Core-Graphen

ein Graphgeneratoren-Praktikum von Thorsten Vogel und Sergej Müller
am Lehrstuhl für Algorithmik
13. Februar 2006

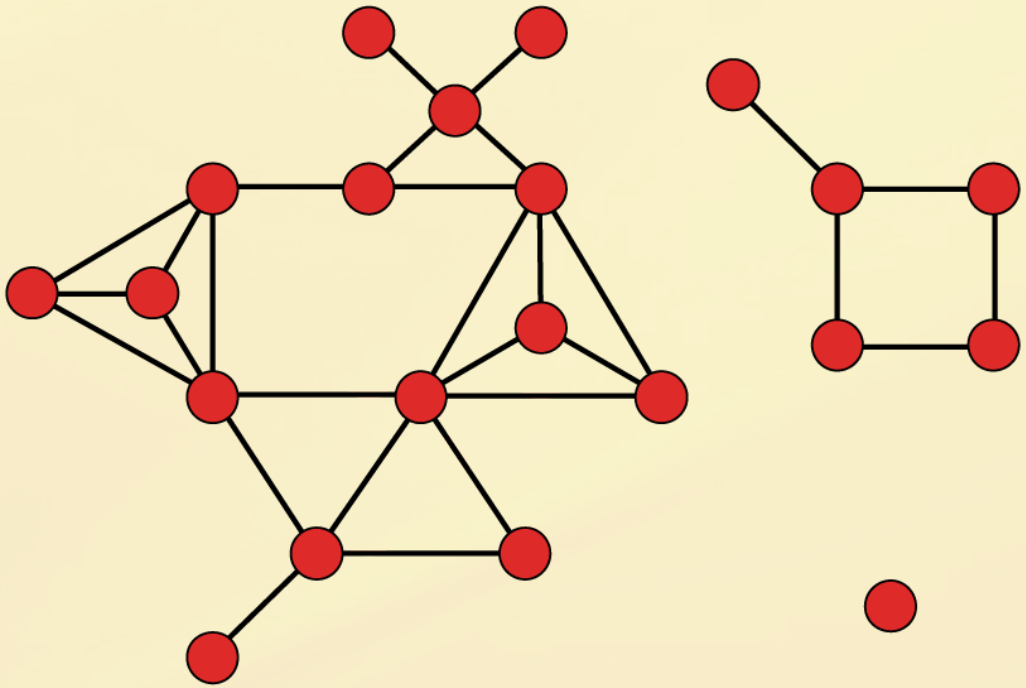
Zielsetzung

Durch Dekomposition von Graphen, beruhend auf k-Cores nach S. B. SEIDMAN (1983), lassen sich ihre strukturellen Eigenschaften leichter erkennen.

- Aufgabe:
  - Generieren eines Graphen mit Core-Struktur

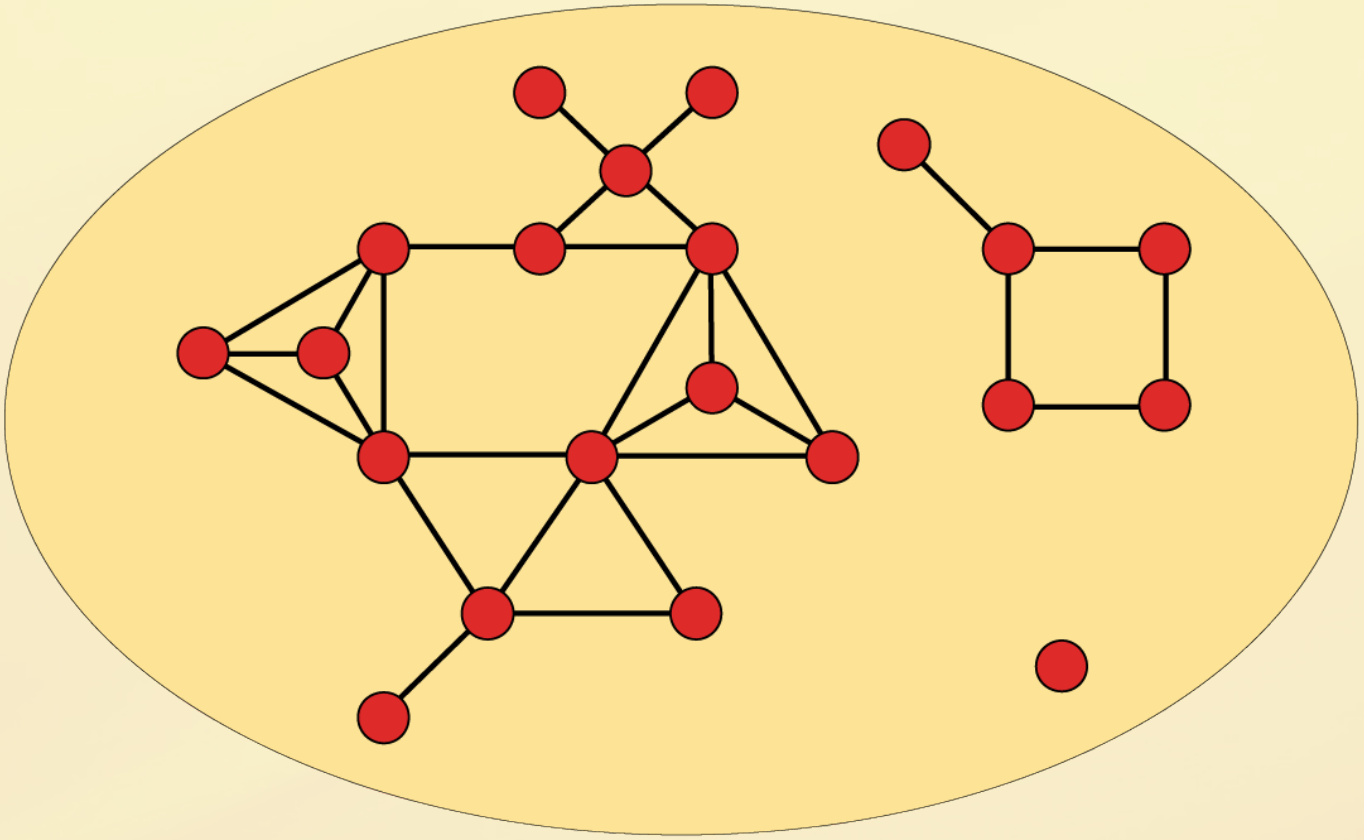
    Parameter:
    - *k* : maximaler k-Core
    - *n* : Anzahl der Knoten

Was sind k-Cores ?



