



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

FAKULTÄT
FÜR MATHEMATIK, INFORMATIK
UND NATURWISSENSCHAFTEN

Databases and Information Systems (DIS)

Dr. Annett Ungethüm
Universität Hamburg



Foto: UHH/Esfandiari

Course Outline



Architectures of Database Systems



Transaction Management



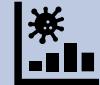
Modern Database Technology



Data Warehouses and OLAP



Data Mining



Big Data Analytics

Exercises

- Will start on 09.04.2024, first deadline 23.04.2024
- Working period: usually 1 or 2 weeks
- Handling
 - Teams of students (~2 students)
 - Mostly practical exercises
 - Exercise hours in Stellingen/Informatikum, different buildings → Look at STiNE for the building and room of your exercise
 - You can also work on exercises at home or at IRZ
- Demonstration of results/solutions during exercise hours, see advisors!

Requirements to obtain certificate

- **Successful and timely submission:** All but one exercise sheets must be submitted and approved on time. What exactly is required for an approval, depends on the specification of the task. For theoretical tasks, 50% of the maximum points are required in any case.
- **Teamwork:** All participants work in groups of two, submitting one common solution. Attempted fraud may lead to the reduction of points or, in severe cases, even to the denial or withdrawal of the certificate.
- **Active participation:** An active participation in the exercises through presentation of solutions and through discussions is also required. Refusal to participate actively may lead to the reduction of points.

Lecture

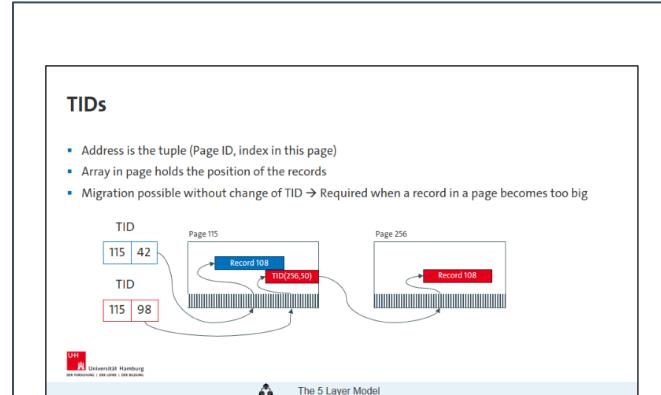
- 2x per week
 - Monday: 12:15 ESA J
 - Wednesday: 10¹⁵ Phil G
- In-person (unless stated otherwise)
- Exam:
 - 120 min
 - 31.07.2024, 9³⁰ (Audi 1)
 - 25.09.2024, 9³⁰ (ESA A)



Moodle

- lernen.min.uni-hamburg.de
→ Exercise sheets and lecture slides

Slideset with handout
(where applicable)



Deleting a record:

- The entry in the page array is marked as invalid
- All other records on the same page can be moved to maximize the free continuous space → only the positions are changed, not their index in the page array
- This way, record addresses are not changed, the same TIDs can still be used

Changing a record:

- **Case 1:** Record becomes smaller → all records are moved within the same page, positions in the page array are changed (same as with a deleted record)
- **Case 2:** Record becomes larger and space on the page is sufficient to store it → see case 1
- **Case 3:** Record becomes larger and space on the page is not sufficient to store it → Record is moved to another page and TID is stored on the original page (see Figure). If record is moved again later, TID in original page is changed again → only one additional reference necessary even if record is changed multiple times.

Course Outline



Architecture of Database Systems

- Towards a relational DBS architecture
- Evolution of the layer model
- The 5 Layer Model
- Properties of transactions



Transaction Management



Modern Database Technology



Data Warehouses and OLAP



Data Mining



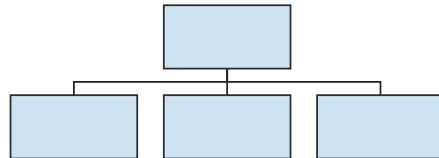
Big Data Analytics



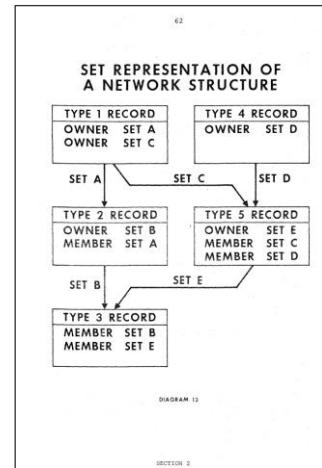
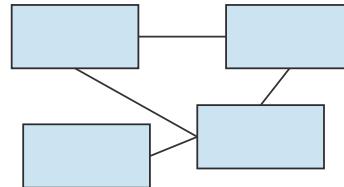
Universität Hamburg

Before The Relational Model, There Were Other Abstraction Models for Database Systems

- Hierarchical Model



- Network Model



Data base task group report to the CODASYL programming language committee, April 1971

Hierarchical Model:

- Each record has only one parent record
- A record is a collection of fields where each field contains one value
- The fields of a record are defined by its type
- Used only in few database systems today, e.g. in IBM Information management System

Network Model:

- Proposed by the Data Base Task Group of the CODASYL programming language committee as the most general form of a data structure → No limitation to the links between records
- Enables complex data structures but only simple operations
- Widely replaced by the relational model

The Relational Model

- Developed by Codd in 1970
- Created to be simple
- Became more popular than the network model with increased computing power
- Postulates independence of the language and implementation

How can this idea be transformed into a Database System?



Towards a relational DBS architecture

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine [the internal representation]. A consulting service which supplies such information is not a satisfactory one. An application program should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are also changed. Changes in data structure and data definition are an integral part of change in query, update, and report traffic, and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of their data. In Section 1, however, a number of these models are discussed. A natural branch on every relation is a normal form for data base relations, and the concept of a universal data sublanguage is introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the system.

KEY WORDS AND PHRASES: data bank, data base, data structure, data representation, hierarchy of data, networks of data, ordering dependency, redundancy, consistency, composition, join, retrieved language, predicate calculus, security, data integrity

CC CATEGORIES: 3.7.0, 3.7.2, 3.7.5, 4.20, 4.22, 4.29

1. Relational Model and Normal Form

1.1. INTRODUCTION

This paper is concerned with the application of elementary relational concepts which provide shared access to large banks of formatted data. Except for remarks by Childs [1], the principal application of relational data systems has been to deductive question-answering systems. Levin and Maron [2] provide numerous references to work in this area.

Section 1 appears to be superior in several respects to the graph or network model [3, 4] prevalent in logical information systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximum independence between the logical structure of data and the mechanical representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency in a natural way, as will be seen.

The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer analysis of the semantic and logical limitations of various formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearest perspective are often in previous parts of this paper. Intentions of systems to support the relational model are not discussed.

1.2. DATA DEDUCIBILITY OR PARSIMONY SYNONYMS

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate the manipulation of data in a variety of representations stored in a data bank. However, the variety of data representation characteristic which can be changed without logically impairing some application programs is still quite limited. In fact, the model of data which users perceive is still cluttered with operational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependencies, indexing dependencies, and dependencies between data elements whose dependencies are not clearly separable from one another.

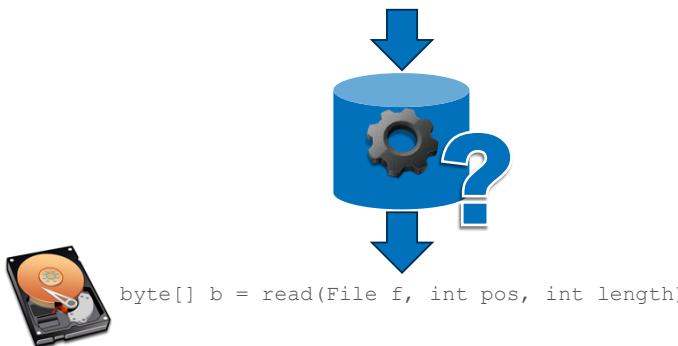
1.2.1. Ordering Dependencies. Elements of data in a data bank may be stored in a variety of ways, some involving no ordering, some ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is

Codd, E.F. (1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (6): 377–387.

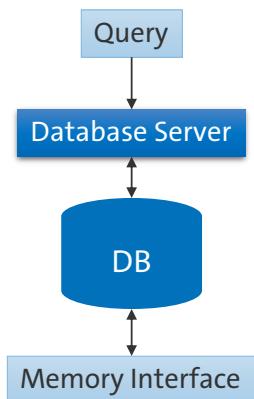
- In the 70s, implementations of the relational model were too slow for productive use
- Computing power grew rapidly
- Now the relational model is the standard abstraction model for database systems
- Declarative queries, the use of values, and set orientation make it easy to use compared to the network model which uses pointers and is record oriented

How are Relational Database Systems Built?

```
SELECT s.firstname, s.lastname, COUNT(l.name)
FROM Student s
INNER JOIN Program p ON s.programId = p.id
INNER JOIN Attendance a ON a.studentId = s.studentId
INNER JOIN Lecture l ON a.lectureId = l.id
GROUP BY s.firstname, s.lastname WHERE p.name='DSE'
```



The Monolithic Approach



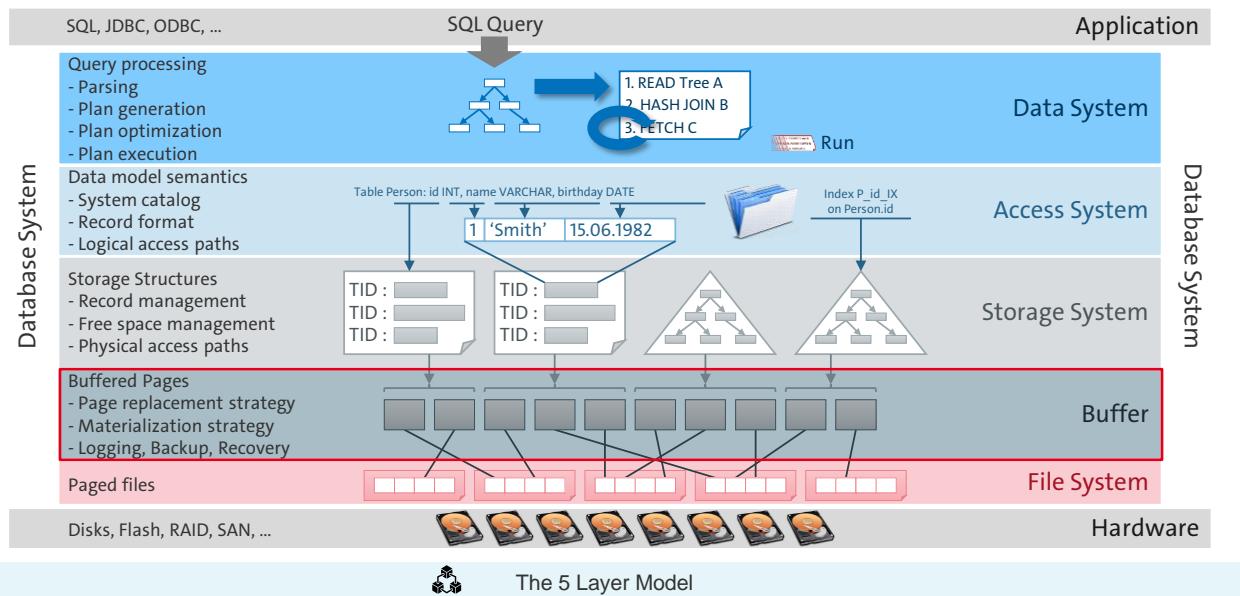
Evolution of Databases introduces more challenges

- New storage structures and access methods
- Changes of storage media
- Additional object types
- Integrity constraints
- Changing interfaces and application programs



Encapsulation via a hierarchical structure

The 5 Layer Model



Additional literature: Härder, Theo. "DBMS Architecture—the Layer Model and its Evolution." *Datenbank-Spektrum* 13 (2005): 45-57.

Received requests from upper layer (examples):

- Data System: Select * from mytable;
- Access System: FIND NEXT/STORE record
- Storage System: insert into B-Tree, store internal record
- Buffer: fetch page
- File System: Read/write block

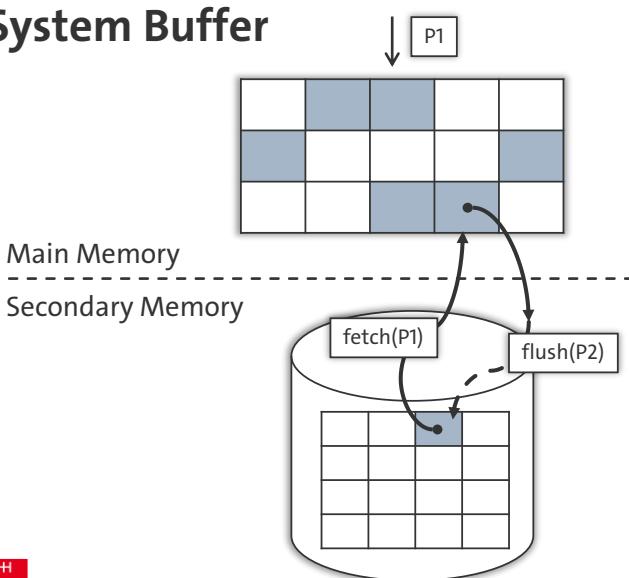
Managed objects:

- Data System: Relations, Views
- Access System: External (logical) records, Index Structures
- Storage System: Internal (physical) records, Hash tables, Trees
- Buffer: Segments, Pages
- File System: Files, Blocks

Layers are ideally independent of each other → For performance reasons they are sometimes merged in a DBS

The following slides show examples of the work of each layer. They are not a complete representation of the layer's tasks.

System Buffer

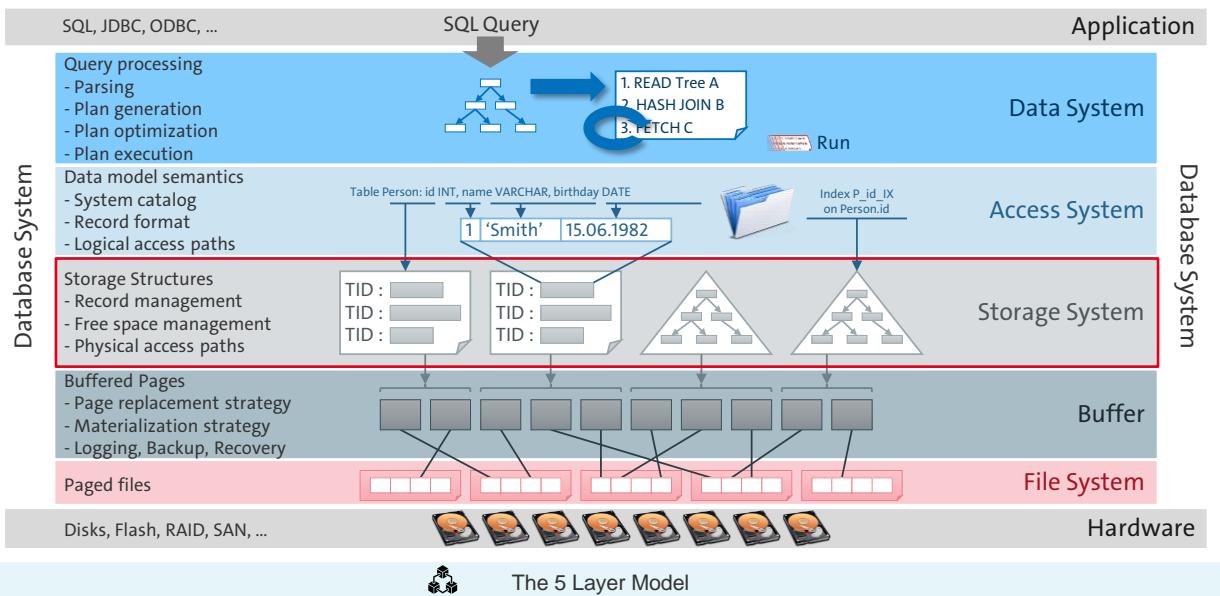


Calls of the System Buffer

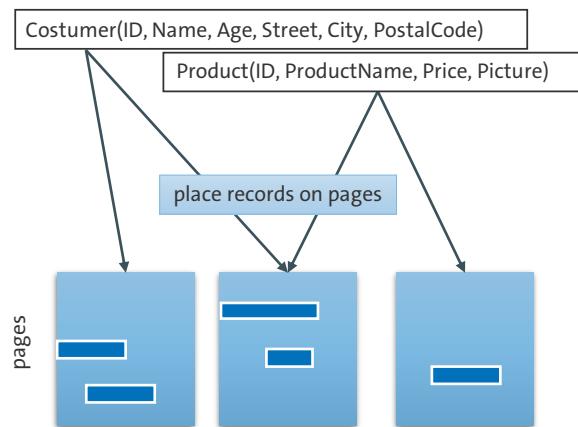
- Provide page (logical reference), might require to replace another page
- FIX – Fix page in the buffer, such that the content can be accessed, usually done when a page is provided
- UNFIX – Release a FIX, such that page can be replaced
- Revision mark – A note that the content has been changed and needs to be physically written to disc when it is replaced !!!The note must be written before applying the change!!!
- Write – Write the buffer to main memory/disc

- Read and write operations use the buffer of a DBS
- Buffer can only hold a fraction of the whole dataset
 - Page replacement strategies are used to manage buffer (FIFO, LIFO, LRU, LFU)
 - Ring buffers exist but are usually applied for stream processing
- Buffer consists of page structured segments and buffer control block (holds information necessary for managing buffer)
- Special feature of DB buffer in contrast to general buffer management (e.g. by the operating system): Application knowledge can be used for buffer management

The 5 Layer Model

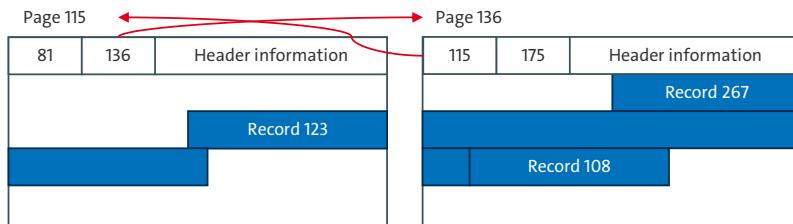


Record Management



- Representation of complete records on pages
- Pages have a fixed length, records can have a variable length
- Addressing of Records
 - Allocation Table
 - TID-concept (Tuple Identifier)
- Heap management

Organization of Pages



Addresses of records are created when they are inserted. They provide a way to access the records later.

Concatenation

Pages are linked together by double linked lists

Recording of free pages: heap management

Page Header

Information about previous and and following page

Optionally also number of the page itself

Information about the type of record (Table Directory)

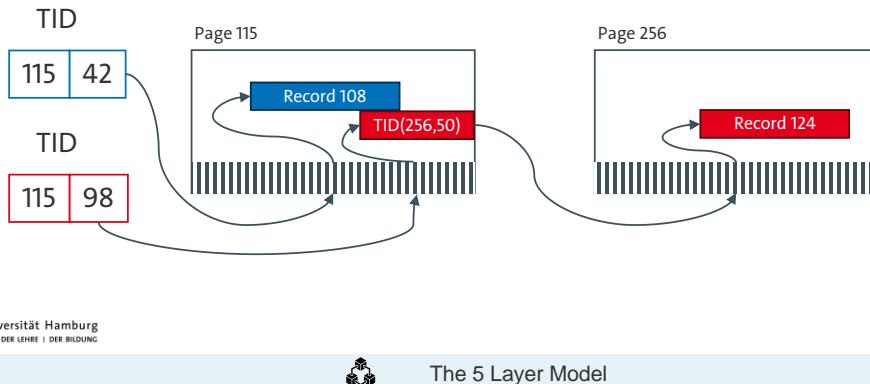
Information about free space

Issues and challenges

- Distinct addressing of records for the entire lifetime of the record
- Support of data migration
- (Runtime) stability against relocation within a page
- Fast access, access as direct as possible
- No necessity for frequent reorganizations

TIDs

- Address is the tuple (Page ID, index in this page)
- Array in page holds the position of the records
- Migration possible without change of TID → Required when a record in a page becomes too big



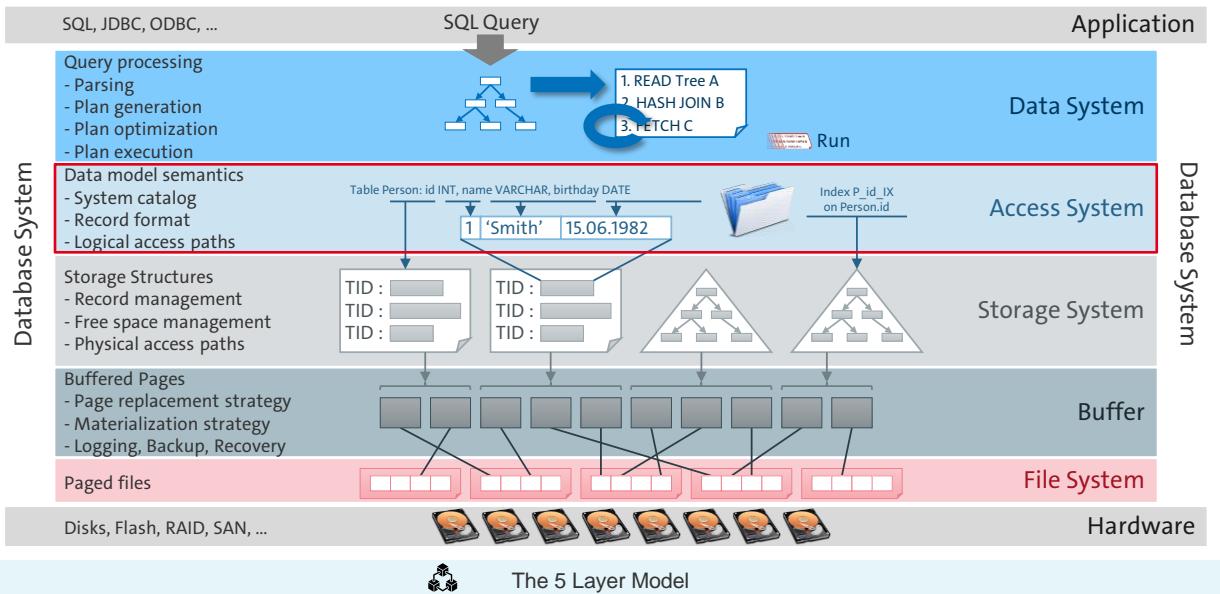
Deleting a record:

- The entry in the page array is marked as invalid
- All other records on the same page can be moved to maximize the free continuous space → only the positions are changed, not their index in the page array
- This way, record addresses are not changed, the same TIDs can still be used

Changing a record:

- **Case 1:** Record becomes smaller → all records are moved within the same page, positions in the page array are changed (same as with a deleted record)
- **Case 2:** Record becomes larger and space on the page is sufficient to store it → see case 1
- **Case 3:** Record becomes larger and space on the page is not sufficient to store it → Record is moved to another page and TID is stored on the original page (see Figure). If record is moved again later, TID in original page is changed again → only one additional reference necessary even if record is changed multiple times

The 5 Layer Model

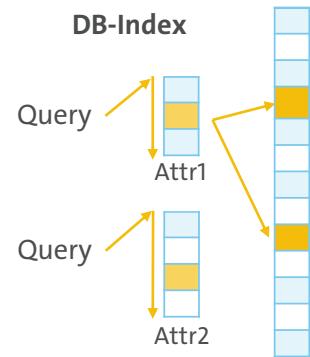
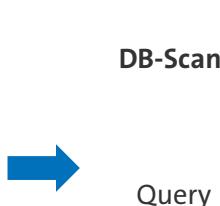


Use of Indexes

- To avoid full scans, indexes are used
→ Extra step to avoid unnecessary memory access and comparisons

How to get
this mapping:

My key value
 TID (123,3)

Types of access:

- Sequential access (scan) → not sufficiently fast for large record types or small result sets
- Sequential access on a sorted attribute
- Direct access via primary key, foreign key(s), composite keys, or search terms
- Navigation from a record to a set of associated records

Requirements:

- Efficient way of finding records, i.e. as direct as possible
- Avoiding sequential search of a dataset
- Facilitate access control by predefined access paths (constraints)
- Keep topological relations

B-Tree revisited

<https://www.cs.usfca.edu/~galles/visualization/BTree.html>

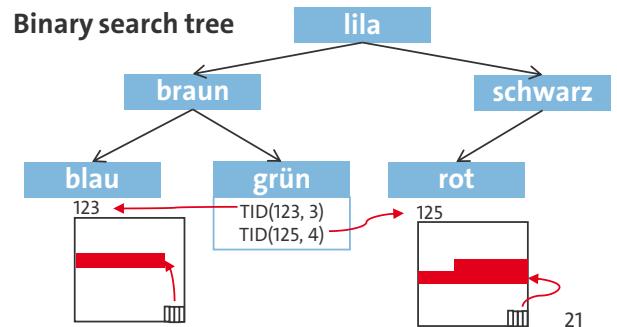
B-Trees were covered in DB Grundlagen (not only at UHH, but also at other universities)

Classification of Indexes

- Primary or Secondary Index (additional access path)
- Simple index or multilevel index
- Storage structure:
 - Tree-structured, e.g. B-Tree
 - Sequential, e.g. Sequential lists
 - Scattered, e.g. Hash regions
- Access method:
 - Key comparison
 - Key transformation, e.g. hashing

Sequential List

blau	
braun	
grün	TID(123, 3) TID(125, 4)
lila	
rot	
schwarz	



- Keys do not always fit into main memory, i. e. it can be useful to chose an index that minimizes disc accesses (e.g. B-Trees and its variants)
- Extension of binary trees: trees with multiple children per node → B-Trees were designed for use in DB systems
- Features: Dynamic reorganization by splitting and merging of pages, direct key access
- Extensions, e.g. B+ tree, B*-tree add more features, e.g. sorted sequential access

Creating an Index in SQL

- `CREATE INDEX my_index ON myTable(myAttr);`
- `CREATE INDEX my_index ON myTable(myAttr,myAttr2);`
- `CREATE UNIQUE INDEX my_index ON myTable (myAttr) INCLUDE (myAttr2)`
- `CREATE UNIQUE INDEX index ON myTable (myAttr) [DIS]ALLOW REVERSE SCANS`
- Some Systems allow computed indexes:
 - `CREATE INDEX my_index ON myTable (a + b * (c - 1), a, b);`



- The order of the attributes in a multilevel index is not commutative
 - An existing index on (myAttr1,myAttr2) can be used if both attributes are queried or only myAttr1. It cannot be used (efficiently) if only my Attr2 is queried.
- UNIQUE does not allow duplicate values
- INCLUDE includes the attribute in the key, but does not use it for the creation of the index, i.e. in a search tree it is only part of the leaf nodes but not necessary for the other nodes
 - Useful if an attribute is part of the column list in a query but not of the WHERE clause
- [DIS]ALLOW REVERSE SCANS
 - Indexes are usually scanned in the order they were created in. This can be used to allow the opposite behavior. The default depends on the DB system.
- Not all of these are available on every system. Always check the docs!

Databases and Information Systems (DIS)

Dr. Annett Ungethüm
Universität Hamburg

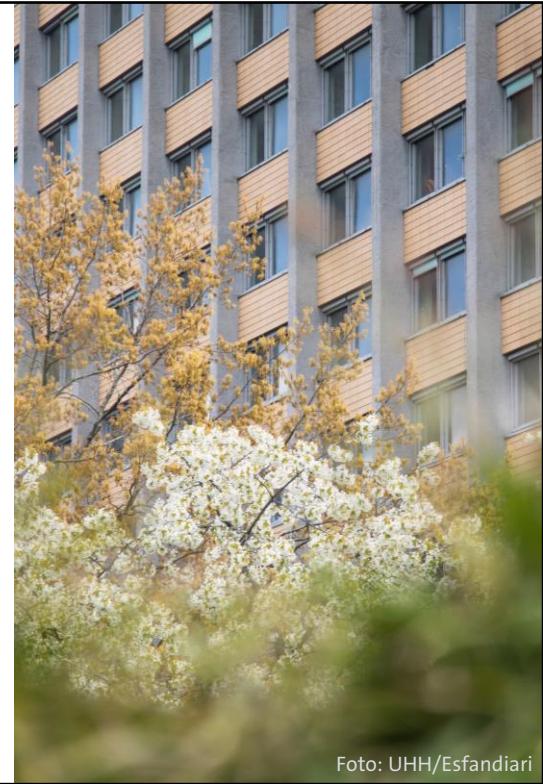
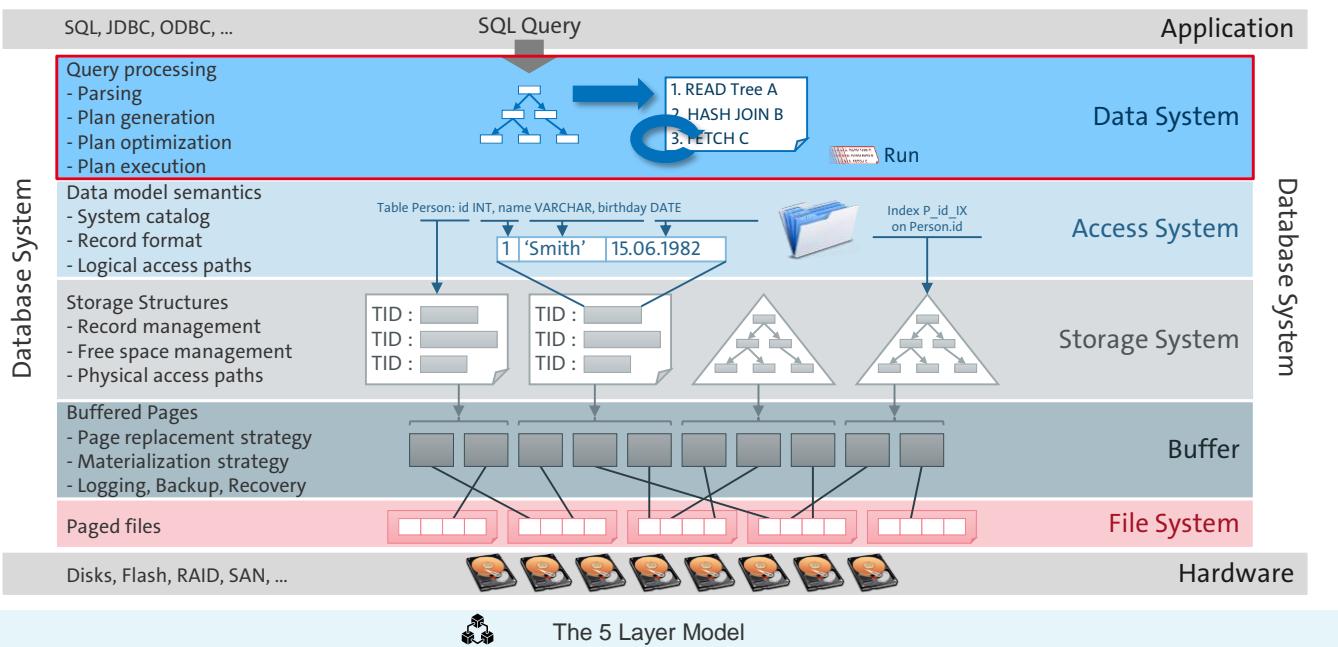


Foto: UHH/Esfandiari

The 5 Layer Model



Logical Query Execution Plan

MensaMeals

Meal	Price
Pizza	6,50
Pasta	4,90
Pie	1,20
Potato Salad	5,80
Pannfisch	7,90

DailyOffers

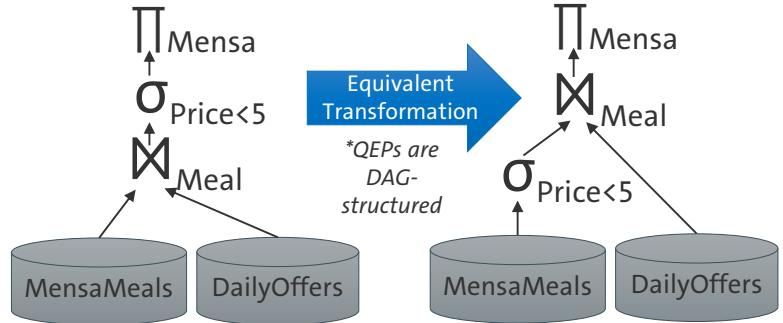
Mensa	Meal
Campus Mensa	Pizza
Mensa Cafe	Pie
Garden Mensa	Pasta
Old Mensa	Potato Salad

 Universität Hamburg
 DER FORSCHUNG | DER LEHRE | DER BILDUNG

```
SELECT Mensa FROM MensaMeals, DailyOffers
WHERE MensaMeals.Meal = DailyOffers.Meal
AND MensaMeals.Price < 5;
```

Plan A: $\prod_{\text{Mensa}} (\sigma_{\text{Price} < 5} (\text{MensaMeals} \bowtie_{\text{Meal}} \text{DailyOffers}))$

Plan B: $\prod_{\text{Mensa}} ((\sigma_{\text{Price} < 5} (\text{MensaMeals})) \bowtie_{\text{Meal}} \text{DailyOffers})$



- Database Systems use a relational algebra for internal representation (see database lecture of your bachelor program)
- Optimizers try to automatically find the most efficient sequence of operators
 - ➔ Conventional approach: Reduce data as early and as cheap as possible
 - ➔ Tool: Cardinality/Selectivity estimation
- The chosen sequence of operators is the final Query Execution Plan (QEP)

Further Reading

Foundations for operator order optimization: *Bringing Order to Query Optimization*, Slivinskas et al.

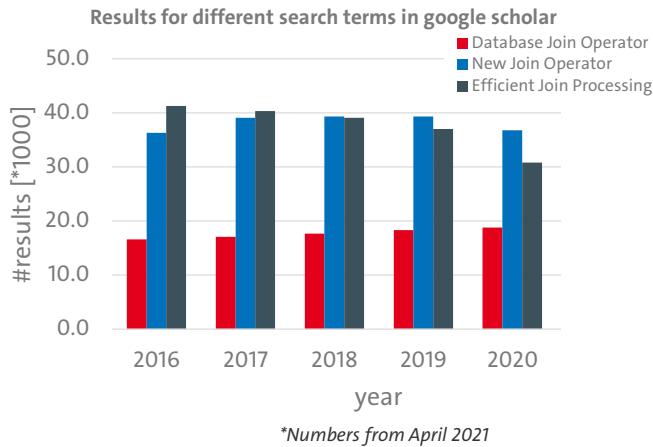
https://www.researchgate.net/publication/2916321_Bringing_Order_to_Query_Optimization

Survey on different cardinality estimation techniques: *Cardinality estimation: An Experimental Survey*, Harmouch and Naumann

<http://www.vldb.org/pvldb/vol11/p499-harmouch.pdf>

Physical Operator Selection

- For each **logical operator** (e.g. join), there can be different **physical operators** (e.g. hash-join, nested-loop-join), i.e. the same operator can be implemented in different ways



- Joins are a bottleneck in most queries → Join optimization is a much-noticed field of research
- Choice of physical operator depends on exact use case. Examples from PostgreSQL:
 - Nested-Loop: full join, one very small table, condition is not an equality
 - Hash Join: similarity joins, small expected hash table
 - Merge Join: sorted data, large tables
- EXPLAIN can be used to see the generated execution plan, often including the physical operators

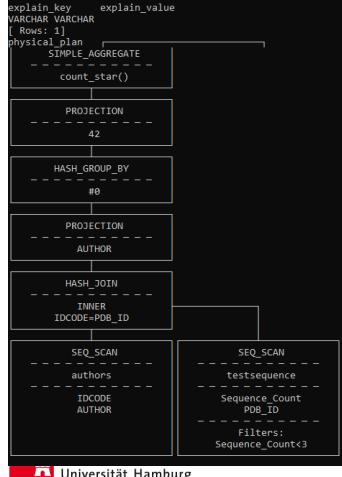
Further Reading

More on join order optimization: *Query optimization through the looking glass, and what we found running the Join Order Benchmark*, V.Leis et al.

Overview on Popular Join algorithms and an alternative:
New algorithms for join and grouping operations, G. Graefe

Output of EXPLAIN

Some show a graph
(duckdb)...



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

...some show an ugly graph (sqlite)...

```

QUERY PLAN
|--CO-ROUTINE 1
|  |--SCAN TABLE testsequence
|  |  |--SEARCH TABLE authors USING AUTOMATIC COVERING INDEX (IDCODE=?)
|  |  |--USE TEMP B-TREE FOR GROUP BY
|  |--SCAN SUBQUERY 1

```

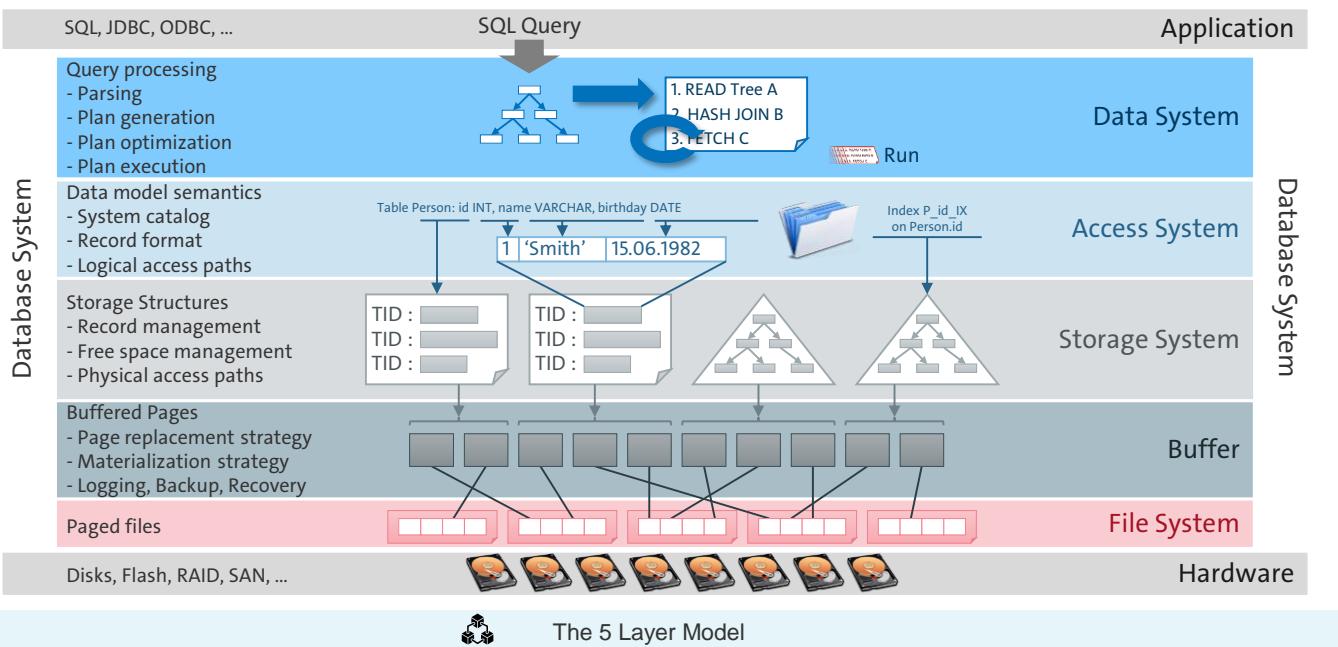
...and some show a formatted version of their internal RA representation (e.g. MonetDB, PostgreSQL)

```

function user.main():void;
  X_1:void := querylog.define("explain select count(*) from (select 1 from testsequence, authors where
");
  X_4:int := sql.mvc();
  C_5:bat[oid] := sql.tid(X_4:int, "sys");
  X_8:bat[:str] := sql.bind(X_4:int, "sys":str, "testsequence":str, "pdb_id":str, 0:int);
  X_15:bat[int] := sql.bindX(4:int, "sys":str, "testsequence":str, "sequence count":str, 0:int);
  C_22:bat[oid] := algebra.ThetaSelect(X_15:bat[int], C_5:bat[oid], 3:int, "<":str);
  C_24:bat[oid] := sql.tid(X_4:int, "sys":str, "authors":str);
  X_26:bat[:str] := sql.bind(X_4:int, "sys":str, "authors":str, "IDCODE":str, 0:int);
  X_31:bat[:str] := sql.bind(X_4:int, "sys":str, "authors":str, "AUTHOR":str, 0:int);
  X_36:bat[:str] := algebra.projection(C_22:bat[oid], X_8:bat[:str]);
  X_38:bat[:str] := algebra.projection(C_24:bat[oid], X_26:bat[:str]);
  X_41:bat[:oid] := algebra.join(X_38:bat[:str], X_36:bat[:str], nil:BAT, nil:BAT, false:bit, nil:lng);
  X_49:bat[:str] := algebra.projectionpath(X_41:bat[:oid], C_24:bat[:oid], X_31:bat[:str]);
  (X_50:bat[:oid], C_51:bat[:oid]) := group.groupdone(X_49:bat[:str]);
  X_53:bat[:str] := algebra.projection(C_51:bat[:oid], X_49:bat[:str]);
  X_56:bat[bte] := algebra.project(X_53:bat[:str], 1:bte);
  X_57:lng := agrr.count(X_56:bat[:bte]);
  X_59:int := sql.resultSet(".%2":str, "%2":str, "bigint":str, 64:int, 0:int, 7:int, X_57:lng);
end user.main;

```

The 5 Layer Model



Transactions

- Transactions change the state of a database
 - Records, TIDs, indexes,... must be changed and brought into a consistent state
 - Transactions have certain properties to ensure this consistent state at the start and end of a transaction



- Transactions include one or more statements that change the state of the database, e. g. INSERT, UPDATE, DELETE, CREATE, DROP
- To execute a transaction, it must be committed (COMMIT)
- Many DB systems have an autocommit mode, CREATE and DROP is usually committed automatically regardless of the mode
- Start of a transaction with multiple statements: BEGIN TRANSACTION
- Abbreviations: BOT (Begin Of Transaction), EOT (End Of Transaction), DML (Data Manipulation Language)
- More useful commands for working with transactions: ROLLBACK, SET TRANSACTION, SAVEPOINT

ACID Properties

- **Atomicity**
→ “All or nothing” property of any DBMS action
- **Consistency and semantic integrity**
→ A successful transaction guarantees that all integrity requirements are met
- **Isolated execution**
→ “Logical single user mode”
- **Durability**
→ Requires that modified data of successful transactions must survive any type of failure

Requires Transaction Management, i.e. Synchronization and Recovery



- Atomicity: Transactions are not split. Either the whole transaction is performed, or the transaction is not performed at all. There is no partial (successful) execution.
- Consistency: e.g. primary keys must be unique, values must be within their assigned value range, custom definitions (e.g. a product cannot have been sold before it was produced)
- Isolation: Multiple transactions are isolated from each other, i.e. they do not use inconsistent intermediate results of each other
- Durability: Successful transactions must be made persistent (unless the database is located entirely in volatile memory, then the result of the transaction is only written to this volatile memory)

Course Outline



Architectures of Database Systems



Transaction Management

- Synchronization
- Logging
- Recovery



Modern Database Technology



Data Warehouses and OLAP



Data Mining



Big Data Analytics

Why We Need Synchronization

Transaction 1

```
SELECT SALARY INTO :salary  
FROM EMPL  
WHERE ENR = 2345  
  
salary := salary + 2000;  
  
UPDATE EMPL  
SET SALARY = :salary  
WHERE ENR = 2345
```

Transaction 2

```
SELECT SALARY INTO :salary  
FROM EMPL  
WHERE ENR = 2345  
  
salary := salary + 1000;  
  
UPDATE EMPL  
SET salary = :salary  
WHERE ENR = 2345
```

Which result do we expect after both transactions have finished?

Which results can we get if both transactions run concurrently?

→ If we let the transactions execute whenever they want, things can (and will) go wrong

- A number of different anomalies can occur during concurrent execution of transactions
- We expect a salary increase of 3000, but this is not guaranteed if we let both transactions run concurrently without further measures
- Why do we not just run all transactions serially?
 - It's slow!
 - It makes sense in many scenarios, e.g.:
 - Multiple Users (real and virtual)
 - Multiple available CPUs, distributed systems in general

Anomalies

Salary change T₁

```
SELECT SALARY INTO :salary  
FROM EMPL  
WHERE ENR = 2345  
  
salary := salary + 2000;  
  
UPDATE EMPL  
SET SALARY = :salary  
WHERE ENR = 2345
```

Salary change T₂

```
SELECT SALARY INTO :salary  
FROM EMPL  
WHERE ENR = 2345  
  
salary := salary + 1000;  
  
UPDATE EMPL  
SET salary = :salary  
WHERE ENR = 2345
```

Database
(ENR, SALARY)

2345 39.000

2345 41.000

2345 40.000

→ Lost Update

time

Synchronization

- Both transactions read salary = 39.000
- The first transaction computes salary = 39.000 + 2.000 = 41.000
- The first transaction commits salary = 41.000
- The second transaction computes salary = 39.000 + 1.000 = 40.000
- Transaction 2 overwrites the result of transaction 1
- Possible solutions:
 - Locking, i.e. do not allow another transaction to write salary
 - Validate at the time of writing the update

Anomalies (II)

Salary change T₁

```
UPDATE EMPL  
SET SALARY = SALARY + 1000  
WHERE ENR = 2345
```

```
ROLLBACK
```

Salary change T₂

```
SELECT SALARY INTO :salary  
FROM EMPL  
WHERE ENR = 2345  
  
salary := salary * 1.05;  
  
UPDATE EMPL  
SET salary = :salary  
WHERE ENR = 2345  
  
COMMIT
```

Database
(ENR, SALARY)

2345 39.000

2345 40.000

2345 42.000

2345 39.000

Dirty Read

time

Synchronization

- Transaction 2 reads a value that is not committed
- Transaction 1 had not finished while transaction 2 was already reading the changed value
 - Database was not in a consistent state when T₂ started
- Solutions:
 - Read data only after it has been committed
- Dirty read is also sometimes referred to as *Inconsistent Read*

Anomalies (III)

Salary change T₁

```
UPDATE EMPL  
SET SALARY = SALARY + 1000  
WHERE ENR = 2345  
UPDATE EMPL  
SET SALARY = SALARY + 2000  
WHERE ENR = 3456  
COMMIT
```

Get salaries T₂

```
SELECT SALARY INTO :g1  
FROM EMPL  
WHERE ENR = 2345  
  
SELECT SALARY INTO :g2  
FROM EMPL  
WHERE ENR = 3456  
  
sum := g1 + g2
```

Database (ENR, SALARY)

2345	39.000
3456	45.000
2345	40.000
3456	47.000

Non-Repeatable Read

time

What is the result for *sum* and which result did we expect?

- Values change while T₂ is processed
- Database was in a consistent state during the start of T₂ and during the second read access of *SALARY*
- Solutions:
 - Lock read access
 - Multiversion concurrency control (mvcc) → DB holds multiple versions with time stamps or transaction numbers → later in this lecture

Anomalies (IV)

Get salaries T₁

```
SELECT SUM(Salary)  
INTO :Sum1  
FROM Empl  
WHERE DepNr = 17
```

```
SELECT SUM(Salary)  
INTO :Sum2  
FROM Empl  
WHERE DepNr = 17
```

Create new record T₂

```
INSERT INTO Empl(ENR, DepNr, Salary)  
Values(4567, 17, 55.000)  
COMMIT
```

Database
(ENR, DepNr, Salary) Sum

...
2345	17	39.000
3456	17	45.000
...		

84.000

...
2345	17	39.000
3456	17	45.000
...		
4567	17	55.000

139.000

Phantom Problem

time

- T₁ computes the sum of *Salary* twice → the results are different
- Resembles a non-repeatable read, but spans multiple records or even the whole relation

Schedules (I)

Salary change T₁

```
SELECT SALARY INTO :salary  
FROM EMPL  
WHERE ENR = 2345  
  
salary := salary + 2000;  
  
UPDATE EMPL  
SET SALARY = :salary  
WHERE ENR = 2345
```

Salary change T₂

```
SELECT SALARY INTO :salary  
FROM EMPL  
WHERE ENR = 2345  
  
salary := salary + 1000;  
  
UPDATE EMPL  
SET salary = :salary  
WHERE ENR = 2345
```



Step	T ₁	T ₂
1	r(S)	
2		r(S)
3	w(S)	
4		w(S)

→ Lost Update

- A schedule shows the order in which the operations of different transactions are executed
- Schedules can be used to identify conflicts
- Possible operations are:
 - r – read
 - w – write
 - a – abort
 - c – commit

Schedules (II)

Salary change T₁

```
UPDATE EMPL  
SET SALARY = SALARY + 1000  
WHERE ENR = 2345
```

ROLLBACK

Salary change T₂

```
SELECT SALARY INTO :salary  
FROM EMPL  
WHERE ENR = 2345  
  
salary := salary * 1.05;
```

```
UPDATE EMPL  
SET salary = :salary  
WHERE ENR = 2345
```

COMMIT



Step	T ₁	T ₂
1	w(S)	
2		r(S)
3		w(S)
4	a	
5		c

→ Dirty Read



Example Schedule

Step	T ₁	T ₂	T ₃	Step	T ₁	T ₂	T ₃
1	r(A)			8		w(C)	
2		r(B)		9		w(A)	
3		r(C)		10			r(A)
4		w(B)		11			r(C)
5	r(B)			12	w(B)		
6	w(B)			13			w(C)
7		r(A)		14			w(A)

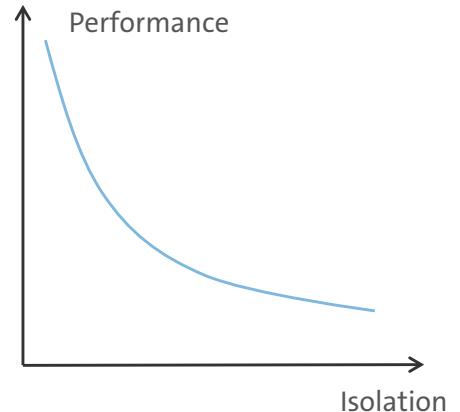
- r – read, w – write, (X) – object X
- DR – Dirty Read, LU – Lost Update
- “<“ is an ordering relation which is defined by the schedule
- Conflicts leading to anomalies, i.e. the sequence of execution is critical:
 - $w_2(B) < r_1(B) \rightarrow DR$
 - ~~$w_1(A) < r_2(A) \rightarrow DR$~~ (changed 08.04.24)
 - $w_2(C) < r_3(C) \rightarrow DR$
 - $w_2(A) < r_3(A) \rightarrow DR$
 - ~~$w_1(A) < w_2(A) \rightarrow LU$~~ (changed 08.04.24)
 - $w_2(A) < w_3(A) \rightarrow LU$
 - $w_2(B) < w_1(B) \rightarrow LU$ (2x)
 - $w_2(C) < w_3(C) \rightarrow LU$

→ If multiple operations work with the same object and at least one of them is a write operation, a concurrent execution leads to conflicts

- The database is not in a consistent state during repeated read accesses. Thus, we do not have to search for non-repeatable reads.
- Phantom problem is not identifiable with the provided information

Isolation in SQL

- Isolation is expensive because it introduces locks
→ Transactions have to wait for each other
- Absolute Serializability (=avoiding anomalies) is not always necessary
→ SQL allows different isolation levels



- It's always a trade-off
- Right level of isolation depends on the use-case

ANSI-SQL isolation levels

Isolation Level	Lost Update possible?	Dirty Read possible?	Non-Repeatable Read possible?	Phantom Read possible?
Read Uncommitted	No	Yes	Yes	Yes
Read Committed	No	No	Yes	Yes
Repeatable Read	No	No	No	Yes
Serializable	No	No	No	No

Higher isolation / protection against anomalies

More concurrency / higher performance

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Synchronization

20

Set isolation level with SQL:

```
SET TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED| READ COMMITTED |
REPEATABLE READ | SERIALIZABLE }
```

Special case: read uncommitted → Should avoid Lost Update, but does not always avoid them in reality

A more in-depth discussion and critique of the isolation levels can be found in the following paper:

Berenson, Hal, et al. "A critique of ANSI SQL isolation levels." *ACM SIGMOD Record* 24.2 (1995): 1-10. (<https://dl.acm.org/doi/pdf/10.1145/568271.223785>)
→ The list of authors might make it look heavily biased, which it might be. But the authors still have a point, and the paper is peer-reviewed



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

FAKULTÄT
FÜR MATHEMATIK, INFORMATIK
UND NATURWISSENSCHAFTEN

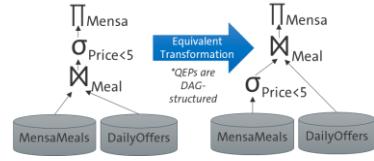
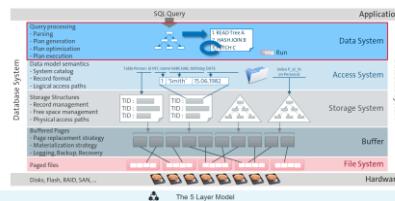
Databases and Information Systems (DIS)

Dr. Annett Ungethüm
Universität Hamburg

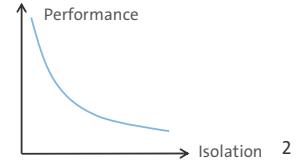


Foto: UHH/Esfandiari

Summary



Step	T ₁	T ₂	T ₃	Step	T ₁	T ₂	T ₃
1	r(A)			8			
2		r(B)		9			
3		r(C)		10		w(A)	
4			w(B)	11			r(C)
5		r(B)		12	w(B)		
6		w(B)		13			w(C)
7		(r(A))		14			w(A)



ANSI-SQL isolation levels

Isolation Level	Lost Update possible?	Dirty Read possible?	Non-Repeatable Read possible?	Phantom Read possible?
Read Uncommitted	No	Yes	Yes	Yes
Read Committed	No	No	Yes	Yes
Repeatable Read	No	No	No	Yes
Serializable	No	No	No	No

UH Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Synchronization

3

Set isolation level with SQL:

```
SET TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED| READ COMMITTED |
REPEATABLE READ | SERIALIZABLE }
```

Special case: read uncommitted → Should avoid Lost Update, but does not always avoid them in reality

A more in-depth discussion and critique of the isolation levels can be found in the following paper:

Berenson, Hal, et al. "A critique of ANSI SQL isolation levels." *ACM SIGMOD Record* 24.2 (1995): 1-10. (<https://dl.acm.org/doi/pdf/10.1145/568271.223785>)
→ The list of authors might make it look heavily biased, which it might be. But the authors still have a point, and the paper is peer-reviewed

Serializability

The concurrent execution of a set of transactions is considered to be correct, if there is a serial execution of the same set of transactions, leading to **the same resulting DB state** as well as **the same output values** as the original sequential execution.

- Each schedule that has the same effect as a serial one is considered to be correct
- A schedule is serializable if there are no loops in its dependency graph

Complexity of topological sorting to get the possible serializable schedules is $(O(N^2))$

More about the correctness of schedules:

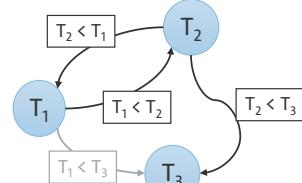
Vossen, Gottfried. "Database transaction models." *Computer Science Today: Recent Trends and Developments* (2005): 560-574.

Example: Dependency Graph

Step	T ₁	T ₂	T ₃	Step	T ₁	T ₂	T ₃
1	r(A)			8		w(C)	
2		r(B)		9		w(A)	
3		r(C)		10			r(A)
4		w(B)		11			r(C)
5	r(B)			12	w(B)		
6	w(A)			13			w(C)
7		r(A)		14			w(A)

Conflicts

- $w_2(B) < r_1(B)$
 - $w_1(A) < r_2(A)$
 - $w_2(C) < r_3(C)$
 - $w_2(A) < r_3(A)$
 - $w_1(A) < w_2(A)$
 - $w_2(A) < w_3(A)$
 - $w_1(A) < w_3(A)$
 - $w_2(B) < w_1(B)$
 - $w_2(C) < w_3(C)$



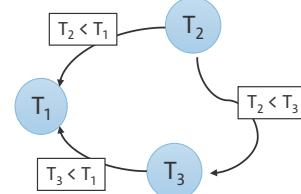
→ Not serializable!



- Cycle: $T_1 \leftrightarrow T_2$
 - $w_x(Z) < r_y(Z) \rightarrow$ Dirty Read
 - $w_x(Z) < w_y(Z) \rightarrow$ Lost Update

Example Schedule (II)

Step	T ₁	T ₂	T ₃
1	r(A)		
2		r(C)	
3		r(B)	
4		w(B)	
5		w(C)	
6			r(C)
7			r(B)
8			w(B)
9	r(B)		

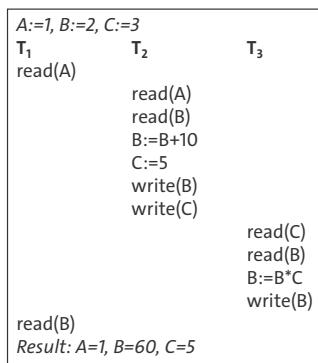


→ Serializable!

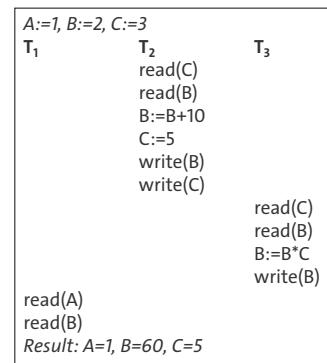
- Conflicts:
 - $w_2(B) < r_1(B) \rightarrow DR$
 - $w_3(B) < r_1(B) \rightarrow DR$
 - $w_2(C) < r_3(C) \rightarrow DR$
 - $w_2(B) < r_3(B) \rightarrow DR$
 - $w_2(B) < w_3(B) \rightarrow LU$

Correct Schedules

Step	T ₁	T ₂	T ₃
1	r(A)		
2		r(C)	
3		r(B)	
4		w(B)	
5		w(C)	
6			r(C)
7			r(B)
8			w(B)
9	r(B)		



Schedule: T₂, T₃, T₁



- Example for incorrect schedule: T₁, T₃, T₂
 → Results: A=1, B=2*3+10=16≠60, C=5

Correct Schedules (II)

Step	T ₁	T ₂	T ₃
1	r(A)		
2		r(A)	
3		r(B)	
4		w(B)	
5		w(A)	
6			r(C)
7			r(B)
8			w(B)
9	r(A)		

Schedule: T₂, T₃, T₁

A:=1, B:=2, C:=3
 T₁ T₂ T₃
 read(A) read(B) B:=B+10
 read(B) B:=B+10 A:=5
 B:=B+10 A:=5 write(B)
 A:=5 write(B) write(A)
 write(A) read(C) read(B)
 read(B) B:=B*C
 B:=B*C write(B)
 read(A) Result: A=5, B=36, C=3

Schedule: T₂, T₁, T₃

A:=1, B:=2, C:=3
 T₁ T₂ T₃
 read(A) read(B) B:=B+10
 read(B) B:=B+10 A:=5
 B:=B+10 A:=5 write(B)
 A:=5 write(B) write(A)
 write(A) read(C) read(B)
 read(B) B:=B*C
 B:=B*C write(B)
 read(A) Result: A=5, B=36, C=3

Schedules are equivalent!

- If T₁ does not read B, B can be changed before or after T₁ has finished
 → T₁ does not require results from T₃ anymore
 → Multiple correct schedules
- Effect of a schedule that is not correct:
 • T₃, T₂, T₁ → result: 2*3+10=16 ≠ 36

Equivalence

Let \lessdot_H and \lessdot_G be the ordering relations of the schedules H and G , then the two execution plans H and G are considered equivalent if the following conditions are true for all objects A , and Transactions i and j :

$$\begin{aligned} \text{read}_i[A] \lessdot_H \text{write}_j[A] &\Leftrightarrow \text{read}_i[A] \lessdot_G \text{write}_j[A] \\ \text{write}_i[A] \lessdot_H \text{read}_j[A] &\Leftrightarrow \text{write}_i[A] \lessdot_G \text{read}_j[A] \\ \text{write}_i[A] \lessdot_H \text{write}_j[A] &\Leftrightarrow \text{write}_i[A] \lessdot_G \text{write}_j[A] \end{aligned}$$

→ Not all correct schedules are equivalent!

Correctness and Equivalency

A:= 1000, B:= 2000

T ₁	T ₂
read(A)	read(A)
A:=A-50	X := A*0.1
write(A)	
read(B)	A := A-X
B := B+50	write(A)
write(B)	read(B)
	B := B+X
	write(B)

Both schedules are correct, but they are not equivalent!

T₁
read(A)
A:=A-50
write(A)
read(B)
B := B+50
write(B)

T₂
read(A)
X := A*0.1
A := A-X
write(A)
read(B)
B := B+X
write(B)

Results
 $A = (1000-50) - (1000-50)*0,1 = 855$
 $B = 2000+50 + (1000-50)*0,1 = 2145$

T₁
read(A)
X := A*0.1
A := A-X
write(A)
read(B)
B := B+X
write(B)

Results
 $A = 1000 - (1000 * 0,1) - 50 = 850$
 $B = 2000 + (1000 * 0,1) + 50 = 2150$

For n Transactions, there are n! possible serial schedules

→ Reminder from slide 13: Correctness means, that a plan is equivalent to one of these serial schedules, i.e. it has the same effect as one of the serial schedules!

Properties of Transaction Schedules

Recoverable (RC)

A transaction can only finish if all transactions it is reading from have already finished

→ Required for Durability

Example for not recoverable schedule
 $H = w_1[x], r_2[x], w_2[y], c_2, a_1$

Strict execution (ST)

Objects that have been changed by a transaction T_i , cannot be read or changed by another transaction before transaction T_i has finished

→ Transactions can be reset using a before-image

Avoid cascading aborts (ACA)

Each transaction T_a can only see results from other transactions T_b when the transactions T_b have finished

RC

- A schedule is recoverable if the following condition is met:
If a transaction T_i reads from another transaction T_j ($i \neq j$), then $c_j < c_i$
→ The transaction that was read from must commit before the reading transaction
- Example of not recoverable schedule H
→ T_2 reads from T_1 , but T_2 commits first

ST

- Physical logging using before images
→ Physical logging stores values (instead of operations)

ACA

- If reading results from another TA T is allowed before it has finished, an abort can lead to cascading resets
→ All TAs that have read from the aborted TA T have to be reset + all TAs that used a result of T indirectly

Strict execution avoids cascades, but not necessarily the other way round

(Counter) Examples ACA & ST

Schedule A

Step	T ₁	T ₂	T ₃	T ₄	T ₅
0.	...				
1.		w ₁ [x]			
2.			r ₂ [x]		
3.			w ₂ [y]		
4.				r ₃ [y]	
5.				w ₃ [z]	
6.					r ₄ [z]
7.					w ₄ [v]
8.					r ₅ [v]
9.	a ₁				

Schedule B



B = r₁[x] w₁[x] w₂[x] c₁ c₂

12

Synchronization

Schedule A:

A \notin ACA \rightarrow When T₁ is aborted, T₂ – T₅ must also be rolled back

Schedule B:

B \notin ST \rightarrow w₂(x) takes place before T₁ has finished

But

B \in ACA \rightarrow No transaction reads the result of the other transaction

Database Scheduler

- Sorts the operations and ensures that the schedule/history is serializable and can be rolled back
- Operations can be
 - Executed immediately
 - Rejected
 - On hold
- Strategies:
 - Optimistic
 - Pessimistic



Pessimistic Scheduling

- Operations are set back → Scheduler finds a sequence for operations
- Used when many transactions are expected to change the data
- Most common implementation is based on Locks → Operations are on hold until according lock is released

Optimistic Scheduling

- Scheduler tries to execute operations as soon as possible
- Used When only few transactions are expected to change the data
 - Damage might happen and must be repaired afterwards
 - Popular implementations: Time stamp based scheduling, optimistic synchronization

Pessimistic Synchronization

- Locks for exclusive access of data objects

		Active lock		
		NL	R	X
Requested lock	R	+	+	-
	X	+	-	-

NL – No Lock
R – Read Lock
X – eXclusive Lock

- Realizes a logical single-user setup
- For each data object, a central lock table stores the lock mode
- R lock: Object can only be read by other transactions, it cannot be changed
- X lock: Object can neither be read nor changed by another transaction
- Example: If an object is already locked with a read lock (R), an exclusive lock (X) cannot be acquired for that object, but another read lock (R) can be acquired.
- Read Lock is also sometimes referred to as “Shared lock (S)”

Working with Locks

Static locks

Claimed at the start of a transaction
(Preclaiming)

Dynamic locks

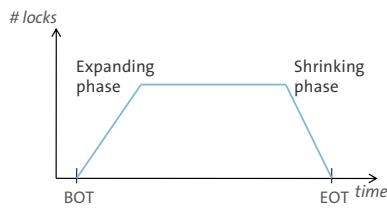
Claimed when needed
→ Possibility of deadlocks

- Locks have to be kept until the transaction finishes to guarantee serializability
→ This constraint can be relaxed for optimization reasons, e.g. early release of read locks

- Remember: We do not always need perfect serializability
- An early release of read locks can lead to different results
- More rules for working with locks:
 - Each object that is used by a transaction, has to be locked before its usage
 - A transaction that is currently holding a lock, cannot claim the same lock again
 - If a lock cannot be claimed (see last 14), the transaction is queued
 - A transaction must not acquire another lock after the first lock has been released (→ 2-phase-locking-protocol)
 - When a transaction ends, it must release all locks

2-Phase-Locking-Protocol (2PL)

- Expanding phase: Locks are acquired but not released
- Shrinking phase: Locks are released but not claimed



- Locks can only start to be released after all locks have been acquired
 - 2PL is almost(!) sufficient for a correct execution of transactions
- Deadlocks can still happen

2PL Example

Step	T ₁	T ₂	Comment
1.	BOT		
2.	lockX[x]		
3.	r[x]		
4.	w[x]		
5.		BOT	
6.		lockR[x]	T ₂ must wait
7.	lockX[y]		
8.	r[y]		
9.	unlockX[x]		Wake up T ₂
10.		r[x]	
11.		lockR[y]	T ₂ must wait
12.	w[y]		
13.	unlockX[y]		Wake up T ₂
14.		r[y]	
15.	commit		
16.		unlockR[x]	
17.		unlockR[y]	
18.		commit	

Expanding phase of T₁: Step 1 to 7

Expanding phase of T₂: Step 5 to 11

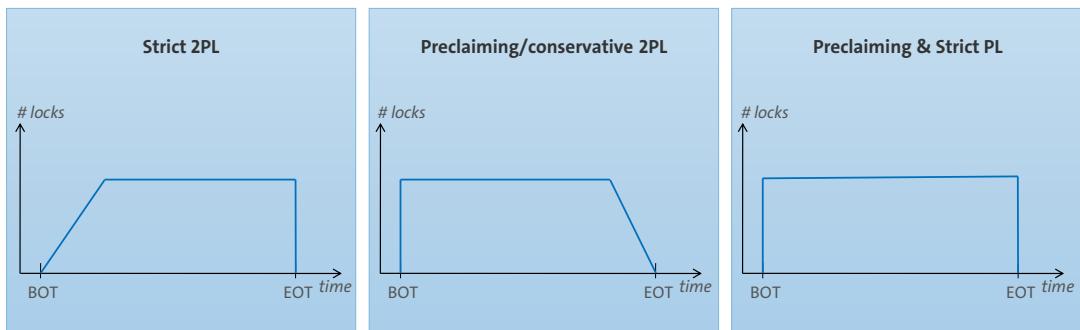
Shrinking phase of T₁: Step 9 to 15

Shrinking phase of T₂: Step 16 to 18

Step 6: T₂ cannot acquire lock for x because T₁ holds an exclusive lock on x → T₂ must wait

Step 9: The lock of x is released in T₁, T₂ can continue

2PL Variations



Strict 2PL

All locks are released after all other operations have finished

- Avoids cascading aborts of transactions that read the results of a rolled back transaction

Preclaiming

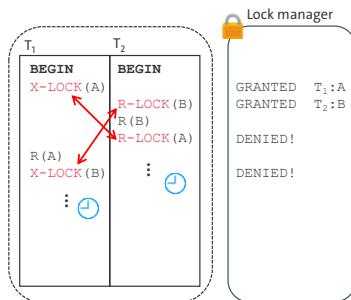
All locks are set before any other operations are executed

- Avoids deadlocks
- Decreases parallelism
- Requires to know all necessary locks at the BOT (not trivial in statements with branches)

Strict 2PL and Preclaiming

Enforces sequential execution

Deadlocks



→ Next week



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

FAKULTÄT
FÜR MATHEMATIK, INFORMATIK
UND NATURWISSENSCHAFTEN

Databases and Information Systems (DIS)

Dr. Annett Ungethüm
Universität Hamburg



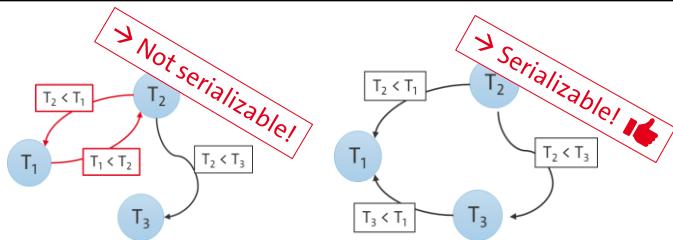
Foto: UHH/Esfandiari

Summary



Equivalency

$\text{read}_i[A] <_H \text{write}_j[A] \Leftrightarrow \text{read}_i[A] <_G \text{write}_j[A]$
 $\text{write}_i[A] <_H \text{read}_j[A] \Leftrightarrow \text{write}_i[A] <_G \text{read}_j[A]$
 $\text{write}_i[A] <_H \text{write}_j[A] \Leftrightarrow \text{write}_i[A] <_G \text{write}_j[A]$



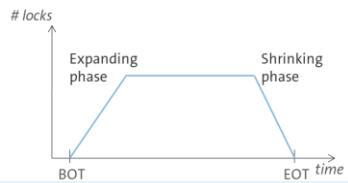
Recoverable (RC)

Strict execution (ST)

Avoid cascading aborts (ACA)

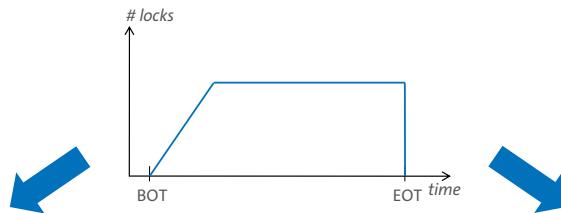


Active lock			
	NL	R	X
R	+	+	-
X	+	-	-



Addition to Strict 2PL

- ...because it will be mentioned in one of the future exercises
- There are 2 versions of Strict 2PL

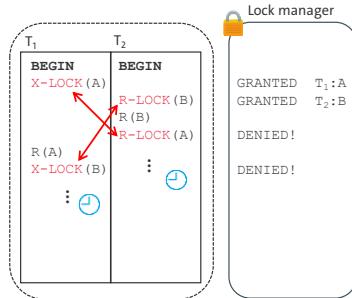


"normal" strict 2PL (**S2PL**) applies only to write locks (exclusive locks), read locks can still be released regularly

Strong strict 2PL (**SS2PL**) applies to read and write locks

- SS2PL schedules are always also S2PL schedules
- SS2PL avoids cascading aborts (ACA)
- SS2PL is sometimes called Rigorous 2PL

Deadlocks



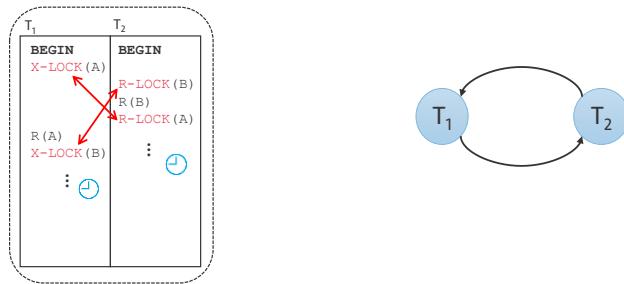
Detect
deadlocks when they happen

Avoid
deadlocks before they happen

- Deadlocks cannot be guaranteed to be avoided when locks are used in pessimistic scheduling
- Transactions wait reciprocally for the release of locks
- Requirements for Deadlocks:
 - Parallel access
 - Claim of exclusive locks
 - Claiming transaction already has a lock on data objects
 - No early release of locks (non-preemption → compare with [non]-preemptive scheduling for CPUs by operating systems)

Deadlock Detection

- System creates a “waits-for” graph
 - One transaction → One node
 - Edge from transaction T_i to transaction T_j if T_i waits for an unlock of T_j
 - Graph is periodically checked for cycles



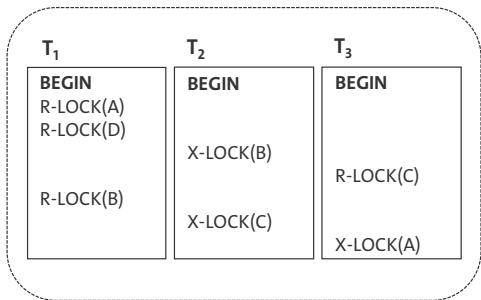
What to do when a deadlock has been detected?

→ Choose one of the transactions and roll it back

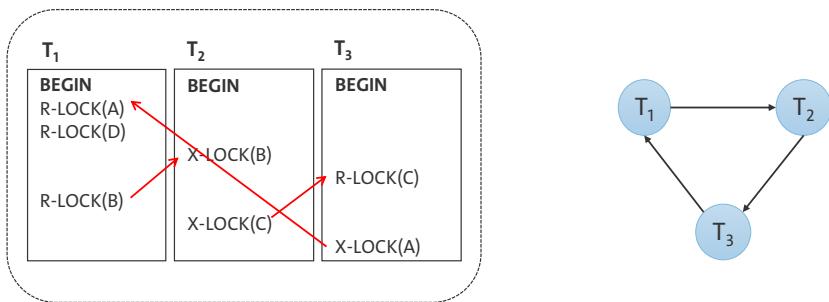
→ Which transaction to choose? Some heuristics:

- Most recent time stamp
- Process (smallest number of processed operations)
- Number of held locks
- Number of transactions reading from this transaction
- Number of previous rollbacks on the same transaction (so the transaction is not rolled back every time and can finally finish)

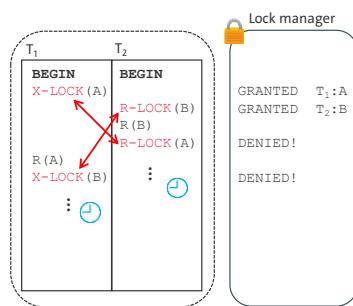
Exercise: Waits-For Graph



Exercise: Waits-For Graph



Deadlocks

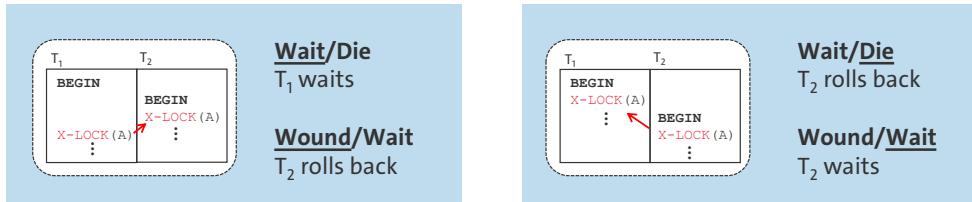


Detect
deadlocks when they happen

Avoid
deadlocks before they happen

Avoiding Deadlocks

- If a transaction claims a lock that is already held by another transaction, one of the transactions is rolled back
- Priorities based on time stamps → Older transaction gets higher priority
- Two strategies: Wait/Die & Wound/Wait



Terminology:

- Name shows what the claiming transaction does if there is a lock conflict
 - Wait: The claiming transaction waits for the lock to be released
 - Wound: Claiming transaction “wounds” the transaction holding the lock, i.e. it tries to roll back the owner of the lock.
 - Die: The claiming transaction is rolled back
- First term: Claiming transaction is older than the transaction holding the lock
- Second term: Claiming transaction is younger than the transaction holding the lock

→ Wait/Die: Claiming transaction waits if it is older than the transaction holding the lock. It rolls back if it is younger than the transaction holding the lock.

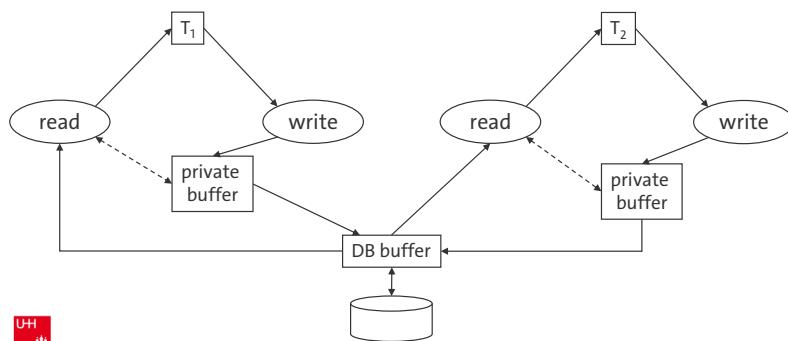
→ Wound/Wait: Claiming transaction tries to roll back the transaction holding the lock if the claiming transaction is older. Claiming transaction waits if it is younger than the transaction holding the lock.

Optimistic Synchronization



Optimistic Synchronization (OCC)

3-phase processing



Validation Strategies:

Backward Oriented (BOCC)

Forward Oriented (FOCC)

- Also called “Optimistic concurrency control” (OCC)
- Used when only few conflicts are expected
- No deadlocks
- Easy reset of transactions
- Enables higher degree of parallelism than pessimistic locking
- Keeps track of the **read set (RS)** and **write set (WS)** of each transaction for the validation phase
 - RS: Elements a transaction reads
 - WS: Elements a transaction writes

Read phase

- Executes the transaction
- Changes are stored in a private buffer

Validate phase

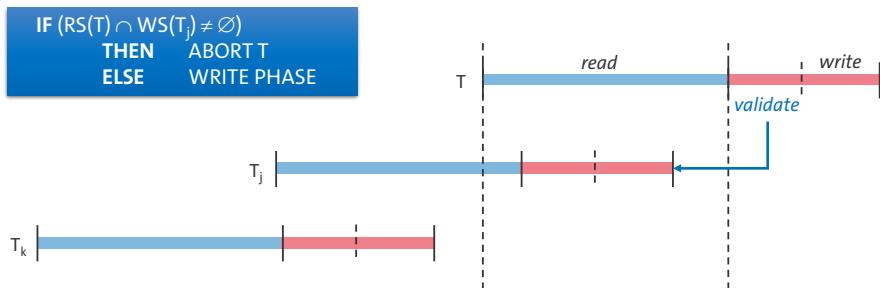
- Check if there are conflicts between parallel transactions (read/write or write/write)
- Resolve conflicts by rolling back transactions
- A transaction can only validate successfully if all changes it has read were from validated transactions
 - Validation sequence defines sequence of serialization

Write phase

- Only when validation is successful
- Reading transactions are done without additional work
- Writing transaction adds log information and propagates the changes (from private buffer to DB)

Backward Oriented Concurrency Control (BOCC)

- Validate against all transactions which have successfully validated since BOT



- T validates against T_j because T_j successfully validated while T was in the read phase
- T does not validate against T_k because T_k was successfully validated before T started

Disadvantages

- High number of comparisons in validation phase
- High risk of long running transactions to be rolled back
- Roll back only at EOT → Unnecessary work was done
- Requires to keep the WS of already finished transactions
- Can lead to unnecessary rollbacks because of faulty conflict detection

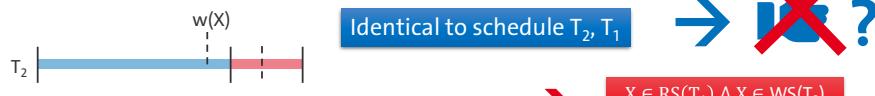
Example for Unnecessary Rollback in BOCC



Case 1: T2 commits after $r_1(X)$



Case 2: T2 commits before $r_1(X)$



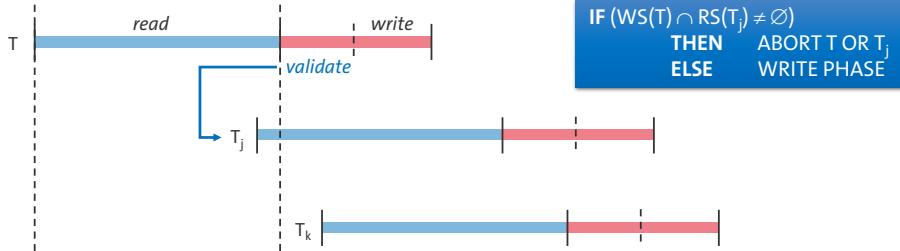
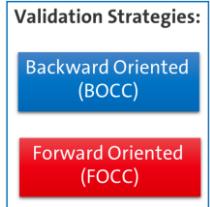
Variation: BOCC+

X gets a time stamp

- ➔ BOCC+ can decide whether X is accessed before or after the commit of T_2
- ➔ T_1 would not have to be rolled back in case 2 with BOCC+

Forward Oriented Concurrency Control (FOCC)

- Only writing transactions validate against running transactions



- T validates against T_j because T_j was running during the read phase of T
- T does not validate against T_k because T_k starts after the read phase of T has finished

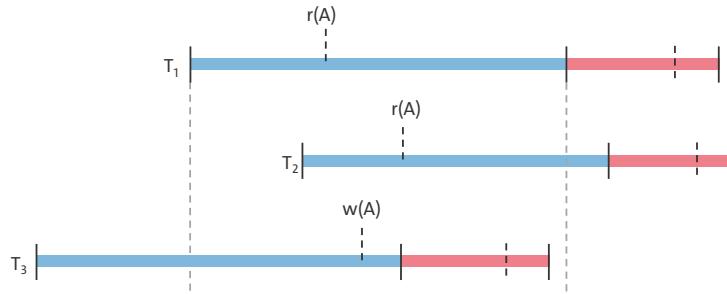
Advantages

- Early rollback possible → Less unnecessary work than in BOCC
- Choice which transaction to abort:
 - Kill approach: the transactions which are still running are aborted
 - Die approach: The validating transaction “dies” (is aborted)
- No need for keeping track of write sets, less work in validation phase
!Track number of rollbacks for each transaction to avoid that individual transactions are rolled back forever → High risk for long running transactions

Disadvantages

- Still a high number of rollbacks possible
- During validation and write phase, $WS(T)$ must be “locked”, such that $RS(T_j)$ does not change

Exercise: Validate T_1 using BOCC and BOCC+



BOCC

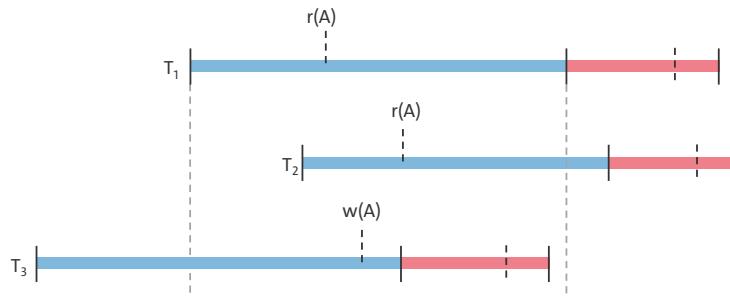
Validate against all transactions which have successfully validated since BOT

BOCC+

Add time stamp to elements

IF $(RS(T) \cap WS(T_j) \neq \emptyset)$
THEN ABORT T
ELSE WRITE PHASE

Exercise: Validate T_3 using FOCC



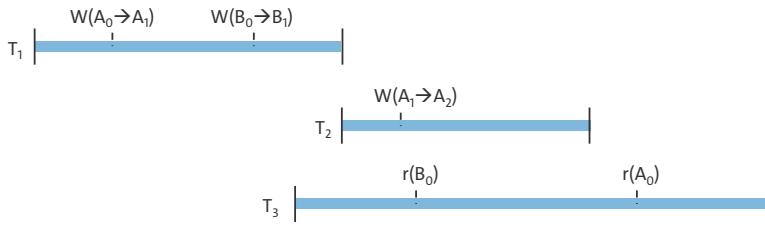
FOCC

Only writing transactions validate against running transactions

IF ($WS(T) \cap RS(T_j) \neq \emptyset$)
THEN ABORT T OR T_j
ELSE WRITE PHASE

Multiversion Concurrency Control (MVCC)

- Each object gets a version number
 - Each change to the object increases the version number
- No synchronization of reading transactions



- Widely used approach, e.g. in PostgreSQL, Sap Hana, Oracle
- Risk of reading older versions of objects but smaller risk of discarded work done → no rollback
- Needs additional storage & maintenance, e.g. for version pool management, garbage collection
- Different methods for realization, e.g.:
 - Transactions read the version that is valid at their BOT/older than the time stamp of their BOT
 - Versions of an object are linked to easily find the most recent version before BOT
 - Version can be removed from the pool as soon as there is a newer version with a time stamp that is smaller than the oldest BOT of a reading transaction
 - Example on slide: T3 reads older versions because these were valid at its BOT, i.e. the newer versions were not committed, yet

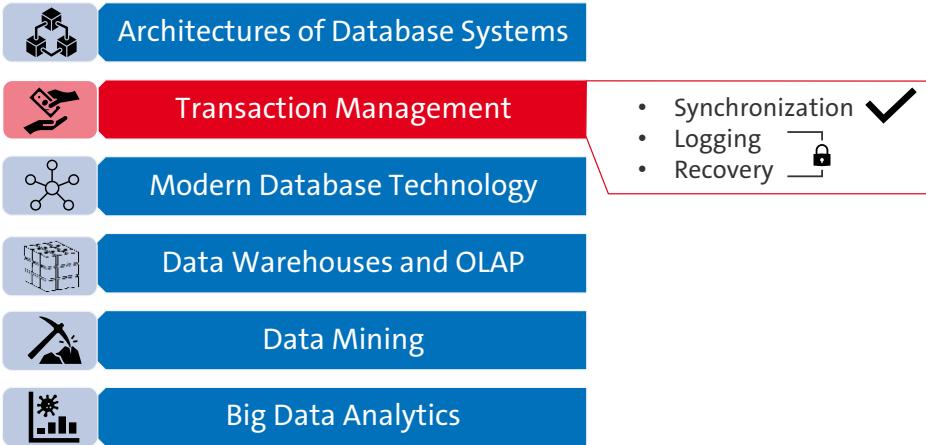
!There are more methods for concurrency control, e.g. predicate locks (address the phantom problem), Locking on B*-trees, Escrow method, MVCC with

snapshot isolation,...

