Vortrag

für Seminar
„Maschinelles Lernen“

von
Nikos Vormwald

Models and Issues in Data Stream Systems

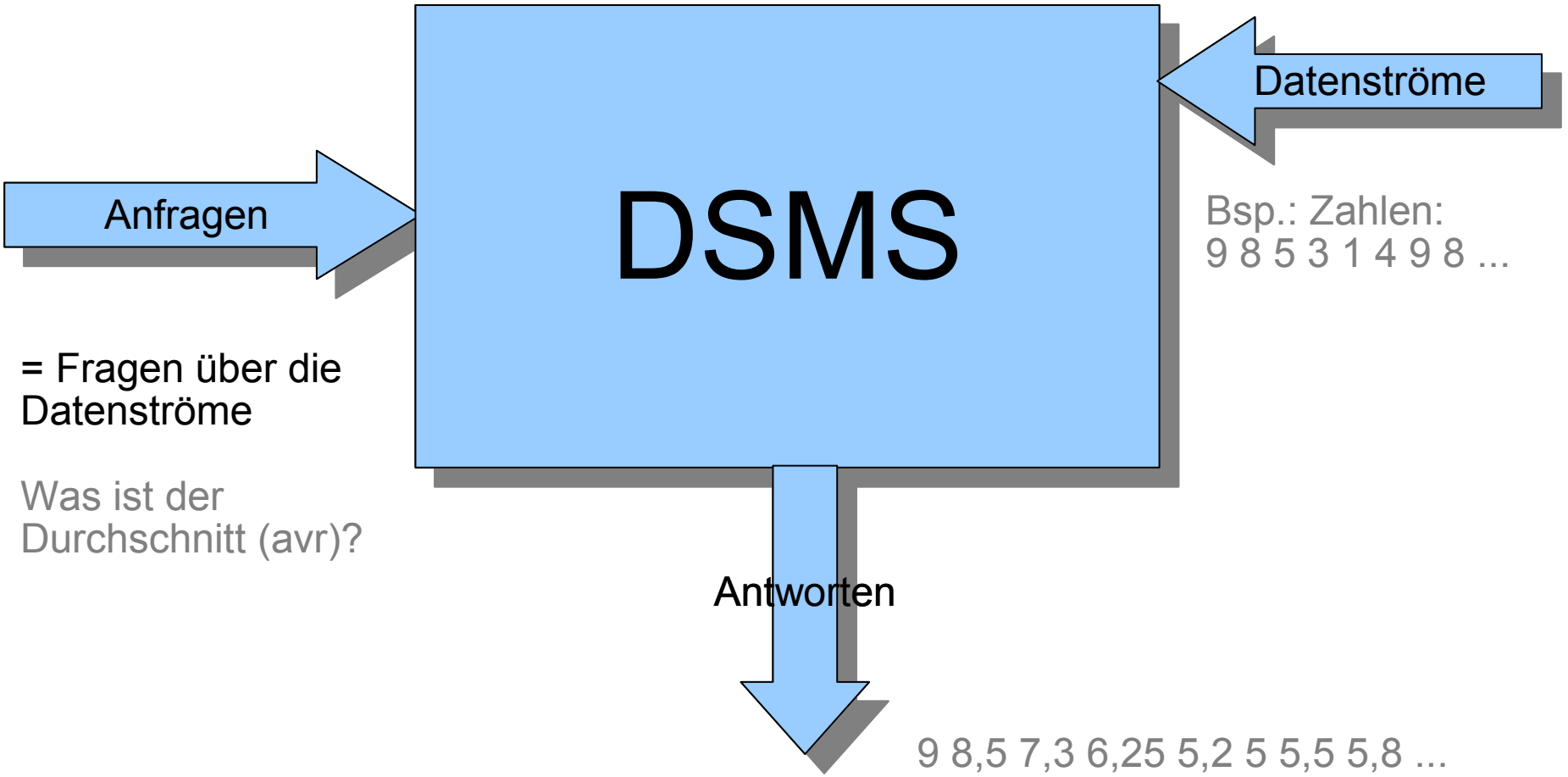
- Neue Klasse von datenintensiven Anwendungen
  - Daten sind Datenstrom
- Beispiele
  - Finanzanwendungen, Netzwerküberwachung
  - Sicherheit, Telekommunikation, Internet
  - Produktion, Sensornetzwerke, etc.

