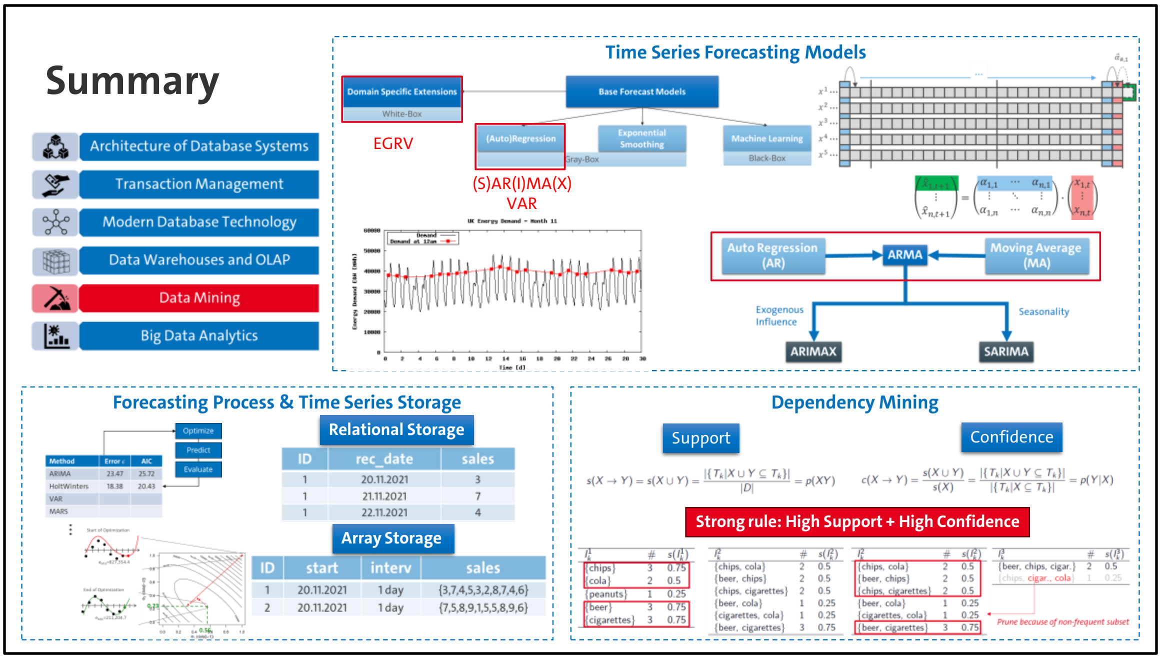


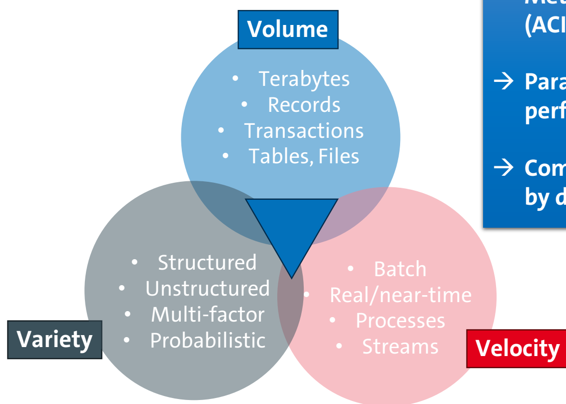
Summary
Architecture of Database Systems
Transaction Management
Modern Database Technology
Data Warehouses and OLAP
Data Mining
Big Data Analytics
Time Series Forecasting Models
Domain Specific Extensions
White-Box
Base Forecast Models
(Auto)Regression
Exponential Smoothing
Gray-Box
Machine Learning
Black-Box
EGRV
(S)AR(I)MA(X)
VAR
UK Energy Demand - Month 11
Auto Regression (AR)
ARMA
Moving Average (MA)
Exogenous Influence
Seasonality
ARIMAX
SARIMA
Forecasting Process & Time Series Storage
Optimize
Predict
Evaluate
Method
Error
AIC
ARIMA
23.47
25.72
HoltWinters
18.38
20.43
VAR
MARS
Relational Storage
ID
rec_date
sales
1
20.11.2021
3
1
21.11.2021
7
1
22.11.2021
4
Array Storage
ID
start
interv
sales
1
20.11.2021
1 day
{3,7,4,5,3,2,8,7,4,6}
2
20.11.2021
1 day
{7,5,8,9,1,5,5,8,9,6}
Dependency Mining
Support
Confidence
$s(X \rightarrow Y) = s(X \cup Y) = \frac{|\{T_k | X \cup Y \subseteq T_k\}|}{|D|} = p(XY)$
$c(X \rightarrow Y) = \frac{s(X \cup Y)}{s(X)} = \frac{|\{T_k | X \cup Y \subseteq T_k\}|}{|\{T_k | X \subseteq T_k\}|} = p(Y|X)$
Strong rule: High Support + High Confidence
Prune because of non-frequent subset

