Overview of NoSQL Databases & HBase

Agenda

- Why NoSQL
- Problems with RDBMS
- What is NoSQL
- CAP Theorem
- NoSQL break down
- HBase Overivew & Architecture

What's Wrong with the RDBMS?

Nothing!!!!!

Scalability problems

Vertical Scaling

More boxes in DB cluster

Turn off logging/journaling

Moving code out of DB to application

Employ Caching layer

De-normalization

What problem do I have?

Or

What are the problems that I can solve with the data I have?

Relational Database

Atomic

Consistency

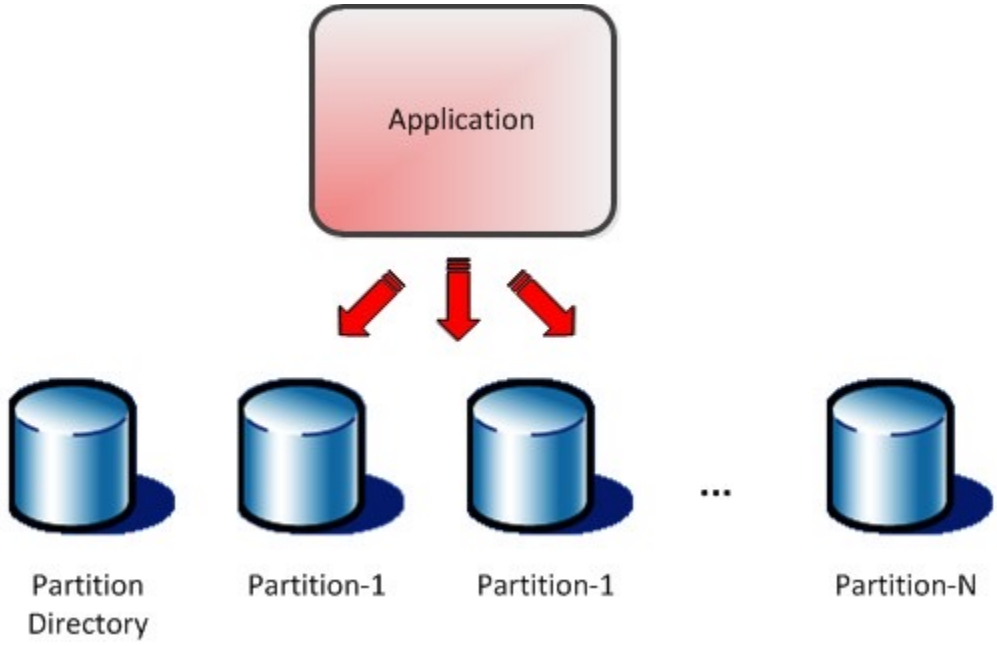
Isolation

Durability

Database Sharding – The "Shared-Nothing" Approach

If you can't split it, you can't scale it

Database Sharding provides a method for scalability across independent servers, each with their own CPU, memory and disk.



