# NOSQL Databases

Dr. Lena Wiese

Institut für Informatik
Research Group Knowledge Engineering
Fakultät für Mathematik und Informatik
Georg-August Universität Göttingen

August/September 2016

# Short CV Dr. Lena Wiese

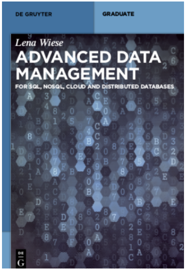
- University of Göttingen (Research Group Leader Knowledge Engineering)
- University of Hildesheim (Visiting Professor for Databases)
- National Institute of Informatics, Tokyo, Japan
- Robert Bosch India Ltd., Bangalore, India
- Master/PhD: TU Dortmund
- Teaching and Research
    - NoSQL databases (lecture, seminars, projects)
    - Database security (encryption for Cassandra and HBase)
- Web: `http://wiese.free.fr/`

# Conference Announcement BTW'17

- 17th Conference on Database Systems for **B**usiness, **T**echnology, and **W**eb
- Conference of German Database community (sponsored by the German Informatics Society GI)
- March 6th through March 10th 2017 at the University of Stuttgart in Germany
- `http://btw2017.informatik.uni-stuttgart.de/`
- Research and Industry Track, Demo Track, Workshops, Tutorials, Student Program, Dissertation Awards, Data Science Challenge
- Paper deadline: 23.9.2016
- Data Science Challenge deadline: 17.10.2016

# Copyright Notice

- Several pictures in this talk taken from my Master's level text book (in English):
  Lena Wiese: Advanced Data Management for SQL, NoSQL, Cloud and Distributed Databases
  © 2015 DeGruyter/Oldenbourg



# Overview

# Content

## SQL

- Tabular row-wise storage: Relational Databases (RDBs)
- Query Language: SQL

versus

## NOSQL (Not Only SQL)

- Graph Databases
- XML Databases
- Key-value Stores
- Column Stores
- Bigtable Databases
- Object Databases and Object-Relational Databases
- ...

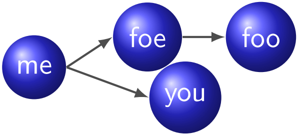
# What is a Database System?

A **database system** is required to

- manage
- huge amounts of data
- in an efficient,
- persistent,
- reliable,
- consistent,
- non-redundant way
- for multiple users

# New requirements

- Data are organized in complex structures (example: social networks)




- Data are constantly changing (frequent updates)
- Data are distributed on a huge number of interconnected servers (example: cloud storage)

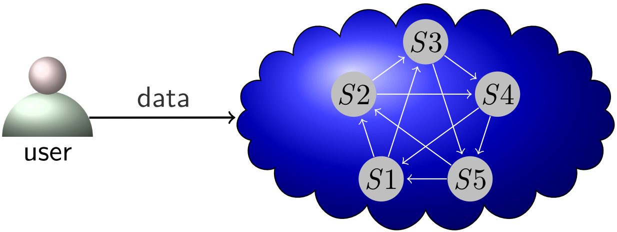
# New requirements

- Data are organized in complex structures (example: social networks)
- Data are constantly changing (frequent updates)


- Data are distributed on a huge number of interconnected servers (example: cloud storage)

# New requirements

- Data are organized in complex structures (example: social networks)
- Data are constantly changing (frequent updates)
- Data are distributed on a huge number of interconnected servers (example: cloud storage)



# New requirements

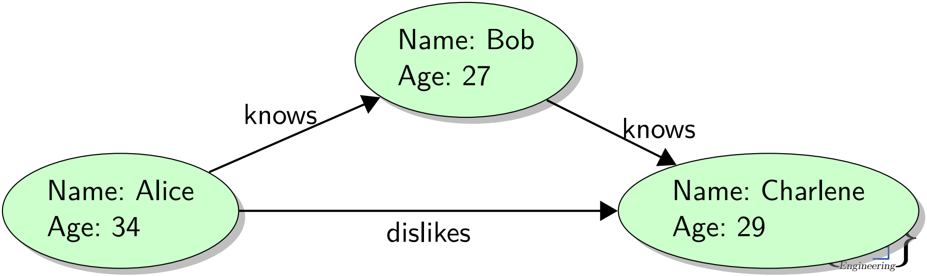
- Data are organized in complex structures (example: social networks)
- Data are constantly changing (frequent updates)
- Data are distributed on a huge number of interconnected servers (example: cloud storage)

