aus Hong et al. [2013], unverändert.

Graphen in der Echten Welt

Eigenschaften:

- Eine große Komponente und viele kleine
- Durchmesser sind extrem klein (Avg: ≤ 6)
 - Facebook: Avg 4.57 im Februar 2016
 - Wikipedia: Avg 3.019
 - Twitter: Avg 3.435
- Großer SCC mit $O(N)$ Knoten
- $O(N)$ viele SCCs der Größe 1
- Viele der kleinen SCCs hängen am Großen

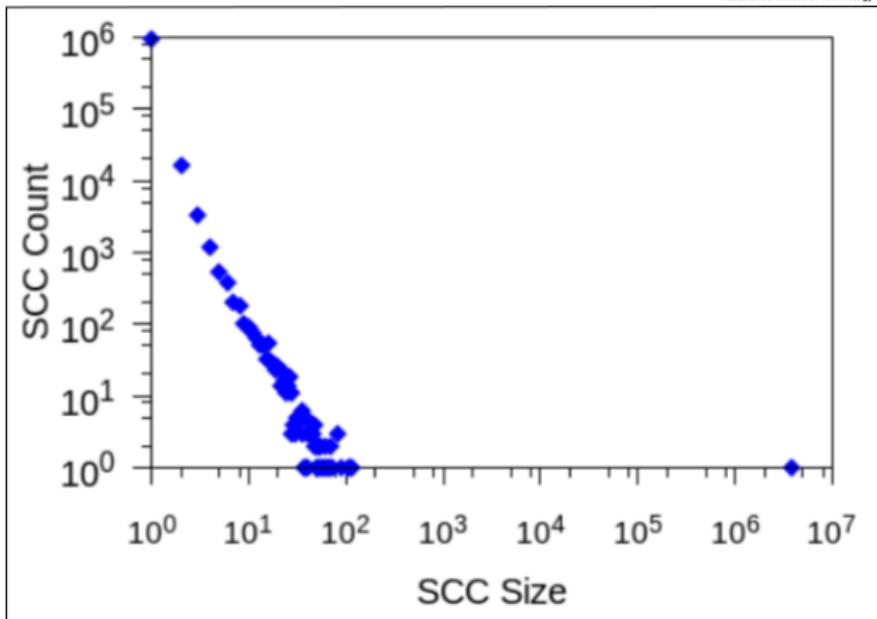
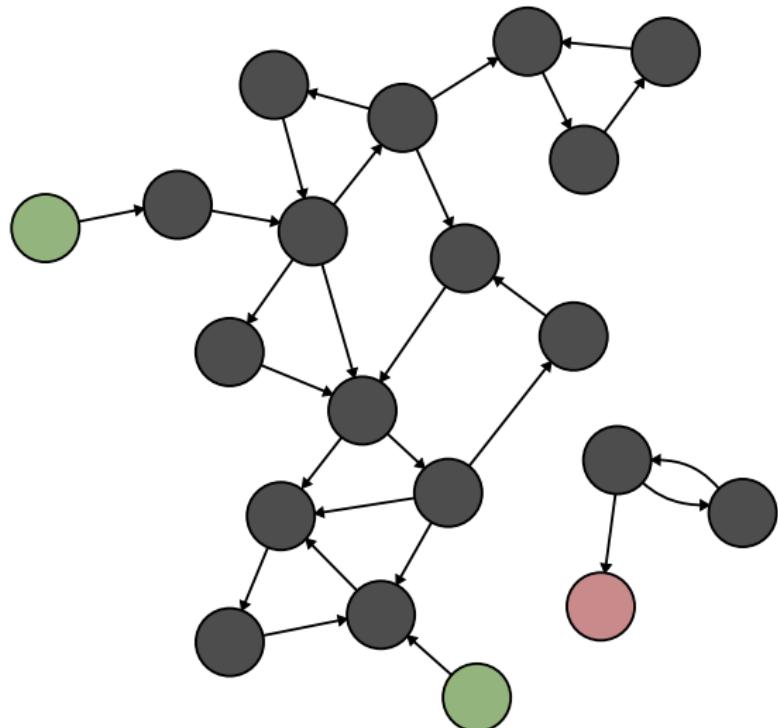


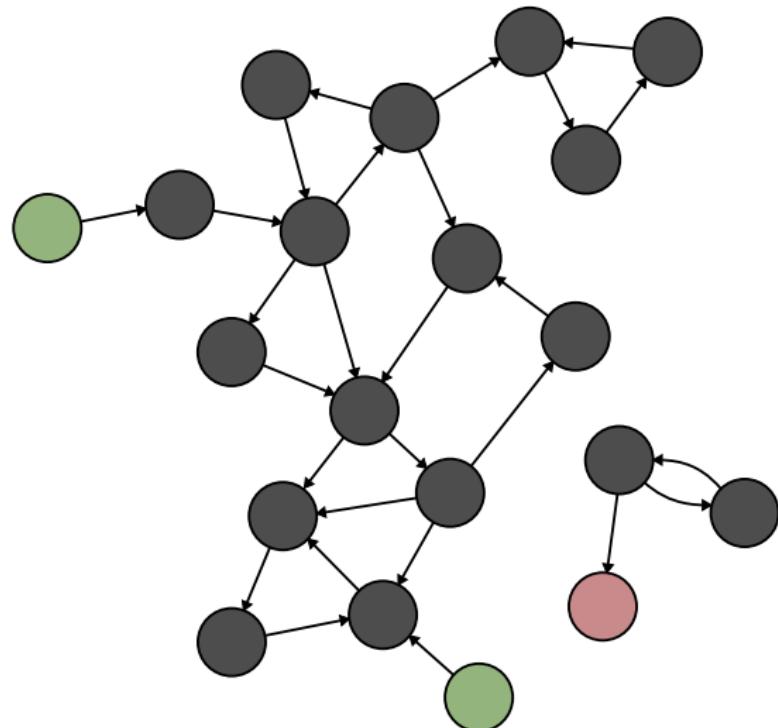
Bild aus Hong et al. [2013], unverändert.

