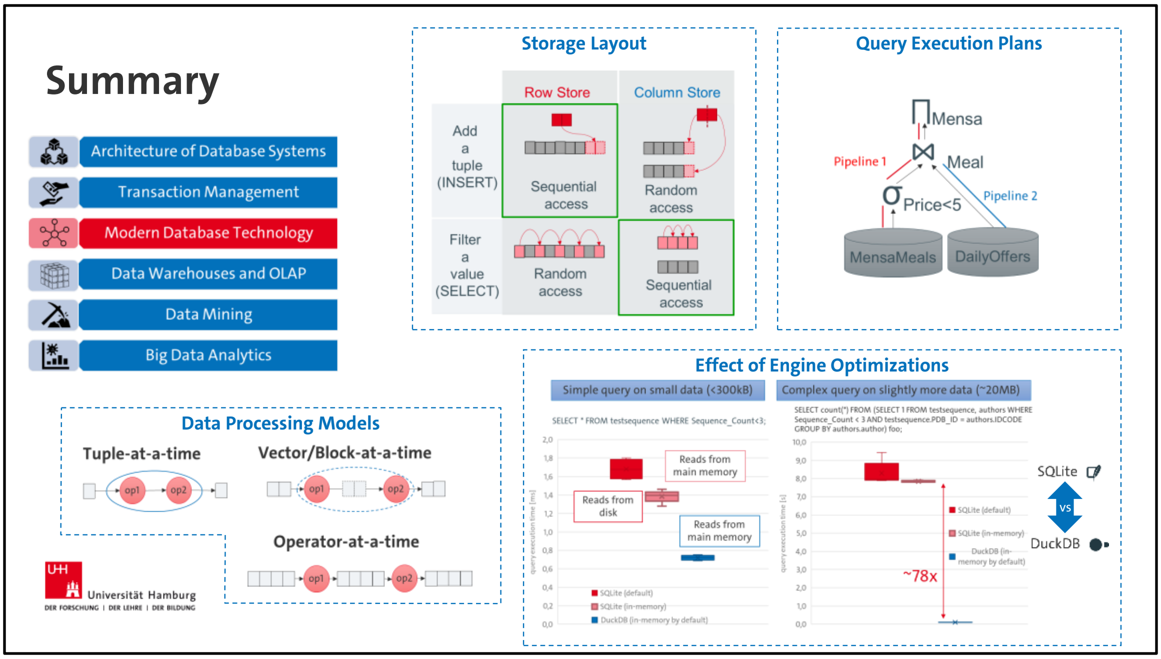
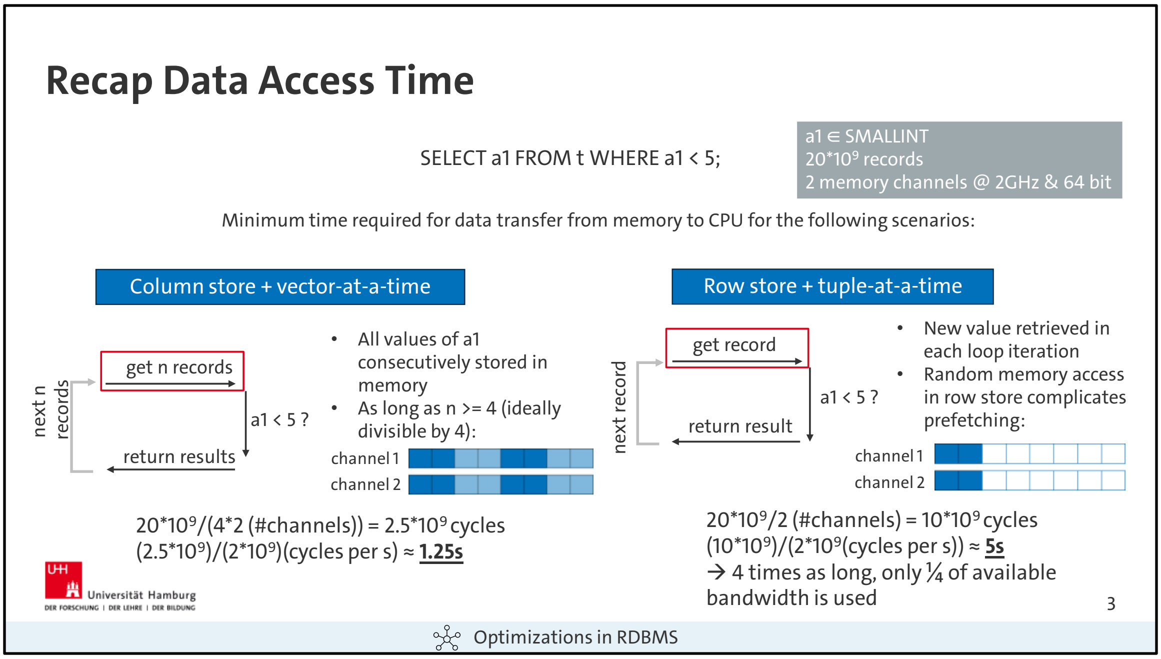

