



redislabs
HOME OF REDIS

10 Ways to Scale with Redis

IMCSUMMIT - NOVEMBER 2019 | DAVE NIELSEN

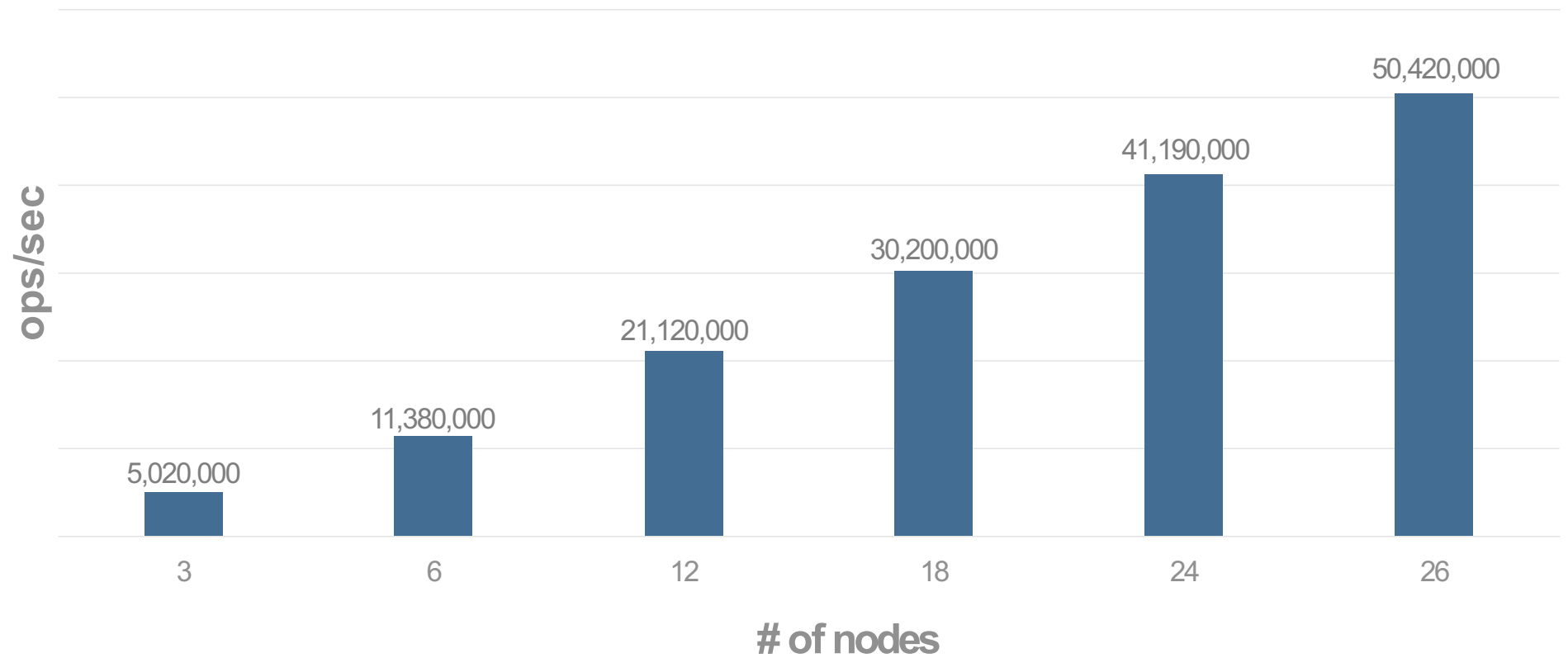


In-memory Multi-model Database



Optionally Persistent

Cluster Throughput (@ 1 msec Latency)