Revival of non-relational data models for novel applications

# Overview

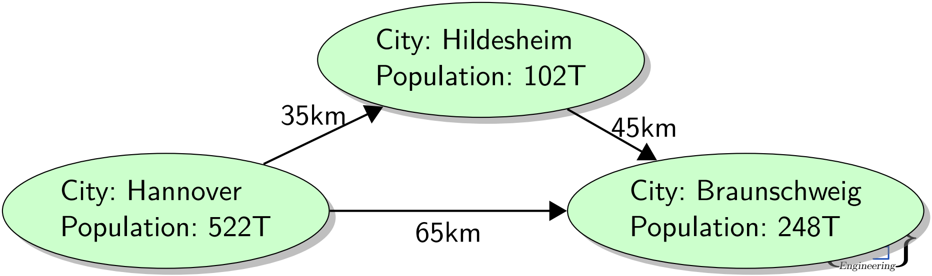
# Why Graph Databases?

- Links between data items are important
  - Example: Social Networks
  - Recommender Systems
  - Semantic Web
  - Geographic Information Systems
  - Bioinformatics
  - ...



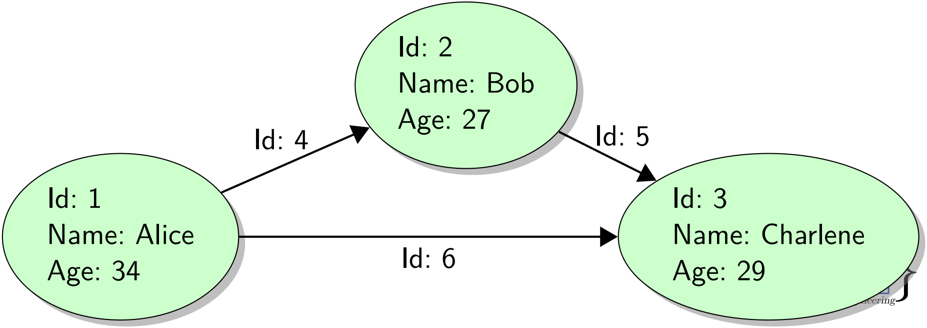
# Why Graph Databases?

- Links between data items are important
  - Social Networks
  - Recommender Systems
  - Semantic Web
  - Example: Geographic Information Systems
  - Bioinformatics
  - ...



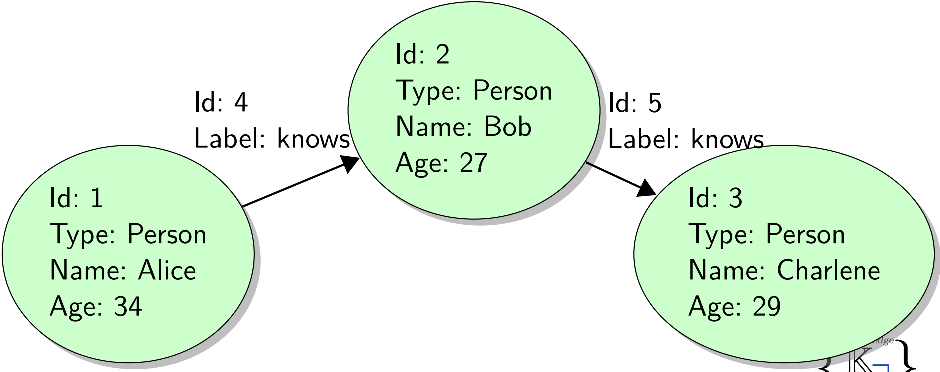
# Property Graph Model

- A Property Graph is a directed multigraph
- Stores information (*properties*) in vertices and on edges
- A Property is a key-value pair like "Name: Alice"
  - Sometimes *multi-value properties*: one key, list of values
- For vertices and edges: predefined property key called Id with unique identifier value