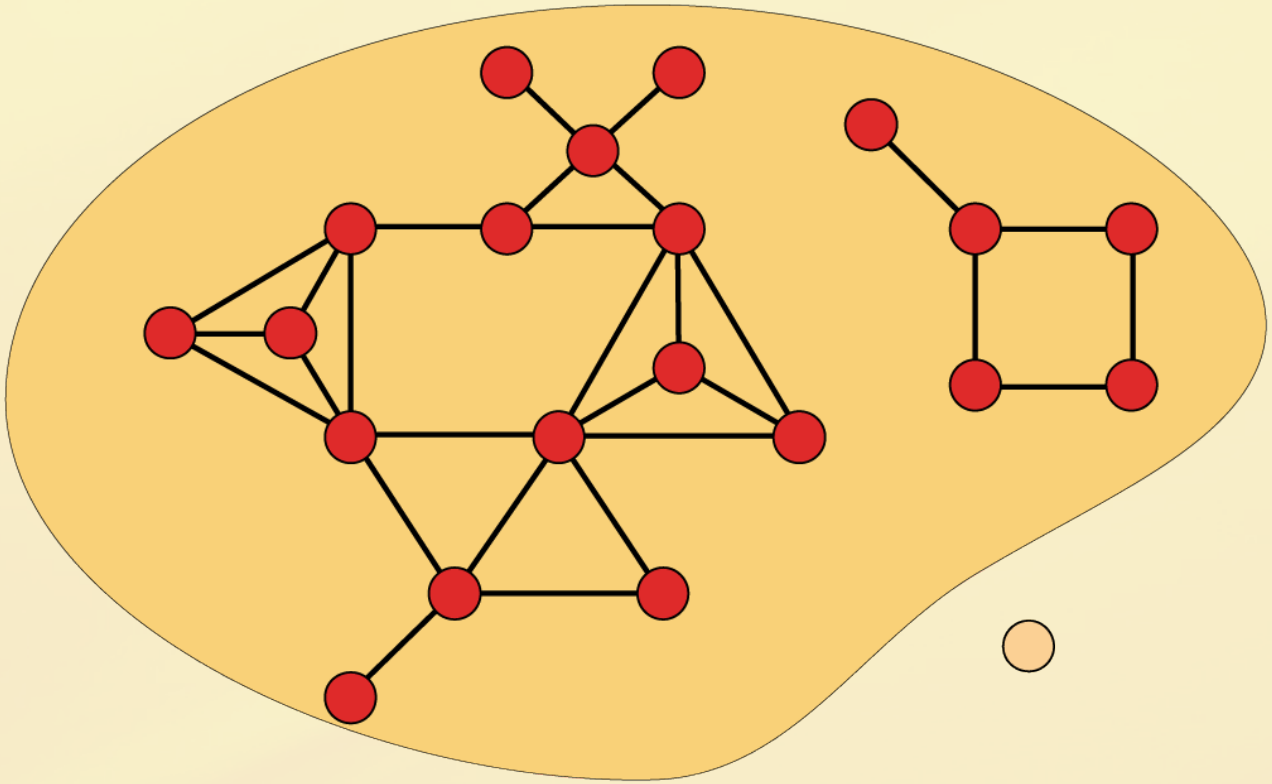
Gegebener Graph

Was sind k-Cores ?



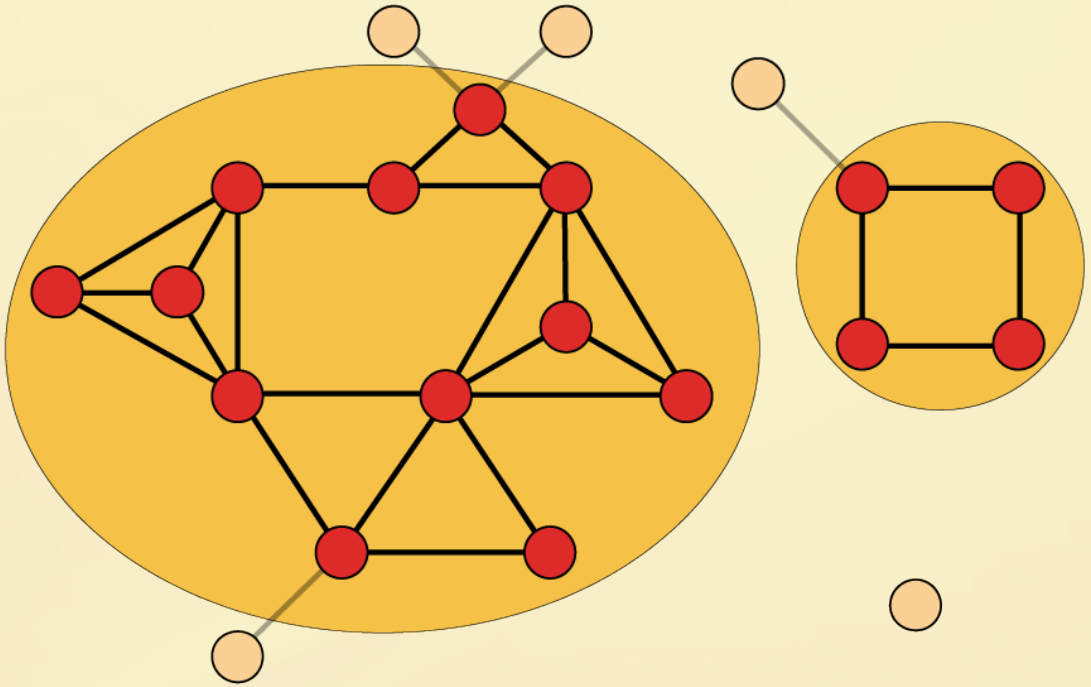
Alle Knoten befinden sich im 0-Core

Was sind k-Cores ?



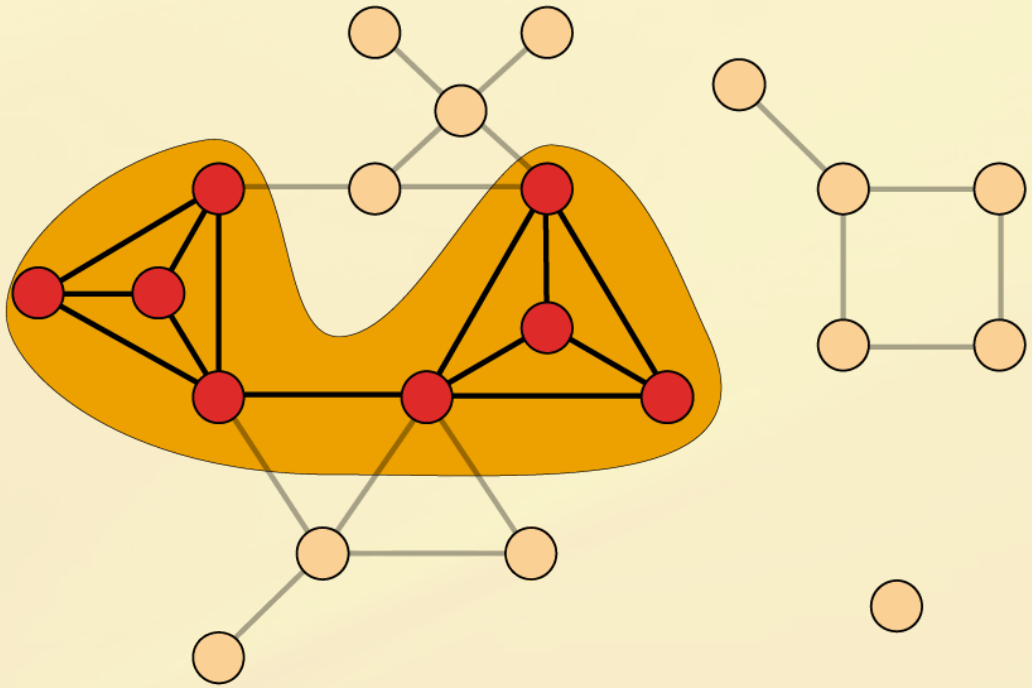
Entferne iterativ alle Knoten mit Grad < 1, Rest liegt im 1-Core

Was sind k-Cores ?



