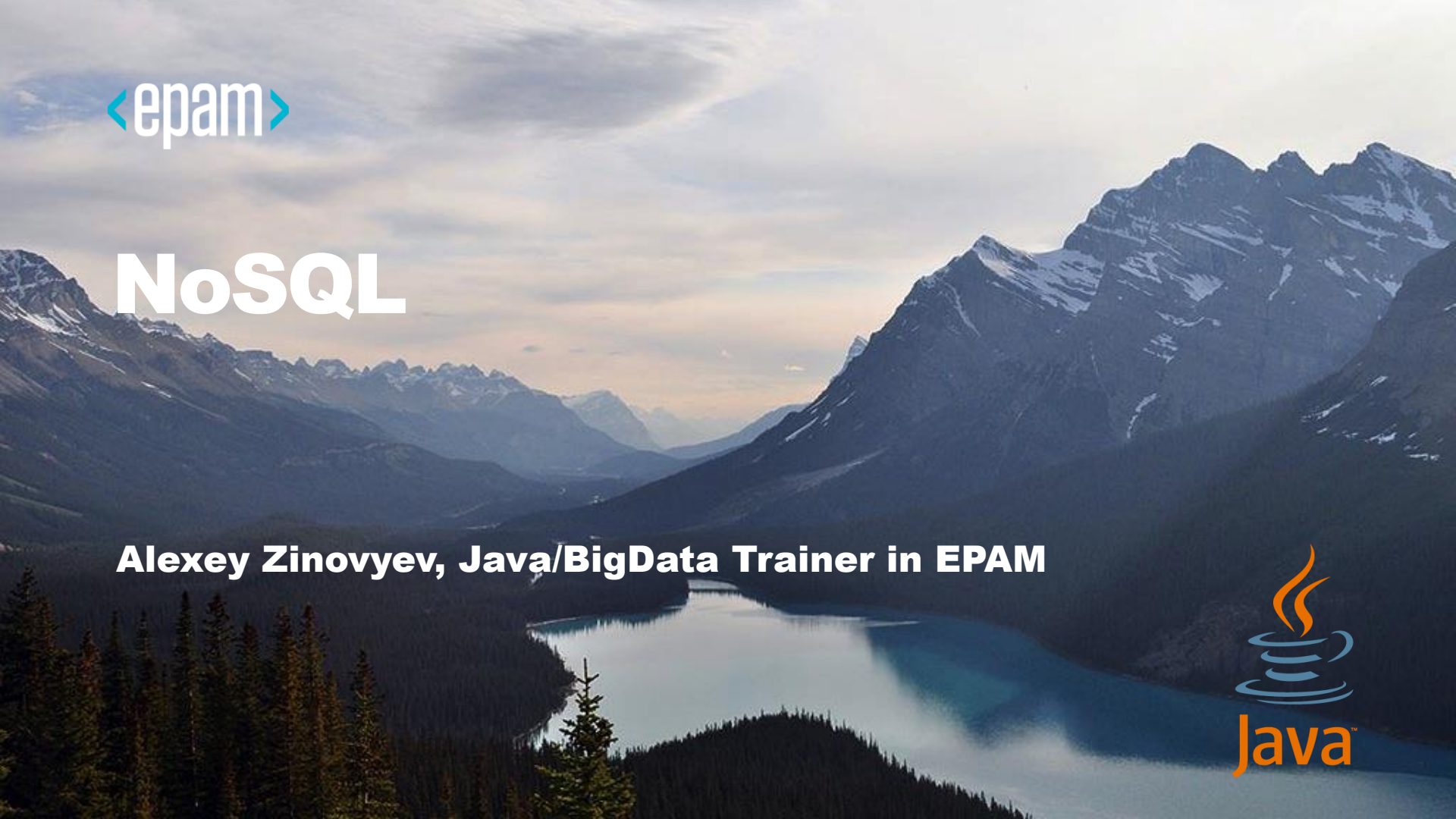

epam
NoSQL
Alexey Zinovyev, Java/BigData Trainer in EPAM
Java

With IT since 2007

With Java since 2009

With Hadoop since 2012

With EPAM since 2015

About

Contacts

E-mail : Alexey_Zinovyev@epam.com

Twitter : @zaleslaw @BigDataRussia

Facebook: https://www.facebook.com/zaleslaw

vk.com/big_data_russia **Big Data Russia**

vk.com/java_jvm **Java & JVM langs**

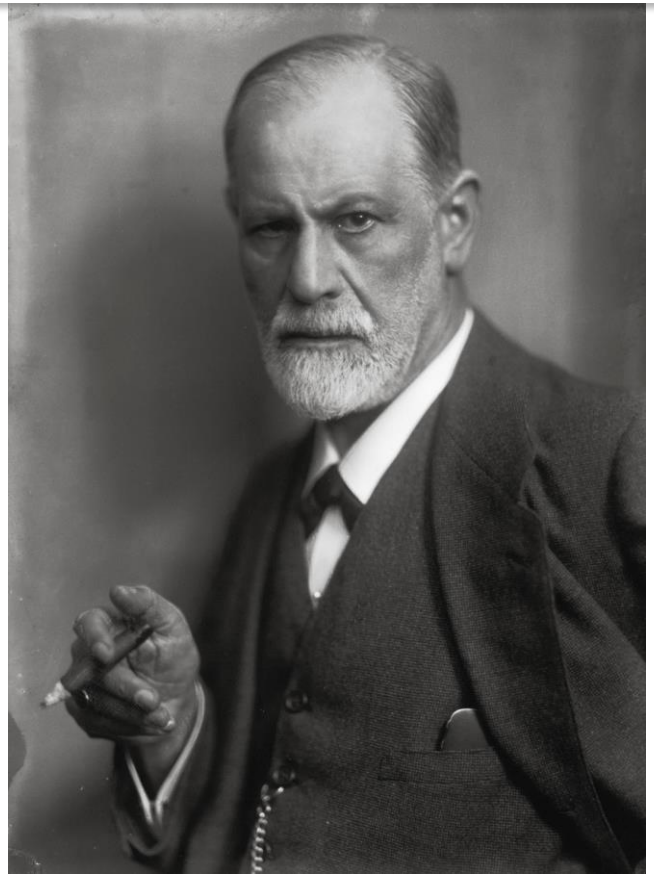
The Good Old Days

One of these fine days...

We have a NoSQL job for you, son!

But you like SQL and HATE nontraditional data

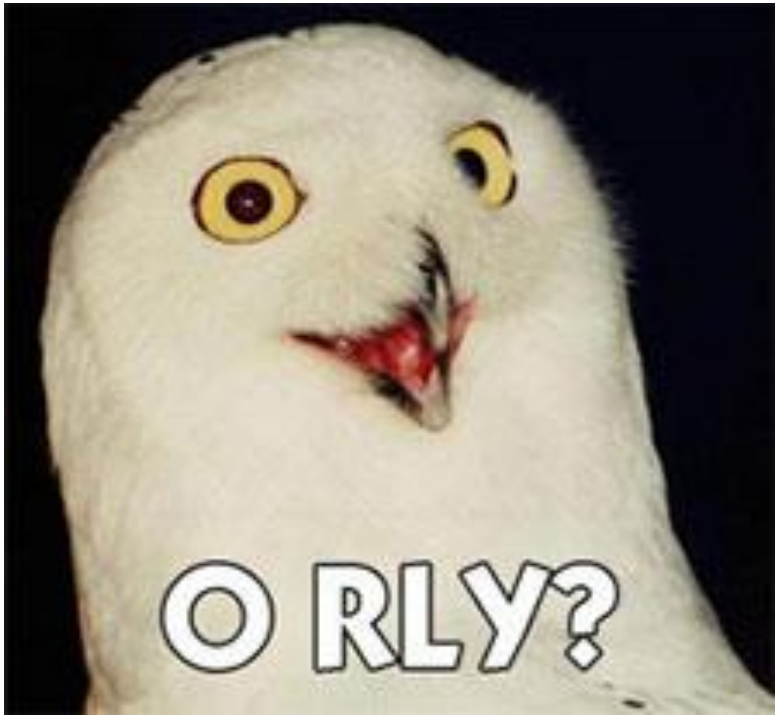
Let’s talk about it, boy...



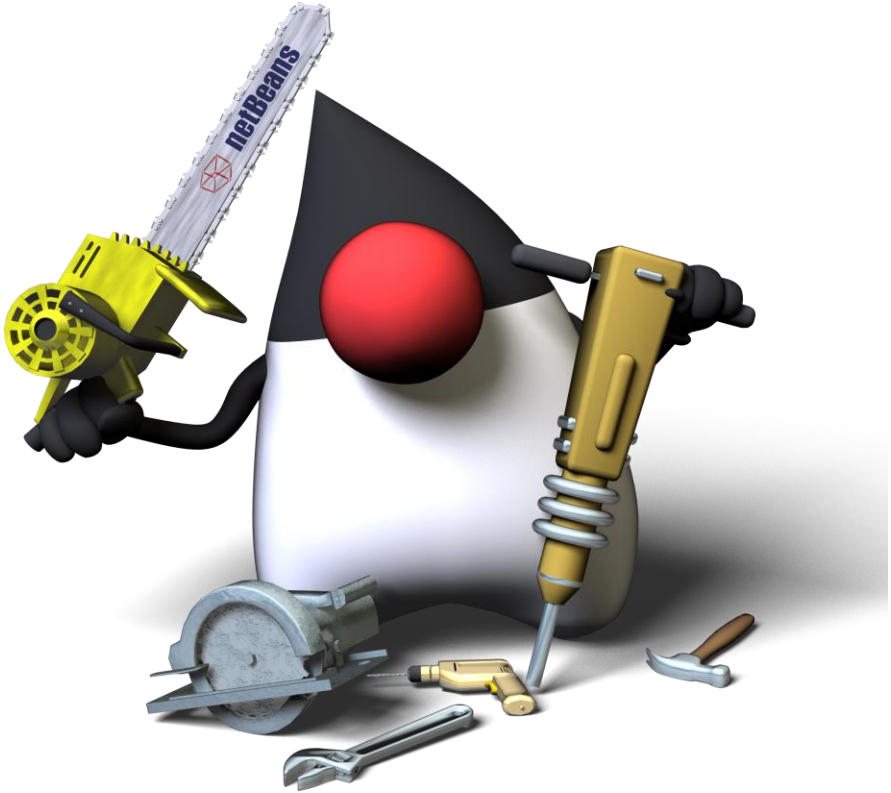
The Database Market

Let’s use Cassandra to keep logs ONLY ..

For logs ONLY?



Case #0 : PosgreSQL & Cassandra

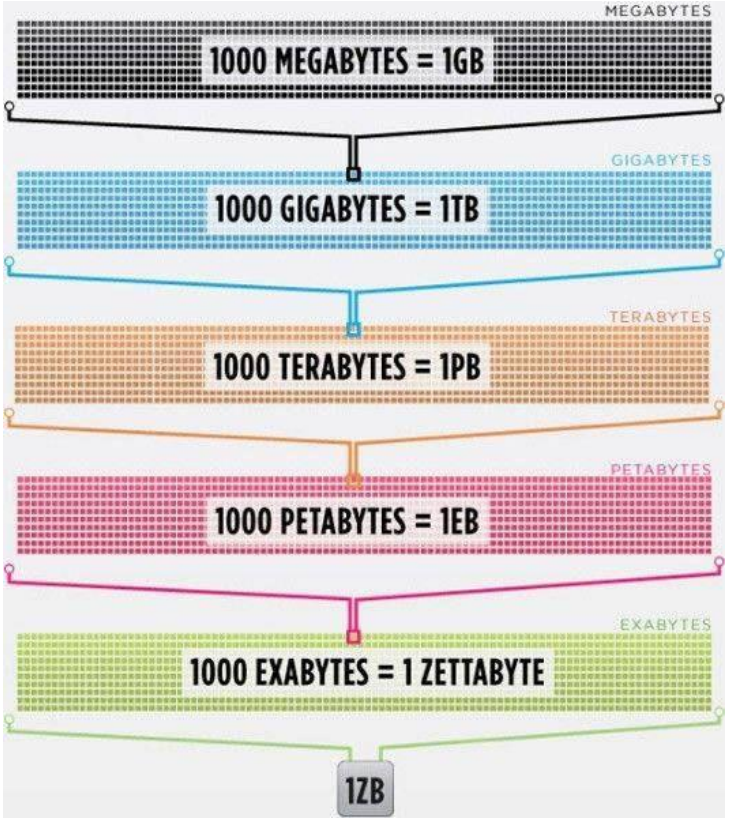



Backend Development in 1997


Backend Development in 2017

10^6 rows in MySQL

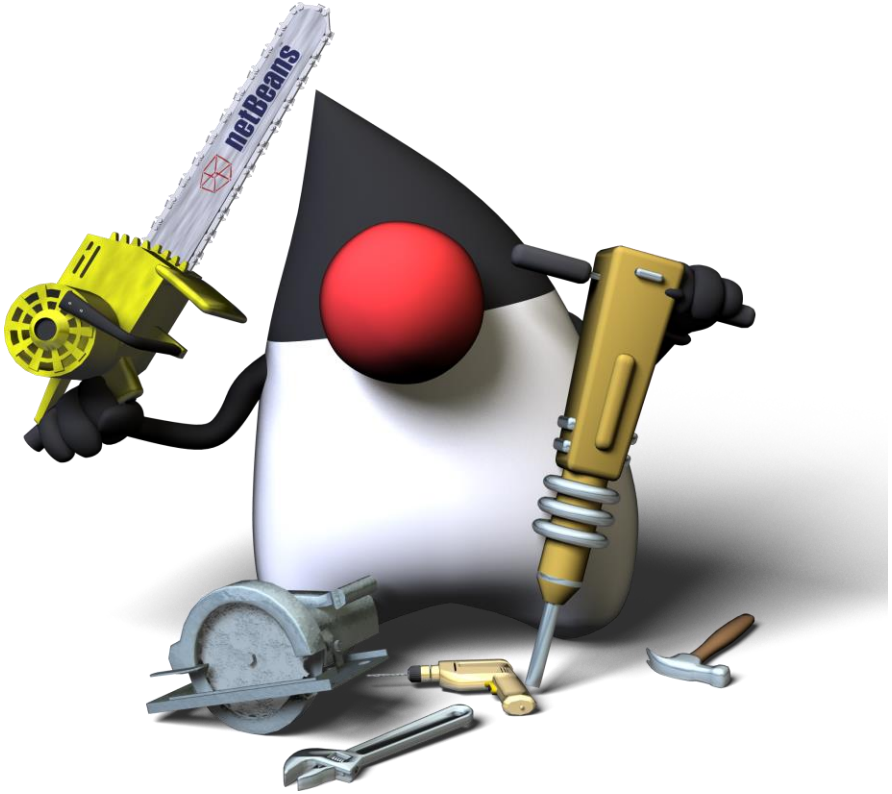
GB->TB->PB->?


MEGABYTES
1000 MEGABYTES = 1GB
GIGABYTES
1000 GIGABYTES = 1TB
TERABYTES
1000 TERABYTES = 1PB
PETABYTES
1000 PETABYTES = 1EB
EXABYTES
1000 EXABYTES = 1 ZETTABYTE
1ZB

This chapter covers …

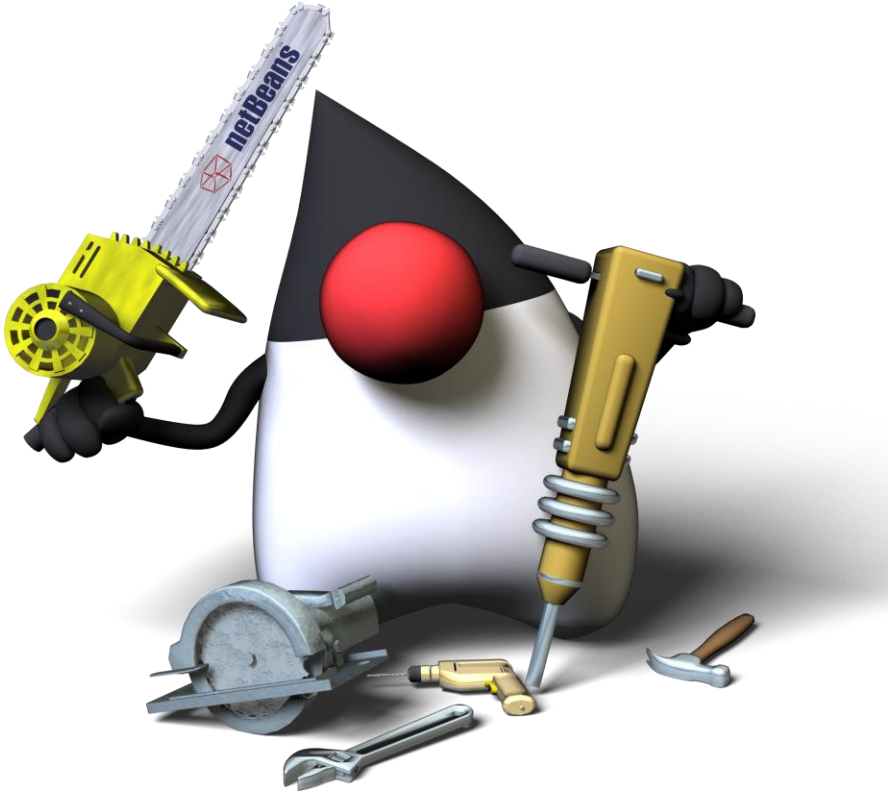
- Highload applications
- Sharding & replication
- NoSQL databases

