Strukturierte Peer-to-Peer Netze

Distributed Operating Systems

Konrad Miller <konrad.miller@ibds.uka.de>

06.07.2009

Ein Großteil der Folien ist von der Vorlesung:
„Protokollanalyse Selbstorganisierender P2P-Systeme“
von Thomas Fuhrmann übernommen.

Outline

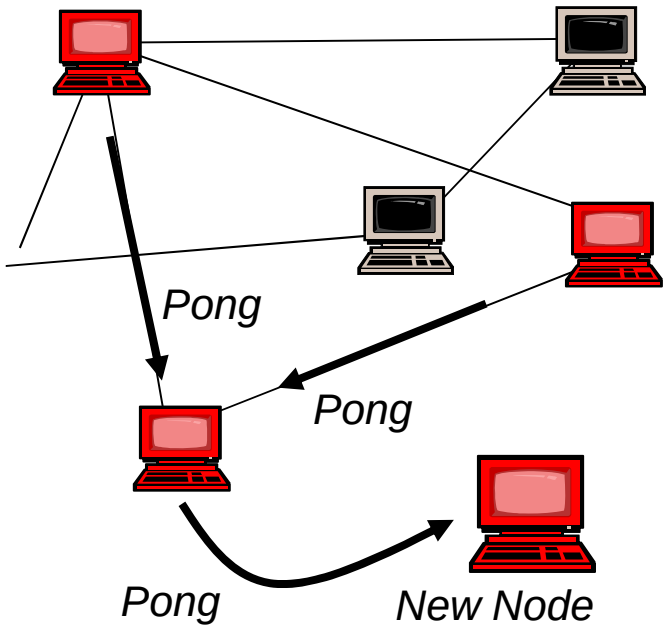
- Was sind P2P Systeme?
- Unstrukturierte P2P Systeme und ihre Eigenschaften
- Verteilte Hash-Tabellen
- Beispiele für Strukturierte P2P Systeme
  - Content Addressable Network
  - Chord
  - Pastry
- Optimierungen in Strukturierten P2P Systemen
  - Proximity Neighbor Selection
  - Proximity Route Selection

Was sind P2P Systeme?

- Client-Server Systeme sind geplant und werden administriert
  - Netzwerkverkehr konzentriert sich am und vor den Servern
  - Skalierbarkeit und Robustheit problematisch
- Peer-to-Peer Systeme bestehen aus **autonomen**, **gleichberechtigten** Systemen die sich **selbst organisieren**
- Diese Definition wird jedoch teilweise verletzt:
  - Skype: benutzt Super-Peers (widerspricht Gleichberechtigung)
  - Bootstrapping sonst schwer realisierbar
- Beispiele für bekannte Peer-to-Peer Applikationen sind:
  - Filesharing: eDonkey, Gnutella
  - VoIP: Skype
  - Email: SMTP
- P2P Anwendungen kommunizieren oft über einen anderen Pfad, als durch die Netzwerkschicht vorgegeben wird (über ein Overlay-Netz)

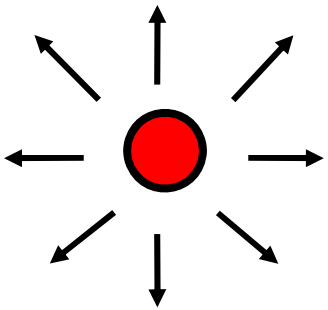
Unstrukturierte P2P Netze

Gnutella Network



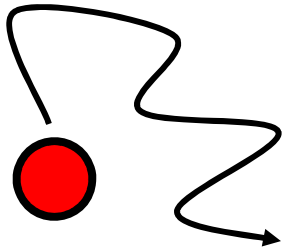
- Zufällige Verbindungen zu anderen Peers
- Die daraus resultierende Topologie ist oft Small-World-Netz:
  - Kleiner Durchmesser
  - Hoher Cluster-Koeffizient
  - Skalen-invariant
- Suche z.B. via:
  - Fluten
  - Random-Walk
  - Super-Peers
- Sucherfolg kann nicht garantiert werden
- Suche erzeugt große Last im Netz

Suche in Unstrukturierten P2P Netzen



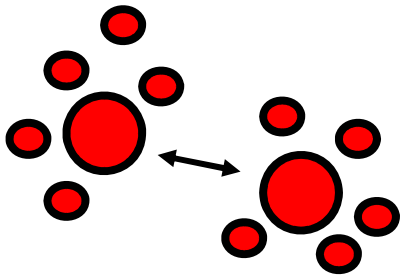
Flooding

- (parallele) Breitensuche
- Schleifen müssen vermieden werden (Message IDs/Path-Record)
- Sehr hohes Nachrichtenaufkommen
- Robust gegen Topologieänderungen und Nachrichtenverlust



Random-Walk

- (sequentielle) Tiefensuche
- Wenig Nachichtenoverhead
- Große Latenz
- Ca. 80% Erfolgswahrscheinlichkeit bei der Suche
- Es gibt verschiedene Varianten (z.B. High-Degree-Seeking)



Super-Peers

- Jeder Peer ist mit genau einem Super-Peer verbunden
- Die Super-Peers sind enger vermascht
- Informationen werden auf die Super-Peers gepusht
- Gesucht wird ausschließlich über die Super-Peers

Strukturierte P2P Netze

Idee Strukturierter Peer-to-Peer-Netze:

- Erzeuge eine bekannte Topologie
- Dies erlaubt eine zielgerichtete „Suche“ nach Inhalten
- Außerdem werden Inhalte garantiert gefunden, falls sie überhaupt im Netz sind!

