

# Nutzung von Indizes auf Sensornetzen

Seminar „Informationsverwaltung in Sensornetzen“

Marc Philipp



Universität Karlsruhe (TH)  
Forschungsuniversität • gegründet 1825

11. Dezember 2006

# Wozu Indexstrukturen auf Sensornetzen?

- Betrachte Sensornetz als verteilte Datenbank
- Benutze Sensorknoten als Datenspeicher
- Indizes sollen effiziente Bearbeitung von Anfragen ermöglichen
- Kosten für Kommunikation deutlich höher als Kosten für lokale Berechnungen
- Wichtigstes Entwurfsziel: Energieeffizienz

# Alternativen ohne Indizes

## Fluten von Anfragen

- Sensoren speichern nur ihre eigenen Daten
- Anfrage wird an alle Sensoren geschickt (*Flooding*)
- Problem: Hoher Energieverbrauch

## Globale Datenbank

- Sensoren schicken ihre Daten an einen zentralen Datenbankknoten
- Datenbank nutzt herkömmliche Technologie zur Verwaltung der Daten
- Problem: Sensorknoten, die nahe an dem Datenbankknoten liegen, werden Flaschenhals.

Also: Indexstrukturen sind notwendig und müssen im Sensornetz gehalten werden.

# Gliederung

- 1 Einleitung
- 2 Distributed Indices for Multi-dimensional Data (DIMs)
  - Überblick
  - Aufteilung in Zonen
  - Abbildung eines Ereignisses auf einen Ort
  - Bearbeitung von Anfragen
- 3 Fractionally Cascaded Information in a Sensor Network
  - Idee
  - Aufbau des QuadTrees
  - Verteilung der Daten auf die Knoten
  - Ablauf einer Anfrage
  - Verarbeitung von Updates
- 4 Fazit

# Gliederung

- 1 Einleitung
- 2 Distributed Indices for Multi-dimensional Data (DIMs)
  - Überblick
  - Aufteilung in Zonen
  - Abbildung eines Ereignisses auf einen Ort
  - Bearbeitung von Anfragen
- 3 Fractionally Cascaded Information in a Sensor Network
  - Idee
  - Aufbau des QuadTrees
  - Verteilung der Daten auf die Knoten
  - Ablauf einer Anfrage
  - Verarbeitung von Updates
- 4 Fazit

# Was sind DIMs?

- Übersetzung: „Verteilte Indizes für mehr-dimensionale Daten“
- Verteilte Datenstruktur:
  - ▶ Jeder Sensorknoten ist für einen gewissen Bereich der Daten zuständig, d. h. er speichert nur diese Daten
- Ziel:
  - ▶ Effiziente Bearbeitung von Bereichsanfragen auf mehr-dimensionalen Daten

# Bereichsanfragen auf mehr-dimensionalen Daten

- Wozu Bereichsanfragen?
  - ▶ Messwerte von Sensoren weisen Rauschen auf
- Warum „mehr-dimensional“?
  - ▶ Korrelation der Daten ausnutzen

## Beispiel

Sensoren messen die Temperatur und Luftfeuchtigkeit.

„Liste alle Messungen mit Temperaturen zwischen 20°C und 30°C und Luftfeuchtigkeit von über 50%.“

# Vorgehensweise

- Sensorfeld wird in Zonen aufgeteilt
- Daten werden einer Zone zugeordnet und dort gespeichert
- Benutzung eines geographischen Routingverfahrens wie *GPSR*<sup>1</sup>.
  - ▶ Adressat ist ein geographischer Ort, nicht ein spezieller Sensorknoten
  - ▶ Nachricht wird demjenigen Sensorknoten zugestellt, der dem Ziel am nächsten ist

## Annahmen

- Sensorknoten kennen ihre Position und die Netzwerkgrenzen
- Sensorknoten sind fest installiert, unbeweglich
- Alle Attributwerte sind normalisiert und liegen zwischen 0 und 1

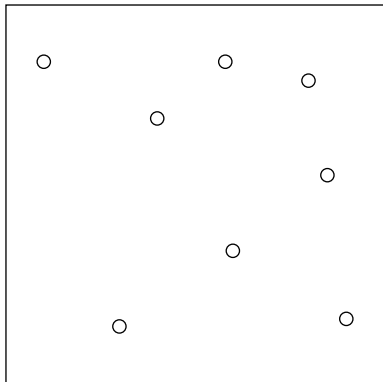
---

<sup>1</sup>Greedy Perimeter Stateless Routing

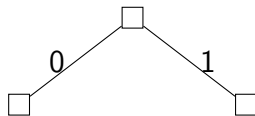
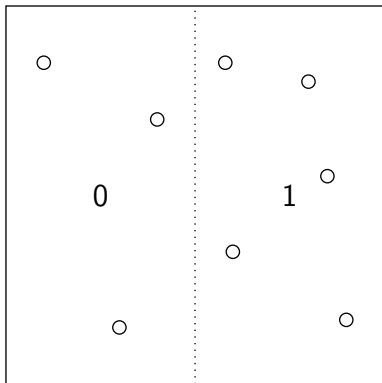
# Aufteilung in Zonen

- Rechteckiges Sensorfeld  $R$  wird abwechselnd vertikal und horizontal aufgeteilt
- Solange, bis in jeder Zone nur noch ein Sensor liegt
- Jeder Zone  $Z$  wird ein eindeutiger Code zugeordnet:
  - ▶ Falls  $Z$  in der linken Hälfte von  $R$ , ist das erste Bit 0, sonst 1
  - ▶ Falls  $Z$  in der unteren Hälfte von  $R$ , ist das zweite Bit 0, sonst 1
  - ▶ Prozedur wird nun rekursiv fortgesetzt
- Repräsentation durch virtuellen Binärbaum
  - ▶ Virtuuell, da der Baum nirgends gespeichert wird sondern implizit durch die Knoten gegeben ist

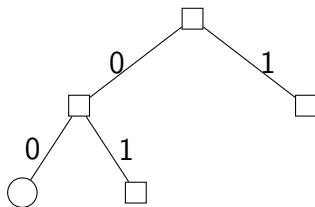
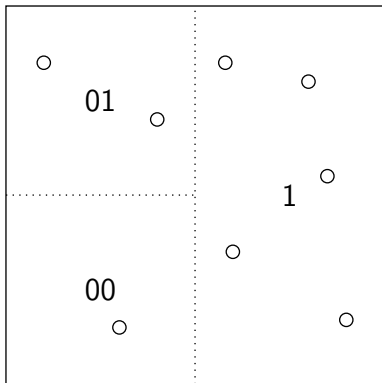
# Beispiel