Baseline-Algorithmus



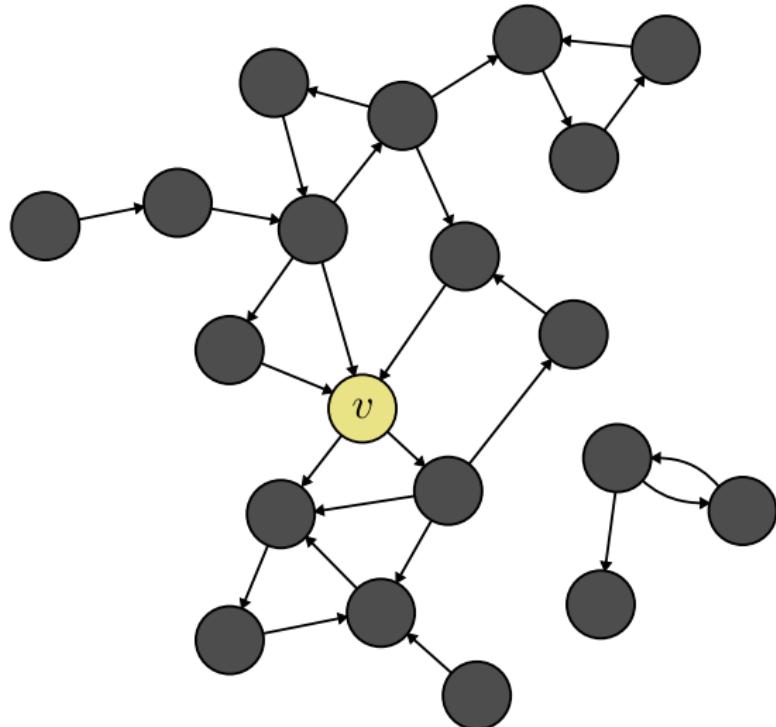
Baseline-Algorithmus

- Paralleles Trim



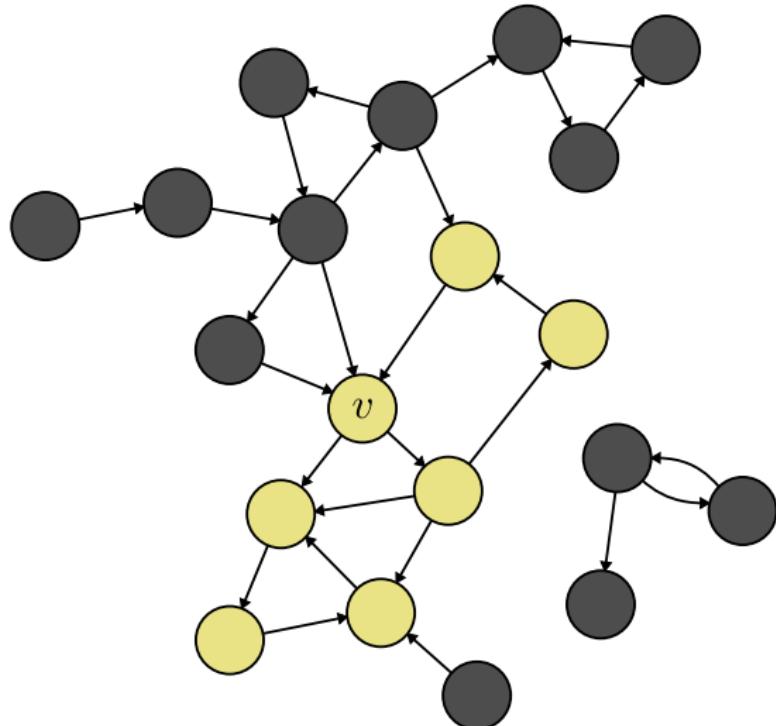
Baseline-Algorithmus

- Paralleles Trim
- 'Klassisches FwBw'



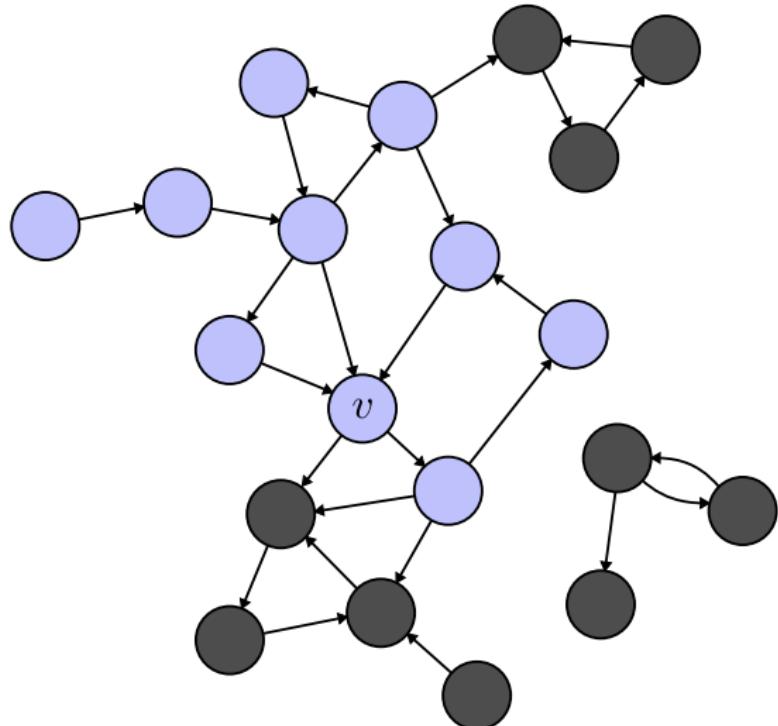
Baseline-Algorithmus

- Paralleles Trim
- 'Klassisches FwBw'
 - Vorwärts (Markieren)