Entferne iterativ alle Knoten mit Grad < 2, Rest liegt im 2-Core

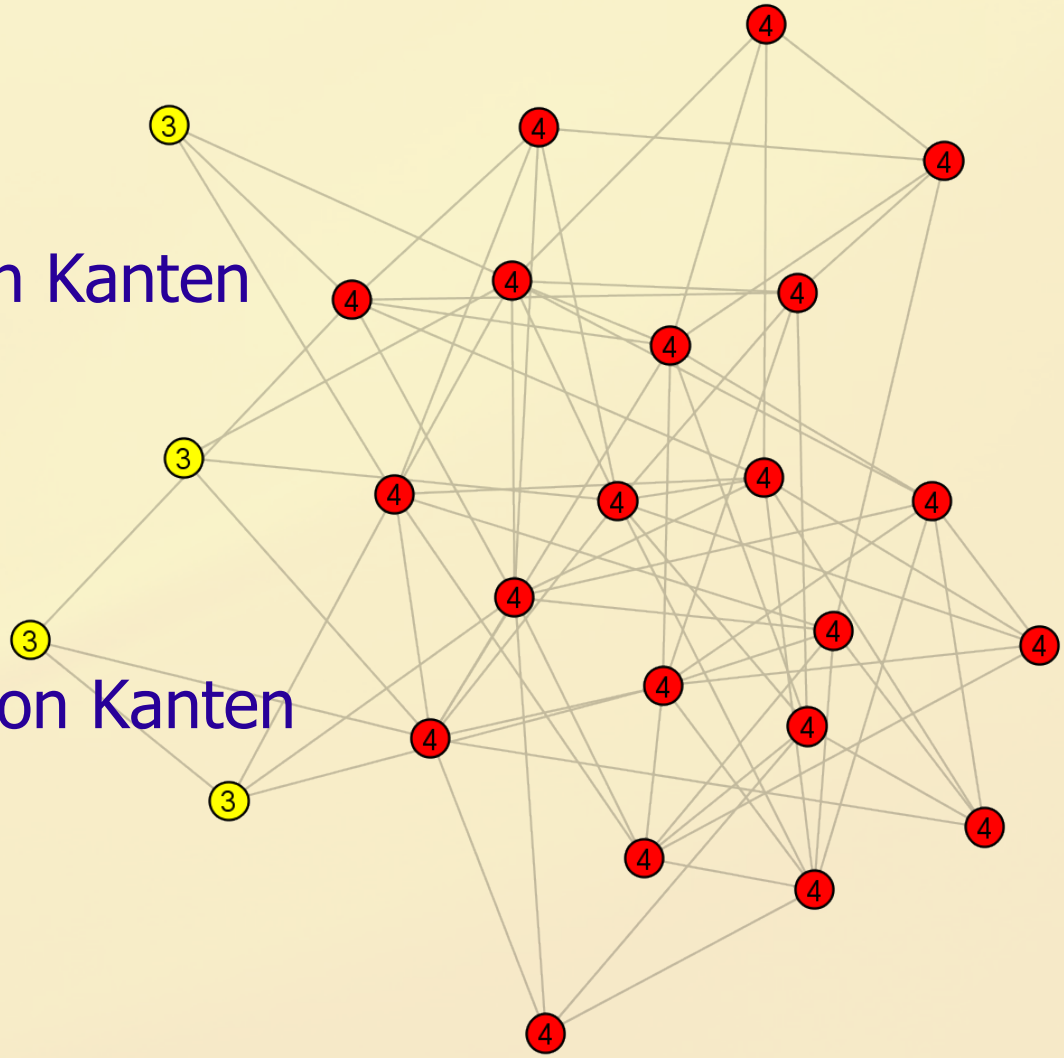
Was sind k-Cores ?



Entferne iterativ alle Knoten mit Grad < 3, Rest liegt im 3-Core

Erste Schritte

- Reduction
  - Vollständiger Graph
  - Iteratives Entfernen von Kanten
- Variation
  - Leerer Graph
  - Iteratives Hinzufügen von Kanten

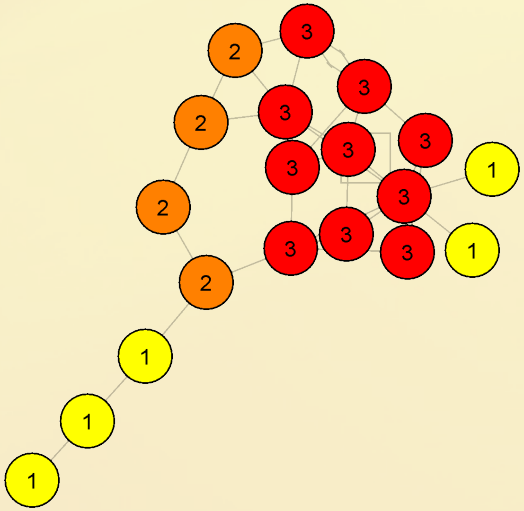


Problem

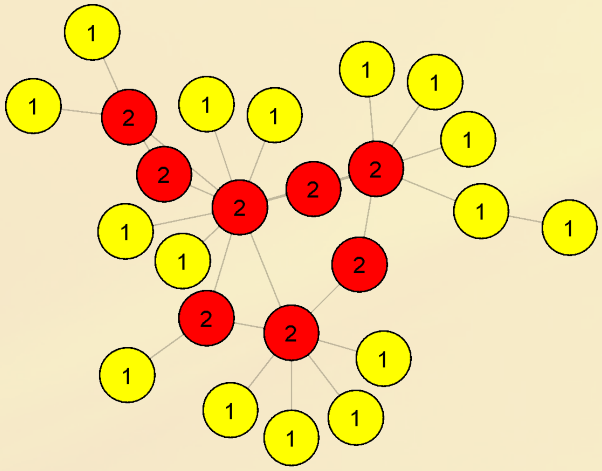
- Graph enthällt fast ausschließlich Knoten aus max. k-Core

- Weitere Informationen:
  - Size and connectivity of the k-core of a random graph, T. Luczak (1991)
  - Sudden emergence of a giant k-core in a random graph, B. Pittel, J. Spencer, N. Wormald (1996)

