An Introduction to Redis for .NET Developers

Presented By: Steve Lorello @slorello

Steve Lorello
Developer Advocate
@Redis

@slorello
github.com/slorello89

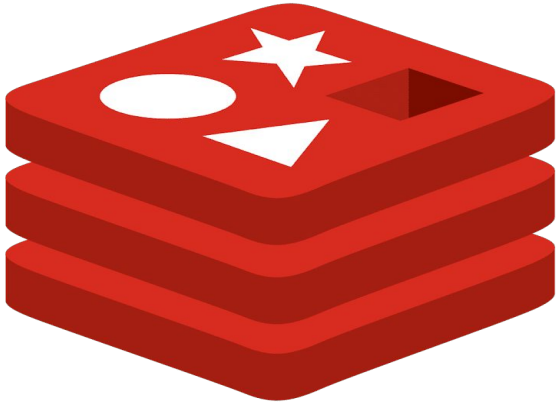
Agenda

- What is Redis?
- Using Redis in your app
- Major Redis Use Cases
- Redis Data Structures
- Redis Design Patterns
- Redis Gotchas and Anti-Patterns

Ecosystem Neutrality and Clients

- 80% ecosystem neutral
- 20% .NET focused
- StackExchange.Redis - Canonical Client

What is redis?



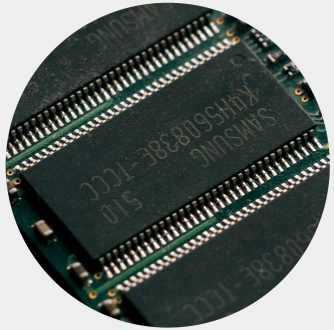
What is Redis?

- REmote DIctionary Server

- Created by Antirez

What is Redis?

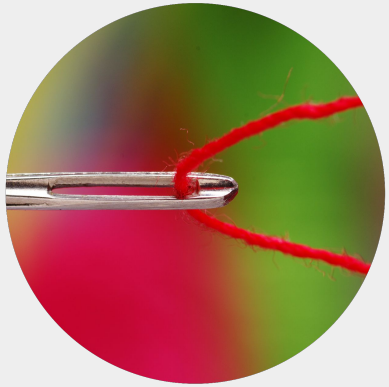
- Completely In Memory



- Key-Value Data Structure Store

What is Redis?

- Redis is Single Threaded



- Blazingly Fast NoSQL DB

What is Redis?

- Easy to Use

- Beloved by Developers

Redis and ACID

ACID in Redis - Atomicity

- Individual Redis commands are completely atomic
- ‘Transactions’ & Scripting for grouping commands

ACID in Redis - Consistency

- Depends on deployment and config
- Single instance always consistent
- Read-only replication guaranteed eventual consistency
  - Forced consistency with “WAIT” command

ACID in Redis - Isolation

- Single Threaded
- Isolated as there’s no concurrency

ACID in Redis - Durability

- Entire database loaded into memory
- Two durability models persistence to disk
- AOF (Append Only File)
- RDB (Redis Database File)

Durability - Append Only File (AOF)

- With each command Redis writes to AOF
- Reconstruct Redis from AOF
- AOF flush to disk is NOT necessarily synchronous
- FSYNC config policy determines level of durability
  - always - Synchronously flush to disk (fully durable)
  - everysec - Flush buffer to disk every second
  - no - OS decides when to write to disk

Durability - Redis Database file (RDB)

- RDBs are snapshots of the database
- Taken in intervals, or by command
- More compact and faster to ingest than AOF
- Less strain on OS than AOF
- Comes at cost of higher potential data loss

Connecting To Redis

ConnectionMultiplexer

- Heart of StackExchange.Redis
- Handles connections to Redis
- Multiplexes all Commands Sent to Redis
- Initialize 1 instance/app
    - Pass around via DI or can be used as a Singleton

Interfaces

- IConnectionMultiplexer
- IDatabase
- IServer
- ITransaction
- ISubscriber

Redis Deployment Models

Redis Standalone

Redis Standalone

- One Redis instance
- Simple
- No HA capabilities
- No Horizontal Scaling

Connecting

- Basic Connection String
- {host}:{port},password=password,ssl=sslSetting

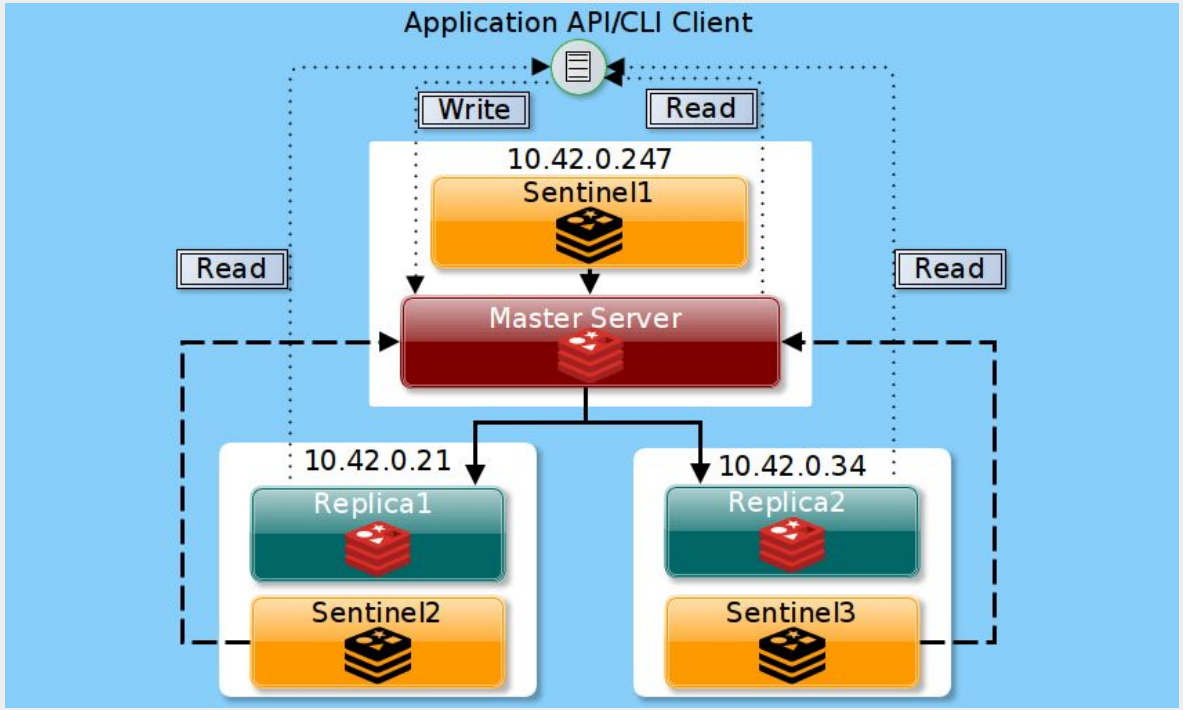
redis:6379,user=default,password=password,ssl=false

Redis Sentinel

Sentinel, Replication for High Availability

- Writable Master
- Read-only Replicas
- “Sentinels” to handle HA & Failover

Sentinel (Replication for High Availability)



Source: https://www.tecmint.com/setup-redis-high-availability-with-sentinel-in-centos-8/

Connection

- Connect to sentinel, not to a master
- Pass in serviceName
  - Signals that it’s a sentinel

```
sentinel:26379,servcieName=mySentinel,password=password
```

Redis Cluster

Redis Cluster - Horizontal Scaling - ‘Hard Mode’

- Keyspace split across multiple shards
- Deterministic hash function on key to determine ‘slot’
- Shards responsible for group of ‘slots’

