Tutorial 18: Going for Speed

09:00	Introduction
09:15	Part 1: Efficient Frontend Design
09:40	Q&A
09:50	Coffee Break
10:00	Part 2: High-Performance Networking
10:40	Q&A
10:50	Coffee Break
11:00	**Part 3: Scalable Backend Architectures**
11:40	Q&A
11:50	Coffee Break
12:00	Part 4: Performance Tracking & Analysis
12:00	The Core Web Vitals (Google Guest Speaker!)
12:30	Measuring Web Performance
12:50	Q&A

Up next!

Baqend

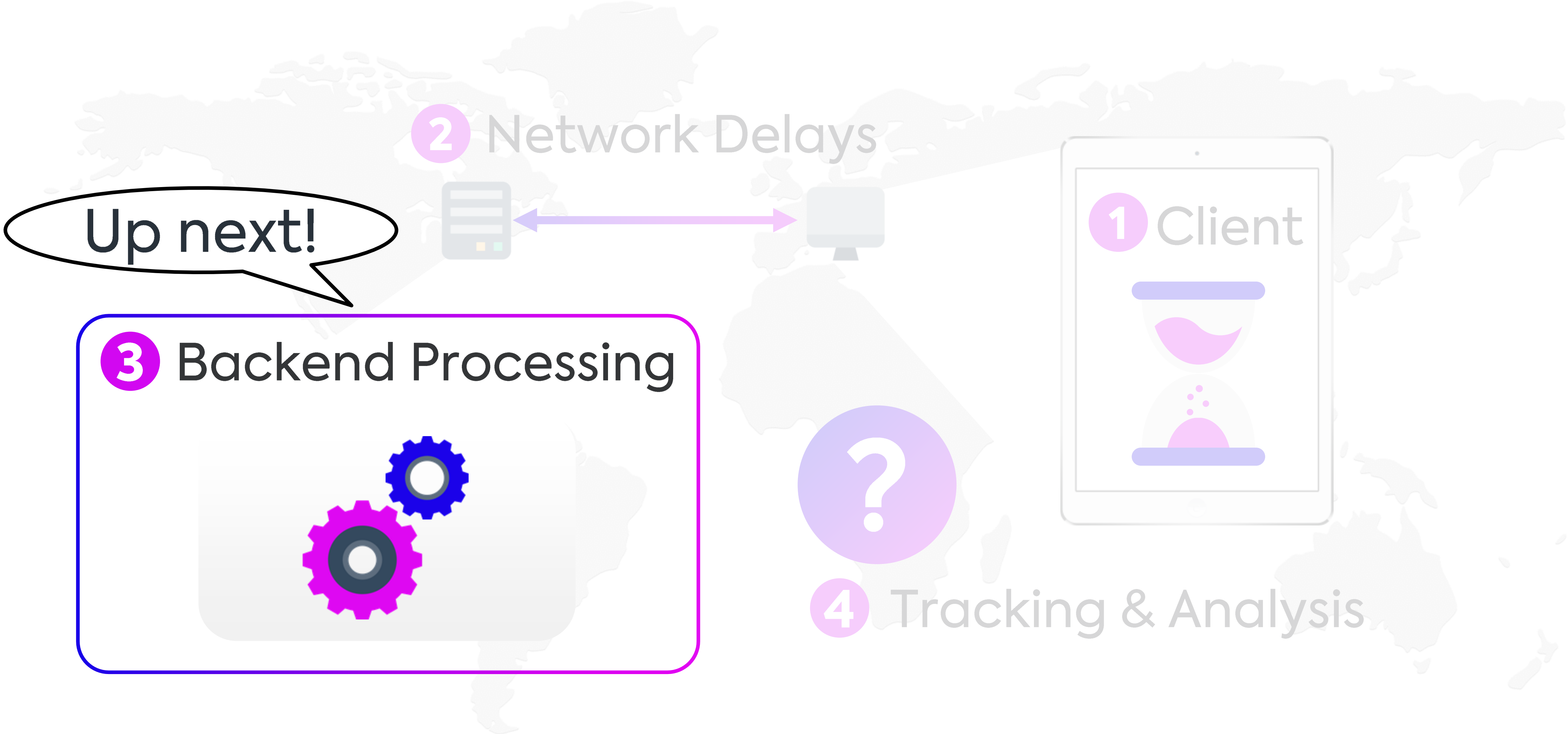
Going for Speed

Part 3:
Scalable Backend Architectures

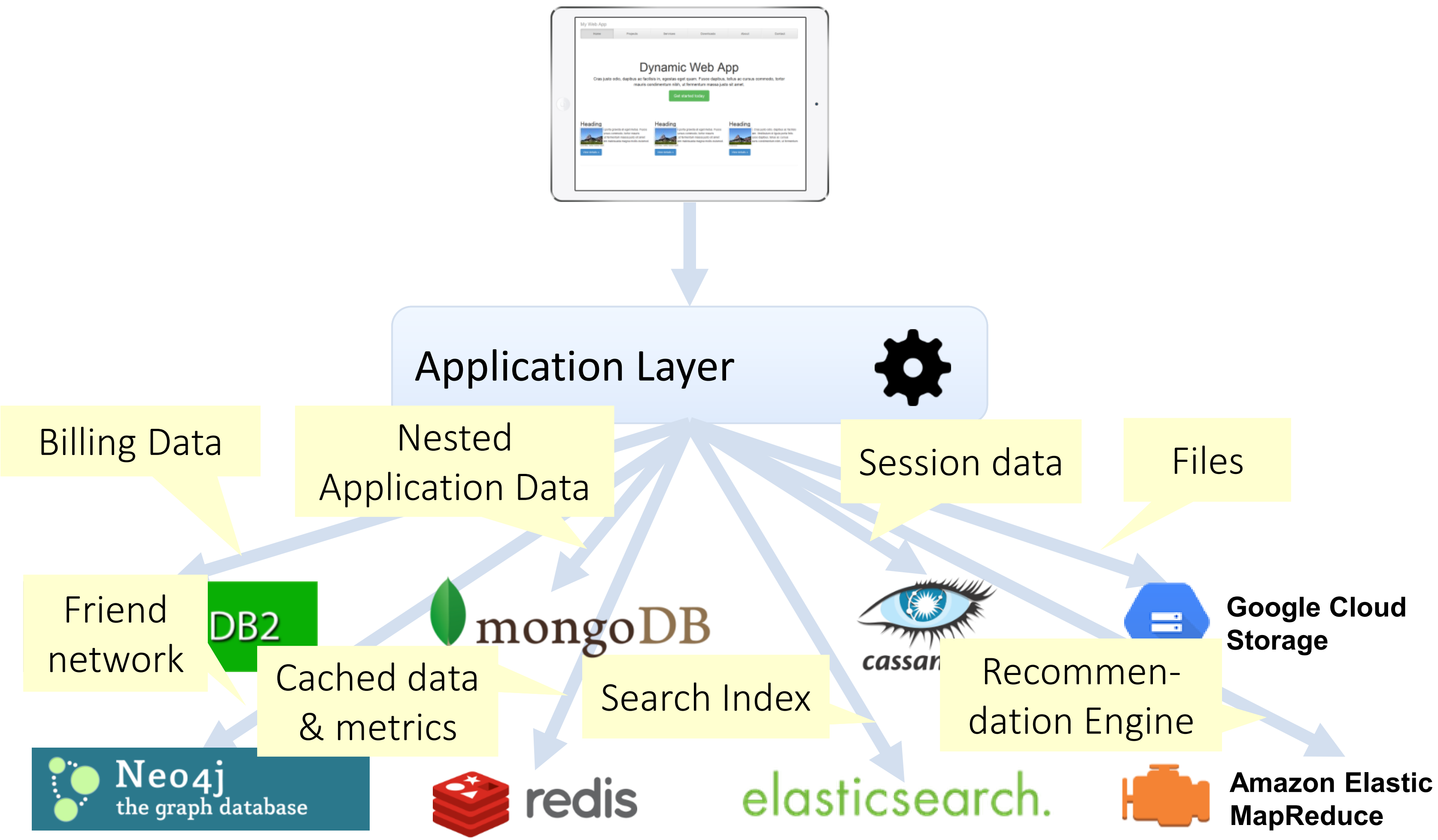
The Web Conference
April 15, 2021
Tutorial

Wolfram Wingerath, Benjamin Wollmer, Felix Gessert, Stephan Succo, Norbert Ritter

The 4 Challenges of Web Performance



How to Choose a Database System



How to Choose a Database System

How to get from requirements to a concrete system?

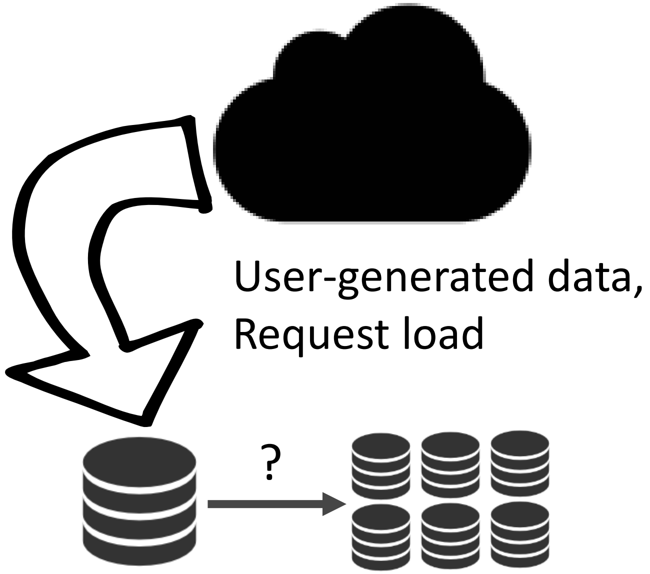
NoSQL!?

- „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

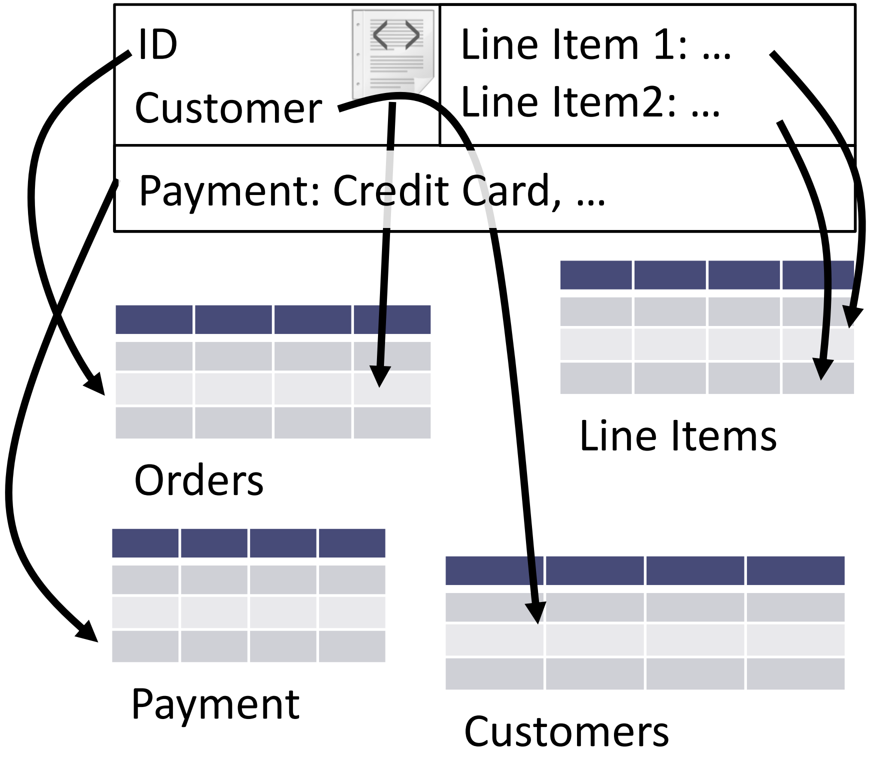
***NoSQL-Databases.org*:**
Current list has over 200 NoSQL systems

