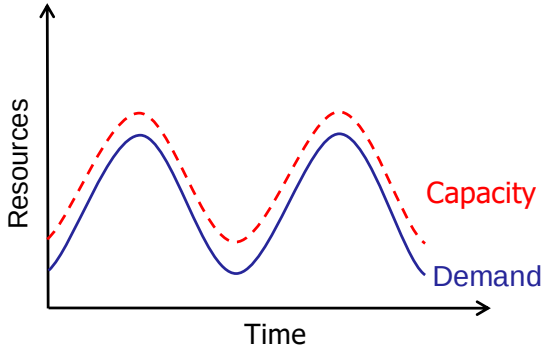
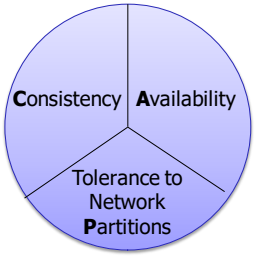
Chapter 4
Big Data & Internet Information Systems





Overview

4.1 Data Management in the Cloud

4.2 Big Data Architectures & Systems

- 4.2.1 Hadoop
- 4.2.2 Spark
- 4.2.3 Kafka

4.3 SQL in Big Data Applications

4.4 Big Data and NoSQL support in Classical DBMS

Literature

P.J. Sadalage, M. Fowler: *NoSQL Distilled*. Addison-Wesley, 2012.
Good overview of the key concepts of NoSQL systems; has a system independent and a system specific part.

E. Redmond, J.R. Wilson: *Seven Databases in Seven Weeks*. Pragmatic Programmers, 2012.
Hands-on book for seven important DBMS, very many technical details, less suited for an overview.

K. Banker: *MongoDB in Action*. Manning, 2012.
Detailed book for MongoDB, addresses also recent features.

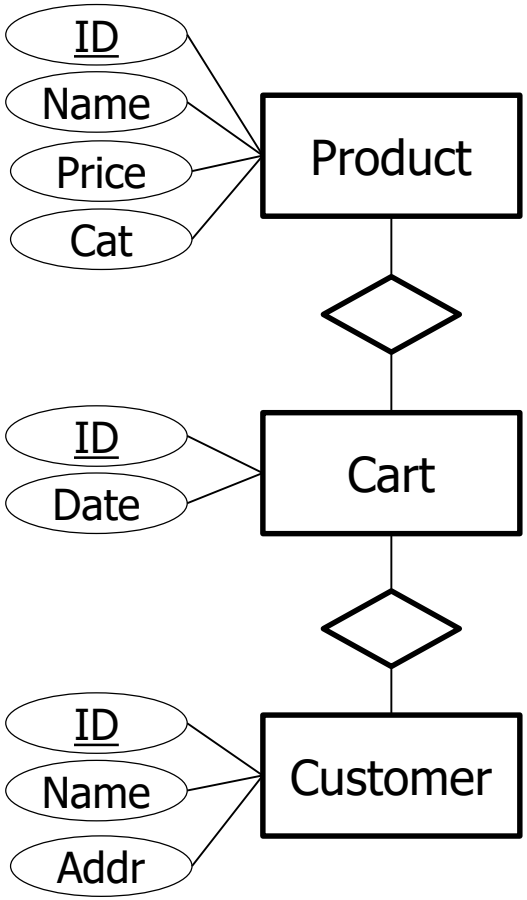
D. McMurtry et al.: *Data Access for Highly-Scalable Solutions: Using SQL, NoSQL, and Polyglot Persistence.* Microsoft, 2013. Good overview of different NoSQL data models, available online: https://www.microsoft.com/en-us/download/details.aspx?id=40327

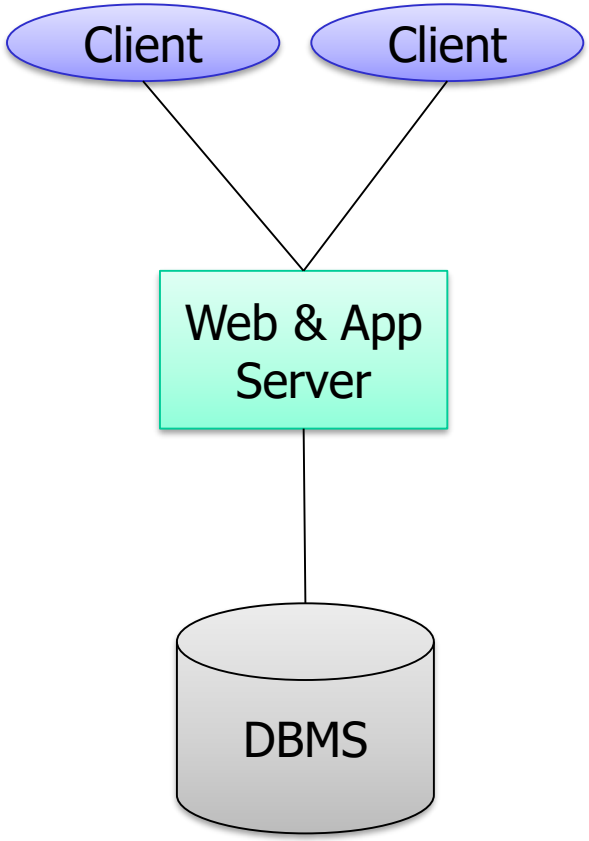
Review of RDBMS Technology

- Simple & efficient data model
  - Relations, tuples, keys, foreign keys, …
  - Standardized for more than 20 years
- Expressive query & update language
  - Enables complex queries with aggregates, functions, …
  - Updates can be based on query results
- Powerful transaction management
  - Enables concurrent access by multiple users & applications
  - ACID principle guarantees
    Atomicity,
    Consistency,
    Isolation, and
    Durability of transactions
- Basis for application integration
  - Applications use the same shared database instance

A Typical Internet Application (1)

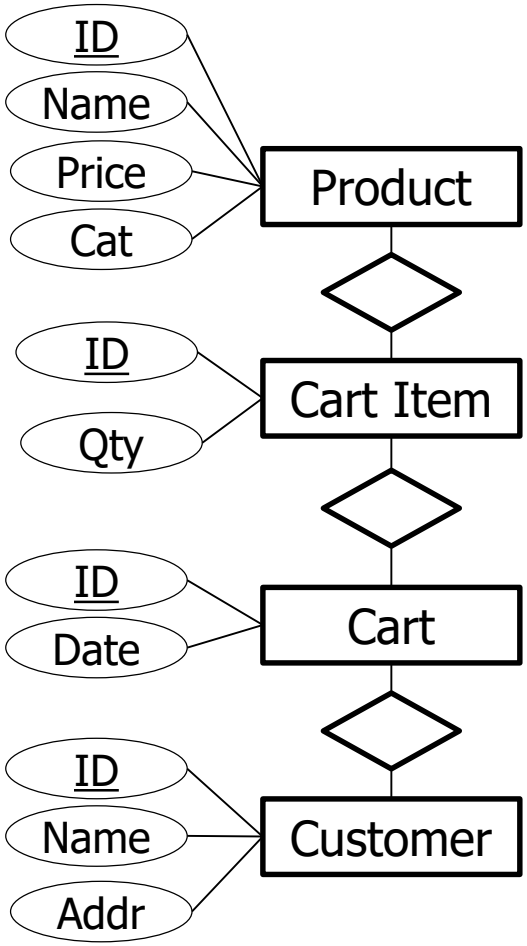
Web Shop with some products, costumers, carts, etc.

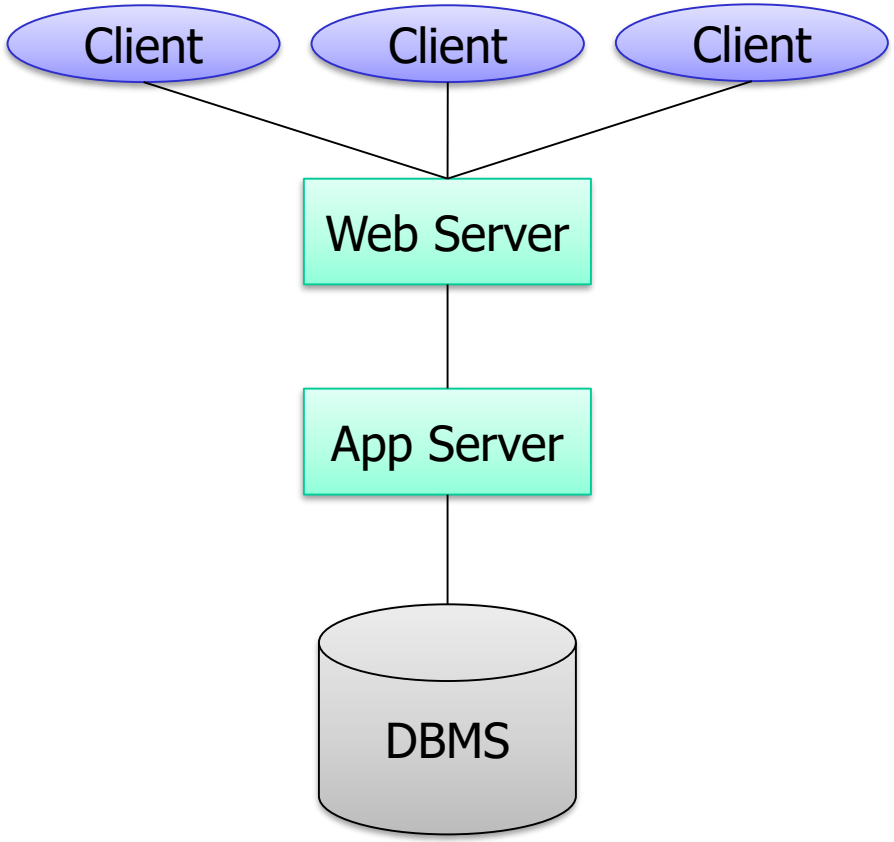




A Typical Internet Application (2)

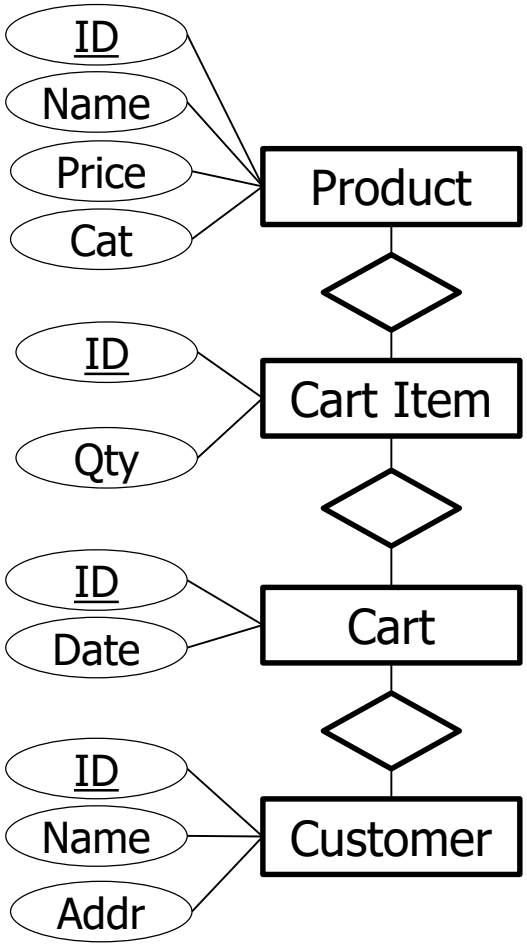
Business is going well, more customers visit your site …

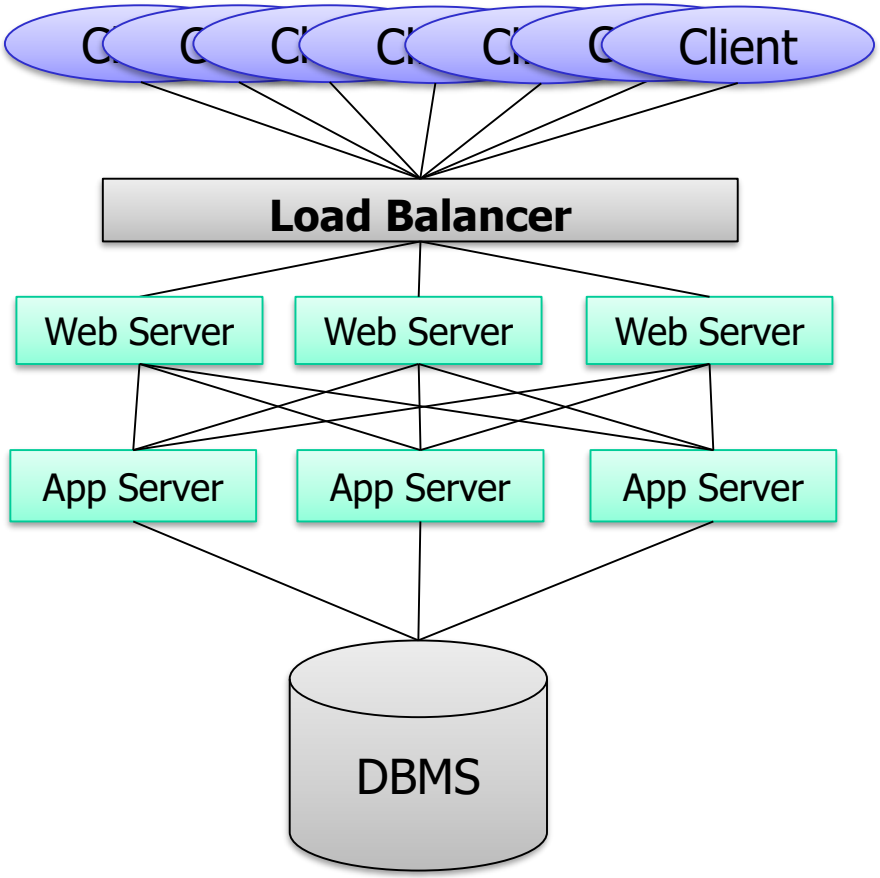




A Typical Internet Application (3)

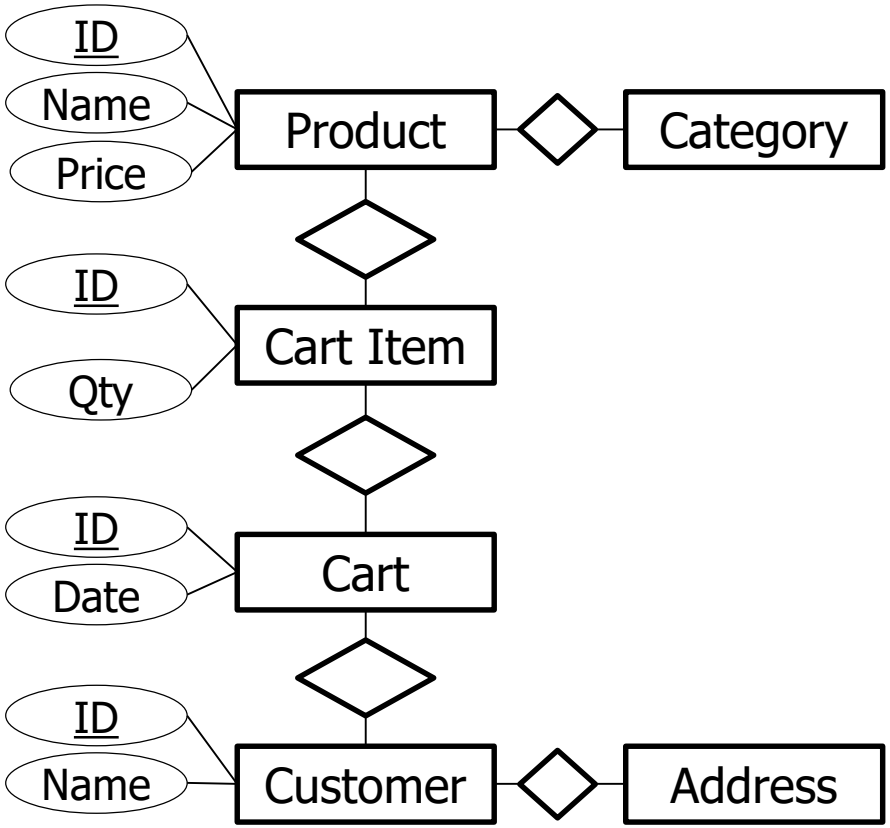
Business is going *very* well, many customers visit your site …

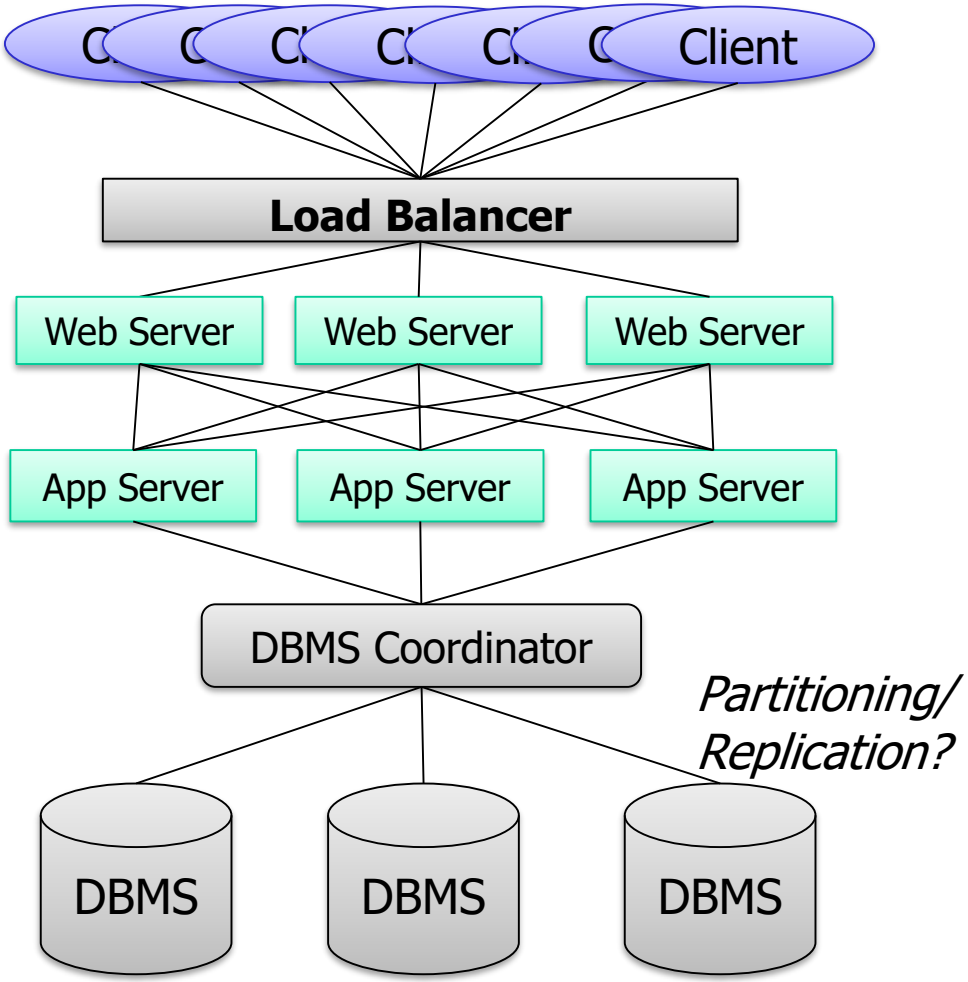




A Typical Internet Application

DBMS is the bottleneck, scaling a relational DBMS is difficult





Example: SQL Schema & Queries

CartItem

ID	QTY	PID

Product

ID	Name	Price

Sum of all items in shopping carts, grouped by product ID

```
SELECT PID, SUM(QTY)
FROM CartItem
GROUP BY PID
```

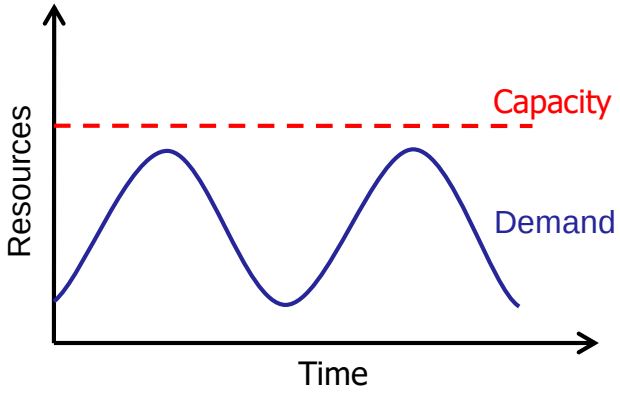
Sum of the value of all items in shopping carts, grouped by product ID

```
SELECT PID, SUM(QTY)*Price
FROM CartItem, Product P
WHERE P.ID=PID
GROUP BY PID
```

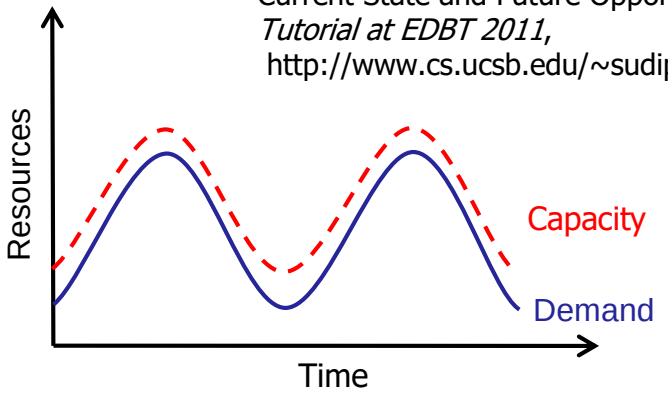
Data Management in the Cloud

- Distribution
  - Data needs to be distributed to multiple nodes to handle workload
  - Queries can be processed in parallel to increase query performance
  - Handling of updates requires additional communication between nodes if consistency is required

- Scalability
  - New systems need to be added/removed to deal with changing demand
  - RDBMS are not well designed for distribution
  - Parallel RDBMS are expensive



D. Agrawal et al.: Big Data and Cloud Computing: Current State and Future Opportunities. *Tutorial at EDBT 2011*, http://www.cs.ucsb.edu/~sudipto/edbt2011/



Problems with RDBMS: Data Model and Query Language

- Effort for database design & schema definition is too high
  - A-priori definition of schema is not possible with frequently changing requirements
  - Relational model does not fit application model
- Many applications require only simple retrieve & update queries for single objects
  - Joins are required due to normalization in RDBMS
  - Result tuples cannot be updated directly
- Application integration done by service-oriented architecture
  - Application database vs. shared database
  - Services are used for application integration rather than direct access to underlying databases

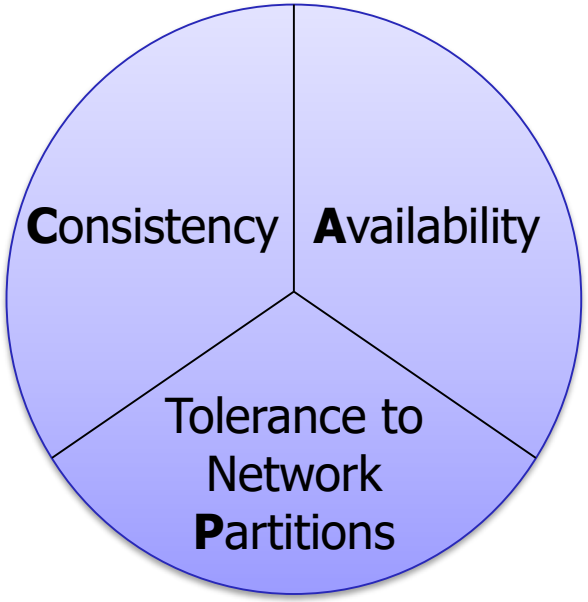
Problems with RDBMS: Transaction Management and Scalability

