NoSQL and Big Data

Lecture

Felix Gessert, Norbert Ritter

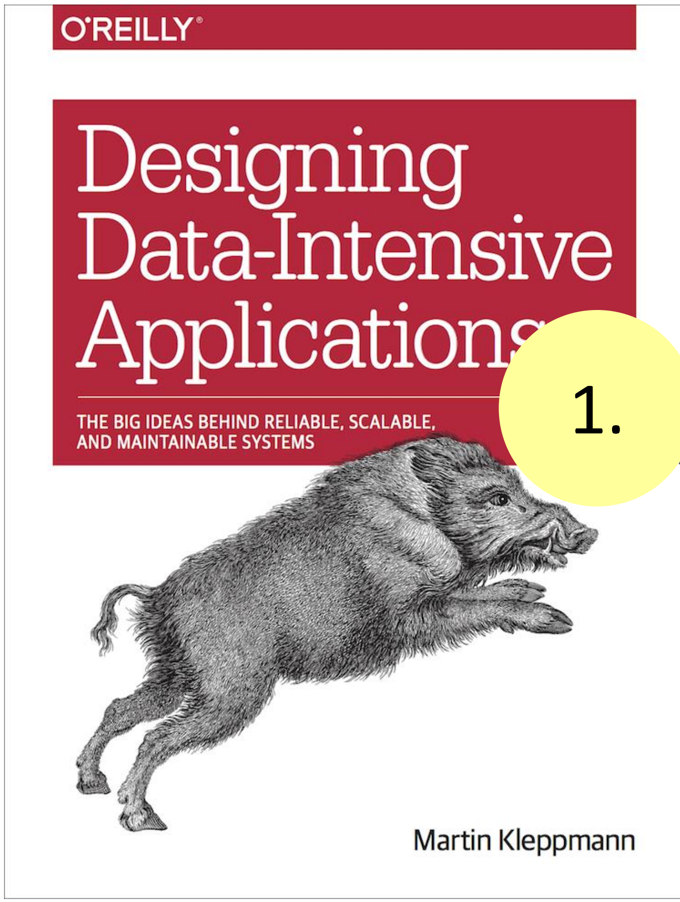
(22.05.2017)

Outline

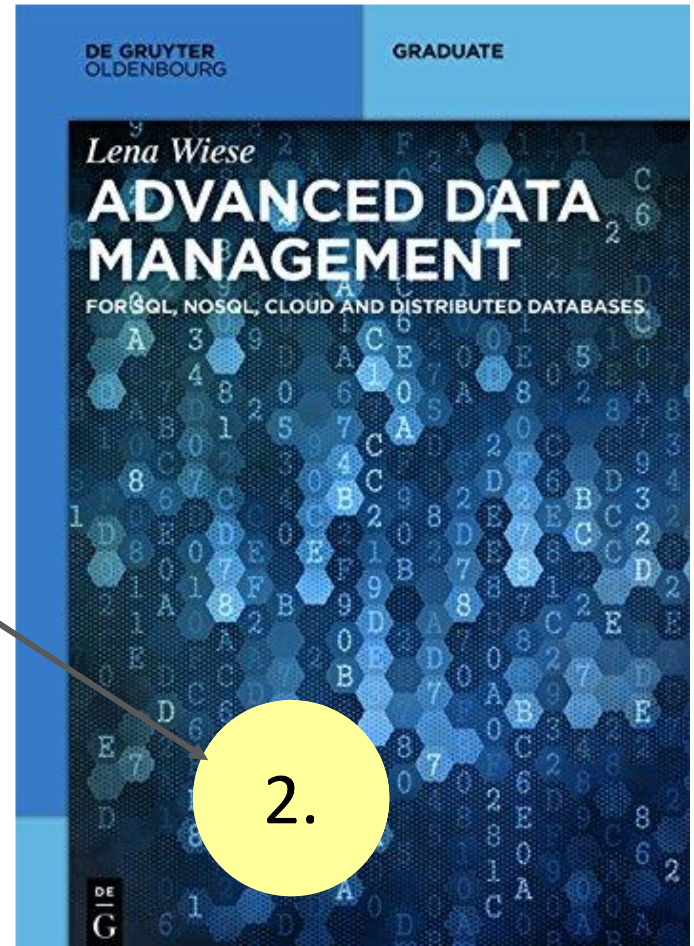
- Foundations: Big Data, Scalability, Avaialbility
- The 4 Classes of NoSQL Databases
- NoSQL Examples: concrete Architectures, Systems, APIs
- Cloud Databases

- Literature
- Motivation
  - Big Data
  - NoSQL
- CAP Theorem
- 2-Phase-Commit
- NoSQL Triangle
- ACID vs BASE

Recommended Literature: NoSQL

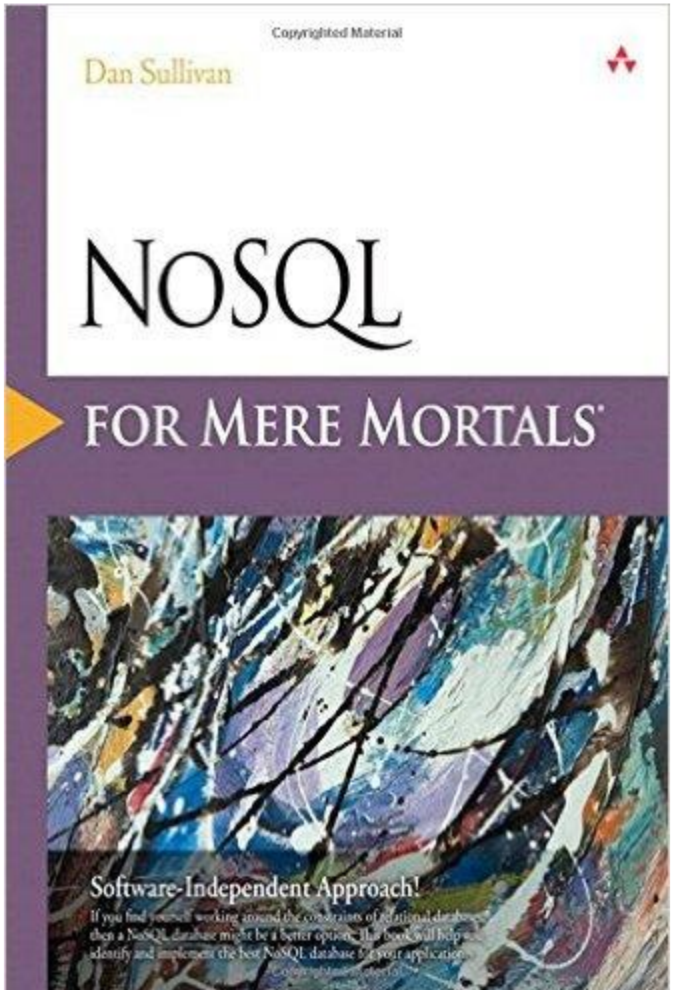


1.



2.

Other Literature



Recommended Literature: Blogs

Martin Kleppmann

https://martin.kleppmann.com/

NoSQL Weekly

http://www.nosqlweekly.com/

DZone

http://www.dzone.com/mz/nosql

BaQend

http://blog.baqend.com/

InfoQ

http://www.infoq.com/nosql/

High Scalability

http://highscalability.com/

The Database Explosion

Sweetspots

RDBMS

General-purpose ACID transactions

Parallel DWH

Aggregations/OLAP for massive data amounts

NewSQL

High throughput relational OLTP

Wide-Column Store

Long scans over structured data

Document Store

Deeply nested data models

Key-Value Store

Large-scale session storage

Graph Database

Graph algorithms & queries

In-Memory KV-Store

Counting & statistics

Wide-Column Store

Massive user-generated content

The Database Explosion

Cloud-Database Sweetspots

Firebase

Realtime BaaS

Communication and collaboration

Amazon RDS

Managed RDBMS

General-purpose ACID transactions

Amazon ElastiCache

Managed Cache

Caching and transient storage

Azure Tables

Wide-Column Store

Very large tables

Amazon DynamoDB

Wide-Column Store

Massive user-generated content

Parse

Backend-as-a-Service

Small Websites and Apps

bonsai

Managed NoSQL

Full-Text Search

Google Cloud Storage

Object Store

Massive File Storage

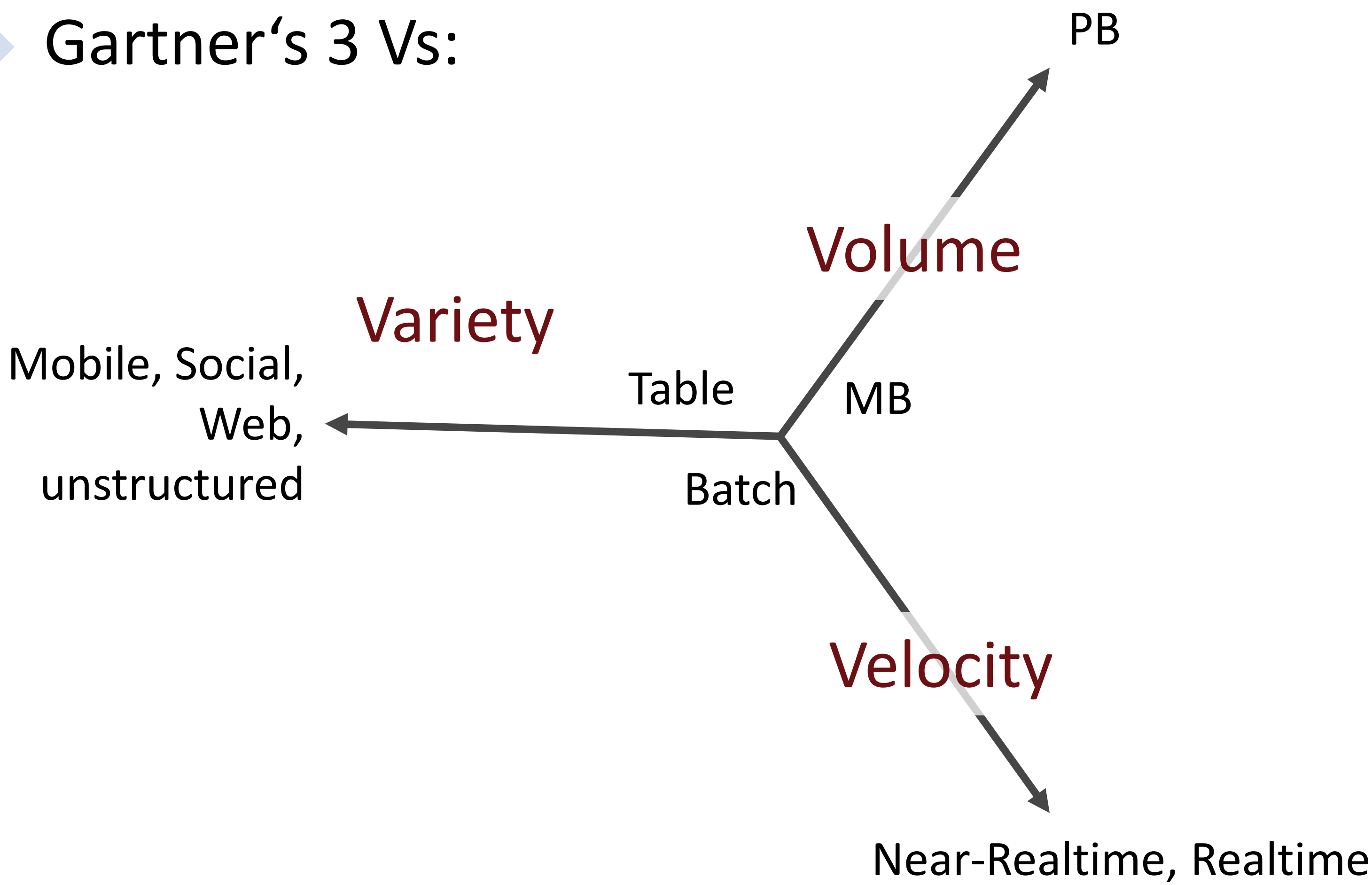
Amazon Elastic MapReduce

Hadoop-as-a-Service

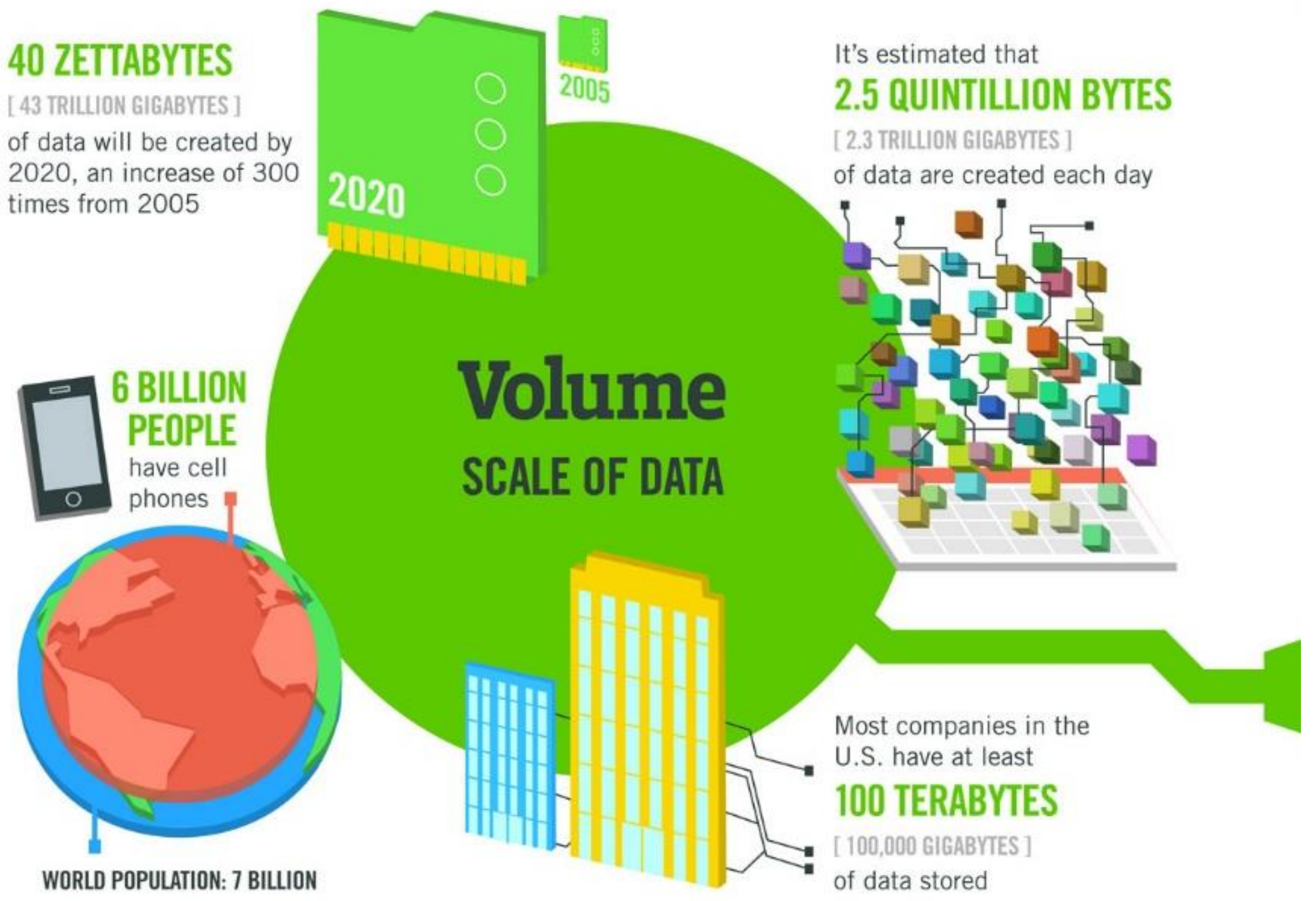
Big Data Analytics

Motivation Big Data

- Gartner‘s 3 Vs:

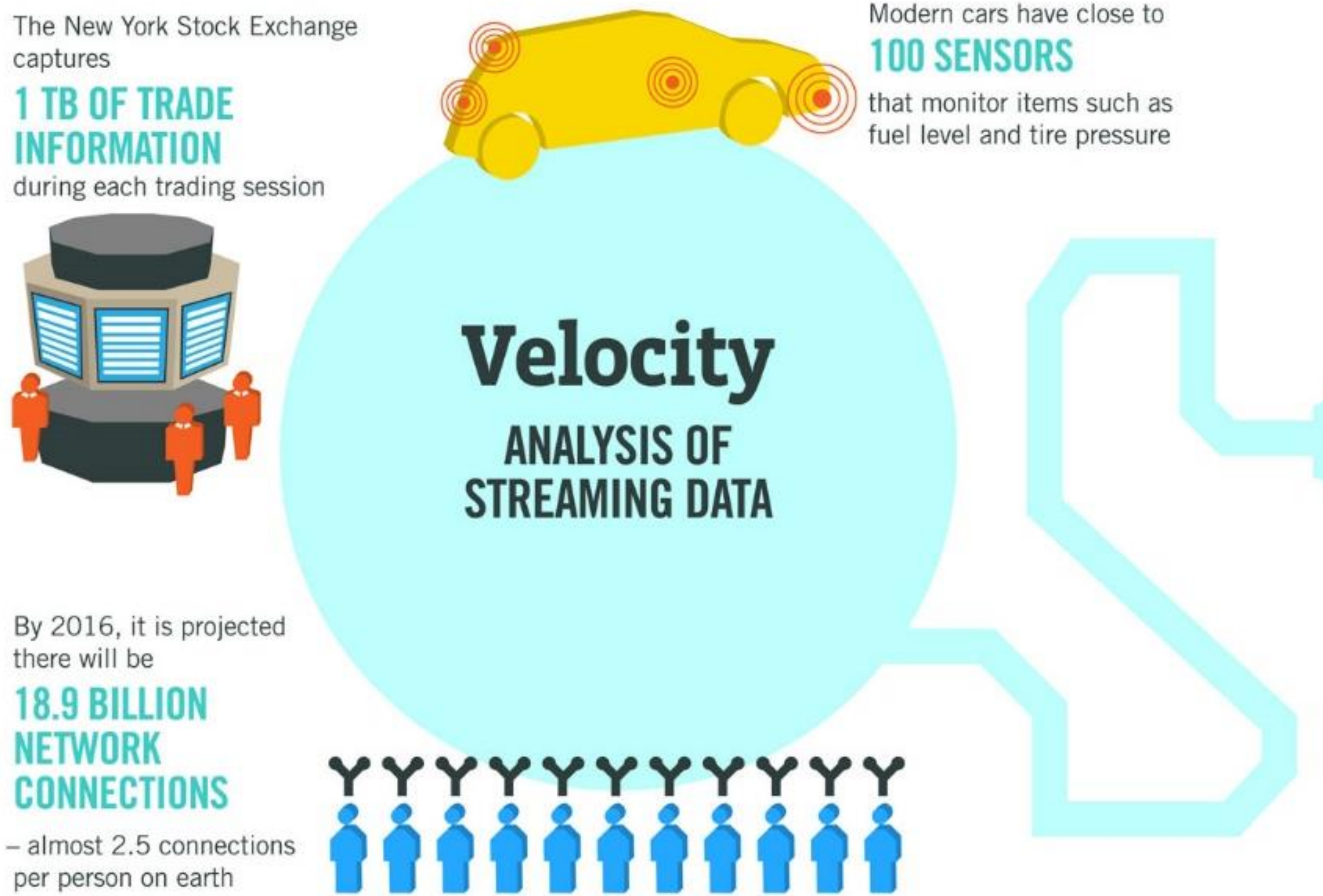


4Vs



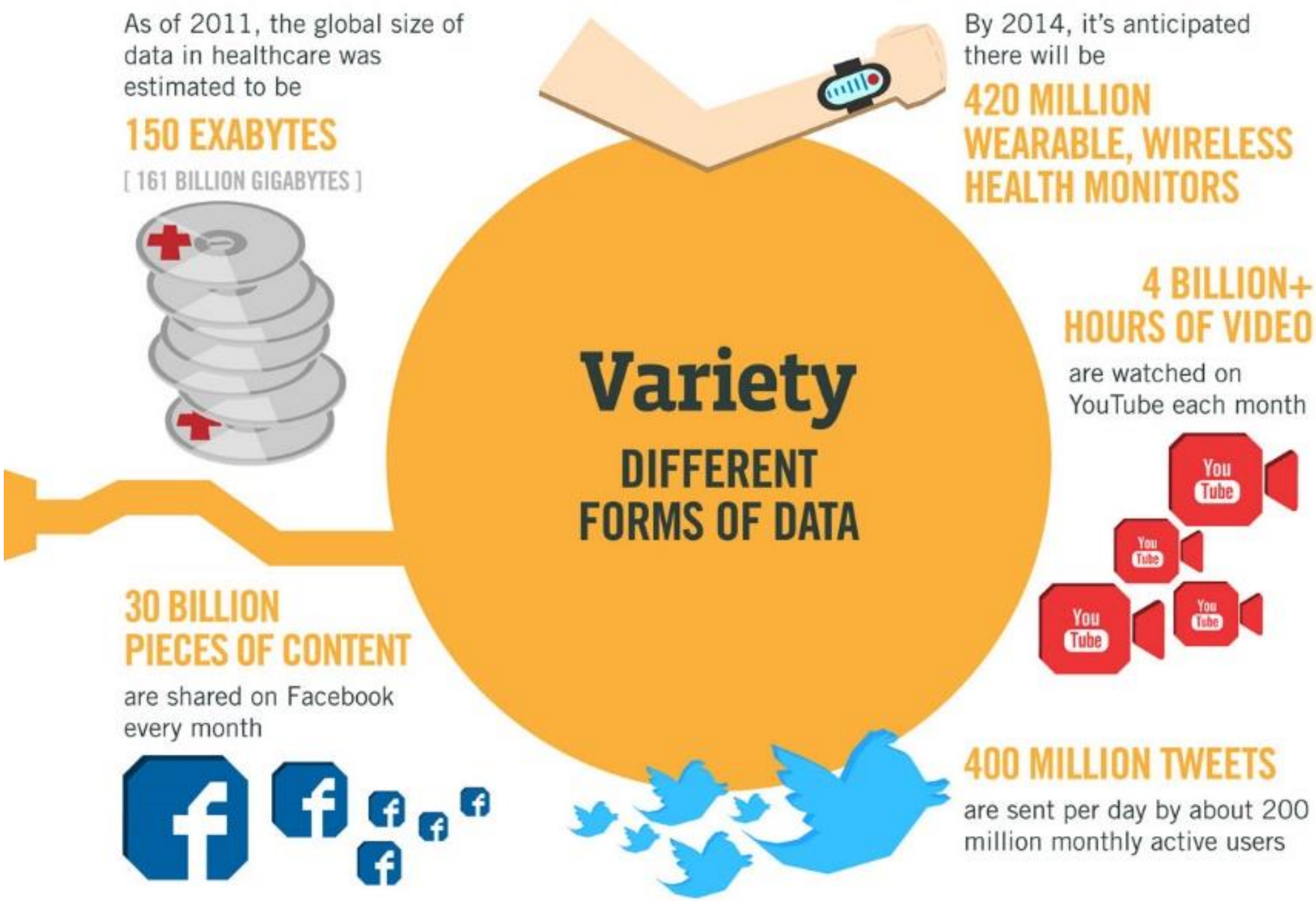
IBM Infographic (McKinsey, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS)

4Vs



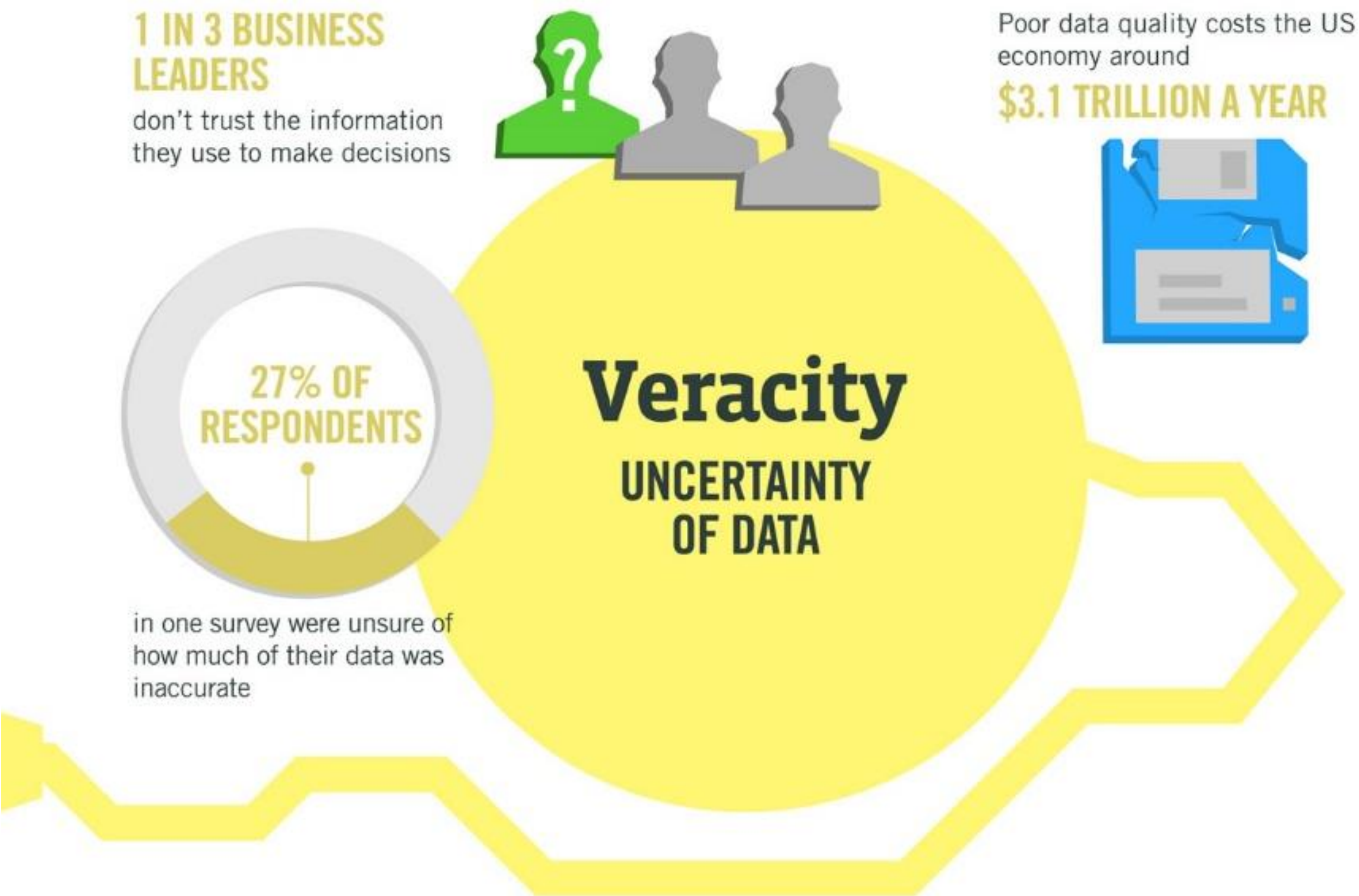
IBM Infographic (McKinsey, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS)

4Vs



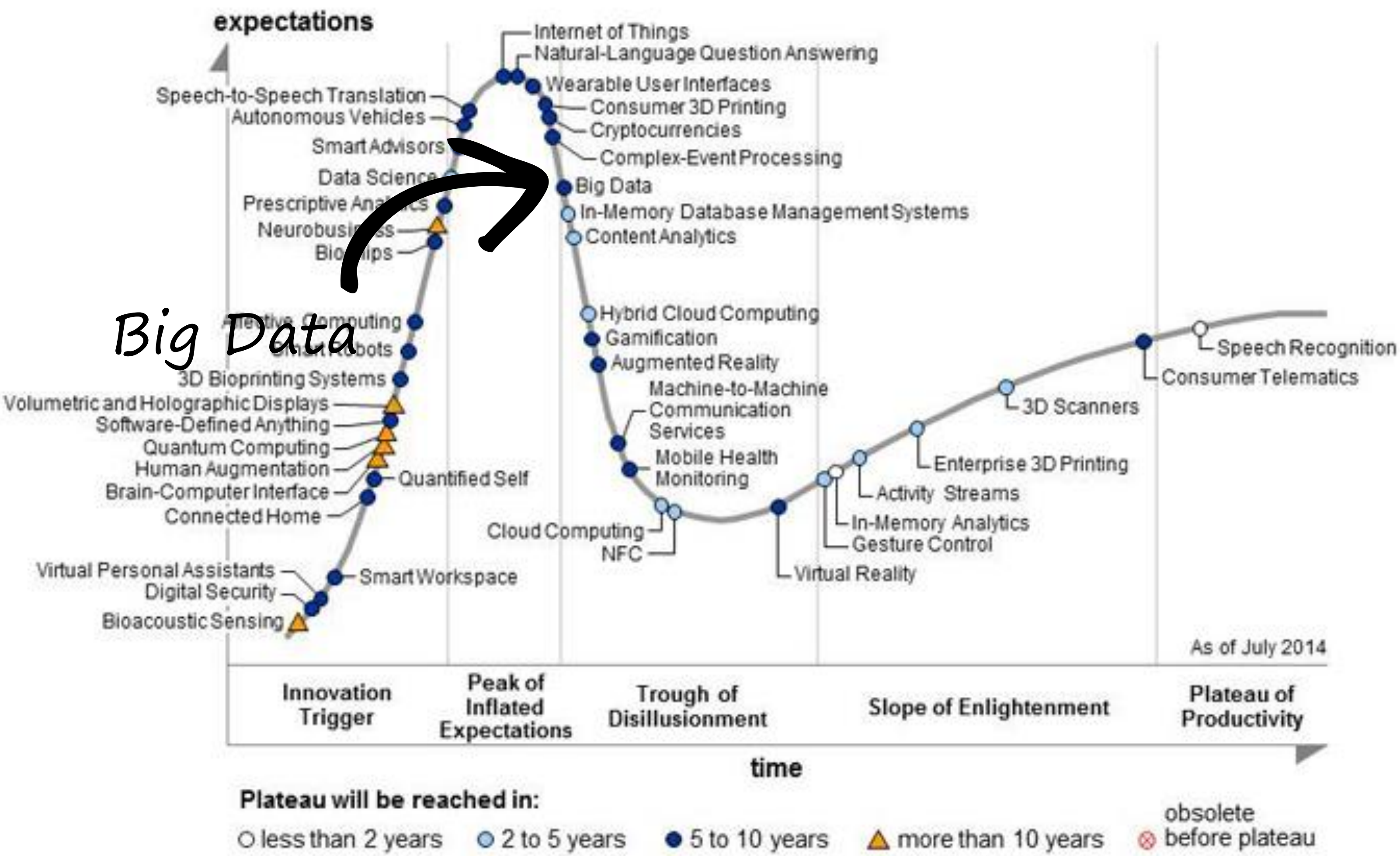
IBM Infographic (McKinsey, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS)

4Vs

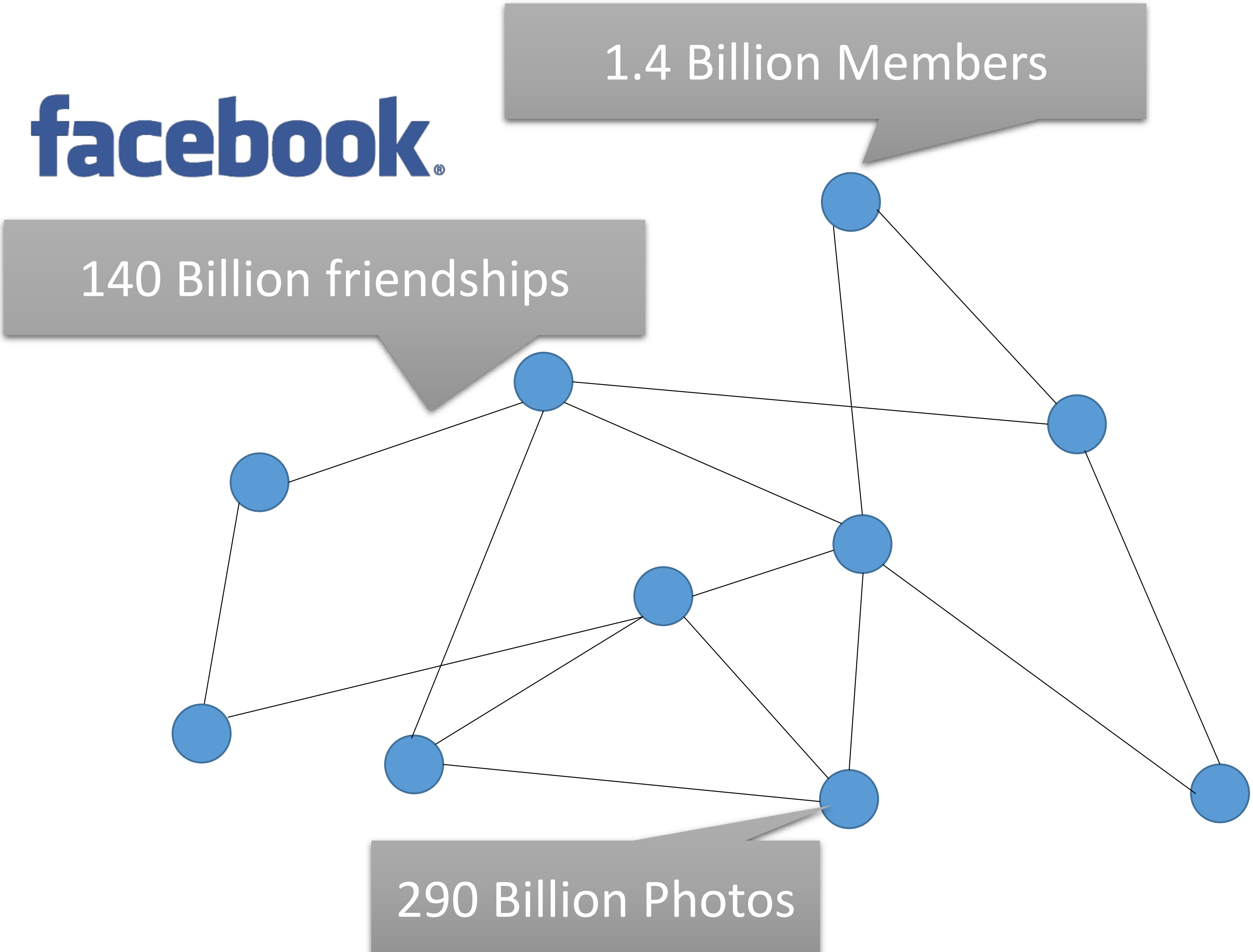


IBM Infographic (McKinsey, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS)

Hype Cycle



http://www.gartner.com/newsroom/id/2819918 facebook


1.4 Billion Members
140 Billion friendships
290 Billion Photos

Data flood

Climate Research: The *Deutsche Klimarechenzentrum (DKRZ)* stores **60 PB** climate data.

Archiving: The Internet Archive stores **10 PB** of archived websites.

Gaming: World of Warcraft needs **1.3 PB** for storing the game state. Steam delivers **over 30 PB** of data per month.

Movies: The CGI-effects in Avatar (2009) needed **over 1 PB** storage for rendering.

