# Verteilte Hash-Tabellen

- Übersicht
  - Was sind Verteilte Hash-Tabellen?
  - Einsatzgebiete
  - Komplexität und Eigenschaften
- Betrachtete Systeme
  - Chord - UC Berkely
  - CAN - UC Berkeley, MIT
  - Pastry - Rice, Microsoft
  - Kademlia - New York University



# Verteilte Hash-Tabellen

- Allgemeine Probleme in P2P-Systemen
  - Wo soll ein Datum gespeichert werden?
    - Publish(“Inhalt“, ...)
  - Wie findet eine Anfrage den Speicherort?
    - Lookup(“Inhalt”)
  - Geringer Aufwand: Kommunikation, Speicher (Routinginformationen etc.)
  - Robust gegen Ausfälle und häufige Änderungen

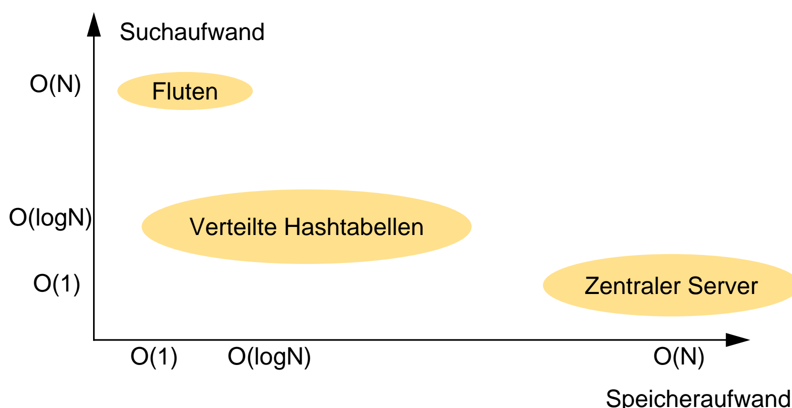
# Verteilte Hash-Tabellen

- Möglichkeiten zur Suche nach Informationen
  - Einfachste Strategie: Client-Server
    - Server speichert Lokationsinformationen
    - Bekannte Probleme: Skalierbarkeit, Aktualität der Daten, Single Point of Failure
    - Meist bestes Prinzip für einfache Anwendungen
  - Breitensuche (Fluten, wie bei Gnutella)
    - Skaliert nicht, Netzbelastung
  - Verteile Hashtabelle
    - Skalierbar: O(log N)
    - Resistent gegen Änderungen: Ausfälle, Attacken, kurzfriste Nutzer (?)


(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg

# Verteilte Hash-Tabellen

- Suchaufwand vs. Speicheraufwand





(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg

# Verteilte Hash-Tabellen

- Grundidee verteilter Hash-Tabellen
  - Verteilung von Daten über alle Knoten
  - Anforderung der Daten durch eine Anfrage beim verantwortlichen Knoten
- Zielsetzungen
  - Gleichmäßige Verteilung der Daten auf alle Knoten
  - Ständige Anpassung bei Ausfall, Beitritt und Austritt von Knoten
    - Vergabe von Zuständigkeiten an neue Knoten
    - Übernahme und Neuverteilung von Zuständigkeiten bei Austritt und Ausfall


(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg

# Verteilte Hash-Tabellen

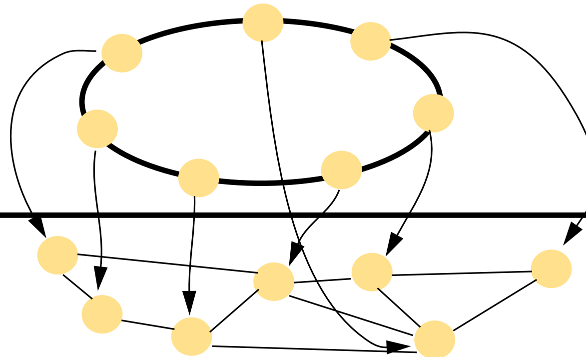
- Prinzipielle Arbeitsweise
  - Abbildung der Inhalte auf einen linearen Wertebereich
    - Meist 0,... $2^{m-1}$ >> Anzahl der gespeicherten Objekte
    - Abbildung des Inhalts in den Wertebereich durch eine Hashfunktion
  - Verteilung des Schlüsselraums über alle Knoten

