Vorstellung des Streamkonzepts

Datenbanken und Informationssysteme

Technische Universität Kaiserslautern
Lehrgebiet Datenverwaltungssysteme
Sommersemester 2005

Benjamin Mock

Inhalt

- Anwendungsgebiete von Datenströmen
- Eigenschaften von Datenströmen
- Anfragen an Datenströme
- Sliding Windows
- Datenstrom-Management-Systeme (DSMS)
  - STREAM, AURORA, SPEX
- Load Shedding

Anwendungsgebiete

- Nachrichtenüberwachung
  - Presse
  - Börse
  - Meteorologie
- Multimediaströme
- Netzwerkverwaltung
- Verkehrsüberwachung

Datenströme

- potentiell unendlich
- Ankunftsraten
  - unvorhersagbar
  - unbeeinflussbar
- benötigen Echtzeit-Verarbeitung
- Unterscheidung zwischen
  - Punkt- / Tupelströme
  - XML-Ströme

Speicherung nur eingeschränkt möglich!

Datenströme II

- push-Kommunikation
- Daten durch mehrere externe Quellen (Sensoren, …)
- Ergebnisse ebenfalls als Datenstrom
- Information des Benutzers
  - zu bestimmten Zeitpunkten
  - bei anormalen Werten
  - bei jedem Tupel
- DBMS-Active, Human-Passive = DAHP

DSMS		DBMS
transient	**Daten**	persistent
persistent	**Anfragen**	transient
nur hinzufügen	**Änderungen**	beliebig
Anfragen	**Indizierung**	Daten
Evtl. angenähert	**Ergebnisse**	exakt
Einpass-Verfahren	**Datenzugriff**	beliebig

Anfragen

- einmalige Anfragen
  - wird aktiv vom Benutzer einmalig gestellt
  - Schnappschuss des Zustands zu einem Zeitpunkt

- kontinuierliche Anfragen
  - befindet sich im System und wird kontinuierlich an neu ankommende Daten gestellt
  - Auswertung über einen Zeitraum

Anfragen II

- Ad-hoc-Anfrage
  - zu beliebigem Zeitpunkt gestellt
  - frühere Daten?
    - ignorieren
    - Zusammenfassungen speichern

- vordefinierte Anfrage
  - bekannt, bevor Daten ankommen
  - kontinuierliche oder geplante einmalige Anfrage

Sliding Windows

- Problem: blockierende Operationen
  - join, avg, max, min …
- Lösung: Sliding Windows
  - zeitbasiert
  - tupelbasiert
  - partitioniert

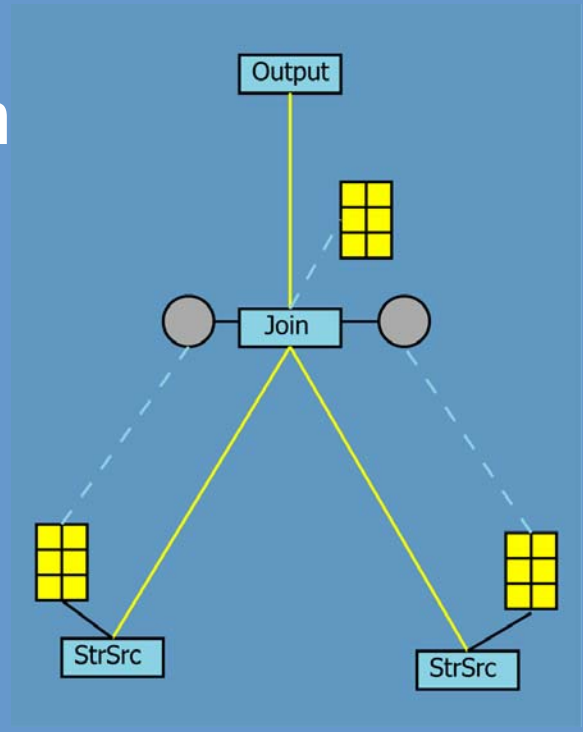
Sliding Windows II

- selbstbeschreibende Technik
- deterministisch: Anfrage über die selben Daten bringt gleiches Ergebnis
- nur neue Daten in Auswertung
- Benutzer
  - versteht die Erstellung des Ergebnisses
  - kann Qualität und Exaktheit in etwa abschätzen
- gute Möglichkeit angenäherte Ergebnisse zu produzieren

