
Seminar

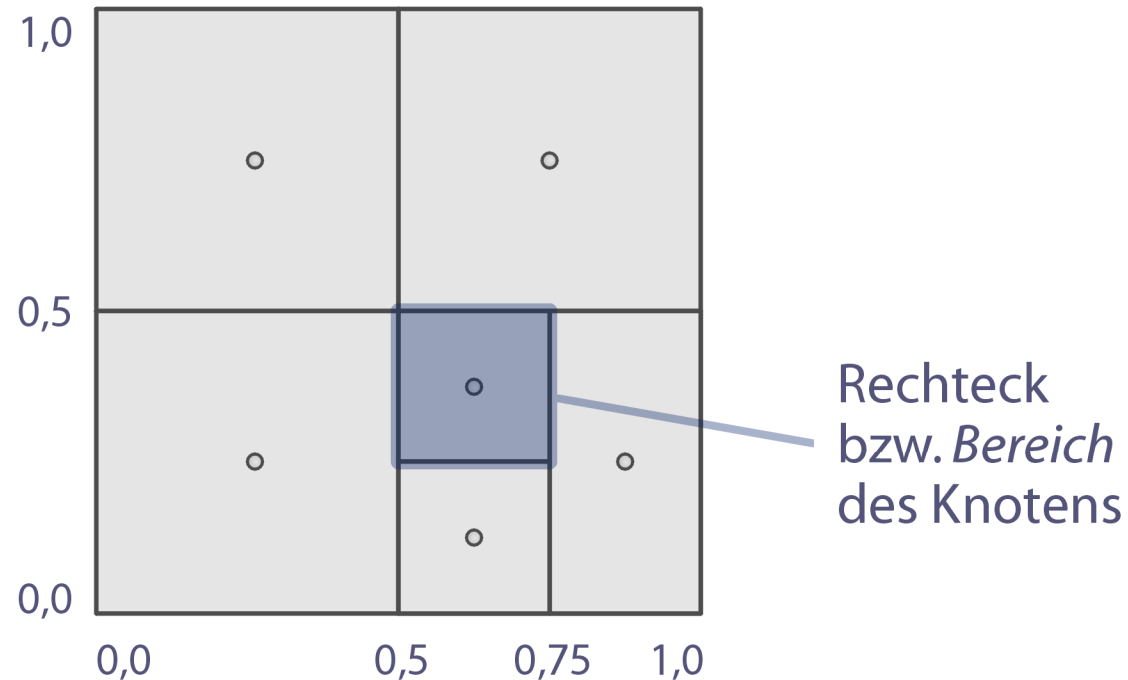
Mehrbereichsabfragen in DHTs II auf Basis von CAN

Benjamin Schnaidt

Prof. Dr.-Ing. Georg Carle
Lehrstuhl für Rechnernetze und Internet
Universität Tübingen

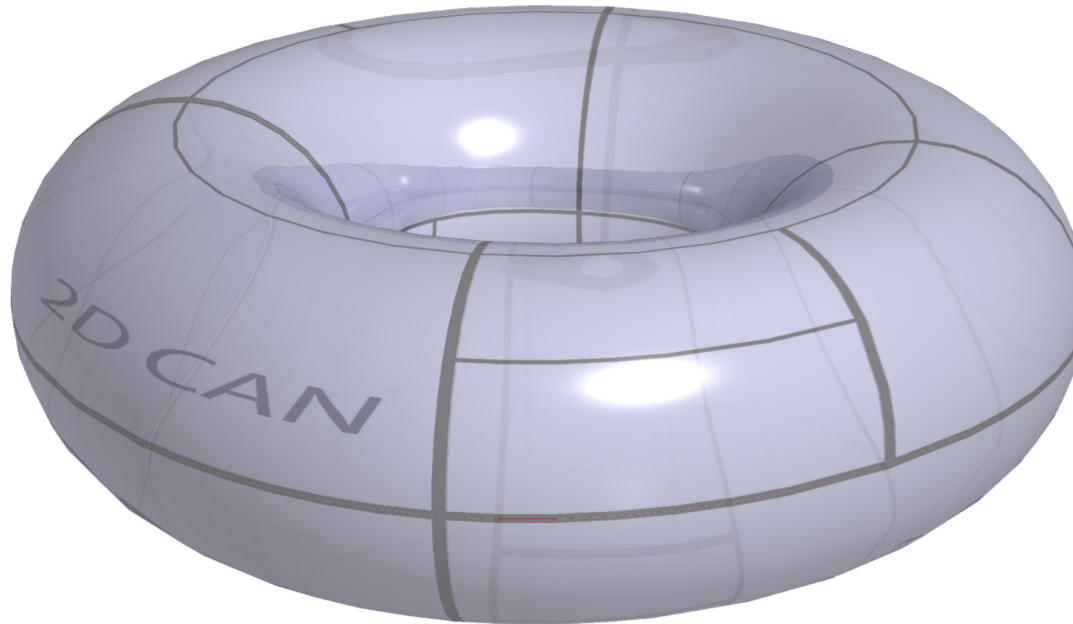
CAN (Content-Addressable Network)

- CAN basiert auf einem d-dimensionalen Raum (2-dimensional eine Fläche).
- Dieser Raum wird in *Hyperquader* (2-dimensional Rechtecke) aufgeteilt, die jeweils zu einem Knoten des Netzwerks gehören.



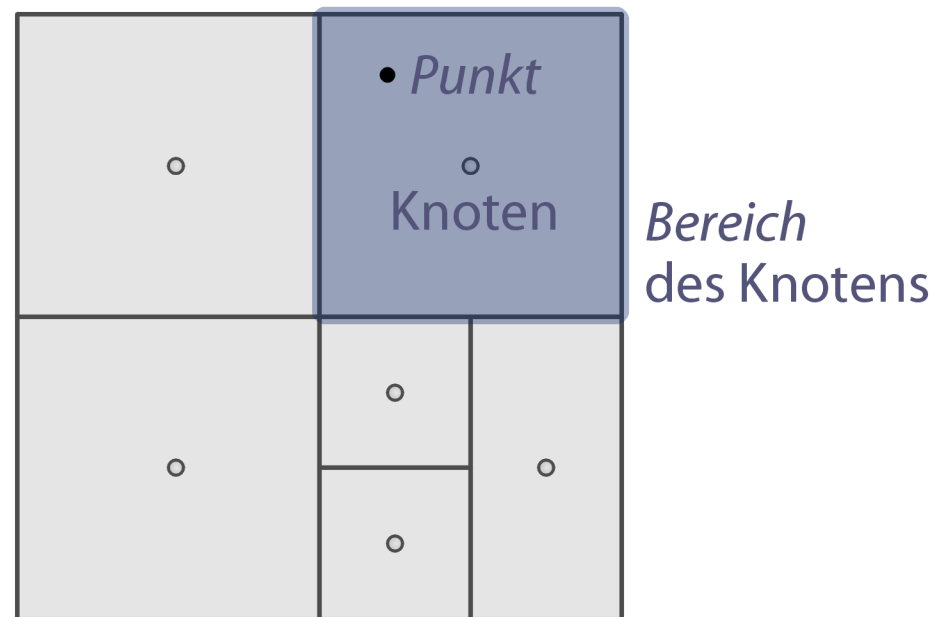
CAN - Torus

- Jede Dimension ist auf dem Intervall $[0; 1]$ definiert.
- Werte außerhalb werden ringförmig auf $[0; 1]$ abgebildet.
- 2-dimensional ergibt sich ein Torus.



CAN - Hash-Tabelle und Einfügen von Keys (Insert)

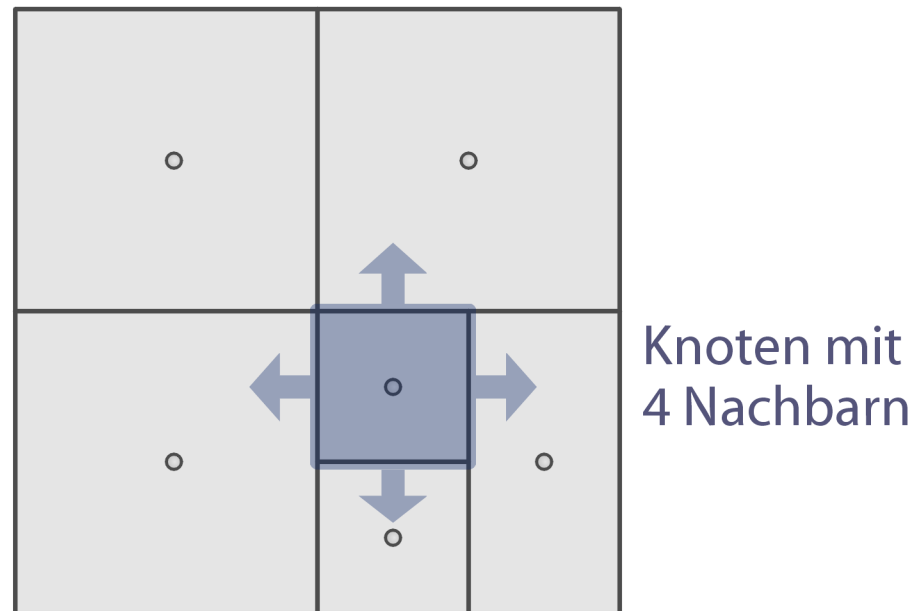
- Eine Hash-Tabelle besteht aus (Key, Value) Paaren.
- Bei CAN wird jeder Key mit einer uniformen Hash-Funktion auf einen *Punkt* im Raum abgebildet.
- Der Knoten, zu dessen *Bereich* der *Punkt* gehört, speichert das (Key, Value) Paar.