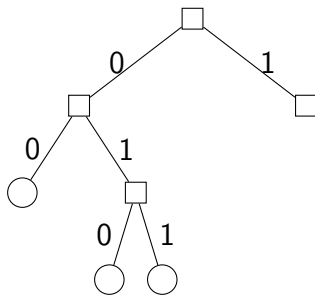
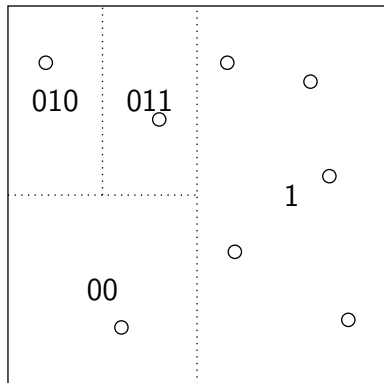
# Beispiel



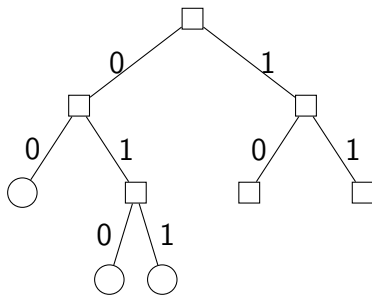
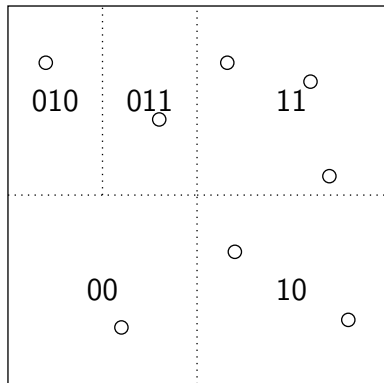
# Beispiel



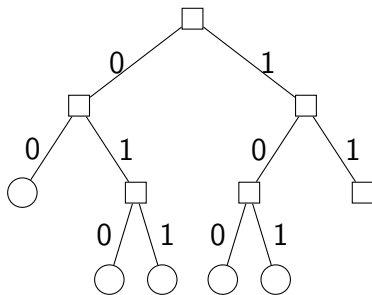
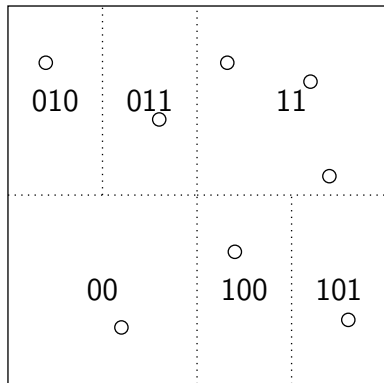
# Beispiel



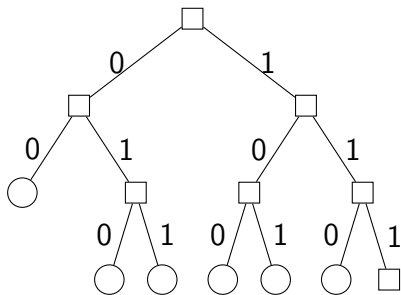
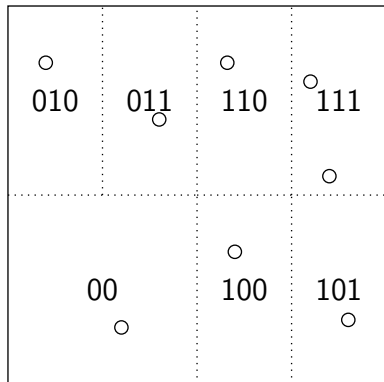
# Beispiel



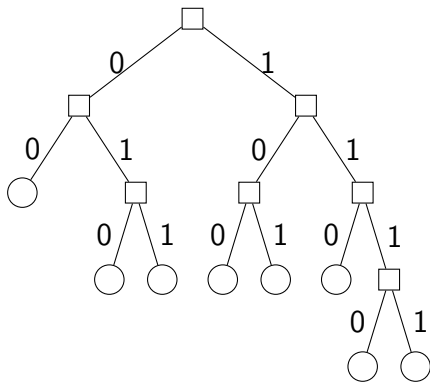
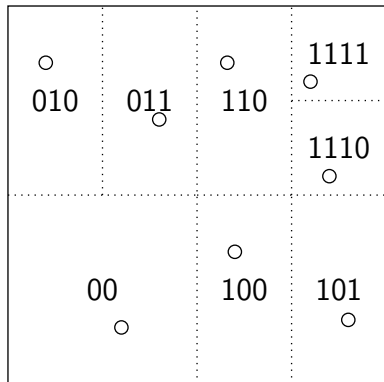
# Beispiel



# Beispiel



# Beispiel



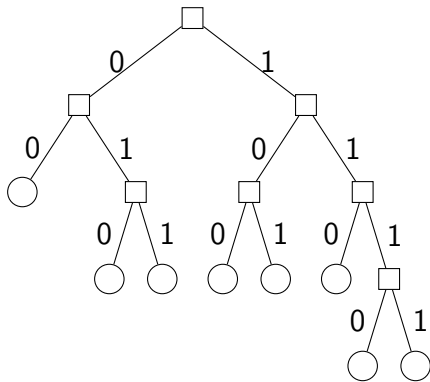
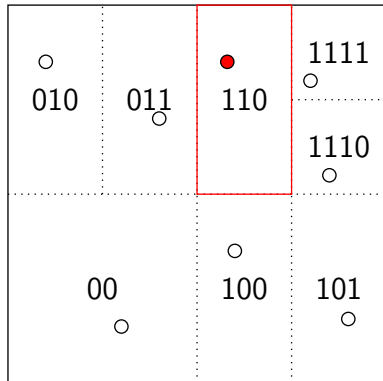
# Konkreter Ablauf

- Algorithmus läuft verteilt ab:
  - ▶ Jeder Sensor versucht seine eigene Zone zu bestimmen
  - ▶ Dabei kommuniziert er ausschließlich mit seinen direkten Nachbarn
- Probleme:
  - ▶ Zonen können leer bleiben  
    ↪ Backup-Zone
  - ▶ Zwei Knoten aus benachbarten Zonen können möglicherweise nicht direkt kommunizieren.  
    ↪ Zonengrenze wird als *unbestimmt* (*undecided*) markiert

# Backup-Zone

Jeder Zone wird eine Backup-Zone zugeteilt:

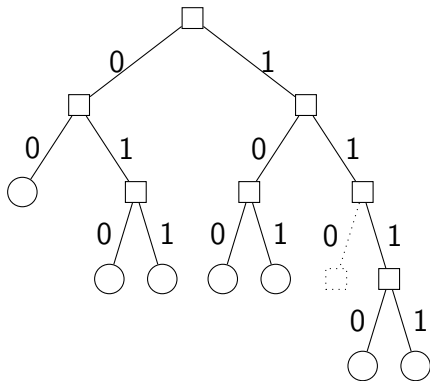
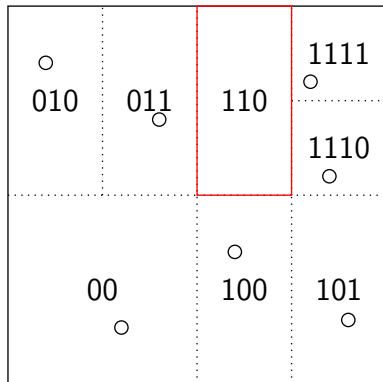
- Knoten, der sich am weitesten links (bzw. rechts) im Teilbaum des rechten (bzw. linken) Bruders befindet.



