Vorlesung
P2P Netzwerke

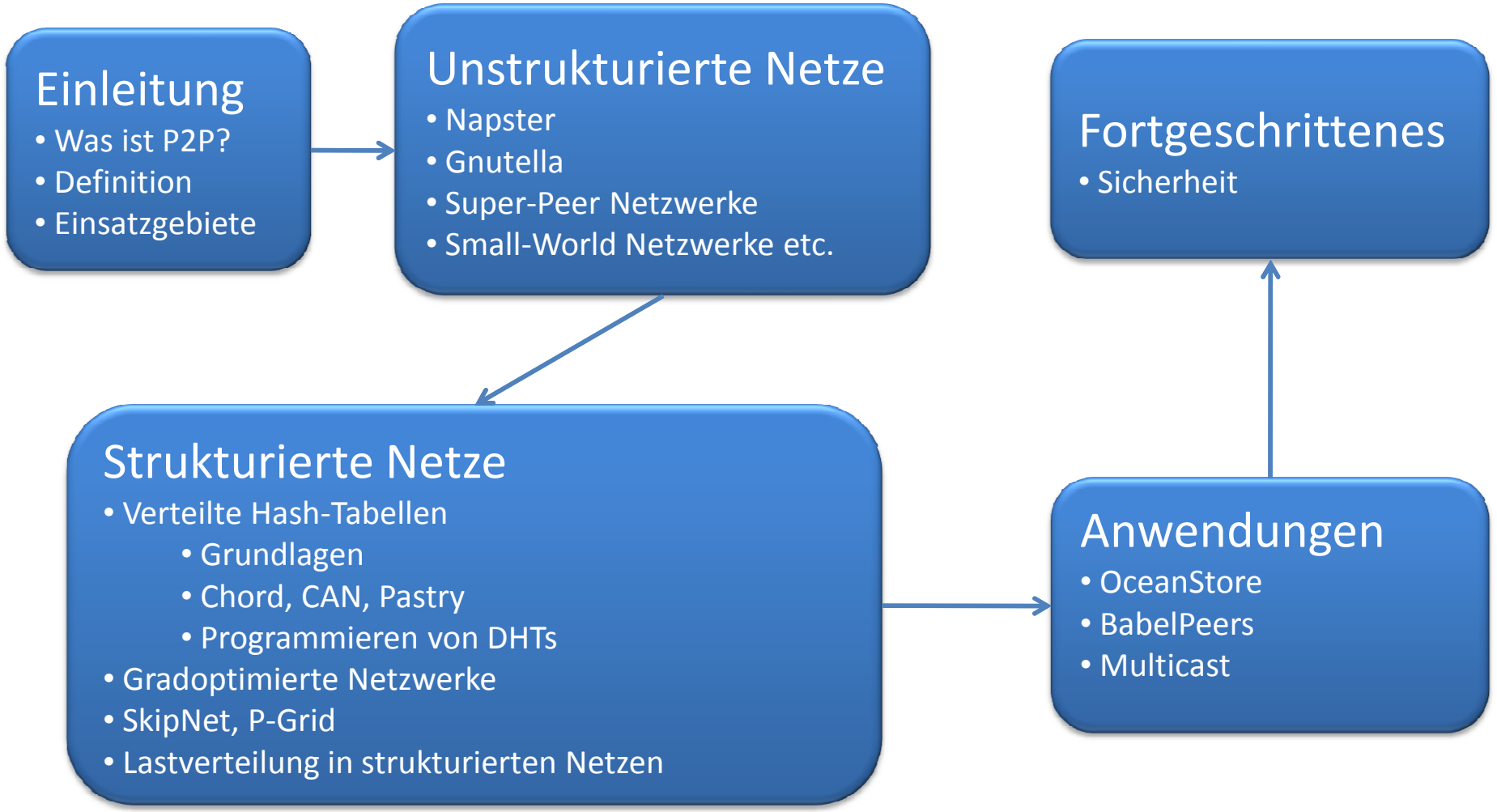
1: Einführung

Dr. Felix Heine

Complex and Distributed IT-Systems

felix.heine@tu-berlin.de

Inhalte der Vorlesung



Ziele der Vorlesung

- Verstehen,
  - … was ein P2P Netzwerk ist
  - … wie es sich von anderen Netzen unterscheidet
  - … wann P2P Netzwerke sinnvoll sind
  - … wie sie funktionieren
  - … welche Topologien es gibt
  - … welche Routing-Algorithmen es gibt
  - … wie Selbstorganisation funktioniert
  - … wie Daten gespeichert werden
  - … und wie diese Daten gefunden werden
  - … welche Anwendungen für P2P Netze es gibt
  - … wie P2P Netzwerke programmiert werden
  - … wie P2P Netzwerke abgesichert werden

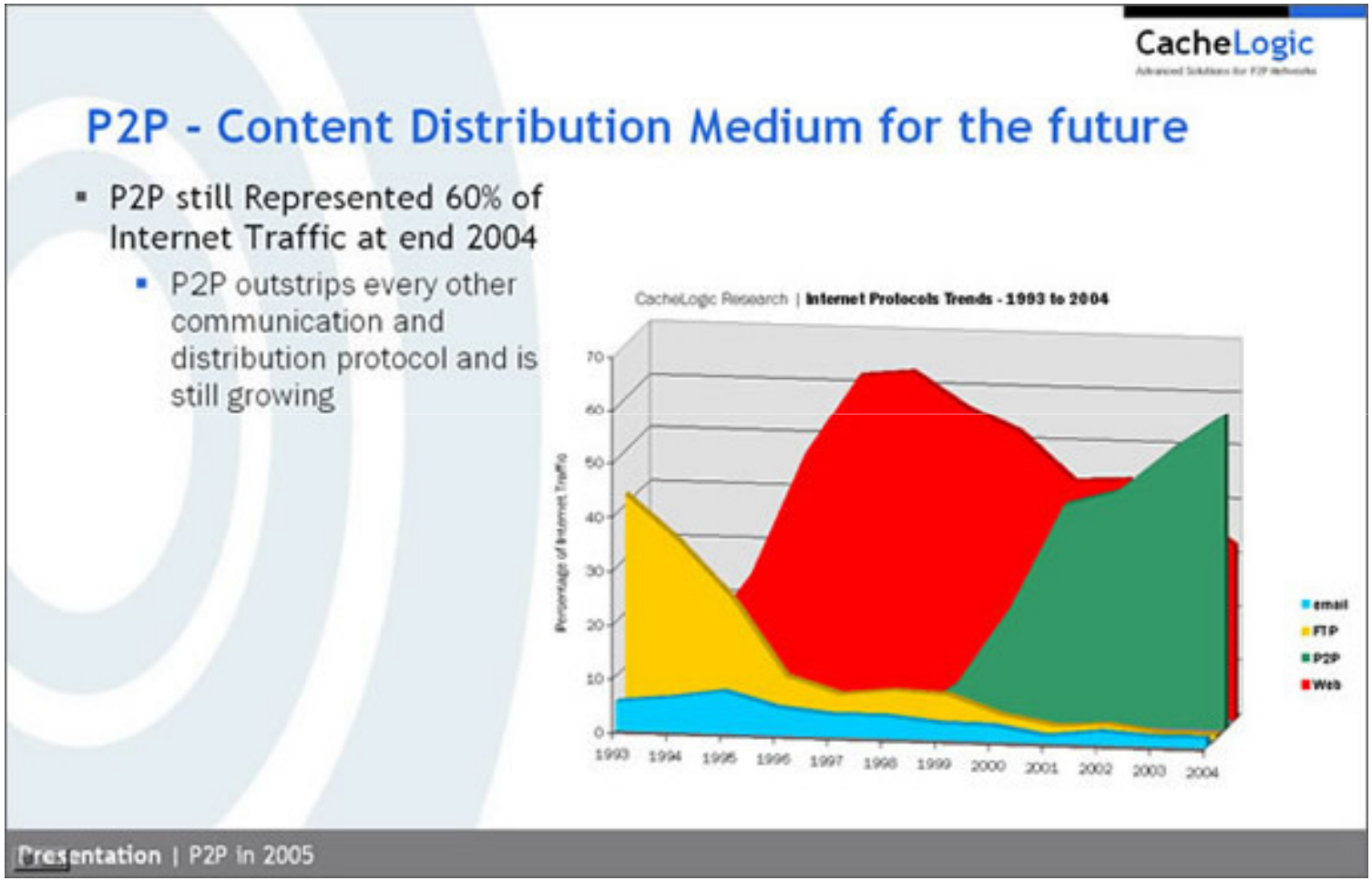
Literatur

- Bücher zu P2P:
  - Ralf Steinmetz, Klaus Wehrle: *Peer-to-Peer Systems and Applications,* Springer, 2005
  - Peter Mahlmann, Christian Schindelhauer: *P2P Netzwerke,* Springer, 2007
  - Andy Oram: Peer-to-Peer: *Harnessing the Power of Disruptive Technologies*, O'Reilly, 2001
- Paper zu speziellen Themen
  - wird jeweils beim Thema bekanntgegeben

P2P in den Medien

- Immer wieder Thema in den Medien
- Spiegel Online
  - **Musikindustrie gegen P2P-Nutzer: Jeden Monat 1000 Anzeigen** (27.12.2006)
    http://www.spiegel.de/netzwelt/web/0,1518,456662,00.html
- Technology Review:
  - **P2P als großer Retter** (9.1.2007)
    http://www.heise.de/tr/result.xhtml?url=/tr/artikel/83216
  - **P2P Netzwerke werden Salonfähig** (20.7.2006)
    http://www.heise.de/tr/result.xhtml?url=/tr/artikel/75613

P2P Traffic im Internet



Quelle: http://www.cachelogic.com/home/pages/studies/2005_07.php

Peer-2-Peer: Was ist das?

- Peer-2-Peer (P2P) ist durch Filesharing bekannt geworden
- Das Konzept ist schon viel älter und hat viel mehr Anwendungen:
  - Arpanet
  - Skype
  - Groove
  - Datenmangement
- Was ist ein Peer? (Merriam-Websters)

> **1 :** one that is of equal standing with another **:** **EQUAL**; *especially* **:** one belonging to the same societal group especially based on age, grade, or status
> **2** *archaic* **:** **COMPANION**
> **3 a :** a member of one of the five ranks (as duke, marquess, earl, viscount, or baron) of the British peerage **b :** **NOBLE** 1
> - **peer** *adjective*

Definitionen

- Wikipedia:
  - *"Kommunikation unter Gleichen"*
- Gribble (2001):
  - *A distributed system in which participants rely on one another for services [...] Peers in the system can elect to provide services as well as consume them.*
- Yang/Garcia-Molina (2002):
  - *Peer-to-peer (P2P) systems are distributed systems in which nodes of equal roles and capabilities exchange information and services directly with each other*

Das P2P Prinzip

- Keine 100% scharfe Definition!
- Ein „Organisationsprinzip“:
  - Jeder Rechner im Netzwerk erfüllt die gleichen Aufgaben
  - Es gibt keine Unterscheidung zwischen Client und Server
- Kernkonzepte (nicht immer alle Erfüllt!)
  - Selbstorganisation, kein zentrales Management
  - Teilen von Ressourcen
    - Nutzung von Ressourcen am "Rand" des Netzwerkes
  - Die Peers sind alle gleich
  - Typischerweise sehr viele Peers
  - Knoten kommen und gehen jederzeit
  - Peers sind autonom (eigene Entscheidung über Angebote)
  - Adressierung auf Anwendungsebene

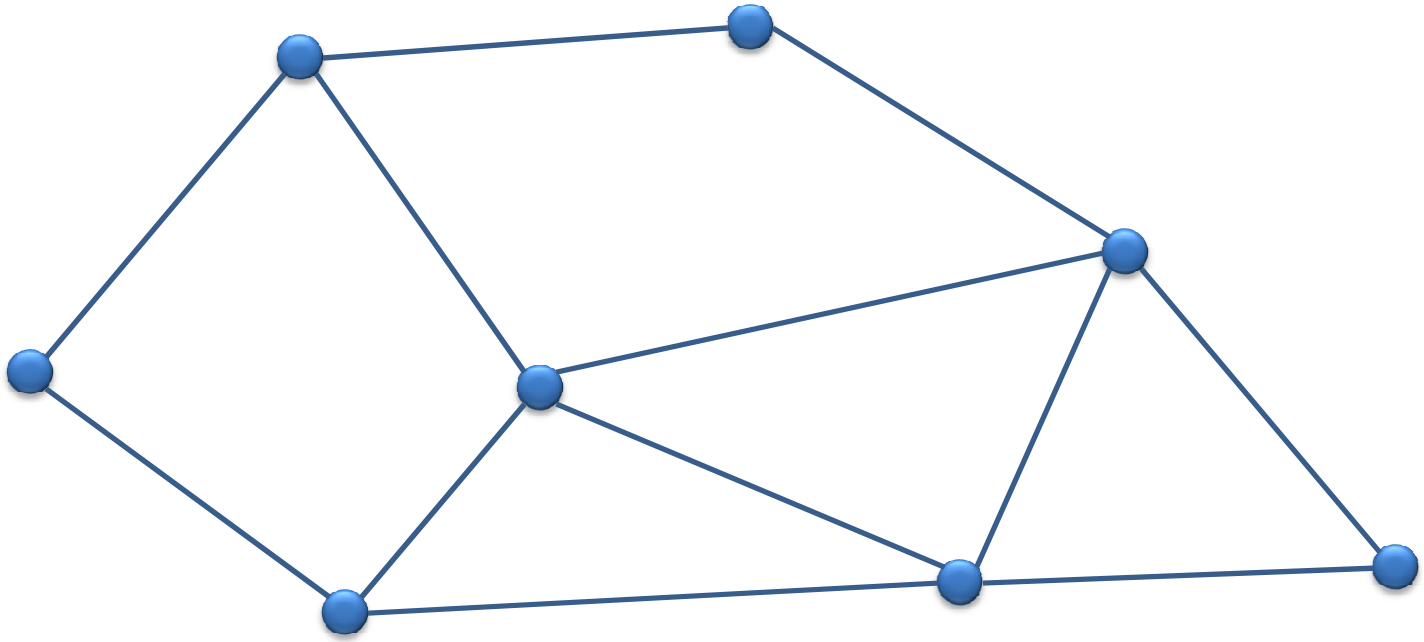
Vorteile von P2P

- Höhere Verfügbarkeit/Ausfallsicherheit: kein „single point of failure“
- Aktualität/Einfachheit des „Publizierens“:
  - Zugriff auf unveröffentlichte Daten
  - trotzdem Anonymität
- Skalierbarkeit
- Geteilte Ressourcen (Rechenleistung, Speicherplatz, Bandbreite)
- Keine Zensur/Manipulation durch zentrale Instanz
- **Nachteile von zentralisierten Systemen (z.B. Google):**
  - schlechte Abdeckung
  - lange Updatezeiten
  - Kosten für Betrieb des Systems
  - Zentrale Instanz steuert (und manipuliert) die Inhalte

Nachteile von P2P

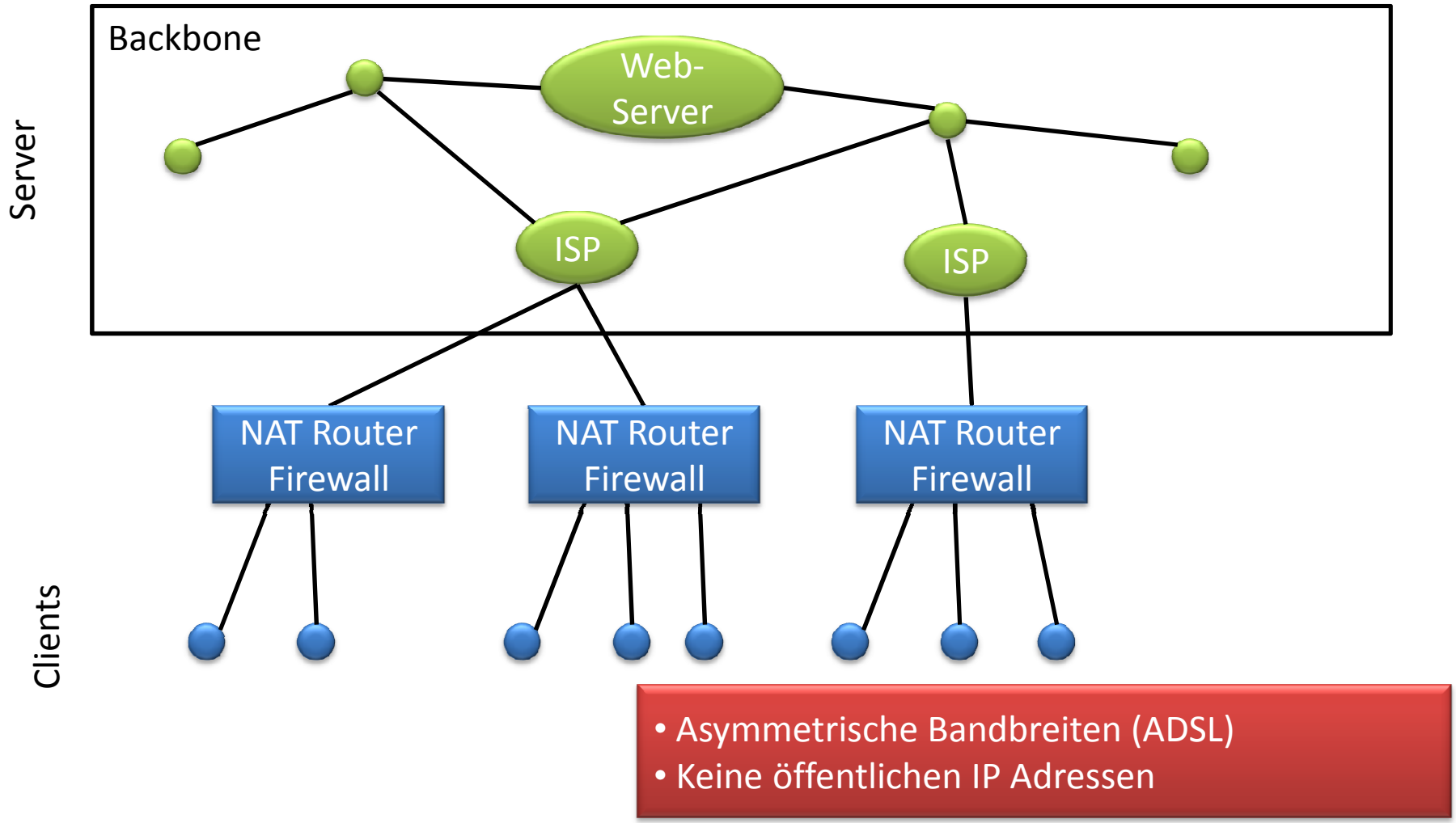
- Erhöhter Aufwand:
  - Programmierung
  - Fehlersuche
  - Sicherheit
  - Routing / Suche
  - Replikation
- Keine Verläßlichkeit ("nur" probabilistische Garantien)
- Vertrauenswürdigkeit der Teilnehmer?

Das Internet



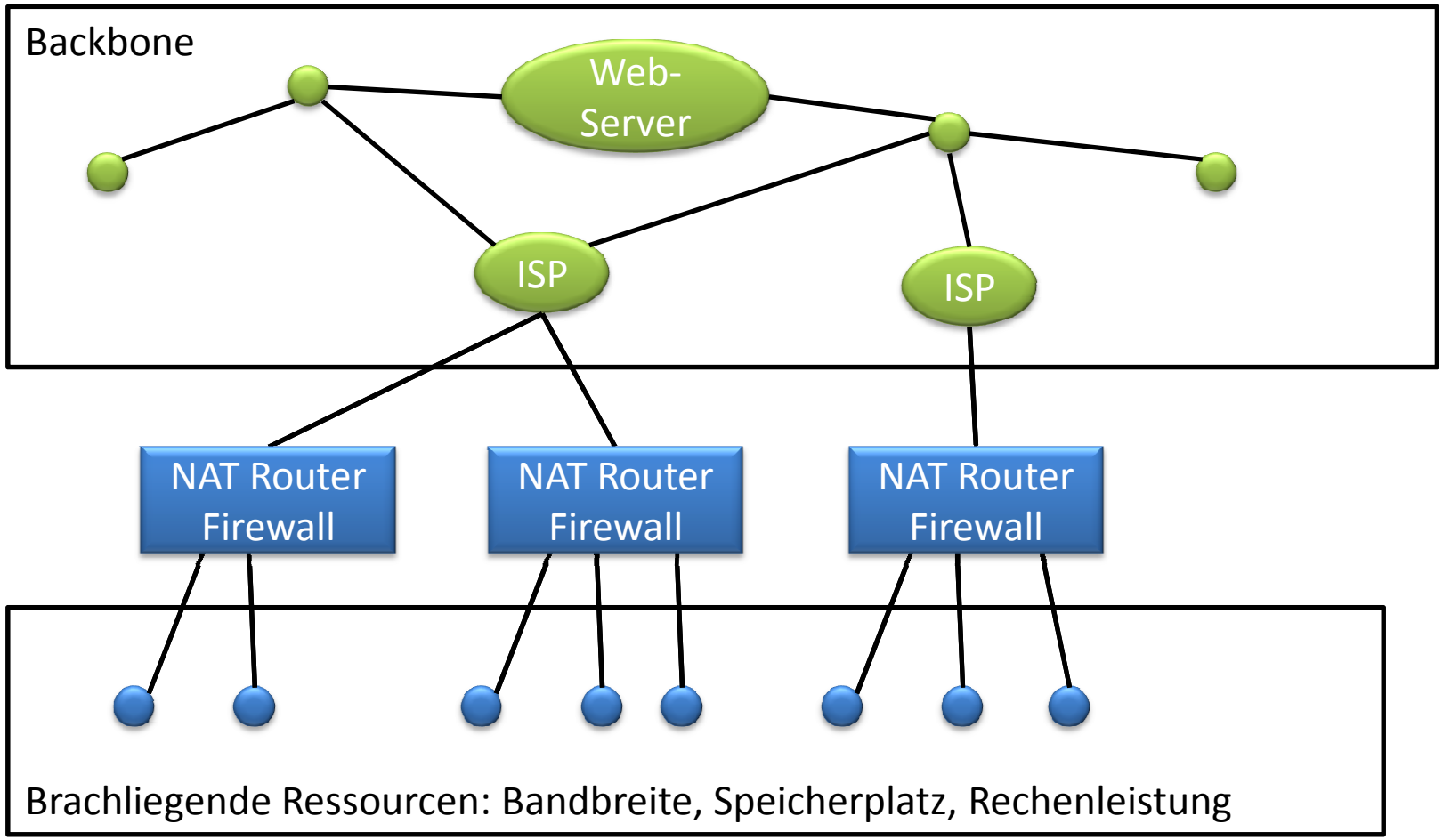
Sieht es so wirklich aus?

Tatsächlich

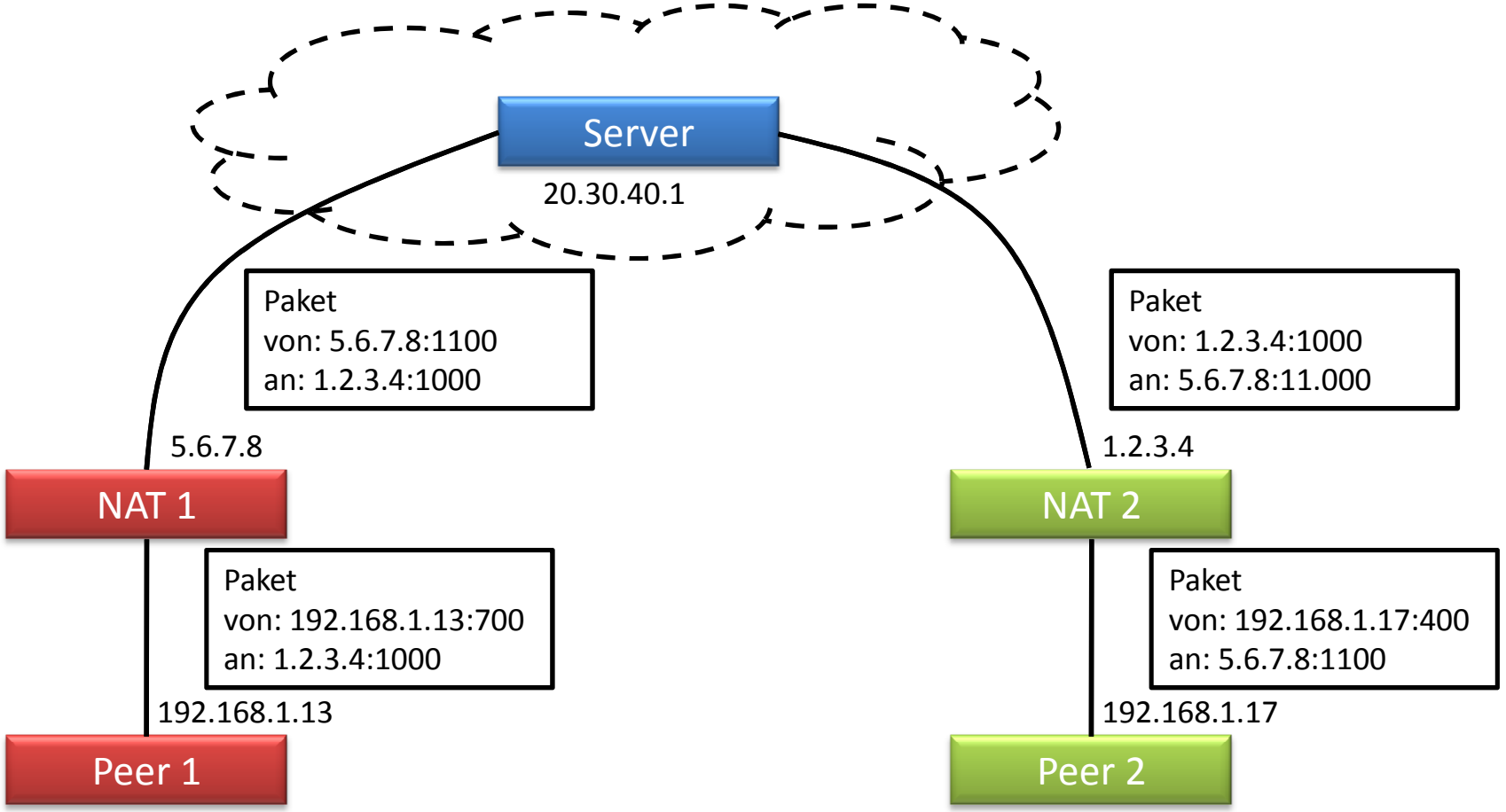


- Asymmetrische Bandbreiten (ADSL)
- Keine öffentlichen IP Adressen

Tatsächlich



Exkurs: UDP Hole Punching



Vorlesung
P2P Netzwerke
2: Unstrukturierte Netze

Dr. Felix Heine

Complex and Distributed IT-Systems

felix.heine@tu-berlin.de

Inhalt

- Napster
  - Erstes "P2P" Netzwerk
  - Kein wirkliches P2P
  - Enormes Medienecho → Popularität für P2P
- Gnutella
  - Erstes "echtes" P2P Netzwerk
  - Interessante Netzwerkeigenschaften
- Eigenschaften von Gnutella
  - Pareto-Netzwerke
  - Small-World Netzwerke
- Super-Peer Netzwerke

Literatur

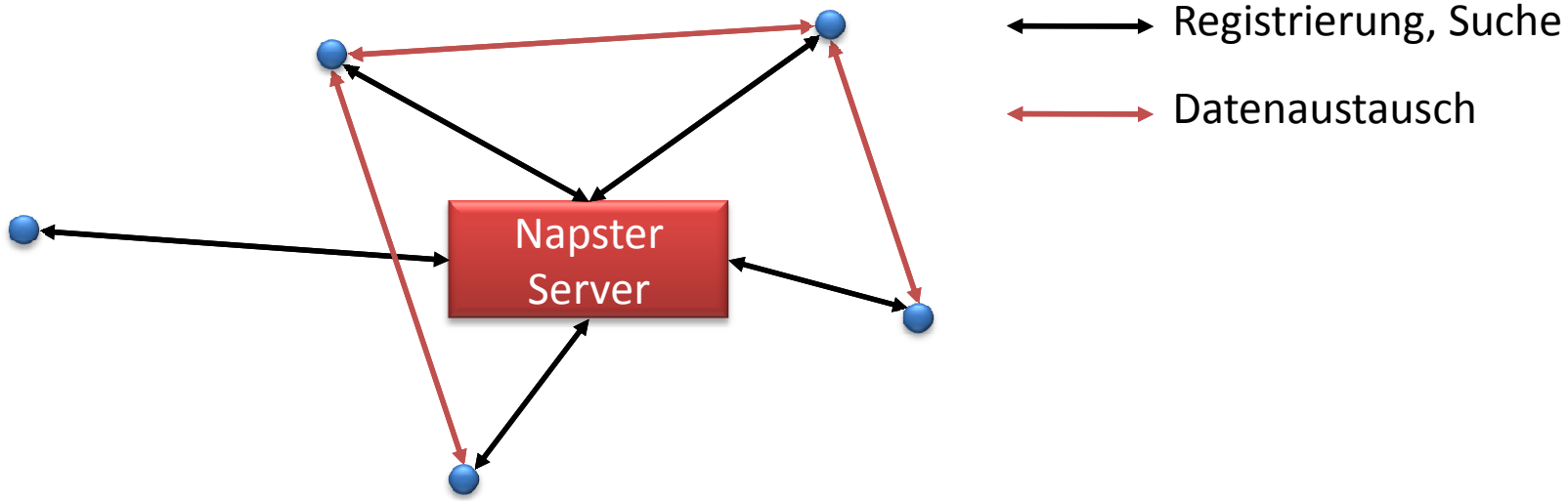
- The Gnutella Protocol Specification v0.4
- Mihajlo A. Jovanovic, Fred S. Annexstein, Kenneth A. Berman: "Scalability Issues in Large Peer-to-Peer Networks A Case Study of Gnutella". Technical report, University of Cincinnati, January 2001.
- Albert-Laszlo Barabasi, Reka Albert: "Emergence of Scaling in Random Networks", Science, Vol 286, 1999.
- Duncan J. Watts, Steven H. Strogatz: "Collective dynamics of 'small-world' networks", Nature, Vol 393, 1998.

Napster

- Netzwerk zum Tauschen von MP3-Dateien
- Geschichte:
  - Mai 1999: Shawn Fanning (Northeastern University) gründet "Napster Online music service"
  - Dezember 1999: Erster Rechtsstreit
  - März 2000: University of Wisconsin: 25% des IP Datenaufkommens sind Napster Pakete
  - Dezember 2000: geschätzte 60 Millionen Anwender
  - Februar 2001: US Circuit Court of Appeals: Napster weiss dass die Anwender Copyright-Rechte verletzen: Napster wird geschlossen

Napster

- Fokussiert: Einfache Anwendung (suche nach Musikdateien)
- Suche über zentralen Server
- Datenaustausch dezentral

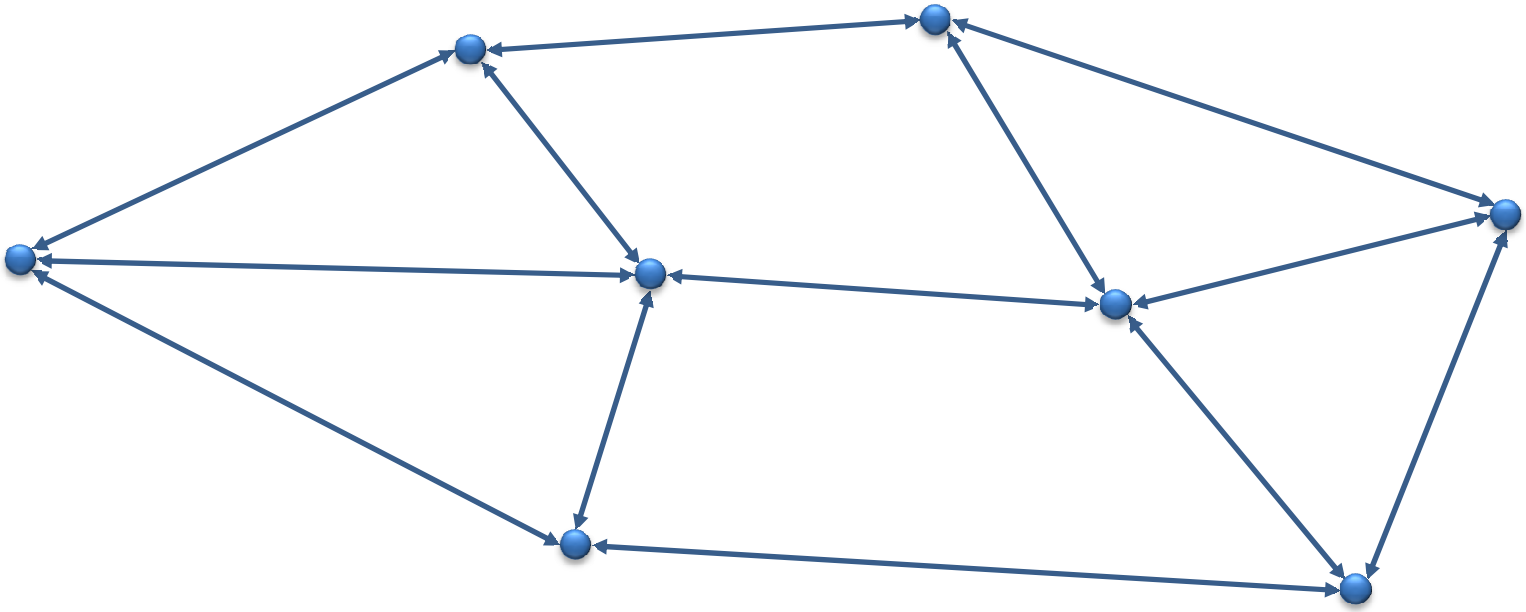


Napster

- Funktionsweise
  - Verbinde mit Napster Server
  - Hochladen der eigenen Dateiliste auf den Server
  - Suche per Schlüsselwort-Liste über den Server
  - Wähle beste Antwort aus
  - Direktverbindung mit Ziel-Peer zum Download
- Nachteile
  - Zentraler Server
    - Single Point of Failure
    - Skalierbarkeit?
  - Rechtliche Probleme

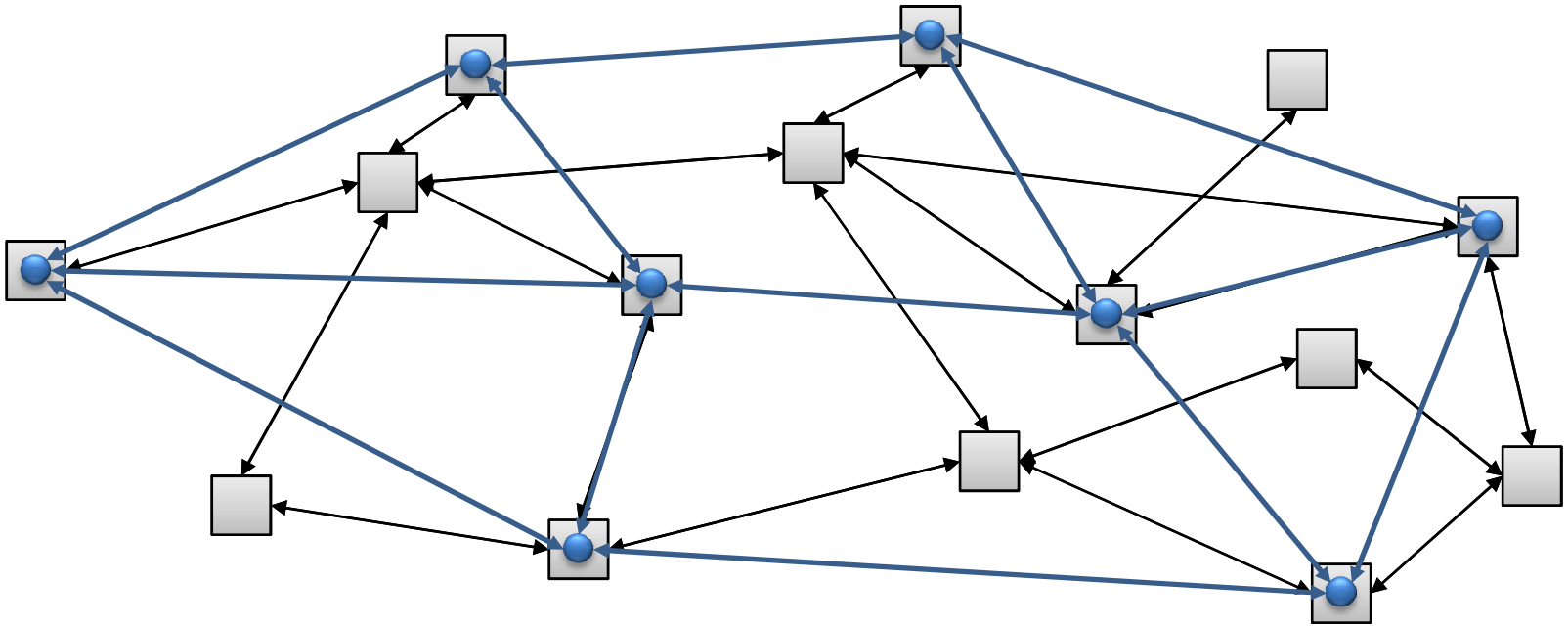
Gnutella

- Kein zentraler Server mehr
- Peers werden **Servents** genannt



Overlay-Netzwerk

- Gnutella ist sog. **Overlay** über dem Internet



Problemstellungen

- Bootstrapping
  - Woher kenne ich Knoten im Netzwerk?
  - Mit welchen Knoten verbinde ich mich?
- Suche
  - Wie finde ich Knoten, die Datei XYZ.MP3 anbieten?
  - Wie finde ich den besten Knoten, der Datei XYZ.MP3 anbietet?
  - Der *beste* Knoten?
    - Verbindung?
    - Auslastung?
    - Qualität der Datei?

Lösungen

- Bootstrapping
  - Allgemein bekannte Knoten (über Webserver)
  - Liste mit bekannten Knoten von vorheriger Session
  - Weitere Knoten über die Startknoten kennenlernen
- Suche
  - Flooding: Suche an alle Knoten weiterschicken
  - Knoten merken sich IDs von gesehen Nachrichten, dies verhindert Schleifen
  - TTL (Time to Live) begrenzt die Laufzeit einer Nachricht

Gnutella Nachrichten

Header für alle Nachrichten	
Descriptor ID	16 Bytes
Payload Descr.	1 Byte
TTL	1 Byte
Hops	1 Byte
Payload Length	4 Bytes

PING (0x00)
keine weiteren Felder

PONG (0x01)	
Port	2 Bytes
IP Address	4 Bytes
Nb. of shared Files	4 Bytes
Nb. of shared KBytes	4 Bytes

QUERY (0x80)	
Min Speed	2 Bytes
Search Criteria	n Bytes

QUERY HIT (0x81)	
Nb. of Hits	1 Byte
Port	2 Bytes
IP Address	4 Bytes
Speed	4 Bytes
Result Set:	
- File Index	4 Bytes
- File Size	4 Bytes
- File Name (+\0\0)	n Bytes
- … (weitere Result.)	
Servent Identifier	16 Bytes

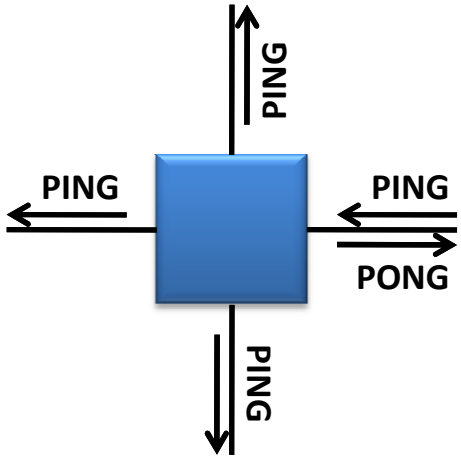
Abholen der Dateien

- Direkte Verbindung zum Servent
- Transfer durch HTTP
- Wenn Servent hinter Firewall liegt:
  - PUSH Nachricht über Gnutella routen

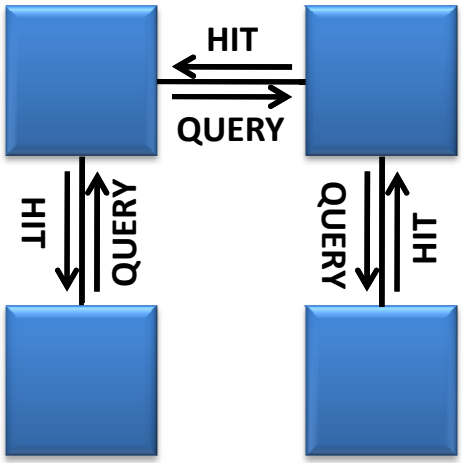
PUSH (0x40)	
Servent Identifier	16 Bytes
File Index	4 Bytes
IP Address	4 Bytes
Port	2 Bytes

Routing in Gnutella

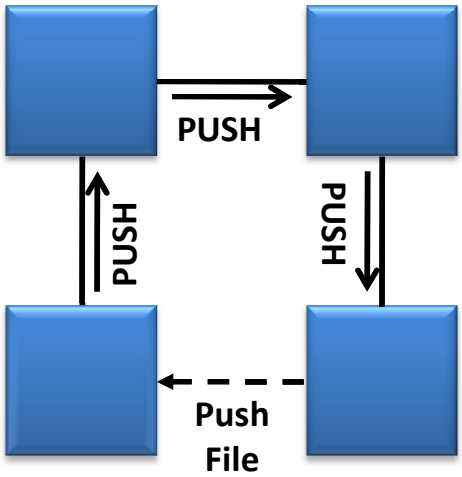
Ping / Pong



Query / Query Hit



Push



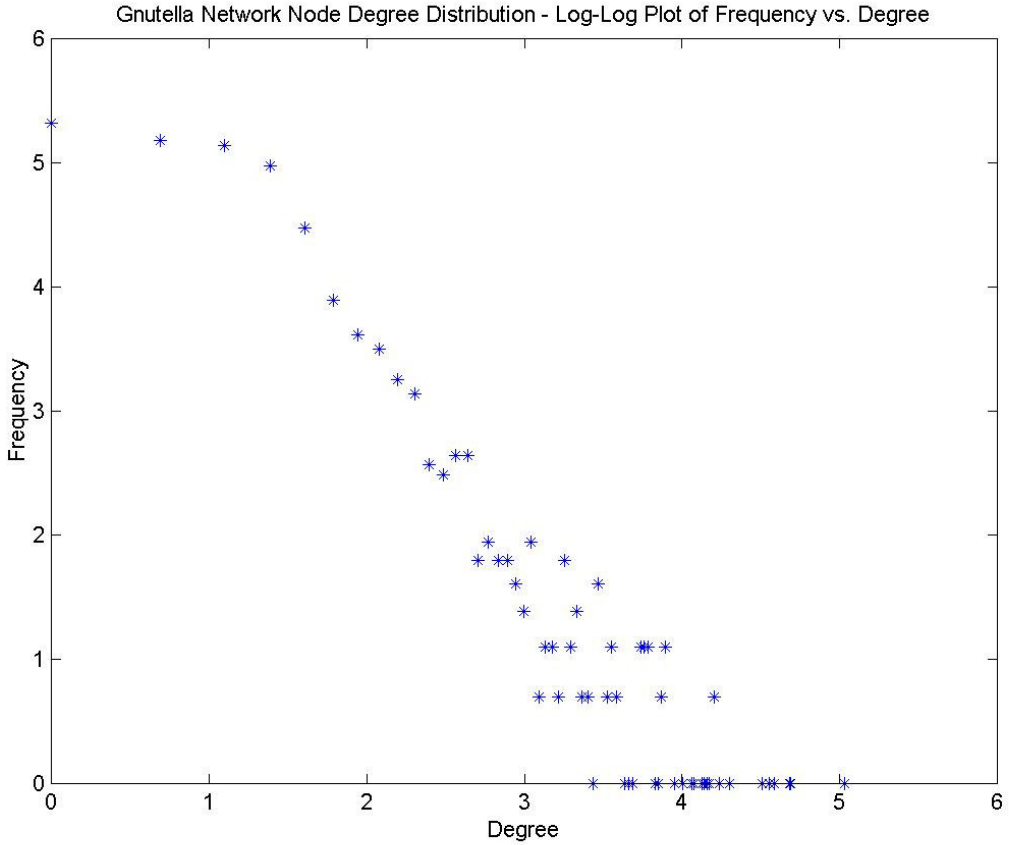
Probleme von Gnutella

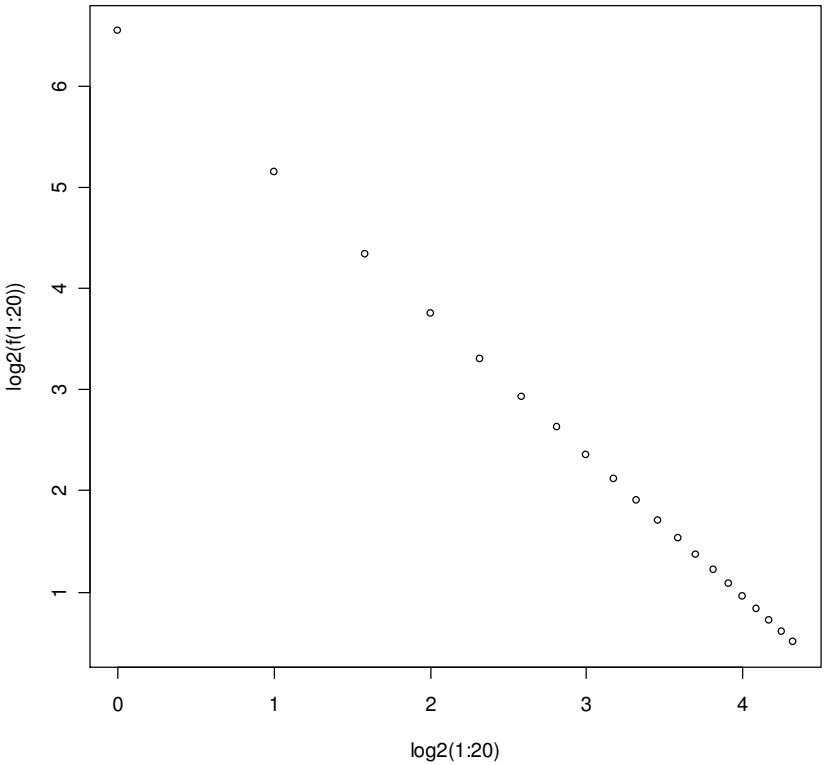
- Hohe Netzwerklast
- Flooding ist äußerst unstrukturierte Vorgehensweise:
  - Ich frage fast jeden, ob er eine Datei liefern kann
  - Begrenzung nur durch TTL
- Durch TTL werden evtl. Dateien nicht gefunden
- Gnutella-Topologie hat nichts mit realer Topologie zu tun
  - Zig-Zack Routen
  - Daraus folgen hohe Latenzzeiten

Eigenschaften von Gnutella

- Was für ein Netzwerk wird aufgebaut?
- Ist das Netzwerk rein zufällig?
- Oder hat es bestimmte Eigenschaften?
- Woher kommen diese Eigenschaften?
- Gibt es sog. "Emergenz" bzw. "Selbstorganisation"?

Gradverteilung





$$f(x) = 94x^{-1.4}$$

Quelle: Jovanovic, Annexstein, Berman:
Scalability Issues in Large Peer-to-Peer Networks – A Case Study of Gnutella

Gradverteilung

- Allgemein: Power-Law (Pareto-Verteilung)

$$\Pr(X = x) = \frac{C}{x^k}$$

- Bei Gnutella: C=94, k=1,4
- Was hat man davon?
  - Power-Law Netzwerke sind robust
  - Es gibt viele Knoten mit kleinem Grad, deren Ausfall nicht so schlimm ist
- Wie kommt das zustande?
  - "Preferential Attachement":
  - Die Reichen werden reicher

Robustheit

- Nimm ein Netzwerk mit 10000 Knoten und entferne zufällig ausgewählte Knoten
- Zufallsnetzwerk:
  - Entferne 5% der Knoten: Größte Komponente hat 9000 Knoten
  - Entferne 18% der Knoten: Alle Komponenten zwischen 1 und 100 Knoten
  - Entferne 45% der Knoten: Nur Gruppen von 1 oder 2 Knoten überleben
- Power-law Netzwerk:
  - Entferne 5% der Knoten: Nur isolierte Knoten brechen weg
  - Entferne 18% der Knoten: Größte Komponente hat 8000 Knoten
  - Entferne 45% der Knoten: Immernoch große Komponenten
- Achtung: gilt nur für **zufällige** Ausfälle!

Modell für Preferential Attachement

- Barabási-Albert-Modell:
  - Starte mit kleinem Netzwerk mit ein paar Kanten und Knoten
  - Pro Schritte füge einen Knoten hinzu
  - Füge Kanten zu den bestehenden Knoten mit einer Wahrscheinlichkeit hinzu, die Proportional zum Grad des Knoten ist
- Effekt: Knoten verbinden sich mit bereits gut verbundenen Knoten mit höherer Wahrscheinlichkeit

Milgram's Experiment

- Wie sehen "reale" Netzwerke aus?
- Soziales Netzwerk von Menschen: Wer kennt wen?
- Milgram's Experiment (1967):
  - Briefe an einen Freund von Milgram wurden an **zufällige** Personen in Nebraska verschickt
  - "Routing-Informationen":
    - Name der Zielperson
    - Ort (Boston)
    - Beruf (Börsenmakler)
  - Anweisung: nur an Du-Freunde weitergeben
- Ergebnis: durchschnittlich 6 Schritte bis zum Ziel!

Randnotiz: Erdös-Nummer

Paul Erdös (1913 – 1996)
Berühmter Mathematiker

Frage: Über wieviele Schritte ist jemand als Co-Autor mit Paul Erdös verbunden?

Beispiel: Odej Kao hat Erdös-Nummer 4:

Odej Kao	ist Co-Autor von	Klaus H. Ecker
Klaus H. Ecker	ist Co-Autor von	Helmut Ratschek
Helmut Ratschek	ist Co-Autor von	Egbert Harzheim
Egbert Harzheim	ist Co-Autor von	Paul Erdös

Milgram's Experiment

- Wie ist das möglich?
- These: Soziale Netzwerke haben folgende Eigenschaften:
  - Großer Clustering-Koeffizient
  - Kleiner Durchmesser bzw. charakteristische Pfadlänge
- **Clustering-Koeffizient** eines Knotens:

$$C(i) = \frac{\left|\{(j,k) \in E \mid j,k \in N(i)\}\right|}{d(i)(d(i)-1)}, \quad N(i) = \{j \mid (i,j) \in E\}, \quad d(i) = \left|N(i)\right|$$

- Clustering-Koeffizient eines Graphen: avg(C(i))
- **Durchmesser**:
  - Längster kürzester Pfad zwischen zwei Knoten
- **charakteristische Pfadlänge**:
  - Durchschnittlicher kürzester Pfad zwischen zwei Knoten

Eigenschaften von Gnutella

- Untersuchung von 2000:

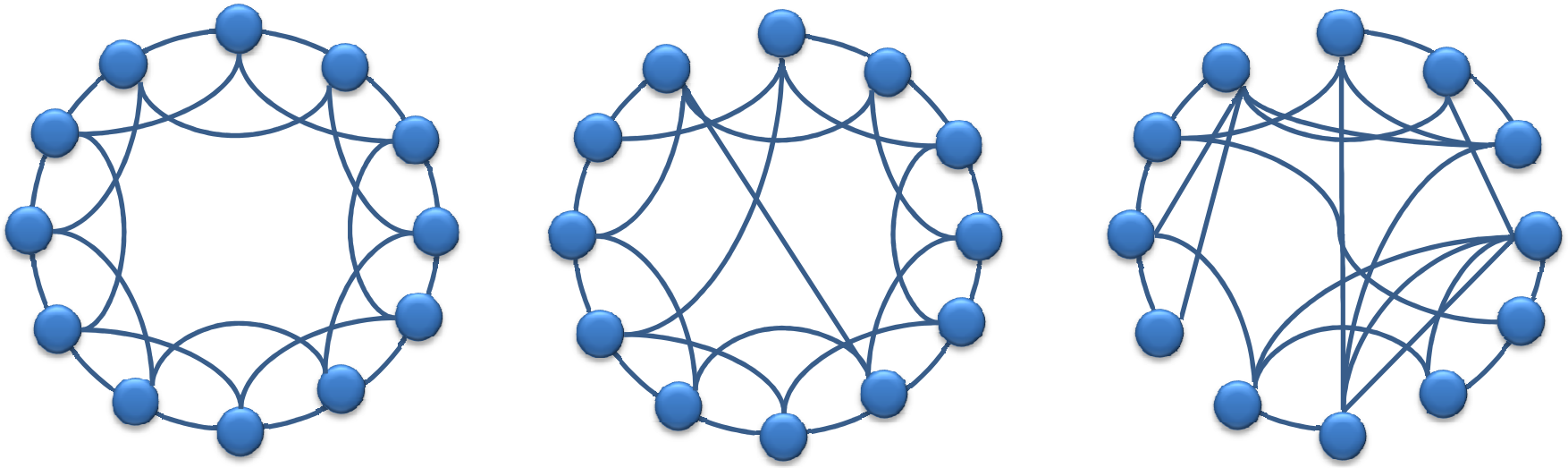
	Knoten	Kanten	Durch-messer	Clustering Koeffizient		charakteristische Pfadlänge	
	Gnutella			Gnutella	Zufallsgr.	Gnutella	Zufallsgr.
13.11.2000	992	2465	9	0.0351	0.0078	3.72	4.49
16.11.2000	1008	1782	12	0.0109	0.0056	4.43	5.54
20.12.2000	1077	4094	10	0.0652	0.0094	3.31	3.66
27.12.2000	1026	3752	8	0.0630	0.0102	3.30	3.71
28.12.2000	1125	4080	8	0.0544	0.0090	3.33	3.77

- Ergebnis: Clustering-Koeffizient deutlich größer als beim Zufallsgraph, charakteristische Pfadlänge ähnlich bis kleiner

Eigenschaften von Gnutella

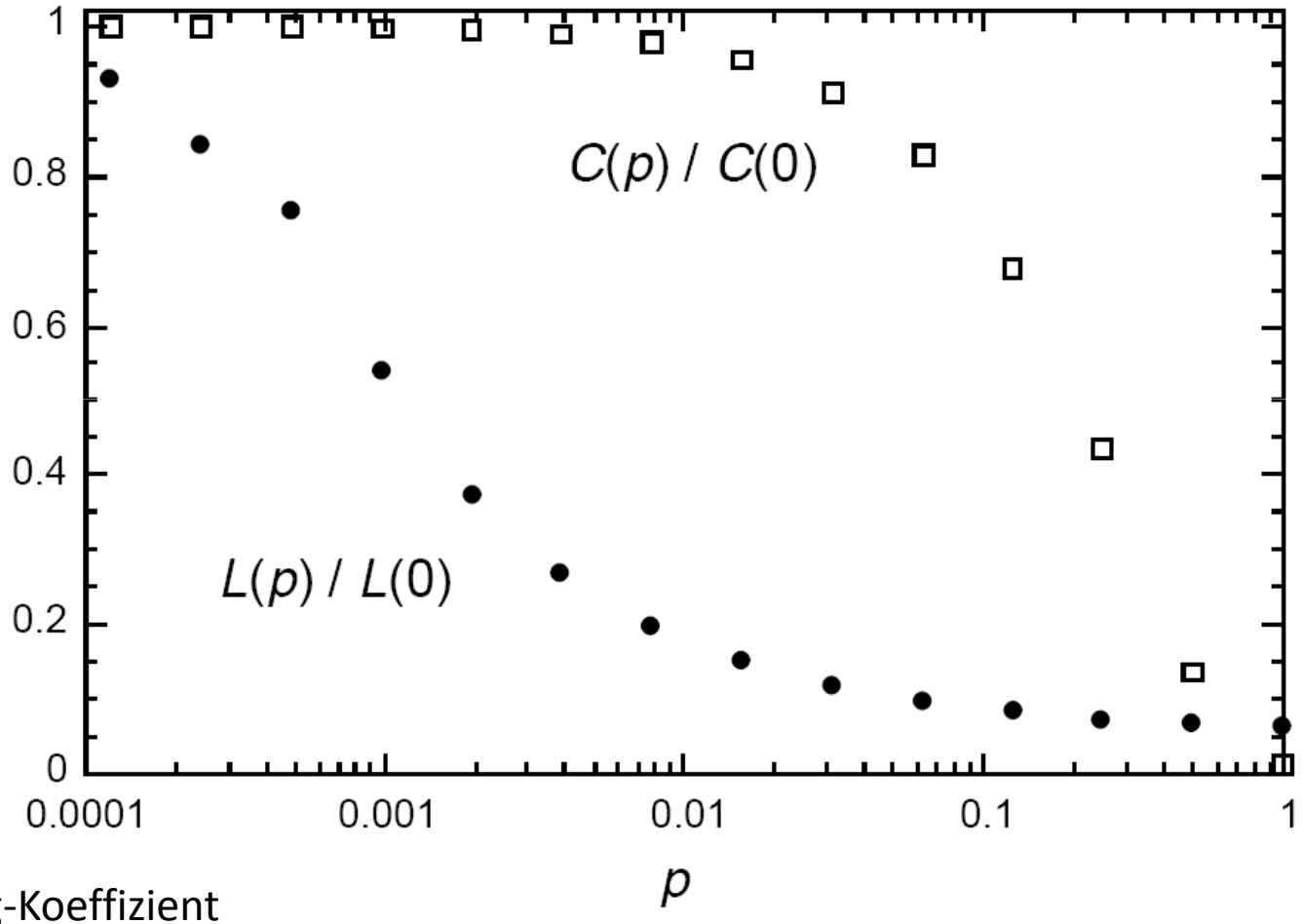
- Was hat man davon?
  - Schnelles Routing ist möglich (vgl. Milgram)
- Wie kommt das zustande?
  - Netzwerk-Modell von Watts und Strogatz:
    - Starte mit einem Ring, jeder Knoten ist mit m Nachfolgern verbunden
    - Tausche mit Wahrscheinlichkeit p jede Kante durch eine zufällige aus
    - Modelliert Übergang zwischen Ordnung und Chaos
  - Eine kleine Anzahl von zufälligen Links reicht aus, um den Durchmesser zu reduzieren

Modell von Watts und Strogatz



p=0 → p=1

Übergang bei Watts / Strogatz



C: Clustering-Koeffizient
L: Charakteristische Pfadlänge

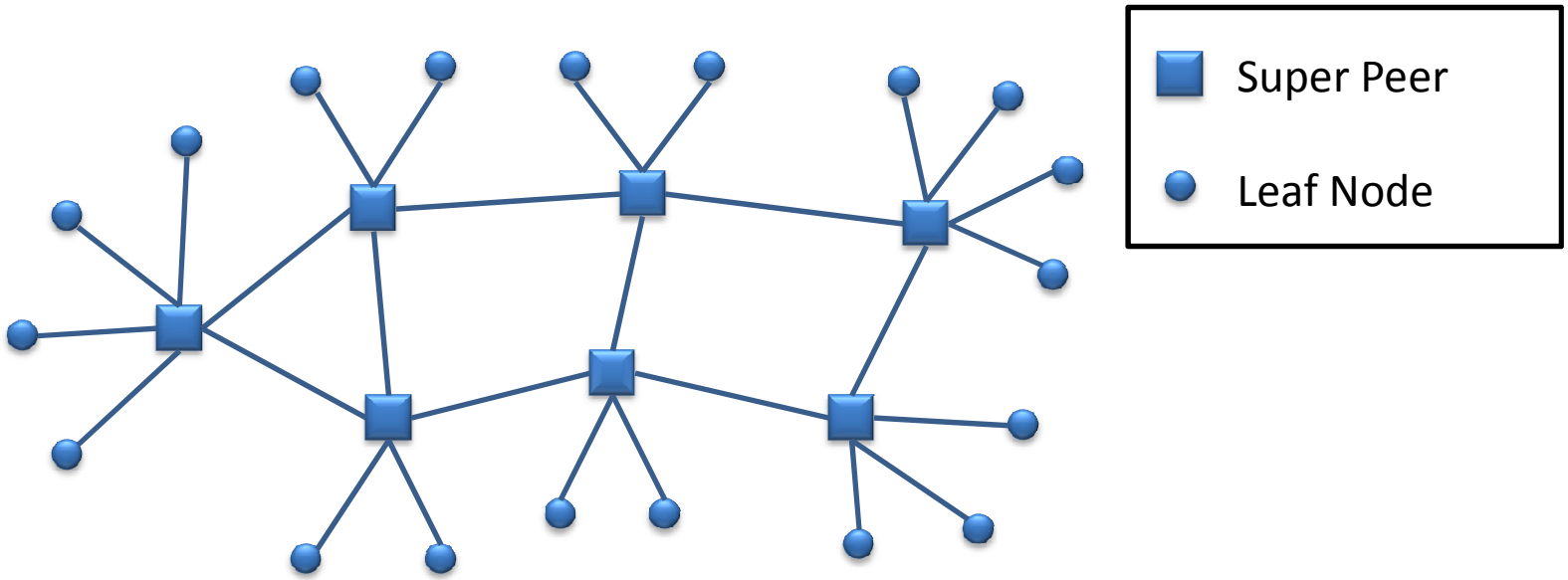
Zusammenfassung Gnutella

- Gnutella hat:
  - Gradverteilung nach Power-Law
  - Einen hohen Clustering-Koeffizienten
  - Einen kleinen Durchmesser

Gnutella erzeugt ein Small-World-Netzwerk
mit Power-Law Gradverteilung
Das passiert **selbstorganisierend**

Super-Peer Netze

- Idee: Kern-Netz aus "Super-Peers"
- Die "normalen" Knoten sind per Client/Server angebunden:



Super-Peer Netze

- Nachteile:
  - Nicht die reine Lehre
  - Super-Peers müssen hohe Last aushalten
  - Wer ist Super-Peer?
    - Automatische Auswahl vs. Selbstbestimmung
- Vorteile:
  - Weniger Fluktuation
  - Kleineres Netzwerk
  - Effizientere Suche
  - Alle P2P-Mechanismen lassen sich im Super-Peer Netz anwenden

Vorlesung
P2P Netzwerke
3: Verteilte Hash-Tabellen, Chord

Dr. Felix Heine
Complex and Distributed IT-Systems
felix.heine@tu-berlin.de

Inhalt

- Konstistentes Hashing
- Verteilte Hash-Tabellen
- Chord

Literatur

- Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan: "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications", IEEE/ACM Transactions on Networking, Vol. 11, No. 1, pp. 17-32, February 2003.
- Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, Hari Balakrishnan: "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications", MIT TR 819.

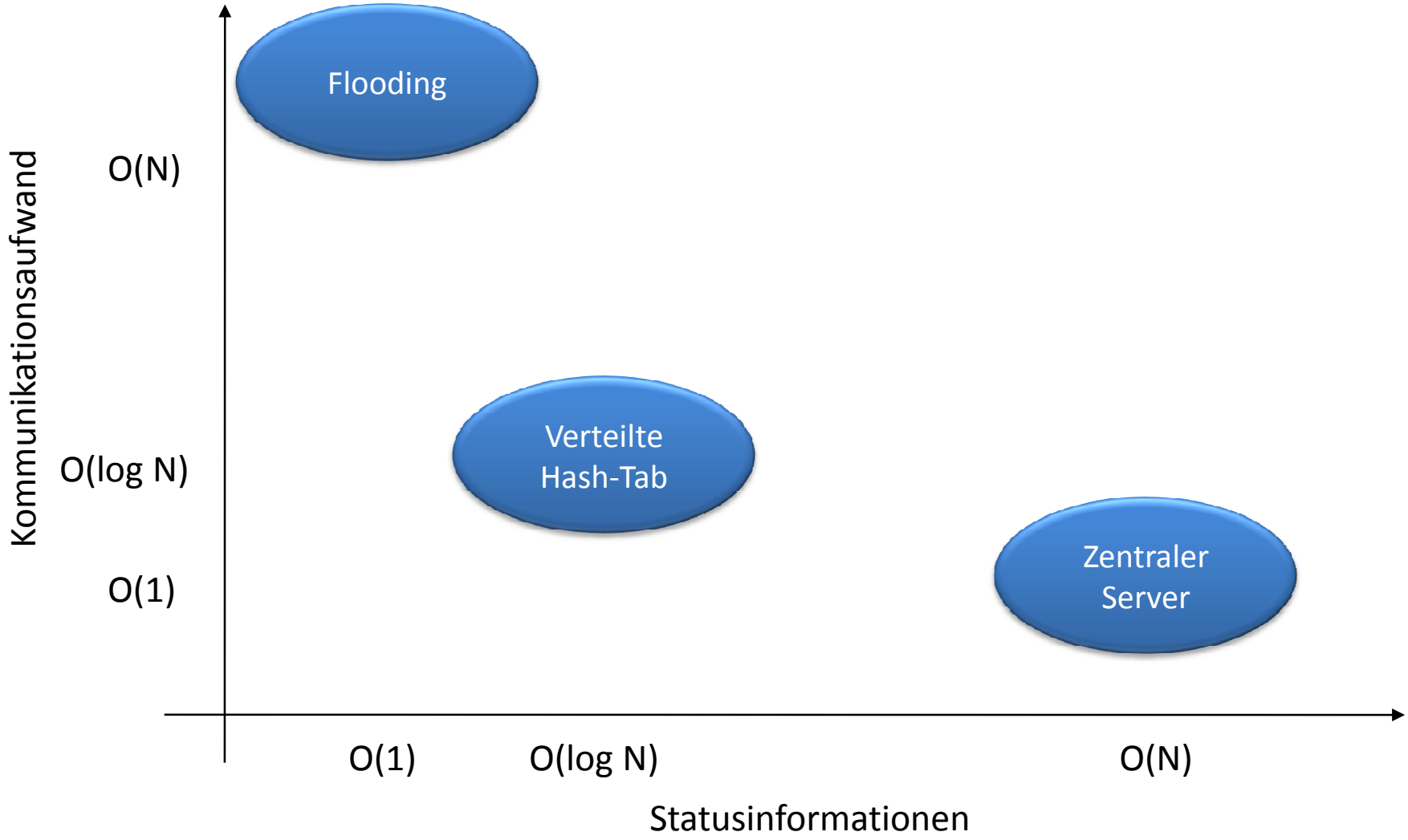
Auffinden von Objekten

- Gegeben:
  - Eine Menge von Knoten
  - Eine Menge von Objekten auf jedem Knoten
- Grundfragen:
  - Wo wird welches Objekt im Netzwerk **gespeichert**?
  - Wie wird das Objekt **gefunden**?
- Objekte? Was für Objekte?
  - Dateien
  - Verweise auf Dateien
  - Datenbanktabellen
  - Einträge in Datenbanktabellen
  - Informationen (unterschiedliche Syntax/Semantik)

Napster vs. Gnutella

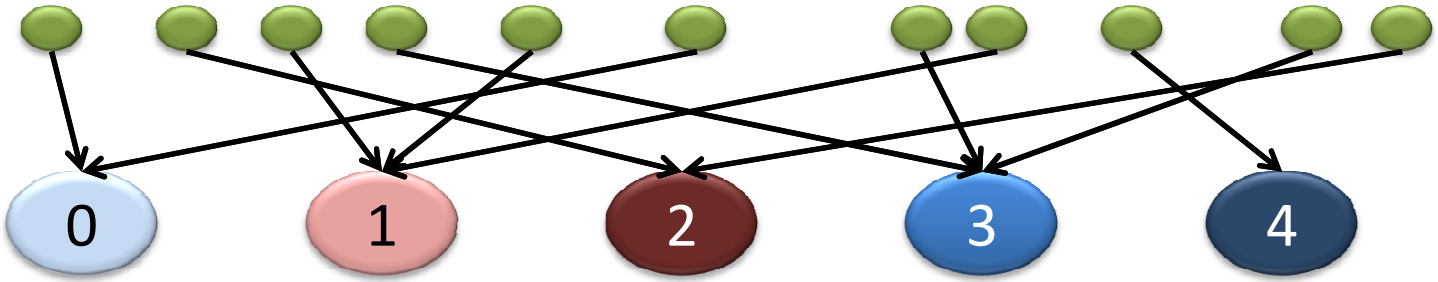
- Bisher betrachtet: Napster und Gnutella
- Napster:
  - Objekt: Metadaten zu Dateien
  - Speicherung und Suche über zentralen Server
  - → Probleme: keine Ausfallsicherheit, keine Skalierbarkeit
- Gnutella:
  - Objekt ist die Datei
  - Suche: Breitensuche durch das Netzwerk
  - → Probleme:
    - Entweder: Flooding, d.h. das ganze Netzwerk wird mit Suchanfragen überflutet
    - Oder: Begrenzung des Suchhorizontes (Time to Live), dann werden nicht alle Treffer gefunden

Napster vs. Gnutella



Konsistentes Hashing

- Grundidee: Knoten als Zellen in einer Hashtabelle auffassen
- Jedes Objekt hat einen Schlüssel
- Hash-Funktion bestimmt, welches Objekt auf welchen Knoten kommt
- Problem: Umsortierung nach Knotenankunft

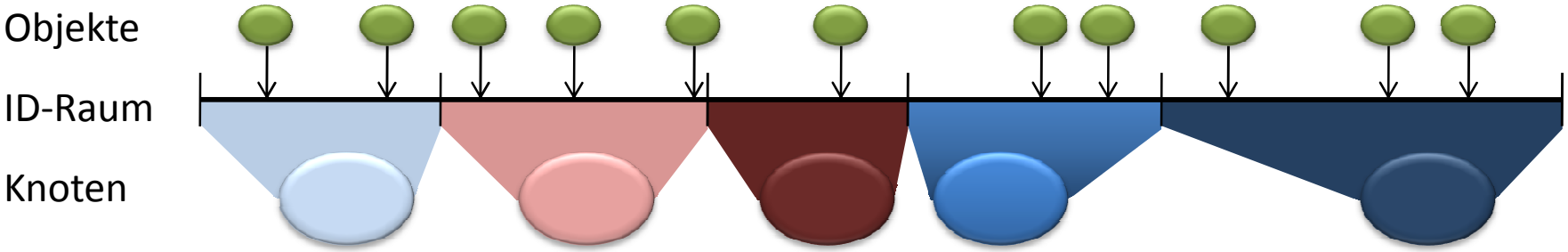


Konsistentes Hashing

- Wie realisiere ich eine Hash-Funktion, so dass bei Vergrößerung / Verkleinerung der Tabelle möglichst wenig Objekte umsortiert werden müssen?
- Idee: "Zwischenschicht einziehen":
  - Extrem große Hash-Tabelle, deren Größe nie geändert wird
  - Jeder Zelle der kleinen Tabelle entspricht einer *Menge von Zellen* in der großen Tabelle
  - Bei Größenänderungen der kleinen Tabelle werden nur wenige Objekte verschoben
  - Größenordnung O(X/N): X Anzahl Objekte, N Anzahl Zellen in der kleinen Tabelle

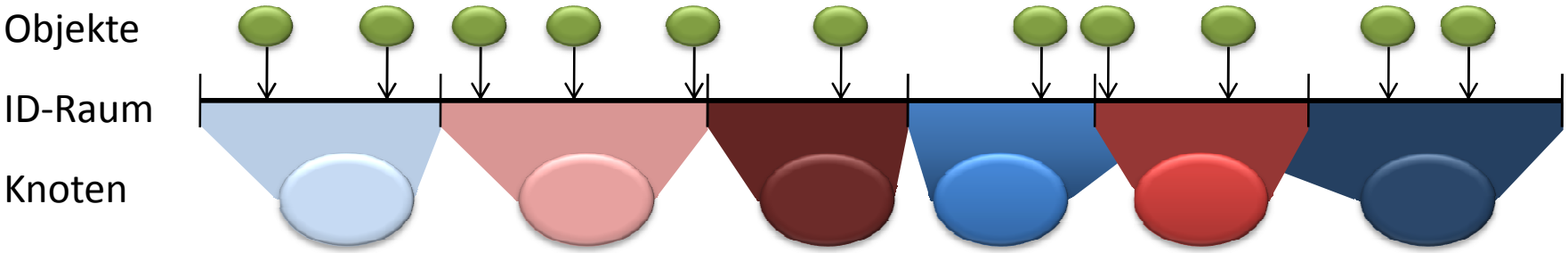
Verteilte Hashtabellen

- Grundidee:
  - Jedes Objekt im Netzwerk (z.B. Datei) bekommt eine ID
  - Jeder Netzknoten ist verantwortlich für einen Bereich von ID‘s
  - Das Netz ist so aufgebaut, dass der für eine ID verantwortliche Knoten schnell aufgefunden werden kann



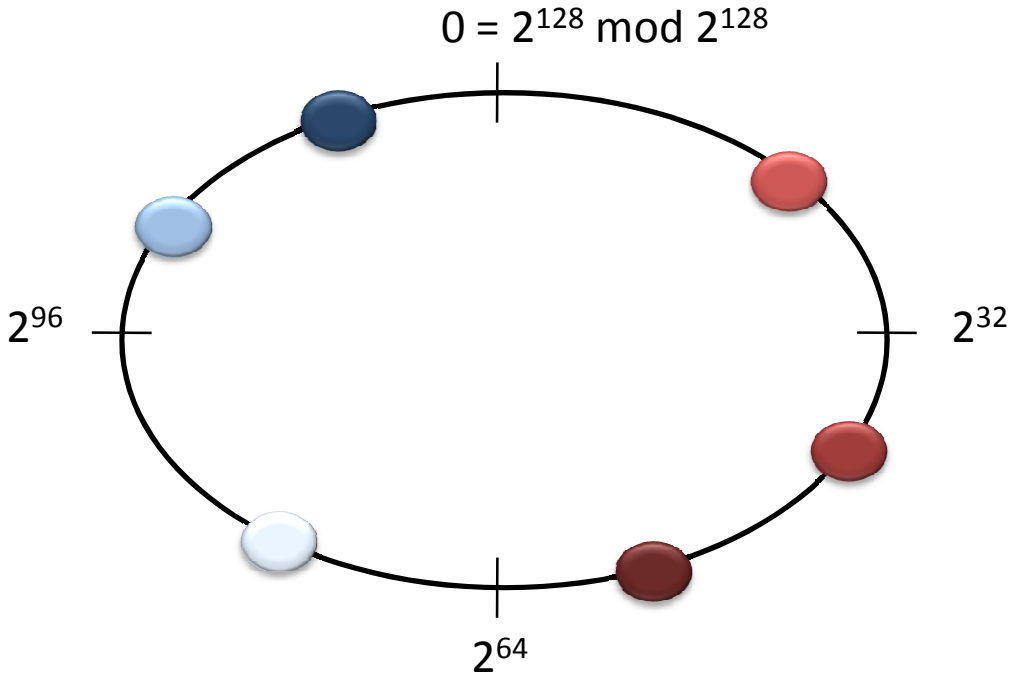
Verteilte Hashtabellen

- Grundidee:
  - Jedes Objekt im Netzwerk (z.B. Datei) bekommt eine ID
  - Jeder Netzknoten ist verantwortlich für einen Bereich von ID‘s
  - Das Netz ist so aufgebaut, dass der für eine ID verantwortliche Knoten schnell aufgefunden werden kann



