Informationsintegration

Optimierung verteilter Anfragen

Ulf Leser

Inhalt dieser Vorlesung

- Optimierungsziele
- Verteilte Anfrageausführung
- Kostenschätzung
- Semi-Joins
- Beschränkte Quellen

Recap: Klassische Anfrageoptimierung

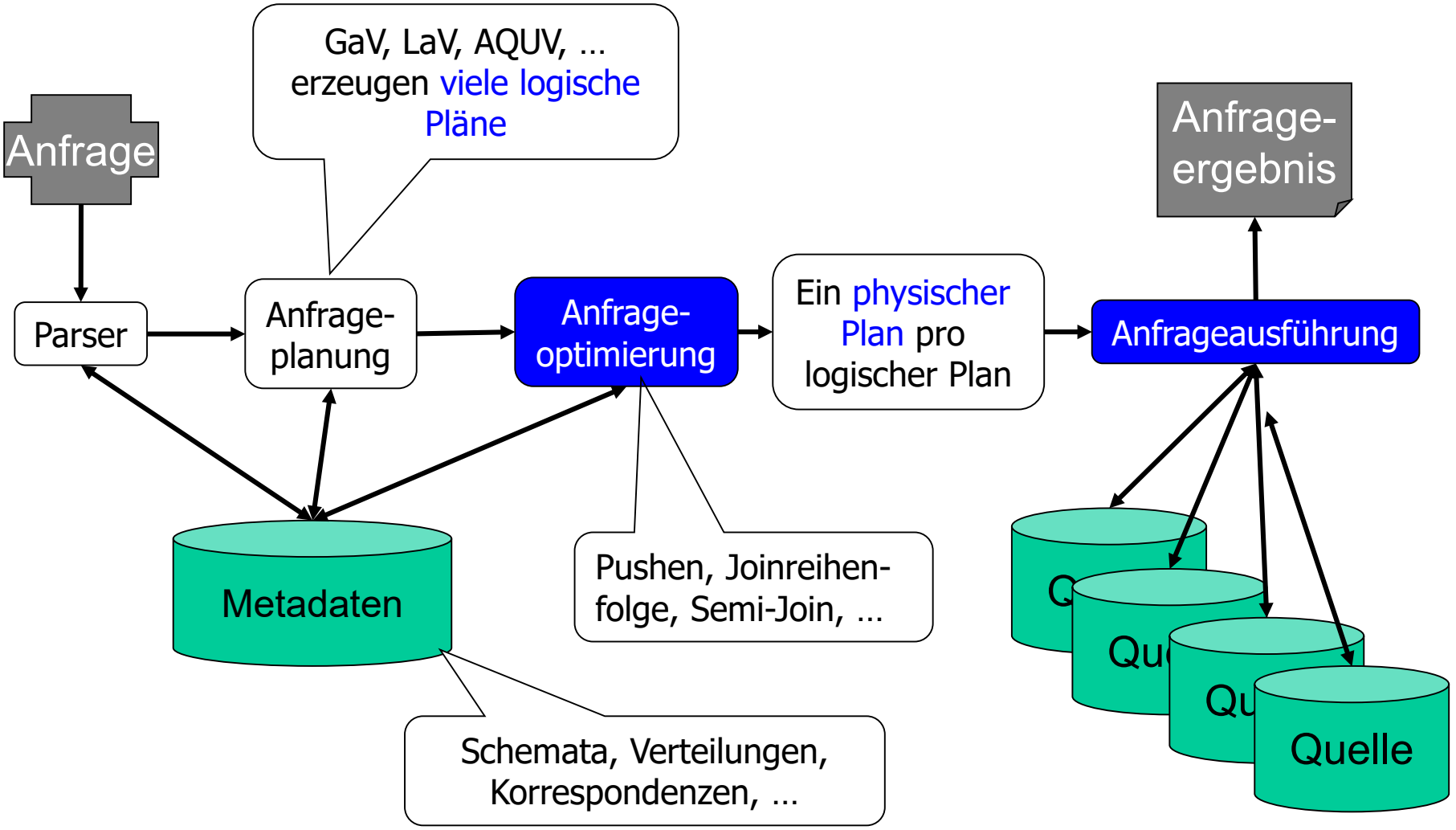
- Zentrale relationale Datenbanken
  - Eine Anfrage – viele Anfragepläne
  - Alle Anfragepläne berechnen dasselbe Ergebnis
  - Optimierung: Berechne dieses Ergebnis in möglichst kurzer Zeit
- Methoden
  - Algebraische Umformungen, Pushen von Selektionen und Projektionen, Join-Order, Zugriff über Indexe, etc.
- Klassische Heuristiken zur Optimierung
  - Was nicht in den Hauptspeicher passt, muss auf Disk
  - Zeit hängt im wesentlichen vom IO Zugriff ab
  - IO-Kosten hängen von der Datenmenge ab
  - Also: Minimiere die insgesamt zu bewegende Datenmenge

Optimierung in der Informationsintegration

- Integration: Verschiedene logische Pläne erzeugen idR verschiedene Ergebnisse
- Netzwerk: Zugriffe dauern unterschiedlich lang
- Kosten: Hängen vor allem vom Netzwerk ab
- Möglichkeiten: Beschränkte Quellen

- Unterschiede zu verteilten Datenbanken
  - Verteilte DB können Verteilung optimieren (Data Placement)
  - Verteilung und Redundanz ist kontrolliert
  - Menge der Knoten steht auf Dauer fest (weniger Wartung)

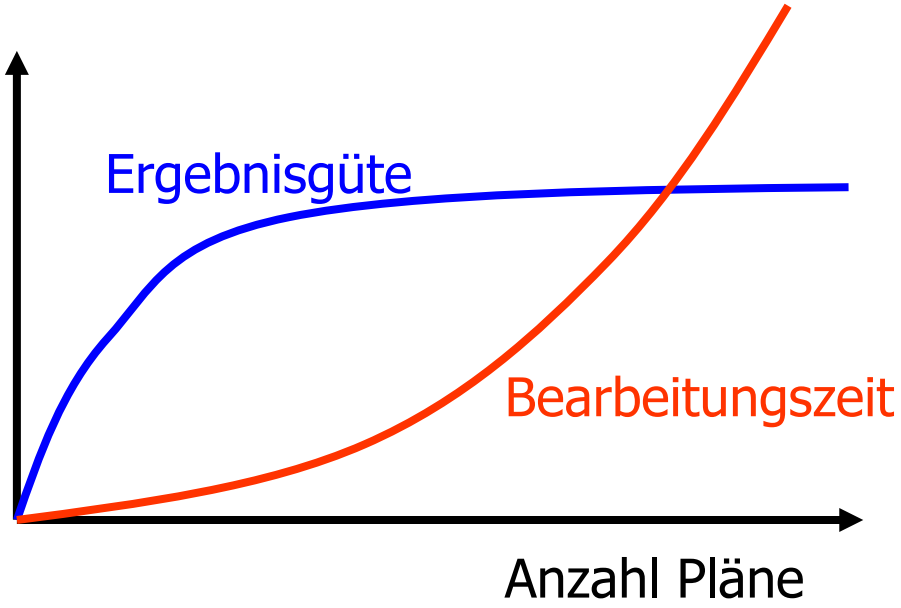
Anfrageoptimierung und –ausführung zur virtuellen Informationsintegration



Optimierungsziele

- Da verschiedene logische Pläne verschiedene Ergebnisse bringen, ist das Optimierungsziel nicht per-se klar
- Möglichkeiten
  - Führe alle semantisch korrekten Pläne aus
    - mit der minimalen Anzahl an Anfragen
    - mit der minimalen Menge an zu übertragenden Daten
    - in kürzester Gesamtzeit
  - Führe die k besten Pläne aus
    - „Guter Plan" – viele und qualitativ hochwertige Ergebnistupel
    - Siehe Anhang / [NLF99]
  - Führe die k Pläne aus, die zusammen das beste Ergebnis liefern
    - Benötigt Abschätzungen über die Überlappung zwischen Ergebnisse
  - Führe die k Pläne aus, die zusammen das beste Ergebnis liefern bei maximalen Kosten c

Typischer Trade-Off



- Abwägung: Kosteneinsatz pro Ergebnisverbesserung
- Problem: Man kann beides nur schlecht abschätzen

Kosten eines physischen Plans

- Wir sprechen immer von abstrakten „Kosten“
  - Einheiten: Zeit, Datenmenge
- Kostenmodelle müssen beinhalten
  - Kosten für den Aufbau der Verbindung (Latenzzeit)
    - Finden der Quellen im Netz (DNS etc.)
    - Quelle muss DB-Verbindung aufbauen, CGI-Script starten etc.
  - Kosten für die Berechnung der Daten in der Quelle
    - „Sehr langsame Quelle + schnell Leitung“ > „Schnelle Quelle + mittlere Leitung“?
  - Kosten für den Datentransport (möglicher versus realer Durchsatz)
  - Kosten für die Nachbearbeitung der Daten im Mediator

Inhalt dieser Vorlesung

- Optimierungsziele
- Verteilte Anfrageausführung
  - Algebraische Optimierung
  - Parallele Ausführung von Aufrufen
  - Puffern: Row blocking
  - Wer schickt wem was?
  - Caching
- Kostenschätzung

Allgemeines Modell

- Wir betrachten ab jetzt meistens nur noch einen Plan
- Im Netzwerk können alle mit allen reden
- Planausführung wird von einem Knoten initiiert
  - Dort müssen auch die Ergebnisse ankommen
- Die Basisrelationen liegen in einer oder mehreren Quellen
- Kooperative, mächtige Quellen. Alle Operatoren eines Plans können prinzipiell auf jedem Knoten ausgeführt werden
- Bestimmung des Ortes der Ausführung von Operationen ist eine Optimierungsaufgabe
  - Um Netzwerkverkehr zu sparen: Operationen zu Daten
  - Trivial: Selektionen und Projektionen
  - Nicht so trivial: Joins

1. Algebraische Optimierung

- Zur Erinnerung
  - $\mathbf{result(q) = \cup\ result(p)}$

$$\mathbf{= p_1 \cup p_2 \cup \ldots \cup p_n}$$
$$\mathbf{(v_{11} \bowtie v_{12} \bowtie \ldots \bowtie v_{1n}) \cup}$$
$$\mathbf{(v_{21} \bowtie v_{22} \bowtie \ldots \bowtie v_{2n}) \cup}$$
$$\mathbf{\ldots \cup}$$
$$\mathbf{(v_{m1} \bowtie v_{m2} \bowtie \ldots \bowtie v_{mn})}$$