Course Outline

- Architecture of Database Systems
- Transaction Management
- Modern Database Technology
- Data Warehouses and OLAP
- Data Mining
- Big Data Analytics

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

The Big Data Vs



→ see lecture 10

→ see lectures 10-15

- → New Data Management Methods and Systems (ACID→BASE, NoSQL,…)
- → Parallelization to meet performance requirements — Today
- → Complexities of scaling hidden by data processing frameworks — Wednesday

More Vs:

- *Veracity*: Is your data (source) trustworthy/ meaningful?
- *Visualization*: How to communicate? insights& knowledge?
- *Value*: How to use data for (machine) learning, optimization, …?
- (Volatility, Vulnerability, Validity, …)

Recap Query Processing

SQL Query (What!)

```
SELECT A_Name, SUM(S_Qty)
FROM Article, Sales
WHERE A_Anr = S_Anr AND
      S_Date >= '2021-01-01'
GROUP BY A_Name
```

YES, but HOW do we get there (efficiently)?

Query Plan (How!)

Result

A_Name	SUM
Article A	12
Acticle B	1

$\gamma_{SUM(S\_Qty);\, A\_Name}$

$\sigma_{S\_Date >= \text{'2021-01-01'}}$

$\bowtie_{S\_Anr=A\_Anr}$

Sales Article

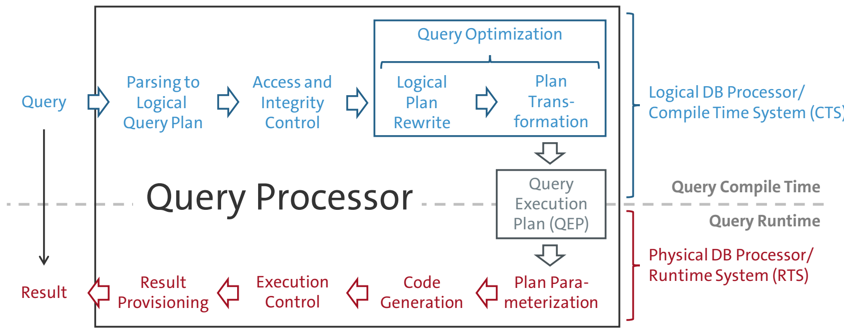
Sales

...	S_Anr	S_Date	S_Qty
...	1	2020-09-20	7
...	1	2021-01-17	2
...	1	2021-02-21	5
...	2	2021-02-22	1
...	1	2021-03-07	5

Article

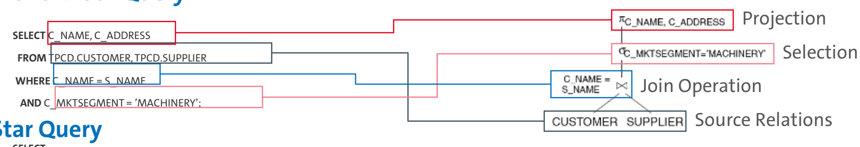
A_Anr	A_Name	...
1	Article A	...
2	Acticle B	...