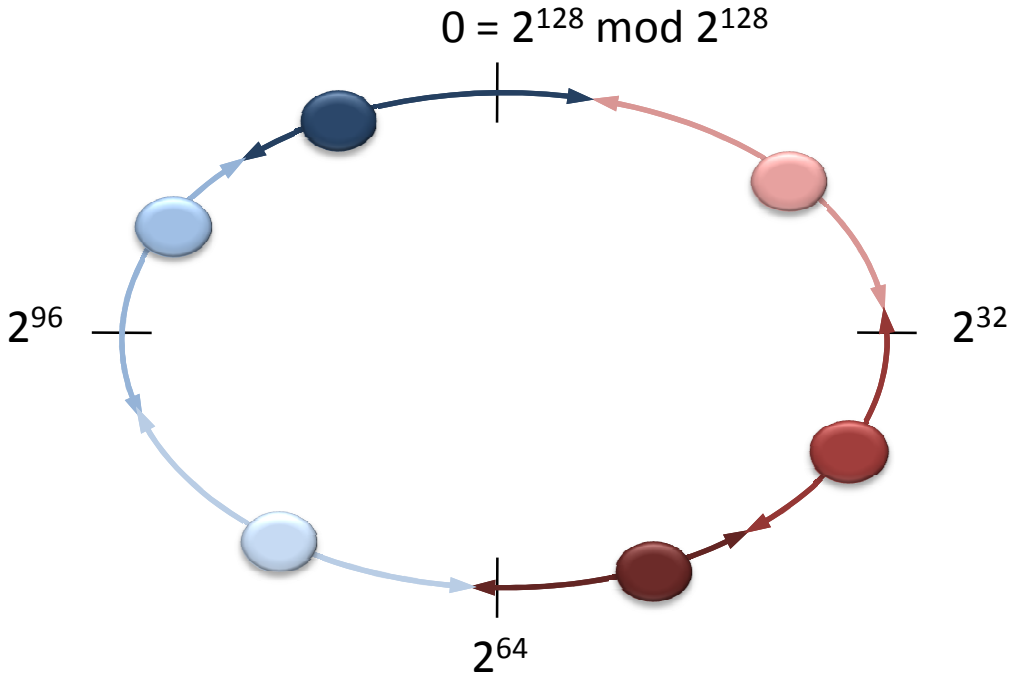
Verteilte Hashtabellen

- Wie sieht der ID-Raum aus?
  - Typisch: Ganzzahlig Numerisch, z.B. 0 bis 2^{128}-1
  - Jeder Knoten hat selbst eine ID aus dem Bereich
  - Jeder Knoten ist verantwortlich für einen Bereich
  - Der Raum ist zirkulär:



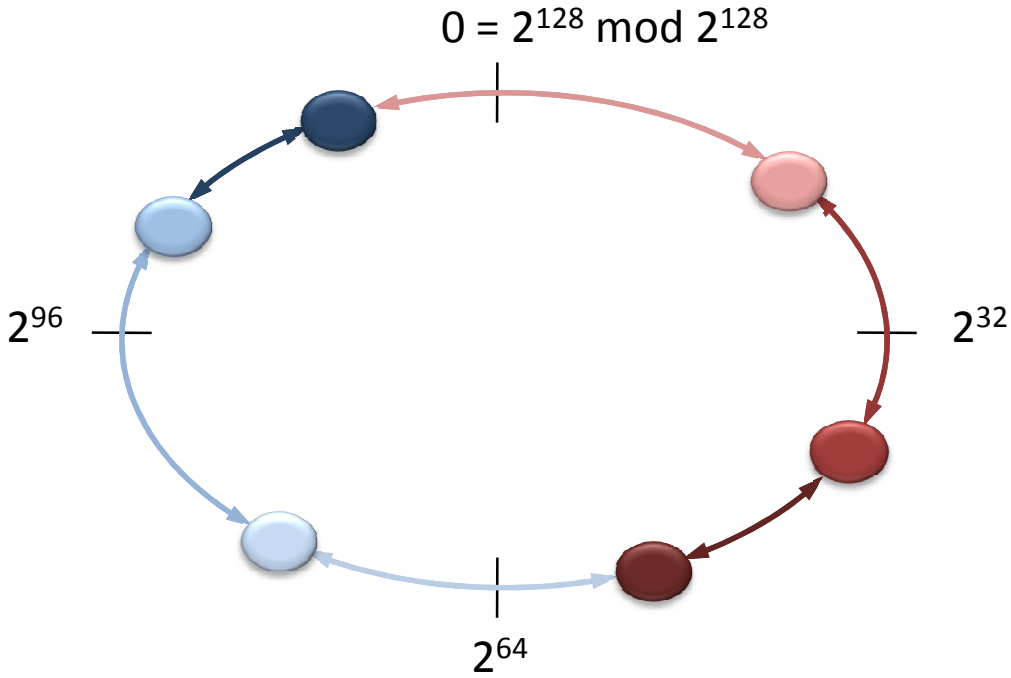
Verteilte Hashtabellen

- Wie sieht der ID-Raum aus?
  - Typisch: Ganzzahlig Numerisch, z.B. 0 bis 2^{128}-1
  - Jeder Knoten hat selbst eine ID aus dem Bereich
  - Jeder Knoten ist verantwortlich für einen Bereich
  - Der Raum ist zirkulär:



Verteilte Hashtabellen

- Wie sieht der ID-Raum aus?
  - Typisch: Ganzzahlig Numerisch, z.B. 0 bis 2^{128}-1
  - Jeder Knoten hat selbst eine ID aus dem Bereich
  - Jeder Knoten ist verantwortlich für einen Bereich
  - Der Raum ist zirkulär:



Verteilte Hashtabellen

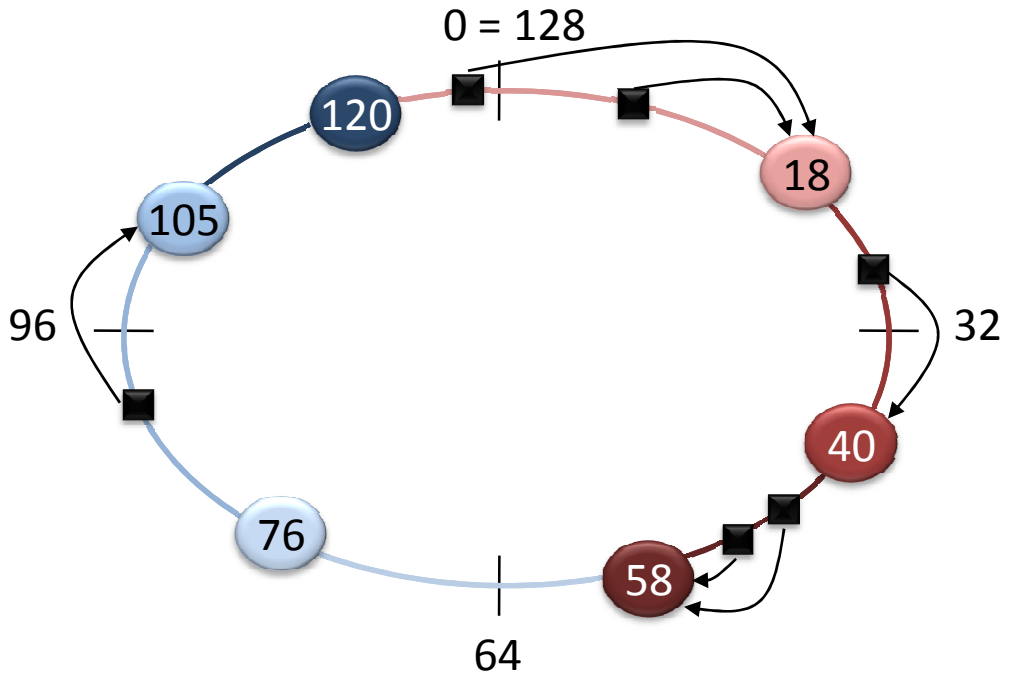
- Fragen:
  - ID-Raum
    - Aufbau
    - Verantwortlichkeit der Knoten
  - Routing
    - Verbindungsstruktur der Knoten
    - Wie wird der verantwortliche Knoten gefunden?
  - Dynamik
    - Integration eines neuen Knotens
    - Ausfall eines Knotens
- Weitere Fragen:
  - Lokalität
  - Lastverteilung
    - Unterschiedliche Performance
- Anwendungs-Aspekte
  - Was speichere ich?
  - Bestimmung der Objekt-ID

Anwendungs-Aspekte

- Speicherung
  - Alternativen: Daten oder Index
  - Daten:
    - Pro: Schneller Zugriff
    - Contra: Updates, Speicherbrauch wegen Redundanz
  - Index:
    - Pro: Aktualität, Effizientere Verteilung
    - Contra: Ein Routing-Schritt mehr
- Wahl der Objekt-ID
  - Anwendungsabhängig
  - So, dass der Suchende die ID bestimmen kann
    - Z.B. Hash des Objektnamens (Dateiname)
  - Auch mehrere unterschiedliche IDs pro Objekt
    - z.B. Bücher: Titel, Autor, ISBN-Nummer, Verlag, …
    - Oder noch feiner: jedes Wort im Titel, weitere Schlagwörter
  - Spielt eine Rolle bei der Lastverteilung

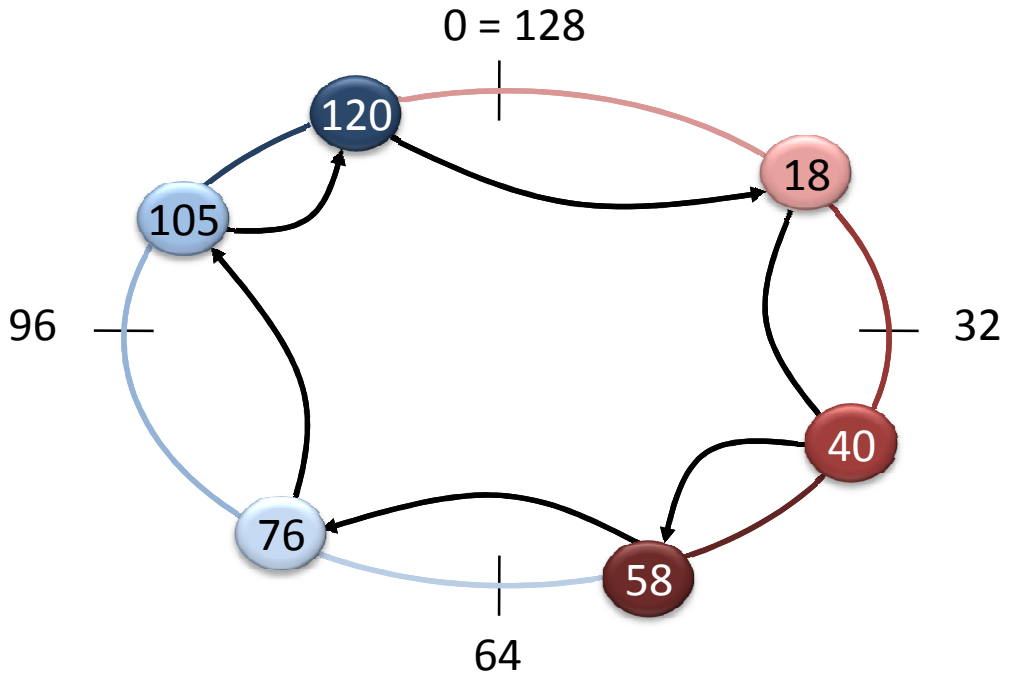
Chord

- Knoten sind in einem Ring organisiert
- Jeder Knoten ist für den Bereich **vor und inkl.** seiner eigenen ID zuständig
- Knoten ID wird zufällig ausgewählt



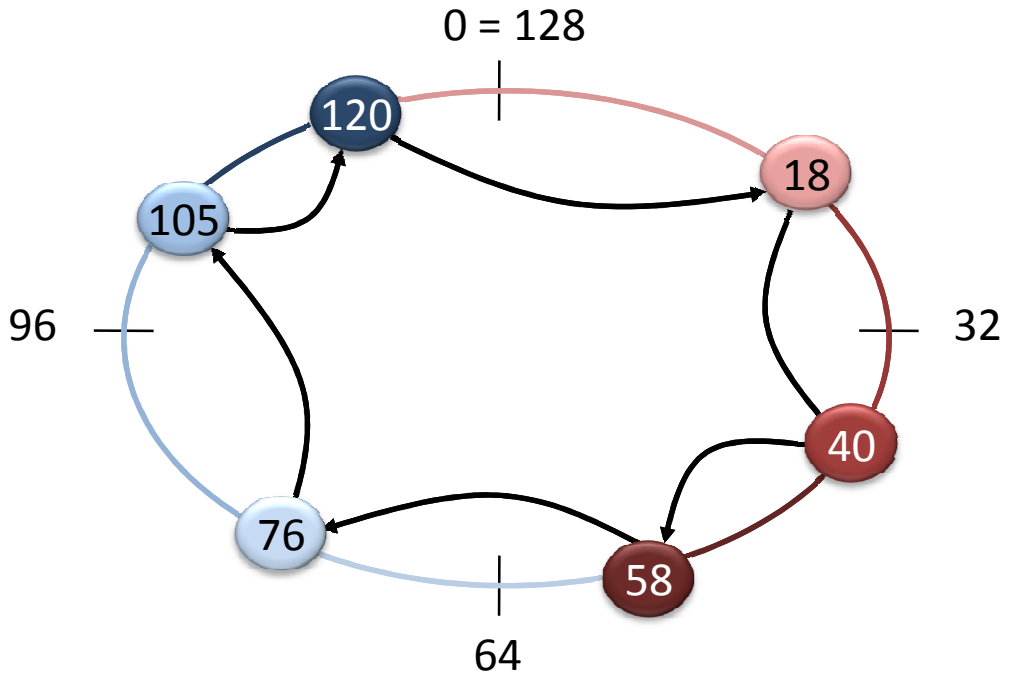
Chord: Routing

- Erster Ansatz: Successor-Pointer
- Jeder Knoten kennt seinen Nachfolger im Ring
- Wie funktioniert das Routing?



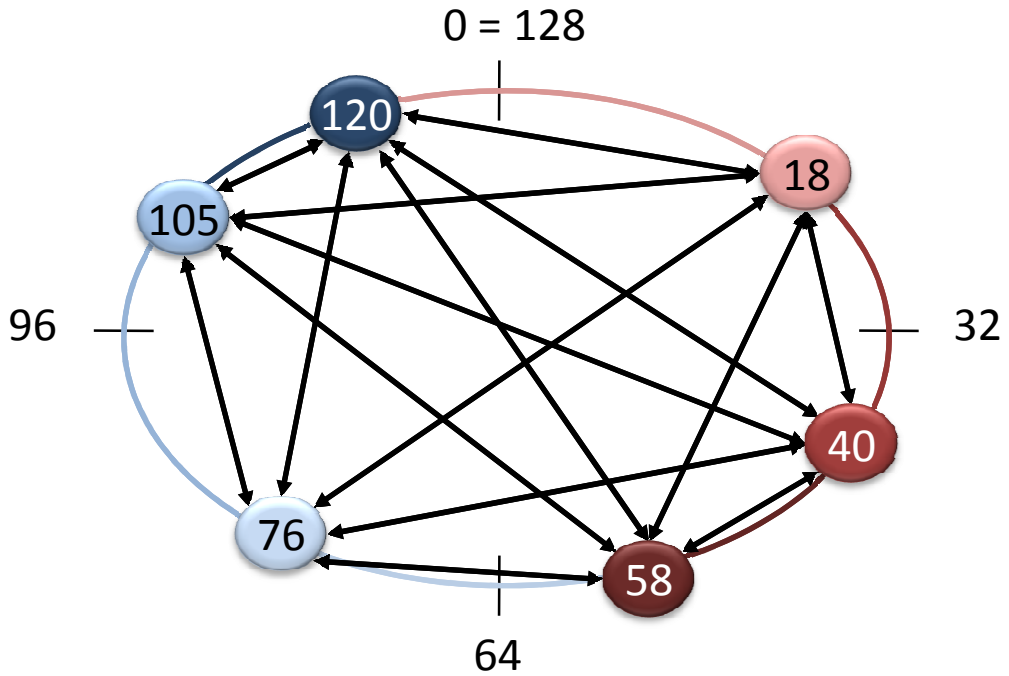
Chord

- Vorteil: Minimaler Knotenstatus
- Probleme:
  - Routing in O(N) Schritten
  - Netz fällt auseinander, wenn ein Knoten ausfällt



Chord: Routing

- Nächster Ansatz: Vollständige Vernetzung
- Routing in einem Schritt
- Perfekter Zusammenhalt
- Aber: **Nicht skalierbar!**

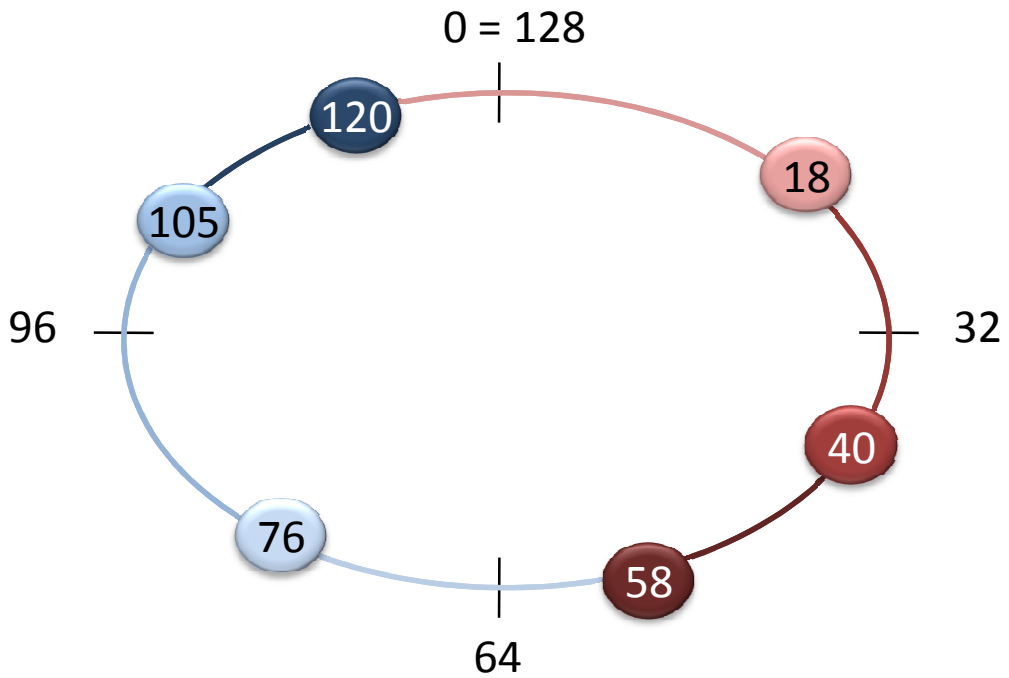


Chord: Routing

- Was wird gebraucht?
- Jeder Knoten muss eine geringe Anzahl an anderen Knoten kennen
- Z.T. direkt benachbarte Knoten, z.T. weit entfernte Knoten
- Lösung: Jeder Knoten kennt den Knoten im Abstand 1, 2, 4, 8, 16, 32, …
- Routing-Tabelle dazu heißt **Finger-Table**
- Und hat **log(maxID)** Einträge

Finger-Table Routing

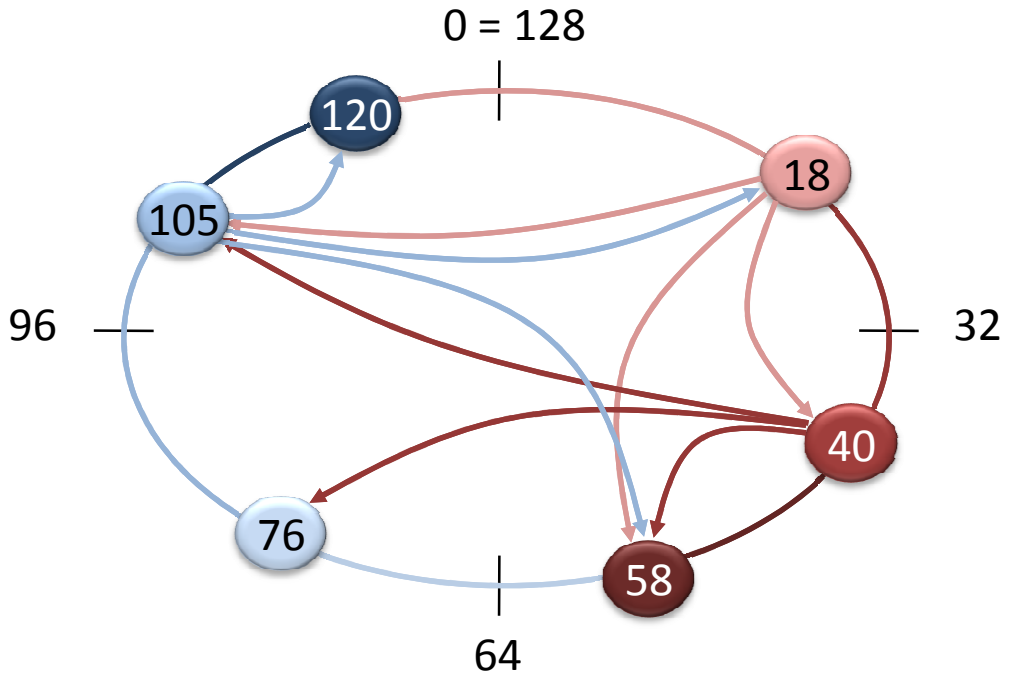
- Beispiel für Knoten 105:



Finger-Table Routing

- Beispiel für Knoten 105:

Offset	1	2	4	8	16	32	64
ID	106	107	109	113	121	9	41
Tats. Knoten	120	120	120	120	18	18	58



Routing in Chord

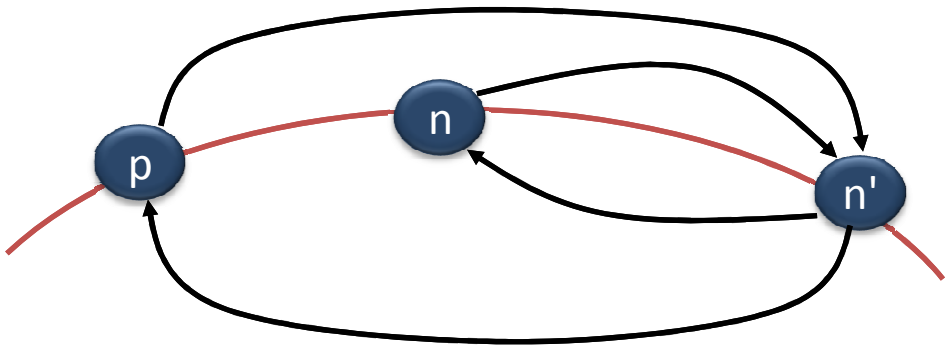
- Idee: *Suche den Knoten in der Finger-Table, der am dichtesten an der Zieladresse ist, aber nicht dahinter*
- Algorithmen:

```
1   // ask node n to find the successor of id
2   n.find successor(id)
3      if id in (n, successor]
4         return successor;
5      else
6         n‘ = closest_preceding_node(id);
7      return n‘.find_successor(id);
```

```
1   // search the local table for the highest predecessor of id
2   n.closest preceding node(id)
3      for i = m downto 1
4         if finger[i] in (n, id)
5              return finger[i];
6      return n;
```

Knotenankunft

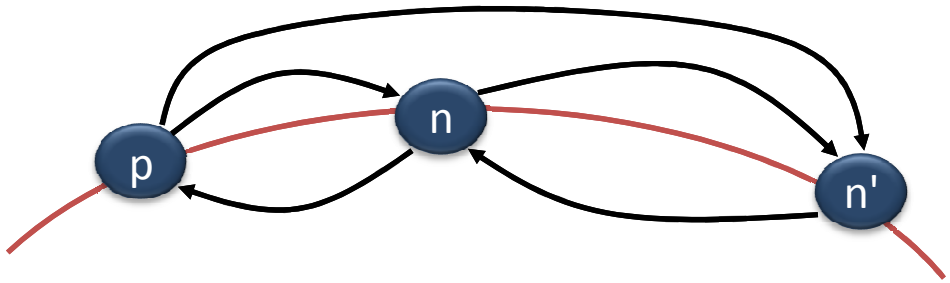
- Jeder Knoten hat zusätzlich einen **predecessor** Zeiger
- Identifier **n** bestimmen
- Neuer Knoten **n** muss *einen* Knoten **o** im Netz kennen
- **n** fragt **o** nach **successor(n)**
  - erhält damit seinen eigenen successor n'
- **n** kontaktiert **n'**, dadurch wird **predecessor(n')** aktualisiert



Stabilisierungsprotokoll

- Regelmäßige Kontrolle der successor / predecessor-Pointer

```
1    // called periodically. verifies n's immediate
2    // successor, and tells the successor about n.
3    p.stabilize()
4        n = successor.predecessor;
5        if n in (p, successor)
6            successor = n;
7        successor.notify(p);
```



Stabilisierungsprotokoll

```
1   // called periodically. verifies n's immediate
2   // successor, and tells the successor about n.
3   n.stabilize()
4       x = successor.predecessor;
5       if x in (n, successor)
6           successor = x;
7       successor.notify(n);
```

```
1   // n' thinks it might be our predecessor.
2   n.notify(n')
3       if (predecessor is nil or n' in (predecessor, n))
4           predecessor = n';
```

Stabilisierung der Finger Pointer

```
1    // called periodically. refreshes finger table entries.
2    // next stores the index of the next finger to fix.
3    n.fix_fingers()
4       next = next + 1;
5       if (next > m)
6          next = 1;
7       finger[next] = find_successor(n + 2^(next-1) );
```

Routing funktioniert auch mit defekter Finger-Tabelle korrekt!

Wie groß werden die Bereiche?

- Zufällige Wahl der ID's
- Ideal: bei N Knoten hat jeder Knoten MAXID/N IDs zu verwalten
- Wie sieht es wirklich aus?

- Normiere Ringgröße auf 1
- → Ideale Bereichsgröße ist 1/N
- Wahrscheinlichkeit, dass ein Bereich mehr als den Faktor log(N) größer ist, oder mehr als den Faktor N kleiner ist, ist sehr klein

Wie groß werden die Bereiche?

- Was ist "kleine" Wahrscheinlichkeit, was ist eine "hohe"?
- Definition:
  - Geringe Wahrscheinlichkeit: $\frac{1}{N^c}, c \geq 1$
  - Hohe Wahrscheinlichkeit: $1 - \frac{1}{N^c}, c \geq 1$

Je größer das Netzwerk, desto dichter ist eine kleine Wahrscheinlichkeit bei 0, und eine hohe Wahrscheinlichkeit bei 1.

Wie groß werden die Bereiche?

1. Die Wahrscheinlichkeit, dass ein beliebiges Intervall kleiner ist als $O(1/N^2)$, ist gering.
2. Die Wahrscheinlichkeit, dass ein beliebiges Intervall größer als $O(N*\log N)$ ist, ist gering.

Größe eines Intervalls: untere Schranke

Betrachte das Intervall I der Größe $\frac{1}{N^2}$ hinter einem Knoten.

E_i : Ereignis, dass Knoten i in dieses Intervall fällt

$$\Rightarrow P(E_i) = \frac{1}{N^2}$$

E : Ereignis, dass irgendein Knoten in I fällt

$$\Rightarrow P(E) = P(\bigcup_{i=1}^{N} E_i) \leq \sum_{i=1}^{\mathrm{N}} P(E_i) = N \frac{1}{N^2} = \frac{1}{N}$$

Größe eines Intervalls

Betrachte das Intervall I der Größe $\frac{1}{N}\log(N)$ hinter beliebigem Knoten

E_i : Ereignis, dass Knoten i in dieses Intervall fällt

$$\Rightarrow P(E_i) = \frac{1}{N}\log(N)$$

$\overline{E_i}$: Ereignis, dass Knoten i nicht in I fällt

$$\Rightarrow P(\overline{E_i}) = 1 - P(E_i) = 1 - \frac{1}{N}\log(N)$$

$\overline{E}$: Ereignis, dass kein Knoten in I fällt

$$\Rightarrow P(\overline{E}) = P(\bigcap_{i=1}^{N} \overline{E_i}) = P(\overline{E_1})^N = (1 - \frac{1}{N}\log(N))^N = (1 - \frac{\log(N)}{N})^{N\frac{\log(N)}{\log(N)}}$$

$$= (1 - \frac{1}{m})^{m\log(N)} = \left[(1 - \frac{1}{m})^m\right]^{\log(N)} \leq \left[\frac{1}{e}\right]^{\log(N)} = \frac{1}{e^{\frac{\ln(N)}{\ln(2)}}} = \frac{1}{N^{\frac{1}{\ln(2)}}} \approx \frac{1}{N^{1,44}}$$

Wieviele unterschiedliche Finger-Zeiger gibt es?

- Betrachte wieder auf 1 normierten Ring
- Finger-Zeiger zeigen auf +½, +¼, +1/8, +1/16, …
- Ab dem Finger dessen Ziel kleiner ist als die Intervallgröße, sind alle Finger Identisch (Nachfolger)
- Als suchen wir die Finger für die gilt:

$$\frac{1}{2^x} > \frac{1}{N^2}$$

$$\Leftrightarrow 2^x < N^2 \Leftrightarrow \log_2(2^x) < \log_2(N^2)$$

$$\boxed{\Leftrightarrow x < 2\log_2(N)}$$

- also: O(log N) unterschiedliche Zeiger

Wie schnell ist das Routing?

- O(log N) Schritte mit hoher Wahrscheinlichkeit
  - s Startknoten, k Ziel-Key, t letzter Knoten vor k
    - ♦ ➔ successor(t) ist der Zielknoten
  - Betrachte Finger-Pointer von s
    - ♦ i-tes Finger-Intervall $s+2^{i-1}$, $s+2^{i}$
  - t Liegt in einem dieser Intervalle, sei i so gewählt, dass $s+2^{i-1} < t \leq s+2^{i}$
  - ➔ i-ter Finger-Pointer f ist ≤ t
  - Nachricht wird an f geroutet
  - $|f-t| \leq 2^{i-1}$ Restdistanz
  - $|s-f| \geq 2^{i-1}$ überbrückte Distanz
  - ➔ Restdistanz ist kleiner als überbrückte Distanz

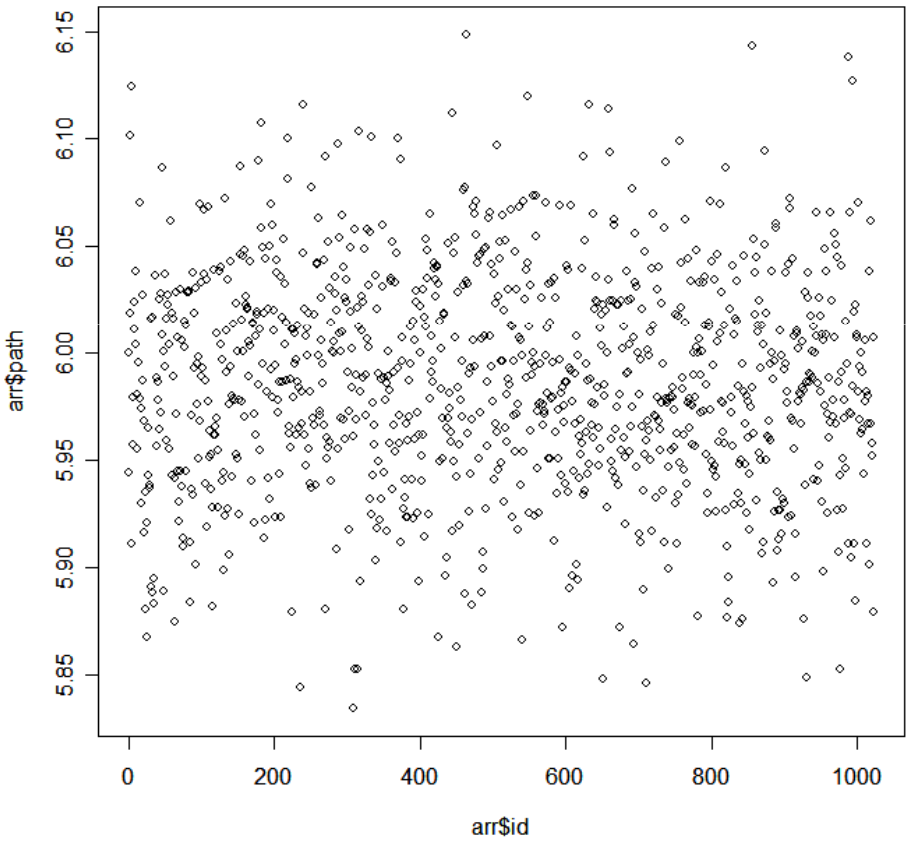
Wie schnell ist das Routing?

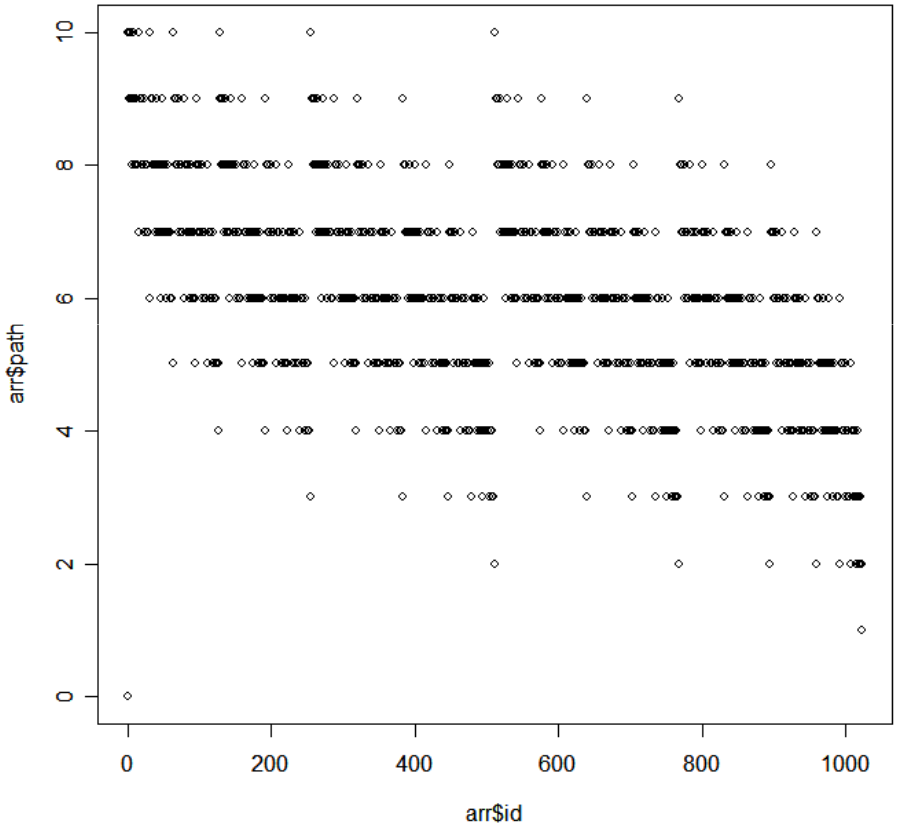
- In jedem Schritt wird die Restdistanz mind. halbiert
- Startdistanz ist max. 1 (normierter Ring)

Schritte	1	2	3	…	log N	2log N
Restdistanz	1/2	$1/2^2$	$1/2^3$	…	$1/2^{\log N}=1/N$	$1/2^{2\log N} = 1/N^2$

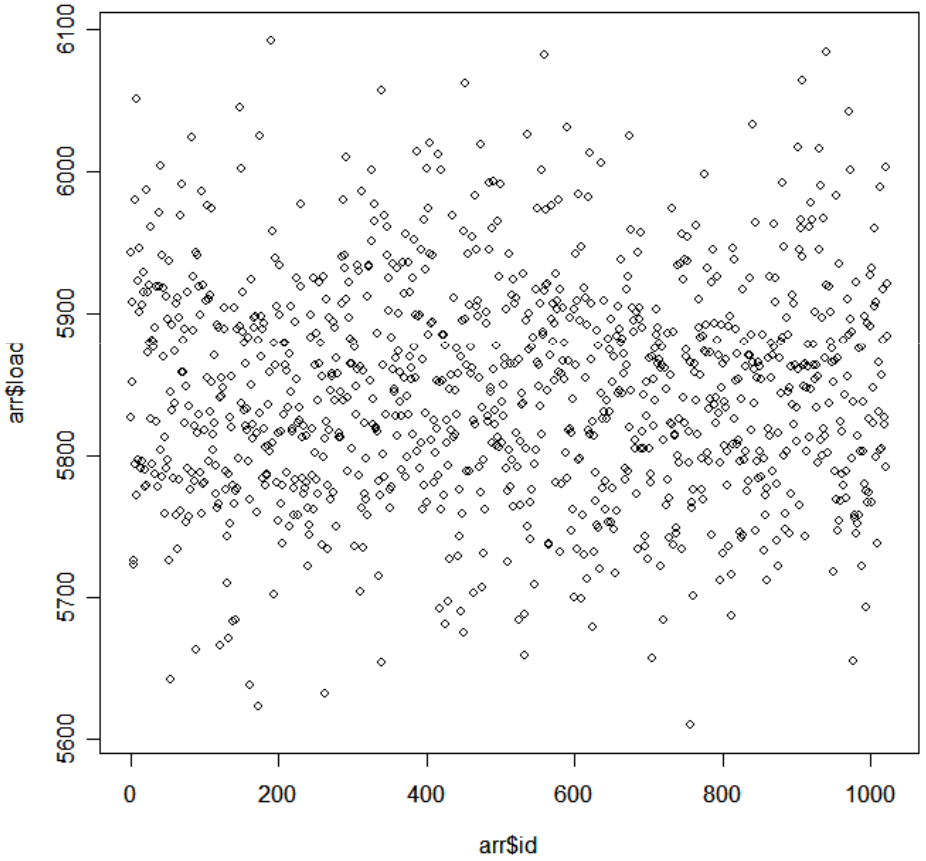
- Nach 2 log N Schritten ist die Restdistanz 1/N^2
- Die Wahrscheinlichkeit, dass in einem solchen Intervall mehr als ein Knoten ist, ist gering
- ➔ Routing erfolgt in O(log N) Schritten mit hoher Wahrscheinlichkeit

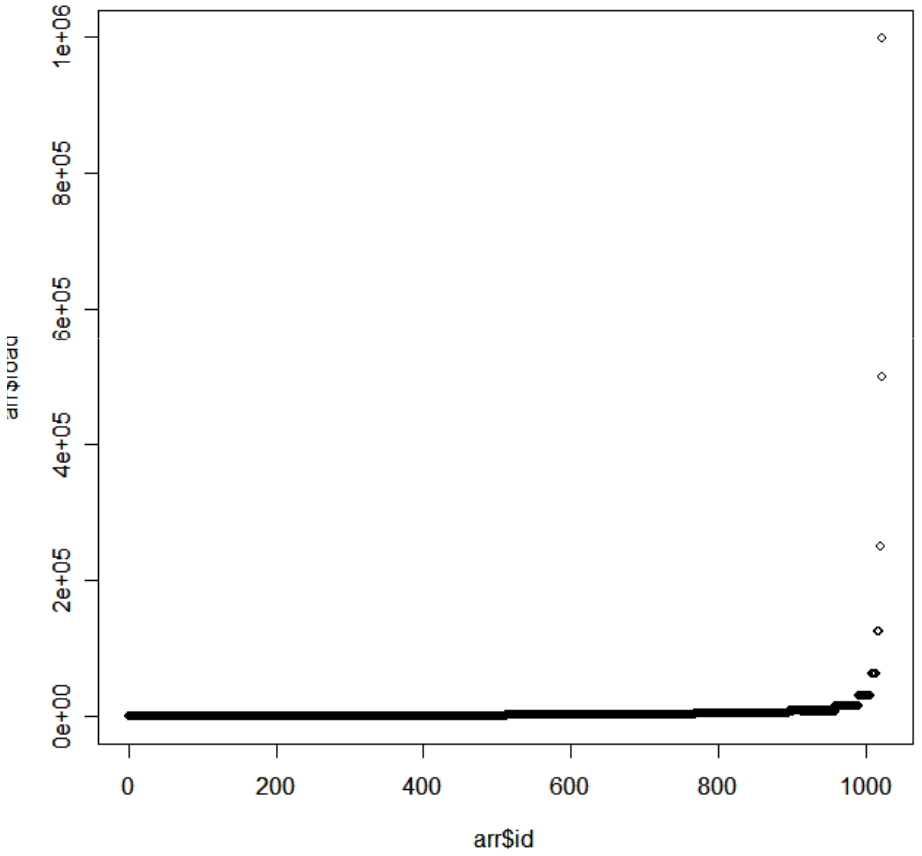
Reguläre vs. Deformierte IDs: Pfadlänge





Reguläre vs. Deformierte IDs: Last



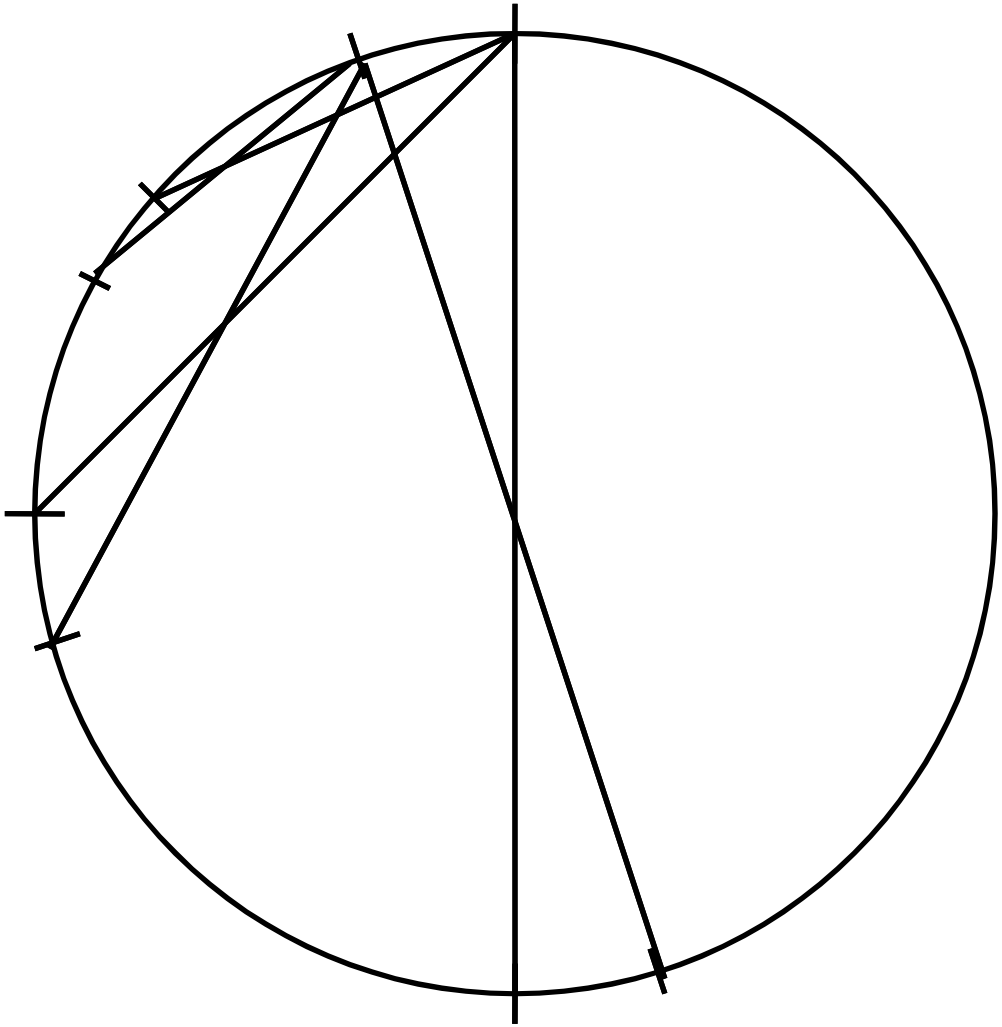


Kosten der Knotenankunft

- Finger-Pointer von Vorgänger übernehmen, anpassen
- Wieviele Finger zeigen auf mich?
- Wie passe ich diese aktiv an?

➔ $O(\log^2 N)$ Aufwand ➔

Welche Finger zeigen auf mich?



- Der eigene Bereich kann um Faktor log(N) vergrößert sein
- Auf diesen zeigen log(N) Bereiche, jeweils der Größe log(N)*1/N
- Also zeigen log(N)*log(N) Knoten auf mich im schlimmsten Fall
- Diese können linear durchlaufen werden

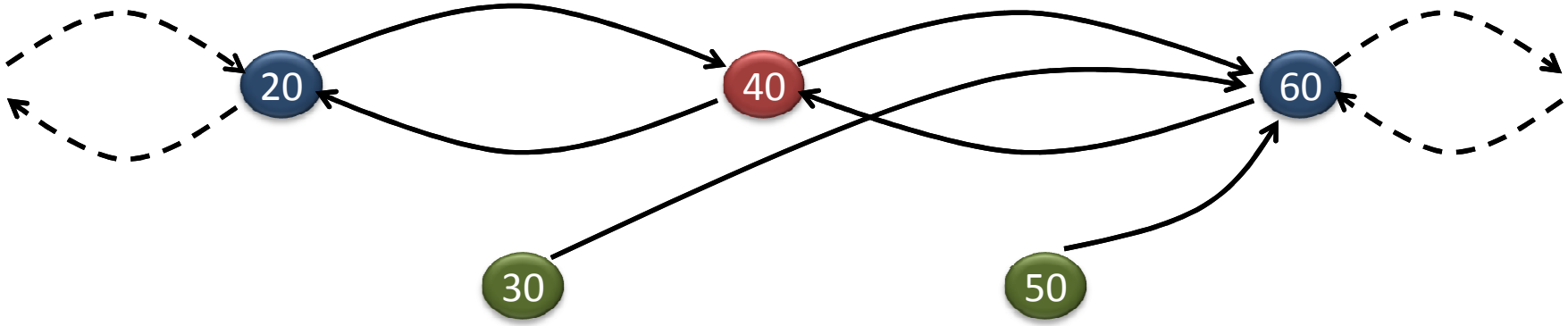
Chord: Knotenausfall

- Knoten bemerkt, dass ein Successor ausgefallen ist
- Woher bekommt er neuen Successor?
- Lösung: Liste von x Nachfolgern
- Liste wird regelmäßig gegen den Uhrzeigersinn weitergegeben und aktualisiert
- Wichtig: Routing ist korrekt, auch wenn Finger-Pointer noch nicht korrigiert!

- Was passiert bei hohem "Churn"?

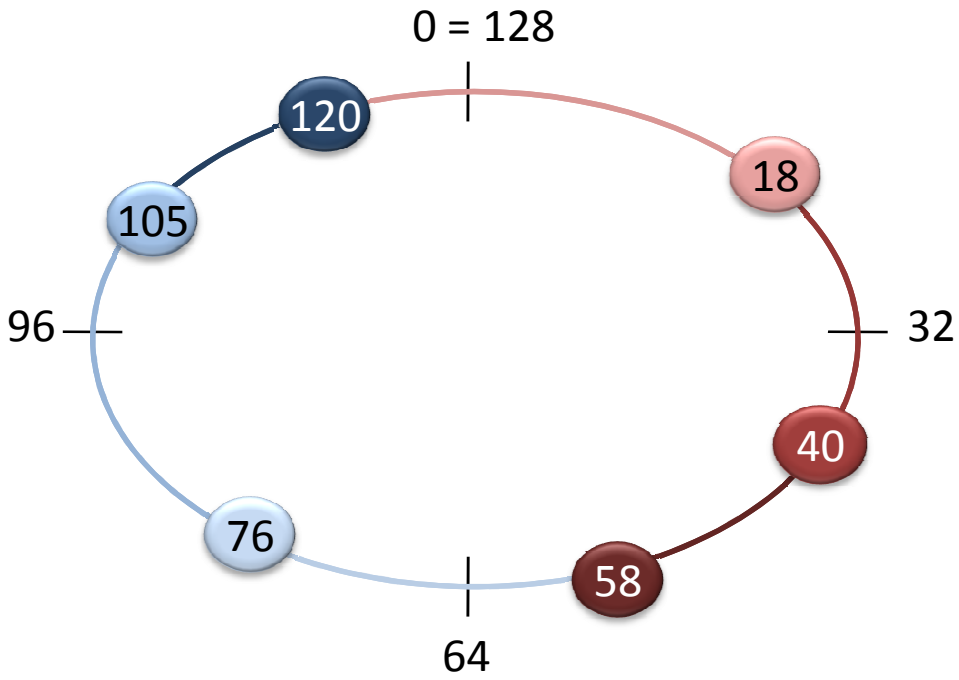
Chord: Hoher Churn

- Beispiel:



Replika-Knoten

- Nach Knotenausfall wird der Nachfolger verantwortlich für den Bereich des ausgefallenen Knotens
- Daher: Replikation der Daten auf Nachfolger

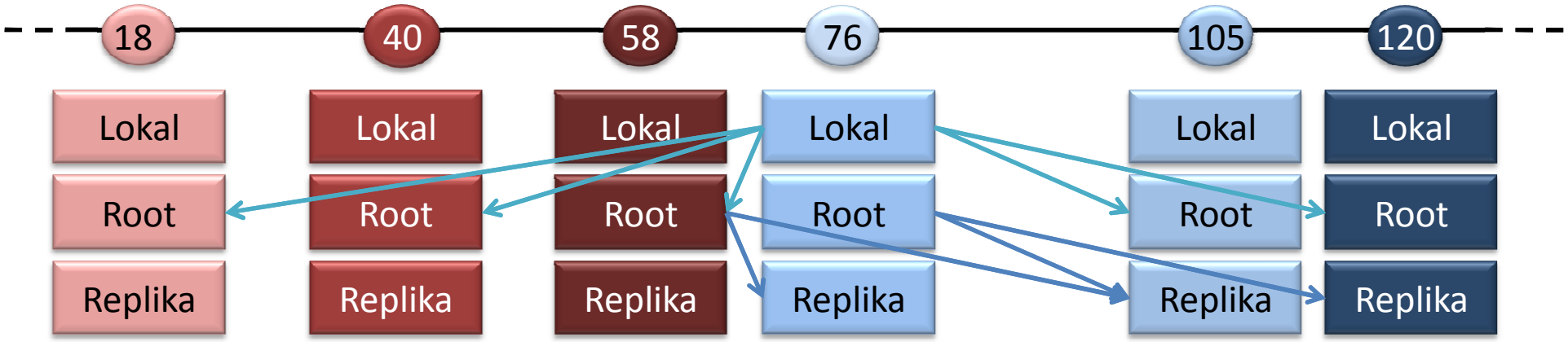


Knoten	Root	R1	R2
18	[121,18]	[106,120]	[77,105]
40	[19,40]	[121,18]	[106,120]
58	[41,58]	[19,40]	[121,18]
76	[59,76]	[41,58]	[19,40]
105	[77,105]	[59,76]	[41,58]
120	[106,120]	[77,105]	[59,76]

Speichern von Daten

- Jeder Knoten hat lokale Daten (z.B. Dateien)
- Er publiziert diese Daten / Indizes im Netzwerk
- Das ganze geht über sog. "Soft-State Updates"
  - Daten haben auf dem Zielknoten ein Timeout
  - Der Original-Knoten muss sie regelmäßig erneuern
- Wenn ein Knoten ausfällt, verschwinden seine Daten daher nach einiger Zeit
- Der für ein Datum verantwortliche Knoten (Root-Knoten) verteilt die Daten weiter an die Replika-Knoten

Datenfluss



Knotenankunft

- Problem: Knoten bekommt Verantwortung für einen Bereich
- Er hat aber noch keine Daten für diesen Bereich
- Lösungsmöglichkeiten:
  - Solange bis alle Daten vorhanden sind, werden Anfragen an alten Knoten weitergeleitet
  - Erst ins Netzwerk einhängen, wenn die Daten da sind
  - Problem ignorieren und auf Soft-State Updates warten

Nachteile von DHTs

- Ohne explizite Behandlung in der Anwendung:
  - Keine Lokalität der Daten: Meier vs. Mayer
  - Un-exakte Suche unmöglich
  - Bereichssuche unmöglich
- Verwaltungsaufwand beim Knotenankunft / ende
  - Daher schlecht bei hoher Dynamik ("Churn")

Interfaces zu DHTs

- Zwei Sichtweisen:
  - Pakete zu einer ID Routen
  - Hashtabelle
- Pakete Routen:
  - send(destId, msg)
  - msg ← receive()
  - Best-effort service
- Hashtabelle
  - put(key, value)
  - value ← get(key)

Zusammenfassung

- DHTs verschieben Daten auf wohldefinierte Knoten
- Dadurch ist effiziente Suche nach Daten möglich
- Zur Zuordnung von Daten zu Knoten wird konsistentes Hashing verwendet
- Fragestellungen für DHTs:
  - Aufbau ID-Raum
  - Verbindungsstruktur
  - Routing-Mechanismus
  - Dynamik: Knotenankunft / Knotenausfall
- Interfaces zu DHTs

Vorlesung
P2P Netzwerke
4: Content Addressable Network

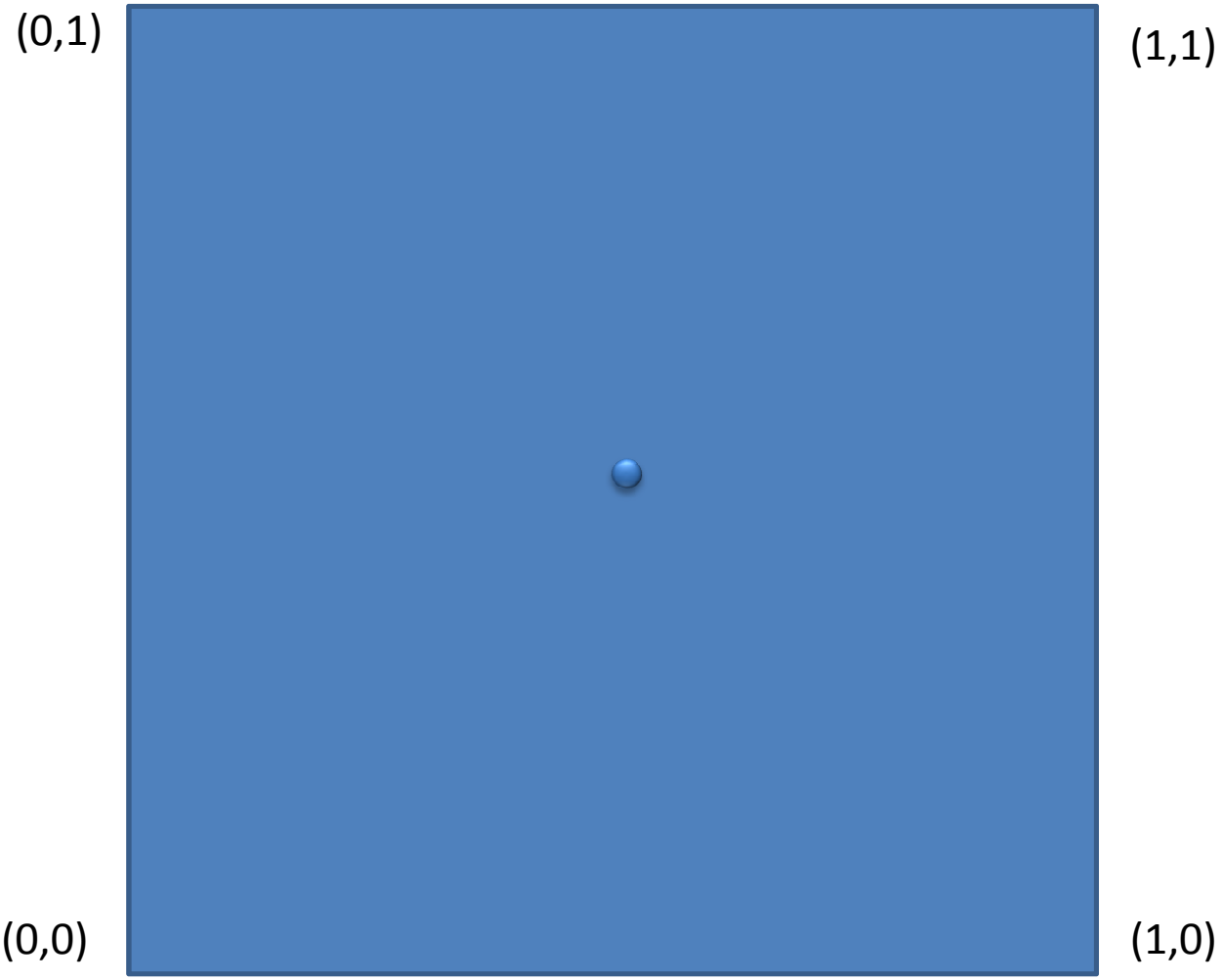
Dr. Felix Heine
Complex and Distributed IT-Systems
felix.heine@tu-berlin.de

CAN: Content Addressable Network

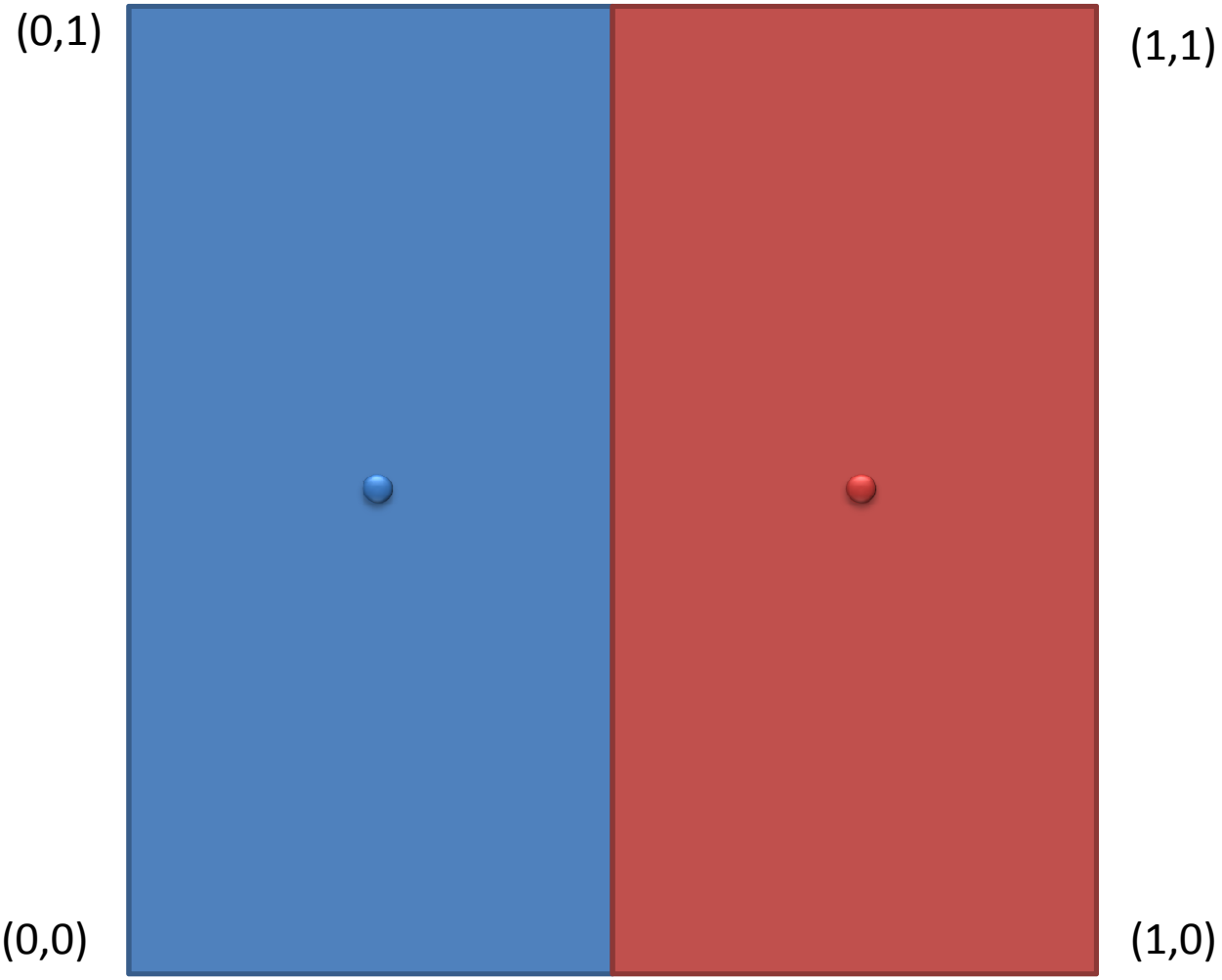
- Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker: "A Scalable Content-Addressable Network", SIGCOMM'01, 2001.

- Alternativer Ansatz:
  - Adresse eines Knotens ist eine 2-dimensionale Koordinate
- Bereiche werden bei Knotenankunft geteilt

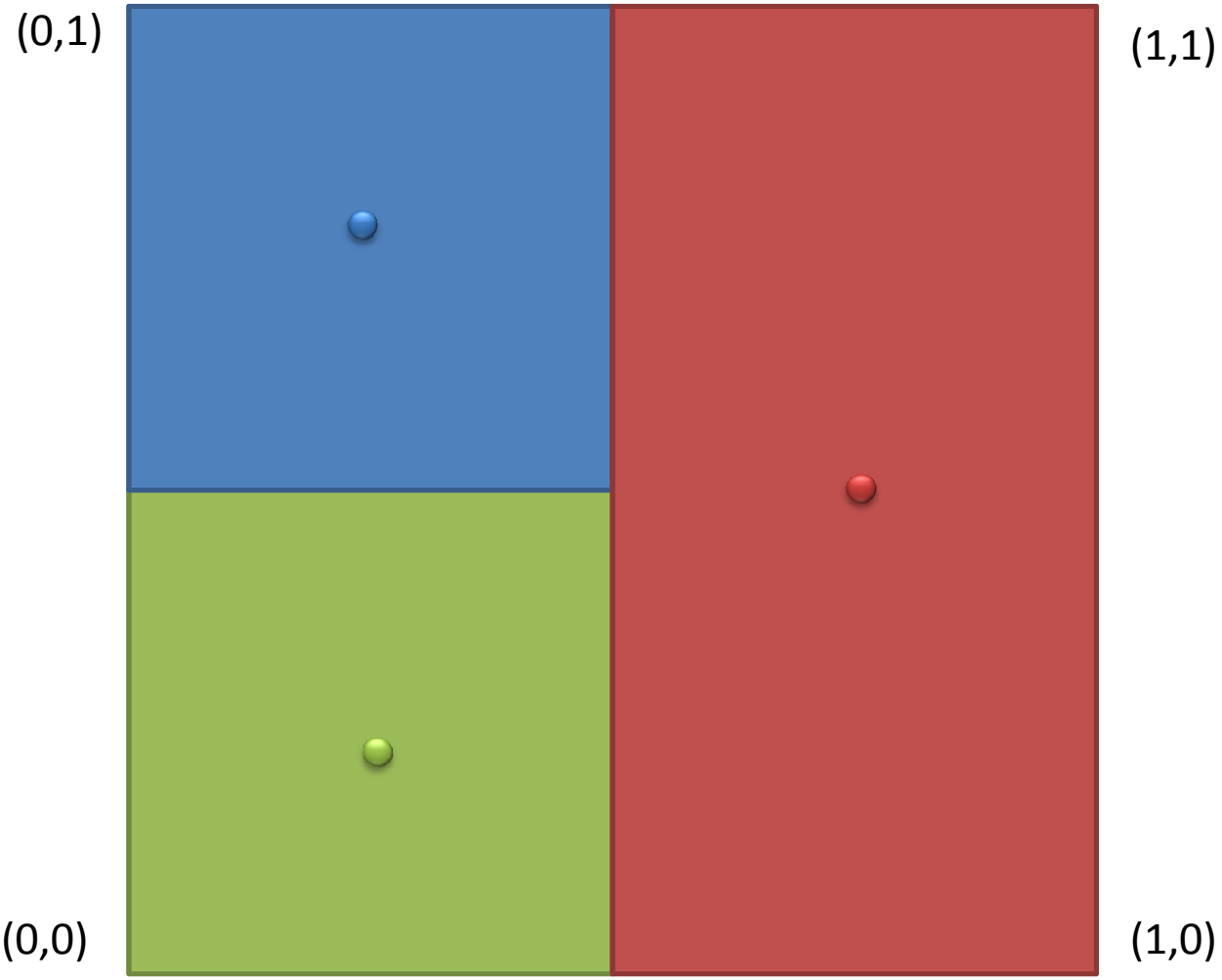
CAN



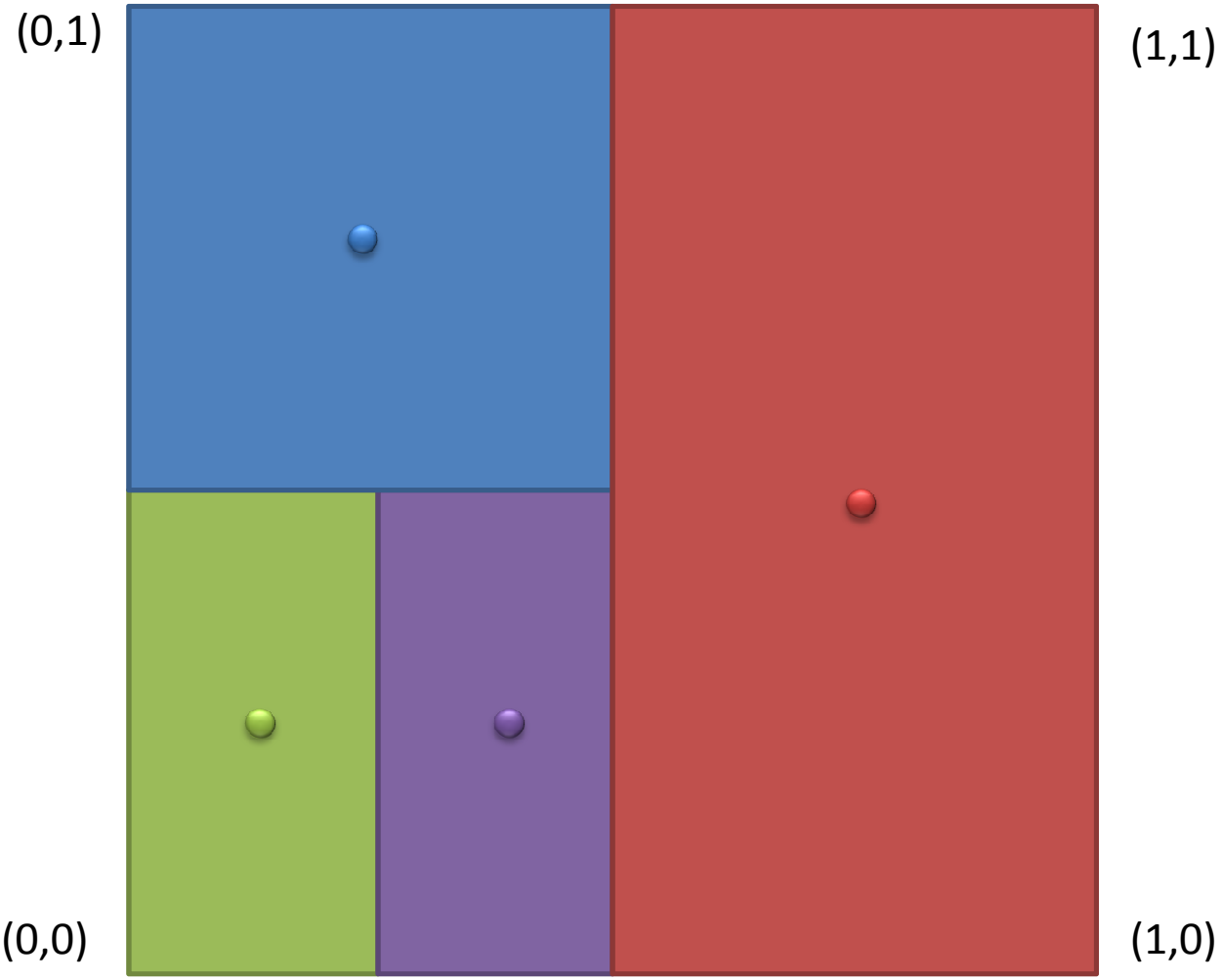
CAN



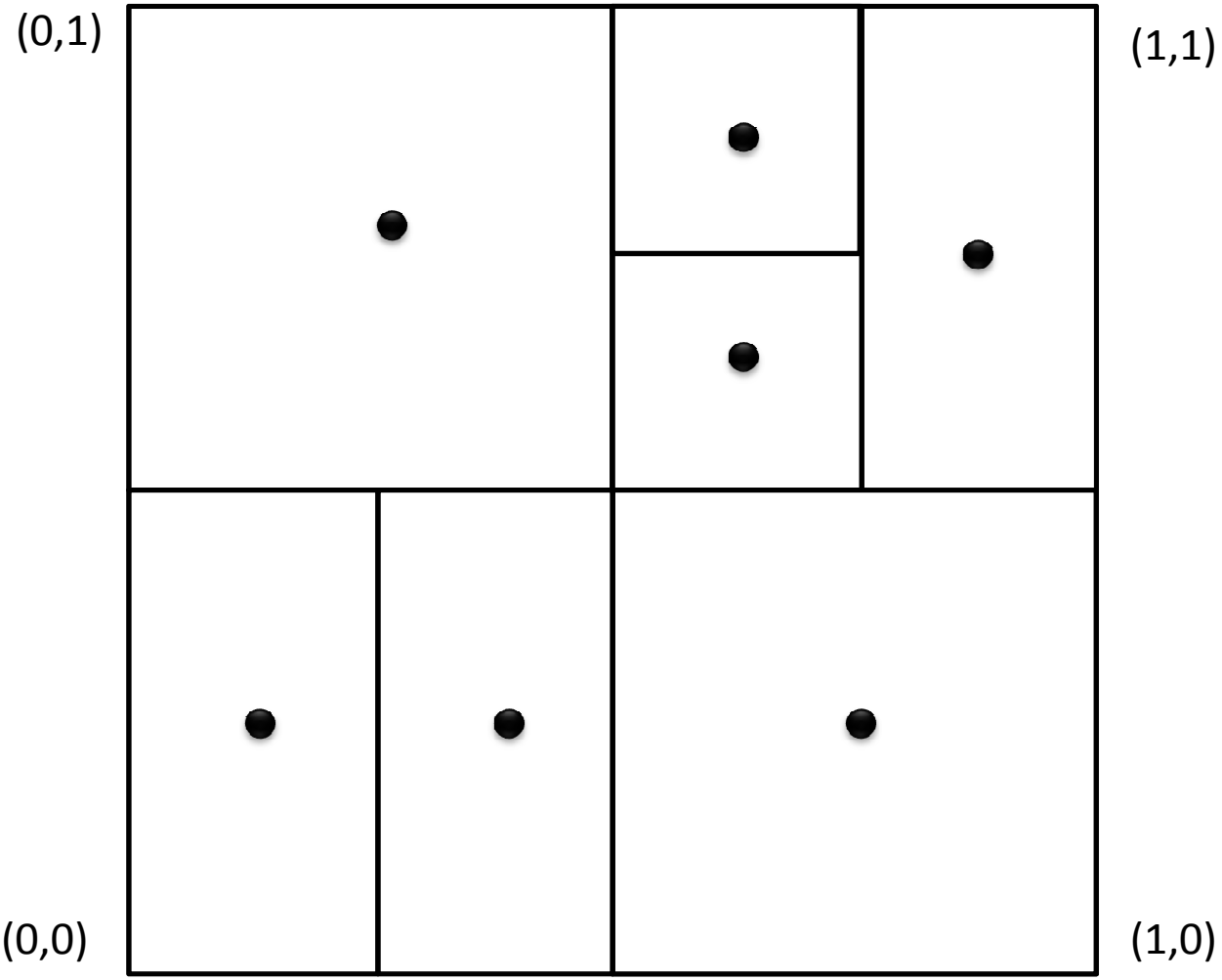
CAN



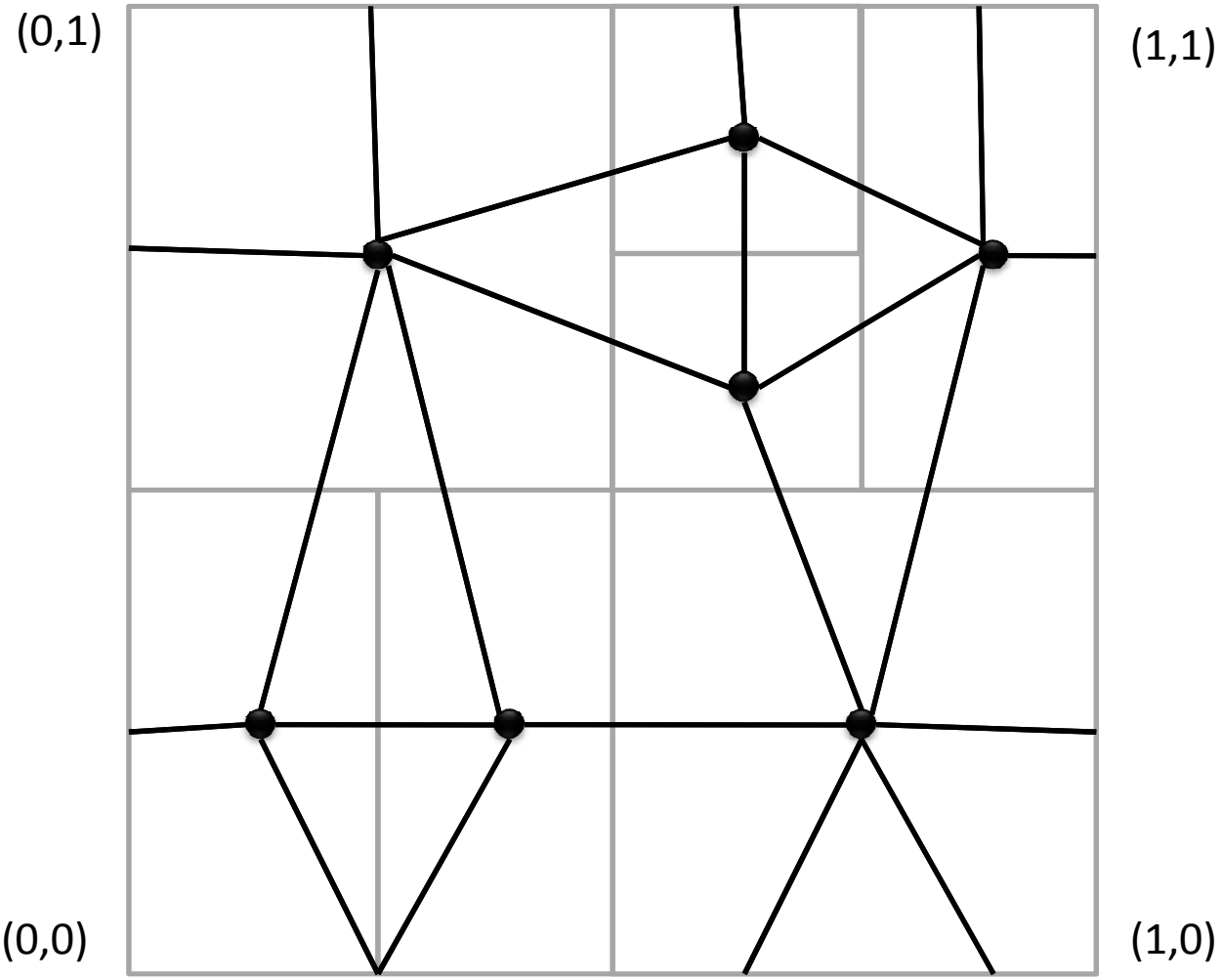
CAN



CAN



Verbindungsstruktur



CAN

- ID-Raum ist 2-Dimensional kontinuierlich [0,1)x[0,1)
- Routing:
  - Wenn Ziel in meine Zone fällt, bin ich fertig
  - Sonst: Schicke Nachricht zu *einem* Nachbarknoten, der eine kleinere (euklidische) Distanz zum Ziel hat
- Knotenankunft:
  - Wähle zufällige ID aus [0,1)x[0,1)
  - Route Nachricht an diesen Punkt
  - Gebiet des Ziel-Peers wird geteilt
  - Baue Verbindungen zu allen Nachbarn des alten Peers auf

Analyse

- Idealisiert: 2D-Gitter
- Nötige Routing-Schritte
  - O(SQRT(N))
- Anzahl Nachbarn
  - 4 Nachbarn, also O(1)
- Vergleiche mit Chord:
  - beides O(log N)
- ➔ Routing langsamer, aber viel weniger Verbindungen und damit weniger Verwaltungsaufwand!
- Trotzdem robustes Netzwerk

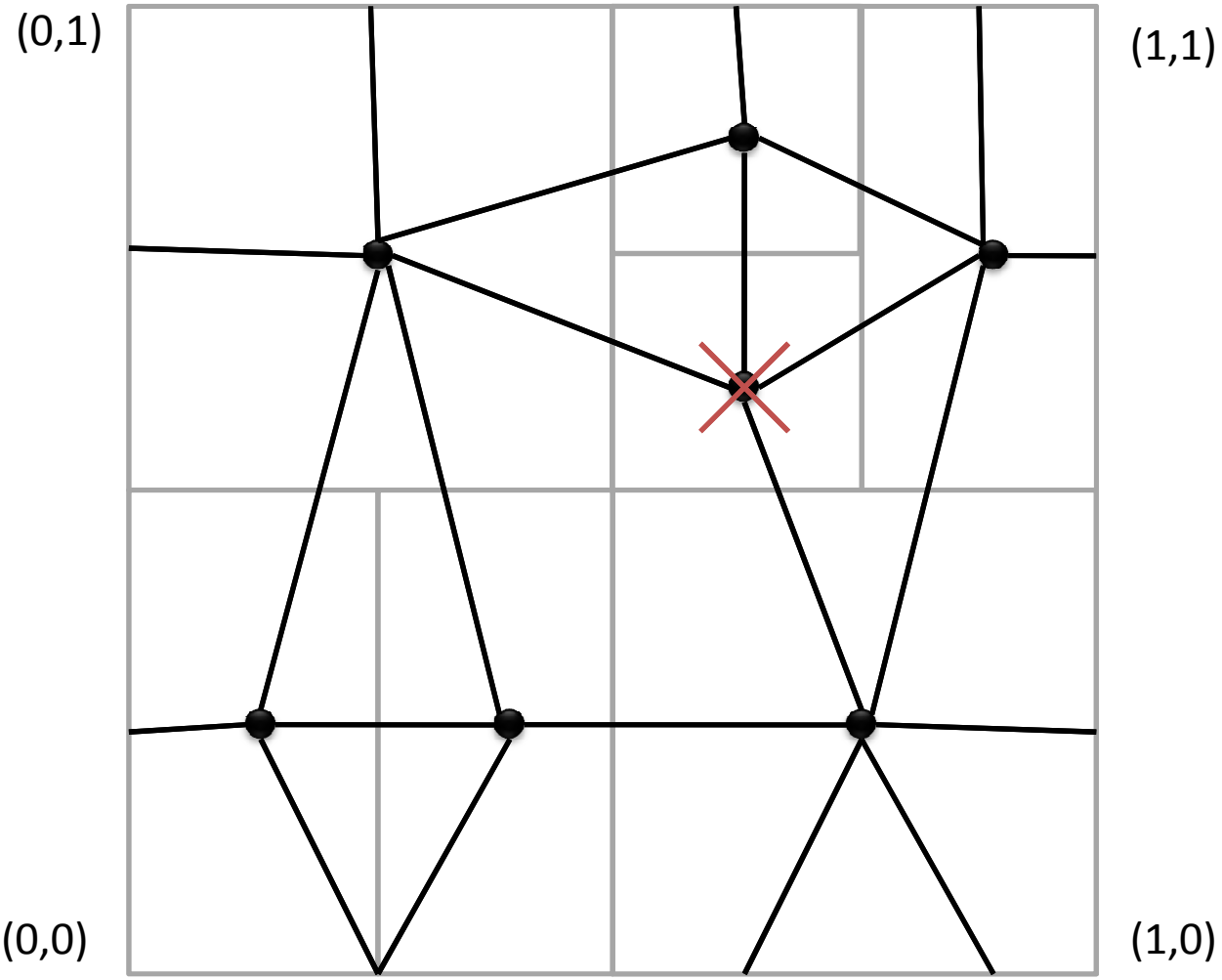
Wie groß werden die Zonen?

