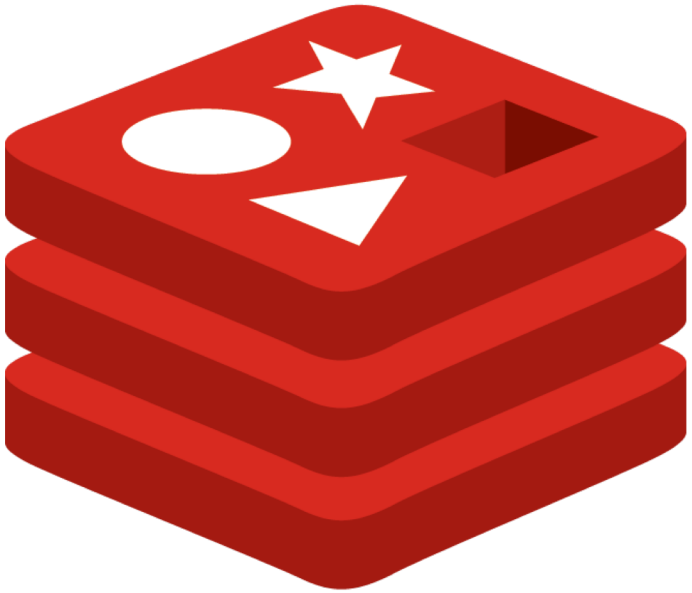


redis

History

- Early 2009 - Salvatore Sanfilippo, an Italian developer, started the Redis project
- He was working on a real-time web analytics solution and found that MySQL could not provide the necessary performance.
- June 2009 - Redis was deployed in production for the LLOOGG real-time web analytics website
- March 2010 - VMWare hired Sanfilippo to work full-time on Redis (remains BSD licensed)
- Subsequently, VMWare hired Pieter Noordhuis, a major Redis contributor, to assist on the project.

Redis 101

REDIS Stands for REmote DIctionary Server.

It's a Advanced In-memory Key-Value Store

Written in: C
Main point: Blazing fast
License: BSD
Protocol: Telnet-like, binary safe

Best used: For rapidly changing data with a foreseeable database size (should fit mostly in memory).

For example: Stock prices. Analytics. Real-time data collection. Real-time communication. And wherever you used memcached before.

Features

- **Advanced key-value store**
  - Think memcached on steroids (the good kind)
  - Values can be binary strings, Lists, Sets, Ordered Sets, Hash maps, ..
  - Operations for each data type, e.g. appending to a list, adding to a set, retrieving a slice of a list, …
  - Provides pub/sub-based messaging
- **Very fast:**
  - In-memory operations
  - ~100K operations/second on entry-level hardware
- **Persistent**
  - Periodic snapshots of memory OR append commands to log file
  - Limits are size of keys retained in memory.
- **Has "transactions"**
  - Commands can be batched and executed atomically

And it can scale

- **Master/slave replication**
  - Tree of Redis servers
  - Non-persistent master can replicate to a persistent slave
  - Use slaves for read-only queries
- **Sharding**
  - Client-side only – consistent hashing based on key
  - Server-side sharding – in near future
- **Run multiple servers per physical host**
  - Server is single threaded => Leverage multiple CPUs
- **Optional "virtual memory"**
  - Ideally data should fit in RAM
  - Values (not keys) written to disc

Docker

Docker info: https://docs.docker.com/engine/reference/run/

Get Docker image of Redis
docker pull redis

Start Docker container with Redis server:
docker run --rm --name redis-server1 -d -p 6379:6379 redis

some useful parameters:
-d: run container in the background and return the container ID
-rm: remove all container resources after stopping the container
-p: bind your container's port to a port on your local host

To verify it is running on the local mapped port, and accepts connections:
nc -vz localhost 6379

To view all running Docker containers:
docker ps -a

To stop the container:
docker stop <container_name>

Docker

To delete the container:
docker rm <container_name>

Get Docker image of ‘redis-cli’ - start as a separate container and connect to a running Redis server:
docker run -it --link redis-server1:redis --rm redis redis-cli -h redis -p 6379

To connect to an already running container which has cli
docker exec -it happy_thompson redis-cli -h redis -p 6379

To see all commands executed on the Redis server:
redis-cli> monitor

Clear all keys/state in the Redis server:
redis-cli> flushall

All Redis Commands: https://redis.io/commands

Installation ….Real Simple

Linux/IOS:

http://redis.io/download

```
$ wget http://download.redis.io/releases/redis-5.0.5.tar.gz
$ tar xzf redis-5.0.5.tar.gz
$ cd redis-5.0.5
$ make
```

OR with brew or npm.

Windows

1. Clone from Git repo: https://github.com/MSOpenTech/redis
2. Unzip file from **/redis/bin/release** (e.g. **redisbin64.zip**) to **/redis**
3. Important files:
   /redis/redis-server.exe
   /redis/redis-cli.exe

Redis – Config File

- **Configuration file: `/redis/redis.conf`**
- **It is possible to change a port (if you wish):**

```
port 6379
```

- **For development environment it is useful to change data persisting policy**

```
save 900 1
save 300 10
save 60 10000
```

```
save 10 1
```

save after 10 sec if at least 1 key changed

If installed with brew on mac path is below
/usr/local/etc/redis.conf

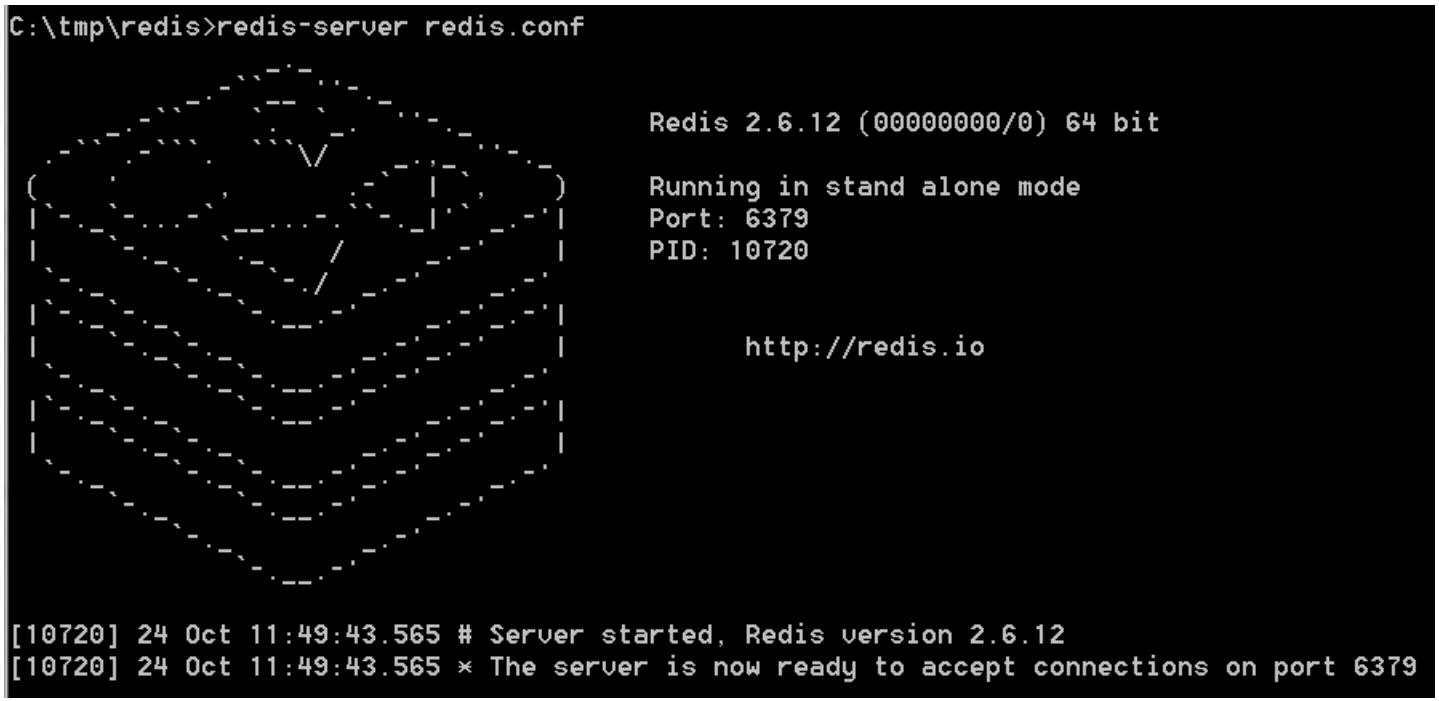
Start Redis as mac starts
ln -sfv /usr/local/opt/redis/*.plist ~/Library/LaunchAgents