Weitere Ansätze

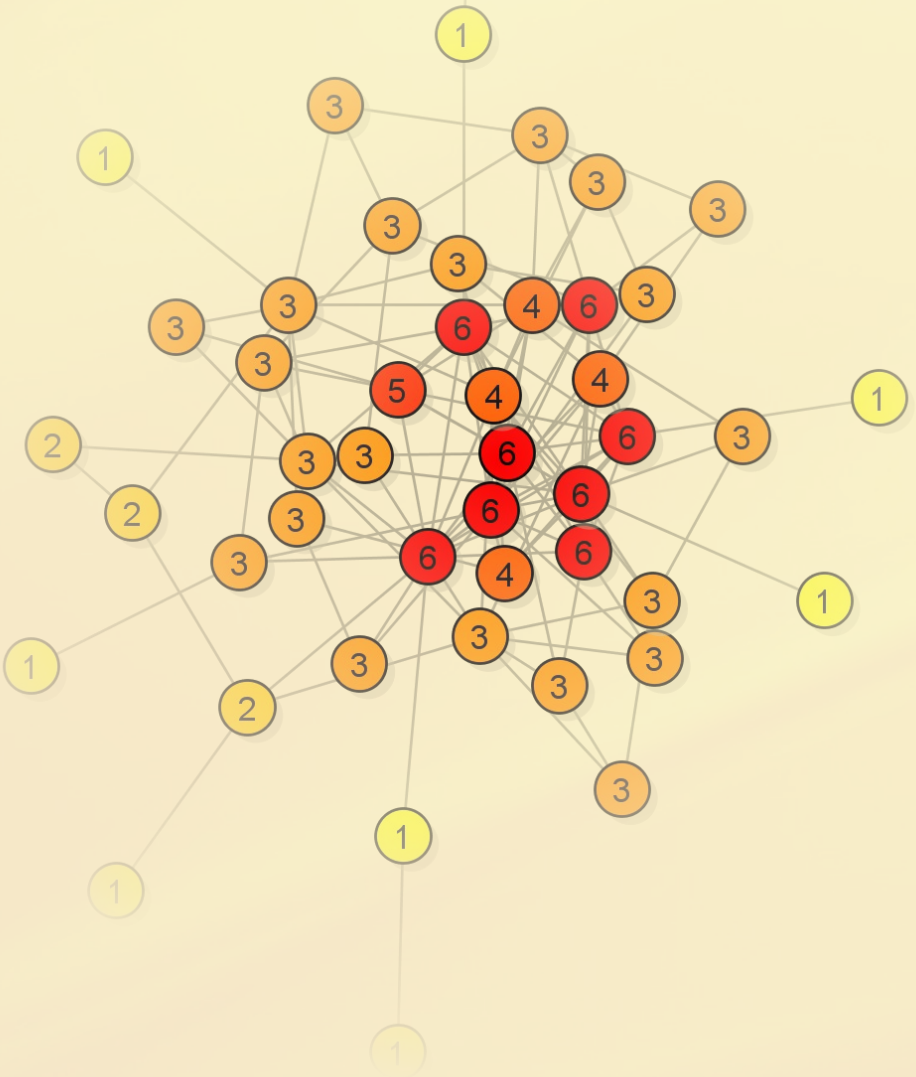


- Preferential Attachment
  - Bevorzugtes Anfügen nach Grad des Knotens



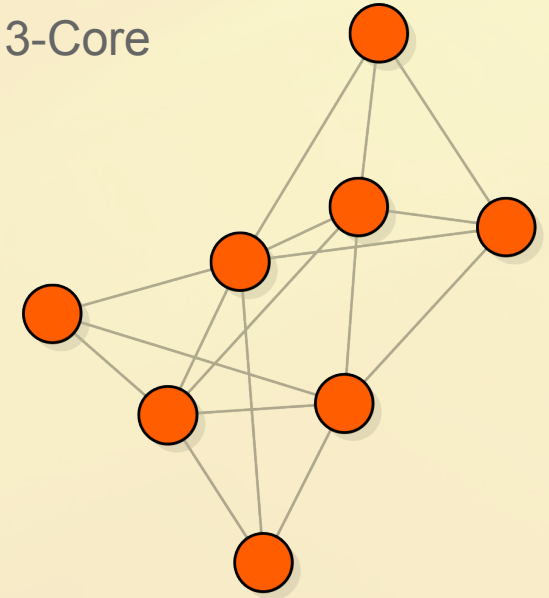
- Budget Attachment
  - Bevorzugtes Anfügen nach Grad und max. Budget des Knotens

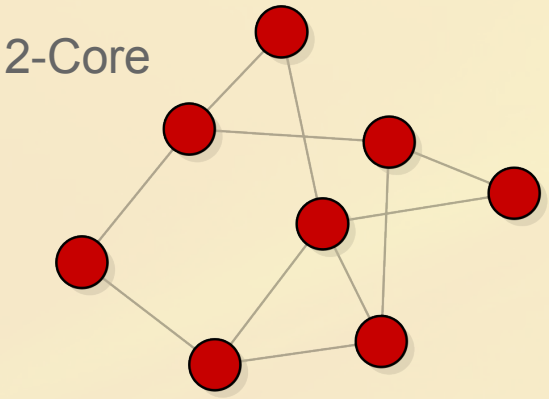
Feststellung



- k-Core Subgraph mit min. k+1 Knoten
  - Hier: 6-Core mit 8 Knoten
- Erzeugung aufteilen
  - Subgraph generieren
  - Satelliten hinzufügen

Strict Core-Builder



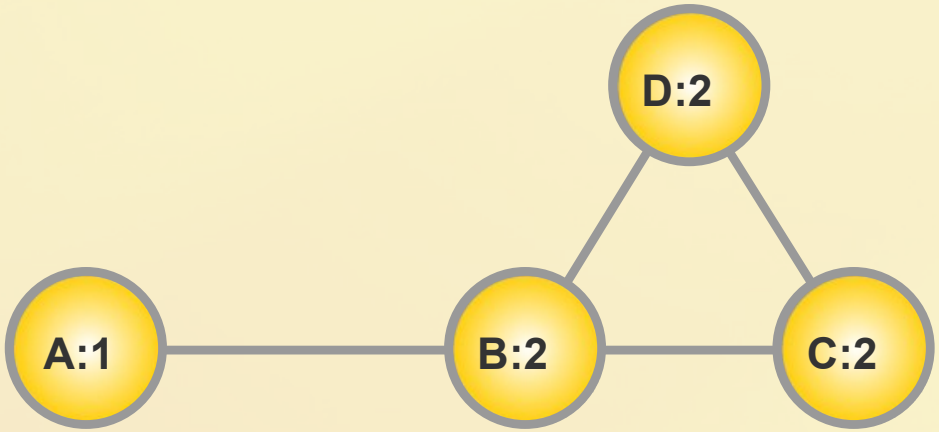


- Generator für Graph mit k-Core
  - Erstelle vollständigen Graphen mit k+1 Knoten
  - Füge iterativ neue Knoten hinzu
  - Lasse zufällig überflüssige Kanten weg
- Vorteil
  - Keine Berechnung der Core-Nummern nötig

PartyPeole

- Anreicherung des Strict Cores durch neue Knoten
- Rekursive Berechnung der Core-Nummern
- Ausnutzung der Lokalität des Umkippens
- Verteilung von Einladungen zu einer k-Core-Party
- Teilnahme möglich, falls
  - Knoten hat Core-Nummer $\geq k$
  - Knoten bekommt k Zusagen

PartyPeople - Beispiel

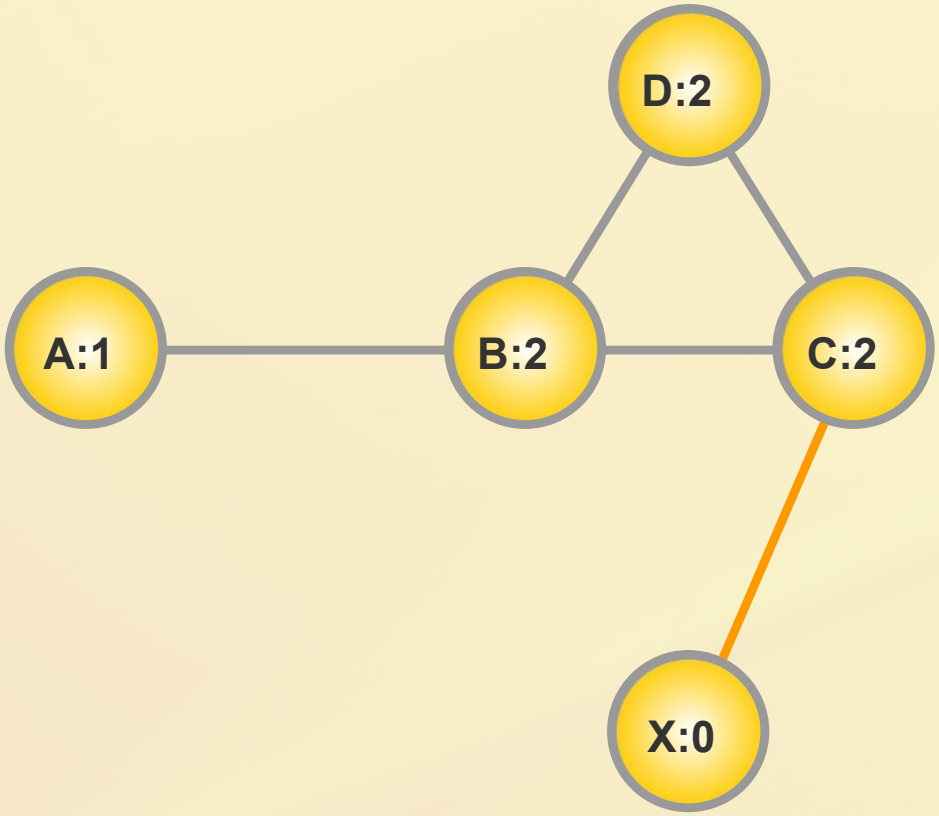


- Neuer Knoten **X**

Einladungsliste:

[]

PartyPeople - Beispiel

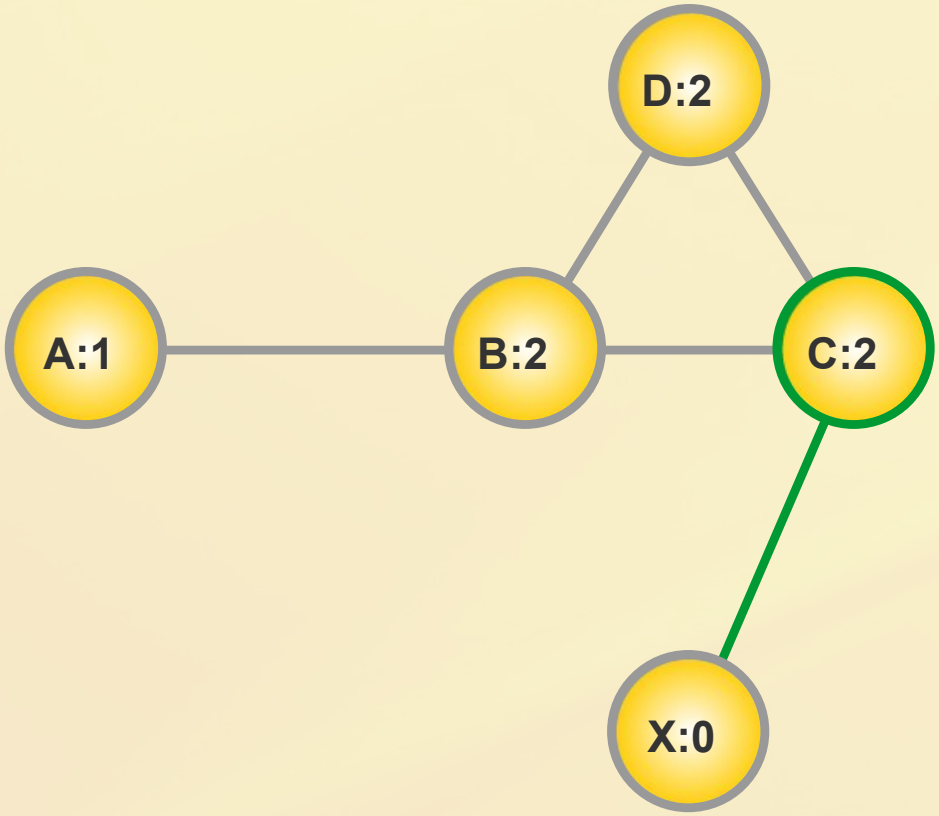


- Hinzufügen einer Kante
- **X** strebt 1er Party an
- **X** in Einladungsliste
- **X** schickt Anfrage an **C**

Einladungsliste:

[**X**]

PartyPeople - Beispiel

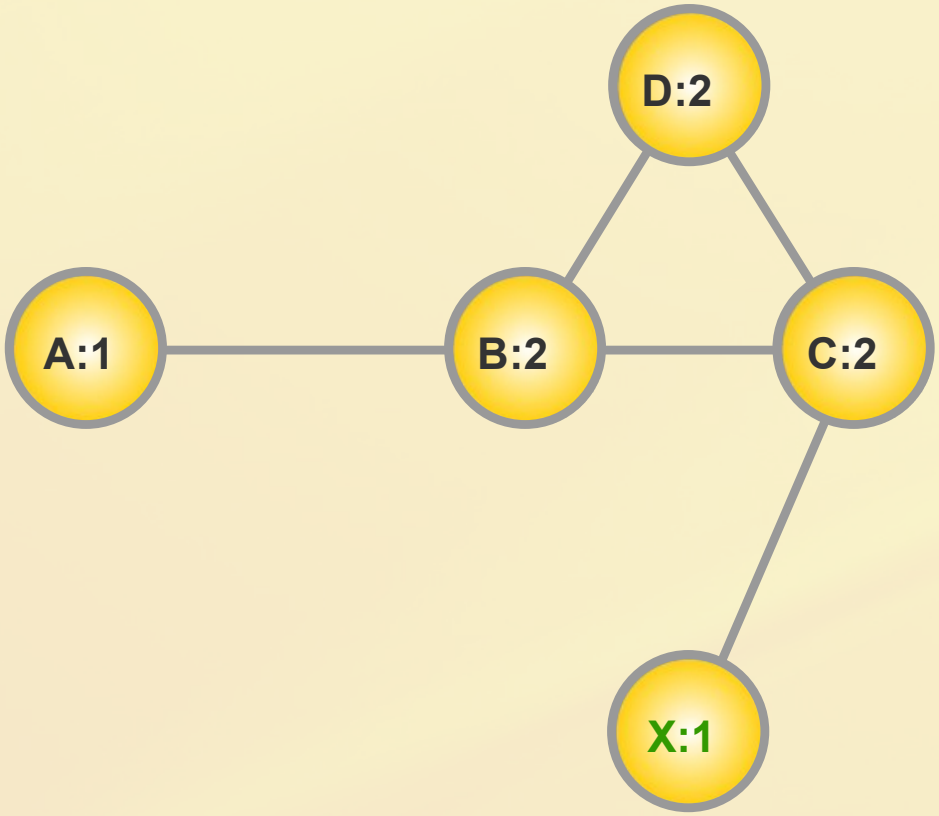


- **C** sagt zu, da Core-Nummer = 2 ≥ 1

Einladungsliste:

[**X**]

PartyPeople - Beispiel

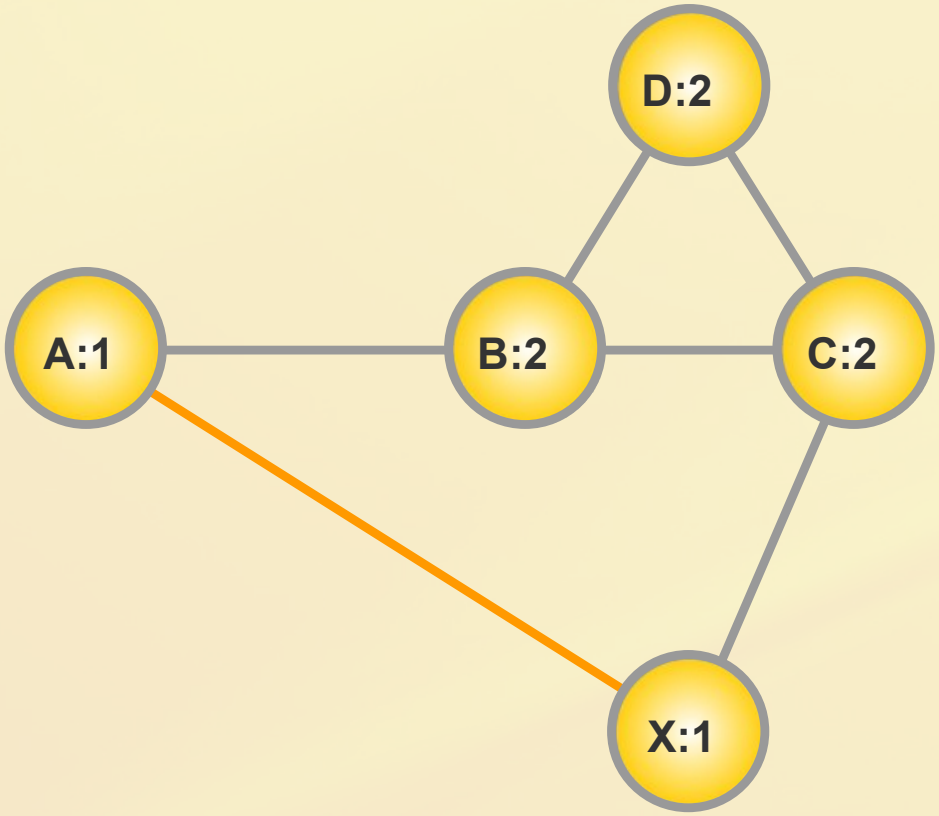


- **X** hat genügend Gäste
- Party findet statt
- **X** steigt in 1-Core auf

Einladungsliste:

[]

PartyPeople - Beispiel

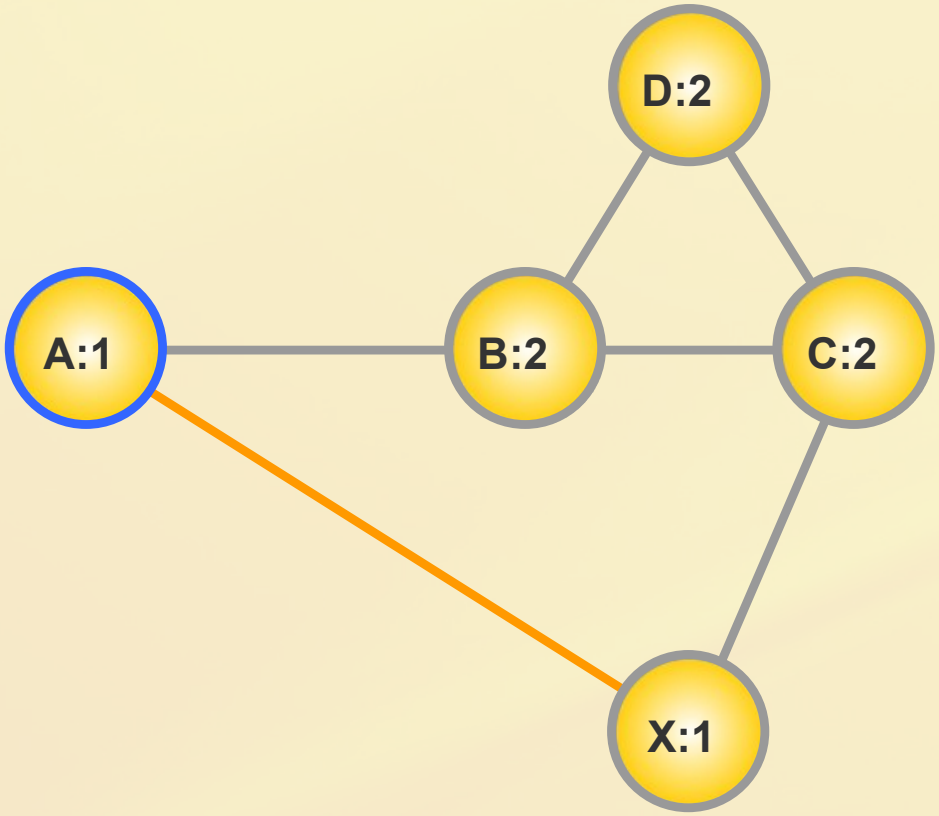


- Neue Kante zu **A**
- **X** strebt 2er Party an
- **X** auf Einladungsliste

Einladungsliste:

[**X**]

PartyPeople - Beispiel

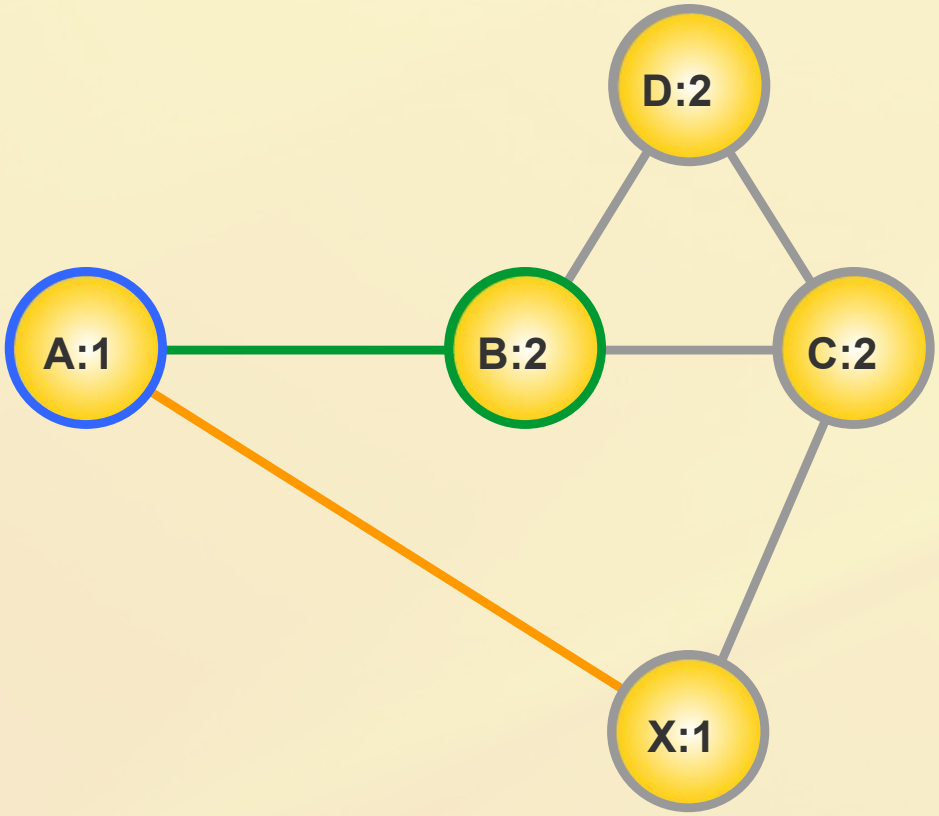


- **X** schickt Einladung an **A**
- **A** darf nur mit mind. zwei Zusagen teilnehmen
- **A** auf Einladungsliste

Einladungsliste:

[**X**, **A**]

PartyPeople - Beispiel

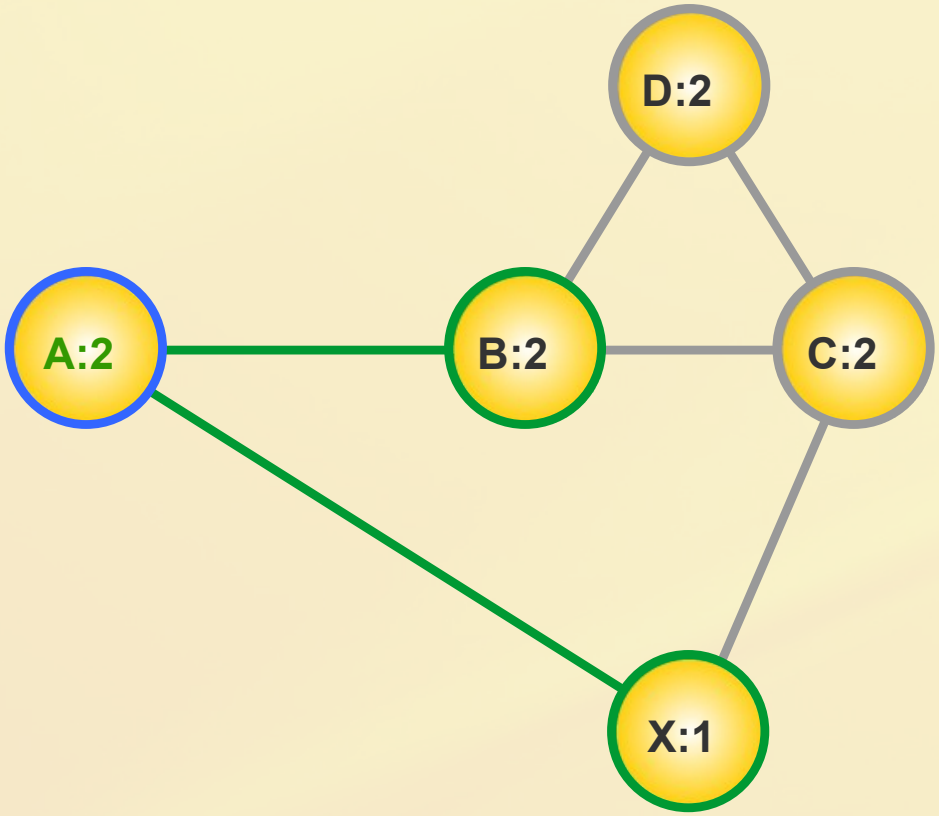


- **A** schickt Einladung an **B** (2er-Party)
- **B** sagt zu, da Core-Nummer = 2 ≥ 2

