Lecture 12

Internet Protokolle II

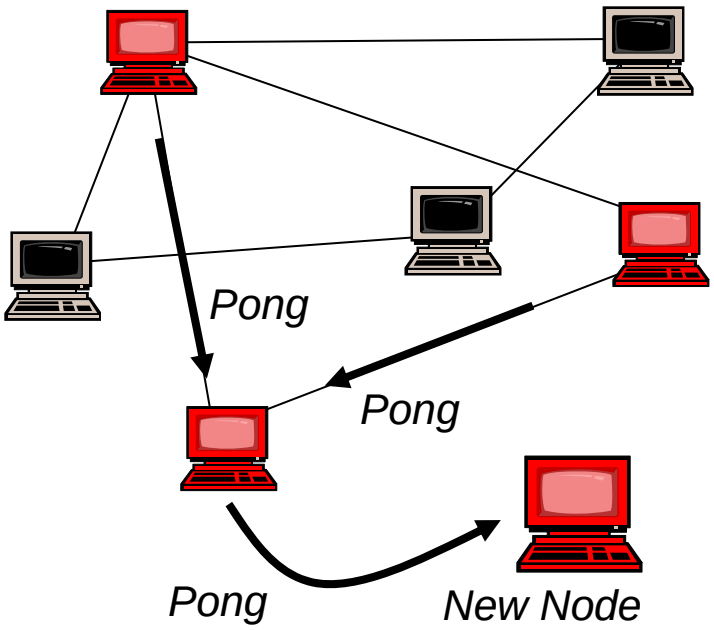
Unstructured & Structured
Peer-to-Peer Networks

Thomas Fuhrmann

Network Architectures
Computer Science Department
Technical University Munich

Unstructured P2P Networks

Gnutella Network

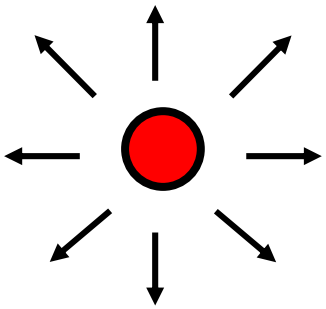


Unstructured P2P networks can be described as random graphs such as

- Erdös-Renyi
  (=equally distributed random edges, not observed in the wild),
- Barabasi-Albert
  (=preferential attachment),
- Adamic-Huberman
  (=gaining links over time), or
- Watts-Strogatz
  (long range link in almost regular structure).

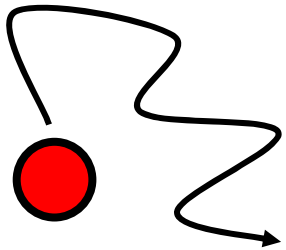
Small world networks are sparse graphs with small diameter and high cluster coefficient. Typically, the node degrees have a power-law distribution.

Search in Unstructured P2P Networks



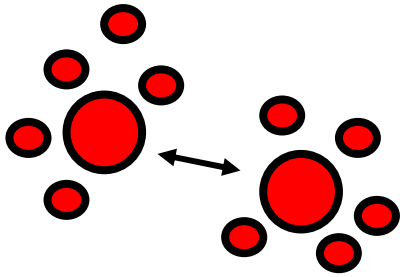
Simple Flooding

- Simple, straight forward
- Needs Message IDs or path record to avoid loops
- Very high message volume
- Robust in face of message loss and sudden topology changes



Random walk

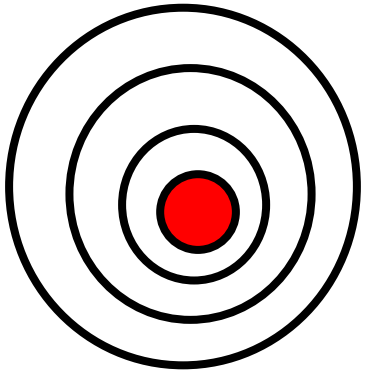
- Depth-first search
- Low message overhead, but long search latency
- In practice only up to 80% success probability (for individual items)
- Comes in different flavors, e.g., high-degree seeking



Super peers

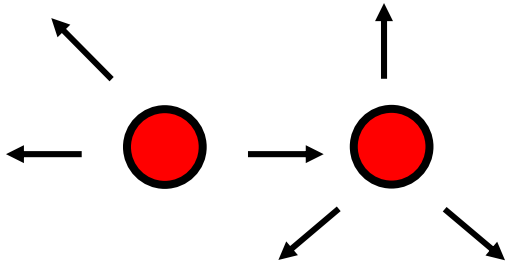
- Each peer is attached to (exactly) one super peer.
- Only super peers actually search.

Search in Unstructured P2P Networks



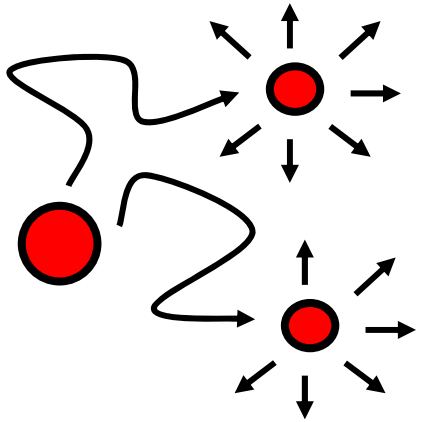
Expanding ring search

- Reduces traffic volume for searching nearby or highly replicated items.
- Trying hard to find an item results in more traffic than simple flooding



Gossiping

- Probabilistic spread of a request
- Reduced message volume (as compared to flooding)
- Reduced request latency (as compared to random walk)



Gossiping & Flooding

- Increase the spread probability beyond a given distance threshold.
- Avoids effect of high cluster coefficient: High coverage & low message overhead.

Super Peers

- Ein wesentlicher Punkt beim günstigen Ergebnis der Random Walk Analyse von Adamic et al. ist die Fähigkeit einiger weniger, stark vernetzter Knoten auch über ihre zweiten Nachbarn Auskunft zu geben.
- D.h. Random Walk und High-Degree Seeking verlagern Last von den vielen Knoten kleinen Grades auf die wenigen Knoten hohen Grades.
- Eine solche Heterogenität ist in der Realität aber häufig sowieso gegeben.
- Man kann sie auf zwei Arten nutzen:
  1. Neuere Versionen von Gnutella nutzen eine ungleiche Belastbarkeit der Knoten dadurch aus, dass leistungsstarke Knoten zu so genannten ***Super Peers*** erklärt werden.
     - Anfragen werden im Overlay-Netz der Super Peers geflutet.
     - Jeder Super Peer gibt Auskunft über die mit ihm verbundenen normalen Peers.
  2. Das Gianduia-System erweitert diese Idee und ordnet seine Knoten auf einer kontinuierliche Leistungsskala an.

Gianduia

- Das Gianduia-Filesharing-System (kurz „Gia“) ersetzt das Fluten von Anfragen durch ein High-Degree-Seeking Random Walk
- Kreditbasierte Flusskontrolle: Knoten informieren ihre Nachbarn, wieviele Anfragen sie bearbeiten können.
  - Ziel: Knoten nicht überlasten, damit Anfragen möglichst rasch durchs Netz laufen
  - Jeder Knoten bestimmt dazu seine Kapazität (= Zahl der Anfragen, die er bearbeiten kann) und teilt diese beim Verbindungsaufbau seinen Nachbarn mit.
  - Jeder Knoten verteilt seine Kredite proportional der Kapazität seiner Nachbarn auf diese.
- Anfragen werden an den Knoten mit der höchsten Kapazität weitergeleitet für den ein Knoten noch Kredite hat.
- Knoten speichern Index der Inhalte ihrer direkten Nachbarn.

Quelle: Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, Scott Shenker, *Making Gnutella-like P2P Systems Scalable*,SIGCOMM, August 2003.

Gia-Topologieformung (1)

1. Bootstrapping: Wie bei Gnutella über Web-basierte Caches
2. Lokaler Cache jedes Knotens wird immer aktualisiert
   - neue Einträge mittels Ping-Pong gewonnen
   - Unerreichbare („tote“) Knoten entfernen
3. Neue Overlay-Kante:
   - Zufällige Auswahl einiger Knoten aus dem lokalen Cache treffen:

     Fall 1: Auswahl enthält mindestens einen Knoten, dessen Kapazität größer als die eigene ist. Dann nimm den Knoten aus der Auswahl, der die größte Kapazität hat.