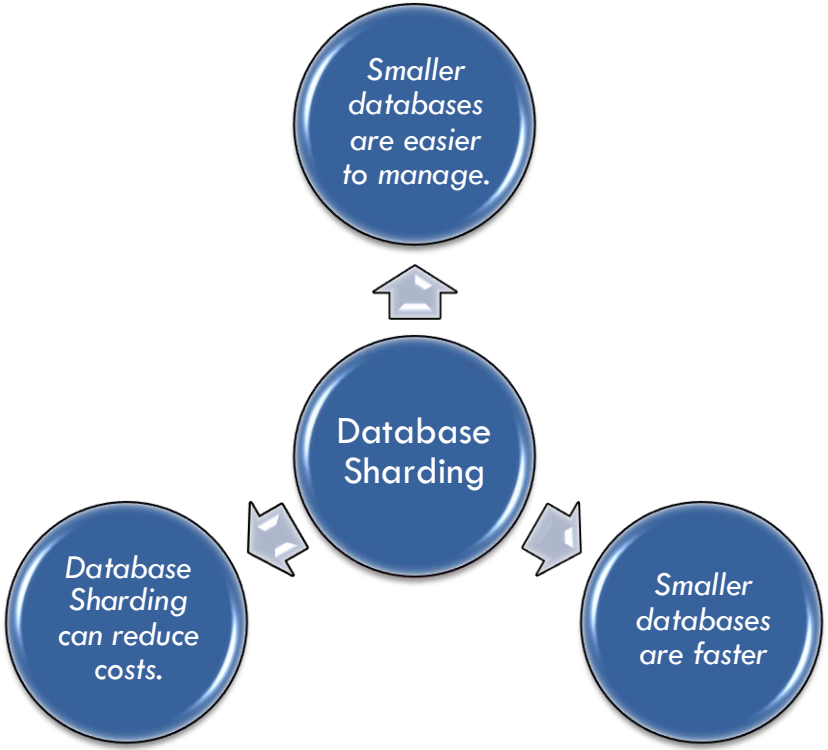
Sharding Strategies

Feature-based shard or functional segmentation

Key-based sharding

Lookup table

Database Sharding – Advantages



Enter NoSQL …

"Next Generation Databases mostly addressing some of the points: being **non-relational**, **distributed**, **open-source** and **horizontally scalable** , … **Schema-free**, **easy-replication support**, **simple API**, **eventually consistent**…"

- nosql-database.org

- Non-Relational
- Distributed
- Open-source
- Horizontally Scalable
- Schema-Free
- Replication Support
- Simple API
- Eventually Consistent

What is NoSQL

It's about using the right tool for the job

- Not all system have the same data needs.
- Sql is not the only option, nor it is always best one.
- Consider all options carefully and choose wisely

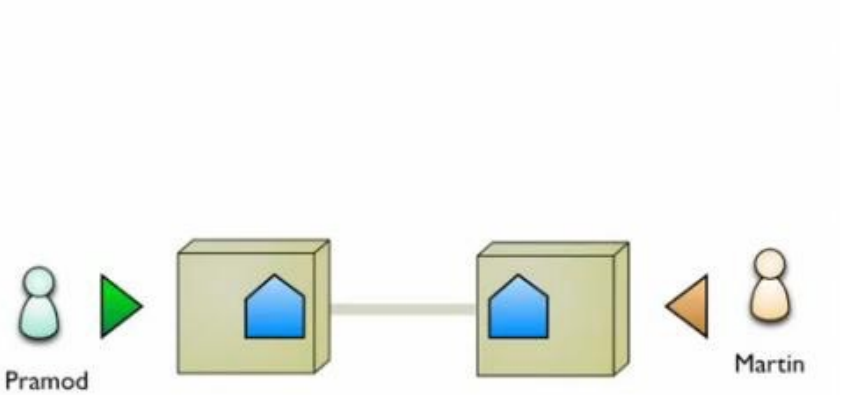
Not Only SQL

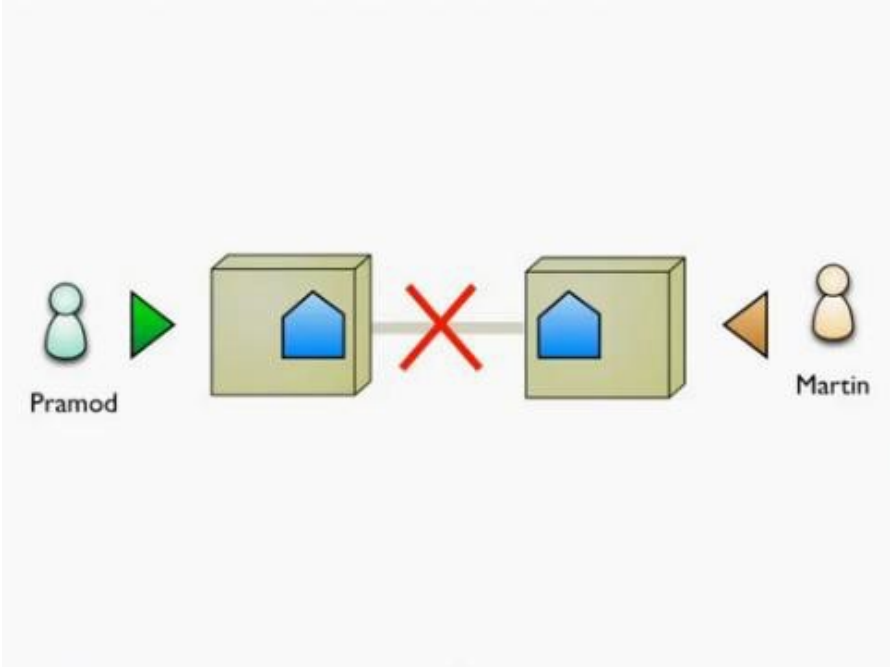
It's about opening our minds to new technologies

