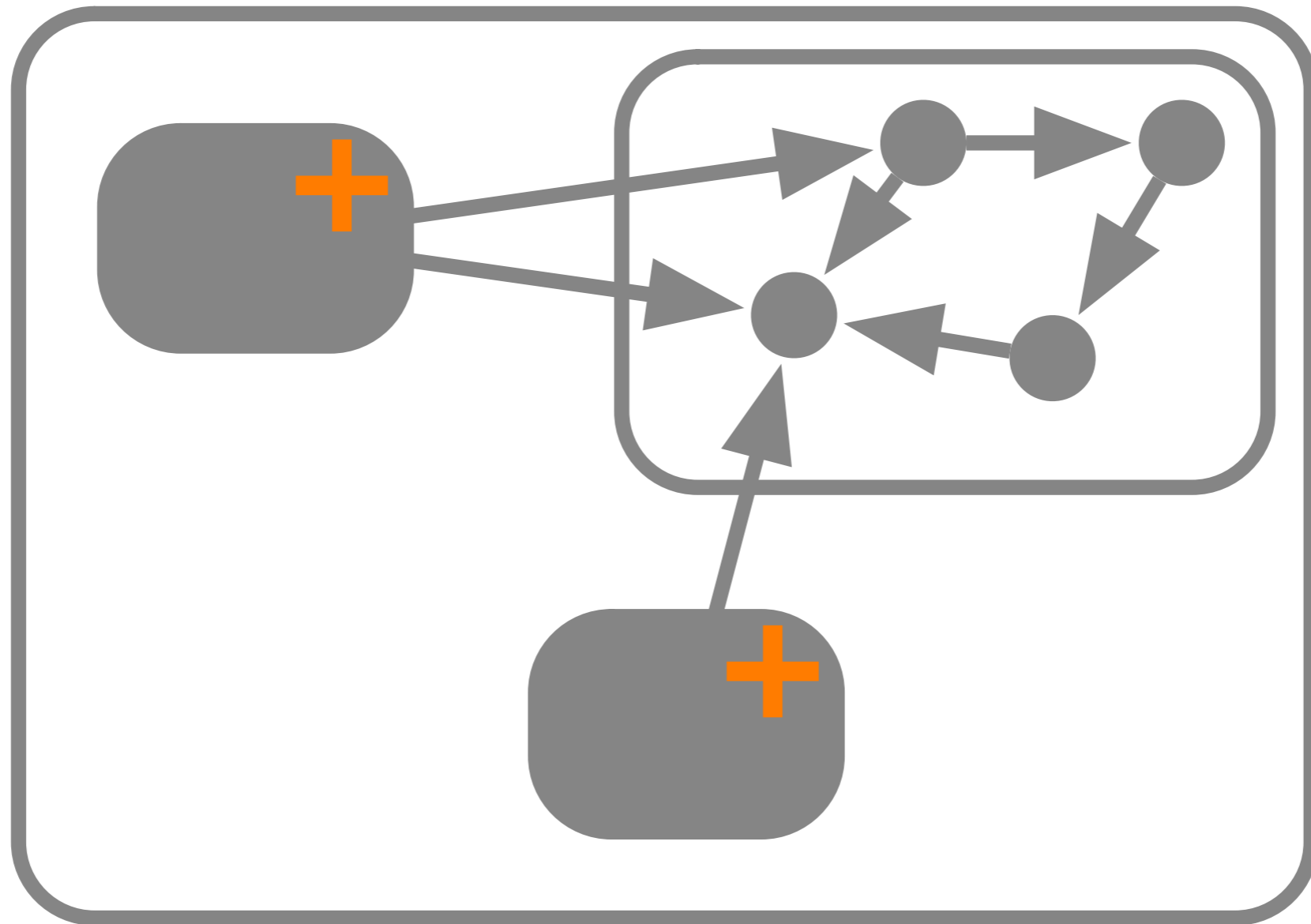


Effiziente visuelle Navigation in geschachtelten Graphen

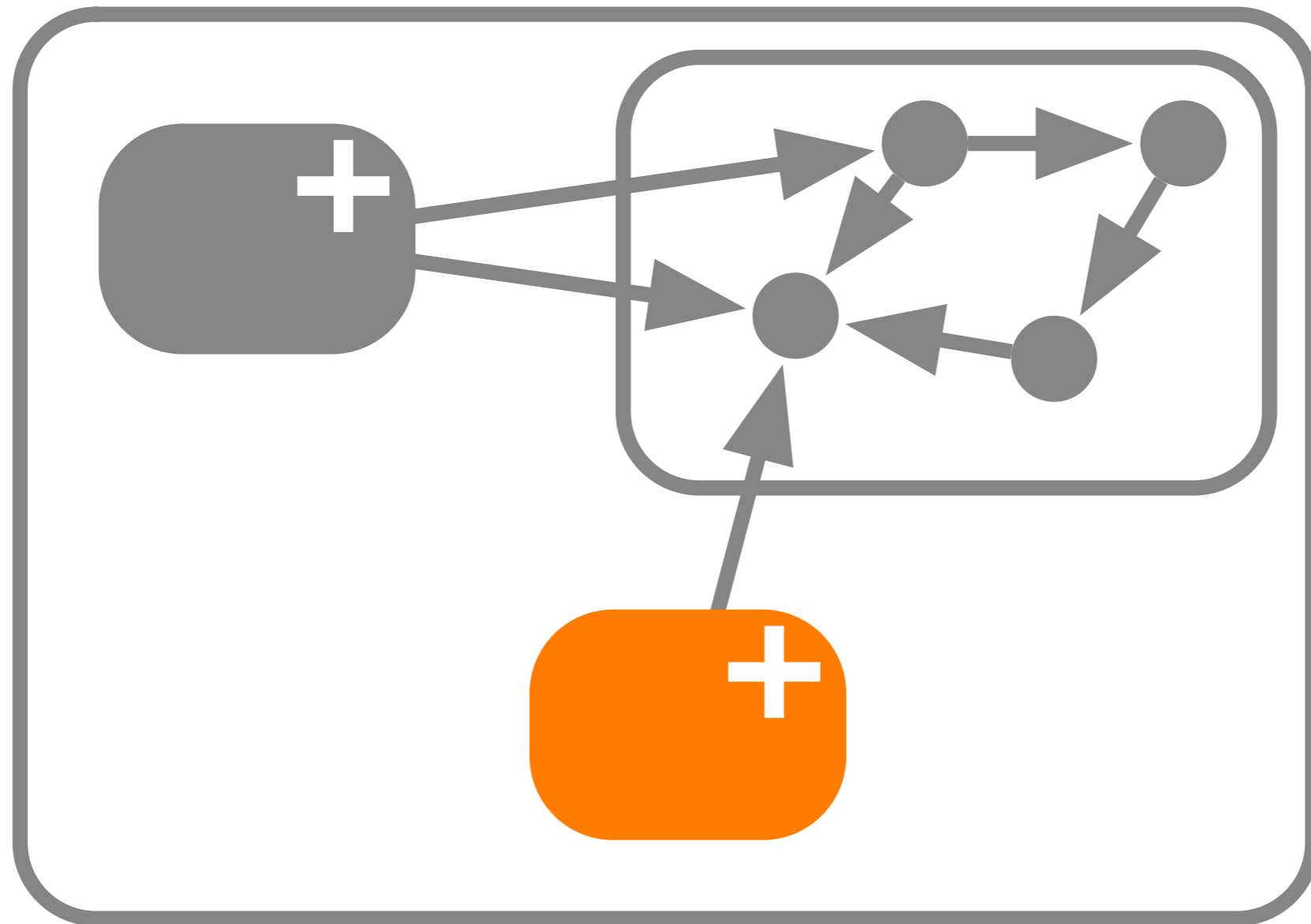
Marcus Raitner

Motivation

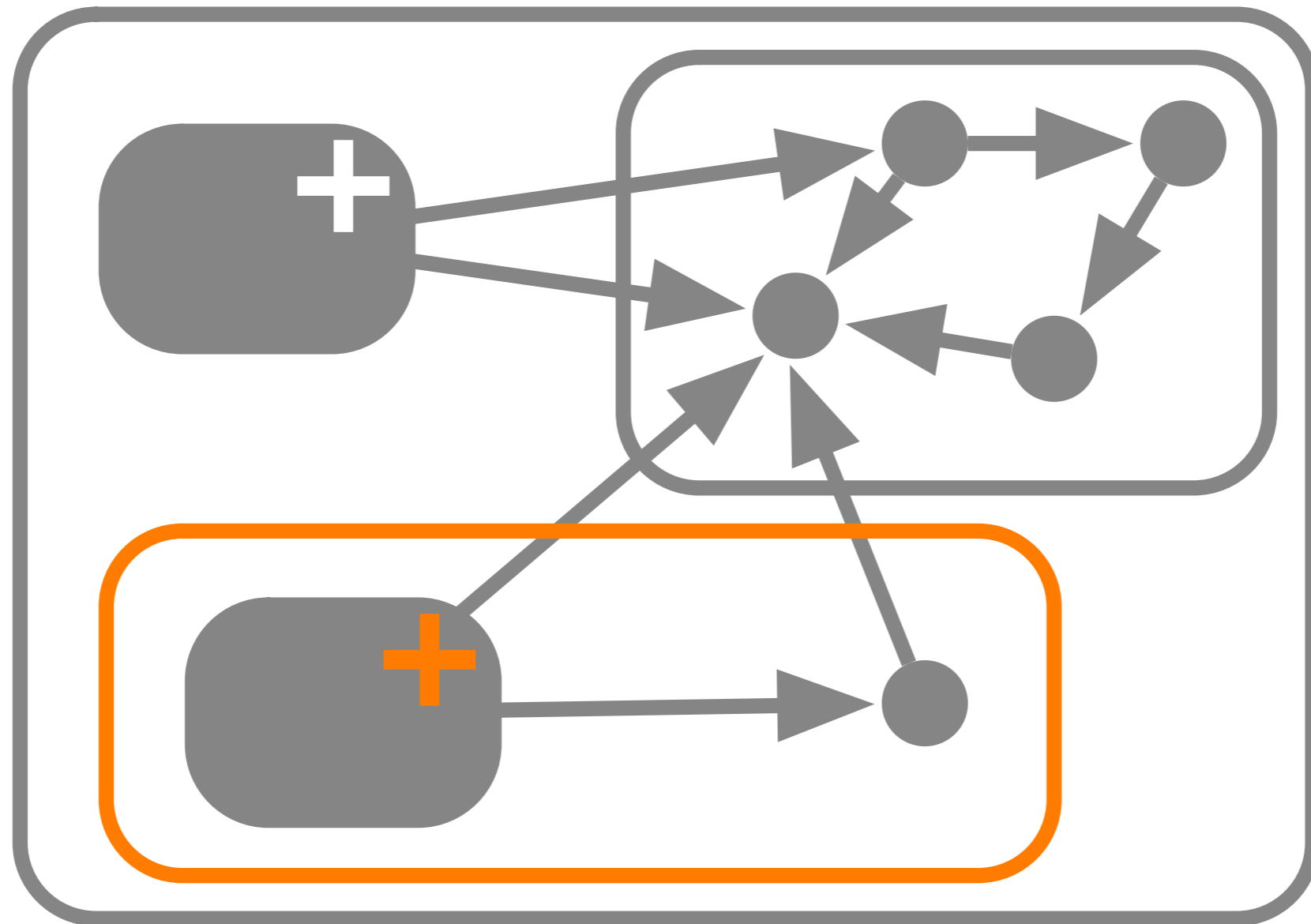
Motivation



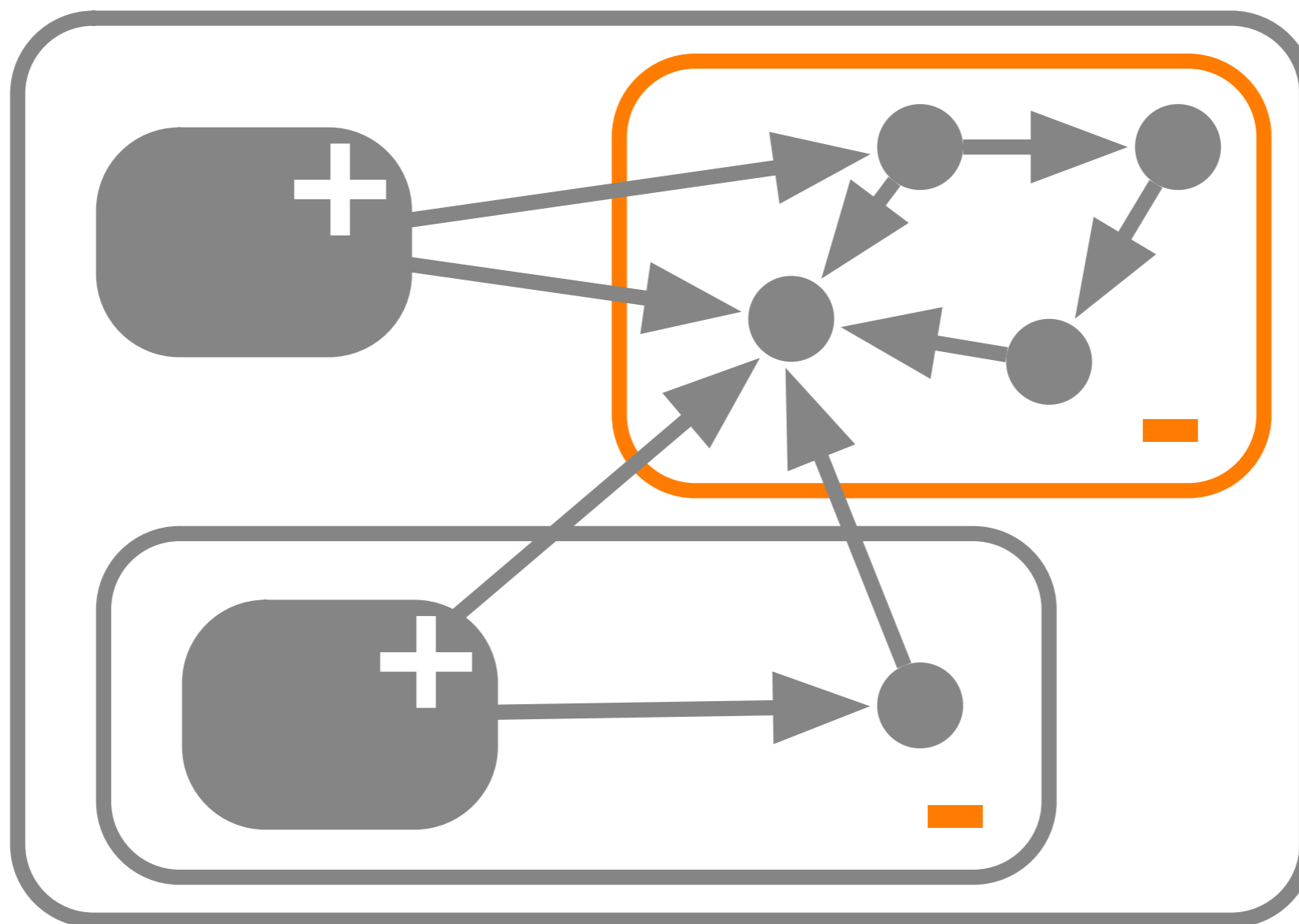
Motivation



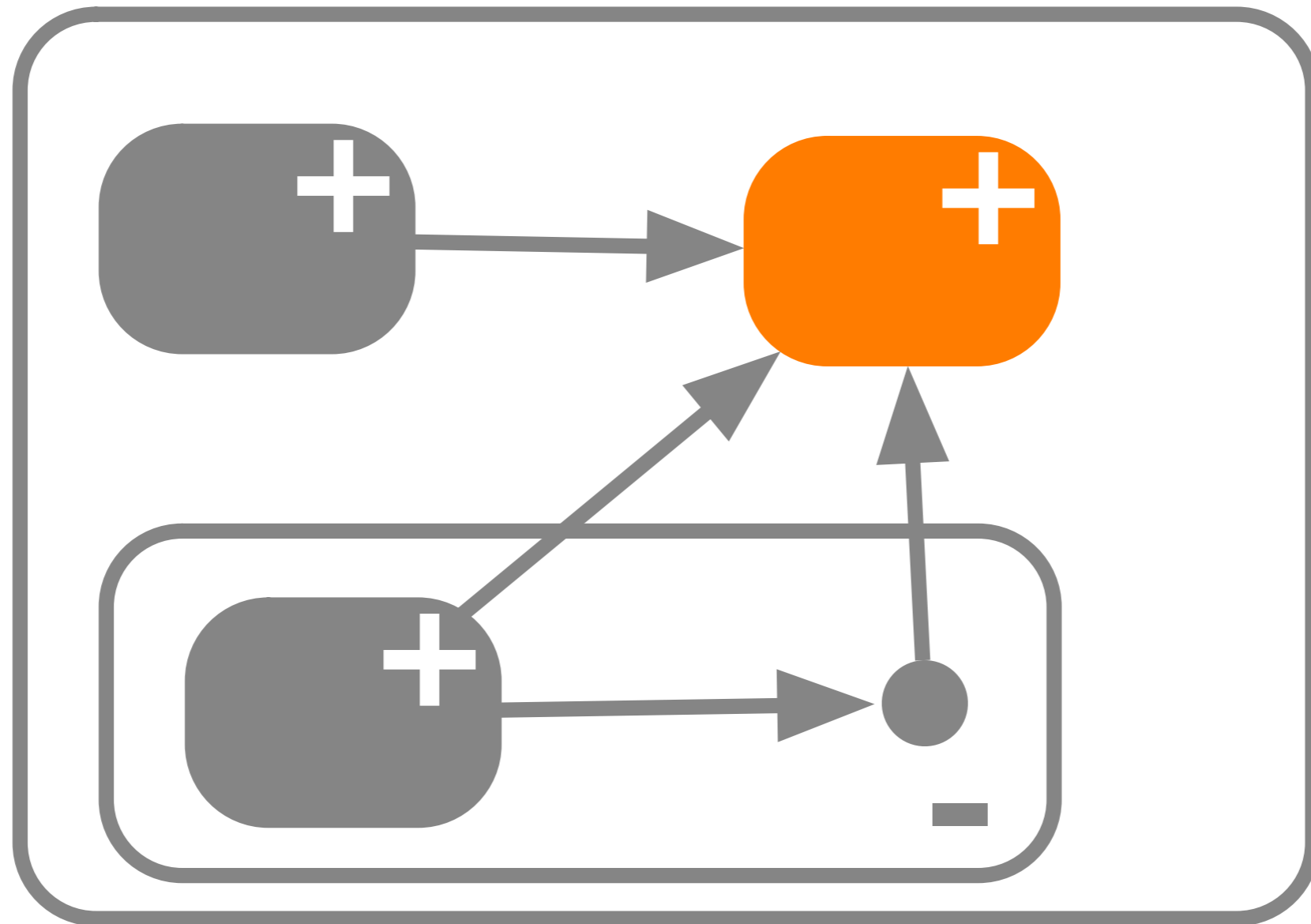
Motivation



Motivation



Motivation



Anwendungen

Biochemische Reaktionsnetze

 BioPath [Schreiber 01]

Software (Re-)Engineering

 Call-Graphen

 Klassendiagramme

Visualisierung großer Netzwerke

 Telefonverkehr

 WWW

Definition

Graph View Maintenance

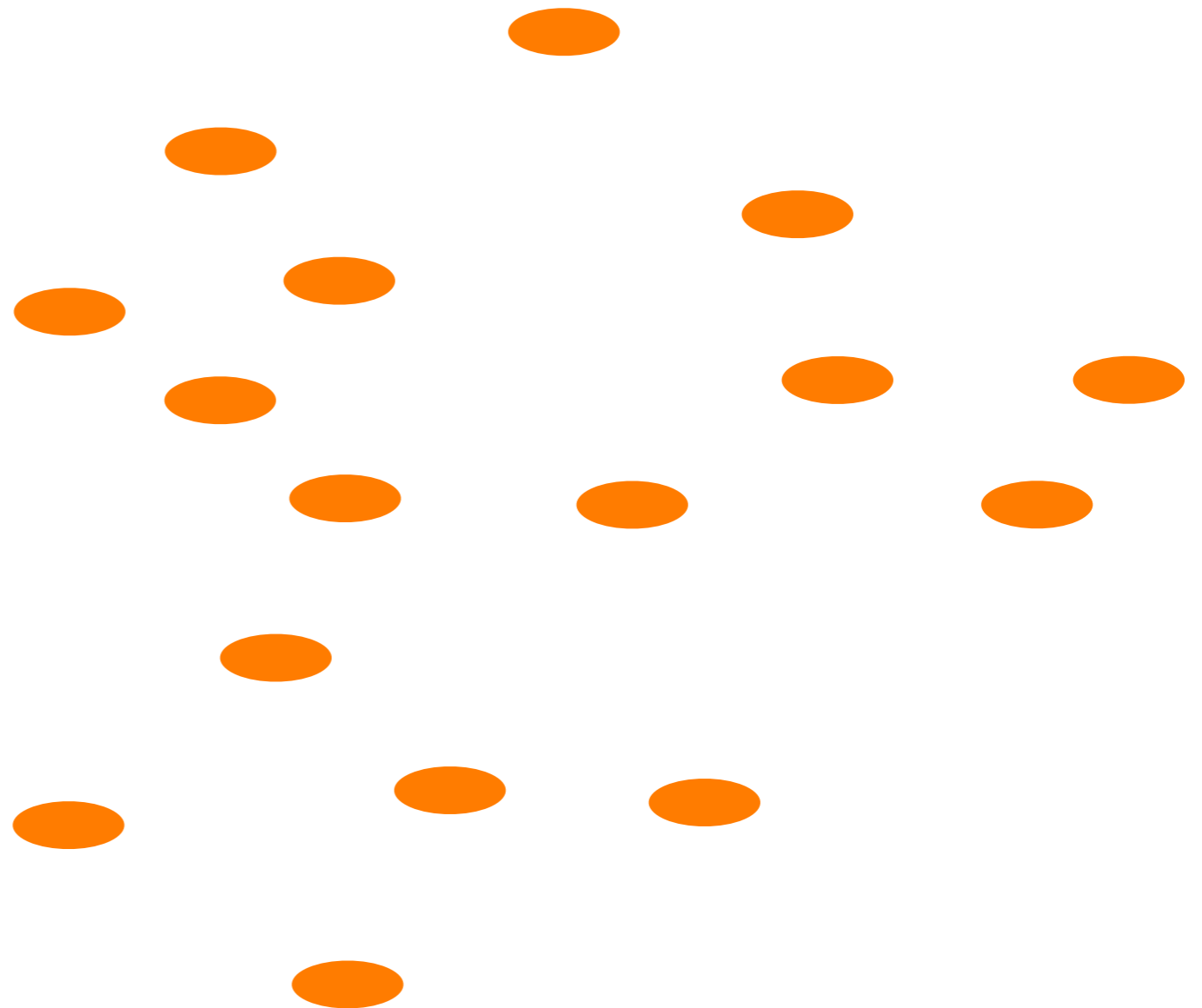
Definition Compound Graph

$D = (V, E, F)$

Inklusionsbaum $T = (V, E)$

Adjazenzkanten F

Inklusionsdiagramm



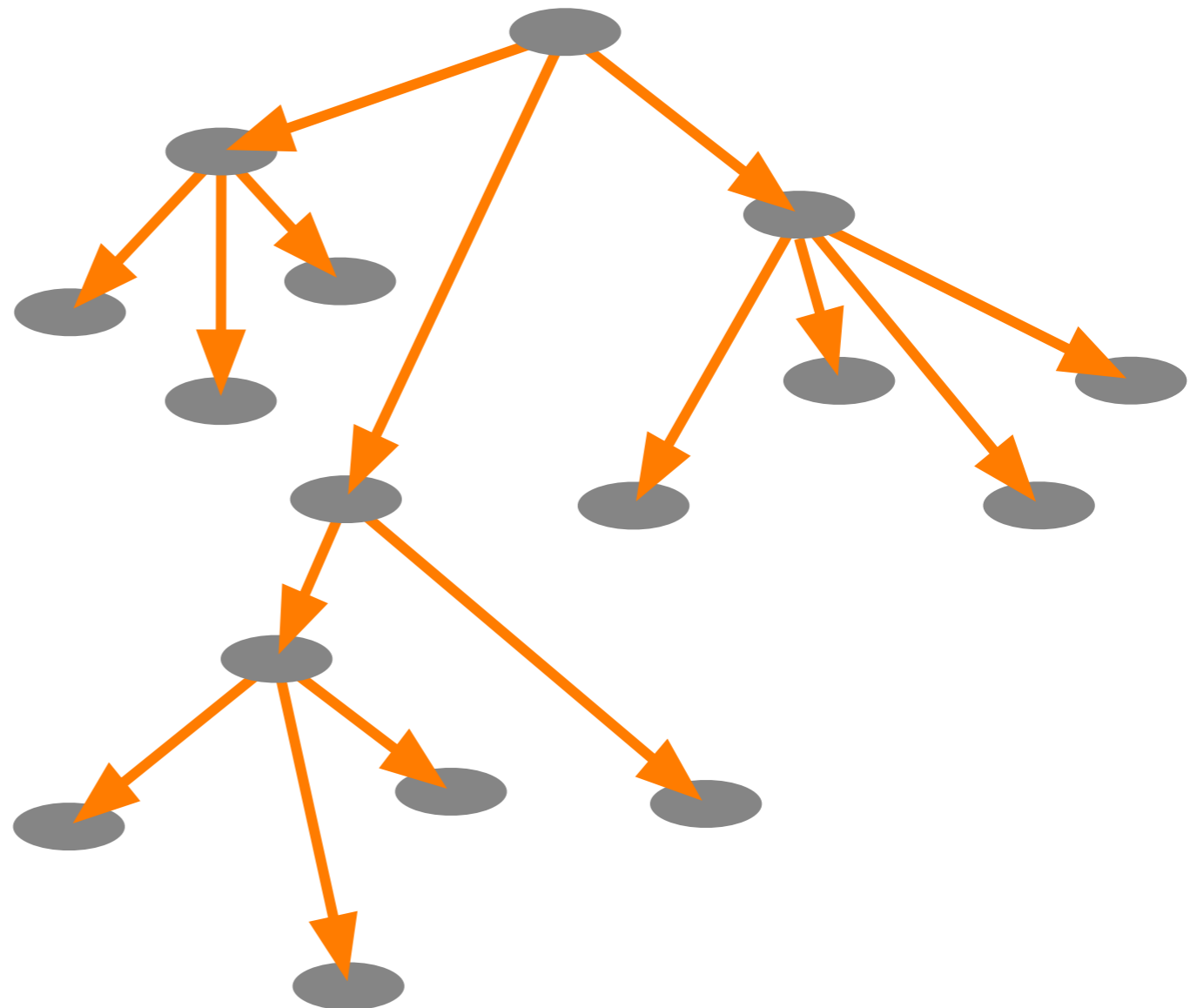
Definition Compound Graph

$D = (V, E, F)$

Inklusionsbaum $T = (V, E)$

Adjazenzkanten F

Inklusionsdiagramm



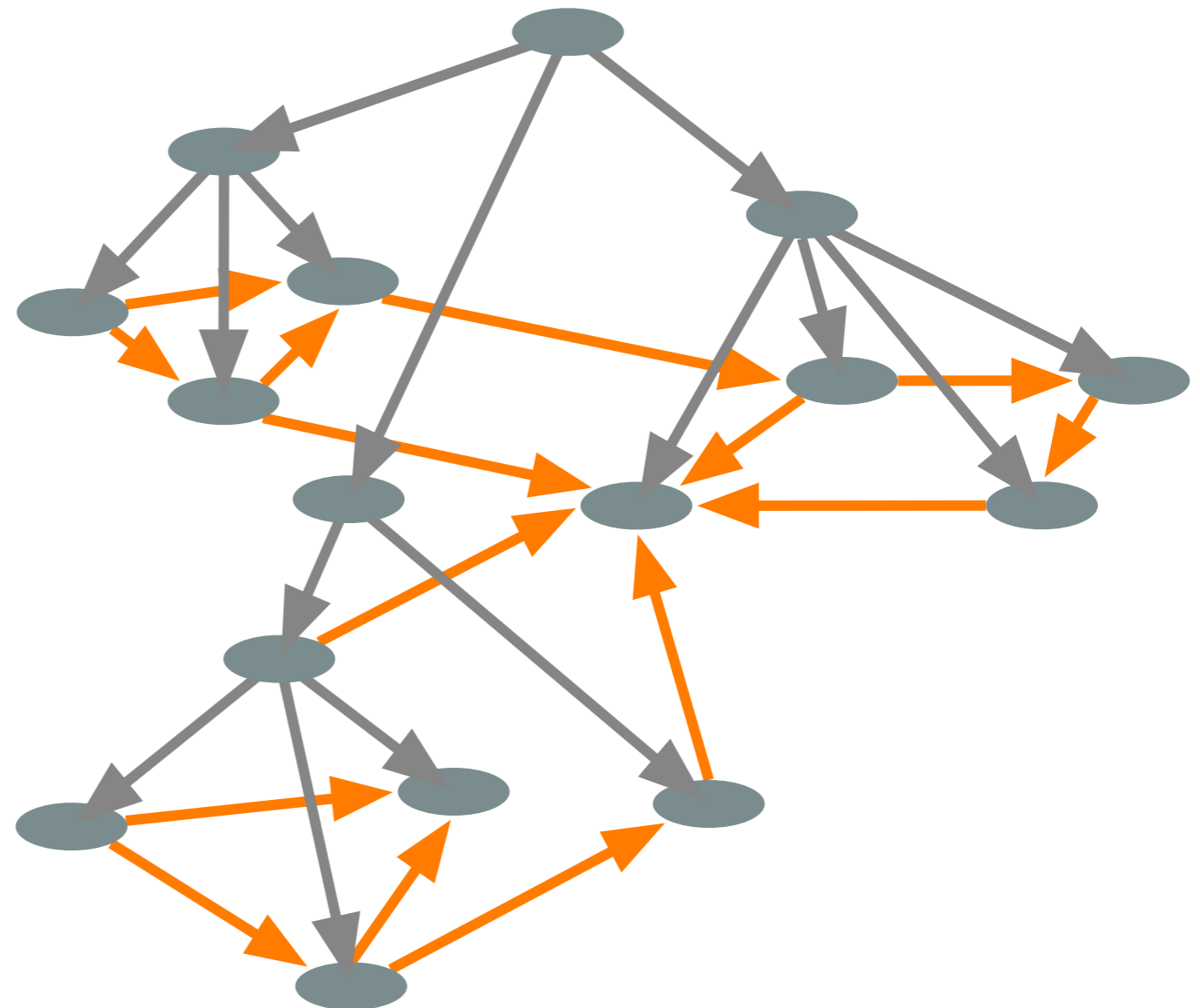
Definition Compound Graph

$D = (V, E, F)$

Inklusionsbaum $T = (V, E)$

Adjazenzkanten F

Inklusionsdiagramm



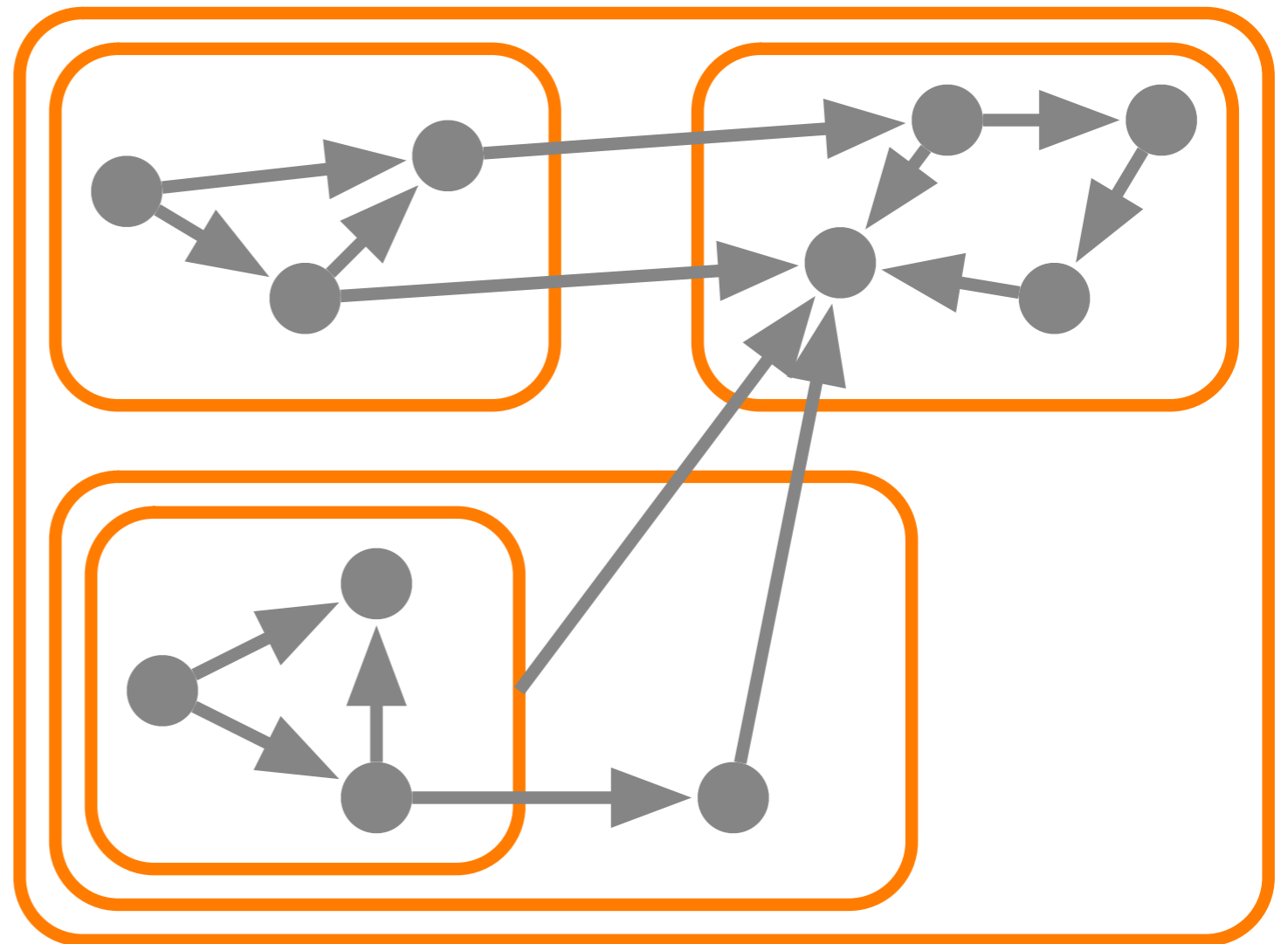
Definition Compound Graph

$D = (V, E, F)$

Inklusionsbaum $T = (V, E)$

Adjazenzkanten F

Inklusionsdiagramm



Definition Sicht

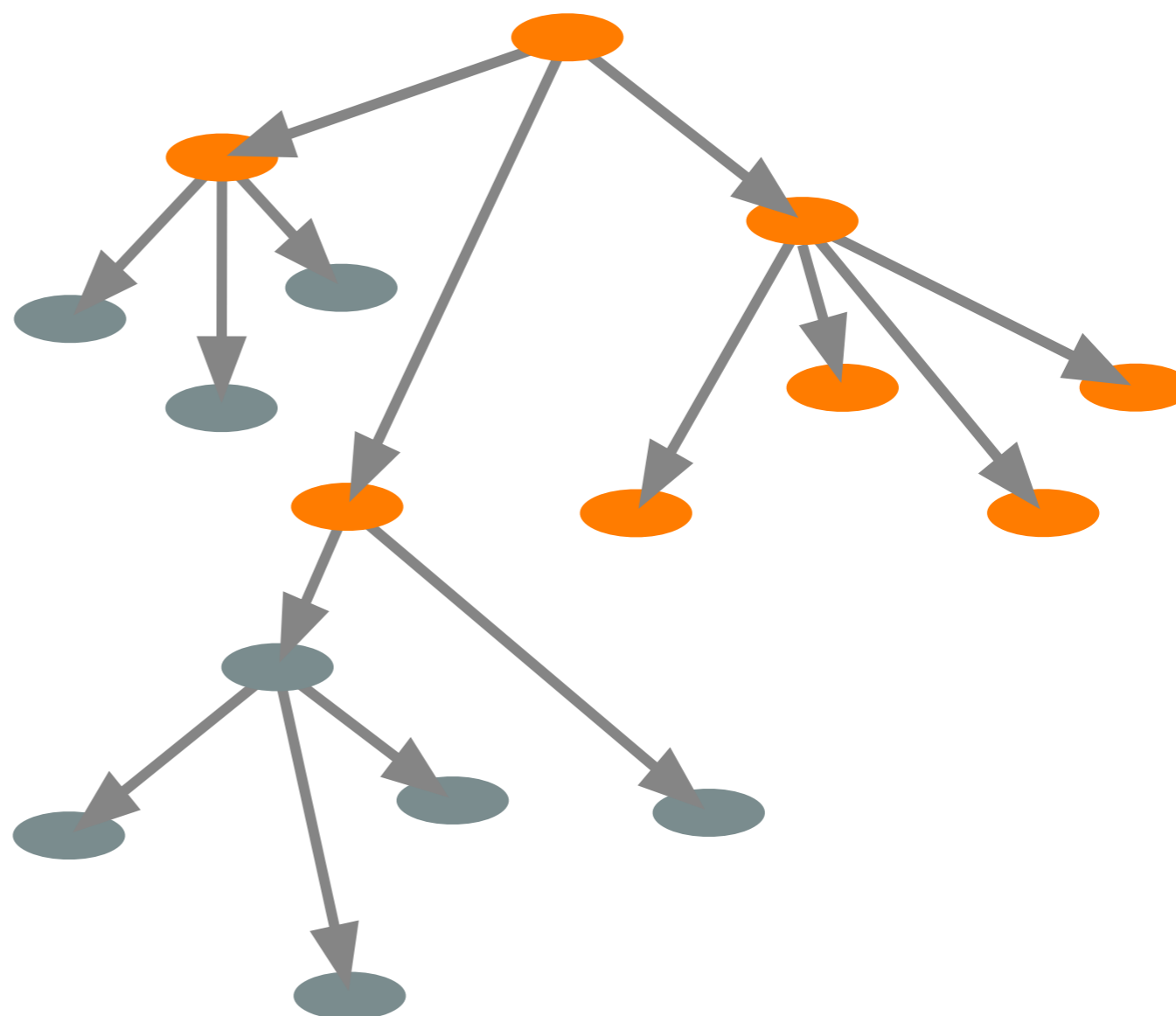
Abstraktion eines
Compound Graphen:

Knoten $U \subseteq V$

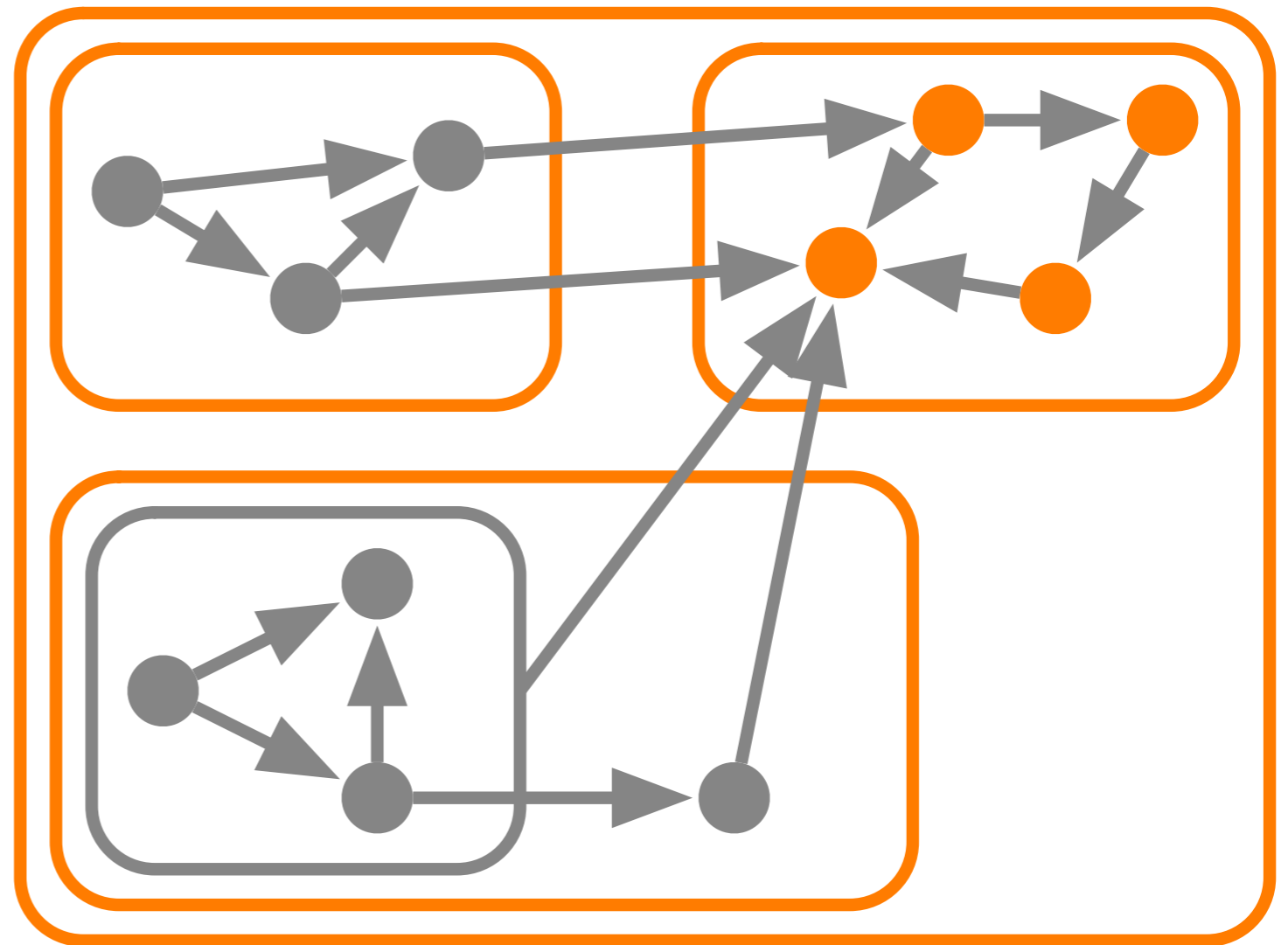
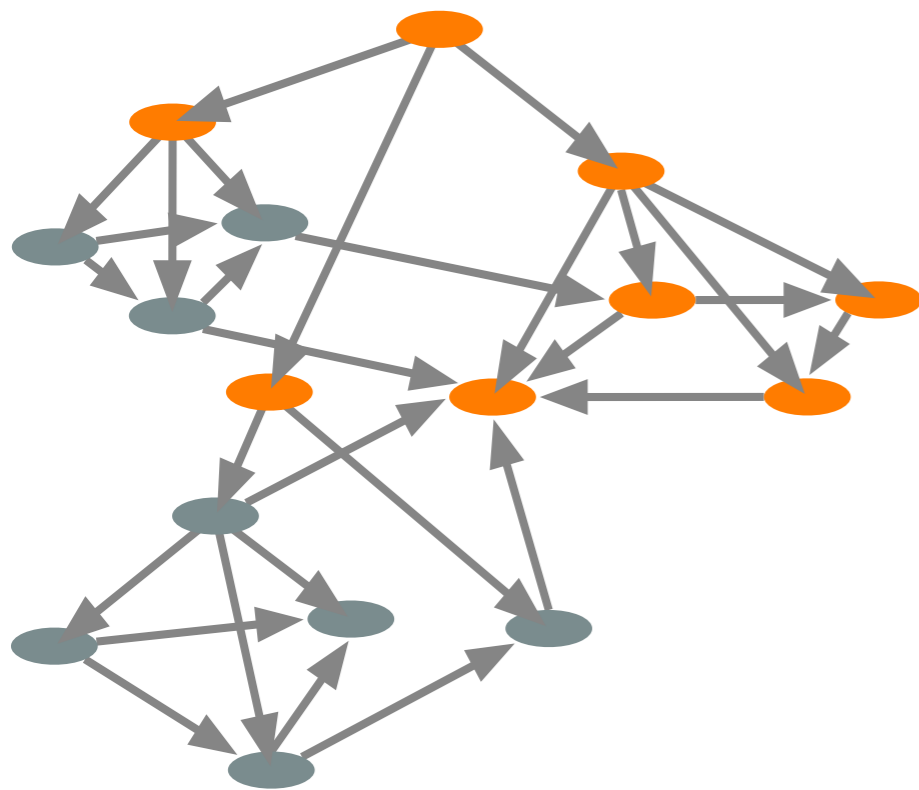
Unterbaum des
Inklusionsbaums, der

❖ die **Wurzel** und

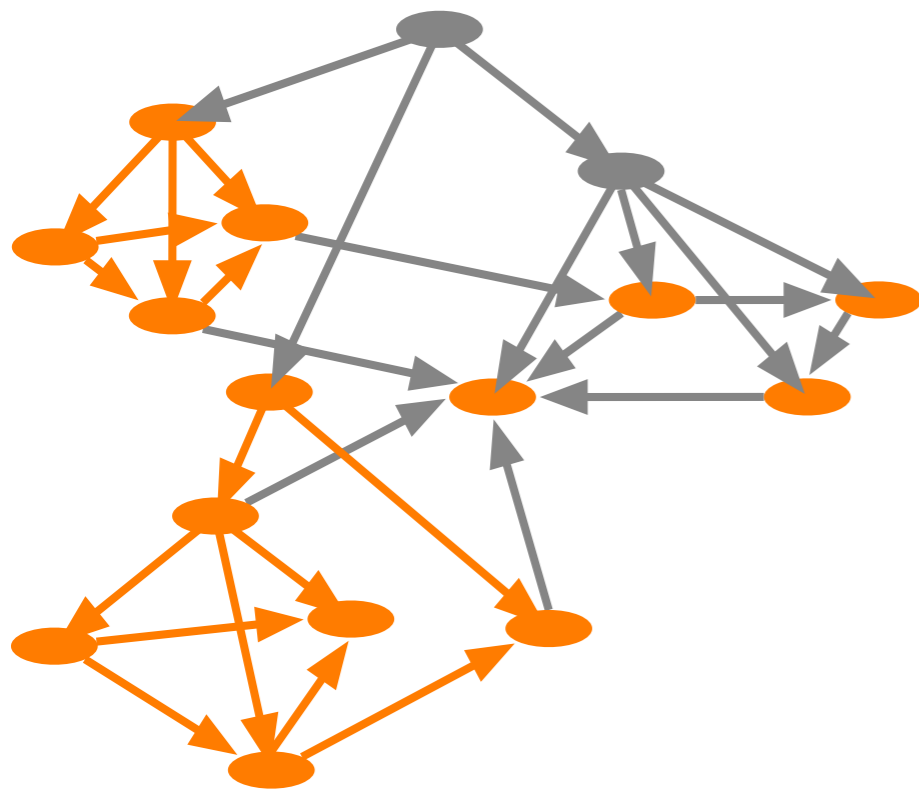
❖ entweder **alle Kinder**
oder **kein Kind** eines
Knotens enthält.