NoSQL: Two Main Motivations

Scalability

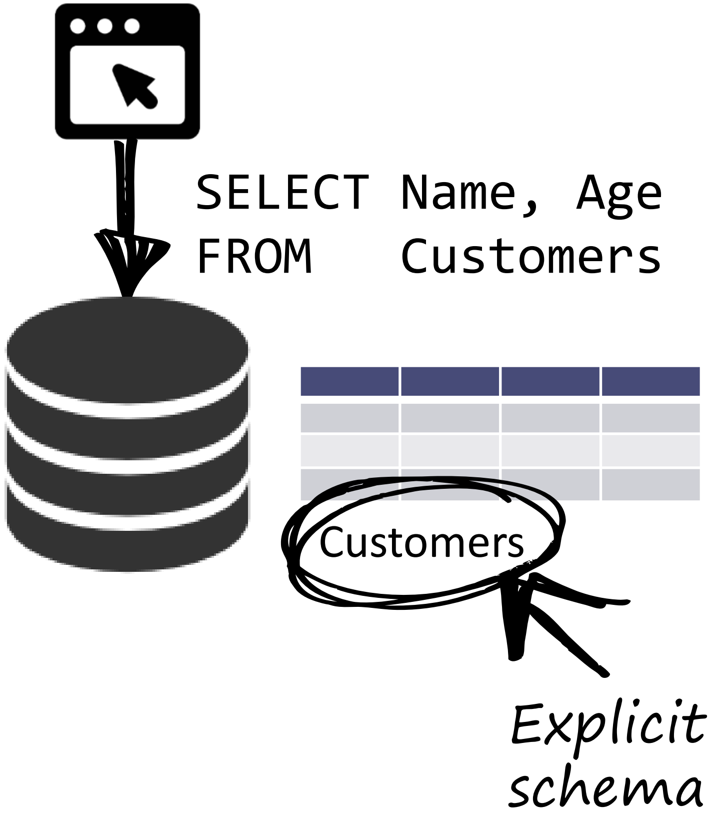


Impedance Mismatch

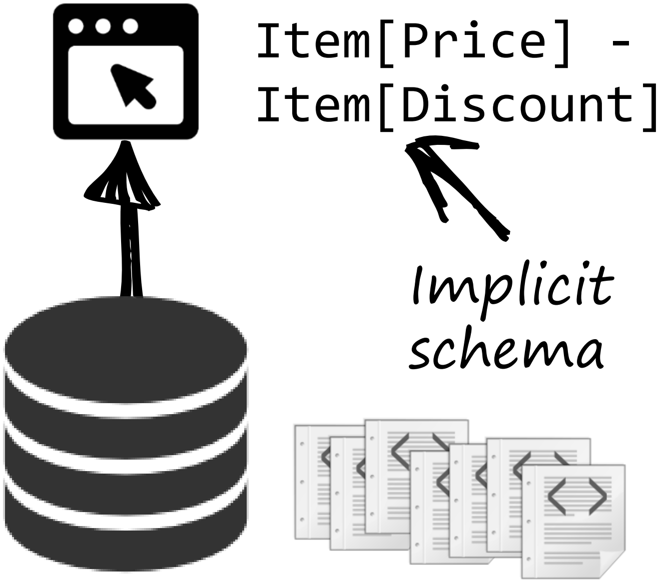


Schema-Free Data Remodeling

RDBMS:

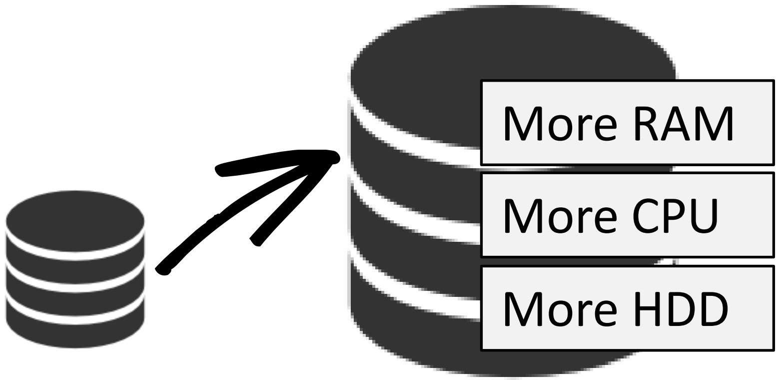


NoSQL DB:

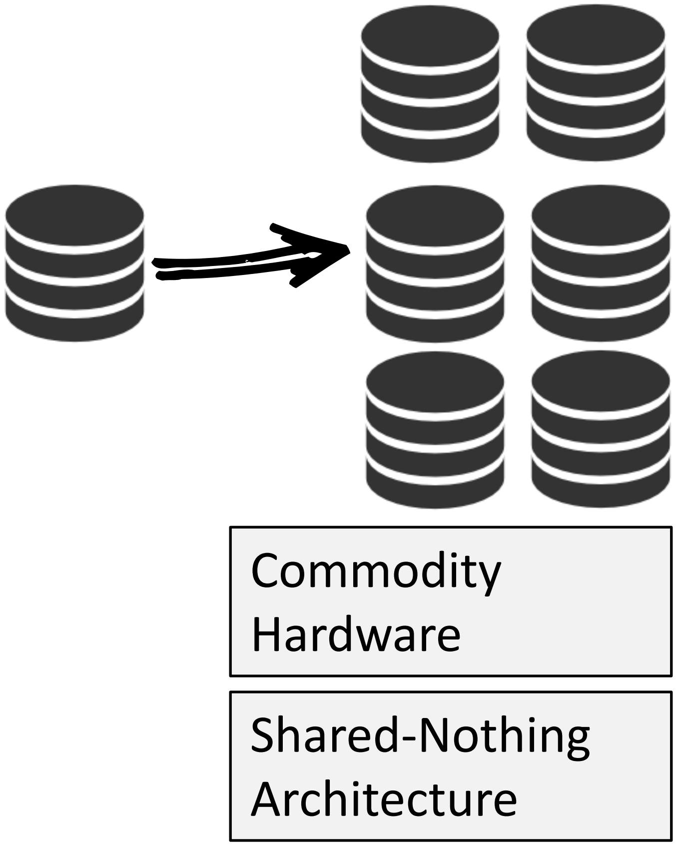


Scale-Up vs. Scale-Out

Scale-Up (*vertical* scaling):



Scale-Out (*horizontal* scaling):

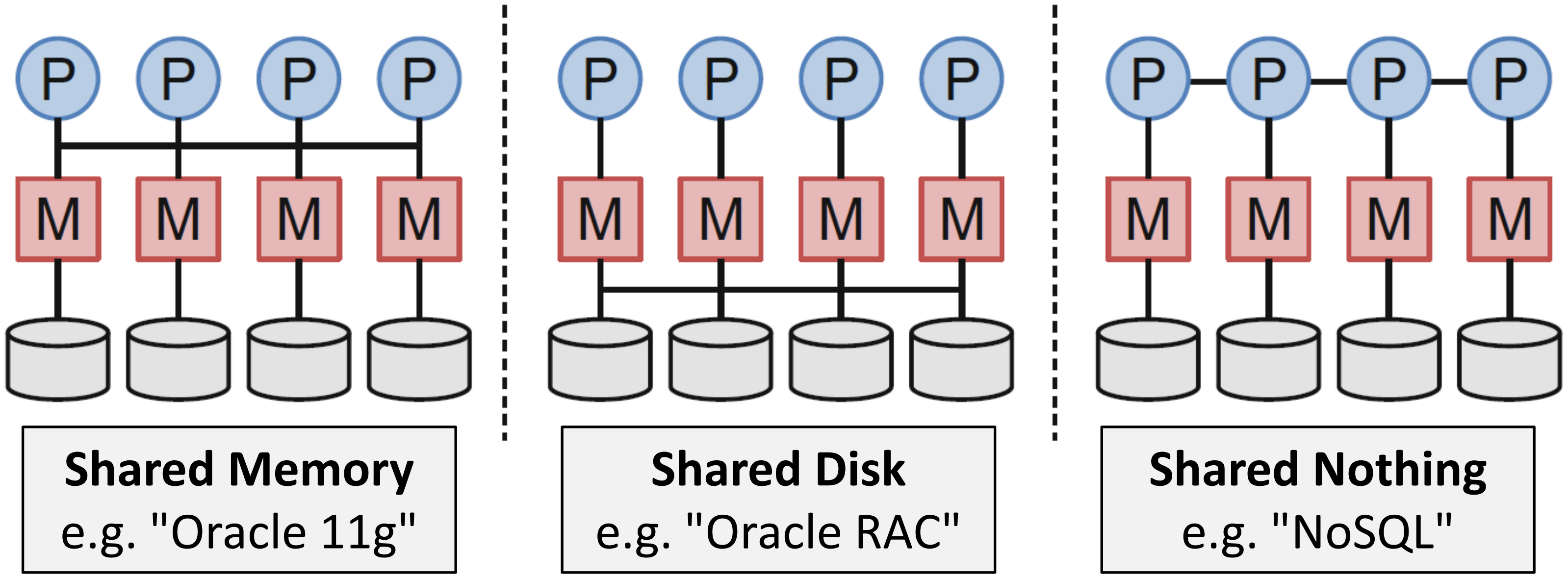


Paradigm Shift: Open-Source & Commodity

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: Shared Nothing

Shift towards higher distribution & less coordination:



Typical Classification Schemes

- Two common criteria:

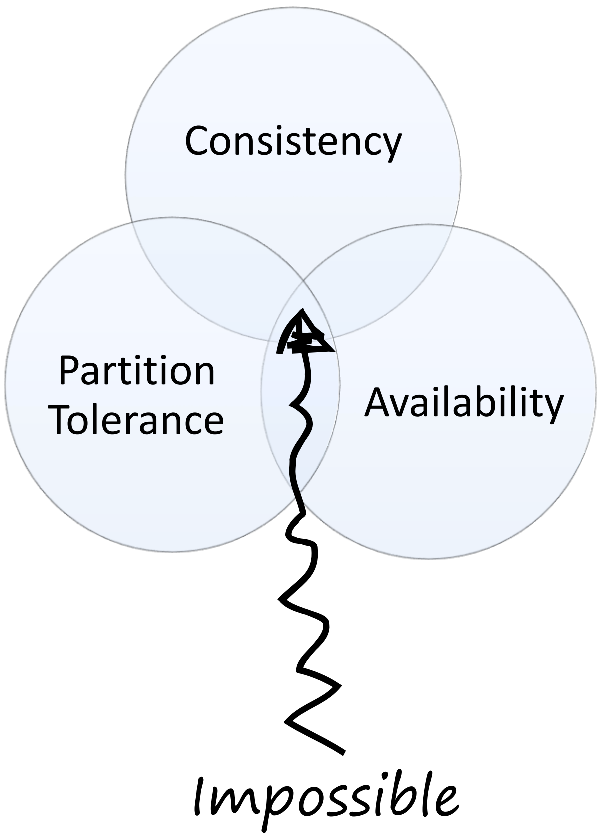
Data Model

- Key-Value
- Wide-Column
- Document
- Graph

Consistency/Availability Trade-Off

- **AP**: Available & Partition Tolerant
- **CP**: Consistent & Partition Tolerant
- **CA**: Not Partition Tolerant

CAP Theorem



In a distributed system, only 2 out of the following 3 properties are achievable at the same time:

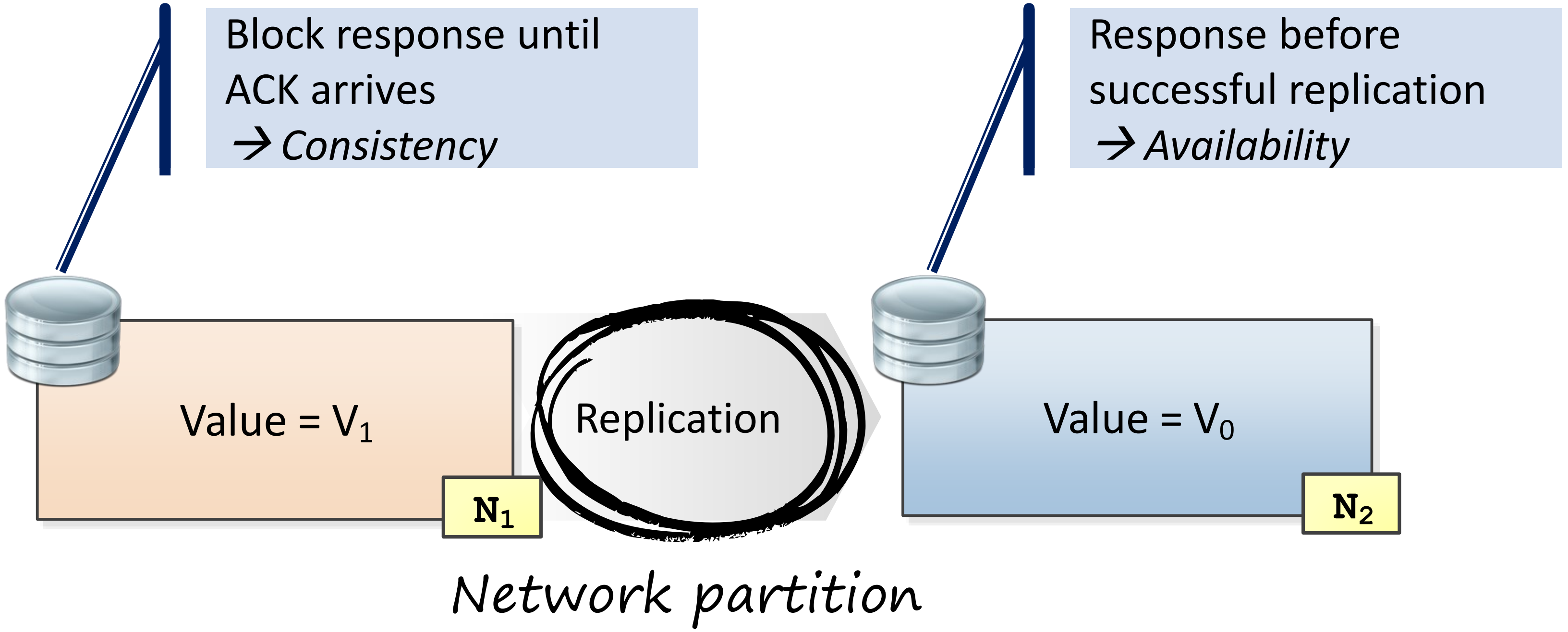
- **Consistency**: all clients have the same view on the data
- **Availability**: every request to a non-failed node most result in correct response
- **Partition tolerance**: the system has to continue working, even under arbitrary network partitions