- Umformungen möglich
  - Z.B: $\mathbf{v_{11} \cup (v_{21} \bowtie v_{22}) = (v_{11} \cup v_{21}) \bowtie (v_{11} \cup v_{22})}$
- Durch die Umformung bekommt man andere Operatoren, die auch anders platziert werden können

Das Problem

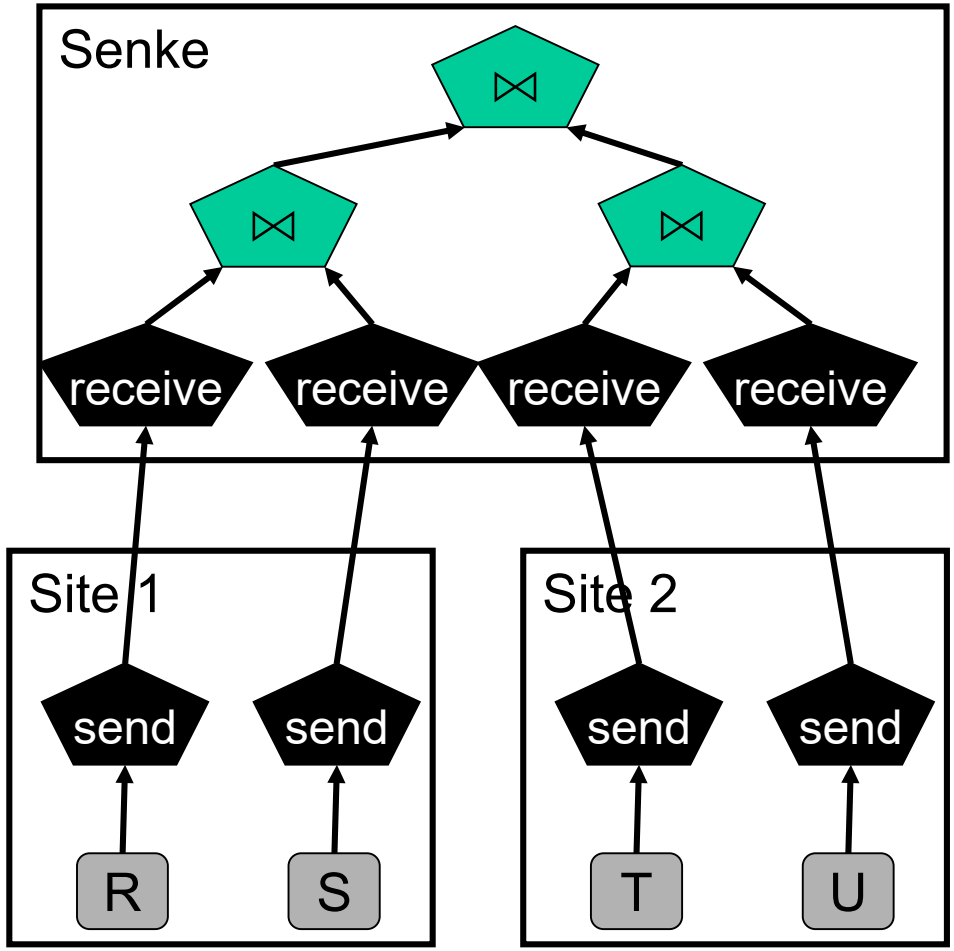
- Eingabe: Logischer Plan P, Verteilung der Tabellen, viele Schätzungen von Kardinalitäten, Übertragungszeiten, etc.
- Gesucht: Optimale physische Plan P‘
  - P’ berechnet das selbe Ergebnis wie P (semantisch äquivalent)
  - Jeder Operator des Plans wurde auf einen Knoten platziert
  - Es existiert kein äquivalenter Plan P” der besser ist als P‘
- Problem: Der Suchraum explodiert
  - Schon Join-Order-Optimierung ist ein NP-schweres Problem
  - Hier noch schlimmer: Platzierung der Operatoren
  - Schätzungen bei autonomen Quellen oft nicht vorhanden

Real-Life

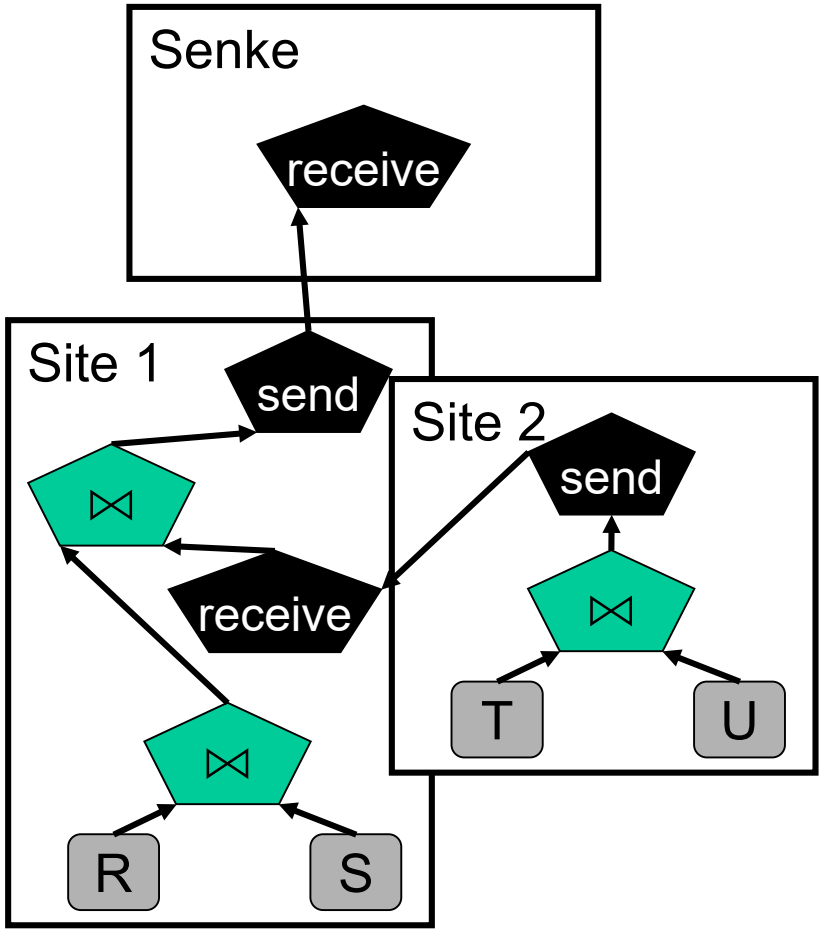
- Heuristiken
  - Zeit ignorieren und nur Zwischenergebnisse minimieren
    - Braucht auch schon gute Schätzungen
  - Operatoren immer dahin legen, wo die größte Eingabe ist
    - Wenn mehrere Eingaben mit unbekannter Größe: zufällig auswählen
  - Joins und Union immer im Mediator
    - Vergrößern meistens das Ergebnis – schlecht
    - Ggf. vorab Semi-Join Plan ausführen (später)
  - …
- Wir besprechen keinen Algorithmus
- Wir diskutieren Möglichkeiten und Fragestellungen

Beispiel: (R⋈S)⋈(T⋈U)