# Backup-Zone

Jeder Zone wird eine Backup-Zone zugeteilt:

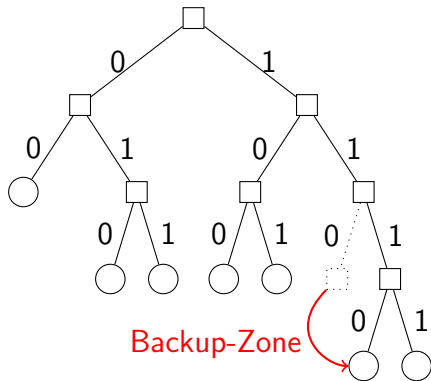
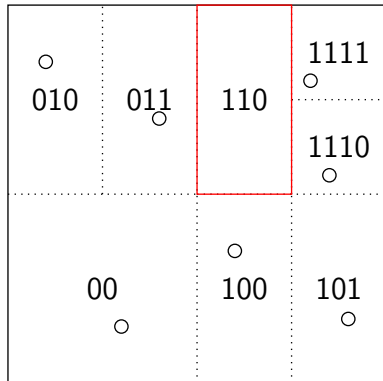
- Knoten, der sich am weitesten links (bzw. rechts) im Teilbaum des rechten (bzw. linken) Bruders befindet.



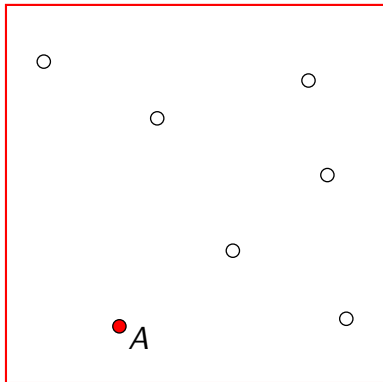
# Backup-Zone

Jeder Zone wird eine Backup-Zone zugeteilt:

- Knoten, der sich am weitesten links (bzw. rechts) im Teilbaum des rechten (bzw. linken) Bruders befindet.



# Unbestimmte Zonengrenzen

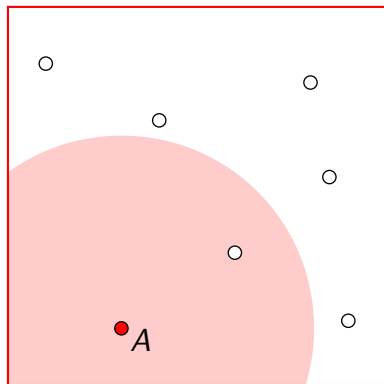


Jeder Knoten versucht seine Zonengrenzen zu bestimmen:

- Überlappung mit Netzwerkgrenze oder Kommunikation mit Nachbarn  
~> **Bestimmte Grenze**
- Keine Kommunikation möglich  
~> **Unbestimmte Grenze**

Auflösung unbestimmter Grenzen erfolgt erst bei Anfragebearbeitung oder dem Einfügen eines Ereignisses.

# Unbestimmte Zonengrenzen

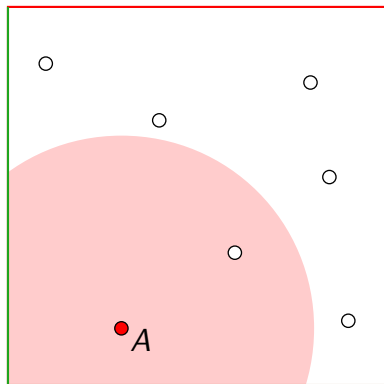


Jeder Knoten versucht seine Zonengrenzen zu bestimmen:

- Überlappung mit Netzwerkgrenze oder Kommunikation mit Nachbarn  
~> **Bestimmte Grenze**
- Keine Kommunikation möglich  
~> **Unbestimmte Grenze**

Auflösung unbestimmter Grenzen erfolgt erst bei Anfragebearbeitung oder dem Einfügen eines Ereignisses.

# Unbestimmte Zonengrenzen

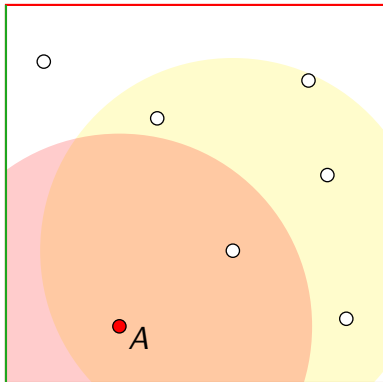


Jeder Knoten versucht seine Zonengrenzen zu bestimmen:

- Überlappung mit Netzwerkgrenze oder Kommunikation mit Nachbarn  
~> Bestimmte Grenze
- Keine Kommunikation möglich  
~> Unbestimmte Grenze

Auflösung unbestimmter Grenzen erfolgt erst bei Anfragebearbeitung oder dem Einfügen eines Ereignisses.

# Unbestimmte Zonengrenzen

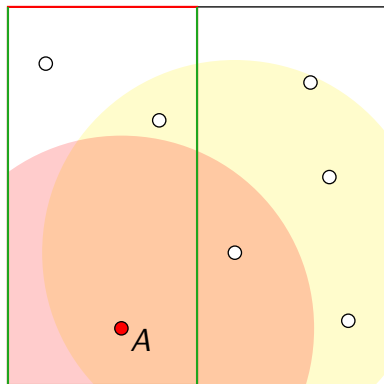


Jeder Knoten versucht seine Zonengrenzen zu bestimmen:

- Überlappung mit Netzwerkgrenze oder Kommunikation mit Nachbarn  
~> **Bestimmte Grenze**
- Keine Kommunikation möglich  
~> **Unbestimmte Grenze**

Auflösung unbestimmter Grenzen erfolgt erst bei Anfragebearbeitung oder dem Einfügen eines Ereignisses.