Verteilte Hash-Tabellen

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

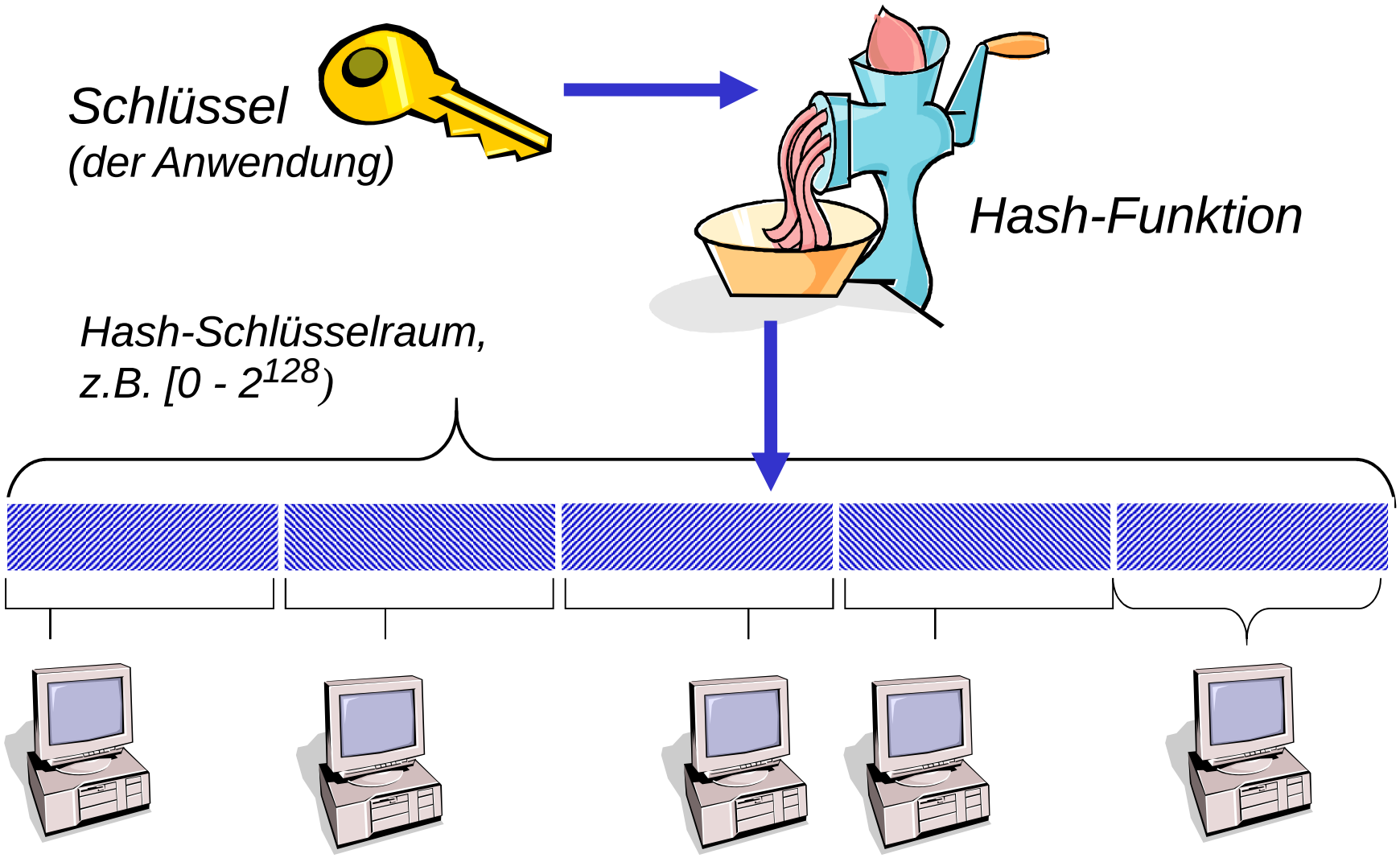
```
void publish(key_t, value_t);

value_t lookup(key_t);
```

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

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

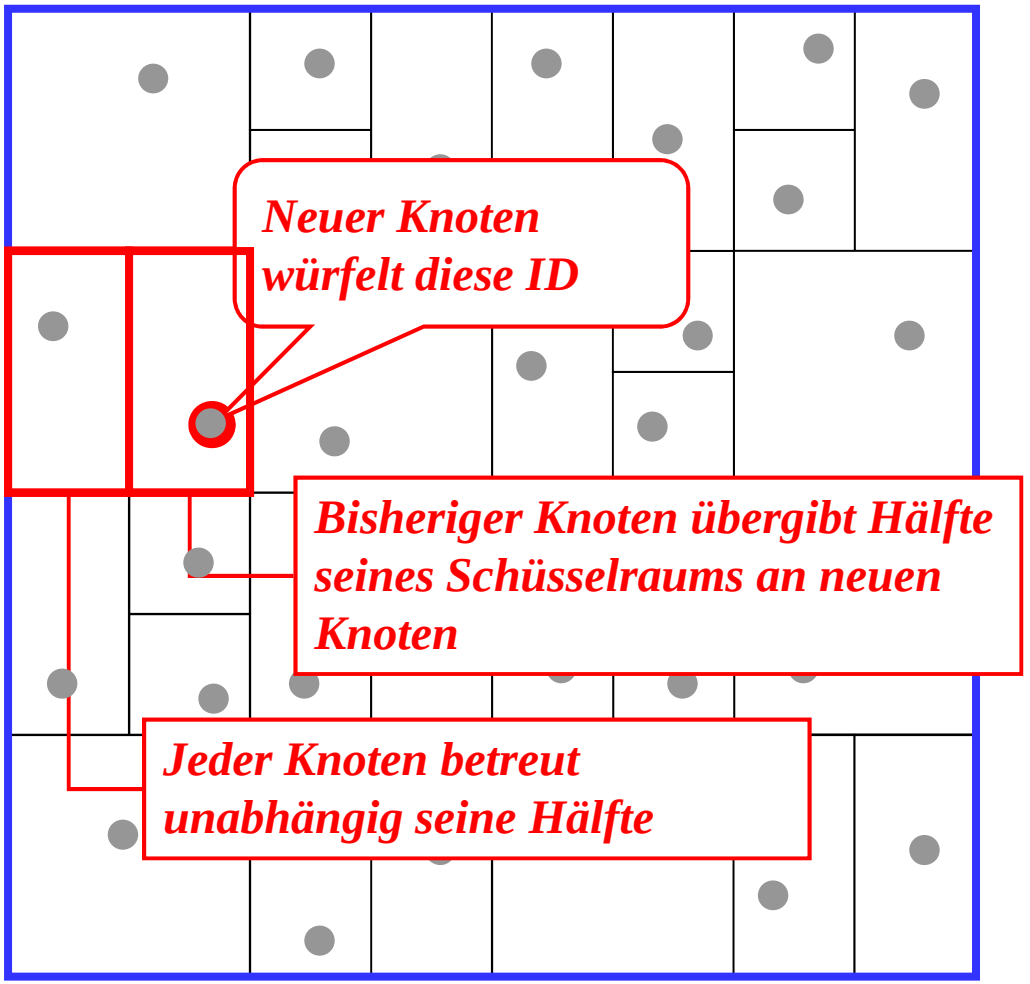
- Schlüssel
  - kann z.B. ein Dateiname oder Titel und Interpret eines Musikstücks sein
  - werden mittels Hash-Funktion auf Zahl fester Länge (z.B. 128 Bit) abgebildet, d.h. keine Eindeutigkeit garantiert, aber Kollisionen sehr unwahrscheinlich
- Knoten speichern Inhalte, deren Schlüssel in ein bestimmtes 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 bei gleichverteilter Nutzung des Schlüsselraums