Redis Cluster

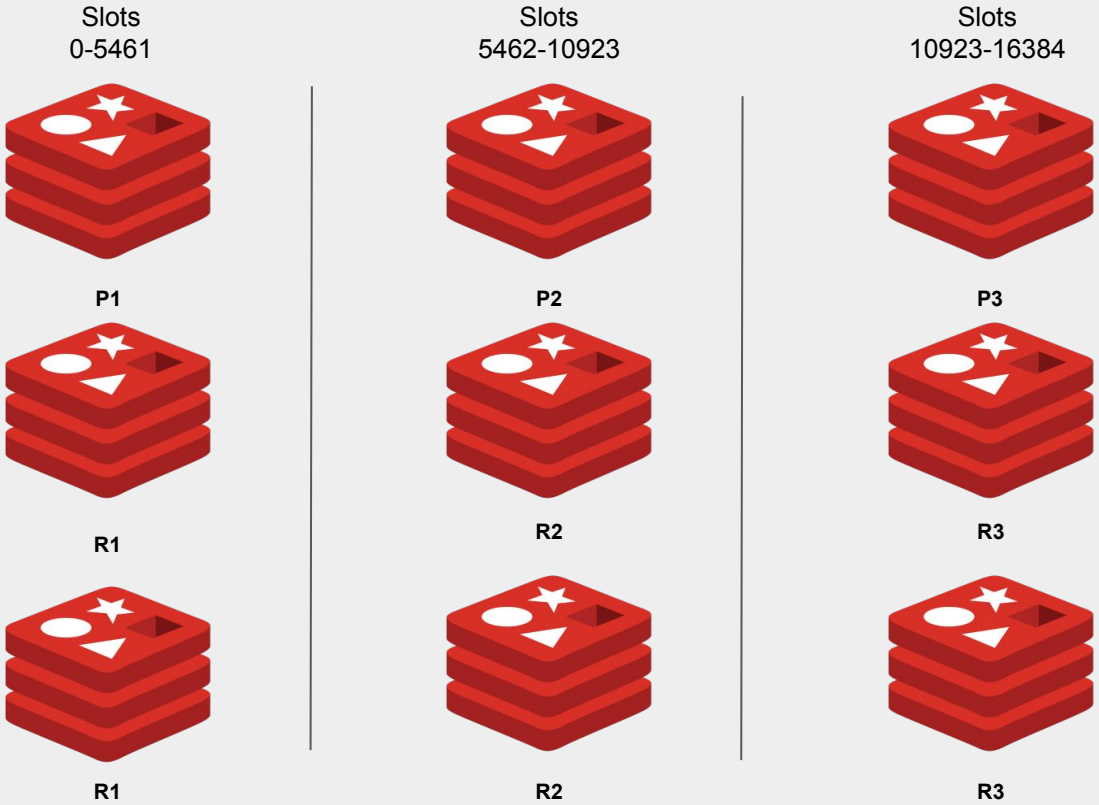
- Horizontally scale reads/writes across cluster
- Multi-key operations become slot limited

Connecting

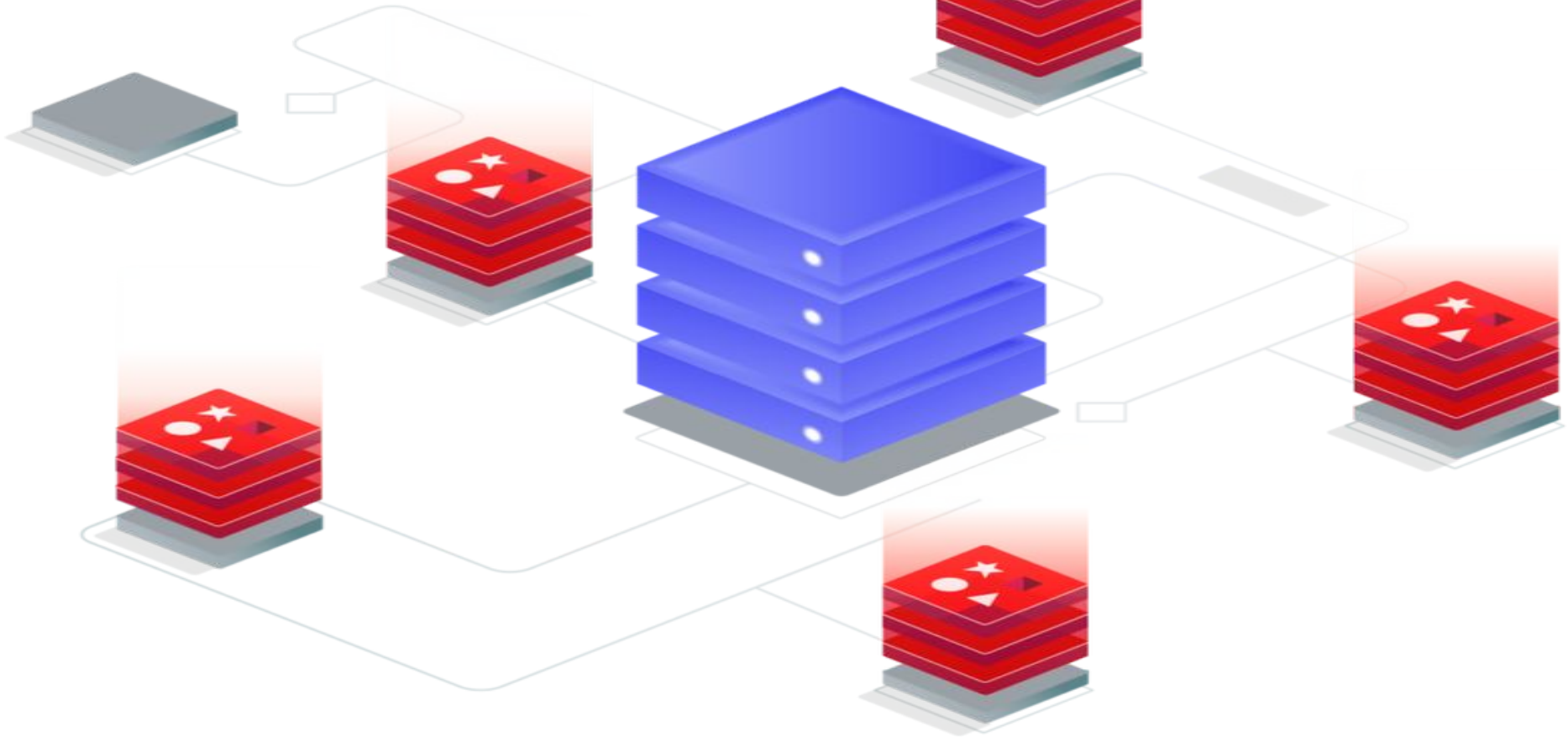
- Similar to Standalone
- You can pass in multiple endpoints
  - Helps in case of failovers

```
redis-1:6379,redis-2:6379,redis-3:6379,ssl=true,password=password
```

Redis Cluster



Redis Cloud



Redis Cloud - Infinite Scaling ‘Easy Mode’

- Have Redis manage your cluster
- Scales endlessly
- Zero out complexity of scaling Redis

Other Redis Cloud Features (pay the bills slide)

- Geo-Distribution with Active-Active
- Tremendous throughput
- Five-nines availability
- Multi-cloud product
- Access to Redis Modules
- Enterprise support

Redis Cloud - A Developer Advocate's Experience

- We run a discord server

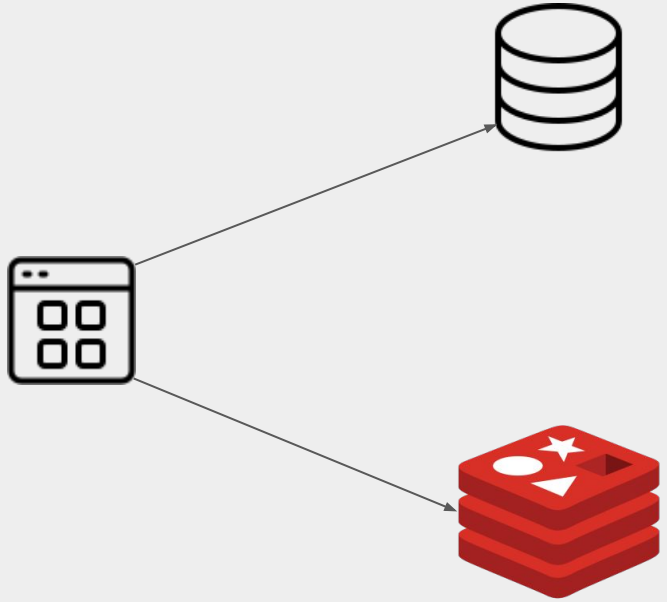
  Join us by the way! https://discord.gg/redis