     Fall 2: Alle Knoten der Auswahl haben geringere Kapazität als die eigene. Dann nimm einen zufälligen Knoten aus der Auswahl.
   - Nun baue Verbindung auf. Der Zielknoten kann diese Verbindung ablehnen, falls dadurch seine maximale Verbindungszahl überschritten werden würde.

Gia-Topologieformung (2)

- Zur Entscheidung betrachtet der Knoten alle seine bisherigen Nachbarn: Er sucht den Nachbar mit der größten Kapazität, die gerade noch kleiner ist als die des anfragenden Knotens.

  Fall 1: Der Grad dieses Knotens ist kleiner als der Grad des anfragenden Knotens. Dann lehne den Verbindungswunsch ab.

  Fall 2: Andernfalls trenne die existierende Verbindung und akzeptiere die neue Verbindung.

  Bemerkung: Dadurch steigt die Summe der Kapazitäten der eigenen Nachbarn, ohne dass der minimale Grad der drei am Vorgang beteiligten Knoten sinkt!

4. Ein Knoten baut so lange weitere Verbindungen auf,
   - bis die Summe der anteiligen Kapazitäten* aller seiner Nachbarn seine eigene Kapazität übersteigt, oder
   - seine maximale Verbindungszahl erreicht ist.

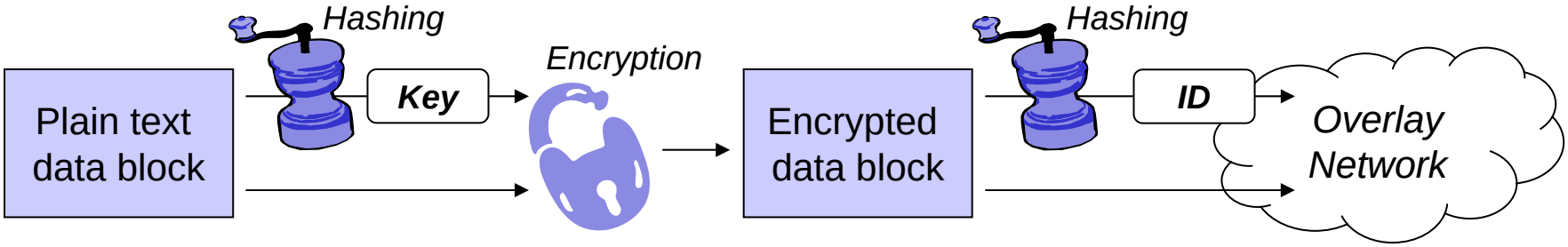
* anteilige Kapazität = Kapazität geteilt durch Grad des Knotens

FreeNet Design Goals

- Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet – A Distributed Anonymous Information Storage and Retrieval System, 2001
- Ziele:
  - Anonymes publizieren von Inhalten
  - Anonymes lesen von Inhalten
  - Anonymes bereithalten von Inhalten
  - Zenrurresistenz
  - Zuverlässigkeit durch Dezentralisierung
  - Skalierbares, effizientes, adaptives Routing
- Peer-to-Peer Netz
- Daten werden als statische Webseiten abgelegt
  - Zugriff über den Webbrowser über lokalen FreeNet Server
- Zusätzlich sind auch andere Anwendungen über die API möglich, es existieren z.B.
  - Message boards
  - File sharing Anwendungen
  - Online Chat

Content Hash Keys in Freenet

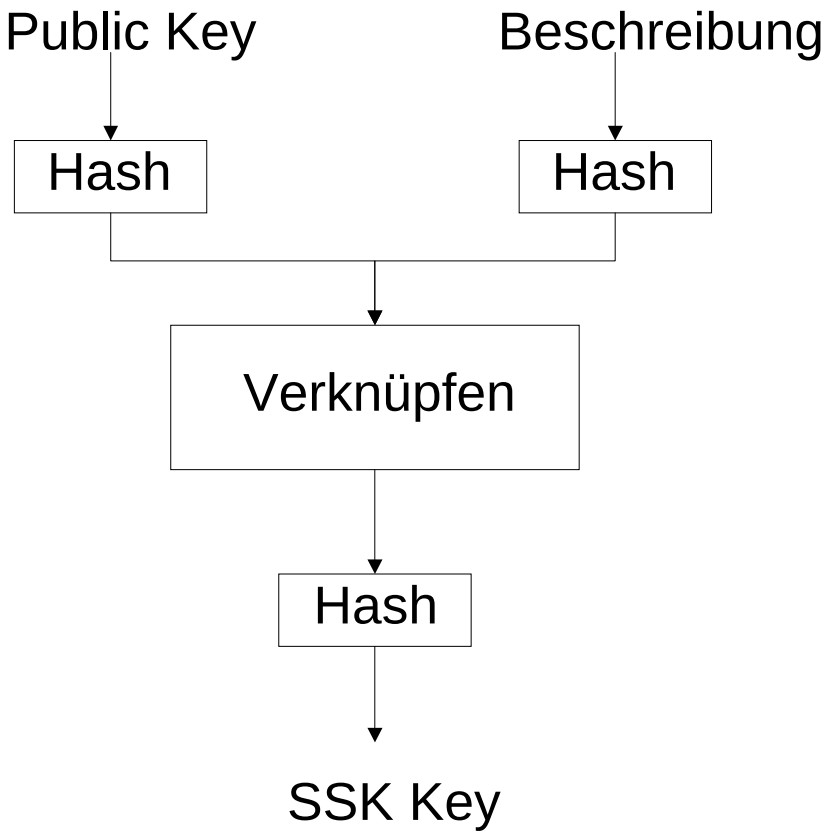
- Alle Daten in Freenet werden durch Keys identifiziert
- Statische Daten wie z.B. größere Textdateien benutzen das Content Hash Key (CHK) Verfahren
- Aufbau: CHK@[hash],[key],[crypto-settings]



Signed Subspace Keys in Freenet

- Für Seiten mit sich ändernden Inhalten ist das Verfahren nicht verwendbar
  - Man bräuchte eine Quelle, die einem immer die aktuellen Hashwerte mitteilt

- Zweites Verfahren um Daten in Freenet zu speichern
  - Signed Subspace Keys (SSK)
  - Benützt public key cryptographie
  - Id wird aus Public-Key und Bezeichner generiert
  - Enthält Versionsnummer