Stop Redis as mac starts
launchctl unload ~/Library/LaunchAgents/homebrew.mxcl.redis.plist

Getting Started

Windows : Run **`/redis/bin/redis-server.exe`** and specify configuration file to use

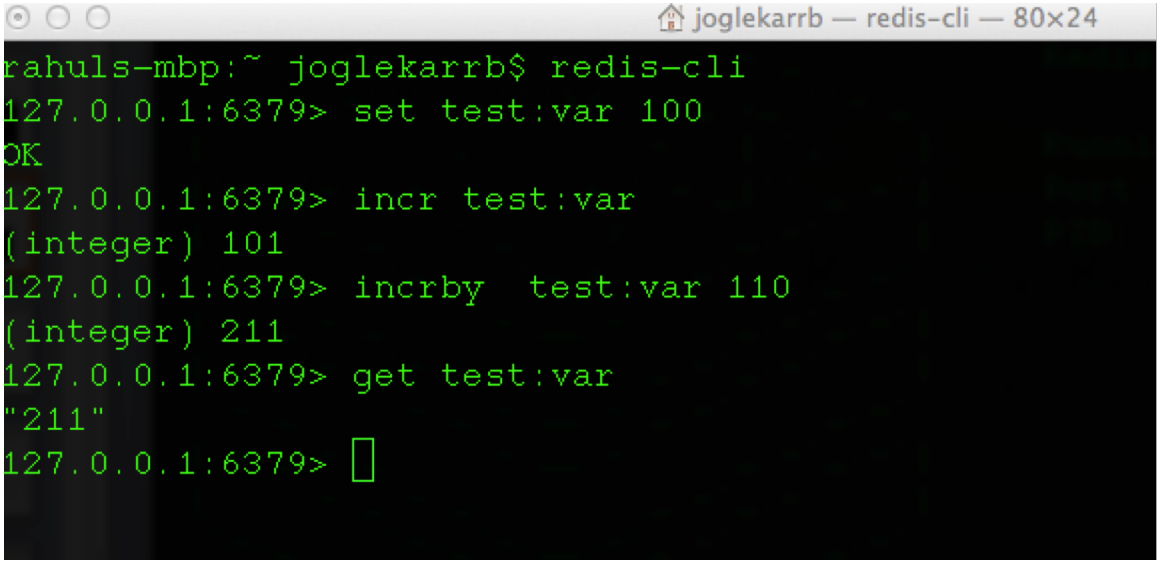
```
redis>redis-server redis.conf
```



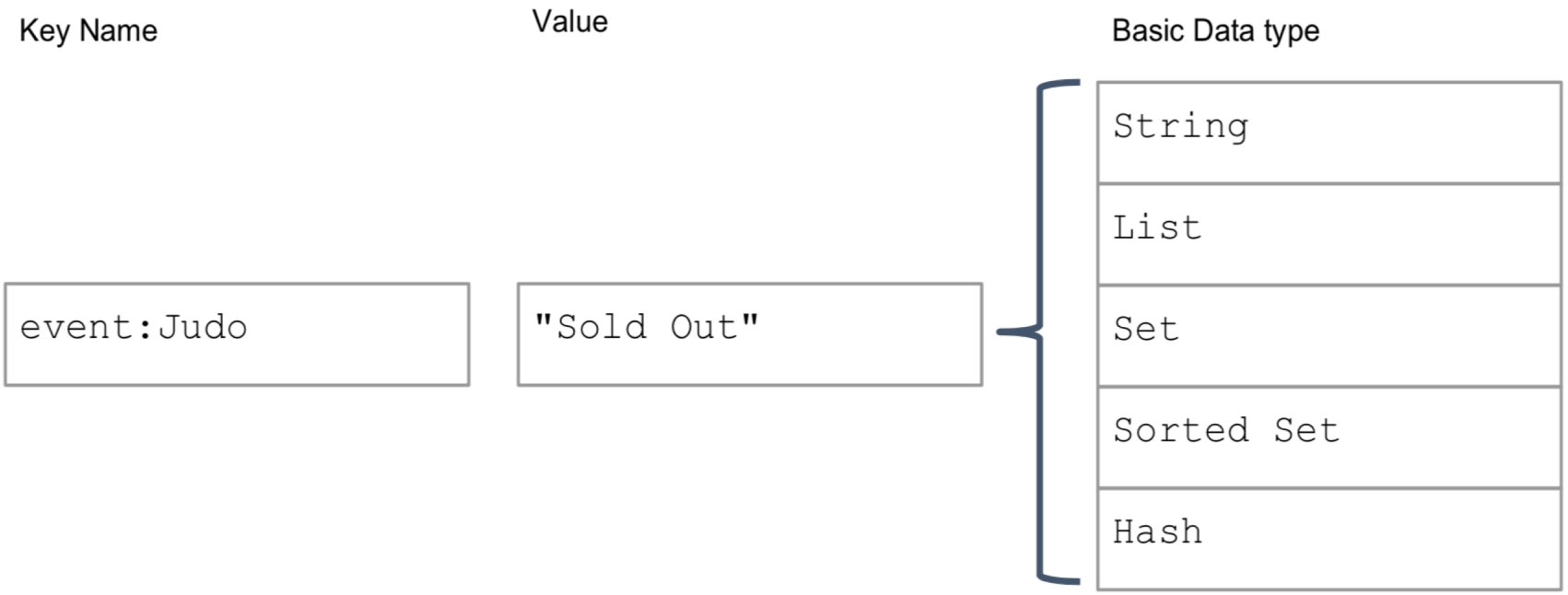
Mac/Linux : Make sure your PATH variable is correct and just type redis-server , it starts on port 6379

