Foundation of Data Engineering

MCF Riccardo Tommasini

http://rictomm.me

riccardo.tommasini@insa-lyon.fr

Key-Value Stores

Why Key-value Store?

(Business) Key -> Value

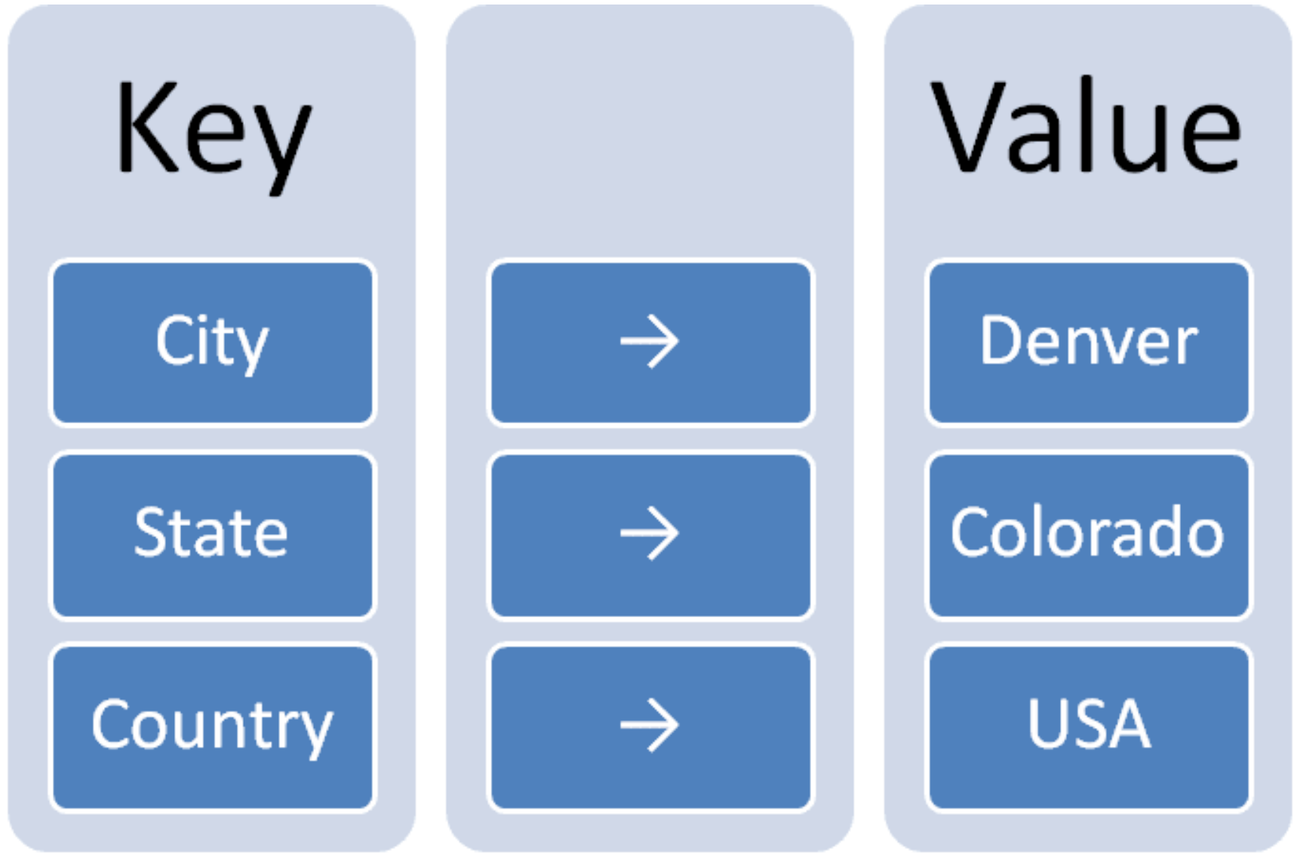
(twitter.com) tweet id -> information about tweet