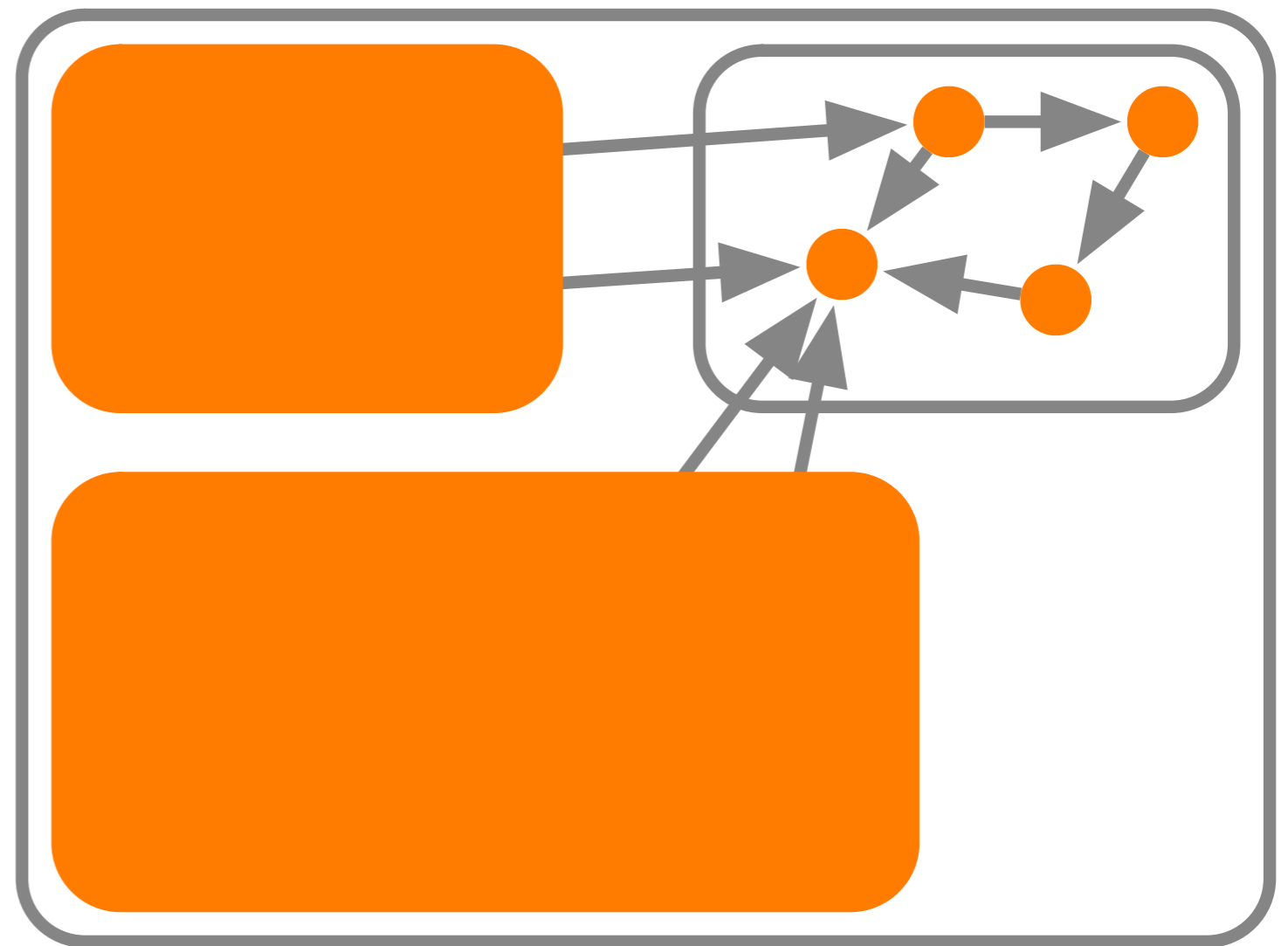
Definition Sicht



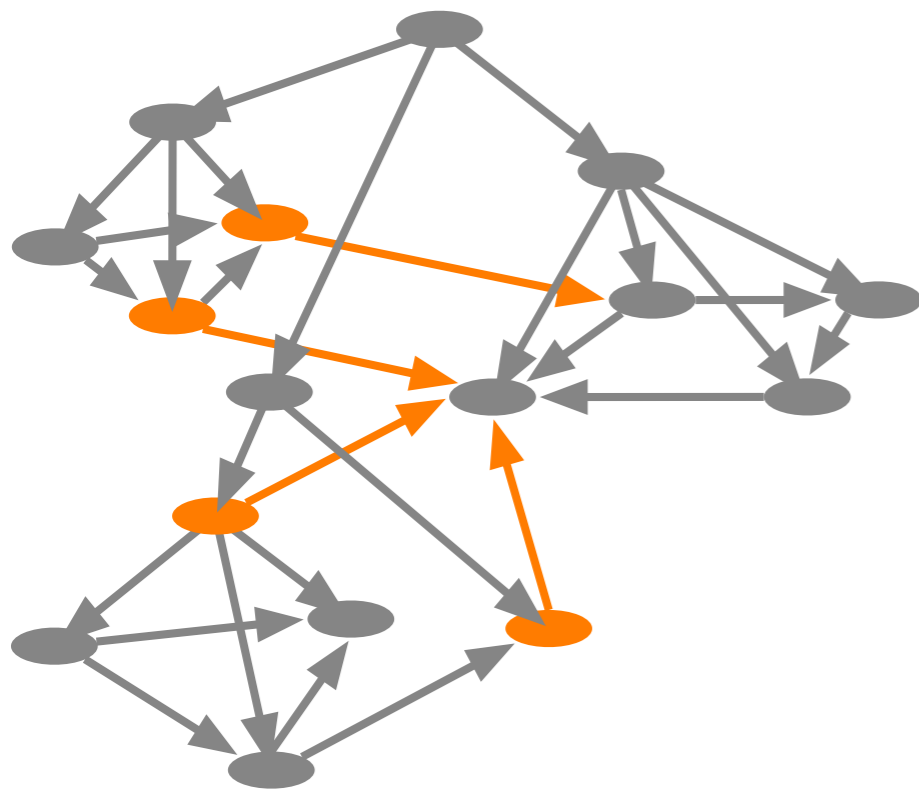
Definition Sicht



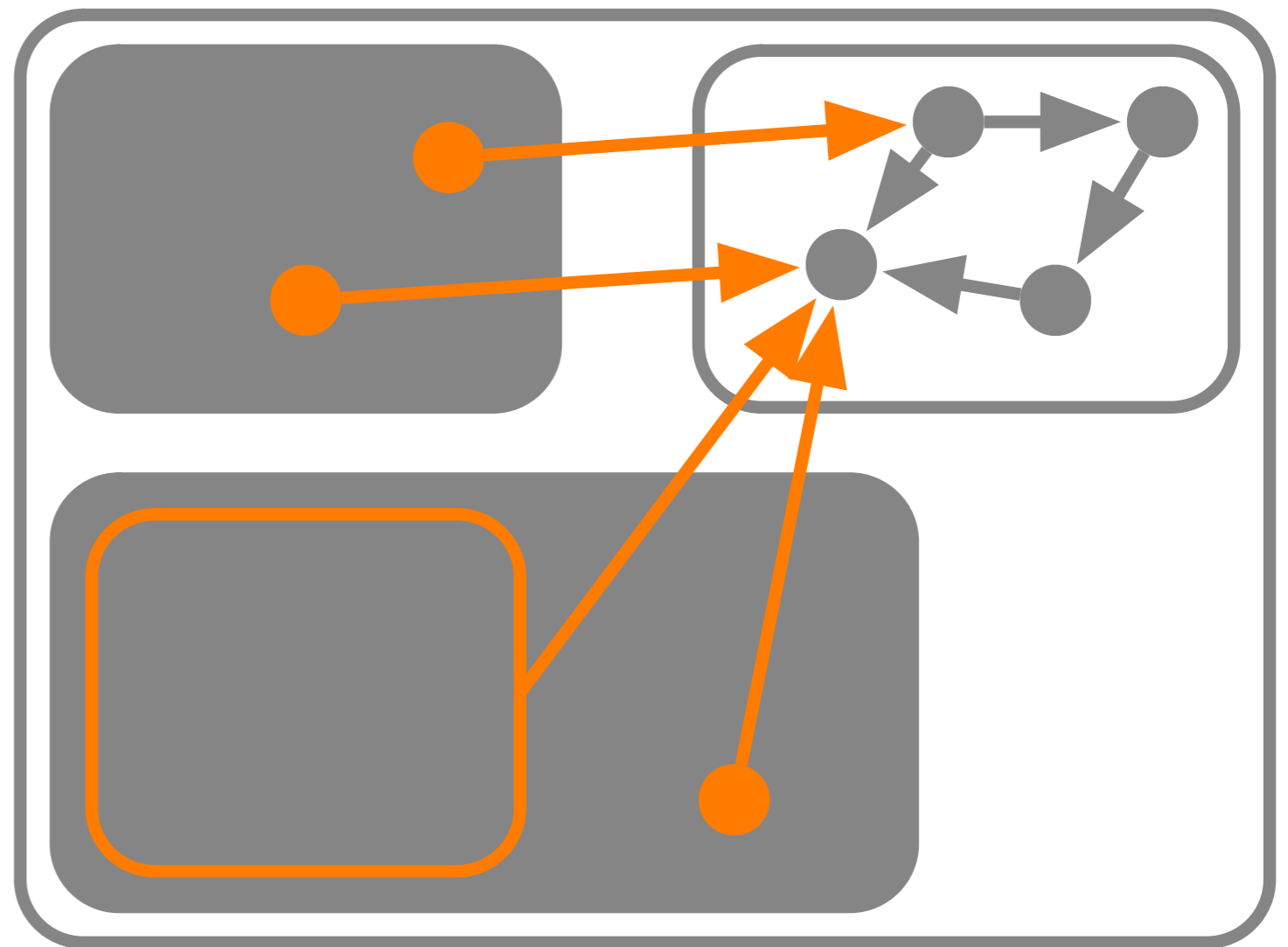
Kontraktion von nicht benötigten Unterbäumen



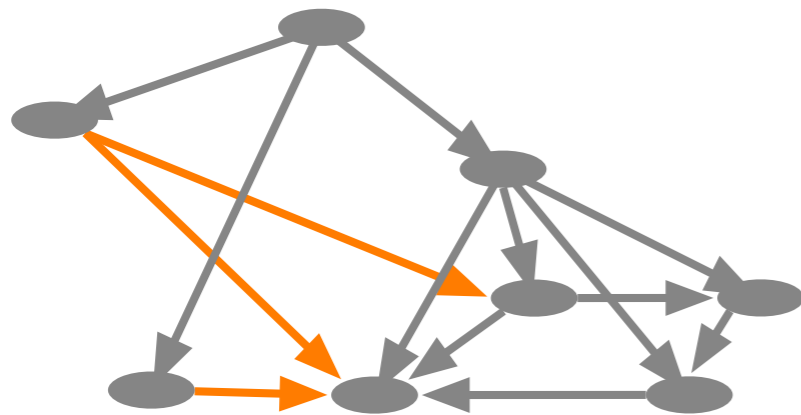
Definition Sicht



Adjazenzkanten von
kontrahierten Knoten...

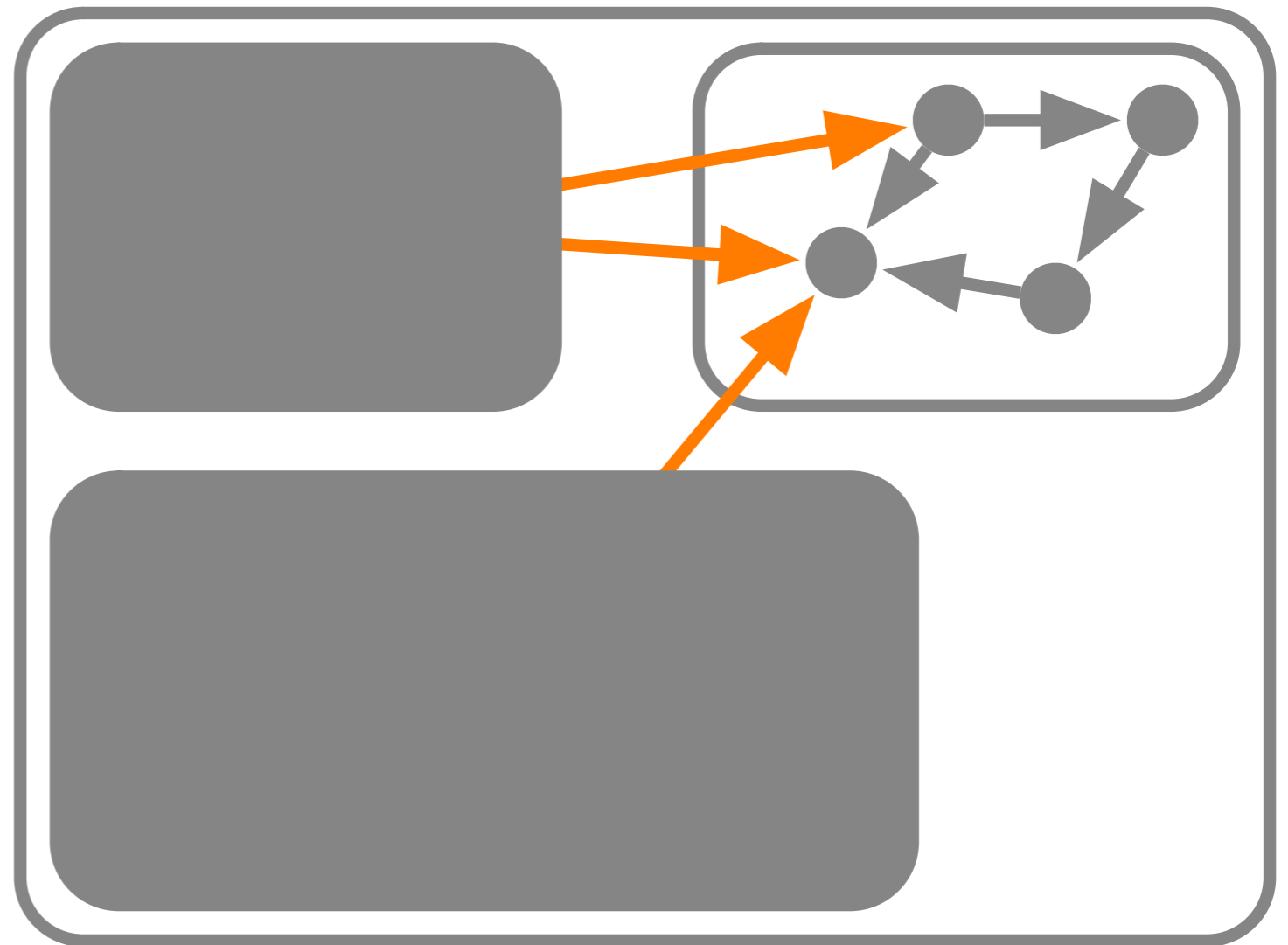


Definition Sicht



...werden zu
abgeleiteten Kanten.

Jede Sicht ist selbst ein
Compound Graph.

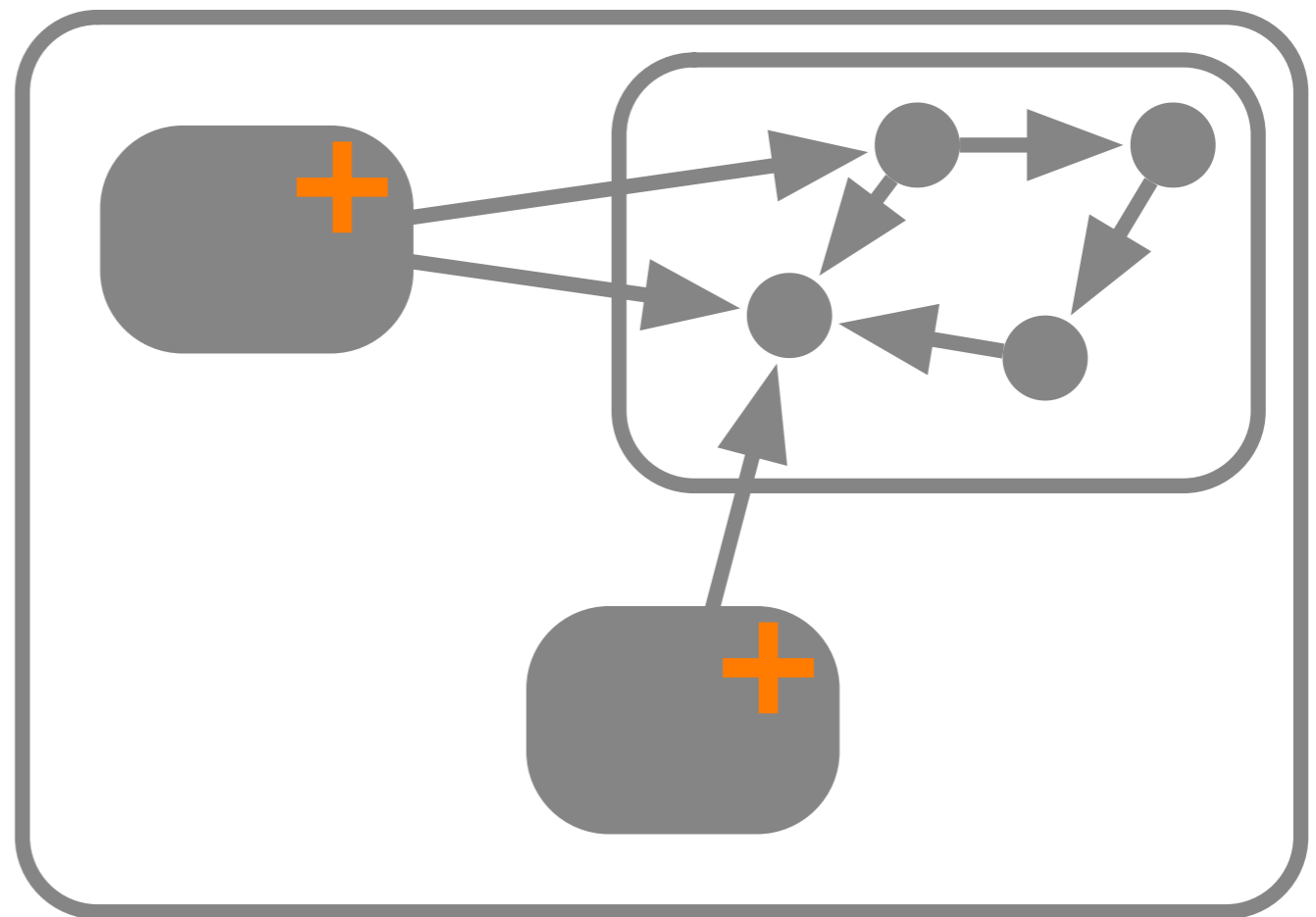


Graph View Maintenance

Effiziente Berechnung
der neuen Sicht nach

Expansion und

Kontraktion

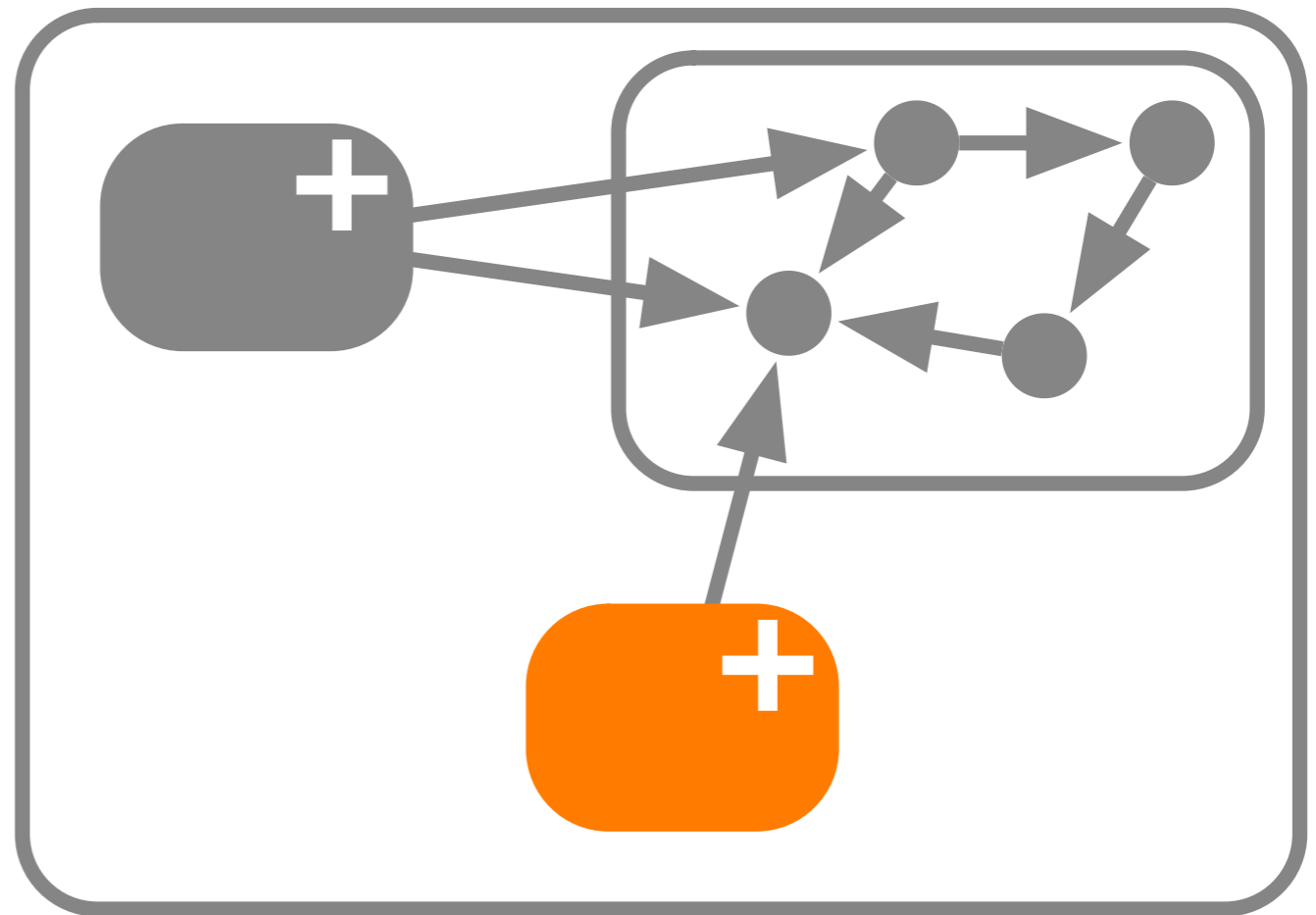


Graph View Maintenance

Effiziente Berechnung
der neuen Sicht nach

Expansion und

Kontraktion

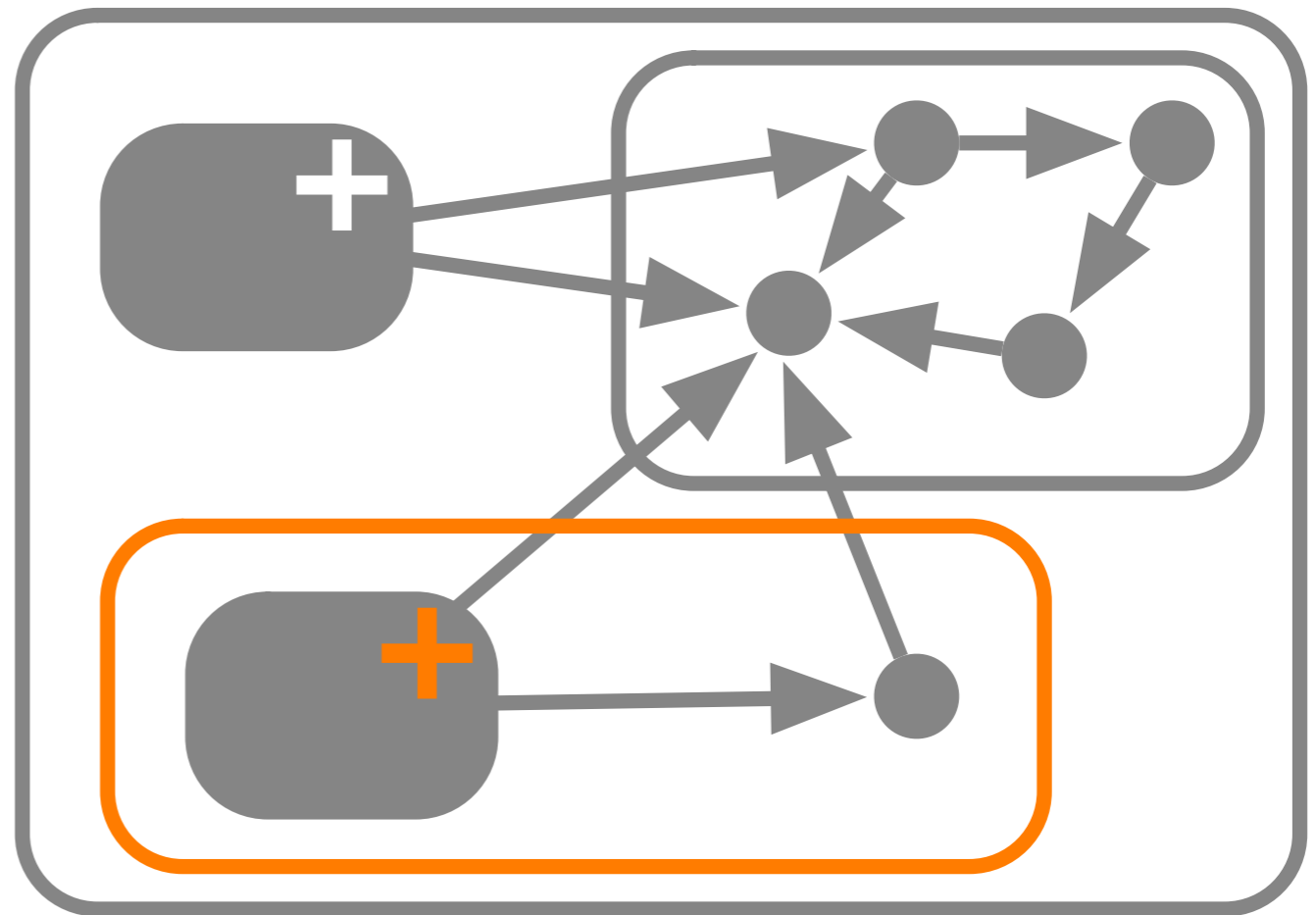


Graph View Maintenance

Effiziente Berechnung
der neuen Sicht nach

Expansion und

Kontraktion

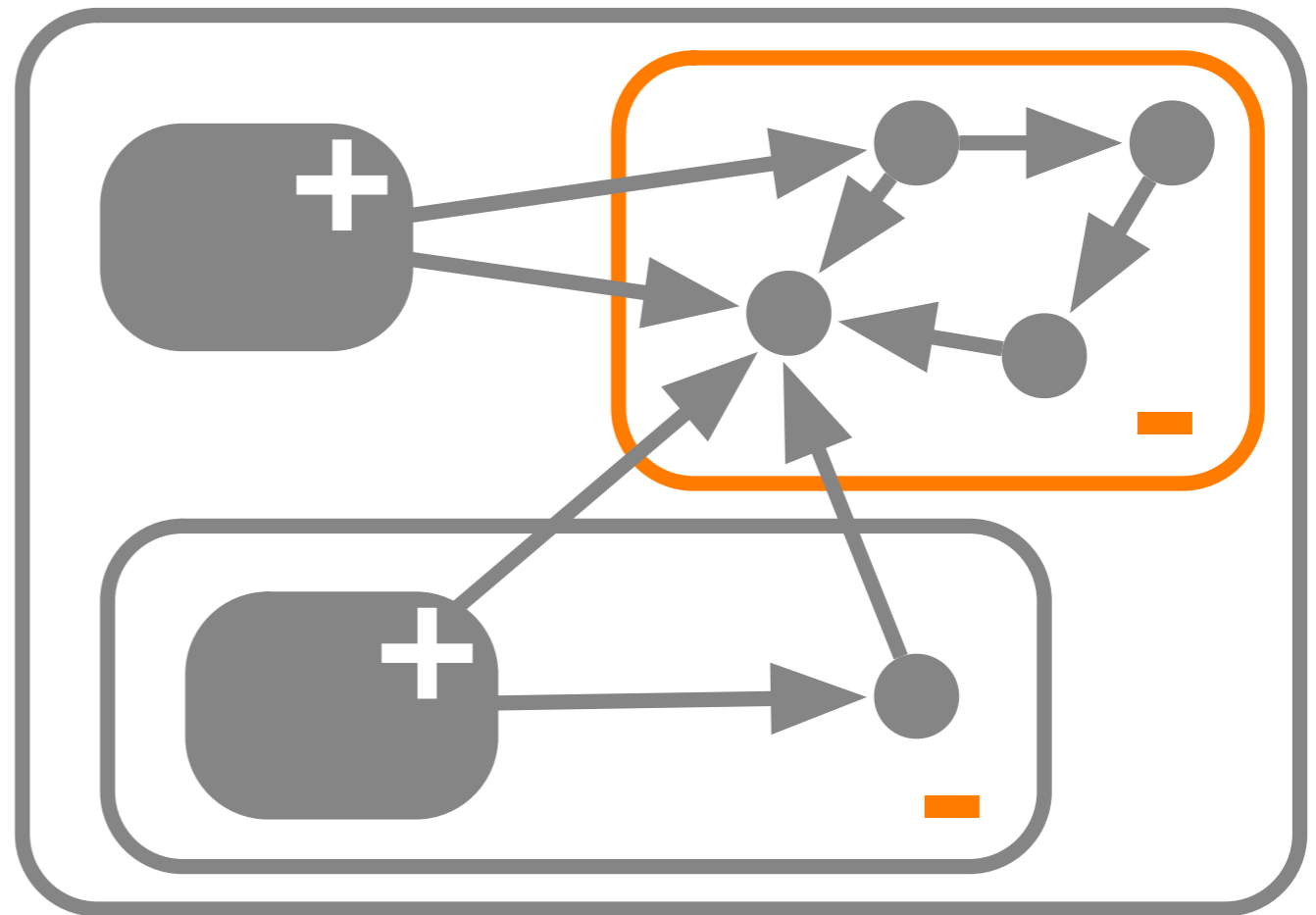


Graph View Maintenance

Effiziente Berechnung
der neuen Sicht nach

Expansion und

Kontraktion

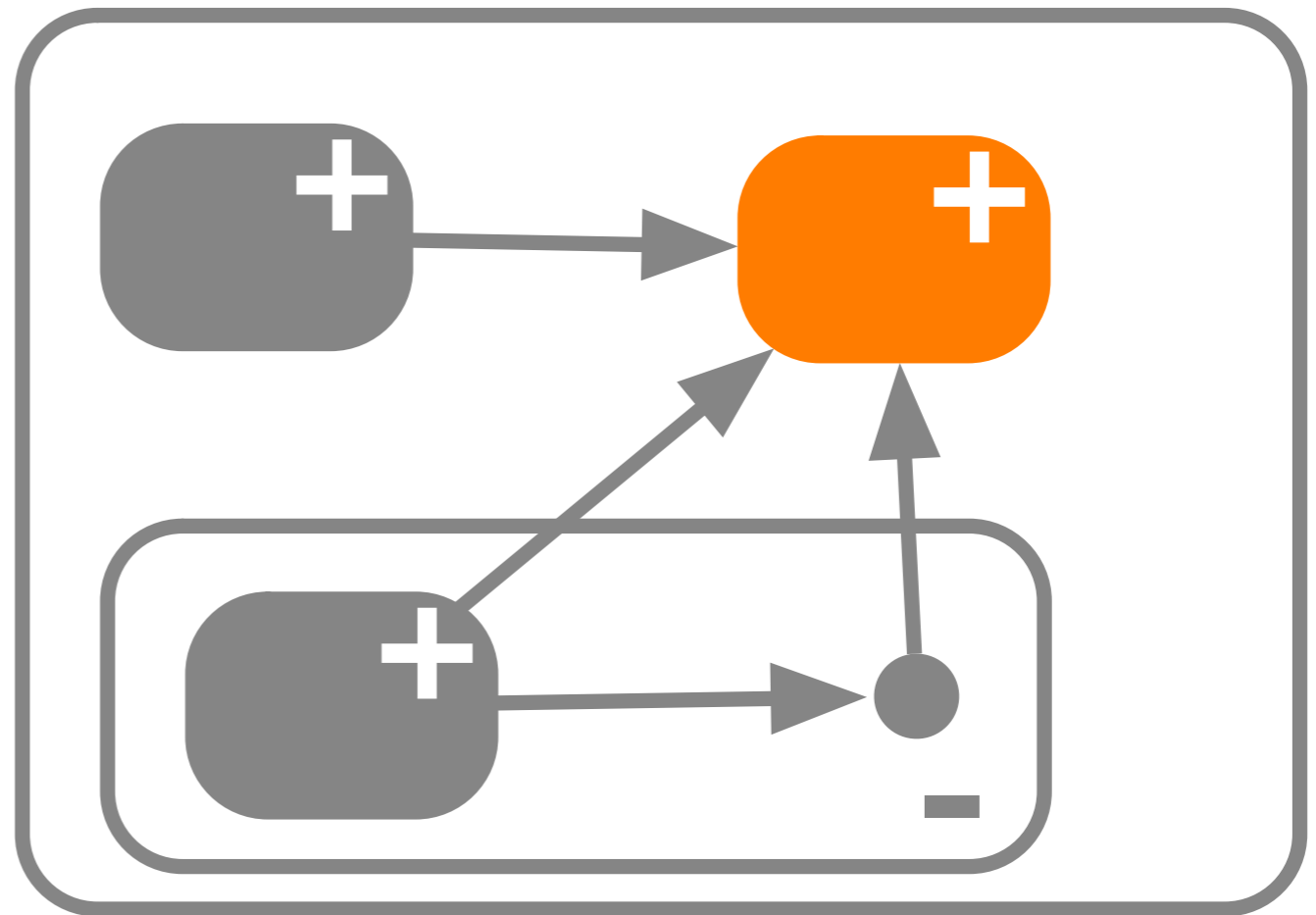


Graph View Maintenance

Effiziente Berechnung
der neuen Sicht nach

Expansion und

Kontraktion



Datenstruktur

Dynamic Tree Cross Products

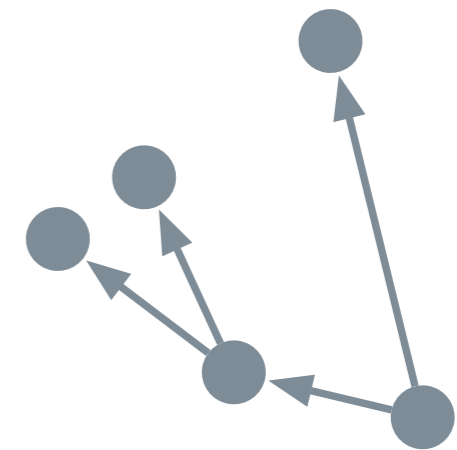
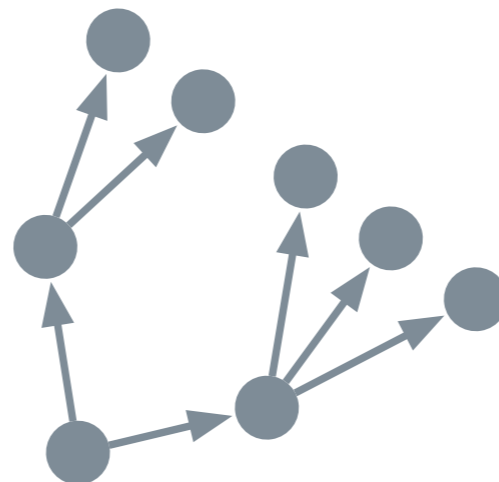
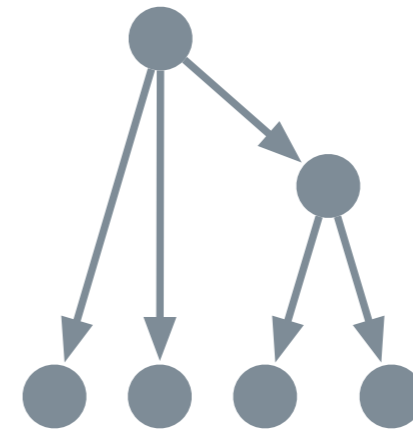
Marcus Raitner. **Dynamic Tree Cross Products**. In *Proc. 15th Intl. Symposium on Algorithms and Computation (ISAAC)*, 2004

Definition Baum-Kreuzprodukt

Ein d-dimensionales
Baum-Kreuzprodukt
besteht aus

Bäumen

$$T_1, \dots, T_d$$



Definition Baum-Kreuzprodukt

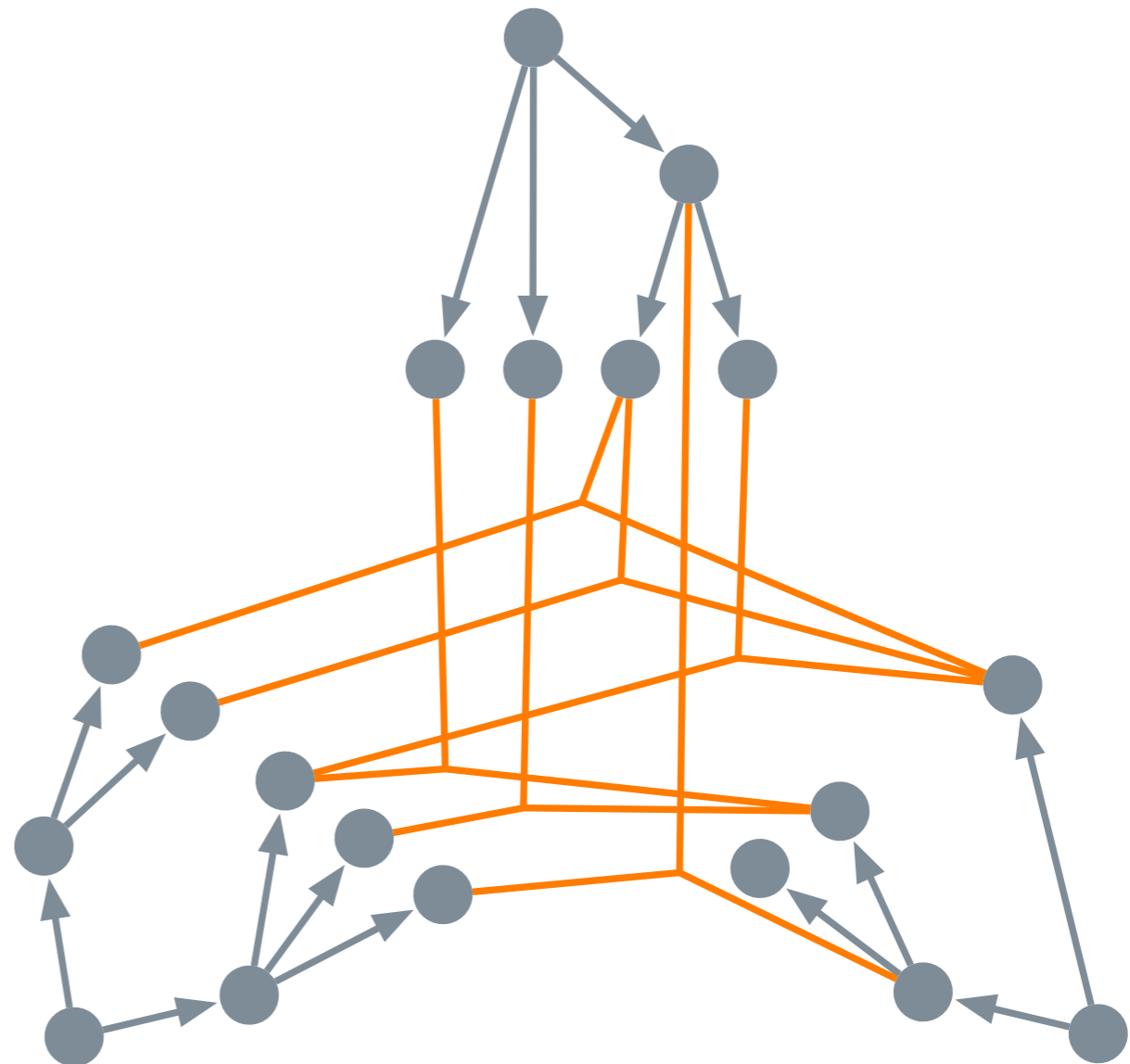
Ein d-dimensionales
Baum-Kreuzprodukt
besteht aus

Bäumen

$$T_1, \dots, T_d$$

und d-dimensionalen
(Hyper-)Kanten

$$E \subseteq \prod_{i=1}^d V(T_i)$$



Definition Baum-Kreuzprodukt

Ein d-dimensionales
Baum-Kreuzprodukt
besteht aus

Bäumen

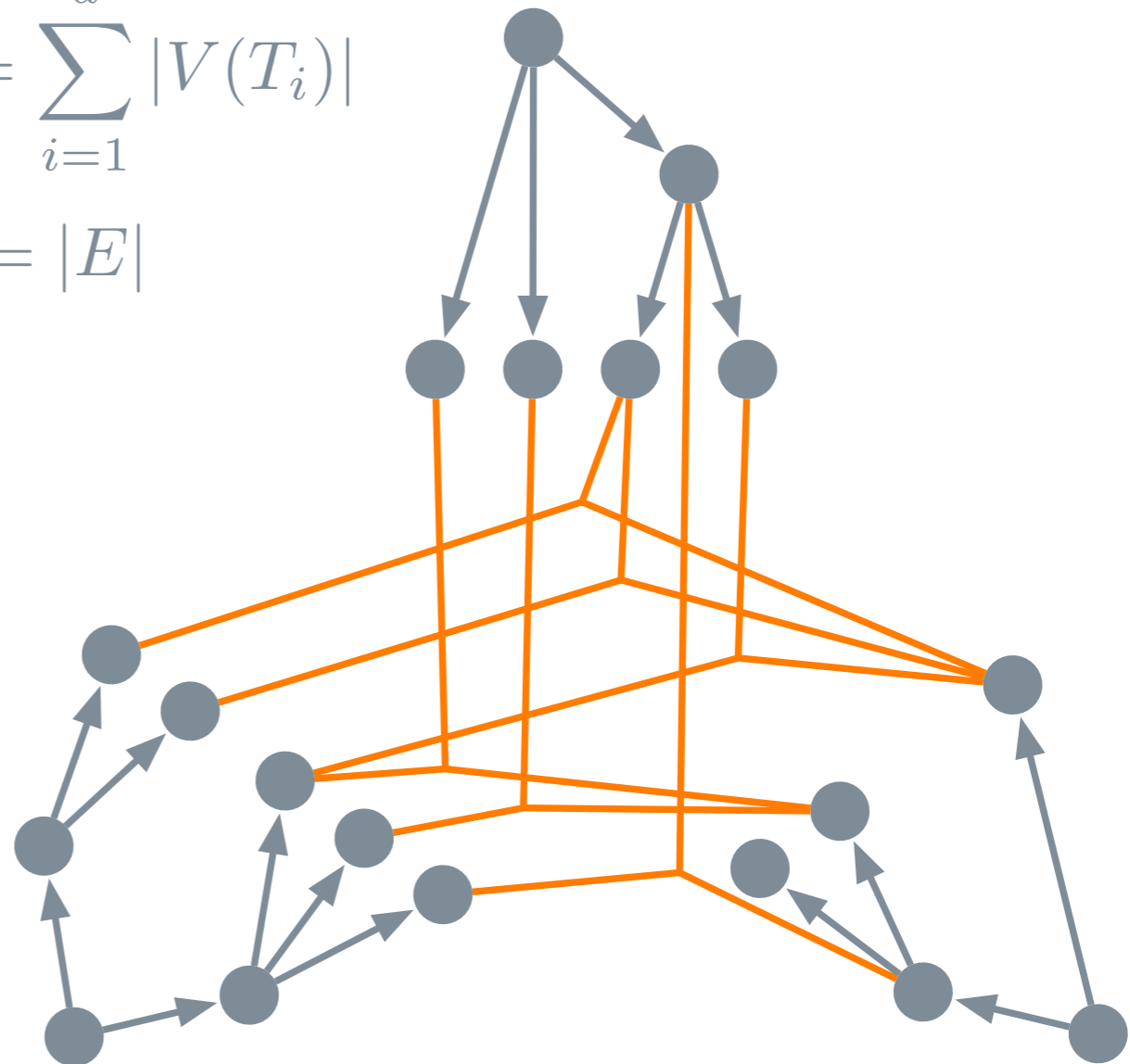
$$T_1, \dots, T_d$$

und d-dimensionalen
(Hyper-)Kanten

$$E \subseteq \prod_{i=1}^d V(T_i)$$

$$n = \sum_{i=1}^d |V(T_i)|$$

$$m = |E|$$

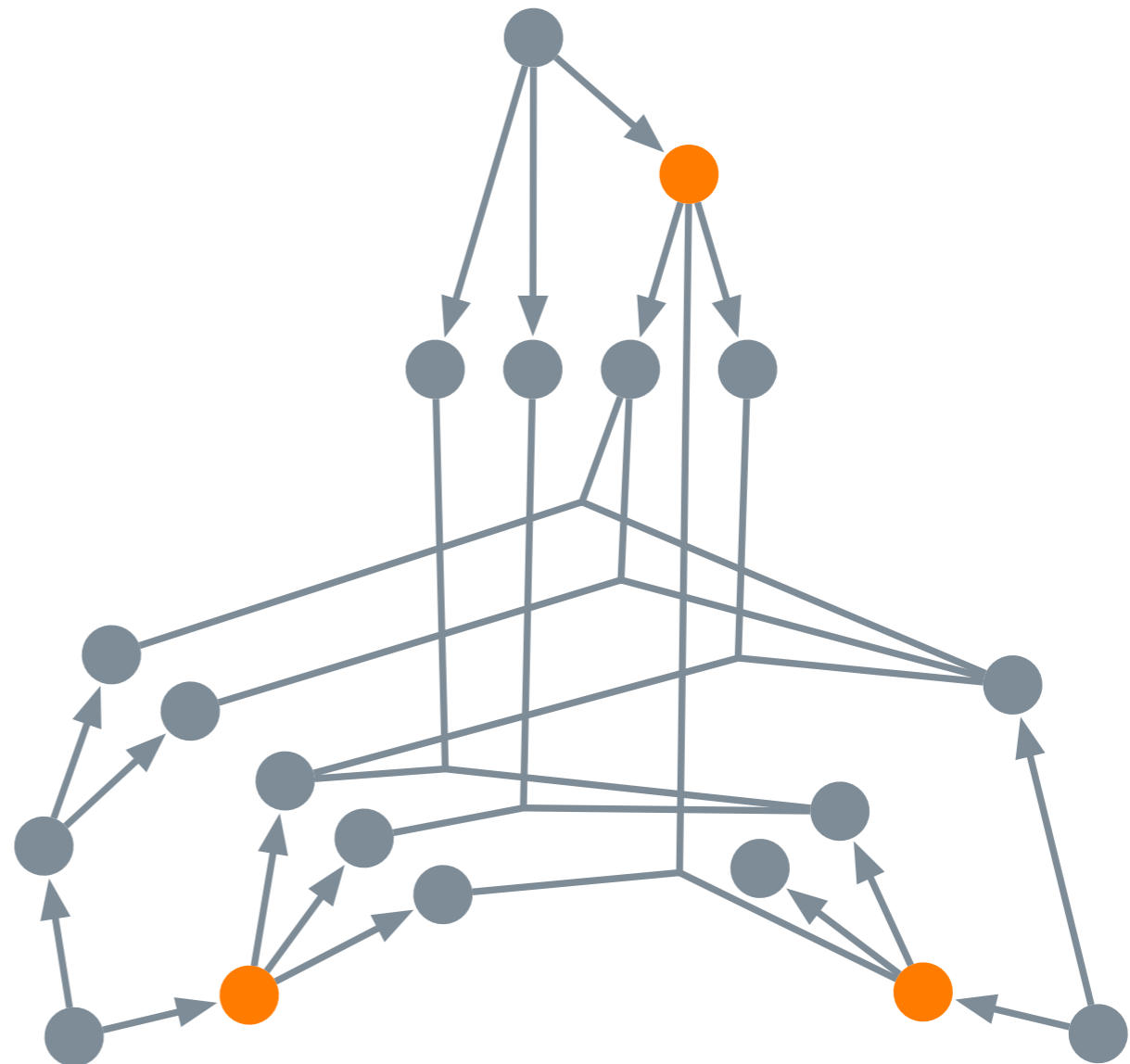


Definition Abgeleitete Kante

Ein Tupel

$$\mathbf{u} = (u_1, \dots, u_d) \in \prod_{i=1}^d V(T_i)$$

heißt **abgeleitete Kante**



Definition Abgeleitete Kante

Ein Tupel

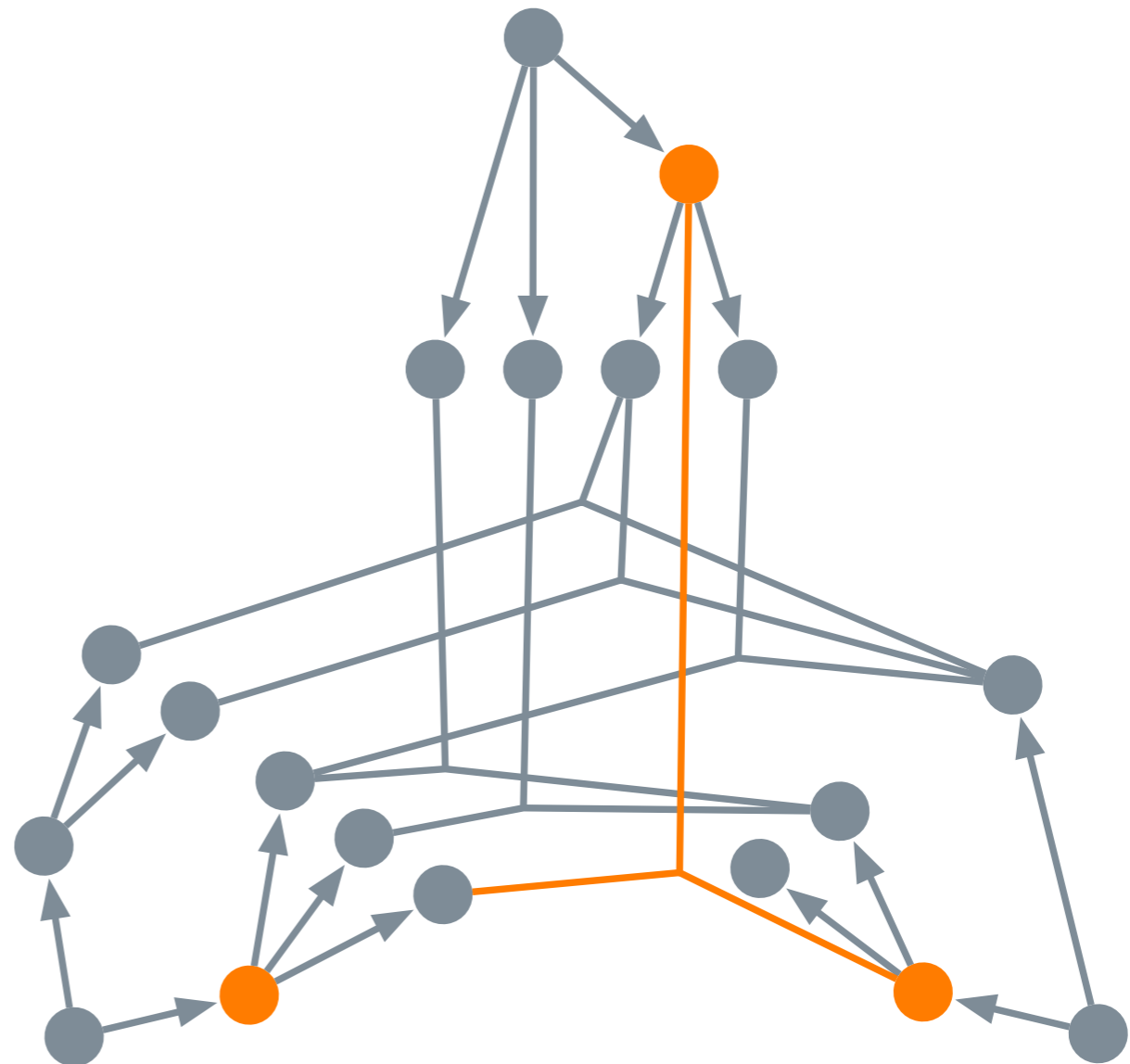
$$u = (u_1, \dots, u_d) \in \prod_{i=1}^d V(T_i)$$