Case #1 : Likes in Classmates

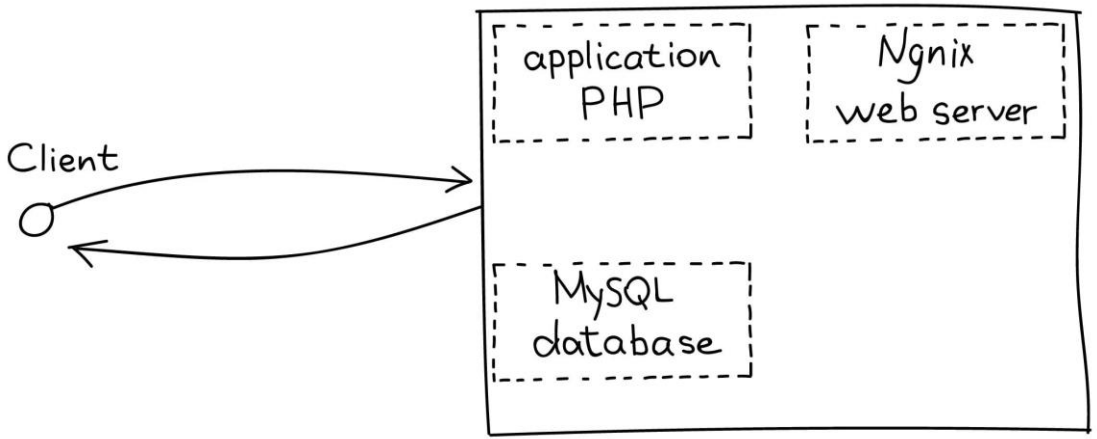


HIGHLOAD APPLICATIONS

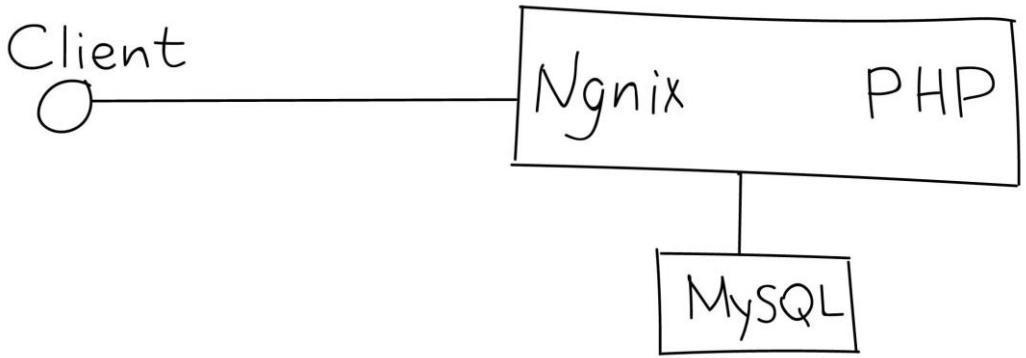
Case #2 : Pet E-Commerce



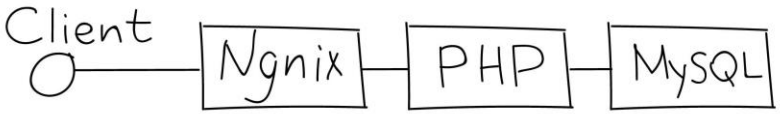
Simple web application #1



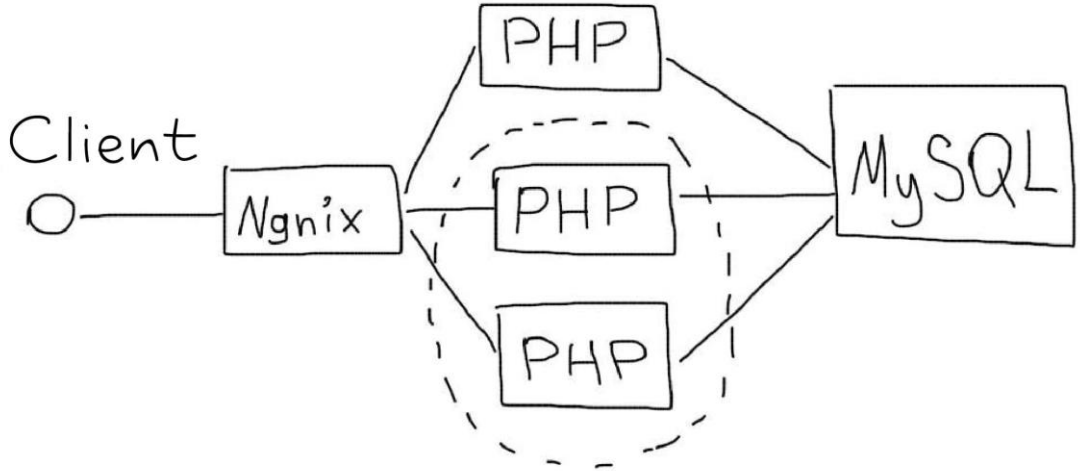
Simple web application #2



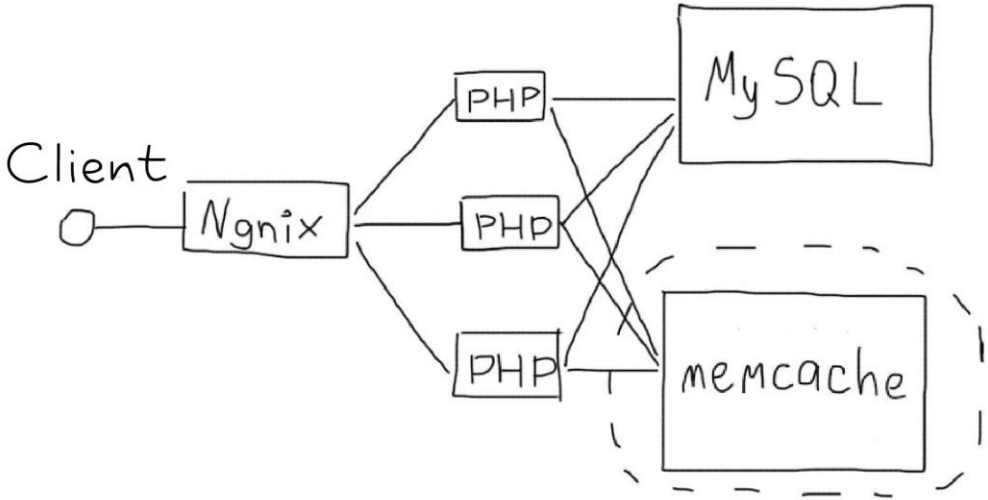
Simple web application #3



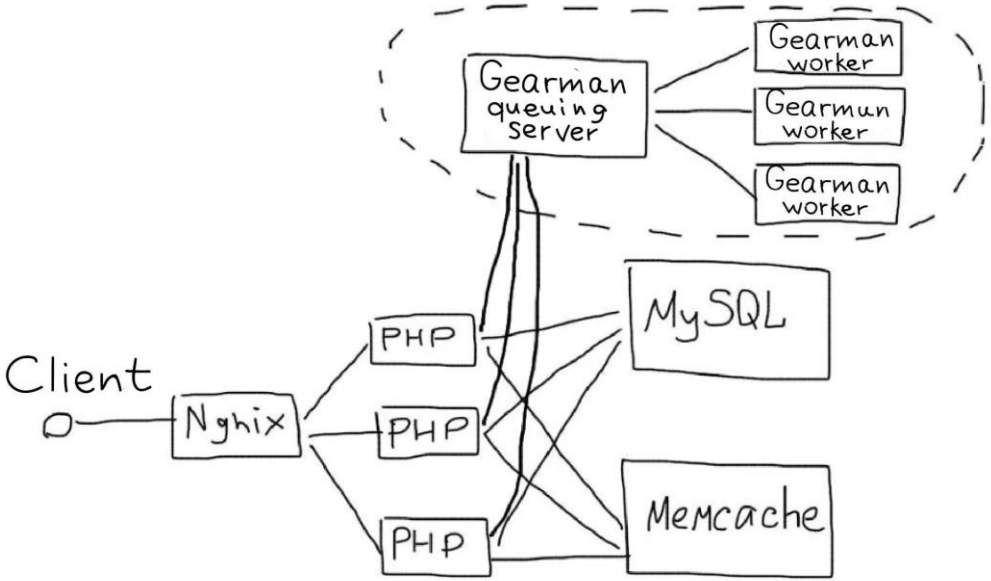
Simple web application #4



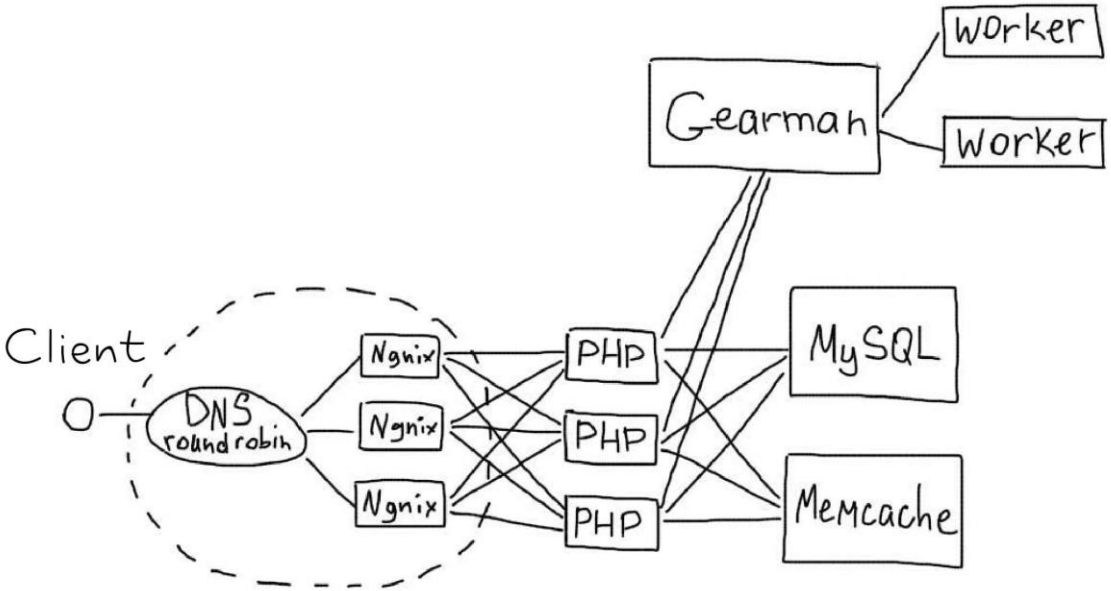
Simple web application #5



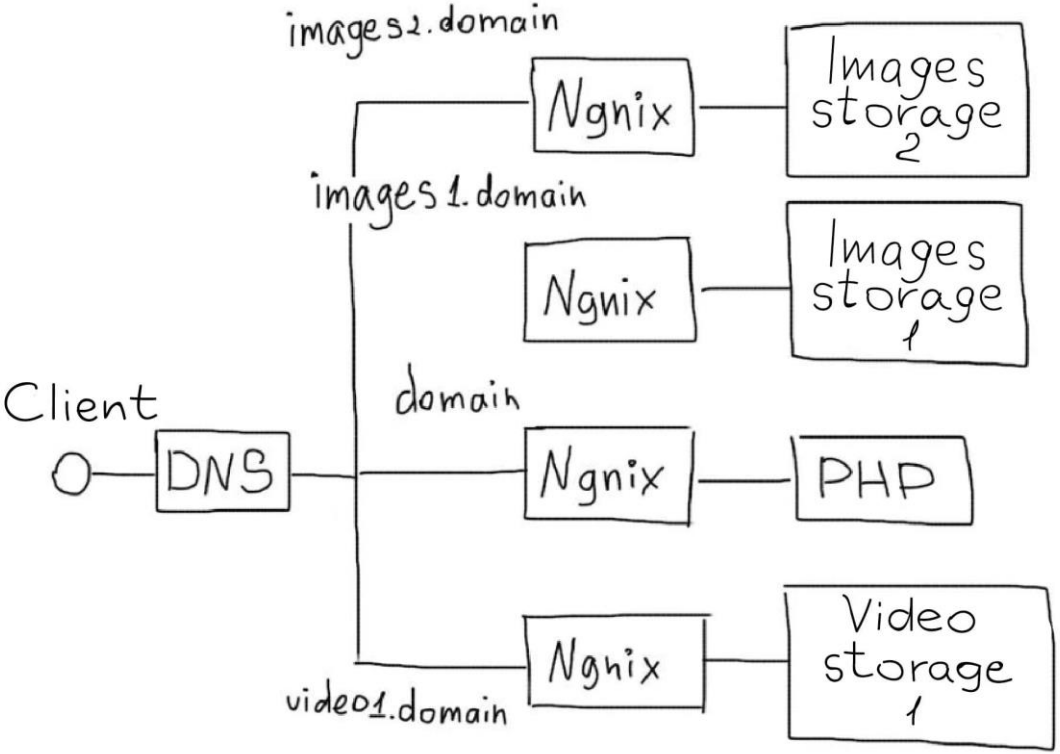
Simple web application #6



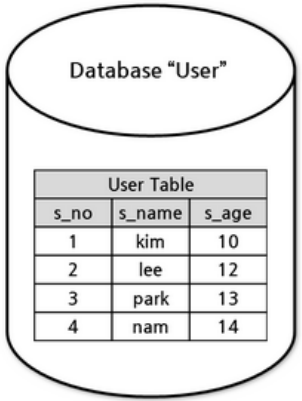
Simple web application #7



Simple web application #8

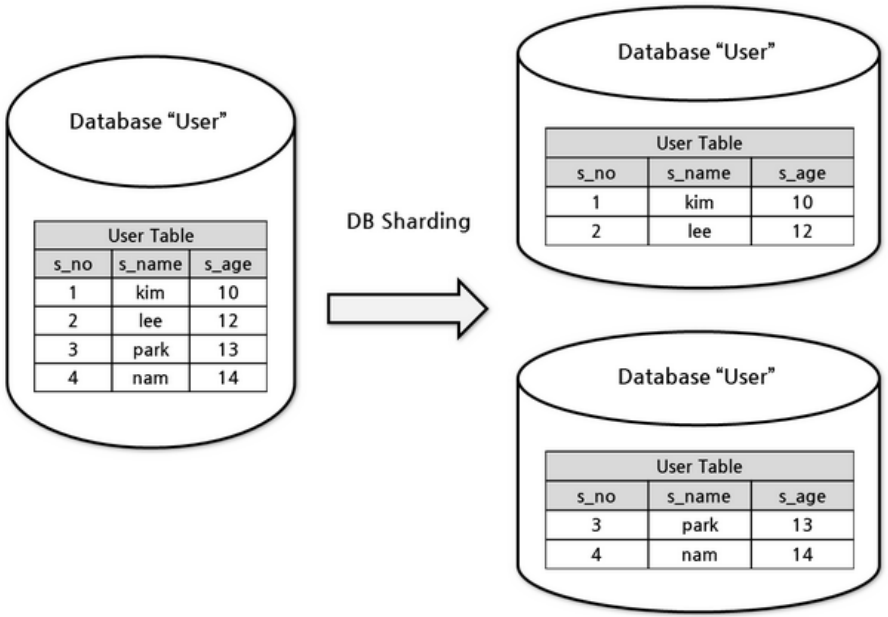


Big Table

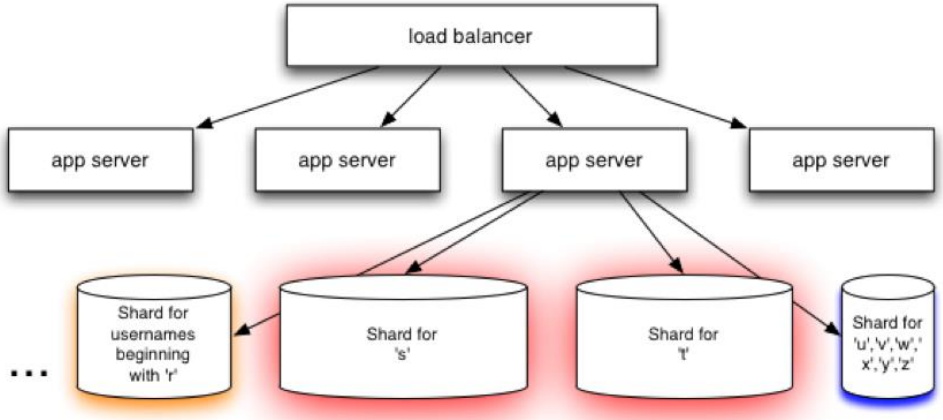


User Table		
s_no	s_name	s_age
1	kim	10
2	lee	12
3	park	13
4	nam	14

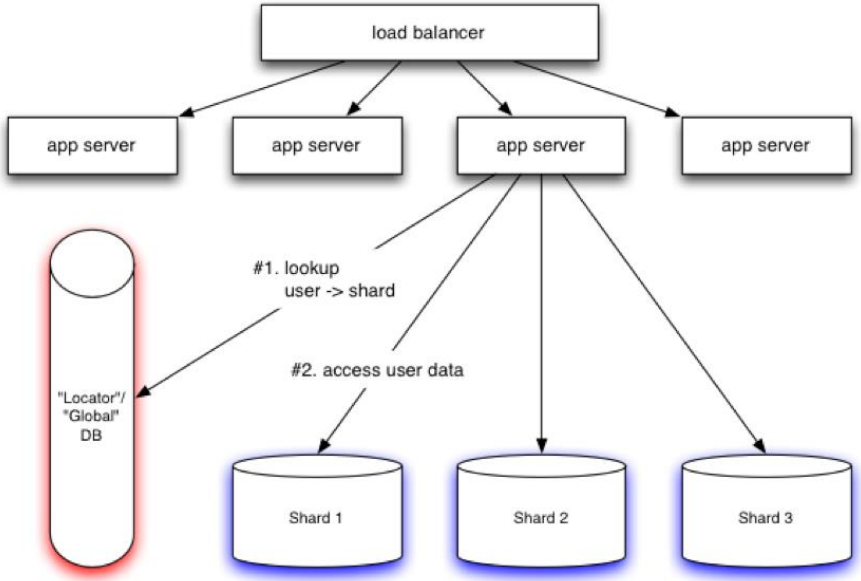
Sharding


Database “User”
User Table
s_no s_name s_age
1 kim 10
2 lee 12
3 park 13
4 nam 14
DB Sharding
Database “User”
User Table
s_no s_name s_age
1 kim 10
2 lee 12
Database “User”
User Table
s_no s_name s_age
3 park 13
4 nam 14