→Further reading for the interested audience

- Cahill, Michael J., Uwe Röhm, and Alan D. Fekete. "Serializable isolation for snapshot databases." *ACM Transactions on Database Systems (TODS)* 34.4 (2009): 1-42.
- O'Neil, Patrick E. "The escrow transactional method." *ACM Transactions on Database Systems (TODS)* 11.4 (1986): 405-430.
- Eswaran, Kapali P., et al. "The notions of consistency and predicate locks in a database system." *Communications of the ACM* 19.11 (1976): 624-633.
<https://dl.acm.org/doi/pdf/10.1145/360363.360369>

Course Outline



Deadlocks

Summary

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

Detect

Avoid

**Wait/Die
Wound/Wait**

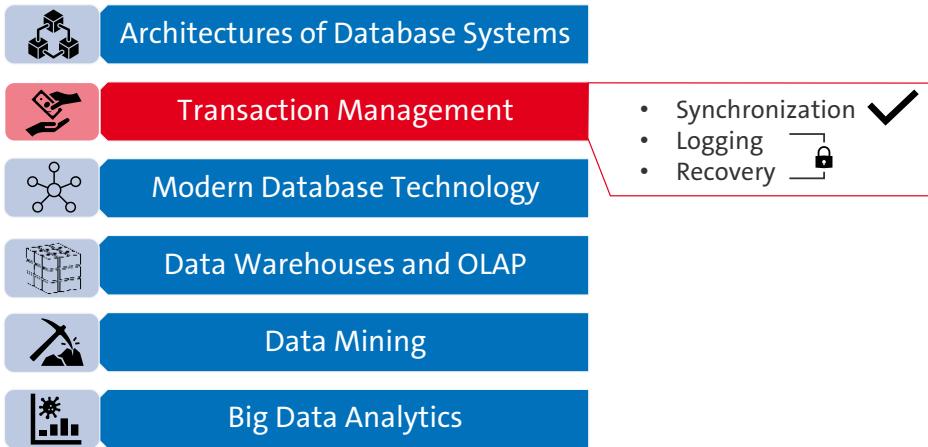
Optimistic Concurrency Control

Backward Oriented (BOCC) (BOCC+)

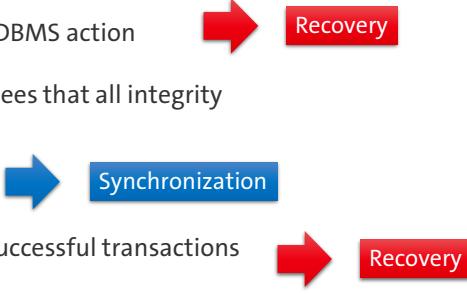
Forward Oriented (FOCC)

Multi Version Concurrency Control (MVCC)

Course Outline



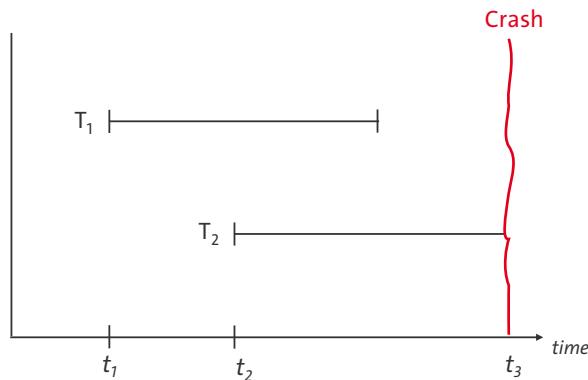
Logging & Recovery

- **Atomicity**
→ “All or nothing” property of any DBMS action
 - **Consistency and semantic integrity**
→ A successful transaction guarantees that all integrity requirements are met
 - **Isolated execution**
→ “Logical single user mode”
 - **Durability**
→ Requires that modified data of successful transactions must survive any type of failure
- 

Synchronization alone is not sufficient to ensure consistency and semantic integrity

→ Additional checks must be performed to check the existing set of rules, e.g. unique entries, type of entries, NULL values,...

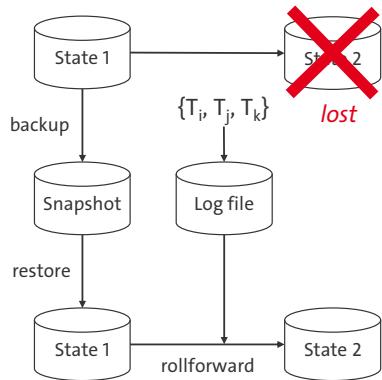
Logging & Recovery Example



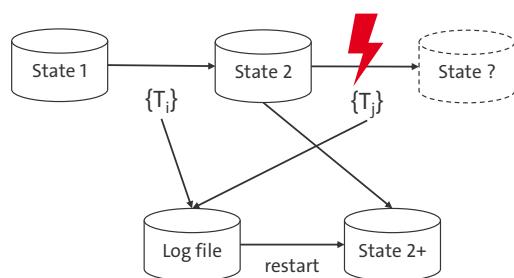
- All changes of the finished transaction T_1 must be in the database → Durability
- All changes of T_2 (only partially executed) must be removed completely from the database → Atomicity
- Restore the most recent transaction-consistent state of the database → Recovery
 - Roll back all changes of transactions that didn't commit
 - Changes of committed transactions might have to be reapplied (if the crash affected the writing of the results to the database)
- Recovery uses backups and log files

Snapshots and Log Files

Example 1: Failure of persistent memory



Example 2: Failure of main memory



- Snapshots are backups of the whole database
- Log files store all changes of the base data
 - Redundant information collected during normal operation
- Log files and snapshots should not be stored on the same machine to avoid complete loss of data in case of failure
- Example 1: Snapshot and log files are used to restore state 2
- Example 2: Restart restores state 2 and the changes of all committed transactions
 - T_j did not commit, changes of T_j are not restored

Types of Failure

Transaction Failure

- Violation of system restrictions, e.g. security regulations or excessive claim of system resources
- “Layer-8 error”, e.g. wrong values
- ROLLBACK

System Failure

System crash with loss of data in main memory → e.g. caused by database system, operating system, hardware, power blackout

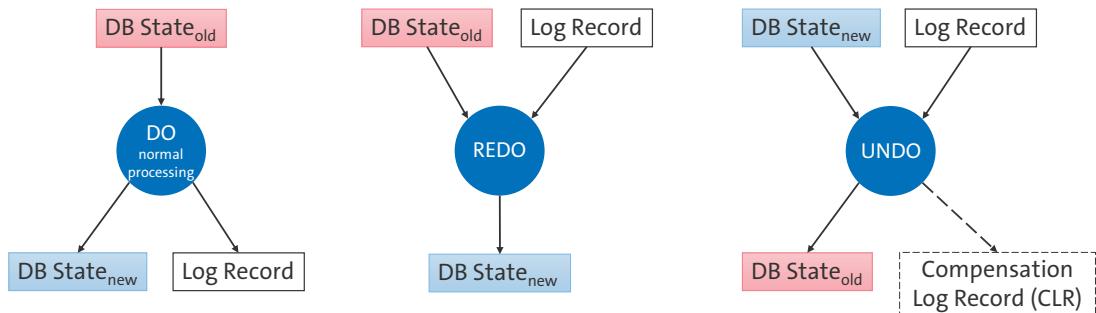
Device Failure

Destruction of secondary storage

Disaster

Destruction of the datacenter

DO-REDO-UNDO



- REDO and UNDO are used to recover a consistent DB state after one or multiple a crashes
- REDO: Re-create changes
- UNDO: Rollback changes
- CLR records undo operations
 - Used in case of a (second) crash during recovery
 - If the system crashes during recovery, rollbacks might not have finished, so a second recovery cannot start from the same state as the first recovery

Recovery Classes



Local failure in an uncommitted transaction
R1-Recovery, partial/local undo



Failure with loss of main memory
R2-Recovery, partial redo
R3-Recovery, global undo



Failure with loss of persistent memory
R4-Recovery, global redo

Different recovery classes for different kinds of failure

R1

- Reasons: Bug in client application, abort, transaction canceled by system → Transaction Failure
- Effect of transaction must be undone

R2 + R3

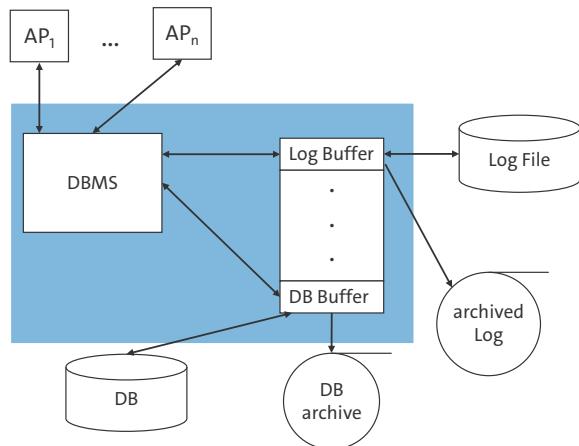
- Reasons: Power outage, hardware failure, bugs or newly updated “features” of the operating system (e.g. planned system shutdown for maintenance) → System Failure
- Committed transactions must be preserved (R2)
- Uncommitted transactions must be rolled back (R3)

R4

- Reasons: natural disaster, head crash (on traditional HDDs) → Device Failure and Disaster
- Recovery via archival copy of backup and log archive in remote system

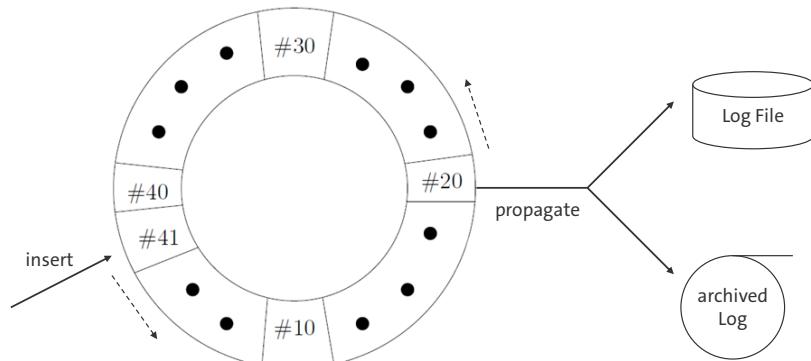
Not formally classified: R5 (log data damaged), R6 (external intervention, e.g. manual treatment and compensation transactions)

System Components for Recovery



- Log buffer in main memory, propagation at commit (at the latest)
- Temporary Log data for handling transaction failures and system failures
- Archived data for device failures and disasters (if archive was stored remotely)

The Log Buffer



- Organized as a ring buffer
- Flushed to disc when full or TA committed → Continuous propagation at the end of the buffer → evenly distributed workload
- Usually smaller than the DB buffer (see last lecture for an example usage of a DB buffer)

Durability modes

Guaranteed durability

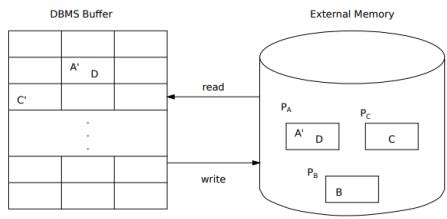
At commit time, the relevant portion of the transaction log is flushed to disk. This log flush operation makes that transaction, and all previously-committed transactions, durable.

Delayed durability

In delayed durability mode, as in guaranteed durability mode, each transaction enters records into the in-memory transaction log as it makes modifications to the database. However, when a transaction commits in delayed durability mode, it does not wait for the transaction log to be posted to disk before returning control to the application. Thus, a non-durable transaction may be lost in the

event of a database failure. However, they execute considerably faster than durable transactions. Eventually, transactions are flushed to disk by the database's subdaemon process or when the in-memory log buffer is full.

The Database Buffer



Combinations of replacement strategies and their effect on recovery

replacement strategy	propagation strategy	
	FORCE	¬FORCE
-STEAL	No Redo No Undo	Redo No Undo
STEAL	No Redo Undo	Redo Undo

FORCE + ¬STEAL appears to be the most useful combination
BUT forced propagation at EOT is expensive.

- Buffer management in a transactional system → cont. buffer management in DB Architecture lecture (1st slideset), but this time we are not only reading pages
- During page access, the page is fixed, i.e. it cannot be replaced
- If data is changed, the page is marked as “dirty” → When and how do we propagate the changes, so we can replace dirty pages?

Replacement Strategies

Steal: Dirty pages are pushed to secondary storage

→ No waiting until the EOT to replace a page. New page “steals” the space of another page

- Advantage: High flexibility for replacing pages
- Disadvantage: Requires UNDO information written to log file before dirty pages are inserted (Write-Ahead-Log/WAL)

NoSteal/¬Steal: Dirty pages are not replaced

→ Only pages of successfully completed transactions are pushed

- Advantage: No UNDO-recovery information in secondary storage necessary
- Disadvantage: Long update transactions with high number of updates might “clog” the buffer

Propagation Strategies

Force: All changed pages are propagated at commit (forced to propagate)

- Advantage: No REDO necessary after failure
- Disadvantages: Many write operations → high overhead
 - Large buffers not optimally used
 - Long response time for update transaction

NoForce/¬Force: Pages can be propagated after commit, e.g. because a page is used frequently

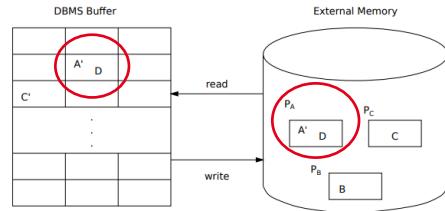
- Requires collection of all necessary REDO-information before a commit so that changes which are not materialized can be recovered
- Advantage: No write through of changes at EOT, only REDO information in log file
 - Disadvantage: Requires REDO recovery after failure

In case of failure:

- Uncommitted transactions → All changes that were already made persistent must be rolled back (UNDO)
- Committed transactions → Retrace all changes that have not been propagated (REDO)

Exercise

Step	T ₁	T ₂
1	BOT	
2		BOT
3	r(A)	
4		r(D)
5		w(D)
6	w(A)	
7	c	



Is there a conflict with this schedule using one of the following combinations? If there is, which one?

- | | |
|---|----------------|
| A | -STEAL, FORCE |
| B | -STEAL, -FORCE |
| C | STEAL, -FORCE |
| D | STEAL, FORCE |

Hint: A and D are on the same page.

Insertion Strategies

- Update-in-Place
 - Each page is always written to the same block on disc/the same address on the persistent memory → This what we did on the last slide
- Twin-Block
 - A page P is assigned to two pages in secondary storage, P_0 and P_1 . They contain the two latest states. Actual “insertion” into DB is done later.
- Shadow Storage
 - Like Twin-Block, but only changes paged are duplicated → Less redundancy

Update-In-Place is direct

Twin-Block and Shadow Storage are indirect

System Configuration for Following Discussion

The following discussion of recovery in DBMS assumes the most general configuration which is also the hardest for recovery.



Steal: Pages which are not fixed can be replaced at all times (and have then to be propagated if they were changed)

¬Force: Changes are propagated continuously

Update-in-Place: Each page refers to exactly one page in secondary storage

Small lock granulates: Transactions can lock and change objects smaller than a page exclusively → Pages can contain changes of committed and uncommitted transactions at the same time

A Fairy Tale About Logs

“Hansel and Gretel left behind a trail of crumbs which would allow them to retrace their steps (by following the trail backwards) and would allow their parents to find them by following the trail forwards. This was the first undo and redo log. Unfortunately, a bird ate the crumbs and caused the first log failure.”



Jim Gray



Also known as one of the people who developed the first relational DBMS and the precursor of SQL.

He is also the guy who described the ACID properties, developed the cube operator (we will talk about this in the data warehousing lectures), and a few more.

Log Types

Logical Logging

UNDO operation to create previous state

REDO operation to create next state

Example

UNDO: A=A+10

REDO: A=A-10

- Before-image is created by applying UNDO operation
- After-Image is created by applying REDO operation

Physical Logging

Before-Image instead of UNDO operation to recover previous state

After-Image instead of REDO operation to create next state

Example

Before Image: A = 20

After Image: A = 10

Physical Logging: inflexible w.r.t. inserting or deleting operations

Both approaches can be combined → “physiological” Logging (physical on page level, logical within a page)

Anatomy of a Log Entry

[Log Sequence Number (LSN), TransactionID, PageID, REDO information, UNDO information, Previous LSN]

[#4, T₂, P_C, C+=100, C-=100, #2]

Example	#	T ₁	T ₂	Log [LSN, TA, PageID, Redo, Undo, PrevLSN]
	1.	BOT		[#1, T ₁ , BOT, 0]
	2.	r(A, a ₁)		
	3.		BOT	[#2, T ₂ , BOT, 0]
	4.		r(C, c ₂)	
	5.	a ₁ :=a ₁ -50		
	6.	w(A, a ₁)		[#3, T ₁ , P _A , A-=50, A+=50, #1]
	7.		c ₂ :=c ₂ +100	
	8.		w(C, c ₂)	[#4, T ₂ , P _C , C+=100, C-=100, #2]
	9.	r(B, b ₁)		
	10.	b ₁ :=b ₁ +50		

- Log entry written for every BOT, commit, write
- PrevLSN refers to the previous LSN of the same transaction
- New logging information is added at the end of file

Exercise

Continue the log file!

Which type of logging is used?

#	T ₁	T ₂	Log [LSN, TA, PageID, Redo, Undo, PrevLSN]
1.	BOT		[#1, T ₁ , BOT, 0]
2.	r(A, a ₁)		
3.		BOT	[#2, T ₂ , BOT, 0]
4.		r(C, c ₂)	
5.	a ₁ :=a ₁ -50		
6.	w(A, a ₁)		[#3, T ₁ , P _A , A-=50, A+=50, #1]
7.		c ₂ :=c ₂ +100	
8.		w(C, c ₂)	[#4, T ₂ , P _C , C+=100, C-=100, #2]
9.	r(B, b ₁)		
10.	b ₁ :=b ₁ +50		
11.	w(B, b ₁)		
12.	commit		
13.		r(A, a ₂)	
14.		a ₂ :=a ₂ -100	
15.		w(A, a ₂)	
16.		commit	



Let's assume that at BOT A=B=C=0. What would the log entry look like with physical logging?

Before or After Image?

During recovery, we have to find out which operations of a transaction have already been propagated.

Log record:

[LSN,...]

Page in buffer:



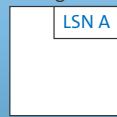
- When creating a Log record, the LSN is stored at a dedicated address of the page
→ When the page is propagated, the LSN is also copied

During Recovery:

Log record:

[LSN A',...]

Page in secondary storage:



LSN A' > LSN A → A is a before image, propagate page (after image) to secondary memory

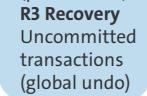
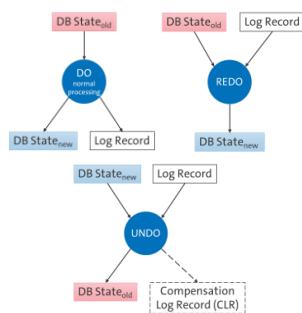
LSN A' <= LSN A → After image was already propagated, do nothing

Recovery



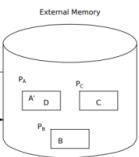
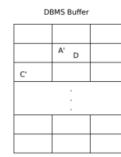
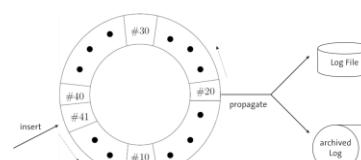
Next week

Summary



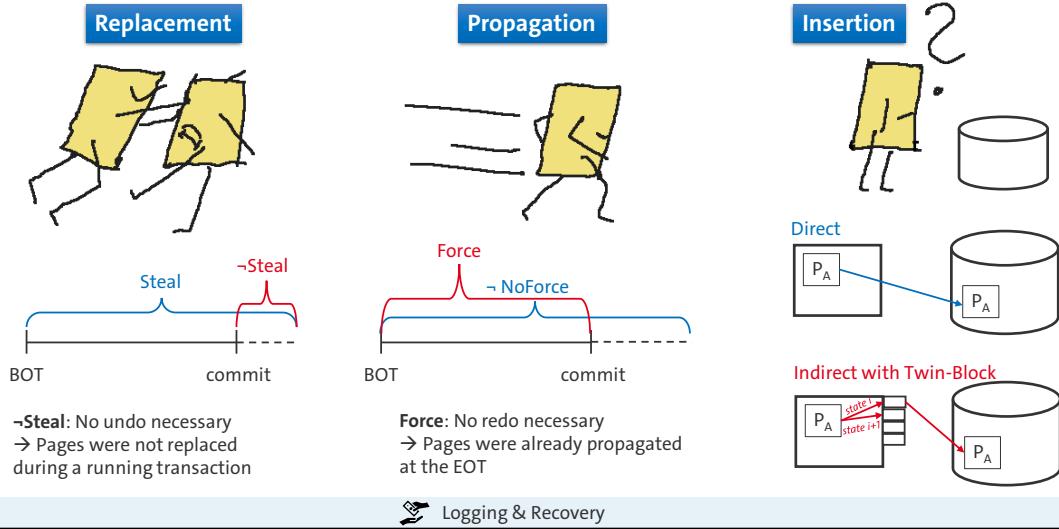
R4 Recovery
(snapshot +) global redo

Example	#	T ₁	T ₂	Log [LSN, TA, PageID, Redo, Undo, PrevLSN]
	1.	BOT		[#1, T ₁ , BOT, 0]
	2.	r(A, a ₁)		
	3.		BOT	[#2, T ₂ , BOT, 0]
	4.		r(C, c ₂)	
	5.	a ₁ :=a ₁ -50		[#3, T ₁ , P _A , A-=50, A+=50, #1]
	6.	w(A, a ₁)		
	7.		c ₂ :=c ₂ +100	[#4, T ₂ , P _C , C+=100, C-=100, #2]
	8.		w(C, c ₂)	
	9.	r(B, b ₁)		
	10.	b ₁ :=b ₁ +50		



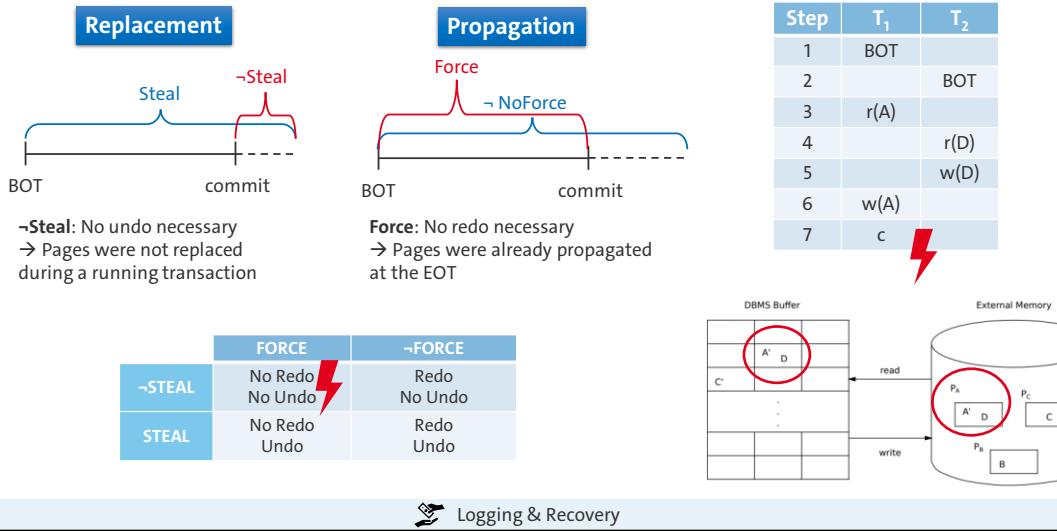
**Replacement,
Propagation,
and Insertion
Strategies**

Recap replacement, propagation, and insertion strategies



Force writes changes at commit time at the latest → No indirect insertion strategy possible

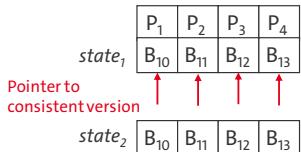
Recap replacement, propagation, and insertion strategies



- Changes of T₁ on page P_A are forced to propagate at commit in step 7 because of FORCE
- T₂ has not committed, yet → Because of ¬STEAL, page P_A (which includes element D that T₂ is working on) must not be replaced while T₂ is still running

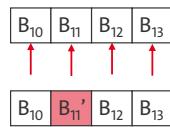
Recap Indirect Insertion with Redundancy (Twin-Block)

Mapping of pages to blocks:

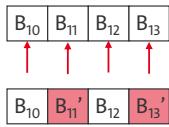


Operations:

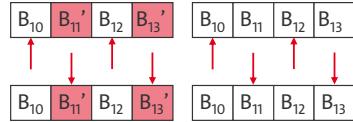
T₁: w(P₂)



T₁: w(P₄)



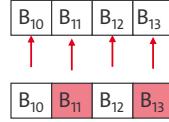
T₁: Commit



- Copy B₁₁'
- Switch pointer

Current state: Like initial state but with switched pointer

T₁: Abort



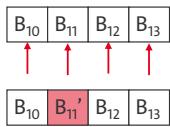
- Copy B₁₁
- Do NOT switch pointer

Current state:
Like initial state

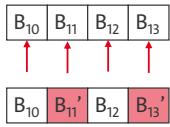
Recap Indirect Insertion with Redundancy (Twin-Block)

Operations:

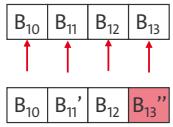
T₁: w(P₂)



T₁: w(P₄)

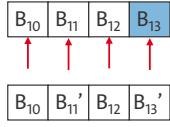


T₂: w(P₄)



Can cause Lost Update
→ Wait for T₁ to commit first

T₂: r(P₄)



T₂ reads the consistent version

Exercise

P ₁	P ₂	P ₃	P ₄
B ₁₀	B ₁₁	B ₁₂	B ₁₃

Assuming each page can only hold a single integer number and P₁ = P₂ = P₃ = P₄ = 1. What would the Twin Block look like after the following operations?

T1: w(P₂=5)

T1: r(P₁)

T2: r(P₂)

T1: A = P₁+3

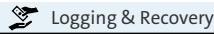
T1: w(P₃=A)

T2: w(P₄=P₂)

T1: Commit

T2: Commit

6



Logging & Recovery

Difference to Shadow Storage

- Shadow storage only creates duplicate if data is changed, i.e. here only for B₁₁ and B₁₃
- Advantage: Less memory consumption
- Disadvantage: Fragmentation (addresses are scattered)

Log Data

[Log Sequence Number (LSN), TransactionID, PageID, REDO information, UNDO information, Previous LSN]
[#4, T₂, P_C, C+=100, C-=100, #2]

Rules for writing Log data

- Write Ahead Log (WAL) principle for UNDO information
→ Log entries must be written to file before an affected page is replaced
- Force-Log-at-Commit
→ Redo information must be written before the commit

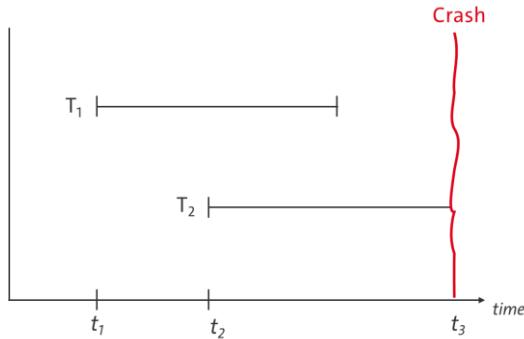
WAL

- Only relevant for STEAL → NOSTEAL does not replace pages during transaction
- Important for direct insert strategies

Force-Log-at-Commit

- Required for Crash-Recovery with NOFORCE (i.e. when changes might not have been propagated)
- Required for R4 recovery (with FORCE and NOFORCE) → even if changes were propagated, they were lost
- For direct and indirect insertion strategies

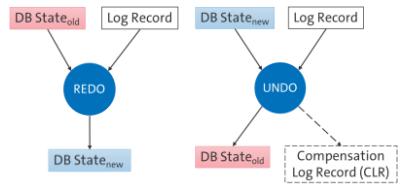
Recovery



Transaction T₁ is a winner → Redo
Transaction T₂ is a loser → Undo

Phases of Recovery

1. **Analysis**
→ Identify winners and losers
2. **Redo**
→ Repetition of history
→ Reads log file forwards
3. **Undo** of losers
→ Reads log file backwards
→ Write CLR



10

Logging & Recovery

Analysis

- Started transactions marked by BOT
- WAL important to not lose this log information in case of a crash, e.g. to not lose a commit that happened but was not entered in the log file
- Winners: Transactions with commit entry
→ T₁ is a winner
- Losers: Transactions without commit entry
→ T₂ is a loser

REDO

- For winners and losers
- Go forward in log file
- Fetch referenced page from secondary storage into buffer
- Compare LSN of page with LSN in log (see slide 22 in last lecture)
 - $LSN_{page} \geq LSN_{log}$: Ok
 - $LSN_{page} < LSN_{log}$: Redo and update LSN of page
- Propagate page

UNDO

- For all loser transactions irrespective of the LSN
- Go backwards through log file
- Skip all entries of winner transactions
- For all loser transactions:
 - Fetch referenced page from secondary storage into buffer
 - Execute undo operation
 - Write CLR
- Propagate Page

Fault Tolerance of Recovery

How to make sure that another crash during the recovery does not change the result of the recovery?
→ REDO and UNDO phase must be idempotent (result must always be the same irrespective of how often the operation has been applied)



Idempotence of REDO
LSN prevents repeated REDO



Idempotence of UNDO
Compensation Log Record for every UNDO operation

For each operation (that changes data), the following must be fulfilled:

- $\text{undo}(\text{undo}(\dots\text{undo}(a)\dots)) = \text{undo}(a)$
- $\text{redo}(\text{redo}(\dots\text{redo}(a)\dots)) = \text{redo}(a)$

An object a with an operator \circ that fulfills the property $a \circ a = a$ is called idempotent in respect of operator \circ .

Compensation Log Records

<LSN, TransactionID, PageID, Redo information, PrevLSN, UndoNxtLSN>

No undo information

<#7', T₂, PA, A+=100, #7, #4>

LSN: Modified version of according log entry used for undo

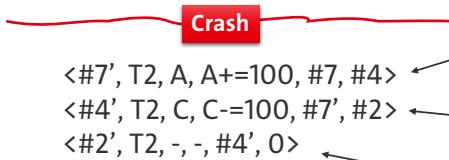
Redo information: Successful undo during recovery

PrevLSN: Previous log entry, can be a CLRs

UndoNxtLSN: Refers to the next change that has to be undone

Recovery Example

[#1, T₁, BOT, 0]
 [#2, T₂, BOT, 0]
 [#3, T₁, A, A-=50, A+=50, #1]
 [#4, T₂, C, C+=100, C-=100, #2]
 [#5, T₁, B, B+=50, B-=50, #3]
 [#6, T₁, commit, #5]
 [#7, T₂, A, A-=100, A+=100, #4]



13

Analysis

Winner: T₁
Loser: T₂

Redo

IF LSN(A) < 3 THEN A-=50 (and replace LSN(A))
 IF LSN(C) < 4 THEN C+=100 (and replace LSN(C))
 IF LSN(B) < 5 THEN B+=50 (and replace LSN(B))
 IF LSN(A) < 7 THEN A-=100 (and replace LSN(A))

Undo → Only for losers (T₂)

Entry #7

- A+=100
- Write CLR
- LSN(A) = #7'

Entry #4

- C-=100
- Write CLR
- LSN(C) = #4'

Entry #2

- Write CLR

- No operation done in BOT and EOT → Only CLR entry required in undo phase
- If checkpoint exists: Redo can start at state of checkpoint, else Redo starts at start of DB

Recovery Exercise

[#1, T2, BOT, 0]

[#2, T2, B, B=B-1, B=B+1, #1]

[#3, T1, BOT, 0]

[#4, T1, A, A=A+2, A=A-2, #3]

[#5, T1, A, A=A*5, A=A/5, #4]

Crash

Analysis

Winners: ?

Losers: ?

Redo

IF LSN(?) < ? THEN ?

...

Undo

For all losers:

Undo operation

CLR entry:

<LSN, TransactionID, PageID, Redo information, PrevLSN, UndoNxtLSN>

<#7', T₂, PA, A+=100, #7, #4>

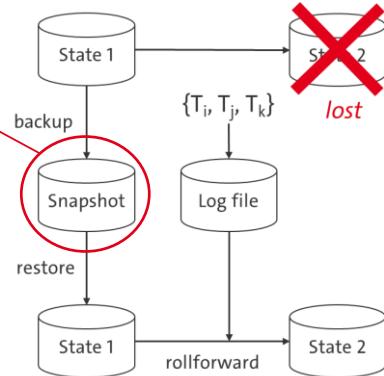
New LSN

Checkpoints

Reduces the amount of REDO operations during recovery after a system failure (or worse)

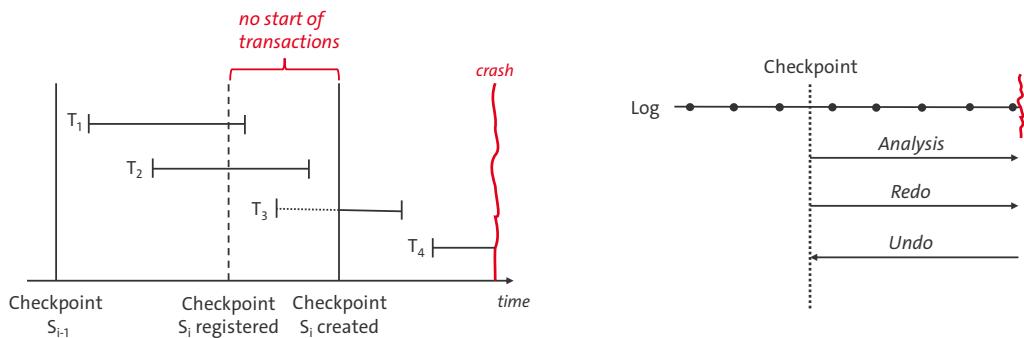
Types of checkpoints

- (Global) transaction consistent save points
- Action consistent save points
- Fuzzy save points



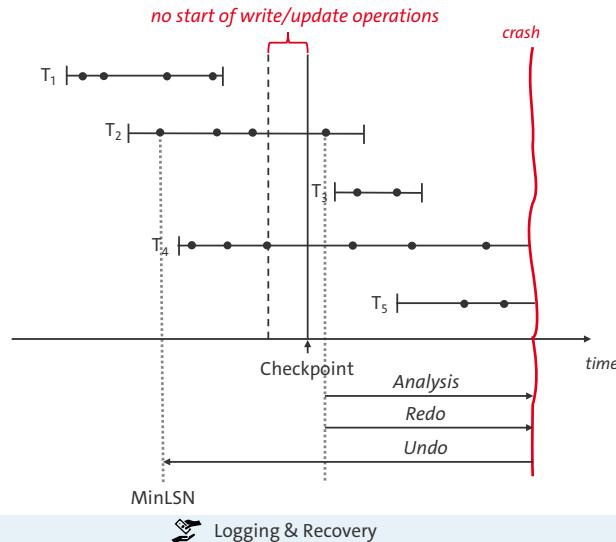
- Without checkpoints, all changes since the start of the database must be redone in case of system failure
- Critical: Changes in hot spot pages (pages which are always held in the buffer because there are many operations using that page)

Transaction Consistent Checkpoints



- Also called “commit consistent”
- Active transactions (when checkpoint is registered) are finished, new transactions must wait until checkpoint was created
- All modified pages and log information are written to secondary memory → Checkpoint contains transaction consistent state (i.e. no half-finished transactions)
- Expensive because of wait times for transactions (=system down time) → Not suited for frequent creation
- Log file requires entries indicating when the checkpoint was registered and created:
 - BEGIN_CHKPT Record
 - END_CHKPT Record
- Log Address of last Checkpoint Record is kept in special system file
- “Transaction Oriented Checkpoint” → Just a different way of stating that FORCE was used
 - Changes are always propagated at the end of transaction
 - No redo necessary

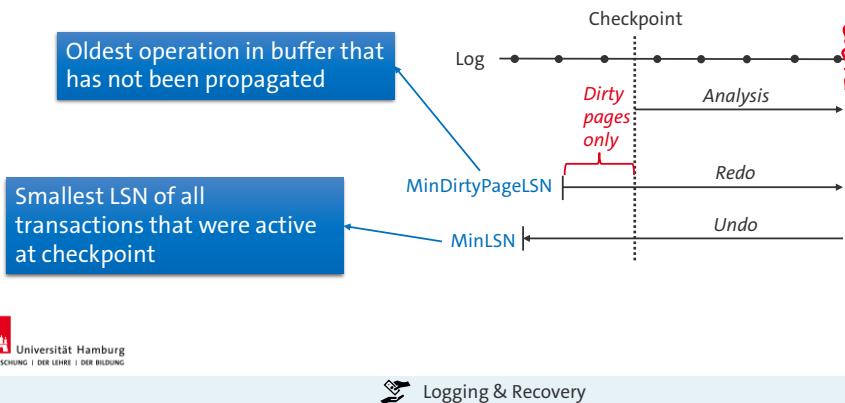
Action Consistent Checkpoints



- Before creating a checkpoint, all operations changing any data must be finished (i.e. any write operations)
- All modified pages are written to secondary memory
- During recovery:
 - No redo information older than checkpoint is required
 - Undo information older than checkpoint still required → back to MinLSN
 - MinLSN = LSN of oldest operation of a transaction that was running during the creation of the checkpoint (here: first operation of T₂)
- Shorter “down time” but potentially more Undo operations in case of recovery

Fuzzy Checkpoints

- Modified pages are not written, only PageID is written



Efficiency of recovery depends on buffer management

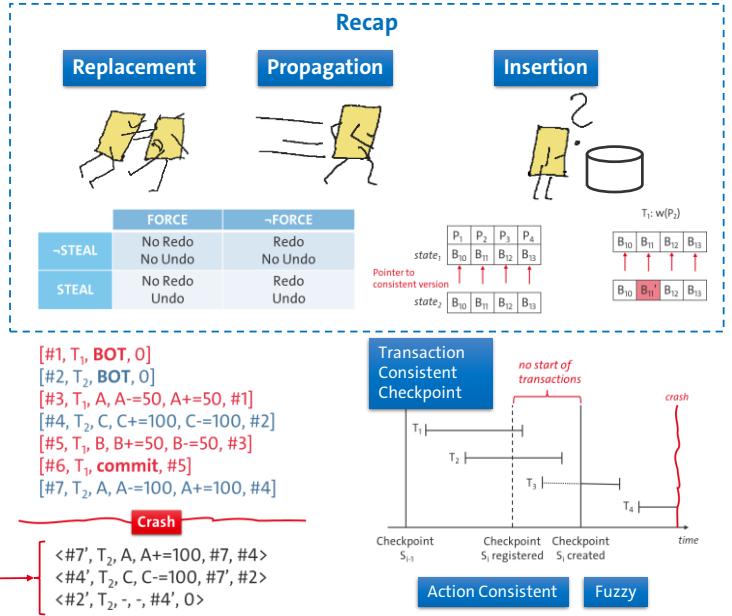
- If hot spot pages remain in the buffer and are never propagated, the redo phase must consider the whole log file since fetching these pages
- Possible solution: Force propagation if a dirty page is part of two consecutive fuzzy checkpoints and not propagated between these two checkpoints

Summary

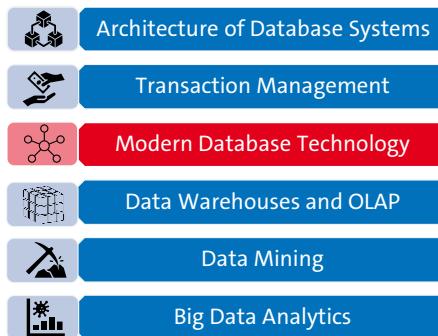


Phases of Recovery

1. **Analysis**
→ Identify winners and losers
2. **Redo**
→ Repetition of history
→ Reads log file forwards
3. **Undo of losers**
→ Reads log file backwards
→ Write CLR

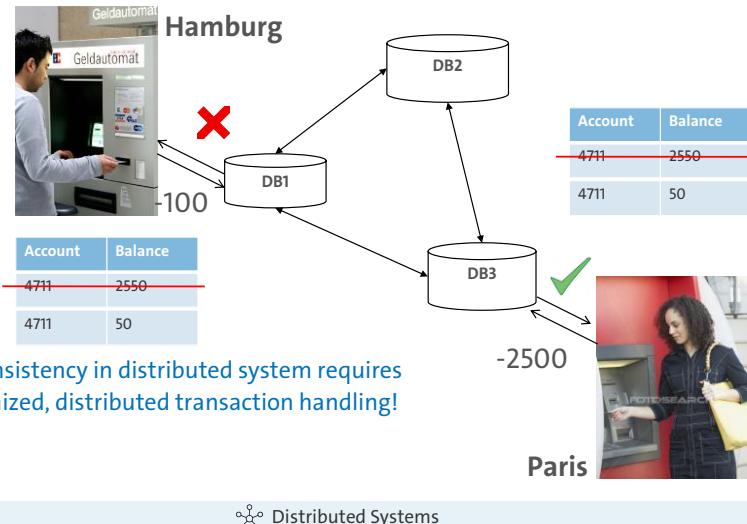


Course Outline



- Distributed Systems
- Optimizations in RDBMS
- NoSQL

Scenario: Distributed Transaction (OLTP)



OLTP = Online Transaction Processing (not to be confused with OTP)

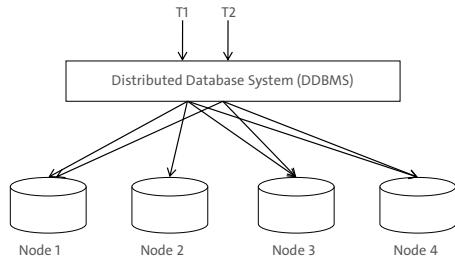
Scenario

- Both clients want to withdraw money at the same time from the same account but at different locations
- Since the balance is not enough to satisfy both requests, one should be denied
- If there was enough to satisfy both requests, both amounts would have to be subtracted from the account balance

Consistency can have different meanings in different contexts, e.g.

- Are distributed replicas of data in the same state?
- Do read operations on distributed data return the same result?
- Are application specific invariants met?

Challenges: Distributed Transaction Management



Global (distributed) transactions

- access/update data at more than one “site” (=node, ...)
- ensure ACID properties during distributed (over multiple nodes) transaction processing

The Atomicity property implies

- Either all nodes must commit, or all must abort
- If any node crashes, all must abort

→ Atomic commitment problem
→ Sounds similar to quorum consensus, but is different

Recall atomicity in the context of ACID transactions

- A transaction either commits or aborts → Atomicity
- If it commits, its updates are durable → Durability
- If it aborts, it has no visible side-effects → Atomicity

Atomic commit vs. Consensus

Consensus

- One or more nodes propose a value
- Any of the proposed values is decided
- Crashed nodes can be tolerated as long as a quorum is working
- There can be different quorum for read and write
- Quorum = Minimum number of nodes in the system that have committed

Atomic commit

- Every node votes whether to commit or abort
- Must commit if all nodes vote to commit, must abort if at least one node votes abort or a participating node crashes

Challenges: Failures in Distributed DBMS

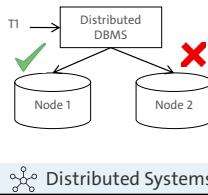
Site Failure

- A node is not available anymore
- Completely or partial

Communication Failure

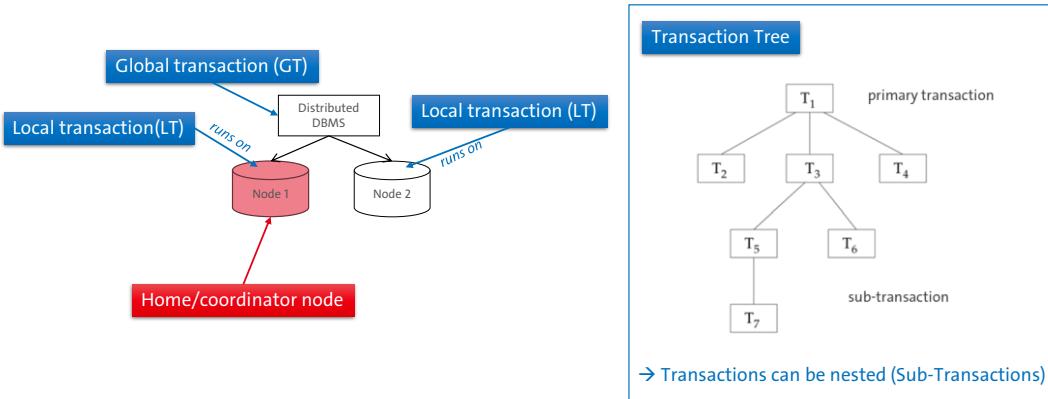
- Failure of a communication link
- Loss of messages/Message corruption/ Message duplication
- Network partitioning

→ Network partitioning and site failures are generally indistinguishable



- A network is defined as being partitioned when it has been split into two or more subsystems that lack any connection between them
- Loss of messages/Message corruption/Message duplication → Handled by network transmission control protocols such as TCP-IP
- Failure of a communication link → Handled by network protocols, by routing messages via alternative links

Terminology



Home/Coordinator node: Node where transaction is started

Local Transaction: Transaction that is completely executed on home node

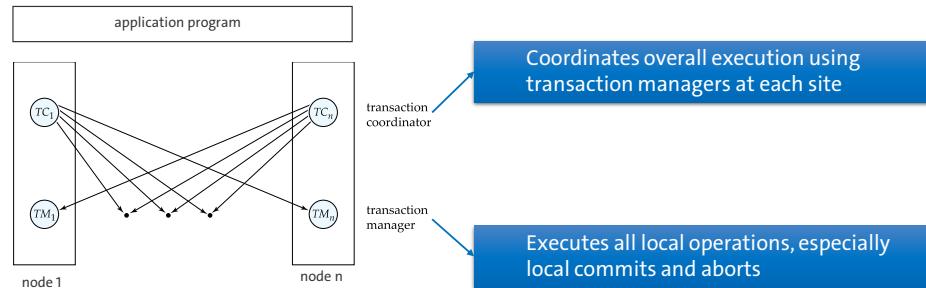
Global Transaction: Transactions that are executed on multiple nodes

Call Hierarchy: Transactions calling each other form a directed, acyclic graph → Transaction tree

Rules for nested transactions

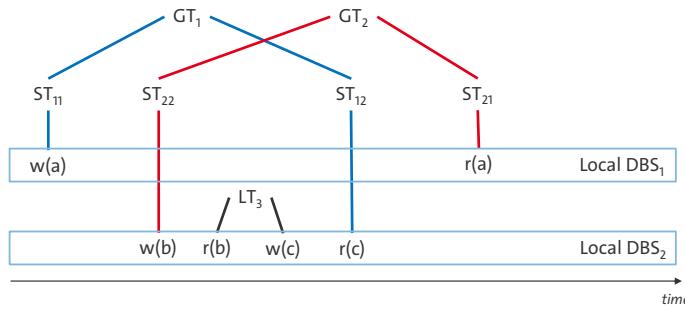
- A transaction may only end, after all sub-transactions are closed
- Every sub-transaction decides independently to finally abort or temporary commit
- In case of an abort of a transaction, all its sub-transactions are aborted as well
- In case a sub-transaction aborts, the parent transaction decides on the final outcome, i.e., may change an abort of a sub-transaction into a global commit
- Sub-transactions may only finally commit after all parent-transactions on the path to the root transaction have finally committed

Transaction Coordinator and Manager



- Each site (node) has a transaction ccordinator and a local transaction manager
- Global transactions can be submitted to any transaction coordinator

Distributed Serializability

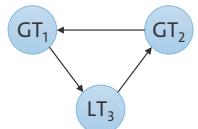


Global dependency
 $GT_1 < GT_2$, because GT_1 accesses *a* earlier than $GT_2 \rightarrow$ Dirty Read



Adding local dependency

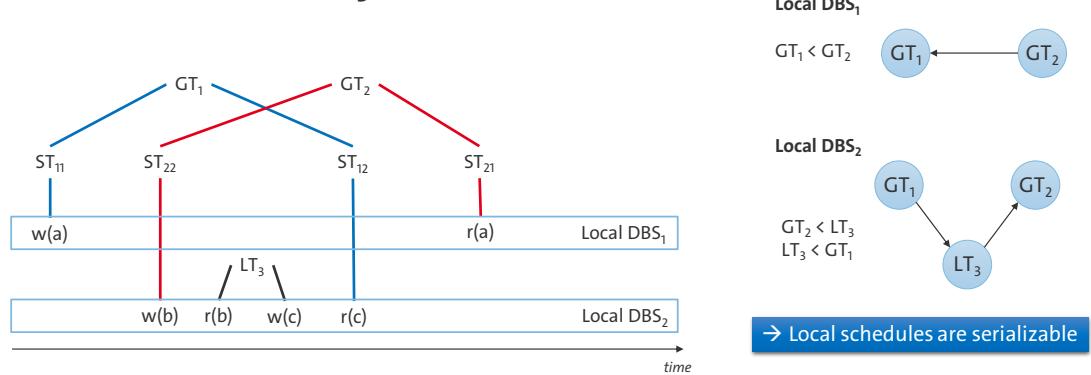
$GT_2 < LT_3 \rightarrow$ Dirty Read
 $LT_3 < GT_1 \rightarrow$ Dirty Read



Some Solutions

- Implement functional limitations
 - Example: restricted set of update capabilities within global transactions
- Provide non-functional limitations
 - Example: relaxing ACID guarantees, e.g. no guarantee of global serializability
- Reduction of autonomy of local systems
 - Do not allow local transactions
 - Mark transactions as global and local and reflect this knowledge within the scheduler
- Reduction of heterogeneity
 - Agree on using compatible synchronization and commit protocols

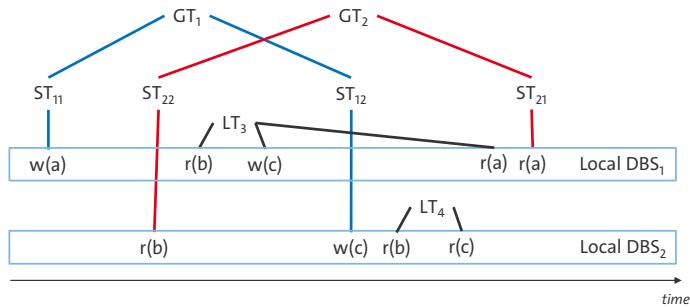
Quasi-Serializability



Quasi-Serializability

- Implementation of weaker correctness criteria
- All local schedules are fully serializable
- Global transactions are executed sequentially

Exercise (Quasi)-Serializability



Atomic Commit Protocols

- Challenge: Ensure atomicity across sites



Failure model:

Nodes may fail independently from each other and implement a fail-recover model

Commit Protocol:

Two-Phase commit enforces integrity of distributed changes

Challenge: Global Atomicity / Single Decision

- All nodes participating in a distributed transaction have to come to a common decision: Either all commit or all abort the transaction
- Atomic commit protocols are used to ensure atomicity across sites → A transaction, which executes at multiple sites, must either be committed at all the sites, or aborted at all the sites
- COMMIT → global transaction is written back on **all** (relevant) nodes
- ABORT → global transaction is rolled back on **all** (relevant) nodes

Conflict: Robustness vs. Efficiency

- Correctness
 - Atomicity (all-or-nothing) and durability have to be guaranteed also in case of failure
- Efficiency
 - Generate as few messages and log entries as possible
- Robustness
 - Non-blocking
 - Each node should be allowed to abort as long possible

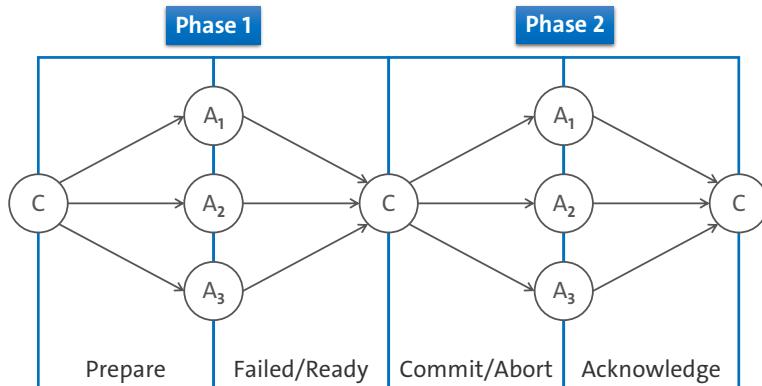
Fail-recover model

- Nodes could crash and at some later date recover from the failure and continue executing normally
- Protocols able to tolerate multiple malicious sites are called byzantine fault-tolerant protocols (which is a research area of its own)

Further reading:

- An example for byzantine fault-tolerant middleware: Luiz, Aldelir Fernando, Lau Cheuk Lung, and Miguel Correia. "Mitra: Byzantine fault-tolerant middleware for transaction processing on replicated databases." *ACM SIGMOD Record* 43.1 (2014): 32-38.
- An early approach to Byzantine Fault-Tolerant Services (not database-specific): Abd-El-Malek, Michael, et al. "Fault-scalable Byzantine fault-tolerant services." *ACM SIGOPS Operating Systems Review* 39.5 (2005): 59-74.
- A recent approach on byzantine fault detection: Yamada, Hiroyuki, and Jun Nemoto. "Scalar DL: Scalable and practical Byzantine fault detection for transactional database systems." *Proceedings of the VLDB Endowment* 15.7 (2022): 1324-1336.

Two-Phase-Commit Protocol (2PC)



- 2PC can be seen as a special case of 2PL for distributed systems
- Global commit is initiated by the transaction coordinator after the last operation of the transaction has been executed
- The protocol involves all the local sites at which the transaction is executed

Phase 1 (PREPARE)

- Assure ability to execute a transaction
- Log changes and commit (After-Image)

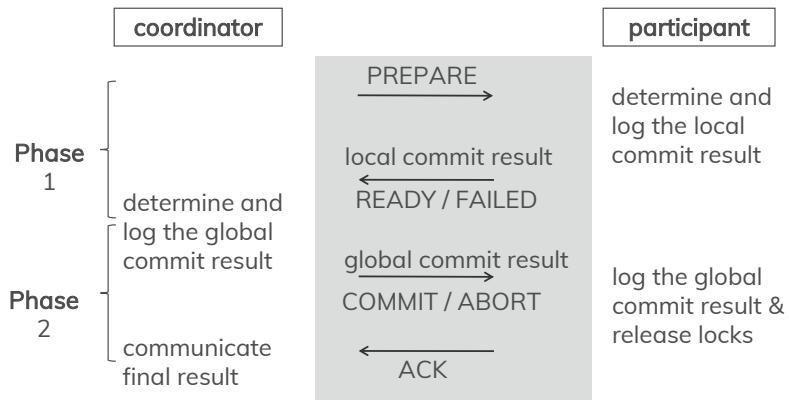
Phase 2 (COMMIT)

- Make changes visible (free locks, delete temp log (UNDO))
- After phase 1, the transaction has a guaranteed outcome

Illustrated Example

- n=4 Participants (includes the coordinator)
→ 4 messages per participant
→ 4(n-1) messages in total

2PC



Step 1: PREPARE

- Coordinator sends PREPARE message to all participants to find out if they can commit the transaction

Step 2: READY/FAILED

- Each participant receives the PREPARE message and sends one of the two messages back
 - READY → If participant is able to commit the transaction, locally
 - FAILED → If participant is not able to commit the transaction, locally (because of failure, consistency, ...)

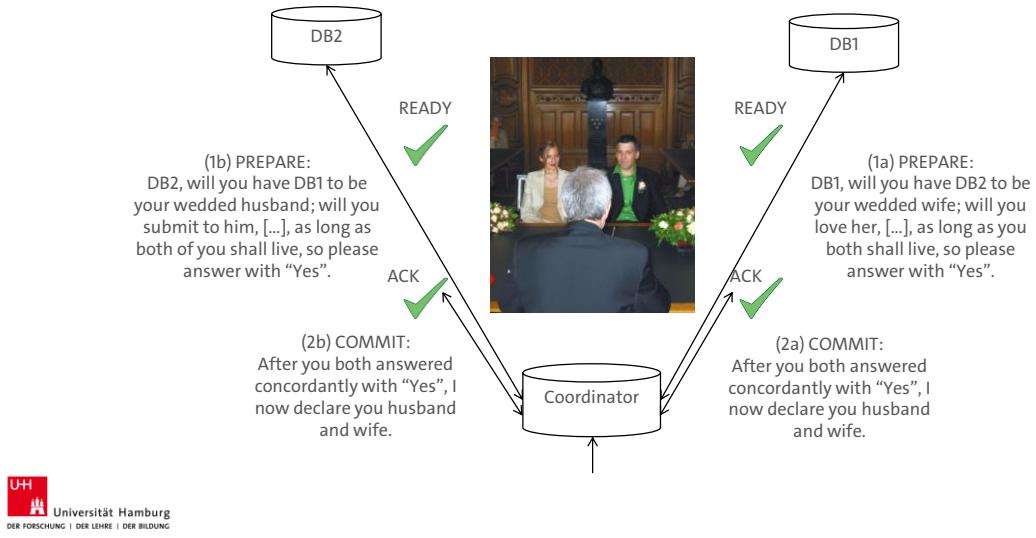
Step 3: COMMIT/ABORT

- If coordinator receives READY back from all participants, the coordinator decides for global COMMIT
- If one of participants answers FAILED or does not answer in time (timeout), coordinator must decide for ABORT
- Coordinator sends decision to all participants for them to COMMIT/ABORT locally

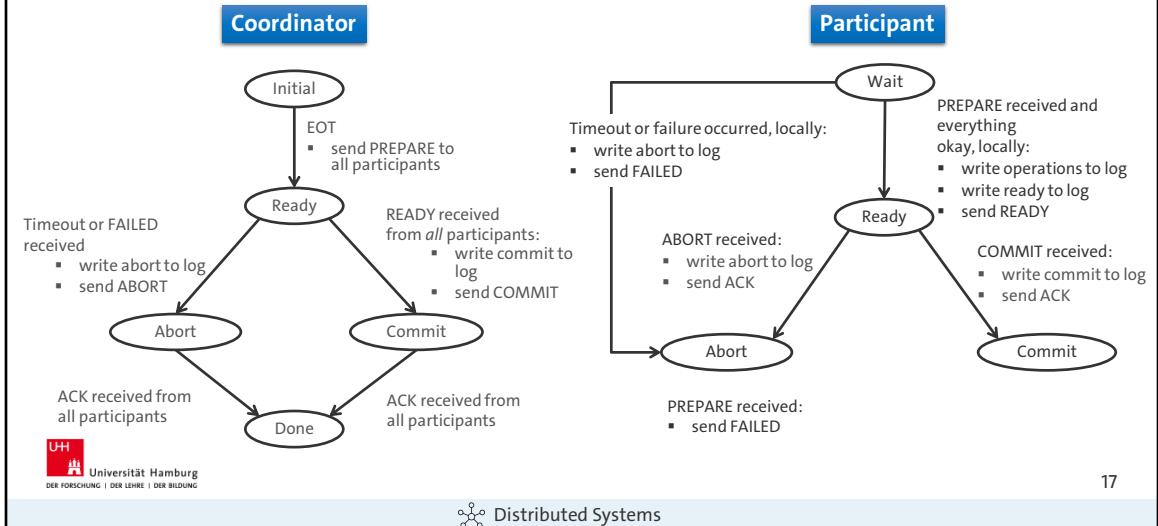
Step 4: ACKNOWLEDGE

- Participants acknowledge reception of coordinator's global decision

Example: “The 2PC Wedding”



2PC state charts



In case of Failures

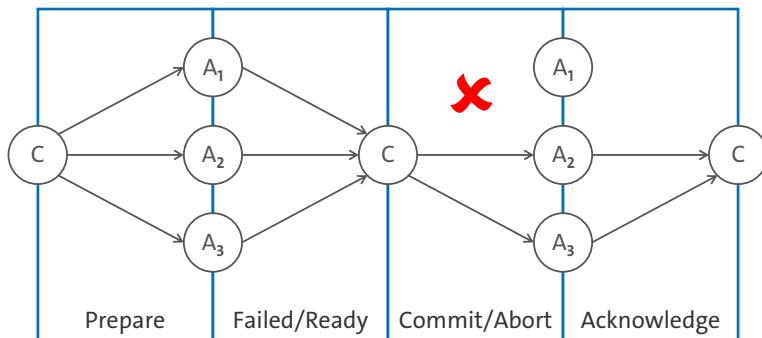
Phase of uncertainty for participants

- From the point a participant replied with READY until the global decision is received
- Participant could actively try to get hold of global decision
→ actively ask coordinator or other participants

Reason of uncertainty situations

- Crash of coordinator
- Crash of other participant (coordinator waits for reply)
- Lost messages

Failure Scenario: Lost Message



Lost in Phase 1

PREPARE message of coordinator gets lost

or

READY/FAILED message of a participant gets lost

→ after timeout coordinator assumes a participant crash and decides for ABORT

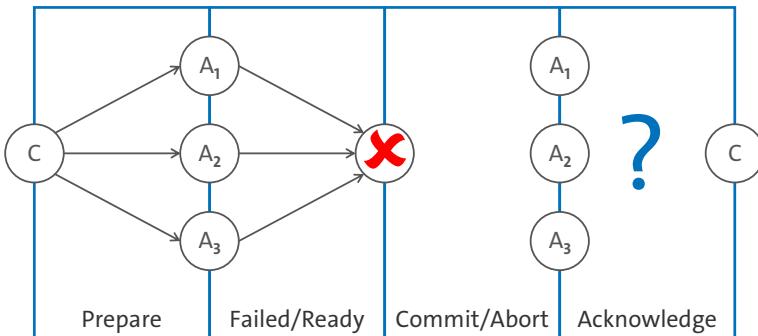
Lost in Phase 2

ABORT/COMMIT message of coordinator is lost, while participants are in READY state

→ participants are blocked, cannot decide on their own

→ participants can only ask coordinator repeatedly for global decision

Failure Scenario: Crash of Coordinator



Participants need to block until the coordinator recovers or a back-up coordinator takes over

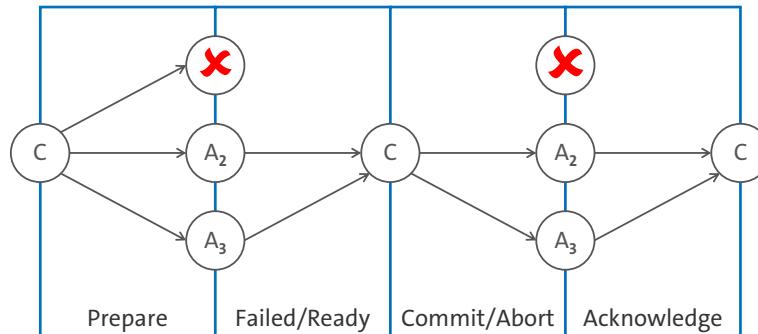
Crash after a participant sent FAILED

- No problem
- Participants know that coordinator must decide for global ABORT

Crash after participants sent READY

- Participants wait for global decision of crashed coordinator
- Participants are blocked → **Main problem of 2PC-Protocol!**
- Availability of participants for other global and local transaction gets restricted
- Solution: 3PC → avoids blocking (but more messages)

Failure Scenario: Crash of Participant



Participant involved in new transactions

- All new transactions that involve a crashed participant can not be executed (for now, we assume data is not replicated)

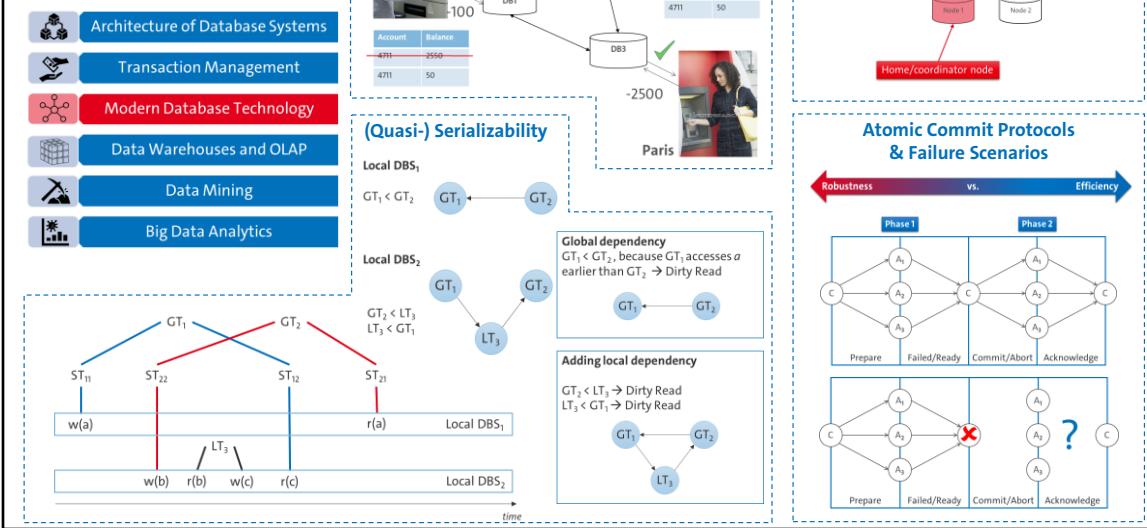
Participant does not answer to PREPARE message in time

- Coordinator has sent PREPARE and waits for answers of participants
- After timeout participant is considered as crashed → **Long blocking** → **Main problem of 2PC-Protocol!**
- Coordinator decides for ABORT -> sends an abort to all participants (as well as to the failed node once it is recovered)

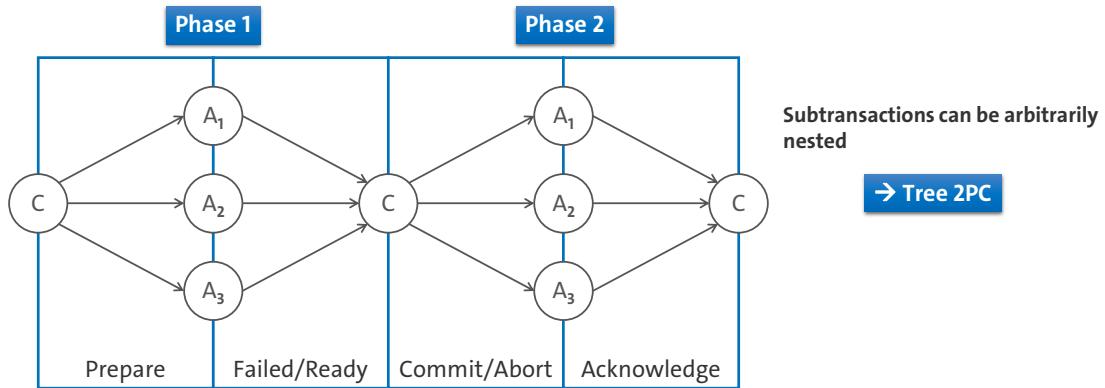
Crashed participant restarts

- Check log file for transaction T
- No READY entry → participant performs a local rollback and sends FAILED to coordinator
- READY entry but no COMMIT entry
 - Participant asks coordinator for global decision
 - Depending on coordinator's answer participant performs UNDO (Abort) or REDO (Commit)
- COMMIT entry → participant performs a local REDO of T

Summary

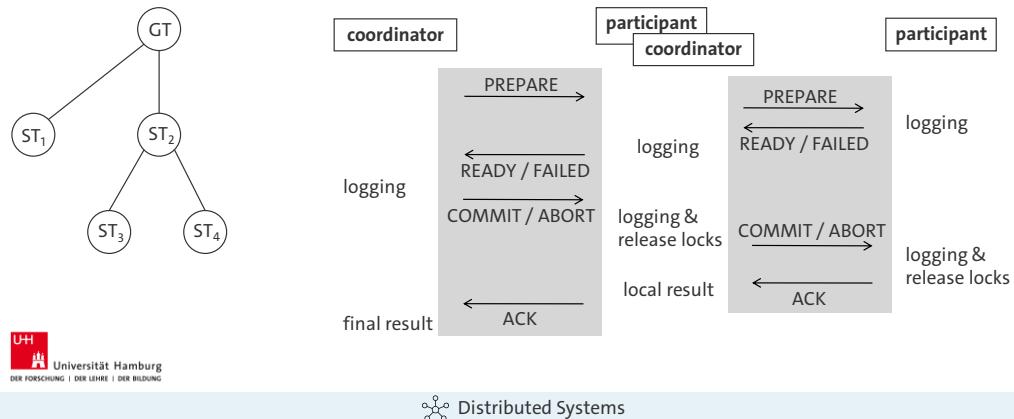


Recap Two-Phase-Commit Protocol



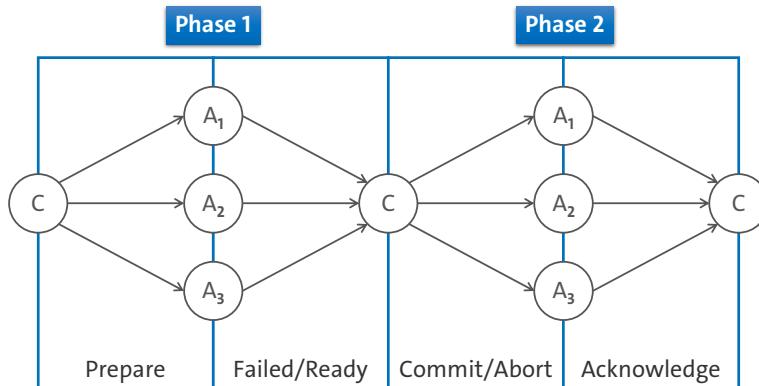
Tree 2PC

General model with arbitrary nesting → participants play the role of a coordinator for hierarchy of sub-transactions



- Answering time increases with nesting depth (lower parallelization)
- Flatten tree to apply “normal” 2PC

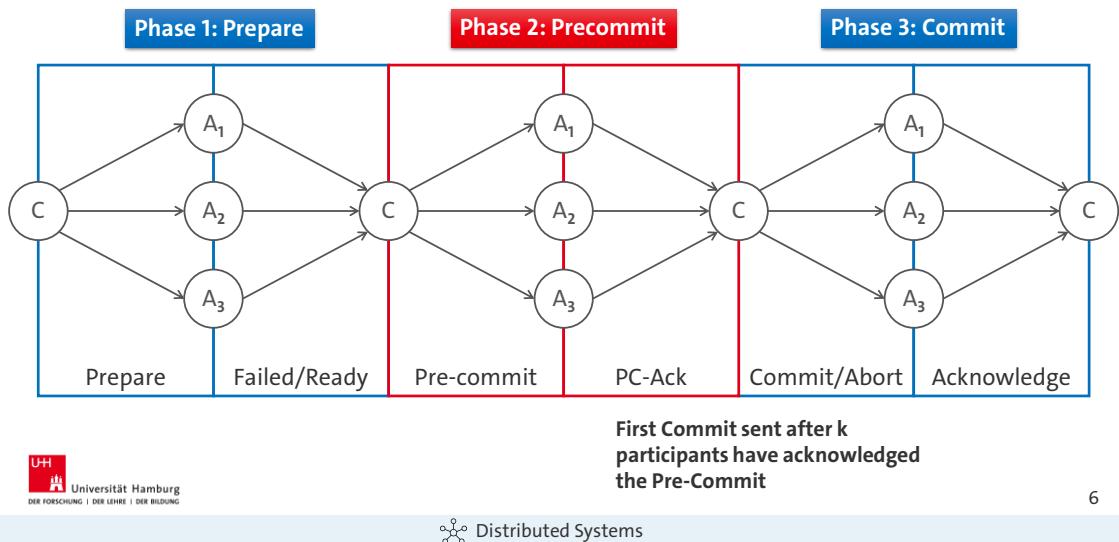
Recap Two-Phase-Commit Protocol



Failures are tolerated BUT
→ During restart of coordinator/participant, state of unfinished transactions must be reconstructed from logs
→ Long blocking → All participants are blocked until recovery is done

→ Solution: 3PC

Three-Phase-Commit Protocol



6

In case of a coordinator crash

- Coordinator crash is detected by timeout
- Selection of new coordinator (usually one of the participants)
- New coordinator asks remaining participants about the decisions logged for transactions affected by crash
- If one participant has COMMIT/ABORT in its log, that is the global decision
- If one participant has PRECOMMIT in its log, coordinator continues with sending out PRECOMMITS
- In any other case, the coordinator aborts the transaction

Advantage

- Non-blocking protocol, allowing $k < n$ coincidental crashes of participants
- Participant, who received a pre-commit, knows that global decision will be Commit
- If the pre-commit is not received (because the coordinator failed or the coordinator is still waiting for an answer of a failed participant), the participant rolls back and releases its blocked resources → no requirement for repeatedly asking the coordinator for a decision

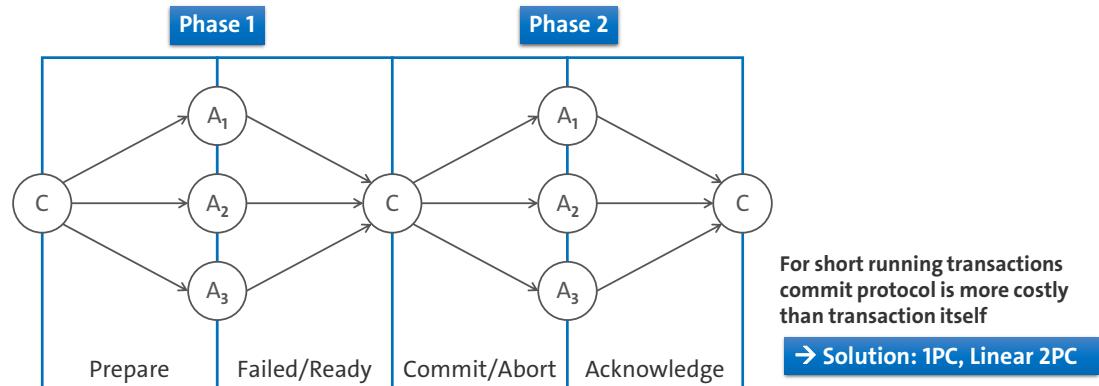
Problem 1: Complexity

- High number of messages
- n participants (including coordinator) require $6(n-1)$ messages in normal run
- $3n \log n$ messages
- Solution: 1PC (among others)

Problem 2: Fails in case of network partitioning

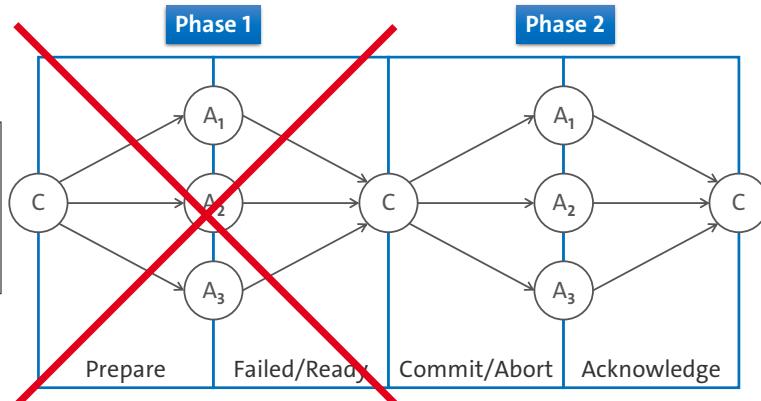
- One partition selects new coordinator while the other still has the old coordinator
- New coordinator could come to other global decision than old coordinator
- Solution: Enhanced Three Phase Commit (E3PC) → Quorum based approach for recovery

Recap Two-Phase-Commit Protocol



One-Phase-Commit Protocol

- Actual work starts here, before phase 1 (after calling a function $f()$)
- READY message is sent when $f()$ has returned, and the participant is aware of the prepare request



The Ready message is not sent after the prepare request, but directly after returning from function $f()$

→ After last operation, participant switches to prepared state

Advantages

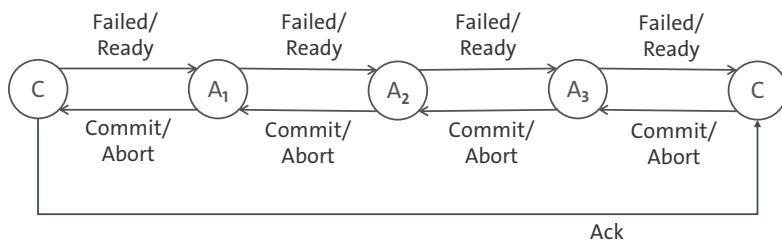
- Only 2(n-1) messages

Disadvantages

- Increased probability of blocking because of early prepared state
- High dependency from coordinator (no early unilateral abort) → Rarely used for distributed systems

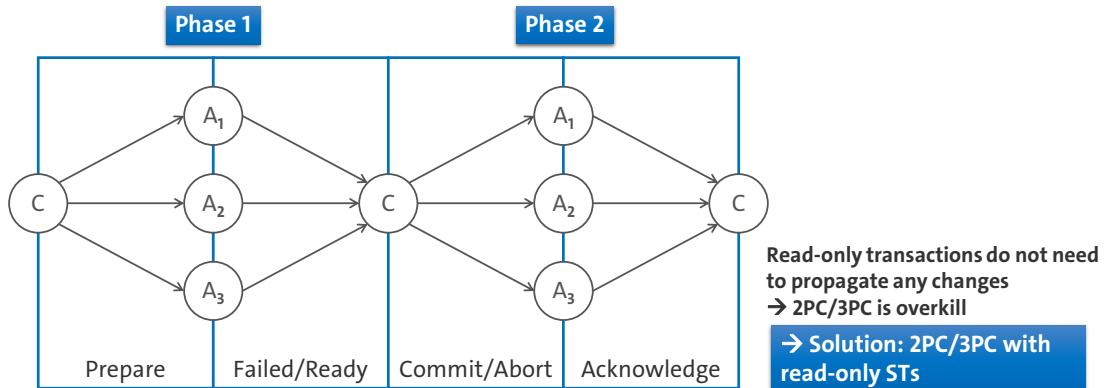
Linear 2PC

- Daisy chaining the communication between the participants
- Phase 1: Forward communication (READY / FAILED)
- Phase 2: Backward communication (COMMIT / ABORT)



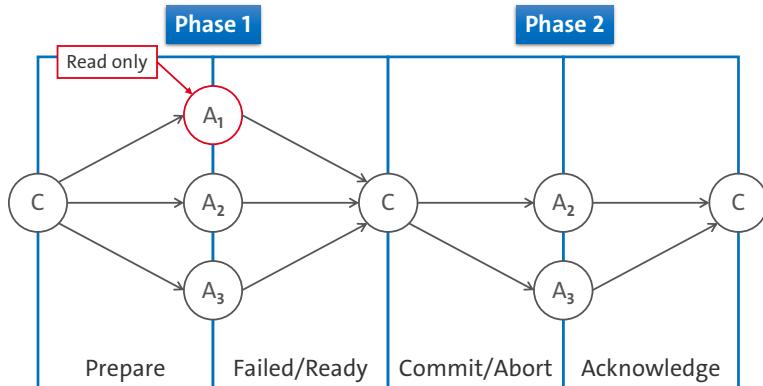
- Coordinator acts only as initiator
 - Goes to PREPARED state and sends local decision (READY/FAILED) to participant next in chain
- Each participant in the chain
 - Sends READY to next if it received READY and it locally decides for READY, too
 - Sends FAILED to next if it received FAILED or it locally decides for FAILED
- Last participant takes coordinator role and log global decision
- Global decision is communicated in reverse order back to the initiator
- Initiator sends ACKNOWLEDGE to last participant
- **Advantage:** Only 2n-1 messages
- **Disadvantage:** Significantly longer response time for large number of participants

Recap Two-Phase-Commit Protocol



Read-Only (Sub-)Transactions with 2PC

No 2nd phase if transaction is only reading



- No logging, no recovery no changes that would have to be undone/redone)
- Commit only releases locks (at the latest, locks might be released earlier, i.e. in phase 1, because they must be released irrespective of the success of the global transaction)

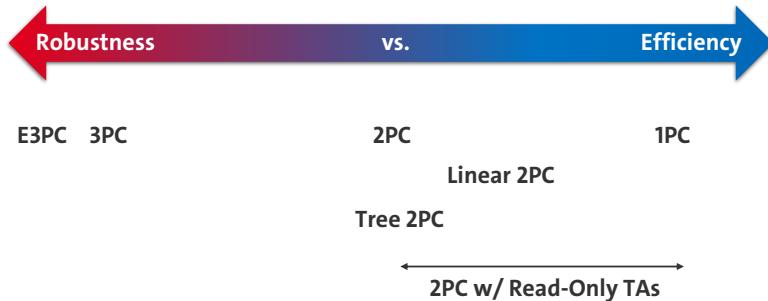
→ Protocol can be optimized

- Assuming m readers among n-1 sub-transactions ($m < n$)
- Reduced message complexity: $4(n - 1) - 2m$ messages
- Reduced log complexity: $2n-m$ log writes
- Special case: read-only global transaction ($m = n$)
 - only $2(n-1)$ message, no log writes (like 1PC)

Atomic Commit Protocols

- Non-blocking
- Each node should be allowed to abort as long possible

Generate as few messages and log entries as possible

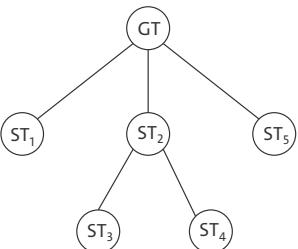


!Here, efficiency is not necessarily equal to the performance of transactions, but only refers to the number of generated messages and log entries!

→ Performance also depends on multiple other factors, e.g. nesting, number of STs, ratio of reading STs to updating STs.

Exercise

Distributed transaction running
on 6 different nodes

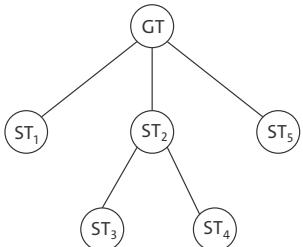


- How many messages are required to perform the 2-Phase-Commit Protocol (with a flattened process tree) for this transaction in case of a successful processing?
- How would that change, if T_5 fails?
- How does it change if 1PC, Tree 2PC, or 3PC is used?
- How many messages are necessary with 2PC when ST_1 and ST_5 are read-only?

Show the communication topology for each variant!

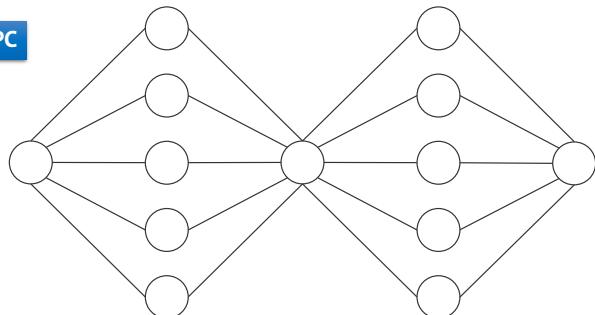
Exercise

Distributed transaction running on 6 different nodes



- How many messages are required to perform the 2-Phase-Commit Protocol (with a flattened process tree) for this transaction in case of a successful processing?
- How would that change, if T_5 fails?
- How does it change if 1PC, Tree 2PC, or 3PC is used?
- How many messages are necessary with 2PC when ST_1 and ST_5 are read-only?

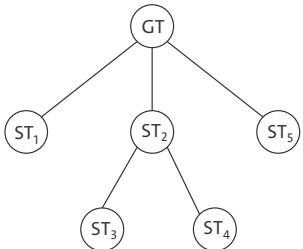
2PC



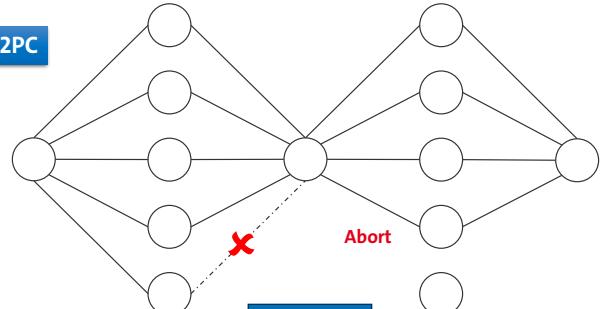
$$4 * (n-1) = 4 * (6-1) = 20$$

Exercise

Distributed transaction running on 6 different nodes

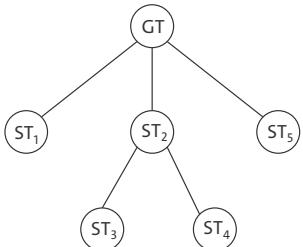


- How many messages are required to perform the 2-Phase-Commit Protocol (with a flattened process tree) for this transaction in case of a successful processing?
- **How would that change, if T_5 fails before sending the Ready message?**
- How does it change if 1PC, Tree 2PC, or 3PC is used?
- How many messages are necessary with 2PC when ST_1 and ST_5 are read-only?



