redislabs
HOME OF REDIS

Redis Enterprise Developer Workshop

Early OCT 2018 – YYZ / BOS / TXL / CDG ONLY

Agenda

8:00 -9:00 am	Registration and Breakfast
9:00 -10:30 am	Intro to Redis + Redis Data Structures
10:30 -10:45 am	Break
10:45 -11:15 am	Transaction & Concurrency Model
11:15 am-Noon	Redis Enterprise Architecture & Demo
Noon-1:00 pm	Lunch
1:00-2:30 pm	Hands-on Lab
2:30-3:00 pm	Redis Enterprise Replication, HA & Active-Active
3:00-3:30 pm	Lua
3:30-4:30 pm	Modules

Setup Dev Environment

- Python 3.0
- Install required Python
  - modules: redis, flatten_json
- Download hands-on code:

http://bit.ly/red-workshop-oct-18

redislabs
HOME OF REDIS

Intro to
Redis & Redis Enterprise

Who We Are

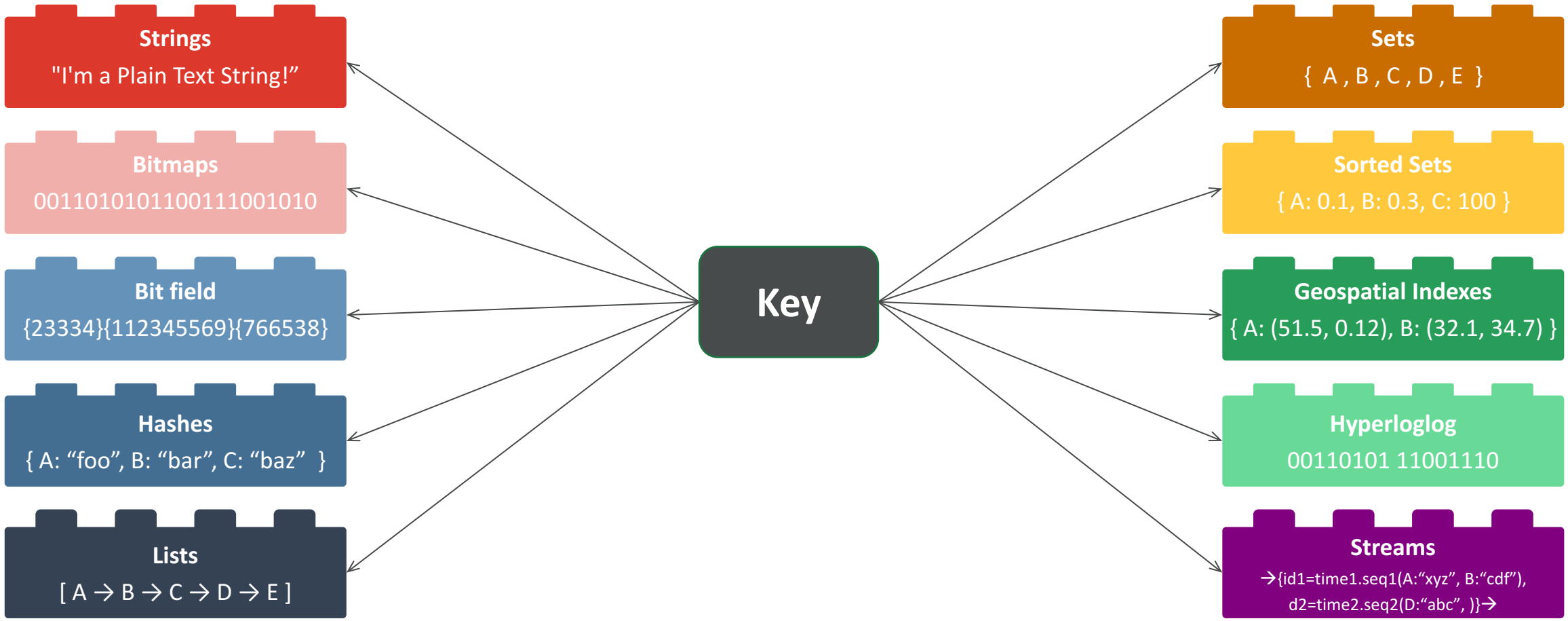
Open source. The leading **in-memory database platform**, supporting any high performance operational, analytics or hybrid use case.

The open source home and commercial provider of **Redis Enterprise** technology, platform, products & services.

What is Redis?

- Redis – (**RE**mote **DI**ctionary **S**erver)
- **Open source**: BSD
- The leading **in-memory database platform**
- **Created** in 2009 by Salvatore Sanfilippo (a.k.a @antirez)
- **Source**: https://github.com/antirez/redis
- **Website**: http://redis.io

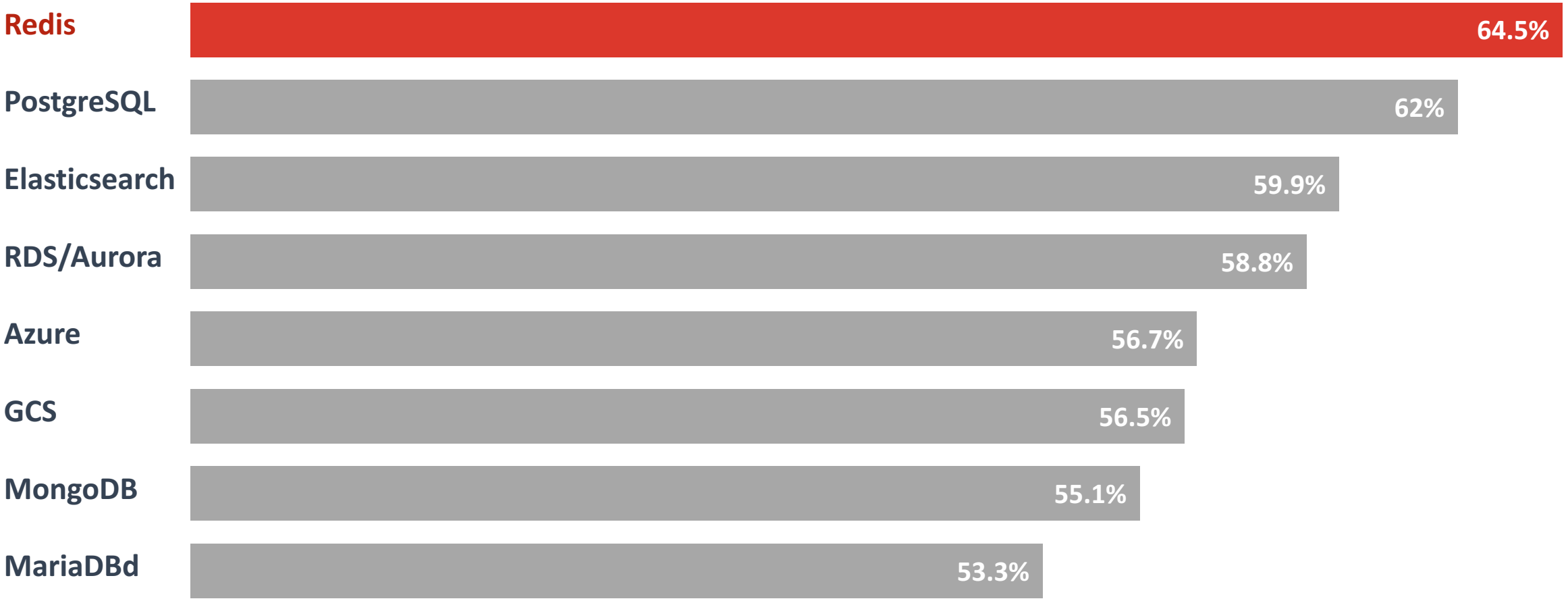
Data Structures - Redis' Building Blocks



"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***

redislabs
HOME OF REDIS

Stack Overflow Survey: The Most Loved Databases 2018

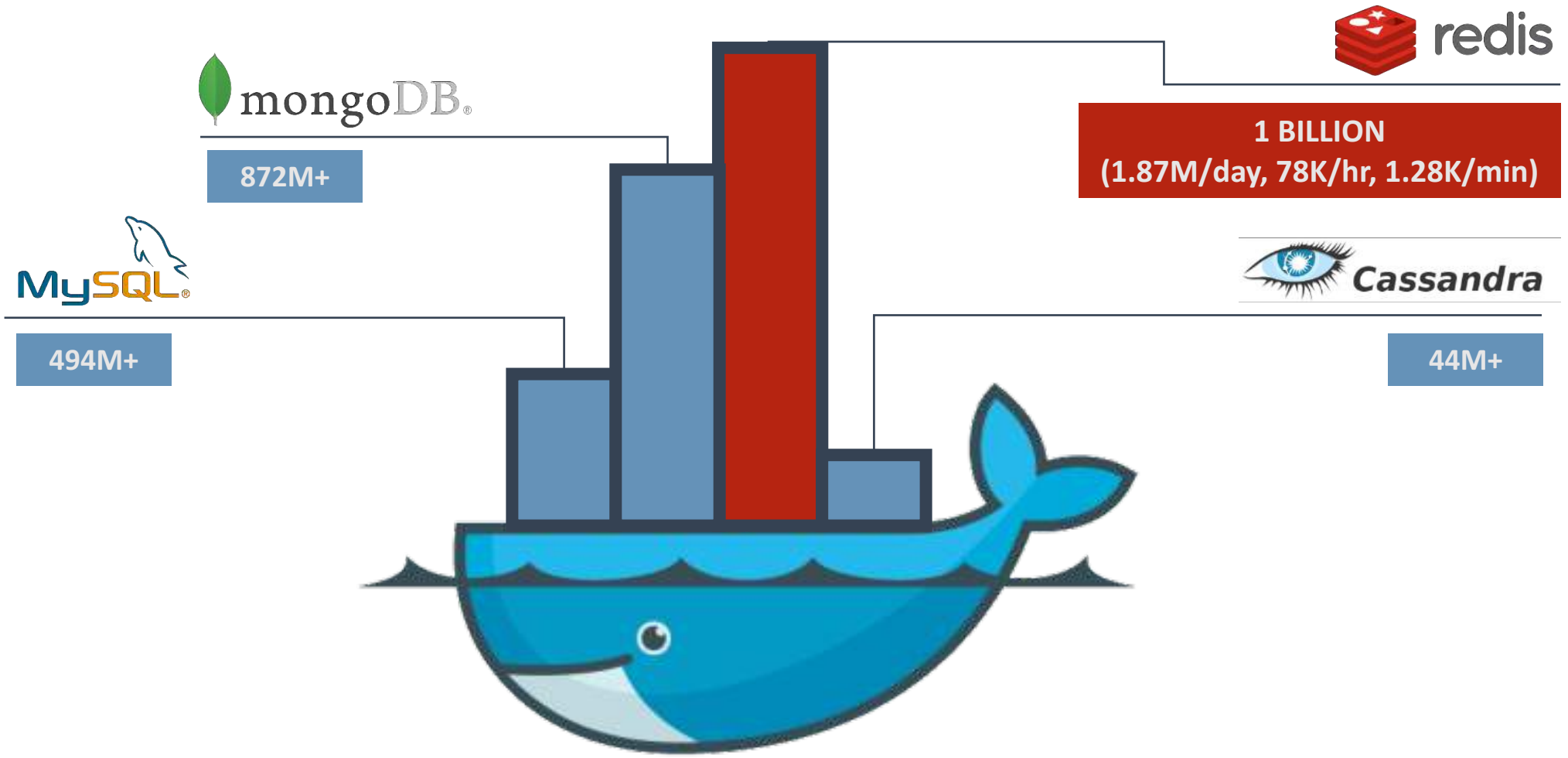


% of devs who expressed interest in continuing to develop with a language/tech

redislabs
HOME OF REDIS

Docker Hub: The Most Popular Database Container

# of containers launched as of Sept 2018



Redis Tops Database Popularity Rankings

#1 NoSQL on AWS	sumologic	#1 Growth Among Top 3 NoSQL Databases	DB-ENGINES	#1 Database on Docker (1BIL+ Downloads)	DATADOG
node.js	#1 Database Used by Node.js Developers	G2 CROWD	#1 NoSQL Database in User Satisfaction	UpScored	#1 Database in Skill Demand
Top B2B Silicon Valley Tech	G2 CROWD	Most Loved Database	stackoverflow	Top 50 Developer Tools	stackshare

redislabs
HOME OF REDIS

Industry Recognition

FORRESTER®

Forrester Names Redis Labs a Leader in the Forrester Wave: Translytical Data Platforms, 2017

Redis Labs recognized by customers for its business value, support for broad use cases, performance and customer support.

