XPath

Markus Stocker

19. März 2018

Rekapitulation

- Wie kommt es, dass ein XML Dokument eine Baumstruktur hat?
- Warum ist die Festlegung auf ein gemeinsames Vokabular wichtig?
- Wie unterstützen Programmiersprachen die Verarbeitung von XML?
- Was bedeutet Wohlgeformtheit?

Übersicht

- Was is XPath
- XPath Konzepte
- Beispiele

What is XPath

- Mit XML kann man Daten strukturieren
- Man kann diese in Mensch- und Maschinenlesbarer form speichern
- Nicht nur in Dateien sondern auch in Datenbanken
- Toll, aber letztlich muss man diese Daten flexibel verarbeiten können
- Dafür immer Programme zu schreiben ist mühsam
- Man benötigt Verarbeitungssprachen

What is XPath

- Sprache zur Verarbeitung von XML Dokumenten
- Auf Teile eines XML Dokumentes zugreifen
- Navigation durch Elemente und Attribute
- Selektion von Elementen und Inhalten
- Einfache Operationen auf Inhalten

Knoten (*node*)

- XPath modelliert ein XML Dokument als Knotenbaum
- Es gibt 7 Knotenarten
  - *Root node*, der virtuelle Elternknoten des Wurzelelements
  - *Element node*, z.B. `<planet>`
  - *Attribute node*, z.B. `radius="6371"`
  - *Text node*, `CDATA`
  - *Namespace node*
  - *Comment node*
  - *Processing instruction node*
- Davon sind `element`, `attribute`, `text` die drei wichtigstens

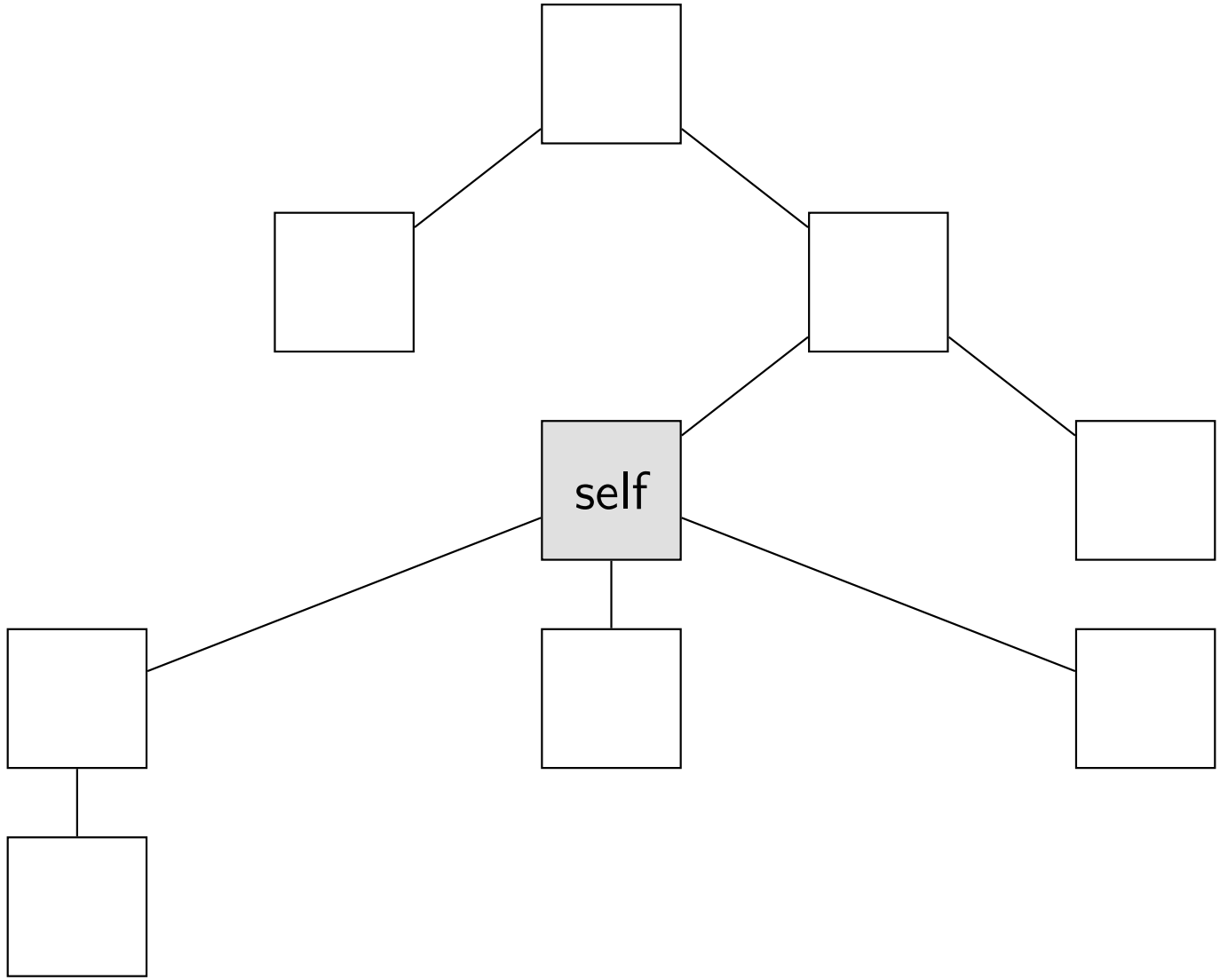
Achsen (*axis*)