Query Processing

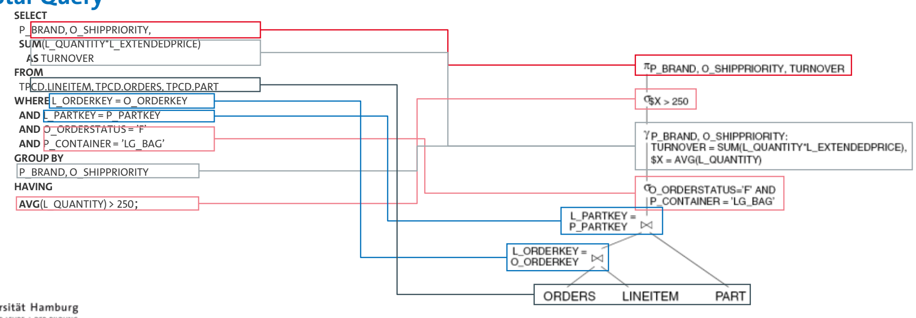


Logical Query Plan

Mono-Block Query



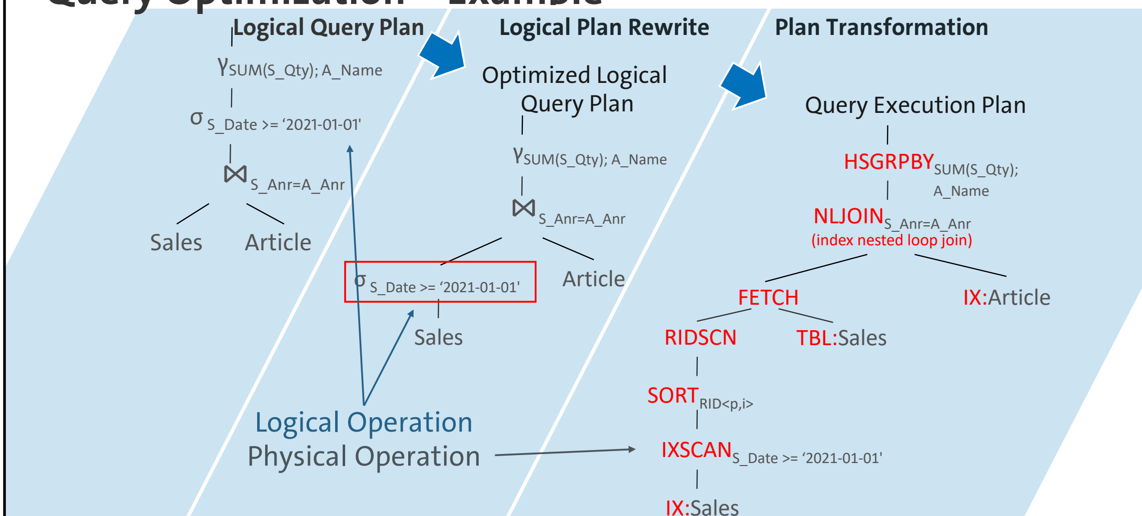
Star Query



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

→ *see lectures 2 and 9*

Query Optimization – Example



Parallel Execution Opportunities

Query Perspective	Operator Perspective

Inter-query parallelism → Important for OLTP

- Parallel execution of multiple queries
- Goal: increase throughput (e.g., 100.000 queries / second better than 10.000 queries / second)

Inter-operator parallelism

- Parallel execution of multiple operators
- Goal: increase operator throughput, but limited through operator dependencies

→ See lectures 7 and 8

Intra-query parallelism → Important for OLAP

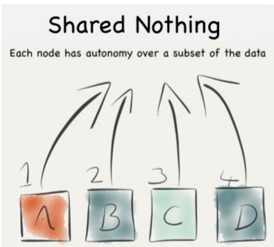
- Parallel execution of a single query
- Goal: decrease response time (10s better than 100s runtime)

Intra—operator parallelism

- Parallel execution of a single operator
- Goal: decrease operator run time

Scale-Out Architecture and Parallelization

→ see lecture 8

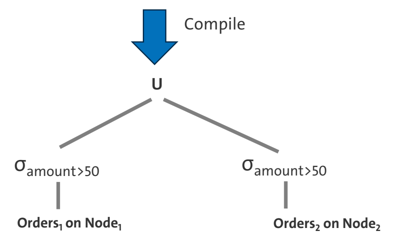


Partition-parallelism can be used for many relational operators:

- sort
- aggregation
- join
- …

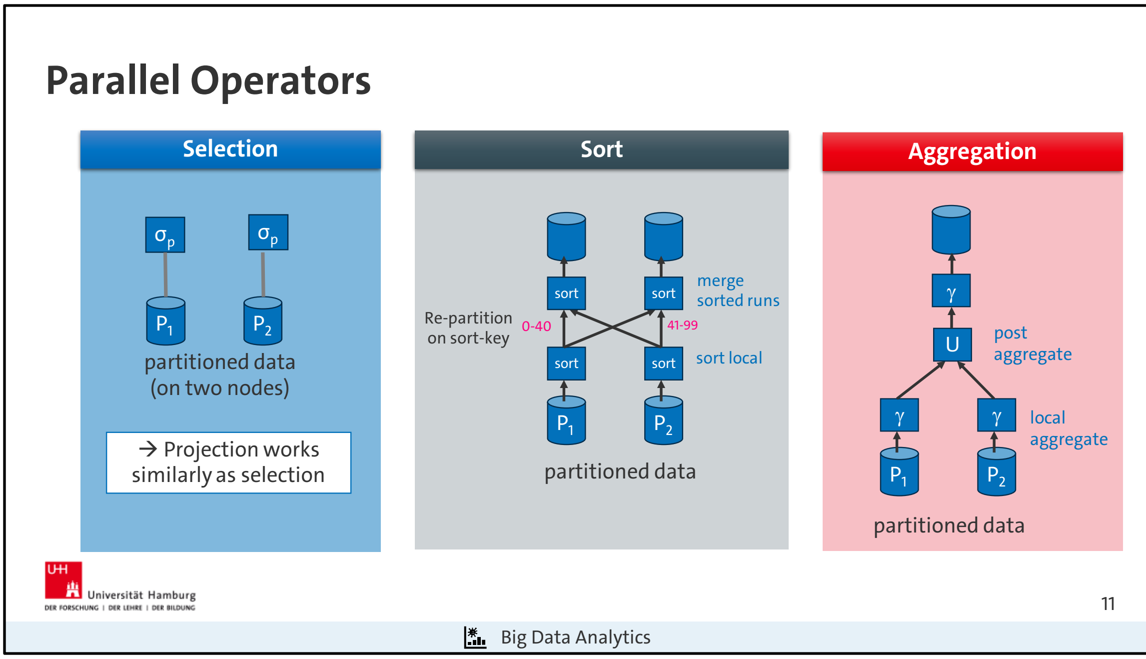
Some are trivial to parallelize: selection, projection

```
SELECT *
FROM orders
WHERE amount>50
```