- Complex transaction management with ACID properties is not necessary for many applications
  - Transactions are managed at application level
  - Optimistic concurrency control often more efficient

- Distributing a RDBMS is a hard job
  - Most RDBMS do not provide solutions for distributed/partitioned databases out-of-the-box
  - Setup and maintenance of distributed RDBMS is complex
  - Changes in configuration often requires downtime, which is not acceptable in online business

The CAP Theorem

- **C**onsistency, **A**vailability, and Tolerance to Network **P**artitions
  - One can only have two of the three together [CAP Theorem, Brewer, 2000]



- Large scale operations – be prepared for network partitions

- Role of CAP – During a network partition, choose between consistency and availability
  - RDBMS choose *consistency*
  - NoSQL databases choose *availability* (low replica consistency)

NoSQL Databases

- Proposed as a simple, scalable data management systems
- Meaning of NoSQL
  - The system does **NO**t provide a **SQL** interface
  - **N**ot **O**nly **SQL**

- Common characteristics of NoSQL Database Systems
  - Not using the relational data model
  - Running well on clusters
  - Open-Source
  - Schemaless
  - Built for web applications
  - Eventual consistency

Data Models of NoSQL Systems

- Key-Value Data Model
  - Keys are used to retrieve a value or an object
  - Objects can be arbitrary complex
- Document-oriented Data Model
  - Data is stored as collections of documents
  - Each document can have an arbitrary structure
- Graph Data Model
- Column-oriented Data Model

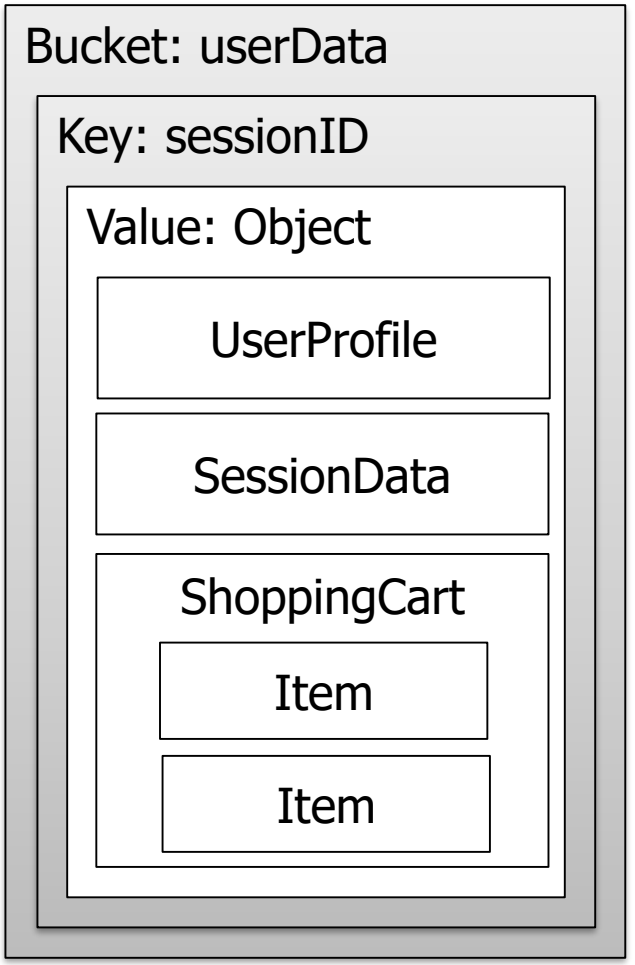
Advantages of NoSQL data models:

Complex objects are not scattered across several relations

➔ Expensive join queries are avoided
➔ Easy mapping to application data model
➔ Increases programmers' productivity

Key-Value Data Model

- Keys identify objects
- Objects can be linked to other objects
- Access is mainly by keys;
  kind of full-text search for values
- Systems:
  - Riak
  - Redis
- Good for:
  - Session information
  - User profiles
  - Shopping Cart
- Not well suited for:
  - Query by data



Document-oriented Data Model

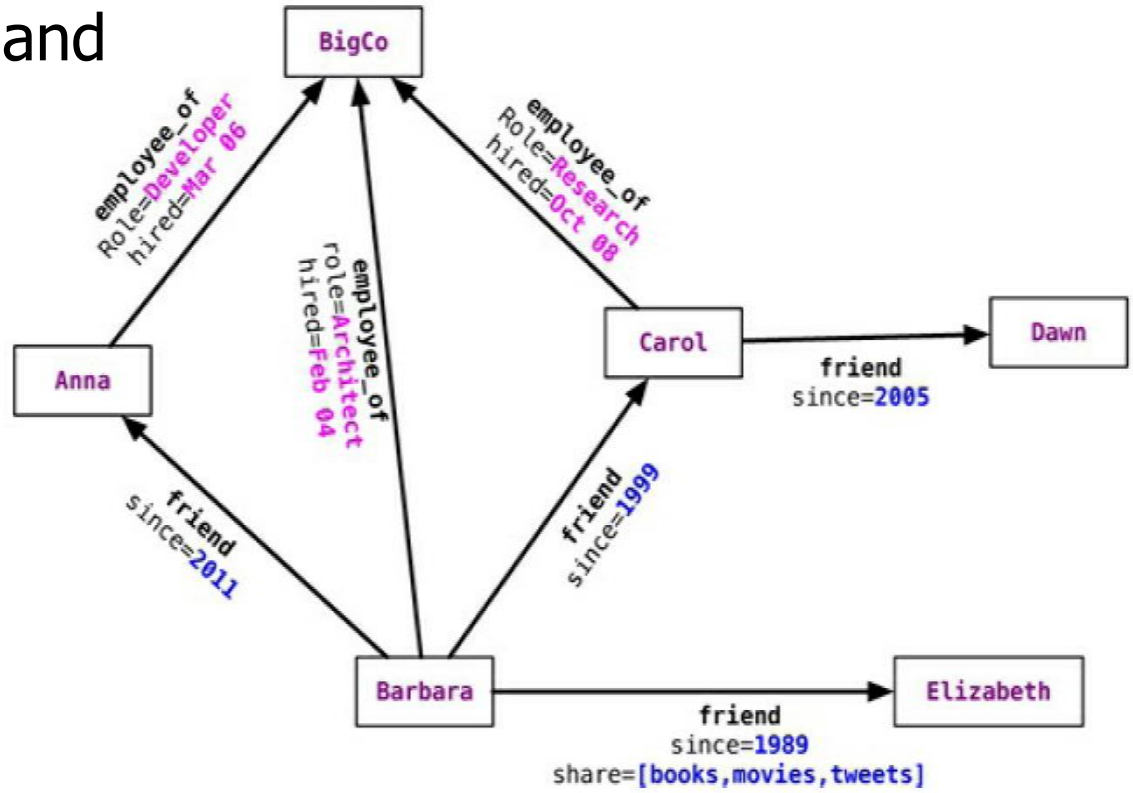
- Document is a list of attribute-value pairs, with possible nesting
- Semi-structured data model, often represented in JSON
- Indexes for efficient querying
- Systems:
  - MongoDB
  - CouchDB
- Good for:
  - Event logging
  - Content mgmt & blogging systems
  - E-Commerce applications
- Not well suited for:
  - Queries with varying result structure

Customers collection

```
{
  firstname: "Harri",
  lastname: "Hirsch",
  address: {
    street: "Ahornstr.",
    city: "Aachen",
    zip: "52074"
  },
  lastsearch: "tablet"
}
{
  firstname: "John",
  lastname: "Doe",
  address: { … },
  cart: [{ prodid: 3456,
           qty: 2},
         { prodid: 6789,
           qty: 1}]
}
```

Graph Data Model

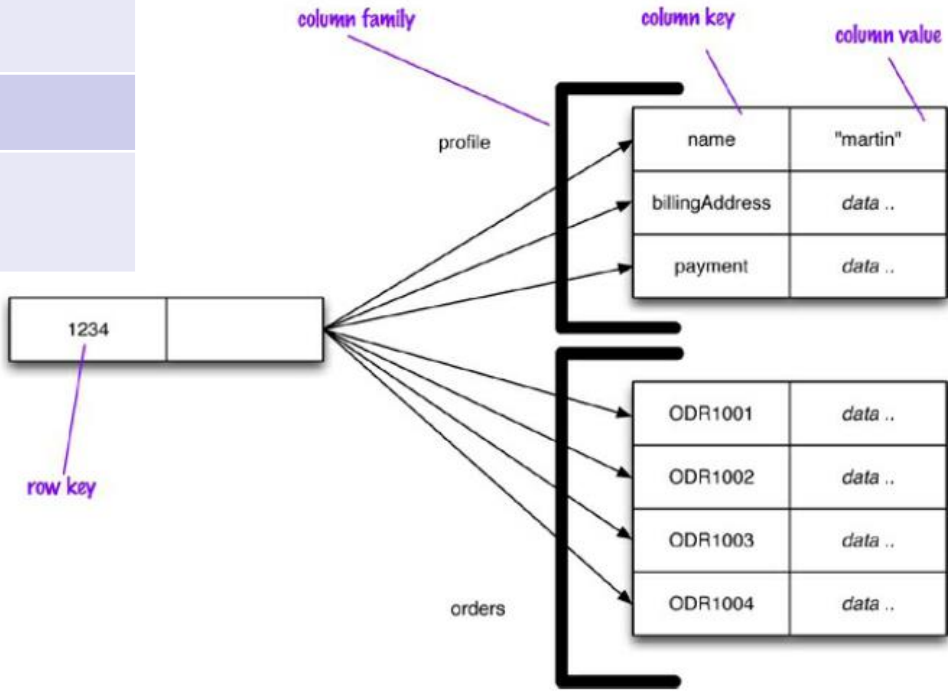
- Graphs with nodes and edges
  - Nodes and edges can have properties
  - Edges have directions
- Queries traverse graph structure
- Stronger consistency and transaction model than in other NoSQL systems



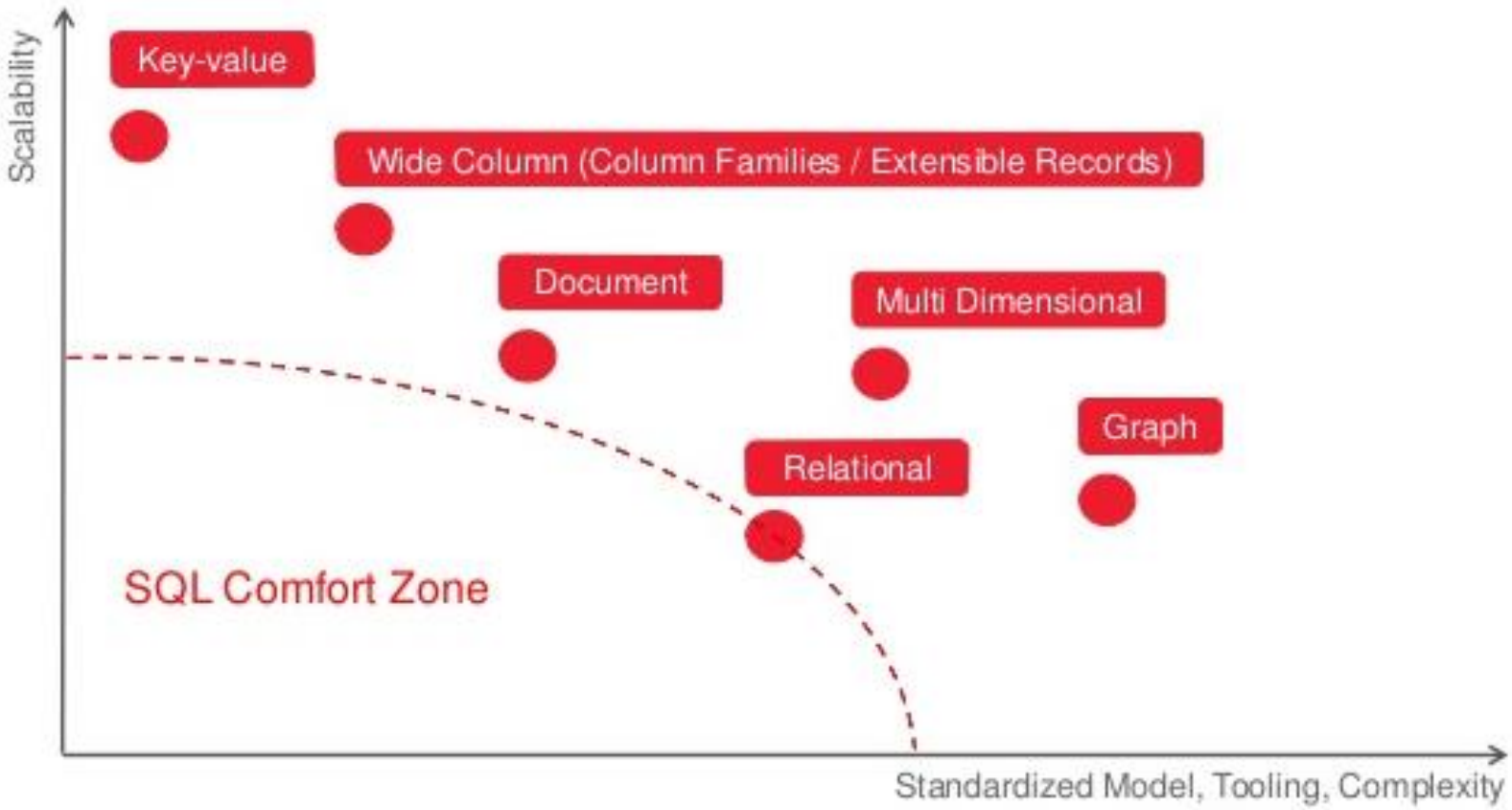
Column-Oriented Data Model

- Similar to key-value data model, but multiple maps
  - One map for different column families
- Not just the Relational Data Model with a column-oriented storage

Relational	Column-oriented DB (e.g., Cassandra)
Database	Keyspace
Table	Column Family
Row	Row
Column (same for all rows)	Column (can be different per row)



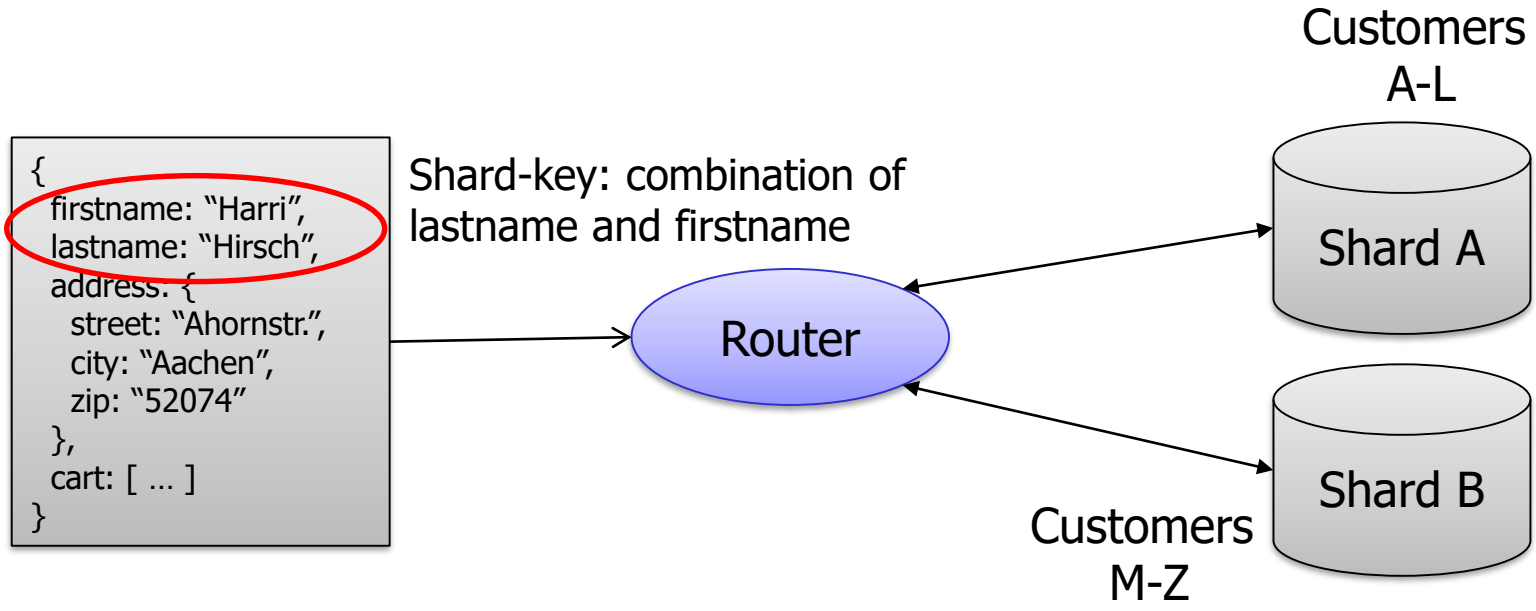
Classification of Data Models



G. Schmutz, P. Salvisberg: SQL vs. NoSQL. https://www.slideshare.net/gschmutz/sql-vs-nosql-43786447

Sharding

- Disjoint subsets of the data are distributed to different servers ➔ increases performance of reads and writes
- Related data should be stored in one server
- Data should be evenly distributed between servers
- NoSQL systems support *auto-sharding*, which requires definition of a *shard-key*
- Sharding can be combined with replication

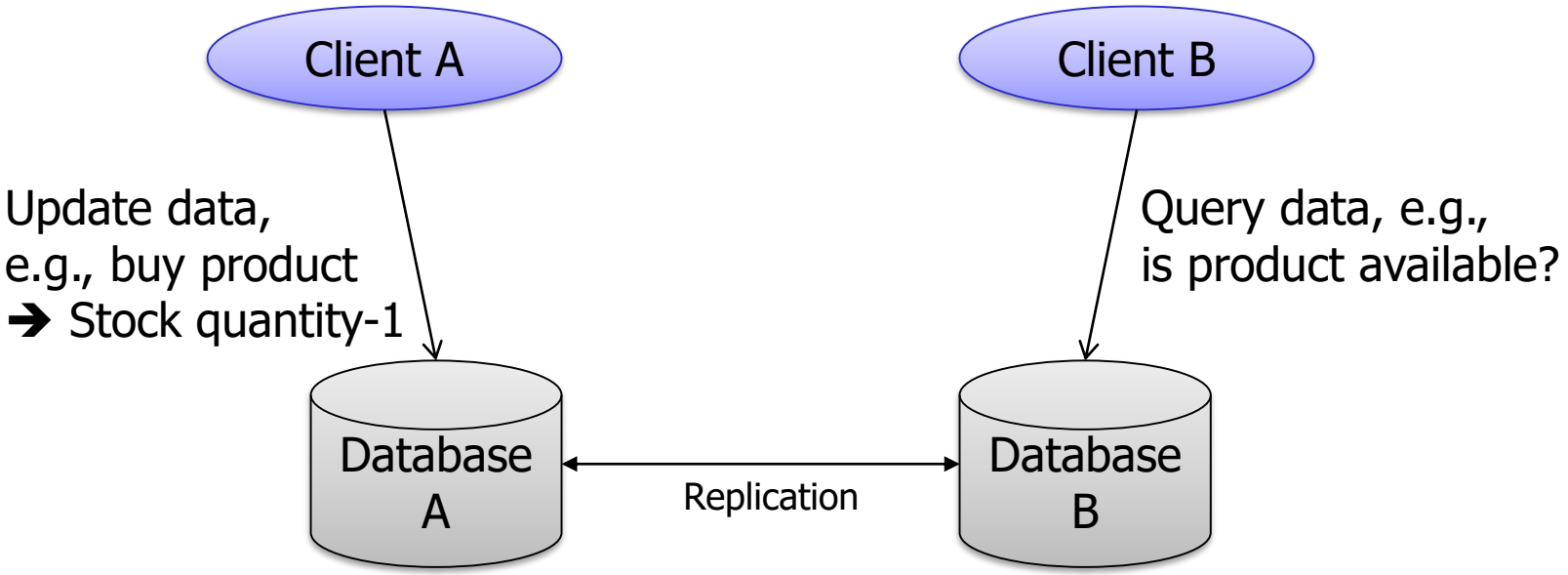


Eventual Consistency

- Replication reduces performance of writes, because data has to be written to multiple nodes
  - Other transactions have to wait until data is committed on all nodes

- Inefficient for distributed web applications

➔ Eventual consistency: clients may see inconsistent intermediate data, as changes are replicated with a delay



Schemaless

- Most NoSQL systems do not require a schema definition, but an application based on NoSQL databases is not really schemaless

- Schema is defined at different level
  - Application code
    Implicitly defined in the code, e.g., checking for attributes
  - Web services
    Explicitly defined by WSDL / XML Schema

- Advantage of NoSQL systems are their flexibility and ease-of-use with respect to schemas
  - However, schema changes can be done in SQL, too

Summary

- Classical RDBMS did not met the requirements of web applications, especially in terms of scalability and fault tolerance

- NoSQL systems offer simple, but yet powerful mechanisms for scalability and programmability
  - However, in terms of abstraction and declarative query languages, they are a step backwards

- Be careful when looking at performance charts, they are sometimes comparing apples & oranges
  - Most systems are best for at least one certain application