FORRESTER®
WAVE
LEADER 2017
Translytical Databases

redislabs
HOME OF REDIS

The Redis Community

redislabs
HOME OF REDIS

Redis Enterprise Customers Span All Verticals

Banks	Financial Services	E-commerce	Social	Media	Advertising
American Express, Crédit Agricole, Charles Schwab, Wells Fargo	MasterCard, Visa, Intuit, Fiserv	Walmart, eBay, The Home Depot, Starbucks	Twitter, Twitch, Shutterfly	MSN, DreamWorks Animation SKG, The Motley Fool, Bleacher Report	Havas, RevMob, Outbrain

Technology	Communications	Business Services	Travel	Gaming	Education
Apple, Cisco, Dell	Comcast, AT&T, Verizon, Vodafone	Ariba, Atlassian, RingCentral	Expedia, priceline.com, Hotel Tonight, TripAdvisor	Electronic Arts, Playtika, Roblox	its learning, HealthStream, Edgenuity

redislabs
HOME OF REDIS

Redis is Extensively & Diversely Used 1/2

Company	Use Case	Scope
Twitter	Timeline, follower, following	0.5-1PB, 30 MM ops/sec
Weibo (Chinese Twitter)	Entire database	+ 100 TB, 10MM ops/sec
Samsung US	Fast data store for mobile apps	50MM users, <100 msec (E2E)
HTC	Fast data store for mobile apps	40TB
Pinterest	Graph database	10+TB
Stack overflow	Local/site/global caching	

Redis is Extensively & Diversely Used 2/2

Company	Use Case	Scope
Booking.com	Online bookings/fast transactions	10-20 TB
Github	Repository router	10+ TB
Tinder	Geo search, user profiles	10-20TB
Snapchat	All messages	40TB

Redis Enterprise Deployment Options

Cloud

Fully managed, server-less and hosted Redis Enterprise database-as-a-service on hosted resources in AWS, MS Azure, GCP, IBM Softlayer, Heroku, PWS

RAM

VPC

Fully managed Redis Enterprise database-as-a-service in your VPCs within AWS, MS Azure, GCP and IBM Softlayer

RAM or Flash

Software

Redis Enterprise software for any cloud or private data center. Downloadable, in containers, on PCF or as an AWS AMI.

RAM or Flash

redislabs
HOME OF REDIS

Redis Enterprise DBaaS: Offered Over IaaS and PaaS

redislabs
HOME OF REDIS

redislabs
HOME OF REDIS

Why Redis?

Redis Powers a Range of Solutions

…AND MANY MORE

Redis is Uniquely Suited to Modern Apps

A full range of capabilities that simplify and accelerate next generation applications

redislabs
HOME OF REDIS

Versatility: Redis Uses Span Many Verticals

Telecommunications

Billing (CDRs, SDRs)

Finance

High-speed Delivery of Prices and Transactions

Business Services

CRM, ERP

Retail/E-commerce

Items Viewed, Similar Purchases, Top Trends

Technology

High-speed Operations

Advertising

Real-time Ad Placements, Personalization

Travel

Recommendations, Online Booking

Media

Notifications, Recommendations, Caching

Education

Subjects, Classes Classification

Social Networking

Timeline, Social Graph, Top Followers, Following

Gaming

Real-time Analytics for Leaderboards, Dashboards and Messaging

Real-Time Transactions are Needed in….

Retail

Payment processing

Inventory management

Supply chain management

Finance

Real-Time trading

Money transfer
and disbursement

Loan management

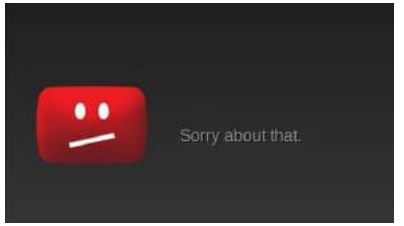
E-Commerce

Order processing

Order fulfillment

Online Payments

Entertainment



Digital rights management

Digital asset management

Ticketing

Travel and Leisure

Reservations

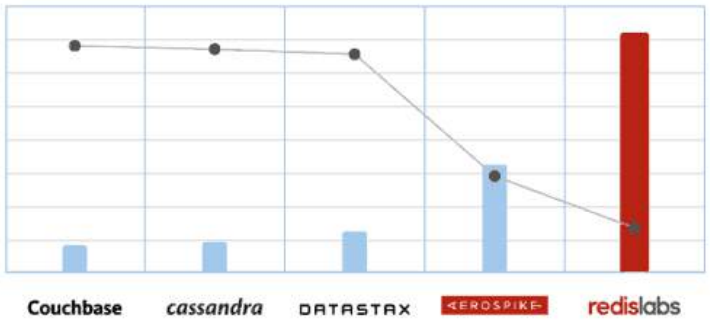
Inventory management

Real-time dispatching

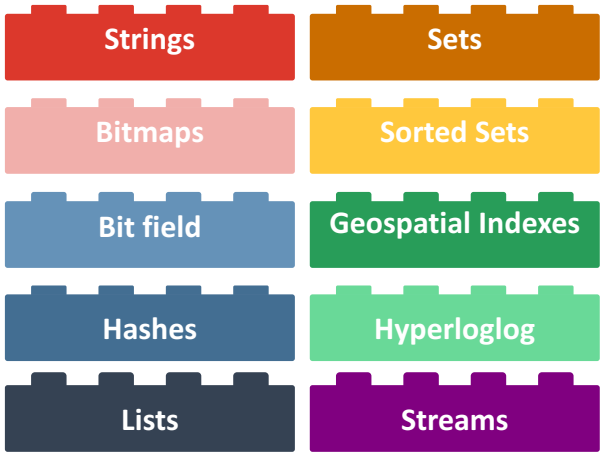
redislabs
HOME OF REDIS

Redis is a game changer.

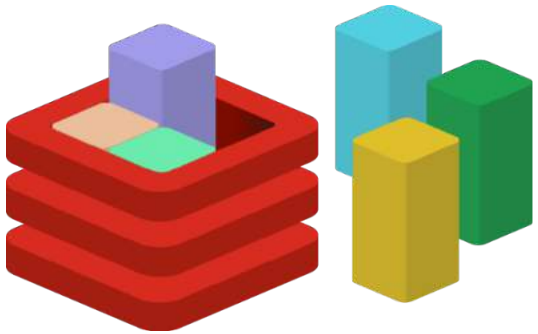
Performance



Simplicity



Extensibility



redislabs
HOME OF REDIS

Performance: Built for Speed

OPTIMIZED ARCHITECTURE

- ✓ Written in C
- ✓ Served entirely from memory
- ✓ Single-threaded, lock free

ADVANCED PROCESSING

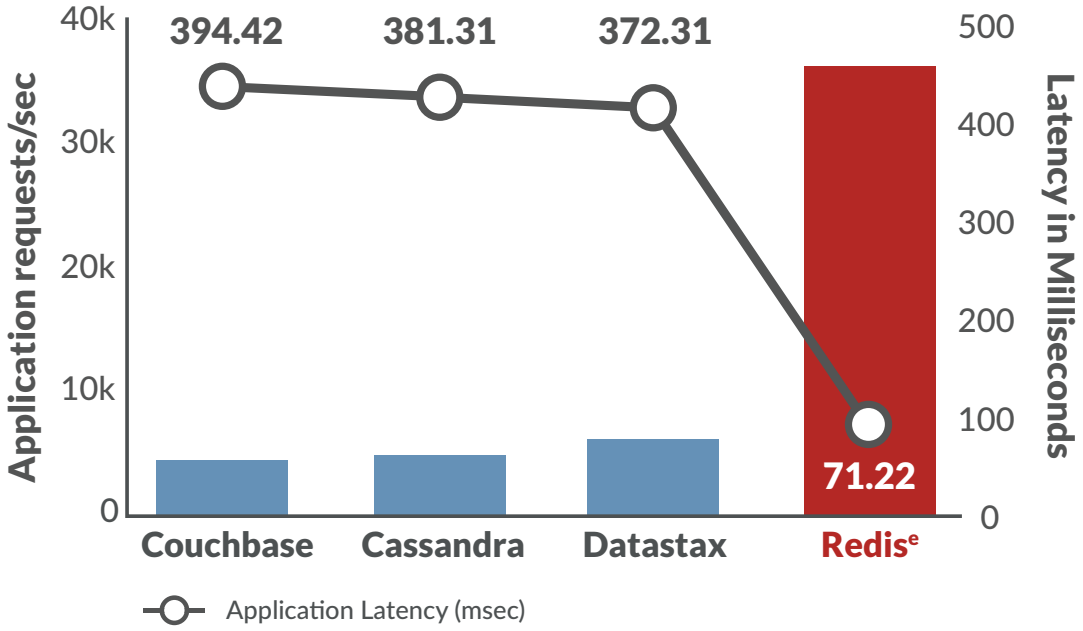
- ✓ Most commands are executed with O(1) complexity
- ✓ Access to discrete elements within objects
- ✓ Reduced bandwidth/overhead requirements

EFFICIENT OPERATION

- ✓ Easy to parse networking protocol
- ✓ Pipelining for reduced network overhead
- ✓ Connection pooling

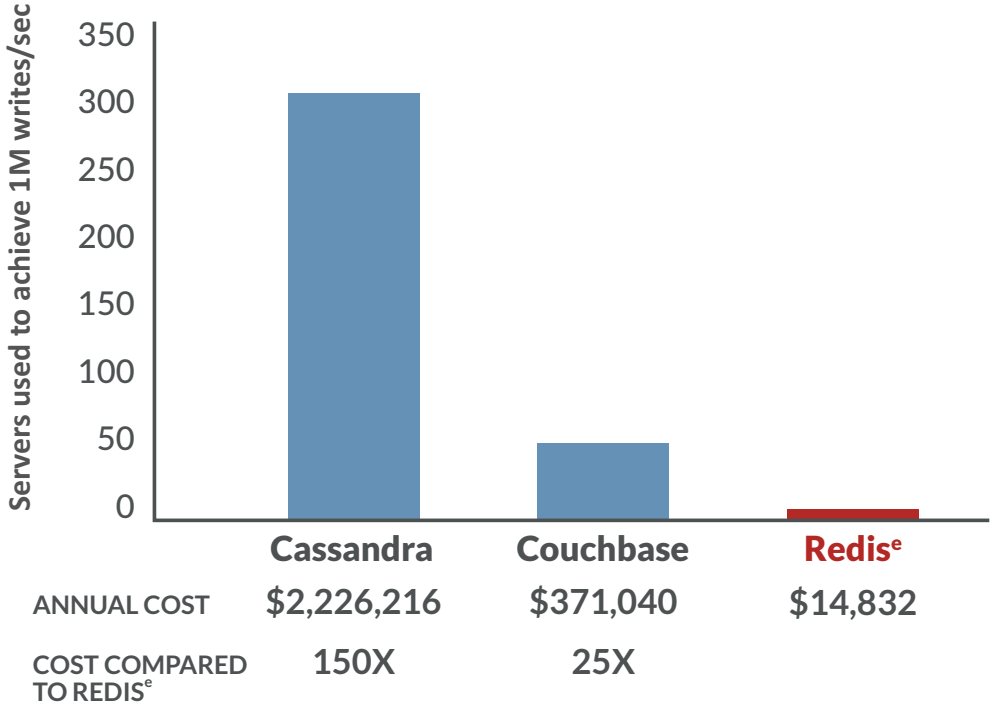
Performance: The Most Powerful Database

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

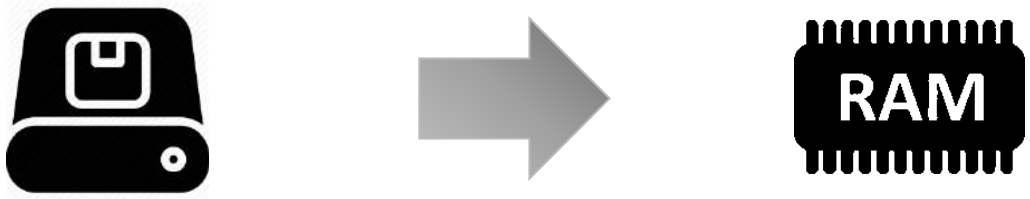
redislabs
HOME OF REDIS

Performance: Effective in OLTP and OLAP

OLTP

The 100 msec de-facto standard for end-to-end app response time requires <1msec DB response time.
Redis is the only database that can support this under heavy load.

