Tutorial on

NoSQL Data Management: Concepts and Systems

Holger Schwarz
Matthias Wieland
IPVS, Universität Stuttgart, Germany

SummerSOC 2015

Overview

- Introduction to NoSQL
- Basic Concepts for NoSQL
- Overview of NoSQL Systems

History of NoSQL

- SQL Databases were dominant for decades
  - Persistent storage
  - Standards based
  - Concurrency Control
  - Application Integration
  - ACID
- → Designed to run on a single „big“ machine

- **Cloud computing** changes that dramatically
  - Cluster of machines
  - Large amount of unreliable machines
  - Distributed System
  - Schema-free unstructured Big Data

Methods to Run a Database

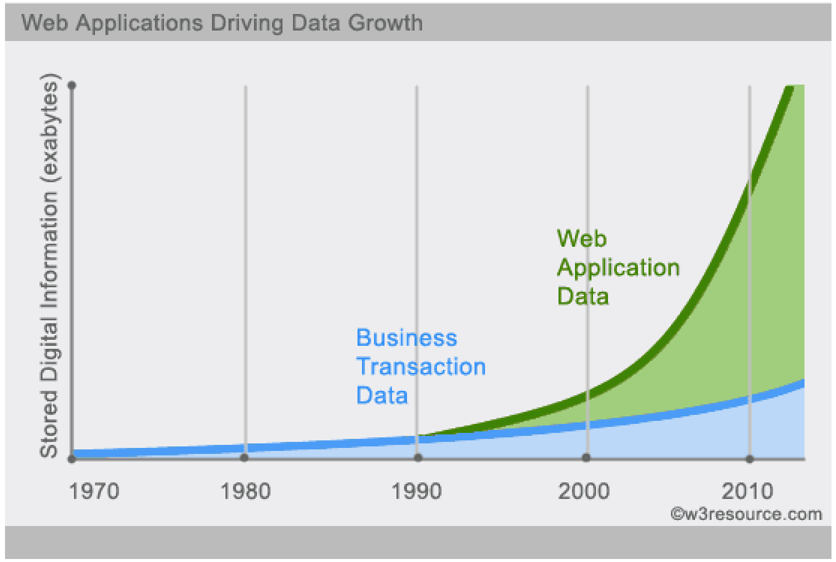
- Virtual Machine Image
  - Users purchase virtual machine instances to run a database on these
  - Upload and setup own image with database, or use ready-made images with optimized database installations
  - E.g. Oracle Database 11g Enterprise Edition image for Amazon EC2 and for Microsoft Azure.
- Database as a service (DBaaS)
  - Using a database without physically launching a virtual machine instance
  - No configuration or management needed by application owners
  - E.g. Amazon Web Services provide SimpleDB, Amazon Relational Database Service (RDS), DynamoDB,
- Managed database hosting
  - Not offered as a service, but hosted and managed by the cloud database vendor
  - E.g. Rackspace offers managed hosting for MySQL
- The TOSCA way
  - Description of Cloud Services as Topology combined with the database stack
  - Vendor-neutral automatic provisioning and management with OpenTOSCA
  - Policies to define security requirements of the Cloud Service
  - Portable and interoperable definition of data security and compliance aspects

Which Data Model?

- Relational Databases
  - Standard SQL database available for Cloud Environments as Virtual Machine Image or as a service depending on the vendor
  - Not cloud-ready: Difficult to scale
- NoSQL databases
  - Database which is designed for the cloud
  - Built to serve heavy read/write loads
  - Good ability to scale up and down
  - Applications built based on SQL data model require a complete rewrite
  - E.g. Apache Cassandra, CouchDB and MongoDB

How to scale the data management?

- Vertical scaling – Scale up

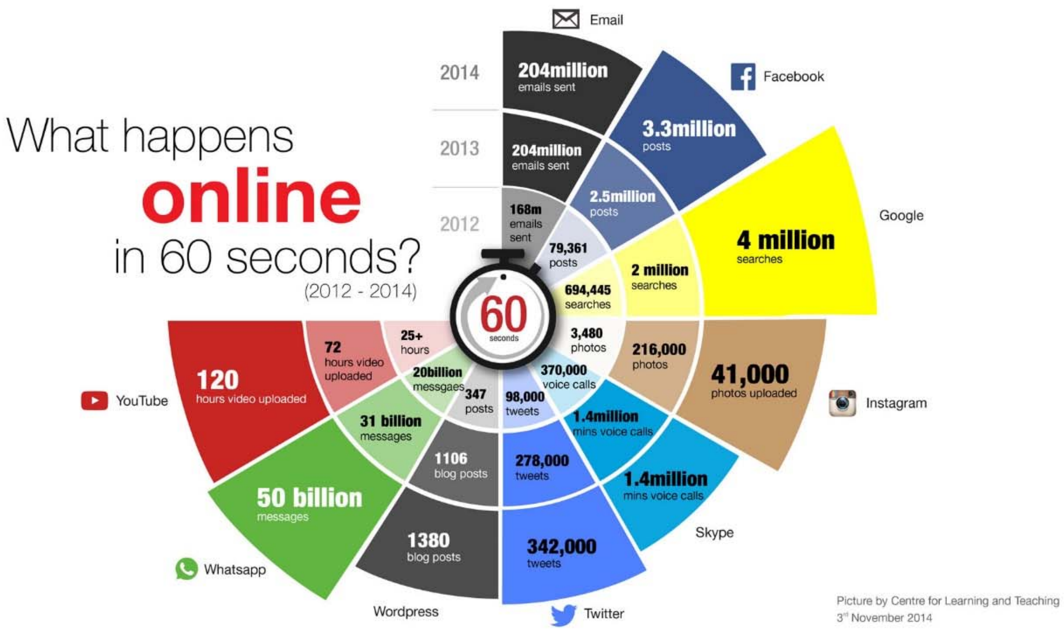


- Horizontal scaling – Scale out

Definition and Goals of NoSQL databases

- No formal NoSQL definition available!
- Store very large scale data called “Big data”
- Typically scale horizontally
- Simple query mechanisms
- Often designed and set up for a concrete application

- Typical characteristics of NoSQL databases are:
  - **Non-relational**
  - **Schema-free**
  - **Open Source**
  - **Simple API**
  - **Distributed**
  - **Eventual consistency**



Source: https://clt.vtc.edu.hk/what-happens-online-in-60-seconds/

Non-relational

- NoSQL databases generally do not follow the relational model
- Do not provide tables with flat fixed-column records
- Work with self-contained (hierarchical) aggregates or BLOBs
- No need for object-relational mapping and data normalization
- No complex and costly features like query languages, query planners, referential integrity, joins, ACID

Schema-free

- Most NoSQL databases are schema-free or have relaxed schemas
- No need for definition of any sort of schema of the data
- Allows heterogeneous structures of data in the same domain

Simple API

- Often simple interfaces for storage and querying data provided
- APIs often allow low-level data manipulation and selection methods
- Often no standard based query language is used
- Text-based protocols often using HTTP REST with JSON
- Web-enabled databases running as internet-facing services

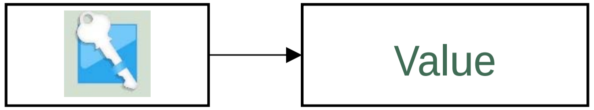
Distributed

- Several NoSQL databases can be executed in a distributed fashion
- Providing auto-scaling and fail-over capabilities
- Often ACID is sacrificed for scalability and throughput
- Often no synchronous replication between distributed nodes is possible, e.g. asynchronous Multi-Master Replication, peer-to-peer, HDFS Replication
- Only providing eventual consistency

Core Categories of NoSQL Systems

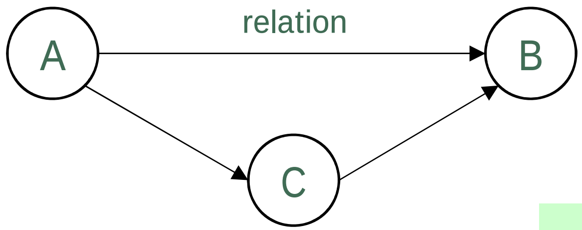
- **Key-Value Stores**
  - Manage associative arrays
  - Big hash table
- **Wide Column Stores**
  - Each storage block contains only data from one column
  - Read and write is done using columns (rather than rows – like in SQL)
- **Document Stores**
  - Store documents consisting of tagged values
  - Data is a collection of key value pairs
  - Provides structure and encoding of the managed data
  - Encoded using XML, JSON, BSON
  - Schema-free
- **Graph DB**
  - Network database using graphs with node and edges for storage
  - Nodes represent entities, edges represent their relationships

- Other NoSQL systems
  - ObjectDB
  - XML DB
  - Special Grid DB
  - Triple Store



Row ID	Columns...		
1	Name	Website	
	Preston	www.example.com	
2	Name	Website	
	Julia	www.example.com	
3	Name	Email	Website
	Alice	example@example.com	www.example.com

```
{
  “id”: 123,
  “name”:”Matthias”,
  “location”:{
          “city”:”Hersonissos”,
          “region”:”Crete”
  }
}
```



Usage of NoSQL in Practice

- Google
  - Big Table
  - Google Apps, Google Search
- Facebook
  - Social network
- Twitter
- Amazon
  - DynamoDB and SimpleDB
- CERN
- GitHub

Overview

- Introduction to NoSQL
- Basic Concepts for NoSQL
  - CAP-Theorem
  - Eventual Consistency
  - Consistent Hashing
  - MVCC-Protocol
  - Query Mechanisms for NoSQL
- Overview of NoSQL Systems

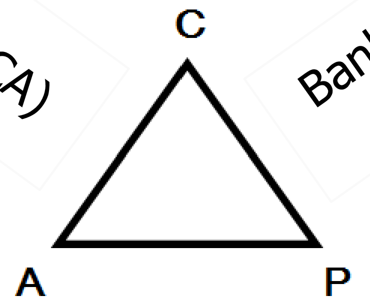
CAP Theorem – Brewer's theorem

- States that it is impossible for a distributed system to provide all three of the following guarantees
  - **C̲onsistency**: all nodes see the same data at the same time
  - **A̲vailability**: every request receives a response about whether it succeeded or failed
  - **P̲artition tolerance**: the system continues to operate despite physical network partitions

Choose two!