6/1/17	424252	Couch	$570
6/1/17	256623	Car	$1123
6/2/17	636353	Bike	$86
6/5/17	662113	Chair	$10

6/7/17	121113	Lamp	$19
6/9/17	887734	Bike	$56
6/11/17	252111	Scooter	$18
6/11/17	116458	Hammer	$8000



Selection

- Execute selection σ_p in parallel on each fragment
  → Filter out data based on some predicate p (e.g., age=30)
- Pruning can be used for parallel selection
  - Query asks for customers with age>40
  - Customer table is range partitioned into two fragments “age<30 and age>=30”
  - Query can prune fragment with age<30

Sort

1. Sort locally (e.g., on age)
2. Re-partition on sort-key (aka “shuffling”) using range-partitioning
3. Sort re-partitioned data (can use merge-sort!)

Aggregation

<u>Main idea</u>

- Aggregate locally
- Then re-partition using N:1 shuffle and post aggregate

- Pictured plan works for SUM, MIN, MAX
  - COUNT to be re-written to use SUM for post-aggregation
  - AVG needs to be re-written to use SUM / COUNT locally
  - COUNT DISTINCT can only eliminate duplicates locally

Example: Parallel Sort

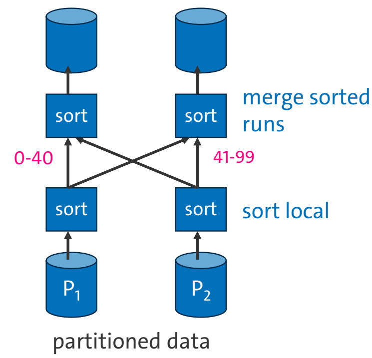
Example: Sort-by Age Ascending

Name	Age
Adler	22
Muller	38

Re-partition on sort-key

Name	Age
Muller	38
Bishop	57

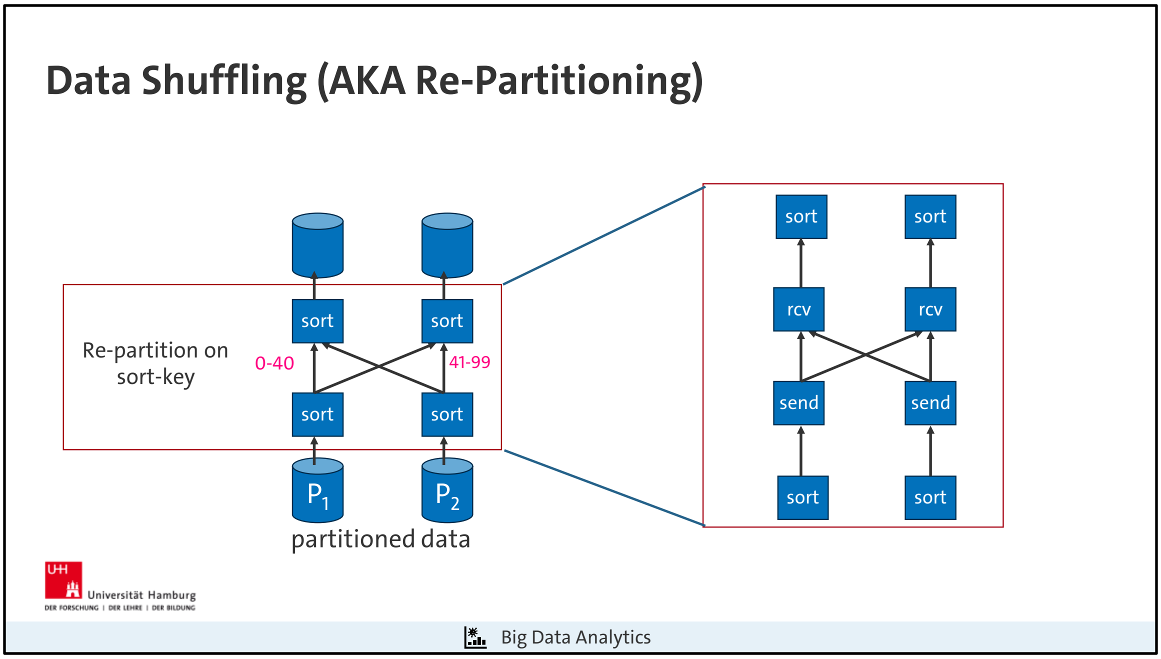
Name	Age
Bishop	57
Muller	38



Name	Age
Zeller	47
Bishop	57

Name	Age
Adler	22
Zeller	47

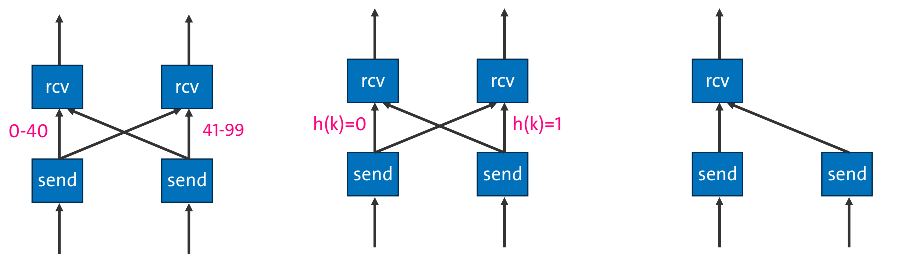
Name	Age
Zeller	47
Adler	22



- Data shuffling is implemented as a separate operator (send-receive)
- Sort operators are un-aware of parallel (distributed) execution