Einladungsliste:

[**X**, **A**]

PartyPeople - Beispiel

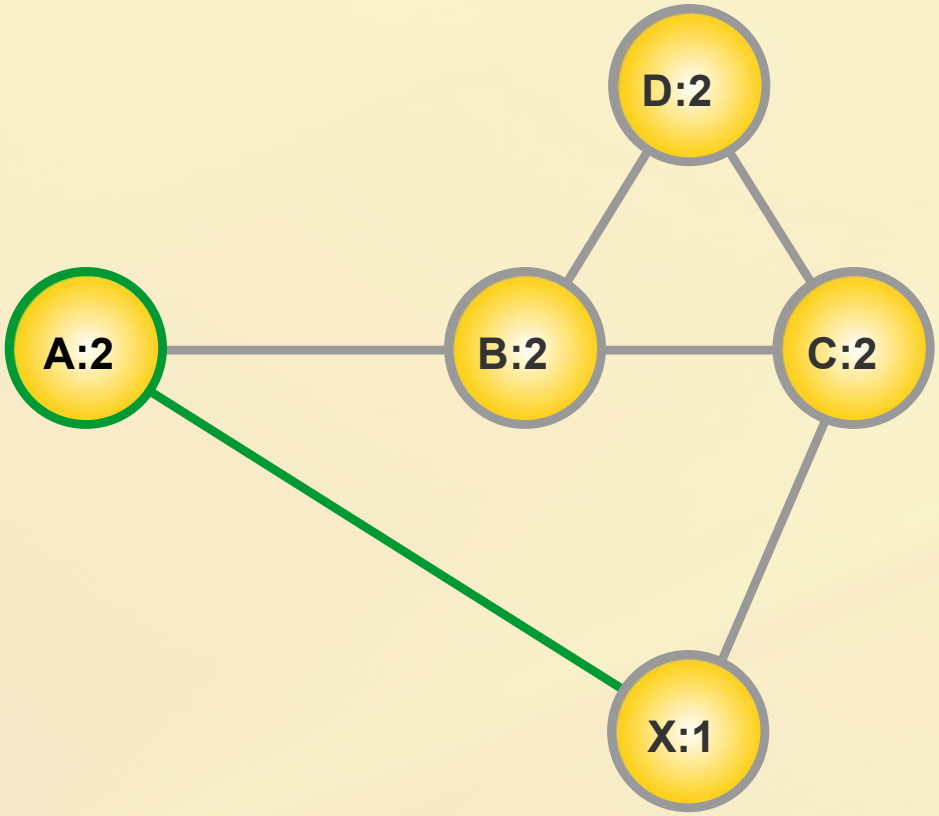


- **A** schickt Einladung an **X** (2er-Party)
- **X** sagt zu, da auf Einladungsliste
- **A** hat genügend Zusagen
- **A** steigt in 2-Core auf

Einladungsliste:

[**X**, **A**]

PartyPeople - Beispiel

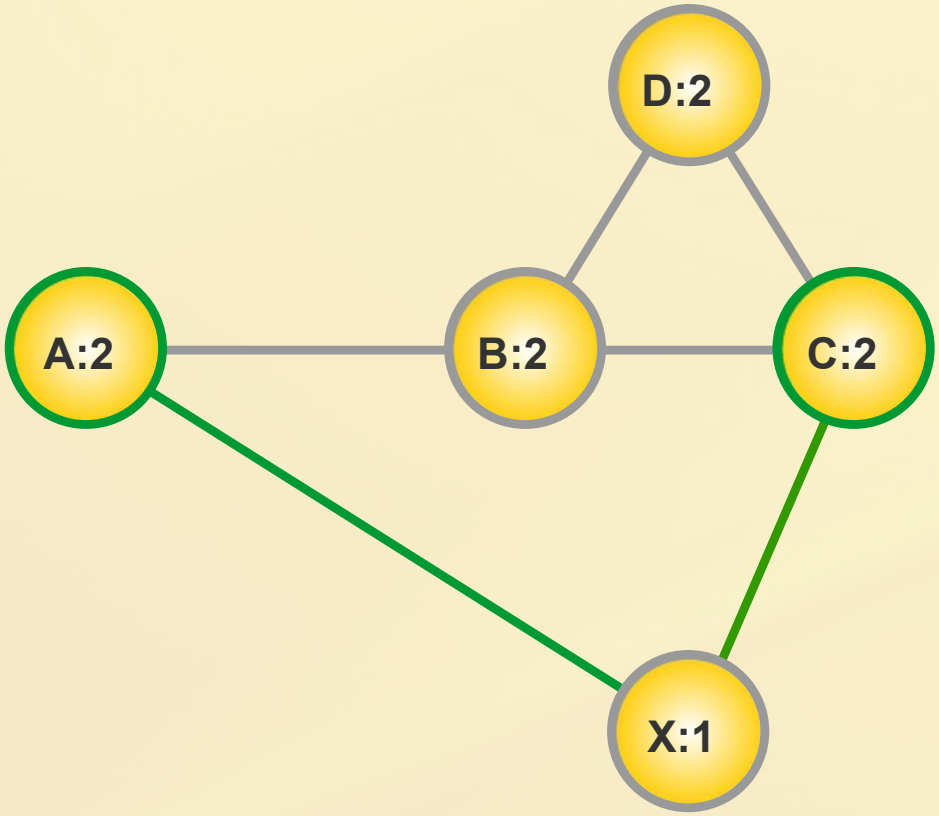


- **X** bekommt Zusage von **A**
- **X** benötigt jedoch zwei Zusagen für Aufstieg

Einladungsliste:

[**X**, **A**]