(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg

# Verteilte Hash-Tabellen

- Prinzipielle Arbeitsweise
  - Jeder der Knoten ist mindestens für einen Teil des Schlüsselraumes zuständig
    - Oftmals sind auch mehrere Knoten für den gleichen Bereich verantwortlich
    - Die Zuständigkeit kann sich dynamisch ändern

Logische Sicht



Reale Topologie



(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg



# Verteilte Hash-Tabellen

- Wie werden Inhalte gespeichert?
    - Direkt - Inhalt wird direkt im für den Schlüssel verantwortlichen Knoten gespeichert
    - Indirekt - Der für den Schlüssel verantwortlich Knoten verwaltet eine Adresse welche auf die Daten verweist
- Auffinden des Inhalts
  - Suche nach Schlüsseln
    - Einstieg bei einem beliebigen Knoten
    - Routing zu gesuchtem Inhalt



(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg

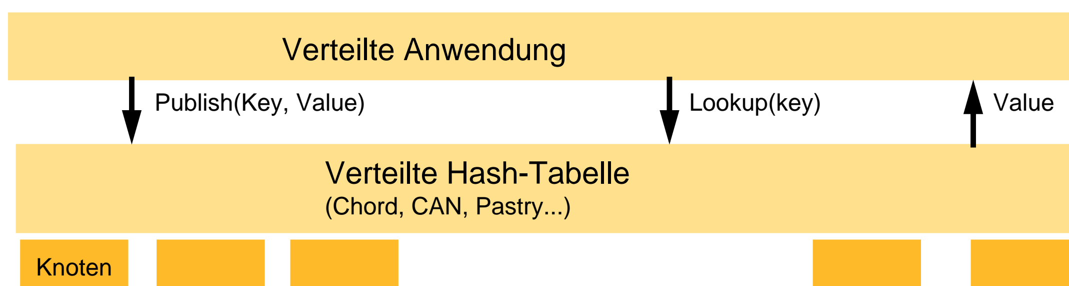
# Verteilte Hash-Tabellen

- Beitritt eines neuen Knotens
  - Zuteilung eines bestimmten Bereichs des Schlüsselraums
  - Initialisierung mit Routing-Informationen und Einbindung in die Routing-Strukturen
- Austritt eines Knotens
  - Aufteilung des Schlüsselraums auf benachbarte Knoten
  - Migration der Daten auf zuständige Knoten
- Ausfall eines Knotens
  - Nutzung redundanter Routing-Wege und Knoten
  - Erneute Zuweisung des Schlüsselraums


(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg

# Verteilte Hash-Tabellen

- Generische Schnittstelle von Verteilten Hash-Tabellen
  - Methode zur Publikation von Informationen
    - Publish(key,value)
  - Methode zum Abrufen von Informationen
    - Lookup(key) --> value
- Algorithmen/Systeme lassen sich somit auswechseln





(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg

# Verteilte Hash-Tabellen

- Grundlage bilden konsistente Hashfunktion mit folgenden Eigenschaften
  - Einwegberechenbarkeit
  - Kollisionsresistenz
  - Effiziente Berechenbarkeit
  - Gleichverteiltheit
  - Beispiel: Secure Hash (SHA-1)


(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg

# Verteilte Hash-Tabellen

- Eigenschaften von verteilten Hash-Tabellen
  - Lastenverteilung - Schlüssel werden gleichmäßig auf alle Knoten verteilt
  - Skalierbarkeit
  - Selbstorganisierend - Keine manuelle Konfiguration nötig
  - Einfache und günstige Umsetzung
  - Unterstützung vieler verschiedener Anwendungen
    - Schlüssel haben keine semantische Bedeutung
    - Verwaltete Inhalte sind anwendungsunabhängig


(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg

# Chord

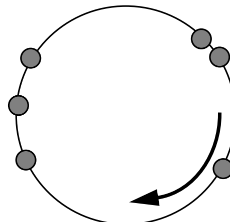
- Chord - ein skalierbares Protokoll für Internet Applikationen

- Literatur:
  - Projektseite:
    http://www.pdos.lcs.mit.edu/chord
  - 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 Transactions on Networking
  - Implementierung:
    http://www.pdos.lcs.mit.edu/chord/snapshots/