- Field lots of complex questions about cluster/sentinel from the community
- Write up lots of complex answers with the typical refrain:
  - "Or you could just use Redis Cloud and you don't have to worry about any of this"

Uses of Redis

As a Cache

- #1 Most Popular Usage
- Low Latency Data Access
- For repeated operations



As a Session State Store

- Distributed Session State
- Low-Latency Access

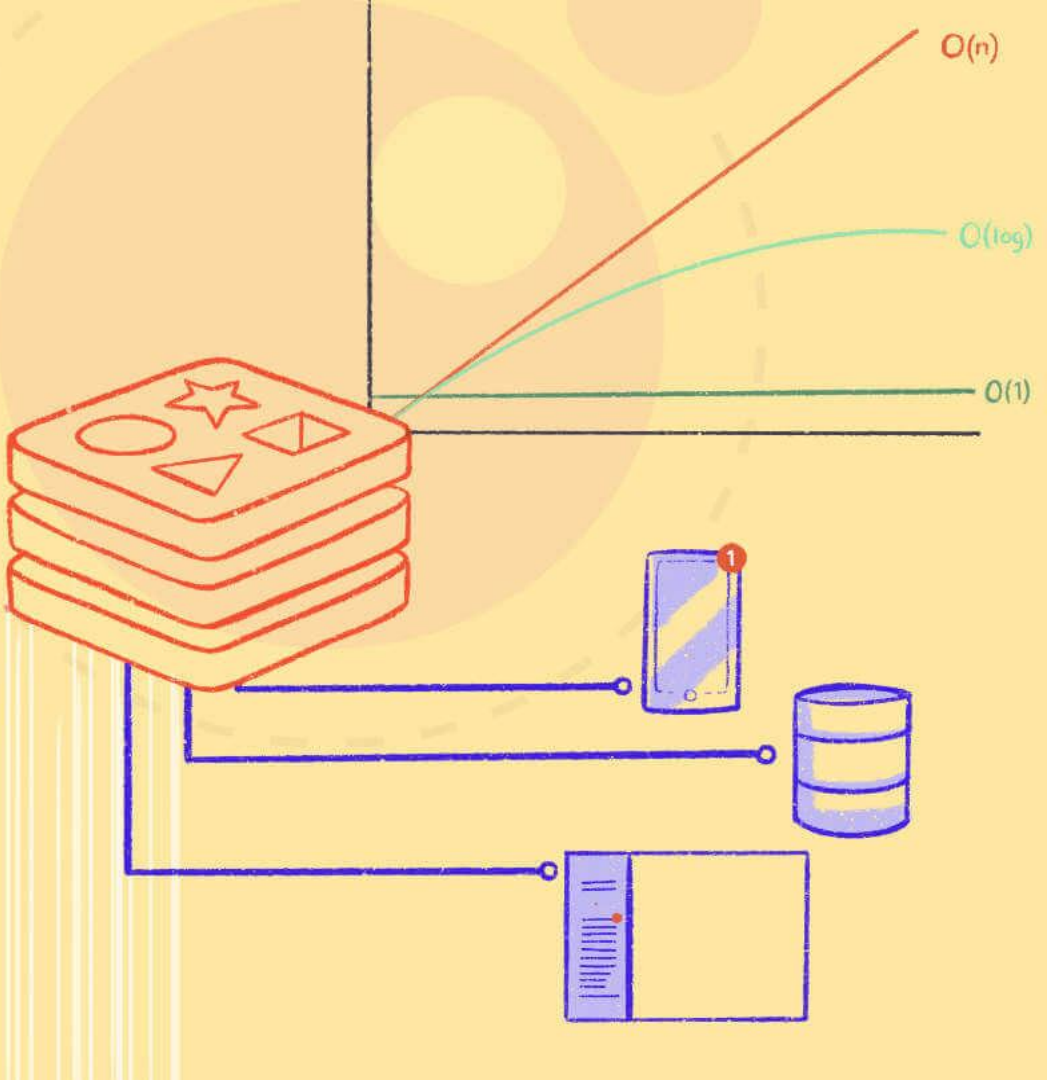
As a Message Bus

- Stream Messages Through Redis with Redis Streams
- Consume message with consumer groups
- “Using Redis Streams saved our company from Bankruptcy”

As a Database

- No extra layers of caching, just use cache as DB!
- Document Data Structures allow for CRUD operations
- Indexing Capabilities to easily find things

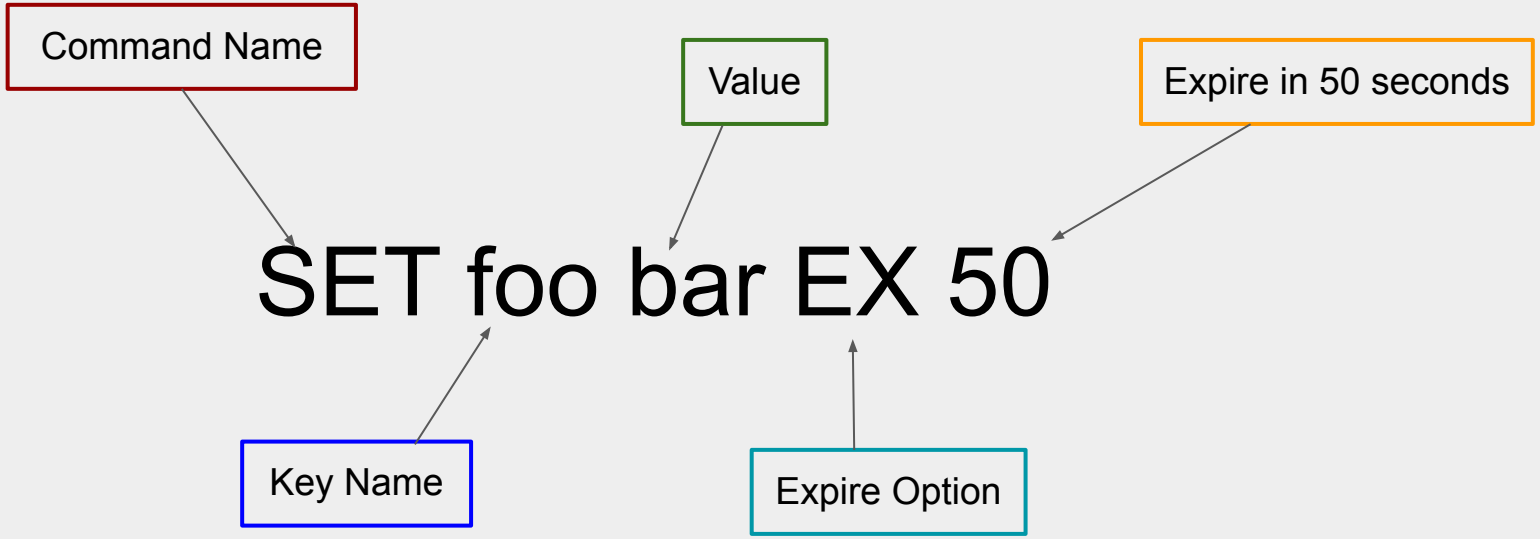
Redis Data Structures



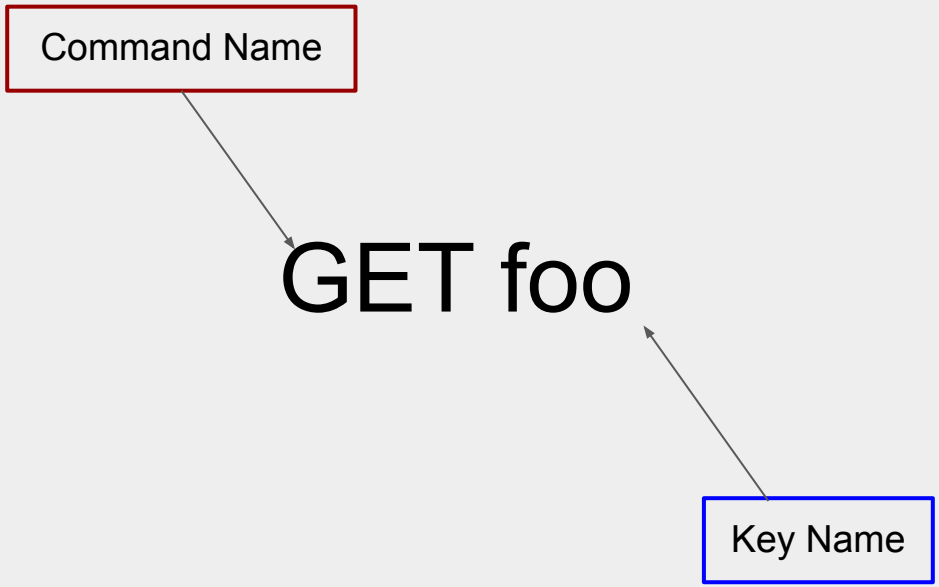
Strings

Strings

- Simplest Redis Data Structure
- Single Key -> Single Value
- Driven primarily by SET/GET like commands