(kayak.com) Flight number -> information about flight

(yourbank.com) Account number -> information about it

(amazon.com) item number -> information about it

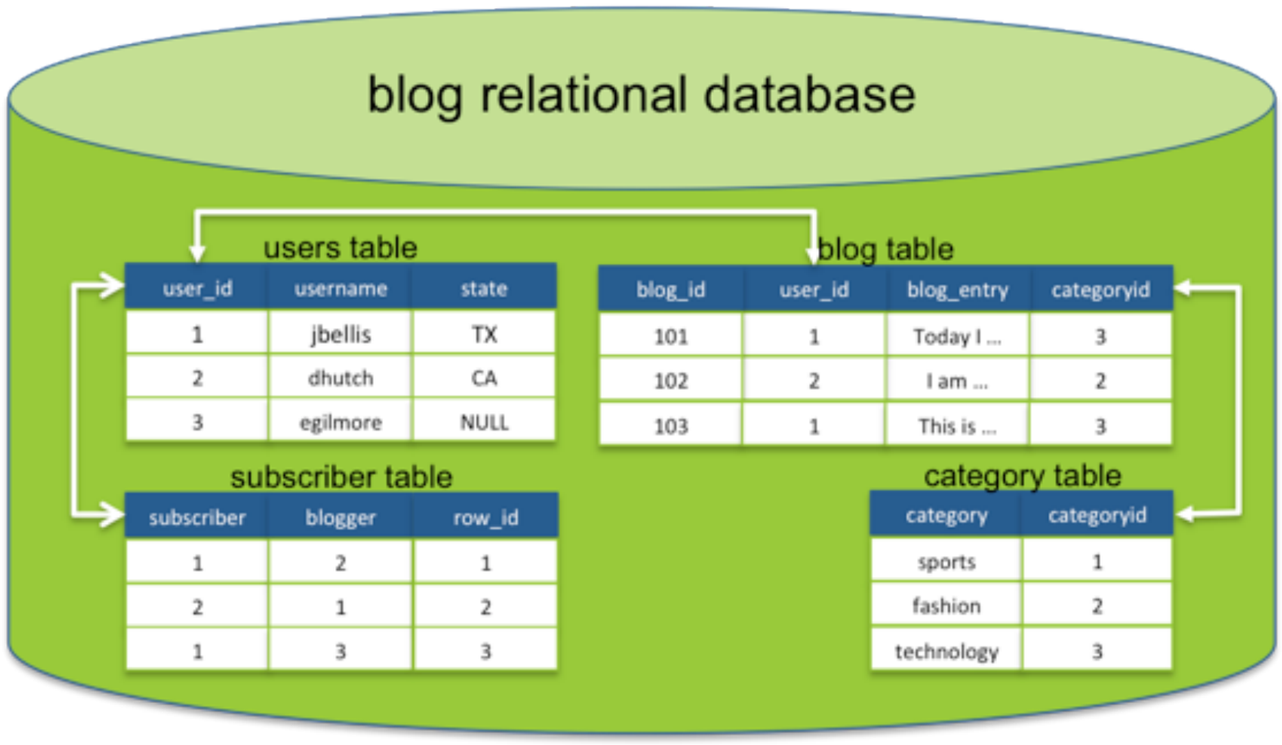
Search by ID is usually built on top of a key-value store



Isn’t that just a database?

- Queried using SQL
- Key-based
- Foreign keys
- Indexes
- Joins

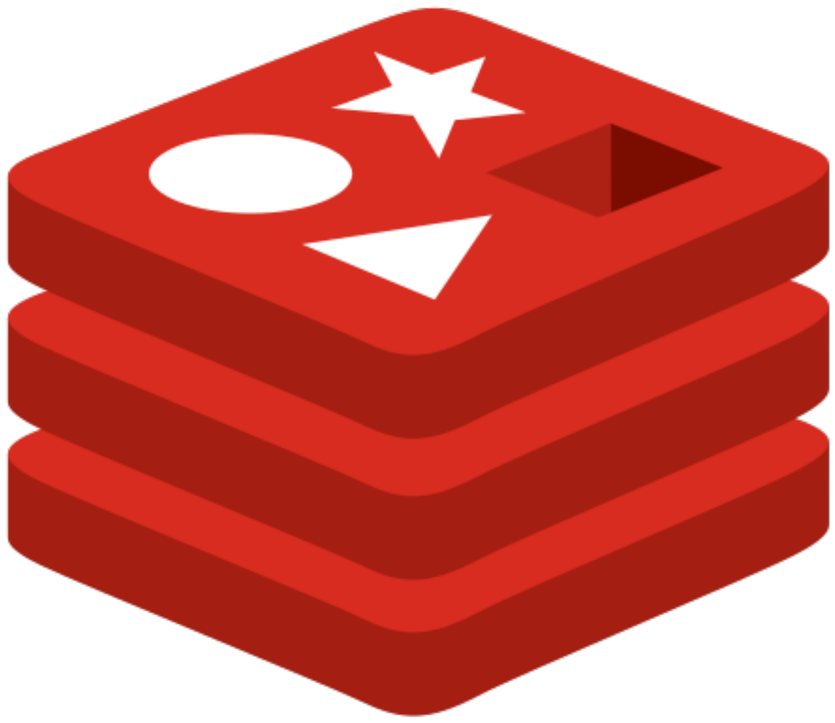
```
SELECT user_id from users WHERE
    username = “jbellis”
```



Systems

- Amazon DynamoDB: Probably *the* most widely used key-value store database, in fact, it was the research into DynamoDB that really started making NoSQL really popular.
- Aerospike: Open-source database that is optimized for in-memory storage.
- Berkeley DB: Another open-source database that is a high-performance database storage library, although it's relatively basic.
- Couchbase: Interestingly allows for text searches and SQL-style querying.
- Memcached: Helps speed up websites by storing cache data in RAM, plus it's free and open-source.
- Riak: Made for developing apps, it works well with other databases and apps.
- Redis: A multi-purpose database that also acts as memory cache and message broker.
- RocksDB
- ETCD

Redis



redis

Redis History

- Written in ANSI C by Salvatore Sanfilippo 🇮🇹
- Works in most POSIX systems like Linux, BSD and OS X.
- **Linux is the recommended** [65]
- Redis is a single-threaded server, not designed to benefit from multiple CPU cores.
- Several Redis instances can be launched to scale out on several cores.
- All operations are atomic (no two commands can run at the same time).
- It executes most commands in O(1) complexity and with minimal lines of code.