Grenzen herkömmlicher Methoden

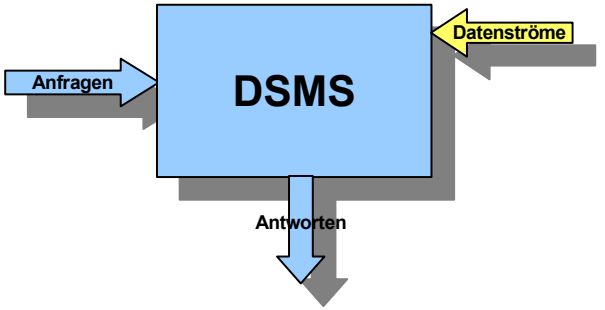
- Alle Daten sammeln + in Datenbank speichern
- DBMS = Database Managment System
  - Anfragen präzise beantworten (alle Daten betrachten)
    - Aber: nicht möglich bei Datenströmen, da Daten zu groß oder nicht verfügbar (kommen erst in Zukunft an)
  - Anfragen sofort beantworten
    - Aber: Datenströme müssen über Zeitraum beobachtet werden (Antwort wird immer genauer)
- => Bedarf für neue Forschung

Data Stream Managment System (DSMS)

- Aufbau

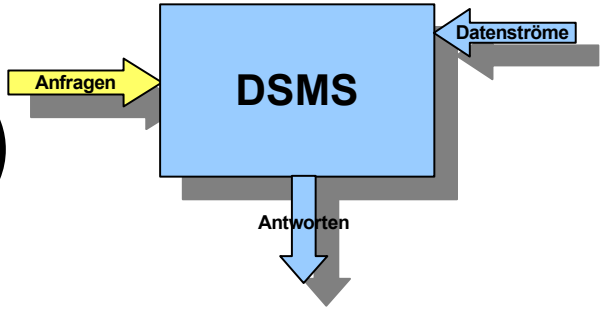


Die Datenströme



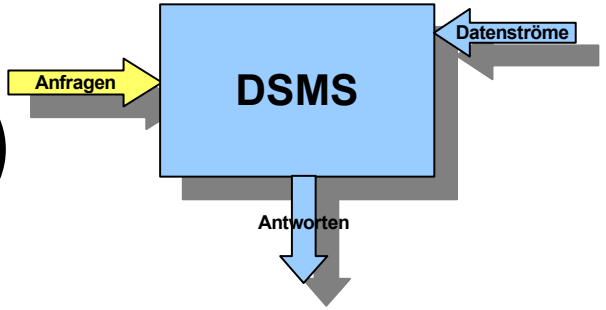
- Einzelne Dateneinheiten sind relationale Tuple
  - Bsp.: (src:144.56.7.2 ; dest:31.77.1.7 ; len:200)
- In mehreren verschiedenen Datenströmen
- Sind unbegrenzt/unberechenbar
- Ankunft: online, sehr/unterschiedlich schnell
- System hat keine Kontrolle über Reihenfolge
- Eingetroffene Elemente werden gelöscht
- Archivierung (langsam, teuer, begrenzt)

Die Anfragen (1)



- Unterscheidung in
  - One-time queries
    - Schnappschuss vom System
    - Antwort basierend auf Schnappschuss
  - Continuous queries
    - kontinuierlich berechnet während Datenstrom eintrifft
    - Antwort entspricht dem Datenstrom, der bis dahin eingetroffen ist
      - Gespeichert und aktualisiert
      - Oder selbst ein Datenstrom

Die Anfragen (2)



- Zweite Unterscheidung in
  - Predefined queries
    - Erreichen DSMS vor relevanten Daten
  - Ad hoc queries
    - Erreichen DSMS online während des Datenstroms
    - Nicht im Vorraus bekannt
    - Korrekte Antwort hängt von Daten ab, die bereits eingetroffen sind (mögl. bereits verloren)

Zahlenbeispiel

predefinied+onetime:
Was ist der
Durchschnitt?

?

adhoc+onetime:
Was ist der
Durchschnitt?

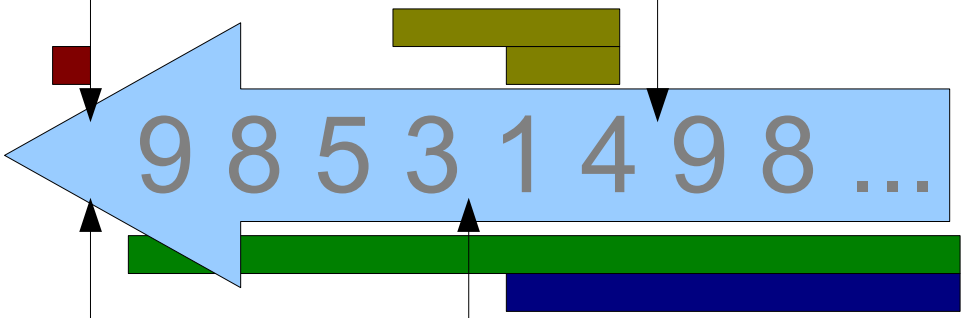
2,5

Wenn sich das System die letzten beiden Zahlen merkt

2,66

Wenn sich das System die letzten drei Zahlen merkt

...



predefinied+continuous:
Was ist der
Durchschnitt?