$ redis-server

Mac/Linux : Start Client
$ redis-cli



REDIS - Keys Values and Datatypes



Redis-Keys and Values

- **Keys**
  - Keys are binary safe - it is possible to use any binary sequence as a key
  - The empty string is also a valid key
  - Too long keys are not a good idea
  - Too short keys are often also not a good idea ("u:1000:pwd" versus "user:1000:password”)
  - Recommended approach is to use some kind of schema, like: "object-type:id:field"

- **Redis is often referred to as a data structure server since values can contain:**
  - Strings
  - Lists
  - Sets
  - Hashes
  - Sorted Sets

Key Naming

Key Name

event:Judo
event:venues

event:Fencing
event:fencing
Event:fencing

Key Name Expiration

```
> set event:Fencing ex 50
```

event:Fencing	50

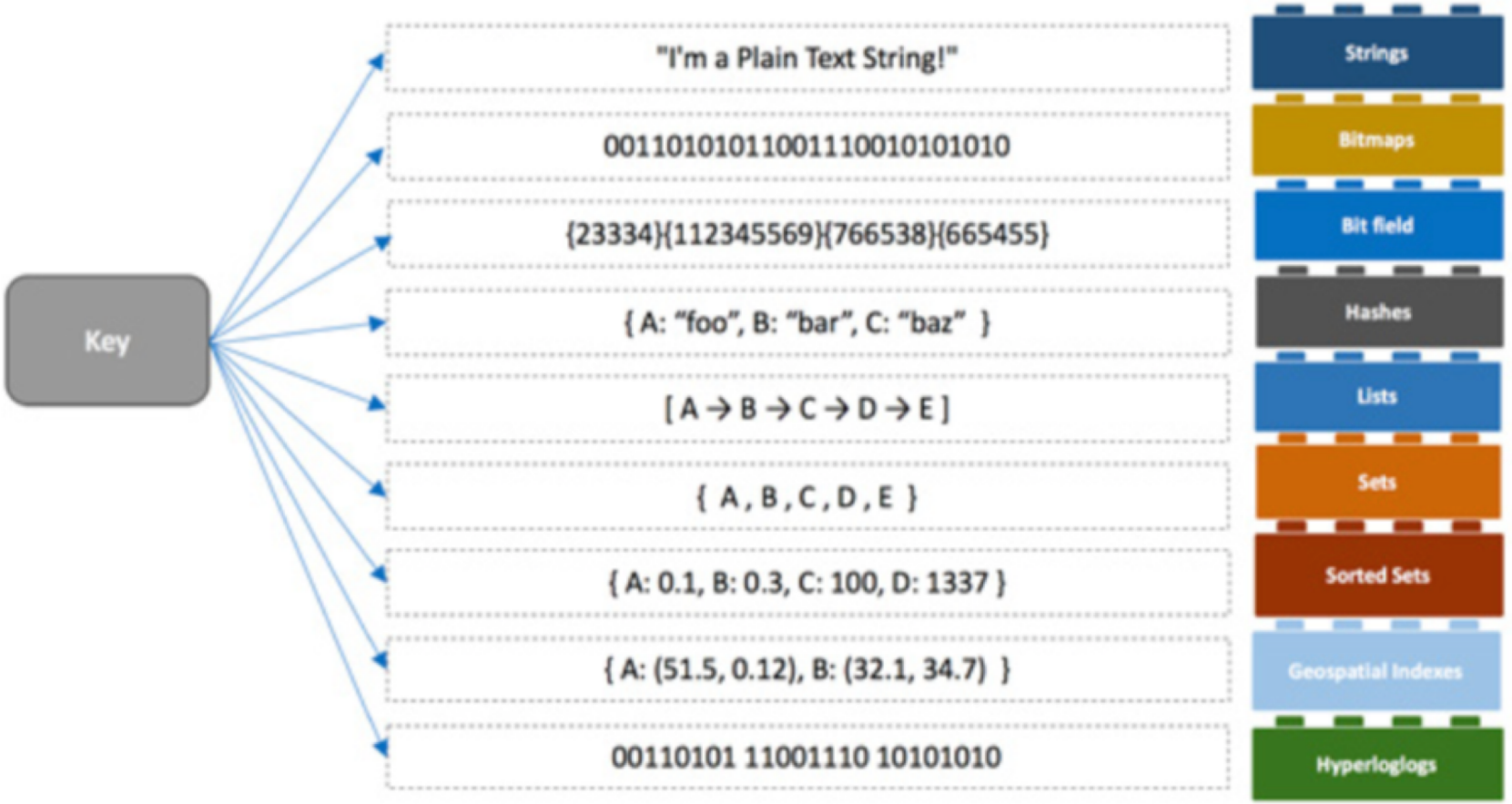
Naming

- Up to you
- Be consistent
- Colon is a typical community convention
  - Domain Object Name
  - Colon
  - Unique ID
- Plurals are nice for Lists, Sets
- Case-sensitive

Expiration or TTL

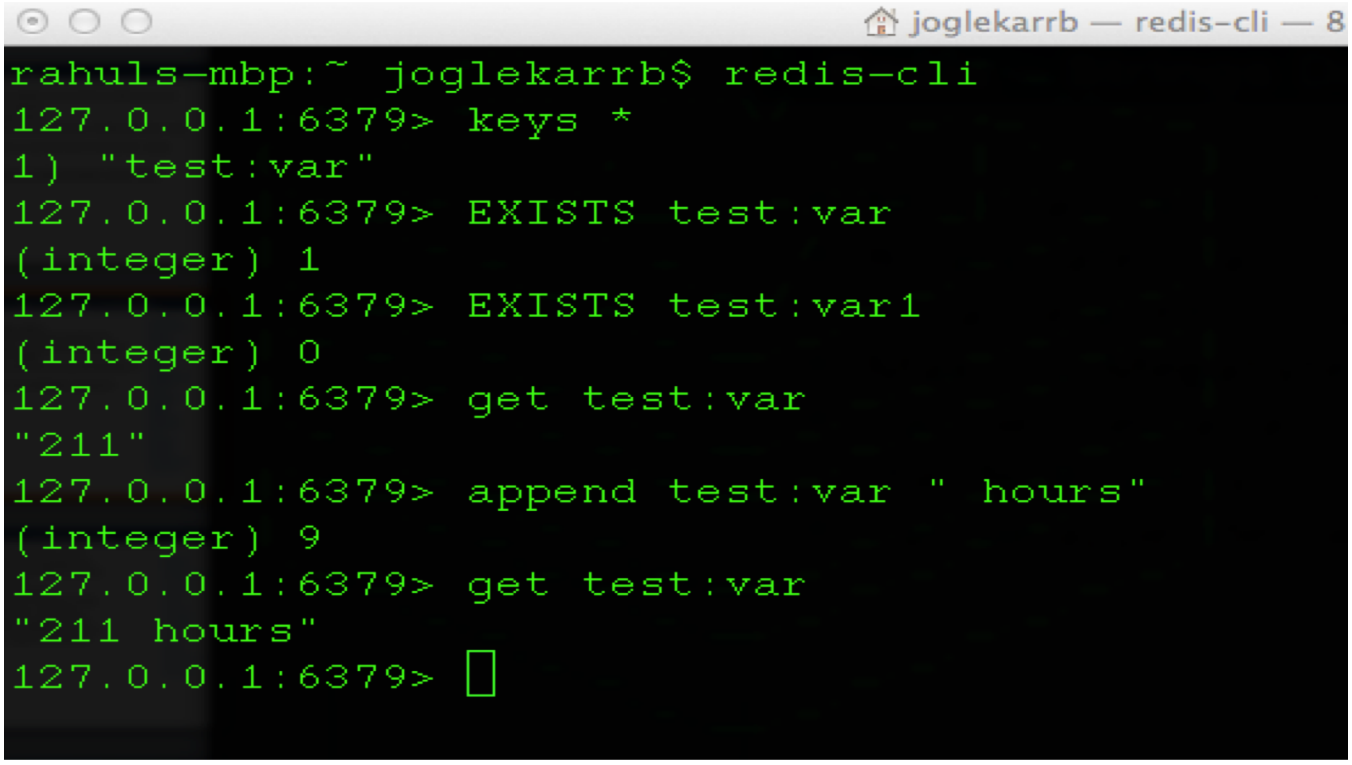
- Default - keys are retained
- Specified in
  - Seconds
  - Milliseconds
  - Unix Epoch
- Added to Key
- Removed from Key