---

[65] No official support for Windows, but Microsoft develops and maintains an open source Win-64 port of Redis*

What Redis is

- An advanced Key-Value Stores, where keys can contain data structures such as strings, hashes, lists, sets, and sorted sets.
- It supports a set of atomic operations on these data types.
- Redis is a different evolution path in the key-value databases where values are complex data types that are closely related to fundamental data structures and are exposed to the programmer as such, without additional abstraction layers.
- Redis Can be used as **Database** [^ 61], a **Caching layer** [62] or a **Message broker** [63]

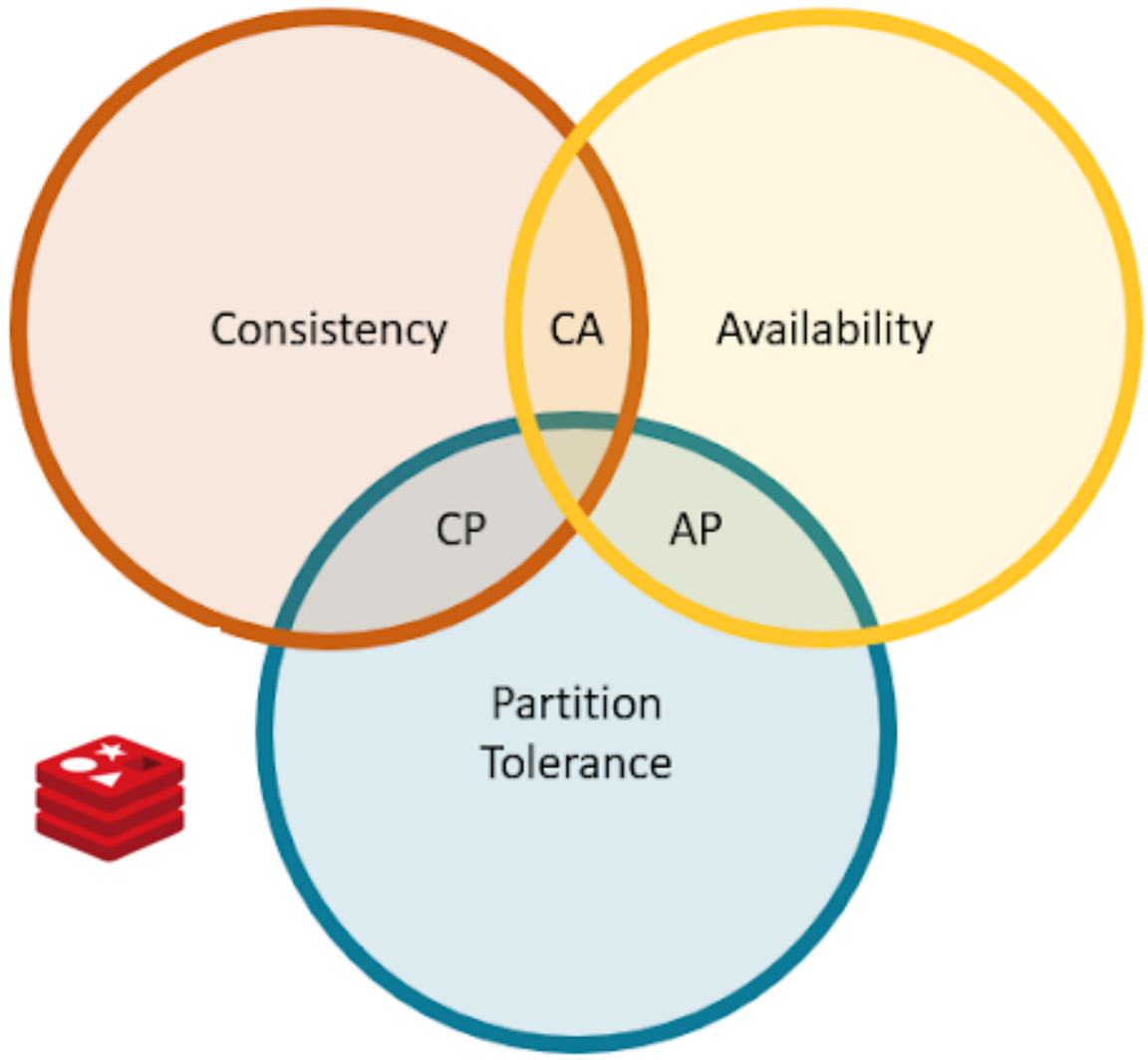
---

[62] it is fast

[63] is not only a key-value store

What Redis is

- An In-Memory Key-Value Store
- Weakly consistency (**C**)
- *Highly* available (**A**)
- Horizontal Scalable (**P**)



[62] it is fast

[61] it is durable

[62]: it is fast

<!-- keys can contain data structures such as strings, hashes, lists, sets, and sorted sets. values are complex data types that are closely related to fundamental data structures and are exposed to the programmer as such, without additional abstraction layers. It supports a set of atomic operations on these data types. Redis Can be used as a **Caching layer**62 or a **Message broker**61 -->

What Redis is NOT

- Redis is not a replacement for Relational Databases nor Document Stores.
- It might be used complementary to a SQL relational store, and/or NoSQL document store.
- Even when Redis offers configurable mechanisms for persistency, increased persistency will tend to increase latency and decrease throughput.
- Best used for rapidly changing data with a foreseeable database size (should fit mostly in memory).