SQL database (CA)

Banking application (CP)



NoSQL (AP)

Domain Name System DNS (AP)

Cloud Computing (AP)

Eventual consistency and BASE

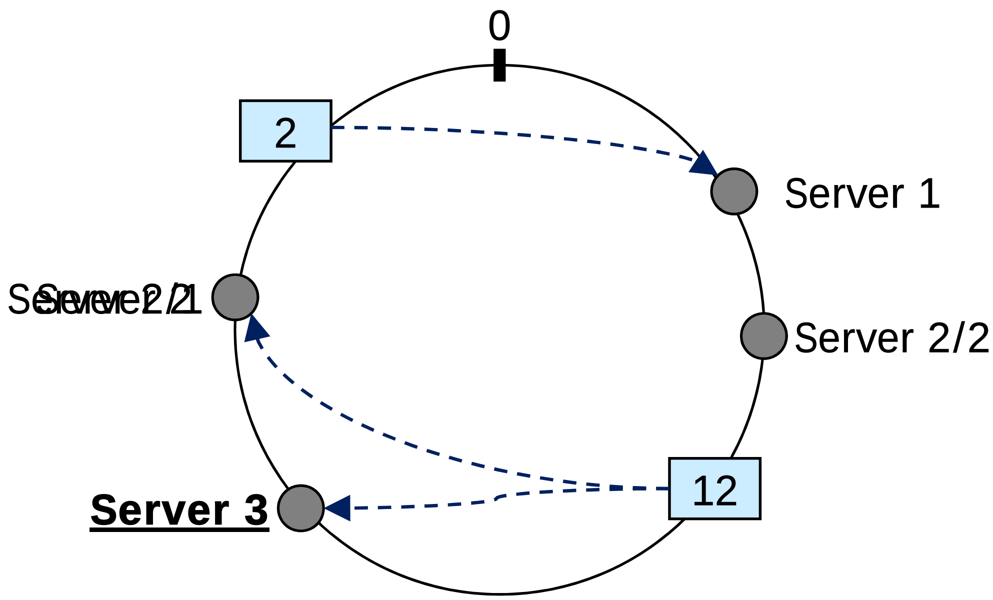
- **The term "eventual consistency"**
  - Copies of data on multiple machines to **achieve high availability and scalability**
  - A change to a data item on one machine has to be propagated to other replicas
  - Propagation is not instantaneous so the copies will be **mutually inconsistent**
  - The change will eventually be propagated to all the copies
  - Fast access requirement dominates
  - Different replicas can return different values for the queried attribute of the object
  - A System that achieved eventual consistency **converged, or achieved replica convergence**

- **Eventual consistency guarantees:**
  - If no new updates are made to a given data item eventually all accesses to that item will return the last updated value

- Eventually consistent services **provide weak consistency using BASE**
  - **B**asically **A**vailable, **S**oft state, **E**ventual consistency
  - **Basically available** indicates that the system guaranteed availability (CAP theorem)
  - **Soft state** indicates that the state of the system may change over time, even without input
  - **Eventual consistency** indicates that the system will become consistent over time

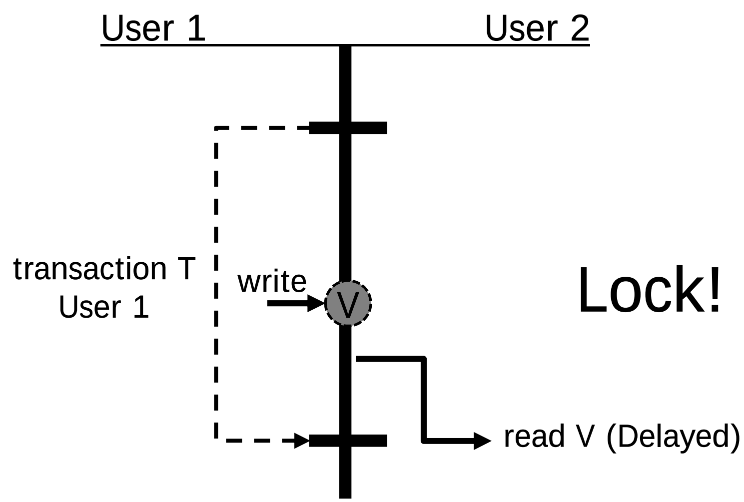
Consistent Hashing

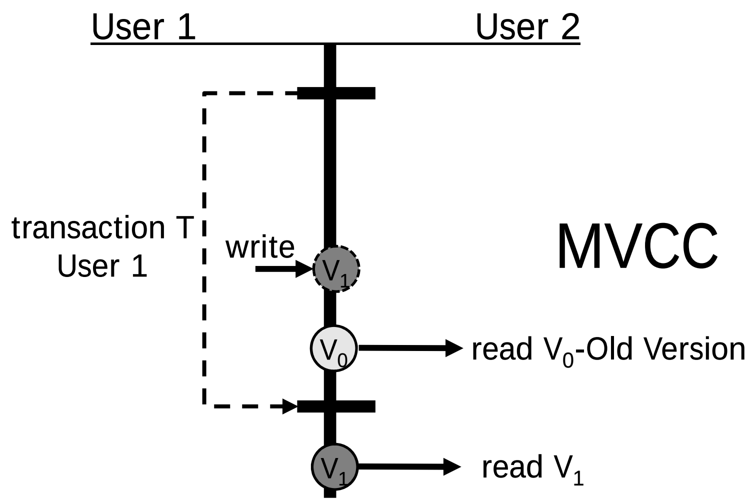
- Technique how to efficiently distribute replicas to nodes
- Consistent hashing is a special kind of hashing
  - When hash table is resized only K/n keys need to be remapped on average
  - K is the number of keys, and n is the number of slots
  - In traditional hash tables nearly all keys have to be remapped
- Insert Servers on ring
  - Hash based e.g. on IP, Name, ...
  - Take over objects between own and processor hash
- Insert Objects on ring
  - Hash based on key
  - Walks around the circle until falling into the first bucket
- Delete Servers
  - Copy objects to next server
- Virtual Servers
  - More than one hash per server
- Replication
  - Place objects multiple times
  - Improves reliability



Multiversion Concurrency Control (MVCC)

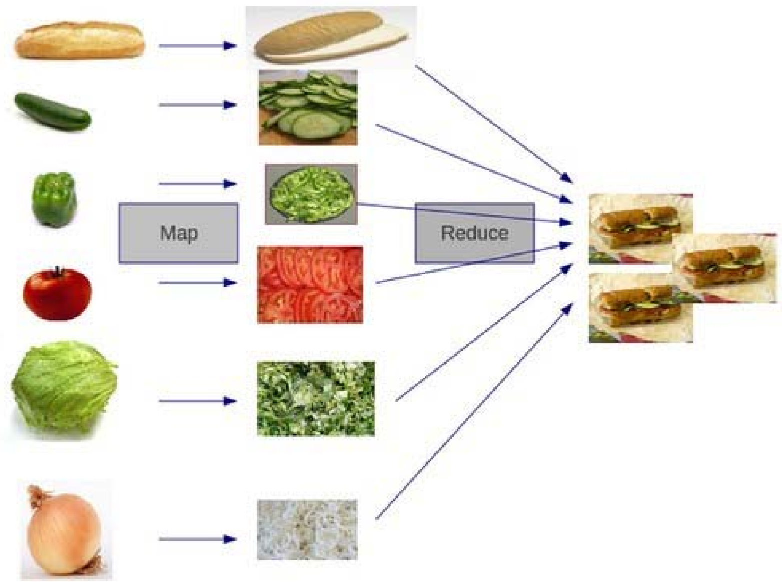
- **Concurrency control method** to provide concurrent access to the database
- **LOCKING**
  - All readers wait until the writer is done
  - **This can be very slow!**
- **MVCC**
  - An write adds a new version
  - Read is always possible
  - Any changes made by a writer will not be seen by other users of the database until they have been committed
  - Conflicts (e.g. V_{1a} ,V_{1b}) can occur and have to be handled





Query Mechanisms for NoSQL

- REST based retrieval of a value based on its key/ID with GET resource
- Document stores allow more complex queries
  - E.g. CouchDB allows to define views with MapReduce
- MapReduce
  - Available in many NoSQL databases
  - Can run fully distributed
  - It is Functional Programming, not writing queries!
  - **Map phase** - perform filtering and sorting
  - **Reduce phase** - performs a summary operation
  - ~ SELECT and GROUP BY of a relational database
  - More details later!
- Apache Spark is an open source big data processing framework providing more operations than MapReduce
- Example use cases for MapReduce
  - Distributed Search
  - Counting – URL, Words
  - Building linkage graphs for web sites
  - Sorting distributed data



Source: @tgrall

Overview

- Introduction to NoSQL
- Basic Concepts for NoSQL
- Overview of NoSQL Systems
  - Key-Value Stores
  - Document Stores
  - Wide-column stores
  - Graph Stores
  - Hadoop Map/Reduce and more ...

Key Value Stores

Memcached

- Developer: Basho Technologies (http://basho.com/)
- Current version: 2.1.1
- Available since: 2009
- Licence: Apache licence 2.0
- Supported operating systems: Linux, BSD, Mac OS, Solaris
- Client libraries for:
  - Java, Ruby, Python, C#, Erlang (the official ones)
  - C, Node.js, PHP (the unofficial ones)
  - even more form the Riak community

Typical Use Cases

- Session data

```
sessionid=A08154711
userlogin="xyz"
date_of_expiry=2015/12/31
```

id

- User profiles

```
{
  "id": "4",
  "name": "Mark Zuckerberg",
  "first_name": "Mark",
  "last_name": "Zuckerberg",
  "link": "http://www.facebook.com/zuck",
  "username": "zuck",
  "gender": "male",
  "locale": "en_US"
}
```

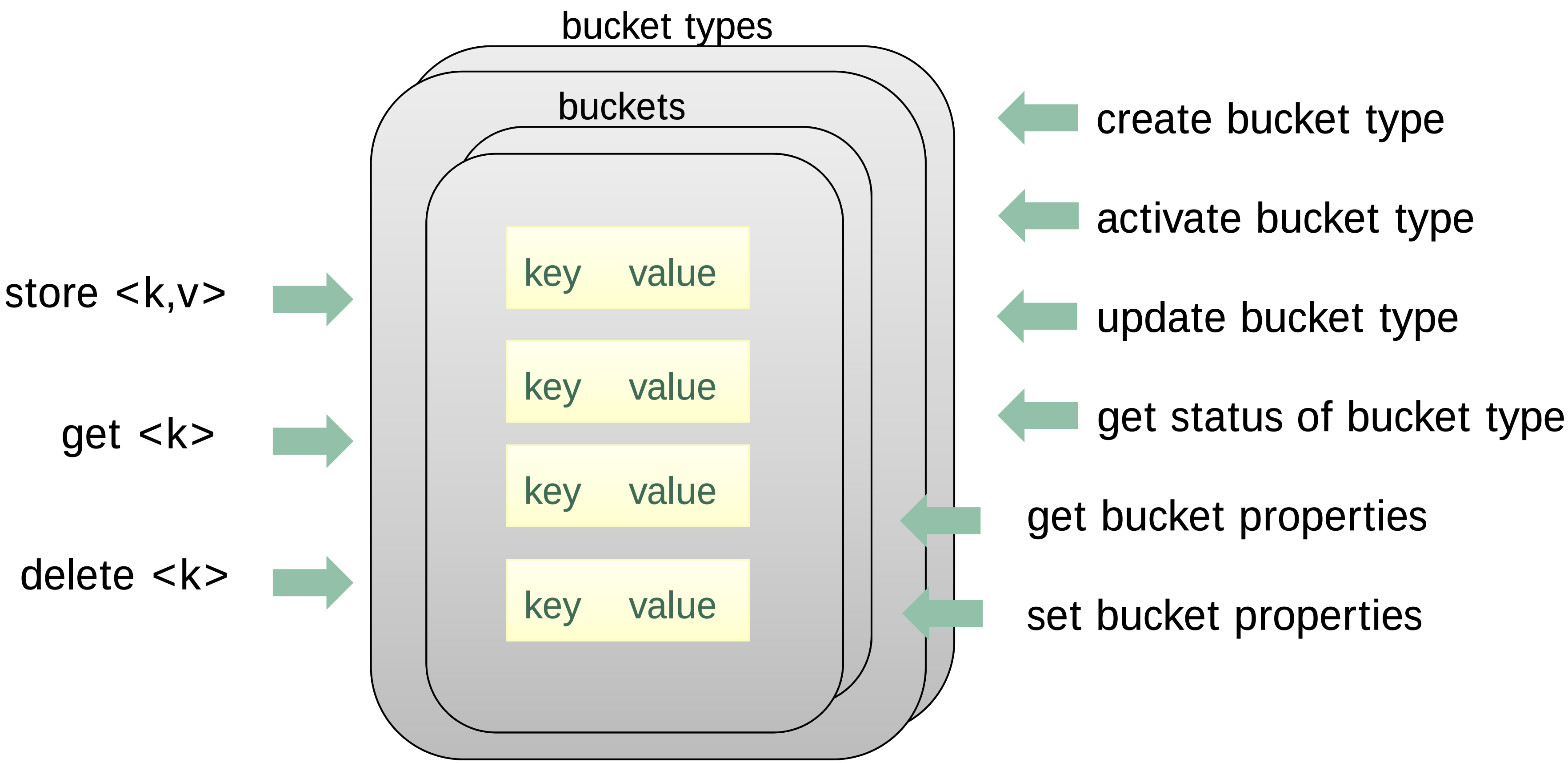
User profile id

- Sensor data (IOT)

key | value

timestamp	x	y	z	temperature
01.01.2014	350	120	78	-10°
01.01.2014	350	120	95	-9
01.01.2014	350	100	78	-10°
02.01.2014	350	120	78	-5°
02.01.2014	350	120	95	-8°
...				