CAP Theorem

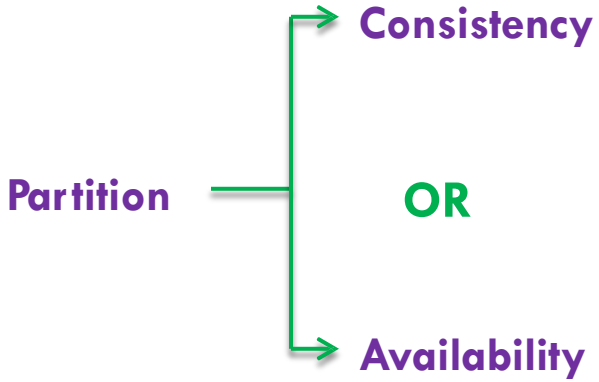
- Requirements to distributed systems
  - Consistency – the system is in a consistent state after an operation
    - All clients see the same data
    - Strong consistency (ACID) vs. eventual consistency(BASE)
  - Availability – the system is "always on", no downtime
    - Node failure tolerance – all clients can find some available replica
    - Software/hardware upgrade tolerance
  - Partition tolerance – the system continues to function even when split into disconnected subsets (by a network disruption), i.e. Tolerance to network partition.
    - Not only for reads, but writes as well!
- CAP Theorem (E. Brewer, N. Lynch)
  - You can satisfy **at most** 2 out of the 3 requirements.

Consistency or Availability?





Consistency or Availability?



CAP Theorem - Explained

- **CA**
  - Single site clusters (easier to ensure all nodes are always in contact)
  - e.g. 2PC
  - When a partition occurs, the system blocks
- **CP**
  - Some data may be inaccessible (availability sacrificed), but the rest is still consistent/accurate
  - e.g. sharded database
  - Requests can complete at nodes that have quorum