➔ **Polyglot persistence**
Organizations will use more than one DBMS technology

4.2 Big Data Systems & Architectures

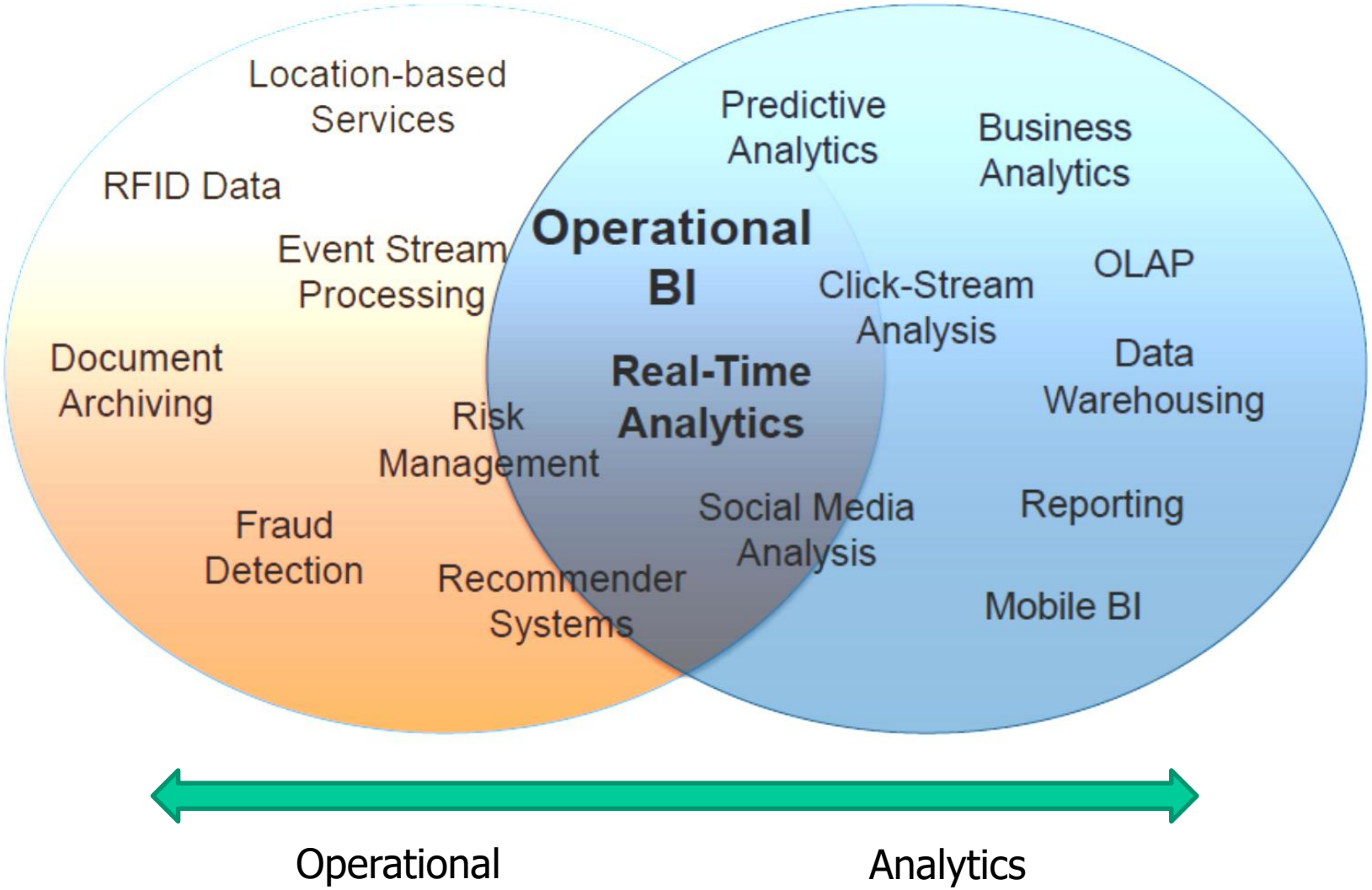
- The n Vs of Big Data (n>=3)
  - Volume
  - Velocity
  - Variety
  - Value
  - Veracity
  - Variability
  - Validity
  - Vulnerability
  - Volatility
  - Visualization
  - …

https://tdwi.org/articles/2017/02/08/10-vs-of-big-data.aspx

Big Data Applications

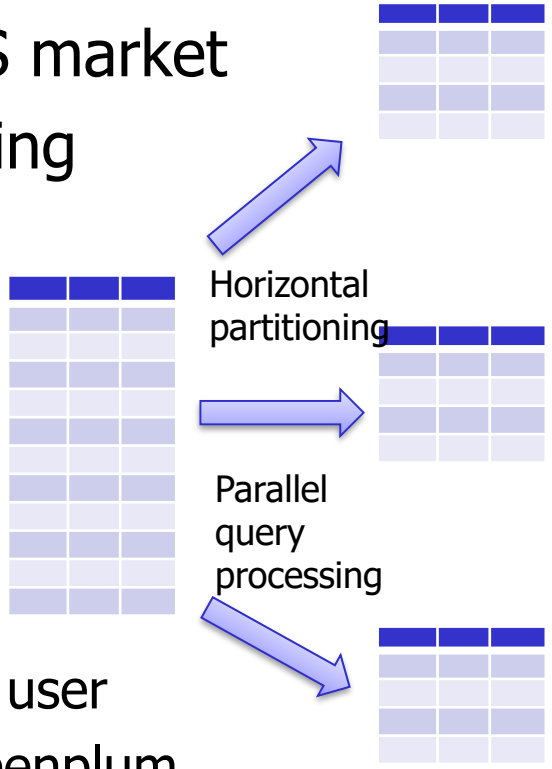
- Marketing
  - What is interesting for our customers?
  - What is the reaction of the customer to certain marketing actions?
- Production („Industrie 4.0“)
  - Quality Control
  - Process Optimization
  - Predictive Maintenance
- Logistics
  - Optimization of storage and product streams
- Controlling/Finance
  - Prediction
  - Risk Analysis
- IT
  - Performance Analysis
  - Security

Characteristics of Big Data Applications



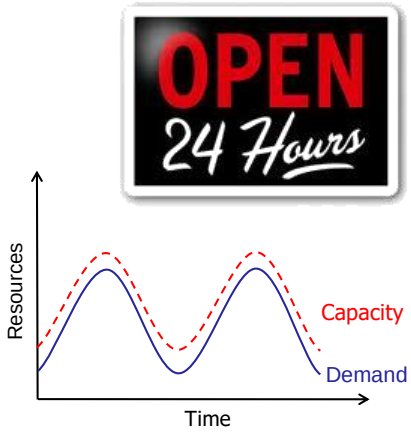
Historical Perspective: (Big) Data Processing before ~2005

- Relational DBMS are dominating the DBMS market
- Parallel DBMS for large scale data processing
  - Shared-nothing architecture
  - Horizontal partitioning
  - Massive parallel processing technology (MPP)
  - Well-designed schema and distribution
  - Declarative query processing with SQL
  - Transaction management based on ACID
  - Parallelization & distribution is transparent for user
  - Example systems: Teradata, IBM Netezza, Greenplum, … (costs >>1m EUR)



Limitations of (Parallel) Relational DBMS

- Parallel Relational DBMS are good when
  - Applications are mostly known a-priori (schema is somehow fixed)
  - You need complex queries across relations, expressible in SQL
- In the area of Big Data, you need systems with support for
  - Flexible schemas and semi-structured data
  - Complex data analytics
  - Ad-hoc analytics with short setup times
  - Easy scalability
  - Fault tolerance



Big Data Architectures

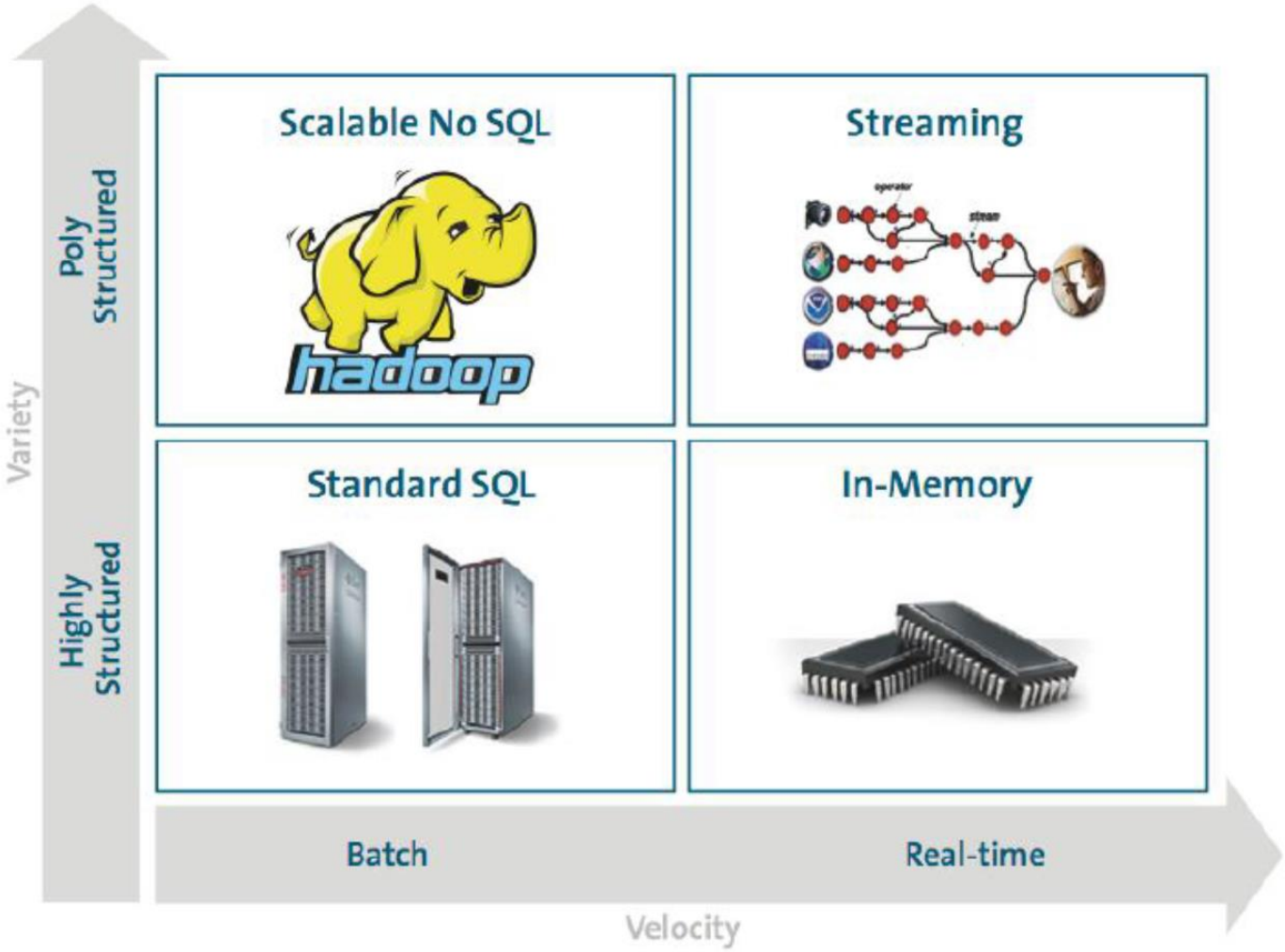
(from chapter 1)

- Big Data requires large, scalable, distributed architectures
- Heterogeneity
  - Sources
  - Systems
  - Requirements
  - Client Applications

➔ Big Data Systems are complex eco-systems in which several independent components have to be integrated

➔ Hadoop is not a Big Data system, it is just a component (but an important component which provides the basic underlying infrastructure for many Big Data systems)

Big Data Technologies



Quelle: Big Data Technologien, Bitkom Leitfaden, 2014 (http://www.bitkom.org/de/publikationen/38337_78776.aspx)

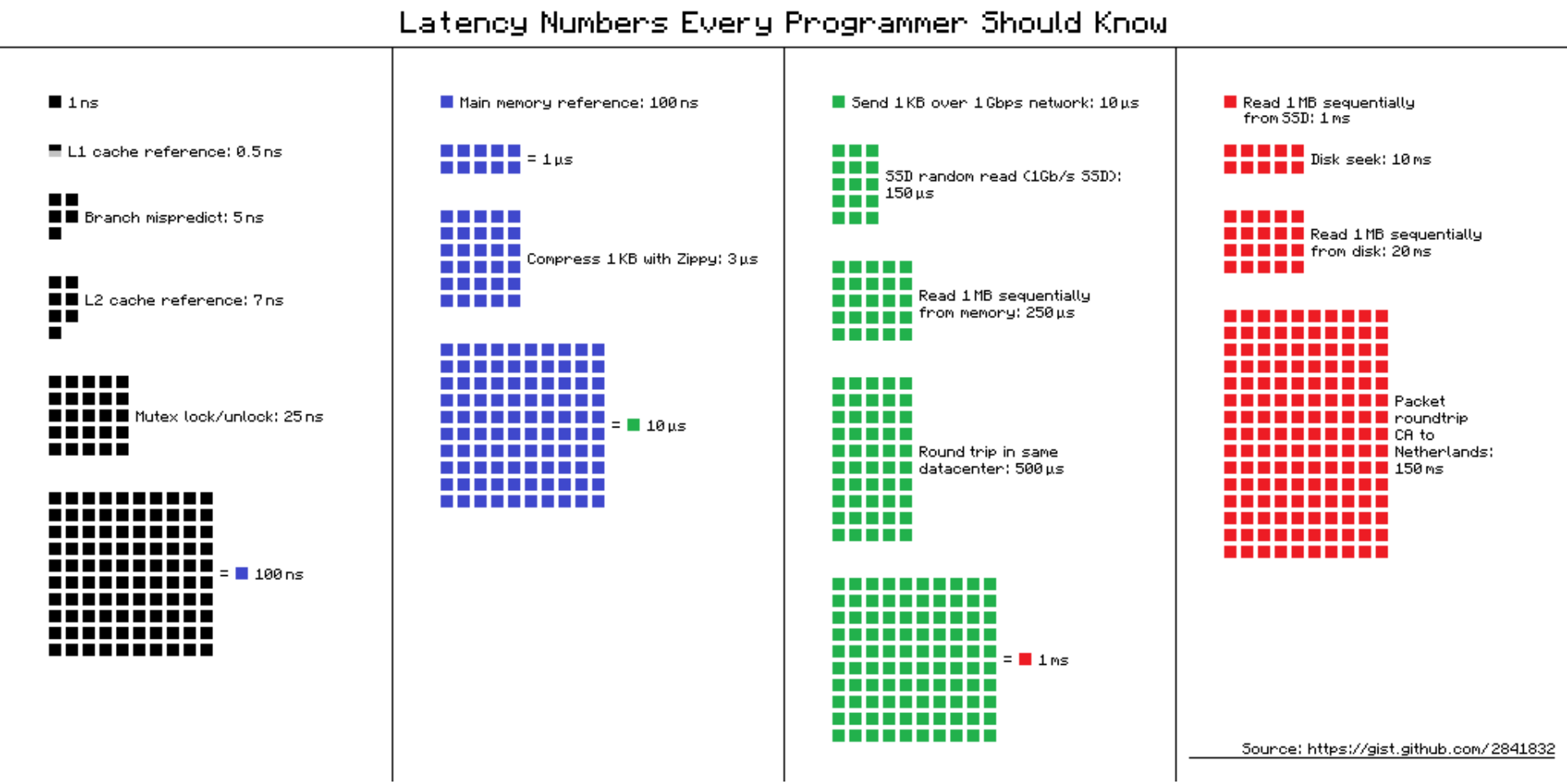
Latency Numbers Every Programmer Should Know

https://gist.github.com/jboner/2841832
http://norvig.com/21-days.html#answers

Memory	ns	µs	ms	Remark
L1 cache reference	0.5			
Branch mispredict	5			
L2 cache reference	7			14x L1 cache
Mutex lock/unlock	25			
Main memory reference	100			20x L2 cache
Compress 1K bytes with Zippy	3,000	3		
Send 1K bytes over 1 Gbps network	10,000	10		
Read 4K randomly from SSD	150,000	150		~1GB/sec SSD
Read 1 MB sequentially from memory	250,000	250		
Round trip within same datacenter	500,000	500	0.5	
Read 1 MB sequentially from SSD*	1,000,000	1,000	1	~1GB/sec SSD, 4X memory
Disk seek	10,000,000	10,000	10	20xdata center round trip
Read 1 MB sequentially from disk	20,000,000	20,000	20	80x memory, 20X SSD
Send packet US→Europe→US	150,000,000	150,000	150	

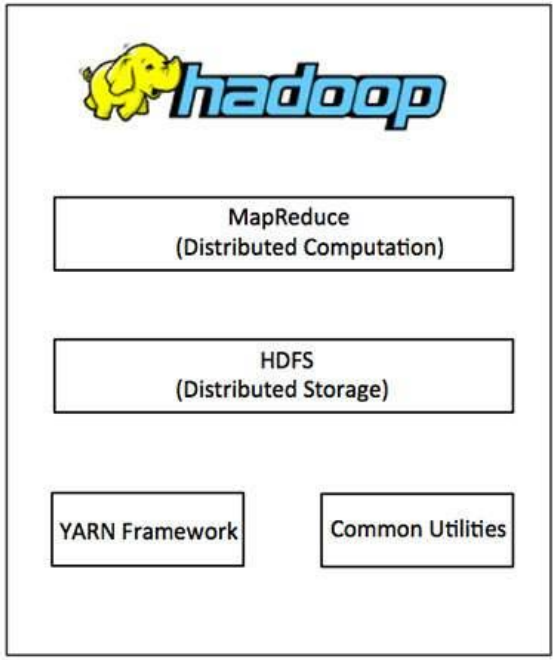
Latency Numbers Every Programmer Should Know

http://i.imgur.com/k0t1e.png



4.2.1 Hadoop
The first Big Data System

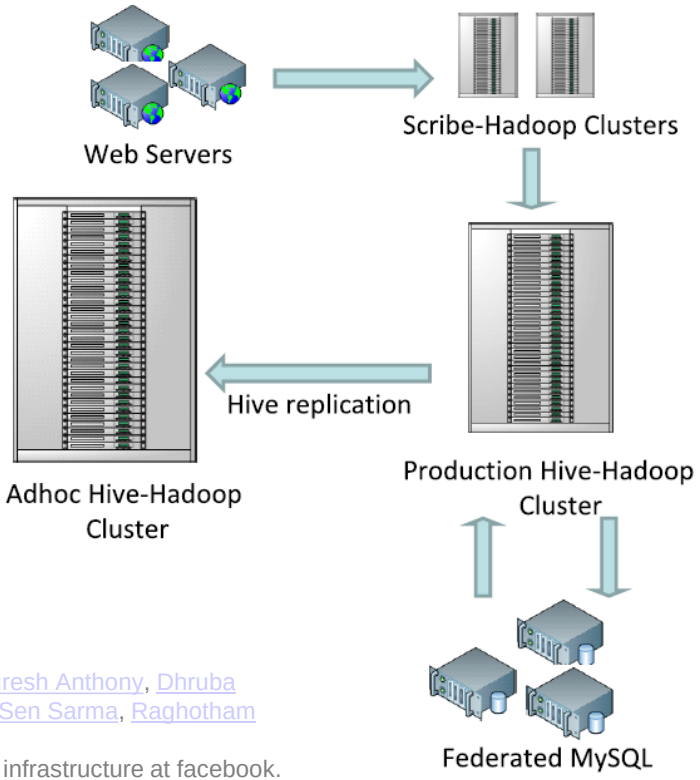
- Inspired by Google's work on GFS (Google File System) and MapReduce
  - Used internally by Google for their data management
- Hadoop 0.1 was released 2006, 1.0 in 2012
- Major components
  - MapReduce for efficient distributed computation
  - HDFS (Hadoop File System) as distributed file system or data store
  - YARN for resource management
- Hadoop is available as a basic service in many cloud computing platforms (e.g., Azure, AWS, Google, …)



https://www.tutorialspoint.com/hadoop/hadoop_introduction.htm

Hadoop: Some Success Stories

- Last.fm
  - Internet radio, music community, …
  - >40 millions of users, >12 million audio tracks
  - Hadoop cluster with 50 nodes and >100 TB data (as of 2010)
- Facebook
  - Data Warehouse based on Hadoop cluster
  - Copying of data between multiple clusters
  - Data sources
    - Web server logs (30 TB data per day)
    - Federated MySQL



Ashish Thusoo, Zheng Shao, Suresh Anthony, Dhruba Borthakur, Namit Jain, Joydeep Sen Sarma, Raghotham Murthy, Hao Liu:
Data warehousing and analytics infrastructure at facebook.
SIGMOD Conference 2010: 1013-1020

Data Processing in RDBMS vs. Hadoop

- **Classical Relational DBMS (Schema on write)**
  - Design schema & create corresponding tables in DBMS
  - Load data
    - In case of heteregeneous data sources: create mappings from the sources to your DB schema, define ETL processes (Extract-Transform-Load) to load the data into the database and execute them

  ➔ Queries are possible, system is ready for users
- **Big Data systems (Hadoop & Co., Schema on read)**
  - Load data
    - Copy data as-is into HDFS
  - Map SQL queries to data files

  ➔Queries are possible, system is ready for users

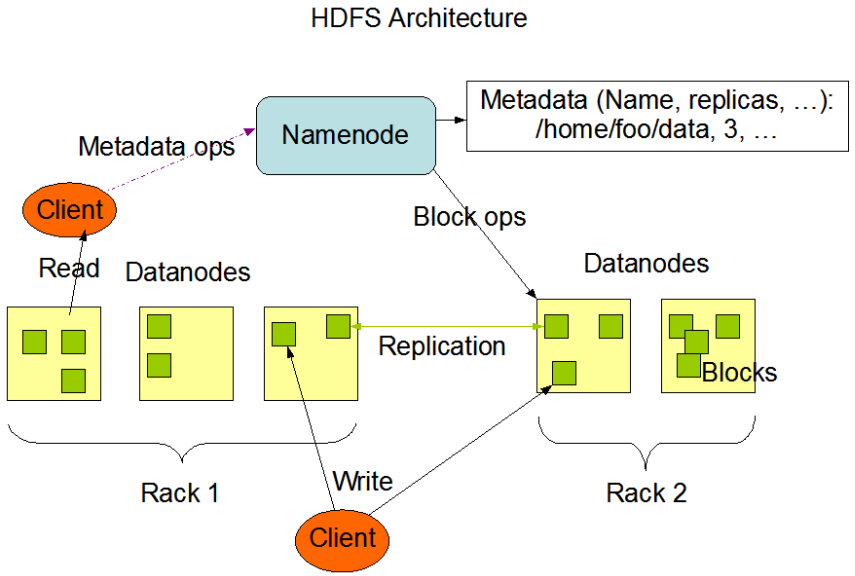
  ➔Shorter time to have a running system, although mapping of SQL queries to Map/Reduce jobs in Hadoop might be complex

HDFS: Main Features

- Basic file system functionality
  - block-oriented storage of arbitrary files in hierarchical directory structure
- Storage of large files across multiple machines
- Immutable files: write-once-read-many
- Replication of data: by default, each data block is replicated three times
- Fault-tolerant
- Designed to be deployed on low-cost hardware
- Batch-oriented data processing
- Computation is done „close to the data“ (→ MapReduce)
- Implemented in Java to be available on multiple platforms

NameNodes manage the metadata in HDFS

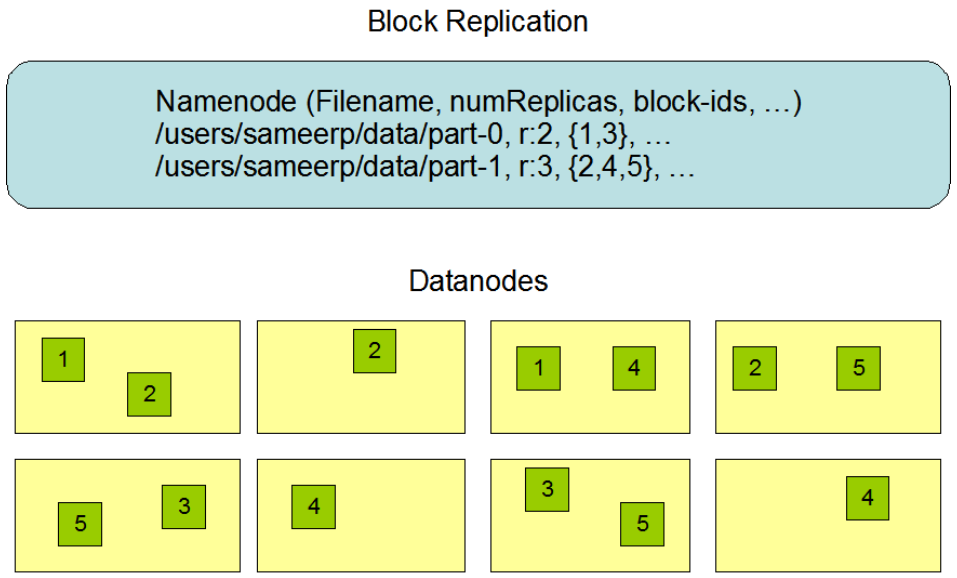
- NameNode
  - Manages metadata (information about files, blocks, replicas, …)
  - Single instance
  - Responsible for distribution and replication (location/rack awareness)
  - Might become bottleneck in case of many small files
  - Since Hadoop 2.0, NameNode can have multiple instances with automatic failover



http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

Data blocks are stored in DataNodes

- DataNode
  - Stores data blocks as assigned by the NameNode
  - Sends Heartbeat to NameNode → Detection of failed nodes
  - Block size typically 128 MB
  - Arbitrary file formats can be used, but new formats (Parquet, AVRO, ORC, …) support
    - Predicate push-down
    - Compression