DSMS: STREAM

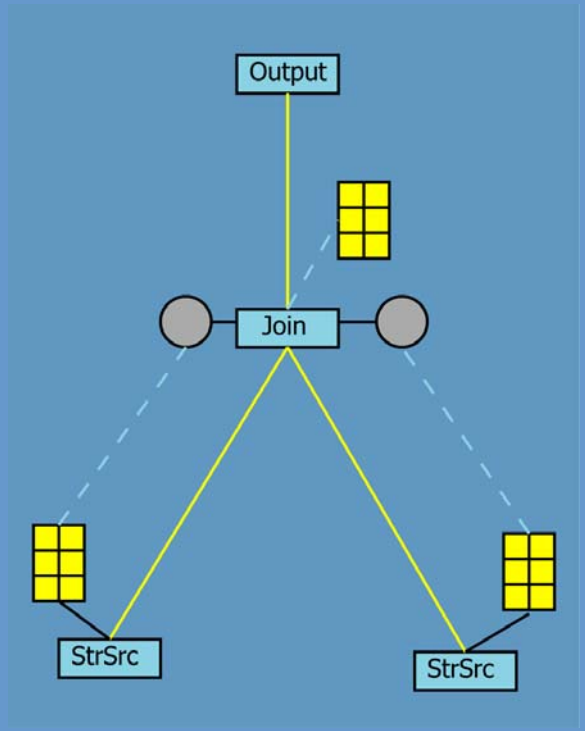
- **ST**anford st**RE**am dat**A M**anager
  - Anfragesprache: CQL continuous query language
  - Bearbeitung kontinuierlicher Anfragen über Datenströme und Relationen
- Anfrageplan aus 3 Komponenten
  - Anfrage-Operatoren
  - Inter-Operator-Warteschlangen
  - Synopsen
- Beispiel:

```
SELECT *
FROM R, S
WHERE R.name = S.name
```