- ➔ Übungsaufgabe

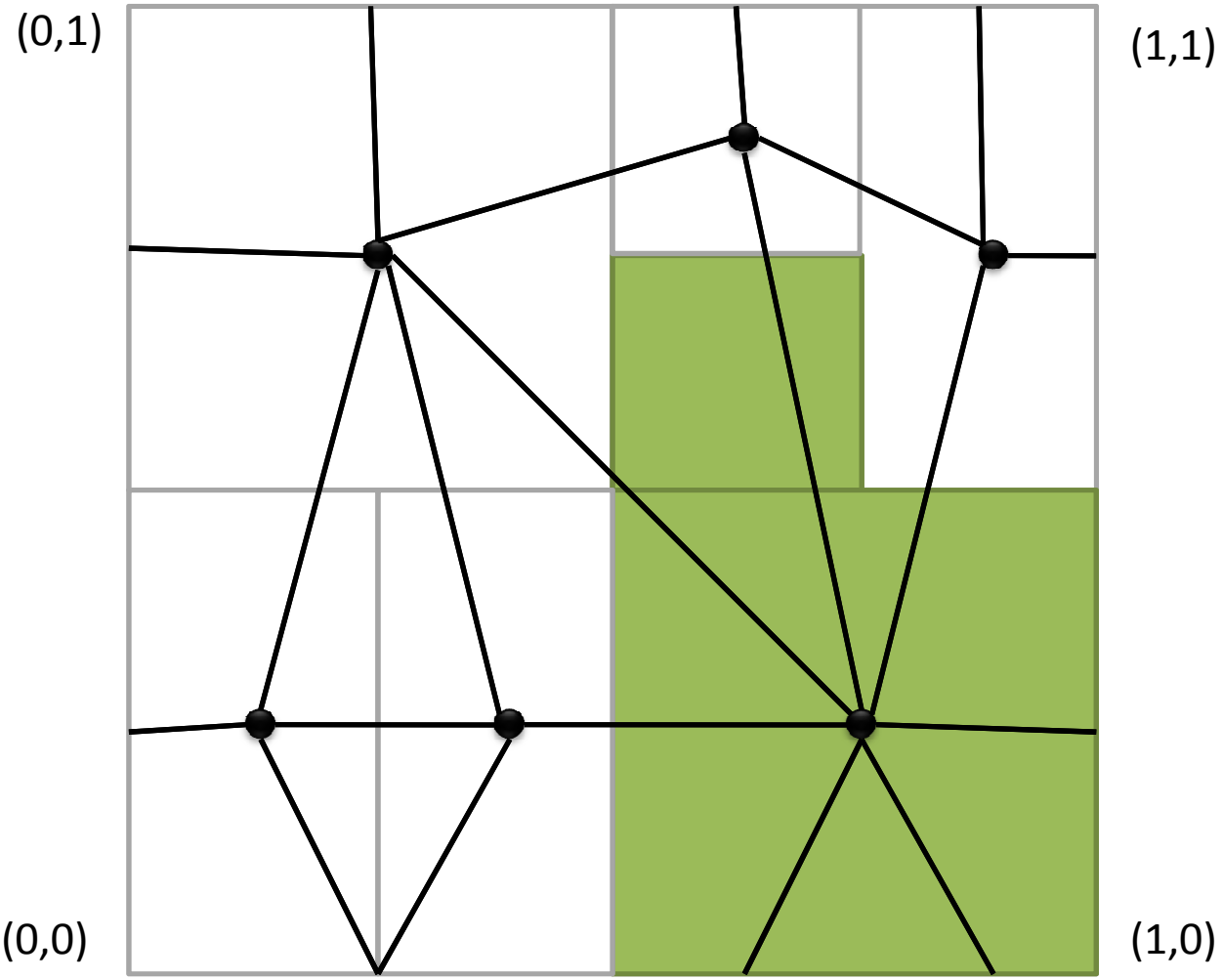
CAN: Ausfall eines Knotens

- Die Nachbarn entdecken, dass der Peer ausgefallen ist
- Ziel:
  - der Nachbar mit der kleinsten Zone soll den ausgefallenen Peer übernehmen
- Daher:
  - Jeder Knoten startet einen Timer proportional zur Größe der eigenen Zone
  - Wenn der Timer abgelaufen ist, sendet er eine Nachricht an alle anderen Nachbarn
  - Wer die Nachricht empfängt, stoppt seinen Timer
  - Danach übernimmt er die Zone des ausgefallenen Peers
- Kann zu **Fragmentierung** führen

Fragmentierung

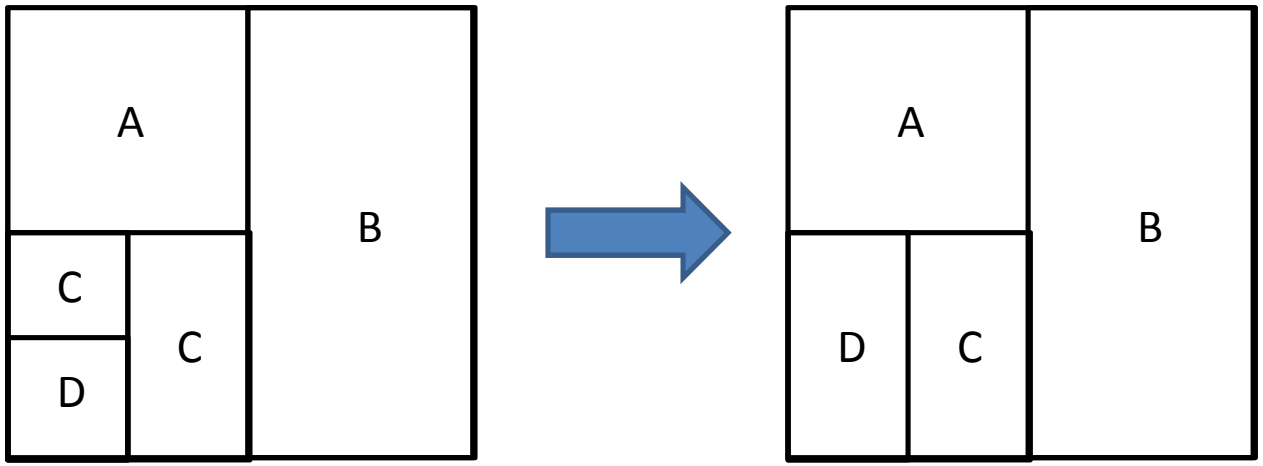


Fragmentierung



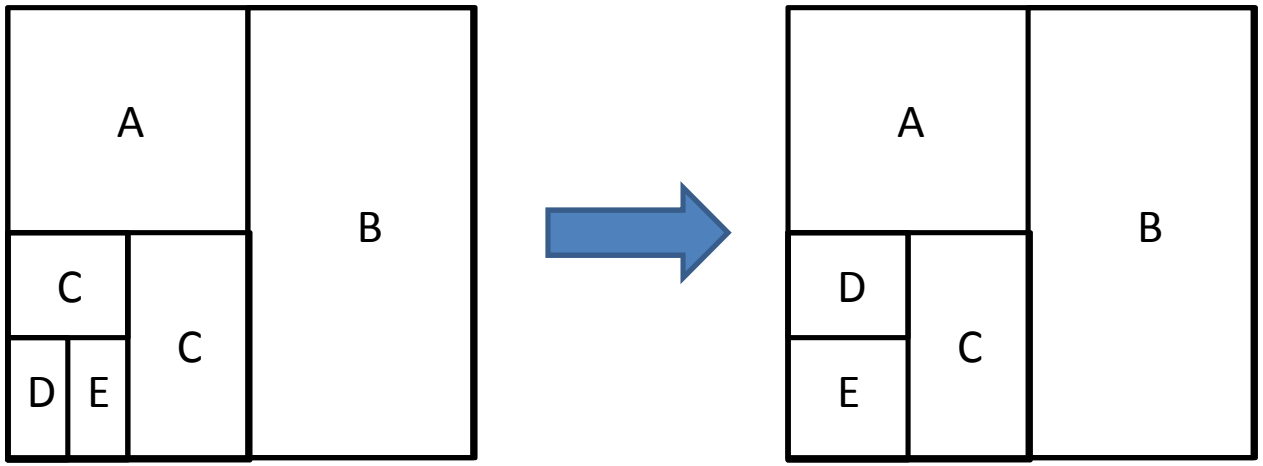
Defragmentierung Teil 1

- Wenn ein Peer mehr als eine Zone hat:
  - Versuche, kleinste Zone loszuwerden
- Wenn die Nachbarzone ungeteilt ist:
  - Übergib die Zone dem Nachbar



Defragmentierung Teil 2

- Wenn die Nachbarzone aufgeteilt ist:
  - Tiefensuche im Baum, bis zwei benachbarte Blätter gefunden wurden
  - Fasse diese Blätter zusammen
  - der freigewordene Peer übernimmt das Gebiet

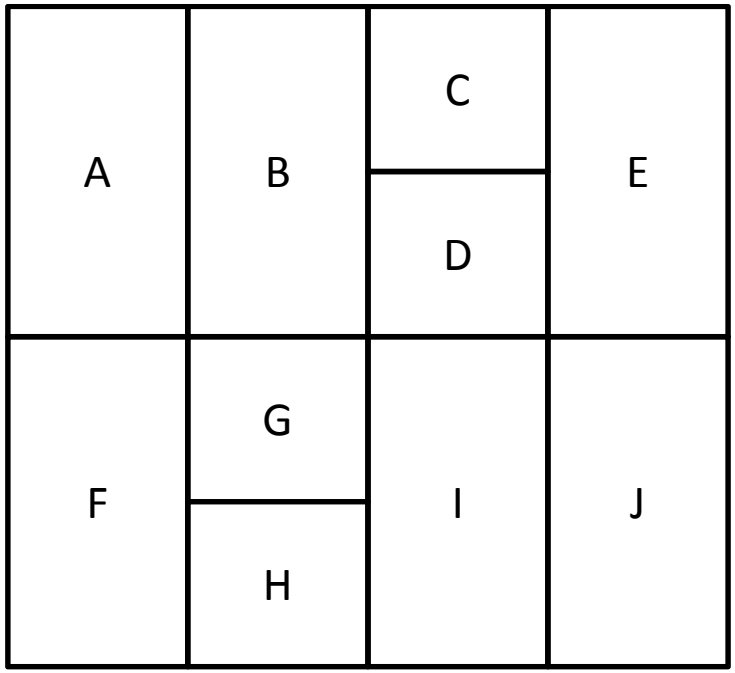


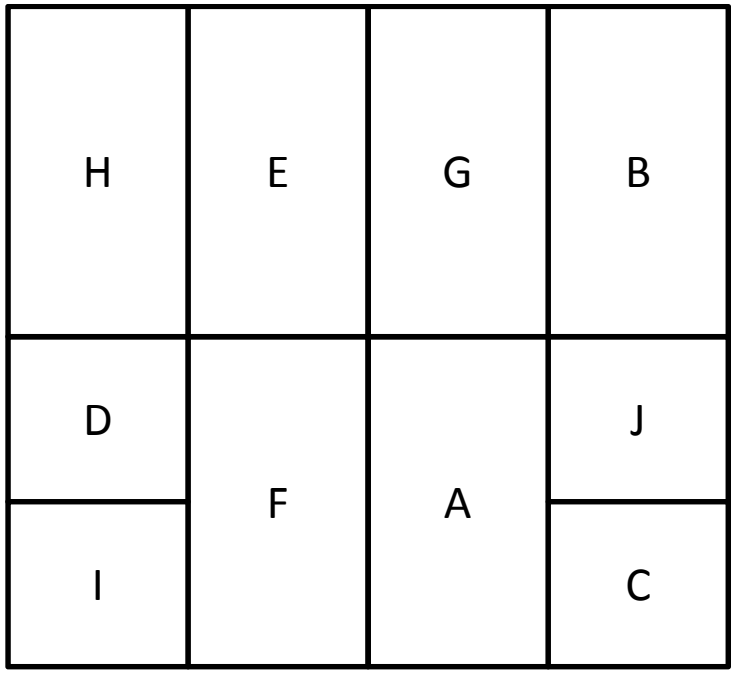
Mehrdimensionales CAN

- Analoge Konstruktion, aber d-Dimensionaler Schlüsselraum
- Bewirkt:
  - Kürzere Routing-Zeiten
  - Mehr Nachbarn
- Genauer:
  - $O(n^{1/d})$ Hops
  - 2d Nachbarn, also O(d) Nachbarn

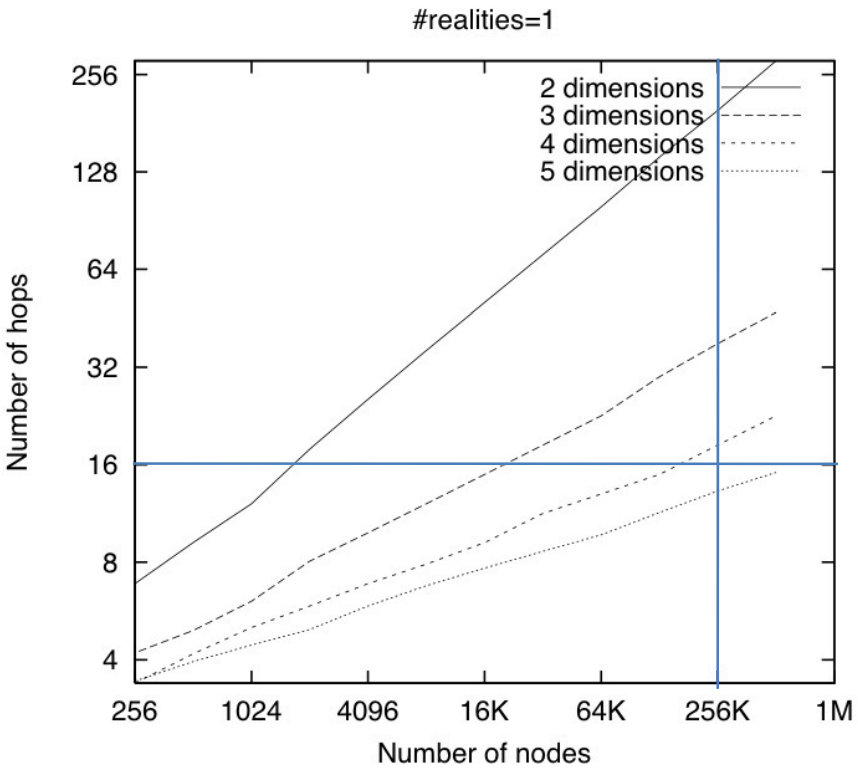
Mehrere Realitäten

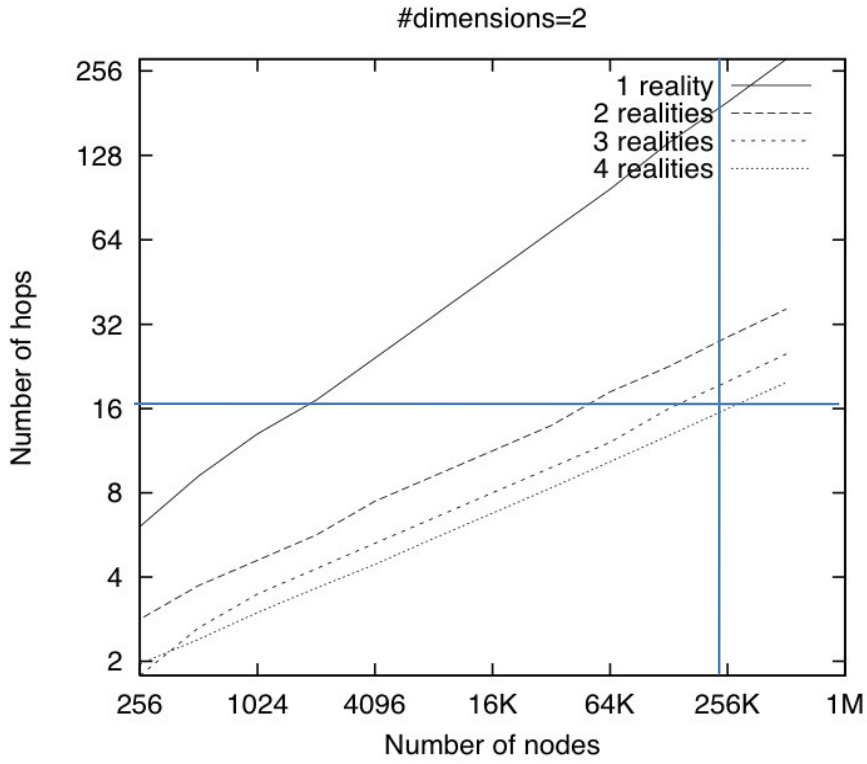
- Es werden gleichzeitig mehrere CAN Netzwerke aufgebaut
- Jedes heißt **Realiät**
- Beim Routing wird zwischen den Realitäten gesprungen
- Bewirkt: Schnelleres Routing, Replikation, Robustheit





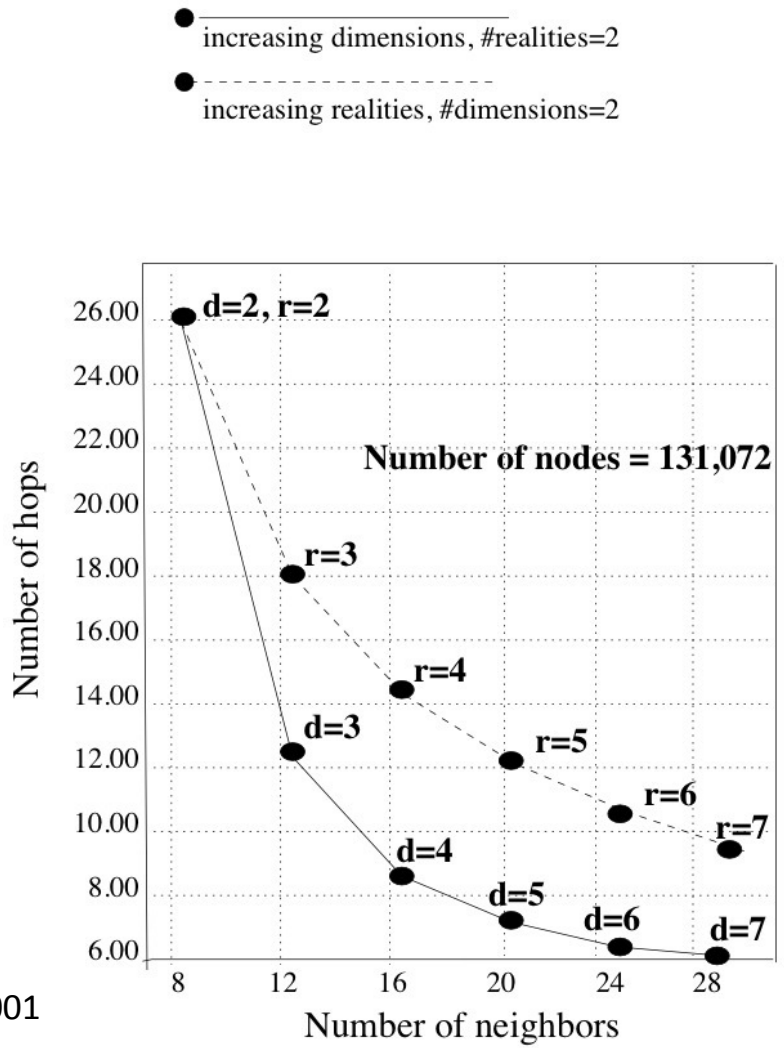
Dimensionen vs. Realitäten





Quelle: Ratnasamy, Francis, Handley, Karp, Shenker:
A Scalable Content-Addressable Network, SIGCOMM'01, 2001

Dimensionen vs. Realitäten



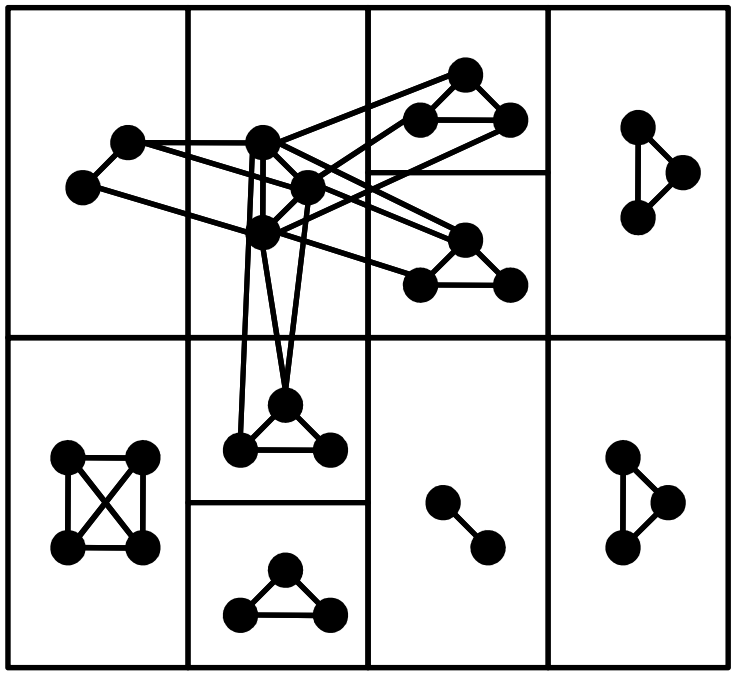
Quelle: Ratnasamy, Francis, Handley, Karp, Shenker:
A Scalable Content-Addressable Network, SIGCOMM'01, 2001

Mehrfaches Hashing

- Daten werden mehrfach abgespeichert
  - indem man k unterschiedliche Hash-Funktionen nimmt
- Dadurch erhöht sich die Robustheit
- Geringere Entfernungen
  - Lookup nur zu nächster Kopie

Überladen von Zonen

- In jeder Zone werden mehrere Peers platziert
- Alle Peers einer Zone kennen sich untereinander
- Jeder kennt mindestens **einen** aus der Nachbarschaft
- Bewirkt: Routing konstant schneller, Replikation

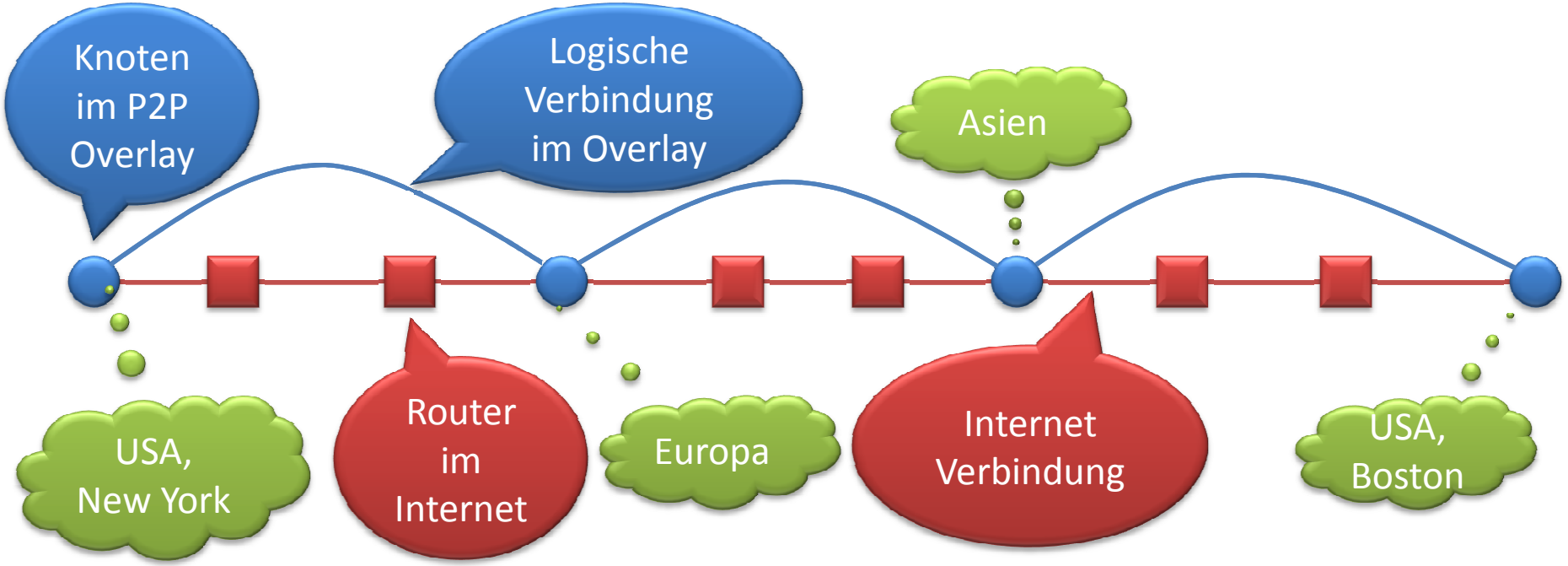


Uniform Partitioning

- Beim Join wird versucht, die Zonengröße besser auszugleichen
- Jeder Peer kennt die Zonen seiner Nachbarn
- Es wird die größte Zone geteilt

Latenzen Berücksichtigen

- Bisher haben wir Routing-Performance nur in Hop-Distanzen im Overlay-Netzwerk gemessen
- Wie sieht es tatsächlich aus?
- Beispiel: Routing mit 3 Hops im Overlay



Latenzen

- Die effektive Latenz im Routing hängt also von 2 Faktoren ab:
  - Anzahl Hops
  - Internet-Latenzen der einzelnen Hops
- Es kann also evtl. schneller sein, mehr Hops zu machen, die aber dafür lokaler sind
- Dazu muss das Routing oder die Verbindungsstruktur eine gewisse Flexibilität haben
- Bei Chord z.B. sind Struktur und Routen determiniert
- CAN ist hier flexibler

Latenzen

- In CAN 3 Ansätze:
  - Im Standard-CAN den nächsten Hop mit bestem Verhältnis ID-Distanz zu Latenz auswählen (statt maximaler ID-Distanz)
  - Beim Überladen von Zonen Verbindung mit dem "dichtesten" Knoten in der Nachbarzone eingehen
    ➔ regelmäßig die anderen Kandidaten prüfen, evtl. wechseln
  - ID nicht zufällig wählen, sondern anhand einer Lokalitäts-Metrik Position im Netzwerk wählen

Netzwerk an Internet-Topologie anpassen

- m spezielle Peers dienen als "Landmarken"
- Latenzzeiten zu diesen Landmarken werden gemessen
- Liste der Latenzzeiten zu den Landmarken wird sortiert
- Diese Sortierung bestimmt Position im CAN
- Dadurch werden nahe Peers auch in CAN nah einsortiert
- Vorteile:
  - Gute Verringerung der Latenzzeiten
- Probleme
  - Wie wähle ich die Landmarken aus?
  - Lastungleichgewichte
  - Gefahr von Partitionierung

CAN: Gesamt-Evaluation

Parameter	bare bones CAN	knobs on full CAN
Dim.	2	10
Realitäten	1	1
Peer / Zone	0	4
Hashfkt.	1	1
Latenz-Optimiertes Routing	Aus	An
Uniform Partitioning	Aus	An
Landmark ordering	Aus	Aus

Metrik	bare bones CAN	knobs on full CAN
Pfadlänge	198	5
Grad	4,57	27,1
Peers	0	2,95
IP Latenz	115,9 ms	82,4 ms
Pfad Latenz	23,008 **sec.**	135,29 ms

CAN: Vorteile/Nachteile

- Vorteile
  - Einfaches Verfahren
  - Balanciert die Datenmenge
  - Kleiner Grad
  - Netzwerk ist stark zusammenhängend, dadurch robust
  - Kennt verschiedene Wege zum Ziel und kann dadurch Routen optimieren
- Nachteile
  - Lange Wege (polynomiell lang)
  - Stabilität durch geringe Nachbarzahl gefährdet

Vorlesung
P2P Netzwerke
5: Pastry und Tapestry

Dr. Felix Heine
Complex and Distributed IT-Systems
felix.heine@tu-berlin.de

Pastry und Tapestry

- Pastry
  - Antony Rowstron and Peter Druschel: "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems", IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001
- Tapestry
  - Kirsten Hildrum, John Kubiatowicz, Satish Rao, Ben Y. Zhao, "Distributed object location in a dynamic network.", SPAA, 2002, pp. 41-52
- Grundlage: Routing nach Plaxton/Rajaraman/Richa
  - C. Greg Plaxton, Rajmohan Rajaraman, A. W. Richa, "Accessing Nearby Copies of Replicated Objects in a Distributed Environment", SPAA, 1997, pp. 311-320

Grundidee: Präfix-Routing

- ID wird zur Basis 2^b dargestellt,
  - z.B. b=4 bedeutet Hexadezimaldarstellung
  - bei 160 Bit sind das 40 Ziffern
- Grundidee:

> Sende die Nachricht an den Knoten aus der Routing-Tabelle, dessen gemeinsamer Präfix mit der Ziel-ID maximale Länge hat

- Was für Einträge braucht man in der Routing-Tabelle?

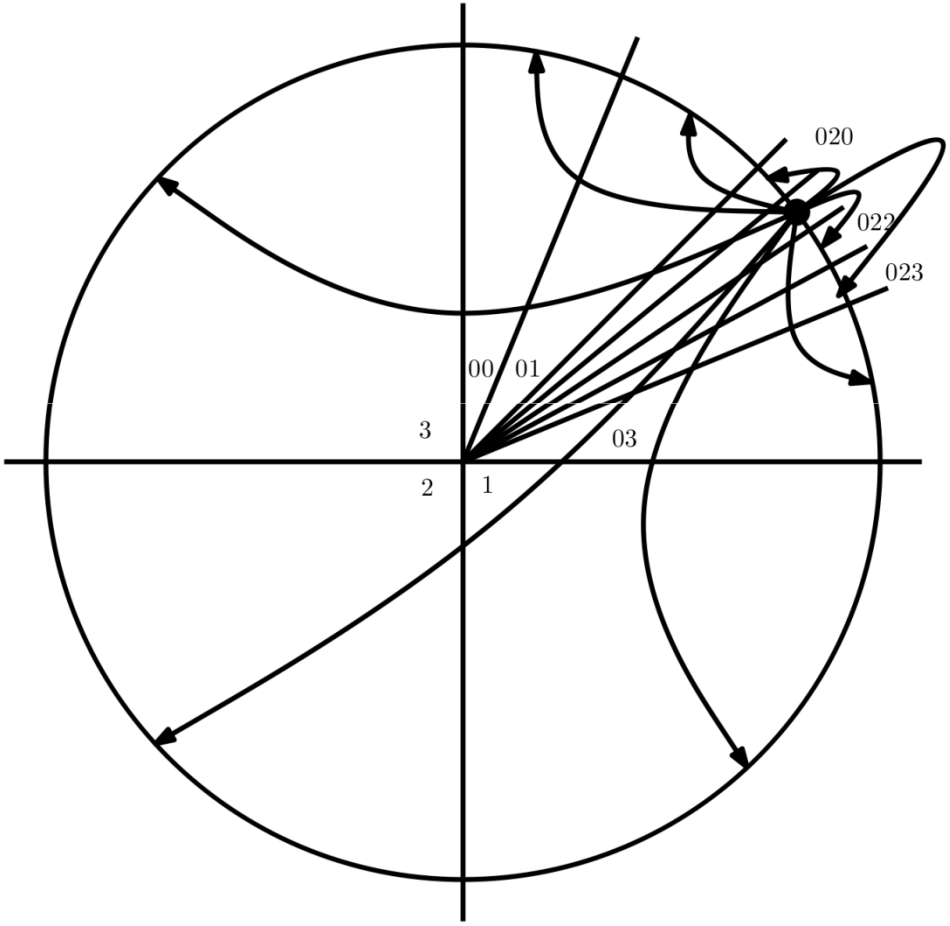
Routing Tabelle

Knoten 53411721, 10.000 Knoten
b=3, d.h. Oktalziffern
ID-Länge 8 Ziffern, d.h. 3*8 = 24 Bit

Präfix	0	1	2	3	4	5	6	7
-	05113420	10421650	25661215	33561253	42604240		66353254	72621233
5	50652006	51615413	52211242		54116142	55103005	56433314	57767105
53	53021530	53110517	53242072	53351527		53557047	53654314	53740336
534	53403564			53435054	53447620	53452243	53467746	53472422
5341	53410425							
53411								
534117								
5341172								

Routing-Tabelle

- **Beispiel:** Graphische Darstellung eines Teils der Routing-Tabelle eines Knotens mit der ID 021xxxx
- Jeweils ein Zeiger in den Sektor 1xxx, 2xxx, 3xxx
- Dann je ein Zeiger in die Sektoren 00xxx, 01xxx, 03xxx
- Dann je ein Zeiger in die Sektoren 020xxx, 022xxx, 023xxx
- etc.



Probleme

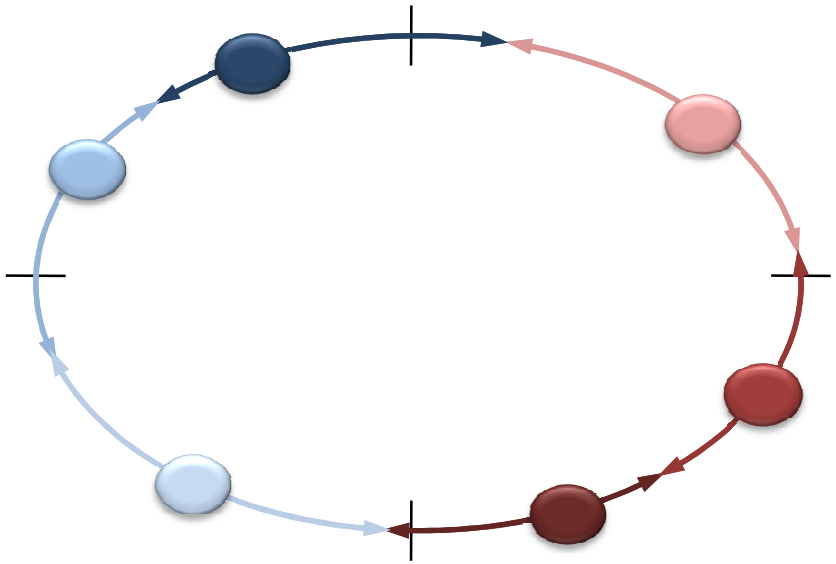
- Wie geht das Routing weiter, wenn man auf einen leeren Eintrag in der Routing-Tabelle trifft?
- Hier unterscheiden sich Pastry und Tapestry
  - Frage: Sind die Routing-Tabellen konsistent?
  - d.h., gibt es für einen leeren Eintrag niemals einen passenden Knoten?
  - Antwort von Pastry: Nicht konsistent
  - Antwort von Tapestry: Konsistent
- Entsprechend dieser Annahme unterscheidet sich das weitere Vorgehen

Pastry

- Zuordnung von Bereichen:
  - Root-Knoten ist der numerisch Dichteste
  - Aufpassen: der numerisch Dichteste kann ein ganz anderes Präfix haben!
  - Beispiel: Knoten 2700, Ziel 2777, nächster Knoten 3000 (Oktal)
- Daher:
  - Pastry wechselt am Ende auf ein anderes Routing-Verfahren
  - Dazu wird ein Leaf-Set verwendet, das die L dichtesten Knoten beinhaltet
- Also:
  - Mit Präfix-Routing wird in die Nähe des Ziels gesprungen
  - Die genaue Positionierung geht dann über das Leaf-Set

Vergleich mit Chord

- In beide Richtungen routen (statt nur im Uhrzeigersinn)
- Restdistanz pro Schritt durch 2^b teilen (statt halbieren)
- Routing-Einträge werden jeweils aus einem Intervall ausgesucht (statt deterministisch einen zu nehmen)
- Verantwortlichkeiten werden anders definiert



Pastry: Routing Algorithmus

```
1    // Knoten A: Route Schlüssel D
2    if LeafSet[-L/2] <= D <= LeafSet[L/2] // im Leafset?
3       weiterleiten an LeafSet[i] s.d. |D-LeafSet[i]| minimal
4    else
5       l = Länge des gemeins. Präfixes von A und D
6       if R[l, D[l]] <> nil
7          weiterleiten an R[l, D[l]]
8       else
9          // seltener Fall
10         suche Knoten T in R und L so dass
11              - Länge des gemeins. Präfixes von T und D >= l
12              - |T-D| < |A-D|
13         weiterleiten and T
```

Pastry: Performance

- Anzahl Einträge in der Routing-Tabelle: $O(\log_{2^b} N)$
- Routing-Performance: $O(\log_{2^b} N)$
- Wenn Routing-Tabellen leer: $O(N / L)$

Lokalität

- Einträge in der Routing-Tabelle können frei gewählt werden (entsprechend den Constraints)
- Tabellen werden mit "nahen" Knoten befüllt
- Art der Distanzbestimmung ist dabei offen gelassen
- Anforderung: es muss eine Metrik sein
- Die Befüllung der Routing-Tabellen mit "nahen" Knoten wird durch den Join-Algorithmus gewährleistet:
  - Annahme dabei: neuer Knoten ist "nahe" zu seinem Join-Knoten A
  - Also sind die Knoten B, C, ... auf dem Pfad auch "nahe"
  - Daher ist die so konstruierte Tabelle schon relativ gut

Ankunft eines Knoten

- Route Nachricht an eigene ID *id* über bekannten Knoten *A*
- Die Nachricht wird über Knoten *B, C, …* an Knoten *Z* geroutet
- Jeder Knoten auf der Route schickt seine Routing-Tabelle zurück
- Daraus baue ich meine eigene Tabelle auf:
  - Zeile 0 wird von Knoten A genommen
  - Zeile 1 wird von Knoten B genommen
  - …
- Leaf-Set wird von Knoten Z genommen
- Alle jetzt bekannten Knoten werden informiert

Varianten beim Einfügen

- Einfache Variante SL (von vorheriger Folie):
  - Jeweils eine Zeile auf dem Weg vom Join-Knoten zum Ziel nehmen
- Variante WT:
  - Gesamten Status von jedem Knoten auf dem Weg zum Ziel nehmen
- Variante WTF (in FreePastry implementiert):
  - Eine Ebene weiter gehen, von allen so bekannten Knoten den Status holen
- In den Varianten WT und WTF wird jeweils die Lokalität geprüft (RTT-Messungen)
- Bei mehreren Möglichkeiten, eine Zelle zu füllen, wird jeweils der dichteste Knoten genommen

Probleme beim Einfügen

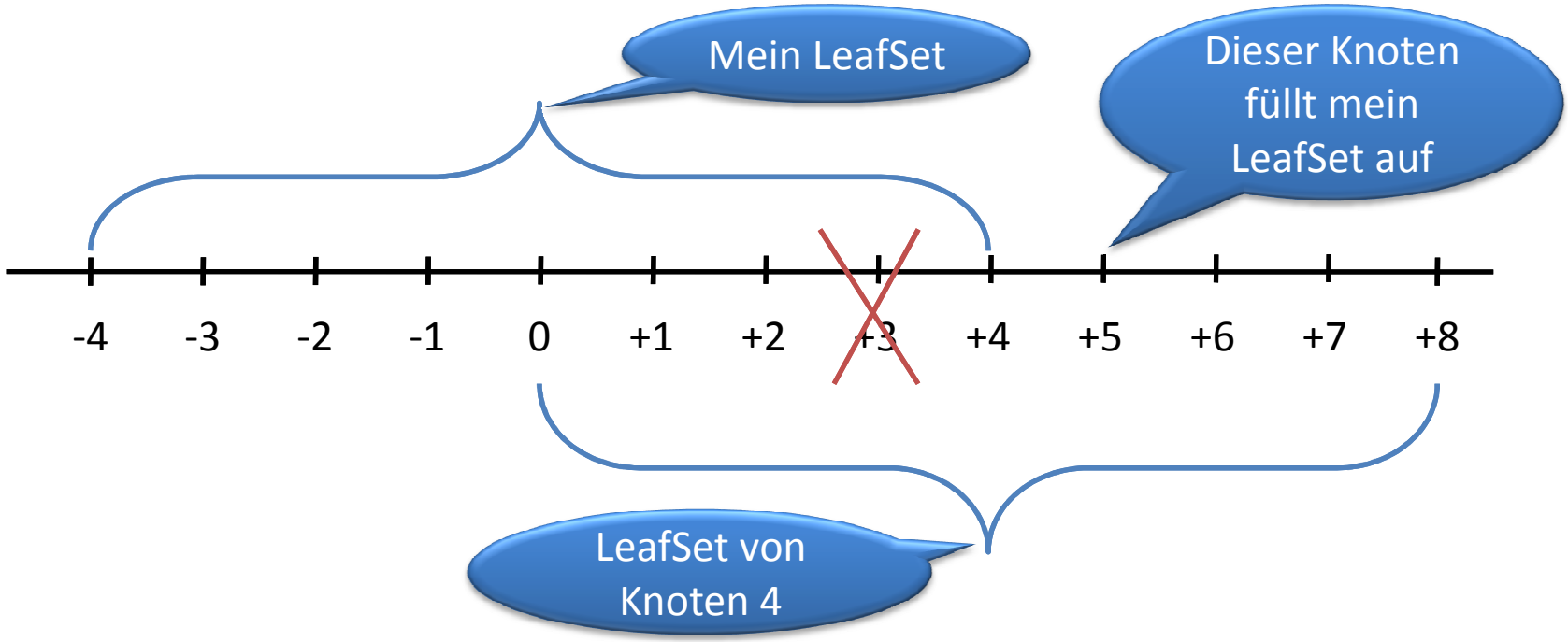
- Ich bin der erste Peer mit einem bestimmten Präfix
- Dann müssen potentiell alle Knoten im Netz mich kennenlernen
- Es lernen mich aber nur Teile des Netzwerkes kennen
- Der Rest wird über den Reparaturmechanismus erledigt

Ausfall eines Knoten

- Auffüllen der Routing-Tabelle:
- Um einen Eintrag R[l, d] zu ersetzen, wird zunächst ein Knoten R[l, i], i != d, gefragt. Wenn dieser einen passenden Eintrag hat, wird er genommen.
- Wenn nicht, wird ein Knoten R[l+1, i], i != d, gefragt, etc.
- Mit hoher Wahrscheinlichkeit wird so ein Knoten gefunden, der die Tabelle auffüllen kann

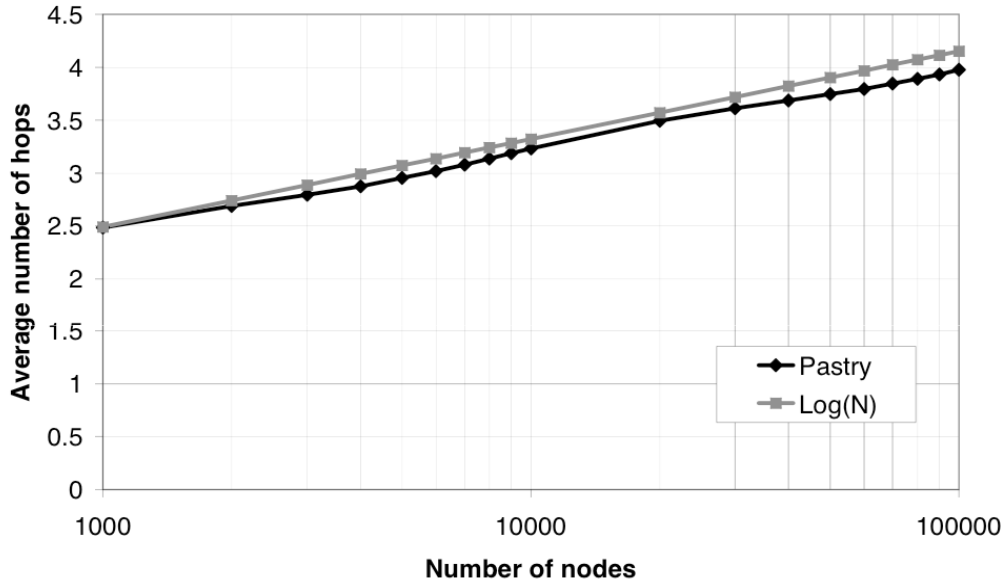
Ausfall eines Knotens

- Wenn ein toter Knoten im LeafSet bemerkt wird:
- Kontaktiere den äußersten lebendigen Knoten im LeafSet, um das LeafSet wieder aufzufüllen



Messungen

- Hop-Anzahl:

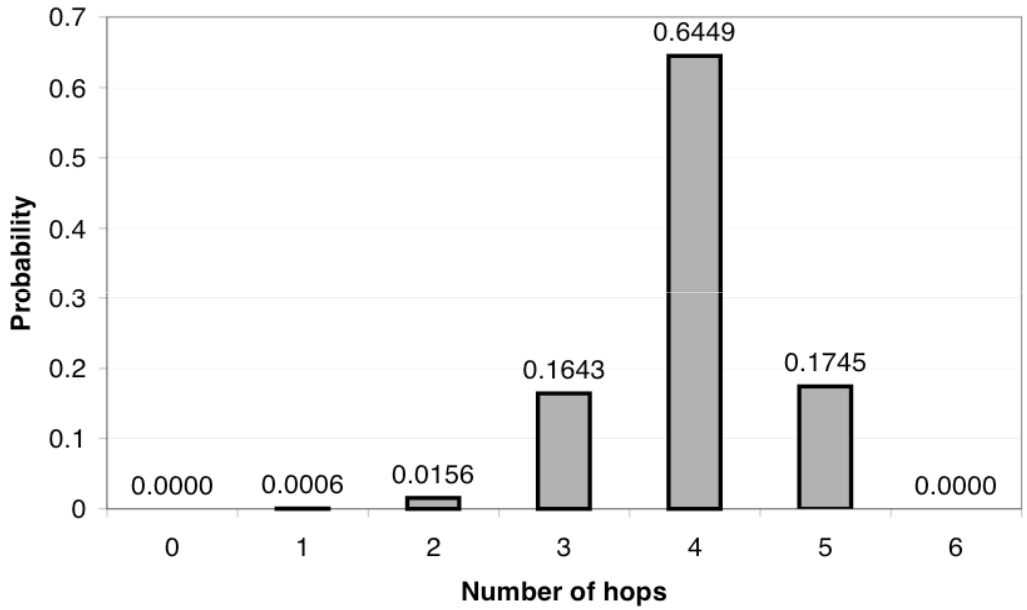


- Parameter b=4, L=16
- also log(N) = $\log_{16}(N)$
- Gute Übereinstimmung Analyse / Messungen

Quelle: Antony Rowstron and Peter Druschel:
Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems

Messungen

- Abweichung von der Hop-Distanz:

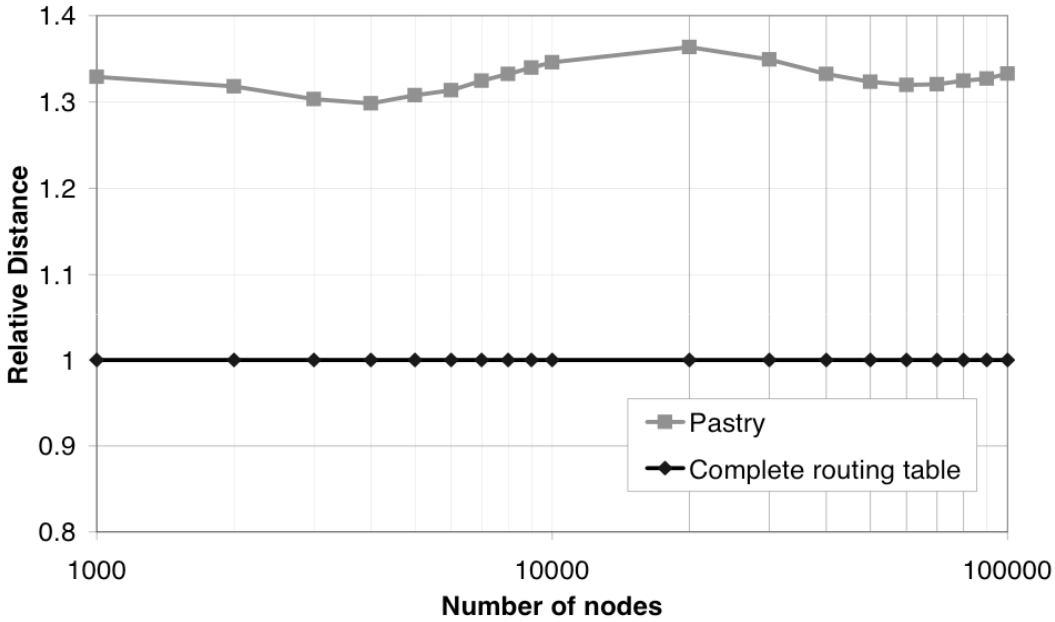


Quelle: Antony Rowstron and Peter Druschel:
Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems

- Geringe Abweichung von erwarteter Hop-Distanz

Messungen

- Messung des sog. **Stretch**: Verhältnis zum optimalen Routing



Quelle: Antony Rowstron and Peter Druschel:
Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems

- Latenz nur ca. 40% langsamer als direktes Internet-Routing

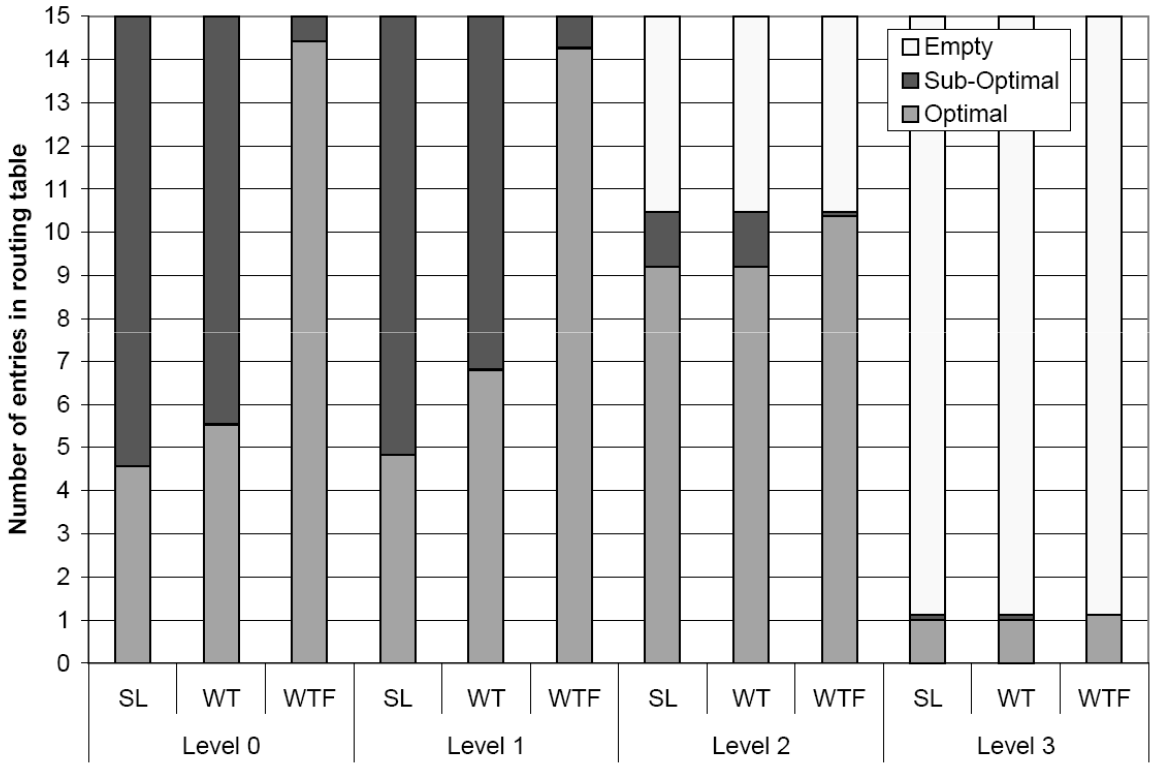
Routing Geschwindigkeit

- Für die Reihe 0 gibt es pro Feld $N/2^b$ Möglichkeiten
- D.h. die Wahrscheinlichkeit, einen nahen Knoten zu finden, ist hoch
- Für Reihe 1 gibt es pro Feld $N/2^{2b}$ Möglichkeiten
- Für Reihe 2 gibt es pro Feld $N/2^{3b}$ Möglichkeiten
- …
- Für den Nachbarn im LeafSet gibt es nur eine Möglichkeit → sehr weit entfernt
- Daraus folgt:
  - Die ersten Schritte sind billig
  - Die letzten Schritte sind teuer

Mehr Schritte sind nicht viel teurer, da die teuren Schritte immer anfallen!

Messungen

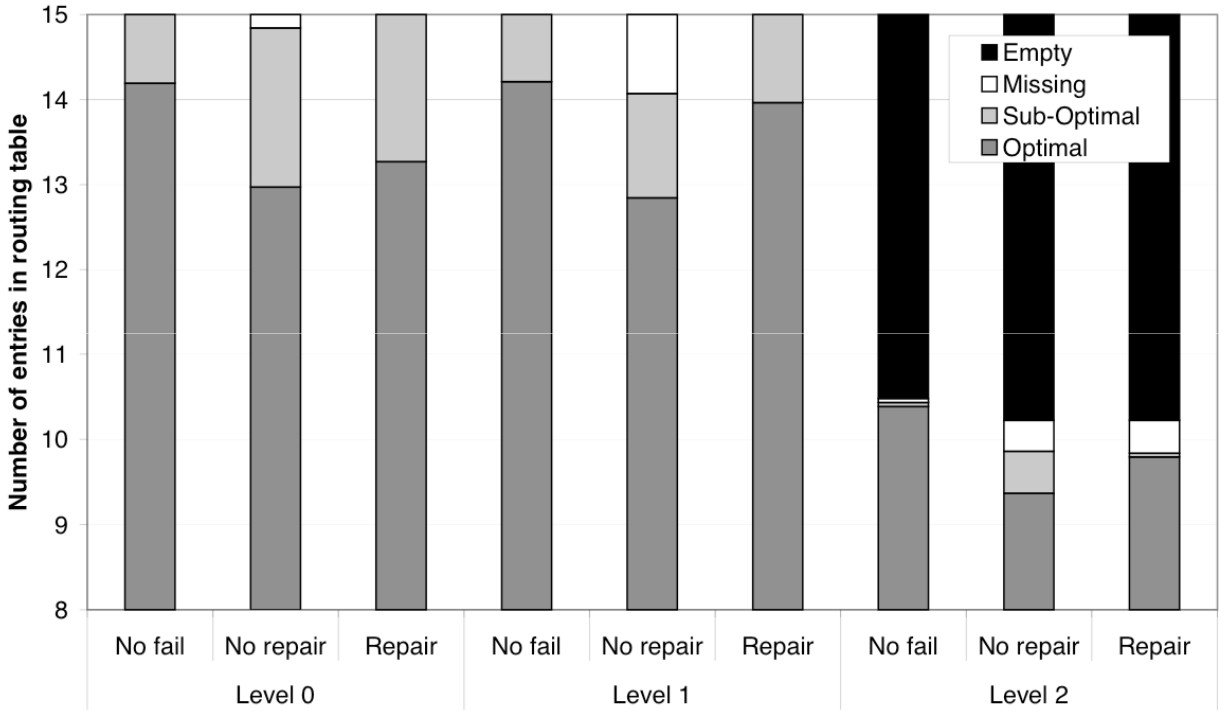
- Qualität der Routing-Tabellen



Quelle: Antony Rowstron and Peter Druschel:
Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems

Messungen

- Knotenausfall: von 5000 Knoten fallen 500 aus



Quelle: Antony Rowstron and Peter Druschel:
Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems

Tapestry

- Routing-Tabellen wie bei Pastry aufgebaut
  - Aber: pro Eintrag werden mehrere Knoten gespeichert
    - Jeweils die k *nächsten* Knoten, die auf einen Eintrag passen (primärer Nachbar / sekundäre Nachbarn)
  - Es gibt kein Leaf-Set
- Tapestry hat **andere Herangehensweise**
  - Die Routing-Tabelle ist nicht Hilfsmittel zur Beschleunigung, sondern elementar für korrektes Routing
  - Tapestry geht daher von konsistenten Routing-Tabellen aus.
  - Die Konsistenz muss immer sichergestellt werden

Tapestry

- Genauer werden folgende Eigenschaften verlangt:

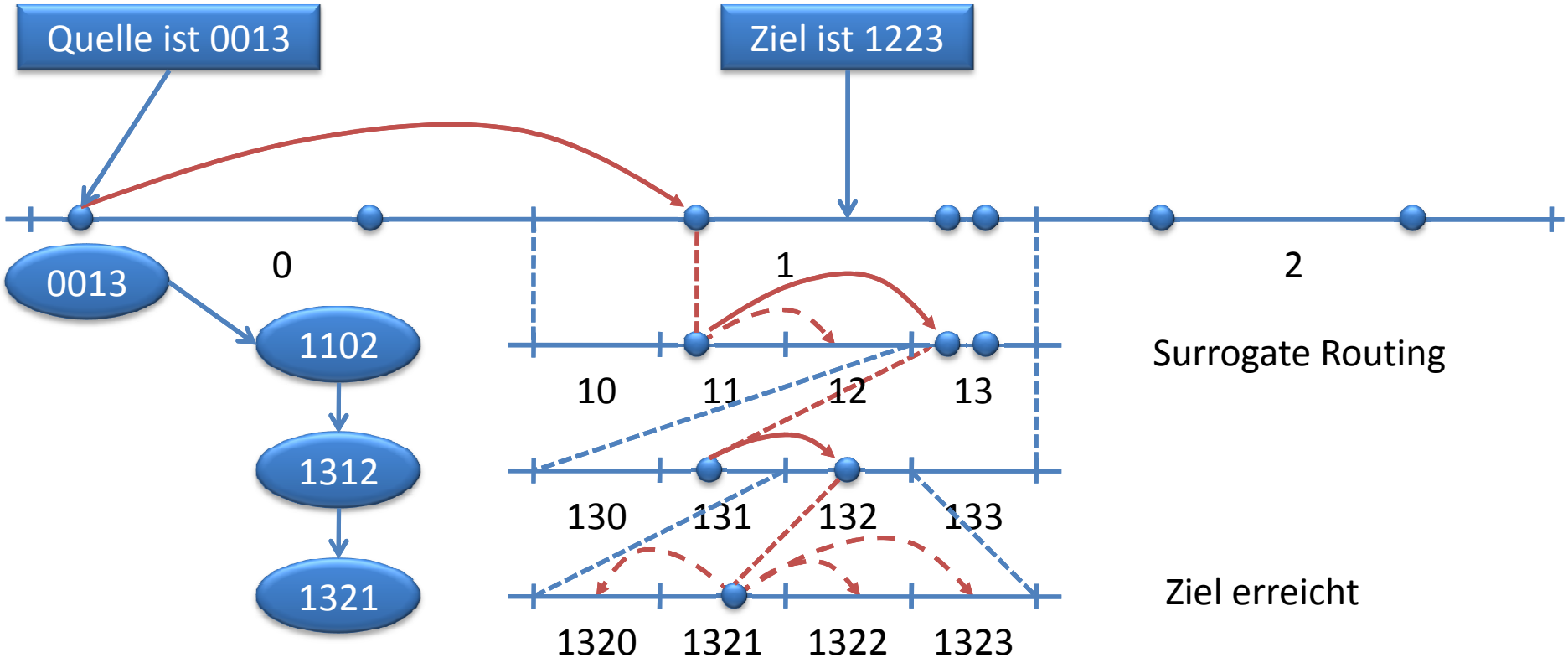
- **Eigenschaft 1: (Konsistenz)**
  - Wenn ein Eintrag in einer Routing-Tabelle leer ist, gibt es keinen passenden Knoten im Netzwerk

- **Eigenschaft 2: (Lokalität)**
  - Jeder Eintrag in einer Routing-Tabelle enthält die dichtesten Nachbarn bzgl. einer gegebenen Metrik. Der dichteste Nachbar ist der primäre Nachbar, die anderen die sekundären Nachbarn

Surrogate Routing

- Kein 2-Stufiges Routing mehr nötig
- Sondern sog. Surrogate-Routing:
  - In jedem Routing-Schritt wird eine Stelle angepasst
  - Wenn ein Eintrag in einer Routing-Tabelle leer ist, wird zu dem nächten Eintrag in der gleichen Zeile geroutet (mod 2^b)
  - Wenn alle Einträge in der aktuellen Zeile leer sind, bin ich der Empfänger
- Dadurch gibt es zu jedem Ziel einen eindeutigen Surrogate-Root Knoten
- **Funktioniert nur, wenn Eigenschaft 1 gilt!**

Surrogate Routing

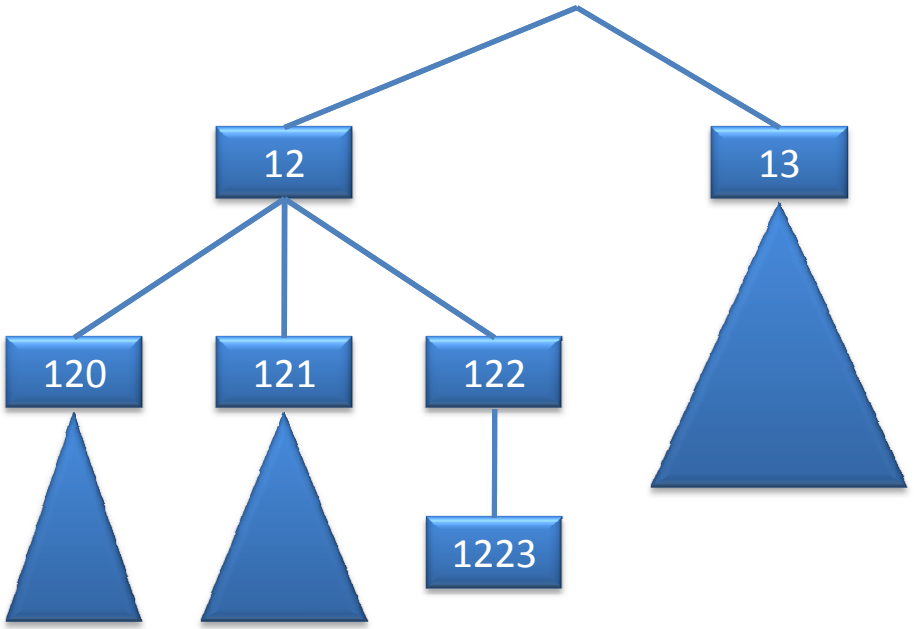


Knotenankunft

- Route Nachricht an Surrogate-Peer
- Baue vorläufige Routing-Tabelle durch Kopieren vom Surrogate auf
- Benachrichtige alle Knoten, die evtl. Löcher in der Routing-Tabelle haben
  - Hiermit wird Eigenschaft 1 hergestellt
  - Wird durch Multicast gelöst
- Baue schrittweise Nachbarschaftsmengen auf
  - Hiermit wird Eigenschaft 2 hergestellt

Acknowledged Multicast

- Jeder Knoten verschickt eine Nachricht an alle Kinder in seiner Routing-Tabelle
- Wenn er die Antworten bekommen hat, antwortet er an den Eltern-Knoten
- Damit wird sichergestellt, dass alle Knoten die Nachricht bekommen haben
- Einstiegspunkt ist der Knoten, der zuerst Surrogate-Routing durchgeführt hat



Aufbau der Nachbarschaftsmengen

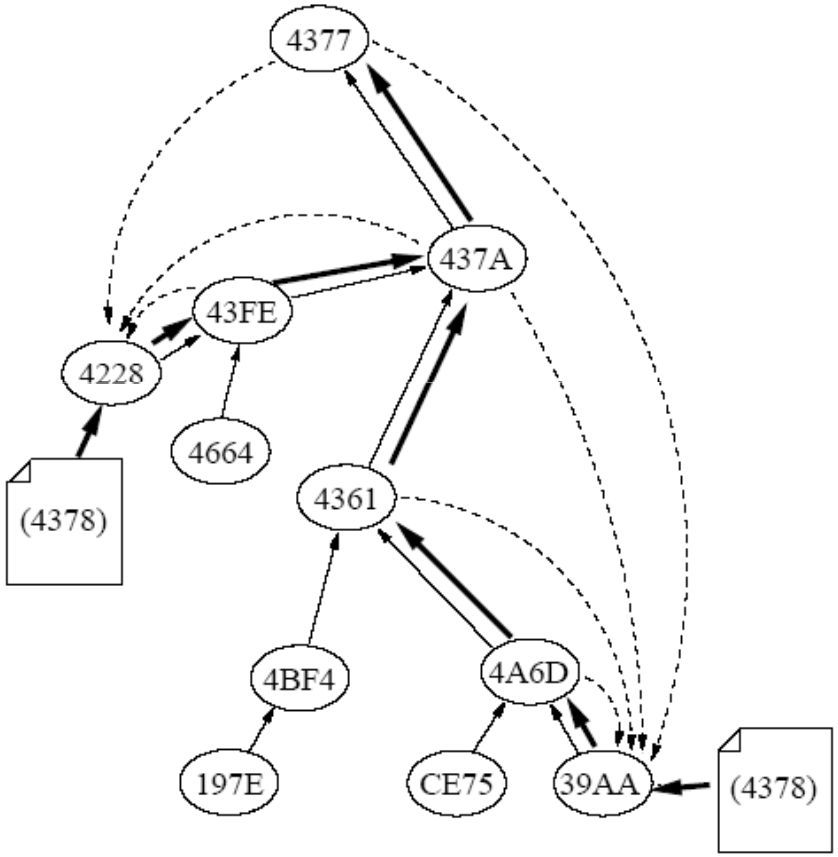
- Aufbau der Routing-Tabelle geht von unten nach oben
- Startpunkt ist die unterste Ebene, d.h. alle Knoten, die mit dem neuen Knoten das längstmögliche Prefix teilen
  - Diese wurden über Acknowledged Multicast gefunden
- Diese werden jeweils nach ihren Nachbarn eine Ebene darüber gefragt
  - Es werden beide Richtungen berücksichtigt (Forward und Backward)
  - Alle Peers werden kontaktiert, um Entfernung festzustellen
- Davon werden jeweils die k dichtesten Knoten genommen
- Dies wird bis zum obersten Level der Tabelle durchgeführt

Ausfall eines Peers

- Ein ausgefallener Peer wird aus der Routing-Tabelle entfernt
- Dabei kann ein Loch in der Routing-Tabelle entstehen, welches Eigenschaft 1 verletzt
  - Dieses wird wiederum per Multicast aufgefüllt bzw. bestätigt
- Außerdem müssen die sekundären Nachbarn aufgefüllt werden
  - Lokale Suche garantiert nicht Eigenschaft 2 (siehe Pastry)
  - Wiederholung des Algorithmus vom Insert

Objekt-Publikation in Tapestry

- Verweise auf Objekte werden auf dem Pfad zum Root-Knoten abgelegt
- Es können auch mehrere Kopien des Objektes im Netz verteilt sein
- Evtl. kann dadurch der Routing-Pfad abgekürzt werden



Quelle: Kirsten Hildrum, John Kubiatowicz, Satish Rao, Ben Y. Zhao, "Distributed object location in a dynamic network."

Vergleich Tapestry/Pastry

- Basieren beide auf Prefix-Routing
- Pastry
  - "Lockeres Verfahren": Routing-Tabellen müssen nicht korrekt sein
  - Leaf-Set reicht zum korrekten Routing
  - Tabellen nur zur Beschleunigung
  - In der Praxis besser einsetzbar
- Tapestry
  - Routing-Tabellen werden immer korrekt gehalten
  - Dadurch beweisbare Eigenschaften
  - Daher theoretisch interessant

Vorlesung
P2P Netzwerke
6: Gradoptimierte Netzwerke

Dr. Felix Heine
Complex and Distributed IT-Systems
felix.heine@tu-berlin.de

Inhalt

- Gradoptimierte Netzwerke (Überblick)
  - Viceroy
  - Distance Halving
  - Koorde

Grad vs. Durchmesser

- Bisher meist O(log N) Grad und O(log N) Durchmesser
- Geht der Grad kleiner, ohne den Durchmesser zu erhöhen?