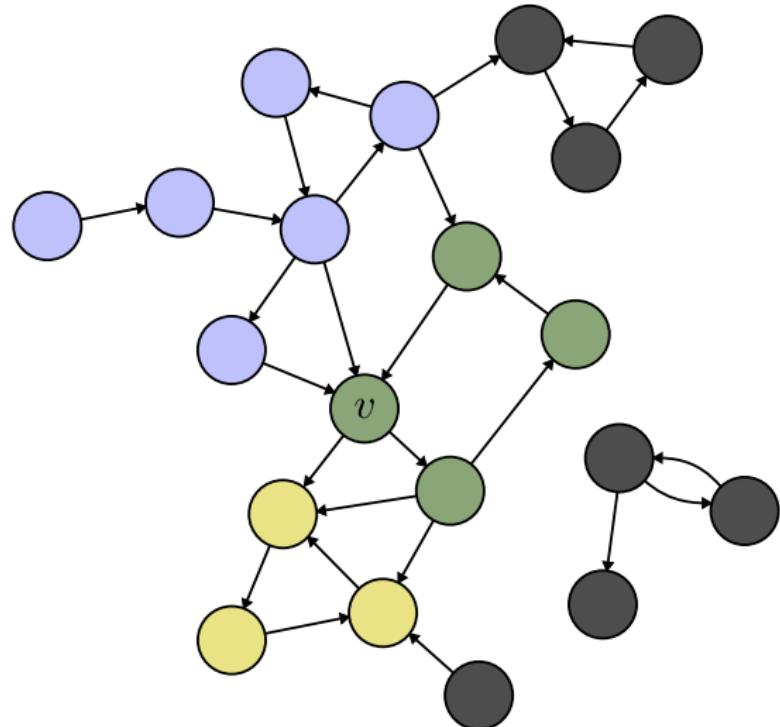
Baseline-Algorithmus

- Paralleles Trim
- 'Klassisches FwBw'
 - Vorwärts (Markieren)
 - Rückwärts (Markieren, Sammeln vom SCC)



Baseline-Algorithmus

- Paralleles Trim
- 'Klassisches FwBw'
 - Vorwärts (Markieren)
 - Rückwärts (Markieren, Sammeln vom SCC)
 - Parallel Rekursion über Teilmengen



Variante 1

Probleme:

Variante 1

Probleme:

- Ungleich verteilte SCC-Größen

Probleme:

- Ungleich verteilte SCC-Größen
- Warten auf das Erkennen des großen SCCs

Probleme:

- Ungleich verteilte SCC-Größen
- Warten auf das Erkennen des großen SCCs

Ansatz:

Probleme:

- Ungleich verteilte SCC-Größen
- Warten auf das Erkennen des großen SCCs

Ansatz:

- Phase 1:

Probleme:

- Ungleich verteilte SCC-Größen
- Warten auf das Erkennen des großen SCCs

Ansatz:

- Phase 1:
- Alle Threads arbeiten auf selben Teilgraphen

Probleme:

- Ungleich verteilte SCC-Größen
- Warten auf das Erkennen des großen SCCs

Ansatz:

- Phase 1:
- Alle Threads arbeiten auf selben Teilgraphen
- Phase 2: (nachdem der große SCC gefunden wurde)

Probleme:

- Ungleich verteilte SCC-Größen
- Warten auf das Erkennen des großen SCCs

Ansatz:

- Phase 1:
 - Alle Threads arbeiten auf selben Teilgraphen
- Phase 2: (nachdem der große SCC gefunden wurde)
- Trim

Probleme:

- Ungleich verteilte SCC-Größen
- Warten auf das Erkennen des großen SCCs

Ansatz:

- Phase 1:
 - Alle Threads arbeiten auf selben Teilgraphen
- Phase 2: (nachdem der große SCC gefunden wurde)
- Trim
- Wechsel auf voriges Modell

Variante 2

Probleme:

Variante 2

Probleme:

- Phase 2 ist immer noch wenig Parallelisiert

SCC	FW	BW	Remain
2	0	0	125432
5	0	0	125427
11	0	0	125416
3	0	0	125413
...			

Probleme:

- Phase 2 ist immer noch wenig Parallelisiert
- Warum eigentlich?

Variante 2

Probleme:

- Phase 2 ist immer noch wenig Parallelisiert
- Warum eigentlich?

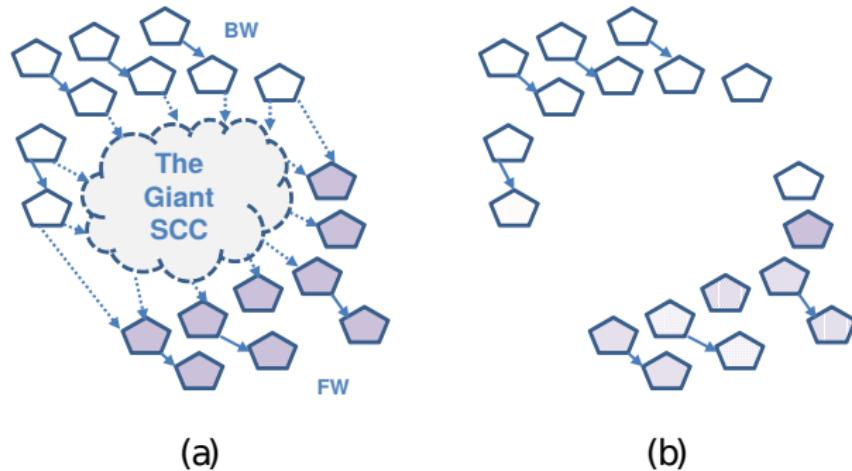


Bild aus Hong et al. [2013], unverändert.

Variante 2

Funktion FwBw-SEARCH($G = (V, E)$)

```
if  $G = \emptyset$  then
    ↳ return
 $v \leftarrow$  any node in  $G$ ;
output  $Pred_G(v) \cap Desc_G(v)$ ;
FwBw-SEARCH( $Pred_G(v) \setminus Desc_G(v)$ );
FwBw-SEARCH( $Desc_G(v) \setminus Pred_G(v)$ );
FwBw-SEARCH( $Rem_G(v)$ );
```

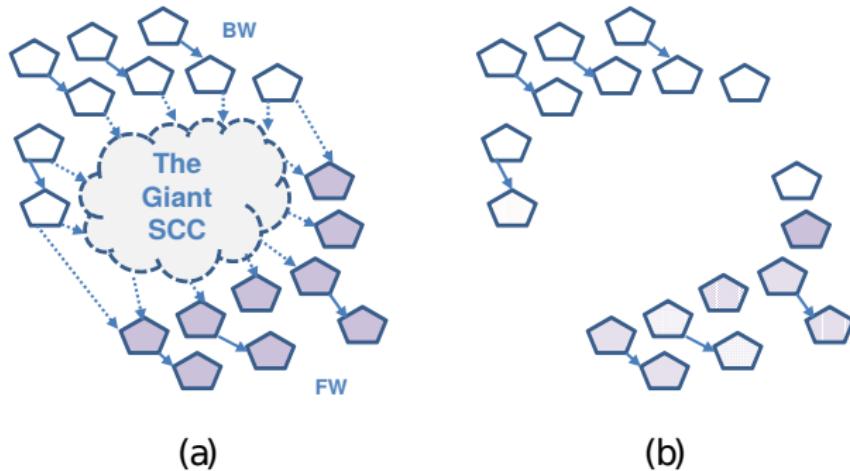


Bild aus Hong et al. [2013], unverändert.

