Intro to Redis

A Support Overview

redis

Matt Stancliff

Redis Engineer
NYC
mstancliff@gopivotal.com

Today

What is Redis?

How does Redis work?

How do we configure Redis?

How do Redis commands work?

How do we manage Redis?

How does Redis break?

How do we fix a broken Redis?

What is Redis?

Big Picture Overview

“Disk is the new tape.”

“Memory is the new disk.”

“Memory is the new disk.”

Task: Read 10,000 database records concurrently from a web app

Disk	RAM
300 reads per second (randomly located)	36,000,000 reads per second (randomly located)
(3 ms to 7 ms per read)	(20 ns to 60 ns per read)
30 seconds to 70 seconds total	0.2 ms to 0.6 ms total

30 s vs 0.0002 s

Task: Read 10,000 database records concurrently from a web app

Disk

RAM

Using RAM-only, you get results

150,000x faster

than disk-backed storage.

“Memory is [still] the new disk.”

Task: Read 10,000 database records concurrently from a web app

SSD	RAM
100,000 reads per second (randomly located)	10,000 reads per second (randomly located)
(0.1 ms to 1 ms per read)	(20 ms to 60 ms per read)
100 ms to 1000 ms total	0.2 ms to 0.6 ms total

300x faster than disk

500x slower than RAM

disk vs. SSD vs. RAM recap

disk

300
reads per second

SSD

100,000
reads per second

300x faster than disk

RAM

36,000,000
reads per second

150,000x faster than disk

500x faster than SSD

What does Redis do?

atomic scripting (transactions)

clustering [soon]

Redis is

a **data structure** server

with **replication**

pub/sub

persistent storage

async queues

structured data cache

queueing system

pub/sub messaging bus

Redis is

complementary infrastructure

use **with** PostgreSQL

MySQL

Hive/Pig/HBase/Hadoop

ranking users (top scores)

low latency analytics

Redis is

standalone infrastructure

primary datastore for

follwing/friending graphs

timelines newsfeeds

Redis is

in-memory

all storage kept live in RAM

150,000x faster than HDs

500x faster than SSD

automatic copying to **multiple standby replicas**

[soon] **clustering** data across dozens of hosts

Redis is

durable

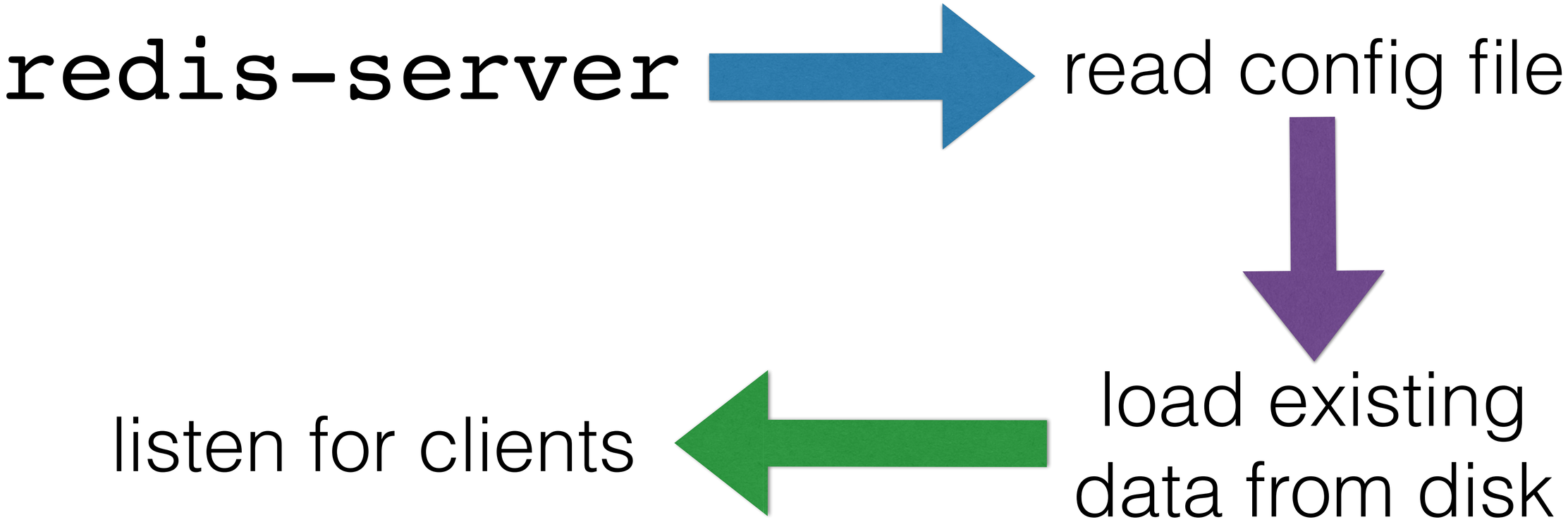
loads previous memory state from disk on startup

tunable **commit-every-update-to-disk** options

How does Redis work?

How does Redis work?

Startup



How does Redis work?

asynchronous

single-threaded

event loop

event loop

```
while
 proce
}
```

It's just a basic while(true) loop.

while(true) runs 500 million iterations

redis handles all operations in this loop.

result:

redis is single threaded.

redis uses only one core.

`processEvents()`

- check for network events
  - new clients (IPv4, IPv6, domain sockets)
  - connected clients running commands
- process scheduled events
  - 10 times per second:
    - replication sanity check
    - force-expire keys
    - persistence sanity check
    - … and a dozen other things.

How does Redis work?

Data

Redis stores **all** data in memory

optional backup-to-disk settings for:

append-only journal file

every change appended to a file

(like binlogs)

complete DB snapshot to disk

writes entire dataset to disk

more compact than append-only file

Redis is **single-threaded**

(except for this)

append-only journal:

- written using a background thread
  - (the only thread in Redis)

complete DB snapshot:

- forks a new `redis-server` process
- serializes the child's frozen/"snapshot" memory to disk
- can cause performance hiccups on EC2 or large data sets

Redis is **single-threaded**

advantage:

- no locks
- your command is the only one running

disadvantage:

- your command is the only one running
- “bad” commands can block the server for seconds
- poorly designed in-server scripts can block forever

How does Redis configure?

redis.conf

plain text file

[name] [value]

sample entries:

```
# Accept connections on the specified port, default is 6379.
# If port 0 is specified Redis will not listen on a TCP socket.
port 6379
save 900 1
save 300 10
save 60 10000
```

```
# The filename where to dump the DB
dbfilename dump.rdb
```

Top 5 Config Settings

Network

```
# Accept connections on the specified port, default is 6379.
# If port 0 is specified Redis will not listen on a TCP socket.
port 6379

bind 192.168.1.100 10.0.0.1
bind ::1

unixsocket /tmp/redis.sock
unixsocketperm 755
```

Top 5 Config Settings

Persistence

```
# Save DB if at least <changes> happen in <seconds>:
#
#   save <seconds> <changes>
save 900 1
save 300 10
save 60 10000
```

```
# Directory path to store DB, AOF, and
# replication/cluster metadata.
# Redis instances *must not* share directories.
dir ./
```

Top 5 Config Settings

More Persistence

```
appendonly yes
```

```
# appendfsync always
appendfsync everysec
# appendfsync no
```

```
# Automatically rewrite the log file implicitly calling
# BGREWRITEAOF when the AOF size grows by percentage.
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
```

Top 5 Config Settings

Memory Limits

```
# Don't use more memory than the specified amount of bytes.
# When the memory limit is reached Redis will try to remove keys
# accordingly to the eviction policy selected (see maxmemmory-policy).

maxmemory <bytes>
```

Top 5 Config Settings

More Memory Limits

“pretend memcache”

safe

```
# volatile-lru -> remove the key with an expire set using an LRU algorithm
# allkeys-lru -> remove any key accordingly to the LRU algorithm
# volatile-random -> remove a random key with an expire set
# allkeys-random -> remove a random key, any key
# volatile-ttl -> remove the key with the nearest expire time (minor TTL)
# noeviction -> never expire at all, just return an error on write operations
```

kinda dumb

weird

also safe

```
maxmemory-policy volatile-lru
```

Top 5 Config Settings

Replication

```
# Master-Slave replication.
# Use slaveof to make a Redis instance a copy of another Redis server.

slaveof <masterip> <masterport>
```

```
slave-serve-stale-data yes
```

serves data while
disconnected from master *and*
while a sync is in progress

(may have only
partial replica of master)

```
slave-read-only yes
```

```
min-slaves-to-write 3
min-slaves-max-lag 10
```

for reliable installations
deny writes if fewer than 3 in-sync replicas are live

Top 5 Config Settings

Cluster

```
# Normal Redis instances can't be part of a Redis Cluster; only nodes that are
# started as cluster nodes can. In order to [illegible] instance as a
# cluster node enable the cluster sup[illegible]ing:

cluster-enabled yes
```

```
cluster-config-file node-6379.conf
```