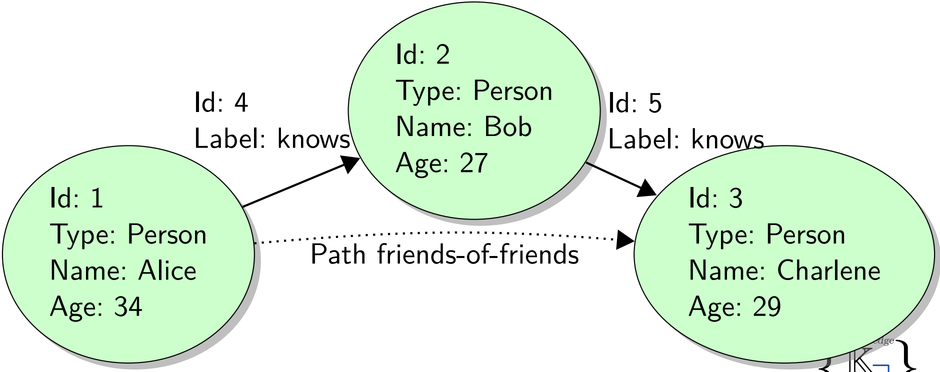
# Property Graph Model: Paths

- Paths are serial concatenations of edges
- End vertex of one edge is start vertex of next edge on the path



# Property Graph Model: Paths

- Path “friends-of-friends” concatenates two edges with “Label: knows”
- Paths can be used as normal edges



# Open Source Systems

- The TinkerPop `http://tinkerpop.apache.org/`
  - graph processing stack: a set of open source graph management modules
- Neo4J graph database `http://neo4j.com/`
  - Cypher query language

```
START alice = (people_idx, name, "Alice")
MATCH (alice)-[:knows]->(aperson)
RETURN (aperson)
```

- HyperGraphDB: `http://www.hypergraphdb.org/`
  - Graph may contain hyperedges that combine more than two nodes

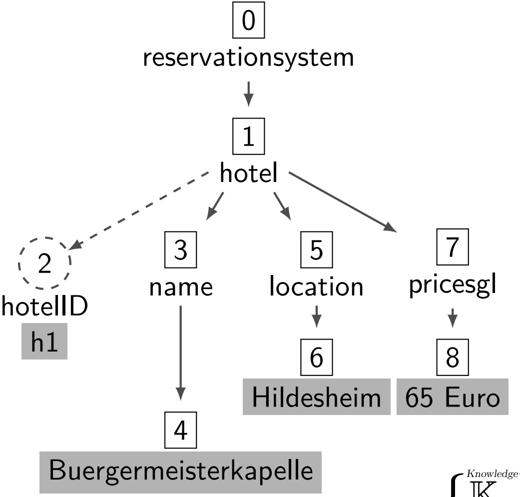
# Overview

# XML

- XML: Extensible Markup Language
- Defined by the WWW Consortium (W3C)
- Intended as a document markup language (not a database language)
- *Tags* divide documents into sections
- Tag: label for a section of data
- Element: section of data beginning with <tagname> and ending with matching </tagname>
- Inside an element:
  - arbitrary text
  - other elements ("nesting")
  - Nothing ("empty element"): abbreviate to <tagname />
- Standardized query languages: XPath and XQuery

# Tree Model of XML Data

```
<reservationsystem>
 <hotel hotelID="h1">
  <name>
   Buergermeisterkapelle
  </name>
  <location>
   Hildesheim
  </location>
  <pricesgl>
   65 Euro
  </pricesgl>
 </hotel>
</reservationsystem>
```



# Numbering Scheme

- assigns each node of an XML tree a unique identifier (a label or node ID which is usually a number)
- Important for database application with frequent updates:
  - How many nodes have to be renumbered in an update?
- simplest scheme: preorder traversal of tree
- increasing a counter for each node:
  - root node is numbered as the first node before numbering any other node
  - this is done recursively for all child nodes
- Renumbering: all nodes in the worst case