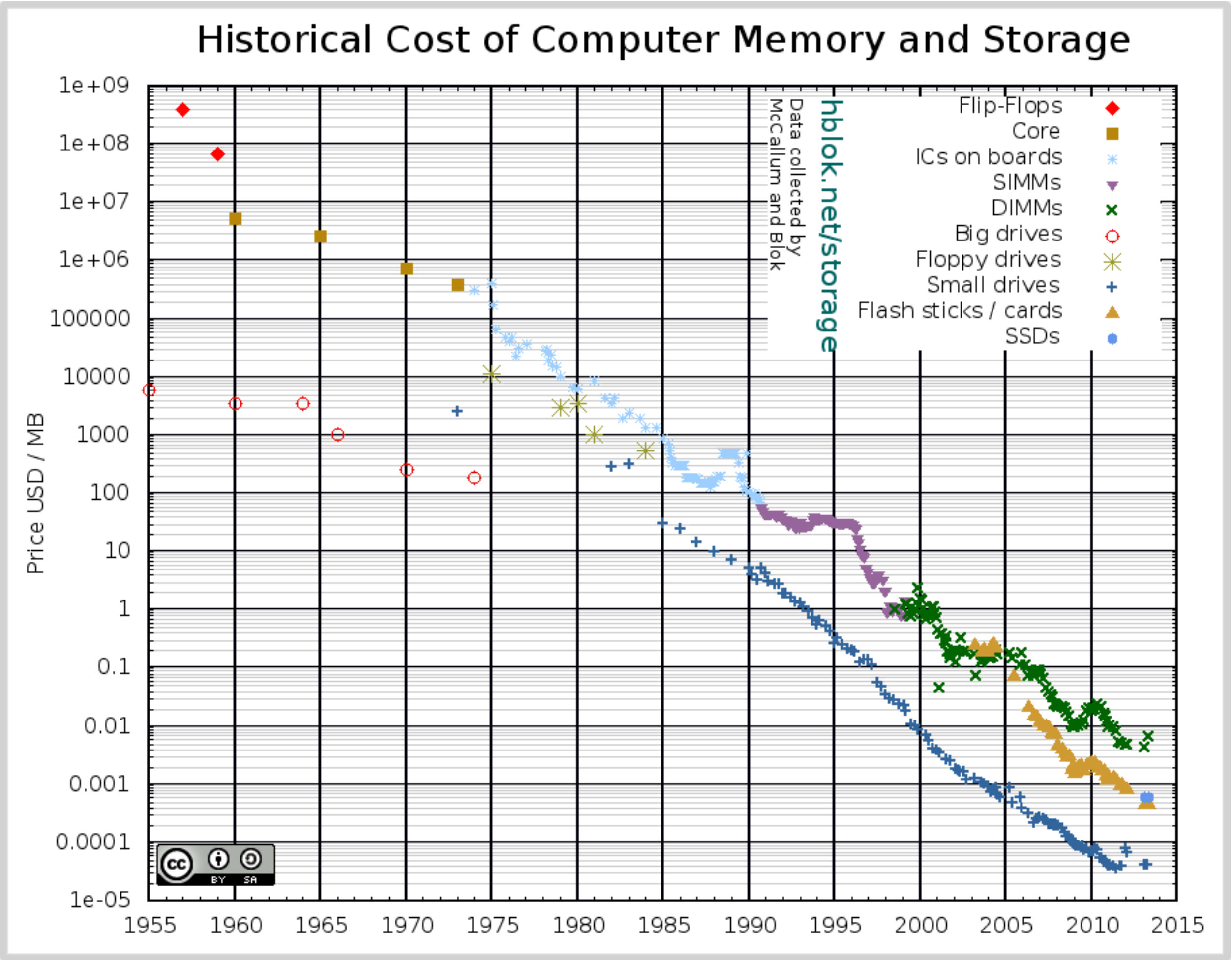
Supercomputing: The Blue Waters Supercomputer is planned to have a storage capacity of **500 PB**.

Particle Physics: In search of the Higgs-Boson CERN gathered **200 PB** of data.

Email: In May 2013 Microsoft announced that for the migration of Hotmail to oulook.com **over 150 PB** user data were transferred.

Wikipedia

Dropping storage costs



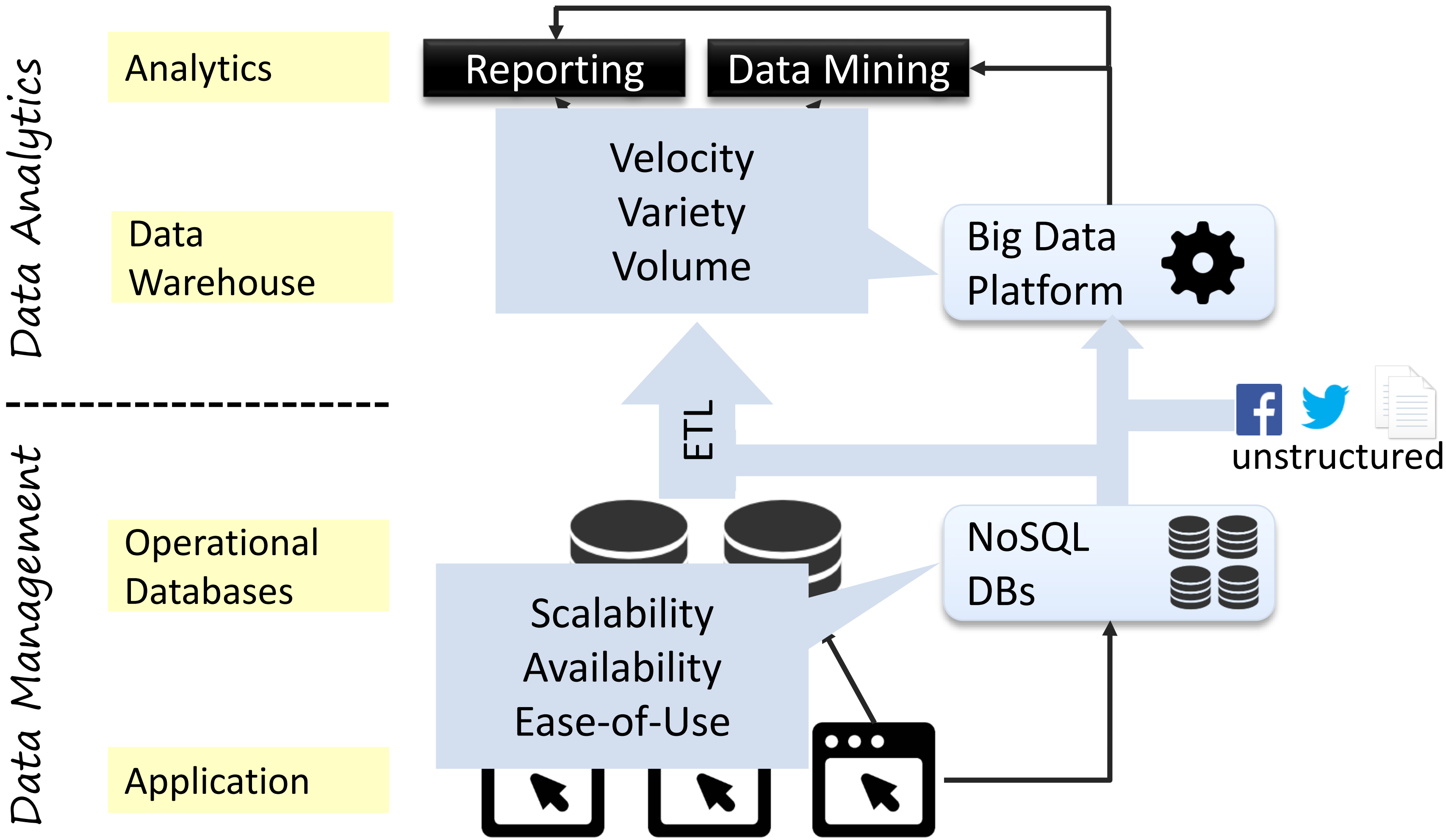
Big Data Defined

- Big Data has two sides:

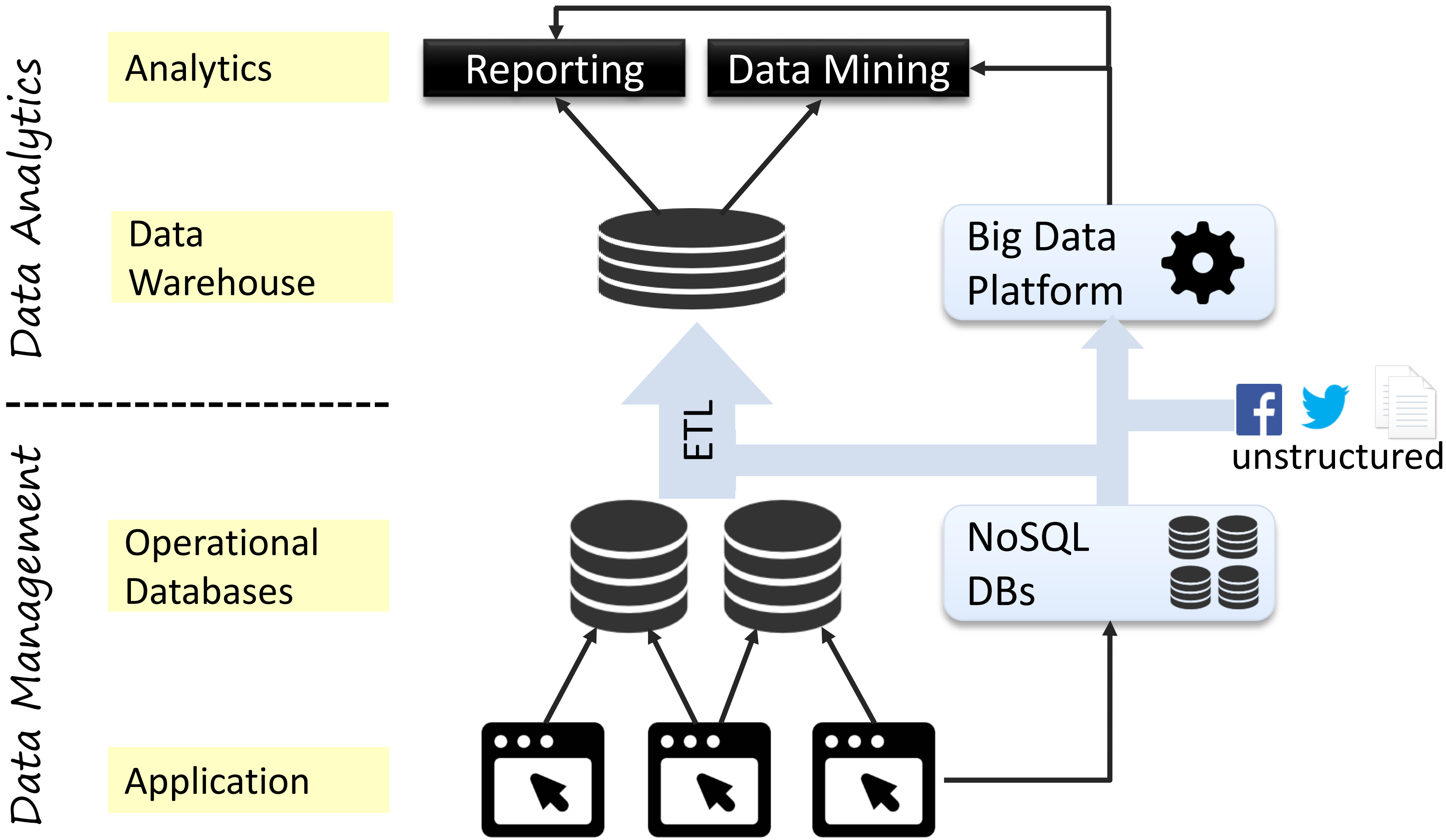
Big Data

Big Data Management	Big Data Analytics
• **OLTP** • Often referred to as "NoSQL" • e.g. MongoDB, HBase, Cassandra	• **OLAP** • Often referred to as „Big Data“ • e.g. Hadoop, Storm

Architectural Change



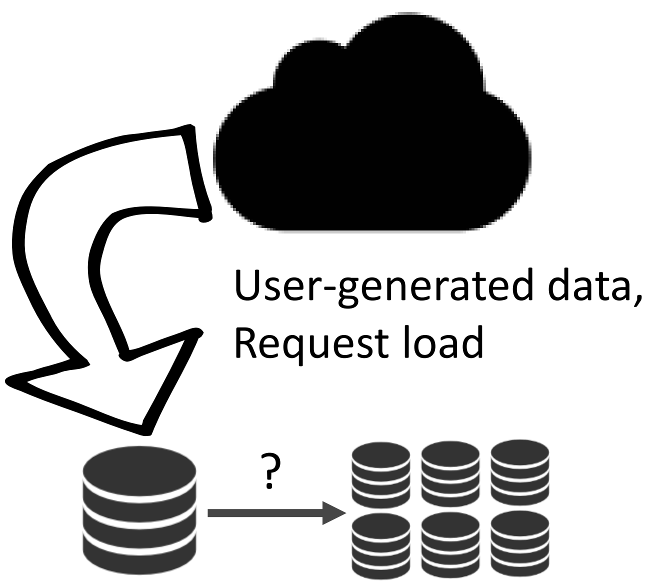
Architectural Change

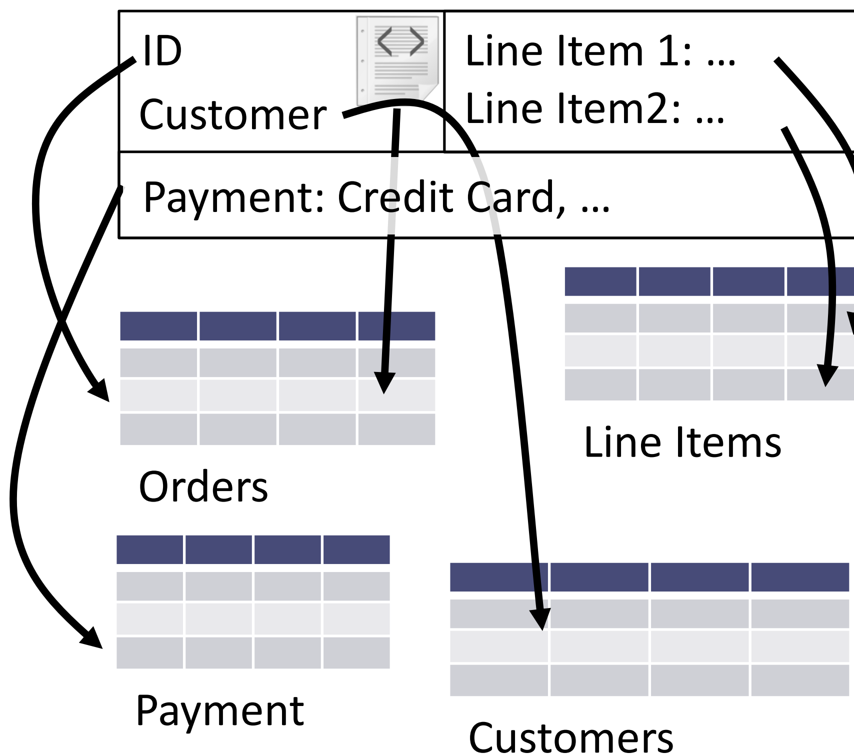


NoSQL Databases

- Two main motivations:

Scalability	Impedance Mismatch





NoSQL Databases

- „NoSQL“ term coined in 2009
- Interpretation: „**N**ot **O**nly **SQL**“
- Typical properties:
  - Non-relational
  - Open-Source
  - Schema-less (*schema-free*)
  - Optimized for distribution (clusters)
  - Tunable consistency

***NoSQL-Databases.org*:**
Current list has over 150
NoSQL systems

Big Data Analytics

- **Idea**: make existing massive, unstructured data amounts usable

sources

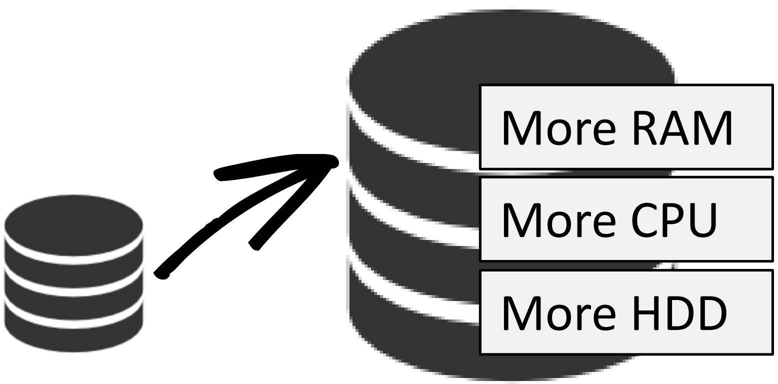
- Structured data (DBs)
- Log files
- Documents, Texts, Tables
- Images, Videos
- Sensor data
- Social Media, Data Services

Analyst, Data Scientist,
Software Developer

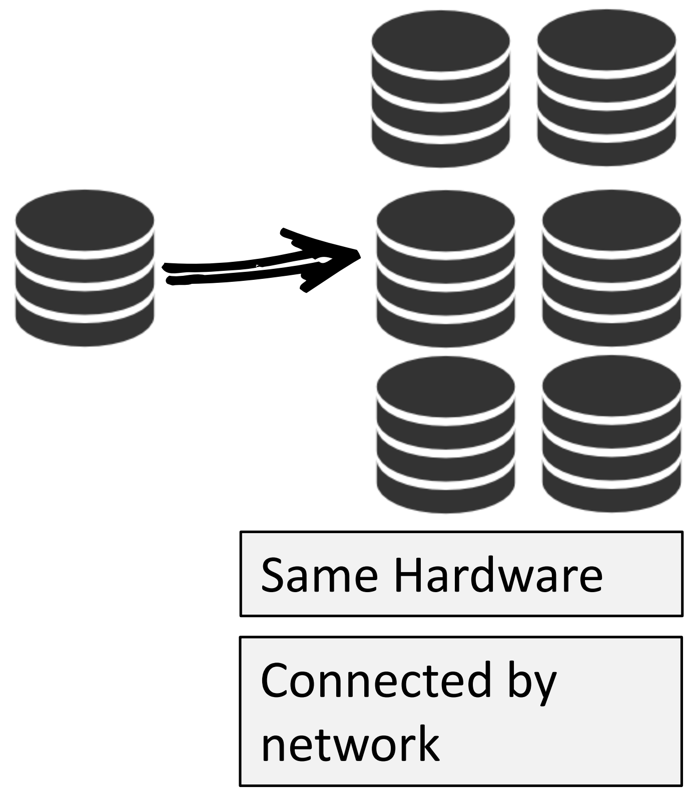
- Statistics, Cubes, Reports
- Recommender
- Classificators, Clustering
- Knowledge

Scale-up vs Scale-out

Scale-Up (*vertical* scaling):

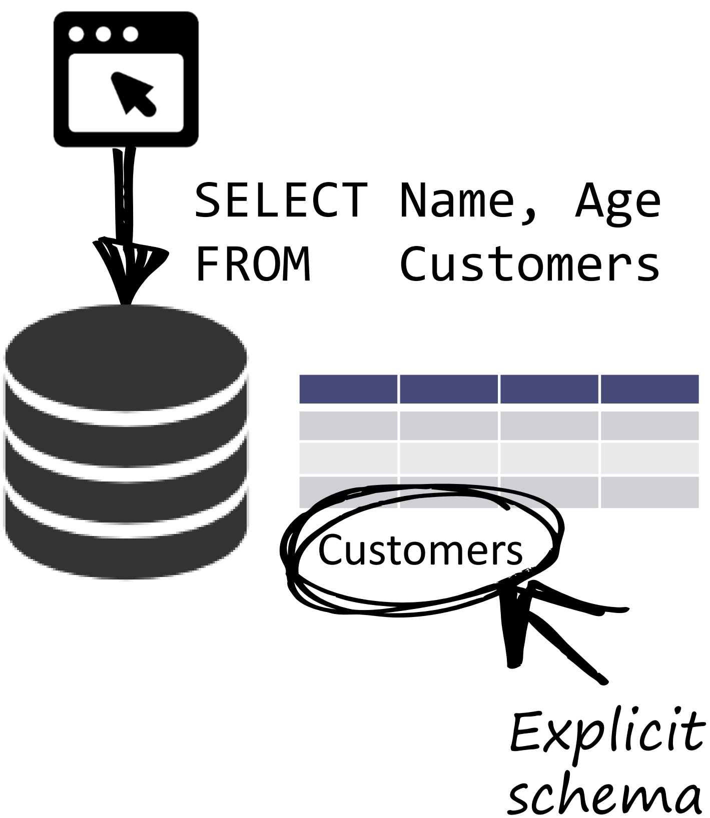


Scale-Out (*horizontal* scaling):

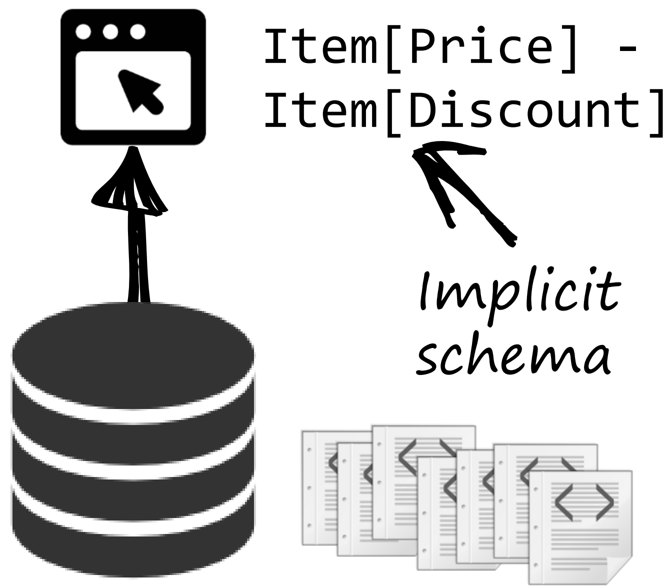


Schema-free Data Modeling

RDBMS:



NoSQL DB:

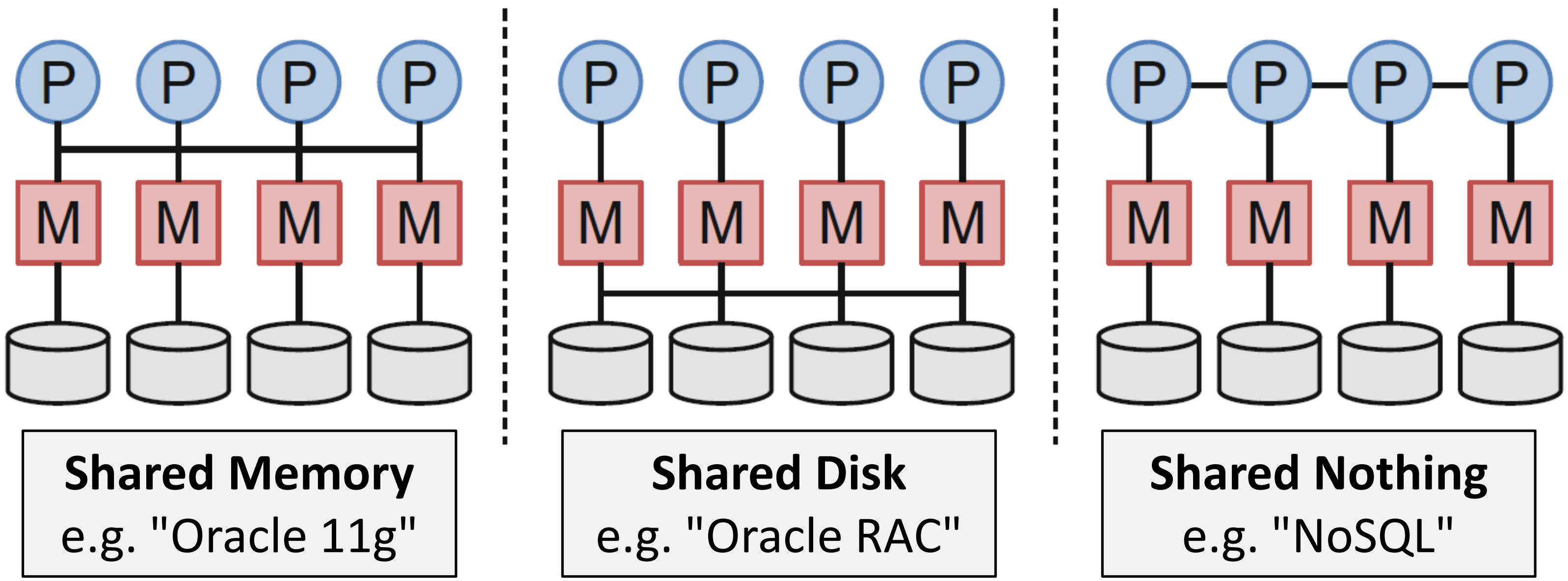


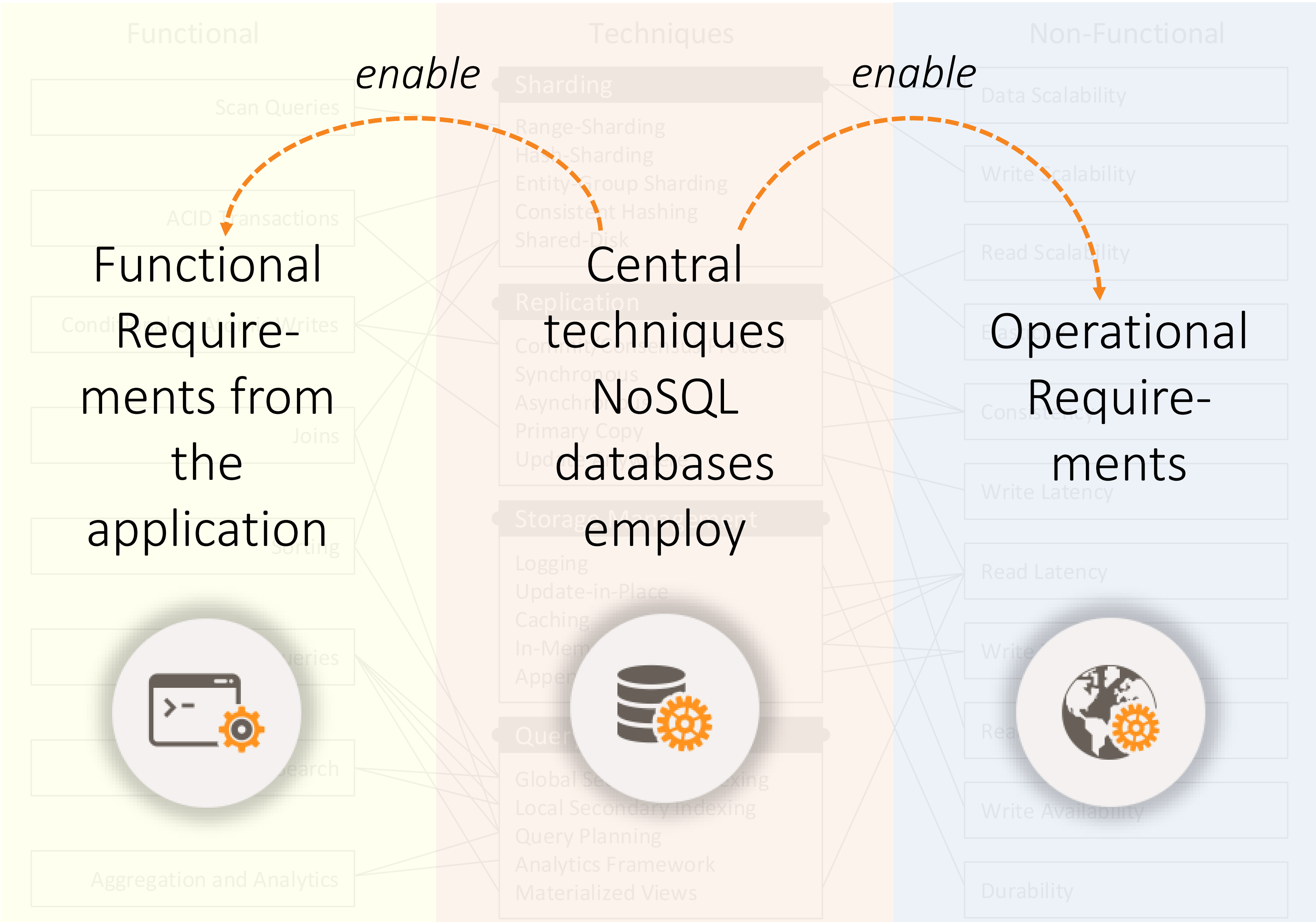
Paradigm Shift

Commercial DBMS	→	Open-Source DBMS
Specialized DB hardware (Oracle Exadata, etc.)	→	Commodity hardware
Highly available network (Infiniband, Fabric Path, etc.)	→	Commodity network (Ethernet, etc.)
Highly Available Storage (SAN, RAID, etc.)	→	Commodity drives (standard HDDs, JBOD)

Paradigm Shift

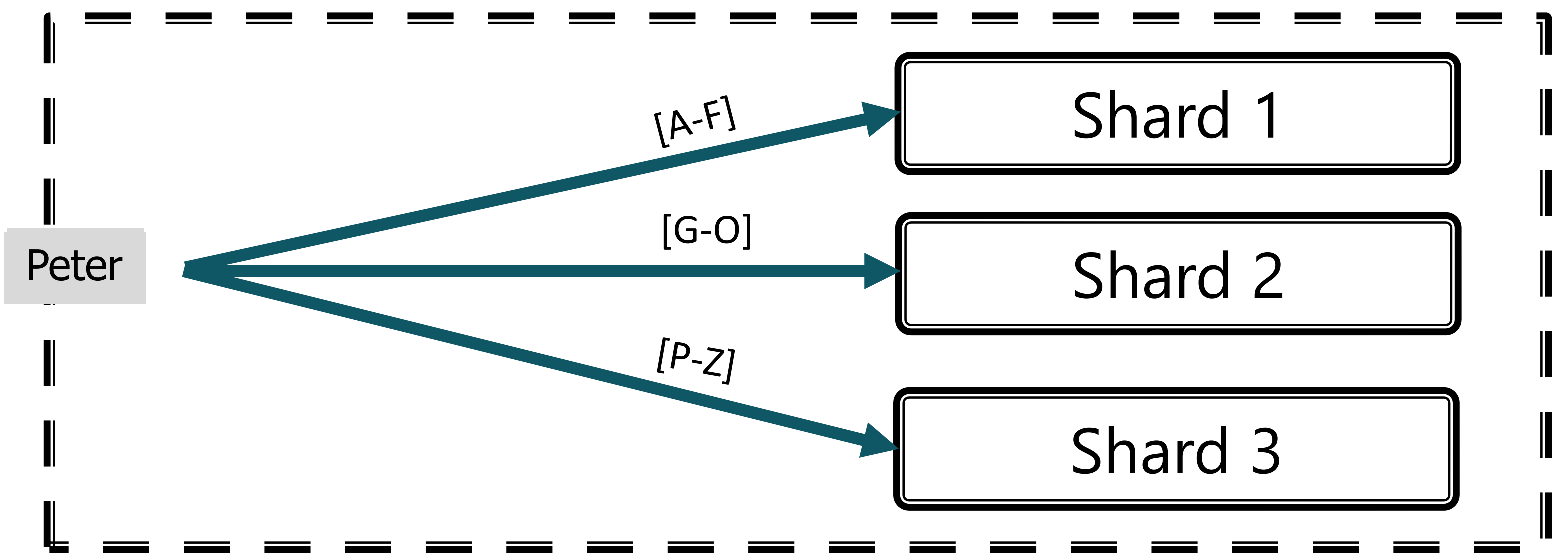
- Shift towards distributed computing architectures




Functional
Techniques
Non-Functional
enable
enable
Functional Require-ments from the application
Central techniques NoSQL databases employ
Operational Require-ments

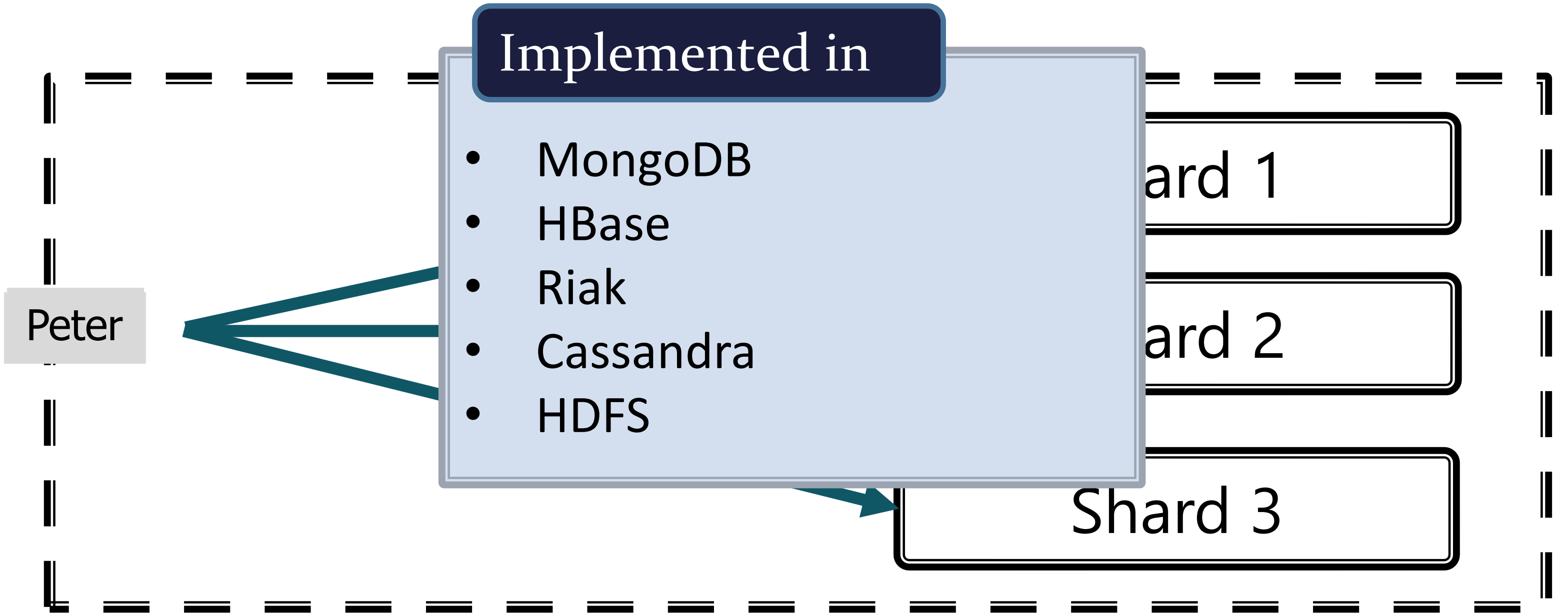
Sharding (aka Partitioning, Fragmentation)

- Horizontal distribution of data over server nodes
- **Paritioning strategies**: Hash-based vs. Range-based
- **Difficulty**: Multi-Shard-Operations (join, aggregation)



Sharding (aka Partitioning, Fragmentation)

- Horizontal distribution of data over server nodes
- **Paritioning strategies**: Hash-based vs. Range-based
- **Difficulty**: Multi-Shard-Operations (join, aggregation)



Implemented in

- MongoDB
- HBase
- Riak
- Cassandra
- HDFS

Sharding

Hash-based Sharding

- Builds hash of data values (e.g. over the key) to determine a partition (shard) for a data item (tuple)
- **Pro**: Perfectly even distribution
- **Contra**: No data locality – data items are pseudorandomly scattered over partitions

Range-based Sharding

- Assigns ranges defined over fields (shard keys) to partitions
- **Pro**: Data locality preserved (for shard keys)
- **Contra**: distribution might grow uneven → repartitioning/balancing required

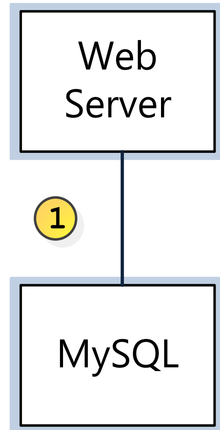
Traditional Sharding

Example: **Tumblr**

- Caching
- Sharding from application

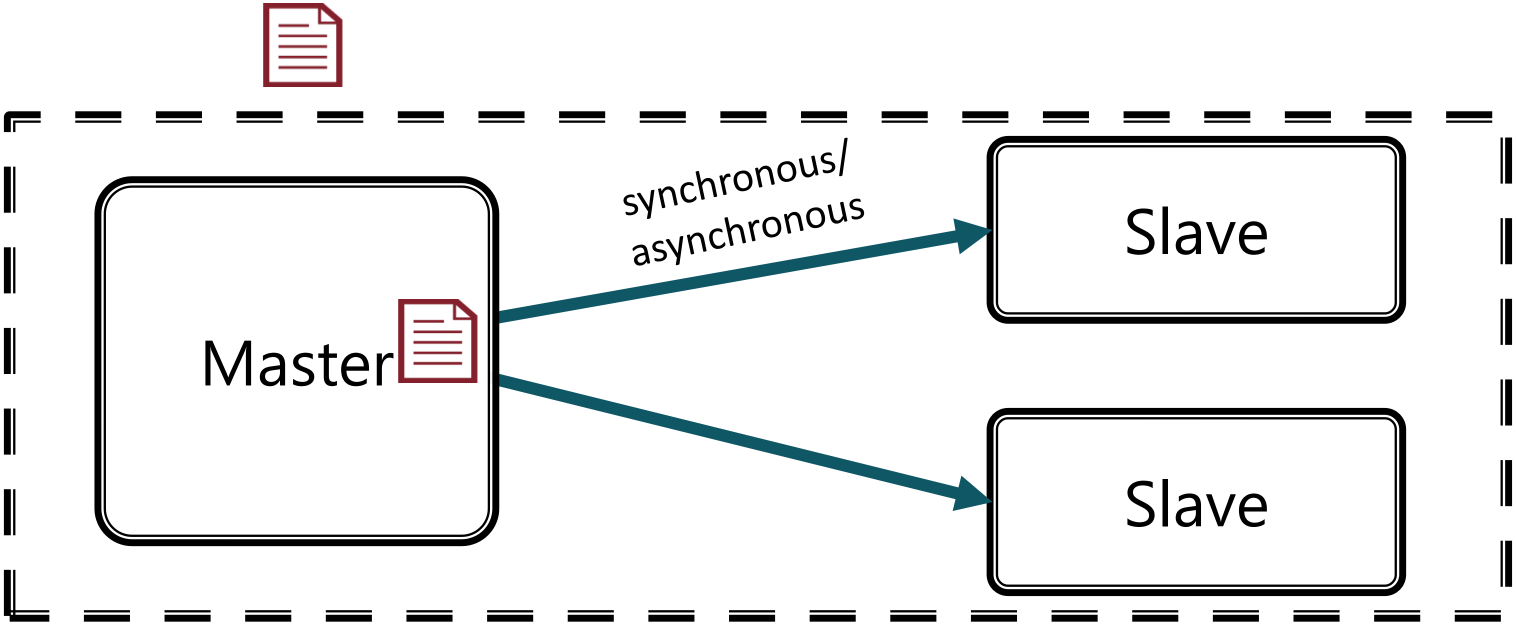
Moved towards:

- Redis
- HBase



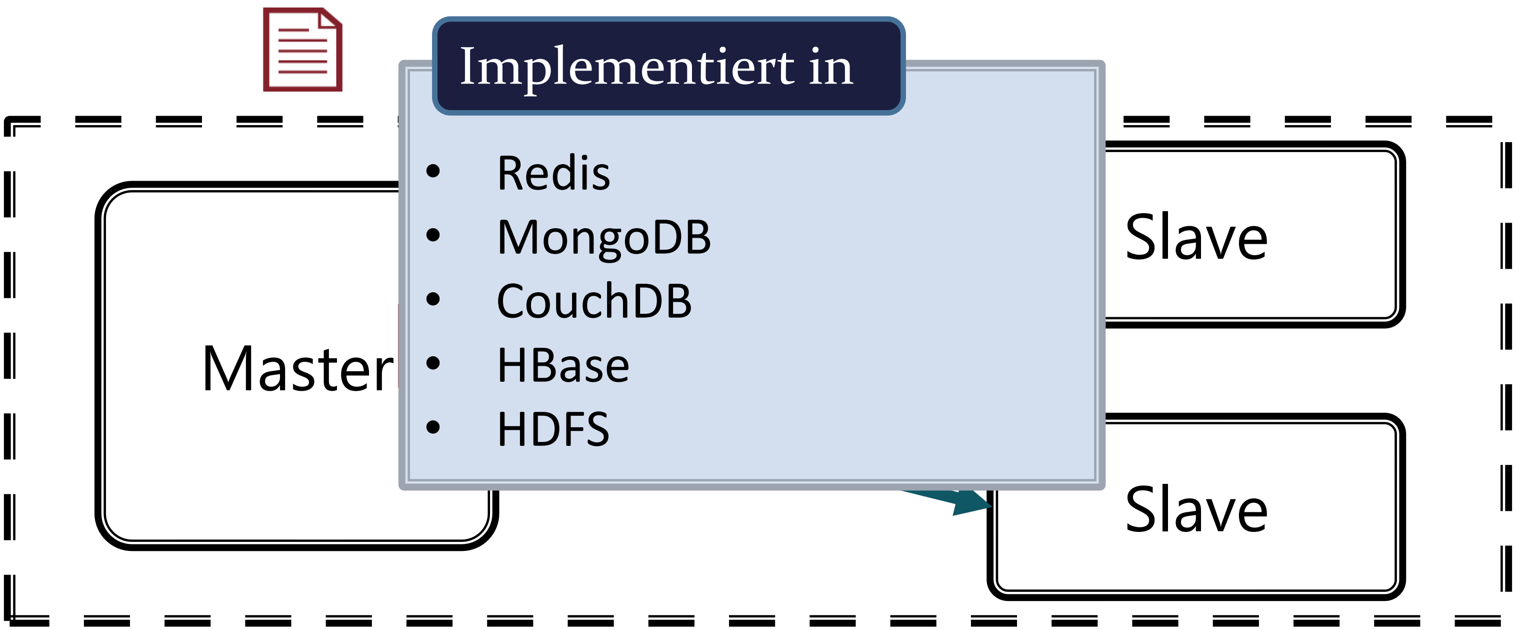
Replication

- Stores *N* copies of each data item
- **Consistency model**: synchronous vs asynchronous
- **Coordination**: Multi-Master, Master-Slave



Replication

- Stores *N* copies of each data item
- **Consistency model**: synchronous vs asynchronous
- **Coordination**: Multi-Master, Master-Slave



Replication: consistency models

Asynchronous

- Writes are acknowledged immdediately
- Performed through *log shipping* or *update propagation*
- **Pro**: Fast writes, no coordination needed
- **Contra**: Replica data potentially stale (*inconsistent*)

Synchronous

- The node accepting writes synchronously propagates updates/transactions before acknowledging
- **Pro**: Consistent
- **Contra**: needs a commit protocol (more roundtrips), unavaialable under certain network partitions

Replication: coordination

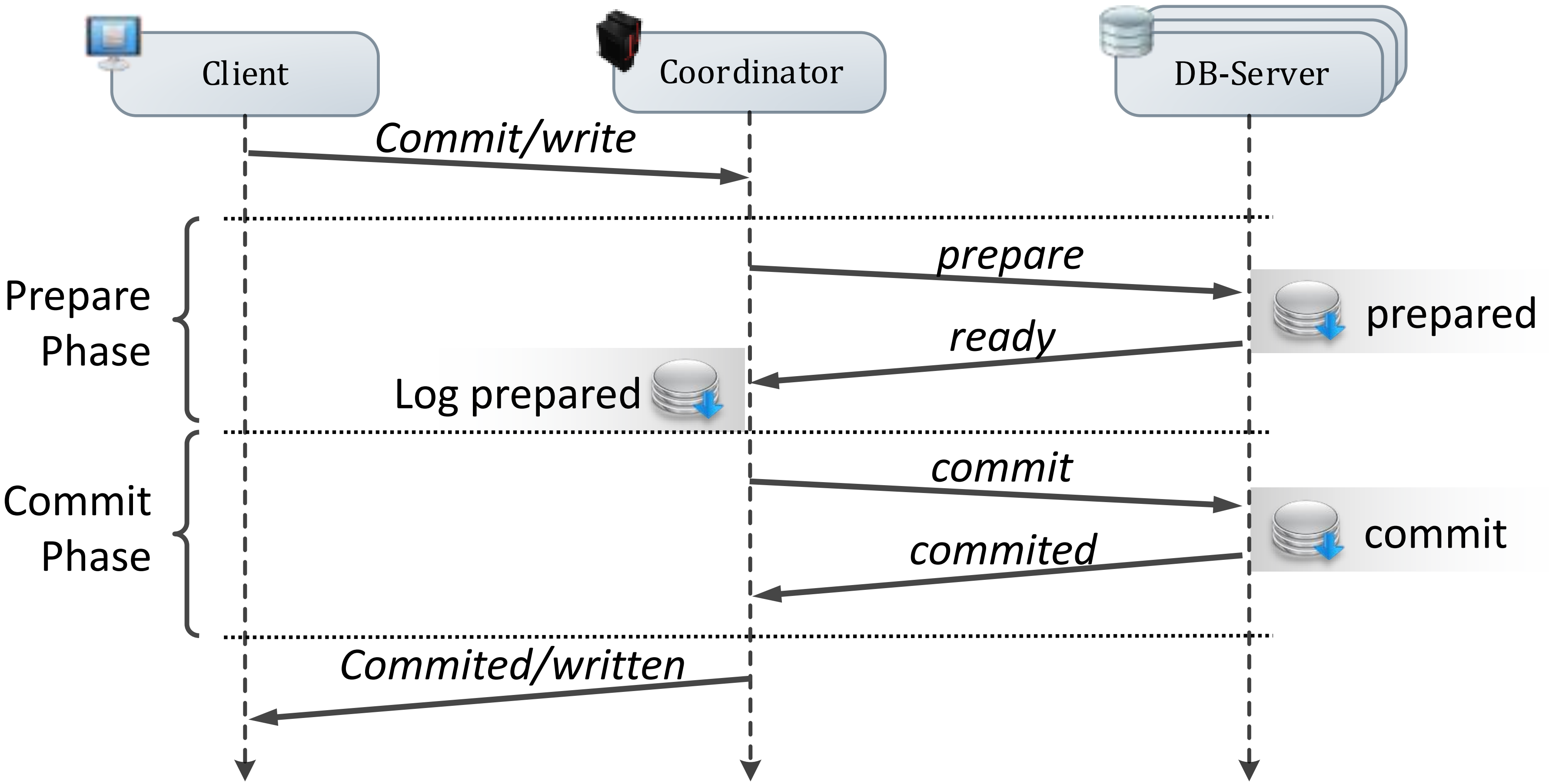
Master-Slave (*Primary Copy*)

- Only a dedicated master is allowed to accept writes, slaves are read-replicas
- **Pro**: reads from the master are consistent
- **Contra**: master is a bottleneck and SPOF

Multi-Master (*Update anywhere*)

- The server node accepting the writes synchronously propagates the update or transaction before acknowledging
- **Pro**: fast and highly-available
- **Contra**: either needs complicated coordination protocols (e.g. Paxos) or is inconsistent

Synchronous Replication: 2PC



Error scenarios (timeouts)

In **INITIAL**:

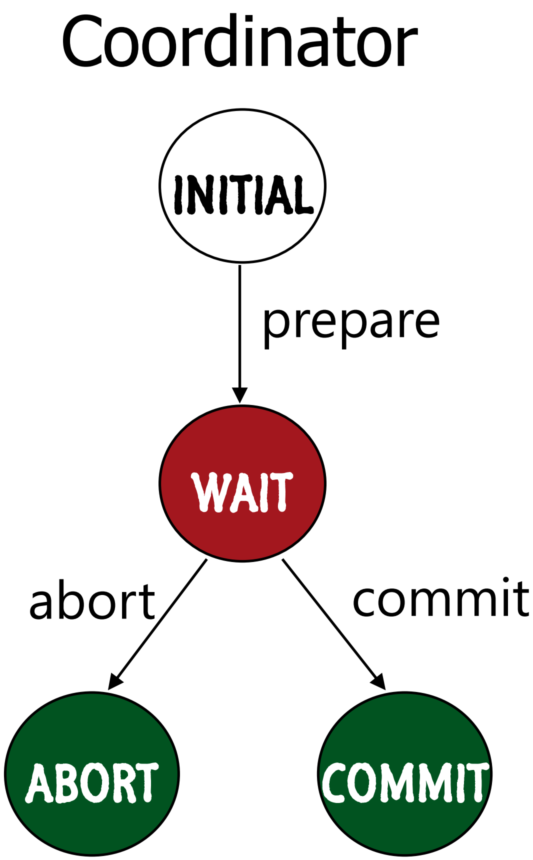
- No consequence

In **WAIT**:

- Abort

In **ABORT** oder **COMMIT**:

- Resend *commit/abort* and wait for all responses



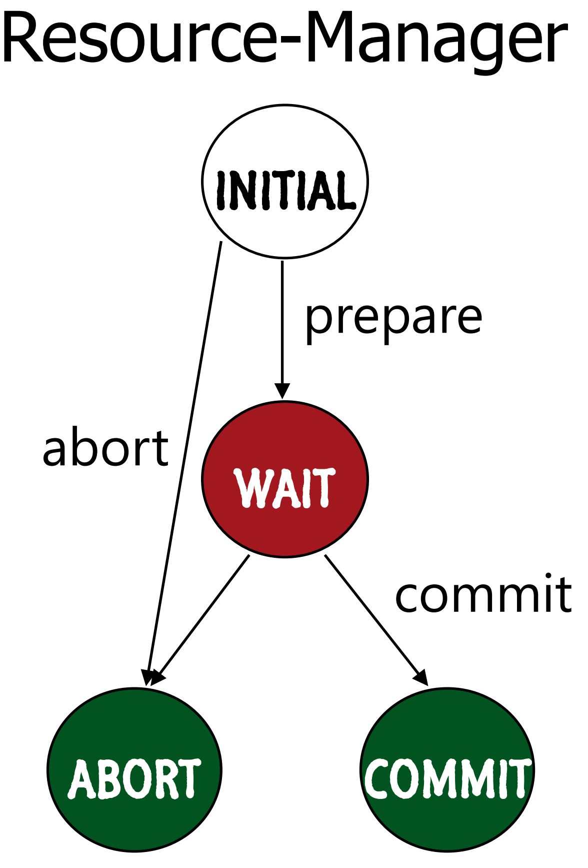
Error scenarios (timeouts)

In **INITIAL**:

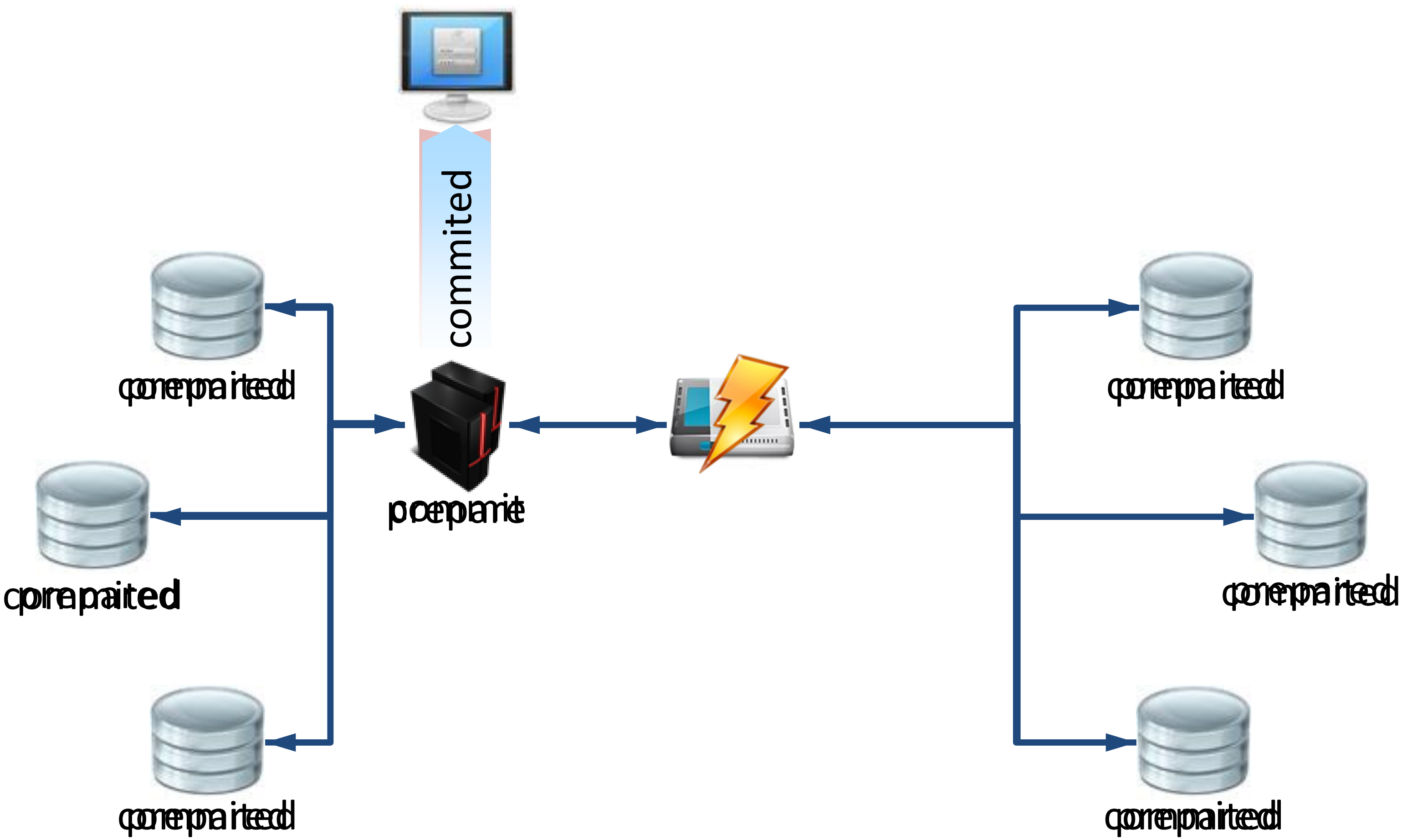
- Coordinator probably crashed → Abort („*Presumed-Abort-Protocol*“)

In **WAIT**:

- Wait for message from coordinator



2PC is not available



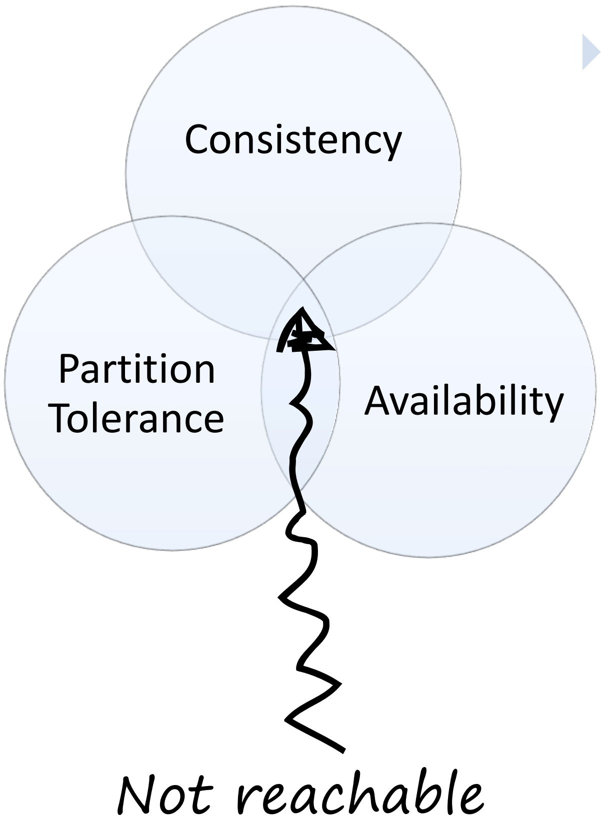
Commit protocols

- Provide atomic propagation of writes or commits
- Most prominent implementation: 2-Phase-Commit

R *Read-RMs,* W *Read/Write-RMs,* N=R+W

Commit protocol	Messages	Property
1-Phase-Commit	2N	Not always possible
Linear 2PC	2N+1	Not parallel
Hierarchical and normal 2PC	4N-2R	Might block indefinitely
3-Phase-Commit	6N-4R	No consistency guarantee
Paxos-Commit	3N+2F(N+1)+1	F failures tolerated
Distributed 2PC	N^2	No 2nd phase

CAP-Theorem



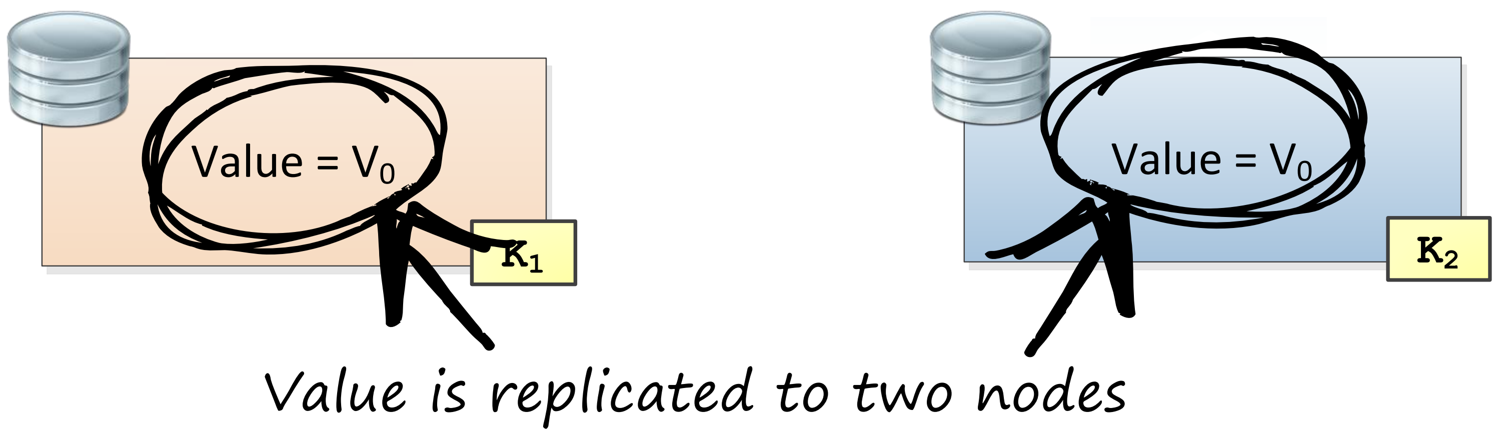
- Classifies distributed databases
- Only 2 out of 3 properties are achievable at a time:
  - **Consistency**: all clients have the same view on the data
  - **Availability**: every request to a non-failed node most result in correct response
  - **Partition tolerance**: the system has to continue working, even under arbitrary network partitions

Eric Brewer, ACM-PODC Keynote, Juli 2000

Gilbert, Lynch: Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services, SigAct News 2002

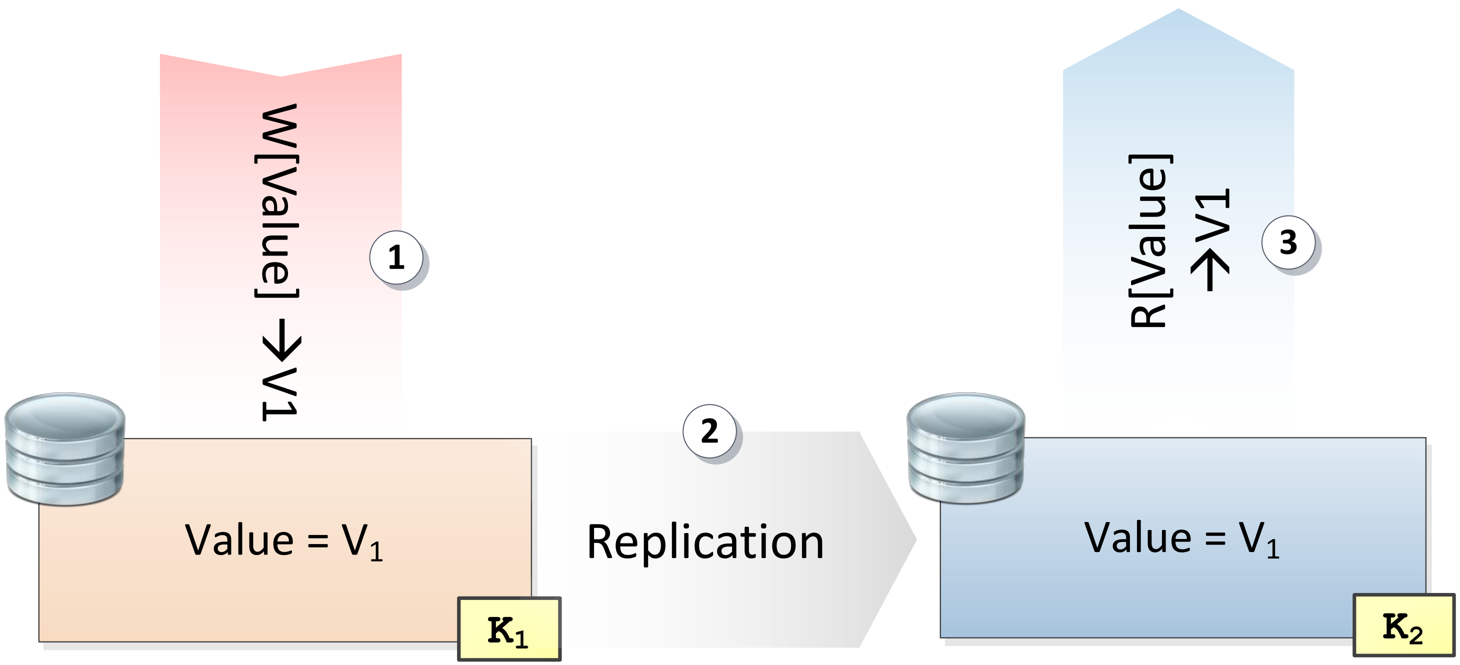
CAP-Theorem: simplified proof

- Intuition for the impossibility of simultaneously achieving **C**, **A** and **P** at once:



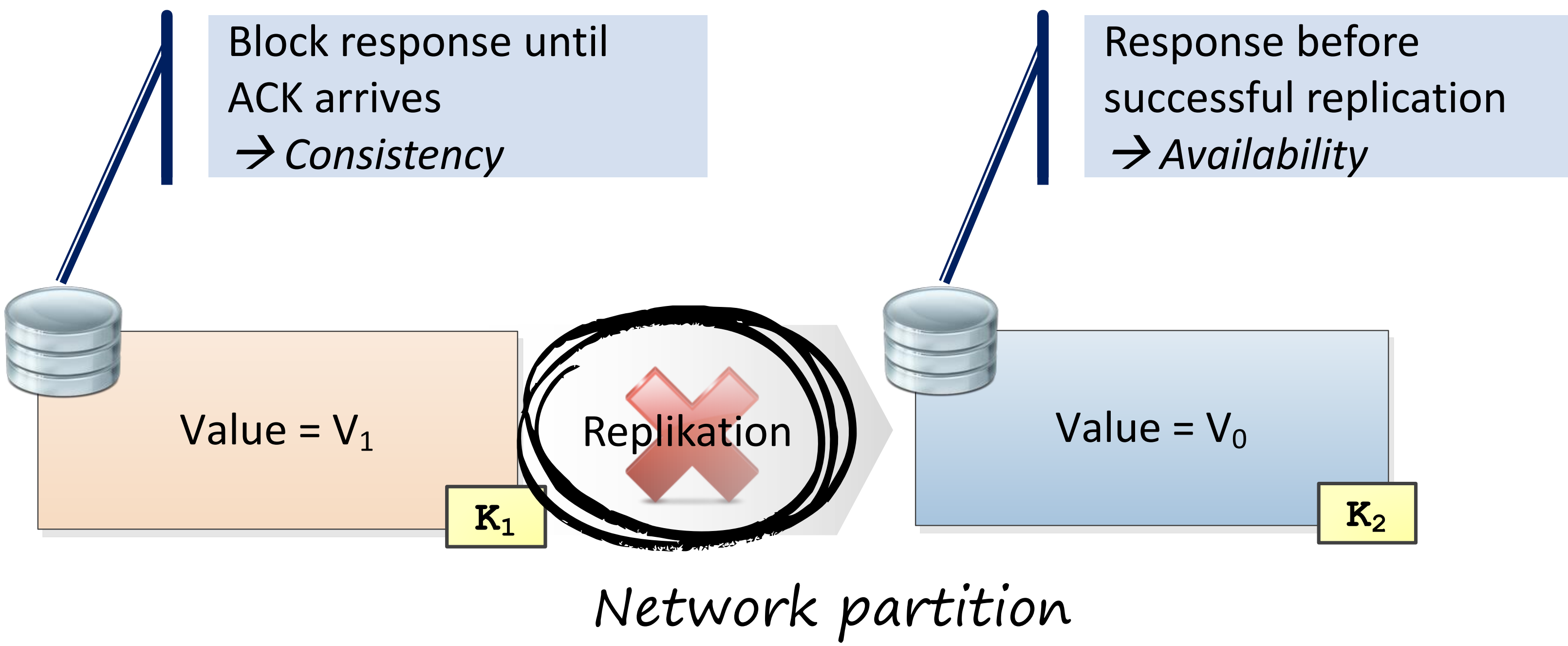
CAP-Theorem: simplified proof

- Failure-free reading and writing:

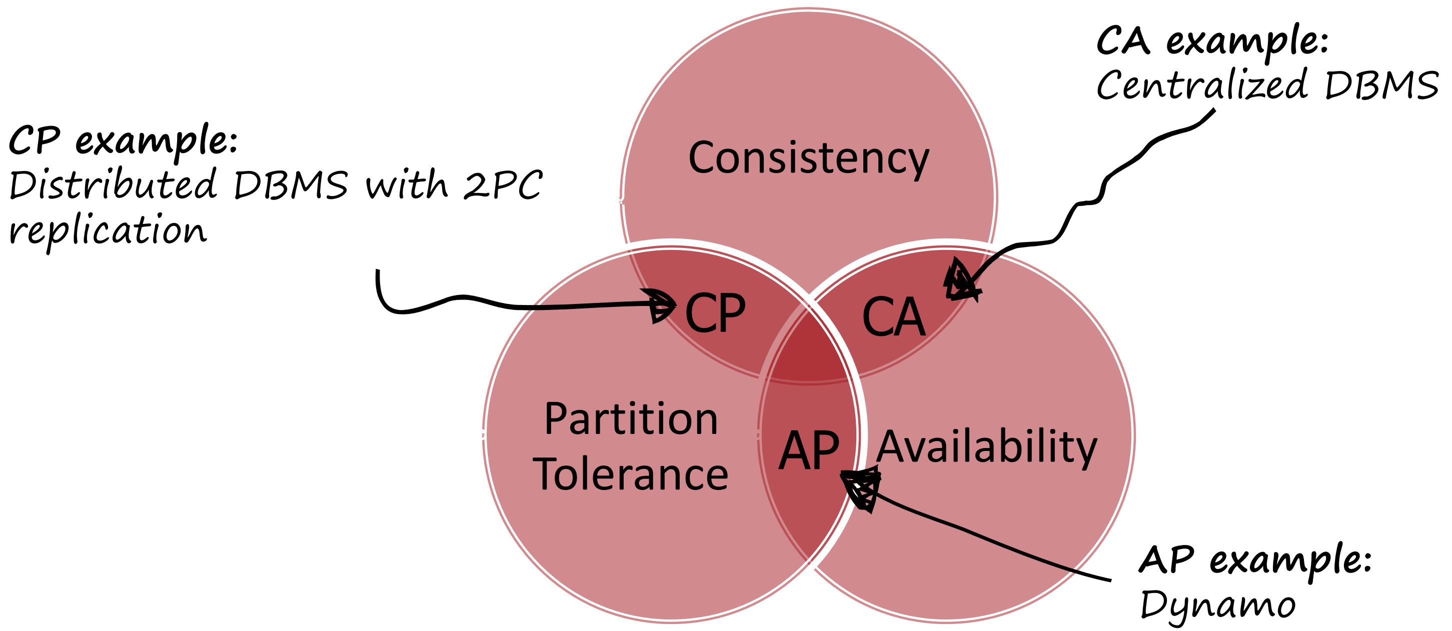


CAP-Theorem: simplified proof

- **Problem**: when a network partition occurs, either consistency or availability have to be given up



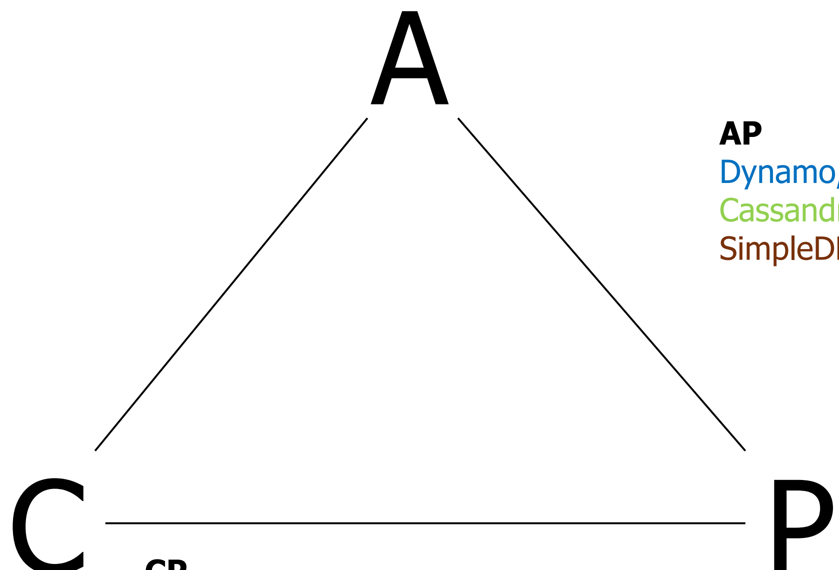
CAP-Theorem: Wrap-up



NoSQL triangle

Data models
- Relational
- Key-Value
- Wide-Column
- Document-Oriented

Every client can always read and write



CA
Oracle, MySQL, …

AP
Dynamo, Redis
Cassandra
SimpleDB, Riak

All clients share the same view on the data

CP
Postgres, MySQL Cluster, Oracle RAC
BigTable, HBase, Azure Table
MongoDB

All nodes continue working under network partitions

Nathan Hurst: Visual Guide to NoSQL Systems
http://blog.nahurst.com/visual-guide-to-nosql-systems

ACID vs BASE

„Gold standard" for RDBMSs

ACID

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

BASE

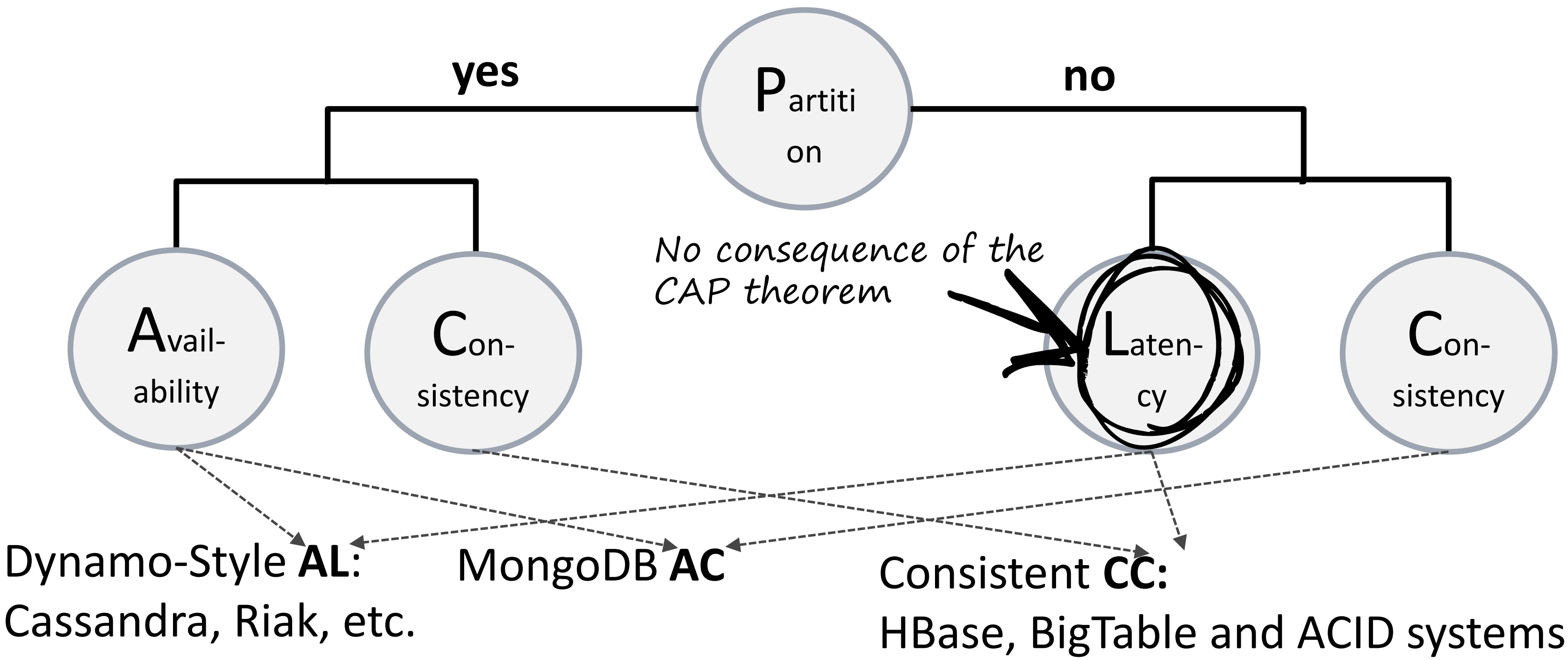
- **B**asically **A**vailable
- **S**oft State
- **E**ventually Consistent

Model of many NoSQL systems

http://queue.acm.org/detail.cfm?id=1394128

PACELC – an alternative CAP formulation

- **Idea**: Classify system according to their heaviour during network partitions



Abadi, Daniel. "Consistency tradeoffs in modern distributed database system design: CAP is only part of the story."