OLAP



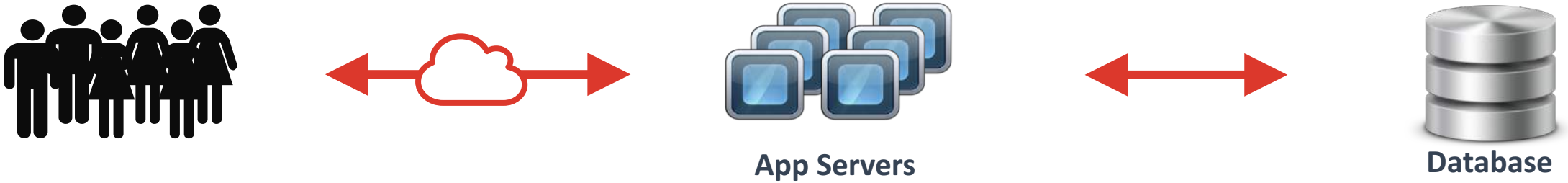
Disk-based

RAM-based

Query time: **Days/Hours**

Query time: **Minutes/Seconds**

Performance: Why use Redis as an operational DBMS?



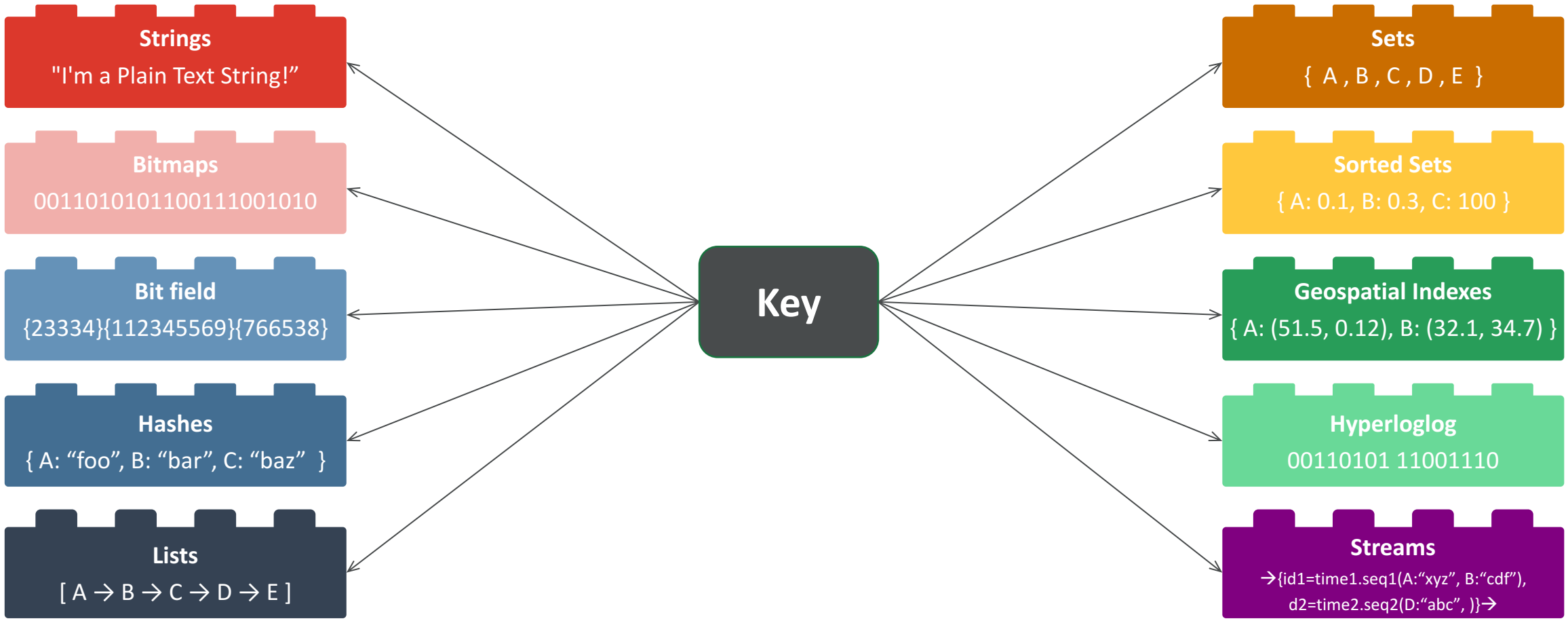
1 msec - required DB response time

50 msec - Required round trip app response time (includes processing & multi-DB access)

50 msec - Average roundtrip internet latency

100 msec - The new standard for E2E application response time, under any load

Data Structures - Redis' Building Blocks – Lego for your App



"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***

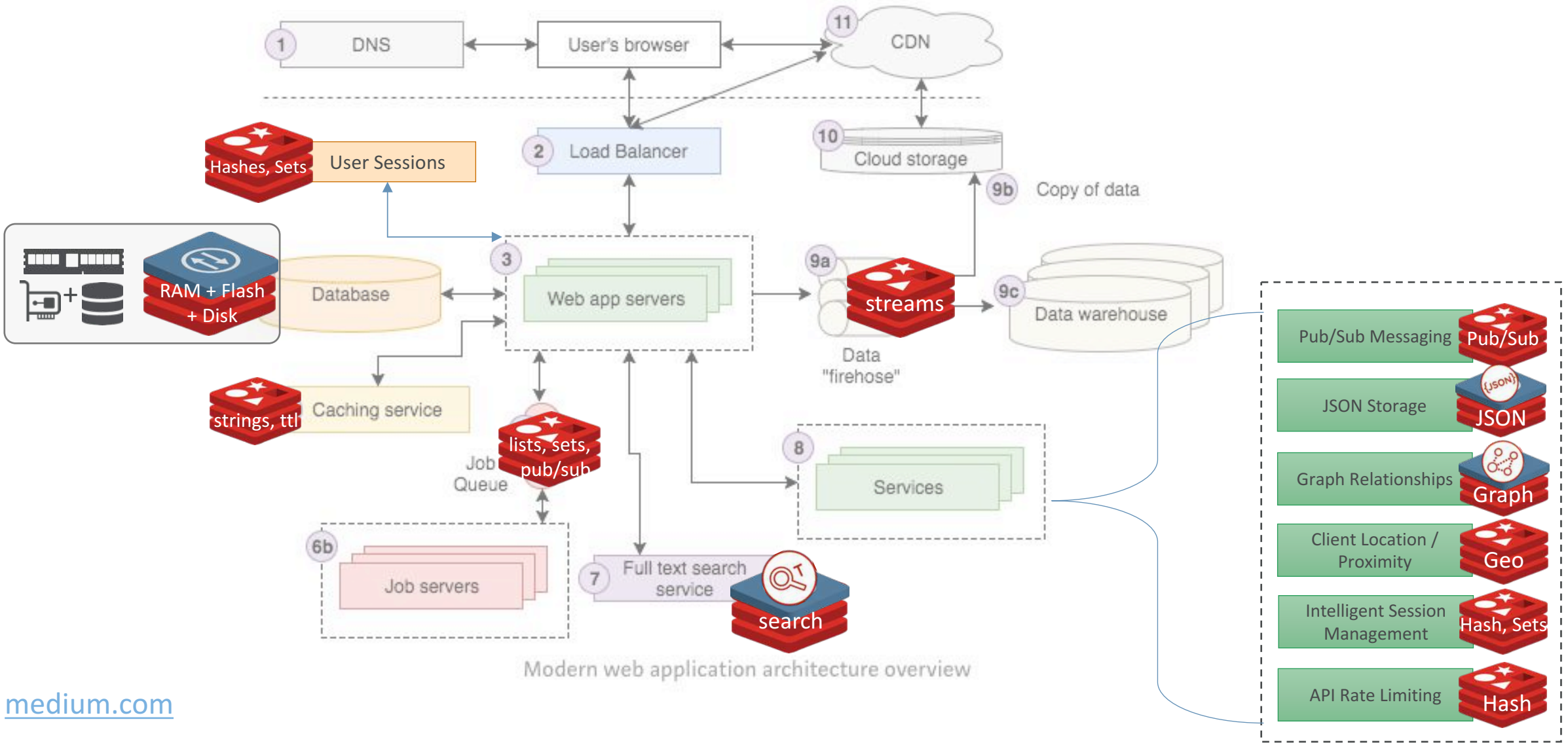
redislabs
HOME OF REDIS

What Can You Do With Redis?

Use as in-memory database, cache or message broker

- Common Uses
  - Cache
  - Message Brokers/Queues
  - User Sessions
  - Real-time Recommendation Engine
  - Leaderboards
  - …More…..much, MUCH more!!!

Web App Architecture 101



medium.com

Simple Cache

The Problem

- Multiple database calls create impossibly slow web page response times

Why Redis Rocks

- *Strings* are perfect for this!
- **SET** lets you save session variables as key/value pairs
- **GET** to retrieve values

Redis Strings for Simple Cache

- Strings store text, which might be made from the results of multiple database queries and HTML
- Can have expiry
- You can register to listen for changes on keys and operations
- Multiple eviction policies supported

```
$ SET userid:1 “8754”
$ GET userid:1
$ EXPIRE userid:1 60
$ DEL userid:1
```

```
jedis.set("userid:1", "8754");
jedis.get("userid:1");
jedis.expire("userid:1", 60);
jedis.del("userid:1");
```

User Sessions

The Problem

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

Why Redis Rocks

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

Redis Hashes for User Sessions

hash key: usersession:1

userid	8754
name	dave
ip	10:20:104:31
hits	2
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
```

```
$ DEL usersession:1
```

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

Redis Hashes for User Sessions

```
Map<String, String> userSession = new HashMap<>();
userSession.put("userid", "8754");
userSession.put("name", "dave");
userSession.put("ip", "10:20:104:31");
userSession.put("hits", "1")
jedis.hmset("usersession:1", userSession);

jedis.hmget("usersession:1", "userid", "name","ip" ,"hits");
jedis.hincrBy("usersession:1", "hits", 1);
jedis.hset("usersession:1", "lastpage", "home");
jedis.hget("usersession:1", "lastpage");
jedis.hdel("usersession:1", "lastpage");
```

Managing Queues of Work

The Problem

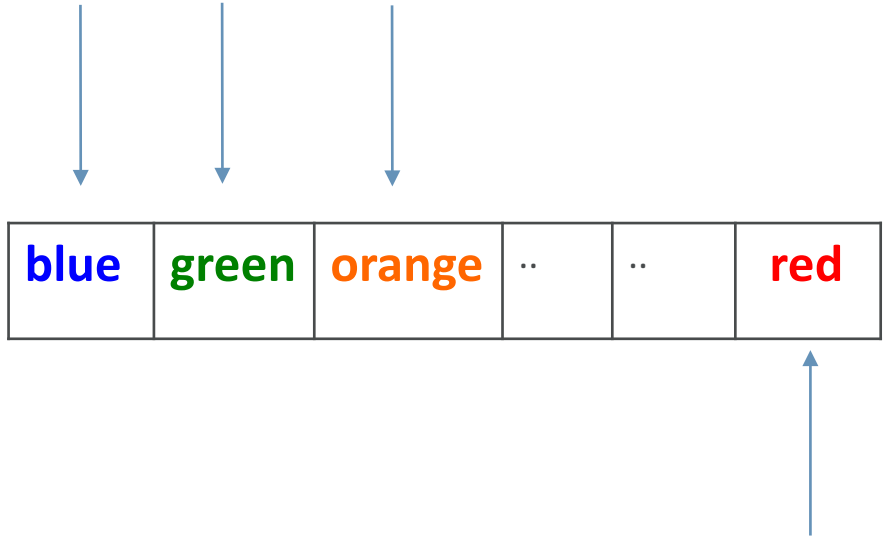
- Tasks need to be worked on async
  to reduce block/wait times
- Lots of items to be worked on
- Assign items to worker process and remove from queue at the same time
- Similar to buffering high speed data-ingestion
- High speed/low latency required

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 for Managing Queues

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 for Managing Queues

```
$ 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

Redis Lists for Managing Queues

```
jedis.lpush("queue1", "orange");
jedis.lpush("queue1", "green");
jedis.lpush("queue1", "blue");
jedis.rpush("queue1", "red")

jedis.rpoplpush("queue1", "queue2");
```

Real-time Recommendation Engine

The Problem

- People who read this article also read these other articles
- Want real time not data mining

Also used for:

- Recommending Similar Purchases
- Identifying Fraud

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
- use **SINTER** to find similar articles tagged with the same tags

Redis Sets for Recommendations

Set: tag:1

article 1	**article 3**	….		