9
8,5
7,3
6,25
5,2
5
5,5
5,8
...

adhoc+continuous:
Was ist der
Durchschnitt?

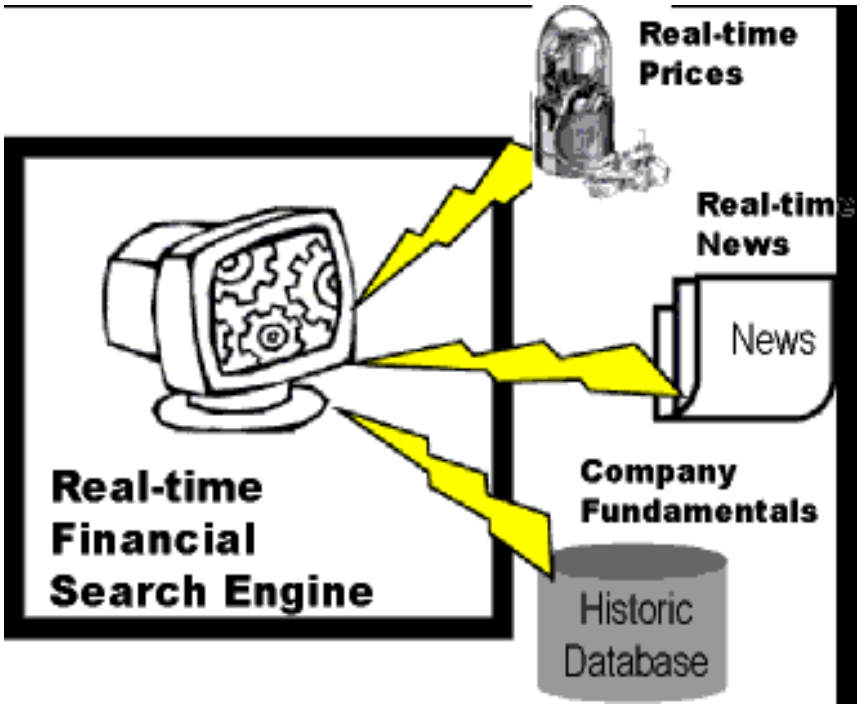
1
2,5
4,66
5,5
...

Könnte auch von gemerkten Zahlen ausgehen

DSMS in Beispielen

- Traderbot (Finanz Such Maschine)
  - Datenströme: Kurs-Ticker, Nachrichten
- iPolicy Networks (Sicherheitsplattform)
  - Datenströme: multi-gigabit Netzwerk Paketströme
  - => Firewall, Intrusion-Detection
- Yahoo (Personalisierung, load-balancing)
  - Datenströme: Clickstreams (web logs)
- Sensorenüberwachung/analyse
  - Datenströme: Daten von Sensoren

