INTRODUCTION TO In-Memory DATABASES

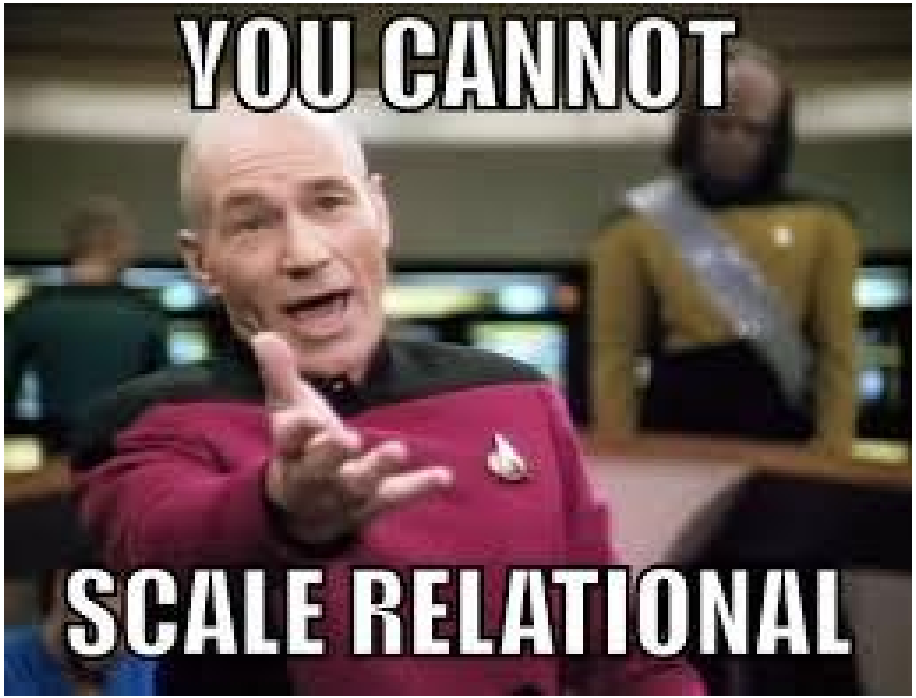
Outline

- Background
- What is NOSQL?
- Who is using it?
- CAP theorem
- NOSQL categories
- Redis
- Conclusion

Background

- Relational databases → mainstay of business
- Web-based applications caused spikes
  - explosion of social media sites (Facebook, Twitter) with large data needs
  - rise of cloud-based solutions such as Amazon S3 (simple storage solution)
- Hooking RDBMS to web-based application becomes trouble

Scaling out RDBMS (SQL)



Issues with ***scaling up***

- Best way to provide ACID and rich query model is to have the dataset on a single machine
- Limits to ***scaling up***

  (or ***vertical scaling*****:** make a “single” machine more powerful)

  → dataset is just too big!
- ***Scaling out***

  adding more smaller servers is a better choice
- Different approaches for horizontal scaling (multi-node database):
  - Master/Slave
  - Sharding (partitioning)

Scaling out RDBMS: Master/Slave

- Master/Slave
  - All writes are written to the master
  - All reads performed against the replicated slave databases
  - Critical reads may be incorrect as writes may not have been propagated down
  - Large datasets can pose problems as master needs to duplicate data to slaves

Scaling out RDBMS: Sharding

- Sharding (Partitioning)
  - Scales well for both reads and writes
  - Not transparent, application needs to be partition-aware
  - Can no longer have relationships/joins across partitions
  - Loss of referential integrity across shards

Other ways to scale out RDBMS

- Multi-Master replication
- INSERT only, not UPDATES/DELETES
- No JOINs, thereby reducing query time
  - This involves de-normalizing data
- In-memory databases

NOSQL


LOTS OF NOSQL
MONGO, CASSANDRA, HBASE, RIAK, REDIS, NEO4J
memegenerator.net

What is NOSQL?

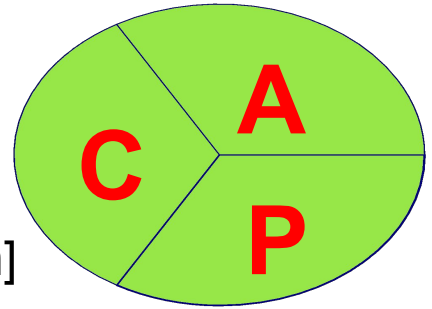
- Key features (advantages):
  - non-relational
  - SQL is not built for large number of small boxes
  - don’t require schema
  - data are replicated to multiple nofast key-value accessdes (so, identical & fault-tolerant) and can be partitioned:
    - down nodes easily replaced
    - no single point of failure
  - horizontal scalable
  - cheap, easy to implement (open-source)
  - massive write performance

What is NOSQL?