Negative Results

In Distributed Computing

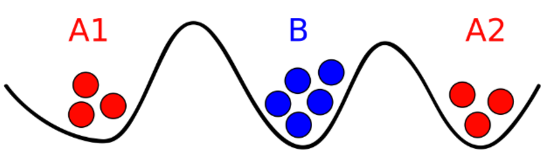
Asychronous **Network,**
Unreliable **Channel**

Atomic Storage

Impossible:
CAP Theorem

Consensus

Impossible:
2 Generals Problem



Asychronous **Network,**
Reliable **Channel**

Atomic Storage

Possible:
Attiya, Bar-Noy, Dolev (ABD) Algorithm

Consensus

Impossible:
Fisher **L**ynch **P**atterson (FLP) Theorem

Negative Results
Consensus Algorithms

Safety Properties

- Consensus:
  - *Agreement*: No two processes can commit different decisions
  - *Validity (Non-triviality)*: If all initial values are same, nodes must commit that value
  - *Termination*: Nodes commit eventually

Liveness Property

- No algorithm *guarantees* termination (FLP)
- Algorithms:
  - **Paxos** (e.g. Google Chubby, Spanner, Megastore, Cassandra Lightweight Transactions)
  - **Raft** (e.g. etcd service)
  - Zookeeper Atomic Broadcast (**ZAB**)

Negative Results

Correctness/Serializability

Distributed ACID and availability are incompatible:

Write A=1
Read B

Write B=1
Read A

$$w_1(a = 1)\, r_1(b = \perp) \qquad w_2(b = 1)\, r_2(a = \perp)$$

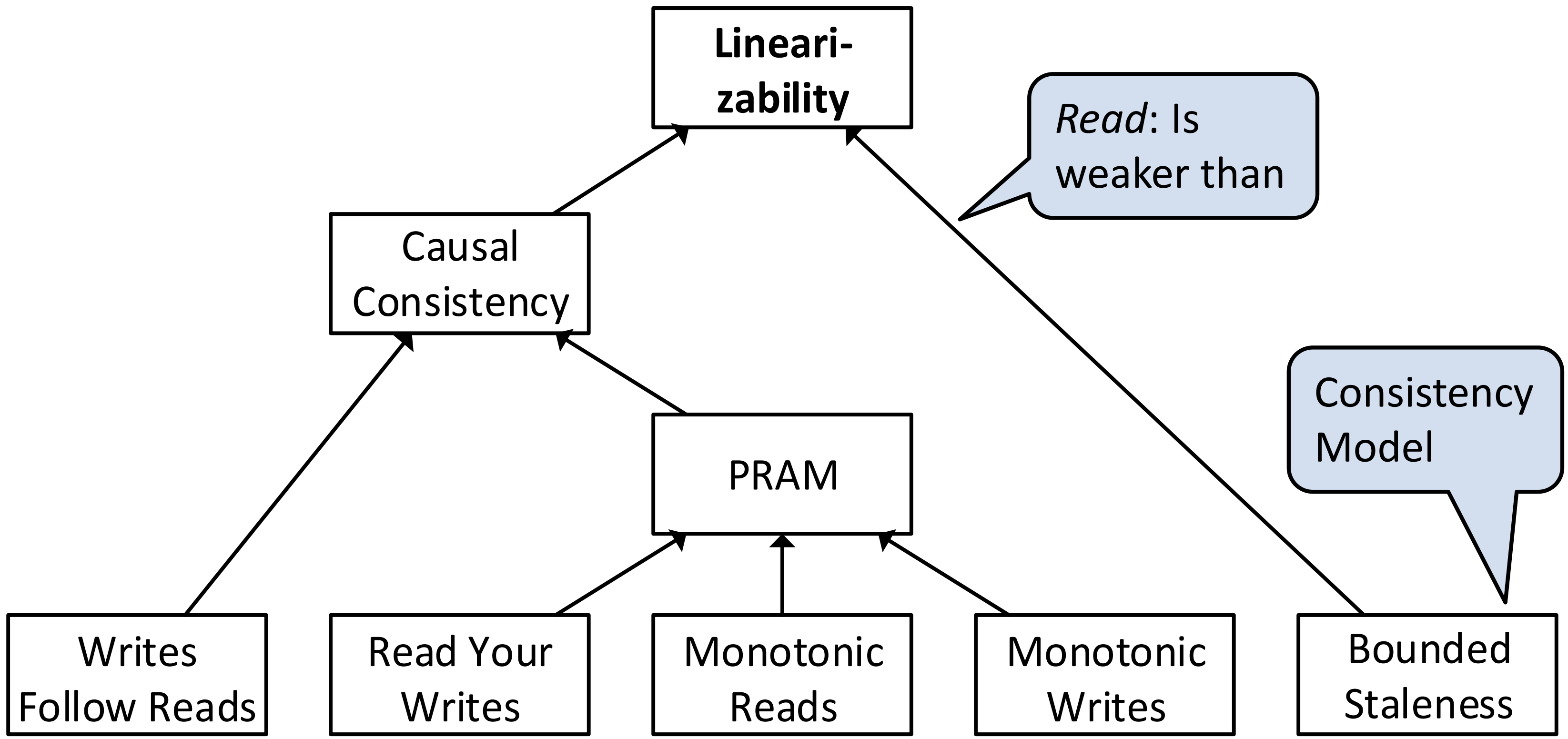
- Weaker isolations levels are possible:
  - RAMP Transactions (P. Bailis, A. Fekete, A. Ghodsi, J. M. Hellerstein, und I. Stoica, „Scalable Atomic Visibility with RAMP Transactions“, SIGMOD 2014)
- **Consequence:** trade-offs are important

Typical Trade-offs

Read Performance	Write Performance
Latency	Durability
Synchronous replication	Asynchronous replication
Row-based	Column-based
Transactions	Availability
REST	RPC
Commodity servers	High-end hardware
Normalisation	Denormalisation
Schemas	Schemafreeness

Client-Centric Consistency Models

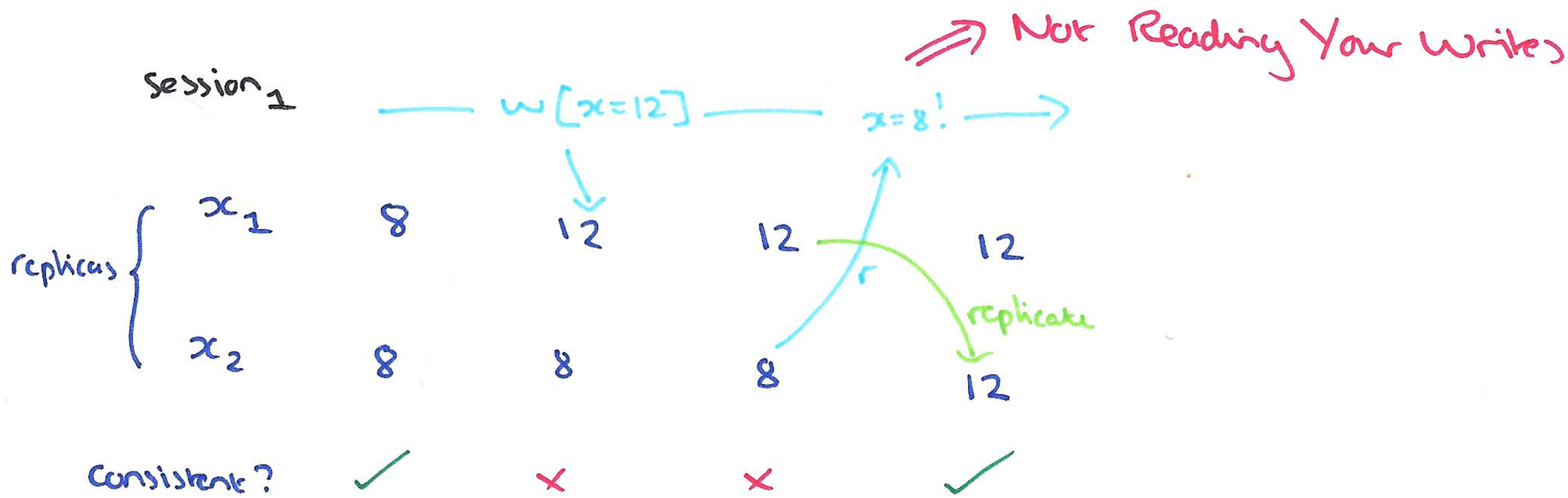
- Define models that relax strong consistency (=linearizability) in different aspects



Bailis, Peter, et al. "Highly available transactions: Virtues and limitations." Proceedings of the VLDB Endowment 7.3 (2013): 181-192.

Read Your Writes (RYW)

Definition: Once the user has written a value, subsequent reads will return this value (or newer versions if other writes occurred in between); the user will never see versions older than his last write.

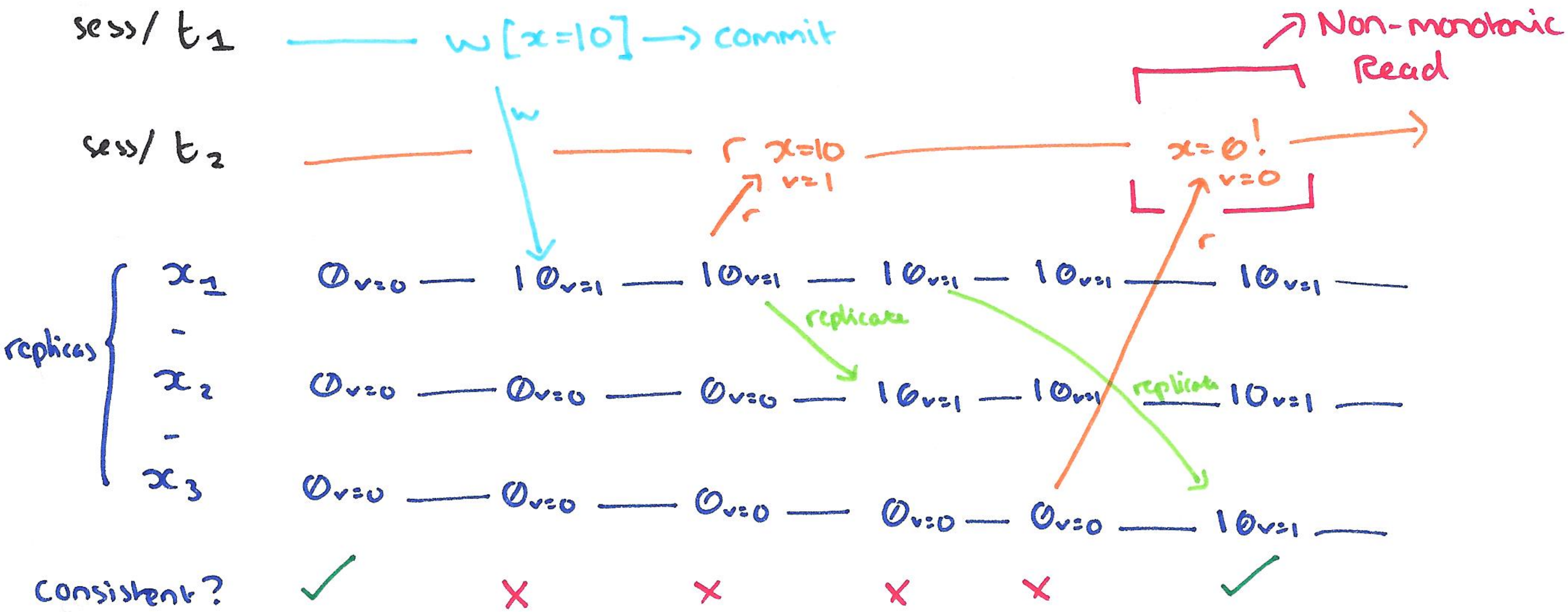


https://blog.acolyer.org/2016/02/26/distributed-consistency-and-session-anomalies/

Wiese, Lena. Advanced Data Management: For SQL, NoSQL, Cloud and Distributed Databases. De Gruyter, 2015.

Monotonic Reads (MR)

Definition: Once a user has read a version of a data item on one replica server, it will never see an older version on any other replica server

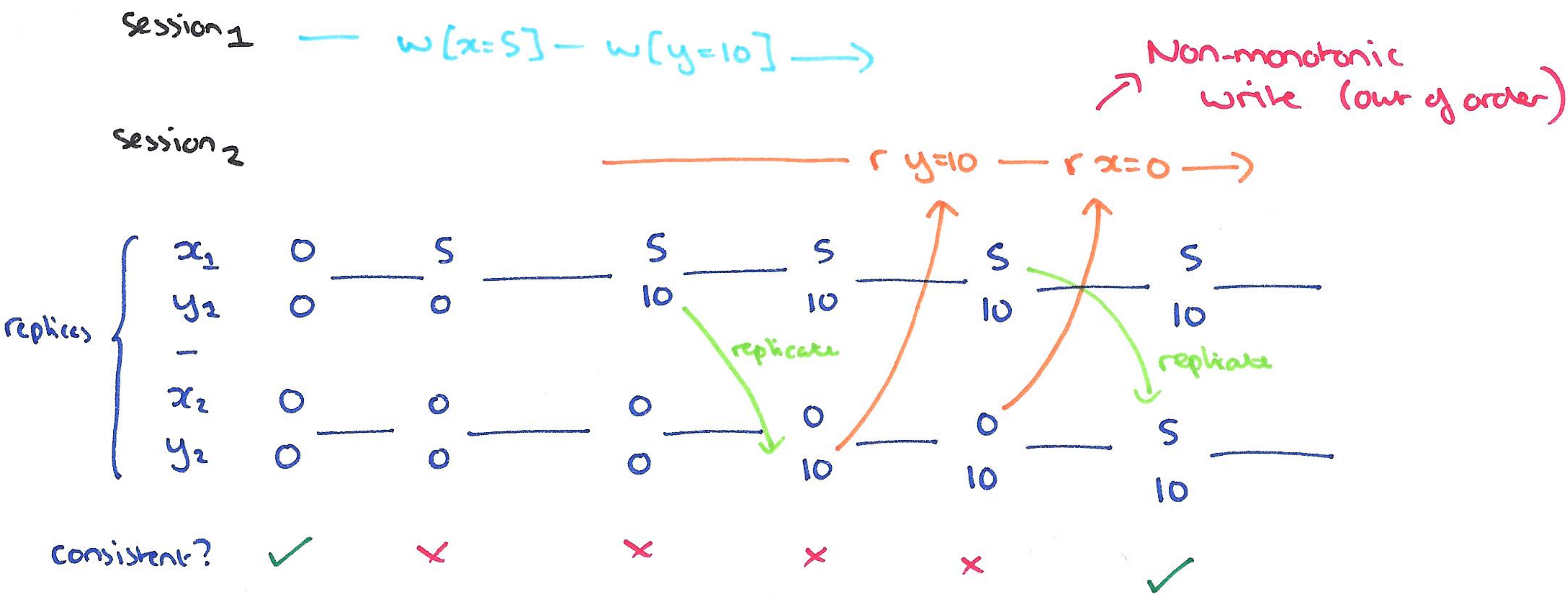


https://blog.acolyer.org/2016/02/26/distributed-consistency-and-session-anomalies/

Wiese, Lena. Advanced Data Management: For SQL, NoSQL, Cloud and Distributed Databases. De Gruyter, 2015.

Montonic Writes (MW)

Definition: Once a user has written a new value for a data item in a session, any previous write has to be processed before the current one. I.e., the order of writes inside the session is strictly maintained.

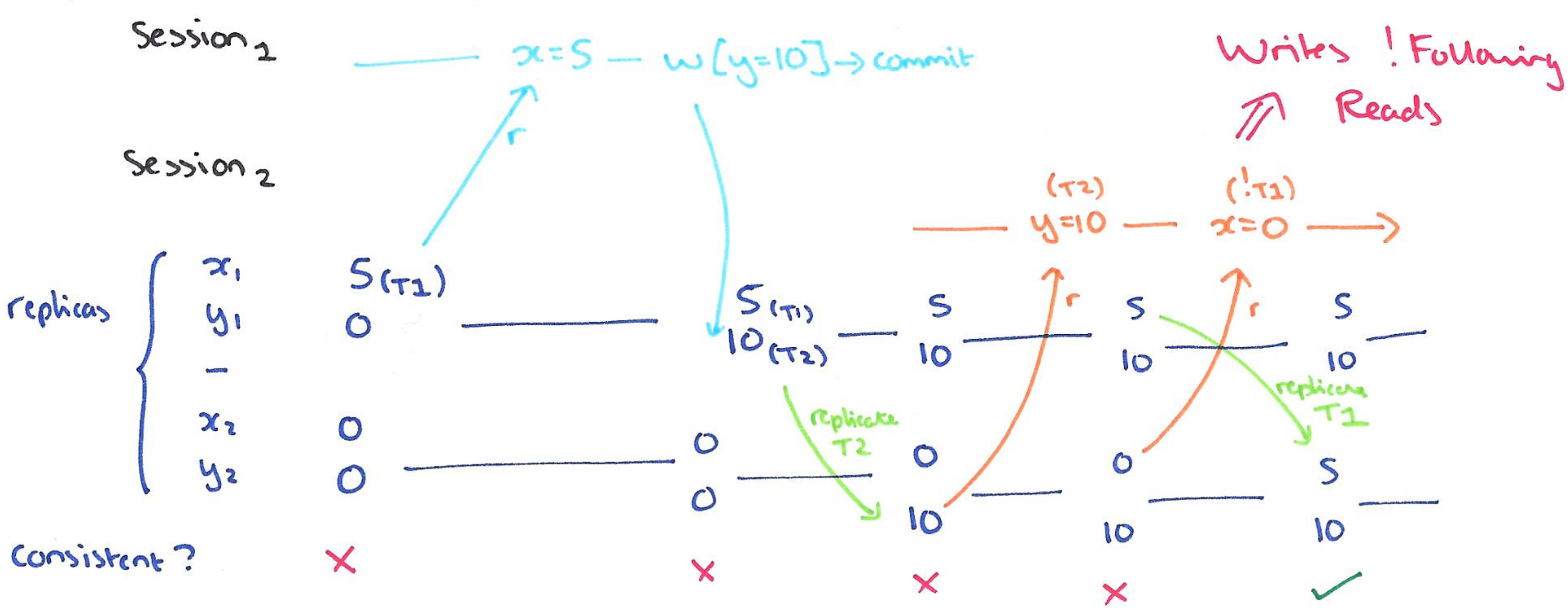


https://blog.acolyer.org/2016/02/26/distributed-consistency-and-session-anomalies/

Wiese, Lena. Advanced Data Management: For SQL, NoSQL, Cloud and Distributed Databases. De Gruyter, 2015.

Writes Follow Reads (WFR)

Definition: When a user reads a value written in a session after that session already read some other items, the user must be able to see those *causally relevant* values too.

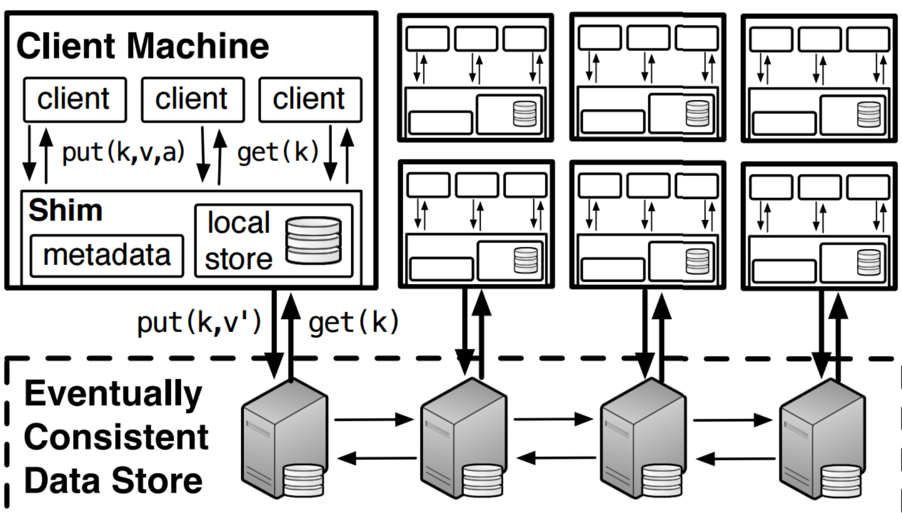


📖 https://blog.acolyer.org/2016/02/26/distributed-consistency-and-session-anomalies/

📖 Wiese, Lena. Advanced Data Management: For SQL, NoSQL, Cloud and Distributed Databases. De Gruyter, 2015.

PRAM and Causal Consistency

- Combinations of previous session consistency guarantess
  - PRAM = MR + MW + RYW
  - Causal Consistency = PRAM + WFR
- All consistency level up to causal consistency can be guaranteed with **high availability**
- Example: Bolt-on causal consistency



Bailis, Peter, et al. "Bolt-on causal consistency." Proceedings of the 2013 ACM SIGMOD, 2013.

Bounded Staleness

- Either **time-based**:

> **t-Visibility (Δ-atomicity):** the inconsistency window comprises at most t time units; that is, any value that is returned upon a read request was up to date t time units ago.

- Or **version-based:**

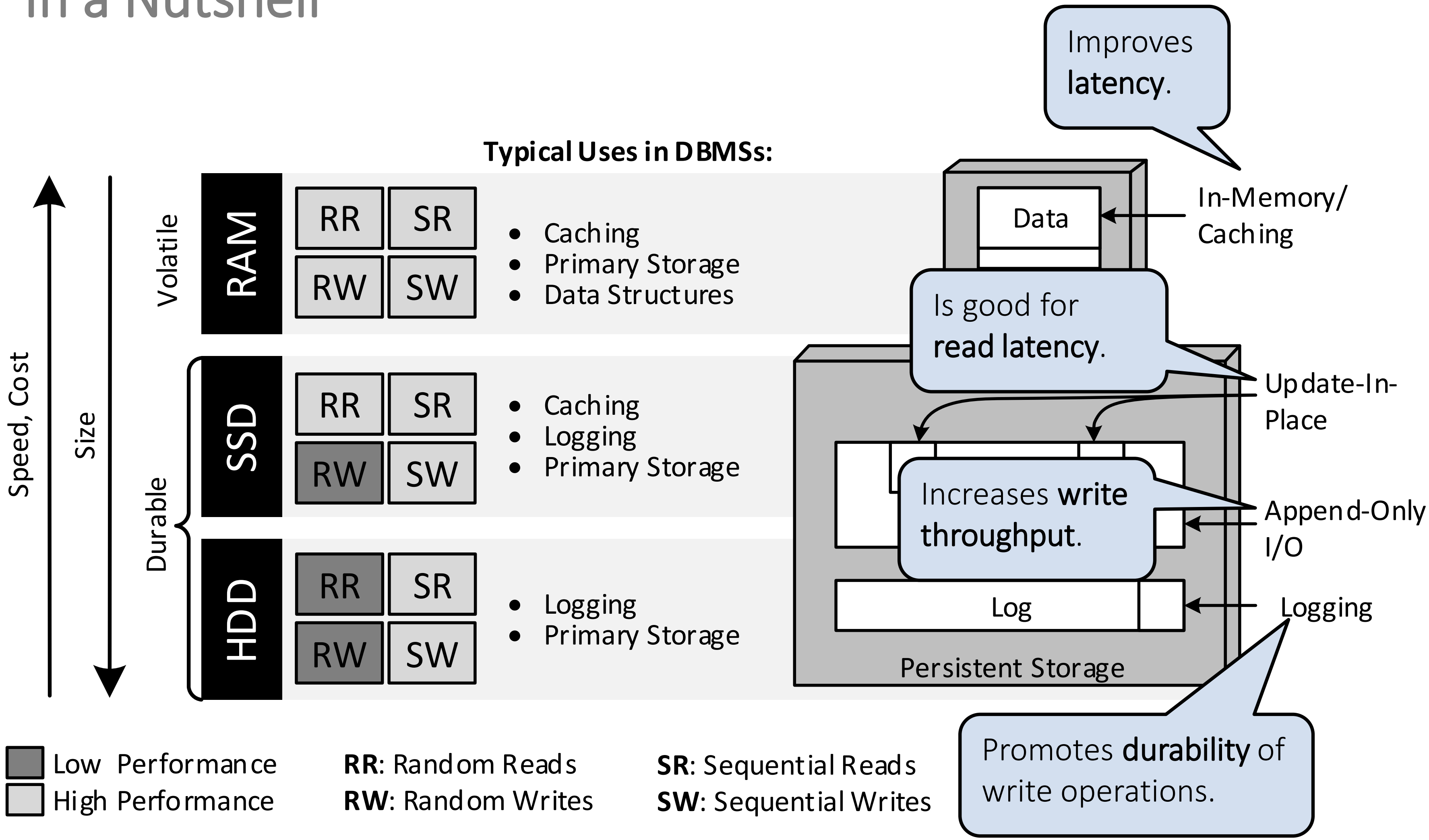
> **k-Staleness:** the inconsistency window comprises at most k versions; that is, lags at most k versions behind the most recent version.

- Both are *not* achievable with high availability

Wiese, Lena. Advanced Data Management: For SQL, NoSQL, Cloud and Distributed Databases. De Gruyter, 2015.

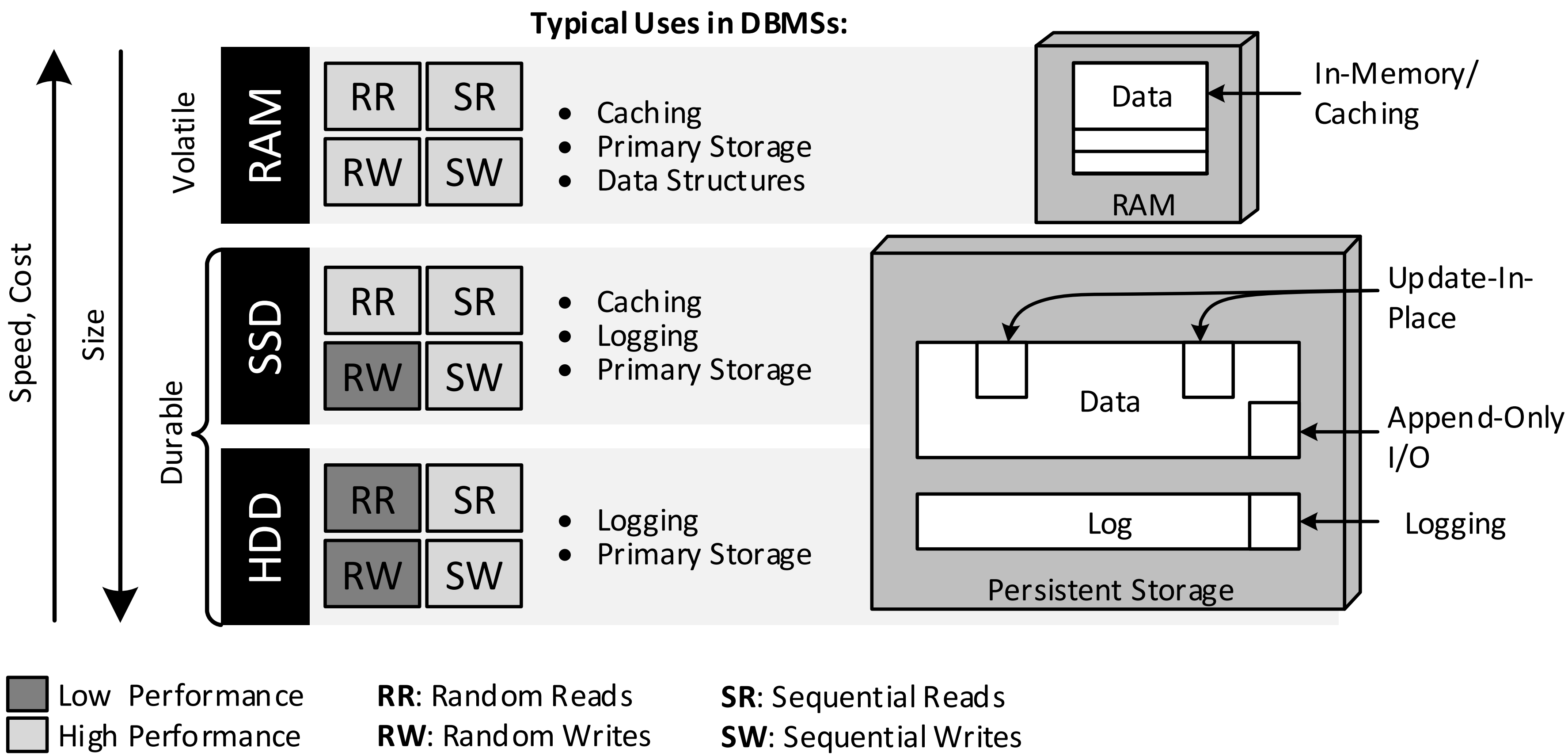
NoSQL Storage Management

In a Nutshell



NoSQL Storage Management

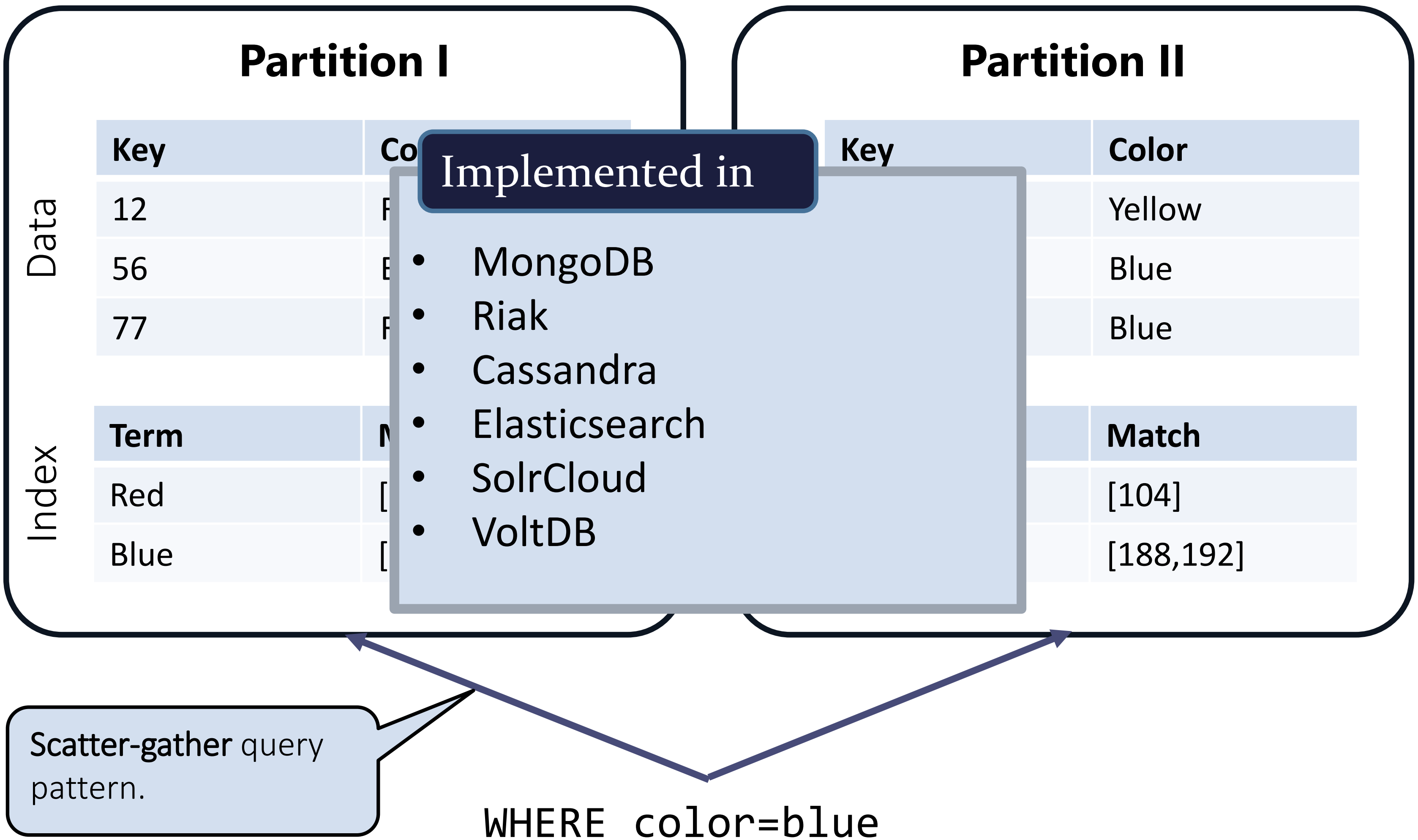
In a Nutshell



RR: Random Reads
RW: Random Writes
SR: Sequential Reads
SW: Sequential Writes

Local Secondary Indexing

Partitioning By Document



Kleppmann, Martin. "Designing data-intensive applications." (2016).

Local Secondary Indexing

Partitioning By Document

Partition I

Data

Key	Color
12	Red
56	Blue
77	Red

Index

Term	Match
Red	[12,77]
Blue	[56]

Partition II

Data

Key	Color
104	Yellow
188	Blue
192	Blue

Index

Term	Match
Yellow	[104]
Blue	[188,192]

Indexing is always **local to a partition.**

Scatter-gather query pattern.

`WHERE color=blue`

Kleppmann, Martin. "Designing data-intensive applications." (2016).

Global Secondary Indexing

Partitioning By Term

Partition I

Data

Key	Color
12	
56	
77	

Index

Term	
Yellow	[
Blue	[56, 188, 192]

Consist…
maint…
distrib…

Partition II

Key	Color
104	Yellow
	Blue
	Blue

	Match
	[12,77]

Implemented in

- DynamoDB
- Oracle Datawarehouse
- Riak (Search)
- Cassandra (Search)

Targeted Query

`WHERE color=blue`

Kleppmann, Martin. "Designing data-intensive applications." (2016).

Global Secondary Indexing

Partitioning By Term

Partition I

Data

Key	Color
12	
56	
77	

Consistent Index-maintenance requires **distributed transaction.**

Index

Term	Match
Yellow	[104]
Blue	[56, 188, 192]

Partition II

Data

Key	Color
104	Yellow
188	Blue
192	Blue

Index

Term	Match
Red	[12,77]

Targeted Query

```
WHERE color=blue
```

Kleppmann, Martin. "Designing data-intensive applications." (2016).