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: Intuitive Explanation

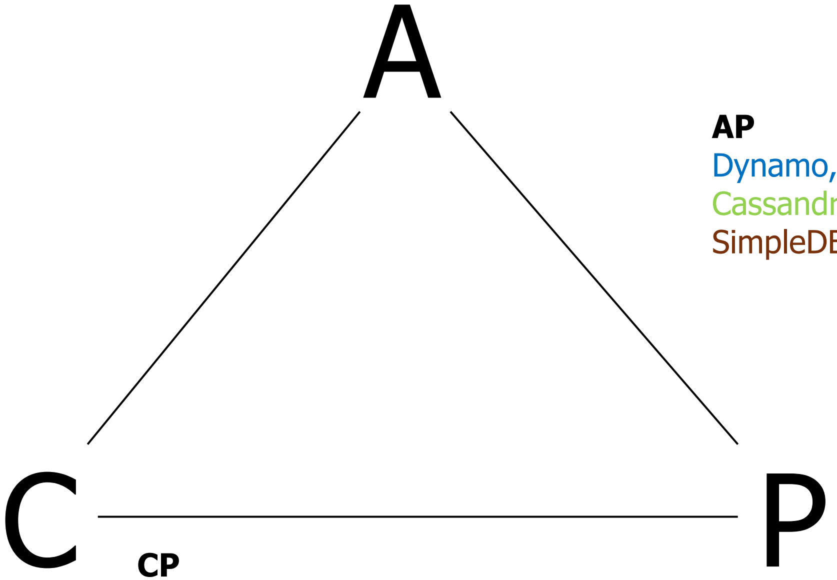
- **Problem**: when a network partition occurs, either consistency or availability have to be given up



NoSQL Triangle

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

Every client can always read and write



CA
Oracle, MySQL, …

AP
Dynamo, Redis, Riak, Voldemort
Cassandra
SimpleDB

CP
Postgres, MySQL Cluster, Oracle RAC
BigTable, HBase, Accumulo, Azure Tables
MongoDB, RethinkDB, DocumentsDB

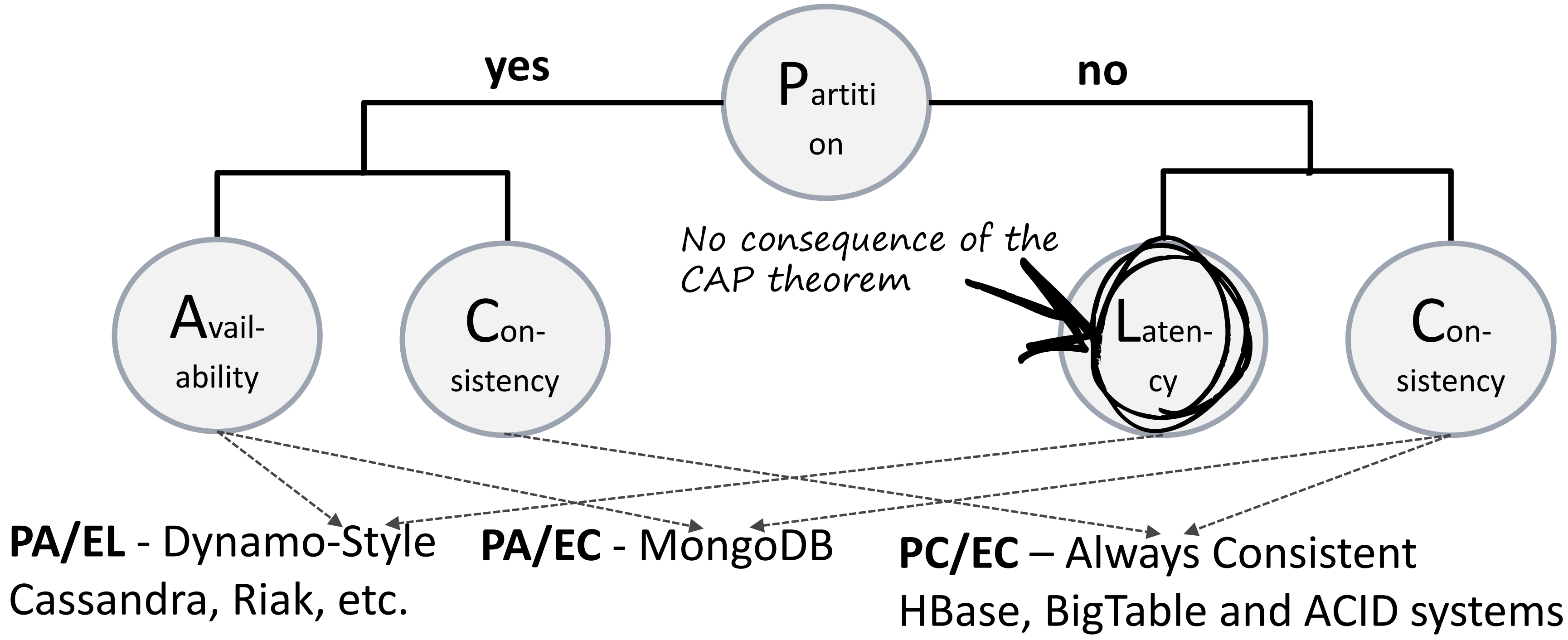
All clients share the same view on the data

All nodes continue working under network partitions

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

PACELC: What About Normal Operation?

- **Idea**: Classify systems by behavior during network partitions *and* normal operation



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

ACID vs. BASE

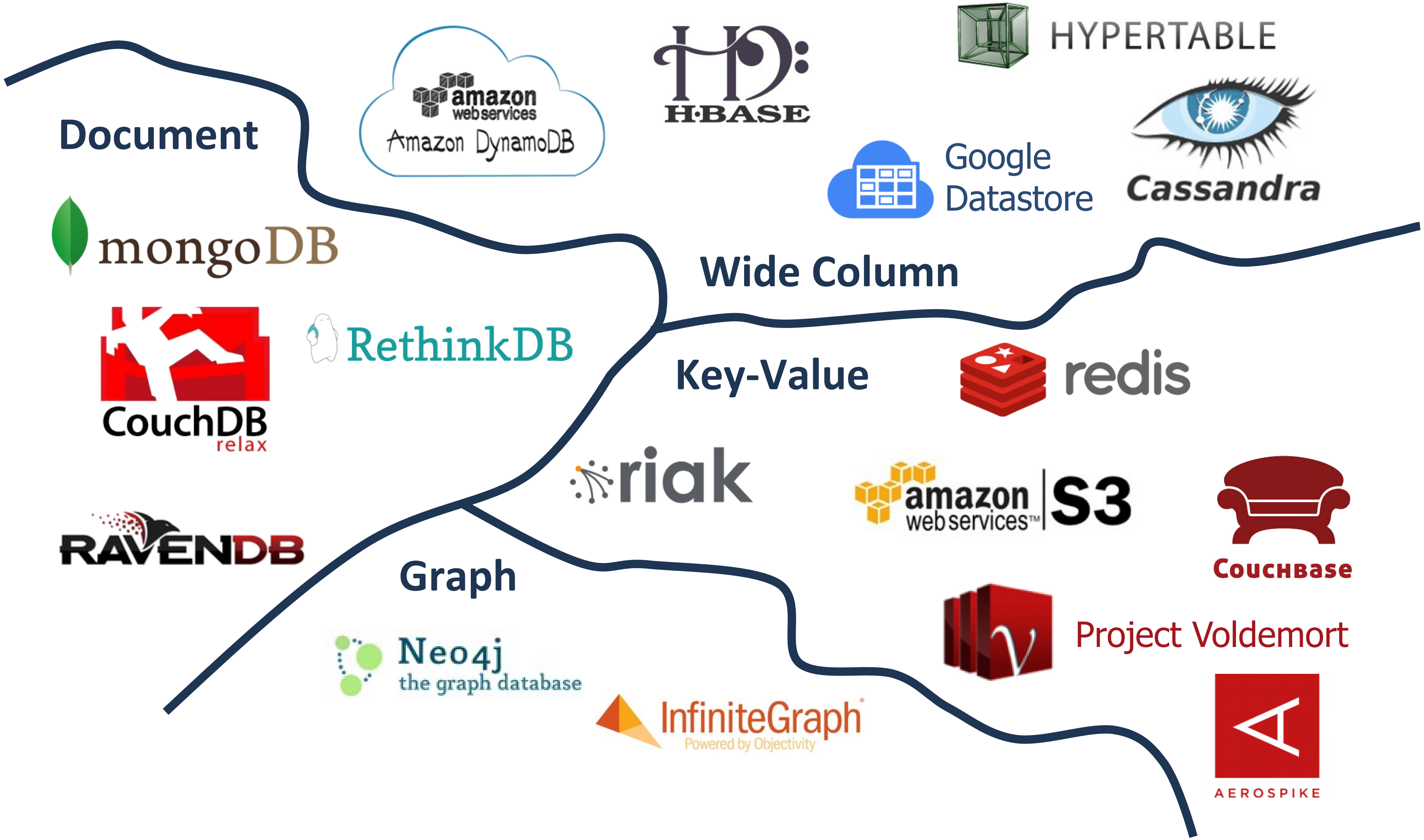
- **ACID**
  - **A**tomicity
  - **C**onsistency
  - **I**solation
  - **D**urability
- **BASE**
  - **B**asically **A**vailable
  - **S**oft State
  - **E**ventually Consistent

ACID: „Gold standard“ for RDBMSs

BASE: Model of many NoSQL systems

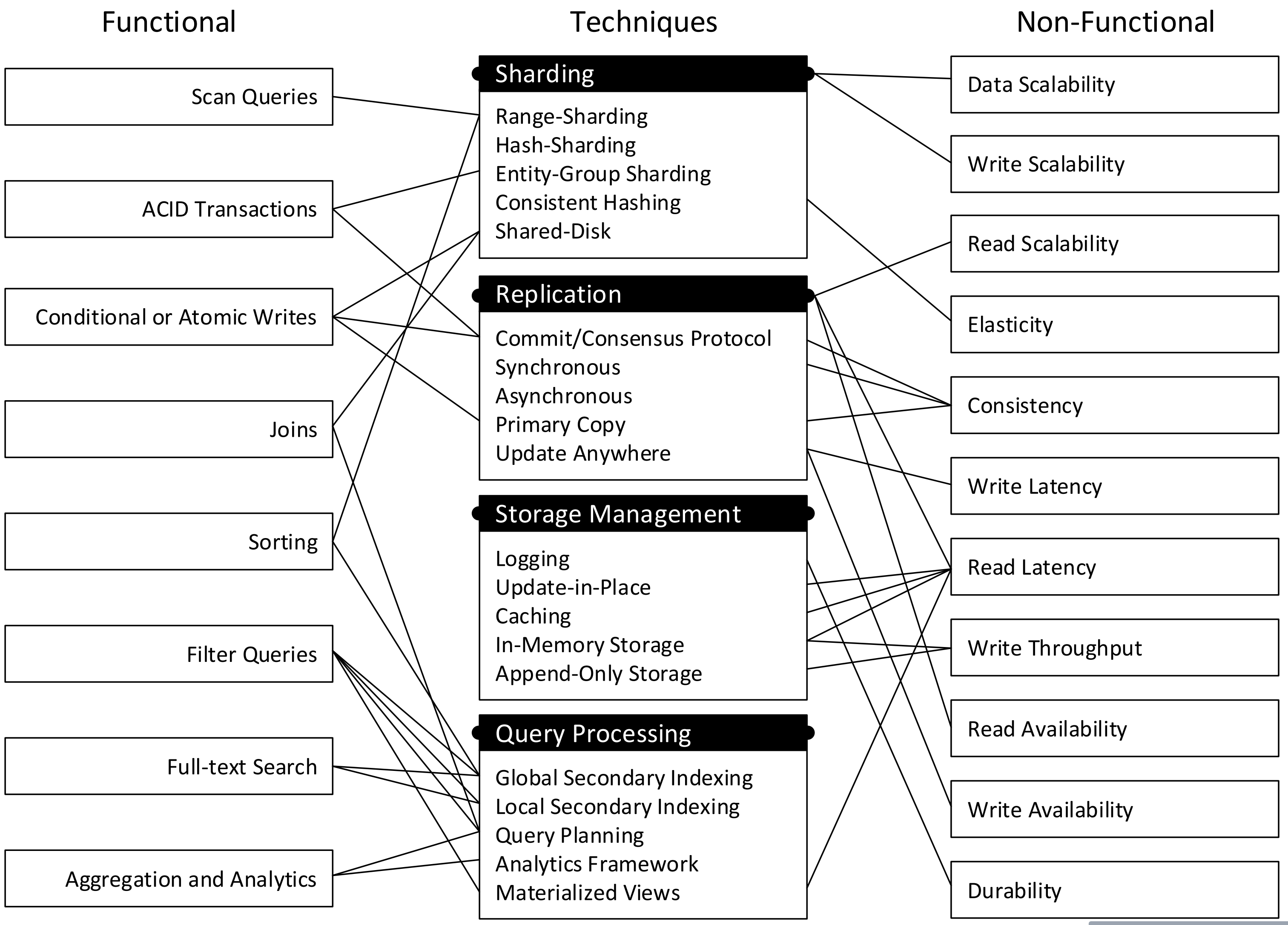
http://queue.acm.org/detail.cfm?id=1394128