REDIS Data


Key
"I'm a Plain Text String!"
00110101011001110010101010
{23334}{112345569}{766538}{665455}
{ A: "foo", B: "bar", C: "baz" }
[A → B → C → D → E]
{ A , B , C , D , E }
{ A: 0.1, B: 0.3, C: 100, D: 1337 }
{ A: (51.5, 0.12), B: (32.1, 34.7) }
00110101 11001110 10101010
Strings
Bitmaps
Bit field
Hashes
Lists
Sets
Sorted Sets
Geospatial Indexes
Hyperloglogs

Redis – “Strings” Data Type and Commands

- Most basic kind of Redis value
- Binary safe - can contain any kind of data, for instance a JPEG image or a serialized Ruby object
- Max 512 Megabytes in length
- Can be used as atomic counters using commands in the INCR family
- Can be appended with the APPEND command



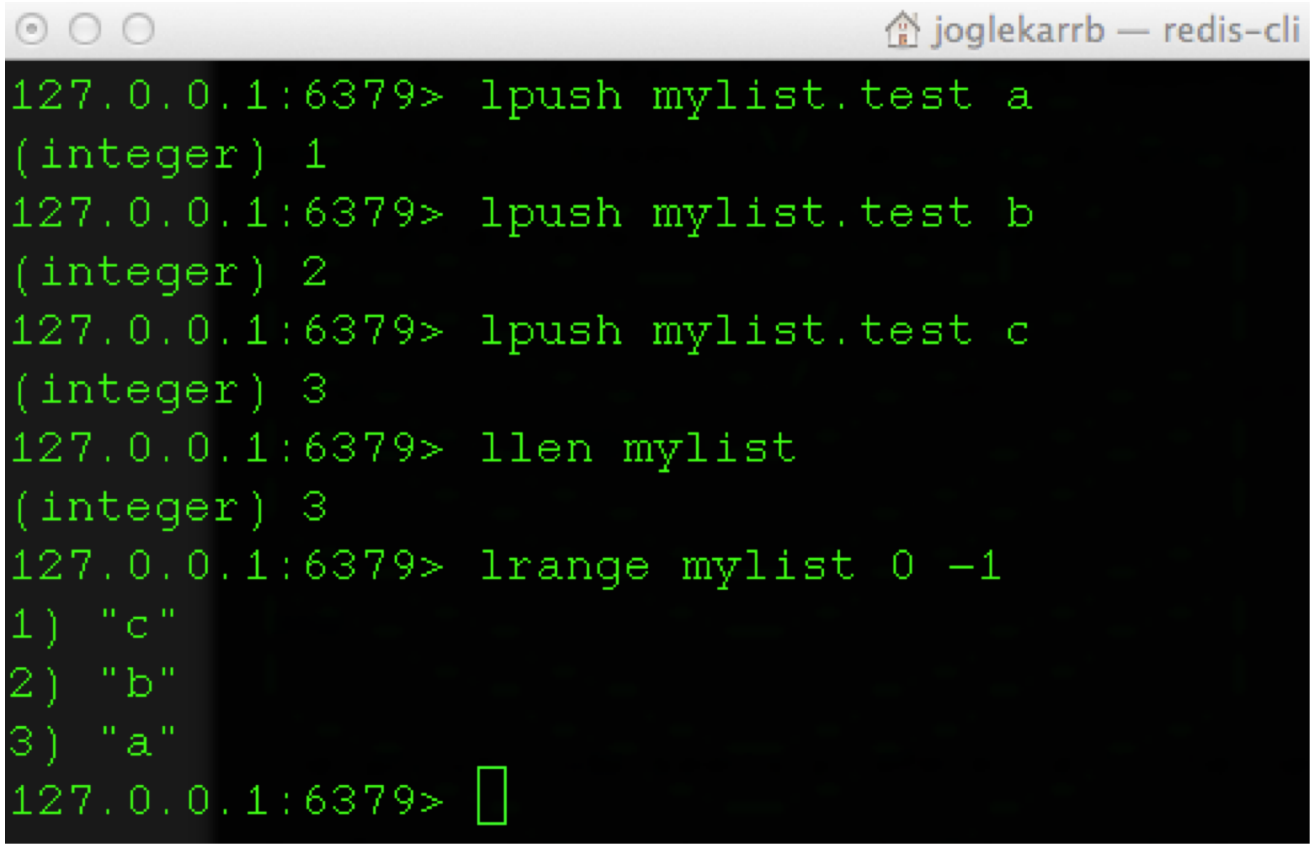
APPEND
BITCOUNT
BITOP
BITPOS
DECR
DECRBY
GET
GETBIT
GETRANGE
GETSET
INCR
INCRBY
MGET
MSET
MSETNX
PSETEX
SET
SETBIT
SETEX
SETNX
SETRANGE
STRLEN

Strings

Key Name	Value	Encoding
`> set event:Judo "Sold Out"`		
`event:Judo`	`"Sold Out"`	`embstr`
`> set event:Judo 42`		
`event:Judo`	`"42"`	`int`

Redis – “Lists” Data Type and Commands

- Think of a Dynamic Arrays , you can insert, append, pop, push, trim
- Add elements to a Redis List pushing new elements on the head (on the left) or on the tail (on the right) of list
- Max length: (2^32 - 1) elements
- Model a timeline in a social network, using LPUSH to add new elements, and using LRANGE in order to retrieve recent items
- Use LPUSH together with LTRIM to create a list that never exceeds a given number of elements