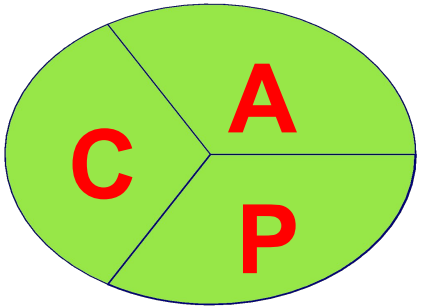
- Disadvantages:
  - Don’t fully support relational features
    - no join, group by, order by operations (except within partitions)
    - no referential integrity constraints across partitions
  - No declarative query language (e.g., SQL) → more programming
  - Relaxed ACID (see CAP theorem) → fewer guarantees

CAP Theorem

- Suppose three properties of a distributed system (sharing data)
  - **Consistency:**
    - all copies have same value[Logical, Replication]
  - **Availability:**
    - reads and writes always succeed
  - **Partition-tolerance:**
    - system properties (consistency and/or availability) hold even when network failures prevent some machines from communicating with others



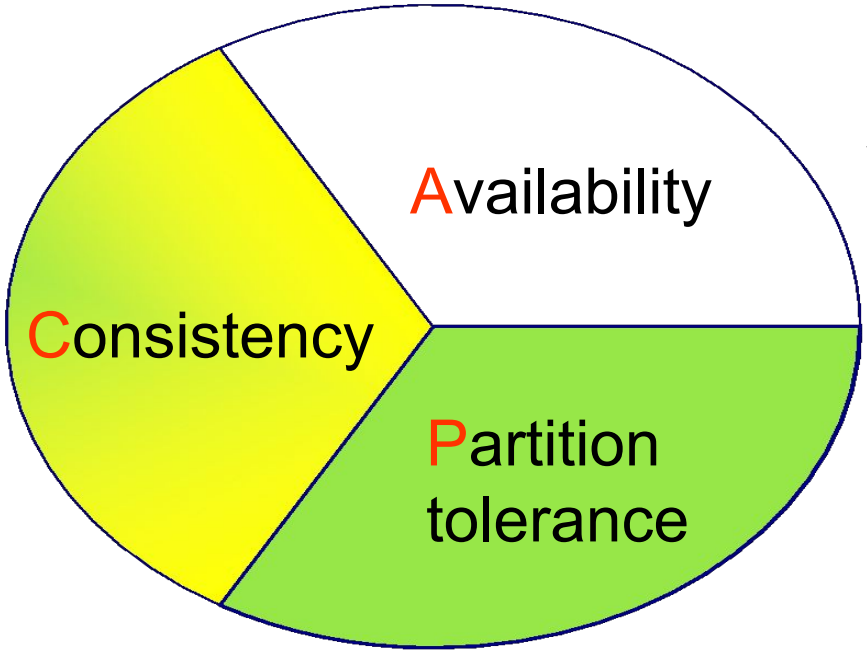
CAP Theorem



CAP Theorem

- **Brewer's CAP Theorem:**
  - *For any system sharing data, it is "impossible" to guarantee simultaneously all of these three properties*
  - You can have at most two of these three properties for any shared-data system
- Very large systems will "partition" at some point:
  - That leaves either **C** or **A** to choose from (traditional DBMS prefers **C** over **A** and **P**)
  - In almost all cases, you would choose **A** over **C** (except in specific applications such as order processing)

CAP Theorem



All client always have the same view of the data

CAP Theorem

- **Consistency**
  - 2 types of consistency:
    1. Strong consistency – ACID (**A**tomicity, **C**onsistency, **I**solation, **D**urability)
    2. Weak consistency – BASE (**B**asically **A**vailable **S**oft-state **E**ventual consistency)

CAP Theorem

- **ACID**
  - A DBMS is expected to support “ACID transactions,” processes that are:
  - **A**tomicity: either the whole process is done or none is
  - **C**onsistency: only valid data are written
  - **I**solation: one operation at a time
  - **D**urability: once committed, it stays that way

- **CAP**
  - **C**onsistency: all data on cluster has the same copies
  - **A**vailability: cluster always accepts reads and writes
  - **P**artition tolerance: guaranteed properties are maintained even