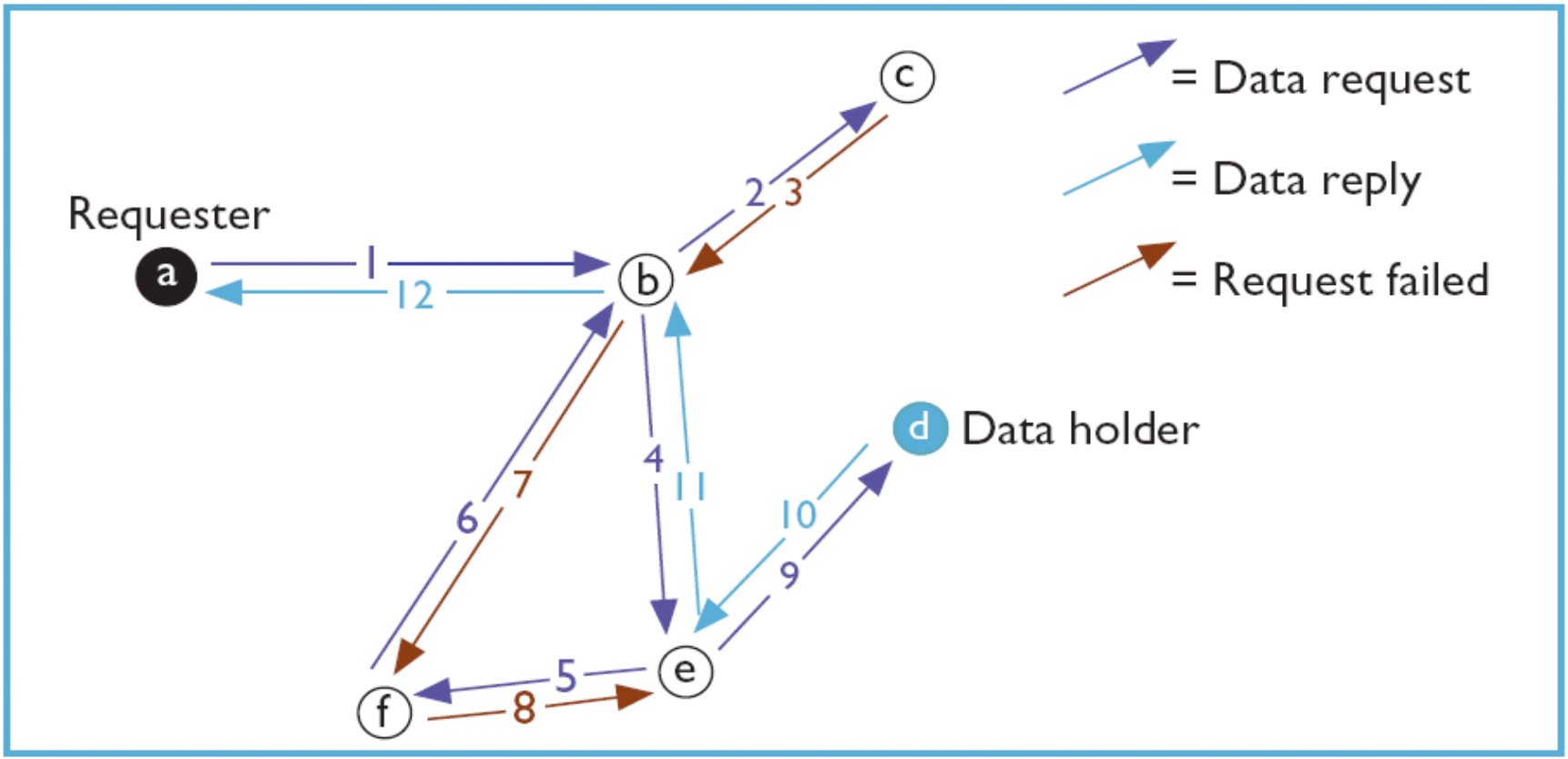
- Beispiel-URL:

  SSK@[public-key-hash],[decrytion-key],[settings]/[name]-[version]

Routing in Freenet

- Steepest Ascent Hill Climbing Search
- Jeder Knoten merkt sich, welche IDs andere Knoten gespeichert haben
  - Liste wird durch Mitlauschen des weitergeleiteten Verkehrs aufgebaut
- Anfragen werden an den Knoten weitergeleitet, der die nächste ID gespeichert hat
- Requests haben TTL; für die Positionsverschleierung kann aber ein Pfad vorgeladen werden
- Neue Daten nehmen den gleichen Weg, wenn sie in das Netz eingefügt werden
- Das Routing von Freenet wird mit der Zeit immer besser, da es immer mehr Informationen über die Nachbarn lernt. Einzelne Knoten spezialisieren sich nach einiger Zeit auf einen bereich von IDs, der über sie geroutet wird.
- Aber: Routing macht z.B. keinen Unterschied zwischen Maschinen mit schneller oder langsamer Anbindung.

Steepest Ascent Hill Climbing Search



Source: Clarke, Miller, Honk, Wiley: Protecting Free Expression Online with Freenet in IEEE Internet Computing 6:1 2001

Freenet, Darknet, and OpenNet (1)

- FreeNet Architektur hat ein Problem bei der Privatheit
  - Wenn man sich mit einem FreeNet Knoten verbindet bekommt man Adressen vieler anderer Knoten genannt
  - Wenn man das bei einigen Knoten wiederholt bekommt man schnell eine Liste vieler Rechner, die FreeNet betreiben
  - Regierungen könnten diese IPs an Providergrenzen blockieren
- Die Idee eines Darknets basiert auf dem Small-World-Network Gedanken
  - Man hat nur noch direkte Verbindungen zu seinen Freunden
  - Harvesting ist so unmöglich, da man keine IPs herausgibt
- In DarkNets funktioniert der alte FreeNet Routingalgorithmus nicht mehr:
  - Bei FreeNet wurden (manchmal) neue Verbindungen aufgebaut zu Knoten bei denen man Inhalte gefunden hat.
  - Stattdessen Key Based Routing: Jeder Knoten hat eine ID und man routet zu Knoten-IDs

Freenet, Darknet, and OpenNet (2)

- Neuere Freenet-Versionen (0.7) haben bis Ende 2007 nur den Darknet-Ansatz zugelassen
- Problem: Es kannten sich nicht genug FreeNet Benutzer, man hat Freunde deswegen über IRC oder Message-Boards gesucht.
  - Keine Verbesserte Sicherheit
  - Schlechte Netztopologie
- Seit Ende 2007 unterstützt Freenet zusätzlich zum Darknet wieder einen OpenNet Modus, der aber immer noch deaktiviert werden kann.

Unstrukturiert versus Strukturiert

- Suchen in Gnutella basieren auf dem Fluten des Netzes
  - Profitiert von der Small-World-Eigenschaft
  - Erzeugt aber viel Verkehr im Netz
- Fortgeschrittene Verfahren nutzen Random Walk bzw. High-Degree Seeking
  - Profitiert von der Potenzverteilung der Knotengrade
  - Langwierige Suche, da viele Knoten der Reihe nach abgesucht werden
- In beiden Fällen kann der Sucherfolg nicht garantiert werden!

- Freenet basiert auf einer Agregation von Hash Keys
  - Jeder Netzbereich spezialisiert sich auf bestimmte Key Ranges (=emergentes Verhalten)
  - Neue Verbindungen werden aufgrund von Key-Nähe geknüpft.
  - Somit funktioniert Steepest Ascent Hill Climbing Search mit der Zeit immer besser.

- Idee strukturierter Peer-to-Peer-Netze:
  - Erzeuge eine einfache (und somit bekannte) Topologie
  - Dies erlaubt eine zielgerichtete „Suche“ nach Inhalten
  - Außerdem werden Inhalte garantiert gefunden, falls sie überhaupt im Netz sind!

Distributed Hash Table (DHT)

- Strukturierte P2P-Netze sind als „Distributed Hash Table“ (DHT) bekannt
- Generische Schnittstelle zu Anwendung definierbar:

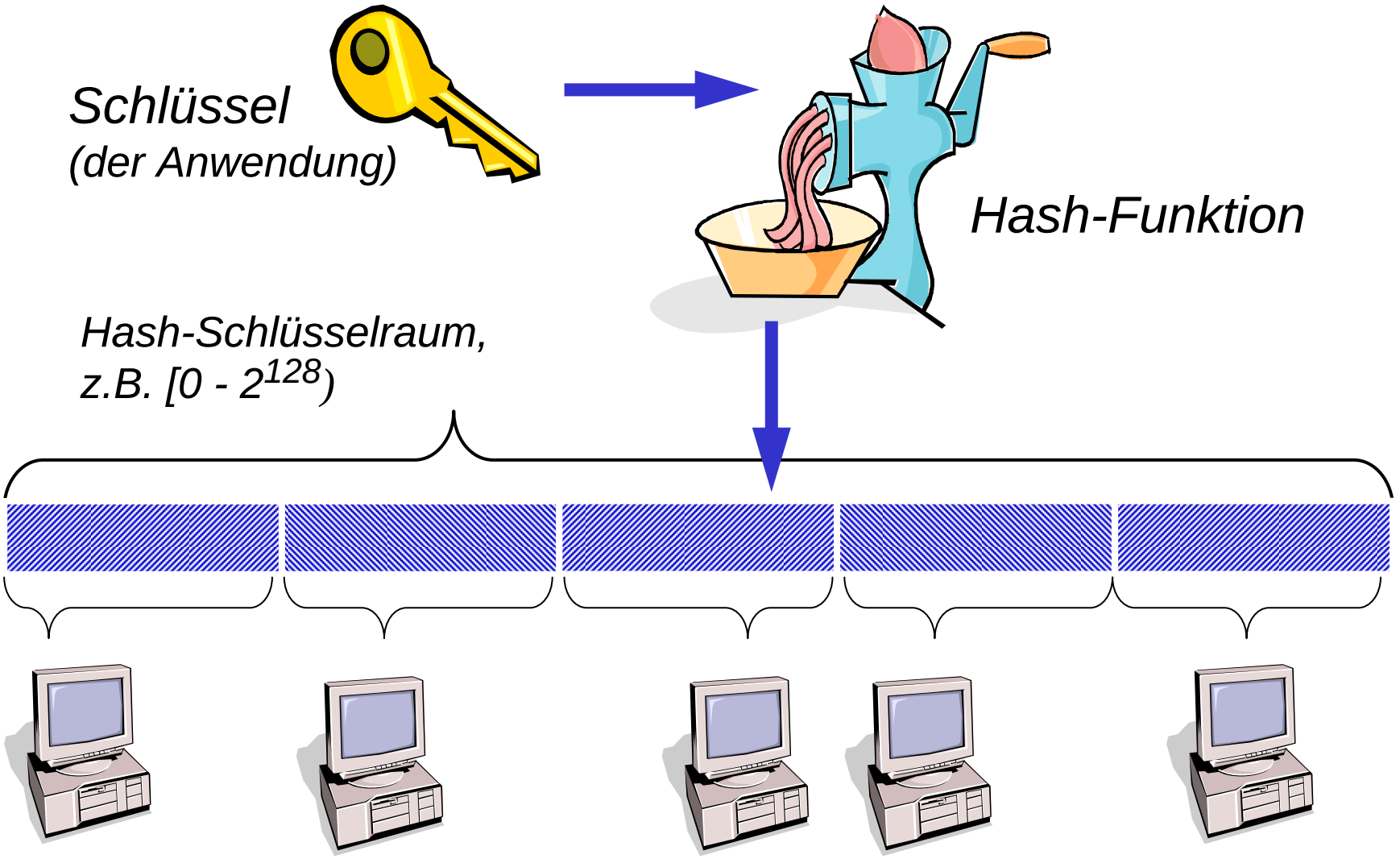
```
void publish(key_t, value_t);
value_t lookup(key_t);
```