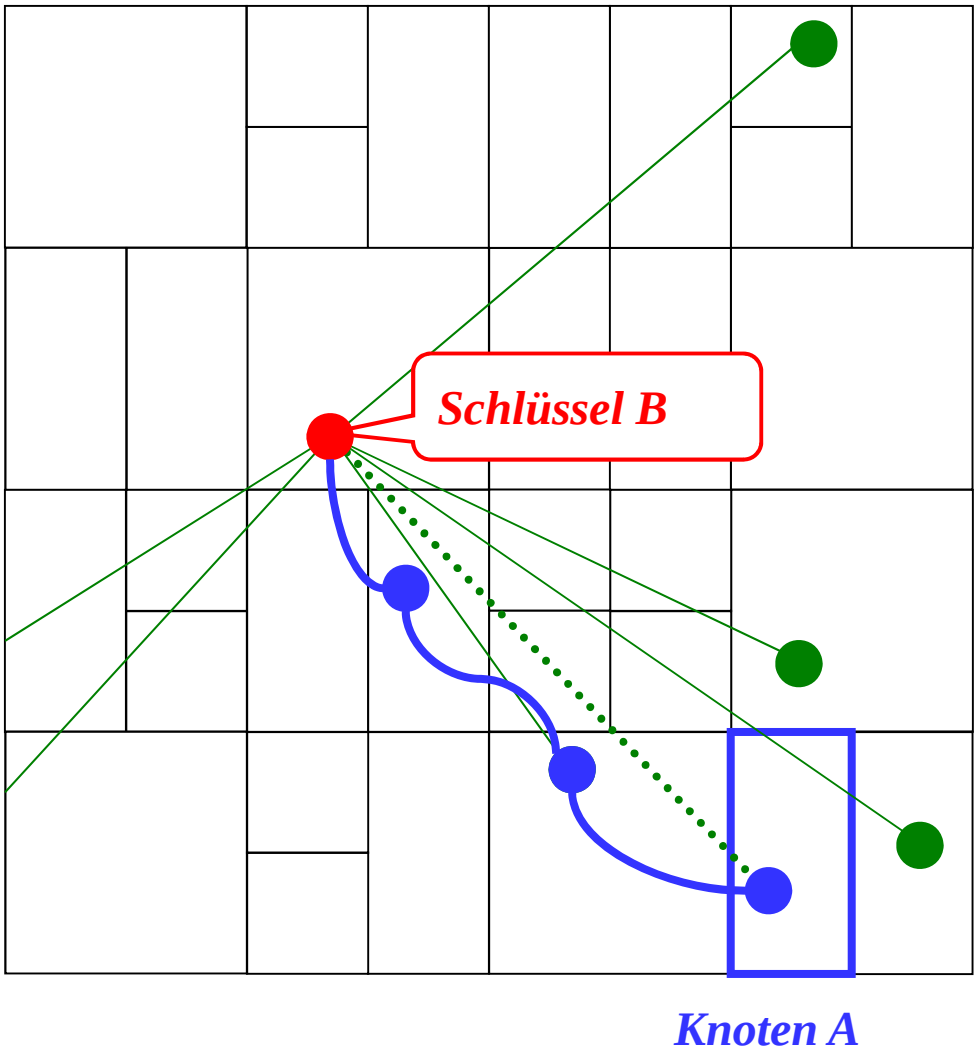


Content Addressable Network (2)

Knoten A sucht Schlüssel B

Die Anfrage kann jeweils in Richtung des Schlüssels weitergeleitet werden:

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



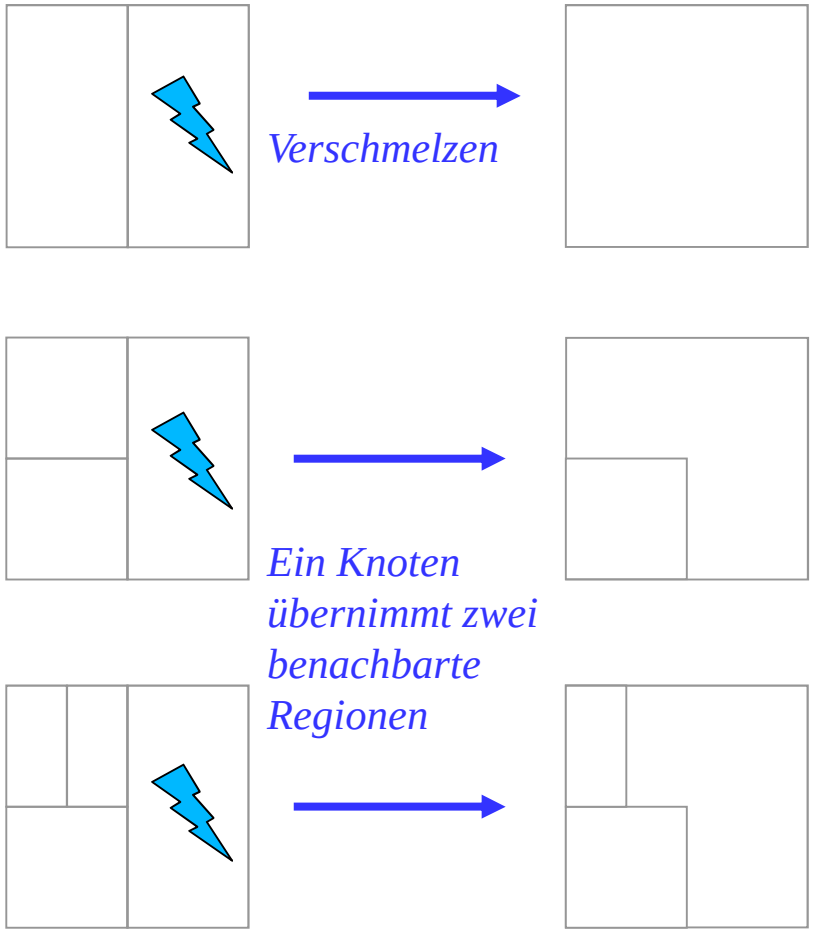
Content Addressable Network (3)