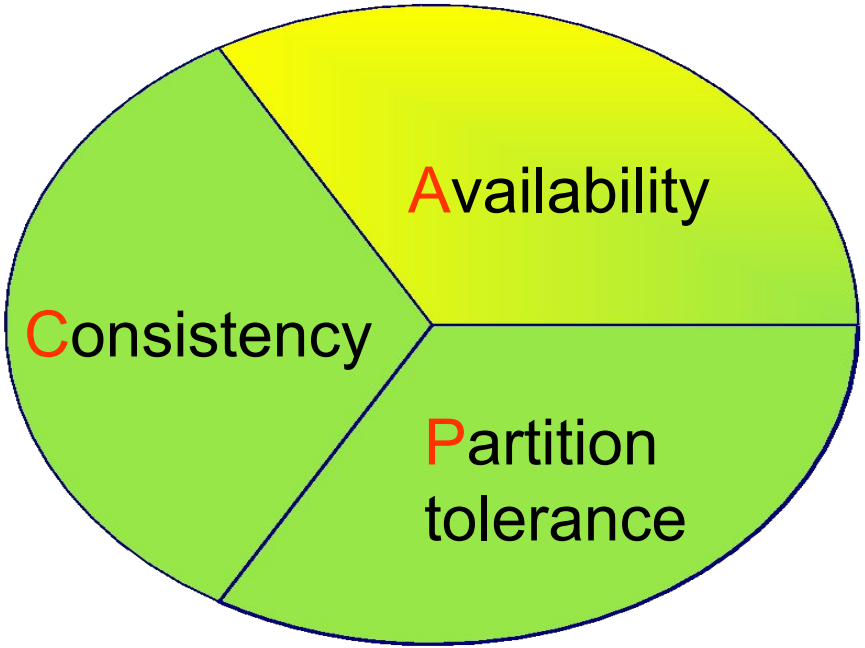
CAP Theorem

- A consistency model determines rules for visibility and apparent order of updates
- Example:
  - Row X is replicated on nodes M and N
  - Client A writes row X to node N
  - Some period of time t elapses
  - Client B reads row X from node M
  - **Does client B see the write from client A?**
  - Consistency is a continuum with tradeoffs
  - **For NOSQL, the answer would be: “maybe”**
  - CAP theorem states: *“strong consistency can't be achieved at the*

CAP Theorem

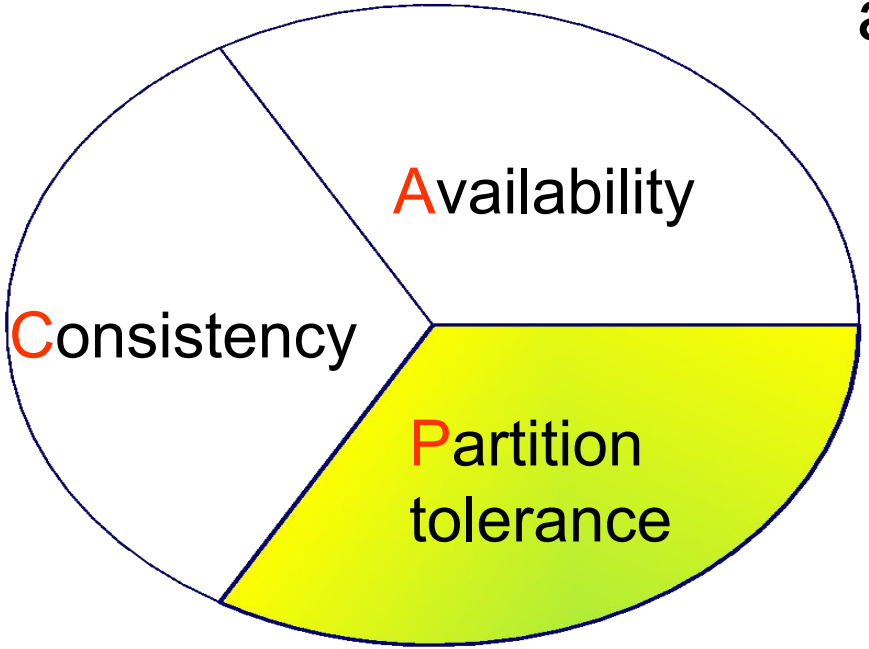
- Eventual consistency
  - When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
- Cloud computing
  - ACID is hard to achieve, moreover, it is not always required, e.g. for blogs, status updates, product listings, etc.

CAP Theorem



Each client always can read and write.

CAP Theorem



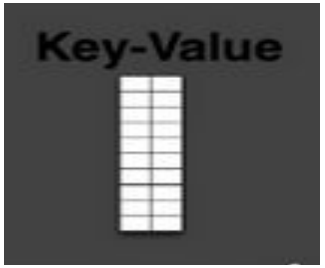
A system can continue to operate in the presence of a network partitions

NOSQL categories

1. Key-value
   - Example: DynamoDB, Voldermort, Scalaris
2. Document-based
   - Example: MongoDB, CouchDB
3. Column-based
   - Example: BigTable, Cassandra, Hbased
4. Graph-based
   - Example: Neo4J, InfoGrid

- “No-schema” is a common characteristics of most NOSQL storage systems
- Provide “flexible” data types

Key-value



- Focus on scaling to huge amounts of data
- Designed to handle massive load
- Based on Amazon's dynamo paper
- Data model: (global) collection of Key-value pairs
- *Dynamo ring partitioning* and *replication*
- Example: (DynamoDB)
  - *items* having one or more attributes (name, value)
  - An *attribute* can be single-valued or multi-valued like set.
  - items are combined into a *table*