# Open Source Systems

- eXistDB: `http://exist-db.org/`
  - numbering scheme that virtually expands the tree into a complete tree such that not all node IDs correspond to existing nodes
  - eXistDB offers several user APIs: RESTful API, XML:DB API, XML-RPC API, SOAP AP
- BaseX: `http://basex.org/`
  - Numbering scheme: Pre/Dist/Size
  - Several language bindings as well as a REST API, an XQJ API and a XML:DB API

# Overview

# Key-Value Stores

- A key value pair is a tuple of two strings $\langle key, value \rangle$
  - You can get (or delete) a value from the store by key
  - Schema-less: you can put arbitrary key-value pairs into the store

```
value = store.get(key)
store.put(key, value)
store.delete(key)
```

- Values can have other data types than just strings
- Values can even be a list or array of atomic values
- Simple but quick
  - Simple data structure
  - No advanced query language
  - Good for "data-intensive" applications
  - Application is responsible or combining key-value pairs into more complex objects

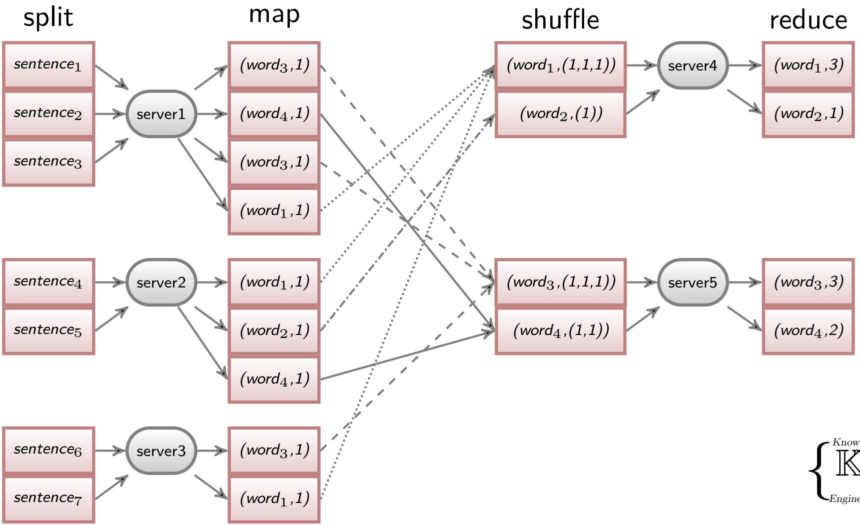
# Open Source Systems

- Redis: `http://redis.io/`
  - in-memory key-value store
  - data types: string, linked lists, unsorted set, sorted set, hash, bit array, hyperloglog
- Riak-KV: `http://basho.com/products/riak-kv/`
  - key-value pairs called Riak objects grouped into buckets
  - convergent replicated data types (CRDTs)
  - Riak's search functionality based on Apache Solr (Yokozuna)

# MapReduce

- Applied at Google
  - Jeffrey Dean / Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", OSDI'04: Sixth Symposium on Operating System Design and Implementation, 2004.
- "The computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce library expresses the computation as two functions: Map and Reduce."
- Four basic steps
  1. **split** input key-value pairs into disjunct subsets
  2. compute **map** function on each input subset
  3. group all intermediate values by key (**shuffle**)
  4. **reduce** values of each group

# MapReduce: Example



# Open Source Systems

- Apache Hadoop: `http://hadoop.apache.org/`
  - Hadoop Distributed File System (HDFS)
- Apache Spark: `http://spark.apache.org/`
  - data flow programming model on top of Hadoop
- Apache Pig: `http://pig.apache.org/`
  - express parallel execution of data analytics tasks

```
input={('alice',{'charlene','emily'}),
       ('bob',{'david','emily'})};
output = FOREACH input GENERATE $0, FLATTEN($1);
```

- Apache Hive: `http://hive.apache.org/`
  - querying and data management layer
  - can serialize tables as files in HDFS
  - HiveQL queries are compiled into Hadoop MapReduce tasks

# Overview

# JSON: JavaScript Object Notation

- human-readable text format
- more compact than XML
- nesting of key-value pairs


```
{
 "firstName":"Alice",
 "lastName" :"Smith",
 "age":31,
 "address" :{
      "street":"Main Street",
      "number":12,
      "city":"Newtown",
      "zip":31141
 } ,
 "telephone":[935279,908077,278784]
}
```