Key Functionality

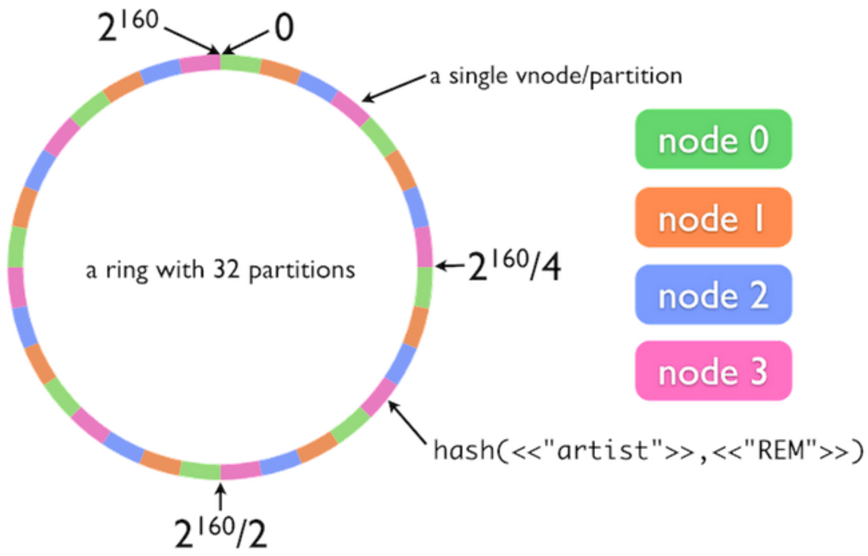


riak

Instances and Vnodes

- Riak runs on potentially large clusters
- Each host in the cluster runs a single **instance** of Riak (Riak node)
- Each Riak node manages a set of virtual node (**vnodes**)

- Mapping of <bucket,key> pairs
  - compute 160-bit hash
  - map result to a ring position
  - ring is divided into partitions
  - each vnode is responsible for one partition

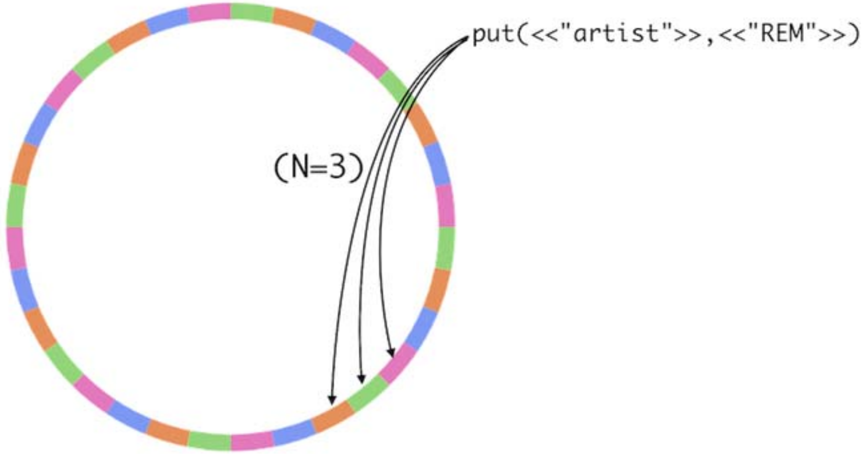


docs.basho.com

riak

Configure Replication

- Some bucket parameters
  - **N**: replication factor
  - **R**: number of servers that must respond to a read request
  - **W**: number of servers that must respond to a write request
  - **DW**: number of servers that must report that a write has successfully been written to disk
  - ...
  - Parameters allow to trade consistency vs. availability vs. performance



docs.basho.com

riak Transactions and Consistency

- No multi-operation transactions are supported
- Eventual consistency is default (and was the only option before Riak 2.0)
  - Vector clocks and Dotted Version Vectors (DVV) used to resolve object conflicts
- Strong consistency as an alternative option
  - A quorum of nodes is needed for any successful operation

Riak Search

- For Raik KV, a value is just a value possibly associated with a type
- Riak Search 2.0
  - Based on Solr, the search platform built on Apache Lucene
  - Define extractors, i.e., modules responsible for pulling out a list of fields and values from a Riak object
  - Define Solr schema to instruct Riak/Solr how to index a value
  - Queries:
    exact match, globs, inclusive/exclusive range queries, AND/OR/NOT, prefix matching, proximity searches, term boosting, sorting, ...

Document Stores

- Developer: MongoDB, Inc. (https://www.mongodb.com/)
- Available since: 2009
- Licence: GNU AGPL v3.0
- Supported operating systems: all major platforms
- Drivers for:
  - C, C++, C#, Java, Node.js, Perl, PHP, Python, Motor, Ruby, Scala, ...

What are Documents?

- Aggregated data
- No fixed schema
- Internal structure matters
- Format: JSON, BSON, ...

```
{
    "firstName": “Paul",
    "lastName": “Adam",
    "age": 45,
    "address":
    {
        "streetAddress": "22 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    }
}
```

```
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address":
    {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    },
    "phoneNumbers":
    [
        {
          "type": "home",
          "number": "212 555-1234"
        },
        {
          "type": "fax",
          "number": "646 555-4567"
        }
    ]
}
```

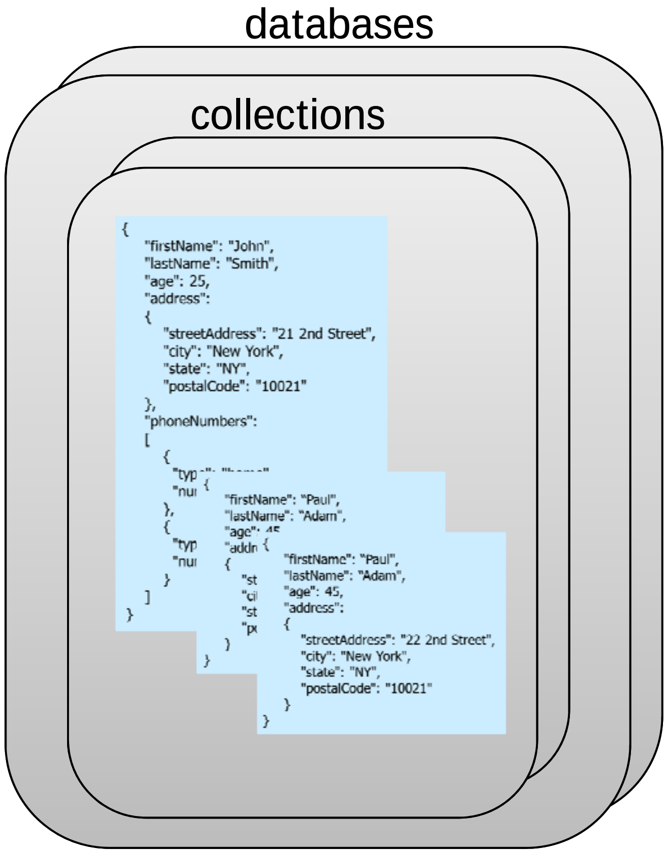
Typical Use Cases

- Simple content management
  - e.g. blogging platforms
- Logging events
  - cope with event type heterogeneity
  - data associated with events changes over time
- E-Commerce applications
  - flexible schema for product and order data

mongoDB

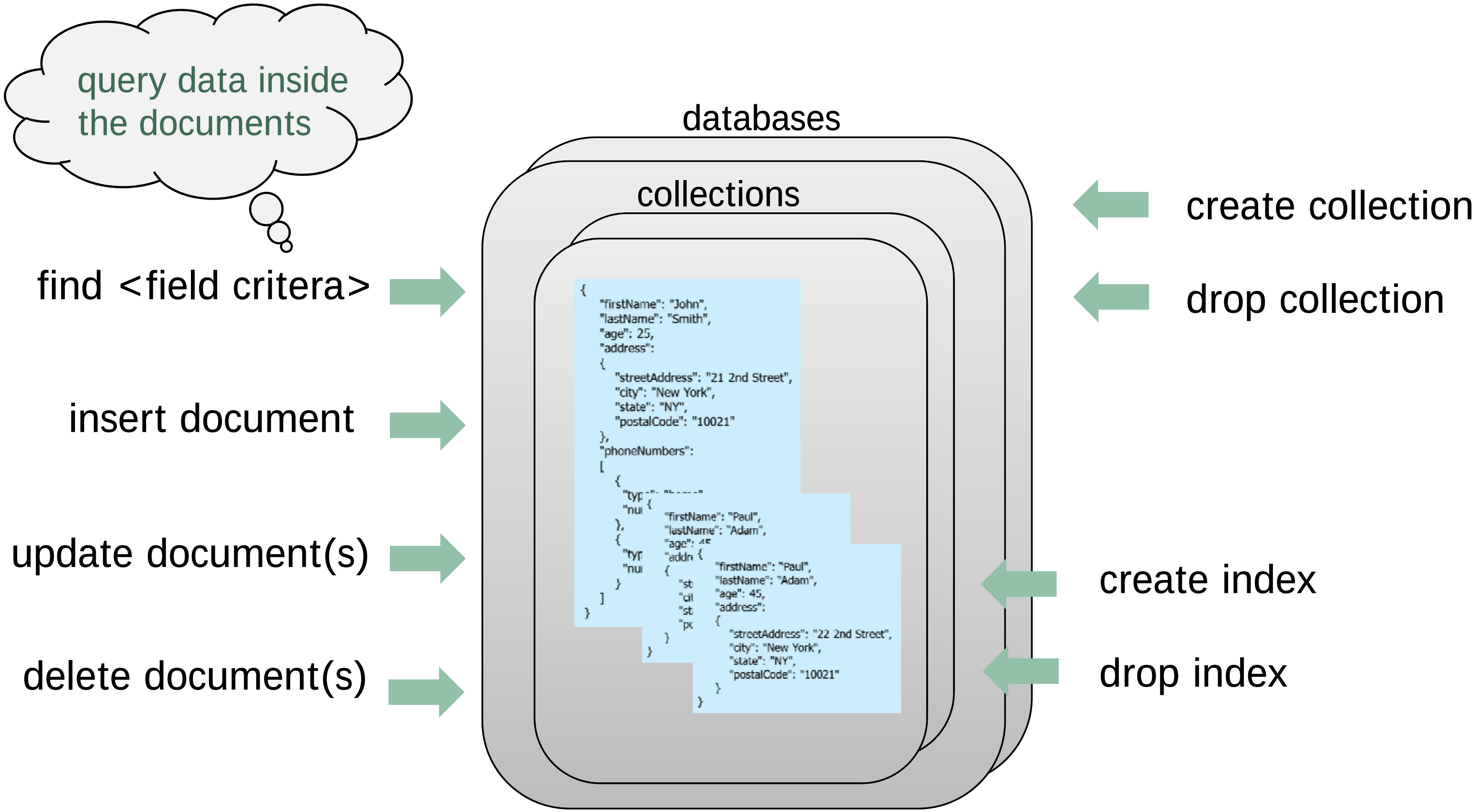
Data Organization

- **Collections** of documents that share indexes (but not a schema)
- Collections are organized into **databases**
- Documents stored in BSON