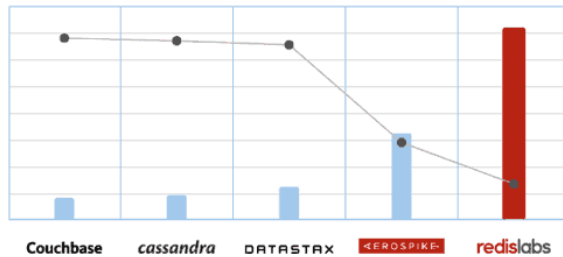
**Developers
+
Redis**

Redis Top Differentiators

1

Performance

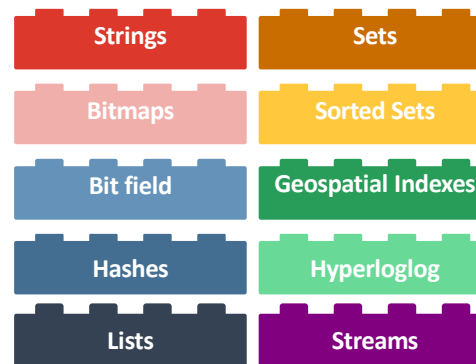
NoSQL Benchmark



2

Simplicity

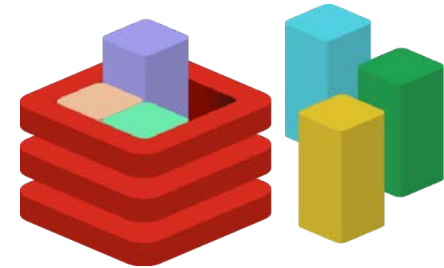
Redis Data Structures



3

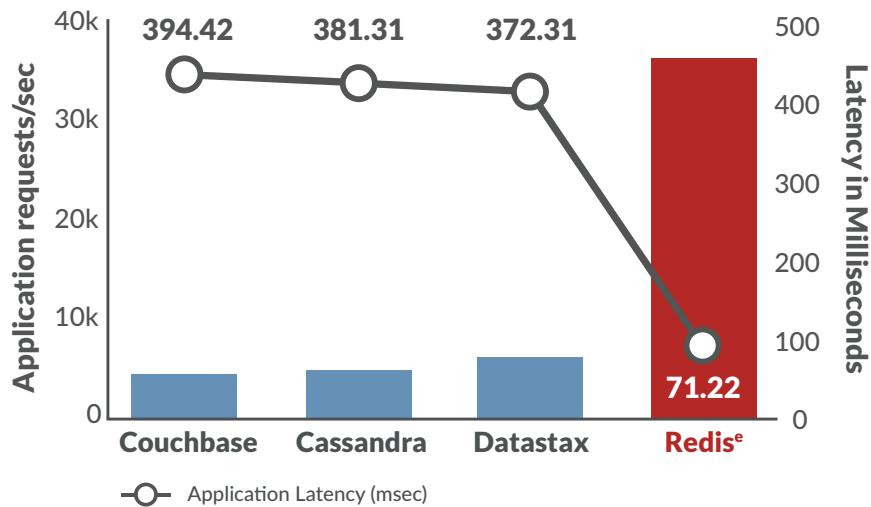
Extensibility

Redis Modules



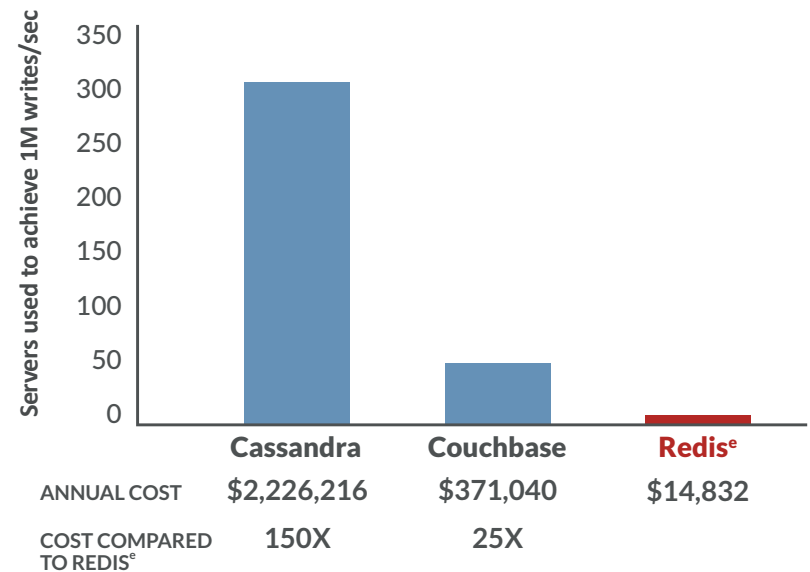
1 Performance: The Most Powerful Database

Highest Throughput at Lowest Latency
in High Volume of Writes Scenario



Benchmarks performed by Avalon Consulting Group

Least Servers Needed to
Deliver 1 Million Writes/Sec



Benchmarks published in the Google blog

2

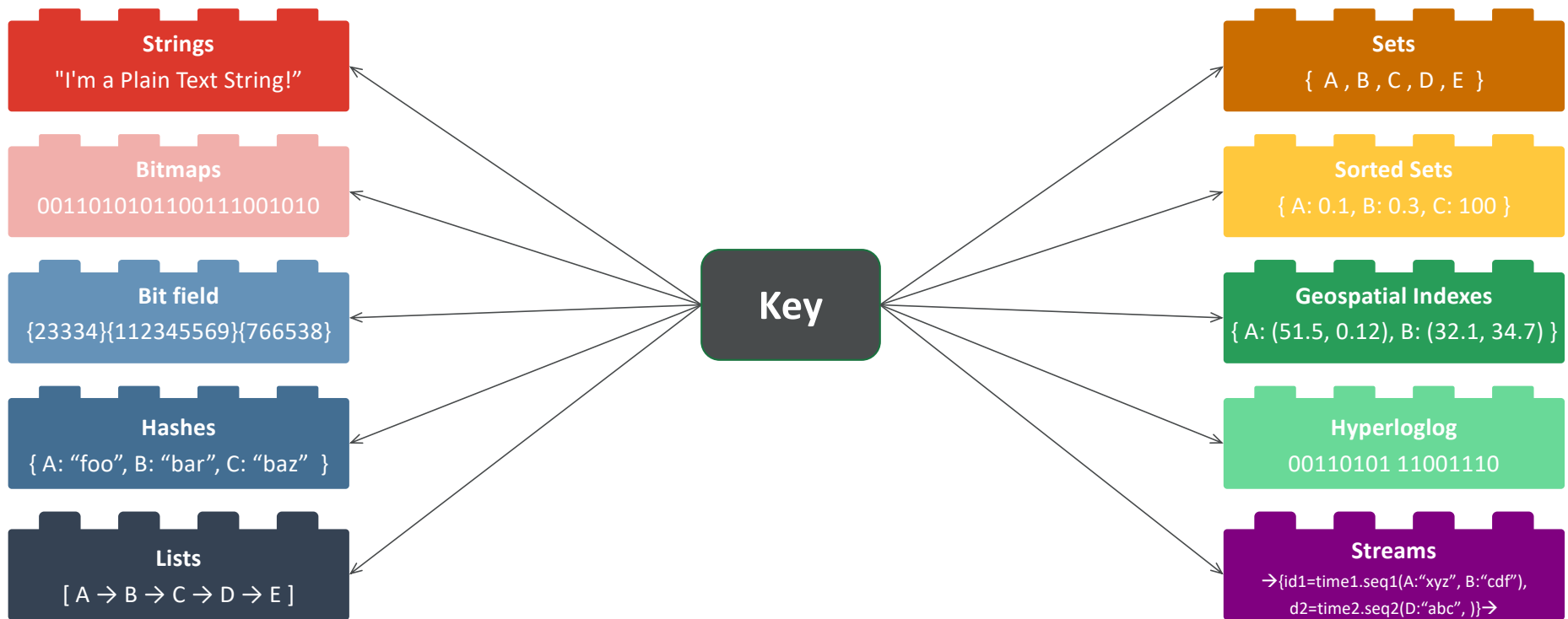
Simplicity: Data Structures - Redis' Building Blocks



“REDIS IS FULL OF DATA STRUCTURES!”