NoSQL Landscape



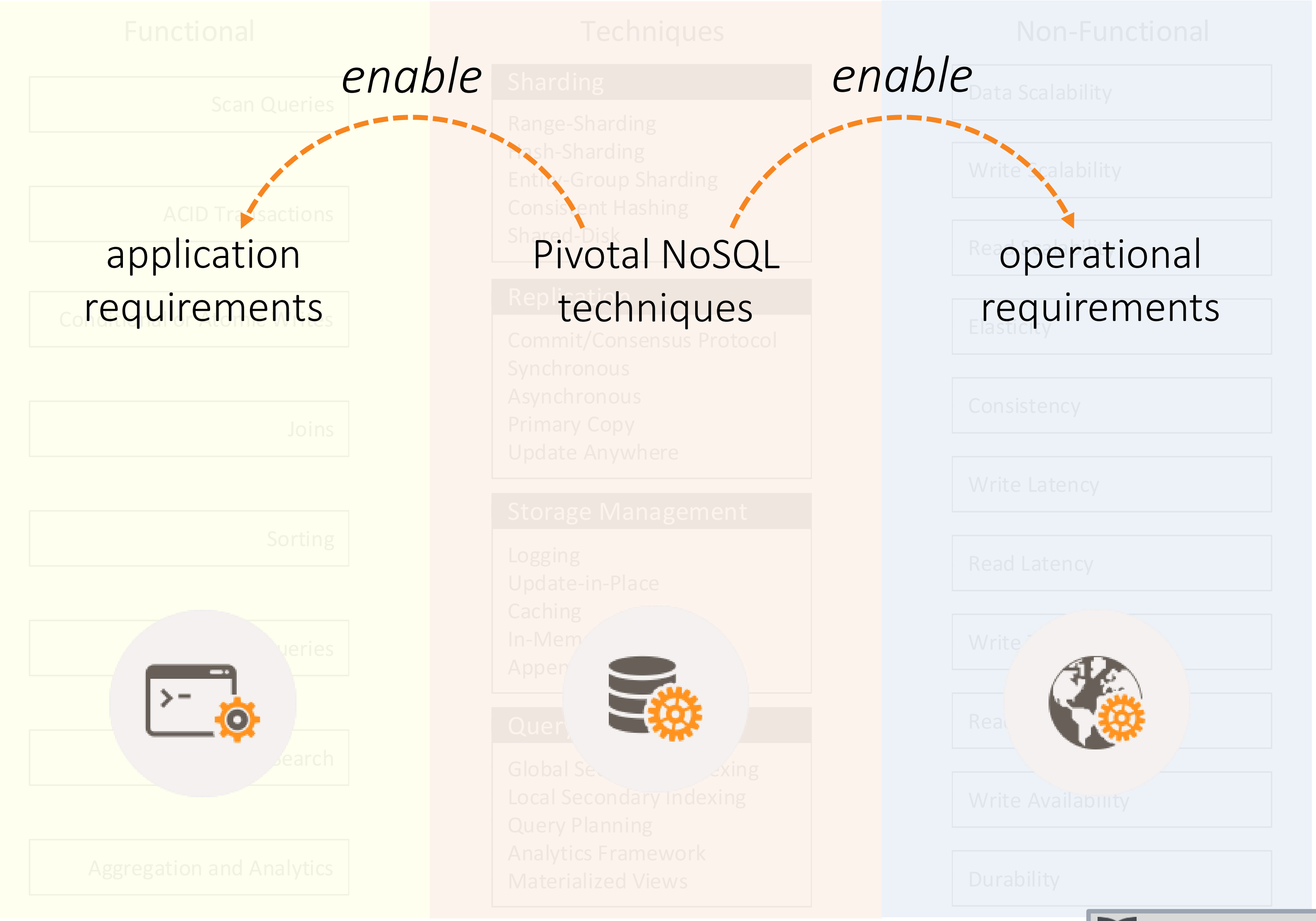
The NoSQL Toolbox

Requirements vs. Techniques

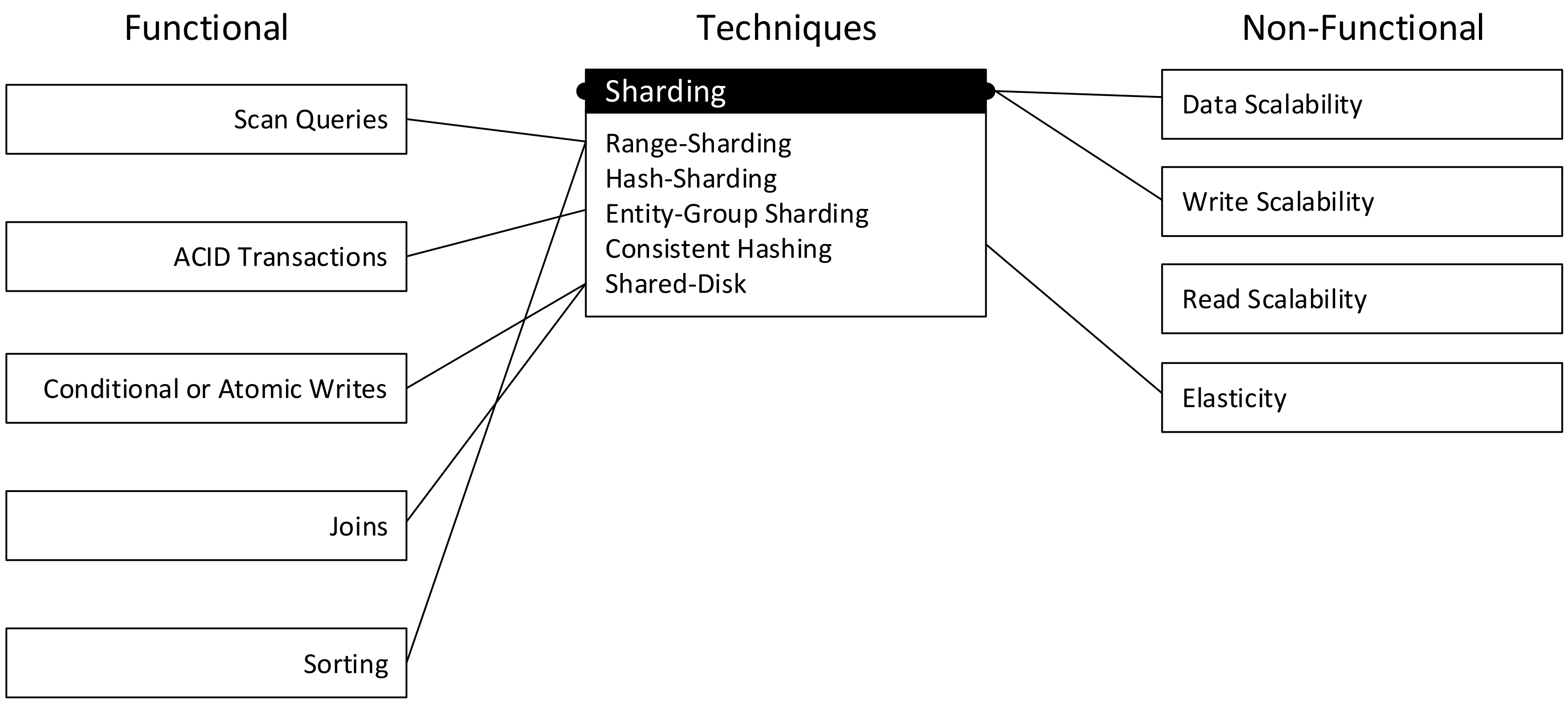


https://www.springer.com/gp/book/9783030435059

Requirements vs. Techniques

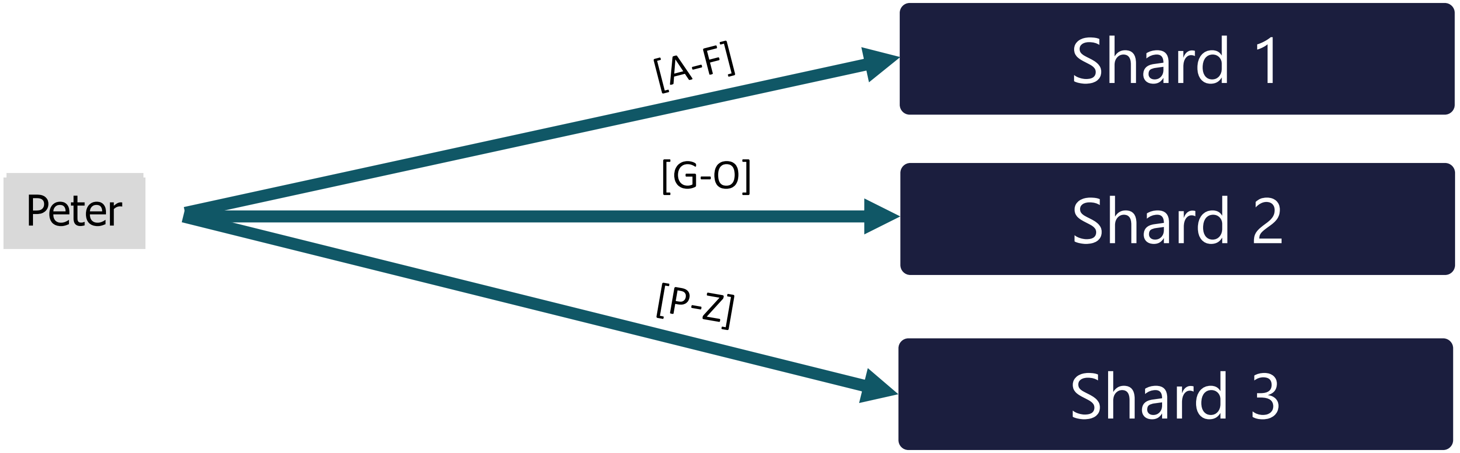


https://www.springer.com/gp/book/9783030435059


Functional
Techniques
Non-Functional
Scan Queries
ACID Transactions
Conditional or Atomic Writes
Joins
Sorting
Sharding
Range-Sharding
Hash-Sharding
Entity-Group Sharding
Consistent Hashing
Shared-Disk
Data Scalability
Write Scalability
Read Scalability
Elasticity

Sharding: Scaling Storage & Throughput

- Horizontal distribution of data over nodes



- **Partitioning strategies**: Hash-based vs. Range-based
- Difficulty: Multi-Shard-Operations (join, aggregation)

Sharding: Approaches

Hash-based Sharding

- Hash of data values (e.g. key) determines partition (shard)
- **Pro**: Even distribution
- **Contra**: No data locality

Implemented in

Dynamo, Cassandra, MongoDB, Riak, Redis, Azure Table,

Range-based Sharding

- Assigns ranges defined over fields (shard keys) to partitions
- **Pro**: Enables *Range Scans* and *Sorting*
- **Contra**: Repartitioning/balancing required

Implemented in

BigTable, HBase, DocumentDB Hypertable, MongoDB, RethinkDB, Espresso

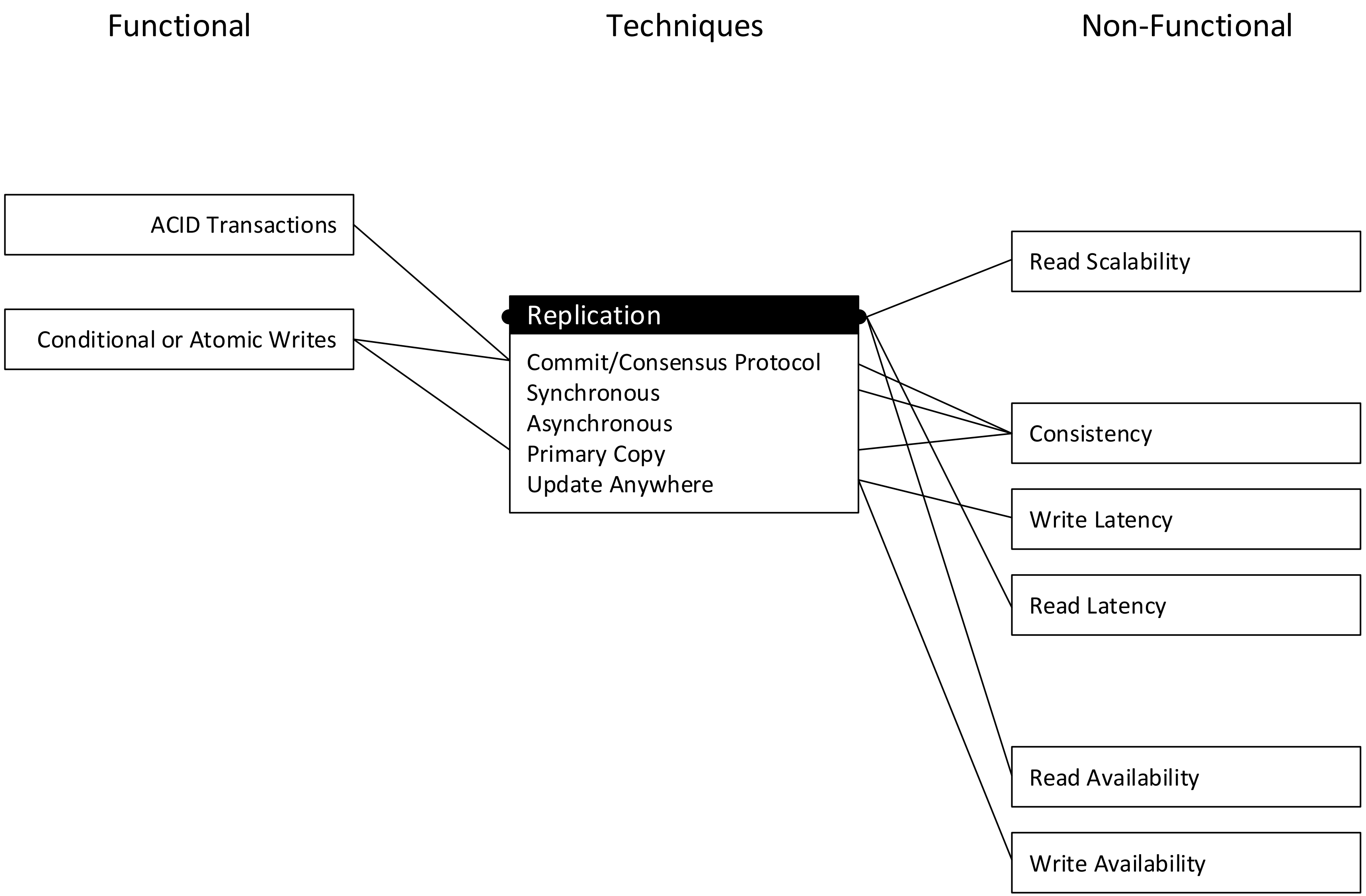
Entity-Group Sharding

- Explicit data co-location for single-node-transactions
- **Pro**: Enables *ACID Transactions*
- **Contra**: Partitioning not easily changable