# Unbestimmte Zonengrenzen

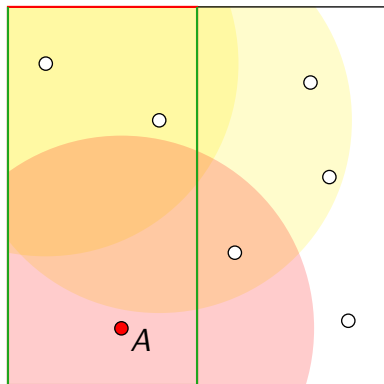


Jeder Knoten versucht seine Zonengrenzen zu bestimmen:

- Überlappung mit Netzwerkgrenze oder Kommunikation mit Nachbarn  
~> **Bestimmte Grenze**
- Keine Kommunikation möglich  
~> **Unbestimmte Grenze**

Auflösung unbestimmter Grenzen erfolgt erst bei Anfragebearbeitung oder dem Einfügen eines Ereignisses.

# Unbestimmte Zonengrenzen

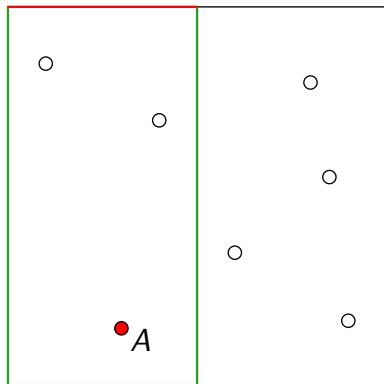


Jeder Knoten versucht seine Zonengrenzen zu bestimmen:

- Überlappung mit Netzwerkgrenze oder Kommunikation mit Nachbarn  
↪ Bestimmte Grenze
- Keine Kommunikation möglich  
↪ Unbestimmte Grenze

Auflösung unbestimmter Grenzen erfolgt erst bei Anfragebearbeitung oder dem Einfügen eines Ereignisses.

# Unbestimmte Zonengrenzen



Jeder Knoten versucht seine Zonengrenzen zu bestimmen:

- Überlappung mit Netzwerkgrenze oder Kommunikation mit Nachbarn  
~> **Bestimmte Grenze**
- Keine Kommunikation möglich  
~> **Unbestimmte Grenze**

Auflösung unbestimmter Grenzen erfolgt erst bei Anfragebearbeitung oder dem Einfügen eines Ereignisses.

# Hashfunktion

- Wie wird ein Datum  $\langle A_1, \dots, A_m \rangle$  einem Ort zugeordnet?
- Wertebereich der Attribute  $[0, 1]$  wird in gleichgroße Intervalle unterteilt, die jeweils mit 0 bzw. 1 kodiert werden.
- Ein Knoten kodiert solange, bis der Code so lang ist, wie sein eigener Zonencode.

## Beispiel

Knoten  $A$  mit Zonencode  $Z_A$  der Länge 5 kodiert  $D = \langle 0,3; 0,8 \rangle$ .

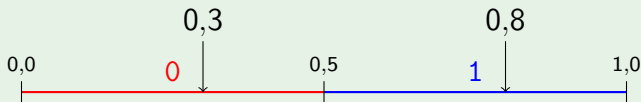


# Hashfunktion

- Wie wird ein Datum  $\langle A_1, \dots, A_m \rangle$  einem Ort zugeordnet?
- Wertebereich der Attribute  $[0, 1]$  wird in gleichgroße Intervalle unterteilt, die jeweils mit 0 bzw. 1 kodiert werden.
- Ein Knoten kodiert solange, bis der Code so lang ist, wie sein eigener Zonencode.

## Beispiel

Knoten  $A$  mit Zonencode  $Z_A$  der Länge 5 kodiert  $D = \langle 0,3; 0,8 \rangle$ .



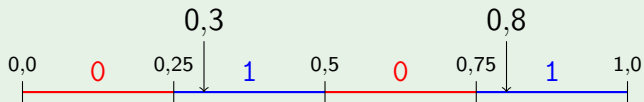
Ergebnis:  $\text{code}_A(D) = 01$

# Hashfunktion

- Wie wird ein Datum  $\langle A_1, \dots, A_m \rangle$  einem Ort zugeordnet?
- Wertebereich der Attribute  $[0, 1]$  wird in gleichgroße Intervalle unterteilt, die jeweils mit 0 bzw. 1 kodiert werden.
- Ein Knoten kodiert solange, bis der Code so lang ist, wie sein eigener Zonencode.

## Beispiel

Knoten  $A$  mit Zonencode  $Z_A$  der Länge 5 kodiert  $D = \langle 0,3; 0,8 \rangle$ .



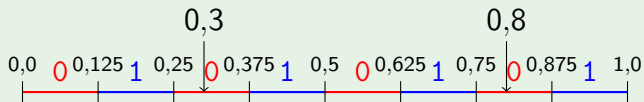
Ergebnis:  $\text{code}_A(D) = 0111$

# Hashfunktion

- Wie wird ein Datum  $\langle A_1, \dots, A_m \rangle$  einem Ort zugeordnet?
- Wertebereich der Attribute  $[0, 1]$  wird in gleichgroße Intervalle unterteilt, die jeweils mit 0 bzw. 1 kodiert werden.
- Ein Knoten kodiert solange, bis der Code so lang ist, wie sein eigener Zonencode.

## Beispiel

Knoten  $A$  mit Zonencode  $Z_A$  der Länge 5 kodiert  $D = \langle 0,3; 0,8 \rangle$ .



Ergebnis:  $\text{code}_A(D) = 01110$

# Routing zum Besitzer (1)

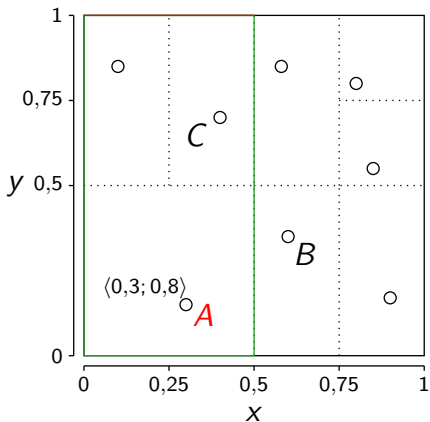
## Ziel:

- Sensorknoten  $A$  misst ein Datum  $D$
  - Dieses soll nun an den Besitzer zugestellt werden.
- 
- Initialisierung:
    - ▶  $A$  berechnet  $\text{code}_A(D)$  und vergleicht den Wert mit seinem Zonencode  $Z_A$
    - ▶ Falls identisch:  $A$  ist Besitzer und speichert Datum
    - ▶ Ansonsten:  $A$  markiert sich als initialer Besitzer des Datums ( $\text{owner}(D)$ ) und sendet eine Nachricht an den Mittelpunkt der Zone  $Z' = \text{code}(D)$ .
    - ▶ Nachricht besteht aus:  $\langle D, \text{code}(D), \text{owner}(D) \rangle$