2

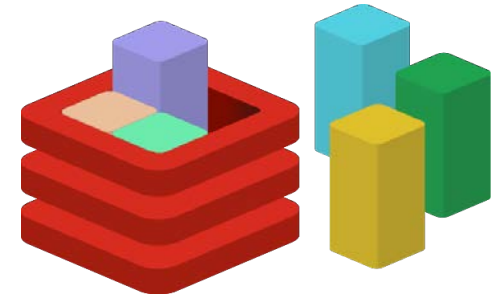
Simplicity: Redis Data Structures – 'Lego' Building Blocks



"Retrieve the e-mail address of the user with the highest bid in an auction that started on July 24th at 11:00pm PST" = **ZREVRANGE 07242015_2300 0 0**

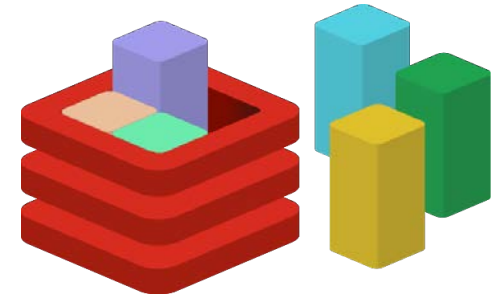
3 Extensibility: Modules Extend Redis Infinitely

- Add-ons that use a Redis API to seamlessly support additional use cases and data structures.
- Enjoy Redis' simplicity, super high performance, infinite scalability and high availability.
- Any C/C++/Go program can become a Module and run on Redis.
- Leverage existing data structures or introduce new ones.
- Can be used by anyone; Redis Enterprise Modules are tested and certified by Redis Labs.
- Turn Redis into a **Multi-Model** database



3 Extensibility: Modules Extend Redis Infinitely

- Redisearch
- RedisTimerseries
- ReJSON
- Rebloom
- RedisGraph
- Neural-Redis
- Redis-Cell
- Redis-Tdigest
- Redis-ML
- Redis-Rating
- Redis-Cuckoofilter
- Cthulhu
- Redis Snowflake
- redis-roaring
- Session Gate
- ReDe
- TopK
- countminsketch



Deep Dive





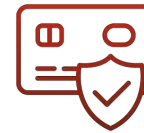
Real Time Analytics



User Session Store



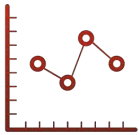
Real Time Data Ingest



High Speed Transactions



Job & Queue Management



Time Series Data



Complex Statistical Analysis



Notifications



Distributed Lock



Caching



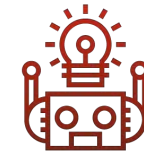
Very Large Data Sets



Geospatial Data



Streaming Data



Machine Learning



Search

Manage Session Stores w/ Redis Hash



A user session store is...

- An chunk of data that is connected to one “user” of a service
 - “user” can be a simple visitor
 - or proper user with an account
- Often persisted between client and server by a token in a cookie*
 - Cookie is given by server, stored by browser
 - Client sends that cookie back to the server on subsequent requests
 - Server associates that token with data
- Often the most frequently used data by that user
 - Data that is specific to the user
 - Data that is required for rendering or common use
- Often ephemeral and duplicated

Session Storage Uses Cases

Traditional

- Username
- Preferences
- Name
- “Stateful” data

Intelligent

- Traditional +
- Notifications
- Past behavior
 - content surfacing
 - analytical information
 - personalization