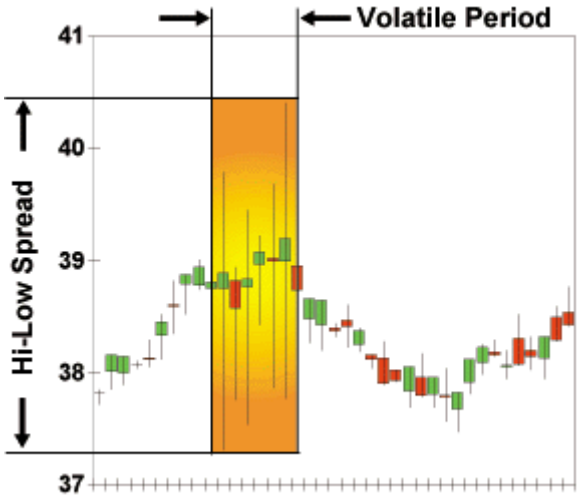
Traderbot.com



Finden, von Ereignissen

Abstürze

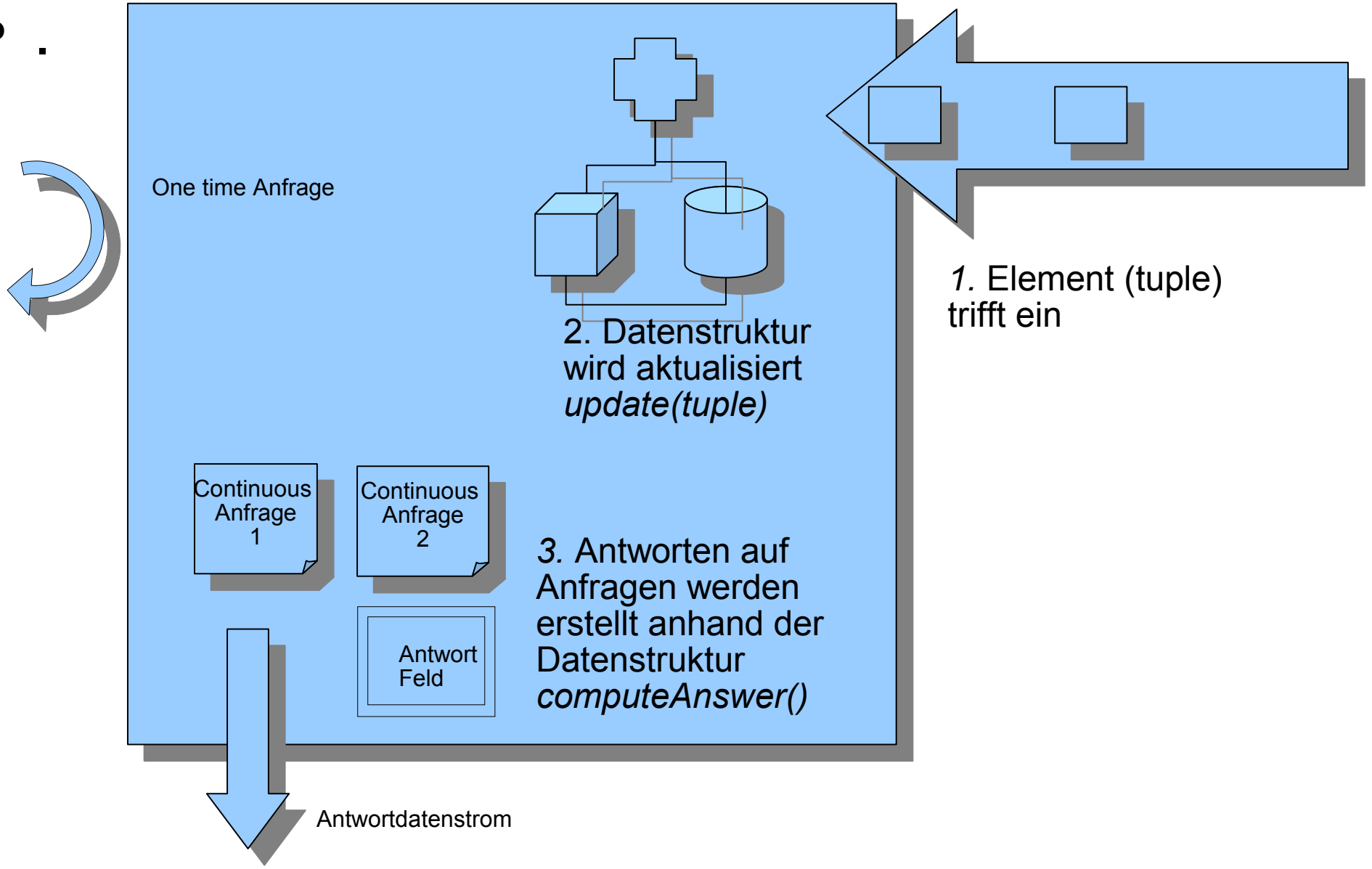
Kurzzeitiges Maximum



Beispielfragen an Traderbot:
Welche Aktien sind um 5% angestiegen in den letzten 10min?

Anfragen beantworten (Inside DSMS)

- .



2 Operationen

- Generell gilt: 2 Operationen:
- update(tuple)
  - Neues Element aus Datenstrom wird verarbeitet
  - Datenstruktur wird verändert/angepasst
- computeAnswer()
  - Antwort auf Anfragen wird erneuert/aktualisiert

Begrenzter Speicher

- Datenströme sind unbegr. -> großer Speicher
- Vorschlag: Festplattenplatz benutzen
- continuous queries
  - Neue Daten treffen ein; alte noch verarbeitet
  - > Algorithmus muss Schritt halten
  - > Beschränkt auf Hauptspeicher (RAM)

Approximierte Antworten

- Exakte Antwort nicht immer möglich
- Aber: Approximationen oft akzeptabel
- Viele Möglichkeiten zur Approximationen
  - Sliding Windows
  - Batch Processing
  - Sampling
  - Synopses/Sketches
    - Decision Trees
    - Cluster
    - ...