BLPOP
BRPOP
BRPOPLPUSH
LINDEX
LINSERT
LLEN
LPOP
LPUSH
LPUSHX
LRANGE
LREM
LSET
LTRIM
RPOP
RPOPLPUSH
RPUSH
RPUSHX

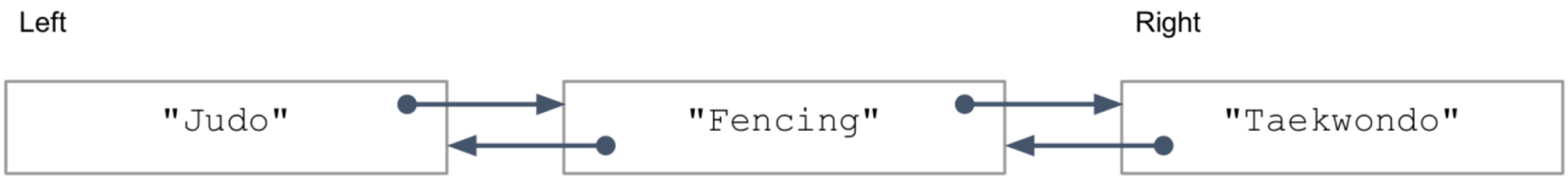
Lists

Key Name

```
events
```

Lists

- Ordered collection of Strings
- Duplicates allowed
- Elements added
  - Left, Right, by position

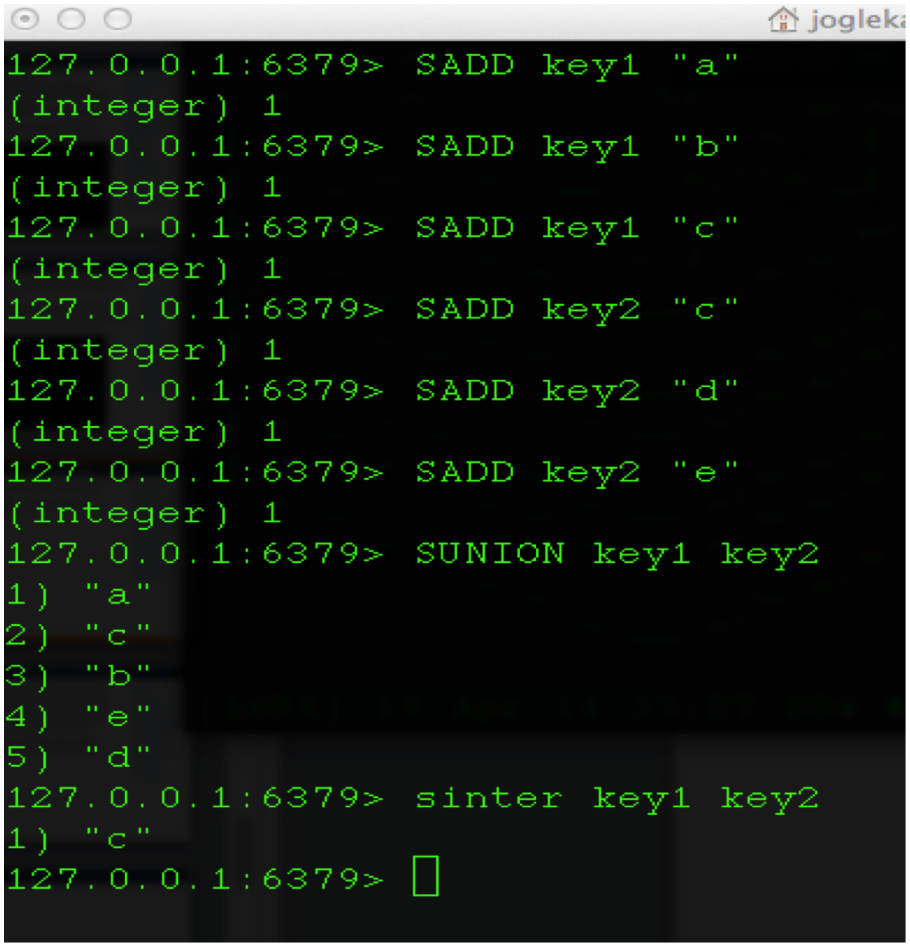



```
> lpop events
"Judo"
```

```
> rpop events
"Taekwondo"
```

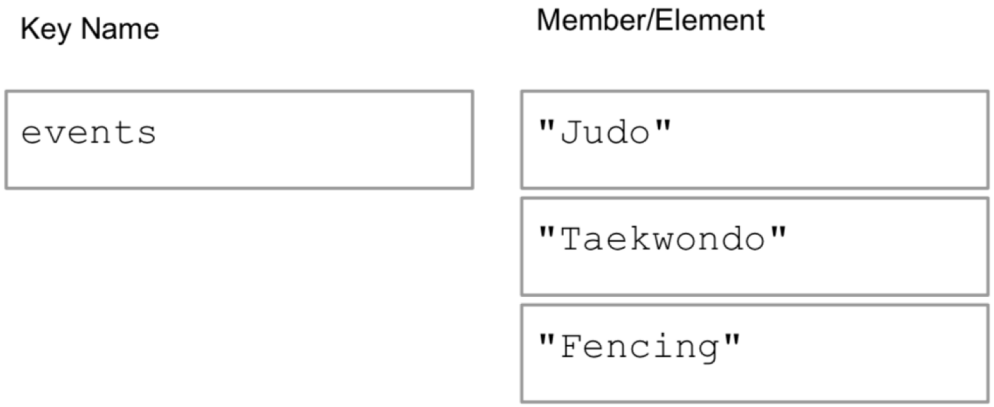
Redis – “Sets” Data Type and Commands

- Sets are a lot like lists, except they provide set-semantics
- You can diff sets via SDIFF, union two sets via SUNION or SUNIONSTORE



SADD
SCARD
SDIFF
SDIFFSTORE
SINTER
SINTERSTORE
SISMEMBER
SMEMBERS
SMOVE
SPOP
SRANDMEMBER
SREM
SSCAN
SUNION
SUNIONSTORE

Sets




```
> smembers events
1)  "Judo"
2)  "Taekwondo"
3)  "Fencing"
```

Sets

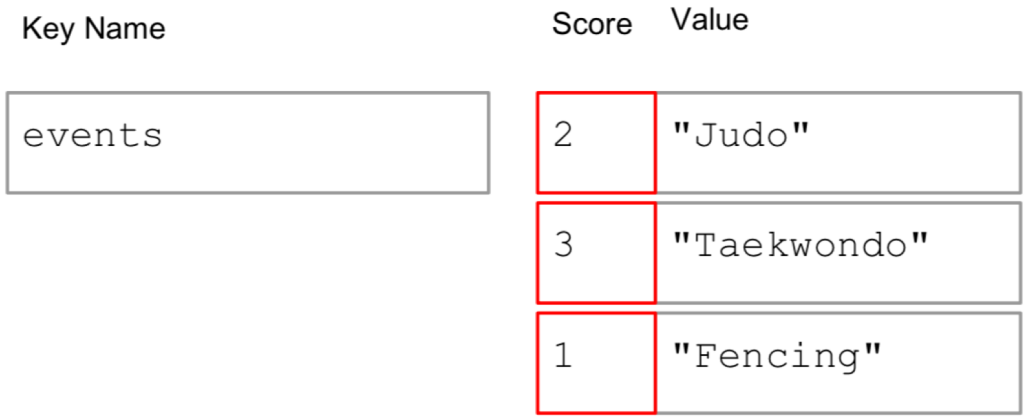
- Unordered collection of unique Strings
- Set operations
  - Difference
  - Intersect
  - Union

Redis – “Sorted Sets” Data Type and Commands

- A sorted set is similar to a set, except each value is associated (and sorted by) a score field that is used in order to take the sorted set ordered, from the smallest to the greatest score
- Probably the most advanced Redis data type