Beim Entfernen eines Knotens

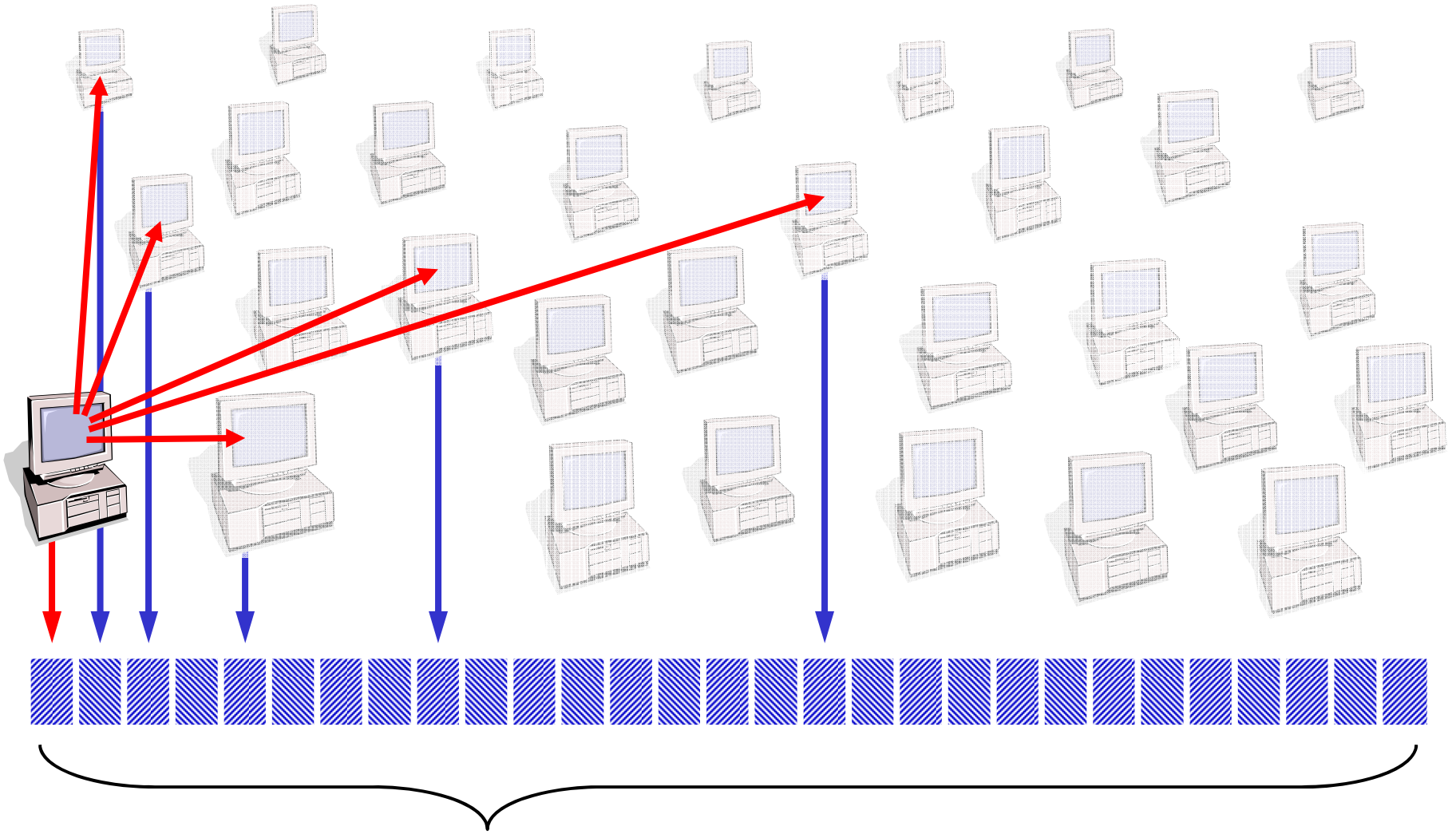
- verschmelzen zwei benachbarte Regionen zu einer Region doppelter Größe
- verwaltet ein Knoten zwei benachbarte Regionen