Summary
Architecture of Database Systems
Transaction Management
Modern Database Technology
Data Warehouses and OLAP
Data Mining
Big Data Analytics
Storage Layout
Row Store
Column Store
Add a tuple (INSERT)
Sequential access
Random access
Filter a value (SELECT)
Random access
Sequential access
Query Execution Plans
Mensa
Meal
Pipeline 1
Pipeline 2
Price<5
MensaMeals
DailyOffers
Data Processing Models
Tuple-at-a-time
Vector/Block-at-a-time
Operator-at-a-time
op1
op2
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG
Effect of Engine Optimizations
Simple query on small data (<300kB)
Complex query on slightly more data (~20MB)
SELECT * FROM testsequence WHERE Sequence_Count<3;
Reads from main memory
Reads from disk
Reads from main memory
SQLite (default)
SQLite (in-memory)
DuckDB (in-memory by default)
~78x
SQLite
vs
DuckDB



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

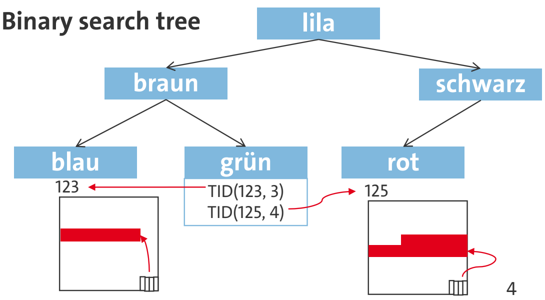
Optimizations by the User: Indexes

- 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

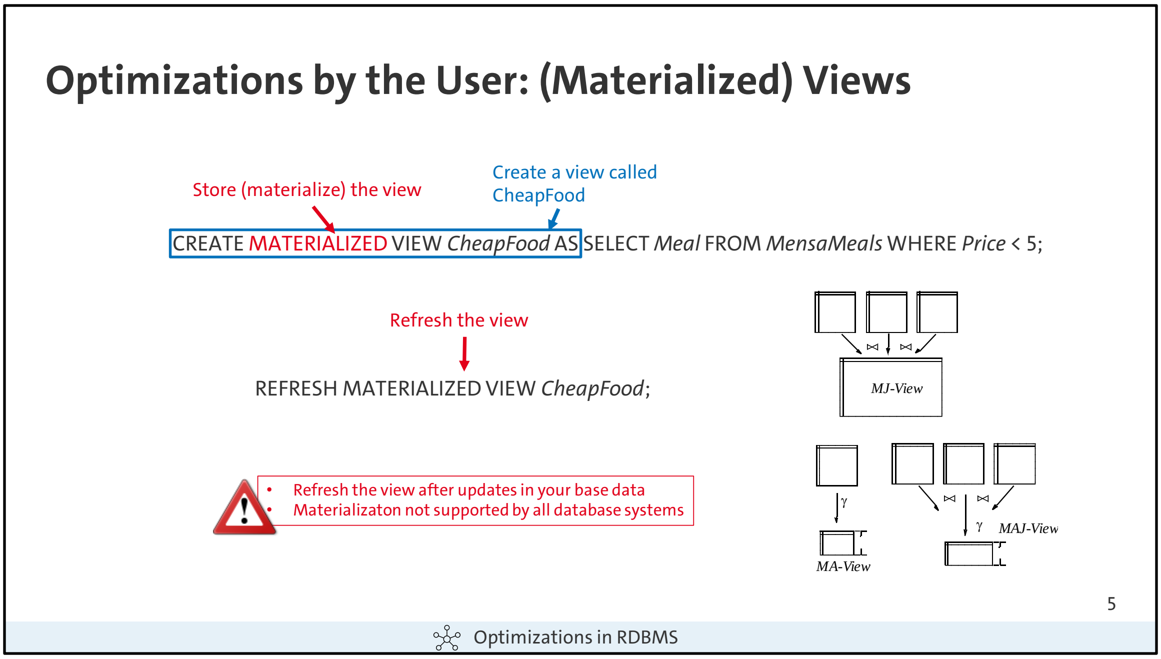
blau	
braun	
grün	TID(123, 3) TID(125, 4)
lila	
rot	
schwarz	