heißt **abgeleitete Kante** falls es mindestens eine Kante

$$u' = (u'_1, \dots, u'_d) \in E$$

gibt derart dass

$$\forall 1 \leq i \leq d : u'_i \in \text{desc}(u_i)$$



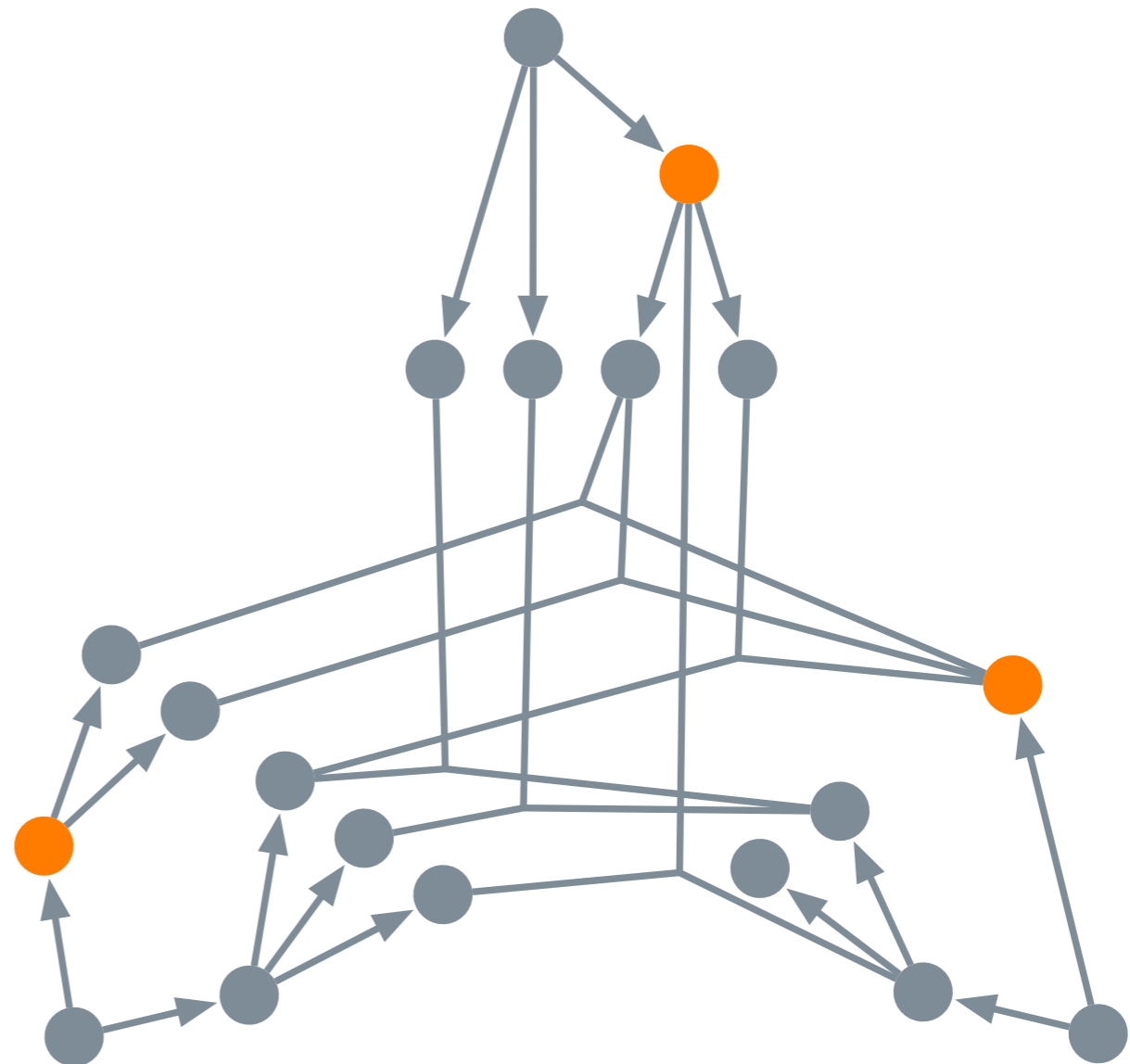
Anfrage edgeQuery

edgeQuery(u)

bestimmt ob ein Tupel

$$u \in \prod_{i=1}^d V(T_i)$$

eine abgeleitete Kante ist.



Anfrage edgeReport

edgeQuery(u)

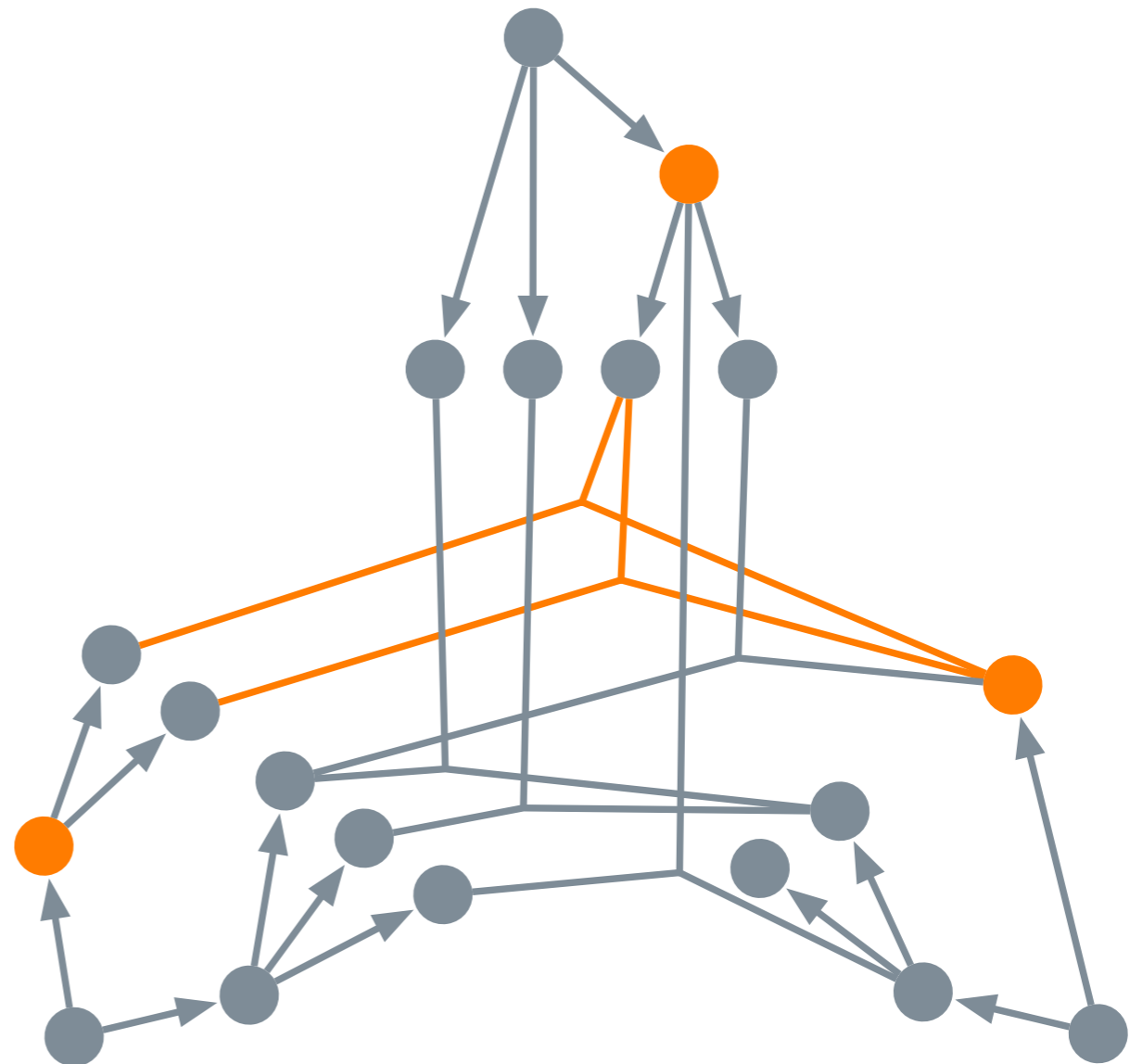
bestimmt ob ein Tupel

$$u \in \prod_{i=1}^d V(T_i)$$

eine abgeleitete Kante ist.

edgeReport(u)

bestimmt alle Kanten zwischen den Unterbäumen.



Anfrage edgeExpand

edgeExpand(u, j)

bestimmt zu einer
abgeleiteten Kante

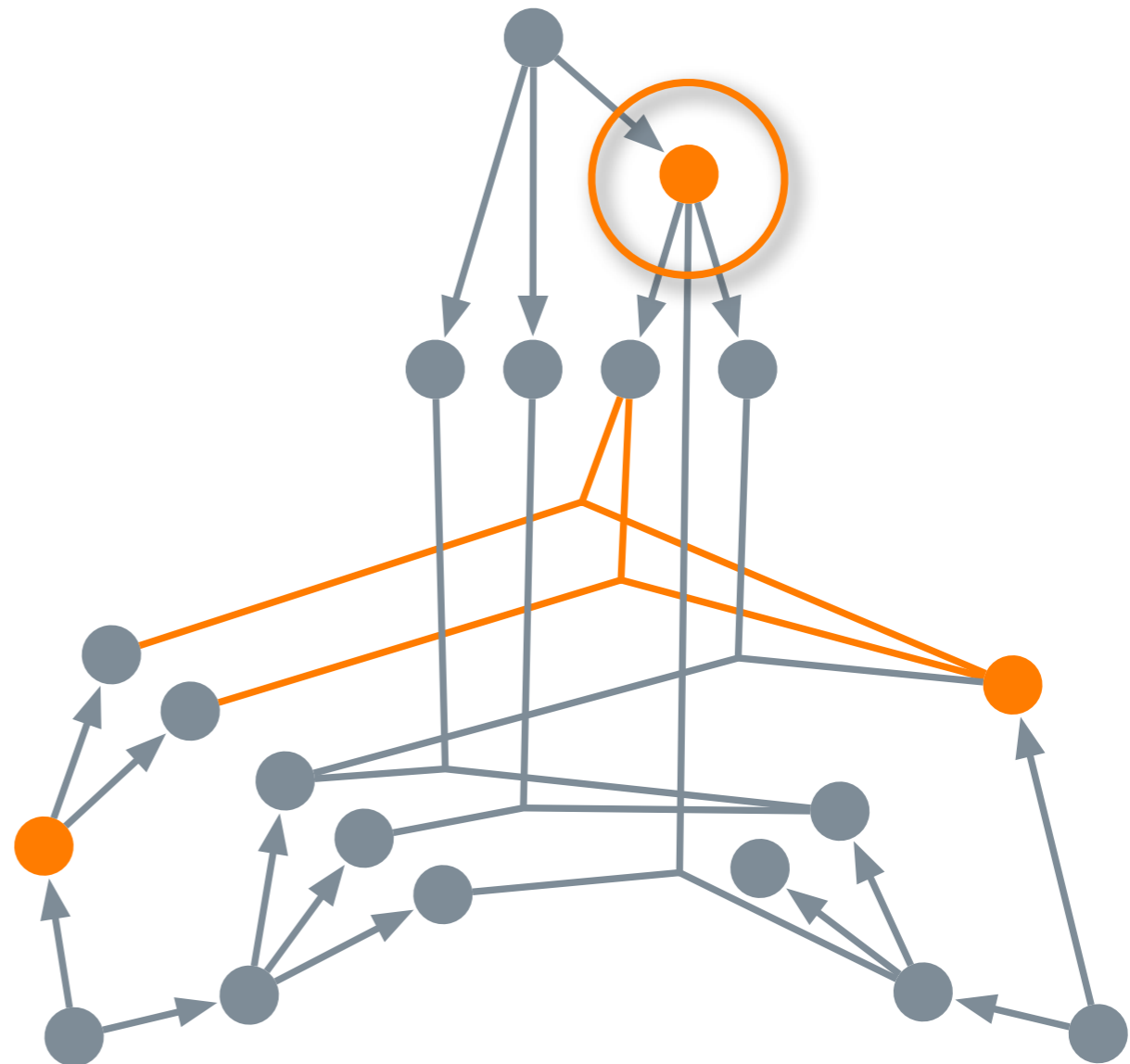
$$u \in \prod_{i=1}^d V(T_i)$$

alle abgeleiteten Kanten

($u_1, \dots, u_{j-1}, x, u_{j+1}, \dots, u_d$)

derart dass

$$x \in \text{children}(u_j)$$



Anfrage edgeExpand

edgeExpand(u, j)

bestimmt zu einer
abgeleiteten Kante

$$u \in \prod_{i=1}^d V(T_i)$$

alle abgeleiteten Kanten

($u_1, \dots, u_{j-1}, x, u_{j+1}, \dots, u_d$)

derart dass

$$x \in \text{children}(u_j)$$



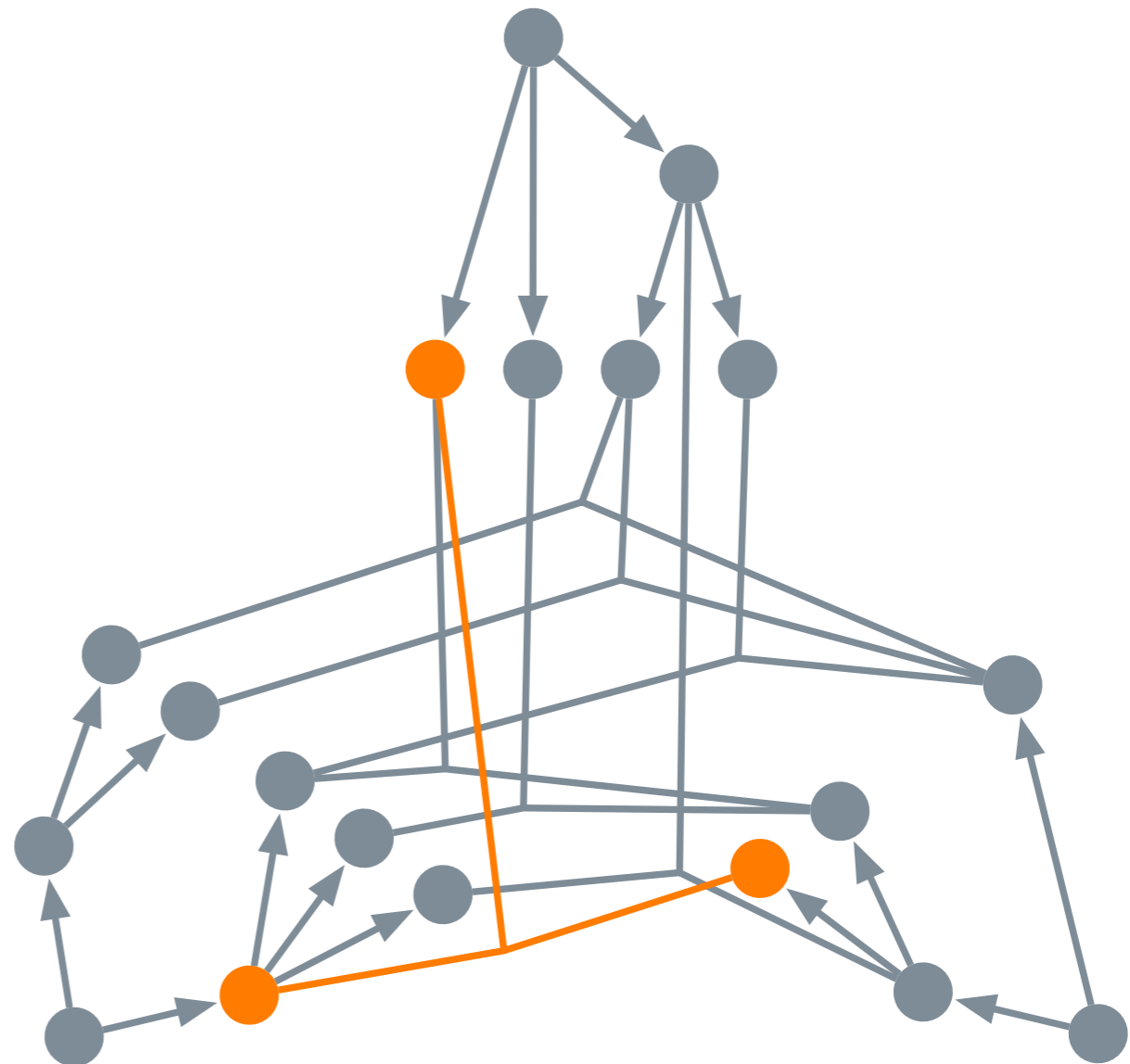
Update Kanten

`newEdge(u)`

fügt eine neue Kante ein

`deleteEdge(u)`

löscht eine Kante



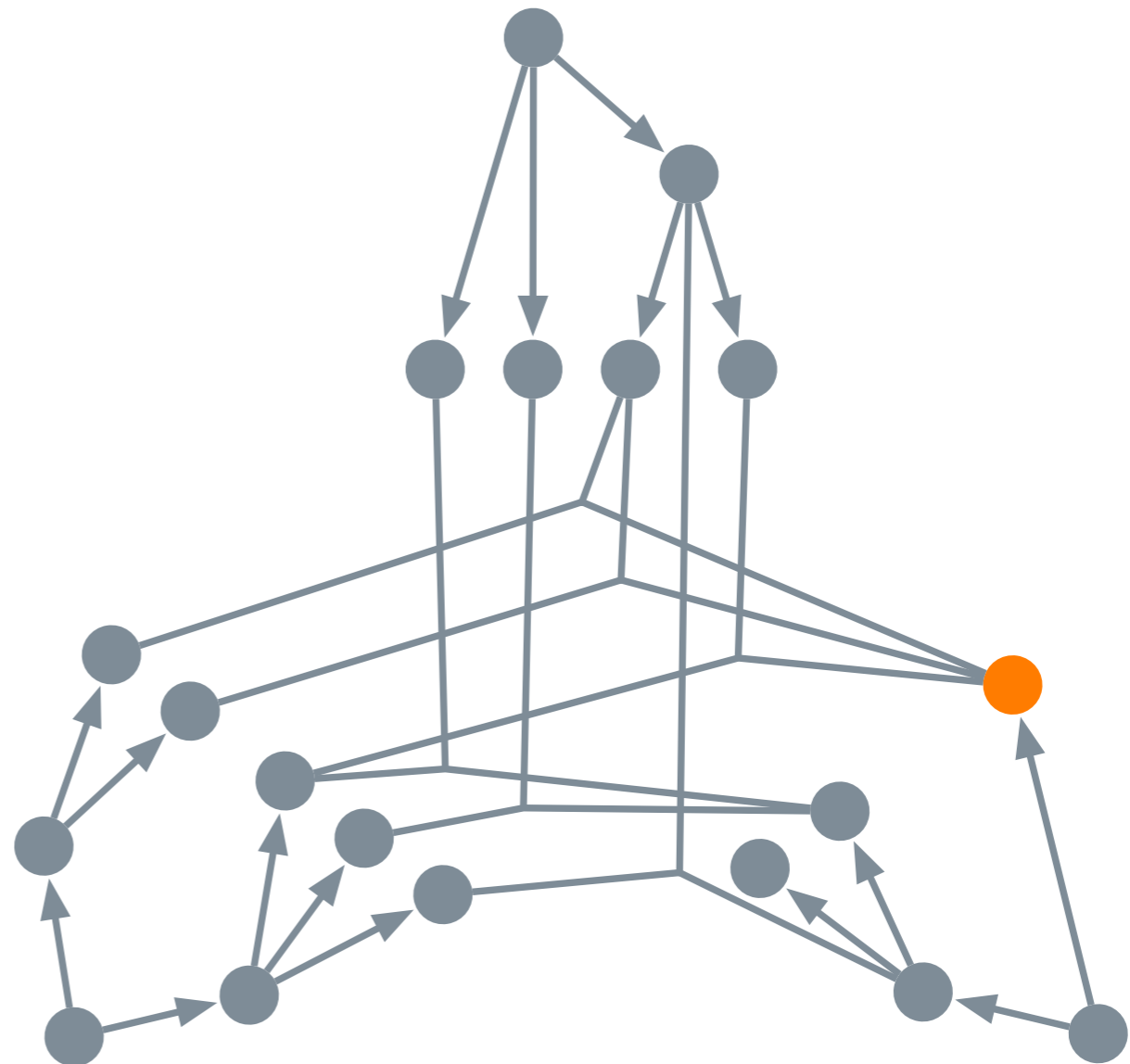
Update Blätter

`newLeaf(u)`

fügt ein neues Blatt an

`deleteLeaf(u)`

löscht ein Blatt



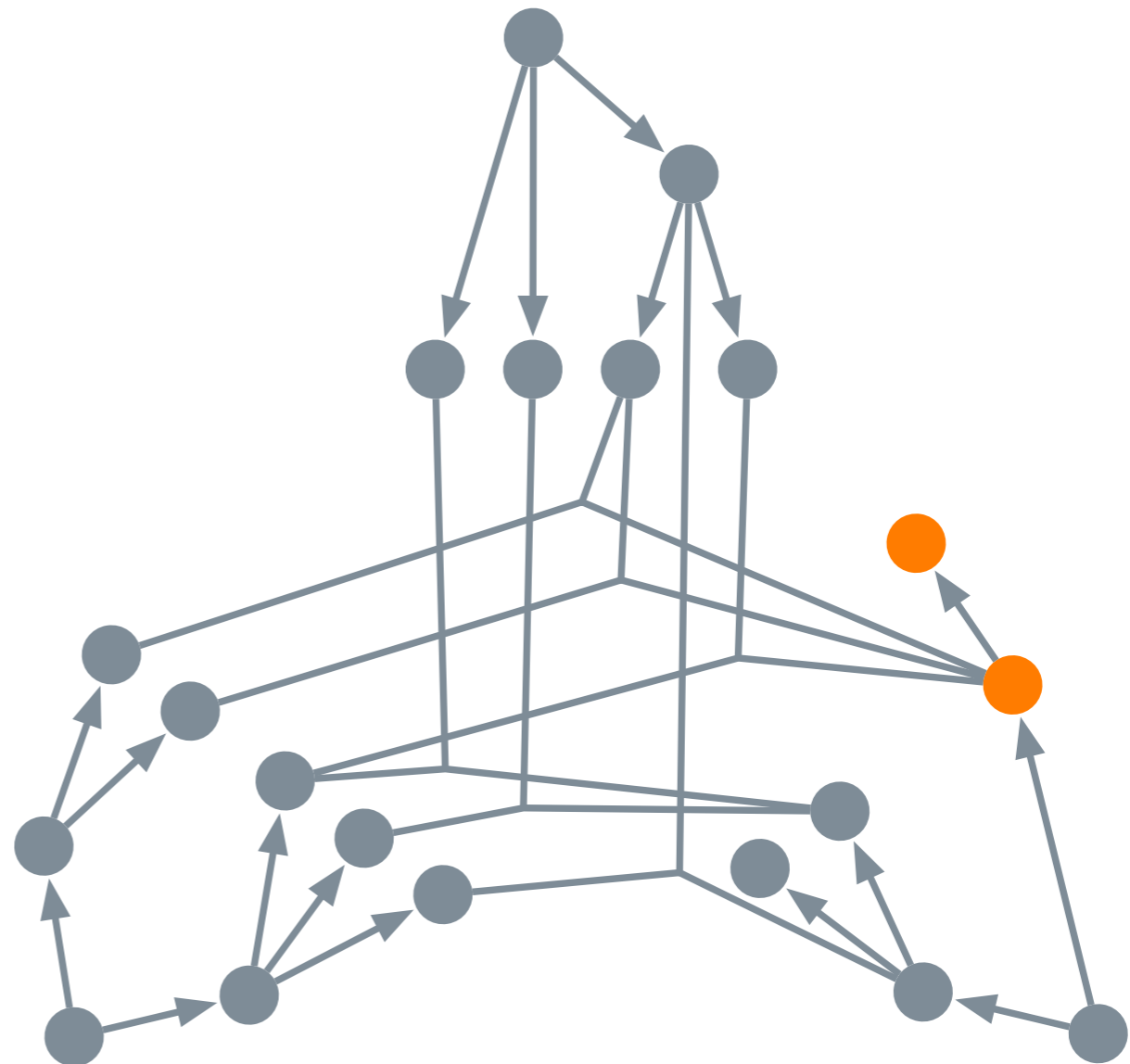
Update Blätter

`newLeaf(u)`

fügt ein neues Blatt an

`deleteLeaf(u)`

löscht ein Blatt



Vorarbeiten

[Buchsbaum et al. '00]

Buchsbaum, Goodrich, Westbrook

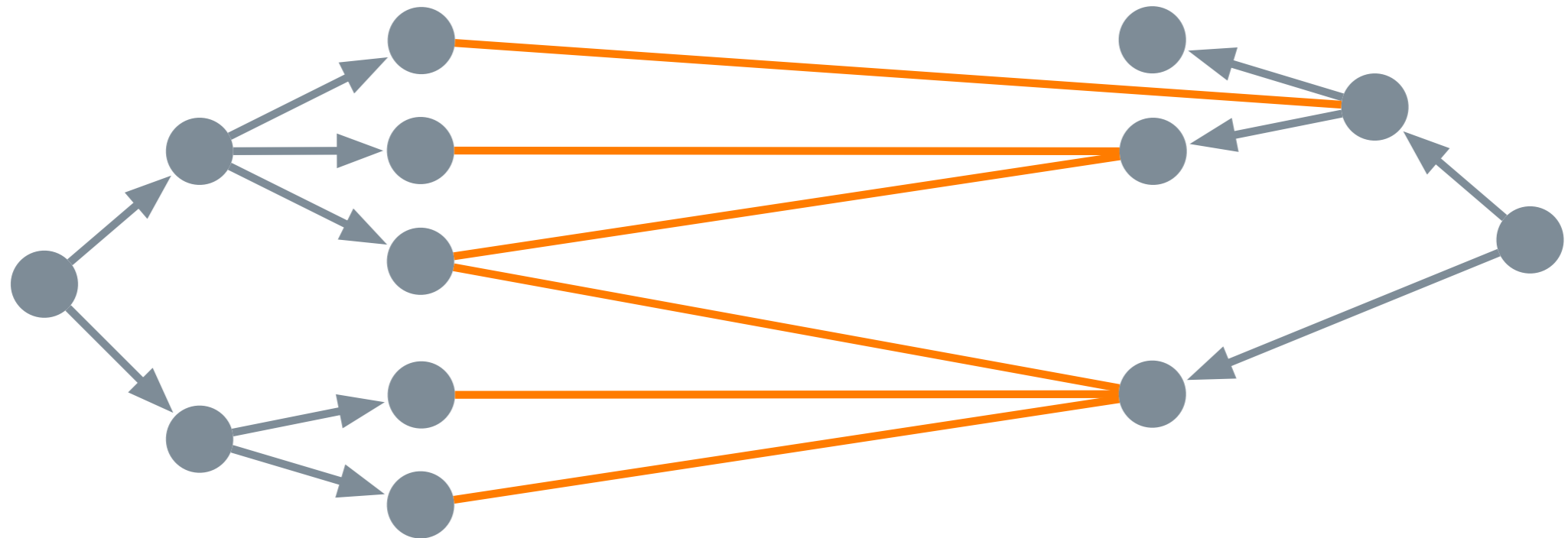
Range Searching over Tree Cross Products

Proc. 8th European Symposium on Algorithms (ESA), 2000.

Nachteil: Knotenmenge **statisch**.

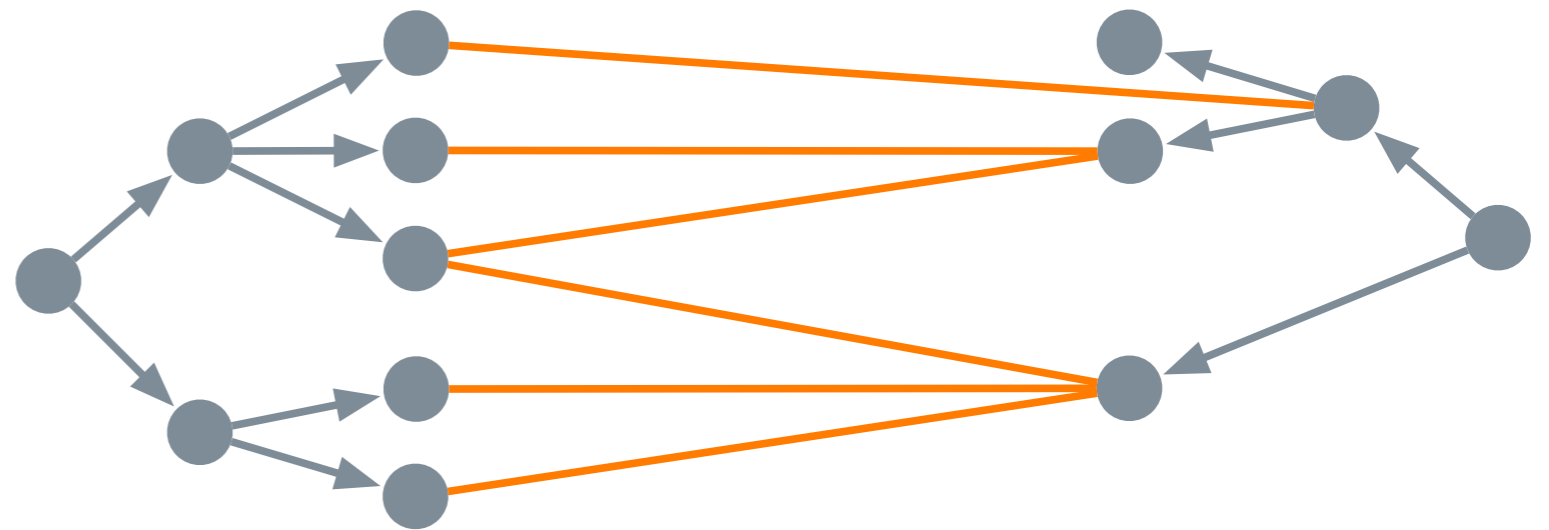
Hier: Erweiterung in Bezug auf **Einfügen und Löschen von Blättern**.

Einfacher Fall Zwei Bäume



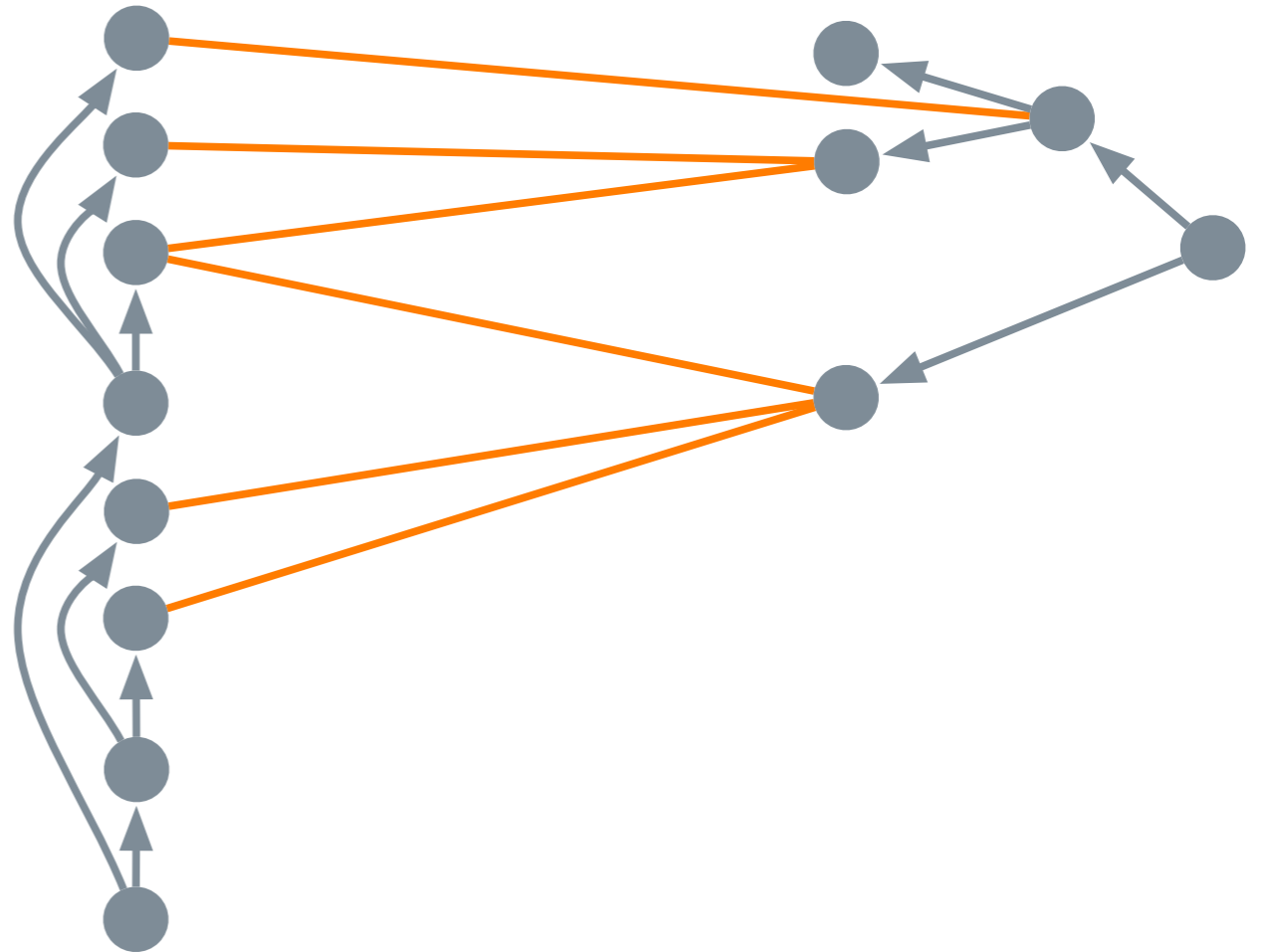
Knotenordnung

In jedem Baum wird eine **totale Ordnung** der Knoten erzeugt (DFS).



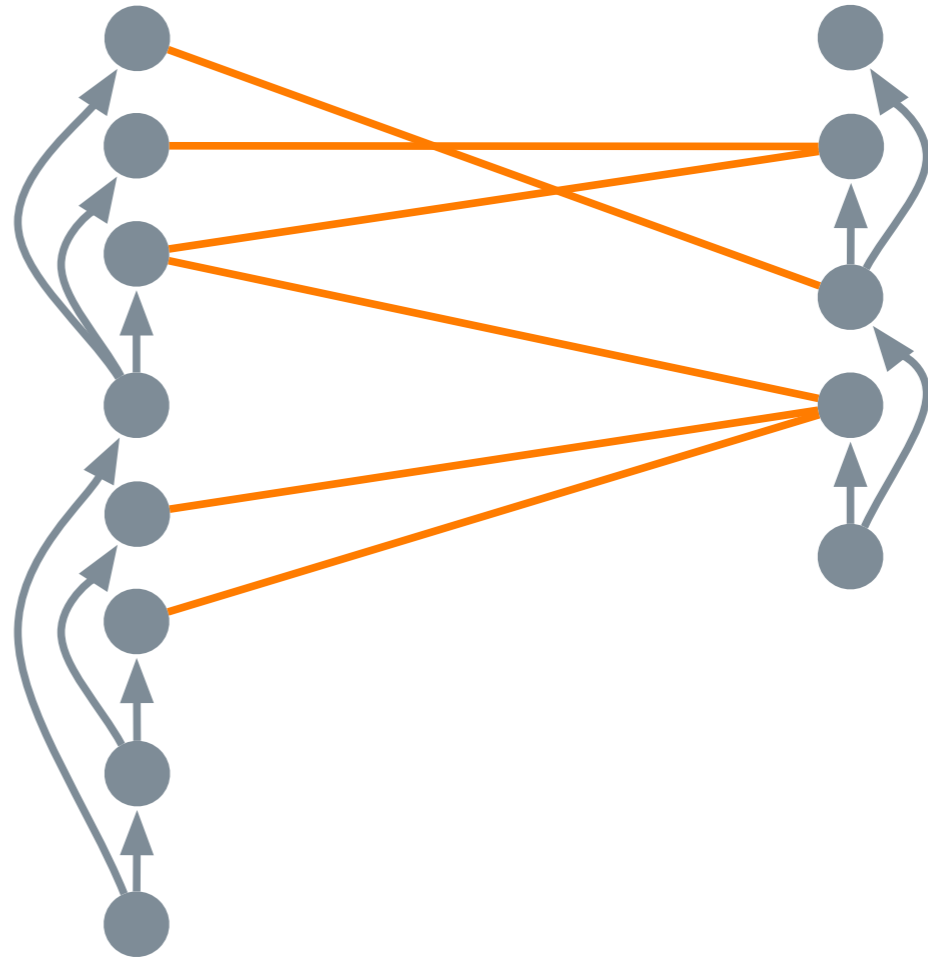
Knotenordnung

In jedem Baum wird eine **totale Ordnung** der Knoten erzeugt (DFS).



Knotenordnung

In jedem Baum wird eine **totale Ordnung** der Knoten erzeugt (DFS).



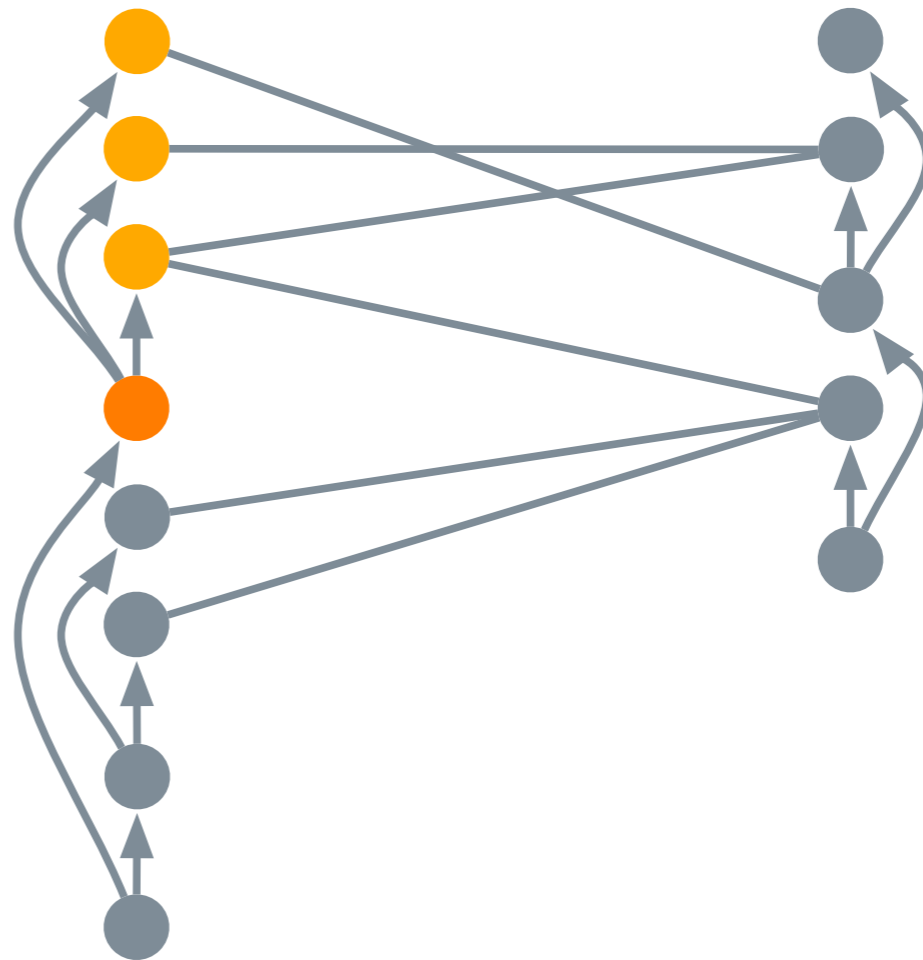
Knotenordnung

In jedem Baum wird eine **totale Ordnung** der Knoten erzeugt (DFS).

Die Nachfolger eines Knotens bilden ein **zusammenhängendes Intervall**

$$[\min(v), \max(v)]$$

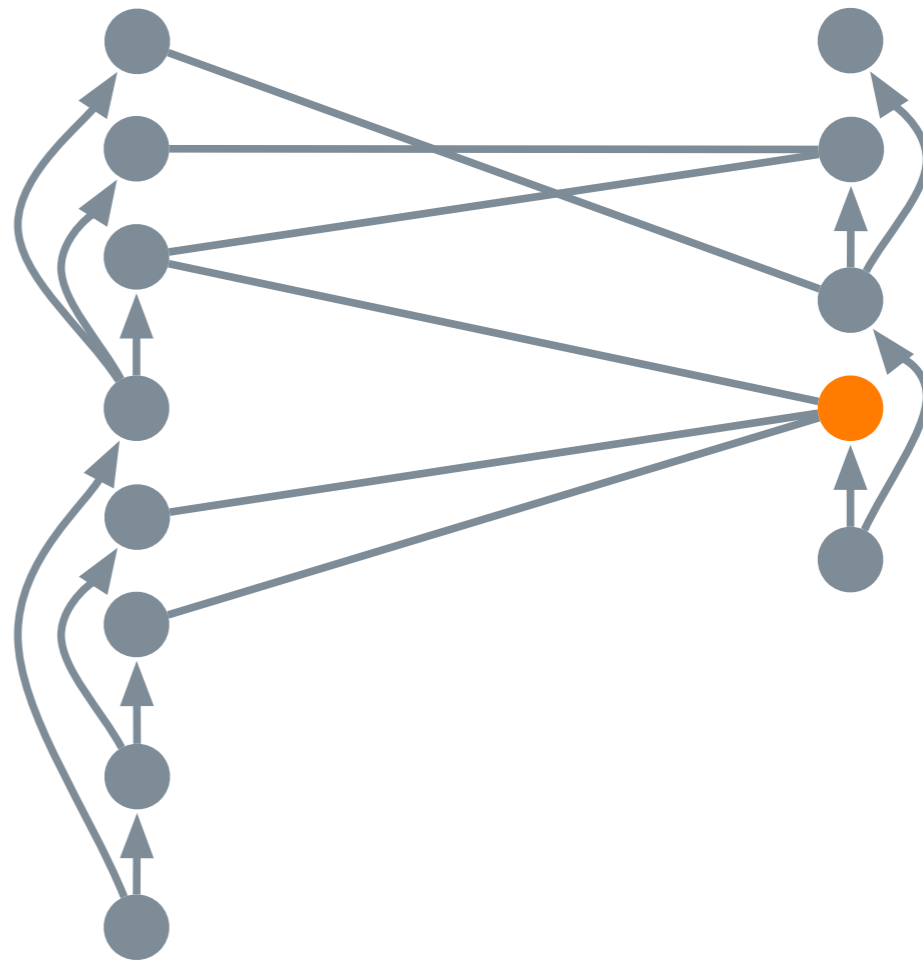
Die Ränder des Intervalls werden an jedem Knoten gespeichert.



Knotenordnung

[Buchsbaum et al. '00]
nummerieren Knoten mit
natürlichen Zahlen.

Hier: Ein neues Blatt
kann an beliebigen
Knoten angehängt
werden!