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

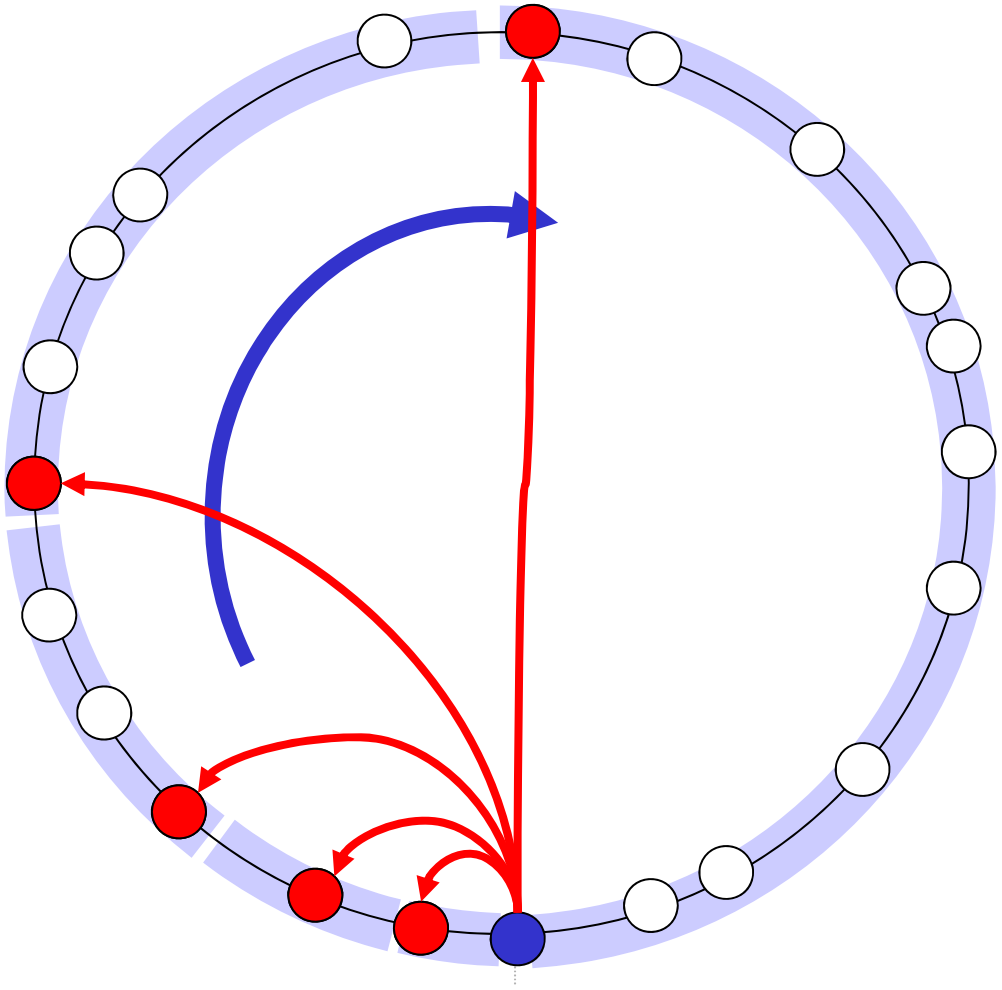
Dies ist nicht sichergestellt, also sollten Knoten die Inhalte redundant unter verschiedenen Schlüsseln speichern.



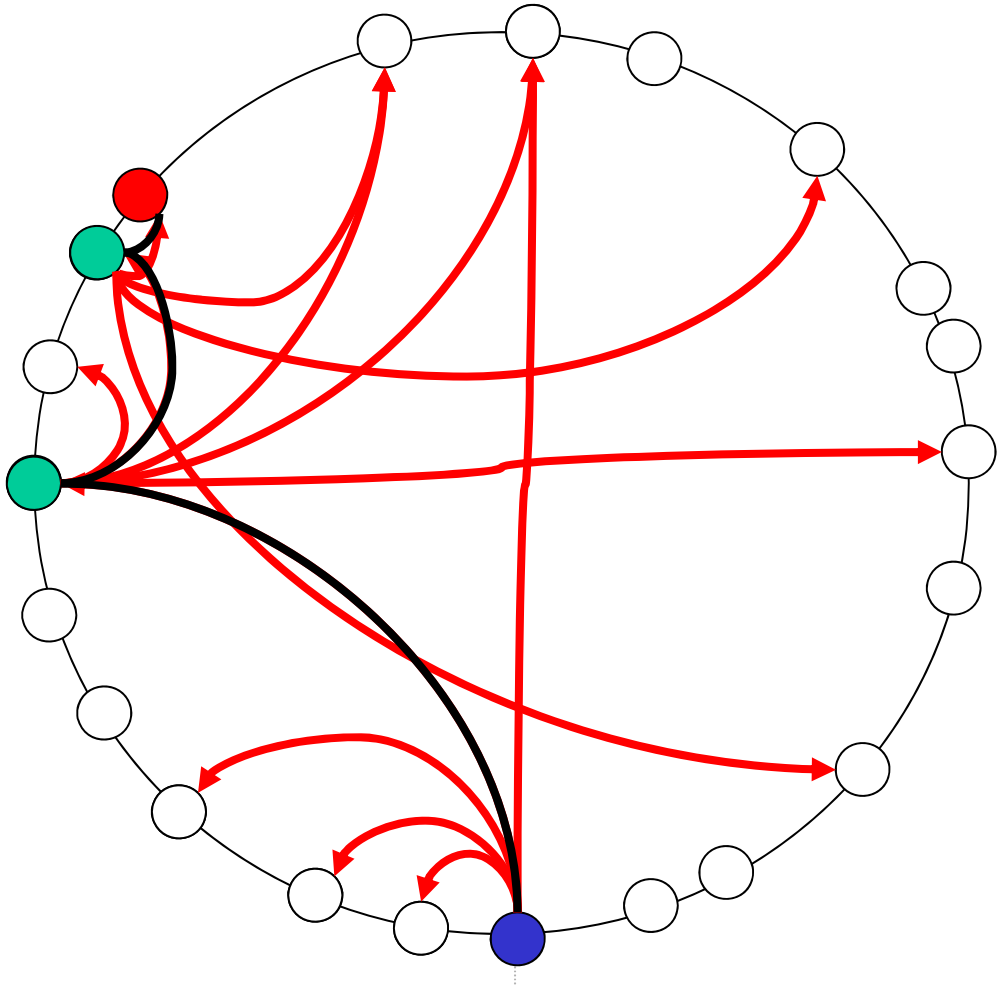
Chord – Eine verteilte Hash-Tabelle (1)



Chord – Eine verteilte Hash-Tabelle (2)