Variante 2

Probleme:

- Phase 2 ist immer noch wenig Parallelisiert
- Warum eigentlich?

Ansatz:

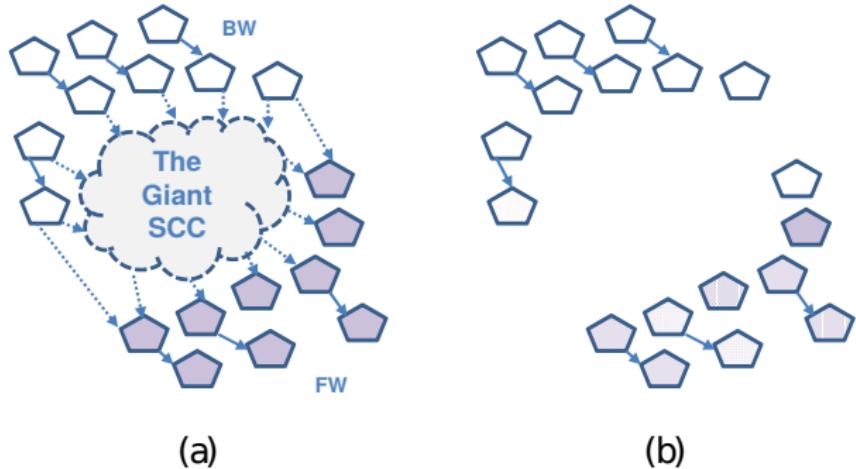


Bild aus Hong et al. [2013], unverändert.

Variante 2

Probleme:

- Phase 2 ist immer noch wenig Parallelisiert
- Warum eigentlich?

Ansatz:

- Parallel WCCs erkennen und andere 'Farbe' geben

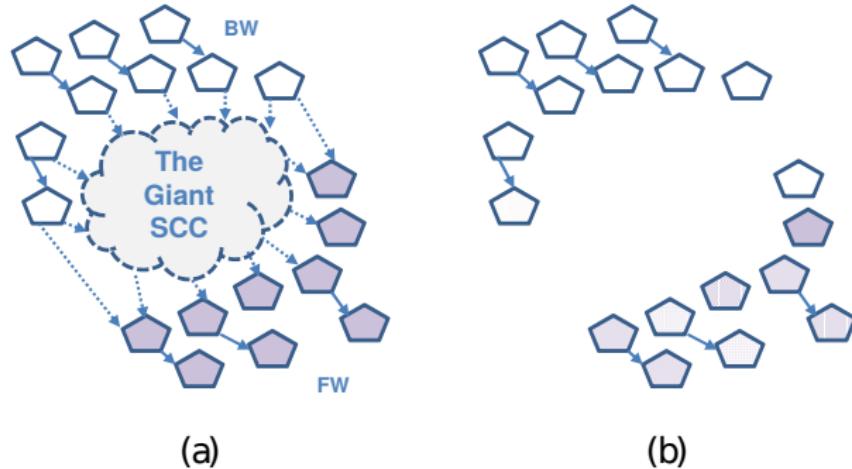


Bild aus Hong et al. [2013], unverändert.

Variante 2, detailliert

Funktion `METHODE2(G, Color)`

// Initialisierung

$\forall n \in G : \text{Color}(n) \leftarrow 0, \text{mark}(n) \leftarrow \text{false}$

// Phase 1: Daten-Parallelisierung

`PAR-TRIM(G, SCC, Color, mark)`

`PAR-FwBw(G, 0, SCC, Color, mark)`

`PAR-TRIM(G, SCC, Color, mark)`

`PAR-WCC(G, Color, mark)`

// Phase 2: Task-Parallelisierung

while `work_queue not empty` **par do**

`c ← work_queue.pop()`

`RECUR-FwBw(G, c, SCC, Color, mark)`

Variante 2, detailliert

Funktion `METHODE2(G, Color)`

// Initialisierung

$\forall n \in G : \text{Color}(n) \leftarrow 0, \text{mark}(n) \leftarrow \text{false}$

// Phase 1: Daten-Parallelisierung

`PAR-TRIM(G, SCC, Color, mark)`

`PAR-FwBw(G, 0, SCC, Color, mark)`

`PAR-TRIM(G, SCC, Color, mark)`

`PAR-WCC(G, Color, mark)`

Implementation

PAR-FwBw
BFS

RECUR-FwBw
DFS

// Phase 2: Task-Parallelisierung

while `work_queue not empty` **par do**

`c ← work_queue.pop()`

`RECUR-FwBw(G, c, SCC, Color, mark)`

Variante 2, detailliert

Funktion `METHODE2(G, Color)`

// Initialisierung

$\forall n \in G : \text{Color}(n) \leftarrow 0, \text{mark}(n) \leftarrow \text{false}$

// Phase 1: Daten-Parallelisierung

`PAR-TRIM(G, SCC, Color, mark)`

`PAR-FwBw(G, 0, SCC, Color, mark)`

`PAR-TRIM(G, SCC, Color, mark)`

`PAR-WCC(G, Color, mark)`

Noch mehr speedup?

// Phase 2: Task-Parallelisierung

while `work_queue not empty` **par do**

`c ← work_queue.pop()`

`RECUR-FwBw(G, c, SCC, Color, mark)`

Variante 2, detailliert

Funktion `METHODE2(G , $Color$)`

// Initialisierung

$\forall n \in G : Color(n) \leftarrow 0, mark(n) \leftarrow false$

// Phase 1: Daten-Parallelisierung

`PAR-TRIM(G , SCC , $Color$, $mark$)`

`PAR-FwBw(G , 0, SCC , $Color$, $mark$)`

`PAR-TRIM(G , SCC , $Color$, $mark$)`

`PAR-WCC(G , $Color$, $mark$)`

// Phase 2: Task-Parallelisierung

while `work_queue not empty` **par do**

$c \leftarrow work_queue.pop()$

`RECUR-FwBw(G , c , SCC , $Color$, $mark$)`

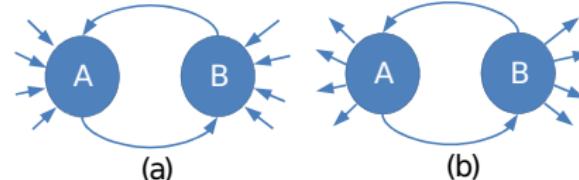


Figure 4: Patterns of size-2 SCCs detected by Trim2. There is a tight cycle between A and B but either (a) there are no other outgoing edges from A and B, or (b) there are no other incoming edges to A and B.