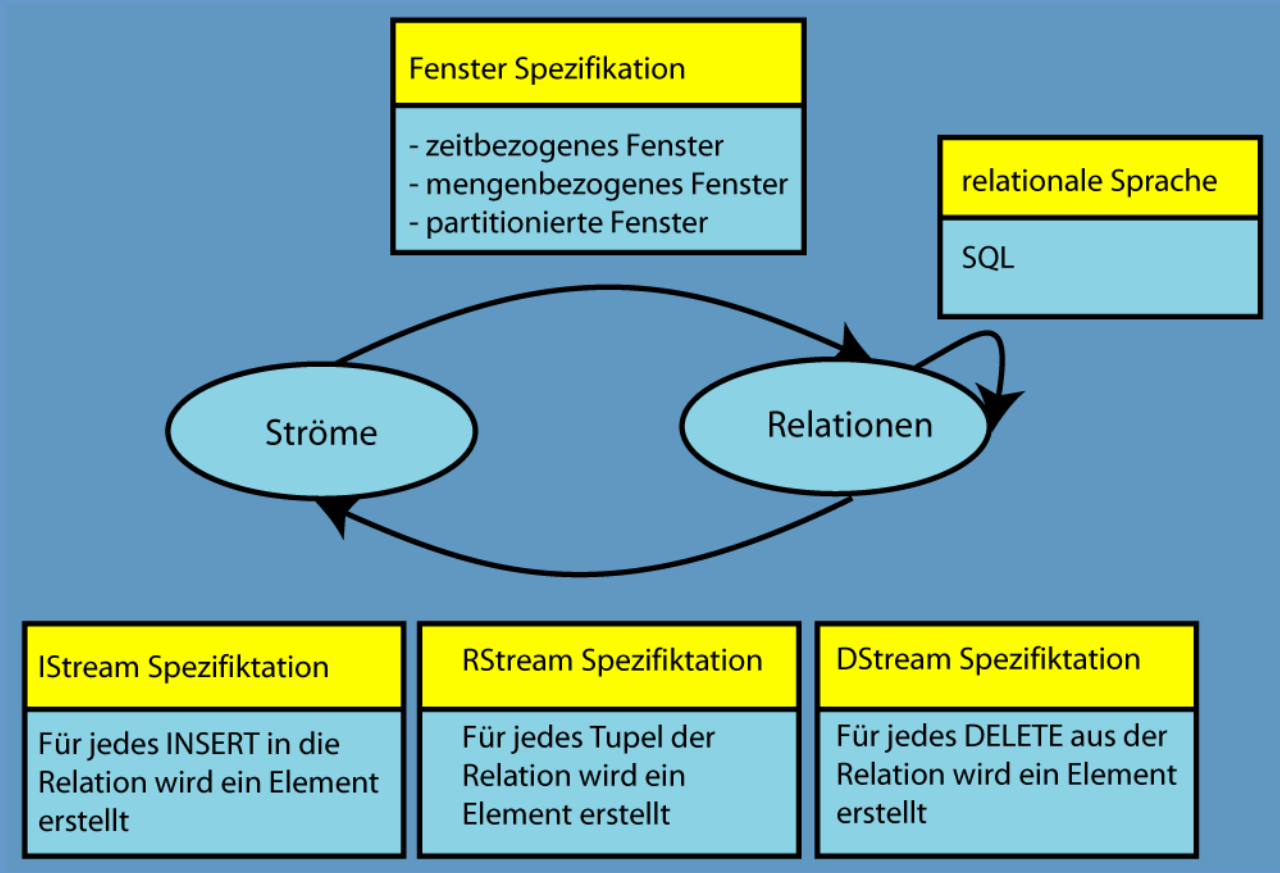
DSMS: STREAM-Komponenten

- Operatoren
  - ähneln denen in DBMS
  - lesen Eingabeströme
  - erstellen Ausgabestrom
- Warteschlangen
  - verbinden Operatoren untereinander
- Synopsen
  - Zusammenfassung bisheriger Daten
  - Trade-off zwischen Präzision und Speicher
  - Sliding Windows als Synopsen