Sliding Windows

- Idee: nicht alle Daten betrachten, nur aktuellste
- 2 Arten:
  - Physikalische (letzten 100 Elemente des Stroms)
  - Logische (Elemente der letzten 15 Minuten)
- Vorteile:
  - Fest definiert, einfach nachzuvollziehen, deterministisch
  - Macht Sinn (aktuellere Daten meistens bedeutender)

Batch Processing

- Update = schnell, computeAnswer = langsam
- Lösung: Daten werden in Batches verarbeitet
- D.h.: Datenelemente werden gebuffert; Antwort auf Anfrage wird periodisch berechnet
- Nachteil:
  - Antwort ist nicht immer die aktuellste Antwort
- Vorteile:
  - Antwort ist korrekt (gewesen in der Vergangenheit)
  - Gut bei impulsartigem Datenstrom
    - Schnelle Übertragung = buffern; langsam = berechnen

Sampling

- Update = langsam, computeAnswer = schnell
- Zeit um Element zu verarbeiten > Intervall zwischen Elementen im Datenstrom
  - > zwecklos alle Elemente verarbeiten zu wollen
- Elemente werden verworfen
- -> Antwort wird mit Auswahl an Elementen berechnet
- Nachteil:
  - Annäherung an richtige Antwort nicht garantiert

Synopsis Data Structures

- Idee: update und computeAnswer schneller machen
- Datenstruktur nicht Exakte Representation, sondern Synopsis/Sketch (= annährende Datenstruktur)
- => Minimum an Berechnungzeit pro Element
- Alternative zu Batch Processing & Sampling
- ergebnisreiches Forschungsgebiet

Blocking Operators (1/2)

- Benötigen gesamte Eingabe um Ergebnis zu produzieren
- Bsp.: sort, sum, count, min, max, avg
- Datenstrom unendlich -> nie Ergebnis
- Aber: Operationen notwendig
- Ansatz 1: Operationen ersetzen durch nicht blockierende mit annähernd selben Resultat
  - sort -> juggle (lokales Umordnen der Datenstroms)
  - Andere Ersetzungen? Fehlerabschätzung?

Blocking Operators (2/2)

- Ansatz 2: Datenstrom erweitern mit Informationen darüber was noch kommen kann
  - „für alle zukünftigen Tuple gilt: daynumber > 10“
  - -> Für 1-10 kann blockende Operation angewendet werden, da alle nötigen Daten vorhanden sind
- Andere Informationen über Datenstrom:
  - geclustert? ab/aufsteigend?
- Kann helfen Operationen zu „entblocken“

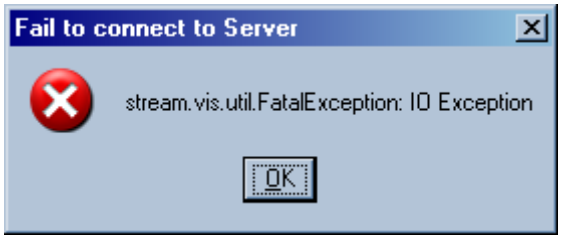
Zugriff auf vergangene Daten

- Datenelement verloren, wenn vorbeigeströmt
- Ad hoc queries mögl. nicht korrekt beantwortbar
- Lösung 1: Ad hoc queries nur für zukünftige Daten berechnen
  - Akzeptabel für viele Anwendungen
- Lösung 2: Zusammenfassung vom Datenstrom bilden

  = annährende Antwort auf zukünftige ad hoc queries
  - Problem: Was speichern? Wie zusammenfassen?

STREAM

- STanford stREam DatA Manager
- Im Web:
  - Projekt eingestellt; Java-Exception
- Übersicht
  - Sprache
  - Timestamps
  - Anfrage Verarbeitungsarchitektur



Sprache

- Sprache für Anfragen: SQL (modifiziert)
  - Neu: window specification
  - *partition by*: jede Gruppe in eigenem Fenster
    - *partition by customer_id*
  - Fenstergröße:
    - *rows 10 preceding* = physische Größe
    - *range 15 minutes preceding* = logische Größe
  - Filter auf das Fenster
    - *where typ = 'long distance'*

Anfrage Beispiel

- SELECT AVG(Window.minutes)
  FROM
    (SELECT Calls.customer_id,Calls.minutes
    FROM Calls, Customers
    WHERE Calls.customer_id = Customers.customer_id
    AND Customers.typ = 'Gold')
    Window [ROWS 3 PRECEDING]
- => liefert durchschnittliche Länge der letzten 3 Gespräche von 'Gold'-Kunden.

Kunden Datenbank

customer_id	customer_typ	...
1	Gold	???
2	Gold	???
3	Standard	???
4	Gold	???
...	...	...

+

Stream der Anrufe

Minutes	customer_id	....
8	2	???
3	1	???
6	3	???
21	4	???
...	...	... .