Wrap-up

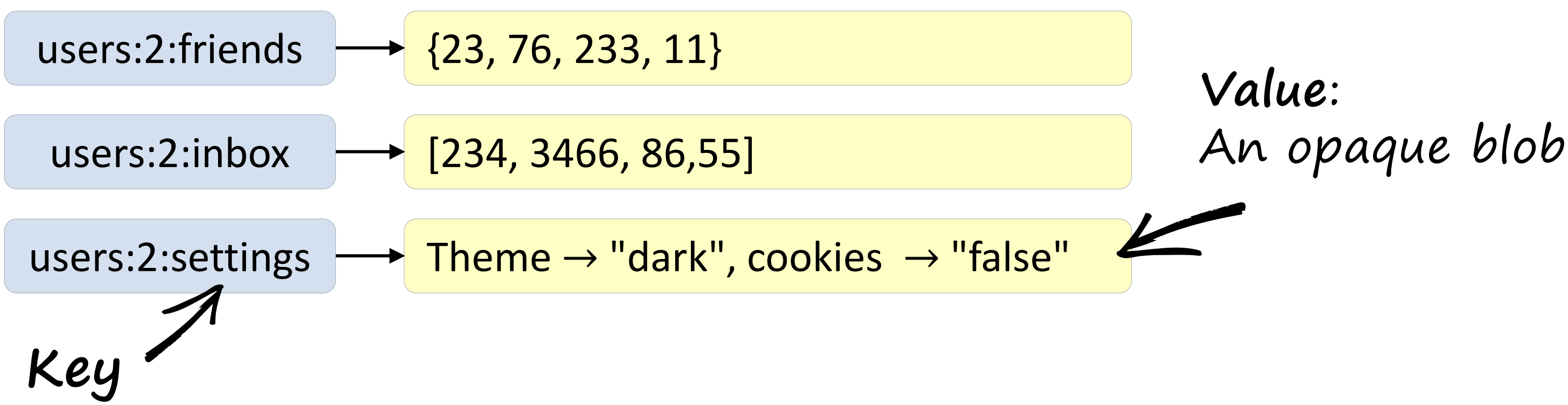
- High data volumes, unstructured sources and new kinds of applications triggered BigData and NoSQL technologies
- Shared Nothing architectures for **horizontal scalability**
  - **Replication** enables read scalability and fault tolerance
  - **Sharding** enables write scalability and data volume scalability
- **CAP Theorem**: Consistency, Availability and Partition Tolerance cannot be achieved at the same time
  - **BASE** (Basically available, soft-state, eventually consistent) paradigm as an alternative to ACID
- **2-Phase-Commit** (2PC): popular protocol for atomic commitment; without availability guarantee
- **Consistency** can be relaxed in various ways

Outline

- Foundations: Big Data, Scalability, Avaialbility
- The 4 Classes of NoSQL Databases
  - Key-Value stores
  - Wide-Column stores
  - Document stores
  - Graph databases
  - Other classes
- NoSQL Examples: concrete Architectures, Systems, APIs
- Cloud Databases

Key-Value Stores

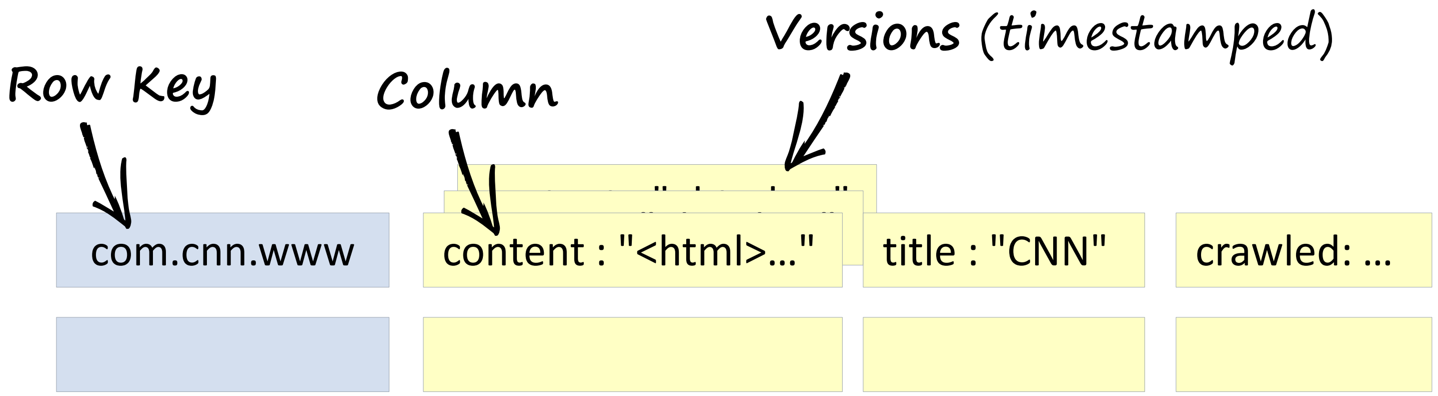
- **Data model:** (key) -> value
- **Interface**: CRUD (Create, Read, Update, Delete)



- Examples: Amazon Dynamo (AP), Riak (AP), Redis (CP)

Wide-Column Stores

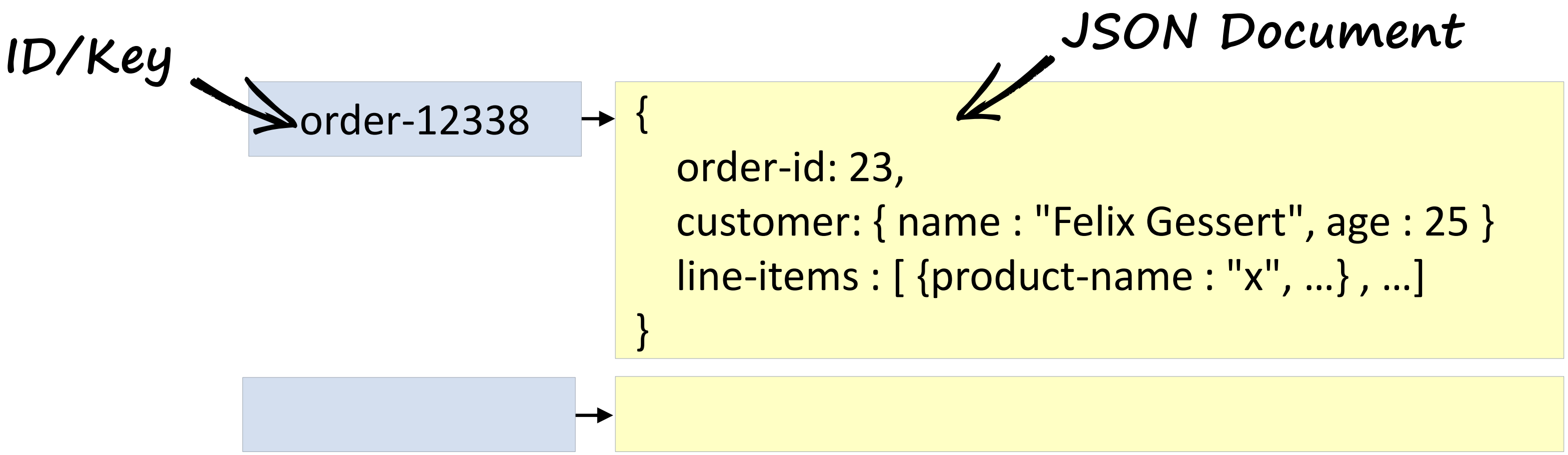
- **Data model:** (rowkey, column, timestamp) -> value
- **Interface**: CRUD, Scan



- Examples: Cassandra (AP), Google BigTable (CP), HBase (CP)

Document Stores

- **Data model:** (collection, key) -> document
- **Interface**: CRUD, Querys, Map-Reduce



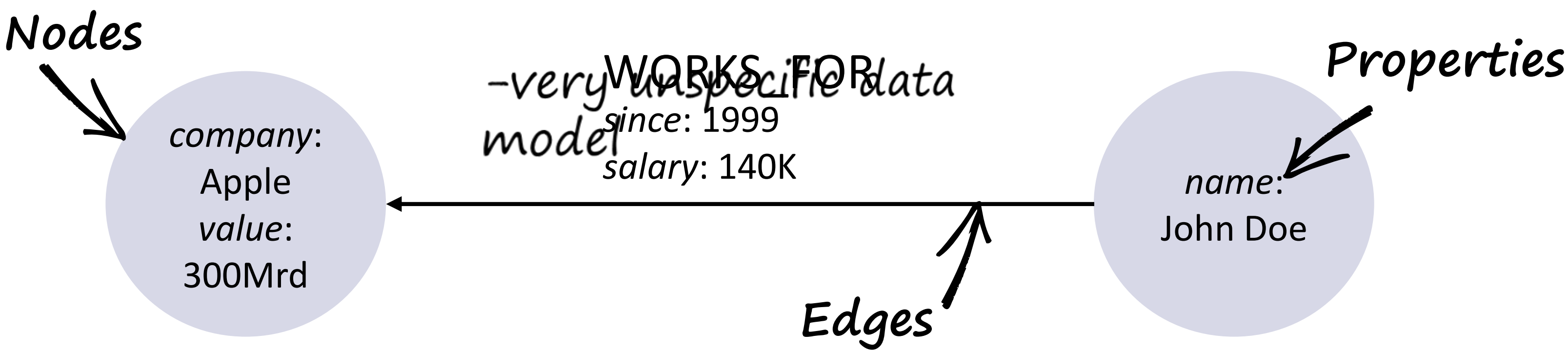
- Examples: CouchDB (AP), Amazon SimpleDB (AP), MongoDB (CP)

Graph Databases

- **Data model:** G = (V, E): Graph-Property Modell
- **Interface**: Traversal algorithms, querys, transactions

-usually unscalable

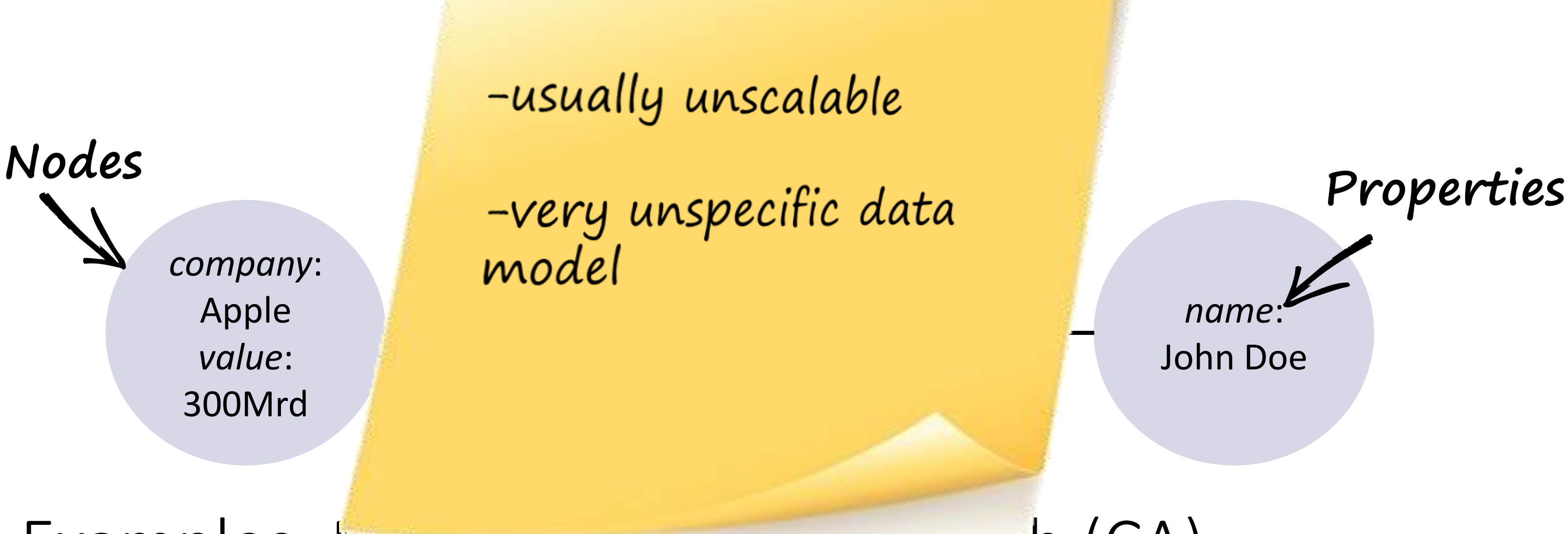
-very unspecific data model



- Examples: Neo4j (CA), InfiniteGraph (CA)

Graph Databases

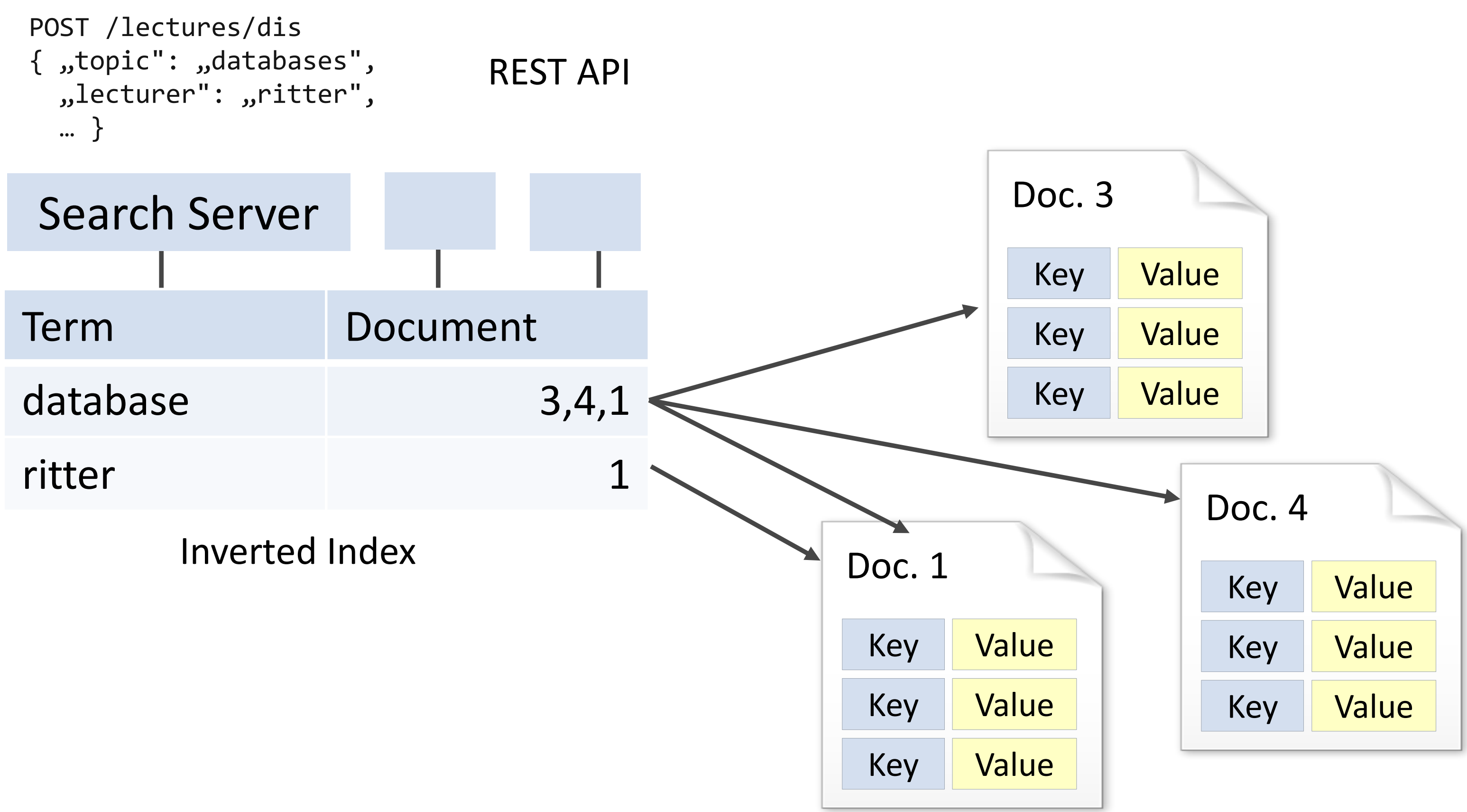
- **Data model:** G = (V, E): Graph-Property Modell
- **Interface**: Trave … , transactions



- Examples: Neo4J (CA), … (CA)

Search Platforms

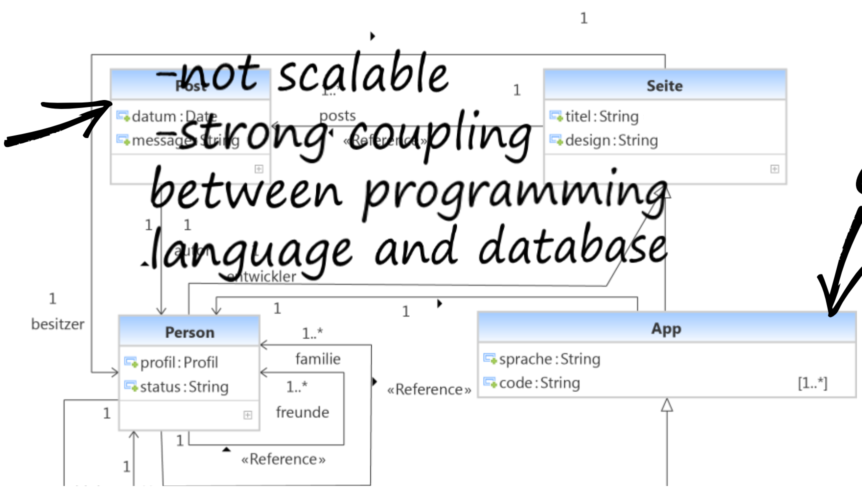
- **Data model:** vectorspace model, docs + metadata
- Examples: Solr, ElasticSearch



Object-oriented Databases

- **Data model:** Classes, objects, relations (references)
- **Interface**: CRUD, querys, transactions

Properties



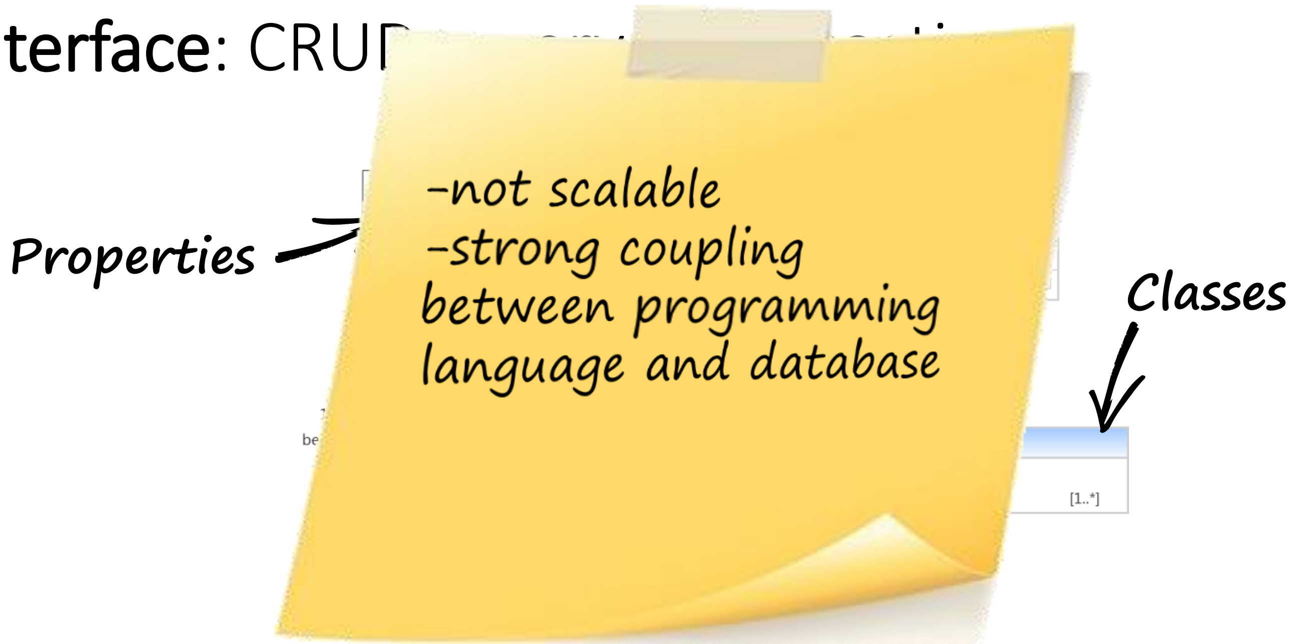
-not scalable
-strong coupling between programming language and database

Classes

- Examples: Versant (CA), db4o (CA), Objectivity (CA)

Object-oriented Databases

- **Data model:** Classes, objects, relations (references)
- **Interface**: CRUD



- Examples: Versant (CA), db4o (CA), Objectivity (CA)

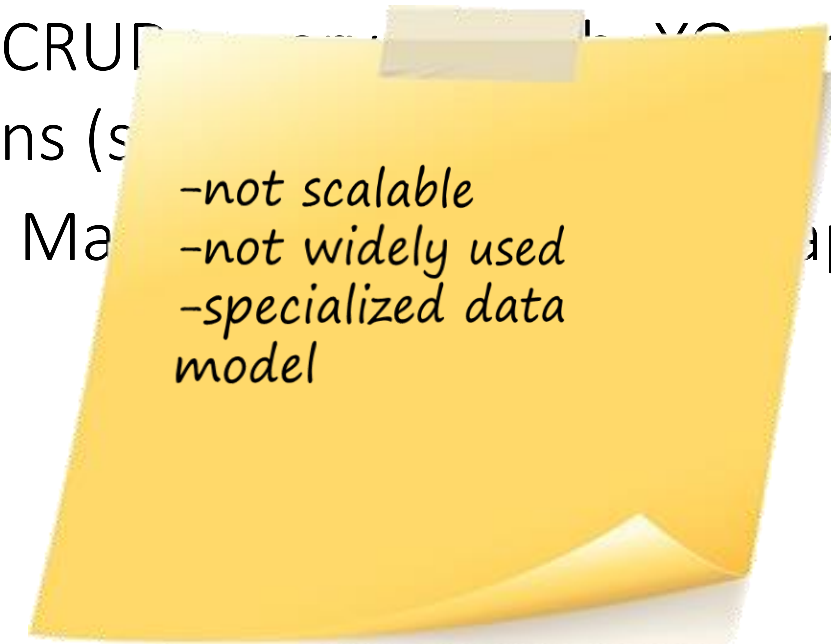
XML databases, RDF Stores

- **Data model:** XML, RDF
- **Interface**: CRUD, querys (XPath, XQuerys, SPARQL), transactions (some)
- Examples: MarkLogic (CA), AllegroGraph (CA)

-not scalable
-not widely used
-specialized data model

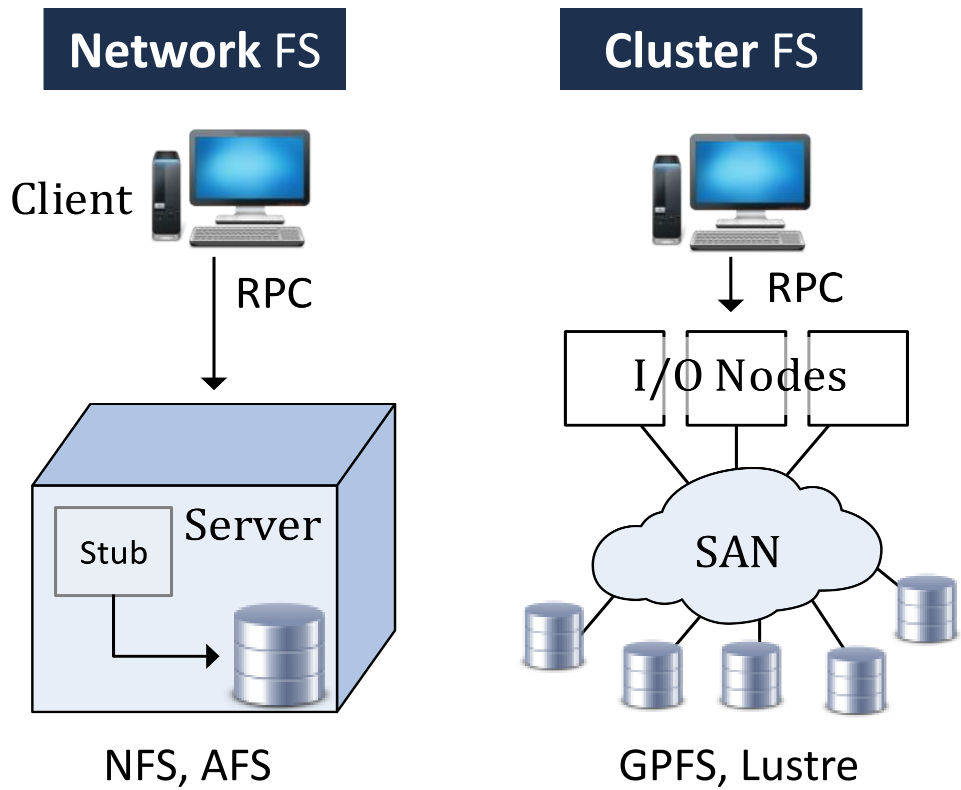
XML databases, RDF Stores

- **Data model:** XML, RDF
- **Interface**: CRUD … rys, SPARQL), transactions (s…
- Examples: Ma… aph (CA)



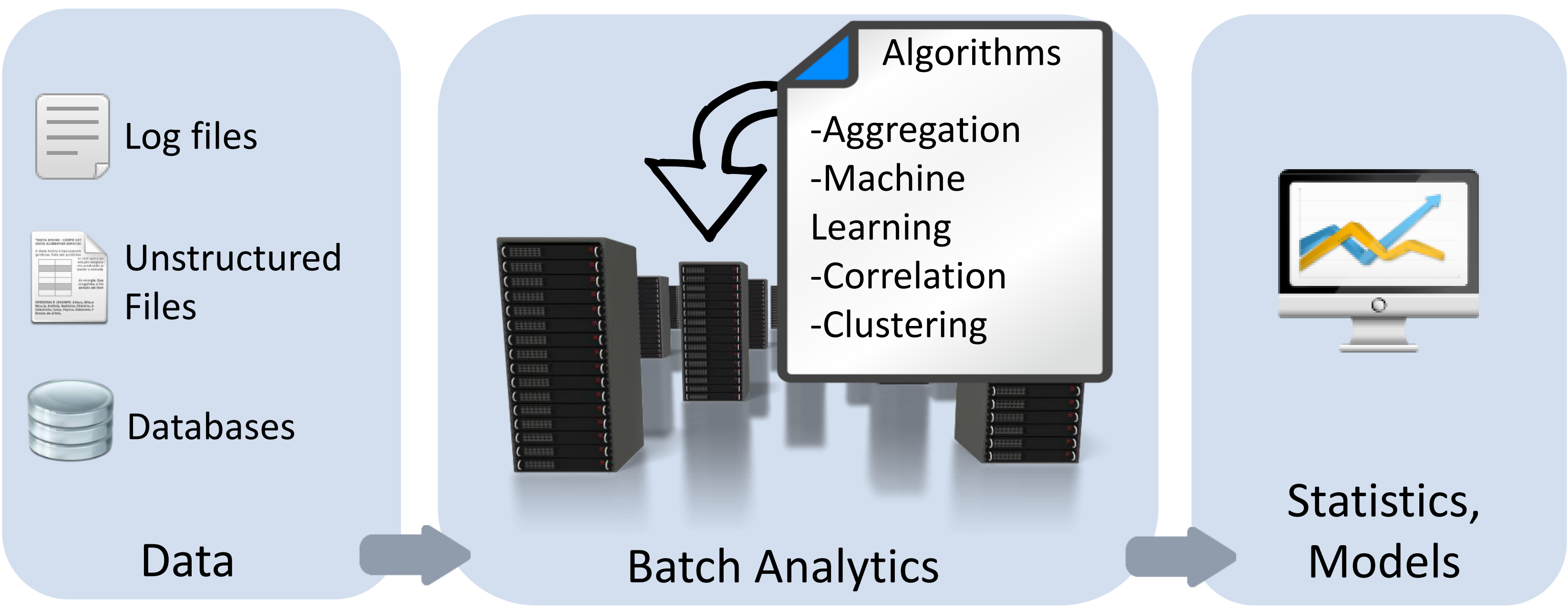
Distributed File System

- **Data model:** files + folders



Big Data Frameworks

- **Data model:** arbitrary (frequently unstructured)
- Examples: Hadoop, Spark, Flink, DryadLink, Pregel



Wrap-up

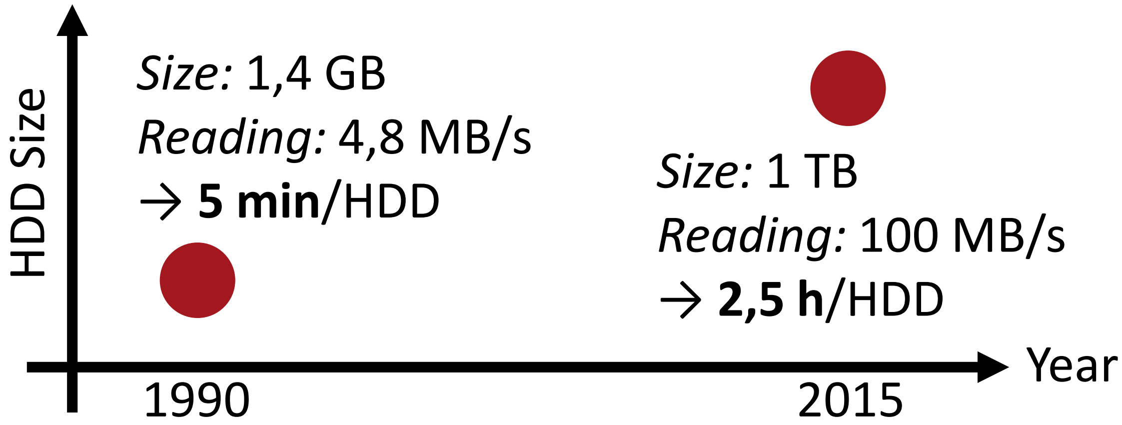
- 4 core NoSQL classes
  - **Key-Value** Stores: store opaque key-value pairs
  - **Document** Stores: store nested, rich, schema-free documents
  - **Wide-Column** Stores: extensible table data model
  - **Graph Databases**: graph-property-model (vertices and edges)
- Other NoSQL-related systems: Object-oriented databases, Search platforms, XML databases, Big Data Frameworks, Distributed File Systems

Outline

- Foundations: Big Data, Scalability, Avaialbility
- The 4 Classes of NoSQL Databases
- NoSQL Examples: concrete Architectures, Systems, APIs
- Cloud Databases

- MapReduce (Hadoop)
- Dynamo (Riak)
- BigTable (HBase)
- MongoDB
- Others

Hadoop Distributed FS (CP)

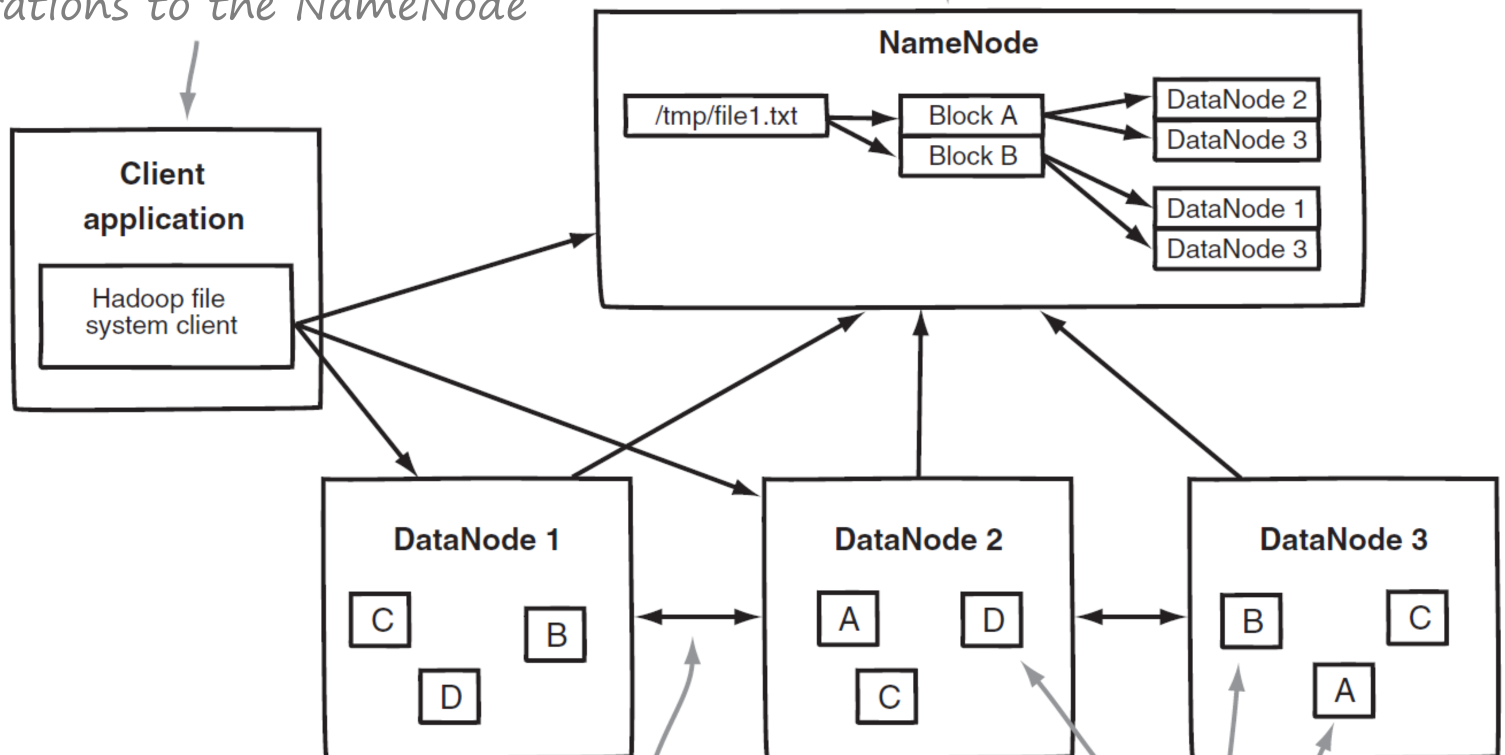


HDFS
Model:
File System
License:
Apache 2
Written in:
Java

- Modelled after: Googles GFS (2003)
- **Master-Slave** Replication
  - **Namenode**: Metadata (files + block locations)
  - **Datanodes**: Save file blocks (usually 64 MB)
- **Design goal**: Maximum Throughput and data locality for Map-Reduce

Sends data operations to DataNodes and metadata operations to the NameNode

Holds filesystem data and block locations in RAM



DataNodes communicate to perform 3-way replication

Files are split into blocks and scattered over DataNodes

Holmes, Alex. Hadoop in Practice. Manning, 2012.