Bild aus Hong et al. [2013], unverändert.

Variante 2, detailliert

Funktion `METHODE2(G , $Color$)`

// Initialisierung

$\forall n \in G : Color(n) \leftarrow 0, mark(n) \leftarrow false$

// Phase 1: Daten-Parallelisierung

`PAR-TRIM(G , SCC , $Color$, $mark$)`

`PAR-FwBw(G , 0, SCC , $Color$, $mark$)`

`PAR-TRIM'(G , SCC , $Color$, $mark$)`

`PAR-WCC(G , $Color$, $mark$)`

// Phase 2: Task-Parallelisierung

while `work_queue` not empty **par do**

$c \leftarrow work_queue.pop()$

`RECUR-FwBw(G , c , SCC , $Color$, $mark$)`

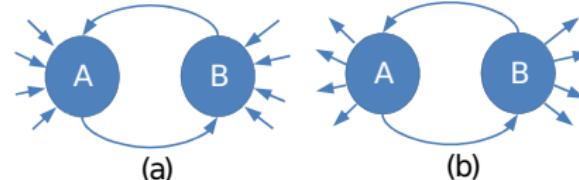


Figure 4: Patterns of size-2 SCCs detected by Trim2. There is a tight cycle between A and B but either (a) there are no other outgoing edges from A and B, or (b) there are no other incoming edges to A and B.

Bild aus Hong et al. [2013], unverändert.

Arbeitsverteilung

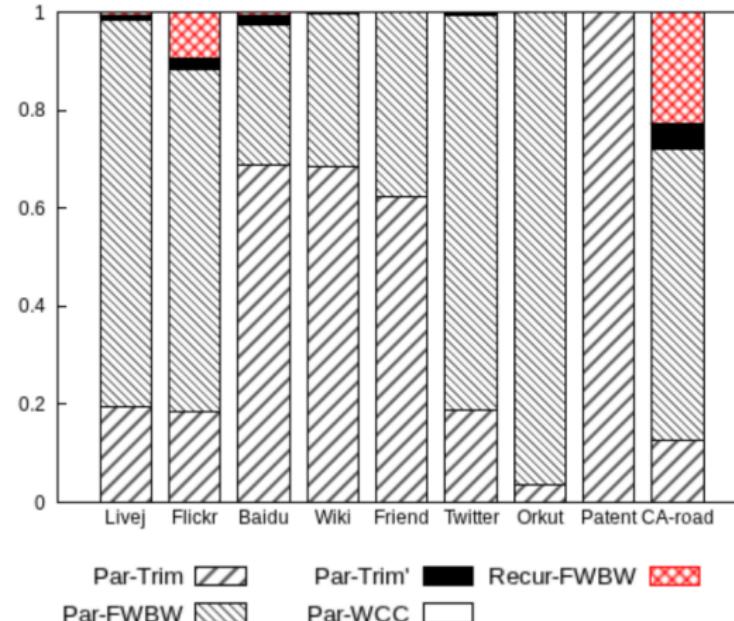
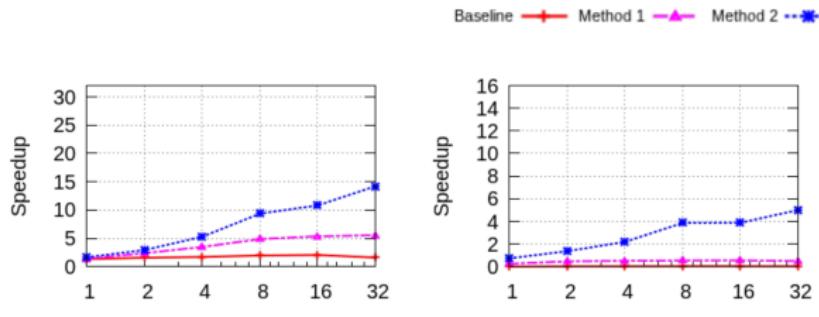


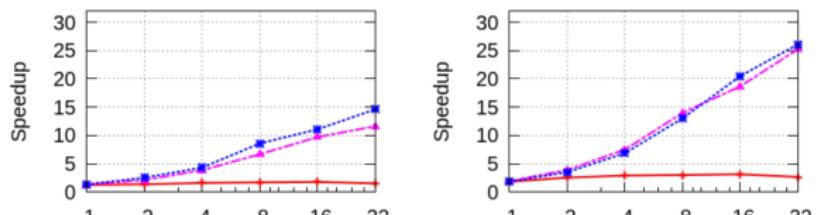
Figure 8: Fraction of nodes whose SCC is identified at each phase of execution for Method 2.

Bild aus Hong et al. [2013], unverändert.

Geschwindigkeitsverbesserungen

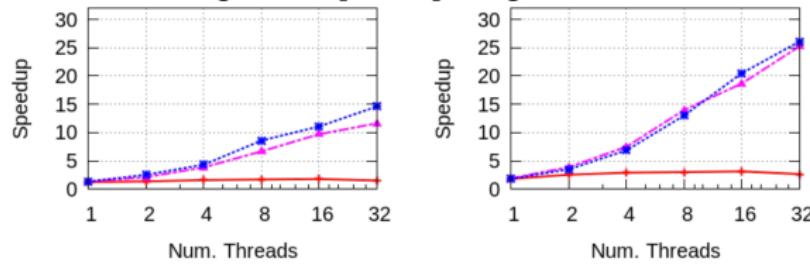


(a) Ljyei

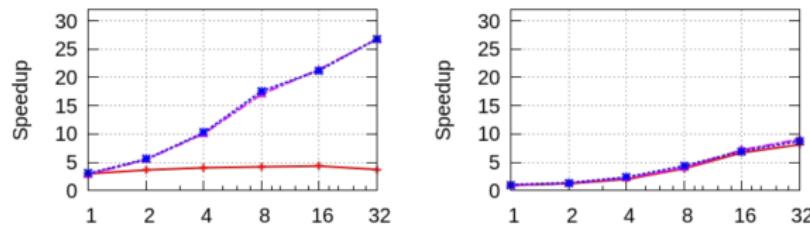


(d) Wiki

Bild aus Hong et al. [2013], zugeschnitten.