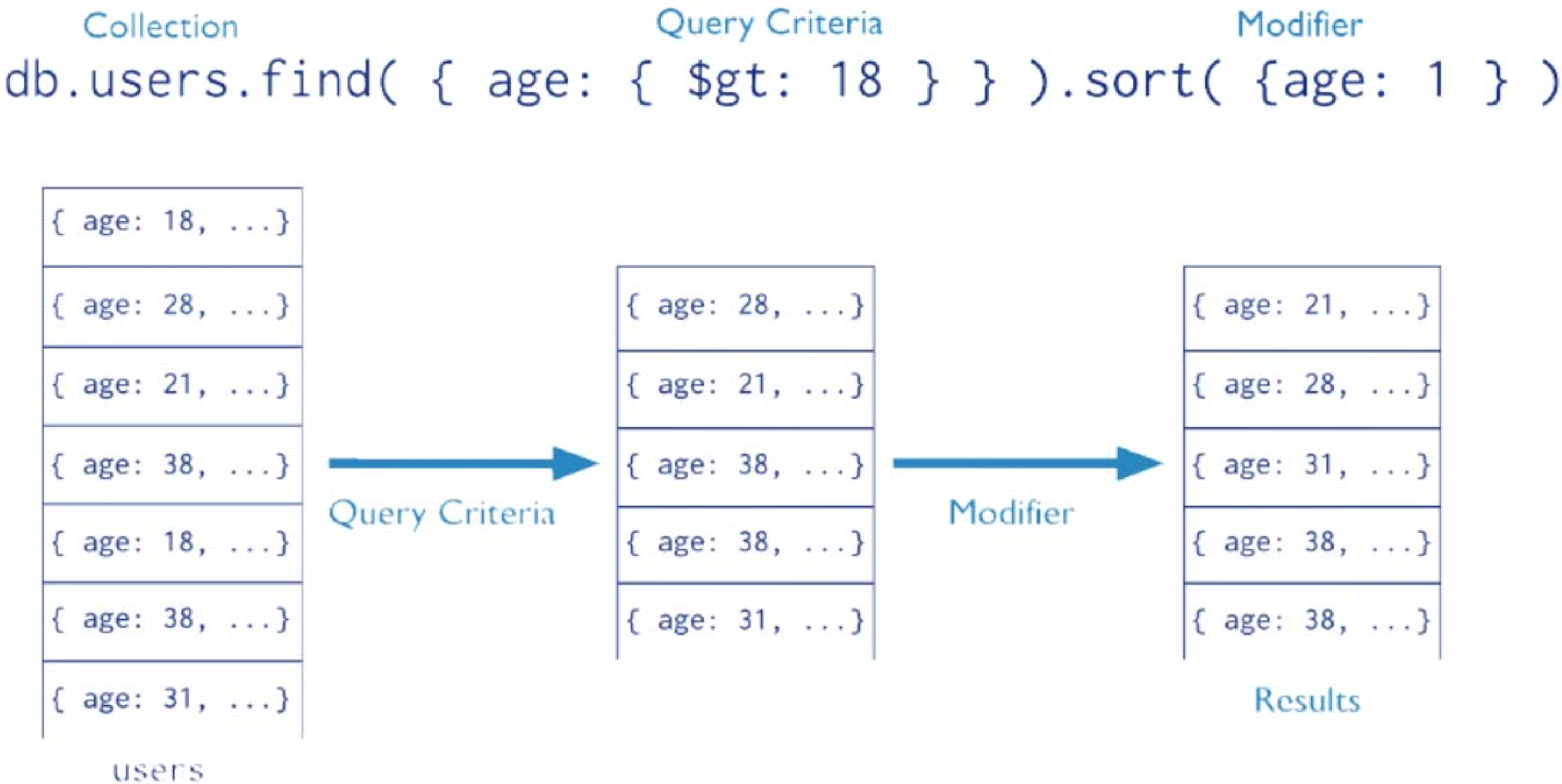
mongoDB

Key Functionality



mongoDB

Querying Documents

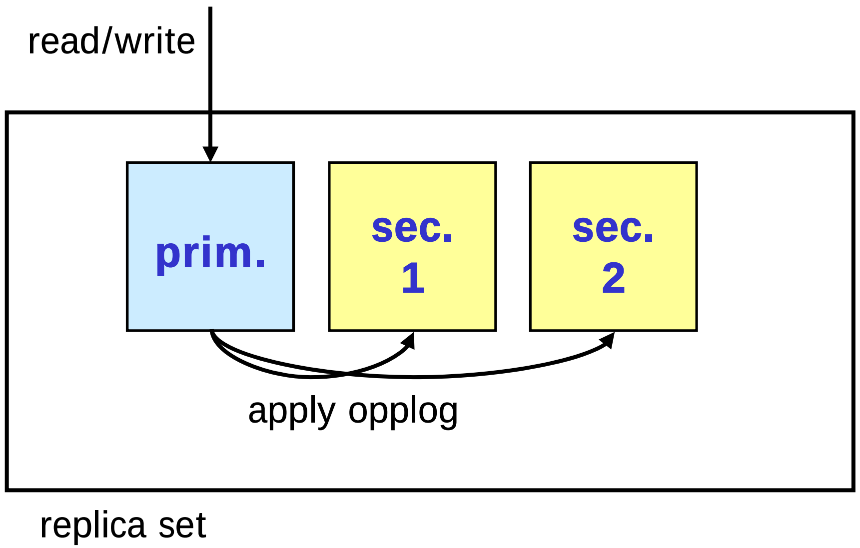


www.mongodb.com

mongoDB

Availability

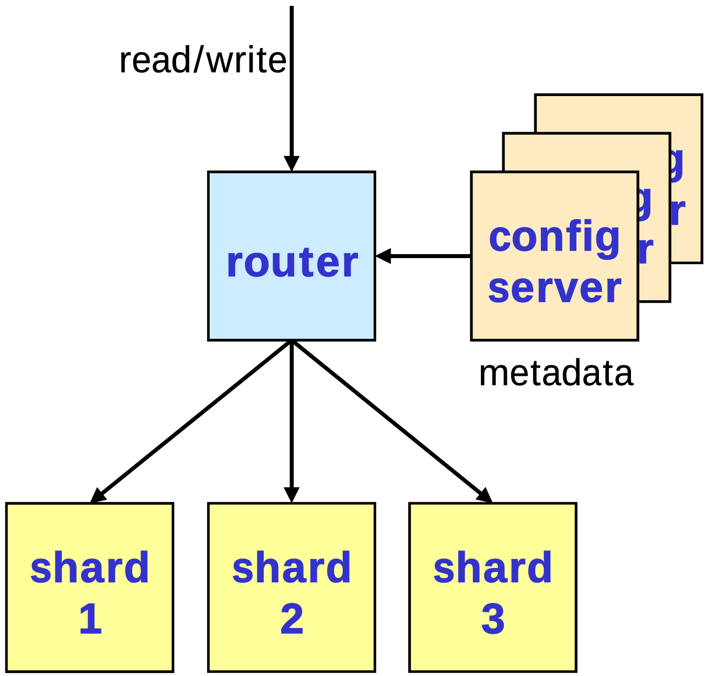
- Each **Replica set** is group of MongoDB instances that hold the same dataset
- one primary instance that takes all write operations
- multiple secondary instances
- changes are replicated to the secondaries
- if the primary is unavailable, the replica set will elect a secondary to be primary
- Specific secondaries
  - priority 0 member
  - hidden member
  - delayed member
  - arbiter



mongoDB

Scalability

- Replica sets for read scalability
  - reading from secondaries may provide stale data
- **Sharding** for write scalability
  - at collection level
  - using indexed field that is available in all documents of the collection
  - range-based or hash-based
  - Each shard is a MongoDB instance
  - background processes to maintain even distribution: splitting + balancer
  - Shards may also hold replica sets



Transactions and Consistency

- **Write concern**, w option
  - default: confirms write operations only on the primary
  - num.: Guarantees that write operations have propagated successfully to the specified number of replica set members including the primary
  - majority: Confirms that write operations have propagated to the majority of voting nodes
- **Read preference**
  - describes how MongoDB clients route read operations to the members of a replica set, i.e., from one of the secondaries or the primary
  - eventual consistency!
- No multi-operation transactions supported.

reading stale data from the primary is possible!

Wide-Column Stores / Column Family Stores

- 2006: originally project of company Powerset
- 2008: HBase becomes Hadoop sub-project.
- 2010: HBase becomes Apache top-level project.
- runs on top of HDFS (Hadoop Distributed File System)
- providing BigTable-like capabilities for Hadoop
- APIs: Java, REST, Thrift, C/C++

Logical Data Model

- Table rows contain:
  - row key
  - versions, typically a timestamp
  - multiple column families per key
    - define column families at design time
    - add columns to a column family at runtime

Row Key	Time Stamp	ColumnFamily contents	ColumnFamily anchor
"com.cnn.www"	t9		anchor:cnnsi.com = "CNN"
"com.cnn.www"	t8		anchor:my.look.ca = "CNN.com"
"com.cnn.www"	t6	contents:html = "<html>..."	
"com.cnn.www"	t5	contents:html = "<html>..."	
"com.cnn.www"	t3	contents:html = "<html>..."	

sparse table

- Metadata
  - there is no catalog that provides the set of all columns for a certain table
  - left to the user/application

Physical Data Model

- Store each column familiy separately
- Sorted by timestamp (descending)
- Empty cells from the logical view are not stored

Row Key	Time Stamp	Column Family anchor
"com.cnn.www"	t9	anchor:cnnsi.com = "CNN"
"com.cnn.www"	t8	anchor:my.look.ca = "CNN.com"

Row Key	Time Stamp	ColumnFamily "contents:"
"com.cnn.www"	t6	contents:html = "<html>..."
"com.cnn.www"	t5	contents:html = "<html>..."
"com.cnn.www"	t3	contents:html = "<html>..."