In a simple world



Internet

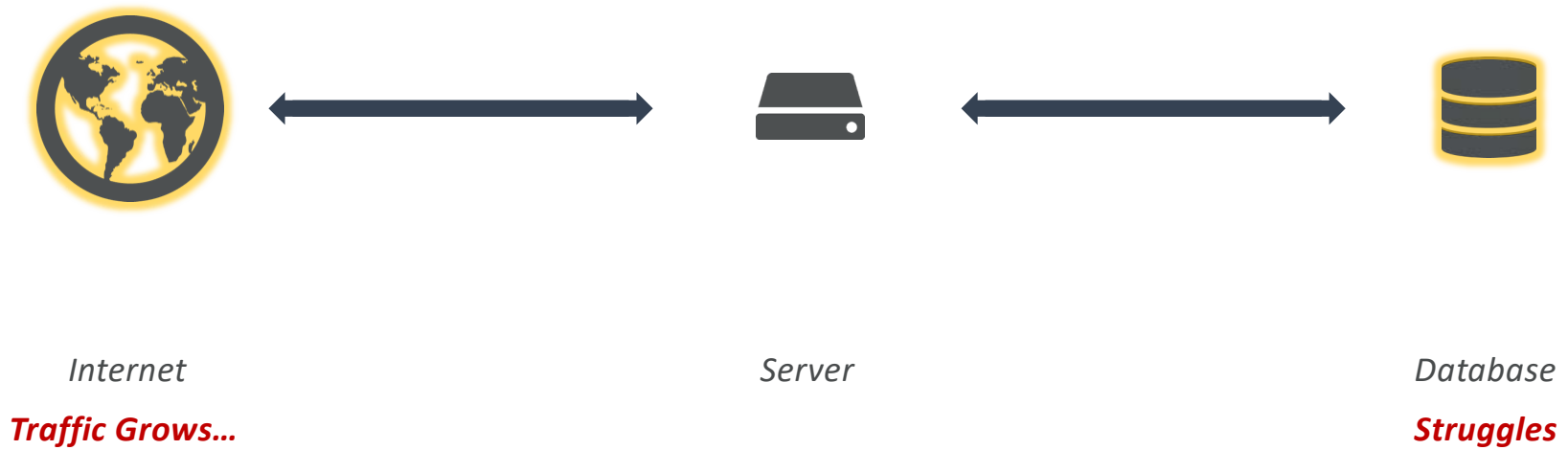


Server

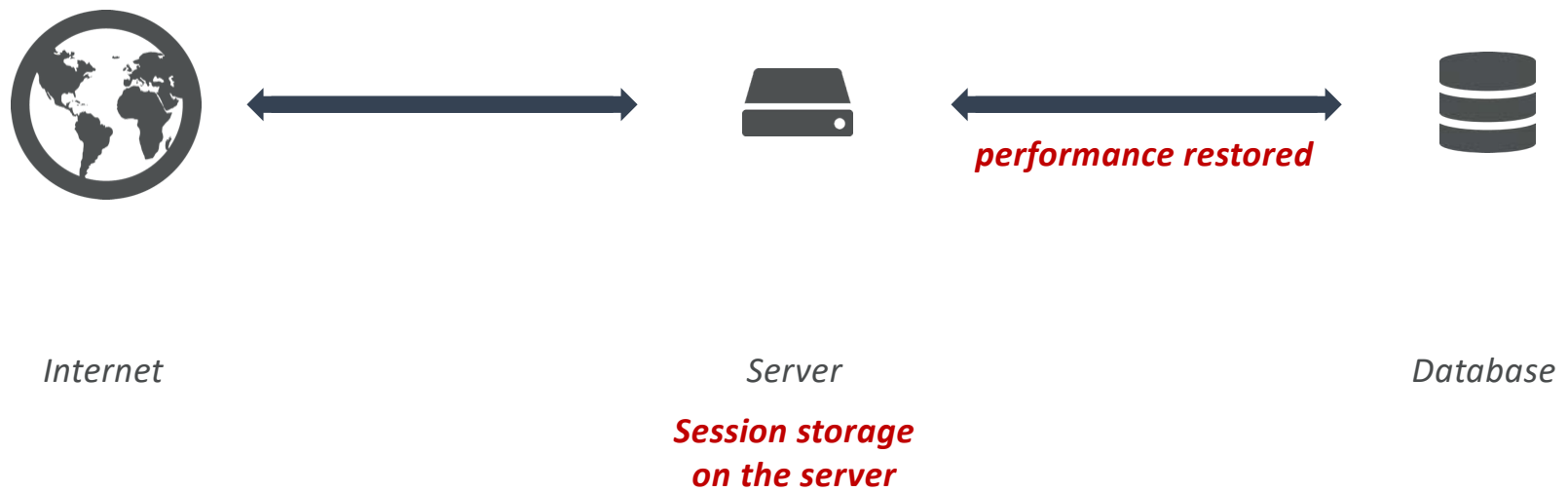


Database

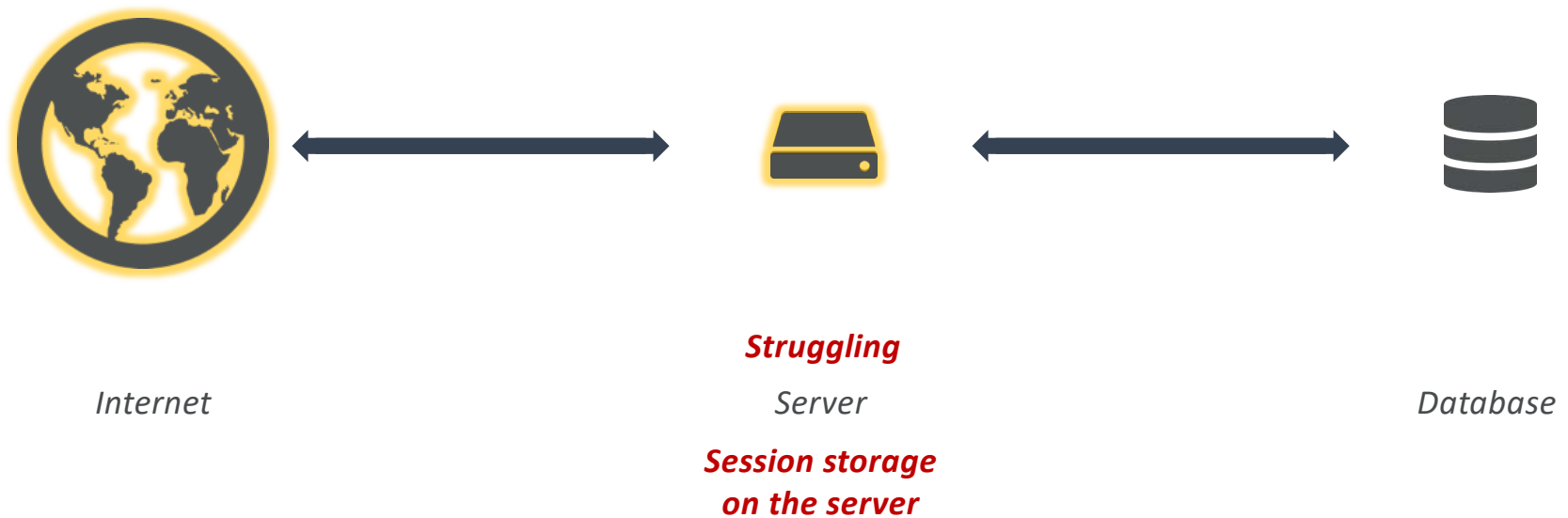