CAN - Nachschlagen von Keys (Lookup)

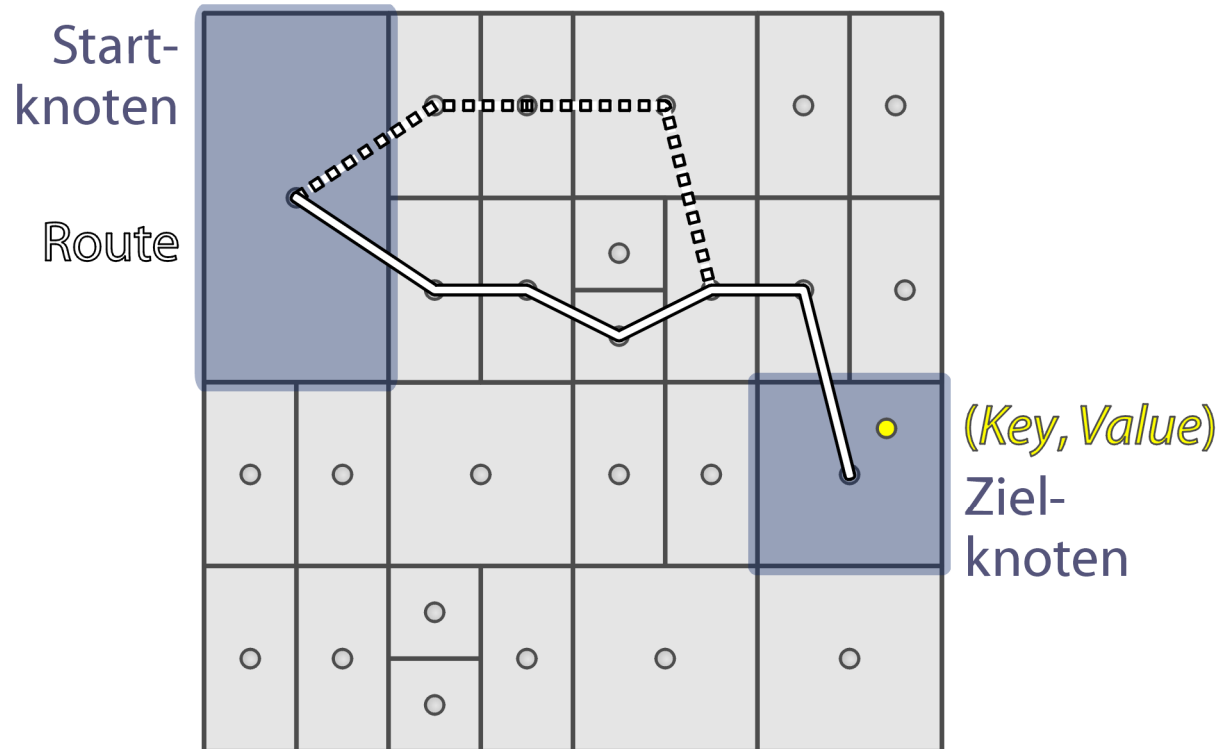
Ermitteln des *Value* zu einem *Key*:

- Anwenden der Hash-Funktion auf den *Key* ergibt den (Ziel-) *Punkt*.
- Senden einer Nachricht zum Knoten mit dem *Punkt*.
 - Für das Routing speichert jeder Knoten die IP-Adressen der Nachbarn.



CAN - Routing

Beim Routing wird die Nachricht jeweils an den Nachbarknoten weitergeleitet, der dem Zielknoten am nächsten ist.



Die Kosten bei CAN sind:

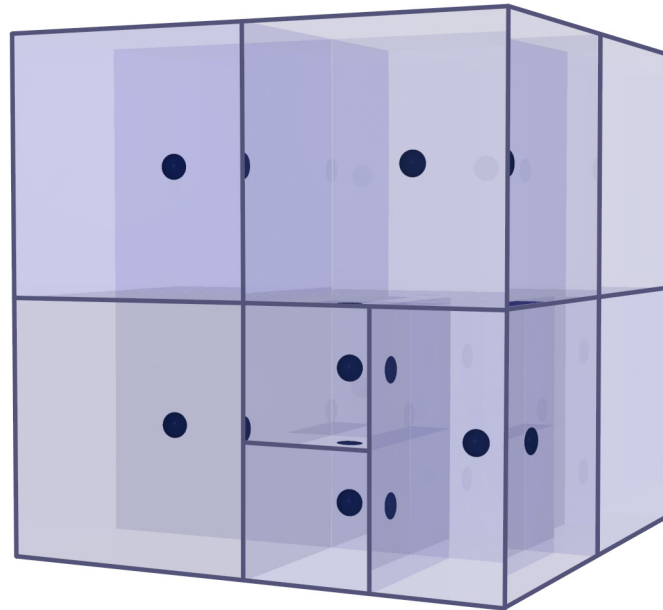
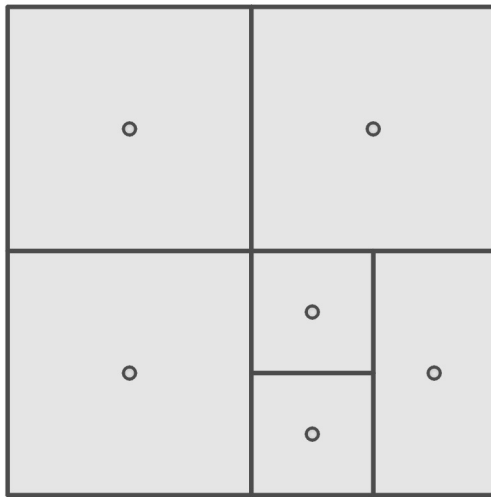
- *Speicheraufwand* : **$O(d)$**
 - Bei 2 Dimensionen 4 Nachbarn
 - Bei d Dimensionen $2d$ Nachbarn
- *Suchaufwand* : **$O(n^{1/d})$**
 - n Knoten im Netzwerk
 - Durchschnittliche Pfadlänge vom Start- zum Zielknoten $(d/4)(n^{1/d})$

Vergleich mit *Chord*:

- Mit $d=(\log_2 n)/2$ erhält man einen *Speicher-* und *Suchaufwand* von $O(\log n)$.
- Allerdings wird ein konstanter Wert für d empfohlen.

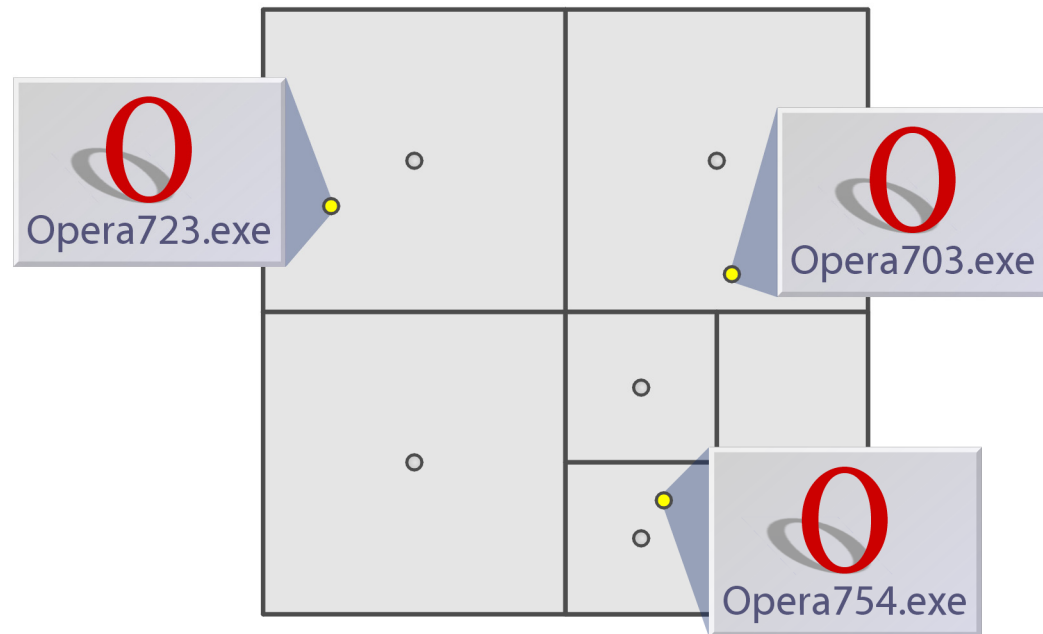
CAN - Erweiterungen

- Es sind viele Erweiterungen möglich, die das Routing schneller und gleichzeitig das Netzwerk ausfallsicherer machen.
- Beispielsweise mehr Dimensionen:



Mehrbereichsabfragen

- DHTs mit (*Key*, *Value*) Paaren werden oft beim *File Sharing* verwendet.
- Eine Hash-Funktion bildet den *Dateinamen* auf einen *Key* ab.
- Gespeichert wird (*Key*, *IP-Adresse*), z. B. (1342, "217.81.152.218").
- *Bereichsabfragen*
hier als Problem:
 - Opera7???.exe
über Einzelanfragen
 - Opera7*
nicht möglich
 - Passende *Keys*
sind im Netzwerk verstreut



Mehrbereichsabfragen

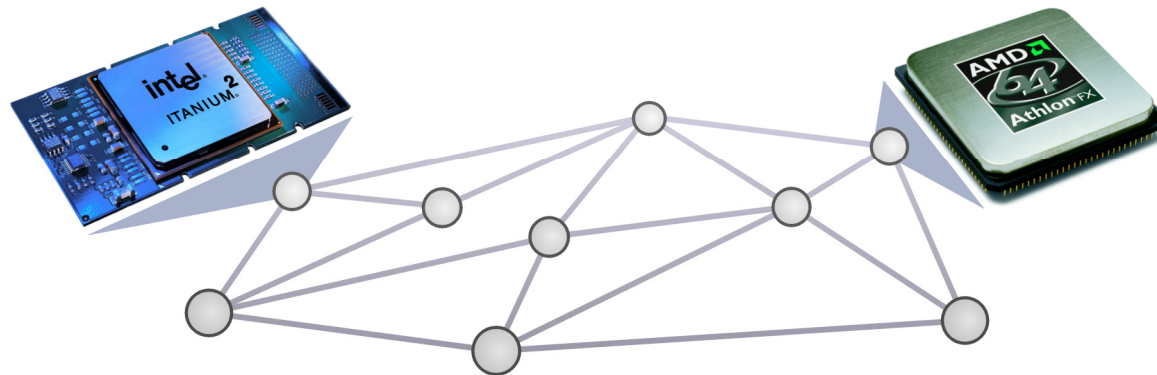


Teil 1

1 - Informationsdienste für Computational Grids

- *Computational Grids* verbinden viele Server, um Rechenleistung oder Festplattenkapazität zu teilen.
- Ein *Informationsdienst* verwaltet die verfügbaren Ressourcen.
- Jeder Server wird durch eine Reihe von *Attributen* beschrieben.

Attributwerte	Attribute	Attributwerte
Intel Itanium 2	Architektur	AMD Athlon 64 FX
Windows Server 2003	Betriebssystem	SUSE LINUX E.S. 9
15 %	CPU-Auslastung	21 %
341 MB	Speicher (frei)	823 MB



1 - CAN als Indexdienst

- Zentraler Bestandteil des *Informationsdienstes* ist der *Indexdienst*.
- Mit CAN ist solch ein *Indexdienst* realisierbar.

Vorteile

- + Selbstorganisierendes Netzwerk
- + Robust gegenüber Fehlern
- + Sehr gut skalierbar (*Such- und Speicheraufwand*)

Nachteile

- Keine Mehrbereichsabfragen
- Schnell ändernde Attribute (Z. B. CPU-Auslastung)

1 - Abfragen

- Jedes *Attribut* hat eine eigene Indexstruktur und einen global bekannten Datentyp (Z. B. *Enumeration*, *Integer*, *Float*).