- Erweiterte Funktionalität wäre wünschenswert, ist aber oft nicht vorhanden, da differenzierte Zugriffsrechte schwierig zu verwirklichen

```
void withdraw(key_t);
```

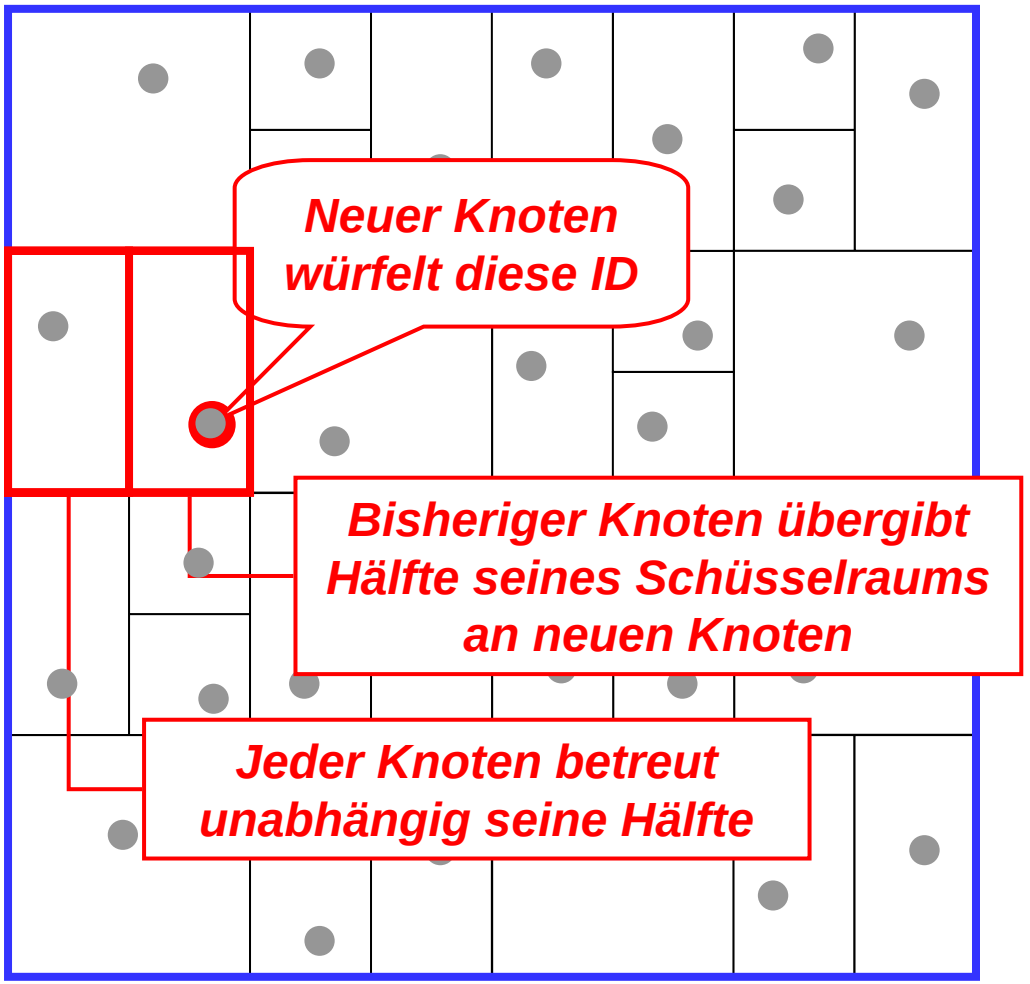
- Schlüssel
  - kann z.B. ein Dateiname oder Titel und Interpret eines Musikstücks sein
  - werden mittels Hash-Funktion auf Zahl fester Länge (z.B. 128 Bit) abgebildet, d.h. keine Eindeutigkeit garantiert, aber Kollisionen sehr unwahrscheinlich
- Knoten speichern Inhalte, deren Schlüssel in ein bestimmtes Intervall fallen
- Platzierung der Knoten meist gleichverteilt zufällig im Schlüsselraum

Verteilte Hash-Tabellen



Content Addressable Network (1)

- Schlüssel und Knoten-IDs werden auf d-dimensionalen Torus abgebildet
- Jeder Knoten übernimmt ein „Rechteck“ aus dem Schlüsselraum
- Beim Hinzufügen eines Knotens teilen sich dieser und der bisher zuständige Knoten diesen Teil
- Somit tragen alle Knoten eine ähnlich große Last



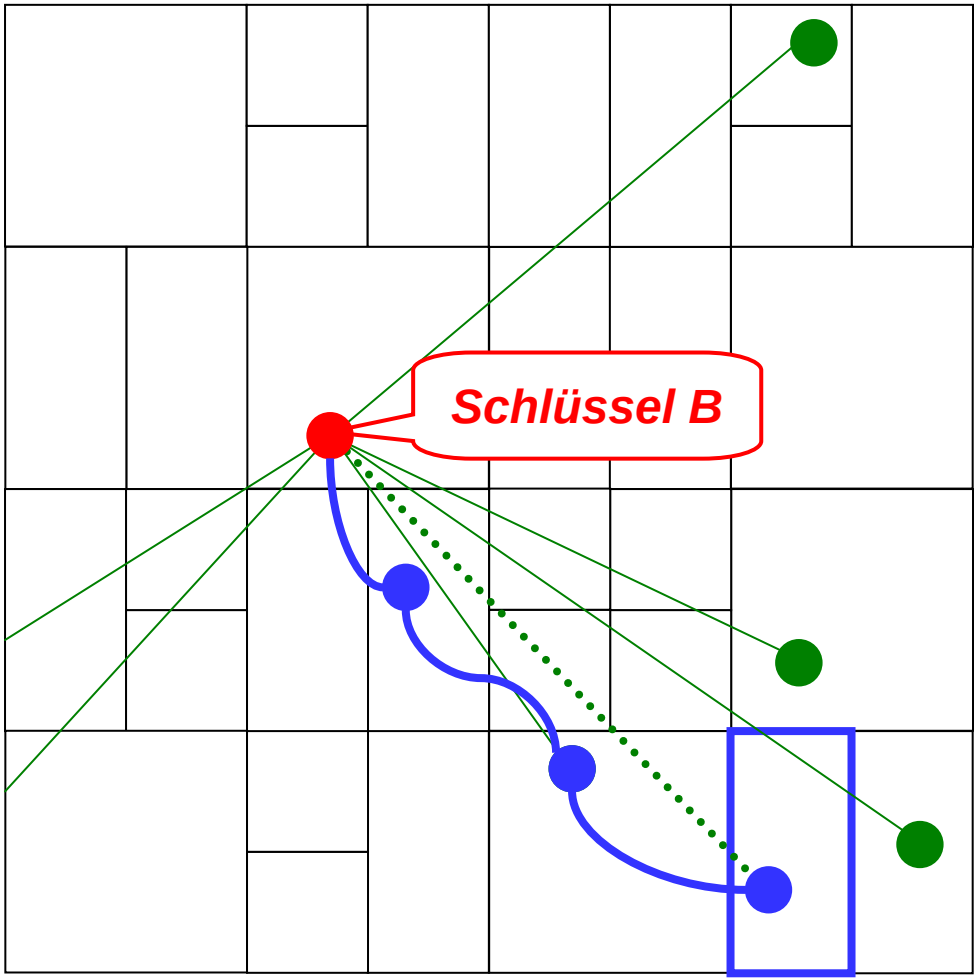
Gesamter Schlüsselraum

Content Addressable Network (2)