Redis Use Cases

- Caching
- Counting things
- Blocking queues
- Pub/Sub (service bus)
- MVC Output Cache provider
- Backplane for SignalR
- ASP.NET Session State provider[64]
- Online user data (shopping cart,…Any real-time, cross-platform, cross-application communication

---

[64] ASP.NET session state providers comparison: http://www.slideshare.net/devopsguys/best-performing-aspnet-session-state-providers

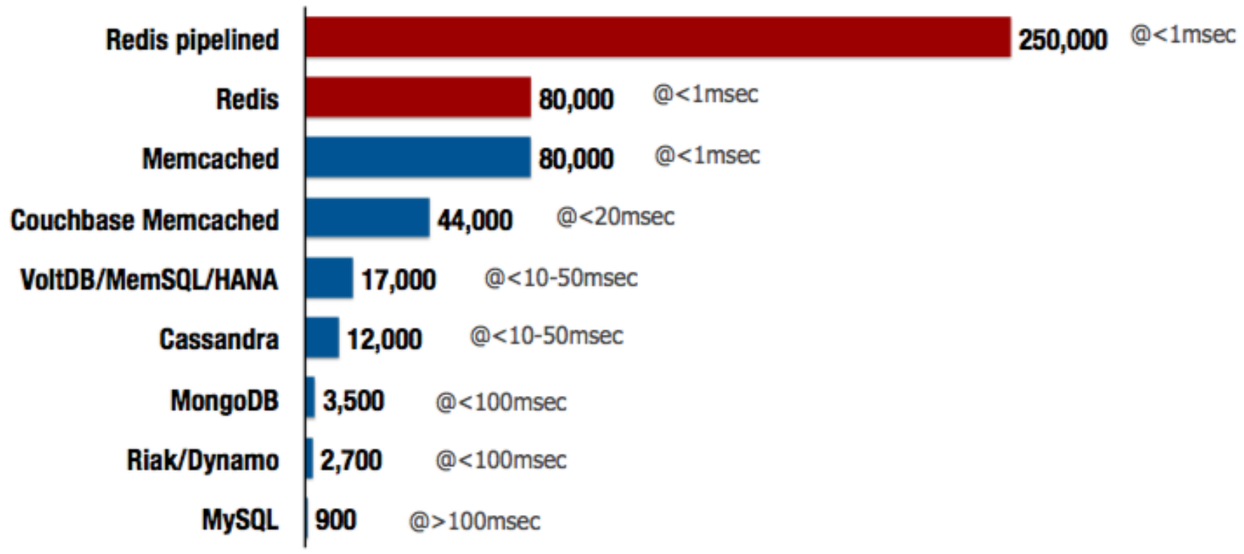
When to consider Redis

- Speed is critical
- More than just key-value pairs
- Dataset can fit in memory
- Dataset is not critical

Advantages

- Performance
- Availability
- Fault-Tolerance
- Scalability (adaptability)
- Portability

source



Data Model

- Key
  - Printable ASCII

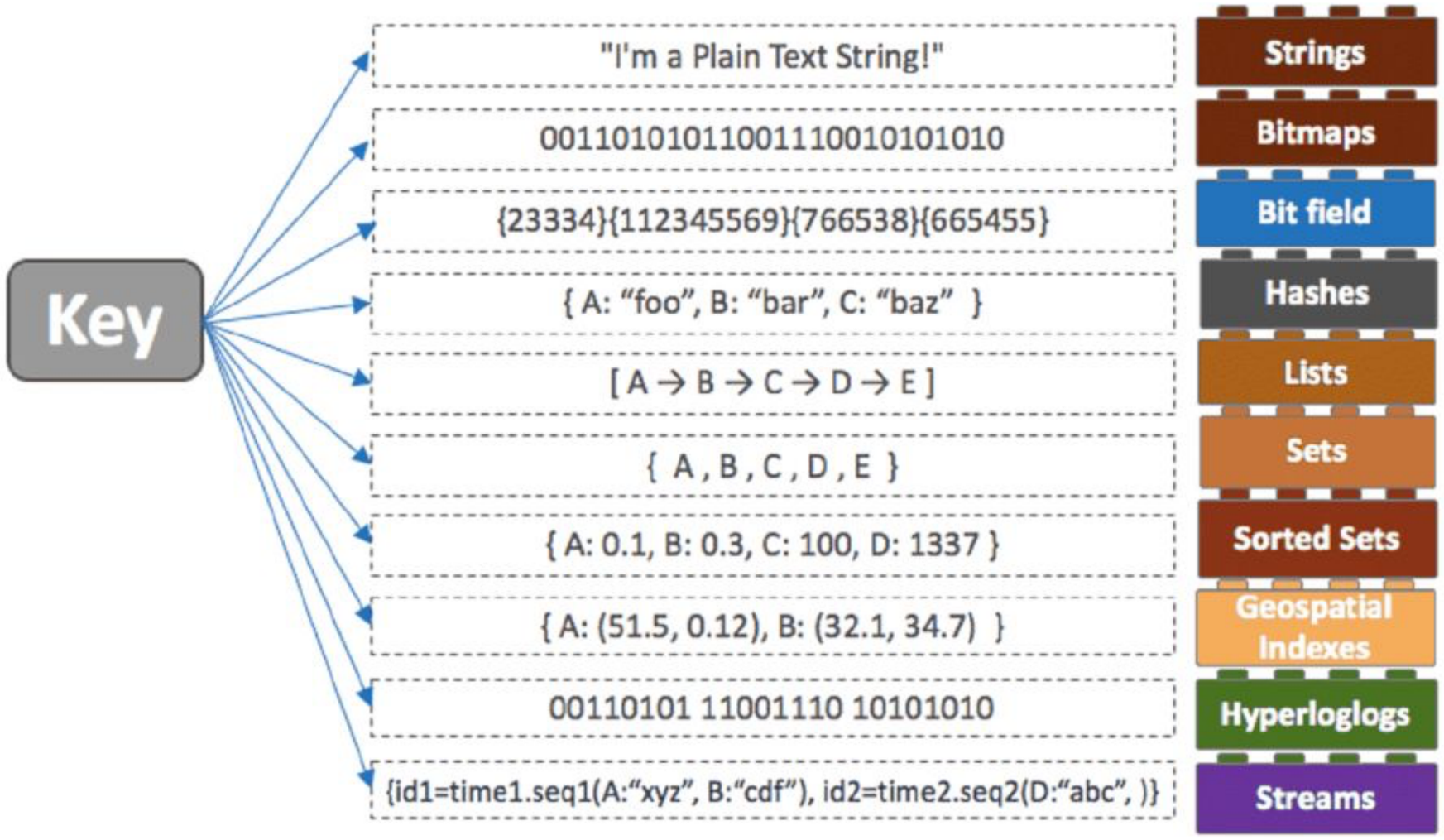
ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