Binary search tree



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Recap of indexes in 1st DIS-lecture



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

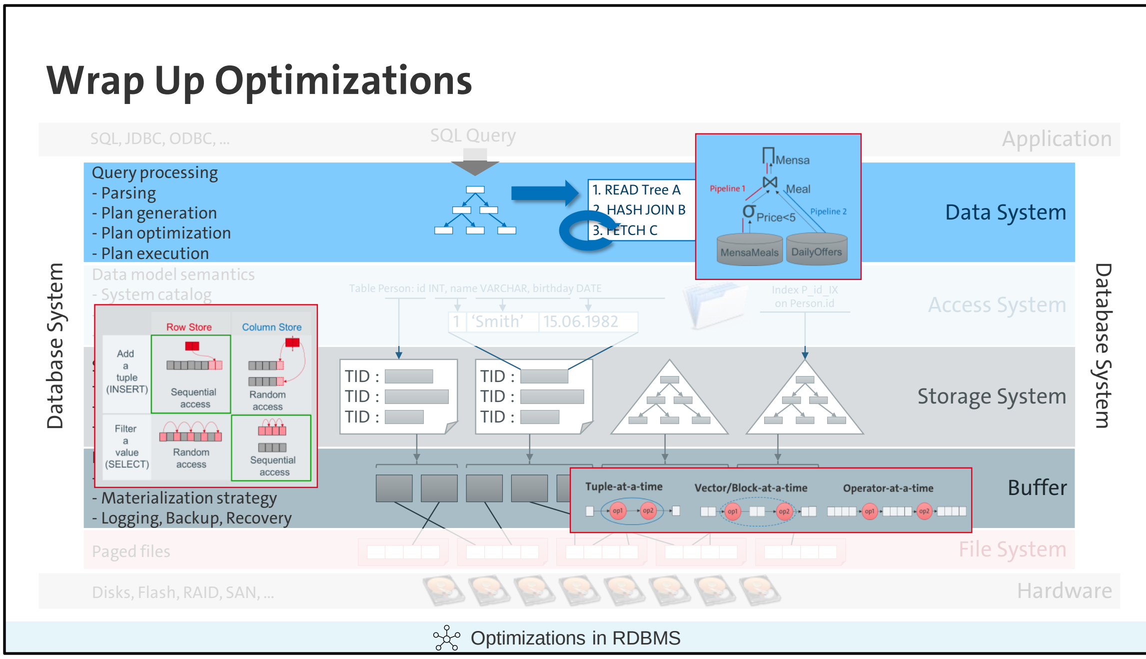
Query 1

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

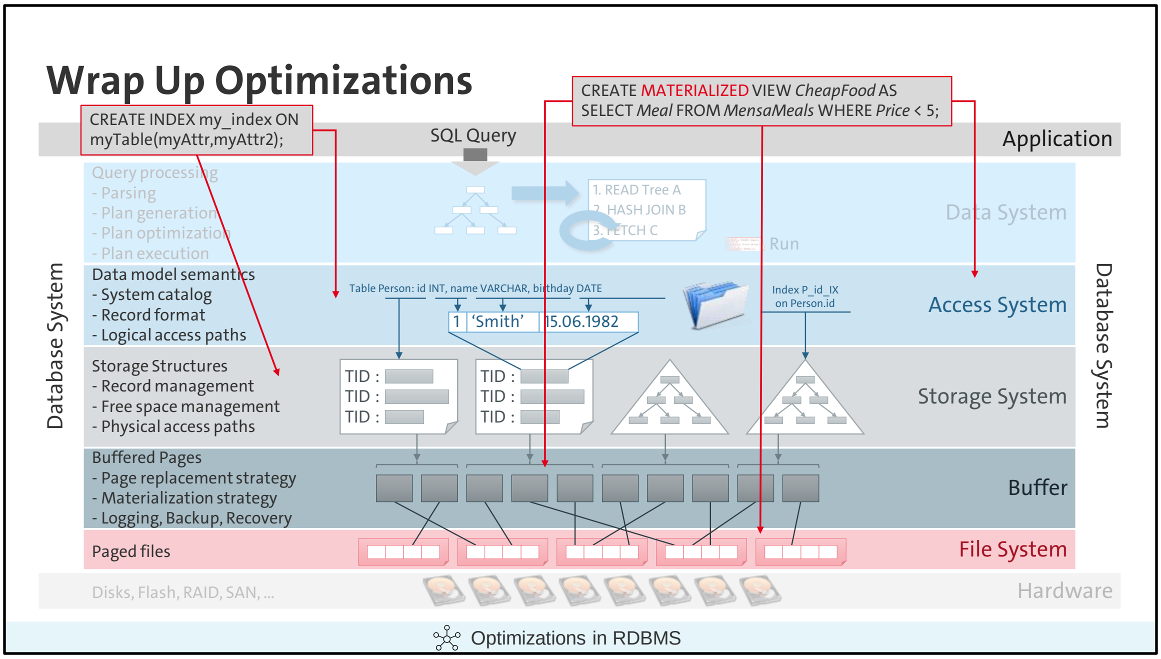
Query 2

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?



- 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