# Open Source Systems

- MongoDB: https://www.mongodb.org/
  - BSON storage format (binary JSON representation)
  - `db.persons.find(age$lt: 34)`
- CouchDB: http://couchdb.apache.org/
  - retrieval process with views defined as map function

```
function(doc) {
   if(doc.lastname && doc.age) {
     emit(doc.lastname, doc.age);
   }
}
```

- Couchbase: http://www.couchbase.com
  - SQL-like query language

# Overview

# Why Column Stores?

- A *row store* is a row-oriented relational database
    - Data are stored in tables
    - On disk, data in a row are stored consecutively
    - Currently used in most commercially successful RDBMSs
- A *column store* is a column-oriented relational database
    - Data are stored in tables
    - On disk, data in a column are stored consecutively
    - In use since the 1970s but less successful than row stores
- Example

| BookLending | BookID | ReaderID | ReturnDate |
|---|---|---|---|
| | 123 | 225 | 25-10-2011 |
| | 234 | 347 | 31-10-2011 |

Storage order in row store:

123,225,25-10-2011,234,347,31-10-2011

Storage order in column store:

123,234,225,347,25-10-2011,31-10-2011

# Advantages of Column Stores

- Only columns (attributes) that are needed are read from disk into main memory, because a memory page contains only values of a column
- Values in a column (that is, values of the same attribute domain) can be compressed better when stored consecutively ("locality")
- Iterating or aggregating over values in a column can be done quickly, because they are stored consecutively
  - For example, summing up all values in a column, finding the average, maximum...
- Adding new columns to a table is easy

# Column Compression

- Columns may contain lots of repetitions of values
- Compression can be more effective on columns
- Option 1: run-length encoding
  - run-length: how many repetitions of a value are stored consecutively?
- Option 2: bit-vector encoding
  - create a bit vector for each value in the column
- Option 3: dictionary encoding
  - create a dictionary for single values or sequences of values
- Option 4: frame of reference encoding
  - store off-set from a reference point
- Option 5: differential encoding
  - store off-set from previous value
- Stavros Harizopoulos / Daniel Abadi / Peter Boncz, "Column-Oriented Database Systems", VLDB Tutorial, 2009

# Example: Run-Length Encoding

| BookLending | BID | RID | RD |
|---|---|---|---|
| | 123 | 225 | 25-10-2012 |
| | 386 | 225 | 20-10-2012 |
| | 938 | 225 | 27-10-2012 |
| | 123 | 347 | 25-11-2012 |
| | 234 | 347 | 31-10-2012 |

- Store ReaderID (RID) in run-length encoding
  - count number of consecutive repetitions
  - format: (value, start row, run-length)
  - RID: ( (225, 1, 3), (347, 4, 2) )
- Answer queries on compressed format
  - How many books does each reader have?
  - `SELECT RID, COUNT(*) FROM BookLending GROUP BY RID`
  - Just return (the sum of) the run-lengths for each ReaderID value
  - Result: (225, 3), (347, 2)

# Systems

- MonetDB: https://www.monetdb.org/
  - open source "column store pioneers"
- Apache Parquet: http://parquet.apache.org/
  - implements column striping: transform nested data to columns
- Commercial systems
  - SAP HANA
  - HP Vertica
  - IBM DashDB

# Overview

# Google BigTable

- Fay Chang / Jeffrey Dean / Sanjay Ghemawat / Wilson C. Hsieh / Deborah A. Wallach / Mike Burrows / Tushar Chandra / Andrew Fikes / Robert E. Gruber, "Bigtable: A Distributed Storage System for Structured Data", OSDI, 2006
- "A Bigtable is a sparse, distributed, persistent, multi-dimensional sorted map"
- Google BigTable is indexed by a row key, column key, and a timestamp
- Map: ( row:string, column:string, time:int64) → string
- A Big Table may have an unbounded number of columns.
- Columns are grouped into sets called **column families**.