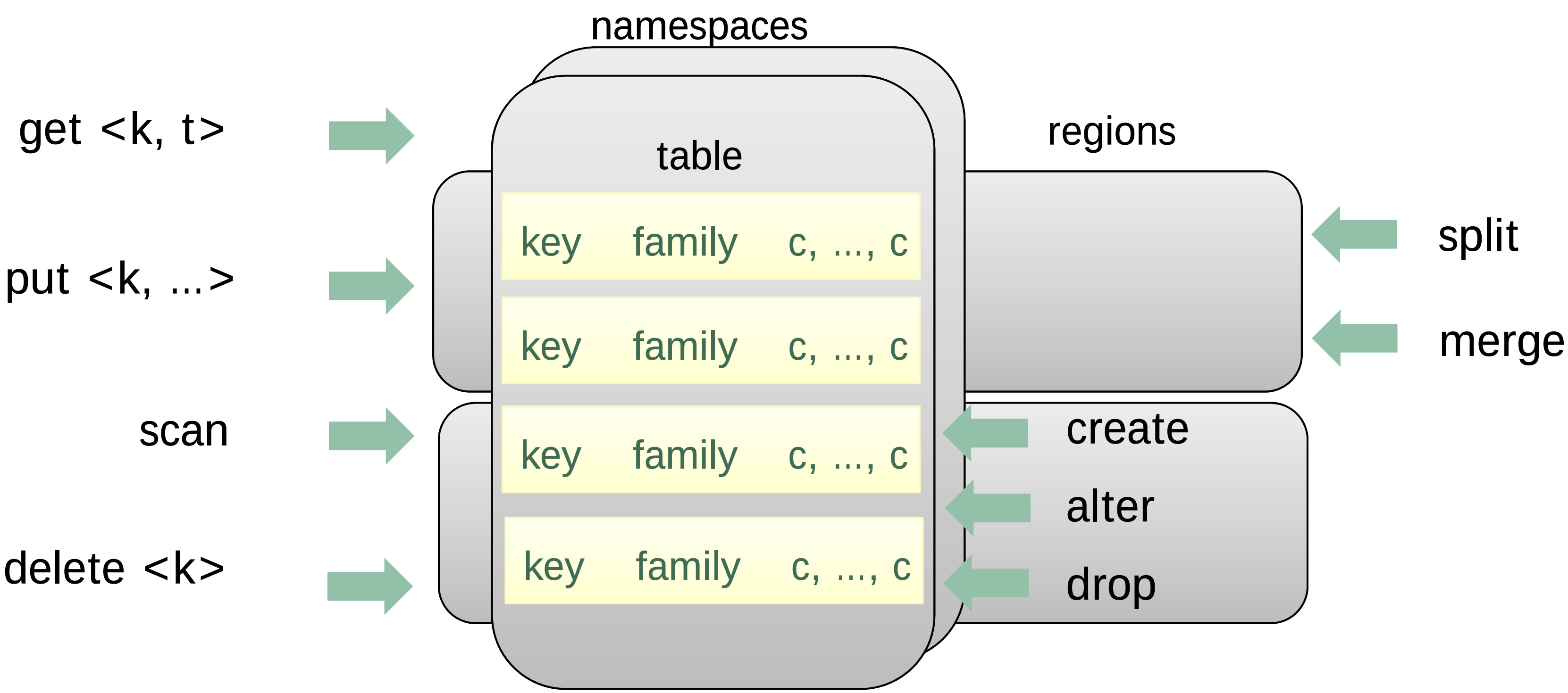
- Key/Value class

keylength	valuelength	key	value

	rowlength	row key	columnfamily length	column family	columnqualifier	timestamp	keytype
e.g.		com.cnn.www	2	anchor	cnnsi.com	t9	put

APACHE HBASE

Key Functionality



APACHE HBASE

MasterServer and RegionServer

HMaster
- monitors RegionServers
- operations related to metadata (tables, column families, regions)

META: list of regions for each table

Backup HMaster

...

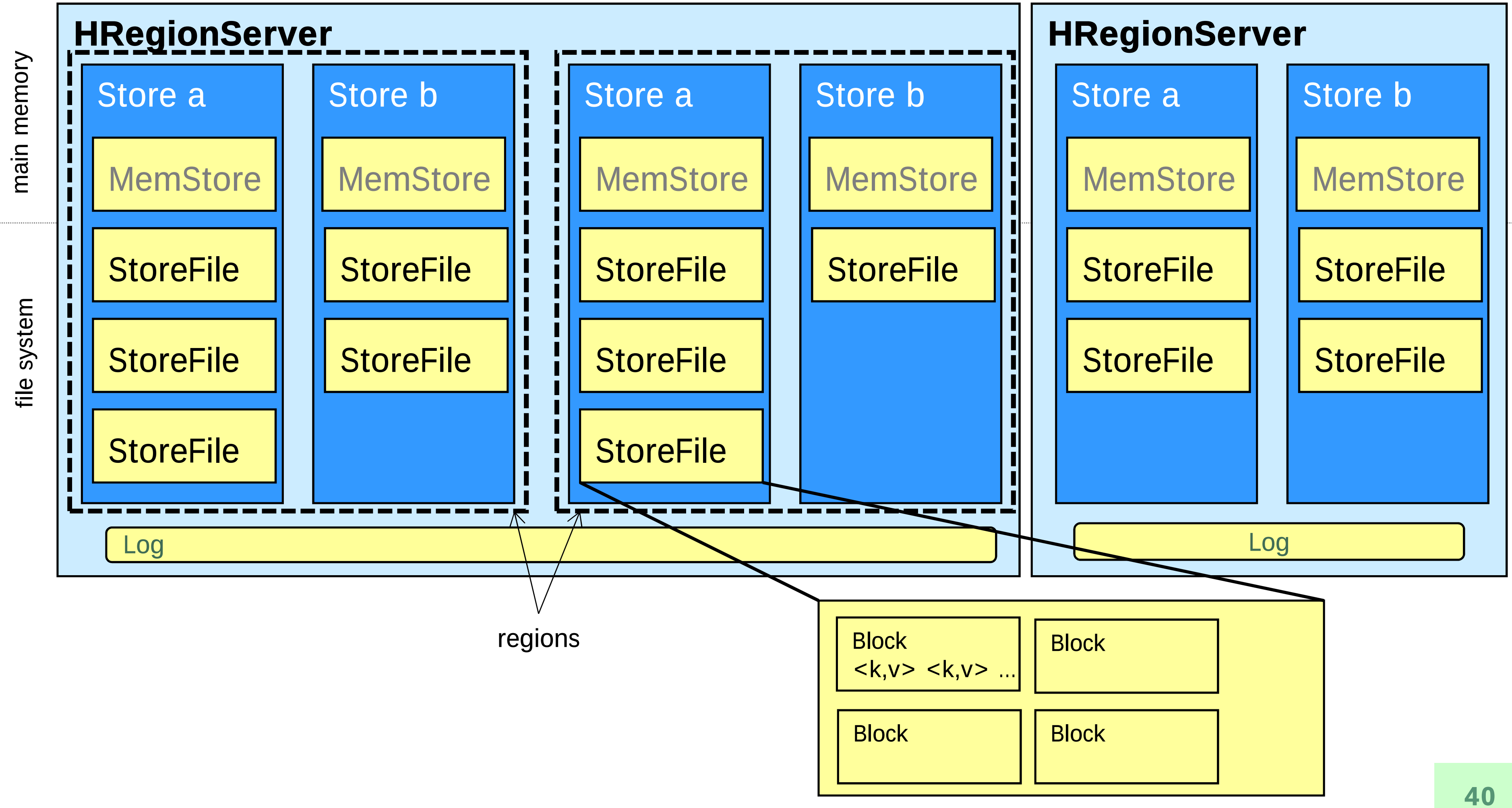
HRegionServer
- manages regions
- operations related to data (put, get, ...)
- operations related to regions (split, merge, ...)

- Failover
  - HBase clients talk directly to the RegionServers, hence they may continue without MasterServer (at least for some time)
  - catalog table META exists as HBase tables, i.e., not resident in the MasterServer

- Failover
  - Region immediately becomes unavailable when the RegionServer is down
  - The Master will detect that the RegionServer has failed
  - region assignments will be considered invalid
  - assign region to a new RegionServer