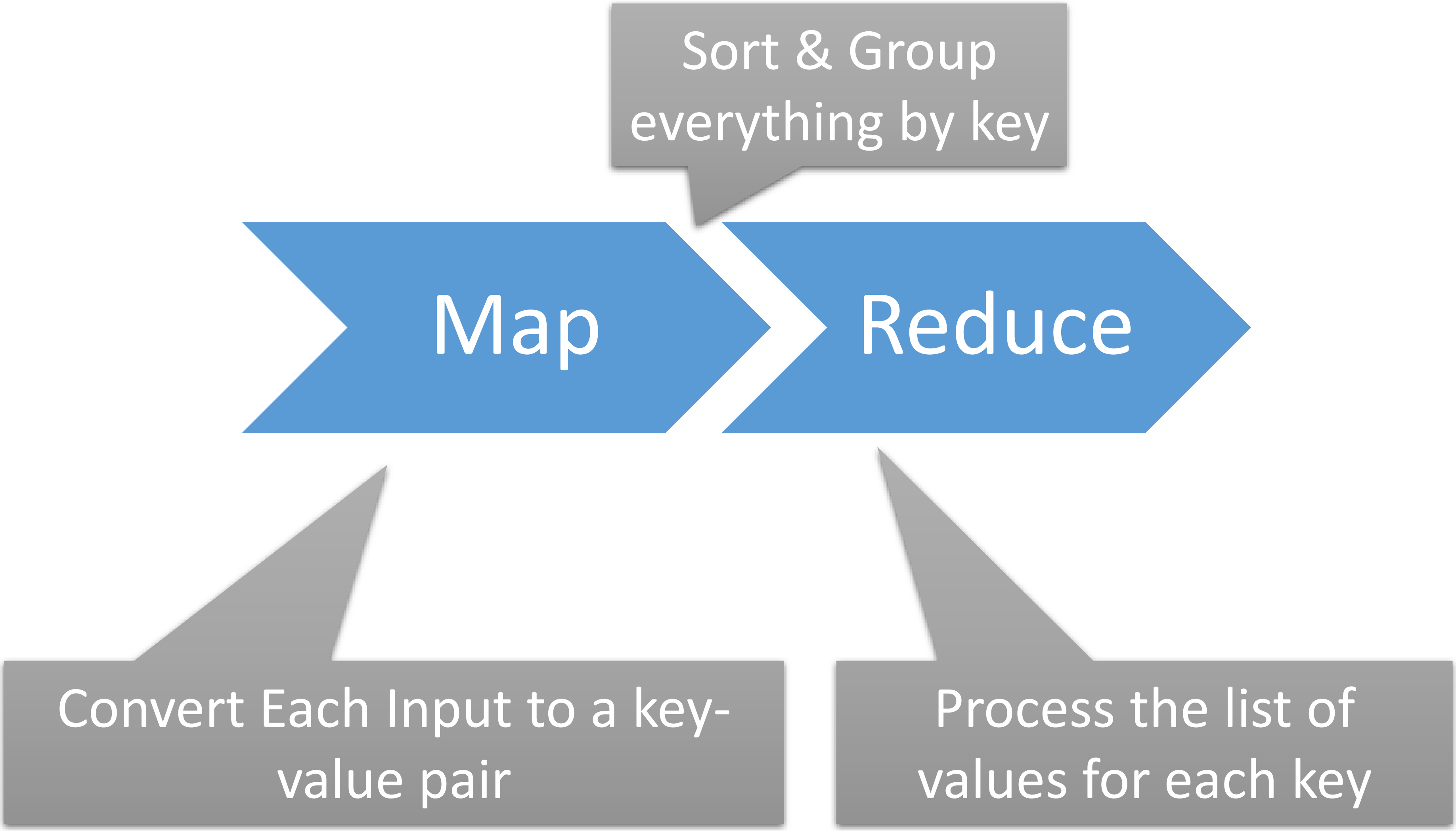
Hadoop

- For many synonymous to *Big Data Analytics*
- Large Ecosystem
- Creator: Doug Cutting (Lucene)
- Distributors: Cloudera, MapR, HortonWorks
- Gartner Prognosis: By 2015 65% of all complex analytic applications will be based on Hadoop
- Users: Facebook, Ebay, Amazon, IBM, Apple, Microsoft, NSA

Hadoop
Model:
Batch-Analytics Framework
License:
Apache 2
Written in:
Java

http://de.slideshare.net/cultureofperformance/gartner-predictions-for-hadoop-predictions

MapReduce


Sort & Group everything by key
Map
Reduce
Convert Each Input to a key-value pair
Process the list of values for each key

MapReduce: Word Count

```
map(zeilennr, text):
   for each word in text:
      emit(word, 1)

reduce(word, values):
   sum = 0
   for each v in values:
      sum = sum + v
   emit(word, sum)
```

1: ich bin ich

(ich, 1) (bin, 1) (ich, 1)

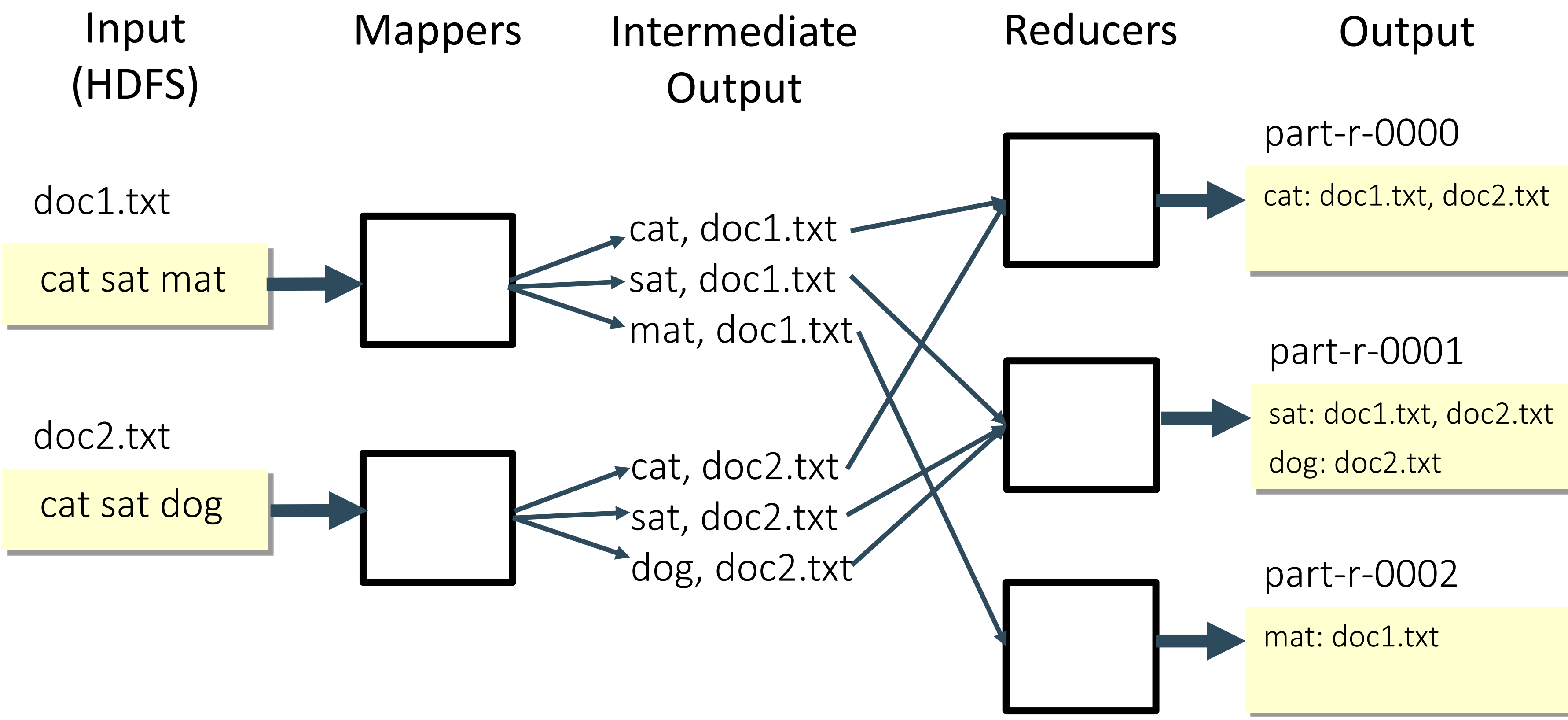
(ich, [1,1])

(bin, [1])

(ich, 2) (bin, 1)

MapReduce: Example

Constructing a reverse-index

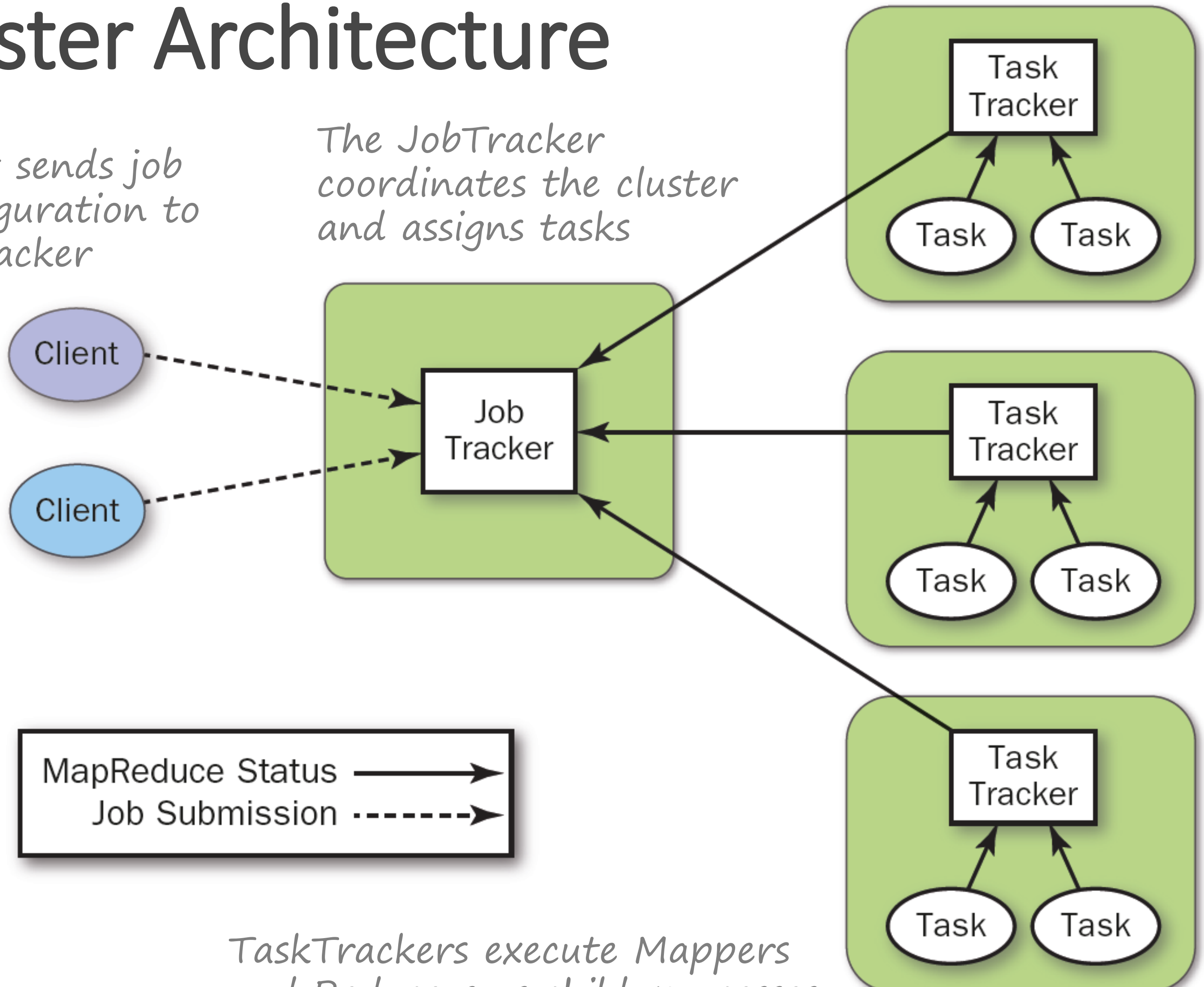


Holmes, Alex. Hadoop in Practice

Cluster Architecture

The client sends job and configuration to the Jobtracker

The JobTracker coordinates the cluster and assigns tasks



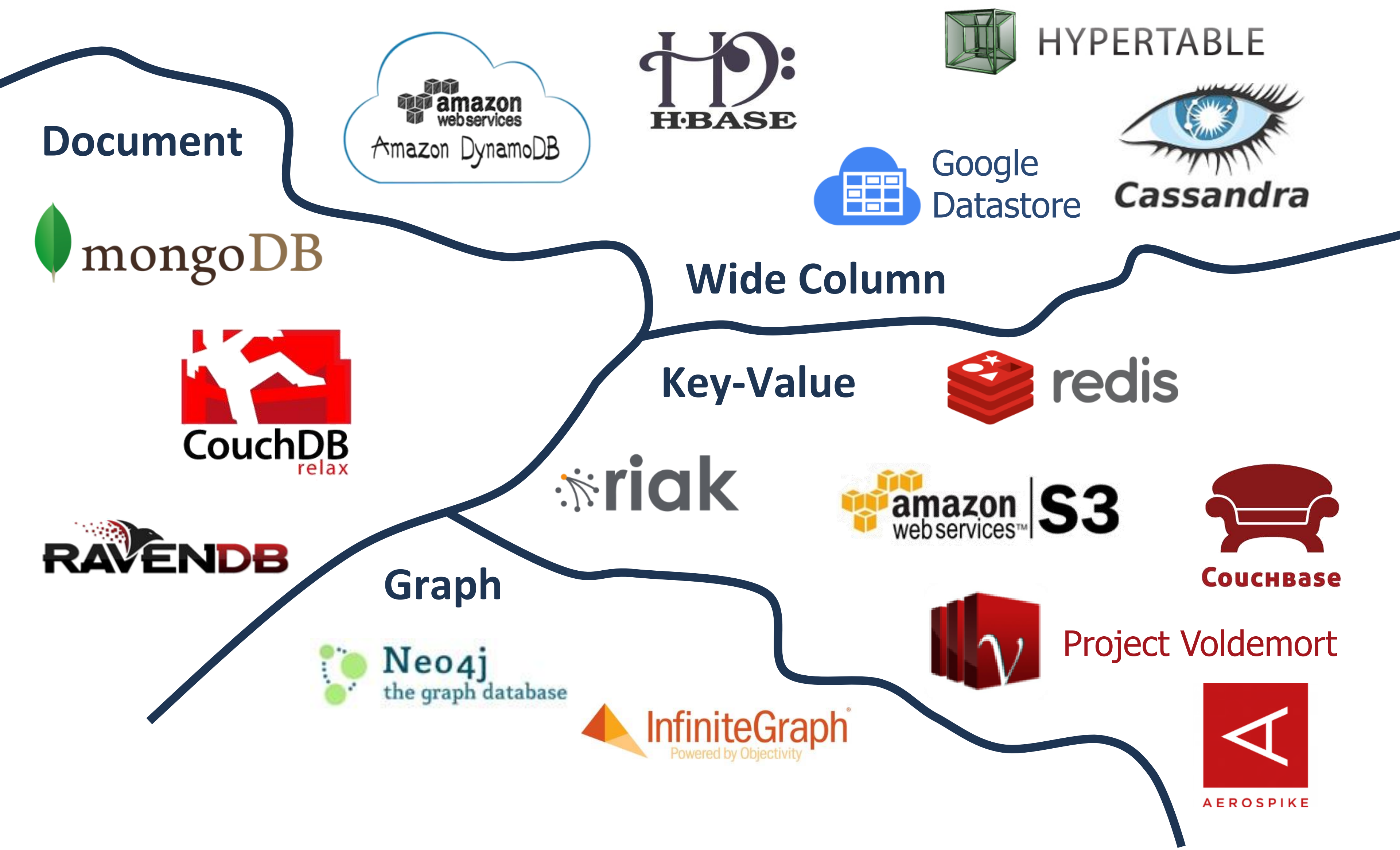
TaskTrackers execute Mappers and Reducers as child-processes

Arun Murthy “Apache Hadoop YARN”

Summary: Hadoop Ecosystem

- **Hadoop**: Ecosystem for Big Data Analytics
- **Hadoop Distributed File System**: scalable, shared-nothing file system for thoughput-oriented workloads
- **Map-Reduce**: Paradigm for performing scalable distributed batch analysis
- 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

NoSQL landscape



Popularity

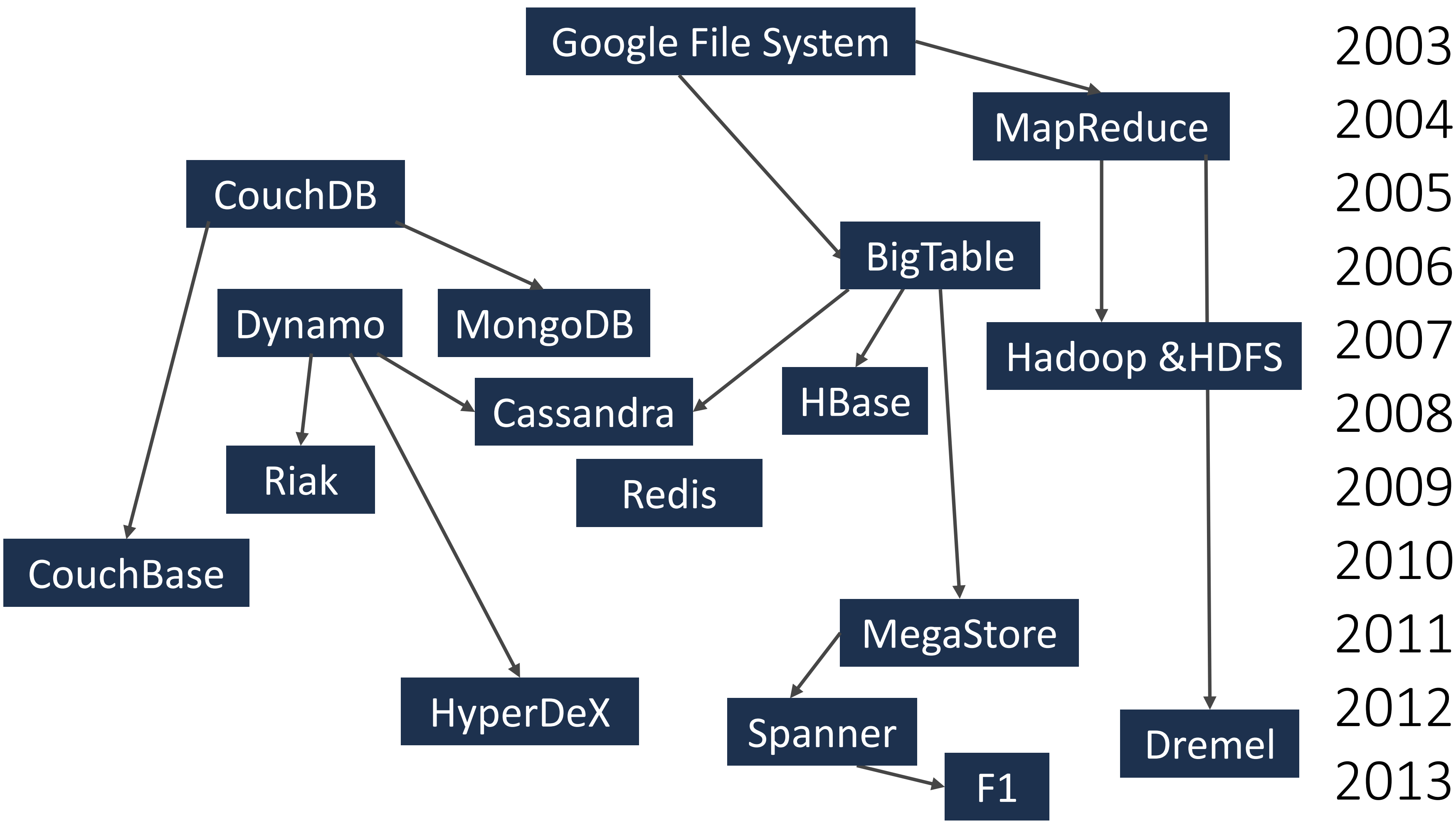
http://db-engines.com/de/ranking

Rang	DBMS	Modell	Punkte
1.	Oracle	Relational DBMS	1514,90
2.	MySQL	Relational DBMS	1334,94
3.	Microsoft SQL Server	Relational DBMS	1286,22
4.	PostgreSQL	Relational DBMS	199,39
5.	DB2	Relational DBMS	177,04
6.	Microsoft Access	Relational DBMS	149,66
7.	**MongoDB**	**Document Store**	**137,49**
8.	Sybase	Relational DBMS	88,41
9.	SQLite	Relational DBMS	87,81
10.	Teradata	Relational DBMS	51,11
11.	Solr	Suchmaschine	46,43
12.	**Cassandra**	**Wide Column Store**	**37,64**
13.	**Redis**	**Key-Value Store**	**34,22**

Rang	DBMS	Modell	Punkte
14.	**Memcached**	**Key-Value Store**	**30,73**
15.	**HBase**	**Wide Column Store**	**25,78**
16.	Informix	Relational DBMS	24,73
17.	Hive	Relational DBMS	22,16
18.	**CouchDB**	**Document Store**	**15,93**
19.	Firebird	Relational DBMS	14,55
20.	Netezza	Relational DBMS	11,44
21.	dBASE	Relational DBMS	10,44
22.	Elasticsearch	Suchmaschine	9,51
23.	Sphinx	Suchmaschine	9,02
24.	**Riak**	**Key-Value Store**	**8,99**
25.	**Neo4j**	**Graph DBMS**	**8,83**

Scoring: Google/Bing results, Google Trends, Stackoverflow, job offers, LinkedIn

History


Google File System
MapReduce
CouchDB
BigTable
Dynamo
MongoDB
Hadoop &HDFS
Cassandra
HBase
Riak
Redis
CouchBase
MegaStore
HyperDeX
Spanner
F1
Dremel
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013

NoSQL foundations

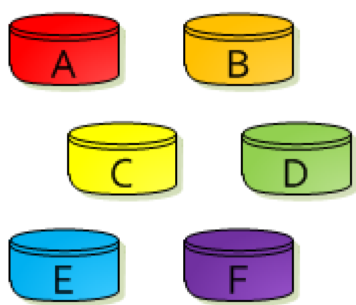
- **BigTable** (2006, Google)
  - **C**onsistent, **P**artition Tolerant
  - Wide-Column data model
  - Master-based, fault-tolerant, large clusters (1.000+ Nodes), **HBase**, **Cassandra**, **HyperTable, Accumolo**
- **Dynamo** (2007, Amazon)
  - **A**vailable, **P**artition tolerant
  - Key-Value interface
  - Eventually Consistent, always writable, fault-tolerant
  - **Riak**, **Cassandra**, **Voldemort**, **DynamoDB**

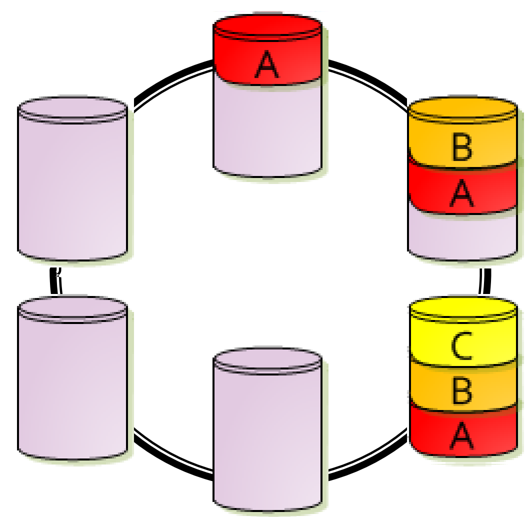
Chang, Fay, et al. "Bigtable: A distributed storage system for structured data."

DeCandia, Giuseppe, et al. "Dynamo: Amazon's highly available key-value store."

Dynamo (AP)

- Developed at Amazon (2007)
- Sharding of data over a ring of nodes
- Each node holds multiple partitions
- Each partition **N**-times replicated

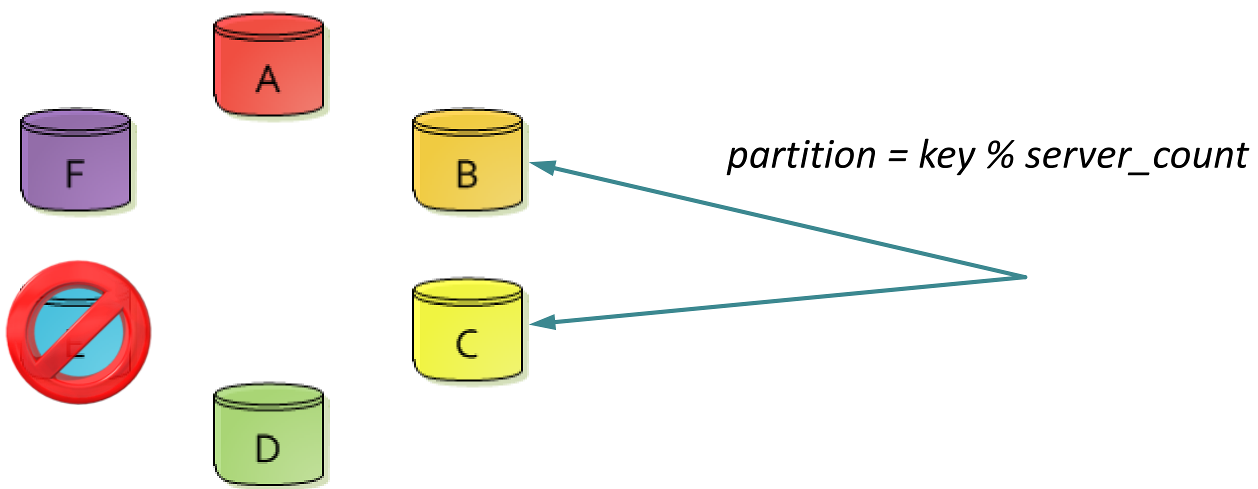




DeCandia, Giuseppe, et al. "Dynamo: Amazon's highly available key-value store."

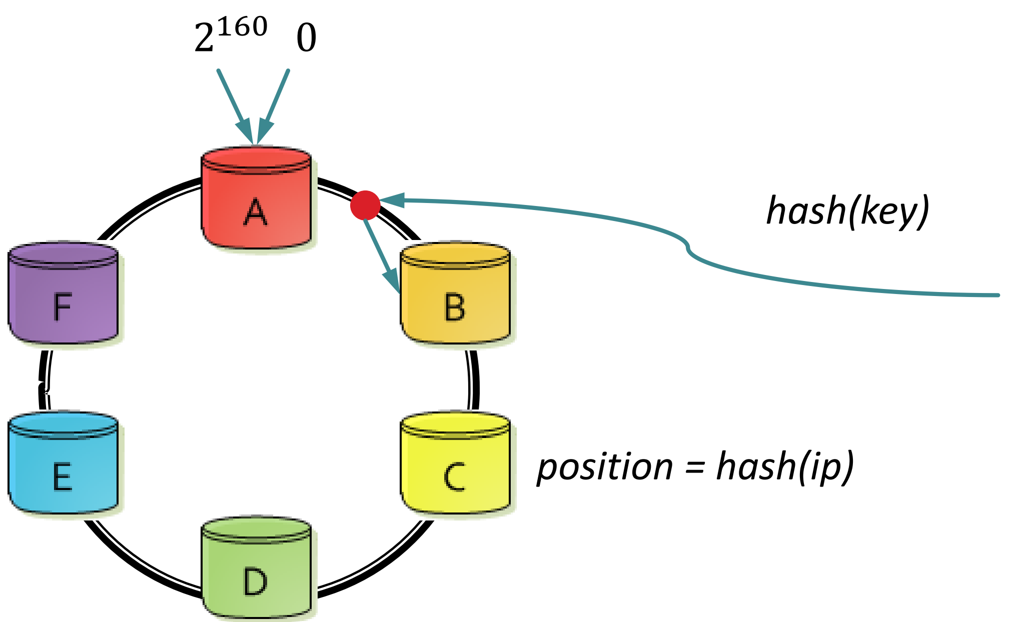
Consistent Hashing

- Naive approach: **Hash-partitioning** (e.g. in Memcache, Redis)



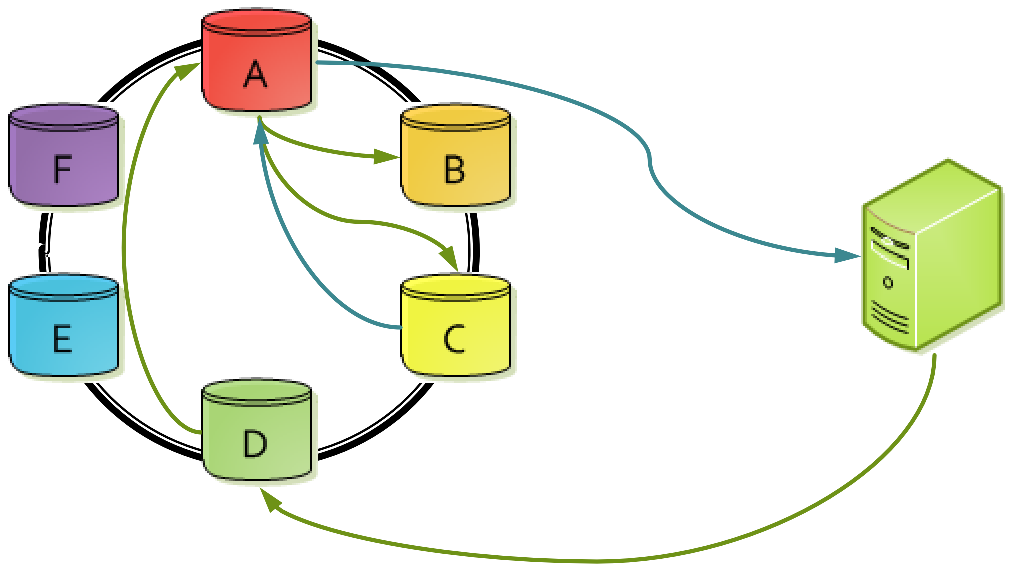
Consistent Hashing

- Solution: **Consistent Hashing** – mapping of data to nodes is stable under topology changes



Reading and Writing

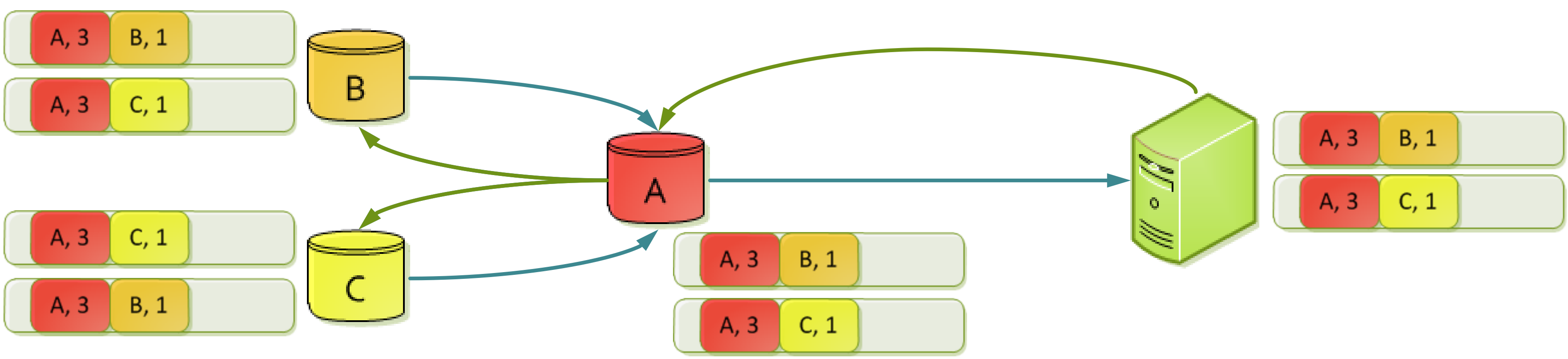
- An arbitrary node acts as a coordinator
- **N**: number of replicas
- **R**: number of nodes that need to confirm a read
- **W**: number of nodes that need to confirm a write



N=3
R=2
W=1

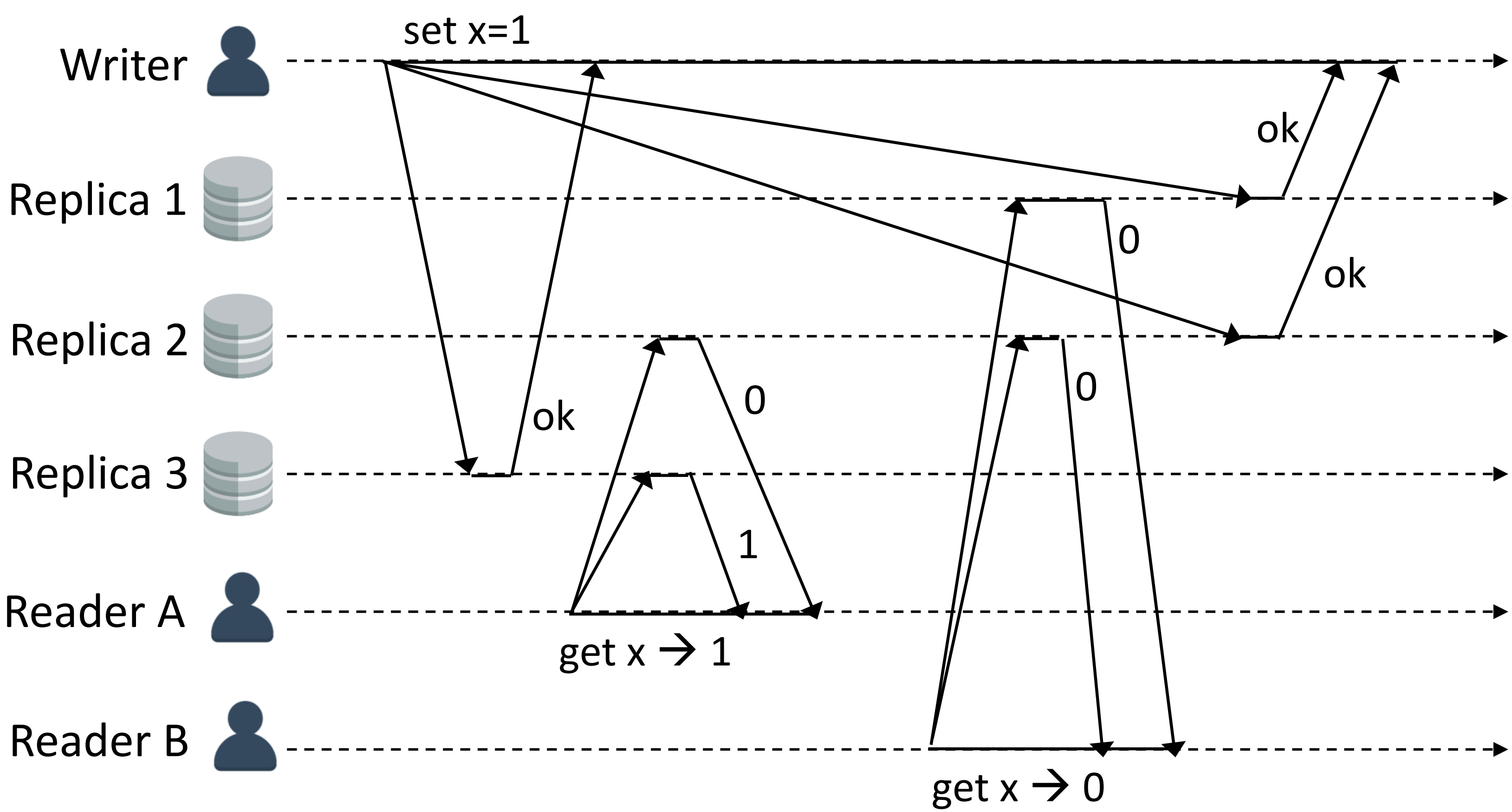
Versioning and Consistency

- $R + W \leq N \Rightarrow$ no consistency guarantee
- $R + W > N \Rightarrow$ newest value included in any read
- **Vector Clocks** used for versioning



$R + W > N$ does not imply linearizability

- Consider the following execution:

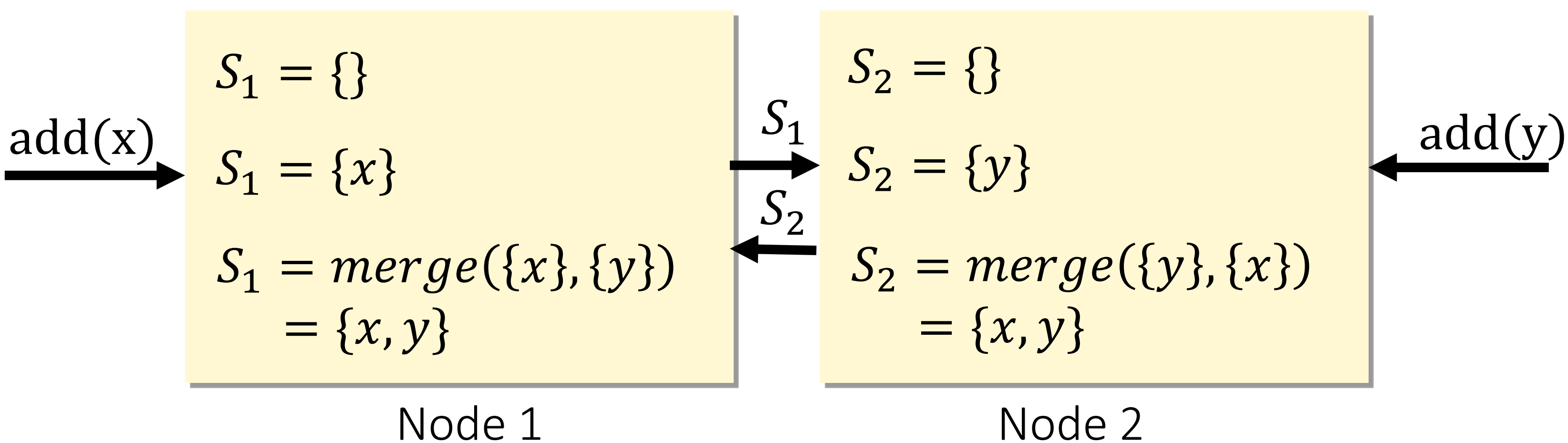


Kleppmann, Martin. "Designing data-intensive applications." (2015).