Set: tag:2

article 3	article 14	Article 22	..	

Set: tag:3

article 2	**article 3**	article 9	..	

Add values (articles) to Sets (tags)

```
$ SADD tag:1 article:3 article:1
$ SADD tag:2 article:22 article:14 article:3
$ SADD tag:3 article:9 article:3 article:2
```

Confirm the values have been added

```
$ SMEMBERS tag:3      (also tag:1 & tag:2)
```

```
1) "article:3"
2) "article:2"
3) "article:9"
```

Find values that exist in all three Sets

```
$ SINTER tag:1 tag:2 tag:3
```

```
1) "article:3"
```

redislabs
HOME OF REDIS

Redis Sets for Recommendations

```
jedis.sadd("tag:1", "article:3" ,"article:1");
jedis.sadd("tag:2", "article:22","article:14", "article:3");
jedis.sadd("tag:3", "article:9", "article:3", "article:2");

jedis.smembers("tag:1")
jedis.sinter("tag:1", "tag:2", "tag:3");
```

Example : Redis For Bid Management

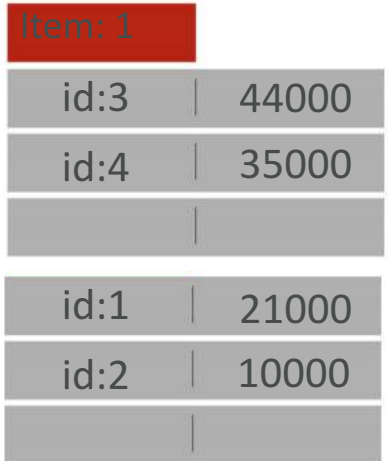
The Application Problem

- Many users bidding on items
- Need to instantly show who's leading, in what order and by how much
- May also need to display analytics like how many users are bidding in what range
- Disk-based DBMS-es are too slow for real-time, high scale calculations

Why Redis Rocks This

- Sorted sets automatically keep list of users and scores updated and in order (ZADD)
- ZRANGE, ZREVRANGE will get your top users
- ZRANK will get any users rank instantaneously
- ZCOUNT will return a count of users in a range
- ZRANGEBYSCORE will return all the users in a range by their bids

Redis Sorted Sets




```
ZADD item:1 10000 id:2 21000 id: 1
ZADD item:1 34000 id:3 35000 id 4
ZINCRBY item:1 10000 id:3
```

```
ZREVRANGE item:1 0 0
id:3
```

```
jedis.zadd("item:1" , 10000 ,"id:2" );
jedis.zadd("item:1" , 21000 ,"id:1" );
jedis.zadd("item:1" , 34000 ,"id:3" );
jedis.zadd("item:1" , 35000 ,"id:4" );
jedis.zincrby("item:1",10000, "id:3");

jedis.zrevrange("item:1", 0,0);
```

Sorted Sets for Leaderboards

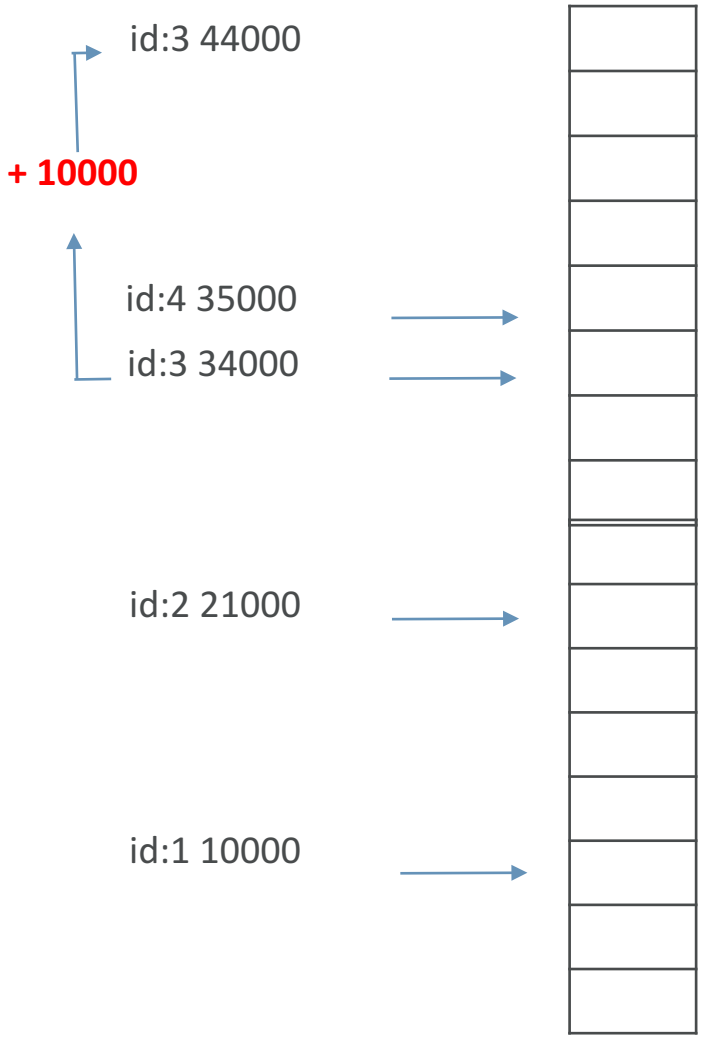
The Problem

- 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




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

Get the Leader Board

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

Redis Sorted Sets

```
jedis.zadd("game:1", 10000, "id:1" );
jedis.zadd("game:1", 21000, "id:2");
jedis.zadd("game:1", 34000, "id:3" );
jedis.zadd("game:1", 35000, "id:4");

jedis.zincrby("game:1", 10000,"id:3");

jedis.zrevrange("game:1", 0,0);
jedis.zrevrangeWithScores("game:1", 0,1);
```

Search By Location

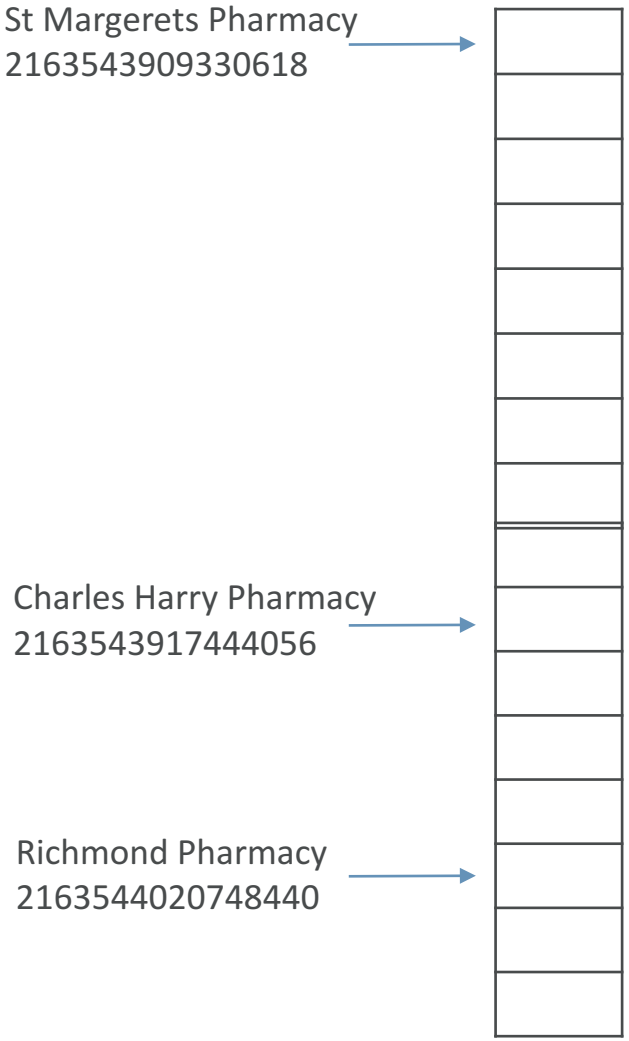
The Problem

- Give me all the pharmacies in 2 km radius
- How far am I from the hospital

Why Redis Rocks

- *GeoSet* is perfect!
- Stores location as Geohash
- **GEOADD** to add a location
- **GEODIST** to get distance
- **GEORADIUS** to get locations in radius

Search By Location




```
GEOADD pharmacies -0.310392 51.456454  "Charles Harry Pharmacy”
GEOADD pharmacies -0.296402 51.462069 "Richmond Pharmacy”
GEOADD pharmacies -0.318604 51.455338 "St Margerets Pharmacy”
```

```
GEORADIUS pharmacies -0.30566239999996014 51.452921 600 m WITHDIST WITHCOORD ASC
1) 1) "Charles Harry Pharmacy"
   2) "511.6979"
   3) 1) "-0.31039327383041382"
   2) "51.456452884598633 09"
```

redislabs
HOME OF REDIS

Search By Location

```
jedis.geoadd("pharmacies", -0.310392, 51.456454, "Charles Harry
Pharmacy");
jedis.geoadd("pharmacies",-0.296402, 51.462069, "Richmond
Pharmacy");
jedis.geoadd("pharmacies", -0.318604, 51.455338, "St Margerets
Pharmacy");

jedis.georadius("pharmacies", -0.30566239999996014, 51.452921,
600 , GeoUnit.M ,
GeoRadiusParam.geoRadiusParam().withCoord().withCoord().withDist(
));
```

Count Unique Visitors

The Problem

- Count unique daily visitors to the site
- How many unique users have clicked on an ad

Why Redis Rocks

- *HyperLogLog* is perfect!
- Keeps Count of each unique element
- **PFADD** to add an element
- **PFCOUNT** to get count

HyperLogLog to Count Unique Visitors

- Stored as String

"HYLL\x01\x00\x00\x00\x04\x00\x00
x00\x00\x00\x00\x00E\xa0\x80S1\x8
4E2\x88\\\x17\x80E\xdd”

- Maximum 12 KB size
- Standard error of 0.81%.

```
PFADD visitors:20160921 86.163.34.208
PFADD visitors:20160921 52.203.210.236
PFADD visitors:20160921 54.87.203.132
PFADD visitors:20160921 54.87.201.121
PFADD visitors:20160921 52.203.210.236
```

```
PFCOUNT visitors:20160921
(integer) 4
```

HyperLogLog to Count Unique Visitors

```
jedis.pfadd("visitors:20160921", "86.163.34.208");
jedis.pfadd("visitors:20160921", "52.203.210.236");
jedis.pfadd("visitors:20160921", "54.87.203.132");
jedis.pfadd("visitors:20160921", "54.87.201.121");
jedis.pfadd("visitors:20160921", "52.203.210.236");

jedis.pfadd("visitors:20160921");
```

Sending information to multiple places

The Problem

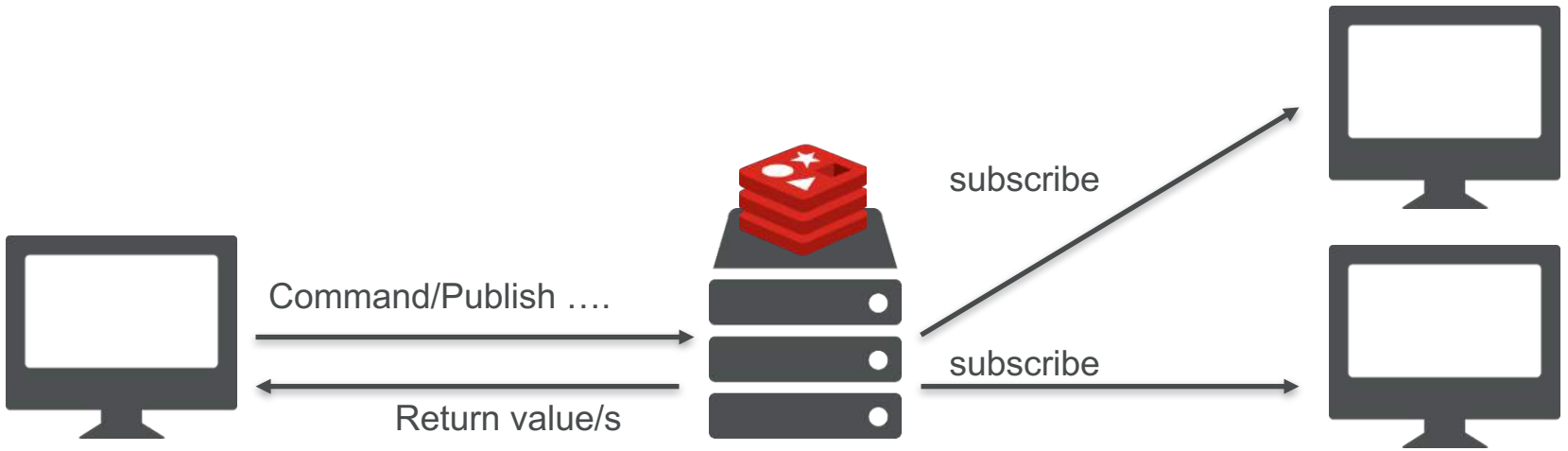
- IoT device sending sensor information to multiple services
- App sending out messages about activities to multiple users

Why Redis Rocks

- *Pub/Sub is the way to go*
- Lightweight way of distributing messages
- No polling, it waits for messages
- **PUBLISH** to send messages
- **SUBSCRIBE** to get messages

Broadcasting messages with Publish/Subscribe

- Redis is a **Pub/Sub server**
  - Much like a topic subscription
  - Fire and forget



```
127.0.0.1:6379> PUBLISH weather:stockholm "7C, Cloudy”
(integer) 2
127.0.0.1:6379> PUBLISH weather:madrid "28C, Sunny”
(integer) 1
```

```
127.0.0.1:6379> SUBSCRIBE weather:stockholm
weather:madrid
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "weather:stockholm"
3) (integer) 1
1) "subscribe"
2) "weather:madrid"
3) (integer) 2
1) "message”
2) "weather:stockholm"
3) "7C, Cloudy"
1) "message"
2) "weather:madrid\"
3) "28C, Sunny"
```

```
127.0.0.1:6379> SUBSCRIBE weather:stockholm
Reading messages... (press Ctrl-C to quit)
1) "subscribe”
2) "weather:stockholm”
3) (integer) 1
1) "message”
2) "weather:stockholm”
3) "7C, Cloudy"
```

Secondary Index?