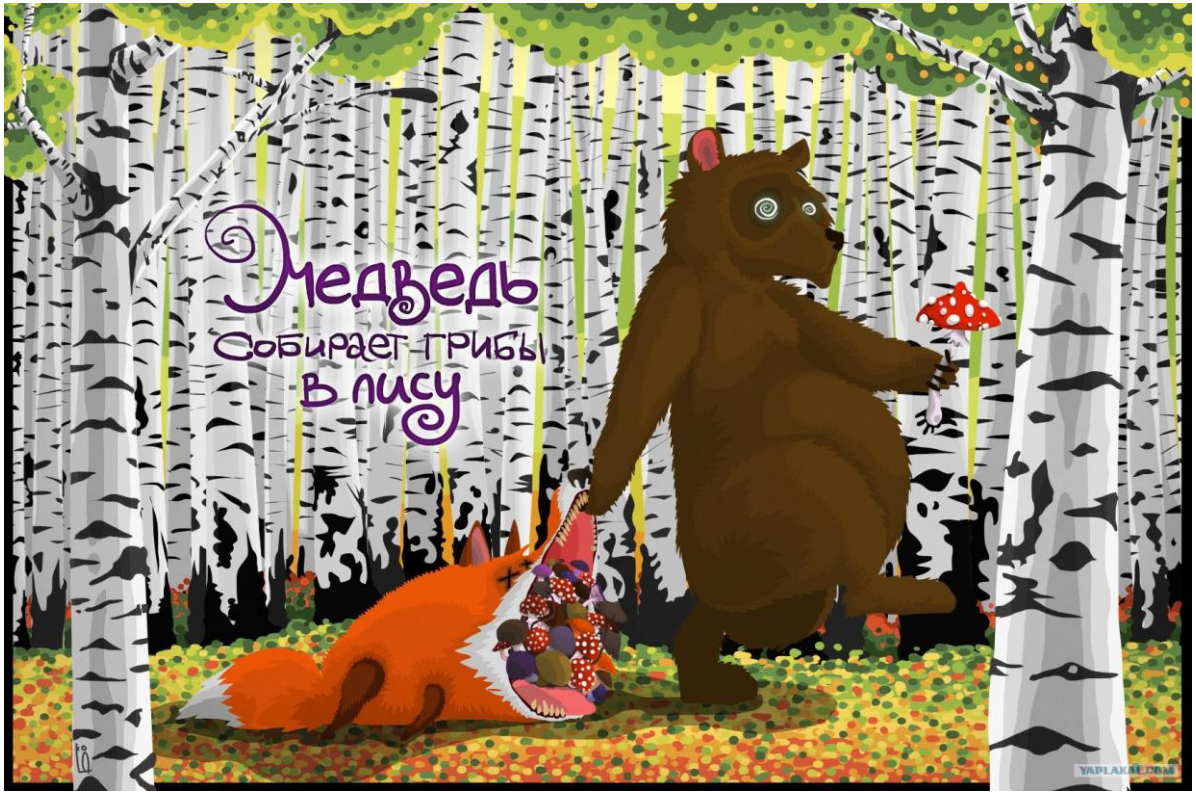
Classic Sharding



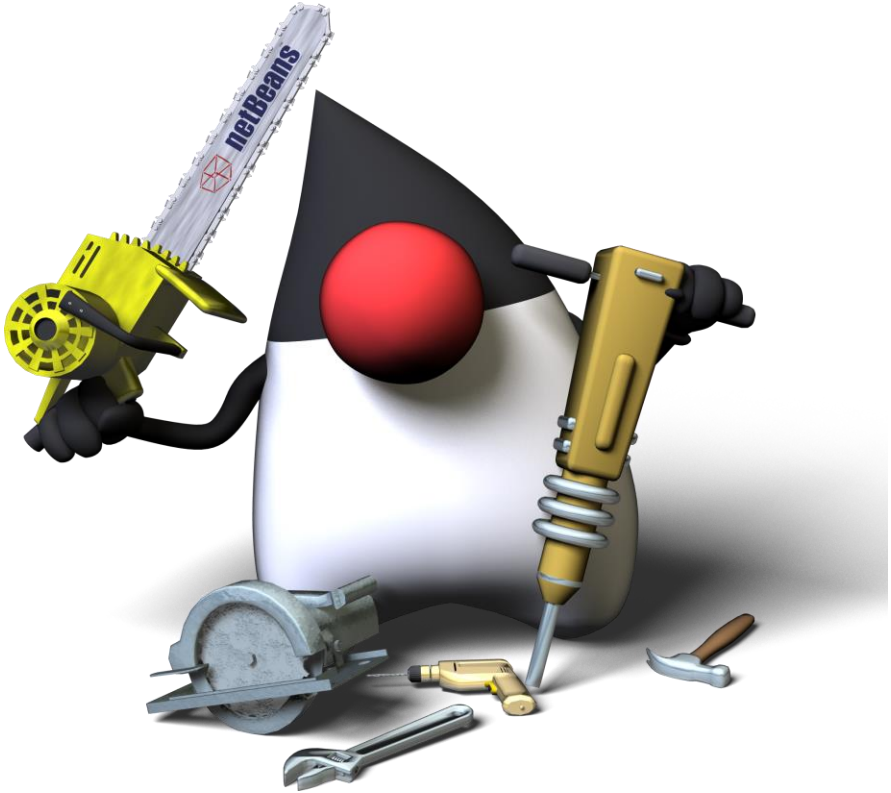
Sharding with Locator



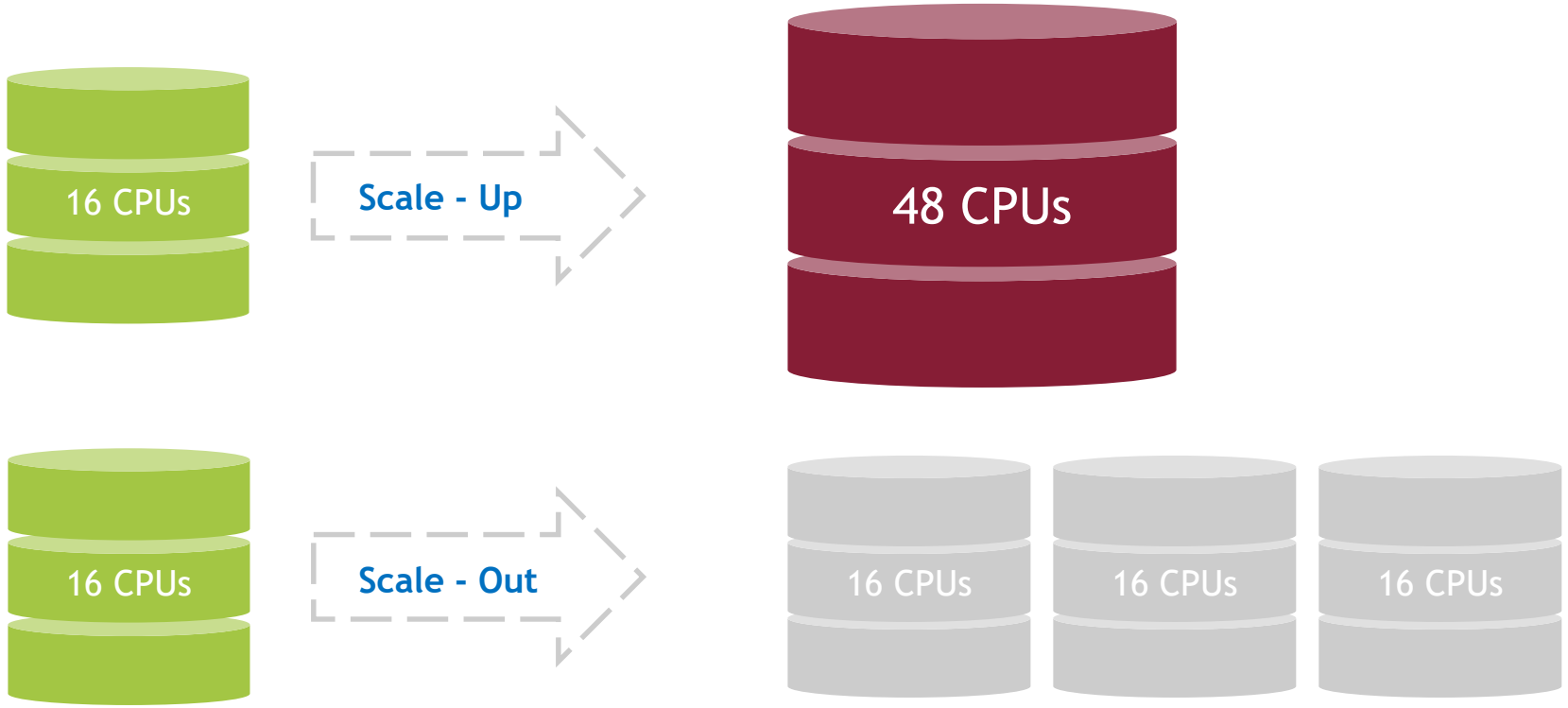
You'd measure performance!