(d) Wiki

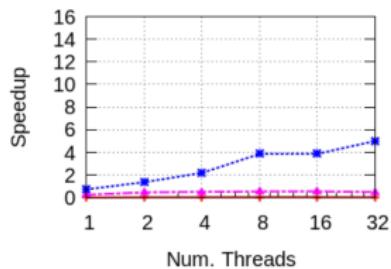


(e) Orkut

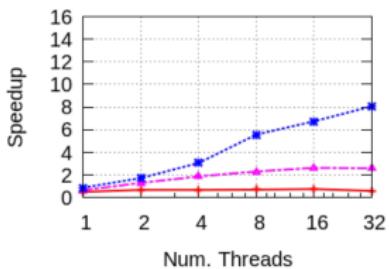
(h) Patent

Geschwindigkeitsverbesserungen

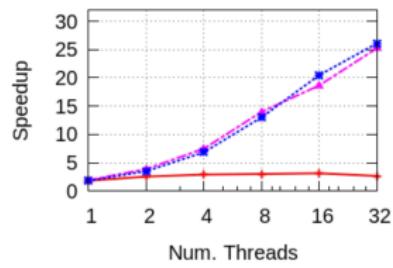
Baseline —+— Method 1 —▲— Method 2 ...*



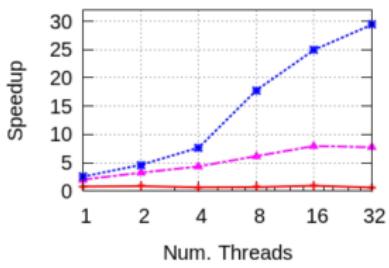
(b) Flickr



(c) Baidu

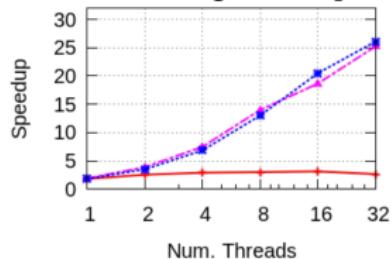


(e) Friend

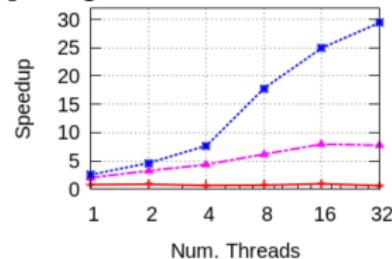


(f) Twitter

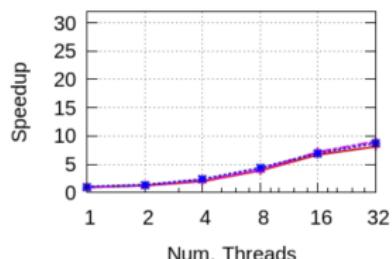
Bild aus Hong et al. [2013], zugeschnitten.



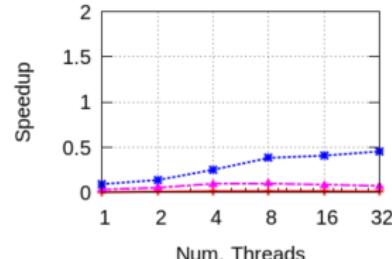
(e) Friend



(f) Twitter



(h) Patent



(i) CA-road

Jiri Barnat, Petr Bauch, Lubos Brim, and Milan Ceška. Computing strongly connected components in parallel on cuda. In *2011 IEEE International Parallel & Distributed Processing Symposium*, pages 544–555. IEEE, 2011.

Mohamad Al Hajj Hassan and Mostafa Bamha. Handling limits of high degree vertices in graph processing using mapreduce and pregel. 2017.

Sungpack Hong, Nicole C Rodia, and Kunle Olukotun. On fast parallel detection of strongly connected components (scc) in small-world graphs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 92. ACM, 2013.

Rohit Kumar, Alberto Abelló, and Toon Calders. Cost model for pregel on graphx. In *European Conference on Advances in Databases and Information Systems*, pages 153–166. Springer, 2017.

William McLendon III, Bruce Hendrickson, Steven J Plimpton, and Lawrence Rauchwerger. Finding strongly connected components in distributed graphs. *Journal of Parallel and Distributed Computing*, 65(8):901–910, 2005.

Tosaka. Cuda processing flow.

[https://en.wikipedia.org/wiki/File:CUDA_processing_flow_\(En\).PNG](https://en.wikipedia.org/wiki/File:CUDA_processing_flow_(En).PNG), 2008.

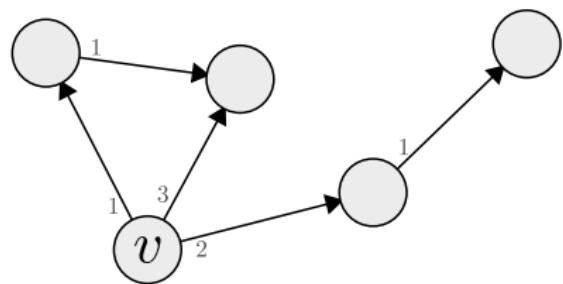
Aufgerufen am 2020-01-13, Lizenziert unter CCO 3.0.