Full Text Search?

Machine Learning?

But Can Redis Do X?

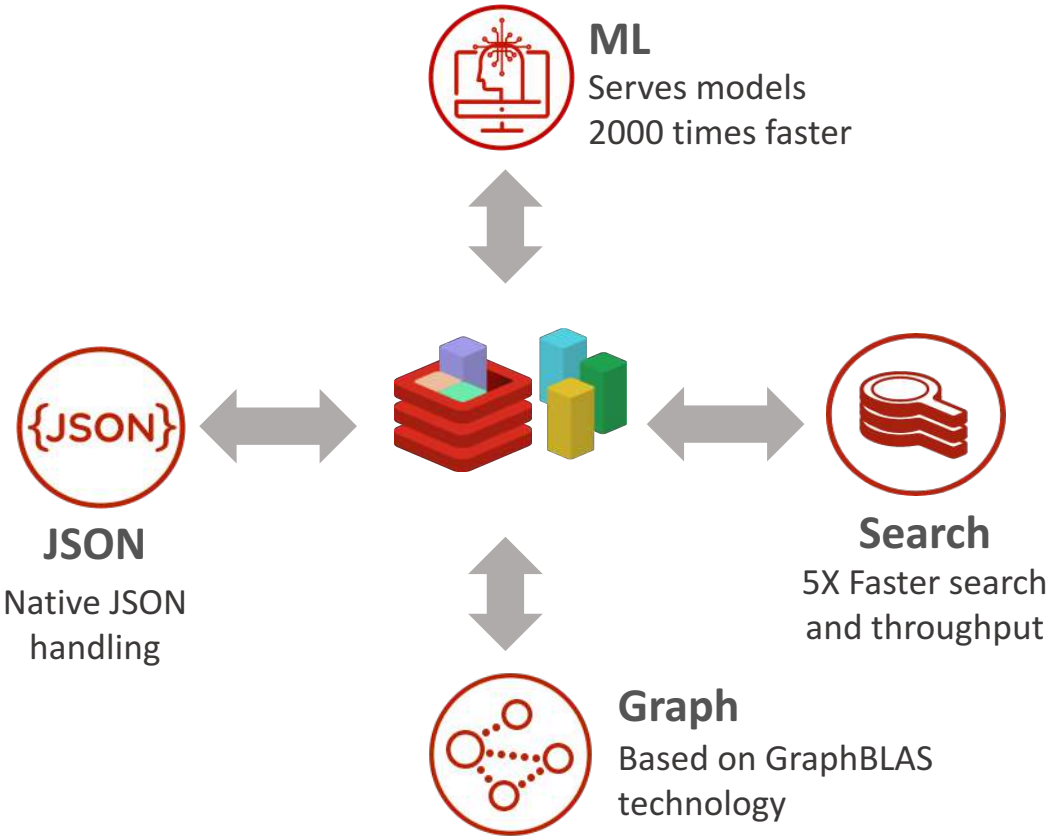
AutoComplete?

Graph?

Time Series?

Extensibility: True Multi-Model Functionality for All Use Cases

- Implemented by Redis Modules, independent of the Redis core
- Add-ons that use a Redis API to seamlessly support additional use cases and data structures
- Loosely coupled design, i.e. load only models needed for your use case
- Optimal data structure implementation for JSON, Graph, Search (and other) functionality, not just APIs
- Add new capabilities and data structures to Redis – in speeds similar to normal Redis commands
- Redis Enterprise Modules are tested and certified by Redis Labs



redislabs
HOME OF REDIS

Redis Modules

Redis Enterprise Modules

- Developed and supported by Redis Labs
- Deployed as part of Redis Enterprise
- Inherit all Redis Enterprise platform benefits
- Available with Redis Enterprise and listed in the Redis Modules Hub

Certified Modules

- Developed by a third party or by Redis Labs
- Code reviewed and tested by Redis Labs, certified for specific versions of Redis Enterprise and OSS Redis
- Support a single instance (or Master + Slave) configuration
- Installed by user
- Listed as certified in Redis Modules Hub

Custom Modules

- Developed per customer requirements
- Code reviewed and tested by Redis Labs, certified for specific versions of Redis Enterprise and OSS Redis
- Support a single instance or cluster configuration
- Installed by user
- Not listed in Redis Modules Hub

Other Modules

- Can be developed by anyone
- Listed at redis.io
- Listed in Redis Modules Hub Modules with proper documentation are shown in Redis Modules Hub (marked “Uncertified”)

Four Popular Redis Enterprise Modules

ReBloom

Probabilistic Data Structures:
Bloom filters, Cuckoo filters

ReJSON

JSON data type for Redis

RediSearch

Extremely fast text-based search,
used for secondary indexing

Redis-Graph

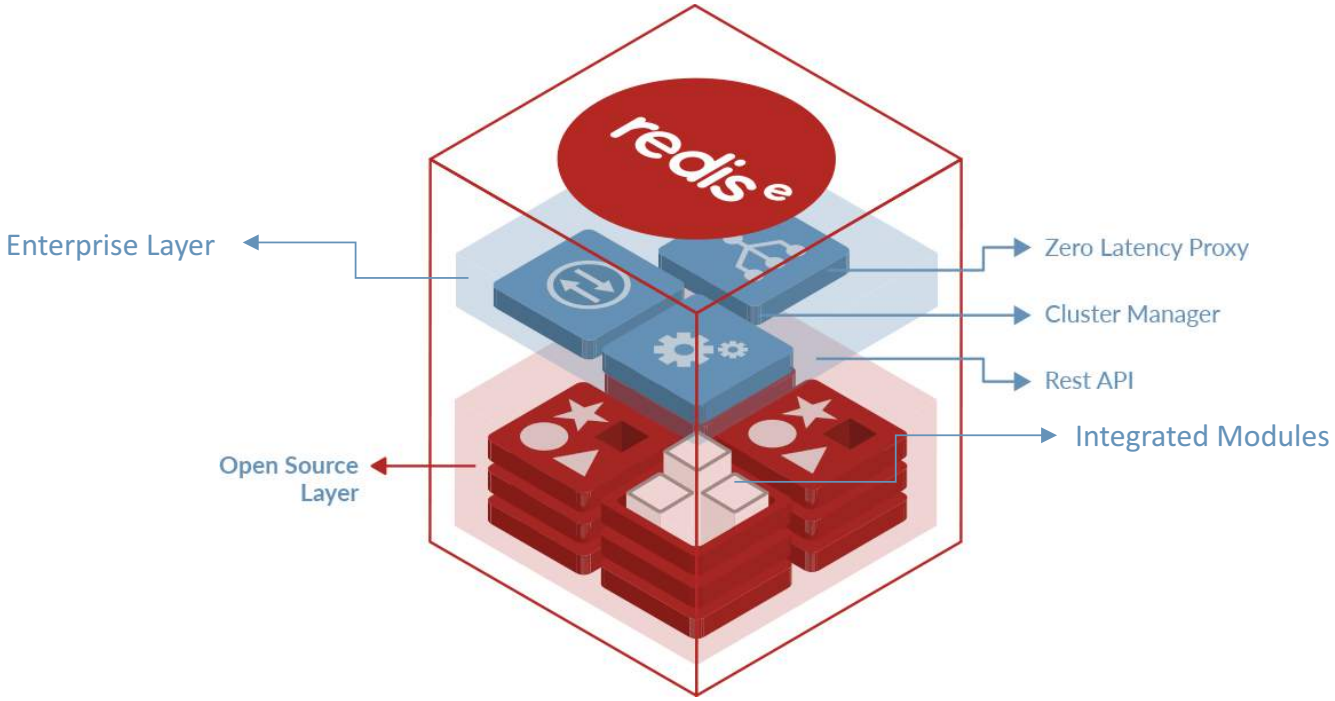
Graph query processing

redislabs
HOME OF REDIS

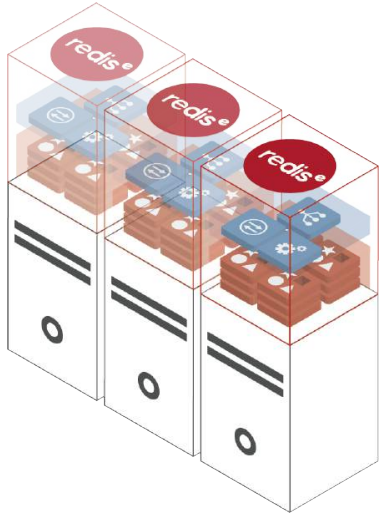
Redis Enterprise Architecture & Demo

Redis Enterprise: Open Source & Enterprise Technology

Redis Enterprise Node



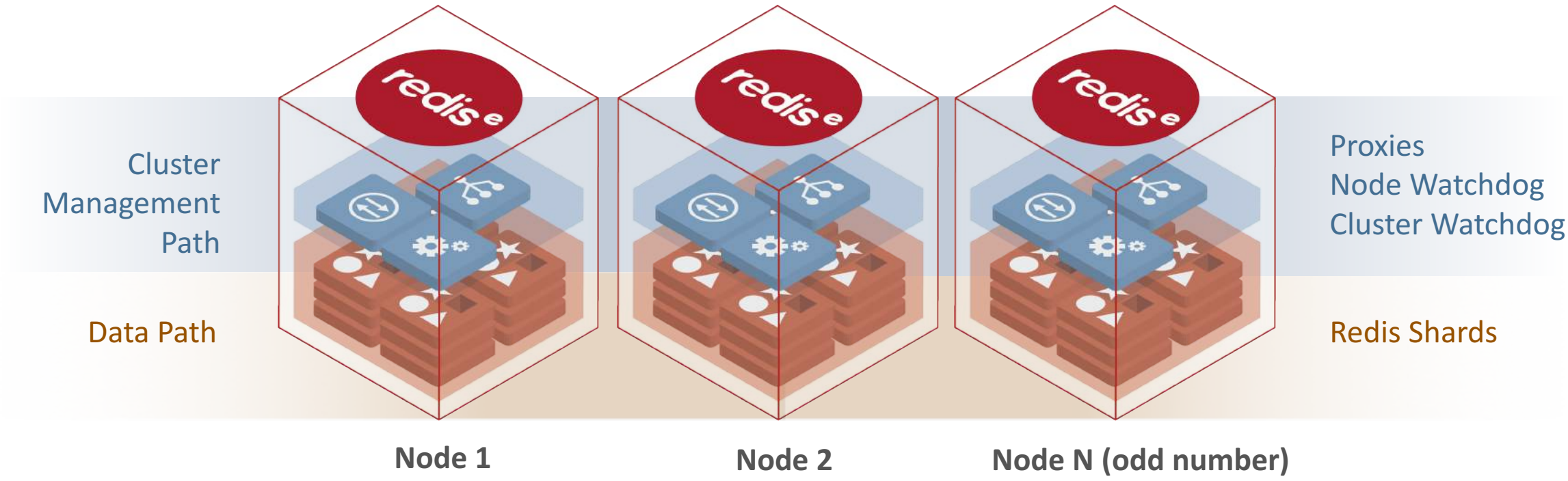
Redis Enterprise Cluster



- Shared nothing cluster architecture
- Fully compatible with open source commands & data structures