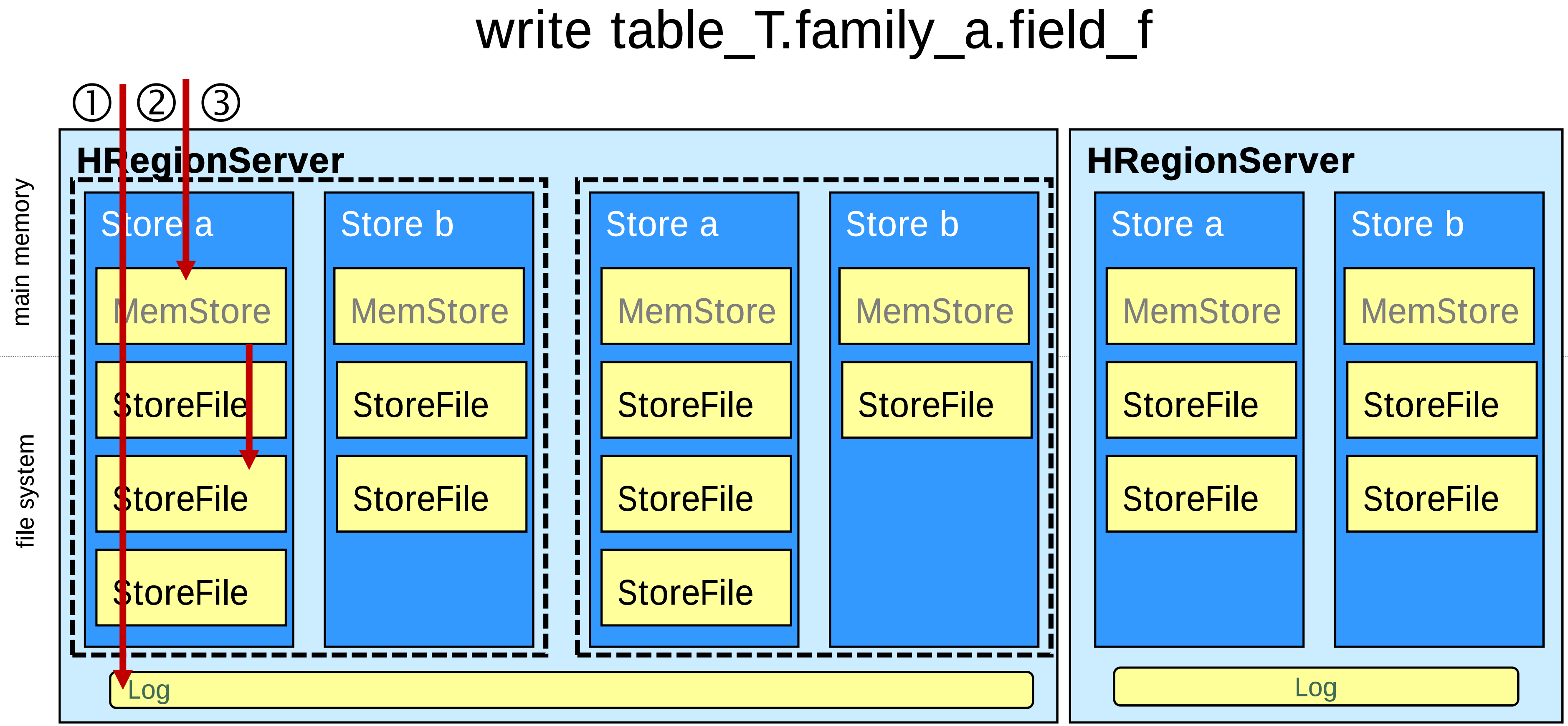
Storage Structure

- Table T with column families a and b



APACHE HBASE

Write Data

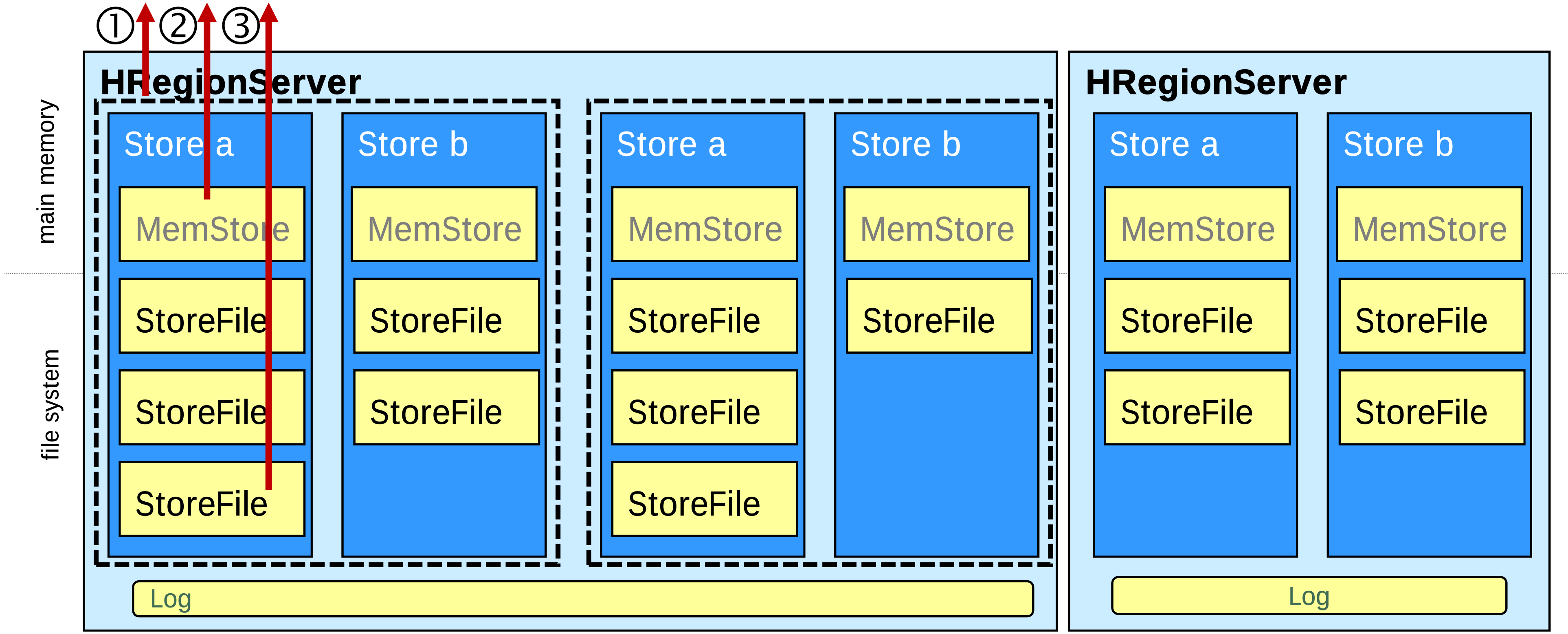


1. Write change to log (WAL)
2. Write change to MemStore
3. Regularily flush MemStore to disk (into StoreFiles) and empty MemStore

APACHE HBASE

Read Data

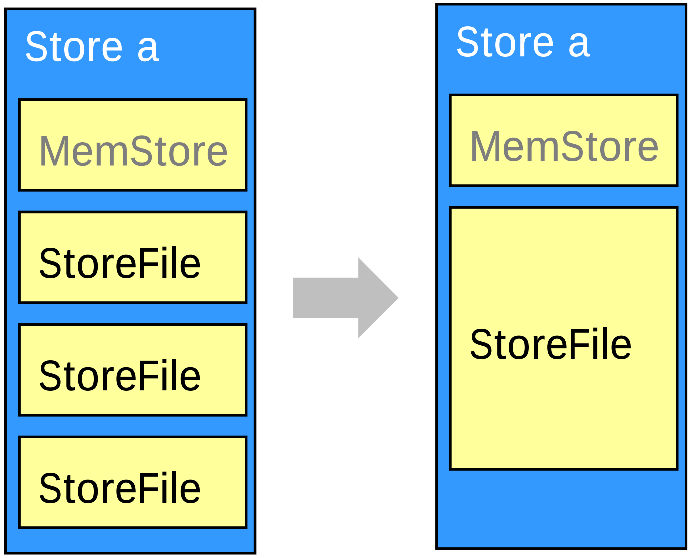
read table_T.family_a.field_f



1. Read from Block Cache
2. Read from MemStore
3. Read from all relevant StoreFiles
4. Merge results

APACHE HBASE

StoreFile Reorganisation



- minor compaction
  - merge various StoreFiles, without considering tombstones etc.
- major compaction
  - reorg of store files, e.g. by removing deleted rows
    → significantly reduces file size
  - at a configurable time interval or manually
- Merge Regions
  - consolidate several regions of one table into a single region
  - offline reorg initiated manually!
- Split Regions
  - distribute data from one region to several regions
  - configurable by parameter *max.filesize* or manually

APACHE HBASE Transactions and Consistency

- No explicit transaction boundaries
- Atomicity
  - atomic row-level operations (either "success" or "failure")
  - operations spanning several rows (batch put) are not atomic
- Consistency
  - Default: Strong consistency by routing all through a single region server
  - Optional: Region replication for high availability
    - Writes only through the primary
    - Reads may also be processed by the secondaries

CQL in Cassandra

```
CREATE KEYSPACE demodb WITH REPLICATION =
{'class' : 'SimpleStrategy', 'replication_factor': 3};
```

```
SELECT *
FROM emp
WHERE empID IN (130,104)
ORDER BY deptID DESC;
```

```
CREATE TABLE users (
user_name varchar,
password varchar,
gender varchar,
session_token varchar,
state varchar,
birth_year bigint,
PRIMARY KEY (user_name));
```

```
SELECT WRITETIME (name)
FROM excelsior.clicks
WHERE url = 'http://apache.org';
```

```
INSERT INTO excelsior.clicks (userid, url, date, name)
VALUES ( 3715e600-2eb0-11e2-81c1-0800200c9a66,
'http://apache.org', '2013-10-09', 'Mary')
USING TTL 86400;
```

cassandra.apache.org

Graph Databases

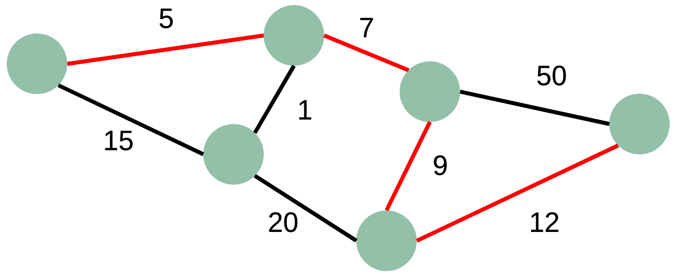
- Developer: Neo Technology (http://www.neotechnology.com)
- Available since: 2007
- Licence: GPLv3 and AGPLv3, commercial
- Supported operating systems: all major platforms
- Drivers for:
  - Java, .NET, JavaScript, Python, Ruby, PHP
  - and R, Go, Clojure, Perl, Haskell

Typical Use Cases

- Highly connected data,
  e.g., social networks,
  employees and their knowledge

smartblogs.com

- Location-based services,
  e.g., planning delivery



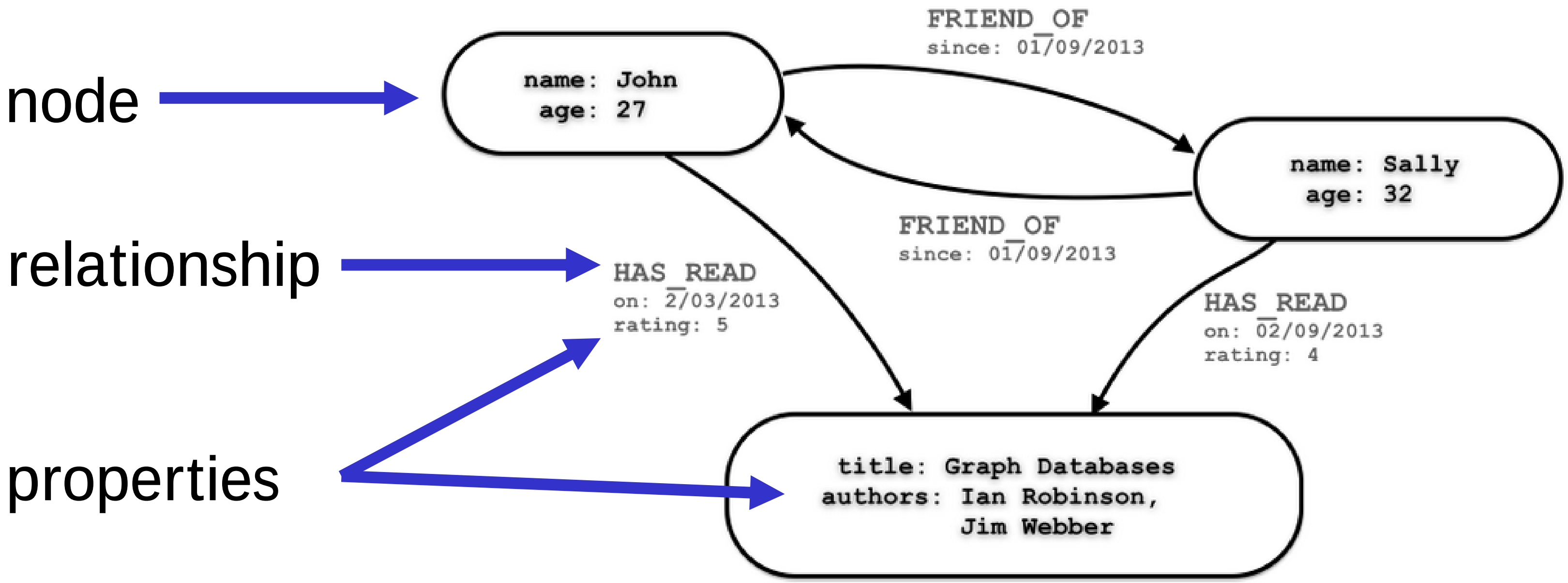
graph with distances

- Recommendation systems,
  e.g., bought products, often-visited attractions

people who visited ...

also visited ...