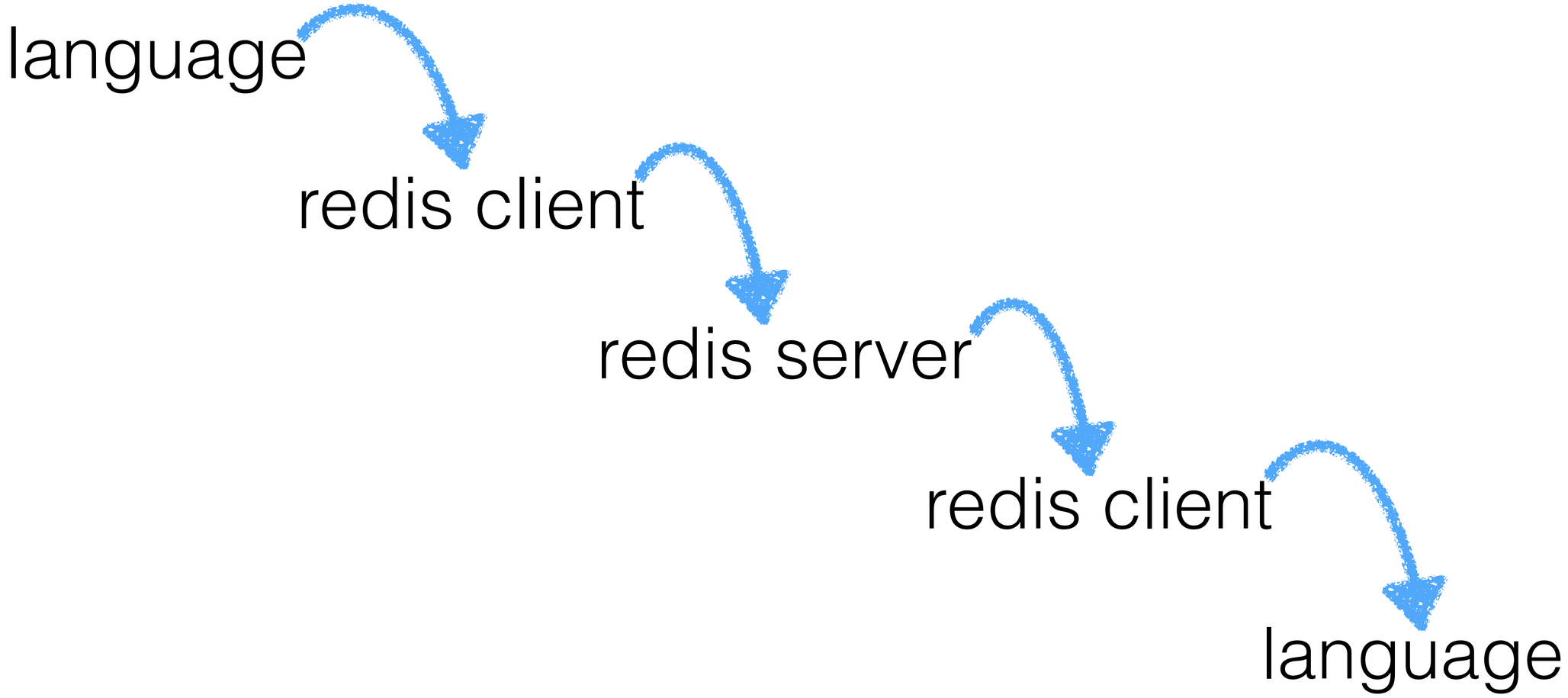
must be a unique file per server.

Only the server writes to **`cluster-config-file`**.

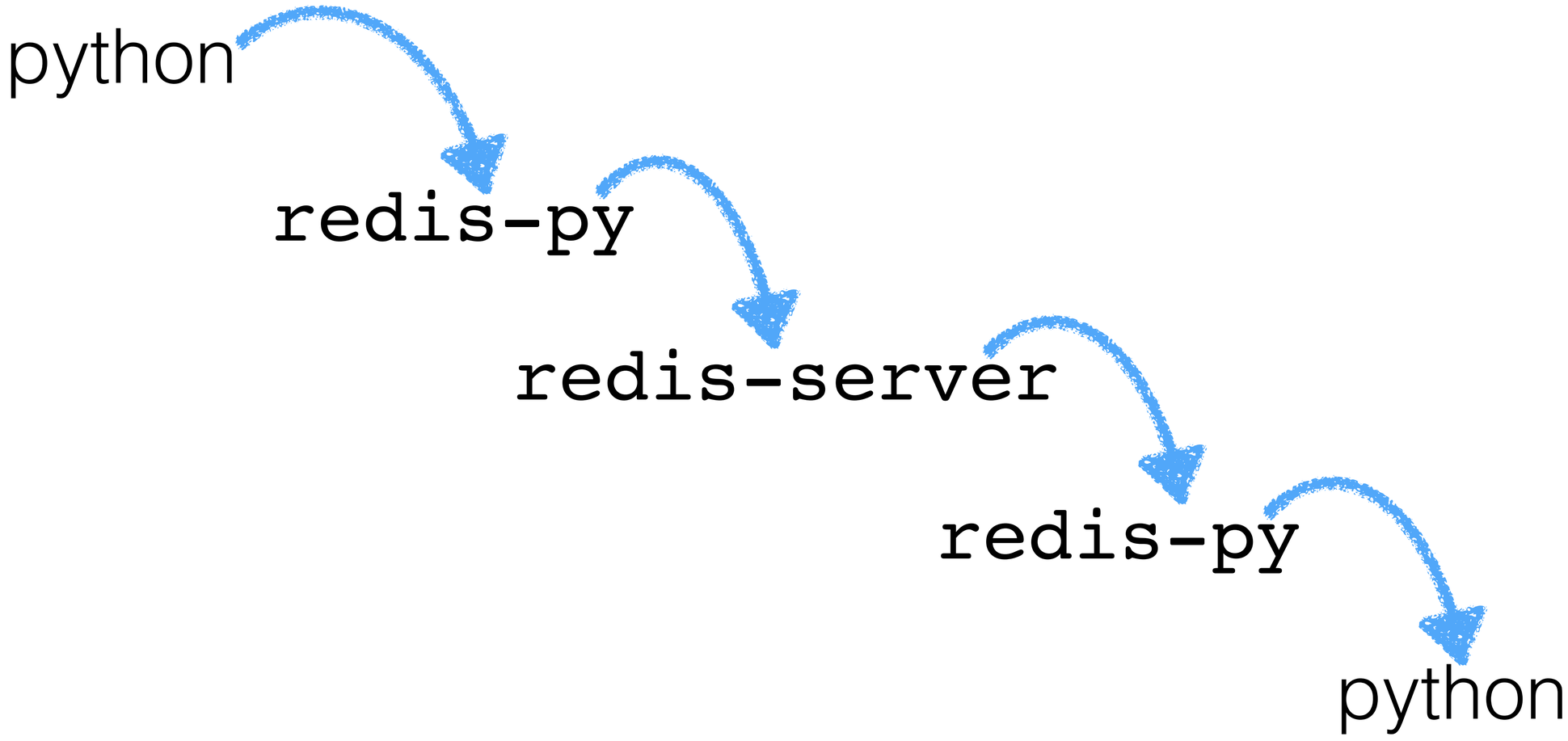
(persists cluster membership information across restarts)

How Redis program?

Programmer’s View of Redis



Programmer's View of Redis



Programmer's View of Redis

Two Types of Clients

every-command-is-a-function

```
r = redis.Redis()
r.set('bing', 'baz')
```

cleaner, has language feel

free command entry fields

```
r = redis.Redis()
r.cmd("SET bing baz")
```

more flexible,
supports arbitrary commands

Programmer's View of Redis

Two Types of Clients

every-command-is-a-function

```
r = redis.Redis()
r.set('w', 34.3)
c = r.incrbyfloat('w', 0.1)
```

c is now the **float** 34.4

free command entry fields

```
r = redis.Redis()
r.cmd("SET w 34.3")
c = r.cmd("INCRBYFLOAT w 0.1")
```

depending on the client,
c is now the **string** "34.4"
- or -
c is now the **float** 34.4

Programmer's View of Redis

Two Types of Clients

every-command-is-a-function

```
r = redis.Redis()
r.set('w', 34.3)
r.incrbyfloat('w', 0.1)
c = r.get('w')
```

c is now the **string** 34.4

free command entry fields

```
r = redis.Redis()
r.cmd("SET w 34.3")
r.cmd("INCRBYFLOAT w 0.1")
c = r.cmd("GET w")
```

c is now the **string** 34.4

Programmer's View of Redis

Clients

http://redis.io/clients

C: hiredis

Java: Jedis

Perl: Redis

PHP: Predis

Python: redis-py

Ruby: redis-rb

Redis commands aren’t always **simple**

```
SORT key [BY pattern] [LIMIT offset count] [GET pattern
[GET pattern ...]] [ASC|DESC] [ALPHA] [STORE
destination]
```

can be represented in languages with optional parameters:

```
sort(name, start=None, num=None, by=None,
get=None, desc=False, alpha=False, store=None)
```

multi-get

```
MGET key1 key2 key3 key4 …
```

```
MSET key1 val1 key2 val2 key3 val3 key4 val4
```

multi-set

```
SET key value [EX seconds] [PX milliseconds] [NX|XX]
```

optional arguments

Redis Command Types

sets

key management

sorted sets

hashes

server management

scripting

strings

lists

pub/sub

transactions

Redis Data Types

Basics

key = value

keys are strings

strings are binary safe

strings have a max size of 512 MB

values have a type:
string, list, hash, set, or sorted set

Redis Data Types

Basics

Quick Note:

http://redis.io/ embeds live redis sessions

Redis Data Types

Strings

```
127.0.0.1:6379> SET location:kitten "in a tree"
OK
127.0.0.1:6379> GET location:kitten
"in a tree"
127.0.0.1:6379> STRLEN location:kitten
(integer) 9
```

Redis Data Types

More Strings

```
127.0.0.1:6379> APPEND location:kitten " in the park"
(integer) 21
127.0.0.1:6379> GET location:kitten
"in a tree in the park"
127.0.0.1:6379> SETRANGE location:kitten 17 mall
(integer) 21
127.0.0.1:6379> GET location:kitten
"in a tree in the mall"
```