## Routing zum Besitzer (2)

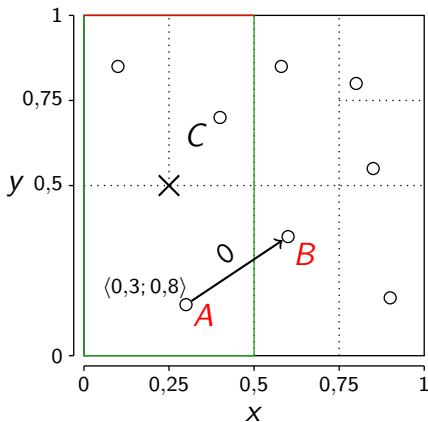
- Für jeden Knoten  $B$  auf dem Weg von  $A$  zum Besitzer von  $D$ :
  - ▶  $B$  berechnet  $\text{code}_B(D)$ . Falls der neue Code länger ist, aktualisiert  $B$  den in der Nachricht enthaltenen Code  $\text{code}(D)$ . Der Adressat wird so also schrittweise verfeinert.
  - ▶  $B$  vergleicht nun  $Z_B$  mit  $\text{owner}(D)$ . Falls  $Z_B$  einen längeren gemeinsamen Präfix mit  $\text{code}(D)$  hat als  $\text{owner}(D)$ , setzt  $B$  sich selbst als neuer Besitzer von  $D$ .
  - ▶ Falls der Zonencode  $Z_B$  exakt mit  $\text{code}(D)$  übereinstimmt, speichert  $B$  das Datum, ansonsten sendet  $B$  eine Nachricht an den Mittelpunkt der Zone  $Z'' = \text{code}(D)$ .
- GPSR garantiert, dass die Nachricht an  $B$  zurückgeschickt wird, sollte sich kein Knoten finden, der besser geeignet ist.

# Beispiel



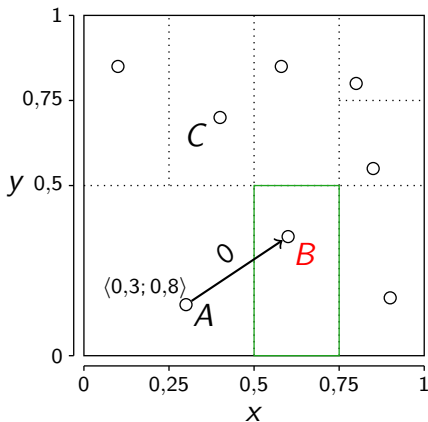
- 1 A erhält Daten  $\langle 0,3; 0,8 \rangle$  und kodiert 0

# Beispiel



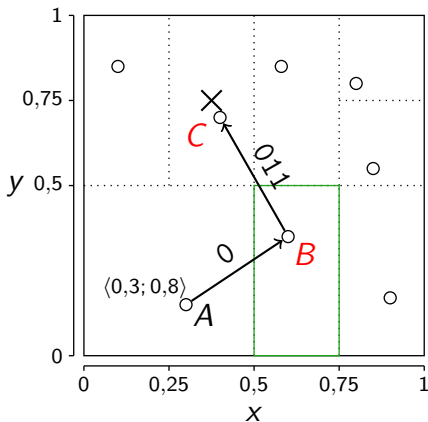
- 1 A erhält Daten  $\langle 0,3; 0,8 \rangle$  und kodiert 0
- 2 Da eine Grenze von A *unbestimmt*, markiert sich A im Paket als vorläufiger Besitzer, leitet es aber an B weiter.

# Beispiel



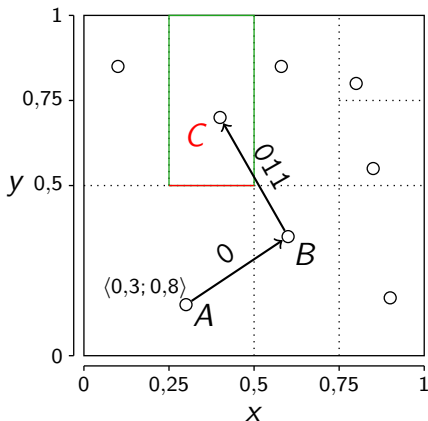
- 1 A erhält Daten  $\langle 0,3; 0,8 \rangle$  und kodiert 0
- 2 Da eine Grenze von A *unbestimmt*, markiert sich A im Paket als vorläufiger Besitzer, leitet es aber an B weiter.

# Beispiel



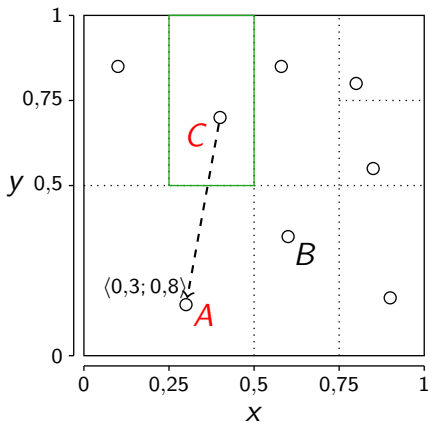
- 1 A erhält Daten  $\langle 0,3; 0,8 \rangle$  und kodiert 0
- 2 Da eine Grenze von A *unbestimmt*, markiert sich A im Paket als vorläufiger Besitzer, leitet es aber an B weiter.
- 3 B berechnet 011 und leitet das Paket daher an C weiter.

# Beispiel



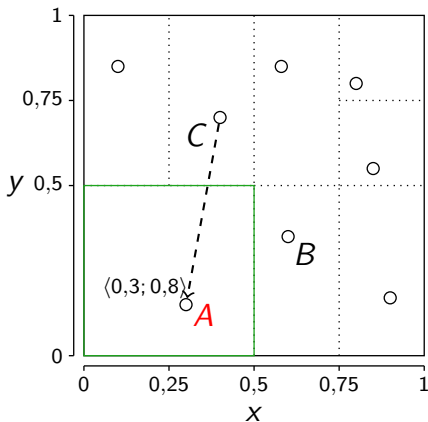
- 1 A erhält Daten  $\langle 0,3; 0,8 \rangle$  und kodiert 0
- 2 Da eine Grenze von A *unbestimmt*, markiert sich A im Paket als vorläufiger Besitzer, leitet es aber an B weiter.
- 3 B berechnet 011 und leitet das Paket daher an C weiter.
- 4 C erkennt sich selbst als Besitzer und speichert die Daten.