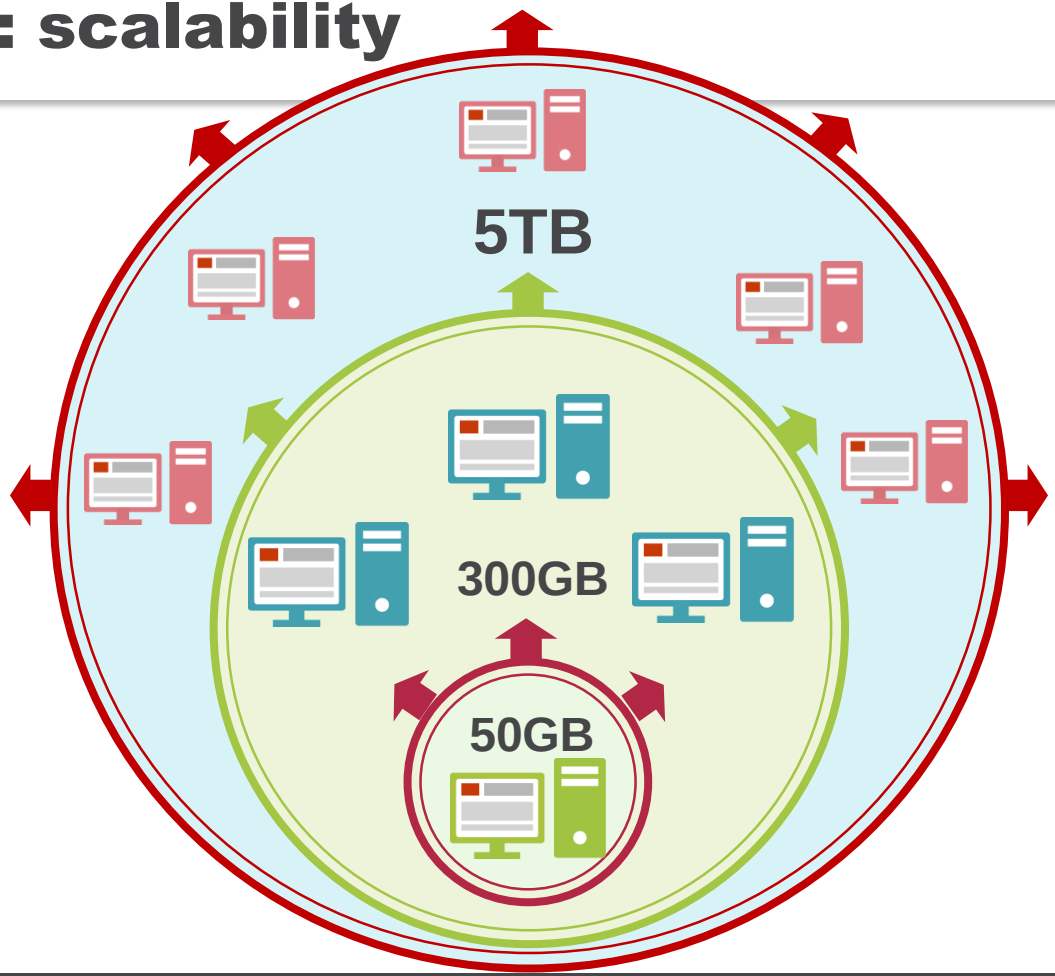
Case #3 : Supercomputer for President



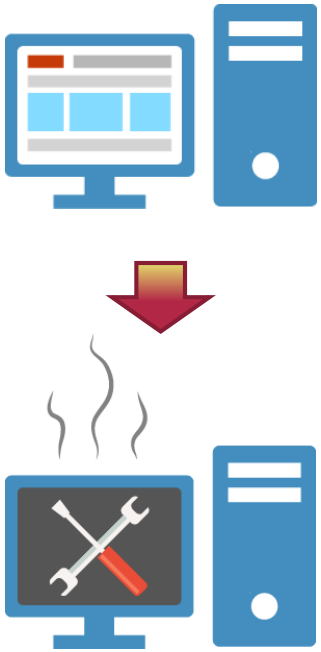
Motivation: Scale-up vs scale-out



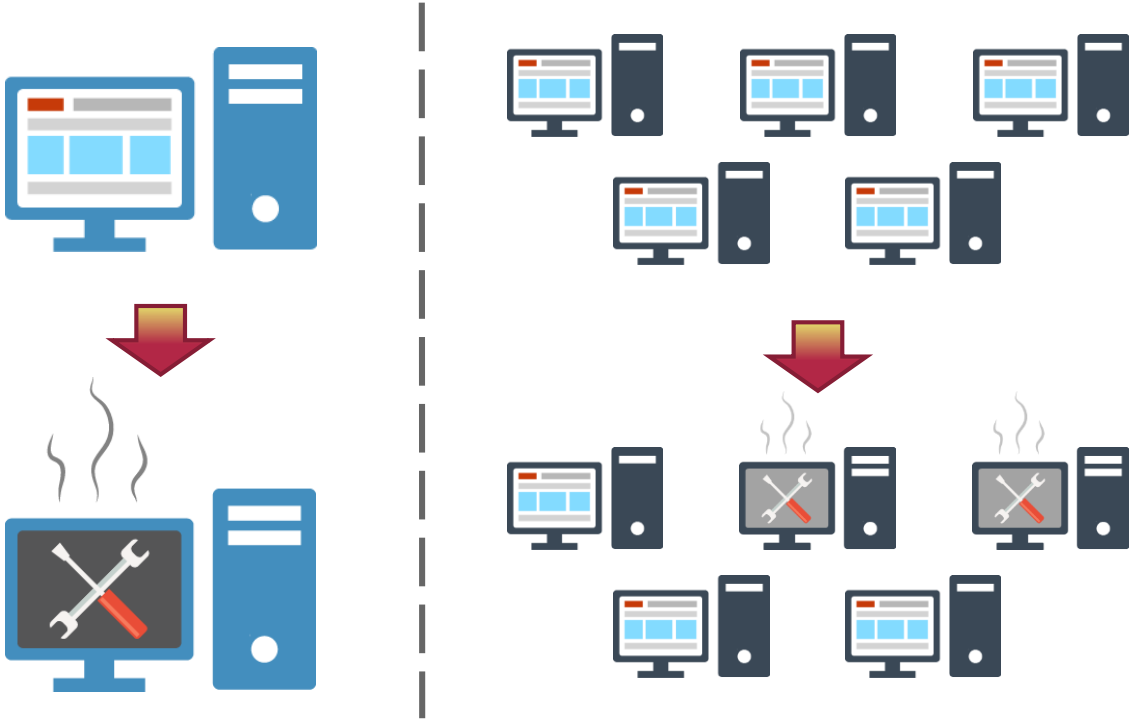
Motivation: scalability



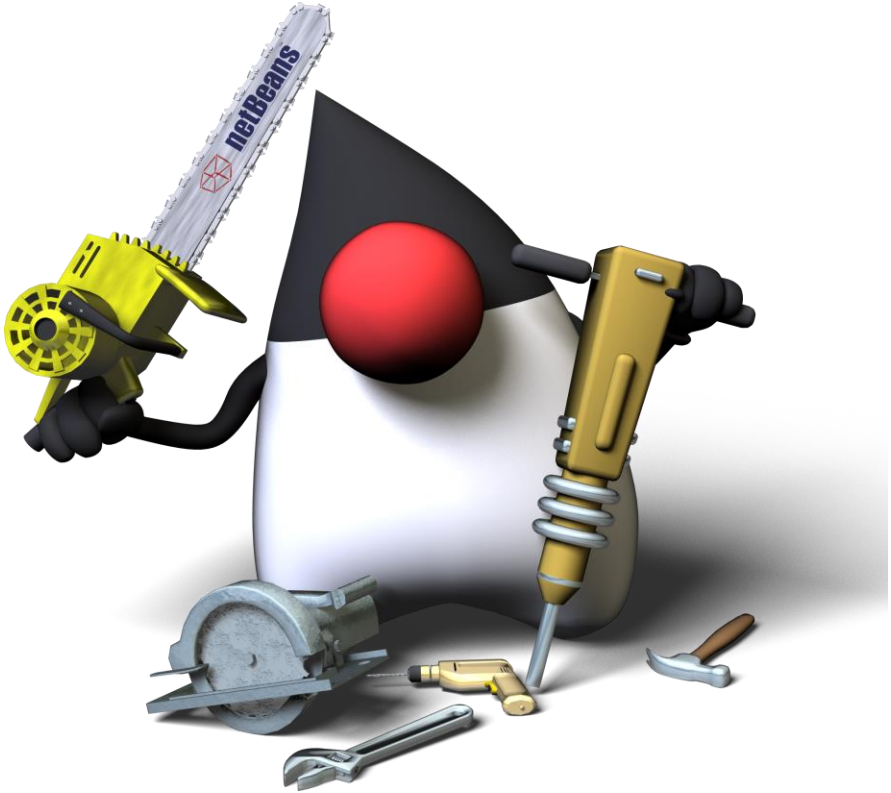
Motivation: Fault tolerance



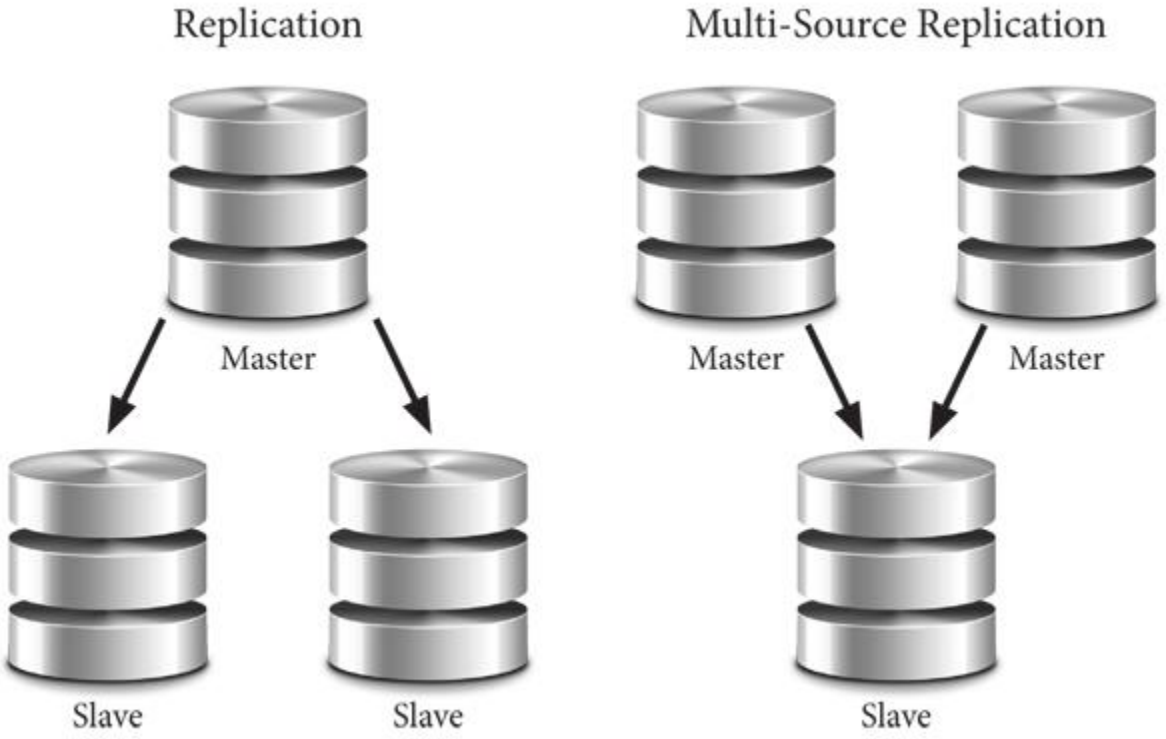
Motivation: Fault tolerance



Case #4 : DDos attack against retail company

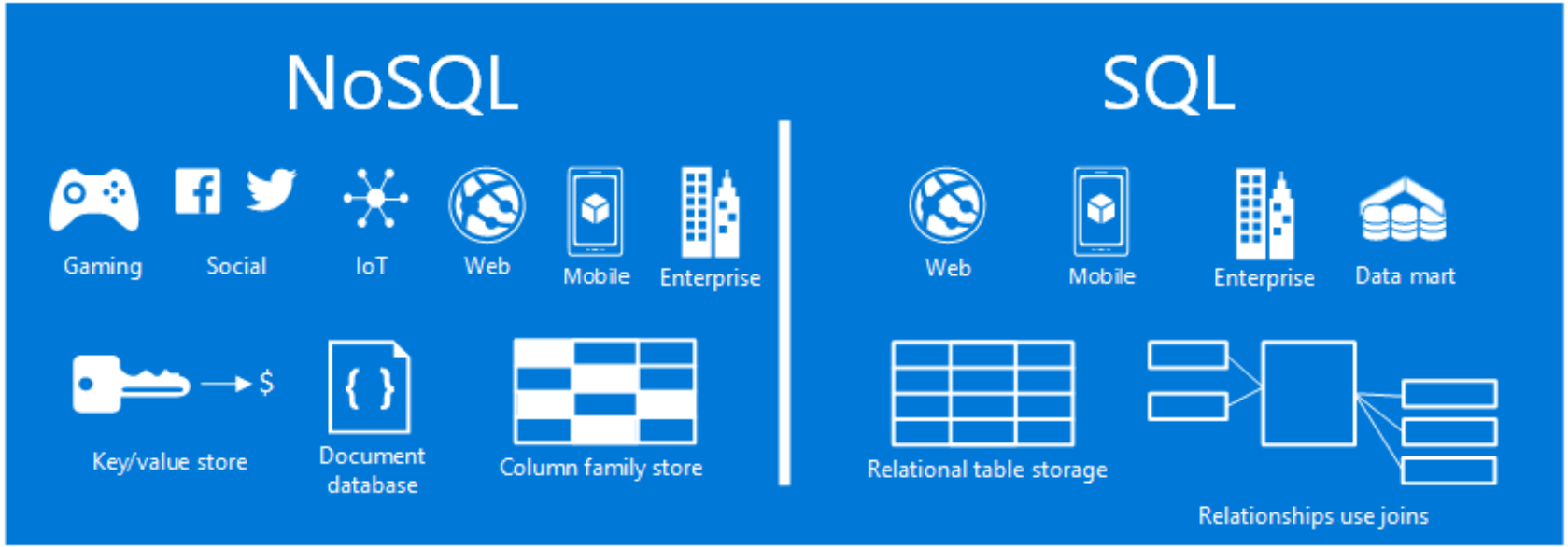


Replication

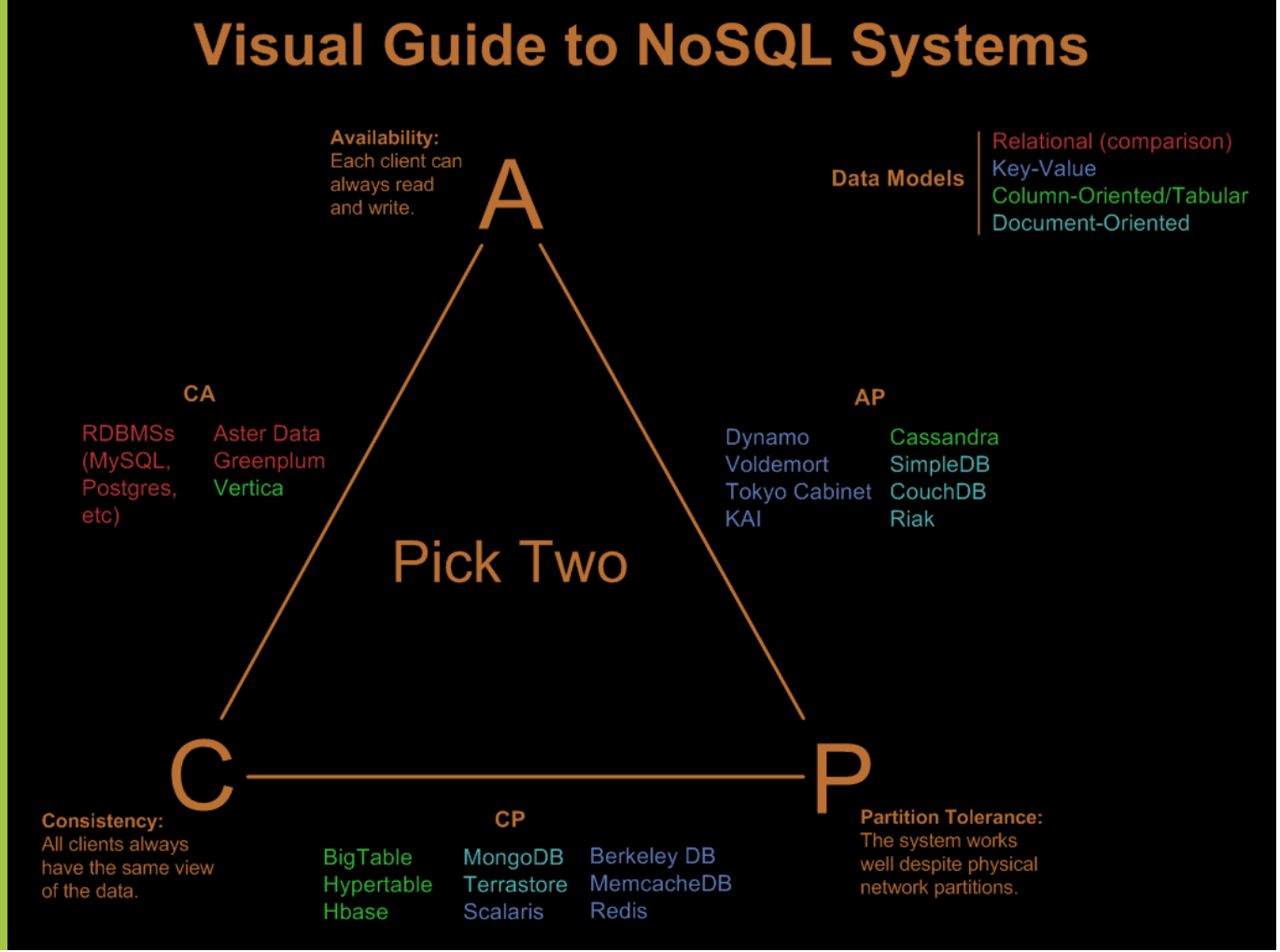


NOSQL

Replication

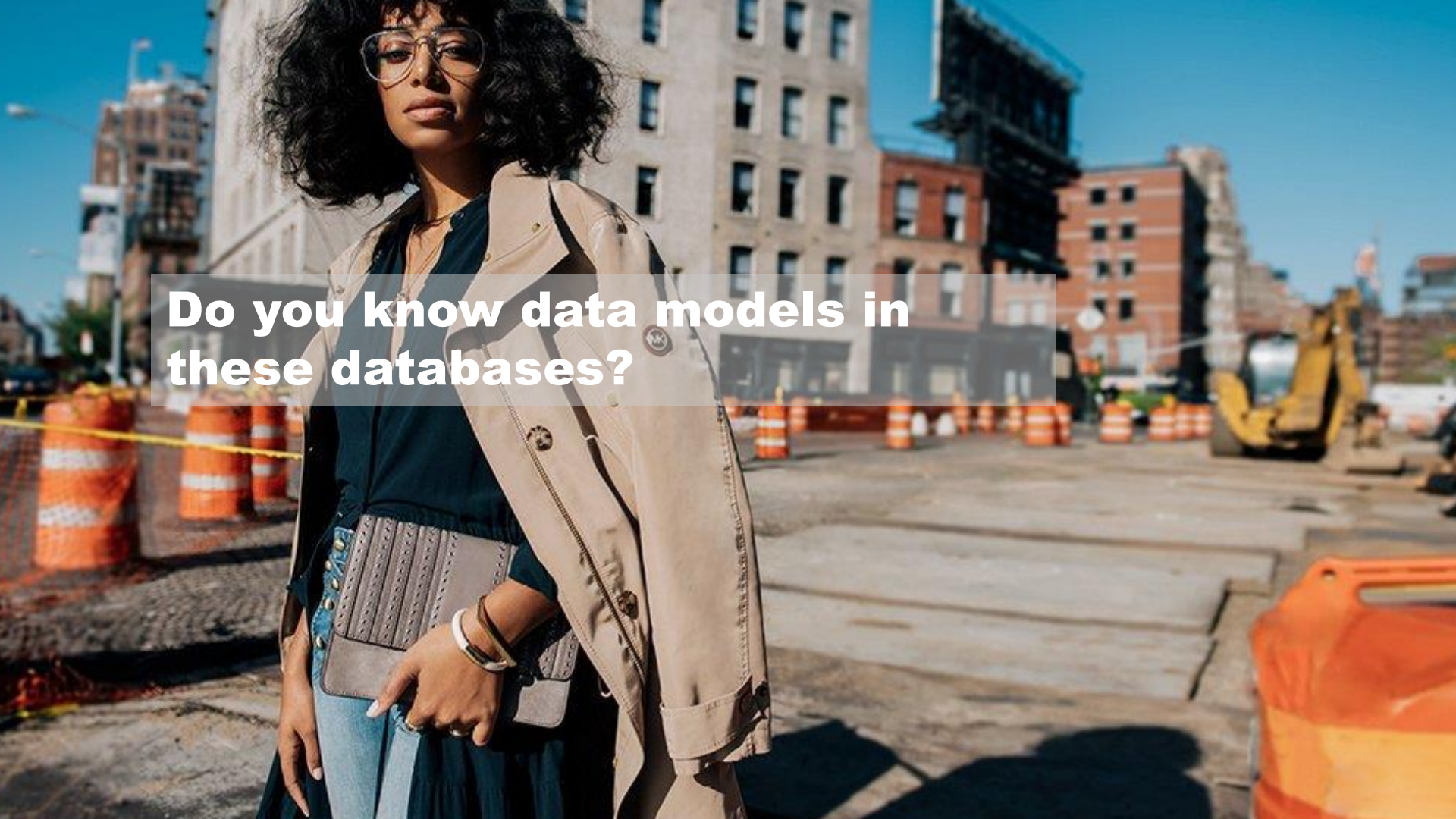


CAP Theorem


Visual Guide to NoSQL Systems
Availability:
Each client can always read and write.
A
Data Models
Relational (comparison)
Key-Value
Column-Oriented/Tabular
Document-Oriented
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Pick Two
C
P
Consistency:
All clients always have the same view of the data.
CP
BigTable
Hypertable
Hbase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance:
The system works well despite physical network partitions.

Databases

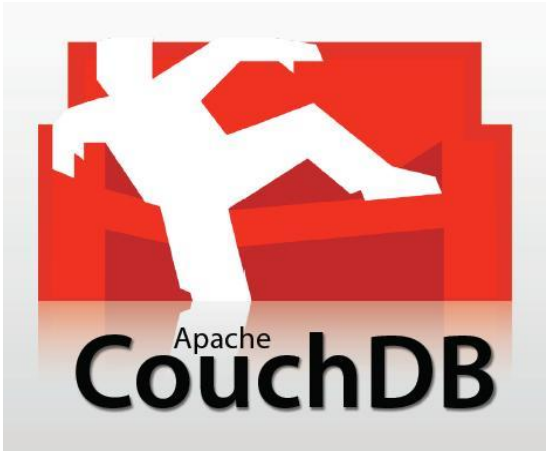
- Cassandra
- Hbase
- Neo4j
- Voldemort


Do you know data models in these databases?

Databases with types

- Cassandra (column family)
- Hbase (column family)
- Neo4j (graph)
- Voldemort (key-value)

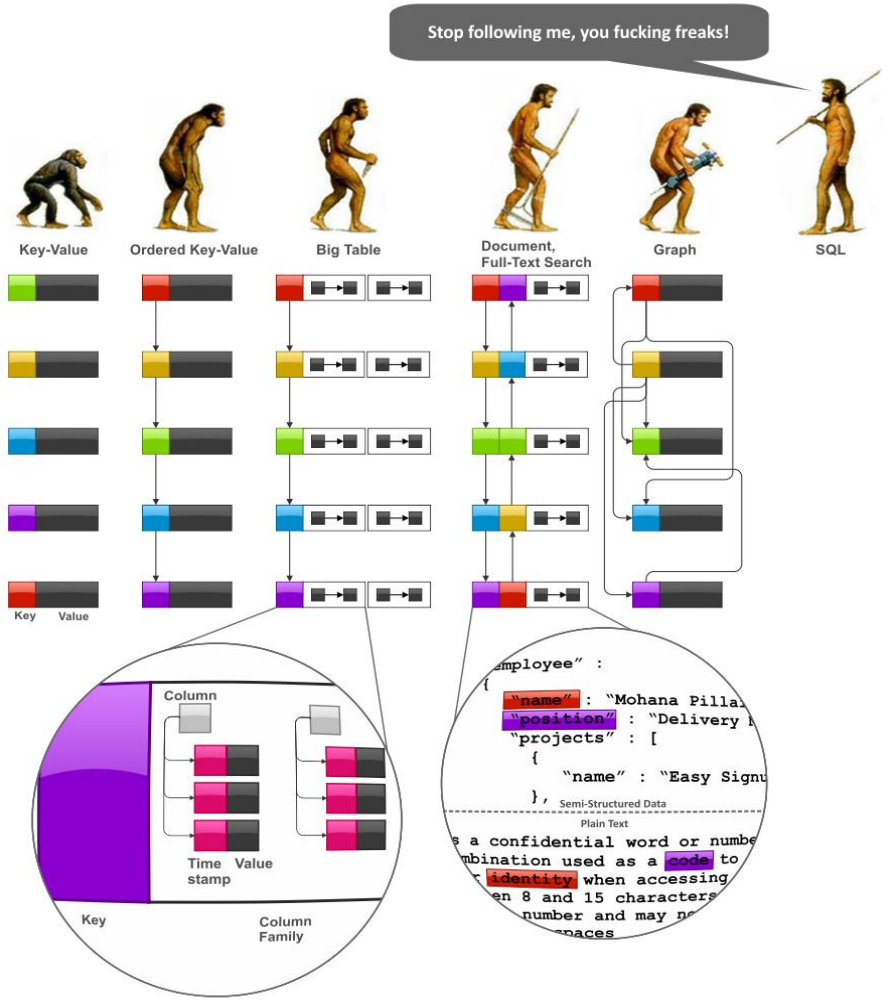
Database party



What's the problem with RBDMS's

- Caching
- Master/Slave
- Cluster
- Table Partitioning
- Sharding

Evolution


Stop following me, you fucking freaks!
Key-Value
Ordered Key-Value
Big Table
Document, Full-Text Search
Graph
SQL
Key
Value
Column
Time stamp
Value
Key
Column Family
"name" : "Mohana Pilla
"position" : "Delivery
"projects" : [
"name" : "Easy Signu
Semi-Structured Data
Plain Text
a confidential word or numbe
bination used as a code to
identity when accessing
en 8 and 15 characters
number and may n

Flowers in your garden

Data Model	Performance	Scalability	Flexibility	Complexity	Functionality
Key–value Stores	high	high	high	none	variable (none)

Flowers in your garden

Data Model	Performance	Scalability	Flexibility	Complexity	Functionality
Key–value Stores	high	high	high	none	variable (none)
Column Store	high	high	moderate	low	minimal

Flowers in your garden

Data Model	Performance	Scalability	Flexibility	Complexity	Functionality
Key–value Stores	high	high	high	none	variable (none)
Column Store	high	high	moderate	low	minimal
Document Store	high	variable (high)	high	low	variable (low)

Flowers in your garden

Data Model	Performance	Scalability	Flexibility	Complexity	Functionality
Key–value Stores	high	high	high	none	variable (none)
Column Store	high	high	moderate	low	minimal
Document Store	high	variable (high)	high	low	variable (low)
Graph Database	variable	variable	high	high	graph theory

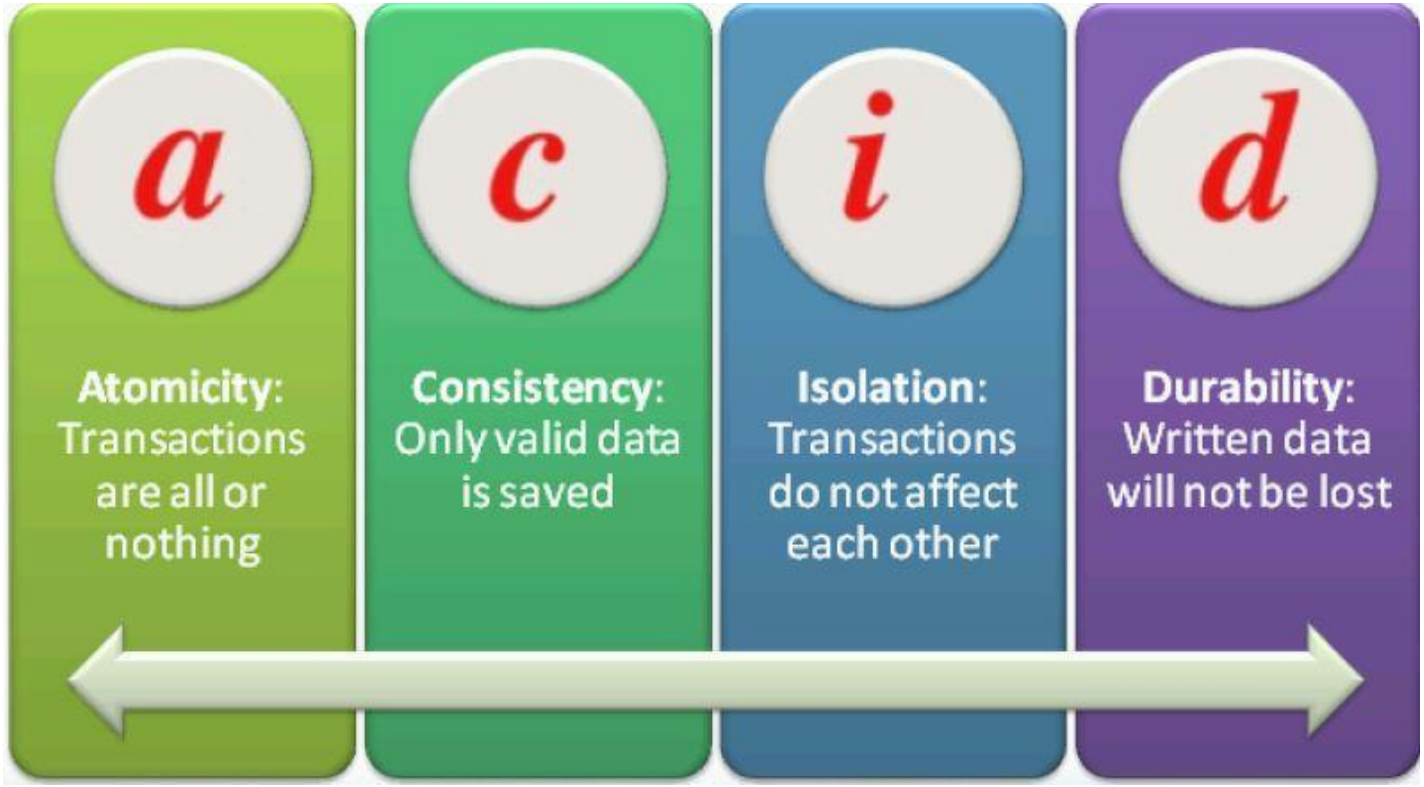
Flowers in your garden

Data Model	Performance	Scalability	Flexibility	Complexity	Functionality
Key–value Stores	high	high	high	none	variable (none)
Column Store	high	high	moderate	low	minimal
Document Store	high	variable (high)	high	low	variable (low)
Graph Database	variable	variable	high	high	graph theory
Relational Database	variable	variable	low	moderate	relational algebra

Gentle NoSQL

- Scalability
- Nodes and Data Centers
- Easy to add new server
- Specific data model
- Eventual consistency

ACID in SQL


a
Atomicity:
Transactions are all or nothing
c
Consistency:
Only valid data is saved
i
Isolation:
Transactions do not affect each other
d
Durability:
Written data will not be lost

Atomicity in NoSQL

- read-write-modify (CAS)
- key/row manipulation is atomic
- API for atomic operations
- bad support of transactions

BASE

- *basic availability* – all queries will be finished
- *soft state* – state can be changed without writing
- *eventual consistency*

Flowers in your garden

Database	Data model	Query API	Data storage system
Cassandra	Column Family	Thrift	Memtable/SSTable

Flowers in your garden

Database	Data model	Query API	Data storage system
Cassandra	Column Family	Thrift	Memtable/SSTable
CouchDB	Documents	Map/Reduce	Append-only-B-tree