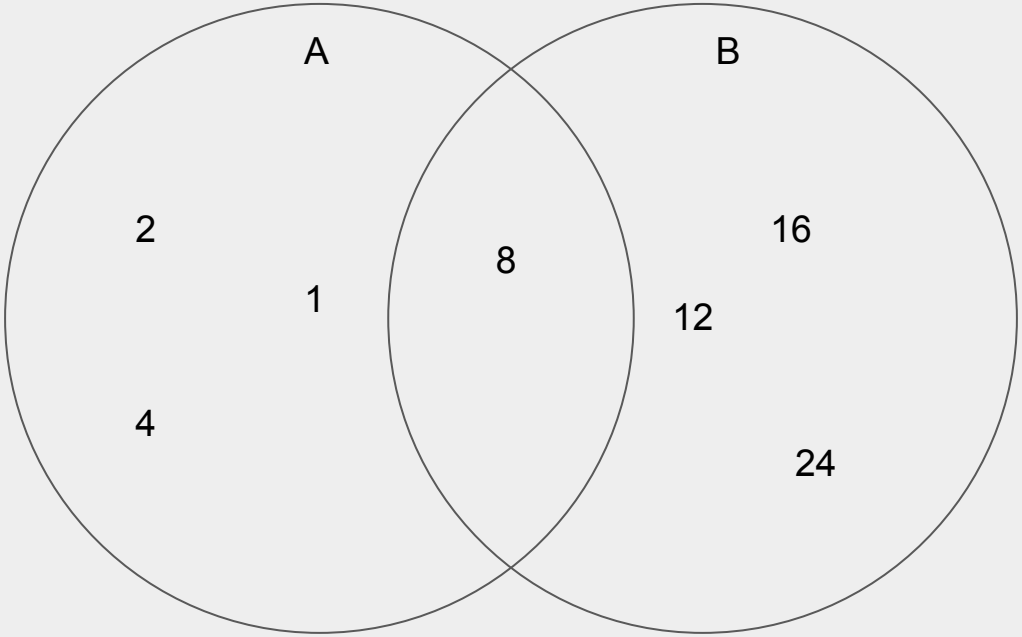



```
db.StringSet("foo", "bar", TimeSpan.FromSeconds(50));
```



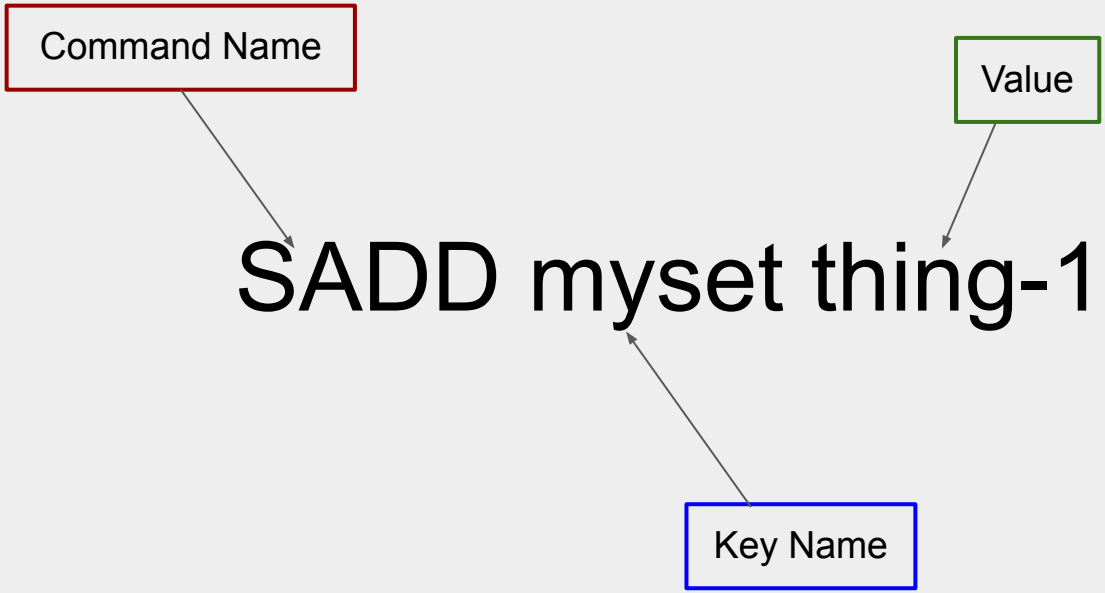

```
db.StringGet("foo");
```

Sets



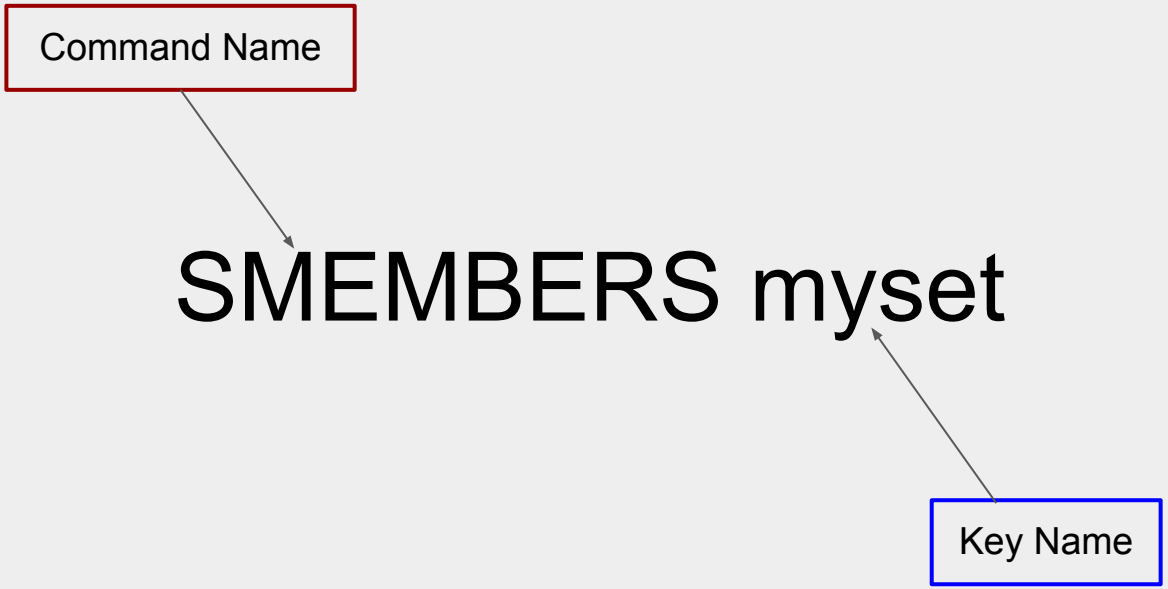
Sets

- Mathematical set
- Collection of Redis Strings
- Unordered
- No Duplication
- Set Combination Operations (Intersect, Union, Difference)