# Chord

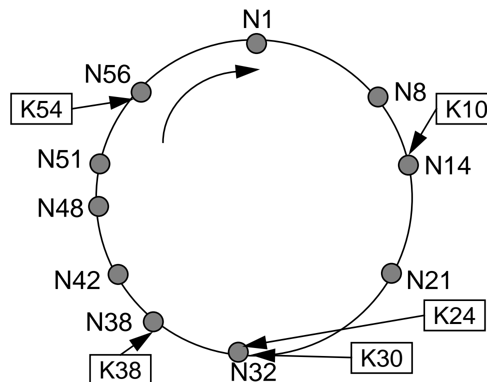
- Grundidee: Chord-Ring
  - Jeder Knoten besitzt eine eindeutige 160 Bit breite ID. Diese ist das Resultat einer Hashfunktion über die IP-Adresse des Knotens.
  - Ringförmige Anordnung der Knoten geordnet nach ihren IDs



# Chord

- Abbildung der Informationen auf die einzelnen Knoten:
  - Für jede Information wird eine eigene Hash-ID erzeugt
  - Die Information wird dem Knoten mit der unmittelbar im Ring folgenden ID übergeben

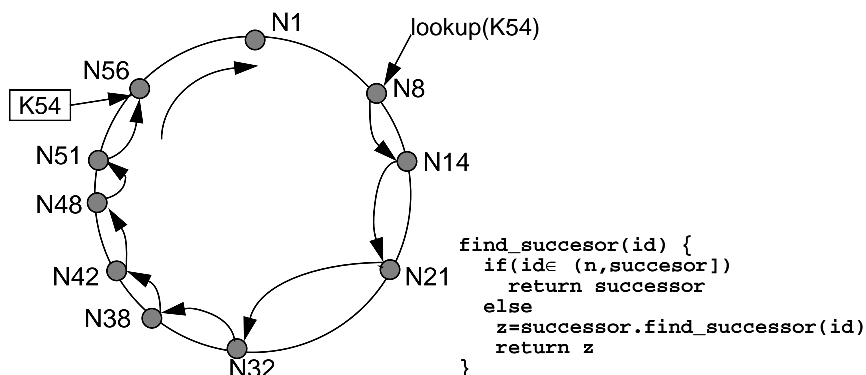
Beispiel: Schlüssellänge m=6, Anzahl der Knoten 10, 5 verwaltete IDs






# Chord

- Naive Methode um Daten zu finden in einem Chord-Ring
  - Jeder Knoten kennt seinen unmittelbaren Nachfolger und leitet eine Anfrage solange weiter, bis der gesuchte Schlüssel zwischen der eigenen ID und der des unmittelbaren Nachfolgers liegt. Dieser Nachfolger verwaltet den gesuchten Schlüssel.




```
find_succesor(id) {
  if(id∈ (n,succesor])
    return successor
  else
   z=successor.find_successor(id)
   return z
}
```

# Chord

- Für die skalierbare Lokalisierung von Daten verfügt jeder Knoten über zusätzliche Routing-Informationen:
  - Eine *Finger*-Tabelle, für die gilt, dass ein Finger in der Zeile k dem ersten Knoten im Ring nach der ID mit dem Wert $(n+2^{k-1}) \bmod 2^m,\ 1 \le k \le m$ entspricht.
    Außer der ID des Knoten wird zusätzlich die IP-Adresse des Rechners verwaltet.
  - Die ID und die IP-Adresse des unmittelbaren Nachfolgers. Diese entspricht dem ersten Eintrag in der Finger-Tabelle
  - Zusätzlich wird noch die ID und IP-Adresse des unmittelbaren Vorgängers verwaltet