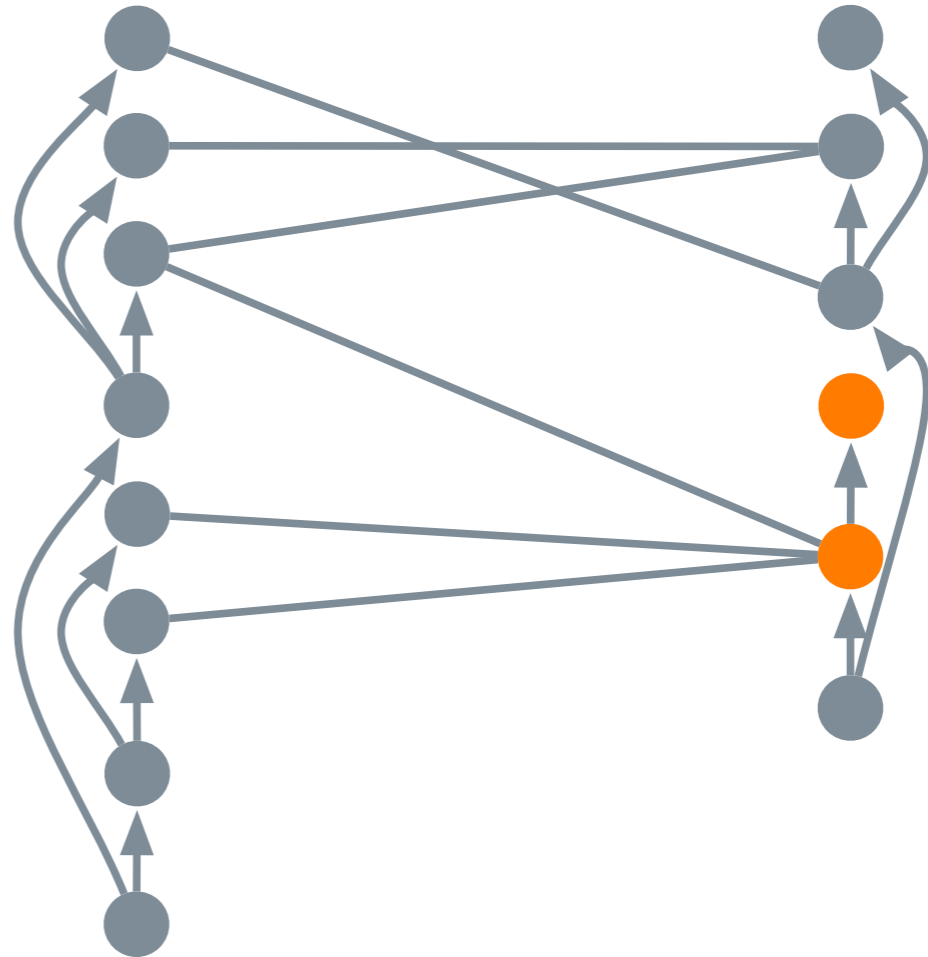


Knotenordnung

[Buchsbaum et al. '00]
nummerieren Knoten mit
natürlichen Zahlen.

Hier: Ein neues Blatt
kann an beliebigen
Knoten angehängt
werden!

Einfügen von neuen
Blättern an **beliebiger**
Position möglich.

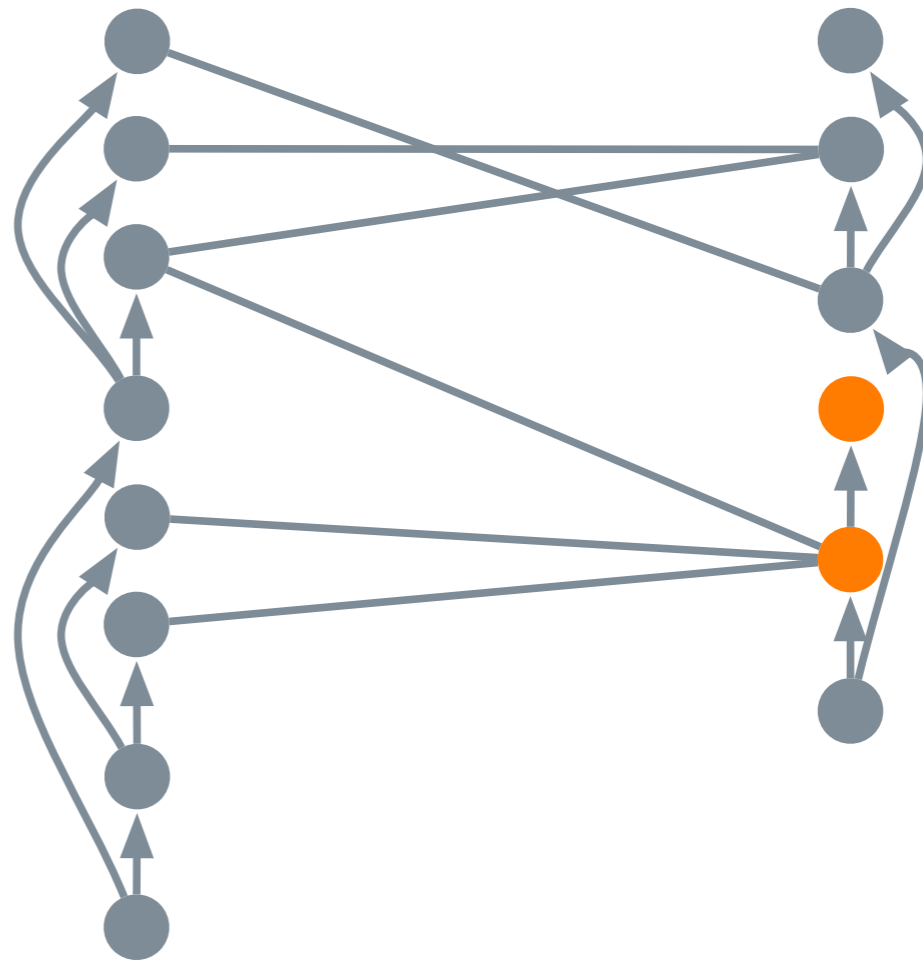


Knotenordnung

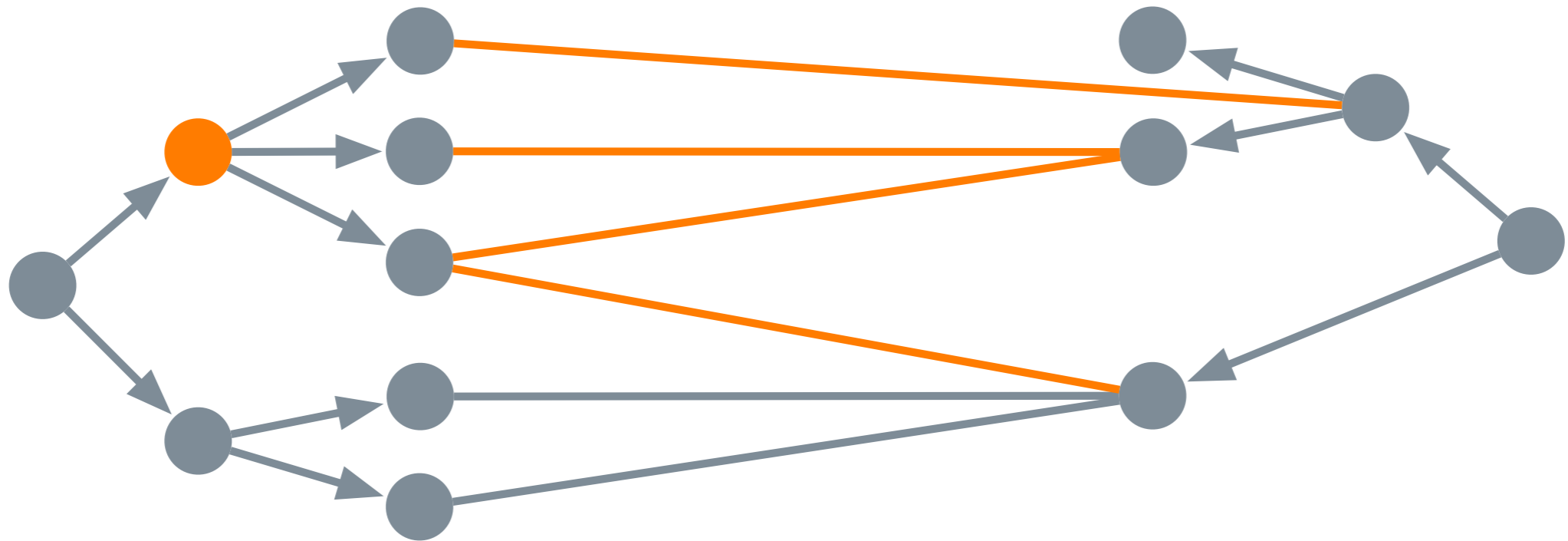
[Buchsbaum et al. '00]
nummerieren Knoten mit
natürlichen Zahlen.

Statt Nummerierung:
order maintenance
Datenstruktur

[Dietz and Sleator, 1987]

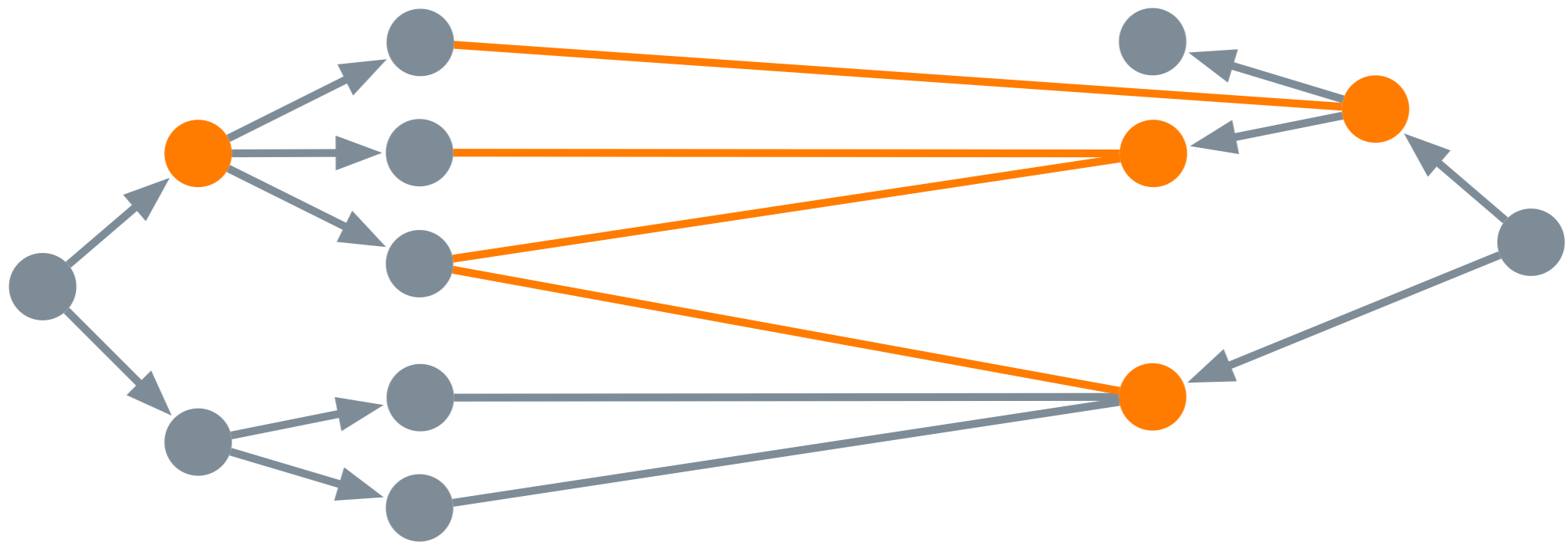


Datenstruktur



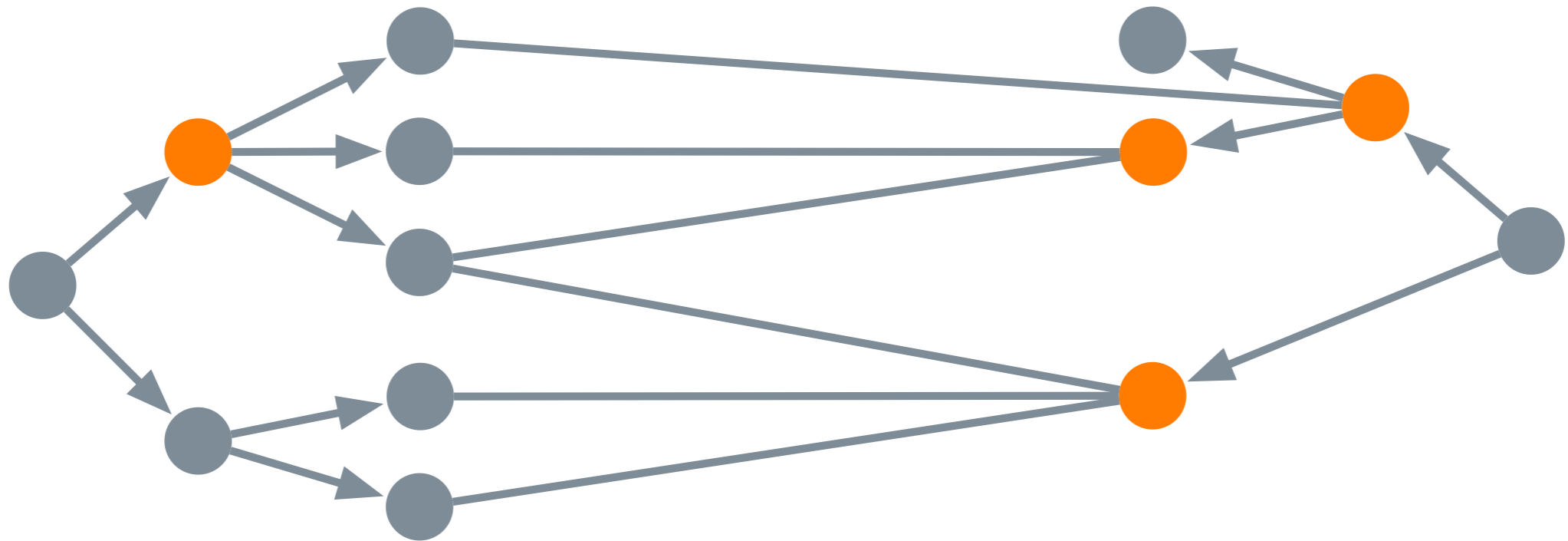
Idee: An jeden Knoten werden alle Kanten die inzident zu einem Nachfolger sind gespeichert.

Datenstruktur



Idee: An jeden Knoten werden alle Kanten die inzident zu einem Nachfolger sind gespeichert. Die **Endknoten im zweiten Baum** dienen als **Schlüssel** zur Verwaltung in einem **balancierten Suchbaum**.

Datenstruktur



$$S(u_1) = \pi_2(\{(u'_1, u_2) \in E \mid u'_1 \in \text{desc}(u_1)\}) \subseteq V(T_2)$$

bezeichne die **Menge der Schlüssel** im Suchbaum von Knoten

$$u_1 \in V(T_1)$$

Implementation edgeQuery

$$\text{succ}(S(u_1), u_2)$$

bestimmt das kleinste

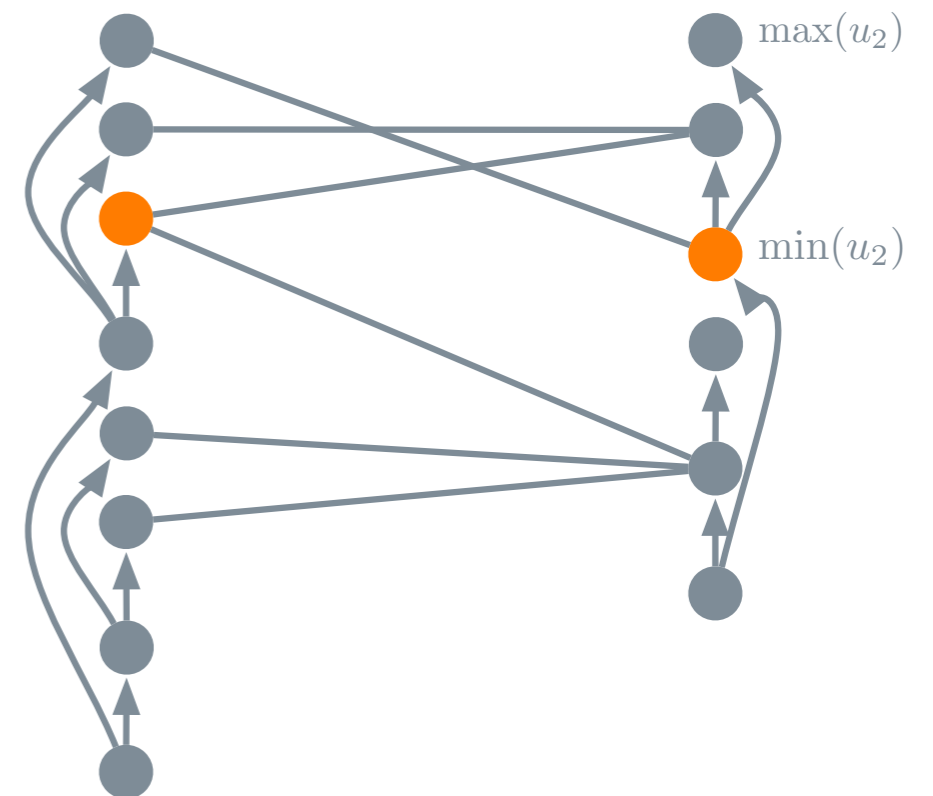
$$v \in S(u_1) : v \geq u_2$$

Daher ist

$$\text{succ}(S(u_1), \min(u_2)) \leq \max(u_2)$$

dann und nur dann **wenn es eine Kante zwischen Nachfolgern gibt**, d.h. wenn

$$\text{edgeQuery}(u_1, u_2) = \text{true}$$



Implementation edgeQuery

$$\text{succ}(S(u_1), u_2)$$

bestimmt das kleinste

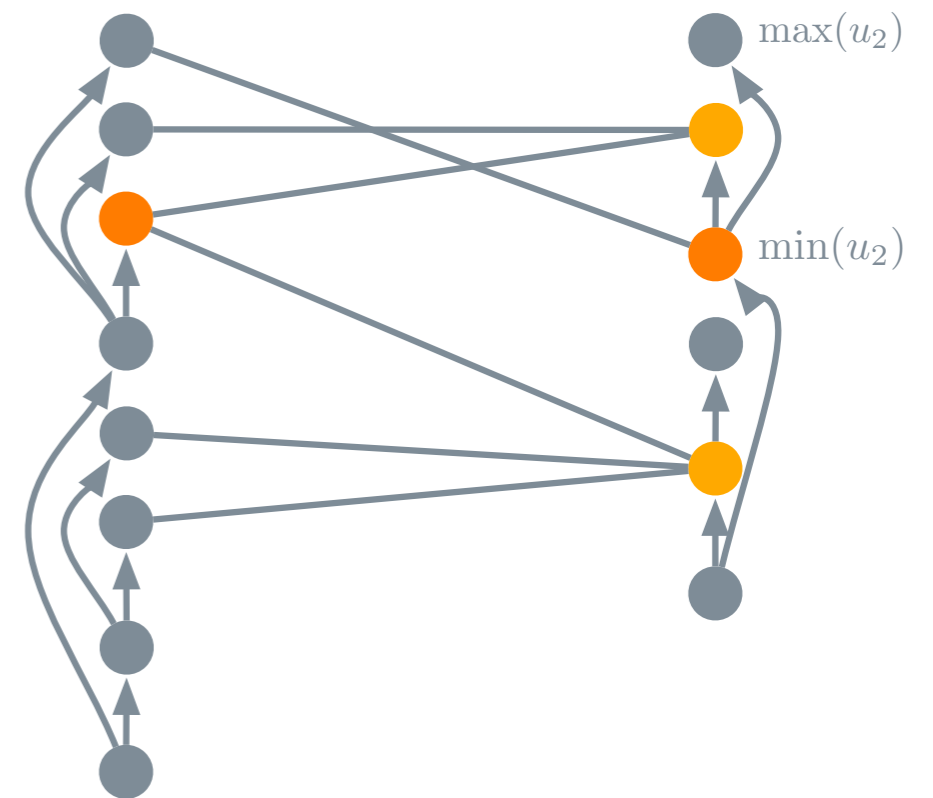
$$v \in S(u_1) : v \geq u_2$$

Daher ist

$$\text{succ}(S(u_1), \min(u_2)) \leq \max(u_2)$$

dann und nur dann **wenn es eine Kante zwischen Nachfolgern gibt**, d.h. wenn

$$\text{edgeQuery}(u_1, u_2) = \text{true}$$



Implementation edgeQuery

$$\text{succ}(S(u_1), u_2)$$

bestimmt das kleinste

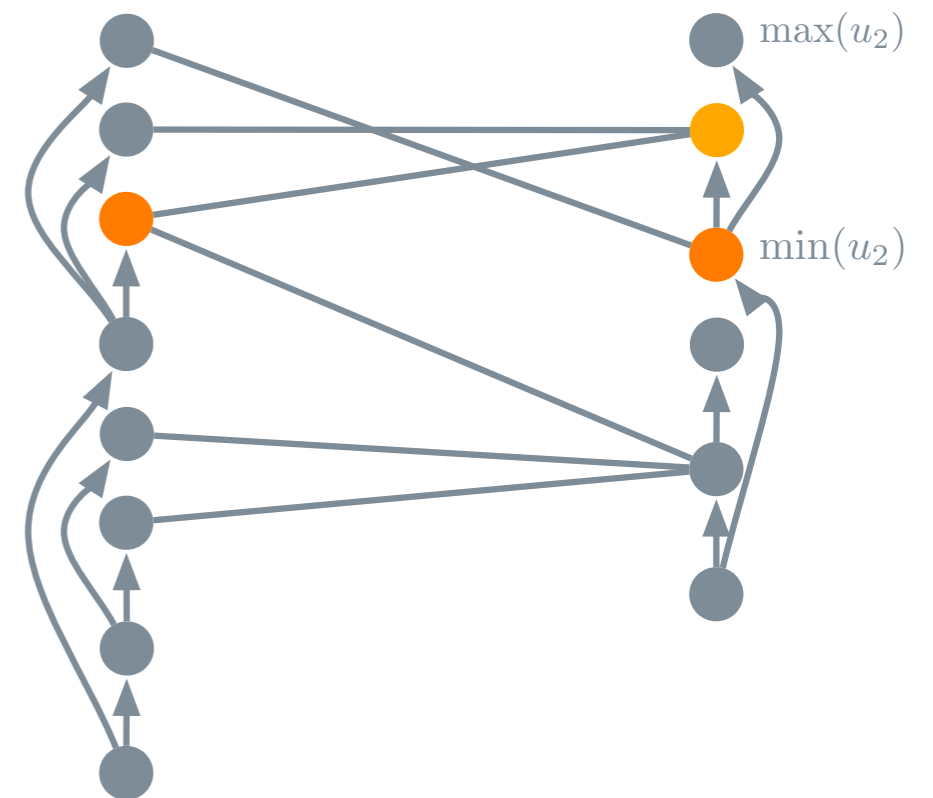
$$v \in S(u_1) : v \geq u_2$$

Daher ist

$$\text{succ}(S(u_1), \min(u_2)) \leq \max(u_2)$$

dann und nur dann **wenn es eine Kante zwischen Nachfolgern gibt**, d.h. wenn

$$\text{edgeQuery}(u_1, u_2) = \text{true}$$



Komplexität

$$\mathcal{O}(\log n)$$

Implementation edgeExpand

`edgeExpand((u1, u2), 1)`

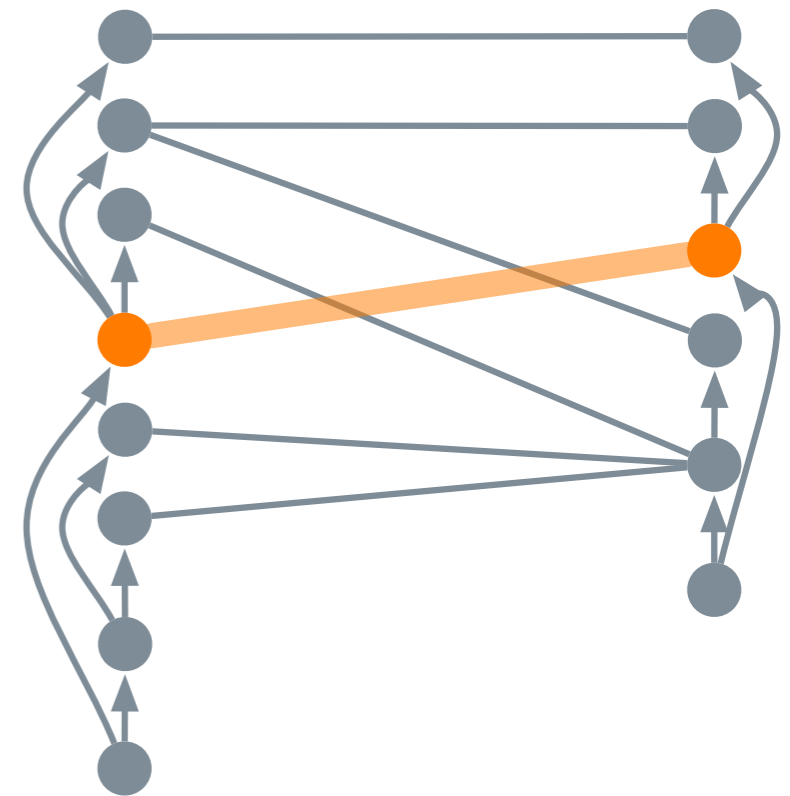
Frage: Welche Kinder

$\{v_1, \dots, v_l\} = \text{children}(u_1)$

erben die Kante?

Einfache Lösung:

`edgeQuery(vi, u2)`



Implementation edgeExpand

$\text{edgeExpand}((u_1, u_2), 1)$

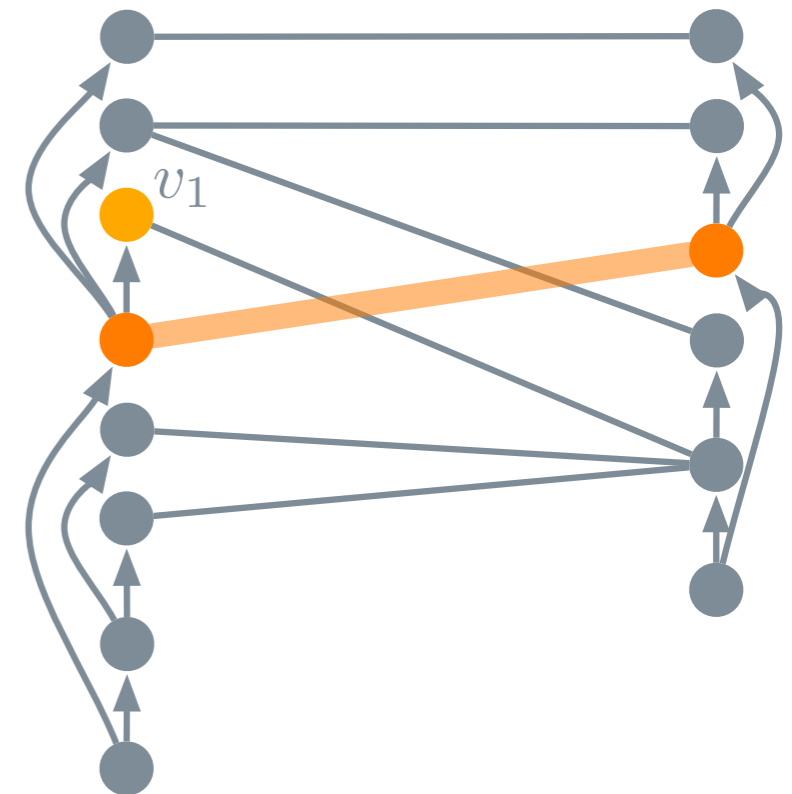
Frage: Welche Kinder

$\{v_1, \dots, v_l\} = \text{children}(u_1)$

erben die Kante?

Besser: Das erste Kind das die Kante erbt ist Vorfahre von

$t = \text{succ}(S(u_2), \min(v_1))$



Implementation edgeExpand

$\text{edgeExpand}((u_1, u_2), 1)$

Frage: Welche Kinder

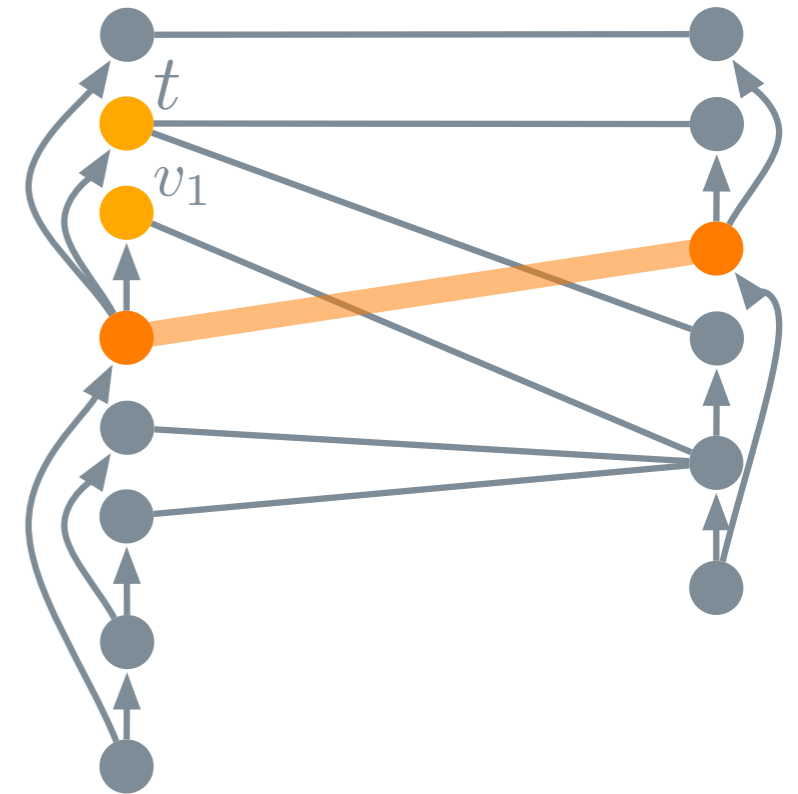
$\{v_1, \dots, v_l\} = \text{children}(u_1)$

erben die Kante?

Besser: Das erste Kind das die Kante erbt ist Vorfahre von

$t = \text{succ}(S(u_2), \min(v_1))$

Wiederhole das Verfahren mit nächst größerem Kind.



Implementation edgeExpand

$\text{edgeExpand}((u_1, u_2), 1)$

Frage: Welche Kinder

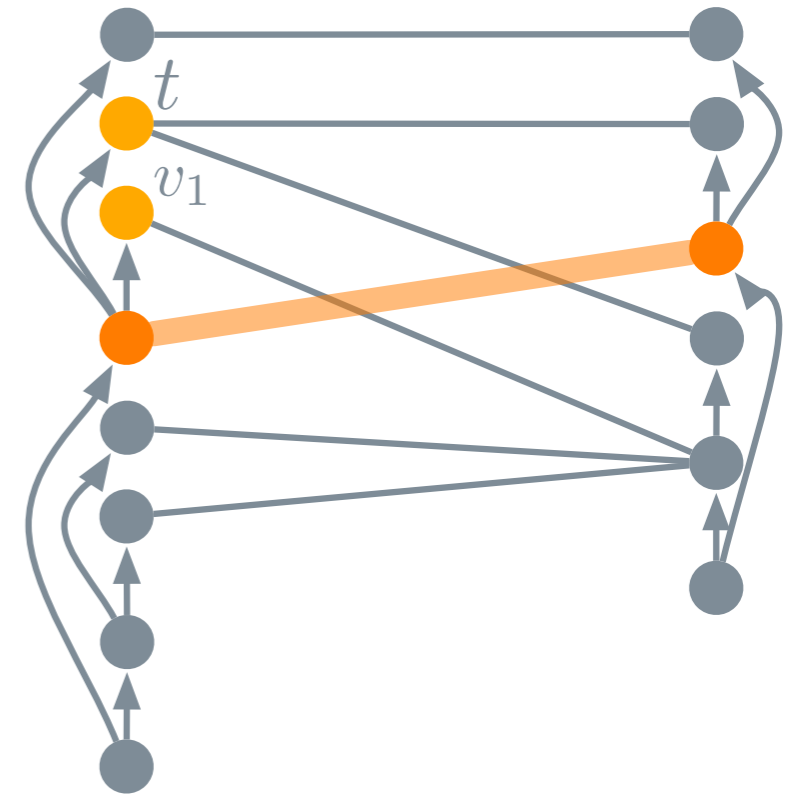
$\{v_1, \dots, v_l\} = \text{children}(u_1)$

erben die Kante?

Besser: Das erste Kind das die Kante erbt ist Vorfahre von

$t = \text{succ}(S(u_2), \min(v_1))$

Wiederhole das Verfahren mit nächst größerem Kind.

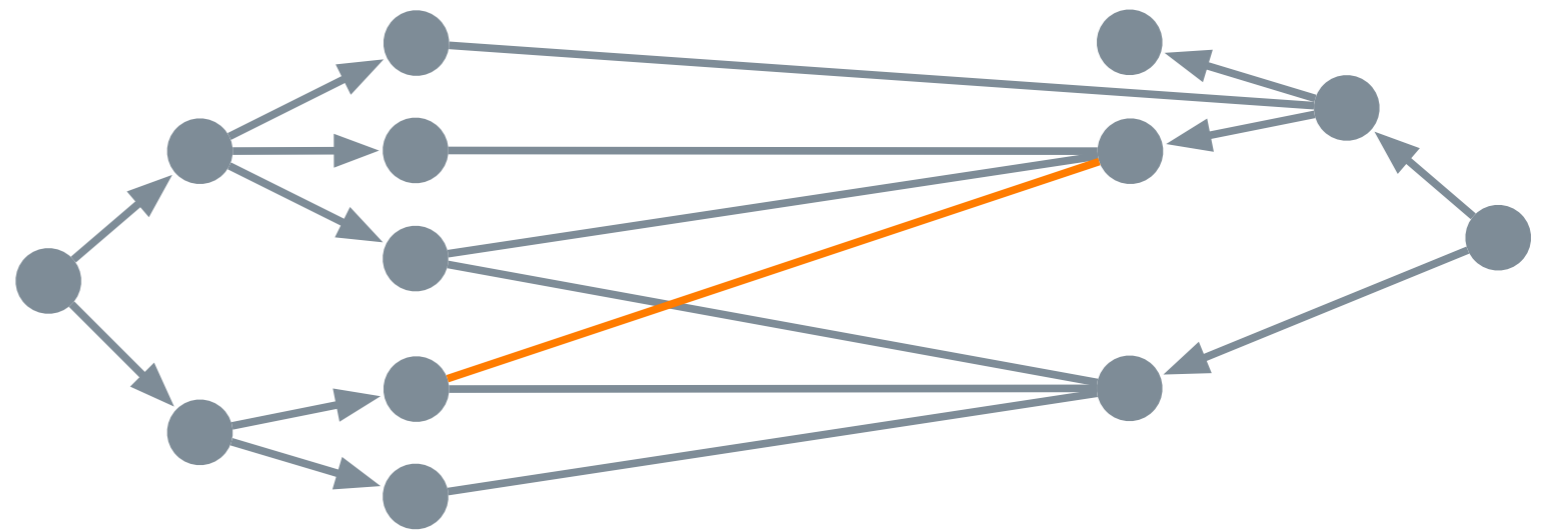


Komplexität

$\mathcal{O}(k \log n)$

Implementation Kanten

Anpassung der Suchbäume an **jedem Vorgänger** der beiden Endknoten.



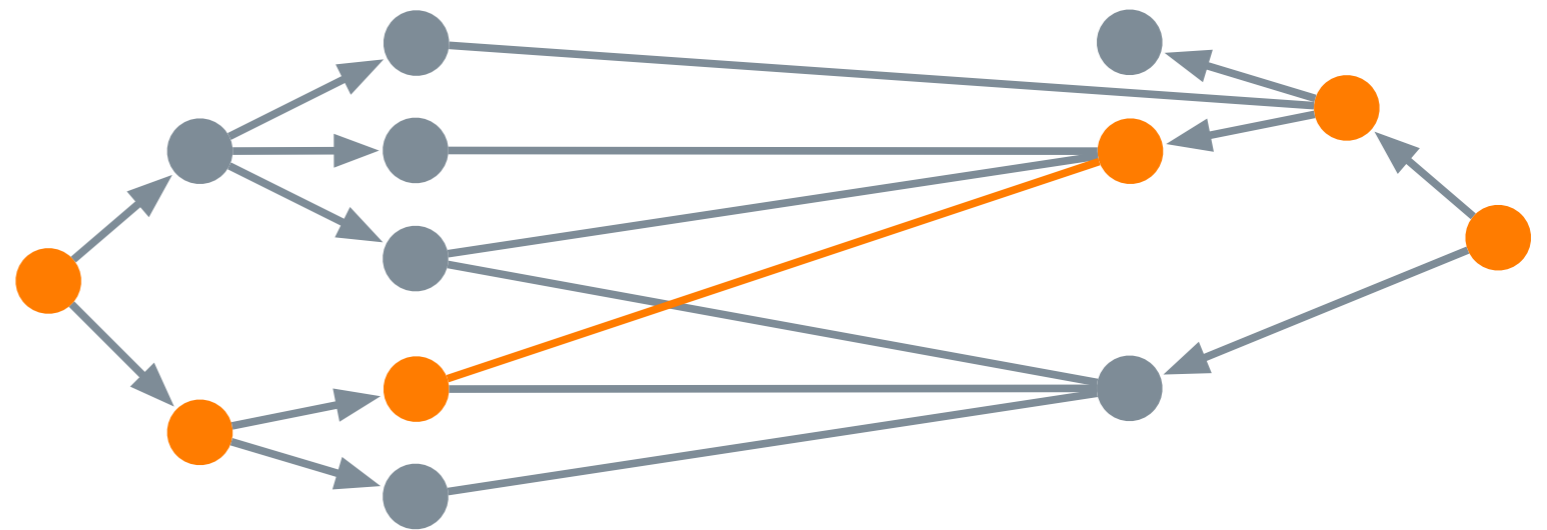
Implementation Kanten

Anpassung der Suchbäume an **jedem Vorgänger** der beiden Endknoten.

Komplexität

$$\mathcal{O}(D \log n)$$

$$D = \max_{i \in \{1,2\}} \text{depth}(T_i)$$



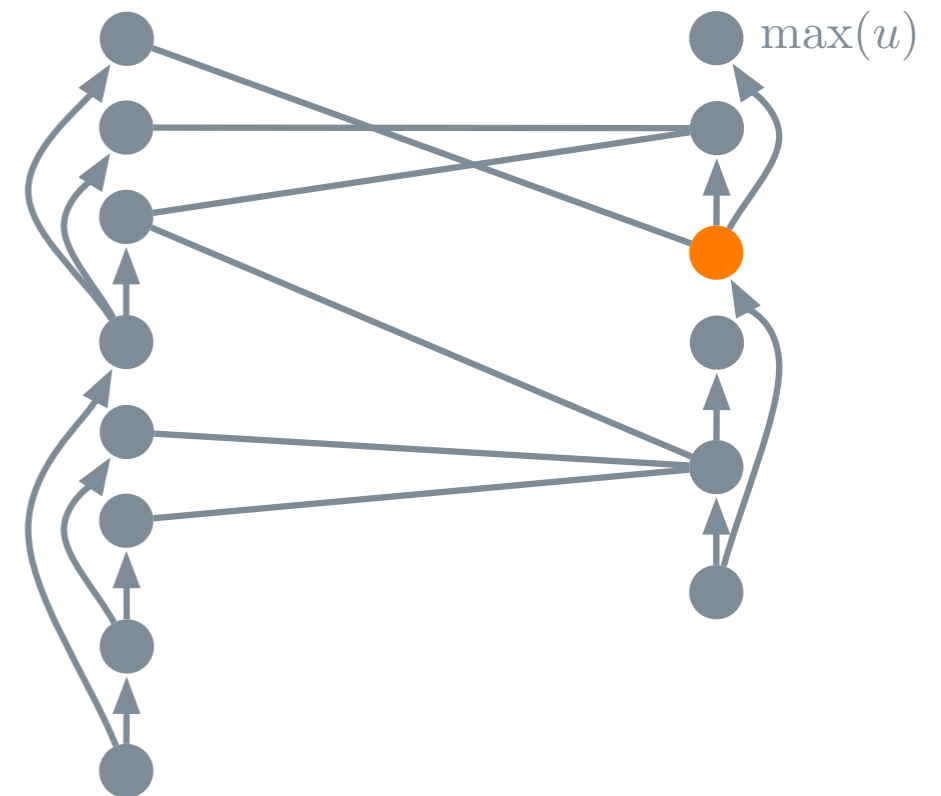
Implementation Blätter

$\text{newLeaf}(u)$

fügt das neue Blatt unmittelbar vor

$\text{max}(u)$

in die **order maintenance** Datenstruktur ein.



Implementation Blätter

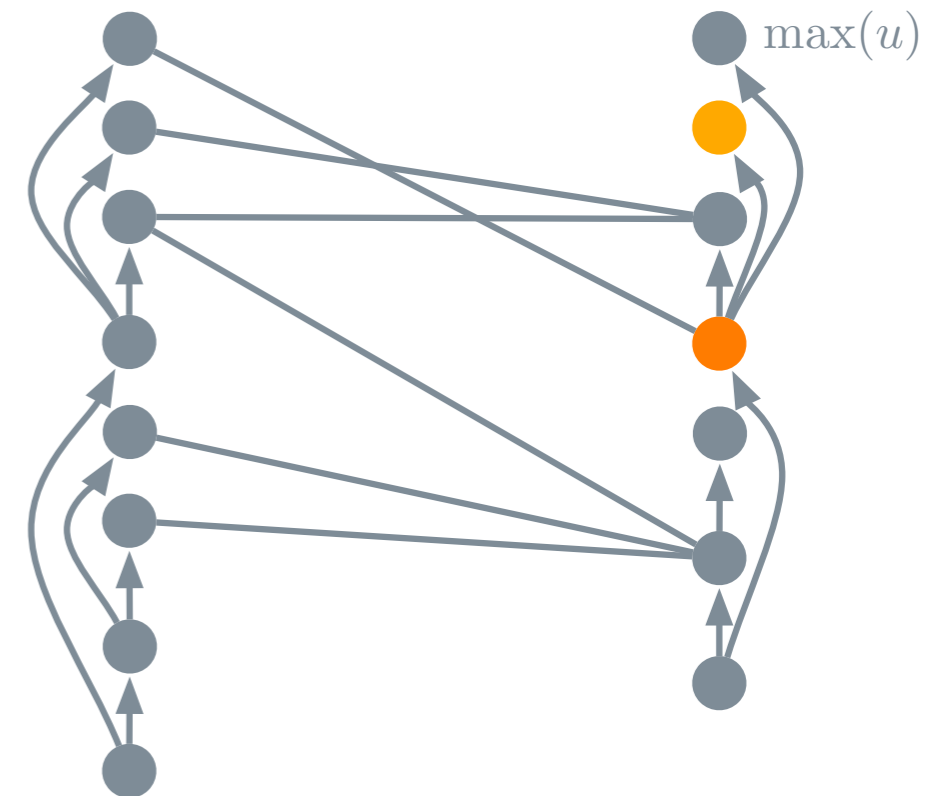
$\text{newLeaf}(u)$

fügt das neue Blatt unmittelbar vor

$\text{max}(u)$

in die **order maintenance** Datenstruktur ein.

min and **max** Werte an den Vorfahren müssen angepasst werden.



Komplexität

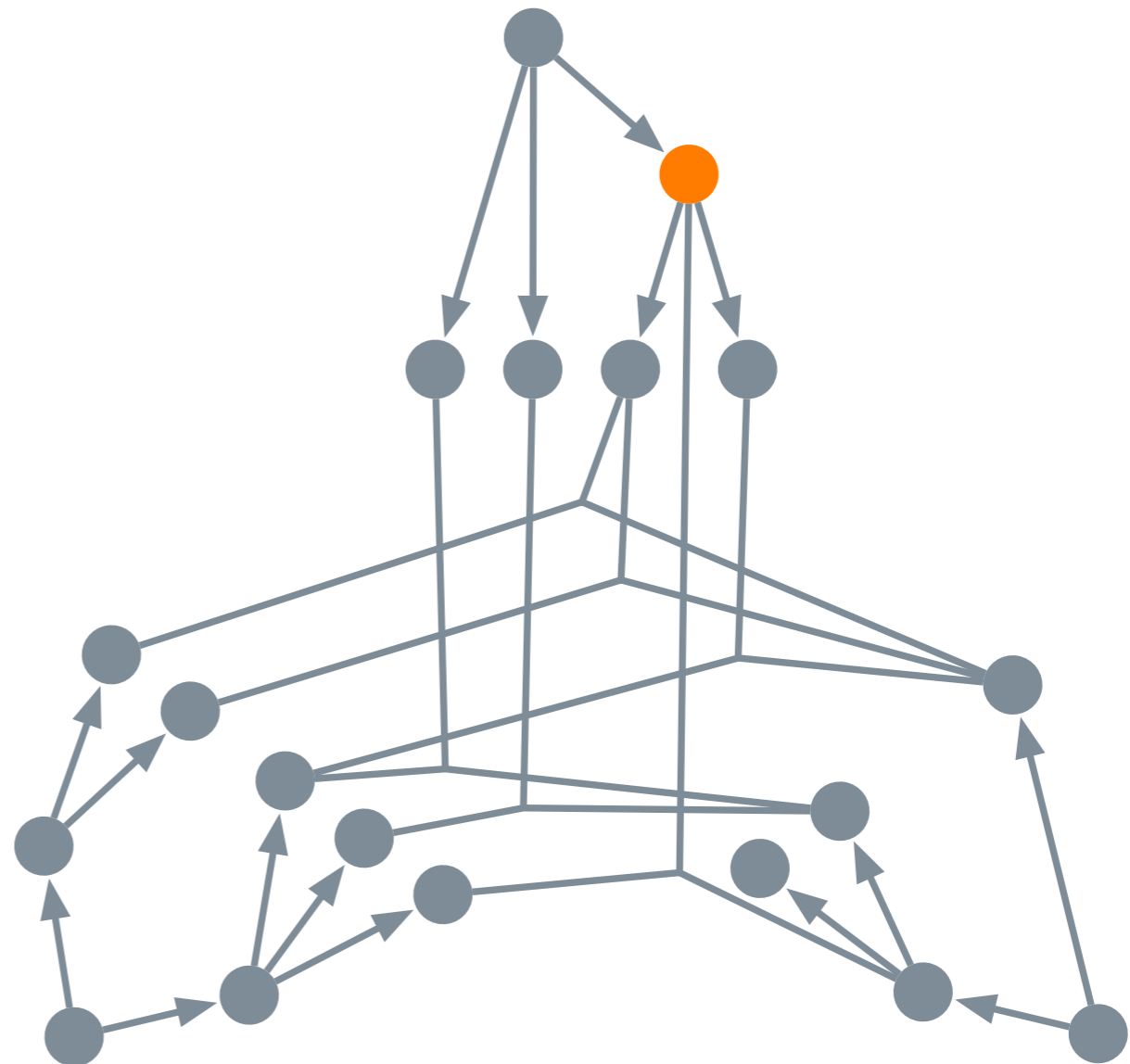
$\mathcal{O}(D)$

Höhere Dimensionen

Rekursive Definition

Basis: zweidimensionale Datenstruktur

An jedem Knoten des **ersten Baums** werden alle Kanten die inzident zu Nachfolgern sind betrachtet.