Implemented in

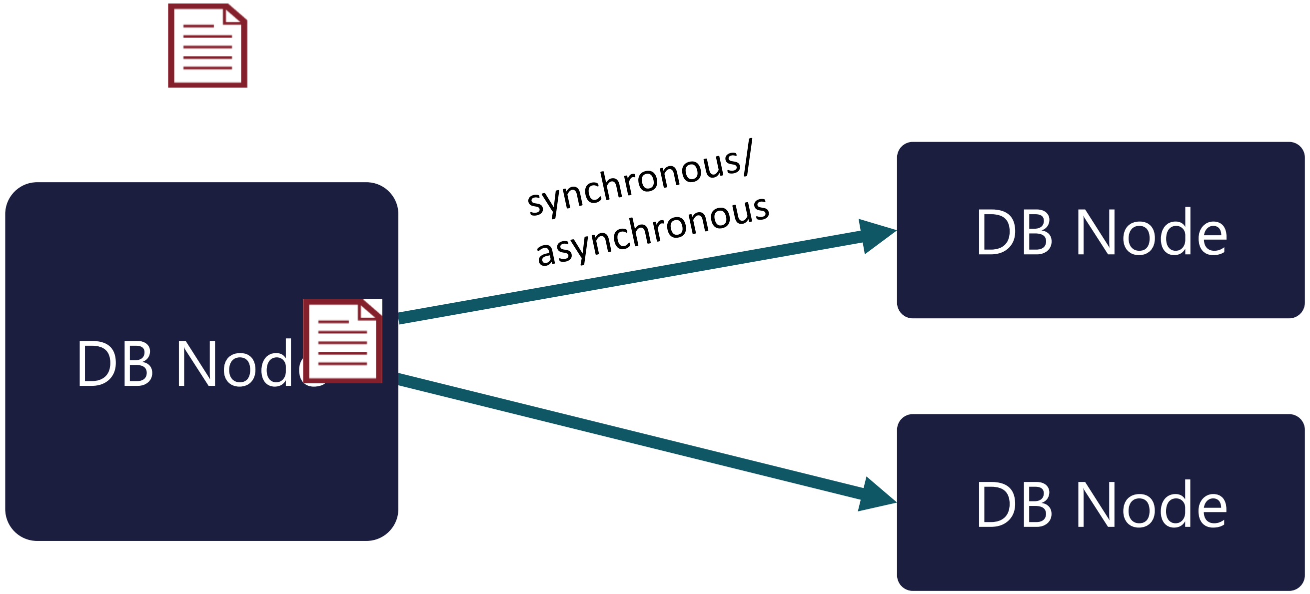
MegaStore, G-Store, Relational Cloud, Cloud SQL Server

David J DeWitt and Jim N Gray: "Parallel database systems: The future of high performance database systems," Communications of the ACM, volume 35, number 6, pages 85–98, June 1992.


Functional
Techniques
Non-Functional
ACID Transactions
Conditional or Atomic Writes
Replication
Commit/Consensus Protocol
Synchronous
Asynchronous
Primary Copy
Update Anywhere
Read Scalability
Consistency
Write Latency
Read Latency
Read Availability
Write Availability

Replication: Read Scalability & Fault Tolerance

- Stores *N* copies of each data item



- **Consistency model**: synchronous vs asynchronous
- **Coordination**: Multi-Master, Master-Slave

Özsu, M.T., Valduriez, P.: Principles of distributed database systems. Springer Science & Business Media (2011)

Replication: When

Asynchronous (lazy)

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

Synchronous (eager)

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

Implemented in

Dynamo , Riak, CouchDB, Redis, Cassandra, Voldemort, MongoDB, RethinkDB

Implemented in

BigTable, HBase, Accumulo, CouchBase, MongoDB, RethinkDB

Charron-Bost, B., Pedone, F., Schiper, A. (eds.): Replication: Theory and Practice, Lecture Notes in Computer Science, vol. 5959. Springer (2010)

Replication: Where

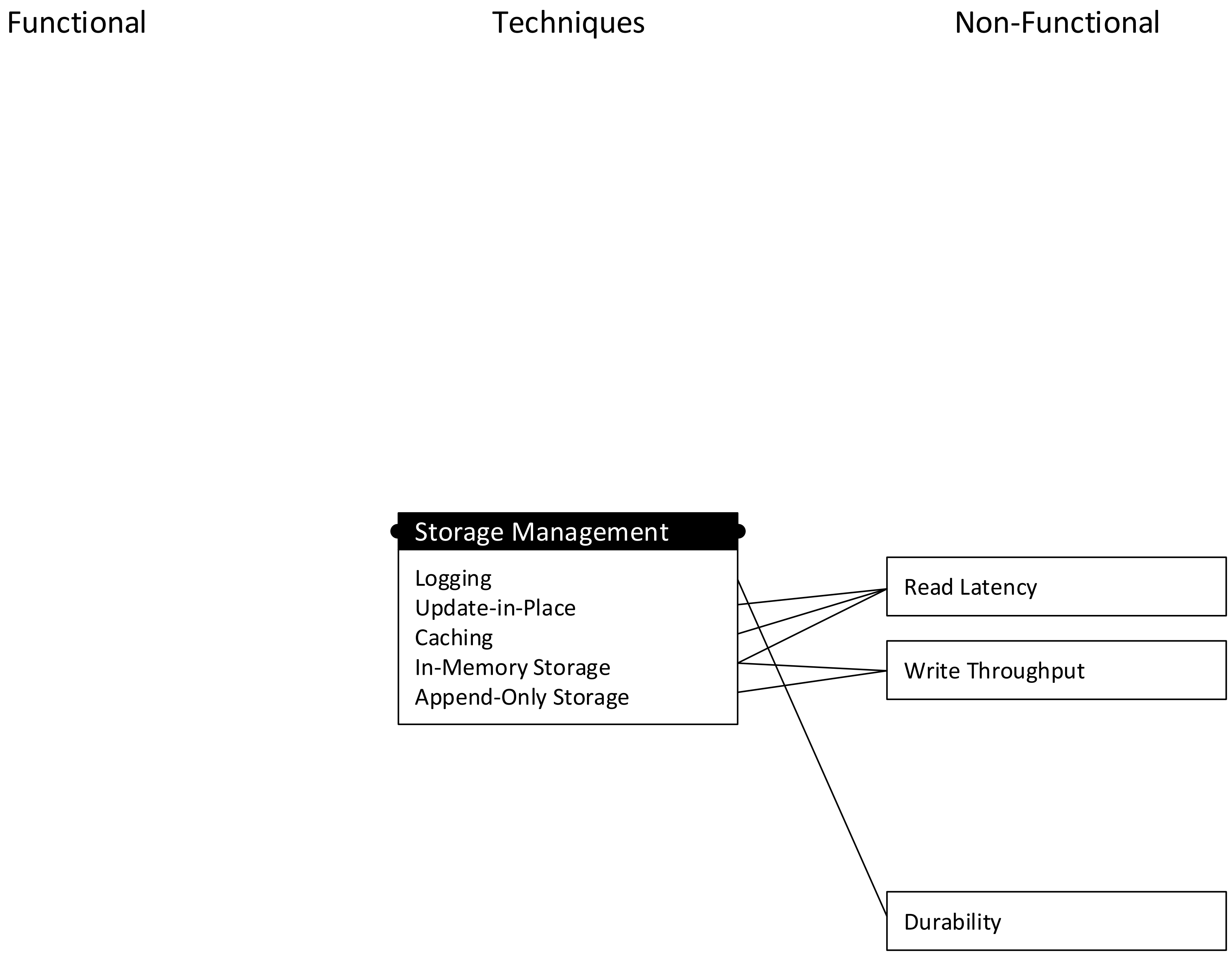
Master-Slave (*Primary Copy*)

- 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

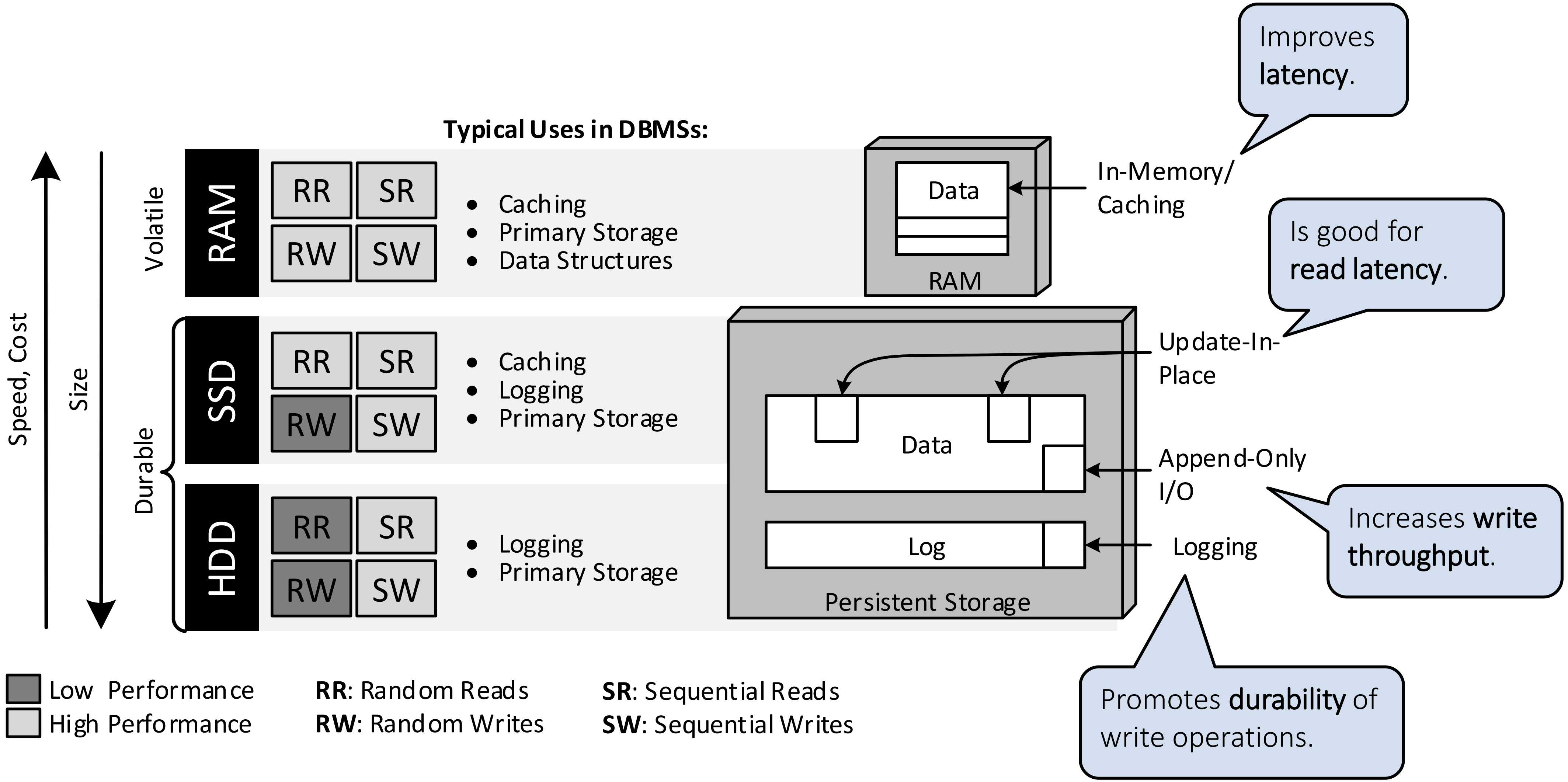
Multi-Master (*Update anywhere*)