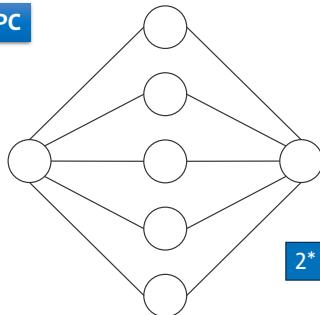
Exercise

Distributed transaction running on 6 different nodes



- How many messages are required to perform the 2-Phase-Commit Protocol (with a flattened process tree) for this transaction in case of a successful processing?
- How would that change, if T_5 fails before sending the Ready message?
- How does it change if 1PC, Tree 2PC, or 3PC is used?
- How many messages are necessary with 2PC when ST_1 and ST_5 are read-only?

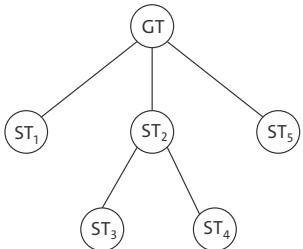
1PC



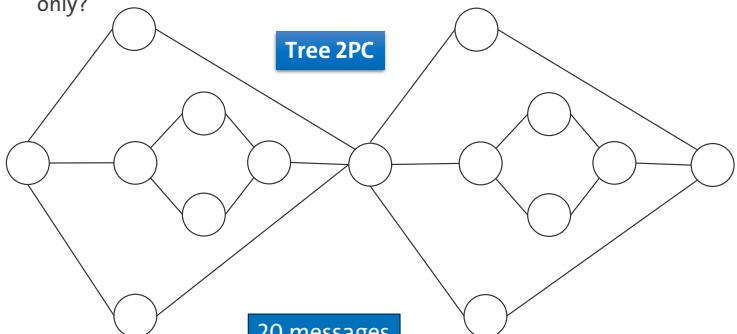
$$2 * (n-1) = 2 * (6-1) = 10$$

Exercise

Distributed transaction running on 6 different nodes

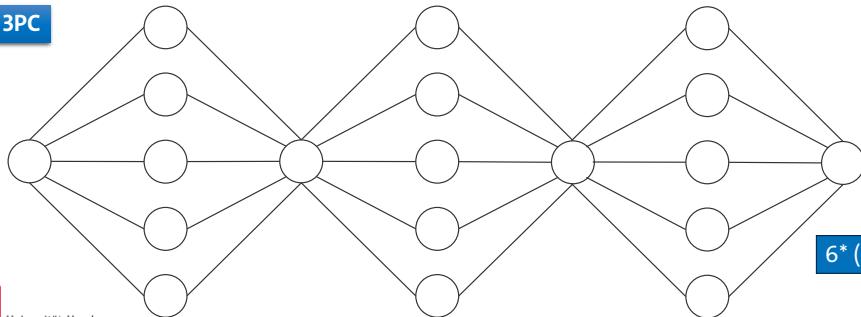


- How many messages are required to perform the 2-Phase-Commit Protocol (with a flattened process tree) for this transaction in case of a successful processing?
- How would that change, if T_5 fails before sending the Ready message?
- How does it change if 1PC, Tree 2PC, or 3PC is used?
- How many messages are necessary with 2PC when ST_1 and ST_5 are read-only?



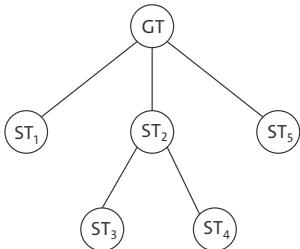
Exercise

- How many messages are required to perform the 2-Phase-Commit Protocol (with a flattened process tree) for this transaction in case of a successful processing?
- How would that change, if T_5 fails before sending the Ready message?
- **How does it change if 1PC, Tree 2PC, or 3PC is used?**
- How many messages are necessary with 2PC when ST_1 and ST_5 are read-only?



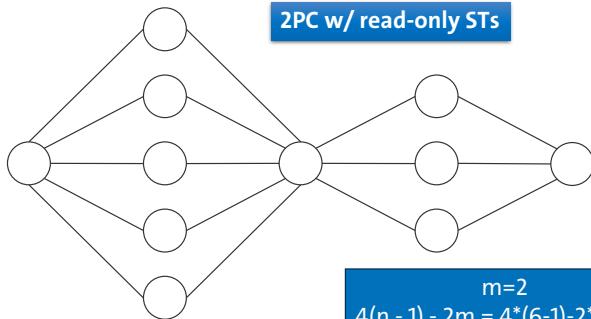
Exercise

Distributed transaction running on 6 different nodes



- How many messages are required to perform the 2-Phase-Commit Protocol (with a flattened process tree) for this transaction in case of a successful processing?
- How would that change, if T_5 fails?
- How does it change if 1PC, Tree 2PC, or 3PC is used?
- How many messages are necessary with 2PC when ST_1 and ST_5 are read-only?

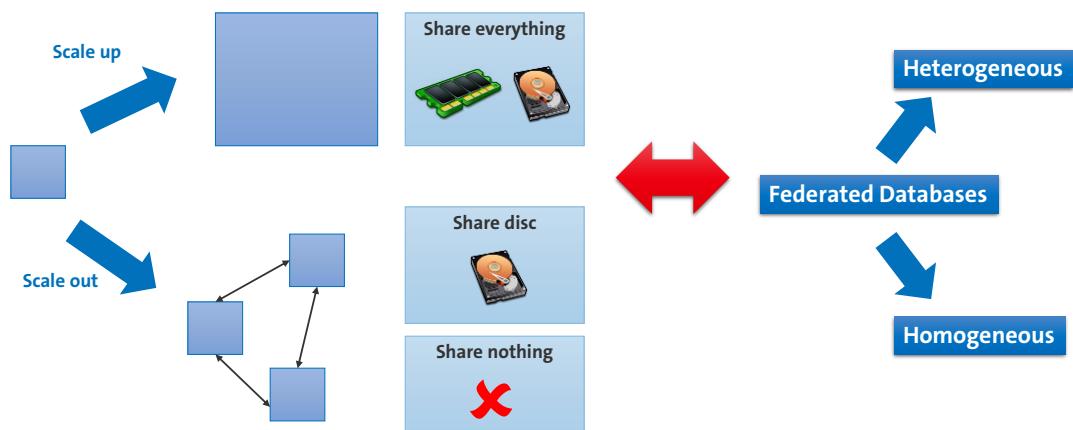
2PC w/ read-only STs



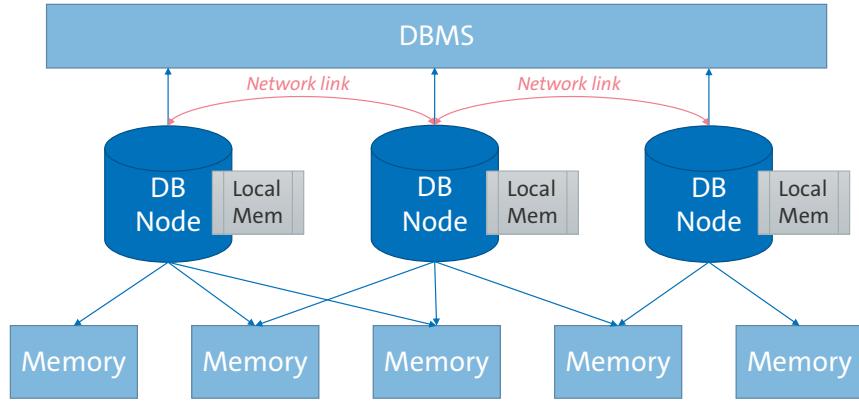
$m=2$

$$4(n - 1) - 2m = 4*(6-1)-2*2 = 16$$

To share or not to share: Types of distributed DBs



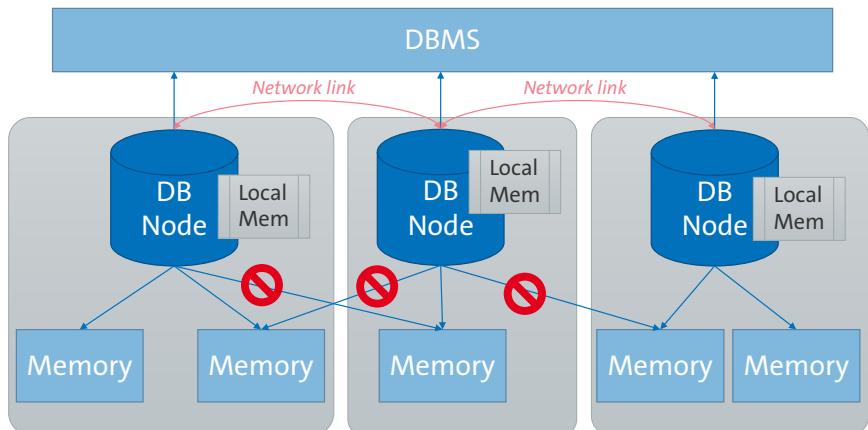
Shared Disc



Shared (Disk) Cluster

- Distributed Database
- One or more servers have equal access to memory (e.g. to discs)
- Servers don't share their own memory

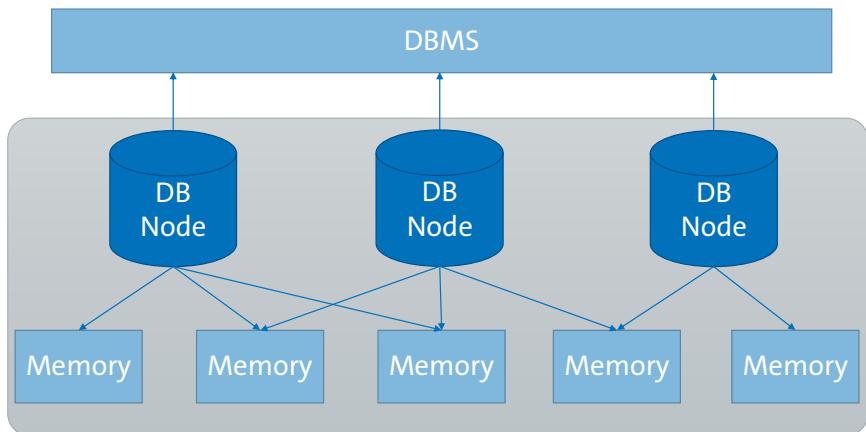
Shared Nothing



Shared Nothing

- Distributed Database
- Each node has exclusive access to its memory
- Servers don't share their own memory

Share Everything



Shared Everything

- One big system instead of multiple (small) systems → scale-up instead of scale-out
- Disk and main memory is shared (NUMA system)

To Share Or Not To Share

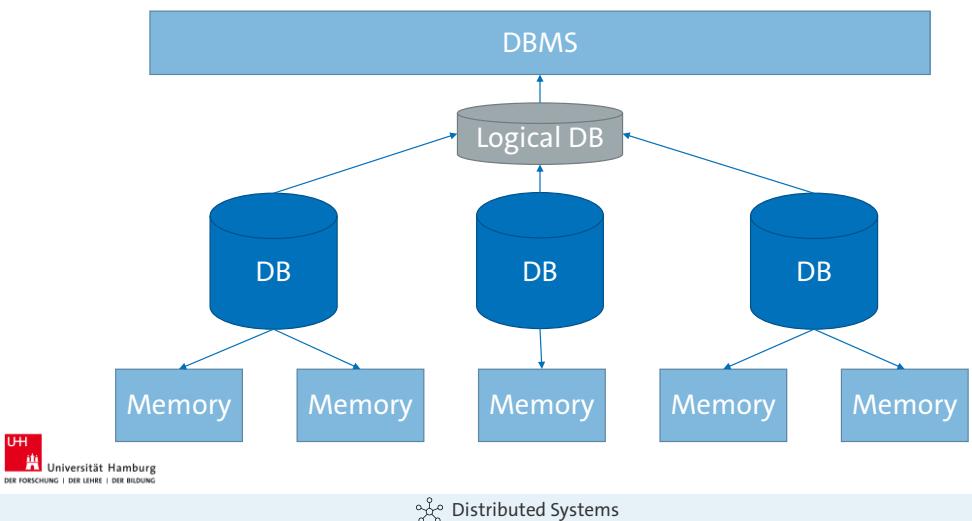
	Scale-out		Scale-up
	Shared Disk	Shared Nothing	Share Everything
Advantages	<ul style="list-style-type: none"> - Robust in case of node failure (discs can be accessed by another node) - Usually easy to set up (if there is already a shared file system) 	<ul style="list-style-type: none"> - Robust in case of disk failure (frequently used data can be replicated across nodes) - No distributed locking necessary - High performance if query is executed on node where most of the data is 	<ul style="list-style-type: none"> - Comes for free with many systems → no additional setup - Faster than accessing remote memory
Disadvantages	<ul style="list-style-type: none"> - Simultaneous disk access is a potential bottleneck - Overhead to maintain cache consistency - Requires complex locking mechanisms for updates 	<ul style="list-style-type: none"> - Partitioning (sharding) of data needs additional care to get optimal performance (store data where it is processed) 	<ul style="list-style-type: none"> - Limited scaling possibilities - Hardware for large systems becomes expensive
Example System(s)	<ul style="list-style-type: none"> - Add-on/Feature of data management systems, e.g. Microsoft Azure shared disk, Oracle RAC - Hybrid Systems (SD & SN), e.g. Snowflake 	Couchbase*, MariaDB SkySQL (not actively developed anymore),...	Each system working with NUMA architectures, e.g. MonetDB, PostgreSQL, SQL Server,...
Comments	Requires shared file system		Usually not noticed by the user

 Distributed Systems

*Couchbase white paper:

http://info.couchbase.com/rs/northscale/images/Couchbase_Architectural_Document_Whitepaper_2015.pdf

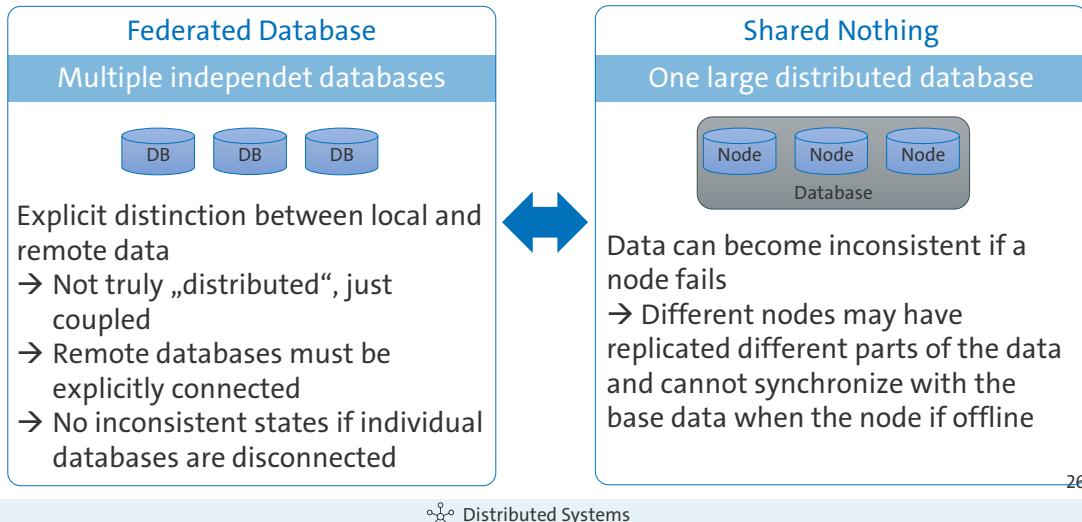
Federated Databases



Federated Database

- Multiple Databases
- Databases are connected to one logical view
- Databases do not directly share data
- Data is not replicated

Federated Databases



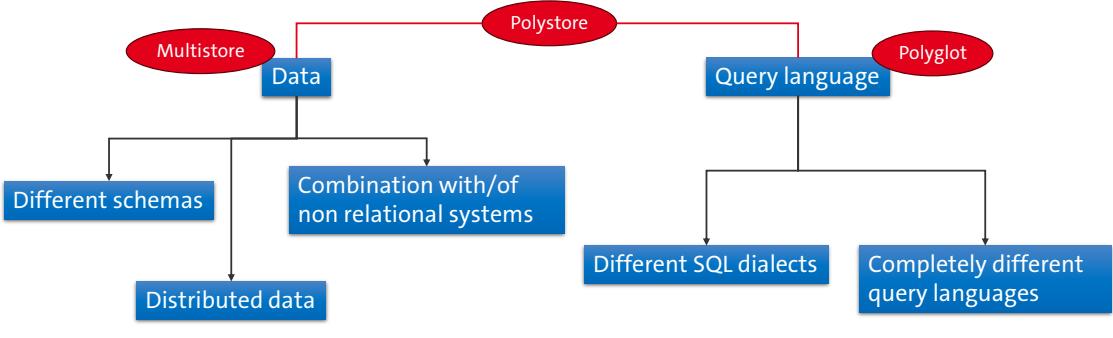
26

 Distributed Systems

- In a shared nothing DB, a node is just a node in the DB system
- In a federated DB, each “node”/component is an autonomous DB system → Data is not copied between these DBs
- A DB as a component can itself be a distributed DB

Heterogeneous Federated DBs

Components can vary in different aspects → This is one of various possible taxonomies



Heterogeneous DBs spawn a variety of research topics, e.g. schema integration and novel interfaces

Further reading

An approach to classify Federated DBMS:

Azevedo, Leonardo Guerreiro, et al. "Modern Federated Database Systems: An Overview." *ICEIS* (1) (2020): 276-283.

Different approach for classification based on Data coupling

Loosely coupled → Local stores accessed by their local language or a common language

Tightly coupled → Local stores accessed by a multistore system using the same language for structured and unstructured data (e.g. Hadoop)

Hybrid → Some Stores are loosely coupled and some are tightly coupled (e.g. Spark SQL, BigDawg)

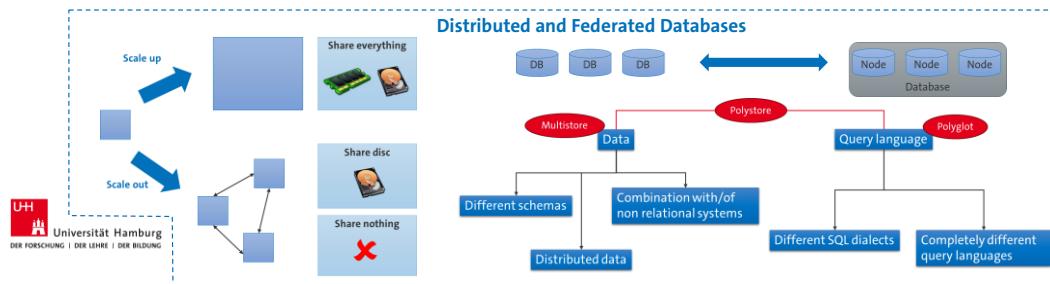
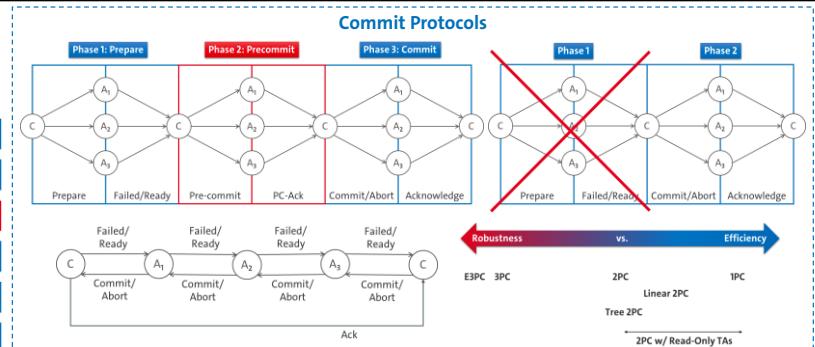
Important information for those who learn with the material of the previous year

→ In the slides Prof. Ritter, yet another taxonomy is used: Homogeneous

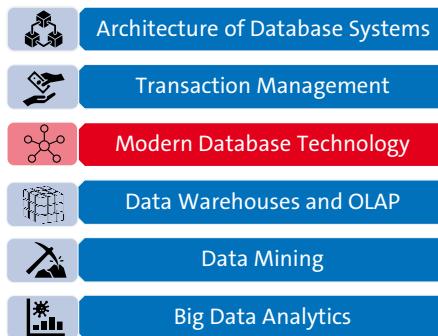
Federations can run only global transactions whereas heterogeneous federations can also run local transactions

→ This describes heterogeneity in terms of autonomy

Summary



Course Outline



- Distributed Systems
- Optimizations in RDBMS
- NoSQL

Storage Layouts

2 main layouts to store your table

Row-Store (tuple-wise)										This is what your traditional relational SQL database does		
MensaMeals	Meal	Price	Pizza	6,50	Pasta	4,90	Pie	1,20	Potato Salad	5,80	Pann-fisch	7,90
	Pizza	6,50	Memory address →									
	Pasta	4,90	Column-Store (attribute-wise)									
	Pie	1,20	Pizza	Pasta	Pie	Potato Salad	Pann-fisch					
	Potato Salad	5,80	6,50	4,90	1,20	5,80	7,90					
	Pannfisch	7,90	Memory address →									



4

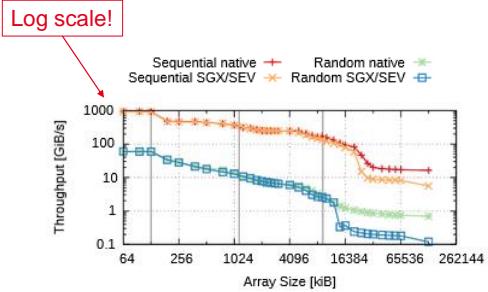
Optimizations in RDBMS

Relations are usually illustrated as tables

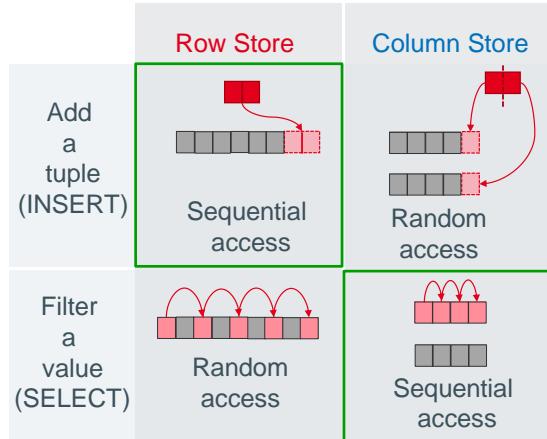
- This tells us nothing about the storage layout
(cf. a matrix that can be stored differently → row- or column-major)

Why should you care?

Memory access is expensive!



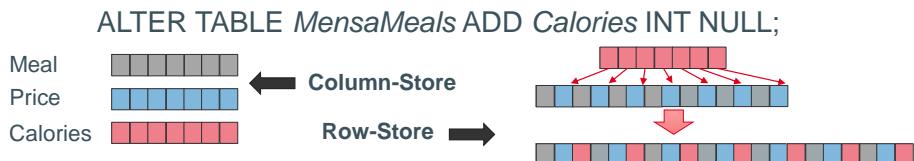
*Throughput on an Intel Xeon E3-1275
Gottel, Christian & Pires, Rafael & Rocha, Isabela & Vaucher, Sébastien & Felber, Pascal & Pasin, Marcelo & Schiavoni, Valerio. (2018). SRDS 2018



Your ideal layout depends on your use-case.
Different systems use different layouts, so choose wisely!

NoSQL and column-oriented DBs: Frequent misunderstandings

Wide-column DBs (NoSQL) = column-stores (SQL)



Remember what we just learned about
random memory access

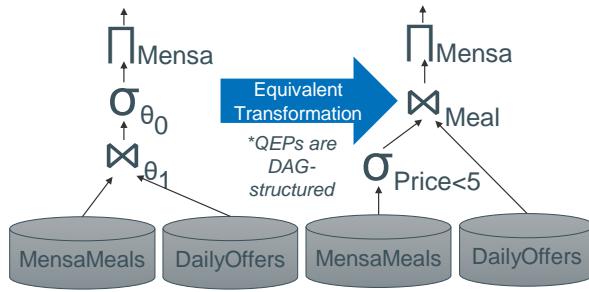
- NoSQL stands for **Not only SQL**
- Wide-column DBs (NoSQL) and column-stores (SQL) are not the same, but both often referenced as column-oriented
 - We will use it to reference column-stores
- Usually, column-oriented databases can be queried using SQL and allow the definition of relations
 - Convenience of SQL, and performance and flexibility of column-stores
 - Example: Fast and easy addition/deletion of attributes

Query Execution Plan Optimization

```
SELECT Mensa FROM MensaMeals, DailyOffers
WHERE MensaMeals.Meal = DailyOffers.Meal
AND MensaMeals.Price < 5;
```

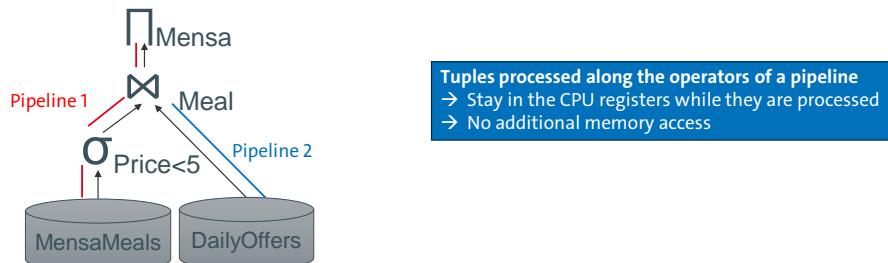
Plan A: $\prod_{\text{Mensa}} (\sigma_{\text{Price} < 5} (\text{MensaMeals} \bowtie_{\text{Meal}} \text{DailyOffers}))$

Plan B: $\prod_{\text{Mensa}} ((\sigma_{\text{Price} < 5} (\text{MensaMeals})) \bowtie_{\text{Meal}} \text{DailyOffers})$



- Database Systems use a relational algebra for internal representation
- Optimizers try to automatically find the most efficient sequence of operators
 - Conventional approach: Reduce data as early and as cheap as possible
 - Tool: Cardinality/Selectivity estimation
- The chosen sequence of operators is the final Query Execution Plan (QEP)
- See 1st lecture of the semester for further reading

Operator Fusion



Step 1: Optimizer identifies pipeline breakers, i.e. operators that must materialize (intermediate) results

Step 2: Query is compiled such that there is one loop for each pipeline

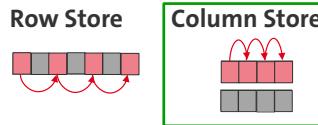
Step 3: Values/individual tuples are loaded into CPU registers

Step 4: Tuples are processed along their pipeline (pushed to the parent operators) without being materialized until the next pipeline breaker

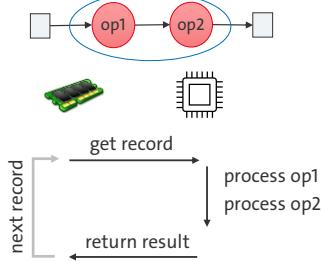
Further Reading

Menon, Prashanth, Todd C. Mowry, and Andrew Pavlo. "Relaxed operator fusion for in-memory databases: Making compilation, vectorization, and prefetching work together at last." *Proceedings of the VLDB Endowment* 11.1 (2017): 1-13.

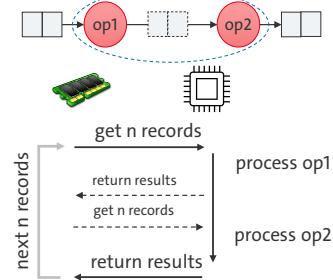
Data Processing Models



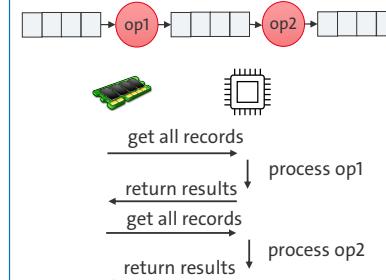
Tuple-at-a-time



Vector/Block-at-a-time



Operator-at-a-time



For parallel or pipelined execution, data must be split

Tuple-at-a-time

- Intermediate tuples not stored, but passed directly to next operator
→ Operators can be fused
- Limited applicability of other optimizations, e.g. prefetching, vectorization, compression,...

Vector/Block-at-a-time

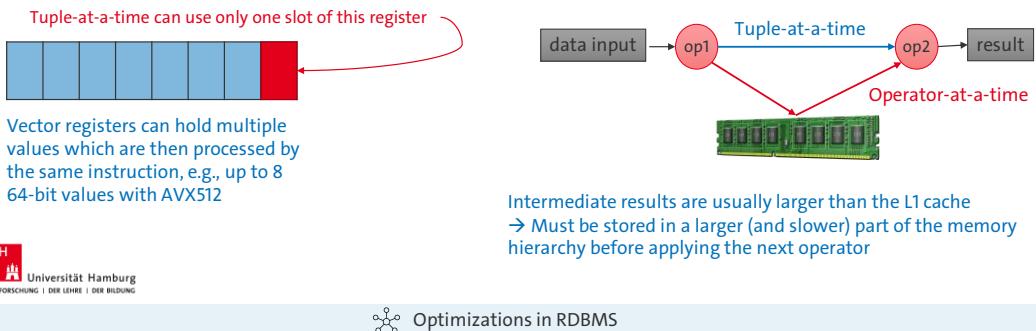
- A part (vector/block) of the column processed at once
→ Operator fusion only for small blocks
- Trade-off between operator fusion and memory access performance

Operator-at-a-time

- Whole operator (all elements of the column) processed at once
- Intermediates materialized
→ No operator fusion, only coarse-grained parallelization
- High potential for optimization of memory reads

Why should I care?

- Different optimizations work with different processing models
- Your hardware limits your optimization space



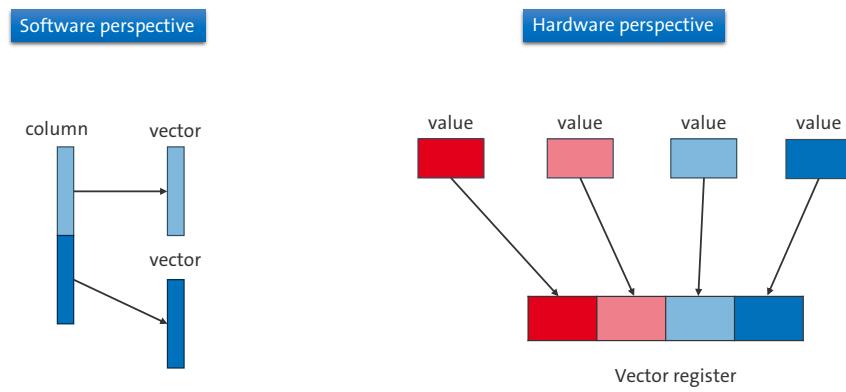
Example A: You have a recent intel server with the AVX512 instruction set for vectorization (under linux, `lscpu` tells you if you have it; no root required)

- A system which implements only tuple-at-a-time is not able to use this instruction set

Example B: You do not have much main memory and writing to it is slow

- Materializing your intermediates becomes a bottleneck and might not work at all with operator-at-a-time or large blocks (block-at-a-time)

Vectorized Processing vs. Vectorized Processing



Software perspective

Data is divided into vectors or blocks and processed vector-wise → Vector-at-a-time

Hardware perspective

A single register (“vector register”) can hold multiple values which are then processed at the same time → Single Instruction Multiple Data (SIMD)

- A vector in vector-at-a-time processing can be as small as a vector register, but can also be significantly larger
- Both approaches can be used at the same time

Exercise Optimizations

- Assume a table $t(a1 SMALLINT, a2 FLOAT)$ that has $20 \cdot 10^9$ records
- The table resides in main memory
- Assume further that we process all queries on a system with two 64-bit memory channels and a maximum memory frequency of 2GHz
- How much time does it take **at least** to copy all values for the following query from main memory to L1 cache if a column store + vector-at-a-time is used?:

SELECT a1 FROM t WHERE a1 < 5;

- How much time does it take for a row store + operator-at-a-time?

smallint → 16 bit
int → 64 bit (on most systems)

$$100\text{bn smallint values} = 100 \cdot 10^9 \cdot 2 / 1024^3 = 186 \text{ GB}$$

Ignored here (among others):

- NOPs
- Computation of the operators
- Computation of addresses
- Materialization of the (intermediate) results

Exercise Optimizations

- Assume a table $t(a1 SMALLINT, a2 INT)$ that has $20 \cdot 10^9$ records
- The table resides in main memory
- Assume further that we process all queries on a system with two 64-bit memory channels and a maximum memory frequency of 2GHz
- How much time does it take **at least** to copy all values for the following query from main memory to L1 cache if a row store + operator-at-a-time + operator fusion is used?:

SELECT a1+a2 FROM t WHERE a1 < 5;

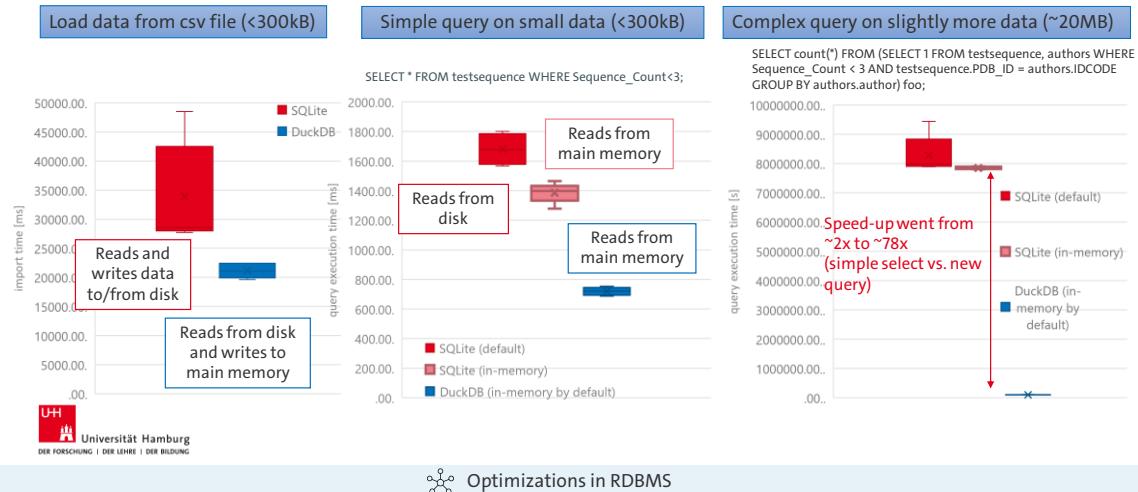
- How much time does it take without operator fusion?
- How much time does it take for a column store + vector-at-a-time (no operator fusion)?

smallint \rightarrow 16 bit
int \rightarrow 64 bit (on most systems)

$$100\text{bn smallint values} = 100 \cdot 10^9 \cdot 2 / 1024^3 = 186 \text{ GB}$$

The Effect of Optimizers and Engines

SQLite  vs. DuckDB 



Data was taken from the Protein Data Bank (www.pdb.org, accessed in 2022)

SQLite

- Row-Store
- Disc-centric
- Tuple-at-a-time processing
- Only 1 Join implementation
- Next to no query optimization
- „Ancient“ and extremely popular

DuckDB

- Column-Store
- In-Memory with out-of-memory option
- Vector-at-a-time processing
- Different Join implementations
- Optimizer actually does stuff (join order optimization, eliminate common subqueries,...)
- Relatively new and less popular than SQLite

Optimizations By The User: (Materialized) Views

```
Store (materialize) the view          Create a view called  
CREATE MATERIALIZED VIEW CheapFood AS SELECT Meal FROM MensaMeals WHERE Price < 5;
```

```
Refresh the view  
REFRESH MATERIALIZED VIEW CheapFood;
```



- Refresh the view after updates in your base data
- Materializaton not supported by all database systems

18

Optimizations in RDBMS

We already talked about indexes in the first lecture this semester

Reusability of queries and query results

- Queries and Subqueries (Views) can be stored and referenced → Nicer queries
- The result of views can be stored → Higher performance for frequently used queries and remote data

Exercise Optimizations by the user

Query 1

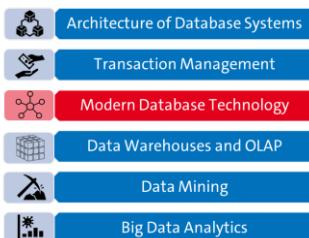
```
SELECT count(*) FROM testsequence, authors WHERE testsequence.Sequence_Count < 3 AND testsequence.PDB_ID = authors.IDCODE;
```

Query 2

```
SELECT authors.IDCODE, authors.name FROM institute, testsequence, authors WHERE testsequence.Sequence_Count < 3 AND testsequence.PDB_ID = authors.IDCODE AND institute.name = authors.institute;
```

Which optimizations can a user apply to increase the query performance?

Summary



Storage Layout

	Row Store	Column Store
Add a tuple (INSERT)	Sequential access	Random access
Filter a value (SELECT)	Random access	Sequential access

Query Execution Plans

```

    graph TD
        Mensa[Mensa] --> Pipeline1[Pipeline 1]
        MensaMeals[MensaMeals] --> Pipeline1
        Pipeline1 --> Join((σ))
        DailyOffers[DailyOffers] --> Pipeline2[Pipeline 2]
        Pipeline2 --> Join
        Join --> Meal[Meal]
    
```

Data Processing Models

	Tuple-at-a-time	Vector/Block-at-a-time	Operator-at-a-time
Diagram			

Effect of Engine Optimizations

Simple query on small data (<300kB)

```
SELECT * FROM testsequence WHERE Sequence_Count < 3;
```

Complex query on slightly more data (~20MB)

```
SELECT count(*) FROM (SELECT 1 FROM testsequence, authors WHERE Sequence_Count < 3 AND testsequence.POB_ID = authors.IDCODE GROUP BY authors.author) f;
```

Engine	Query Type	Time (ms)	Time (s)
SQLite (default)	Simple query	~1.8	~0.0018
SQLite (in-memory)	Simple query	~1.4	~0.0014
DuckDB (in-memory by default)	Simple query	~0.1	~0.0001
SQLite (default)	Complex query	~8.5	~0.0085
SQLite (in-memory)	Complex query	~8.0	~0.0080
DuckDB (in-memory by default)	Complex query	~0.1	~0.0001

SQLITE vs DuckDB

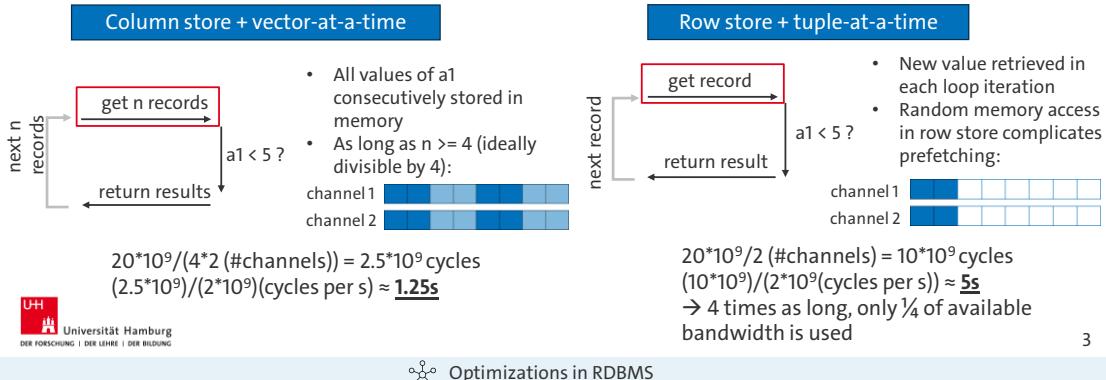
~78x

Recap Data Access Time

SELECT a1 FROM t WHERE a1 < 5;

a1 ∈ SMALLINT
 $20 \cdot 10^9$ records
 2 memory channels @ 2GHz & 64 bit

Minimum time required for data transfer from memory to CPU for the following scenarios:



Thinking about what happens on the hardware side is always a useful tool for optimizing performance

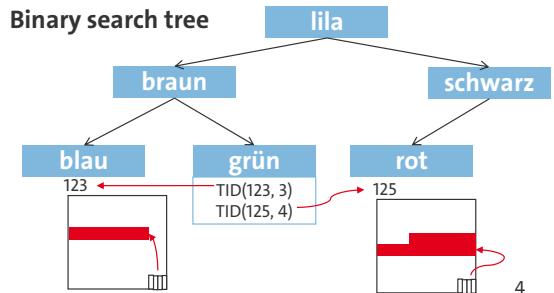
More interesting aspects of this example:

- The example assumes that the data is stored in a compact format in the column store, i.e. only the space needed for the data format is used, not the size of a whole processor word. If this is not the case, the required time increases → This is the absolute „minimum“ if the memory is used optimally
- The time required for the comparison itself can most likely be masked because it is done faster than the retrieval of the new record(s) which happens at the same time
- Writing data is slower than reading it → There is a break-even point when operator-fusion (if there was more than 1 operator) becomes more beneficial than vector-at-a-time processing, but it is different depending on the data type (i.e. its size), the (physical) operators, and the hardware

Optimizations by the User: Indexes

- Primary or Secondary Index (additional access path)
- Simple index or multilevel index
- Storage structure:
 - Tree-structured, e.g. B-Tree
 - Sequential, e.g. Sequential lists
 - Scattered, e.g. Hash regions
- Access method:
 - Key comparison
 - Key transformation, e.g. hashing

Sequential List	
blau	
braun	
grün	TID(123, 3) TID(125, 4)
lila	
rot	
schwarz	



Recap of indexes in 1st DIS-lecture

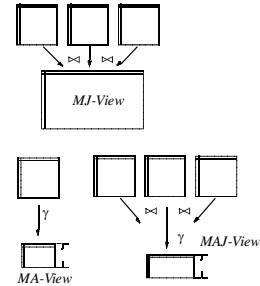
Optimizations by the User: (Materialized) Views

Store (materialize) the view
CREATE MATERIALIZED VIEW CheapFood AS SELECT Meal FROM MensaMeals WHERE Price < 5;

Create a view called CheapFood

Refresh the view
REFRESH MATERIALIZED VIEW CheapFood;

- ! Refresh the view after updates in your base data
- Materialization not supported by all database systems



5

Optimizations in RDBMS

Reusability of queries and query results

- Queries and Subqueries (Views) can be stored and referenced → Nicer queries
- The result of views can be stored → Higher performance for frequently used queries and remote data

Types of Materialized Views

- Materialized join view
- Materialized aggregate view
- Materialized aggregate-join view

Derivability of queries

- Query Q may be replaced by Q'
 - Q' uses the contents of mat. view V
 - Q' delivers a semantically equivalent results to Q
 - Q' is called compensation query
- Identification of derivability is NP-hard
- Predicate P_M und P_Q : $P_Q \subseteq P_M$

- General undecidability of predicate logic
- Reduction to elemental predicates after standardization in disjunctive normal form
- Projected attributes are not considered (but would be necessary for decidability)
- More about computability theory in general: “Theoretische Informatik - kurzgefasst” by Uwe Schöning, or in any textbook about the theory of informatics
- More about predicate logic: “Logik und Logikprogrammierung” by Steffen Hölldobler (chapter 4.8.4 is about the undecidability of predicate logic)

Exercise Optimizations by the User

Query 1

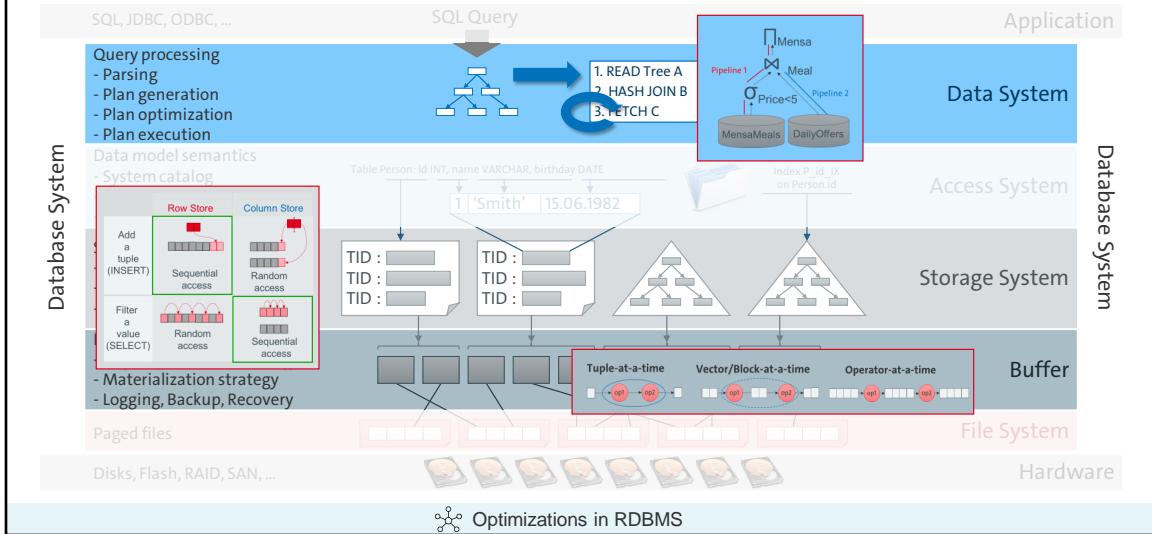
```
SELECT count(*) FROM testsequence, authors WHERE testsequence.Sequence_Count < 3 AND testsequence.PDB_ID = authors.IDCODE;
```

Query 2

```
SELECT authors.institute, authors.name FROM institute, testsequence, authors WHERE testsequence.Sequence_Count < 3 AND testsequence.PDB_ID = authors.IDCODE AND institute.name = authors.institute;
```

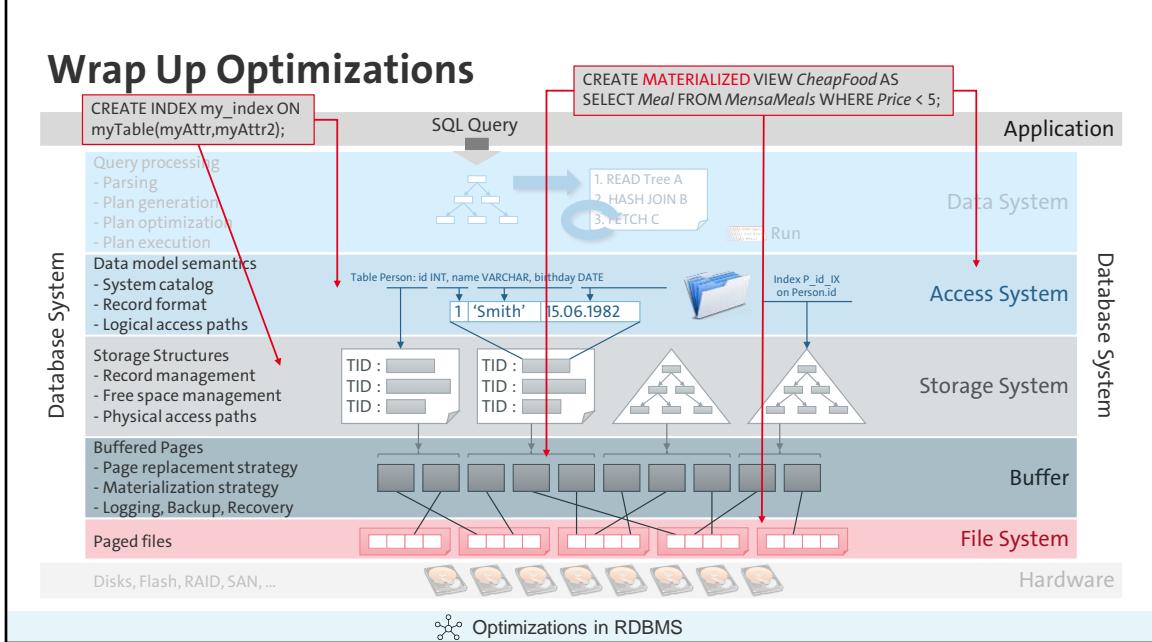
Which optimizations can a user apply to increase the query performance?

Wrap Up Optimizations



- There are countless more optimizations that can be done, e.g. different physical operators, intelligent prefetching,...
- The Data System takes into account the features of the Access System for its optimization, e.g. available indexes
- Column Stores usually provide a better performance for analytical queries than row stores

Wrap Up Optimizations



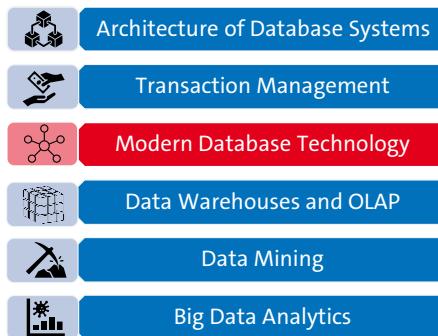
- The optimizations a user can apply (which are not changing/enabling/disabling a feature of the DBS as a whole) are usually triggered by SQL queries
- They change the behavior in the lower layers
 - A materialized view can change the files on disc, materializes a query result that might otherwise stay in the buffer, and adds an additional access path to data resulting from the query in question
 - Indexes add additional access paths (stored in the storage system and used by the access system)
- Some other optimizations, e.g. query rewriting, might be useful but their effect depends on the used DBS

Survey



<https://evasys-online.uni-hamburg.de/evasys/online.php?pswd=J3Q6H>

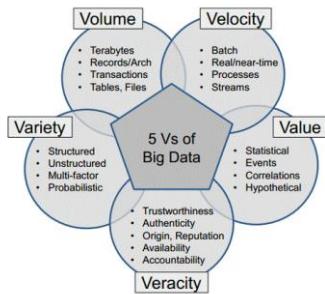
Course Outline



- Distributed Systems
- Optimizations in RDBMS
- NoSQL

Big Data Management

2000's – Today: Big Data



"data comes first, schema comes second" (or never)



Need to revisit Data Management

The number of Vs depends on who you ask, it started with just Variety, Volume, and Velocity

Challenges for Data Management

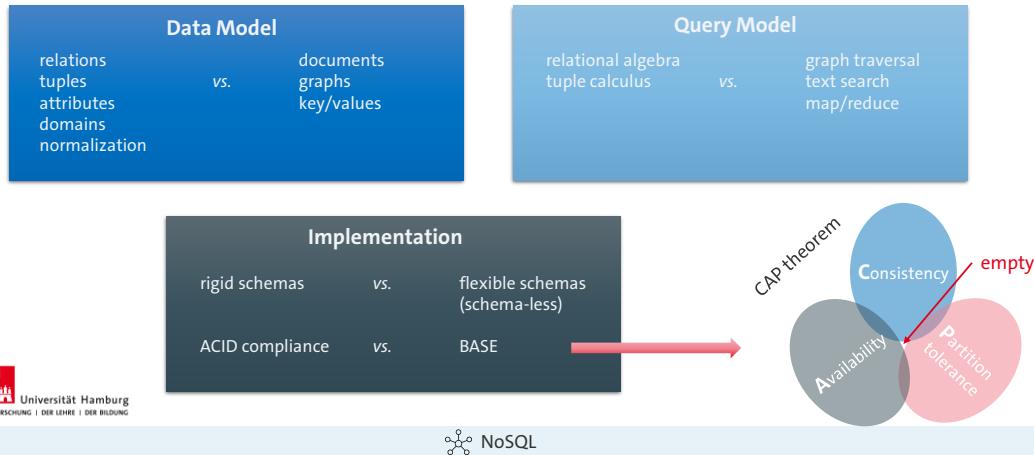
- Volume: From GB's to PB's ...
- Variety: Tables, Text, Images, ...
- Velocity: Sensors continuously generate data
- Value: More complex analytics (from SQL to deep learning)
- Veracity: Data quality, data bias, ...

Big data scenarios try to avoid a priori data modelling

- Use case unknown, data sources unknown, source schemas unknown/permanently changing, etc.
- NoSQL as movement to reflect an agile “working with existing data”, instead of “reflecting a real world by a schema/logical model”
- If a logical data model is created prior to a NoSQL implementation, the Logical Model and the Physical Model may differ – even substantially, especially due to the extreme de-normalization and flattening that occurs within NoSQL

Non-Relational Databases

Most NoSQL data systems diverge from standard relational systems



ACID → Atomicity, Consistency, Isolation, Durability

BASE: Availability over consistency

- Basically available: The database is available for changes, even if these changes are applied later, i.e. it is not necessarily in a consistent state when the user accesses it
- Soft state: Data can have a temporary state, e.g. when multiple applications change it at the same time
- Eventually consistent: Consistency is reached eventually, but only when (finally) all changes have been applied that were made at the same time

→ BASE scales easier than ACID

CAP theorem:

- In a distributed system, only 2 of the 3 properties Availability, Consistency, and partition tolerance can be reached at the same time
- ACID prioritizes Consistency (CA, CP)
- BASE prioritizes Availability (AP)

Types of Data Systems

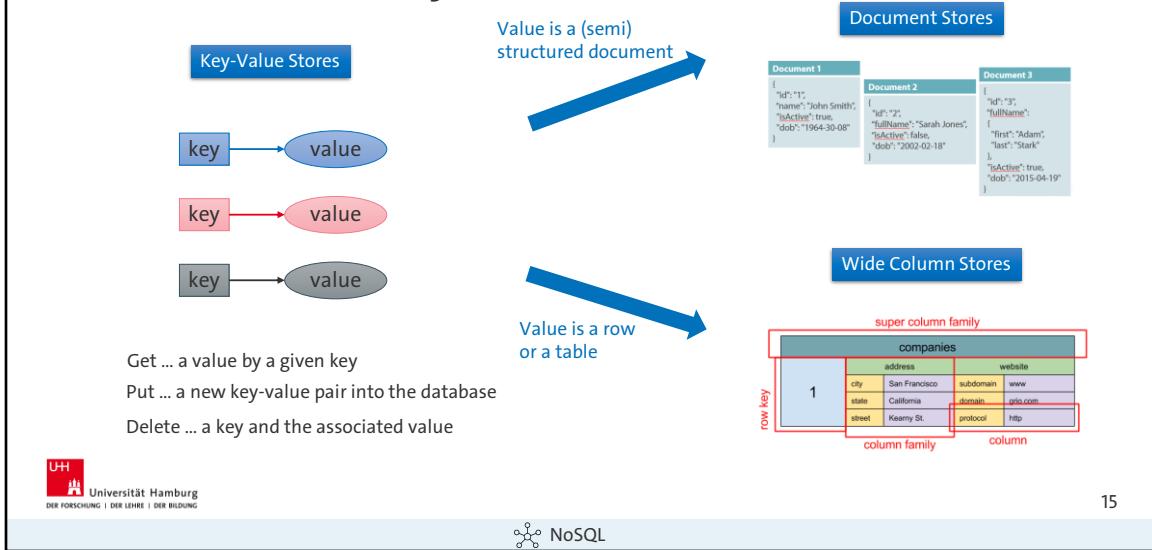
Relational Databases (SQL)

- Popular Systems
 - IBM DB2 PostgreSQL, MS SQL Server, SQLite, MySQL
- NewSQL Database
(Scale-out using distributed processing)
 - Google Spanner, VoltDB (H-Store), SAP HANA, MemSQL

Non-relational/Not only SQL Databases (NoSQL)

- Key-Value Pair (KVP) Databases
 - Redis, Riak, RocksDB
- Document Databases
 - MongoDB, CouchDB
- Wide Column Databases
 - Hbase, Cassandra, Druid, Vertica
- Graph Databases
 - Neo4J, Allegro, InfiniteGraph, OrientDB, Virtuoso

NoSQL Databases: Key-Value Stores



Key-value Stores

- Minimal set of functions:
 - void put(key,value)
 - value = get (key)
 - void delete(key)
- Might implement additional features, e.g. collections, merge operator, purge operator,...
- Does not differentiate between datatypes, i.e. the content of arbitrary documents can be stored (in a binary format)

Wide Column Stores

- Not the same as column stores (see last lecture)
- Image source: <https://vishnuch.tech/4-types-of-nosql-databases>, accessed: 21.02.2022
- Suitable for distributed systems
- Queries can look similar to queries in key-value store, e.g. (Hbase):
 - put 'students','student1','account:name','Alice'
 - put 'students','student1','address:country','GB'
- Queries can also look similar to SQL, e.g. (Hbase):

```
SELECT CONVERT_FROM(row_key, 'UTF8') AS studentid,  
       CONVERT_FROM(students.account.name, 'UTF8') AS name,  
       CONVERT_FROM(students.address.country, 'UTF8') AS country  
  FROM students;
```

Document Stores

- Image source: <https://lennilobel.wordpress.com/2015/06/01/relational-databases-vs-nosql-document-databases/>
- Usually supports json and/or xml
- Allows querying individual fields of the supported documents
- Different languages might be supported, e.g. (MongoDB shell):
 - db.inventory.find({ status: "A" }) (*inventory is a collection*)

NoSQL Databases: Graph DBs

Nodes (Dots)

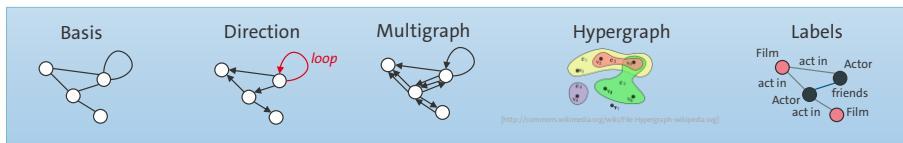


- Like an entity in ER
- Exist on their own
- Have object identity

Edges (Lines)



- Like a relationship in ER
- Exist only between nodes
- Identity depends on the nodes they connect



16

NoSQL

Most popular GraphDB: Neo4j

Basis

- Assuming a domain of objects \mathcal{O}
- A graph is a structure (V, E)
- Vertices are objects: $V \subseteq \mathcal{O}$
- Edges are 2-element subset of vertices: $E \subseteq \{\{x, y\} | x, y \in V\}$

Direction

- Edges are ordered pairs of vertices $E \subseteq V \times V$

Multigraph

- Multiple edges per node
- A graph is a structure (V, E, ρ)
- Edges are objects: $E \subseteq \mathcal{O}$
- $\rho: E \rightarrow \{\{x, y\} | x, y \in V\}$ (undirected) / $\rho: E \rightarrow V \times V$ (directed)
is a function assigning to each edge
a 2-element subset of / ordered pair of vertices

Hypergraphs

- Edges can connect more than two vertices
- $G=(V,E)$ with $E \subseteq 2^V$

Labels

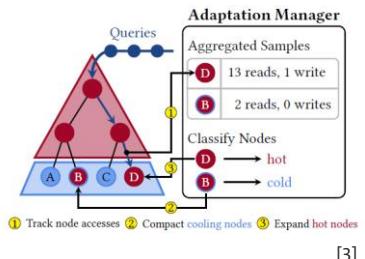
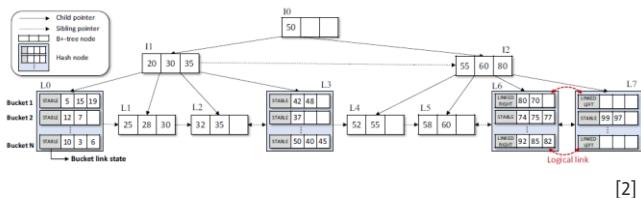
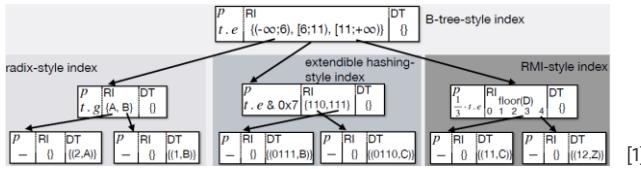
- Descriptive type information tagged to the objects
- Indicate which kind of real world entity an object represents
- Assuming a domain of labels \mathcal{L}
- A graph is a structure (V,E,λ)
- $\lambda:V \rightarrow \mathcal{L}$ (vertex label)
 $\lambda:E \rightarrow \mathcal{L}$ (edge label)
 $\lambda:V \cup E \rightarrow \mathcal{L}$ (vertex and edge label)
 $\lambda:V \rightarrow 2^{\mathcal{L}}$ (vertex labels)
 $\lambda:E \rightarrow 2^{\mathcal{L}}$ (edge labels)
 $\lambda:V \cup E \rightarrow 2^{\mathcal{L}}$ (vertex and edge labels)
- is a partial / total function assigning to each vertex / edge / vertex and edge a label / a set of labels

Food for thought: (No)SQL

- Which kind of database would be suited for the following use cases in your opinion?
 - A social network
 - The cover text of books and their ISBN number
 - The mapping between students/employees and their transponder number
 - The structure of different molecules (= atoms and their bindings)
 - A lab book
 - The inventory of a library
 - Banking

Yes, multiple systems might be suitable if there is a good explanation

Addendum on Optimizations: Hybrid Indexes

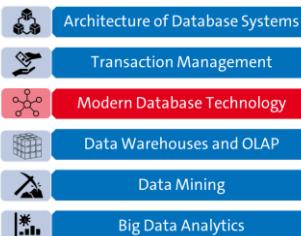


[1] Dittrich, Jens, Joris Nix, and Christian Schön. "The next 50 years in database indexing or: the case for automatically generated index structures." *Proceedings of the VLDB Endowment* 15.3 (2021): 527-540.

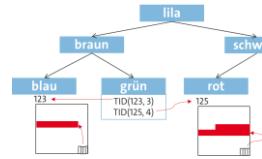
[2] Cha, Hokeun, et al. "Blink-hash: An adaptive hybrid index for in-memory time-series databases." *Proceedings of the VLDB Endowment* 16.6 (2023): 1235-1248.

[3] Anneser, Christoph, et al. "Adaptive hybrid indexes." *Proceedings of the 2022 International Conference on Management of Data*. 2022.

Summary

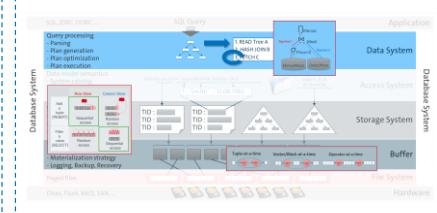


User optimizations: Indexes and Materialized Views

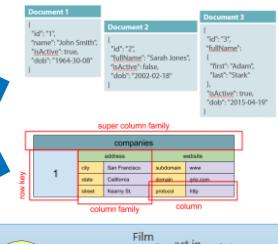


CREATE MATERIALIZED VIEW CheapFood AS
 SELECT Meal FROM MensaMeals
 WHERE Price < 5;
 REFRESH MATERIALIZED VIEW CheapFood;

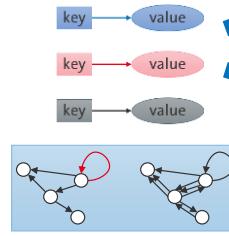
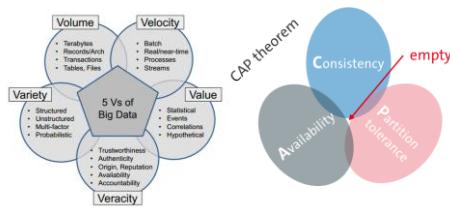
Wrap up optimizations



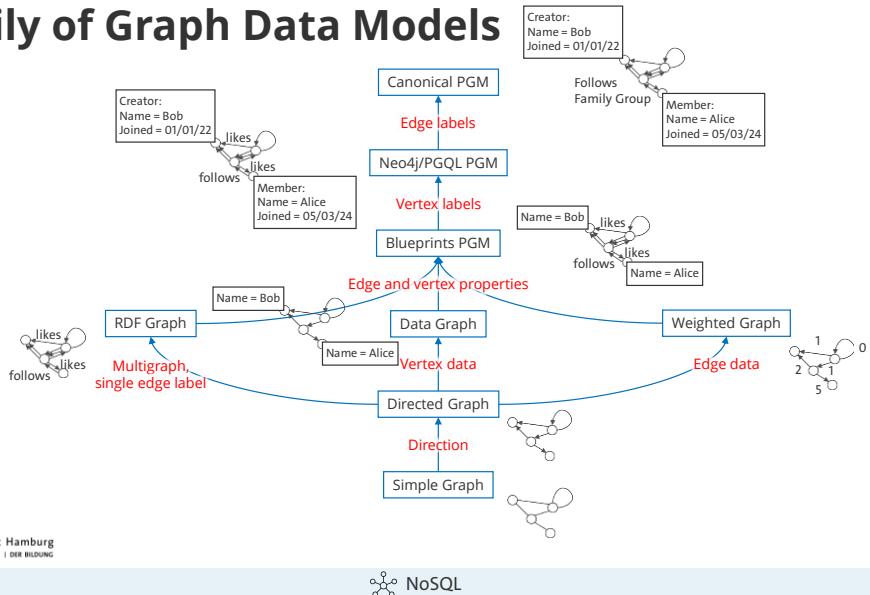
NoSQL Systems



NoSQL DBS: Challenges & Characteristics



Family of Graph Data Models



PGM - Property Graph Model

RDF - Resource Description Framework

- Stored in triple format (node, edge label, node)
- Each property must be modeled as a triple Label: Descriptive Type information

PGQL – Property Graph Query Language (tries to look like SQL)

Property/Data - Actual data

Label – Descriptive (type) information

- Labels are sometimes called “type”, especially in the context of edges

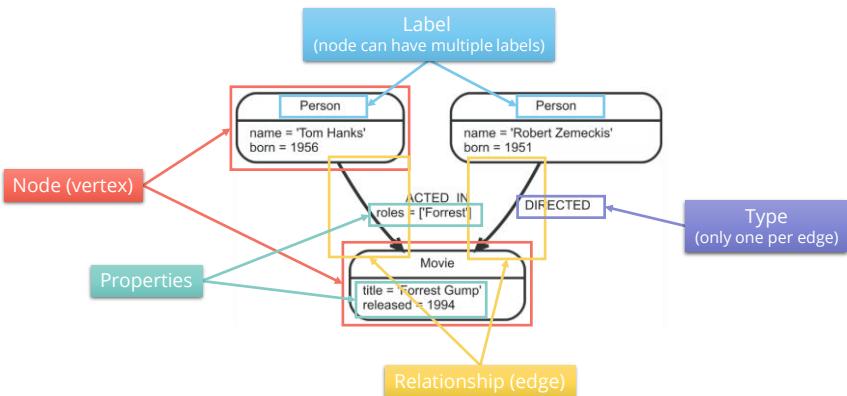
Properties/Data:

- Assuming a domain of data / values \mathcal{D} and a domain of property keys / attributes \mathcal{K}
- A graph is a structure (V, E, ν)
- $\nu: V \rightarrow \mathcal{D}$ (vertex data)
- $\nu: E \rightarrow \mathcal{D}$ (edge data)
- $\nu: V \cup E \rightarrow \mathcal{D}$ (vertex and edge data)
- $\nu: V \times \mathcal{K} \rightarrow \mathcal{D}$ (vertex properties)
- $\nu: E \times \mathcal{K} \rightarrow \mathcal{D}$ (edge properties)

$v: (V \cup E) \times \mathcal{K} \rightarrow \mathcal{D}$ (vertex and edge properties)
is a partial function assigning data / values to each
vertex / edge / vertex and edge

Neo4j PGM

[<http://neo4j.com/docs/developer-manual/current/introduction/#graphdb-concepts>]



Property Graph Modelling

Entity with Properties

- Entity relationship model



- Schema typically implicit, i.e. given with instances

(ja:Person { name: 'Jane Austen', yearOfBirth: 1775 })

- Instance



Vertex id is system generated in Neo4j

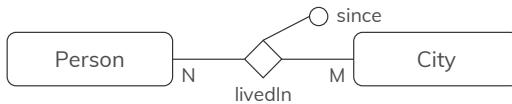
Cypher (Neo4j) uses the keyword CREATE to create a new vertex, e.g.

CREATE (ja:Person { name: 'Jane Austen', yearOfBirth: 1775 })

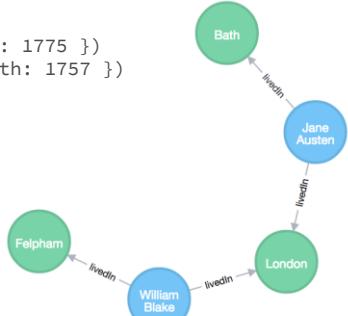
Property Graph Modelling

Relationships (N:M)

- Entity relationship model



- Vertices
(ja:Person { name: 'Jane Austen', yearOfBirth: 1775 })
(wb:Person { name: 'William Blake', yearOfBirth: 1757 })
(lo:City {name: 'London'})
(ba:City {name: 'Bath'})
(fe:City {name: 'Felpham'})
- Edges
(ja)-[:livedIn {since: 1775}]->(lo)
(ja)-[:livedIn {since: 1800}]->(ba)
(wb)-[:livedIn {since: 1757}]->(lo)
(wb)-[:livedIn {since: 1800}]->(fe)



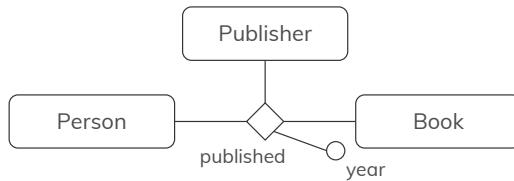
An edge can be created with the CREATE keyword in Cypher, just like vertices,
e.g.

CREATE (ja)-[:livedIn {since: 1775}]->(lo)

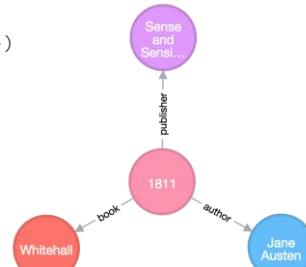
Property Graph Modelling

n-ary relationships (with n>2)

- Entity relationship model



```
(ja:Person { name: 'Jane Austen', yearOfBirth: 1775 })
(wh:Publisher { name: 'Whitehall' })
(sas:Book {title: 'Sense and Sensibility' })
(pub:Publication {year: 1811 })
(pub)-[:author]->(ja)
(pub)-[:book]->(wh)
(pub)-[:publisher]->(sas)
```

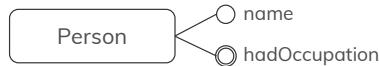


Not directly expressible since edge connect exactly two vertices, not more, not less

→ Solution: Reification of relationship

Property Graph Modelling

Multivalued properties



(wb:Person {name: 'William Blake', hadOccupation: ['Poet','Painter','Printmaker']})



Person <id>: 183 hadOccupation: Poet,Painter,Printmaker name: William Blake

Expressibility depends on datatype system of specific system in use

→ Possible in Neo4j (see slide)

→ If not expressible: Reification of values of multivalued property

Multiple labels can be connected with a : or a & in Neo4j, e.g. (wb:Person&Artist {name: 'William Blake', hadOccupation: ['Poet','Painter','Printmaker']})

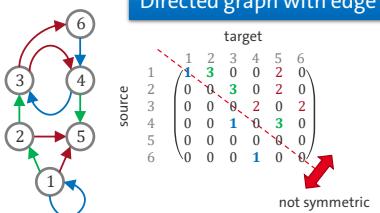
Exercise Neo4j

- Your fridge is stocked with cheese, butter, milk, and pizza
- There are also 4 tomatoes and two bars of chocolate in the kitchen
- Each item has an expiration date and a quantity: cheese (15 May, 10), butter (01 June, 1), milk (11 May, 2), Pizza (25 Aug, 5)
- You have a recipe for luxury frozen pizza that is called “PiDeLuxe”. It requires a slice of cheese and a tomato per frozen pizza. You want to prepare one Pizza “PiDeLuxe” today.

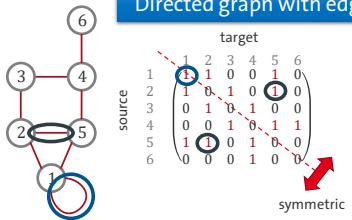
How can this be modeled using the ER model?

How could it be expressed with Neo4j?

Representations: Adjacency Matrix



Directed graph with edge labels

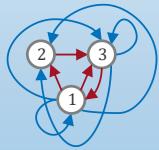


target						
1	2	3	4	5	6	
1	0	0	0	0	0	
0	1	0	1	0	0	
0	1	0	1	0	0	
0	0	1	0	1	1	
1	1	0	1	0	0	
0	0	1	1	0	0	
0	0	0	1	0	0	

Formalisms to define graph operations

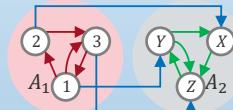
Reachability

$$\begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$



Graph Isomorphism

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$



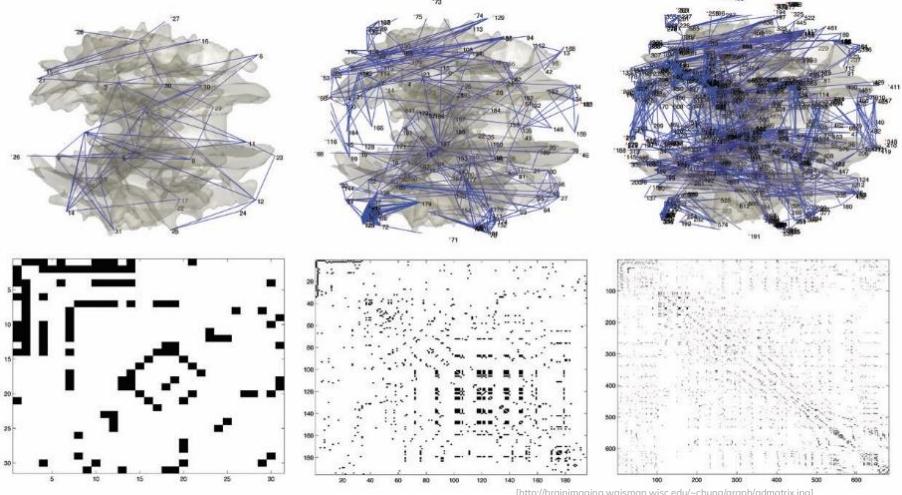
 NoSQL

Reachability: $A^2 = A \cdot A$ shows all paths of length 2 between nodes of graph represented by A

Graph isomorphism: Graphs represented by A_1 and A_2 are isomorphic if there exists a permutation matrix P such that $PA_1P^{-1}=A_2$

Note: Most definitions work only for unlabeled graphs!

Adjacency Matrix



14

NoSQL

Adjacency matrices are often sparse

→ Compression

Compress Sparse Row (CSR)

$$\begin{pmatrix} 0 & 1 & 2 & 3 \\ a & b & c & e \\ d & i & f & \\ h & g & & \end{pmatrix}$$

Position for row # in other two arrays:

Row position array:

#	0	1	2	3
	0	2	4	7

Column index array:

#	0	1	2	3	4	5	6	7	8
	0	2	1	3	0	1	2	1	2

Cell value array:

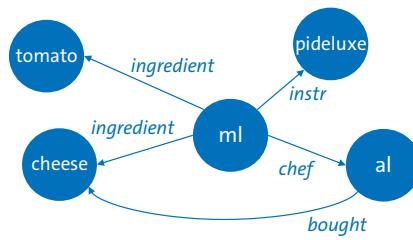
a	c	b	e	d	i	f	h	g
---	---	---	---	---	---	---	---	---

15

NoSQL

Also works column-wise → Compress Sparse Column (CSC)

Exercise CSR



What does the adjacency matrix for this graph look like?
Apply Compress Sparse Row.

Representations: Adjacency List

Source vertex with outgoing edges...

0	0	1	2	3
1	a	b	c	e
2	d	i	f	
3	h	g		

...without edge labels

0 -> (0,2)
 1 -> (1,3)
 2 -> (0,1,2)
 3 -> (1,2)

...with edge labels

0 -> ([0,a],[2,c])
 1 -> ([1,b],[3,e])
 2 -> ([0,d],[1,i],[2,f])
 3 -> ([1,h],[2,g])

...with edge properties

0 -> ([0,a,(weight=4)],[2,c,(weight=3)])
 1 -> ([1,b,(weight=3)],[3,e,(weight=2)])
 2 -> ([0,d,(weight=5)],[1,i,(weight=2)],...)
 3 -> ([1,h,(weight=9)],[2,g,(weight=7)])

The same!

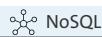
Almost the same!

Compressed
Sparse Row

Compressed
Sparse Column

source-oriented

target-oriented



Representations: Relational Representations I

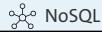
Triple Table: A table with three columns: subject, predicate, object

Subject	Predicate	Object
<http://www.w3.org/People/EM/contact#me>	<http://www.w3.org/2000/10/swap/pim/contact#fullName>	"Eric Miller"
<http://www.w3.org/People/EM/contact#me>	<http://www.w3.org/2000/10/swap/pim/contact#mailbox>	<mailto:e.miller123(at)example>
<http://www.w3.org/People/EM/contact#me>	<http://www.w3.org/2000/10/swap/pim/contact#personalTitle>	"Dr."
<http://www.w3.org/People/EM/contact#me>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2000/10/swap/pim/contact#Person>

Vertex and Edge Table: Two universal tables, one for vertices, one for edges

ID	Type	Color	Name	RAM	Nationality	Source	Target	Type	Rating
1	Product	black	"Apple iPad MC707LL/A"	64 GB		1	7	in	
...
4	Category		"Cell Phones & Accessories"			5	4	part of	
5	Category		"Phones"			7	6	part of	
...

19



Triple Table

- Generic, simple, straightforward approach
- Can be built on relational storage engines
- Add indexing for faster lookups
- Queries need to operate on all triples (no data partitioning based on schema information)
- High storage redundancy

Vertex and Edge Table

- Requires efficient handling of NULL values
 - Given for instance in column stores
- Queries need to operate on all vertices/edges (no data partitioning based on schema information)

Further reading

Daniel J. Abadi et al. Scalable Semantic Web Data Management Using Vertical Partitioning. VLDB 2007

Representations: Relational Representations II

Clustered Property Table: Groups properties that tend to be defined together in a table

Property Table

Subj.	Type	Title	copyright
ID1	BookType	"XYZ"	"2001"
ID2	CDType	"ABC"	"1985"
ID3	BookType	"MNP"	NULL
ID4	DVDType	"DEF"	NULL
ID5	CDType	"GHI"	"1995"
ID6	BookType	NULL	"2004"

Left-Over Triples

Subj.	Prop.	Obj.
ID1	author	"Fox, Joe"
ID2	artist	"Orr, Tim"
ID2	language	"French"
ID3	language	"English"

Property-Class Table: Cluster similar sets of subjects together in the same table

Class: BookType

Subj.	Title	Author	copyright
ID1	"XYZ"	"Fox, Joe"	"2001"
ID3	"MNP"	NULL	NULL
ID6	NULL	NULL	"2004"

Class: CDType

Subj.	Title	Artist	copyright
ID2	"ABC"	"Orr, Tim"	"1985"
ID5	"GHI"	NULL	"1995"

Left-Over Triples

Subj.	Prop.	Obj.
ID2	language	"French"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"

20

NoSQL

Clustered Property Table

- Example:
 - Properties: type, title, and copyright date
 - Table stores all triples with one of these properties
- Multiple property tables with different clusters of properties may be created
- One particular property may only appear in at most one property table
- Triples that not fit any property table are stored in a left-over triple table
- Multivalued attributes are problematic
- Queries that have unspecified property are problematic
- Introduce NULL values

Property-Class Table

- Exploits the type property of subjects
- A property may exist in multiple property-class tables
- A subject may also exist in multiple tables
- Multivalued attributes are problematic
- Queries that do not select on class type are problematic
- Introduce NULL values

Property Table Approaches

Triple Table

Subj.	Prop.	Obj.
ID1	type	BookType
ID1	title	"XYZ"
ID1	author	"Fox, Joe"
ID1	copyright	"2001"
ID2	type	CDType
ID2	title	"ABC"
ID2	artist	"Orr, Tim"
ID2	copyright	"1985"
ID2	language	"French"
ID3	type	BookType
ID3	title	"MNO"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"
ID5	type	CDType
ID5	title	"GHI"
ID5	copyright	"1995"
ID6	type	BookType
ID6	copyright	"2004"

(Clustered) property table

Property Table

Subj.	Type	Title	copyright
ID1	BookType	"XYZ"	"2001"
ID2	CDType	"ABC"	"1985"
ID3	BookType	"MNP"	NULL
ID4	DVDType	"DEF"	NULL
ID5	CDType	"GHI"	"1995"
ID6	BookType	NULL	"2004"

Property-Class Table

Class: BookType

Subj.	Title	Author	copyright
ID1	"XYZ"	"Fox, Joe"	"2001"
ID3	"MNP"	NULL	NULL
ID6	NULL	NULL	"2004"

Class: CDType

Subj.	Title	Artist	copyright
ID2	"ABC"	"Orr, Tim"	"1985"
ID5	"GHI"	NULL	"1995"

Left-Over Triples

Subj.	Prop.	Obj.
ID1	author	"Fox, Joe"
ID2	artist	"Orr, Tim"
ID3	language	"French"

Subj.	Prop.	Obj.
ID2	language	"French"
ID3	language	"English"
ID4	type	DVDType
ID4	title	"DEF"

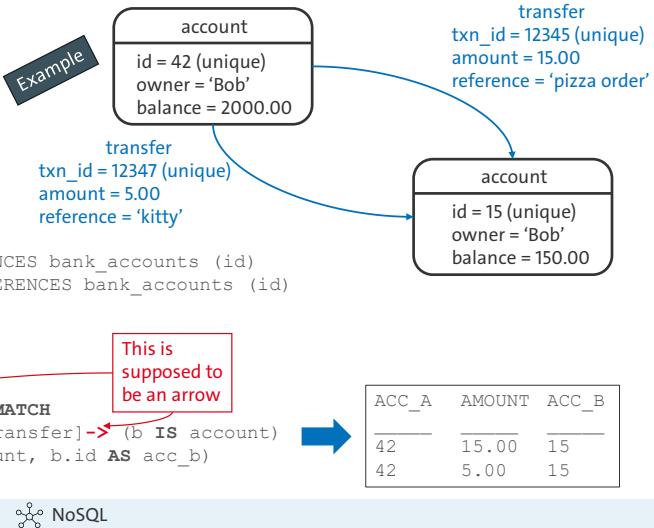
Reduce numbers of subject-subject self joins necessary to reconstruct entities

NoSQL

Property Graph Queries in SQL

```
CREATE PROPERTY GRAPH bank_sql_pg
VERTEX TABLES (
    bank_accounts
    KEY (id)
    LABEL account
    PROPERTIES (owner, balance)
)
EDGE TABLES (
    bank_txns
    KEY (txnid)
    SOURCE KEY (from_acct_id) REFERENCES bank_accounts (id)
    DESTINATION KEY (to_acct_id) REFERENCES bank_accounts (id)
    LABEL transfer
    PROPERTIES (amount, reference)
);
```

```
SELECT * FROM GRAPH_TABLE (bank_sql_pg MATCH
(a IS account WHERE a.id = 42) -[e IS transfer]-(b IS account)
COLUMN (a.id AS acc_a, e.amount AS amount, b.id AS acc_b)
);
```



- PGQs were recently adopted in the SQL standard (2023): ISO/IEC DIS 9075-16
- First implementation in Oracle version 23.2: <https://docs.oracle.com/en/database/oracle/property-graph/23.2/spgdg/sql-property-graphs.html>
- **Only because PGQs are supported by SQL, there is no guarantee that the DBS implementing it is optimized for graph processing!**

Databases and Information Systems (DIS) - Quiz

Universität Hamburg

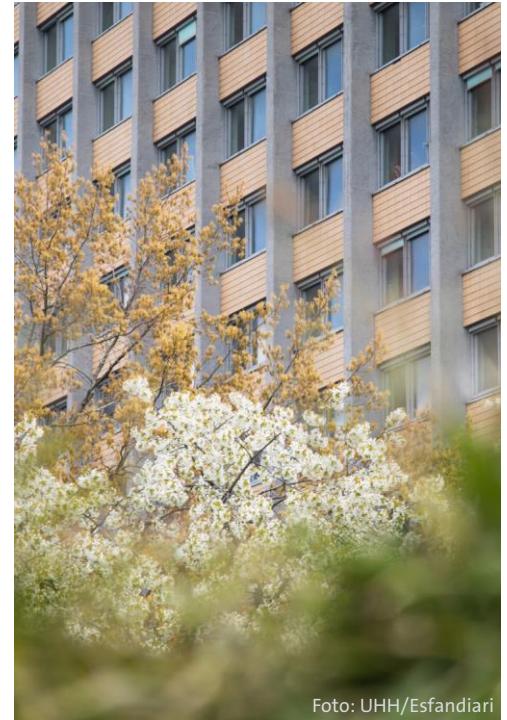
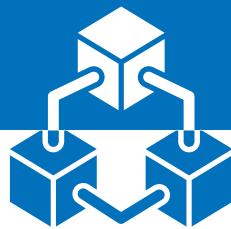


Foto: UHH/Esfandiari

1

Architecture of Database Systems



Which of the following layers are a part of the 5-Layer-Model?

Data System

Access System

Network System

Buffer

Which of the following layers are a part of the 5-Layer-Model?