Redis Data Types

Even More Strings

BITCOUNT
GETBIT
SETBIT
BITOP

SETEX
PSETEX

INCR
INCRBY
INCRBYFLOAT
DECR
DECRBY

GETSET

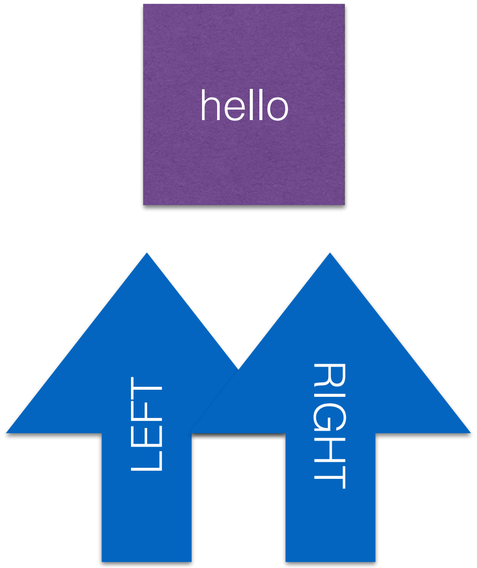
SETNX
MSETNX

MGET
MSET

Redis Data Types

Lists

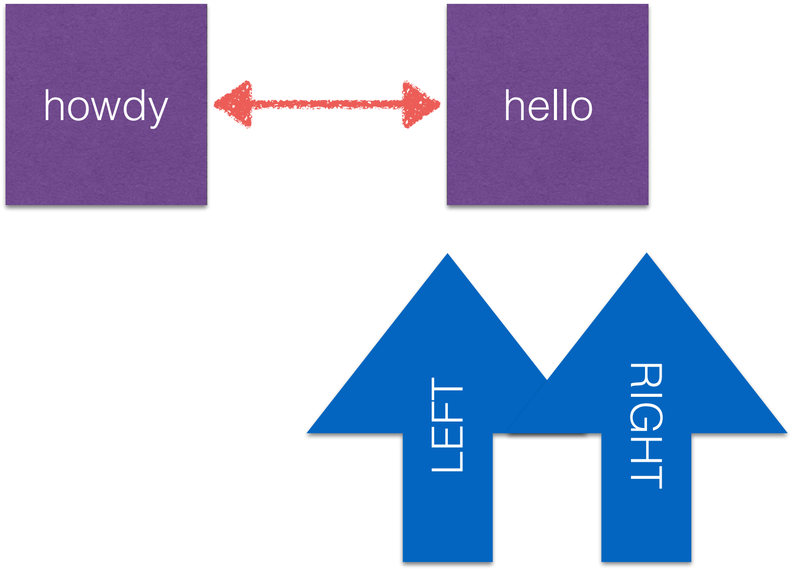
```
LPUSH aList hello
```



Redis Data Types

Lists

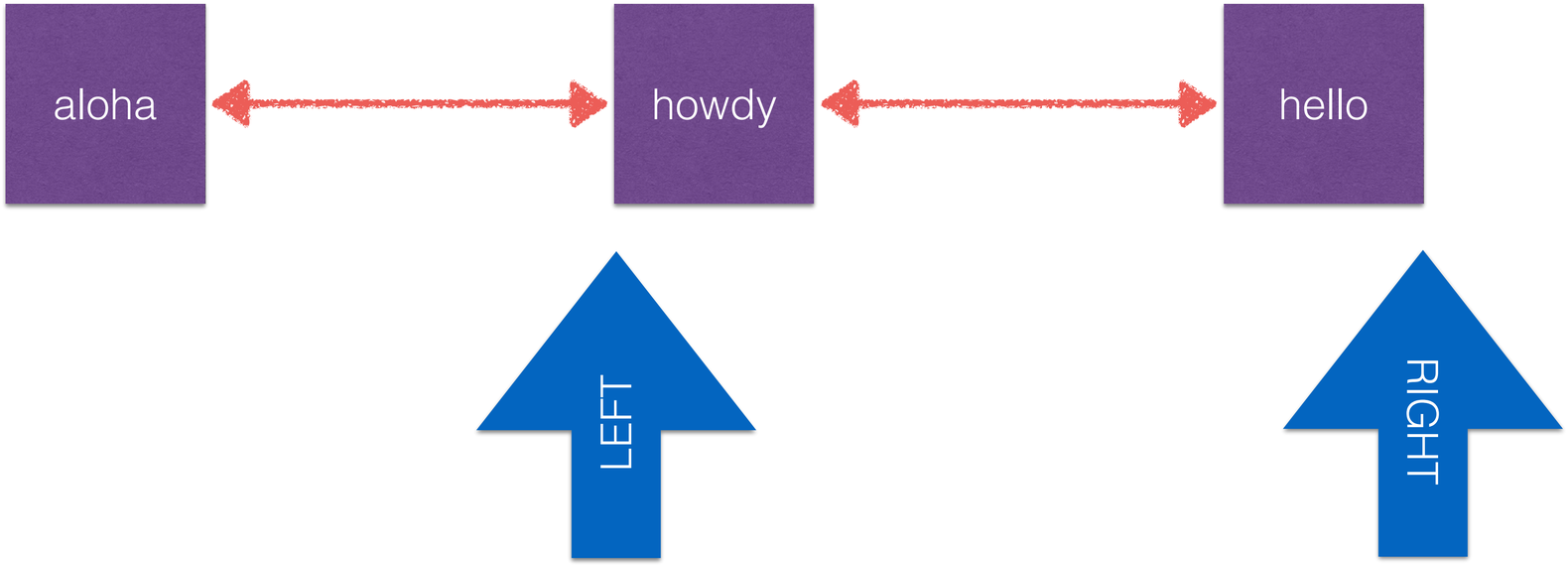
```
LPUSH aList howdy
```



Redis Data Types

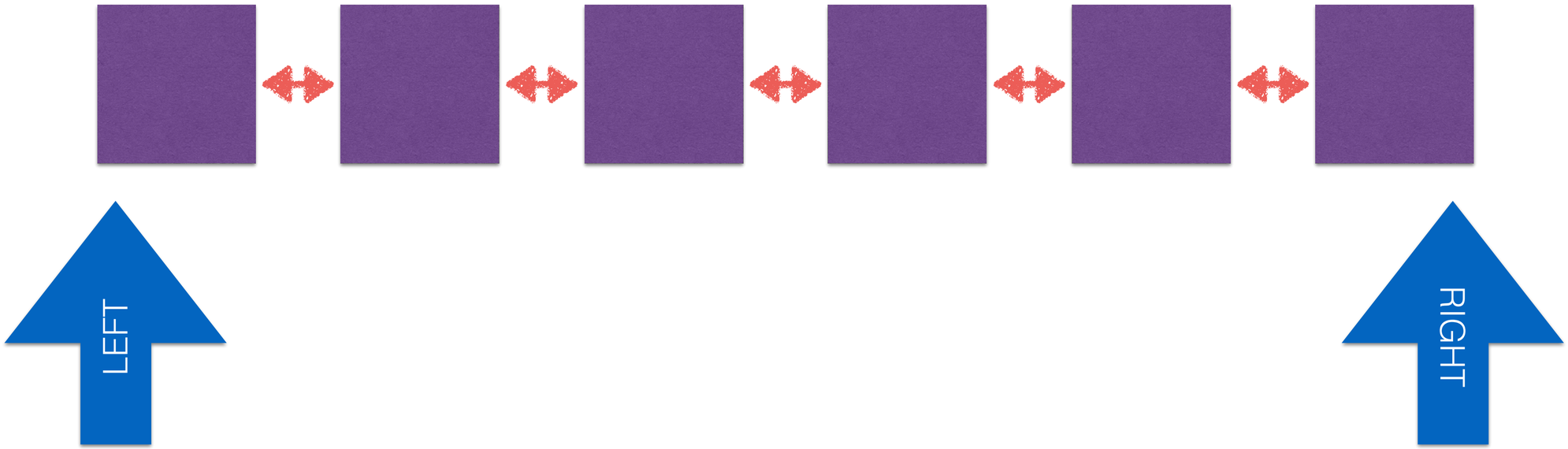
Lists

```
LPUSH aList aloha
```



Redis Data Types

Lists



Redis Data Types

Lists

```
127.0.0.1:6379> LPUSH mention:redis redis.io
(integer) 1
127.0.0.1:6379> LPUSH mention:redis news.ycombinator.com
(integer) 2
127.0.0.1:6379> LPUSH mention:redis github.com/antirez
(integer) 3
127.0.0.1:6379> LRANGE mention:redis 0 -1
1) "github.com/antirez"
2) "news.ycombinator.com"
3) "redis.io"
```

Redis Data Types

Lists

```
127.0.0.1:6379> LPOP mention:redis
"github.com/antirez"
127.0.0.1:6379> RPOP mention:redis
"redis.io"
127.0.0.1:6379> LPOP mention:redis
"news.ycombinator.com"
```

Redis Data Types

More Lists

```
127.0.0.1:6379> LPOP mention:redis
(nil)
127.0.0.1:6379> BLPOP mention:redis 3
(nil)
(3.66s)
```

Redis Data Types

Even More Lists

BLPOP
BRPOP

RPOPLPUSH
BRPOPLPUSH

LLEN

LREM
LSET
LTRIM

LINDEX
LINSERT

LPUSHX
RPUSHX

Redis Data Types

Hashes

dictionaries

maps

hash tables

collections

Redis Data Types

Hashes

values are **only** strings

key =

field1 = val1
field2 = val2
field3 = val3
field4 = val4
.
.