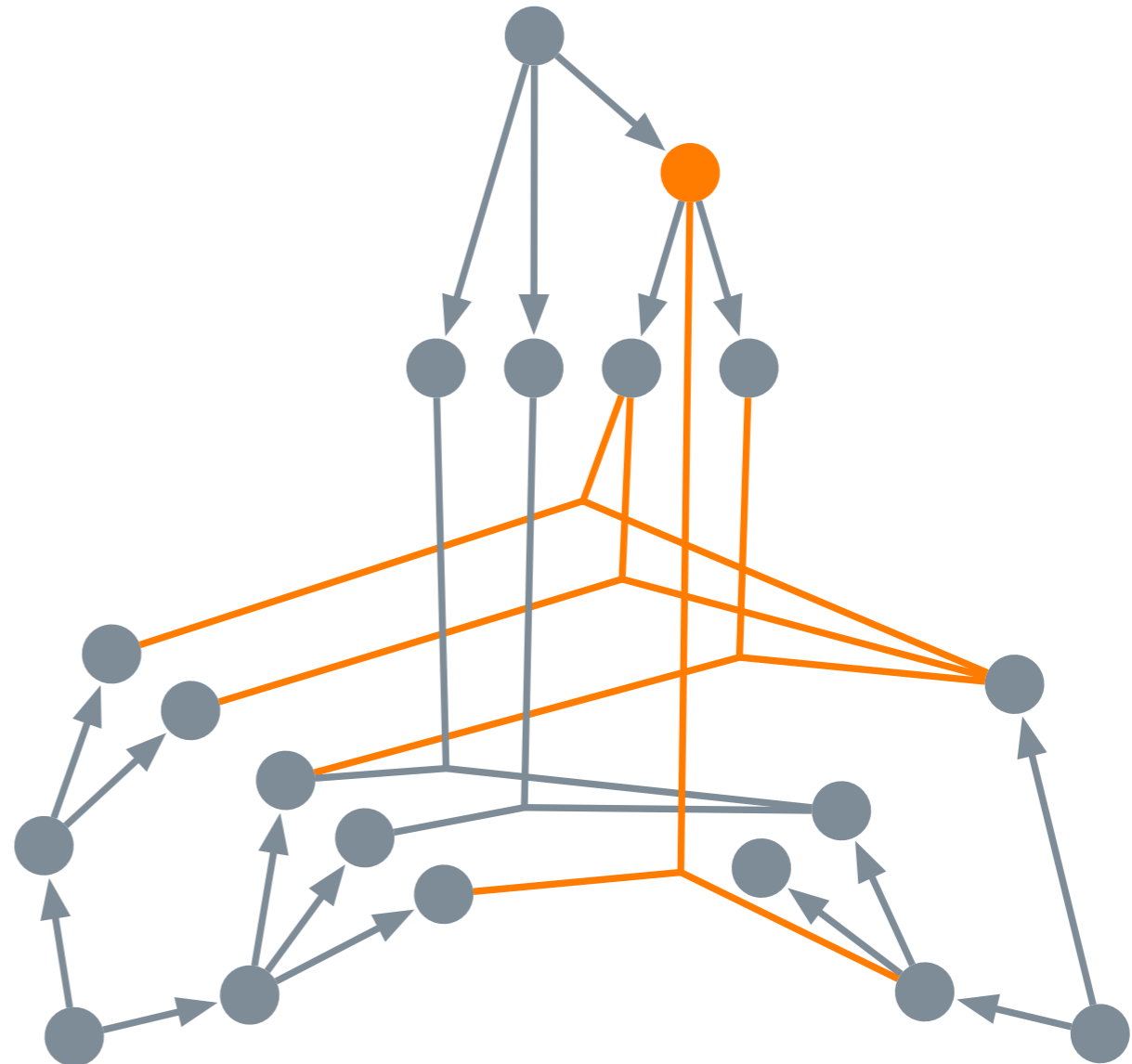


Höhere Dimensionen

Rekursive Definition

Basis: zweidimensionale Datenstruktur

An jedem Knoten des **ersten Baums** werden alle Kanten die inzident zu Nachfolgern sind betrachtet.



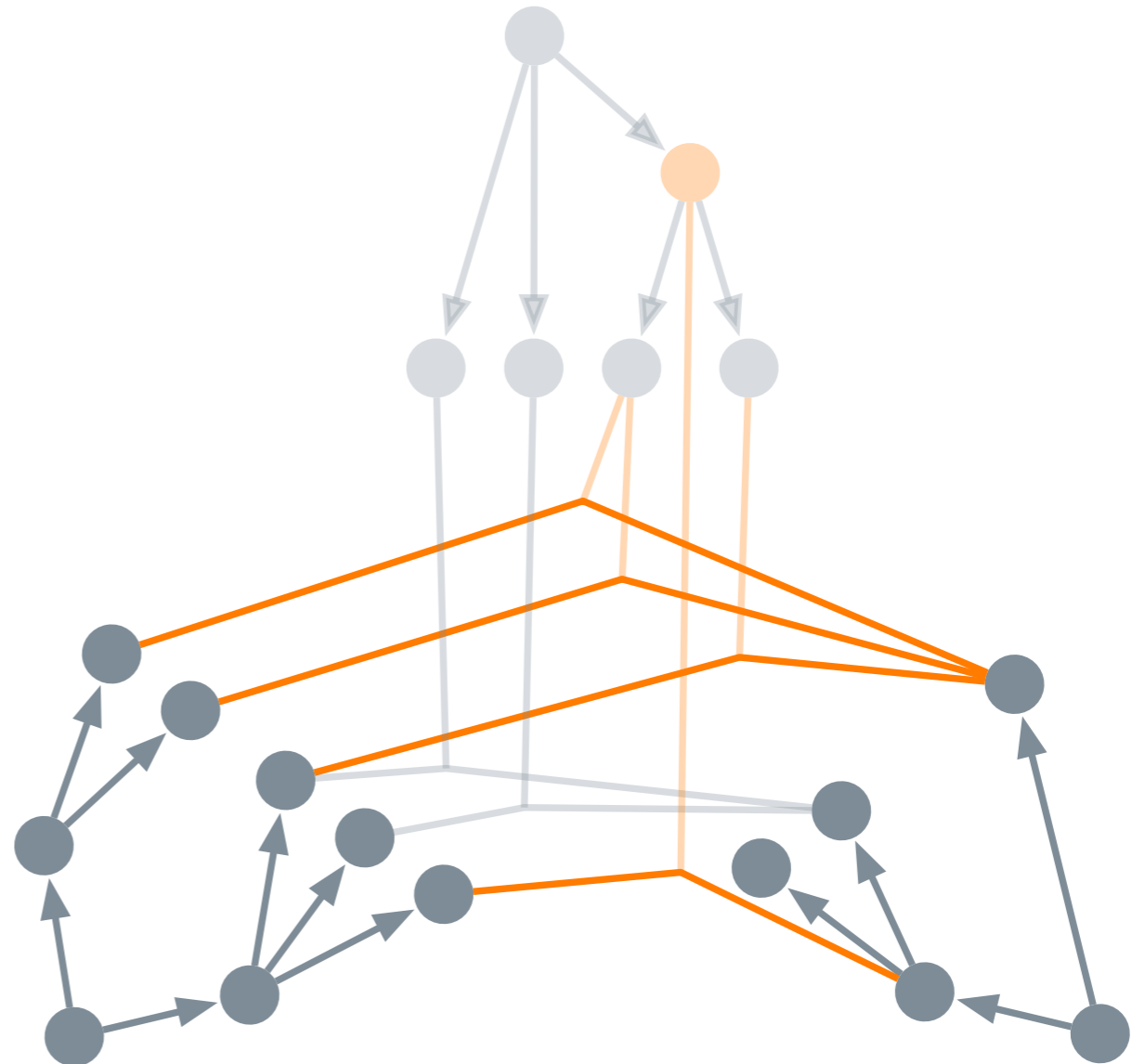
Höhere Dimensionen

Rekursive Definition

Basis: zweidimensionale Datenstruktur

An jedem Knoten des **ersten Baums** werden alle Kanten die inzident zu Nachfolgern sind betrachtet.

Und in einer Datenstruktur **eine Dimension kleiner** gespeichert.



	[Buchhsbaum et al. '00] (korrigiert)	Hier
edgeQuery(u)	$\mathcal{O}(d + \log \log n)$	$\mathcal{O}(d + \log n)$
edgeReport(u)	$\mathcal{O}(d + \log \log n + k)$	$\mathcal{O}(d + \log n + k)$
edgeExpand(u, j)	$\mathcal{O}(d + k \log \log n)$	$\mathcal{O}(d + k \log n)$
newEdge(u) deleteEdge(u)	$\mathcal{O}((2D)^{d-1} \log \log n)$	$\mathcal{O}((2D)^{d-1} \log n)$
newLeaf(u) deleteLeaf(u)	—	$\mathcal{O}(D)$

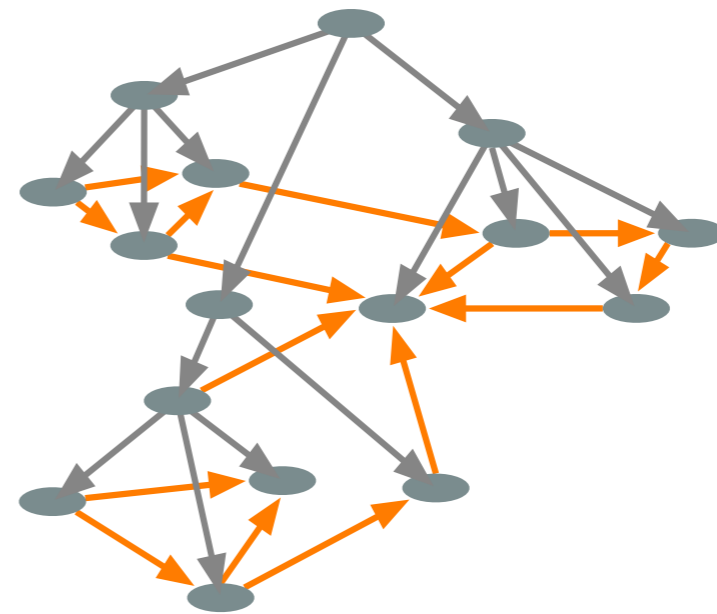
Anwendung

Graph View Maintenance

Modellierung

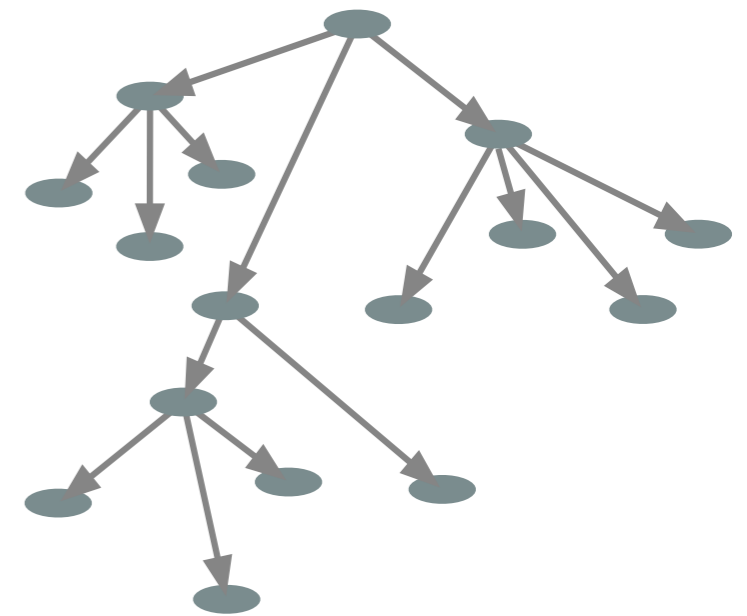
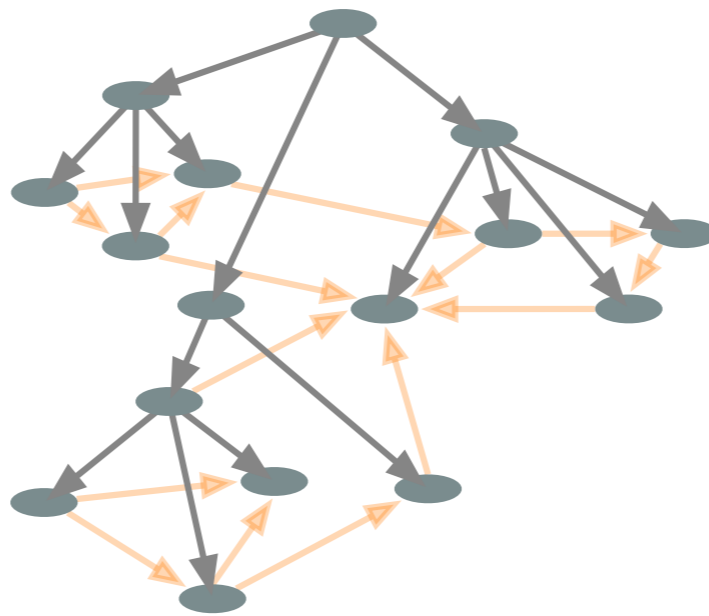
[Buchsbaum et. al '00]
Clustered Graph wird
modelliert als zwei-
dimensionales Baum-
Kreuzprodukt.

Hier: Erweiterung auf
Compound Graphs.



Modellierung

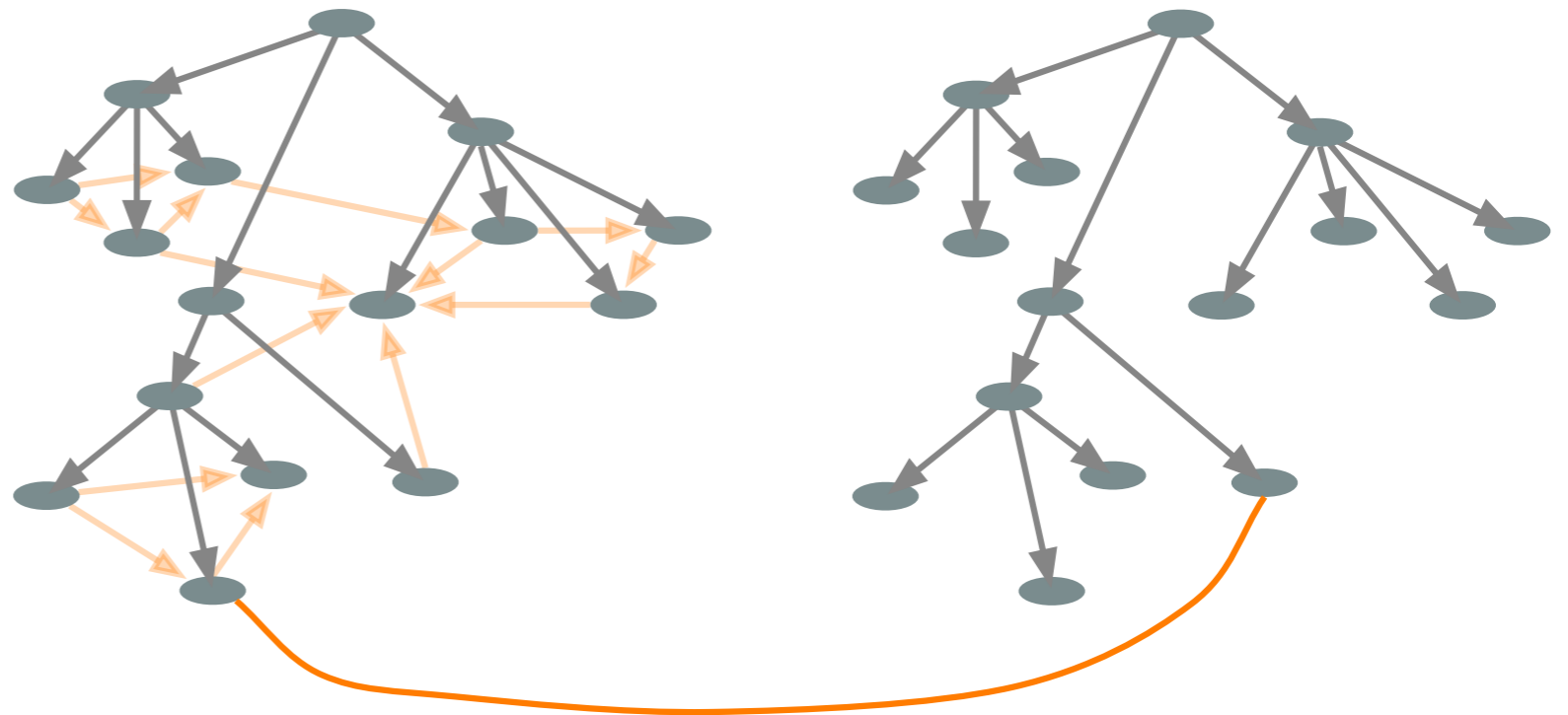
Inklusionsbaum wird
dupliziert



Modellierung

Inklusionsbaum wird dupliziert

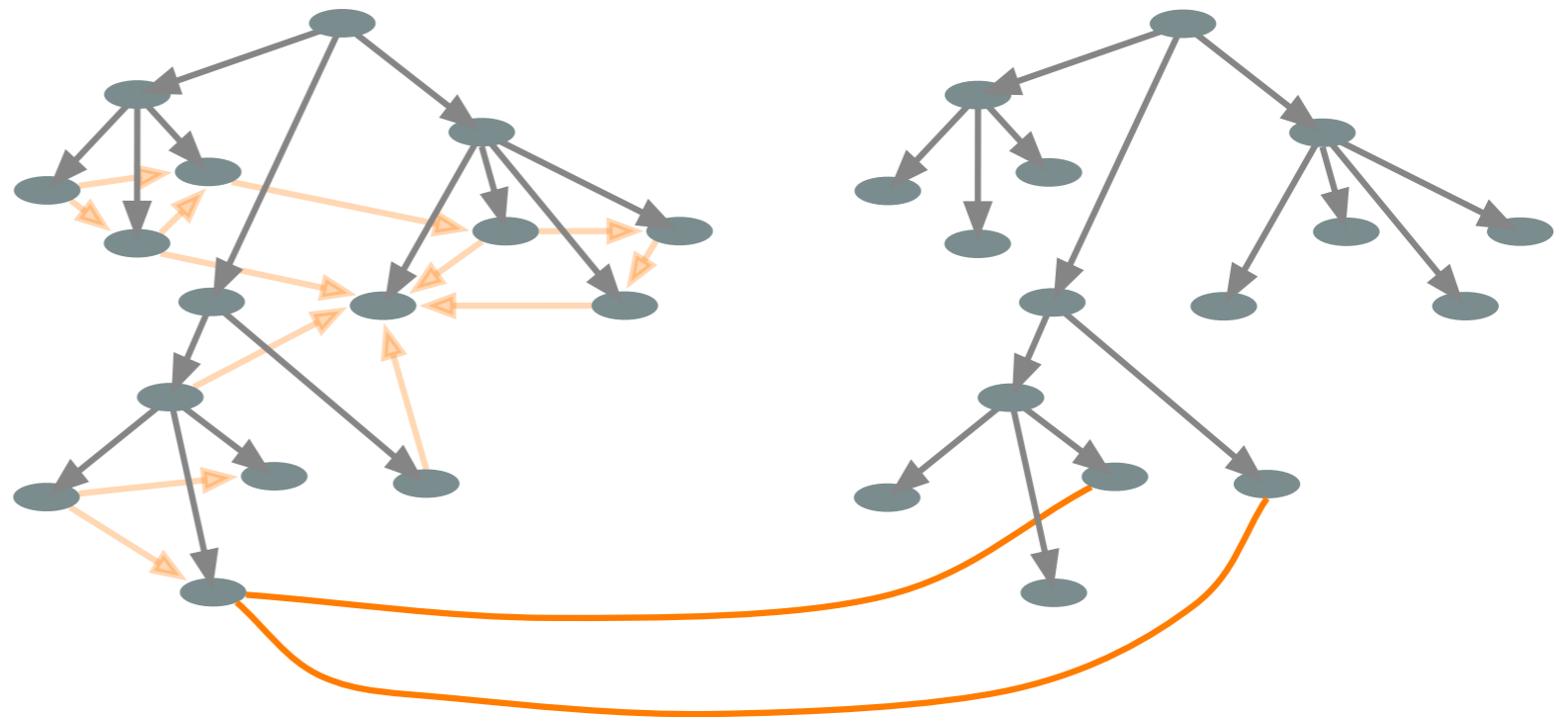
Adjazenzkanten werden zu Kanten des Baum-Kreuzprodukts.



Modellierung

Inklusionsbaum wird dupliziert

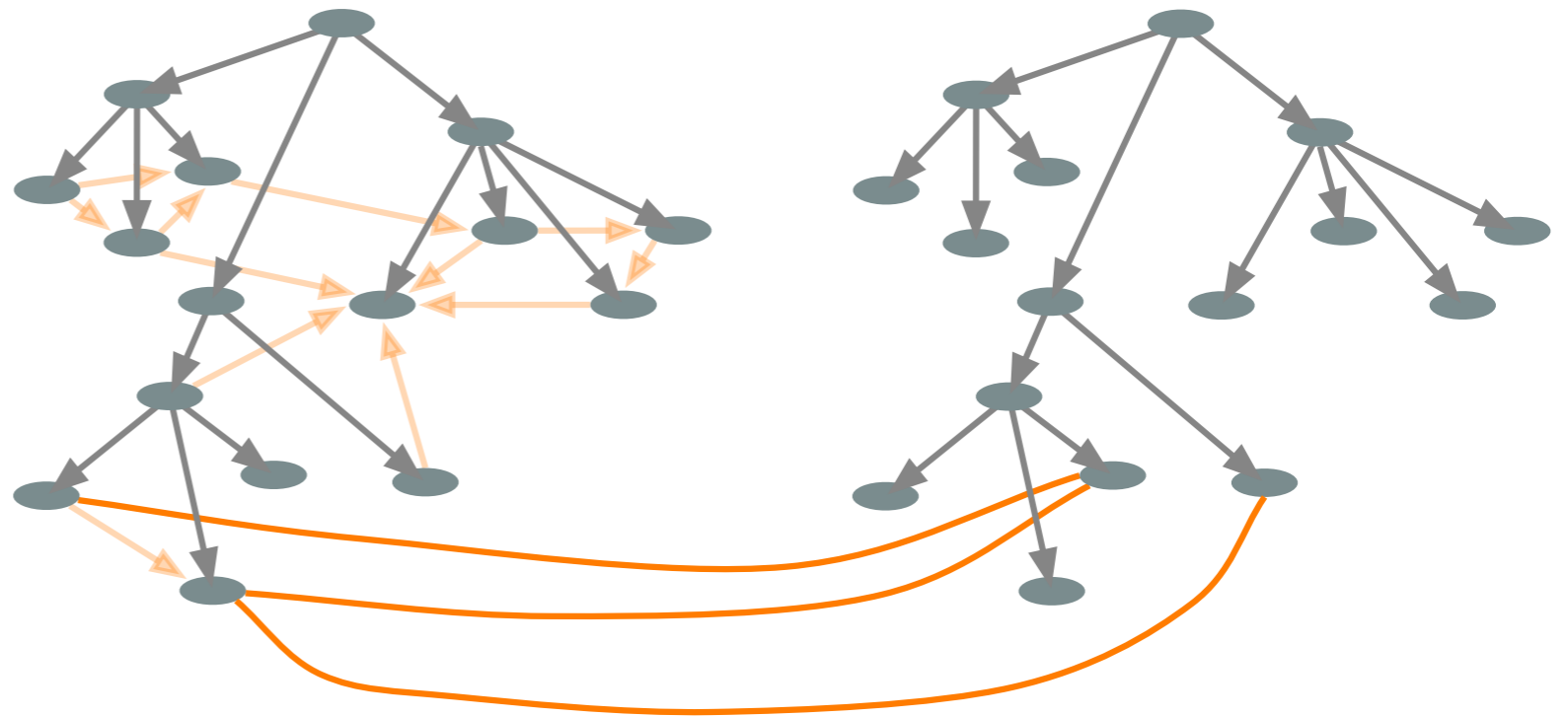
Adjazenzkanten werden zu Kanten des Baum-Kreuzprodukts.



Modellierung

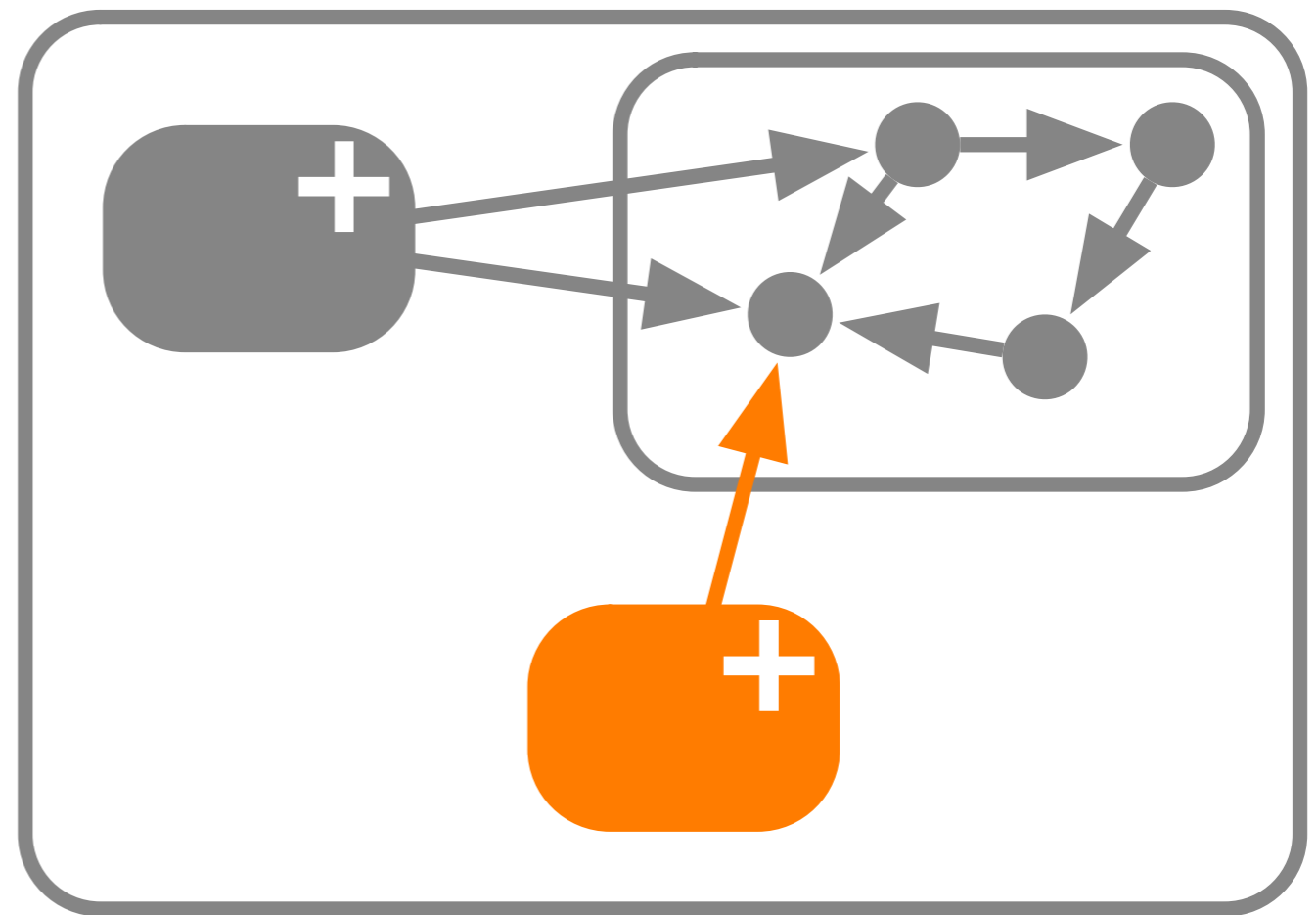
Inklusionsbaum wird dupliziert

Adjazenzkanten werden zu Kanten des Baum-Kreuzprodukts.



Expandieren

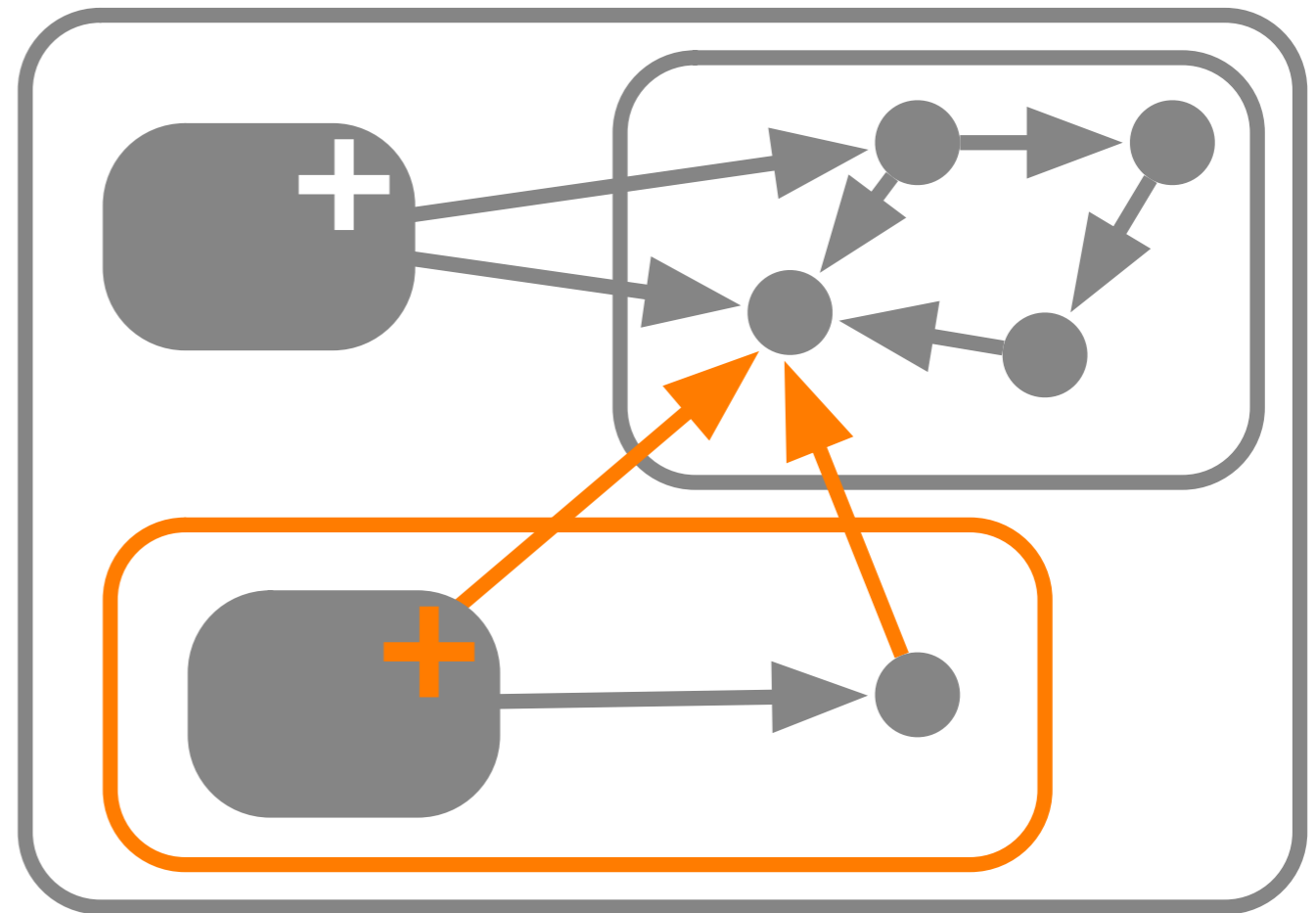
Idee: Expandiere alle inzidenten Kanten.



Expandieren

Idee: Expandiere alle inzidenten Kanten.

Abgeleitete Kanten zwischen zwei Kindern werden explizit gespeichert.



Restliche Operationen

Kontrahieren benötigt keine besonderen Datenstrukturen.

Einfügen und **Löschen von Kanten** wird vom Baum-Kreuzprodukt unterstützt.

Einfügen und **Löschen von Blättern**: immer in beiden Bäumen.

Fazit: Dynamischer als bisherige Lösungen

[Buchsbaum und Westbrook, '00] Buchsbaum, Westbrook. *Maintaining Hierarchical Graph Views*. Proc. 11th SODA, 2000.

[Buchsbaum et al., '00] Buchsbaum, Goodrich, Westbrook. *Range Searching over Tree Cross Products*. Proc. 8th ESA, 2000.

Visualisierung Hierarchische Zeichnungen

Marcus Raitner. **Visual Navigation of Compound Graphs.** In *Proc. 12th Intl. Symposium on Graph Drawing (GD)*, 2004

Hierarchische Zeichnungen

Klassisches Verfahren nach [Sugiyama et al. '81] für **gerichtete azyklische Graphen**

Vier Phasen:

- 🍷 Einteilung in Schichten
- 🍷 Normalisierung der Kanten
- 🍷 Kreuzungsreduktion
- 🍷 Koordinatenberechnung

Erweiterungen auf **Compound Graphen** von [Sugiyama und Misue '91] oder [Sander '96].

Mental Map

Ziel: Schnelles Wiedererkennen der Zeichnung eines lediglich **lokal geänderten** Graphen.

Kriterien:

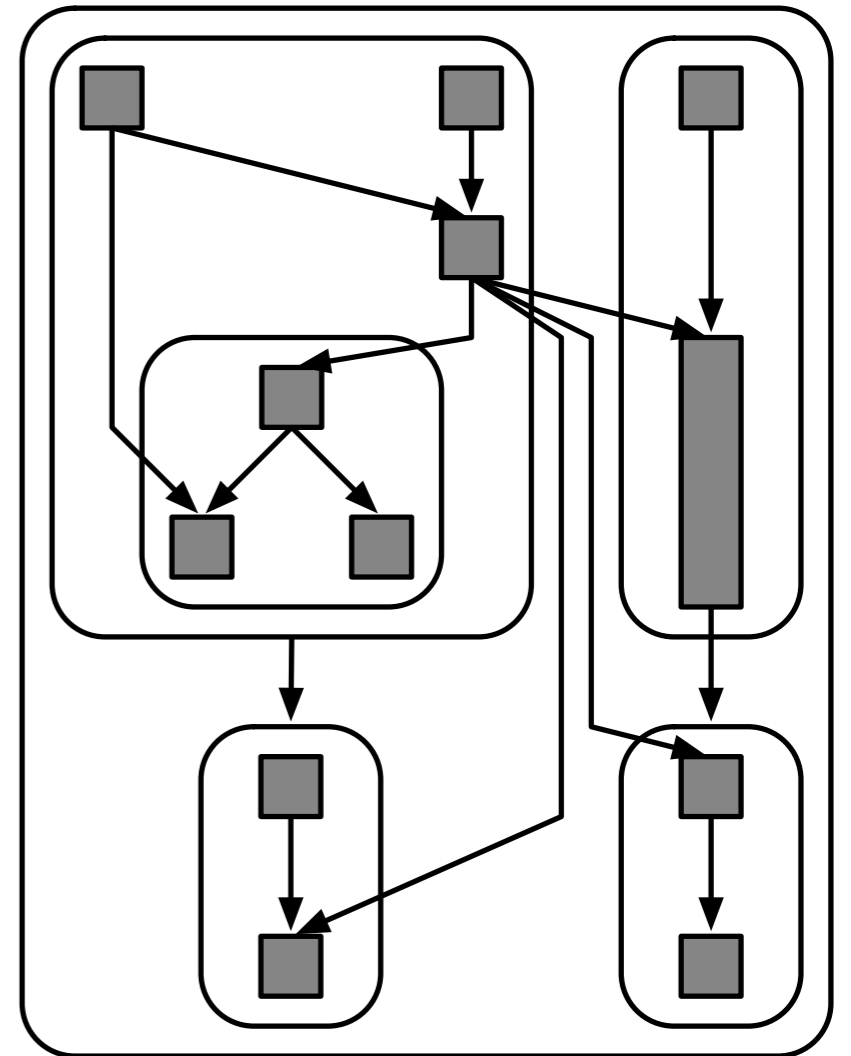
- ✦ Expandieren und Kontrahieren sind visuell invers.
- ✦ Alle nicht betroffenen Knoten bleiben in ihrer Schicht.
- ✦ Je zwei nicht betroffene Knoten in derselben Schicht behalten ihre Reihenfolge bei.
- ✦ Expandierte Kanten verlaufen wie die zugehörige kontrahierte Kante.

Update-Schema

Basis: [Sugiyama und Misue '91]



Lokale Updates der
Zwischenergebnisse der
einzelnen Phasen.