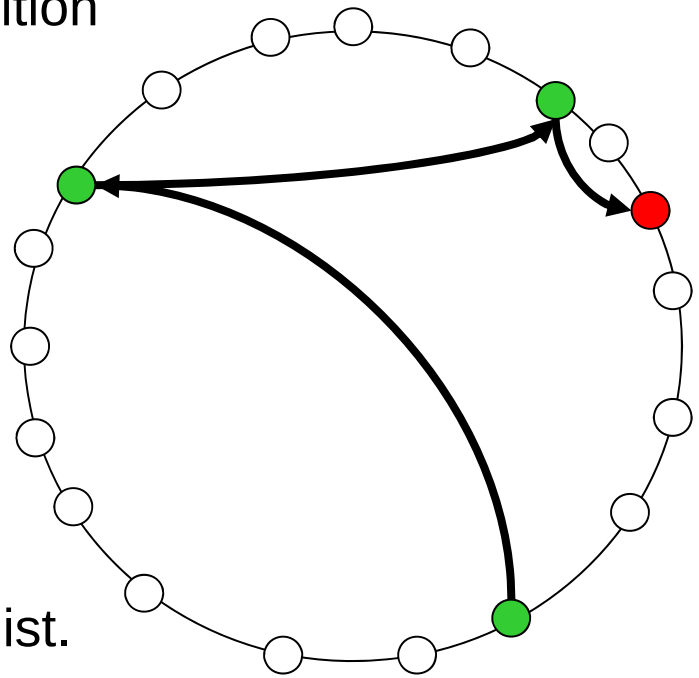


Chord – Eine verteilte Hash-Tabelle (2)

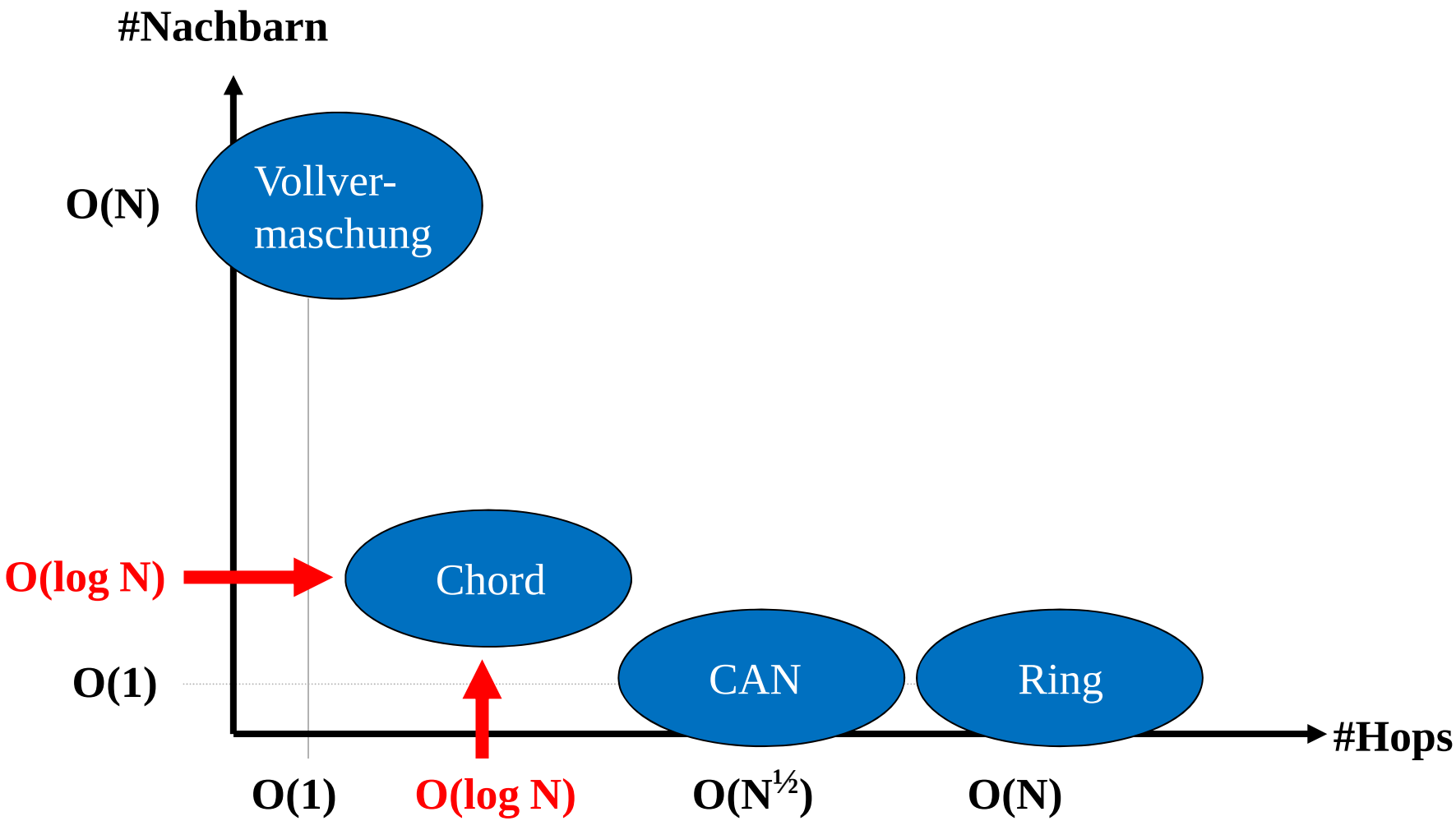


Chord im Überblick

- 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.
- Die Schlüssel eines Segments werden im „Folgeknoten“ verwaltet.