Good problems



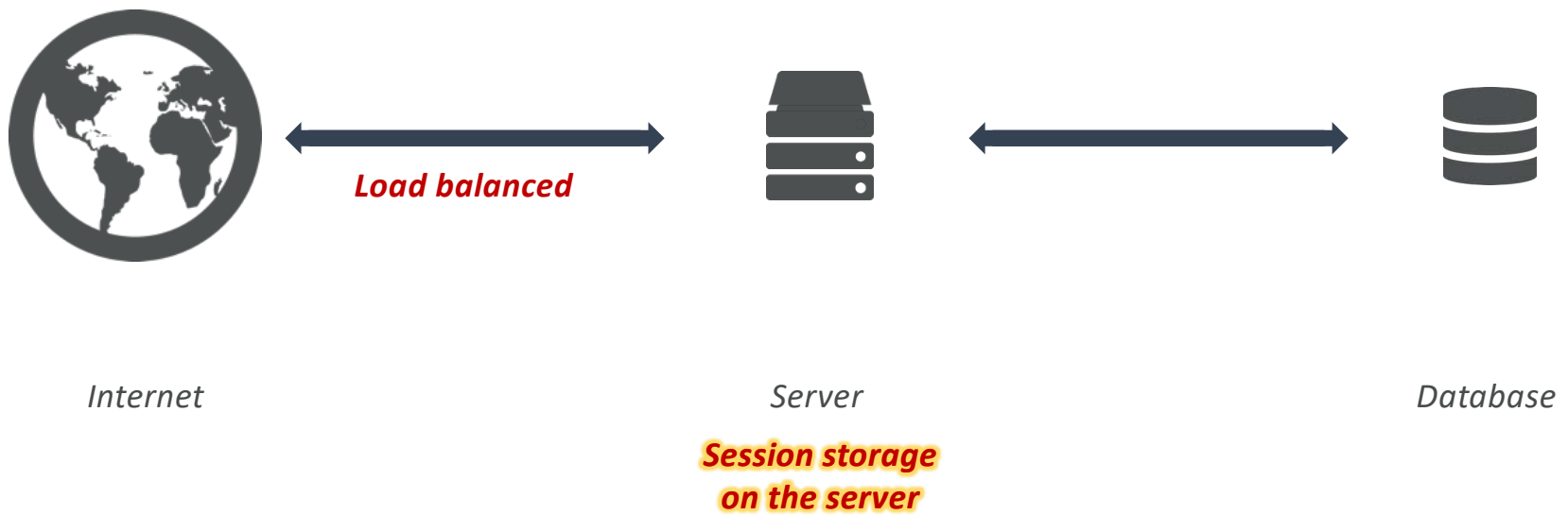
Good solution



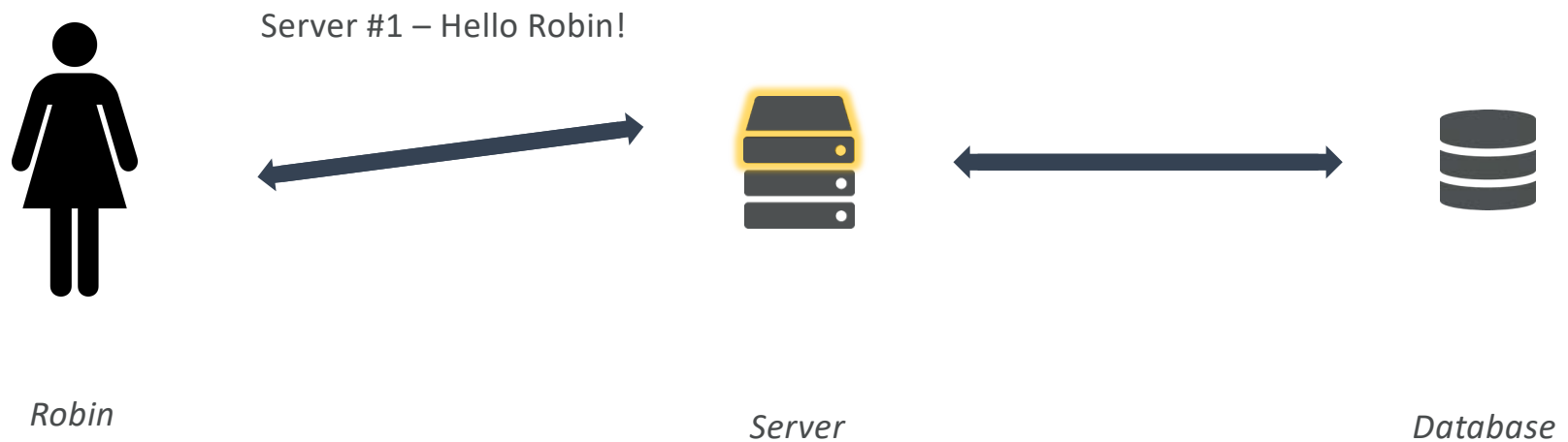
More good problems



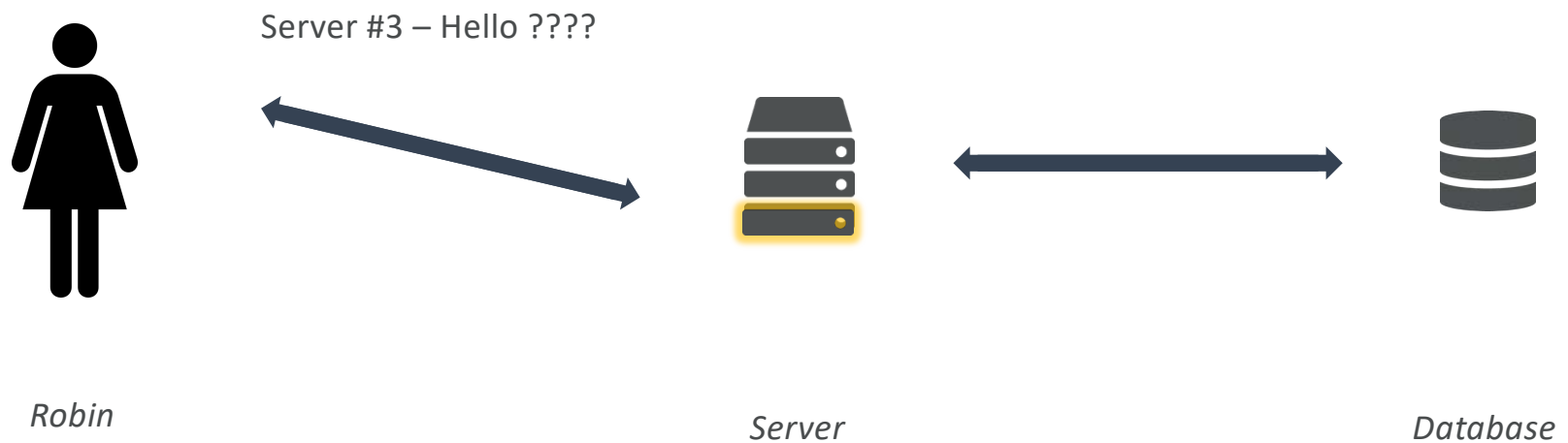