Key-value

- Basic API access:
  - get(key): extract the value given a key
  - put(key, value): create or update the value given its key
  - delete(key): remove the key and its associated value
  - execute(key, operation, parameters): invoke an operation to the value (given its key) which is a special data structure (e.g. List, Set, Map .... etc)

Key-value

Pros:

- very fast
- very scalable (horizontally distributed to nodes based on key)
- simple data model
- eventual consistency
- fault-tolerance

Cons:

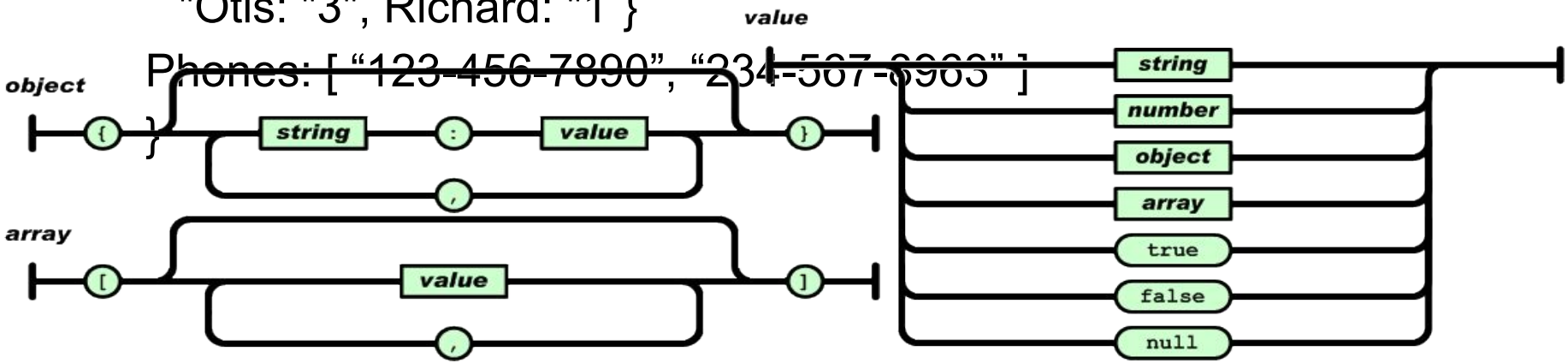
- Can’t model more complex data structure such as objects

Document-based

- Can model more complex objects
- Data model: collection of documents
- Document: JSON (**J**ava**S**cript **O**bject **N**otation is a data model, key-value pairs, which supports objects, records, structs, lists, array, maps, dates, Boolean with **nesting**), XML, other semi-structured formats.

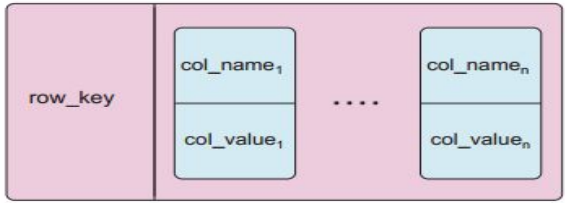
Document-based

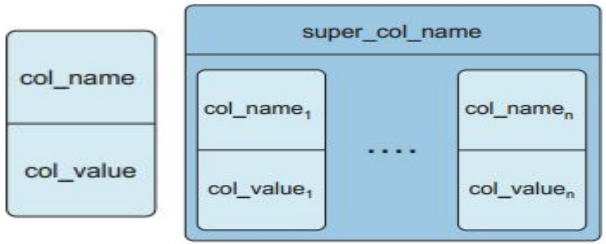
- Example: (MongoDB) document
  - {Name:"Jaroslav",
    Address:"Malostranske nám. 25, 118 00 Praha 1”,
    Grandchildren: {Claire: "7", Barbara: "6", "Magda: "3", "Kirsten: "1",
    "Otis: "3", Richard: "1“}
    Phones: [“123-456-7890”, “234-567-8963”]

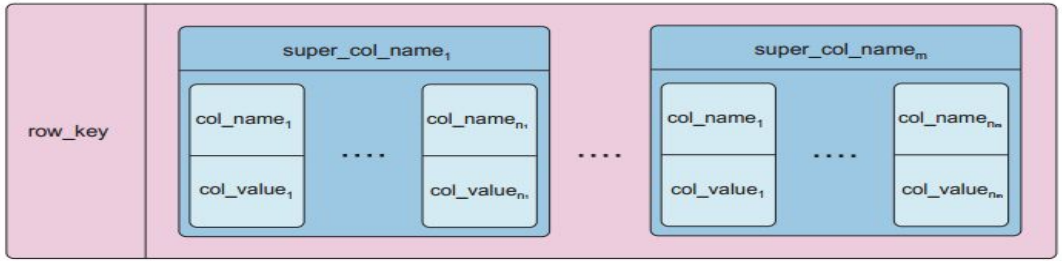


Column-based

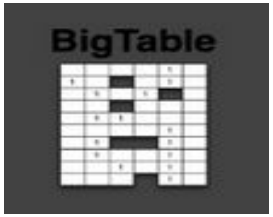
- Based on Google's BigTable paper
- Like column oriented relational databases (store data in column order) but with a twist
- Tables similarly to RDBMS, but handle semi-structured
- Data model:
  - Collection of Column Families
  - Column family = (key, value) where value = set of **related** colum
  - indexed by *row key*, *column key* and *timestamp*