# Beispiel



- 1 A erhält Daten  $\langle 0,3; 0,8 \rangle$  und kodiert 0
- 2 Da eine Grenze von A *unbestimmt*, markiert sich A im Paket als vorläufiger Besitzer, leitet es aber an B weiter.
- 3 B berechnet 011 und leitet das Paket daher an C weiter.
- 4 C erkennt sich selbst als Besitzer und speichert die Daten.
- 5 C sendet eine Verkleinerungsnachricht (*shrink message*) an A.

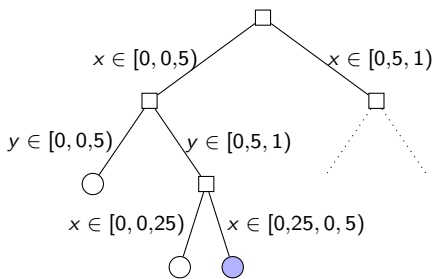
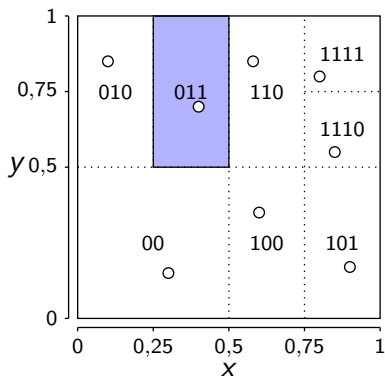
# Beispiel



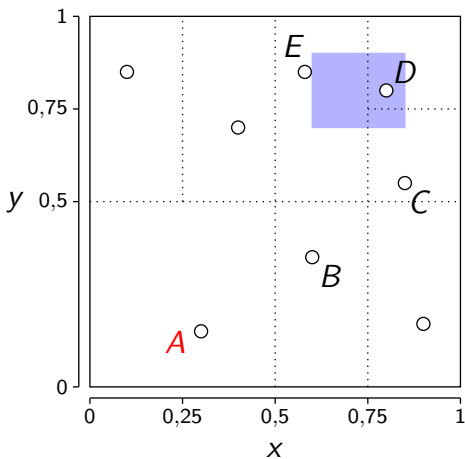
- 1 A erhält Daten  $\langle 0,3; 0,8 \rangle$  und kodiert 0
- 2 Da eine Grenze von A *unbestimmt*, markiert sich A im Paket als vorläufiger Besitzer, leitet es aber an B weiter.
- 3 B berechnet 011 und leitet das Paket daher an C weiter.
- 4 C erkennt sich selbst als Besitzer und speichert die Daten.
- 5 C sendet eine Verkleinerungsnachricht (*shrink message*) an A.
- 6 A verkleinert sein Zone zu 00.

# Bearbeitung von Bereichsanfragen

- Jeder Sensorknoten kann eine Anfrage absetzen
- Prinzip der Verarbeitung:
  - ▶ Routing zu einer Zone, die den gesamten Bereich abdeckt
  - ▶ Aufspalten der Anfrage in Teilanfragen

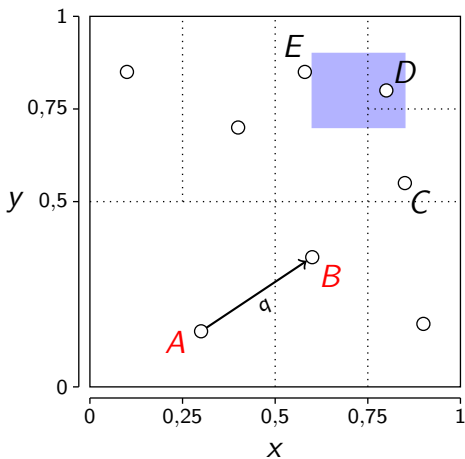


# Beispiel



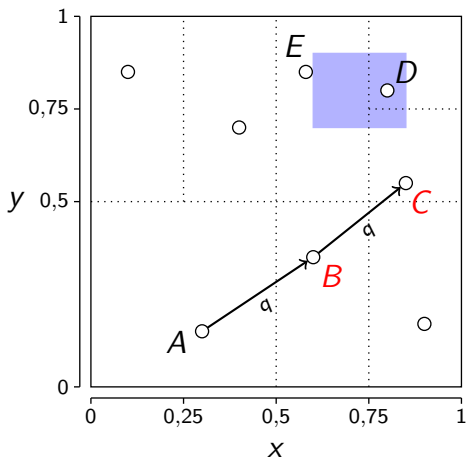
- 1 A stellt Anfrage  
 $q = \langle 0,6 - 0,85; 0,7 - 0,9 \rangle$

# Beispiel



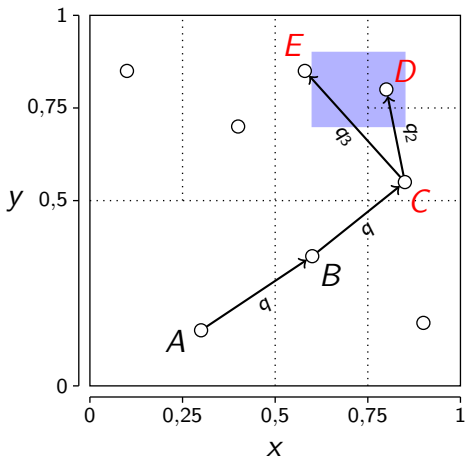
- 1 A stellt Anfrage  
 $q = \langle 0,6 - 0,85; 0,7 - 0,9 \rangle$
- 2 Aus Sicht von A ist Zone 1  
 zuständig  $\rightsquigarrow B$ .

# Beispiel



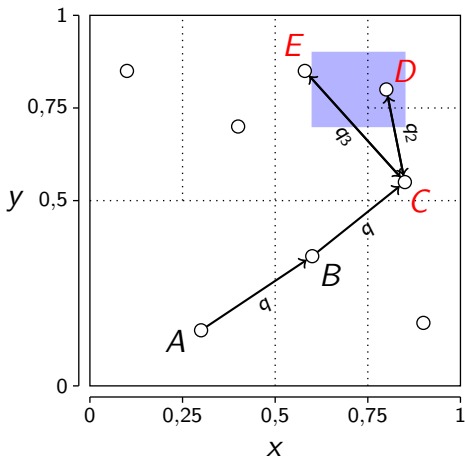
- 1 A stellt Anfrage  
 $q = \langle 0,6 - 0,85; 0,7 - 0,9 \rangle$
- 2 Aus Sicht von A ist Zone 1 zuständig  $\rightsquigarrow B$ .
- 3 B leitet  $q$  an C weiter