Redis Enterprise: Shared Nothing Symmetric Architecture

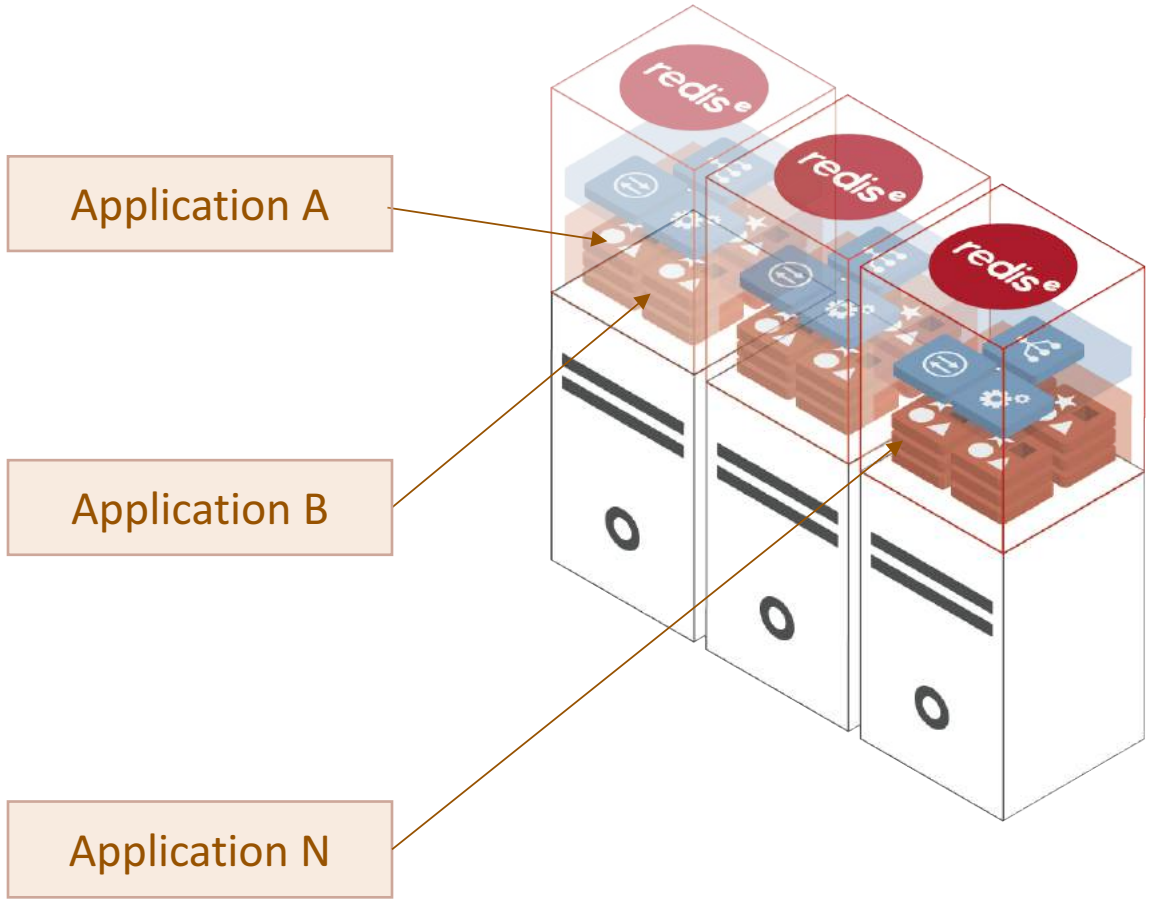
Distributed Proxies, Single or Multiple Endpoints



Unique multi-tenant container - like architecture enables running hundreds of databases over a single, average cloud instance without performance degradation and with maximum security provisions.

Redis Enterprise : Multi-Tenancy Maximizes Resource Utilization

200+ applications or shards on a single 4vcore cloud instance



- Shard isolation/protection
- Noisy-neighbor cancellation
- Minimizing CPU consumption of inactive applications

What we're covering

- Redis Enterprise UI
- Setting up a cluster
- Supplying load
- Failover

redislabs
HOME OF REDIS

Redis Transactions

Agenda

- Pipelines
- What are transactions
- Redis Execution Model
- Transaction Commands: MULTI, EXEC, DISCARD
- Optimistic Concurrent Control: WATCH, UNWATCH
- Durability

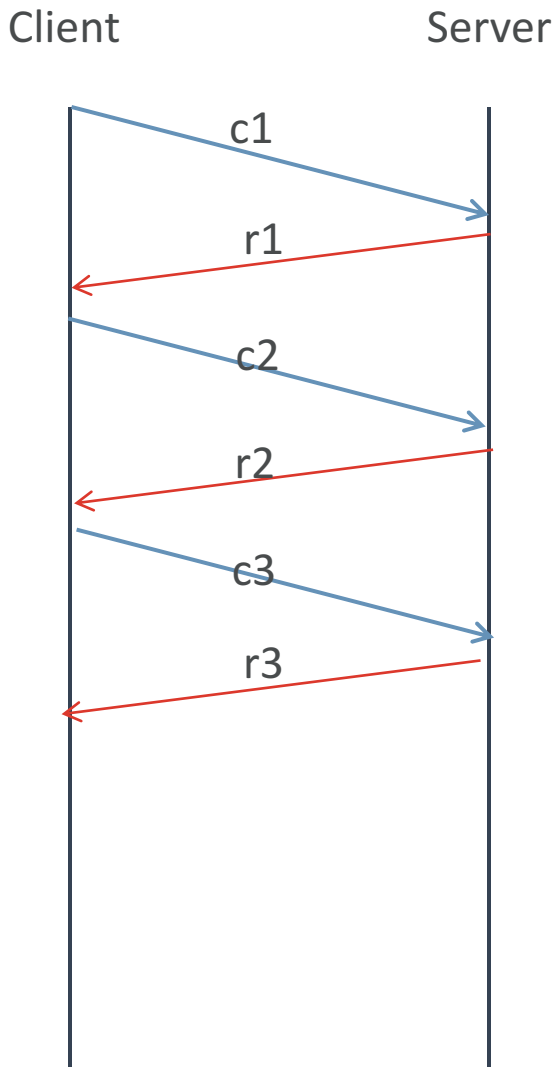
Redis Pipelines

Pipelining

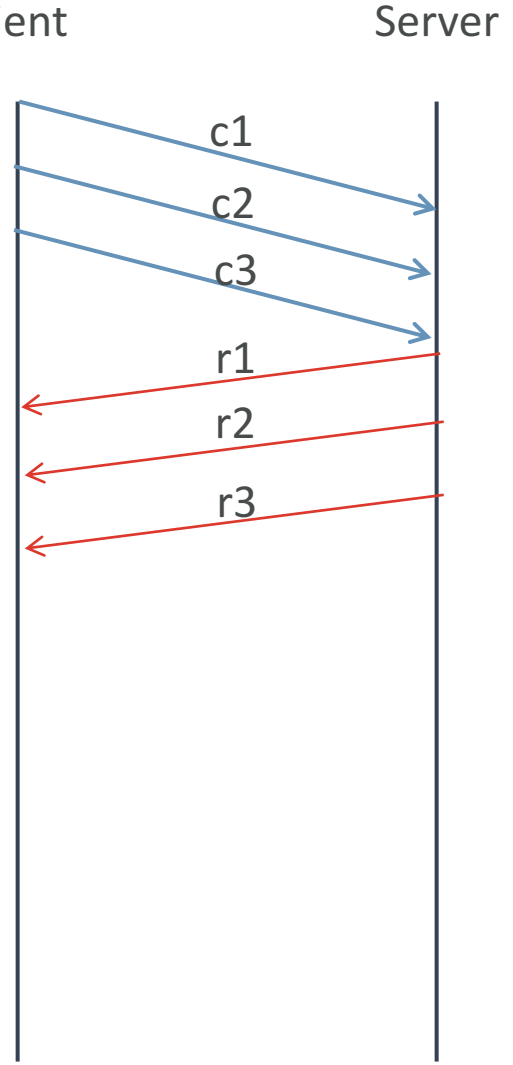
- Not a transaction!
- Avoids the RTT (Round Trip Time)
- Redis server can process new requests even if the client didn't read previous responses.
- Oftentimes capable of achieving a signifigant performance improvement
- Should not be used if dependent updates are required
- Some clients/languages do auto-pipelining