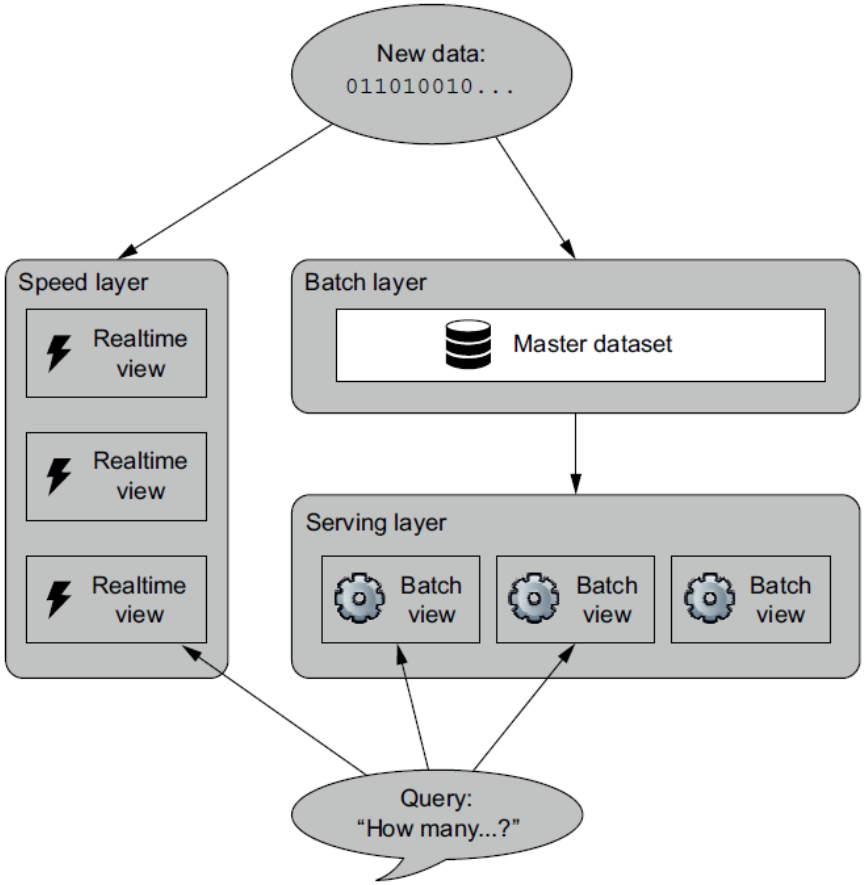


http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

Easy Setup of a Pseudo-Distributed Single Node Cluster

- Fortunately, you don't have to spend $$$ to run a HDFS
  - Download binary from http://hadoop.apache.org/
  - Follow instructions on http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html
  - Format a HDFS: `$ bin/hdfs namenode -format`
  - Start the nodes: `$ sbin/start-dfs.sh`
  - Browse the NameNode at http://localhost:50070/

Lambda Architecture

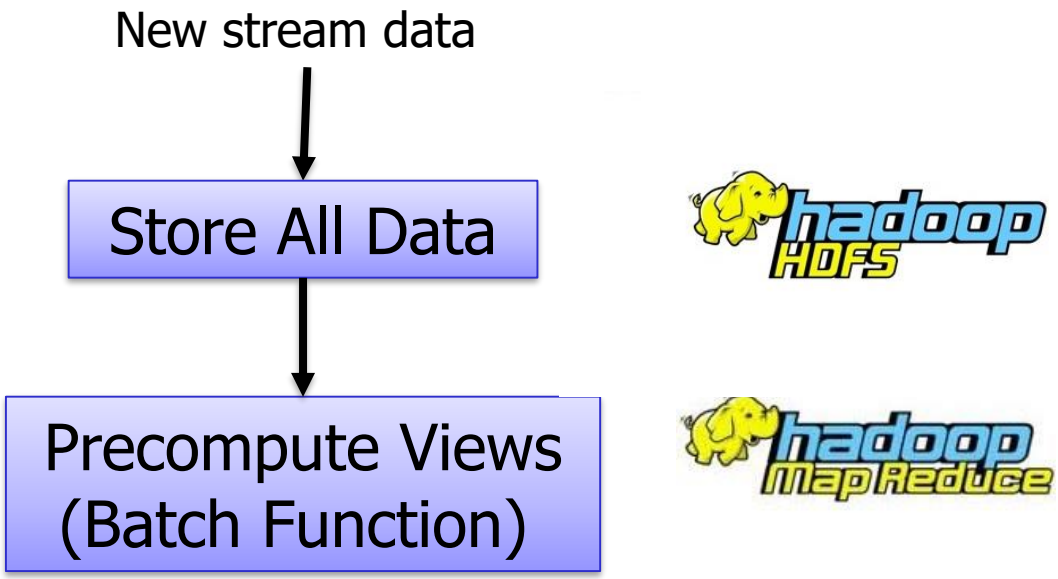


- **Goal:** All-time availability of *all* processed data while keeping up performance
- Incoming data is dispatched to Batch Layer and Speed Layer
- Functions on both layers prepare views on serving layer for querying
- Users can query merged results

Source: Nathan Marz, James Warren, Big Data: Principles and best practices of scalable real-time data systems, Manning, 2015

Lambda Architecture – Batch Layer

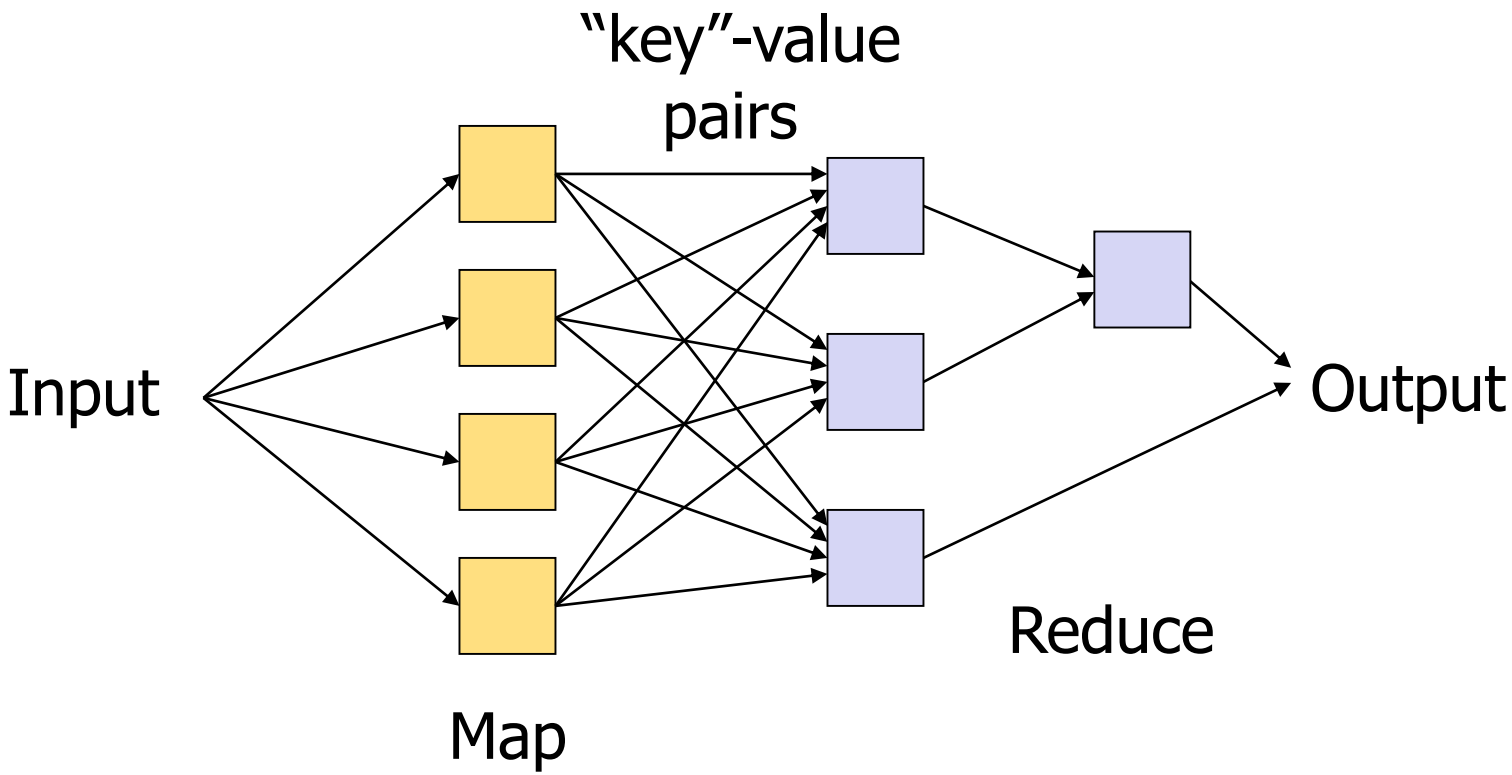
- Manages master data set: immutable, append-only set of raw data
- Distributed precomputation of batch views
- Processes **all** data



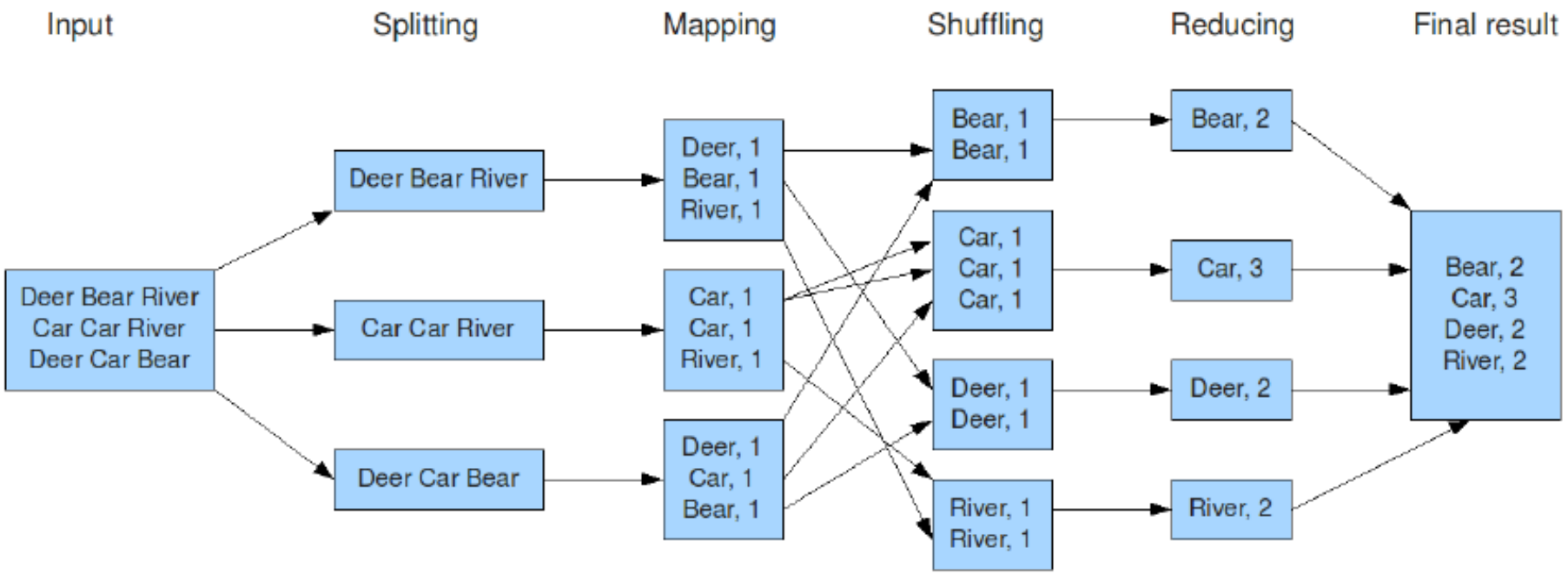
Map-Reduce

- Programming pattern for parallel computation in distributed system
- Function Map(data) → (key, value)
  - Reads input data and emits key-value pairs
  - Keys are not necessarily unique in emitted pairs
- Function Reduce(key, values) → (key, values)
  - Gets input from Map function, a set of values for a single key
  - Input and output structure should be the same
- Map and Reduce jobs can run on different nodes

Map-Reduce



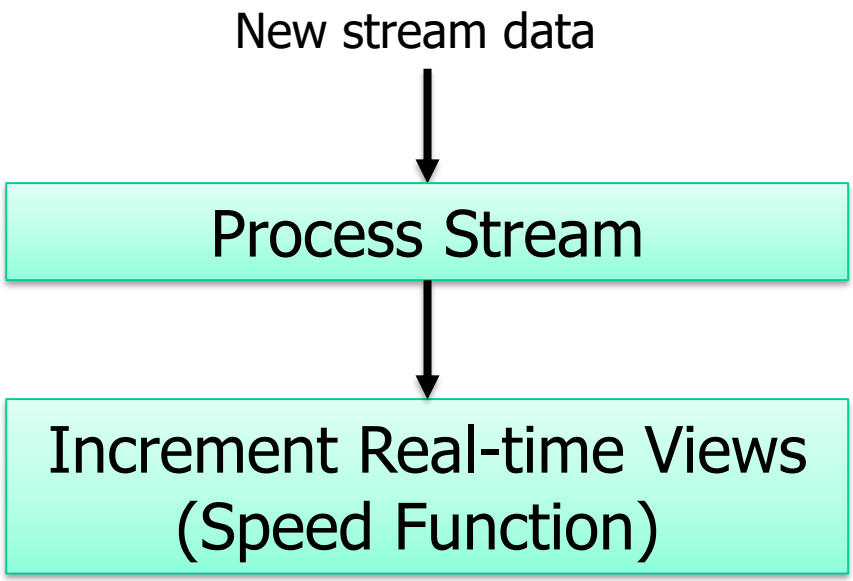
MapReduce Example: Word Count



Picture adapted from: Hadoop Mapreduce Framework in Big Data Analytics, Vidyullatha Pellakuri, Rajeswara Rao

Lambda Architecture – Speed Layer

- Computes **recent** data only
- Stores and updates realtime views
- Compensates high latency of updates at serving layer
- Not 100% accurate or complete



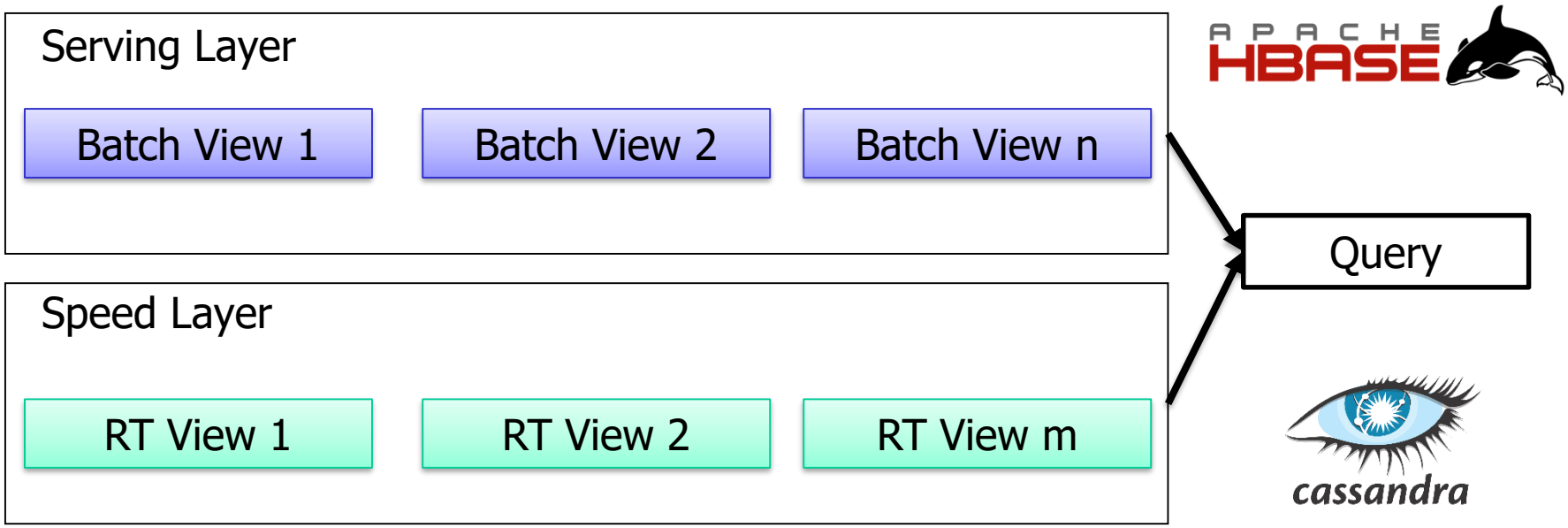
Apache Kafka

STORM

Spark Streaming

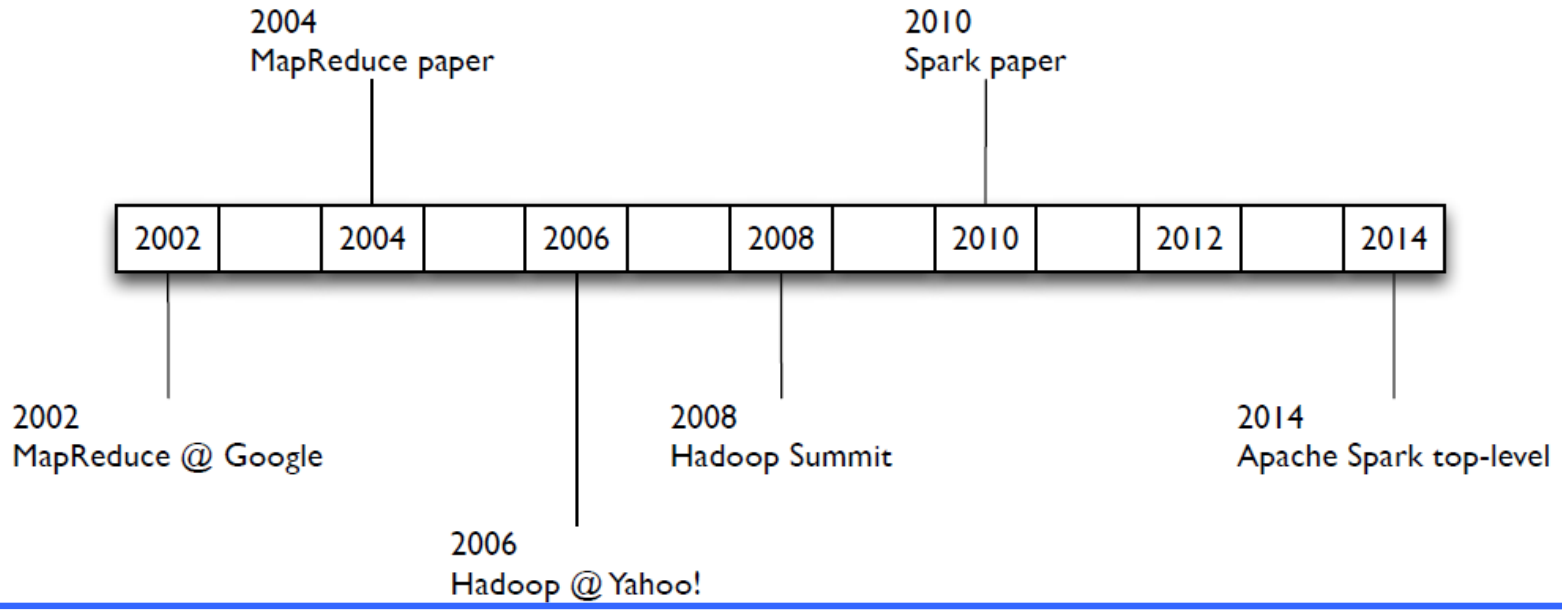
Lambda Architecture – Serving Layer

- Maintains & indexes batch views
- Combines batch and real-time views
- Provides ad-hoc query access to all data



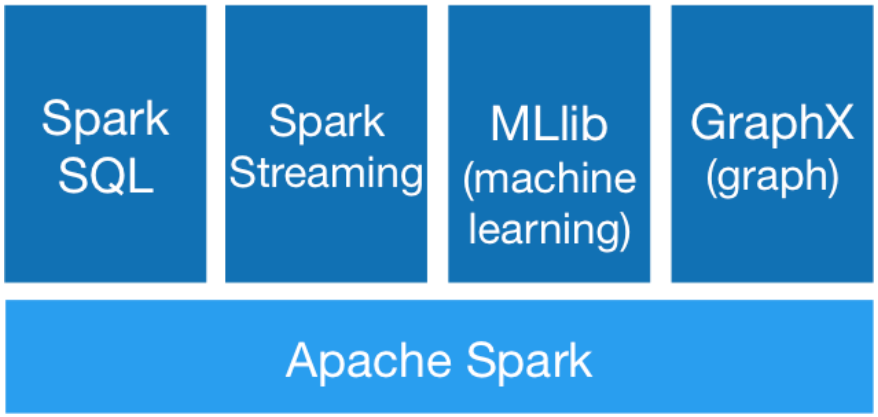
4.2.2 Apache Spark

- A brief introduction to the main concepts of Apache Spark
  - Based on:
    - Intro to Apache Spark, a tutorial by Databricks, available at http://training.databricks.com/workshop/itas_workshop.pdf
    - Courses by Heather Miller on „Parallel Programming and Data Analysis" (http://heather.miller.am/teaching/cs212/slides/) and „Big Data Analysis with Scala and Spark" (https://www.coursera.org/learn/scala-spark-big-data/home/info) at EPFL/Coursera



What is Apache Spark?

- Open Source Cluster Computing Framework (Source: Wikipedia)
- Uses a distributed data structure called RDD (Resilient Distributed Dataset)
- Should avoid multiple reads & writes of data in a map-reduce job as it keeps data in some type of distributed shared memory
- Provides access to multiple heterogeneous sources by providing a common data processing platform
- Key features
  - handles batch, interactive, and real-time within a single framework
  - native integration with Java, Python, Scala
  - programming at a higher level of abstraction
  - more general: map/reduce is just one set of supported constructs
  - lazy evaluation

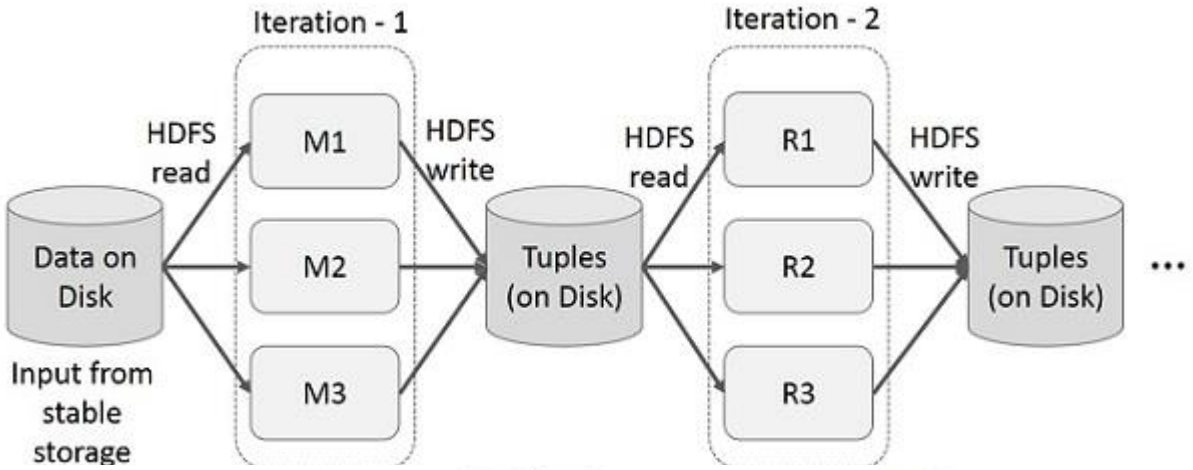


Why yet another system?