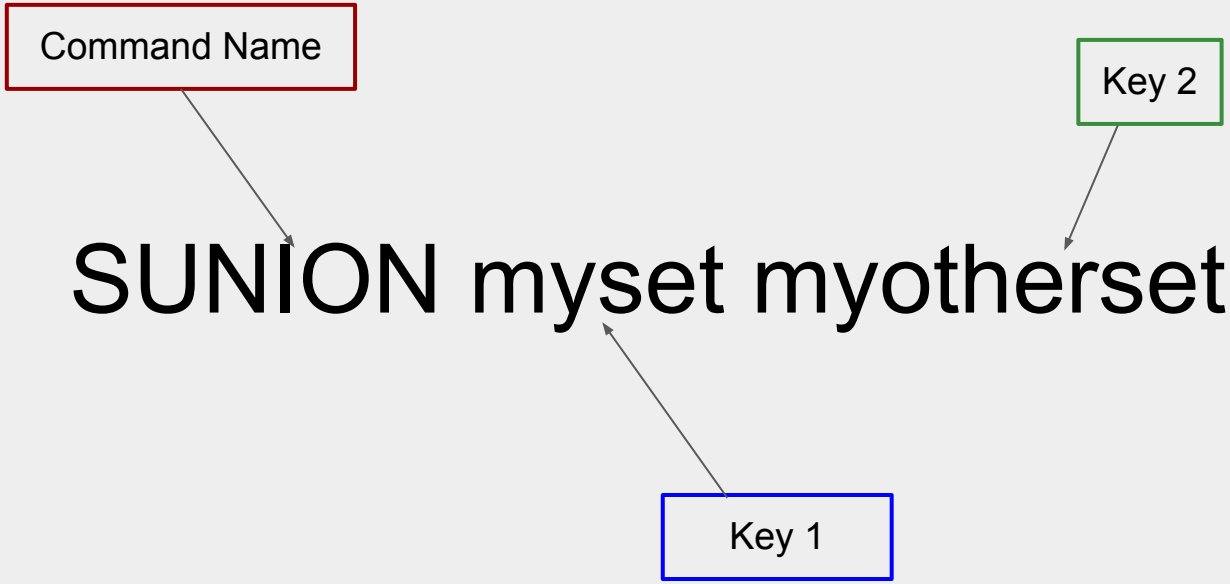



```
db.SetAdd("myset1", "thing-1");
```


```
db.SetContains("myset", "thing-1");
```



db.SetMembers("myset");



db.SetCombine(SetOperation.Union,"myset","myset2");

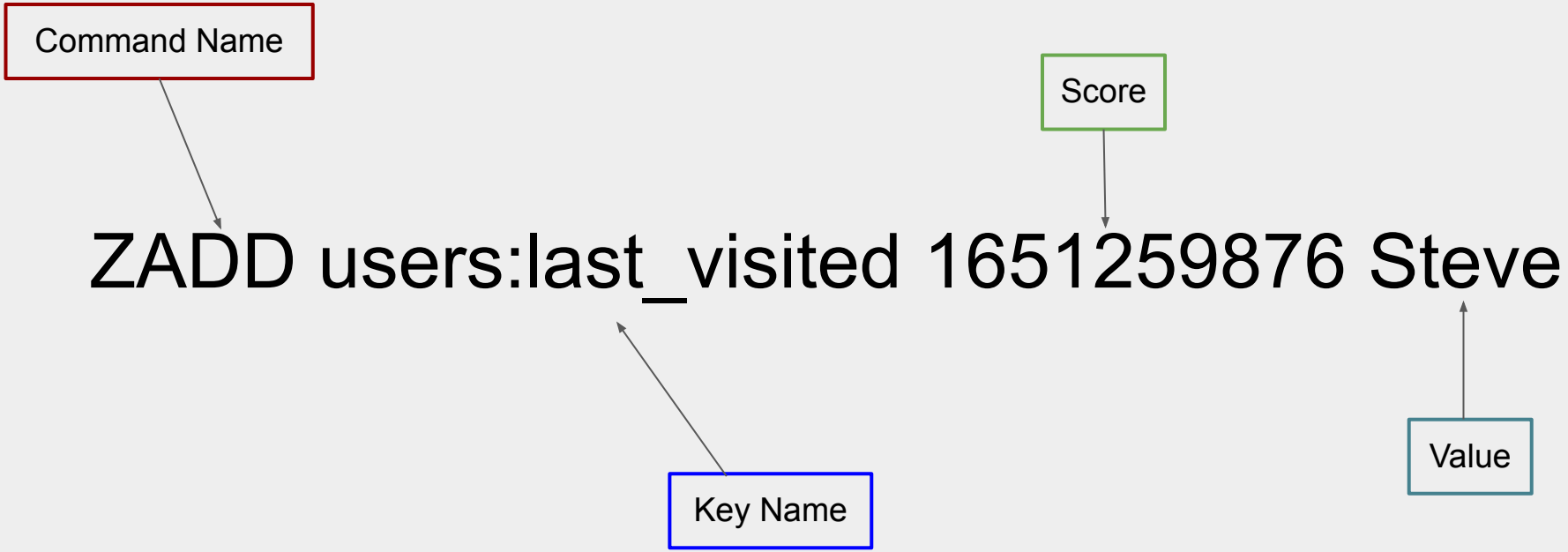
Sorted Sets

Sorted Sets

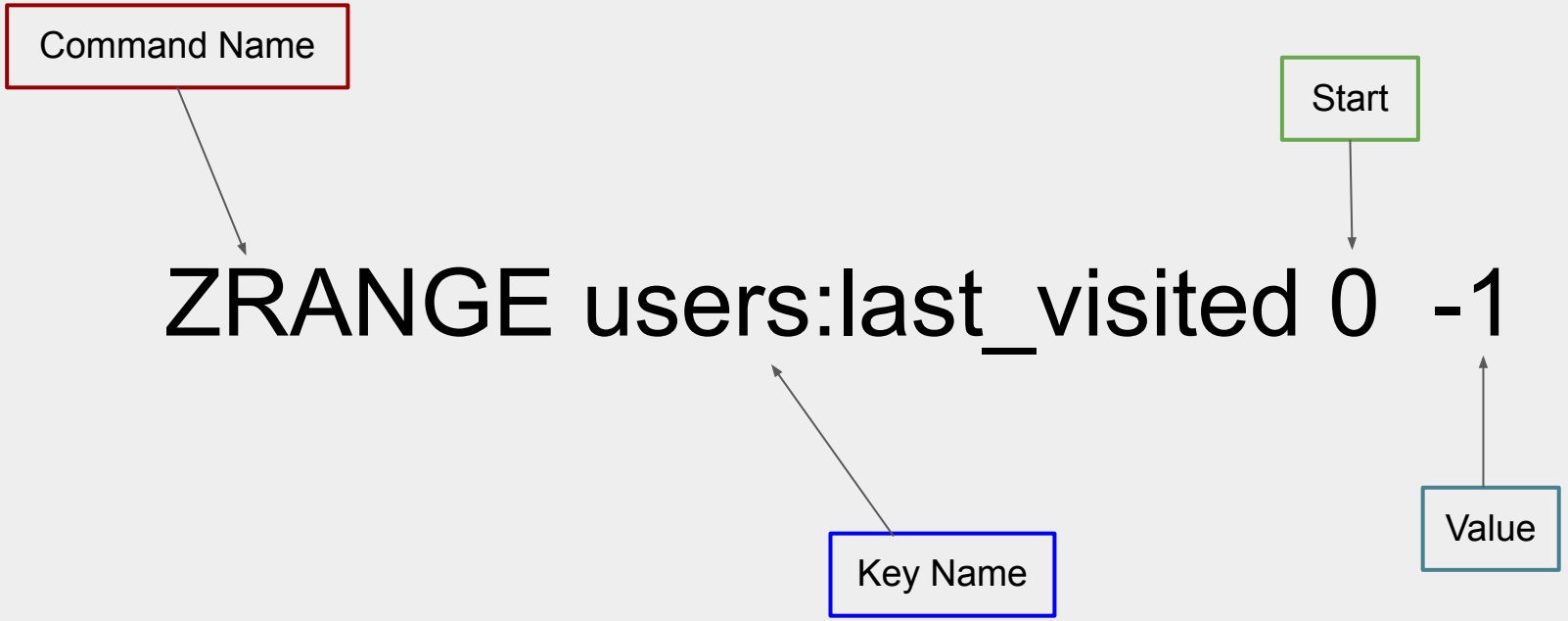
- Same membership rules as sets
- Set Sorted by Member ‘score’
- Can be read in order or reverse order