Flowers in your garden

Database	Data model	Query API	Data storage system
Cassandra	Column Family	Thrift	Memtable/SSTable
CouchDB	Documents	Map/Reduce	Append-only-B-tree
Hbase	Column Family	Thrift, REST	Memtable/SSTable on HDFS

Flowers in your garden

Database	Data model	Query API	Data storage system
Cassandra	Column Family	Thrift	Memtable/SSTable
CouchDB	Documents	Map/Reduce	Append-only-B-tree
Hbase	Column Family	Thrift, REST	Memtable/SSTable on HDFS
MongoDB	Documents	Cursor	B-tree

Flowers in your garden

Database	Data model	Query API	Data storage system
Cassandra	Column Family	Thrift	Memtable/SSTable
CouchDB	Documents	Map/Reduce	Append-only-B-tree
Hbase	Column Family	Thrift, REST	Memtable/SSTable on HDFS
MongoDB	Documents	Cursor	B-tree
Neo4j	Edges/Verticies	Graph	On-disk linked lists

Flowers in your garden

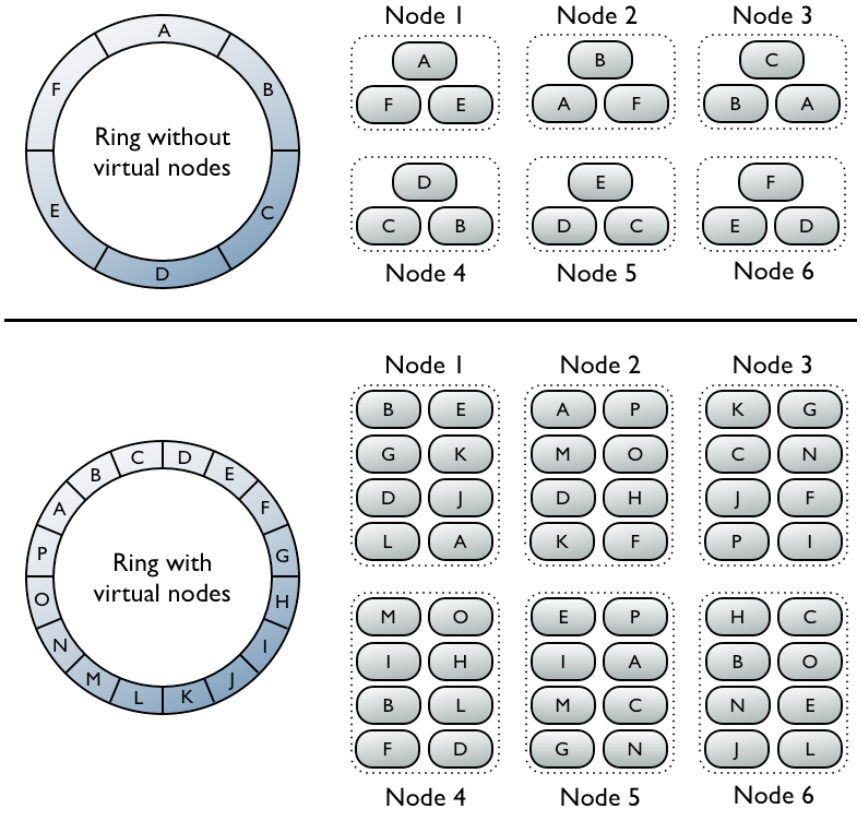
Database	Data model	Query API	Data storage system
Cassandra	Column Family	Thrift	Memtable/SSTable
CouchDB	Documents	Map/Reduce	Append-only-B-tree
Hbase	Column Family	Thrift, REST	Memtable/SSTable on HDFS
MongoDB	Documents	Cursor	B-tree
Neo4j	Edges/Verticies	Graph	On-disk linked lists
Riak	Key/Value	Nested hashes, REST	Hash


What about queries?

Mongo

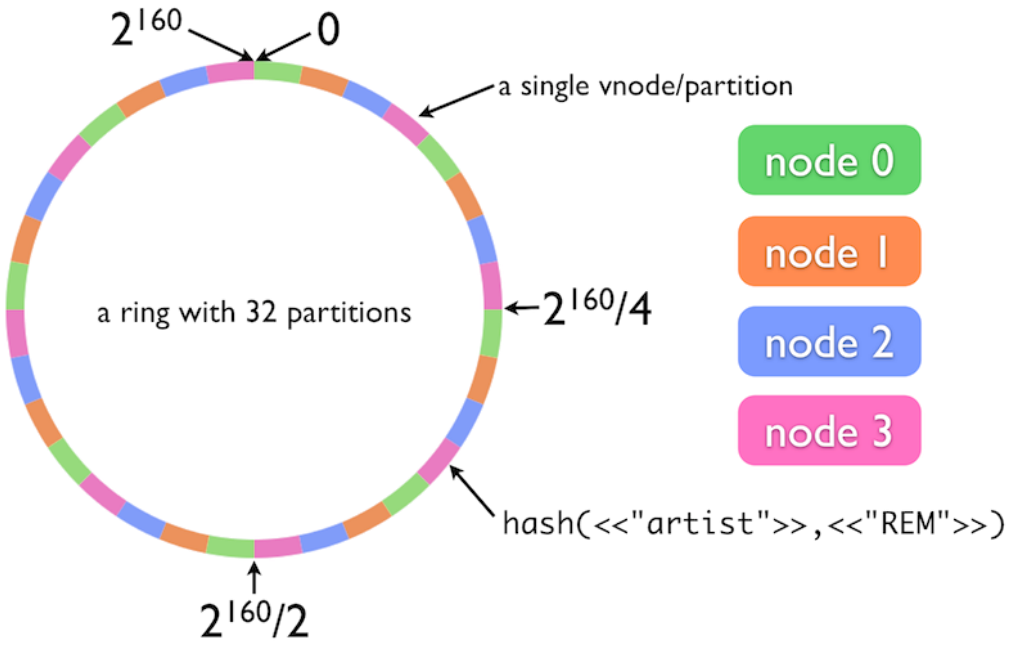
- 16 mb
- JavaScript at the bottom
- 2d, 3d, B-tree indexes
- 3.0 version is very hot
- integration with Kafka

Cassandra



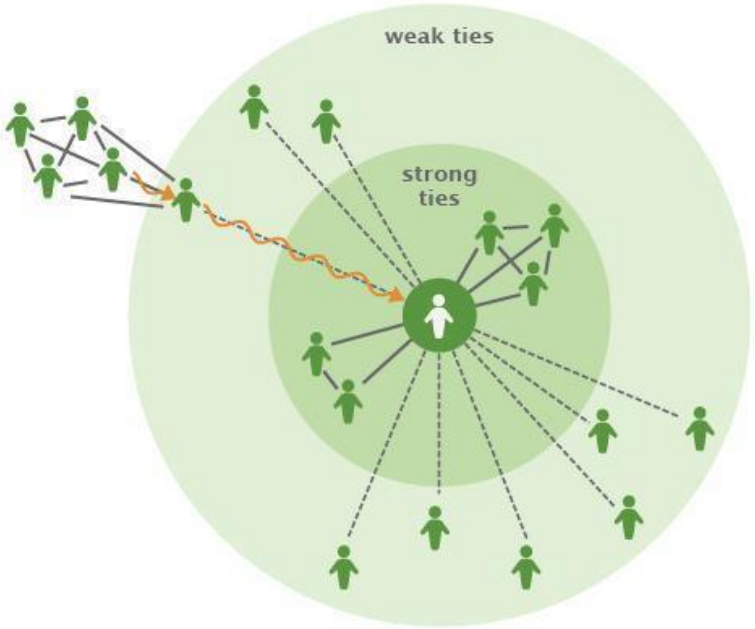
- CQL
- 2 billions columns in row
- No ACID, of course
- Can spend all your RAM
- JVM - based

Riak



- Links to another keys
- REST
- Consistency level in each query
- Ring of nodes

Neo4j



- Vertices, edges
- ACID
- REST API + Cypher
- 2d index
- Not so good for distributed data


And what should we choose?

Network Rule

Can your data be presented as network or graph?

If yes -> Neo4j

If no -> continue

BigData Rule

Do you have TB or PB of data?

If yes -> NoSQL

If no -> Maybe you should stay with SQL?

Easy Rule

Do you need in more complex operation than read/write by key?

If yes -> continue

If no -> Riak, Memcached

Hierarchy Rule

Do you have nested data? Is R >> W in your system?

If yes -> MongoDB, CouchDB

If no -> continue

Hadoop Rule

Is Hadoop integration required? Can your data be presented as flat table?

If yes -> Hbase

If no -> continue

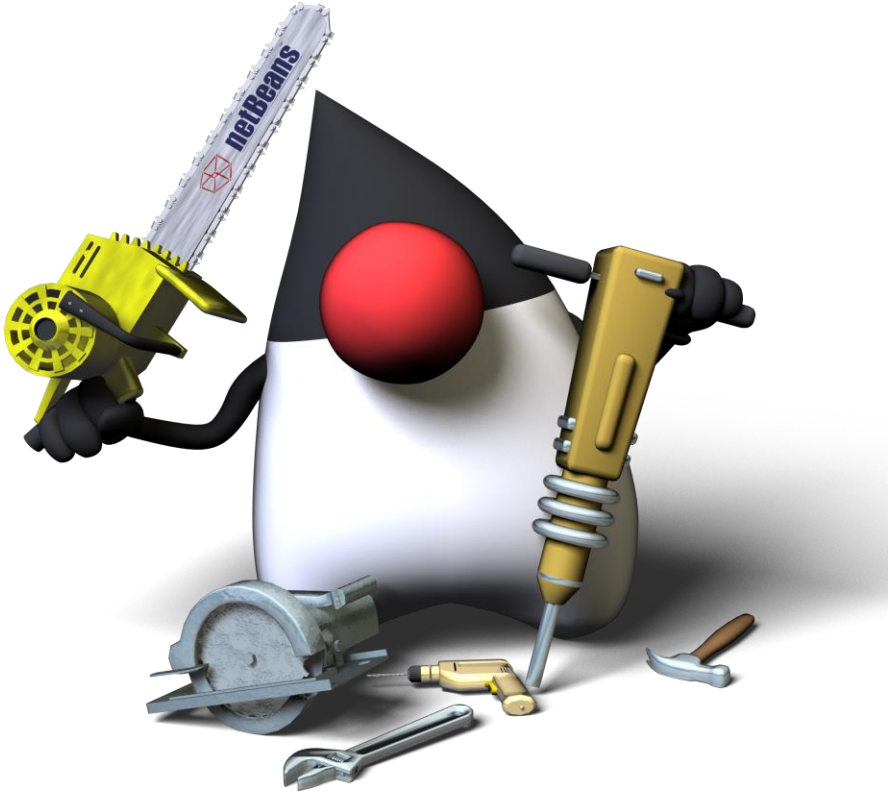
Availability Rule

Do you need in high availability? Are you agree with eventual consistency?

If yes -> Cassandra

If no -> continue

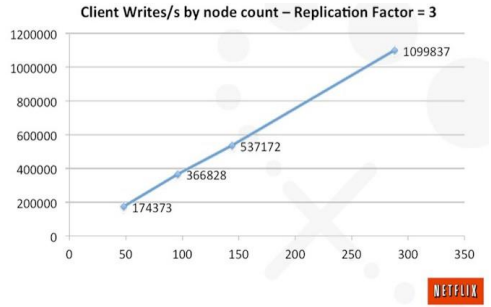
Case #5 : Go back to PostgreSQL



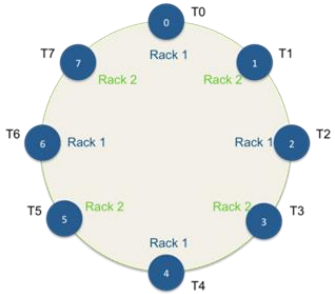
<epam>

CASSANDRA

Cassandra Features



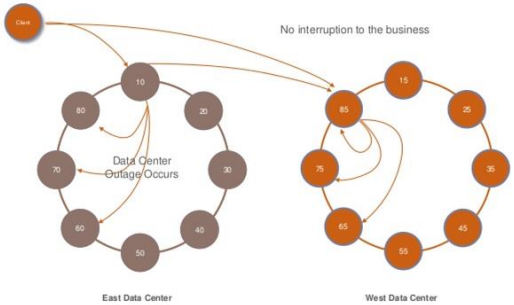
LINEAR SCALABILITY



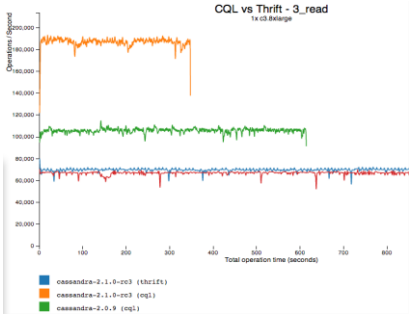
MASTER-LESS ARCHITECTURE

ALL	QUORUM	ONE
Strong Consistency	Quorum == (nodes / 2) + 1	High Availability / Eventual Consistency