Redis Data Types

Hashes

Logically:

hash with {field1, field2, … fieldN}

Redis Data Types

Hashes

Same as strings:

string:field1

string:field2

...

string:fieldN

Redis Data Types

Strings

repetition

wasted bytes

excess pointers

(8 bytes each)

account:3391:field1 → value1

account:3391:field2

account:3391:field3

account:3391:field4

account:3391:field5 → value5

account:3391:field6 → value6

compact, pointer-less* representation

Hash

one key

account:3391

one pointer

multiple fields

field1
value1
field2
value2
field3
value3
field4
value4
field5
value5
field6
value6

Redis Data Types

Hashes

```
127.0.0.1:6379> HSET user:matt name Matt
(integer) 1
127.0.0.1:6379> HSET user:matt company GoPivotal
(integer) 1
127.0.0.1:6379> HGETALL user:matt
1) "name"
2) "Matt"
3) "company"
4) "GoPivotal"
```

Redis Data Types

More Hashes

```
127.0.0.1:6379> HINCRBY user:matt loginCount 1
(integer) 1
127.0.0.1:6379> HINCRBY user:matt loginCount 1
(integer) 2
127.0.0.1:6379> HGETALL user:matt
1) "name"
2) "Matt"
3) "company"
4) "GoPivotal"
5) "loginCount"
6) "2"
```

Redis Data Types

More Hashes

```
127.0.0.1:6379> HMSET user:matt created 2013-10-28 lastSeen 1385393884 geohash dr5rm7w
OK
127.0.0.1:6379> HGETALL user:matt
 1) "name"
 2) "Matt"
 3) "company"
 4) "GoPivotal"
 5) "loginCount"
 6) "2"
 7) "created"
 8) "2013-10-28"
 9) "lastSeen"
10) "1385393884"
11) "geohash"
12) "dr5rm7w"
```

Redis Data Types

Even More Hashes

HKEYS

HVALS

HDEL

HINCRBYFLOAT

HGET

HMGET

HSETNX

HEXISTS

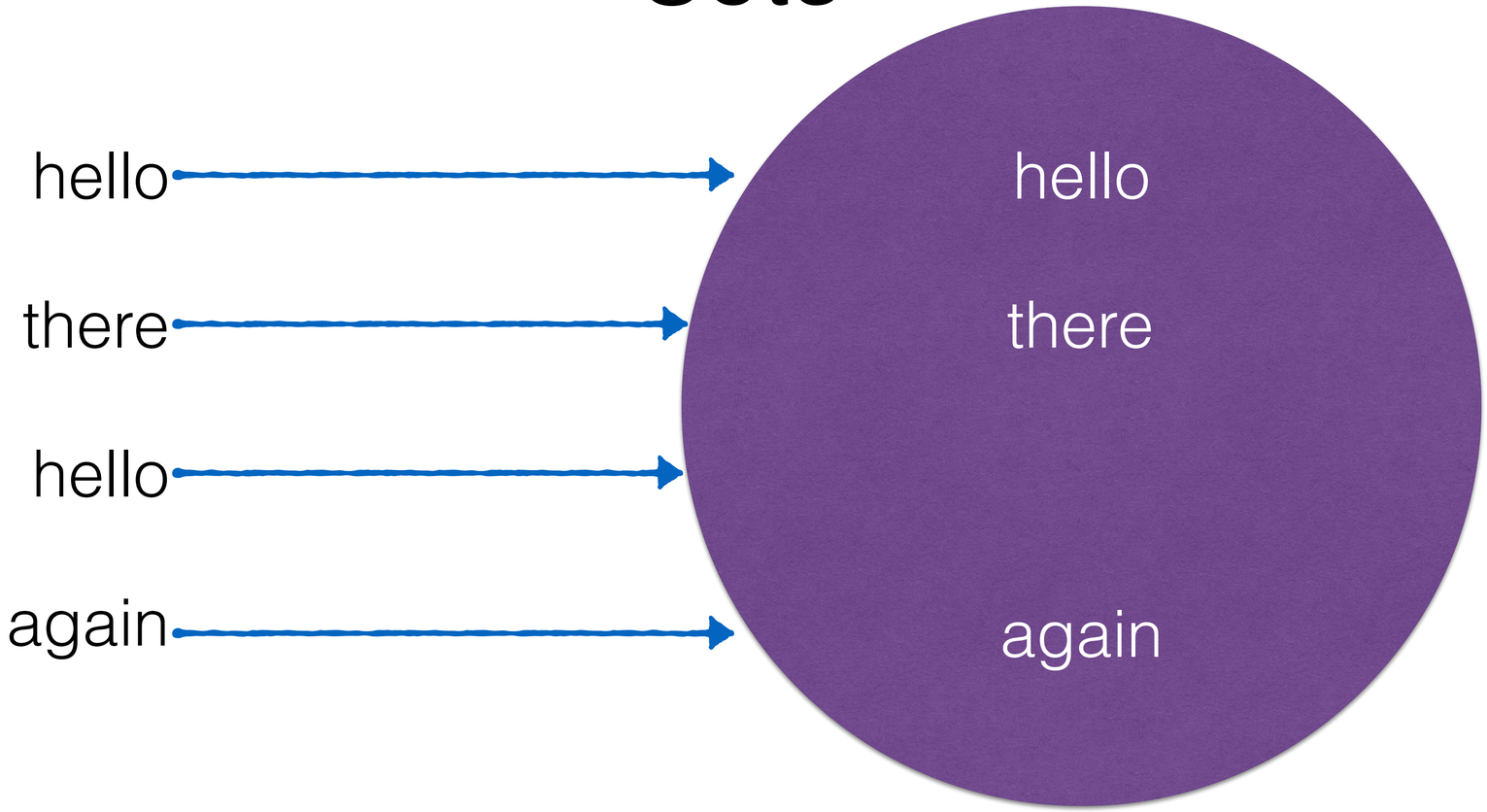
Redis Data Types

Sets

A collection of strings
no duplicates allowed
no order preserved

Redis Data Types

Sets



Redis Data Types

Sets

elements added

```
0.0.1:6379> SA le hello
(integer) 1
127.0.0.1:6379> there
(integer) 1
127.0.0.1:637 > SADD purpleCircle hello
(integer) 0
127.0.0.1:6379> SADD purpleCircle again
(integer) 1
```

duplicate.
nothing added.

Redis Data Types

Sets

```
127.0.0.1:6379> SMEMBERS purpleCircle
1) "there"
2) "hello"
3) "again"
```

Redis Data Types

More Sets

elements added

```
127.0.0.1:6379> SADD circle hello there hello again
(integer) 3
127.0.0.1:6379> SMEMBERS circle
1) "there"
2) "hello"
3) "again"
```

duplicate.
nothing added.

Redis Data Types

Even More Sets

SREM

SMOVE

SPOP

SCARD

SISMEMBER

SINTER

SINTERSTORE

SRANDMEMBER

SDIFF

SDIFFSTORE

SUNION

SUNIONSTORE

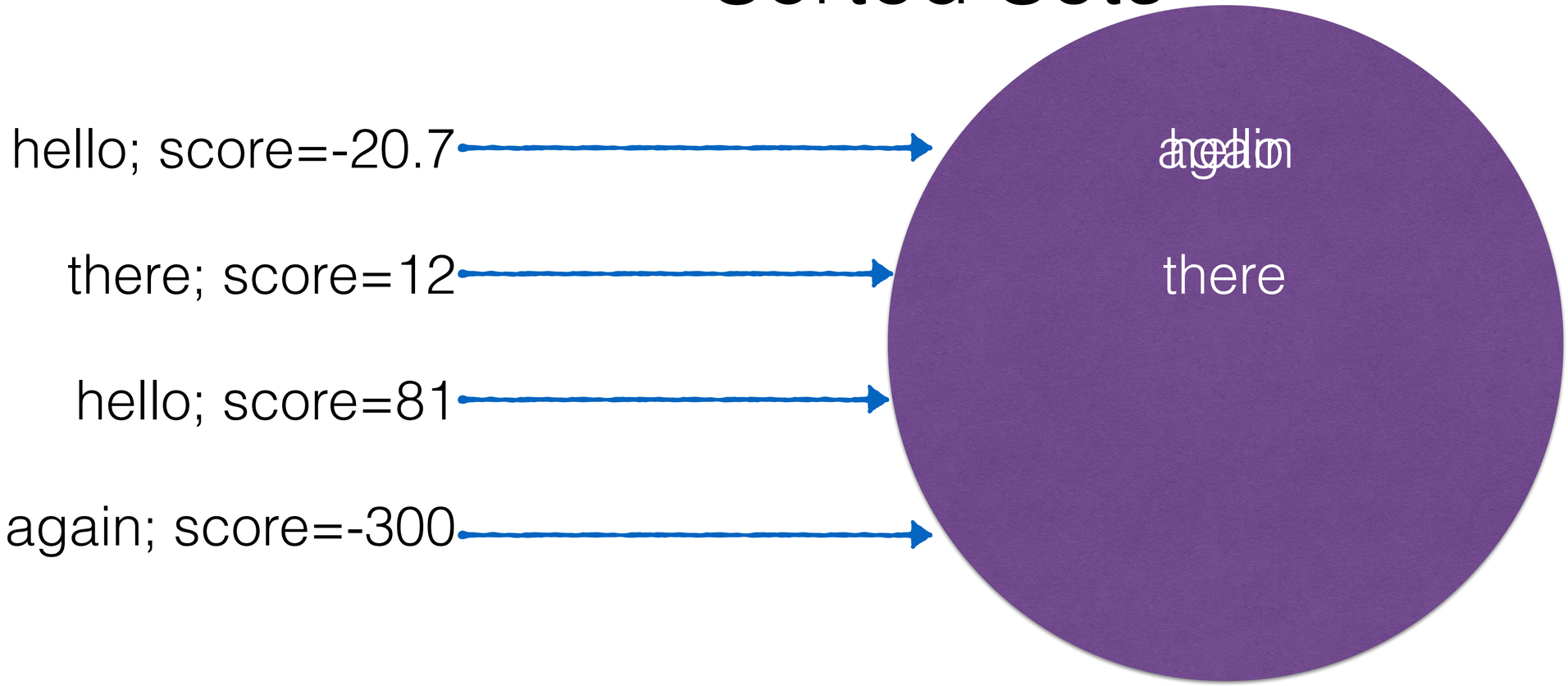
Redis Data Types

Sorted Sets

A collection of strings
no duplicates allowed
user-defined ordering

Redis Data Types

Sorted Sets



Redis Data Types

Sorted Sets

```
127.0.0.1:6379> ZADD purpleCircle -20.7 hello
(error) WRONGTYPE Operation against a key holding the wrong kind of value
127.0.0.1:6379> DEL purpleCircle circle
(integer) 2
```

multi-delete

Redis Data Types

Sorted Sets

elements added

```
127.0.0.1:6379> Z                    -20.7 hello
(integer) 1
127.0.0.1:6379> ZRANGEBYSCORE p                    cl
1) "hello"
127.0.0.1:6379> ZRANGEBYSCORE p                    WITHSCORES
1) "hello"
2) "-20.699999999999999"
```

score

element

because floating point

Redis Data Types

More Sorted Sets

```
127.0.0.1:6379> ZADD purpleCircle 12 there 81 hello -300 again
(integer) 2
```