Grad g, Durchmesser d, Knotenanzahl N

Nach x Hops erreiche ich maximal g^x Knoten

Mit d Hops erreiche ich jeden Knoten im Netz

Also: $g^d = N$

Nach d aufgelöst: $\log(g^d) = \log(N)$, d.h. $d = \frac{\log(N)}{\log(g)}$

Es folgt: mit $g = \log(N)$ ist bester Durchmesser $d = \frac{\log(N)}{\log(\log(N))}$

Mit konstantem Grad ist aber ein logarithmischer Durchmesser möglich!

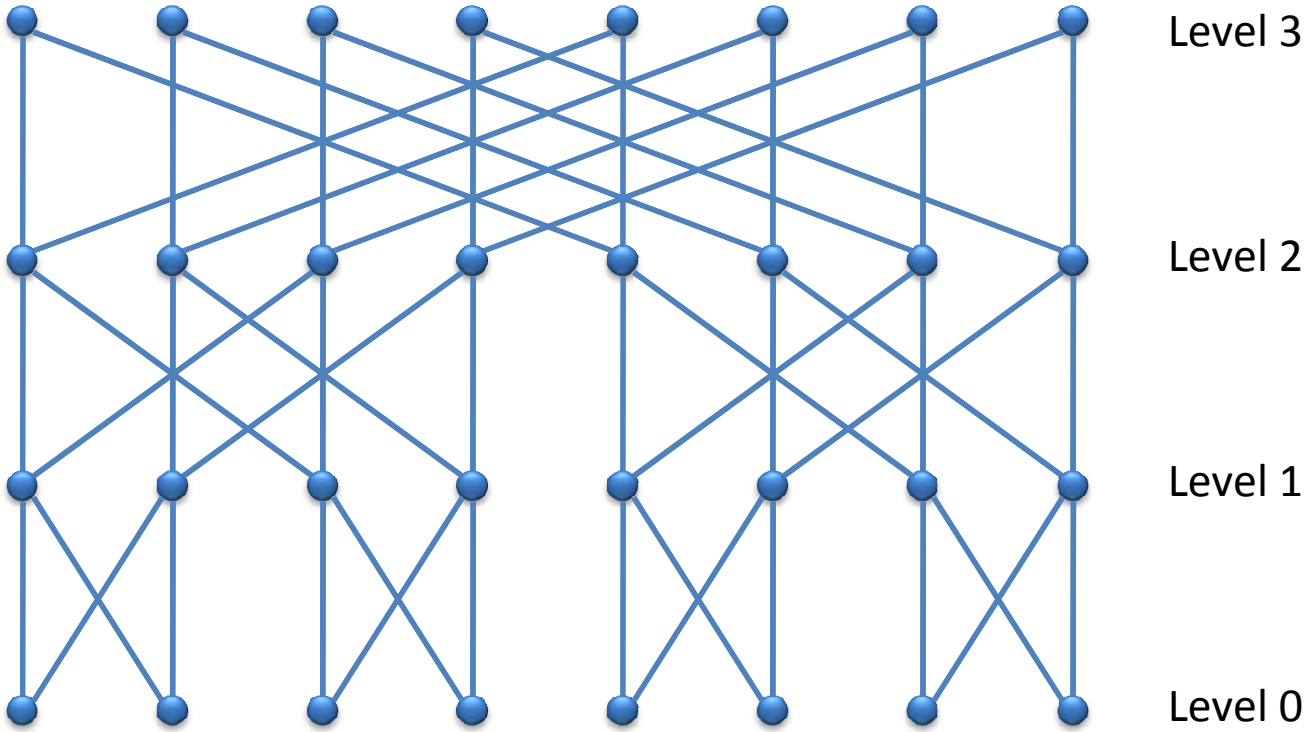
Netzwerke mit konstantem Grad

- Viceroy:
  - Dalia Malkhi, Moni Naor and David Ratajczak: "Viceroy: A Scalable and Dynamic Emulation of the Butterfly", ACM Symp. Principles of Distributed Computing, 2002
- Distance Halving:
  - Moni Naor, Udi Wieder: "Novel Architectures for P2P Applications: the Continuous-Discrete Approach", SPAA 2003
- Koorde:
  - M. Frans Kaashoek, David R. Karger: "Koorde: A simple degree-optimal distributed hash table", Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), 2003

Viceroy

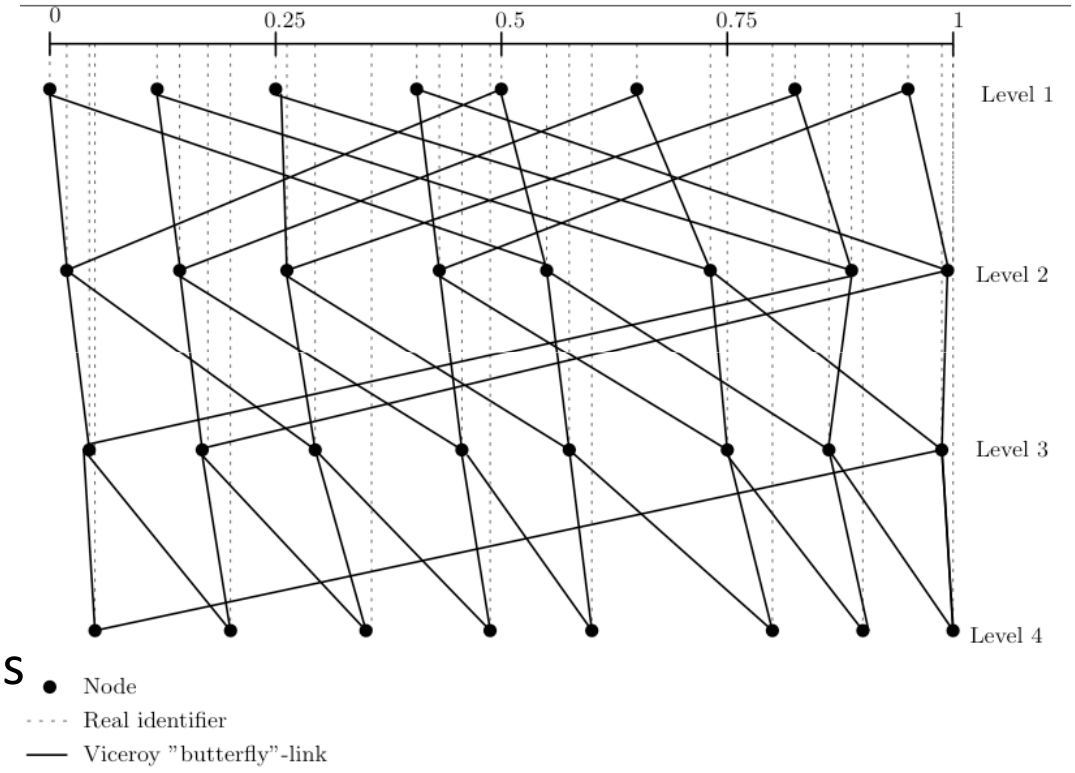
- Idee: Butterfly-Graph als Basis für P2P Netzwerke nehmen
  - Resultiert in einem Netz mit konstantem Grad
  - Ist aber ein relativ kompliziertes Verfahren
- Interessante "Abfallprodukte":
  - Wie schätze ich log(N)?
  - Wie stelle ich besser verteilte Intervalle her?
    - Speziell: Maximal konstante Abweichung, also z.B. Länge des größten Intervalls ist max. 4mal die Länge des kleinsten Intervalls
    - ➔ Lösung: Prinzip der mehrfachen Auswahl

Butterfly-Graphen



Viceroy Verbindungsstruktur

- Ein Ring mit allen Knoten
  - successor, predecessor
- Ein Ring pro Level
  - nextonlevel, prevonlevel
- Butterfly-Kanten
  - left: succ(l+1, s)
  - right: succ(l+1, s+1/2^l)
  - up: succ(l-1, s)
- Notation:
  - succ(s) ist Nachfolger von s in allgemeinem Ring
  - succ(l, s) ist Nachfolger von s auf Level l Ring



Quelle: Dalia Malkhi, Moni Naor and David Ratajczak:
"Viceroy: A Scalable and Dynamic Emulation of the Butterfly"

Routen in Viceroy

- cur: Aktueller Knoten, x: Ziel
- Routing in 3 Phasen
- **Phase 1**: Gehe zu Level 1
  - Solange dem up-Pointer folgen, bis cur.level = 1
- **Phase 2**: Nutzung der Butterfly-Kanten
  - Wenn der right-Pointer existiert und nicht zu weit zeigt
    - Folge dem right-Pointer
  - Wenn der left-Pointer existiert
    - Folge dem left-Pointer
  - Sonst mache mit Phase 3 weiter
- **Phase 3**: Auf dem Ring suchen
  - Folge dem successor-Pointer, bis Ziel erreicht

Knotenankunft

- Wähle eine ID s zufällig
- In allgemeinen Ring einfügen
  - Suche per Routing succ(s)
  - Passe Zeiger an (succ, pred, succ.pred, pred.succ)
- Levelauswahl
  - n0 = 1/distanz(s,succ(s))
  - Wähle einen Level l aus [1,…,floor(log n0)] zufällig
- In Level-Ring einfügen
  - Schrittweise Suche über allgemeinen Ring
  - Passe Zeiger an (nextonlevel, predonlevel, etc.)
- Butterfly-Kanten einfügen
  - up := succ(l-1, s) (Single-Stepping)
  - left := succ(l+1, s) (Single-Stepping)
  - right := succ(l+1, s+$(1/2^{l+1})$) (Lookup, dann Single-Stepping)

Abschätzung von log(N)

- Sei d der Abstand zum Nachbarn
- Wenn d=1/n, gilt log(n)=-log(d)
- Es gilt aber nur:
  - $1/n^2 \leq d \leq \log(n)/n$ mit hoher Wahrscheinlichkeit
- Betrachte -log(d):
  - Wenn $d = 1/n^2$, dann ist $-\log(d) = -\log(n^{-2}) = 2\log(n)$
  - Wenn $d = \log(n)/n$, dann ist $-\log(d) = -\log(\log(n)*n^{-1}) = \log(n) - \log(\log(n))$
- Daher gilt:
  - $2\log(n) \geq -\log(d) \geq \log(n) - \log(\log(n))$
  - Also ist -log(d) eine gute Abschätzung für log(n)

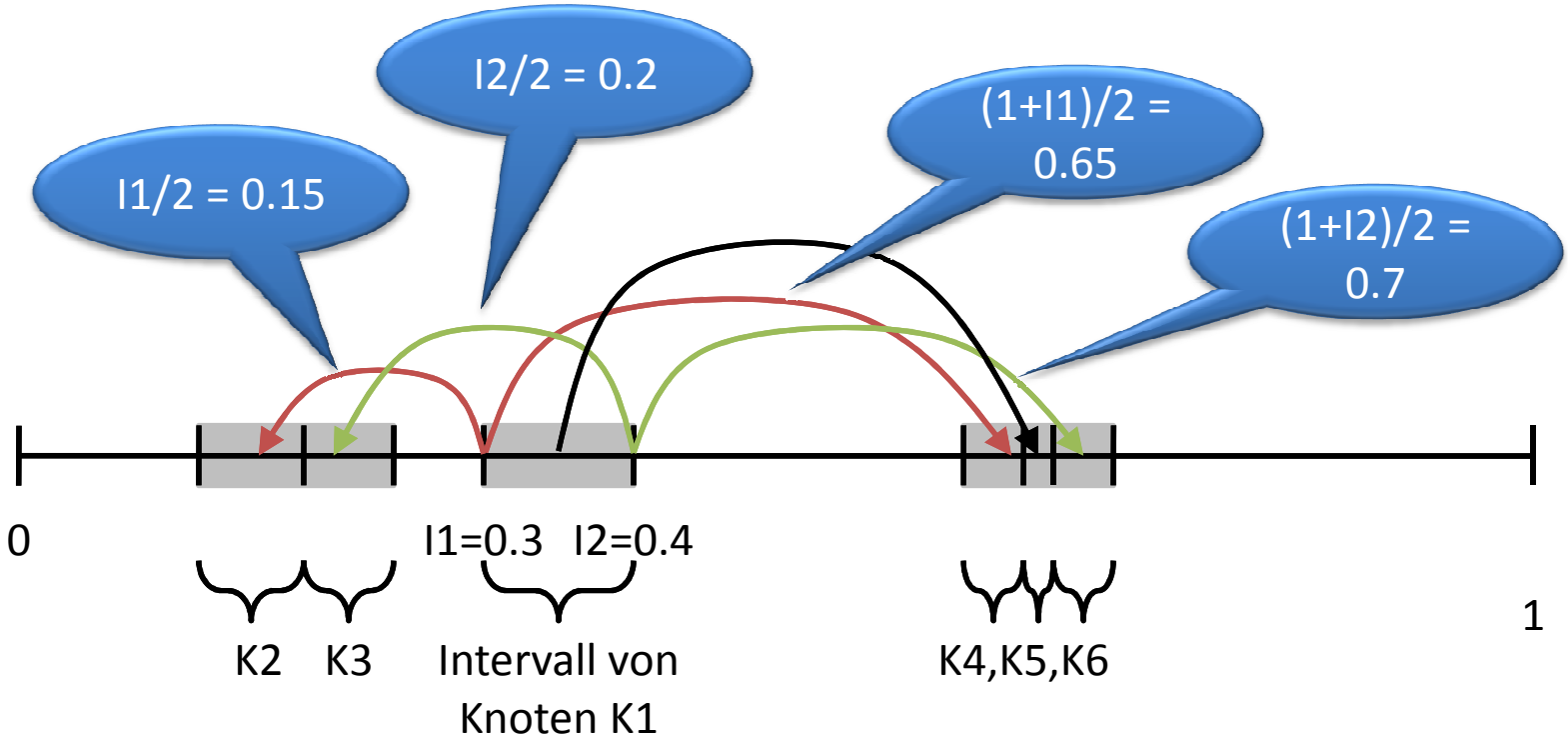
Prinzip der mehrfachen Auswahl

- Problem: wie erreiche ich konstanten Ein-Grad?
- Dazu müssen die Intervalle besser angepasst sein
- Bisher: Abweichungen sind:
  - um Faktor N kleiner
  - um Faktor log(N) größer
- Ziel:
  - Kleinstes Intervall ist 1/2N, größtes ist 2/N
  - Also Abweichung um max. Faktor 4 (konstant)
- Lösung: mehrfache Auswahl
  - Wähle bei der Knotenankunft zufällig ceil(log(N)) IDs
  - Positioniere den Knoten in der Mitte des größten Intervalls

Distance Halving

- ID-Bereich ist Intervall [0,1)
- Diesmal nicht zyklisch!
- Jeder Knoten bekommt ein Intervall nach dem Prinzip der mehrfachen Auswahl zugewiesen
  - Also hat jeder Knoten ein Intervall der Größe $1/2N \leq I \leq 2/N$
- Sei das Intervall des Knotens [I1, I2)
- Für jedes I aus [I1, I2) hat der Knoten Zeiger auf I/2 sowie (1+I)/2
- Es werden auch die Rückwärtskanten verwaltet!
- Damit ist sowohl der Ein-Grad aus auch der Aus-Grad Konstant!

Distance Halving



Routing-Tabelle von K1 enthält Kanten zu jedem Peer in den Ziel-Intervallen, plus die Rückwärts-Kanten. Dies ergibt einen konstanten Grad!

Routing in DH

- Wie erreiche ich vom Knoten I=[I1, I2) eine ID, die im Intervall von Knoten J=[J1, J2) liegt?
- Verfolge die Links-Kanten beider Knoten:
  - I trifft auf I1, I1/2, I1/4, I1/8, …
  - J trifft auf J1, J1/2, J1/4, J1/8, …
- Irgendwann liegen beide Zeiger in einem Intervall:

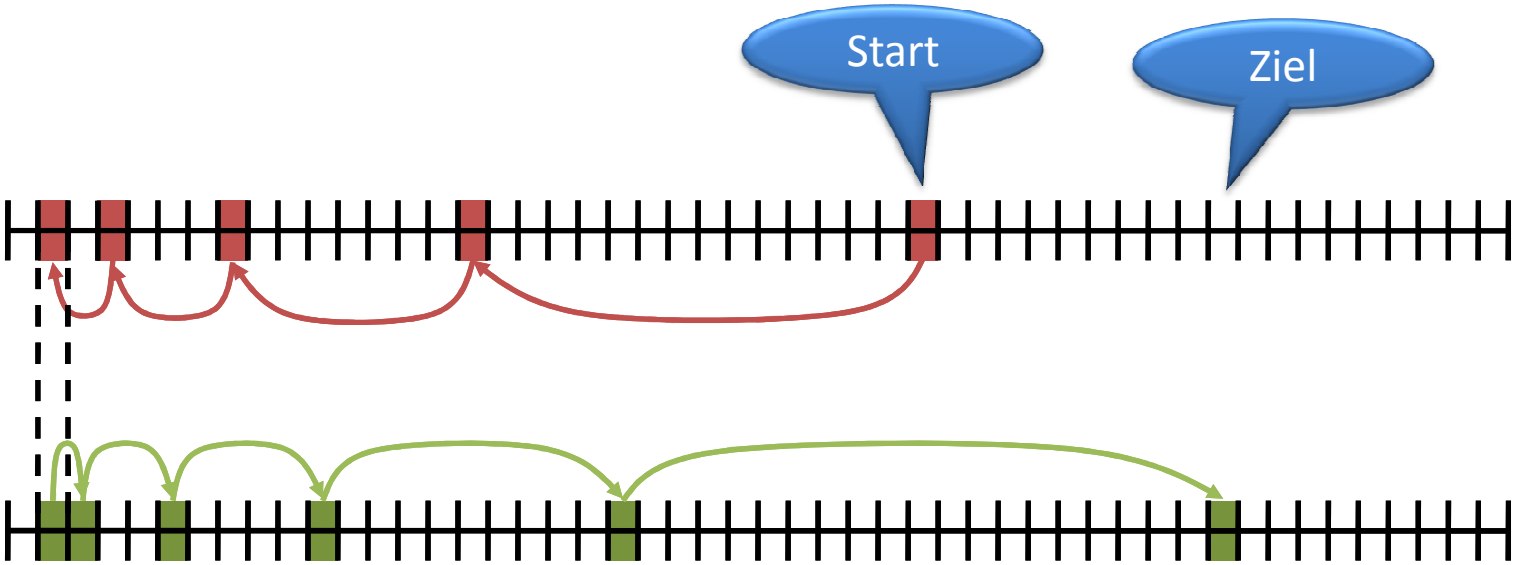
O.B.d.A. sei $J_1 > I_1$

Gesucht ist ein k mit : $\frac{J_1}{2^k} - \frac{I_1}{2^k} \leq \frac{2}{N}$ (maximale Intervallgröße)

$$\Leftrightarrow \frac{J_1 - I_1}{2^k} \leq \frac{2}{N} \Leftrightarrow \frac{2^k}{J_1 - I_1} \geq \frac{N}{2} \Leftrightarrow 2^k \geq N\frac{J_1 - I_1}{2} \Leftrightarrow k \geq \log(N\frac{J_1 - I_1}{2})$$

Ab diesem k sind beide Pointer auf dem gleichen Knoten. D.h. es werden max. $O(\log N)$ Schritte benötigt!

Routing in DH



Dieses Verfahren kann zu Überlast auf den vorderen Knoten führen!

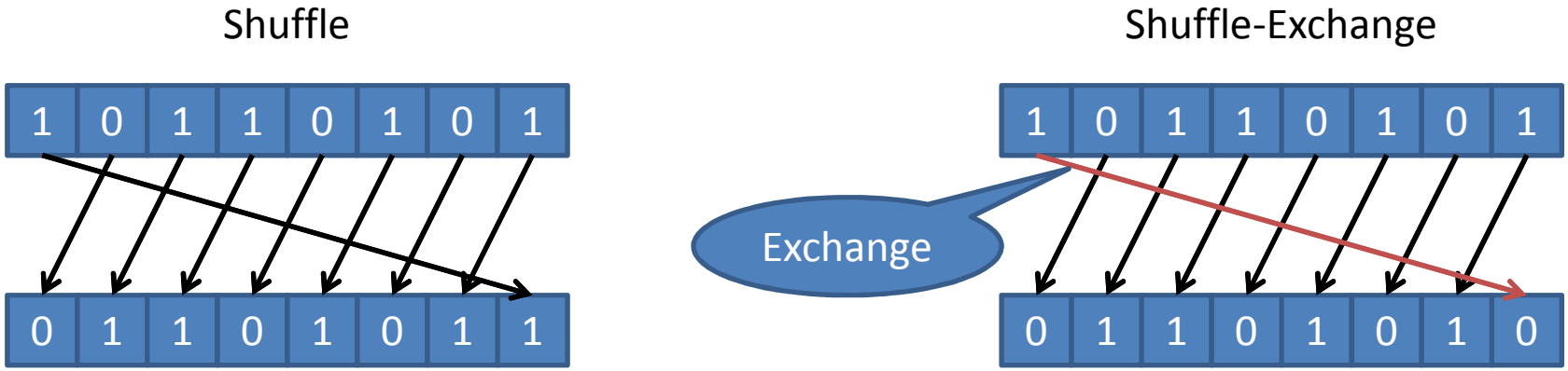
Gibt es noch alternative Wege?

Alternative Routen

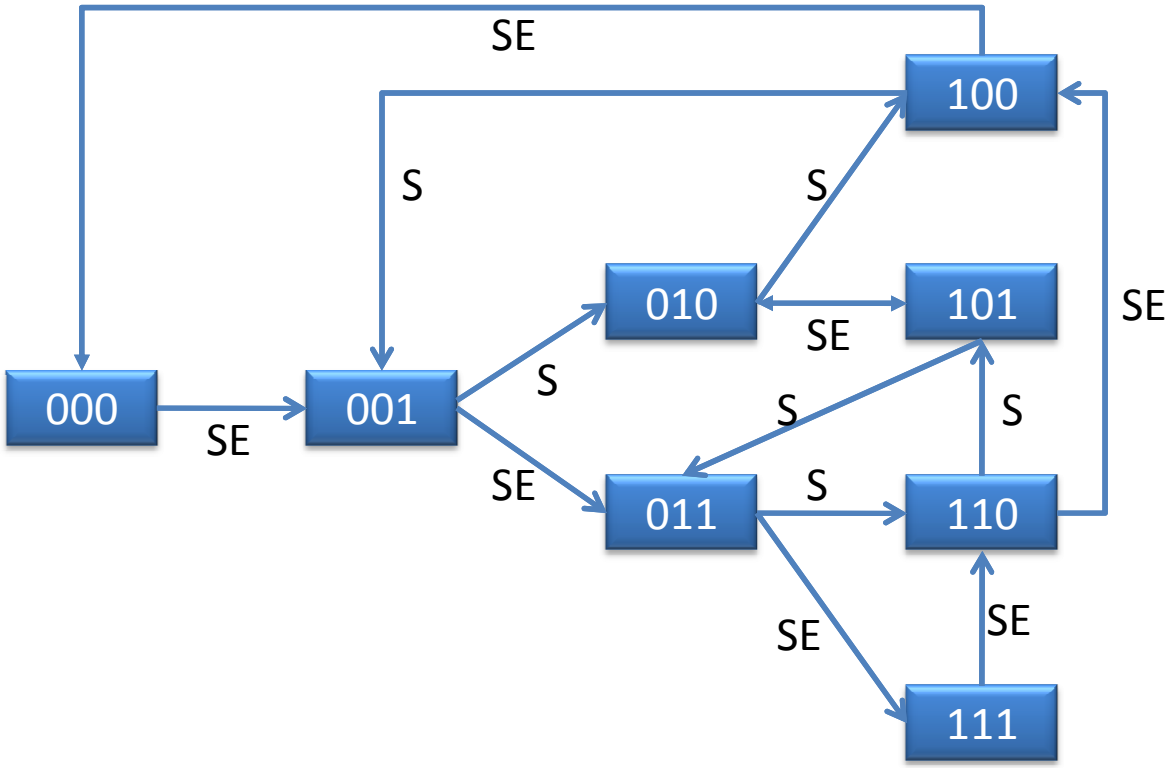
- Beobachtung:
  - I/2-J/2 = (I-J)/2
  - (1+I)/2-(1+J)/2 = (I-J)/2
- D.h., egal ob ich einen Doppel-Sprung nach Links oder nach Rechts mache:
  - Die Restdistanz wird immer halbiert
- Daher:
  - Zur Lastbalancierung wird die Reihenfolge zufällig bestimmt
  - z.B.: Recht – Links – Links – Rechts – Links etc.

Koorde

- Betrachte folgende Operationen auf Bit-Strings:
  - Shuffle: Bits um eine Stelle zyklisch rotieren
  - Shuffle-Exchange: Bits um eine Stelle zyklisch rotieren, dabei ein Bit invertieren



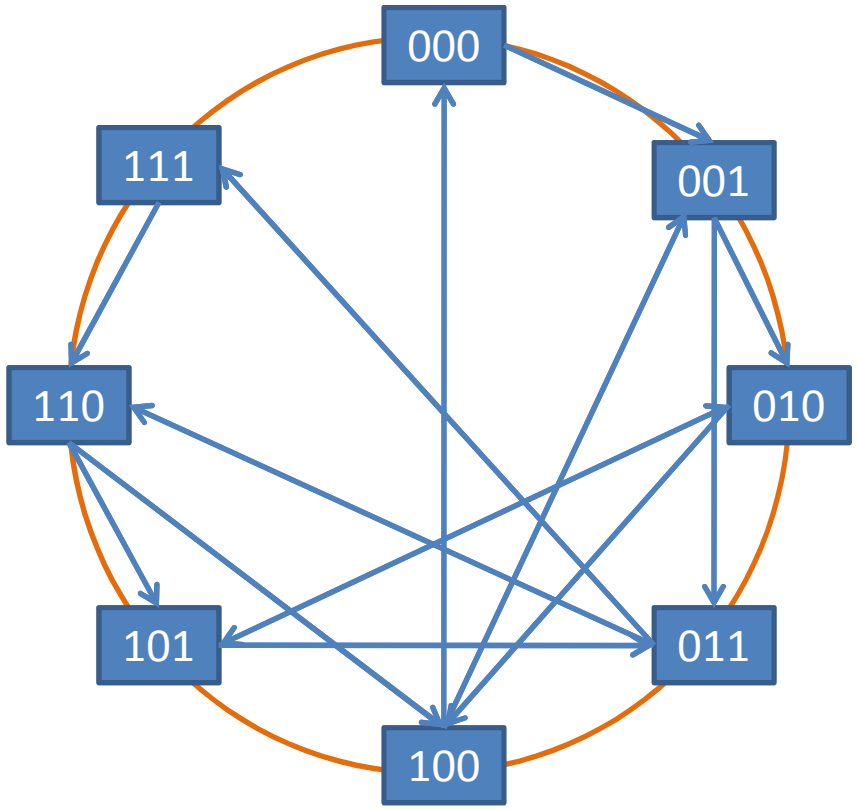
Der DeBruijn Graph



Der Graph ist verbunden und hat
konstanten Ein- und Ausgrad!

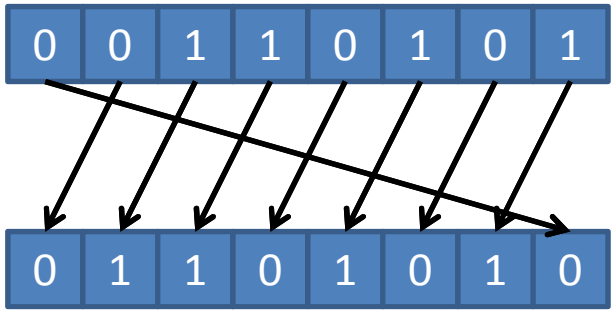
DeBruijn Graph als P2P Netz

- Knoten werden auf Ring ansortiert (wie Chord)
- Statt Chord-Fingerzeigern werden DeBruijn-Kanten eingesetzt



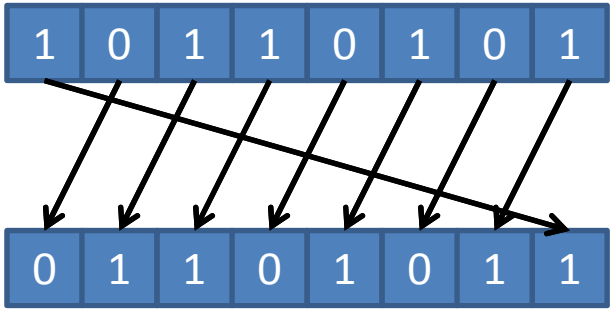
Wohin zeigen die DeBruijn-Kanten?

Shuffle mit führender 0



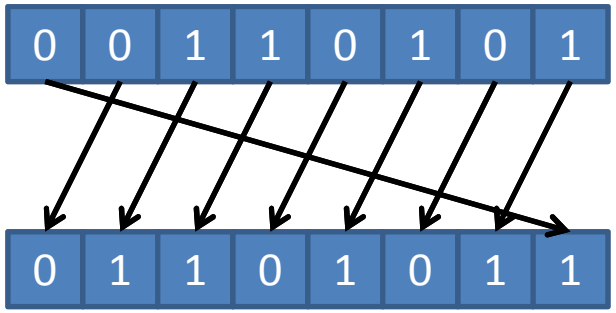
ergibt (2*x)

Shuffle mit führender 1



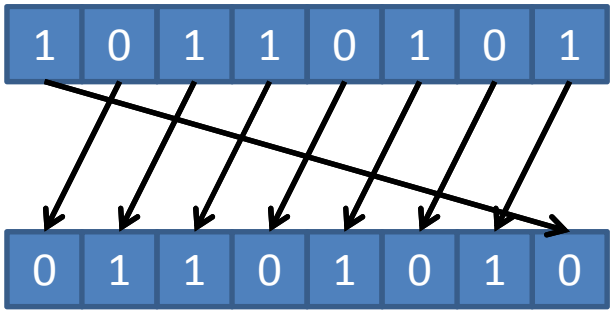
ergibt (2*x+1) mod 2^m

SE mit führender 0



ergibt (2*x+1)

SE mit führender 1



ergibt (2*x) mod 2^m

Ersatz-Knoten

- Netz wird als DeBruijn-Graph mit 2^m Knoten aufgebaut (m Bits)
- Knoten werden mit dem Prinzip der mehrfachen Auswahl auf den Ring gemapped
- Jeder Knoten ist zusätzlich für die nachfolgenden imaginären DeBruijn-Kanten verantwortlich
- Er verwaltet aber nur einen 2n mod 2^m-Zeiger
- Der (2n+1) mod 2^m-Zeiger geht sowieso mit hoher Wahrscheinlichkeit auf den gleichen realen Knoten
- Verantwortlichkeiten sind wie bei Chord geregelt

Routing in Koorde

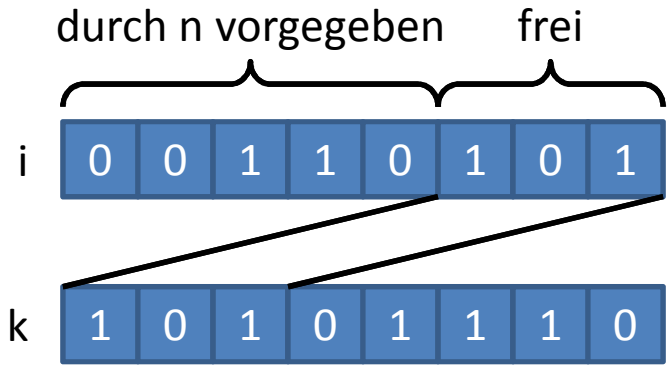
```
1    n.lookup(k, kshift, i)
2       if k in (n, successor]
3          return successor;
4       else if i in (n, successor] then
5          return d.lookup(k, (kshift o 0) mod 2^m,
                              (i o topBit(kshift)) mod 2^m)
6       else
7          return successor.lookup(k, kshift, i)
```

Erklärungen:

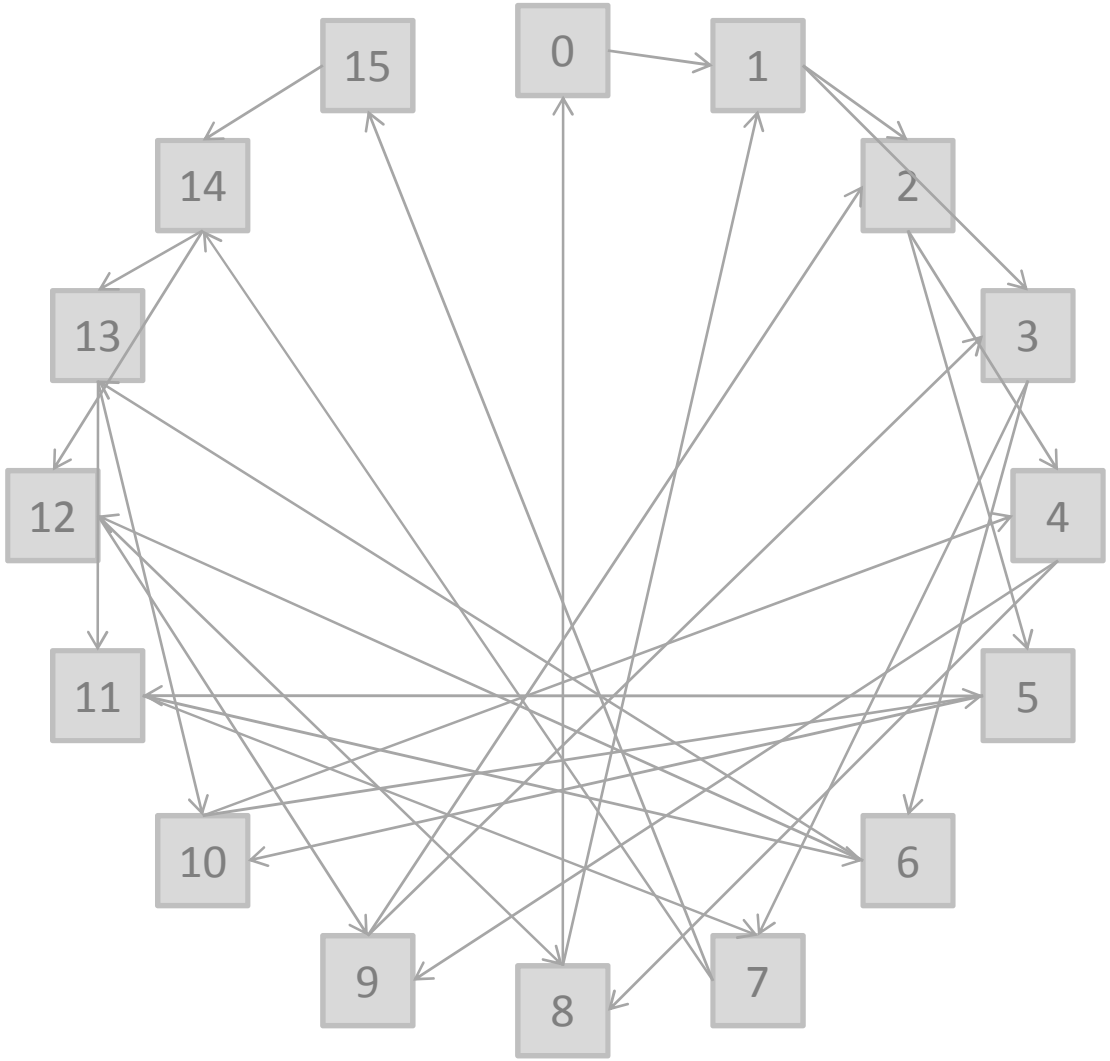
n	Aktueller Knoten
k	Key, zu dem geroutet wird
successor	Nachfolger von n auf dem Ring
d	Vorgänger von 2n (DeBruijn-Kante)
i	Imaginärer DeBruijn-Knoten
kshift	Anfangs kshift=k, dann immer weiter geshiftet

Imaginärer Startknoten

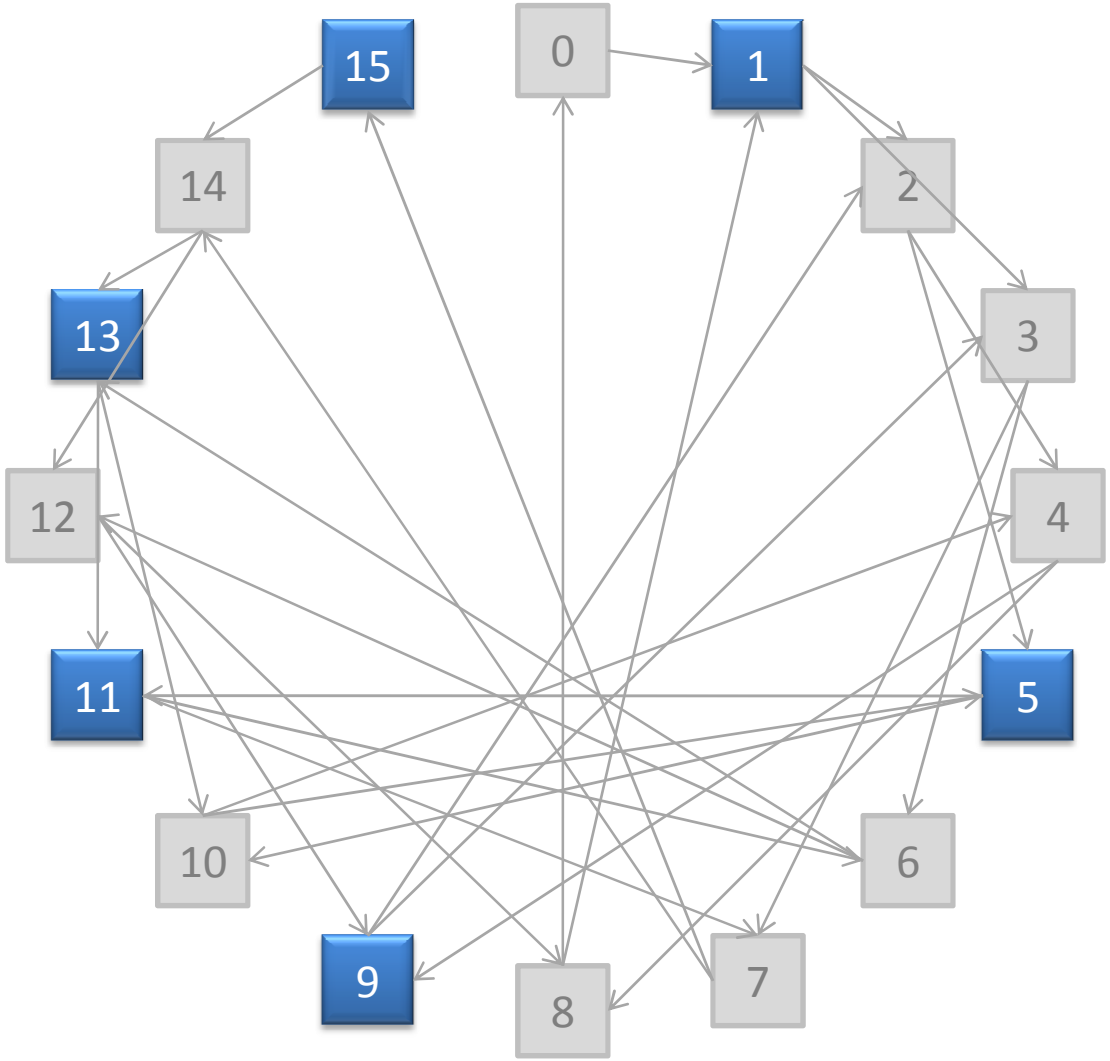
- Routing-Schritte bisher:
  - Anzahl Bits im Identifier (m)
  - plus Anzahl successor-Schritte
- Wie erreiche ich log(N) Routing?
- In der Wahl des imaginären Startknotens habe ich Freiheiten:
  - die unteren Bits können frei gewählt werden
  - Ich kann sie an die oberen Bits des Schlüssels anpassen:



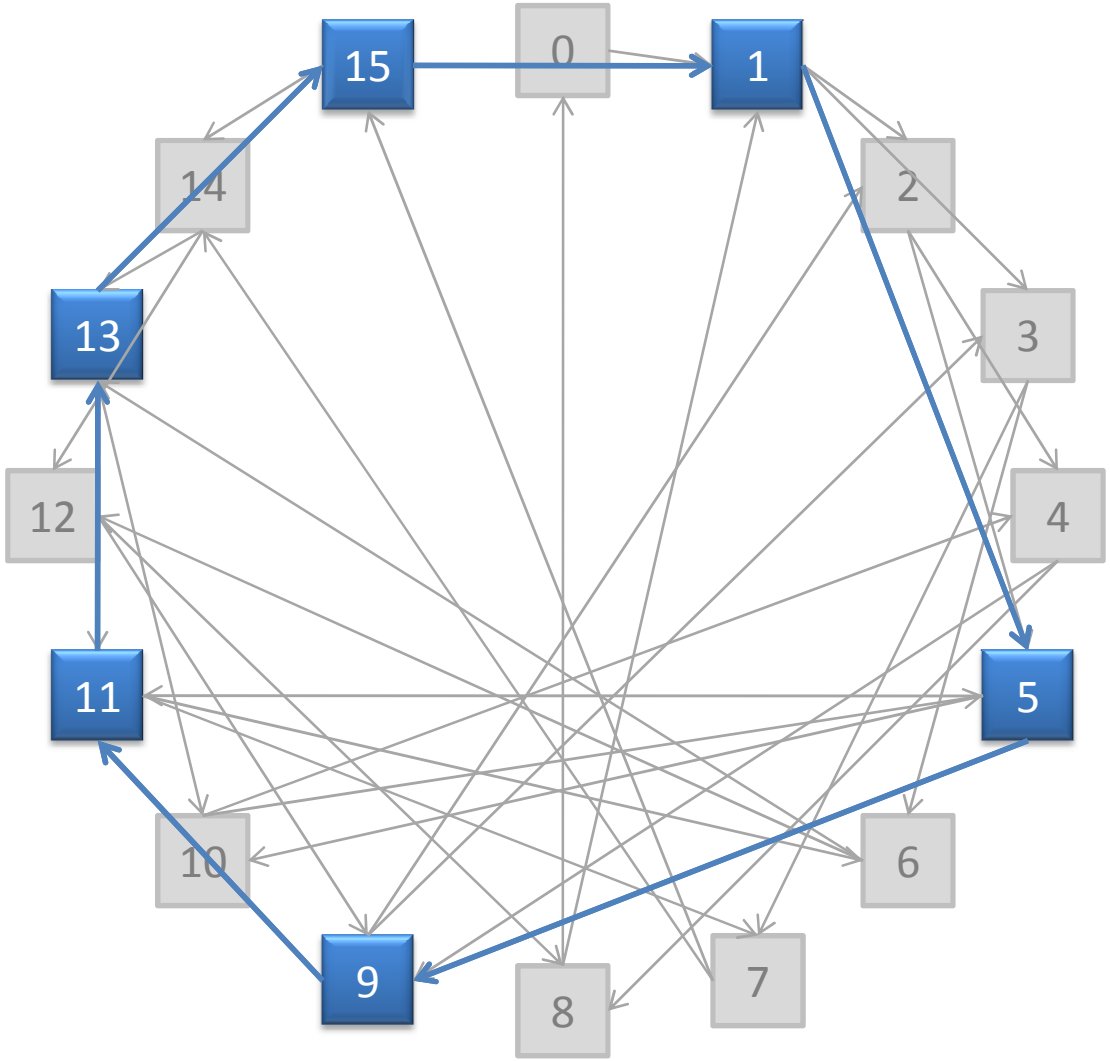
Koorde



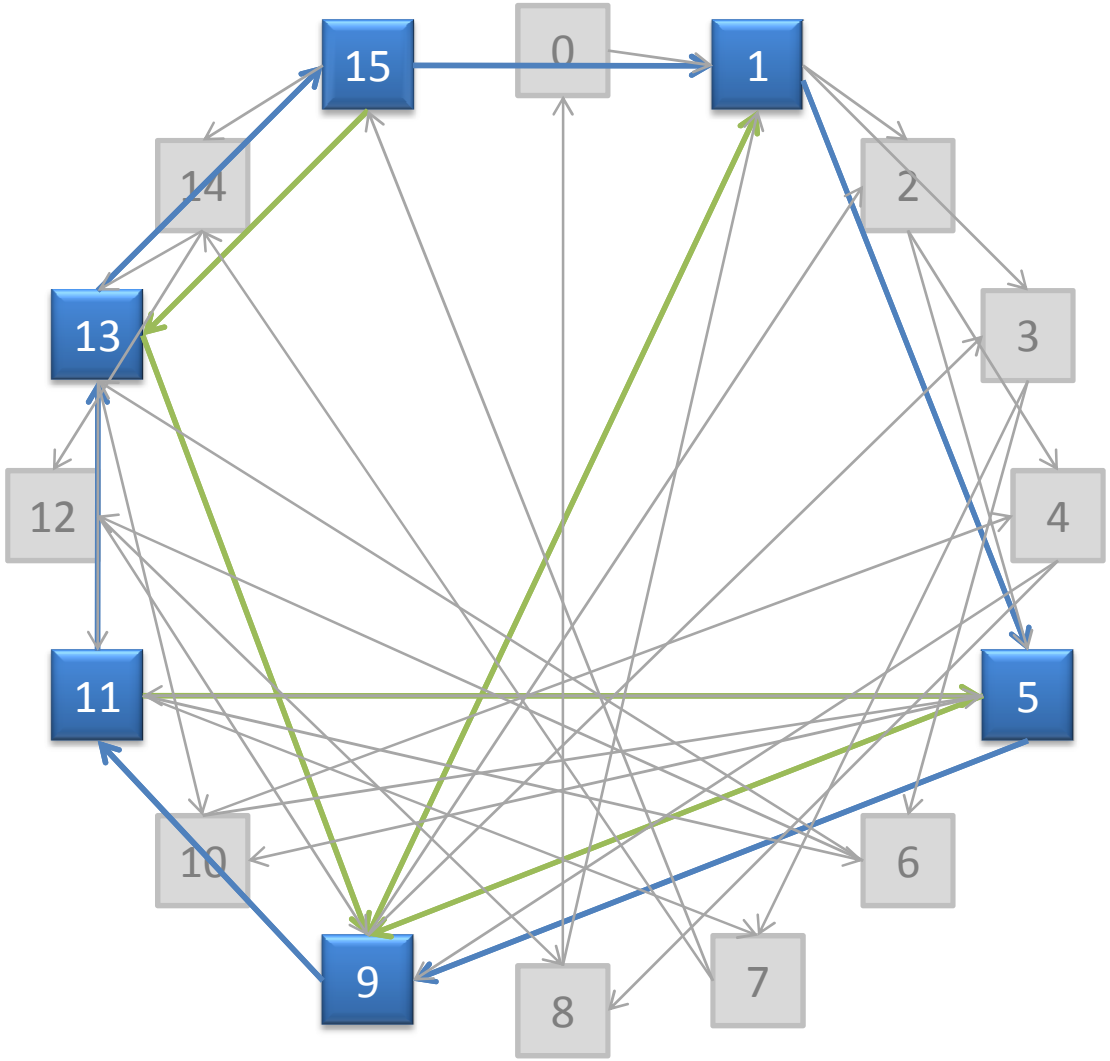
Koorde



Koorde



Koorde



Route von 13 nach 12:

Quelle kann sein:
1101 (13)
1110 (14)

Ziel ist
1100 (12)

Imaginäre Route:
1110 -> (SE) -> 1100
14 -> 12

Tatsächliche Route:
13 -> 9 -> 11

Zusammenfassung

- Gradoptimierte Netze:
  - Bei Grad O(1) ist Durchmesser O(log(N)) erreichbar!
- Ansätze:
  - Viceroy: Butterfly-Graph simulieren
  - Distance Halving: Distanz in jedem Schritt halbieren
  - Koorde: DeBruijn-Graph simulieren
- Bewertung:
  - Interessante Ansätze, in der Praxis nicht erprobt
  - Stabilisierung, Lokalität etc. schwierig bzw. nicht erforscht
  - Höherer Grad für Zusammenhalt durchaus wünschenswert
- "Abfallprodukte":
  - Schätzung von log(N)
  - Prinzip der mehrfachen Auswahl

Vorlesung
P2P Netzwerke

7: SkipNet, P-Grid

Dr. Felix Heine

Complex and Distributed IT-Systems

felix.heine@tu-berlin.de

SkipNet

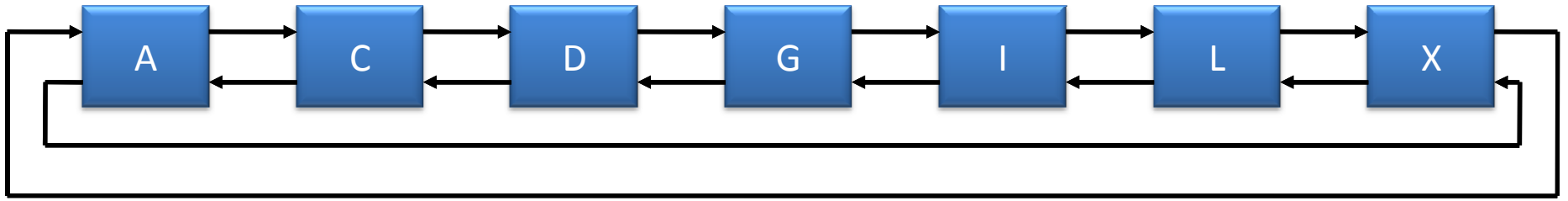
- Nicholas J.A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, Alec Wolman: "SkipNet: A Scalable Overlay Network with Practical Locality Properties", Proceedings of USITS, 2003
- James Aspnes, Gauri Shah: "Skip Graphs", Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2003, pp. 384–393
- William Pugh: "Skip Lists", Commun. ACM, 33:6, pp. 668-676, 1990

Probleme mit DHTs

- DHTs "zerhacken" die Daten (wie normale Hash-Tabellen auch)
- Daten "Meyer" und "Meier" landen auf völlig unterschiedlichen Knoten
- Daher sind keine Bereichsanfragen möglich (z.B.: Alle Daten die mit "M" beginnen)
- Eine Lösung dafür: **SkipNet**
- SkipNet basiert auf der Idee der Skip-Graphen, die wiederum auf Skip-Listen basieren

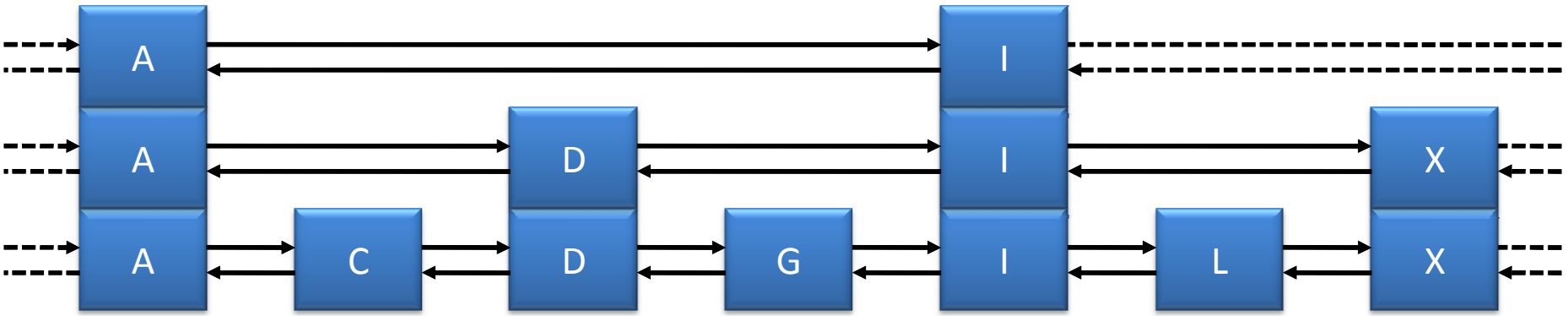
Skip-Listen

- Ausgangspunkt: Verkettete Liste
- Hier: direkt zu einem Ring verbunden
- Suche ist in nur in linearer Zeit möglich
- Zur Beschleunigung: Weiter entfernte Listenelemente "verzeigern"



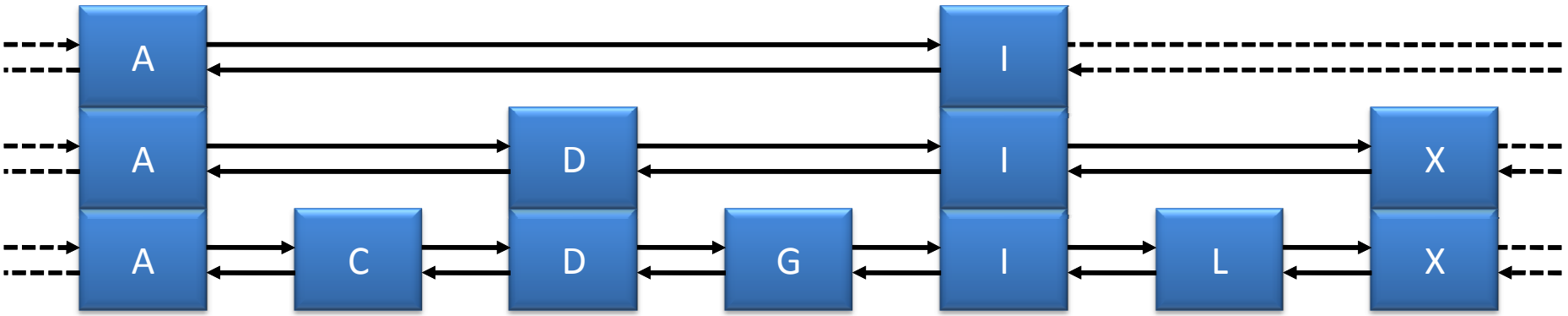
Skip-Listen

- Elemente bekommen eine zufällige Höhe
- Zusätzliche Verzeigerung in jeder Höhe



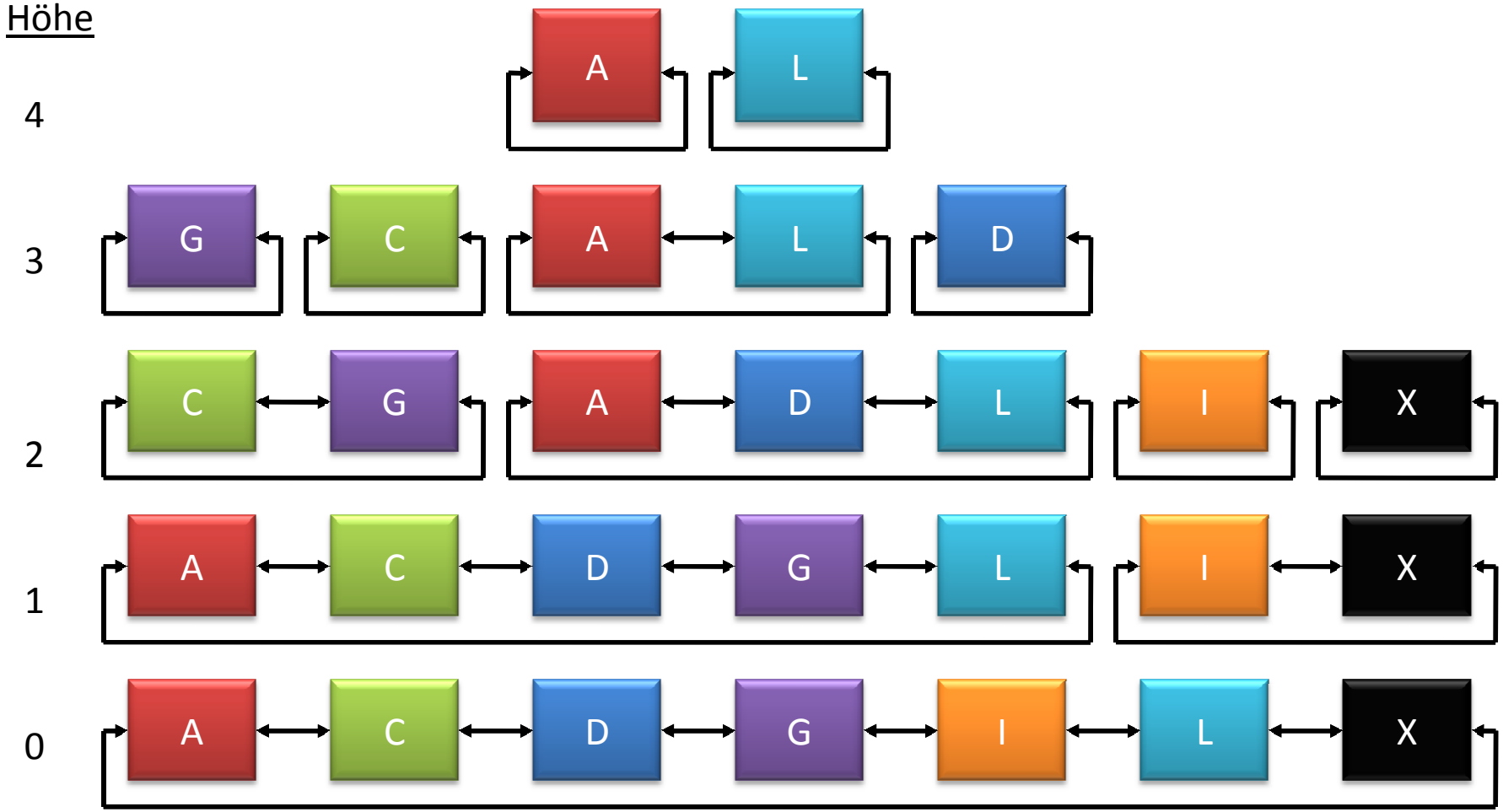
Skip-Listen

- Skip-Liste als P2P Netzwerk?
- Jedes Listenelement ist ein Knoten?
- Probleme: Knotenausfälle, Last



Skip-Listen → Skip-Graphs

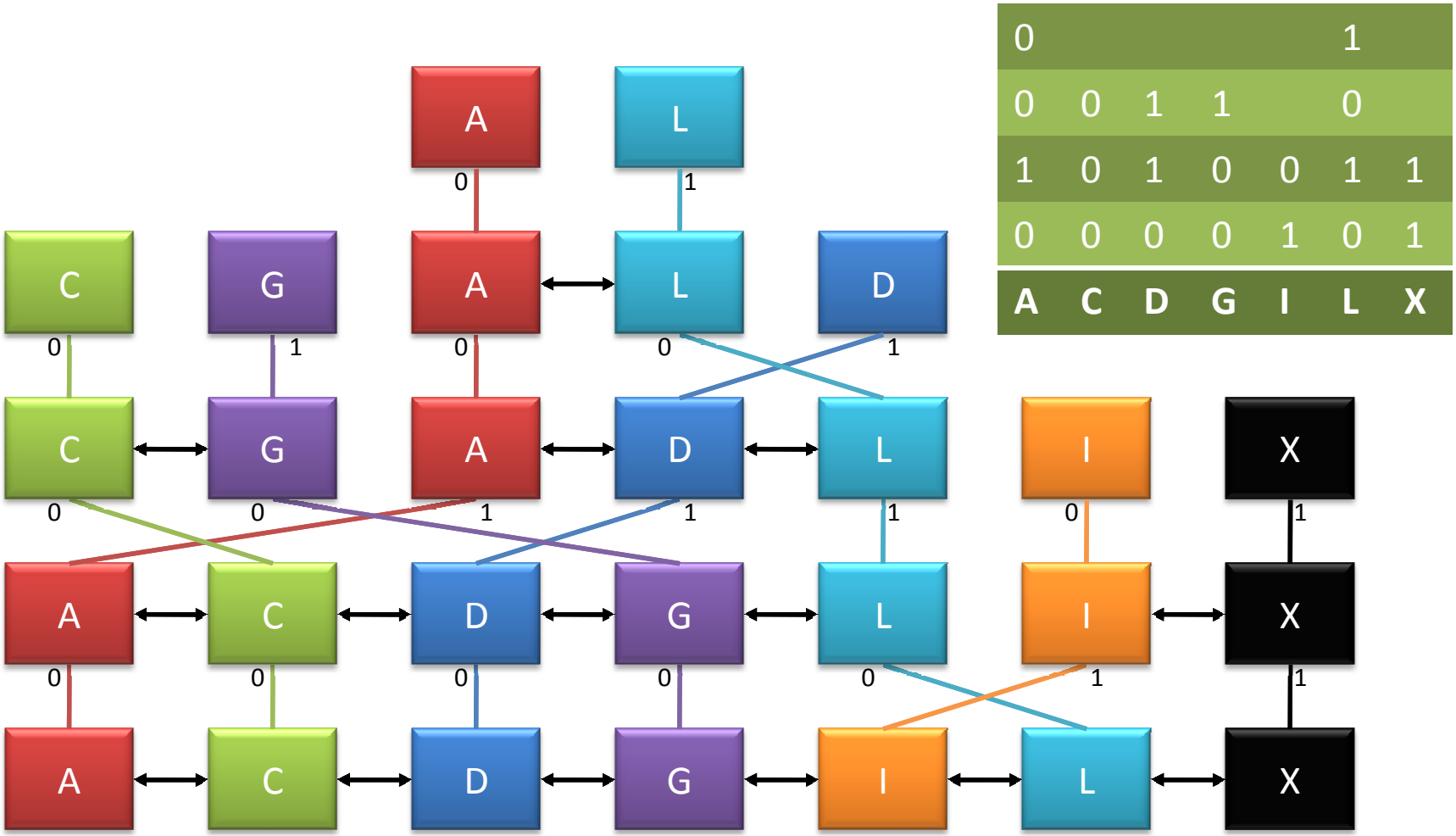
- Idee: Durch weitere Verbindungen Zusammenhang stärken



Routing

```
1    // route a msg by name ID
2    RouteByNameID(msg)
3        h = localNode.maxHeight
4        while h >= 0
5            nbr = localNode.RouteTable[h]
6            if nbr.nameID in [localNode.nameID, msg.nameID]
7                SendToNode(msg, nbr)
8                return
9            h = h - 1;
10       // h<0 implies we are the closest node.
11       DeliverMessage(msg)
```

Skip-Net: Num-ID



0						1
0	0	1	1		0	
1	0	1	0	0	1	1
0	0	0	0	1	0	1
A	**C**	**D**	**G**	**I**	**L**	**X**

In welche Ringe wird ein Knoten aufgenommen?

- numID eines Knotens bestimmt die Ring-Teilnahme
- Zufällige Bitfolge (128 Bit)
- Wird beim Start vom Knoten ausgewürfelt

- Wie findet der Knoten die Ringe?
  - Ein Knoten des Netzwerkes muß bekannt sein
  - Dieser wird kontaktiert
  - Über eine Nachricht wird ein Knoten gesucht, dessen numID möglichst start mit der des neuen Knotens übereinstimmt
  - → Routing nach numID

Routing nach numID

- Statusinformationen:
  - **ringLvl** (Anfangswert 0)
    - Auf welchem Level bewegt sich die Nachricht?
  - **startNode** (Anfangswert nil)
    - Bei welchem Knoten begann der Durchlauf im aktuellen Ring?
  - **bestNode** (Anfangswert nil)
    - Welcher Knoten ist der bisher dichteste?
  - **finalDestination** (Anfangswert false)
    - Hat die Nachricht bereits den Zielknoten erreicht?

Routing nach numID

```
1    // route msg via numeric ID
2    RouteByNumericID(msg)
3        if msg.numID == localNode.numID || msg.finalDestination
4            DeliverMessage(msg)
5            return
6        if localNode == msg.startNode
7            msg.finalDestination = true
8            SendToNode(msg.bestNode)
9            return
10       h = CommonPrefixLen(msg.numID, localNode.numID)
11       if h > msg.ringLvl
12           msg.ringLvl = h
13           msg.startNode = msg.bestNode = localNode
14       else if |localNode.numID-msg.numID| <
                     |msg.bestNode.numID-msg.numID|
15           msg.bestNode = localNode
16       nbr = localNode.RouteTable[msg.ringLvl]
17       SendToNode(nbr)
```

ID's in SkipNet

- Jeder Knoten hat eine Name-ID und eine Num-ID
- Die Name-ID ist frei wählbar, die Num-ID ist eine gewürfelte Bitfolge
- Die Name-ID bestimmt die Reihenfolge im Root-Ring
- Die Num-ID bestimmt, in welchen Ringen auf höheren Ebenen der Knoten partizipiert
- Routing ist über beide IDs möglich

Hinzufügen von Knoten

- Über bekannten Knoten wird Nachricht an die Num-ID des neuen Knotens geschickt
- Dieser kennt alle Ringe, in die der neue Knoten eingefügt wird
- Jeder Ring wird linear durchlaufen, um die Position des Knotens zu finden
- Dies erfolgt absteigend von oben
- Die Nachbarn werden zunächst nur aufgesammelt
- Am Ende wird in einer Operation überall der Insert durchgeführt

Stabilisierung des Root-Rings

- Im Root-Ring Liste von x Nachfolgern sammeln (z.B. x=8)
- Wenn der Nachfolger ausfällt, ist der neue Nachfolger bekannt
- Neuer Nachfolger wird über seinen neuen Vorgänger informiert
- Liste der Nachfolger wird jeweils an den Vorgänger weitergegeben

Stabilisierung der höheren Ringe

- In den höheren Ringen ist nur jeweils ein Nachfolger bekannt
- Wenn dieser ausfällt, ist der Ring zunächst defekt
- Routing ist trotzdem möglich
  - wie?
- Ein höherer Ring wird repariert, indem der neue Nachfolger im niedrigeren Ring gesucht wird
- D.h., die Ringe werden nacheinander "von unten nach oben" repariert
- Alternative: Auch in höheren Ringen mehrere Nachfolger speichern

Aufwand

- Routing nach name ID:
  - O(log N)
- Routing nach num ID:
  - O(log N)

- Intuition:
  - Die Zeiger in der Routing-Tabelle sind exponentiell verteilt, daher wird in jedem Schritt die Hälfte der restlichen Distanz übersprungen

Zusammenfassung

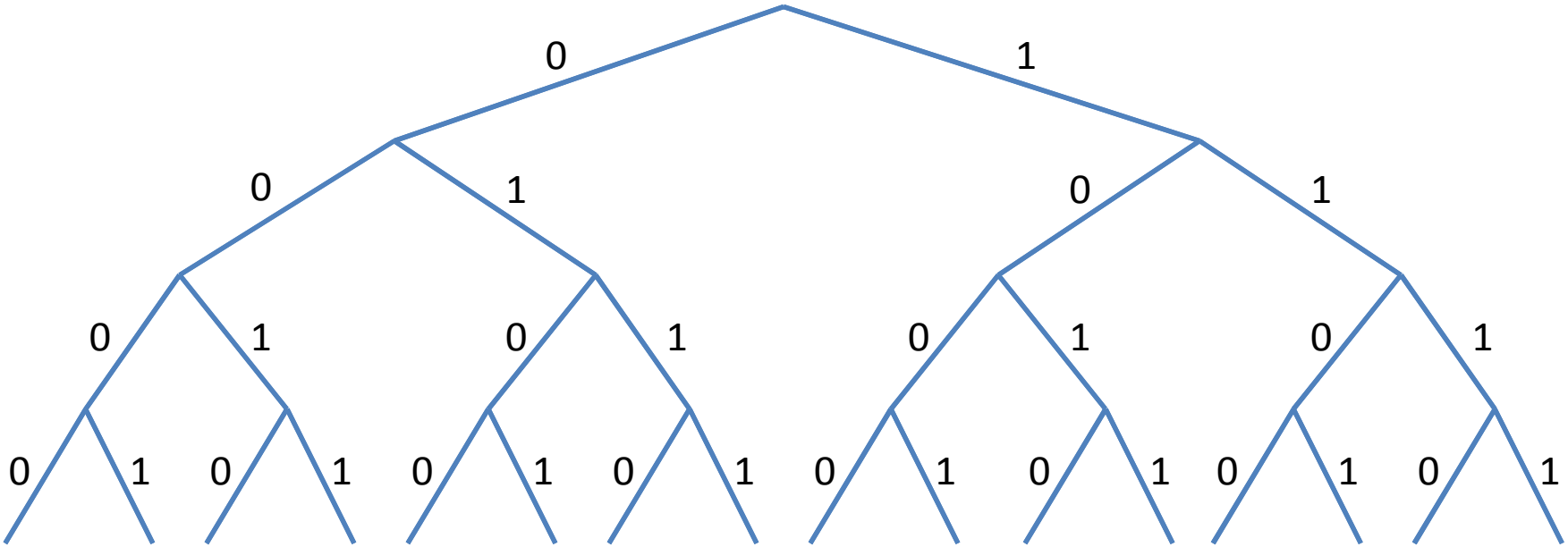
- SkipNet: Strukturiertes P2P Netzwerk ohne Hashing
- Daten werden sortiert auf einem Ring abgelegt
- Zur schnelleren Suche wird der Ring rekursiv in Teilringe aufgeteilt
- Knoten haben eine Name-ID und eine Numerische ID
- Suche ist nach beiden IDs möglich
- Name-ID Suche durchläuft die Ringe von oben nach unten
- Num-ID Suche durchläuft die Ringe von unten nach oben

P-Grid

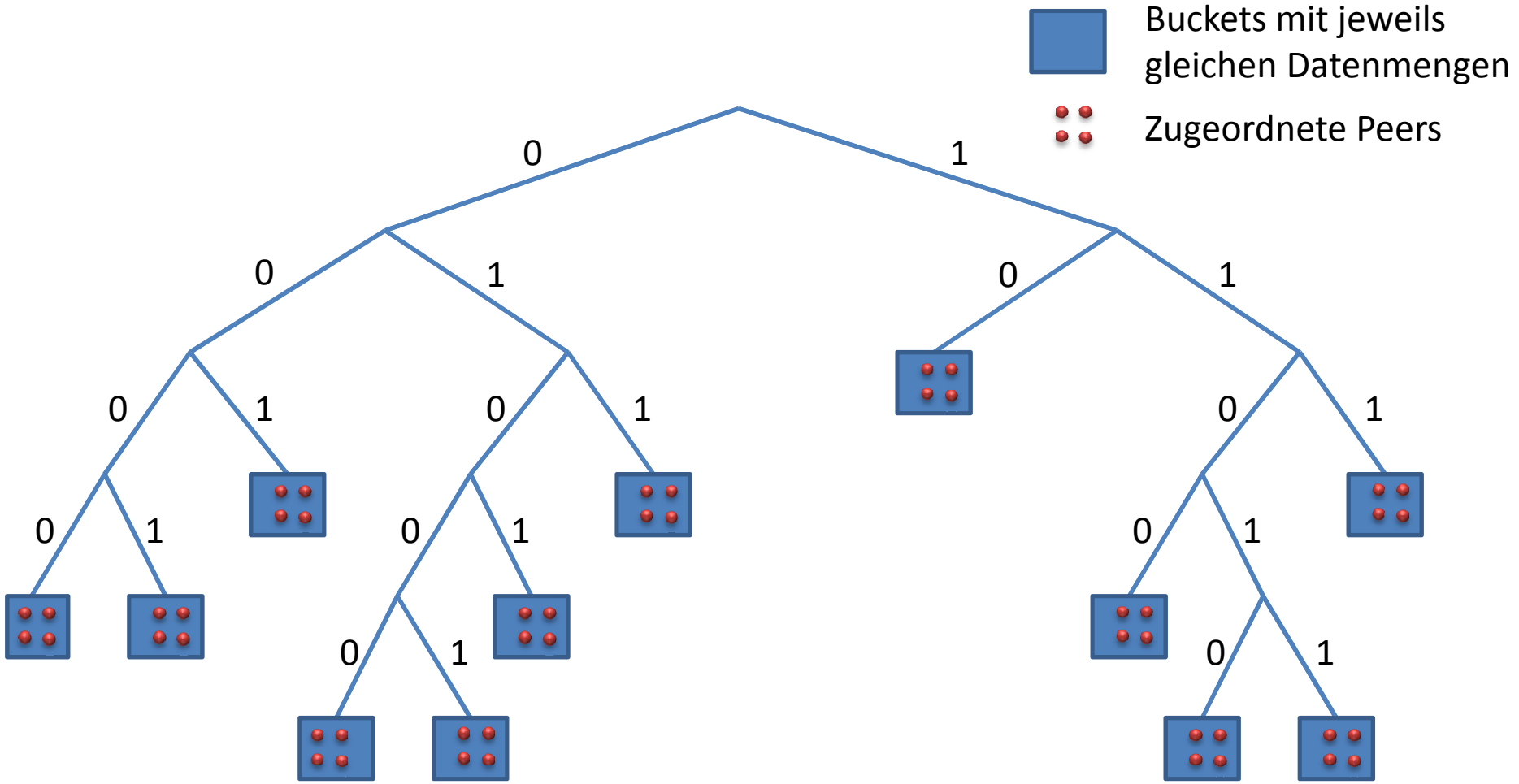
- Karl Aberer, Anwitaman Datta, Manfed Hauswirth, Roman Schmidt: "Das P-Grid-Overlay-Netzwerk: Von einem einfachen Prinzip zu einem komplexen System", Datenbank-Spektrum 13, 2005
- Karl Aberer: "P-Grid: A Self-Organizing Access Structure for P2P Information Systems", Sixth International Conference on Cooperative Information Systems (CoopIS 2001), Trento, Italy, Lecture Notes in Computer Science 2172, Springer Verlag, Heidelberg, 2001.
- www.p-grid.org

P-Grid

- Grundidee:
  - Schlüssel werden **nicht** gehasht
  - Schlüssel werden auf binäre Zeichenketten abgebildet
  - Routing erfolgt über einen Binärbaum



Aufbau von P-Grid



Routing in P-Grid

- Die ID eines Peers ist seine eindeutige Kennung im Netz
- Kennung hat nichts mit der Position im Baum zu tun
- Die Position wird dynamisch festgelegt und verändert
- Die Position nennt sich der **Pfad** des Peers
- Der Pfad ist eine binäre Zeichenkette beliebiger Länge
- Für jedes Präfix seines Pfades hat der Peer einen Zeiger auf andere Peers mit
  - gleichem Präfix
  - dann anderer Fortsetzung (0 → 1, 1 → 0)
  - falls so ein Peer existiert
- Woran erinnert das?
- Wie funktioniert das Routing?