- Value
  - Primitives
    - Strings
  - Containers (of strings)
    - Hashes
    - Lists
    - Sets
    - Sorted Sets

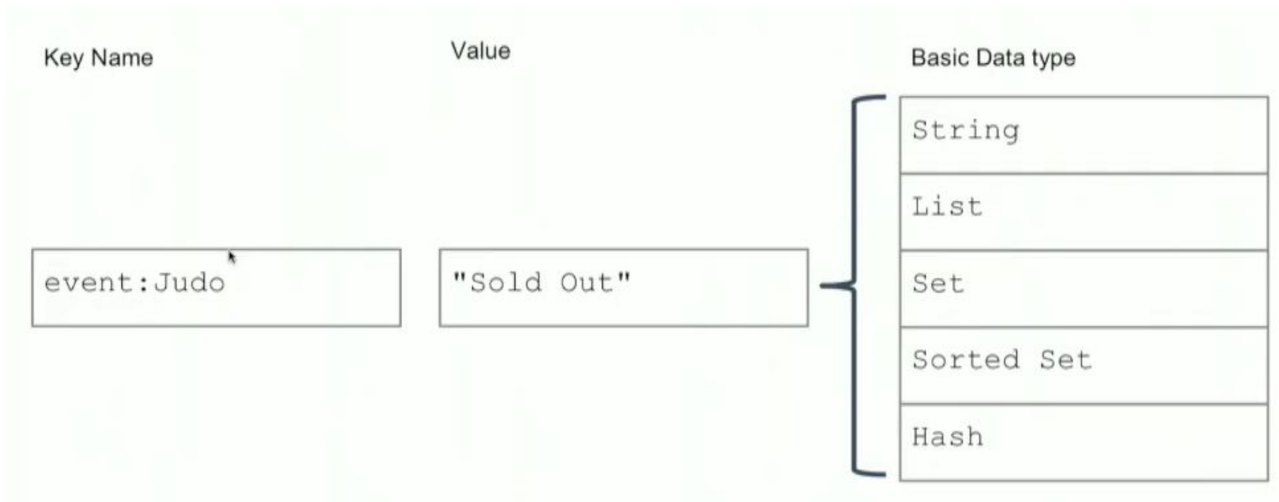
Redis data types

Collection	Contains	Read/Write Ability
String	Binary-safe strings (up to 512 MB), Integers or Floating point values, Bitmaps.	Operate on the whole string, parts, increment/ decrement the integers and floats, get/set bits by position.
Hash	Unordered hash table of keys to string values	Add, fetch, or remove individual ítems by key, fetch the whole hash.
List	Doubly linked list of strings	Push or pop items from both ends, trim based on offsets, read individual or multiple items, find or remove items by value.
Set	Unordered collection of unique strings	Add, fetch, or remove individual items, check membership, intersect, union, difference, fetch random items.
Sorted Set	Ordered mapping of string members to floating-point scores, ordered by score	Add, fetch, or remove individual items, fetch items based on score ranges or member value.
Geospatial Index	Sorted set implementation using geospatial information as the score	Add, fetch or remove individual items, search by coordinates and radius, calculate distance.
HyperLogLog	Probabilistic data structure to count unique things using 12Kb of memory	Add individual or multiple items, get the cardinality.


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
{id1=time1.seq1(A:“xyz”, B:“cdf”), id2=time2.seq2(D:“abc”,)}
Strings
Bitmaps
Bit field
Hashes
Lists
Sets
Sorted Sets
Geospatial Indexes
Hyperloglogs
Streams

Key-Value Essentials

- keys must be unique
  - namespacing (next slide)
  - do not follow relational patterns
  - aggregation over join
- keys must be portable
  - less than 512mb
  - binary safe (serialization)
  - binary comparable is a plus (key arithmetic)



Naming

- You are free to choose your own naming conventions, but..
  - Be Consistent
  - Use Namespaces
    - domain*object:uniqueID*, *e.g.*, *event:Balon*DOR
  - Case sensistive
  - Distinguish values from collections (plural)

Redis Commands - Strings [Quiz]

Command	**Abstract Syntax**	**Complexity**
Get/Set strings	SET [key value] GET [key]	?
Increment numbers	INCRBY [key increment]	?
Get Multiple Keys at once	MGET [Key key ...]	?
Set Multiple Keys at once	MSET [Key key ...]	?
Get String Length	STRNEL [key]	?
Update Value and Get old one	GETSET	?