`hello` already existed

score got updated

Redis Data Types

More Sorted Sets

start position

end position

```
127.0.0.1:6379> ZRANGE purpleCircle 0 -1
1) "again"
2) "there"
3) "hello"
127.0.0.1:6379> ZRANGE purpleCircle 0 -1 WITHSCORES
1) "again"
2) "-300"
3) "there"
4) "12"
5) "hello"
6) "81"
```

replaced
-20.7

Redis Data Types

Even More Sorted Sets

ZREM

ZREMRANGEBYRANK

ZREMRANGEBYSCORE

ZRANK

ZREVRANK

ZCARD

ZINTERSTORE

ZUNIONSTORE

ZRANGE

ZRANGEBYSCORE

ZREVRANGE

ZREVRANGEBYSCORE

How Redis manage?

Running

You can set config parameters:

on the command line (as arguments)
or
in the config file (the “normal” way)

Running

You can modify most parameters
live during runtime with **`CONFIG SET`**

Read settings with **`CONFIG GET [name]`**
or
`CONFIG GET *`
for all current settings

Running

Redis can update your existing config file

`CONFIG REWRITE`

Comments, ordering, and structure get preserved.

Stand Alone

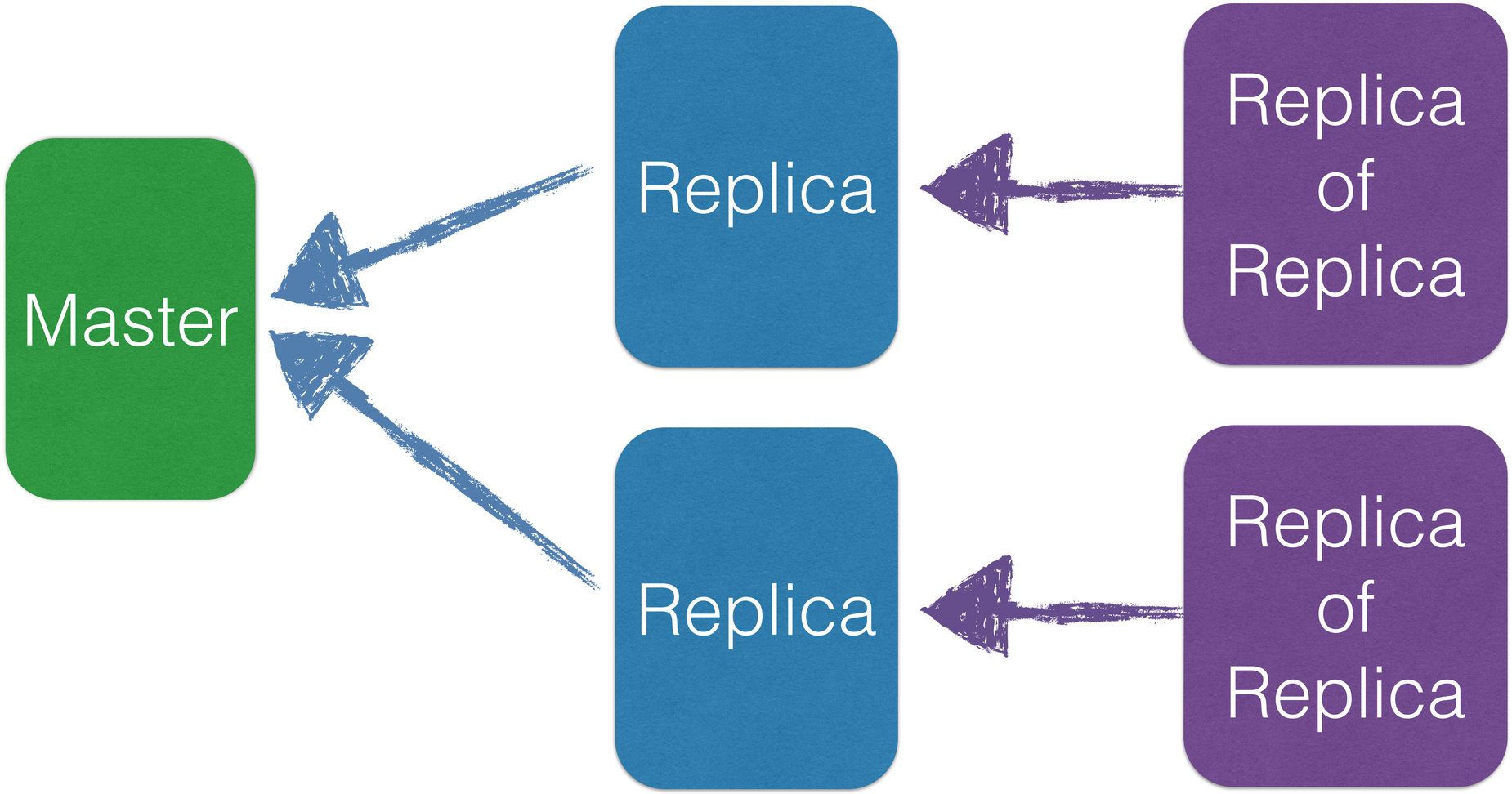
`redis-server`

lonely |ˈlōnlē| adjective (**lonelier** , **loneliest**)
sad because one has no friends or company

```
matt@ununoctium:/Volumes/matt/repos/redis/src% ./redis-server
[17997] 25 Nov 19:13:00.937 # Warning: no config file specified, using the default config. In order to
specify a config file use ./redis-server /path/to/redis.conf
[17997] 25 Nov 19:13:00.938 * Max number of open files set to 10032
                _.-
            _.-``__ ''-._
       _.-``    `.  `_.  ''-._           Redis 2.9.11 (6f4fd557/0) 64 bit
   .-`` .-```.  ```\/    _.,_ ''-._
  (    '      ,       .-`  | `,    )     Running in stand alone mode
  |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6379
  |    `-._   `._    /     _.-'    |     PID: 17997
   `-._    `-._  `-./  _.-'    _.-'
  |`-._`-._    `-.__.-'    _.-'_.-'|
  |    `-._`-._        _.-'_.-'    |           http://redis.io
   `-._    `-._`-.__.-'_.-'    _.-'
  |`-._`-._    `-.__.-'    _.-'_.-'|
  |    `-._`-._        _.-'_.-'    |
   `-._    `-._`-.__.-'_.-'    _.-'
       `-._    `-.__.-'    _.-'
           `-._        _.-'
              `-.__.-'

[17997] 25 Nov 19:13:00.979 # Server started, Redis version 2.9.11
[17997] 25 Nov 19:13:00.979 * The server is now ready to accept connections on port 6379
```

Replication



Replication

On Replica

```
127.0.0.1:3700> KEYS *
(empty list or set)
127.0.0.1:3700> SLAVEOF 127.0.0.1 6379
OK
```

Replication

On Replica

```
127.0.0.1:3700> KEYS *
 1) "d"
 2) "abcdef"
 3) "a"
 4) "purpleCircle"
 5) "c"
 6) "f"
 7) "location:kitten"
 8) "user:matt"
 9) "b"
10) "abc"
11) "name"
```

Replication

On Replica

```
[17594] 25 Nov 16:45:58.495 # Server started, Redis version 2.9.11
[17594] 25 Nov 16:45:58.498 * The server is now ready to accept connections on port 3700
[17594] 25 Nov 16:46:32.946 * SLAVE OF 127.0.0.1:6379 enabled (user request)
[17594] 25 Nov 16:46:33.802 * Connecting to MASTER 127.0.0.1:6379
[17594] 25 Nov 16:46:33.802 * MASTER <-> SLAVE sync started
[17594] 25 Nov 16:46:33.802 * Non blocking connect for SYNC fired the event.
[17594] 25 Nov 16:46:33.802 * Master replied to PING, replication can continue...
[17594] 25 Nov 16:46:33.802 * Partial resynchronization not possible (no cached master)
[17594] 25 Nov 16:46:33.802 * Full resync from master: 2bc50a54a9a532c9be6193341e74ba2af718db73:1
[17594] 25 Nov 16:46:33.874 * MASTER <-> SLAVE sync: receiving 297 bytes from master
[17594] 25 Nov 16:46:33.883 * MASTER <-> SLAVE sync: Loading DB in memory
[17594] 25 Nov 16:46:33.897 * MASTER <-> SLAVE sync: Finished with success
```

Replication

On Master

```
[4752] 25 Nov 16:58:51.099 * Slave asks for synchronization
[4752] 25 Nov 16:58:51.099 * Full resync requested by slave.
[4752] 25 Nov 16:58:51.099 * Starting BGSAVE for SYNC
[4752] 25 Nov 16:58:51.099 * Background saving started by pid 17644
[17644] 25 Nov 16:58:51.159 * DB saved on disk
[4752] 25 Nov 16:58:51.173 * Background saving terminated with success
[4752] 25 Nov 16:58:51.185 * Synchronization with slave succeeded
```