Architektur = Intel Itanium 2
ein Wert



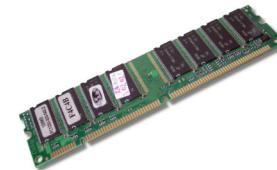
Betriebssystem = Windows Server *
mehrere Werte



CPU-Auslastung $\leq 10\%$
Bereich von Werten



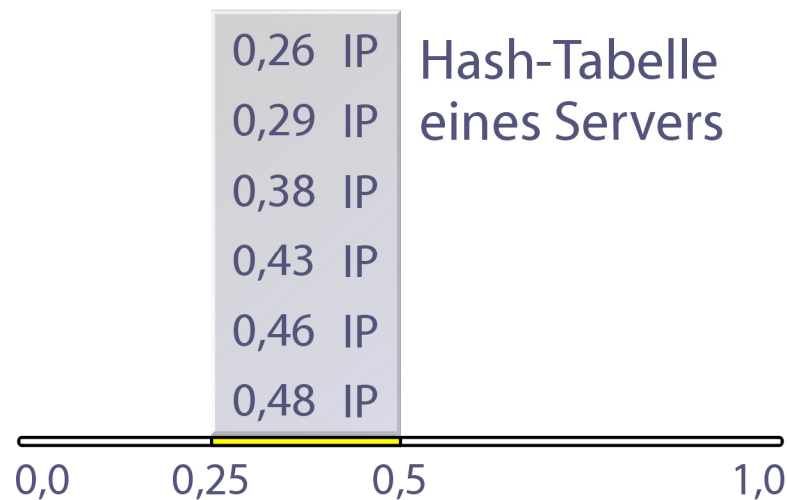
Speicher (frei) $\geq 512\text{ MB}$
Bereich von Werten



- Abfragen mit mehreren *Attributen* werden durch *Joins* verbunden.

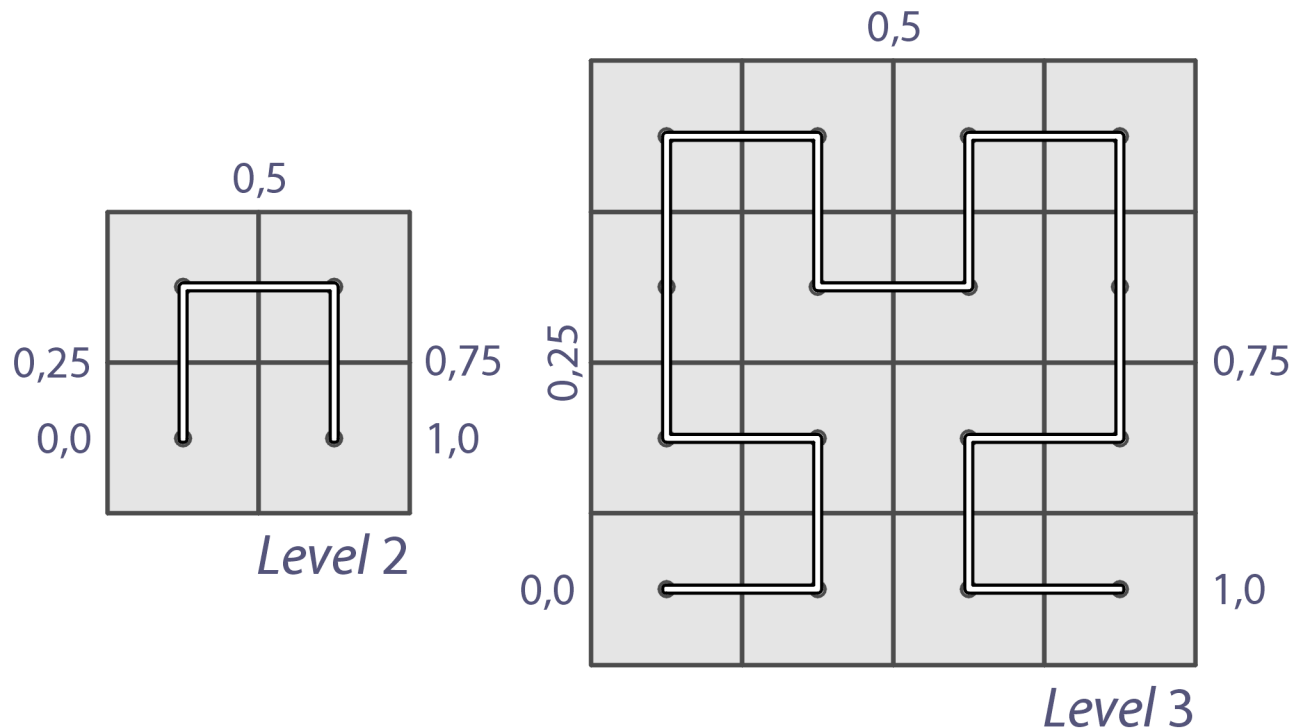
1 - Hash-Tabelle

- Im Folgenden wird nur ein *Attribut* betrachtet.
- Das *Attribut* liegt im Intervall $[0; 1]$.
- (*Key, Value*) Paare werden für (*Attributwert, IP-Adresse*) verwendet.
- Z. B. ("0,15", "217.81.152.218") für 15 % CPU-Auslastung bei der IP.
- Jeder Server verwaltet die Paare eines *Teilintervalls* von $[0; 1]$.



1 - Hilbertfunktion (1)

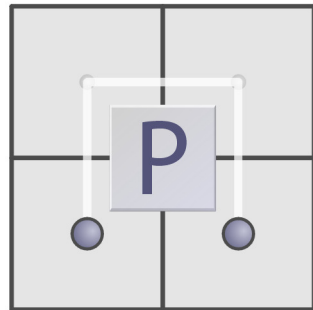
- Die Abbildung eines *Teilintervalls* auf einen *Bereich* im CAN geschieht mit der raumfüllenden *Hilbertfunktion*.
- **Bedingung:** Bei Aufteilung eines *Teilintervalls* in z. B. zwei *Teilintervalle* muss sich der zugehörige *Bereich* in zwei *Bereiche* aufteilen lassen.



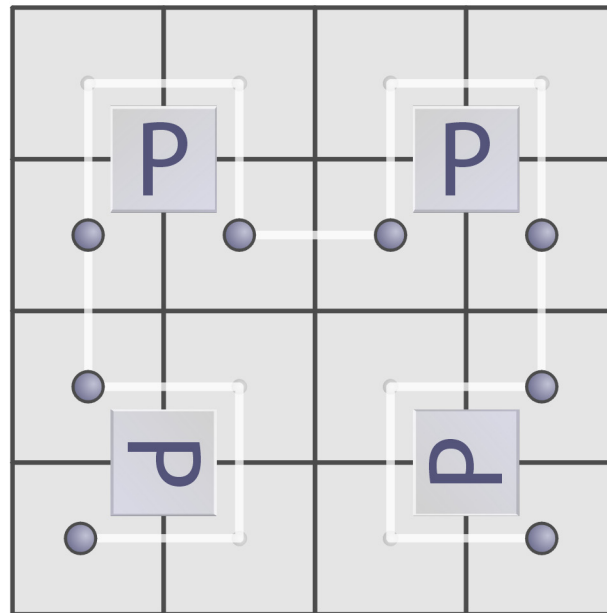
1 - Hilbertfunktion (2)

Rekursive Konstruktion mit einem einfachen Schema:

- Jeder *Bereich* wird in vier *Bereiche* aufgeteilt.
- Das Muster von *Level l* wird viermal für *Level l+1* verwendet.



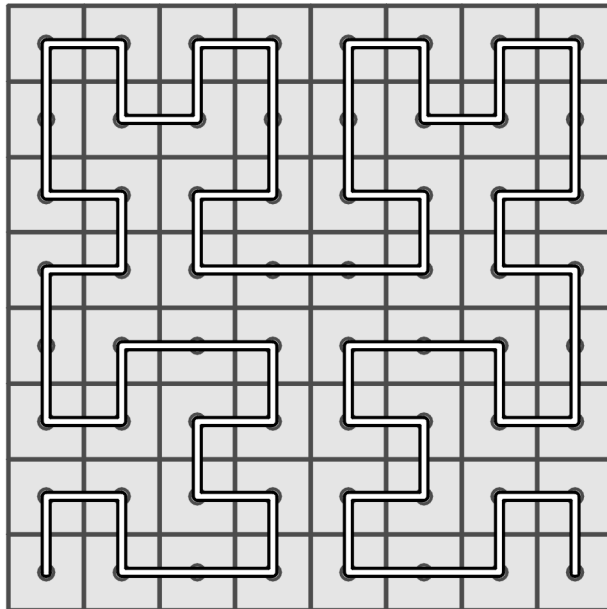
Level 2



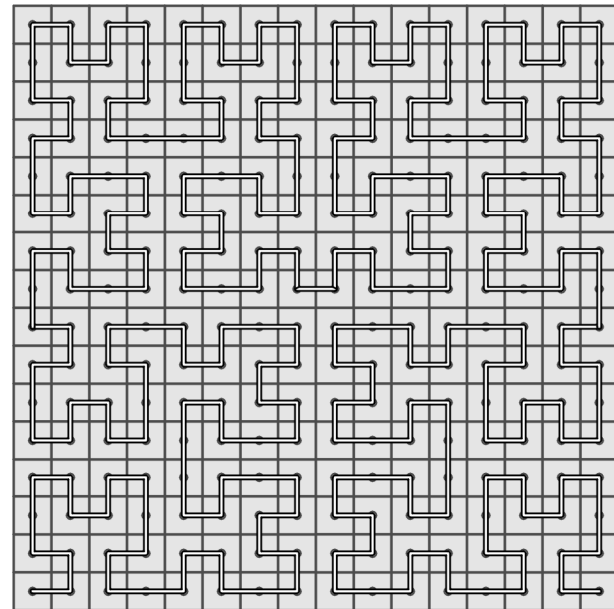
Level 3

1 - Hilbertfunktion (3)

Fortsetzung des rekursiven Schemas:



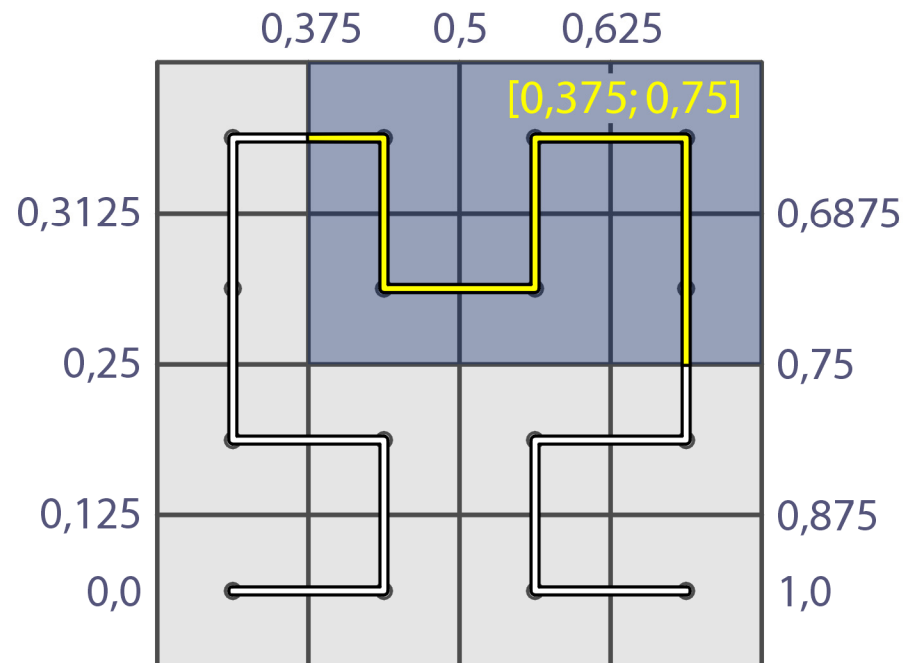
Level 4



Level 5

1 - Bereichsabfragen (Beispiel)

- Annahme: Dimension d und Level l sind global bekannt.
- Berechnen des Rechtecks (*Hyperquaders*) zum *Intervall der Abfrage*.
- Schicken einer Nachricht an die entsprechenden Knoten.