ZADD
ZCARD
ZCOUNT
ZINCRBY
ZINTERSTORE
ZRANGE
ZRANGEBYLEX
ZRANGEBYSCORE
ZRANK
ZREM
ZREMRANGEBYRANK
ZREMRANGEBYSCORE
ZREVRANGE
ZREVRANGEBYSCORE
ZREVRANK
ZSCAN
ZSCORE
ZUNIONSTORE

Sorted Sets




```
> zrange events 0 1
1)  "Fencing"
2)  "Judo"
```

Sets

- Ordered collection of unique Strings
- Manipulation by value, score, position or lexicography
- Set operations
  - Intersect
  - Union

Redis – “Hash” Data Type and Commands

- Map between string fields and string values
- Control over individual fields unlike a string
- Perfect data type to represent objects

```
HMSET user:1000 username rjoglekar password
P1pp0 age 34
HGETALL user:1000
HSET user:1000 password 12345
HGETALL user:1000
```

HDEL
HEXISTS
HGET
HGETALL
HINCRBY
HINCRBYFLOAT
HKEYS
HLEN
HMGET
HMSET
HSCAN
HSET
HSETNX
HVALS

Hash

Key Name	Score	Value
event:Judo	venue	Super Dome
	capacity	32000
	subway	True

```
> hget event:Judo capacity
"32000"
```

Hashes

- Field & Value pairs
- Single level
- Dynamically add remove fields

REDIS Transactions

- Redis has Transactions!
  - Atomic
  - Isolated
- Commands queued
- Queued commands executed sequentially as a single unit

```
> multi
> set event:Judo 100
> incr event:Judo
> exec
```

```
> multi
> set event:Judo "Sold Out"
> discard
```

https://redis.io/topics/transactions

MULTI, EXEC, DISCARD and WATCH are the foundation of transactions in Redis. They allow the execution of a group of commands in a single step, with two important guarantees:

- All the commands in a transaction are serialized and executed sequentially. It can never happen that a request issued by another client is served **in the middle** of the execution of a Redis transaction. This guarantees that the commands are executed as a single isolated operation.
- Either all of the commands or none are processed, so a Redis transaction is also atomic. The EXEC command triggers the execution of all the commands in the transaction

Optimistic Concurrency

WATCH is used to provide a check-and-set (CAS) behavior to Redis transactions.

WATCHed keys are monitored in order to detect changes against them.
If at least one watched key is modified before the EXEC command, the whole transaction aborts, and EXEC returns a Null reply to notify that the transaction failed.

WATCH makes the EXEC conditional: we are asking Redis to perform the transaction only if none of the WATCHed keys were modified. (But they might be changed by the same client inside the transaction without aborting it.

Data Modeling in REDIS

- Understand
  - Your Use Case & operations required
  - Data cardinality & Distribution
  - Select Data Structure

- Think about
  - Flat name space - what is your naming convention?
  - Transactions - atomic, enable safe multi key writes
  - Relationships - flatten into Hashes, or combine with Sets

Shopping cart example applied to REDIS

Relational Model

carts

CartID	User
1	james
2	chris
3	james

cart_lines

Cart	Product	Qty
1	28	1
1	372	2
2	15	1
2	160	5
2	201	7

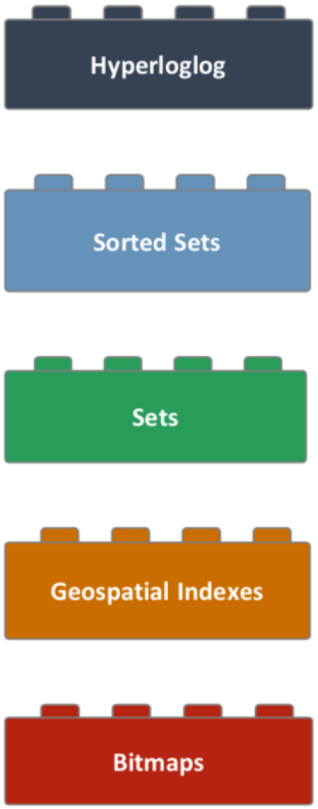
```
UPDATE cart_lines
SET    Qty = Qty + 2
WHERE  Cart=1 AND Product=28
```

Redis Model

```
set carts_james ( 1 3 )
set carts_chris ( 2 )
hash cart_1 {
   user       : "james"
   product_28 : 1
   product_372: 2
}
hash cart_2 {
   user       : "chris"
   product_15 : 1
   product_160: 5
   product_201: 7
}
```

HINCRBY cart_1 product_28 2

Analytics – What Data Structures to use ?



Probabilistic estimates of counts for anomaly detection

Real time range analyses, top scorers, bid ranges

Cardinality for fraud mitigation

Location based searches

Real-time population counting for activity monitoring

REDIS Modules

Redis ML

Machine learning models and model serving

Redis-cell

Rate-limiting

Rebloom

Probabilistic membership queries

T-digest

Rank-based statistics estimator

Topk

Track the top-k most frequent elements in a stream

Countminsketch

Approximate frequency counter

RediSearch

Extremely fast text-based search, used for secondary indexing

Redis Graph

Graph query processing

Time Series

Range analyses, built-in aggregations (min, max, sum.avg)

FRAUD – Hypothetical UseCase & implementation in REDIS

User Geographic Check

- Is the user logging in from a place that seems rational?
  - Someone who lives in the US suddenly logs in from Mongolia.
- Determine rationality:
  - Simple: Check if the person has ever visited the location of the login before. If not, then require additional verification.
  - Advanced: Check bordering regions (someone from the US can login from Mexico or Canada)

How it Works in Redis

- Every valid login location is added to Bloom filter.
- Before login, check to see if the location for the login attempt is in the Bloom filter.
- Properties of Bloom filter means that you'll know with certainty if a user has never been to this location.
  - False positives are possible, but controllable
- ReBloom Module

REDIS Modules

redis-server --loadmodule /path/to/neuralredis.so

OR

Redis.conf: loadmodule /path/to/module.so

REDIS BLOOM

docker run -p 6379:6379 --name redis-redisbloom redislabs/rebloom:latest

A Bloom filter is a probabilistic data structure which provides an efficient way to verify that an entry is certainly *not* in a set. This makes it especially ideal when trying to search for items on expensive-to-access resources (such as over a network or disk): If I have a large on-disk database and I want to know if the key *foo* exists in it, I can query the Bloom filter first, which will tell me with a certainty whether it potentially exists (and then the disk lookup can continue) or whether it does *not* exist, and in this case I can forego the expensive disk lookup and simply send a negative reply up the stack.

"An 80% solution today is much better than an 100% solution tomorrow."