Replication

On Replica

```
127.0.0.1:3700> SLAVEOF NO ONE
OK
```

Replication

On Replica

```
[17636] 25 Nov 16:55:44.413 * Caching the disconnected master state.
[17636] 25 Nov 16:55:44.413 * Discarding previously cached master state.
[17636] 25 Nov 16:55:44.413 * MASTER MODE enabled (user request)
```

SLAVEOF NO ONE = MASTER MODE

On Replica; Master Down Failed Replication

```
[17636] 25 Nov 16:59:08.900 * Caching the disconnected master state.
[17636] 25 Nov 16:59:09.244 * Connecting to MASTER 127.0.0.1:6379
[17636] 25 Nov 16:59:09.244 * MASTER <-> SLAVE sync started
[17636] 25 Nov 16:59:09.244 # Error condition on socket for SYNC: Connection refused
.
.
.
[17636] 25 Nov 16:59:50.559 * Connecting to MASTER 127.0.0.1:6379
[17636] 25 Nov 16:59:50.559 * MASTER <-> SLAVE sync started
[17636] 25 Nov 16:59:50.559 # Error condition on socket for SYNC: Connection refused
[17636] 25 Nov 16:59:51.567 * Connecting to MASTER 127.0.0.1:6379
[17636] 25 Nov 16:59:51.567 * MASTER <-> SLAVE sync started
[17636] 25 Nov 16:59:51.568 * Non blocking connect for SYNC fired the event.
[17636] 25 Nov 16:59:51.568 * Master replied to PING, replication can continue...
[17636] 25 Nov 16:59:51.568 * Trying a partial resynchronization (request
2bc50a54a9a532c9be6193341e74ba2af718db73:1052).
[17636] 25 Nov 16:59:51.568 * Full resync from master: 48e4a31e11913b52b6f6816b116e16774cac3e7e:1
[17636] 25 Nov 16:59:51.568 * Discarding previously cached master state.
[17636] 25 Nov 16:59:51.675 * MASTER <-> SLAVE sync: receiving 297 bytes from master
[17636] 25 Nov 16:59:51.687 * MASTER <-> SLAVE sync: Loading DB in memory
[17636] 25 Nov 16:59:51.698 * MASTER <-> SLAVE sync: Finished with success
```

Replication Management
Sentinel

Beta since June 2012, rewritten November 2013.

Manages redis replication and availability with redis.

Provides auto-promotion of replicas.
Provides a service where you ask for the current redis master servers.

```
SENTINEL GET-MASTER-ADDR-BY-NAME userDB
```

Replication Management
Sentinel

Alternative to hard-coding database IPs in your config files.

Just ask Redis Sentinel for the current write master address.

Sentinel auto-notifies all clients about replica promotions to master.

Direct knowledge of DB state.

No waiting for timeouts or load balancers to switch over.

Replication

That's it for replication.

Replicas can replicate other replicas.

The replica instance stays in-sync (async) with its master node.

The replica is an exact copy of its master as long as `slave-read-only yes`

Clustering

Almost ready.

Under development for three years.

Distributes keys across master instances.

Each master instance has multiple identical replicas.

Replicas sanely promote to master if failure is detected.

Capacity Planning

(and capacity-related failure scenarios)

memory

network

cpu

disk

Capacity Planning

memory

牛

COW

Copy On Write

Capacity Planning

memory 牛

during **BGSAVE** DB write/update operations copy their original values to **BGSAVE** memory

64 GB RAM DB

references all data at time of **BGSAVE**

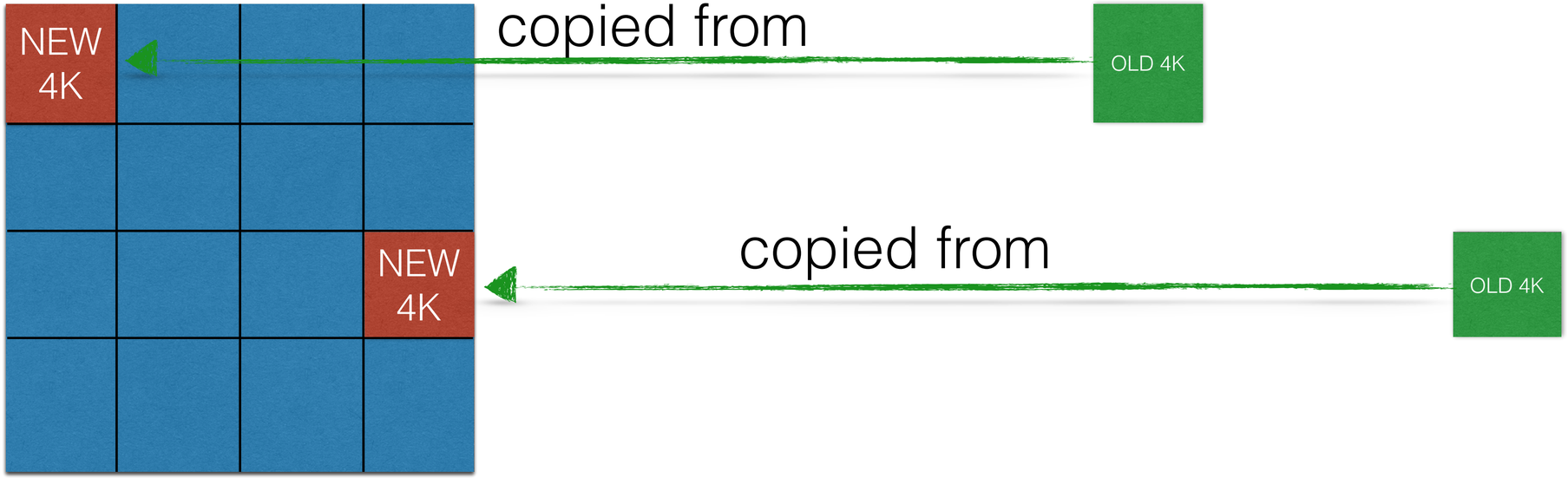
serializes memory to disk

uses negligible space since it only *references* original memory

BGSAVE

Capacity Planning

memory 牛



Something writes 4K to DB during **BGSAVE**

Capacity Planning

memory 牛

Implications

OS must enable memory overcommit

64 GB usage + 64 GB fork != 128 GB physical usage

```
[5358] 14 Nov 11:25:09.466 # WARNING overcommit_memory is set to 0! Background save may fail
under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf
and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
```

Capacity Planning

memory 牛

Implications

High-write DBs near memory limits can break

64 GB usage + 1 GB/sec updates + 2 sec **`BGSAVE`**

64 GB memory + 2 GB  = 66 GB = OOM Killer

Capacity Planning

memory 牛

Note

Xen has horrible forking performance.

Hardware fork latency: 80ms
VMware fork latency: 77ms
Xen fork latency: 1460ms

Capacity Planning

memory 牛

Note

A process is blocked until `fork` returns.

Normally a tiny less-than-100ms hiccup.

On Xen, you can notice a multiple second “outage.”

Capacity Planning

network

Recommendations

Have a network.

Know where you are in your network.

Try to at least be in the same building as your DBs.

Capacity Planning

network

Protocol

```
*[argument count]\r\n
$[byte count of argument]\r\n
[data]\r\n
.
.
$[byte count of last argument]\r\n
[data]\r\n
```

Capacity Planning

network

Protocol

```
SET name Matt
```

```
*3
$3
SET
$4
name
$4
Matt
```

Capacity Planning

network

Protocol

```
SET name Matt
```

```
*3\r\n$3\r\nSET\r\n$4\r\nname\r\n$4\r\nMatt\r\n
```

33 bytes

Capacity Planning

network

Protocol

```
SADD names Matt
SADD names GoPivotal
SADD names Ireland
SADD names Barcelona
```

Capacity Planning

```
SADD names Matt
SADD names GoPivotal
SADD names Ireland
SADD names Barcelona
```

network

Protocol

```
*3\r\n$4\r\nSADD\r\n$5\r\nnames\r\n$4\r\nMatt\r\n
*3\r\n$4\r\nSADD\r\n$5\r\nnames\r\n$9\r\nGoPivotal\r\n
*3\r\n$4\r\nSADD\r\n$5\r\nnames\r\n$7\r\nIreland\r\n
*3\r\n$4\r\nSADD\r\n$5\r\nnames\r\n$9\r\nBarcelona\r\n
```

34 bytes

39 bytes

37 bytes

39 bytes

149 bytes total

Capacity Planning

```
SADD names Matt
SADD names GoPivotal
SADD names Ireland
SADD names Barcelona
```

network

Protocol

```
SADD names Matt GoPivotal Ireland Barcelona
```

```
*6\r\n$4\r\nSADD\r\n$5\r\nnames\r\n$4\r\nMatt\r\n$9\r
\nGoPivotal\r\n$7\r\nIreland\r\n$9\r\nBarcelona\r\n
```

78 bytes total

Capacity Planning

network Protocol

```
SADD names Matt
SADD names GoPivotal
SADD names Ireland
SADD names Barcelona
```

VS.

```
SADD names Matt GoPivotal Ireland Barcelona
```

149 bytes total VS. 78 bytes total

55% less network traffic

Capacity Planning

Pipelining

network

Protocol

```
SISMEMBER names Matt
1

SISMEMBER names GoPivotal
1

SISMEMBER names Croatia
0

SISMEMBER names Barcelona
1
```

8 network round trips

Capacity Planning

Pipelining

network

Protocol

```
SISMEMBER names Matt
SISMEMBER names GoPivotal
SISMEMBER names Croatia
SISMEMBER names Barcelona
```

```
1
1
0
1
```

results returned in execution order

2 network round trips

Capacity Planning

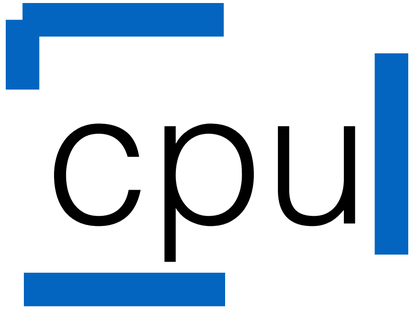
cpu

Redis uses one thread for data manipulation.