DSMS: STREAM III

- Zusammenspiel der Operatoren

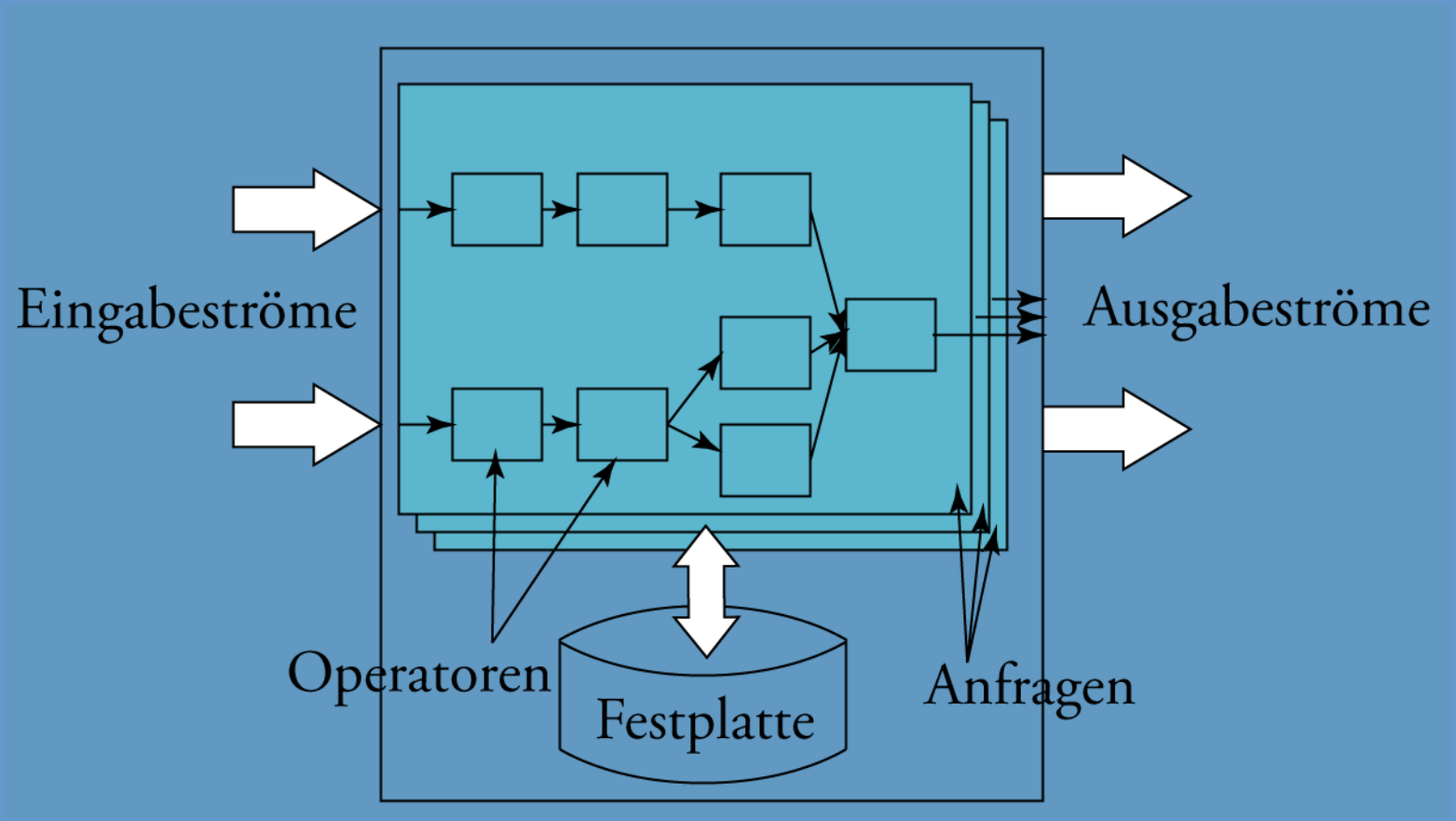


DSMS: AURORA

- Kollaboration von Brandeis University, Brown University, MIT
- unterstützt kontinuierliche / Ad-hoc-Anfragen, Sichten
- Anfragesprache: SQuAl für **S**tream **Qu**ery **Al**gebra
  - Box-and-Arrow-Modell
  - sieben Operatoren
    - filter, map, union, bsort, aggregate, join, resample

DSMS: AURORA II

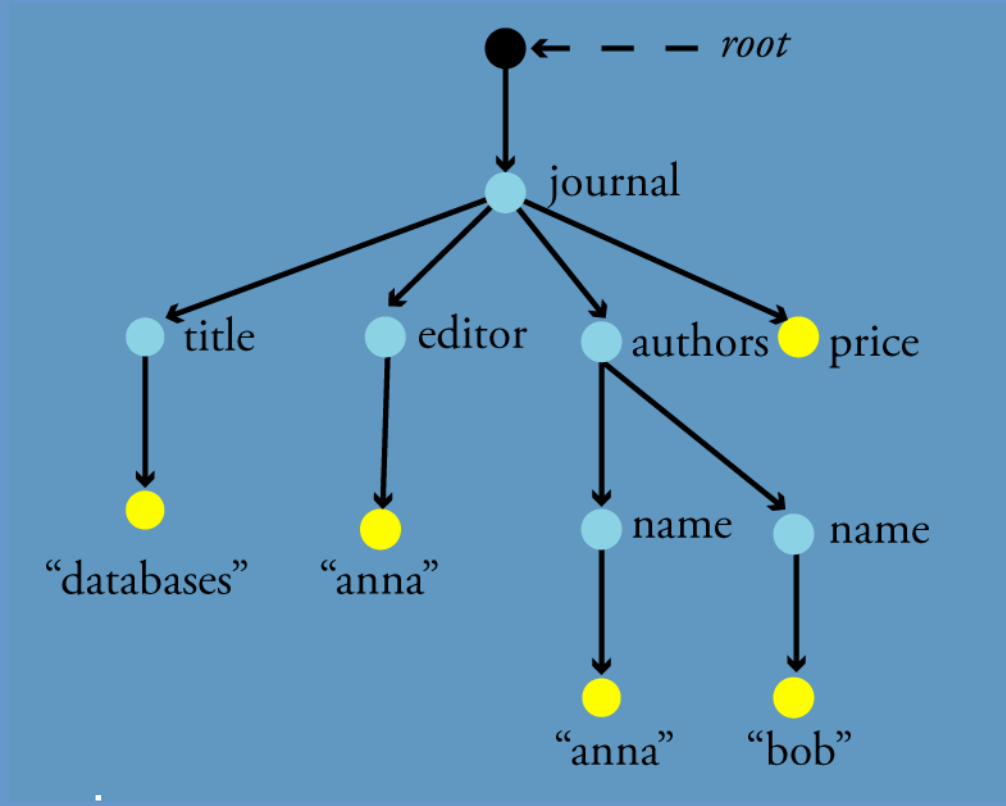
- Box-and-Arrow-Modell



XPATH

- Adressierung von Teilen eines XML-Dokuments mittels Pfad-Notation

```
<journal>
   <title>databases</title>
    <editor>anna</editor>
    <authors>
        <name>anna</name>
        <name>bob</name>
    </authors>
    <price />
</journal>
```



Kontextknoten: journal
Lokalisierungspfad: child::*

→ title, editor, authors, price

XPATH Achsensymmetrie

Vorwärtsachsen	Rückwärtsachsen
child	parent
descendant	ancestor
following-sibling	preceding-sibling
following	preceding
descendant-or-self	ancestor-or-self

XML-Parser: DOM vs. SAX

- programmiersprachenunabhängiger Zugriff auf XML-Dokumente

- DOM = Document Object Model
- \- hoher Bedarf an Hauptspeicher
- \+ beliebiger Zugriff

- SAX = Simple API for XML
- \+ geringerer Bedarf an Hauptspeicher
- \- nur sequentieller Zugriff

nicht geeignet für Datenströme geeignet

XML-Parser: SAX Beispiel

```
<impressum>
Version 1.0 of the Simple API for XML (SAX)
</impressum>
```