Graph Data Model



- No need to define a schema

neo4j.com

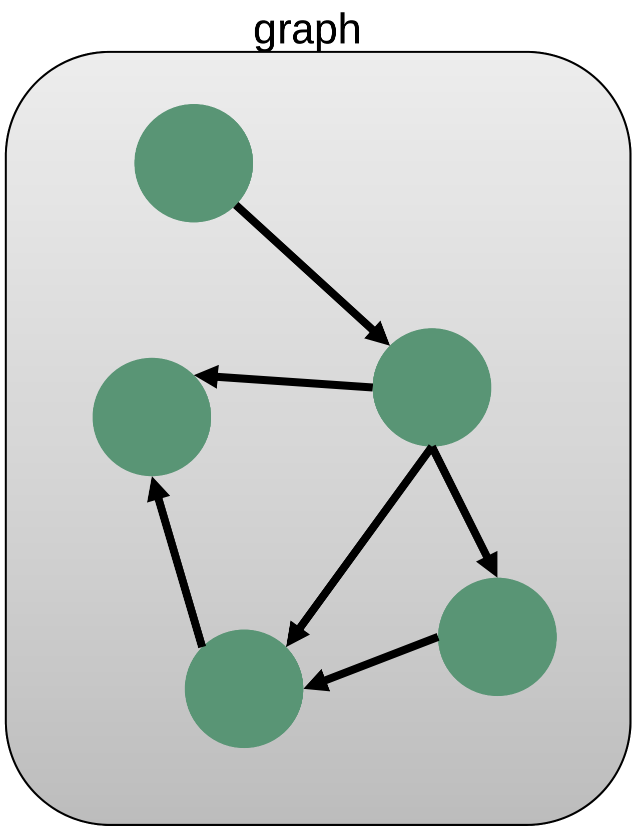
neo4j

Basic Functionality

match node patterns

graph traversal

query index



create node/relationship

delete node/relationship

set property

remove property

create index

Cypher Example

- Many query languages: Cypher, Gremlin, G, GraphLog, GRAM, GraphDB, ...
- No standard

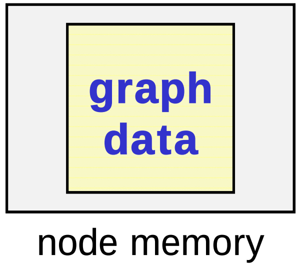
```
CREATE (me:PERSON {name:”Holger”})
CREATE (mat:PERSON {name:”Matthias”})
CREATE (fra:PERSON {name:”Frank”})
CREATE (me) -[knows:KNOWS]-> (mat)
CREATE (me) -[knows:KNOWS]-> (fra)
CREATE (mat) -[knows:KNOWS]-> (me)

MATCH (n {name:”Holger”})-[:KNOWS]->(m)
```

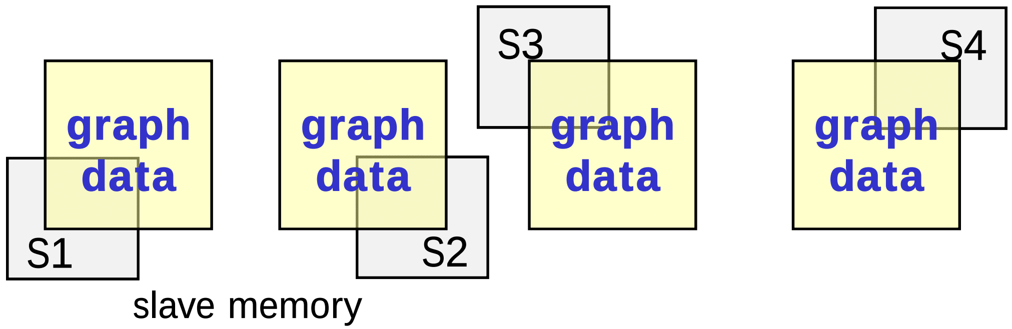
Scalability

neo4j

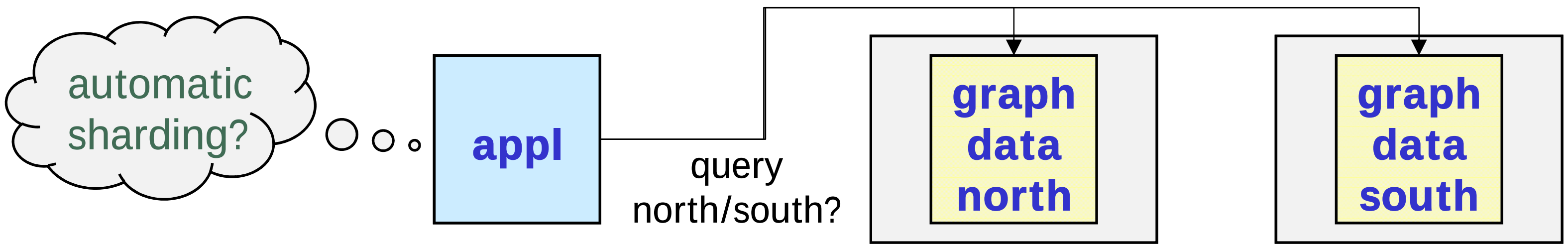
- Strategies for read scaling
  a) large enough memory
     for the working set of nodes



  b) adding read-only slaves



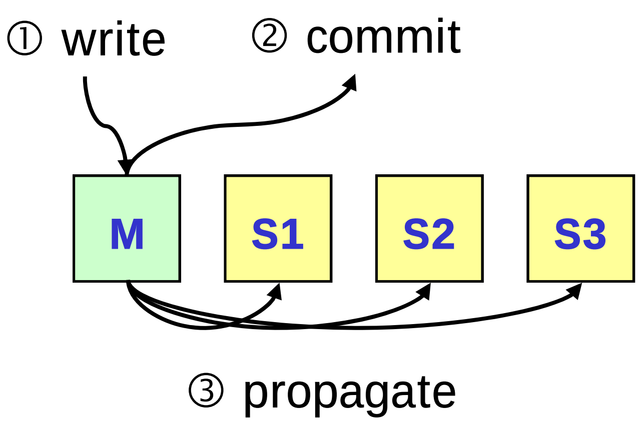
  c) application-level sharding



High Availability

- HA availability feature in neo4j
  - cluster of 1 master and n slave nodes
  - continues to operate from any number of nodes down to a single machine
  - nodes may leave and re-join the cluster
  - in case of master failure, a new master will be elected
  - read operations are possible on any node
  - write operations are possible on any node and propagated to the others

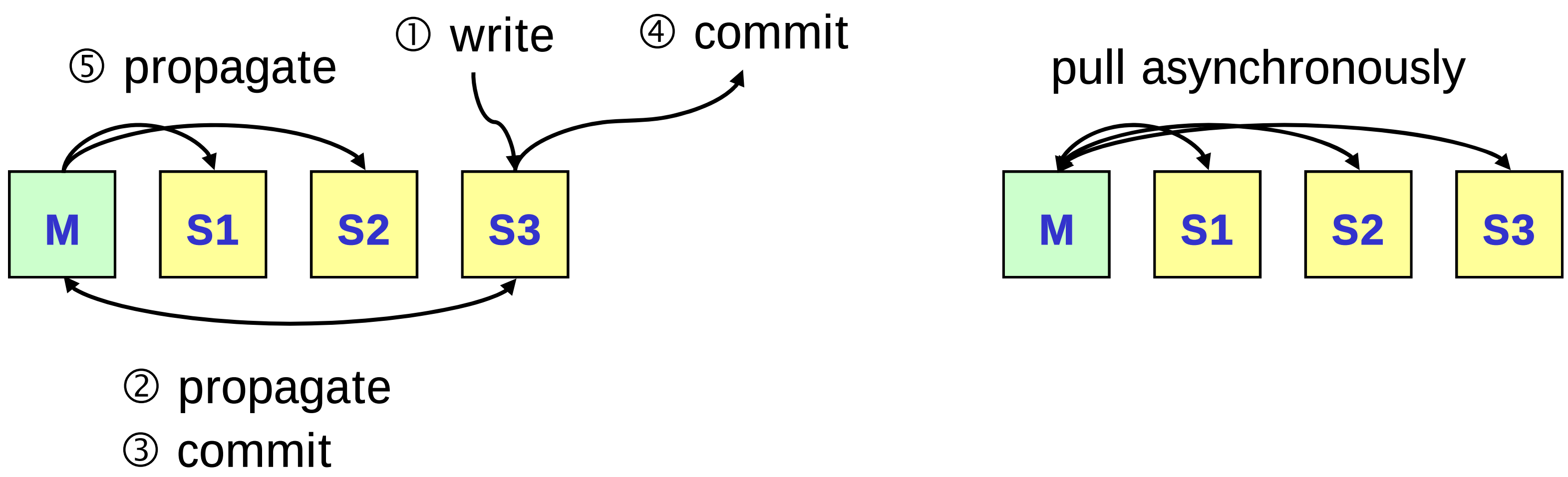
write on master



High Availability

- HA availability feature in neo4j
  - cluster of 1 master and n slave nodes
  - continues to operate from any number of nodes down to a single machine
  - nodes may leave and re-join the cluster
  - in case of master failure, a new master will be elected
  - read operations are possible on any node
  - write operations are possible on any node and propagated to the others

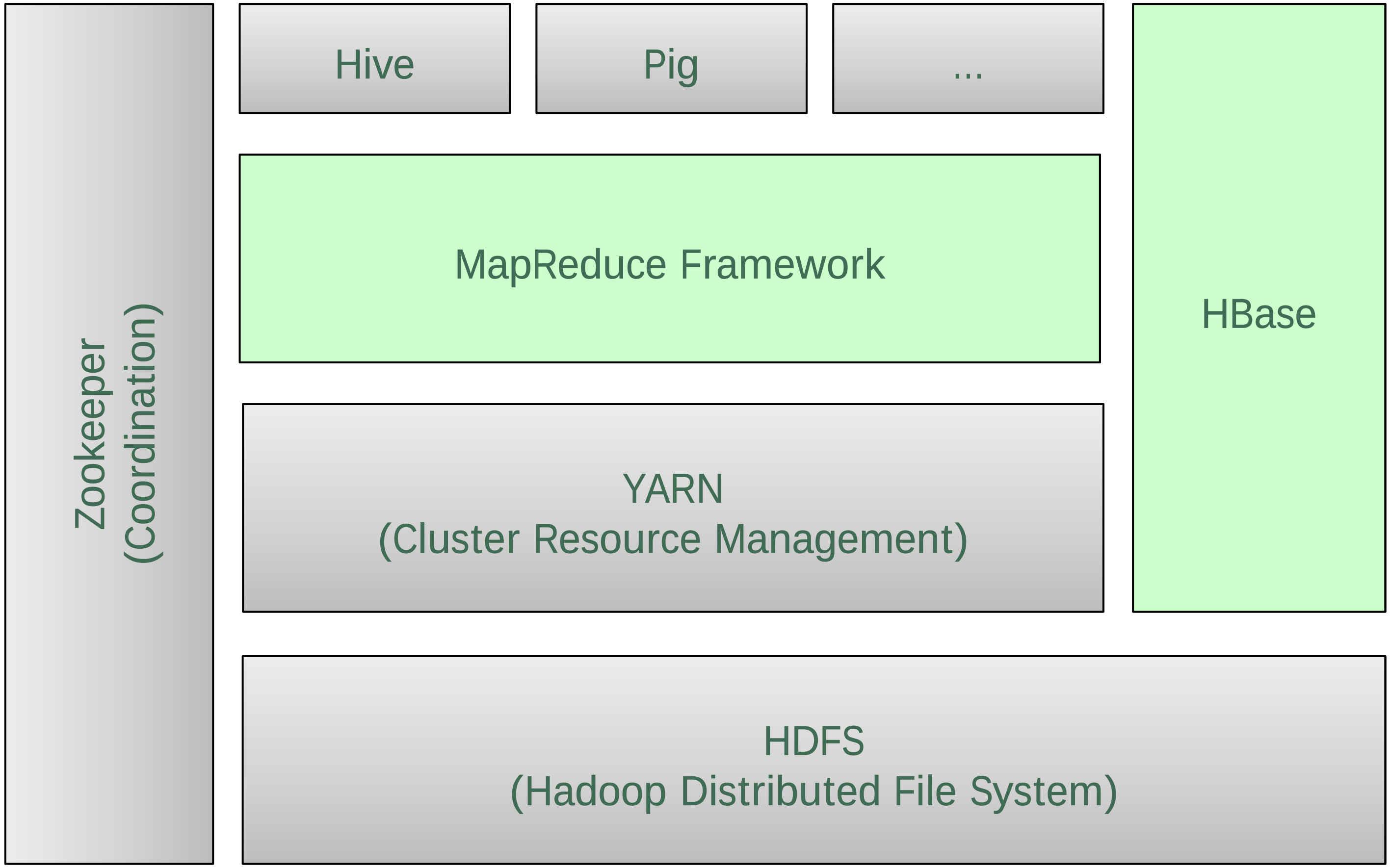
write on slave



Transactions and Consistency

- Set transaction boundaries explicitly
- Transactions are possible on any node in the cluster
- Transactions are atomic and durable
- Writes are eventually consistent
  - optimistically pushed to slaves
  - slaves can also be configured to pull updates asynchronously