Heuristik: Alle Joins im Mediator

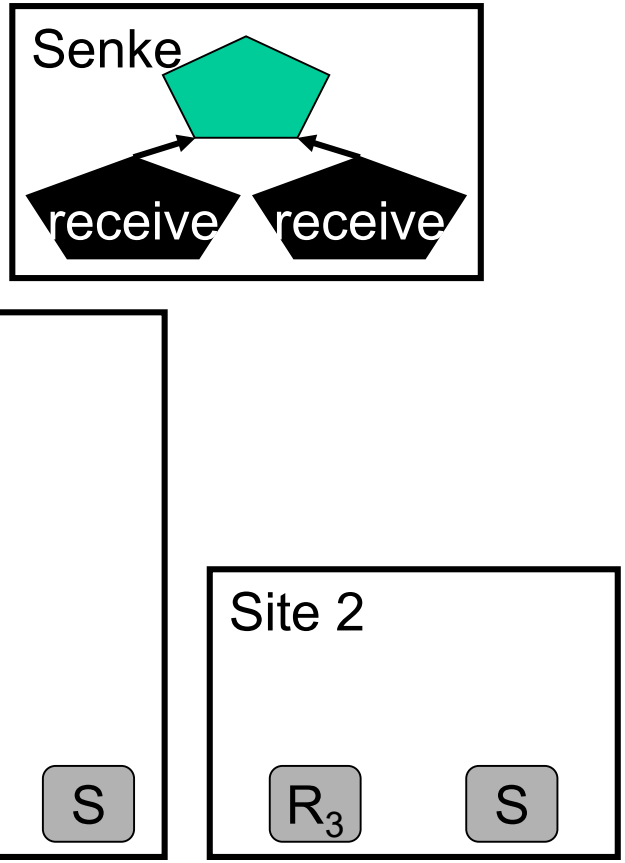


Joins bei Datenquelle

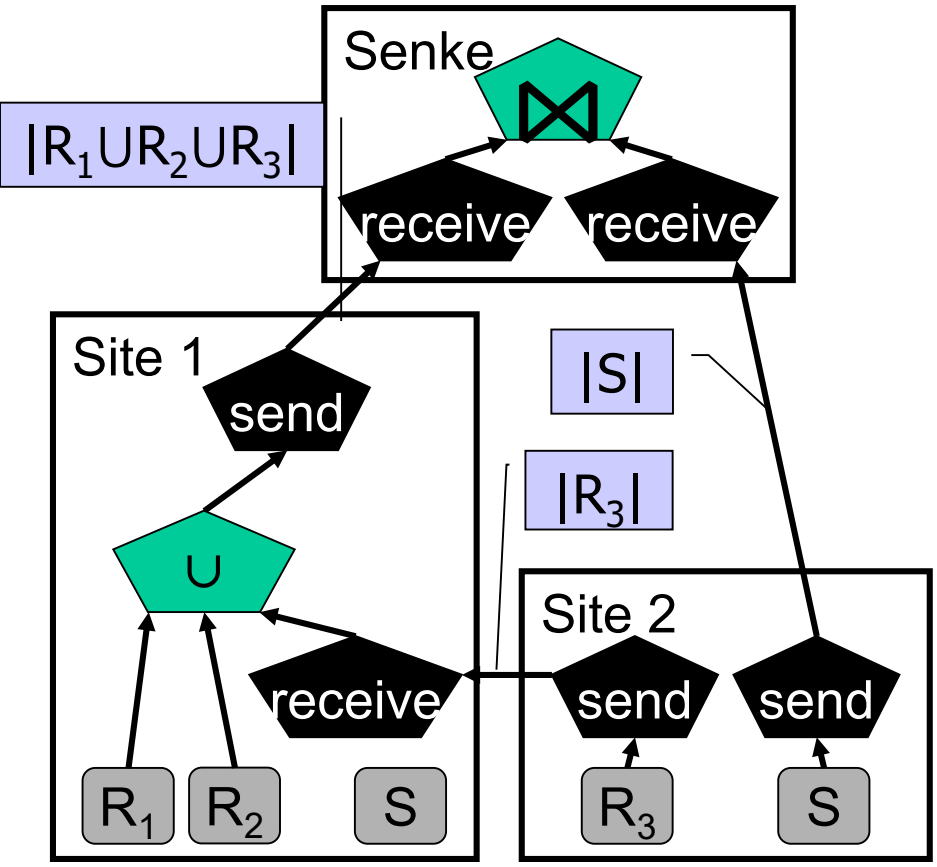


Beispiel: $(R_1 \cup R_2 \cup R_3) \bowtie S$

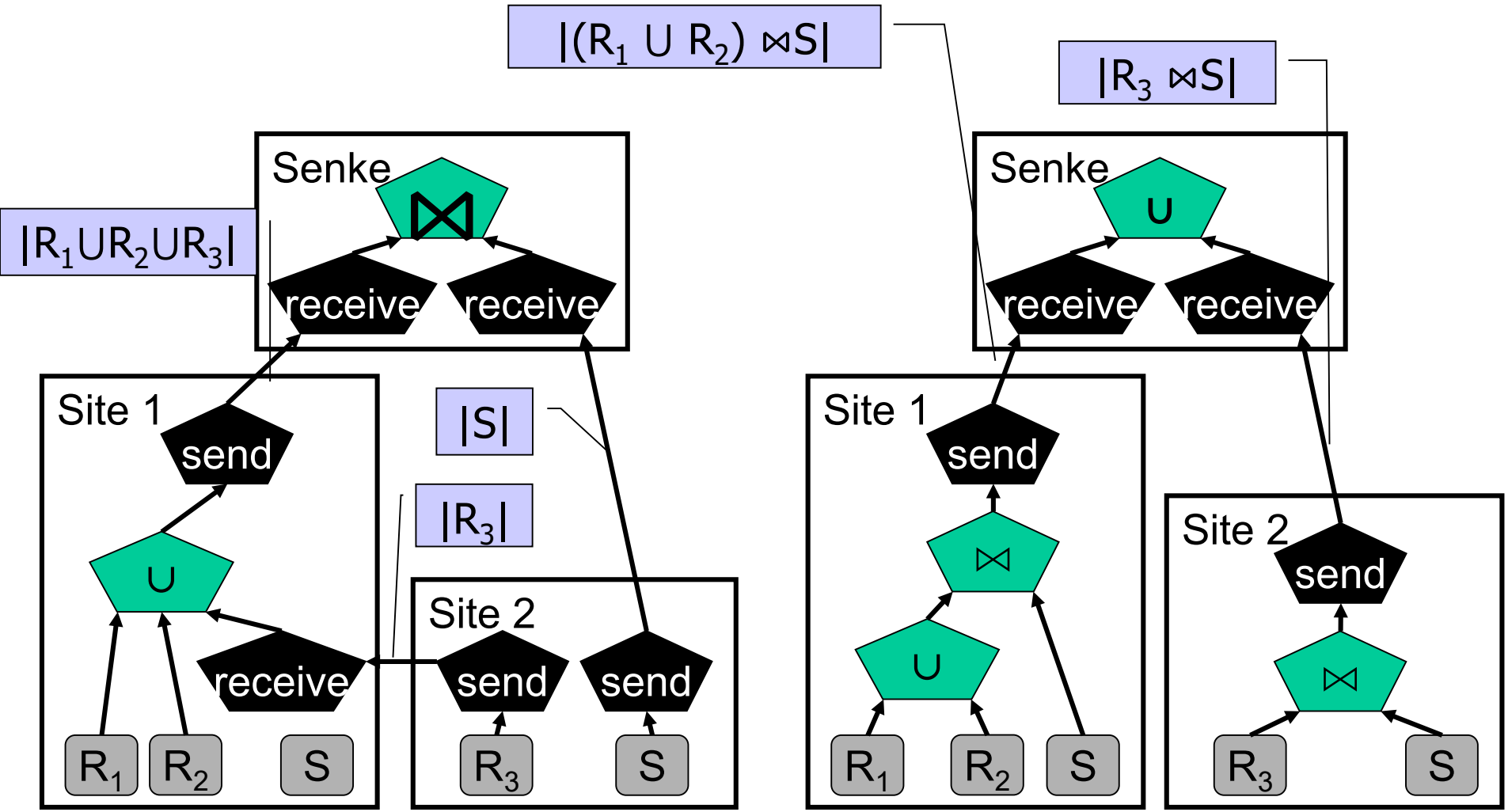
Frage: Wie viele Daten werden übertragen?



Beispiel: $(R_1 \cup R_2 \cup R_3) \bowtie S$

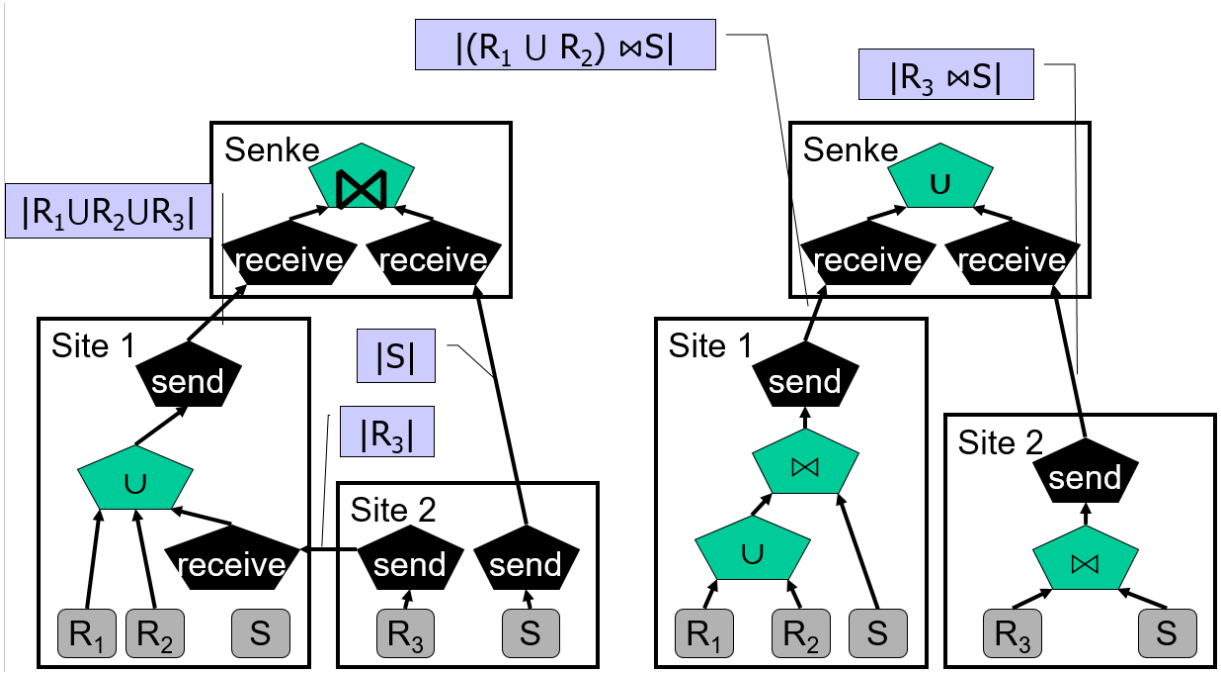


Alternative: $(R_1 \cup R_2 \cup R_3) \bowtie S = ((R_1 \cup R_2) \bowtie S) \cup (R_3 \bowtie S)$