Problematic Solutions



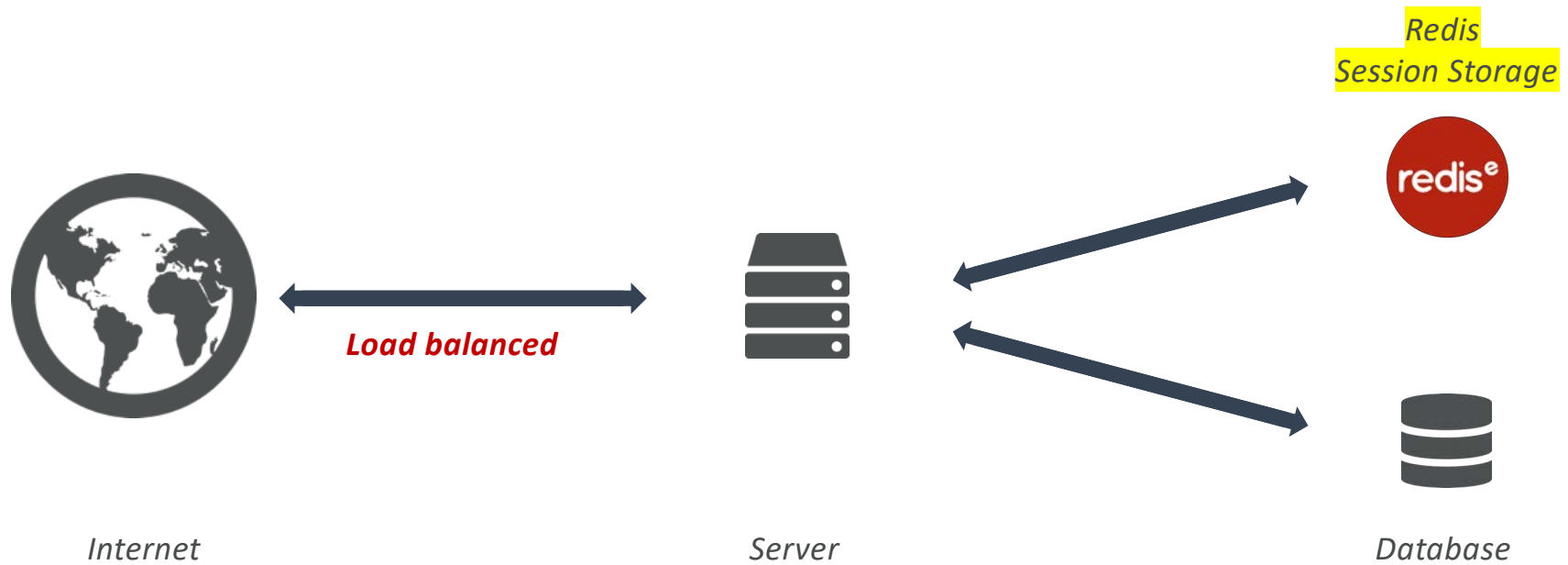
Multiple Servers + On-server Sessions?



Multiple Servers + On-server Sessions?