Pipelining

No Pipelining



Pipelining



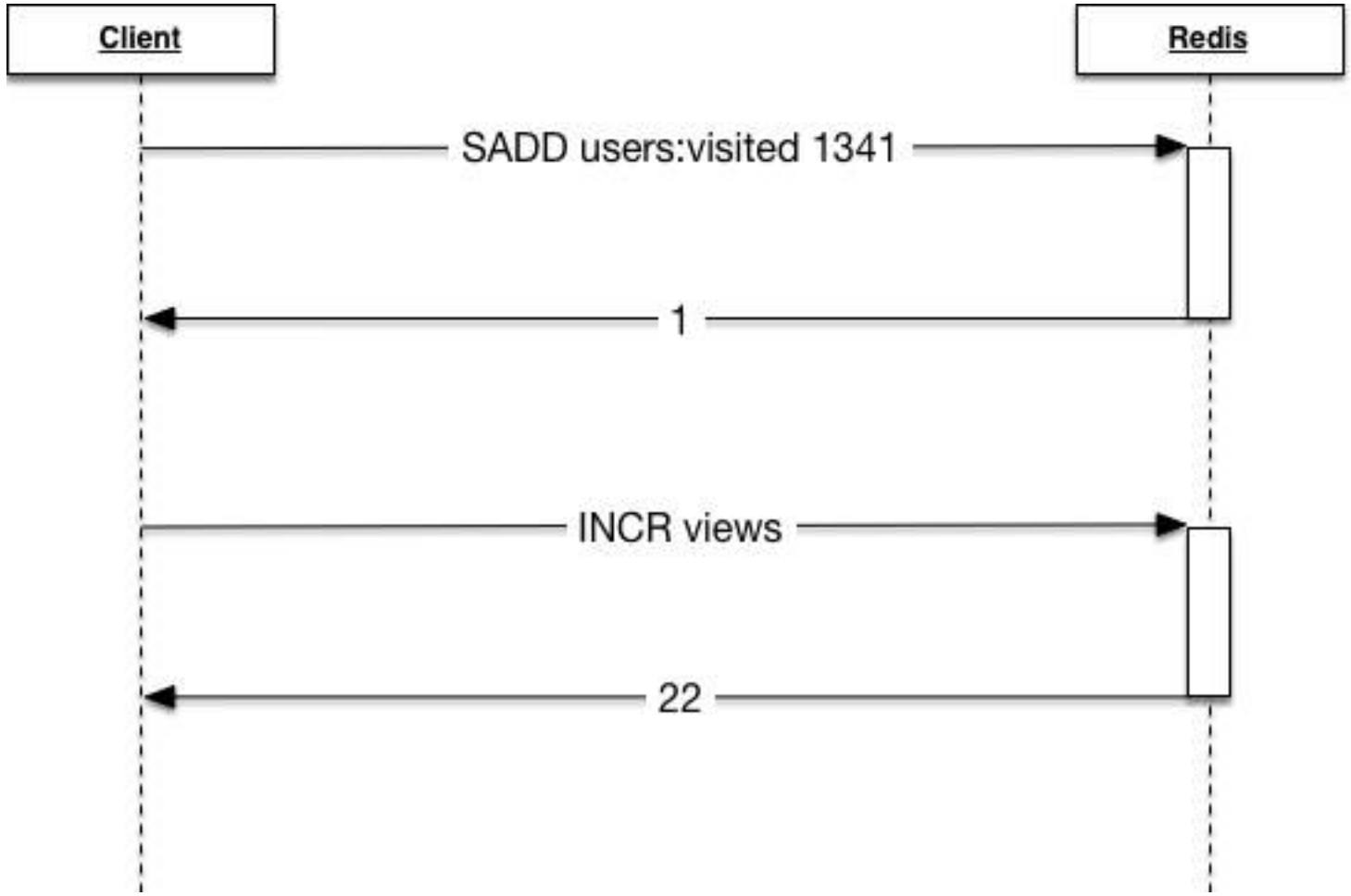
ACID Transactions

ACID Transactions

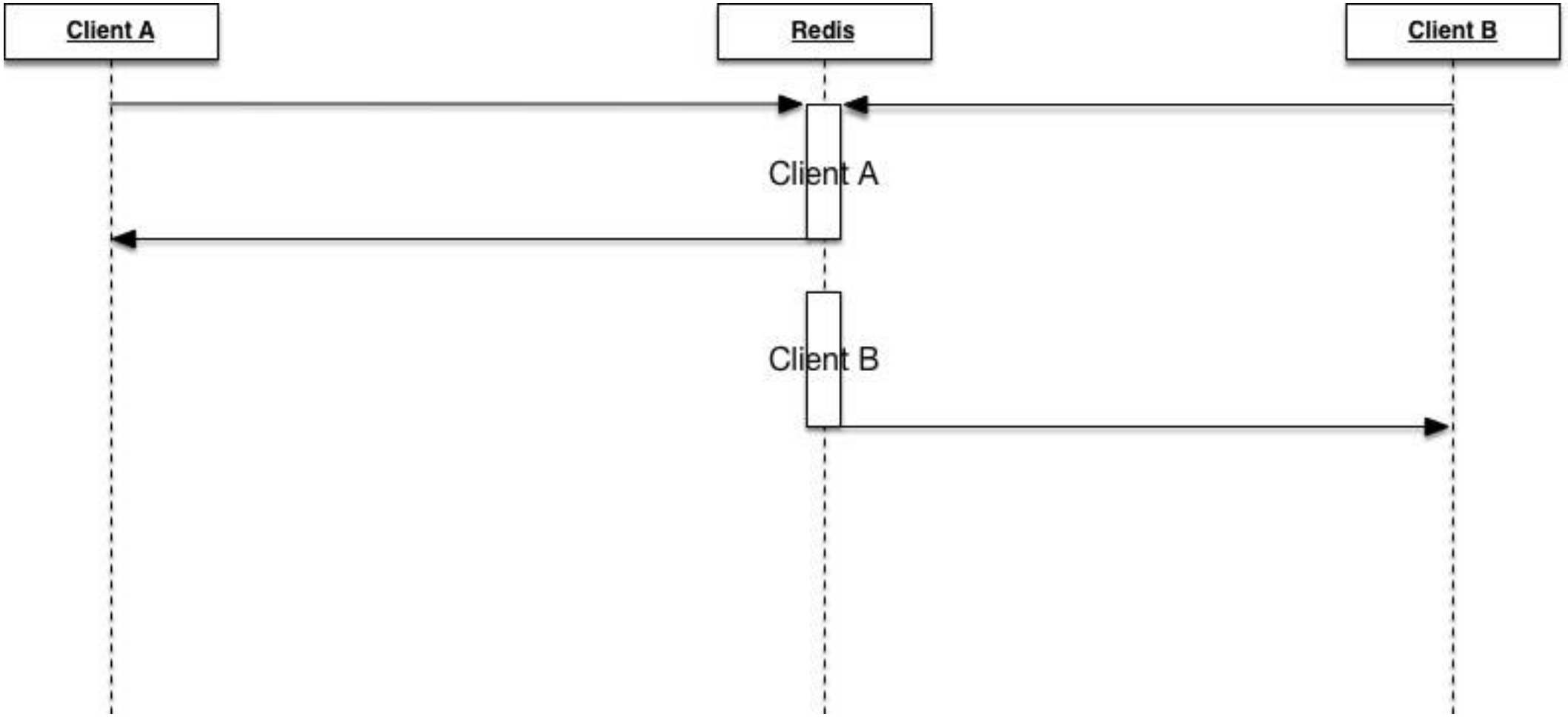
A - Atomicity
  -- Transaction executes as an indivisible unit
C - Consistency
  -- Transaction takes database from one valid state to another
I – Isolation
  -- Transactions result in a state as if they were executed sequentially
D – Durability
  -- Transaction changes are available event in the event of failure

Redis Execution Model

Single Client – Execution Flow



Two Client – Execution Flow



Blocking and Non-Blocking Commands

- Most Redis commands are synchronous

- Non-Blocking Commands
  - BGSAVE, BGREWRITEAOF
  - UNLINK (v4)
- Client Blocking Commands
  - SUBSCRIBE
  - BLPOP, BRPOP, BRPOPLPUSH
  - MONITOR
  - WAIT

Variadic (Dynamic Arity) Commands

- Variable number of arguments
- Examples (Non-exclusive)
  - MSET
  - MGET
  - HGET (v4)
  - HSET (v4)

- Executed as a single command

Multiple Command Transactions

Multiple Command Transactions

- MULTI to start transaction block
- EXEC to close transaction block
- DISCARD to abort transaction block

- Commands are queued until exec
- All commands or no commands are applied
- Transactions can have errors

MULTI Example

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> sadd site:visitors 124
QUEUED
127.0.0.1:6379> incr site:raw-count
QUEUED
127.0.0.1:6379> hset sessions:124 userid salvatore ip 127.0.0.1
QUEUED
127.0.0.1:6379> EXEC
1) (integer) 1
2) (integer) 1
3) (integer) 2
```

DISCARD Example

```
127.0.0.1:6379> sadd site:visitors 124
QUEUED
127.0.0.1:6379> incr site:raw-count
QUEUED
127.0.0.1:6379> DISCARD
OK
```

Transactions with Errors – Syntactic Error

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> set site:visitors 10
QUEUED
127.0.0.1:6379> ste site:raw-count 20
(error) ERR unknown command 'ste'
127.0.0.1:6379> EXEC
(error) EXECABORT Transaction discarded because of previous errors.
```

Transactions with Errors – Semantic Error

```
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> set messages:hello "Hello World!"
QUEUED
127.0.0.1:6379> incr messages:hello
QUEUED
127.0.0.1:6379> EXEC
1) OK
2) (error) ERR value is not an integer or out of range
```

Conditional Execution/Optimistic Concurrency Control

- WATCH to conditionally execute transaction if key unchanged
- UNWATCH clear c
- DISCARD to abort transaction block (CLI)

- All commands or no commands are applied
- Transactions **can** have errors

Dependent Modifications– Incorrect Version

```
def incorrectCheckBalanceAndTransferAmount(debit, credit, amount):
        amount = float(amount)
        debitkey = 'account:{}'.format(debit)
        creditkey = 'account:{}'.format(credit)
        fname = 'balance'
        balance = r.hget(debitkey, fname)

        # Potential race condition - start
        if balance >= amount:
                tx = r.pipeline()
                tx.hincrbyfloat(debitkey, fname, -amount)
                tx.hincrbyfloat(creditkey, fname, amount)
                return tx.execute()
        # Potential race condition - end
        else:
                raise Exception('insufficient funds')
```

Dependent Modifications – Correct Example

```
def checkBalanceAndTransferAmount(debit, credit, amount):
        amount = float(amount)
        debitkey = 'account:{}'.format(debit)
        creditkey = 'account:{}'.format(credit)
        fname = 'balance'
        while True:
             try:
                    tx = r.pipeline()
                    tx.watch(debitkey)
                    balance = float(tx.hget(debitkey, fname))
                    tx.multi()
                    if balance >= amount:
                       tx.hincrbyfloat(debitkey, fname, -amount)
                       tx.hincrbyfloat(creditkey, fname, amount)
                       return tx.execute()
                    else:
                       raise Exception('insufficent funds - time to get a job')
        except WatchError:
                # Reaching here means that the watched 'balance' value had changed,
                # so we can just retry or use any other backoff logic
                continue
```

Durability

Disk Based Persistence - Options

- Redis continues to serve commands from main memory
- Multiple Persistence modes
  - **Snapshot (RDB):** store a compact point-in-time copy every 30m, hourly, or daily – tunable (recommended with Active/active DBs)
  - **Append-only-file (AOF):** write to disk (fsync) every second or every write - tunable
- Provides durability of data across power loss
  - Look into replication to prevent data loss in case of node loss

RDB Persistence

- Persistence
  - Fork Redis process
  - Child process writes new RDB file
  - Atomic replace old RDB file with new

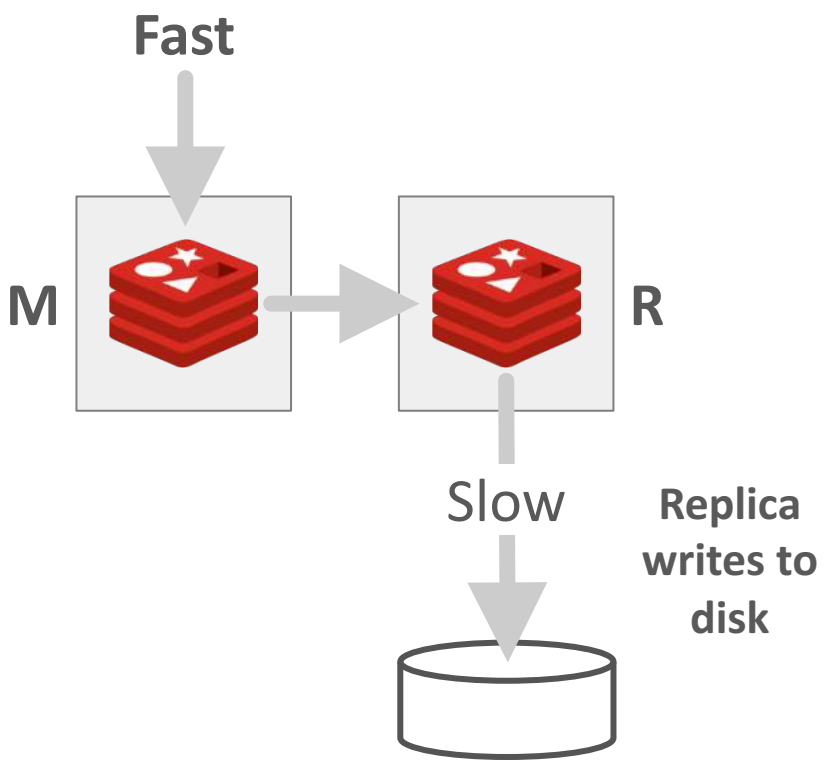
- Configuration
  - SAVE directive (Redis.conf): SAVE <seconds> <min-changes>
  - Runtime : CONFIG SET SAVE "60 1000 120 100 180 1"

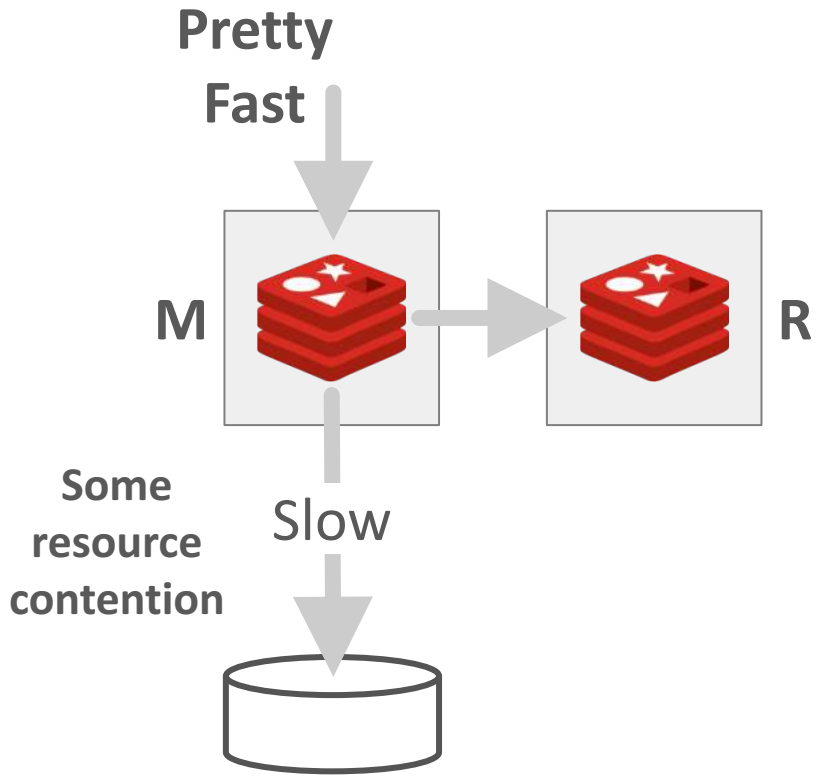
- Trigger manually
  - SAVE command (synch)
  - BGSAVE (background)

AOF Persistence

- Configuration
  - APPENDONLY directive (Redis.conf): APPENDONLY YES
  - Runtime : CONFIG SET APPENDONLY YES
- AOF File fsync options
  - Trade off speed for data security
  - Options: None, everysecond, always
- BGREWRITEAOF
  - aof file grows indefinitely
  - BGREWRITEAOF – trigger compaction of AOF file

Redis Enterprise Durability - Persistence Topologies





Transactions Review

Transactions Summary

- Pipelines aren't transactions, but reduce roundtrips
- *Mostly* ACID Transactions
  - Atomic – through MULTI/EXEC
  - Isolation, Consistency – single threaded nature
  - Durability – persistence modes: snapshots and append-only-file
- No Rollback – transaction commands are queued then sent to server
- Single threaded event-loop for serving commands
- WATCH for optimistic concurrency control

redislabs
HOME OF REDIS

Hands-on Development

Reminder:

Grab the code

bit.ly/red-workshop-oct-18

Exercise I: Hello, Redis.