=

Sliding Window

customer_id	Minutes
2	8
1	3
4	21

avg

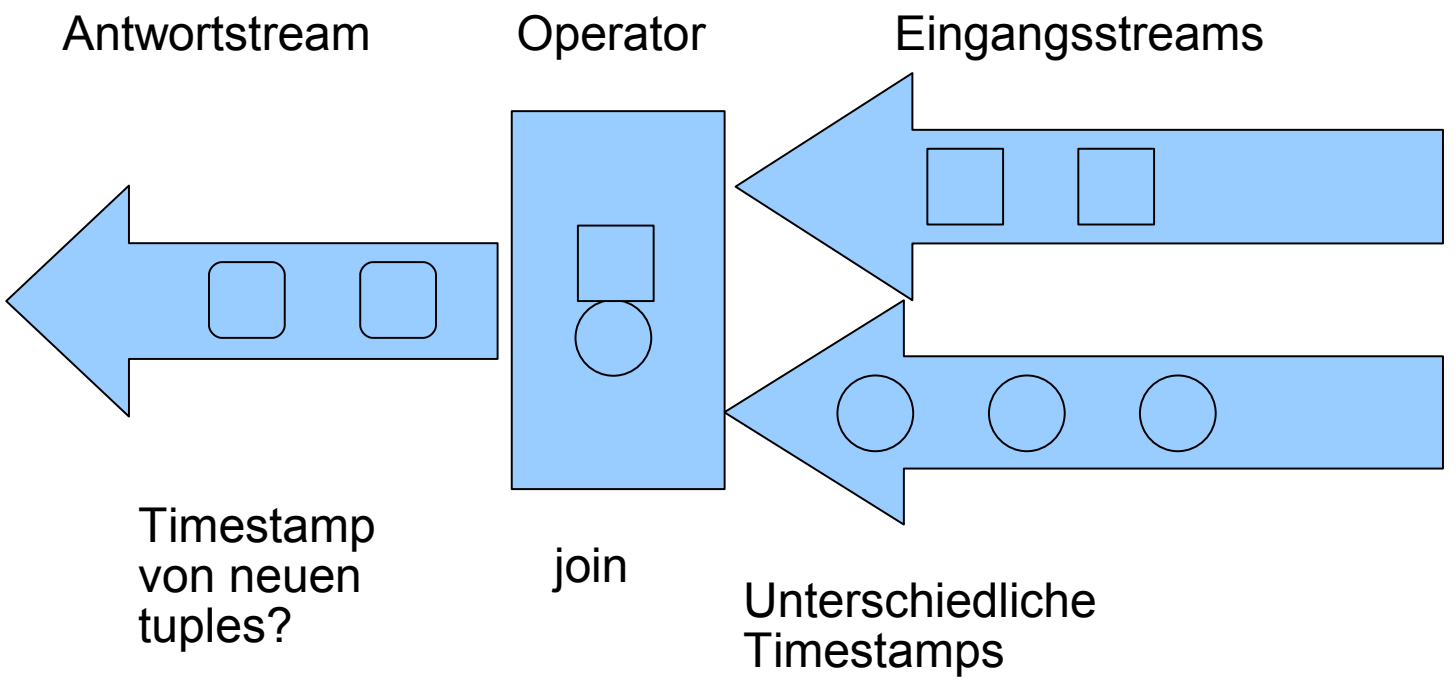
10,3

Timestamps
Arten

- Implicit
  - Zeit nicht im Tuple angegeben / nicht wichtig
  - Spezielles Feld zu jedem Datentuple hinzugefügt
  - ~ Ankunftszeit des Tuples im System
- Explicit
  - Datenfeld wird für Timestamp verwendet
  - Tuple entspricht Ereignis in realer Welt
  - ~ Entstehungszeit des Tuples
  - Nachteil: können in falscher Reihenfolge am DSM-System eintreffen
    - Aber ungefähr korrekt, Ausgleich durch Buffering

Timestamps Kombination (1/3)

- Aber: was bei Kombination aus 2 Streams?