„Suchen“ im CAN:
Knoten A sucht Schlüssel B

Die Anfrage kann direkt in Richtung des entsprechenden Knotens weitergeleitet werden:

- Knoten A kennt seine direkten Nachbarn.
- Anfrage wird an den Nachbarn weitergeleitet, dessen euklidischer Abstand vom Ziel im Schlüsselraum am kleinsten ist.



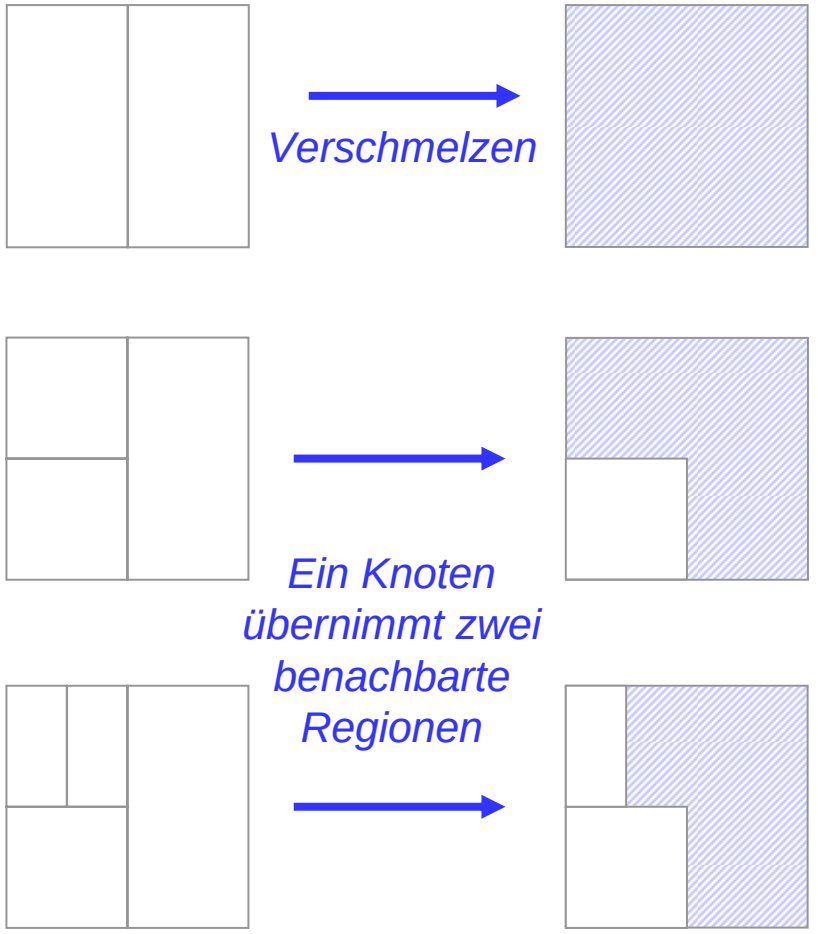
Content Addressable Network (3)

Beim Entfernen eines Knotens

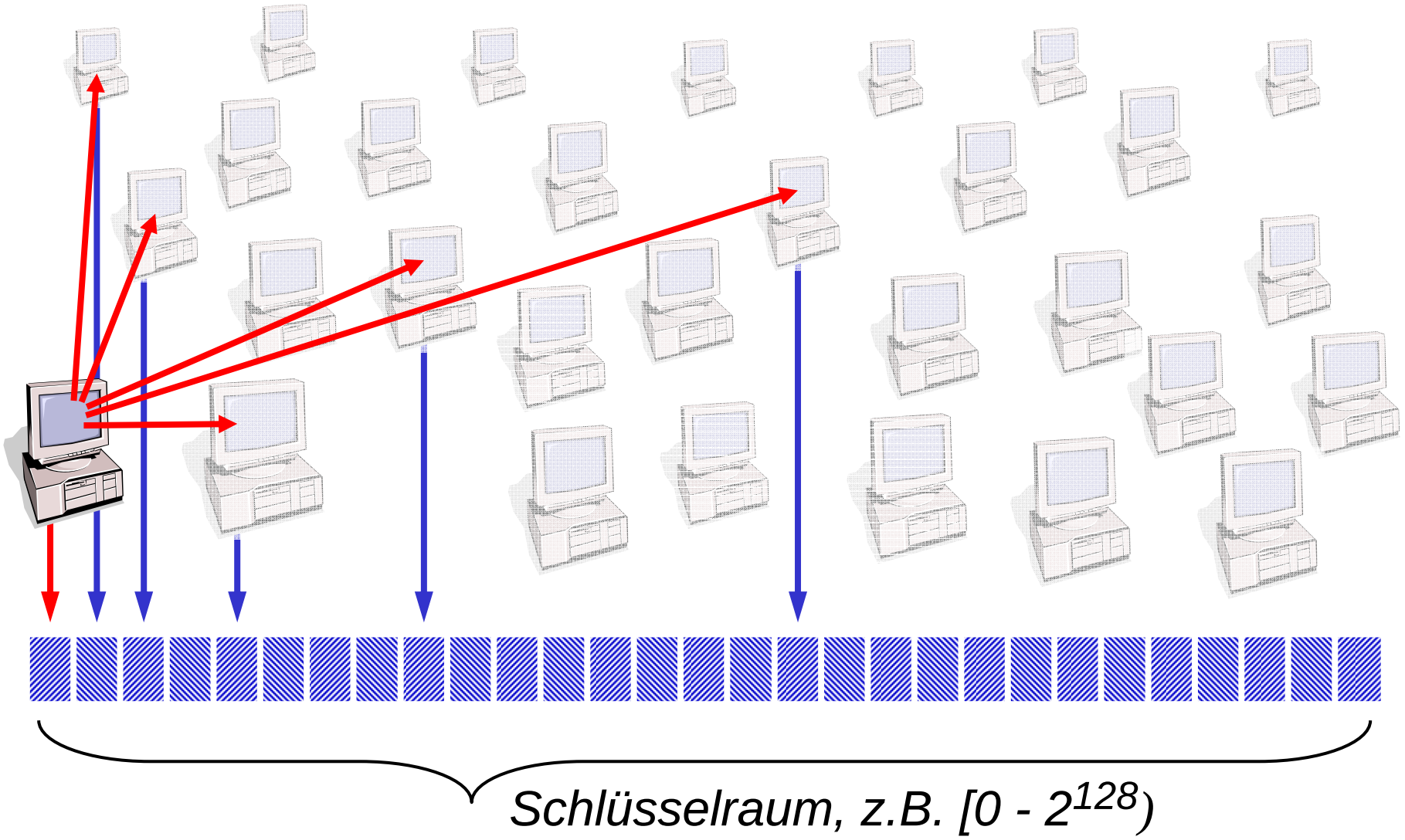
- verschmelzen zwei benachbarte Regionen zu einer Region doppelter Größe
- verwaltet ein Knoten zwei benacharte Regionen

Problem: Knoten müssen das CAN „graceful“ verlassen, d.h. zunächst ihren Zustand auf den Nachbar übertragen!