Routing in P-Grid

- Präfixrouting wie bei Pastry / Tapestry
- Es gibt kein Surrogate-Routing, da nur 2 Spalten
- Pfade können aber beliebig lang werden
- Routing ist trotzdem schnell:
  - Es können häufig mehrere Bits in einem Schritt aufgelöst werden

Aufbau von P-Grid

- Grundidee:
  - Bestehendes unstrukturiertes Netz
  - Neuer Index (P-Grid Struktur) wird über das Netz gelegt
- Startpunkt: Baum nur mit Wurzel, alle Peers und alle Daten befinden sich in der Wurzel
- Rekursive Aufteilung:
  - Solange genügend Peers in einem Bucket:
  - Teile Bucket in zwei Hälften (0-Pfad, 1-Pfad)
  - Verteile Peers zahlenmäßig entsprechend der Datenmengen
- Dann dynamische Operationen
  - Peers kommen und gehen
  - Daten kommen und gehen
  - **hier nicht weiter betrachtet**

Zusammenfassung
Strukturierte P2P Netze

- Chord
  - log(N)-Routing bei log(N)-Grad
  - Intuitiv und einfach, gut zum Einstieg
- CAN
  - Fülle Interessanter Ideen, aber polynomielles Routing
- Pastry, Tapestry
  - Ähnlich zu Chord, berücksichtigen Lokalität
  - Pastry: Praktisch gut einsetzbar, Tapestry: Theorie interessant
- Viceroy, Distance Halving, Koorde
  - Gradoptimierte Netze
  - Interessante Ansätze, aber nicht praxistauglich
  - Wichtige Abfallprodukte

- SkipNet
  - Interessante Alternative zum Hashing, aber keine Lokalität
- P-Grid
  - Präfix-Ansatz ohne Hashing, Lokalität könnte berücksichtigt werden
  - Konzept des Bootstrappings eines Netzwerkes

Zusammenfassung
Strukturierte P2P Netze

- Chord
  - log(N)-Routing bei log(N)-Grad
  - Intuitiv und einfach, gut zum Einstieg
- CAN
  - Fülle Inte
- Pastry, Tapes
  - Ähnlich z
  - Pastry: Pr
- Viceroy, Distance Halving, Koorde
  - Gradoptimierte Netze
  - Interessante Ansätze, aber nicht praxistauglich
  - Wichtige Abfallprodukte

Hashing

-
-

Kein Hashing

Vorlesung
P2P Netzwerke
8: Lastverteilung

Dr. Felix Heine
Complex and Distributed IT-Systems
felix.heine@tu-berlin.de

Inhalt Lastbalancierung

- Welche Last balancieren?
- Balancierungseigenschaften von DHTs
- Techniken:
  - I: Replikation, Bereichsgrenzen ändern
  - II: Power of Two Choices
  - III: Virtuelle Peers
  - IV: Relokation
- Erkennung von Ungleichgewichten
  - Schätzverfahren
- Lastbalancierung in SkipNet

Literatur

- John Byers, Jeffrey Cosidine, Michael Mitzenmacher: "Simple Load Balancing for Distributed Hash Tables", Second International Workshop on Peer-to-Peer Systems, 2003
- André Höing: "Lastbalancierung von Speicherbedarf und Anfragen in BabelPeers", Diplomarbeit, Universität Paderborn, 2006
- Prasanna Ganesan, Mayank Bawa, Hector Garcia-Molina: "Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems", Proceedings of the 30th VLDB Conference, 2004

Lastverteilung

- Welche Ressource ist das Problem?
  - Ressourcen:
    - Netzwerklast: Bandbreitenverbrauch
    - Speicherlast: Festplattenverbrauch / Hauptspeicher?
    - Rechenlast: Verbrauch von CPU-Zyklen
  - Netzwerklast und Rechenlast sind momentbezogen
  - Speicherlast ist dauerhaft
  - Welche Last ist das Problem?
    - Das ist Anwendungsabhängig!
- Was soll erreicht werden?
  - Ist die ideale Lastverteilung die Gleichverteilung?
  - Oder eher angepasst an die Leistungfähigkeit des Knotens?

Lastverteilung beim Routen

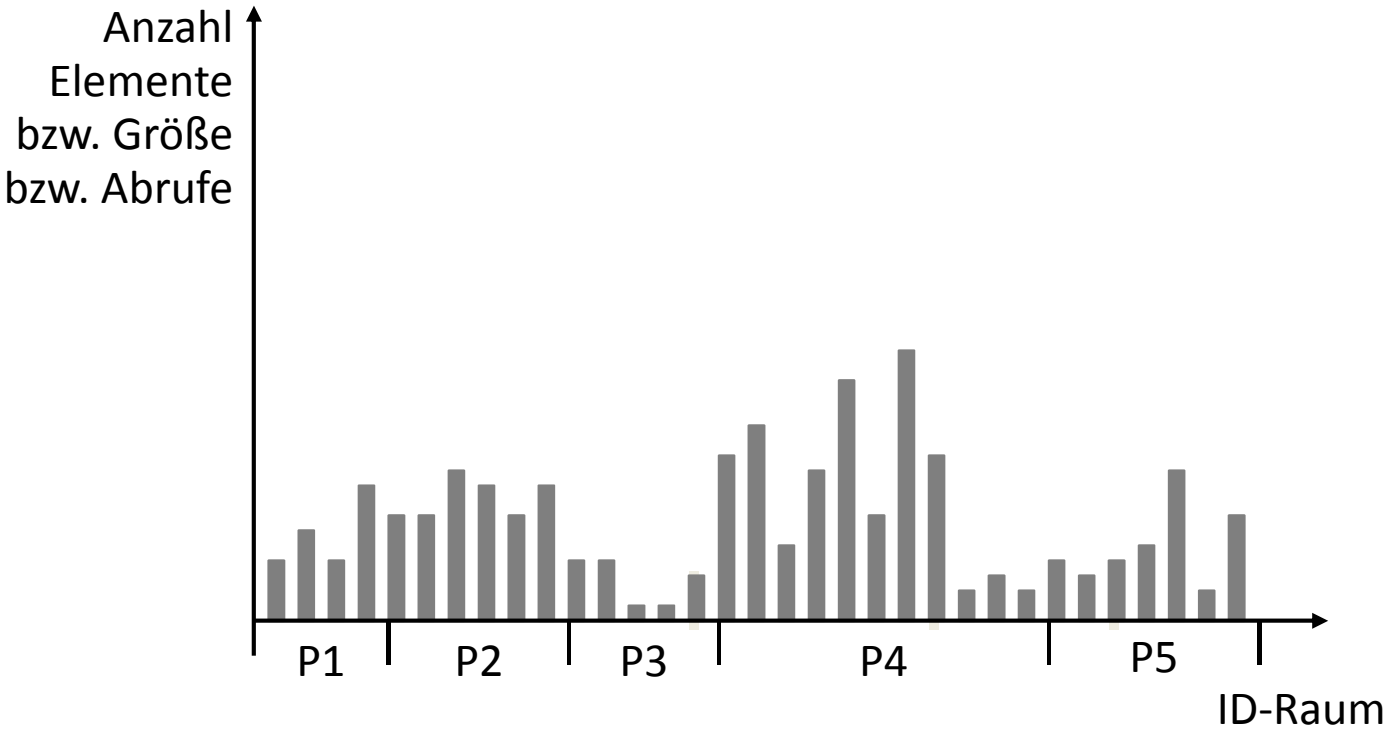
- Muss jeder Peer in etwa die gleiche Anzahl Nachrichten weiterleiten?
- Normalerweise durch die Symmetrie des Netzwerkaufbaus gegeben
- Probleme evtl. bei:
  - Viceroy: höheren Schichten evtl. mehr belastet
  - Distance Halving: Knoten am Rand
  - Skip-Listen als P2P-Netzwerk (daher Skip-Graphen)
- **hier nicht weiter betrachtet**

Lastbalancierung von DHTs

- DHTs haben folgende Annahme:
  - Die Bereiche der Knoten sind ähnlich groß
  - ➔ Prinzip der mehrfachen Auswahl
  - Die Hashfunktion liefert jede ID mit gleicher Wahrscheinlichkeit
  - Damit sind die Daten gleichmäßig über die Knoten verteilt
- Probleme:
  - Nicht jeder Knoten kann gleich viel Speichern
  - Evtl. sind die Daten nicht gleichverteilt (andere Hashfkt)
  - Hot-Spots: Schlüssel mit sehr vielen Daten

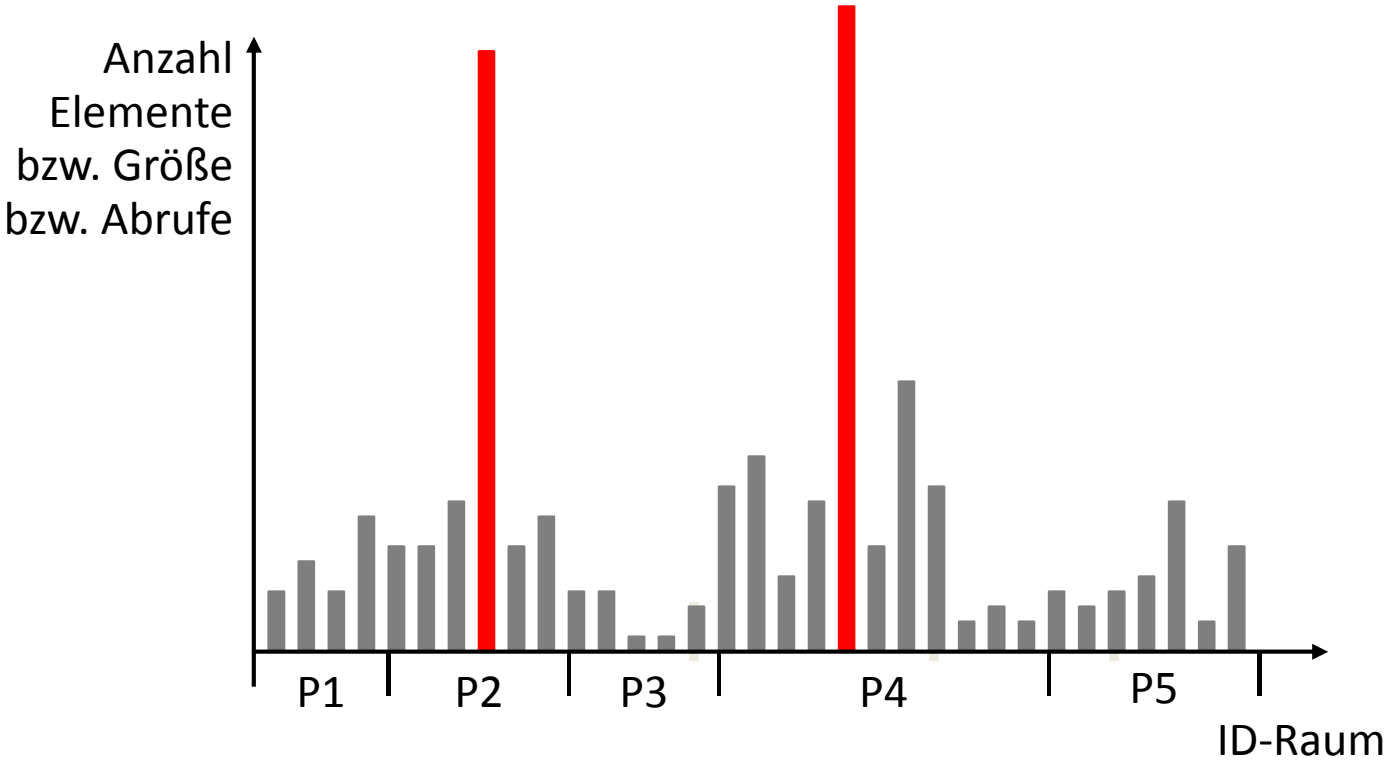
Ungleiche Last

- Beispiel-Verteilung:



Ungleiche Last

- Beispiel-Verteilung:



Ungleiche Last

- Ziel ist nicht mehr, dass jeder Knoten einen gleichen Teil vom ID-Raum bekommt
- Sondern: Gleich viel Last!
  - d.h.: großer ID-Raum mit wenig Last
  - oder kleiner ID-Raum mit großer Last
- Problem allerdings:
  - Hot-Spots, d.h. Schlüssel mit sehr viel Last
  - Diese können nicht aufgelöst werden, wenn nur die ID-Bereiche verschoben werden
- Lösungsansätze
  - Schlüssel eindeutig machen → Range-Queries
  - Vergleiche mit P-Grid Lösung, bzw. mit SkipNet
  - Lösungen in der Anwendung: z.B. Datei in Blöcke aufteilen
  - vgl. Techniken

Techniken I

- Bereichsgrenzen anpassen
  - Besonders belastete Bereiche verkleinern
  - Unausgelastete Bereiche vergrößern
  - Kann Hot-Spots an einer ID nicht ausgleichen
  - Schiebt "Last-Wellen" um den Ring
  - Daher: wenig ausgelastete Knoten neu Positionieren
- Replikation
  - Systemweiter Replikations-Faktor (z.B. jedes Datum auf 4 Knoten)
  - "Glättet" die Speicherlast, aber erhöht sie
  - Wird allerdings sowieso benötigt
  - Wenn bei Abfragen zufällig ein Replika-Knoten ausgewählt wird, wird auch Bandbreite / CPU balanciert
  - Allerdings immer nur begrenzt

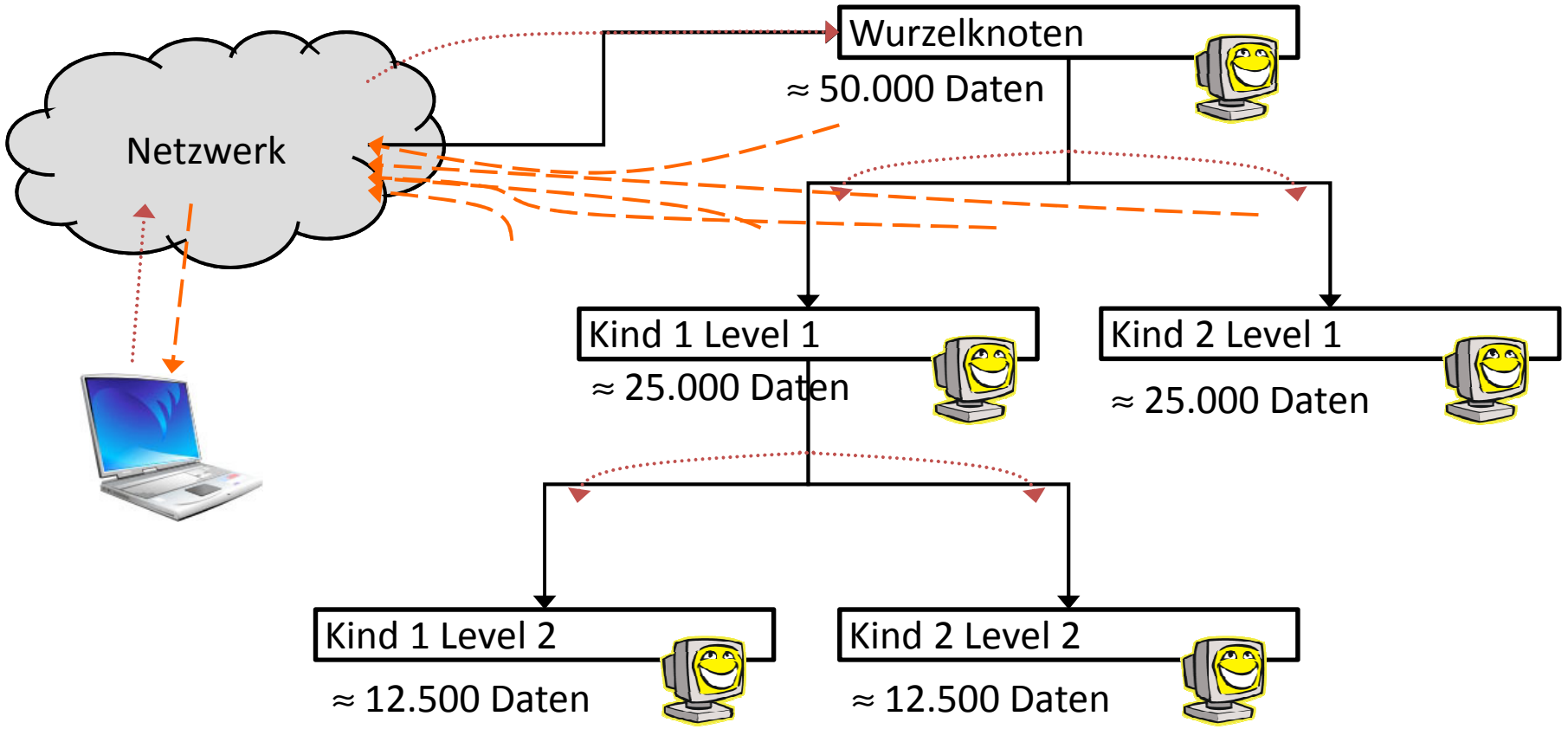
Techniken II: Power of Two Choices

- Idee: Verwende mehrere Hash-Funktionen h_1 bis h_d
- Speichern eines Datums mit Schlüssel k:
  - Kontaktiere parallel alle Peers $h_1(k)$ bis $h_d(k)$
  - Speichere das Datum auf dem am wenigsten ausgelasteten Peer
  - Speichere auf allen anderen Peers Verweise zum Datum
- Beim Abrufen eines Datums mit Schlüssel k:
  - Wähle ein i aus [1,d] zufällig
  - Kontaktiere Peer $h_i(k)$
- Es kann alles balanciert werden:
  Speicher, CPU, oder Bandbreite
- Alternative: keine Verweise speichern, sondern alle Fragen
- Wichtig:
  - **Weiterleitung muss billiger als selber machen sein**

Techniken III: Virtuelle Peers

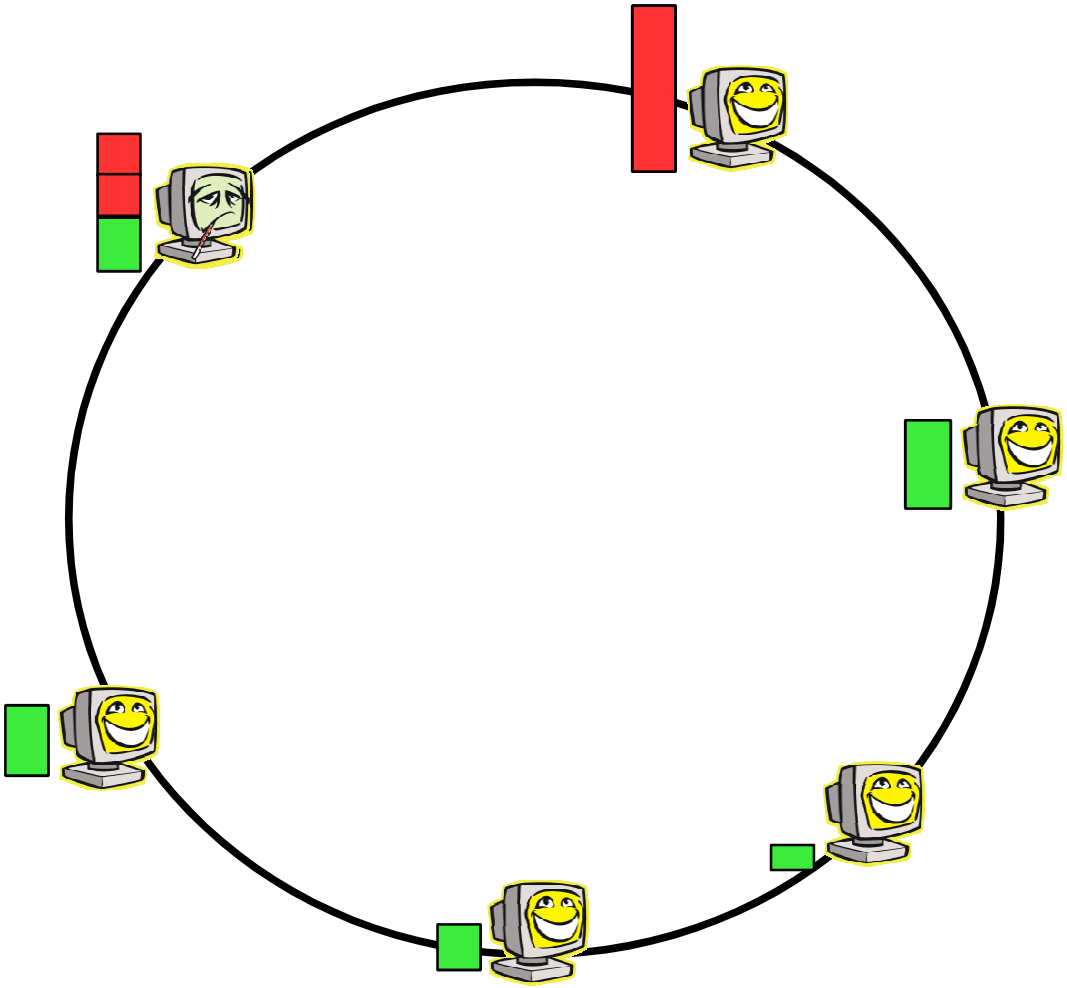
- Jeder physische Knoten hat mehrere virtuelle Peers
- Wenn Last zu hoch: Stoppe einige der Peers
- Wenn Last zu klein: Starte neue Peers
- Vorteile: einfach zu realisieren
- Nachteile:
  - Verwaltungsaufwand beim Starten / Stoppen
  - Netz wird virtuell größer und damit langsamer
  - Der Grad erhöht sich. Bei log(N) virtuellen Peers wird der Grad eines reellen Peer $\log^2(N)$
- Auch hier können individuelle Hot-Spots nicht aufgelöst werden

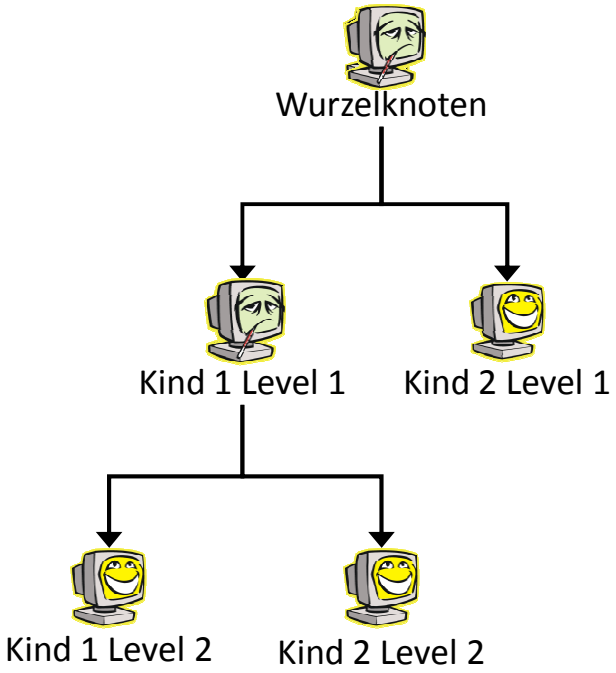
Techniken IV: Relokation



Quelle: André Höing

Techniken IV: Relokation





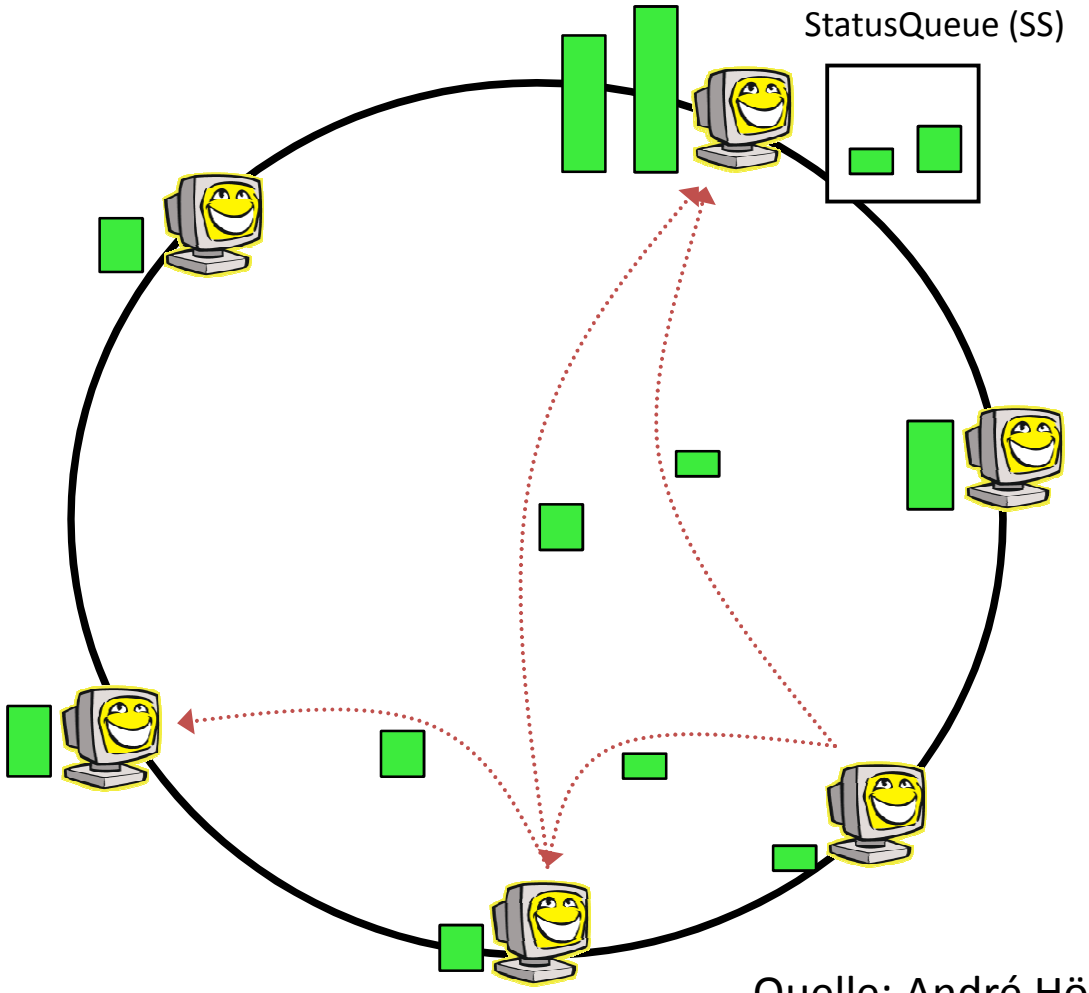
Quelle: André Höing

Lasterkennung I

- Grundsätzliches Problem:
  - Globale Informationen schätzen, ohne jeden Knoten zu fragen
- Woher weiß ein Knoten, ob er Überlast hat?
  - Gleichverteilung der ID-Bereiche erkennen:
  - Jeder Peer speichert 50 Keys zufällig im DHT
  - Wer >>50 Keys erhält, hat einen zu großen Bereich
  - Wer <<50 Keys erhält, hat einen zu kleinen Bereich

Lasterkennung II

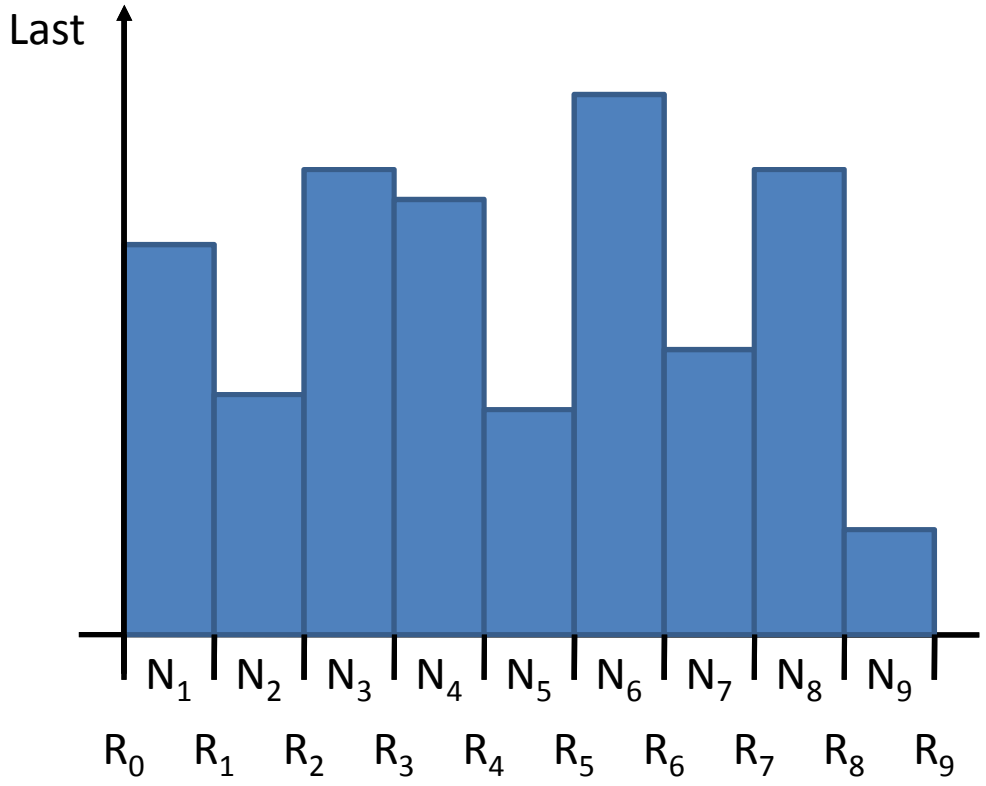
- Knoten versenden Lastinformationen an andere Knoten
  - Anzahl Daten
  - Verbrauchte Bandbr.
  - Mittlere Wartezeit
- SampleSize (SS)
  - Samplegröße
- SampleFactor (SF)
  - Multiplikator
- Überlastet, wenn
  - ***myLoad > SF * otherLoad***



Quelle: André Höing

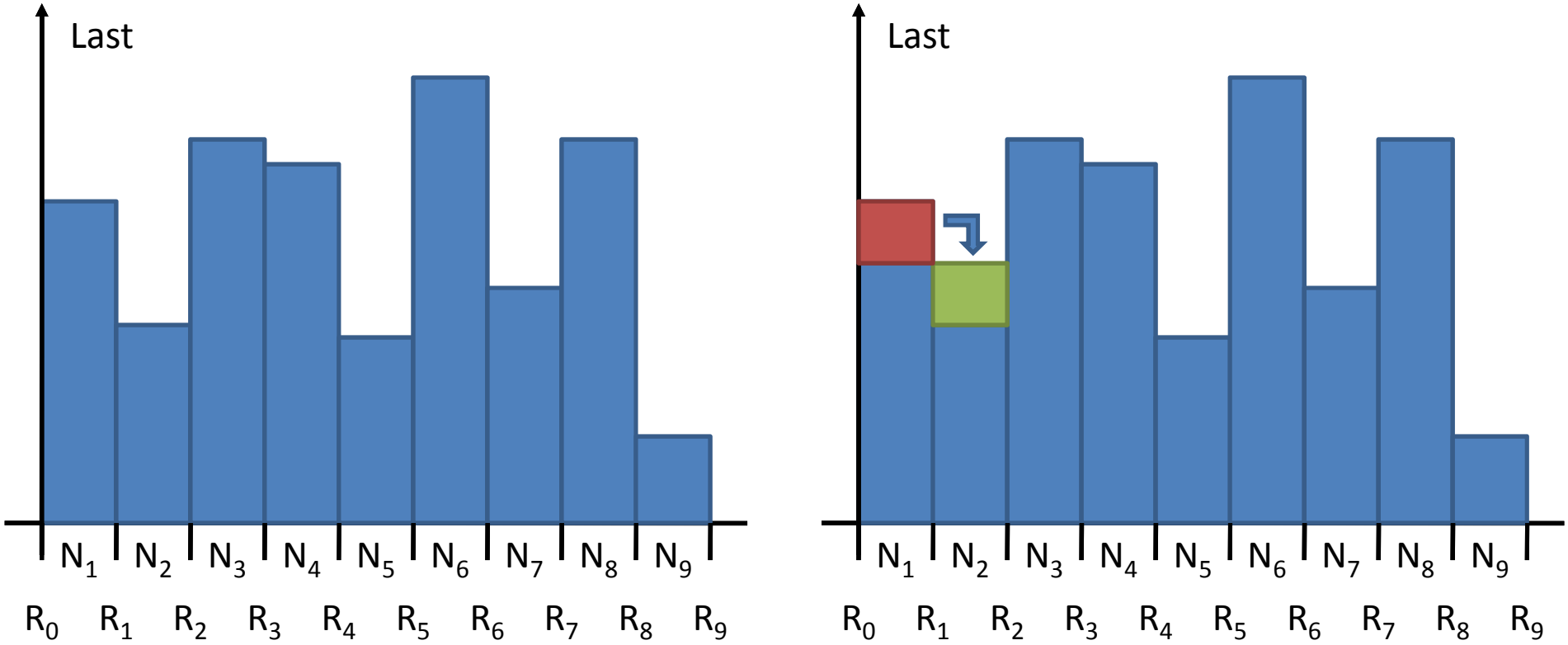
Lastbalancierung in SkipNet

- Knoten N_i ist für Daten im Bereich $[R_{i-1}, R_i)$ verantwortlich
- Die Bereiche haben beliebige Größe
- Wichtig ist, dass die Datenmenge ausgeglichen ist
- Die Bereichsgrenzen R_i werden ständig angepasst



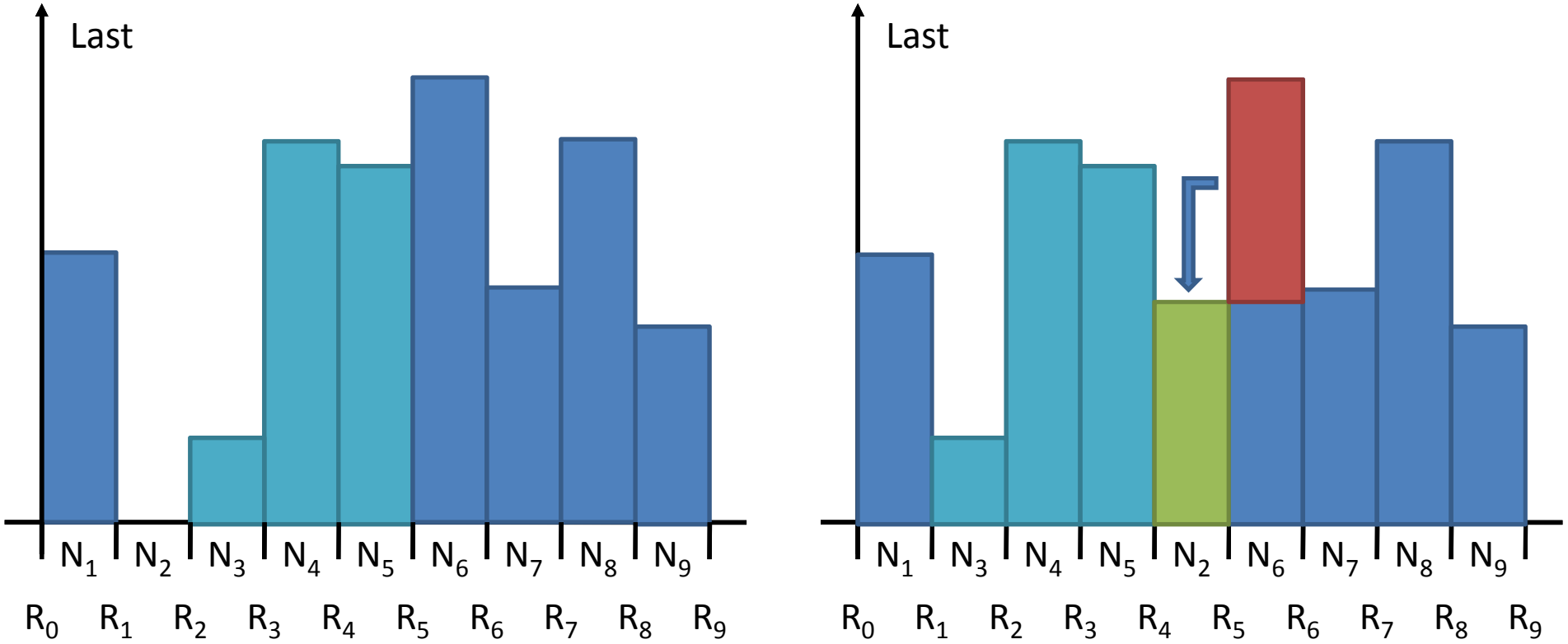
Lastbalancierung in SkipNet

- Operation **NeighborAdjust**:
- Zwei benachbarte Knoten passen ihre Bereichsgrenzen an



Lastbalancierung in SkipNet

- Operation **Reorder**:
- Ein leerer Knoten sucht sich eine neue Position



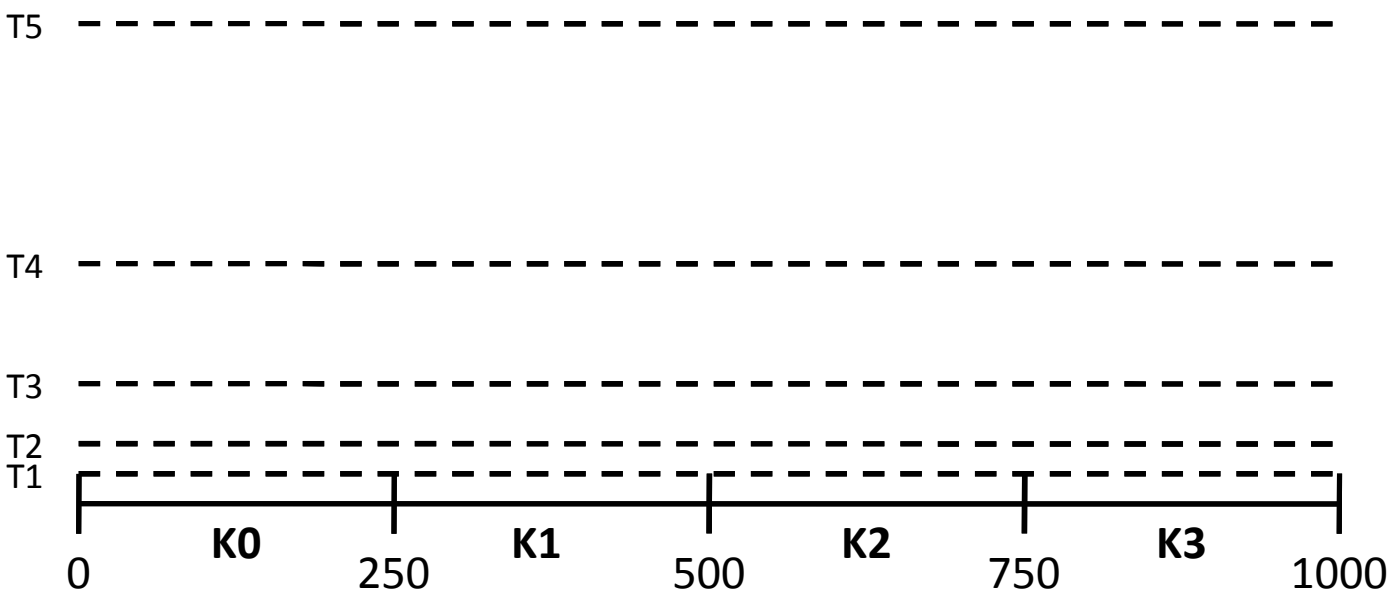
Verfahren

- Iteratives Vorgehen
- Wir starten mit leerem Netzwerk
- Bei jeder Insert- und jeder Delete-Operation wird eine Balancierung ausgeführt
- Betrachte Schwellenwerte:
  - $T_1=c$, $T_2=2c$, $T_3=4c$, $T_4=8c$, $T_5=16c$, etc.
  - Wenn die Last einen Knotens über einen Schwellwert kommt, wird Balancierung angestoßen

Algorithmus

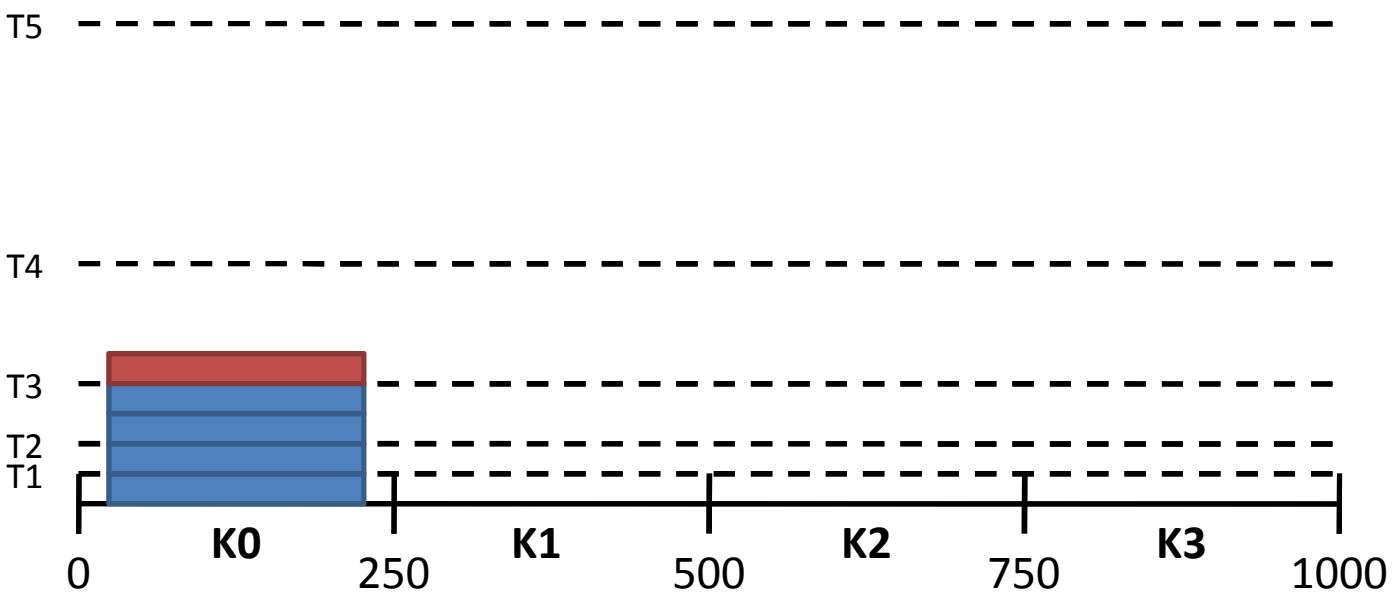
```
1    // Rufe nach insert eines Datums auf N_i auf
2    AdjustLoad(Node N_i)
3        Sei L(N_i) = x in (T_m, T_m+1)
4        Sei N_j der weniger belastete von N_i-1 und N_i+1
5        if L(N_j) <= T_m-1 then
6            NeighborAdjust(N_i ,N_j)
7            AdjustLoad(N_i)
8            AdjustLoad(N_j)
9        else
10           Suche den am wenigsten ausgelasteten Knoten N_k
11           if L(N_k) <= T_m-2 then  // weniger als ¼ der Last
12               Transferiere alle Daten von N_k zu N := N_k±1
13               Reorder(N_i ,N_k)
14               AdjustLoad(N)
15           else
16               // System ist ausreichend balanciert
15           end if
16       end if
```

Beispiel



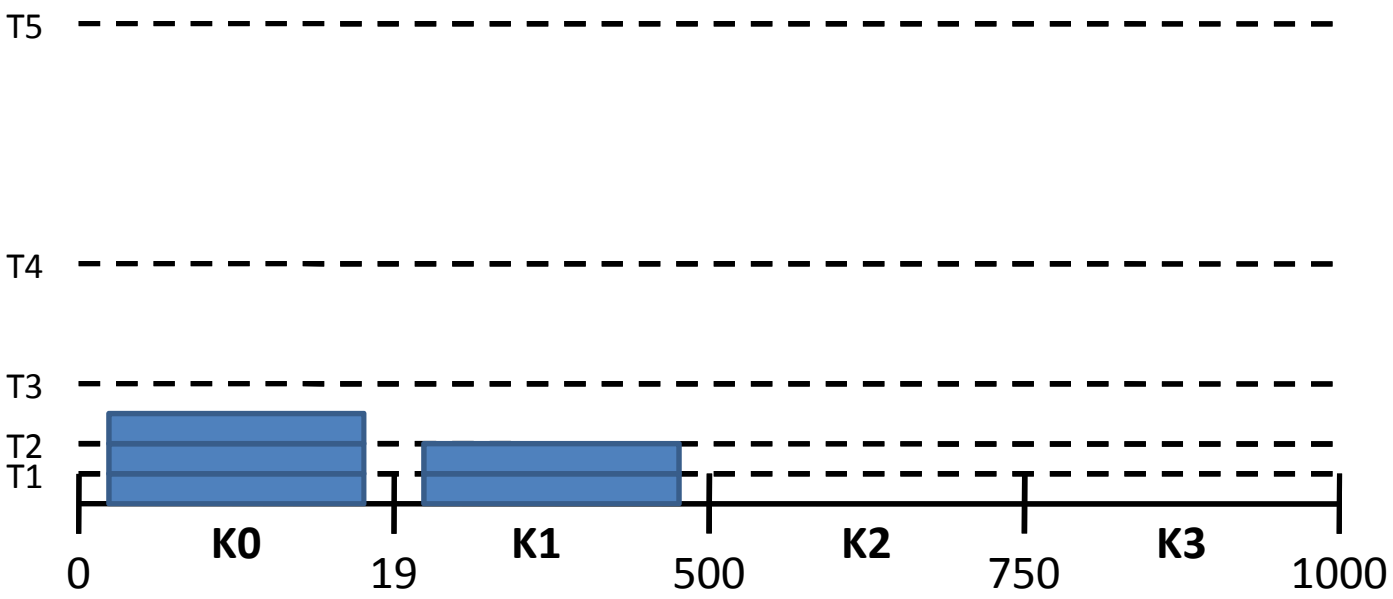
Beispiel

Einfügen: 9, 115, 1, 11, 19



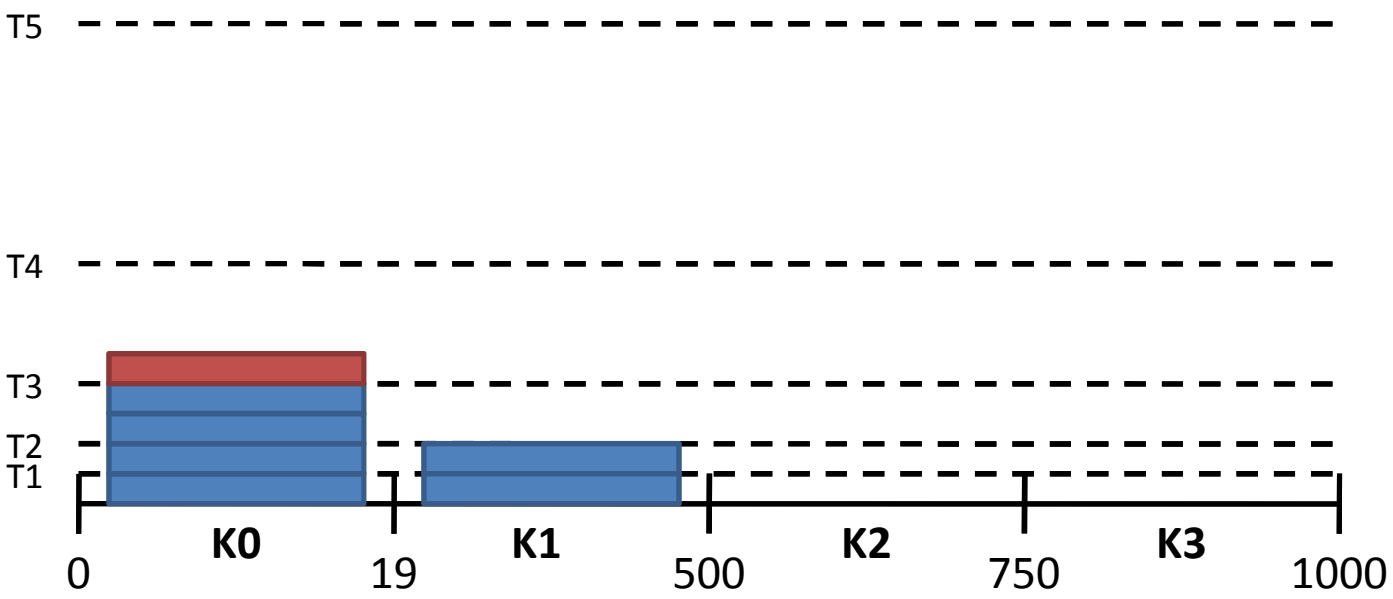
Beispiel

NeighborAdjust



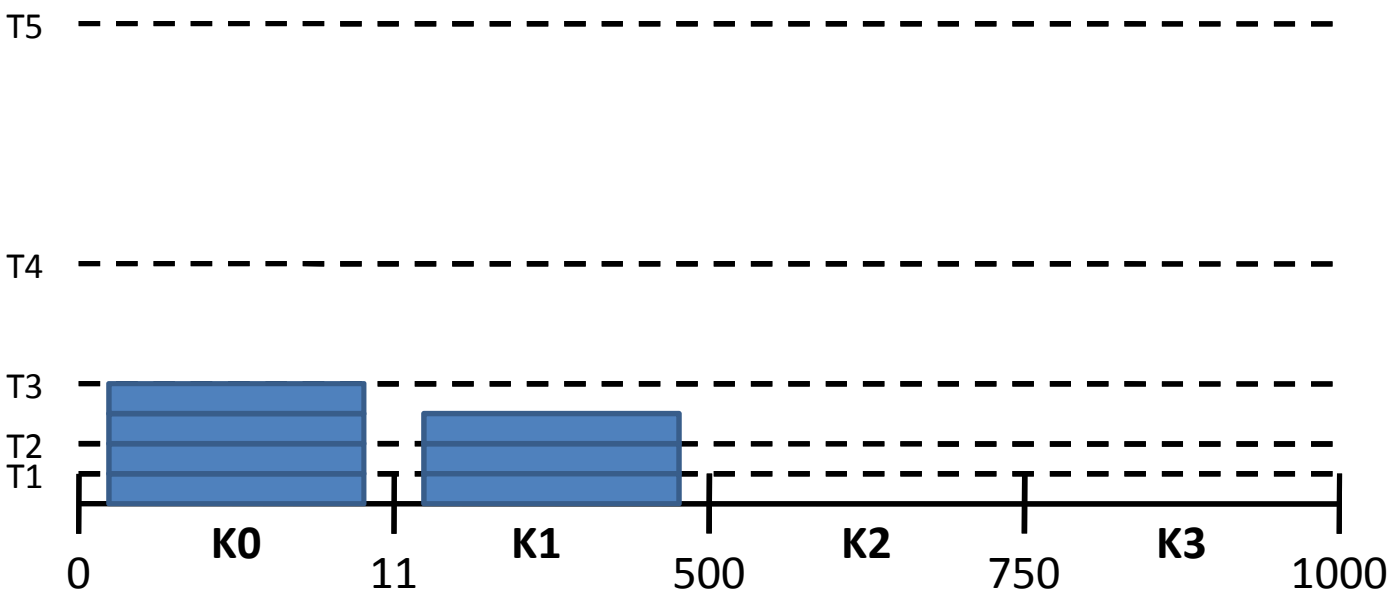
Beispiel

Einfügen: 0, 2



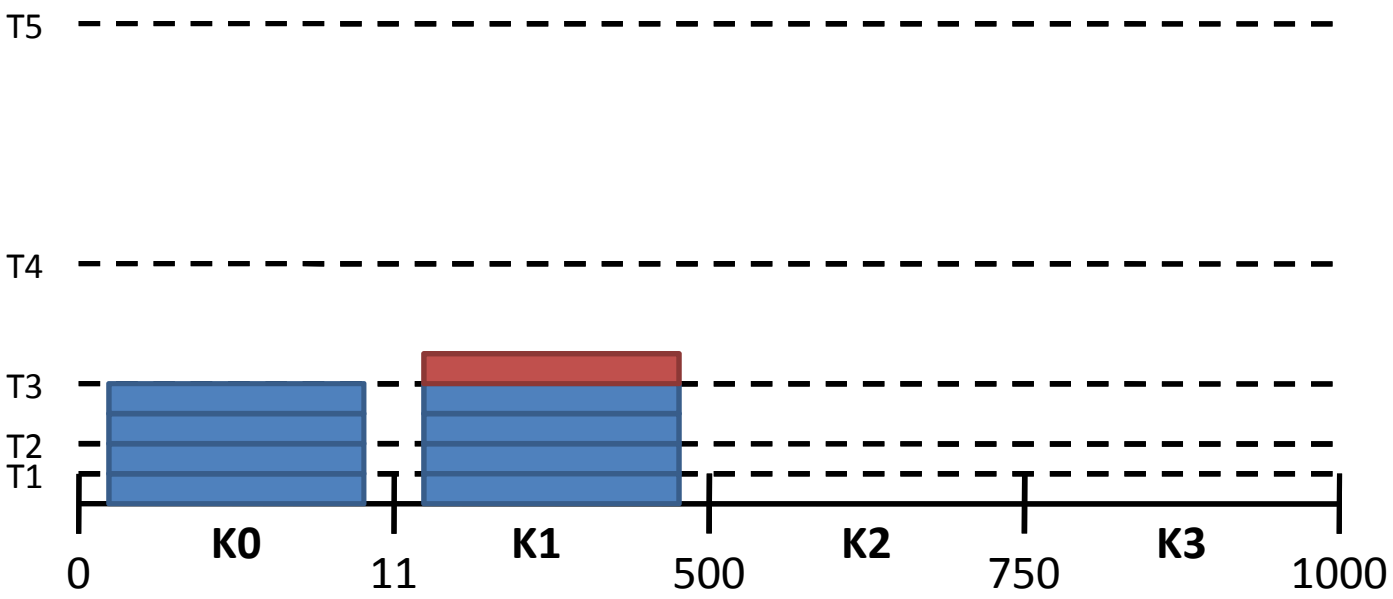
Beispiel

NeighborAdjust



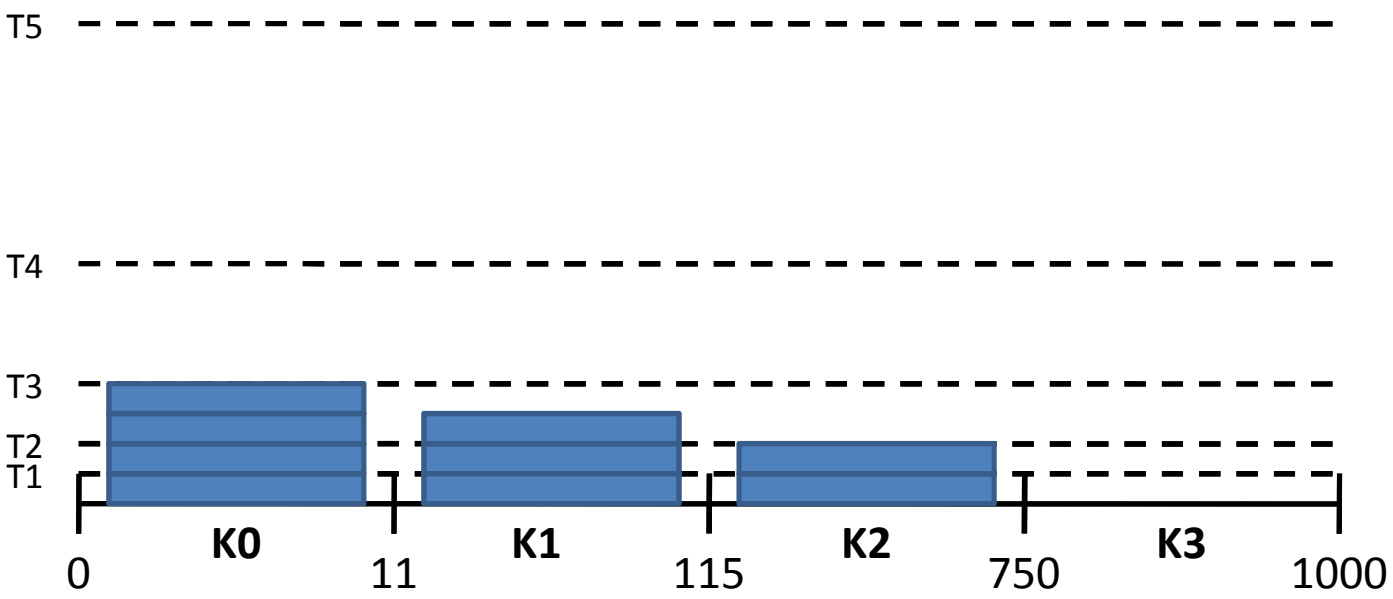
Beispiel

Einfügen: 281, 53



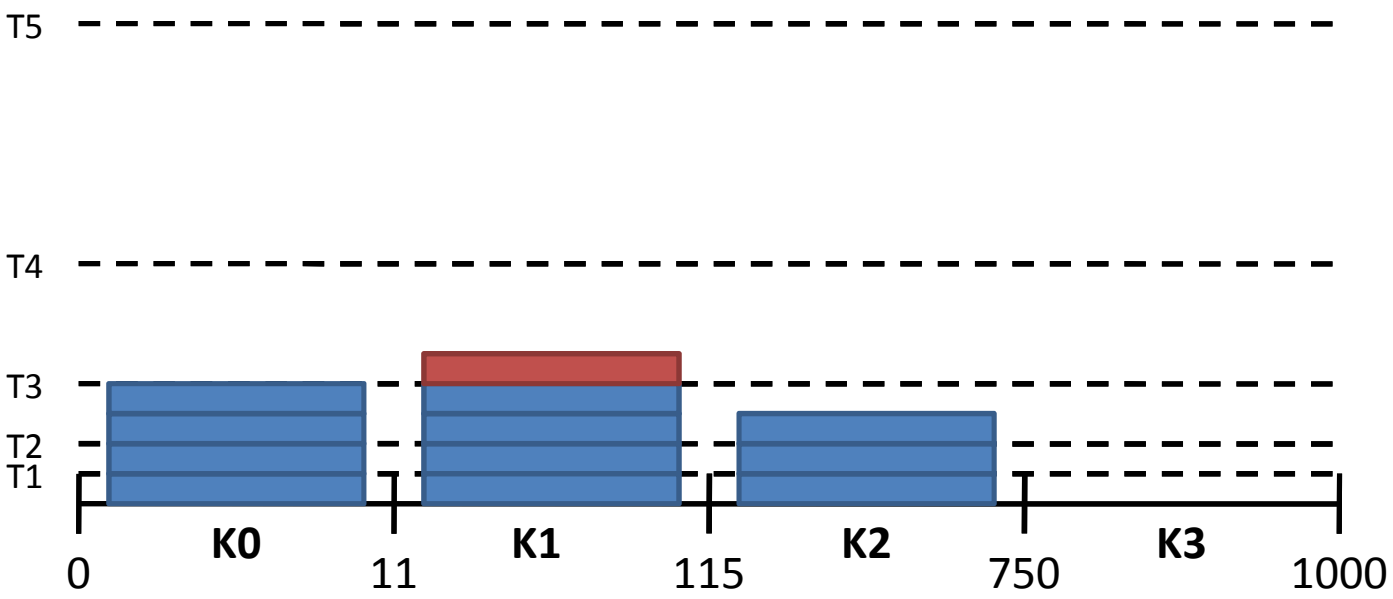
Beispiel

NeighborAdjust



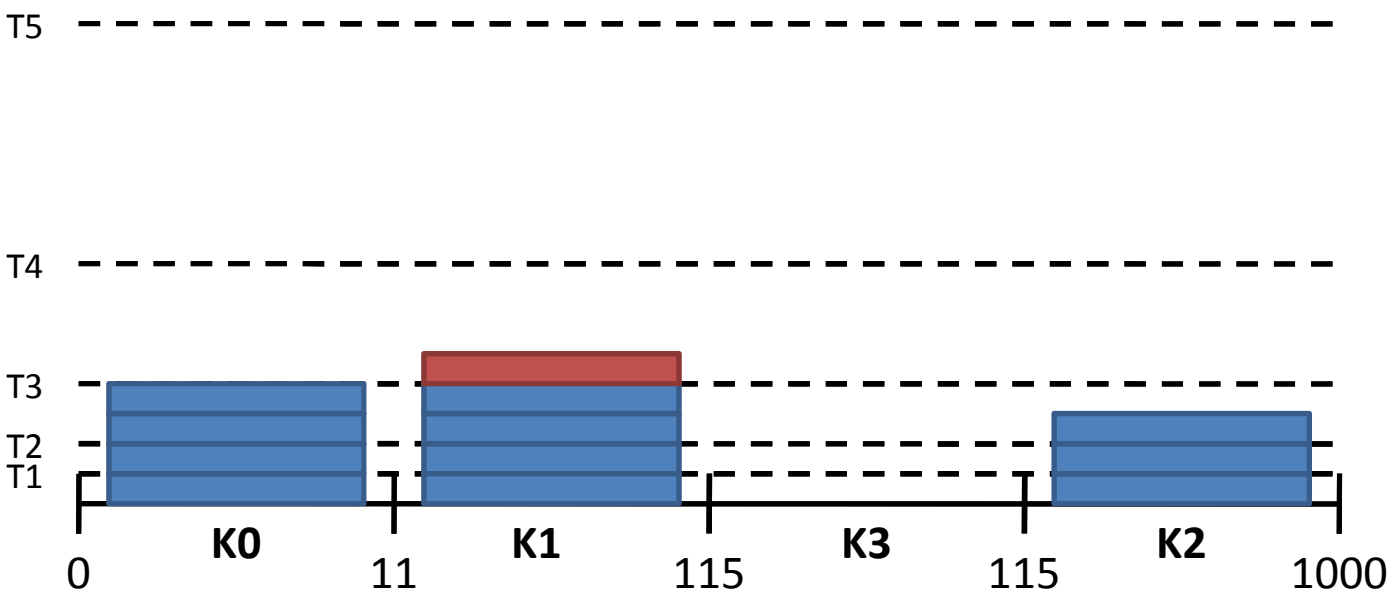
Beispiel

Einfügen: 13, 432, 40



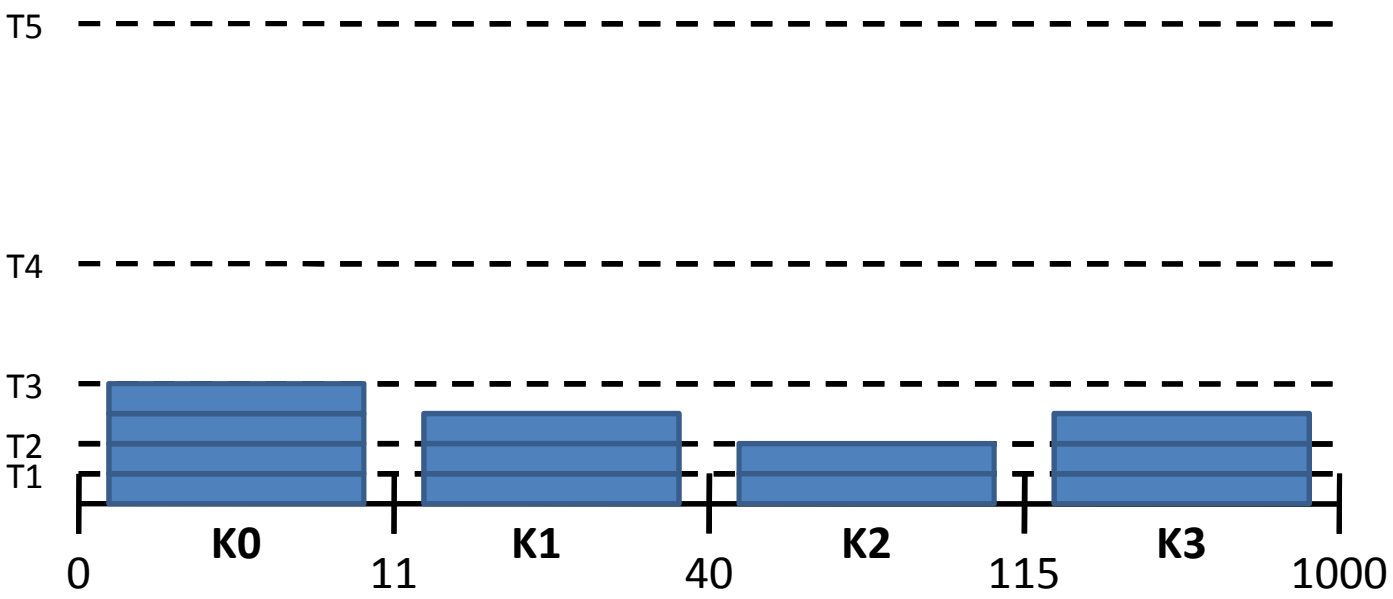
Beispiel

Reorder



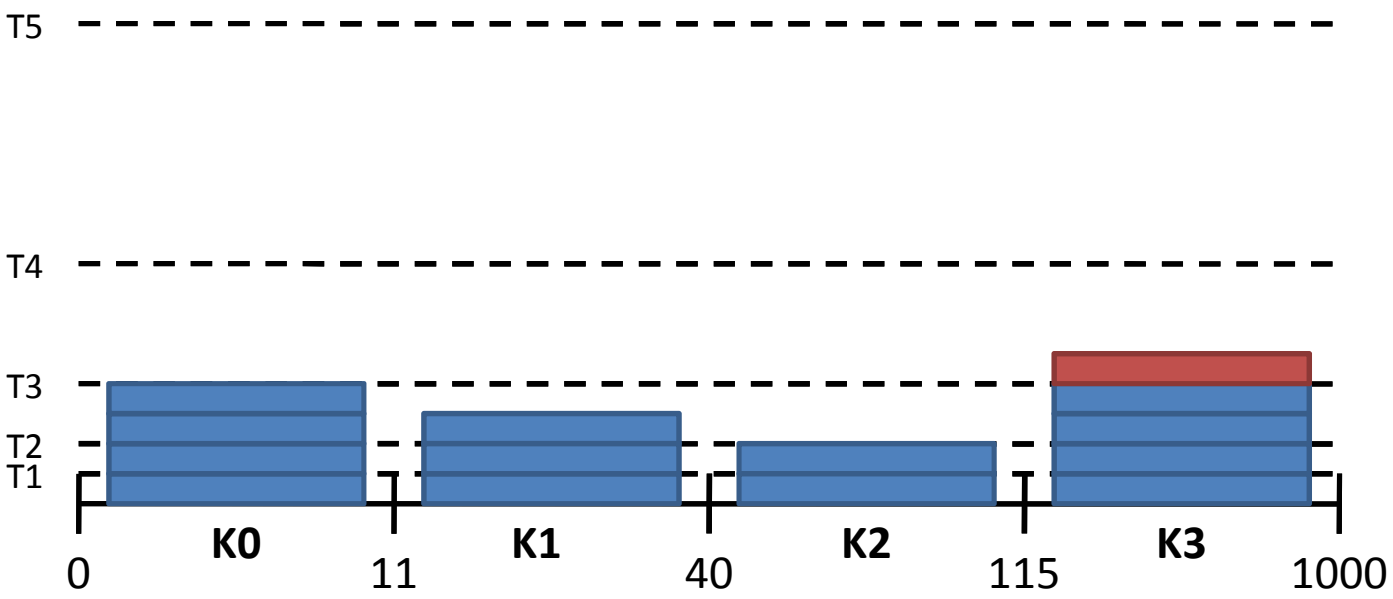
Beispiel

NeighborAdjust



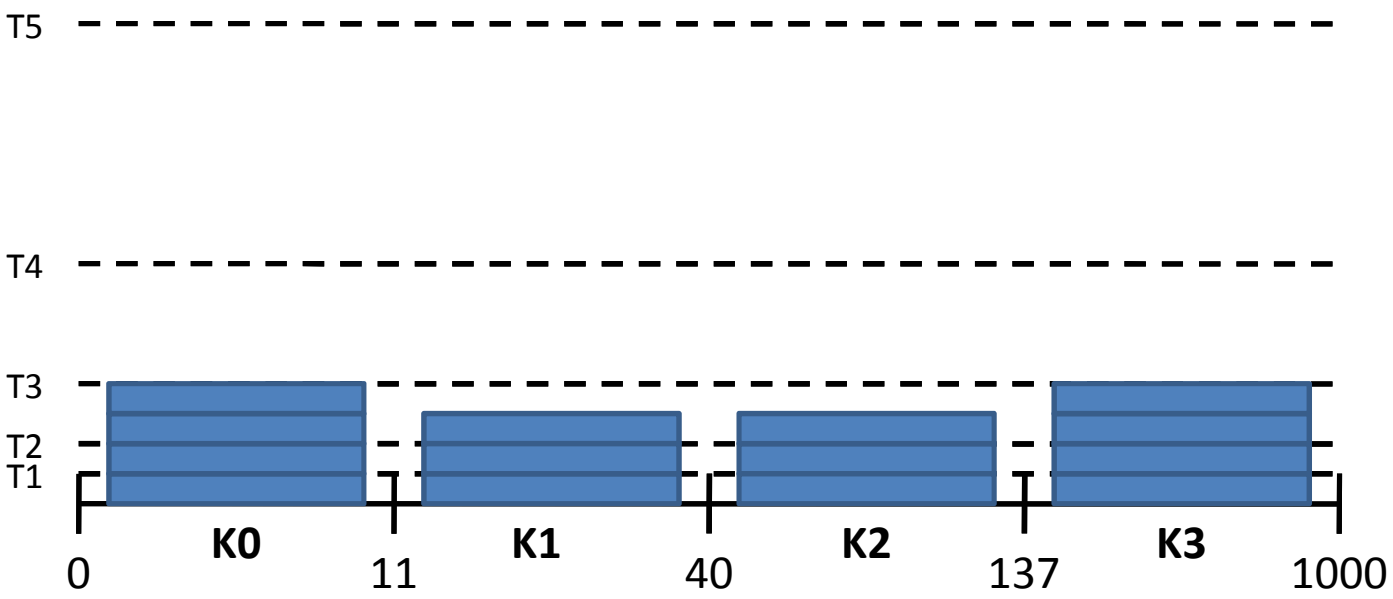
Beispiel

Einfügen: 137, 205



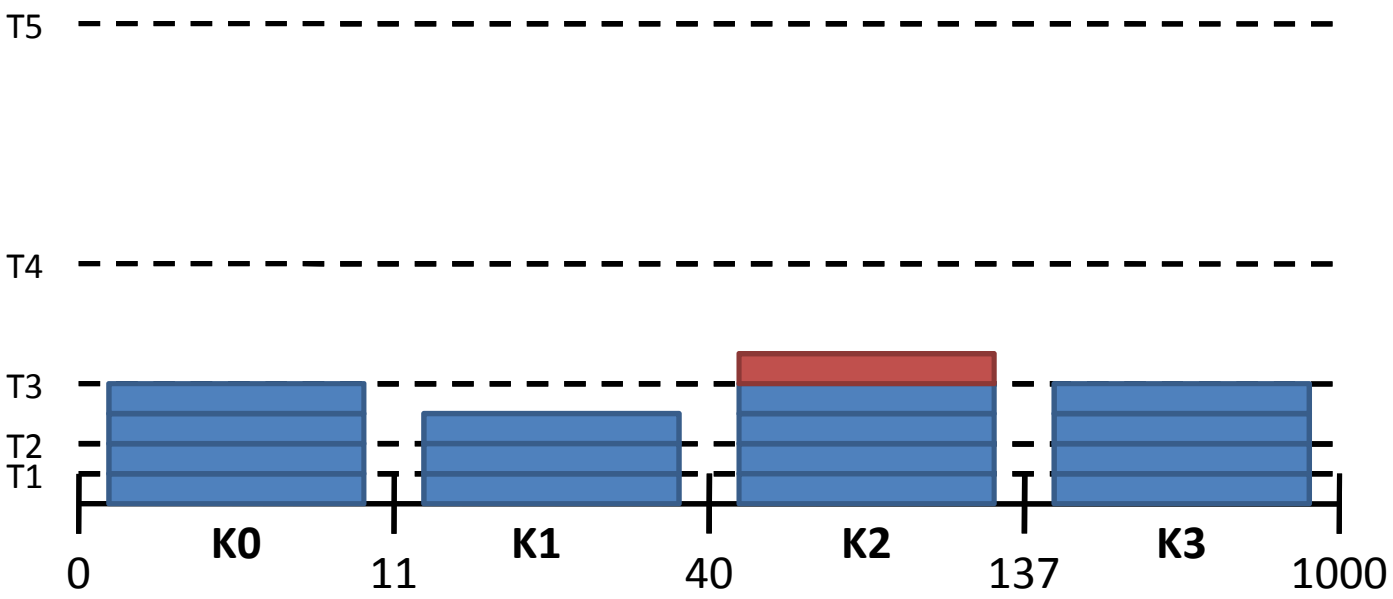
Beispiel

Einfügen: 137, 205



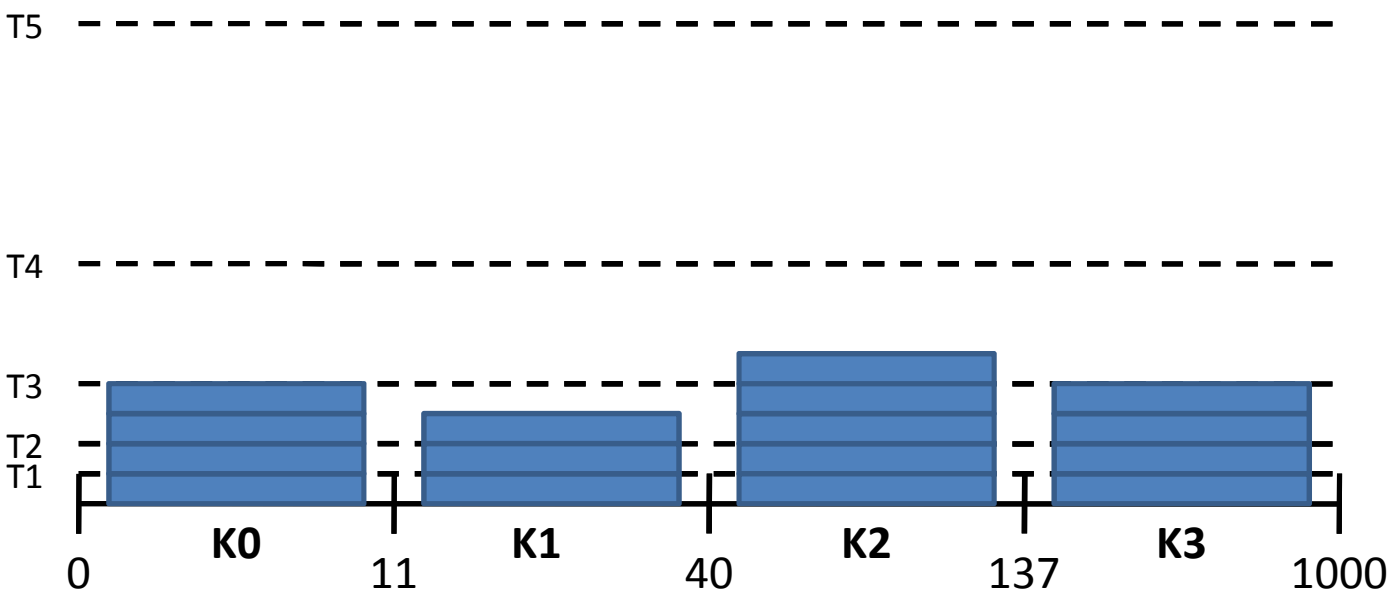
Beispiel

Einfügen: 205, 82, 130



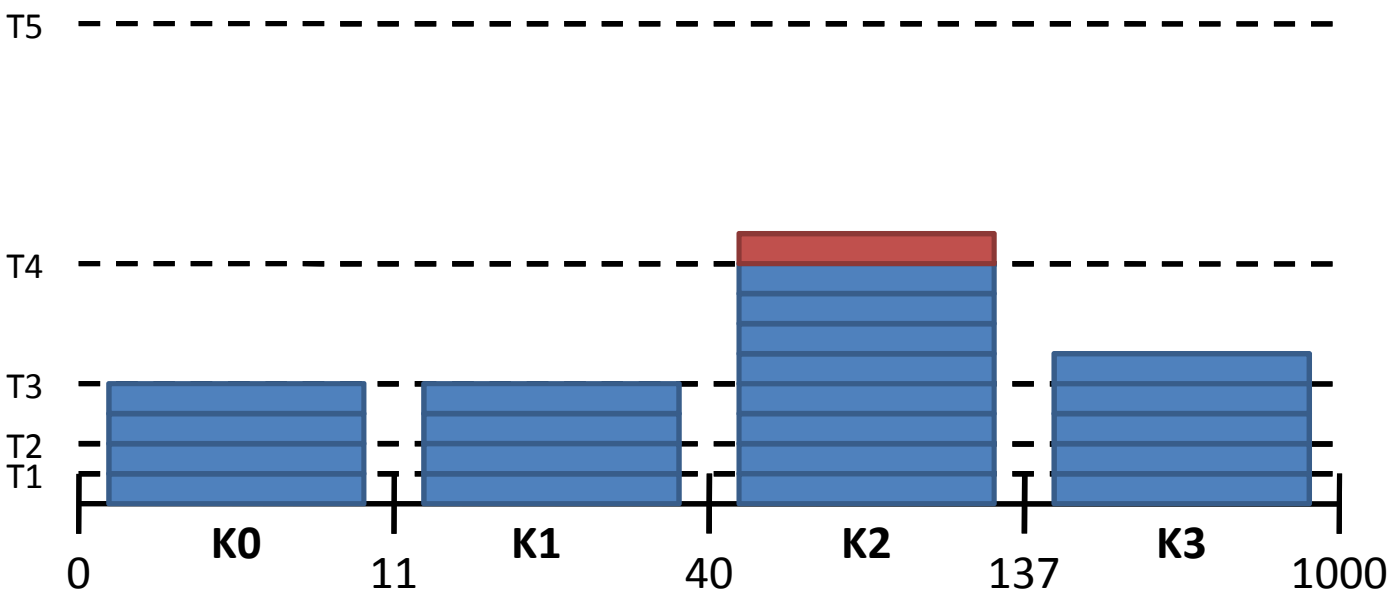
Beispiel

Keine Balancierung!