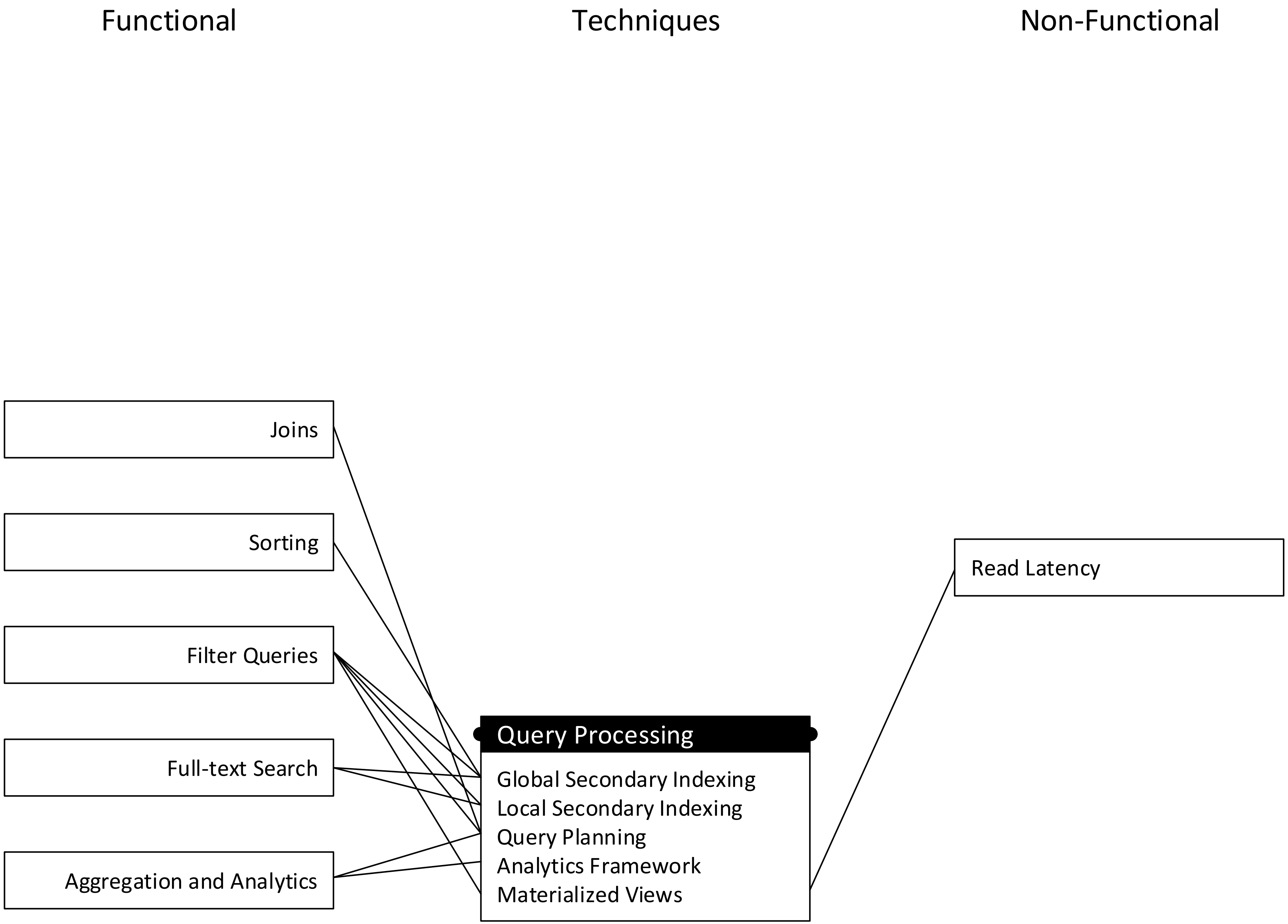
- The server node accepting the writes synchronously propagates the update or transaction before acknowledging
- **Pro**: fast and highly-available
- **Contra**: either needs coordination protocols (e.g. Paxos) or is inconsistent

Charron-Bost, B., Pedone, F., Schiper, A. (eds.): Replication: Theory and Practice, Lecture Notes in Computer Science, vol. 5959. Springer (2010)


Functional
Techniques
Non-Functional
Storage Management
Logging
Update-in-Place
Caching
In-Memory Storage
Append-Only Storage
Read Latency
Write Throughput
Durability

Storage Management




Functional
Techniques
Non-Functional
Joins
Sorting
Filter Queries
Full-text Search
Aggregation and Analytics
Query Processing
Global Secondary Indexing
Local Secondary Indexing
Query Planning
Analytics Framework
Materialized Views
Read Latency

Local Secondary Indexing

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.

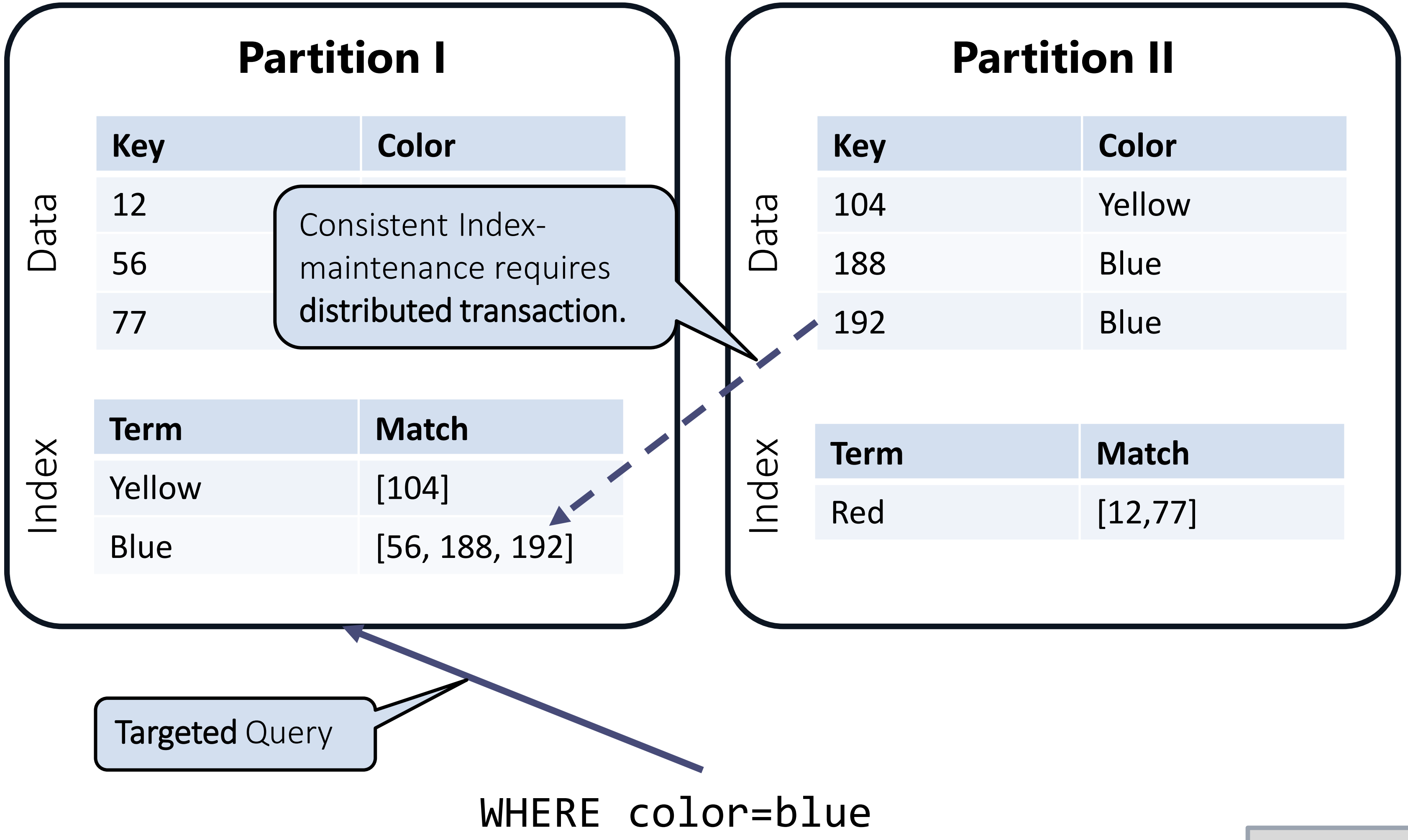
```
WHERE color=blue
```

Implemented in

- MongoDB
- Riak
- Cassandra
- Elasticsearch
- SolrCloud
- VoltDB

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

Global Secondary Indexing



Implemented in

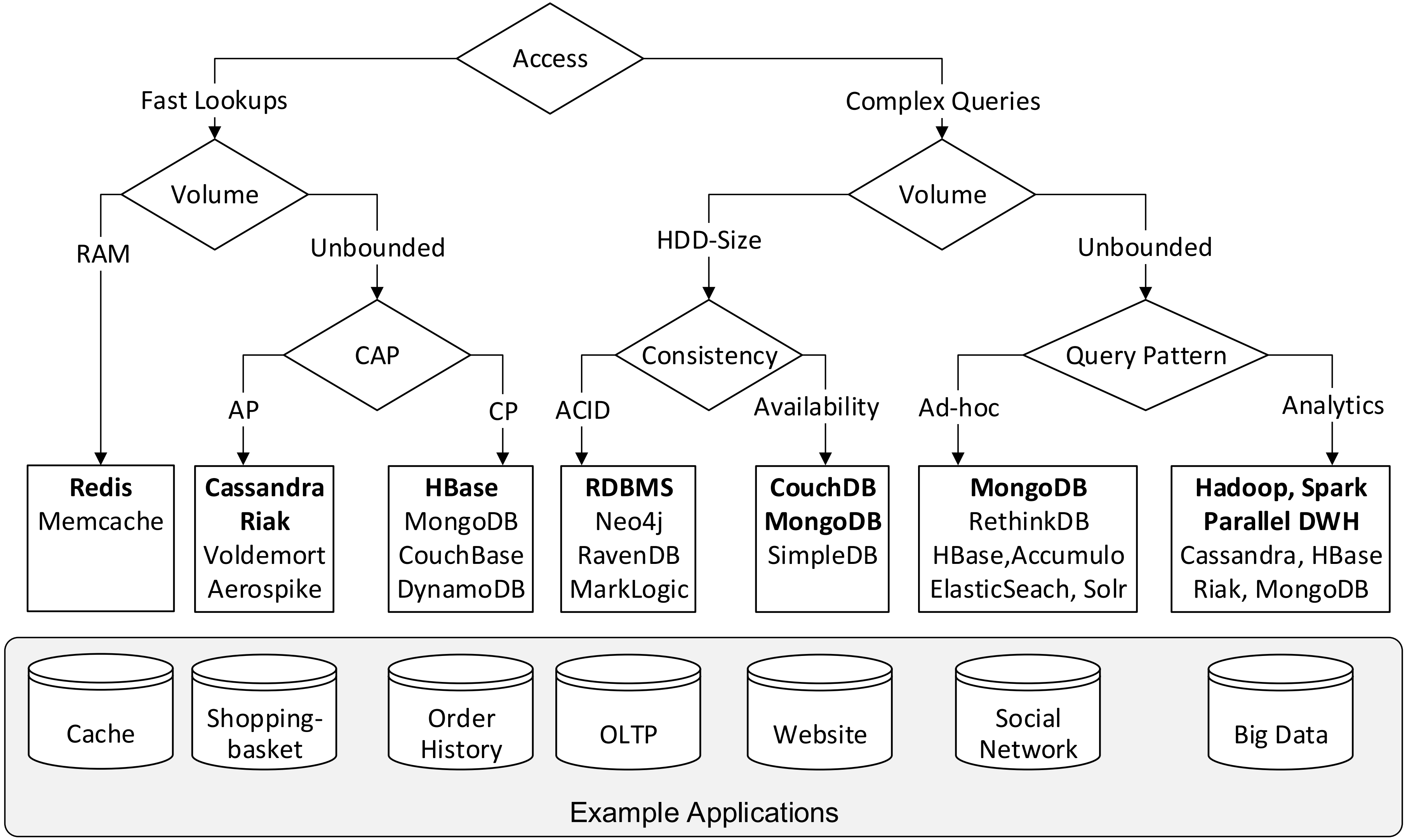
- DynamoDB
- Oracle Datawarehouse
- Riak (Search)
- Cassandra (Search)