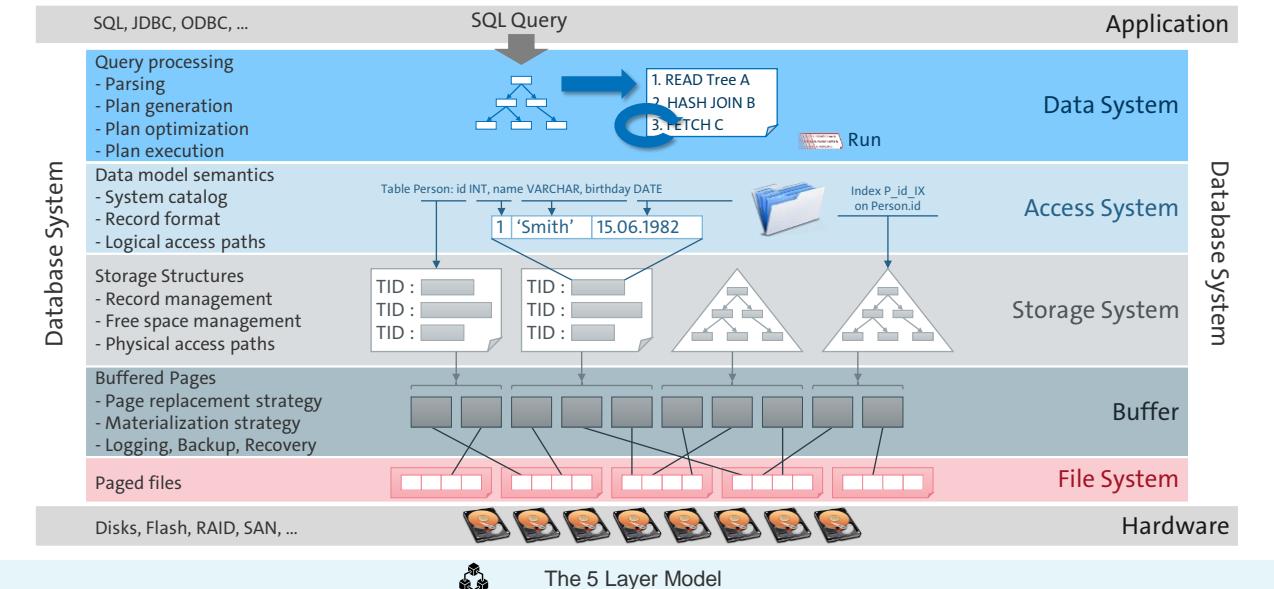
Data System

Access System

Network System

Buffer

The 5 Layer Model

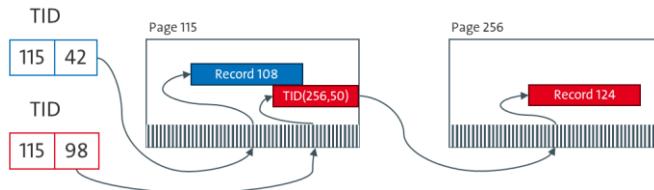


5

Explain with your own words what happens to the TID of a record when it is getting larger. What happens if the record is migrated afterwards?

Explain with your own words what happens to the TID of a record when it is getting larger. What happens if the record is migrated afterwards?

1. Nothing happens to the original TID.
2. The original TID still stays the same.
3. In both cases the TID of the current version of the record is stored at the address found via the original TID.



Long answer

Case 1: Space on the page is sufficient to store larger record

→ All records are moved within the same page, in the page array, the positions of the records on the page are changed (same as with a deleted record), but not the TID

Case 2: Space on the page is not sufficient to store larger record

→ Record is moved to another page and TID is stored on the original page (see Figure). If the record is moved again later, the TID in the original page is changed again → only one additional reference necessary even if record is changed multiple times

7

Transaction Management: Synchronization



Which of the following describe an anomaly that can happenin a Database System without synchronization?

Phantom Problem

Unrepeatable Write

Unrepeatable Read

Lost Update

Dirty Read

Destructive Write

Which of the following describe an anomaly that can happenin a Database System without synchronization?

Phantom Problem

Unrepeatable Write

Unrepeatable Read

Lost Update

Dirty Read

Destructive Write

Anomalies

Salary change T₁

```
SELECT SALARY INTO :salary
FROM EMPL
WHERE ENR = 2345

salary := salary + 2000;

UPDATE EMPL
SET SALARY = :salary
WHERE ENR = 2345
```

Salary change T₂

```
SELECT SALARY INTO :salary
FROM EMPL
WHERE ENR = 2345

salary := salary + 1000;

UPDATE EMPL
SET salary = :salary
WHERE ENR = 2345
```

Database
(ENR, SALARY)

2345	39.000
2345	41.000
2345	40.000

→ Lost Update

time

11

⌚ Synchronization

11

Anomalies (II)

Salary change T₁

```
UPDATE EMPL
SET SALARY = SALARY + 1000
WHERE ENR = 2345
```

ROLLBACK

Salary change T₂

```
SELECT SALARY INTO :salary
FROM EMPL
WHERE ENR = 2345

salary := salary * 1.05;

UPDATE EMPL
SET salary = :salary
WHERE ENR = 2345

COMMIT
```

Database
(ENR, SALARY)

2345	39.000
2345	40.000
2345	42.000
2345	39.000

Dirty Read

time

12

⌚ Synchronization

12

Anomalies (III)

Salary change T₁

```
UPDATE EMPL
SET SALARY = SALARY + 1000
WHERE ENR = 2345
UPDATE EMPL
SET SALARY = SALARY + 2000
WHERE ENR = 3456
COMMIT
```

Get salaries T₂

```
SELECT SALARY INTO :g1
FROM EMPL
WHERE ENR = 2345

SELECT SALARY INTO :g2
FROM EMPL
WHERE ENR = 3456

sum := g1 + g2
```

Database (ENR, SALARY)

2345	39.000
3456	45.000
2345	40.000
3456	47.000

Non-Repeatable Read

What is the result for sum and which result did we expect?

time

13

 Synchronization

13

Anomalies (IV)

Get salaries T₁

```
SELECT SUM(Salary)
INTO :Sum1
FROM Empl
WHERE DepNr = 17
```

```
SELECT SUM(Salary)
INTO :Sum2
FROM Empl
WHERE DepNr = 17
```

Create new record T₂

```
INSERT INTO Empl(ENR, DepNr, Salary)
Values( 4567, 17, 55.000)
COMMIT
```

Database (ENR, DepNr, Salary)

2345	17	39.000
3456	17	45.000
...		

Sum

84.000

Phantom Problem

2345	17	39.000
3456	17	45.000
...		
4567	17	55.000

139.000

time

14

 Synchronization

14

Which ANSI-SQL isolation level is supposed to avoid all anomalies?

Read Uncommitted

Read Committed

Repeatable Read

Serializable

Which ANSI-SQL isolation level is supposed to avoid all anomalies?

Read Uncommitted

Read Committed

Repeatable Read

Serializable

ANSI-SQL isolation levels

Isolation Level	Lost Update possible?	Dirty Read possible?	Non-Repeatable Read possible?	Phantom Read possible?
Read Uncommitted	No	Yes	Yes	Yes
Read Committed	No	No	Yes	Yes
Repeatable Read	No	No	No	Yes
Serializable	No	No	No	No

Are two serializable schedules of the same transactions always equivalent? Elaborate on your answer.

Are two correct, i.e. serializable, schedules of the same transactions always equivalent? Elaborate on your answer.

No, e.g. assume that two transactions, T_1 and T_2 , are executed without concurrency, i.e. all operations of the same transaction are finished before the other transaction starts.

→ Whether T_1 is executed first or T_2 is executed first does not matter in term of correctness. But the result might be different depending on which transaction starts, e.g. if non-commutative operations are used.

Correctness and Equivalency

A:= 1000, B:= 2000	
T_1	T_2
read(A)	read(A)
A:=A-50	X := A*0.1
write(A)	A := A-X
read(B)	read(B)
B := B+50	B := B+X
write(B)	write(B)

Both schedules are correct, but they are not equivalent!

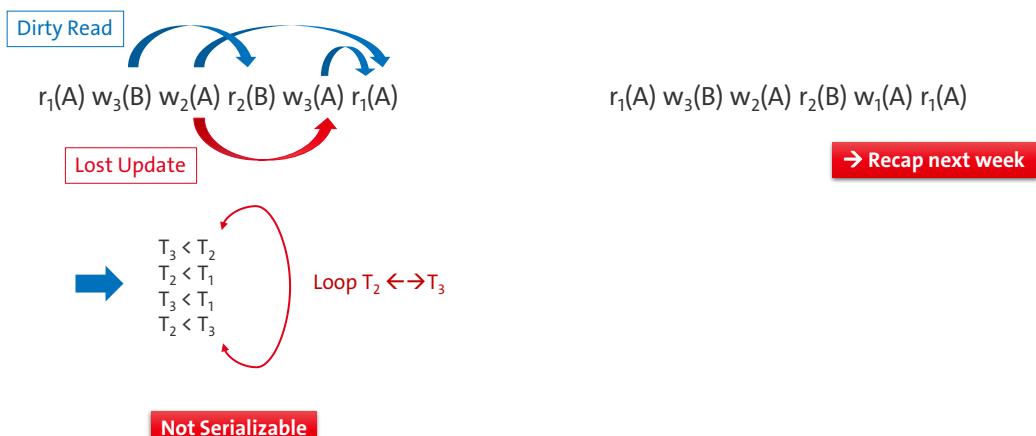
T_1	T_2
read(A)	read(A)
A:=A-50	X := A*0.1
write(A)	A := A-X
read(B)	read(B)
B := B+50	B := B+X
write(B)	write(B)
read(A)	read(A)
X := A*0.1	A := A-X
A := A-X	write(A)
write(A)	read(B)
read(B)	B := B+X
B := B+X	write(B)
write(B)	
Results	Results
$A = (1000 - 50) - (1000 - 50) * 0,1 = 855$	$A = 1000 - (1000 * 0,1) - 50 = 850$
$B = 2000 + 50 + (1000 - 50) * 0,1 = 2145$	$B = 2000 + (1000 * 0,1) + 50 = 2150$

Which of the following schedules are serializable?

$r_1(A) w_3(B) w_2(A) r_2(B) w_3(A) r_1(A)$

$r_1(A) w_3(B) w_2(A) r_2(B) w_1(A) r_1(A)$

Which of the following schedules are serializable?



Transaction Management: Logging & Recovery



Which phases do we typically execute during a recovery after a crash?

Which phases do we typically execute during a recovery after a crash?

Analysis

Redo

Undo

Which of the following statements is not correct?

During the analysis phase, winners and losers are identified.

The Redo phase is applied to winners and losers.

The Undo phase is applied to winners and losers.

Compensation Log Records are created during the Undo phase.

Which of the following statements is not correct?

During the analysis phase, winners and losers are identified.

The Redo phase is applied to winners and losers.

The Undo phase is applied to winners and losers.

Compensation Log Records are created during the Undo phase.

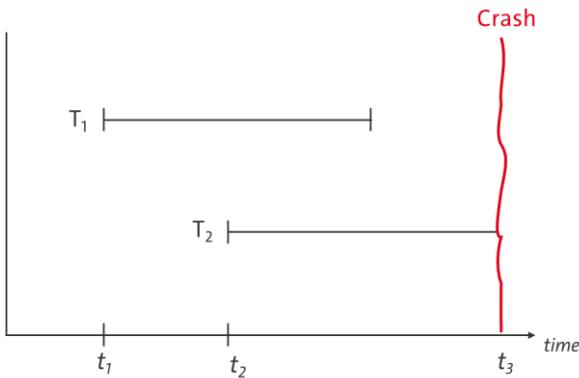
The undo phase is only applied to losers

28

 Logging and Recovery

28

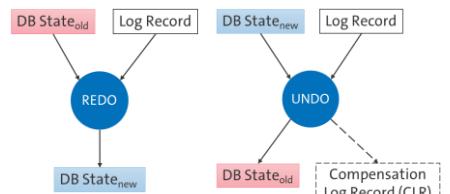
Recovery



Transaction T₁ is a winner → Redo
Transaction T₂ is a loser → Undo

Phases of Recovery

1. **Analysis**
→ Identify winners and losers
2. **Redo**
→ Repetition of history
→ Reads log file forwards
3. **Undo of losers**
→ Reads log file backwards
→ Write CLR



29

 Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

 Logging & Recovery

29

Recovery Example

[#1, T₁, BOT, 0]
 [#2, T₂, BOT, 0]
 [#3, T₁, A, A-=50, A+=50, #1]
 [#4, T₂, C, C+=100, C-=100, #2]
 [#5, T₁, B, B+=50, B-=50, #3]
 [#6, T₁, commit, #5]
 [#7, T₂, A, A-=100, A+=100, #4]

Crash

<#7', T₂, A, A+=100, #7, #4>
 <#4', T₂, C, C-=100, #7', #2>
 <#2', T₂, -, -, #4', 0>

Analysis

Winner: T₁
Loser: T₂

Redo

IF LSN(A) < 3 THEN A-=50 (and replace LSN(A))
 IF LSN(C) < 4 THEN C+=100 (and replace LSN(C))
 IF LSN(B) < 5 THEN B+=50 (and replace LSN(B))
 IF LSN(A) < 7 THEN A-=100 (and replace LSN(A))

Undo → Only for losers (T₂)

Entry #7

- A+=100
- Write CLR
- LSN(A) = #7'

Entry #4

- C-=100
- Write CLR
- LSN(C) = #4'

Entry #2

- Write CLR

30

Which of the following checkpoints introduces the longest system downtime?

Action Consistent CP

Transaction Consistent CP

Fuzzy CP

31

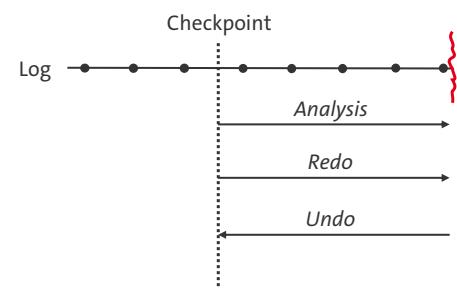
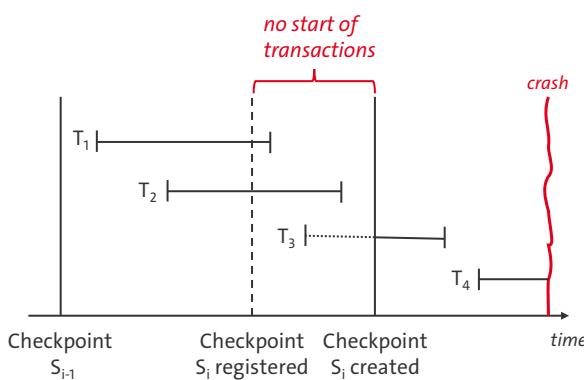
Which of the following checkpoints introduces the longest system downtime?

Action Consistent CP

Transaction Consistent CP

Fuzzy CP

Transaction Consistent Checkpoints



Modern DBS: Distributed Systems



Which of the following Commit Protocols is the most efficient in terms of the least number of messages and log entries created?

1PC

2PC

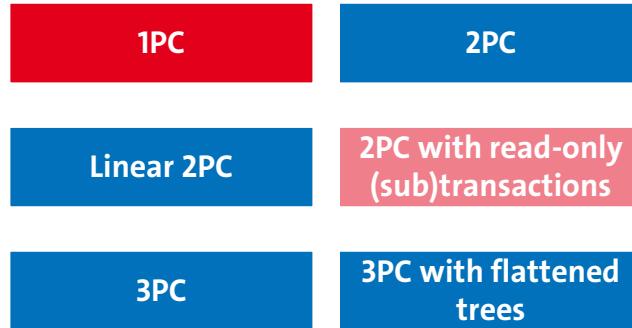
Linear 2PC

2PC with read-only transactions

3PC

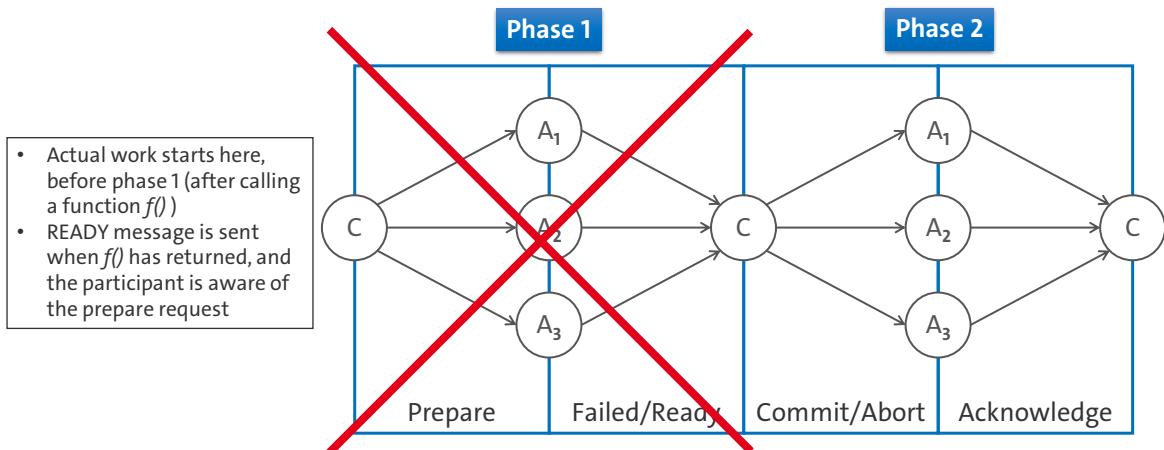
3PC with flattened trees

Which of the following Commit Protocols is the most efficient in terms of the least number of messages and log entries created?



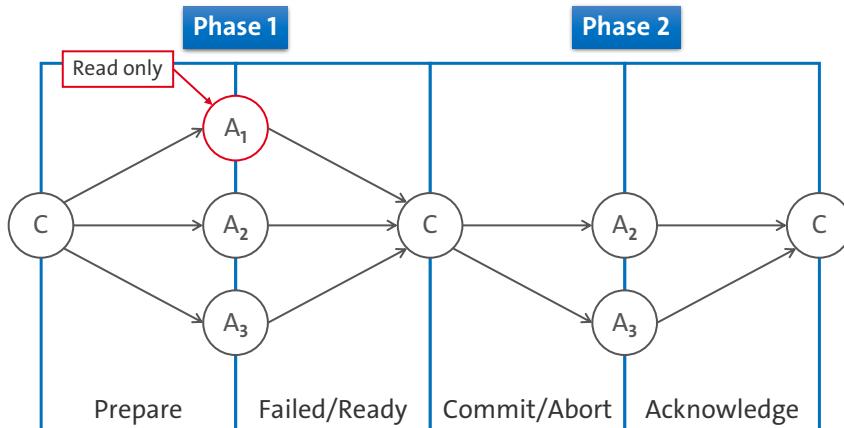
If all transactions are read-only, 2PC w/ read-only TAs require just as many messages as 1PC.

One-Phase-Commit Protocol



Read-Only (Sub-)Transactions with 2PC

No 2nd phase if transaction is only reading



Explain one taxonomy for Heterogeneous Federated Databases of your choice

Explain one taxonomy for heterogeneous Federated Databases of your choice

- By autonomy
 - Homogeneous: Local nodes run no local transactions
 - Heterogeneous: Local nodes can run local transactions
- By data coupling
 - Loosely coupled → Local stores accessed by their local language or a common language
 - Tightly coupled → Local stores accessed by a multistore system using the same language for structured and unstructured data (e.g. Hadoop)
 - Hybrid → Some Stores are loosely coupled and some are tightly coupled (e.g. Spark SQL, BigDawg)
- By Data and query language
 - Homogeneous: Data format/storage/location and query language are the same
 - Heterogeneous: Data and/or query language differs

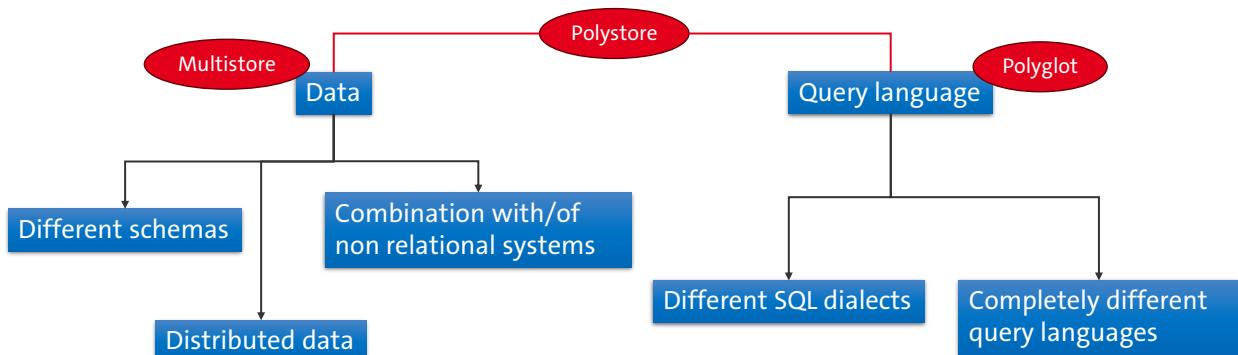
Further reading

An approach to classify Federated DBMS:

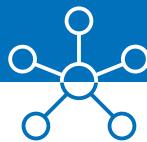
Azevedo, Leonardo Guerreiro, et al. "Modern Federated Database Systems: An Overview." ICEIS (1) (2020): 276-283.

Heterogeneous Federated DBs

Components can vary in different aspects → This is one of various possible taxonomies



Modern DBS: Optimizations



42

Which of the following storage layouts is he most efficient for analytical queries? Why?

Row Store

Column Store

43

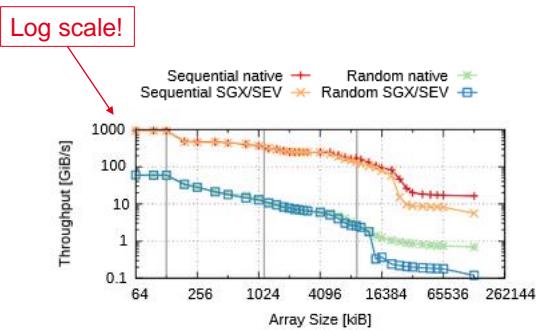
Which of the following storage layouts is he most efficient for analytical queries? Why?

Row Store

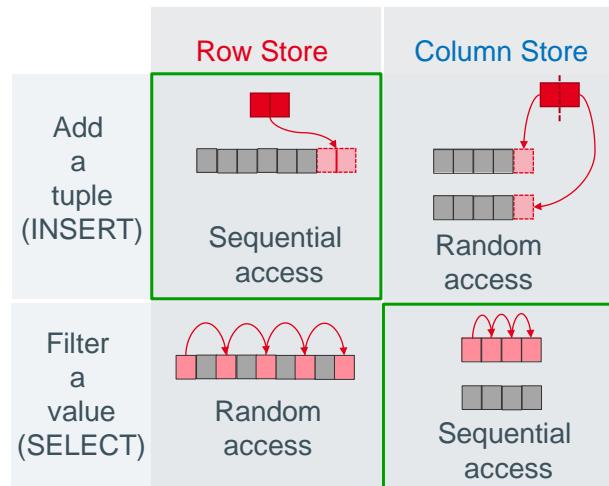
Column Store

Why should you care?

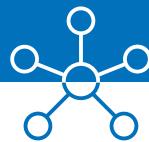
Memory access is expensive!



*Throughput on an Intel Xeon E3-1275
Gottel, Christian & Pires, Rafael & Rocha, Isabela & Vaucher, Sébastien & Felber, Pascal & Pasin, Marcelo & Schiavoni, Valerio. (2018). SRDS 2018



Modern DBS: NoSQL Systems



46

Which of the following paradigms is applied in most NoSQL Systems?

ACID

BASE

47

Which of the following paradigms is applied in most NoSQL Systems?

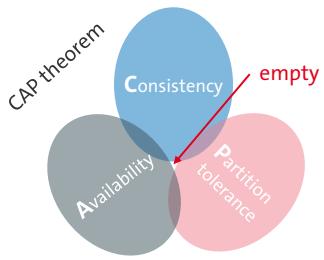
ACID

BASE

- Basically available: The database is available for changes, even if these changes are applied later, i.e. it is not necessarily in a consistent state when the user accesses it
- Soft state: Data can have a temporary state, e.g. when multiple applications change it at the same time
- Eventually consistent: Consistency is reached eventually, but only when (finally) all changes have been applied that were made at the same time

What is the CAP theorem and how is it connected to the BASE and ACID paradigms?

What is the CAP theorem and how is it connected to the BASE and ACID paradigms?



- In a distributed system, only 2 of the 3 properties Availability, Consistency, and partition tolerance can be reached at the same time
- ACID prioritizes Consistency (CA, CP)
- BASE prioritizes Availability (AP)

Summary

Quiz

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

Graph Data Models

...without edge labels
0 > (0,2)
1 > (1,3)
2 > (0,1,2)
3 > (1,2)

Graph Representations

Position for row # in other two arrays:
Row position array: [0 1 2 3 4 5 6 7]
Column index array: [0 2 1 3 0 1 2 1 2]
Cell value array: [a c b e d i f h g]

Adjacency matrix
...with edge labels
0 > ([0,a],[2,d])
1 > ([1,b],[3,h])
2 > ([0,d],[1,f],[2,l])
3 > ([1,l],[0,g])
...with edge properties
0 > ([0,a],[2,d])
1 > ([1,b],[3,e],[weight=3])
2 > ([0,d],[1,f],[2,l],[weight=2])
3 > ([1,l],[0,g],[weight=1])

Relational Representation

Triple Table			Customized property table			Property Class Table		
Subj	Prop	Obj	Subj	Type	Title	Subj	Title	Author
B01	type	BookType	B01	COPYTYPE	ABC	B01	NAME	NULL
B01	author	Woo, Joo	B02	NAME	NULL	B02	NAME	NULL
B01	copyright	2000	B03	PUBTYPE	NULL	B03	NAME	2000
B01	title	"Van Gogh"	B04	PUBTYPE	OPP	B04	NAME	NULL
B01	language	French	B05	language	French	B05	language	French
B01	bookType	BookType	B06	language	English	B06	language	English
B01	copyright	2000	B07	title	NULL	B07	title	NULL
B01	author	NULL	B08	author	NULL	B08	author	NULL
B01	name	NULL	B09	name	NULL	B09	name	NULL

PGQs in SQL

```

CREATE PROPERTY GRAPH bank_sql_pg
VERTEX TABLES (
    bank_accounts
    KEY (id)
    LABEL account
    PROPERTIES (owner, balance)
)
EDGE TABLES (
    bank_txns
    KEY (txn_id)
    SOURCE KEY (from_acct_id) REFERENCES bank_accounts (id)
    DESTINATION KEY (to_acct_id) REFERENCES bank_accounts (id)
    LABEL transfer
    PROPERTIES (amount, reference)
);

```

```

SELECT * FROM GRAPH_TABLE (bank_sql_pg MATCH
(a IS account WHERE a.id = 42) -[e IS transfer]-> (b IS account)
COLUMNS (a.id AS acc_a, e.amount AS amount, b.id AS acc_b)
);

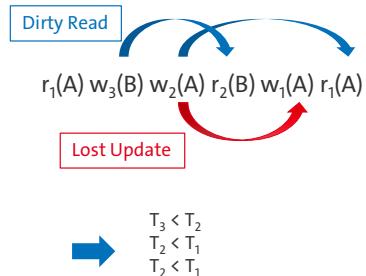
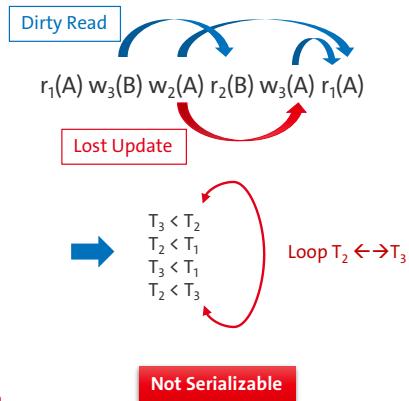
```

ACC_A AMOUNT ACC_B

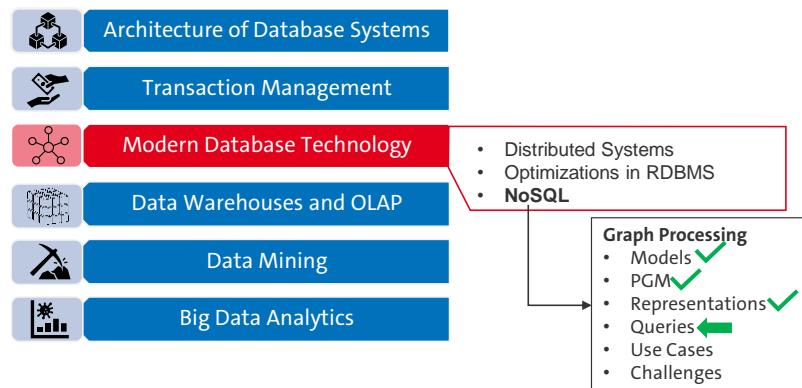
42	15.00	15
42	5.00	15

Which of the following schedules are serializable?

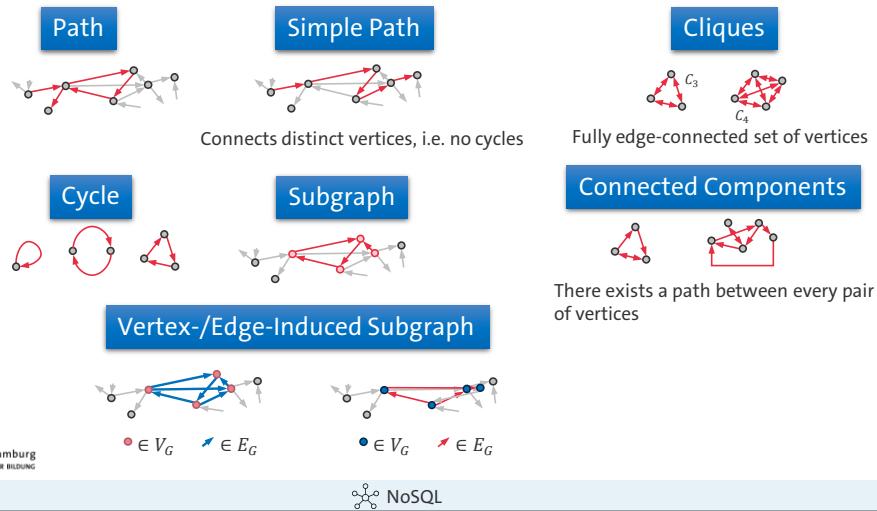
Quiz Recap



Course Outline



Basic Concepts: Paths



Path

- Sequence of edges connecting vertices
- $w_{v_1, v_n} = v_1 e_1 v_2 \dots v_{n-1} e_{n-1} v_n$ with $e_i = (v_i, v_{i+1}) \in E, 1 \leq i < n$
- n is the length of the path

Simple Path

- $p_{v_1, v_n} = v_1 e_1 v_2 \dots v_{n-1} e_{n-1} v_n$
with $\forall v_i, v_j: i \neq j \rightarrow v_i \neq v_j$ and $e_i = (v_i, v_{i+1}) \in E, 1 \leq i < n$

Cycle

- Walk with $v_1 = v_n$ is a cycle
- Graph is acyclic if $\nexists v \in V: w_{v,v} \in G$

Subgraph

- A graph whose vertices and edges are fully part of a larger graph is a subgraph of that larger graph
- $G(V_G, E_G)$ is a subgraph of $H(V_H, E_H)$ if $V_G \subseteq V_H$ and $E_G \subseteq E_H$
- G is a subgraph of H denoted as $G \subseteq H$

Vertex-/Edge-Induced Subgraph

- Subgraph defined by a subset of vertices / edges of a given graphs
- A subgraph of $H(V_H, E_H)$ induced by vertex set $V_G \subseteq V_H$ / $E_G \subseteq E_H$
is a graph $G(V_G, E_G)$ with $E_G = \{(x, y) \in E_H | x, y \in V_G\} / V_G = \{v \in V_H | (v, \cdot) \in E_G \wedge (\cdot, v) \in E_G\}$

Cliques

- Fully edge-connected set of vertices
- $C \subseteq V$ with $\forall v_i, v_j \in C: (v_i, v_j) \in E$

Connected Components

- Fully path-connected set of vertices, i.e. there exists a path between every pair of vertices

- $C \subseteq V$ with $\forall v_i, v_j \in C \exists p \in G: p = v_i, \dots, v_j$
- With considering direction: strongly connected
- Without considering direction: weakly connected
 - Directed edges are treated like undirected ones

Basic Concepts: Measures

Degree (Valency)

Degree of a vertex v



$$\deg(v) = \deg_{out}(v) + \deg_{in}(v)$$

Degree of a graph G

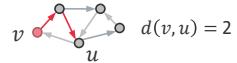
$$\deg(G) = \max_{v \in V} \deg(v)$$

Degree distribution

→ Probability distribution of vertex degree in a graph



Distance



→ Number of edges in a shortest path connecting two vertices

Eccentricity



→ Longest shortest path starting from v



Average Distance

→ Average shortest path
→ Average eccentricity of any vertex in the graph

Radius

→ Minimum eccentricity of any vertex in the graph

$$r(G) = 2$$

Diameter

→ Maximum eccentricity of any vertex in the graph



NoSQL

Degree (Valency)

- In/out degree of a vertex: Number of incoming/outgoing edges of that vertex
 - $\deg_{out}(v) = |\{(v, u) \in E\}|$
 - $\deg_{in}(v) = |\{(u, v) \in E\}|$
 - $\deg(v) = \deg_{out}(v) + \deg_{in}(v)$
- Degree of a graph: Maximum vertex degree
 - $\deg(G) = \Delta(G) = \max_{v \in V} \deg(v)$

Distance

- Distance between two vertices in a graph is number of edges in a shortest path connecting them
- $d(v, u) = \min_{p_{v,u} \in G} |p_{v,u}|$

Eccentricity

- Greatest distance of vertex to any other vertex
- $\epsilon(v) = \max_{u \in V} d(v, u)$

Radius

- $r(G) = \min_{v \in V} \epsilon(v) = \min_{v \in V} (\max_{u \in V} d(v, u))$

Average distance

- $L(G) = \frac{\sum_{v \in V} \epsilon(v)}{|V|}$

Diameter

- $d(G) = \max_{v \in V} \epsilon(v) = \max_{v \in V} (\max_{u \in V} d(v, u))$

Queries: Graph Pattern Matching

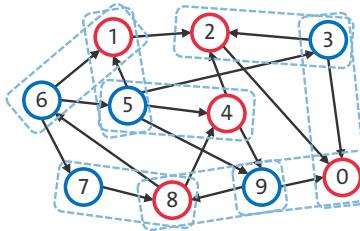
Subgraph pattern p



Graph with place holders x and y

Matching p on G

Finds all subgraphs in G that fit to p



Intuitively, how many matches does p have on G ?

	Number of Results
A	5
B	8
C	10
D	14

?

7

Pattern Syntax in neo4j: Vertex Pattern

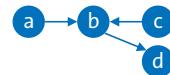
- () unidentified vertex
- (matrix) vertex identified by variable *matrix*
- (:Movie) unidentified vertex with label *Movie*
- (matrix:Movie:Action) vertex with labels *Movie* and *Action* identified by variable *matrix*
- (matrix:Movie {title: "The Matrix"}) + property *title* equal the string “The Matrix”
- (matrix:Movie {title: "The Matrix", released: 1997}) + property *released* equal the integer 1997

 NoSQL

These lists are not exhaustive, refer to the manual for more options.
<http://neo4j.com/docs/developer-manual/current/cypher/syntax/patterns/>

Pattern Syntax in neo4j : Edge Pattern

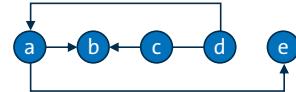
- -- unidentified undirected edge (matches either direction)
- --> unidentified directed edge
- -[role]-> directed edge identified by variable *role*
- -[:ACTED_IN]-> unidentified directed edge with label *ACTED_IN*
- -[role:ACTED_IN]-> directed edge with label *ACTED_IN* identified by variable *role*
- -[role:ACTED_IN {roles: ["Neo"]}]-> + property *roles* contains the string “Neo”
- (a)-->(b)<--(c), (b)-->(d) branches



Pattern Syntax: Path pattern and Graph pattern

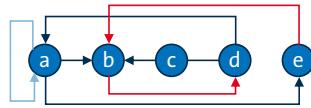
Path patterns

- `(a)-->(b)<--(c)--(d)-->(a)-->(e)`
- `(keanu:Person:Actor {name: "Keanu Reeves"})`
 `-[role:ACTED_IN {roles: ["Neo"]}]> (matrix:Movie {title: "The Matrix"})`



Graph patterns

- Should have at least one shared variable
- `(a)-->(b)<--(c)--(d)-->(a)-->(e), (e)-->(b)-->(d), (a)-->(a)`



NoSQL

Path patterns

- String of alternating vertex pattern and edge pattern
- Starting and ending with a vertex pattern

Graph Patterns

- One or multiple path patterns
- Path patterns should have at least one shared variable
- Without shared variable graph pattern is disconnected
 - Results in a cross-product of the results for connected sub patterns
 - Quadrating blow up in result size and computational complexity

Exercise: Pattern Syntax I

Which of these are valid patterns in Neo4j?

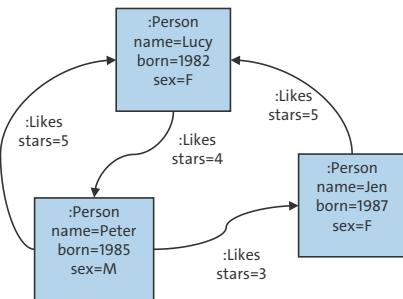
- A: (a,b:Movie)-[:SHOWN_IN]->(e),(f)
- B: (a:Movie)-[:SHOWN_IN]->()
- C: (:Movie)-[:SHOWN_IN]->
- D: ()<--(a:Movie)

Exercise: Pattern Syntax II

Which pattern specifies a loop?

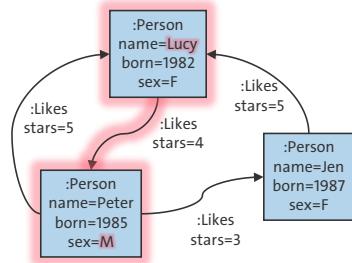
- A: (a:Movie {name: "Matrix"})-->(a)
- B: (a:Movie {name: "Matrix"})-->(b:Movie {name: "Matrix"})
- C: (a:Movie {name: "Matrix"})-->(a:Movie {name: "Matrix"})
- D: (a:Movie {name: "Matrix"})-->({name: "Matrix"})

Query Syntax: Matching



```

MATCH (p:Person)-[:Likes]->(f:Person)
RETURN p.name, f.sex
  
```



Number of Results	
A	2
B	3
C	4
D	5

?

p.name	f.sex
Lucy	M
Peter	F
Jen	F
Peter	F

15

NoSQL

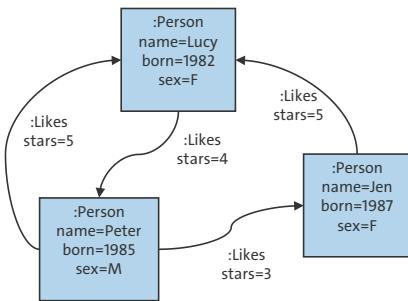
Documentation:

<http://neo4j.com/docs/developer-manual/current/cypher/clauses/match/>
<http://neo4j.com/docs/developer-manual/current/cypher/clauses/return/>

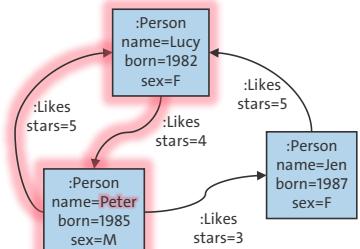
MATCH Clause

- Primary way of getting data from a Neo4j database
- Allows you to specify the patterns
- Named pattern element, e.g. (p:Person), will be bound to the match instance
- Query can have multiple MATCH clauses

Query Syntax: Matching



```
MATCH (p:Person)-[:Likes]->(:Person)-[:Likes]->(fof:Person)
RETURN p.name, fof.name
```



Number of Results

A	2
B	3
C	4
D	5



p.name	fof.name
Lucy	Jen
Peter	Lucy
Peter	Peter
Jen	Peter
Lucy	Lucy

16

Query Syntax: Optional Match

```
MATCH (a:Movie)
OPTIONAL MATCH (a)-[:WROTE]->(x)
RETURN a.title, x.name
```



a.title	x.name
The Matrix	null
The Matrix Reloaded	null
The Matrix Revolutions	null
The Devil's Advocate	
A Few Good Men	Aaron Sorkin
Top Gun	Jim Cash
Jerry Maguire	Cameron Crowe
Stand By Me	null
As Good as It Gets	null
What Dreams May Come	null
Snow Falling on Cedars	null
You've Got Mail	null
Sleepless in Seattle	null
Joe Versus the Volcano	null
When Harry Met Sally	Nora Ephron
That Thing You Do	null

17

NoSQL

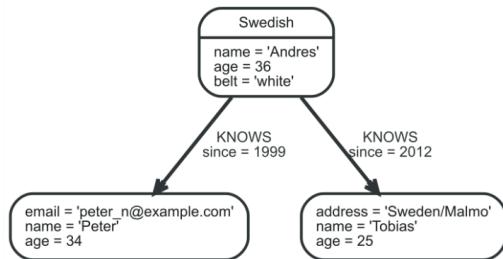
Documentation:

<http://neo4j.com/docs/developer-manual/current/cypher/clauses/optional-match/>

OPTIONAL MATCH clause

- Matches patterns against your graph database, just like MATCH
- Matches the complete pattern or not
- If no matches are found, OPTIONAL MATCH will use NULLs as bindings
- Like relational outer join

Query Syntax: Filtering



```
MATCH (n)
WHERE n.name = 'Peter' XOR (n.age < 30 AND n.name = "Tobias")
OR NOT (n.name = "Tobias" OR n.name="Peter")
RETURN n
```

n
Node[0]{name:"Andres",age:36,belt:"white"}
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

18

NoSQL

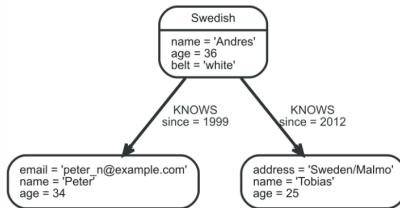
Documentation:

<http://neo4j.com/docs/developer-manual/current/cypher/clauses/where/>

WHERE clause

- After an (OPTIONAL) MATCH, it adds constraints to the (optional) match
- WHERE becomes part of the pattern
- After a WITH, it just filters the result
- Syntax: WHERE <expression>

Query Syntax: Filtering



- Filter on node label

```
MATCH (n) WHERE n:Swedish RETURN n
```

```
n  
Node[0]{name:"Andres",age:36,belt:"white"}
```

- ... on node property

```
MATCH (n) WHERE n.age < 30 RETURN n
```

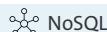
```
n  
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}
```

- ... on relationship type

```
MATCH (n)-[k]-(f) WHERE type(k) = 'KNOWS'
```

```
AND k.since < 2000 RETURN f
```

```
n  
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}
```



More filtering options

On Lists

```
MATCH (n) WHERE a.name IN ["Peter", "Tobias"] RETURN n
```

```
n  
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}  
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}
```

On string properties

```
MATCH (n) WHERE n.name = 'Peter' RETURN n
```

```
- prefixes MATCH (n) WHERE n.name STARTS WITH 'Pet' RETURN n
```

```
- suffixes MATCH (n) WHERE n.name ENDS WITH 'ter' RETURN n
```

```
- infixes MATCH (n) WHERE n.name CONTAINS 'ete' RETURN n
```

```
- regular expressions MATCH (n) WHERE n.name =~ 'P[et]+r?' RETURN n
```

```
n  
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}
```

On property (non-)existence

```
MATCH (n) WHERE exists(n.belt) RETURN n
```

```
false is default for missing values MATCH (n) WHERE NOT n.belt = 'white' RETURN n
```

```
n  
Node[0]{name:"Andres",age:36,belt:"white"}  
MATCH (n) WHERE NOT exists(n.belt) RETURN n  
MATCH (n) WHERE n.belt IS NULL RETURN n
```

```
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}
```

Using patterns for filtering

```
MATCH (tobias { name: 'Tobias' }), (others)
WHERE others.age > 30 AND (tobias) <--(others)
```

```
RETURN others
```

```
n
```

```
Node[0]{name:"Andres",age:36,belt:"white"}
```

Query Syntax: Projection

```
MATCH (n)
RETURN n ["node " + id(n) + " is " +
CASE WHEN n.title IS NOT NULL THEN "a Movie"
      WHEN EXISTS(n.name) THEN "a Person"
      ELSE "something unknown"
END AS about
```

n	about
released: 1999 title: The Matrix tagline: Welcome to the Real World	node 175 is a Movie
born: 1964 name: Keanu Reeves	node 176 is a Person
born: 1967 name: Carrie-Anne Moss	node 177 is a Person
born: 1961 name: Laurence Fishburne	node 178 is a Person

Returned 174 rows in 46 ms.

20

NoSQL

Documentation:

<http://neo4j.com/docs/developer-manual/current/cypher/clauses/return/>
<http://neo4j.com/docs/developer-manual/current/cypher/syntax/expressions/>

RETURN clause

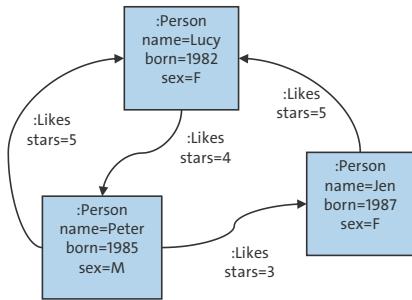
- Defines what to include in the query result set
- Comparable with relational projection
- Only once per query
- Allows to return nodes, edges, properties, or any expressions
- Column can be renamed using AS <new name>

More clauses adapted from SQL

- ORDER BY, e.g. MATCH (n) RETURN n ORDER BY n.name DESC
- LIMIT, e.g. MATCH (n) RETURN n ORDER BY n.name LIMIT 3
- SKIP, e.g. MATCH (n) RETURN n ORDER BY n.name SKIP 3
- Unique results, e.g. MATCH (a { name: "A" })-->(b) RETURN DISTINCT b
- Aggregations, e.g. MATCH (n) RETURN count(*)

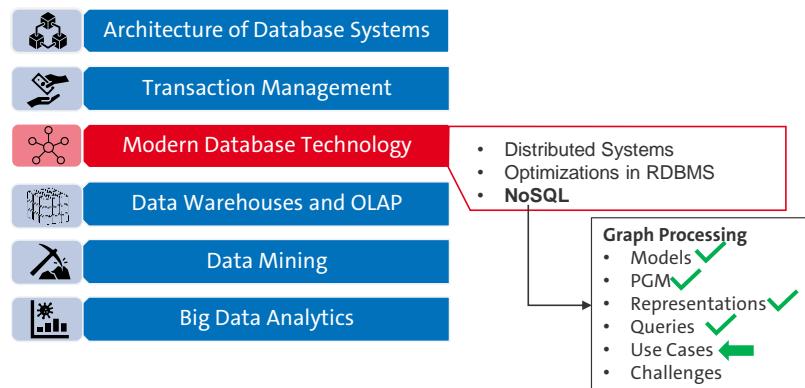
Exercise @home: neo4j Queries

- a) Find all pairs of people who like each other. Return their names.
- b) Return all names in the graph that are not “Peter”. Order them alphabetically.
- c) Return the highest star rating of a “LIKES” relationship in the graph.
- d) Return the name and birth year of all people who like Lucy. The result should have the columns “GivenName” and “year”.



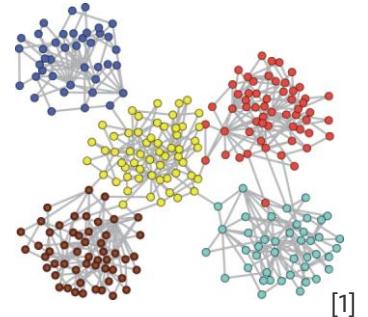
Solution: White on white

Course Outline

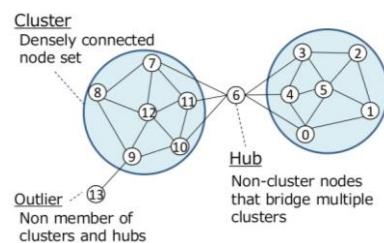


Use Cases: Clustering

- Many different approaches, e.g. min-max cut methods
- Structure-connected clusters → Extends notion of a density-based cluster to graph
- Algorithm examples: SCAN (original), LinkSCAN (uses edge sample), SCAN++ (more efficient)



[1]



[2]

23

NoSQL

Clustering

Division of a graph into a set of subgraphs such that there are a dense set of edges within every cluster and few edges between clusters

Sources:

[1] <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2542420/>

[2] <https://www.slideshare.net/LazyShion/scan-efficient-algorithm-for-finding-clusters-hubs-and-outliers-on-largescale-graphs-vldb-2015>

The SCAN paper: Xu et al. SCAN: A Structural Clustering Algorithm for Networks. KDD. 2007

A very simplified description of SCAN

- Neighborhood of a vertex: Each vertex that is connected via an edge and the vertex itself
- Distance (specific to SCAN and different from our general definition on slide 6): Intersection of the neighborhoods of two vertices
- Normalized distance: Distance normalized by the geometric mean of the size of the two neighborhoods

$$\sigma(v, w) = \frac{|\Gamma(v) \cap \Gamma(w)|}{\sqrt{|\Gamma(v)| \cdot |\Gamma(w)|}}$$

Normalized distance between v and w
Neighborhood intersection between v and w
Sizes of the neighborhoods

- Structural Similarity: If $\sigma(v, w) > \epsilon$, v and w are considered structurally similar
- Core: Vertices with more than μ structural similar neighbors are core vertices

Example

$$\sigma(c, d) = \frac{3}{\sqrt{3 \cdot 4}} = 0.866$$

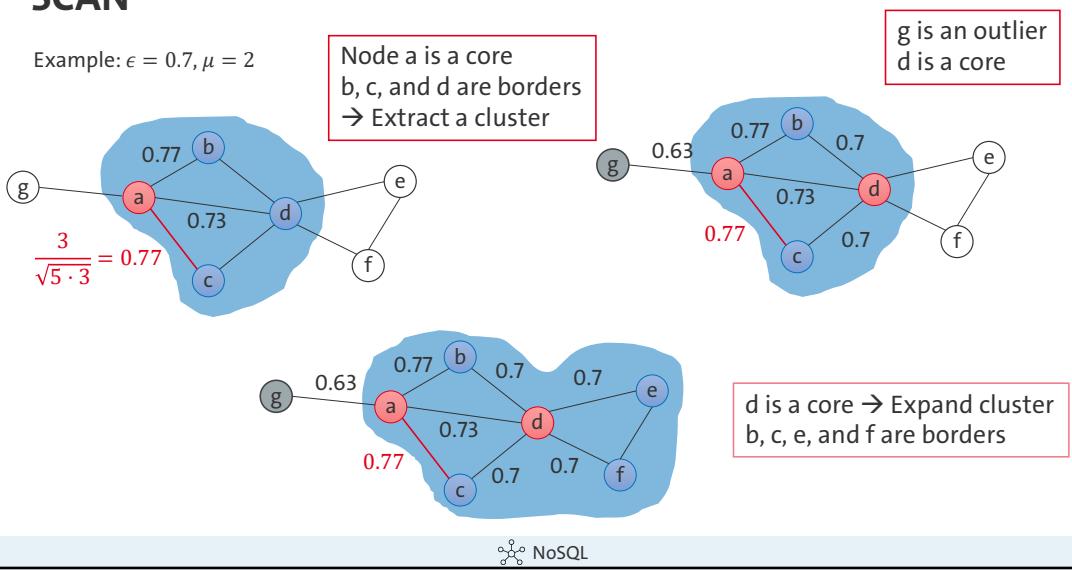
24

NoSQL

- If a core is found, the cluster is expanded
- All structurally similar vertices that are reachable from a core, are initially a border
- If a vertex is not part of a cluster and all its neighbors either belong to one or no cluster, it is an outlier
- If a vertex is not part of a cluster but has neighbors in two or more clusters, it is a hub

SCAN

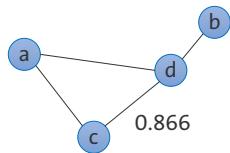
Example: $\epsilon = 0.7, \mu = 2$



25

Exercise Clustering

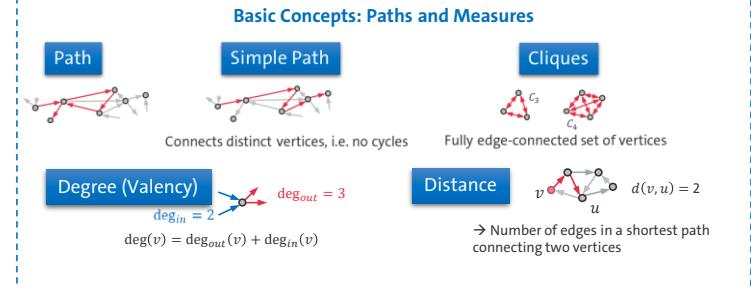
$\epsilon = 0.8, \mu = 2$



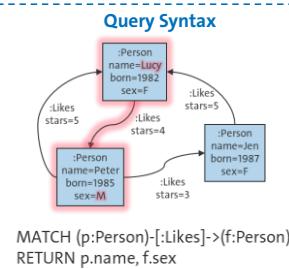
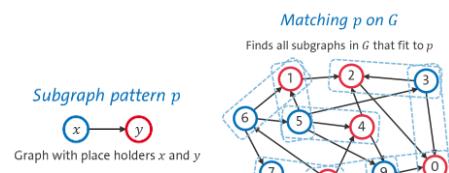
26

Summary

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



Graph Pattern Matching



Clustering

$$\sigma(v, w) = \frac{|\Gamma(v) \cap \Gamma(w)|}{\sqrt{|\Gamma(v)| \cdot |\Gamma(w)|}}$$

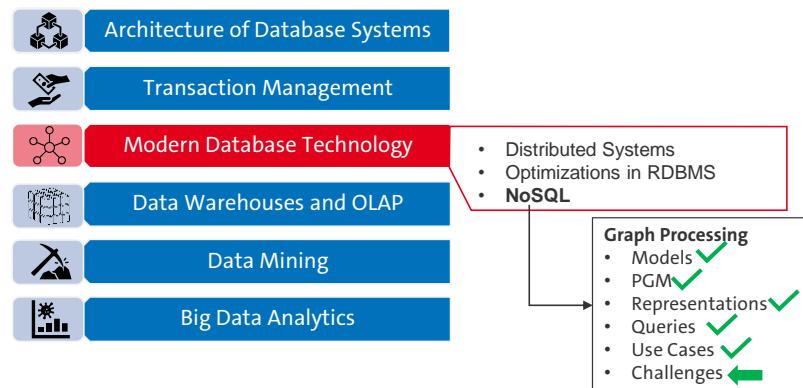
$\frac{3}{\sqrt{5 \cdot 3}} = 0.77$

Research News – This month @ ICDE

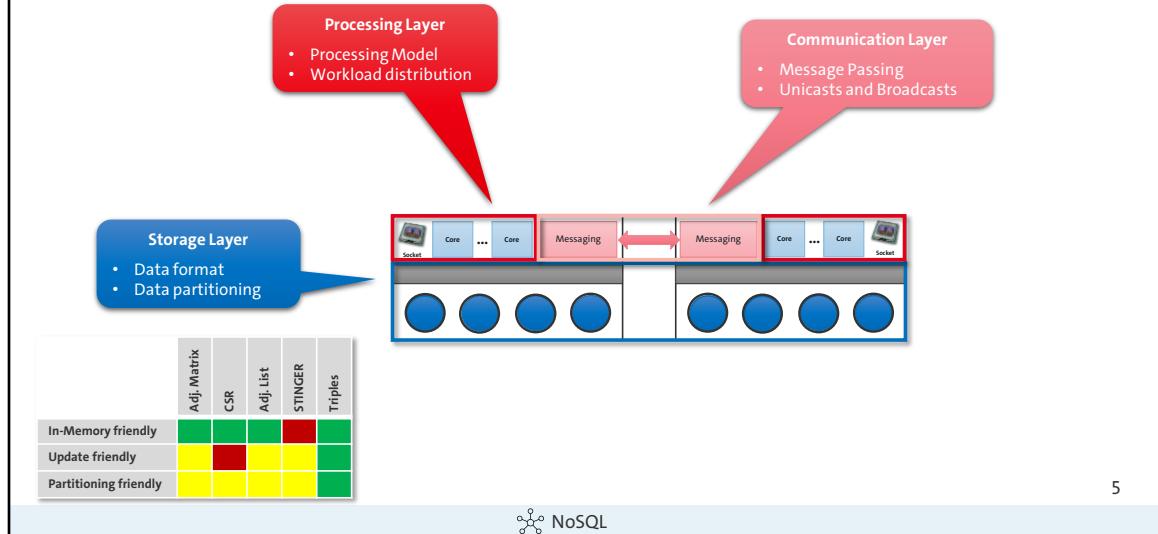
- **Yet another B-Tree paper:** *Improving the Relationship between B+-Tree and Memory Allocator for Persistent Memory*, Wei Yan (Xi'an Jiaotong University); Xingjun Zhang (Xi'an Jiaotong University)
- **...and another one:** *Bwe-tree: an Evolution of Bw-tree on Fast Storage*, Rui Wang (Alibaba Group); xinjun Yang (Alibaba Group); Feifei Li (Alibaba Group); David B Lomet (retired); Xin Liu (Alibaba Inc); panfeng zhou (Alibaba); Yongxiang Chen (Alibaba Group); David Zhang (Alibaba Group); Jingren Zhou (Alibaba Group); Jiesheng Wu (Alibaba)
- **5 papers about LSM trees** (which we will cover when we talk about KV stores)
- **Multiple papers about clustering**, e.g. *McCatch: Scalable Microcluster Detection in Dimensional and Nondimensional Datasets*, Braulio Sánchez (University of São Paulo); Robson Cordeiro (University of São Paulo); Christos Faloutsos (CMU)
- An invited talk about Duckdb's approach to benchmarks with a reference to a paper describing a **PGQ implementation for DuckDB**: *DuckPGQ: Efficient Property Graph Queries in an analytical RDBMS*, Wolde et al., CIDR'23
- Multiple papers on **transaction processing and/or recovery**, e.g. *Fast Parallel Recovery for Transactional Stream Processing on Multicores*, Jianjun Zhao (Huazhong University of Science and Technology); Haikun Liu (Huazhong University of Science and Technology); Shuhao Zhang (Nanyang Technological University); Zhuohui Duan (Huazhong University of Science and Technology); Xiaofei Liao (HUST); Hai Jin (Huazhong University of Science and Technology); Yu Zhang (Huazhong University of Science and Technology)

The Bw tree is controversial within the database community with opinions ranging from “The ‘Influential Paper award’ at ICDE23 was totally justified.” to “‘Influential’ does not necessarily mean that it’s a good influence.”

Course Outline



Architecture Example for Scale-Up Graph Processing Systems



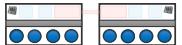
5

- Challenges grow with the system → We will have a look at graph query processing in scale-up systems
- Each layer introduces different challenges:
 - Communication Layer: Scalability (communication must not impact scaling), NUMA friendly (optimizable for concurrency and remote communication)
 - Processing Layer: Flexibility (adapt to workload changes and parallel queries), Parallelism (leverage hardware)
 - Storage Layer: In-Memory friendly, update friendly (easy and fast updates, from a system perspective), partitioning friendly (especially if data model is altered)

Further reading

Another storage format for graphs: Bader, David A., et al. "Stinger: Spatio-temporal interaction networks and graphs (sting) extensible representation." *Georgia Institute of Technology, Tech. Rep* (2009).

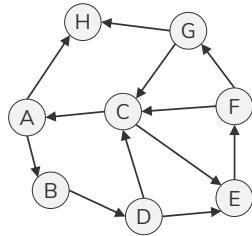
Graph pattern matching on scale-up systems: Krause, Alexander, et al. "Asynchronous graph pattern matching on multiprocessor systems." *New Trends in Databases and Information Systems: ADBIS 2017 Short Papers and Workshops, AMSD, BigNovelTI, DAS, SW4CH, DC, Nicosia, Cyprus, September 24–27, 2017, Proceedings 21*. Springer International Publishing, 2017.



Graph Partitioning Classification

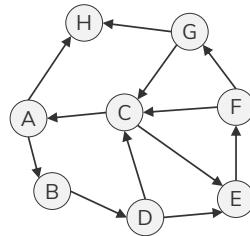
Edges

- Partition based solely on edges
- Algorithm
 - RoundRobin



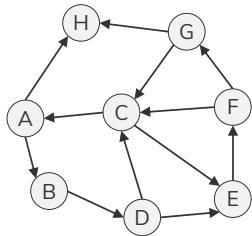
Vertices

- Partition by Vertices, i.e. distribute a vertex and its adjacency to a partition
- Algorithms
 - V/V: RoundRobinVertices
 - V/E: BalancedEdges
 - V/E: DistributedSkew



Components

- Partition by components, distribute a vertex and its adjacency to a partition
- Algorithm
 - C/V: k-Way



Alexander Krause, et al.: Partitioning Strategy Selection for In-Memory Graph Pattern Matching on Multiprocessor Systems. Euro-Par 2017: 149-163

6

- There is no single best algorithm
- The most suited algorithm depends on the workload and the data

Further Reading

George Karypis and Vipin Kumar: A Coarse-Grain Parallel Formulation of Multilevel k-way Graph Partitioning Algorithm, PPSC. 1997

Alexander Krause, et al.: Partitioning Strategy Selection for In-Memory Graph Pattern Matching on Multiprocessor Systems. Euro-Par 2017: 149-163

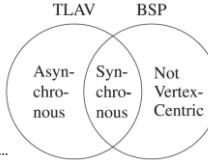
Graph Processing



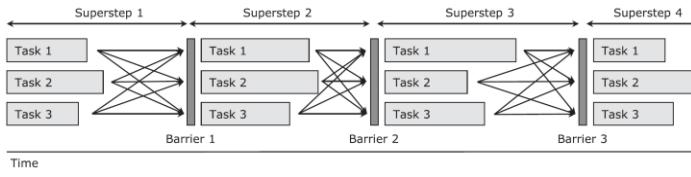
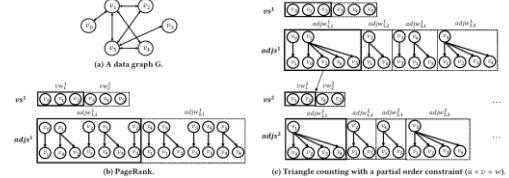
Bulk Synchronous Processing /

Think Like A Vertex

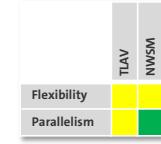
- Multistep Process
- MPI for communication
- Local computation
- Exchange intermediates
- Giraph, GraphX, Pregel, Powergraph, ...



Nested Windowed Streaming Model



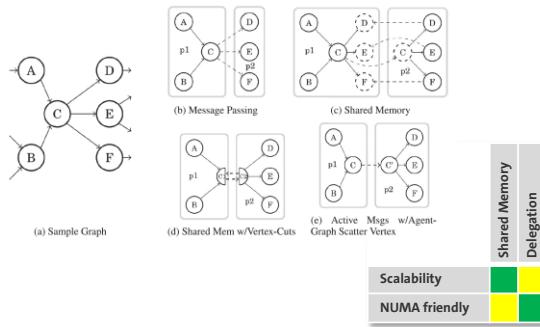
Robert Ryan McCune, et al.: Thinking Like a Vertex: A Survey of Vertex-Centric Frameworks for Large-Scale Distributed Graph Processing. ACM Comput. Surv. 48(2): 25:1-25:39 (2015).
Seongyun Ko, et al.: Turbograph++: A Scalable and Fast Graph Analytics System. SIGMOD, 2018



Communication Models

Selected Available Protocols

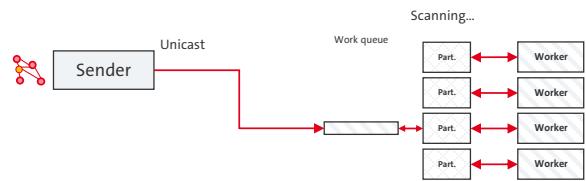
- Shared Memory
- Message Passing / Delegation



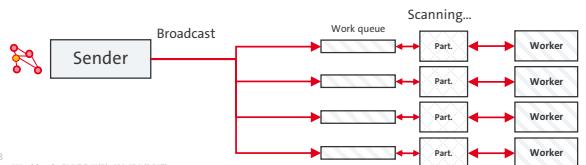
Irina Calciu, et al.: Message Passing or Shared Memory: Evaluating the Delegation Abstraction for Multicore, OPODIS, 2013
 Raja Appuswamy, et al.: Analyzing the Impact of System Architecture on the Scalability of OLTP Engines for High-Contention Workloads, PVLDB 11(2): 121-134 (2017)
 Robert Ryan McCune, et al.: Thinking Like a Vertex: A Survey of Vertex-Centric Frameworks for Large-Scale Distributed Graph Processing, ACM Comput. Surv. 48(2): 25:1-25:39 (2015)

Messaging

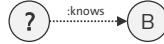
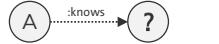
- Unicasts for direct lookups



- Broadcasts for low selectivity or missing locality information



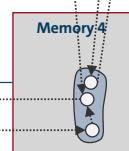
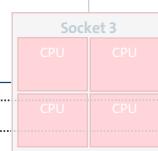
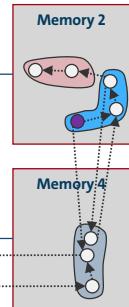
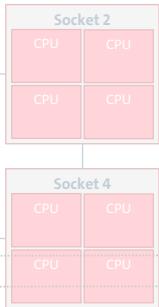
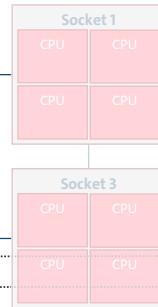
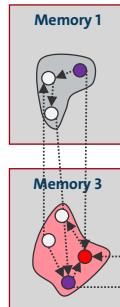
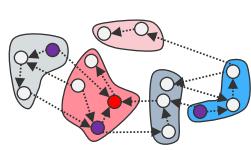
Messaging for Graph Pattern Matching



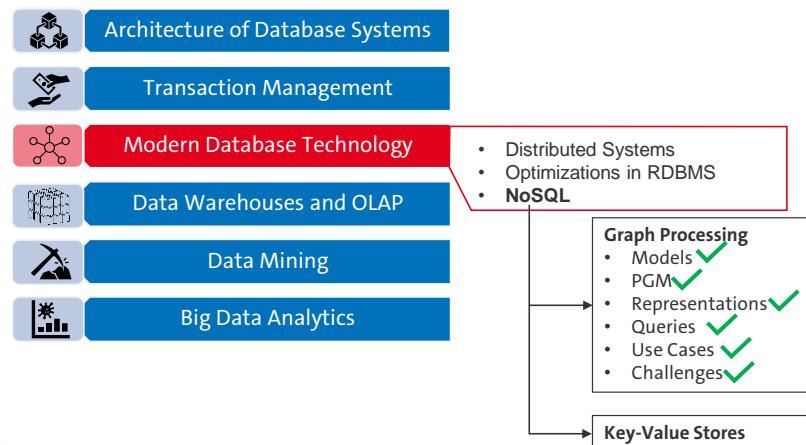
RoutingTable	
$V_{1_out} \rightarrow P_1$	$V_{37_out} \rightarrow P_3$
$V_{2_out} \rightarrow P_3$	$V_{38_out} \rightarrow P_{19}$
$V_{3_out} \rightarrow P_{27}$	$V_{39_out} \rightarrow P_1$
...	...

RoutingTable	
$V_{1_in} \rightarrow P_{14}$	$V_{37_in} \rightarrow P_{14}$
$V_{2_in} \rightarrow P_6$	$V_{38_in} \rightarrow P_{33}$
$V_{3_in} \rightarrow P_3$	$V_{39_in} \rightarrow P_1$
...	...

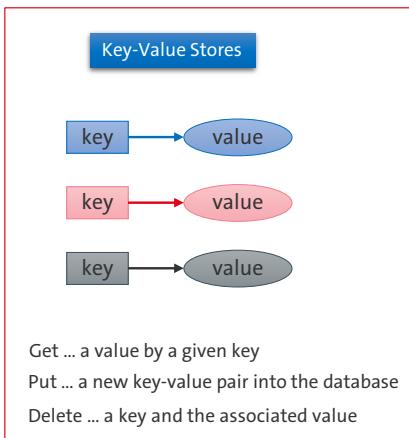
Bloom Filter	
$P_{14} \rightarrow V_{1_in}, V_{37_in}$	
$P_6 \rightarrow V_{2_in}, V_{39_in}$	
$P_{33} \rightarrow V_{3_in}, V_{38_in}$	
...	...



Course Outline



NoSQL Databases: Key-Value Stores



Value is a (semi) structured document

Document Stores

Document 1	Document 2	Document 3
{"id": "1", "name": "John Smith", "isActive": true, "dob": "1964-08-08"}	{"id": "2", "name": "Sarah Jones", "isActive": false, "dob": "2002-02-18"}	{"id": "3", "name": "Adam Stark", "first": "Adam", "last": "Stark", "isActive": true, "dob": "2015-04-19"}

Value is a row or a table

Wide Column Stores

super column family
 companies
 address website
 city subdomain www
 state domain gis.com
 street protocol http

row key 1
 column family column

Key-Value Stores

Basic key-value mapping with a simple API for **CRUD** operations

Create Read Update Delete

Horizontal partitioning

users:1:a	4711
users:1:b	“[12, 34, 45, 67, 89]”
users:2:a	01101010010110010101001...
users:2:b	“[12, ABC, 3212, 0xff]”

CRUD realized by at least 3 types of queries

Put: Add a new pair

Get: Retrieve a pair

Delete

Additional Operators implemented by some systems

Merge

MultiGet/MGet

MSet

...

Motivation

- Basic key-value mapping via simple API (more complex data models can be mapped to key-value representations)
- Reliability at massive scale on commodity HW (cloud computing)

System Architecture

- Database is a collection of key/value pairs
- Key-value maps, where values can be of a variety of data types
- The key for each pair is unique
- APIs for CRUD operations (create, read, update, delete)
- Scalability via sharding (horizontal partitioning)

Example Systems

- Redis (2009, CP/AP) → not a plain kv-store, but “data structure server” with persistent log
- LevelDB
- RocksDB
- Memcached (started as a simple caching tool)

Futher Reading

Giuseppe DeCandia et al: Dynamo: amazon's highly available key-value store. SOSP 2007

Examples

Memcached via telnet

```
Add a new KV-pair  
set AgeAlice 0 120 1 [Press Enter]  
26 [Press Enter]  
Retrieve a KV-pair  
get AgeAlice  
Delete a KV-pair  
delete AgeAlice
```

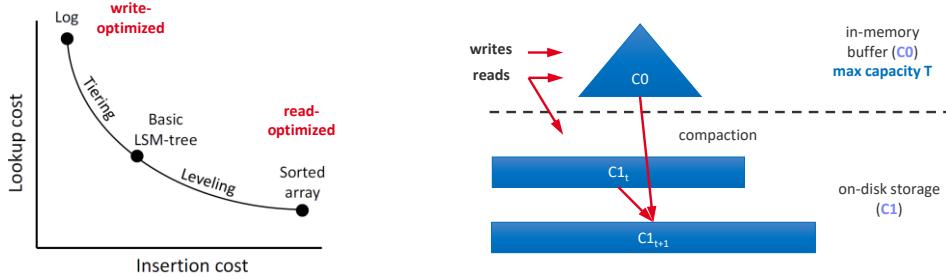
RocksDB with C++

```
Add a new KV-pair  
database->Put(WriteOptions(), AgeAlice, "26");  
Retrieve a KV-pair  
std::string content;  
database->Get(ReadOptions(), AgeAlice, &content);  
Delete a KV-pair  
database->Delete(WriteOptions(), AgeAlice)
```

Syntax of Set in Memcached via telnet

```
set KEY META_DATA EXPIRATION_TIME_IN_S LENGTH_IN_BYTES  
[Press Enter]  
VALUE  
[Press Enter]
```

Log-Structured Merge Trees (LSM Trees)



LSM Overview

- Many KV-stores rely on LSM-trees as their storage engine (e.g., BigTable, DynamoDB, LevelDB, Riak, RocksDB, Cassandra, HBase)
- Approach: Buffers writes in memory, flushes data as sorted runs to storage, merges runs into larger runs of next level (compaction)

System Architecture

- Writes in C_0
- Reads against C_0 and C_{1_t} (w/ buffer for C_{1_t})
- Compaction (rolling merge): sort, merge, including deduplication

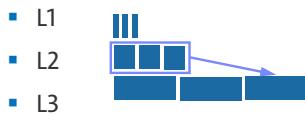
Further Reading

Patrick E. O'Neil, Edward Cheng, Dieter Gawlick, Elizabeth J. O'Neil: The Log-Structured Merge-Tree (LSM-Tree). Acta Inf. 1996

LSM Trees: Merging

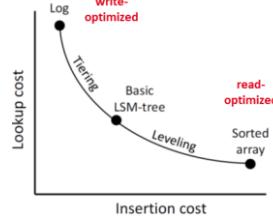
LSM Tiering

- Keep up to T-1 (sorted) runs per level L
- Merge all runs of L_i into 1 run of L_{i+1}



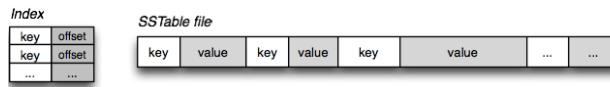
LSM Leveling

- Keep 1 (sorted) run per level L
 - Sort-Merge run of L_i with L_{i+1}
- L1
 - L2
 - L3
-

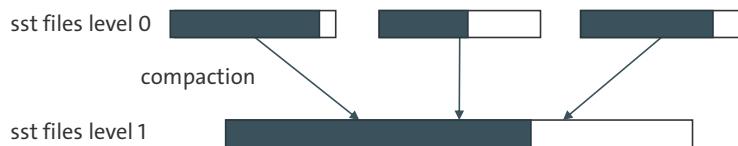


Storing key-value pairs: Sorted String Tables (sst)

All keys are sorted within an sst file
→ New sst file if new key doesn't keep order



*image: <https://www.igvita.com/2012/02/06/sstable-and-log-structured-storage-leveldb/>



→ Up to 7 levels (originally developed for LevelDB)

Sorted String Tables

- Popular format for key-value stores
- Stores data in a compact format
- Basically implements an lsm tree
- For large SSTables, index can be kept separately in-memory

SSTables example

K,V-Pairs are inserted in the following order:

(2,'an')
(3,'example.')
(0,'This')
(1,'is')

Assume each key is stored as a character and each character is stored using 1 byte and memory is byte-addressable

Index 1

2	0
3	3

SSTable 1

2	an	3	an example.
---	----	---	-------------

Index 2

0	0
1	5

SSTable 2

0	This	1	is
---	------	---	----

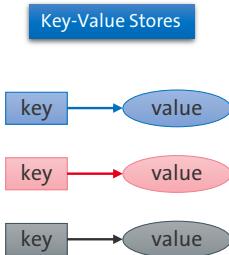
In practice, the length of a key is no necessarily constant

→ Delimiters or a fixed maximum key size are used to separate the key from the offset

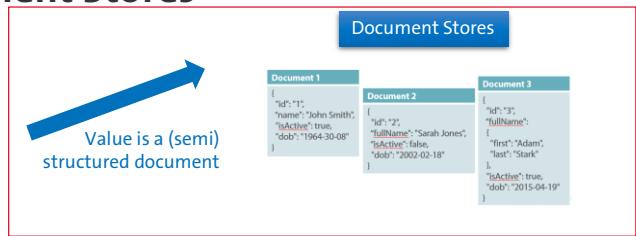
Exercise: SSTables

- Insert the following (key,value)-pairs into an empty sst:
(5,'!'), (0,'H'), (1,'ey', h'),(3,'l'),(4,'o'), (2,'el')
- Assume each character takes up 1 Byte, each key is a character, and memory is byte-addressable
 - How many SSTables are created?
 - What do they look like?
 - What does the index look like?

NoSQL Databases: Document Stores

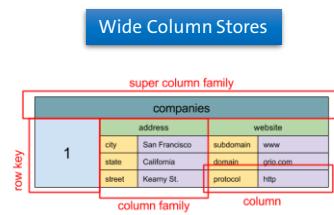


Get ... a value by a given key
 Put ... a new key-value pair into the database
 Delete ... a key and the associated value



Value is a row or a table

Document Stores



20

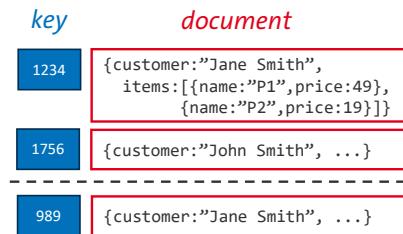
NoSQL

Basic Data Model

- The general notion of a document – words, phrases, sentences, paragraphs, sections, subsections, footnotes, etc.
- Flexible schema: subcomponent structure may be nested, and vary from document-to-document
- Essentially, they support the embedding of documents and arrays within other documents and arrays
- Document structures do not have to be predefined, so are schema-free (XML, mostly JSON)

Document Stores

- Application-oriented management of structured, semi-structured, and unstructured information
 - Collections of (key, document)-pairs



Motivation

- Application-oriented management of structured, semi-structured, and unstructured information
 - Scalability via parallelization on commodity HW (cloud computing)

System Architecture

- Collections of (key, document)
 - Scalability via sharding (horizontal partitioning)
 - Custom SQL-like or functional query languages

Example Systems

- MongoDB (C++, 2007, CP) → RethinkDB, Espresso, Amazon DocumentDB (Jan 2019)
 - CouchDB (Erlang, 2005, AP) → CouchBase

Recap: JSON (JavaScript Object Notation)

JSON Data Model

- Data exchange format for semi-structured data
- Not as verbose as XML (especially for arrays)
- Popular format

```
{"students": [  
    {"id": 1, "courses": [  
        {"id": "INF.01017UF", "name": "DM"},  
        {"id": "706.550", "name": "AMLS"}]},  
    {"id": 5, "courses": [  
        {"id": "706.520", "name": "DIA"}]},  
]
```

Query Languages

- Most common: libraries for tree traversal and data extraction
- JSONiq: XQuery-like query language
- JSONPath: XPath-like query language

```
JSONiq Example:  
declare option jsoniq-version "...";  
for $x in collection("students")  
  where $x.id lt 10  
  let $c := count($x.courses)  
  return {"sid":$x.id, "count":$c}
```

[<http://www.jsoniq.org/docs/JSONiq/html-single/index.html>]

Example MongoDB

[Credit: <https://api.mongodb.com/python/current>]

Creating a Collection

```
import pymongo as m
conn = m.MongoClient("mongodb://localhost:123/")
db = conn["dbs19"]      # database dbs19
cust = db["customers"] # collection customers
```

Inserting into a Collection

```
mdict = {
    "name": "Jane Smith",
    "address": "Innfeldgasse 13, Graz"
}
id = cust.insert_one(mdict).inserted_id
# ids = cust.insert_many(mlist).inserted_ids
```

Querying a Collection

```
print(cust.find_one({"_id": id}))

ret = cust.find({"name": "Jane Smith"})
for x in ret:
    print(x)
```

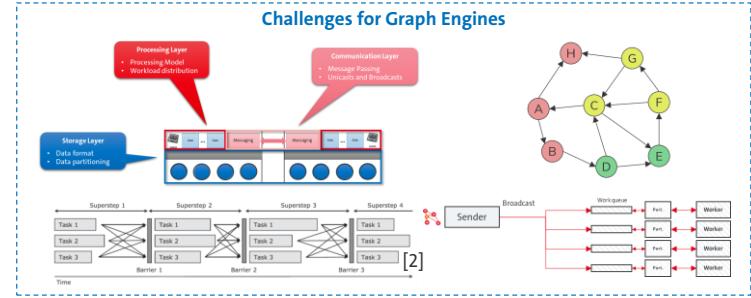


Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

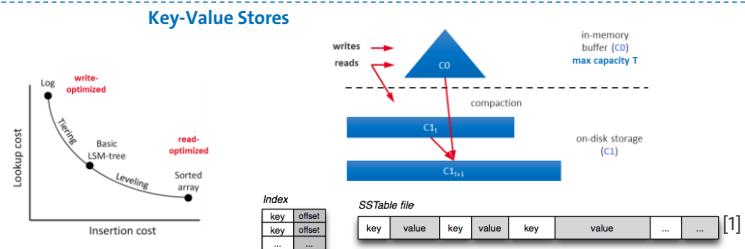
NoSQL

Summary



Memcached via telnet

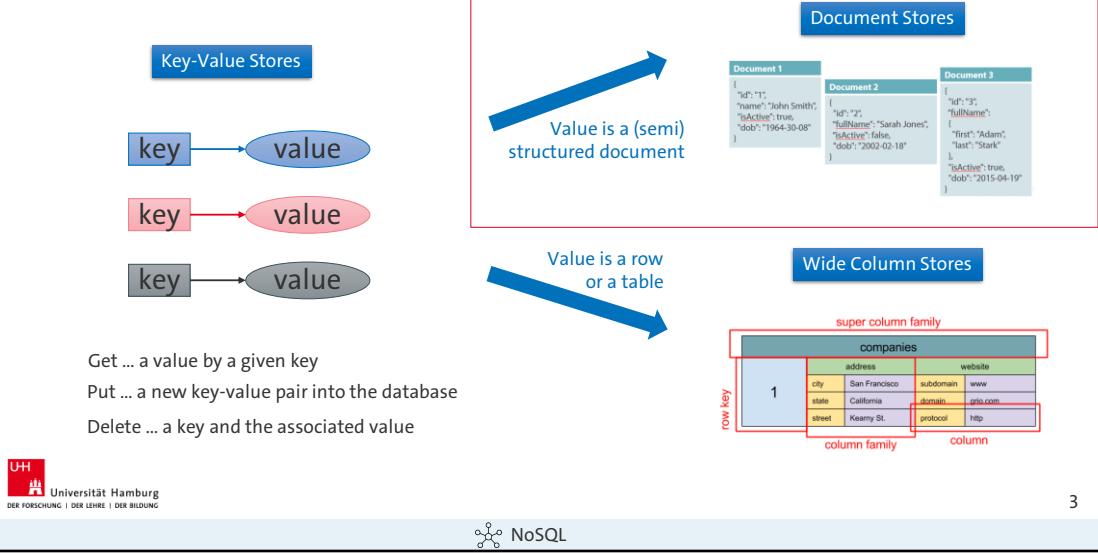
```
Add a new KV-pair
set AgeAlice 0 120 1 [Press Enter]
26 [Press Enter]
Retrieve a KV-pair
get AgeAlice
Delete a KV-pair
delete AgeAlice
```



[1] Image: <https://www.igvita.com/2012/02/06/sstable-and-log-structured-storage-leveledb/>

[2] Robert Ryan McCune, et al.: Thinking Like a Vertex: A Survey of Vertex-Centric Frameworks for Large-Scale Distributed Graph Processing. ACM Comput. Surv. 48(2): 25:1-25:39 (2015),

NoSQL Databases: Document Stores

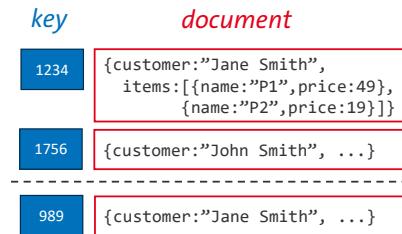


Basic Data Model

- The general notion of a document – words, phrases, sentences, paragraphs, sections, subsections, footnotes, etc.
- Flexible schema: subcomponent structure may be nested, and vary from document-to-document
- Essentially, they support the embedding of documents and arrays within other documents and arrays
- Document structures do not have to be predefined, so are schema-free (XML, mostly JSON)

Document Stores

- Application-oriented management of structured, semi-structured, and unstructured information
- Collections of (key, document)-pairs



Motivation

- Application-oriented management of structured, semi-structured, and unstructured information
- Scalability via parallelization on commodity HW (cloud computing)

System Architecture

- Collections of (key, document)
- Scalability via sharding (horizontal partitioning)
- Custom SQL-like or functional query languages

Example Systems

- MongoDB (C++, 2007, CP) → RethinkDB, Espresso, Amazon DocumentDB (Jan 2019)
- CouchDB (Erlang, 2005, AP) → CouchBase

Recap: JSON (JavaScript Object Notation)

JSON Data Model

- Data exchange format for semi-structured data
- Not as verbose as XML (especially for arrays)
- Popular format

```
{"students": [  
    {"id": 1, "courses": [  
        {"id": "INF.01017UF", "name": "DM"},  
        {"id": "706.550", "name": "AMLS"}]},  
    {"id": 5, "courses": [  
        {"id": "706.520", "name": "DIA"}]},  
]
```

Query Languages

- Most common: libraries for tree traversal and data extraction
- JSONiq: XQuery-like query language
- JSONPath: XPath-like query language

```
JSONiq Example:  
declare option jsoniq-version "...";  
for $x in collection("students")  
where $x.id lt 10  
let $c := count($x.courses)  
return {"sid":$x.id, "count":$c}
```

[<http://www.jsoniq.org/docs/JSONiq/html-single/index.html>]

Example MongoDB

[Credit: <https://api.mongodb.com/python/current>]

Creating a Collection

```
import pymongo as m
conn = m.MongoClient("mongodb://localhost:123/")
db = conn["dbs19"]      # database dbs19
cust = db["customers"] # collection customers
```

Inserting into a Collection

```
mdict = {
    "name": "Jane Smith",
    "address": "Inffeldgasse 13, Graz"
}
id = cust.insert_one(mdict).inserted_id
# ids = cust.insert_many(mlist).inserted_ids
```

Querying a Collection

```
print(cust.find_one({"_id": id}))

ret = cust.find({"name": "Jane Smith"})
for x in ret:
    print(x)
```

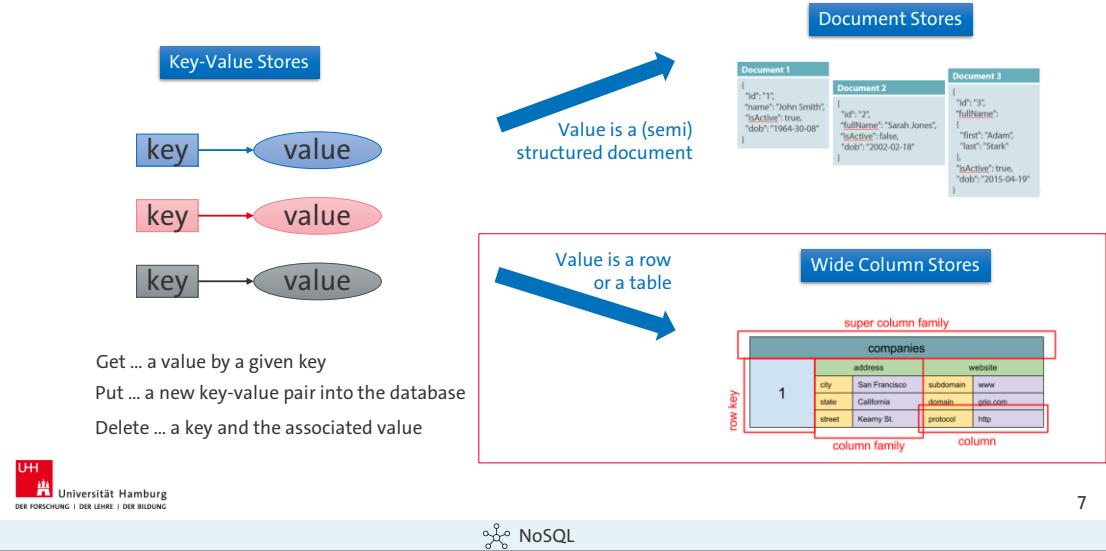


Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

NoSQL

NoSQL Databases: Wide Column Stores



- Wide Column Stores are sometimes called “Extensible Record Stores”
- Column Store != Wide Column Store

Basic Data Model

- Database is a collection of key/value pairs
- Key consists of 3 parts: a row key, a column key, and a time-stamp (i.e., the version)
- Flexible schema: the set of columns is not fixed, and may differ from row-to-row

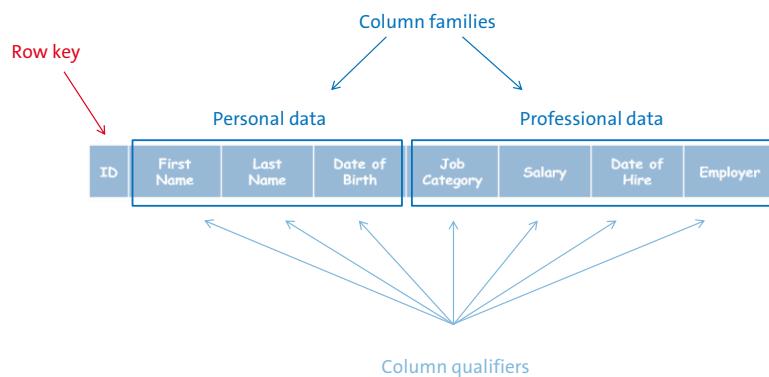
Example Systems

- Google Bigtable
- HBase

Further Reading

Chang, Fay, et al. "Bigtable: A distributed storage system for structured data." *ACM Transactions on Computer Systems (TOCS)* 26.2 (2008): 1-26.