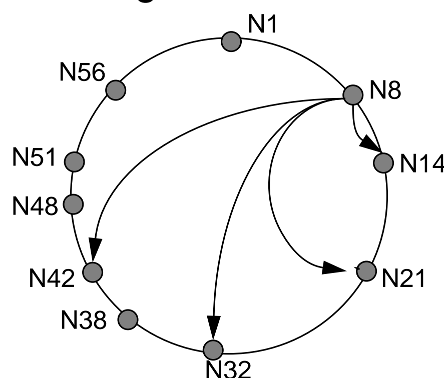


(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg


P2P.fm: 03.02.04 14:00


# Chord

- Beispiel: Finger-Tabelle des Knoten N8 (m=6)




| ID | Verantworlicher Knoten |
|---|---|
| 8+1 | 14 |
| 8+2 | 14 |
| 8+4 | 14 |
| 8+8 | 21 |
| 8+16 | 32 |
| 8+32 | 42 |

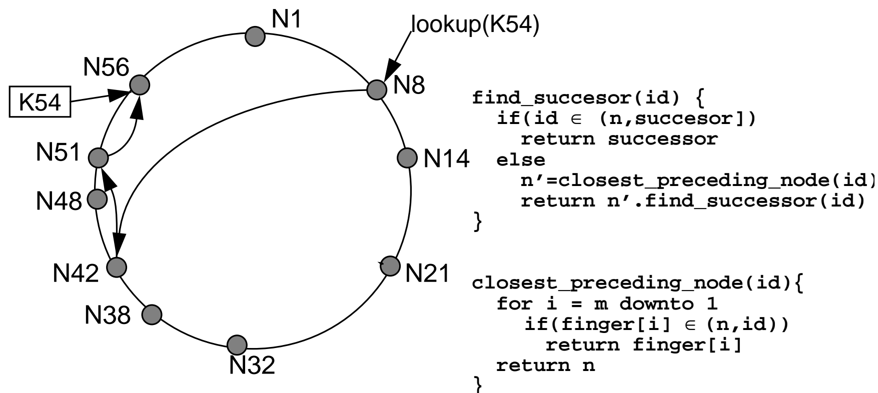


(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg


P2P.fm: 03.02.04 14:00

# Chord

- Suche nach einem Schlüssel mit Hilfe der Finger-Tabelle




- Anfrage-Aufwand O(log n)



*(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg*


# Chord

- Erzeugung eines neuen Chord-Rings:

```
create(){
        predecessor= nil
        successor =n
}
```

- Beitritt eines neuen Knotens in den Chord-Ring:

```
join(n'){
        predecessor = nil
        sucessor = n'.find_successor(n)
}
```



*(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg*

# Chord

- Um die Struktur des Chord-Rings auch bei einem oder mehreren neuen Knoten aufrecht halten zu können, ruft jedes Peer periodisch die Funktion **`stabilize()`** auf

```
stabilize(){
      x = successor.predecessor
      if( x ∈ (n,successor))
            successor = x
      successor.noftify(n)
}
notify(n’){
      if (predessor is nil or n’∈ (predecessor,n))
            predecessor = n’
}
check_predecessor(){
      if(predecessor has failed)
            predecessor = nil
}
```


(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg


# Chord

- Damit auch weiterhin ein schnelles Auffinden von Daten möglich ist, aktualisiert jeder Knoten periodisch seine Finger-Tabelle

```
fix_finger(){
      next = next + 1
      if (next > m){
            next = ⌊log(succesor - n)⌋ + 1
      }
      finger[next] = find_sucessor(n + 2^(next-1))
}
```


(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg

# Chord

- Erweiterung der Routing-Informationen um eine Nachfolgerliste
  - Es kann nun der gleichzeitige Ausfall von *r-1* aufeinanderfolgender Knoten toleriert werden, wobei *r* der Länge der Nachfolgerliste entspricht.
  - Anpassung der **`stabilize`**-Funktion, damit die Nachfolgerliste mit der Liste des eigenen Nachfolgers abgeglichen wird. Ist dieser nicht erreichbar, wird der nächste Nachfolger in der Liste benachrichtigt.


(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg

# Chord

- Erweiterung der Routing-Informationen um eine Nachfolgerliste (Fortsetzung)
  - Die Funktion **`closest_preceding_node`** wird so modifiziert, dass nicht nur in der Finger-Tabelle der Vorgänger gesucht wird, sondern zusätzlich in der Nachfolgerliste.
  - Sollte während des Aufrufs der Funktion **`find_successor`** ein Knoten ausfallen, kann dies durch den Ablauf eines Timers erkannt werden. In diesem Fall wird einfach der nächste, geeignete Knoten in der Nachfolgerliste und der Finger-Tabelle gesucht und aufgerufen.