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

Query Processing Techniques: Summary

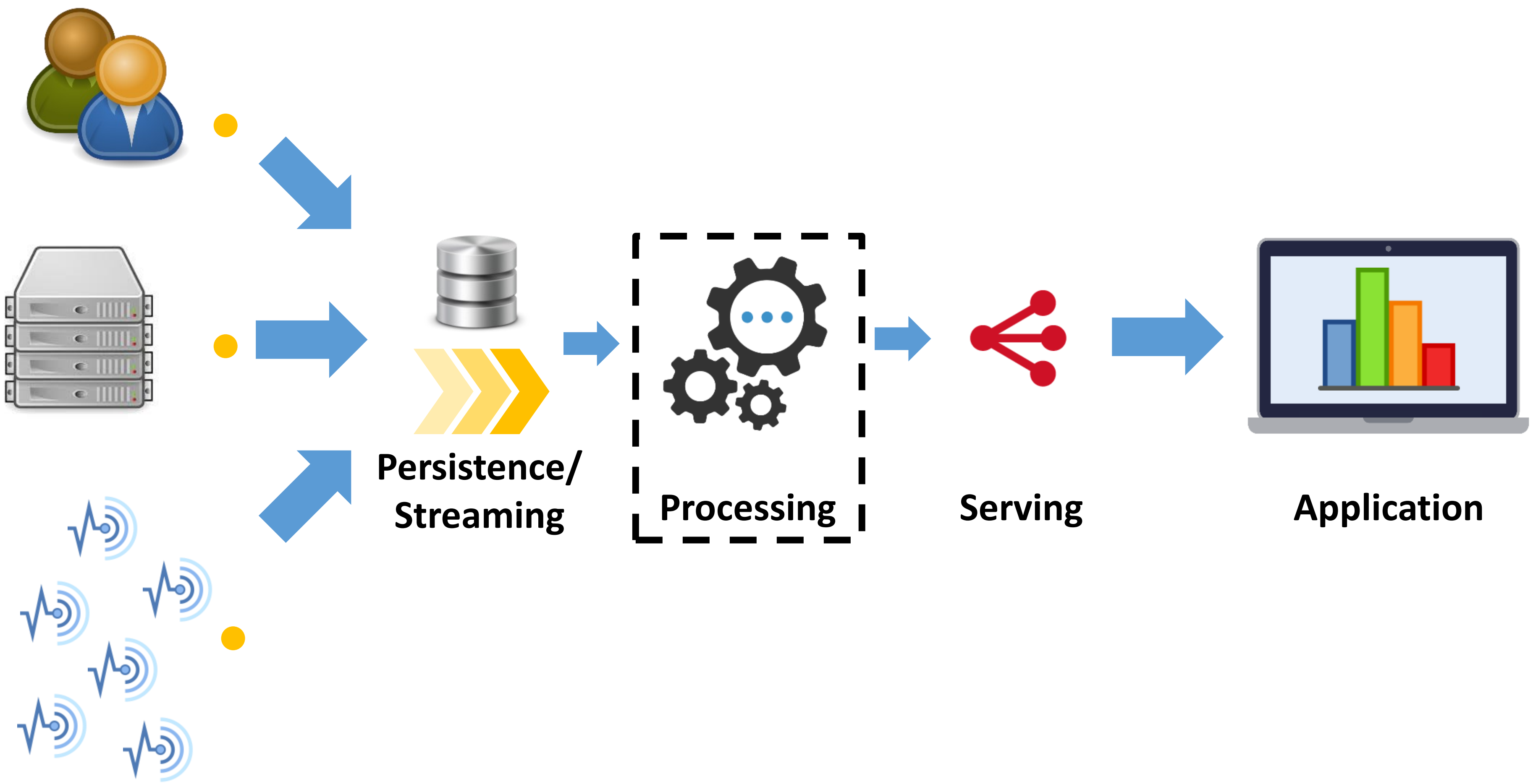
- **Local Secondary Indexing:** Fast writes, scatter-gather queries
- **Global Secondary Indexing:** Slow or inconsistent writes, fast queries
- **(Distributed) Query Planning**: scarce in NoSQL systems but increasing (e.g. left-outer equi-joins in MongoDB and θ-joins in RethinkDB)
- **Analytics Frameworks**: fallback for missing query capabilities
- **Materialized Views**: similar to global indexing

NoSQL Decision Tree



What About Push-Based Systems?

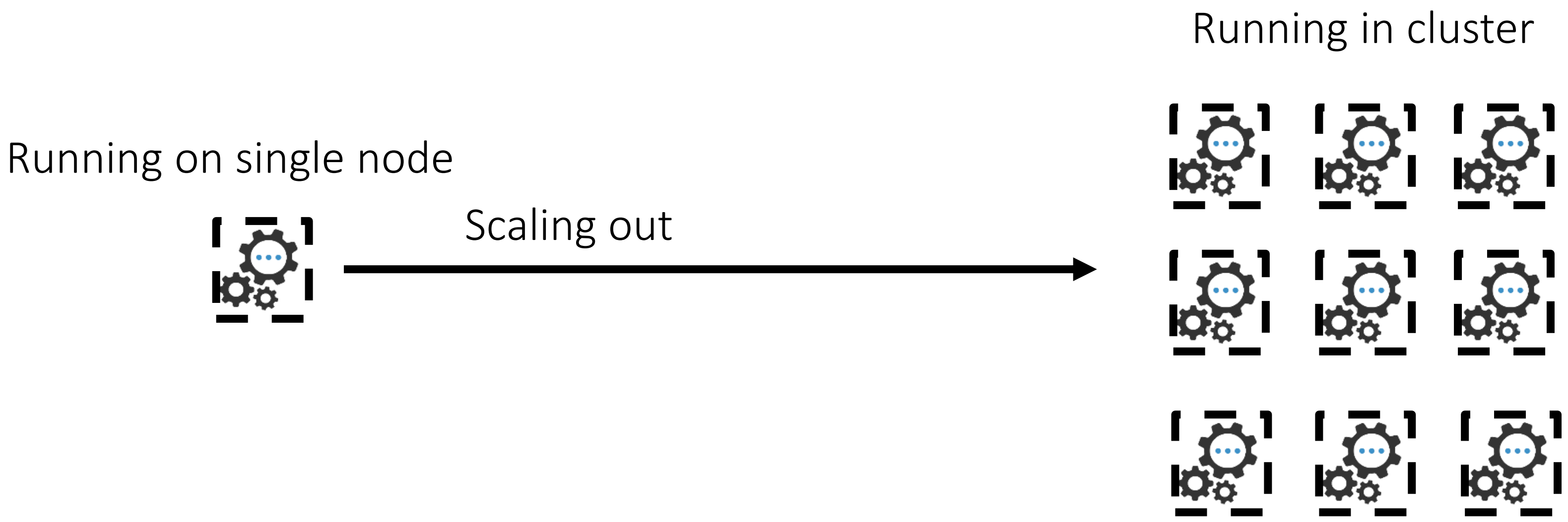
A Data Processing Pipeline



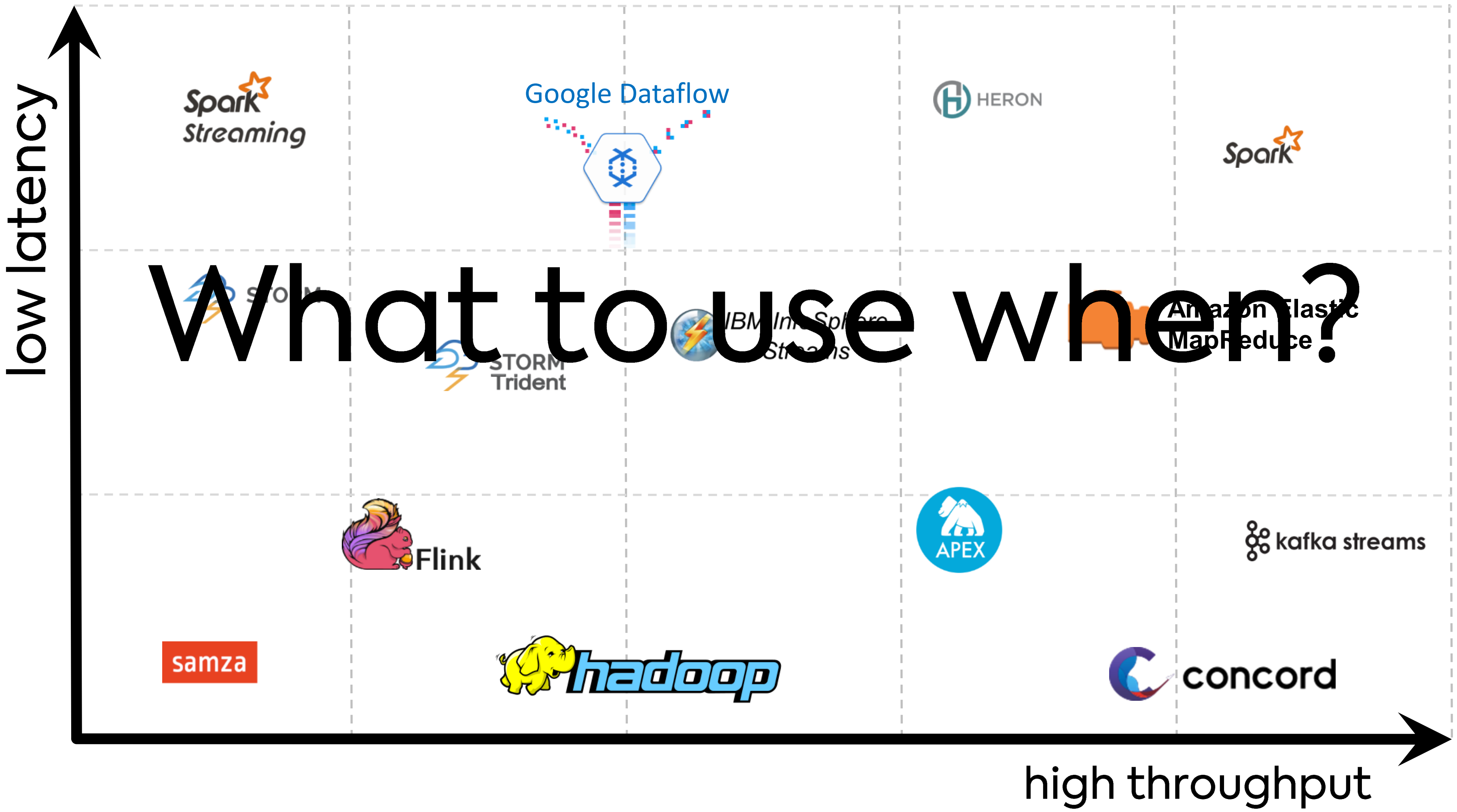
Scale-Out Made Feasible

Data processing frameworks **hide complexities of scaling**, e.g.:

- **Deployment -** code distribution, starting/stopping work
- **Monitoring -** health checks, application stats
- **Scheduling -** assigning work, rebalancing
- **Fault-tolerance** - restarting workers, rescheduling failed work



Big Data Processing Frameworks



Big Data Processing Models

stream	micro-batch	batch
Flink		hadoop
STORM	STORM Trident, Spark Streaming	Spark
samza		Amazon Elastic MapReduce

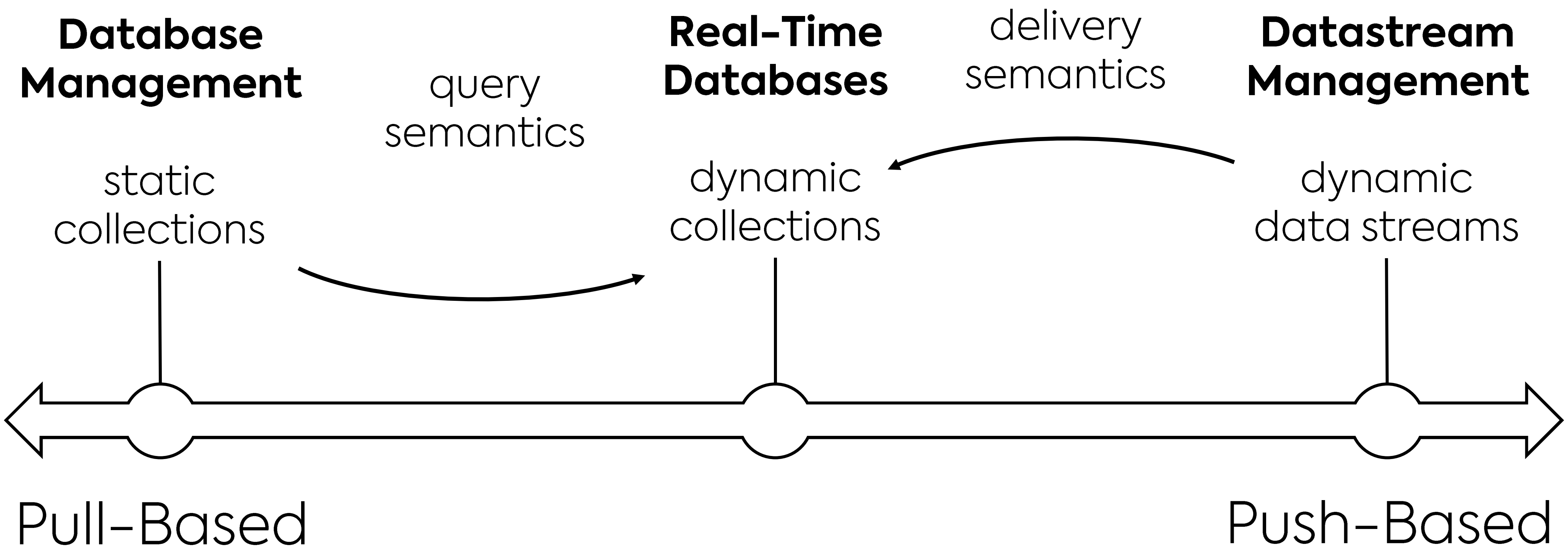
low latency ⟷ high throughput

Stream Processing System Comparison

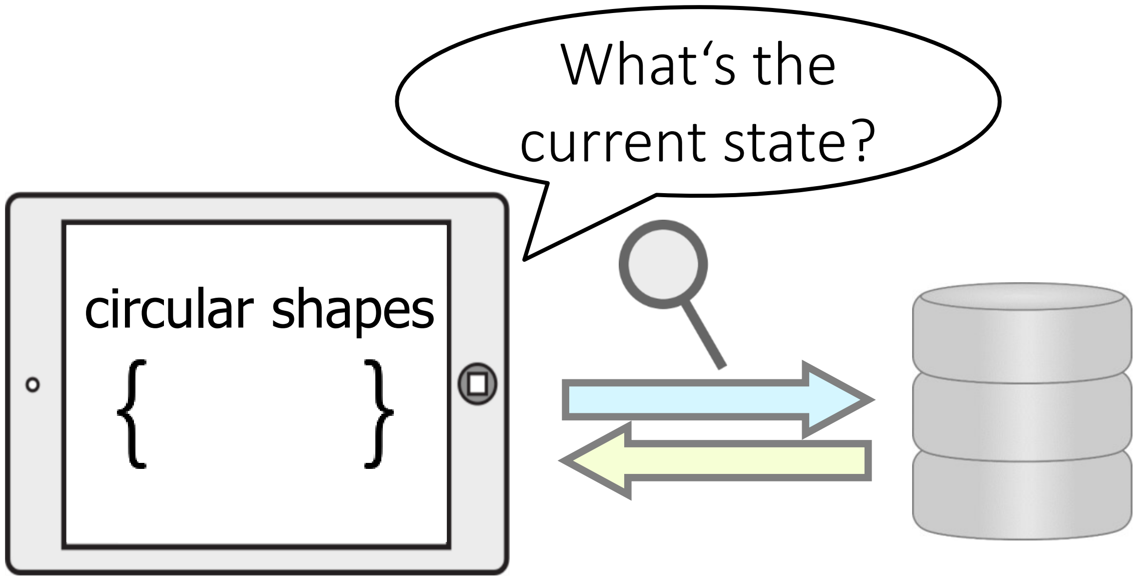
	Storm	Trident	Samza	Spark Streaming	Flink (streaming)
Strictest Guarantee	at-least-once	exactly-once	at-least-once	exactly-once	exactly-once
Achievable Latency	≪100 ms	<100 ms	<100 ms	<1 second	<100 ms
State Management	○ (small state)	○ (small state)	✓	✓	✓
Processing Model	one-at-a-time	micro-batch	one-at-a-time	micro-batch	one-at-a-time
Backpressure	✓	✓	no (buffering)	✓	✓
Ordering	✗	between batches	within partitions	between batches	within partitions
Elasticity	✓	✓	✗	✓	✗

Real-Time Databases: Combining Push & Pull

Push vs. Pull: Trade-Offs in Data Management



Traditional Databases: No Request, No Data!