Example: Wide Column Data Model



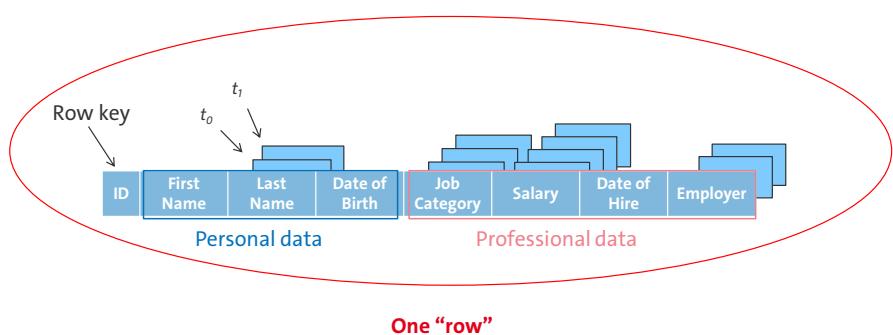
Example: Wide Column Data Model

Personal data				Professional data			
ID	First Name	Last Name	Date of Birth	Job Category	Salary	Date of Hire	Employer
ID	First Name	Middle Name	Last Name	Job Category	Employer	Hourly Rate	
ID	First Name	Last Name	Job Category	Salary	Employer	Group	Seniority
ID	Last Name	Job Category	Salary	Date of Hire	Employer	Insurance ID	Emergency Contact

Medical data

single “table”

Example: Wide Column Data Model



One “row”

One “row” in a wide-column NoSQL database table

=

Many rows in several relations/tables in a relational database

10

Overview NoSQL Systems

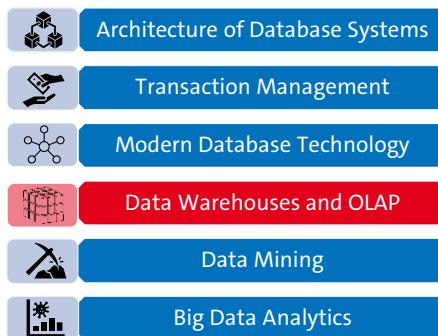


11

Further Reading

Ronald Barber et al: Evolving Databases for New-Gen Big Data Applications.
CIDR 2017

Course Outline



Motivation for a Data Warehouse

Preparation

- Multitude of different data sources
- Data cleansing / cleaning → historic reason for DWH

Adaptation/Standardization/Interpretability

- Central defined key performance indicators (KPI Management) and dimensions
- One single truth, no different interpretations

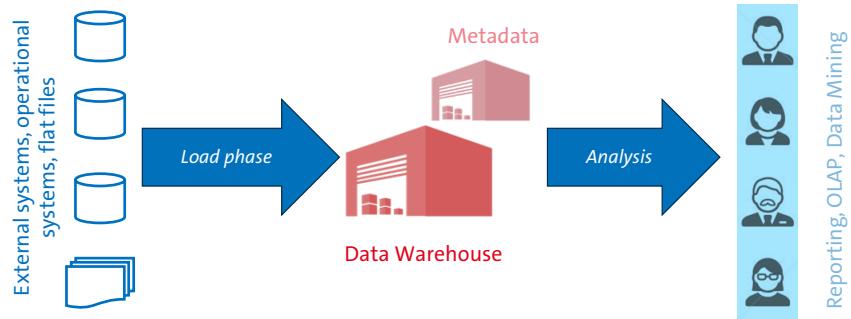
Allows new business processes

- Statistical methods (e.g. correlation analysis)
- Examples: customer segmentation, -evaluation

More

- Organization: data from multiple business units
- Technical: different query models, usage patterns, data volumes
- Consolidation: heterogeneity (schema, data quality, “garbage in, garbage out!”)

Data-Warehouse concept



Goal: (analytic) access to consistent data

- Integration of external/operational data sources in a centralized database
- Extraction and multi-stage loading process
- Integrated (and historicized) data form a base for analytic tools

Descriptive Statistics

Processing step	Actions	Data
<i>Data recording</i>	Creation of raw data calibration / checking optional: anonymization	Raw data „Microcosmos“
<i>Data preparation</i>	Structural adaptation Validation (filtering, quality checking) Statistical corrections (imputation, estimation, „missing values“) Optional: anonymization, pre-aggregation	processed raw data
<i>Data analysis</i>	Application-specific aggregation Representation / interpretation	Enriched data/aggregates, „Macrocosmos“

Explorative statistics

- Find “undiscovered” structures and connections to generate new hypotheses
- Based on samples

Mathematical statistics

- Also called “statistical inference” or “inductive statistics”, in German also “schließende Statistik”
- Based on samples
- Uses stochastic models → provides probability of error (in contrast to descriptive statistics)

DWH Definition

A data warehouse is a central database which is optimized for analysis purposes and contains data of several, usually heterogeneous data sources.

Erhard Rahm

Physical database as an integrated view on (arbitrary) data with a focus on the evaluation aspect, where data is often but not necessarily historized.”

GI

Some general approaches

“Data warehouse is a subject-oriented, integrated, time-varying, non-volatile collection of data in support of the management's decision-making process.”
(Bill Inmon)

“Data Warehouse is an environment, not a product.” (Berson/ Smith)

Characteristics of DWHs

Analysis-oriented organization of data

Domain-oriented, models a specific application goal

Integration of data from different source systems

Integrated database, integration on structural and data level of multiple databases

No user updates

(almost) no updates and deletes (technical updates / deletes only, quality assurance)

Historic data

Data is kept over a long period of time

- Operations in operational environment: Insert, Delete, Update, Select
- Operations in a data warehouse: Insert the initial and additional loading of data by (batch) processes, Select the access of data
- Non-volatile/stable database, loaded data is not deleted or modified, read-only access

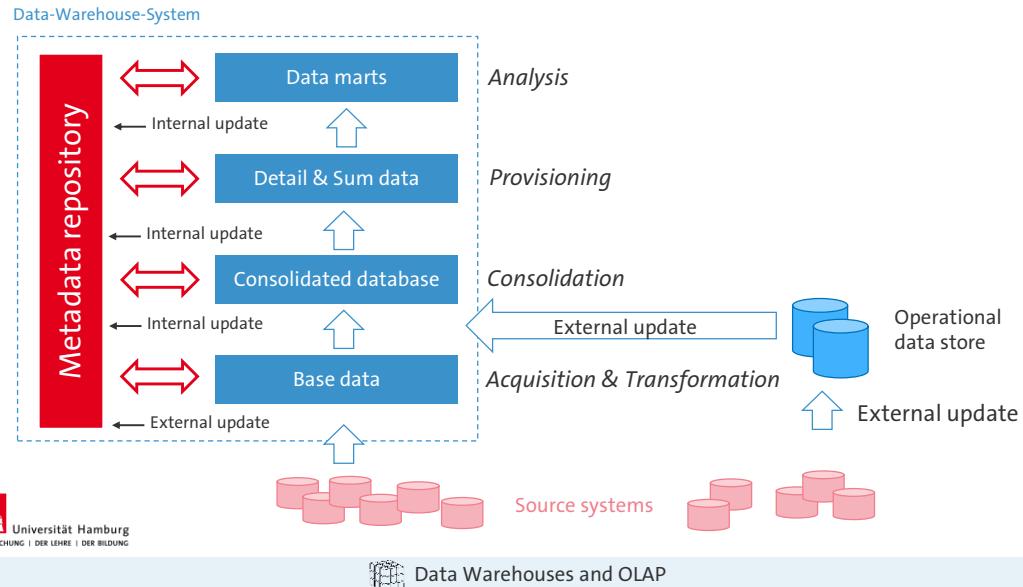
Non-volatile Data

- What happens in the OLTP system if the customer cancels his booking?
 - Delete operation in OLTP
 - Seat gets available again and can be sold to another passenger
- What happens in the DWH?
 - Insert operation in DWH with, e.g., a flag indicating that the customer cancelled/deleted his booking
 - Business can make analysis about cancelled booking: why might the customer have cancelled? How to prevent the customer or other customers to cancel next time?

Tales from the reality of data science

- “Compulsive data hoarders” exist in many (but not all) projects, so data is collected before it is defined what this data will be used for or if it will be used at all → Exponential data growth
- Especially in natural science, it’s not always clear which parts of the data might be useful at a later point in time → Everything is collected → Data growth is only restricted by the ratio of failed experiments/failed hardware

Reference architecture for DWH



Further Reading

Wolfgang Lehner: „Datenbanktechnologie für Data-Warehouse-Systeme: Konzepte und Methoden“, dpunkt-Verlag, 2003

Metadata examples

- Description of the Data Warehouse System
- Names, definitions, structure, and content of a DWH
- Identification of data sources
- Integration and transformation rules for filling the DWH
- Integration and transformation rules for end-user rules
- Operational information like updates, versions
- Usage and performance of the Data Warehouse (Monitoring)
- Security, access rights

Example DWH architecture

Data analytics



Data provisioning



Data consolidation



Data transformation



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

ZID	MID	Salary	ZID	MID	Salary
1	1	3.300€	1	2	6.176€
2	1	3.500€	2	2	5.176€

ZID	Time	ZID	MID	Salary	MID	Name
1	T1	1	1	3.300€	1	Max Mueller
2	T2	1	2	6.176€	2	Tim Mueller

		2	1	3.500€	
		2	2	5.176€	

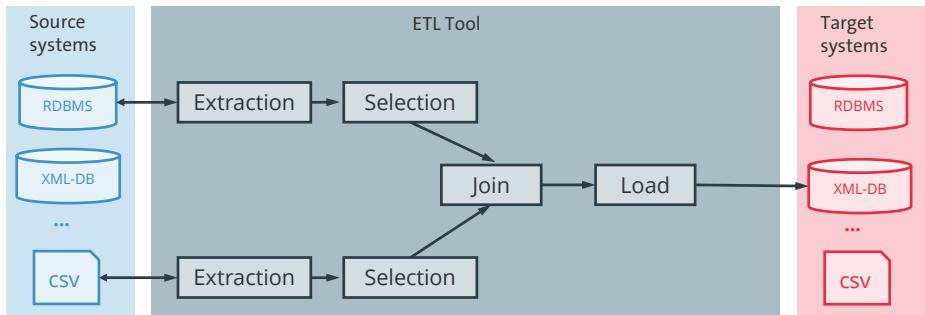
Zeit	Name	Salary
T1	Max Mueller	3.300€
T1	Tim Mueller	6.176€
T2	Max Mueller	3.500€
T2	Tim Mueller	5.176€

Name		Salary	
Name	Gehalt	Name	Salary
Max Müller	3.500€	Tim Mueller	7,400\$



Data Warehouses and OLAP

Data acquisition and transformation



Goal

- Provisioning of data for the consolidated database
- Efficiency vs. Actuality

What are data sources?

- Heterogeneous systems or files
- Local schemata and semantics

Describing attributes of data sources

- Utilization of data for the Data Warehouse
- Origin (internal or external data)
- Cooperation (active sources, snapshot-sources,)
- Availability of source data (legal, social, organizational, technical)
- Cost of acquisition
- Quality (consistency, correctness, completeness, accuracy,...)

Staging Area

- “Landing Zone” for data coming into a DWH
- Temporary memory for integrating extracted data after an external update

Schema integration

- Overcoming semantic/structural heterogeneity
- Integration of different local schemata /data models into one global schema
- Decoupling of transformation from the source systems and the consolidated database

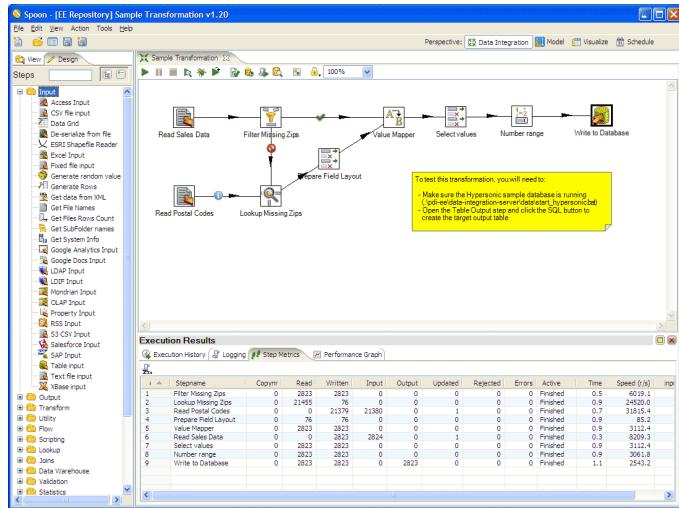
Data integration

- Adaptation of data formats etc.
- Correction of different spellings (abbreviations, etc.)

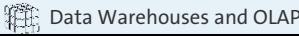
ETL-components

- Extraction-, Transformation- and Load component

Example ETL Tools



<https://www.hitachivantara.com/en-us/products/data-management-analytics/pentaho/download-pentaho.html>



Commercial Systems

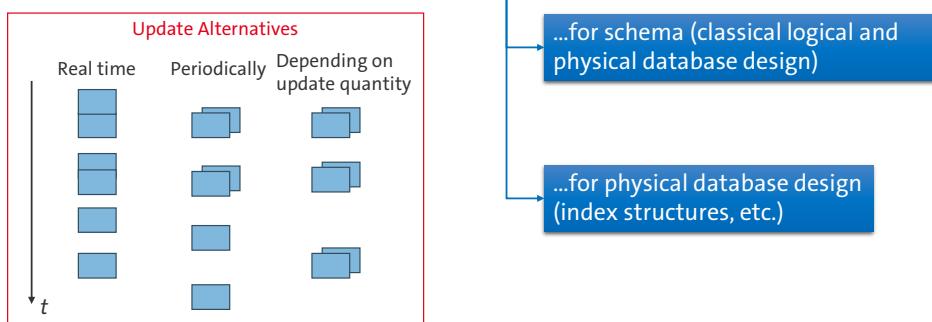
- Informatica PowerCenter
- Cognos Decision-Stream
- Oracle DW Builder
- IBM InfoSphere DataStage
- IBM DB2 Warehouse Enterprise Edition
- AB Initio

Open Source

- Pentaho Data Integration (a.k.a Kettle)
- Talend ETL Integration Suite
- Clover ETL Data Integration
- JasperETL Open Source

Data Consolidation

- Integrated database from cleaned data
- Not specifically modeled, **no specific optimizations**



21



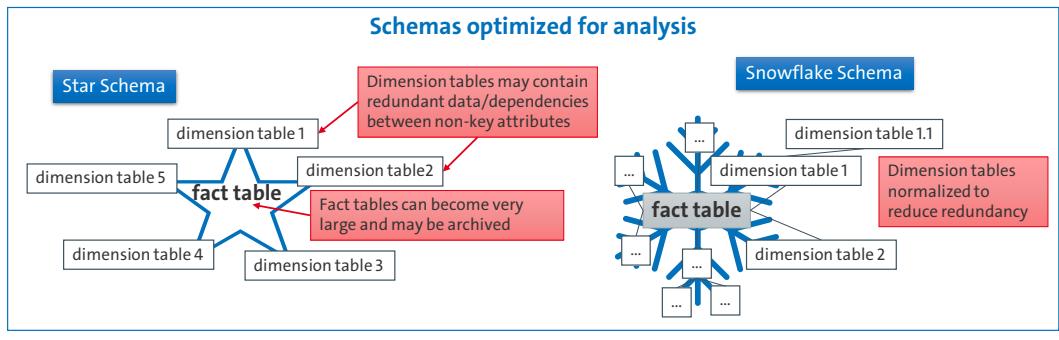
Data Warehouses and OLAP

Function

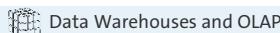
- Organization-spanning and application-independent data storage
- Collection, integration and distribution
- Analytic functionality for operative use

Data Provisioning and Analysis

- Dispositive database derived from consolidated data
- Mostly historic, maybe detailed data, mostly complete



22



More optimization for analysis

- Logical access paths
- Partitioning
- Pre-calculation of summed data (e.g. materialized views)
- Physical access paths: specific index structures (e.g. bitmap index)

Data analysis

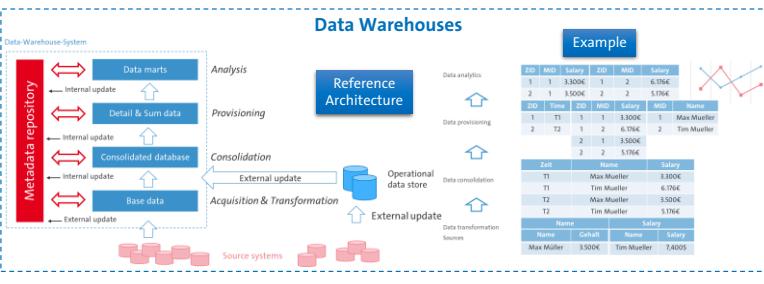
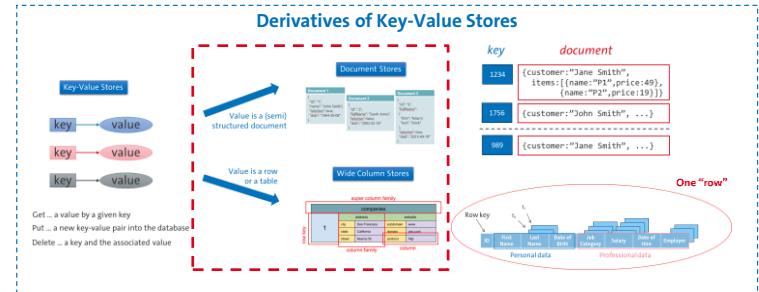
- Data-Mart databases derived from dispositive database
- Specific extracts for a specific class of applications
- Mostly proprietary formats on the physical level (e.g. MOLAP-systems)

Survey



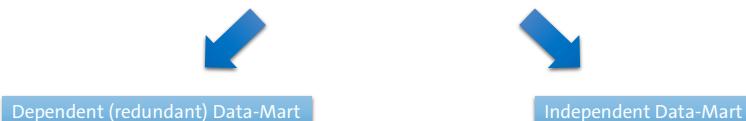
<https://evasys-online.uni-hamburg.de/evasys/online.php?pswd=J3Q6H>

Summary



Data-Mart Concept

Structural/content related subset of the complete DWH data, e.g.
Geographic/Organization/Function/Competition oriented Data-Marts



In reality, the distinction between data warehouse and data-mart can be difficult, because data warehouses do always provide a company-wide integration.

Why?

- Read-optimized layer: Data is stored in a denormalized data model for better read performance and better end user usability/understanding
- The Data Mart Layer is providing typically aggregated data or data with less history (e.g. latest years only) in a denormalized data model
- Dividing independent topics, ideally one subject per data mart
- Privacy aspects
- Reduction of data volume → Queries on the whole data warehouse can become a bottleneck
- Performance/Load distribution

DWH 2.0

- Structured and “unstructured” data
- Life cycle of data with different storage areas



Hot data: High speed,
expensive storage for most
recent data

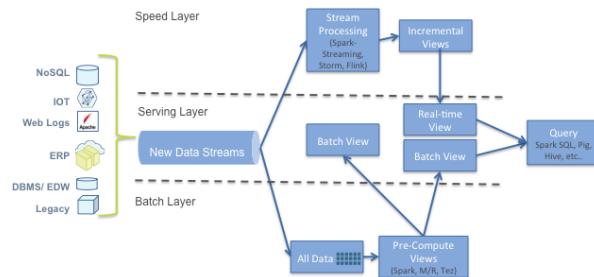


Cold data: Low speed, inexpensive,
large storage for old data; archival
data model with compression

- Metadata is an integral part of the DWH, not just an afterthought



Lambda Architecture (Big Data)



Batch Layer

- DWH alike, correct & complete
- Output in Read-only DB, complete replacement
- Hadoop/Spark de facto standard

Speed Layer

- Stream processing
- Real-time, latency is king, „batch gap“
- Correct-/completeness minor concern
- Apache Storm, Spark etc.
- Output into fast NoSQL DBs
- Indexes most recently added data

Serving Layer

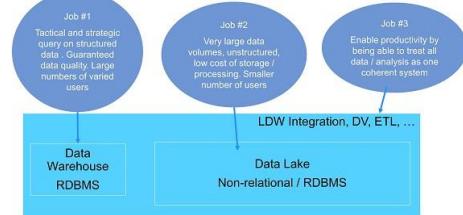
- Query processing
- Stores outputs, builds views
- Druid, Cassandra, HBase

Example: Streaming service analytics

- Speed layer: we need information about service issues → aggregate only needed to check if any threshold is hit and the system must react
- Batch layer: analytics about user groups

Logical Data Warehouse

- Focus on Jobs to be done
- DWH
 - Query processing on structured data
 - Guaranteed quality
- Data Lake
 - Very large volumes of unstructured data
- Logical Data Warehouse
 - Provide access to both underlying systems
 - Requires ETL functionality
 - Virtual integration

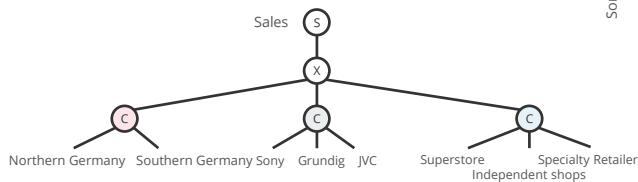
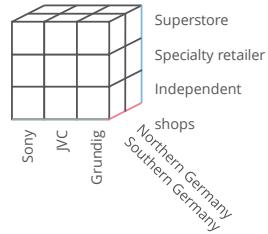


Data Lakes

- With cheap storage costs, people promote the concept of the data lake
 - Combines data from many sources and of any type
 - Allows for conducting future analysis and not miss any opportunity
- Collect everything
 - All data, both raw sources over extended periods of time as well as any processed data
 - Decide during analysis which data is important, e.g., no “schema” until read
- Dive in anywhere
 - Enable users across multiple business units to refine, explore and enrich data on their terms
- Flexible access
 - Enable multiple data access patterns across a shared infrastructure: batch, interactive, online, search, and others

Multidimensional Modelling

		Sales	Sony	JVC	Grundig
Northern Germany		Superstore			
		Specialty retailer			
		Independent shops			
Southern Germany		Superstore			
		Specialty retailer			
		Independent shops			

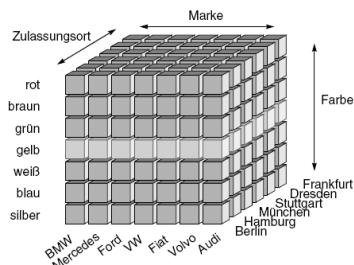


Static table also called “summary table”

Motivation: Reporting and interactive Analysis

- OLAP (Online Analytical Processing) on multidimensional data model
- Data Mining: Search for unknown patterns or relations in data
- Visualization

Basic Concept



Descriptive information (cube edges)
→ Dimensions (partially ordered set D of categorical attributes)

Quantified information (cube cells)
→ Facts (Measures / key performance indicators for analysis / aggregation)



Multidimensional schema Ω

- Set of dimension hierarchies (D^1, \dots, D^m)
- Set of measures (M^1, \dots, M^n)

Central data structure: multidimensional cube

- Descriptive data (categorical attributes)
- Quantified data (sum attributes)

Multidimensional modeling

“Predict” analytic patterns of users

- Drill-paths for navigation operators
- Limit to **meaningful** aggregation options

Data structures

- **Descriptive information** (cube edges) → dimensions
 - Hierarchies, dimensional attributes
 - Structural basis for selection and aggregation
- **Quantified information** (cube cells) → facts
 - Measures / key performance indicators for analysis / aggregation

Goal

- Orthogonal dimensional descriptions
- Clear separation of measures

Dimensions / Dimension hierarchy

- Partially ordered set D of categorical attributes

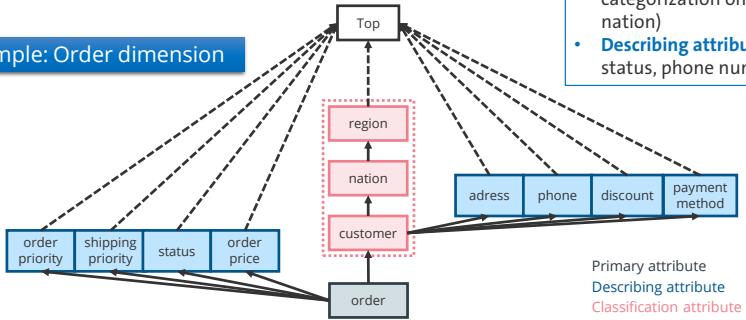
$(\{D_1, \dots, D_n, [Top]_D\}; \rightarrow)$

- Top_D is a generic maximum element w.r.t. “ \rightarrow ”, i.e.
 $\forall i (1 \leq i \leq n) : D_i \rightarrow [Top]_D$
- There is a D_i with finest granularity, i.e. $D_i \rightarrow D_j$ for all D_j
- “ \rightarrow ” denotes the functional dependency
- Partial ordering allows arbitrary parallel hierarchies

!!!Multidimensional Model is conceptual not physical!!

Schema of a Dimension

Example: Order dimension



Roles of categorical attributes

- **Primary attribute:** finest granularity (e.g. order)
- **Classification attribute:** implies multi-stage categorization on value level (e.g. customer, nation)
- **Describing attribute:** additional description (e.g. status, phone number)

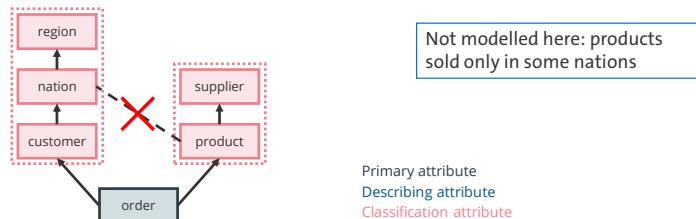
Orthogonality

- No functional dependencies between attributes from different dimensions

There is a functional dependency $A \rightarrow B$ if for all $a \in A$ there is exactly one $b \in B$, e.g. Germany \rightarrow Europe, but not Europe \rightarrow Germany

Schema of a Dimension

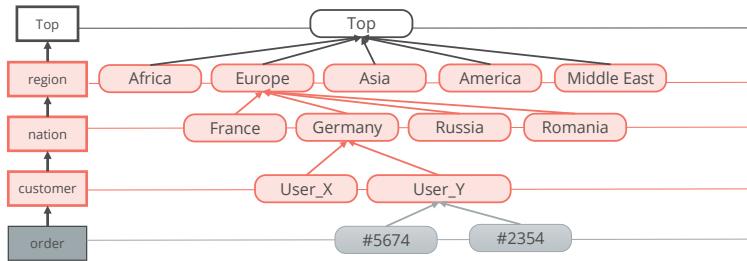
Example: Multiple hierarchies



Orthogonality

- No functional dependencies between attributes from different dimensions

Dimensional Values



→ Top can be reached via multiple paths

Functional dependencies define tree structure on instances

- Functional dependency corresponds to 1:N-relation!
- Every path from a classification attribute to Top defines a classification hierarchy

Facts and Measures

Quantifying part of a multidimensional data schema

Fact	Measure
<ul style="list-style-type: none">Defined as $F = (G, \text{SumType})$Granularity: $G = \{G_1, \dots, G_n\}$Subset of all categorical attributes of all existing dimensional hierarchies D_1, \dots, D_nSummation type: $\text{SumTyp} \in \{\text{FLOW}, \text{STOCK}, \text{VPU}\}$ <p><u>Example:</u> Delivered quantity with granularity $G = \{\text{orderNo}, \text{partNo}, \text{supplierNo}\}$</p>	<ul style="list-style-type: none">$M = (G, f(F_1, \dots, F_k), \text{SumType})$Calculation formula $f()$<ul style="list-style-type: none">Scalar function $+, -, *, /, \text{mod}$ etc., e.g. sales tax = quantity \cdot price \cdot tax rateAggregation functions, e.g. $\text{SUM}(), \text{MAX}()$, $\text{COUNT}()$, e.g. $\text{SUM}(\text{quantity} \cdot \text{price})$Order-based functions, e.g. cumulation, top-k calculationsNon-empty subset of all existing Facts F_1, \dots, F_k
<p>Example Fact: Number and price per product Scalar measure: Sum of price and sales tax for one product Aggregated measure: Aggregated price for a completed order Order-based measure: The k most expensive products of the order</p>	

Summability

Problem

- Not all functions can be summed up (median, quantiles, standard deviation)
- Even simple aggregation functions (SUM, AVG, MIN, MAX, COUNT, ...) cannot be aggregated further at will

Change of granularity

- $G = (G_1, \dots, G_n)$ is finer (same) $G' = (G'_1, \dots, G'_k)$, i.e. $G \leq G'$ iff, for each $G_j \in G$ exists a $G'_j \in G'$, s.t. $G_j \rightarrow G'_j$
- Coarsening / refinement of granularity adding / removing categorical attributes
- Example
 $(\text{orderNo}, \text{partNo}) \leq (\text{customerNo}, \text{brand}) \leq (\text{market segment})$

Necessary characteristics for summability

- Disjointness, completeness, type compatibility

Characteristics of Summability

Disjointness

- An individual value is considered **exactly once** during calculation
- General: Summability is given if functional dependencies exist between the categories that should be summed up

Number of students	2020	2021	2022	Total
Computer science	15	17	13	28
Economics	10	15	11	21
Total	25	32	24	49

Case 1: sum by year gives correct results

Case 2: sum up number of computer science students

- naive: $15+17+13 = 45$ gives wrong result (2 year overlap)
- Assume faculty founding in 2020: $15 + (17 - 15) + (13 - 2) = 28$

Completeness

- Measures on higher aggregation-level are completely derived from measures of lower aggregation-levels



Number of restaurants	2010	2011	2012	2013
Dresden	34	38	31	29
Leipzig	123	121	128	131
Chemnitz	21	18	24	27
OTHER	11	13	13	12
Total	189	190	196	199

- Number of restaurants in the triangle Dresden-Leipzig-Chemnitz
- OTHER for inns, not located in one of the cities

- Possibly, the whole space cannot be captured
 - Example: assume Berlin is not modeled as a federal state (city → federal state)
 - Weak functional dependency ($A \Rightarrow B$): for each $a \in A$ exists at most one $b \in B$
 - example: city ⇒ federal state
- Remove weak functional dependencies
 - NULL, OTHER, dummy values

Summation types

Summability

	FLOW	STOCK: aggregation over temporal dimension?		VPU
		no	yes	
MIN/MAX	✓		✓	✓
SUM	✓	✓	✗	✗
Avg	✓		✓	✓
COUNT	✓		✓	✓

FLOW

Can be aggregated freely, usually measured in <thing> per time

STOCK

States which can be aggregated freely except for a temporal dimension

Value-per-Unit

States which cannot be summed (except for min, max, and avg), e.g. exchange rate

→ Values of different points in time might not be aggregated into one value

Summation Types

Flow (event at time T)

- Can be aggregated freely
- Examples: order quantity of a certain item per day, number of traffic deaths per month, sales, earnings per year,...

Stock (status at time T)

- Can be aggregated freely, without a temporal dimension
- Items in stock over time → disjointness voided
- Example: number of school kids per month,

Value-Per-Unit (VPU)

- Current states, that cannot be summed
- Use of COUNT()-, MIN()-, MAX()- und AVG() is allowed
- Example: exchange rate, unit costs, ...

Exercise Summation Types

	FLOW	STOCK	VPU
Inventory			
Value added tax rate			
Ordered items per day			
#inhabitants per city			
Stock price			
Exams per semester			
SWS (Semesterwochenstunden/ semester hours)			
Exam grade			

The Data Cube

Data cube: $C = (DS, M)$

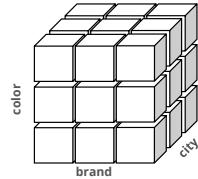
- Set of dimensional hierarchies: $DS = \{D^1, \dots, D^n\}$
- Set of fact-based measures: $M = \{M^1, \dots, M^m\}$

Domain of a data cube

- Cartesian product of all value ranges of all attributes of the cube schema

Data cube instance

- All cube cells from the cube domain



Cube is just a metaphor!

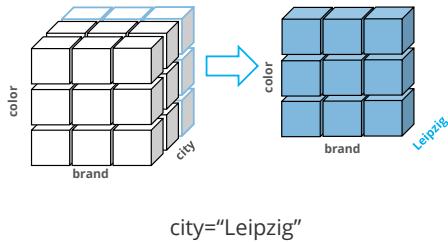
- Almost never, all cells are present (sparsity)
- Non-existent values on the implementation level become NULL or 0 on the model level!

There are usually more than 3 dimensions!

Operations on Data Cubes I

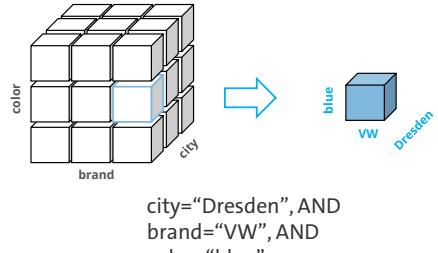
Slicing → Select “slices of a cube”

→ Selection via specification of classification attributes of **one dimension**



Dicing → Select a sub-cube

→ Selection by specification of classification attributes of **multiple dimensions**

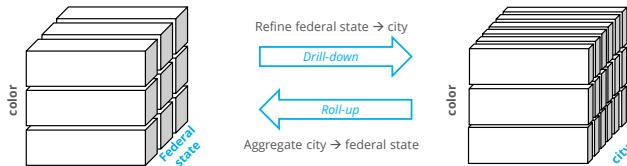


Operations on Data Cubes II

Drill-Down and Roll-Up

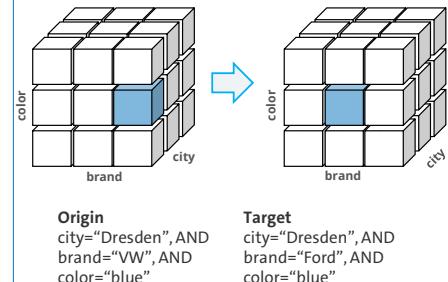
→ Moving along the dimension hierarchy
Drill-down: Disaggregation of measures into sub-measures
Roll-up: Aggregation of measures into super-measures

Hierarchy: Region ← Nation ← Federal state ← City



Drill-Across

→ Change from one sub-cube to another



Drill-Down and Roll-Up

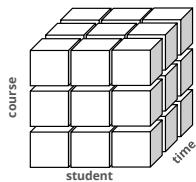
- Changes granularity, not Categorization

Drill-Across

- Dimension stays on the same hierarchy-level, but selection value changes
- Also change of data cube (Join of multiple data cubes)

Exercise Data Cube

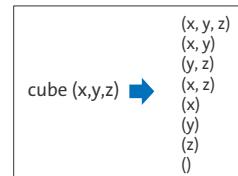
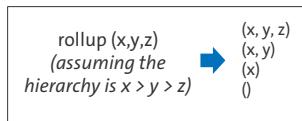
- Which operations are described? (slicing, dicing, roll-up, drill-down, drill-across)
- How could you visualize the results?



- a) name = "Jane Doh"
- b) refine name → first
- c) student="Jane Doh" AND time = "2 pm" AND course = "AD"
- d) time = "4 pm" AND course = "Math I"
- e) aggregate first → name
- f) student="Jane Doh" AND time = "2 pm" AND course = "AD"
→ student="Jane Doh" AND time = "10 am" AND course = "AD"

Data Cube in PostgreSQL

- Install extension: CREATE EXTENSION cube;
- Create a cube: SELECT cube(array[X,Y,Z]); //creates cube with dimensions X, Y, and Z
- Selected functions:
 - Cube_union (cube,cube)
 - Cube_inter(cube,cube) → intersection
 - cube_subset (cube, integer[]) → new cube from existing group using list of dimension indexes, e.g. for dicing
- Rollup is a separate function and a subclause of GROUP BY



<https://www.postgresql.org/docs/current/cube.html>

<https://www.timescale.com/learn/postgresql-extensions-cube>

- Cube can also be used in GROUP BY
- Common use of rollup: aggregation of data by year, month, and day

Multidimensional database design

	Classical relational database design	Multidimensional database design	
Conceptual schema (semi-formal)	Variants of entity-relationship modelling	Different modelling languages, e.g. mE/R, mUML, ADAPT,...	
Logical schema (formal)	Relations with attributes	Data cube: facts and measures	Dimensional hierarchy with categorical attributes: classifying and describing attributes
Internal/physical schema	Memory organisation (primary/secondary indexes, partitioning,...)	Relational storage (ROLAP): Star/Snowflake schema patterns	Multidimensional storage (MOLAP): native implementation

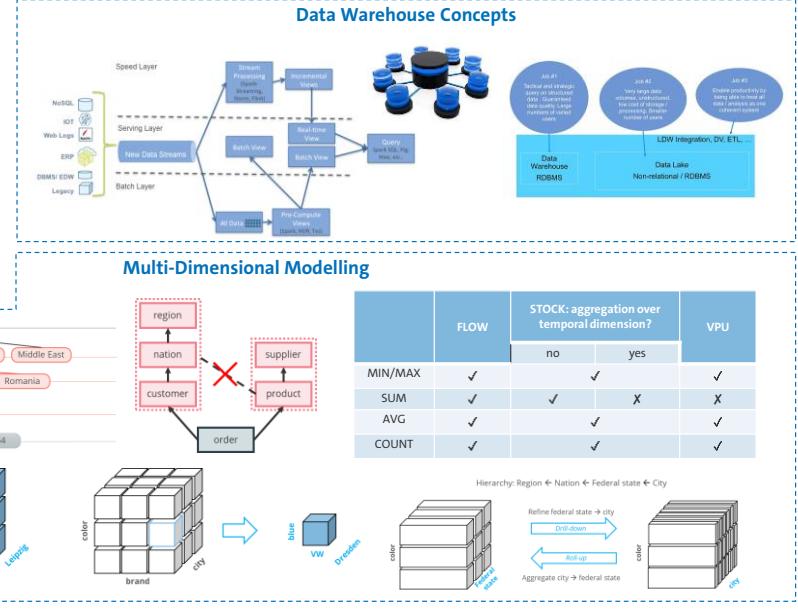
ROLAP = Relational OLAP

MOLAP = Multidimensional OLAP

HOLAP → Hybrid solution

Summary

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



Multidimensional database design

	Classical relational database design	Multidimensional database design	
Conceptual schema (semi-formal)	Variants of entity-relationship modelling	Different modelling languages, e.g. mE/R, mUML, ADAPT,...	
Logical schema (formal)	Relations with attributes	Data cube: facts and measures	Dimensional hierarchy with categorical attributes: classifying and describing attributes
Internal/physical schema	Memory organisation (primary/secondary indexes, partitioning,...)	Relational storage (ROLAP): Star/Snowflake schema patterns	Multidimensional storage (MOLAP): native implementation

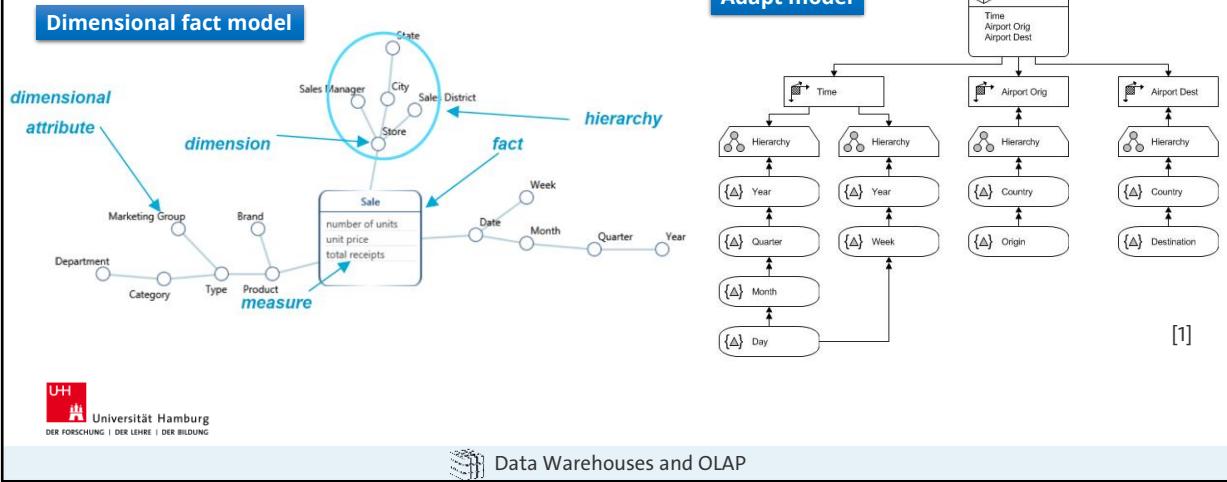
ROLAP = Relational OLAP

MOLAP = Multidimensional OLAP

HOLAP → Hybrid solution

Conceptual Models

Modelling Approaches



[1] source: <https://www.datenbanken-verstehen.de/data-warehouse/data-warehouse-design/anforderungsanalyse/adapt-notation/>

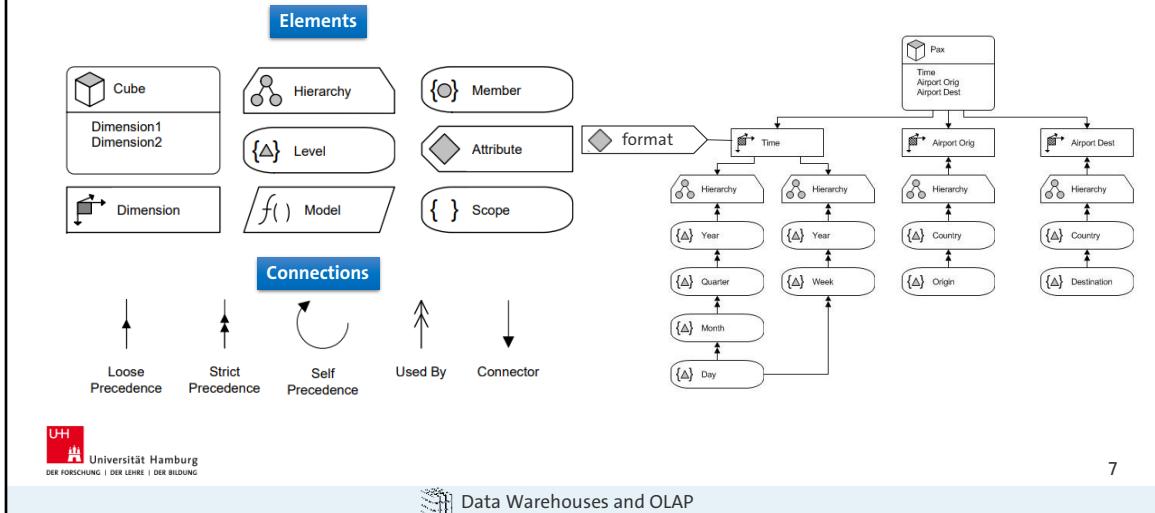
Software for ADAPT: Midrosoft visio with ADAPT Add-On

Further reading

Getting Started with ADAPT:

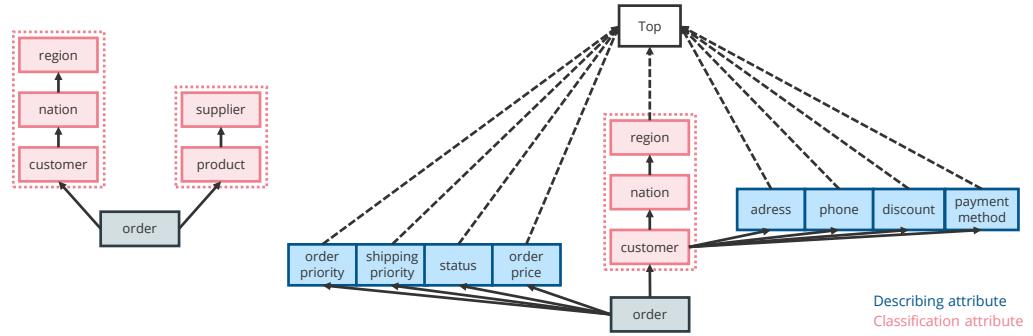
http://www.symcorp.com/downloads/ADAPT_white_paper.pdf

Conceptual Models: ADAPT



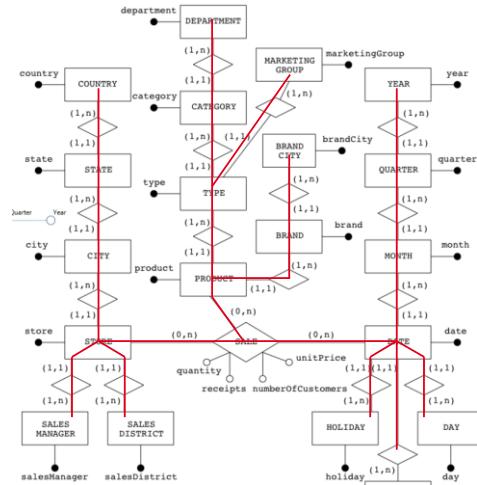
- There are also different dimension types and scopes which we will not cover here
- *Used by* is used with *Models*

Exercise: ADAPT



Conceptual Models

Multidimensional ER



Relational Data model

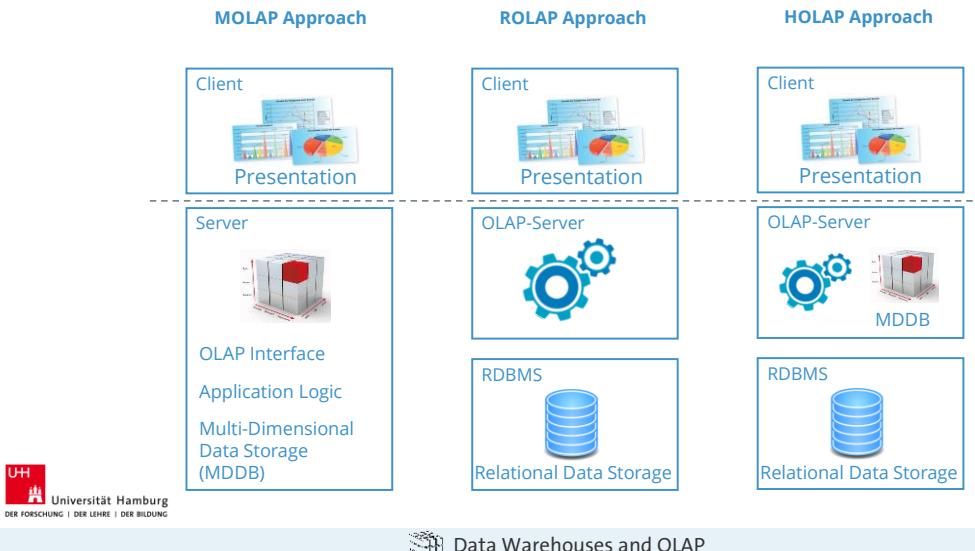
- Avoiding redundancies
- OLTP queries
- Preferably efficient querying of individual entries

In Data Warehouses

- OLAP queries usually affect multiple entries
- Value ranges and/or aggregates
- Redundancy can shorten query response time

→ Different relational data models

Mapping alternatives



MOLAP

Properties

- Raw data & aggregates stored in OLAP server
- Direct storage as cubes

Pros

- No transfer of modeling concepts
- Separation of descriptive und quantified attributes fixed in the model
- Multiple integrated statistical operators (esp. OLAP functions)
- Fast response times through pre-calculation
- Very efficient due to optimized storage structures

Cons

- Proprietary systems
- Often compression is missing (sparsity) , scalability (MB to GB)
- High storage requirements (factor 10 to 100) due to aggregation/pre calculation
- Often limited number of dimensions (10 to 64)

Products

- Oracle Express, Hyperion Essbase, Cognos Powerplay, Seagate Holos

ROLAP

Properties

- Data is stored in RDBMS
- Information is created via SQL
- Additional tables for aggregates

Pros

- Proven database technology
- Scalability for big volume data / high number of dimensions

Cons

- Installation overhead (interaction of different components)
- Overhead for mapping the Data Cube and complex queries to SQL
- Extensive Meta data management for parameters
- Slower

Products

- Almost all RDBMS vendors (Oracle, DB2, Microsoft, etc.), SAP BW / SIMCE

HOLAP

Properties

- „best of both worlds“
- Data partially stored in MOLAP Cubes and RDBMS

Pros

- Proven database technology
- Improved scalability / response times

Cons

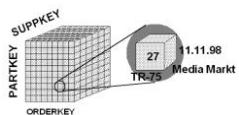
- Installation overhead (interaction of different components)
- Complexity / Handling, Which data should be stored where?

Products

- MicroStrategy, Essbase, Mondrian, Microsoft Analysis Services

ROLAP Schemas

- Table with primary key as compound of foreign keys from dimensions
- Only populated cube cells are stored as tuples



ORDERKEY	SUPPKEY	Partkey	QUANTITY
TR-75	MediaMarkt	11.11.98	27
AC300	ProMarkt	18.11.12	5
..
..
..

Separation of structure and content leads to



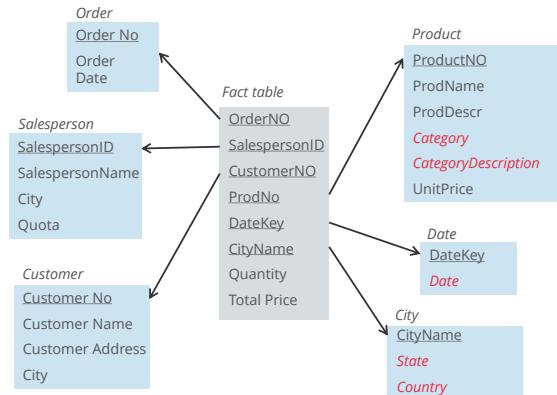
Fact table (content)

- Central table
- Compound primary key from foreign keys of dimension tables (explicit assignment of facts)
- Few columns, many tuples (millions, billions)
- Typically, only numeric data types

Dimension tables (structure)

- Classifying and descriptive attributes
- Many columns, but few tuples (< 10% of fact table)
- Used for selection (by utilizing bitmap-indices)
- Used for aggregation (group-by-clause)
- Foreign key links dimension table and fact table

Star Schema



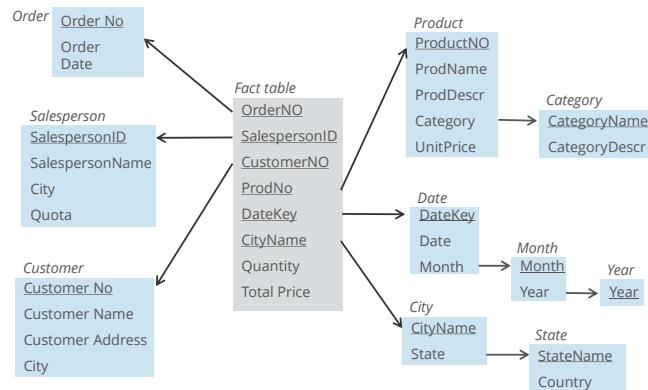
Pros

- Intuitive for users
- Low response time because of read-optimization
- Adaptability to changing structure
- Partitioning and optimization possible
- Less redundancy

Cons

- Not that update friendly
- Multi-hierarchies can be only modeled indirectly
- No distinction between classifying and descriptive attributes (only implicit hierarchies)

Snowflake Schema



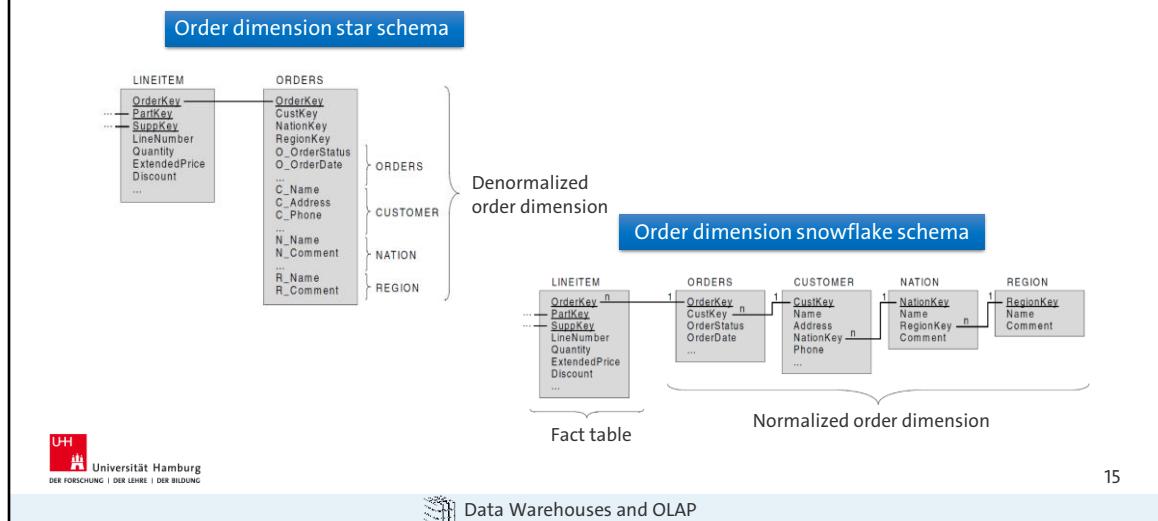
Main property: normalized dimension tables

- +No redundancy
- +Update-friendly
- +Inherent mapping of structural information of dimension hierarchies
- Less performance due to additional joins

Redundancy: “Resist the urge to normalize”

- Fact table size of hundreds of GBs/TBs
- Plus, index on fact table of similar size
- Dimension tables size of several hundred MBs
- Little storage savings from normalization
- Only favorable for large dimensions or many updates

Comparison of Star and Snowflake



Benefits Star over Snowflake

- Higher query performance
 - Analytic queries typically address higher aggregation levels
 - Less joins because of denormalization
- Data volume
 - Dimension tables small in comparison to fact tables
 - Additional storage required for denormalized dimension tables rather than small
- Changes of classification structure (meta-instance level)
 - Occur rarely
 - Drawback: higher change costs due to duplicate identification for denormalized dimension tables
- Simplicity of structure
 - Fewer dimension tables than snowflake schema
 - Simpler creation of SQL queries by the OLAP-server

Combining Star & Snowflake Schema

Properties

- Some dimensions normalized

- Some dimensions denormalized

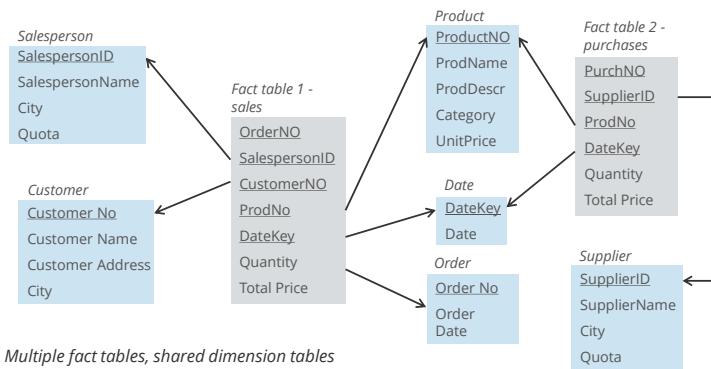
Frequency of updates

- If update frequency of a dimension is high, it makes sense to normalize this dimension to drastically reduce update costs

Number of dimension elements

- High number of finest-grain dimension elements leads to high savings through normalization
- High number of aggregation levels leads to high savings through normalization
- Multiplicative relationship

Galaxy Schema



Relational Mapping of Hierarchies

Vertical Mapping

Product			Product group			Product family			Category	
ID	Product	ID2	ID	Product group	ID3	ID	Product family	ID4	ID	Category
1	Flour X	1	1	Flour	1	1	Bakery goods	1	1	Food
2	Sugar Y	2	2	Sugar	1	2	Beverages	1		
3	Water Z	3	3	Water	2					

Horizontal Mapping

ID	Product	Product group	Product family	Category
1	Flour X	Flour	Bakery goods	Food
2	Sugar Y	Sugar	Bakery goods	Food
3	Water Z	Water	Beverages	Food

Recursive Vertical Mapping

ID	Product	Product group	Product family	Category
1	Flour X	Flour	Bakery goods	Food
2	Sugar Y	Sugar	Bakery goods	Food
3	Water Z	Water	Beverages	Food

ID	Product	ParentID
1	Food	NULL
2	Bakery goods	1
3	Beverages	1
4	Flour	2
5	Sugar	2
6	Water	3
7	Flour X	4
8	Sugar Y	5
9	Water Z	6

Vertical mapping

- Explicit mapping using separate relations
- Normalized
- Ref. snowflake schema

Horizontal mapping

- Implicit mapping of equally ranked attributes
- Denormalized
- Ref. star schema

Recursive vertical mapping

- Explicit mapping of arbitrary deep hierarchies
- Combination with horizontal mapping possible
- Extension for irregular hierarchies possible
- Recursive resolution of hierarchy via a set of self-joins

Information Assurances

SQL level: Add primary/foreign key references

- Assure referential integrity at CREATE TABLE, or later with ALTER TABLE command
- Example order dimension ORDERS :

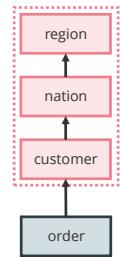
```
ALTER TABLE TPCD.ORDERS
ADD FOREIGN KEY ORDERS_FK1 (O_CUSTKEY)
    REFERENCES TPCD.CUSTOMER;

SET INTEGRITY FOR TPCD.ORDERS IMMEDIATE CHECKED;
```

Declaration of functional dependencies between attributes of a dimension requires extensions

Example:
Oracle SQL-dialect

```
CREATE DIMENSION ORDER_DIM
LEVEL ORDER IS ORDERS.ORDERKEY
LEVEL CUSTOMER IS ORDERS.CUSTOMERKEY
LEVEL NATION IS ORDERS.NATIONKEY
LEVEL REGION IS ORDERS.REGIONKEY
HIERARCHY ORDER_HIERARCHY (
    ORDER CHILD OF
    CUSTOMER CHILD OF
    NATION CHILD OF
    REGION )
ATTRIBUTE ORDER DETERMINES
    (O_ORDERSTATUS, O_ORDERDATE,...)
ATTRIBUTE CUSTOMER DETERMINES
    (C_NAME, C_ADDRESS, C_PHONE,...);
```



Primary/foreign key relation between dimension- & fact table

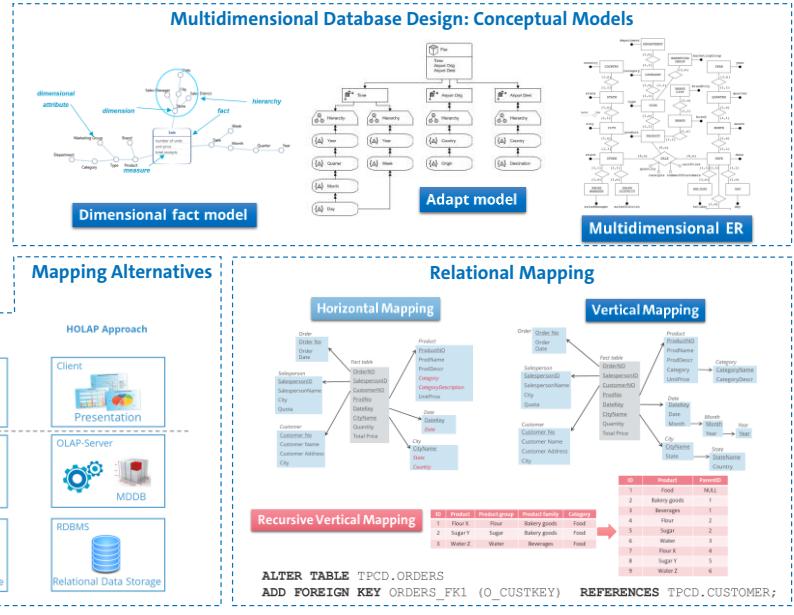
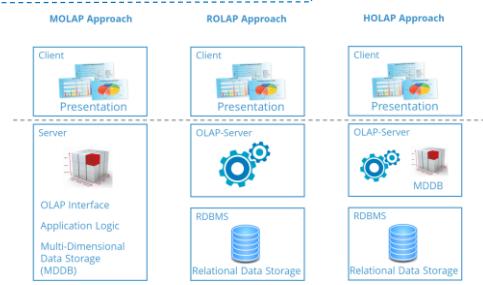
- Referential integrity: assure that every quantified part can be evaluated against the descriptive part of a multidimensional schema
- Every part of the composite primary key of the fact table has to be a foreign key to the primary key of the corresponding dimension table

Problem

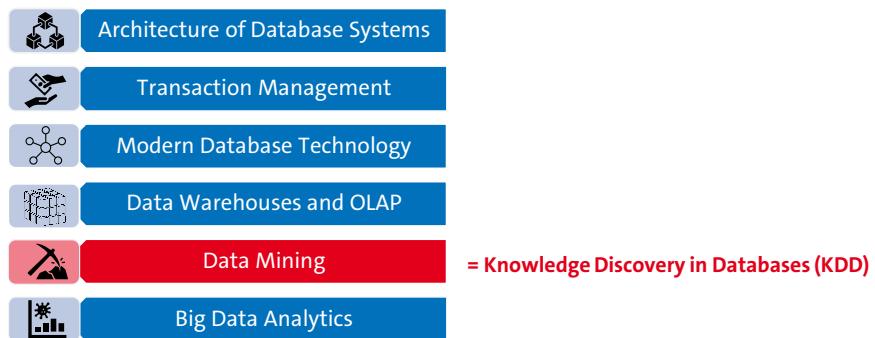
- PK/FK relations can be declared using classical DB means, **BUT** declaration of functional dependencies between attributes of a dimension → requires extensions , e.g. CREATE DIMENSION (Oracle)
- Knowledge about functional dependencies is necessary to check disjointness
- Information assurances describe functional dependencies, but inside a relation (e.g. denormalized dimension tables in star schemas)

Summary

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



Course Outline



Mining as an explorative process

- Finding cues
- Making hypotheses
- Evaluating Hypotheses
- Getting interesting/useful information

Knowledge Discovery in Databases



- Many different techniques
- Extremely laborious parameter tuning
- Few clues for performance predictions

- (Semi-)automatic extraction of knowledge from databases which is:
 - Valid (in the statistical sense)
 - Unknown so far
 - Potentially useful
- Combination of approaches from databases, statistics, and machine learning
- Differences to querying a database:
 - No precise semantics
 - Not 100% perfect results
 - No perfect results
 - Solutions hard to port to similar application domains

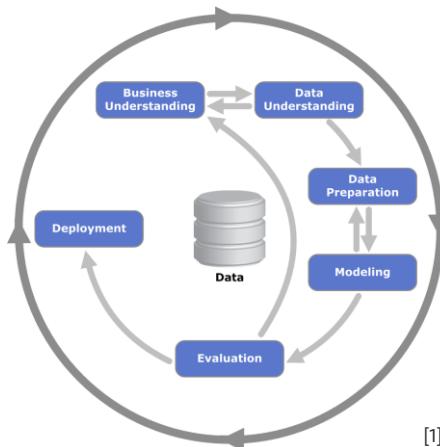
Example tasks

- Customer relationship management
 - Grouping of customer populations (tailored marketing)
 - Prediction of customer behaviour (individualized marketing)
 - Risk assessment (risky credits, fraudulent credit card use)
- Fault analysis
 - Interdependencies between faults
 - Interdependencies between production processes or maintenance

procedures and faults

- Time-series analysis
 - Trend detection
 - Stock market development
 - Event prediction (stock market crashes, bankruptcies, natural disasters)
 - Intrusion detection
- Web Usage and Text Mining

The Data Mining Process – CRISP-DM



[1]

 Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

5



[1] source: Kenneth Jensen@Wikipedia based on
<ftp://public.dhe.ibm.com/software/analytics/spss/documentation/modeler/18.0/en/ModelerCRISPDM.pdf>

- Cross industry standard process for data mining
- Life-cycle model

1. Business understanding phase

- Analysis of objectives and requirements
- Problem definition
- Initial strategy development

2. Data understanding phase

- Data collection
- Exploratory data analysis
- Assessment of data quality

3. Data preparation phase

- Cleansing, transformation etc.

4. Modelling phase

- Selection of modelling techniques and tools

- Parameter tuning / optimization
- Data analysis

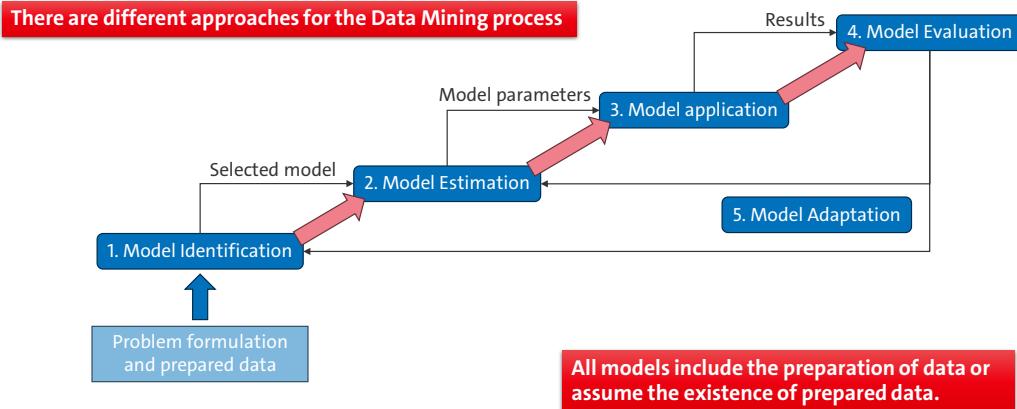
5. Evaluation Phase

- Evaluation of the model
- Comparison of the outcome to the initial objectives
- Deployment decision

6. Deployment phase

- Reporting
- Transfer to other application cases
- If applicable: introduction into day-to-day business

The Data Mining Process



1. **Model Identification** – Choose the optimal model type
2. **Model Estimation** – Instantiation of the model by training its model parameters
3. **Model Application** – Usage of the model to calculate the next results
4. **Model Evaluation** – Compare the model's results with real values using an error measure
5. **Model Adaption** – Adaption of the model parameters or the model type

Data Preprocessing

Data Types

Metrics

Handling of Missing Data

Outlier Detection

Dimensionality Reduction

Value Count Reduction

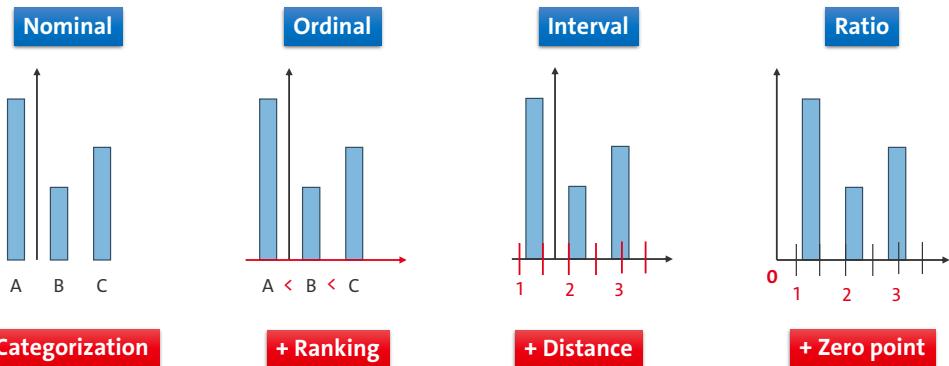
Goals

- Decrease runtime of data mining process
- Decrease resource requirements of data mining process
- Increase quality of mining result

Important aspects

- Handling of missing data
- Detecting outliers
- Reducing the number of dimensions
- Reducing the number of values
- Transforming data values (e.g. binning, rescaling)
- Depends on data type
- Uses similarity/distance measures

Data Types



Nominal scale

- No problem-specific order and distance relation
- Mathematical Operators: $=$, \neq
- Central Tendency: mode (most often occurring value)
- Examples: color, zip-code

Ordinal scale

- Problem-specific order relation
- No problem-specific distance relation
- Mathematical Operators: $=$, \neq , $>$, $<$
- Central Tendency: mode, median
- Examples: income classes, medal ranks, age

Interval scale

- Problem-specific order and distance relation
- No problem-specific zero point
- Differences meaningful, ratios meaningless
- Mathematical Operators: $=$, \neq , $>$, $<$, $+$, $-$
- Central Tendency: mode, median, arithm. mean, deviation

- Examples: temperature (Celsius), date (B.C./A.D.)

Ratio scale

- Problem-specific zero point (absence of the feature)
- Differences and ratios meaningful
- Mathematical Operators: $=$, \neq , $>$, $<$, $+$, $-$, \cdot , $/$
- Central Tendency: mode, median, arithm./geom. mean, deviation
- Examples: temperature (Kelvin), speed, length, age, quantity

Metrics

Many data mining techniques are based on some notion of distance, similarity or dissimilarity.

Data Types

Metrics

Handling of Missing Data

Outlier Detection

Dimensionality Reduction

Value Count Reduction



Examples

Numeric values/points: Minkowski distance

$$d(\vec{x}, \vec{y}) = (\sum_{i=1}^n |x_i - y_i|^m)^{1/m}$$

$m=1$: Manhattan distance
 $m=2$: Euclidian distance
 $m=\infty$: Tchebychev distance

Sets/Bags/Vectors:

- Cosine similarity = $\cos(\theta)$
- Jaccard Coefficient

$$J(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

Signatures, histograms, Probability distributions:

- Earth Mover's distance/Wasserstein metric

Strings:

- Soundex (phonetic)
- Monge-Elkan similarity (sequence- and set-based)
- Levenshtein distance (sequence based)

9

Levenshtein Distance

String edit distance = number of insertions/deletions/exchanges necessary to transform one string into another

Examples

- “Bear” and “Bar” → 1 (insert “e”)
- “Bear” and “Goal” → 3 (exchange “B”, “e”, and “r” for “G”, “o”, and “l”)

Algorithm for distance between word x and word y

1. Create initial matrix of size ($|x|+1, |y|+1$)
2. Fill 1st row and column with ascending numbers starting at 0
3. Compute the rest pf the matrix according to the following rule:

$$D_{i,j} = \min \begin{cases} D_{i-1,j-1} & \text{if } x_i = y_i \\ D_{i-1,j-1} + 1 & (\text{exchange}) \\ D_{i,j-1} + 1 & (\text{insert}) \\ D_{i-1,j} + 1 & (\text{delete}) \end{cases}$$

4. Get the distance from the lower right cell in the matrix

- Finding the minimum distance is an optimization problem
- Space and time requirements $O(m \cdot n)$

Levenshtein Distance: Example

1. Create initial matrix of size $(|x|+1, |y|+1)$
2. Fill 1st row and column with ascending numbers starting at 0
3. Compute the rest pf the matrix according to the following rule:

$$D_{i,j} = \min \begin{cases} D_{i-1,j-1} & \text{if } x_i = y_i \\ D_{i-1,j-1} + 1 & \text{(exchange)} \\ D_{i,j-1} + 1 & \text{(insert)} \\ D_{i-1,j} + 1 & \text{(delete)} \end{cases}$$

4. Get the distance from the lower right cell in the matrix

		b	e	a	r
	0	1	2	3	4
g	1	1	2	3	4
o	2	2	2	3	4
a	3	3	3	2	3
l	4	4	4	3	3

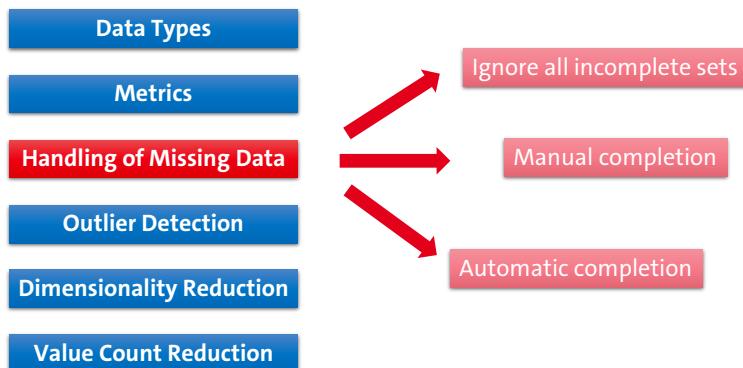
Translation for step 3



Exercise: Levenshtein Distance

Determine the Levenshtein distance (and the according matrix) between “english” and “danish”

Data Preprocessing



Some data mining tools are insensitive to missing data

Ignore all incomplete objects

- Might result in the loss of a substantial amount of data
- Problem: maybe a systematic correlation between target of mining process and missing values (e.g. customers who do not answer a specific question of a survey)

Manual completion

- Expensive in time and money

Automatic completion

- Using a global constant
- Using the global mean value
- Using a class-dependent mean value
- Use a predictive model, e.g. based on feature correlations

Data Preprocessing

Data Types

Metrics

Handling of Missing Data

Outlier Detection

Dimensionality Reduction

Value Count Reduction

Finding tuples which are

- Considerably dissimilar
- Exceptional
- Inconsistent with the remaining data

Example: Distance based detection

1. Build the distance matrix
2. Find the neighborhood of all points → all points where the distance is below a defined threshold are in the neighborhood
3. All points where the number of neighbors is below a defined threshold, are outliers

- Not applicable if the application is aimed at outlier detection, e.g. fraudulent credit card transactions

Manual detection supported by visualization tools

- Only for low dimensional data

Statistical methods

- Threshold for the variance, e.g. two times variance
- Only applicable if the distribution is known

Using domain knowledge

- Value restrictions, e.g. $0 < \text{age} < 150$
- Less applicable for multi-dimensional data

Distance-based detection

- A sample is an outlier if it has not enough neighbours

Deviation-based methods

- Measure the dissimilarity of a data set (e.g. variance)
- Determine the smallest subset of data that if removed results in the largest reduction of dissimilarity
- Combinatorics of subset selection → extremely expensive

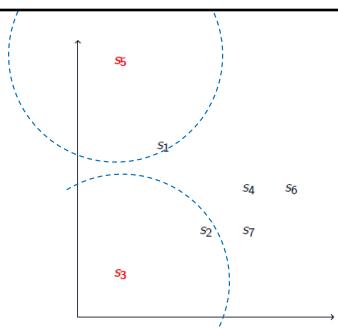
Outlier Detection: Example

Data Set $S = \{s_1, \dots, s_7\} = \{(2, 4), (3, 2), (1, 1), (4, 3), (1, 6), (5, 3), (4, 2)\}$

Distance matrix

Neighborhood: $d \leq \theta = 3$

	s_1	s_2	s_3	s_4	s_5	s_6	s_7
s_1	2.236	3.162	2.236	2.236	3.162	2.828	
s_2	2.236	2.236	2.236	1.414	4.472	2.236	1.000
s_3	3.162	2.236	2.236	3.605	5.000	4.472	3.162
s_4	2.236	1.414	3.605	4.242	1.000	1.000	
s_5	2.236	4.472	5.000	4.242	5.000	5.000	
s_6	3.162	2.236	4.472	1.000	5.000	1.414	
s_7	2.828	1.000	3.162	1.000	5.000	1.414	



Number of neighbors per point

Sample	s_1	s_2	s_3	s_4	s_5	s_6	s_7
	4	5	1	4	1	3	4



If threshold for outliers is set to < 2 neighbors, s_3 and s_5 are outliers

Exercise Outlier Detection

$S = \{S1, \dots, S4\} = \{(2,3), (1,1), (3,3), (3,2)\}$

Data Preprocessing

Data Types

Metrics

Handling of Missing Data

Outlier Detection

Dimensionality Reduction

Value Count Reduction

High-dimensional spaces are sparse

→ Keeping the same object density in a space with more dimensions requires exponentially more objects

→ To enclose a prespecified portion of objects, an increasingly large part of the hypercube needs to be “encircled”

Example

Portion p	Dimensions n	Edge length e	$e^n = p$
0.1	1	0.1	$0.1^1 = 0.1$
0.1	2	0.316	$0.316^2 = 0.1$
0.1	3	0.464	$0.464^3 = 0.1$
		...	
0.1	10	0.794	$0.794^{10} = 0.1$
		...	
0.1	100	0.977	$0.977^{100} = 0.1$

- Almost every object is closer to an edge of the cube than to another sample object
- Almost every object is an outlier

Which data can be discarded without sacrificing the quality of the data mining results?

Too many dimensions:

- Mining results degrade (insufficient amount of data)
- Resulting model is incomprehensible
- Problem becomes intractable

Too few dimensions:

- Data dependencies are lost
- Mining results degrade (limited expressiveness)

Feature selection approaches:

Idea: discard features (i.e. attributes, dimensions) which

- have many inaccurate/inconsistent values
- have many missing values

- do not provide much (relevant) information
- contribute least to the overall class distinction capability (task specific criterion)

Approaches: Feature ranking, minimum subset selection

Feature composition approaches:

Idea: features are composed into a new feature set with reduced dimensionality

→ Given feature space is transformed into a more compact feature space
without losing relevant information

Approach: Principal component analysis (PCA)

Data Preprocessing

Data Types

Metrics

Handling of Missing Data

Outlier Detection

Dimensionality Reduction

Value Count Reduction

Increase Performance (less values to process)
Simplify mining process (e.g. finding of rules)



One dimensional

Feature discretization (binning)
→ Mapping values to intervals

Multi dimensional

Clustering of feature vectors
→ Splitting techniques



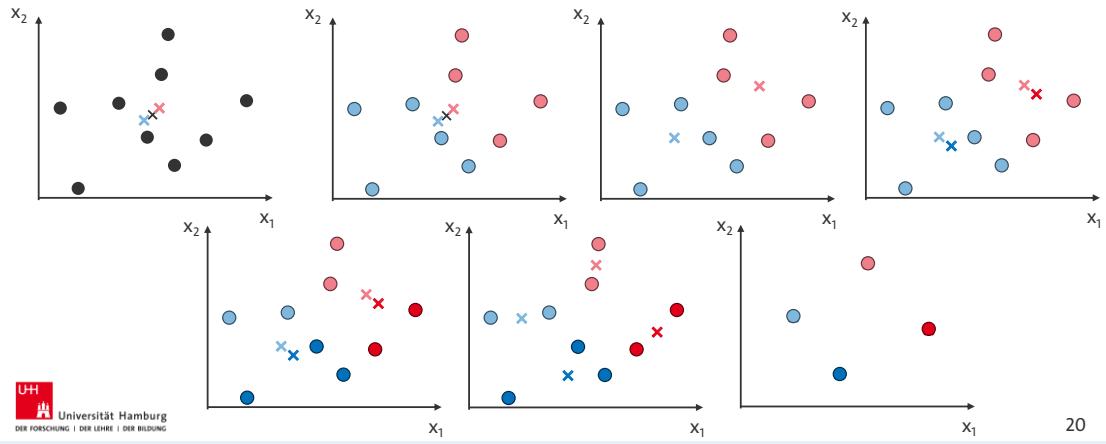
Splitting Techniques: Splitting of bins (vector quantization)

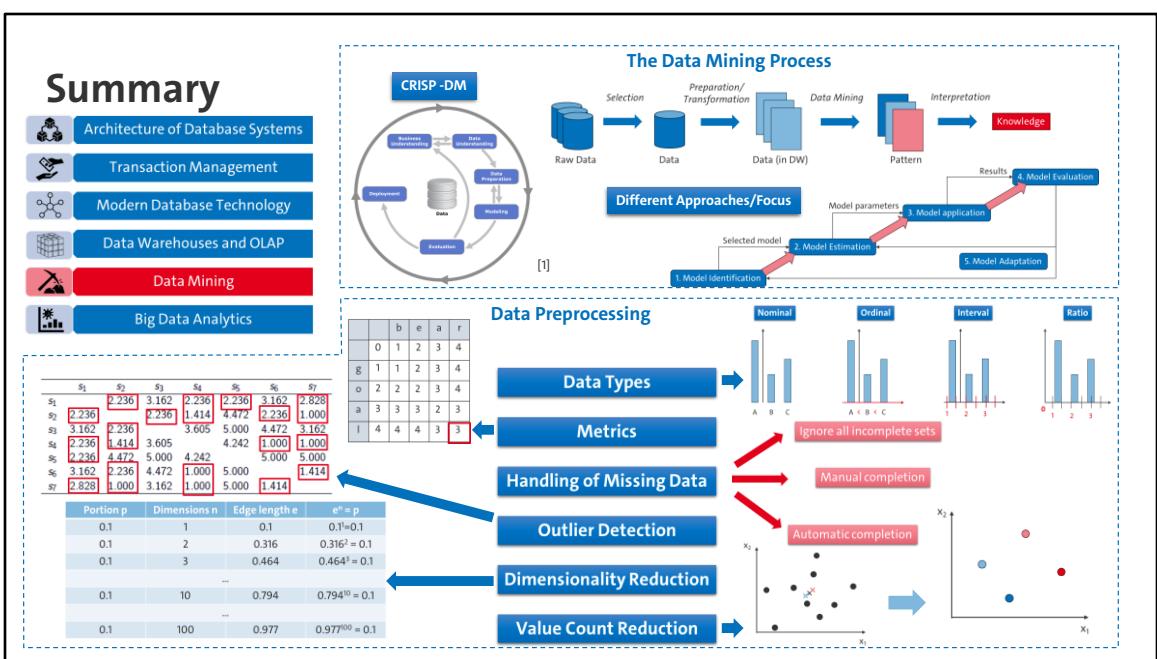
1. Start: All values in a single bin/cluster
2. Compute the mean of all data values (centroid)
3. Split each centroid into 2 or more centroids (bins)
4. Assign data points to the nearest centroid (bin)
5. Continue until enough bins have been generated

Splitting of bins

Input: 9 values
Output: 4 values

1. Start: All values in a single bin/cluster
2. Compute the mean of all data values (centroid)
3. Split each centroid into 2 or more centroids (bins)
4. Assign data points to the nearest centroid (bin)
5. Continue until enough bins have been generated





[1] source: Kenneth Jensen@Wikipedia based on
<ftp://public.dhe.ibm.com/software/analytics/spss/documentation/modeler/18.0/en/ModelerCRISPDM.pdf>

Time Series Model

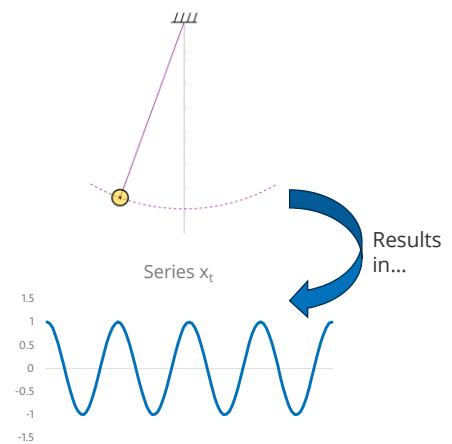
Idea

- Consider a time series as an unknown process combined with an unpredictable deviation
- It is measured at equidistant time instances and results in a sequence
- The unknown process is represented by a descriptive model P_t
- The unpredictable deviation res_t is assumed to be random, with
 - Expected value is 0
 - Variance is constant
 - Normally distributed
- Thus, the time series is represented by a descriptive model:

$$x_t = P_t + res_t \quad \begin{matrix} (P_t \dots \text{process} \\ res_t \dots \text{residuals}) \end{matrix}$$

$$\underline{x}^T = (x_1, \dots, x_t, \dots, x_T)$$

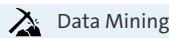
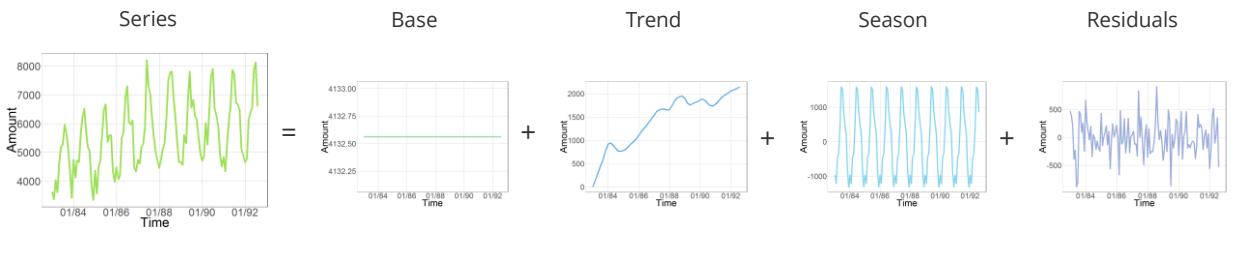
Example: Pendulum



Be aware of the sampling theorem!