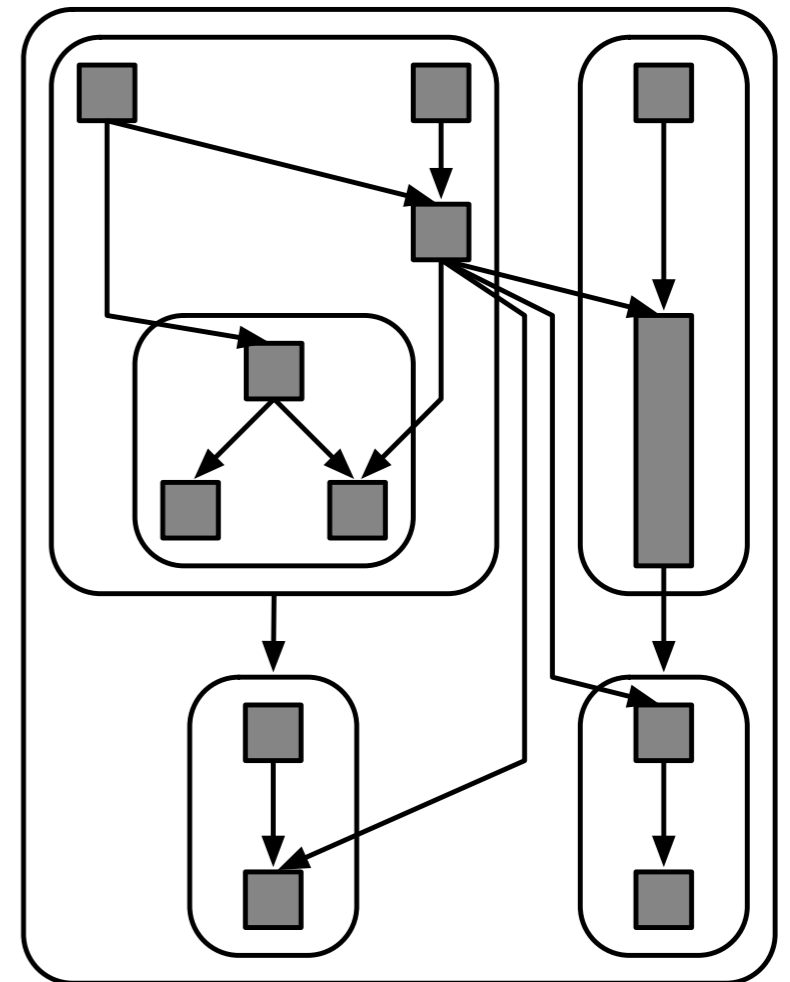


Erste Phase

Einteilung in Schichten

Struktur der Schichten

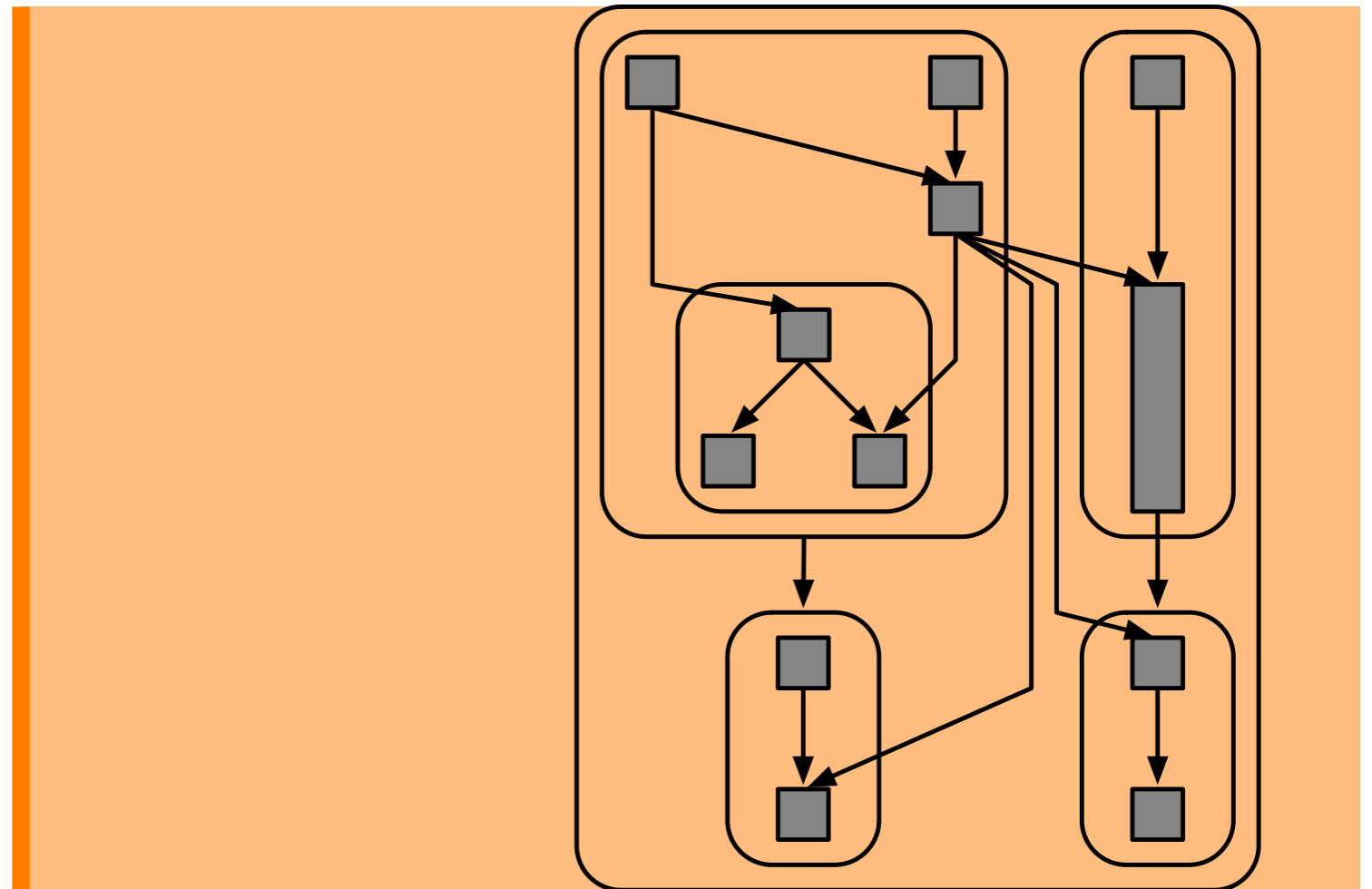
Rekursiv verschachtelt



Struktur der Schichten

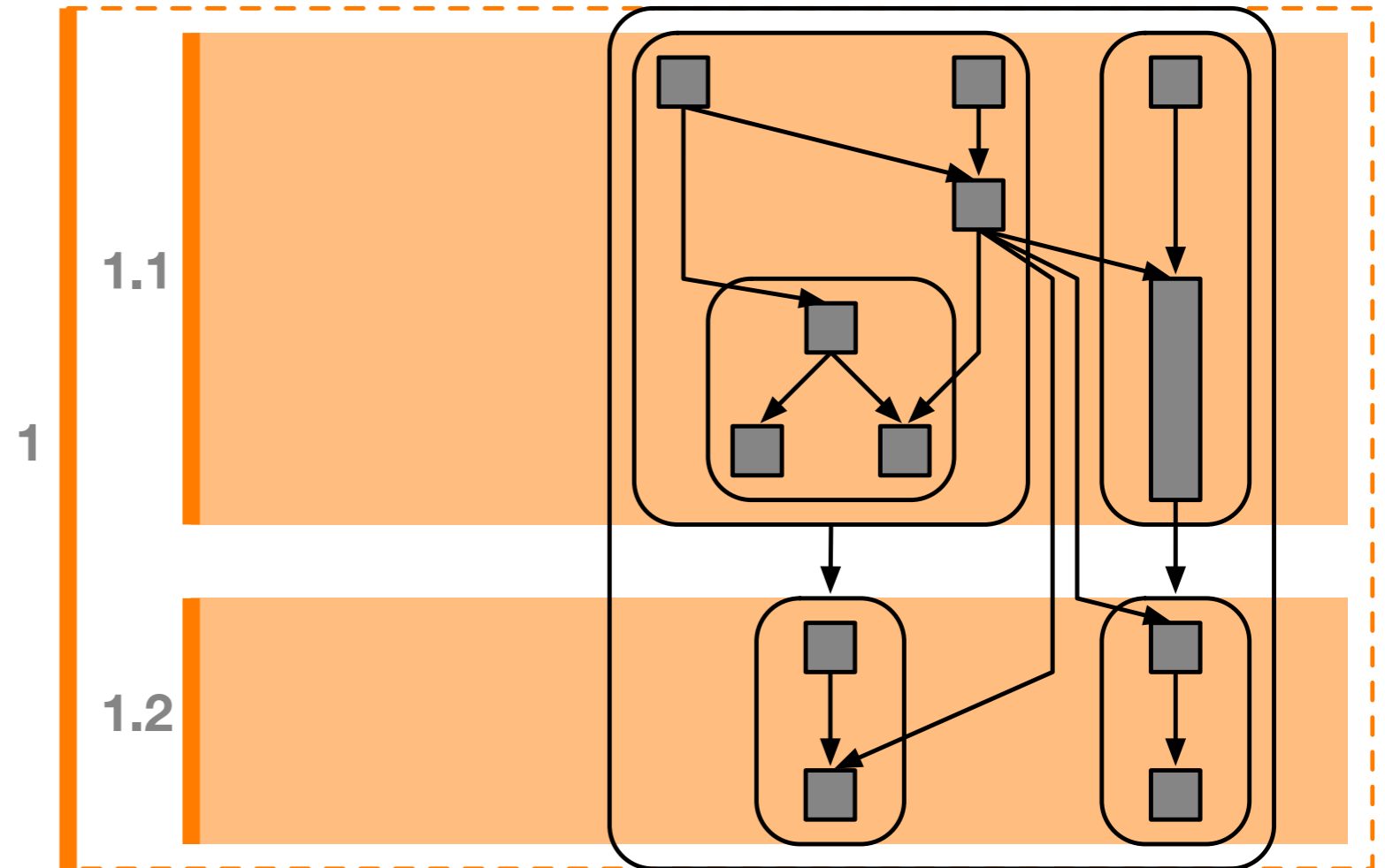
Rekursiv verschachtelt

1



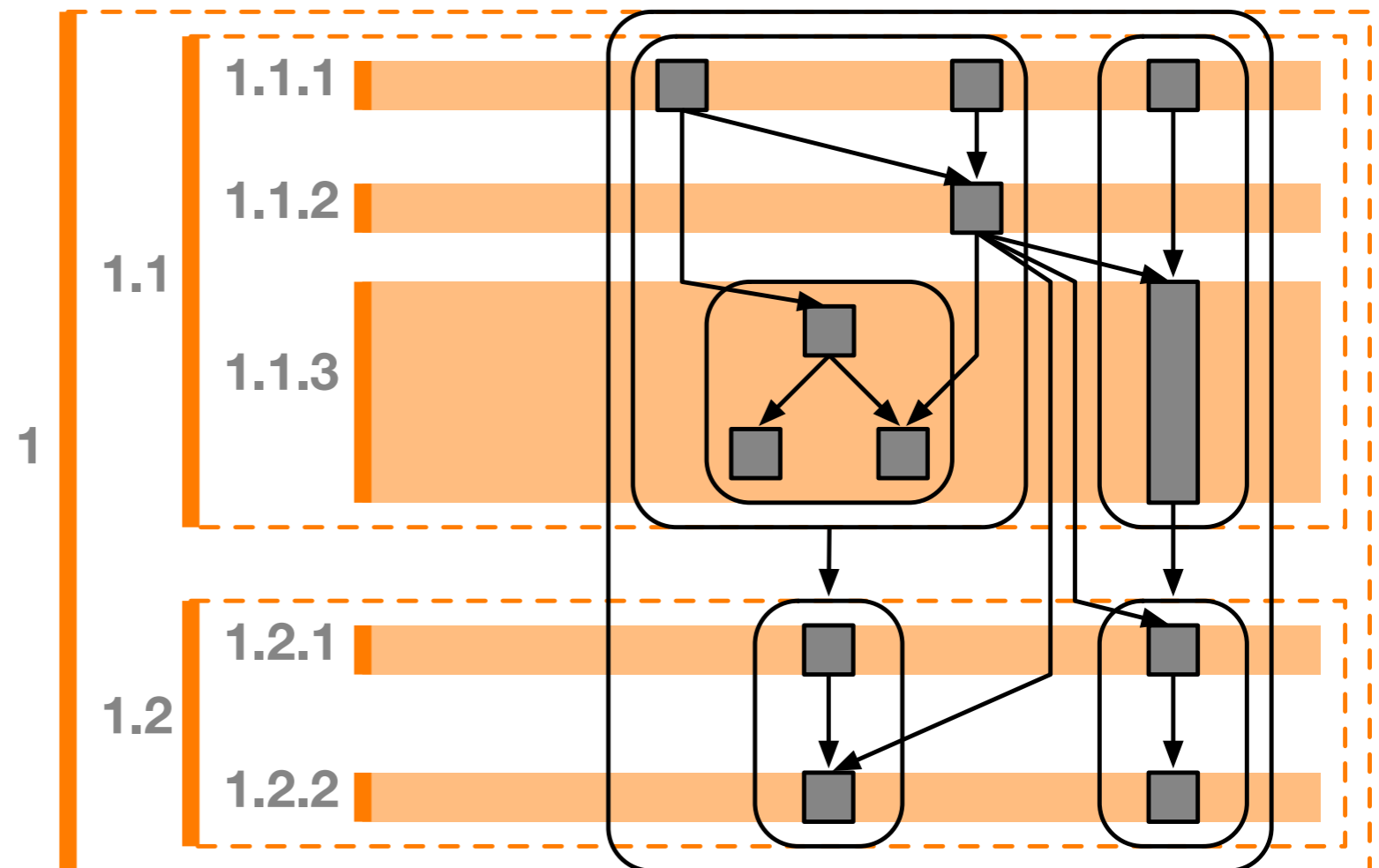
Struktur der Schichten

Rekursiv verschachtelt



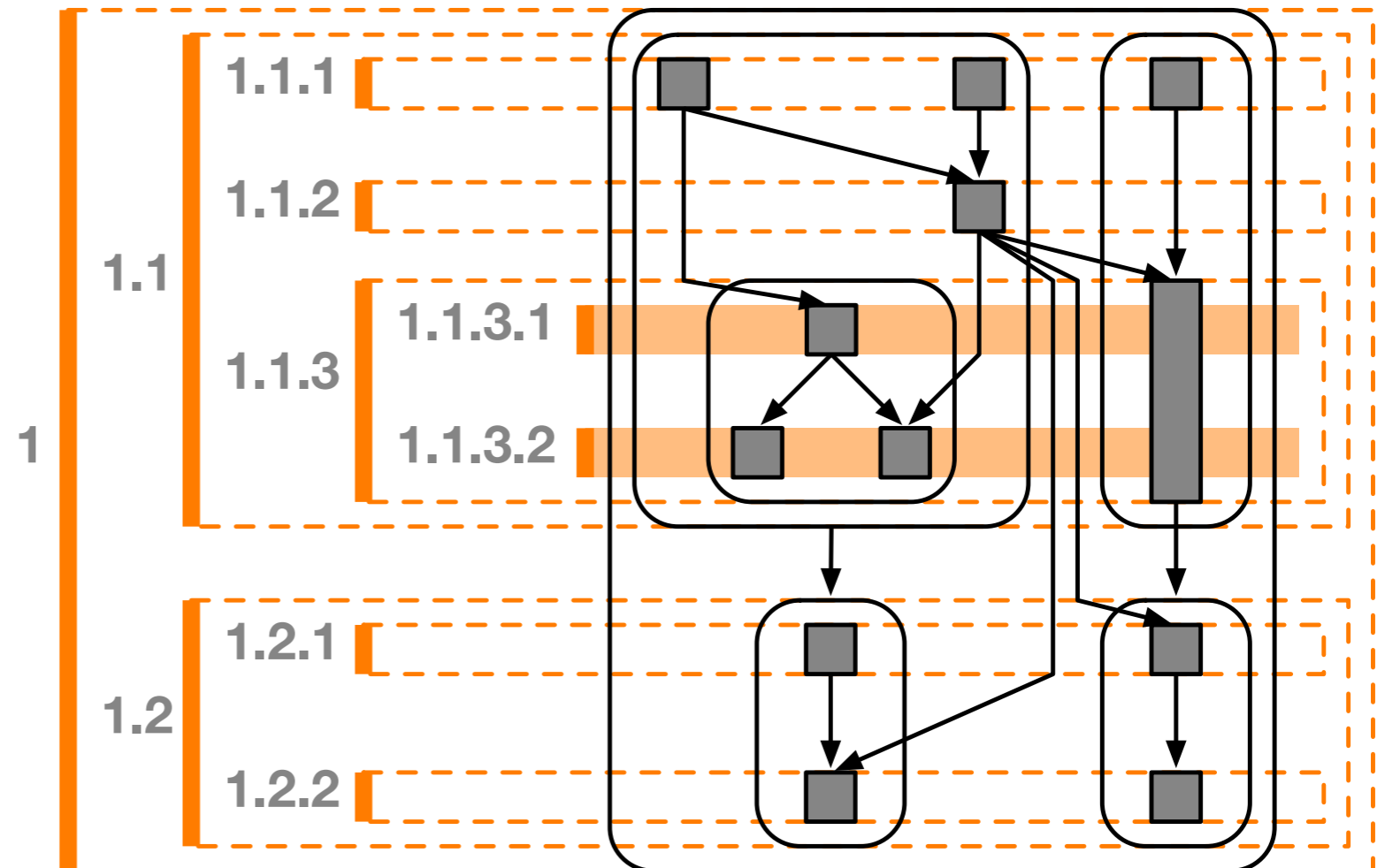
Struktur der Schichten

Rekursiv verschachtelt



Struktur der Schichten

Rekursiv verschachtelt

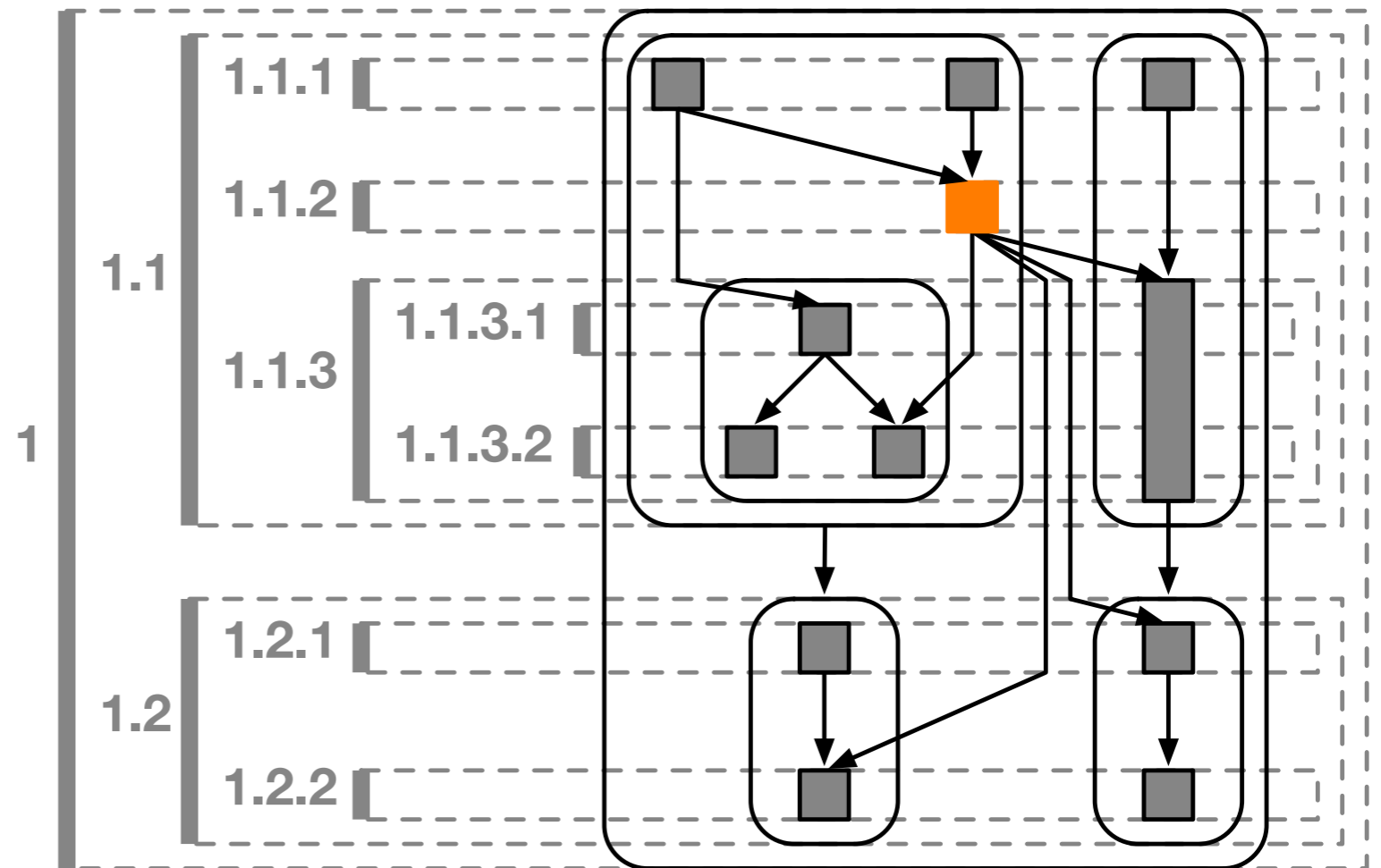


Aktualisierung der Schichten

Alte Knoten bleiben in ihren Schichten.

Schichten der neuen Kinder liegen innerhalb der Schicht des expandierten Knotens.

Standardverfahren zur Bestimmung der lokalen Schichten der Kinder.

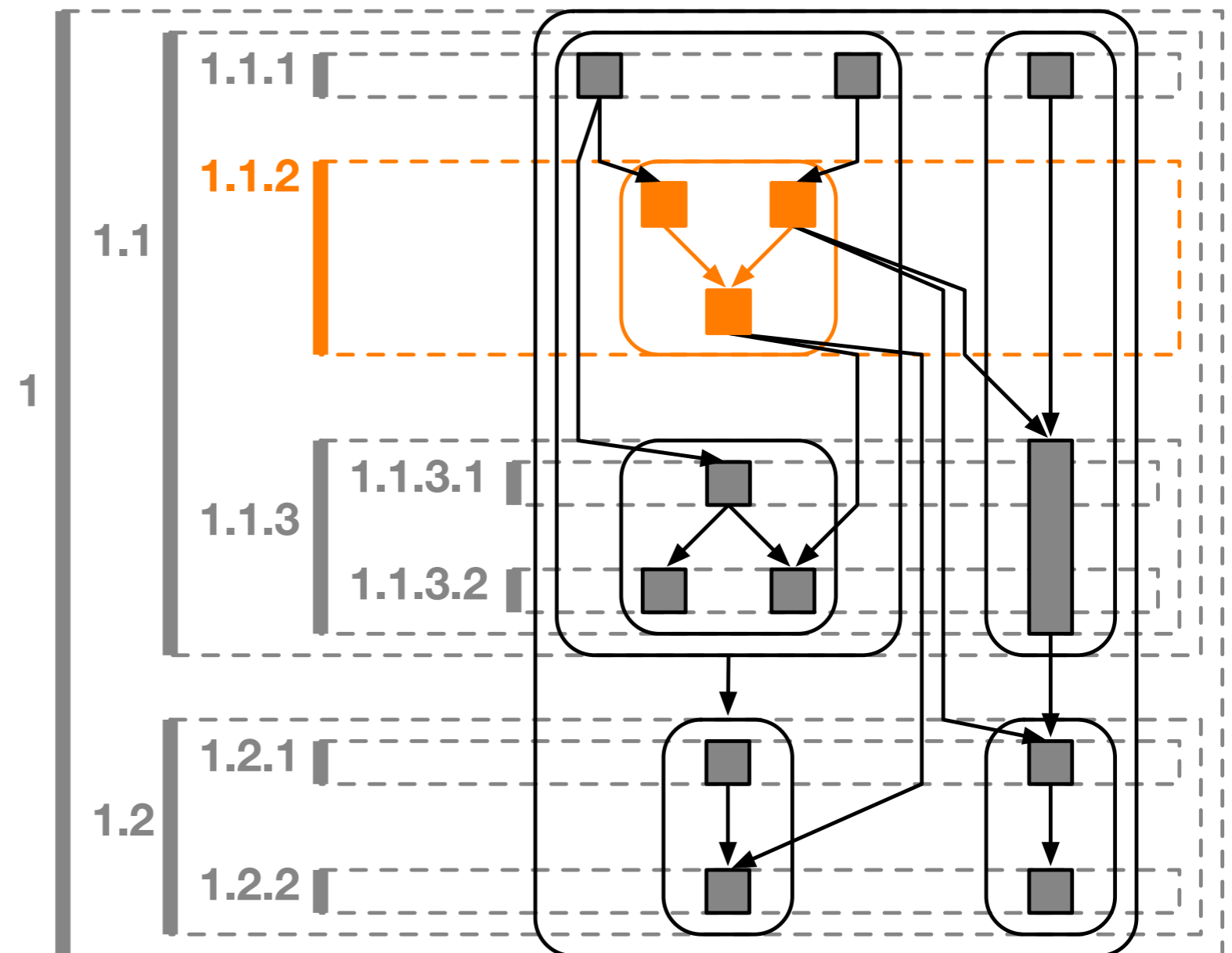


Aktualisierung der Schichten

Alte Knoten bleiben in ihren Schichten.

Schichten der neuen Kinder liegen innerhalb der Schicht des expandierten Knotens.

Standardverfahren zur Bestimmung der lokalen Schichten der Kinder.

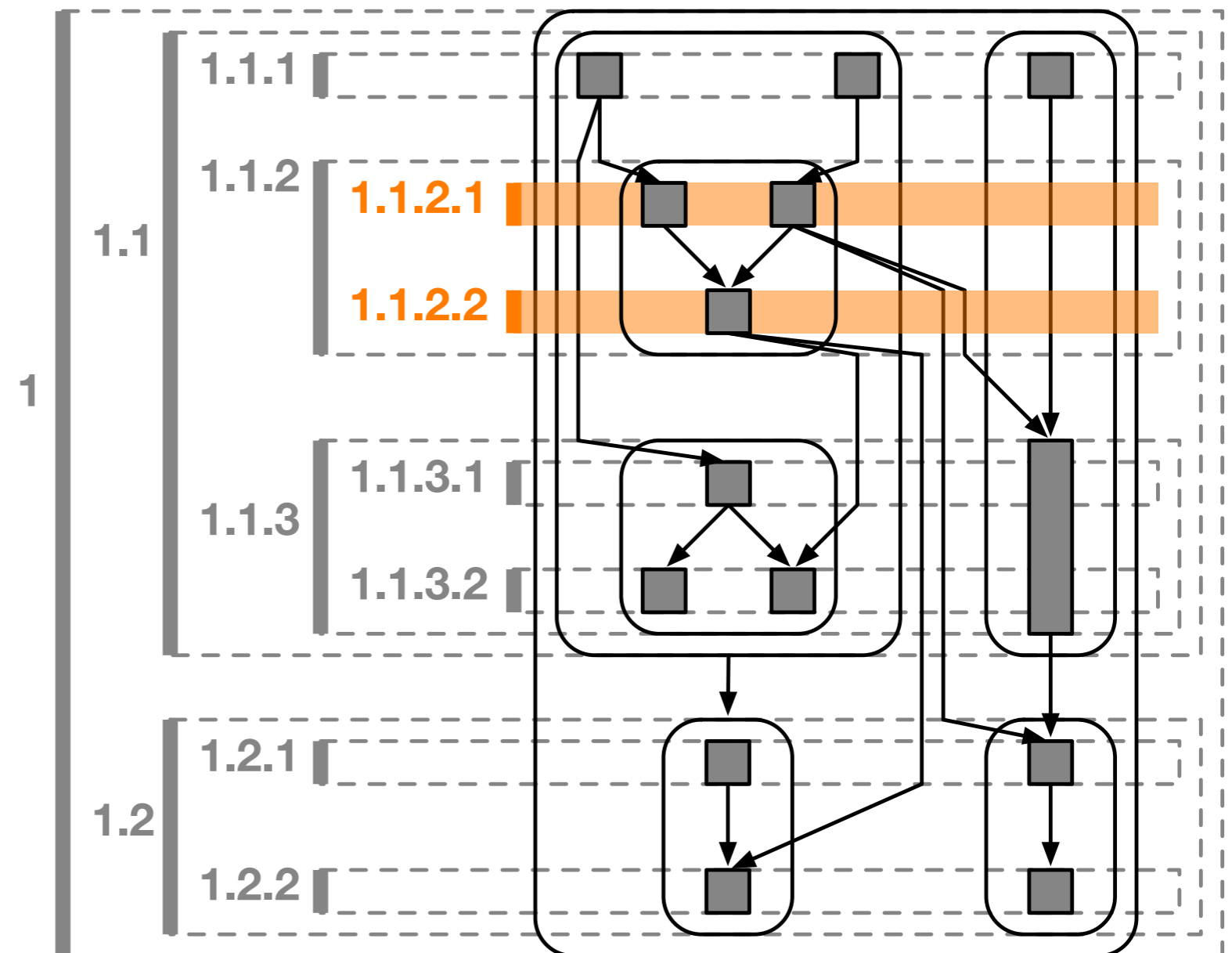


Aktualisierung der Schichten

Alte Knoten bleiben in ihren Schichten.

Schichten der neuen Kinder liegen innerhalb der Schicht des expandierten Knotens.

Standardverfahren zur Bestimmung der lokalen Schichten der Kinder.



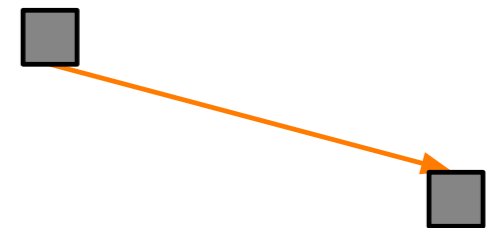
Zweite Phase

Normalisierung der Kanten

Wohlgeformte Kanten

Eine Kante (u,v) ist **wohlgeformt** falls

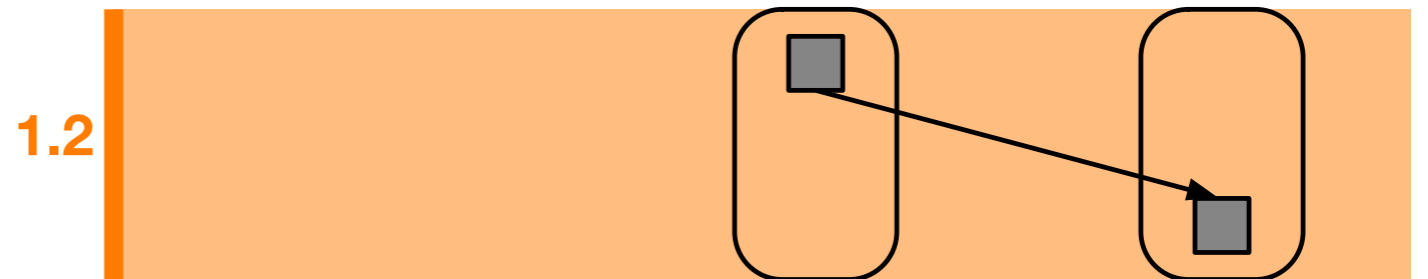
- die Eltern von u und v in derselben Schicht liegen und
- die lokalen Schichten von u und v sich nur um 1 unterscheiden.



Wohlgeformte Kanten

Eine Kante (u,v) ist **wohlgeformt** falls

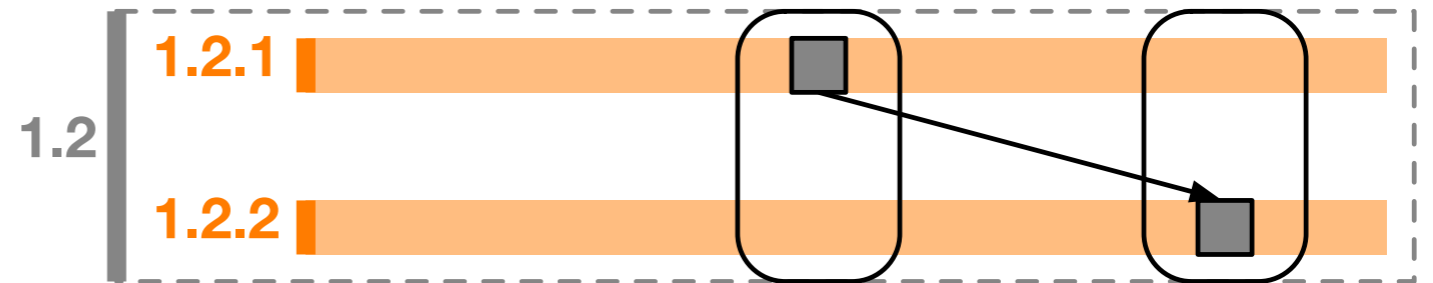
- die Eltern von u und v in **derselben Schicht** liegen und
- die lokalen Schichten von u und v sich nur um **1** unterscheiden.



Wohlgeformte Kanten

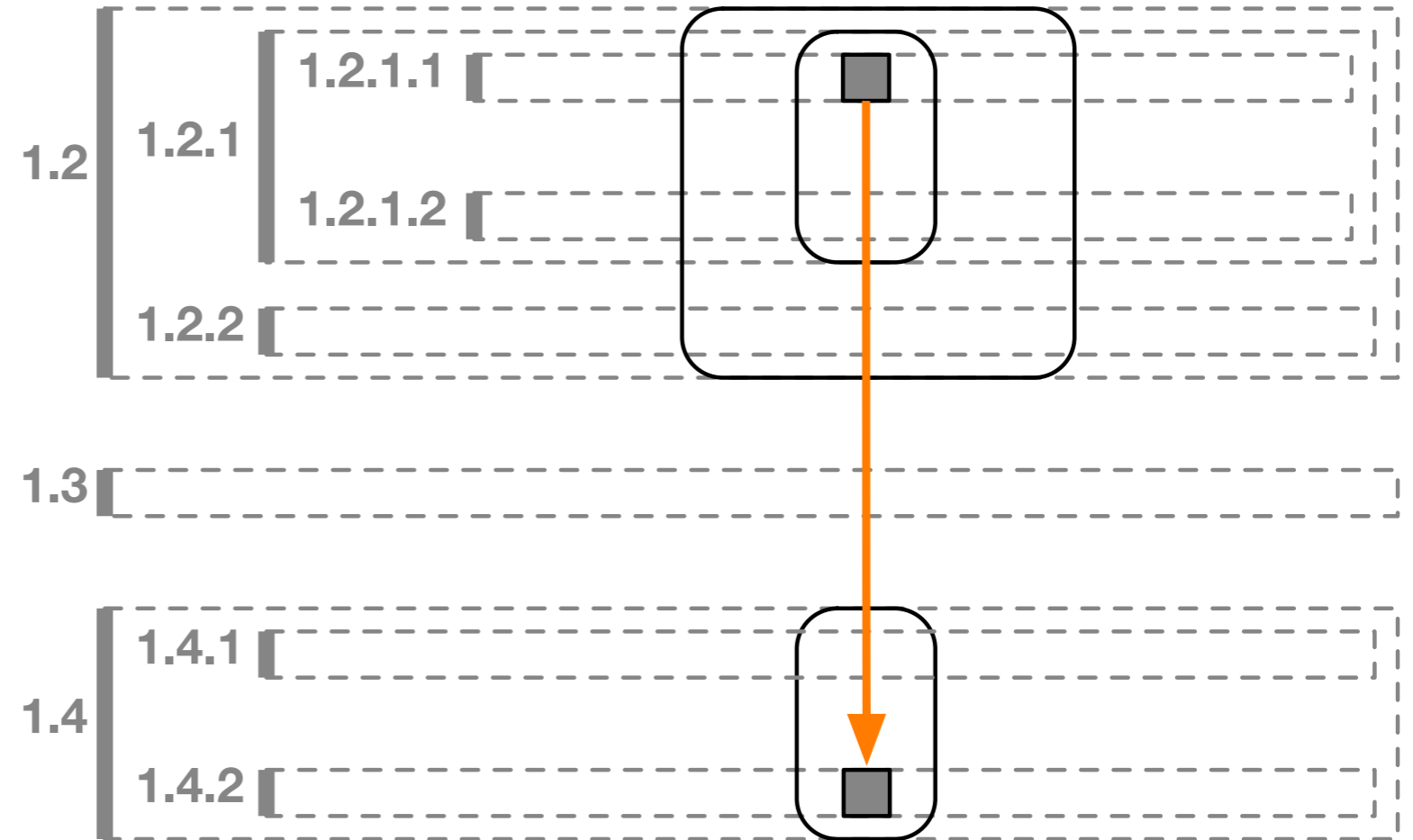
Eine Kante (u,v) ist **wohlgeformt** falls

- die Eltern von u und v in **derselben Schicht** liegen und
- die **lokalen Schichten** von u und v sich **nur um 1 unterscheiden**.



Normalisierung

Eine nicht wohlgeformte Kante...



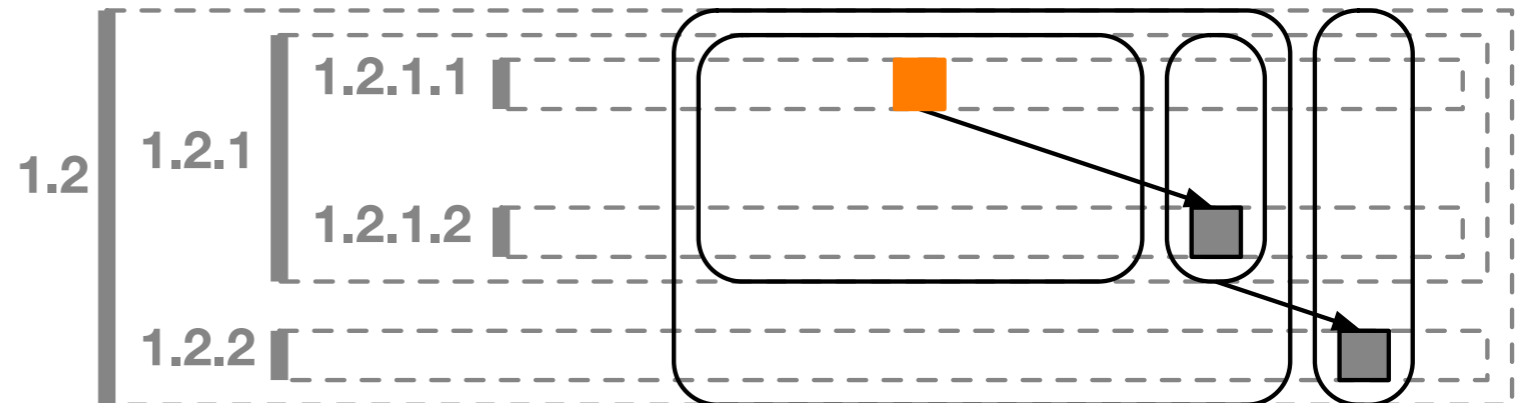
Aktualisierung

Alle Kanten sind wohlgeformt vor dem Expandieren.

Alle alten Knoten bleiben in ihren Schichten.

Nicht wohlgeformte Kanten sind immer inzident zu neuen Kindern.

Normalisierung wie beschrieben.



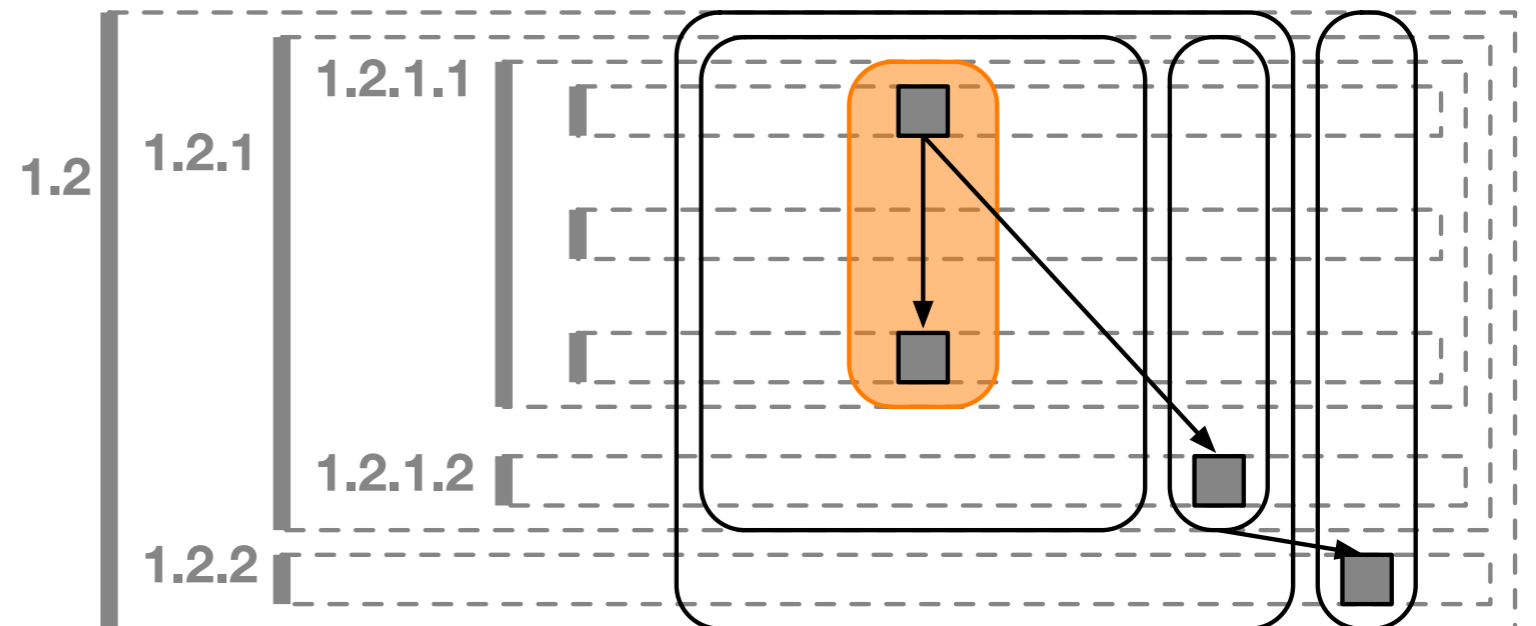
Aktualisierung

Alle Kanten sind wohlgeformt vor dem Expandieren.

Alle alten Knoten bleiben in ihren Schichten.

Nicht wohlgeformte Kanten sind immer inzident zu neuen Kindern.

Normalisierung wie beschrieben.



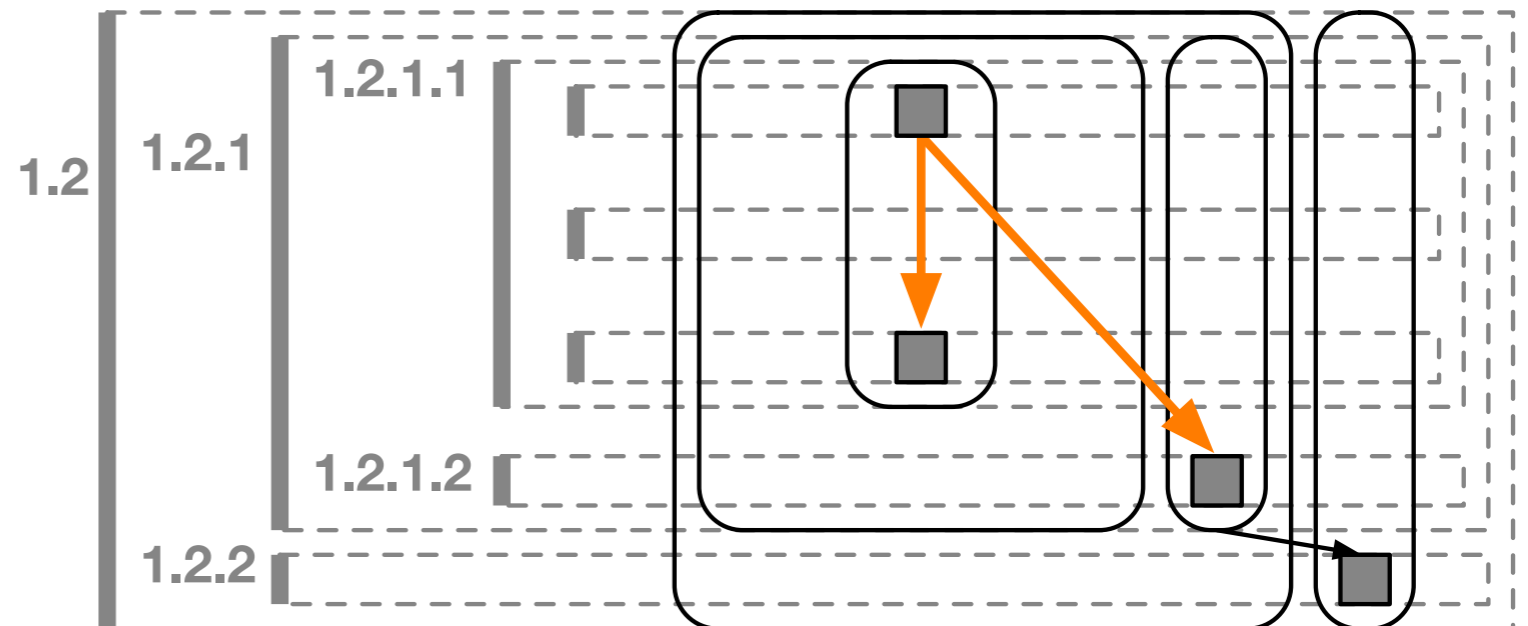
Aktualisierung

Alle Kanten sind wohlgeformt vor dem Expandieren.

Alle alten Knoten bleiben in ihren Schichten.

Nicht wohlgeformte Kanten sind immer inzident zu neuen Kindern.

Normalisierung wie beschrieben.



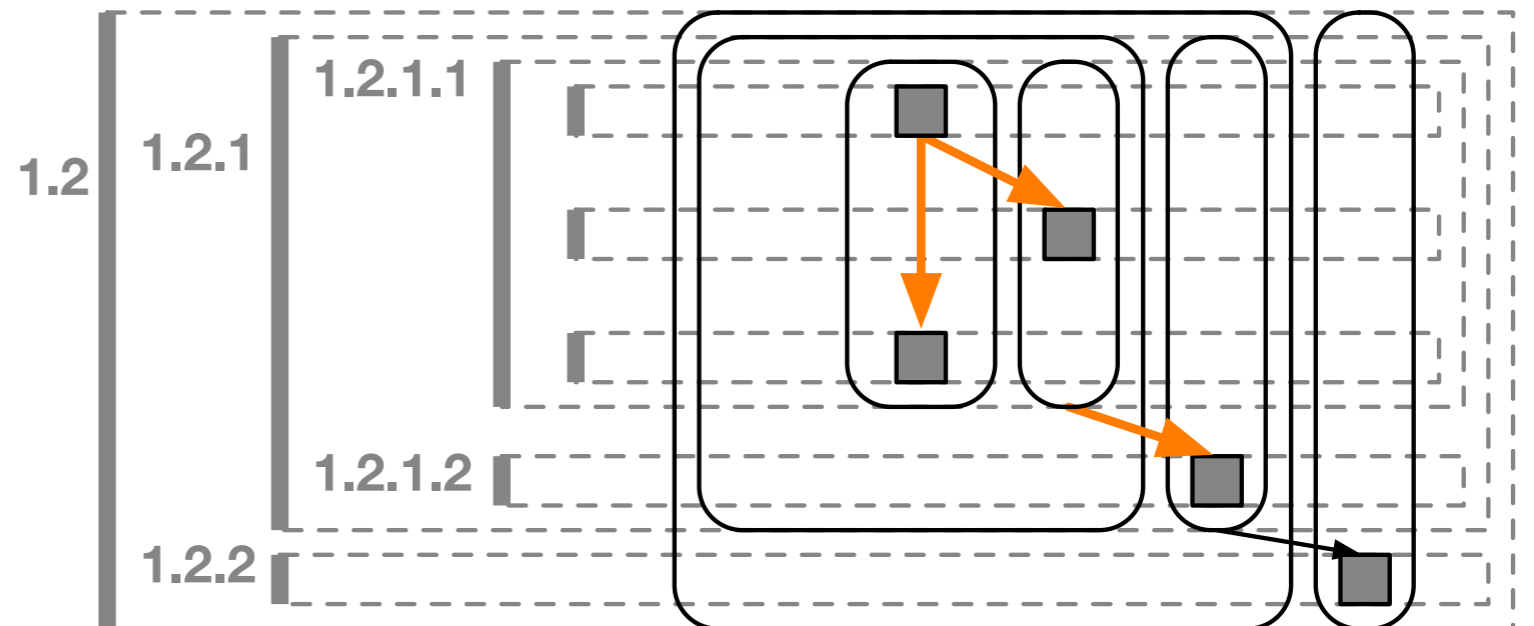
Aktualisierung

Alle Kanten sind wohlgeformt vor dem Expandieren.

Alle alten Knoten bleiben in ihren Schichten.

Nicht wohlgeformte Kanten sind immer inzident zu neuen Kindern.

Normalisierung wie beschrieben.



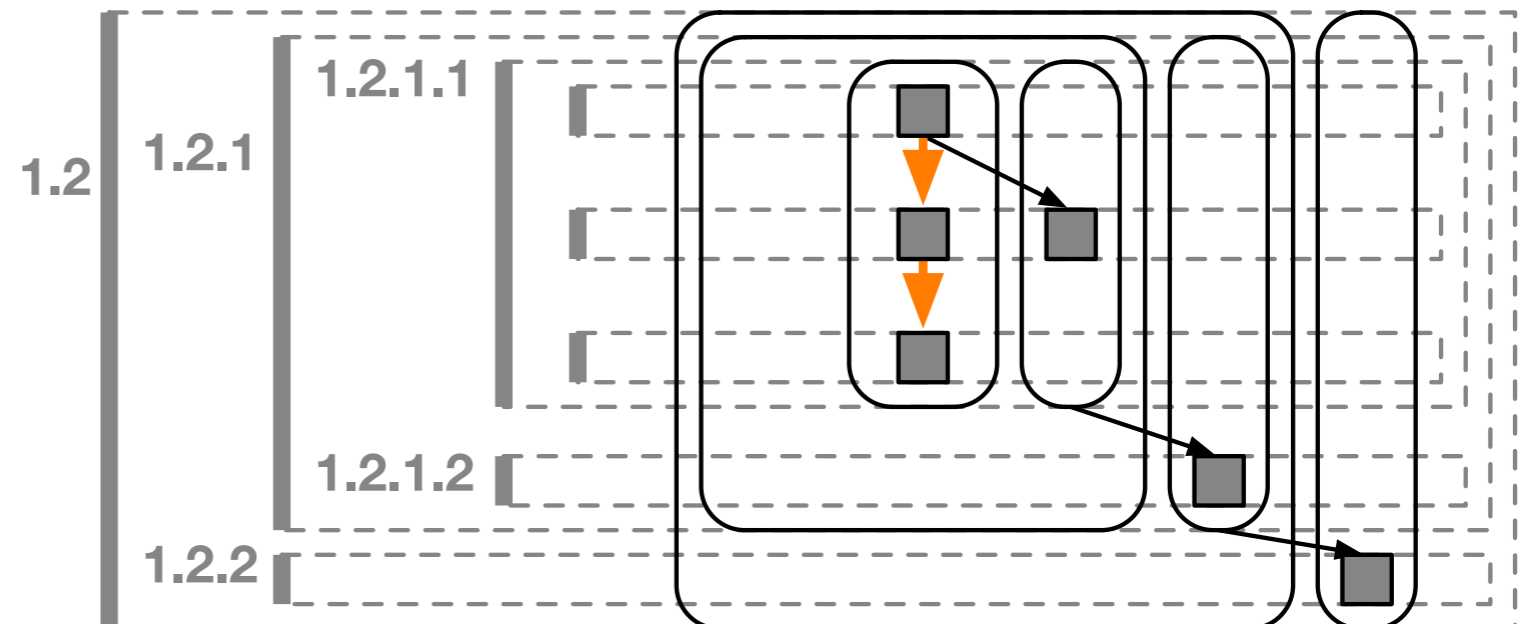
Aktualisierung

Alle Kanten sind wohlgeformt vor dem Expandieren.

Alle alten Knoten bleiben in ihren Schichten.

Nicht wohlgeformte Kanten sind immer inzident zu neuen Kindern.

Normalisierung wie beschrieben.



Dritte Phase

Kreuzungsreduktion

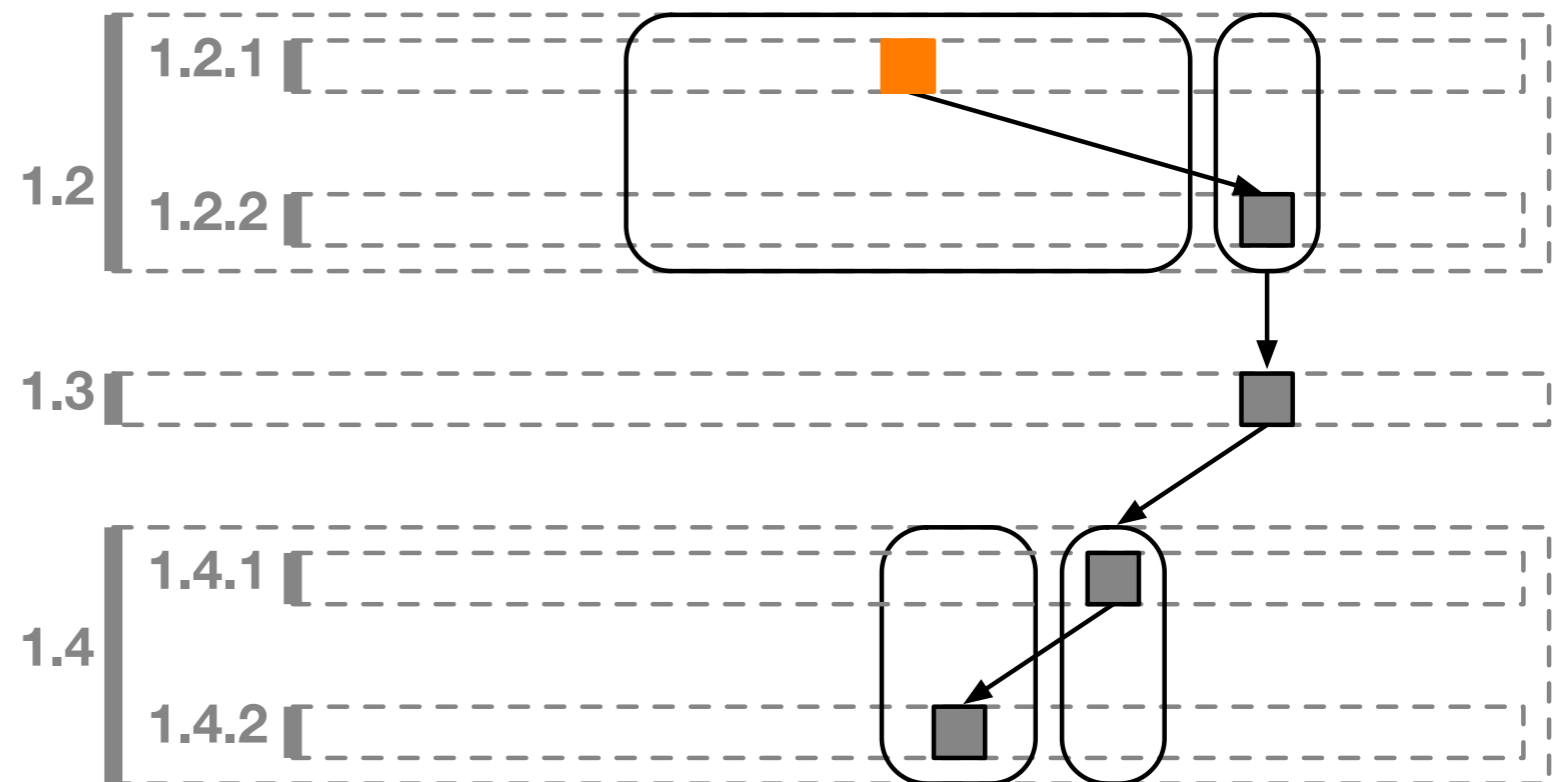
Aktualisierung

Alle alten Knoten bleiben in ihren Schichten.

Behalte relative Ordnung der alten Knoten bei.

Nur **neue Knoten** und **neue Hilfsknoten** müssen eingeordnet werden.

Zuerst die Hilfsknoten.



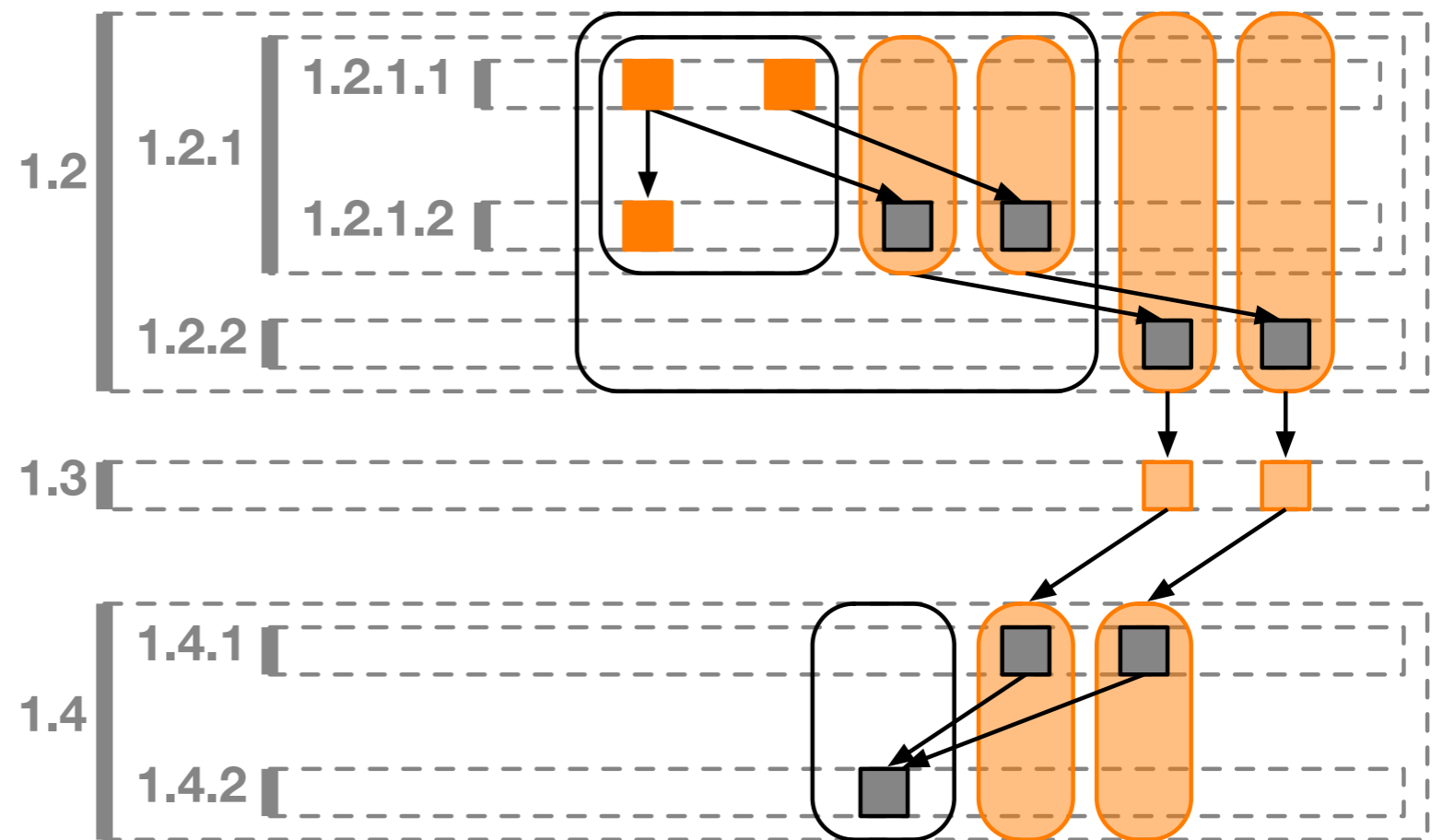
Aktualisierung

Alle alten Knoten bleiben in ihren Schichten.