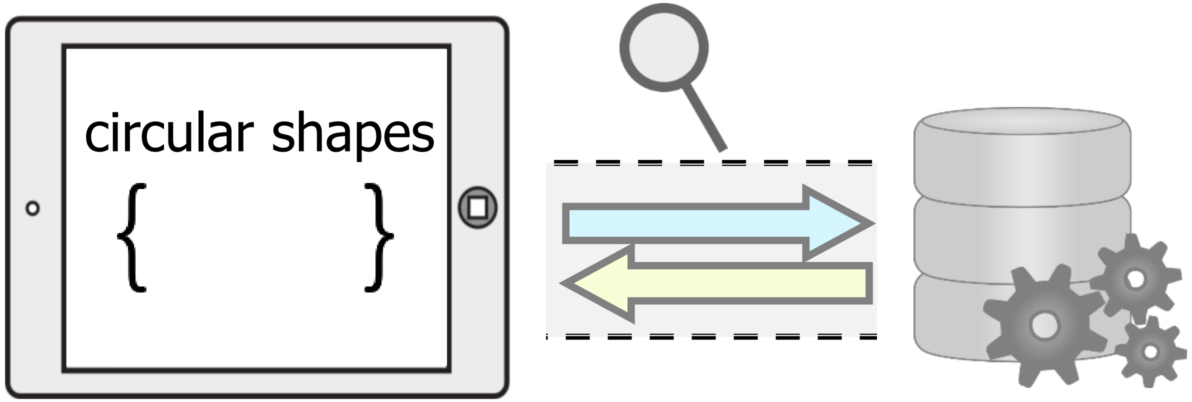


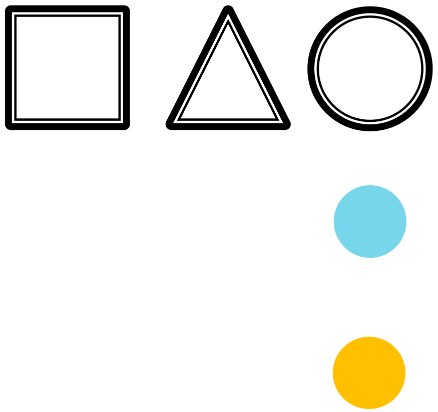
Periodic Polling for query result maintenance:

→ inefficient

→ slow

Real-Time Databases: Always Up-to-Date





Real-Time Queries for query result maintenance:

→ efficient

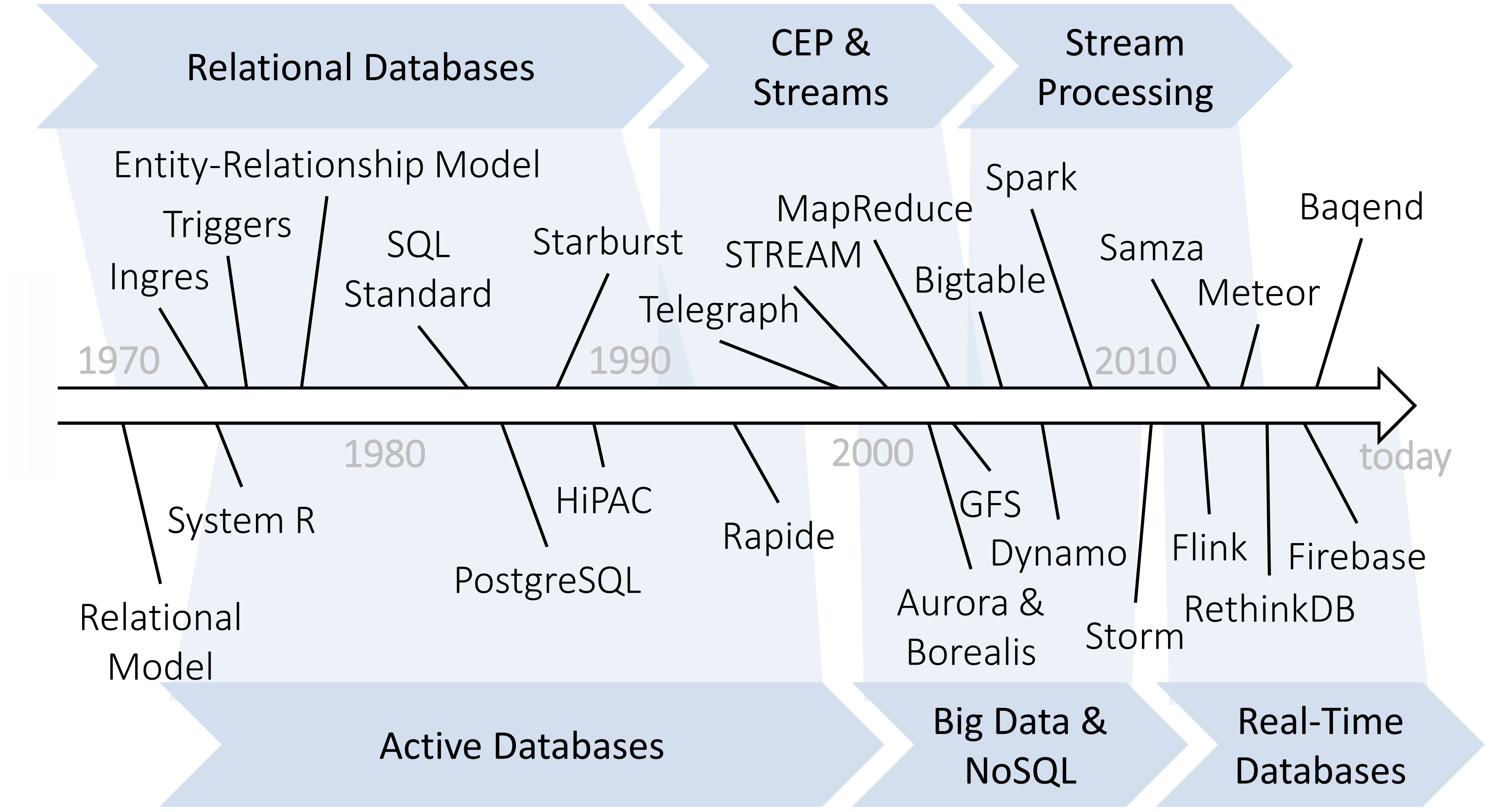
→ fast

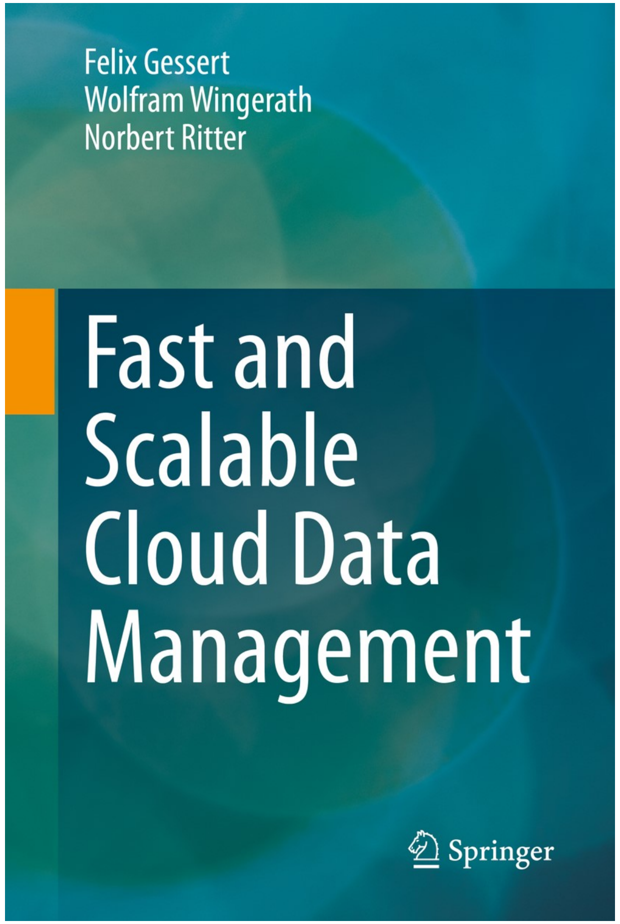
Real-Time Database Comparison

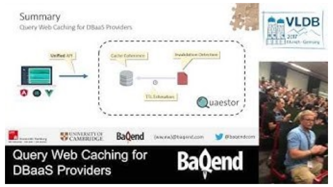
	Meteor	RethinkDB	Parse	Firebase
	Poll-and-Diff	Change Log Tailing		Unknown
Write Scalability	✓	✗	✗	✗	✗
Read Scalability	✗	✓	✓	✓	? (100k connections)
Composite Filters (AND/OR)	✓	✓	✓	✓	○ (AND In Firestore)
Sorted Queries	✓	✓	✓	✗	○ (single attribute)
Limit	✓	✓	✓	✗	✓
Offset	✓	✓	✗	✗	○ (value-based)
Self-Maintaining Queries	✓	✓	✗	✗	✗
Event Stream Queries	✓	✓	✓	✓	✓

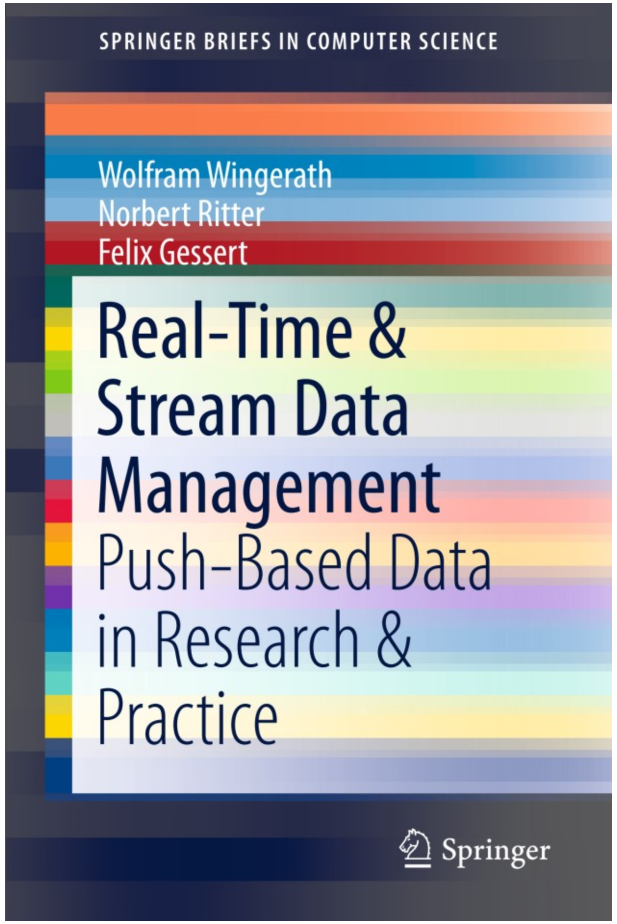
W. Wingerath, F. Gessert, N. Ritter: InvaliDB: *Scalable Push-Based Real-Time Queries on Top of Pull-Based Databases (Extended)*, VLDB 2020

Wrapup: A Short History of Data Management









For videos & books,
visit slides.baqend.com!

Tutorial 18: Going for Speed

09:00	Introduction
09:15	Part 1: Efficient Frontend Design
09:40	Q&A
09:50	Coffee Break
10:00	Part 2: High-Performance Networking
10:40	Q&A
10:50	Coffee Break
11:00	Part 3: Scalable Backend Architectures
11:40	Q&A
11:50	Coffee Break
12:00	**Part 4: Performance Tracking & Analysis**
12:00	The Core Web Vitals
12:30	Measuring Web Performance
12:50	Q&A

Google Guest Speaker!

Up next!