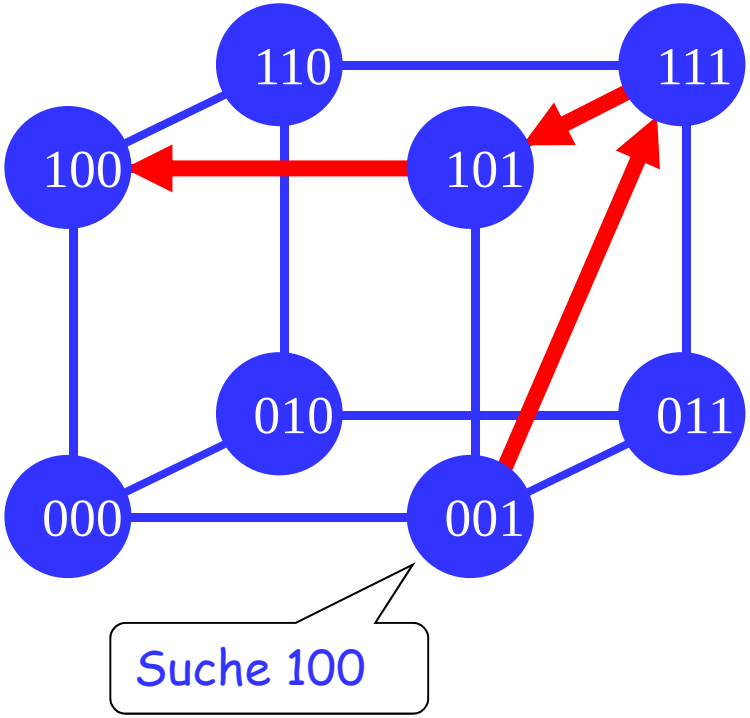
Trade-Off bei verteilten Hash-Tabellen



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)

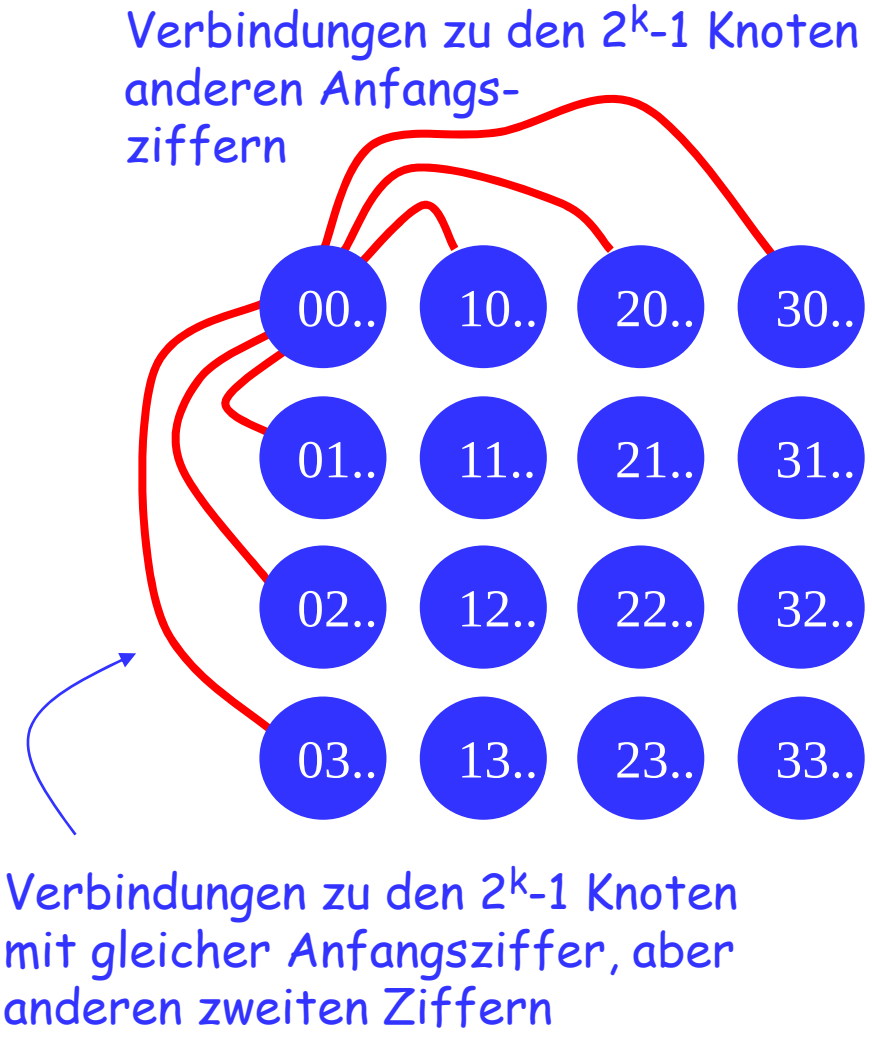
- Bei bis zu N Knoten unterscheiden sich die Knoten-IDs an höchstens $(\log_2 N)/k$ Stellen.
- Jeder Knoten muss somit

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

  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 kann man nutzen, 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, Chord und Pastry

- 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.

Vergleich der Verfahren

Unstrukturiert

- Suche mittels Fluten oder Random Walk
- Kein garantierter Erfolg
- Unscharfe Suchen möglich (z.B. mit Wildcards)

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)

Optimierungen in DHTs

- Strukturierte P2P-Systeme bauen spezifische Overlay-Topologien auf
  - CAN: d-dimensionaler Torus
  - Chord: Ring
  - Pastry: Hypercubus
- Dadurch ist ein gezieltes Routing zu Hash-Schlüsseln möglich (Key Based Routing, „Suche“)
- Nachteile die daraus entstehen (gegenüber Unstrukturierten Netzen):
  - Es ist notwendig diese Struktur aufzubauen und zu pflegen (v.a. bei starkem churn schwierig und teuer).
  - Peers müssen fremde Daten indizieren.
  - Knoten dürfen das System nicht abrupt verlassen.
  - Die durch das System vorgegebene Struktur entspricht nicht der physischen Struktur des Netzwerks (ein kurzer Sprung im Overlay kann einen langen Sprung im Underlay bedeuten).
- Im Folgenden werden Optimierungen vorgestellt die einige diese Nachteile lindern