Data Shuffling: Details

Data Shuffling can use different partitioning strategies (range vs. hash-partitioning, N:M vs. N:1) to re-partition data during query execution



Range-based N:M **Hash-based N:M** **N:1 (no part. function)**

Not only parallel sort uses data shuffling but also join, aggregation, …

Example: Parallel Aggregation

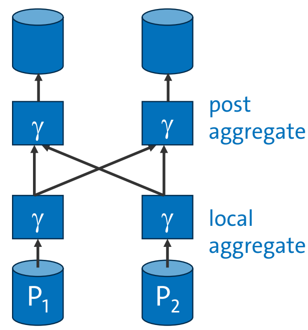
Example: SELECT SUM(Salary) FROM Census GROUP BY Gender

Gender	Salary
m	107k

re-partition on
group-key

Gender	Salary
m	95k

Gender	Salary
m	38k
m	57k



partitioned data

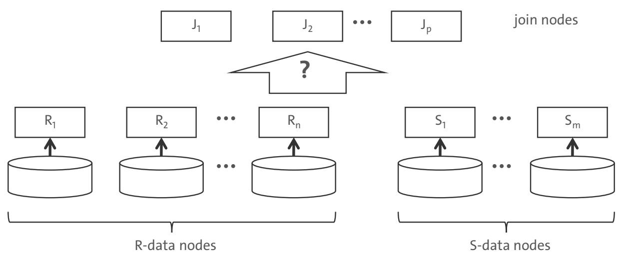
Gender	Salary
f	47k

Gender	Salary
f	47k
m	22k

Gender	Salary
f	47k
m	22k

Parallel Join

Scenario
- Equality join between R and S
  - $R = \cup (R_1, R_2, \ldots, R_n)$
  - $S = \cup (S_1, S_2, \ldots, S_m)$
- |S| is less than |R|
- Join computation on p join processors



- Joins are the most expensive distributed operation
  - might need to shuffle large intermediate results

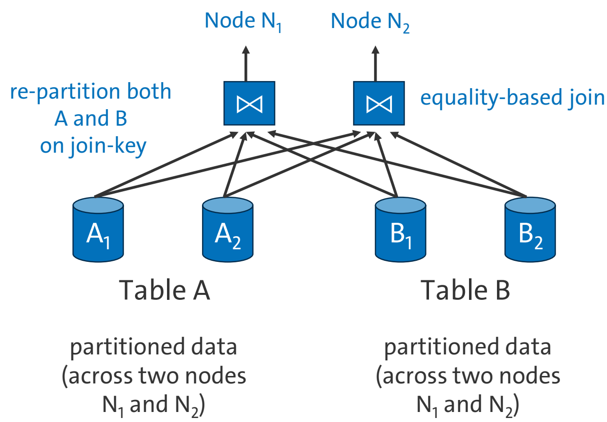
- Different Strategies for Parallel Execution
  - symmetric repartitioning
    (synonym: (fully) re-partitioned join)
  - asymmetric re-partitioning
    (synonym: one-way re-partitioned join, directed join)
  - fragment and replicate
    (synonym: broadcast join)

Join: Symmetric Repartitioning

„Repartitioned join“

- Both join partners are repartitioned after the join attribute
- High communication costs

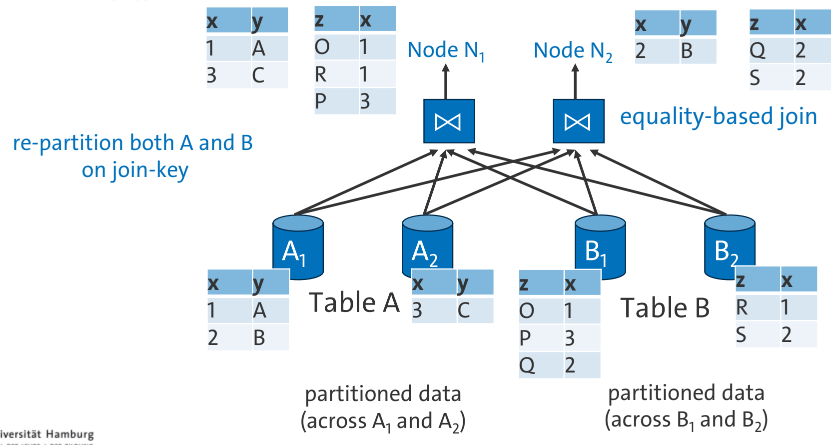
  → Avoid!