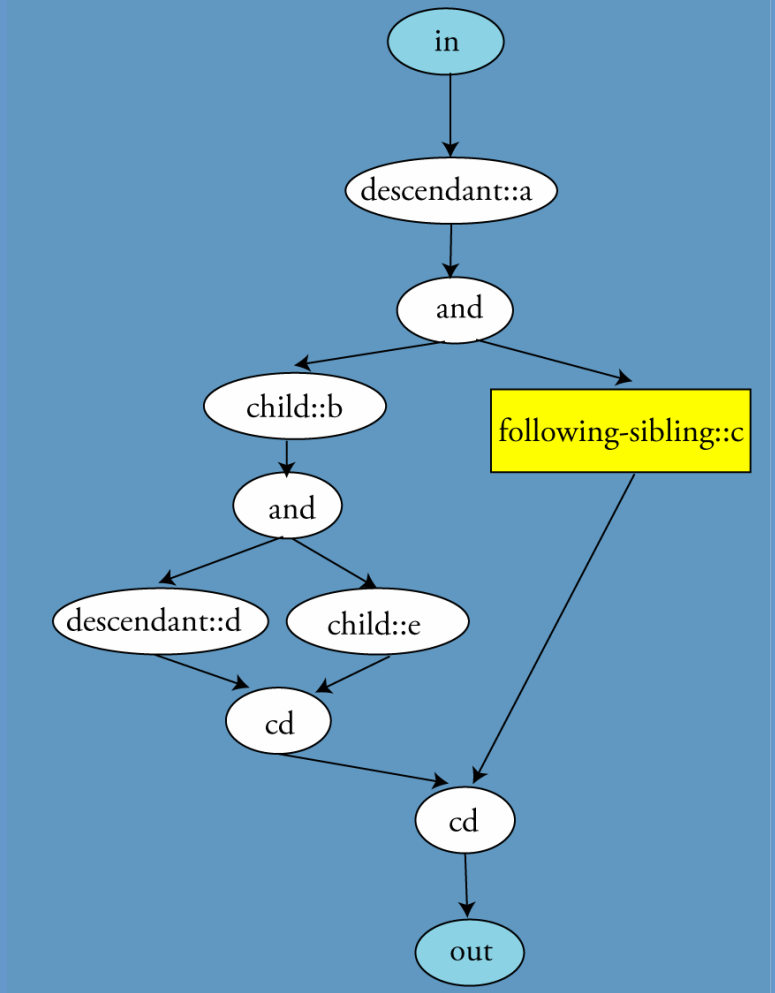
- Jedes XML-Token ist SAX-Ereignis
  - öffnende Markierung <impressum>
  - Text
  - schließende Markierung </impressum>
- entsprechender Eventhandler für jedes Ereignis

DSMS: SPEX

- Streamed and Progressive Emulator for XPATH
- wertet vorwärtsgerichtete XPATH-Anfragen gegen XML-Ströme aus
- überführt diese Anfragen in Automatennetzwerke
- Auswertung in 4 Schritten

DSMS: SPEX Anfrageplan

/descedant::a[child::b[descendant::d or child::e]]following-sibing::c



- Jedes Token des Eingabestroms entspricht SAX Ereignis
- Automaten
  - nutzen Stack um Position der XML-Elemente zu bestimmen
  - führen genau einen Lokalisierungsschritt durch
  - annotieren Strom um Kontextknoten zu markieren, oder leiten unverändert weiter

Load Shedding

- Problem: CPU zu langsam, Hauptspeicher zu klein für ankommenden Datenstrom = Überlast
- Lösung: kontrolliertes Verwerfen von Daten ohne hohen Genauigkeitsverlust = Load Shedding
- am Beispiel Aurora: Daten verwerfen, aber
  - wann
  - wo im Anfrageplan
  - wie viel

  - zufällig
  - semantisch mit QoS Graphen
    - Latency-Graph
    - Loss-Tolerance-Graph
    - Value-Based-Graph

Zusammenfassung

bekannte Techniken aus traditionellen DBMS nicht einfach übertragbar

- keine Speicherung vor Verarbeitung möglich
- push-Kommunikation
- Einpass-Verfahren, weil nur einmaliger sequentieller Zugriff auf Daten
- Echtzeitverarbeitung
- exakte Anfrageauswertung nicht möglich bei
  - Ad-hoc-Anfragen bezüglich vergangener Daten
  - Überlast
  - aggregierenden oder sortierenden Operationen