Sorted Set Range Types

- By Rank
- By Score
- By Lex

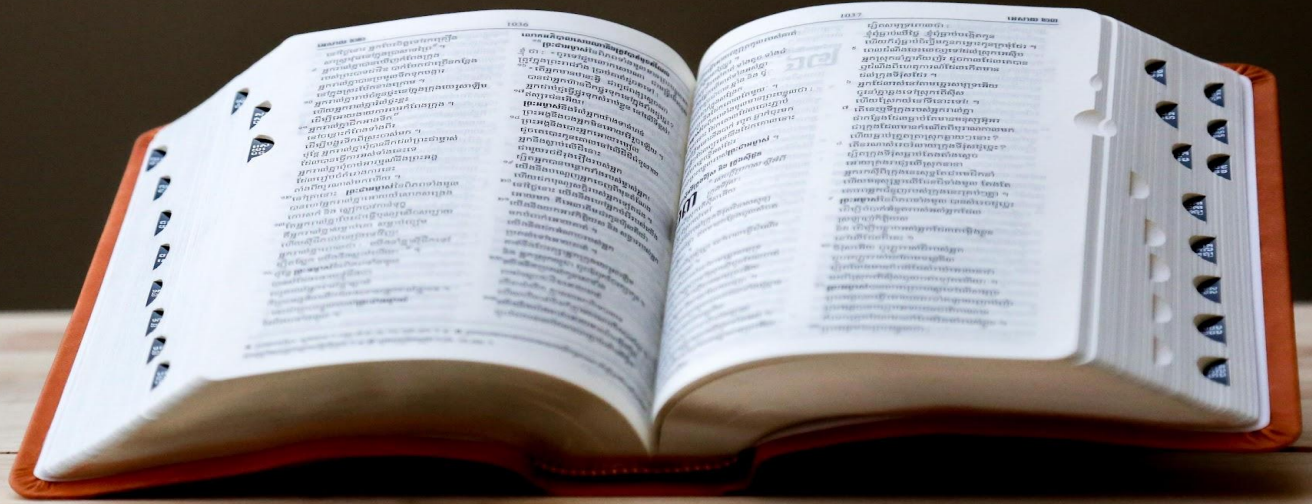



```
db.SortedSetAdd("users:last_visited", "steve", 1651259876 );
```



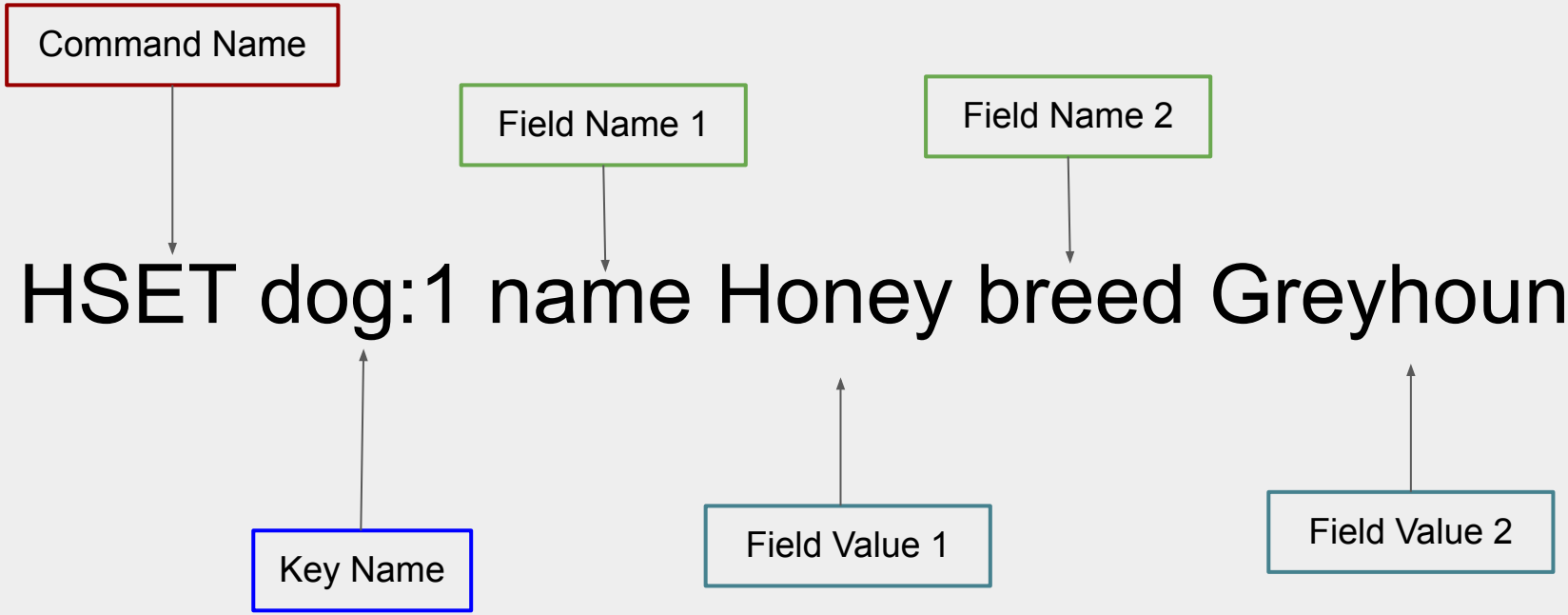
db.SortedSetRangeByRank("users:last_visited");

Hashes

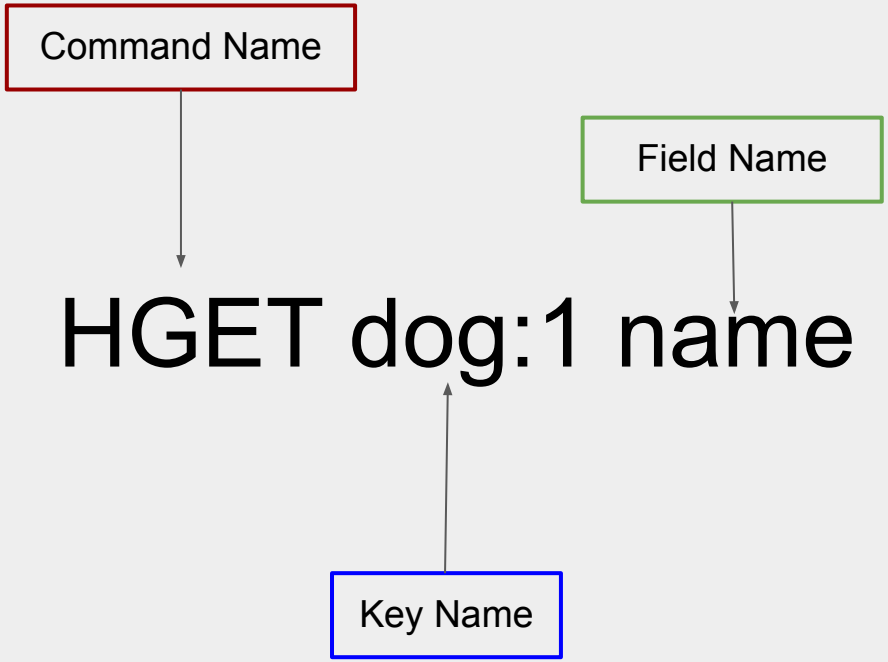


Hashes

- Field value stores
- Appropriate for flat objects
- Field Names and Values are Redis Strings




```
db.HashSet("dog:1", new HashEntry[] {new ("name", "Honey"), new
("breed", "Greyhound")});
```