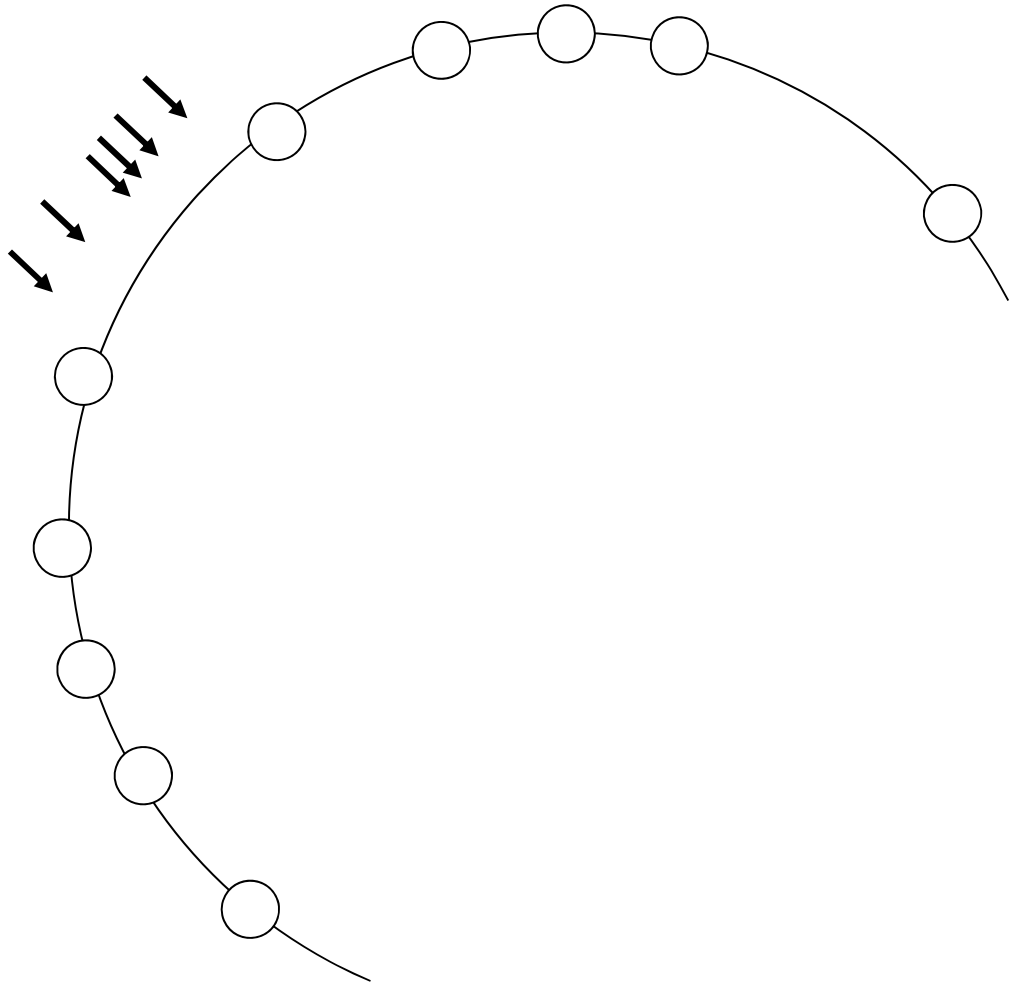
Nähengewahrheit

- Pastry erlaubt explizit die Freiheit an jeder Stelle einen beliebigen Peer auszuwählen, der dem grundsätzlichen Kriterium („Ziffer kann geändert werden“) genügt.
- Chord funktioniert auch wenn man die Peers nicht genau in exponentiell steigendem virtuellen Abstand wählt.
- Die Wahl physisch guter Peers erhöht die Leistungsfähigkeit des Systems:
  - Wahl guter Peers zum Aufbau von Overlay-Verbindungen (Proximity Neighbor Selection, PNS)
  - Wahl guter Peers während des Routings (Proximity Route Selection, PRS)

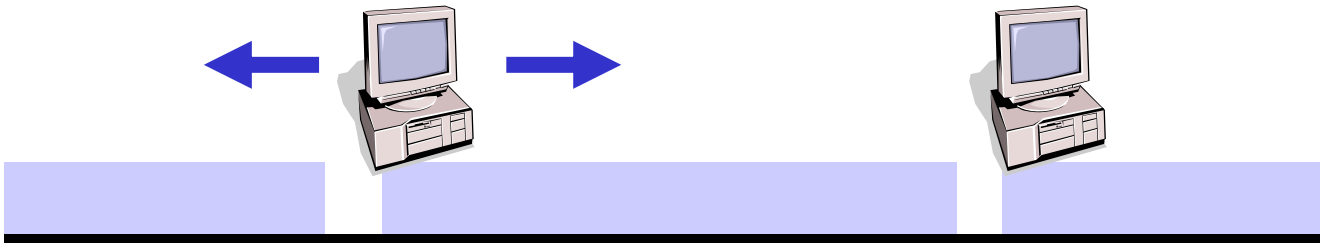
Replikation

- Mehrere Hash-Funktionen
  - z.B. CubeHash und SHA-256
  - Indiziert jeden Datenblock mit hoher Wahrscheinlichkeit auf verschiedenen Peers
  - Routing zum näheren Hash
  - Falls Knoten ausgefallen, routen zum Zweithash
- Mehrere Peers pro Segment zuständig
  - Bei CAN z.B. erst splitten wenn min. n Peers für das Segment zuständig
  - Bei Chord z.B. nicht nur für Segment bis zum Vorgänger sondern die letzten zwei Segmente speichern

Hash Key Distribution



Load Balancing



Zusammenfassung

- Unstrukturierte P2P Systeme bauen zufällige Verbindungen auf
  - Nutzen Eigenschaften von Small-World-Netzen zum Suchen
  - Sucherfolg kann i.A. nicht garantiert werden
- Strukturierte P2P Systeme geben eine Overlay-Topologie vor und ermöglichen so ein Key-Based-Routing (KBR)
  - Key wird gefunden falls er im Netz ist
  - PNS nutzt die Freiheit bei der Wahl der Overlay-Verbindungen zwischen verschiedenen Peers wählen zu können.
  - PRS nutzt die Freiheit während der Weiterleitung von Nachrichten zwischen verschiedenen Overlay-Verbindungen wählen zu können.
  - Replikation hilft den Hash trotz starker Knotenfluktuation im Netz zu halten