- Navigation mittels XPath erfolgt von einem Kontextknoten
- Achsen bezeichnen Verwandtschaftsverhältnisse zum Kontextknoten
- Dreizehn unterschiedlichen Achsen

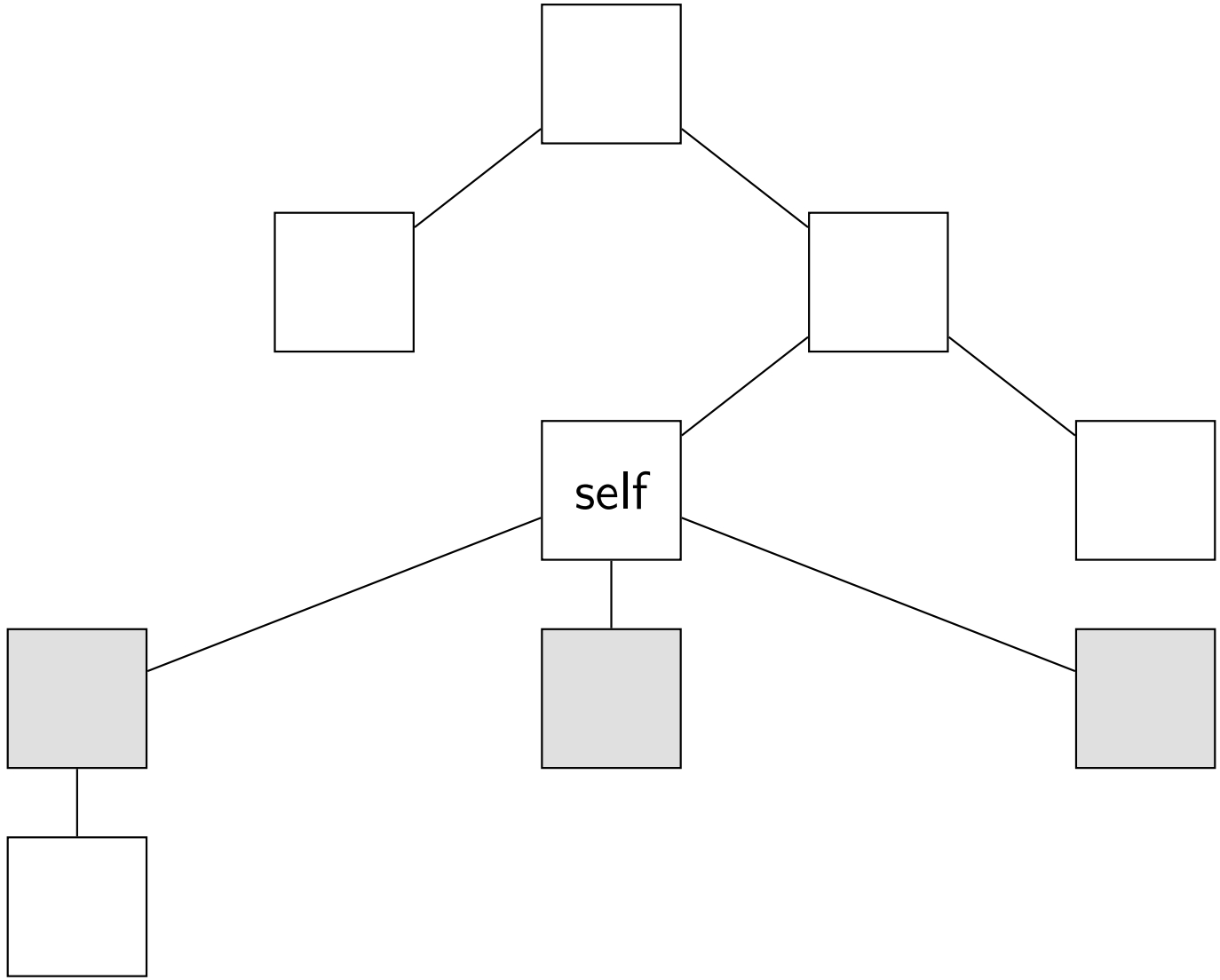
Achsen: Die Wichtigsten im Überblick

- *self*: Kontextknoten selbst
- *child*: Kindelemente des Kontextknotens
- *parent*: Elternelement des Kontextknotens
- *preceding-sibling*: Vorgängige Geschwister des Kontextknotens
- *following-sibling*: Nachfolgende Geschwister des Kontextknotens
- *anchestor*: Vorfahren des Kontextknotens
- *descendant*: Nachkommen des Kontextknotens