- 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

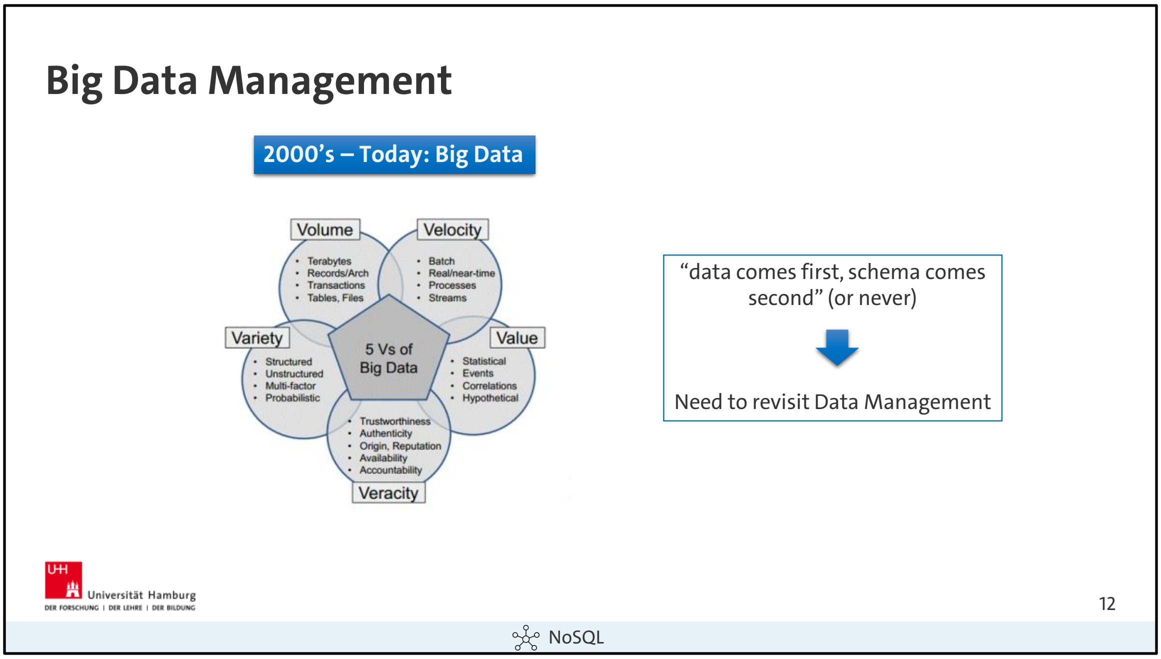
Survey

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

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Course Outline

- Architecture of Database Systems
- Transaction Management
- Modern Database Technology
    - Distributed Systems
    - Optimizations in RDBMS
    - **NoSQL**
- Data Warehouses and OLAP
- Data Mining
- Big Data Analytics