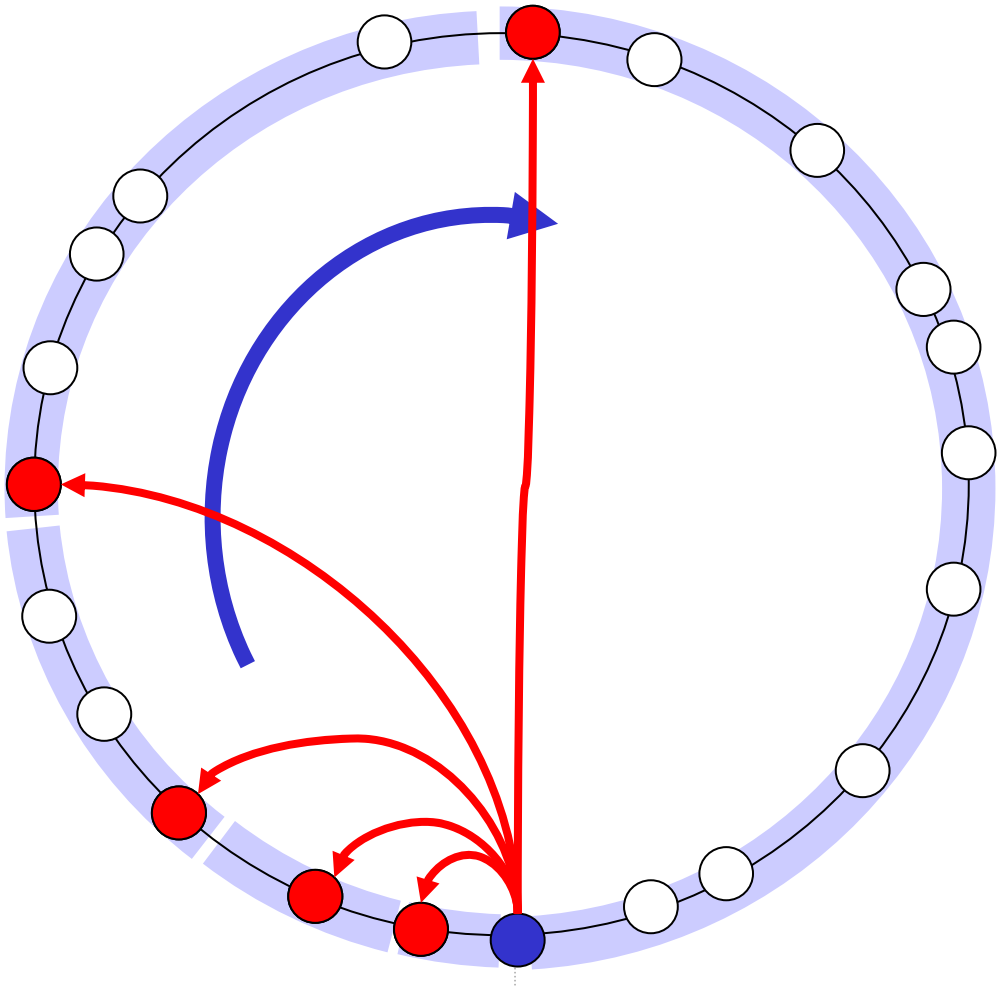
Dies ist nicht sichergestellt, also sollten Knoten die Inhalte redundant unter verschiedenen Schlüsseln speichern.



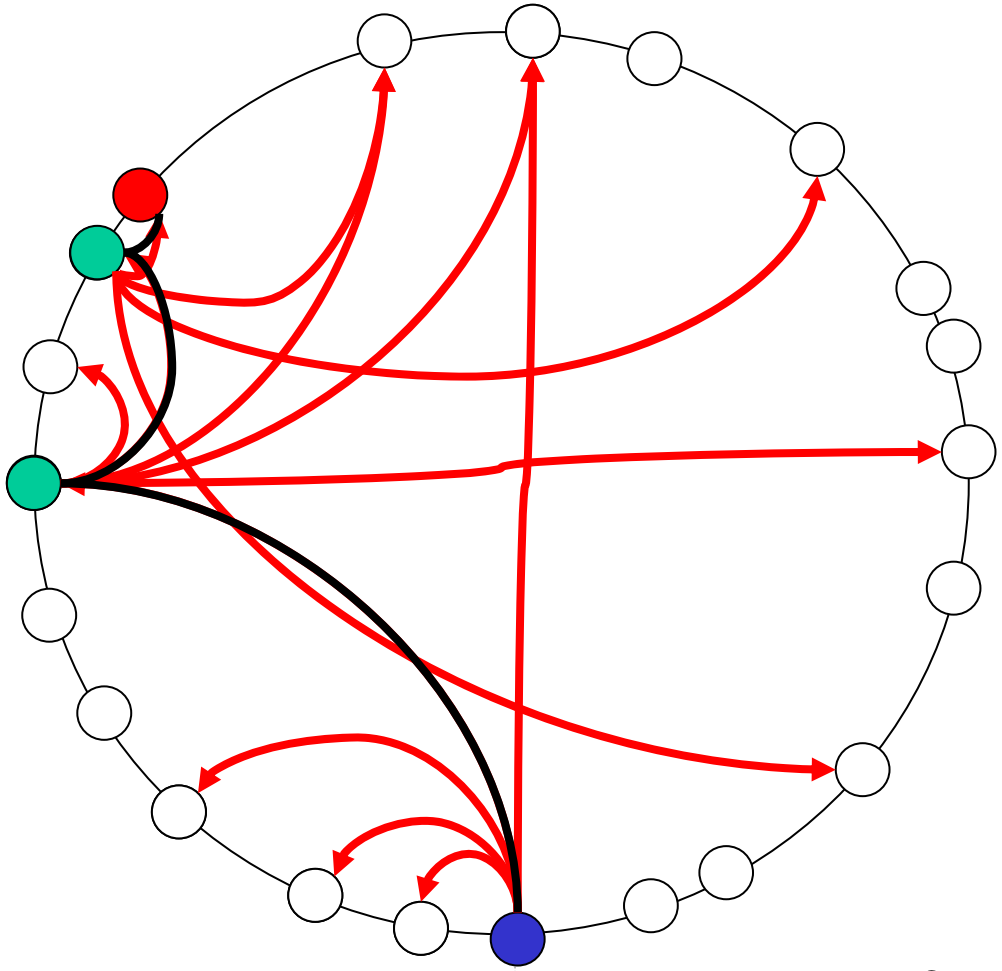
Chord – Eine verteilte Hash-Tabelle (1)



Chord – Eine verteilte Hash-Tabelle (2)



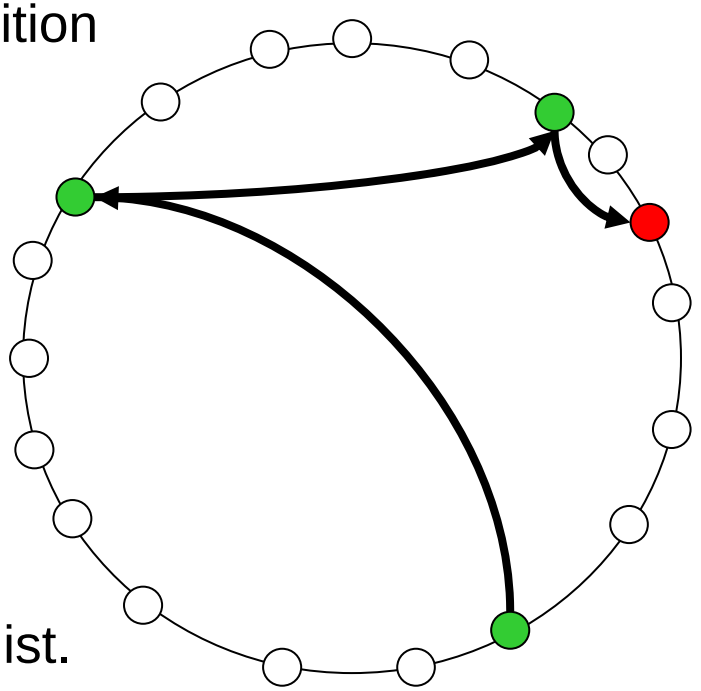
Chord – Eine verteilte Hash-Tabelle (2)