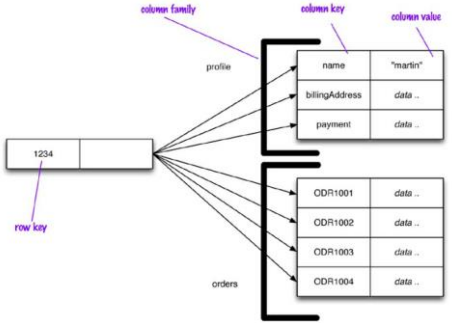
TUNABLE CONSISTENCY



MULTI DATA CENTER SUPPORT

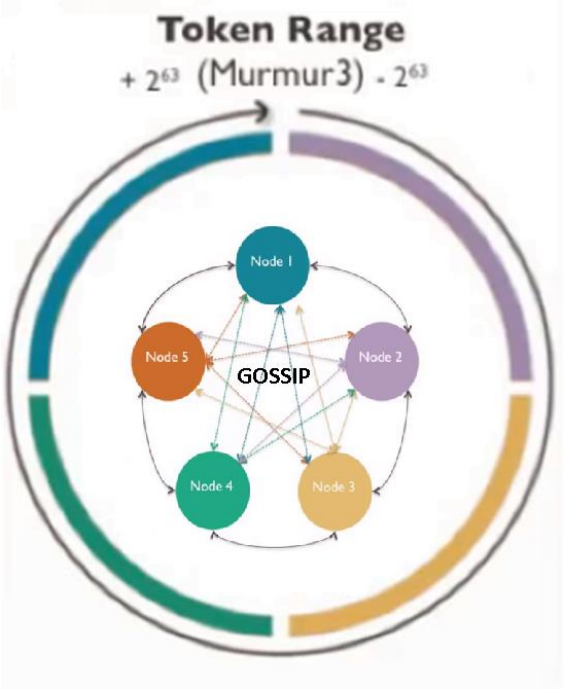


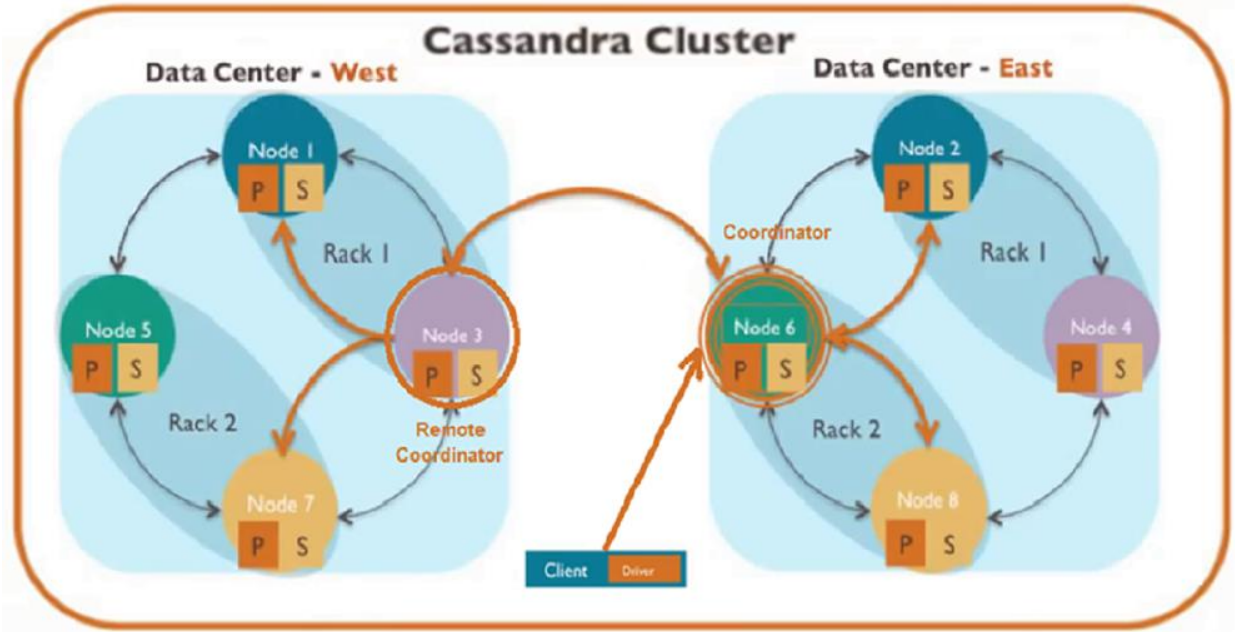
MASSIVE WRITE AND FAST RESPONSE



FLEXIBLE DATA MODEL

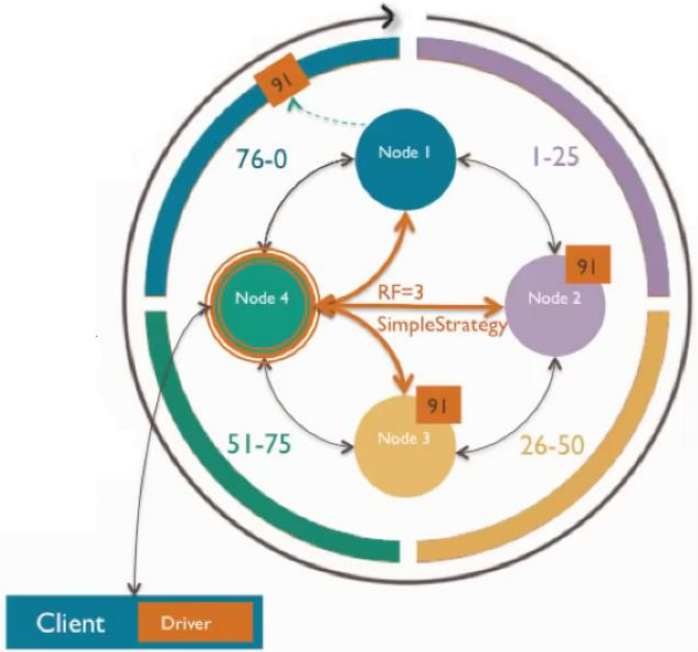
Cassandra “Ring” Architecture

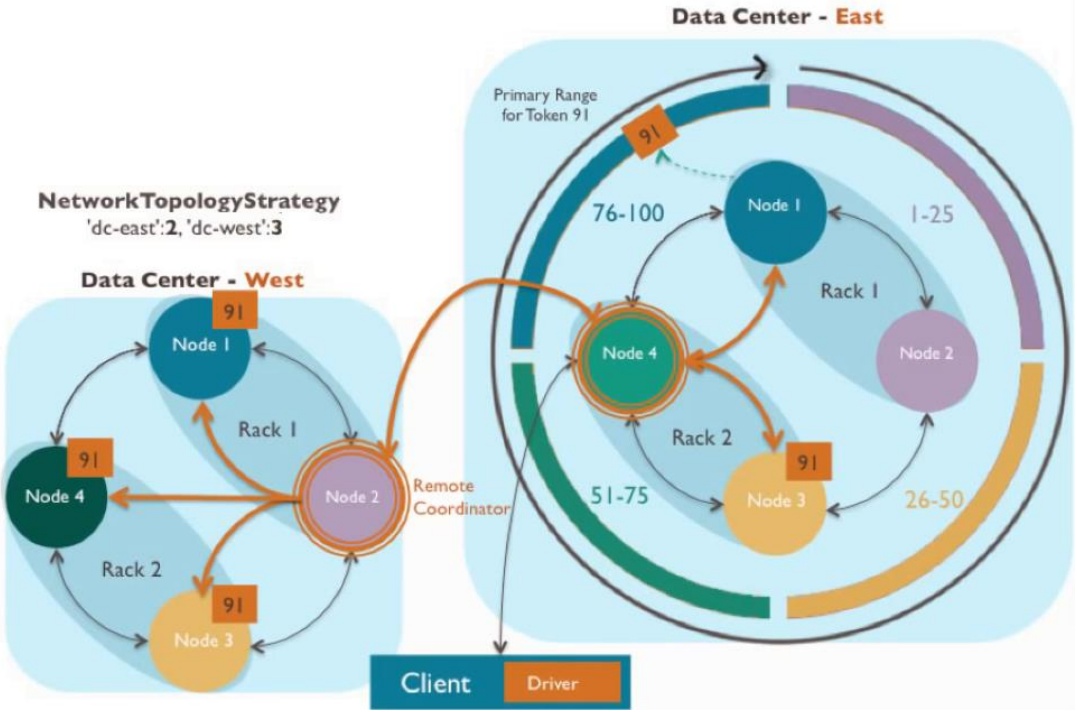




Cassandra Replication

SimpleStrategy





NetworkTopologyStrategy

Cassandra Tunable Consistency

Write Consistency

ALL

EACH_QUORUM

QUORUM

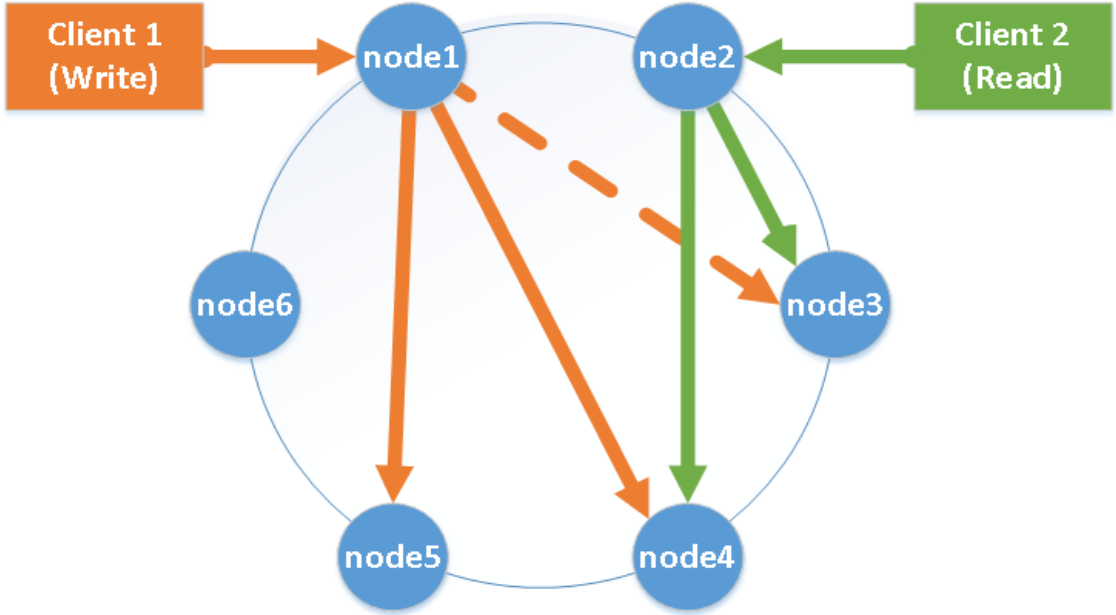
LOCAL_QUORUM

ONE

TWO

LOCAL_ONE

ANY



Replication Factor: 3
Write Consistency Level: QUORUM
Read Consistency Level: QUORUM

Read Consistency

ALL

EACH_QUORUM

QUORUM

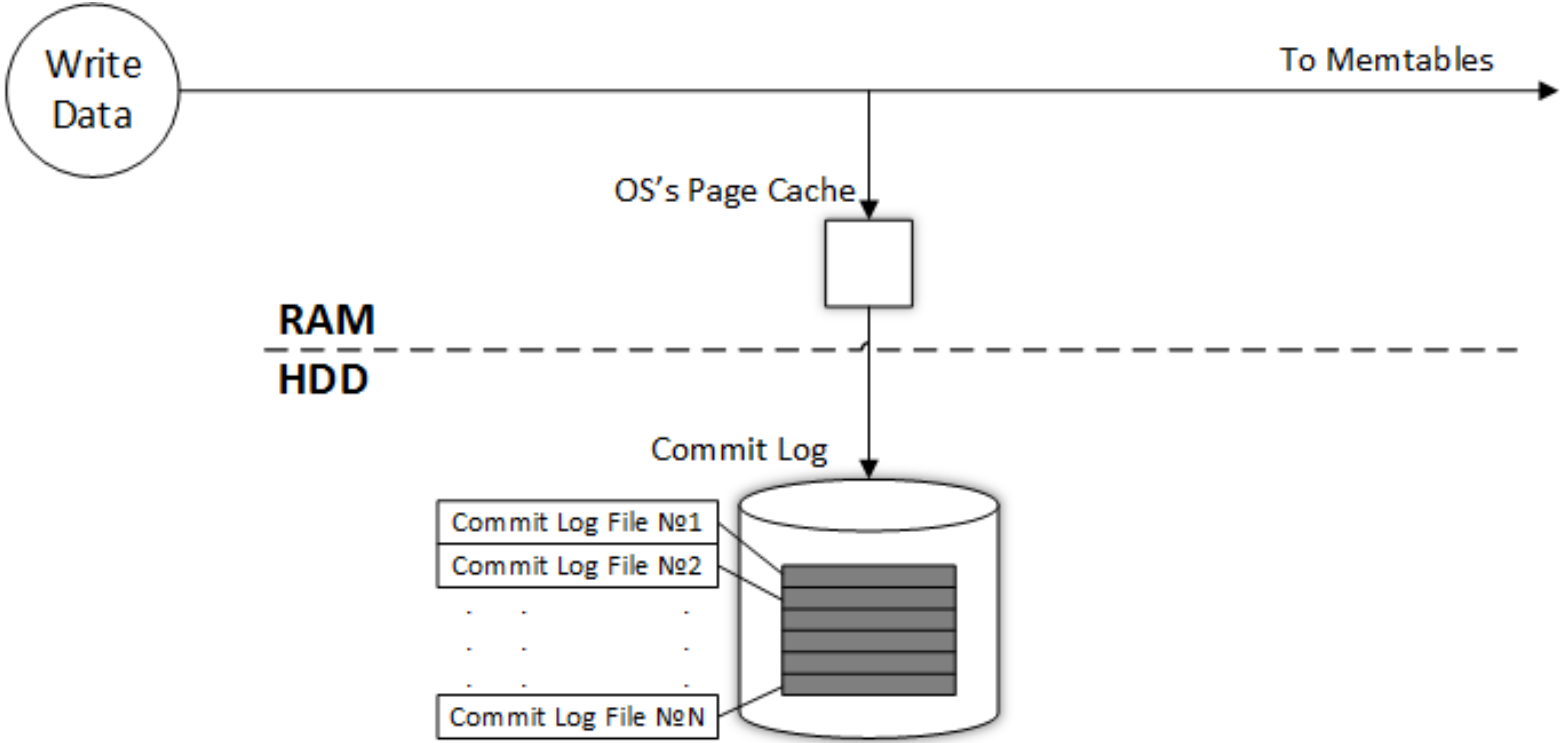
LOCAL_QUORUM

ONE

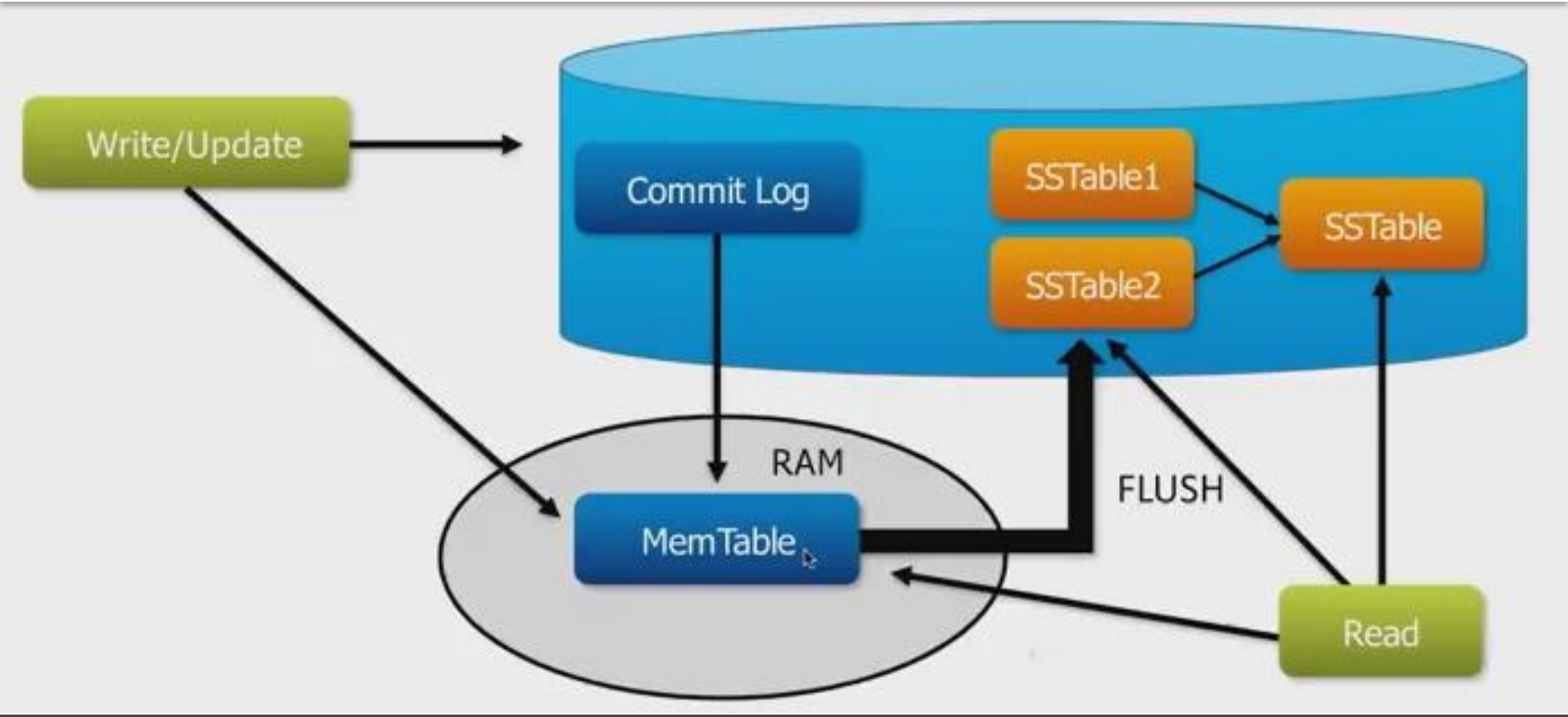
TWO

LOCAL_ONE

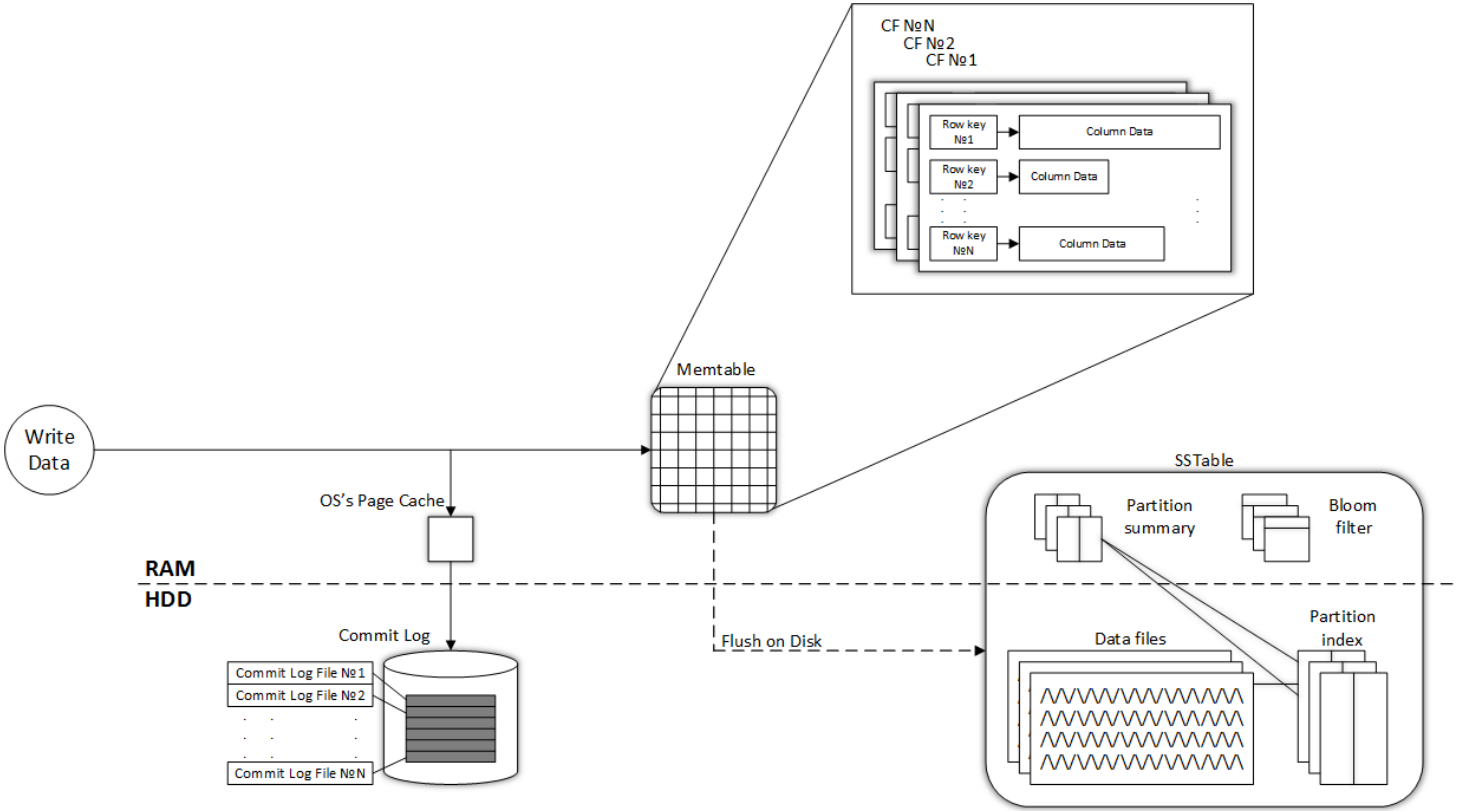
Cassandra write path (COMMITLOG)



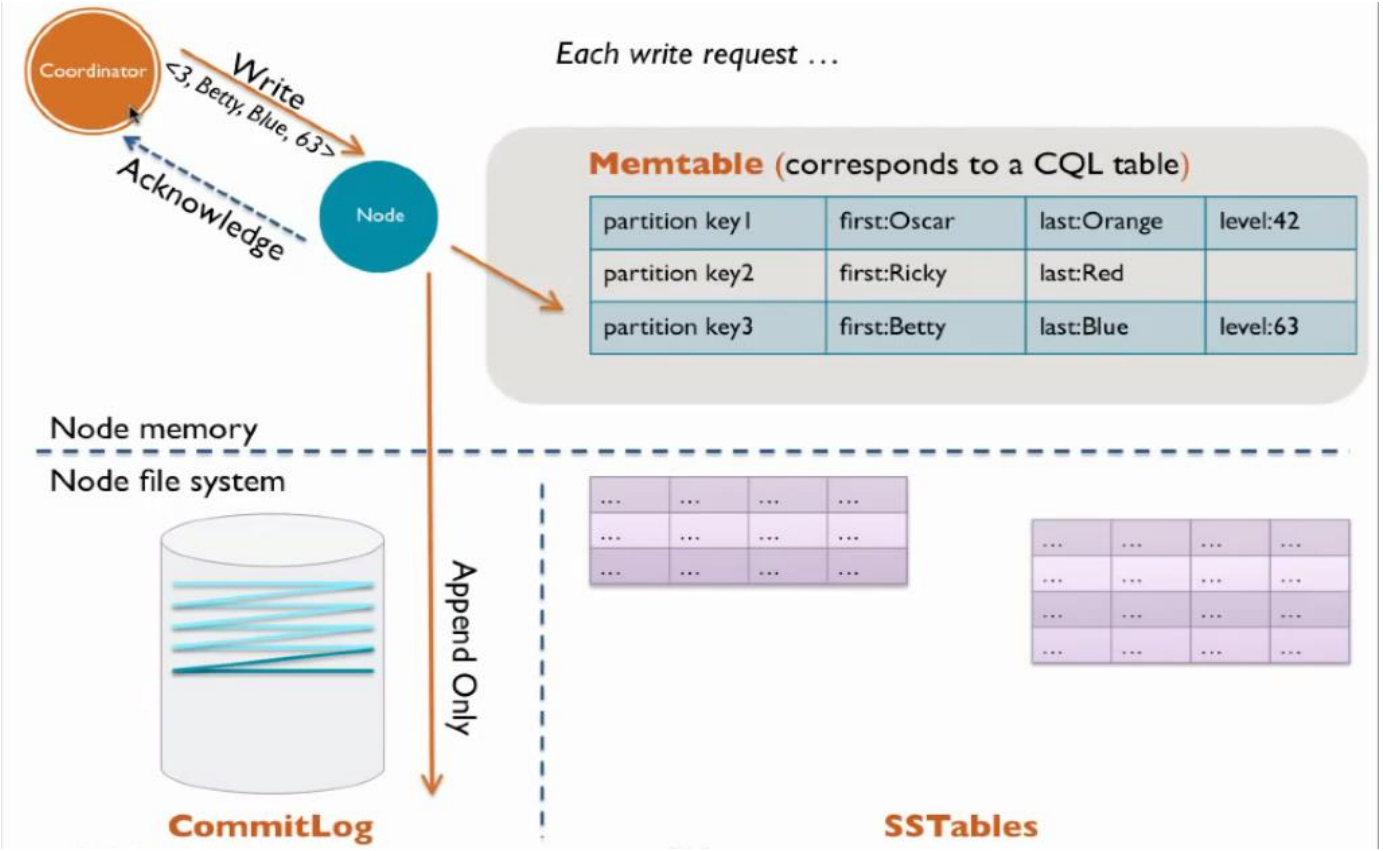
Cassandra write path (MemTable)