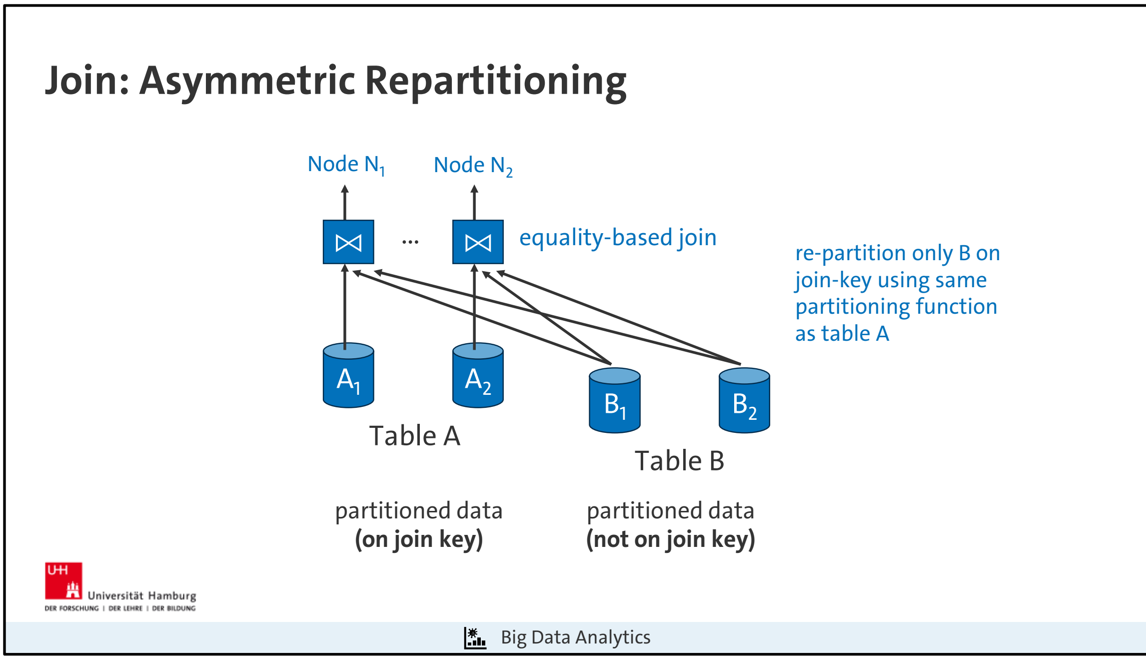
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Join: Symmetric Repartitioning

Example: A $\bowtie_{A.x=B.x}$ B



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

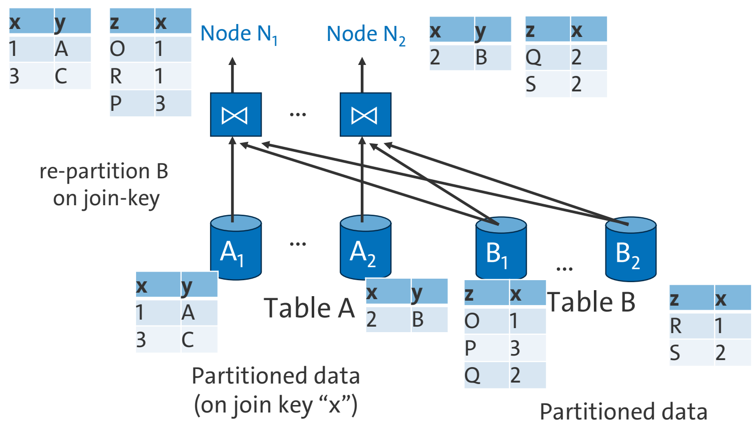


One-way redistribution join "directed join“

- One of the two join partners is partitioned after the join attribute
- Partitions of the other join partner are partitioned newly at runtime after the join attribute
- Example:
    - Order relation is partitioned according to the customer key
    - Repartitioning by the attribute O_ORDERKEY