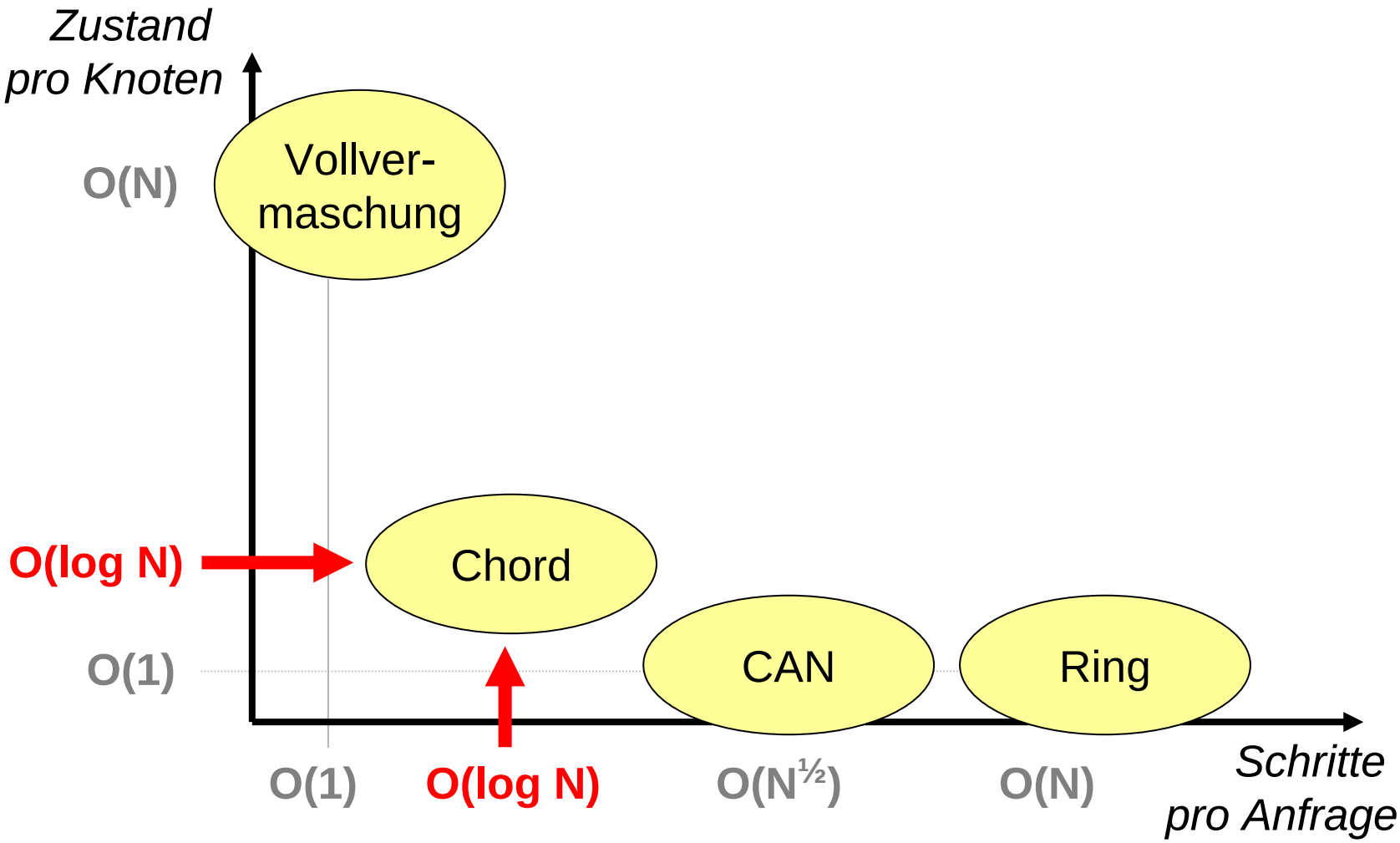
Stoica, et al. Chord: A scalable P2P lookup service, SIGCOMM 2001.

Chord im Überblick

- Jeder Peer wählt sich eine zufällige Position im Schlüsselraum (=virtuelle Adresse).
- Jeder Peer unterhält O(log N) Verbindungen ...
  - ... und zwar zu Peers in exponentiell wachsendem Abstand im Schlüsselraum.
- Eine Anfrage wird in Ringrichtung weitergereicht bis der Ziel-Peer erreicht ist.



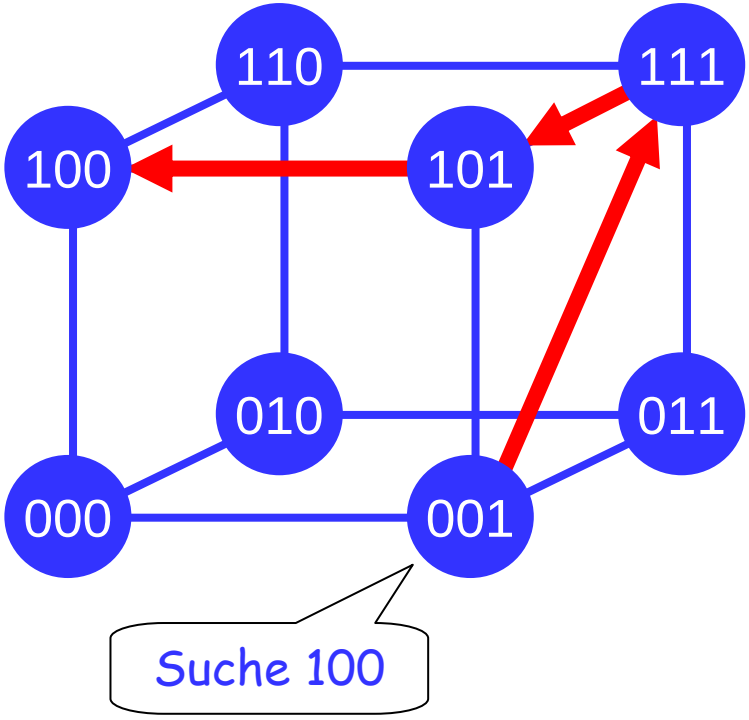
Trade-Off bei verteilten Hash-Tabellen



Pastry (1)

- Schlüssel und Knoten-IDs werden als Ziffernfolge im 2^ker-System aufgefasst und auf Hyperkubus abgebildet.
- Jede Stelle der Ziffernfolge entspricht der Koordinate einer Dimension des Hyperkubus.
- Jeder Knoten übernimmt einen (oder mehrere*) Zahlenbereiche mit einheitlichen Anfangsziffern.

** bis zu $2^k/2$ benachbarte Bereiche, z.B. 0xxx bis 4xxx im Zehnersystem.*



- Suchanfragen werden so weitergeleitet, dass schrittweise mehr und mehr Anfangsziffern zwischen gesuchtem Schlüssel und betreutem Schlüsselraum übereinstimmen.
- Dabei können auch Diagonalen des Hyperkubus verwendet werden.

Pastry (2)