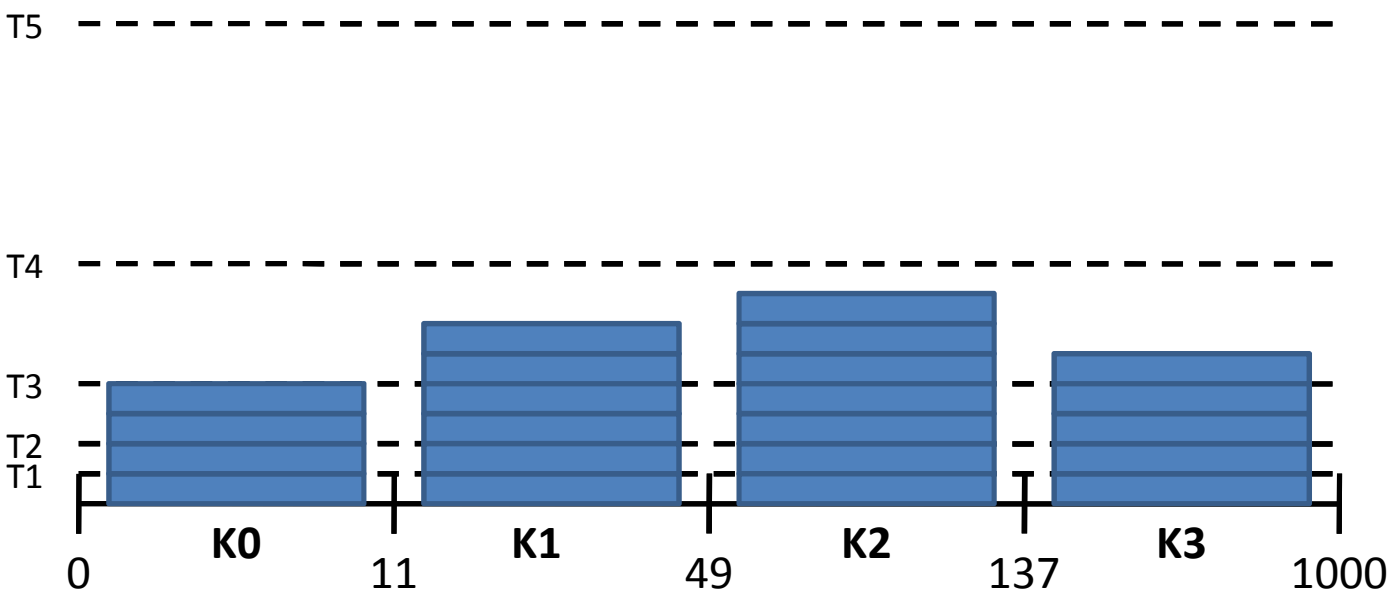
Beispiel

Einfügen: 26, 78, 41, 110, 679, 49:



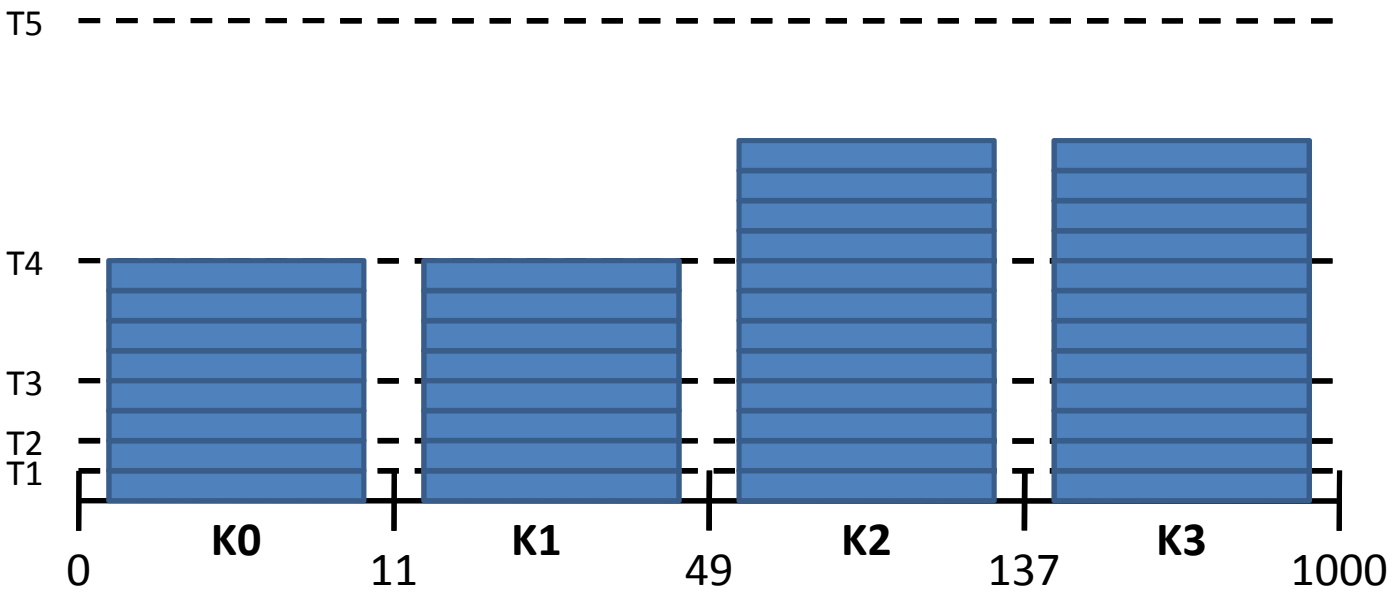
Beispiel

NeighborAdjust



Beispiel

Einfügen: 7, 69, 163, 197, 506, 6, 3, 8, 182, 173, 416, 133, 42,
108, 18, 56, 556, 123



Lastbalancierung in SkipNet

- Durch lokale Entscheidungen wird globale Balancierung erreicht
- Durch Anpassen der Bereichsgrenzen wird neue Last passend einsortiert
  - Solange das Lastprofil gleich bleibt!
- Beweisbar gutes Verfahren
  - Kosten sind max. O(X) bei X eingefügten Daten
  - Garantiert, dass $\min_i \{L(N_i)\} \leq 8 * \max_i \{L(N_i)\} + c_0$
- Problem beim P2P-Einsatz: Wie finde ich den am wenigsten belasteten Knoten?
  - Lösung: Zweite SkipGraph-Struktur, nach Last sortiert

Zusammenfassung

- Last-Arten: CPU, Speicher, Bandbreite
- Automatische Balancierung von DHTs
- Techniken:
  - I: Bereichsgrenzen ändern, Replikation
  - II: Power of Two Choices
  - III: Virtuelle Peers
  - IV: Relokation
- Erkennung von Ungleichgewichten
- Lastbalancierung in SkipNet nach Ganesan et al.

Vorlesung
P2P Netzwerke
9: Anwendungen: OceanStore

Dr. Felix Heine

Complex and Distributed IT-Systems

felix.heine@tu-berlin.de

Inhalt

- Daten-Orientierte Anwendungen
- Anforderungen
- Beispiel: OceanStore

Daten-Orientierte Anwendungen

- Welche Daten?
  - Dateien
    - beliebiger Inhalt
    - Multimedia, Textdateien, HTML-Seiten
  - Strukturierte Daten:
    - Relationale Daten, XML-Daten
    - Daten aus dem Semantic Web (RDF, etc.)
- Welche Abfragen?
  - Volltextsuche
  - Schlagwortsuche
  - Komplexe Abfragen (SQL, etc.)
- Veränderungen?
  - Nur der Original-Knoten
  - Jeder Knoten → Konsistenz nach Updates

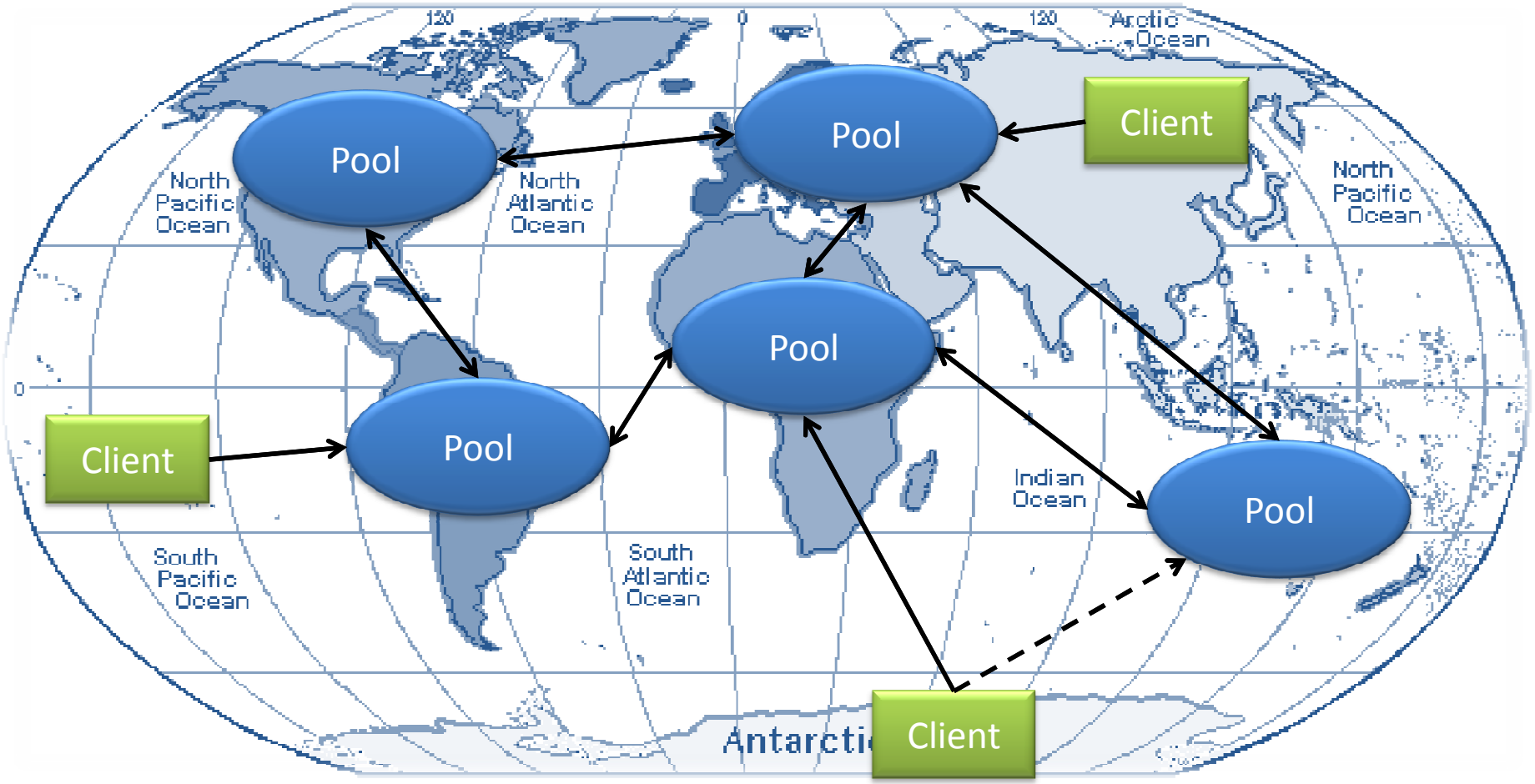
Anforderungen

- Universeller Zugriff, Verfügbarkeit
  - egal wo ich mich befinde, ich möchte auf alle Daten zugreifen können
- Privater vs. öffentlicher Zugriff
  - genaue Definition der Lese / Schreibrechte auf bestimmte Daten
- Integrität
  - Es wird genau das zurückgeliefert, was gespeichert wurde
- Konsistenz
  - Simultane Updates auf Daten werden serialisiert (vgl. Datenbanken)
- Performance: Read-Performance vs. Write-Performance
- Automatisches Management

OceanStore: Literatur

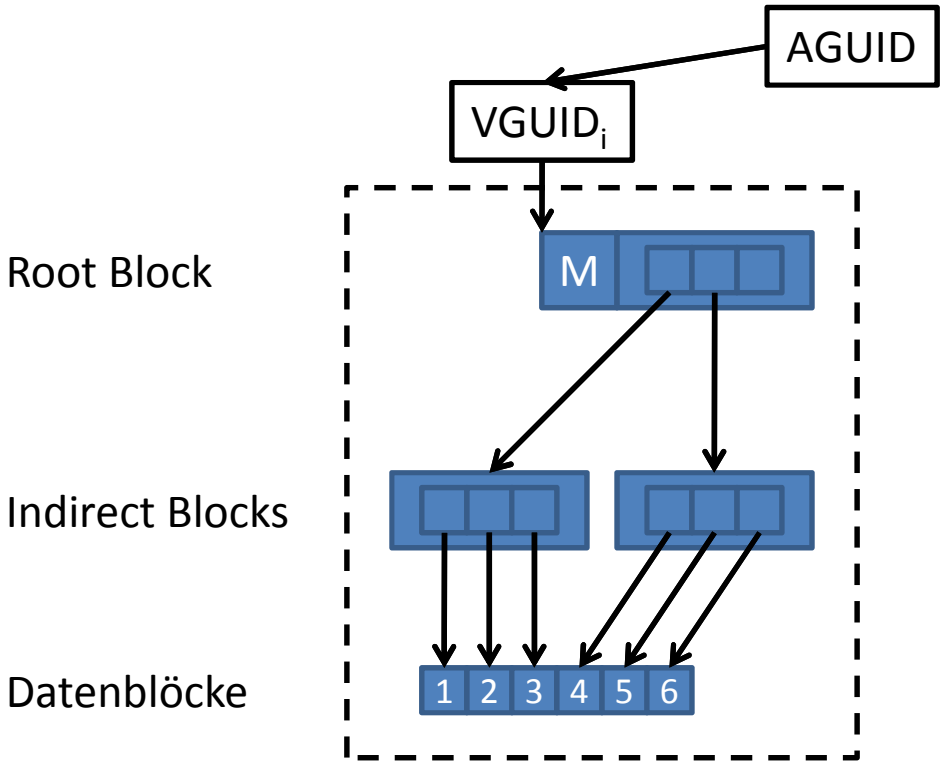
- John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, Ben Zhao: "OceanStore: An Architecture for Global-Scale Persistent Storage", Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), November 2000
- Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, John Kubiatowicz: "Pond: the OceanStore Prototype", Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03), March 2003
- http://oceanstore.cs.berkeley.edu

Ocean-Store Modell



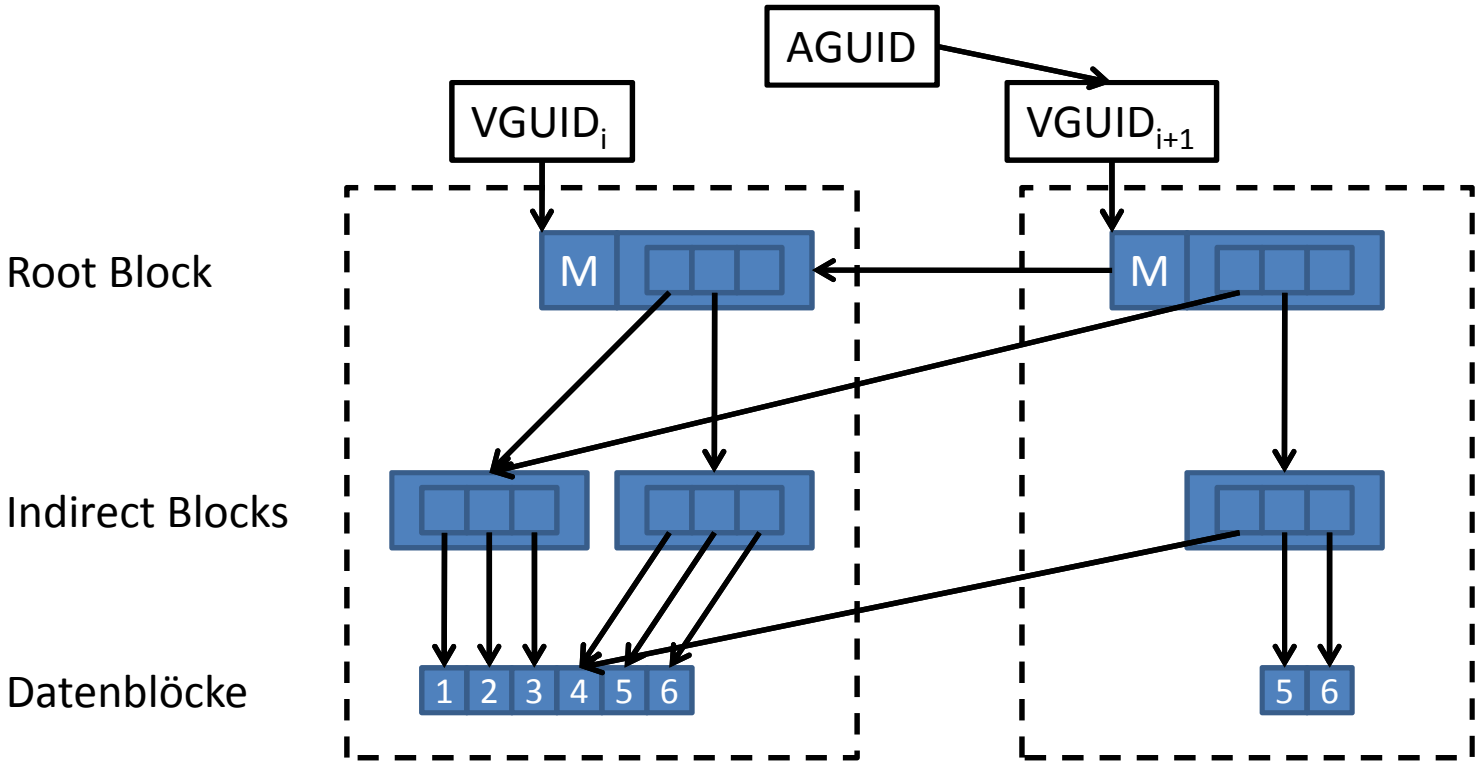
Datenobjekte in OceanStore

- Es werden "Datenobjekte" gespeichert – also Dateien
- Jede Version eines Datenobjektes wird für immer archiviert
- Ein Objekt wird in Blöcke aufgeteilt:



Datenobjekte in OceanStore

- Es werden "Datenobjekte" gespeichert – also Dateien
- Jede Version eines Datenobjektes wird für immer archiviert
- Ein Objekt wird in Blöcke aufgeteilt:



GUIDs in OceanStore

- GUID's werden zur Identifikation und zum Routen verwendet:
  - ActiveGUID:
    - Identifiziert ein gesamtes Datenobjekt
    - Berechnet als SHA des Objektnames + Public Key des Eigentümers
  - VersionGUID:
    - Identifiziert eine Version eines Datenobjektes
    - SHA des Blockinhaltes
  - BlockGUID:
    - Identifiziert einen Block eines Datenobjektes
    - Ebenfalls SHA des Blockinhaltes
- Heartbeat: Mapping der AGUID auf die VGUID

Updates in OceanStore

- Bestehende Versionen sind read-only
- Ein Update führt immer zu einer neuen Version
- Betrachte:
  - Mehrere Nutzer führen gleichzeitig Updates durch
  - Wie verhindere ich Konflikte?
- Lösung:
  - Ein Update besteht aus "Prädikaten" und "Aktionen"
  - Die Aktionen werden nur ausgeführt, wenn die Prädikate "wahr" ergeben
  - Mögliche Prädikate: compare-version, compare-block
  - Mögliche Aktionen: replace-block, insert-block, delete-block, append-block

Der innere Ring

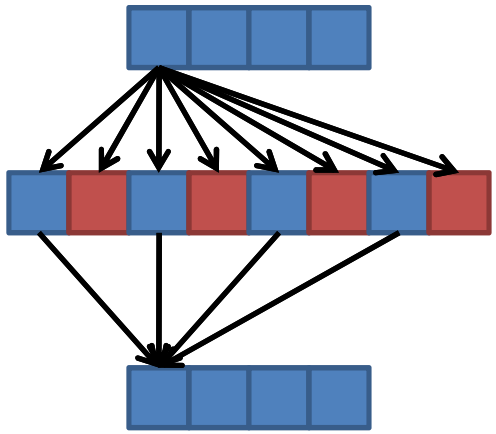
- Problem: Jeder Replika-Knoten muss die Updates in der gleichen Reihenfolge anwenden
- Ideal: Ein primärer Knoten bekommt alle Updates und serialisiert sie, dann Weiterverteilung an sekundäre Knoten
- Problem: Ausfallsicherheit, Abhängigkeit von primärem Knoten
- Lösung: "Inner Ring", d.h. kleine Anzahl primärer Replika-Knoten
- Alle Knoten des inneren Rings kennen sich untereinander
- Diese Knoten nutzen ein Abstimmungs-Protokoll zur Bestimmung der Reihenfolge
- Das Ergebnis wird digital signiert

Proactive Threshold Signatures

- Digitale Signatur:
  - Paar aus öffentlichem Schlüssel und privatem Schlüssel
  - Verschlüssele Nachricht mit privatem Schlüssel
  - Kann anschließend nur mit öffentlichem Schlüssel dekodiert werden
- Threshold Signatur:
  - Paar aus einem öffentlichen Schlüssel und l privaten Schlüssel-Anteilen
  - Mit jedem Schlüssel-Anteil wird ein Signatur-Anteil generiert
  - Aus k Signatur-Anteilen kann eine vollständige Signatur generiert werden
- Proactive Threshold Signatur:
  - Es kann jederzeit ein neuer Satz von l privaten Schlüsselanteilen generiert werden (wenn der innere Ring geändert wird)
- Wir nehmen an, dass max. f Server fehlerhaft sind
- Setze $l = 3f+1$, $k=f+1$, Bsp: $l=4$, $k=2$
- Eine korrekte Signatur beweist die Übereinstimmung der Knoten des Inner Ring

OceanStore: Archivierung

- Jeder Datenblock soll sicher und dauerhaft gespeichert werden
- Dafür ist eine hohe Anzahl Replikas nötig
- Daher bessere Lösung: Erasure-Coding
- Fehler-Korrigierende Codes, die folgendes ermöglichen:
  - Teile einen Datenblock in n Teile auf
  - Kodiere diese n Blöcke als n+k Blöcke (Redundanz)
  - Beliebige n dieser n+k Blöcke genügen zur Rekonstruktion



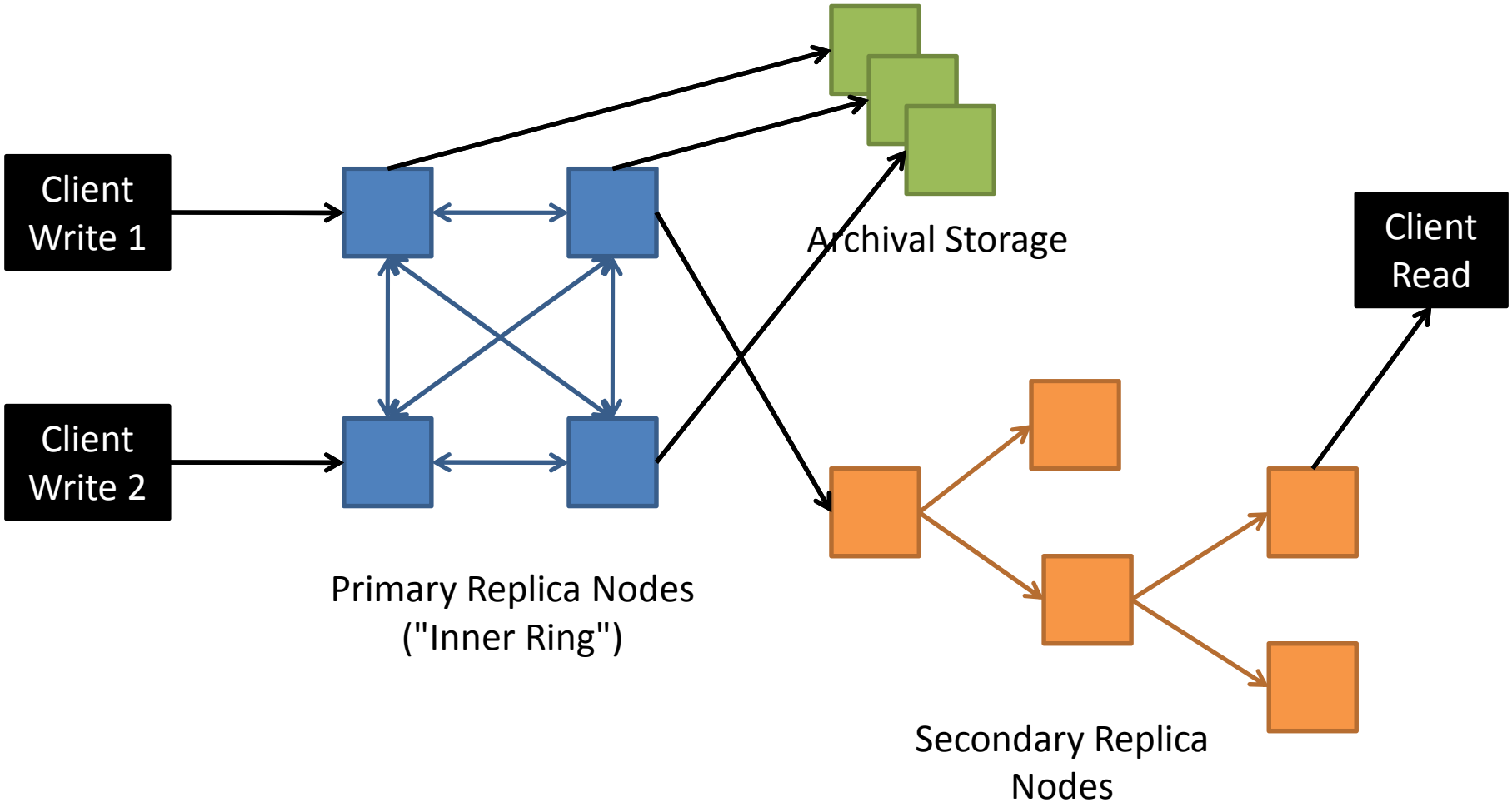
Exkurs: Reed-Solomon Codes

- Betrachte Symbole der Nachricht als Elemente des Restklassenrings $Z_p := \{ 0, \ldots, p-1 \}$
- Ein Datenblock aus n Symbolen $a_0, \ldots, a_{n-1}$ (a_i aus Z_p) wird als Polynom aufgefasst:
  $a(X) := a_{n-1}X^{n-1} + \ldots + a_2X^2 + a_1X + a_0$
- Dazu werden m Stützstellen u_0 bis u_{m-1} fest gewählt.
- Die Kodierung der Nachricht ist die Auswertung des Polynoms a(X) an den Stützstellen:
  $a(u_0), \ldots, a(u_{m-1})$
- Wenn bis max. (m-n) Zeichen der Kodierung fehlen, kann die Nachricht durch Interpolation zurückgewonnen werden
- Anm: Es können max. (m-n)/2 Fehler dekodiert werden

OceanStore: Archivierung

- Problem: Rekonstruktion der Datenblöcke aus den Codes ist aufwändig
- Daher unterteilt OceanStore
  - Archiv-Blöcke sind nur für dauerhafte Archivierung
  - Sekundäre Replika-Knoten sind für schnellen Zugriff und kodieren nicht
- Sekundäre Replika-Knoten werden durch Multicast von Änderungen informiert

Updates in OceanStore



Zusammenfassung

- OceanStore:
  - Komplexes System; Ideen-Schatzkiste
  - Nicht vollständig beschrieben
  - Idee der Erasure-Codes wurde vielfach wiederverwendet (DHash, PAST)
- Kernpunkte:
  - Datenobjekte in Blöcke aufteilen, Versionieren
  - Update-Konsistenz gewährleisten durch Serialisierung im inneren Ring
  - Archivierung mit Erasure Codes
  - Vollständige Virtualisierung

Vorlesung
P2P Netzwerke

10: BabelPeers

Dr. Felix Heine

Complex and Distributed IT-Systems

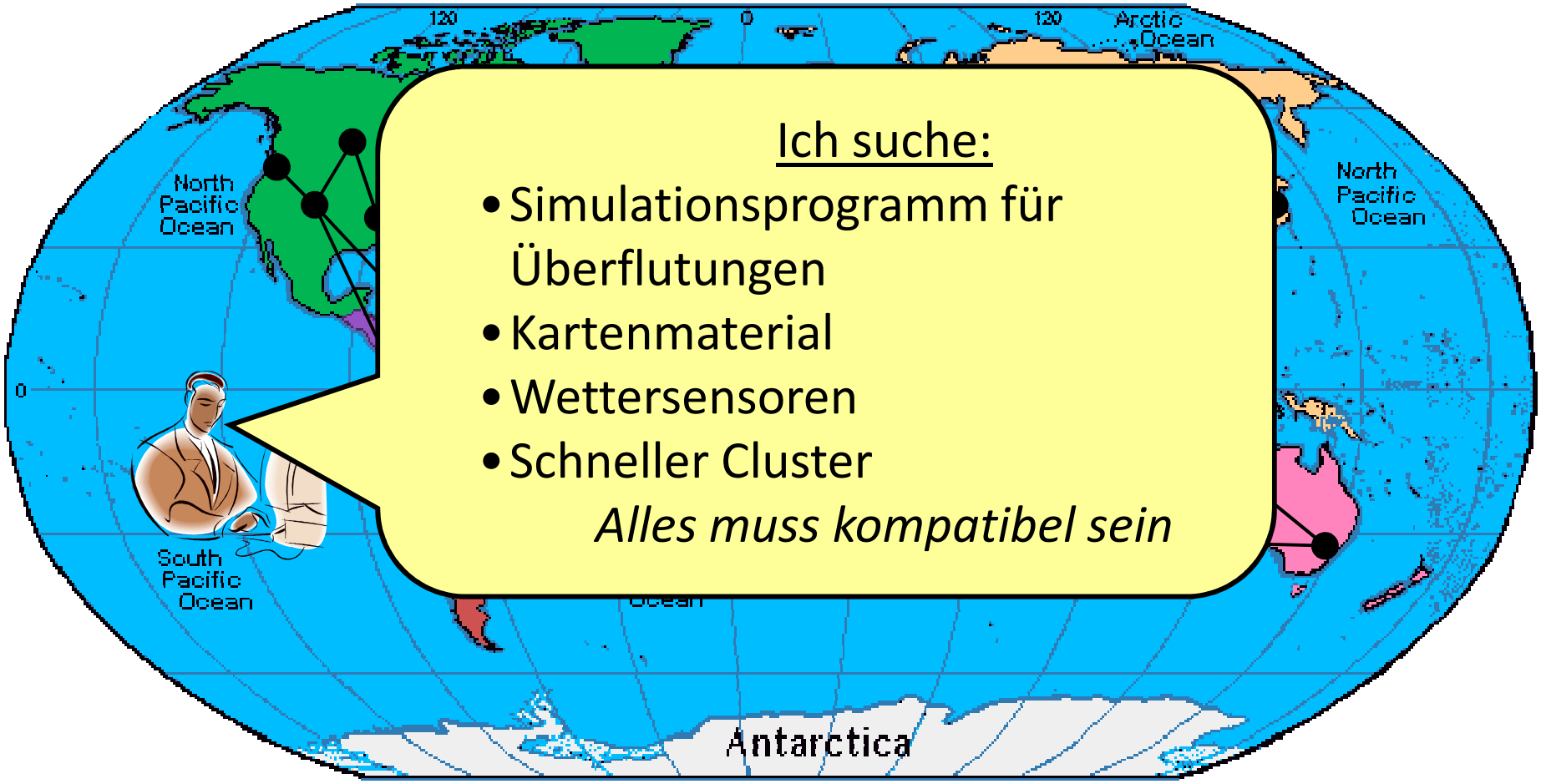
felix.heine@tu-berlin.de

BabelPeers

- Forschungsprojekt der TU Berlin im Rahmen des EU-Projektes "DELIS"
- Literatur:
  - Felix Heine: "Scalable P2P based RDF Querying", First International Conference on Scalable Information Systems (INFOSCALE'06), ACM Press, 2006
  - Dominic Battré, Felix Heine, Odej Kao: "Top k RDF Query Evaluation in Structured P2P Networks", Euro-Par 2006 Parallel Processing Conference, Springer, 2006

Motivation: Suche nach Ressourcen

- Vision: weltweites Grid mit vielfältigen Ressourcen



Motivation: Suche nach Ressourcen

Biete **Programm SimFlood**
SimFlood läuft ab Pentium
SimFlood benötigt GeoInfo Karten

Cluster:
- Itanium 2 Prozessoren
- …

Hardwareinfos:
- Itanium ist kompatibel zu Pentium
- Itanium 2 ist kompatibel zu Itanium

Ich bin ein Dienst der **Karten** anbietet im Format GeoInfo

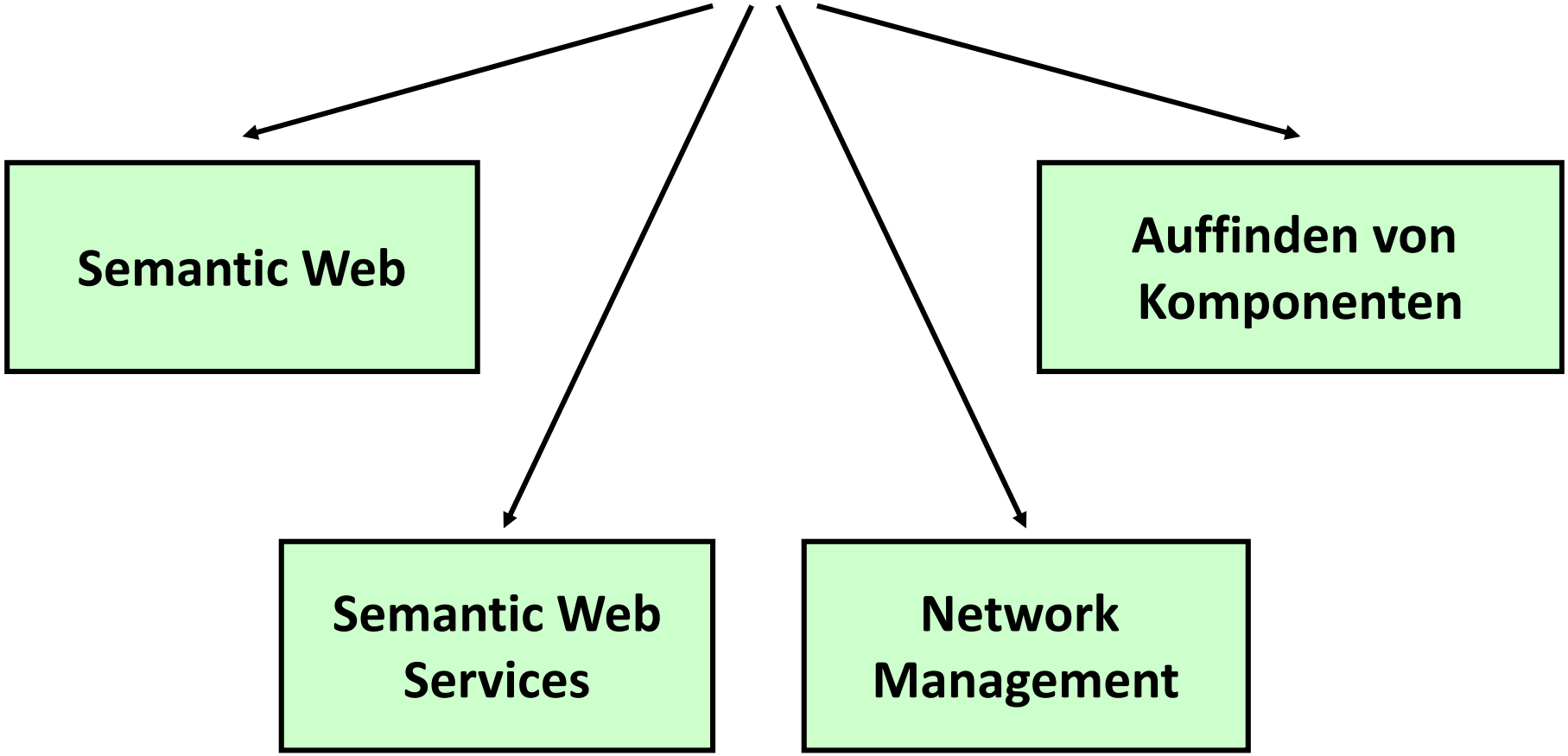
Netz von Sensoren
- Für Wetterdaten in Europa

Ziele von BabelPeers

- Verteiltes System zur Auffindung von Grid-Ressourcen
- Elementarer Bestandteil eines weltweiten Grids
- Die Informationen sind
  - weltweit verteilt
  - dynamisch
  - sehr detailliert
  - bestehen aus Ressourceninformationen sowie
  - allgemeinem Wissen

Weitere Anwendungsgebiete

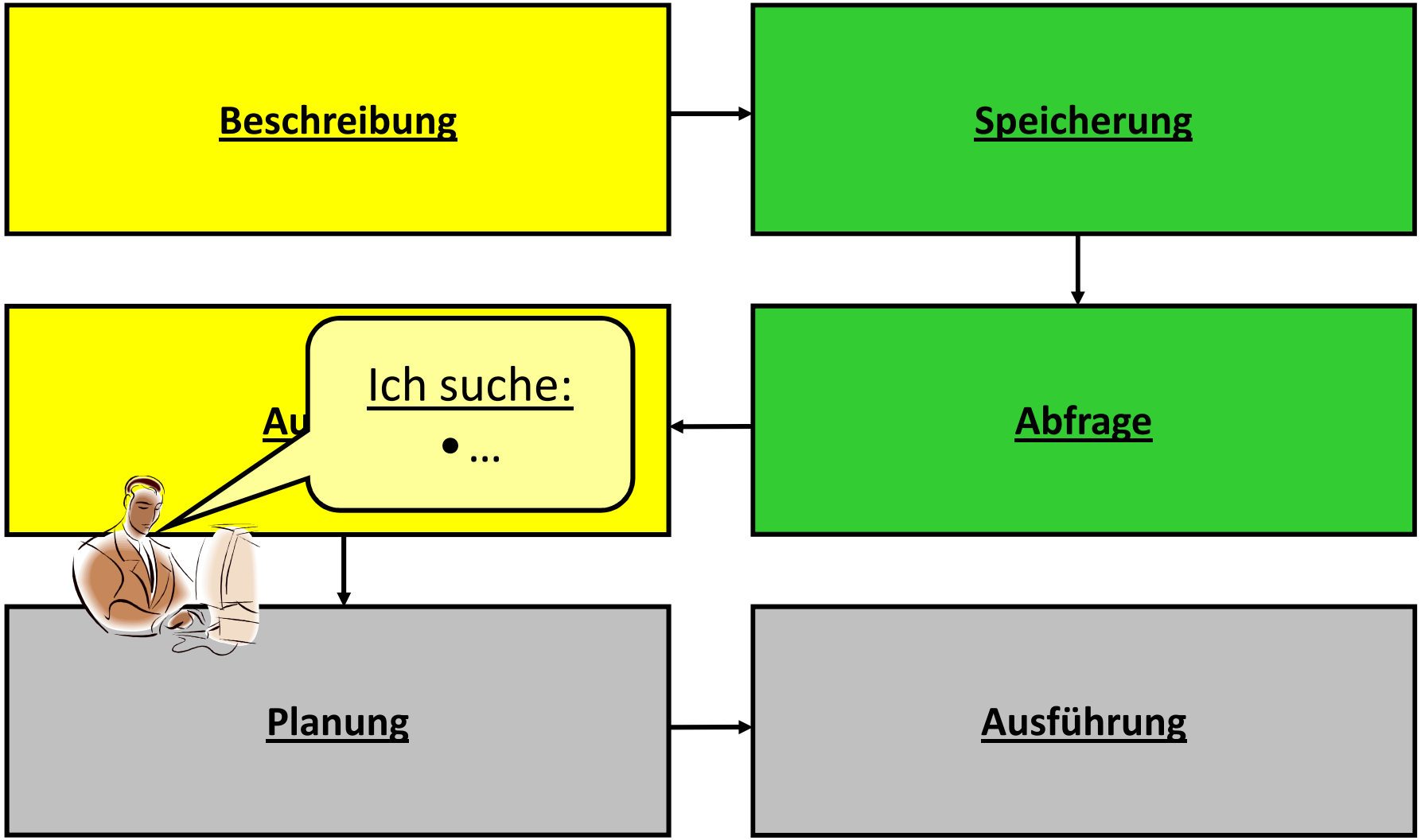
Skalierbares, verteiltes System zum Speichern und Auffinden von heterogenem Wissen



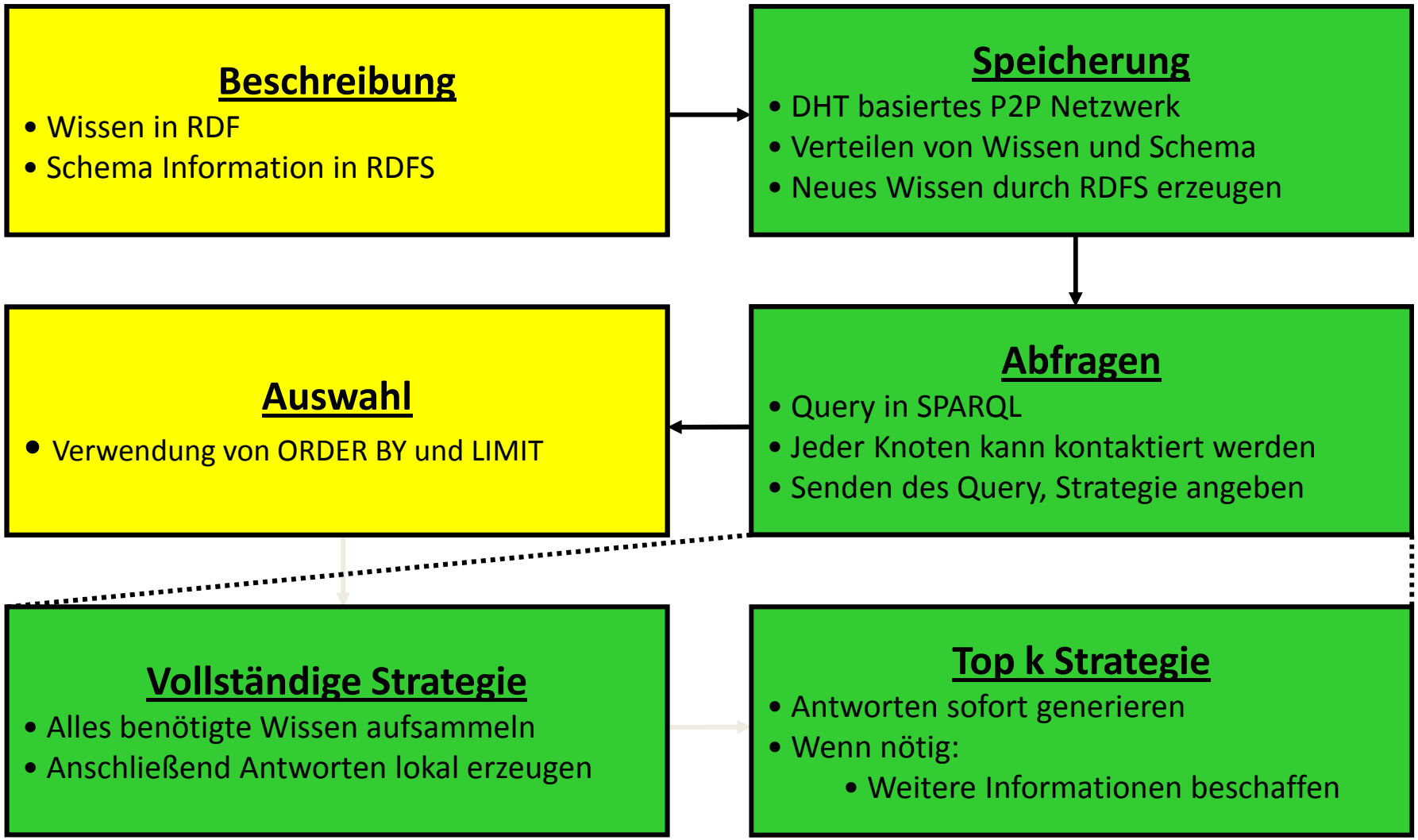
Inhaltsverzeichnis

- Jobabarbeitung im Grid
- Modellierung mit RDF
- Systemüberblick
- Verteilung der Daten
  - RDF Schema Regeln
- Query Auswertung
  - Vollständige Auswertung
  - Optimierungen
- Ergebnisse
- Zusammenfassung

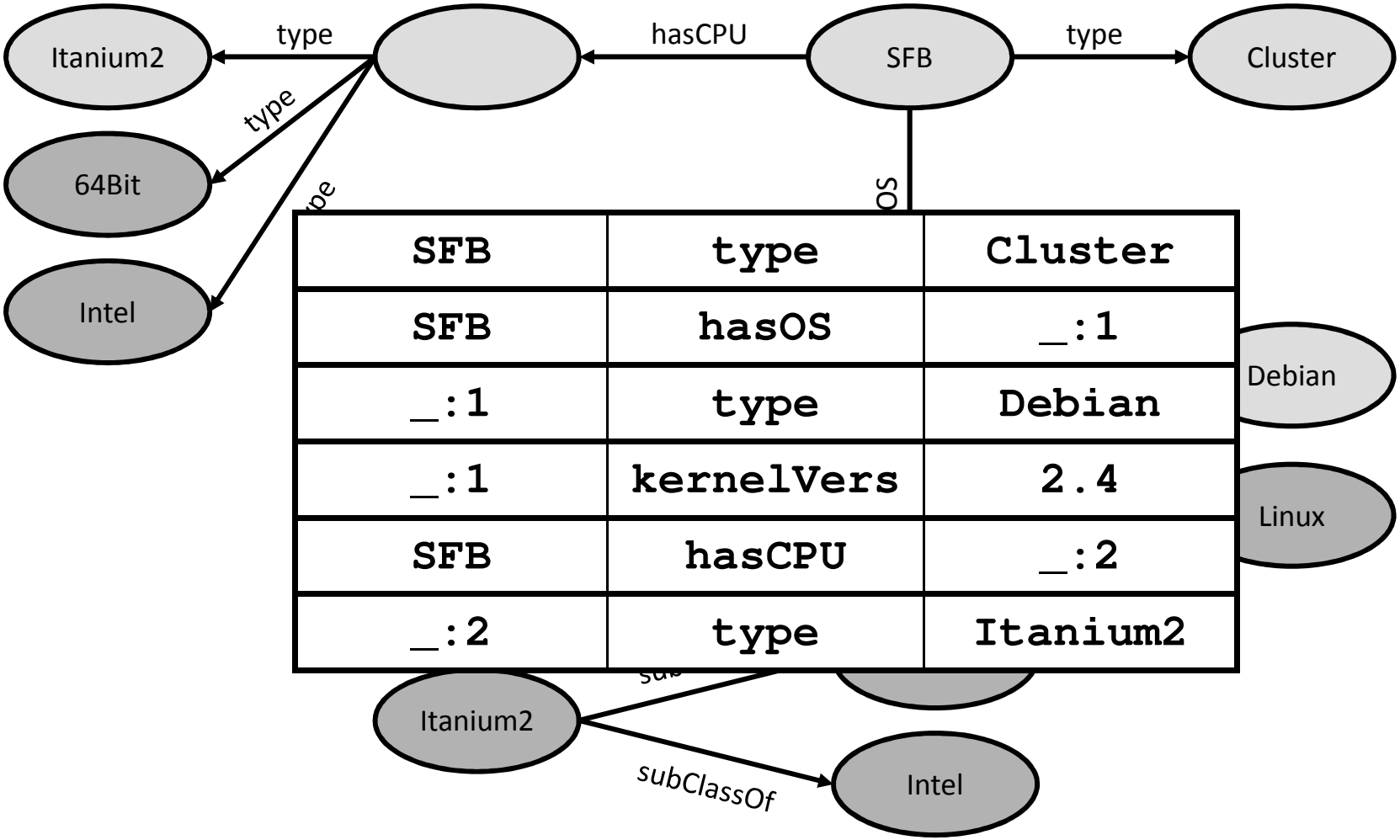
Jobabarbeitung im Grid



Jobabarbeitung im Grid

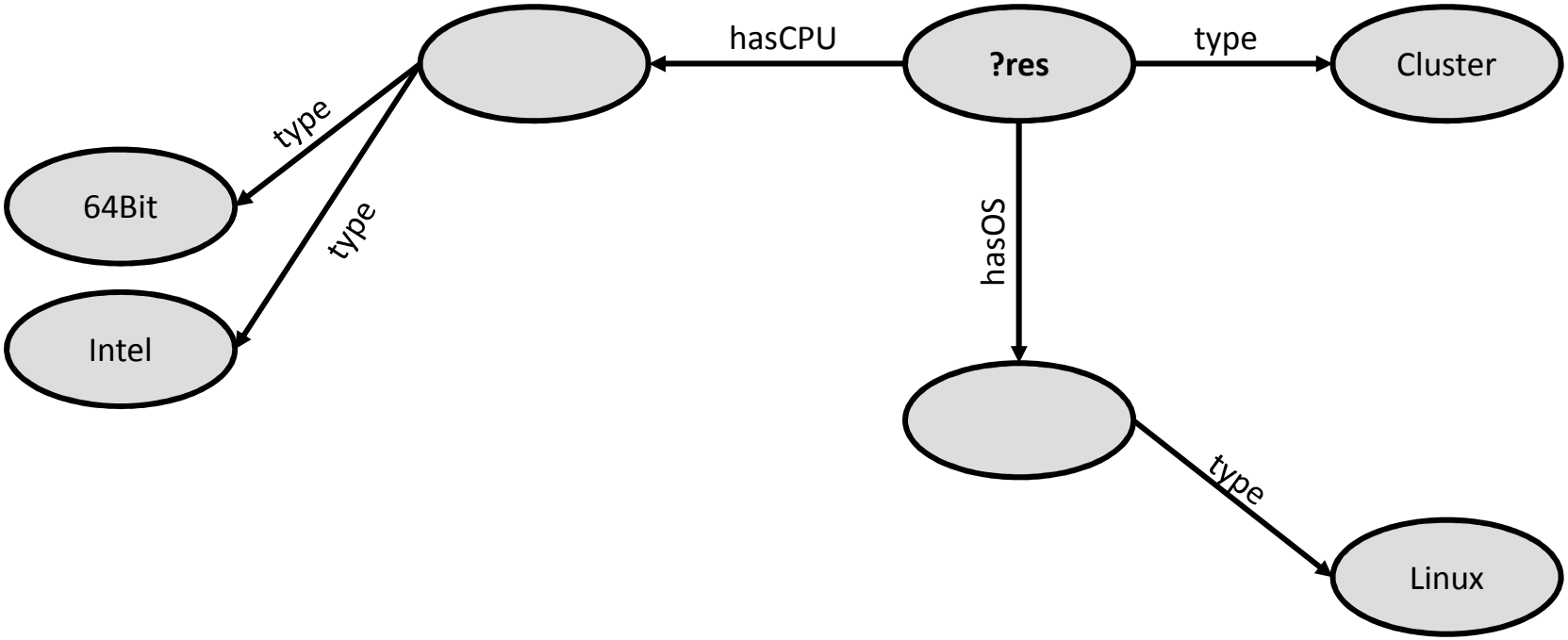


Modellierung: RDF Graph



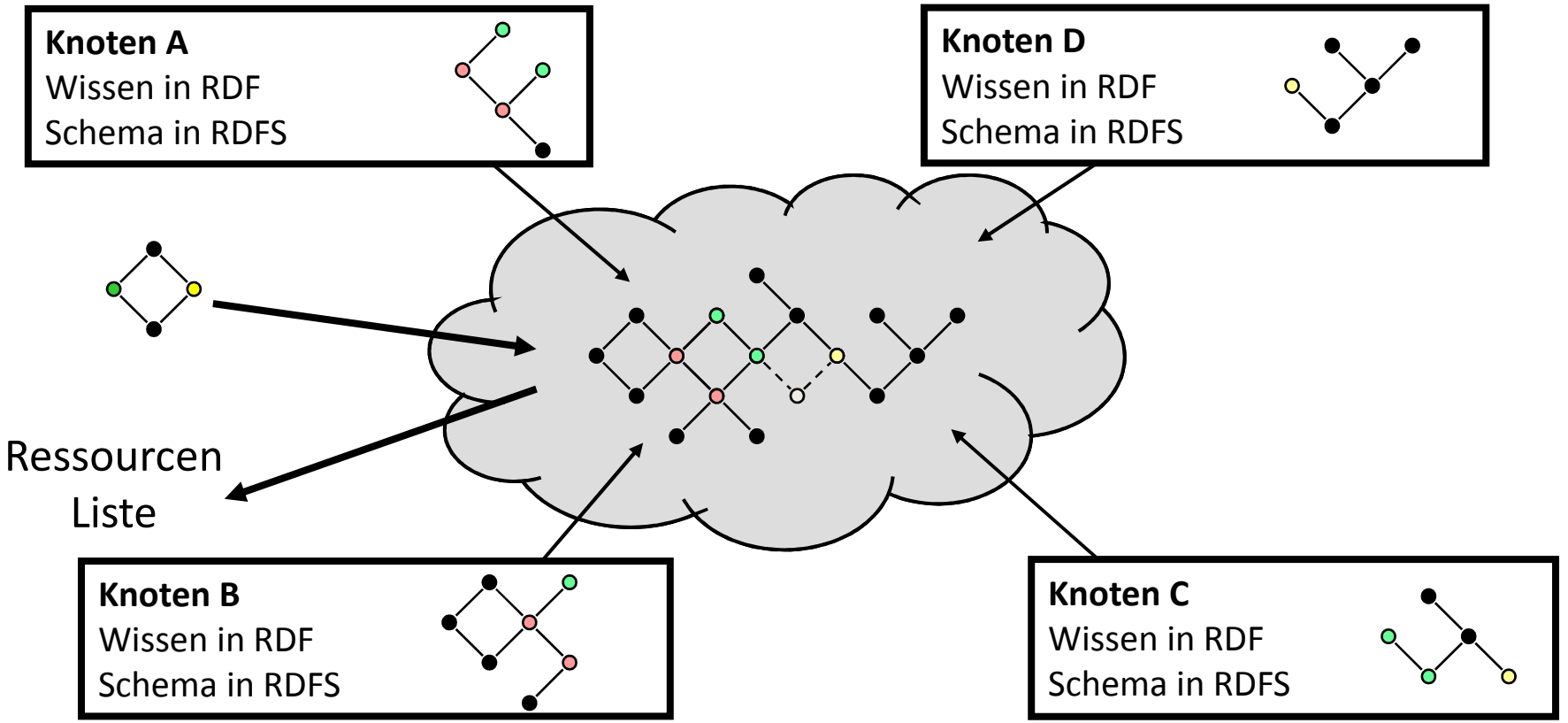
SFB	type	Cluster
SFB	hasOS	_:1
_:1	type	Debian
_:1	kernelVers	2.4
SFB	hasCPU	_:2
_:2	type	Itanium2

Modellierung: RDF Query

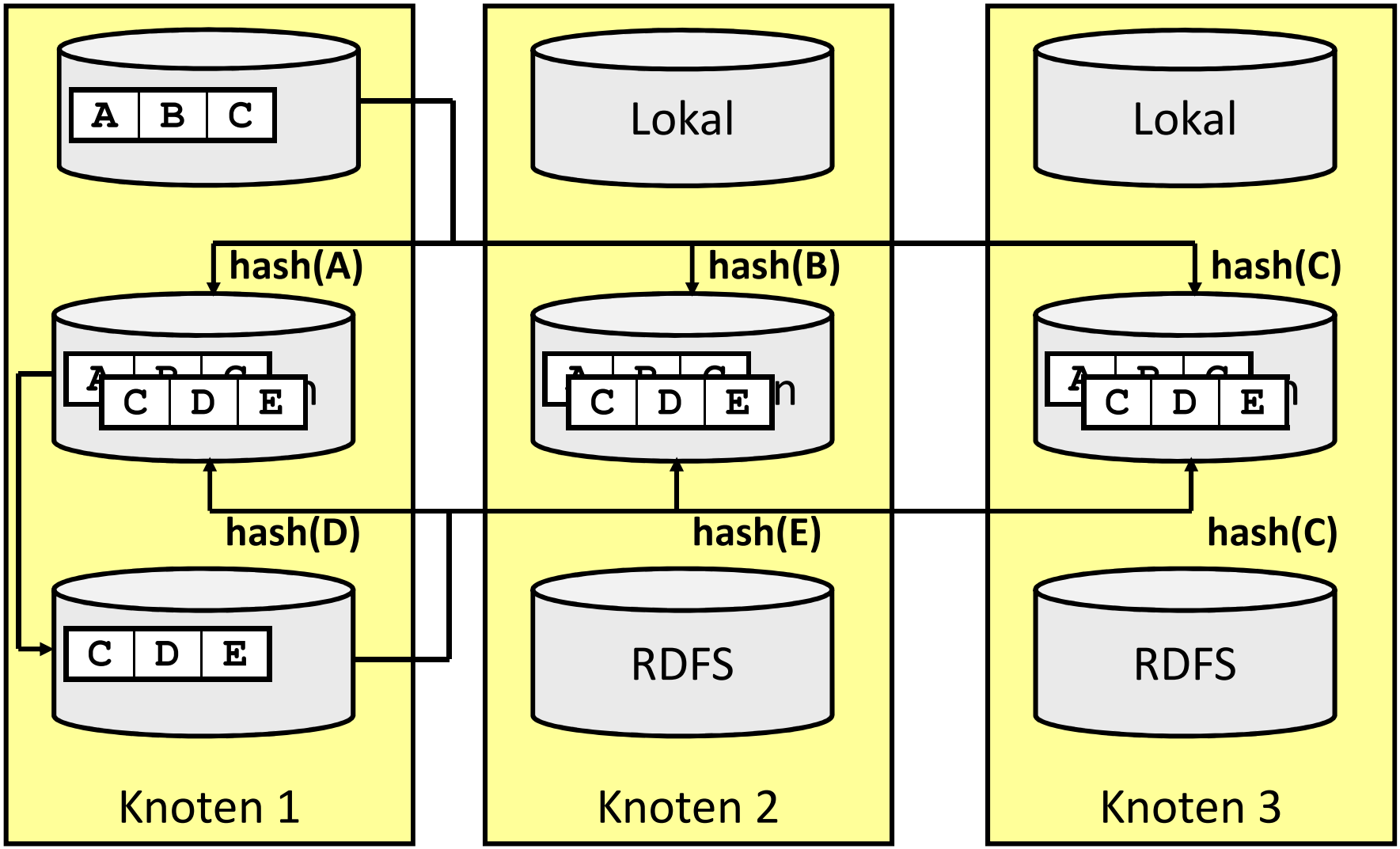


Systemüberblick

- Anfrage: RDF Graph mit Variablen
- Gesucht: Alle passenden Teilgraphen
- Bedingung: RDF Schema Regeln berücksichtigen

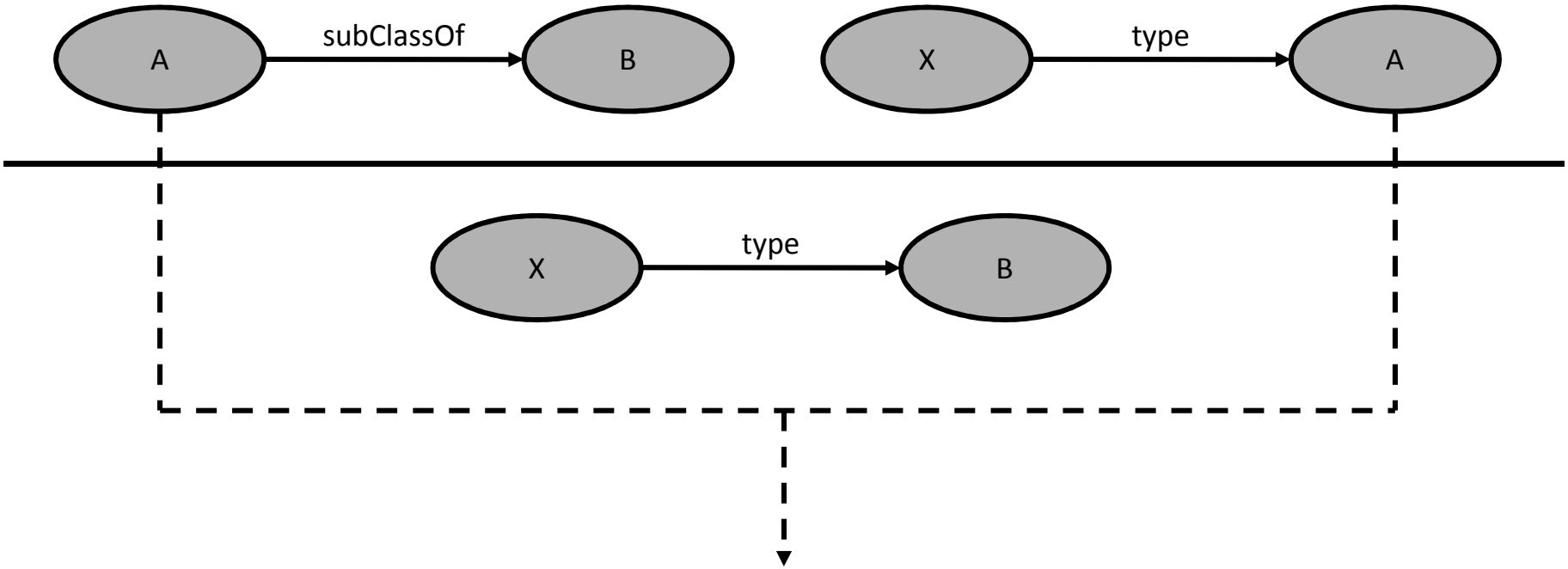


Verteilung der Triple



RDF Schema Regeln

- RDF Schema Semantik: Menge von Regeln



Gleicher DHT-Index ➔ Gleicher Knoten

➔ lokale Auswertung möglich!