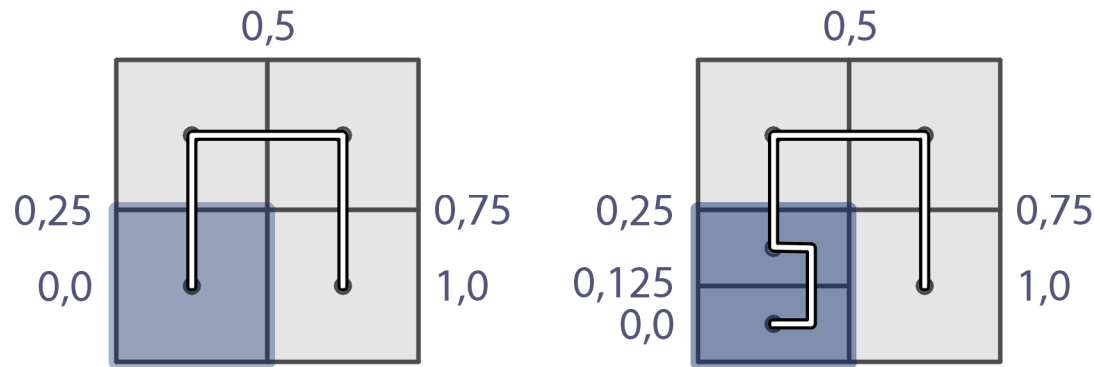
1 - Dynamische Anpassung

Stärkere Belastung eines Servers durch:

- Dichte Verteilung der Werte in einem bestimmten *Teilintervall*
- Häufige Updates von Werten

Lösung:

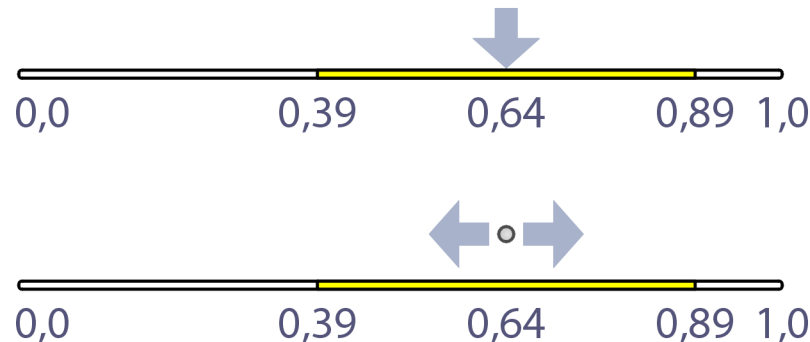
- Dynamische Anpassung der betreffenden *Bereiche*
- Aufspaltung in zwei *Bereiche* falls Anzahl berichtender Server über Schwellwert



1 - Bereichsabfragen

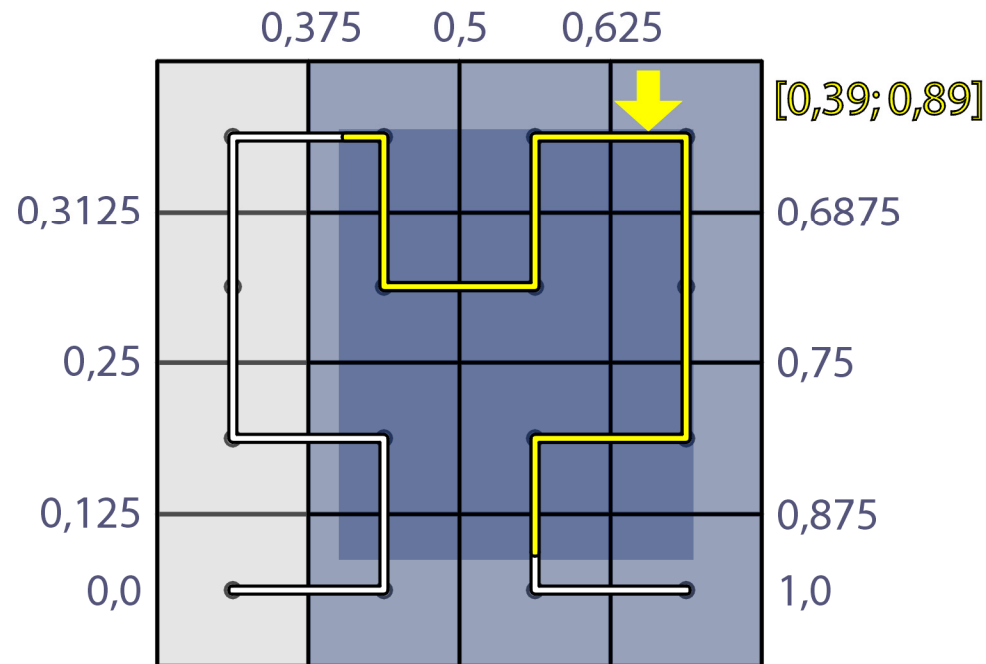
Allgemeine Vorgehensweise:

- Berechnung der Mitte des *Intervalls der Abfrage*.
- Senden einer Nachricht an den Server mit dem passenden *Bereich*.
- Von dort aus rekursive Weiterleitung an die Nachbarn, bis alle nötigen Server besucht worden sind.
- Definiert werden drei verschiedene Strategien.



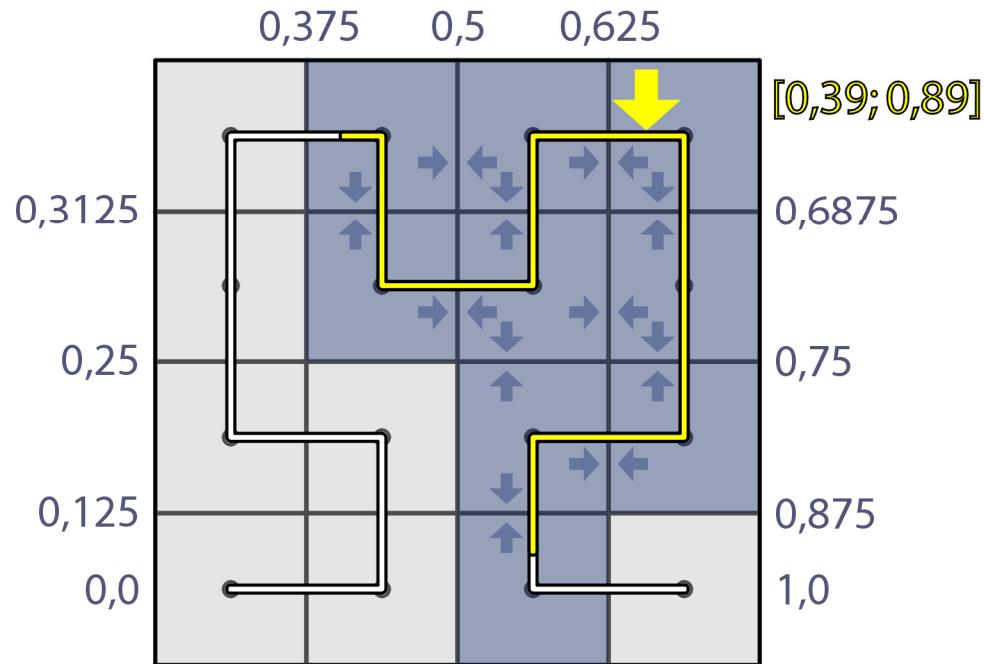
1 - Strategie 1: Normales Fluten

- Berechnen des kleinsten Rechtecks (*Hyperquaders*), welches alle nötigen *Teilintervalle* bzw. *Bereiche* abdeckt.
- Fluten vom mittleren Server aus ähnlich zur *Breitensuche* (*BFS*).
- Nachteil: Nicht alle besuchten Knoten gehören tatsächlich zur Abfrage.



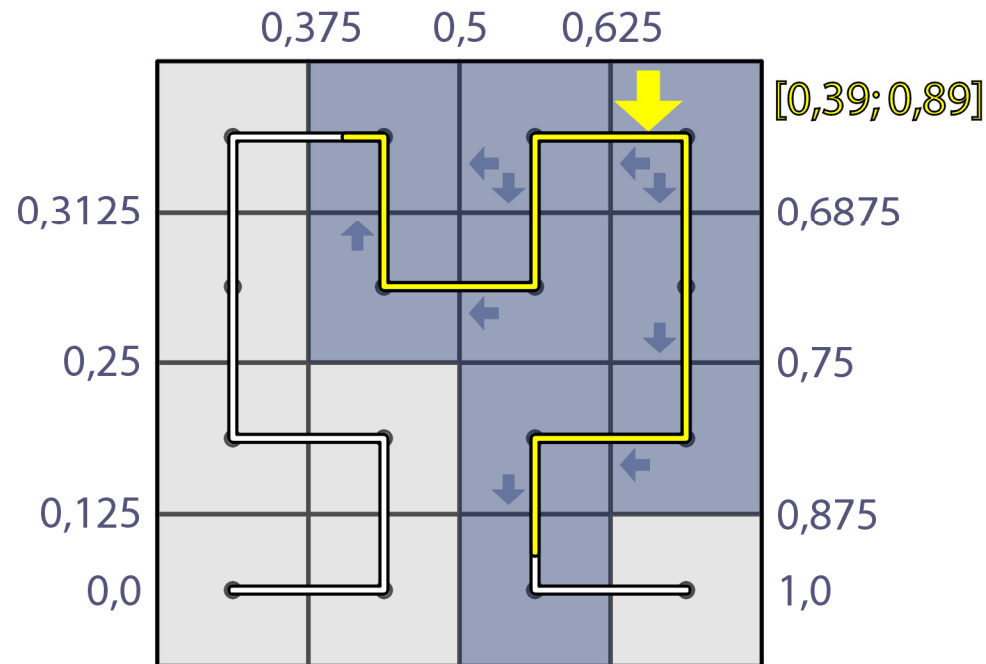
1 - Strategie 2: Kontrolliertes Fluten

- Weiterleitung nur an Nachbarn, die zum Intervall der Abfrage gehören.
- Nachteile
 - Ein Server erhält evtl. mehrere Nachrichten zur selben Abfrage.
 - Die Abarbeitung kann weniger parallel als bei Strategie 1 sein.



1 - Strategie 3: Gerichtetes kontrolliertes Fluten

- Strategie 2 und Nachricht wird in zwei Schritten weitergeleitet:
 - 1. Schritt: Weiterleitung nur an *Bereiche* mit höherem *Teilintervall*.
 - 2. Schritt: Weiterleitung nur an *Bereiche* mit niedrigerem *Teilintervall*.
- Vorteil: Weniger duplizierte Nachrichten als bei Strategie 2.