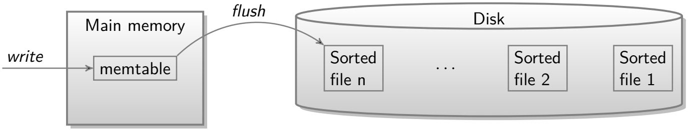
# BigTable & HBase Data Structure

- Store data that is accessed together in a column family
  - Columns in a single column family can vary arbitrarily for each row.
  - Only fetch column families of columns that are required by query
  - Data locality: Store data in a column family together on disk

| table<br>Library | row key<br>**BID** | column family<br>**BookInfo** | column family<br>**LendingInfo** |
|---|---|---|---|
| | 123 | **Title**: Databases; **Author**: Miller | **25-10-2012**: Mayer; **25-11-2012**: Green |
| | 386 | **Title**: Algorithms; **Author**: Jacobs | **20-10-2012**: Mayer |
| | 938 | **Title**: Programming; **Author**: Brown | **27-10-2012**: Mayer |
| | 234 | **Title**: SQL; **Author**: Smith | **31-10-2012**: Green |

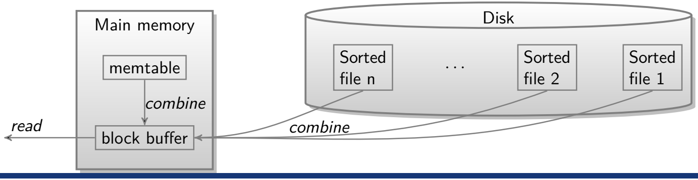
# Writing to memory tables and data files

- The most recent writes are collected in a main memory table (*memtable*) of fixed size.
- All data records written to the on-disk store will only be appended to the existing records.
- Once written, these records are read-only and cannot be modified: they are **immutable** data files.
- Any modification of a record must hence also be simulated by appending a new record in the store.
- Deletions are treated by writing a new record (tombstone) for a key.



# Reading from memory tables and data files

- The downside of immutable data files is that they complicate the read process:
  - retrieving all the relevant data that match a user query requires combining records from several on-disk data files and the memtable.
- This combination may affect records for different search keys that are spread out across several data files; but it may also apply to records for the same key of which different versions exist in different data files.
- In other words, all sorted data files have to be searched for records matching the read request.



# Open Source Systems

- Apache Cassandra: `http://cassandra.apache.org/`
  - column families in a keyspace
  - CQL: SQL-like query language

```
INSERT INTO bookinfo (bookid, title, author)
                VALUES (1002,'Databases','Miller');
```

- Apache HBase: `http://hbase.apache.org/`
  - stores tables in namespaces
  - tables contain column families

# Overview

# Polyglot Data Management

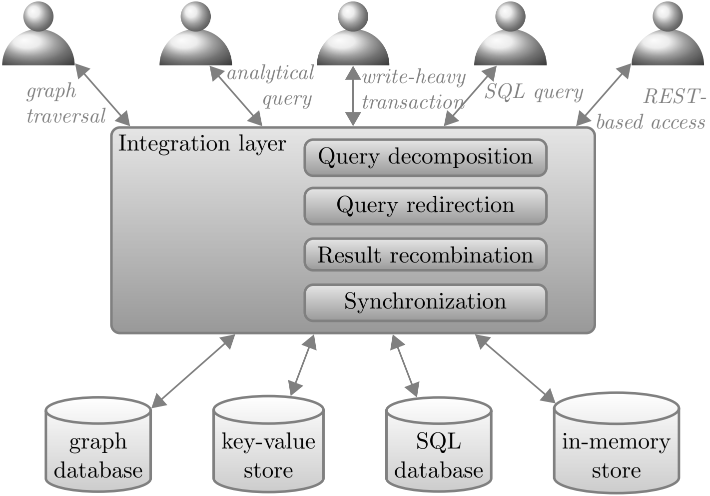
- Data management layer has to handle contradictory requirements
  - access patterns: write-heavy workloads vs read-heavy workloads
  - data model: data of different structures
  - access method: web application access via REST vs programmatic access vs query language
- Consider a database and storage architecture that includes all these requirements (well, at least some...)
  - Polyglot Persistence
  - Lambda Architecture
  - Multi-Model Databases