DFS-ORDER

„Bei einer DFS wird auf einem gerichteten Graphen G angefangen bei Knoten v der Knoten x vor dem Knoten y besucht“

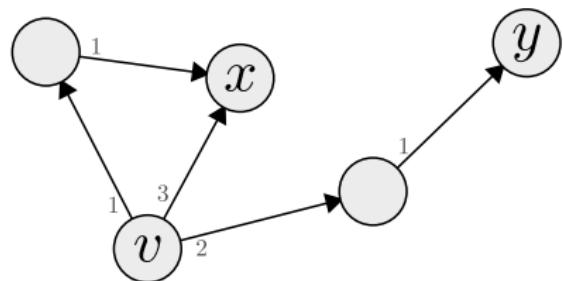
DFS-ORDER

„Bei einer DFS wird auf einem gerichteten Graphen G angefangen bei Knoten v der Knoten x vor dem Knoten y besucht“



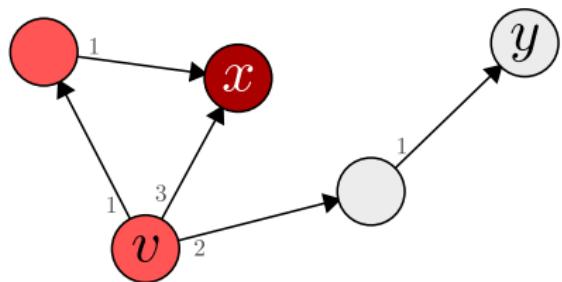
DFS-ORDER

„Bei einer DFS wird auf einem gerichteten Graphen G angefangen bei Knoten v der Knoten x vor dem Knoten y besucht“



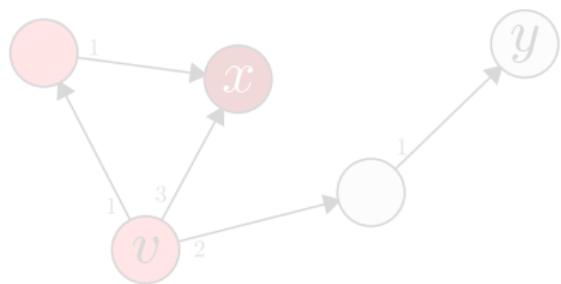
DFS-ORDER

„Bei einer DFS wird auf einem gerichteten Graphen G angefangen bei Knoten v der Knoten x vor dem Knoten y besucht“



DFS-ORDER

„Bei einer DFS wird auf einem gerichteten Graphen G angefangen bei Knoten v der Knoten x vor dem Knoten y besucht“



Theorem

DFS-ORDER ist P-vollständig

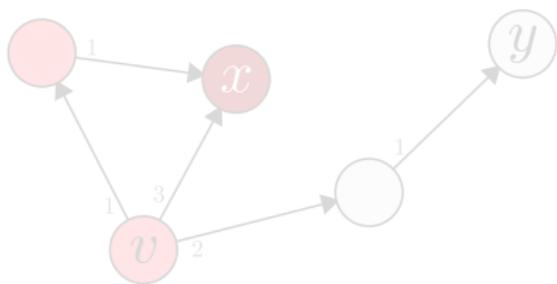
DFS-ORDER

„Bei einer DFS wird auf einem gerichteten Graphen G angefangen bei Knoten v der Knoten x vor dem Knoten y besucht“

log-space-Reduzierbarkeit $L \leq_{\text{log}} L'$

$f : L' \rightarrow L$ mit $\omega \in \Sigma^* \Leftrightarrow f(\omega) \in L$

f ist von einer Turing-Maschine berechenbar, die zusätzlich zur unveränderlichen Eingabe nur *log*-viel Speicher benötigt



Theorem

DFS-ORDER ist P-vollständig

DFS-ORDER

„Bei einer DFS wird auf einem gerichteten Graphen G angefangen bei Knoten v der Knoten x vor dem Knoten y besucht“

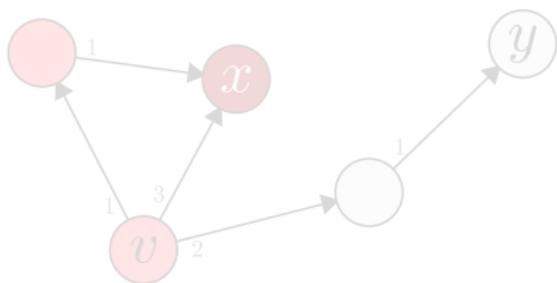
log-space-Reduzierbarkeit $L \leq_{\log} L'$

$f : L' \rightarrow L$ mit $\omega \in \Sigma^* \Leftrightarrow f(\omega) \in L$

f ist von einer Turing-Maschine berechenbar, die zusätzlich zur unveränderlichen Eingabe nur log-viel Speicher benötigt

Theorem

Das Auswertungsproblem bei booleschen Schaltkreisen ist P-vollständig.

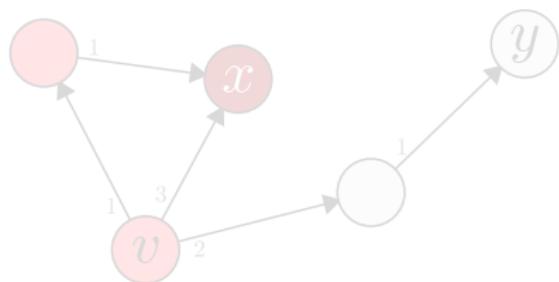


Theorem

DFS-ORDER ist P-vollständig

DFS-ORDER

„Bei einer DFS wird auf einem gerichteten Graphen G angefangen bei Knoten v der Knoten x vor dem Knoten y besucht“



Theorem

DFS-ORDER ist P-vollständig

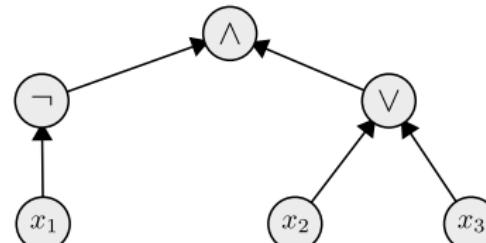
log-space-Reduzierbarkeit $L \leq_{\log} L'$

$f : L' \rightarrow L$ mit $\omega \in \Sigma^* \Leftrightarrow f(\omega) \in L$

f ist von einer Turing-Maschine berechenbar, die zusätzlich zur unveränderlichen Eingabe nur log-viel Speicher benötigt

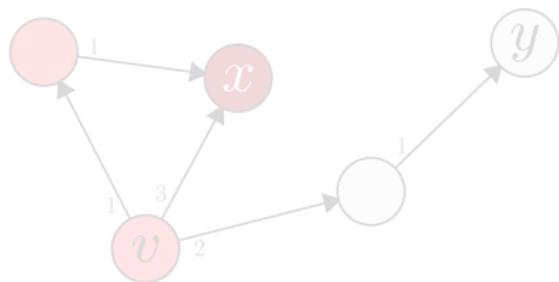
Theorem

Das Auswertungsproblem bei booleschen Schaltkreisen ist P-vollständig.