Better solution



User Session Store

- The Problem
- Maintain session state across multiple servers
- Multiple session variables
- High speed/low latency required

Why Redis Rocks

- **Hashes** are perfect for this!
- HSET lets you save session variables as key/value pairs
- HGET to retrieve values
- HINCRBY to increment any field within the hash structure

Redis Hashes Example - <https://redis.io/commands#hash>

hash key: usersession:1

userid	8754
name	dave
ip	10:20:104:31
hits	1
lastpage	home

```
HMSET usersession:1 userid 8754 name dave ip 10:20:104:31 hits 1
HMGET usersession:1 userid name ip hits
HINCRBY usersession:1 hits 1
```

```
HSET usersession:1 lastpage "home"
HGET usersession:1 lastpage
HDEL usersession:1 lastpage
```

```
EXPIRE usersession:1 10
```

or

```
DEL usersession:1
```

Hashes store a mapping of keys to values – like a dictionary or associative array – but faster

Managing Queues w/ Redis Lists



Managing Queues of Work

- The Problem

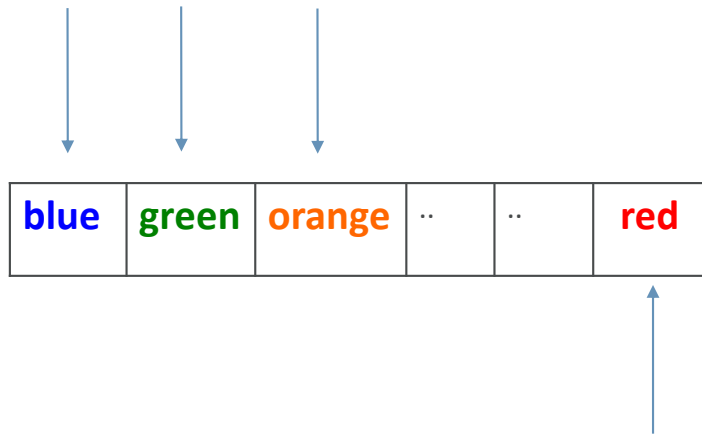
- Tasks need to be worked on asynch to reduce block/wait times
- Lots of items to be worked on
- Assign items to worker process and remove from queue at the same time
- Similar to buffering high speed data-ingestion

Why Redis Rocks

- **Lists** are perfect for this!
- LPUSH, RPUSH add values at beginning or end of queue
- RPOPLPUSH – pops an item from one queue and pushes it to another queue

Redis Lists Example - <https://redis.io/commands#list>

LPUSH adds values to head of list



RPush adds value to tail of list

```
LPUSH queue1 orange
LPUSH queue1 green
LPUSH queue1 blue
RPush queue1 red
```

Redis Lists Example - <https://redis.io/commands#list>

```
Lpush queue1 orange  
Lpush queue1 green  
Lpush queue1 blue  
Rpush queue1 red
```



```
RPOPLPUSH queue1 queue2
```

RPOPLPUSH pops a value from one list and pushes it to another list

```
LLEN queue1  
LINDEX queue1 0  
LRANGE queue1 0 2
```

Managing Tags w/ Redis Sets



Managing Tags Example

- The Problem
- Loads of tags
- Find items with particular tags
- High speed/low latency required

Also used for:

- Recommending Similar Purchases
- Recommending Similar Posts

Why Redis Rocks

- **Sets** are unique collections of strings
- SADD to add tags to each article
- SISMEMBER to check if an article has a given tag
- SMEMBERS to get all the tags for an article
- SINTER to find which articles are tagged with tag1, tag2 and tag77

Redis Sets Example – <https://redis.io/commands#set>

article 1

tag1	tag22	tag24	tag28		
------	-------	-------	-------	--	--

tag 1

article 1	article 3	...		
-----------	-----------	-----	--	--

tag 2

article 3	article 14	article 22	..	
-----------	------------	------------	----	--

```
SADD article:1 tag:1 tag:22 tag:24 tag:28
SADD tag:1 article:1
SADD tag:1 article:3
SADD tag:2 article:22
SADD tag:2 article:14
SADD tag:2 article:3
```

```
SISMEMBER article:1 tag:1
SMEMBERS article:1
```

```
SINTER tag:1 tag:2
```

Managing Leaderboards w/ Redis Sorted Sets



Leaderboard with Sorted Sets Example

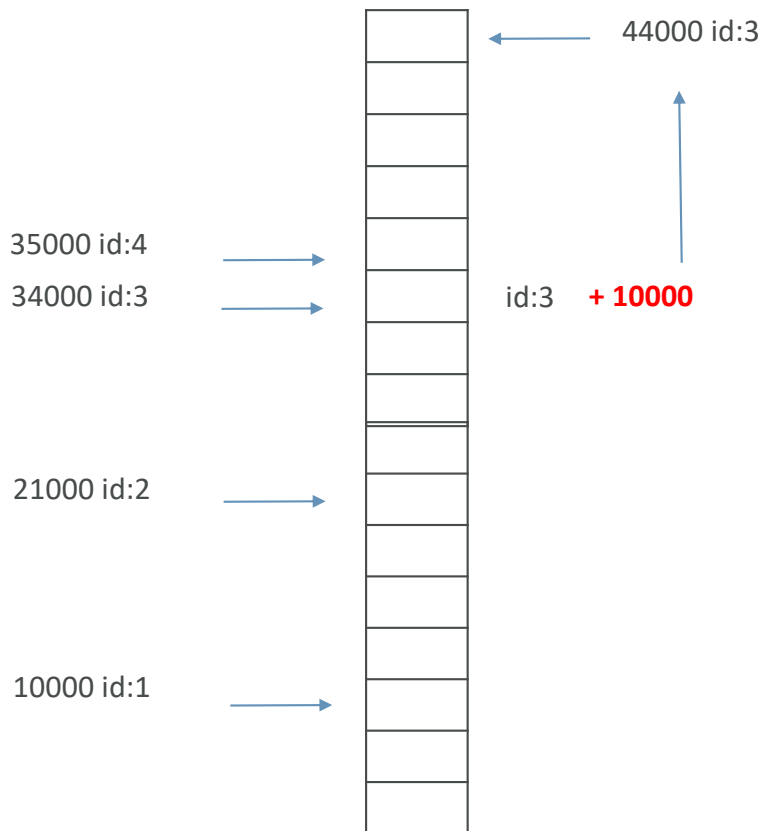
- The Problem

- MANY users playing a game or collecting points
- Display real-time leaderboard.
- Who is your nearest competition
- Disk-based DB is too slow