# Polyglot Persistence

- Choose as many databases as needed
  Fowler, M.J., Sadalage, P.J.: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Prentice Hall (2012)
- Example: Apache Drill `http://drill.apache.org/`
  - Apache Drill is inspired by the ideas developed in Google's Dremel system

  Melnik, S., Gubarev, A., Long, J.J., Romer, G., Shivakumar, S., Tolton, M., Vassilakis, T.: Dremel: interactive analysis of web-scale datasets. Proceedings of the VLDB Endowment 3(1-2), 330–339 (2010)
- Introduces an integration layer
  - decomposing queries in to several subqueries
  - redirecting queries to the appropriate databases
  - recombining the results obtained from the accessed databases

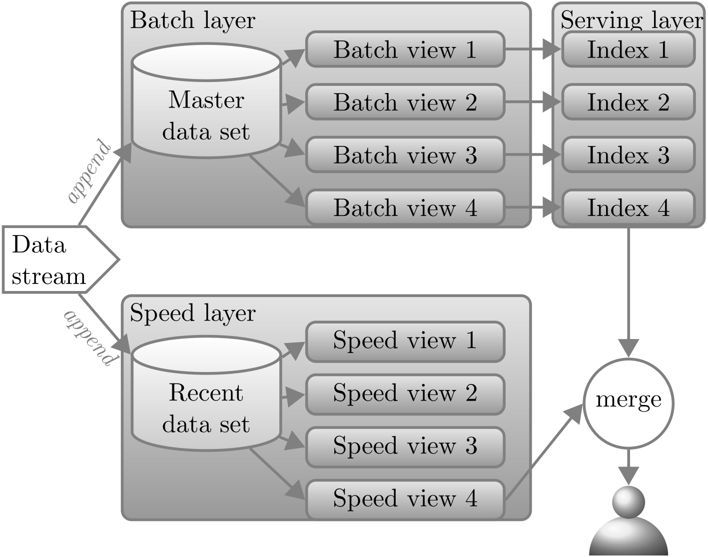
# Polyglot Persistence



# Lambda Architecture

- For real-time / streaming data
- Combination of a slower batch processing layer and a speedier stream processing layer
  - Speed layer: only the most recent data delivered in several real-time views
  - Batch layer: data stored in an append-only and immutable fashion in a "master dataset" delivered in so-called batch views
  - Serving layer: makes batch views accessible to user queries by maintaining indexes
- User queries answered by merging data from batch views and real-time views
- Open source implementation following the ideas of a lambda architecture is Apache Druid `http://druid.io/` (streaming data in real-time nodes and batch data in historical nodes)

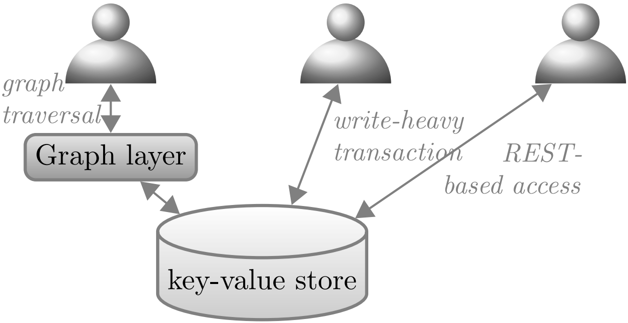
# Lambda Architecture



# Multi-Model Databases

- Data in a single store but providing access to the data with different APIs (according to different data models)
- Either support different data models directly inside the database engine or offer layers for additional data models on top of a single-model engine
- OrientDB `http://orientdb.com/`
    - a document API, an object API, and a graph API (Java Graph API is compliant with Tinkerpop)
    - extensions of the SQL standard to interact will all three APIs
- ArangoDB `https://www.arangodb.com/`
    - a graph API, a key-value API and a document API
    - Query language AQL (ArangoDB query language) resembles SQL but adds several database-specific extensions to it

# Multi-Model Databases



# Overview

# Conclusion

- Many, many other data models than just relational tables
- Lots of different query languages (no standards)
- Problems with reliability (no long-term experience, open source development teams)
- Which database you choose depends on your needs