Terminology

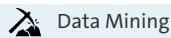
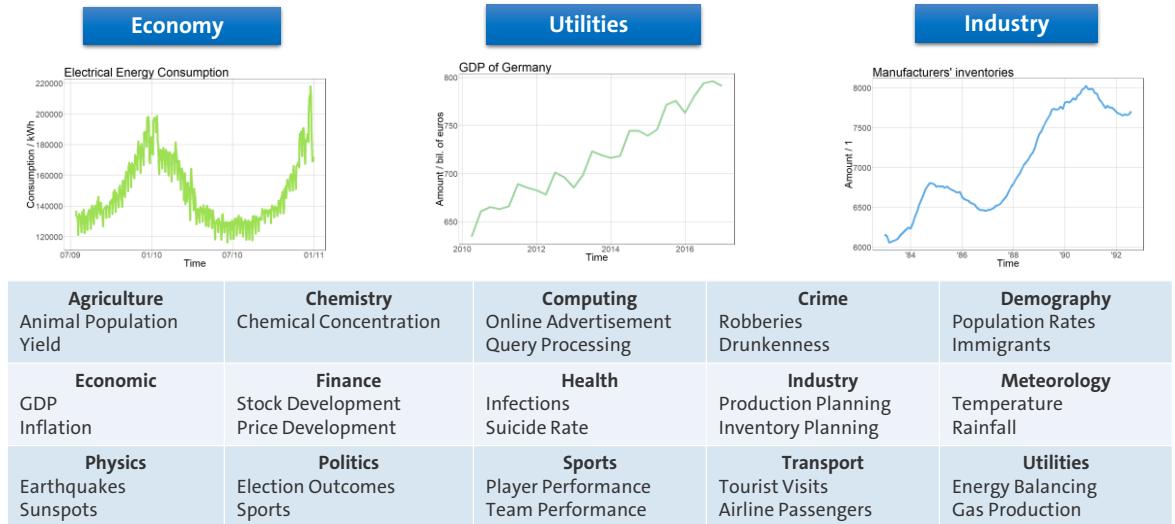
Additive composition $x_t = \text{base}_t + \text{trend}_t + \text{season}_t + \text{res}_t$



- Time series components
- Base: stationary part of the time series
- Trend: long-term change in the mean level
- Season: cyclical repeated behaviour
- Residuals: unstructured information assumed to be random

→ Often, base is part of the trend component!

Time Series Occur in Many Domains

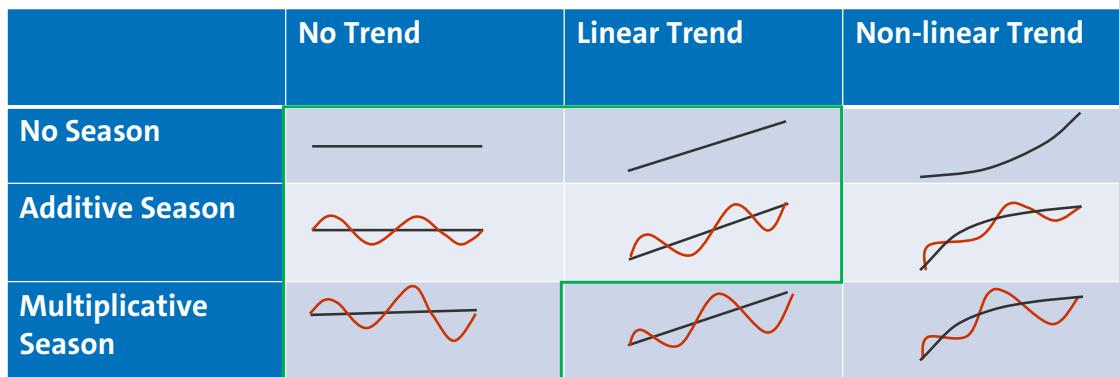


GDP – Gross domestic product

Time series analysis: Further reading

- Fulcher, B. D., Little, M. A., & Jones, N. S. (2013). Highly comparative time-series analysis: the empirical structure of time series and their methods.
- Wang, X., Smith, K., & Hyndman, R. J. (2006). Characteristic-based clustering for time series data. Data Mining and Knowledge Discovery.

Composition Types



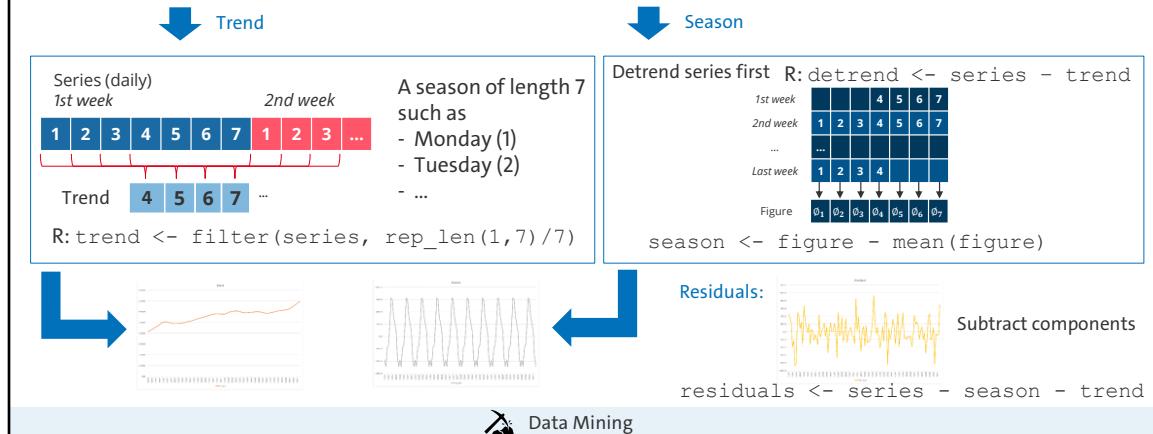
The most common models

Pegels, C. C. (1969). Exponential Forecasting: Some New Variations. *Manage. Sci.*, 15(5), 311–320.



Decomposition of Time Series: Classical Decomposition

- Moving-average filter of continuous windows of size N (N is odd)
- Extraction of trend by windows that take into account the season length
- Extraction of season by averaging each time instance of the same seasonal position (all Mondays, all Tuesdays,...)



Disadvantage: Does not decompose the endpoints

Average centering

- A technique needed if season length is even
- Take two moving averages and average their result

More techniques

X11

- Method published by the U.S. Bureau of the Census and used adopted by several statistical agencies
- Based on classical decomposition with several moving-average steps
- Advantages
 - Endpoints decomposed using predictions from ARIMA forecasting method
 - Extensions for holiday effects
 - Support slowly varying season representing changes in seasonal behavior
- Disadvantage: Only for quarterly and monthly time series

STL

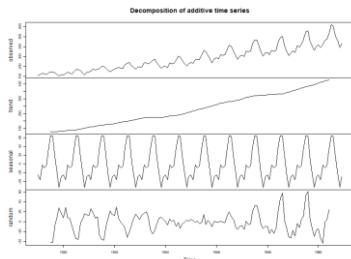
- Application of regression technique (*Loess smoothing*)

- Recursive application of smoothing and robustness checks
- Advantages
 - Support arbitrary season lengths
 - Versatile and robust decomposition technique
- Disadvantage: Parameters are not automatically retrieved

Examples in R

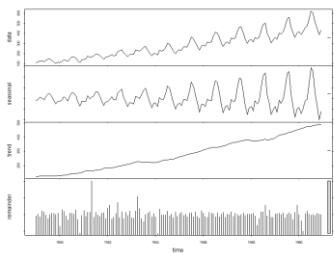
Classical decomposition

```
decomposed <-
decompose(AirPassengers)
```



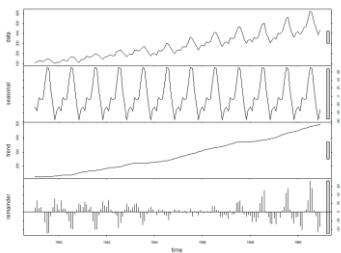
X11

```
library(seasonal)
decomposed <-
seas(AirPassengers, x11
= "")
```



STL

```
decomposed <-
stl(AirPassengers,
s.window = "period")
```



1st row: observed series

2nd row: trend (CD), season (X11/STL)

3rd row: seasonal (CD), trend (X11/STL)

4th row: residuals

Decomp. Features	DEC	X-11	STL
Arbitrary season length	✓	-	✓
Slowly varying season	-	✓	✓
Robustness	-	✓	✓
Endpoints	-	✓	✓

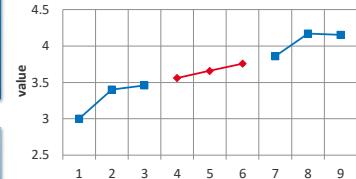
Handling Missing Data: Imputation

Substitute values

- Pick value from the same time series at an earlier time
 - e.g. 9.00am values from Tuesday when 9.00am from Wednesday is missing
- Pick value from other similar time series (similar behavior, similar attributes, similar value range)

Linear interpolation, mean interpolation

- Fill small gaps via linear interpolation or mean of known values
- Not suited for large gaps since fluctuations are eliminated



Aggregate extrapolation

- Extrapolate Aggregate of time series based on portion of monitored series
- Horvitz-Thompson Estimator uses probability of a time series to be missing π and the expectation value μ

$$agg = \sum_i \pi_i^{-1} \cdot \mu_i$$

Ignoring/deleting all sets with incomplete cases does not work for time series

- Missing values have to be randomly distributed over the data set, otherwise deletion introduces bias
- Decrease reliability of analysis by decreasing sample size
- Aggregates would be too low
- Series where forecasts are requested are missing

Imputation

→ Replace missing values with substitute values

Base data

- Hot Deck – Use current data set for the replacement value calculation (most common)
- Cold Deck – Use another data set (e.g. older survey answers) for replacement value calculation

Singular imputation

Use only one technique to calculate replacement values

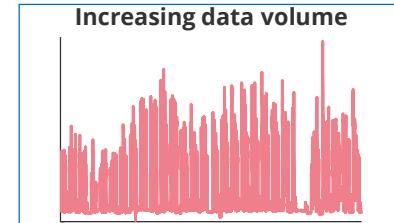
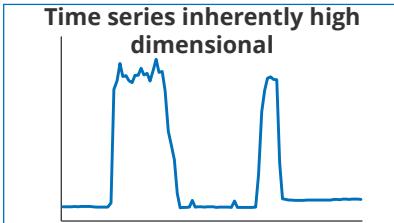
Multiple imputation (Ensemble)

Apply several single imputation techniques to impute the same value

Combine their replacement values in the final result

Data Types
Metrics
Handling of Missing Data
Outlier Detection
Dimensionality Reduction
Value Count Reduction

Curse of Dimensionality



- How to capture the most meaningful information?
- How to prepare this information for data-mining tasks?

Time Series Engineering

Raw-value-based

Shape-based

Feature-based

Model-based



Data Mining

Data Types
Metrics
Handling of Missing Data
Outlier Detection
Dimensionality Reduction
Value Count Reduction

Shape-based Representation

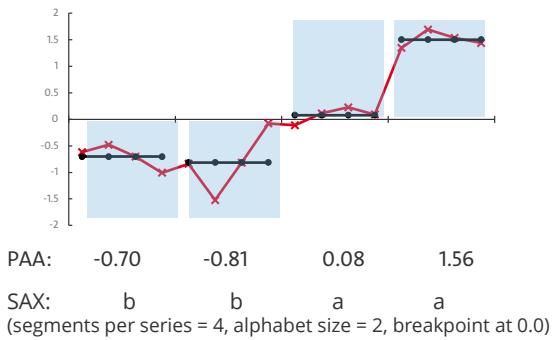
- Transform a series to segments
- Time-dependent representation
 - High compression



Piecewise aggregate approximation (PAA)
→ Represent each segment by its mean



Symbolic aggregate approximation (SAX)
→ Discretize the mean into an alphabet



Different data types enable different distance measures

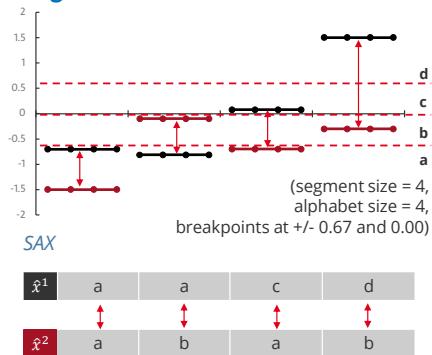
Further reading

Jessica Lin, Eamonn J. Keogh, Stefano Lonardi, and Bill Chiu. A Symbolic Representation of Time Series, with Implications for Streaming Al- gorithms. In Workshop Proc. of SIGMOD, 2003

The SAX Distance

Data Types
Metrics
Handling of Missing Data
Outlier Detection
Dimensionality Reduction
Value Count Reduction

Segments



$$d_{SAX}(\underline{\hat{x}}^i, \underline{\hat{x}}^j) = \sqrt{T/W} \cdot \sqrt{\sum_{w=1}^W \text{cell}(\hat{x}_w^i, \hat{x}_w^j)^2}$$

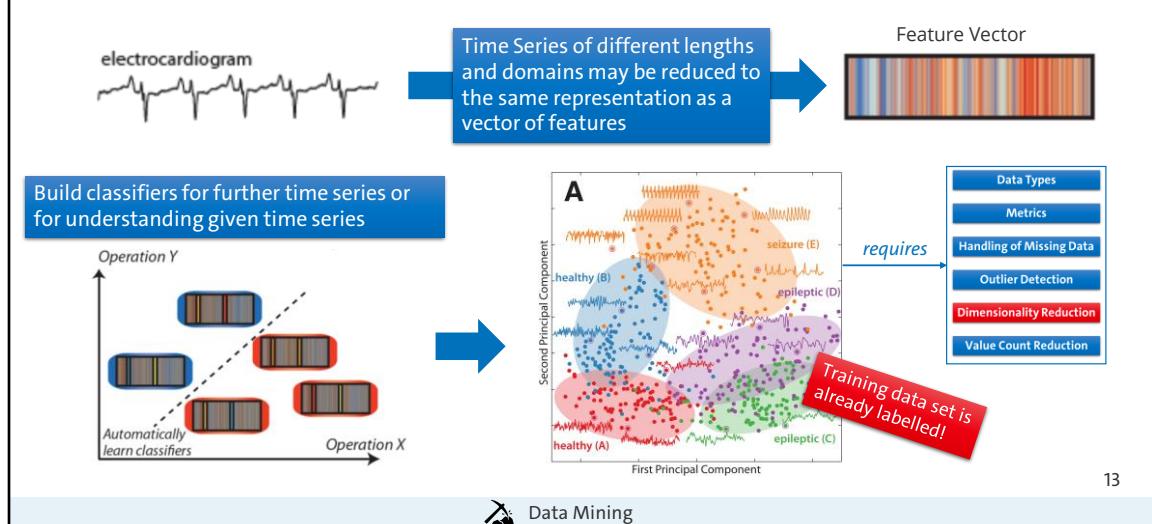
cell returns the minimum distance of two symbols:

	a	b	c	d
a	0	0	0.67	1.34
b	0	0	0	0.67
c	0.67	0	0	0
d	1.34	0.67	0	0

- Same and neighbouring symbols = 0
- T – length of time series
- W – length of string/number of segments
- PAA distance measure

$$d_{PAA}(\underline{\bar{x}}^i, \underline{\bar{x}}^j) = \sqrt{T/W} \cdot \sqrt{\sum_{w=1}^W (\bar{x}_w^i, \bar{x}_w^j)^2}$$

Time Series Classification



13



Represent time series by their features

- Features cover large time series domains
- Time Series of different lengths and domains may be reduced to the same representation as a vector of features
- Fulcher reports ~9000 features from the literature

Classification

- Select interesting features with high classification performance
- Build classifiers for further time series or for understanding given time series

Advantages

- Gain insights into differences between classes of labeled time series datasets

Case Study: EEG Recordings

- EEG data (time series) from healthy patients (A, B), epileptic patients (C, D), and patients with epileptic seizure (E)
- Build classifier that differentiates between groups A and E

Results

- Features are reduced to 2 with Principal Component Analysis
- Groups A and E can be well discriminated
- Feature vectors are as performant as other analytical, but more complex time

series transformations

- Support vector machine
- Discrete wavelet transform

Further Reading

Fulcher, B. D., Little, M. A., & Jones, N. S. (2013). Highly comparative time-series analysis: the empirical structure of time series and their methods.

PCA

Standardize all values per dimension
 $\text{value-mean}/\text{standard deviation}$

$$\text{std} \begin{bmatrix} \text{blue} \\ \text{red} \\ \text{green} \end{bmatrix} = \begin{bmatrix} \text{blue} \\ \text{red} \\ \text{green} \end{bmatrix}$$

Compute the covariance matrix

$$\text{cov}(x,y) = (\sum(x - \text{mean}(x))(y - \text{mean}(y))) / \text{number of data points}$$

$$\text{cov} \begin{bmatrix} \text{blue} \\ \text{red} \\ \text{green} \end{bmatrix} = \begin{bmatrix} \text{blue} \\ \text{red} \\ \text{green} \end{bmatrix}$$

Standardized data set * Feature vector
= New Dataset (reduced to x dimensions)

$$\begin{bmatrix} \text{blue} \\ \text{red} \\ \text{green} \end{bmatrix} * \begin{bmatrix} \text{blue} \\ \text{red} \\ \text{green} \end{bmatrix} = \begin{bmatrix} \text{blue} \end{bmatrix}$$

Compute Eigen values and Eigen vectors of the covariance matrix

$$\begin{bmatrix} \text{blue} \\ \text{red} \\ \text{green} \end{bmatrix} v = \lambda v \rightarrow \begin{bmatrix} \lambda \\ \text{blue} \\ \text{red} \\ \text{green} \end{bmatrix} v$$

Sort elements of Eigenvectors in descending order and pick those that belong to top x Eigen values
→ Feature Vectors

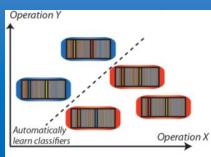
↓ SORT ↓

PCA = Principal Component Analysis
Method for reducing dimensions by merging them

Data Mining Applications

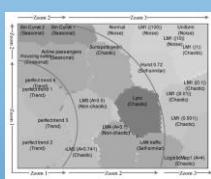
Classification

- Detect characteristics that describe different classes, such that objects can be assigned to classes
- Find a mapping $f: D \rightarrow C$ that assigns objects to classes



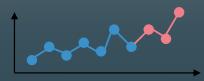
Clustering

- Automatic identification of a finite set of categories, classes, or groups (clusters) in the given data
- Data points are grouped according to their inherent structure



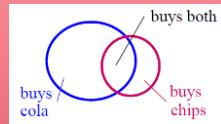
Forecasting (time series)

- Fit a model to a given time series and other influences
- Extrapolate series for prediction



Association Rules/Dependency Mining

- Prediction of events commonly occurring together, e.g. which items are often purchased together
- Not applicable to time series



Classification → Find (and use) the classifiers

Classes are predefined → No unsupervised learning

No object belongs to several classes

Given

- Set of objects (database) $D = \{o_1, o_2, \dots, o_m\}$ where each object o_i corresponds to a k -dimensional vector $\langle o_{i,1}, \dots, o_{i,k} \rangle$
- Set of classes $C = \{c_1, c_2, \dots, c_n\}$ (usually: $|D| \gg |C|$)
- Set of labelled training objects $T \subset D$

Goal

- Find mapping f that assigns objects to classes

Clustering → Find the classes/groups/categories

- Procedure: Grouping of data points according to their inherent structure
 - Points within the same cluster should be as similar as possible
 - Points in different clusters should be as dissimilar as possible
 - Learning without teacher
- Clustering approaches: partitioning, hierarchical, incremental, ...
- Problem: What is the optimal clustering? What is the meaning of “optimal”?

Dependency Mining

- Association rules: Rules of the form $a \wedge b \wedge \dots \wedge c \rightarrow d \wedge e$
 - Example: $\text{buys cola} \rightarrow \text{buys chips}$
- Challenge: Finding good combinations of premises and conclusions is a combinatorial problem

Classification Methods

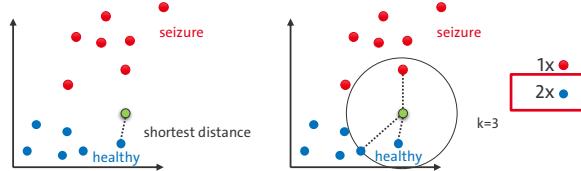
kNN

Threshold based

Decision trees

k-Nearest Neighbors

→ Requires training data set and distance function, e.g. Euclidean distance



- Training data set is only stored → "lazy learning"
- No generalization of the available training data
- Classification effort grows linearly with amount of training data → One distance computation per training object
- Optimization possible, e.g. via filtering, pruning

16



Nearest Neighbor

Direct approach: training objects are

- directly stored in the classifier and
- used for classification

Given:

- Training data set $T = \{o_1, o_2, \dots, o_r\}$ with class labels $c_t: T \rightarrow C$
- Object distance function d

Nearest neighbor:

- The class of an object o is set to the class label of its nearest training object o , i.e.

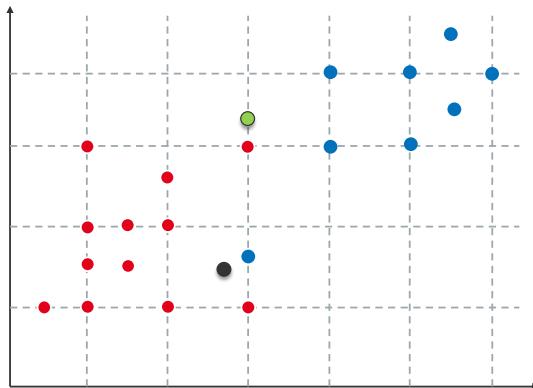
$$c(o) = c_t(o^*) \text{ where } o = \arg \min_{o' \in T} d(o, o')$$

(here we assume that the distance $d(o, o')$ is different for every $o' \in T$)

kNN

- Determine the set N of the k nearest neighbors of o in T
- Choose the class with the maximum number of training objects in N , i.e.
 $c(o) = \arg \max_{c \in C} |\{o' | o' \in N, c_t(o') = c\}|$
- More robust against singular data points
- But more expensive

Exercise kNN



- Which cluster(s) do the green and black objects belong to if $k = 1, 2, 3$, or 4 ?
- Which problems can occur?
- How can we solve them?

Classification Methods

kNN

Threshold based

Decision trees

Simple generalizing model for classification with two classes

Threshold divides the data space into two subspaces along a single dimension

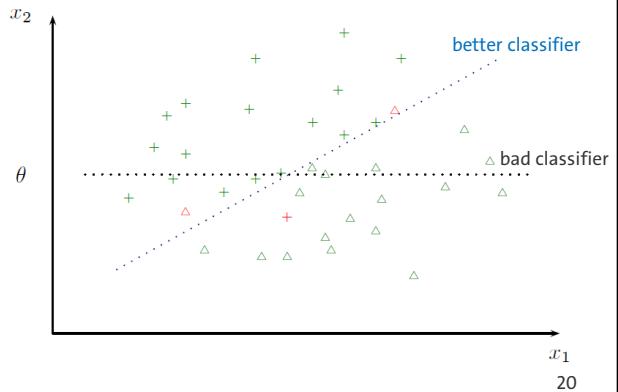
$$c(o_i) = \begin{cases} c_1, & o_{i,j} > \theta \\ c_2, & \text{else} \end{cases}$$

→ Analogue separation criteria for non-numeric data

Optimal threshold: Minimizes classification error for training objects

→ For numeric samples: minimal total distance (sum) of misclassified samples:

$$\theta = \arg \min_{\theta} \sum_{o_i \in T, c(o_i) \neq c_r(o_i)} |o_{i,j} - \theta|$$



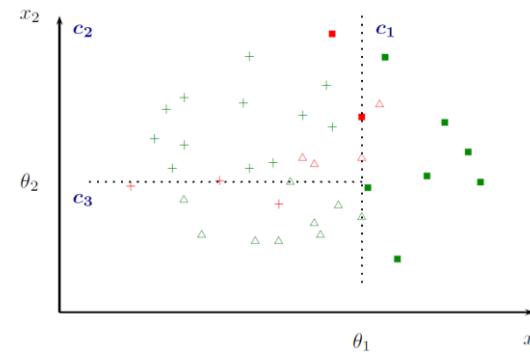
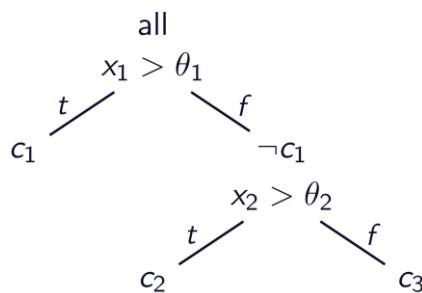
Classification Methods

kNN

Threshold based

Decision trees

Decomposition of threshold-based classifier into sequence of sub-decisions



21

Data Mining

- Multi-branch splits possible
- Each leaf node represents a class $c \in C$
- Finding the optimal decision tree is NP complete
→ Deterministic (non-backtracking), greedy algorithms

Classification Methods

kNN

Threshold based

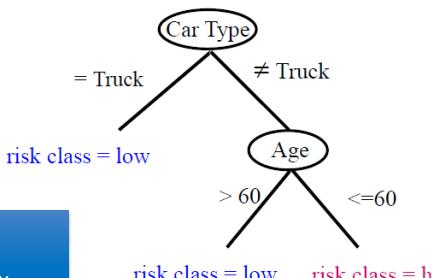
Decision trees

Decomposition of threshold-based classifier into sequence of sub-decisions

ID	Age	Car Type	Risk
1	23	Family	high
2	17	Sport	high
3	43	Sport	high
4	68	Family	low
5	32	Truck	low

Representation in the form of rules

If (Car Type = Truck) then risk class = low
 If (Car Type ≠ Truck AND Age > 60) then risk class = low
 If (Car Type ≠ Truck AND Age ≤ 60) then risk class = high



22

Data Mining

- Usage of the decision tree to make predictions:
 - Top-down traversing of the tree from the root to one of the leaf nodes
 - Assignment of the object to the class of the resultant leaf node

Construction of a Decision Tree

Basic algorithm:

- Initially, all training objects belong to the root
- Selection of the next attribute (split strategy), e.g. by maximization of the information gain
- Partitioning of the training objects with the selected split attribute
- Algorithm is applied to each partition recursively

Stop criterion:

- No further split attributes
- All training objects of a node belong to the same class

Types of splits:

- Categorical attribute: Split condition of the form “attribute = a” or “attribute ∈ set” (many possible subsets)
- Numerical attribute: Split condition of the form “attribute < a” (many possible split points)

- Example: ID3 - split along a dimension as to maximize information gain
- Decision rules can be extracted from a decision tree
 - IF part: combine all tests on the path from the root node to the leaf node
 - THEN part: the final classification
- Enables a simple extraction of ‘real’ knowledge from the learned classifier

Classification Methods

kNN

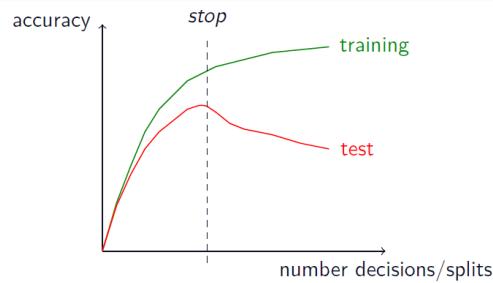
Threshold based

Decision trees

Decomposition of threshold-based classifier into sequence of sub-decisions

The Problem of overfitting

- Splitting until no object is misclassified usually means to adapt the classifier too much to the training data
- Cut-off-criterion or post-pruning required



23

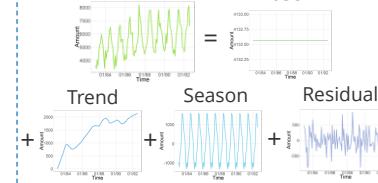


Data Mining

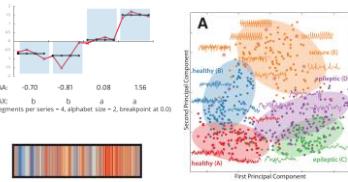
Summary

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

Time Series & TS Decomposition Base

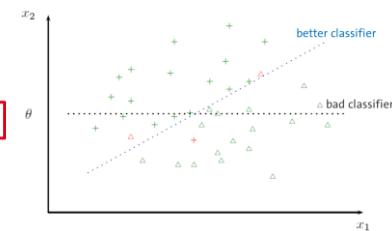


Data Preprocessing for Time Series

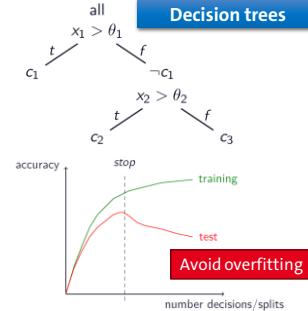


Data Mining Applications: Classification

Threshold based



Decision trees



Evaluation of Classifications

Basic idea: Reserve some labelled data for evaluation

- Held out data**
- 30-50% of the data is reserved for testing
 - Error estimation depends on partitioning → repeat the measurement with different partitionings and average the results

- Leave one out**
- Use $n - 1$ samples for training and evaluate on the n -th one
 - Repeat with all n samples
 - Extremely expensive

- N-fold cross validation**
- Combines hold-out and leave-one-out
 - Divide data set into p partitions
 - Use $p - 1$ partitions for training; evaluate on the remaining one
 - Repeat with different partitionings

Error rate

$$e = \frac{|M|}{|S|}$$

Misclassified test objects
Test set

Accuracy

$$a = 1 - e = \frac{|S| - |M|}{|S|}$$

Contrastive analysis: comparison with a baseline case

- Absolute improvement/degradation:
$$\Delta_{\text{abs}} a = a_n - a_{n-1}$$

$$\Delta_{\text{abs}} e = e_n - e_{n-1}$$
- Relative improvement/degradation:
$$\Delta_{\text{rel}} a = (a_n - a_{n-1}) / a_{n-1}$$

$$\Delta_{\text{rel}} e = (e_n - e_{n-1}) / e_{n-1}$$
- Essential difference between absolute and relative improvement/degradation
- Often used manipulation:
 - Relative improvement for positive effects
 - Absolute improvement for negative (adverse) effects
- There are more error measures, e.g. weighted error measures

Further important quality aspects

- Compactness, e.g. size of the decision tree (size determines classification time)
- Interpretability (how many insights can be imparted by the classifier to the user?)
- Efficiency of the construction process and application of the constructed

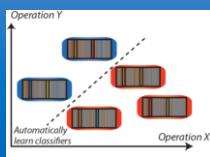
classifier

- Scalability with respect to large data sets (construction as well as application)
- Robustness against noise and missing values

Data Mining Applications

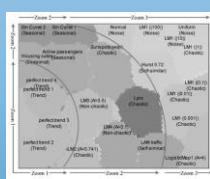
Classification

- Detect characteristics that describe different classes, such that objects can be assigned to classes
- Find a mapping $f: D \rightarrow C$ that assigns objects to classes



Clustering

- Automatic identification of a finite set of categories, classes, or groups (clusters) in the given data
- Data points are grouped according to their inherent structure



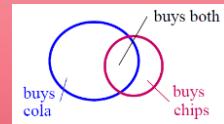
Forecasting (time series)

- Fit a model to a given time series and other influences
- Extrapolate series for prediction



Association Rules/Dependency Mining

- Prediction of events commonly occurring together, e.g. which items are often purchased together
- Not applicable to time series



- Definition of the optimal clustering depends on
 - Application domain (i.e. structure/semantics of the data, meaning of distance)
 - Data mining target (i.e. object correlations that should be detected)
- Computing the optimal clustering is computationally infeasible → greedy, sub-optimal approaches
- Different clustering algorithms might lead to different clustering results

Clustering of Time Series

Goal

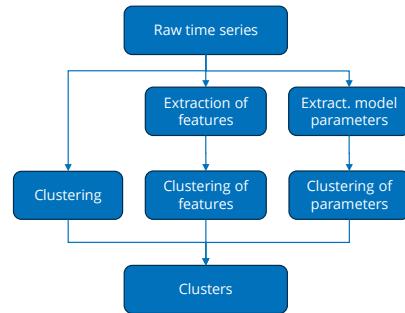
Partition a set of time series into groups in such a way that homogenous time series are grouped together

Challenges

- Often too large to fit in main memory
- Highly dimensional and it is difficult for clustering algorithms to handle them
- Homogeneity strongly depends on the selected similarity measure

Applications

- Which patterns appear frequently?
- Which patterns appear surprisingly?

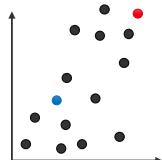


5

- Raw-value-based approach
 - Raw time series is matched as well as possible
 - Usage of stretching and contraction
- Feature-based approach
 - Reduce time series to feature vector
 - Similarity of vector, usually by Euclidean distance
- Model-based approach
 - Fit a model to each given time series
 - Time series distances are calculated based on model parameters

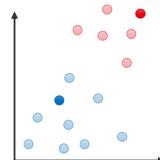
Clustering Methods

K-Means



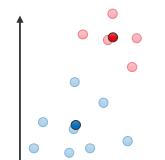
Step 1: Randomly choose k centroids

Canopy Clustering



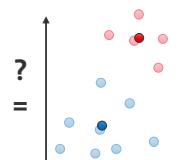
Step 2: Assign all elements to the cluster of the nearest centroid

Hierarchical Clustering



Step 3: Compute centroid of each cluster, e.g. arithmetic mean

Incremental Clustering



Step 4: Does the new centroid change clustering?
Yes: Done
No: Go to step 2

Other possible termination criterions:
• Number of iterations
• Clustering changes only minimally
• ...

6

Data Mining

- Also works with 1 dimension or more than 2 dimensions (if an according distance measure exists)

Problems:

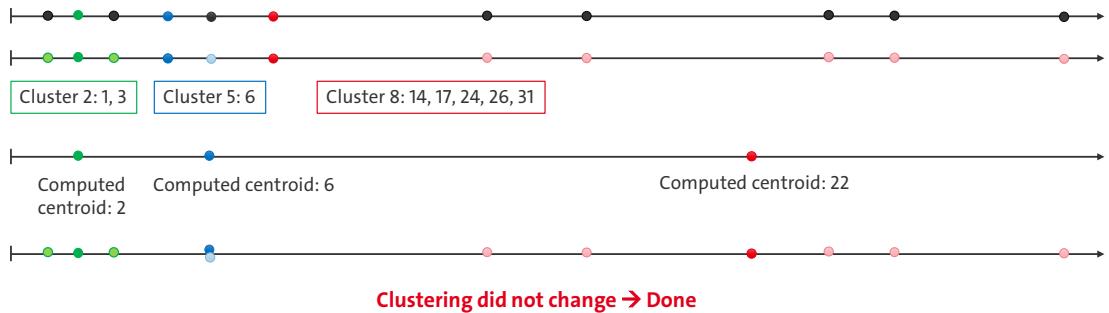
- Converges to local minima, i.e. the clustering is not necessarily optimal
- Workaround: Start the algorithm several times
- Relatively high effort to compute distances and cluster centroids

Example K-Means

1, 3, 6, 14, 17, 24, 26, 31

K=3

Randomly chosen centroids: 2, 5, 8



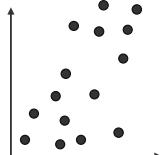
Exercise K-Means

- Clustering of the numbers 1, 3, 6, 14, 17, 24, 26, 31
- k=3
- Centroids randomly chosen in step 1: 10, 21, 29

8

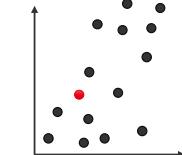
Clustering Methods

K-Means



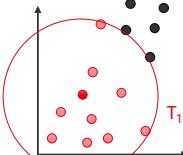
Step 1: Initialize candidate list for canopy centroids with all objects

Canopy Clustering



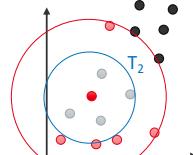
Step 2: Canopy centroid Z is selected randomly from candidate list

Hierarchical Clustering



Step 3: All objects with distance to $Z < T_1 \rightarrow$ assign to canopy Z

Incremental Clustering



Step 4: All objects with distance to $Z < T_2 \rightarrow$ remove from candidate list

Data Mining

10

Construction of overlapping clusters (canopies)

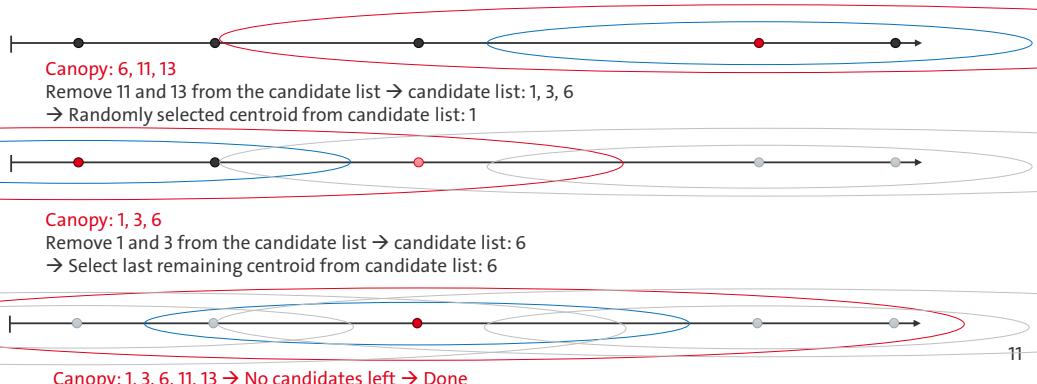
- Non-Partitioning Clustering
- Often used as a first step in a multi-step analysis approach
- Scalable for large data sets
- Can be used for string data
- Requires distance function and 2 thresholds T_1 and T_2 ($T_1 > T_2$)

Canopy Clustering Example

Clustering of the numbers 1, 3, 6, 11, 13

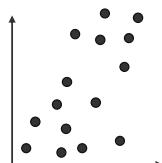
$T_1 = 8$, $T_2 = 4$

Randomly selected centroid: 11



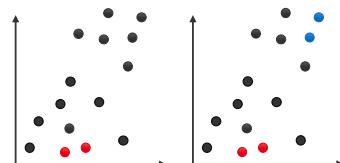
Clustering Methods

K-Means



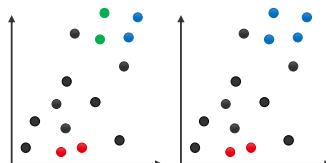
Step 1: Each data point is an individual cluster

Canopy Clustering



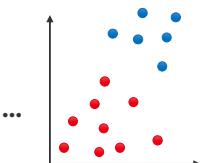
Step 2: Merge least distant clusters into a new cluster

Hierarchical Clustering



Step 3: Repeat

Incremental Clustering



Step 4: Stop if all clusters have been merged or the minimal cluster distance exceeds a predefined threshold

Clustering Methods

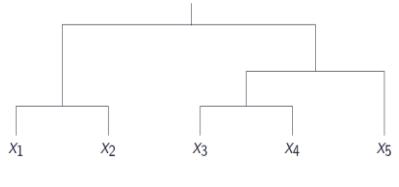
K-Means

Canopy Clustering

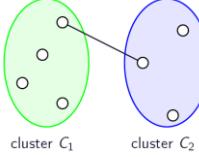
Hierarchical Clustering

Incremental Clustering

Results can be displayed as a dendrogram

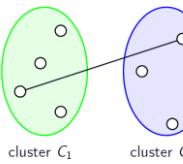


Distance measures for clusters

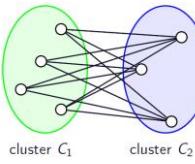


Single link

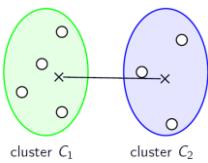
Minimal distance between two data points



Complete link
Maximal distance between two data points



Average link
Average distance between two data points



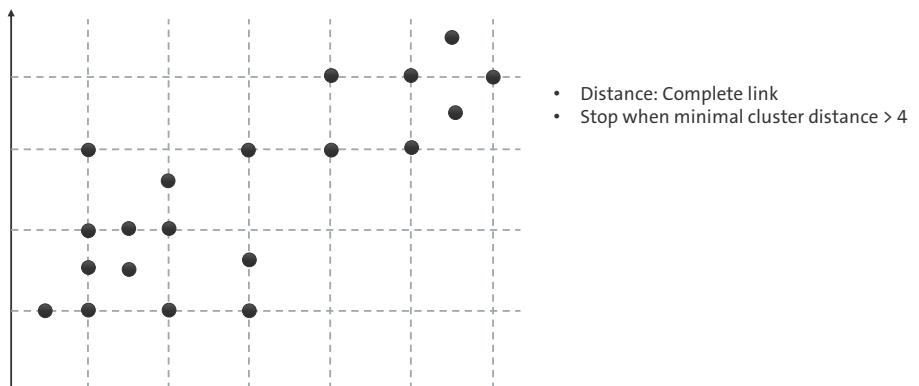
Canonical entity
Distance between two cluster representatives (e.g. the centroids)



Data Mining

13

Exercise: Hierarchical Clustering



14

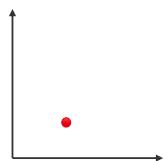
Clustering Methods

K-Means

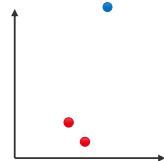
Canopy Clustering

Hierarchical Clustering

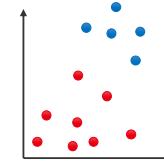
Incremental Clustering



Step 1: Assign first data point(s) to the first cluster(s)



Step 2: Assign new data points to an existing cluster or create a new cluster



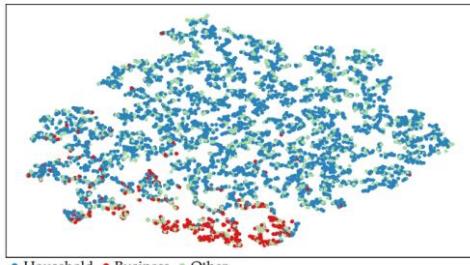
Step 3: Recompute cluster description and continue until all data points were processed

- Is a concept to enable the use of clustering algorithms, e.g. k-means, with large data sets, i.e. data sets which do not fit into main memory
- Problem: Result depends on the order in which data points are processed

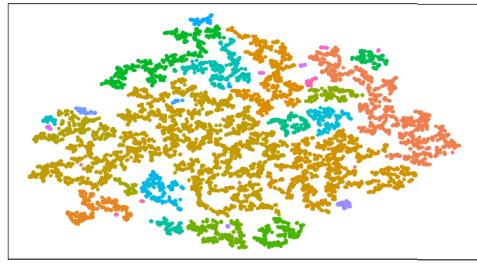
Cluster description consists at least of

- Centroid
- Number of data points in the cluster
- “Radius” of the cluster (e.g. the maximal distance of a point to the centroid)

Clustering Time Series



(a) Class-based



(b) Feature-based

[1]

Metering Dataset

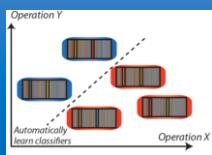
[1] Source: Feature-based Time Series Analytics, Lars Kegel, 2020

Class-based: Dataset provides class labels

Data Mining Applications

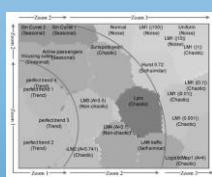
Classification

- Detect characteristics that describe different classes, such that objects can be assigned to classes
- Find a mapping $f: D \rightarrow C$ that assigns objects to classes



Clustering

- Automatic identification of a finite set of categories, classes, or groups (clusters) in the given data
- Data points are grouped according to their inherent structure



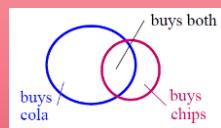
Forecasting (time series)

- Fit a model to a given time series and other influences
- Extrapolate series for prediction



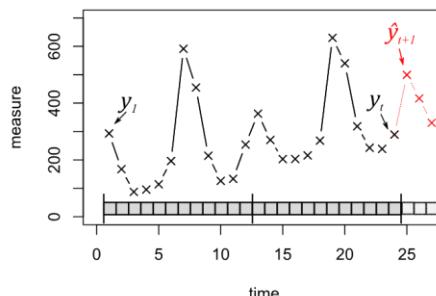
Association Rules/Dependency Mining

- Prediction of events commonly occurring together, e.g. which items are often purchased together
- Not applicable to time series

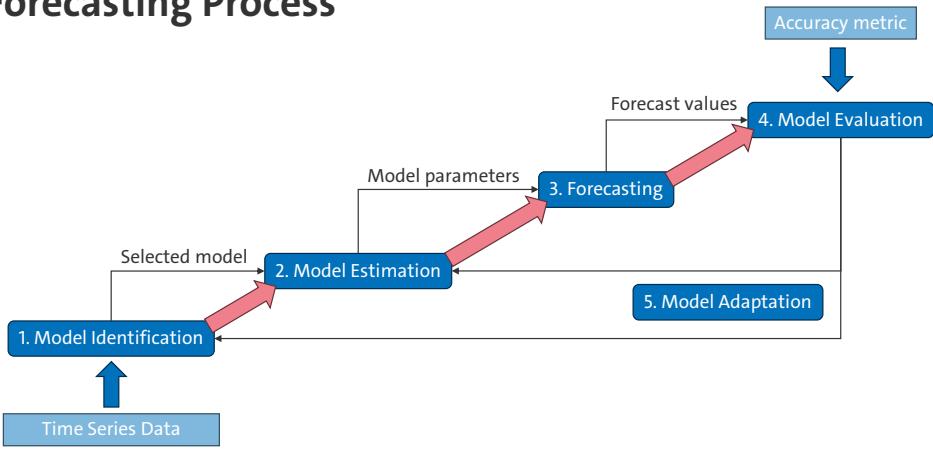


Forecast Models

- One model represents one time series
- Trend (long-term changes)
- Season (regular reoccurring changes)
- Accuracy is the only requirement

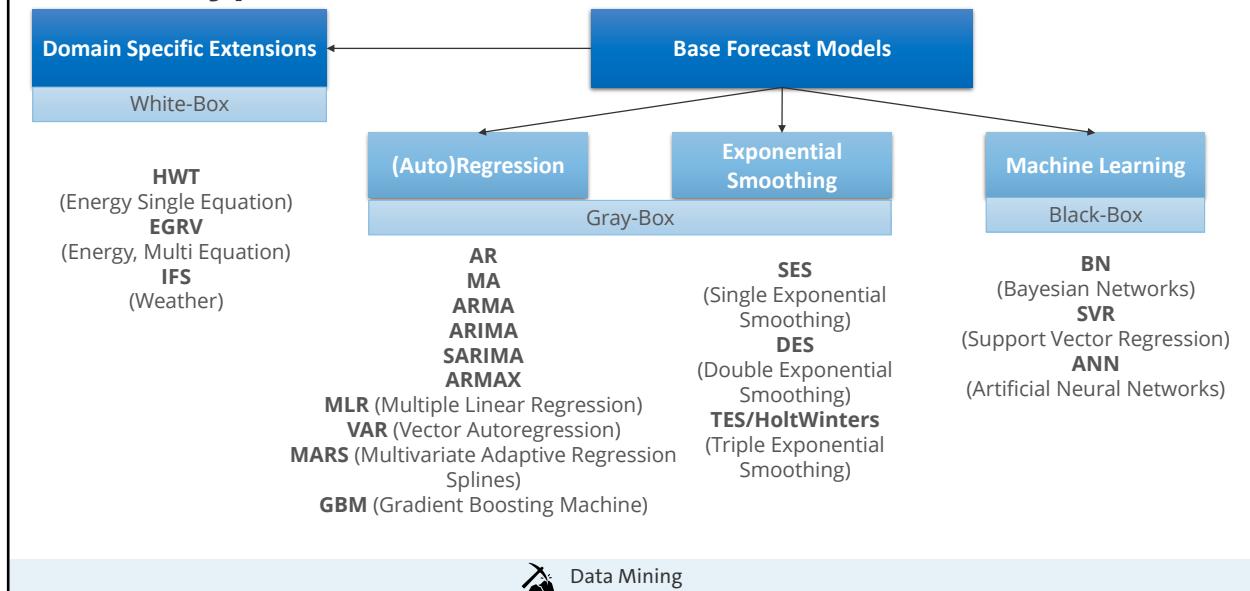


The Forecasting Process



- 1. Model Identification** – Choose the optimal model type
- 2. Model Estimation** – Instantiation of the model by training its model parameters
- 3. Forecasting** – Usage of the model to calculate the next time series values
- 4. Model Evaluation** – Compare the model's results with real time series values using an error measure
- 5. Model Adaption** – Adaption of the model parameters or the model type

Model Types



Exponential Smoothing

Weight time series values with smoothing factor

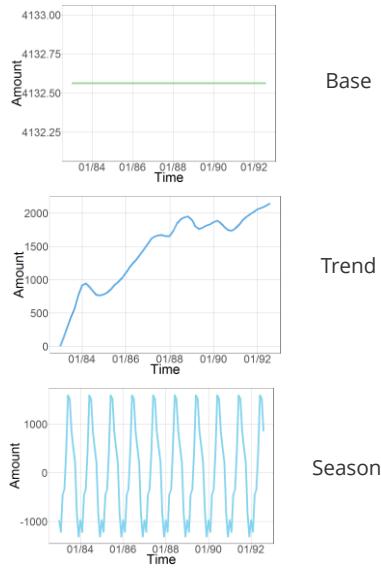
- Declines exponentially for older time series values

Model variants

- Simple Exponential Smoothing (SESM): Base
- Double Exponential Smoothing (DESM): Base, Trend
- Triple Exponential Smoothing (TESM): Base, Trend, Season Holt-Winters-Model

Example Base:

$$\begin{aligned}x_1^* &= x_1 \\x_t^* &= \alpha \cdot x_t + (1 - \alpha) \cdot x_{t-1}^* \\x_{t+1}^* &= x_t^*\end{aligned}$$



Further Reading

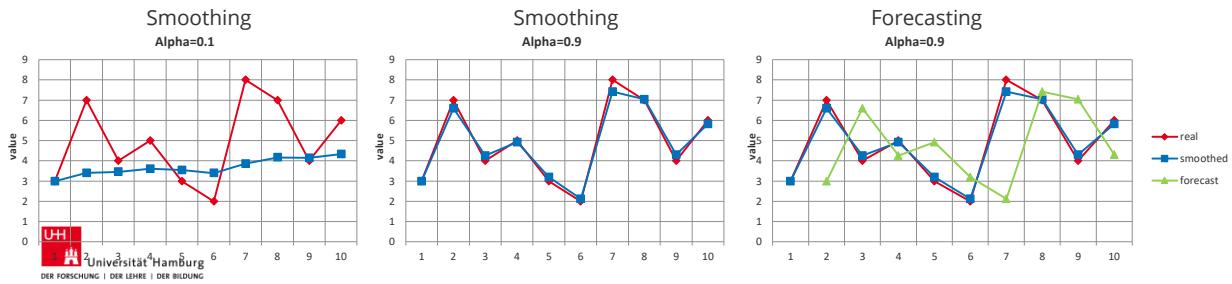
Holt, C. C. (2004). Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 20(1).

Exponential Smoothing

Example Single Exponential Smoothing

$$\begin{array}{l} \text{Base: } x_1^* = x_1 \\ x_t^* = \alpha \cdot x_t + (1 - \alpha) \cdot x_{t-1}^* \quad \text{Smoothing} \\ \hline \hat{x}_{t+1} = x_t^* \quad \text{Forecasting} \end{array}$$

t	1	2	3	4	5	6	7	8	9	10
x_t	3	7	4	5	3	2	8	7	4	6
x_t^* ($\alpha=0.9$)	3	6.6	4.26	4.926	3.19	2.12	7.412	7.041	4.304	5.83

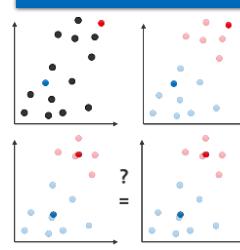


- Often bad results for data with a trend → double/triple exponential smoothing
- Double exponential smoothing: Include the difference between data points and smooth it
- Triple exponential smoothing: Include and smooth seasonal correction factors, different smoothing factors for base, trend, and season possible

Summary

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

K-Means



Evaluation of Classifications

Basic idea: Reserve some labelled data for evaluation

Error rate

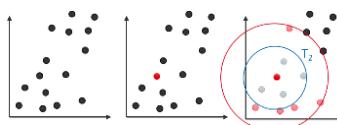
$$e = \frac{|M|}{|S|} = \frac{\text{Misclassified test objects}}{\text{Test set}}$$

Accuracy

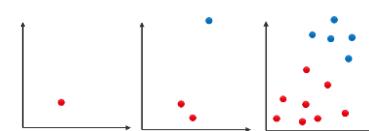
$$a = 1 - e = \frac{|S| - |M|}{|S|}$$

Clustering

Canopy Clustering



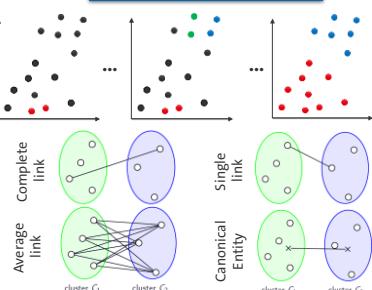
Incremental Clustering



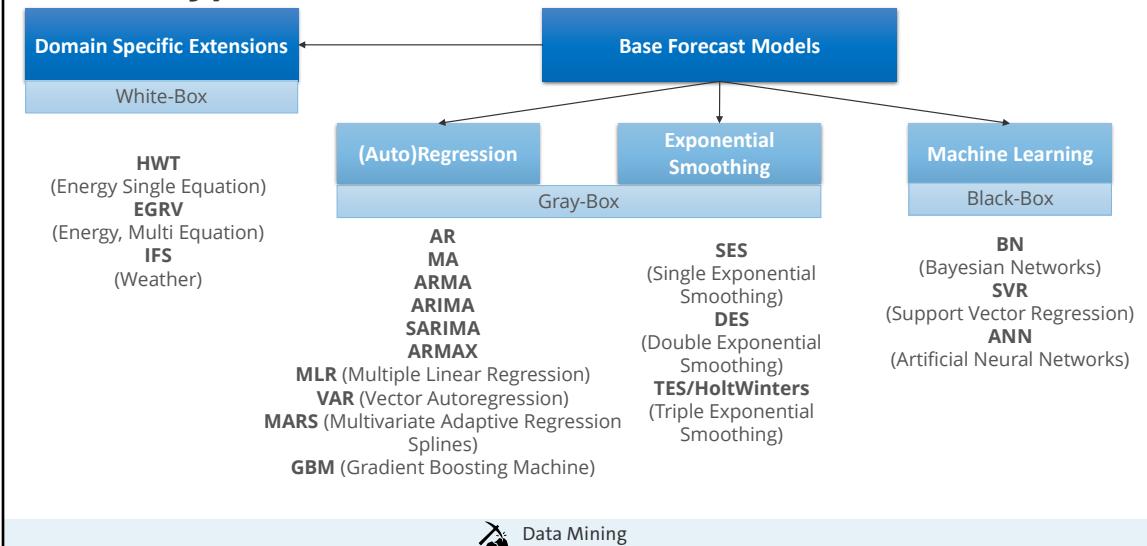
Time Series Forecasting (part 1)



Hierarchical Clustering



Model Types



 Data Mining

The ARMA Models

$$\hat{x}_t = c + \sum_{i=1}^p \phi_i \cdot x_{t-i}$$
$$\hat{x}_t = \sum_{i=1}^p \phi_i \cdot x_{t-i} + \sum_{i=1}^q \theta_i \cdot \varepsilon_{t-i} + c$$
$$\hat{x}_t = \sum_{i=1}^q \theta_i \cdot \varepsilon_{t-i}$$



Exogenous Influence → ARIMAX Seasonality → SARIMA

$$\hat{x}_t = \sum_{i=1}^p \phi_i \cdot x_{t-i} + \sum_{i=1}^q \theta_i \cdot \varepsilon_{t-i} + c + \sum_{i=1}^b \gamma_i \cdot W_i$$

4

ARMA Models

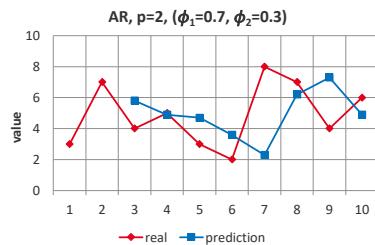
Box and Jenkins Methodology

- Modeling time series with AutoRegression

Classification and Model Hierarchy

$$\text{AR}(p): \hat{x}_t = c + \sum_{i=1}^p \phi_i \cdot x_{t-i}$$

AutoRegression

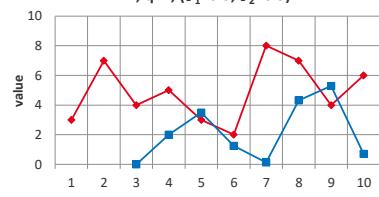


$$\text{MA}(q): \hat{x}_t = \sum_{i=1}^q \theta_i \cdot \varepsilon_{t-i}$$

Moving Average

$$\varepsilon_0, \dots, \varepsilon_q = 0, \varepsilon_i = x_{t-i} - \hat{x}_{t-i}$$

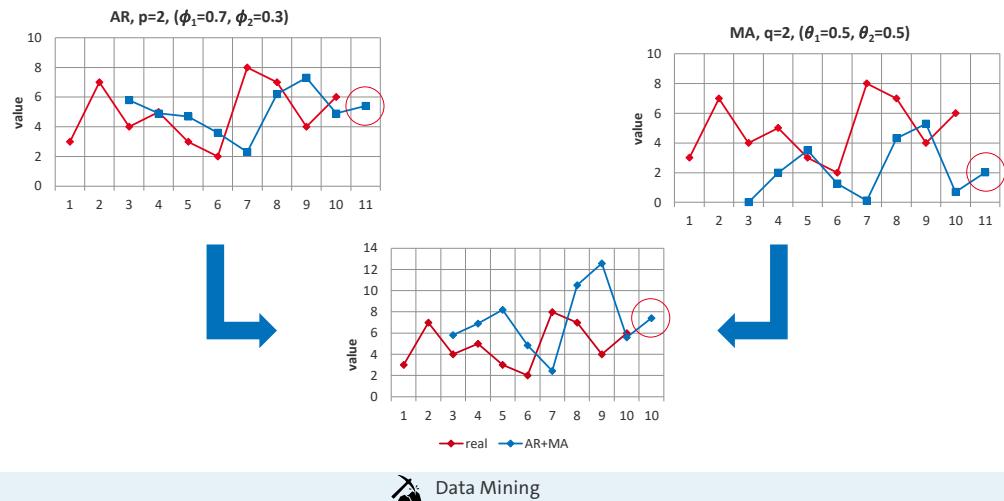
$$\text{MA, } q=2, (\theta_1=0.5, \theta_2=0.5)$$



Here: $c=0$

Box and Jenkins do not assume trend and season components, but regard time series modelling as a stochastic process.

AR+MA



Exercise ARMA

$$\phi_1=0.7, \phi_2=0.3$$

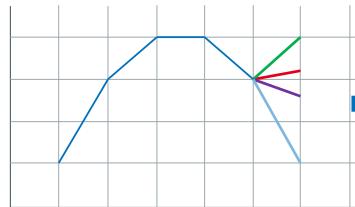
$$\theta_1=0.5, \theta_2=0.1$$

t	1	2	3	4	5	6
x _t	1	3	4	4	3	?
AR						
MA						
ε_i						

$$\hat{x}_t = \sum_{i=1}^p \phi_i \cdot x_{t-i}$$

$$\hat{x}_t = \sum_{i=1}^q \theta_i \cdot \varepsilon_{t-i}$$

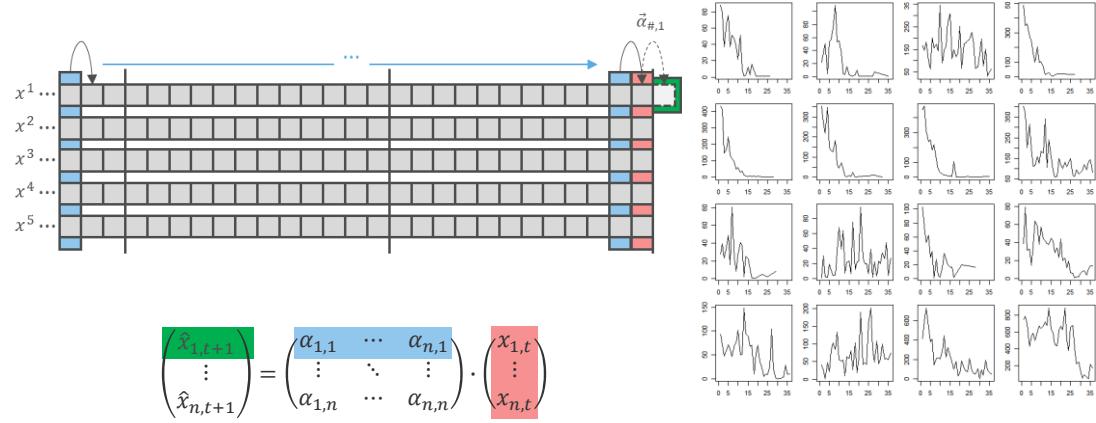
$$\varepsilon_0, \dots, \varepsilon_q = 0, \varepsilon_i = x_{t-i} - \hat{x}_{t-i}$$



Which one is the ARMA prediction?

7

Vector Autoregression (VAR)



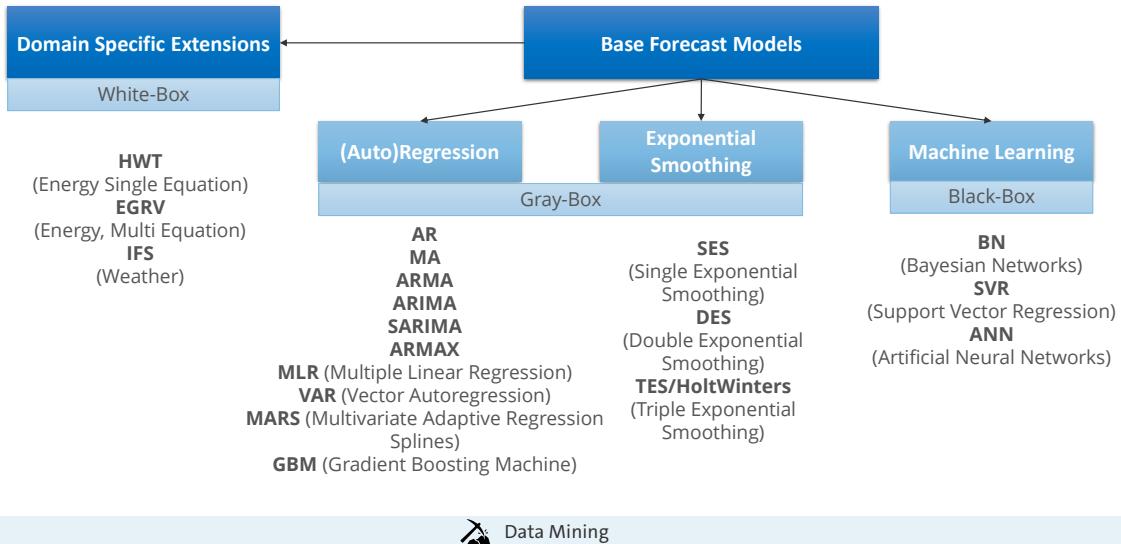
9

- Multivariate Modeling Technique
 - Based on the assumption, that time series from the same domain influence each other
 - Capable of compensating unexplainable behavior of individual time series
 - Can be extended by external influences
- Individual Model Optimization
 - In practice every line in the parameter matrix is implemented as an individual optimization process

Further Reading

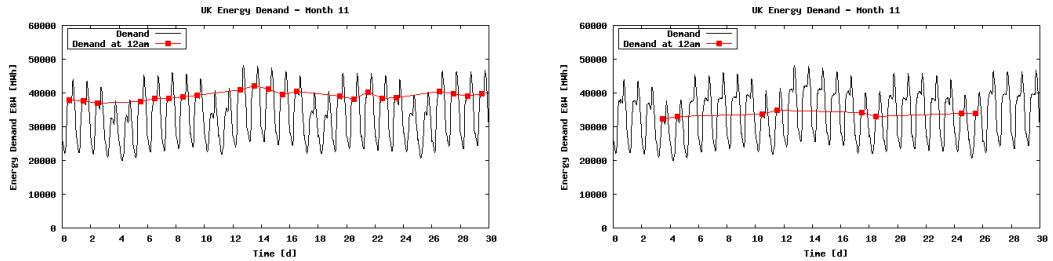
Riise, T., & Tjøstheim, D. (1984). Theory and practice of multivariate arma forecasting. *Journal of Forecasting*, 3(3).

Model Types



 Data Mining

EGRV Model (Engle, Granger, Ramanathan, and Vahid-Araghi)



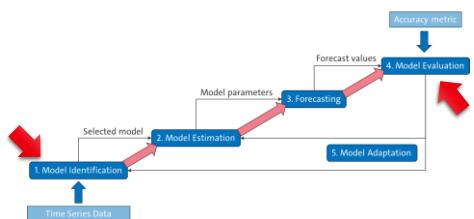
Data Mining

Multi Equation Model

- One model for each hour (half hour, etc.) of a day (separation in weekdays/weekends)
- White-Box model tailor-made for energy demand
- Decomposition leads to almost constant time series that are easy to predict
- Basic idea works for all types of time series that follow certain pattern

Model Identification

Method	Error ϵ	AIC
ARIMA	23.47	25.72
HoltWinters	18.38	20.43
VAR		
MARS		
⋮		



Akaike Information Criterion (AIC)

$$AIC = 2k - 2 \ln(L)$$

$$k = \# \text{parameters}, \ln(L) \approx - \sum_{t=k}^T \epsilon_t^2$$

→ If two models have an equally good fit, prefer the less complex model

(Semi)Automatic Model Identification

- Test for every available model, how it performs on a test section of the current time series
- Use the error or more complex measures like AIC and BIC to find the optimal model

Akaike Information Criterion (AIC)

- A more sophisticated way to decide which is the optimal model
- Use the model with the lowest AIC
- Use the fit of the model to the training data and the model complexity
- L → Maximum likelihood

Model Estimation

Problem

Find the model parameters that minimize the error on the training data

Example

Forecast Model Type AR(2):

$$\hat{x}_t = \phi_1 \cdot x_{t-1} + \phi_2 \cdot x_{t-2}$$

Error Metric: MSE

$$\frac{1}{n} \sum_{i=1}^n (x_i - \hat{x}_i)^2$$

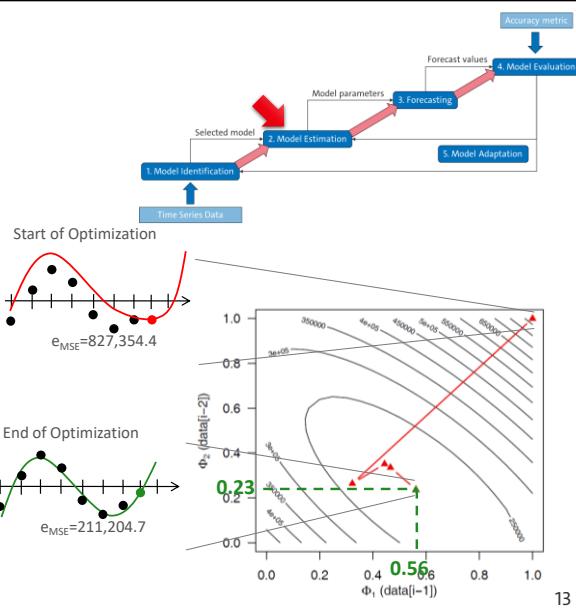
Parameter Estimator

L-BFGS-B



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG



Data Mining

13

AR(2): Auto Regression with p=2

MSE: Mean Squared Error

L-BFGS-B: Limited memory Broyden–Fletcher–Goldfarb–Shanno with box constraints

Original BFGS was published independently by each of the 4 authors:

Broyden: <https://doi.org/10.1093/imamat%2F6.1.76>

Fletcher: <https://doi.org/10.1093/comjnl%2F13.3.317>

Goldfarb: <https://doi.org/10.1090/S0025-5718-1970-0258249-6>

Shanno: <https://doi.org/10.1090/S0025-5718-1970-0274029-X>

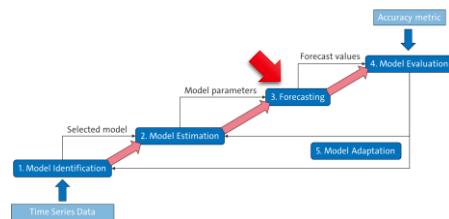
Model Usage

Model based Prediction

- Application of the identified and trained (optimized) model
- Typically very fast compared to the training process

Horizon

- Describes the number of forecasted values
- One step ahead (only one value), long range forecasting (many values, up to several seasons)



Example (Long Range)

- Single Exponential Smoothing, horizon 10 values
- Continue forecast calculation based on already calculated forecast values

Base:

$$\begin{aligned} x_t^* &= x_1 \\ x_t^* &= \alpha \cdot x_t + (1 - \alpha) \cdot x_{t-1}^* \\ \hat{x}_{t+1} &= x_t^* \end{aligned}$$

```
for(i in 1:10)
  x_{t+i-1}^* = alpha * x_{t+i-1} + (1 - alpha) * x_{t-1}^*
  x_{t+i} = x_{t+i-1}^*
```

Maintenance strategies

Error Threshold

- Define error threshold τ
- Update model parameter when Error $> \tau$
- May miss opportunities to improve the model performance or reestimate too often, based on whether τ is too high or too low

Time Threshold

- Reestimate the model parameter after a fixed number of update values
- May reestimate too often and without model improvement or tolerate too high errors within an interval

...as usual a combination works best

- A somewhat higher error threshold
- And a somewhat longer time threshold

Time Series Storage

Relational Storage

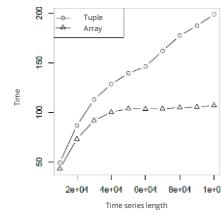
ID	rec_date	sales
1	20.11.2021	3
1	21.11.2021	7
1	22.11.2021	4
1	23.11.2021	5
1	24.11.2021	3
1	25.11.2021	2
1	26.11.2021	8
1	27.11.2021	7
1	28.11.2021	4
1	29.11.2021	6
2	20.11.2021	7
2	21.11.2021	5
2	22.11.2021	8

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}

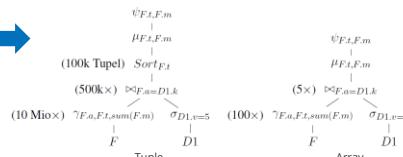
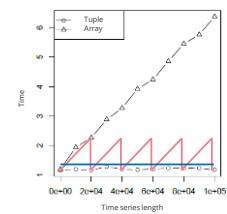
Read performance

- Read entire data set into a forecast model
- Array storage outperforms tuple per value



Insert performance

- Append values to stored time series
- Tuple per value wins, less overhead
 - Counter steer by partitioning
- Ideal case, constant insert time



15

Relational Storage

- One tuple for each individual measure value
- Store alongside with
 - Time/date stamp
 - Time series identifier, e.g.
 - ID
 - Equipment
 - Sensor name

Access

- Usually no order guarantees
- Order by
 - Identifier first
 - Time last

Array Storage

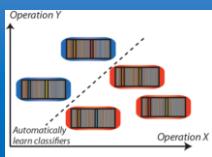
- Store entire time series
 - Time series values are stored in an array
 - Alongside with
 - Time series identifier

- Start
- Time interval between values
- Constraints: Time series must be
 - Complete
 - Equidistant

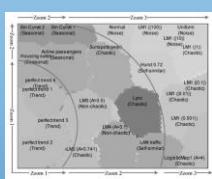
Data Mining Applications

Classification

- Detect characteristics that describe different classes, such that objects can be assigned to classes
 - Find a mapping $f: D \rightarrow C$ that assigns objects to classes



Clustering



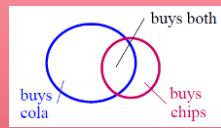
Forecasting (time series)

- Fit a model to a given time series and other influences
 - Extrapolate series for prediction



Association Rules/Dependency Mining

- Prediction of events commonly occurring together, e.g. which items are often purchased together
 - Not applicable to time series



Goal: Prediction of events commonly occurring together

Market basket analysis: Which items are often purchased together

- Placement of items in a store
 - Layout of mail-order catalogues
 - Targeted marketing campaigns

Association rules: Rules of the form $a \wedge b \wedge \dots \wedge c \rightarrow d \wedge e$

Example: buys cola → buys chips

Challenge: Finding good combinations of premises and conclusions is a combinatorial problem

Association Rules

Example (transaction) database

trans-action	item	trans-action	items
001	cola	001	{chips, cola, peanuts} T_k
001	chips	002	{beer, chips, cigarettes}
001	peanuts	003	{beer, chips, cigarettes, cola}
002	beer	004	{beer, cigarettes}
002	chips		
002	cigarettes		
...	...		

$I = \text{SELECT DISTINCT item FROM example_table;}$



D

$I_k, i=2$

$s(\{beer\}) = 3/4$

If ($s_{cut} = 1/2$) $\rightarrow s(\{beer\})$ is a frequent itemset

Definitions

Set of n different items I $I = \{x_1, \dots, x_n\}$

Itemset I_k $I_k \subseteq I$

i -itemset $I_k^i \subseteq I, |I_k^i| = i$

Transaction T_k $T_k \subseteq I$

Database D $D = \{(k, T_k) \mid k = 1, \dots, m\}$

Support of an itemset
= share of transactions
which contain the itemset

$$s(I_i) = \frac{|\{T_k \mid I_i \subseteq T_k\}|}{|D|}$$

Frequent (strong, large)
itemset $s(I_i) \geq s_{cut}$

Data Mining

17

Downward closure:

- Every subset of a frequent itemset is also a frequent itemset
- Every superset of a non-frequent itemset is also a non-frequent itemset

Association Rules

Association Rule

$$X \rightarrow Y \text{ where } X, Y \subseteq I, X \cap Y = \emptyset$$

Support of a rule: Share of transactions which contain both, premise and conclusion of the rule

$$s(X \rightarrow Y) = s(X \cup Y) = \frac{|\{T_k | X \cup Y \subseteq T_k\}|}{|D|} = p(XY)$$

Confidence of a rule: Share of transactions supporting the rule from those supporting the premise

$$c(X \rightarrow Y) = \frac{s(X \cup Y)}{s(X)} = \frac{|\{T_k | X \cup Y \subseteq T_k\}|}{|\{T_k | X \subseteq T_k\}|} = p(Y|X)$$

Lift of a rule: Ratio of the observed support to that expected if X and Y were independent

$$lift(X \rightarrow Y) = \frac{s(X \cup Y)}{s(X) \times s(Y)} = \frac{p(Y|X)}{p(Y)}$$

Strong rule: High Support + High Confidence

trans-	items
action	
001	{chips, cola, peanuts}
002	{beer, chips, cigarettes}
003	{beer, chips, cigarettes, cola}
004	{beer, cigarettes}

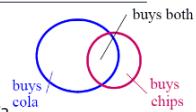
Example:

[buys] chips \rightarrow [buys] cola

$$s(\text{chips} \rightarrow \text{cola}) = 2/4 = 0.5$$

$$c(\text{chips} \rightarrow \text{cola}) = 0.5/(3/4) = 2/3$$

$$lift(\text{chips} \rightarrow \text{cola}) = (2/3)/(0.75 * 0.5) = 1.78$$



Customers who buy chips are 1.78x as likely to also buy cola as compared to independent purchases.



- $lift(X) > 1 \rightarrow$ positive association
- $lift(X) < 1 \rightarrow$ negative association
- $lift(X) = 1 \rightarrow$ items are independent

Detection of strong rules: Two pass algorithm

1. Find frequent (strong, large) itemsets (Apriori algorithm)
 - Necessary to generate rules with strong support
 - Uses the downward closure
 - Itemsets are ordered
2. Use the frequent itemsets to generate association rules
 - Find strong correlations in a frequent itemset

Apriori

I_k^1	#	$s(I_k^1)$
{chips}	3	0.75
{cola}	2	0.5
{peanuts}	1	0.25
{beer}	3	0.75
{cigarettes}	3	0.75

Step 1: Start with all itemsets of size one: I^1
Step 2: Select all items with sufficient support, e.g. where $s(I_k) \geq 0.5$

I_k^2	#	$s(I_k^2)$
{chips, cola}	2	0.5
{beer, chips}	2	0.5
{chips, cigarettes}	2	0.5
{beer, cola}	1	0.25
{cigarettes, cola}	1	0.25
{beer, cigarettes}	3	0.75

Step 3: From the selected itemsets I^{l-1} generate larger itemsets I^l
 → Already blocks some of the non-frequent itemsets, but not all of them

I_k^3	#	$s(I_k^3)$
{chips, cola}	2	0.5
{beer, chips}	2	0.5
{chips, cigarettes}	2	0.5
{beer, cola}	1	0.25
{cigarettes, cola}	1	0.25
{beer, cigarettes}	3	0.75

I_k^3	#	$s(I_k^3)$
{beer, chips, cigar.}	2	0.5
{chips, cigar., cola}	1	0.25

Prune because of non-frequent subset

Step 4: Remove itemsets which still contain a non-frequent subset
 → Cannot have enough support because of downward closure

Step 5: Repeat from step 2 until no further frequent itemsets can be generated

D

trans-	items
action	
001	{chips, cola, peanuts}
002	{beer, chips, cigarettes}
003	{beer, chips, cigarettes, cola}
004	{beer, cigarettes}

Resulting frequent itemsets
 i=3: {beer, chips, cigarettes}
 i=2: {chips, cola}, {beer, chips}, {chips, cigarettes}, {beer, cigarettes}
 i=1: {beer}, {chips}, {cigarettes}, {cola}



Apriori: Finding frequent itemsets of increasing size (itemsets are ordered!)

- Step 4 in the example: no items to prune
- Note: Itemset {beer, chips, cola} is not constructed because we only combine sets of I_k^2 that have the first element in common
- However, this set cannot be frequent because otherwise the set {beer, cola} would be in I_k^2 (and then we would have constructed {beer, chips, cola})

Exercise Apriori

$k = T_k$
1 {butter, bread}
2 {butter, bread, cheese, wine}
3 {bread, soda}
4 {cheese, pencils, pasta}
5 {cheese, pasta, wine}

$$s_{cut} = 0.4$$

Generation of Strong Association Rules

- For all frequent itemsets I_j with $|I_j| \geq 2$ determine all non-empty subsets I_k for which
$$c = \frac{s(I_j)}{s(I_k)} \geq c_{min}$$
- Add rule $I_k \rightarrow Y$ with $Y = I_j - I_k$ to the rule set

Example

$$s(\{\text{chips}\}) = 0.75, s(\{\text{cola}\}) = 0.5, s(\{\text{chips}, \text{cola}\}) = 0.5$$

rule	confidence
$\{\text{cola}\} \rightarrow \{\text{chips}\}$	1.00
$\{\text{chips}\} \rightarrow \{\text{cola}\}$	0.67

Confidence

$$c(X \rightarrow Y) = \frac{s(X \cup Y)}{s(X)} = \frac{|\{T_k | X \cup Y \subseteq T_k\}|}{|\{T_k | X \subseteq T_k\}|} = p(Y|X)$$

Interesting association rules: Only those for which the confidence is greater than the support of the conclusion

$$c(X \rightarrow Y) > s(Y) \quad (\equiv \text{lift}(X \rightarrow Y) > 1)$$

Rule modification: Confidence can be increased by shifting items from the conclusion to the premise

Negative border: Used to derive negative association rules (e.g. customers who buy cola and chips do not buy beer)

$$\{I_j \mid s(I_j) < s_{cut} \wedge \forall I_k \subset I_j: s(I_k) \geq s_{cut}\}$$

Apriori: Number of potential itemsets is exponential in the number of items

But:

- Data is sparse: $|T_i| \ll |I|$
- Itemsets are generated in separate scans of the database
- Size of generated itemsets grows monotonically
- Large itemsets are usually useless
- Only k scans required ($k \ll |I|$)

Modifications and Extensions of Apriori

Hierarchical Apriori

In addition to the base level of items, determine also frequent itemsets on a higher level in an is-a hierarchy
→ Sometimes regularities can only be found at higher levels of abstraction



Partitioned Apriori

- 1st step:
- Partition the database
 - Compute locally frequent itemsets on each partition
- 2nd step:
- Determine the global support of all locally frequent itemsets
 - Remove all itemsets which do not satisfy the minimal support
- Heuristics:**
- Idea: if an itemset is globally frequent it will be so locally in at least one partition
 - Second step deals with a superset of possible frequent itemsets

More

- Sampled transactions
- Incremental rule mining
- Non uniform support thresholds
- Class association rules
- Rule mining on relational data

Association Rules with Relational Data

- Relational data has to be transformed into transaction data
- Apriori requires categorical data → binning must be performed
- The same category can appear as value of different attributes
 - Values must be combined with their attribute
 - Attribute-value pairs are taken as items

Summary

Architecture of Database Systems

Transaction Management

Modern Database Technology

Data Warehouses and OLAP

Data Mining

Big Data Analytics

Time Series Forecasting Models

Forecasting Process & Time Series Storage

Dependency Mining

Support: $s(X \rightarrow Y) = s(X \cup Y) = \frac{|(T_k | X \cup Y \subseteq T_k)|}{|D|} = p(XY)$

Confidence: $c(X \rightarrow Y) = \frac{s(X \cup Y)}{s(X)} = \frac{|(T_k | X \cup Y \subseteq T_k)|}{|(T_k | X \subseteq T_k)|} = p(Y|X)$

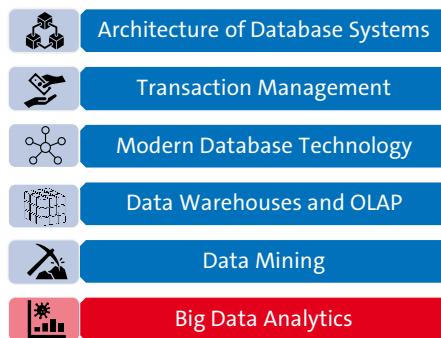
Strong rule: High Support + High Confidence

I^1	#	$s(I^1)$	I^2	#	$s(I^2)$	I^3	#	$s(I^3)$	I^4	#	$s(I^4)$
[chips]	3	0.75	[chips, cola]	2	0.5	[chips, cola]	2	0.5	(beer, chips, cigar)	2	0.5
[cola]	2	0.5	[beer, chips]	2	0.5	[beer, chips]	2	0.5	[chips, cigarettes]	2	0.5
[peanuts]	1	0.25	[chips, cigarettes]	2	0.5	[chips, cigarettes]	2	0.5	[beer, cigar]	1	0.25
[beer]	3	0.75	[beer, colas]	1	0.25	[beer, colas]	1	0.25	[cigarettes, colas]	1	0.25
[cigarettes]	3	0.75	[cigarettes, colas]	1	0.25	[cigarettes, colas]	1	0.25	[beer, cigarettes]	3	0.75

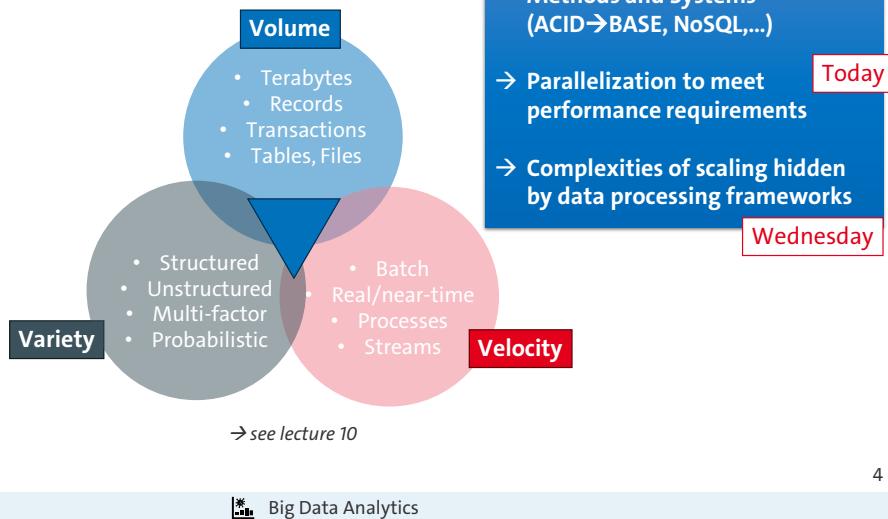
Prune because of non-frequent subset

2

Course Outline



The Big Data Vs



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!)