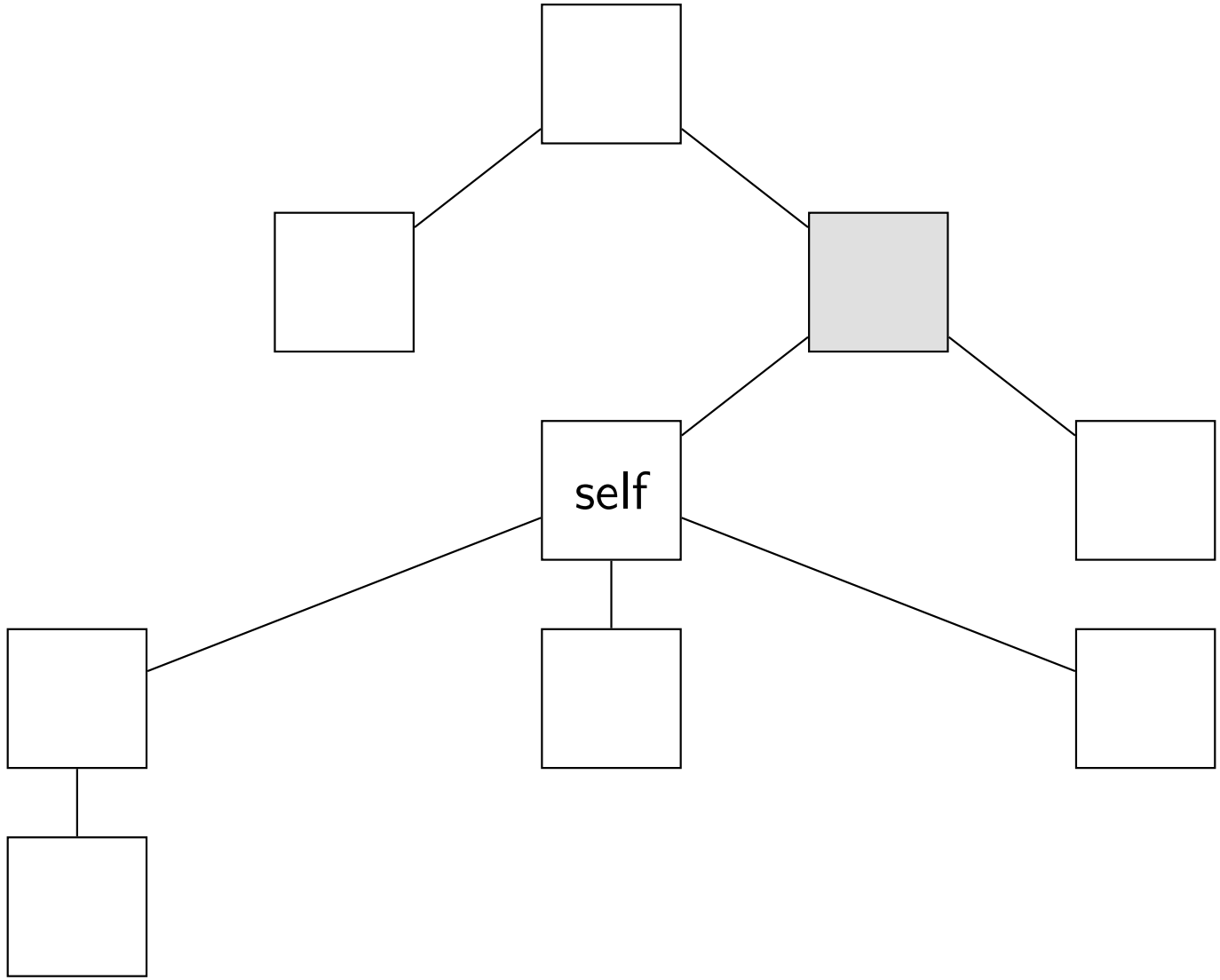
Achsen: *self*, der Kontextknoten



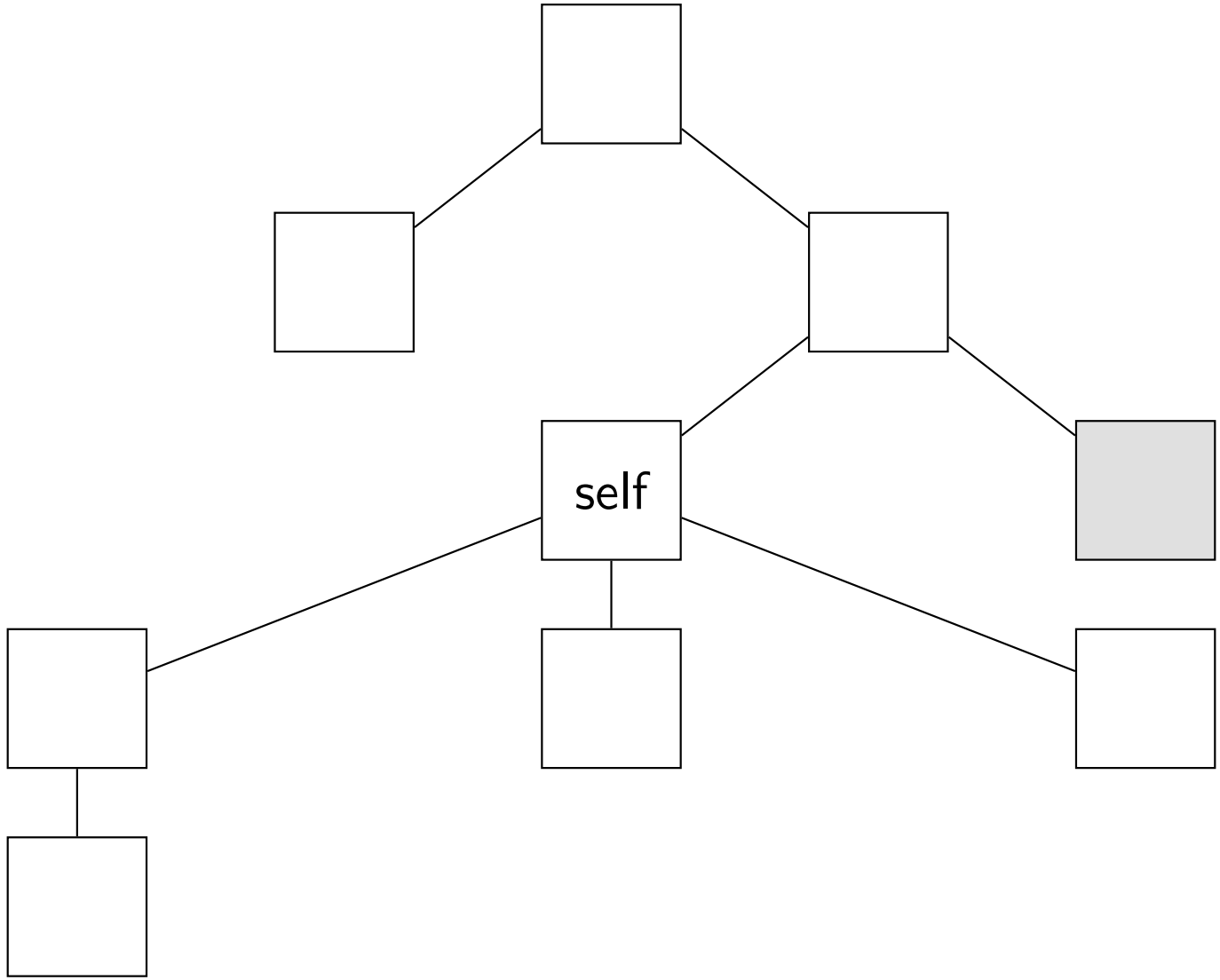
Achsen: *child*, die Kinder



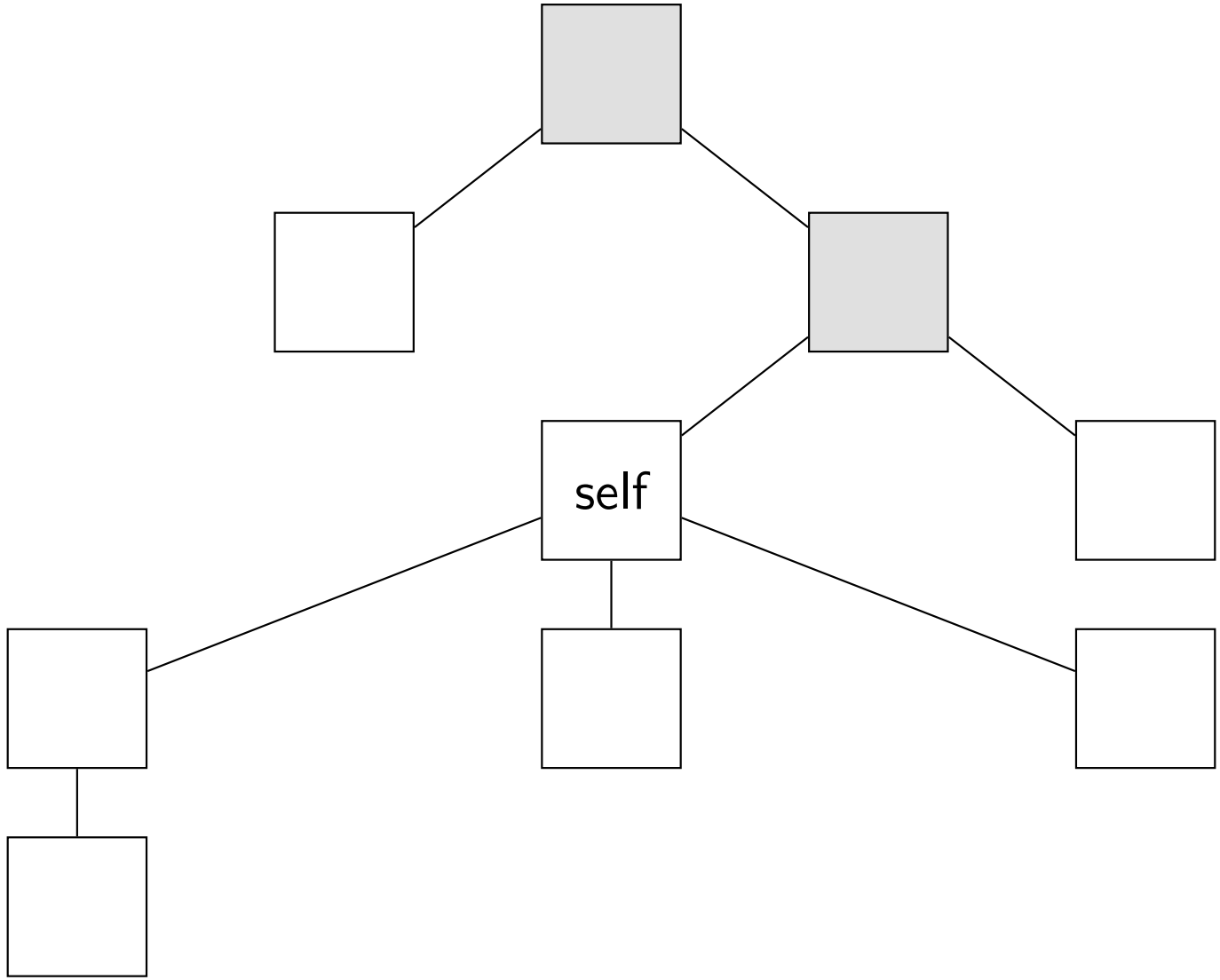
Achsen: *parent*, das Elternteil



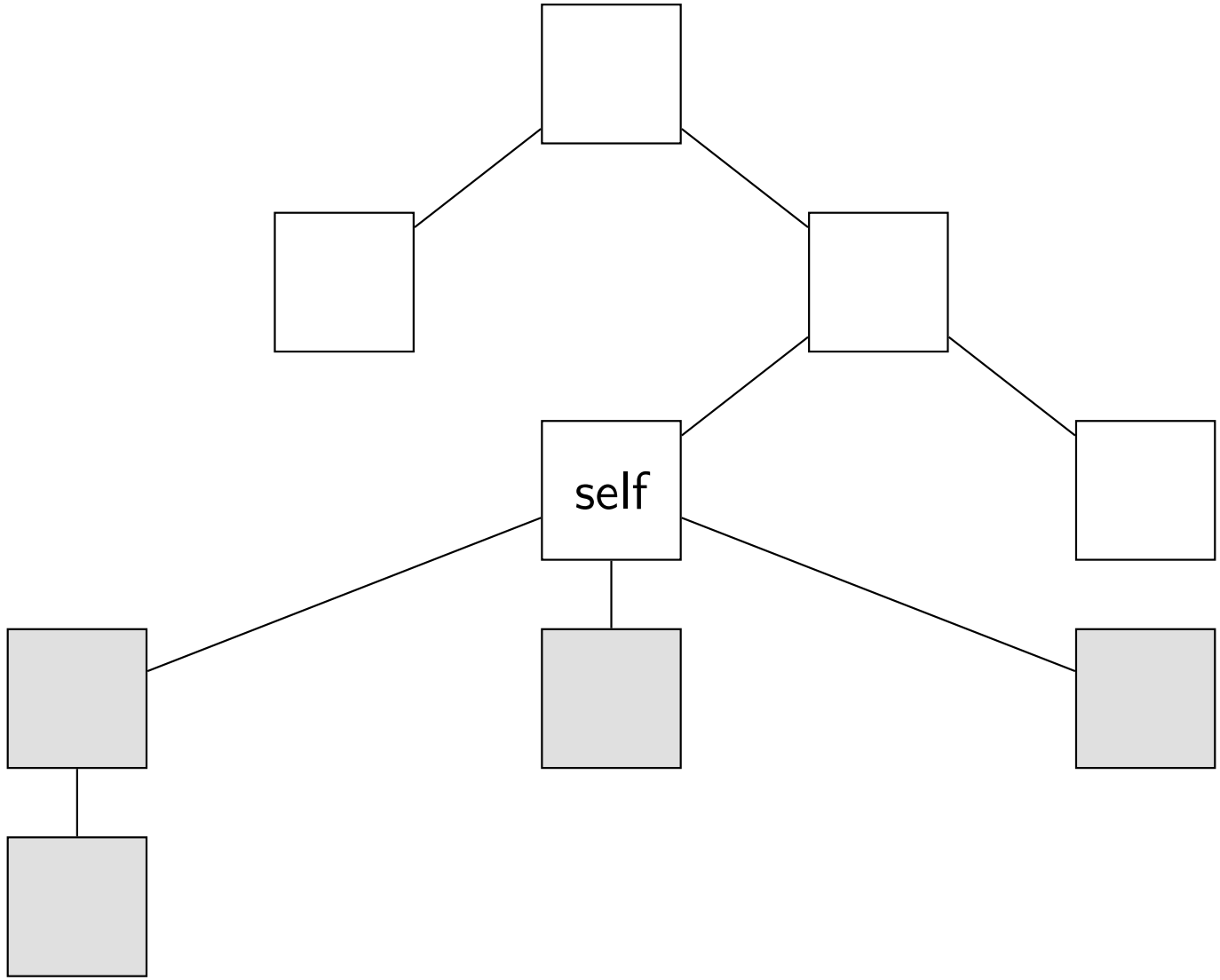
Achsen: *following-sibling*, die nachfolgende Geschwister