Column-based



- One column family can have variable numbers of columns
- Cells within a column family are sorted “physically”
- Very sparse, most cells have null values
- **Comparison:** RDBMS vs column-based NOSQL
  - Query on multiple tables
    - **RDBMS:** must fetch data from several places on disk and glue together
    - **Column-based NOSQL:** only fetch column families of those columns that are required by a query (all columns in a column family are stored together on the disk, so multiple rows can be retrieved in one read operation → data locality)

Column-based

- Example: (Cassandra column family--timestamps removed for simplicity)

UserProfile = {

Cassandra = { emailAddress:”casandra@apache.org” , age:”20”}

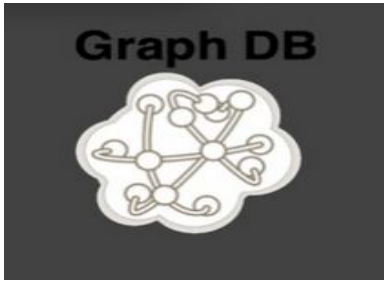
TerryCho = { emailAddress:”terry.cho@apache.org” , gender:”male”}

Cath = { emailAddress:”cath@apache.org” ,
age:”20”,gender:”female”,address:”Seoul”}

}

Graph-based

- Focus on modeling the structure of data (*interconnectivity*)
- Scales to the complexity of data
- Inspired by mathematical Graph Theory (G=(E,V))
- Data model:
  - (Property Graph) nodes and edges
    - Nodes may have properties (including ID)
    - Edges may have labels or roles
  - Key-value pairs on both
- Interfaces and query languages vary
- *Single-step* vs *path expressions* vs *full recursion*



Conclusion

- NOSQL database cover only a part of data-intensive cloud applications (mainly Web applications)
- Problems with cloud computing:
  - SaaS (**S**oftware **a**s **a** **S**ervice or on-demand software) applications require enterprise-level functionality, including ACID transactions, security, and other features associated with commercial RDBMS technology, i.e. NOSQL should not be the only option in the cloud
  - Hybrid solutions:
    - Redis with MySQL Write through Cache

Redis


WHAT IF I TOLD U
REDIS IS SOLUTION OF ALL PROBLEMS
imgflip.com

- NOSQL datastore
- Remote dictionary server
- In memory
- Supports wide variety of data structures
- High availability via Redis Sentinel and automatic partitioning with Redis Cluster (Redis 3.0, third party for < 3.0)
- Single threaded!!
- Simple text-based protocol
- We use it a lot (the kbb team has at least 2TB of redis*, around 500+ redis instances)

* we have boxes of that size dedicated for redis usage.