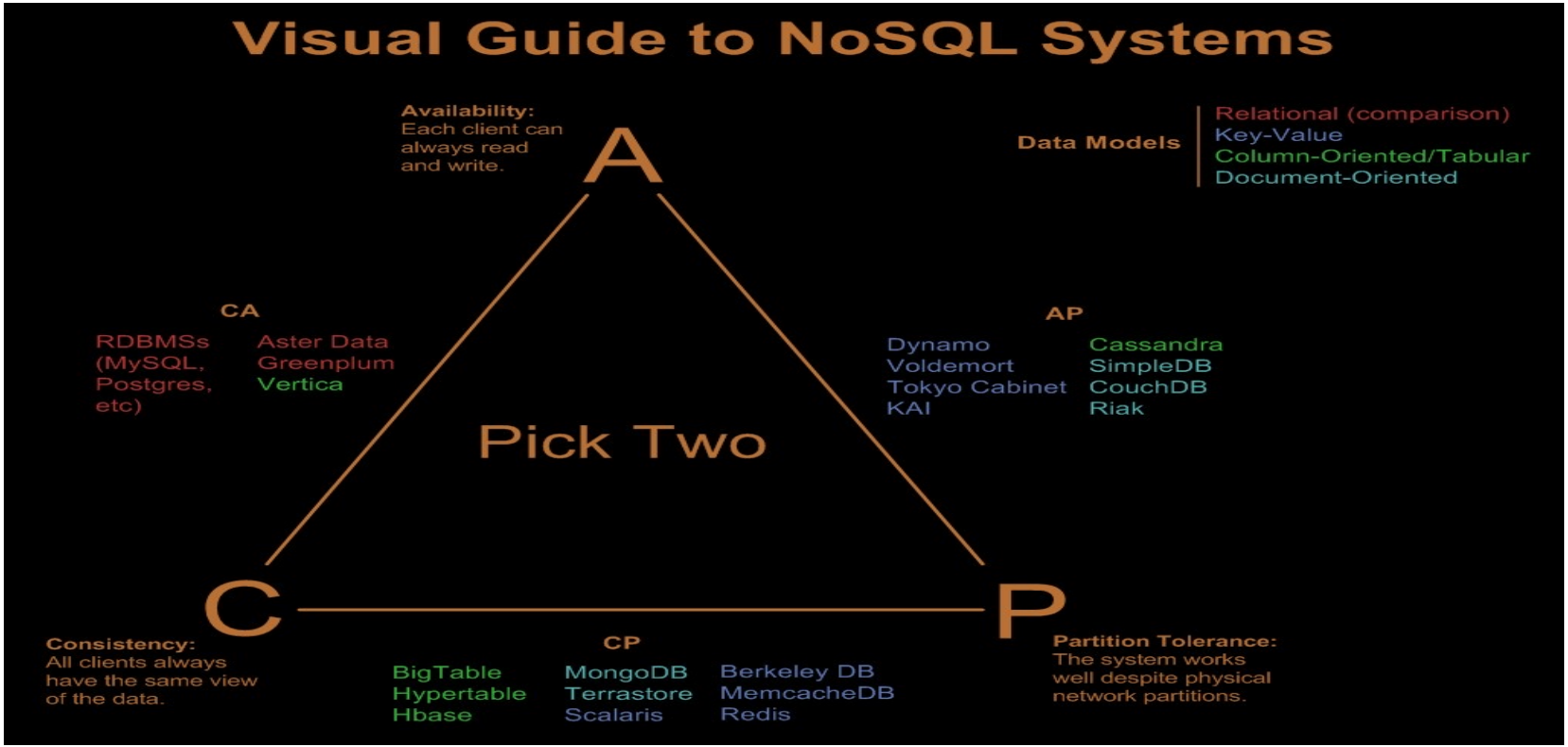
CAP Theorem - Explained

- **AP**
  - System is still available under partitioning, but some of the data returned may be inaccurate
  - e.g. Master/Slave replication
  - Need some conflict resolution strategy
  - Request can complete at any live node, possibly violating strong consistency.

Eventually Consistency for Availability

- BASE (basically available Soft state Eventually consistence)
  - Weak Consistency (stale data ok)
  - Availability first
  - Faster
  - Approximate answers ok
- ACID (Atomicity, Consistency, Isolation, Durability)
  - Strong consistency (NO stale data)
  - Isolation
  - Safer
  - Availability?