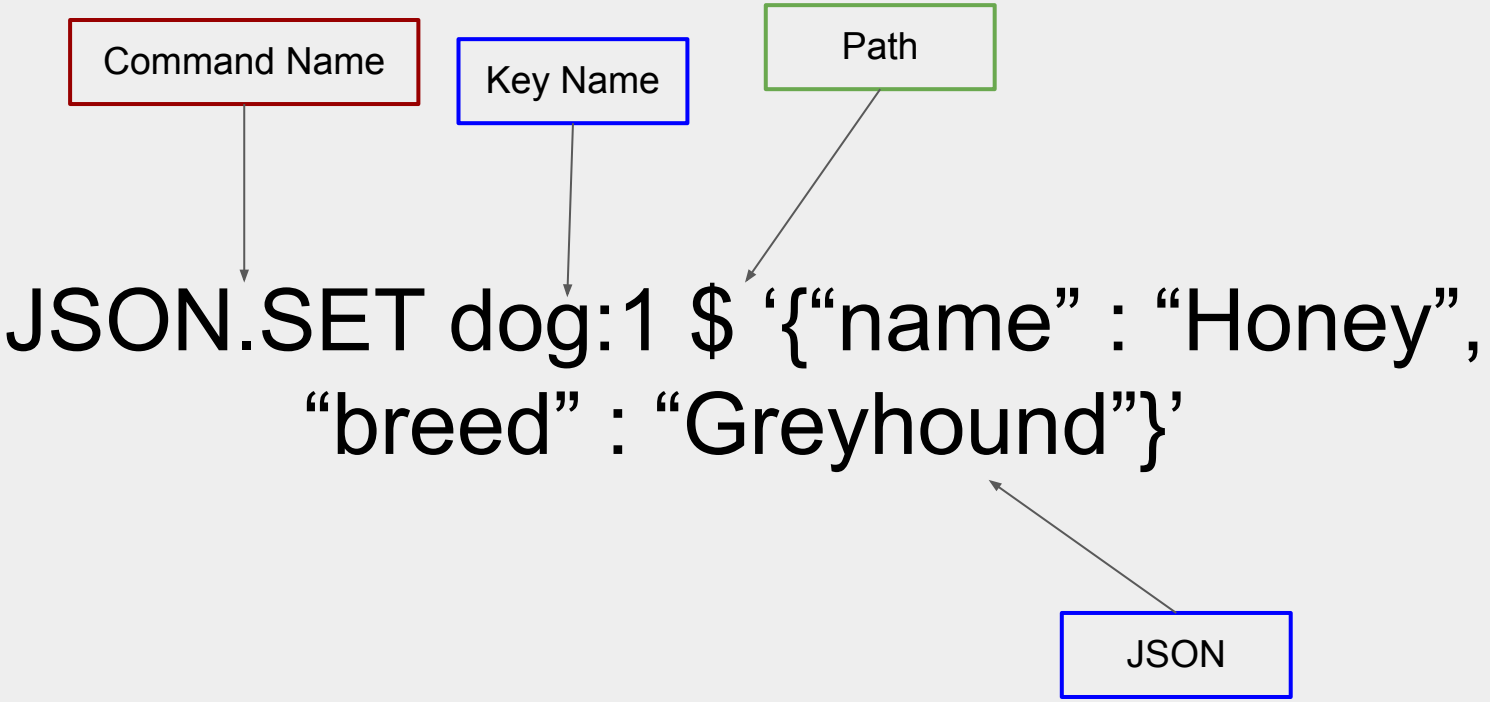



```
db.HashGet("dog:1", "name");
```

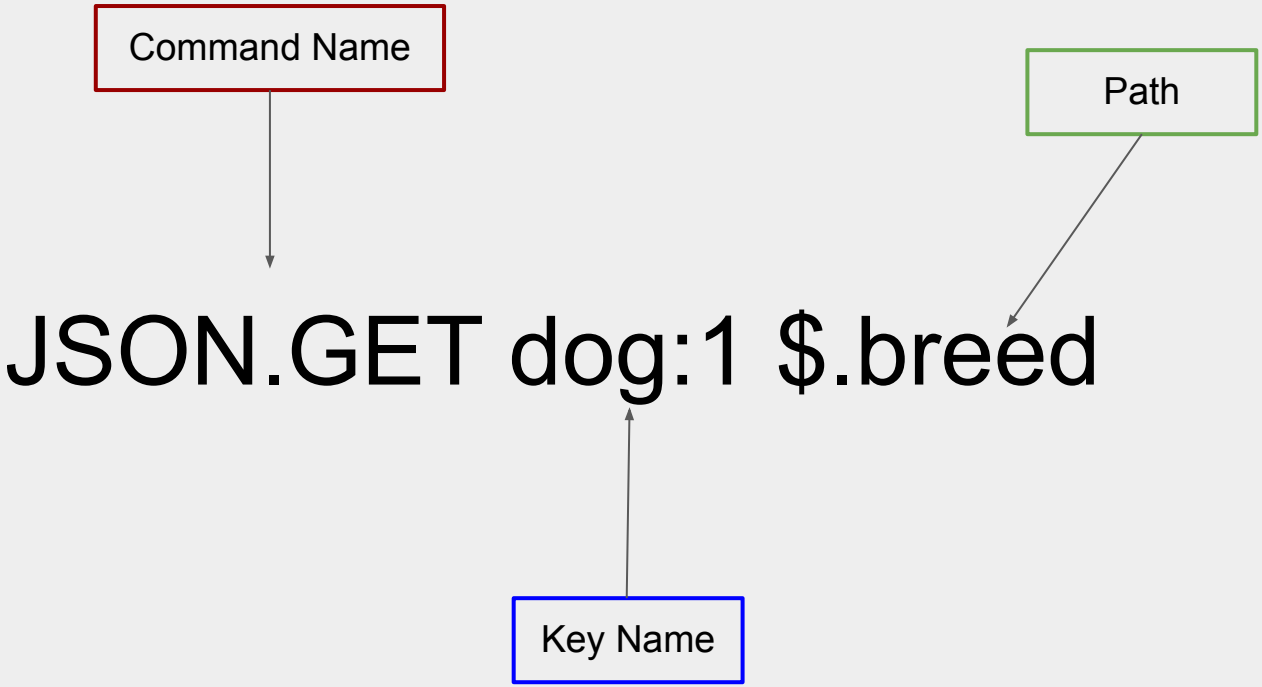
JSON

JSON

- Store full JSON objects in Redis
- Access fields of object via JSON Path
- Redis Module




```
db.Execute("JSON.SET", "dog:1", "$",
"{\"name\":\"Honey\",\"breed\":\"Greyhound\"}");
```




```
db.Execute("JSON.GET", "dog:1", "$.breed");
```

Streams

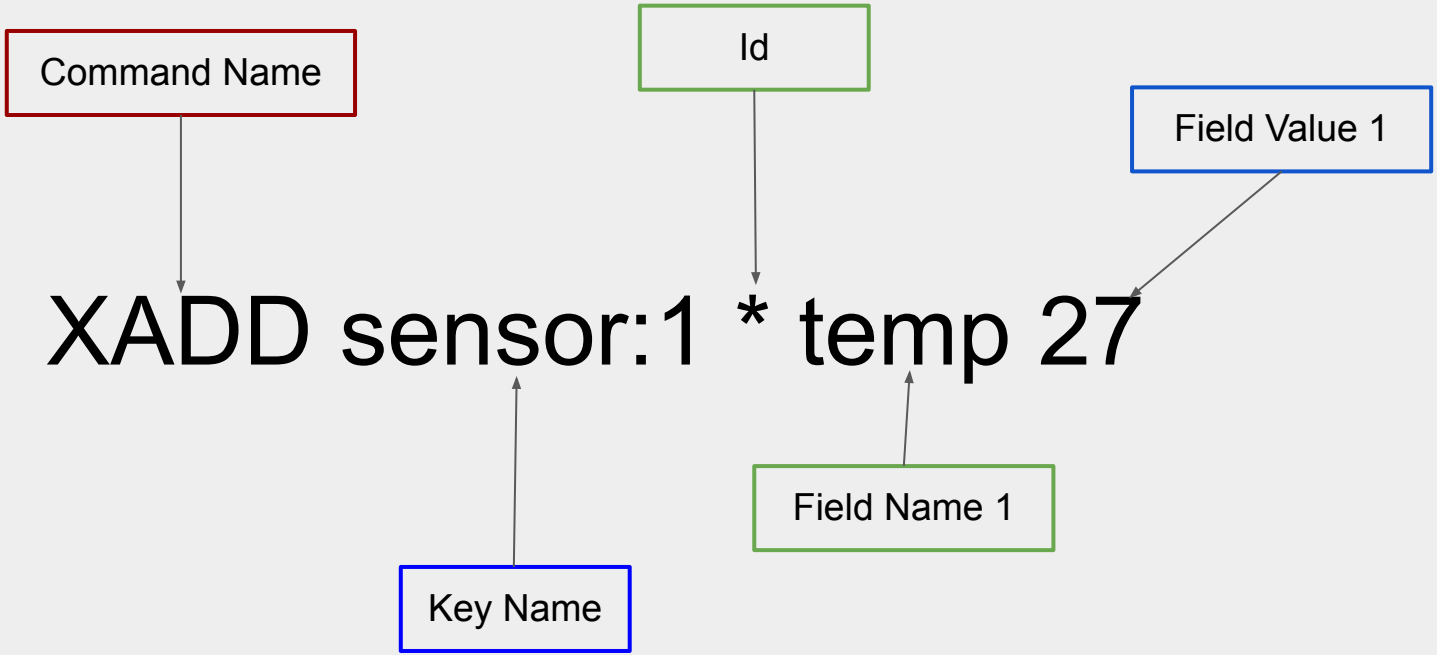
Streams

- Implements Log Data Structure
- Send messages through Streams as Message Queue
- Messages Consist of ID & list of field-value pairs
- Consumer Groups for coordinated reads