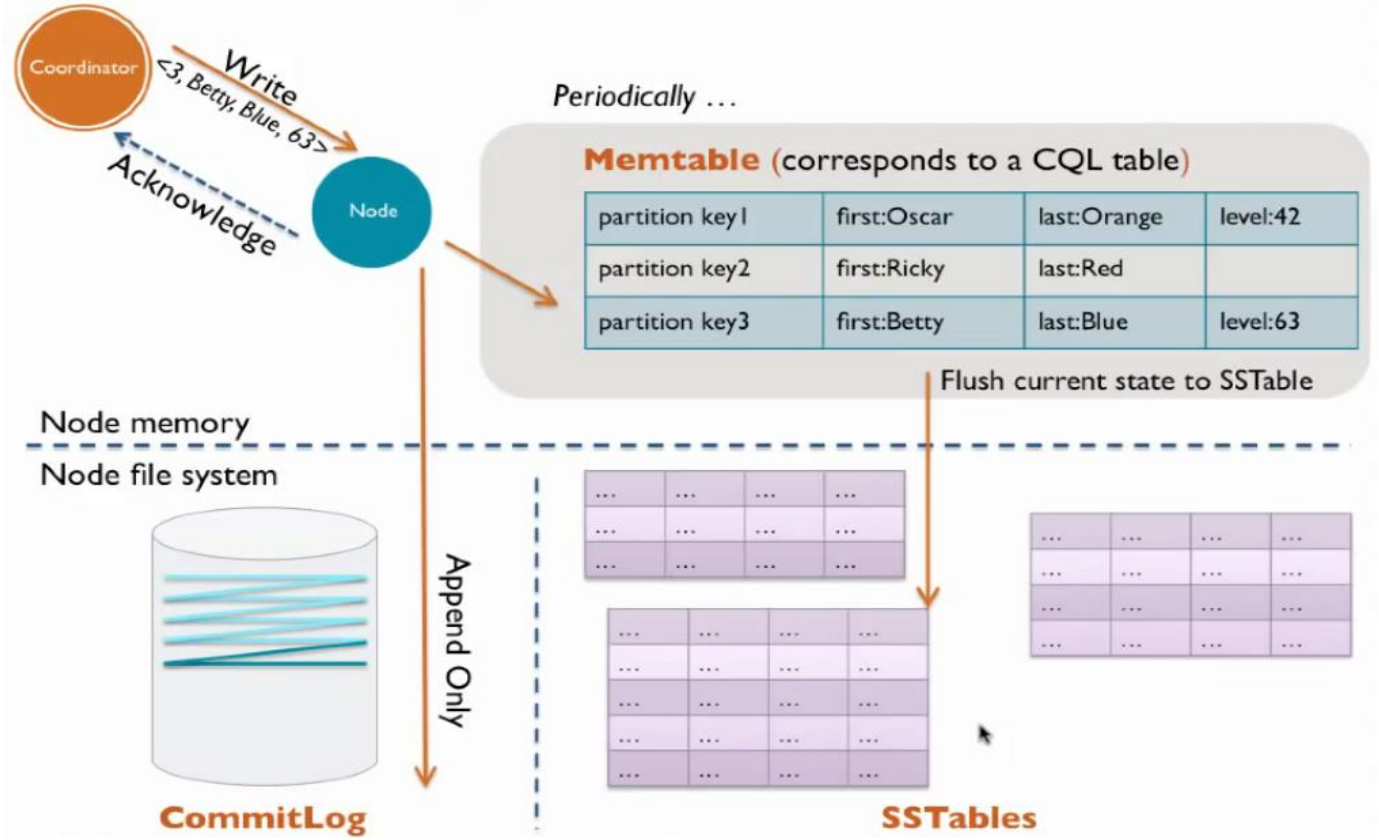
Cassandra write path (SSTables)



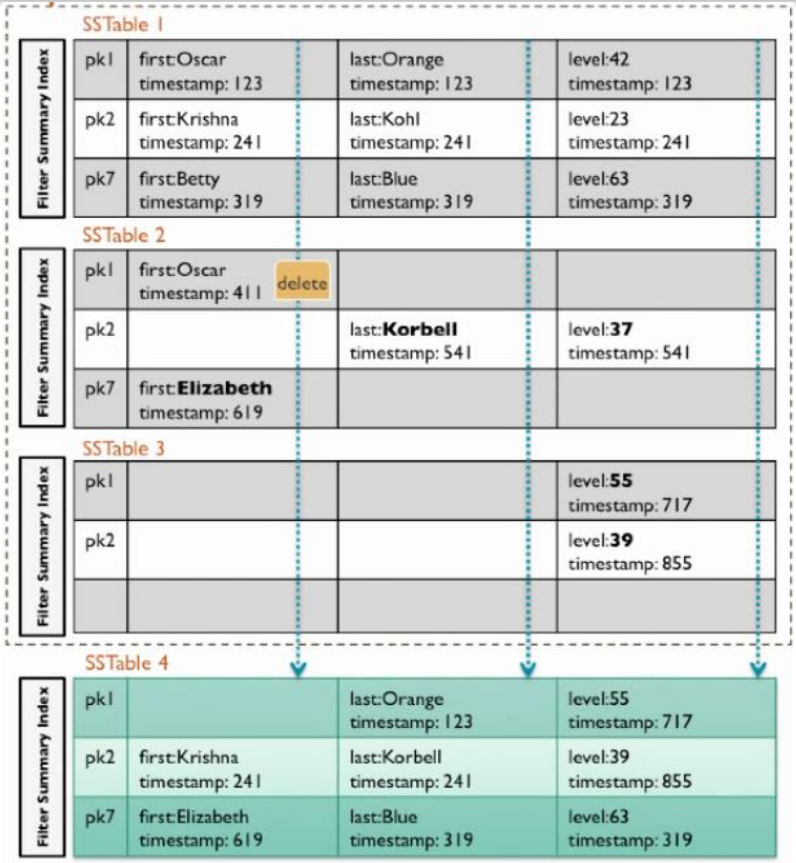
Cassandra write path



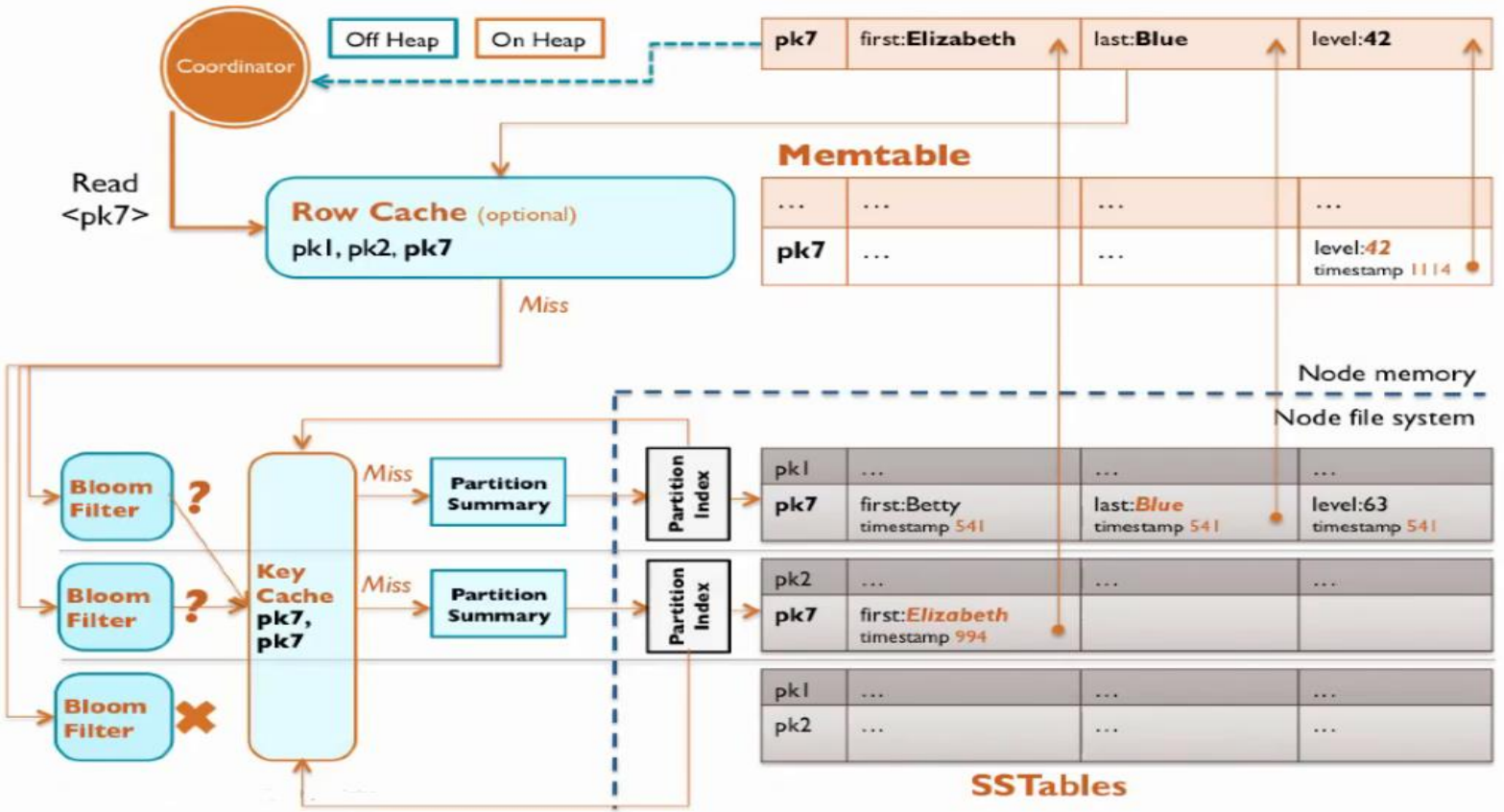
Cassandra write path (FLUSH)



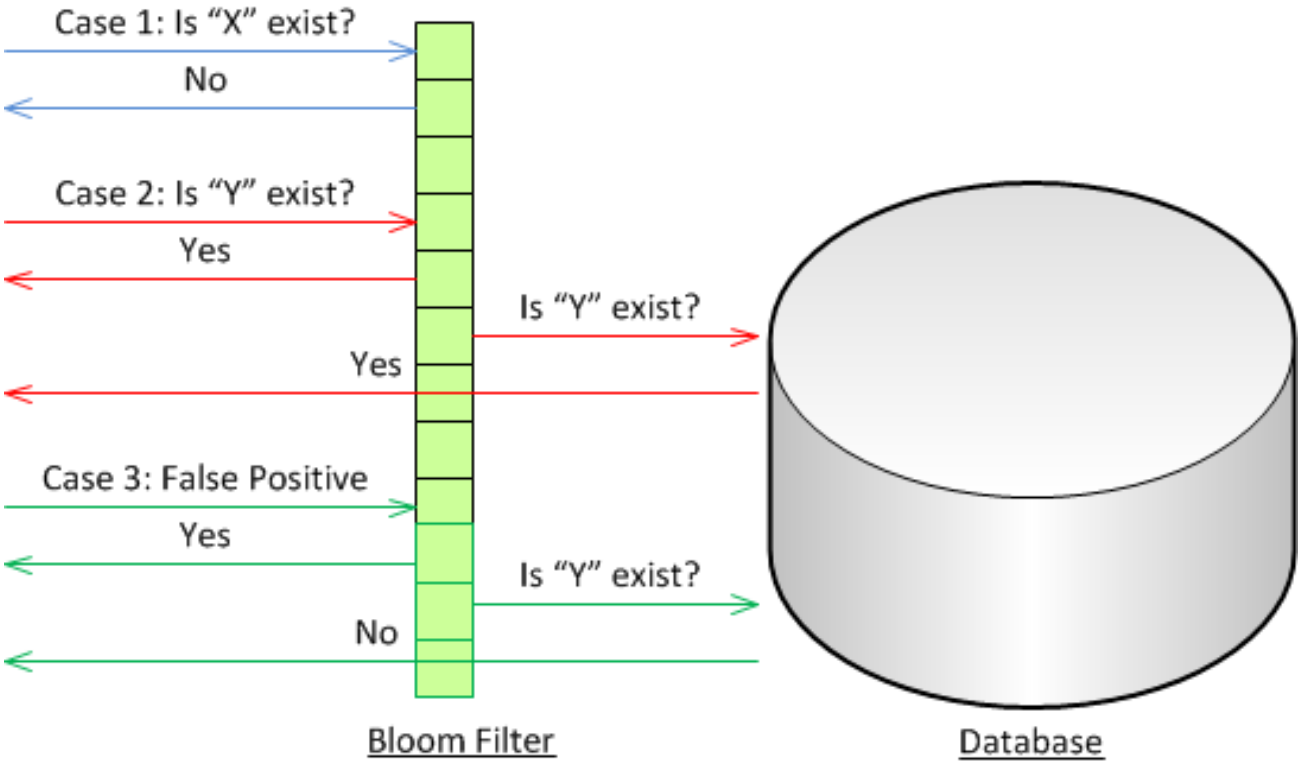
Cassandra write path (COMPACTION)



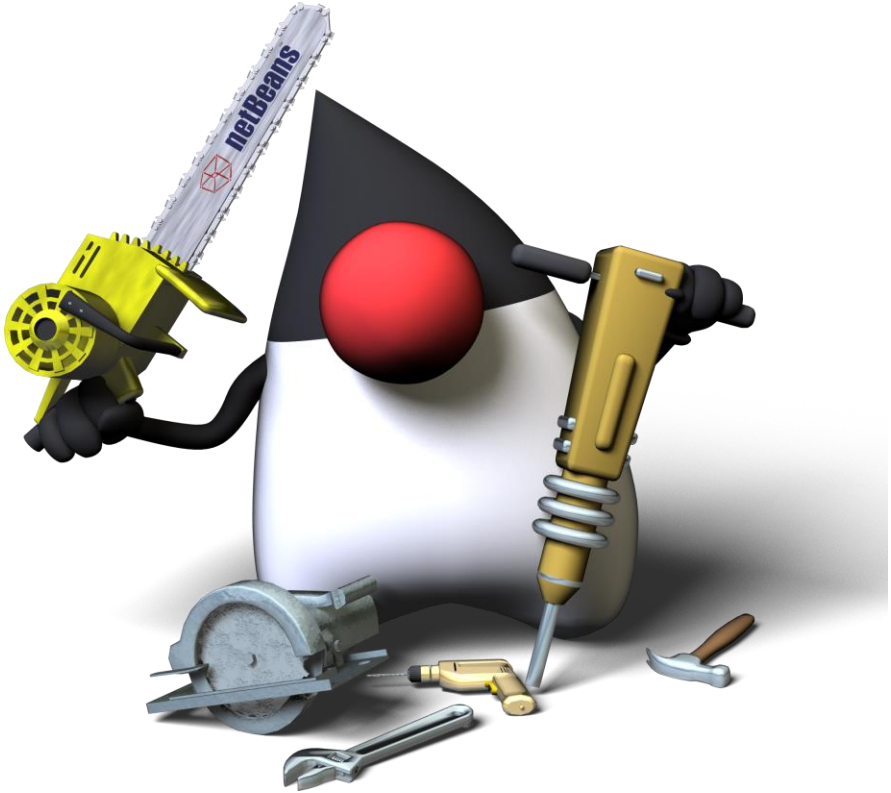
Cassandra READ path



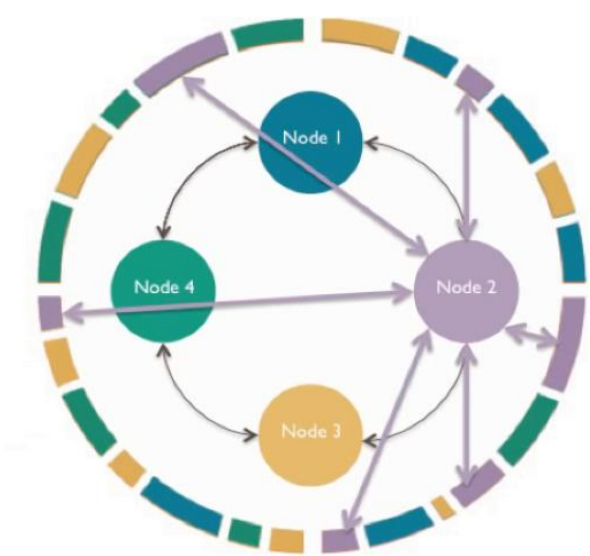
Bloom Filter

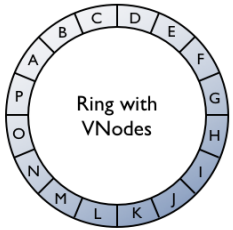


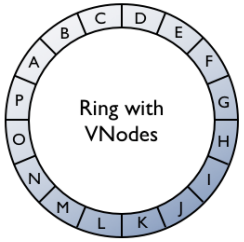
Case #6 : Strong Wall with Bloom Filters

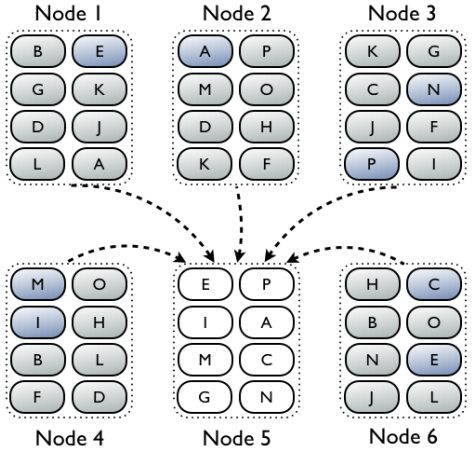


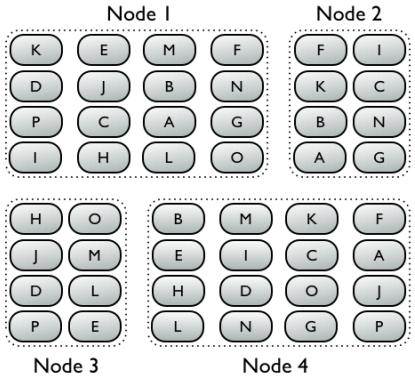
Cassandra Virtual nodes





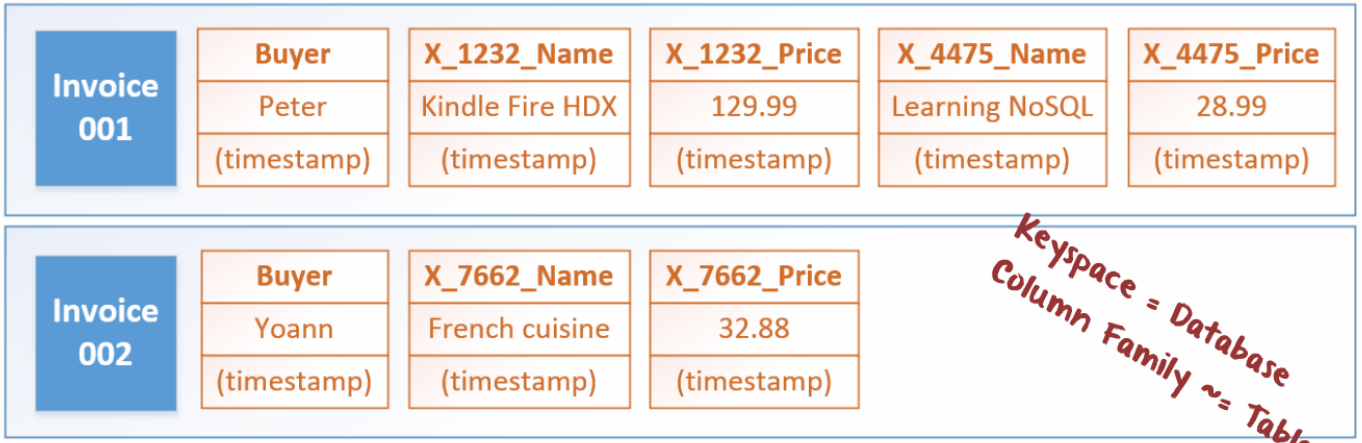






DATA MODEL

Data Model: Map<Map<Key, Value>>



Wide rows (2 billion cols)
Column names are data

```
set ColumnFamily['rowkey']['column'] = 'value';

get ColumnFamily['rowkey'];
get ColumnFamily['rowkey']['column'];

list ColumnFamily;

del ColumnFamily['rowkey'];
del ColumnFamily['rowkey']['column'];
```