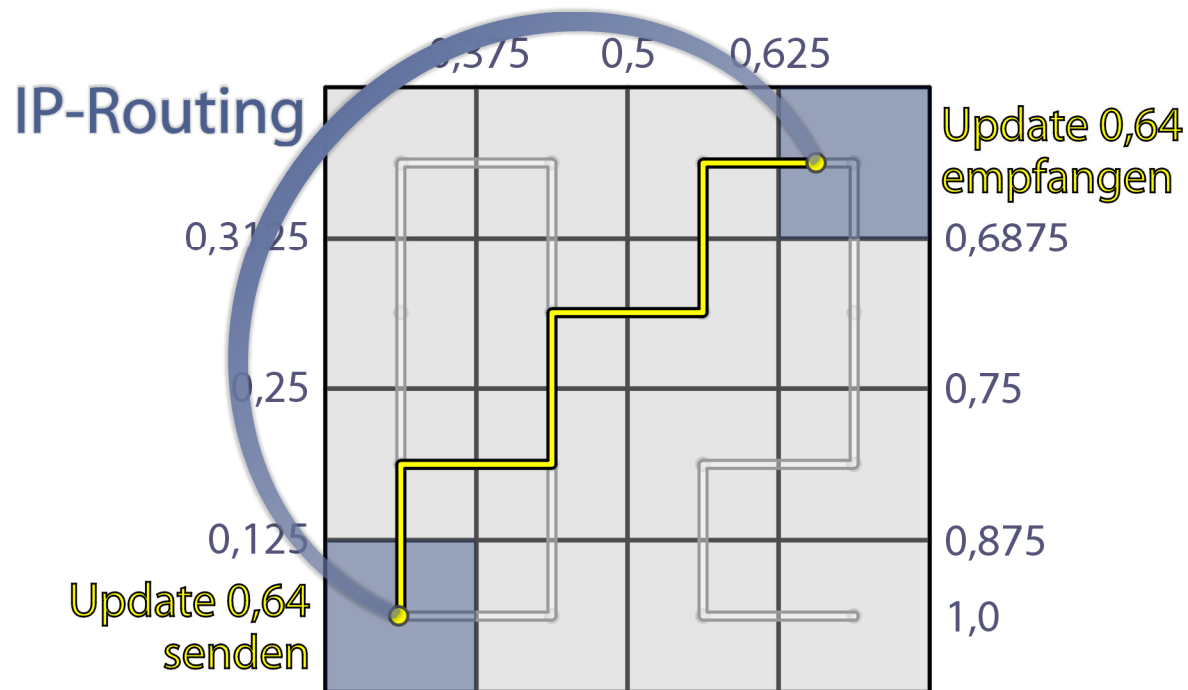


1 - Updates

- Ein Server sendet seinen aktuellen *Attributwert* in regelmäßigen Abständen zum Server mit dem passenden *Teilintervall* bzw. *Bereich*.
- Erhält der *Empfänger* in der nächsten Updaterunde keine neue Nachricht, wird der Eintrag (*Attributwert*, *IP*) gelöscht.
- Vorteile
 - + Keine zusätzlichen Nachrichten bei Änderung des *Attributwerts*.
 - + Keine Benachrichtigung bei Aufspaltung oder Änderung des *Bereichs* des *Empfängers* nötig.
 - + Bei Ausfall des *Empfängers* stehen nach einer Updaterunde die Einträge wieder zur Verfügung.
 - + Damit: Hohe Fehlertoleranz und selbstorganisierendes Netzwerk
- Nachteil
 - Langsames CAN-Routing wird verwendet.

1 - Updates mit Cache (1)

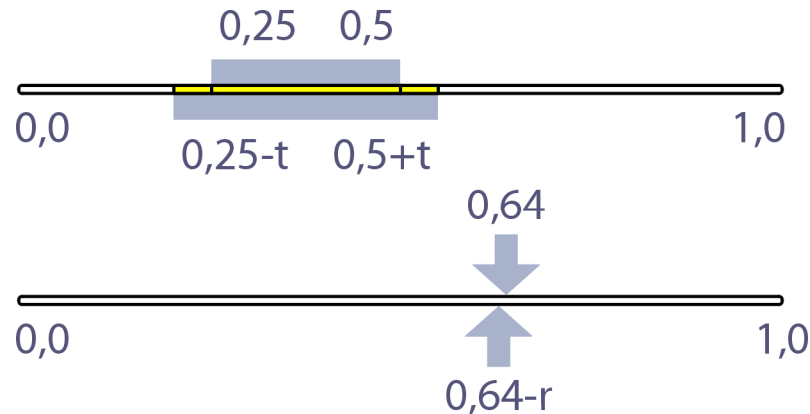
- CAN-Routing vom *Sender* zum *Empfänger*.
- Der *Empfänger* sendet seine IP und sein *Teilintervall* zurück.
- Der *Sender* cacht beides für das IP-Routing.



1 - Updates mit Cache (2)

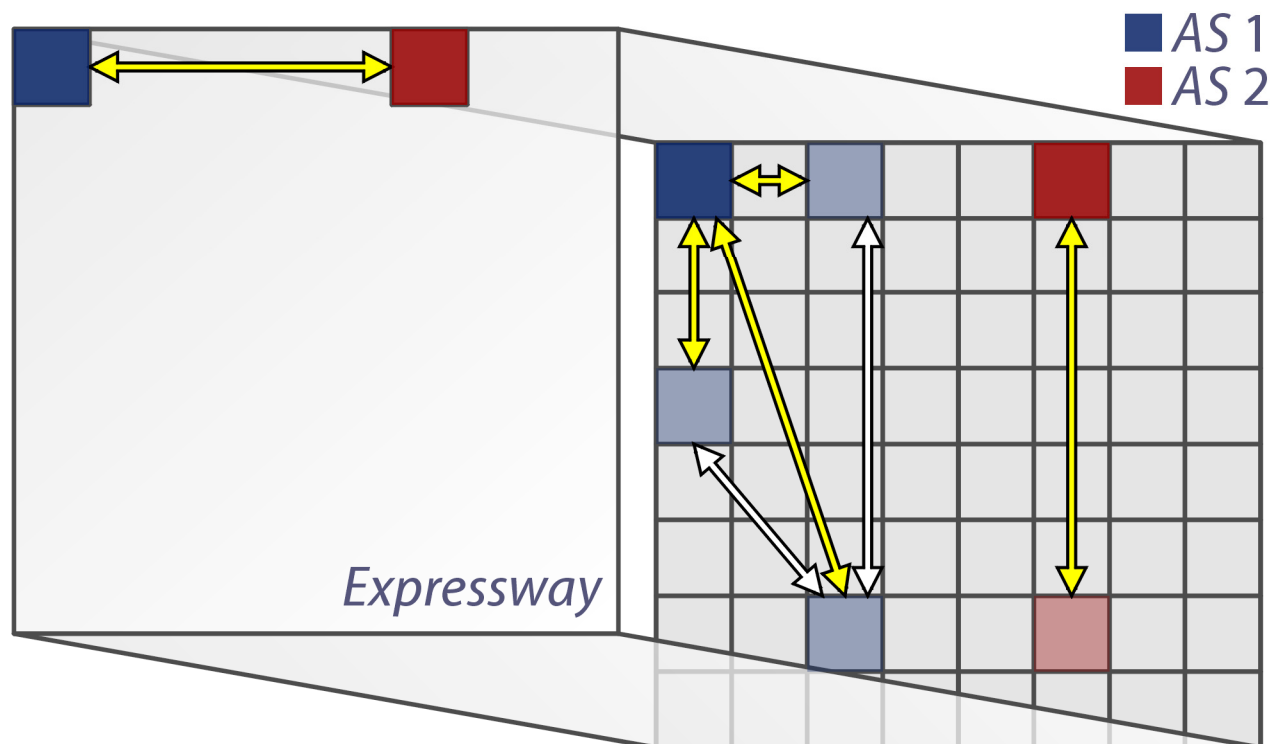
Verbesserungen bei "ungefähren" Anfragen:

- *Toleranzlevel t*
 - Vergrößerung des im Cache gespeicherten *Teilintervalls*.
 - Ermöglicht effizientes Caching bei kleinen *Teilintervallen*.
- *Randomisierungslevel r*
 - Der *Sender* randomisiert seinen *Attributwert* um $[-r; r]$.
 - Verteilt die Last auf angrenzende Server um den *Empfänger*.



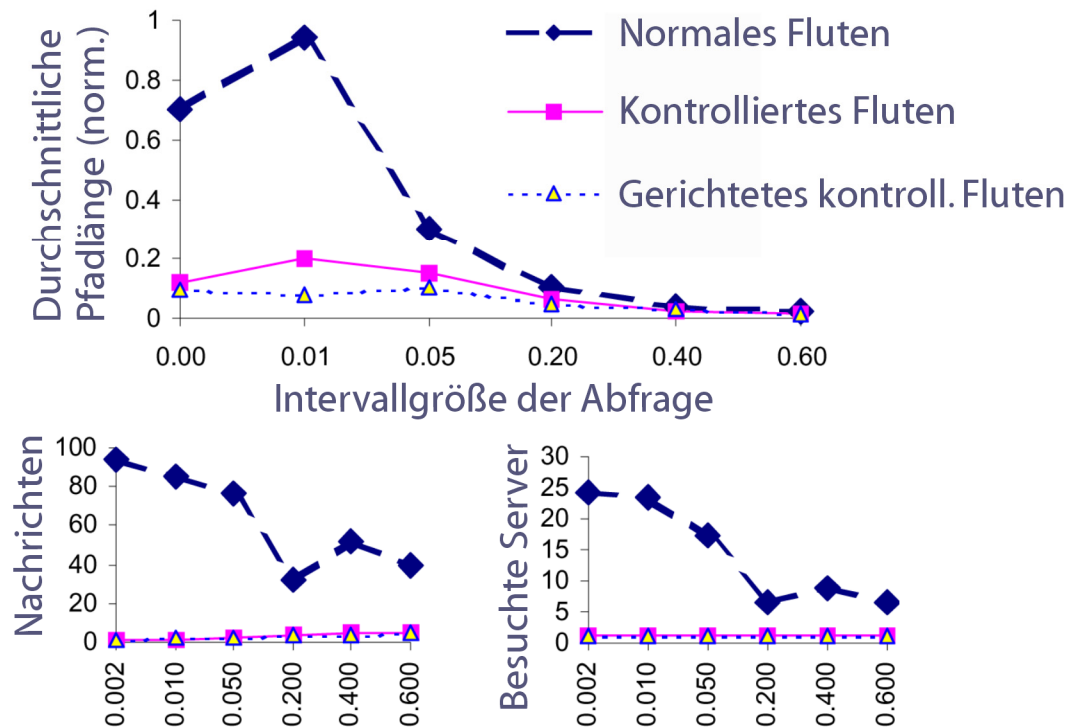
1 - Updates mit Expressway

- *Expressway Knoten* liegen in der Nähe von Gateways oder Routern.
- *Normale Knoten* bauen zum *Expressway Knoten* ihres *autonomen Systems (AS)* eine Verbindung auf. \Rightarrow Fast optimales Routing