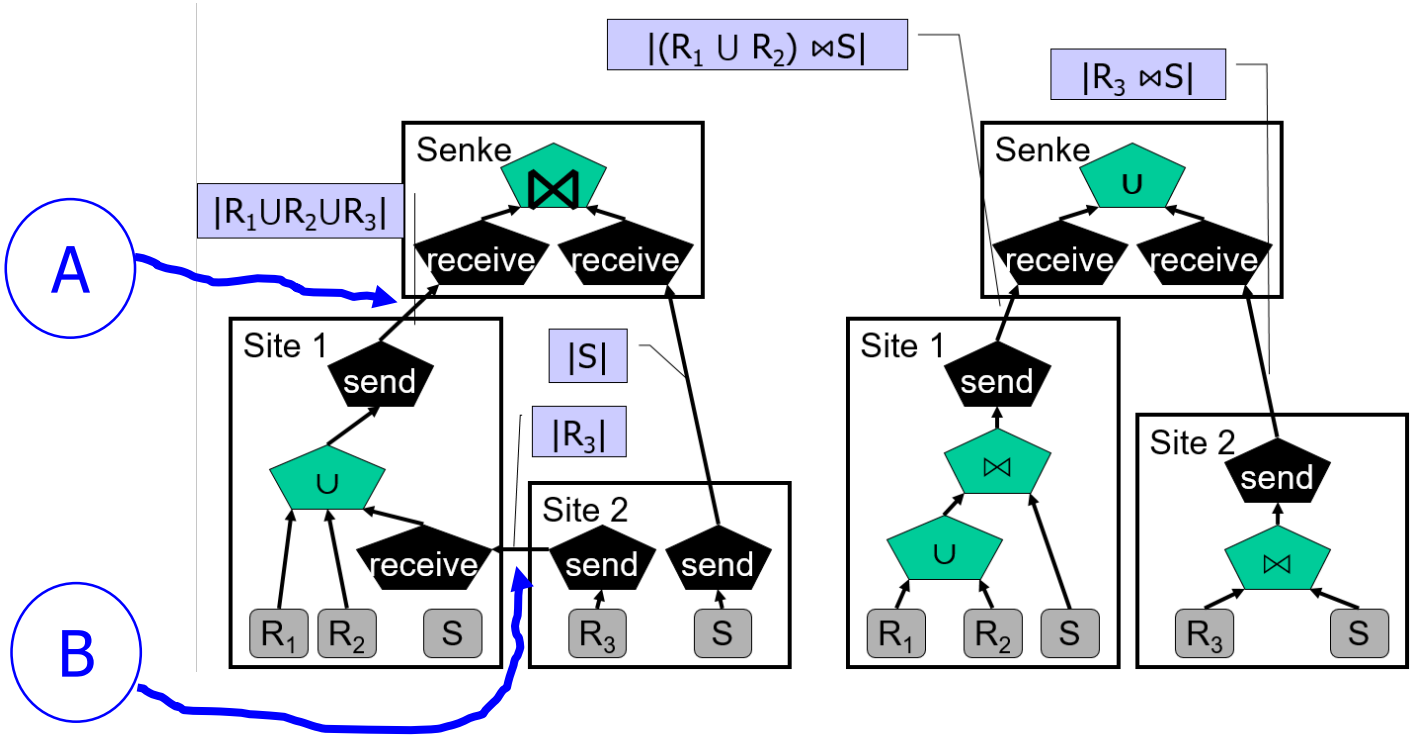
Optimierung auf Gesamt-Datentransfer

Ist $|R_1 \cup R_2 \cup R_3| + |R_3| + |S| > |(R_1 \cup R_2) \bowtie S| + |R_3 \bowtie S|$?



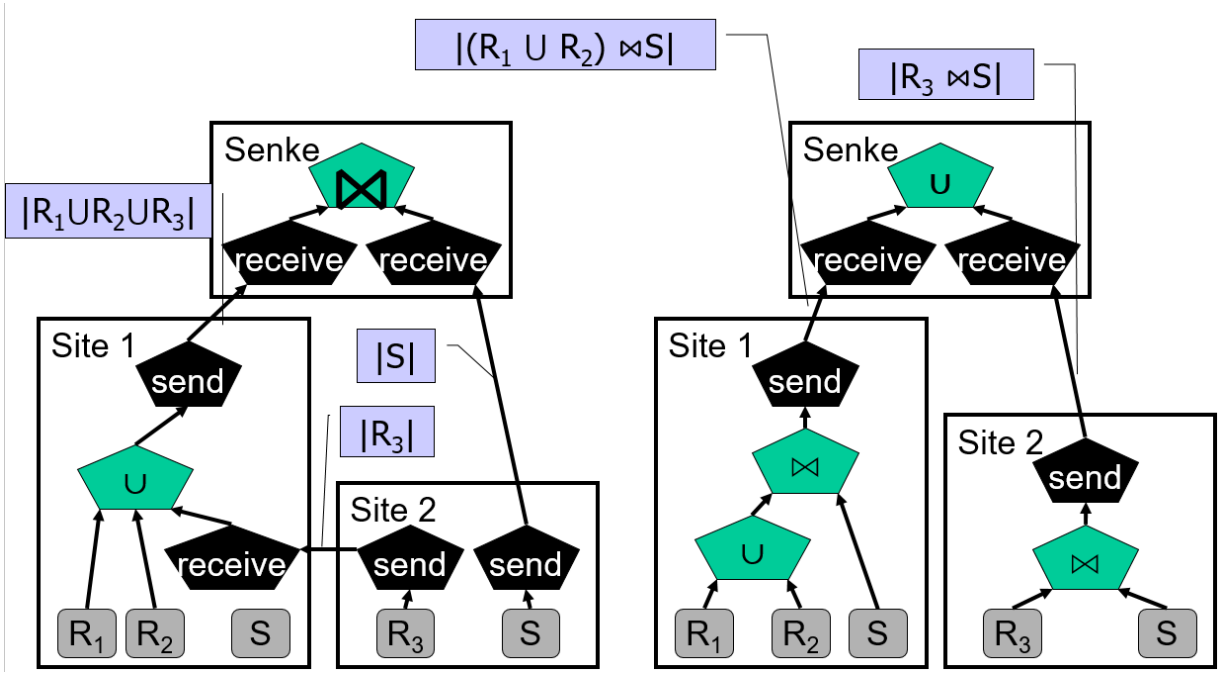
Optimierung auf Gesamt-Übertragungszeit

Dauert der Transport von $|R_1 U R_2 U R_3|$ über Leitung A +
Transport von $|R_3|$ über Leitung B +... länger als ... ?

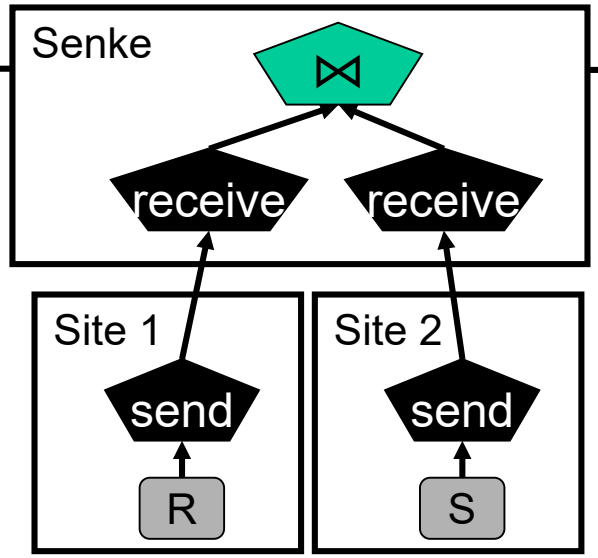


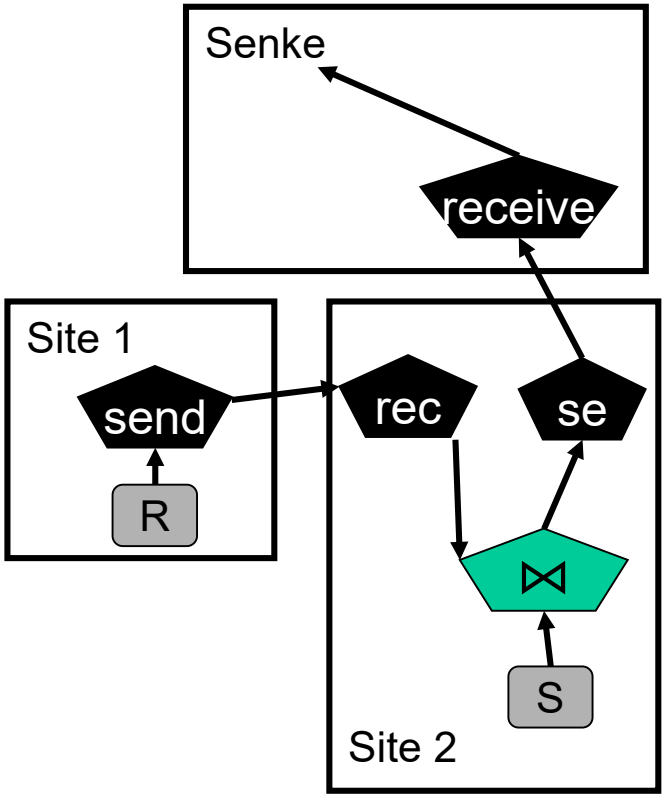
Optimierung auf Wartezeit

Parallel : Ist max(time($R_1 U R_2 U R_3$)+time(R_3)), time(S)) > max(time((R_1 U R_2) ⋈S), time(R_3 ⋈S)) ?