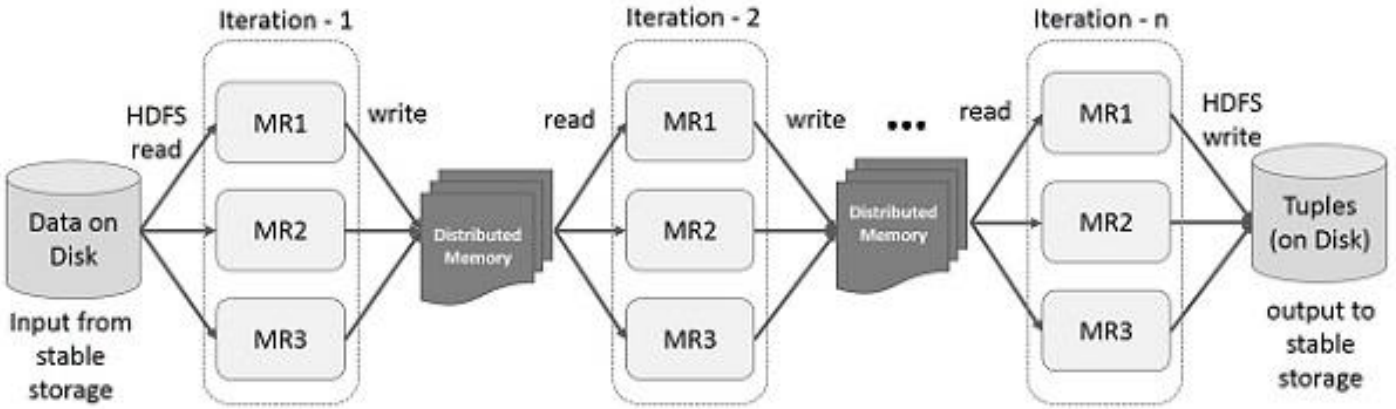
- MapReduce use cases showed two major limitations:
  - difficultly of programming directly in MapReduce
  - performance bottlenecks, or batch not fitting the use cases
- In short, MapReduce does not compose well for large applications
- Therefore, people built *specialized systems* as workarounds…

Data Processing in Hadoop vs. Spark

Hadoop

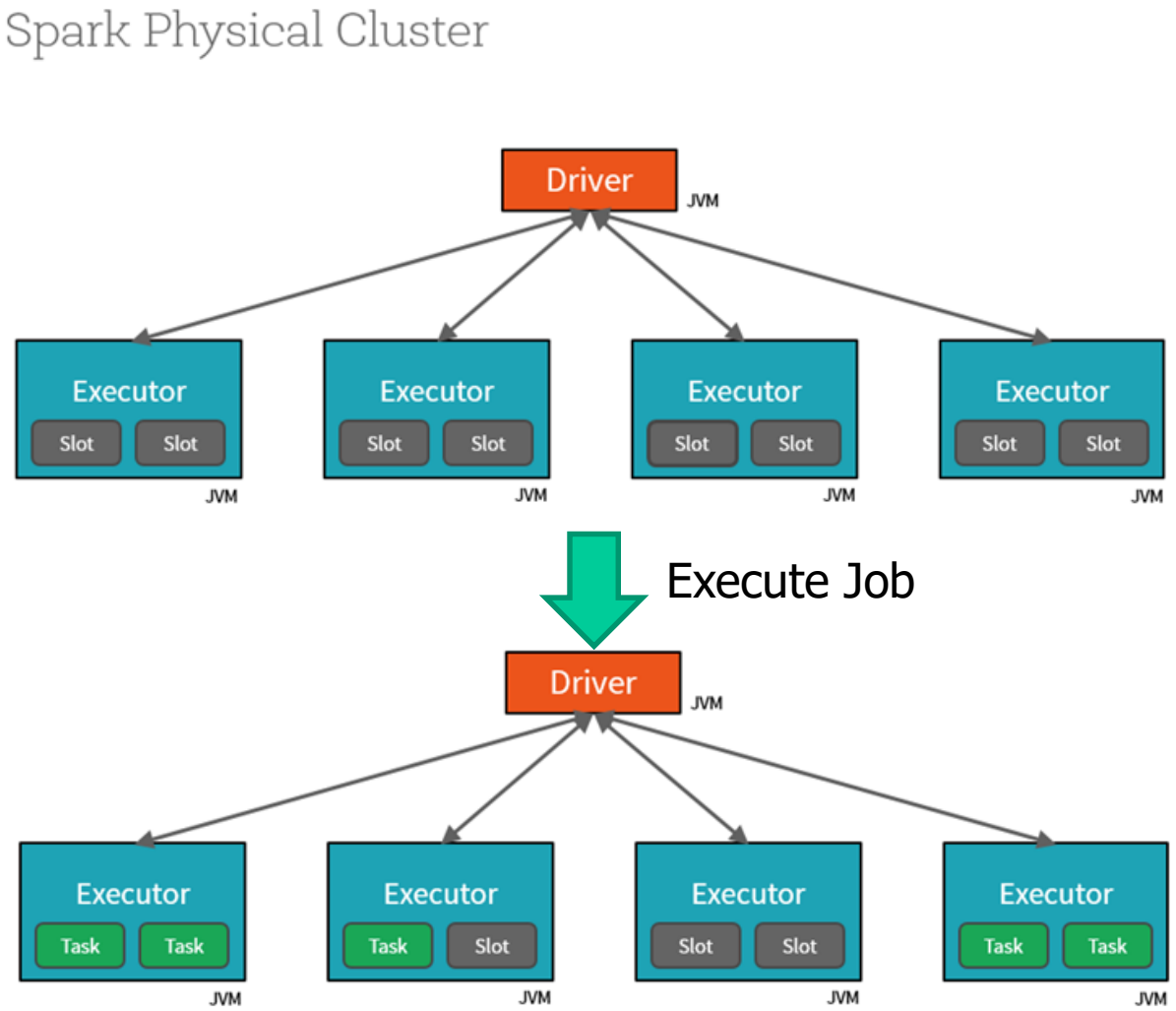


Spark



Source: http://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm

Distributed Computing in Spark



Spark is fast

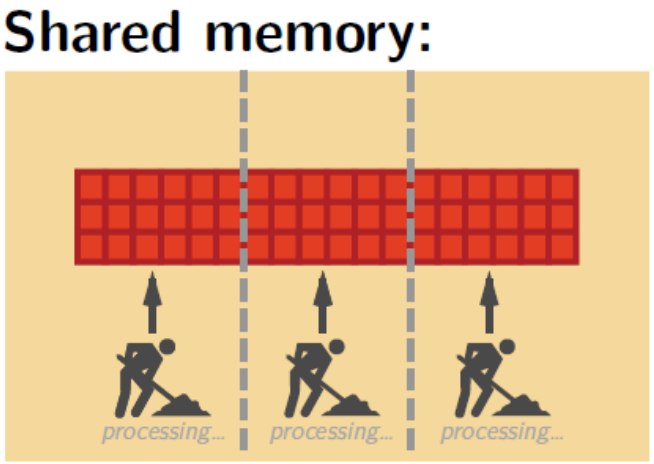
	Hadoop MR Record	**Spark Record**	**Spark 1 PB**
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**

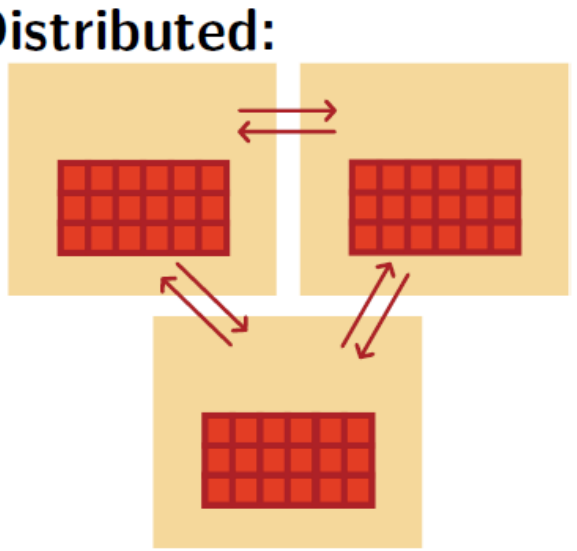
General Remark:
Caution! Don't trust benchmarks too much.

https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

Data-Parallel vs. Distributed Data-Parallel

http://heather.miller.am/teaching/cs212/slides/





- **Shared memory case:** Data-parallel programming model. Data partitioned in memory and operated upon in parallel
- **Distributed case:** Data-parallel programming model. Data partitioned between machines, network in between, operated upon in parallel

RDDs

- **R**esilient **D**istributed **D**atasets (RDD) are the primary abstraction in Spark – a fault-tolerant collection of elements that can be operated on in parallel
  - Implementation of the distributed data-parallel model
  - Similar to (parallel) collections in Scala

- An RDD is a collection of data objects of any type (→ heterogeneity)
- There are also data frames (collection of row objects) and datasets (collection of typed-objects)

Word Count in Apache Spark

```
// Create an RDD
val rdd = spark.textFile("hdfs://...")

val count = rdd.flatMap(line => line.split(" ")) // separate lines into words
               .map(word => (word, 1))           // include something to count
               .reduceByKey(_ + _)               // sum up the 1s in the pairs
```

Operations on RDDs

- Two types of operations on RDDs
  - Transformations (e.g., filter, group, sort, join, etc.)
    - These are lazy: the result is not computed directly, only when an action is performed
    - The transformations applied to one dataset are recorded
      ➔ optimize the sequence of transformations
      ➔ recover from lost data
  - Actions (e.g., aggregation, reduce, output, foreach)

```
val largeList: List[String] = ...
val wordsRdd = sc.parallelize(largeList)
val lengthsRdd = wordsRdd.map(_.length)
val totalChars = lengthsRdd.reduce(_ + _)
```

Transformations

Action

Persistence

- RDDs can be persisted in main memory (RAM) or disk
- Spark can persist (or cache) a dataset in memory across operations
- Each node stores in memory any slices of it that it computes and reuses them in other actions on that dataset
- Use of RAM or disk depends on available RAM and options of the transformation

Level	*Space used*	*CPU time*	*In memory*	*On disk*
MEMORY_ONLY	**High**	**Low**	**Y**	**N**
MEMORY_ONLY_SER	Low	High	Y	N
MEMORY_AND_DISK*	High	Medium	Some	Some
MEMORY_AND_DISK_SER†	Low	High	Some	Some
DISK_ONLY	Low	High	N	Y

Default

* Spills to disk if there is too much data to fit in memory

† Spills to disk if there is too much data to fit in memory. Stores serialized representation in memory.

Partitioning

- The data within an RDD is split into several *partitions*.
- Properties of partitions:
  - Partitions never span multiple machines, i.e., tuples in the same partition are guaranteed to be on the same machine.
  - Each machine in the cluster contains one or more partitions.
  - The number of partitions to use is configurable. By default, it equals the *total number of cores on all executor nodes*.
- Partitioning methods
  - Hash partitioning
  - Range partitioning

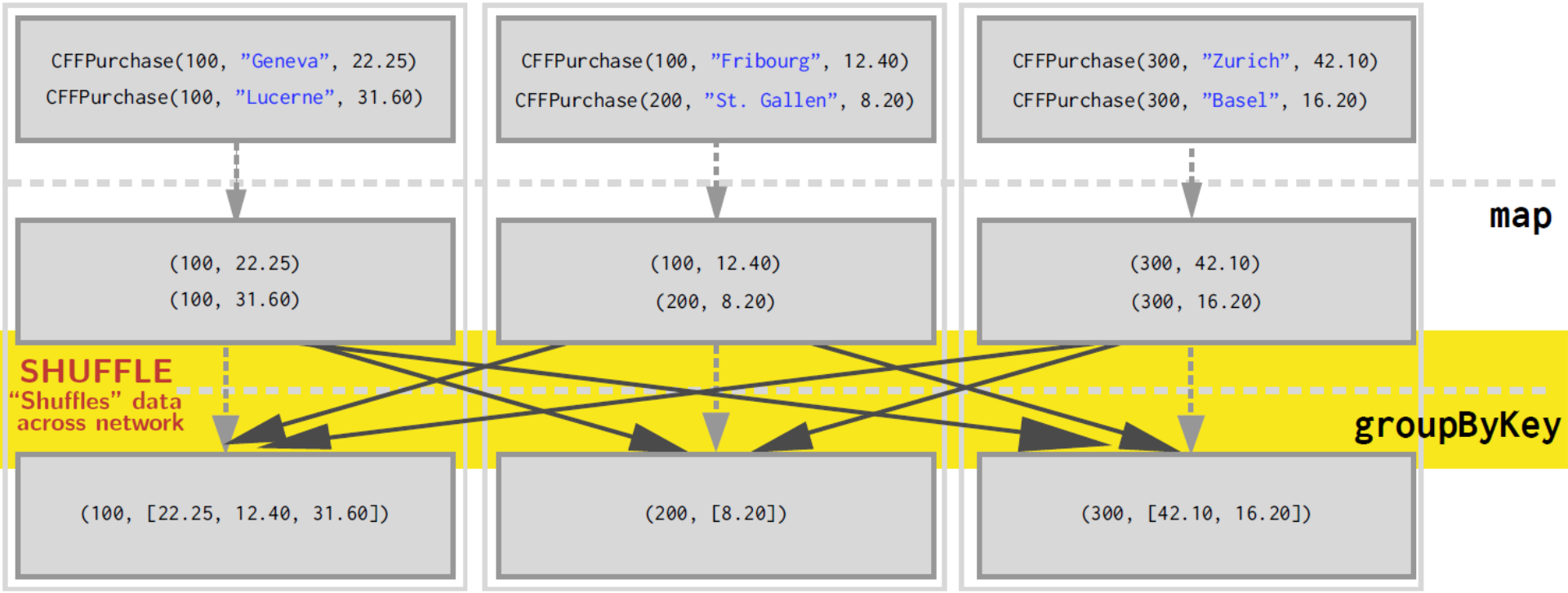
Shuffling

- Shuffling moves RDDs between **different nodes** of a cluster
- It can occur for operations that cannot be processed within a single partition, e.g.,
  - groupByKey
  - join
  - distinct
  - intersection
  - …
- As data has to be transferred between nodes, **network latency** is an issue

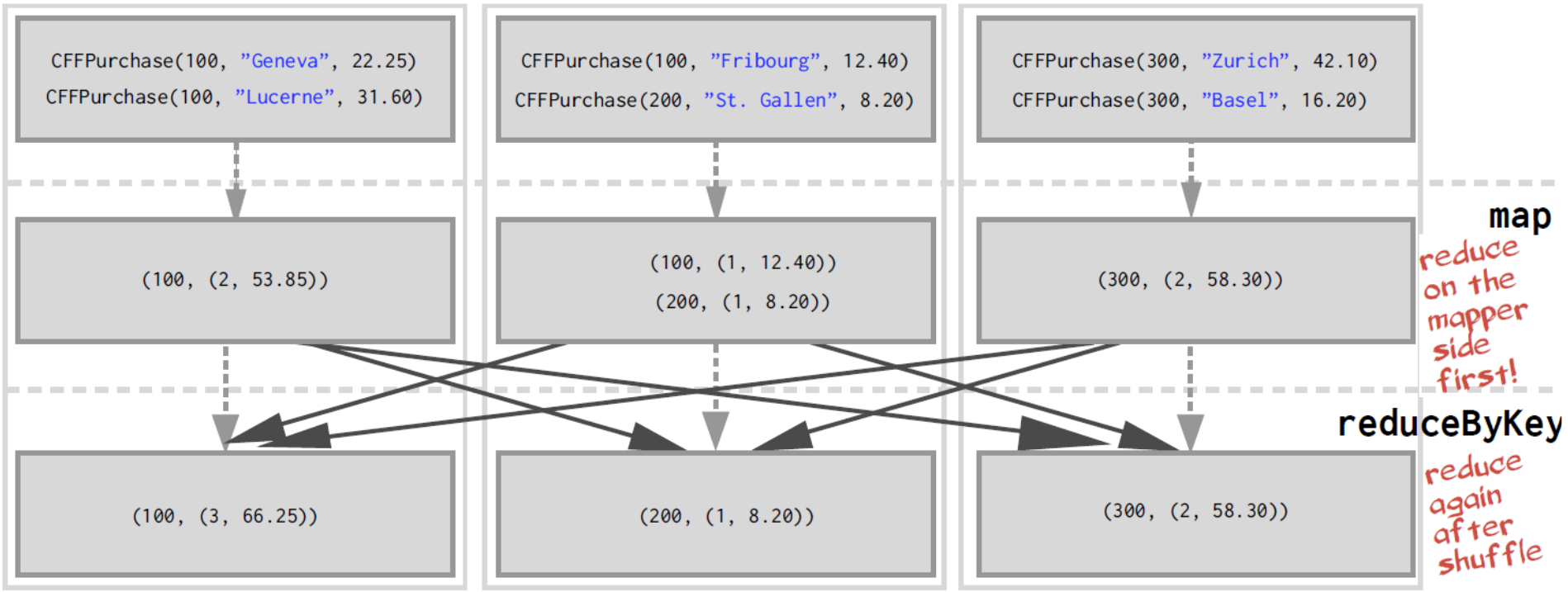
  ➔ Shuffle operation should be avoided!

Shuffle Example

```
case class CFFPurchase(customerId: Int, destination: String, price: Double)
```



Optimization: Reduce before Shuffle

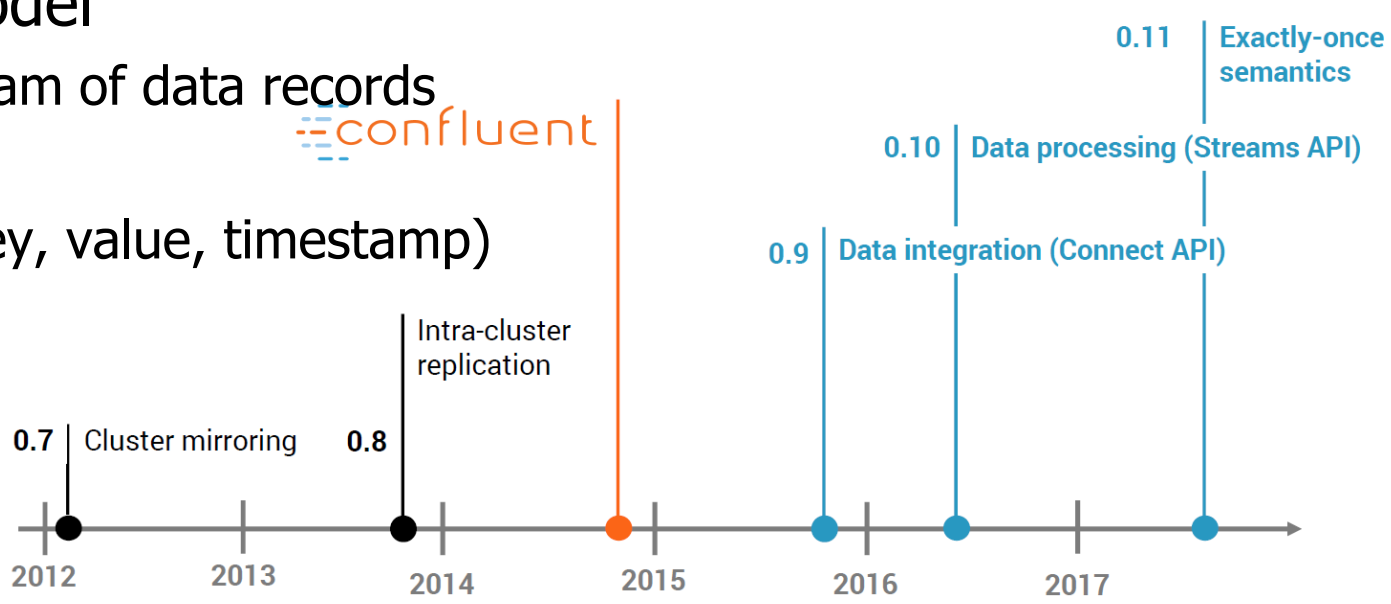


4.2.3 Apache Kafka

http://kafka.apache.org

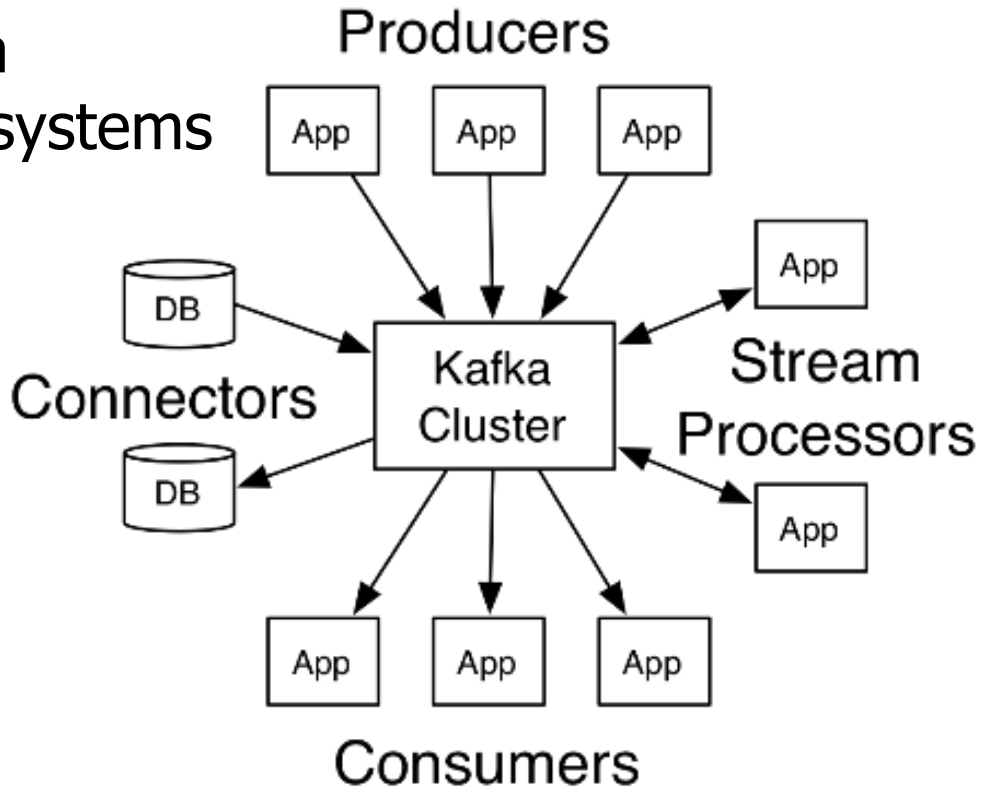
Slides are based on M.J. Sax:
Introduction to Apache Kafka.
http://materials.dagstuhl.de/files/17/17441/17441.MatthiasJ.Sax.Slides.pdf

- Distributed streaming platform
  - Publish & Subscribe to data streams
  - Fault-tolerant
- „Real-time" streaming
  - Data pipelines for reliably transferring data between systems
  - Applications that transform or react to the data streams
- Basic data model
  - Topic = Stream of data records
  - Record = (key, value, timestamp)