Stream ID

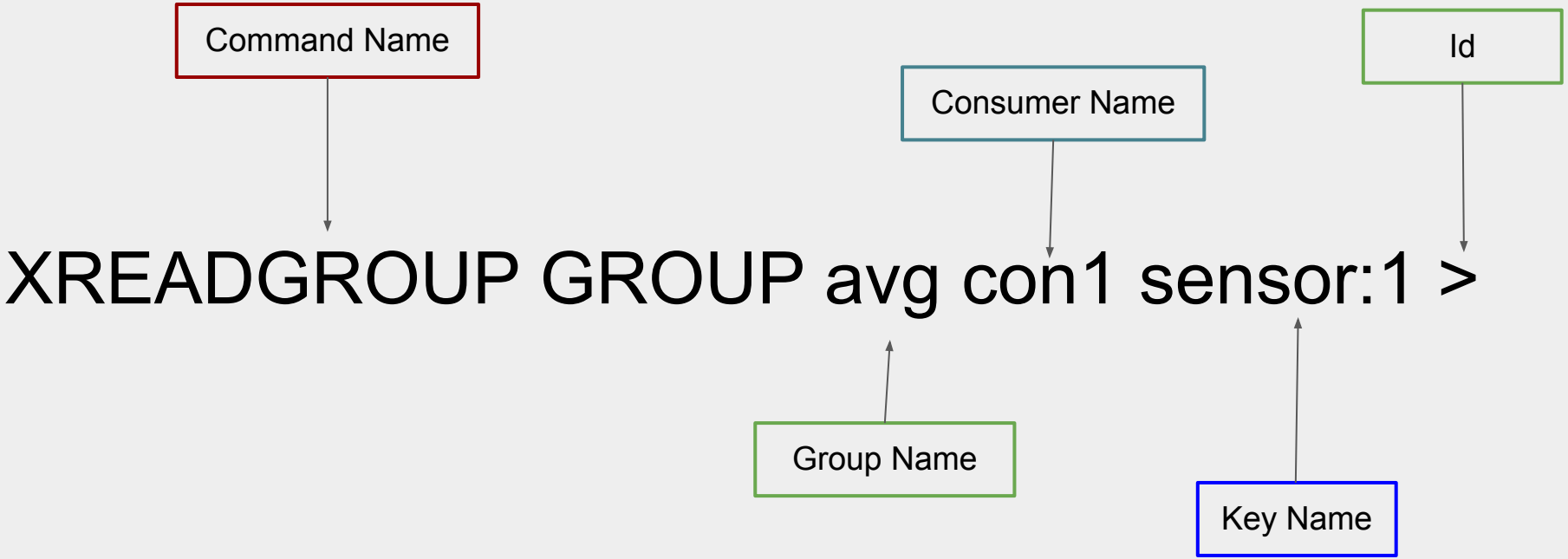
Special IDs

Id	Command	Description
*	XADD	Auto-Generate ID
$	XREAD	Only retrieve new messages
>	XREADGROUP	Next Message for Group
-	XRANGE	Lowest
+	XRANGE	Highest



db.StreamAdd("sensor:1", "temp", 27);

db.StreamRead("sensor:1", 0);




```
db.StreamReadGroup("sensor:1", "avg", "con1", ">");
```

Redis Design Patterns

Round Trip Management

The Root of Most Redis Bottlenecks

- Op-times in Redis Measured in Microseconds
- RTTs in Redis Measured in Milliseconds
- Minimize RTT
- Minimize number of Round Trips

Variadic Commands

- Many commands are variadic
- Leverage variadicity of commands

Always Be Pipelining

- Send multiple commands simultaneously
- Appropriate when intermediate results are unimportant
- Can dramatically increase throughput


Object Storage
9
8
7

Document Storage

- 3 Models of Document Storage
  - Hashes
  - Raw JSON
  - JSON Data Structure

Hashes

- Native flat-object Data structure
- HSET (which is variadic) to create/update
- HGET/HMGET/HGETALL to get fields in the hash
- HDEL/UNLINK to delete fields/objects

Pros:

- Native
- Performant

Cons:

- Breaks down on more complicated objects
- Breaks down with collection storage

JSON Blobs

- Native data structure (a string)
- SET to create/update
- GET to read
- UNLINK to delete

Pros:

- Native
- Simple
- Your input is probably in this format

Cons:

- Updates are expensive—O(N)
- Reads are expensive—O(N)

JSON Data Structure

- Exists in the Redis JSON module
- JSON.SET to create/update
- JSON.GET to read
- JSON.DEL to remove fields
- UNLINK to delete

Pros:

- All operations are fast
- Organized retrieval/update of data within object
- Works great with rich objects

Cons:

- Needs a module

Finding Stuff

Indexing

- Finding by value requires the use of a pattern
- Two patterns
  - Indexing With Sorted Sets
  - Indexing With RediSearch

Sored Sets

- Construct Sorted sets to Index
- e.g.
  - Person:firstName:Steve{(0, Person:1)}
  - Person:age{(33, Person:1)}
- Query with ZRANGE commands
- Complex queries with Set Combination Commands

Pros:

- Native, in any Redis
- Performant

Cons:

- Devilishly tricky to maintain consistency
- Cross-shard updates NOT atomic

RediSearch + Redis OM

- Predefine the Index in model
- Save insert objects into Redis:
  - `collection.Insert(steve);`
- Query with LINQ
  - collection.Where(x=>x.Name == “Steve”);

Pros:

- Fast
- Easy to use
- Cross-shard indexing taken care of

Cons:

- Need to use a separate module

Transactions and Scripts

Transactions

- Apply multiple commands to server atomically
- Start with MULTI
- Issue commands
- End with EXEC
- WATCH keys to make sure they aren’t messed with during transaction
- If commands are valid Transaction will Execute

Lua Scripts

- Allow scripted atomic execution
- Allows you to use intermediate results atomically

```
local id = redis.call('incr', 'next_id')
redis.call('set','key:' .. id, 'val')
return id
```

Prepared Scripts

- StackExchange.Redis feature
- Tracks script sha
- Use @ to parameterize scripts
- Pass in object with parameters

```
var script = LuaScript.Prepare("return @name");
var scriptResult = db.ScriptEvaluate(script, new {name="Steve"});
```

Redis Gotchas and Anti-Patterns

Timeouts

- #1 issue encountered
- Configurable timeout (defaults to 5 seconds)
- Clock starts when the library queues command internally
- Variety of causes
  - Slow Redis commands block redis
  - Very large payloads block connection
  - Thread pool starvation
  - Thread theft

Reuse connections

- Connecting to Redis is Expensive
- Pool connections when possible
- Use one ConnectionMultiplexer

SCAN, please don’t use Keys

- KEYS command can lock up Redis
- Use cursor-based SCAN instead

Hot Keys

- Single keys with very heavy access patterns
- redis-cli --hotkeys
- Mitigates scaling advantages of clustered Redis
- Consider duplicating data across multiple keys
- Consider client-side caching

Steve Lorello
Developer Advocate
@Redis

@slorello
github.com/slorello89

Resources

Redis
https://redis.io

Redis-Stack docker Image
https://hub.docker.com/r/redis/redis-stack/

StackExchange.Redis
https://github.com/stackExchange/StackExchange.Redis

Redis OM
https://github.com/redis/redis-om-dotnet

Code Examples
https://github.com/slorello89/dotnet-redis-examples

Slides
https://www.slideshare.net/StephenLorello/an-introduction-to-redis-for-net-developerspdf

Come Check Us Out!

Redis University:
https://university.redis.com

Discord:
https://discord.gg/redis

redis