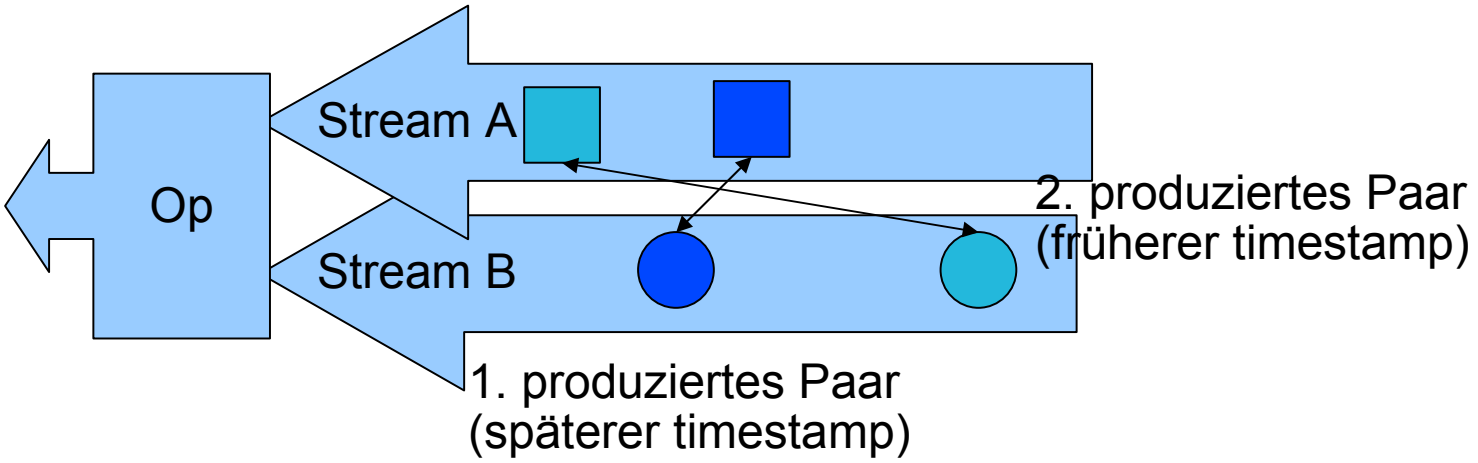
Timestamps Kombination (2/3)

- „best effort“-Ansatz:
  - Keine Garantie für Reihenfolge der neuen Tuples
  - Annahme: frühere Ankunft -> frühere Verarbeitung
  - Ausgabereihenfolge hängt von Implementierung/Zeitplan des Operators ab
  - Tuple bekommt implizites Timestampfeld angehängt bei Produktion durch Operator

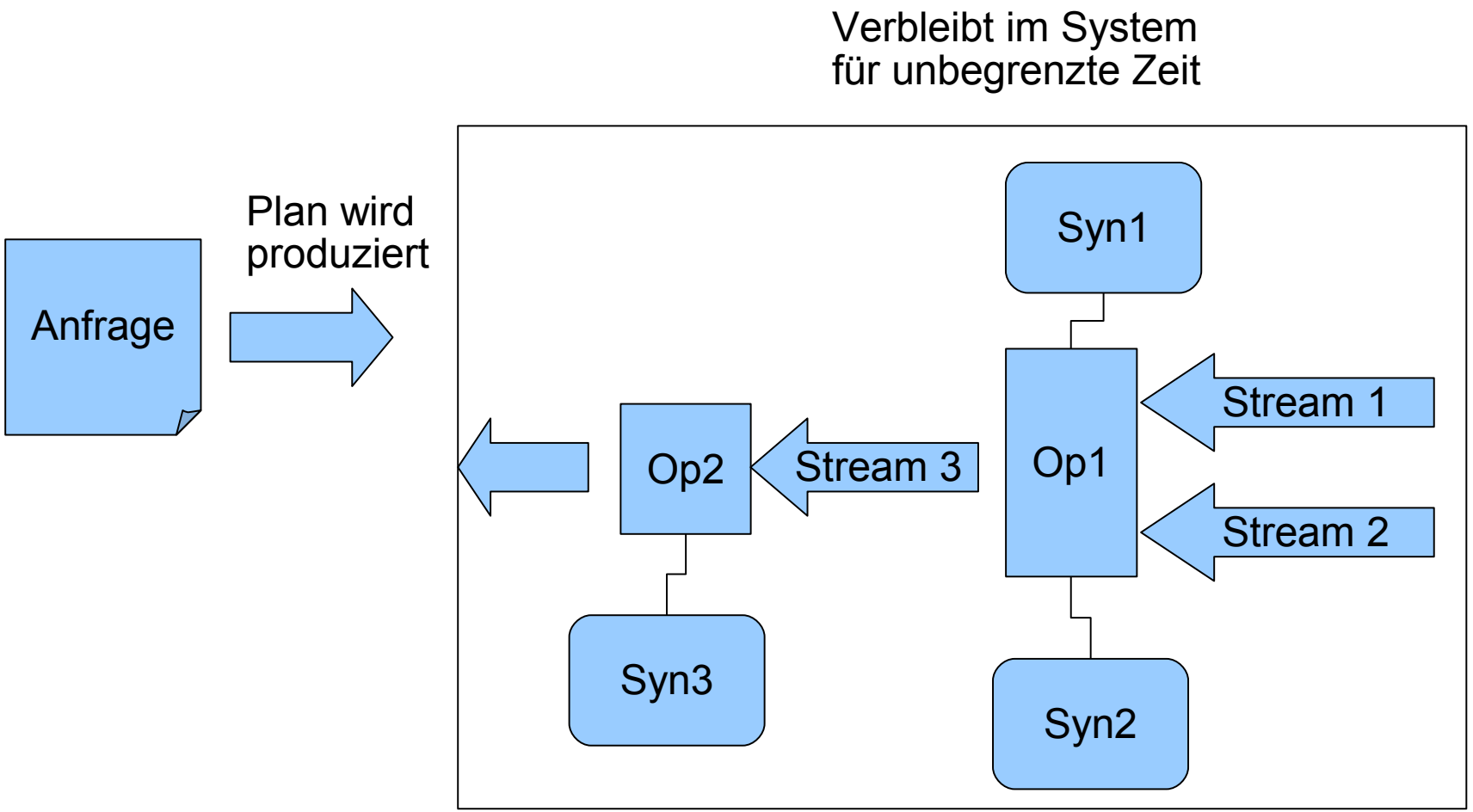
Timestamps Kombination (3/3)