Redis Commands - Strings [Answers]

Command	**Abstract Syntax**	**Complexity**
Get/Set strings	SET [key value] GET [key]	O(1)
Increment numbers	INCRBY [key increment]	O(1)
Get Multiple Keys at once	MGET [Key key ...]	O(n) n=#Keys
Set Multiple Keys at once	MSET [Key key ...]	O(n) n=#Keys
Get String Length	STRNEL [key]	O(1)
Update Value and Get old one	GETSET [Key Value]	O(1)

Redis Commands - Keys [Quiz]

Command	**Abstract Syntax**	**Complexity**
Key Removal	DEL [key ...]	?
Text Existance	EXISTS [key...]	?
Get the type of a key	TYPE [Key]	?
Rename a key	RENAME [Key NewKey]	?

Redis Commands - Keys [Answers]

Command	Abstract Syntax	Complexity
Key Removal	DEL [key ...]	O(1)
Text Existance	EXISTS [key...]	O(1)
Get the type of a key	TYPE [Key]	O(1)
Rename a key	RENAME [Key NewKey]	O(1)

Redis Commands [Answers]

Lists

Command	Abstract Syntax	Complexity
Push on either end	RPUSH/LPUSH [key value]	?
Pop from either end	RPOP/LPOP [key]	?
Blocking Pop	BRPOP/BLPOP [key value]	?
Pop and Push to other list	RPOPLPUSH [src dst]	?
Get an element by index	LINDEX [key index]	?
Get a range of elements	LRANGE [key start stop]	?

Hashes

Command	Abstract Syntax	Complexity
Set a hashed value	HSET [key field value]	?
Set multiple fields	HMSET [key field value ...]	?
Get a hashed value	HGET [key vield]	?
Get all the values in a hash	HGETALL [key]	?
Increment a hashed value	HINCRBY [key field incr]	?

Redis Commands [Answers]

Lists

Command	Abstract Syntax	Complexity
Push on either end	RPUSH/LPUSH [key value]	O(1)
Pop from either end	RPOP/LPOP [key]	O(1)
Blocking Pop	BRPOP/BLPOP [key value]	O(1)
Pop and Push to other list	RPOPLPUSH [src dst]	O(1)
Get an element by index	LINDEX [key index]	**O(n)**
Get a range of elements	LRANGE [key start stop]	**O(n)**

Hashes

Command	Abstract Syntax	Complexity
Set a hashed value	HSET [key field value]	O(1)
Set multiple fields	HMSET [key field value ...]	O(1)
Get a hashed value	HGET [key vield]	O(1)
Get all the values in a hash	HGETALL [key]	O(N) : N=size of hash
Increment a hashed value	HINCRBY [key field incr]	O(1)

Redis Commands [Quiz]

Sets

Command	Abstract Syntax	Complexity
Add member to a set	SADD [key member]	?
Pop random element	SPOP [key]	?
Get all elements	SMEMEBRS [Key]	?
Union multiple sets	SUNION [Key Key ...]	?
Diff multiple sets	DIFF [Key key ...]	?

Sorted Sets

Command	Abstract Syntax	Complexity
Add member to a sorted set	ZADD [key member]	?
Get rank member	ZRANK [key member]	?
Get elements by score range	ZRANGEBYSCORE [key min max] [Key]	?
Increment score of member	ZINCRBY [Key incr member]	?
remover range by score	ZREMRANGEBYSCORE [key min max]	?

Redis Commands [Answers]

Sets

Command	Abstract Syntax	Complexity
Add member to a set	SADD [key member]	O(1)
Pop random element	SPOP [key]	O(1)
Get all elements	SMEMEBRS [Key]	O(n) : n=size of set
Union multiple sets	SUNION [Key Key ...]	O(n)
Diff multiple sets	DIFF [Key key ...]	O(n)

Sorted Sets

Command	Abstract Syntax	Complexity
Add member to a sorted set	ZADD [key member]	O(log(n))
Get rank member	ZRANK [key member]	O(log(n))
Get elements by score range	ZRANGEBYSCORE [key min max] [Key]	O(log(n))
Increment score of member	ZINCRBY [Key incr member]	O(log(n))
remover range by score	ZREMRANGEBYSCORE [key min max]	O(log(n))