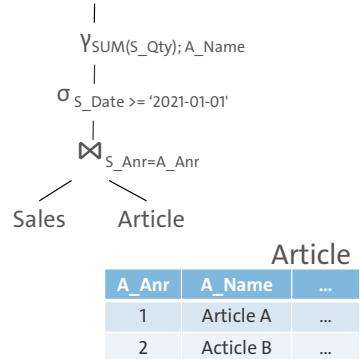
```
SELECT A_Name, SUM(S_Qty)
FROM Article, Sales
WHERE A_Anr = S_Anr AND
      S_Date >= '2021-01-01'
GROUP BY A_Name
```

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

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

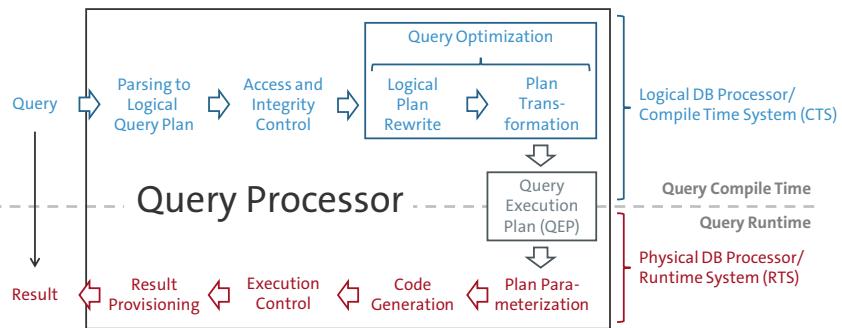
A_Name	SUM
Article A	12
Article B	1

Query Plan
(How!)



Big Data Analytics

Query Processing



Logical Query Plan

Mono-Block Query

```

SELECT C_NAME, C_ADDRESS
FROM TPCD.CUSTOMER,TPCD.SUPPLIER
WHERE C_NAME = S_NAME
AND C_MKTSEGMENT = 'MACHINERY';
    
```

Projection
Selection
Join Operation
Source Relations

Star Query

```

SELECT
P_BRAND,O_SHIPPRIORITY,
SUM(L_QUANTITY*L_EXTENDEDPRICE)
AS TURNOVER
FROM
TPCD.LINEITEM,TPCD.ORDERS,TPCD.PART
WHERE L_ORDERKEY = O_ORDERKEY
AND L_PARTKEY = P_PARTKEY
AND O_ORDERSTATUS = 'F'
AND P_CONTAINER = 'LG_BAG'
GROUP BY
[P_BRAND,O_SHIPPRIORITY]
HAVING
AVG(L_QUANTITY) > 250;
    
```

Projection
Selection
Join Operation
Source Relations

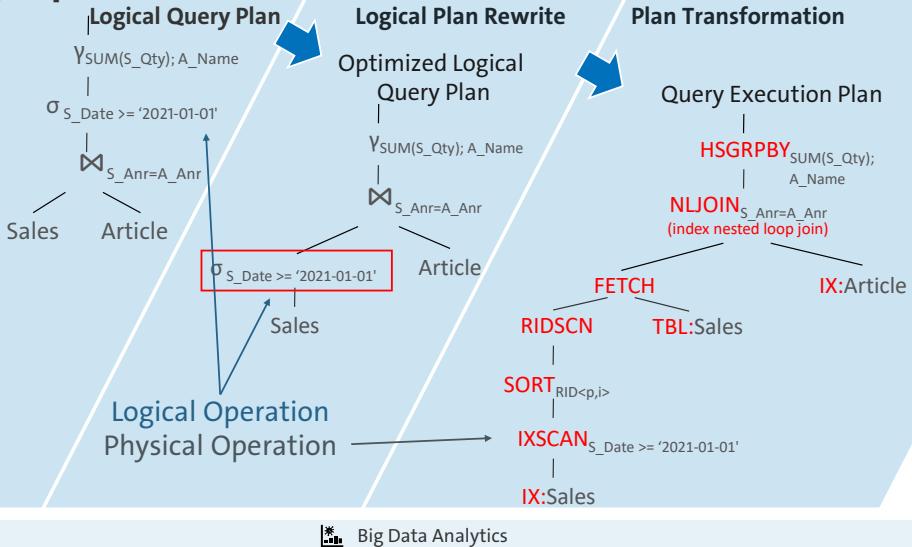


Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

→ see lectures 2 and 9

Query Optimization – Example



Parallel Execution Opportunities

Query 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)

Intra-query parallelism → Important for OLAP

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

Operator Perspective

Inter-operator parallelism

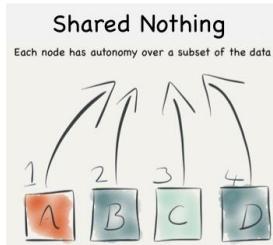
- Parallel execution of multiple operators
- Goal: increase operator throughput, but limited through operator dependencies
→ See lectures 7 and 8

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
```

Compile

U

$\sigma_{amount>50}$ $\sigma_{amount>50}$

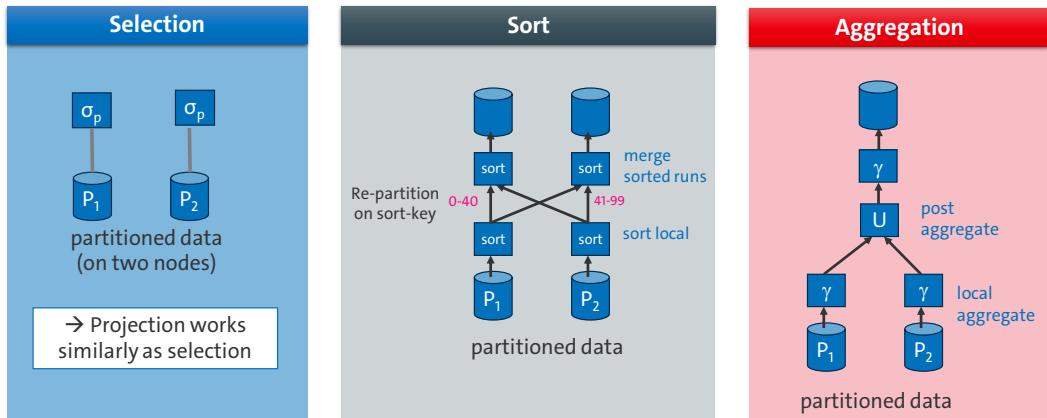
Orders₁ on Node₁

Orders₂ on Node₂

6/7/17	623152	Couch	\$370
6/7/17	268822	Car	\$1120
6/7/17	400381	Bike	\$86
6/7/17	589111	Chair	\$10

6/7/17	171111	Laptop	\$350
6/7/17	337754	Box	\$50
6/7/17	329111	Scooter	\$14
6/7/17	116468	Hammock	\$8000

Parallel Operators



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 $age \geq 30$ ”
 - Query can prune fragment with $age < 30$

Sort

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

Aggregation

Main idea

- 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

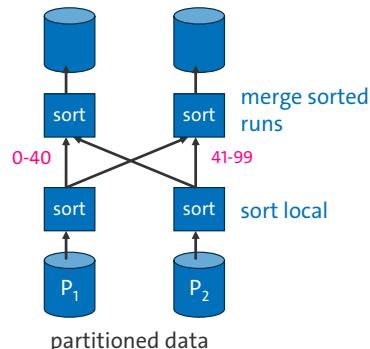
Example: Sort-by Age Ascending

Name	Age
Adler	22
Muller	38

Re-partition on sort-key

Name	Age
Muller	38
Bishop	57

Name	Age
Bishop	57
Muller	38

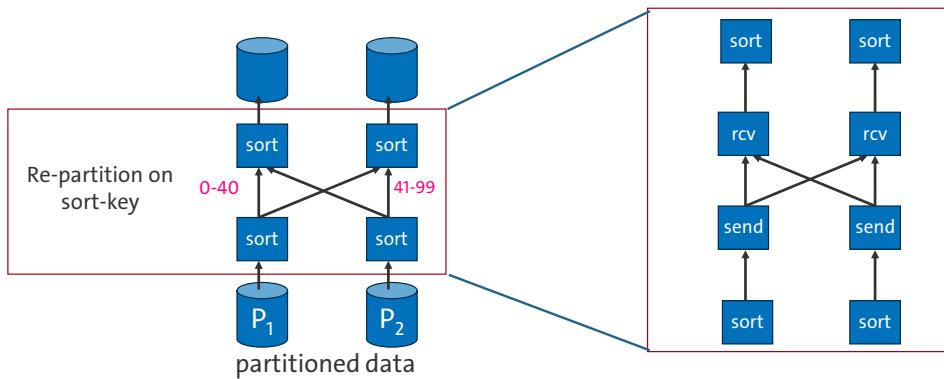


Name	Age
Zeller	47
Bishop	57

Name	Age
Adler	22
Zeller	47

Name	Age
Zeller	47
Adler	22

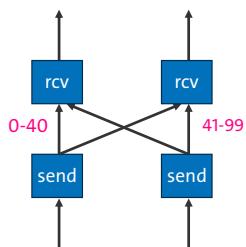
Data Shuffling (AKA Re-Partitioning)



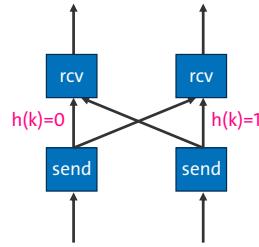
- Data shuffling is implemented as a separate operator (send-receive)
- Sort operators are un-aware of parallel (distributed) execution

Data Shuffling: Details

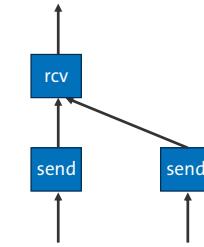
Data Shuffling can use different partitioning strategies (range vs. hash-partitioning, N:M vs. N:1) to re-partition data during query execution



Range-based N:M



Hash-based N:M

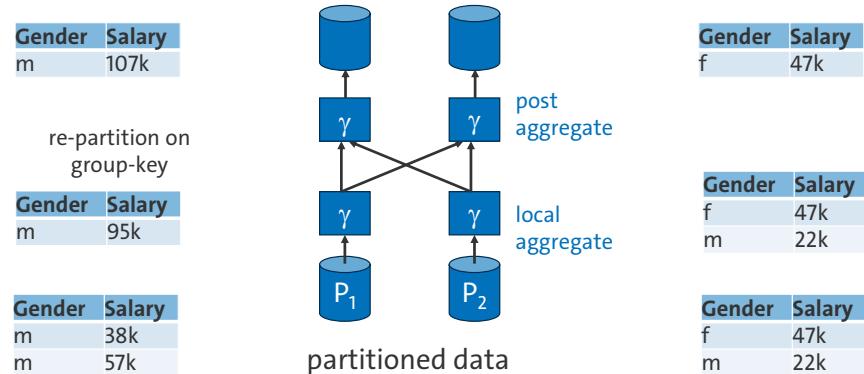


N:1 (no part. function)

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

Example: Parallel Aggregation

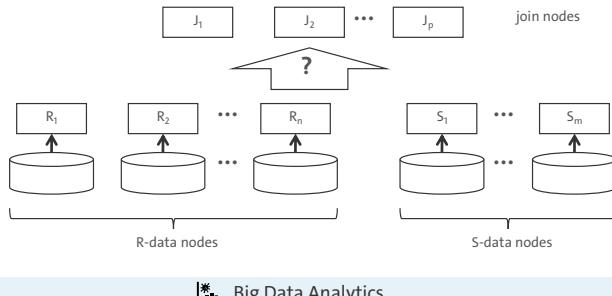
Example: `SELECT SUM(Salary) FROM Census GROUP BY Gender`



Parallel Join

Scenario

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

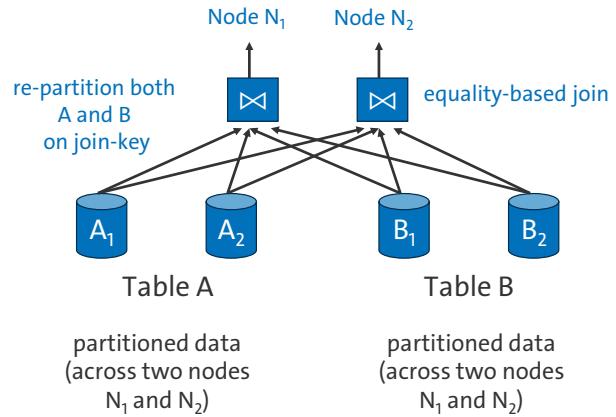


- Joins are the most expensive distributed operation
 - might need to shuffle large intermediate results
- Different Strategies for Parallel Execution
 - symmetric re-partitioning
(synonym: (fully) re-partitioned join)
 - asymmetric re-partitioning
(synonym: one-way re-partitioned join, directed join)
 - fragment and replicate
(synonym: broadcast join)

Join: Symmetric Repartitioning

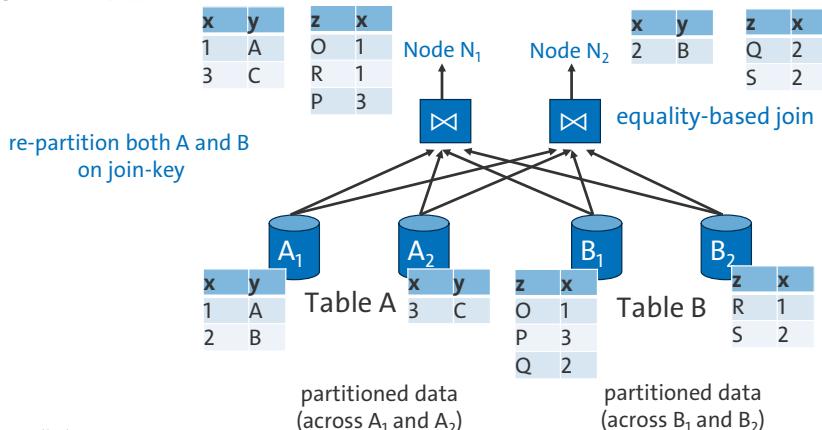
„Repartitioned join“

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

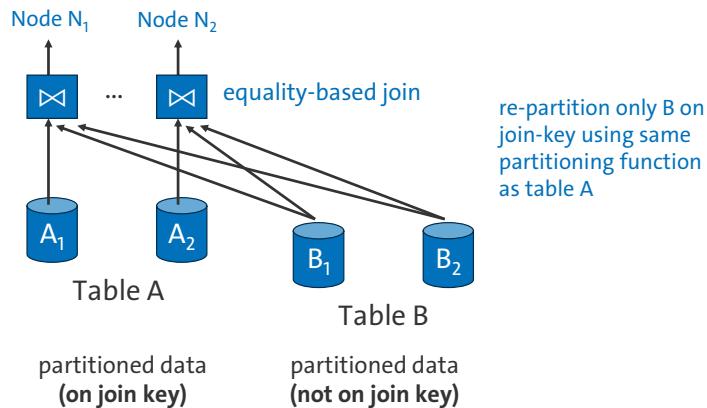


Join: Symmetric Repartitioning

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



Join: Asymmetric Repartitioning

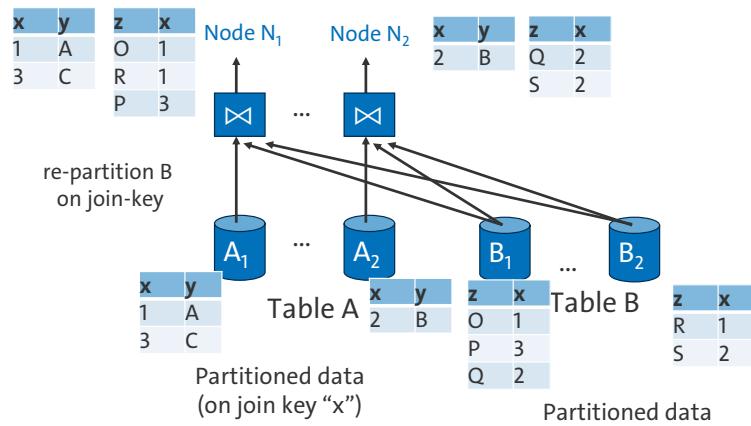


One-way redistribution join "directed join"

- 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

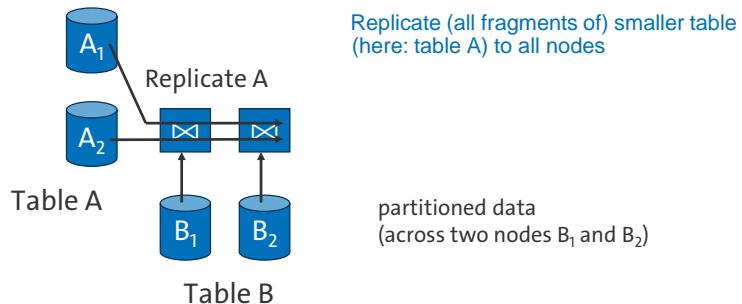
Example: Asymmetric Repartitioning

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



Join: Fragment and Replicate

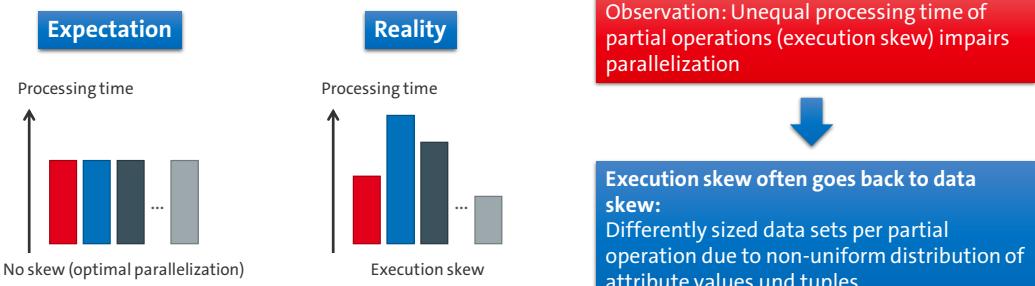
partitioned data
(across two nodes
 A_1 and A_2)



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

The Data Skew



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

Transparency for Query Formulation

Data Management can support different levels of transparency
→ A transparency level defines how much users need to know about how data is distributed when writing a SQL query



Typical Transparency Levels:

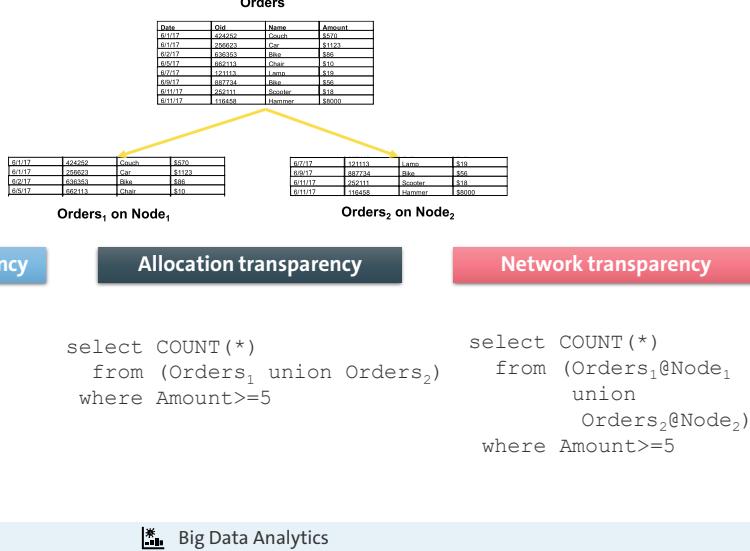
Fragmentation transparency
User needs to know nothing about distribution

Allocation transparency
Users need to know fragmentation scheme

Network transparency
Users need to know fragmentation & allocation

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),...

Transparency levels



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

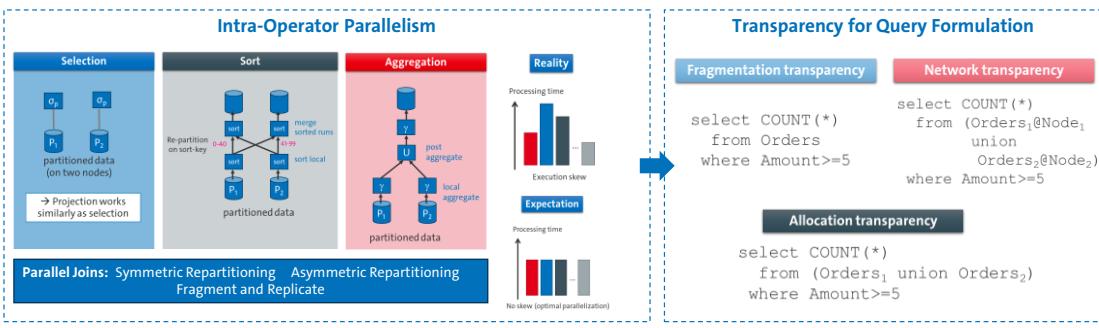
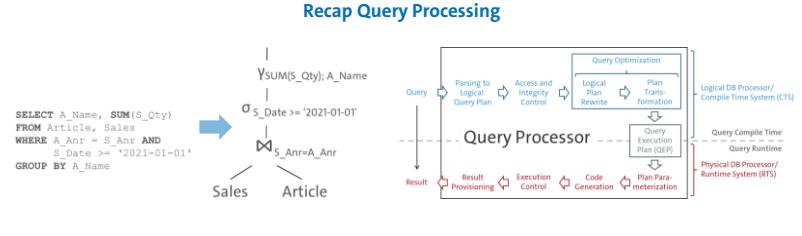
Network transparency: Users need to know fragmentation and allocation

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

Which level of transparency?

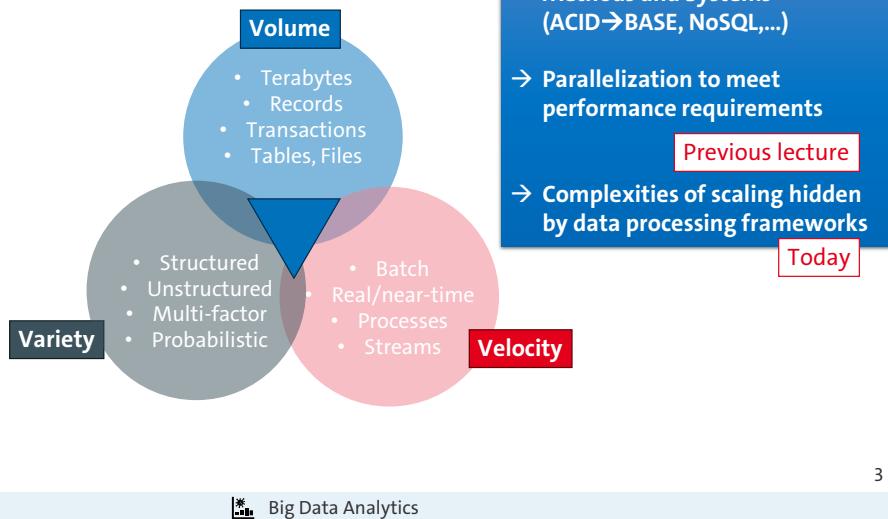
- Compromise: Ease of use versus ability to control query processing by user
 - Choice often depends on application
 - Full transparency often poor choice for geographically distributed DBMSs; Users want to control which operations are executed remotely
 - However, parallel DBMSs often implement full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

Summary



The Big Data Vs

→ see lectures 10-15



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, ...)

Complexities of scaling beyond plain operator parallelization

- Monitoring (health checks, application statistics)
- Scheduling (e.g. rebalancing)
- Fault-tolerance, e.g. restarting workers, rescheduling failed work

Batch & Stream Processing

Batch

- Operates on complete data
 - Periodic jobs, e.g. during night times
- Efficient but high latency

Volume

Stream

- Operates on partial data (one-at-a-time)
→ Low end-to-end latency
- Challenges
- Long-running jobs
 - Data may arrive delayed or out-of-order
 - Incomplete input

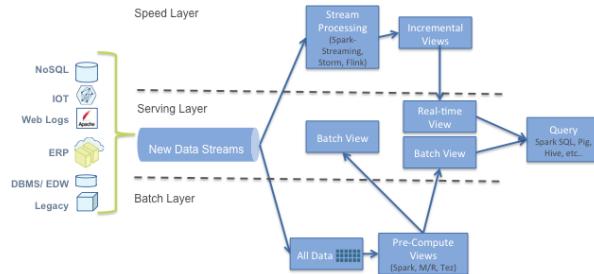
Velocity

4

Lambda Architecture

→ see lecture 16

→ Batch & Stream processing

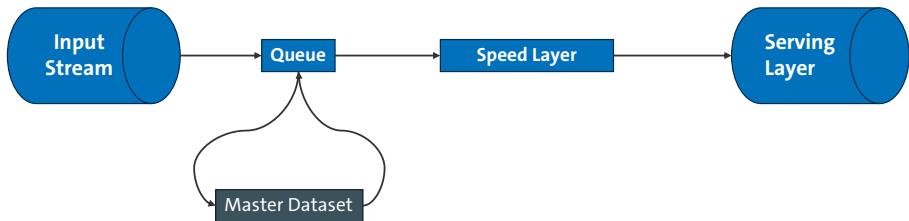


Disadvantage

2 code bases & 2 deployments, e.g. Hadoop & Storm

Kappa Architecture

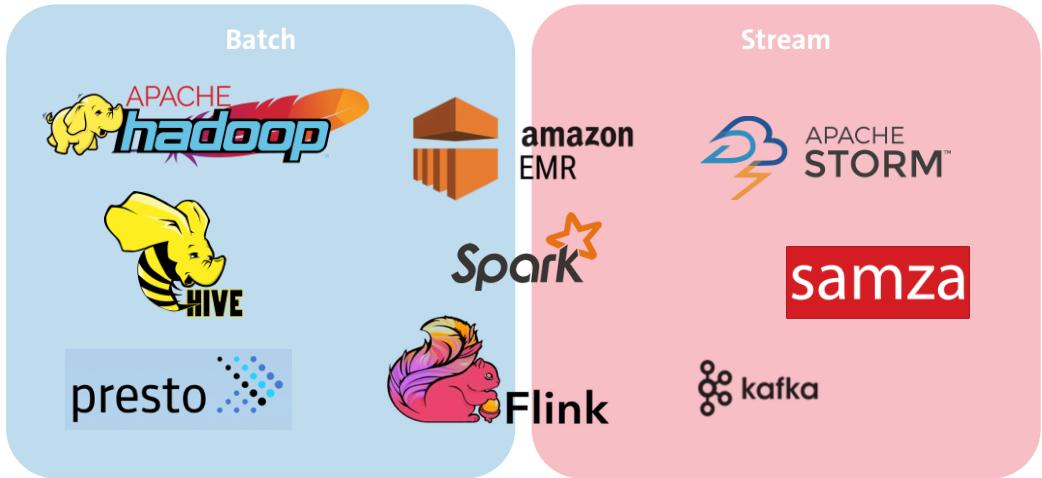
→ No batch layer!



Disadvantage
Real time processing only

Introduces backpressure (without large enough buffer)

Framework Overview



7

Big Data Analytics

- EMR – Elastic Map reduce, provided as web service, provides managed Hadoop, spark, presto
- Presto – Distributed (SQL) query engine, allows using different data sources, e.g. Hadoop, kafka, MongoDB, MySQL,...
- Kafka → Kappa architecture

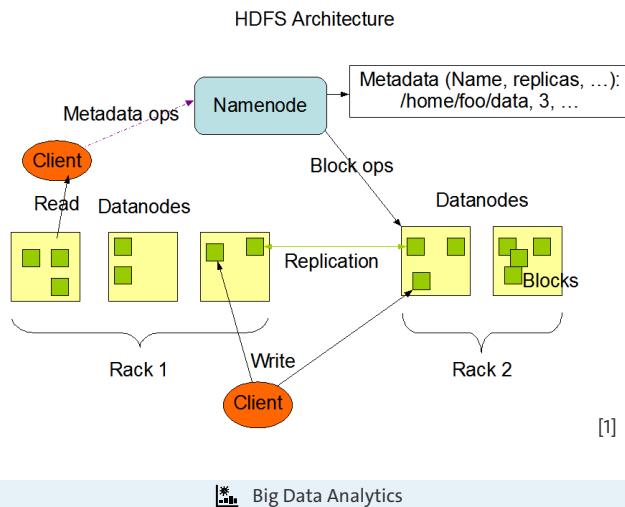
Framework Overview



8

- Uses the Map-Reduce Paradigm
- HDFS: Scalable , shared nothing file system for throughput-oriented workloads
- Other Hadoop projects
 - Hive : SQL(-dialect) compiled to YARN jobs (Facebook)
 - Pig : workflow-oriented scripting language (Yahoo)
 - Mahout : Machine Learning algorithm library in Map Reduce
 - Flume : Log Collection and processing framework
 - Whirr : Hadoop provisioning for cloud environments
 - Giraph : Graph processing à la Google Pregel
 - Drill , Presto, Impala : SQL Engines

Hadoop File System



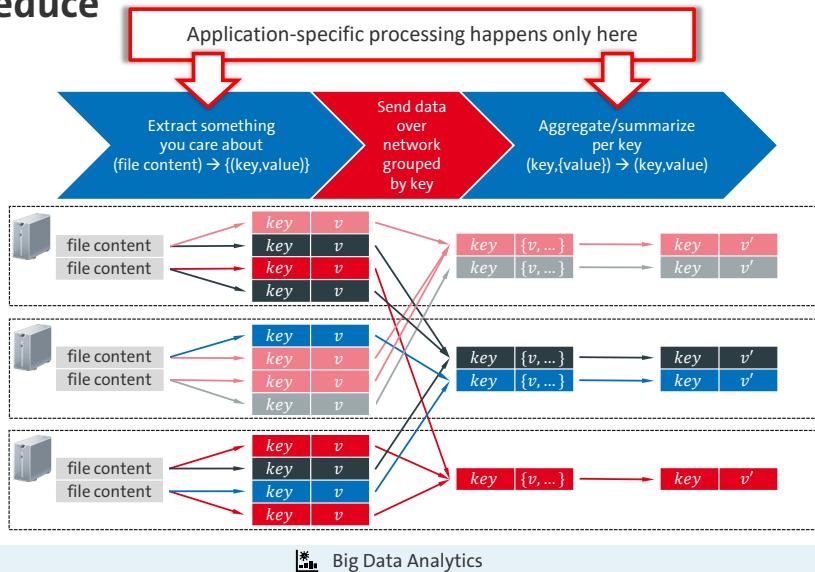
9

[1] Big Data Analytics

- Modelled after: Googles GFS (2003)
- Cluster Nodes:
 - Single namenode: Metadata (files + block locations)
 - Single master server: Manages file system namespace and regulates access to files by clients
 - Multiple datanodes: Save fileblocks (usually 64 MB), blocks replicated for fault tolerance and read performance → Data can be used where it is stored, usually one data node per physical cluster node, serve read and write requests
- Design goal: Maximum Throughput and data locality for Map-Reduce

[1] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html, accessed 02.07.2024

Map Reduce



Data Foundation

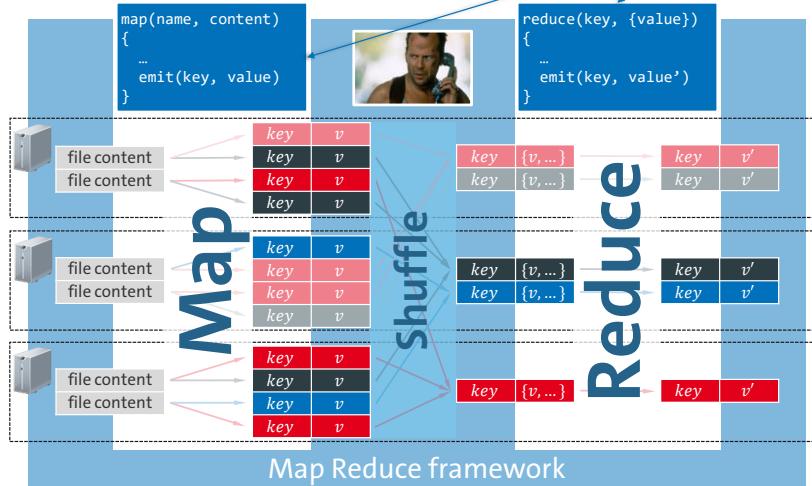
- Key-Value Pairs (key, value)
- Key and Values can be everything

Map Reduce

- Framework for parallel computing
- Programmers get simple API
- Do not have to worry about
 - Parallelization
 - Data distribution
 - Load balancing
 - Fault tolerance
- Allows everybody to process huge amount of data (terabytes/petabytes/...) on thousands of computer nodes

Map Reduce

- Executed locally only
 - Can be written like non-parallel program
- Yeah!*



- The programmer essentially only specifies two (sequential) functions
- Framework takes care of all parallelization
- Ships code of work nodes
- Performs shuffle step
- Monitors worker progress
- Handles node failures

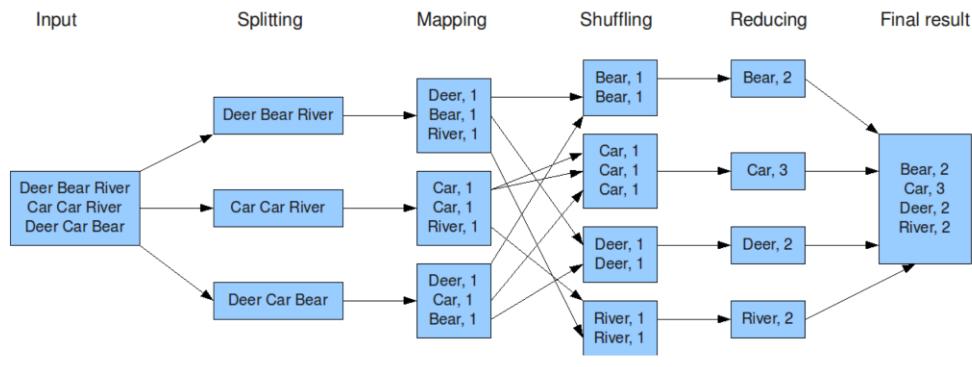
Map: $\text{map}(k1, v1) \rightarrow \text{list}(k2, v2)$

- Inputs data record and outputs a set of intermediate key-value pairs, each of type k2 and v2
- Types can be simple or complex user-defined objects
- Each map call is fully independent (no execution ordering, sync or communication)

Reduce: $\text{reduce}(k2, \text{list}(v2)) \rightarrow \text{list}(k3, v3)$

- Combines information across records that share the same intermediate key
- Each reduce call is fully independent

Example: Word Count



Map Reduce Limitations

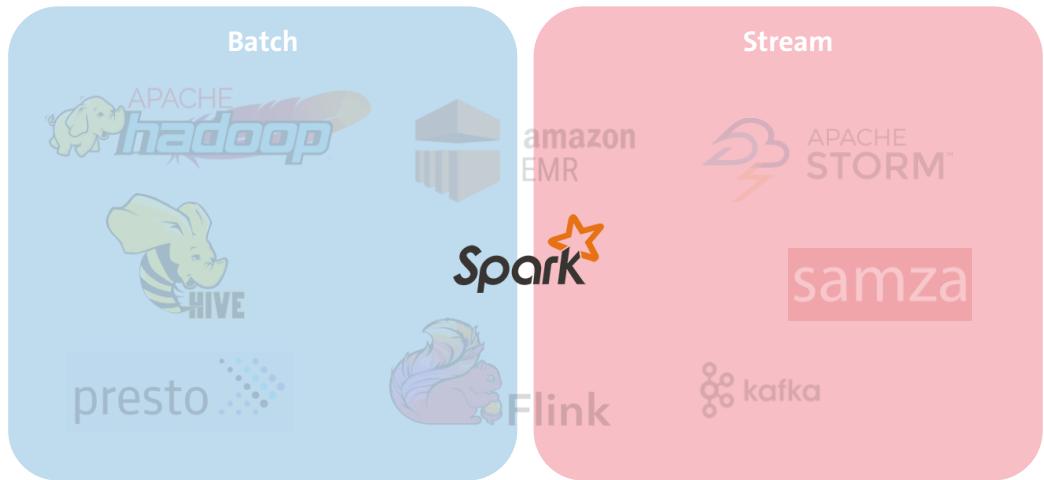
API

- Very simple
- Not every problem necessarily fit into map-reduce abstraction
- Does not compose very well into larger programs
- No support for cyclic programs
- Fixed data flow

Performance

- Pure disk-based data processing implies performance bottle necks in larger programs
- Loops write state complete to disk in every iteration (might be peta bytes of disk IO each time)

Framework Overview

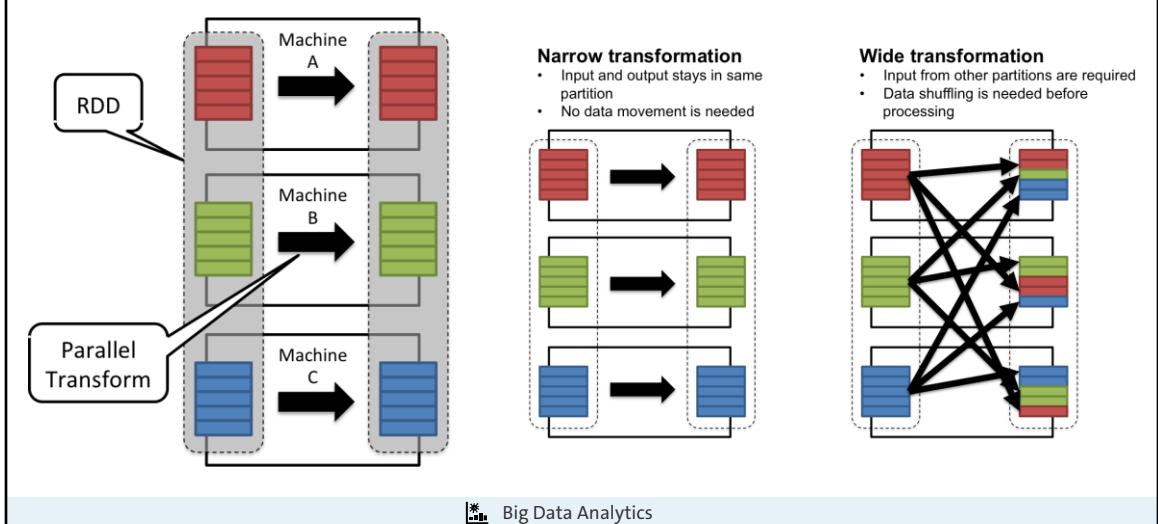


13

Big Data Analytics

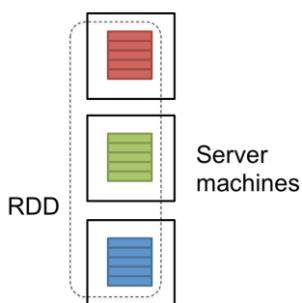
- Basic idea: „In-Memory“ Hadoop optimized for iterative processing (e.g. k-means)
- Resilient Distributed Datasets (RDDs): partitioned, in-memory set of records

Parallel Processing of RDDs

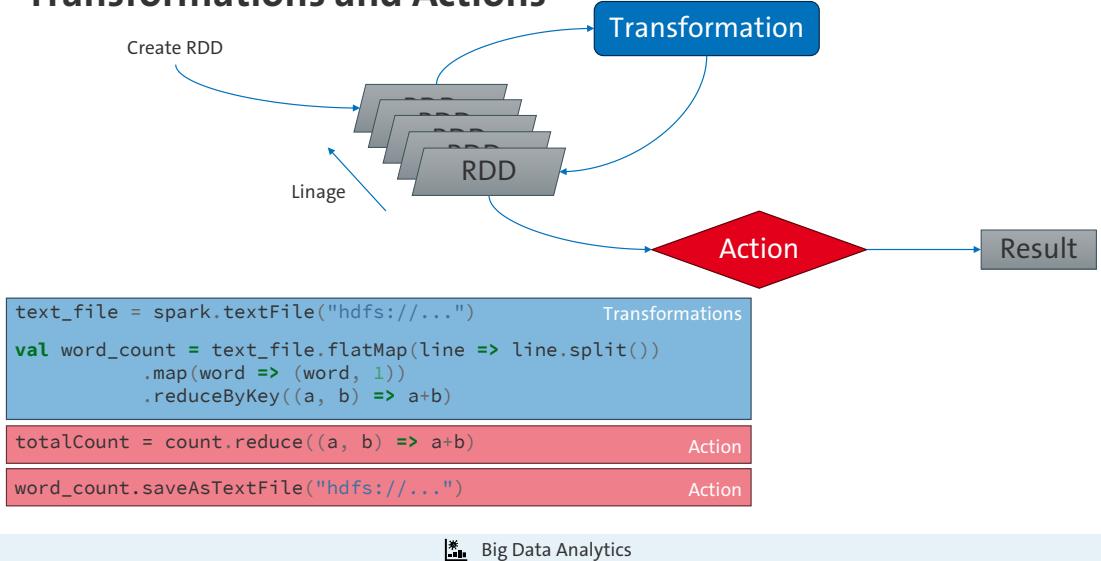


Resilient Distributed Datasets (RDDs)

- Sparks original in-memory data structure
- Fault-tolerant immutable distributed collection of data
- Strongly typed but not necessarily structured like a table, e.g. a collection of documents
- No schema imposed
- Manipulated with functional programming constructs in low-level transformation and actions
- Two ways to create RDDs
 - parallelizing an existing collection in your driver program
 - referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, etc.



Transformations and Actions



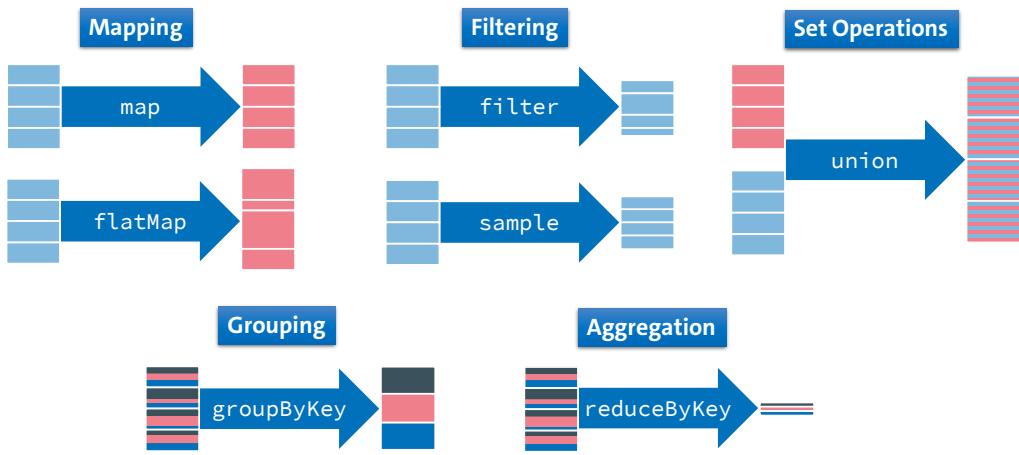
Transformations

- (High-order) functions to create an RDD/Dataset from existing RDDs/Datasets
- Just build up the data flow DAG when executed in the driver program (nothing happens with the data)

Actions

- Return a value (actual data) to the driver program (or write out data)
- Trigger execution of whole data flow DAG
- Examples: collect, count, reduce, countByKey, saveAsTextFile, saveAsObjectFile
- List of actions: <http://spark.apache.org/docs/latest/programming-guide.html#actions>

RDDs Transformations



Big Data Analytics

- **map(func):** Return a new distributed dataset formed by passing each element of the source through a function *func*.
 - **flatMap(func):** Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).
 - **filter(func):** Return a new dataset formed by selecting those elements of the source on which *func* returns true.
 - **sample(withReplacement, fraction, seed):** Sample a fraction *fraction* of the data, with or without replacement, using a given random number generator *seed*.
 - **union(otherDataset):** Return a new dataset that contains the union of the elements in the source dataset and the argument.
 - **groupByKey([numPartitions]):** When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs.
 - **reduceByKey(func, [numPartitions]):** When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V,V) => V. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.
- List of all transformations: <http://spark.apache.org/docs/latest/programming-guide.html#transformations>

RDD Transformation Exercise

Which transformations and/or actions can be used to implement the following scenarios?

1. Count all items in a basket that belong to the item group “food”
2. Create a histogram
3. Create an estimated histogram for a very large dataset, i.e. where reading the whole dataset for creating the histogram would take too long
4. Calculate the end-of-year bonus (based on yearly salary) from a table containing the monthly salaries.
5. Rename a department

17

Basic Spark Program

Read data from distributed file system into an in-memory data structure ...

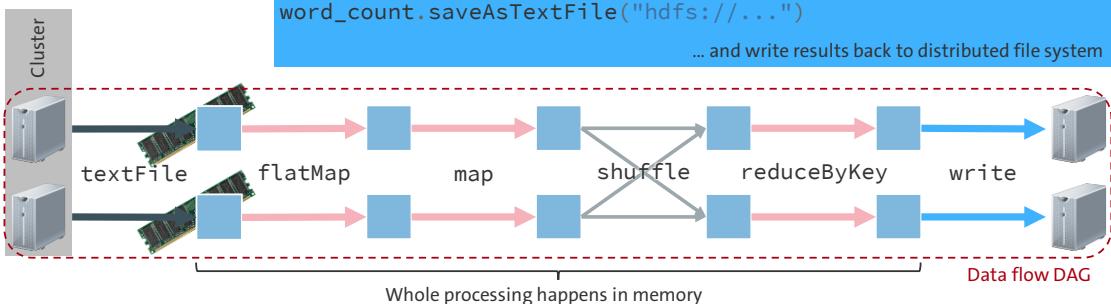


```
val text_file = spark.textFile("hdfs://...")
```

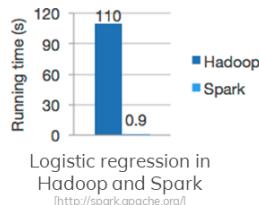
```
val word_count = text_file.flatMap(line => line.split())
    .map(word => (word, 1))
    .reduceByKey((a, b) => a+b) ... process it in parallel ...
```

```
word_count.saveAsTextFile("hdfs://...")
```

... and write results back to distributed file system



- Generalization of Map Reduce
 - No specialization
 - Support for wide range of applications in a single engine
 - Map Reduce is just one set of supported constructs
- Two major improvements
 - Keeps data memory while processing
 - Generalizes data flow to DAGs and lets user define data flow
- Key features
 - Handles batch, interactive, and real-time processing within a single framework
 - Native integration with Java, Python, and Scala
 - Programming at a higher level of abstraction



Language Integrations

Scala, Java, and Python

```
val text_file = spark.textFile("hdfs://...")  
val word_count =  
    text_file.flatMap(line => line.split())  
        .map(word => (word, 1))  
        .reduceByKey((a, b) => a+b)  
word_count.saveAsTextFile("hdfs://...")
```



```
JavaRDD<String> text_file = spark.textFile("hdfs://...")  
JavaRDD<Integer> word_count =  
    text_file.flatMap(line -> line.split())  
        .map(word -> (word, 1))  
        .reduceByKey((a, b) -> a+b)  
word_count.saveAsTextFile("hdfs://...")
```



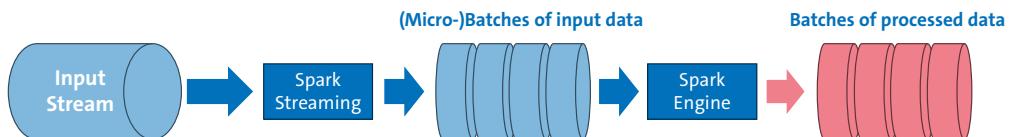
```
text_file = spark.textFile("hdfs://...")  
word_count =  
    text_file.flatMap(lambda line: line.split())  
        .map(lambda word: (word, 1))  
        .reduceByKey(lambda a, b: a+b)  
word_count.saveAsTextFile("hdfs://...")
```



There's also an API for R

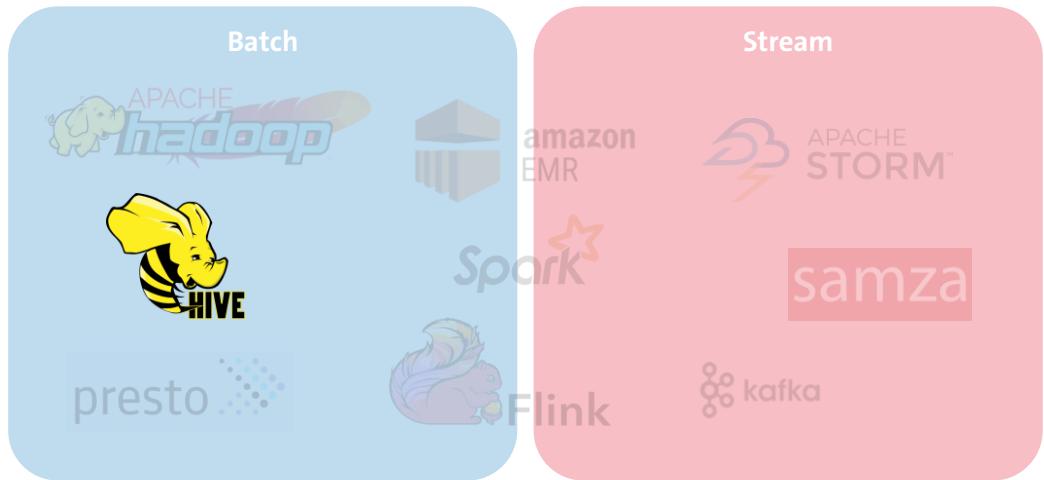
Spark Streaming

Split stream into micro-batches to process with Spark engine



DStream: Discretized RDD
→ RDDs are processed in order
but no ordering within RDD

Framework Overview



22

- Data warehouse on top of Map Reduce
- HiveQL SQL-like query language
- Supports different formats, e.g. Iceberg, RDD,...
- Goal: make Map Reduce more easily accessible

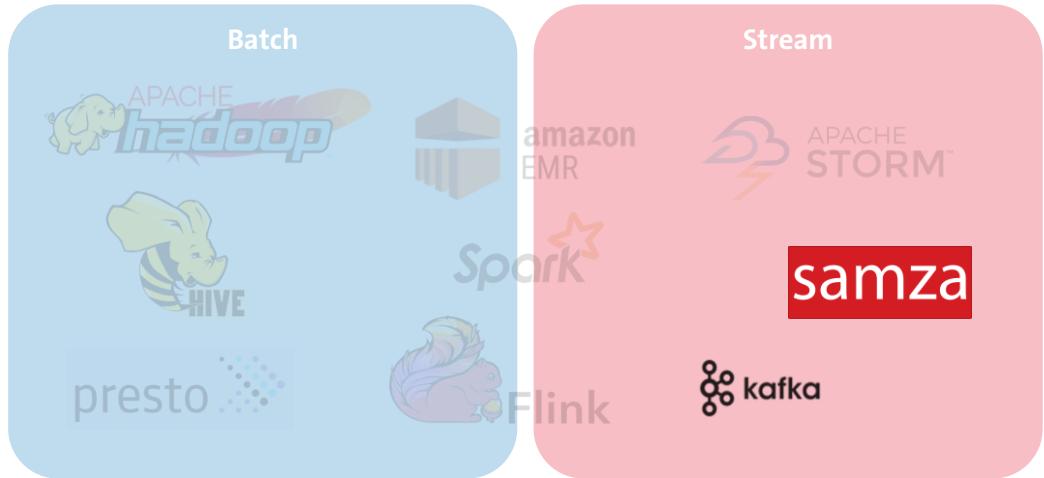
Apache Hive

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
                      page_url STRING, referrer_url STRING,
                      friends ARRAY<BIGINT>, properties MAP<STRING, STRING>,
                      ip STRING COMMENT 'IP Address of the User')
COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
CLUSTERED BY(userid) SORTED BY(viewTime) INTO 32 BUCKETS
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '1'
  COLLECTION ITEMS TERMINATED BY '2'
  MAP KEYS TERMINATED BY '3'
STORED AS SEQUENCEFILE;
```

Example code from <https://cwiki.apache.org/confluence/display/Hive/Tutorial>

Comments can be added at column level and at table level

Framework Overview



24

Big Data Analytics

Kafka

- A storage format and a streaming platform (kappa architecture)
- Storage format is also used/can be processed by other systems, e.g. SAMZA, Flink,...

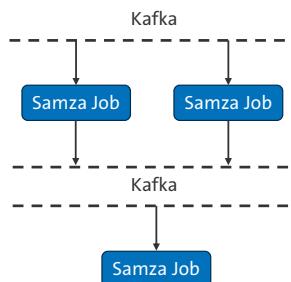
SAMZA

- Co-developed with Kafka
- Simple by design: only single-step jobs
- Local state
- Native stream processor: low latency
- Users: LinkedIn, Uber, Netflix, TripAdvisor, Optimizely, ...

History

- Developed at LinkedIn
- 2013: open-source (Apache Incubator)
- 2015: Apache top-level project

Samza



Kafka Storage

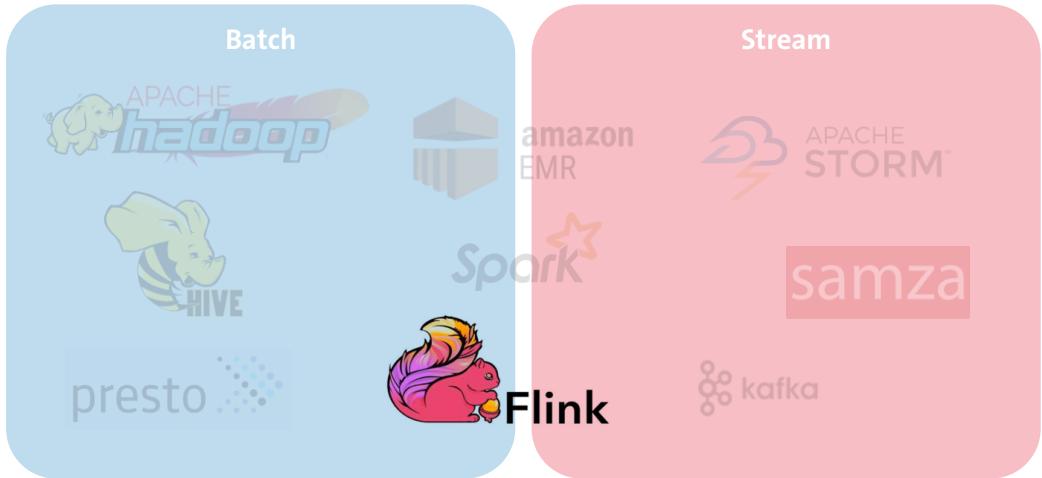
- Not intended as permanent storage → Maximum time or amount of data can be defined before data is purged
- Stores events/messages in partitions which are segmented (usually 1GB)
- Compression can be enabled

25

Big Data Analytics

- Job : processing step
 - Robust
 - But: often several jobs
- Task : Job instance (parallelism)
- Message : single data item
- Output persisted in Kafka
 - Easy data sharing
 - Buffering no back pressure
 - But: Increased latency
- Ordering within partitions

Framework Overview



26

Overview

- Native stream processor: Latency <100ms feasible
- Abstract API for stream and batch processing, stateful, exactly-once delivery
- Many libraries: Table and SQL, CEP, MachineLearning , Gelly...
- Users: Alibaba, Ericsson, Otto Group, ResearchGate, Zalando...

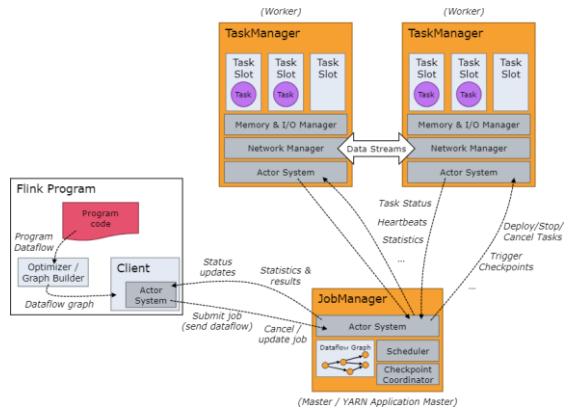
History

- 2010: Start as Stratosphere at TU Berlin, HU Berlin, and HPI Potsdam
- 2014: Apache Incubator, project renamed to Flink
- 2015: Apache top-level project

Flink

2 Execution Modes: Batch & Streaming

- Batch: Requires bounded jobs, i.e. all input coming from the source(s) is known before execution)
- Streaming: Can be used for bounded and unbounded jobs

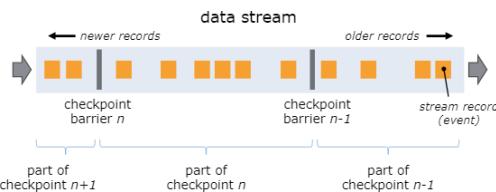


<https://nightlies.apache.org/flink/flink-docs-master/docs/concepts/flink-architecture/>

Automatic backups of local state

→ Stored in RocksDB, Savepoints written to HDFS

Checkpointing in Flink (Stream processing)



- Checkpoints are always between different records, they never overtake them
- Intermediate operator emits a barrier n if it received the same barrier n from all its input streams
 - Final/sink operator acknowledges snapshot n to the checkpoint coordinator
 - A snapshot n is completed when all sink operators acknowledged n
 - No records from before snapshot n will be needed anymore

Further Reading

Lightweight Asynchronous Snapshots for Distributed Dataflows, Carbone et al.
<https://arxiv.org/abs/1506.08603>

How to decide?

As always: It depends on the use case and its requirements
→ Tradeoffs have to be made



Latency vs. throughput
Latency vs. fault-tolerance
Simplicity vs. control
Historically grown system vs. completely new setup
Required features, e.g. state management, consistency guarantees, ordering,...
...

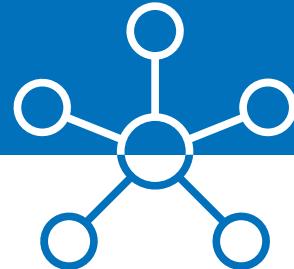
Databases and Information Systems (DIS) – Quiz 2

Universität Hamburg



Foto: UHH/Esfandiari

Modern DBS: NoSQL Systems



Which graph models include edge and vertex properties?

Which model is used in neo4j?

RDF Graph

Weighted Graph

Property Graph

Weighted Graph

Directed Graph

Canonical Property
Graph

Which graph models include edge and vertex properties?

Which model is used in neo4j?

RDF Graph

Weighted Graph

Property Graph

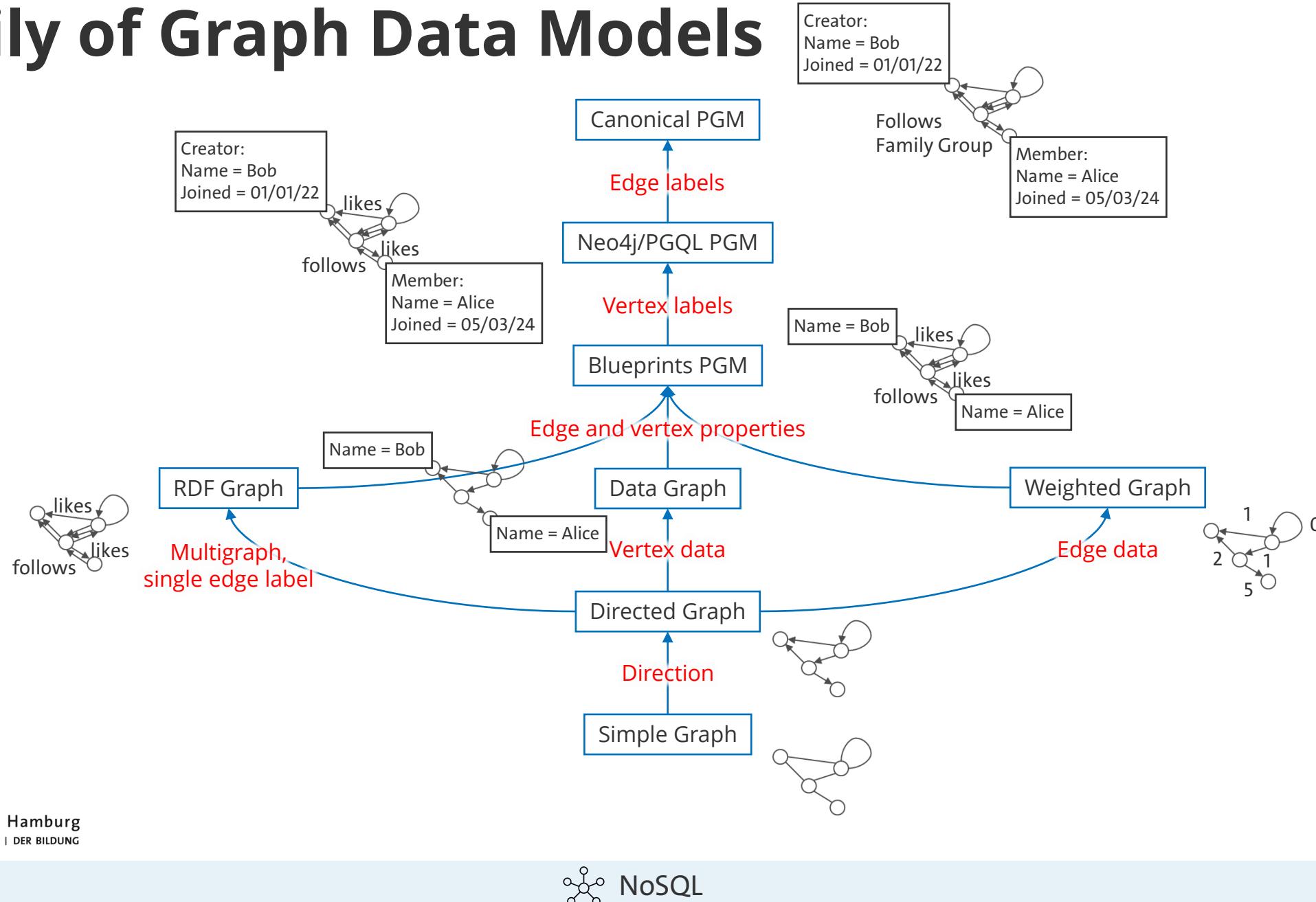
Weighted Graph

Directed Graph

Canonical Property
Graph



Family of Graph Data Models



Which of the following elements can directly be modelled using neo4j?

Entities with properties

Edge properties

Multivalued properties

N-ary relationships

N:M relationships

Which of the following elements can directly be modelled using neo4j?

Entities with properties

Edge properties

Multivalued properties

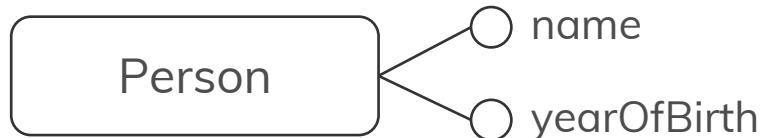
N-ary relationships

N:M relationships

Property Graph Modelling

Entity with Properties

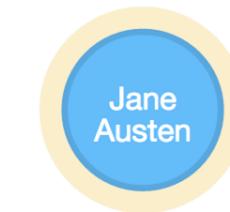
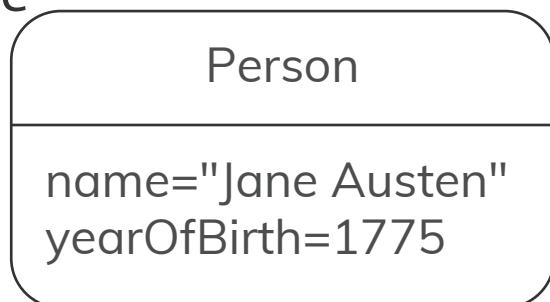
- Entity relationship model



- Schema typically implicit, i.e. given with instances

(ja:Person { name: 'Jane Austen', yearOfBirth: 1775 })

- Instance



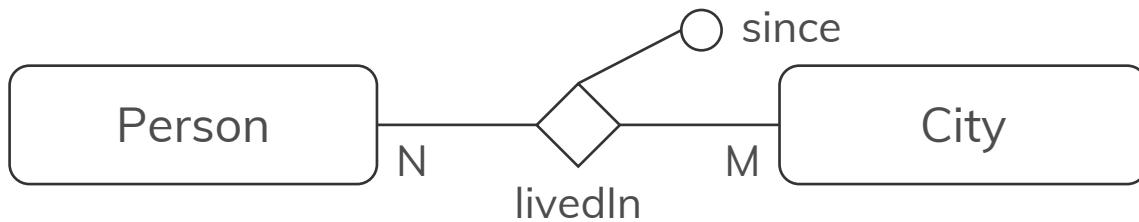
Person

<id>: 175 yearOfBirth: 1775 name: Jane Austen

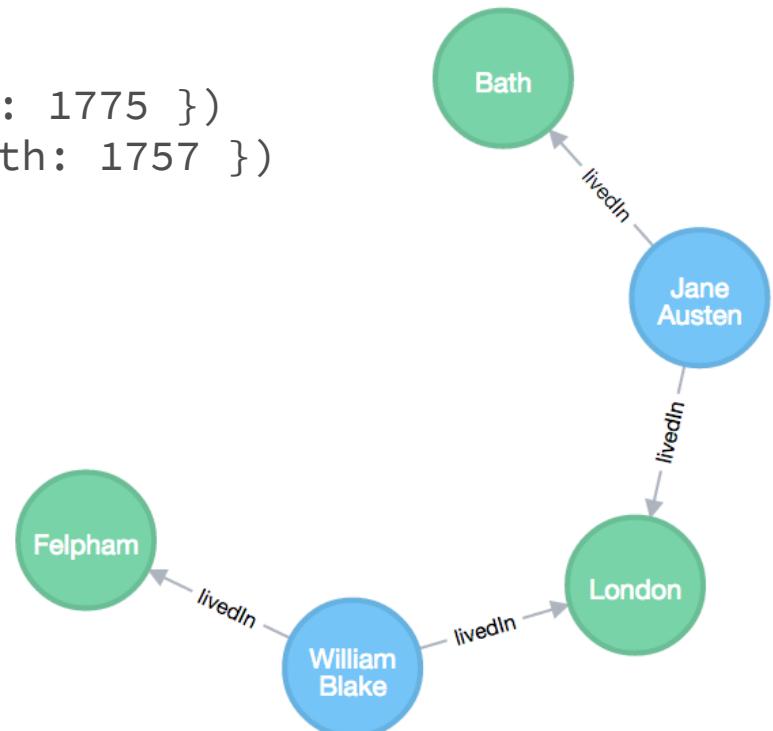
Property Graph Modelling

Relationships (N:M)

- Entity relationship model



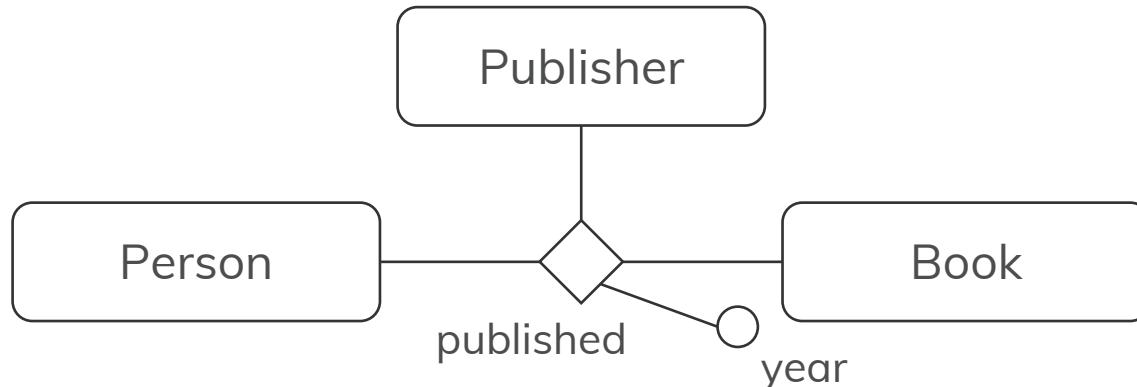
- Vertices
 - (ja:Person { name: 'Jane Austen', yearOfBirth: 1775 })
 - (wb:Person { name: 'William Blake', yearOfBirth: 1757 })
 - (lo:City {name: 'London'})
 - (ba:City {name: 'Bath'})
 - (fe:City {name: 'Felpham'})
- Edges
 - (ja)-[:livedIn {since: 1775}]->(lo)
 - (ja)-[:livedIn {since: 1800}]->(ba)
 - (wb)-[:livedIn {since: 1757}]->(lo)
 - (wb)-[:livedIn {since: 1800}]->(fe)



Property Graph Modelling

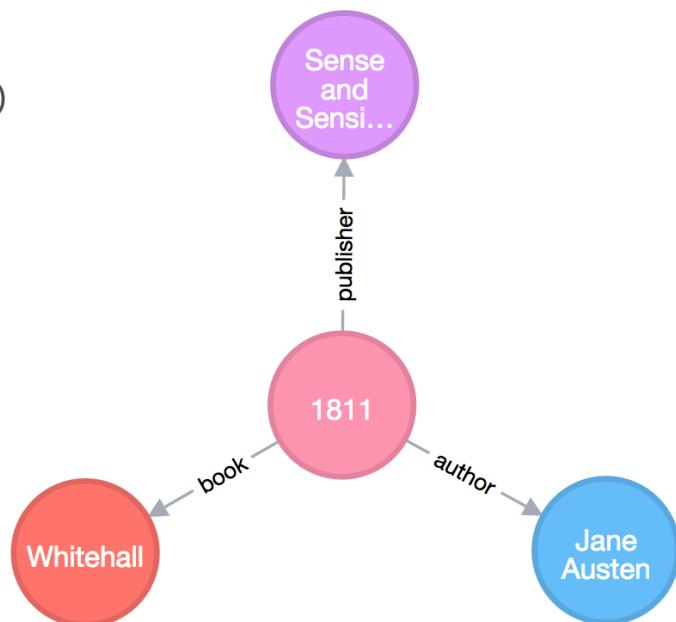
n-ary relationships (with n>2)

- Entity relationship model



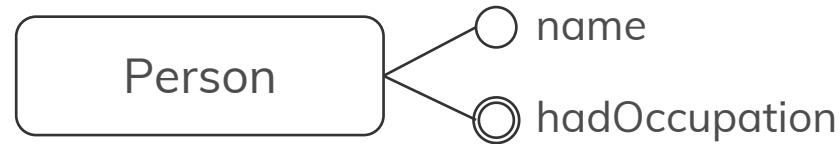
```
(ja:Person { name: 'Jane Austen', yearOfBirth: 1775 })
(wh:Publisher { name: 'Whitehall' })
(sas:Book {title: 'Sense and Sensibility' })
(pub:Publication {year: 1811 })

(pub)-[:author]->(ja)
(pub)-[:book]->(wh)
(pub)-[:publisher]->(sas)
```



Property Graph Modelling

Multivalued properties



```
(wb:Person {name: 'William Blake', hadOccupation: ['Poet','Painter','Printmaker']})
```

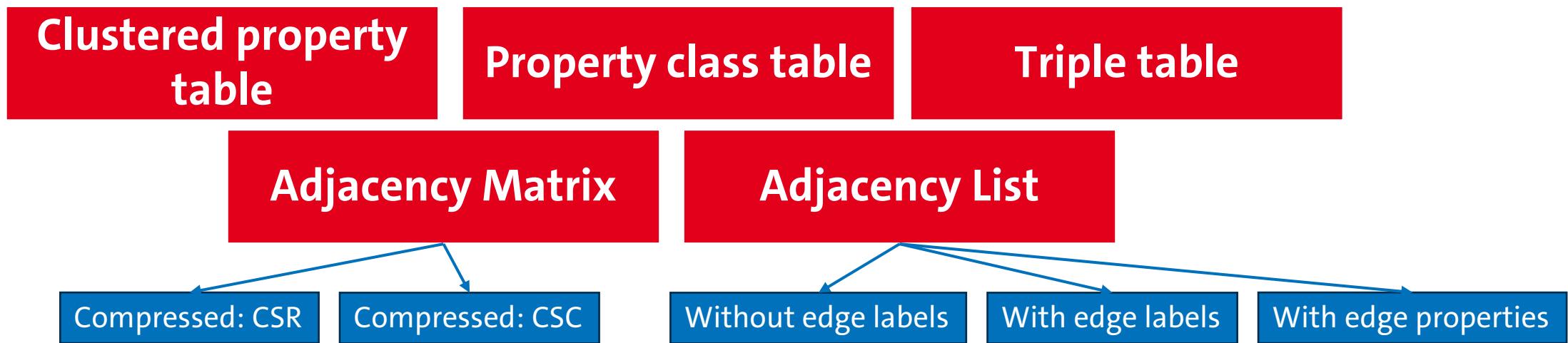


Person

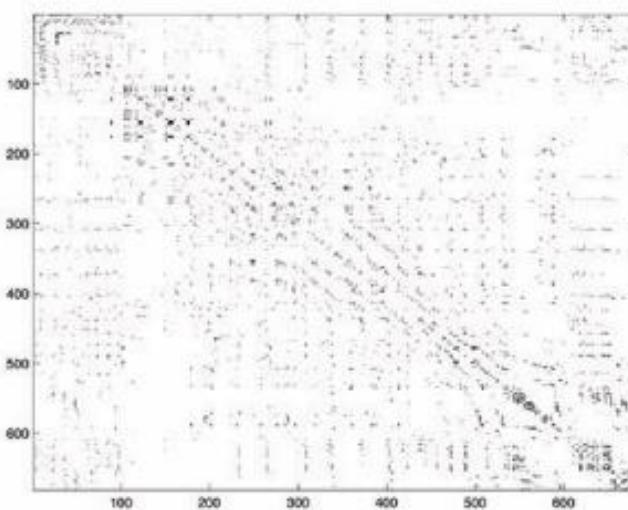
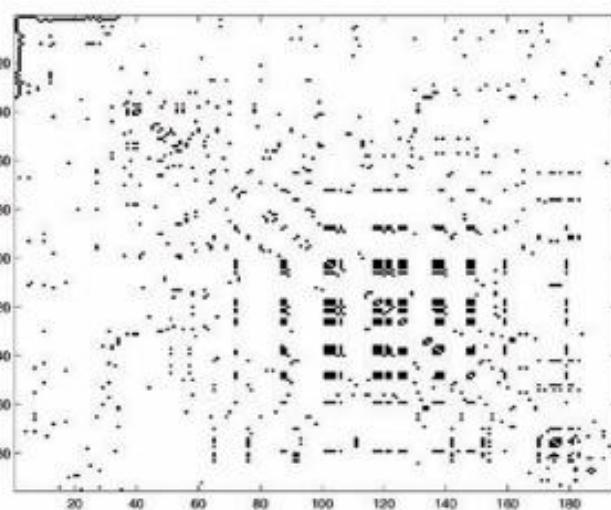
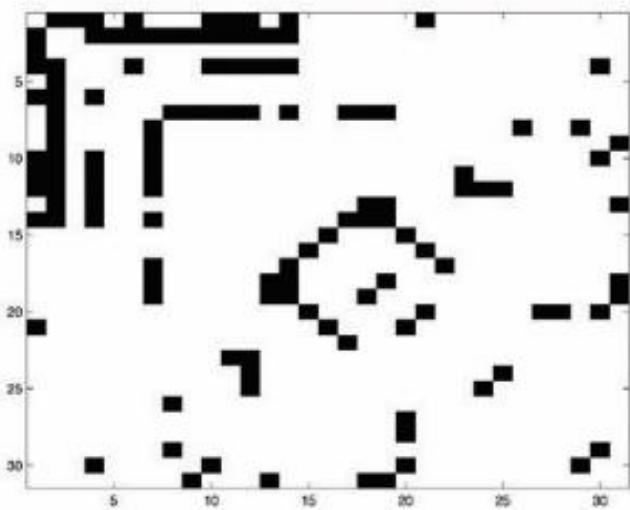
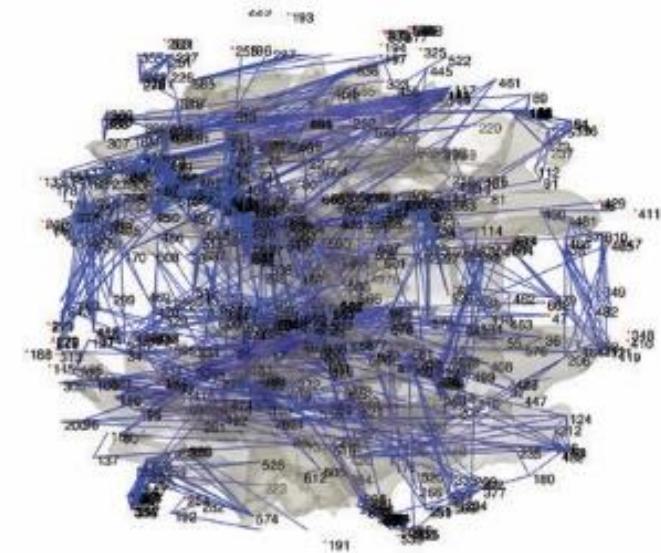
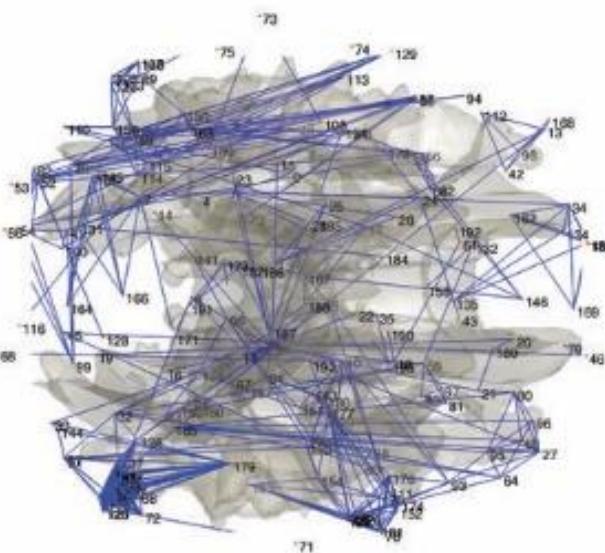
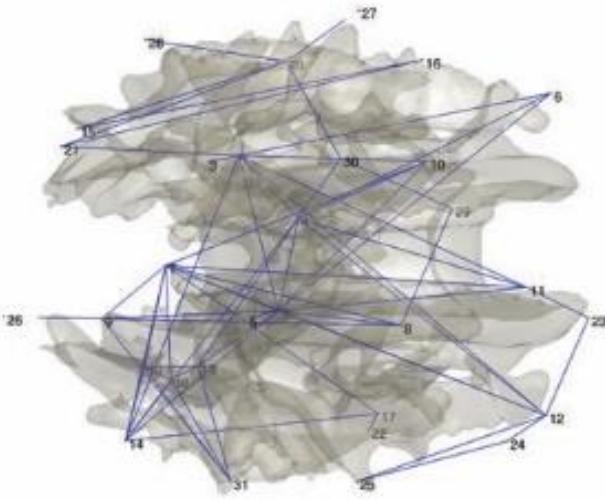
<id>: 183 hadOccupation: Poet,Painter,Printmaker name: William Blake

Name 3 different ways of representing graph data in a database system.

Name 3 different ways of representing graph data in a database system.



Adjacency Matrix



[<http://brainimaging.waisman.wisc.edu/~chung/graph/admatrix.jpg>]

Compress Sparse Row (CSR)

	0	1	2	3
0	a		c	
1	b			e
2	d	i	f	
3		h	g	

Position for row # in other two arrays:

Row position array:

#	0	1	2	3
	0	2	4	7

Column index array:

#	0	1	2	3	4	5	6	7	8
	0	2	1	3	0	1	2	1	2
	a	c	b	e	d	i	f	h	g

Cell value array:

Representations: Adjacency List

Source vertex with outgoing edges...

0	a	1	c	2		3	
1		b			e		
2	d	i	f				
3		h	g				

...without edge labels

0 -> (0,2)
1 -> (1,3)
2 -> (0,1,2)
3 -> (1,2)

...with edge labels

0 -> ([0,a],[2,c])
1 -> ([1,b],[3,e])
2 -> ([0,d],[1,i],[2,f])
3 -> ([1,h],[2,g])

...with edge properties

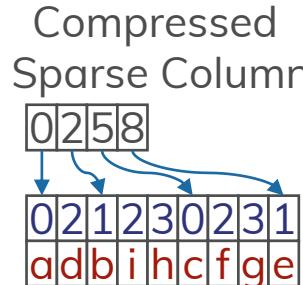
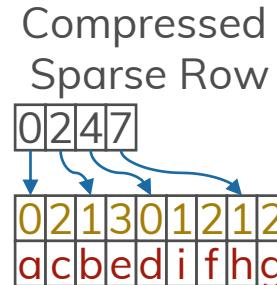
0 -> ([0,a,(weight=4]],[2,c,(weight=3)])
1 -> ([1,b,(weight=3]),[3,e,(weight=2)])
2 -> ([0,d,(weight=5]),[1,i,(weight=2]),...)
3 -> ([1,h,(weight=9]),[2,g,(weight=7)])

The same!

Almost the same!

source-oriented

target-oriented



Property Table Approaches

Triple Table

Subj.	Prop.	Obj.
ID1	type	BookType
ID1	title	“XYZ”
ID1	author	“Fox, Joe”
ID1	copyright	“2001”
ID2	type	CDType
ID2	title	“ABC”
ID2	artist	“Orr, Tim”
ID2	copyright	“1985”
ID2	language	“French”
ID3	type	BookType
ID3	title	“MNO”
ID3	language	“English”
ID4	type	DVDType
ID4	title	“DEF”
ID5	type	CDType
ID5	title	“GHI”
ID5	copyright	“1995”
ID6	type	BookType
ID6	copyright	“2004”

(Clustered) property table

Property Table

Subj.	Type	Title	copyright
ID1	BookType	“XYZ”	“2001”
ID2	CDType	“ABC”	“1985”
ID3	BookType	“MNP”	NULL
ID4	DVDType	“DEF”	NULL
ID5	CDType	“GHI”	“1995”
ID6	BookType	NULL	“2004”

Left-Over Triples

Subj.	Prop.	Obj.
ID1	author	“Fox, Joe”
ID2	artist	“Orr, Tim”
ID2	language	“French”
ID3	language	“English”

Property-Class Table

Class: BookType

Subj.	Title	Author	copyright
ID1	“XYZ”	“Fox, Joe”	“2001”
ID3	“MNP”	NULL	NULL
ID6	NULL	NULL	“2004”

Class: CDType

Subj.	Title	Artist	copyright
ID2	“ABC”	“Orr, Tim”	“1985”
ID5	“GHI”	NULL	“1995”

Left-Over Triples

Subj.	Prop.	Obj.
ID2	language	“French”
ID3	language	“English”
ID4	type	DVDType
ID4	title	“DEF”

Reduce numbers of subject-subject self joins necessary to reconstruct entities

Which of the following are distance measures in graphs?

Degree

Eccentricity

Diameter

Radius

Which of the following are distance measures in graphs?

Degree

Eccentricity

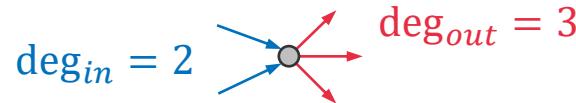
Diameter

Radius

Basic Concepts: Measures

Degree (Valency)

Degree of a vertex v



$$\deg(v) = \deg_{\text{out}}(v) + \deg_{\text{in}}(v)$$

Degree of a graph G

$$\deg(G) = \max_{v \in V} \deg(v)$$

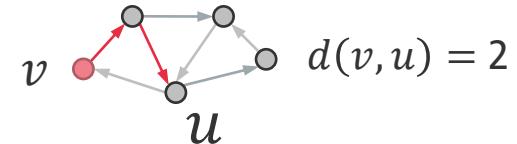
Degree distribution

→ Probability distribution of vertex degree in a graph



Distance

→ Number of edges in a shortest path connecting two vertices



Average Distance

→ Average shortest path
→ Average eccentricity of any vertex in the graph

Eccentricity



→ Longest shortest path starting from v



Radius

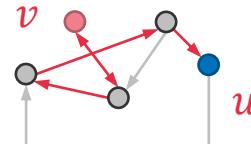
→ Minimum eccentricity of any vertex in the graph

$$r(G) = 2$$



Diameter

→ Maximum eccentricity of any vertex in the graph



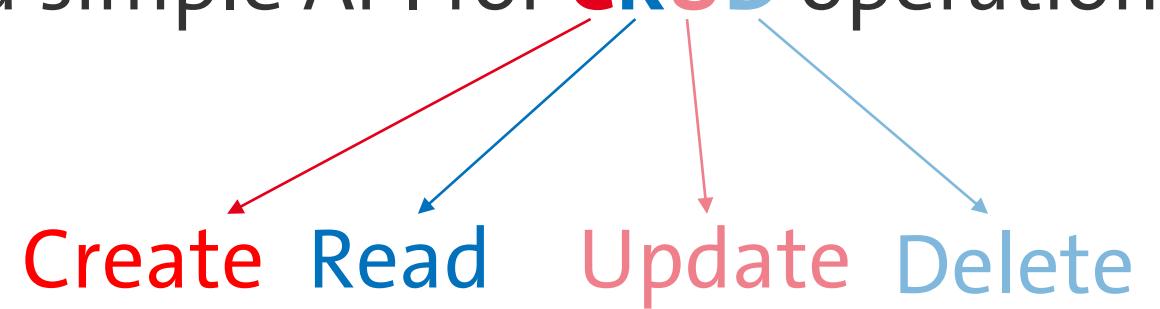
$$d(G) = 4$$

What do the letters CRUD (in relation to NoSQL DBs) stand for?