Behalte relative Ordnung der alten Knoten bei.

Nur **neue Knoten** und **neue Hilfsknoten** müssen eingeordnet werden.

Zuerst die Hilfsknoten.



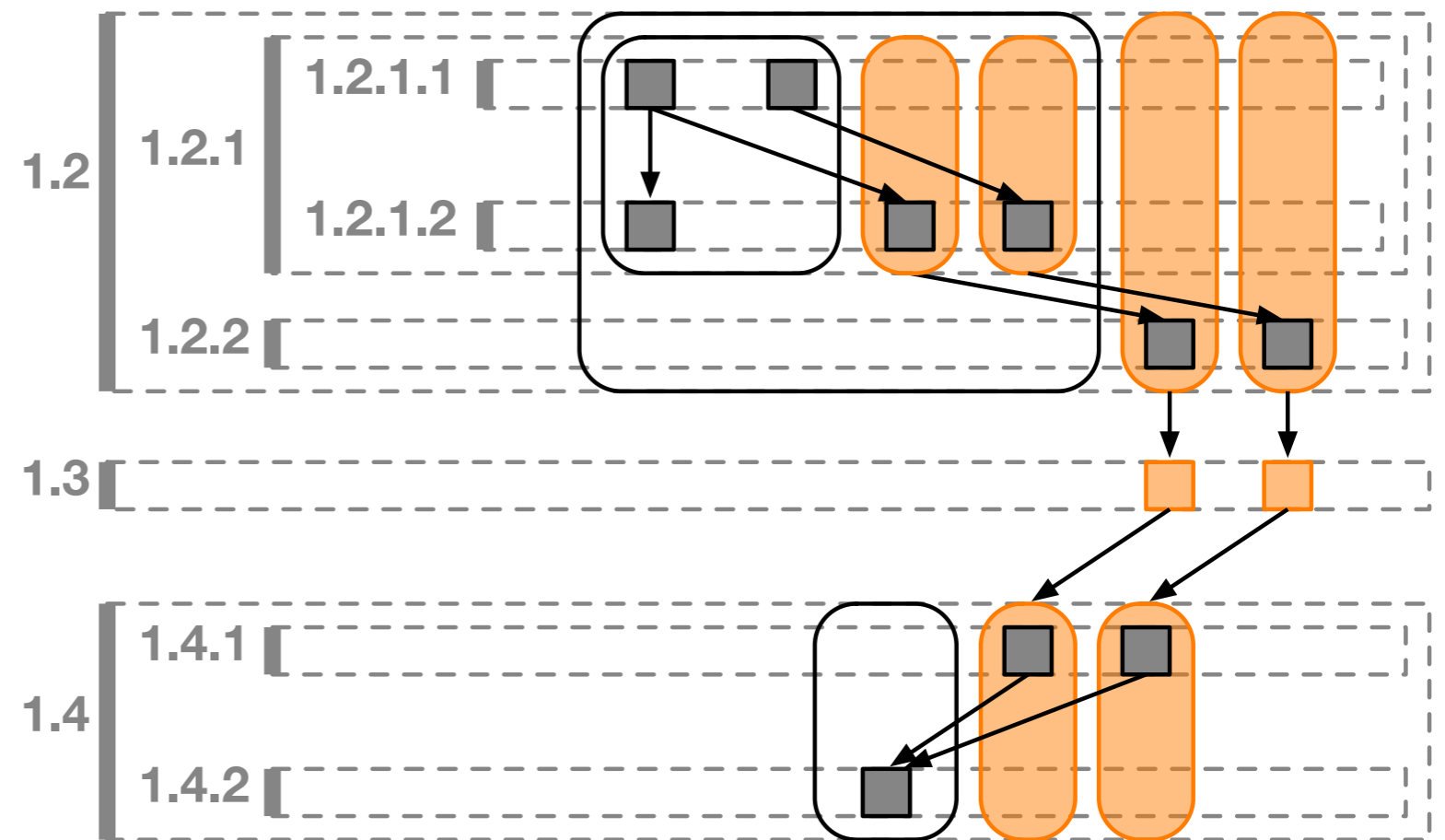
Aktualisierung

Alle alten Knoten bleiben in ihren Schichten.

Behalte relative Ordnung der alten Knoten bei.

Nur **neue Knoten** und **neue Hilfsknoten** müssen eingeordnet werden.

Zuerst die Hilfsknoten.

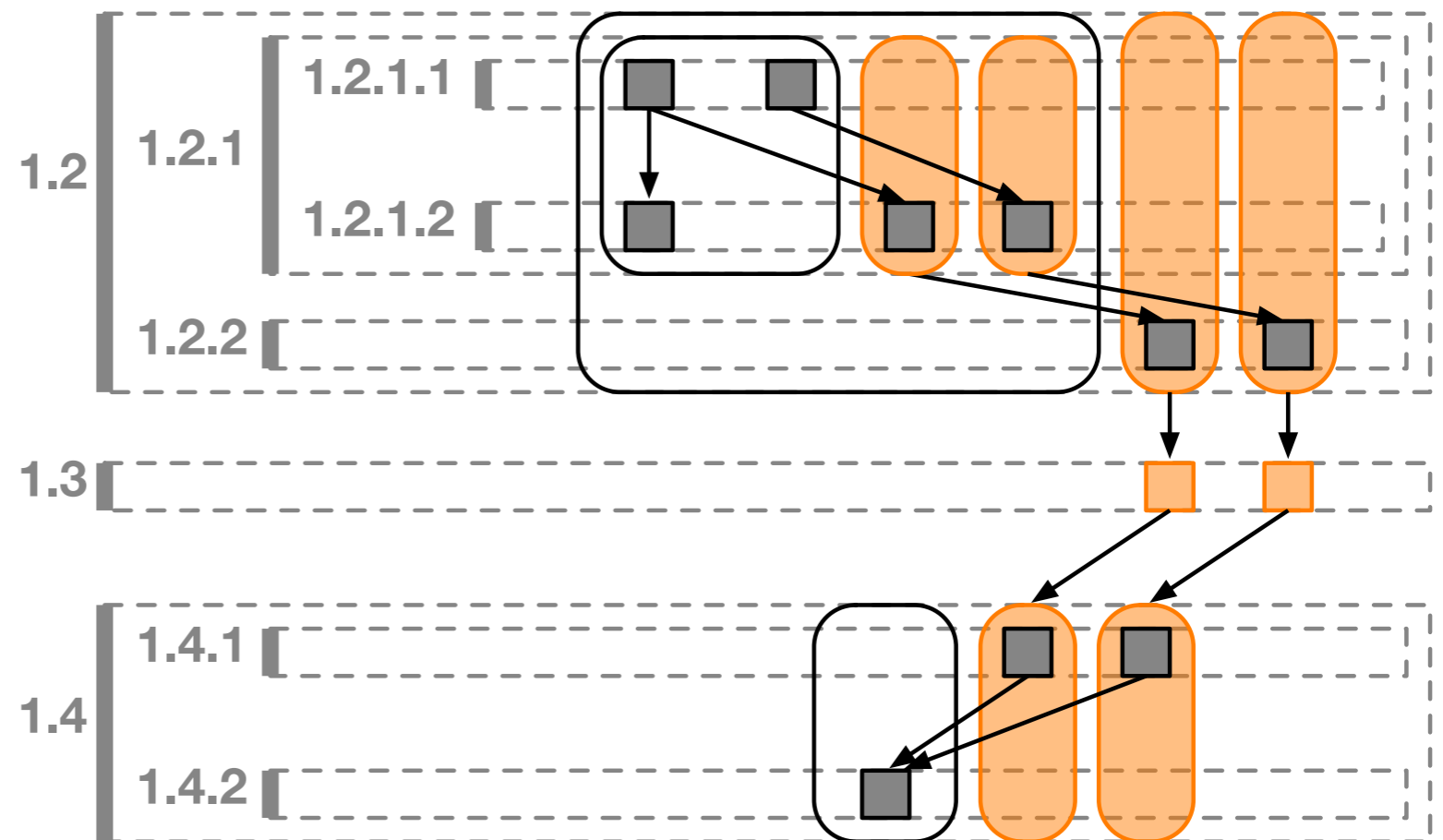


Gruppieren von Hilfsknoten

Ziel: Expandierte Kanten erben den Verlauf der kontrahierten Kante.

Alle expandierten Kanten zu einer kontrahierten haben den gleichen Verlauf.

Ihre Hilfsknoten werden in **Blöcke** gruppiert.

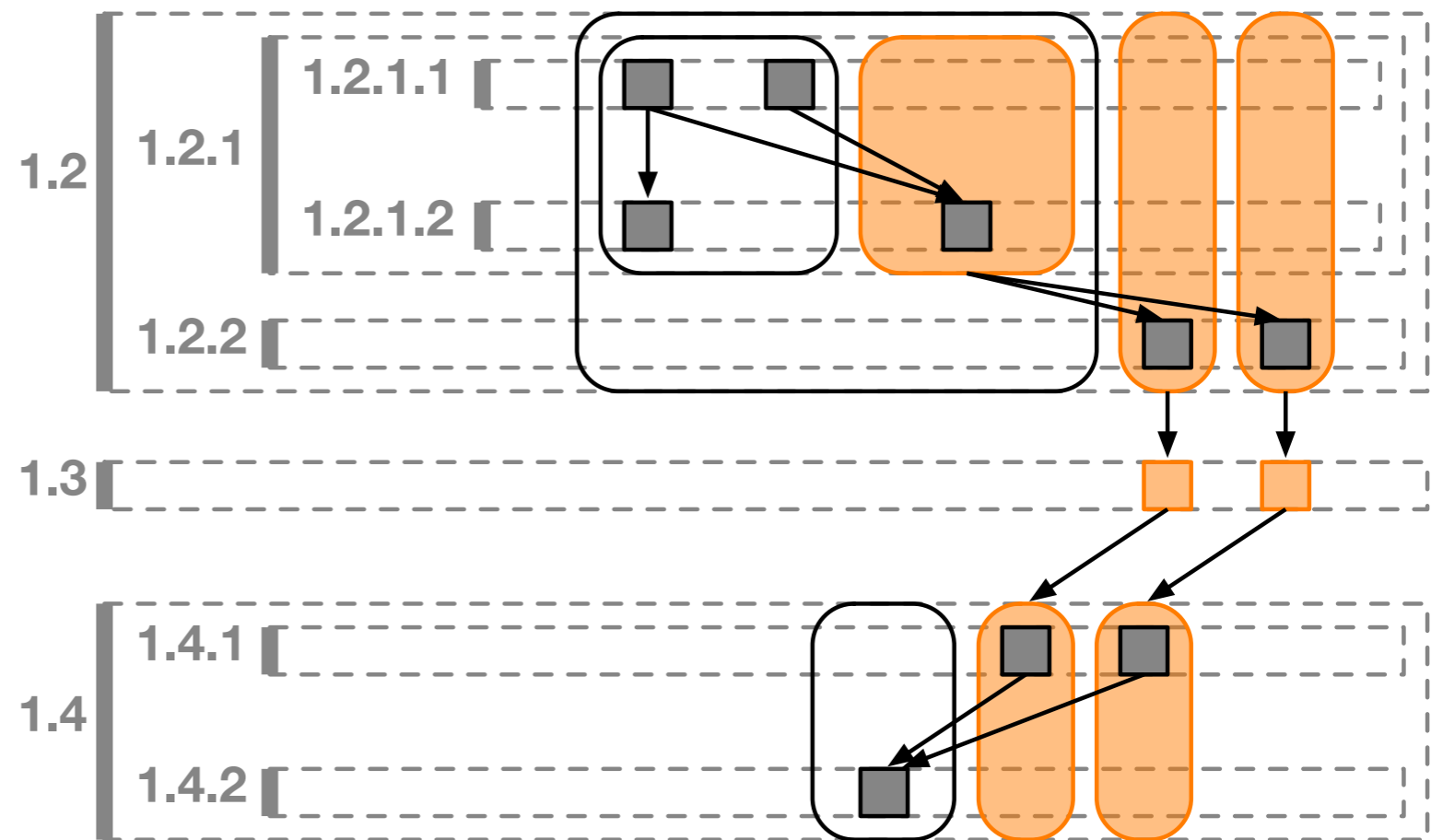


Gruppieren von Hilfsknoten

Ziel: Expandierte Kanten erben den Verlauf der kontrahierten Kante.

Alle expandierten Kanten zu einer kontrahierten haben den gleichen Verlauf.

Ihre Hilfsknoten werden in **Blöcke** gruppiert.

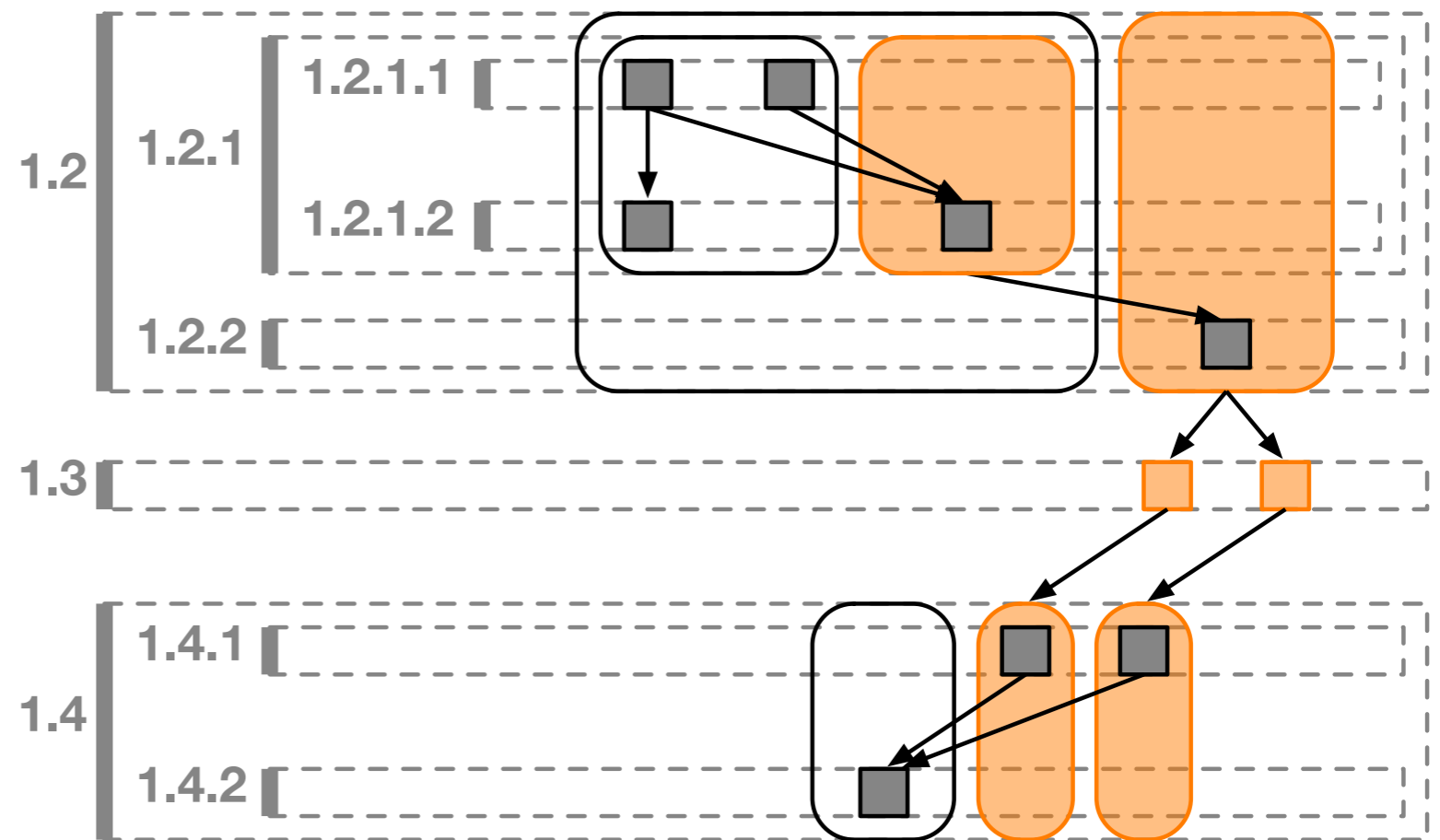


Gruppieren von Hilfsknoten

Ziel: Expandierte Kanten erben den Verlauf der kontrahierten Kante.

Alle expandierten Kanten zu einer kontrahierten haben den gleichen Verlauf.

Ihre Hilfsknoten werden in **Blöcke** gruppiert.

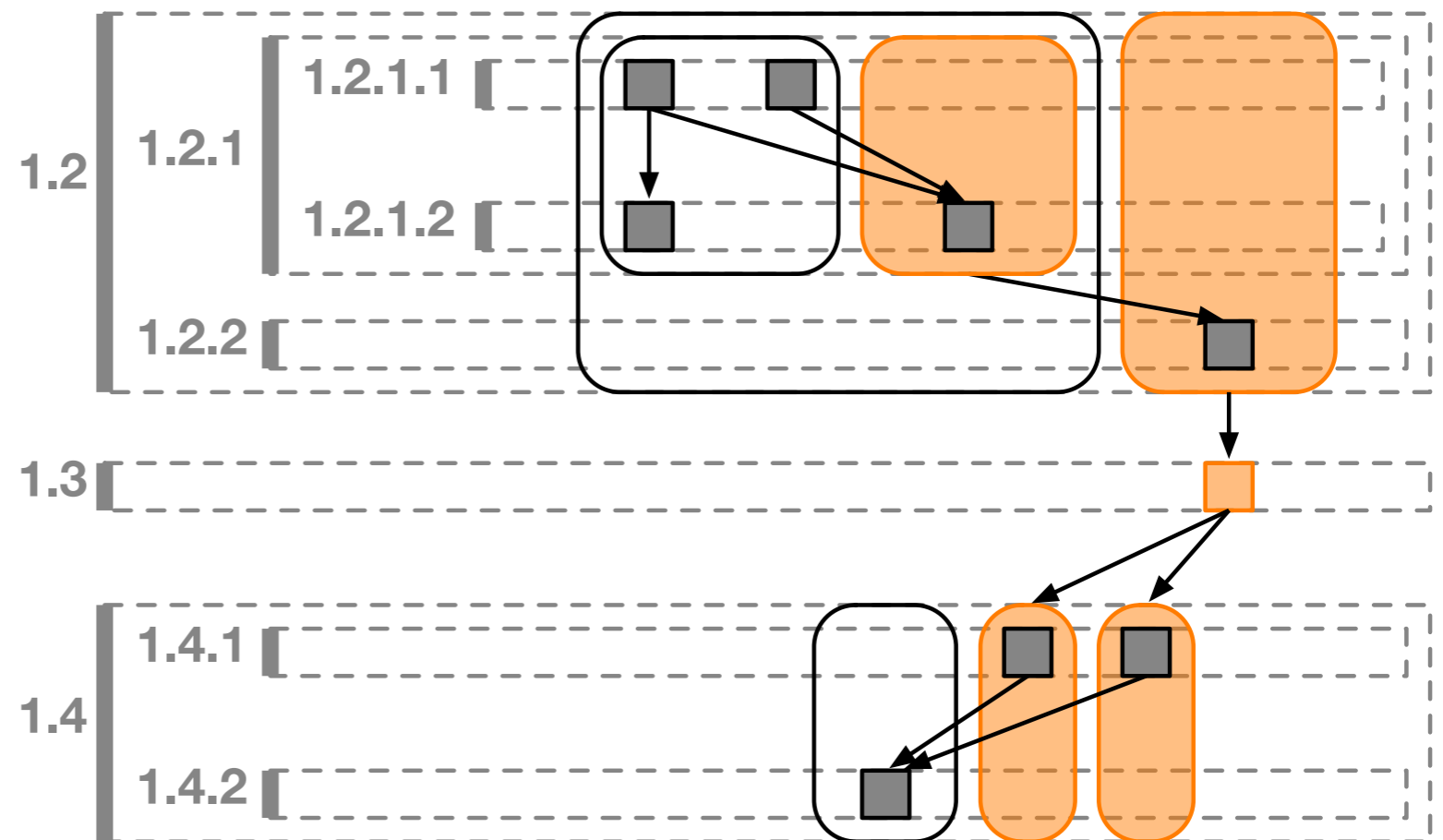


Gruppieren von Hilfsknoten

Ziel: Expandierte Kanten erben den Verlauf der kontrahierten Kante.

Alle expandierten Kanten zu einer kontrahierten haben den gleichen Verlauf.

Ihre Hilfsknoten werden in **Blöcke** gruppiert.

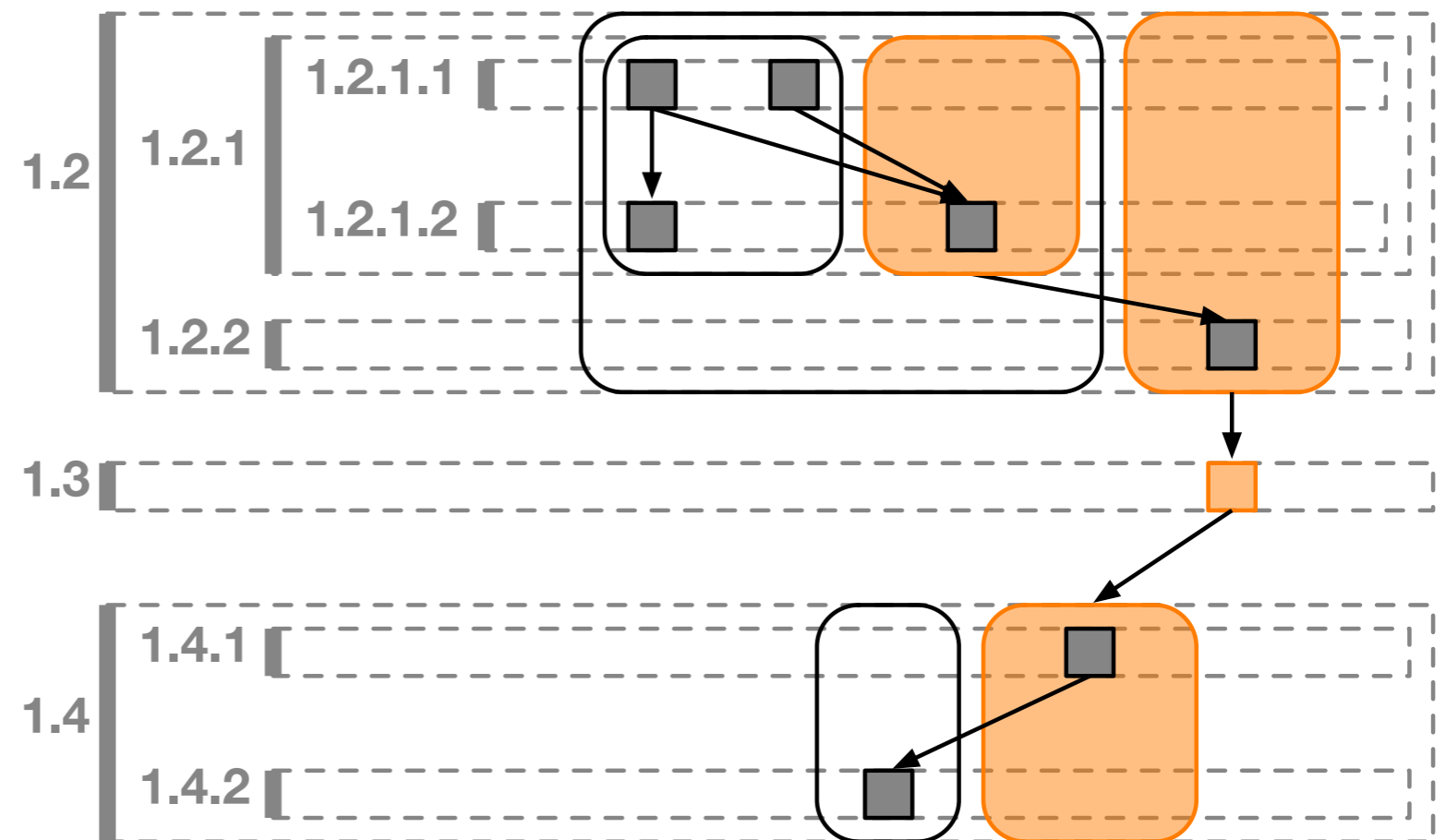


Gruppieren von Hilfsknoten

Ziel: Expandierte Kanten erben den Verlauf der kontrahierten Kante.

Alle expandierten Kanten zu einer kontrahierten haben den gleichen Verlauf.

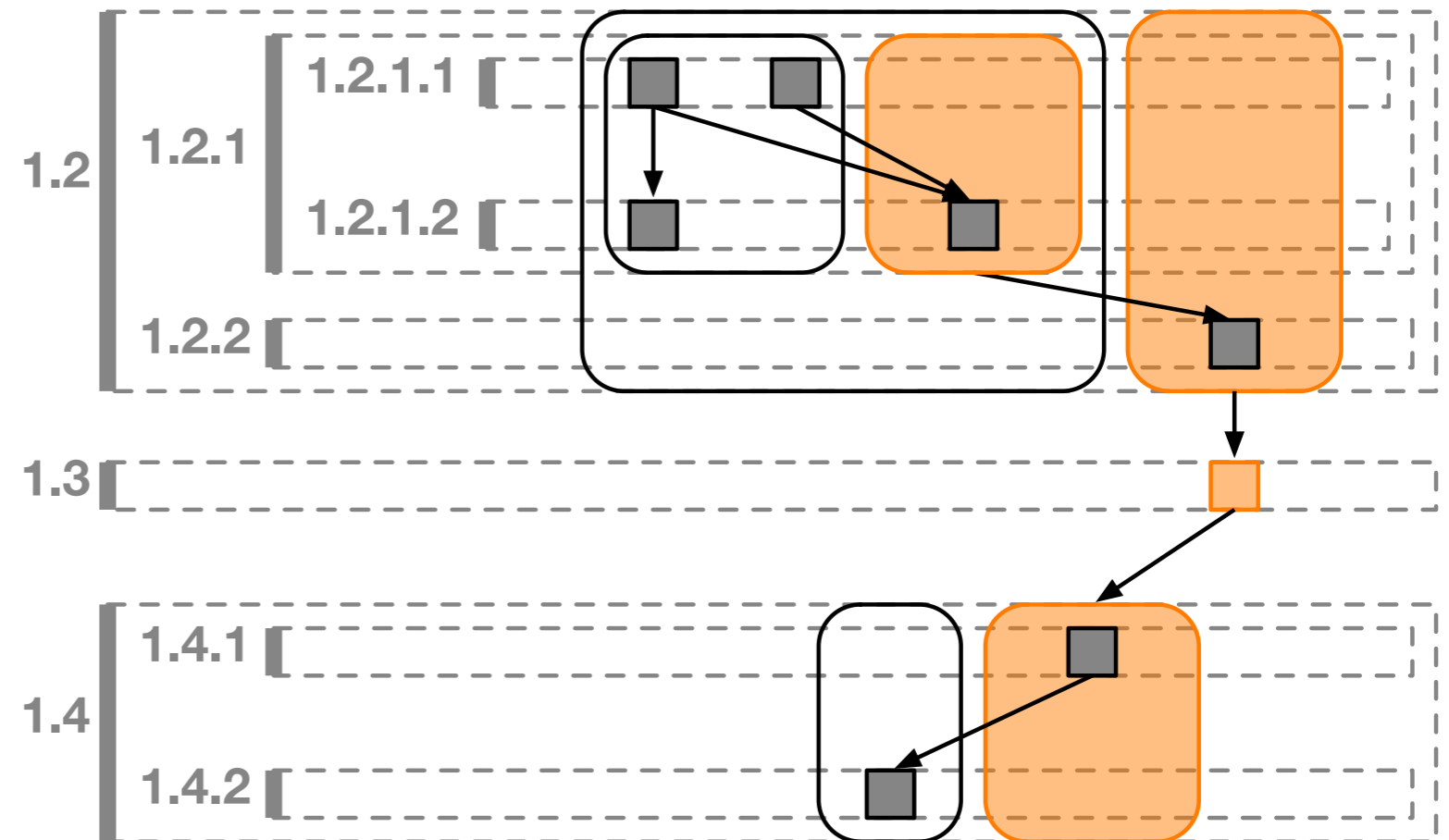
Ihre Hilfsknoten werden in **Blöcke** gruppiert.



Wiederverwendung

Viele **Blöcke** entsprechen Hilfsknoten der kontrahierten Kante.

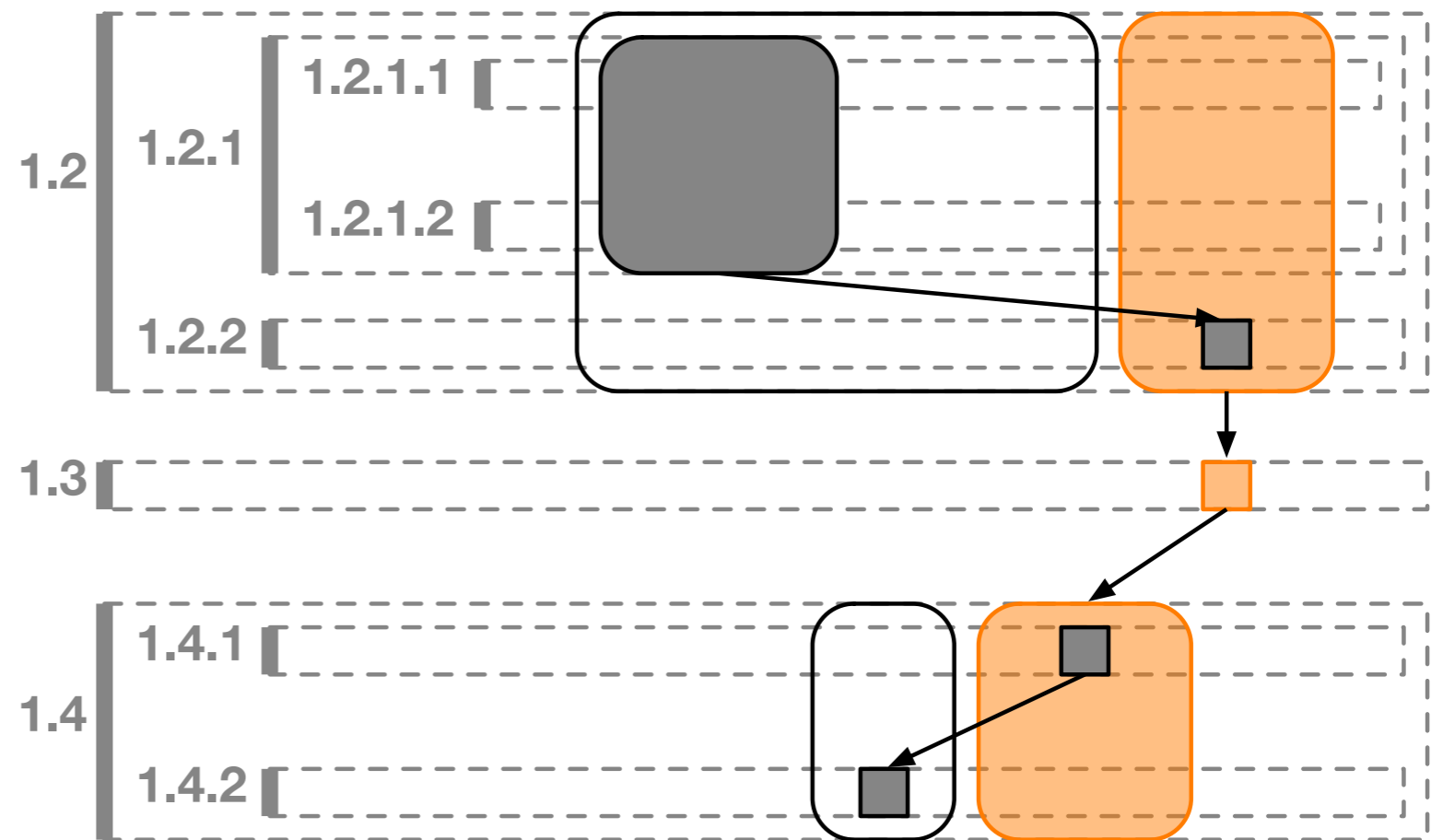
Diese Blöcke **erben die Positionen** der alten Hilfsknoten.



Wiederverwendung

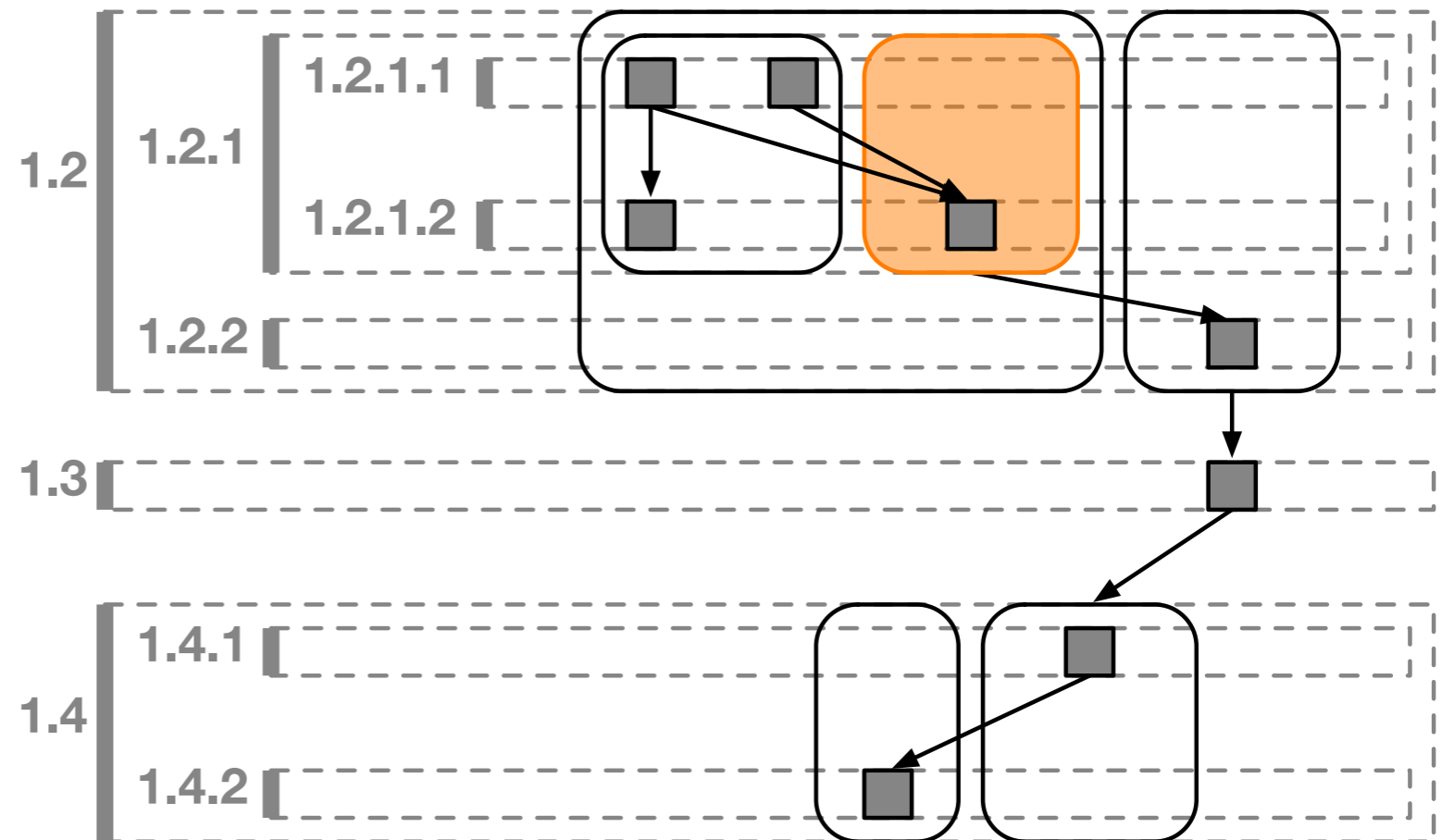
Viele **Blöcke** entsprechen Hilfsknoten der kontrahierten Kante.

Diese Blöcke **erben die Positionen** der alten Hilfsknoten.



Positionen neuer Blöcke

Neu zu ordnende Blöcke sind **Geschwister** des **expandierten Knotens**.

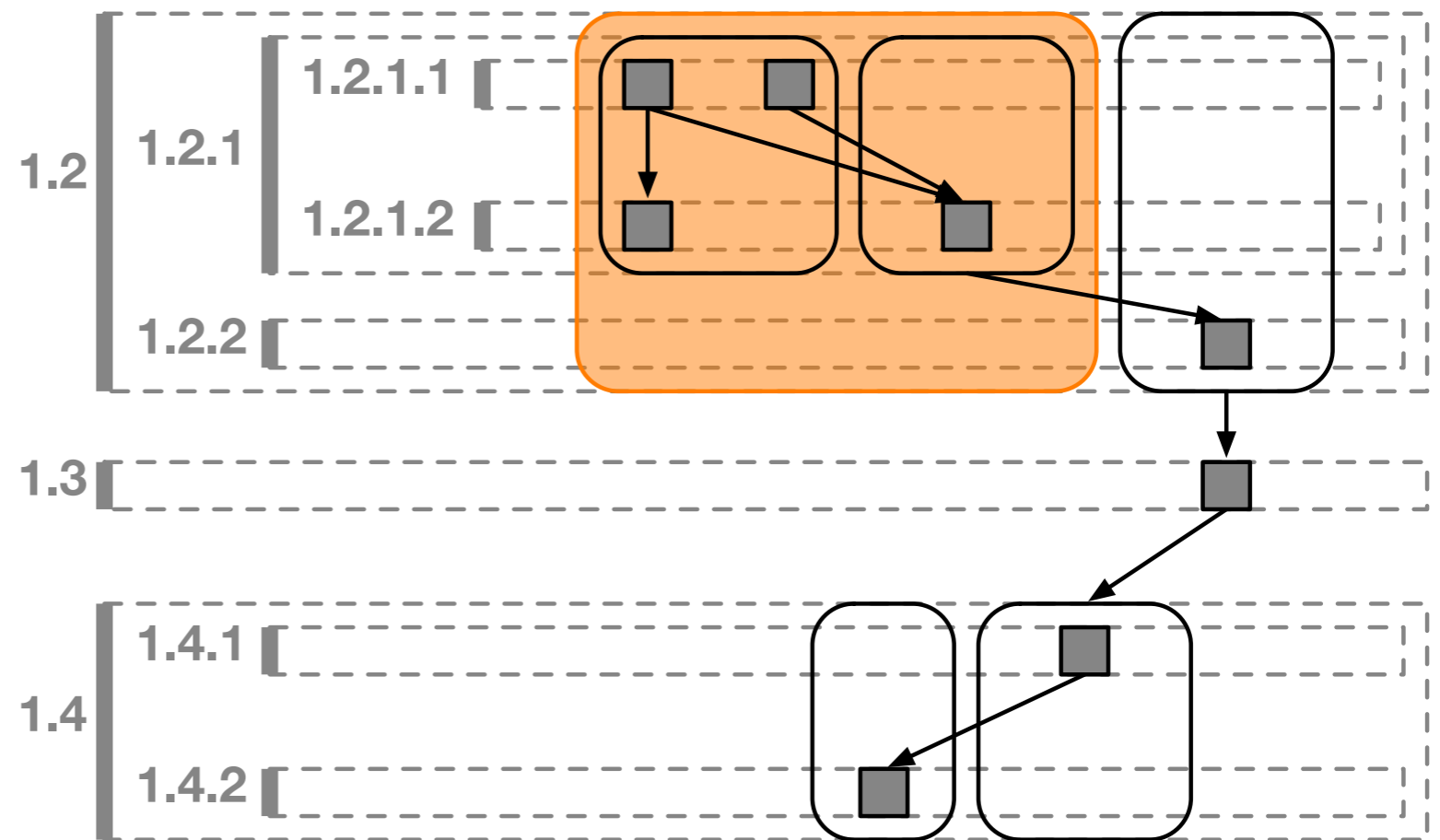


Positionen neuer Blöcke

Neu zu ordnende Blöcke sind **Geschwister** des **expandierten Knotens**.

Variante des **Original-Algorithmus**:

- Start beim **Vorgänger** des expandierten Knotens.
- Eingeschränkt auf die **Schicht** des expandierten Knotens

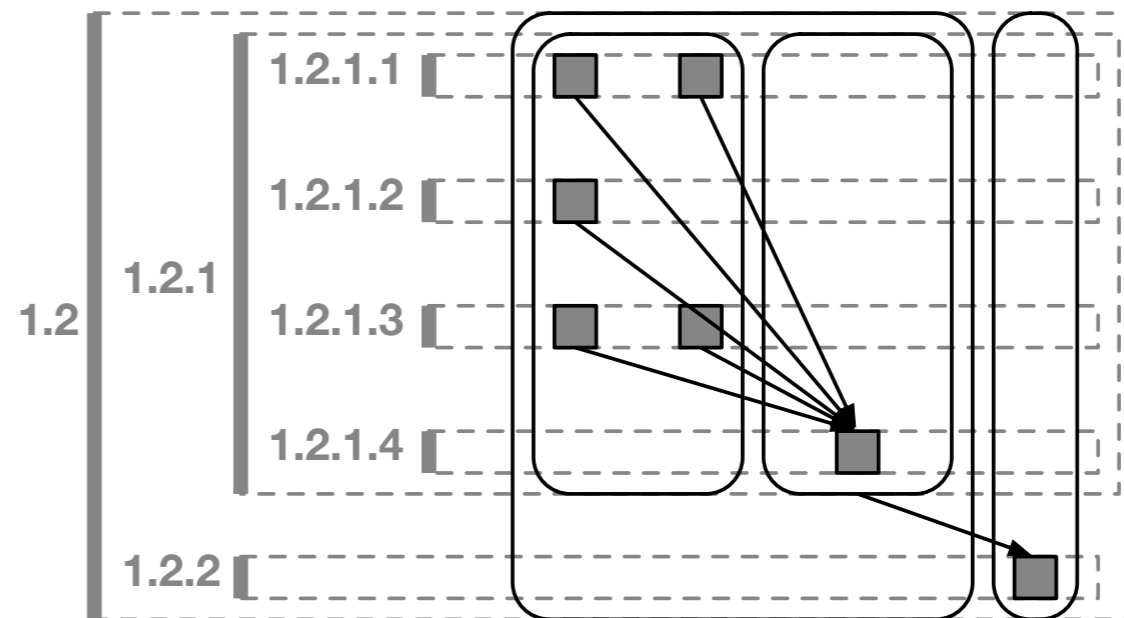


Aufspalten der Blöcke

Position der Blöcke fest

Ziel: Bestimmung der Ordnung der Hilfsknoten innerhalb jedes Blocks.

Kann aus der Ordnung der neuen Kinder abgeleitet werden.

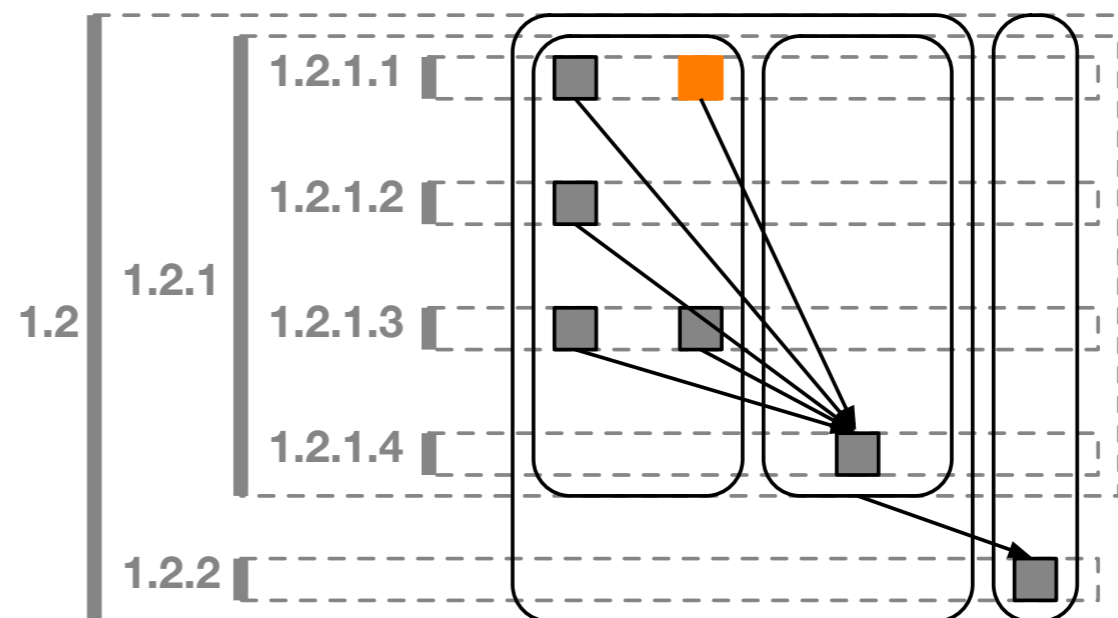


Aufspalten der Blöcke

Position der Blöcke fest

Ziel: Bestimmung der Ordnung der Hilfsknoten innerhalb jedes Blocks.

Kann aus der Ordnung der neuen Kinder abgeleitet werden.