Installation

$sudo apt-get install redis-server

The ConfiG - `$cat /etc/redis.conf`

Port

Timeout

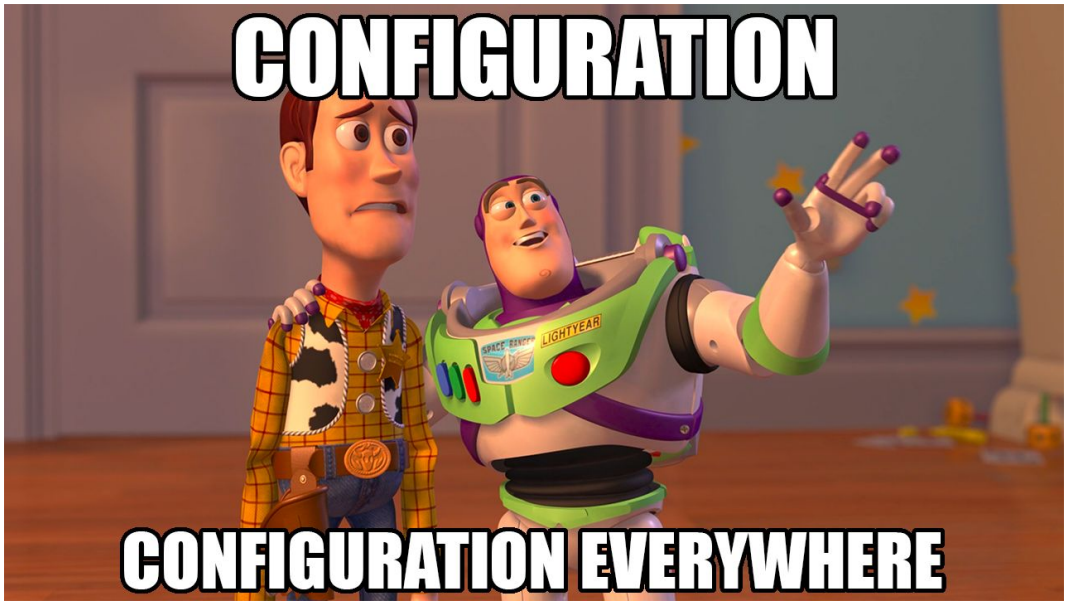
Save

Slaveof

Maxclients

Maxmemory

Maxmemory-policy



Data Structures

Strings

Hashes

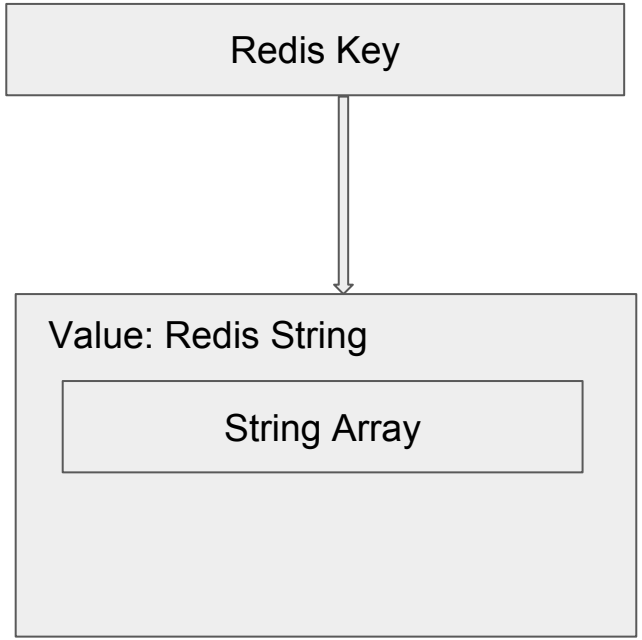
Lists

Sets

Sorted Sets

Logical Data Model

- Data Model
  - Key
    - Printable ASCII
  - Value
    - Primitives
      - Strings
    - Containers (of strings)
      - Hashes
      - Lists



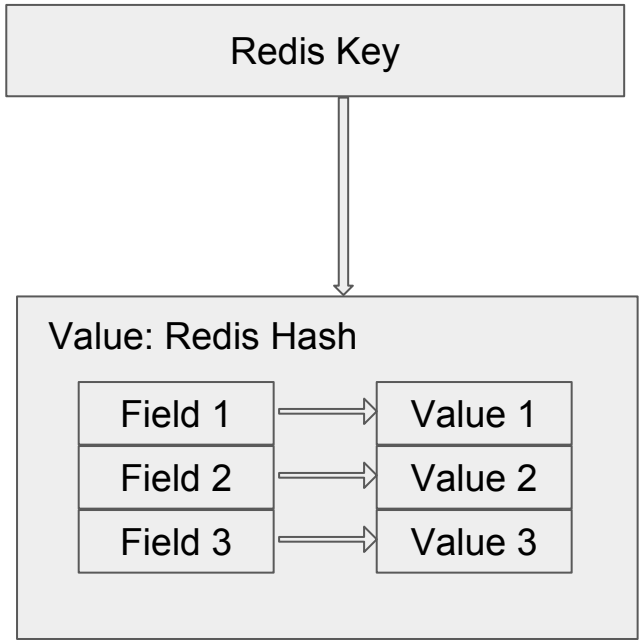
Logical Data Model - Hashes

>hmset user:1000 username antirez birthyear 1977 verified 1

>hget user:1000 username

>hget user:1000 birthyear

>hgetall user:1000



Logical Data Model - LIsts

Linked List

upside - Fixed time to add to head or tail

downside - Access is slower

> rpush mylist A

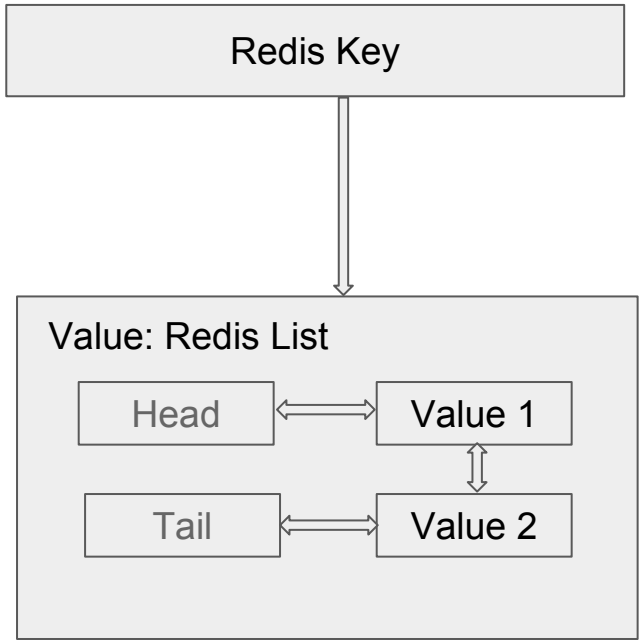
> rpush mylist B

> lpush mylist first

> lrange mylist 0 -1

> rpush mylist 1 2 3 4 5 "foo bar"