# Beispiel



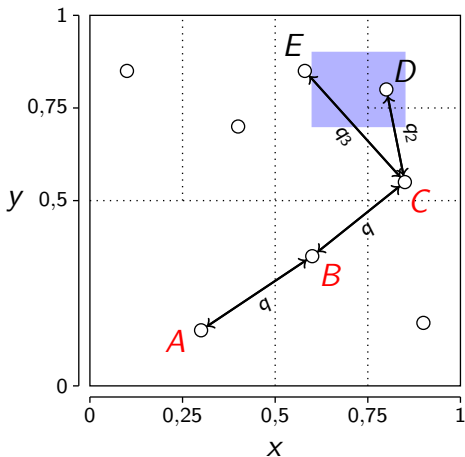
- 1 A stellt Anfrage  
 $q = \langle 0,6 - 0,85; 0,7 - 0,9 \rangle$
- 2 Aus Sicht von A ist Zone 1 zuständig  $\rightsquigarrow$  B.
- 3 B leitet  $q$  an C weiter
- 4 C teilt  $q$  in Teilanfragen auf:  
 $C: q_1 = \langle 0,75 - 0,85; 0,7 - 0,75 \rangle$   
 $D: q_2 = \langle 0,75 - 0,85; 0,75 - 0,9 \rangle$   
 $E: q_3 = \langle 0,6 - 0,75; 0,7 - 0,9 \rangle$

# Beispiel



- 1 A stellt Anfrage  
 $q = \langle 0,6 - 0,85; 0,7 - 0,9 \rangle$
- 2 Aus Sicht von A ist Zone 1 zuständig  $\rightsquigarrow$  B.
- 3 B leitet  $q$  an C weiter
- 4 C teilt  $q$  in Teilanfragen auf:  
 C:  $q_1 = \langle 0,75 - 0,85; 0,7 - 0,75 \rangle$   
 D:  $q_2 = \langle 0,75 - 0,85; 0,75 - 0,9 \rangle$   
 E:  $q_3 = \langle 0,6 - 0,75; 0,7 - 0,9 \rangle$
- 5 D und E senden ihre Teilergebnisse an C zurück.

# Beispiel



- 1 A stellt Anfrage  
 $q = \langle 0,6 - 0,85; 0,7 - 0,9 \rangle$
- 2 Aus Sicht von A ist Zone 1 zuständig  $\rightsquigarrow$  B.
- 3 B leitet q an C weiter
- 4 C teilt q in Teilanfragen auf:  
 C:  $q_1 = \langle 0,75 - 0,85; 0,7 - 0,75 \rangle$   
 D:  $q_2 = \langle 0,75 - 0,85; 0,75 - 0,9 \rangle$   
 E:  $q_3 = \langle 0,6 - 0,75; 0,7 - 0,9 \rangle$
- 5 D und E senden ihre Teilergebnisse an C zurück.
- 6 C setzt das Ergebnis aus den Teilergebnissen zusammen und leitet es über B an A zurück.

# Zusammenfassung

## Prinzip:

- Benutze Hashfunktion zur Indizierung der Daten
  - ▶ Gewährleistet Lokalität bzgl. der Daten, d. h. Daten mit ähnlichen Attributwerten werden nahe beieinander gespeichert
- Basiert auf geographischem Routingverfahren (GPSR)

## Probleme:

- Funktioniert nur für gleichverteilte Daten
  - ▶ Sonst werden bestimmte Sensoren zu Hotspots
- Räumliche Lokalität der Daten geht verloren
  - ▶ Anfragen, die sich nur auf gewisse Orte beziehen, werden nicht unterstützt

# Gliederung

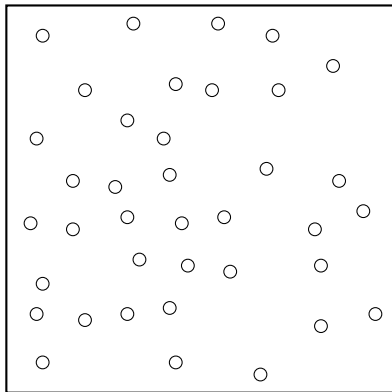
- 1 Einleitung
- 2 Distributed Indices for Multi-dimensional Data (DIMs)
  - Überblick
  - Aufteilung in Zonen
  - Abbildung eines Ereignisses auf einen Ort
  - Bearbeitung von Anfragen
- 3 Fractionally Cascaded Information in a Sensor Network
  - Idee
  - Aufbau des QuadTrees
  - Verteilung der Daten auf die Knoten
  - Ablauf einer Anfrage
  - Verarbeitung von Updates
- 4 Fazit

# Fractionally Cascaded Information

- Versuch einer Übersetzung: „Stufenförmig verteilte Informationen“
- Prinzip:
  - ▶ Jeder Sensor kennt einen Teil der Daten entfernter Sensoren
  - ▶ Räumliche Lokalität: Mehr Informationen von nahen Sensoren als von weiter entfernten
- Ziel: Effiziente Bearbeitung von Bereichsanfragen

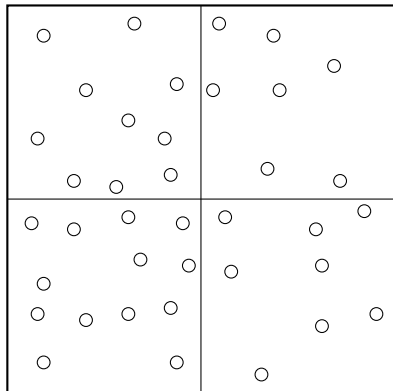
# QuadTrees

- Zugrundeliegende Datenstruktur
- Viertelung der Quadrate bis in einem Quadrat maximal ein Sensor liegt
- Sensoren sind Blätter des Baumes
- Wie bei DIMs: Datenstruktur nur virtuell vorhanden



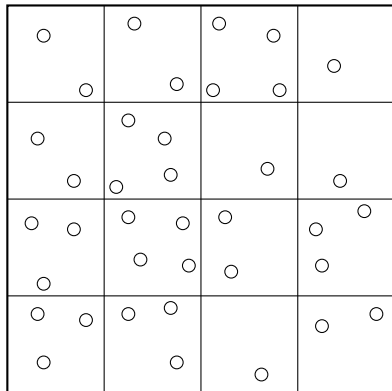
# QuadTrees

- Zugrundeliegende Datenstruktur
- Viertelung der Quadrate bis in einem Quadrat maximal ein Sensor liegt
- Sensoren sind Blätter des Baumes
- Wie bei DIMs: Datenstruktur nur virtuell vorhanden