Beispiel: R⋈S

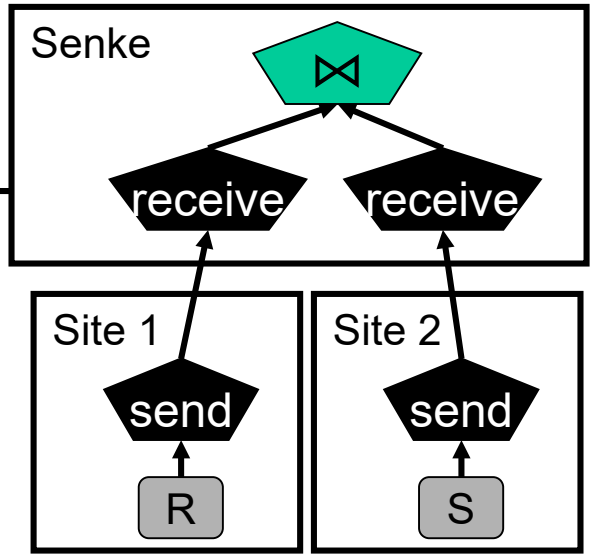


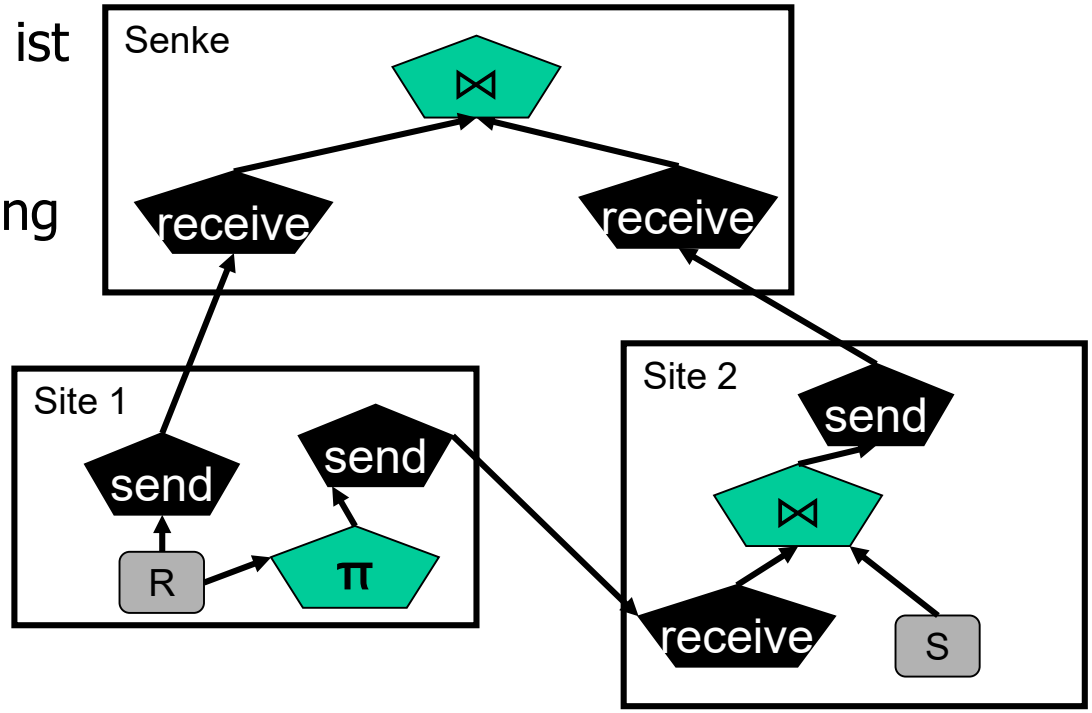


- Alternative: Join pushen
  - Site1 schickt Tupel direkt an Site2
    - Geht natürlich auch umgekehrt
  - Site2 schickt Join-Tupel an Senke, wenn Partner in S vorhanden
- Selektiver Join: Weniger Datentransfer
- Geringe Selektivität: Mehr Datentransfer

Vorschau: Semi-Joins

- Auftrennung: Join-Attribute, andere Attribute
- Semi-Join besser wenn
  - S sehr breit ist
  - Join R, S sehr selektiv ist
- Optimal?
  - Siehe nächste Vorlesung



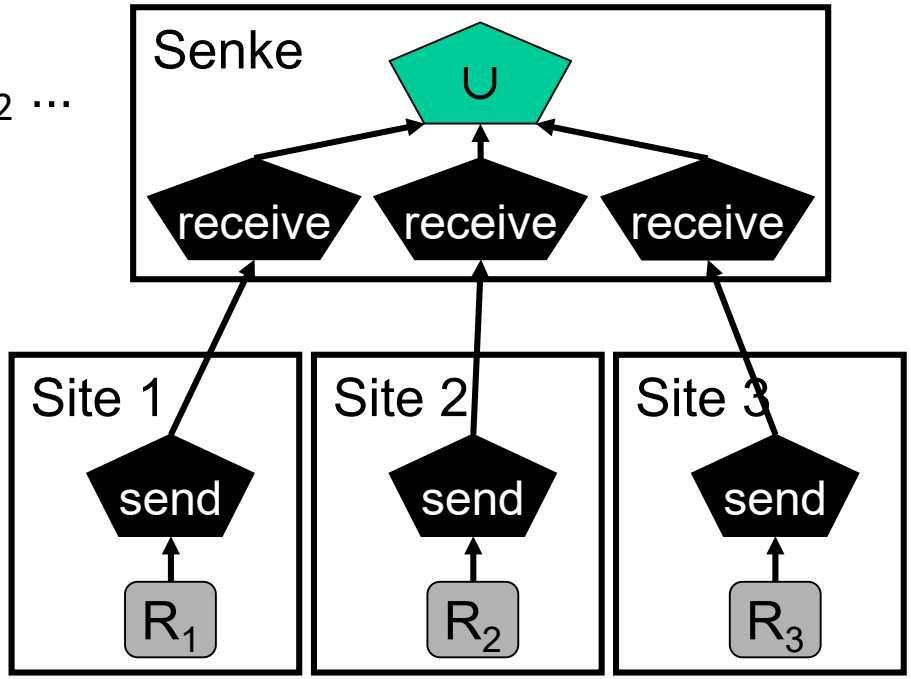


Inhalt dieser Vorlesung

- Optimierungsziele
- Verteilte Anfrageausführung
  - Algebraische Optimierung
  - Parallele Ausführung von Aufrufen
  - Puffern: Row blocking
  - Wer schickt wem was?
  - Caching
- Kostenschätzung

2. Parallele Ausführung Union: R_1 U R_2 U R_3

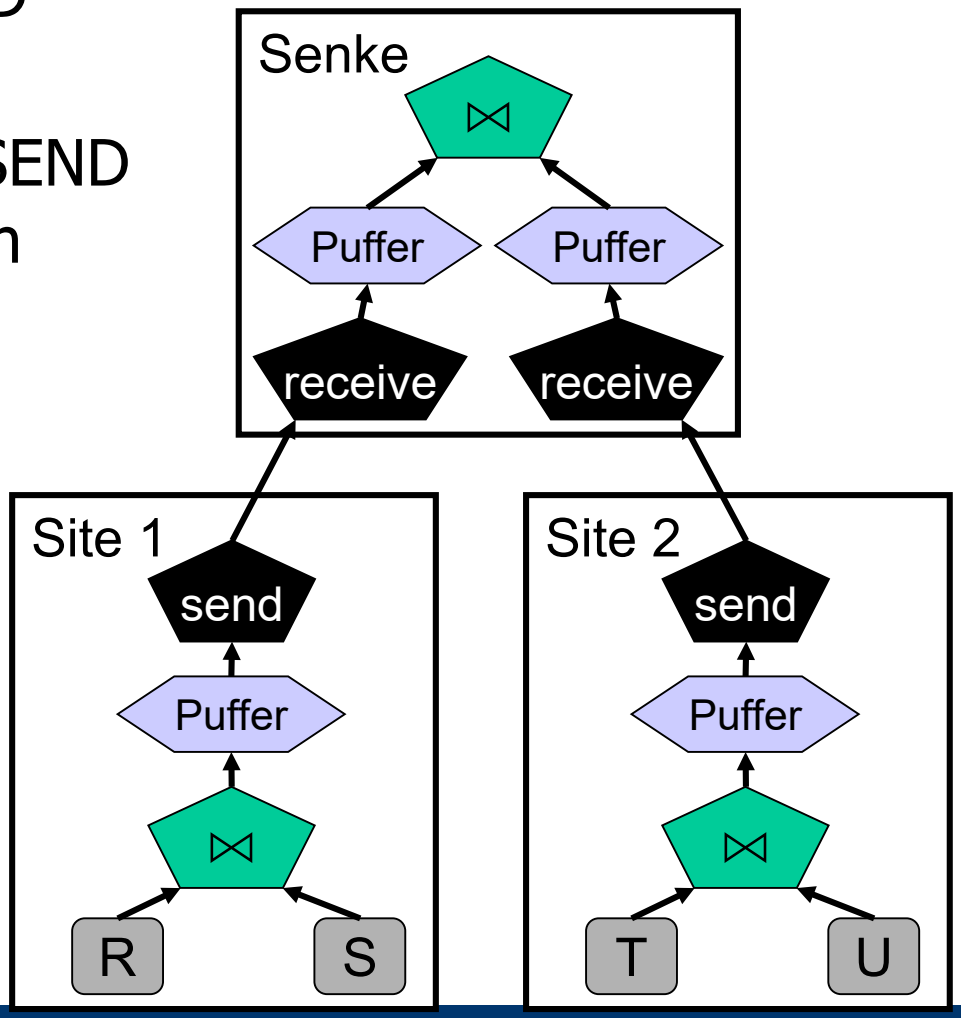
- Seriell
  - Hole alle Daten von R_1, dann R_2 ...
- Parallel
  - Verarbeite einkommende Tupel sofort
  - Benutze spez. Datenstruktur für Online-Union
  - Schneller, weil Quellen parallel liefern können
  - Erlaubt Pipeline-Modus



- Alternative serielle Strategie
  - Round-Robin Bearbeitung der receive Operationen
  - Wenn Tupel langsam kommen, gleichwertig zu multi-threaded

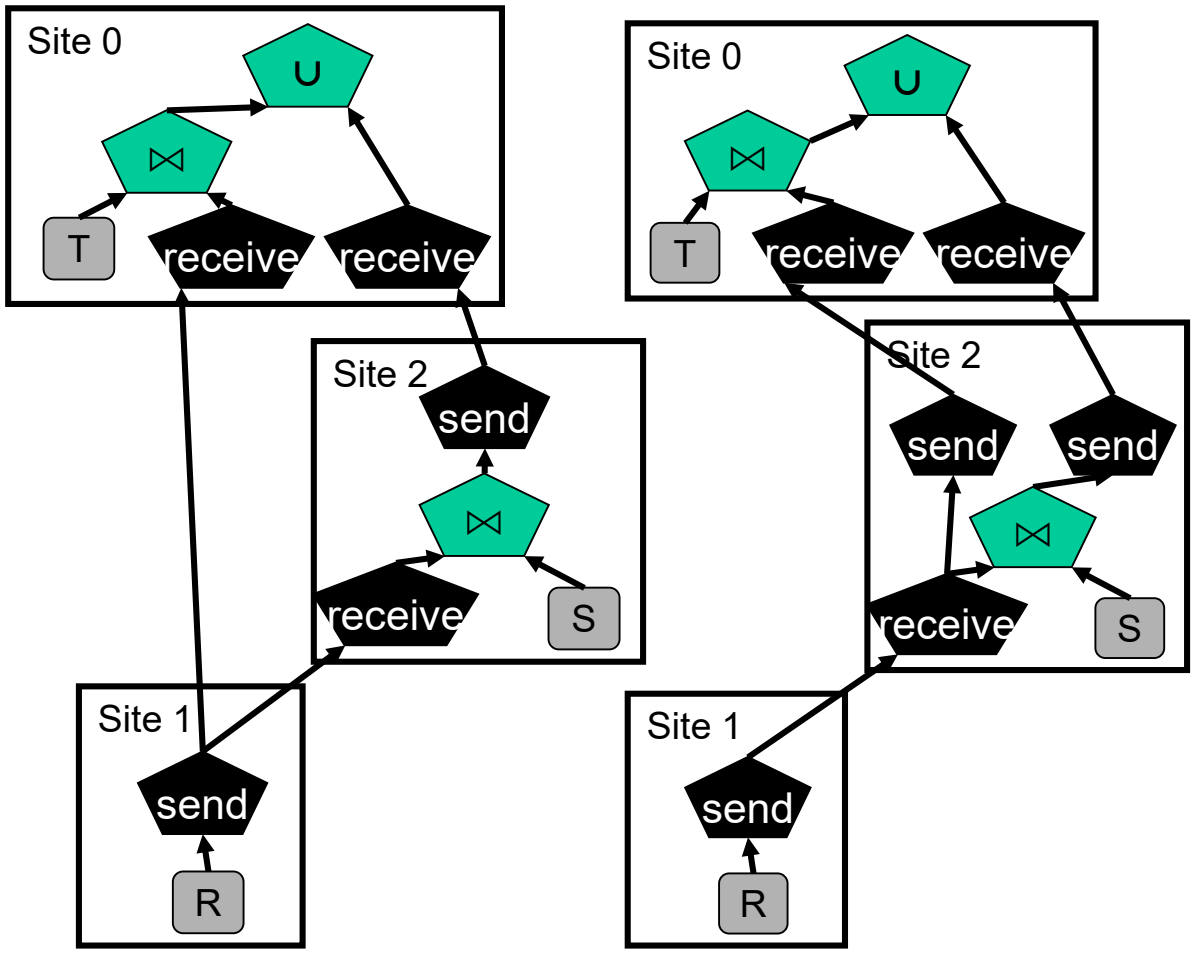
3. Row Blocking: (R⋈S)⋈(T⋈U)