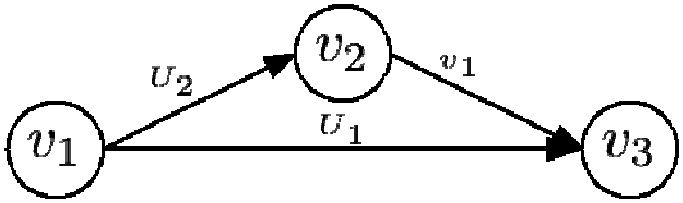
Query Auswertung

- Vollständige Auswertung
  - Erste Phase: Kandidaten aufsammeln
  - Zweite Phase: Treffer lokal ermitteln
  - Optimierungen für erste Phase, verschiedene Strategien

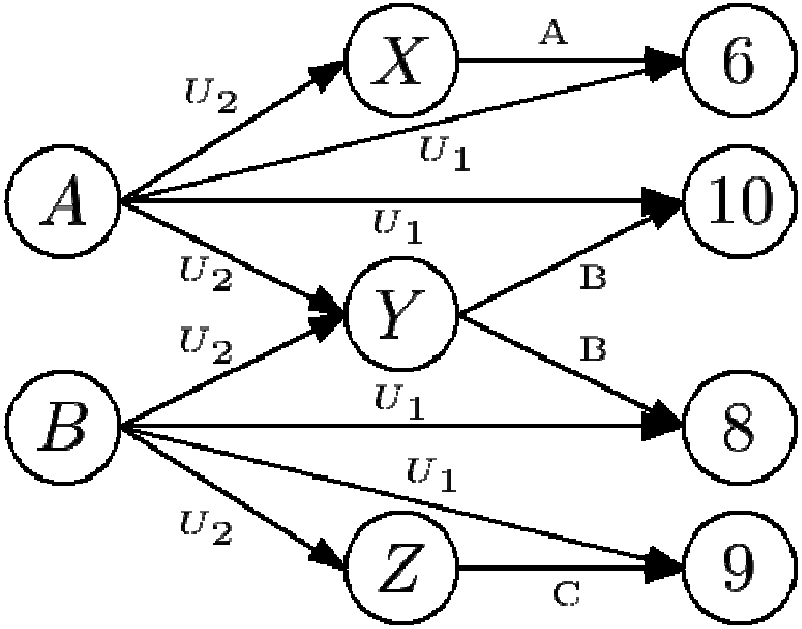
- Top k Auswertung
  - Eine integrierte Phase
  - Kandidaten werden geholt wenn sie benötigt werden
  - Kombination von Caching und Vorausschau
  - Resultate nach einem Attribut sortieren

Beispiel

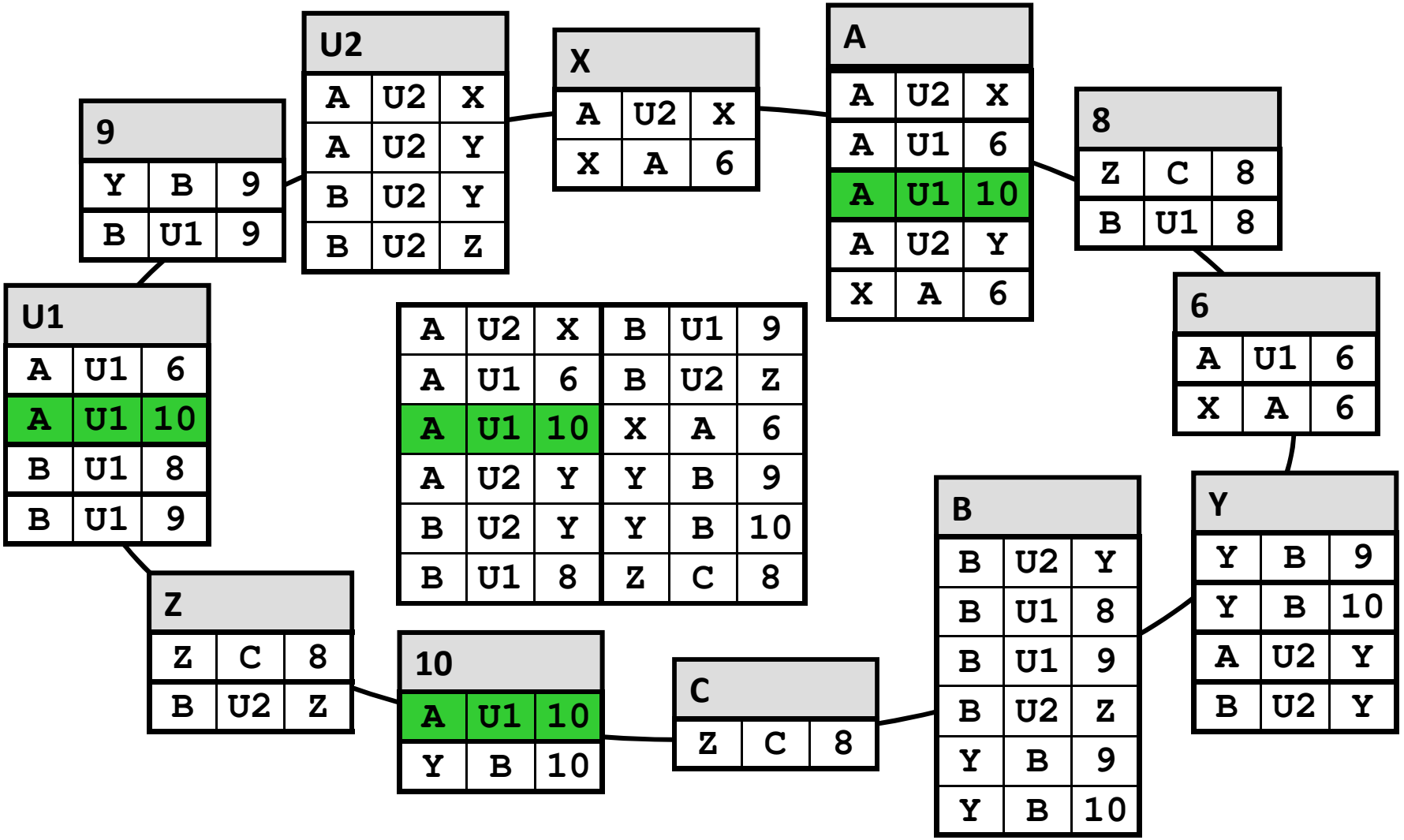
Query



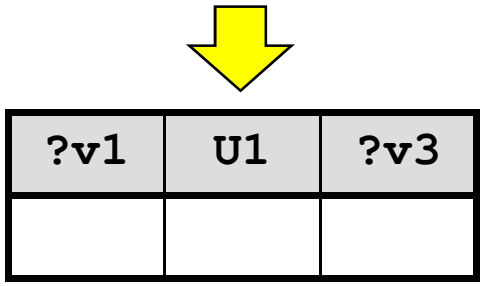
Modell



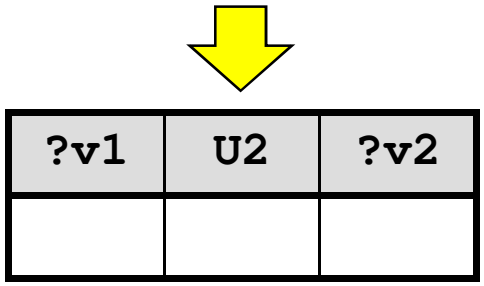
Verteilung der Triple



Query Auswertung



?v1	U1	?v3



?v1	U2	?v2

?v2	?v1	?v3

Query Auswertung

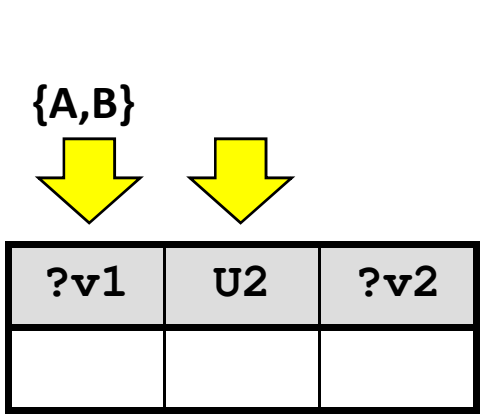
?v1	U1	?v3
A	U1	6
A	U1	10
B	U1	8
B	U1	9

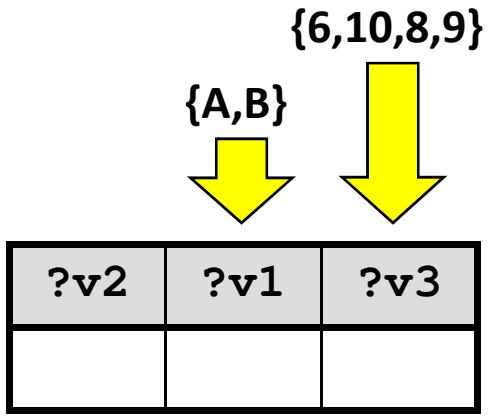
?v1	U2	?v2

?v2	?v1	?v3

Query Auswertung

?v1	U1	?v3
A	U1	6
A	U1	10
B	U1	8
B	U1	9





Query Auswertung

?v1	U1	?v3
A	U1	6
A	U1	10
B	U1	8
B	U1	9

{A,B}

?v1	U2	?v2
A	U2	X
A	U1	6
A	U1	10
A	U2	Y
B	U2	Y
B	U1	8
B	U1	9
B	U2	Z

?v2	?v1	?v3

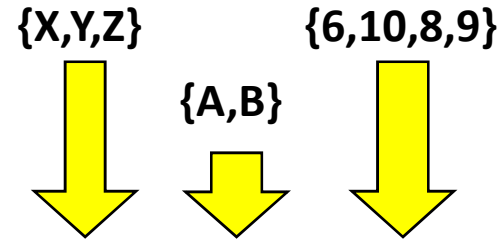
Query Auswertung

?v1	U1	?v3
A	U1	6
A	U1	10
B	U1	8
B	U1	9

?v1	U2	?v2
A	U2	X
A	U1	6
A	U1	10
A	U2	Y
B	U2	Y
B	U1	8
B	U1	9
B	U2	Z

?v2	?v1	?v3

Query Auswertung



?v1	U1	?v3
A	U1	6
A	U1	10
B	U1	8
B	U1	9

?v1	U2	?v2
A	U2	X
A	U2	Y
B	U2	Y
B	U2	Z

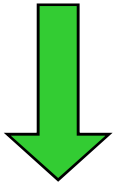
?v2	?v1	?v3

Query Auswertung

?v1	U1	?v3
A	U1	6
A	U1	10
B	U1	8
B	U1	9

?v1	U2	?v2
A	U2	X
A	U2	Y
B	U2	Y
B	U2	Z

{X,Y,Z}



?v2	?v1	?v3
X	A	6
Y	B	9
Y	B	10
Z	C	8

Query Auswertung

?v1	U1	?v3
A	U1	6
A	U1	10
B	U1	8
B	U1	9

?v1	U2	?v2
A	U2	X
A	U2	Y
B	U2	Y
B	U2	Z

?v2	?v1	?v3
X	A	6
Y	B	9
Y	B	10
Z	C	8

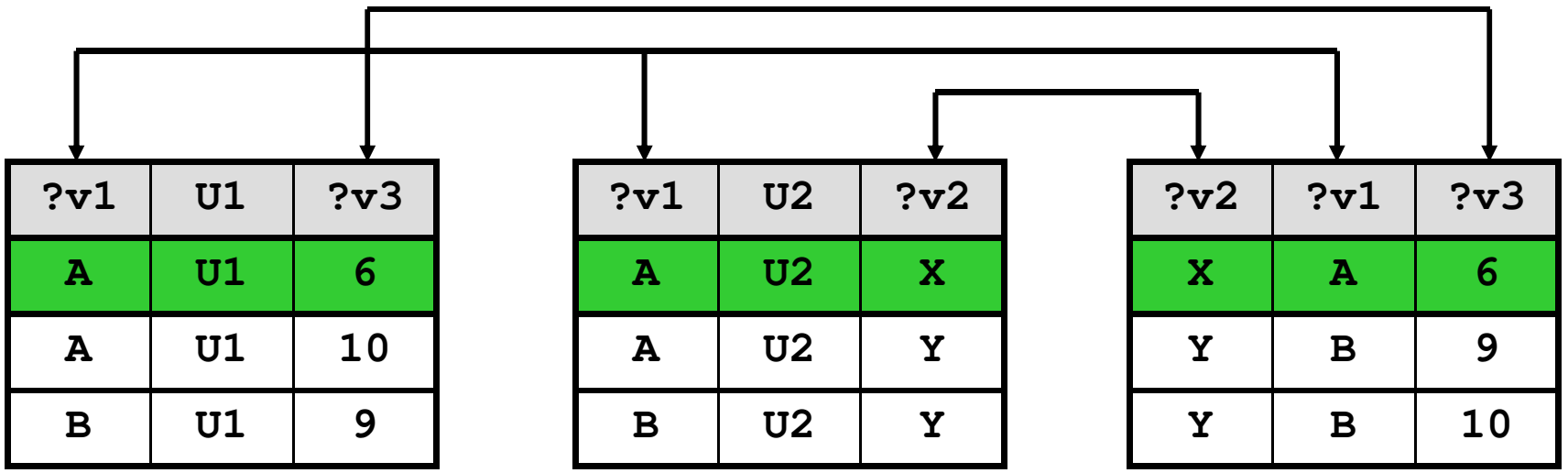
Query Auswertung

?v1	U1	?v3
A	U1	6
A	U1	10
B	U1	9

?v1	U2	?v2
A	U2	X
A	U2	Y
B	U2	Y

?v2	?v1	?v3
X	A	6
Y	B	9
Y	B	10

Backtracking

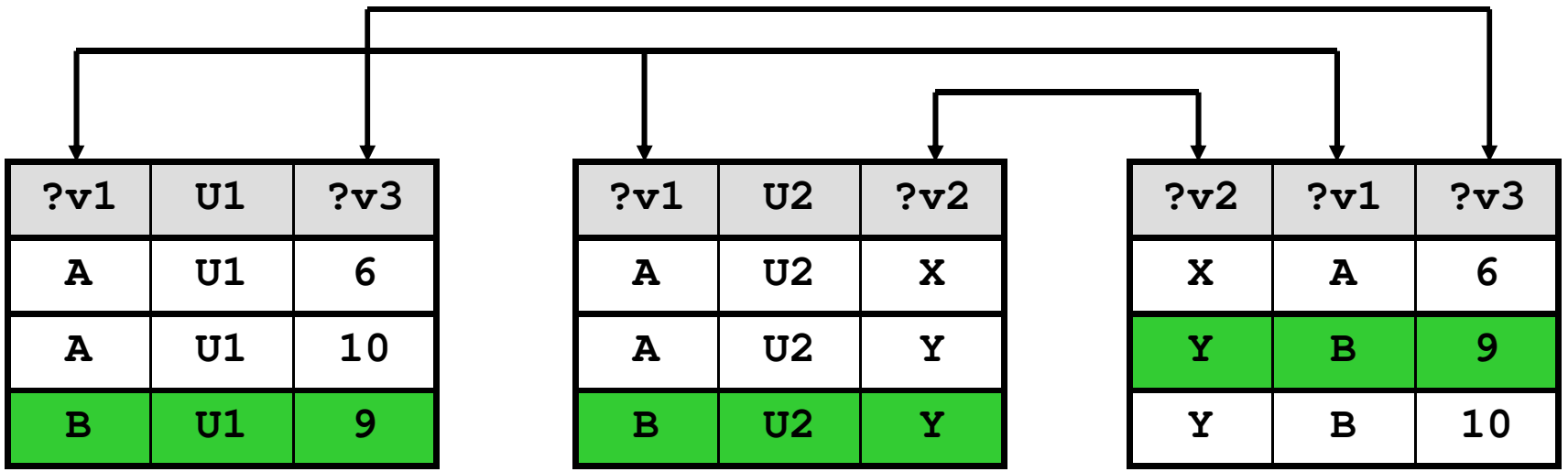


?v1	U1	?v3
A	U1	6
A	U1	10
B	U1	9

?v1	U2	?v2
A	U2	X
A	U2	Y
B	U2	Y

?v2	?v1	?v3
X	A	6
Y	B	9
Y	B	10

Backtracking

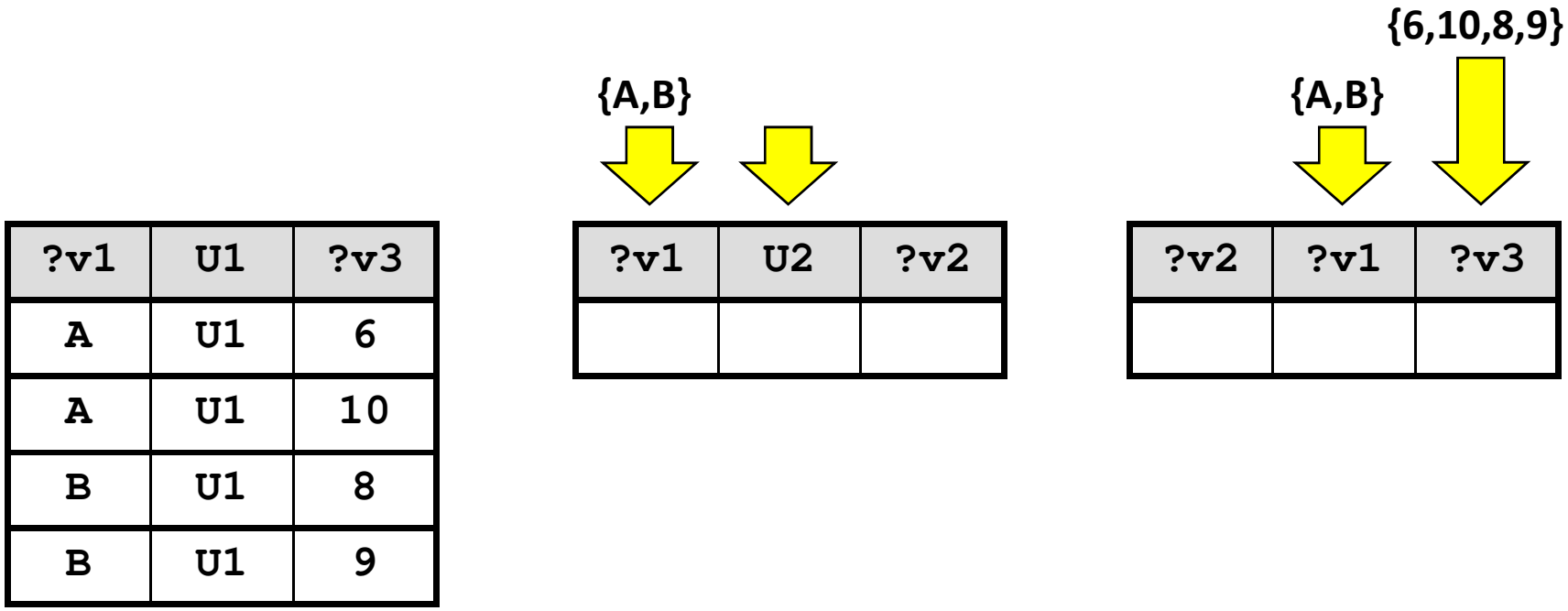


?v1	U1	?v3
A	U1	6
A	U1	10
B	U1	9

?v1	U2	?v2
A	U2	X
A	U2	Y
B	U2	Y

?v2	?v1	?v3
X	A	6
Y	B	9
Y	B	10

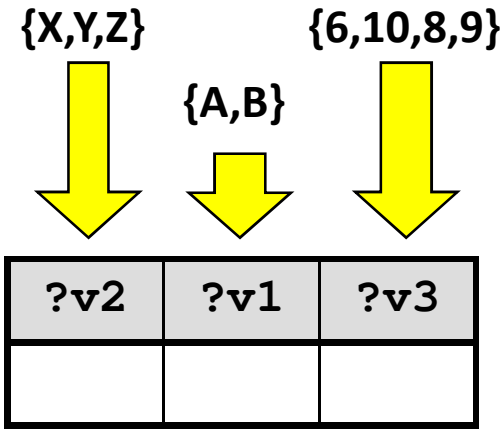
Wie wähle ich den nächsten Index?



- Für jede Möglichkeit die erwartete Resultatsgröße bestimmen
- Werte während einer Query Cachen

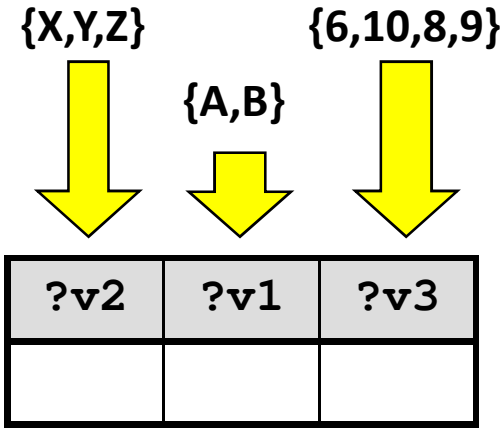
Bloom Filter

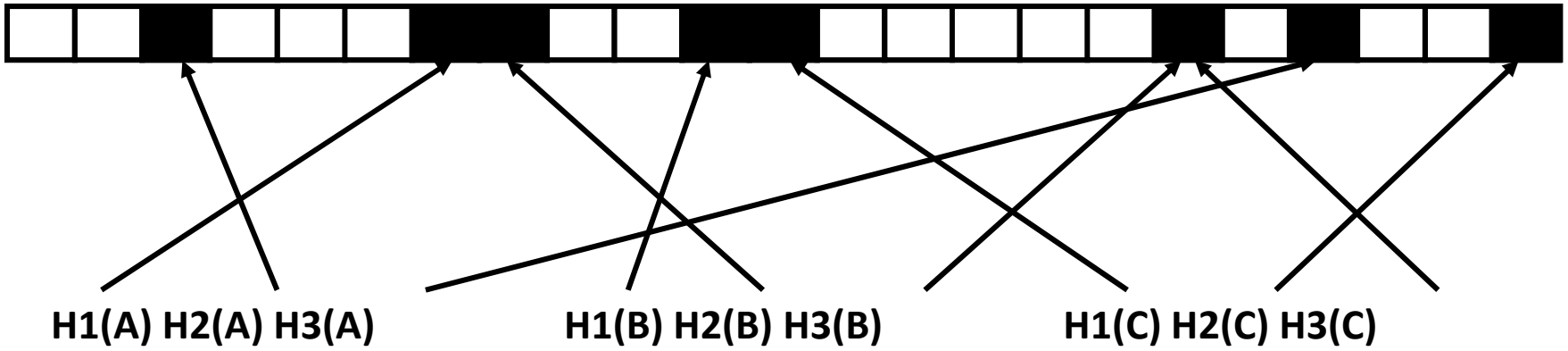
- Typische Situation:
  - Mehrere Variablen in einem Triple
  - Übertragung der Kandidaten verkleinert Resultat
  - Aber: Kandidaten übertragen ist auch teuer!



Bloom Filter

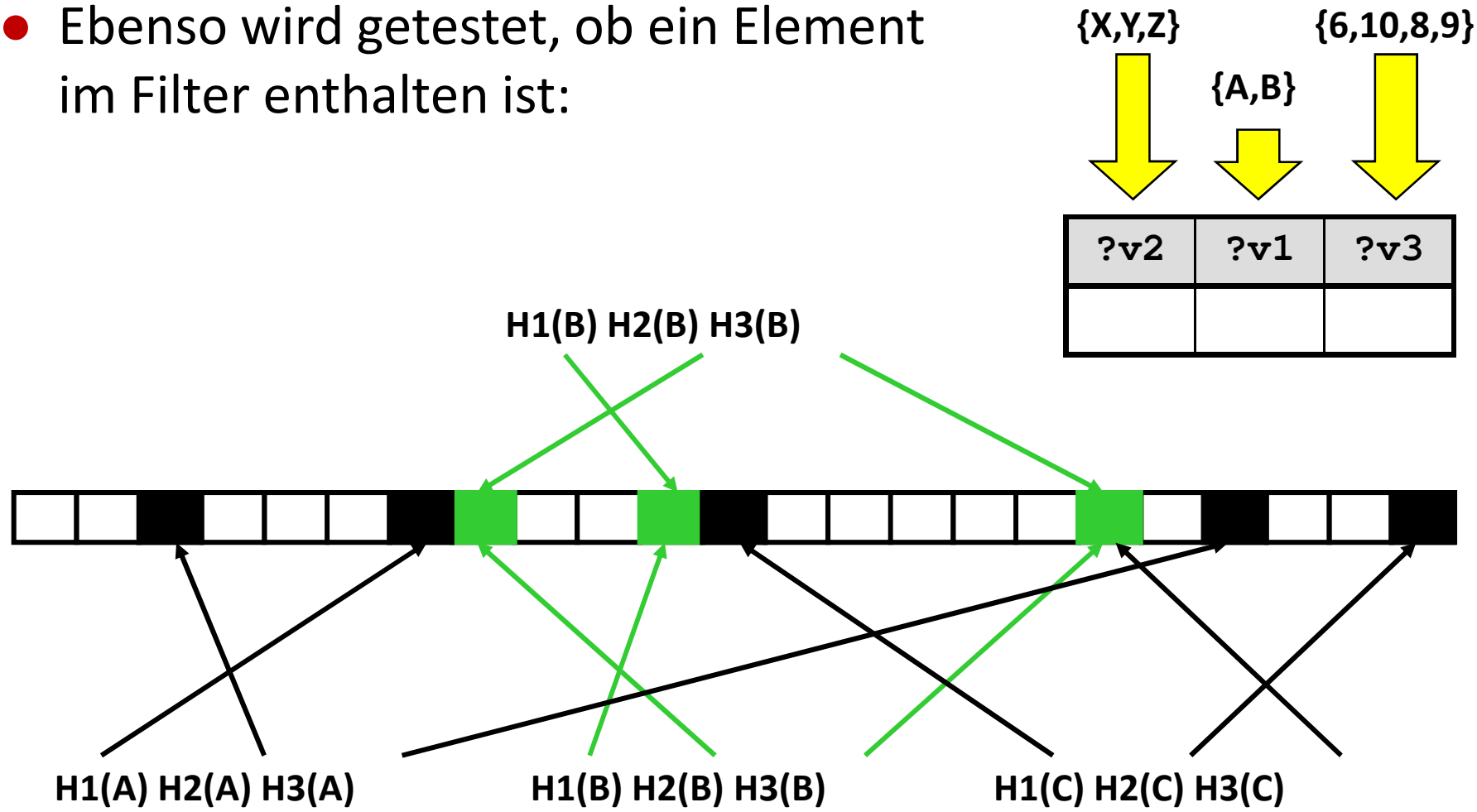
- Bit-Array fester Größe
- Fester Satz von Hash-Funktionen
- Elemente werden gespeichert, indem passende Bits gesetzt werden:





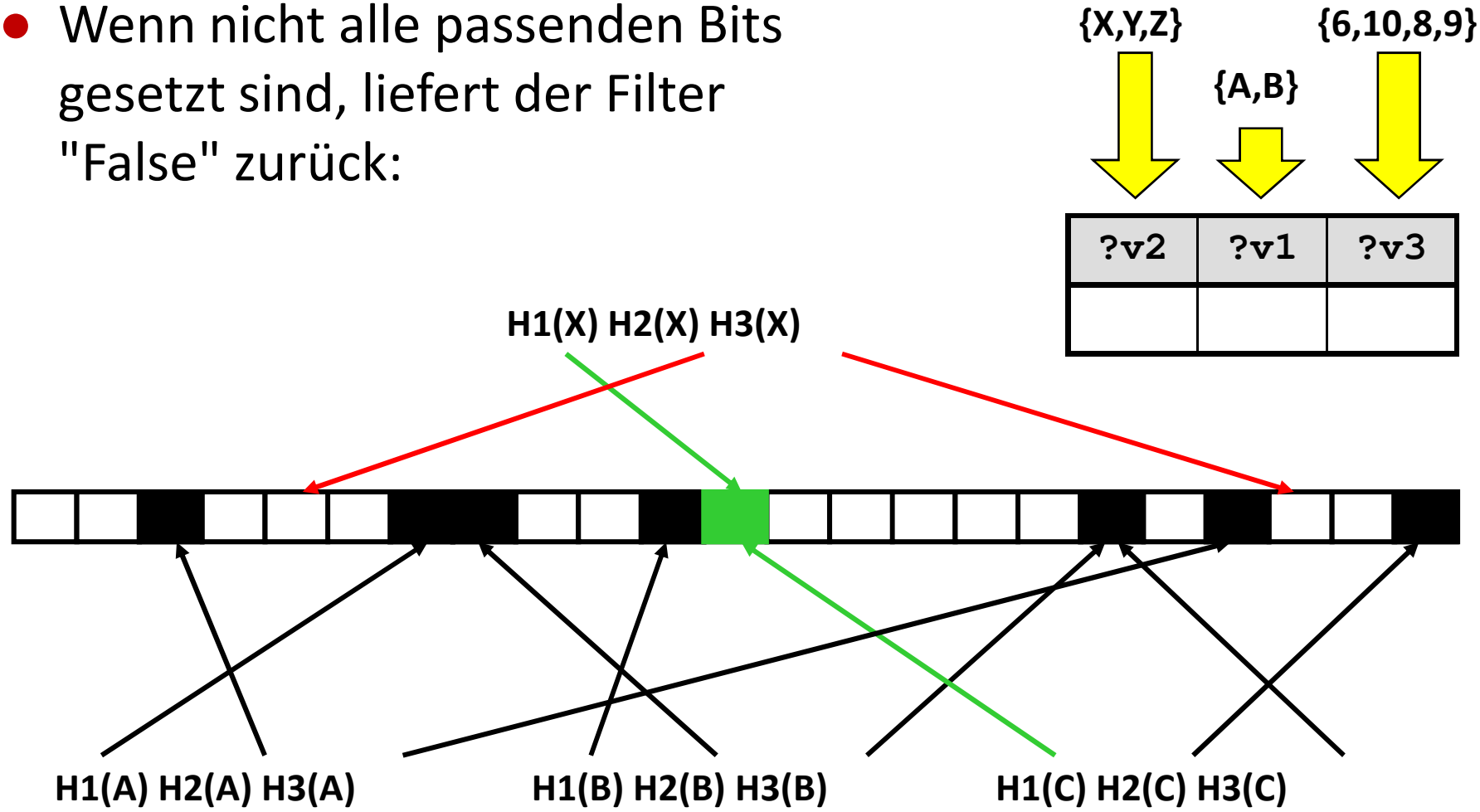
Bloom Filter

- Ebenso wird getestet, ob ein Element im Filter enthalten ist:



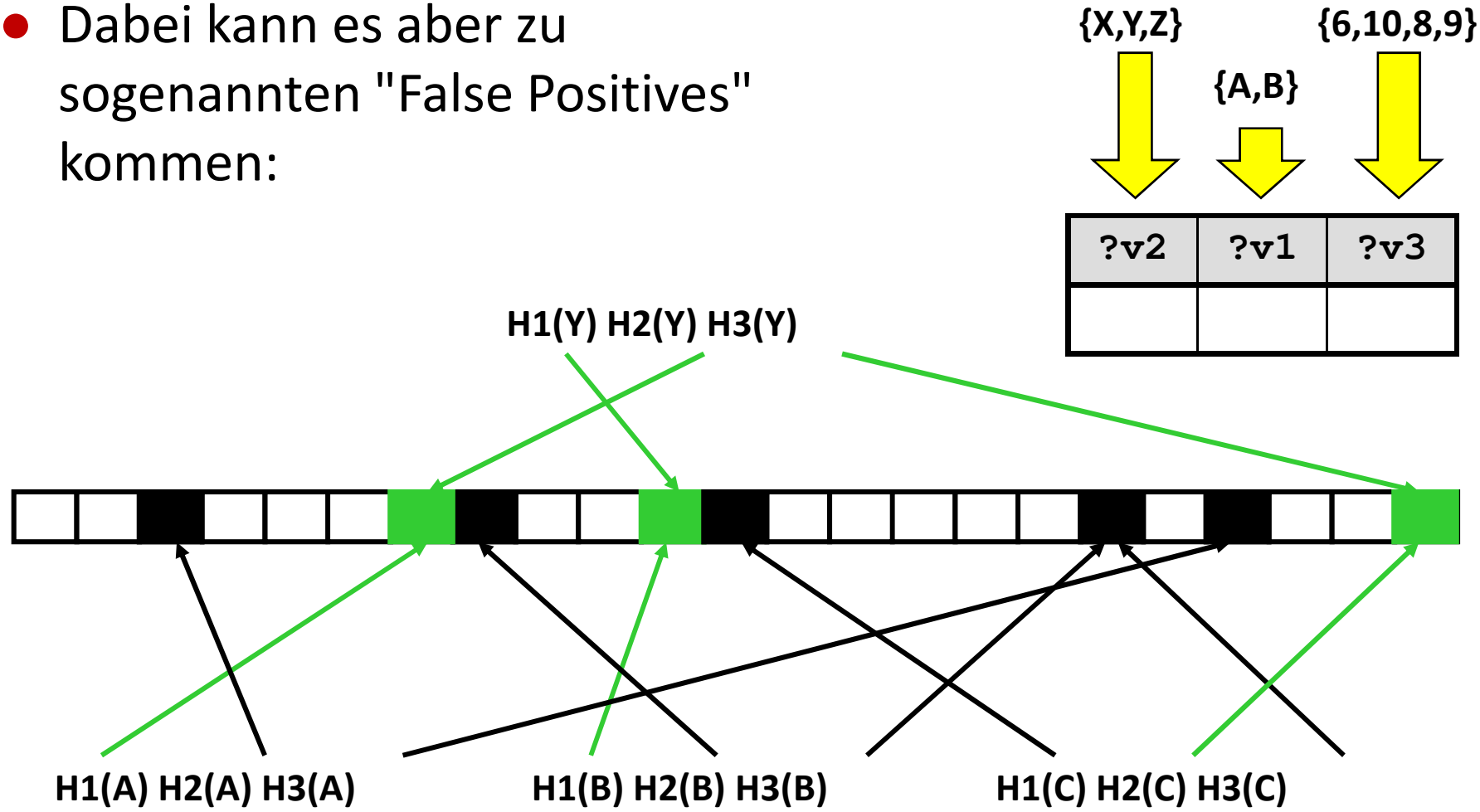
Bloom Filter

- Wenn nicht alle passenden Bits gesetzt sind, liefert der Filter "False" zurück:



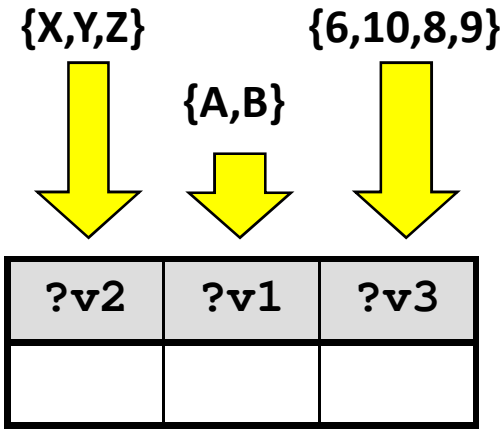
Bloom Filter

- Dabei kann es aber zu sogenannten "False Positives" kommen:



Verwendung der Bloom Filter

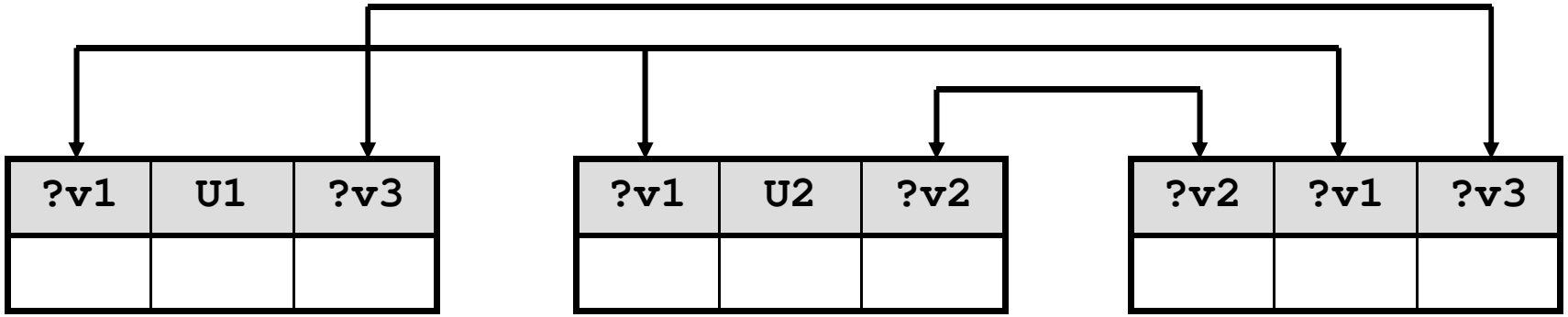
- Beim Abholen der Kandidaten:
  - Bekannte Kandidatenmengen für Variablen als Bloom-Filter mitschicken
  - Ziel-Peer filtert das Resultat entsprechend
  - Lokal muss nochmal gefiltert werden, um die False Positives auszuschließen
  - Es gehen aber keine Treffer verloren!
- Bei der Vorschau auf die Größe der Ergebnismenge



Top k Query Evaluation

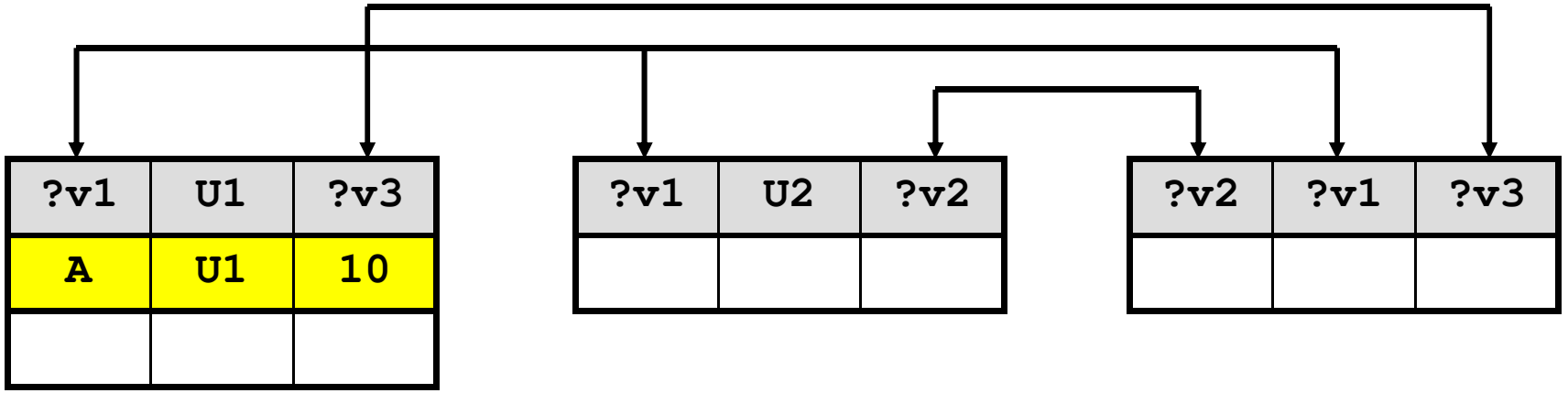
- Szenario: Queries mit großer Ergebnismenge:
  - Viele Daten
  - Strukturell sehr ähnliche Daten
  - Sehr generische Abfrage
- Kein Interesse an allen Resultaten (vgl. Google)
- Typisches Interesse: Die 10 *besten* Resultate
- Aktuelle Lösung:
  - Resultate nach dem Wert einer Variable sortieren
  - Beide Phasen der Query-Auswertung integrieren

Top k Query Evaluation



*	U1	*
A	U1	10
B	U1	9
B	U1	8
A	U1	6

Top k Query Evaluation



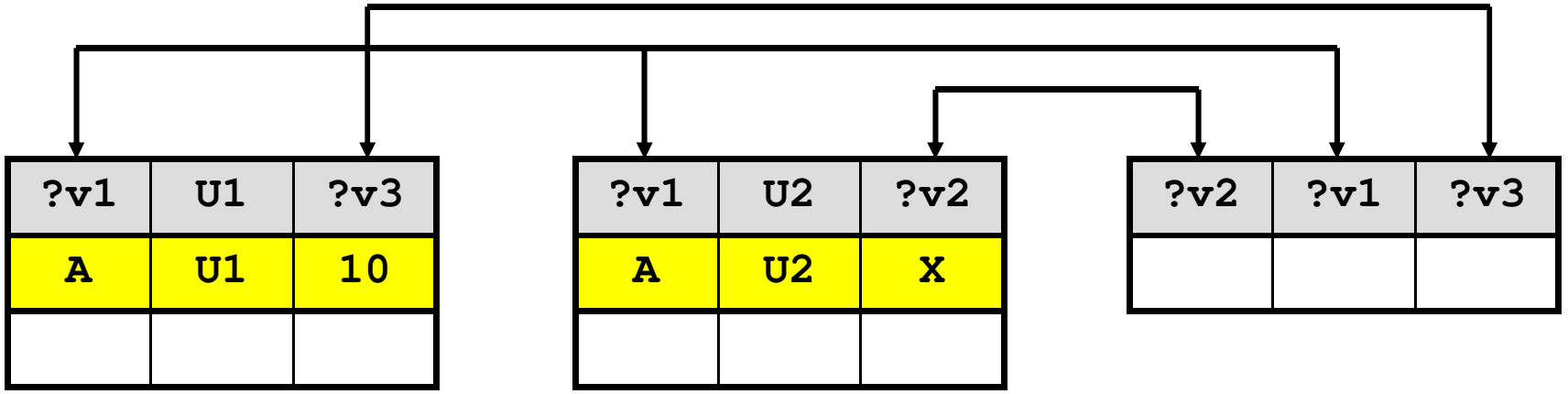
?v1	U1	?v3
A	U1	10

?v1	U2	?v2

?v2	?v1	?v3

*	U1	*
A	U1	10
B	U1	9
B	U1	8
A	U1	6

Top k Query Evaluation



?v1	U1	?v3
A	U1	10

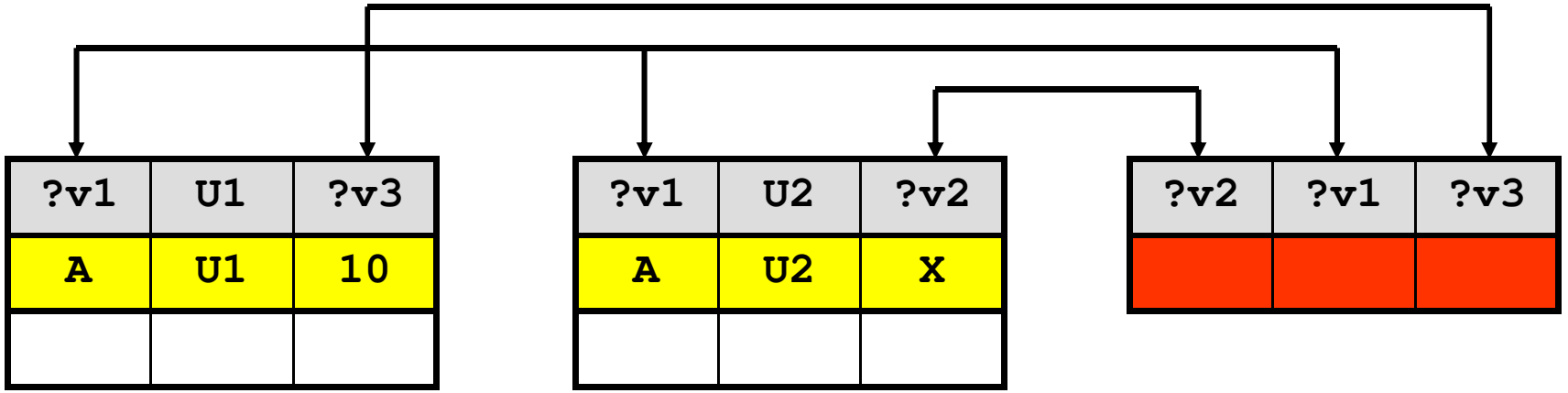
?v1	U2	?v2
A	U2	X

?v2	?v1	?v3

*	U1	*
A	U1	10
B	U1	9
B	U1	8
A	U1	6

A	U2	*
A	U2	X
A	U2	Y

Top k Query Evaluation



?v1	U1	?v3
A	U1	10

?v1	U2	?v2
A	U2	X

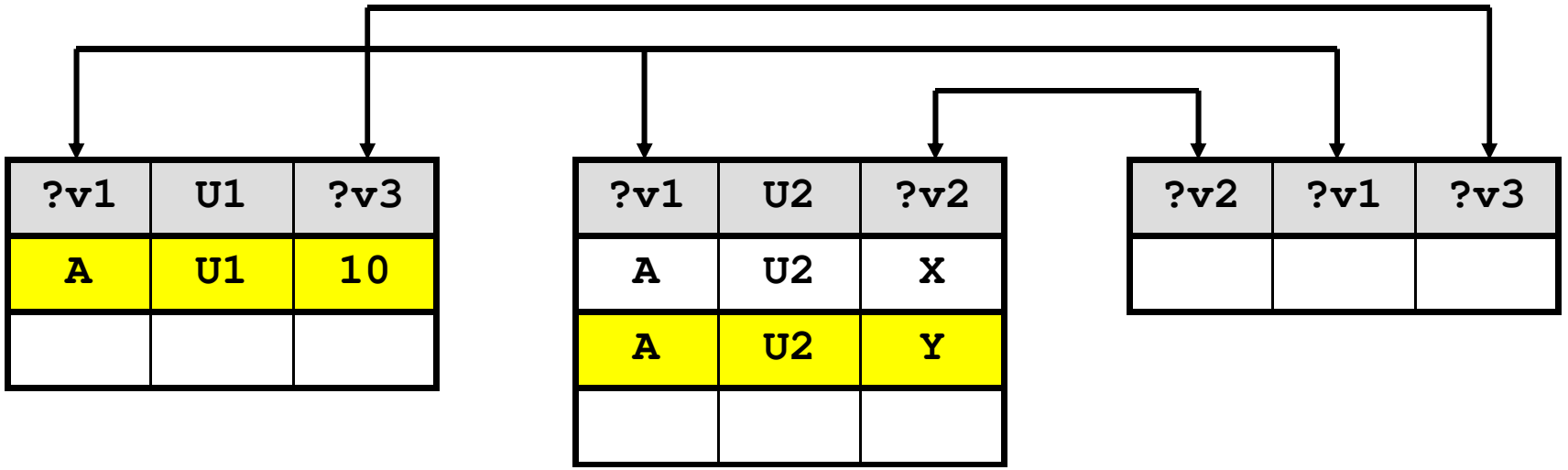
?v2	?v1	?v3

*	U1	*
A	U1	10
B	U1	9
B	U1	8
A	U1	6

A	U2	*
A	U2	X
A	U2	Y

X	A	10

Top k Query Evaluation



?v1	U1	?v3
A	U1	10

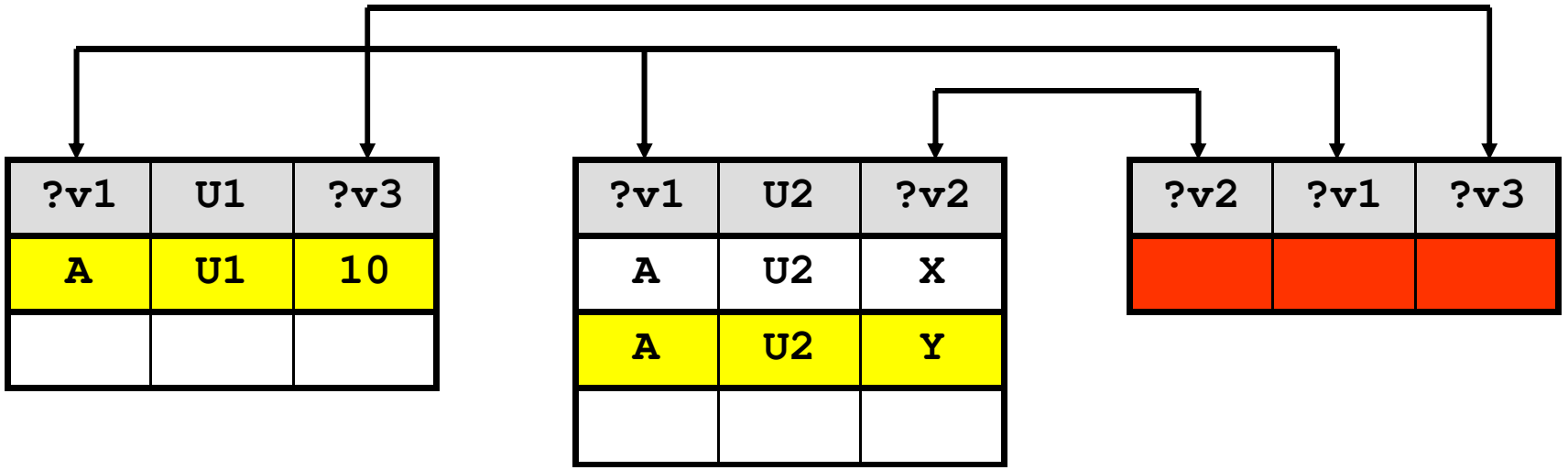
?v1	U2	?v2
A	U2	X
A	U2	Y

?v2	?v1	?v3

*	U1	*
A	U1	10
B	U1	9
B	U1	8
A	U1	6

A	U2	*
A	U2	X
A	U2	Y

Top k Query Evaluation



?v1	U1	?v3
A	U1	10

?v1	U2	?v2
A	U2	X
A	U2	Y

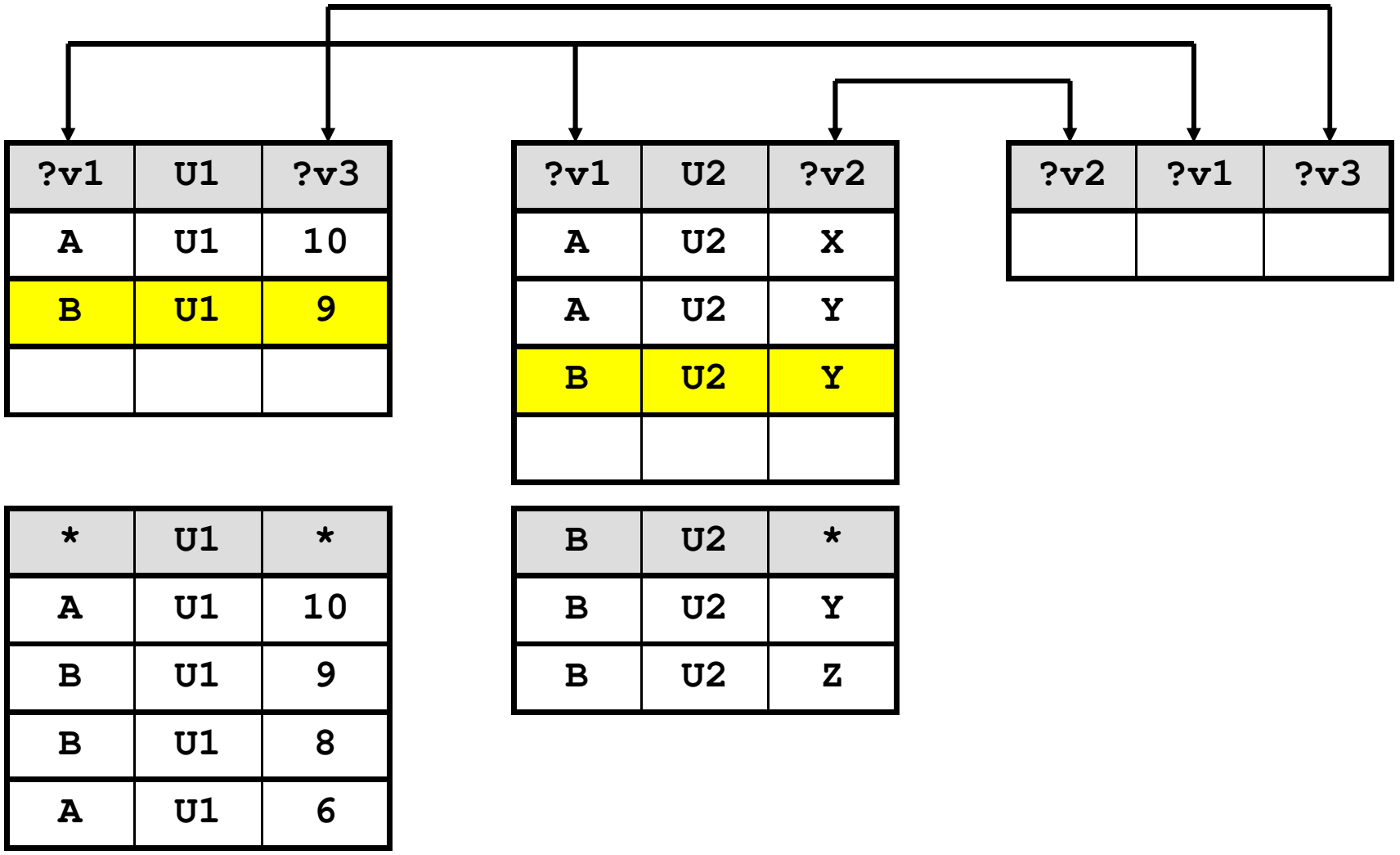
?v2	?v1	?v3

*	U1	*
A	U1	10
B	U1	9
B	U1	8
A	U1	6

A	U2	*
A	U2	X
A	U2	Y

Y	A	10

Top k Query Evaluation



?v1	U1	?v3
A	U1	10
B	U1	9

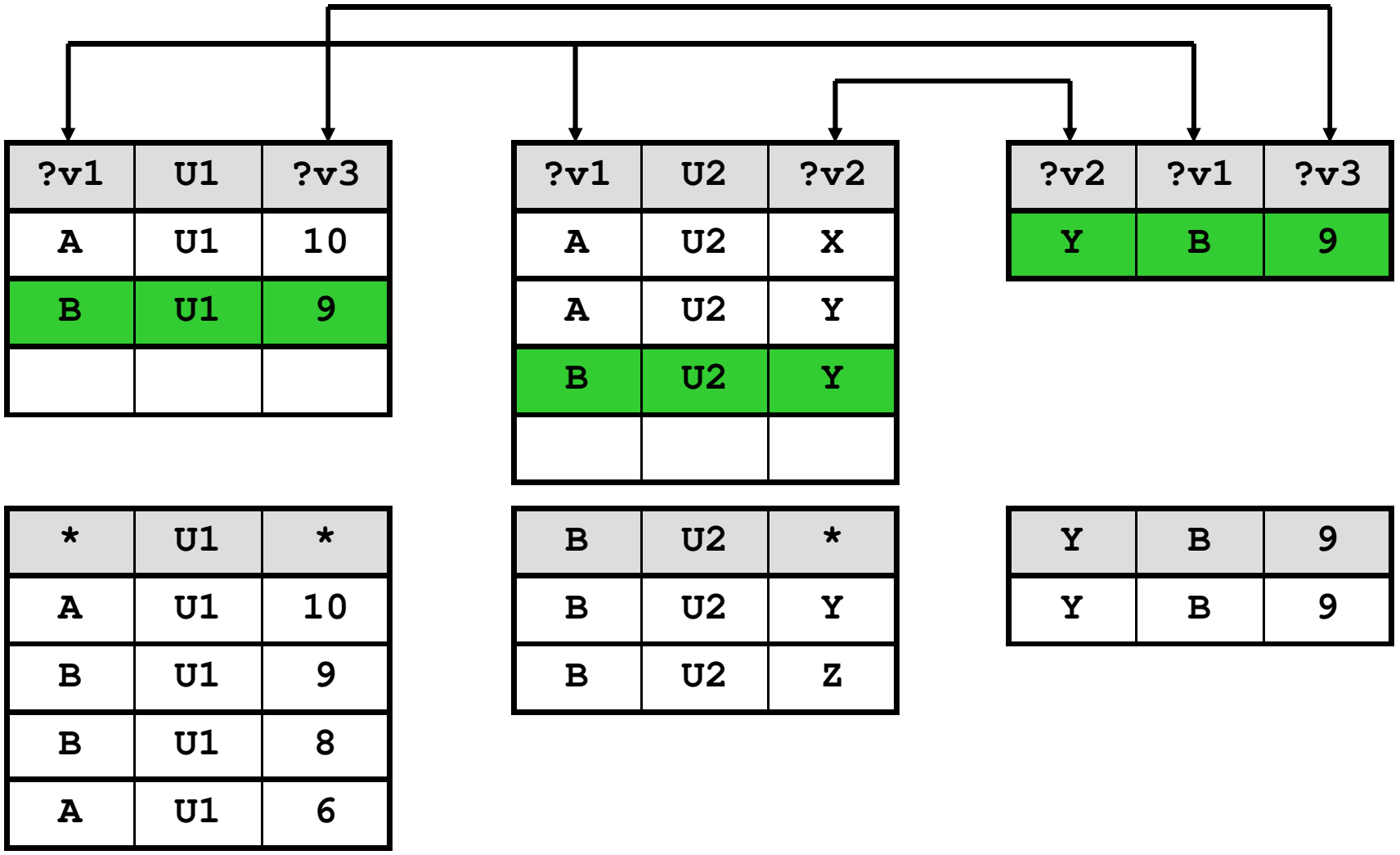
?v1	U2	?v2
A	U2	X
A	U2	Y
B	U2	Y

?v2	?v1	?v3

*	U1	*
A	U1	10
B	U1	9
B	U1	8
A	U1	6

B	U2	*
B	U2	Y
B	U2	Z

Top k Query Evaluation



?v1	U1	?v3
A	U1	10
B	U1	9

?v1	U2	?v2
A	U2	X
A	U2	Y
B	U2	Y

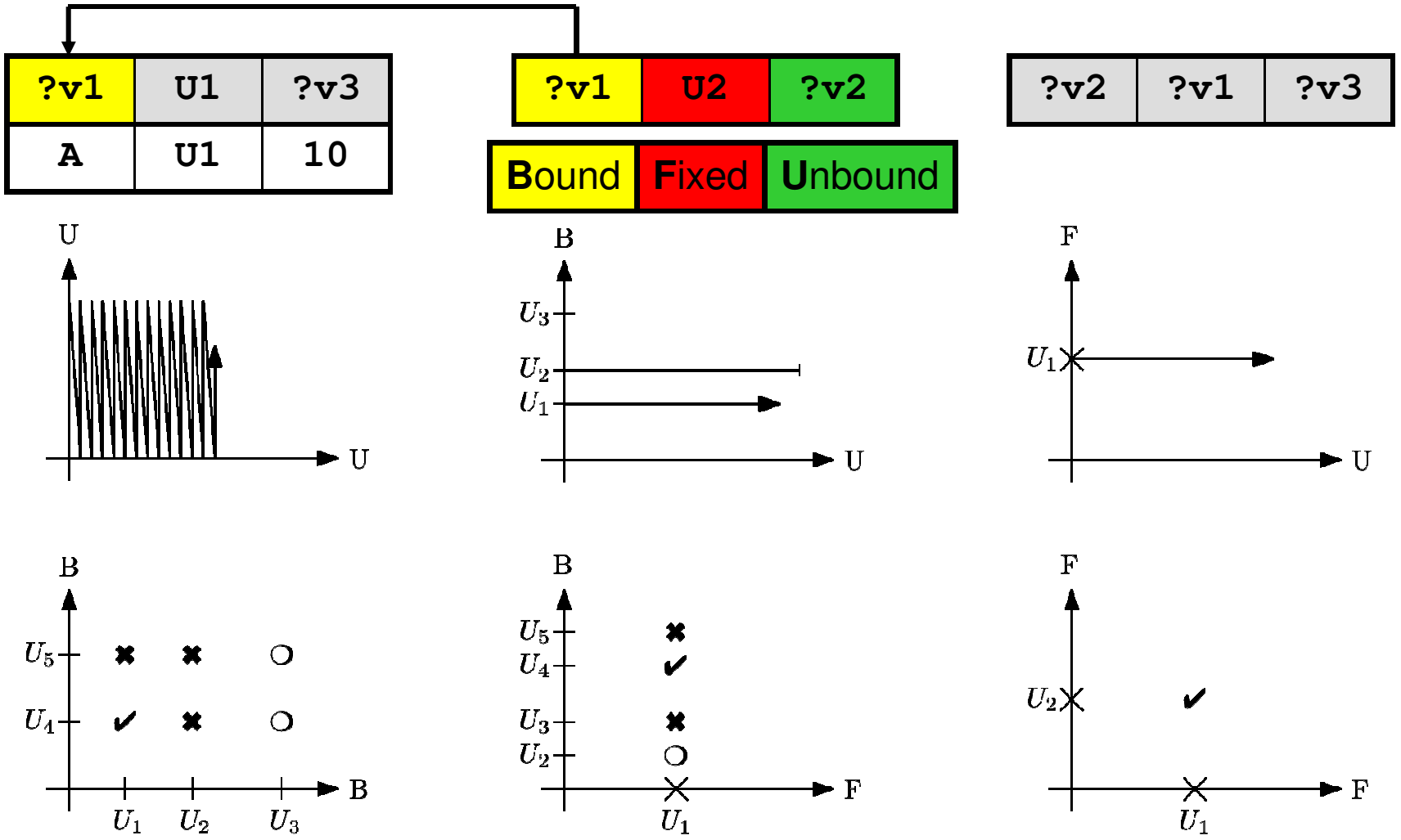
?v2	?v1	?v3
Y	B	9

*	U1	*
A	U1	10
B	U1	9
B	U1	8
A	U1	6

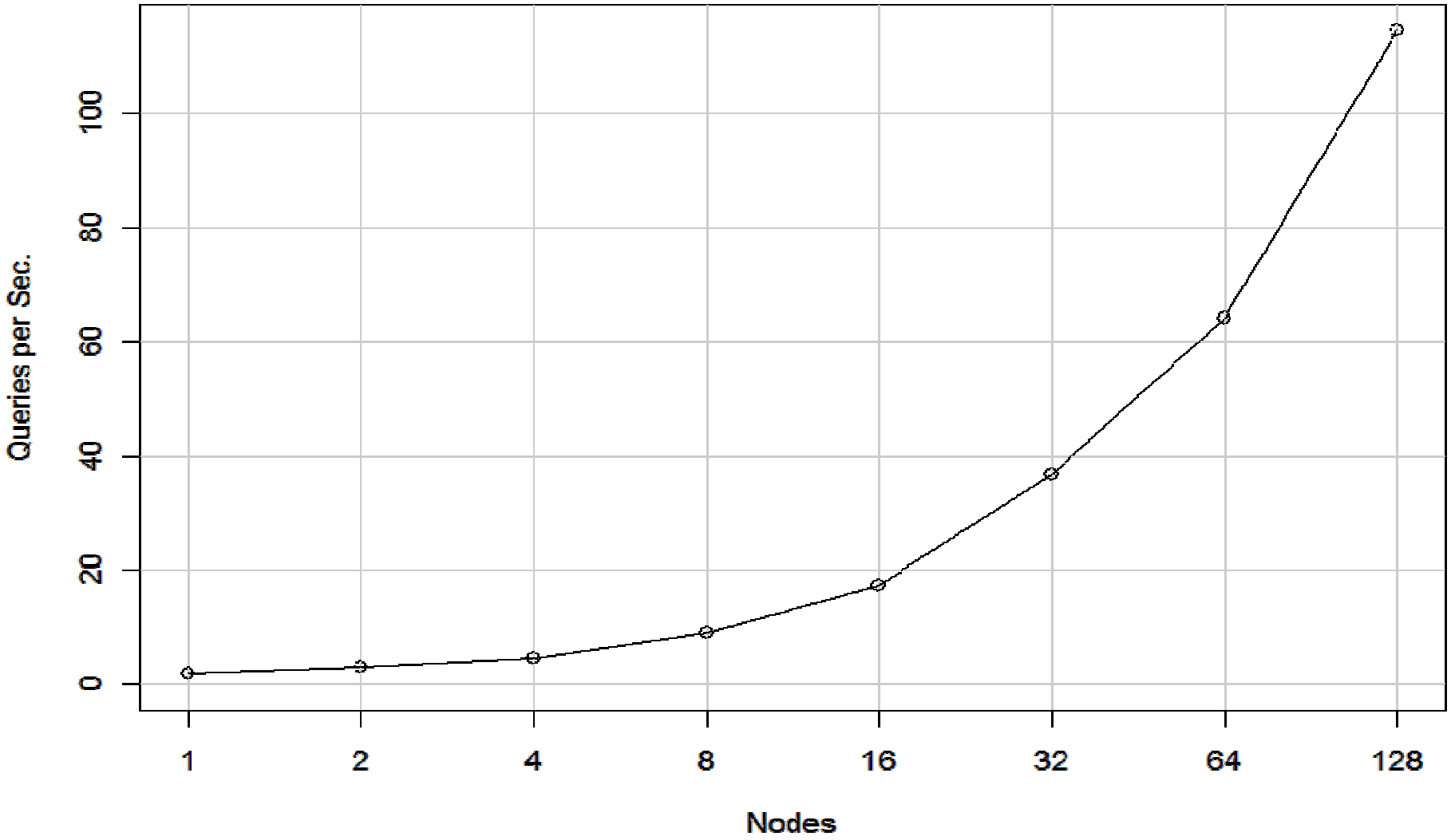
B	U2	*
B	U2	Y
B	U2	Z

Y	B	9
Y	B	9

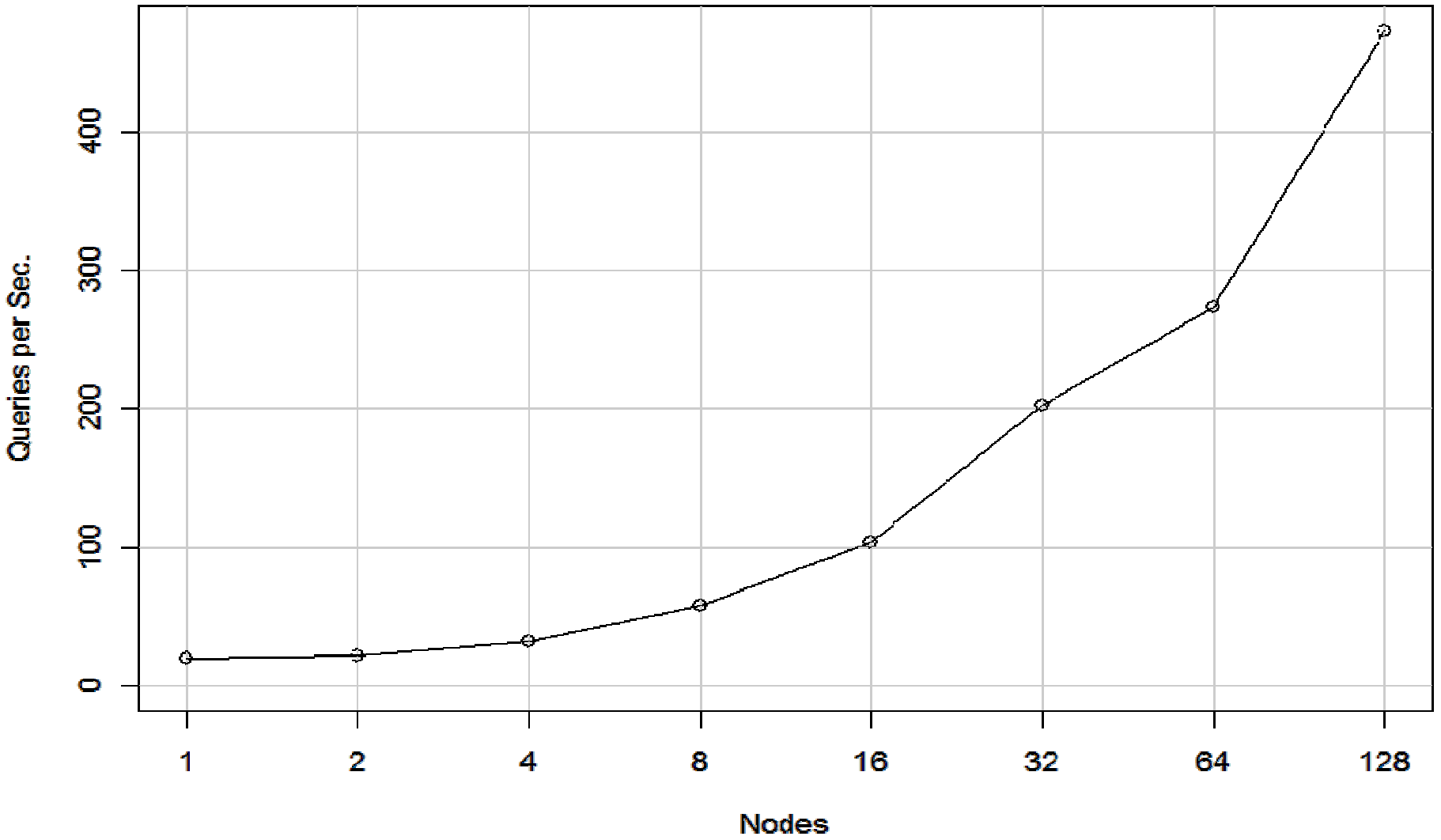
Caching und Look-Ahead



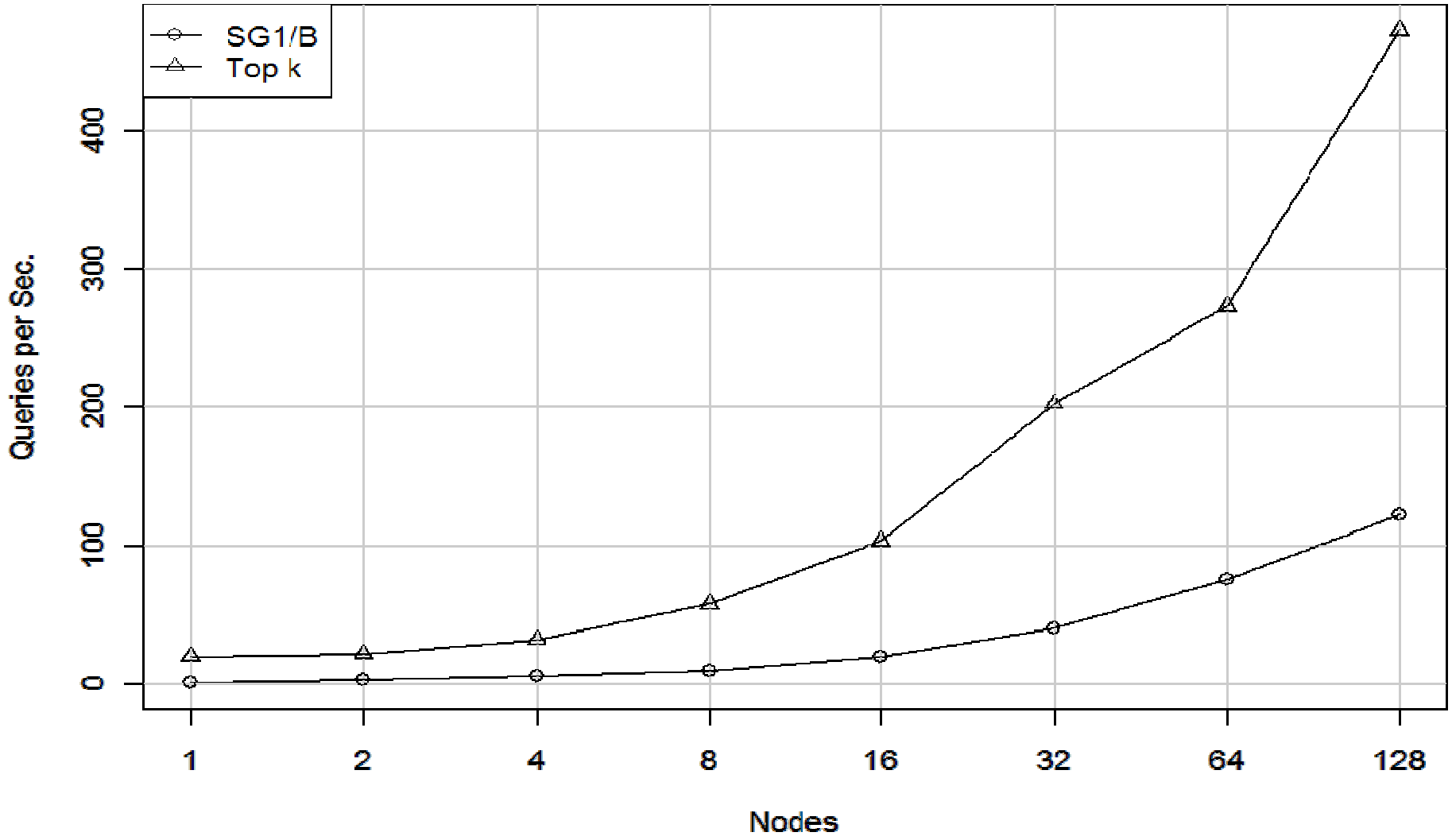
Ergebnisse: Vollständige Auswertung



Ergebnisse: Top k Auswertung



Vergleich


SG1/B
Top k
Queries per Sec.
0
100
200
300
400
1
2
4
8
16
32
64
128
Nodes

Zusammenfassung

- Skalierbare Architektur, basierend auf DHT
- Berücksichtigung **aller** Informationsquellen
- Verteilungsalgorithmus
  - Berücksichtigung der RDF Schema Regeln
- Algorithmen zur Query-Evaluierung
  - Vollständige Evaluierung
  - Top k Evaluierung
- Prototypische Implementierung
  - Benchmarks auf Cluster
- Ausblick
  - Umstellung auf SkipNet + Ganesan-Lastbalancierung
  - Mehr Semantik und automatische Schlußfolgerungen

Vorlesung
P2P Netzwerke

11: Multicast

Dr. Felix Heine

Complex and Distributed IT-Systems

felix.heine@tu-berlin.de

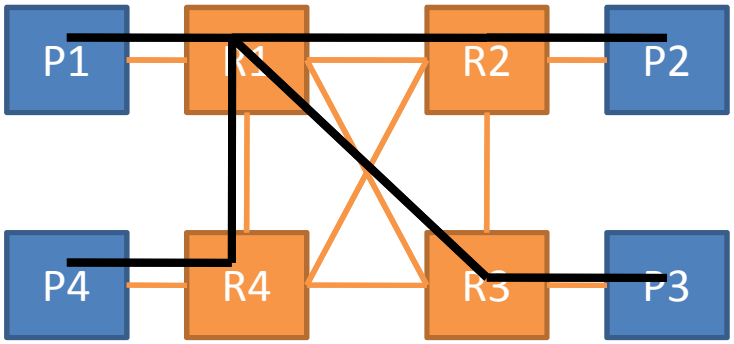
Inhalt

- Application-Layer Multicast
- Scribe
- SplitStream
- BitTorrent

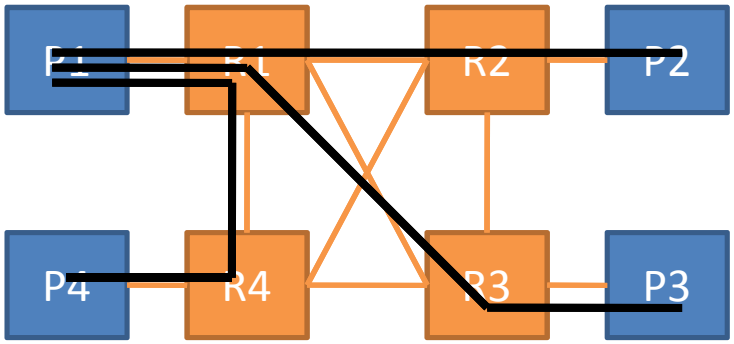
Multicast

- IP Multicast ist eine tolle Idee, aber
  - kaum unterstützt
  - daher nicht praktisch einsetzbar
- Alternativen?
- Application-Layer Multicast!

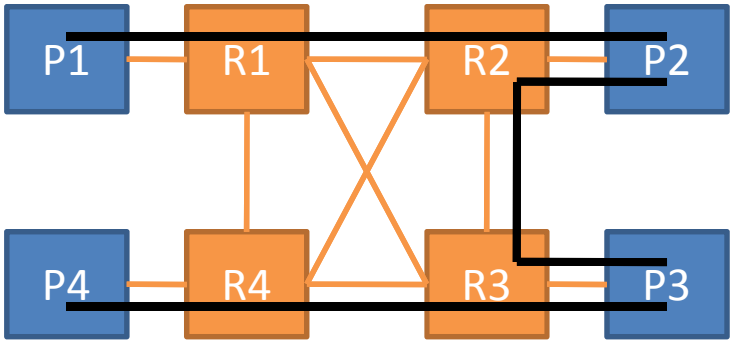
Application-Layer Multicast



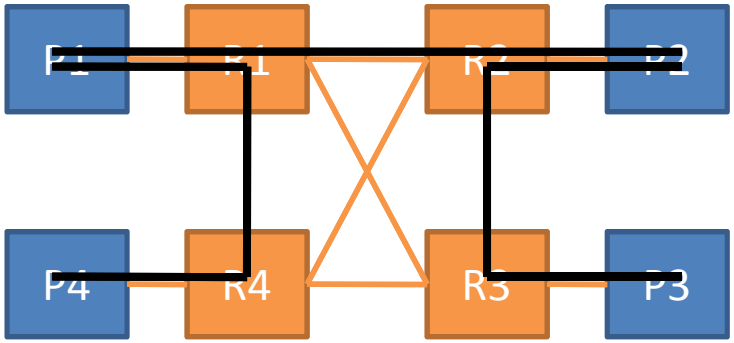
Max Stress = 1
Avg Stretch = 1



Max Stress = 3
Avg Stretch = 1



Max Stress = 1
Avg Stretch = 2



Max Stress = 2
Avg Stretch = 1,33

Scribe: Literatur

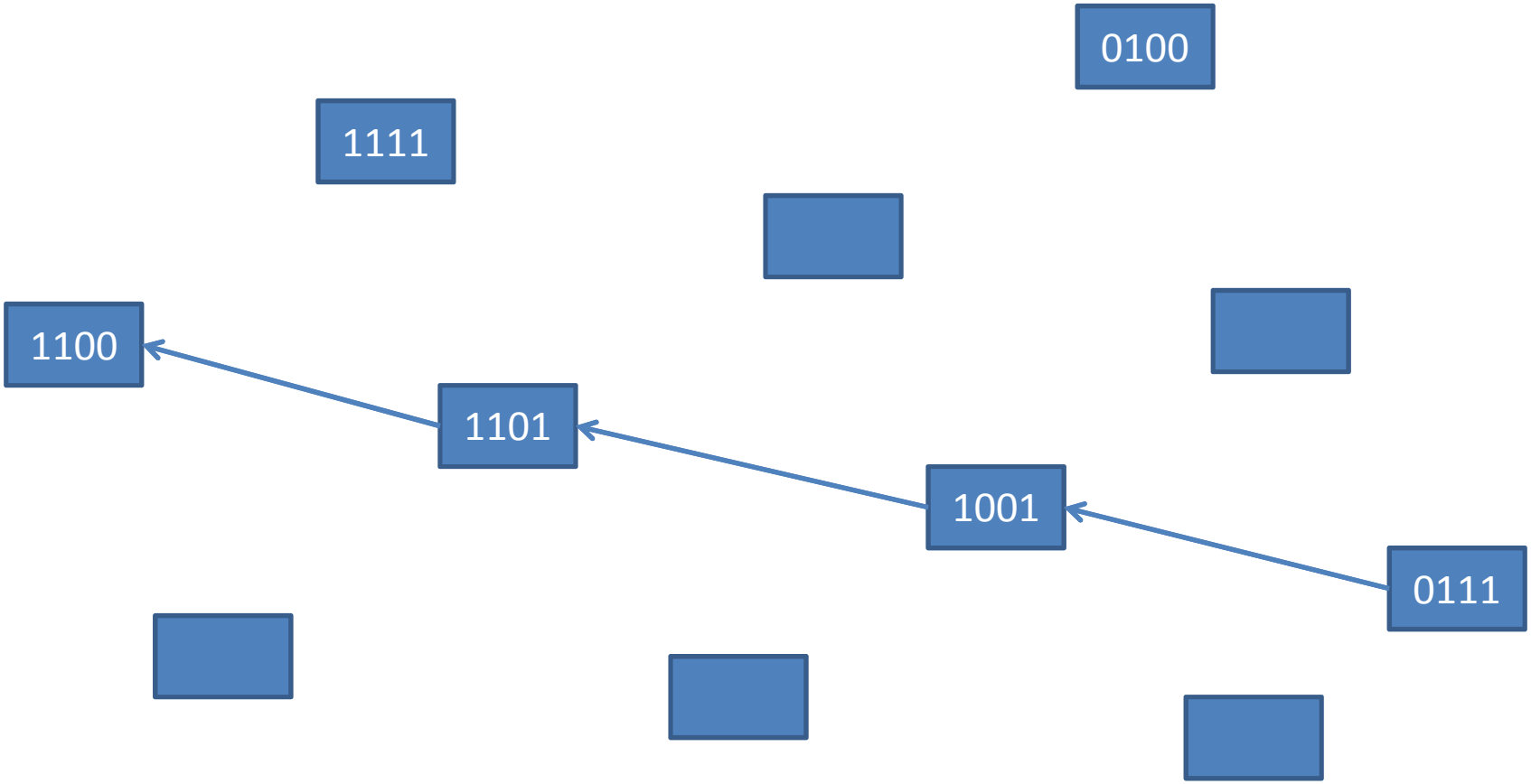
[1] M. Castro, P. Druschel, A-M. Kermarrec, A. Rowstron: "SCRIBE: A large-scale and decentralised application-level multicast infrastructure", IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications), 2002.

[2] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron, A. Singh: "SplitStream: High-bandwidth multicast in a cooperative environment", OSP'03,Lake Bolton, New York, October, 2003.