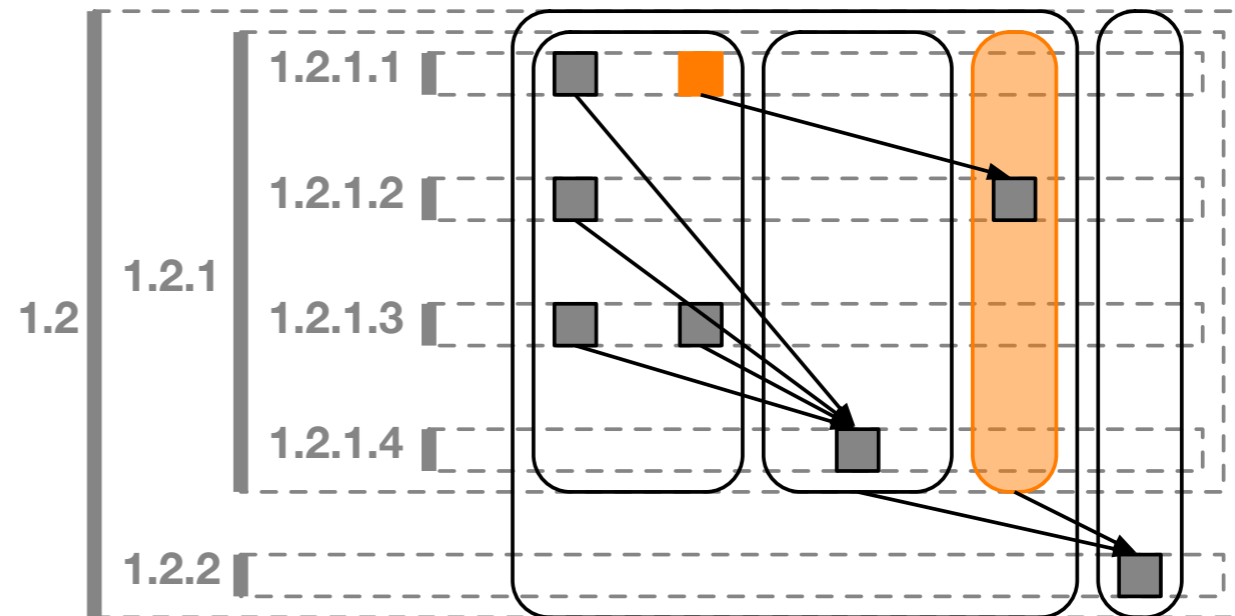


Aufspalten der Blöcke

Position der Blöcke fest

Ziel: Bestimmung der Ordnung der Hilfsknoten innerhalb jedes Blocks.

Kann aus der Ordnung der neuen Kinder abgeleitet werden.

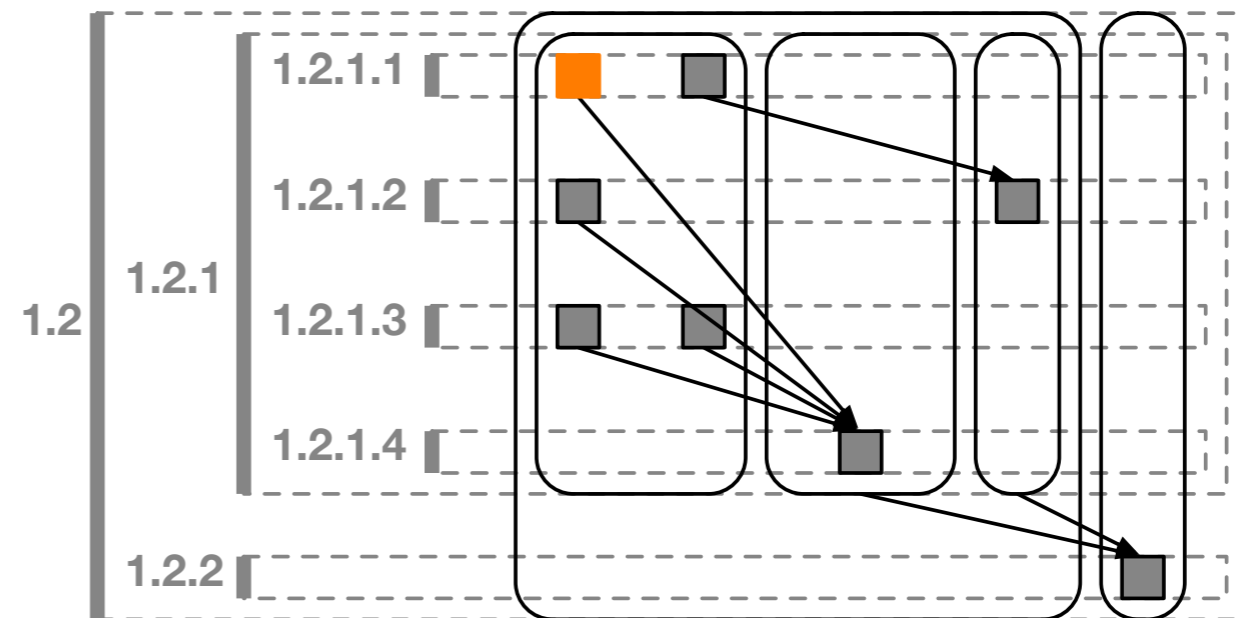


Aufspalten der Blöcke

Position der Blöcke fest

Ziel: Bestimmung der Ordnung der Hilfsknoten innerhalb jedes Blocks.

Kann aus der Ordnung der neuen Kinder abgeleitet werden.

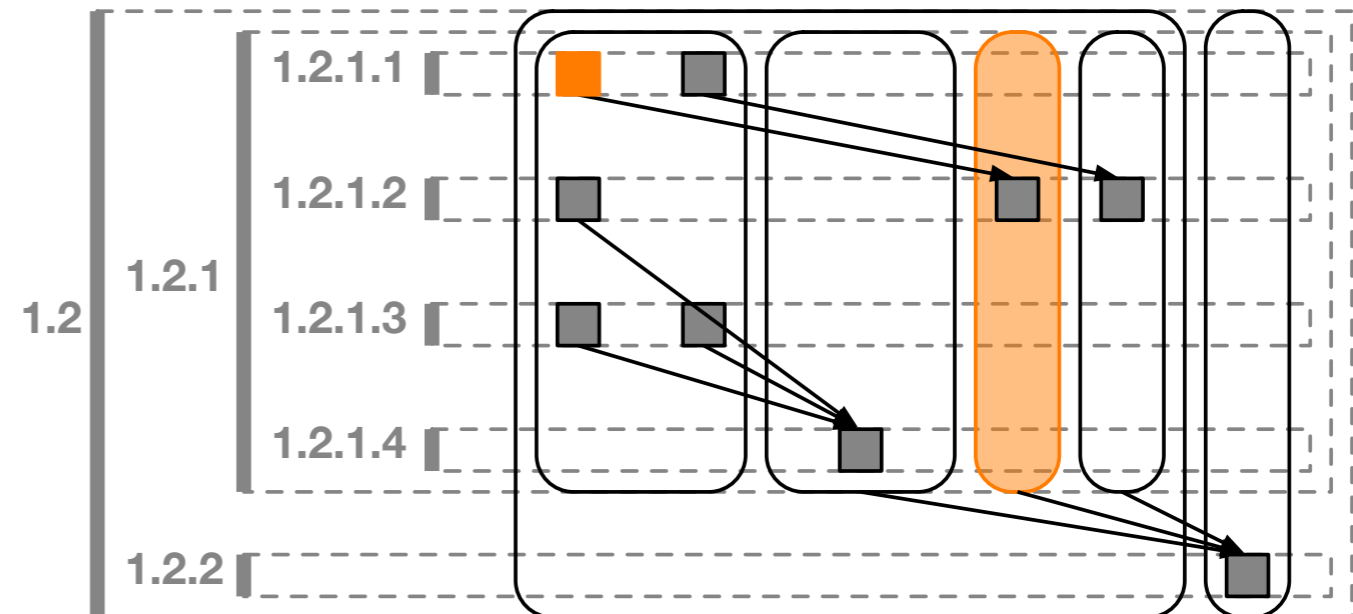


Aufspalten der Blöcke

Position der Blöcke fest

Ziel: Bestimmung der
Ordnung der Hilfsknoten
innerhalb jedes Blocks.

Kann aus der Ordnung
der neuen Kinder
abgeleitet werden.

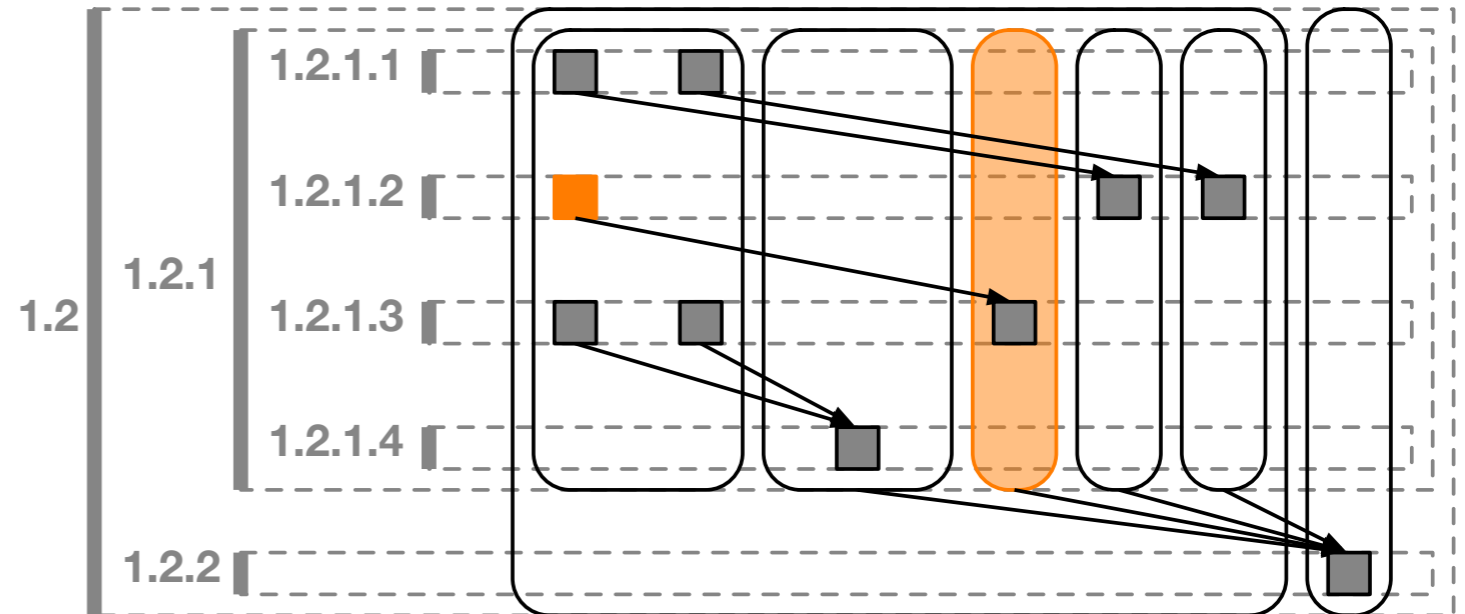


Aufspalten der Blöcke

Position der Blöcke fest

Ziel: Bestimmung der Ordnung der Hilfsknoten innerhalb jedes Blocks.

Kann aus der Ordnung der neuen Kinder abgeleitet werden.

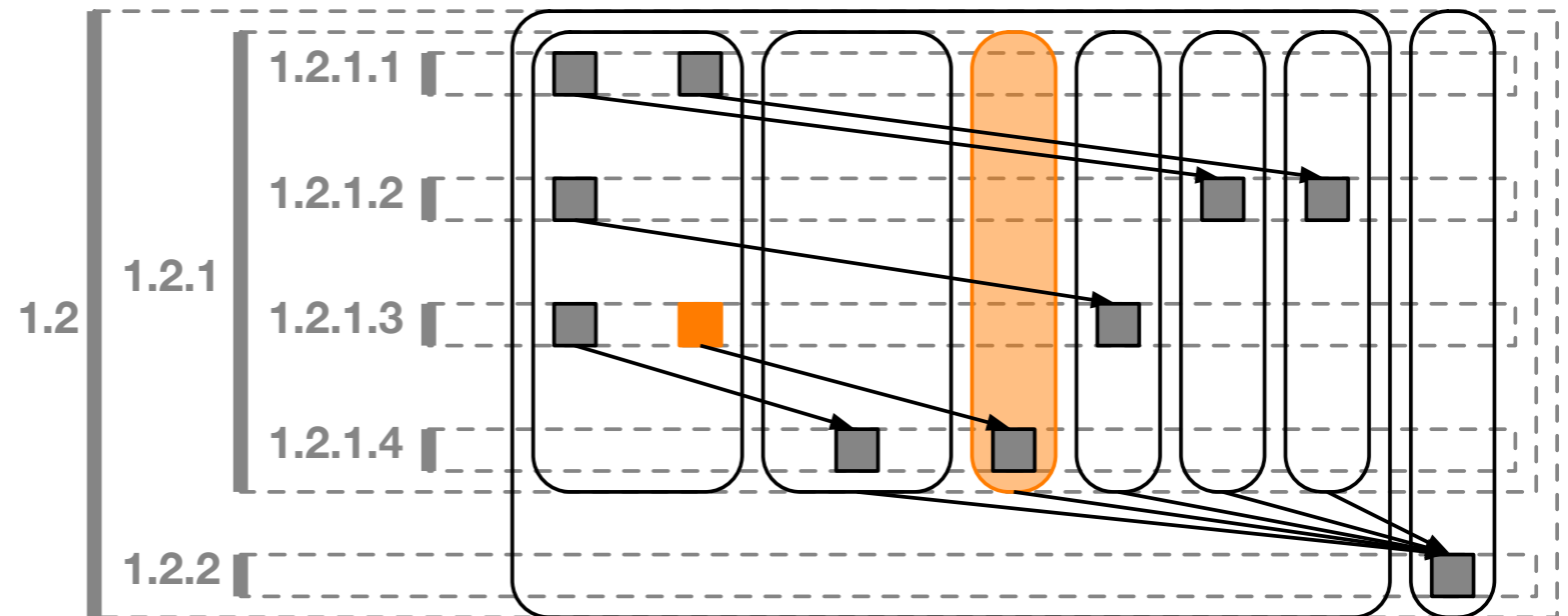


Aufspalten der Blöcke

Position der Blöcke fest

Ziel: Bestimmung der Ordnung der Hilfsknoten innerhalb jedes Blocks.

Kann aus der Ordnung der neuen Kinder abgeleitet werden.





Vierte Phase

Koordinatenberechnung



Prinzip

Ziel: Koordinaten und **Abmessungen** für jeden Knoten

Zweistufiger Algorithmus

- ✦ **Relative Koordinaten** für die Kinder jedes Knotens (bottom-up)

- ✦ **Absolute Koordinaten** durch Aufsummieren (top-down)

Zurückführung auf Koordinatenberechnung in hierarchischen
Zeichnung von normalen gerichteten azyklischen Graphen.

- ✦ **Viele bekannte Verfahren** einsetzbar

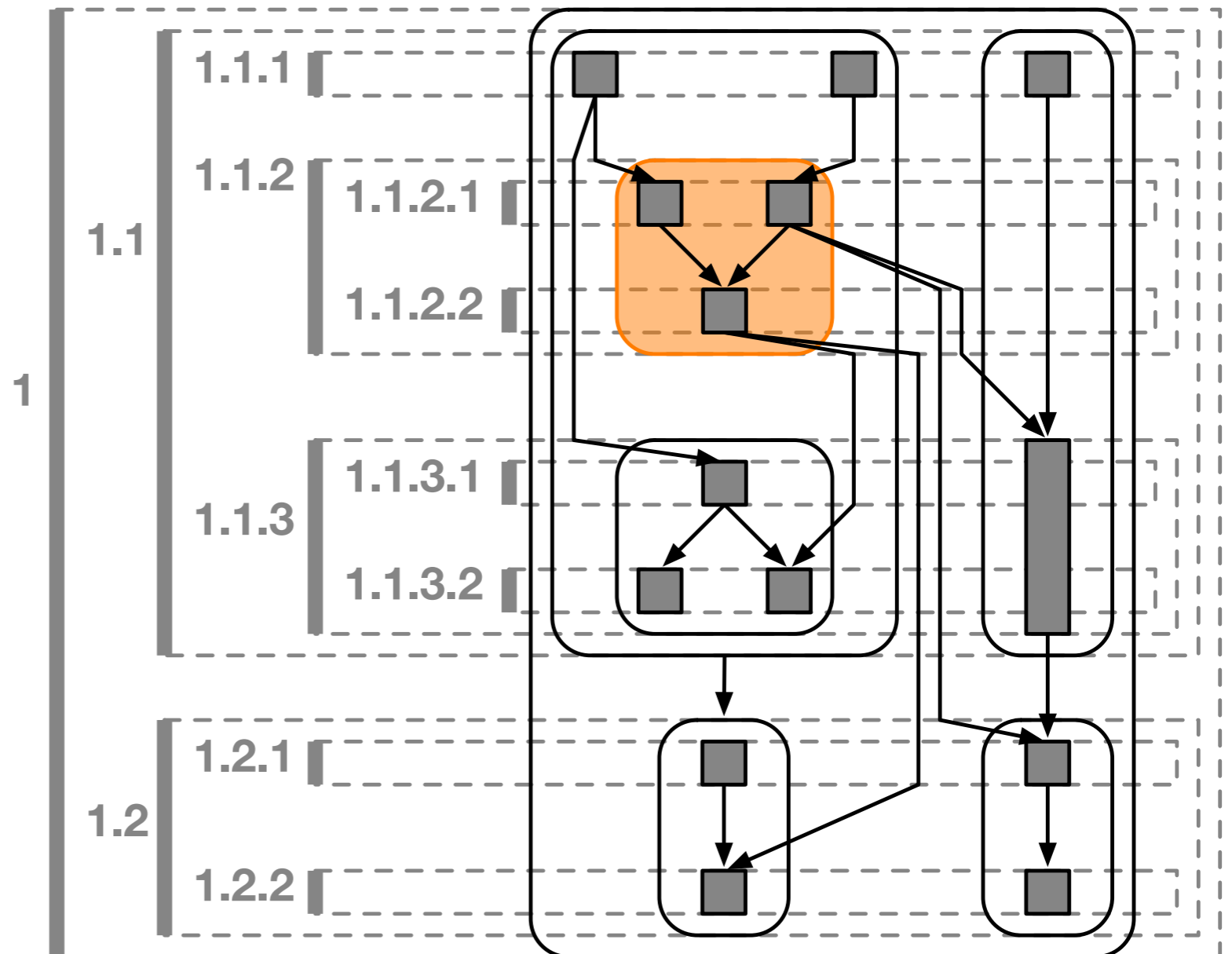
- ✦ **Hier:** [Brandes und Köpf, 2001]

Wiederverwendung

**Expandierter Knoten
ändert Höhe und Breite**

**Folge: Anpassung von
Koordinaten und
Abmessungen von
Geschwistern und
Vorgängern notwendig**

**Koordinaten und
Abmessungen von
anderen Knoten werden
wiederverwendet**

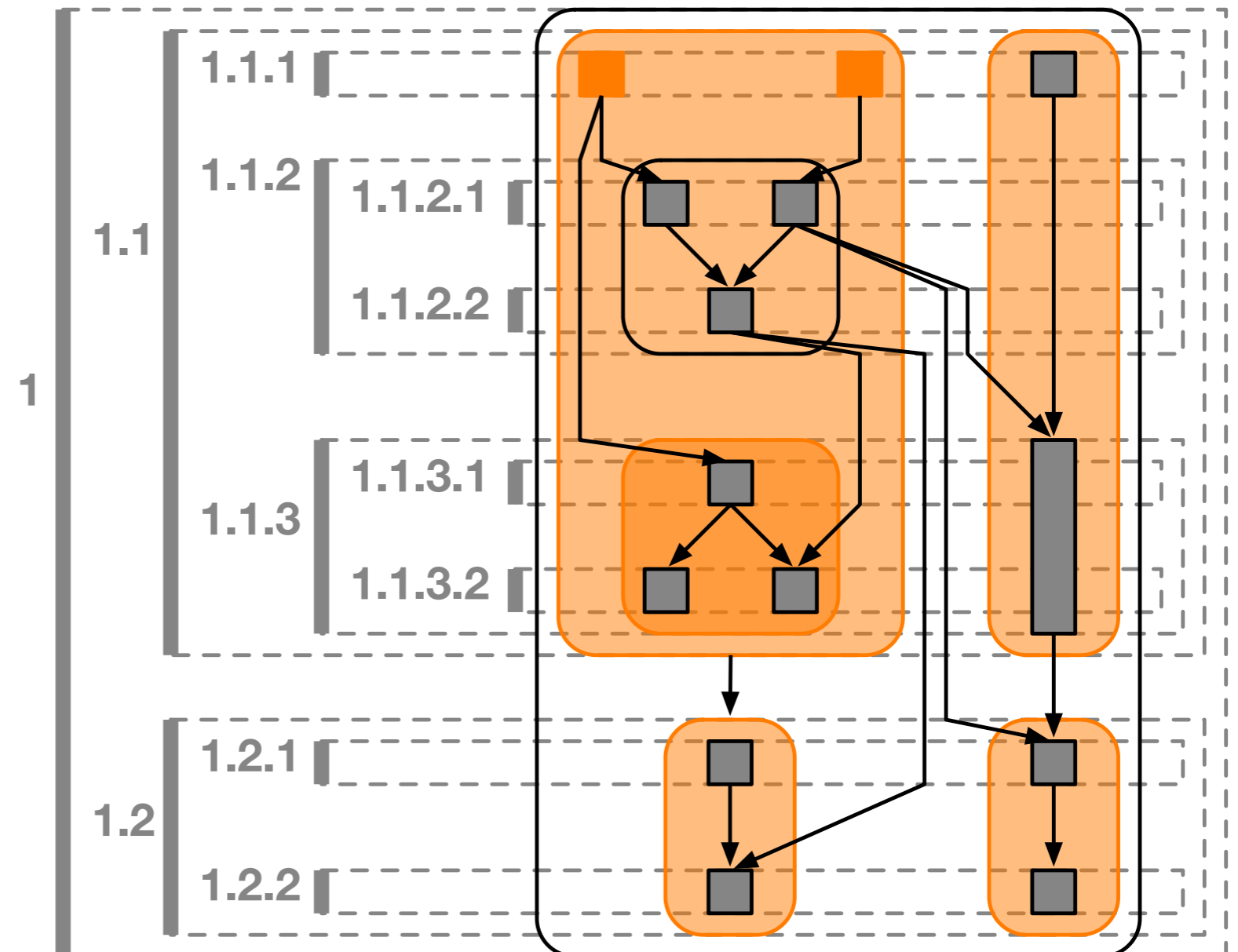


Wiederverwendung

Expandierter Knoten
ändert Höhe und Breite

**Folge: Anpassung von
Koordinaten und
Abmessungen von
Geschwistern und
Vorgängern notwendig**

Koordinaten und
Abmessungen von
anderen Knoten werden
wiederverwendet

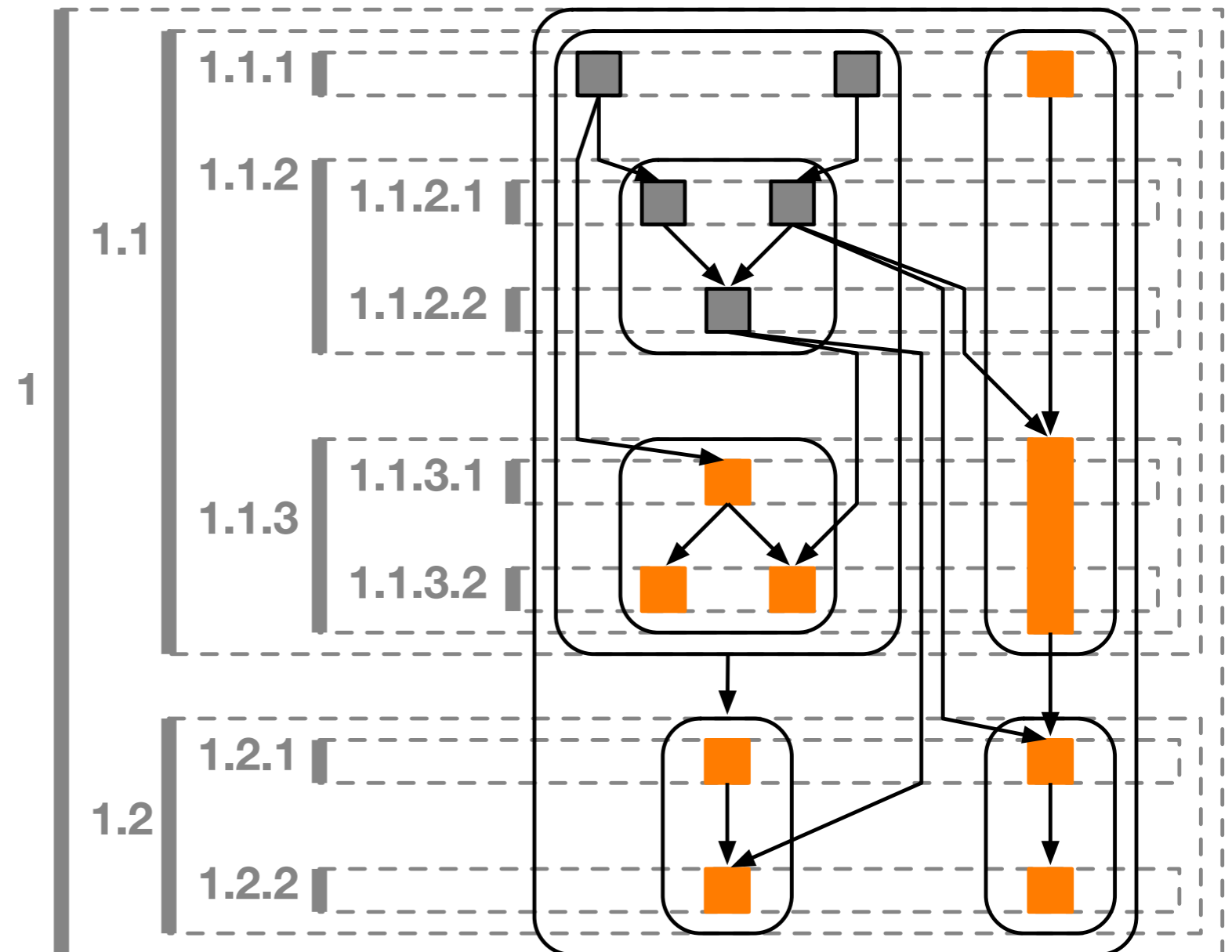


Wiederverwendung

Expandierter Knoten
ändert Höhe und Breite

Folge: Anpassung von
Koordinaten und
Abmessungen von
Geschwistern und
Vorgängern notwendig

Koordinaten und
Abmessungen von
anderen Knoten werden
wiederverwendet



Kontrahieren von Knoten

Nach Kontraktion: Knoten sind **Teilmenge** der vorherigen Knoten.

Update der ersten drei Phasen: **Einschränkung** auf diese Teilmenge.

Update von Koordinaten und Abmessungen: **analog zum Expandieren**

Nur möglich für Knoten die vorher bereits expandiert wurden.

Expandieren und Kontrahieren sind auch **visuell invers** zueinander.

Realisierung

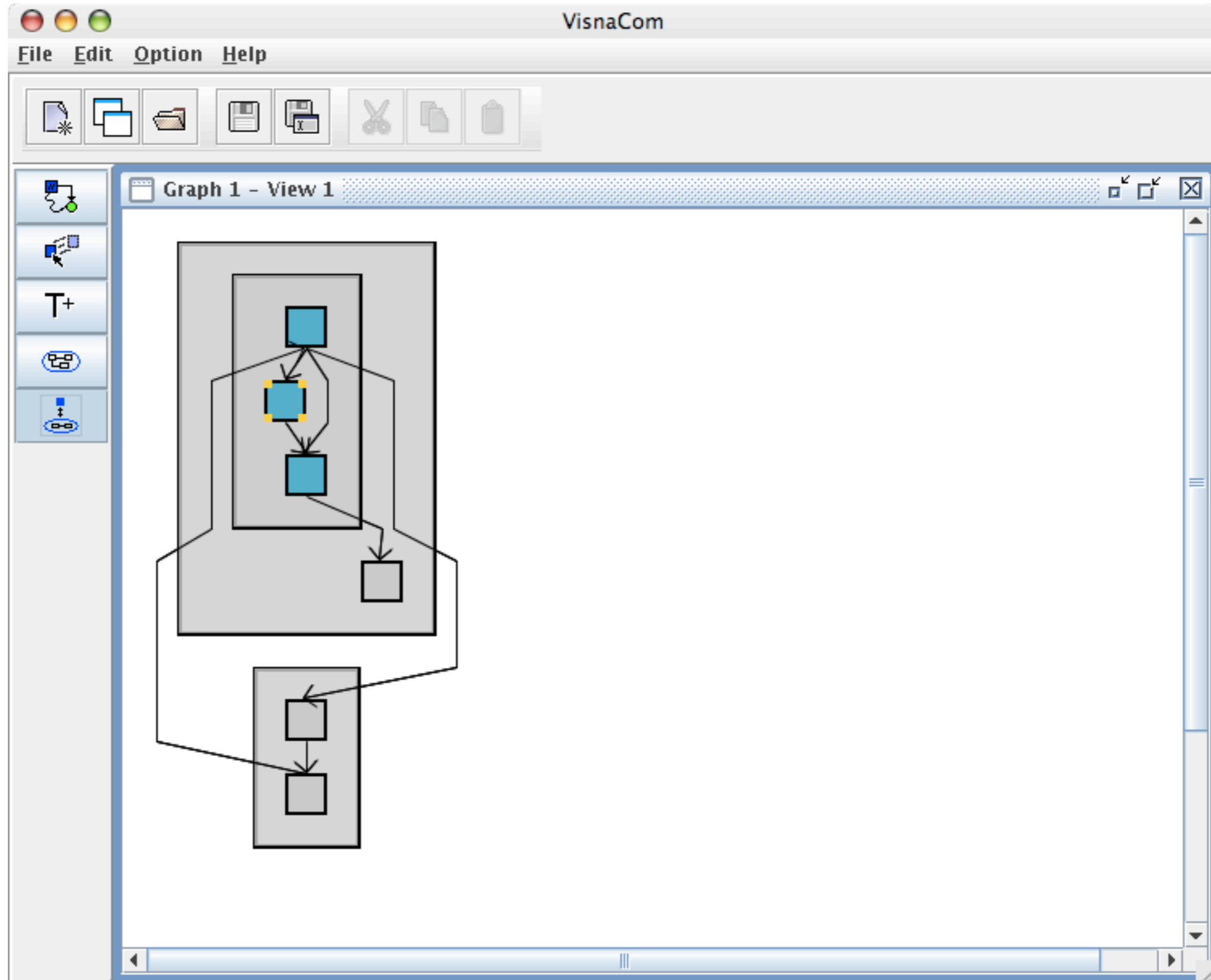
Vorstudie

Marcus Raitner: **HGV: A Library for Hierarchies, Graphs, and Views**. In: *10th International Symposium on Graph Drawing (GD), 2002*.

Prototyp (<http://www.infosun.fmi.uni-passau.de/VisnaCom/>)

Franz Pfeiffer. **Implementation eines Editors für Compound Graphen**.
Diplomarbeit, Universität Passau, 2005

Michael Pröpster. **Visuelle Navigation in Compound Graphen**.
Diplomarbeit, Universität Passau, 2005



Visualisierung

Experimentelle Bewertung

Verfahren

Experimentelle Analyse des Update-Schemas im Vergleich zu einem vollständigen Neulayout

Start mit **vollständig kontrahierter Sicht**

Knotenweise Expansion bis zu **vollständig expandierter Sicht**

Messgrößen:

- 🔺 **Durchschnittliche Laufzeit einer Expansion**
- 🔺 **Benötigte Zeichenfläche der vollständig expandierten Sicht**
- 🔺 **Anzahl der Kreuzungen in der vollständig expandierten Sicht**

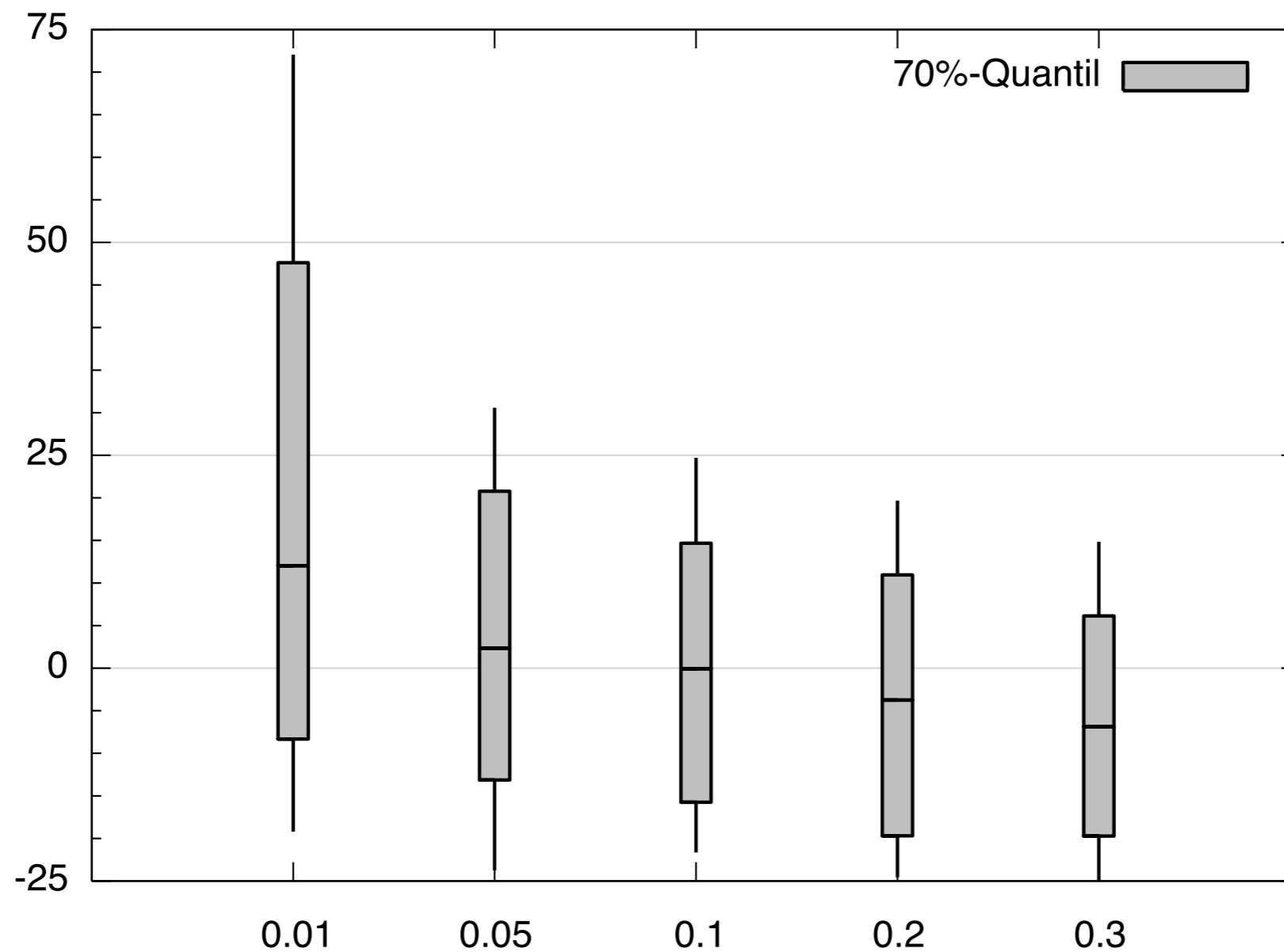
Eingaben

1500 zufällig generierte Compound Graphen

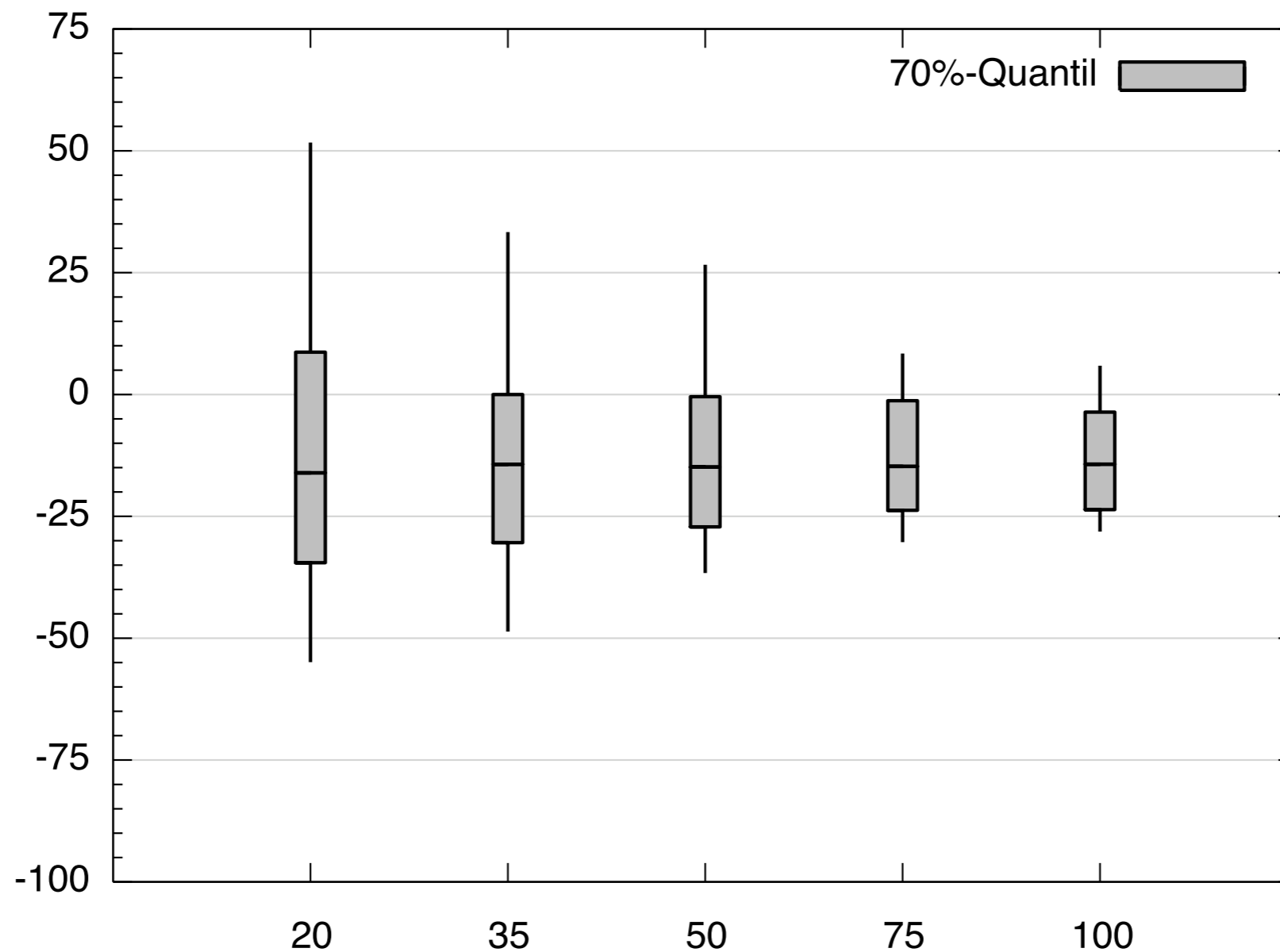
Parameter:

- 🍯 Knotenanzahl $n \in \{20, 35, 50, 75, 100\}$
- 🍯 Mittlerer Verzweigungsgrad des Inklusionsbaums $\gamma \in \{2, 4, 6, 8, 10, 15\}$
- 🍯 Dichte $\delta \in \{0.01, 0.05, 0.1, 0.2, 0.3\}$

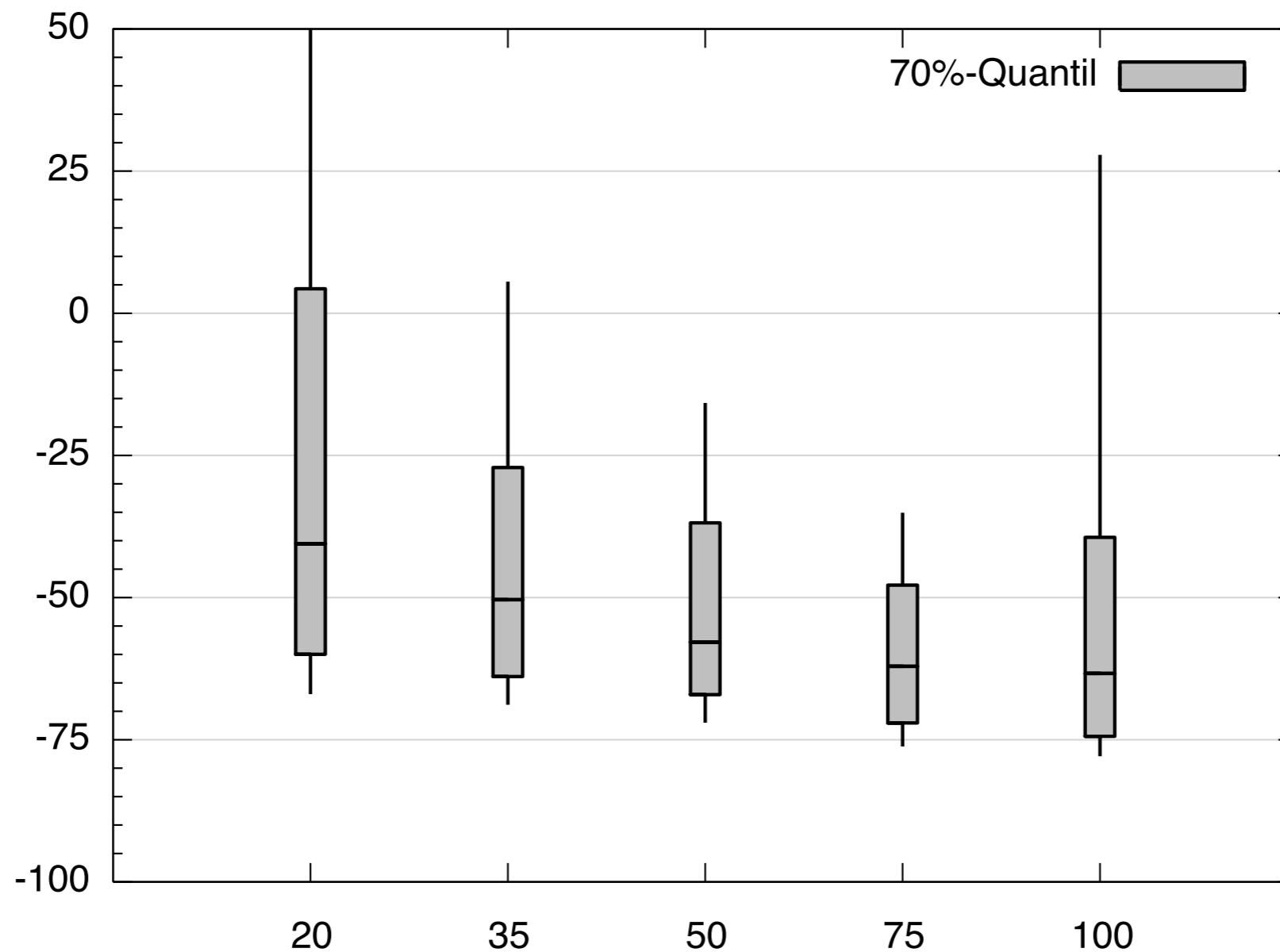
Zeichenfläche / Dichte



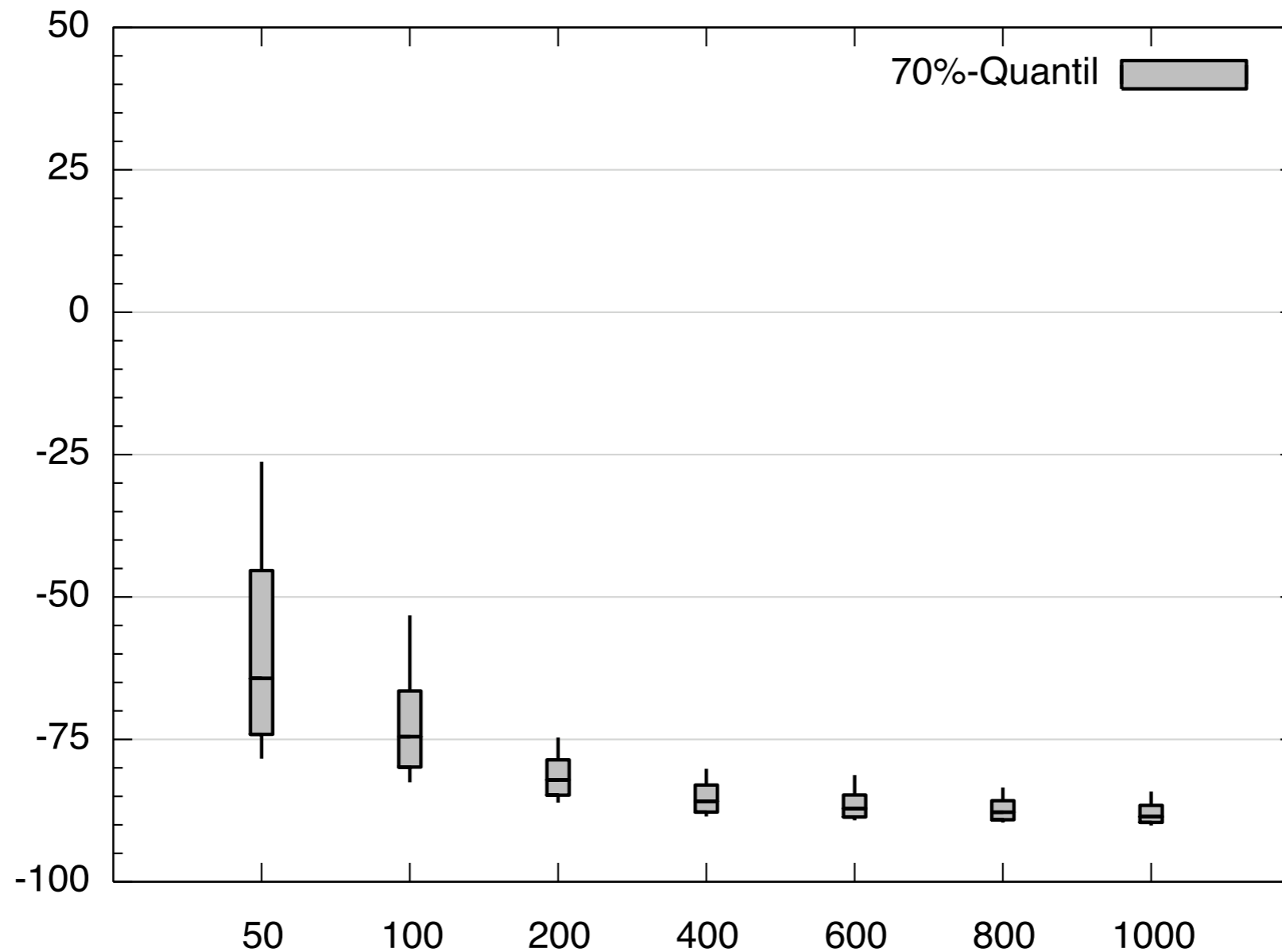
Kreuzungen / Knotenanzahl



Laufzeit / Knotenanzahl



Laufzeit / Knotenanzahl



Zusammenfassung

Datenstruktur

Dynamischere Variante von Baum-Kreuzprodukten

- ❖ Neu: **Anlegen** und **Löschen** von Blättern
- ❖ Adaptation von [Buchsbaum et al. '00]
- ❖ Nummerierung ersetzt durch **order maintenance** Datenstruktur

Anwendung auf **Graph View Maintenance**

- ❖ Neu: **Anlegen** und **Löschen** von Blättern
- ❖ Offenes Problem: Anlegen von inneren Knoten (Cluster).

Visualisierung

Verglichen mit einem vollständigen Neulayout...

...ist das Update-Schema deutlich **effizienter**:

Aktualisierungen aller “teuren” Schritte sind lokal.

...haben die aktualisierten Zeichnungen vergleichbare **Qualität**:

Im Durchschnitt weniger Kreuzungen bei gleicher Fläche.

...erhält das Update-Schema die **mental map** besser:

Knoten bleiben in ihren Schichten und in derselben Reihenfolge.

Expandierte Kanten verlaufen wie die zugehörigen kontrahierten.