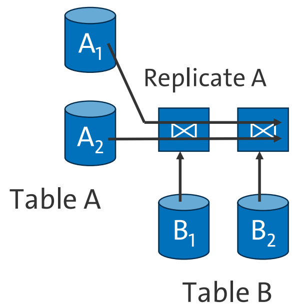
Example: Asymmetric Repartitioning

Example: A $\bowtie_{A.x=B.x}$ B



Join: Fragment and Replicate

partitioned data (across two nodes A_1 and A_2)



Replicate (all fragments of) smaller table (here: table A) to all nodes

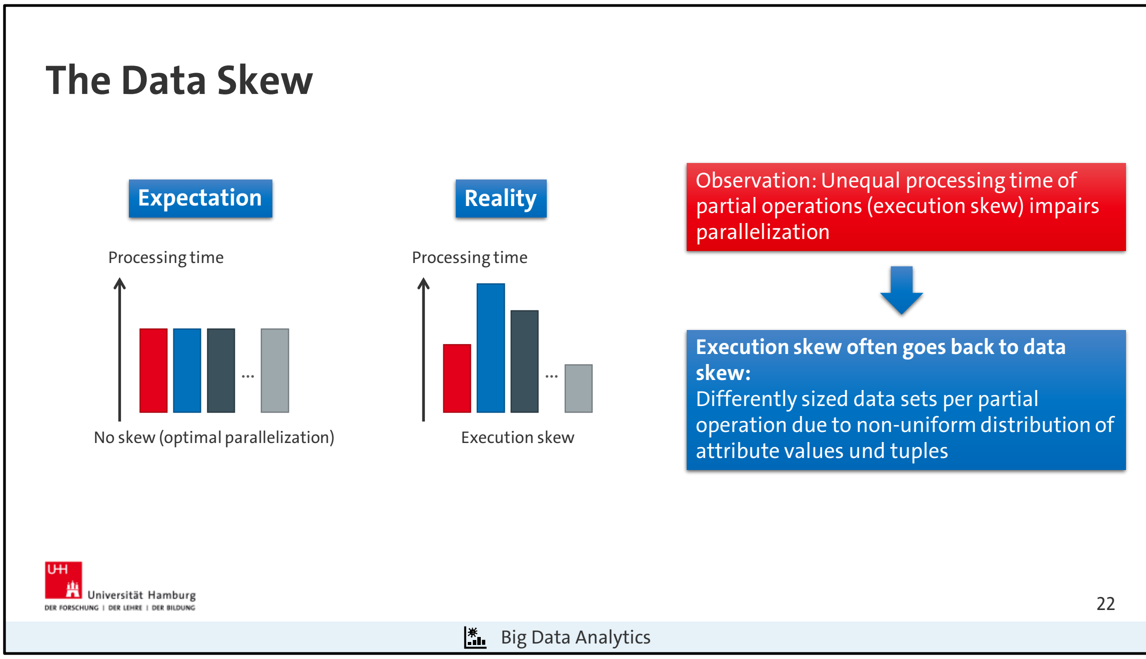
partitioned data (across two nodes B_1 and B_2)

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Broadcast Join

- Partitioning with small relations is not worth it. …
- Assign a copy of the smaller join partner to the partitions of the larger join partner

  → Advantage: no relation must be partitioned after the join attribute



Data Distribution Skew (tuple placement skew)

- Different partition sizes
- Uneven duration of scanning operations
- Treatment: Best knowledge of the distribution of values for distribution attributes
    - Histograms
    - Sampling
    - Computed during sorting on sort merge joins

Redistribution Skew

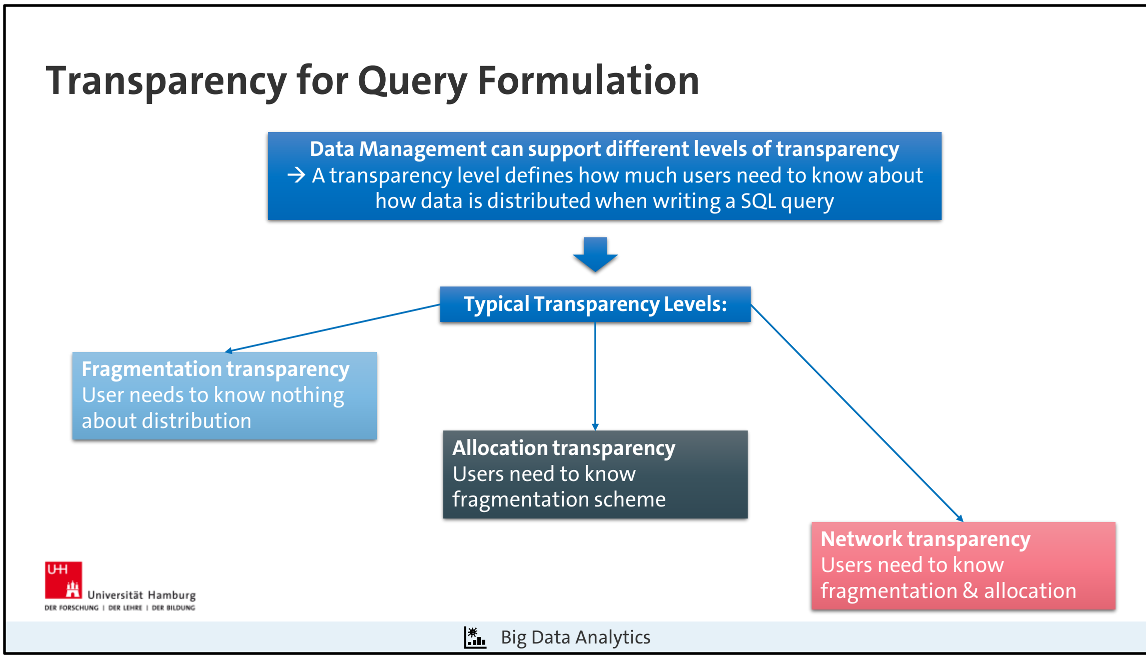
- Distribution function leads to different fragment sizes
- Treatment: Same as data distribution skew