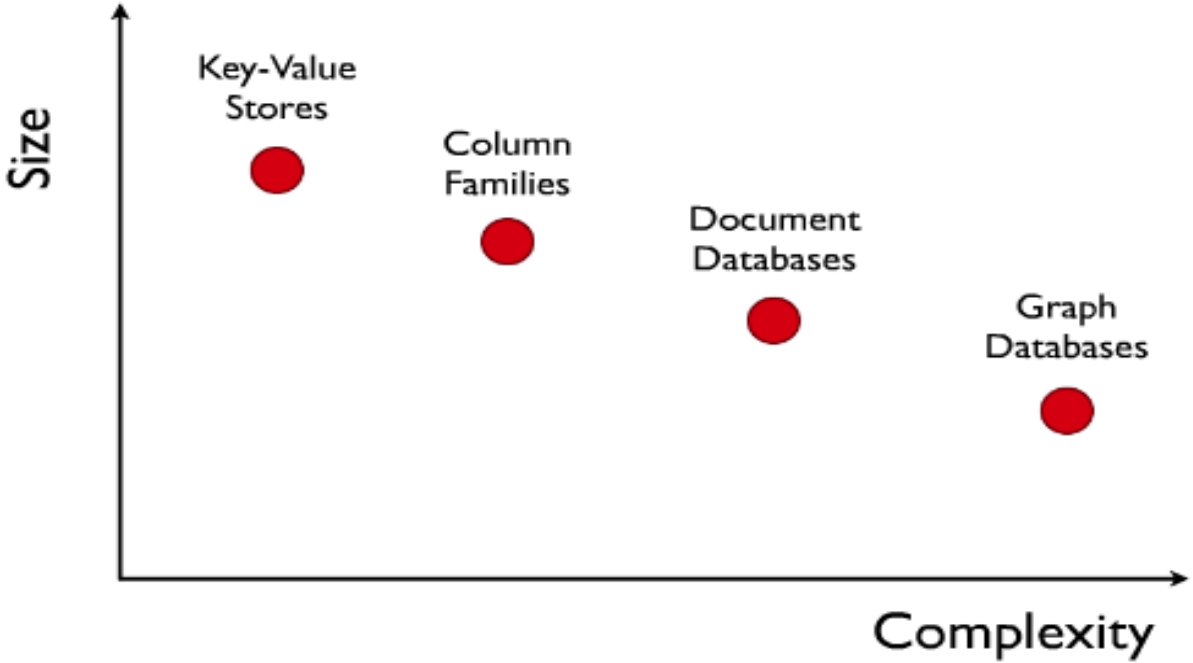
CAP Theorem


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.

NoSQL Break-down

- Key-Value stores
- Column Families
- Document-Oriented
- Graph Databases

Focus of Different Data Models




NOSQL
Key Value
In-memory
Memcached
Disk Based
Redis
Tokyo Cabinet
Dynamo
Voldemort
Column
SimpleDB
Google BigTable
HBase
Cassandra
HyperTable
Azure TS
Document
CouchDB
MongoDB
Lotus Domino
Riak
Graph
Neo4j
FlockDB
InfiniteGraph

When to use NoSQL

- **Bigness**
- **Massive write performance**
- **Fast key-value access**
- **Flexible schema and flexible datatypes**
- **Schema migration**
- **Write availability**
- **Easier maintainability, administration and operations**
- **No single point of failure**
- **Generally available parallel computing**
- **Use the right data model for the right problem**
- **Distributed systems support**
- **Tunable CAP tradeoffs**

Drawbacks of NoSQL

- **Data Integrity –** Developer need to take care.
- **SQL –** Still very few NoSQL systems provide a SQL interface
- **Ad-hoc queries –** If you need to answer the real time questions about your data that you can't predict in advance, RDBMS are generally winner here.
- **Complex relationships –** Some NoSQL systems supports relationships, but RDBMS is still winner in relating
- **Maturity and Stability –** RDBMS still have the edge here.

HBase

- **Strongly consistent** reads/writes: HBase is not an "eventually consistent" DataStore. This makes it very suitable for tasks such as high-speed counter aggregation.
- **Automatic sharding**: HBase tables are distributed on the cluster via regions, and regions are automatically split and re-distributed as your data grows.
- **Automatic RegionServer failover**
- **Hadoop/HDFS Integration**: HBase supports HDFS out of the box as its distributed file system.

HBase

- **MapReduce:** HBase supports massively parallelized processing via MapReduce for using HBase as both source and sink.
- **Java Client API:** HBase supports an easy to use Java API for programmatic access.
- **Thrift/REST API:** HBase also supports Thrift and REST for non-Java front-ends.
- **Block Cache and Bloom Filters:** HBase supports a Block Cache and Bloom Filters for high volume query optimization.

When to use HBase?

- HBase isn't suitable for every problem.
- When you need Random write or Random reads on HDFS data
- Write-dominated workloads : Examples like time-series databases, user analytics etc are very write-heavy
- Large-scale workloads, if data growing at 250MB/month. This is hard to manage this with a traditional RDBMS.
- HBase has been insufficient for analytics because its design center is to support simple operations such as create, read, update, and delete rather than other operations such as aggregation.