Redis uses one thread for background **`AOF`** writes.

Redis forks for **`BGSAVE`**.

Capacity Planning

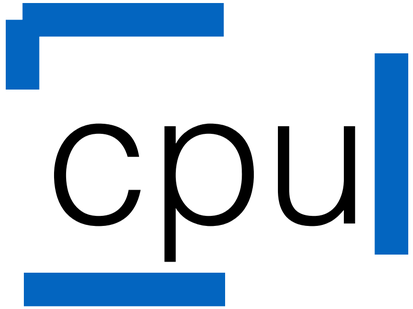


Run one redis-server instance per core.

Leave a few cores free for **`AOF`** and **`BGSAVE`** scheduling.

Cluster will make running multiple instances per host simpler.

Capacity Planning



Each Redis command has individual performance characteristics.

The Big-Oh of each command is listed in the documentation.

Capacity Planning

cpu

Redis provides excellent performance and latency for:

1) O(1) operations.
2) Logarithmic operations (most basic sorted sets operations, including ZRANK).
3) O(N) seek + O(M) work (for example LTRIM) every time you can make sure to take M small. Example: capped collections implementation.
4) Log(N) seek + O(M) work (for example removing ranges of elements from a sorted set).

Capacity Planning

:disk:

Disks are useful for saving data and storing configs.

`AOF` and `BGSAVE` and `redis.conf` and `sentinel.conf` and `cluster.conf`

Capacity Planning

disk

Redis needs to own a directory with write access.

The `dir` config parameter.

Redis saves `AOF`/`BGSAVE`/cluster state to the `dir`.

How Redis fail?

Failure Scenarios

Failure Scenarios

memory corruption

configuration not matching expectations

coding errors

users abusing the DB

Failure Scenarios

the error report

```
=== REDIS BUG REPORT START: Cut & paste starting from here ===
[2285] 04 Nov 15:19:02.148 # Redis 2.7.104 crashed by signal: 11
[2285] 04 Nov 15:19:02.148 # Failed assertion: (:0)
[2285] 04 Nov 15:19:02.148 # --- STACK TRACE
redis-rdb-bgsave *:6379(logStackTrace+0x3e)[0x443c1e]
redis-rdb-bgsave *:6379(rdbSaveStringObject+0x0)[0x42c2d0]
.
.
.
=== REDIS BUG REPORT END. Make sure to include from START to END. ===
```

Failure Scenarios

the error report

sections:
server
clients
memory
persistence
stats
replication

CPU
command stats
cluster
keyspace
list of clients
list of client state
register contents

Failure Scenarios

memory corruption

cause:
not using ECC memory

these days nobody knows
what hardware they use.

Failure Scenarios

memory corruption

symptoms:
unexplained segfaults
gibberish in error reports

usually at the same time

Failure Scenari

ory

what's that?

not even
regul

what's that?

```
used_memory:18
used_memory
used_memo                    3835776
used_m                       1877760896
used                         k_human:11.06G
u                            lua:47104
                             ntation_ratio:0.00
                             ocator:libc
```

memory corruption

Failure Scenarios

memory corruption

actions:

If Redis crashes and memory is suspected, without rebooting after the crash,

`redis-server —test-memory [MB]`

`

memory corruption

example:

```
redis-server —test-memory 16384
```

Failure Scenarios

memory corruption

check **`dmesg`** for memory errors

if available, check **`ipmi`** output:

`ipmitool sel list`

```
Memory #0x01 | Correctable ECC
Memory #0x01 | Correctable ECC
Memory #0x01 | Correctable ECC
Memory #0x01 | Correctable ECC
Memory #0x02 | Uncorrectable ECC
```

Failure Scenarios

memory corruption

better test:
if user has hardware access, run MemTest86

pre-boot memory testing

http://www.memtest86.com/

Failure Scenarios

memory corruption

Most reported Redis crashes are due to hardware memory corruption.

Always recommend running with ECC memory.

Failure Scenarios
persistence problems

`BGSAVE` degrading performance

`AOF` corruption

Failure Scenarios

persistence problems

`BGSAVE` = binary, compressed, DB snapshot

`AOF` = command-by-command DB reconstruction

Failure Scenarios

persistence problems

`BGSAVE` = **`fork()`** latency varies by platform.

hardware = 10ms per GB.

xen = 250ms per GB.

`AOF` = flushes to disk every second

(or always or never)

Failure Scenarios
persistence problems

fork latency reported as:
`latest_fork_usec` in `INFO` output

Failure Scenarios
persistence problems

if AOF gets corrupted, you can edit it

remove the end-of-file corruption
(lose or recover the last key)

Failure Scenarios

persistence problems

`BGSAVE` = faster restarts
(writes DB to disk based on usage)

`AOF` = more up-to-date persistence
(flushes to disk once a second)

Failure Scenarios
configuration delusions

Redis is telling me I can't write.

Did `BGSAVE` fail while you're running with `stop-writes-on-bgsave-error yes`

Failure Scenarios

configuration delusions

Redis is telling me I can’t write.

Are you running with disconnected replicas and these configs set

```
min-slaves-to-write 3
min-slaves-max-lag 10
```

Failure Scenarios
configuration delusions

Redis is telling me I can’t write.

Did you overrun your memory limit?

```
maxmemory 16GB
```

Failure Scenarios
configuration delusions

Redis is telling me I can’t connect.

Did you overrun your client limit?

```
maxclients 10000
```

Failure Scenarios
configuration delusions

Redis is telling me I can’t read.

Are you reading a disconnected replica with `slave-serve-stale-data no`

Failure Scenarios
users abusing the DB

Redis is intermittently stalled.

```
CONFIG SET slowlog-log-slower-than [µs]
CONFIG SET slowlog-max-len [entries]

SLOWLOG GET 10
SLOWLOG RESET
```

Failure Scenarios

users abusing the DB

Redis is stalled.

Did someone run **`KEYS *`** on a large DB?

(**`KEYS`** is deprecated in favor of the Redis 2.8 **`SCAN`** interface)

Failure Scenarios

users abusing the DB

Redis is stalled.

Is a poorly behaved script running?

`lua-time-limit 5000`

Failure Scenarios

users abusing the DB

`lua-time-limit` does **not** kill scripts.

After `lua-time-limit` is reached,
the server accepts commands again, but only
allows `SCRIPT KILL` or `SHUTDOWN NOSAVE`

Failure Scenarios
users abusing the DB

Watch out for latency complaints when Redis is paging out to swap space.

Failure Scenarios
users abusing the DB

For serious, production-level DB usage:

Run dedicated DB hardware with swap off.

Failure Scenarios

users abusing the DB

Check how much of Redis is paged to disk:

```
cat /proc/<pid of redis-server>/smaps
```

Failure Scenarios

users abusing the DB

Servers should not swap.

Servers should not have swap enabled.

If your server is mis-sized,
you deserve the OOM Killer.

How Redis debug?

Automated data collection

Debug Info

If you failed, you have an error report.

Debug Info

```
=== REDIS BUG REPORT START: Cut & paste starting from here ===
[2285] 04 Nov 15:19:02.148 # Redis 2.7.104 crashed by signal: 11
[2285] 04 Nov 15:19:02.148 # Failed assertion: (:0)
[2285] 04 Nov 15:19:02.148 # --- STACK TRACE
redis-rdb-bgsave *:6379(logStackTrace+0x3e)[0x443c1e]
redis-rdb-bgsave *:6379(rdbSaveStringObject+0x0)[0x42c2d0]
.
.
.
=== REDIS BUG REPORT END. Make sure to include from START to END. ===
```

Debug Info

If you aren’t failed, grab `INFO`

Debug Info

```
[::1]:4003> INFO
# Server
redis_version:2.9.11
redis_git_sha1:6f4fd557
.
.
.
# CPU
used_cpu_sys:3.49
used_cpu_user:2.65
used_cpu_sys_children:0.00
used_cpu_user_children:0.00

# Keyspace
db0:keys=12,expires=0,avg_ttl=0
```

Debug Info

Quick Overview of `INFO` Fields

Debug Info

Sections

- `server`: General information about the Redis server
- `clients`: Client connections section
- `memory`: Memory consumption related information
- `persistence`: RDB and AOF related information
- `stats`: General statistics
- `replication`: Master/slave replication information
- `cpu`: CPU consumption statistics
- `commandstats`: Redis command statistics
- `cluster`: Redis Cluster section
- `keyspace`: Database related statistics

Debug Info

Server

- `redis_version`: Version of the Redis server
- `redis_git_sha1`: Git SHA1
- `redis_git_dirty`: Git dirty flag
- `os`: Operating system hosting the Redis server
- `arch_bits`: Architecture (32 or 64 bits)
- `multiplexing_api`: event loop mechanism used by Redis
- `gcc_version`: Version of the GCC compiler used to compile the Redis server
- `process_id`: PID of the server process
- `run_id`: Random value identifying the Redis server (to be used by Sentinel and Cluster)
- `tcp_port`: TCP/IP listen port
- `uptime_in_seconds`: Number of seconds since Redis server start
- `uptime_in_days`: Same value expressed in days
- `lru_clock`: Clock incrementing every minute, for LRU management

Debug Info

Clients

- `connected_clients`: Number of client connections (excluding connections from slaves)
- `client_longest_output_list`: longest output list among current client connections
- `client_biggest_input_buf`: biggest input buffer among current client connections
- `blocked_clients`: Number of clients pending on a blocking call (BLPOP, BRPOP, BRPOPLPUSH)

Debug Info

Memory

- `used_memory`: total number of bytes allocated by Redis using its allocator (either standard **libc**, **jemalloc**, or an alternative allocator such as **tcmalloc**)
- `used_memory_human`: Human readable representation of previous value
- `used_memory_rss`: Number of bytes that Redis allocated as seen by the operating system (a.k.a resident set size). This is the number reported by tools such as **top** and **ps**.
- `used_memory_peak`: Peak memory consumed by Redis (in bytes)
- `used_memory_peak_human`: Human readable representation of previous value
- `used_memory_lua`: Number of bytes used by the Lua engine
- `mem_fragmentation_ratio`: Ratio between `used_memory_rss` and `used_memory`
- `mem_allocator`: Memory allocator, chosen at compile time.