Scaling Redis

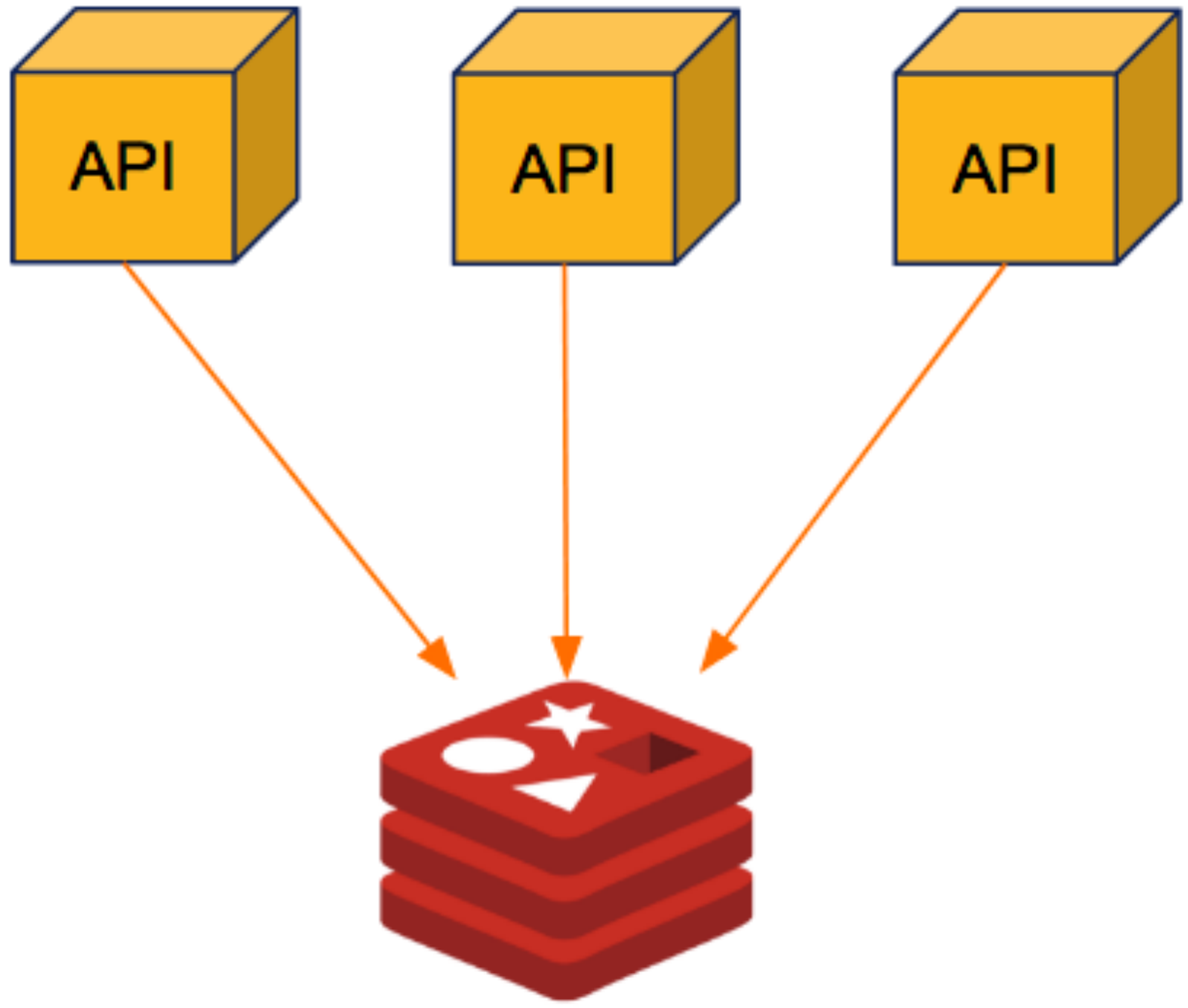
- **Replication**
  - A Redis instance, known as the **master** , ensures that one or more instances kwown as the **slaves** become exact copies of the master
  - Clients can connect *to the master or to the slaves*
  - Slaves are* read-only* by default
- **Partitioning**
  - Breaking up data and distributing it across different hosts in a cluster.
  - Can be implemented in different layers:
    - Client: Partitioning on client-side code
    - Proxy: An extra layer that proxies all redis queries and performs partitioning (i.e. Twemproxy)
    - Query Router: instances will make sure to forward the query to the right node. (i.e Redis Cluster)

Scaling Redis

- **Persistence**
  - Redis provides two mechanisms to deal with persistence: Redis database snapshots (RDB) and append-only files (AOF)
- **Failover**
  - Manual
  - Automatic with Redis Sentinel (for master-slave topology)
  - Automatic with Redis Cluster (for cluster topology)

Redis topologies

- Standalone
- Sentinel (automatic failover)
- Twemproxy (distribute data)
- Cluster (automatic failover and distribute data)



Redis topologies - Standalone

The master data is optionally replicated to slaves.

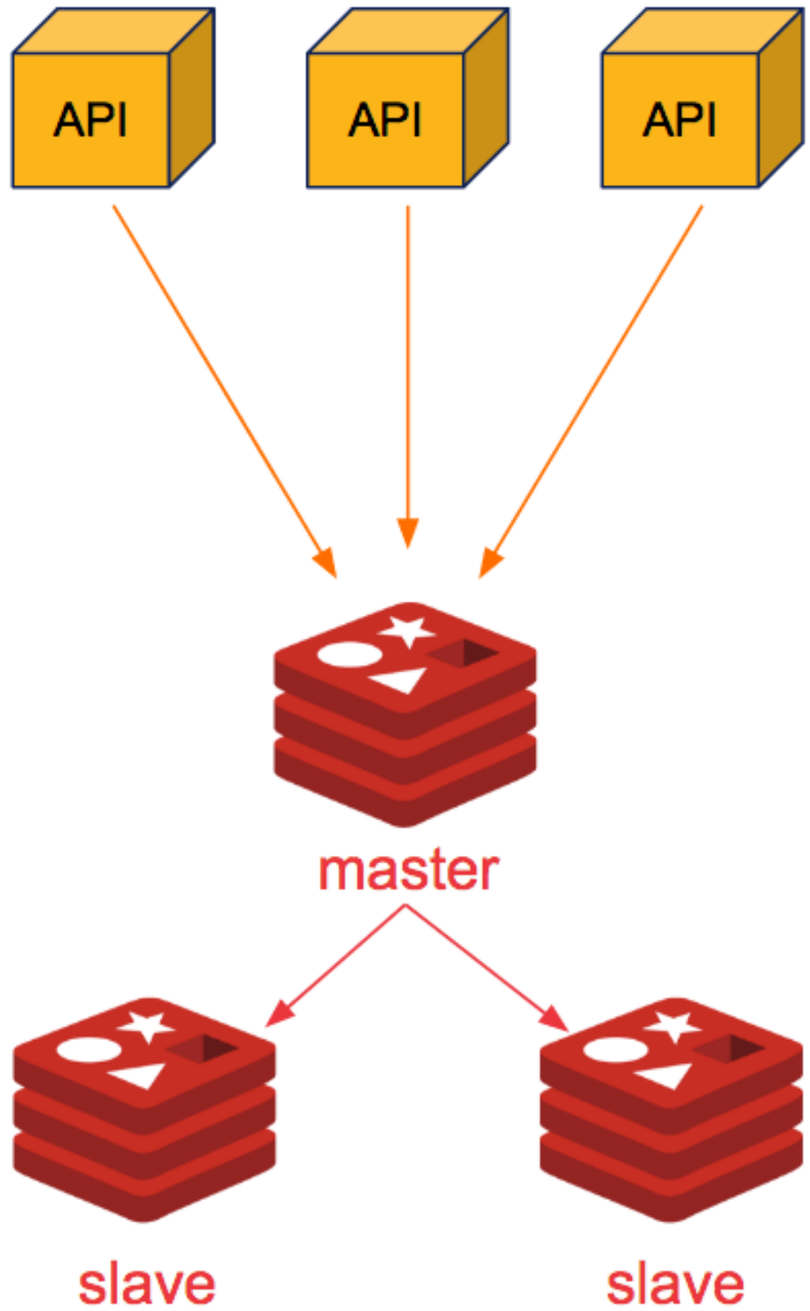
The slaves provides data redundancy, reads offloading and save-to-disk offloading.