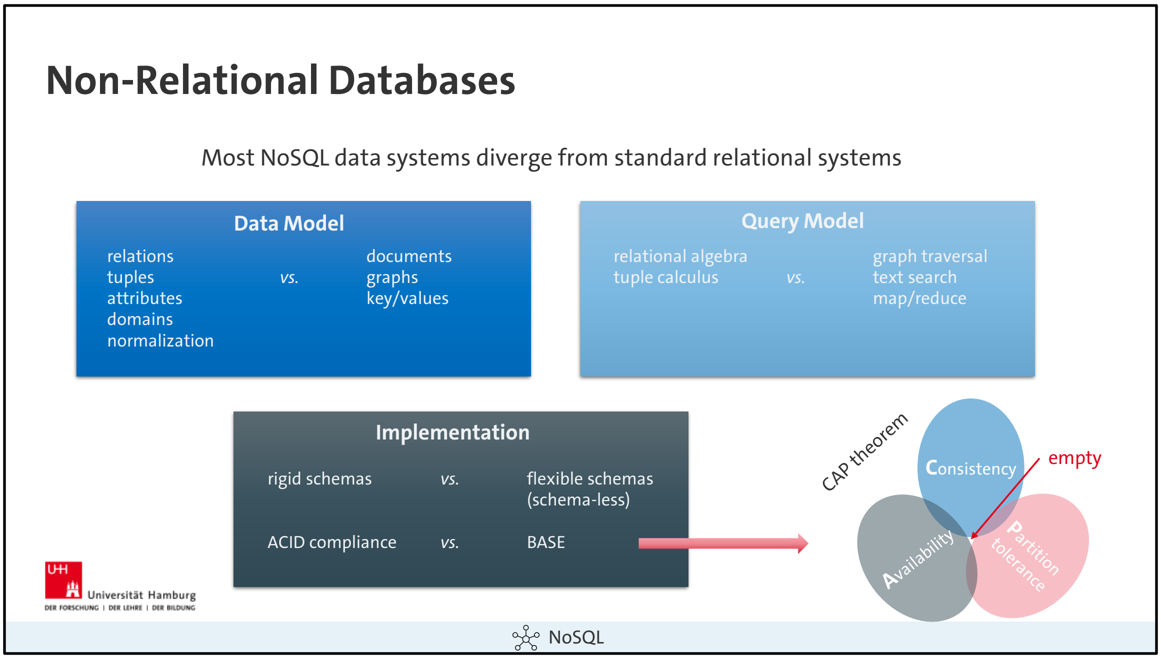
The number of Vs depends on who you ask, it started with just Variety, Volume, and Velocity

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



ACID → Atomicity, Consistency, Isolation, Durability

BASE: Availability over consistency

- **B**asically **a**vailable: 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
- **S**oft state: Data can have a temporary state, e.g. when multiple applications change it at the same time
- **E**ventually 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)