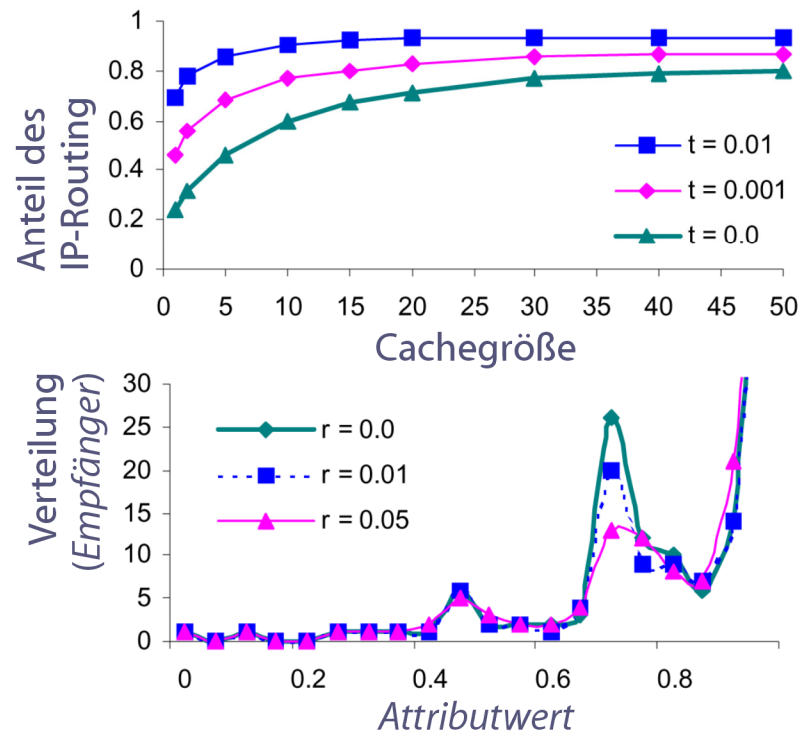
1 - Simulation (Bereichsabfragen)

- CPU-Auslastung als *Attribut*.
- Protokoll eines Rechenzentrums liefert Verteilung der *Attributwerte*.
- Netzwerk mit 1000 Servern und CAN mit 4 Dimensionen.



1 - Simulation (Updates)

- 33 Updates alle 5 Minuten für 24 Stunden.
- Effekt von *Toleranzlevel* t und *Randomisierungslevel* r untersucht.

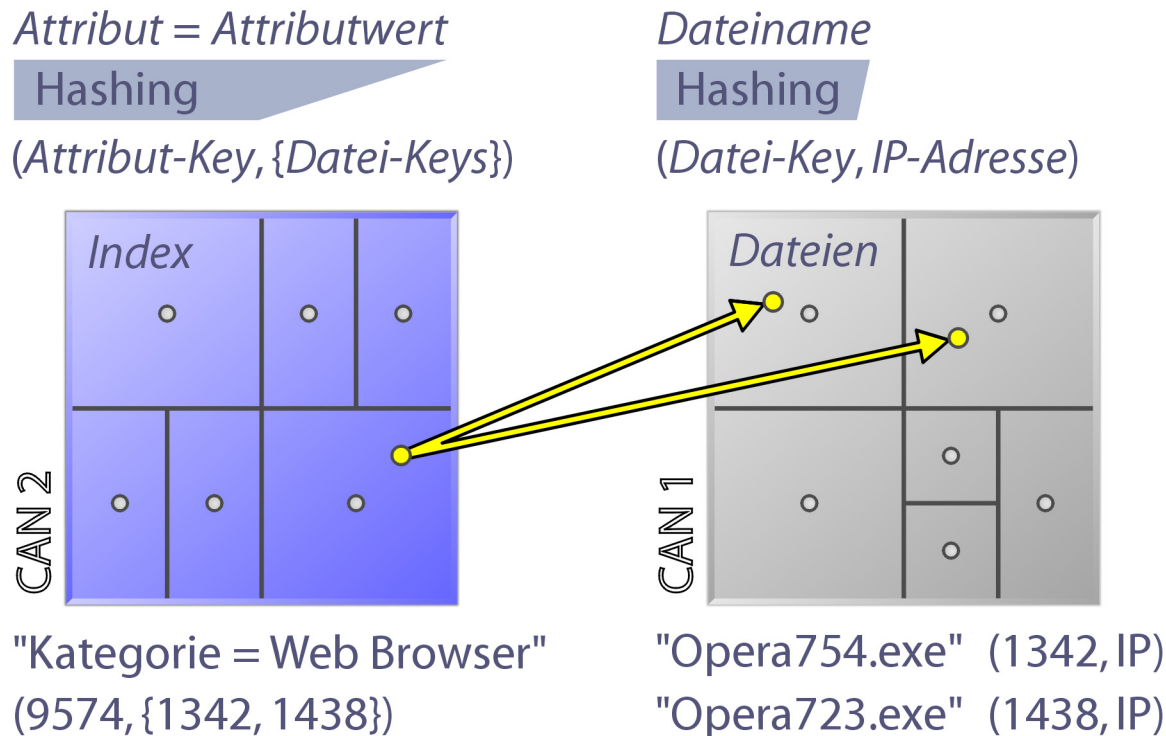


Mehrbereichsabfragen

Teil 2

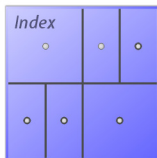
2 - CANDy (Content-Addressable Network DirectorY)

- Eine *Datei* wird durch den *Datei-Key* und *Attributwerte* beschrieben.
- CAN 1 (*Dateien*): Nachschlagen von einzelnen *Datei-Keys*.
- CAN 2 (*Index*): Index für *Abfragen* nach *Attributwerten*, liefert *Datei-Keys*.



2 - Deskriptor eines Attributs

- Der *Deskriptor* wird im *Index* gespeichert.
- Liefert den *Datentyp* eines *Attributs* und die Art der Suche nach gegebenen *Attributwerten*.



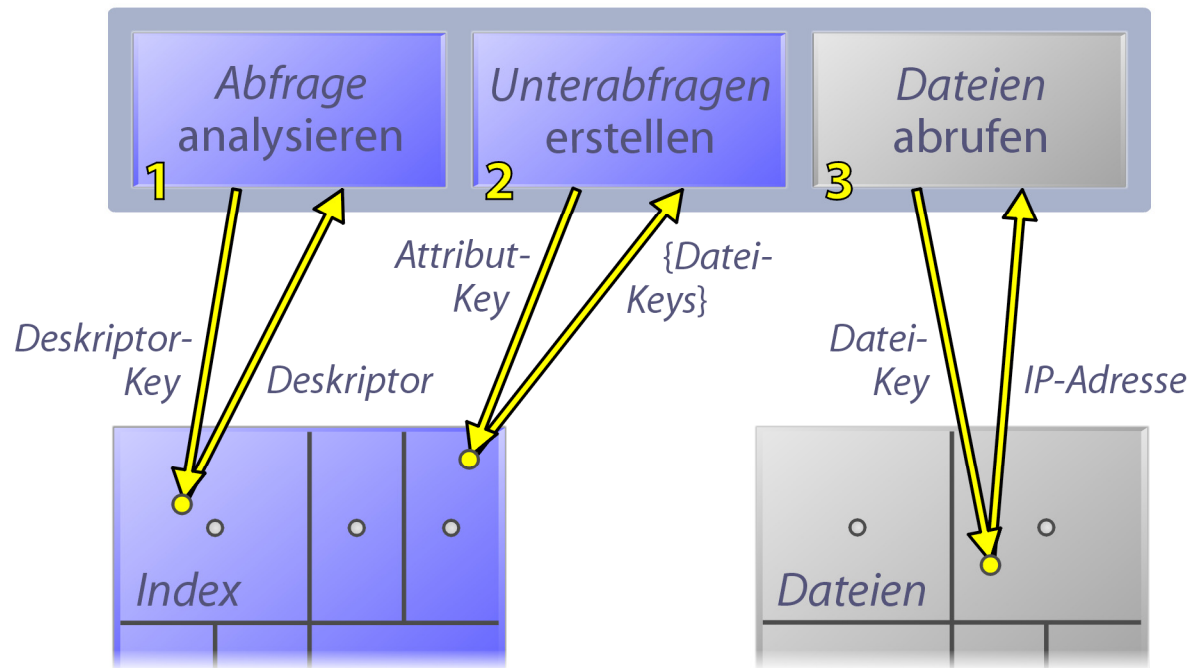
Attribut
Hashing
(*Deskriptor-Key, Deskriptor*)

<i>Attribut</i>	<i>Datentyp</i>	<i>Suche</i>	<i>Beispielwert</i>
Dateiname	String	Präfix	Opera754.exe
Dateigröße	Integer	Bereich	3,67 MB
Kategorie	String	Wortmenge	Web Browser



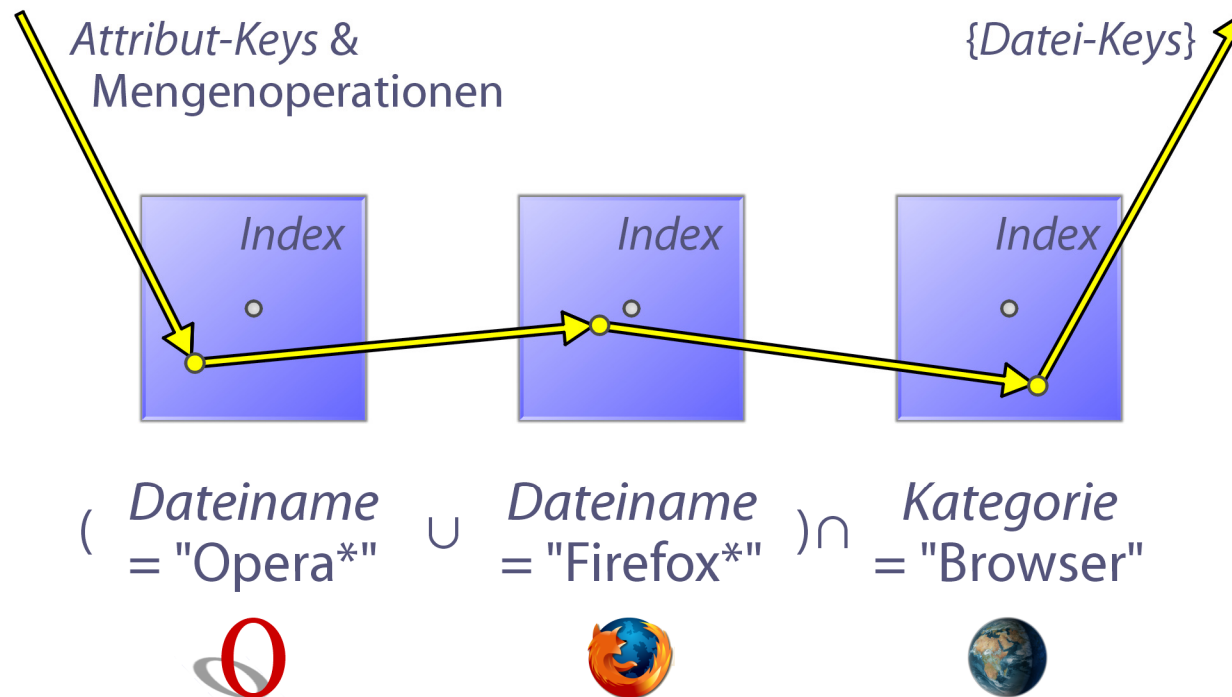
2 - Abfragen

- Schritt 1: Ermitteln des *Datentyps* der *Attribute* mit Hilfe des *Deskriptors*.
- Schritt 2: *Unterabfragen* mit Mengenoperationen verknüpfen.
- Schritt 3: *Datei-Keys* können nachgeschlagen werden.