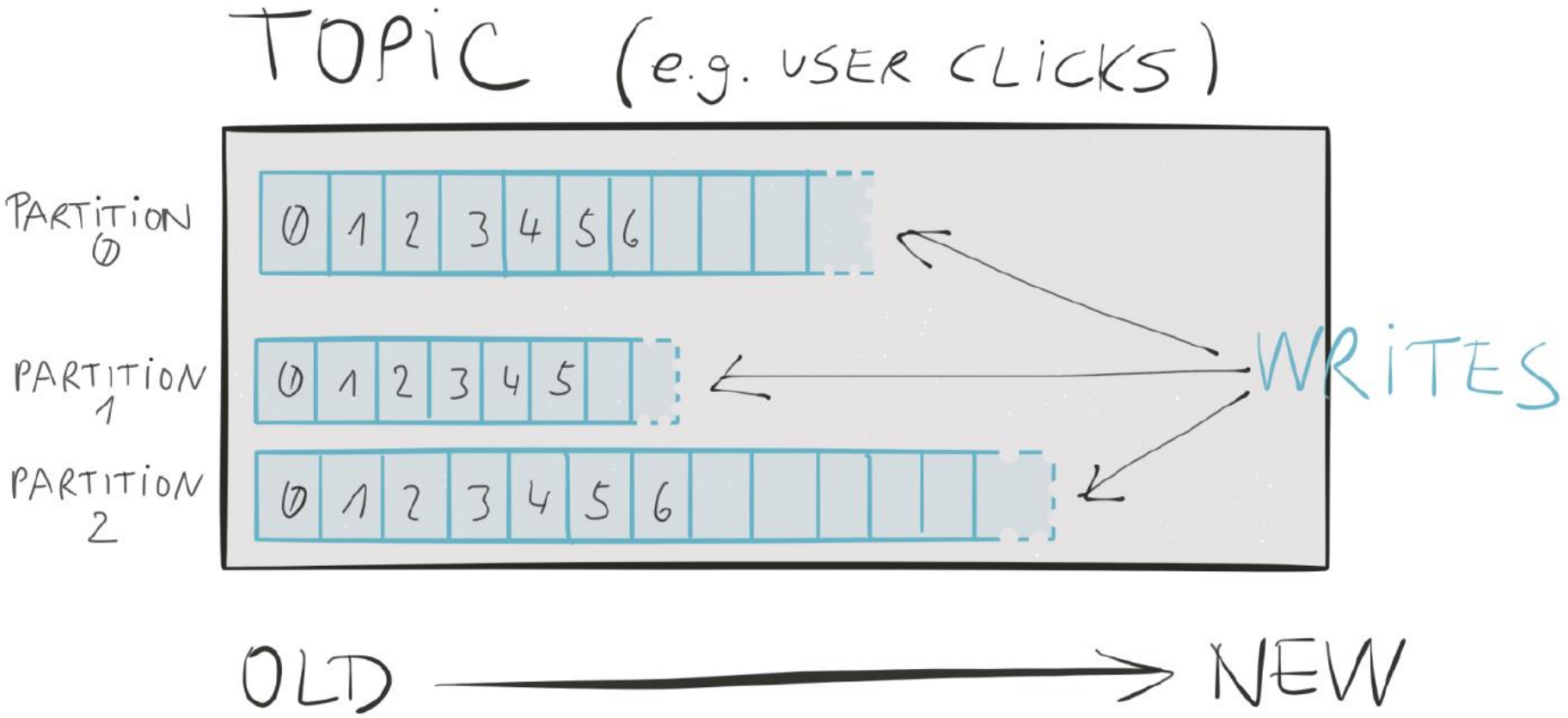


Kafka Architecture

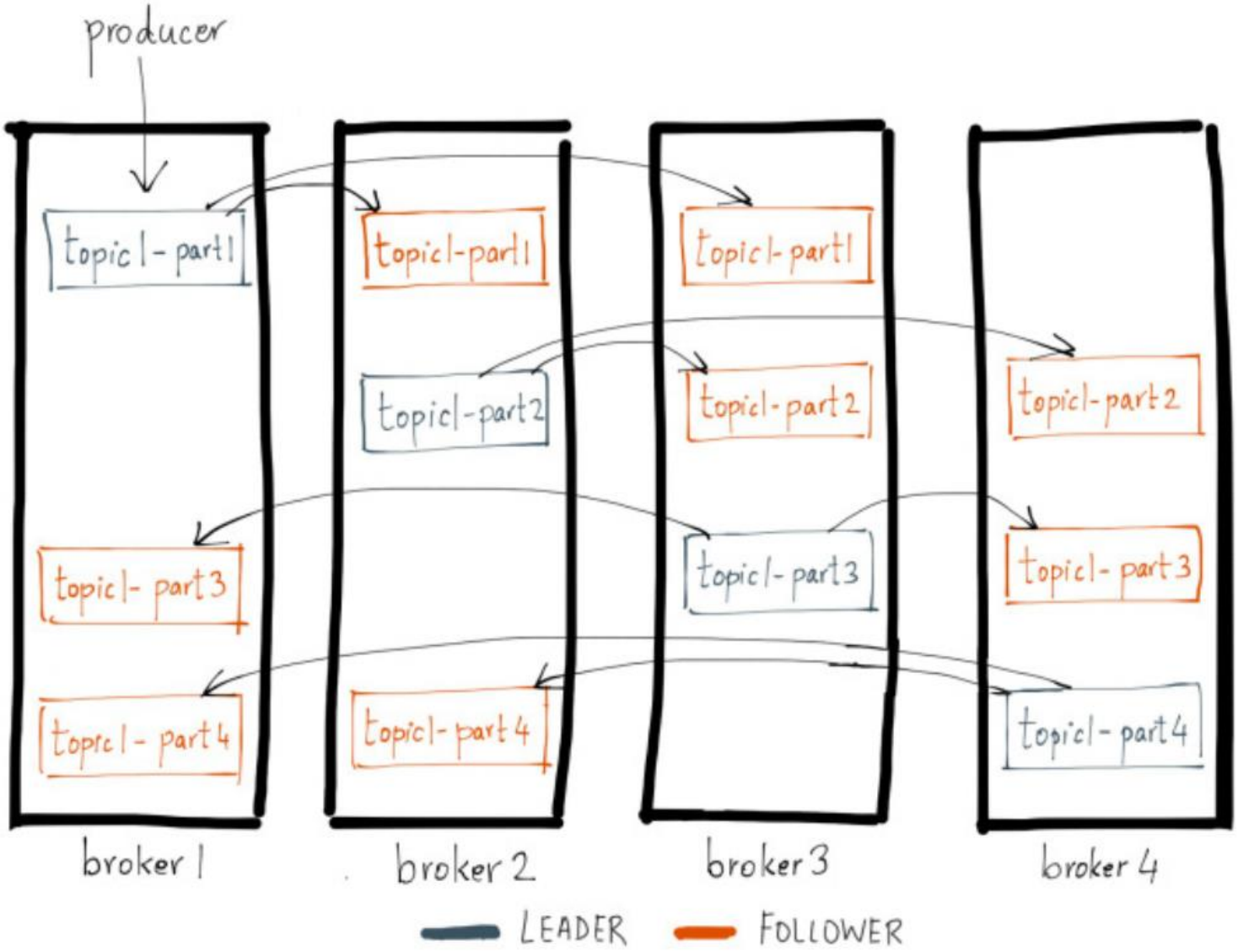
- Kafka started as a kind of a message broker
- Now, it is more a data stream management systems
  - Provides data streams
  - Enables queries



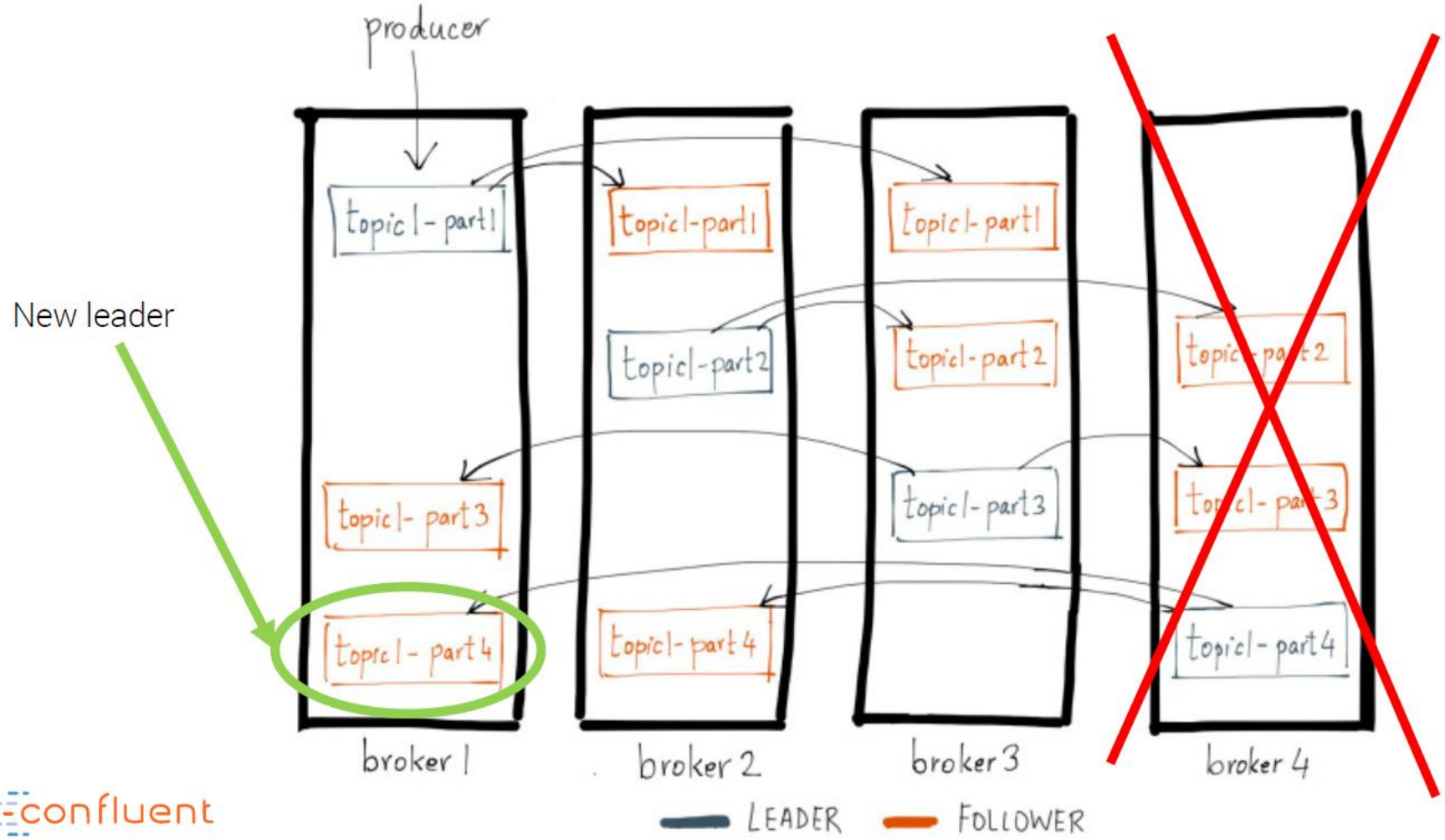
Topics and Partitions



Topic and Partition Placement in a Cluster



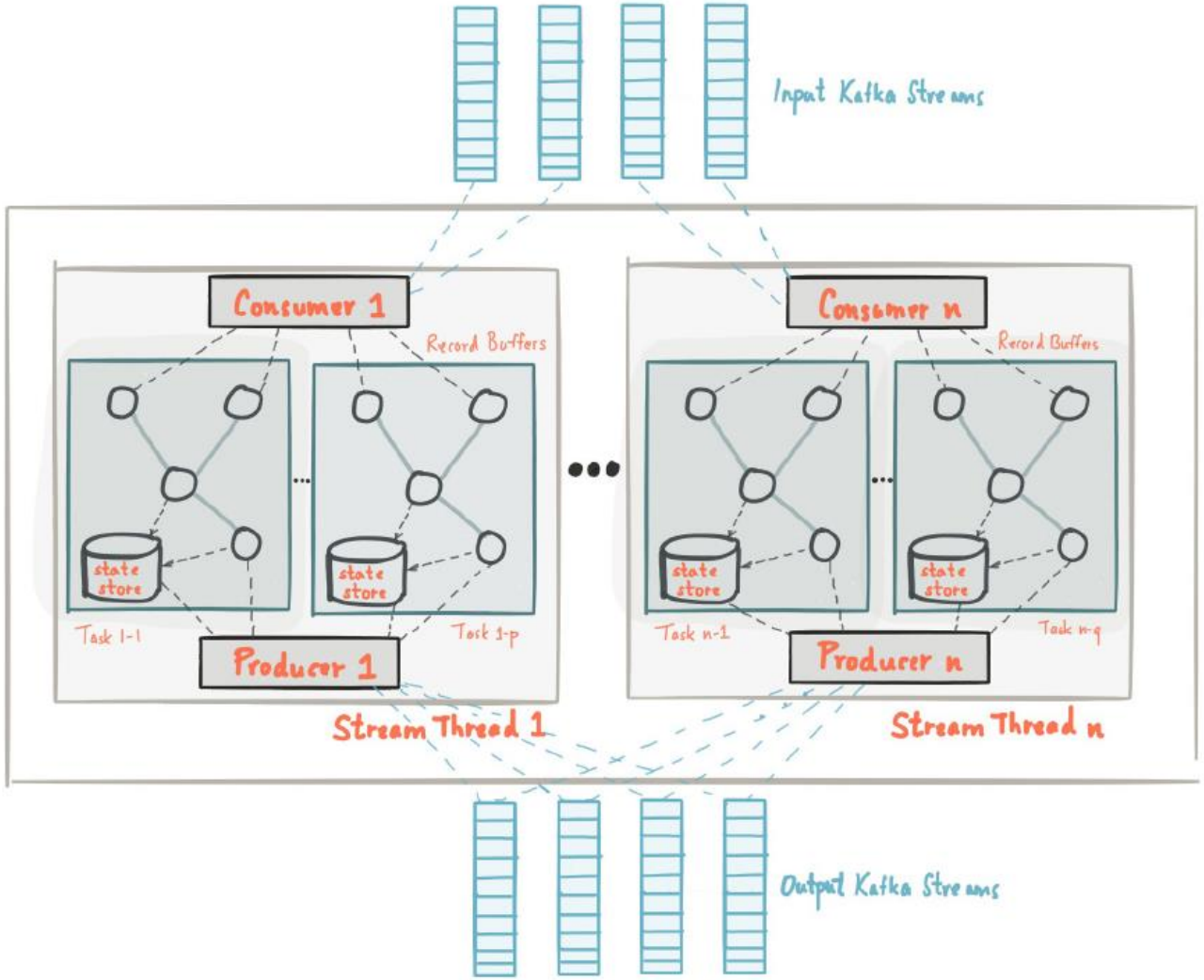
Fault-Tolerance



Kafka Streams API

- Easy to use and powerful DSL plus low level Processor API
  - Record-by-record processing (ms latency)
  - Single message transform (filter, map, etc)
  - Aggregations / Join
  - Windows (time, session)
  - Stream-Table duality
  - Rich time semantics (event time, ingestion time, processing time)
- Elastic, scalable, fault-tolerant (including state)

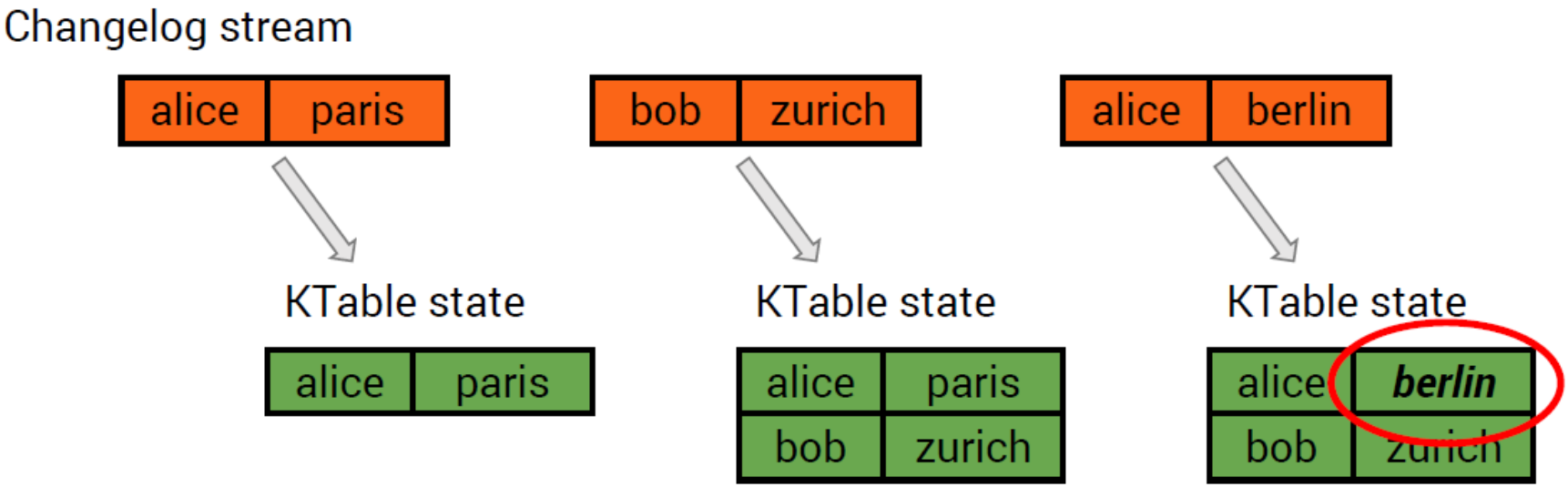
Internal Architecture



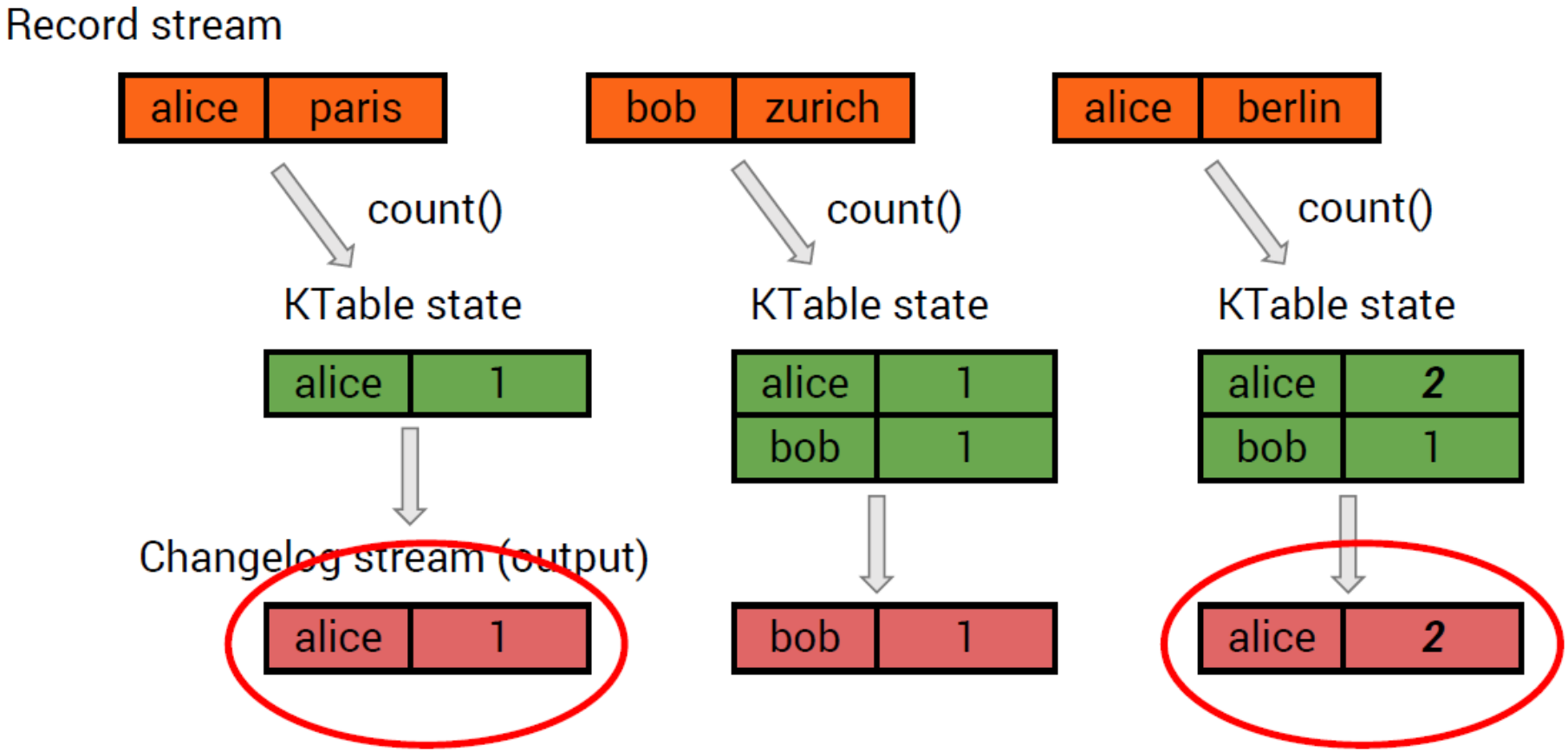
KStreams and KTables

- KStream
  - Record stream
  - Each record describes an event in the real world
  - Example: click stream
- KTable
  - Changelog *stream*
  - Each record describes a *change* to a previous record
  - Example: position report stream
  - In Kafka Streams:
    - KTable holds a materialized view of the latest update per key as *internal* state

KTable Example



Changelog Streams

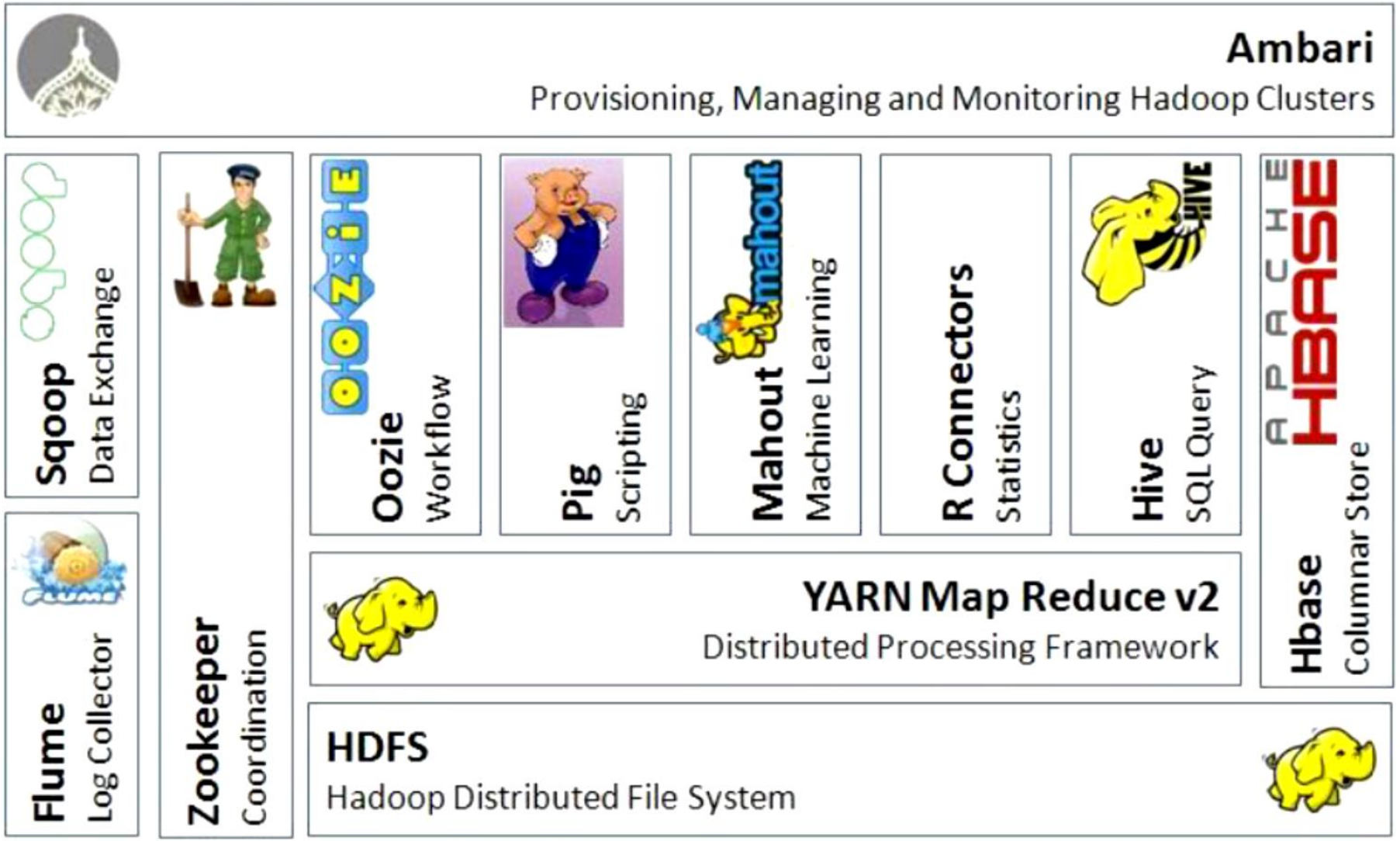


4.3 SQL in Big Data Applications

- As SQL and JDBC/ODBC are well-established standards to access and retrieve data (especially in business intelligence tools), there is the need that Big Data & NoSQL systems provide a SQL interface