DFS-ORDER

„Bei einer DFS wird auf einem gerichteten Graphen G angefangen bei Knoten v der Knoten x vor dem Knoten y besucht“



DFS-ORDER ist P-vollständig

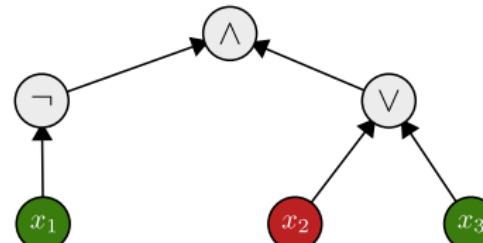
log-space-Reduzierbarkeit $L \leq_{\log} L'$

$f : L' \rightarrow L$ mit $\omega \in \Sigma^* \Leftrightarrow f(\omega) \in L$

f ist von einer Turing-Maschine berechenbar, die zusätzlich zur unveränderlichen Eingabe nur log-viel Speicher benötigt

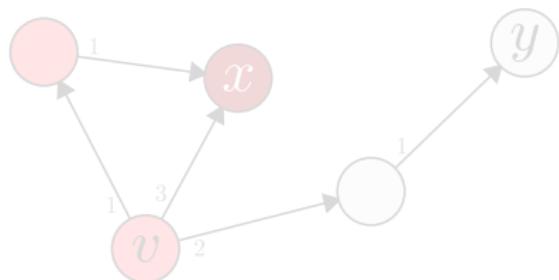
Theorem

Das Auswertungsproblem bei booleschen Schaltkreisen ist P-vollständig.



DFS-ORDER

„Bei einer DFS wird auf einem gerichteten Graphen G angefangen bei Knoten v der Knoten x vor dem Knoten y besucht“



DFS-ORDER ist P-vollständig

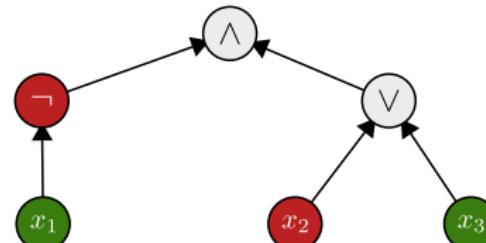
log-space-Reduzierbarkeit $L \leq_{\log} L'$

$f : L' \rightarrow L$ mit $\omega \in \Sigma^* \Leftrightarrow f(\omega) \in L$

f ist von einer Turing-Maschine berechenbar, die zusätzlich zur unveränderlichen Eingabe nur log-viel Speicher benötigt

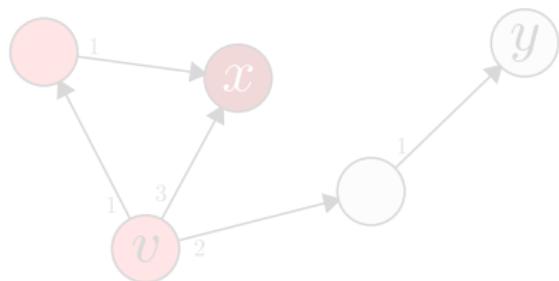
Theorem

Das Auswertungsproblem bei booleschen Schaltkreisen ist P-vollständig.



DFS-ORDER

„Bei einer DFS wird auf einem gerichteten Graphen G angefangen bei Knoten v der Knoten x vor dem Knoten y besucht“



DFS-ORDER ist P-vollständig

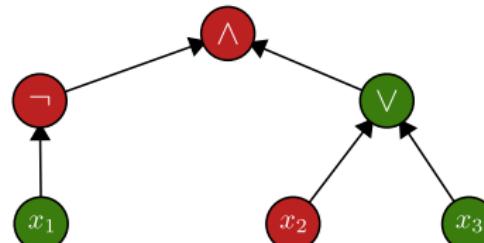
log-space-Reduzierbarkeit $L \leq_{\log} L'$

$f : L' \rightarrow L$ mit $\omega \in \Sigma^* \Leftrightarrow f(\omega) \in L$

f ist von einer Turing-Maschine berechenbar, die zusätzlich zur unveränderlichen Eingabe nur log-viel Speicher benötigt

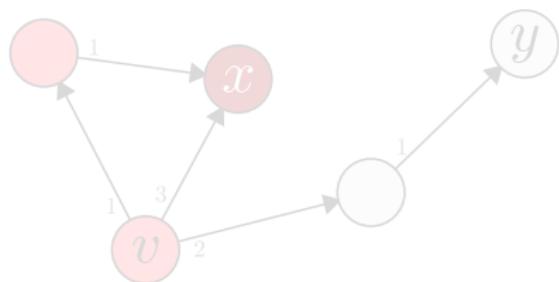
Theorem

Das Auswertungsproblem bei booleschen Schaltkreisen ist P-vollständig.



DFS-ORDER

„Bei einer DFS wird auf einem gerichteten Graphen G angefangen bei Knoten v der Knoten x vor dem Knoten y besucht“



Theorem

DFS-ORDER ist P-vollständig

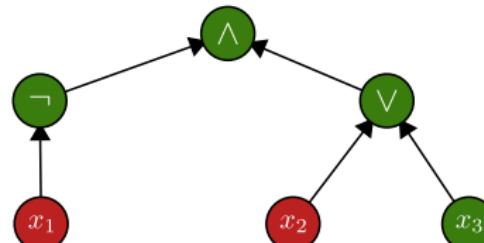
log-space-Reduzierbarkeit $L \leq_{\log} L'$

$f : L' \rightarrow L$ mit $\omega \in \Sigma^* \Leftrightarrow f(\omega) \in L$

f ist von einer Turing-Maschine berechenbar, die zusätzlich zur unveränderlichen Eingabe nur log-viel Speicher benötigt

Theorem

Das Auswertungsproblem bei booleschen Schaltkreisen ist P-vollständig.



Kleine-Welt-Graphen Implementierung

- Viele Fettnäpfchen
- in C++ mit OpenMP
- auch CSR-Repräsentation
- Mark ist $O(N)$ bool Array
- Color ist $O(N)$ int Array
- Für Phase2: lookup-set für alle Farben \mapsto 10x Verbesserung
- Verbindungsgrade sind power-law verteilt \mapsto Dynamische Workload-Verteilung für neighborhood exploration tasks
- Global / Lokale Task queue, ziehen wenn man selbst keine mehr hat, drauf schieben bei zu vielen
- Lokale Größe für Phase1: 1, Phase2: 8 (8 werden auf global geschoben bei Größe 16)