- Zweiter Ansatz
  - User spezifiziert in Anfrage, wie neuer Timestamp erstellt wird
  - Anfrage: „Stream A mit Stream B kombinieren“

    -> Timestamp vom erstgenannten Stream übernommen
- Nachteil: Buffering nötig um sortierte Ausgabe zu produzieren



Anfrage Verarbeitungsarchitektur



Operator1 = z.B.: join, Operator2 = z.B.: avg
Synopsis können Sliding Windows etc sein

Zentraler scheduler

- Erlaubt Operatoren die Ausführung
  - Zeitbasiert: „Operator1 darf 10 Sekunden laufen“
  - Quantitativ:
    - „Operator1 darf 10 Eingangstuple verarbeiten“
    - „Operator1 muss 10 Ausgangstuple produzieren“
- Ausführung eines Operators beinhaltet:
  - Aktualisieren der Synopsis (update(tuple))
  - Ausgabe des Ergebnisses (computeAnswer())
- Verschiedene Policies möglich
  - Welcher Operator wann? Wie lange? Unter welchen Bedingungen?

Operatoren

- In STREAM adaptiv designed
  - > je mehr Speicher, desto genauere Ergebnisse
  - Speicher kann online für Operatoren verringert/erhöht werden

  = Speicher <-> Genauigkeit Tradeoff:
- Fragen
  - Wie können annährende Ergebnisse erzeugt werden?
  - Wie verhalten sich annährende Ergebnisse wenn man sie miteinander kombiniert?
  - Wie soll Speicher an Operatoren verteilt werden, um genauere Ergebnisse zu produzieren?
  - Wie optimalen Plan aus Anfrage erstellen?

Herrausforderungen

- Mehrere laufende Anfragen -> mehrere Pläne müssen Ausführungszeit zugeteilt bekommen
- Zeitvarrierende Raten der Eingangsströme/Ausgangströme
  - Muss Synchronisiert werden
  - Buffer für Eingangsstreams verwalten
  - Verändern der Speicherzuteilung online
  - Verändern des Ausführungsplans online

Algorithmen

- Data Stream Algorithmus:
  - Eingabe: x1,...,xn,... (Datenstrom)
    - Nur einmal gescannt in steigender Reihenfolge
  - Ausgabe: Wert einer Funktion f erhalten, anhand des bisher gesehen Datenstroms
- Komplexität: Speicherverbrauch, Zeit
  - Sollte unabhängig sein von N (=Anzahl gescannter Elemente = unendlich) -> nicht möglich
  - Unter O(poly(log N)) als „well-solved“ betrachtet
    - Machmal nichtmal mit Annäherungen möglich

Algorithmen

- Viele Bemühungen vorhandene Algorithmen auf Data Stream Modell anzupassen und zu optimieren
- Aproximation um Zeit/Speicher zu sparen
- Forschungsarbeiten zu Vielzahl von Algorithmen
- Suche nach neuen Optimierungsmöglichkeiten
- Viele offene Fragen...

Zusammenfassung

- Arbeiten mit Datenströmen erfordert neue Überlegungen für
  - Datenmanagment
  - Anfrageverarbeitung
  - Algorithmen
- “Meta”-Fragen:
  - Erweiterung von normaler Datenbank-Technologie ausreichend? (oder komplett neue Methoden?)
  - Jede Anwendung von Grund auf neu entwickeln? (oder vielseitig einsetzbare Grundsysteme entwickeln?)
  - Unlösbare Anwendungen für DSMS?

Quellen

- Paper „Models and Issues in Data Stream Systems“
- www.traderbot.com
- www-db.stanford.edu/stream