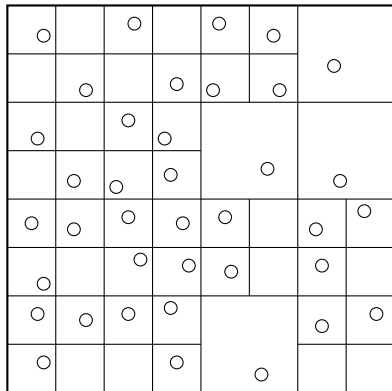
# QuadTrees

- Zugrundeliegende Datenstruktur
- Viertelung der Quadrate bis in einem Quadrat maximal ein Sensor liegt
- Sensoren sind Blätter des Baumes
- Wie bei DIMs: Datenstruktur nur virtuell vorhanden



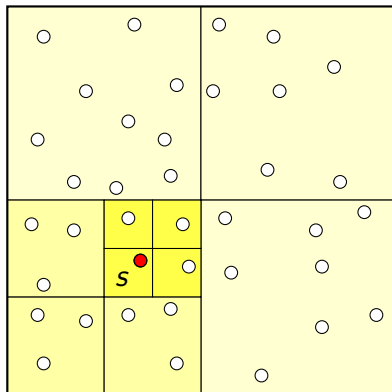
# QuadTrees

- Zugrundeliegende Datenstruktur
- Viertelung der Quadrate bis in einem Quadrat maximal ein Sensor liegt
- Sensoren sind Blätter des Baumes
- Wie bei DIMs: Datenstruktur nur virtuell vorhanden



# Allgemeiner Aufbau

- Der Wert eines Knotens im QuadTree wird in allen Sensoren in seinem Bereich gespeichert.
- Jeder Sensor  $s$  speichert aggregierte Daten aller Geschwister eines Knoten auf dem Pfad von  $s$  zur Wurzel.
- Impliziert Aufteilung in Kacheln



# Beispiel

- Sensoren messen Temperatur
- Jeder Sensor  $s$  speichert die minimale ( $\min_t(u)$ ) und maximale Temperatur  $\max_t(u)$  der jeweiligen Kacheln
- Eine Anfrage besteht aus drei Komponenten  $(q, R, T)$ :
  - $q$ : Sensor, der Anfrage absetzt
  - $R$ : rechteckiger Zielbereich
  - $T$ : Temperaturbereich  $[T_1, T_2]$
- Typische Anfragen:
  - ▶ Wieviele Sensoren mit einer Temperatur zwischen  $20^\circ\text{C}$  und  $30^\circ\text{C}$  befinden sich im Bereich  $R$ ?  
 $\rightsquigarrow (q, R, [20, 30])$
  - ▶ Liste alle heißen Sensoren im Bereich  $R$ .  
 $\rightsquigarrow (q, R, [T, \infty))$

# Anfragebearbeitung

## Definition

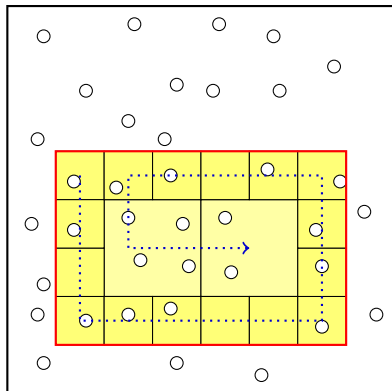
Ein Knoten  $u$  des QuadTrees heißt *kanonisch*, falls der Bereich von  $u$  in  $R$  enthalten ist, der Bereich des Vaterknotens  $p(u)$  jedoch nicht.

## Ablauf

- 1 Sende Nachricht von der Quelle zur angefragten Region
- 2 Besuche kanonische Knoten  $u$   
(d. h. einen beliebigen Sensor im Bereich von  $u$ )
- 3 Für jeden kanonischen Knoten:  
Laufe Unterbaum ab und melde passende Sensoren

# Beispiel (Fortsetzung)

- Gesucht: Heiße Sensoren im Bereich  $R$
- Formal:  $(q, R, [T, \infty))$
- Besuche für jeden kanonischen Knoten  $u$  einen Sensor  $s$
- Falls  $\max_t(u) < T$ , trägt der Knoten nichts zur Lösung bei
- Sonst: Besuche rekursiv die Kinder von  $u$



# Verarbeitung von Updates

- Wenn sich der Wert eines Sensors ändert, muss dies im QuadTree nach oben propagiert werden.
- Falls die Knoten z.B. das Maximum speichern, kann die Aktualisierung stoppen, sobald sich der Wert nicht mehr ändert.
- Im schlimmsten Fall setzt sich die Aktualisierung jedoch bis zur Wurzel fort.  
⇒ Alle Sensoren müssen benachrichtigt werden

# Zusammenfassung

## Prinzip:

- Information ist stufenförmig verteilt
- Räumliche Lokalität gewahrt: Sensoren wissen mehr über nahe Knoten als über weiter entfernte

## Probleme:

- Echte Bereichsanfragen (z. B. Temperaturbereich  $[T_1, T_2]$ ) werden durch Speicherung der Minima/Maxima in den Knoten nur bedingt beschleunigt
- Update eines Sensors kann Benachrichtigung aller Sensoren nach sich ziehen


# Gliederung

- 1 Einleitung
- 2 Distributed Indices for Multi-dimensional Data (DIMs)
  - Überblick
  - Aufteilung in Zonen
  - Abbildung eines Ereignisses auf einen Ort
  - Bearbeitung von Anfragen
- 3 Fractionally Cascaded Information in a Sensor Network
  - Idee
  - Aufbau des QuadTrees
  - Verteilung der Daten auf die Knoten
  - Ablauf einer Anfrage
  - Verarbeitung von Updates
- 4 Fazit

# Fazit

- Zwei Ansätze für Indizes in Sensornetzen wurden vorgestellt
- Es existieren zahlreiche (!) weitere Lösungen
- Verfahren unterscheiden sich häufig bezüglich des genauen Problems, das gelöst werden soll
- Es existiert keine Patentlösung, die auf alle Szenarien anwendbar ist
- Geeignetes Verfahren abhängig von der Einsatzumgebung

# Literatur

-  J. Gao, L. J. Guibas, J. Hershberger und L. Zhang  
Fractionally cascaded information in a sensor network  
Proceedings of the third international symposium on Information processing in sensor networks (ISPN 2004), S. 311–319, 2004.
-  B. Karp und H. T. Kung  
GPSR: greedy perimeter stateless routing for wireless networks  
Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000), S. 243–254, 2000.
-  X. Li, Y.J. Kim, R. Govindan und W. Hong  
Multi-dimensional range queries in sensor networks  
Proceedings of the first ACM conference on embedded networked sensor systems (SenSys 2003), S. 63–75, 2003.