- Does not apply to your homework

Redis Clients and Language support

JRedis - Java Client for Redis

Go-Redis

ServiceStack.Redis

redis-rb

HIREDIS

phpredis

carmine

Python

1. Install the python Redis library

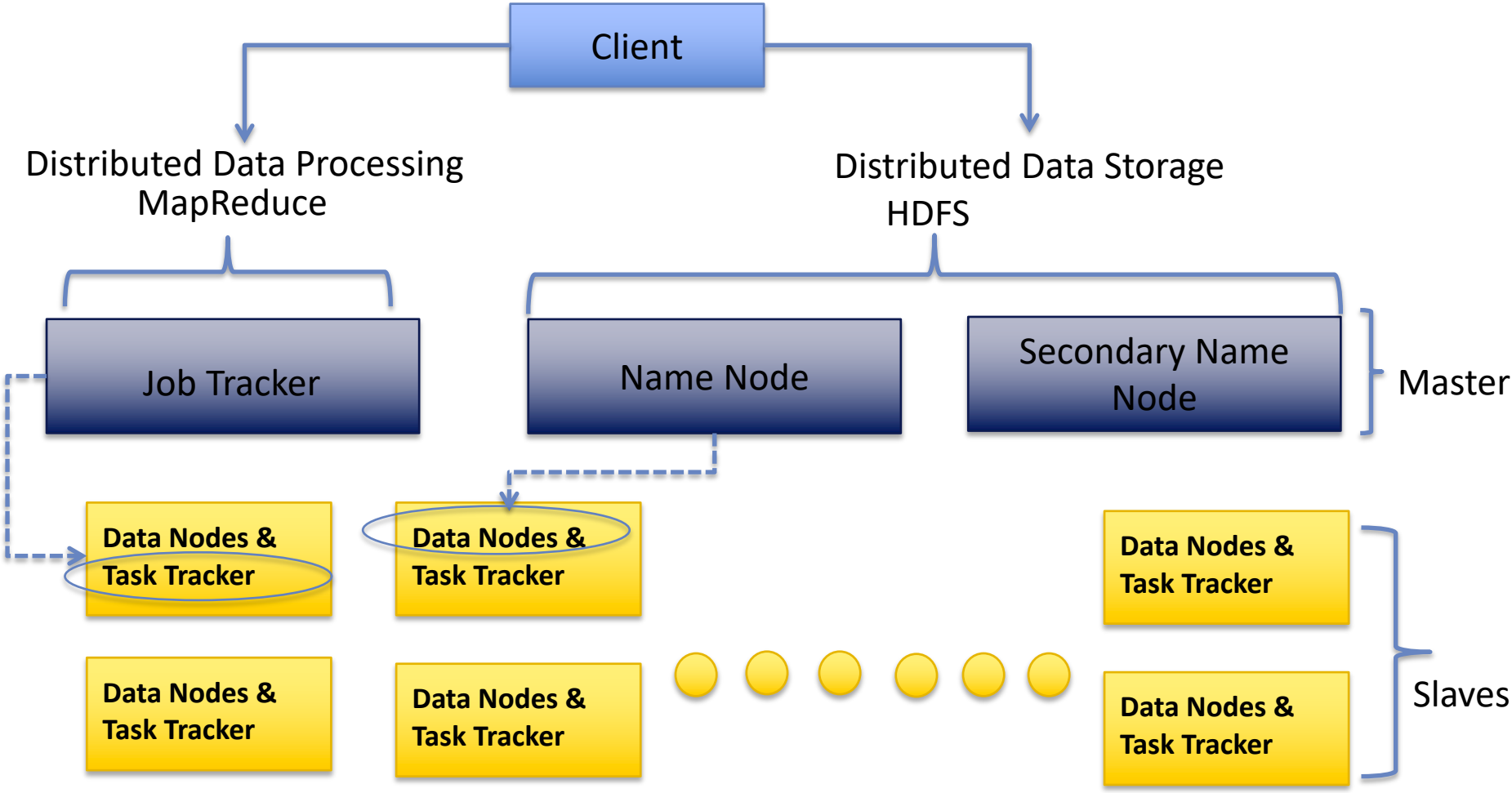
$ pip install redis

2. Check if all is well with doing a import

```
rjoglekar74 at RAHULs-MBP in ~
$ python3
Python 3.7.3 (default, Apr  3 2019, 08:57:53)
[Clang 10.0.1 (clang-1001.0.46.3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import redis
```

HDFS

Physical Architecture - Hadoop Server Roles

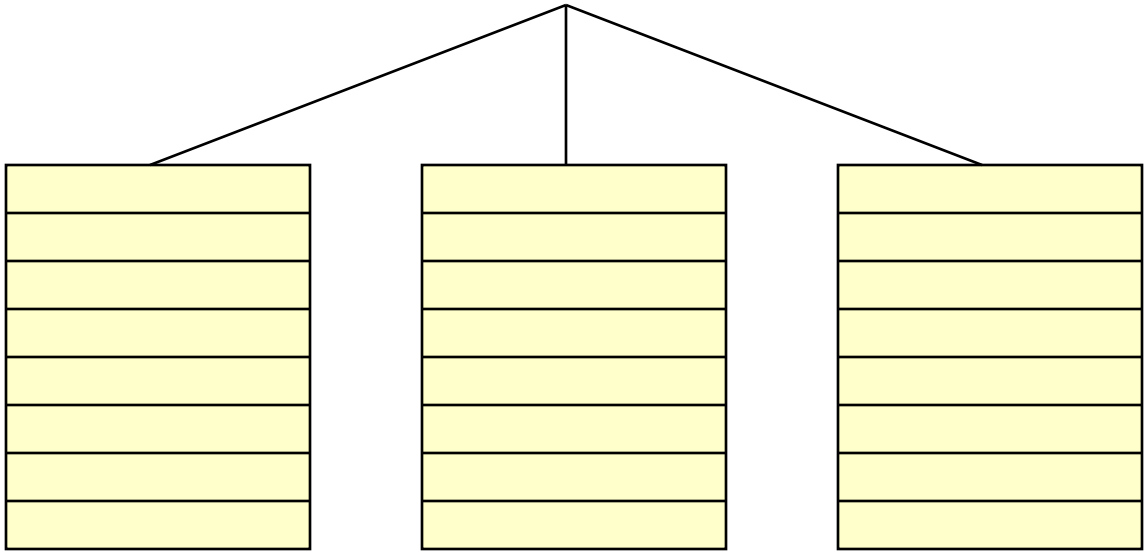


Hadoop Architecture – Data Node and Name Node

- ✓ NameNode
  - Stores all metadata: filenames, locations of each block on DataNodes, file attributes, etc…
  - In memory stores for fast lookup
  - Filesystem metadata size is limited to the amount of available RAM on NameNode
  - DataNodes exchange heartbeats with NameNode
  - If no heartbeat received within a certain time period – DataNode is assumed to be lost
    - NameNode determines which blocks were on the lost node
    - NameNode finds other copies of these 'lost' blocks and replicates them to other nodes
    - Block replication is actively maintained

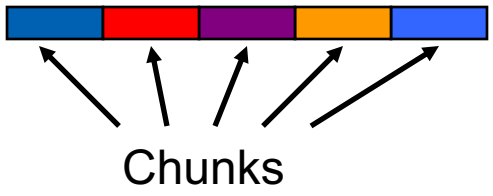
- ✓ DataNode
  - Stores file contents as blocks
  - Different blocks of the same file are stored on different DataNodes
  - Same block is replicated across several DataNodes for redundancy
  - Periodically sends a report of all existing blocks to the NameNode