> rpop mylist



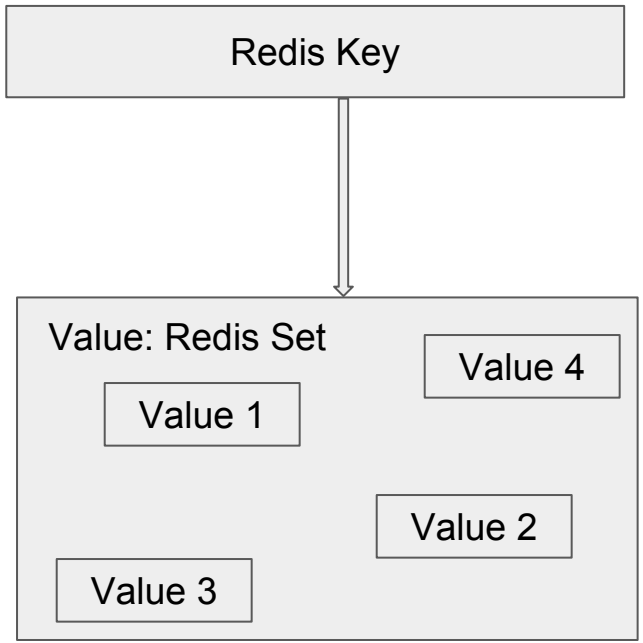
Logical Data Model - Sets

unordered collections of strings

> sadd myset 1 2 3

> smembers myset

> sismember myset 3



Logical Data Model - Sorted Sets

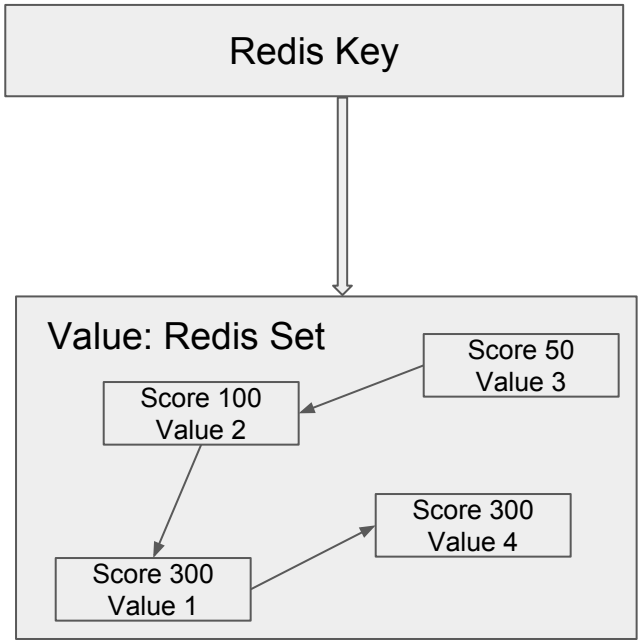
every member of a sorted set is associated with score

> ZADD myzset 1 "one"

> ZRANGE myzset 0 -1

> ZRANK myzset "one"