CRDTs

Convergent/Commutative Replicated Data Types

- **Goal**: avoid manual conflict-resolution
- **Approach**:
  - **State-based** – commutative, idempotent merge function
  - **Operation-based** – broadcasts of commutative upates
- Example: State-based Grow-only-Set (G-Set)

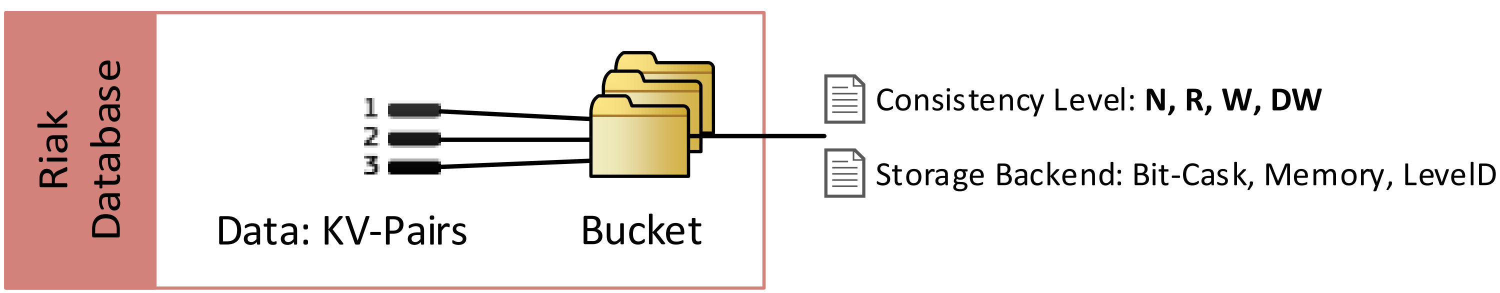


Marc Shapiro, Nuno Preguica, Carlos Baquero, and Marek Zawirski "Conflict-free Replicated Data Types"

Riak (AP)

Riak
Model:
Key-Value
License:
Apache 2
Written in:
Erlang und C

- Open-Source Dynamo-Implementation
- Extends Dynamo:
  - Keys are grouped to **Buckets**
  - KV-pairs may have **metadata** and **links**
  - Map-Reduce support
  - Secondary Indices, Update Hooks, Solr Integration
  - **REST-API**



Summary: Dynamo and Riak

- **Consistent Hashing**: hash-based distribution with stability under topology changes (e.g. machine failures)
- Parameters: **N** (Replicas), **R** (Read Acks), **W** (Write Acks)
  - N=3, R=W=1 → fast, potentially inconsistent
  - N=3, R=3, W=1 → slower reads, most recent object version contained
- Available and Partition-Tolerant
- **Vector Clocks**: concurrent modification can be detected, inconsistencies are healed through the application
- **API**: Create, Read, Update, Delete (CRUD) on key-value pairs
- **Riak**: Open-Source Implementation of the Dynamo paper

Redis (CA)

Redis
Model:
Key-Value
License:
BSD
Written in:
C

- **Re**mote **Di**ctionary **S**erver
- In-Memory Key-Value Store
- Asynchronous Master-Slave Replication
- Data model: rich data structures stored under key
- **Tunable persistence**: logging and snapshots
- Single-threaded event-loop design (similar to Node.js)
- Optimistic batch transactions (*Multi blocks*)
- Very high performance: >100k ops/sec on one machine
- Redis Cluster adds sharding

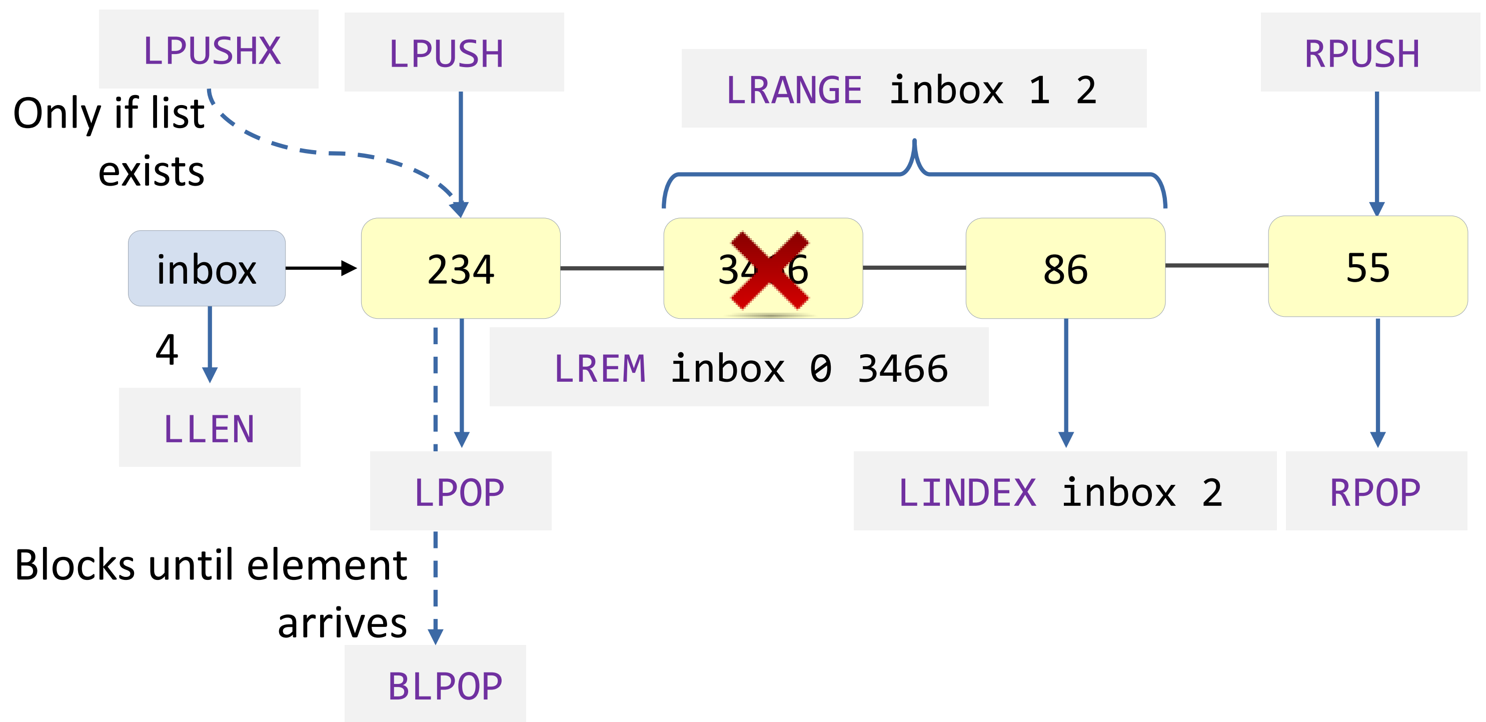
Data structures

- String, List, Set, Hash, Sorted Set

String	web:index	"<html><head>…"
Set	users:2:friends	{23, 76, 233, 11}
List	users:2:inbox	[234, 3466, 86,55]
Hash	users:2:settings	Theme → "dark", cookies → "false"
Sorted Set	top-posters	466 → "2", 344 → "16"
Pub/Sub	users:2:notifs	"{event: 'comment posted', time : …"

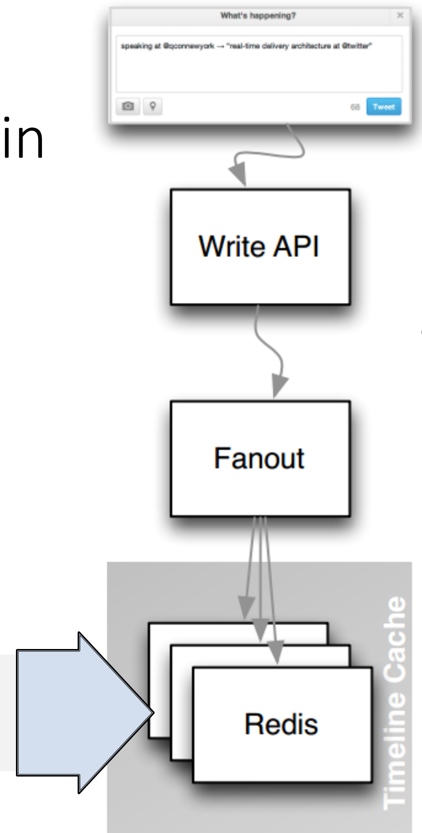
Example Redis Data Structure: lists

- (Linked) Lists:



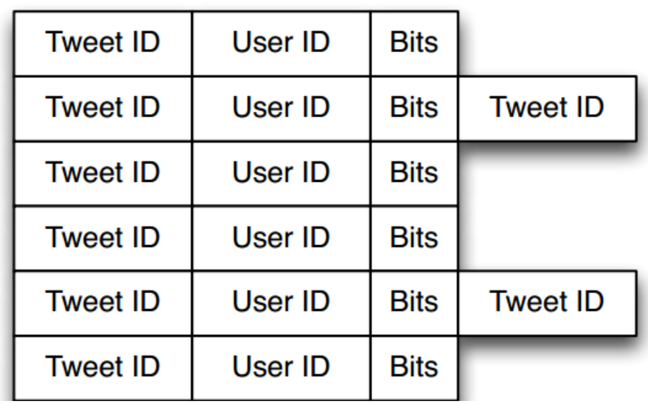
Example Redis Use-Case: Twitter

- Per User: one materialized timeline in Redis
- Timeline = List
- Key: User ID



>150 million users
~300k timeline querys/s

```
RPUSHX user_id tweet
```



http://www.infoq.com/presentations/Real-Time-Delivery-Twitter

Classification: Redis

Techniques

Sharding	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared Disk
Replication	Trans-action Protocol	Sync. Replica-tion	Async. Replica-tion	Primary Copy	Update Anywhere
Storage Management	Logging	Update-in-Place	Caching	In-Memory	Append-Only Storage
Query Processing	Global Index	Local Index	Query Planning	Analytics	Materialized Views

Google BigTable (CP)

- Published by Google in 2006
- Original purpose: storing the Google search index

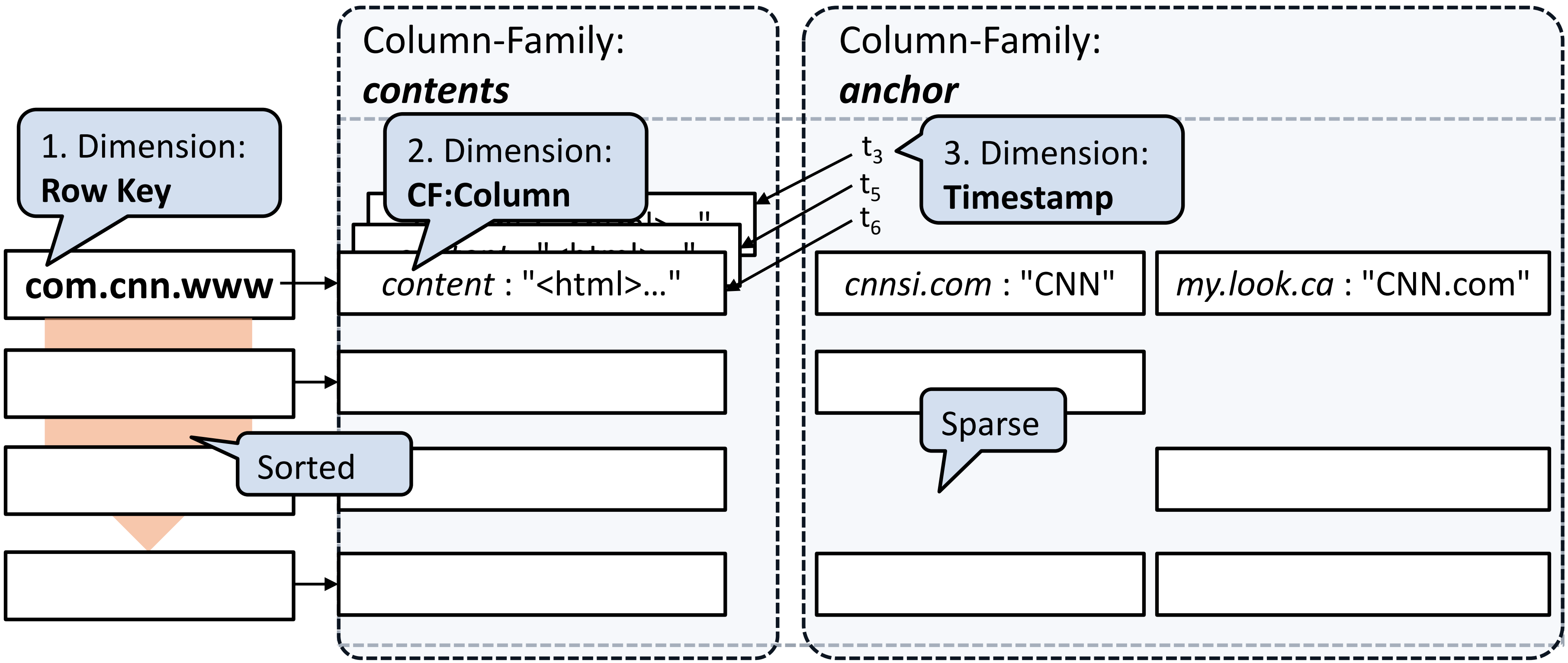
> A Bigtable is a sparse, distributed, persistent multidimensional sorted map.

- Data model also used in: **HBase**, **Cassandra**, **HyperTable, Accumolo**

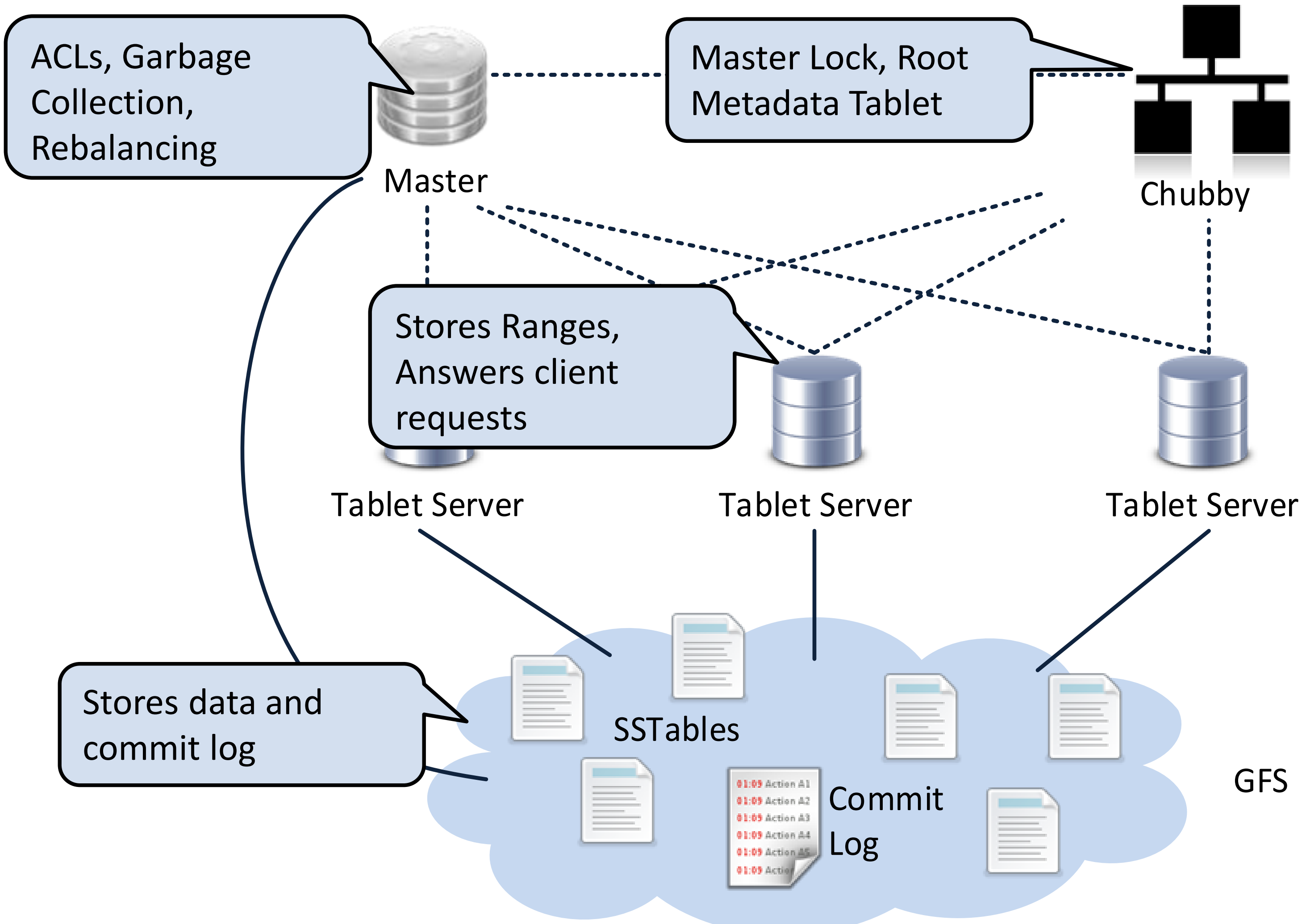
Chang, Fay, et al. "Bigtable: A distributed storage system for structured data."

Wide-Column Data Modelling

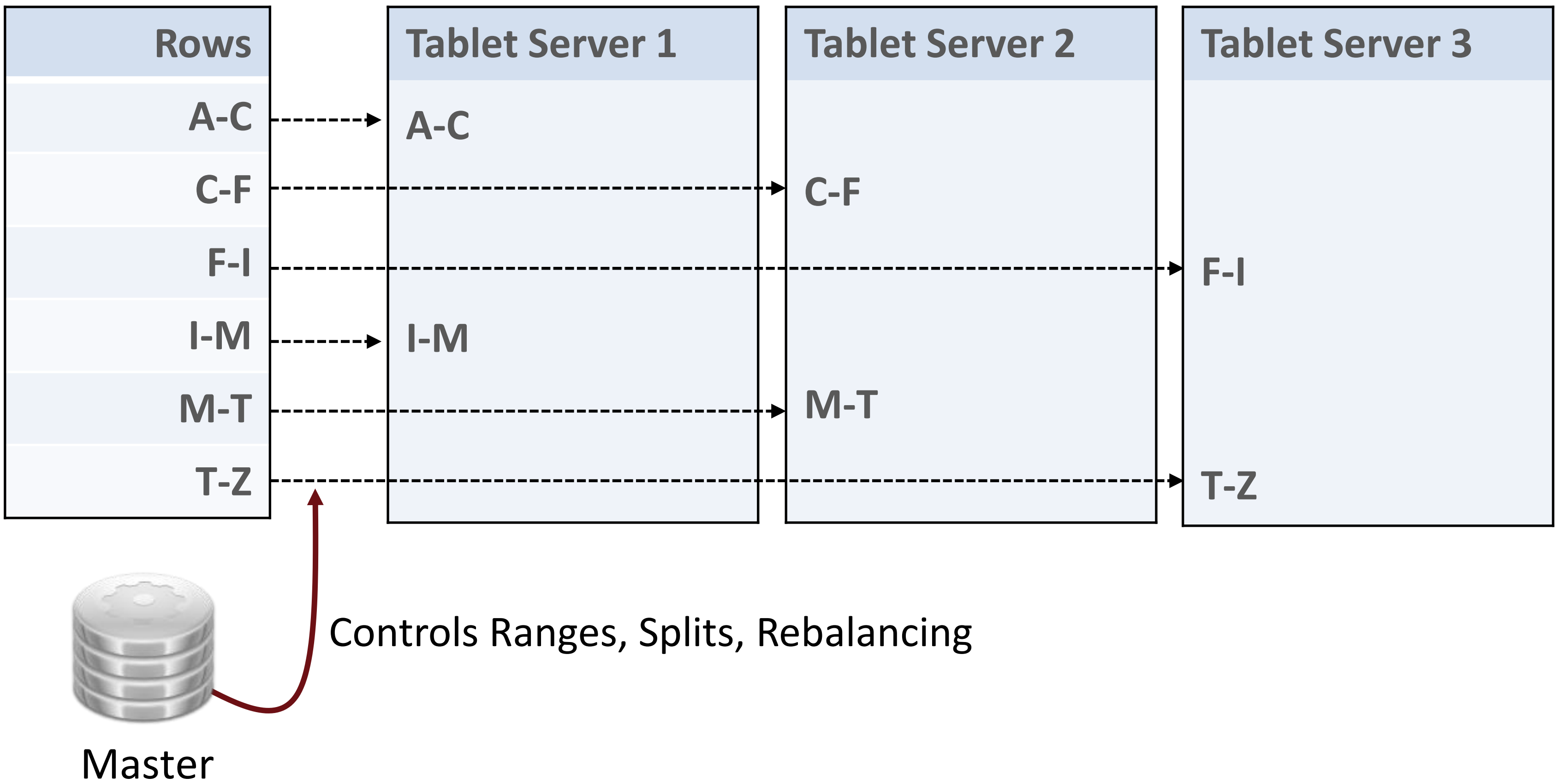
- Storage of crawled web-sites („Webtable"):



Architecture

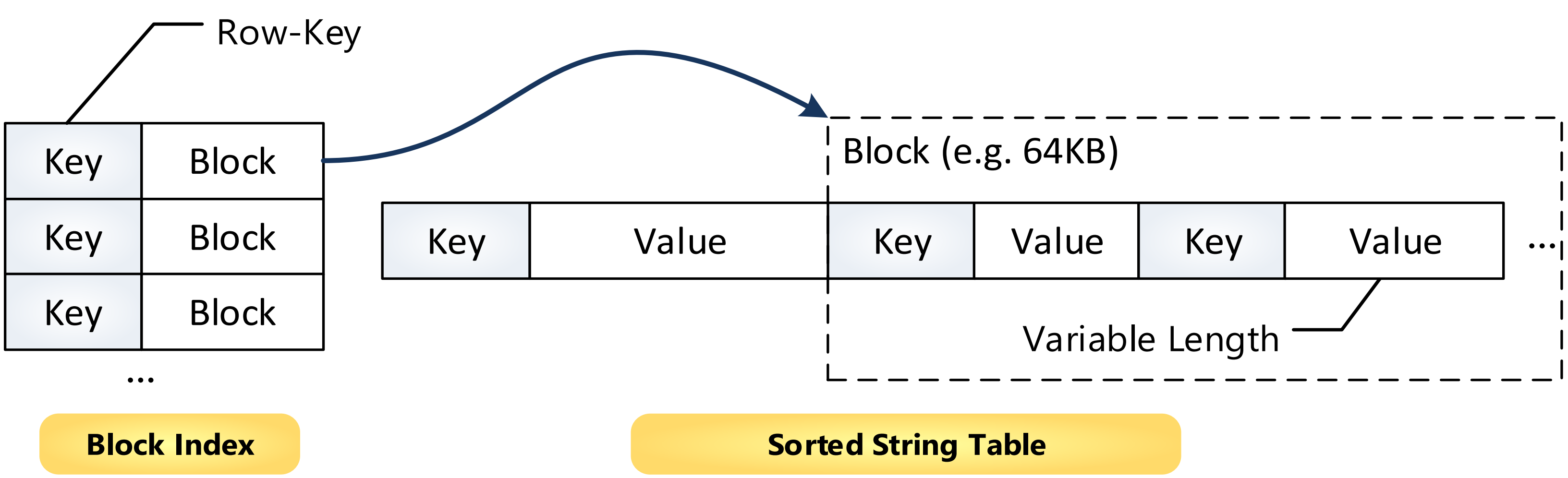


Range-based Sharding



Storage: Sorted-String Tables

- **Goal**: Append-Only IO when writing (no disk seeks)
- Achieved through: **Log-Structured Merge Trees**
- **Writes** go to an in-memory *memtable* that is periodically persisted as an *SSTable* as well as a *commit log*
- **Reads** query memtable and all SSTables



Apache HBase (CP)

HBase
Model:
Wide-Column
License:
Apache 2
Written in:
Java

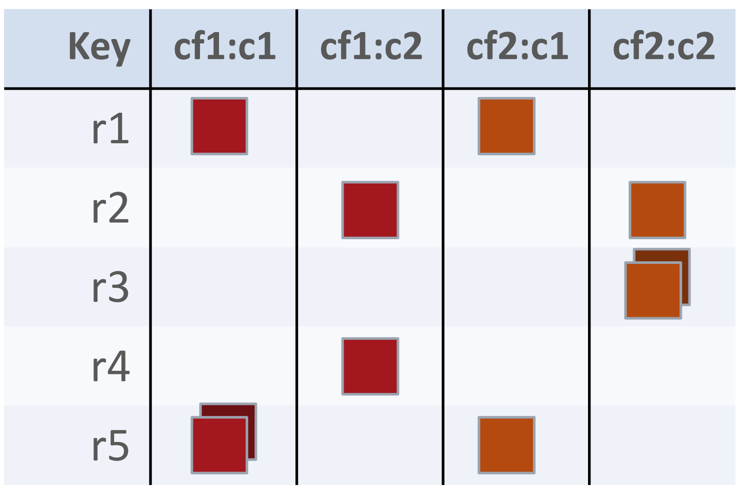
- Open-Source Implementation of BigTable
- Hadoop-Integration
  - Data source for Map-Reduce
  - Uses Zookeeper and HDFS
- Data modelling challenges: key design, tall vs wide
  - **Row Key**: only access key (no indices) → key design important
  - **Tall**: good for scans
  - **Wide**: good for gets, consistent (*single-row atomicity*)
- No typing: application handles serialization
- Interface: REST, Avro, Thrift

HBase Storage

- Logical to physical mapping:

In Value	r2:cf2:c2:t1:<value>
In Key	r2-<value>:cf2:c2:t1:_
In Column	r2:cf2:c2<value>:t1:_

Key Design – where to store data:
r2:cf2:c2:t1:<value>
r2-<value>:cf2:c2:t1:_
r2:cf2:c2<value>:t1:_



r1:cf2:c1:t1:<value>
r2:cf2:c2:t1:<value>
r3:cf2:c2:t2:<value>
r3:cf2:c2:t1:<value>
r5:cf2:c1:t1:<value>

HFile cf2

r1:cf1:c1:t1:<value>
r2:cf1:c2:t1:<value>
r3:cf1:c2:t1:<value>
r3:cf1:c1:t2:<value>
r5:cf1:c1:t1:<value>

HFile cf1

George, Lars. HBase: the definitive guide. 2011.

Example: Facebook Insights

MD5(Reversed Domain) + Reversed Domain + URL-ID — Row Key

6PM Total	6PM Male	…	01.01 Total	01.01 Male	…	Total	Male	…
10	7		100	65			567	

CF:Daily — CF:Monthly — CF:All

Atomic HBase **Counter**

TTL – automatic deletion of old rows

Lars George: “Advanced HBase Schema Design”

Summary: BigTable, HBase

- Data model: $(rowkey, cf : column, timestamp) \rightarrow value$
- **API**: CRUD + Scan(*start-key*, *end-key*)
- Uses distributed file system (GFS/HDFS)
- Storage structure: **Memtable** (in-memory data structure) + **SSTable** (persistent; append-only-IO)
- **Schema design**: only primary key access → implicit schema (key design) needs to be carefully planned
- **HBase**: very literal open-source implementation BigTable
- **Cassandra**: combination of Dynamo and BigTable ideas

Classification: HBase

Techniques

Sharding	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared Disk
Replication	Trans-action Protocol	Sync. Replica-tion	Async. Replica-tion	Primary Copy	Update Anywhere
Storage Management	Logging	Update-in-Place	Caching	In-Memory	Append-Only Storage
Query Processing	Global Index	Local Index	Query Planning	Analytics	Materialized Views

Apache Cassandra (AP)

- Published 2007 by Facebook
- **Idea**:
  - BigTable‘s wide-column data model
  - Dynamo ring for replication and sharding
- Cassandra Query Language (CQL): SQL-like query- and DDL-language
- **Compound indices**: *partition key* (shard key) + *clustering key* (ordered per partition key) → Limited range queries
- **Secondary indices**: hidden table with mapping → queries with simple equality condition

Cassandra
Model:
Wide-Column
License:
Apache 2
Written in:
Java

Classification: Cassandra

Techniques

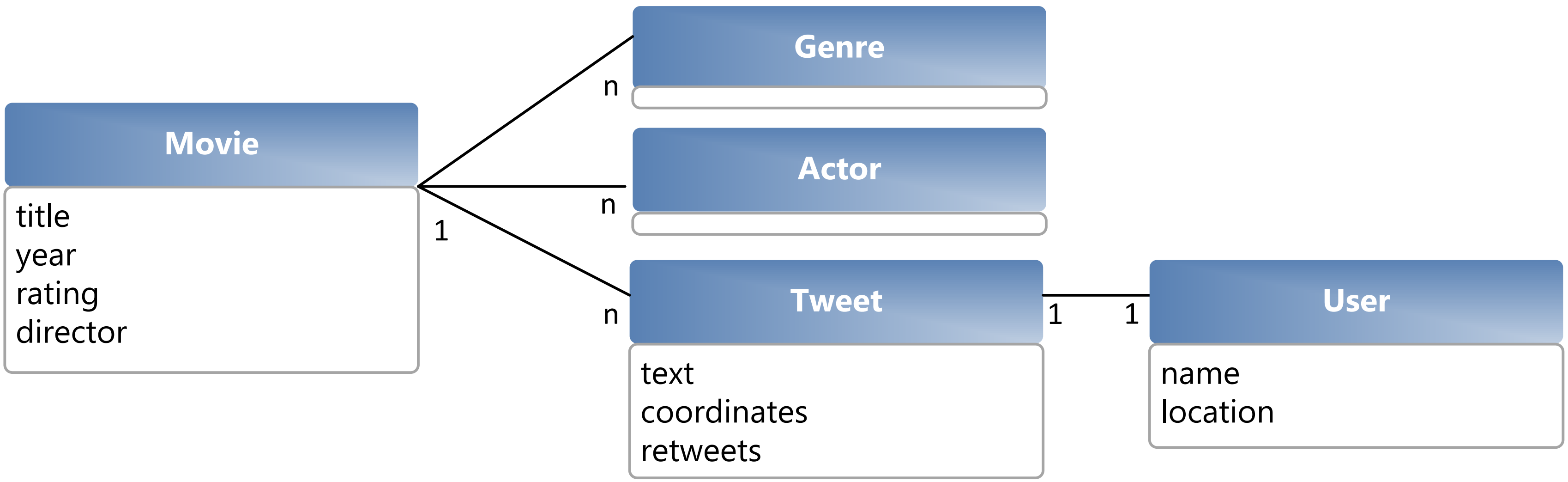
Sharding	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared Disk
Replication	Trans-action Protocol	Sync. Replica-tion	Async. Replica-tion	Primary Copy	Update Anywhere
Storage Management	Logging	Update-in-Place	Caching	In-Memory	Append-Only Storage
Query Processing	Global Index	Local Index	Query Planning	Analytics	Materialized Views

MongoDB (CP)

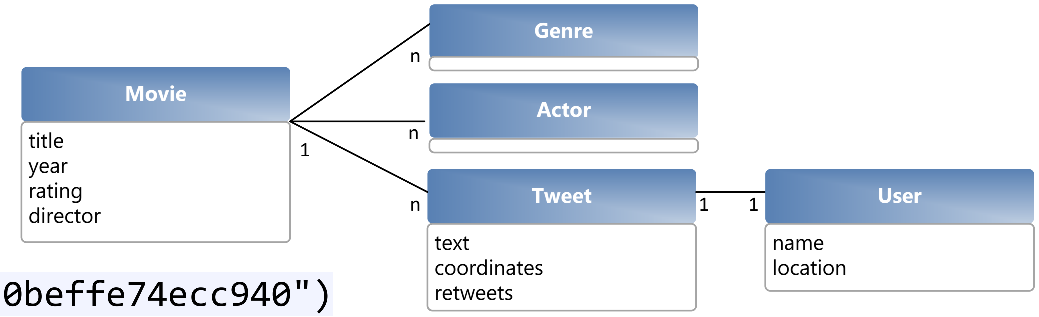
MongoDB
Model:
Document
License:
GNU AGPL 3.0
Written in:
C++

- From hu**mongo**us ≅ gigantic
- Tunable consistency
- Schema-free document database
- Allows complex queries and indexing
- **Sharding** (either range- or hash-based)
- **Replication** (either synchronous or asynchronous)
- Storage Management:
  - **Write-ahead logging** for redos (*journaling*)
  - **Memory-mapped** storage files, buffer management handled by operating system (paging)

Data Modelling



Data Modelling




```
{
        "_id" : ObjectId("51a5d316d70beffe74ecc940")
        title : "Iron Man 3",
        year : 2013,
        rating : 7.6,
        director: "Shane Block",
        genre : [ "Action",
                  "Adventure",
                  "Sci -Fi"],
        actors : ["Downey Jr., Robert",
                  "Paltrow , Gwyneth"],
        tweets : [ {
           "user" : "Franz Kafka",
           "text" : "#nowwatching Iron Man 3",
           "retweet" : false,
           "date" : ISODate("2013-05-29T13:15:51Z")
        }]
}
```

Movie Document

Denormalisation instead of joins

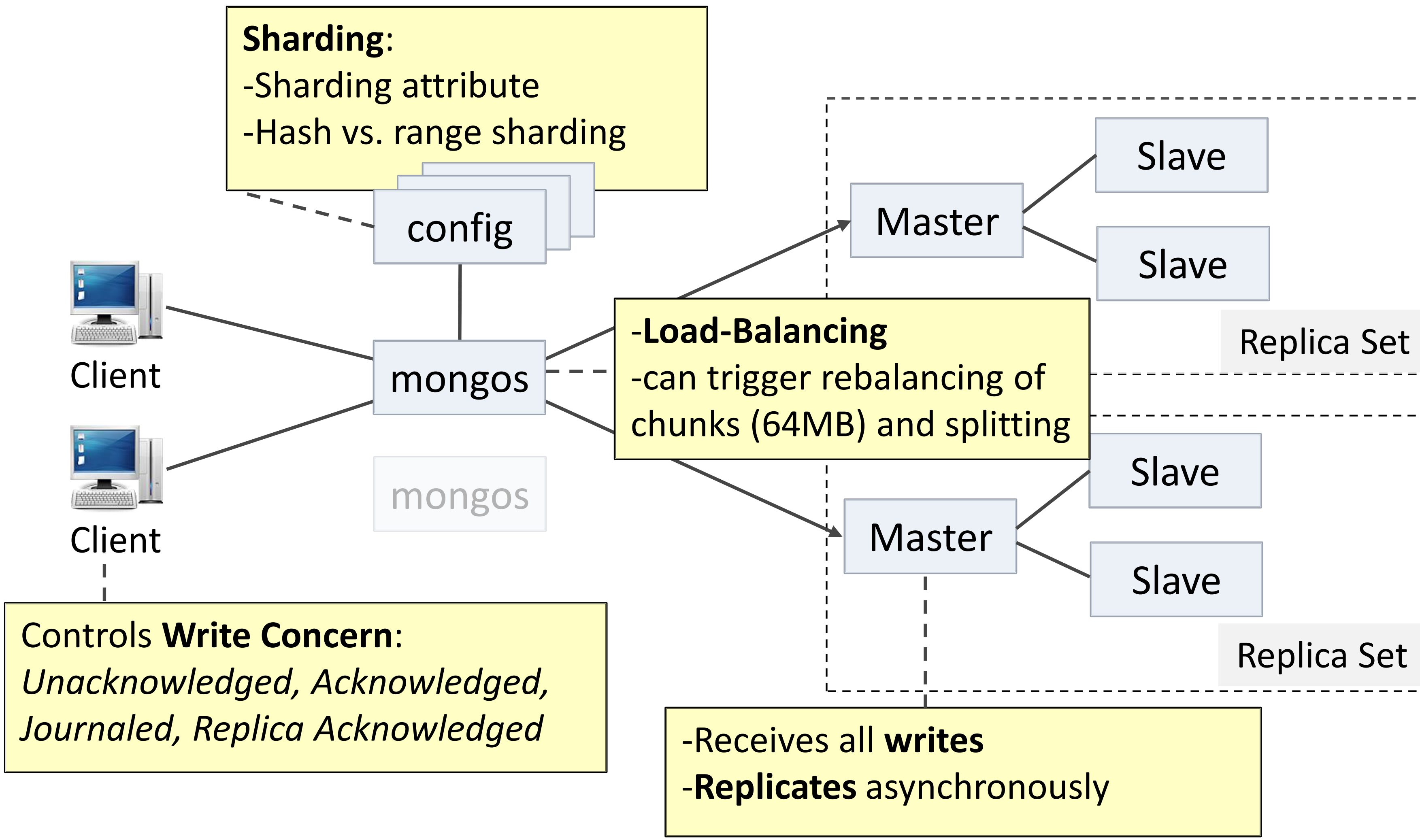
Nesting replaces 1:n and 1:1 relations

Schemafreeness: Attributes per document

Unit of atomicity: document

Principles

Sharding und Replication



Classification: MongoDB

Techniques

Sharding	Range-Sharding	Hash-Sharding	Entity-Group Sharding	Consistent Hashing	Shared Disk
Replication	Trans-action Protocol	Sync. Replica-tion	Async. Replica-tion	Primary Copy	Update Anywhere
Storage Management	Logging	Update-in-Place	Caching	In-Memory	Append-Only Storage
Query Processing	Global Index	Local Index	Query Planning	Analytics	Materialized Views