Clients can connect to the Master for read/ write operations or to the Slaves for read operations.

Slaves can also replicate to its own slaves.

There is no automatic failover.

Master-slave multi-level

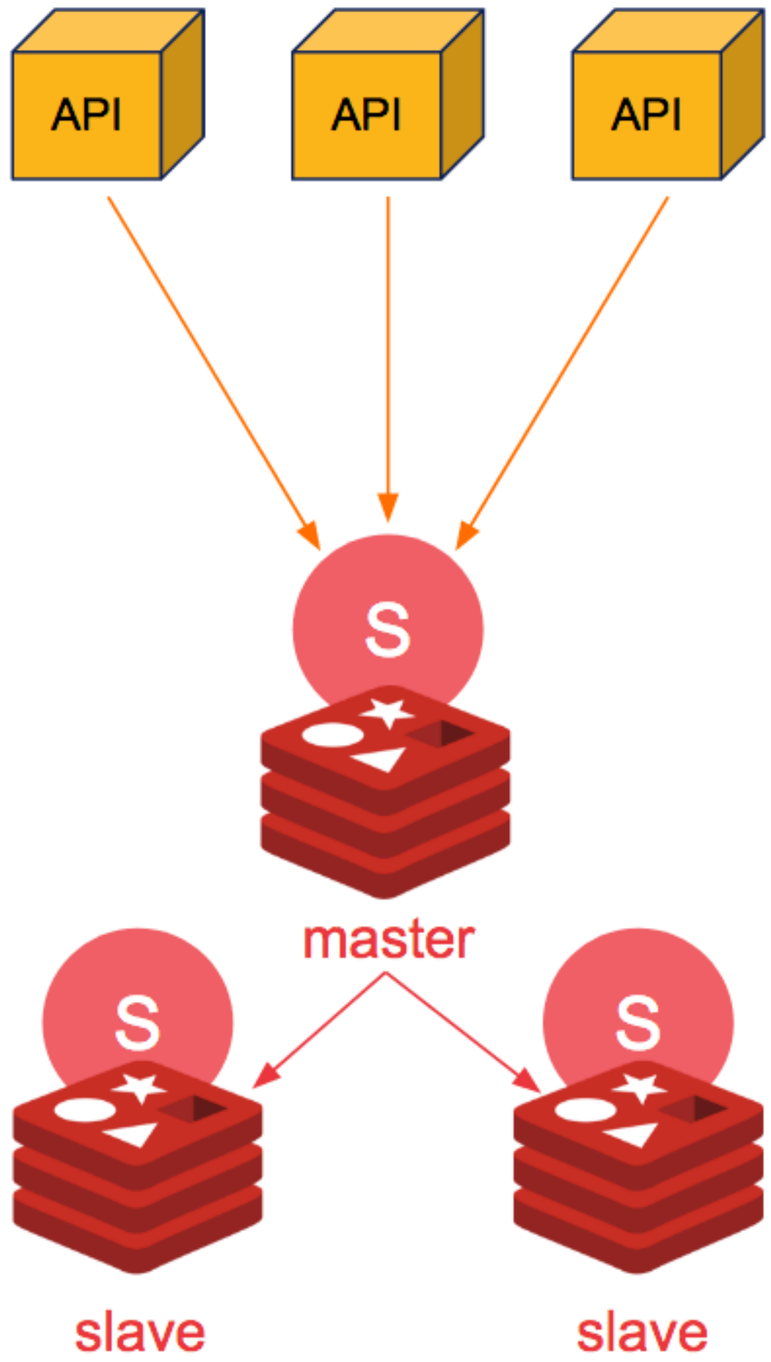


Redis topologies - Sentinel

Redis Sentinel provides a reliable automatic failover in a master/slave topology, automatically promoting a slave to master if the existing master fails.

Sentinel does not distribute data across nodes.

Master-slave with Sentinel



Redis topologies - Cluster

Redis Cluster distributes data across different Redis instances and perform automatic failover if any problem happens to any master instance.

All nodes are directly connected with a service channel.

The keyspace is divided into hash slots. Different nodes will hold a subset of hash slots.

Multi-key commands are only allowed for keys in the same hash slot.