Hadoop Ecosystem



- Apache project http://hadoop.apache.org/

Principles of Map Reduce

- User provides data in files
- Data model: key/value pairs (k, v)
- Based on higher-order functions MAP and REDUCE
- Tasks of the programmer
  - User-defined functions *m* and *r* serve as input to MAP and REDUCE
  - *m* and *r* define what the job actually does
- MAP m: $(K_m, V_m) \mapsto (K_r, V_r)^*$
- REDUCE r: $(K_r, {V_r}^*) \mapsto (K_r, V_r)$
- Example: Aggregate salary per department:

(employee, <name, department, salary, ...>)

MAP

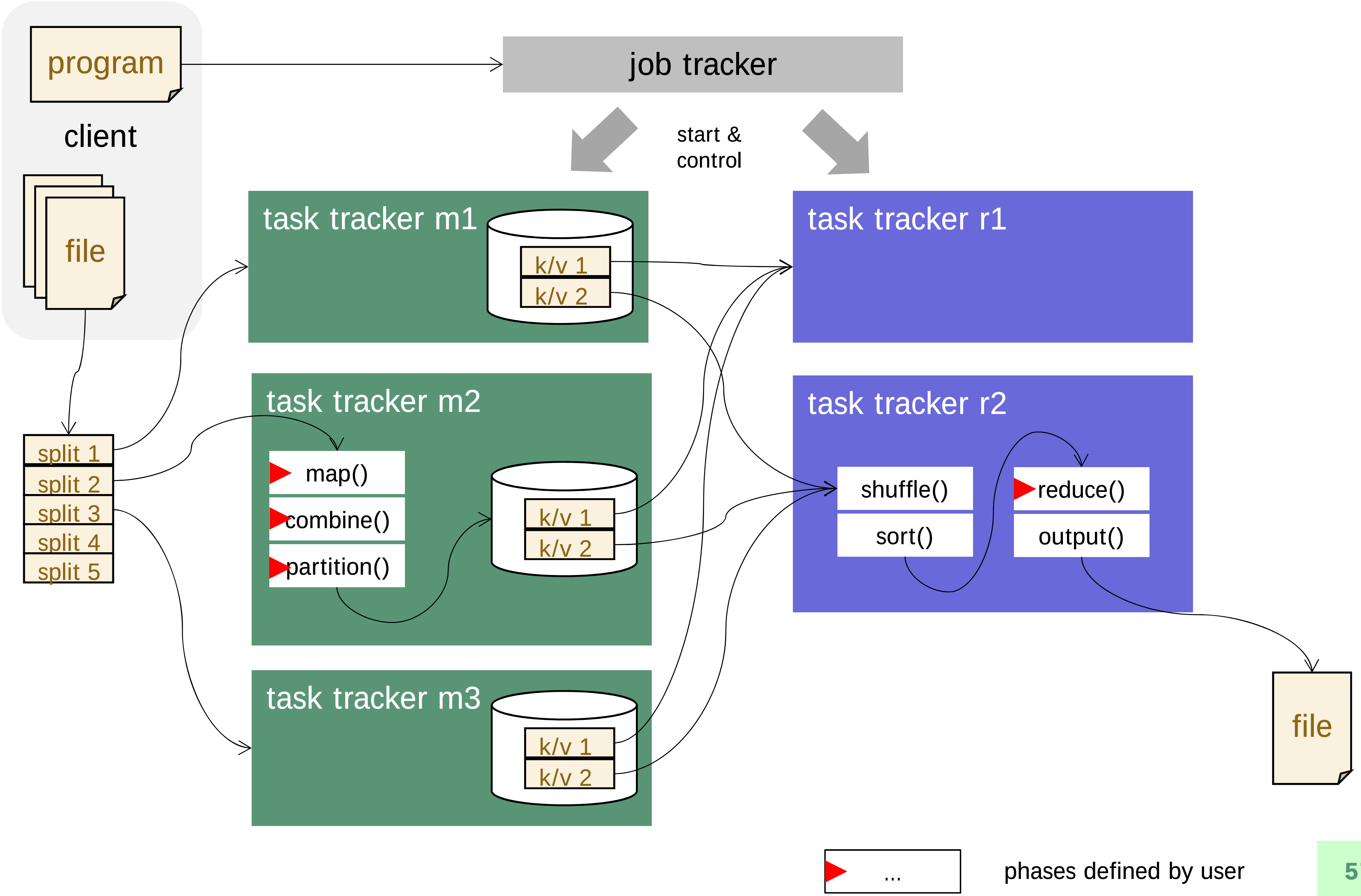
(department, salary)

(department, <salary, salary, ...>)

RED

(department, salary)

Execution of Map/Reduce Jobs

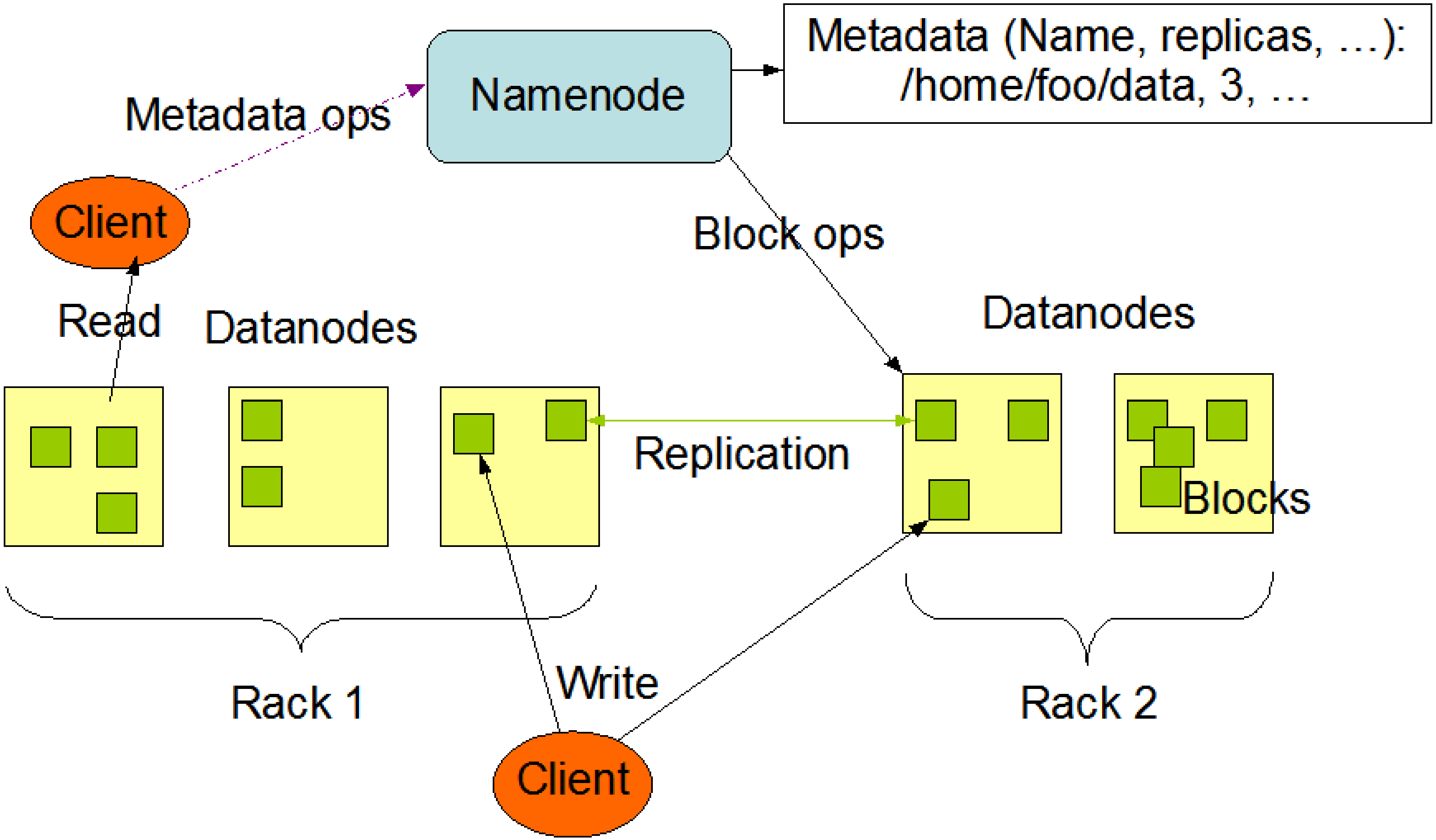


Fault Tolerance

- map node fails
  - job tracker receives no report for a certain time -> mark node as failed
  - restart map job on a different node
  - new job reads another copy of the necessary input split
- reduce node fails
  - job tracker receives no report for a certain time -> mark node as failed
  - restart reduce job on a different node
  - read necessary intermediate input data from map nodes
- To make this work, all relevant data has to be stored in a distributed file system, in particular
  - the input splits
  - intermediate data produced by map jobs

Hadoop Files System (HDFS)

HDFS Architecture



HDFS Users Guide at http://hadoop.apache.org

Is that all?

- Other systems build on or extend this basic functionality
- Build an SQL layer on to of Hadoop MapReduce
  - Hive
  - Pig
- Others focus on datastream and in-memory processing
  - Spark
  - Flink

What we also Skipped Today

- Further classes of NoSQL systems
  - Triple stores, ...
- NewSQL
- Cloud offerings for the various types of NoSQL data stores
  - e.g., Riak CS (Cloud Storage)
- More cloud platforms
  - IBM Bluemix
  - Google app engine

Conclusion

- **Relational Databases provide**
  - Data spread over many tables
  - Schema needs to be defined
  - Structured query language (SQL)
  - Transactions
  - Strong Consistency
  - General purpose applicability
- **NoSQL**
  - Aggregated data in one object (identified by a key)
  - No predefined schema
  - No declarative query language
  - Limited transactional capability
  - Eventual consistency rather ACID property
  - Focus on scalability and availability
  - Often selected and customized for a concrete application scenario

To make a proper decision, carefully examine your application

- the data model that is most appropriate
- the query complexity
- the consistency needs
- the transactional requirements