2 - Unterabfragen (1)

- Jeder Knoten bearbeitet eine *Unterfrage* bzw. einen *Attribut-Key*.
- **Abarbeitung:** Weiterleitung von Knoten zu Knoten (CAN-Routing) und Verknüpfung der Ergebnisse (*Datei-Keys*) mit Mengenoperationen.



2 - Unterabfragen (2)

Mengenoperationen auf *Datei-Keys*:

- *Vereinigung* $A \cup B$
- *Schnitt* $A \cap B$
- *Differenz* $A \setminus B$
- *Komplement* $\overline{A} = C \setminus A$ (C Basismenge)

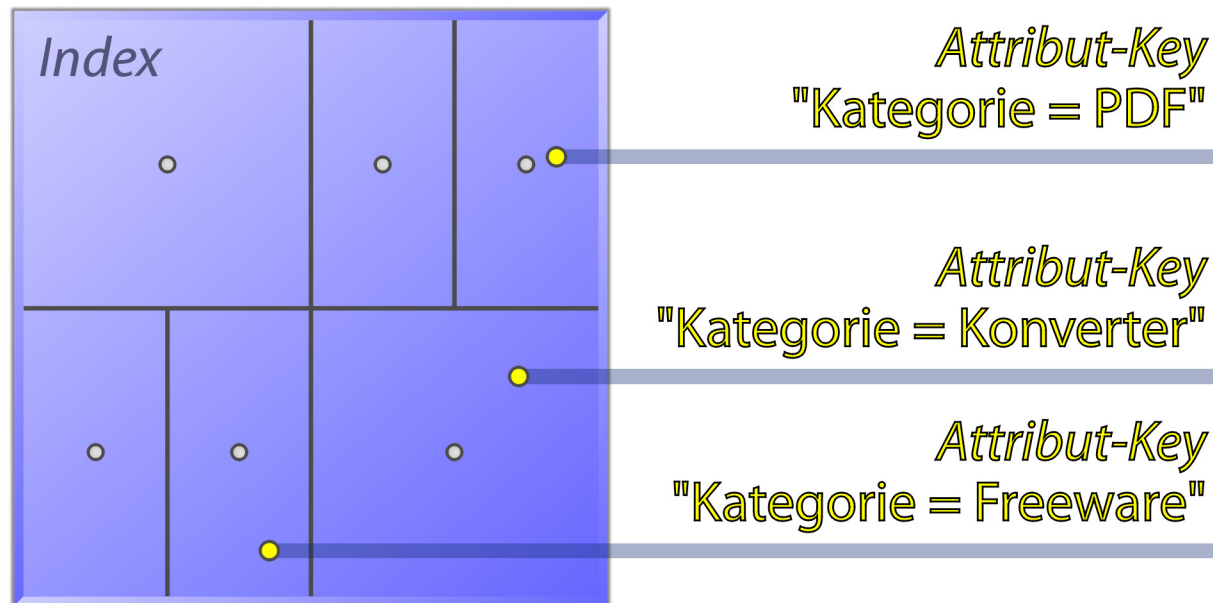
Anmerkungen:

- Statt dem *Komplement* kann nur eine *Komplement-Markierung* übertragen werden.
- *Streaming* ist möglich, wenn alle Mengen nach einem bestimmten Kriterium geordnet sind.

2 - Suche auf Wortmengen

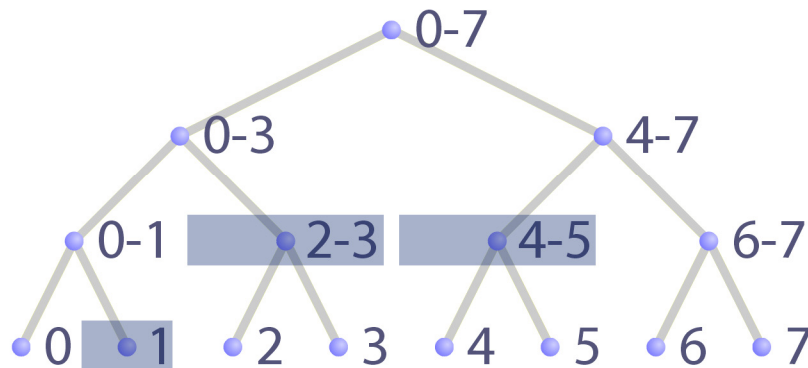
- Jeweils ein *Attribut-Key* für jedes Wort eines *Attributwertes*.
- Vorteile
 - Ermöglicht Suche nach einzelnen Wörtern.
 - Führt zu einer stärker verteilten Abarbeitung von *Unterabfragen*.

"Kategorie = PDF Konverter (Freeware)"



2 - Bereichssuche

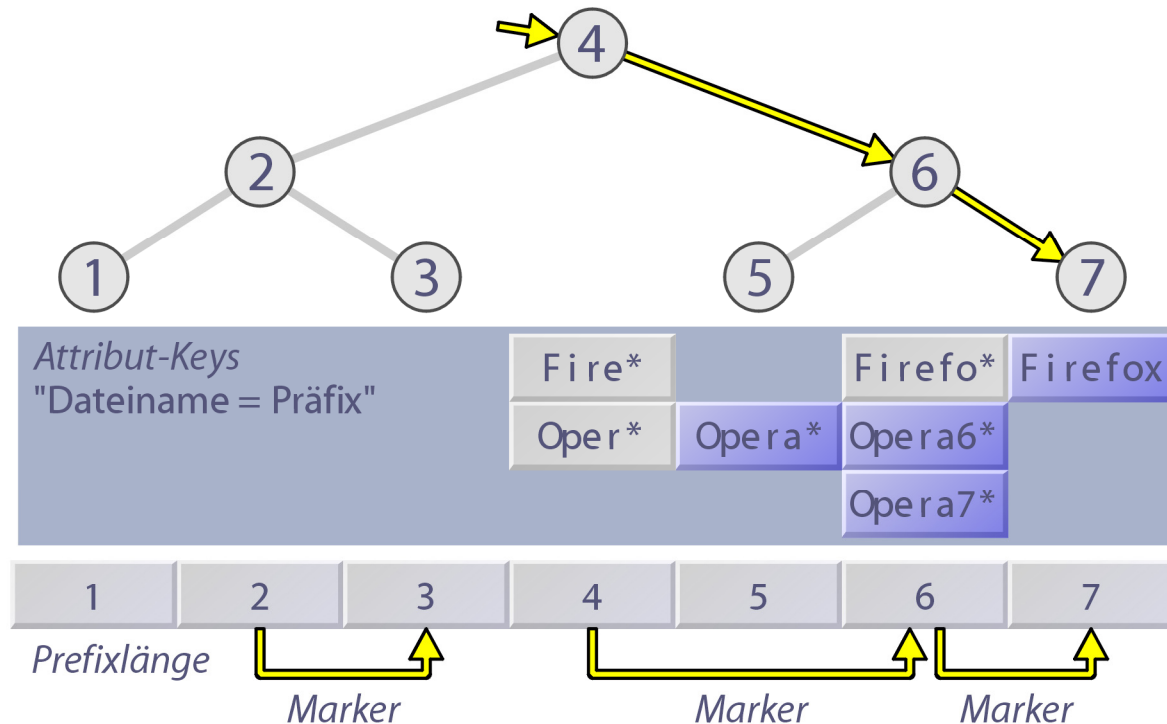
- Gegeben: Ein *Intervall* mit diskreten Werten (Z. B. [0; 1; 2; ...; 7]).
- Verwendung eines *RST* (*Range Search Tree*):
 - Suche der größten passenden *Teilintervalle* zum *Intervall* der Abfrage.
 - Beispiel: Die *Abfrage* 1-5 führt zu den *Unterabfragen* 1, 2-3 und 4-5.
- Der *Deskriptor* speichert den *Knotengrad* k und die *Tiefe* des Baumes.
- *Speicheraufwand* $O(\log_k w)$, *Suchaufwand* $O(\log_k w)$ (Z. B. $w = 8$).



Index	• 2-3 • •
• 4-7 •	• 0-3 •
• 0-1 •	• 6-7 •
• 4-5 •	• 0-7 •

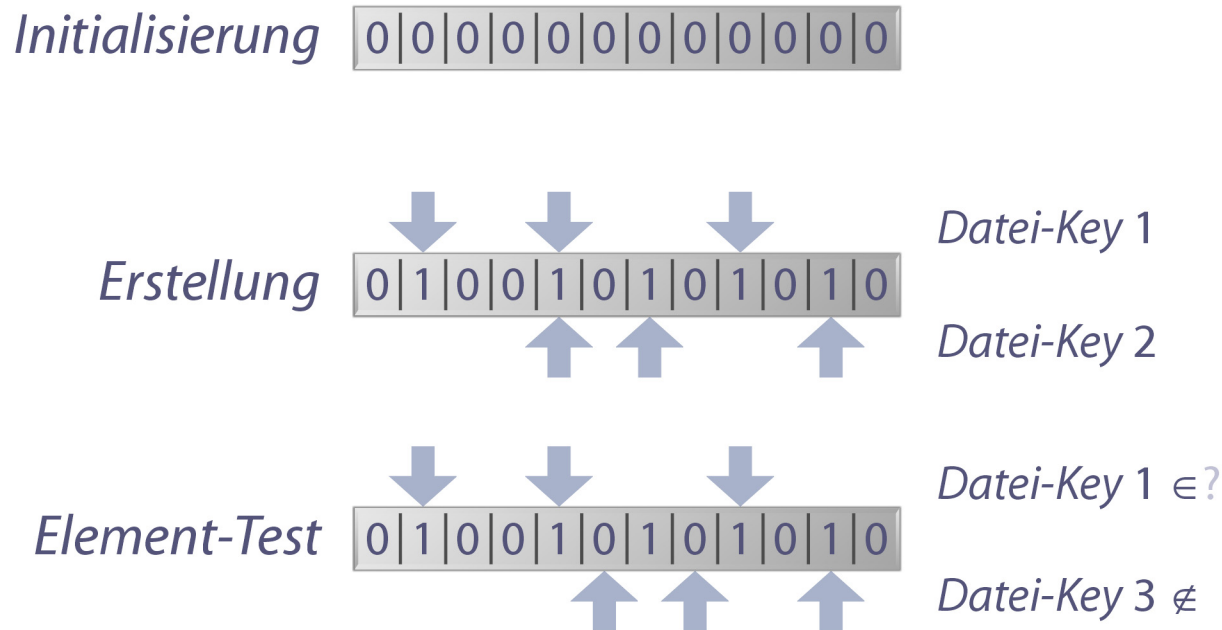
2 - Präfixsuche

- Jedes *Präfix* wird als *Attribut-Key* im *Index* gespeichert (Blau).
- *Längstes passendes Präfix* wird mit *binärer Suche* ermittelt.
- *Binäre Suche* benötigt Richtungsinformation, gelöst mit *Markern* (Grau).