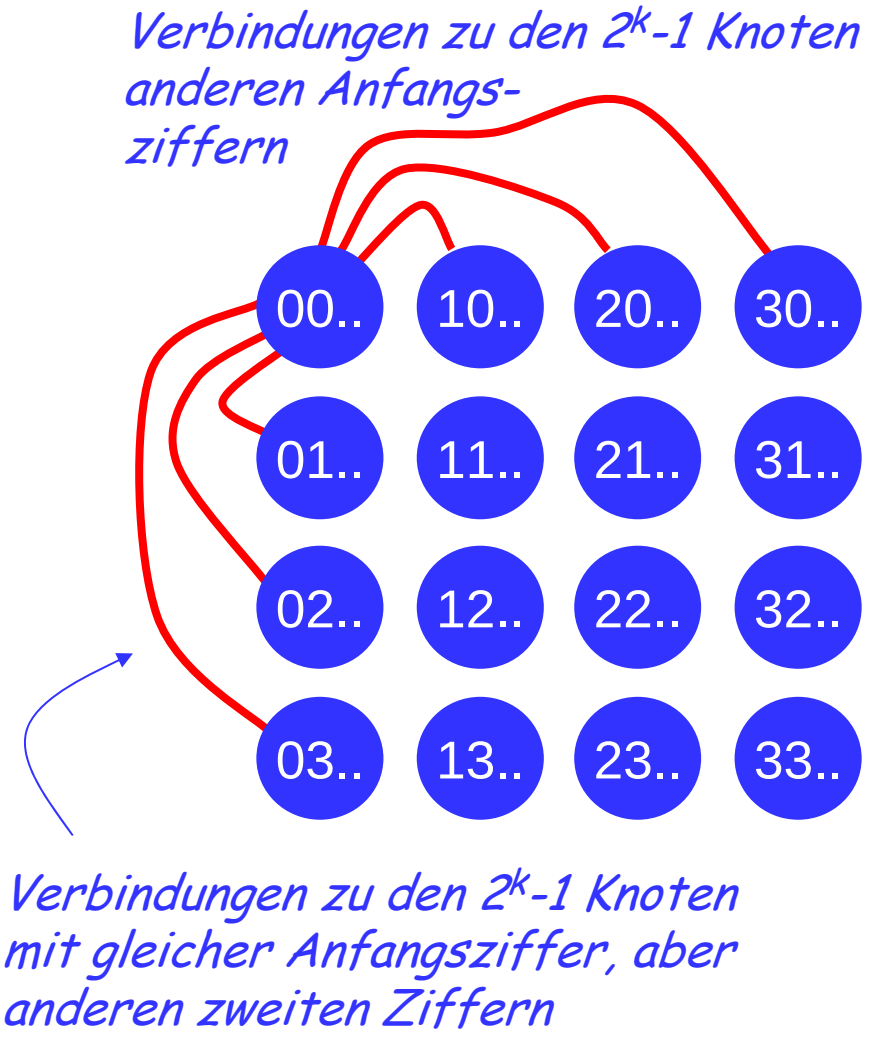
- Bei bis zu N Knoten unterscheiden sich die Knoten-IDs an höchstens $(\log_2 N)/k$ Stellen.
- Jeder Knoten muss somit

  $(2^k\text{-}1)\ (\log_2 N) / k$

  Overlay-Verbindungen unterhalten.
- Inhalte werden in höchstens

  $(\log_2 N) / k$

  Schritten gefunden.



Pastry (3)

- Das schrittweise „In-Übereinstimmung-Bringen“ der Anfangsziffern lässt eine gewisse Freiheit bei der Wahl der Peers für die Overlay-Verbindungen:
  - Im i-ten Schritt sind von ($\log_2 N$) Bits erst *ik* Bits festgelegt, d.h. es bleiben $N/2^{ik}$ mögliche Peers zur Auswahl.
- Diese Freiheit nutzt man, um eine redundante Routing-Tabelle aufzubauen:
  - Zu jedem Schritt speichert man *m* gleichwertige Peers.
  - Ist ein Peer nicht verfügbar, kann man auf einen alternativen Eintrag ausweichen.
- Diese Freiheit kann man auch nutzen, um eine Anpassung der Overlay-Topologie an die Topologie des darunterliegenden Netzes zu erreichen:
  - Zu jedem Schritt speichert man nur die besten der gleichwertigen Peers.
  - Dabei kann „beste Peers“ verschiedenes bedeuten, z.B. „kleinste RTT“, „größte Kapazität“, etc.

Chord und Pastry

- Betrachtet man Chord und Pastry von einem abstrakten, graphen-theoretischen Standpunkt aus, stellt man Ähnlichkeiten fest:
  - Chord springt in jedem Schritt zu einem Knoten, dessen Abstand (in Vorwärtsrichtung) vom Ziel eine um eins längere Anzahl führender Nullen (in der Binärdarstellung) hat.
  - Pastry springt so, dass eine immer größere Übereinstimmung zwischen Knoten-ID und Schlüssel erreicht wird.
- Dabei entscheidet sich das ursprüngliche Chord dafür, für jeden Knoten die Nachbarn eindeutig vorzuschreiben, Pastry lässt eine gewisse Freiheit bei der Wahl der Nachbarn.
  - Entsprechend hat die Chord-Topologie einen kleinen Cluster-Koeffizienten. (Im symmetrischen Fall kann man zeigen, dass er $1/\log_2 N$ beträgt.)
  - In Pastry hängt der Cluster-Koeffizient von der Wahl der Nachbarn ab.
- Vergleicht man Knotengrad und Durchmesser der Chord- und Pastry-Graphen, sieht man, dass Pastry den Durchmesser verringert, und zwar auf Kosten eines höheren Knotengrades.

CAN und Chord

- Aus dem Vergleich von Chord und Pastry erkennt man, dass für k=2 gehen beide DHT-Varianten bezüglich Knotengrad und Durchmesser in einander übergehen.

	Grad	Durchmesser
CAN	2d	$\frac{1}{2}d\ N^{1/d}$
Chord	$\log_2 N$	$\log_2 N$
Pastry	$(2^k-1)\ (\log_2 N) / k$	$(\log_2 N) / k$

- Für $d = \frac{1}{2} \log_2 N$ geht auch CAN bezüglich Grad und Durchmesser in das Ergebnis von Chord über.
- Betrachtet man die Zahl der Knoten in einem solchen „entarteten“ CAN, sieht man, dass pro Dimension nur $N^{1/d} = 4$ Knoten vorhanden sind. Bei zwei Nachbarn pro Dimension erreicht CAN hier also fast direkt zum korrekten Knoten (in dieser Dimension) springen.

Siehe auch: Loguinov et al. „Graph-Theoretic Analysis of Structured Peer-to-Peer Systems: Routing Distances and Fault Resilience“, SIGCOMM 2003

Vergleich der Verfahren

Unstrukturiert

- Suche mittels Fluten oder Random Walk
- Kein garantierter Erfolg
- Unscharfe Suchen möglich

CAN

- Suche mittels „geographischem Routen“ im Schlüsselraum
- Nachbarn eindeutig festgelegt, aber hohe Redundanz der Pfade
- Suchaufwand $O(N^{1/d})$
- Speicheraufwand O(1)

Chord

- Binäre Suche mit immer kleineren Sprüngen auf das Ziel zu
- Nachbarn eindeutig festgelegt, leichte Redundanz der Pfade
- Suchaufwand O(log N)
- Speicheraufwand O(log N)

Pastry

- Suche mittels sukzessivem Präfixmatching im Schlüsselraum
- Freiheit bei der Wahl der Nachbarn
- Suchaufwand O(log N)
- Speicheraufwand O(log N)

Zusammenfassung

- Unstrukturierte Peer-to-Peer Systeme
  - Geeignet um vorhandene Ressourcen zu nutzen
  - Teilnehmende Systeme bilden ein Overlay
  - Suchanfragen per Flooding oder Random-Walk verteilt
- Gnutella bildet ein Overlay und flutet darin die Suchanfragen
  - Einfacher Overlay-Bildungsmechanismus führt zu Small-World-Netz
  - Aber: Ursprüngliches System kollabierte wegen Überlastung
  - Neues System nutzt „Super Peers“ und berücksichtigt so Heterogenität der Peers
- Gianduia führt das Gnutalla Prinzip weiter
  - High Degree Seeking statt Fluten
  - Spezielle Lastregelungsmechanismen
- Freenet sucht Inhalte aufgrund von Hash-Keys
- Das Routing lernt welche Nachbarn welche Anfrage beantworten konnte.
- Manchmal neue Verbindungen zu Peers aufbauen, die die eigene Anfrage beantworten konnten.
- Strukturierte P2P Systeme bauen spezifische Overlay-Topologien auf
  - CAN: d-dimensionaler Torus
  - Chord: Ring
  - Pastry: Hypercubus
- Vorteile:
  - Effiziente Suche über Hash-Schlüssel
  - System kann negative Antworten geben (falls das System konsistent ist)
- Nachteile:
  - Aufbau und Pflege des Overlays ist aufwändig
  - Teilnehmende Rechner müssen fremde (Index-) Daten speichern
  - Gesamtsystem muss ggf. Daten replizieren, um gegen Verlust einzelner Rechner geschützt zu sein.

Questions?

Thomas Fuhrmann

Department of Informatics
Self-Organizing Systems Group
c/o I8 Network Architectures and Services
Technical University Munich, Germany

fuhrmann@net.in.tum.de