Scribe: Basis

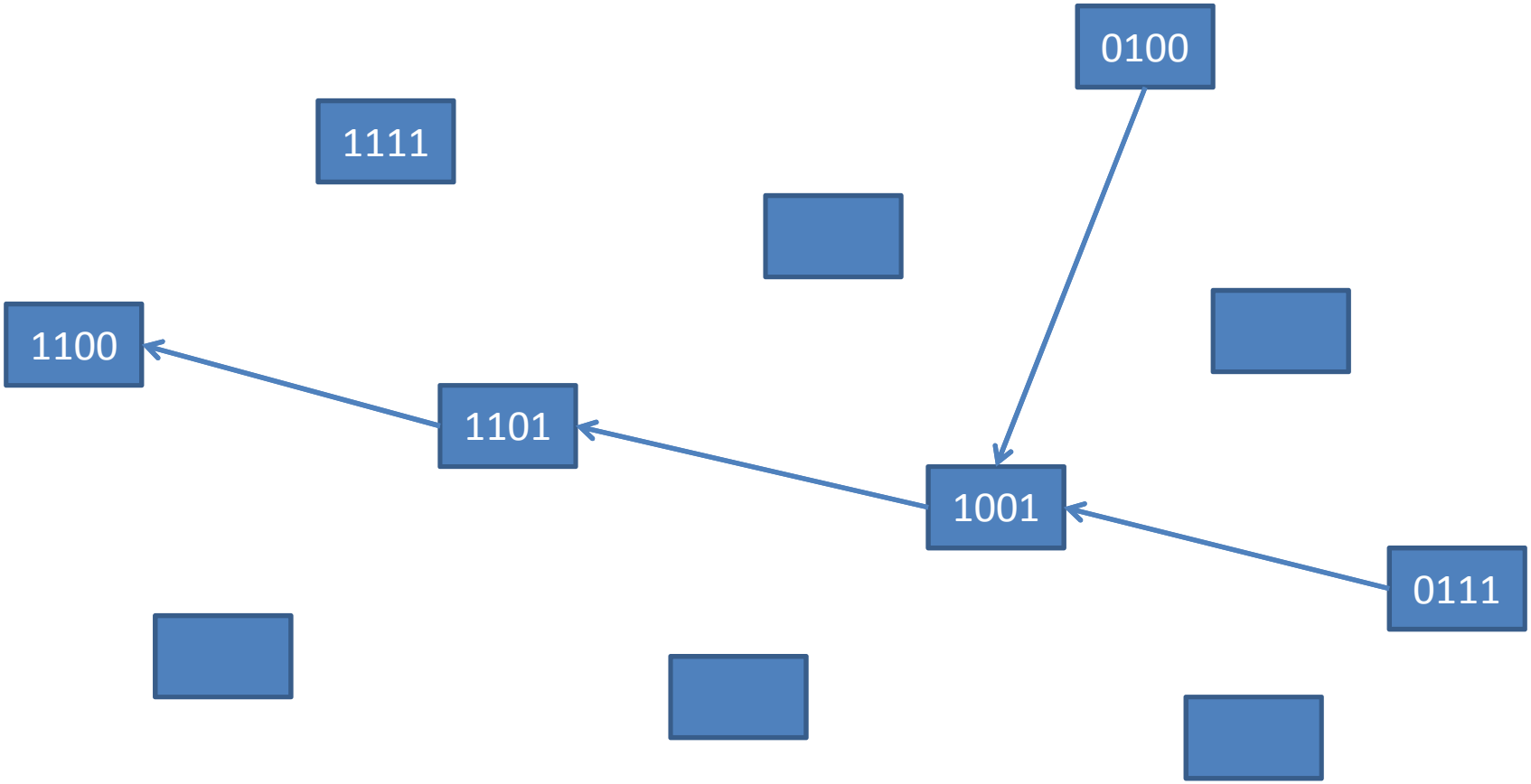
- Baut auf Pastry auf
- Multicast-Kommunikation in Gruppen
- Jede Gruppe hat eine ID grpID
- Der Root-Knoten für grpID in Pastry verwaltet die Gruppe
- Join von Knoten n:
  - Sende Nachricht JOIN an grpID
  - Jeder Knoten auf dem Weg überprüft, ob er in der Gruppe ist
  - Füge n zur Kinder-Liste für grpID hinzu
  - Nachricht wird nur weitergeleitet, wenn der Knoten noch nicht in der Gruppe war
- Es wird dadurch ein Baum für jede Gruppe aufgebaut

Scribe und SplitStream



Quelle: [1]

Scribe und SplitStream



Quelle: [1]

Scribe: Verlassen einer Gruppe

- Leave von Knoten n aus Gruppe grpID:
  - Wenn Kinder-Liste leer ist, route Leave-Message Richtung Root-Knoten
  - Sonst Markiere Gruppe nur als "verlassen"
- Wenn eine Leave-Nachricht von einem Kind empfangen wurde:
  - Lösche Kind aus Kinder-Liste
  - Wenn Kinder-Liste leer ist, und der Knoten selbst nicht in der Gruppe ist:
    - Sende Leave-Nachricht an Parent und lösche die Gruppe lokal

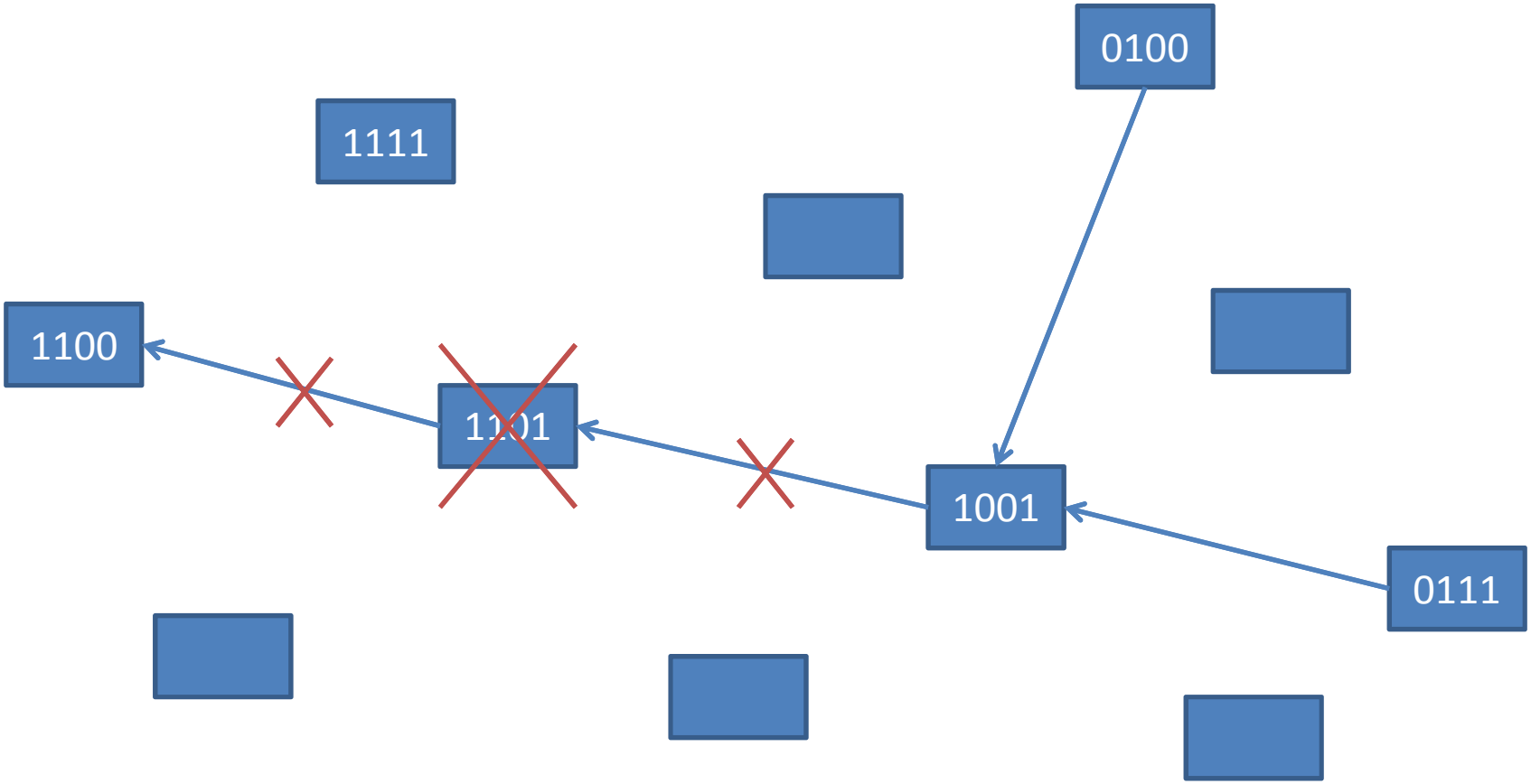
Scribe: Senden einer Nachricht

- Kanonische Lösung:
  - Nachrichten werden immer zur Wurzel geschickt
  - Dazu wird das NodeHandle gecacht
  - Wenn es falsch oder unbekannt ist, wird per Pastry-Routing die Wurzel gesucht
  - Von der Wurzel aus sendet jeder Knoten die Nachricht an seine Kind-Knoten
- Alternative:
  - Verteilen der Nachricht direkt vom aktuellen Knoten aus
  - Jede Nachricht beinhaltet eine Liste von Knoten, die die Nachricht bereits gesehen haben
  - Die Nachricht wird an den Parent-Knoten und an alle Kind-Knoten verschickt

Scribe: Reparatur

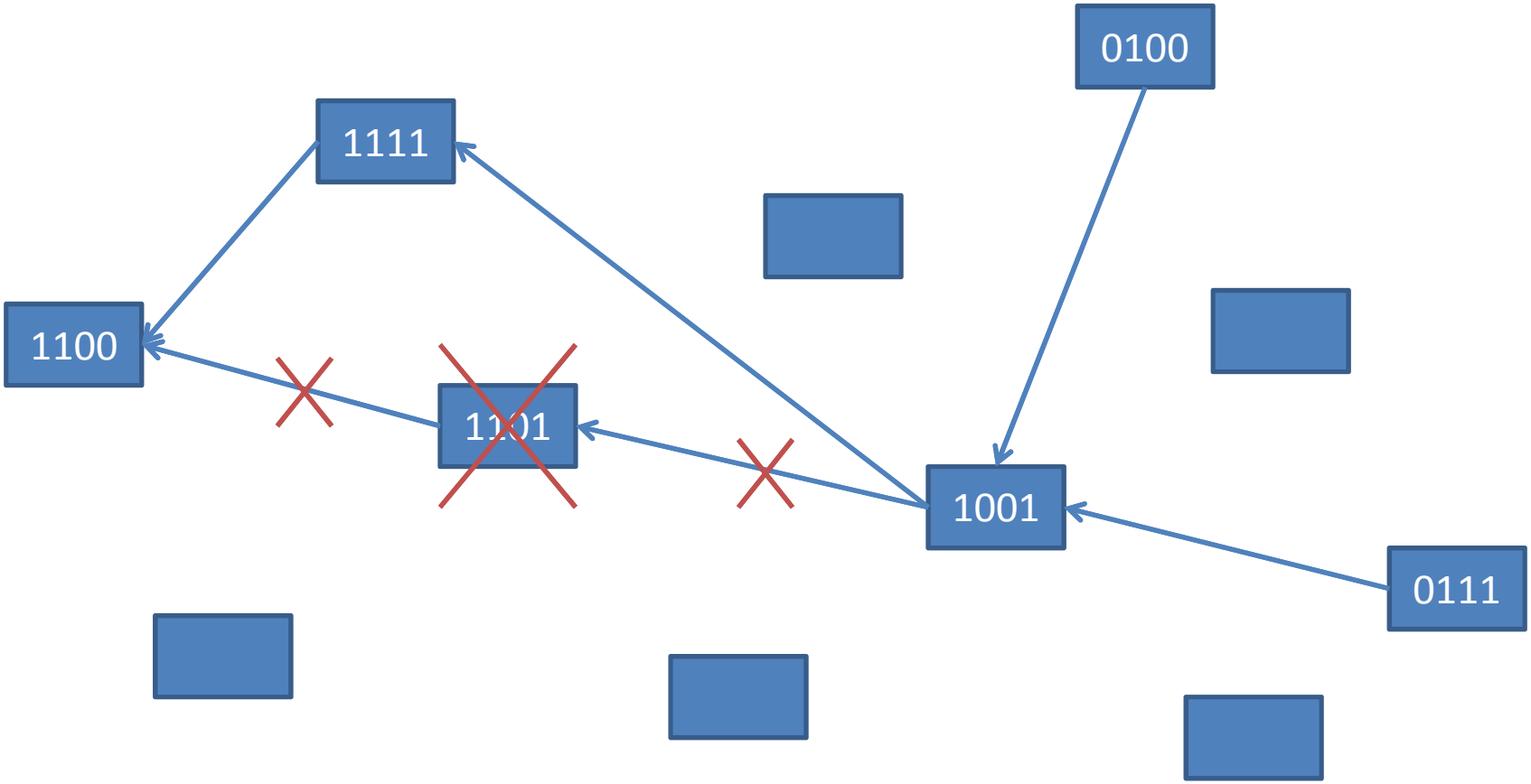
- Die Wurzel sendet regelmäßig "Heartbeat"-Nachrichten
- Normale Nachrichten sind implizit Heartbeats
- Wenn der Heartbeat vom Parent-Knoten ausbleibt, sendet er eine neue JOIN-Nachricht
- Dadurch wird der Baum repariert
- Die Wurzel repliziert ihren Zustand (wie gewohnt) über das Leafset
- Ein neuer Root-Knoten übernimmt direkt nach dem Ausfall der Wurzel das Verteilen und Aussenden der Heartbeats

Scribe



Quelle: [1]

Scribe

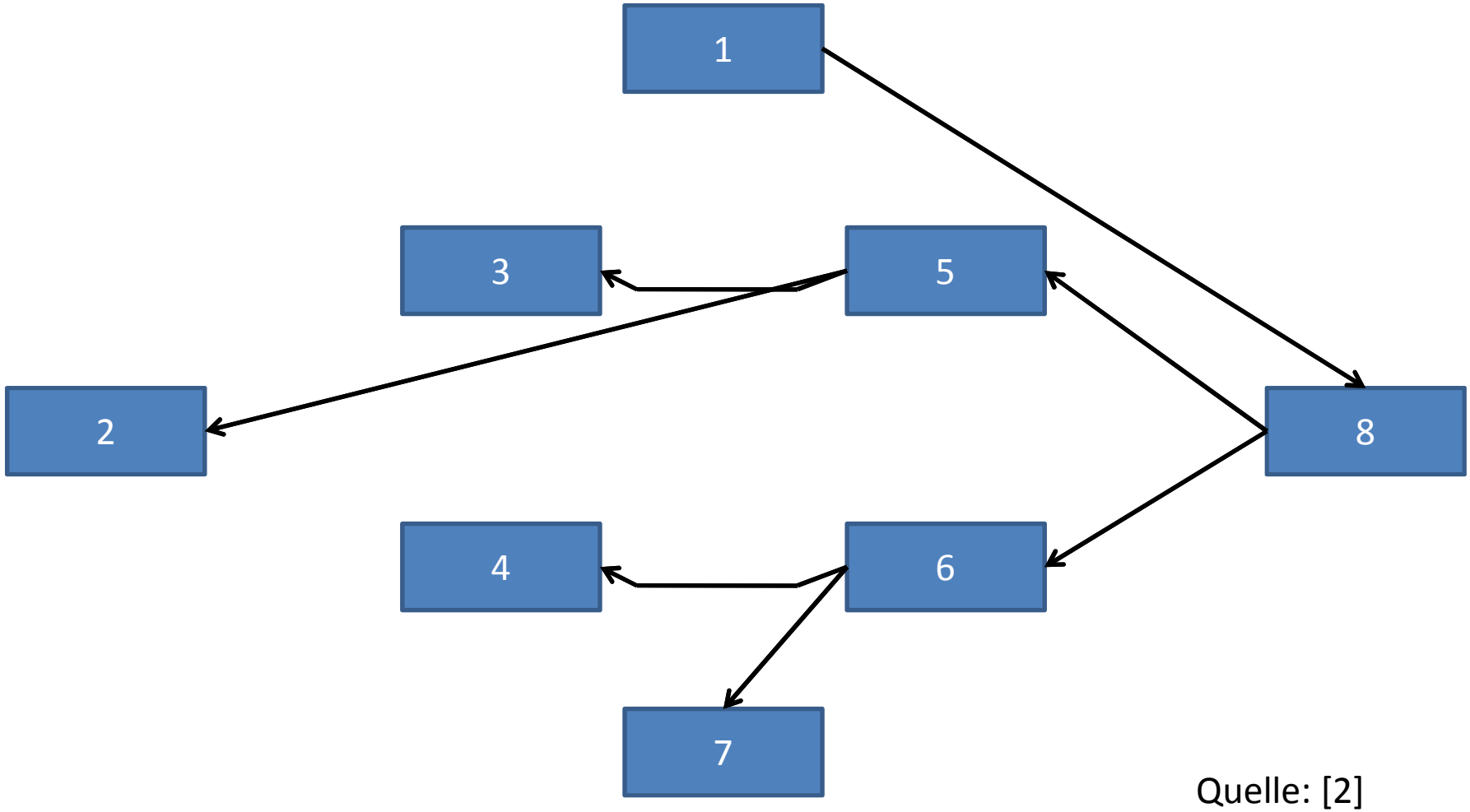


Quelle: [1]

Scribe: Probleme

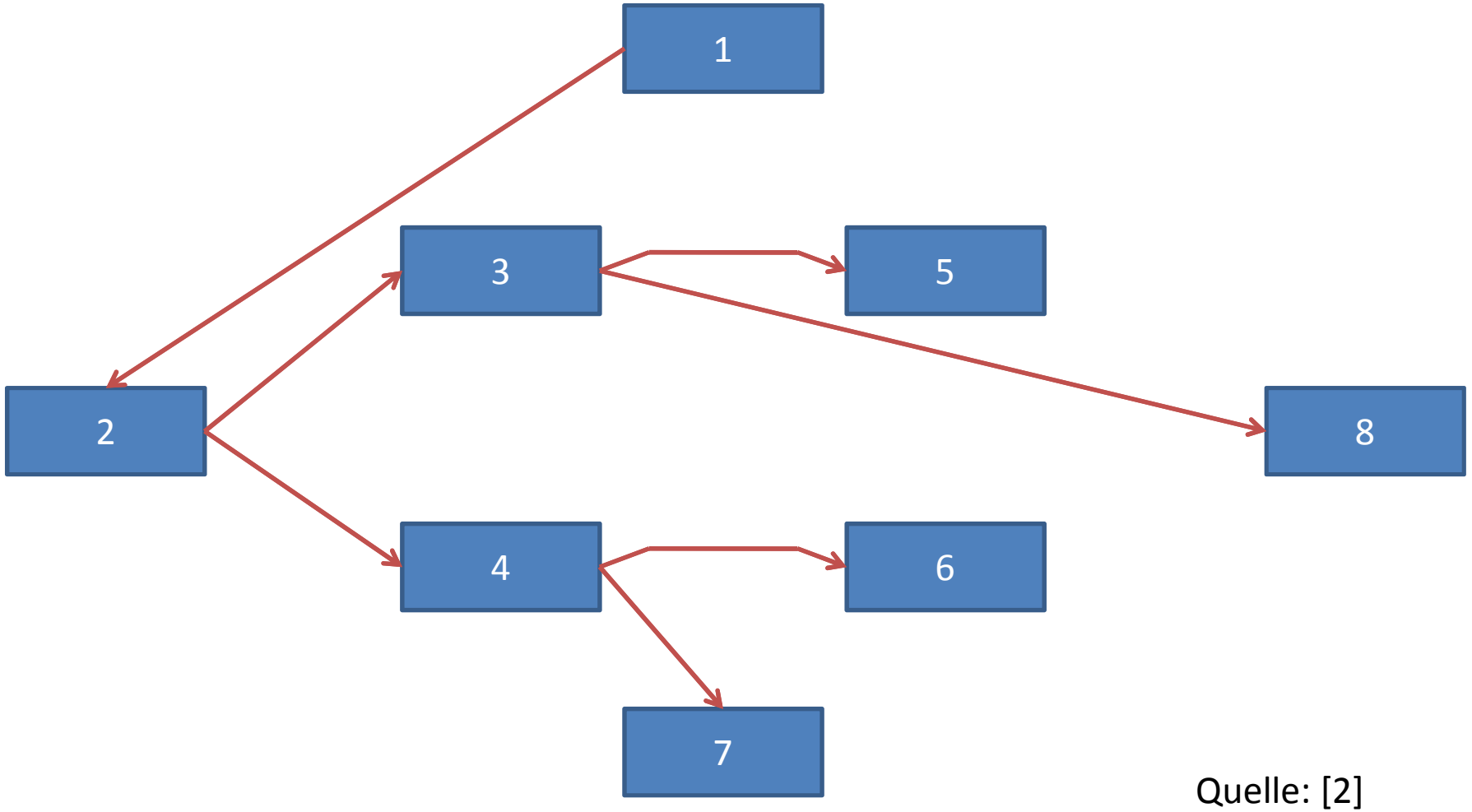
- Innere Knoten des Baumes müssen Weiterleiten
- Blattknoten müssen nur Empfangen
- Wieviele innere Knoten gibt es?
  - Fanout f, Höhe h, Balancierter Baum: $(f^h-1)/(f-1)$
- Last der inneren Knoten:
  - Ausgangs-Bandbreite f-mal der Datenstrom
- ➔ Sehr ungleiche Lastverteilung im Baum
- Lösung: **SplitStream**
- Grundidee:
  - Aufteilen des Datenstroms in mehrere Teilströme
  - Für jeden Teilstrom einen eigenen Baum aufbauen

SplitStream



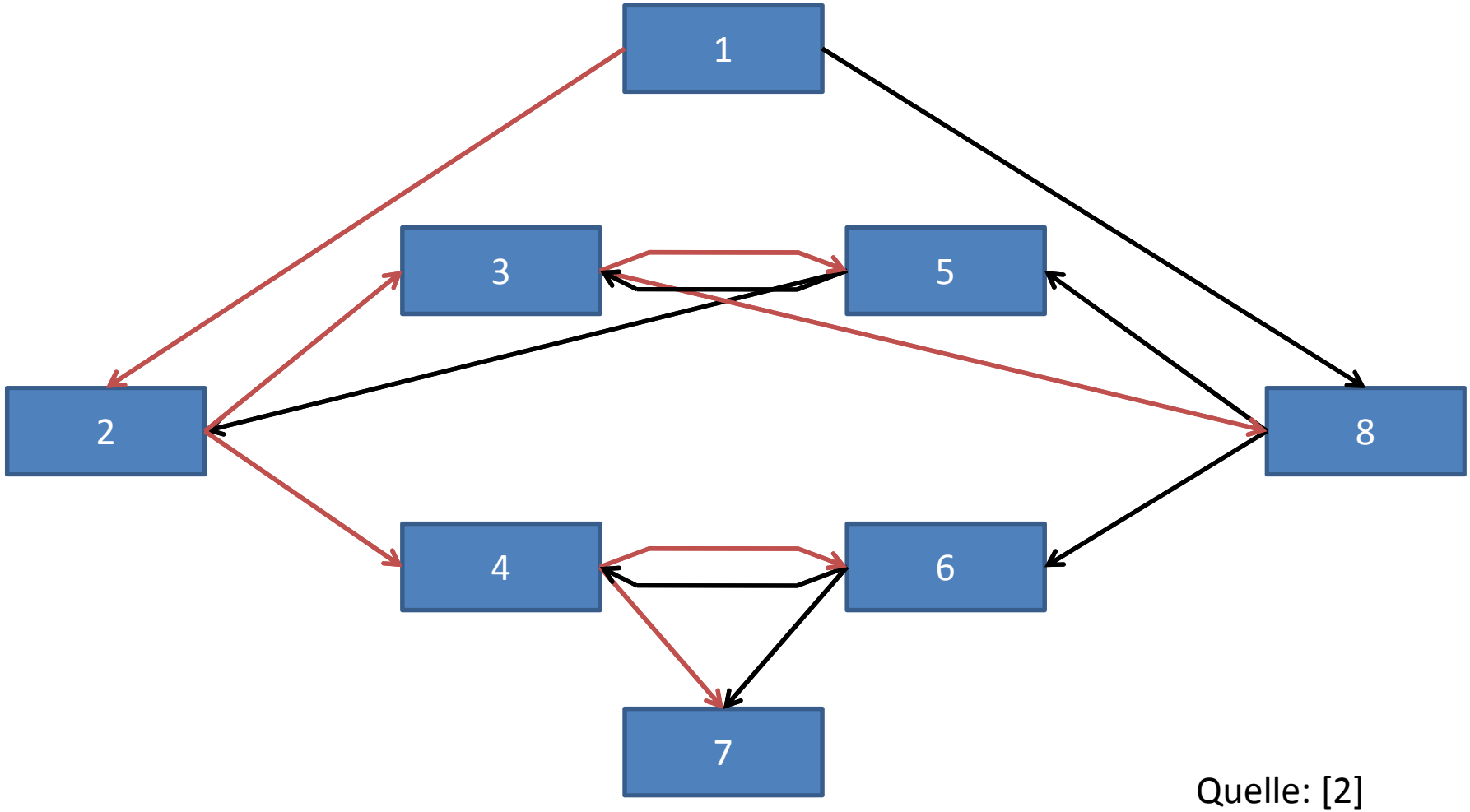
Quelle: [2]

SplitStream



Quelle: [2]

SplitStream



Quelle: [2]

SplitStream

- Datenstrom mit Bandbreite B wird verteilt
- Der Datenstrom wird in k Stripes eingeteilt
- Für jeden Stripe wird ein eigener Baum aufgebaut
- Es muss nicht jeder Knoten alle Stripes empfangen
  - Warum macht das Sinn?
- Jeder Knoten kann seine Upload- und Download-Bandbreite in Schritten von B/k kontrollieren
- Dadurch Berücksichtigung von:
  - Knoten mit unterschiedlicher Bandbreite
  - Ungleiche Up- und Download-Bandbreite

Aufbau der Bäume

- Jeder Stripe bekommt eigene ID
- Die ID's unterscheiden sich in der höchsten Ziffer
- Jeder Empfänger-Knoten meldet sich bei allen Stripe-IDs an
- Zu jeder ID wird ein eigener Scribe-Baum aufgebaut
- Je dichter ein Knoten an der Wurzel ist, desto länger ist der gemeinsame Präfix
- Ideal für diese Konstruktion ist: $k = 2^b$
- Jede Route von einem Knoten zu einer Stripe-ID stellt im ersten Schritt Übereinstimmung in der ersten Stelle sicher
- Die weitere Route verläuft nur über Knoten mit der gleichen ersten Stelle
- Alle Bäume haben also disjunkte Mengen innerer Knoten

Kontrolle der Bandbreite

- Die eingehende Bandbreite entspricht der Menge der abonnierten Stripes, oder
- max. 1, wenn kein Stripe abonniert wurde, aber der Knoten innerer Knoten eines Stripe-Baumes ist
- Aber: die ausgehende Bandbreite könnte überschritten werden
- Methode in *Scribe* zur Kontrolle:
  - Wenn ein Knoten einen neuen Kind-Knoten bekommt, der die Bandbreite überschreitet:
  - Sende dem Knoten bisherige Kind-Liste
  - Der Knoten versucht sich bei dem Kind mit der geringsten Latenz anzumelden
  - Dies wird rekursiv fortgesetzt
- Diese Methode führt in SplitStream zu Problemen!

Kontrolle der Bandbreite

- Jeder Knoten nimmt jedes neue Kind zunächst an
- Wenn seine Bandbreite überschritten wird:
  - Selektiere das Kind mit dem kürzesten gemeinsamen Präfix
  - Hänge diese Kind ab
- Das verwaiste Kind muss sich einen neuen Parent-Knoten suchen
  - Zunächst rekursiver Versuch, sich bei den anderen Kindern des alten Parent-Knoten einzuhängen
  - Dabei werden nur Kinder mit zum Stripe passenden Präfix genutzt
  - Sonst nutze die "Spare capacity group"
- Spare Capacity Group
  - Scribe-Gruppe in der alle Knoten mit freien Kapazitäten sind

Spare Capacity Group

- Tiefensuche im Baum der Spare Capacity Group
- Wenn Knoten gefunden, der den Stripe empfängt:
  - Prüfe, ob kein Kreis entstehen würde – wie kann das sein?
  - Dazu speichert jeder Knoten den ganzen Pfad zur Wurzel
- Wenn kein Knoten gefunden, der den Stripe empfängt:
  - Kapazität des Netzwerkes ist erschöpft ☹
- Wenn nur zyklische Knoten gefunden
  - Suche beliebigen Blatt-Knoten im Baum des Stripes der nicht Kind des aktuellen Knotens ist
  - Dieser tauscht mit dem aktuellen Knoten
  - Der ehemalige Blatt-Knoten wird über die SCG eine neue Position finden

Wie verwende ich SplitStream?

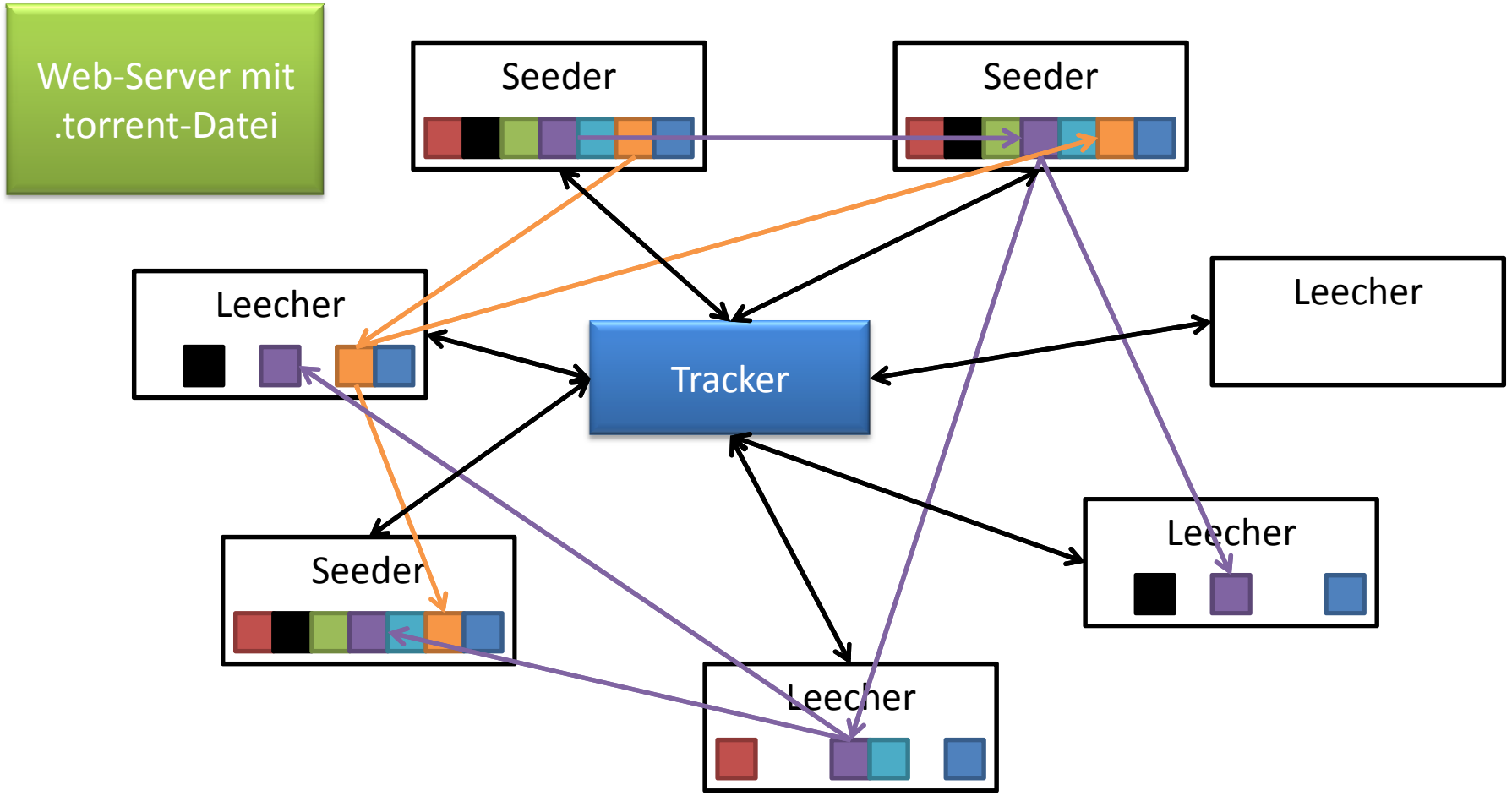
- Jeder einzelne Stripe könnte "Aussetzer" haben:
- Reparaturzeiten nach Knotenausfall, Umorganisation, etc.
- Daher: Einsatz von fehlertoleranter Kodierung
  - Empfang aller k Stripes ist nicht nötig
  - Ausfall einzelner Stripes kann kompensiert werden

BitTorrent: Literatur

[1] Bram Cohen: "Incentives Build Robustness in BitTorrent", 2003.

[2] Ashwin R. Bharambe, Cormac Herley, Venkata N. Padmanabhan: "Analyzing and Improving BitTorrent Performance", Microsoft Research Technical Report MSR-TR-2005-03, 2005.

BitTorrent


Web-Server mit .torrent-Datei
Seeder
Seeder
Leecher
Tracker
Leecher
Seeder
Leecher
Leecher

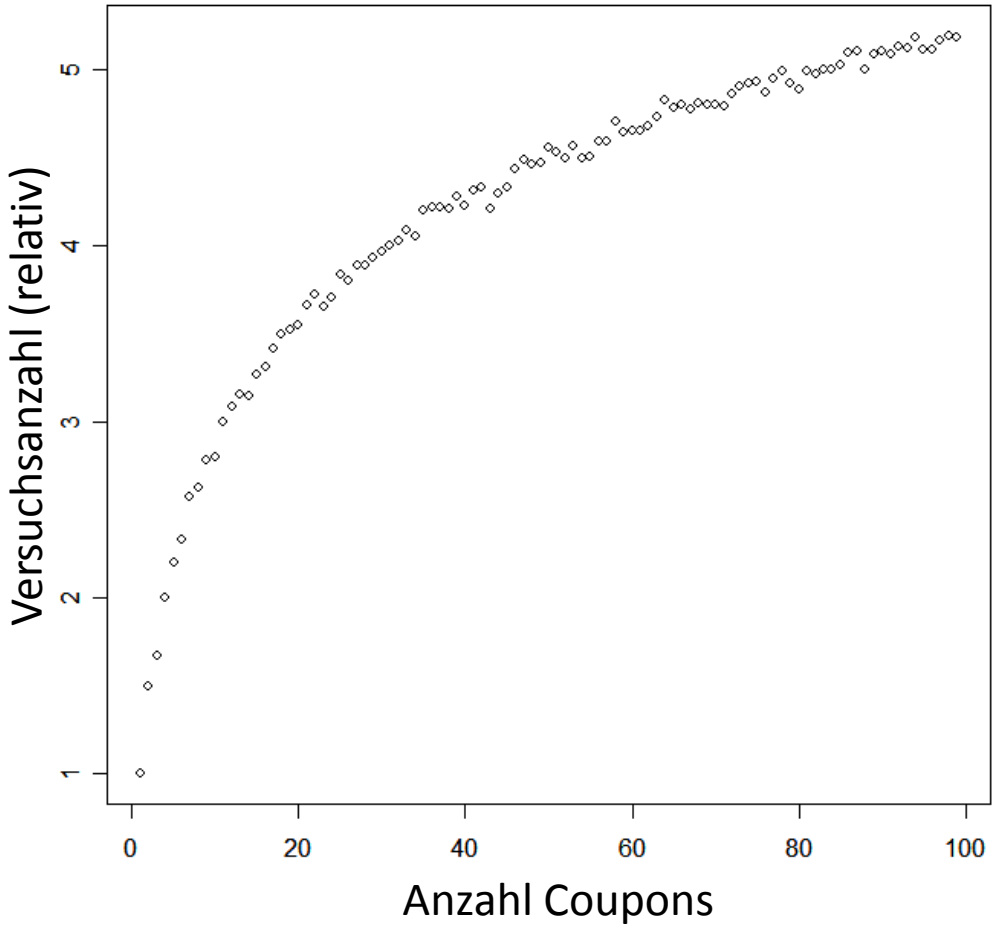
BitTorrent: Funktionsweise

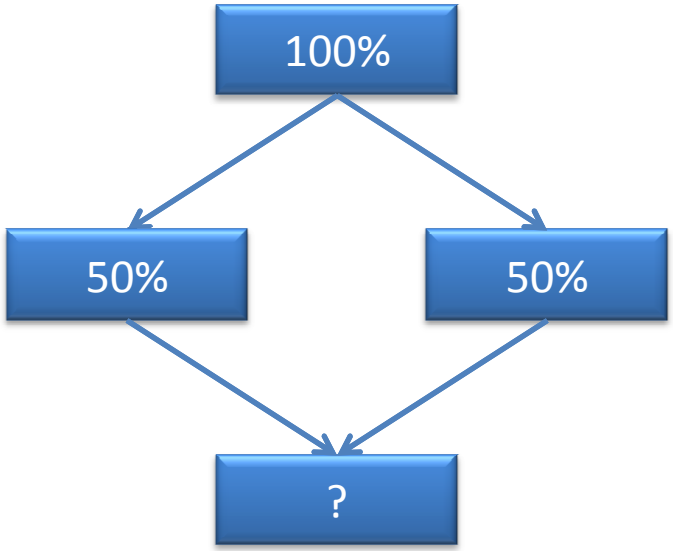
- Ein Peer lädt eine Torrent-Datei von einem Webserver
  - Dateiinfos: Name, Länge etc.
  - URL des Trackers
  - SHA1-Werte der Dateiblöcke
- Es muss mind. einen Seeder anfänglich geben
- Ein Peer meldet sich beim Tracker an
- Der Tracker schickt eine zufällige Auswahl an Peers (ca. 40)
- Der Peer baut Direktverbindungen zu diesen Peers auf
- Der Peer lädt jetzt die Datei blockweise von seinen Nachbar-Peers runter
- Wenn ein Peer weniger als 20 Nachbarn hat, holt er neue vom Tracker

BitTorrent: Details

- Versendung der Blöcke über TCP-Verbindungen
- Jeder Block wird in Sub-Blöcke aufgeteilt (16 KB)
- Diese werden im Pipelining-Verfahren verschickt
- Bevor ein neuer Block angefordert wird, werden erst fehlende Sub-Blöcke von alten Blöcken angefordert
- Welches Paket wird als nächstes angefordert?
  - "Rarest First": Paket, welches am wenigsten repliziert ist unter den bekannten Peers
  - ➔ Coupon Collector Problem
  - "Random First Piece": Erstes Paket wird zufällig gewählt
  - ➔ Um schnell an Daten zum Weiterverteilen zu kommen
  - "Endgame Mode": Am Ende alles von überall anfordern

Coupon Collector: Beispiele





BitTorrent: Tit for Tat

- Wie verhindere ich, dass jemand nicht weiterverteilt?
- Grundsätzliche Idee:
  - Ich schicke nur denen etwas, die auch selbst etwas weiterverteilen
- Probleme:
  - Peers, die noch nichts zum Verteilen haben
  - Peers, die eine schlechte Anbindung haben, und von denen daher niemand etwas runterladen möchte
  - Woher bekomme ich die Informationen?

BitTorrent: Choking

- Eine Verbindung kann aktiv oder passiv sein (choking)
- Es sind immer nur 4 Verbindungen aktiv (unchoking)
- 3 werden über die besten Download-Raten bestimmt
  - 20 sec. rollierender Durchschnitt
- Alle 10 sec. werden die aktiven Verbindungen neu gewählt
  - wegen TCP ramp up
- Zusätzlich: Ein "Optimistic Unchoke"
  - Eine zufällige Verbindung wird aktiviert
  - Für neue Peers und solche mit schlechter Upload-Bandbreite
  - Rotiert alle 30 sec.
- Wenn keine Pakete mehr ankommen
  - Mehr als ein Optimistic Unchoke
- Wenn Download komplett
  - Wähle aktive Peers anhand Upload-Rate

Zusammenfassung

- Problemstellung: Multicast, d.h. Verteilung von Daten an eine große Empfängergruppe
- IP-Level Multicast wäre optimal
- hat sich aber nicht durchgesetzt
- Daher: Application-Level Multicast
- Mit P2P gut zu realisieren
- Prinzip: Baumförmige Struktur aufbauen
- Probleme: Fairness sowie gute Ausnutzung der vorhandenen Kapazitäten
- Lösungen: Daten in mehrere Teile zerhacken, für jeden Teil eigenen Baum aufbauen
- Systeme: Scribe, SplitStream, BitTorrent

Vorlesung
P2P Netzwerke
12: Sicherheit

Dr. Felix Heine
Complex and Distributed IT-Systems
felix.heine@tu-berlin.de

Inhalt

- Sicherheit
  - Grundlagen
  - Modell
  - Ziel
- Vergabe von Node-ID's
- Sicheres Befüllen von Routing-Tabellen
- Sicheres Routing
  - Fehler-Test für das Routing
  - Redundantes Routing

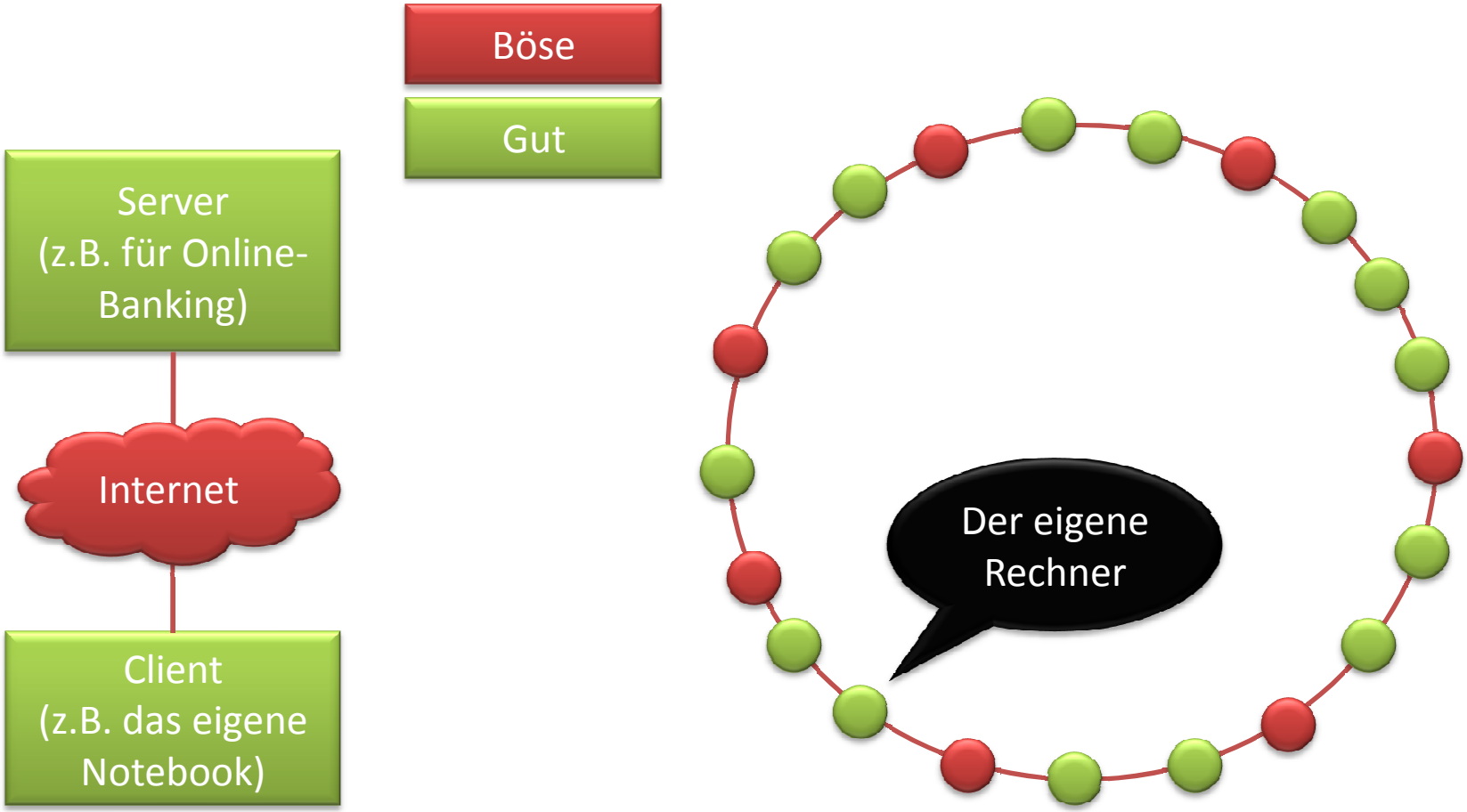
Sicherheit: Grundlagen

- In P2P können alle bekannten Sicherheits-Technologien zum Einsatz kommen:
  - Symmetrische und asymmetrische Kryptographie
  - Public-Key Verfahren
  - Digitale Unterschriften
  - Zertifikate
  - CA-Hierarchien
- Es gibt aber zusätzliche Problemen gegenüber Client/Server Systemen
- Das ist Thema dieser Vorlesung:
  - Spezifische Sicherheitsprobleme von P2P Netzwerken

Literatur

- Emil Sit and Robert Morris: "Security Considerations for Peer-to-Peer Distributed Hash Tables", 1st International Workshop on Peer-to-Peer Systems, 2002.
- John R. Douceur: "The Sybil Attack", 1st International Workshop on Peer-to-Peer Systems, 2002.
- Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, Dan S. Wallach: "Secure routing for structured peer-to-peer overlay networks", Proc. of the 5th Usenix Symposium on Operating Systems Design and Implementation, 2002.
- Nikita Borisov: "Computational Puzzles as Sybil Defenses", Sixth IEEE International Conference on Peer-to-Peer Computing, 2006.

Modelle


Böse
Gut
Server
(z.B. für Online-
Banking)
Internet
Client
(z.B. das eigene
Notebook)
Der eigene
Rechner

Modell

- System mit N Knoten
- Anteil von $0 \leq f \leq 1$ bösen Knoten
- Böse Knoten verhalten sich beliebig und operieren in Koalitionen von max. cN Knoten.
- Allgemein: $c \leq f$, wenn $c = f$ dann sind alle bösen Knoten eine große Koalition
- Unterscheide:
  - Network-Level Kommunikation
  - Overlay-Level Kommunikation
- Grund-Absicherung durch Kryptographie

Was soll erreicht werden?

- Ein bösartiger Knoten könnte Daten, für die er verantwortlich ist:
  - Modifizieren, Löschen, alte Kopien ausliefern, etc.
- Er könnte geroutete Pakete löschen, ändern, etc.
- Daher:

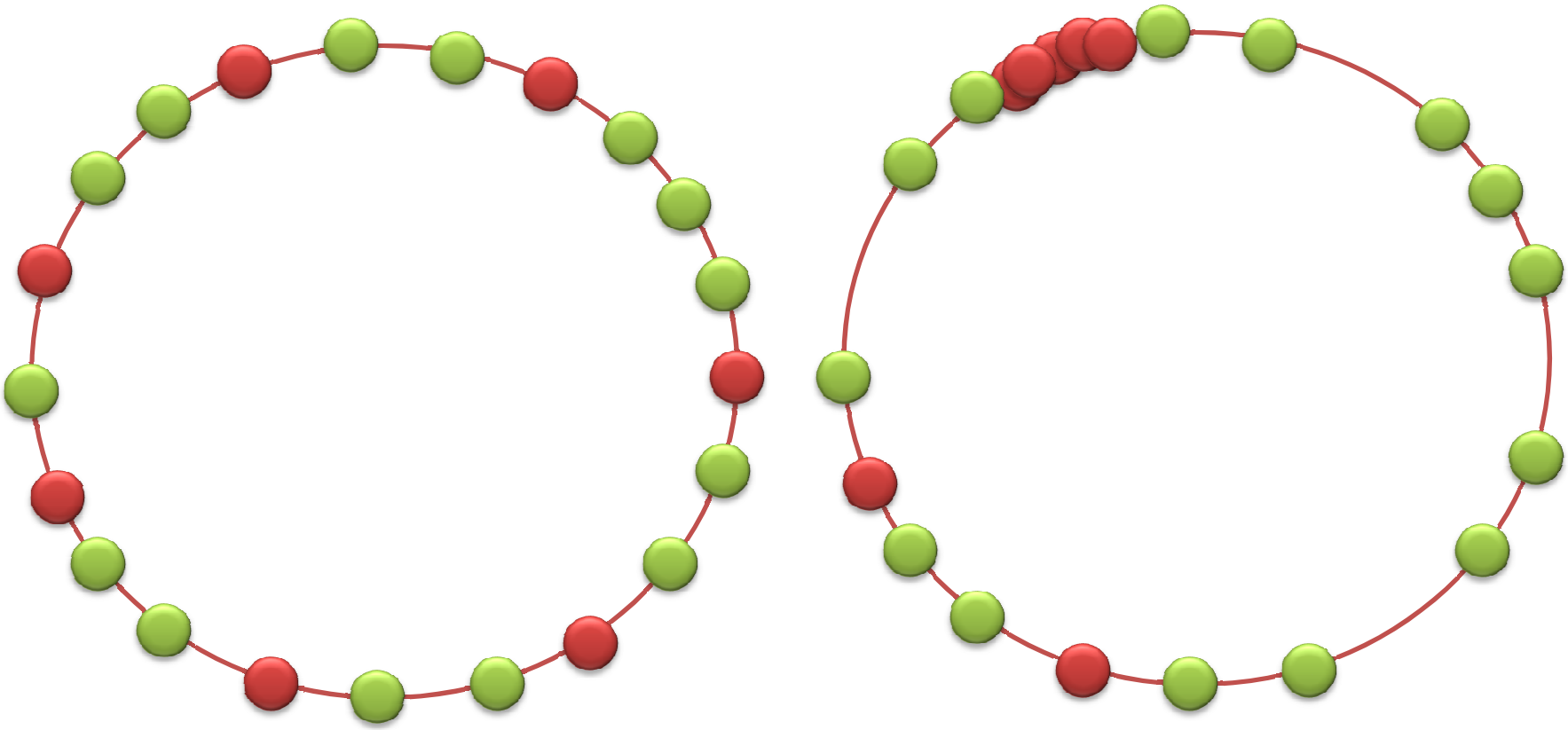
Sicheres Routing

Sicheres Routing stellt sicher, dass eine Nachricht, die ein gutartiger Knoten an den Schlüssel k sendet, mit hoher Wahrscheinlichkeit alle gutartigen Replika-Knoten für k erreicht.

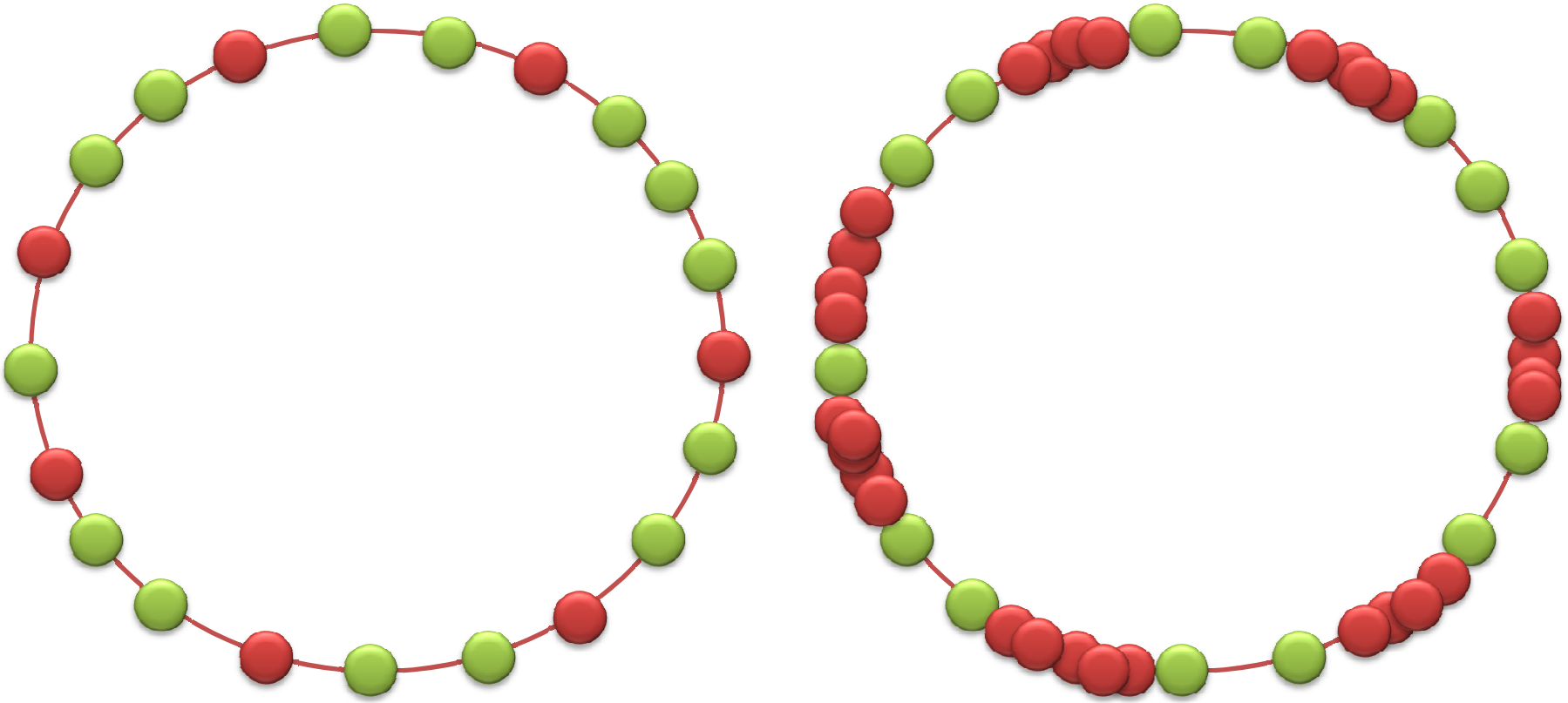
Schritte zum sicheren Routing

- Sicheres Vergeben von Knoten-IDs
  - Verhindere, dass ein Angreifer seine Knoten-ID selbst auswählen kann
- Sicheres Bestücken der Routing-Tabellen
  - Stelle sicher, dass der Anteil von bösartigen Knoten in jeder Routing-Tabelle nicht größer ist als der Anteil bösartiger Knoten im gesamten Netzwerk
- Sicheres Weiterleiten von Nachrichten
  - Stelle sicher, dass mindestens eine Kopie einer Nachricht jeden gutartigen Replika-Knoten für einen Schlüssel erreicht

Angriffe I: Besetzen einer ID



Angriffe II: Sybil



Node-ID Vergabe: Angriffe

- Koalition besetzt bestimmtes ID-Gebiet (inkl. aller Replika-Knoten)
  - Koalition kann dann ein bestimmtes Datenelement kontrollieren
- Koalition aus Knoten versucht, alle Einträge in der Routing-Tabelle eines Opfers zu füllen
  - Koalition kontrolliert dann sämtliche ausgehenden Nachrichten des Opfers (Eclipse-Attacke)
- Sybil Attacken: Einzelner Knoten nimmt über viele virtuelle Knoten unter vielen IDs am Netzwerk teil
  - Gleiche Angriffe wie oben, aber auch ohne Koalition

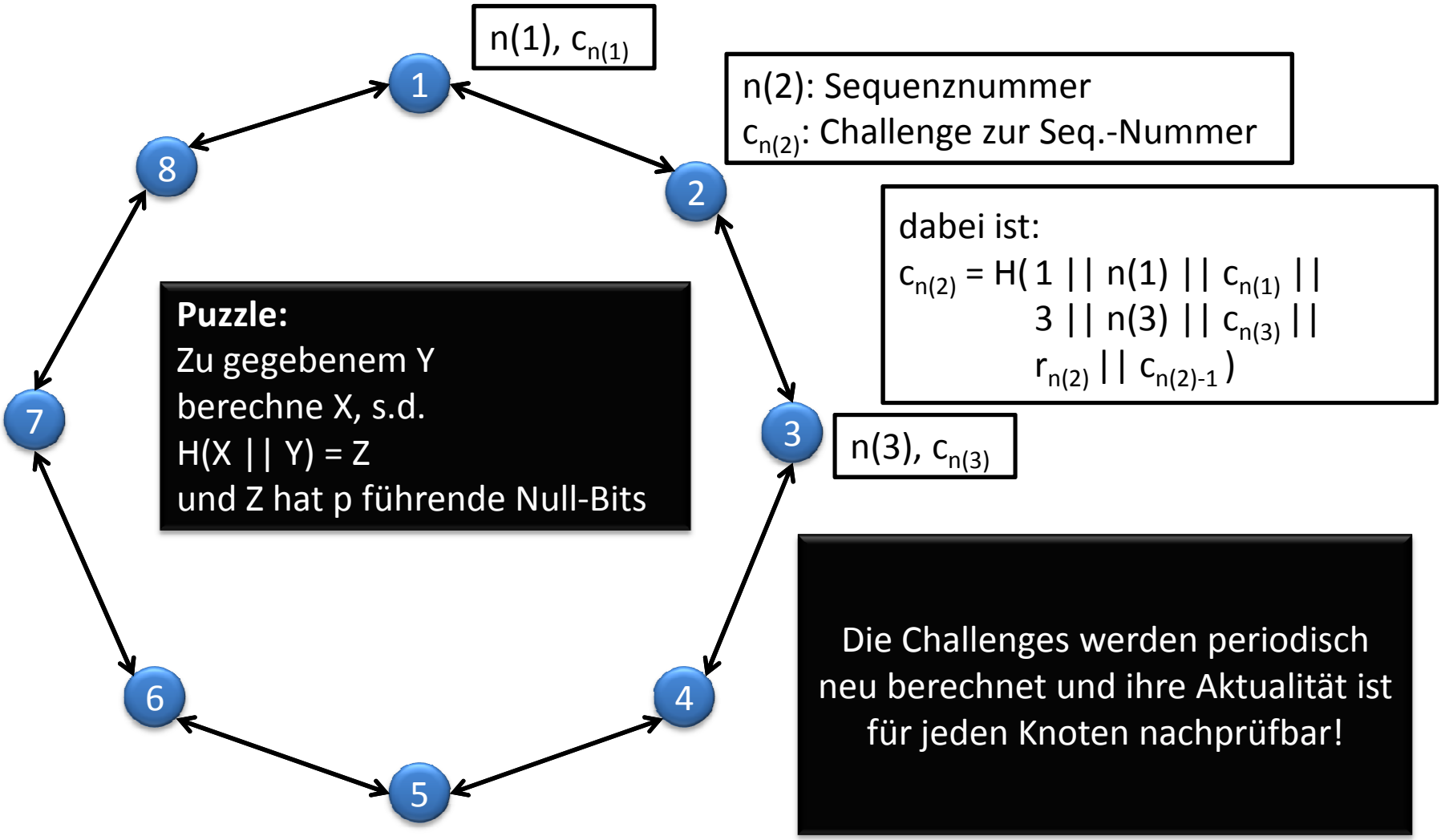
Sichere Node-ID Vergabe

- Externe Infrastruktur mit Certificate Authorities
- CA wählt eine zufällige Knoten-ID
- Es wird ein Zertifikat erstellt mit:
  - Öffentlichen Schlüssel des Knotens
  - ID des Knotens
  - IP-Adresse des Knotens
- Einbindung der IP-Adresse nötig
  - Sonst könnten Zertifikate zwischen unterschiedlichen Knoten getauscht werden, um sicherzustellen, dass der Knoten überall lokal erscheint
  - Bereitet Schwierigkeiten bei DHCP
- Fälschung von IP-Adressen ➔ Rückweg testen

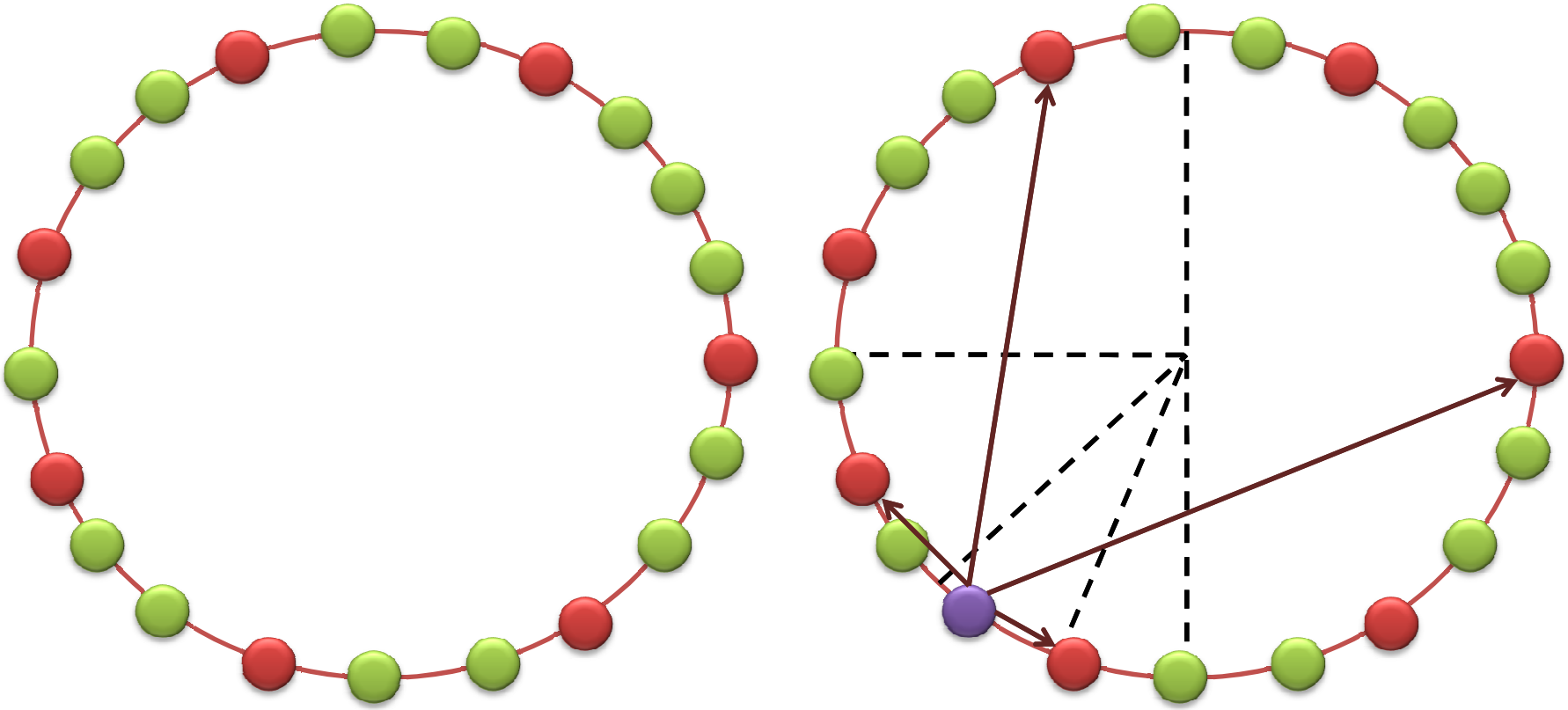
Abwehr der Sybil-Attacke

- Ein Knoten könnte sich immer noch sehr viele Zertifikate mit vielen unterschiedlichen Knoten-IDs besorgen
- Lösung 1: Zertifikate müssen bezahlt werden
- Lösung 2: Zertifikate an Objekte der realen Welt binden (Person, Organisation)
- Lösung 3: (umstritten, aber auch ohne Zertifikat möglich)
  - Ausgabe eines Challenge, dessen Beantwortung Rechenzeit kostet
  - Muss regelmäßig wiederholt werden
  - Verschwendet aber CPU-Zeit
  - Wie berücksichtige ich Heterogenität?
  - Aber: Heterogenität führt immer zu Angriffsmöglichkeiten

Computational Puzzles



Eclipse-Angriff



Routing-Tabelle: Eclipse Attacke

- Versuche, die Links eines Peers auf böse Peers umzulegen
- Damit wird das echte Netz vor einem attackierten Peer verborgen, er sieht nur noch böse Peers
- Das geht um so einfacher, je mehr Freiheiten ein Knoten hat, seine Nachbarn auszuwählen:
  - Gnutella: Nachbarn können beliebig gewählt werden
  - Pastry: Nachbarn können eingeschränkt gewählt werden
  - Chord: Nachbarn sind vollständig determiniert
- Dementsprechend ist Gnutella am anfälligsten

Beispiel

- Suche neuen Nachbarn aus den Nachbarlisten der aktuellen Nachbarn (insgesamt X Knoten)
- Anteil böser Knoten ist f, Anteil guter ist g=1-f
- Ein böser Knoten schickt nur böse Knoten als potentielle Nachbarn raus
- Über die bösen erhält man fX böse Knoten
- Über die guten Knoten erhält man insgesamt gX Neue
  - darunter fgX böse Konten
  - und ggX = g^2X gute Knoten
- Der Anteil guter Knoten ist also von g auf g^2 gesunken!
- ➔ Der Anteil böser Knoten ist jetzt deutlich größer!
- Beispiel: 90% gutartiger Knoten wird zu:
  - 81%, 66%, 43%, 19%, 3%

Routing-Tabelle: Abwehr

- Je beschränkter die Routing-Tabelle, desto schwerer der Angriff
- Beispiel: Bei Chord gibt es für jeden Eintrag der Routing-Tabelle nur einen passenden Knoten
- Daher ist Chord per se gut gewappnet
- Diese Abwehrstrategie verhindert aber PNS
- Lösung: Zwei Routing-Tabellen
  - Eine beschränkte Tabelle
    - pro Eintrag nur eine Möglichkeit
    - In Pastry-Knoten i, Eintrag auf Ebene l, Ziffer d:
      dichtester Knoten zu einer ID p mit
      p hat gemeinsames Präfix der Länge l mit i
      p hat Ziffer d an Position l+1
      p hat die nächsten Ziffern wieder identisch mit i
  - Eine unbeschränkte Tabelle
    - Ausnutzung von Lokalität

Befüllen der beschränkten Routing-Tabelle

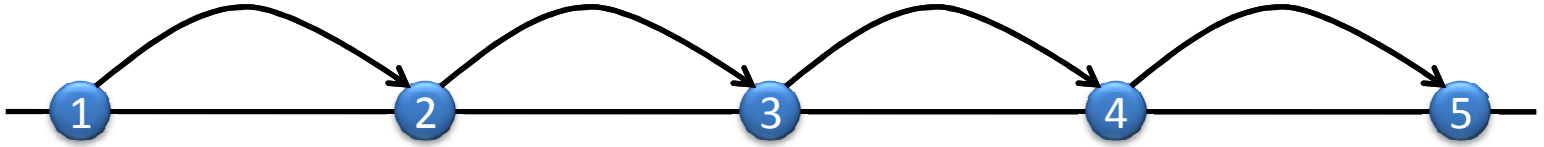
- Einfache, aber teure Methode:
  - Sicheres Routing nutzen, um Einträge neu zu füllen
- Menge von Boot-Knoten verwenden
  - Groß genug, dass mindestens ein gutartiger Knoten enthalten ist
- Routing-Tabellen von allen Boot-Knoten holen
- Jeden Slot mit dem besten Knoten füllen, der die Bedingungen erfüllt
- Von diesen Knoten wiederum die Tabellen holen
- Einträge dann eventuell durch bessere Knoten ersetzen
- Stabilisierung:
  - Regelmäßige Kommunikation mit den Nachbarn

Sicheres Routing

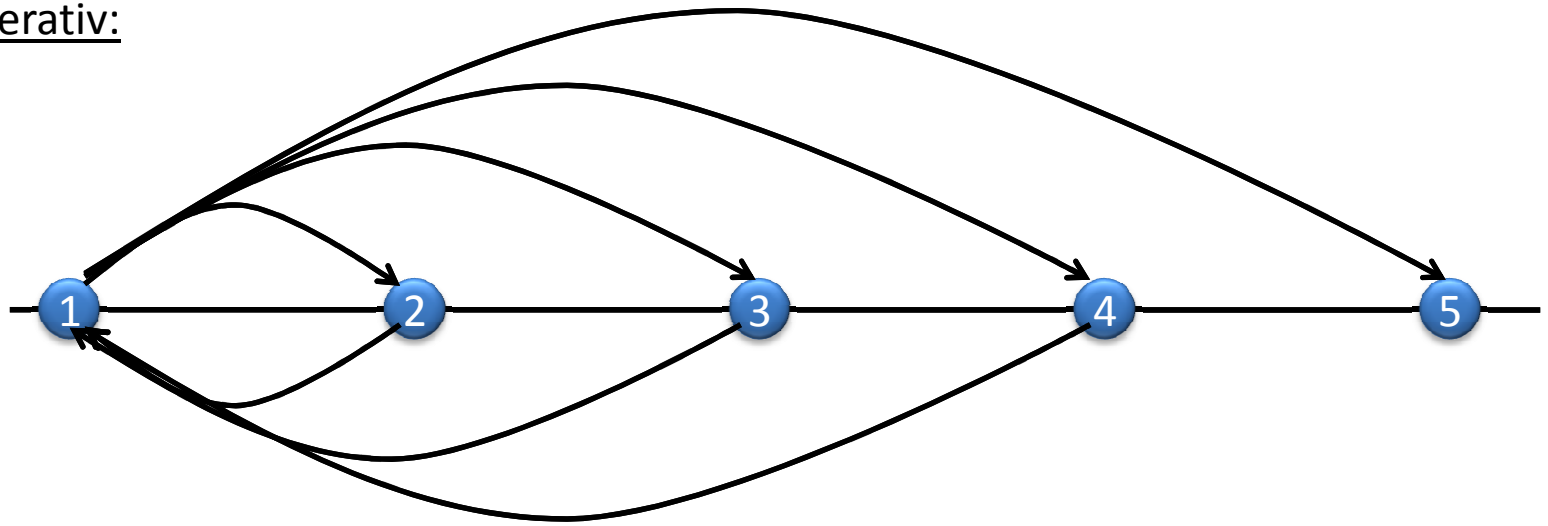
- Die bisherigen Maßnahmen garantieren:
  - Jede beschränkte Routing-Tabelle beinhaltet nur einen Anteil von f bösartigen Knoten
- Aber: ein bösartiger Knoten auf einer Route ist genug!
  - Er könnte behaupten, dass er der Root-Knoten ist
  - Er könnte die Nachricht stillschweigend löschen
  - Er könnte sie an einen anderen (falschen) Knoten weiterleiten
- Wahrscheinlichkeit für erfolgreiches Routing: $(1-f)^{h-1}$
  - h ist dabei die Anzahl Knoten auf dem Pfad
- Der Root-Knoten selbst könnte bösartig sein
  - Verwendung von Replika-Knoten

Iteratives Routing

Rekursiv:



Iterativ:



Sicheres Routing: Lösung

- Zunächst Standard-Routing nutzen (für Effizienz, d.h. Ausnutzung von Lokalität)
- Der Root-Knoten antwortet mit einer Menge von Replika-Knoten
- Jetzt wird ein Fehlertest angewendet
- Wenn der Fehlertest anschlägt:
  - Verwende redundantes Routing
  - Es werden mehrere Kopien der Nachricht auf verschiedenen Routen an die unterschiedlichen Replika-Knoten gesendet
  - Dies ist teuer, aber die Erfolgswahrscheinlichkeit ist höher

Fehler-Test für das Routing

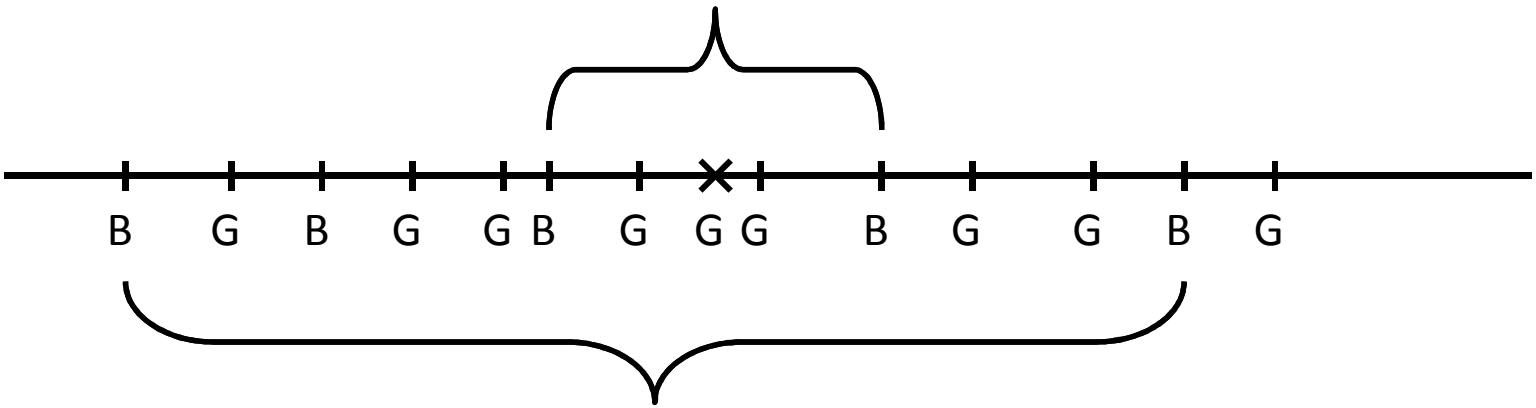
- Wenn keine Antwort nach einem Timeout zurückkommt, antwortet der Test positiv
- Sonst: Eingabe für den Test ist der Ziel-Schlüssel plus eine Menge von potentiellen Replika-Knoten
- Beobachtung:
  - Die durchschnittliche Dichte von Knoten pro ID-Distanz ist höher als die durchschnittliche Dichte von bösartigen Knoten
- Daher wird verglichen:
  - Dichte der Knoten in der Nachbarschaft des aktuellen Knoten
  - Dichte der Replika-Knoten
- Wenn ein Knoten nur bösartige Knoten aus seiner Koalition zurückliefert, ist die Dichte geringer als die normale Dichte

Fehler-Test für das Routing

B: Bösartiger Knoten

G: Gutartiger Knoten

Korrekte Menge von Replika-Knoten



Von bösartigem Knoten zurückgelieferte Menge

Dichte ist viel geringer

Redundantes Routing I

- Sende r Nachrichten über verschiedene Nachbarn an das Ziel.
- Jede Nachricht beinhaltet ein nonce.
- Es wird nur die beschränkte Tabelle verwendet.
- Wenn ein Knoten die Nachricht erhält, der Nachbar von x' Root-Knoten ist, schickt er sein NodeId-Zertifikat und das nonce, unterschrieben mit seinem Privaten Schlüssel, zurück.
- Der Original-Knoten sammelt die zu x dichtesten Node-ID-Zertifikate in einer Liste N, aber nur wenn sie gültig unterschieben sind. Alle stehen zunächst auf "Pending".

Redundantes Routing II

- Nach einem Timeout oder nachdem r Antworten kamen, sendet der Knoten die Liste N an jeden Knoten, der noch auf "Pending" steht.
- Jeder Knoten leitet die Originalnachricht an Nachbarn weiter, die nicht in der Liste stehen. Ansonsten sendet er eine Bestätigung, dass es keine weiteren Knoten gibt. Die neuen Knoten kommen auch auf die Liste.
- Wenn von jedem Knoten eine Bestätigung erhalten wurde, oder 3 Runden vorbei sind, wird aus N die Menge der Replika-Knoten gebildet.

Zusammenfassung

- P2P hat besondere Sicherheitsanforderungen im Vergleich zu Client/Server
- "Standard"-Sicherheitsmaßnahmen sind notwendig, aber nicht hinreichend
- Man muss immer mit bösartigen Knoten rechnen
- Das gesamte Netzwerk soll auch dann noch funktionieren, wenn ein gewisser Anteil der Knoten bösartig ist
- Typische Angriffe:
  - Sybil: Viele virtuelle Knoten
  - Eclipse: Verdecken des Netzes durch bösartige Knoten
- Abwehrmaßnamen sind möglich, aber teuer