Selectivity Skew

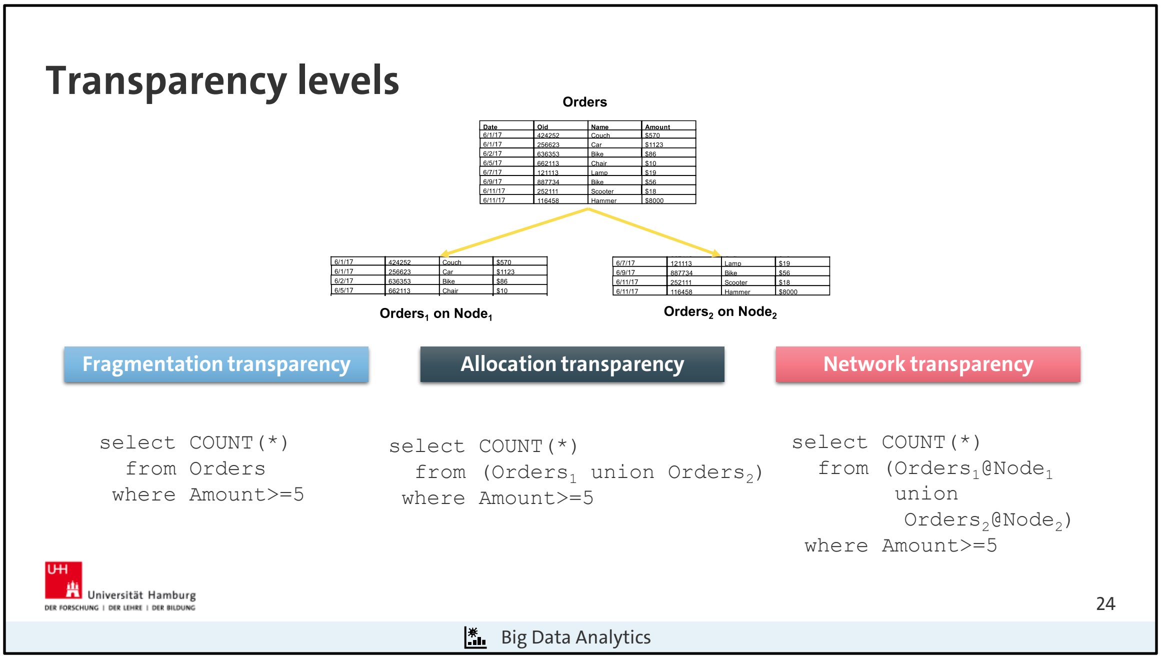
- Different hit rates per partition, e.g. range queries regarding distribution attribute for range partitioning
- Treatment: Hardly treatable, as determined by request and data transfer

Join Product Skew

- Different join selectivity per node
- Treatment:
    - Estimation of the total size of the join result as well as the resulting value distribution for the join attribute
    - Determine area partitioning, which provides a roughly equal partial result for each of the p join processors



More examples for transparency levels which might or might not be available in your chosen system: location transparency (like network transparency), naming transparency (no equal names must exist on two different sites),...



In a distributed DBMS with fragmentation transparency, users only need to know the database schema (tables & attributes)

- Fragmentation and allocation are "hidden" from users
- Behaves like a non-distributed database system

Allocation transparency: Users need to know fragmentation information (i.e., number and names of partitions, as well as partitioning function used)

- Allocation is "hidden" from users
- Users have to be aware of the partitioning scheme

Network transparency: Users need to know fragmentation and allocation

- However, users do not need to know physical network addresses (e.g., IPs) of machines

Which level of transparency?

- Compromise: Ease of use versus ability to control query processing by user
- Choice often depends on application
    - Full transparency often poor choice for geographically distributed DBMSs; Users want to control which operations are executed remotely
    - However, parallel DBMSs often implement full transparency