(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg

# Chord

- Lastenverteilung
  - Simulationen bei einer Netzwerkgröße von $10^4$ Knoten und der Abbildung von $10^5$ bis $10^6$ zufälligen Schlüsseln auf den Chord-Ring ergeben eine zu ungleichmäßige Verteilung der Schlüssel.
    Es gibt Knoten die keine Schlüssel zugeteilt bekommen, andere hingegen verwalten eine unverhältnismäßig hohe Anzahl.


*(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg*

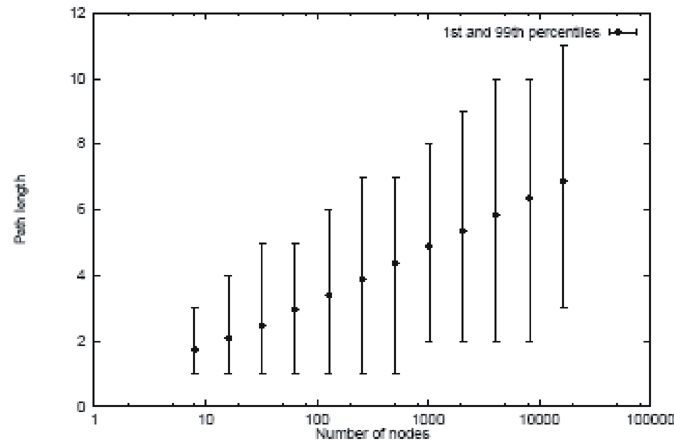
# Chord

- Lastenverteilung
  - Als Lösung wird die Einführung von mehren virtuellen IDs pro Knoten vorgeschlagen. Jeder Knoten erhält beispielsweise log(*N*) virtuelle Knoten IDs.
  - Zusätzlich können durch die Anzahl der verwalteten IDs pro Knoten Leistungsunterschiede zwischen verschiedenen Knoten berücksichtigt werden.


*(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg*

# Chord

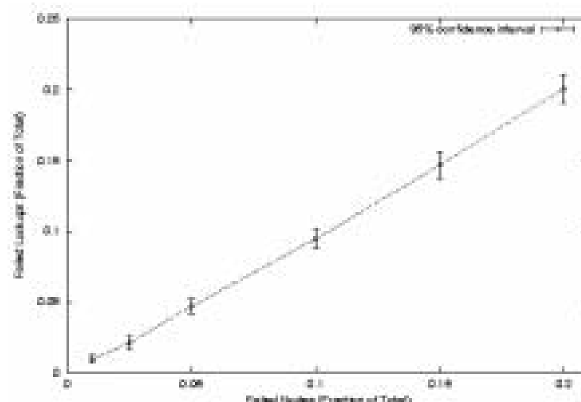
- Durchschnittliche Pfadlänge von Anfragen: $\frac{1}{2}\log_2(N)$
  - $2^k$ Knoten;100*$2^k$ Schlüssel; k=3-14






# Chord

- Auswirkung von parallelen Knotenausfällen
  - Ausgangszustand: $10^4$ Knoten, $10^6$ Schlüssel
  - Messung: Ein Teil der Knoten fällt aus und der Stabilisierungsprozess wird abgewartet. Es findet keine Replikation von Schlüsseln statt.



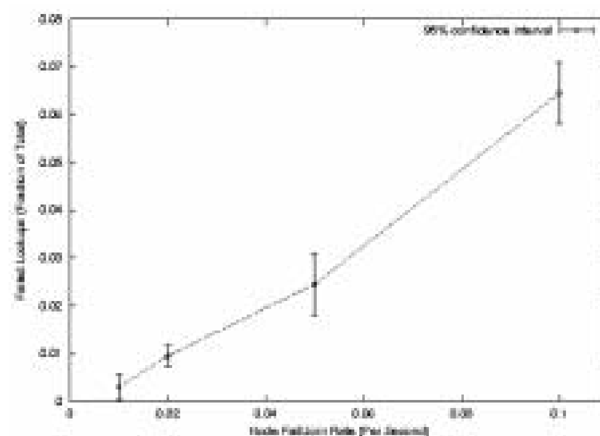
# Chord

- Auswirkung von kontinuierlichen Ausfällen und Beitritten von Knoten
  - Ausgangszustand:
    - 500 Knoten
    - Durchschnittlich 1 Anfrage pro Sekunde
    - Jeder Knoten führt alle 30 Sekunden die Routinen zur Stabilisierung des Netzwerks auf
  - Beitritte oder Ausfälle von Knoten werden durch einen Poisson-Prozess mit der Ankunftsrate R erfasst. Eine Rate von 0.01 entspricht dem durschnittlichen Ausfall oder Beitritt eines Knotens pro 100 Sekunden.