Why Redis Rocks

- **Sorted Sets** are perfect!
- Automatically keeps list of users sorted by score
- ZADD to add/update
- ZRANGE, ZREVRANGE to get user
- ZRANK will get any users rank instantaneously

Redis Sorted Sets - https://redis.io/commands#sorted_set



```
ZADD game:1 10000 id:1  
ZADD game:1 21000 id:2  
ZADD game:1 34000 id:3  
ZADD game:1 35000 id:4  
ZADD game:1 44000 id:3  
or  
ZINCRBY game:1 10000 id:3
```

```
ZREVRANGE game:1 0 0  
ZREVRANGE game:1 0 1 WITHSCORES
```

Searching within a Geospatial Index



Search within a Geographic Area Example

- The Problem

- MANY moving items within a geographical area
- Display real-time locations.
- What is the nearest item NOW
- SQL Queries are too slow

Why Redis Rocks

- GEOADD to add an item
- Redis Geo uses Sorted Sets so related Z-commands such as ZRANGE & ZREM are useful too
- GEODIST to find distance between to points
- GEORADIUS to find all points within an area

Redis Geospatial Index - <https://redis.io/commands#geo>

RedisConf

-122.3931400 37.7681300

SFO

-122.375 37.618889



Rome

41.9, 12.5

Campobello di Licata

37.25, 13.916667

GEOADD locations 37.25 13.916667 “Campobello di Licata”

GEOADD locations 41.9 12.5 Rome

GEOADD locations -122.375 37.618889 SFO

GEOADD locations -122.3931400 37.7681300 Redisconf

ZRANGE locations 0 -1

GEODIST locations “Campobello di Licata” Redisconf mi

GEOPOS locations Redisconf SFO

GEOHASH locations Redisconf

ZSCORE locations Redisconf

GEORADIUS locations -122.41942 37.77493 15 mi

GEORADIUSBYMEMBER locations Redisconf 15 mi

WITHDIST ASC

ZREM locations SFO

Fire and Forget with Pub/Sub



Fire and Forget with Pub/Sub

- The Problem
- Communicate with clients in real-time

Why Redis Rocks

- PUBLISH
- SUBSCRIBE

Pub/Sub Demo: <https://redis.io/commands#pubsub>

Server Setup

```
$ redis-server  
$ keys *  
$ flushall
```

Subscriber 1 - Steps

```
2. SUBSCRIBE channel1  
5. CONTROL-C  
7. redis-cli  
8. SUBSCRIBE channel1
```

Will receive:

```
4. hello 2  
9. hello world 4
```

Publisher - Steps

```
1. PUBLISH channel1 "hi 1"  
4. PUBLISH channel1 "hello 2"  
6. PUBLISH channel1 "hellooo 3"  
9. PUBLISH channel1 "hello world 4"
```

Subscriber 2 - Steps

```
3. SUBSCRIBE channel1
```

Will receive:

```
4. hello 2  
6. hellooo 3  
9. hello world 4
```

Reliable messaging with Redis Streams



Reliable messaging with Redis Streams

- The Problem

- Communicate with clients in real-time without missing data

Why Redis Rocks

- XADD
- XREAD

Redis Streams Demo - <https://redis.io/commands#stream>

Server Setup

```
redis-server  
keys *  
flushall
```

Producer - Steps

```
1. XADD mystream1 * greeting "hello 1"  
5. XADD mystream1 * greeting "hello 2"  
8. XADD mystream1 * greeting "hello 3"
```

Consumer 1 - Steps

```
2. XREAD COUNT 10 STREAMS mystream1 0  
4. CONTROL-C  
6. redis-cli  
9. XREAD COUNT 10 STREAMS mystream1 0
```

Will receive:

1. hello 1
5. hello 2
8. hello 3

Consumer 2 - Steps

```
3. XREAD COUNT 10 STREAMS mystream1 0  
7. XREAD COUNT 10 STREAMS mystream1 0  
10. XREAD COUNT 10 STREAMS mystream1 0
```




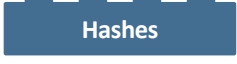
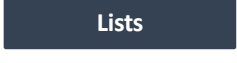


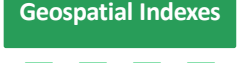


Will receive:

1. hello 1
5. hello 2
8. hello 3

More examples



Redis Data Structures

1.  Strings - Cache SQL queries, pages, fast data
2.  Bitmaps - Store 1s/0s (Online/Offline) like Pinterest
3.  Bit fields - Store arrays of complex numbers
4.  Hashes - Store record-like info (user sessions, favorites, etc.)
5.  Lists - Store ordered data (for ingesting data)
6.  Sets - Store unique/unordered data (tags, unique IPs, etc.)
7.  Sorted Sets - Store real-time sorted data like a MMPORG Leaderboard
8.  Geospatial Indexes - Store real-time location (Uber, Lyft, etc.)
9.  Hyperloglog - Store probabilistic data (Google Search count)
10.  Streams - Store and share event data (similar to Kafka)

In-memory, Keys & Data Structures



Redis Keys – Ground Rules

- It's all about the keys
- No Querying
- No Indexes
- No Schemas
- Keys must be unique (like primary keys in an SQL database)

Thank you!
dave@redislabs.com



redislabs
HOME OF REDIS