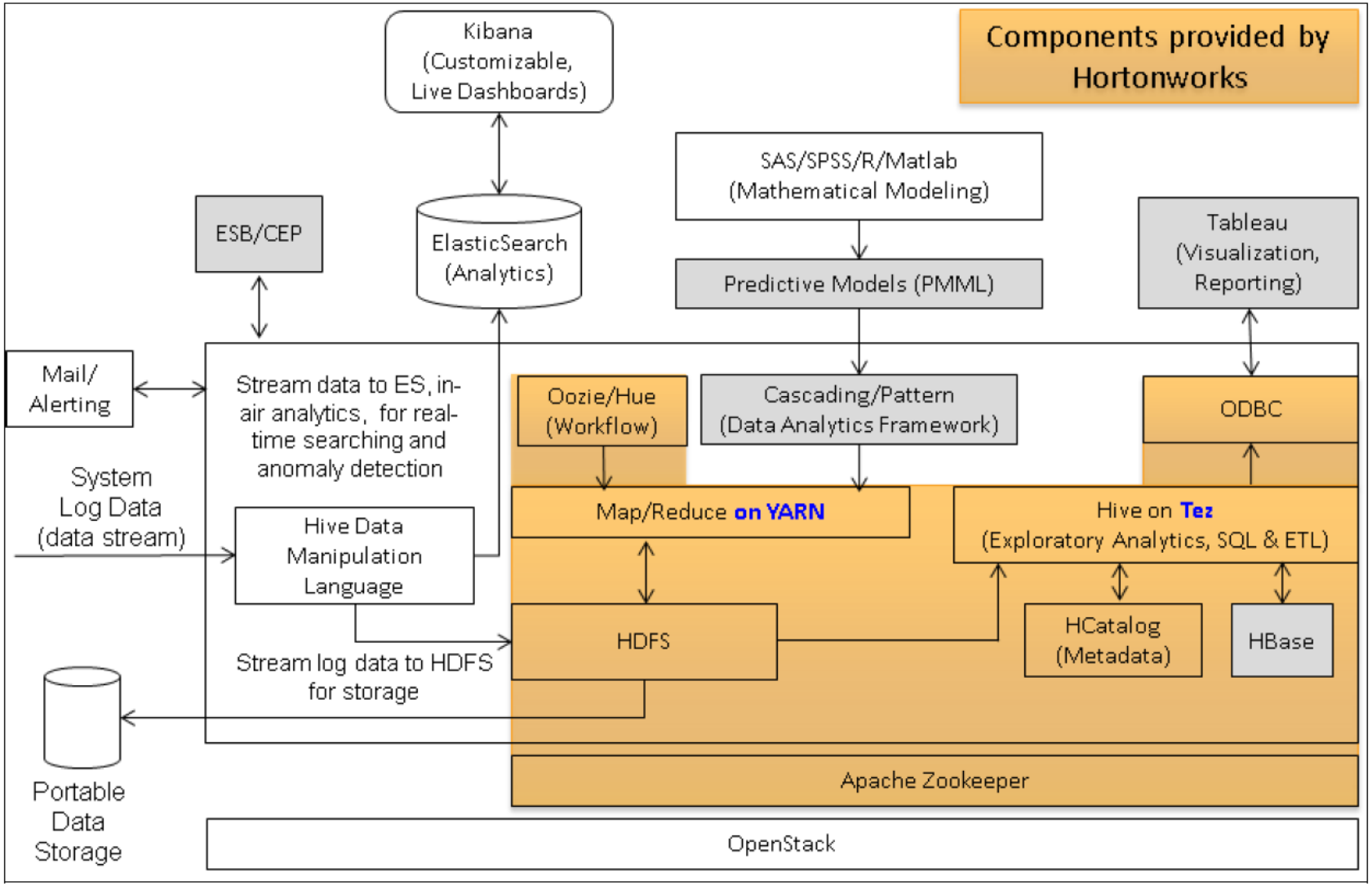
- Sources:
  - J. Albrecht. Big-Data-Technologien - Überblick. https://www.ihk-nuernberg.de/de/media/PDF/Innovation-Umwelt/e-commerce/big-data-technologien-ein-ueberblick.pdf
  - J. Albrecht. Processing Big Data with SQL on Hadoop. http://www.sigs.de/download/tdwi_2015_muc/files/t4a-2_Albrecht.pdf
  - D. McMurtry et al.: Data Access for Highly-Scalable Solutions: Using SQL, NoSQL, and Polyglot Persistence. Microsoft, 2013.

Hadoop Eco-System



Quelle: http://techblog.baghel.com/index.php?itemid=132

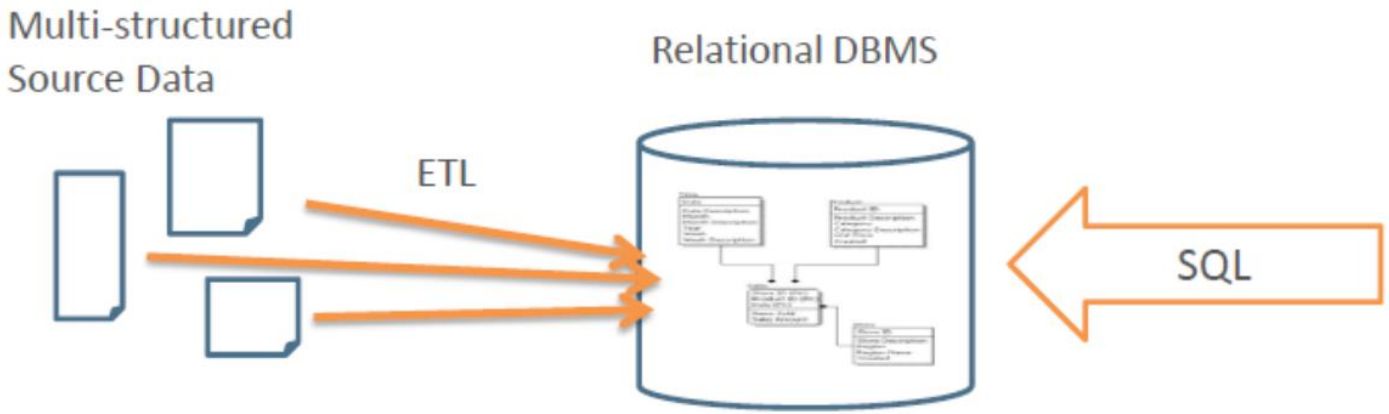
Example Big Data Architecture



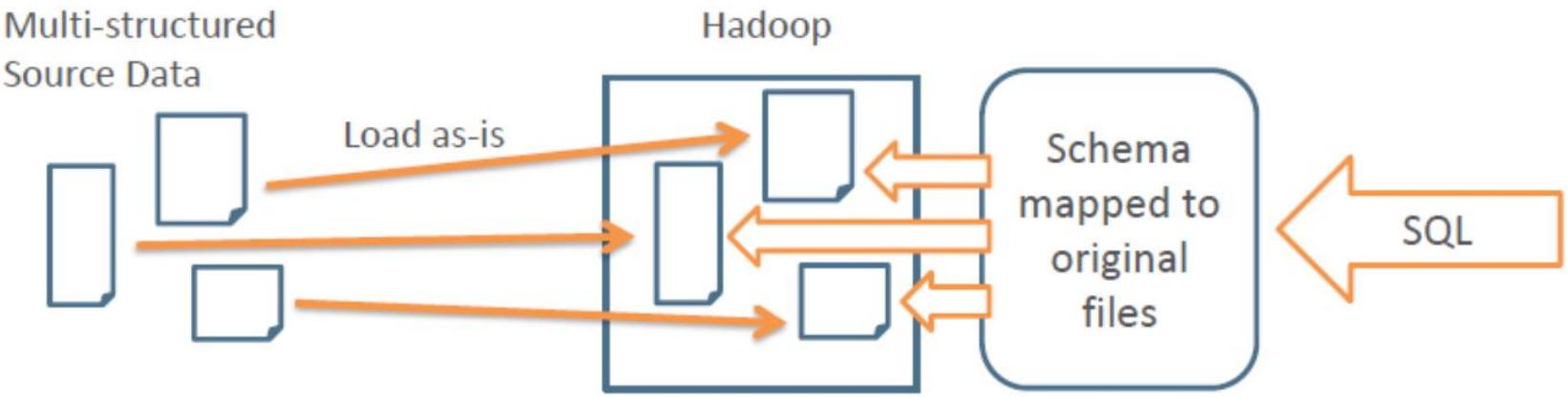
Boci, E. & Thistlethwaite, S.: A novel big data architecture in support of ADS-B data analytic
Proc. Integrated Communication, Navigation, and Surveillance Conference (ICNS), **2015**, C1-1-C1-8

Schema-on-Write vs. Schema-on-Read

- **Relational Database: Schema-on-Write**



- **Big Data Processing: Schema-on-Read**



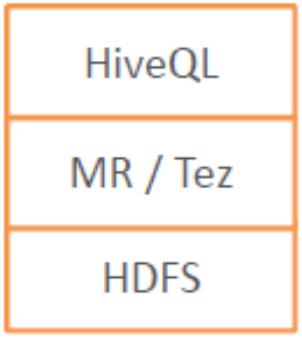
Why SQL on Hadoop?

- Mature technology
- Broad knowledge available
- Powerful query language
- High interactive performance
- Many third party tools for data analysis and visualization

- **Flexible data structures**
  - Semi-structured data
  - Changing schemas
- **Self-Service**
  - Data integration on-the-fly
- **Scalability**
  - Analysis, integration, volumen
- **(Relatively) Low Cost**
  - Commodity Hardware, Open Source

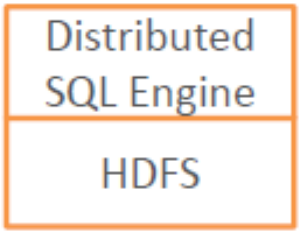
SQL on Hadoop

Hive (Native Hadoop)



Hortonworks Stinger

Pure Hadoop SQL Engines



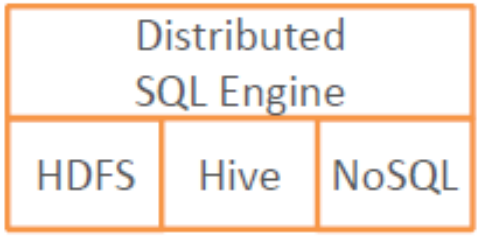
cloudera IMPALA

IBM Big Insights

EMC² HD HAWQ

Spark

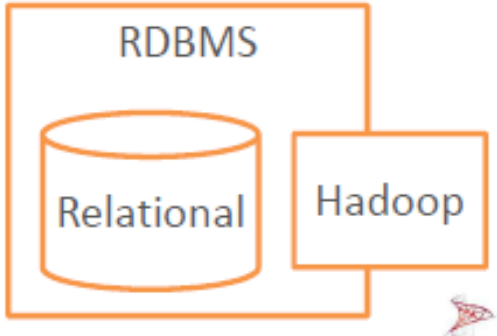
Format-agnostic SQL Engines



APACHE DRILL MAPR

presto facebook

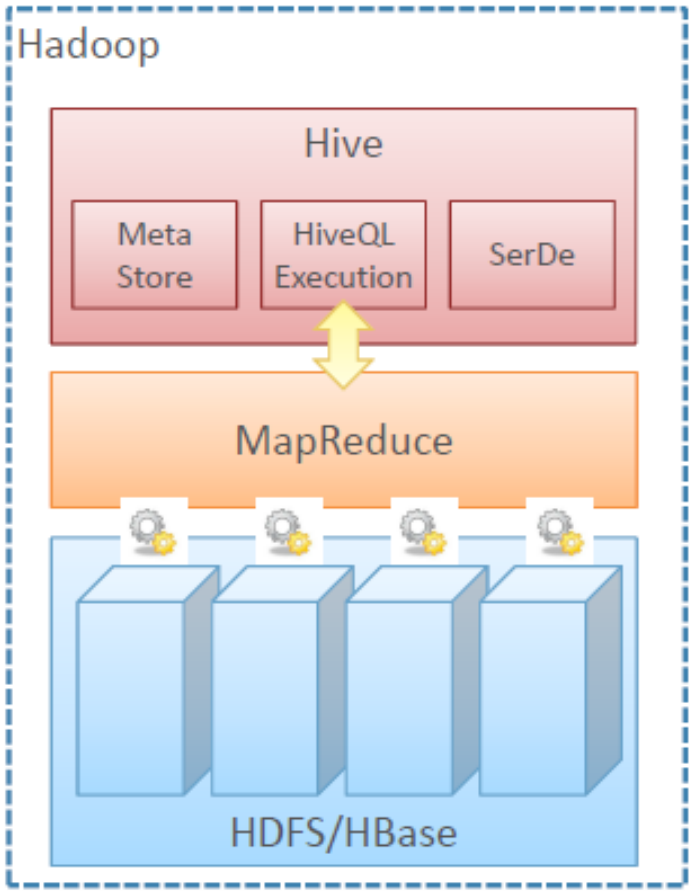
RDBMS with Hadoop Access



Microsoft SQL Server

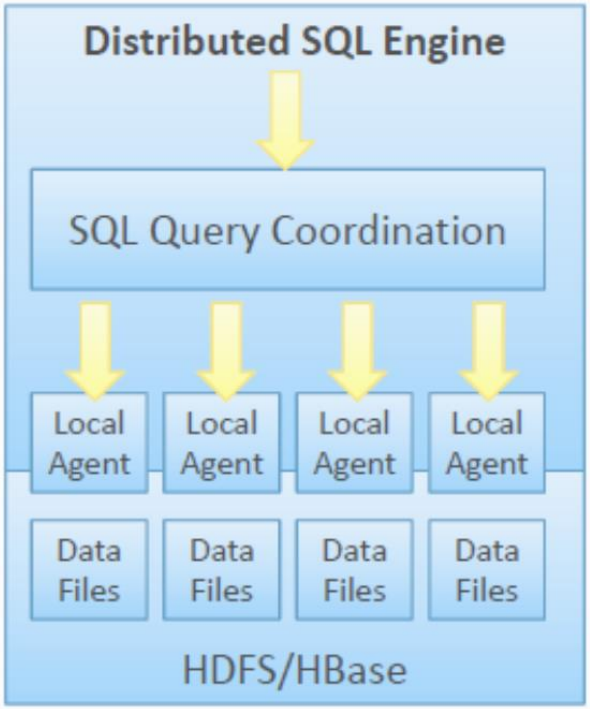
ORACLE

Hive



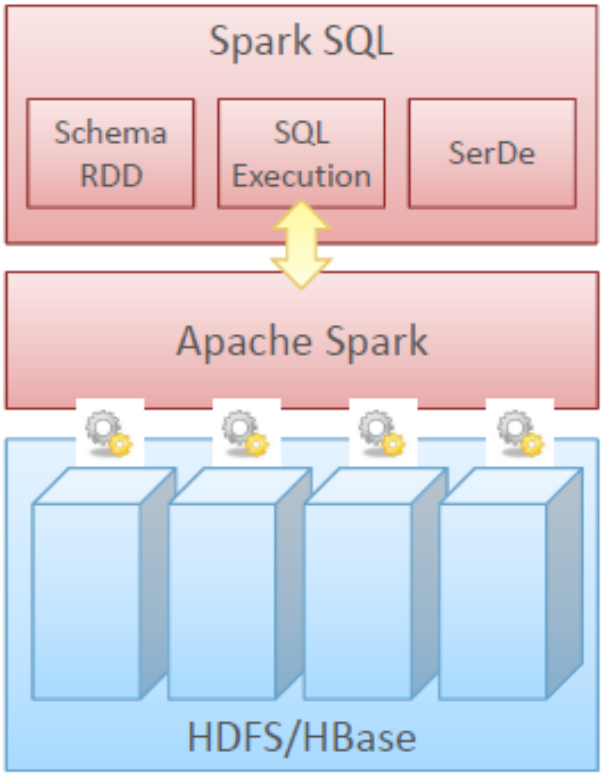
- **General**
  - Developed initially by Facebook
  - SQL-processing for HDFS and HBase
  - Table definitions in Hive Meta Store
  - Generation of MapReduce Code
  - **Schema-on-Read** via SerDe
- **Advantages**
  - Mature part of every Hadoop distribution
  - Simple setup
  - Java-API for UDFs
  - Usage of many data formats via SerDe
- **Disadvantages**
  - Batch-oriented, slow

Pure Hadoop SQL Engines



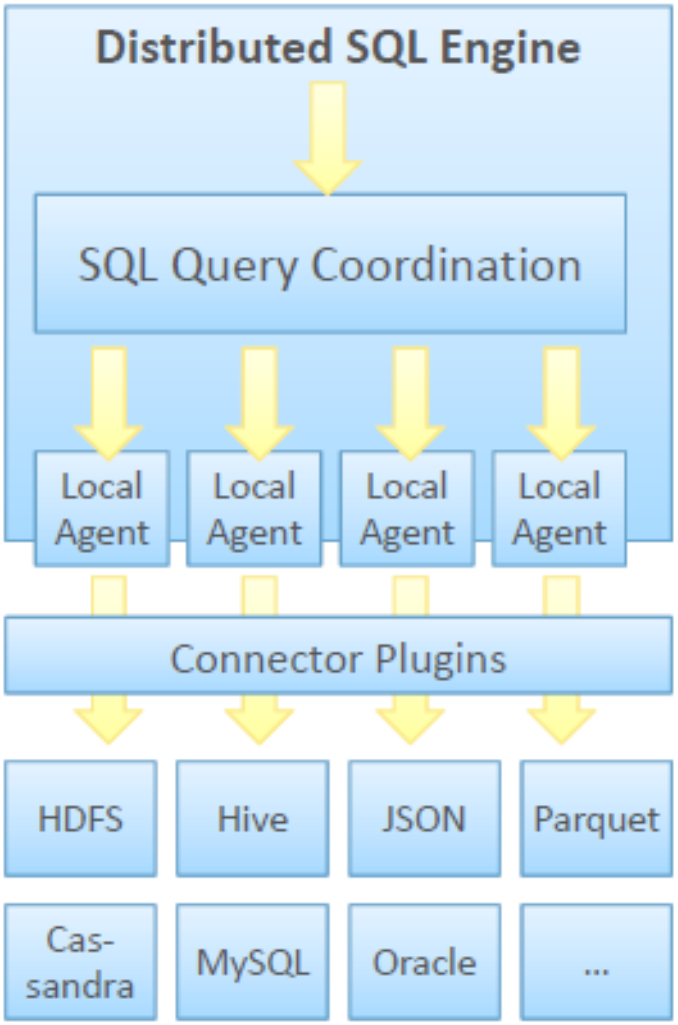
- **Approach**
  - Distributed, parallel SQL engine
  - Often usage of Hive Metadata
  - Support of optimized data formats
  - Hadoop as mandatory basis
- **Advantages and Disadvantages**
  - Significantly faster as Hive
  - Low latency through dedicated engine
  - Operator pipelining and result caching
- **Diffentiation of solutions**
  - Supported SQL functionality
  - Point querying
  - Cost-based optimizer / performance
  - Transaction support

Apache Spark & Spark SQL



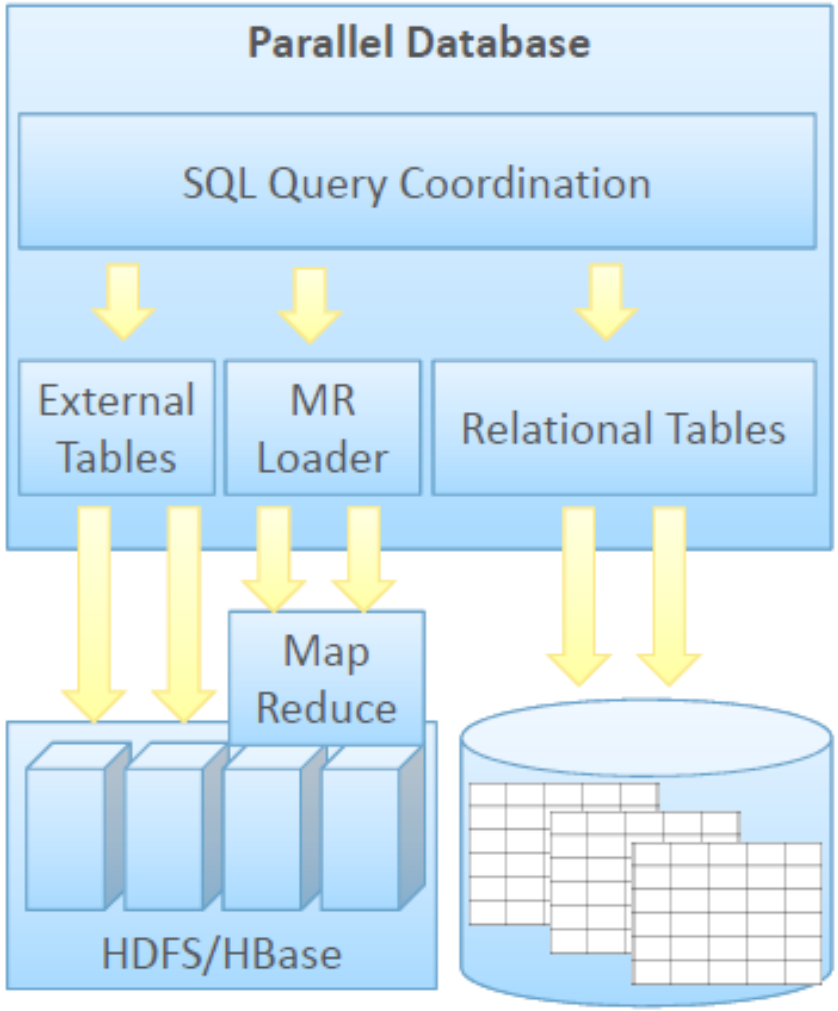
- **General**
  - SQL engine based on Spark
  - Data access via data frames (former SchemaRDD)
  - In-Memory columnar format
  - HDFS / HBase as file format
- **Advantages**
  - Spark as general-purpose parallel computing framework
  - Support of Hive extensions like UDFs and SerDes and Hive metadata
- **Disadvantages**
  - Not yet fully mature
  - Not yet as fast as competitors