> ZRANGEBYSCORE myzset -inf +inf



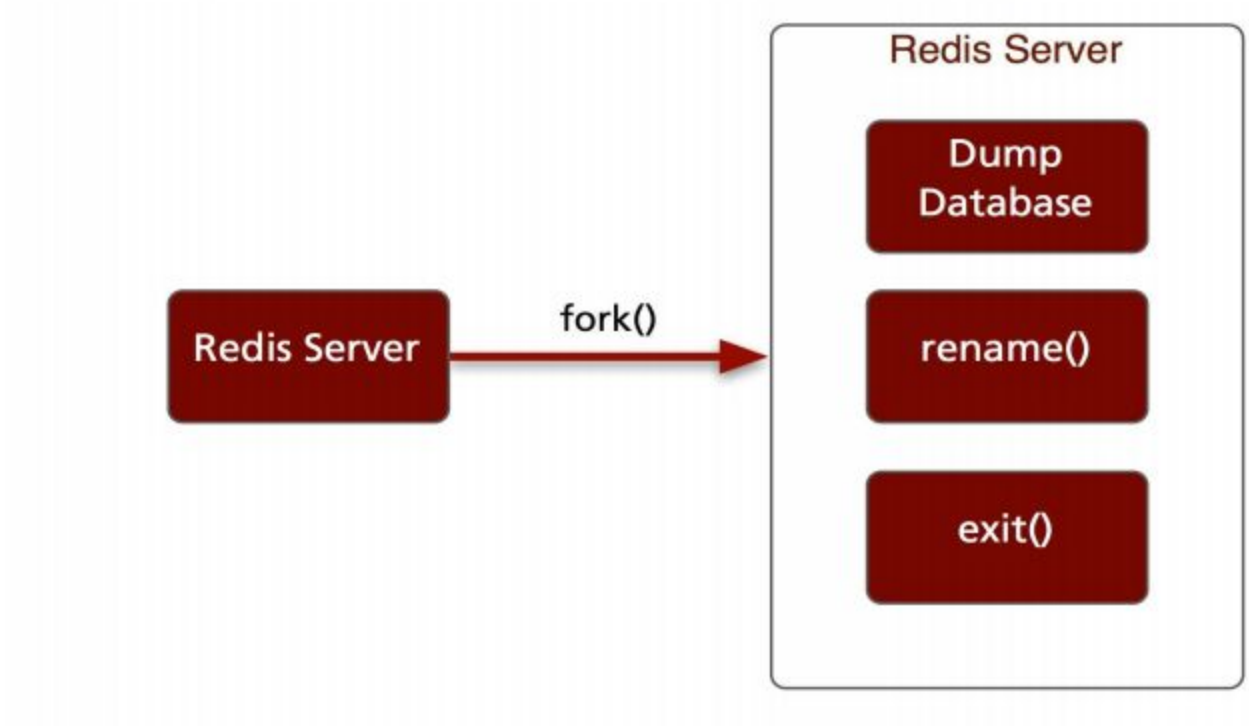
PipeLining

Data Persistence

RDB(Redis snapshotting) and AOF(Append-only file)

- The RDB persistence performs point-in-time snapshots of dataset at specified intervals.
- The AOF persistence logs every write operation received by the server, that will be played again at server startup, reconstructing the original dataset. Commands are logged using the same format as the Redis protocol itself, in an append-only fashion.

Data Persistence



Replication

1 Master - N Slaves

Asynchronous on Master

Synchronous on Slave

Like dumping the database, only to a different file descriptor.

Scaling - write to master, read from slaves

Replication

Replication Process Chronology

SLAVE: Connects to master, sends "SYNC" command

MASTER: Begins "background save" of DB to disk

MASTER: Begins recording all new writes

MASTER: Transfers DB snapshot to slave

SLAVE: Saves snapshot to disk

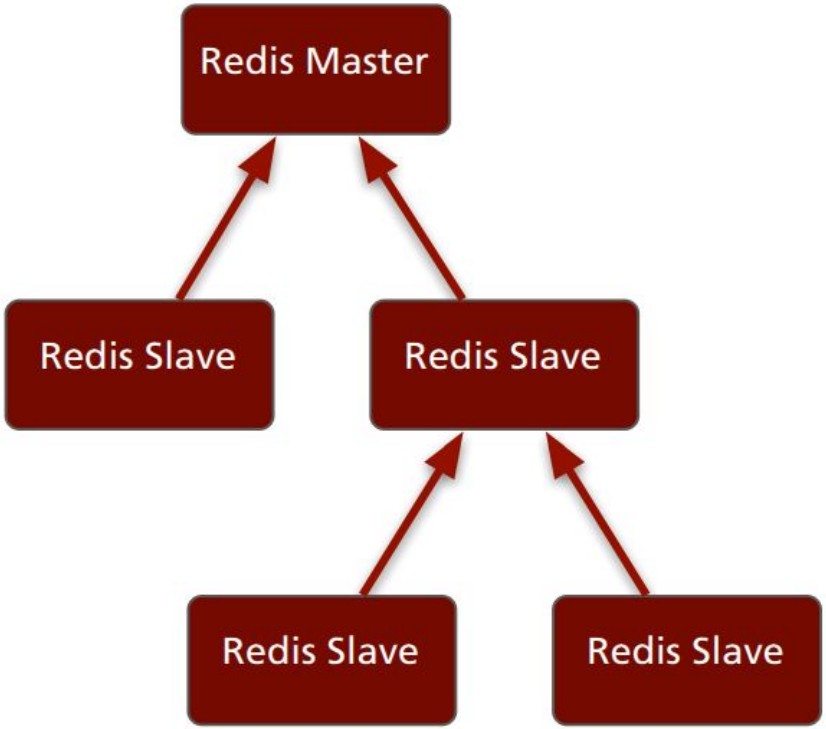
SLAVE: Loads snapshot into RAM

MASTER: Sends recorded write commands

SLAVE: Applies recorded write commands

SLAVE: Goes live, begins accepting requests

Replication



Scaling

Run multiple Redis processes

Sharding - client side or use 3rd party tools like nutcracker

Use Case

To redis:

- Caching
- Statistical data
- Recoverable state
- Worker queue

Not to redis:

- Data is larger than memory

Common Issues

Single threaded

Running monitor

Client-output-buffer limit

Expiry