PartyPeople - Beispiel

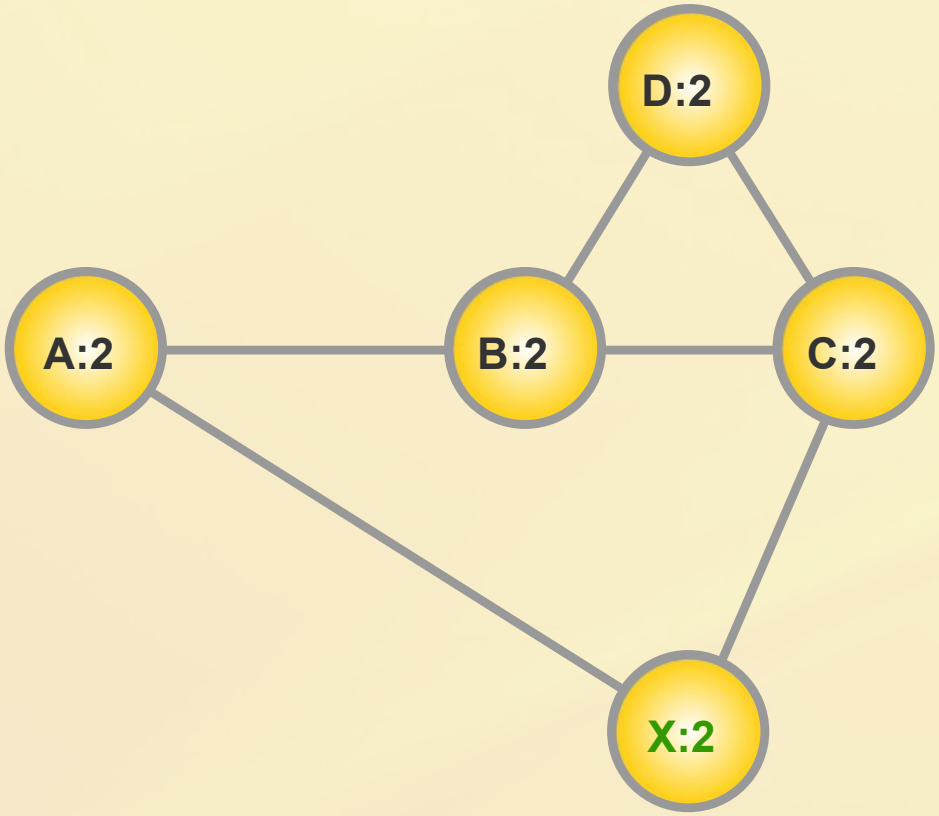


- **X** schickt Einladung an **C**
- **C** sagt zu, da Core-Nummer = 2 ≥ 2

Einladungsliste:

[**X**, **A**]

PartyPeople - Beispiel



- **X** hat genügend Zusagen
- Party findet statt
- **X** steigt in 2-Core auf

Einladungsliste:

[]

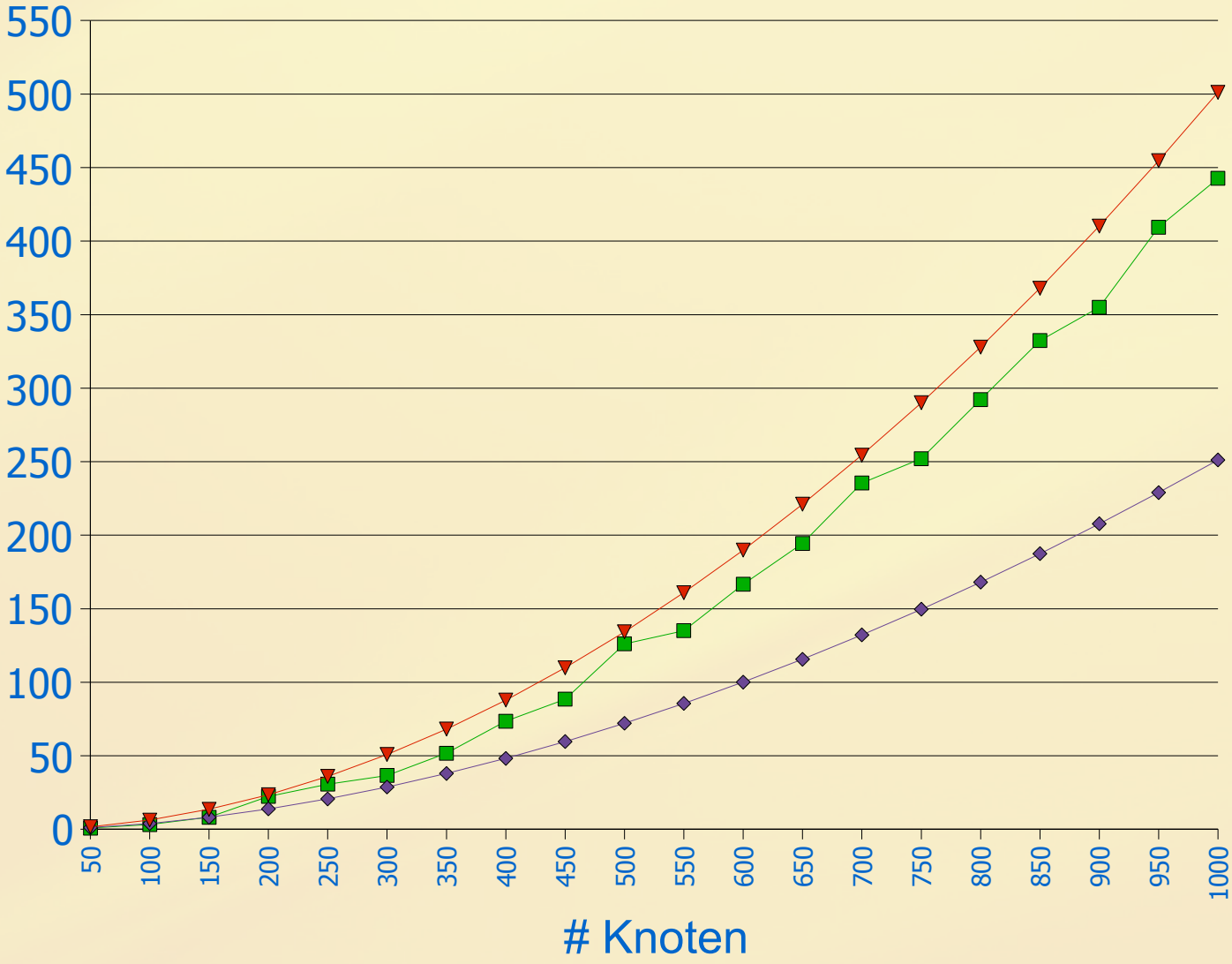
Pseudocode

```
ladeEin(Knoten, angestrebteCoreNr)
    if ((CoreNr(Knoten) >= angestrebteCoreNr)
        or (schonEingeladen(Knoten)))
        return angestrebteCoreNr;

    MarkiereAlsEingeladen(Knoten);

    forall (Nachbarn as Nachbar)
        möglicheCoreNr = ladeEin(Nachbar, angestrebteCoreNr);
        if (möglicheCoreNr == angestrebteCoreNr) Zusagen++;

    if (Zusagen >= angestrebteCoreNr)
        setzeCoreNr(Kntoen, angestrebteCoreNr);
        return angestrebteCoreNr;
    else return CoreNr(Knoten);
```

Aufwand

- Worst-case Abschätzung nicht aussagekräftig
- Armortisierte Betrachtung angebracht
- Idee: Messung
  - Zählen der Rekursionsdurchläufe
  - Durchschnitt über der Anzahl der Knoten auftragen

Messung



$n^{1,9}$

Rekursionen in 1000 bei

Max. Core-Numer: 10

Strict Core-Größe: 10%

$n^{1,8}$

Verbesserungsmöglichkeiten

- Beschränkung der Rekursionsschritte durch
  - Markierung unmöglicher Kandidaten
  - Einschränkung durch mehr Parameter
- Bessere Datenstrukturen
  - Haltung der Knoten in einem Heap

Gewonnene Erkenntnisse

- Selbstständiges Arbeiten
- Entwurf von PartyPeople wesentlich zeitintensiver als gedacht
- Erst denken, dann schreiben
- Umgang mit Eclipse und CVS
- Einarbeitung in yFiles Bibliothek
- Erste Schritte in LaTeX
- Nette Leute kennengelernt :)

Demo

Referenzen

- [1] Seidman S. B. (1983) Network structure and minimum degree, Social Networks, 5, 269-287.
- [2] Batagelj, V. and Zaversnik, M. (2001) Generalized cores
- [3] T. Luczak (1991) Size and connectivity of the k-core of a random graph
- [4] B. Pittel, J. Spencer, N. Wormald (1996), Sudden emergence of a giant k-core in a random graph