SQL-Engine with Pluggable Storage



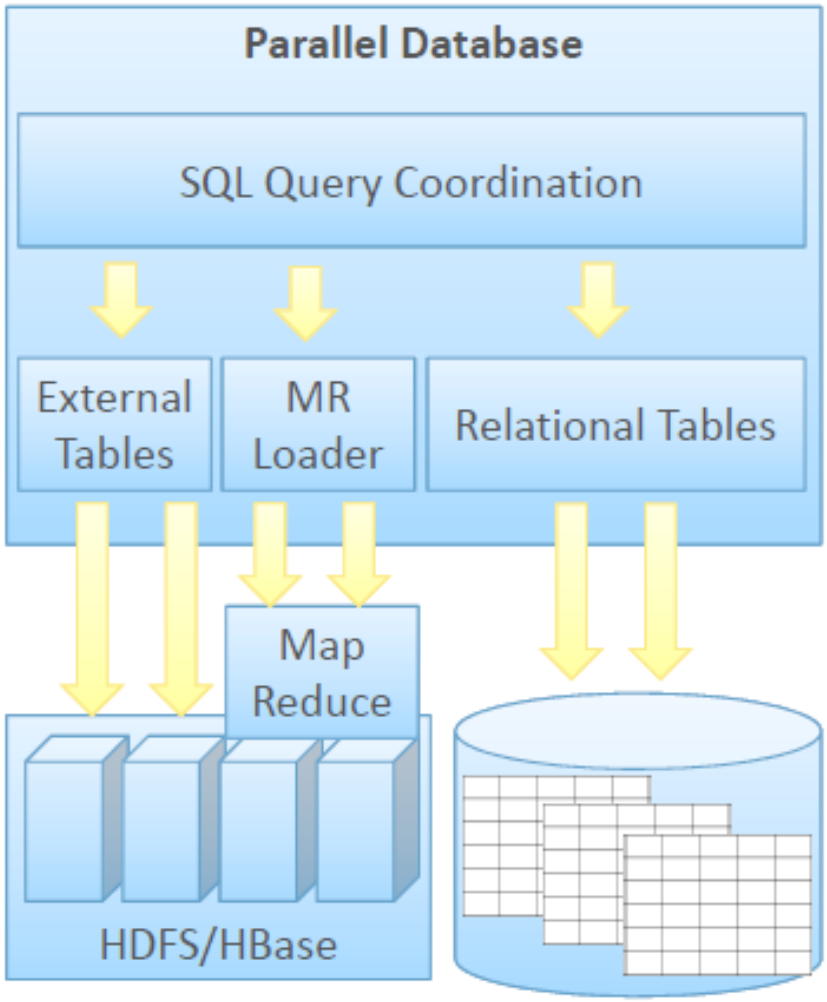
- Approach
  - Distributed, parallel SQL Engine
  - Often uses Hive Metadata
  - Support for optimized data formats
  - Hadoop obligatory as base system
- Advantages
  - Faster than Hive
  - Low latency as Map-Reduce is avoided
  - Pipeling & Caching
  - Scalability
- Disadvantages
  - Usually no transaction management

RDBMS with Hadoop Integration



- **Approach**
  - Towards genuine integration of Hadoop into RDBMS
  - Utilize Hadoop's computational power
  - Cost-based choice
- **Advantages**
  - Easiest way to use Hadoop as data source
  - Combined access to traditional and new data sources
- **Disadvantages**
  - Cost
  - Limited data sources
  - Vendor lock-in

RDBMS with Hadoop Integration



- **Products**
  - Microsoft Polybase (part of MS Analytics Platform)
  - Oracle Big Data SQL (part of Oracle Big Data Appliance in combination with Exadata)
- **Use Cases**
  - Extension of traditional BI System
  - Data-lake scenario with RDBMS as primary system and Hadoop for mass data
  - Mix of analytic and transactional load

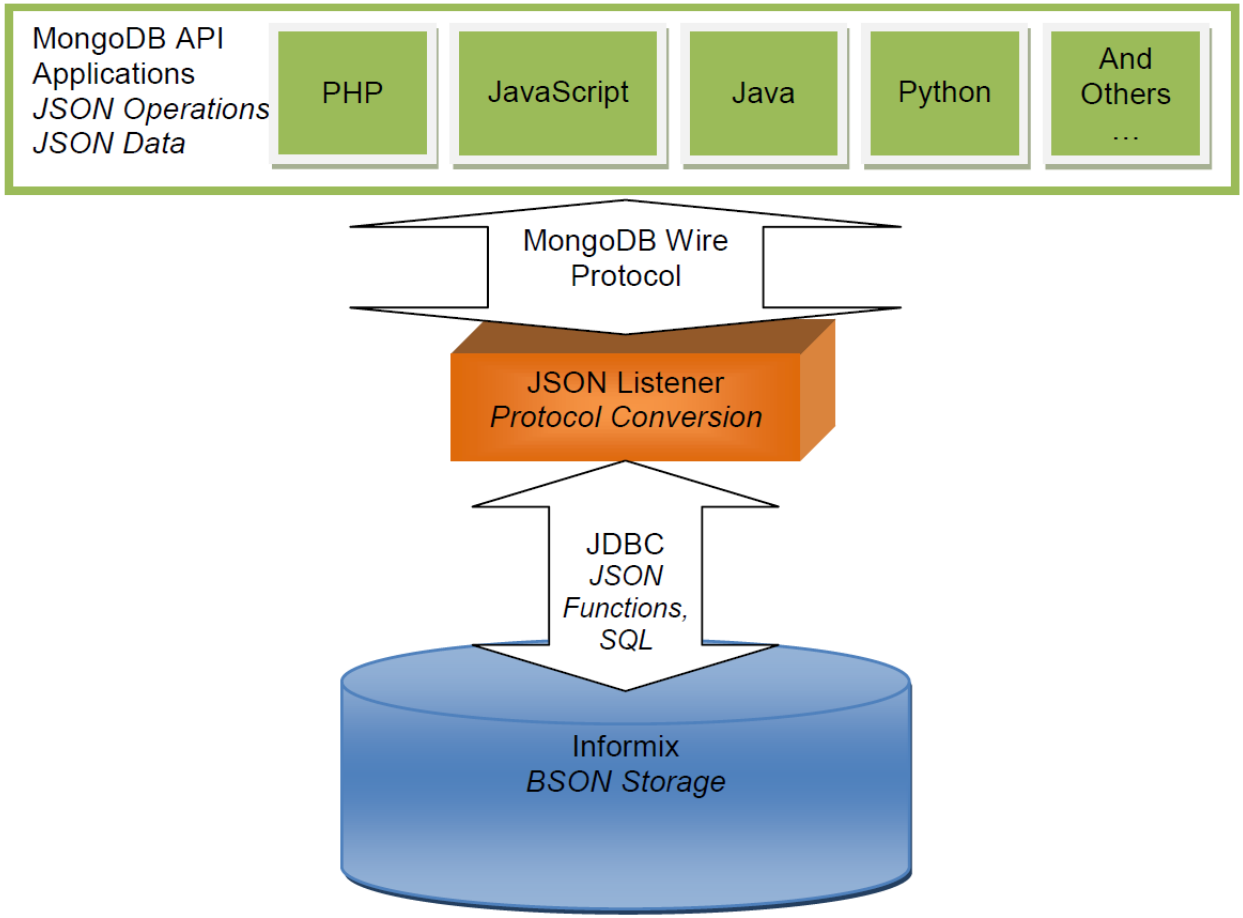
4.4 Big Data and NoSQL support in Classical DBMS

- Magic Quadrant
- IBM Informix
- Oracle NoSQL
- MS PolyBase/SQL Server/Azure

Gartner Magic Quadrant for Data Management Solutions for Analytics

IBM Informix

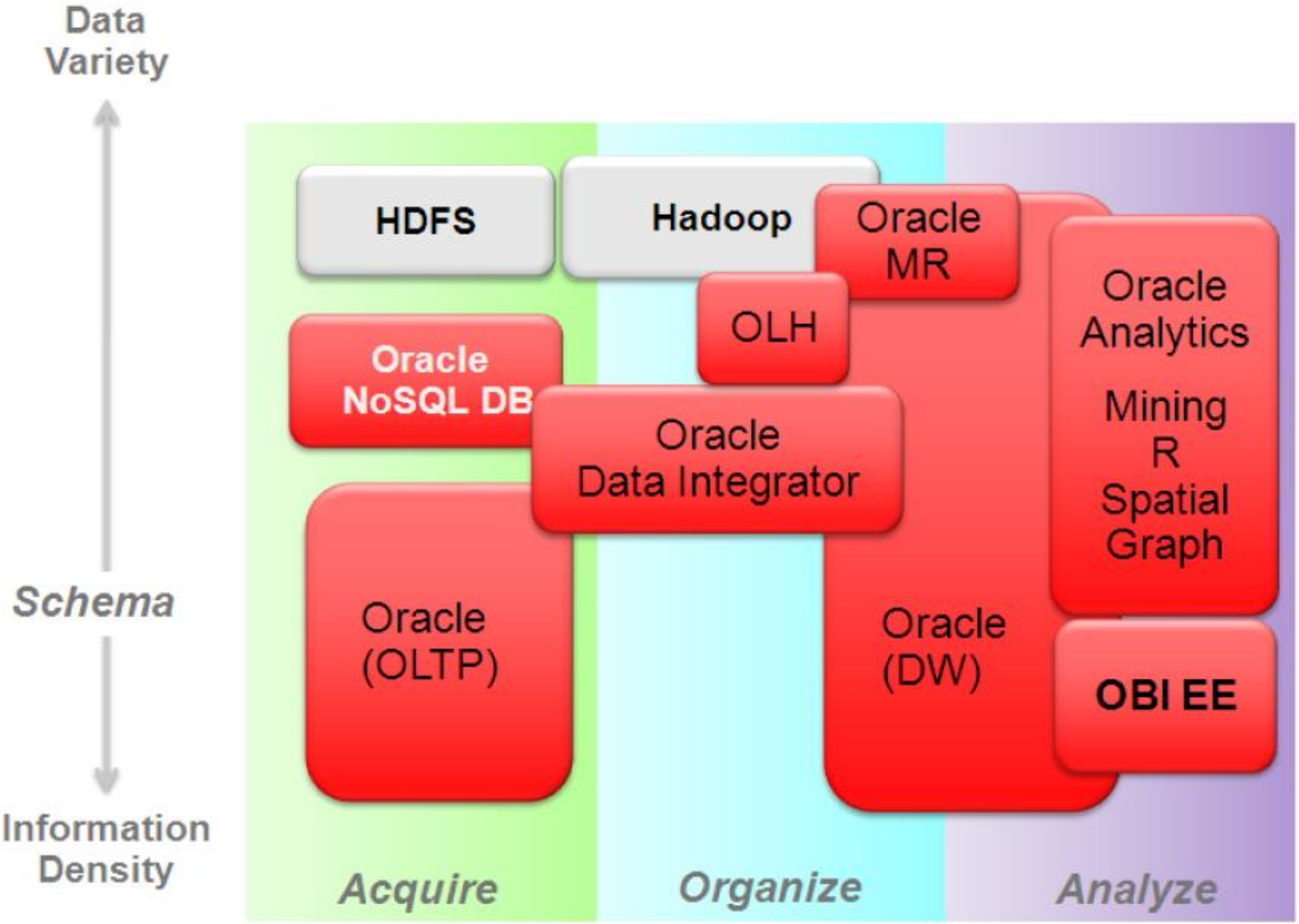
- Informix is an object-relational DBMS which implements JSON as datatype
- Provides an API similar to MongoDB



IBM Informix

- Hybrid system: Relational and non-relational data in one system
- Index support for JSON/BSON data types
- Row-level locking for JSON documents
- Native operators and comparator functions allow for direct manipulation of the BSON data type
- Sharding is also possible

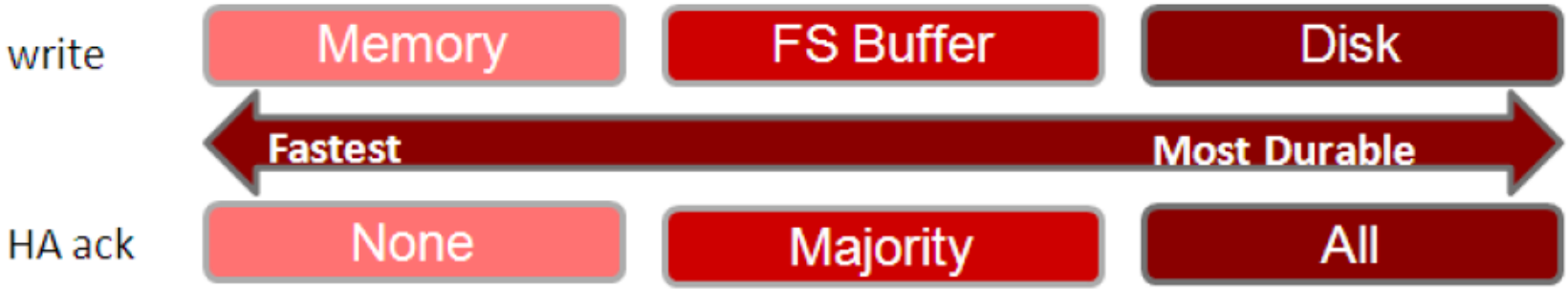
Oracle NoSQL
Overview of Data Management Ecosystem



Oracle NoSQL Policies

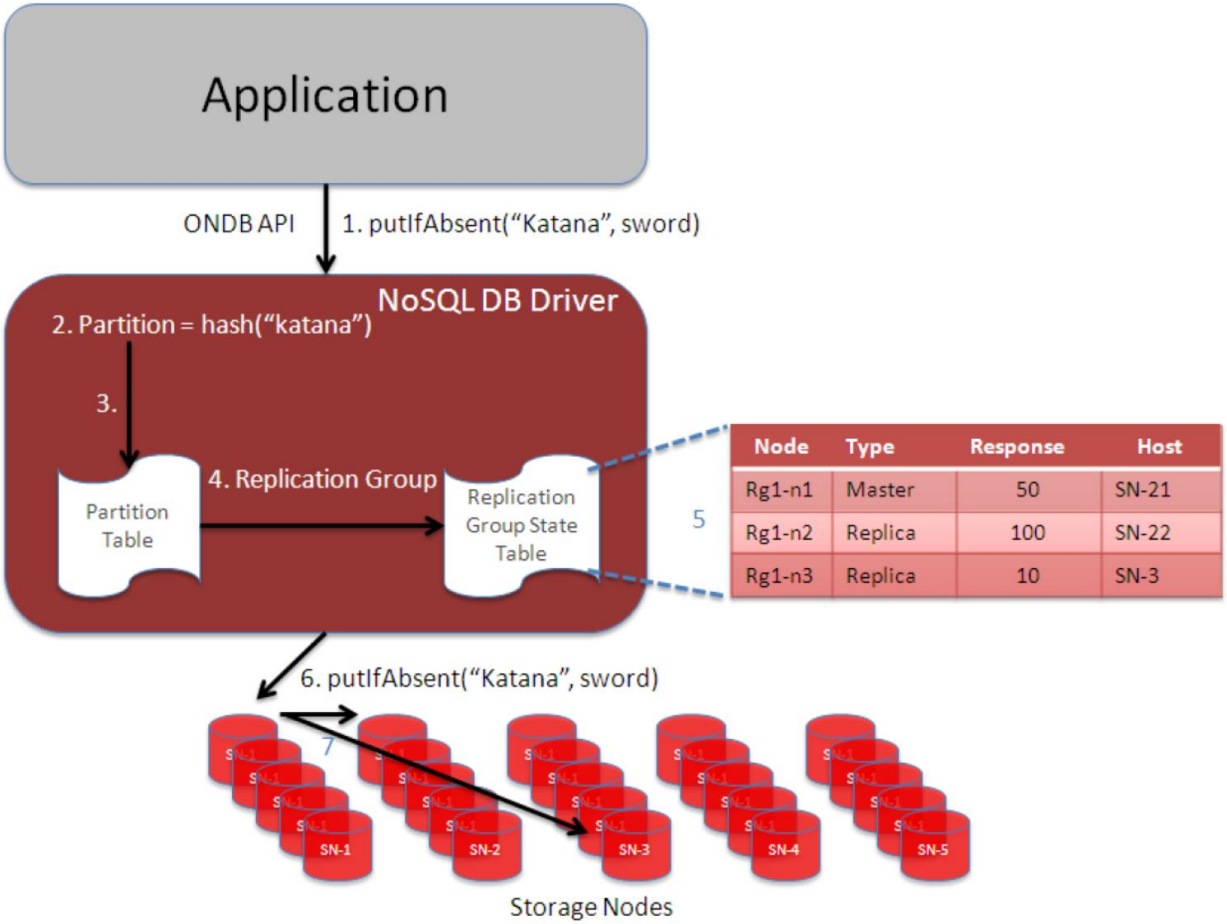
- Consistency Policies

- Durability Policies



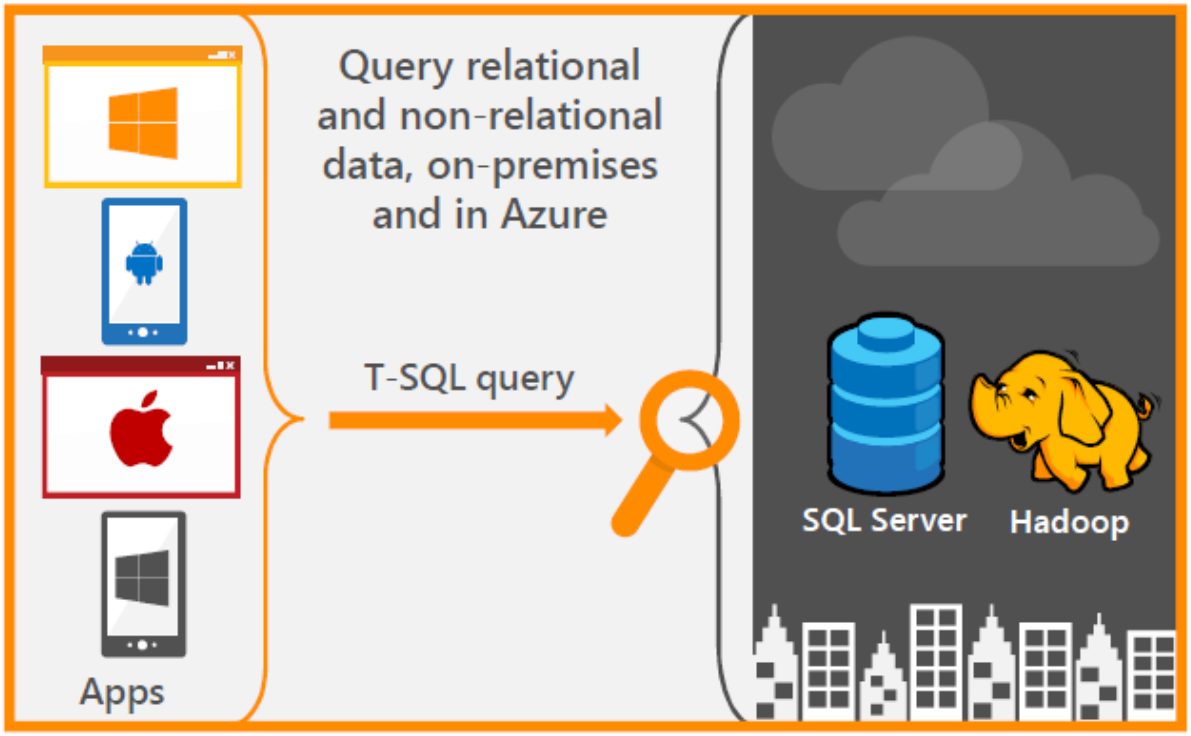
Oracle NoSQL API and Sharding

- API provides basic CRUD operations (create, read, update, and delete)
- Sharding is done by mapping keys to partitions which are mapped to shards



Microsoft PolyBase

Query relational and non-relational data with T-SQL



Capability

T-SQL for querying relational and non-relational data across SQL Server and Hadoop

Benefits

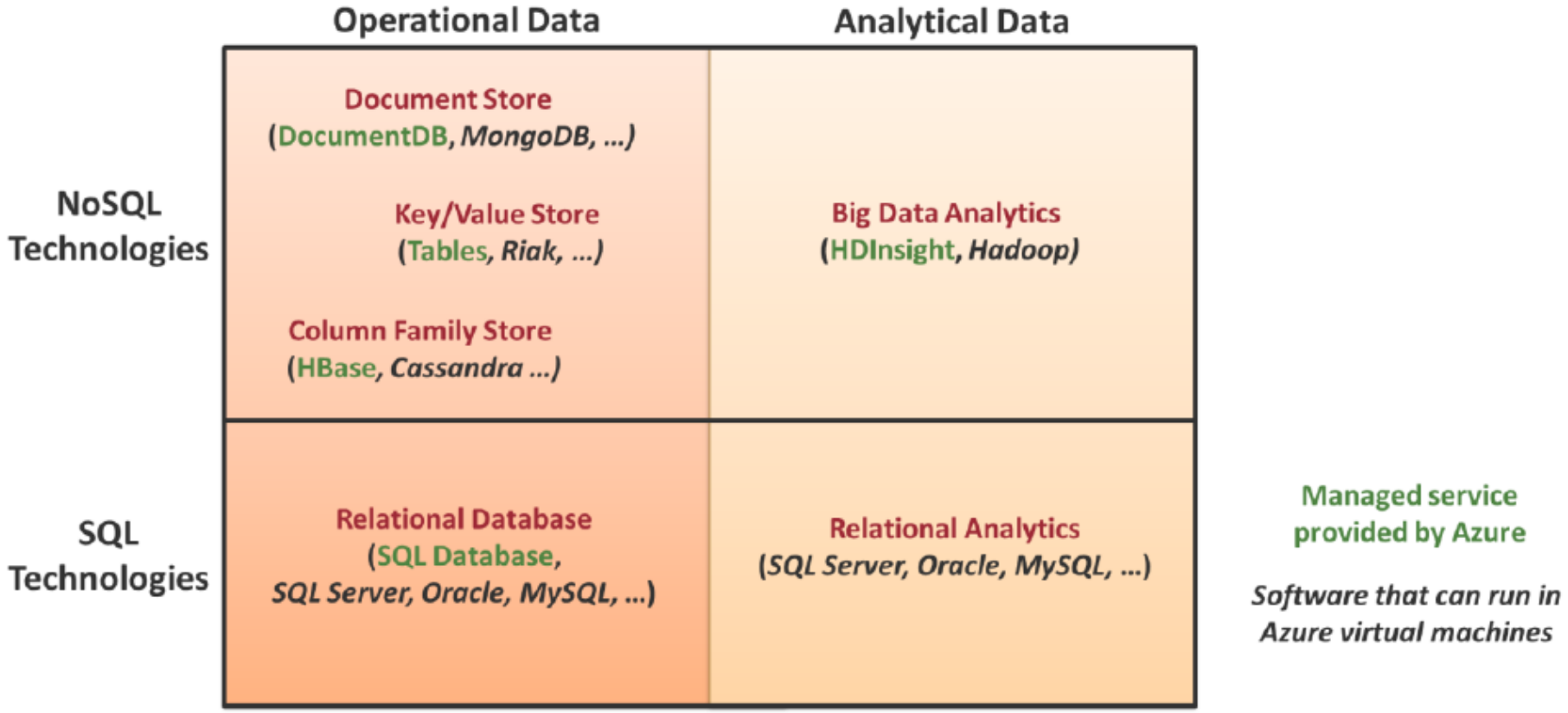
New business insights across your data lake

Leverage existing skillsets and BI tools

Faster time to insights and simplified ETL process

Data Stores on Azure

- Azure: Cloud platform offered by Microsoft
- Offers managed services (e.g., DocumentDB, Tables) for data storage, but can run any DBMS inside a VM



Summary

- Pervasive broadband Internet enables data management in the cloud
- Pay-as-you-go computing
- CAP-Theorem: Consistency, Availability, Partitioning
  ➔ Choose two out of three
- NoSQL DBMS offer new models for data management, featuring
  - Scalability
  - Distributed architectures
  - Schemaless data management
    (no a-priori schema definition)
  - New data models (graph, document, …)
- „NewSQL" DBMS providing scalable, distributed DBMS functionality and still use SQL and the relational model as a logical data model
- Classical RDBMS providers are catching up with new technologies
- Polyglot persistence
  - Data management today requires a heterogeneous set of data storage technologies, which are optimized for certain applications

Review Questions

- What are the limitations of RDBMS in Internet or Big Data applications?
- Explain the CAP theorem!
- What are the four basic data models for NoSQL DBMS?
- Explain the concepts „Sharding“ and „Eventual Consistency“!
- What is the role of NameNodes and DataNodes in Hadoop?
- How does MapReduce work?
- What are RDDs, partitions, and shuffling in Spark?
- Why is SQL still used also in Big Data applications?
- What is polyglot persistence?