Outline

Foundations: Big Data, Scalability, Avaialbility

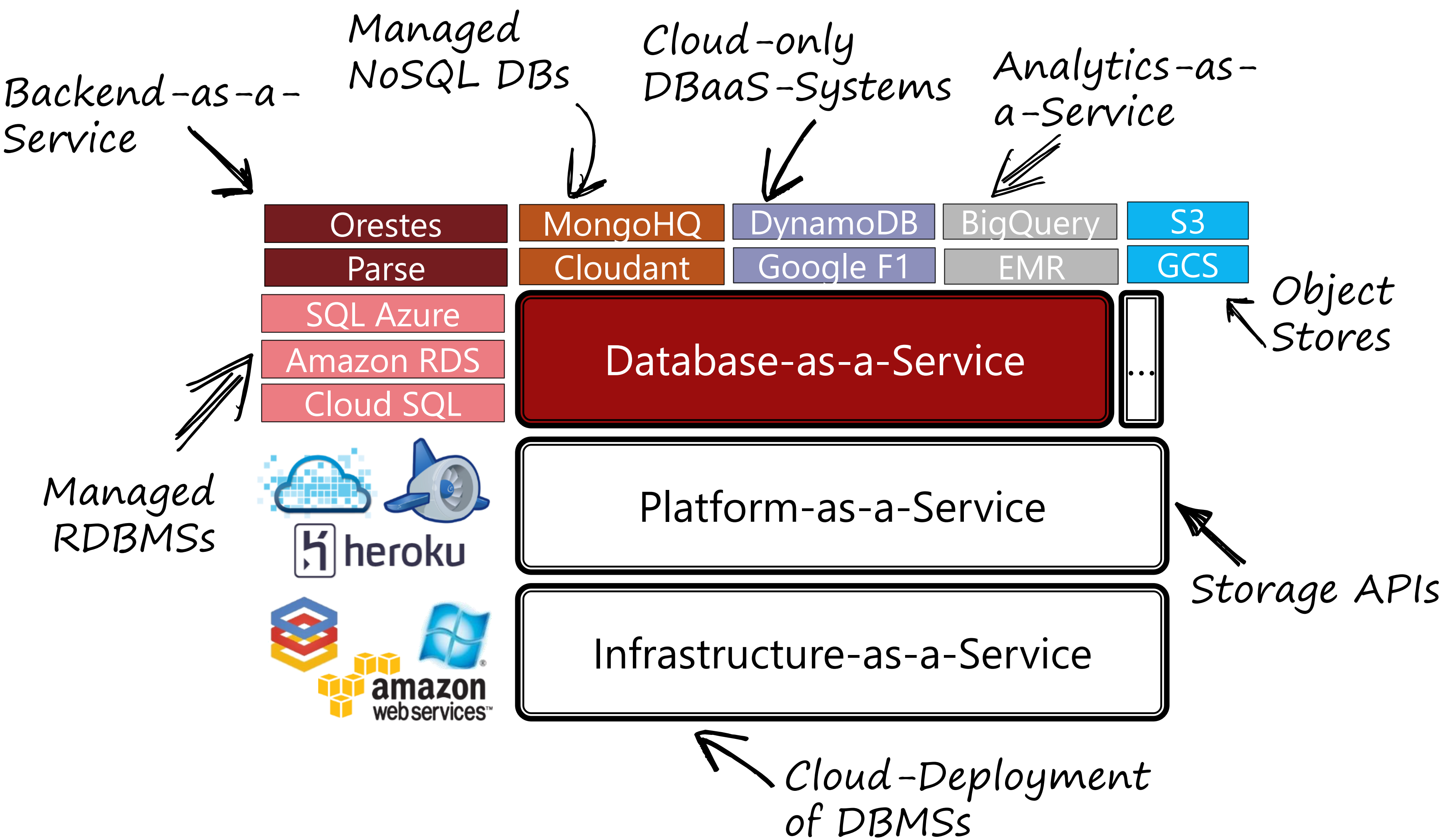
The 4 Classes of NoSQL Databases

NoSQL Examples: concrete Architectures, Systems, APIs

Cloud Databases

- Database-as-a-Service
- Backend-as-a-Service

Cloud Databases



Database-as-a-Service

- Cloud databases with a **pay-per-use** pricing model
- **Managed Database Service**: Existing DBMS deployed and managed in the cloud
  - Managed NoSQL System (e.g. MongoHQ, Redis2Go)
  - Managed RDBMS (e.g. Amazon Relational Database Service)
- **Proprietary Database Service**: special DBMS built for cloud environments (e.g. Amazon DynamoDB)
- **Object Stores**: cloud-based file storage (e.g. Amazon S3)
- **Backend-as-a-Service**: Database + Implementation of standard app concerns (e.g. user management, push)

The Latency Problem

100ms 400ms 500ms 1s

Average: 9,3s

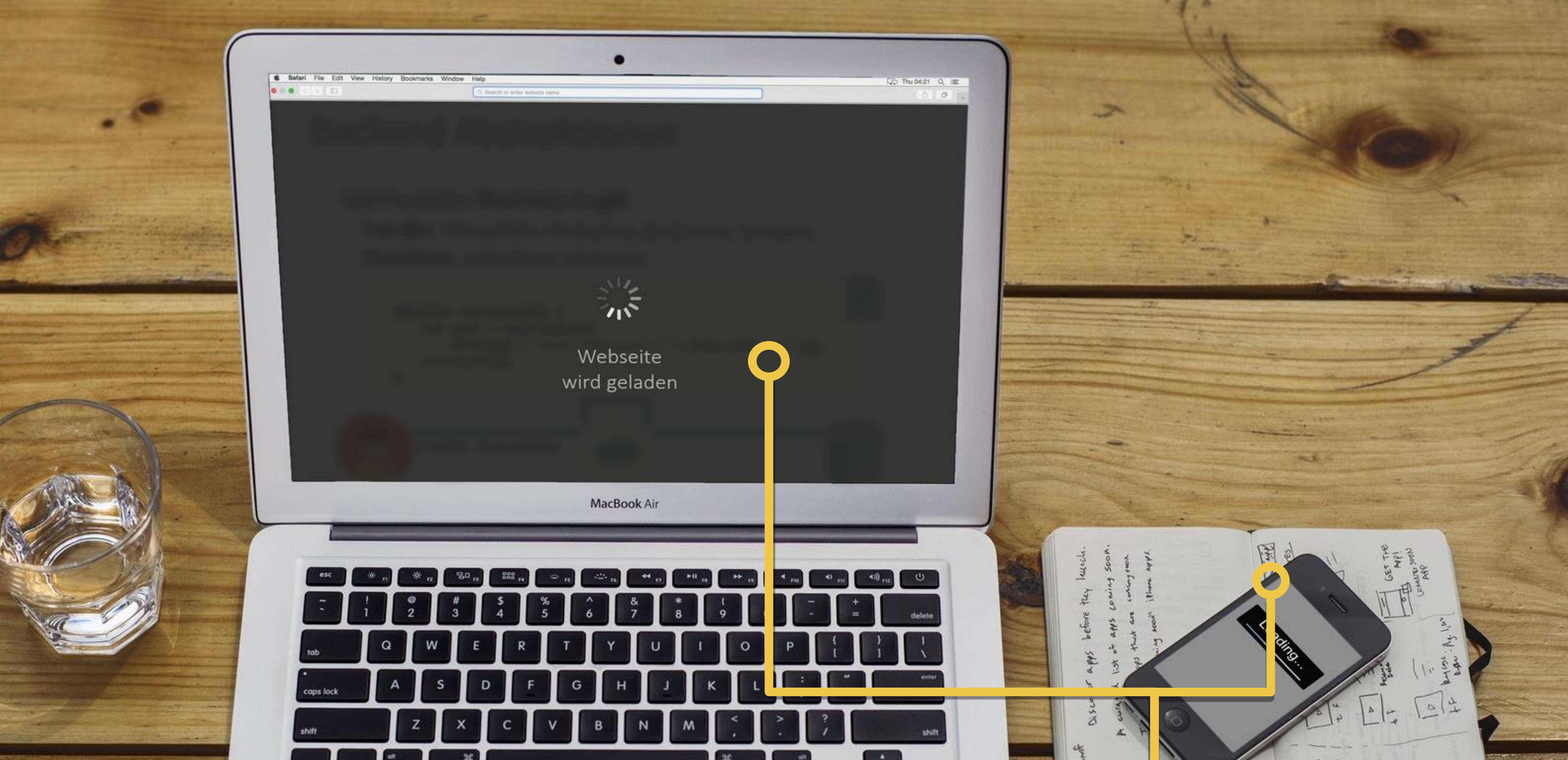
Loading…

Conversions — Aberdeen Group

% Traffic — Google

-9% Visitors — Yahoo!

-1% Revenue — amazon.com

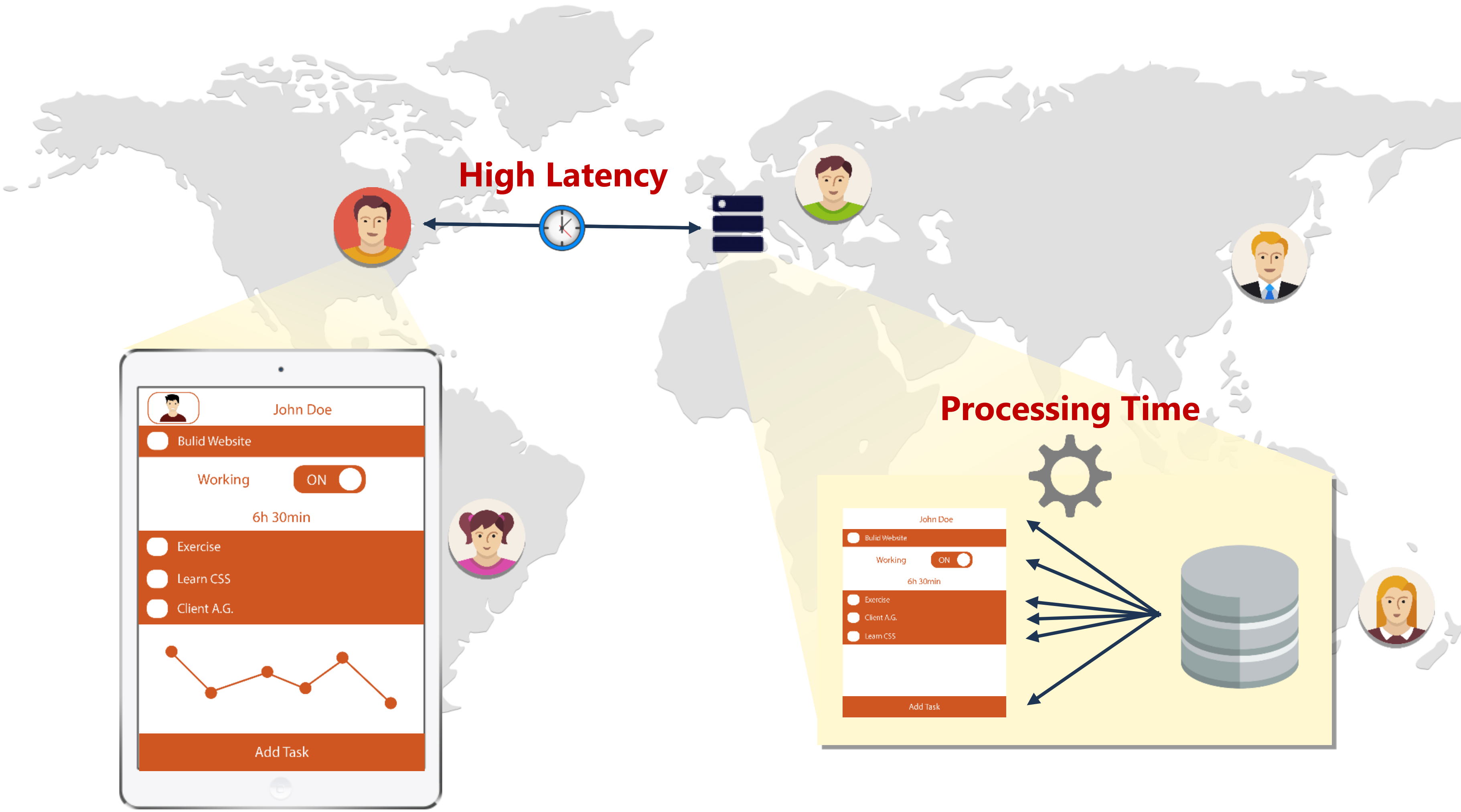


If perceived speed is such an import factor

...what causes slow page load times?

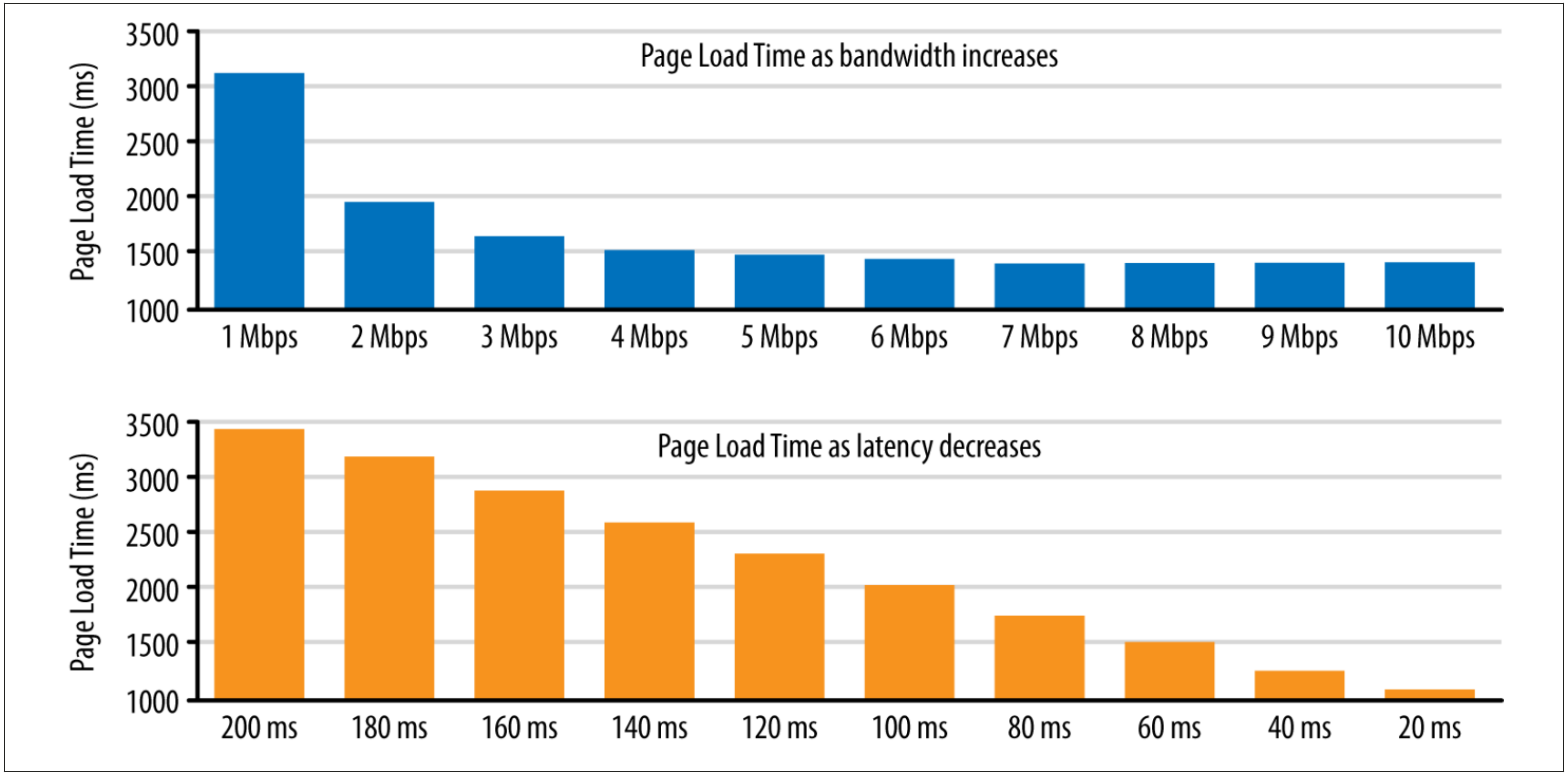
State of the art

Two bottlenecks: latency und processing



Network Latency

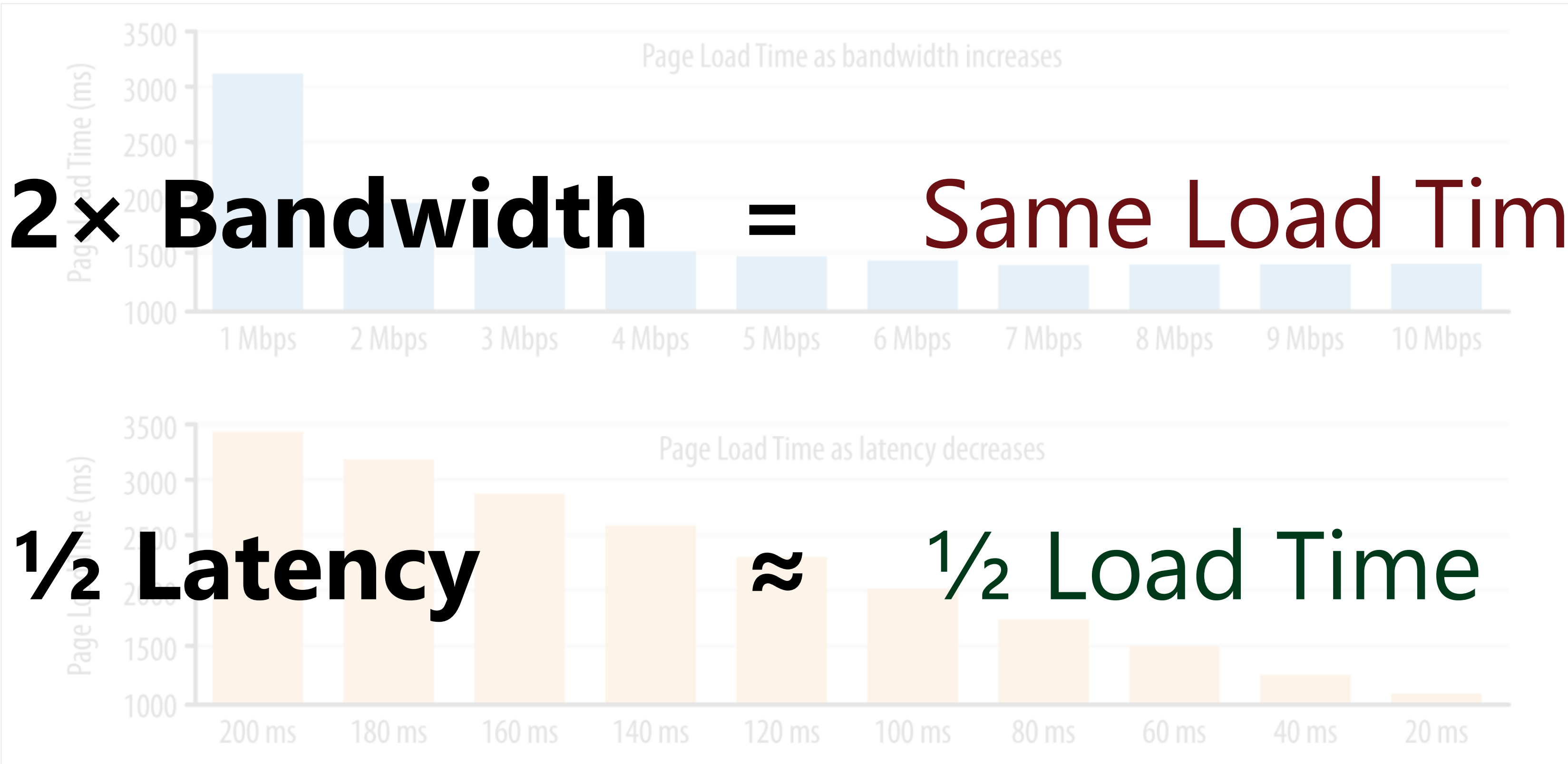
The underlying problem of high page load times



I. Grigorik, High performance browser networking. O'Reilly Media, 2013.

Network Latency

The underlying problem of high page load times



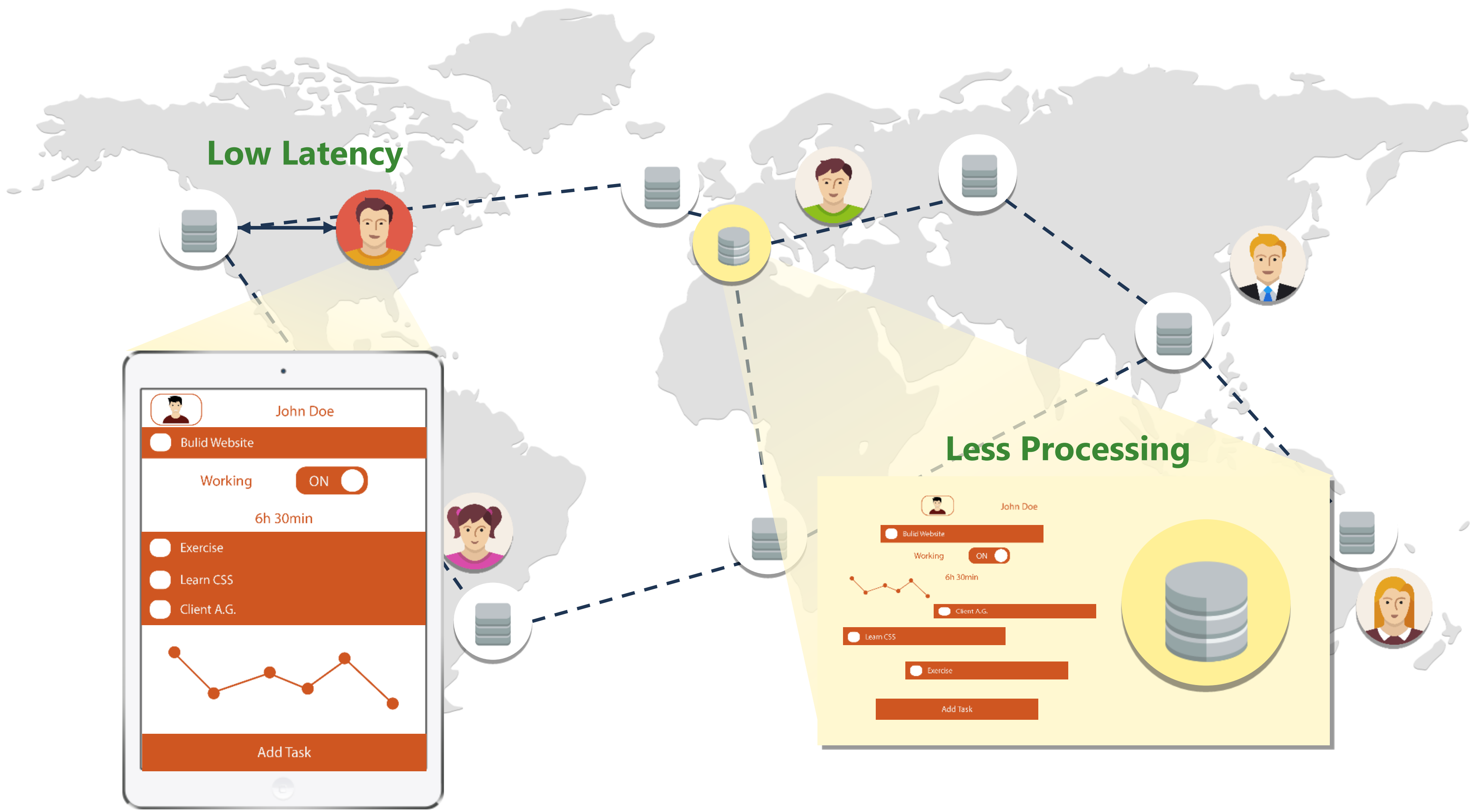
2× Bandwidth = Same Load Time

½ Latency ≈ ½ Load Time

I. Grigorik, High performance browser networking. O'Reilly Media, 2013.

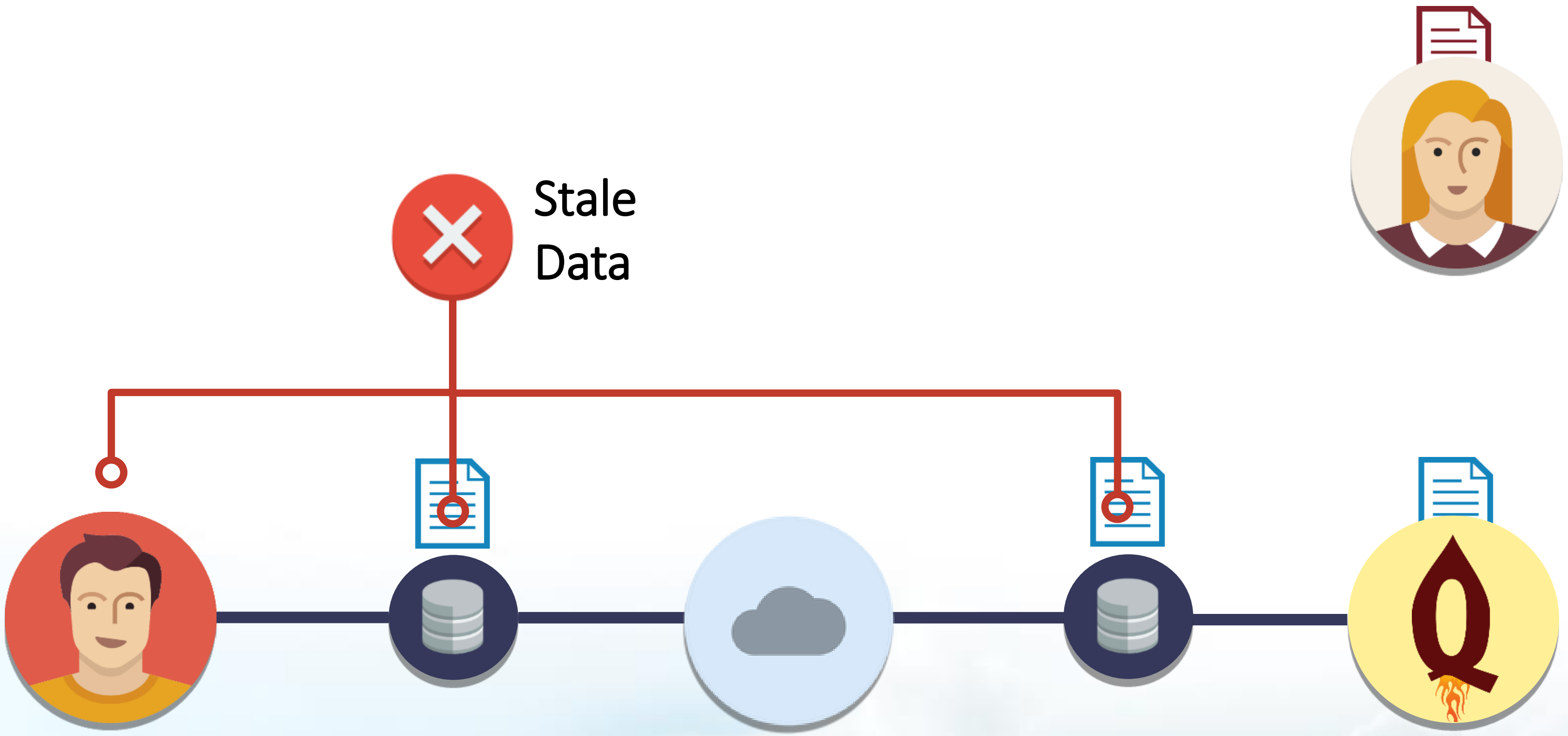
The low-latency vision

Data is served by ubiquitous web-caches



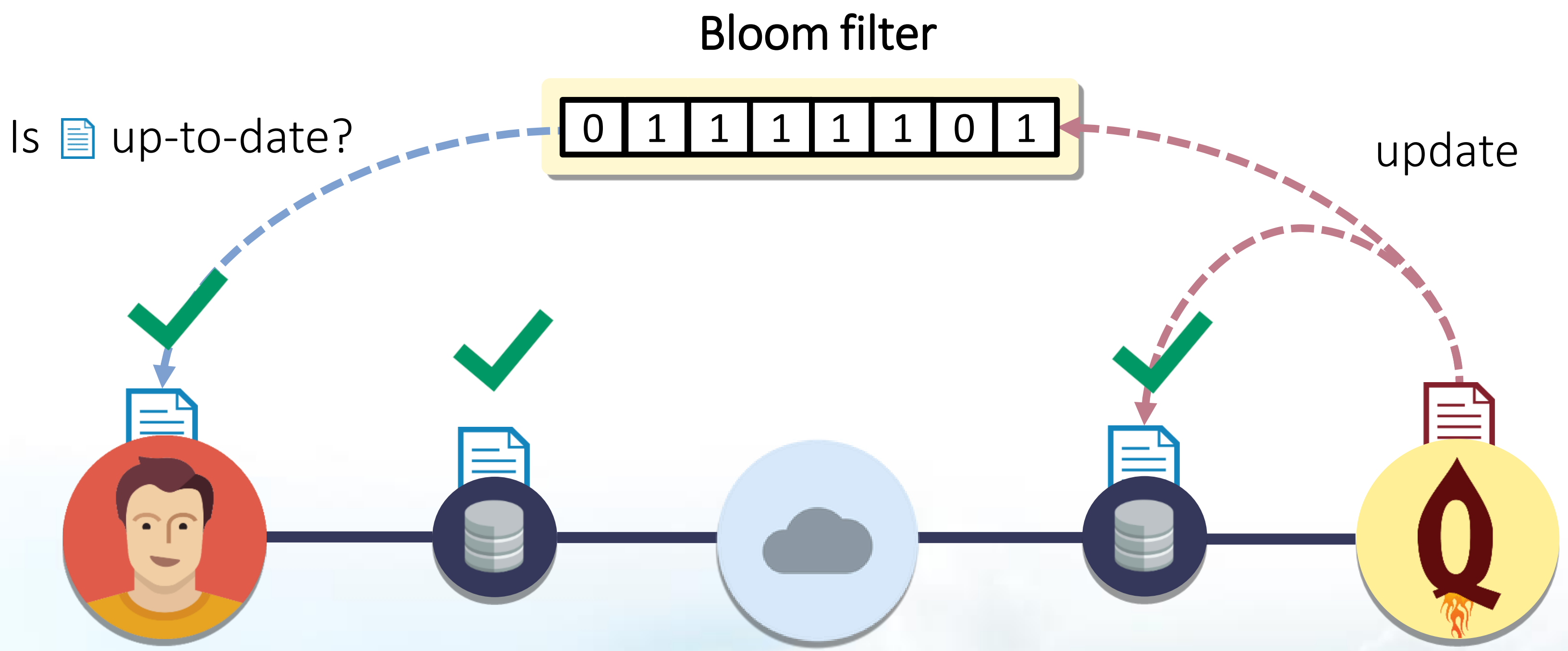
The Problem with today‘s caching

Changes invalidate cached data



Our Research

Keep Data up-to-date through Cache Sketches




BaQend
Build a faster web.

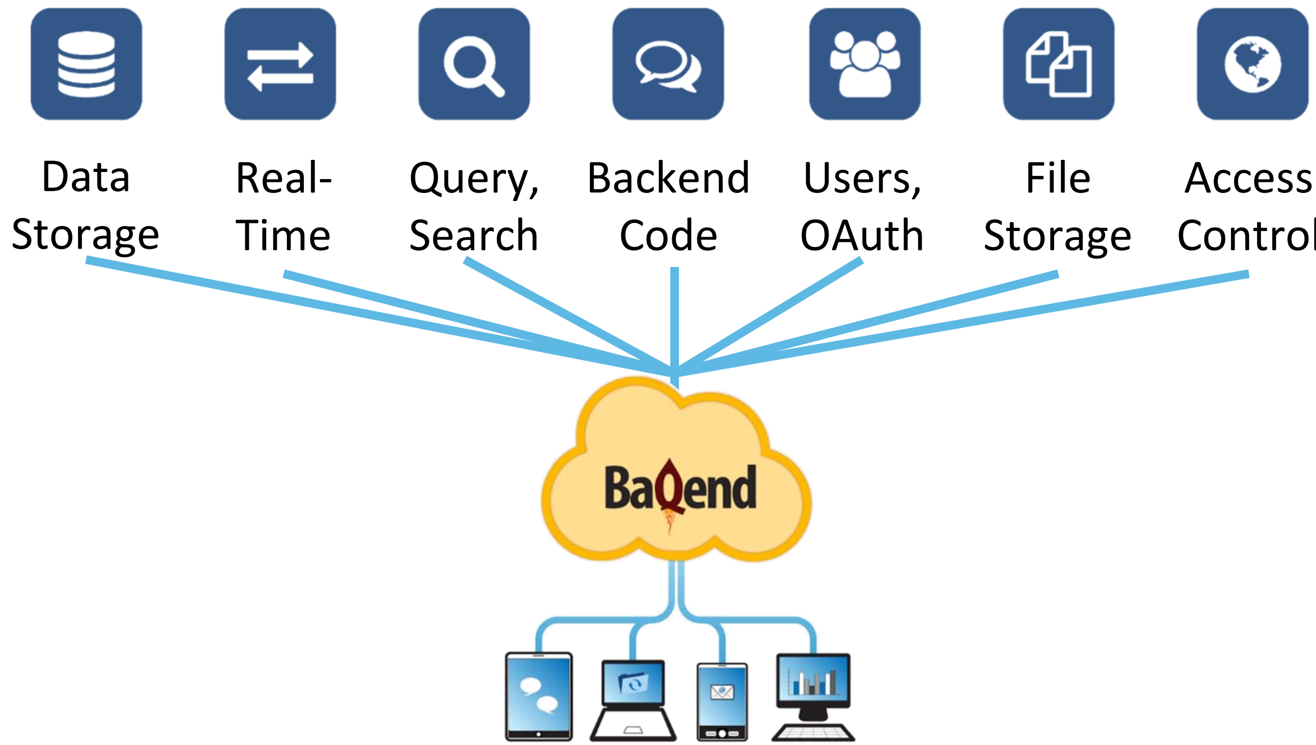
Backend-as-a-Service

Feature Sets

API for application features

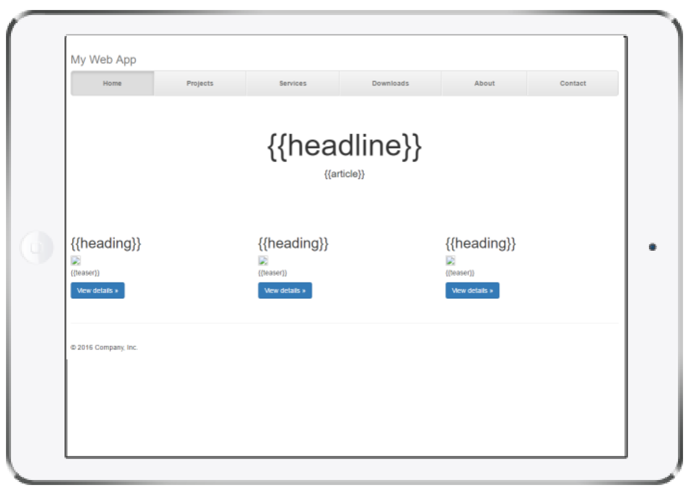
Hosting and **Delivery**

REST API and **JS SDK**



Frontend

GET /app.html
GET /js/main-34da93.js
GET /css/main-9ad7ca3.css



Compatible with:

ANGULARJS by Google

ember

ionic

React

handlebars

UNDERSCORE.JS

Lodash

Knockout.

Vue.js

Frontend

GET /app.html
GET /js/main-34da93.js
GET /css/main-9ad7ca3.css

```
db.Page.load('main')
  .done(...);
```

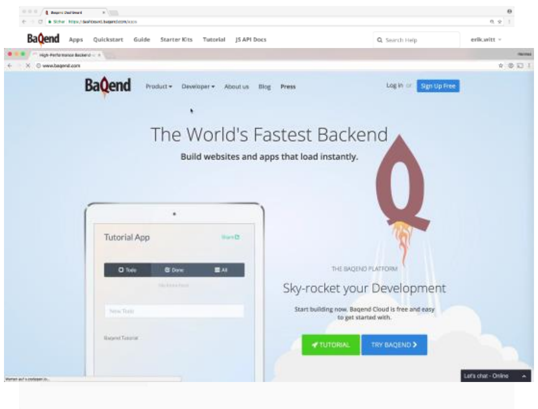
```
db.Page.find()
  .descending('published')
  .limit(3)
  .resultList(...);
```

GET /img/pic005.jpg
GET /img/pic017.jpg
GET /img/pic022.jpg

Development

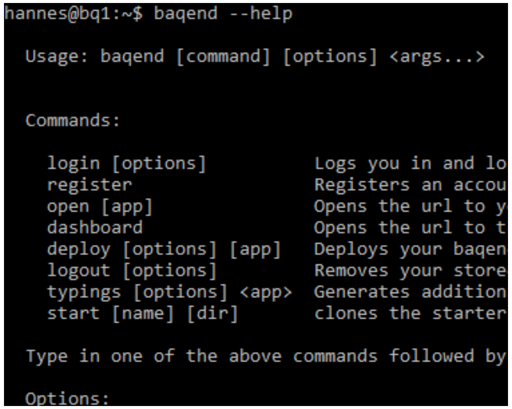
On Baqend

Dashboard



Create Schema, configure, browse data, etc.

CLI



Develop, deploy and test frontend und backend Code

REST & SDK

Website logic: load site, get data, login, etc.

Orestes & Baqend

Learn more

If you are interested in topics combining web/mobile with scalable data management:

Bachelor and Master thesis topics
http://tiny.cc/orestes (frequently updated)

Hiwi/student positions

Tasks: building real applications using cutting edge technology (e.g. ES6, Angular, React, MongoDB, Redis, Node.js,…)

Contact me directly or at fg@baqend.com

Summary

- Variety of different NoSQL systems:
  - **HDFS and Hadoop**: Map-Reduce platform for batch analytics
  - **Dynamo and Riak**: KV-store with consistent hashing
  - **Redis**: replicated, in-memory KV-store
  - **BigTable, HBase, Cassandra**: wide-column stores
  - **MongoDB**: sharded and replicated document store
- Cloud Databases
  - **Database-as-a-Service**: managed (NoSQL) database provided as a pay-per-use service
  - **Orestes and Baqend**: Backend-as-a-Service research project and startup with the goal of solving the web‘s latency problem