- Datentransfer benötigt SEND und RECEIVE Operatoren
- Naiv: Für jedes Tupel eine SEND und eine RECIEVE Operation
- Besser: Row blocking
  - Tupel werden gesammelt und en bloc versendet
    - Einfügung von Puffern vor/nach Netzwerk-operationen
  - Optimale Blockgröße abhängig von Netzwerk-konfiguration



4. Multicasts versus Unicast: (R⋈S) U (R⋈T)

- Multicast: Im Netzwerk kann man meistens Daten gleichzeitig an viele Empfänger schicken
- Mehr Datentransfer, aber gleiche Laufzeit
  - Vlg. zu mehreren Unicasts



5. Caching

- RDBMS cachen Blöcke
  - Werden als ganzes gelesen oder geschrieben
  - Unabhängig von Anfragen
  - Prefetching, um unnötige Plattenpositionierung zu vermeiden
- Semantisches Caching
  - Anfrage muss zusätzlich zum Ergebnis gespeichert werden
    - Mit den tatsächlichen Variablenbindungen
  - Bevor Anfrage v ausgeführt werden
    - Prüfen, ob ein v‘ existiert mit $v \subseteq v'$ und berechenbarem Unterschied zwischen v und v‘
    - Wenn ja: v nicht ausführen, sondern v aus v‘ berechnen
- Aktualisierung, Verdrängungsstrategie, Prefetching, …
  - Aktualisierung darf nicht auf Benachrichtigung von Quellen hoffen

Inhalt dieser Vorlesung

- Optimierungsziele
- Verteilte Anfrageausführung
- Kostenschätzung

Kostenschätzungen

- Kosten hängen Metadaten ab
  - Selektivität von Prädikaten, Größe von Joins, Dauer Transfer, …
- Basis: Verteilungen der Werte in Quellen / Tabellen
  - Größe und Breite von Basisrelationen
  - Histogramme über Werte eines Attributes
  - Durchschnittliche Längen bei VARCHAR
  - …
- Werden meist geschätzt
  - Basierend auf effizienter Approximation der tatsächlichen Daten
- Besondere Schwierigkeiten in der Informationsintegration
  - Autonome Quellen stellen keine Metadaten zur Verfügung
  - Änderungen in Verteilungen ohne Benachrichtigung des Mediators

Zwei Wege

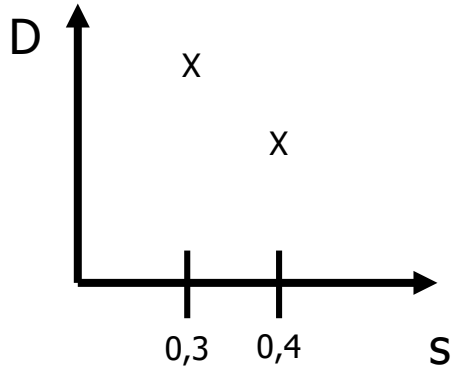
- Offline: Spezielle Anfragen an die Quelle schicken
  - Z.B. Schätzen der Größe und Breite von Relationen
    - „SELECT avg(len(A)) FROM …“, „SELECT COUNT(*) FROM … “
  - Komplettes „Ausmessen“ erzeugt sehr vielen Anfragen – notwendig?
  - Genaue Statistiken: Mächtige Quellen oder Kopieren der Daten
    - E.G. Equal-width Histogramm über Attribute A erfordert sortieren und „binnen“ aller Werte aus A
  - Pro: Genaue, gezielte und aktuelle Werte
  - Contra: Erzeugt also ggf. erheblich zusätzliche Last
  - Gut, wenn Verteilungs-Anfragen die „echten“ Anfragen erahnen
    - Nur messen, was man später braucht
- Inline: Ergebnisse echter Anfragen auswerten

Zwei Wege

- Offline: Spezieller Anfragen an die Quelle schicken
- Inline: Ergebnisse „echter" Anfragen auswerten
  - Berechnet Statistiken nur für tatsächlich verwendete Queries
  - Aktualisiert sich automatisch bei Gebrauch
  - Schätzungen für neue Anfragen fehlen
    - Vorsicht: Man kann von einer komplexeren Query nicht aus Basisrelationen schliessen
    - Wenn ich nur q=R⋈S kennen, kann ich kaum Wissen über q'=R oder R"=R⋈T ableiten
  - Pro: Keine zusätzliche Last
  - Contro: Für viele Anfragen keine Informationen vorhanden
  - Gut bei homogener Workload

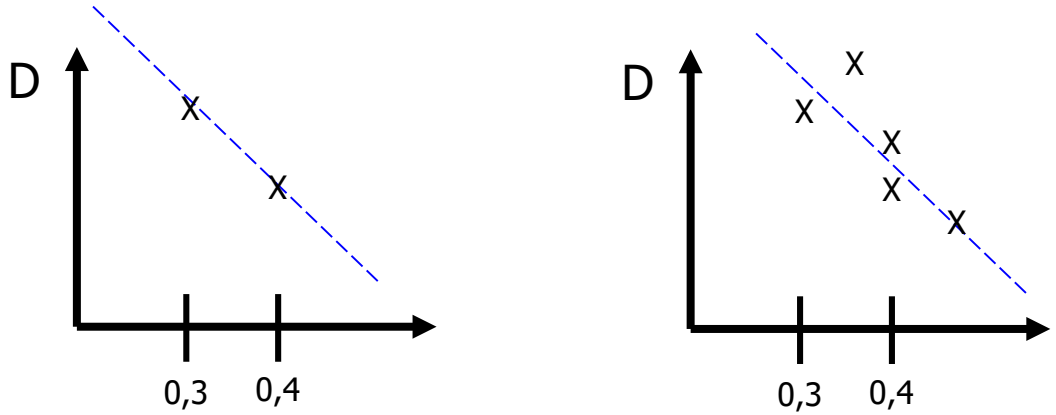
Inter-/ Extrapolation

- Oft muss man inter-/extrapolieren
  - Hilft, um von Queries zu abstrahieren
  - Beispiel: Datenmengen D, D‘ von q=<R,t≥5>, q‘=<R,t≥10> bekannt
  - Ausserdem t_{min} und t_{max}
  - Selektivität schätzen: $s(q) = (t_{max}-t)/t_{max}-t_{min})$
  - Welches D“ erwartet man für q“=<R,t≥15>?



Regression

- Bekannte Methode: Regression
  - Abschätzen einer Funktion D=f(q, s)
  - f abhängig von Query q mit freiem Parameter s
  - Einfachstes Modell: f(s) = a+s*b
  - Bestimmung a,b z.B.: durch Minimierung der quadratischen Fehler
    - Lineare Regression



Komplexere Kostenmodelle

- Einfache lineare Regression ist oft zu einfach / ungenau
- Beispiel für ein komplexeres Modell
  - Ziel: Bearbeitungsdauer einer Anfrage q mit Parameter s schätzen
  - Mögliches Modell: $f(t) = c_0 + c_1 * s * |q| + c_2 * |q|$
    - c_0: Verbindungsaufbau, Latenz
    - c_1: Übertragung der Tupel
    - c_2: Berechnung der Tupel in der Quelle
  - Auch hier: Schätzen von c_0, c_1, c_2 durch Regression
- Problem: Parameter sind spezifisch für genau eine Query
  - Gut bei homogener Workload mit wechselnden Selektivitäten

Literatur

- [OV99] Oezsu, M. T. and Valduriez, P. (1999). "Principles of Distributed Database Systems". New Jersey, Prentice Hall, Inc.
- [NLF99] Naumann, F., Leser, U. and Freytag, J. C. (1999). "Quality-driven Integration of Heterogeneous Information Systems". 25th Conference on Very Large Database Systems, Edinburgh, UK. pp 447-458.
- [Kos00] Kossmann, D. (2000). "The State of the Art in Distributed Query Processing." *ACM Computing Surveys 32(4): 422-269.*

Appendix

- Quality-Based Query Optimization [NLF99]
  - Or: What is a "good" plan?

Problem in Query Planning

- Trade-Off
  - Many plans need a very long time
  - Many plans do not add (much) new (better) information

Who needs all plans?

Information / Data Quality

- Data quality is a major issue in IT operations
  - A.T. Kearny: 25%-40% of IT costs root in bad data quality
  - SAS: Only 18% of German enterprises trust their own data
  - …
- Low data quality is difficult to detect and measure
  - Wrong data, missing data, outdated data, biased data, …
- Data quality is a major issue in information integration
  - Autonomous sources: Quality cannot be improved
  - Heterogeneity leads to errors (wrong correspondences …)
  - Network adds instability
- Eventually, integration is all about improving quality: More complete, better results

Aspects of Data Quality

- Definition: "Fitness for use"
- Many many different aspects (dimensions)
  - IQ = { Understandability, (Verstehe ich die Daten?)
    Reputation, (Herkunft vertrauenswürdig?)
    Reliability, (Daten verlässlich?)
    Timeliness, (Aktualität?)
    Availability, (Wann und wie oft verfügbar?)
    Consistency, (In sich konsistent?)
    Response time, (Wie schnell kriege ich Antwort?)
    Density, (% der Felder sind gefüllt?)
    Completeness, (Alle Informationen da?)
    Amount, (Zahl an Ergebnisse?)
    Accuracy, (Genauigkeit der Daten?)
    Relevancy, (Nur relevante Daten?)
    ...