Debug Info

Persistence I (General)

- `loading`: Flag indicating if the load of a dump file is on-going
- `rdb_changes_since_last_save`: Number of changes since the last dump
- `rdb_bgsave_in_progress`: Flag indicating a RDB save is on-going
- `rdb_last_save_time`: Epoch-based timestamp of last successful RDB save
- `rdb_last_bgsave_status`: Status of the last RDB save operation
- `rdb_last_bgsave_time_sec`: Duration of the last RDB save operation in seconds
- `rdb_current_bgsave_time_sec`: Duration of the on-going RDB save operation if any
- `aof_enabled`: Flag indicating AOF logging is activated
- `aof_rewrite_in_progress`: Flag indicating a AOF rewrite operation is on-going
- `aof_rewrite_scheduled`: Flag indicating an AOF rewrite operation will be scheduled once the on-going RDB save is complete.
- `aof_last_rewrite_time_sec`: Duration of the last AOF rewrite operation in seconds
- `aof_current_rewrite_time_sec`: Duration of the on-going AOF rewrite operation if any
- `aof_last_bgrewrite_status`: Status of the last AOF rewrite operation

Debug Info

Persistence II (AOF)

- `aof_current_size`: AOF current file size
- `aof_base_size`: AOF file size on latest startup or rewrite
- `aof_pending_rewrite`: Flag indicating an AOF rewrite operation will be scheduled once the on-going RDB save is complete.
- `aof_buffer_length`: Size of the AOF buffer
- `aof_rewrite_buffer_length`: Size of the AOF rewrite buffer
- `aof_pending_bio_fsync`: Number of fsync pending jobs in background I/O queue
- `aof_delayed_fsync`: Delayed fsync counter

Debug Info

Persistence III (Loading)

- `loading_start_time`: Epoch-based timestamp of the start of the load operation
- `loading_total_bytes`: Total file size
- `loading_loaded_bytes`: Number of bytes already loaded
- `loading_loaded_perc`: Same value expressed as a percentage
- `loading_eta_seconds`: ETA in seconds for the load to be complete

Debug Info

Stats

- `total_connections_received`: Total number of connections accepted by the server
- `total_commands_processed`: Total number of commands processed by the server
- `instantaneous_ops_per_sec`: Number of commands processed per second
- `rejected_connections`: Number of connections rejected because of maxclients limit
- `expired_keys`: Total number of key expiration events
- `evicted_keys`: Number of evicted keys due to maxmemory limit
- `keyspace_hits`: Number of successful lookup of keys in the main dictionary
- `keyspace_misses`: Number of failed lookup of keys in the main dictionary
- `pubsub_channels`: Global number of pub/sub channels with client subscriptions
- `pubsub_patterns`: Global number of pub/sub pattern with client subscriptions
- `latest_fork_usec`: Duration of the latest fork operation in microseconds

Debug Info

Replication

- `role`: Value is "master" if the instance is slave of no one, or "slave" if the instance is enslaved to a master. Note that a slave can be master of another slave (daisy chaining).
- `connected_slaves`: Number of connected slaves

Debug Info

Replication (if replica/slave)

- `master_host`: Host or IP address of the master
- `master_port`: Master listening TCP port
- `master_link_status`: Status of the link (up/down)
- `master_last_io_seconds_ago`: Number of seconds since the last interaction with master
- `master_sync_in_progress`: Indicate the master is SYNCing to the slave

Debug Info

Replication (if currently `SYNC`ing)

- `master_sync_left_bytes`: Number of bytes left before SYNCing is complete
- `master_sync_last_io_seconds_ago`: Number of seconds since last transfer I/O during a SYNC operation

Debug Info

Replication (if master unreachable)

- `master_link_down_since_seconds`: Number of seconds since the link is down

Debug Info

CPU

- `used_cpu_sys`: System CPU consumed by the Redis server
- `used_cpu_user`:User CPU consumed by the Redis server
- `used_cpu_sys_children`: System CPU consumed by the background processes
- `used_cpu_user_children`: User CPU consumed by the background processes

Debug Info

Commandstats (for each command)

- `cmdstat_XXX:calls=XXX,usec=XXX,usec_per_call=XXX`

Debug Info

Keyspace

For each database, the following line is added:

- dbXXX:keys=XXX,expires=XXX

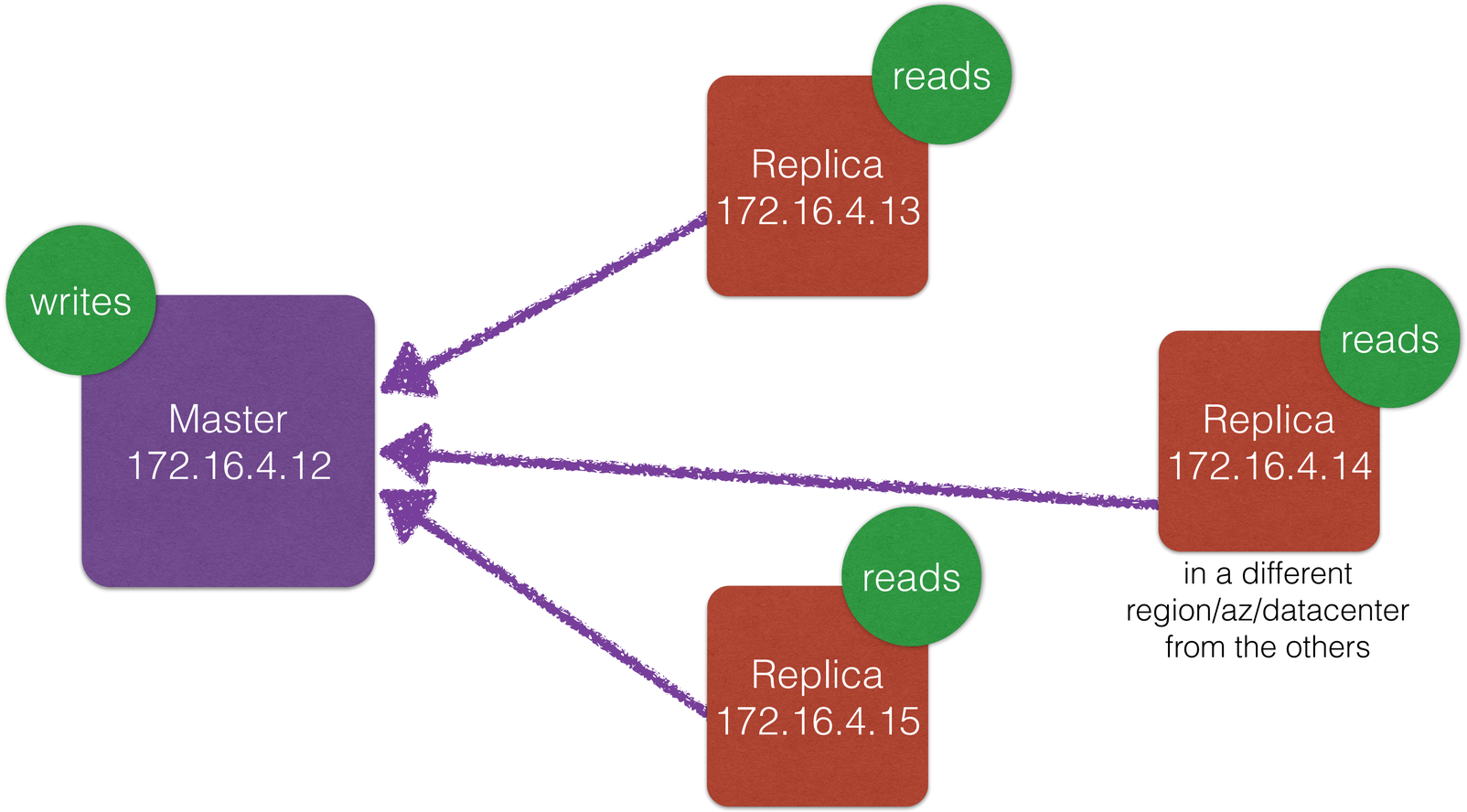
Debug Info

Details about each `INFO` field:
`http://redis.io/commands/info`

Manual data collection

Manual Data Collection

Node topology



Manual Data Collection

Node Info

Make pictures

Config file from each instance

`INFO` or error report from each instance

Manual Data Collection

Client Info

Programming language version

Redis client and its version

Direct communication to Redis?
Using a proxy? Sharding? Hashing?

Manual Data Collection

System Info

OS vendor with version

Memory information

```
Linux: vmstat -SM 1 10; vmstat -SM -s;
free -m; top -b -d 3 -n 3 -M -m; df -h; iostat -k 1 12
OS X: memory_pressure; vm_stat -c 10 1;
top -o rsize -i 3 -l 3; df -h; iostat -c 12
```

Manual Data Collection

Network Info

On-site vs. hosted vs. VPS?

```
ping -c 20 [redis server]
redis-cli --latency -h [host] -p [redis port]
```

Any VPN or ssh tunneling?

(including, but not limited to: stunnel, stud, spiped)

Figure it out

Figure it out

http://redis.io/documentation

Explains specific Redis details
with proper usage examples
(and fixes)

Figure it out

http://redis.io/commands

Detailed, accurate

Notes changes across versions

Figure it out

Ask Google

Figure it out

Ask Us

redis