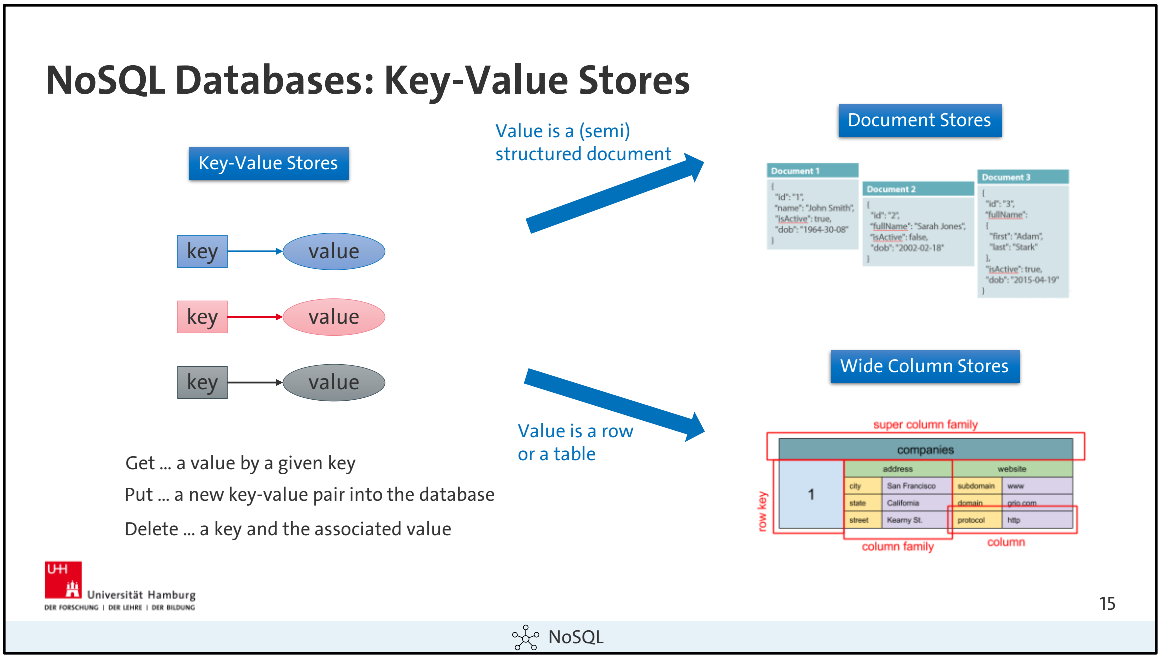
Types of Data Systems

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



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)

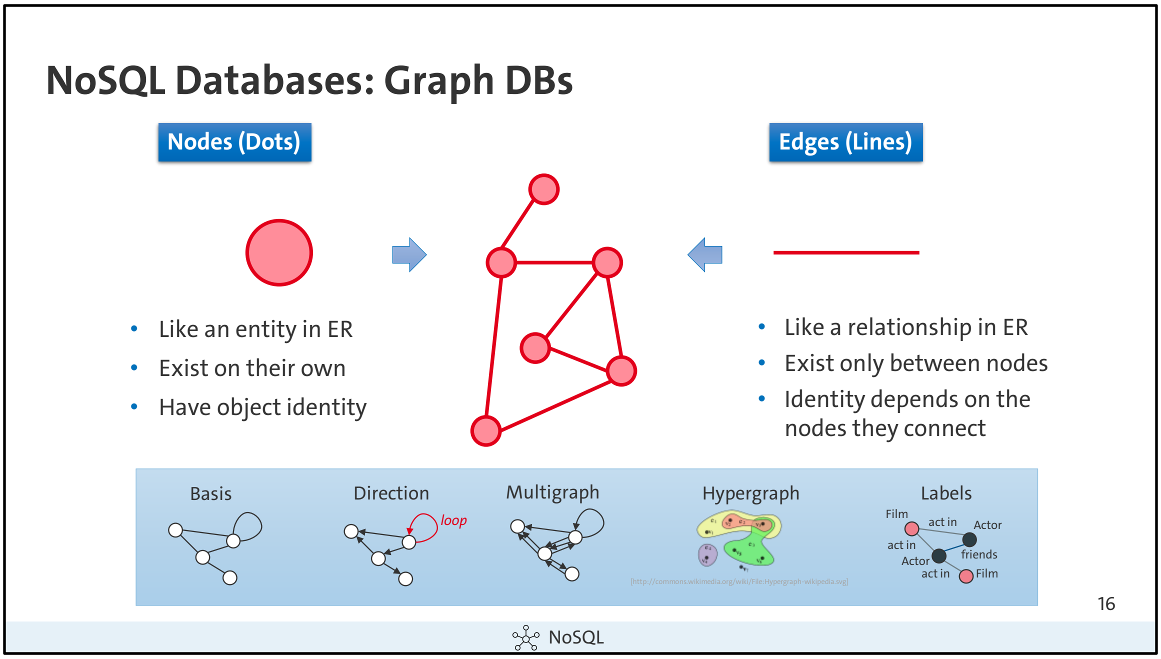
Wide Column Stores

- 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 distibuted 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)*



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: $E \subseteq \{\{x, y\} | x, y \in \mathrm{V}\}$

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: $E \subseteq \mathcal{O}$
- $\rho: E \rightarrow \{\{x,y\} \mid x,y \in \mathrm{V}\}$ (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⊆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)
- λ:V→$\mathcal{L}$ (vertex label)
  λ:E→$\mathcal{L}$ (edge label)
  λ:V∪E→$\mathcal{L}$ (vertex and edge label)
  λ:V→2^$\mathcal{L}$ (vertex labels)
  λ:E→2^$\mathcal{L}$ (edge labels)
  λ:V∪E→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