Quality Based Query Optimization

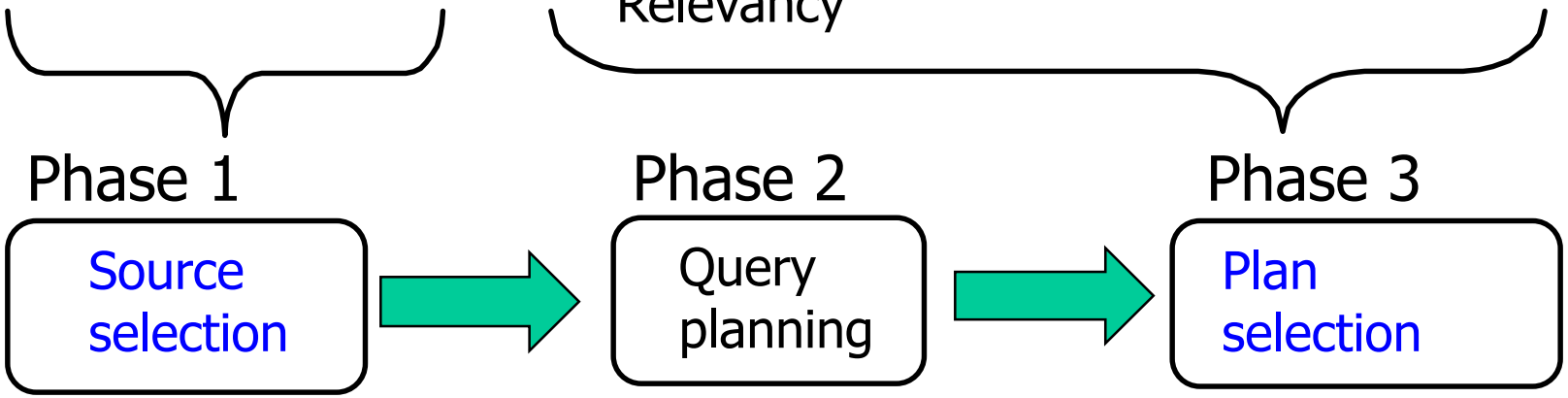
- Idea: Treat quality as cost
  - But: Quality has multiple dimensions
  - But: Quality doesn't easily sum-up
  - But: Quality affects multiple entities (e.g. attributes, sources, …)
- Idea: Annotate correspondences with quality vectors
- Aggregate quality vectors for obtain quality of a plan
- Use to prune / select plans

IQ criteria can be ...

Source-specific
- Understandability
- Reputation
- Reliability
- Timeliness

Corresp.-specific
- Availability
- Price
- Repr. consistency
- Response time
- Accuracy
- Relevancy

Attribute-specific
- Completeness
- Amount



IQ scores (example)

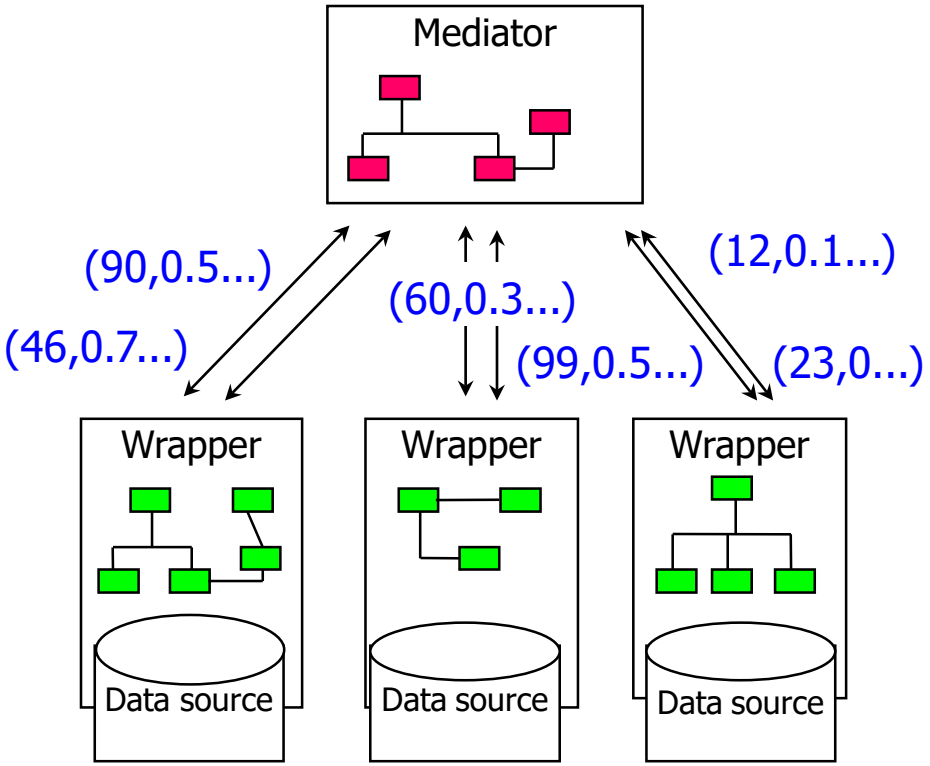
IQ(view)=(Un,Rep,Rel,Ti, Av,Pr,RC,RT,Ac,Re);

	S_1 v_1	S_2 v_2	S_3 v_3	S_4 rA_4	QCA_5	S_5 QCA_6	QCA_7
Underst.	5	7	7	8		6	
Reputation	5	5	7	8		7	
Reliability	2	6	4	6		6	
Timeliness	30	30	2	1		7	
Availability	99	99	60	80	99	95	95
Price	0	0	0	0	1	0	0
R.Consist.	1	1	0.5	0.7	0.2	0.7	0.7
Resp. Time	0.2	0.2	0.2	3	0.1	1	1
Accuracy	99.9	99.9	99.8	99.95	99.95	99.95	99.95
Relevancy	60	80	90	80	80	60	60

Unit:
rank
rank
rank
days
%
USD
effort
sec.
%
%

Phase 1. Preparation

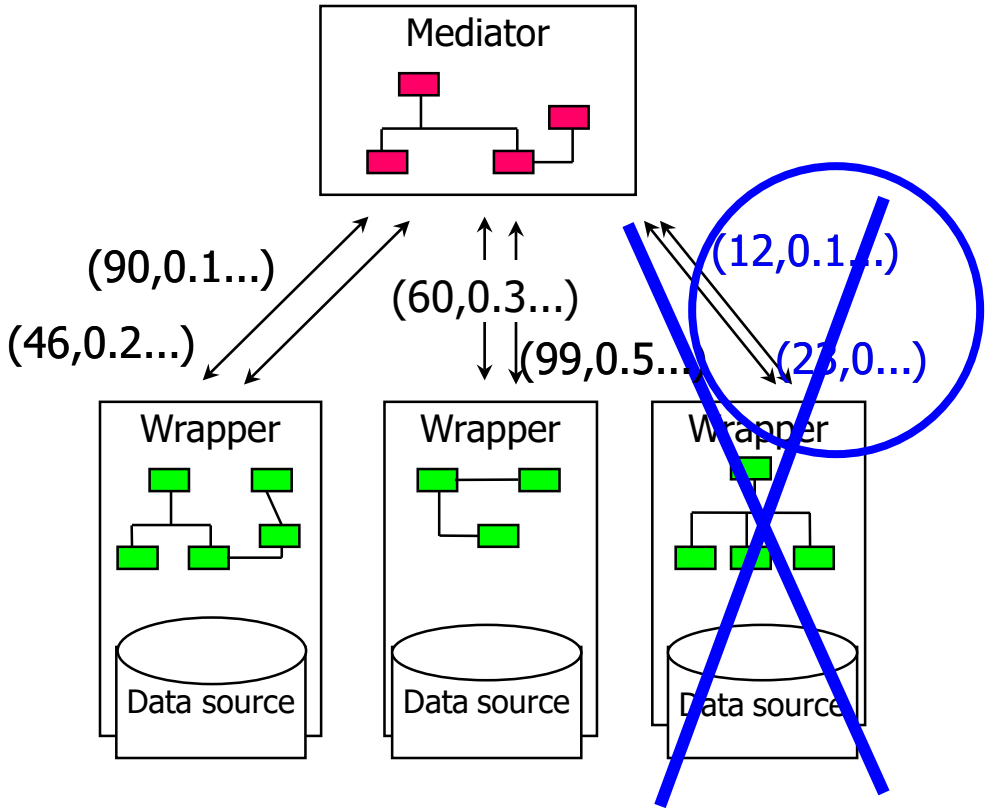
Annotate correspondences with information quality



Phase 2. Source selection

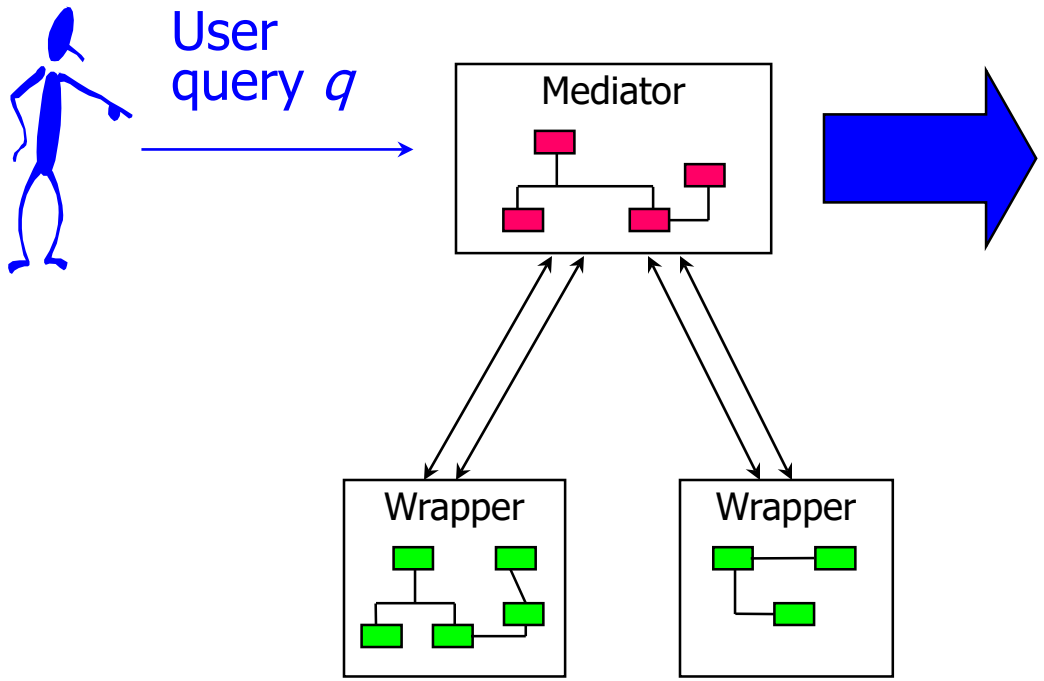
Use IQ vectors to remove the worst (dominated) sources