KeySpace

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

CREATE KEYSPACE "Excalibur" WITH REPLICATION ={'class' :
'NetworkTopologyStrategy', 'dc1' : 3, 'dc2' : 2};
```

Tables

```
CREATE TABLE users
( id int PRIMARY KEY, first_name text,
last_name text, age int, hobbies list<test>);


CREATE TABLE users
( id int, first_name text, last_name text,
age int, hobbies list<test>, PRIMARY KEY(id));
```

Cassandra CQL (KEYS)

No

- NO joins
- NO subqueries
- NO Group By functionality
- NO aggregation functions

Use

- Data denormalization
- Data duplication
- PK, Indexes
- Counters,
- CAS Transactions, Batches
- Advanced Data Structures

Keys Types

- Simple primary key `PRIMARY KEY (empID)` — partition key
- Compound primary key `PRIMARY KEY (empID, subdeptID, depID)` — partition key, clustering key
- Composite primary key `PRIMARY KEY ((block_id, breed), color)` — partition key, clustering key

User-defined types

```
CREATE TYPE address (
   street text,
   city text,
   zip_code int,
   phones set<text> );
```

User-defined types

```
CREATE   INDEX state_key ON users (state);


SELECT * FROM users WHERE gender = 'f' AND state = 'TX'
ALLOW FILTERING;
```

Tables

```
CREATE TABLE users (
  user_id text PRIMARY KEY,
  first_name text, last_name text,
  emails set<text> );
INSERT INTO users (user_id, first_name, last_name, emails)
VALUES ('frodo', 'Frodo', 'Baggins',
  {'f@baggins.com', 'baggins@gmail.com'});
```

TTL

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

Ordering

```
CREATE TABLE timeseries (
  event_type text,
  insertion_time timestamp,
PRIMARY KEY (event_type, insertion_time))
WITH CLUSTERING ORDER BY (insertion_time DESC);
SELECT * FROM emp
WHERE empID IN (130,104) ORDER BY deptID DESC;
```

Slicing

```
//To retrieve events for the 12th of January 2014 between
3:50:00 and 4:37:30:

SELECT * FROM timeline WHERE day='12 Jan 2014' AND (hour,
min) >= (3, 50) AND (hour, min, sec) <= (4, 37, 30);
```

Batching

```
BEGIN BATCH

INSERT INTO purchases (user, balance) VALUES ('user1', -8)
IF NOT EXISTS;

INSERT INTO purchases (user, expense_id, amount,
description, paid) VALUES ('user1', 1, 8, 'bur', false);

APPLY BATCH;
```

Counters

```
CREATE TABLE counters.page_view_counts (counter_value
counter, url_name varchar, page_name varchar, PRIMARY KEY
(url_name, page_name) );


UPDATE counters.page_view_counts SET counter_value =
counter_value + 1 WHERE url_name='www.datastax.com' AND
page_name='home';
```

Lightweight Transactions

```
INSERT INTO users (login, email, name, login_count) VALUES
('jdoe', 'jdoe@abc.com', 'Jane Doe', 1) IF NOT EXISTS;

UPDATE users SET email = ‘janedoe@abc.com’ WHERE login =
'jdoe' IF email = ‘jdoe@abc.com’;
```

DataStax Java Driver

- DataStax developed a new protocol that doesn't have RPC limitations (Asynchronous I/O)
- Low-level API with simple mapping
- Works with CQL3
- QueryBuilder reminds CriteriaAPI
- Accessor-annotated interfaces

Async Query

```
public class AsynchronousExample extends SimpleClient {
    public AsynchronousExample() {
    }

    public ResultSetFuture getRows() {
        Query query = QueryBuilder.select().all().from("simplex", "songs");
        return getSession().executeAsync(query);
    }

    public static void main(String[] args) {
        AsynchronousExample client = new AsynchronousExample();
        client.connect("127.0.0.1");
        client.createSchema();
        client.loadData();
        ResultSetFuture results = client.getRows();
        for (Row row : results.getUninterruptibly()) {
            System.out.printf( "%s: %s / %s\n",
                    row.getString("artist"),
                    row.getString("title"),
                    row.getString("album") );

        }
        client.dropSchema("simplex");
        client.close();
    }
}
```

Accessor

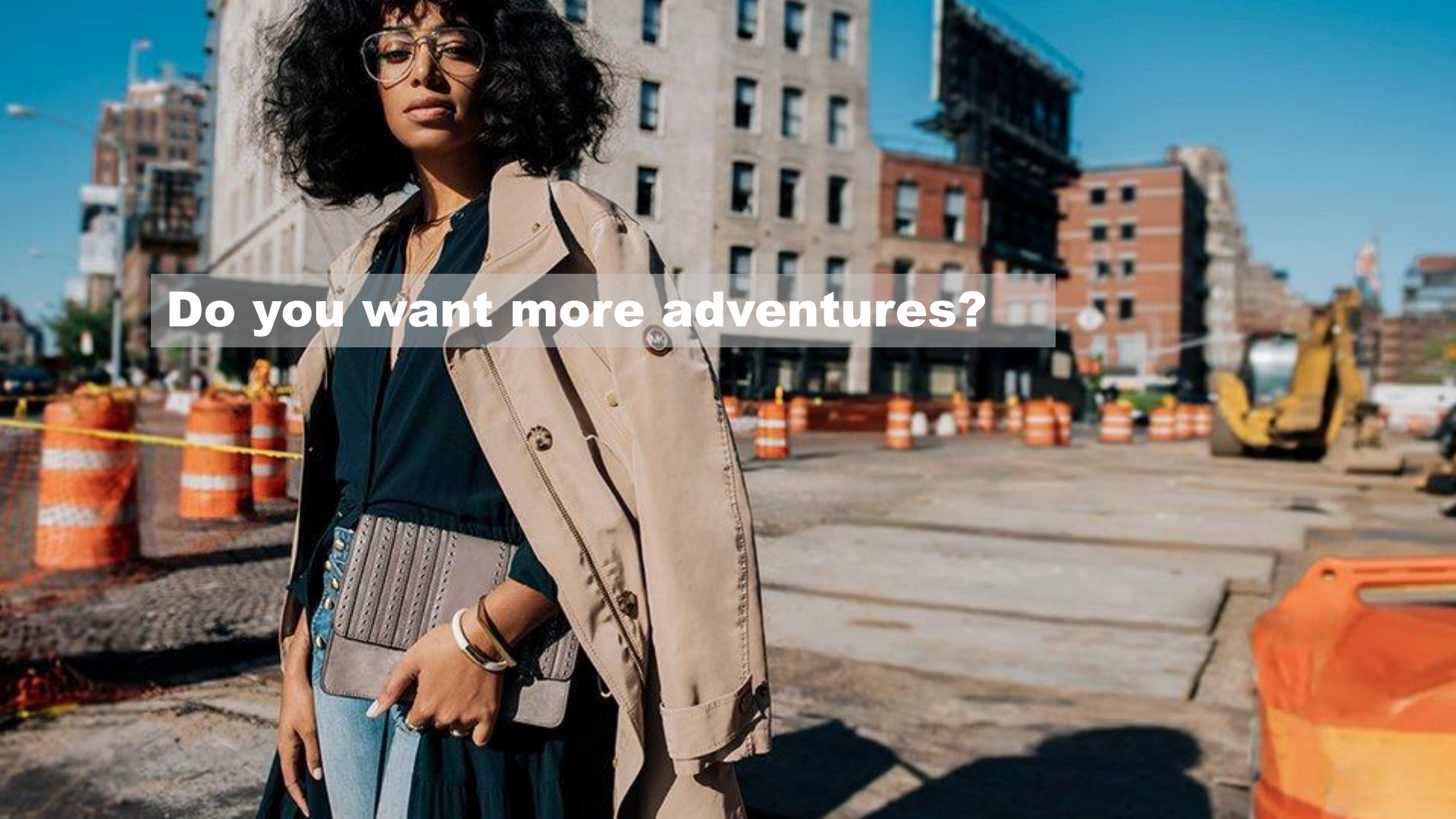
```
@Accessor
public interface UserAccessor {
    @Query("SELECT * FROM complex.users WHERE id = :id")
    User getUserNamed(@Param("userId") UUID id);

    @Query("SELECT * FROM complex.users WHERE id = ?")
    User getOnePosition(UUID userId);

    @Query("UPDATE complex.users SET addresses[:name]=:a
    ResultSet addAddress(@Param("id") UUID id, @Param("

    @Query("SELECT * FROM complex.users")
    public Result<User> getAll();

    @Query("SELECT * FROM complex.users")
    public ListenableFuture<Result<User>> getAllAsyn
```

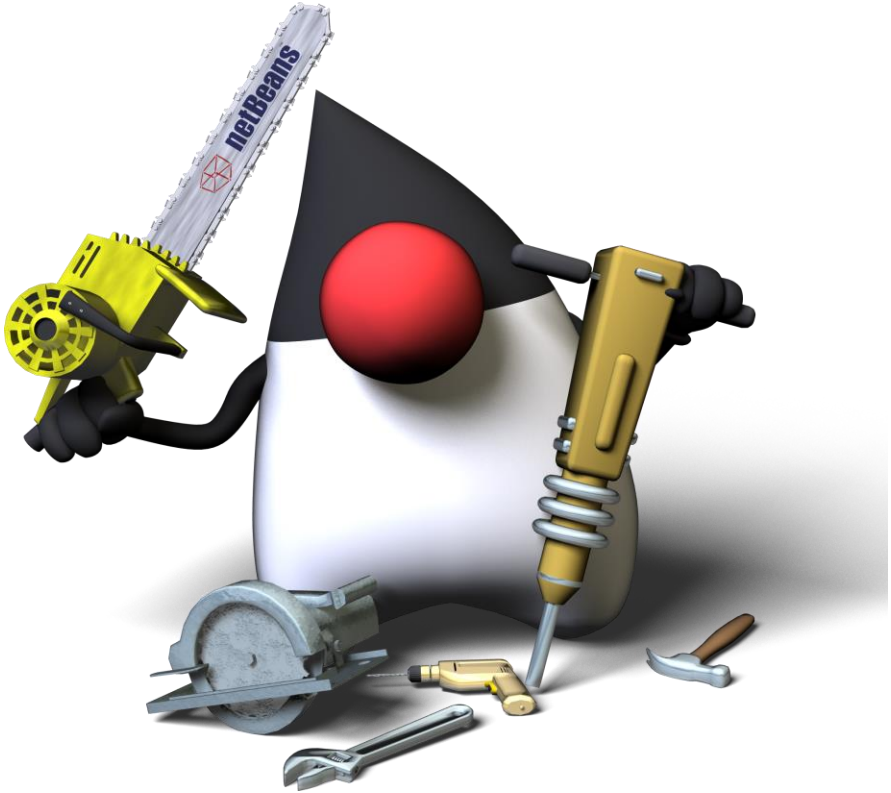

Do you want more adventures?

Other Cassandra's OM

- Achilles : well documented and provides transactions
- Astyanax : connection pool, thread safety and pagination
- Pelops : old project, good bycicle
- PlayORM : strange but powerful thing
- Easy-Cassandra : simple annotations + CRUD
- Thrift as low level API

MONGO WORLD

Case #7 : Customer wants MongoDB



Mongo Driver

```
DBCollection table = db.getCollection("user");

BasicDBObject document = new BasicDBObject();
document.put("name", "alex");
document.put("age", 27);
document.put("createdDate", new Date());
table.insert(document);


BasicDBObject query = new BasicDBObject();
query.put("name", "alex");

BasicDBObject newDocument = new BasicDBObject();
newDocument.put("name", "alex-updated");

BasicDBObject updateObj = new BasicDBObject();
updateObj.put("$set", newDocument);

table.update(query, updateObj);
```


Do you have an ORM framework?

Morphia Entity

```
@Entity("employees")
class Employee {

  @Id ObjectId id;

  Address address;

  Key<Employee> manager;

  @Reference List<Employee> underlings = new ArrayList<

  @Property("left") Date endDate;

  //fields can be indexed for better performance
  @Indexed boolean active = false;

  //fields can loaded, but not saved
  @NotSaved String readButNotStored;

  //Lifecycle methods -- Pre/PostLoad, Pre/PostPersist.
  @PostLoad void postLoad(DBObject dbObj) { ... }
```

Morphia Entity

```
@EntityListeners(BackAccountWatcher.class)
Public class BankAccount {
   @Id String id;
   Date lastUpdated = new Date();
}

class BankAccountWatcher{

   @PrePersist void prePersist(BankAccount act) {act.lastUpdated

}
```

.. or

```
class BankAccount {
   @Id String id;
   Date lastUpdated = new Date();

   @PrePersist void prePersist() {lastUpdated = new Date();}
}
```

Other Mongo's OM

- Jongo : mongo - shell queries in Java-code
- EclipseLink : different support of different NoSQL databases
- MJORM : Google Code, XML mapping + MQL (SQL syntax for Mongo data extracting)
- DataNucleus : support many Js as JDO, JPA

Modern Web App

Polyglot Persistance

- Redis: Rapid access for reads and writes. No need to be durable
- RBDMS: Needs transactional updates and has tabular structure.
- Riak: Needs high availability across multiple locations. Can merge inconsistent writes

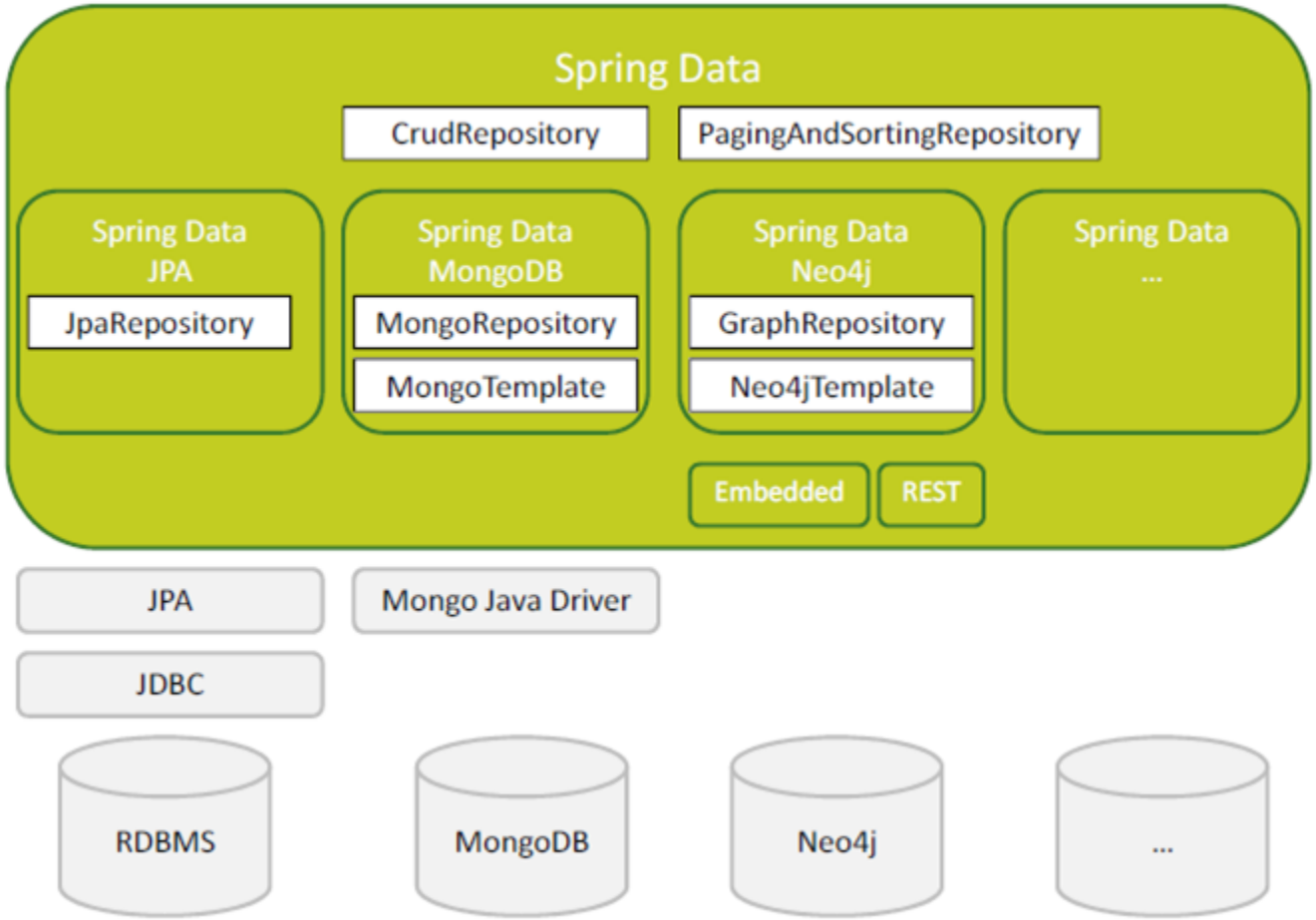
Polyglot Persistance

- Neo4j: Rapidly traverse links between friends and ratings.
- MongoDB: Lots of reads, infrequent writes. Powerful aggregation mechanism.
- Cassandra: Large-scale analytics on large cluster. High volume of writes on multiple nodes

<epam>

SPRING INTEGRATION

Spring Data


Spring Data
CrudRepository
PagingAndSortingRepository
Spring Data JPA
JpaRepository
Spring Data MongoDB
MongoRepository
MongoTemplate
Spring Data Neo4j
GraphRepository
Neo4jTemplate
Spring Data ...
Embedded
REST
JPA
Mongo Java Driver
JDBC
RDBMS
MongoDB
Neo4j
...

Spring Data MongoDB

- Templating : connection configs, collection lifecycle (create, drop), Map/Reduce + Aggregation
- Mapping: @Document, @Index, @Field
- Repository support: geospatial queries, queries derived from method signatures (at runtime)
- Paging, sorting, CRUD operations

Spring & MongoDB

```
public class Order {
    @Id private String id;
    private Date date;
    @Field("custInfo") private String customerInfo;
    List<Item> items;
}

public class Item {
    private int quantity;
    private double price;
    @Field("desc") private String description;
}

public interface OrderRepository extends MongoRepository

List<Order> findByItemsQuantity(int quantity);

List<Order> findByItemsPriceGreaterThan(double price);

}
```

Demo with JavaDriver 3.3

Best Practice: SQL for NoSQL

Any questions?