*(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg*

# Chord

- Auswirkung von kontinuierlichen Ausfällen und Teilnahmen von Knoten





*(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg*

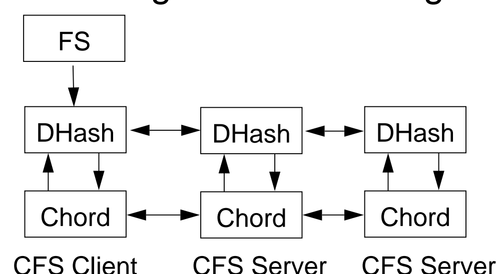
# Chord

- Berücksichtigung der realen Netzwerkstruktur
  - Einfaches Verfahren:
    - Auswahl des nächsten befragten Knotens nicht nur auf Grund seiner Nähe zum gesuchten Schlüssel, sondern zusätzlich auf Grund der Verbindungsqualität
  - Komplexeres Verfahren mit Erweiterung der Routing-Information:
    - Zu jedem Knoten der Finger-Tabelle wird zusätzlich eine Nachfolgerliste verwaltet
    - Nachrichten werden an Knoten mit der besten Verbindung aus der Liste weitergeleitet


(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg

# Chord

- Cooperative File System (CFS), ein verteiltes Dateisystem basierend auf Chord
- Allgemeiner Aufbau
  - CFS: Interpretiert Blöcke als Teile von Dateien oder Verwaltungsstrukturen des Dateisystems
  - DHash: Verwaltet und repliziert Datenblöcke
  - Chord: Lokalisierung und Verwaltung von Hash-Schlüsseln





(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg

# Chord

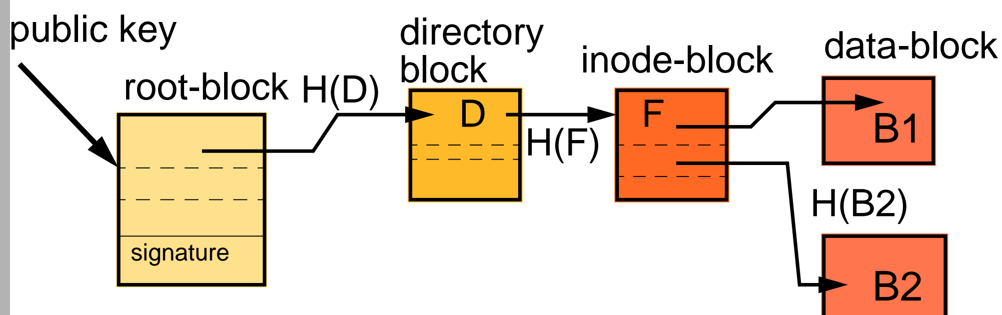
## DHash

- Replikation von Datenblöcken:
  - Jeder Datenblock wird von k aufeinander folgenden Servern repliziert
- Caching von Datenblöcken:
  - Jeder Server verfügt über einen Cache, der nach dem least-recently-used Verfahren aktualisiert wird
  - Bei einer erfolgreichen Datenübertragung übermittelt der Initiator anschließend die Daten an alle Knoten der Anfragekette. Populäre Daten werden so stark repliziert und die Anfragelast verteilt sich auf mehrere Knoten.


(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg


# Chord

## FS-Schicht




- Root-Block wird abgesichert durch eine Signatur
- Die Integrität von Directory-Blocks, Inode-Blocks sowie Data-Blocks wird durch den Hash über ihren Inhalt gewährleistet


(C) 2003 Hans P. Reiser, Rüdiger Kapitza, Informatik 4, Universität Erlangen-Nürnberg