Achsen: *anchestor*, die Vorfahren



Achsen: *descendant*, die Nachkommen



Lokalisierungspfade

- Ermöglichen die Adressierung von Knoten oder Knotenmengen
- Setzen sich aus mehreren Einzelschritten zusammen
- Diese werden durch Schrägstriche (/) voneinander getrennt

Lokalisierungspfade: Schritt (*location step*)

- Bestandteile eines Schrittes
  - Achse: Richtung in die aus dem Kontextknoten navigiert wird
  - Knotentest: Der gewünschte Knoten
  - Prädikat: Null, eins oder mehrere Filterbedingungen

```
Achse::Knotentest[Prädikat]*
```

Schritt: Beispiele

```
<planets>
  <planet>
    <name radius="6371">Earth</name>
  </planet>
</planets>

child::planets
descendant::planet[name="Earth"]
descendant::*
```

Lokalisierungspfade

- Kommen in ausführlicher oder verkürzter Form vor
- Können relativ oder absolut sein

Ausführliche und verkürzte Formen: Beispiele

Ausführliche Form	Verkürzte Form
`child::`	*Unterlassen*
`attribute::`	`@`
`descendant-or-self::node()/`	`//`
`self::node()`	`.`
`parent::node()`	`..`

Absolute Lokalisierungspfade

- Ausgehend vom Wurzelknoten (nicht Wurzelelement)
- Der erste Schrägstrich referenziert den Wurzelknoten
- Ermöglicht auch die Adressierung von Kommentare

```
<planets>
  <planet>
    <name>Earth</name>
    <radius>6371</radius>
  </planet>
</planets>
```

```
/child::planets/child::planet/child::name
/planets/planet/name
```