What do the letters CRUD (in relation to NoSQL DBs) stand for?

Key value store:

Basic key-value mapping with a simple API for **CRUD** operations



Which operations are offered by every key-value store?

Put

Get

Merge

Delete

Which operations are offered by every key-value store?

Put

Get

Merge

Delete

Key-Value Stores

Basic key-value mapping with a simple API for **CRUD** operations



Horizontal partitioning

users:1:a	4711
users:1:b	“[12, 34, 45, 67, 89]”

users:2:a	0110101001011001010101001...
users:2:b	“[12, ABC, 3212, 0xff]”

CRUD realized by at least 3 types of queries

Put: Add a new pair
Get: Retrieve a pair
Delete

Additional Operators implemented by some systems

Merge
MultiGet/MGet
MSet
...

Which are the variants for merging levels in an LSM tree?

Which are the variants for merging levels in an LSM tree?

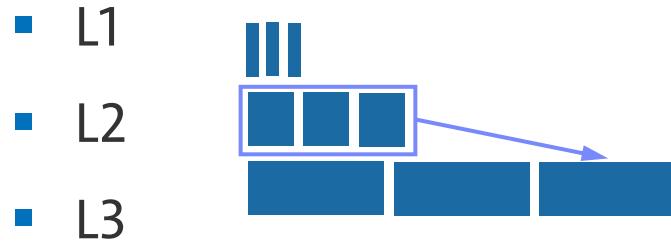
Tiering

Leveling

LSM Trees: Merging

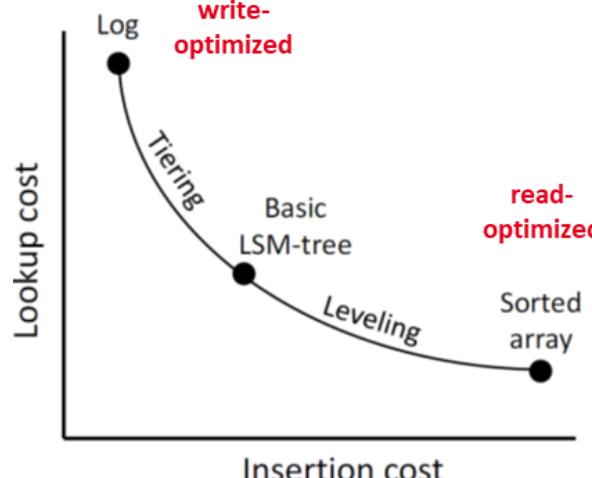
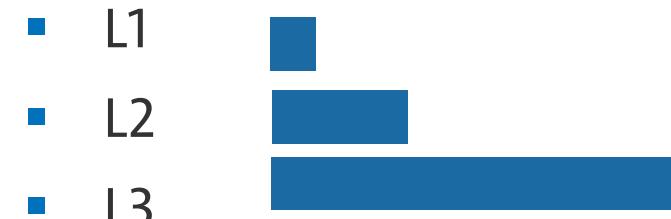
LSM Tiering

- Keep up to $T-1$ (sorted) runs per level L
- Merge all runs of L_i into 1 run of L_{i+1}

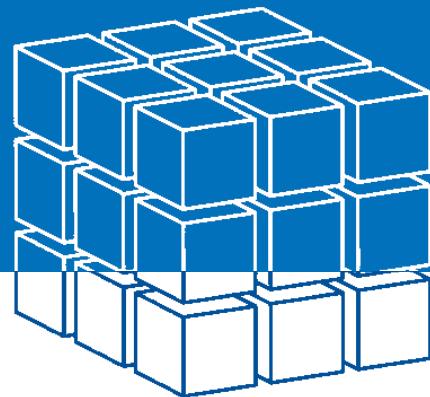


LSM Leveling

- Keep 1 (sorted) run per level L
- Sort-Merge run of L_i with L_{i+1}



Data Warehouses and OLAP



Which of the following are characteristics of Data Warehouses?

Classical

Integration of data from different source systems

No user updates

Transaction-oriented organization of data

Historic data

Analysis-oriented organization of data

Real time data



Which of the following are characteristics of Data Warehouses?

Classical

Integration of data from different source systems

No user updates

Transaction-oriented organization of data

Historic data

Analysis-oriented organization of data

Real time data



Characteristics of DWHs

Analysis-oriented organization of data

Domain-oriented, models a specific application goal

No user updates

(almost) no updates and deletes (technical updates / deletes only, quality assurance)

Integration of data from different source systems

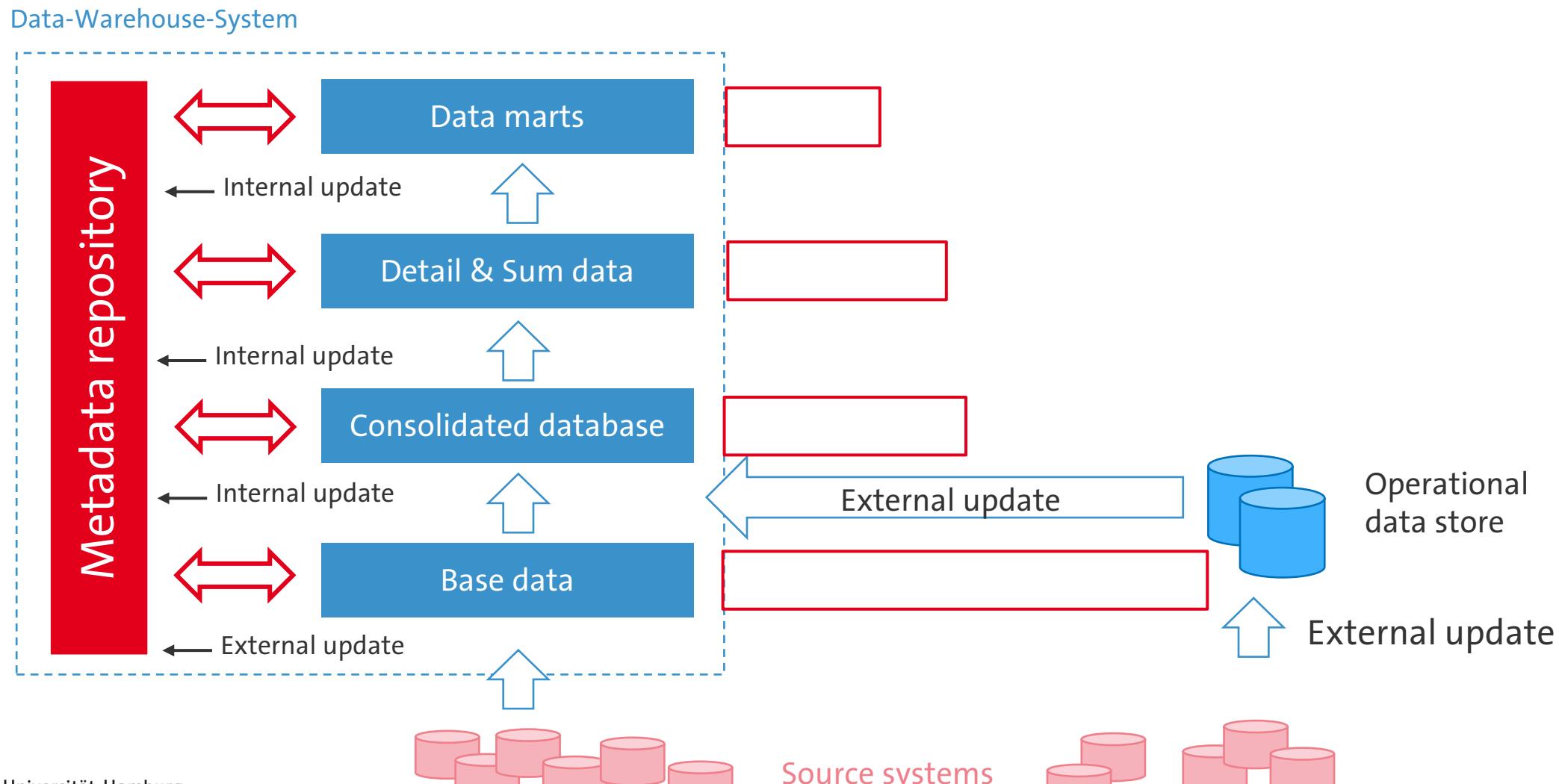
Integrated database, integration on structural and data level of multiple databases

Historic data

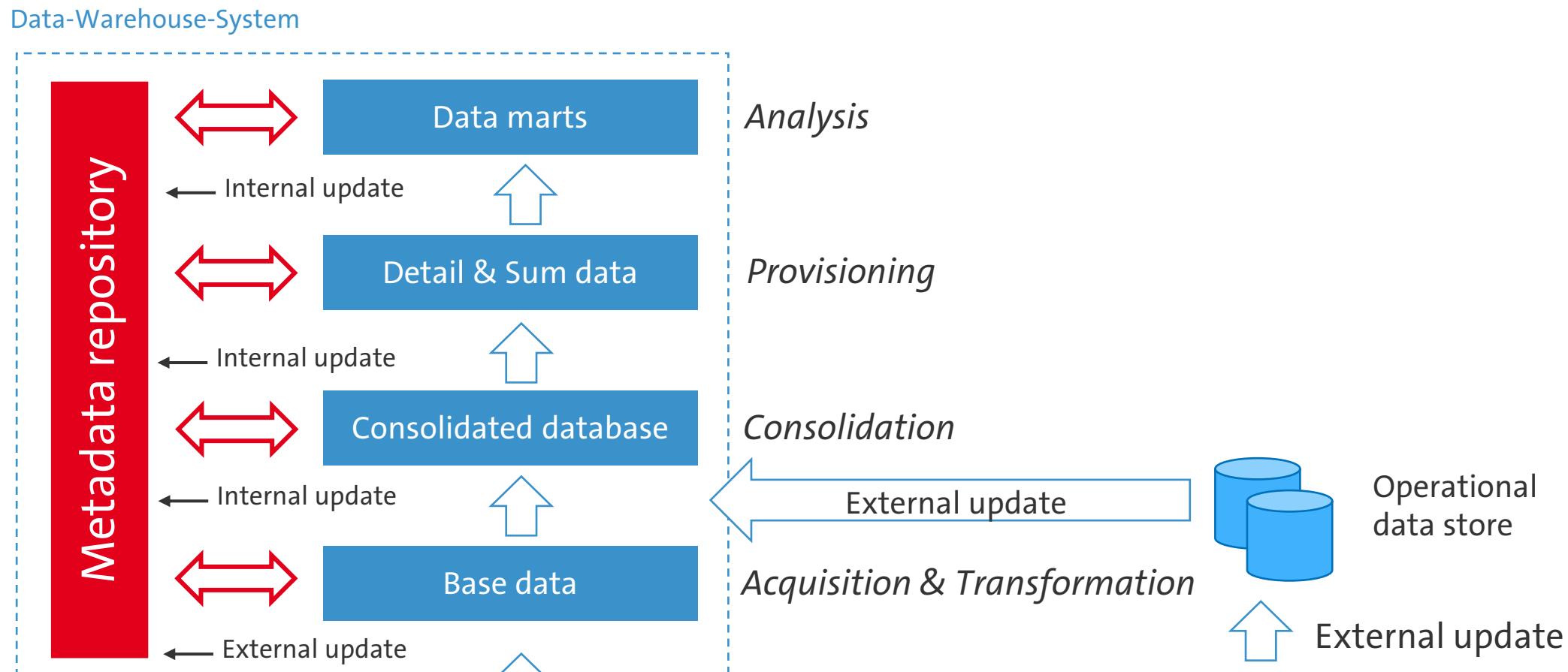
Data is kept over a long period of time



Fill in the steps involved in creating the Data Warehouse

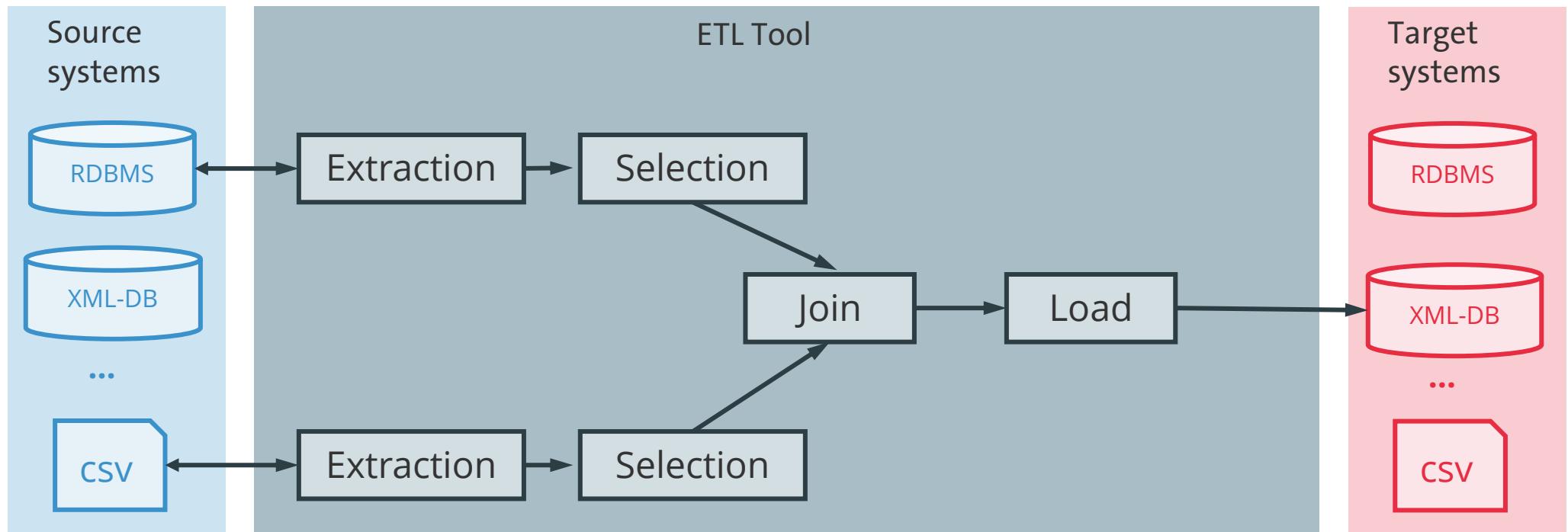


Fill in the steps involved in creating the Data Warehouse



What happens during the ETL step?

Data acquisition and transformation



Which of the following statements about ADAPT are wrong?

Different hierarchies of the same dimension can have different leaf nodes.

Attributes can be assigned to a whole cube, a dimension, and individual levels.

Self-precedence can be modelled

A cube must have exactly 3 dimensions.



Which of the following statements about ADAPT are wrong?

Different hierarchies of the same dimension can have different leaf nodes.

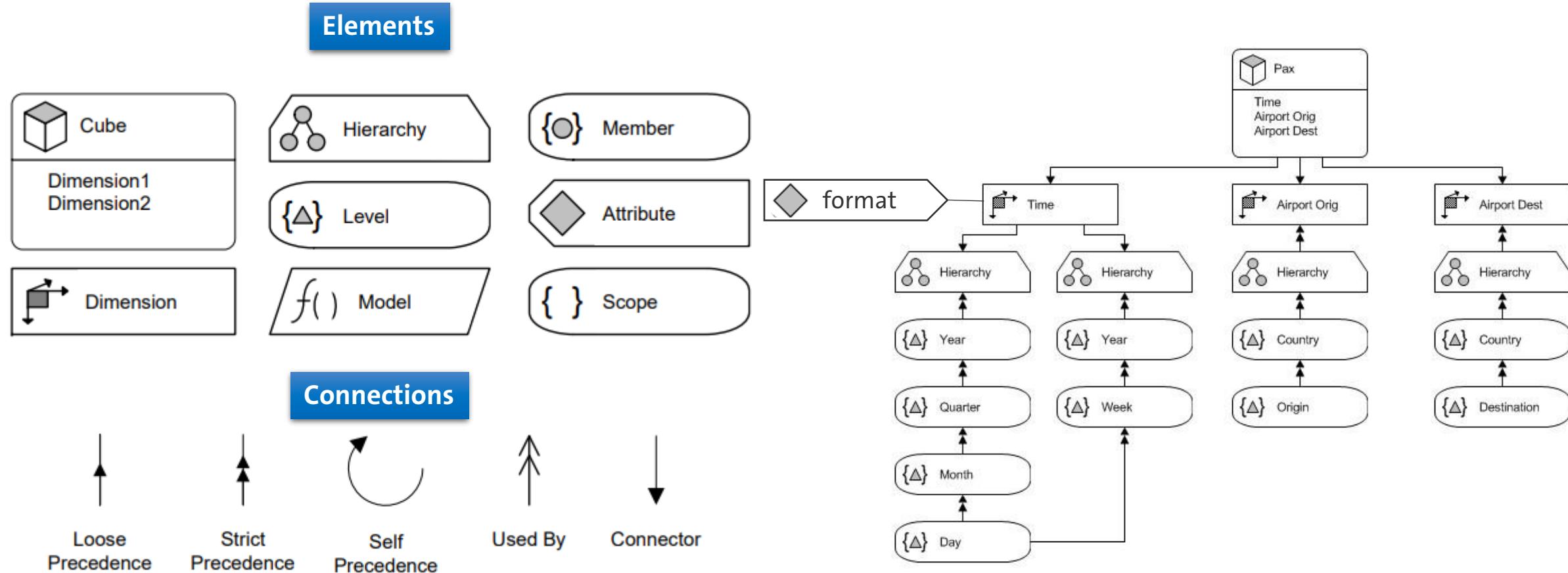
Attributes can be assigned to a whole cube, a dimension, and individual levels.

Self-precedence can be modelled

A cube must have exactly 3 dimensions.



Conceptual Models: ADAPT



Name two relational schemas typically used in OLAP and briefly explain them.



Name two relational schemas typically used in OLAP and briefly explain them.

Star schema

- One fact table and multiple dimension tables
- Dimension tables are not necessarily normalized
- Fact table is linked directly to all dimension tables via a foreign key

Snowflake schema

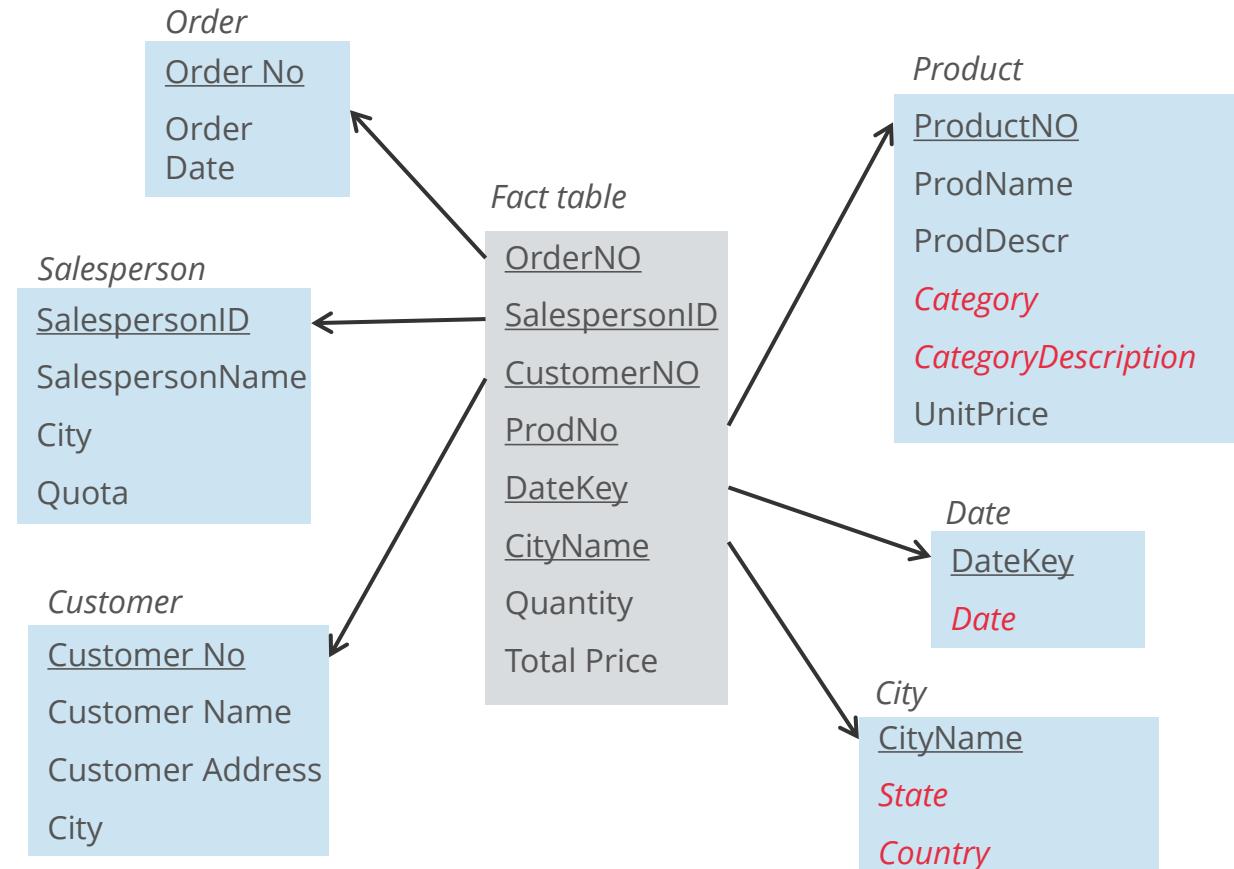
- One fact table and multiple dimension tables
 - Smaller memory footprint than star schema
- Dimension tables are normalized
 - Introduces more joins in the queries

Galaxy schema

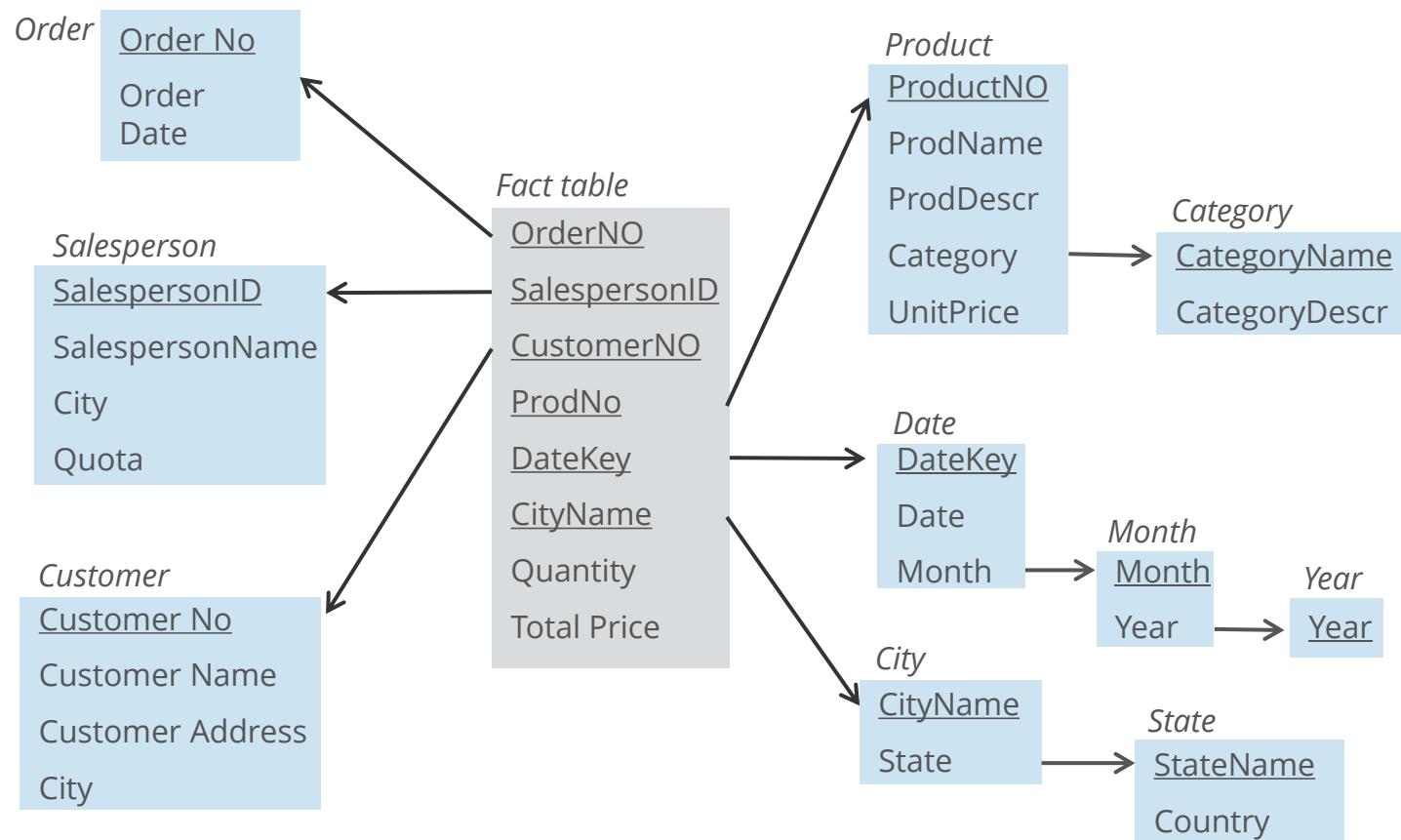
- Multiple fact tables
- Dimension tables are shared
- Joining fact tables (which are usually huge) is extremely inefficient
 - Long query execution times



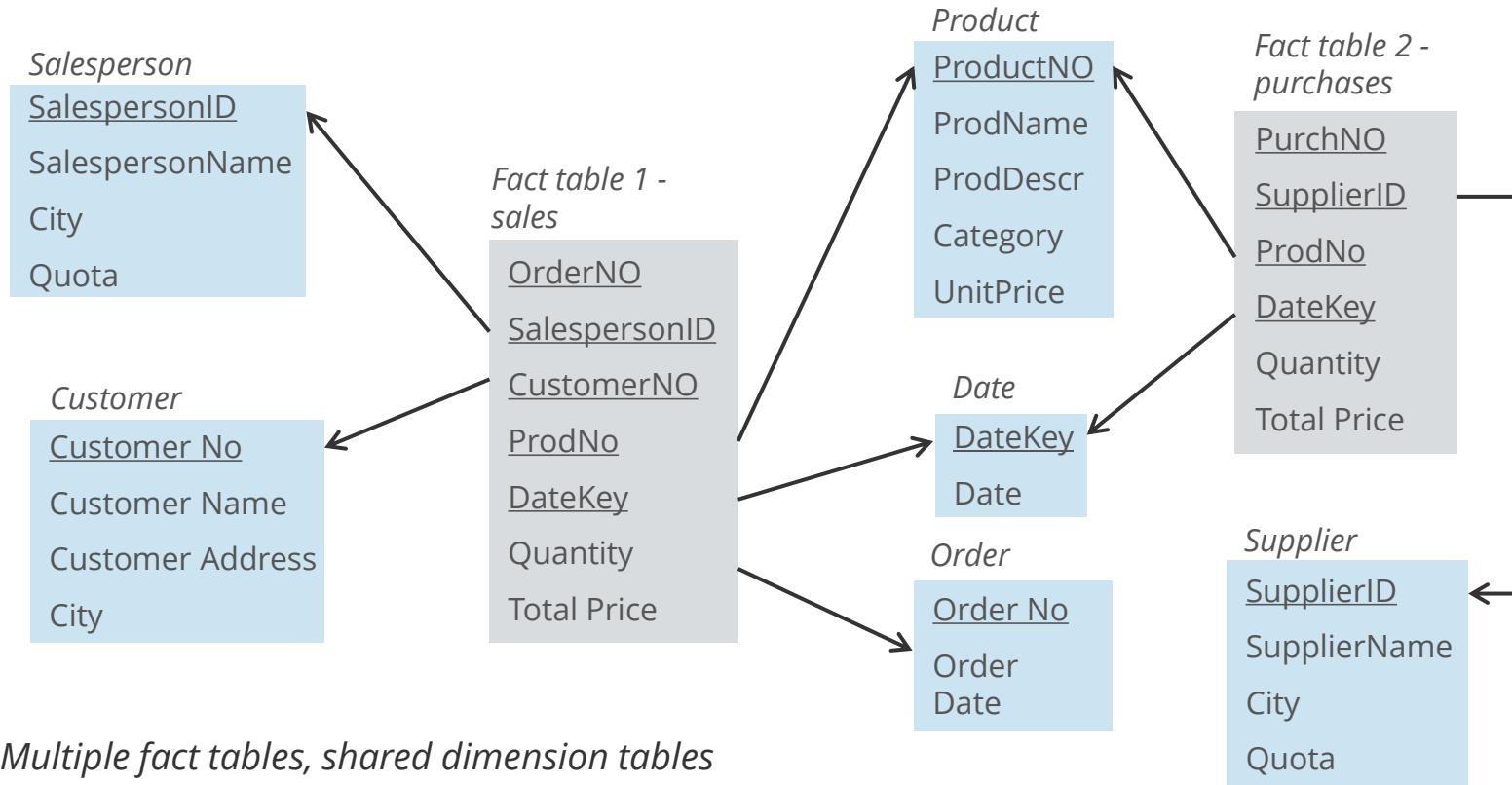
Star Schema



Snowflake Schema



Galaxy Schema



Which of the following is/are not (a) typical method(s) of relational mapping of hierarchies?

Horizontal Mapping

Recursive Vertical
Mapping

Diagonal Mapping

Iterative Horizontal
Mapping

Vertical Mapping

Random Mapping



Which of the following is/are not (a) typical method(s) of relational mapping of hierarchies?

Horizontal Mapping

Recursive Vertical
Mapping

Diagonal Mapping

Iterative Horizontal
Mapping

Vertical Mapping

Random Mapping



Relational Mapping of Hierarchies

Vertical Mapping

Product			Product group			Product family			Category	
ID	Product	ID2	ID	Product group	ID3	ID	Product family	ID4	ID	Category
1	Flour X	1	1	Flour	1	1	Bakery goods	1	1	Food
2	Sugar Y	2	2	Sugar	1	2	Beverages	1		
3	Water Z	3	3	Water	2					

Horizontal Mapping

ID	Product	Product group	Product family	Category
1	Flour X	Flour	Bakery goods	Food
2	Sugar Y	Sugar	Bakery goods	Food
3	Water Z	Water	Beverages	Food

Recursive Vertical Mapping

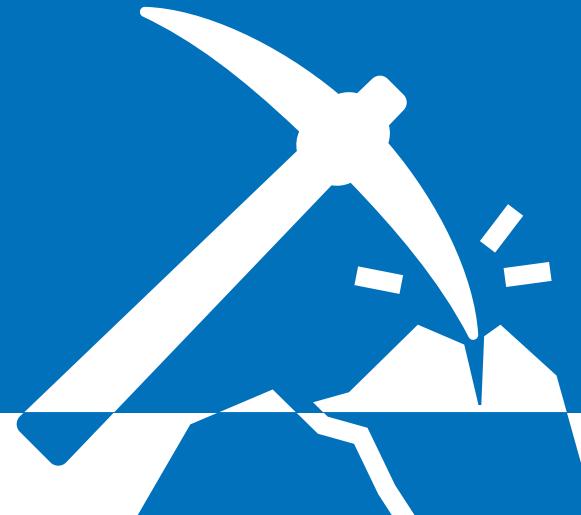
ID	Product	Product group	Product family	Category
1	Flour X	Flour	Bakery goods	Food
2	Sugar Y	Sugar	Bakery goods	Food
3	Water Z	Water	Beverages	Food

Relation product

ID	Product	ParentID
1	Food	NULL
2	Bakery goods	1
3	Beverages	1
4	Flour	2
5	Sugar	2
6	Water	3
7	Flour X	4
8	Sugar Y	5
9	Water Z	6

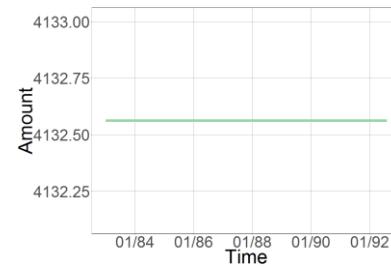


Data Mining

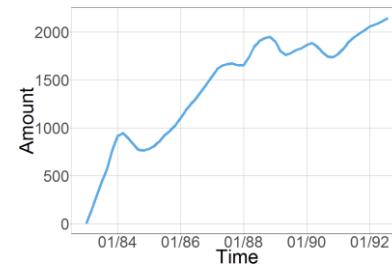


Into which components can time series typically be decomposed?

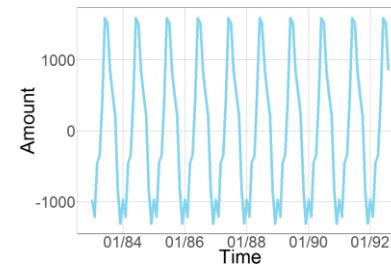
Into which components can a time series typically be decomposed?



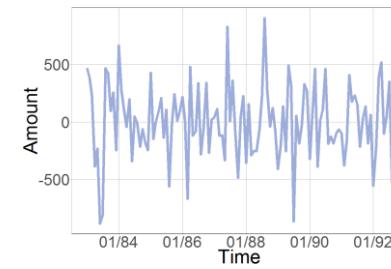
+



+



+



Base

Trend

Season

Residuals

Is Piecewise aggregate approximation (PAA) a shape-based or a feature based representation for time series?

Shape-based

Feature-based

What is the difference to symbolic aggregate approximation?

Is Piecewise aggregate approximation (PAA) a shape-based or a feature based representation for time series?

Shape-based

Feature-based

What is the difference to symbolic aggregate approximation?

In SAX, values are discretized into an alphabet.

Shape-based Representation

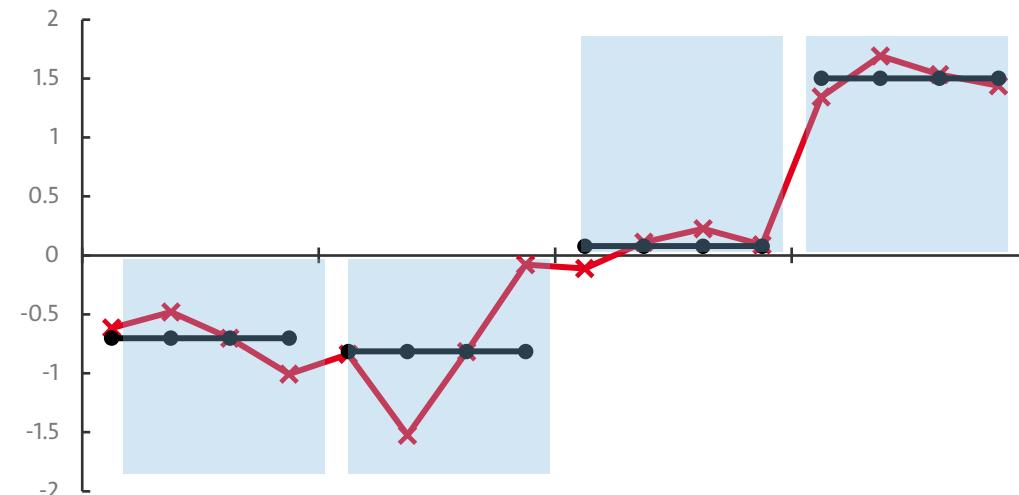
- Transform a series to segments
- Time-dependent representation
 - High compression



Piecewise aggregate approximation (PAA)
→ Represent each segment by its mean



Symbolic aggregate approximation (SAX)
→ Discretize the mean into an alphabet



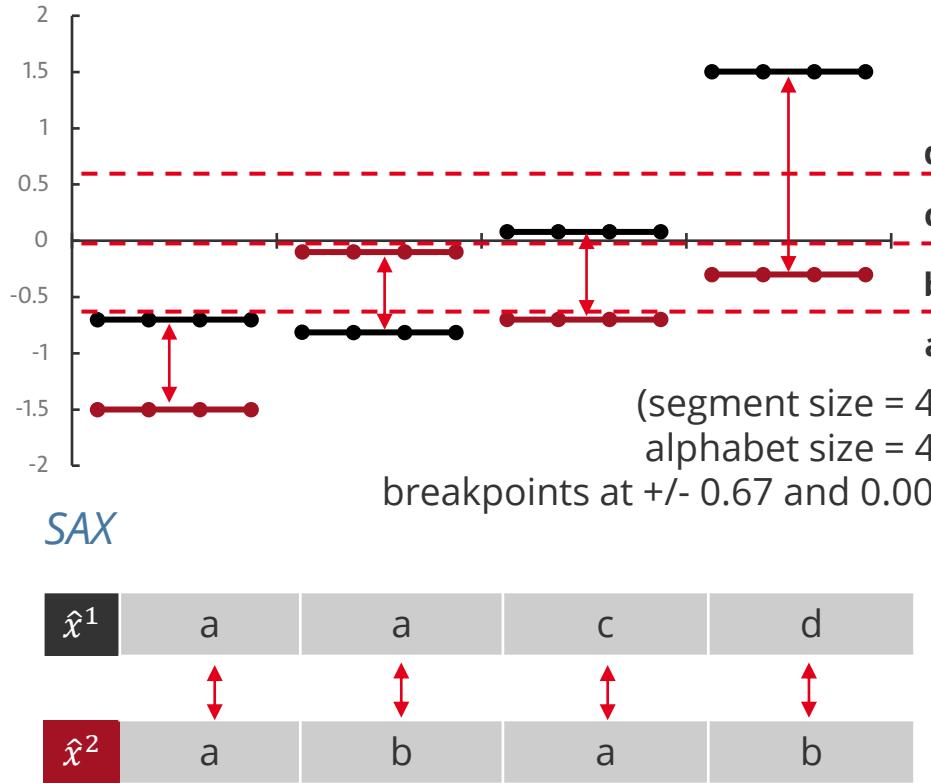
PAA: -0.70 -0.81 0.08 1.56

SAX: b b a a

(segments per series = 4, alphabet size = 2, breakpoint at 0.0)

The SAX Distance

Segments



$$d_{SAX}(\hat{x}^i, \hat{x}^j) = \sqrt{T/W} \cdot \sqrt{\sum_{w=1}^W \text{cell}(\hat{x}_w^i, \hat{x}_w^j)^2}$$

cell returns the minimum distance of two symbols:

	a	b	c	d
a	0	0	0.67	1.34
b	0	0	0	0.67
c	0.67	0	0	0
d	1.34	0.67	0	0

PCA

Standardize all values per dimension

value-mean/standard deviation

$$\text{std} \begin{bmatrix} \text{gray} & \text{gray} & \text{gray} & \text{gray} \end{bmatrix} = \begin{bmatrix} \text{blue} & \text{blue} & \text{blue} & \text{blue} \end{bmatrix}$$

Standardized data set * Feature vector
= New Dataset (reduced to x dimensions)

$$\begin{bmatrix} \text{blue} & \text{blue} & \text{blue} & \text{blue} \end{bmatrix} * \begin{bmatrix} \text{red} & \text{dark gray} \end{bmatrix} = \begin{bmatrix} \text{pink} & \text{pink} \end{bmatrix}$$

Compute the covariance matrix

$$\text{cov}(x,y) = (\sum(x - (\text{mean}(x))(y - \text{mean}(y))) / \text{number of data points})$$

$$\text{cov} \begin{bmatrix} \text{blue} & \text{blue} & \text{blue} & \text{blue} \end{bmatrix} = \begin{bmatrix} \text{red} & \text{red} & \text{red} & \text{red} \end{bmatrix}$$

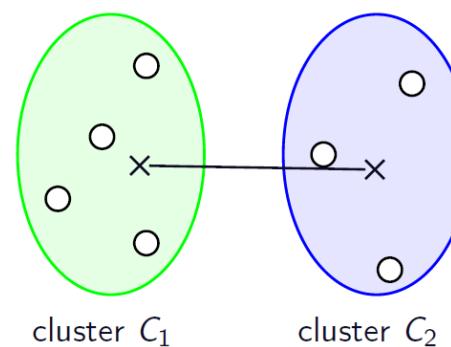
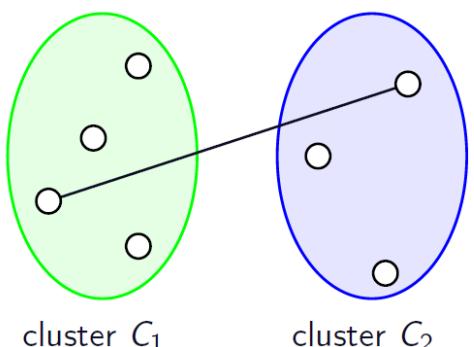
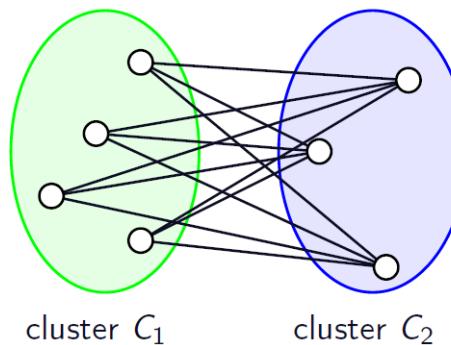
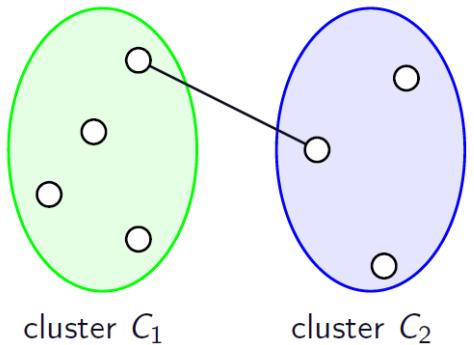
Compute Eigen values and Eigen vectors of the covariance matrix

$$\lambda \begin{bmatrix} \text{dark gray} & \text{dark gray} & \text{dark gray} \end{bmatrix} v = \lambda v \rightarrow v$$

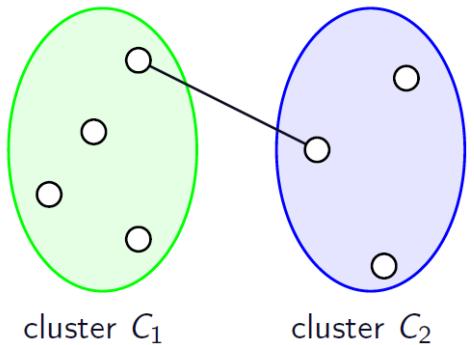
Sort elements of Eigenvectors in descending order
and pick those that belong to top x Eigen values
→ Feature Vectors

$$\text{sort} \downarrow \begin{bmatrix} \text{dark gray} & \text{dark gray} & \text{dark gray} \end{bmatrix}$$

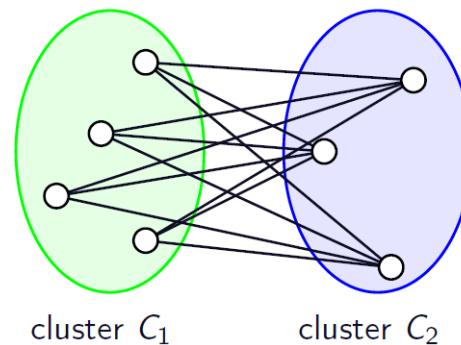
Which distance measures for clusters are shown here?



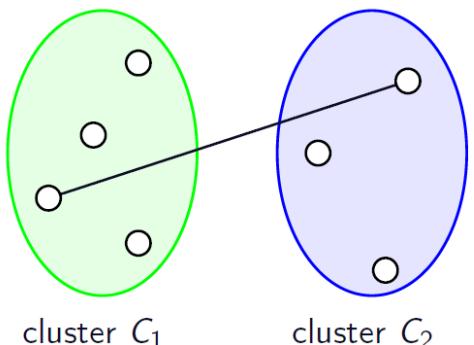
Which distance measures for clusters are shown here?



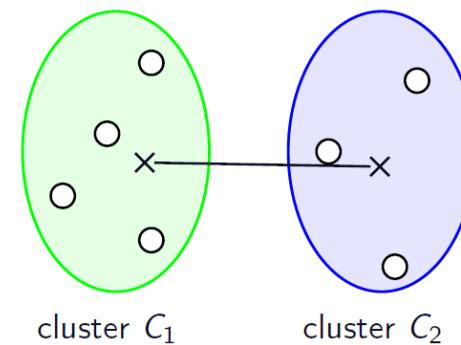
Single link



Average link



Complete link



Canonical entity

Clustering Methods

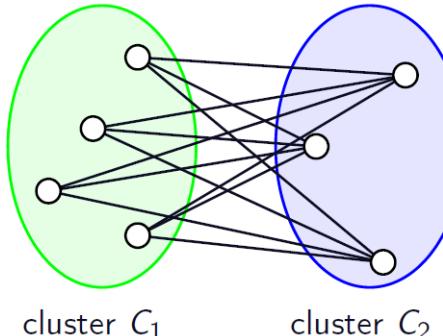
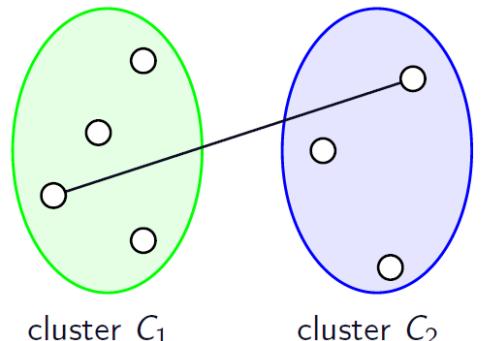
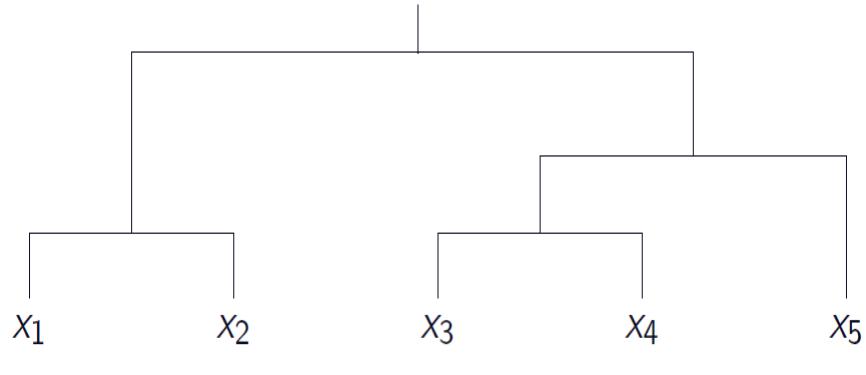
K-Means

Canopy Clustering

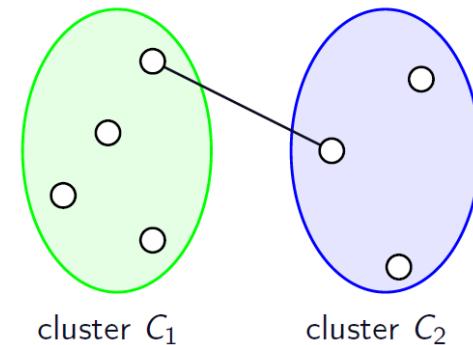
Hierarchical Clustering

Incremental Clustering

Results can be displayed as a dendrogram

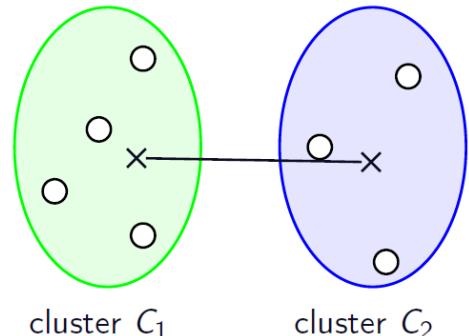


Distance measures for clusters

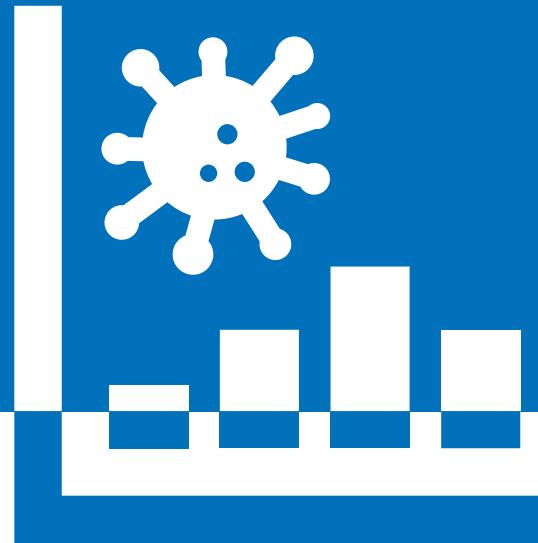


Single link

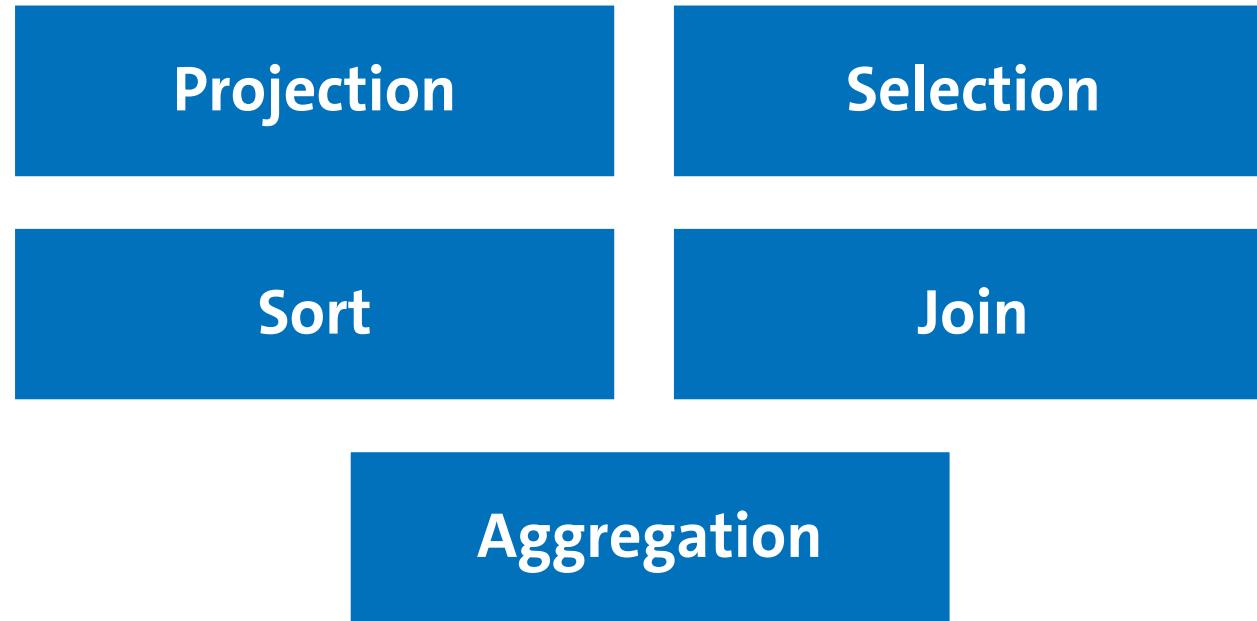
Minimal distance between
two data points



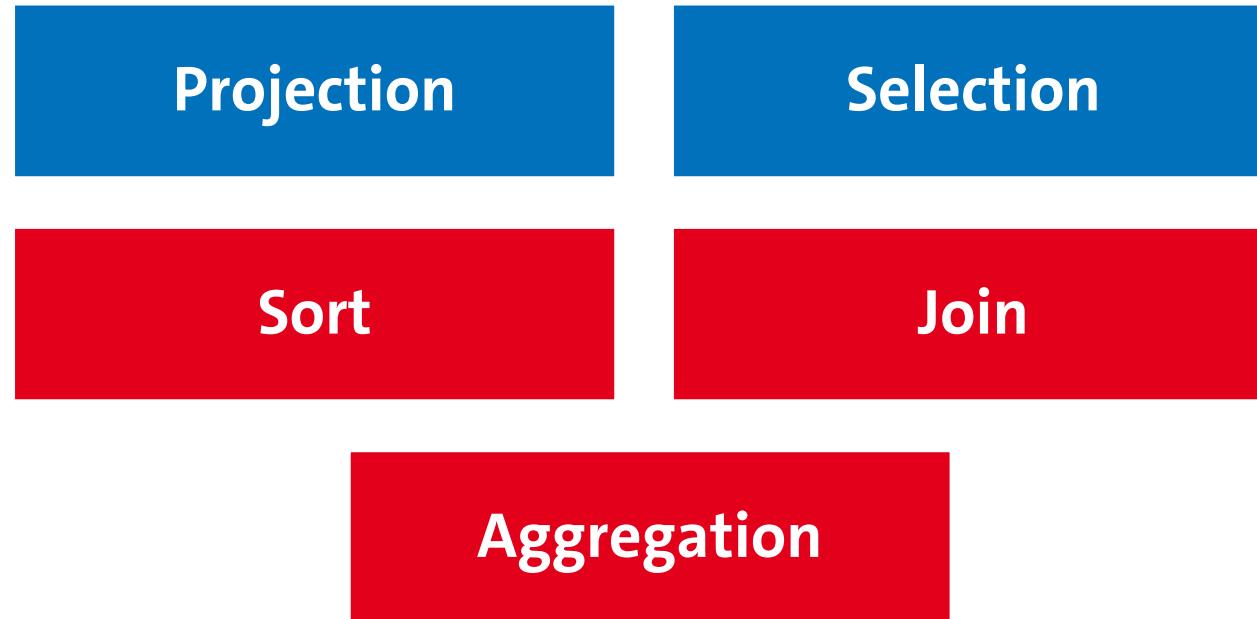
Big Data Analysis



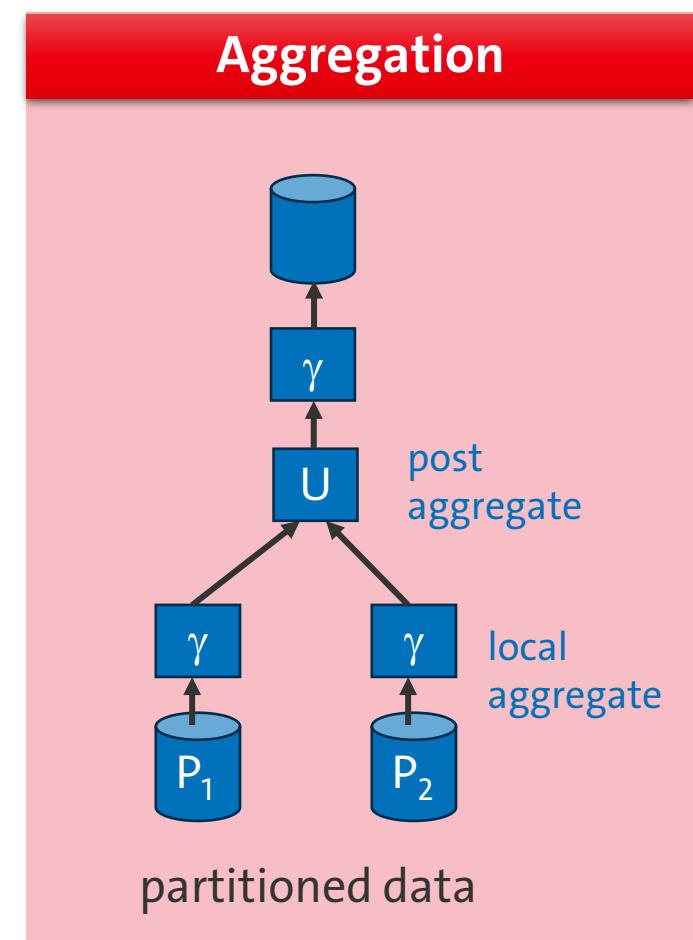
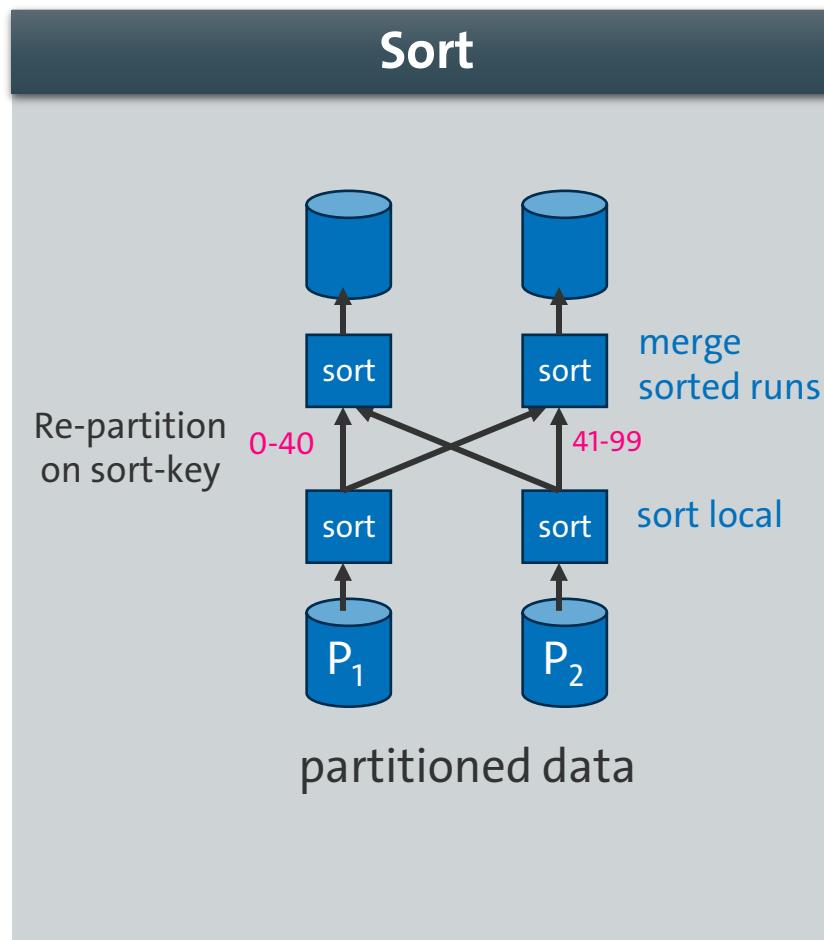
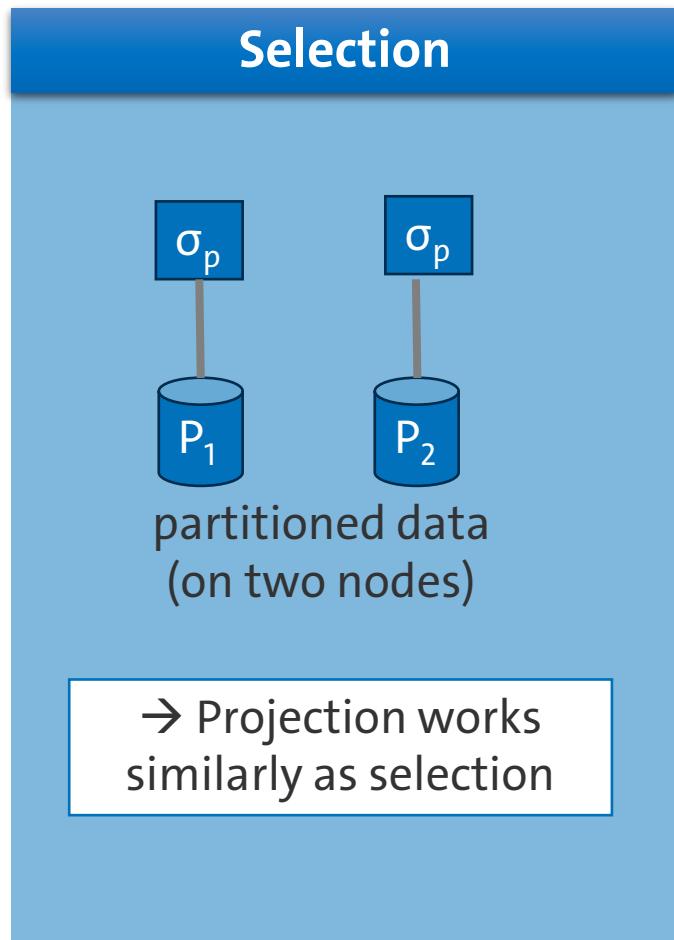
Which operators require shuffling if they are distributed?



Which operators require shuffling if they are distributed?



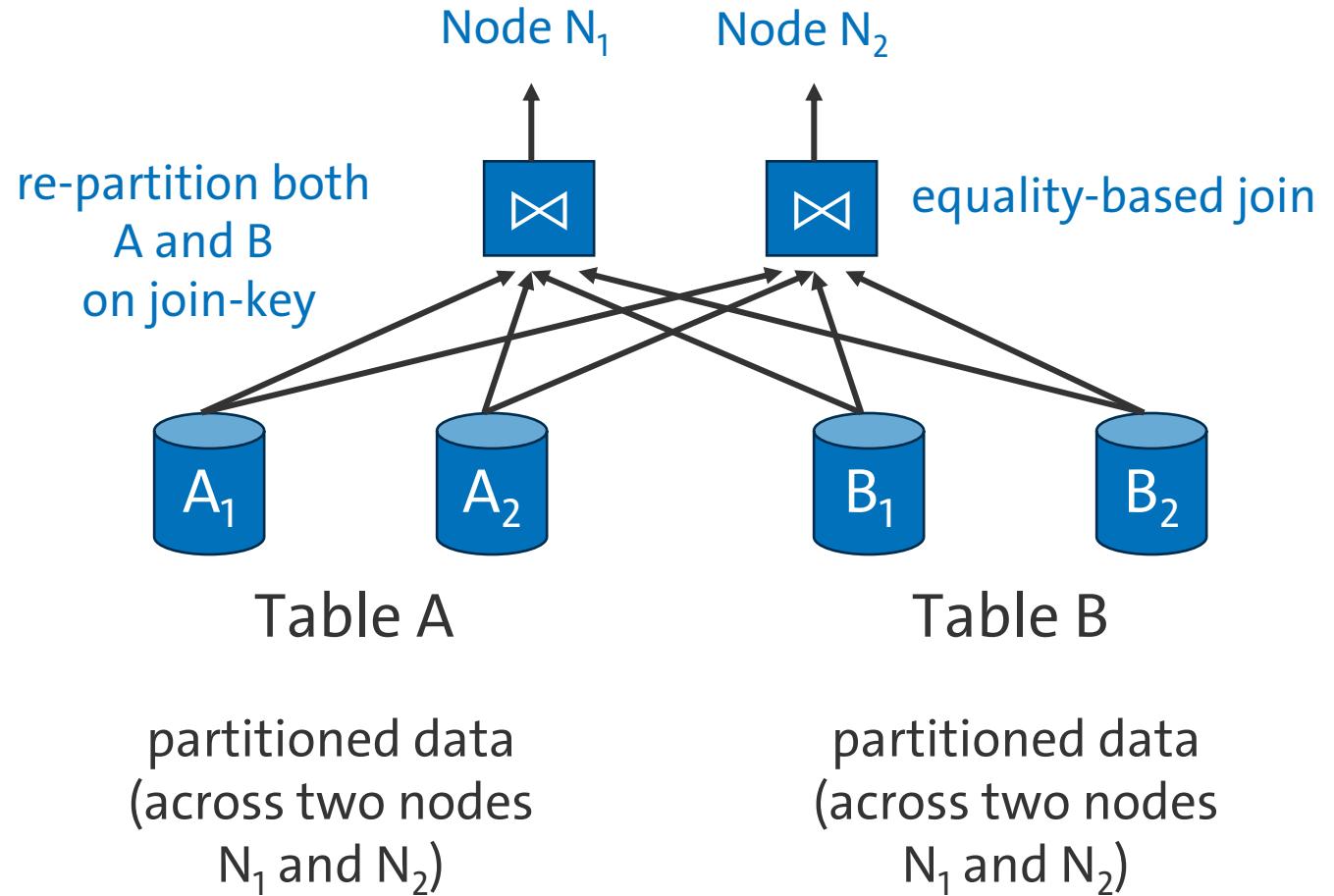
Parallel Operators



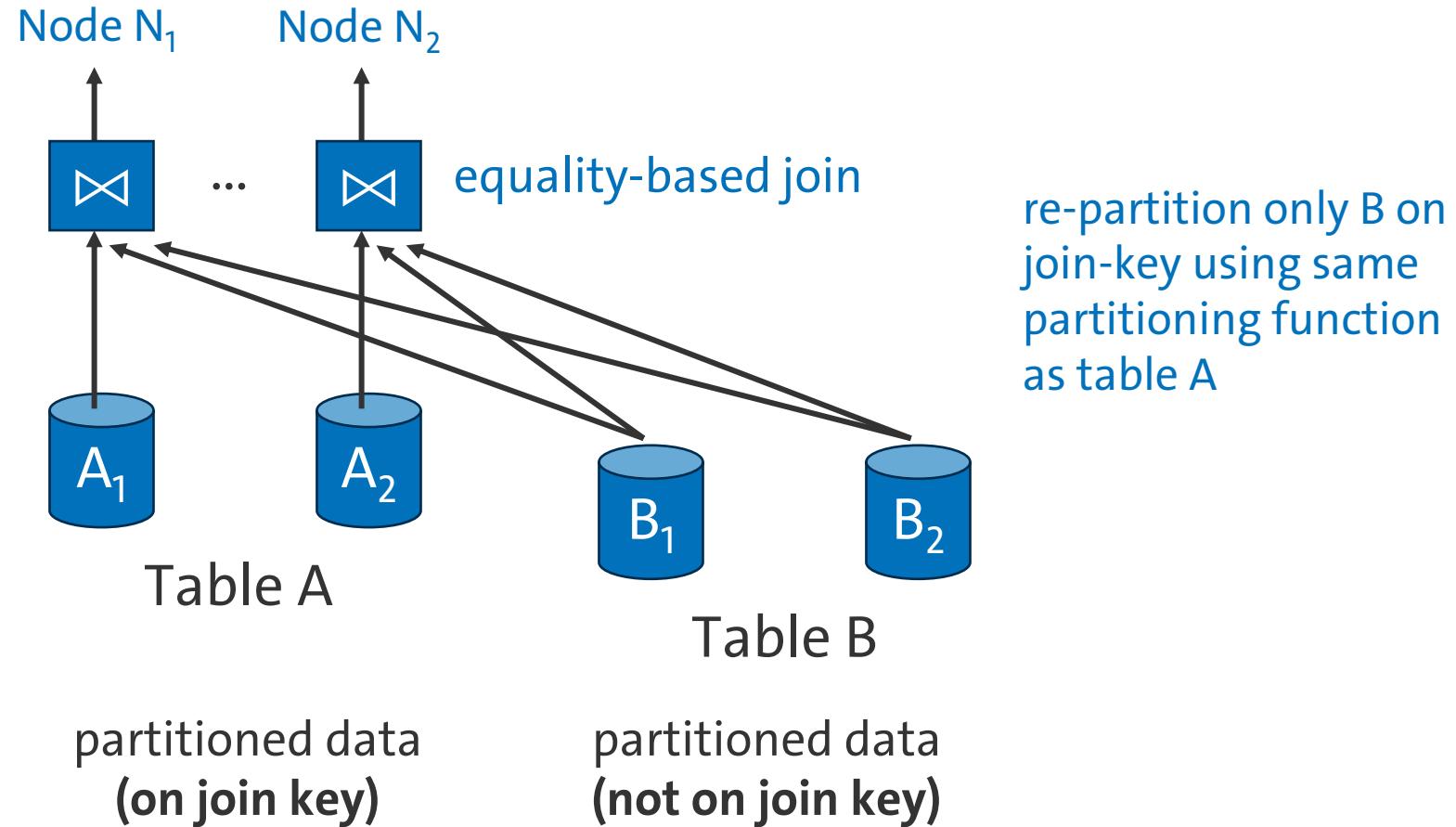
Join: Symmetric Repartitioning

„Repartitioned join“

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

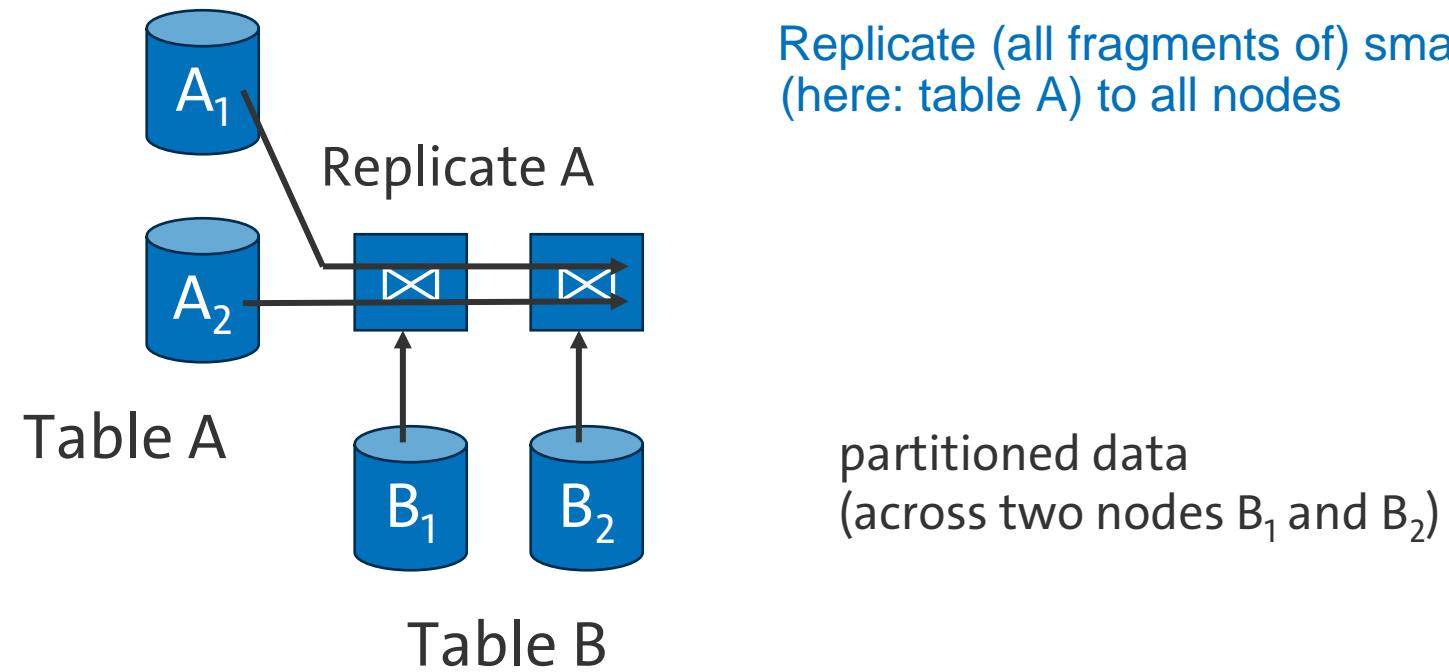


Join: Asymmetric Repartitioning



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

Which of the following are shuffling methods for distributed operators?

Range-based N:M

Hash-based N:M

Column-based N:M

1:1

N:1

Remote N:1

Which of the following are shuffling methods for distributed operators?

Range-based N:M

Hash-based N:M

Column-based N:M

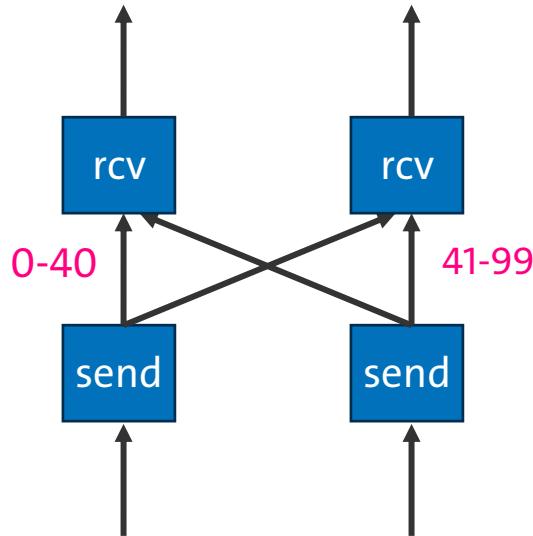
1:1

N:1

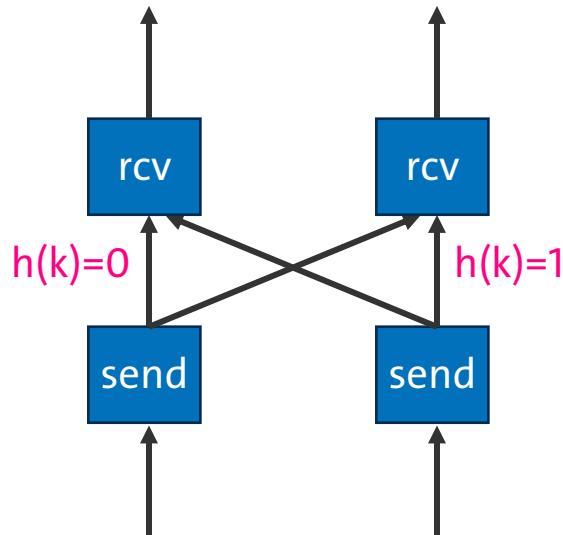
Remote N:1

Data Shuffling: Details

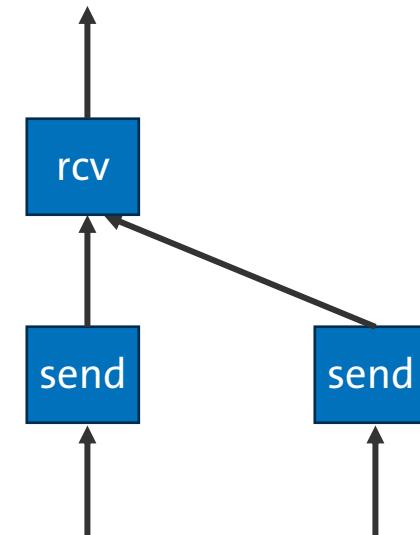
Data Shuffling can use different partitioning strategies (range vs. hash-partitioning, N:M vs. N:1) to re-partition data during query execution



Range-based N:M



Hash-based N:M



N:1 (no part. function)

Name 3 possible reasons for data skew

Name 3 possible reasons for data skew

Different partition sizes

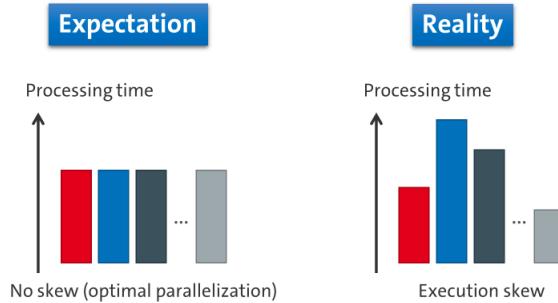
Distribution function leading to
different fragment sizes

Different hit rates per partition
in range queries

...

Name 3 possible reasons for data skew

The Data Skew



Observation: Unequal processing time of partial operations (execution skew) impairs parallelization

Execution skew often goes back to data skew:
Differently sized data sets per partial operation due to non-uniform distribution of attribute values und tuples



See handout!

What is the main difference between the Kappa architecture and the Lambda architecture?

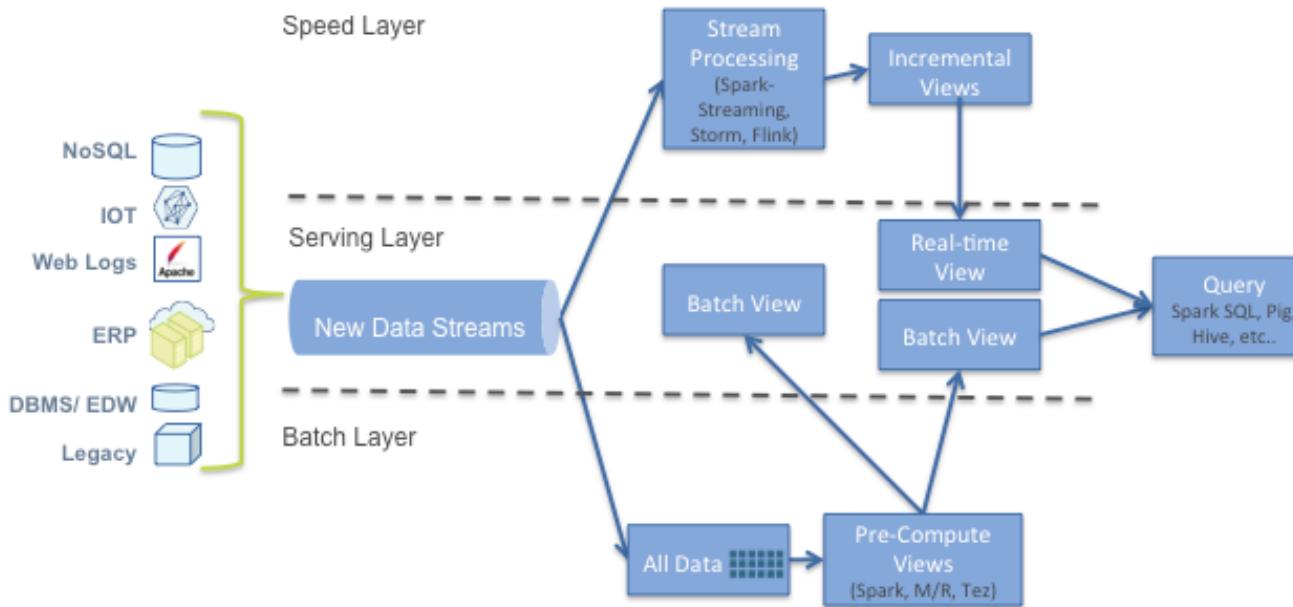
What is the main difference between the Kappa architecture and the Lambda architecture?

The Kappa architecture only has a speed layer while the Lambda architecture also has a batch layer

Lambda Architecture

→ see lecture 16

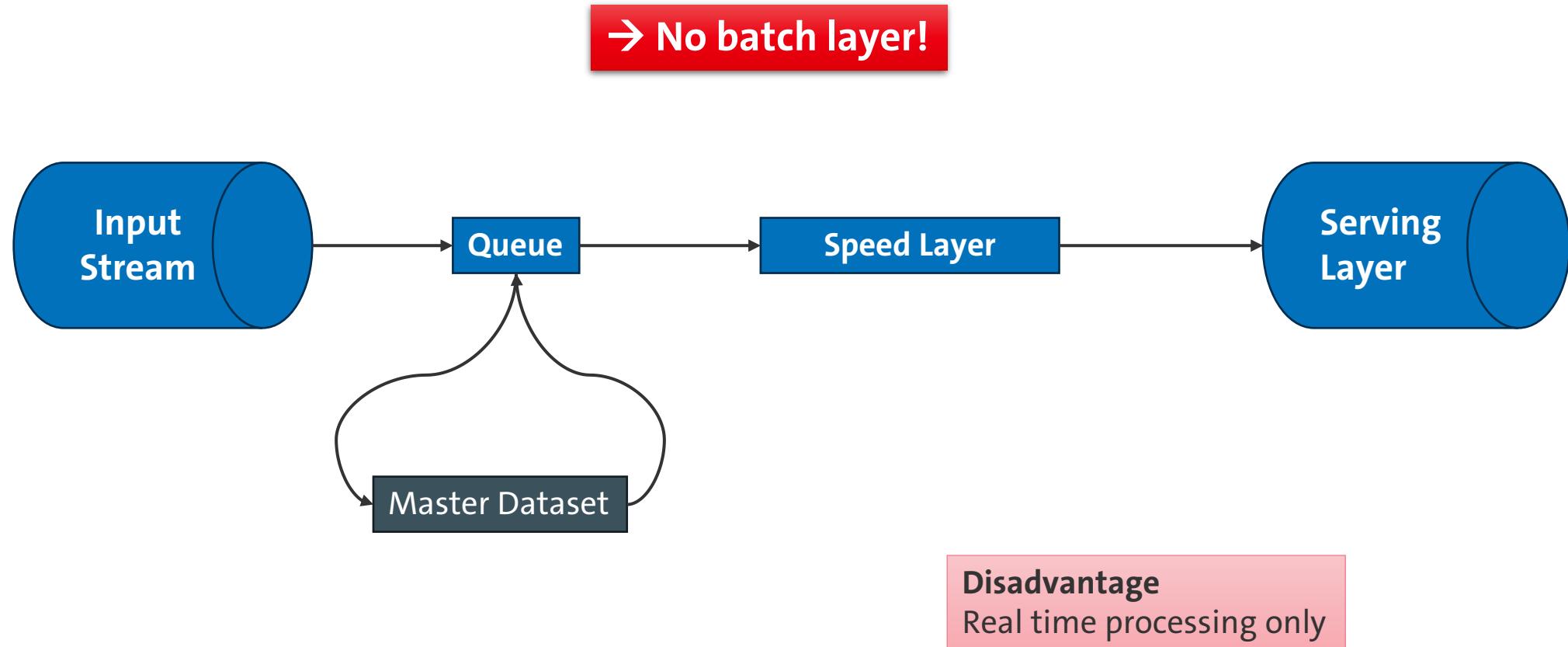
→ Batch & Stream processing



Disadvantage

2 code bases & 2 deployments, e.g. Hadoop & Storm

Kappa Architecture



Which functions must be implemented by the application developer when using a map-reduce framework?

Map

Shuffle

Reduce

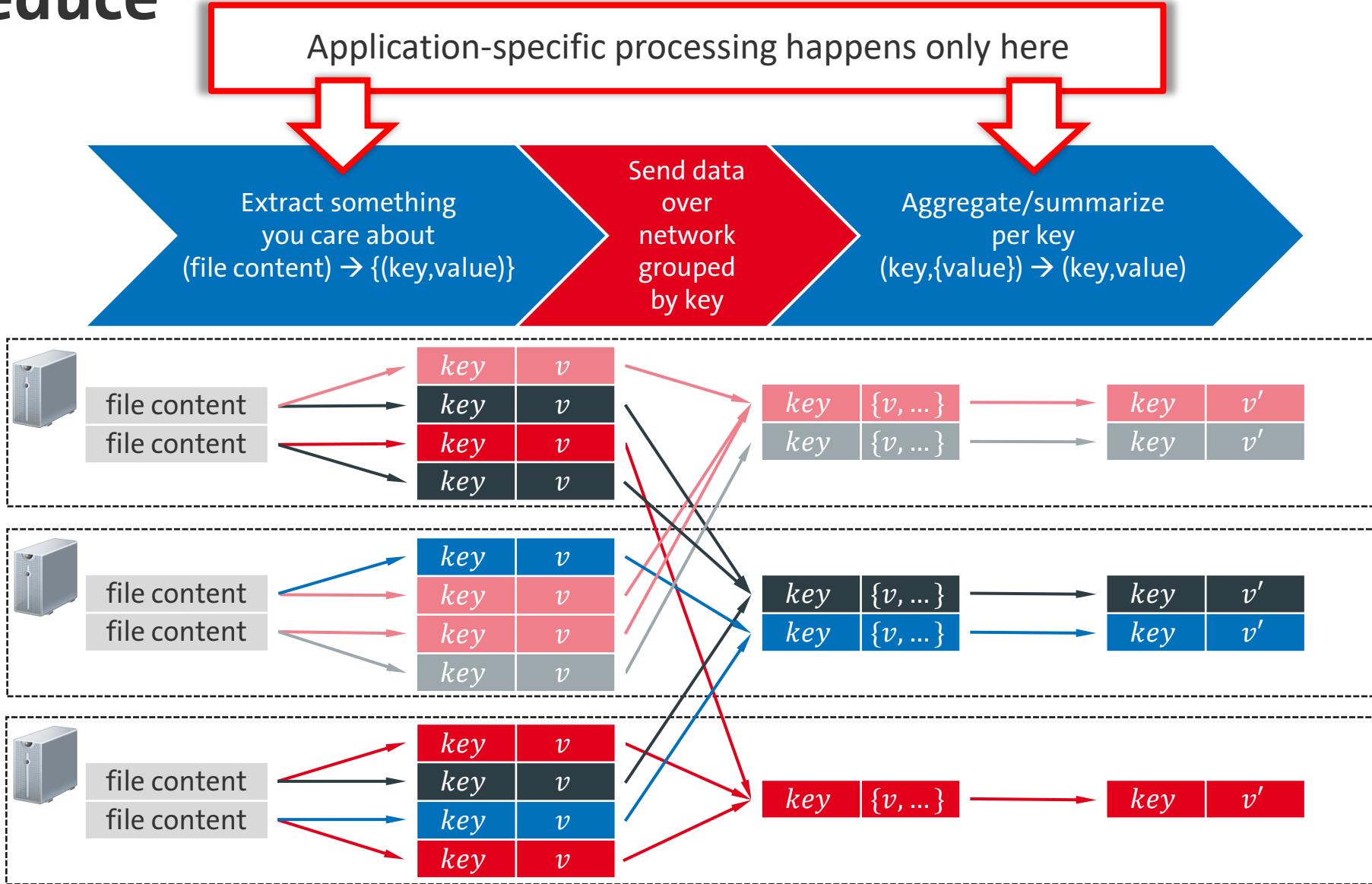
Which functions must be implemented by the application developer when using a map-reduce framework?

Map

Shuffle

Reduce

Map Reduce

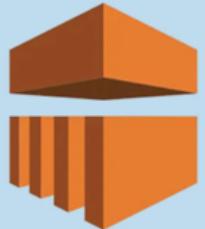


Name 3 frameworks for batch processing



Framework Overview

Batch



amazon
EMR

Spark



Flink

Stream



Samza