Use Only source specific IQ criteria



Phase 3. Query planning

Compute all plans



$p_1 := q_1, q_2, q_3, q_4, \ldots$

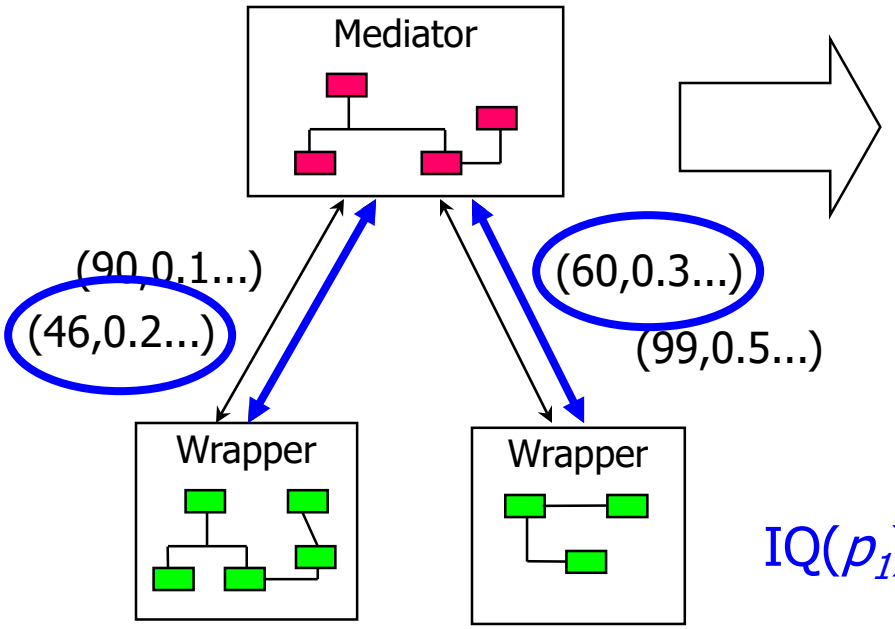
$p_2 := q_3, q_5, q_6, q_7, \ldots$

$p_3 := q_1, q_3, q_5, q_9, \ldots$

etc.

Phase 4. Plan selection

Use IQ scores to find the best plans



$p_1 := q_1, q_2, q_3, q_4, \ldots$ 3

$p_2 := q_3, q_5, q_6, q_7, \ldots$ 1

$p_3 := q_1, q_3, q_5, q_9, \ldots$ 2

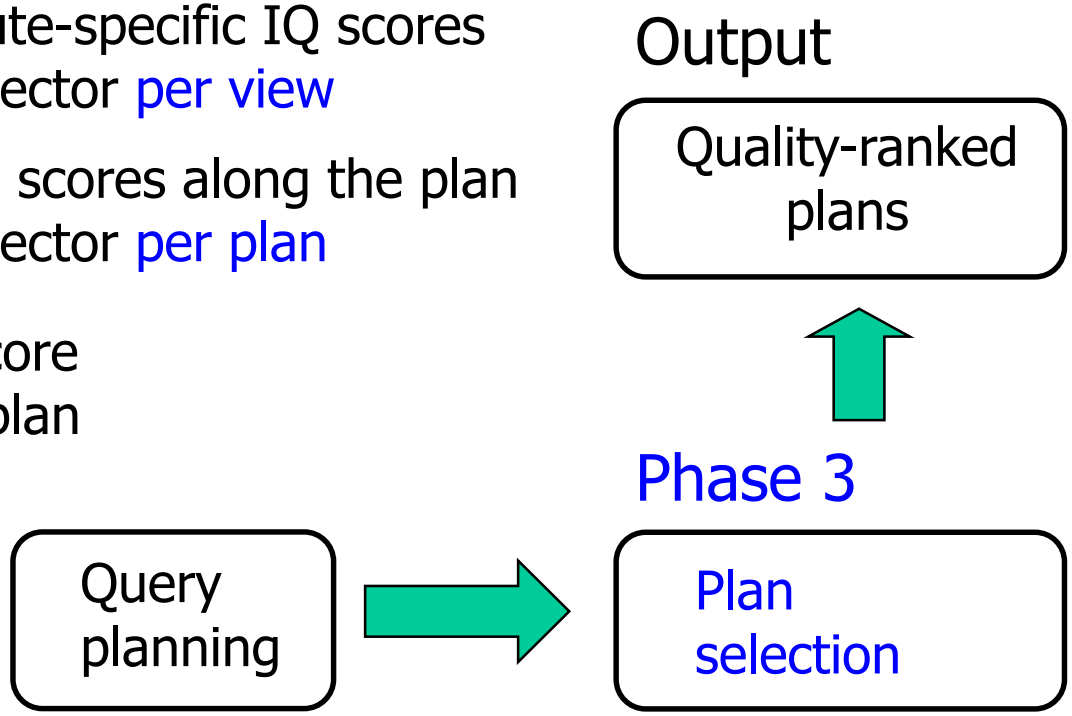
etc.

$IQ(p_1) = (60,0.3\ldots) \bowtie (46,0.2\ldots) \bowtie \ldots$

Steps to determine Plan Quality Vectors

Three steps for each plan

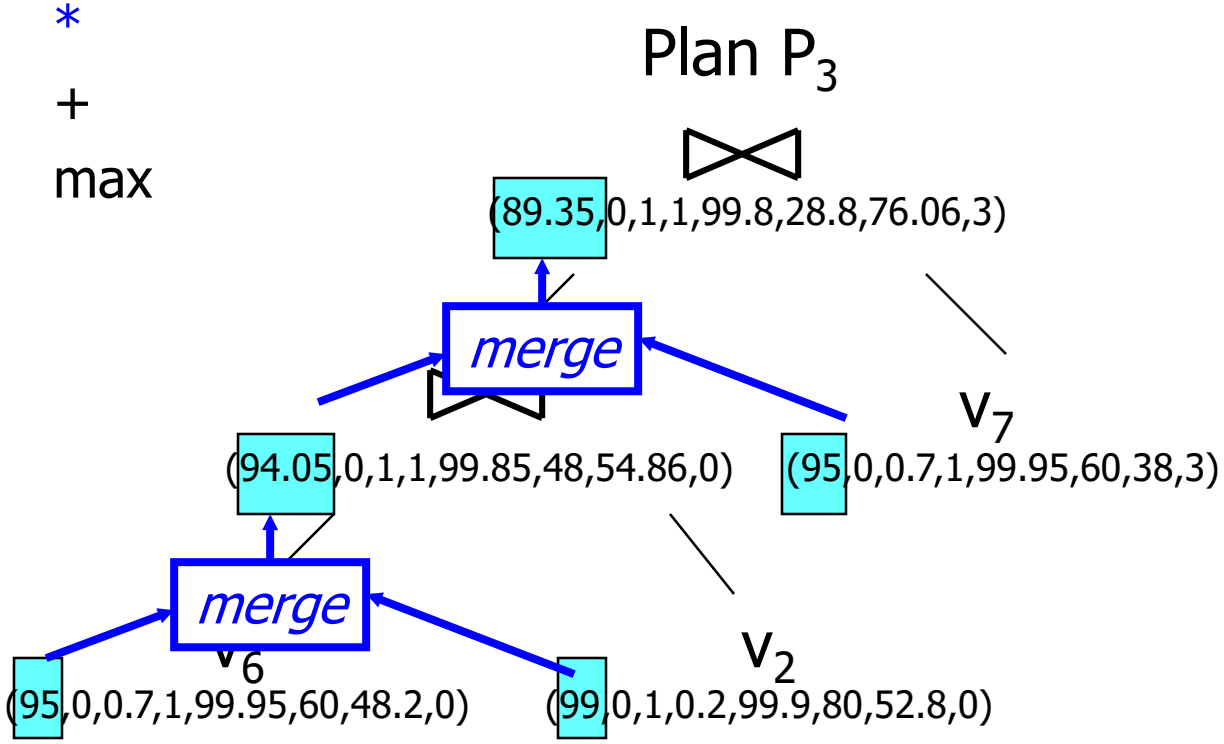
1. Determine attribute-specific IQ scores
   ⇒ Complete IQ vector per view
2. Aggregate the IQ scores along the plan
   ⇒ Complete IQ vector per plan
3. Find overall IQ score
   ⇒ IQ scalar per plan



Step 2: Aggregate IQ Vectors

Merge functions:

- Availability: *
- Price: +
- Response time: max
- …



Step 3: From Quality Vectors to Quality Scalars

Simple Additive Weighting (SAW)

P_3: (89.35,0,1,1,99.8,28.8,76.06,3)

Normalize

$$v_{ij} = \frac{d_{ij} - d_j^{\min}}{d_j^{\max} - d_j^{\min}} \qquad v_{ij} = \frac{d_j^{\max} - d_{ij}}{d_j^{\max} - d_j^{\min}}$$

P_3: (1, 1, 0, 1, 1, 0.62, 0.51, 1)

Aggregate

$$\mathrm{IQ}(P_i) = \sum_{j=1}^{8} w_j v_{ij}$$

$IQ(P_3) = 0.77$