Relative Lokalisierungspfade

- Relative Pfade benötigen einen Kontextknoten
- Der Pfad wird relativ zu diesem Knoten ausgewertet

```
<planets>
  <planet>
    <name>Earth</name>
    <radius>6371</radius>
  </planet>
</planets>
```

```
child::planet/child::name
planet/name
```

Adressierung verschiedener Knotentypen

`planet/name/text()`	Textknoten des Elements `name`
`planet/node()`	Alle Knotentypen (ausser Attribute)
`planet/comment()`	Kommentarknoten des Elements `planet`

Prädikate

- XPath Ergebnisse filtern
- Ermöglicht komplexere Problemstellungen
- Indem eine genauere Zielmenge definiert werden kann
- Ausdrücke die einen booleschen Wert liefern
- Also, wahr oder falsch
- Resultierende Knoten müssen auf Prädikate mit *wahr* testen
- XPath Ausdruck liefert eine Knotenmenge
- Diese wird mittels Prädikat Tests nochmals eingeschränkt

Prädikate: Beispiel

```
<planets>
  <planet>
    <name radius="6371">Earth</name>
  </planet>
  <planet>
    <name>Mars</name>
  </planet>
</planets>

/planets/planet/name[@radius]
/planets/planet/name[@radius = "6371"]
planet/name[@radius = 6371]
planet/name[@radius="6371"]/text()
planet[name]
```

Prädikate: Boolsche Operatoren

- Prädikate können boolsche Operatoren enthalten
- Insbesondere die Operatoren `and` und `or`

```
<planet>
  <name radius="6371" temperature="14.9">Earth</name>
</planet>
```

```
planet/name[@radius and @temperature]
planet/name[@radius=6371 and @temperature=14.9]
```

Prädikate: Boolsche Operatoren

```
<planets>
  <planet>
    <name radius="6371" temperature="14.9">Earth</name>
  </planet>
  <planet>
    <name radius="3389">Mars</name>
  </planet>
</planets>

planet/name[@radius=3389 and @temperature=14.9]
planet/name[@radius=3389 or @temperature=14.9]
```

Kaskadierende Prädikate

- Hintereinanderschaltung von Filtern
- Eine Knotenmenge wird gefiltert
- Es resultiert ein Ergebnis
- Dieses ist Ausgangsmente für das nächste Prädikat

```
<planet>
  <name>Earth</name>
  <radius>6371</radius>
  <temperature>14.9</temperature>
</planet>
```

```
/planet[name="Earth"][radius=6371]
```

Vereinigungsmengen: Mehrere Knotentests

```
<planets>
  <planet>
    <name>Earth</name>
    <radius>6371</radius>
  </planet>
  <planet>
    <name radius="3389">Mars</name>
  </planet>
</planets>
```

```
planet[name="Earth"] | planet/name[@radius=3389]
```

Funktionen

- Erweiterte Operationen und Abfragen auf Knotentests
- Stringanalyse von Textknoten und Attributwerten
- Verwendung in XPath-Ausdrücken oder Prädikaten
- Funktionen können ein Argument erhalten
- Geben immer einen Wert zurück

Funktionen: Beispiele

```
<planets>
  <planet>
    <name>Earth</name>
    <radius>6371</radius>
  </planet>
  <planet>
    <name radius="3389">Mars</name>
  </planet>
</planets>
```

```
planet[position()=2] (oder auch planet[2])
planet[starts-with(name, 'E')]/radius/text()
planet[not(radius)]/name/text()
count(planet)
planet[last()]
```

Zusammenfassung

- XPath unterstützt die Verarbeitung von XML
- Gezielt auf Inhalte zugreifen
- Adressierung mittels Lokalisierungspfade
- Resultierende Knotenmenge kann weiter eingeschrenkt werden