2 - Bloom-Filter (1)

- Ein *Bloom-Filter* ist eine kompakte Darstellung einer Menge (m Bit).
- Ein Element der Menge wird durch mehrere 1-Bits repräsentiert.
- Hash-Funktionen bestimmen die Position dieser Bits.



2 - Bloom-Filter (2)

Nachteile

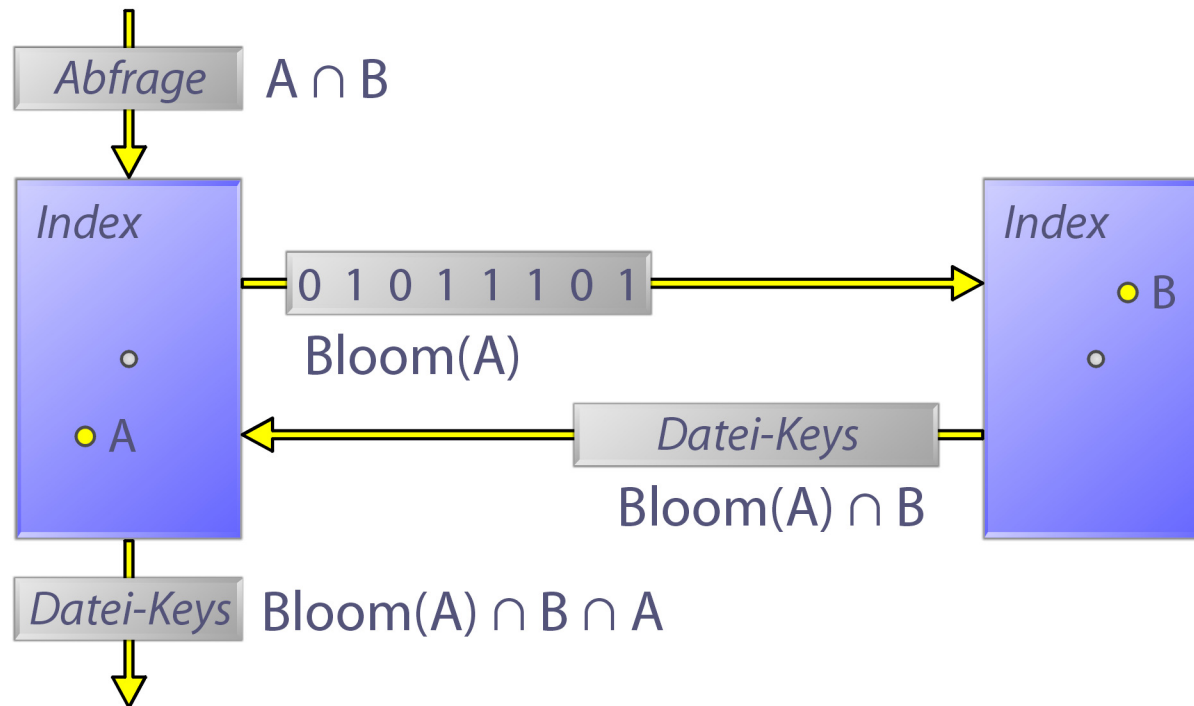
- Es kommen *falsche Elemente* hinzu (*False Positives*).
- Die Menge ist nicht mehr aufzählbar.

Operationen

- *Erstellung eines Bloom-Filters* aus einer Menge:
 $A \rightarrow \text{Bloom}(A)$
- *Schnitt zweier Bloom-Filter*:
 $\text{Bloom}(A \cap B) = \text{Bloom}(A) \cap \text{Bloom}(B) = \text{Bloom}(A) \text{ AND } \text{Bloom}(B)$
- *Vereinigung zweier Bloom-Filter*:
 $\text{Bloom}(A \cup B) = \text{Bloom}(A) \cup \text{Bloom}(B) = \text{Bloom}(A) \text{ OR } \text{Bloom}(B)$
- *Element-Test* führt zu *falschen Elementen*:
 $\text{Bloom}(A) = A \cup \varepsilon(A)$, hier $(A \cup \varepsilon(A)) \cap B$ mit B aufzählbar

2 - Schnitt

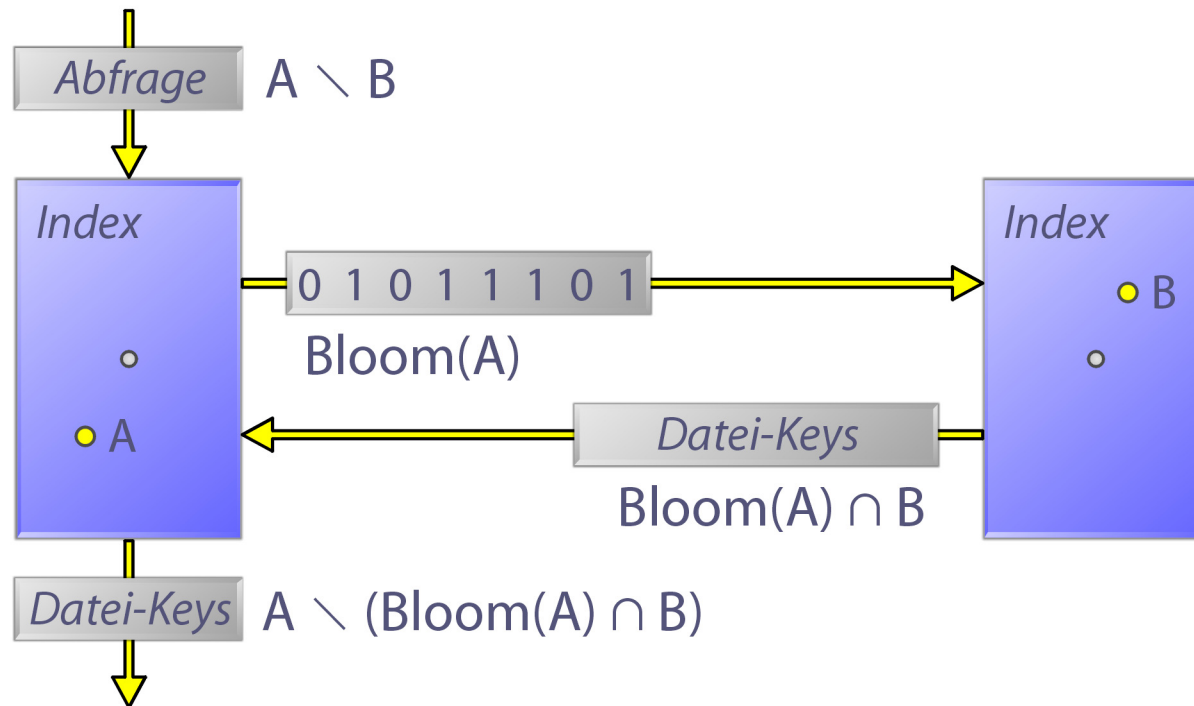
- A und B sind Mengen mit *Datei-Keys*.
- Nur $\text{Bloom}(A)$ und $\text{Bloom}(A) \cap B$ werden übertragen, nicht A und B.



- Verallgemeinerung auf $A \cap \dots \cap Z$ möglich.

2 - Differenz

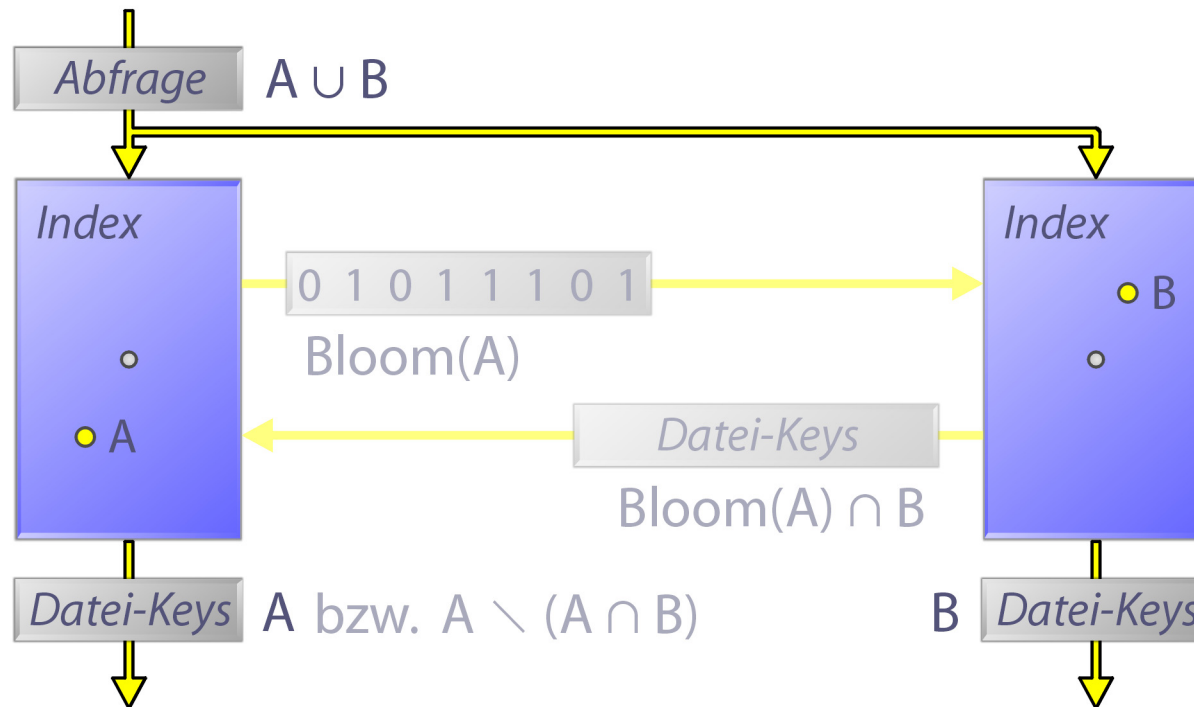
- Bloom-Filter ermöglichen kein Komplement oder Löschen.
- Aber $A \setminus B = A \setminus (\text{Bloom}(A) \cap B)$ anwendbar.



- Verallgemeinerung auf $(A \cap \dots \cap M) \setminus (N \cap \dots \cap Z)$ möglich.

2 - Vereinigung

- Jeder Knoten schickt *Datei-Keys* getrennt zum Empfänger.
- Gemeinsame Elemente von *A* und *B* werden doppelt übertragen.
- Berechnung des *Schnitts* kann Last für den Empfänger verringern.



- Beide Techniken ermöglichen (Mehr-)Bereichsabfragen.

Bereichsabfragen mit Hilbertfunktionen

- Erlaubt dynamische Anpassung von *Teilintervallen*.
- Regelmäßige Updates von *Attributwerten* werden unterstützt.

CANDy

- Ermöglicht drei verschiedene Arten der Suche: (1) *Suche auf Wortmengen*, (2) *Bereichssuche*, (3) *Präfixsuche*
- *Unterabfragen* können effizient mit Mengenoperationen verknüpft werden.