Physical Architechture - GFS/HDFS

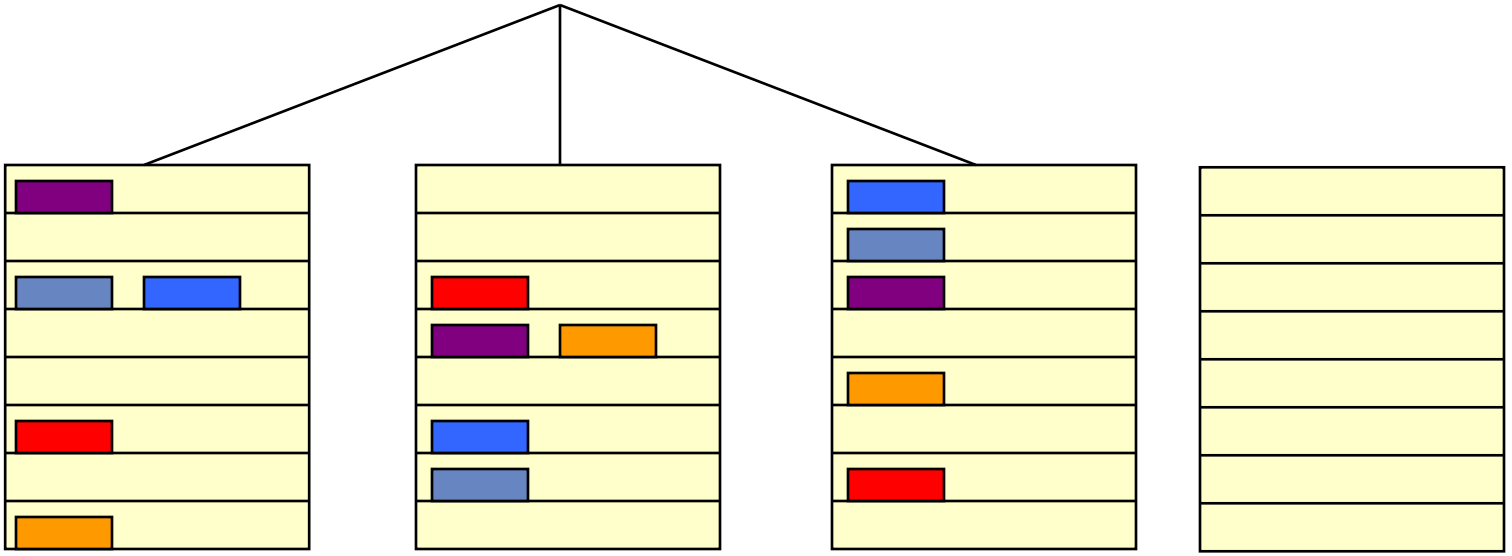


Racks of Compute Nodes

File



Physical Architecture - GFS/HDFS



3-way replication of files, with copies on different racks.

HDFS – Concepts and Design

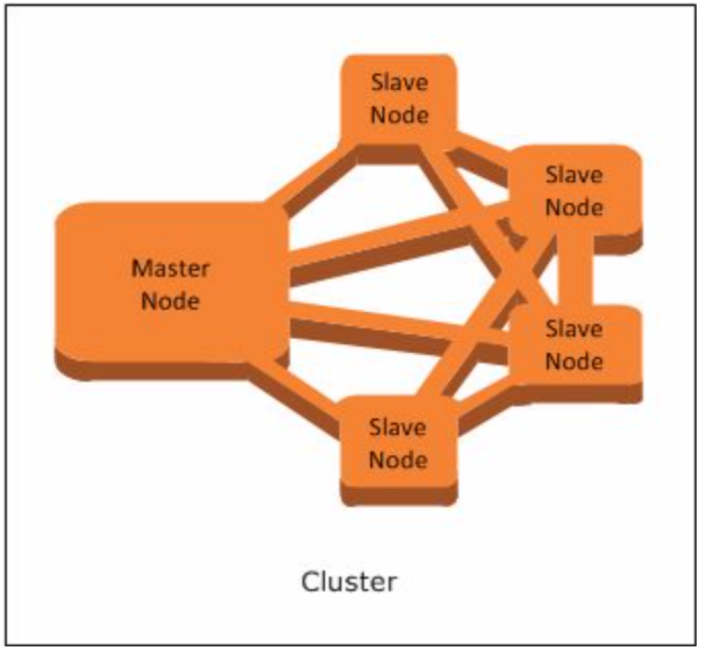
- ✓ Based on Google's GFS (Google File System)
- ✓ Provides redundant storage of massive amounts of data using commodity hardware
- ✓ Data is distributed across all nodes at load time
- ✓ Runs on commodity hardware
  - Assumes high failure rates of the components
- ✓ Works well with lots of large files
  - Hundreds of Gigabytes or terabytes in size
  - Block Size 128 MB
- ✓ Built around the idea of "write-once, read-many-times"
  - Large streaming reads
- ✓ Not random access
  - High throughput is more important than low latency

Configs in **hdfs-site.xml**

```
$ hadoop fs -rm -r -skipTrash delete/test2
```

AWS EMR Cluster Setup - DEMO

1. Create Cluster
   a. Use the latest emr release
   b. Select m4.large instance type - least expensive option
   c. Have logging option enabled
   d. Select **Core Hadoop** as an application
2. Security Groups
3. Logs
4. Cluster access (login as hadoop user)
5. Sample commands
6. Running examples
   a. find / -name hadoop-mapreduce-examples*.jar | grep examples
   b. hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar



https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-overview.html

References-

http://www.redis.io

"NoSQL -- Your Ultimate Guide to the Non - Relational Universe!"
http://nosql-database.org/links.html
"NoSQL (RDBMS)"
http://en.wikipedia.org/wiki/NoSQL
PODC Keynote, July 19, 2000. *Towards Robust. Distributed Systems*. Dr. Eric A. *Brewer*. Professor, UC Berkeley. Co-Founder & Chief Scientist, Inktomi .
*www.eecs.berkeley.edu/~**brewer**/cs262b-2004/PODC-keynote.pdf*
"Brewer's CAP Theorem" posted by Julian Browne, January 11, 2009. http://www.julianbrowne.com/article/viewer/brewers-cap-theorem
"How to write a CV" Geek & Poke Cartoon http://geekandpoke.typepad.com/geekandpoke/2011/01/nosql.html

"Exploring CouchDB: A document-oriented database for Web applications", Joe Lennon, Software developer, Core International.
http://www.ibm.com/developerworks/opensource/library/os-couchdb/index.html
"Graph Databases, NOSQL and Neo4j" Posted by Peter Neubauer on May 12, 2010 at: http://www.infoq.com/articles/graph-nosql-neo4j
"Cassandra vs MongoDB vs CouchDB vs Redis vs Riak vs HBase comparison", Kristóf Kovács. http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis
"Distinguishing Two Major Types of Column-Stores" Posted by Daniel Abadi onMarch 29, 2010
http://dbmsmusings.blogspot.com/2010/03/distinguishing-two-major-types-of_29.html
"Scalable SQL", ACM Queue, Michael Rys, April 19, 2011
http://queue.acm.org/detail.cfm?id=1971597
"a practical guide to noSQL", Posted by Denise Miura on March 17, 2011 at http://blogs.marklogic.com/2011/03/17/a-practical-guide-to-nosql/

Thank you.

Questions?