- `hello_redis` directory
- Run the python code as-is - see what happens
  - Insert your connection information in the code
  - Follow the comment-based instructions
- Don’t cheat, but there is a solution in the directory

Exercise II: Pub/Sub

- `pubsub` directory
- Two Parts:
  - Run the python code for `publish.py` as-is - see what happens
    - Insert your connection information in the code
    - Follow the comment-based instructions
  - Run the python code for `subscribe.py` as-is - see what happens
    - Insert your connection information in the code
    - Follow the comment-based instructions
- Don’t cheat, but there is a solution in the directory

Bonus Exercise: JSON

- `json` directory
- `raw_rejson.py` contains some working sample code.
  - Supply your own credentials
- Write your own Python application that:
  - Writes a new JSON object to an empty key
  - Reads a specific value from the JSON object
  - Does an in place update of the JSON
  - Manipulates an array inside the JSON

redislabs
HOME OF REDIS

Redis Enterprise Replication, High Availability & Active-Active

3 Replication Techniques with Redis Enterprise

1. Active – Passive

   Passive server is a cold standby

   Uses: High Availability, Disaster Recovery, Data Durability

2. Active – Read-replica

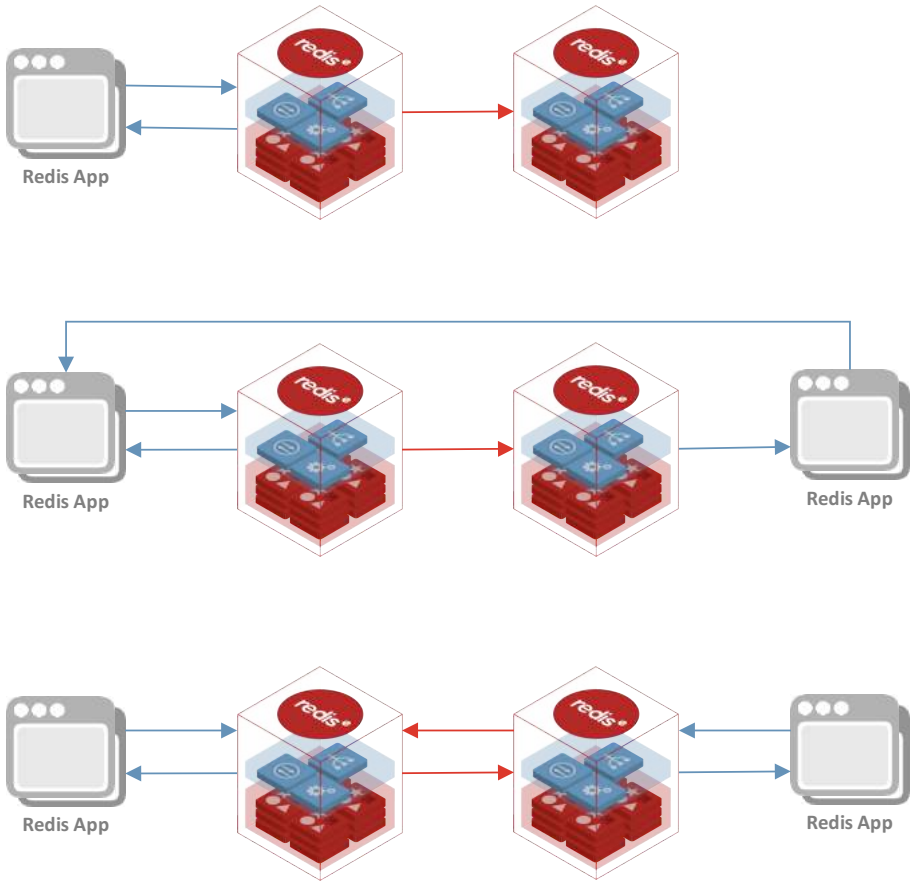
   Read-replica is available in the read-only mode

   Uses: Distributed caching

3. Active – Active

   All database instances are available for read and write operations

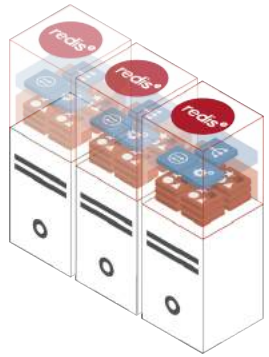
   Uses: Local latencies for geo-distributed apps, load distribution, data consolidation



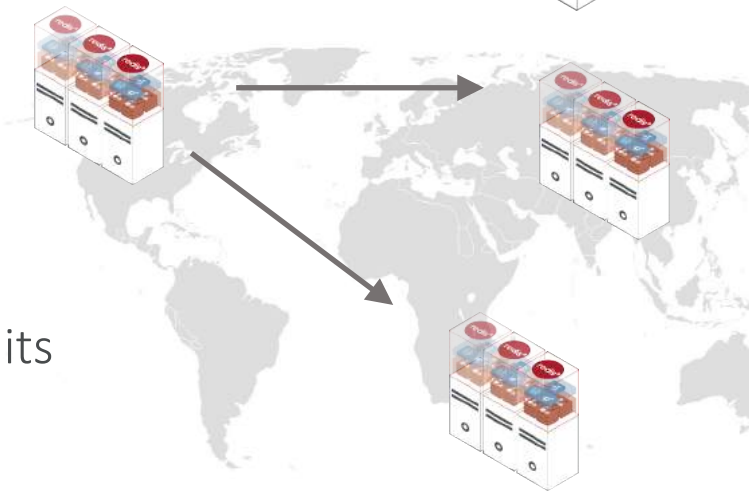
Active-Read Replica

Low Replication Lag & High Replication Throughput

- Local Replication: Built for LAN
  - Higher bandwidth
  - Lower latency
  - High quality links susceptible to fewer failures and retransmits

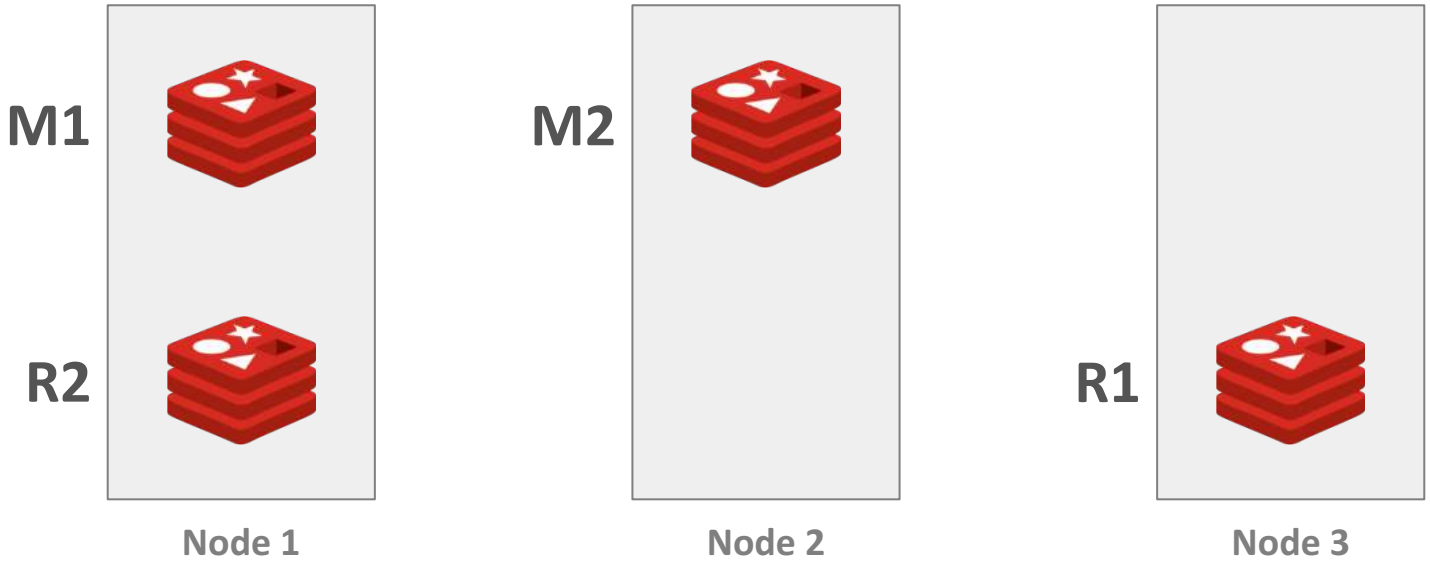


- Cross-Geo Replication: Built for WAN
  - Lower bandwidth
  - Higher latency
  - “Noisier” network quality susceptible to more failures and retransmits



High Availability

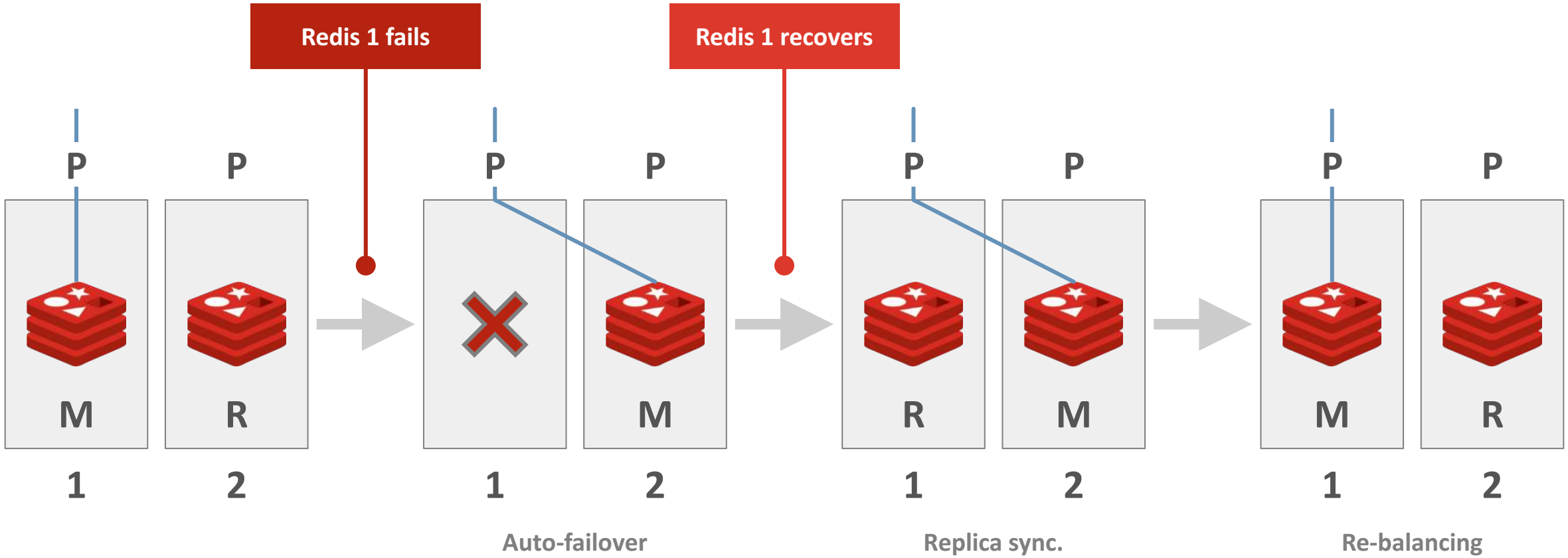
High Availability: Shard Replication



M Master | **R** Replica

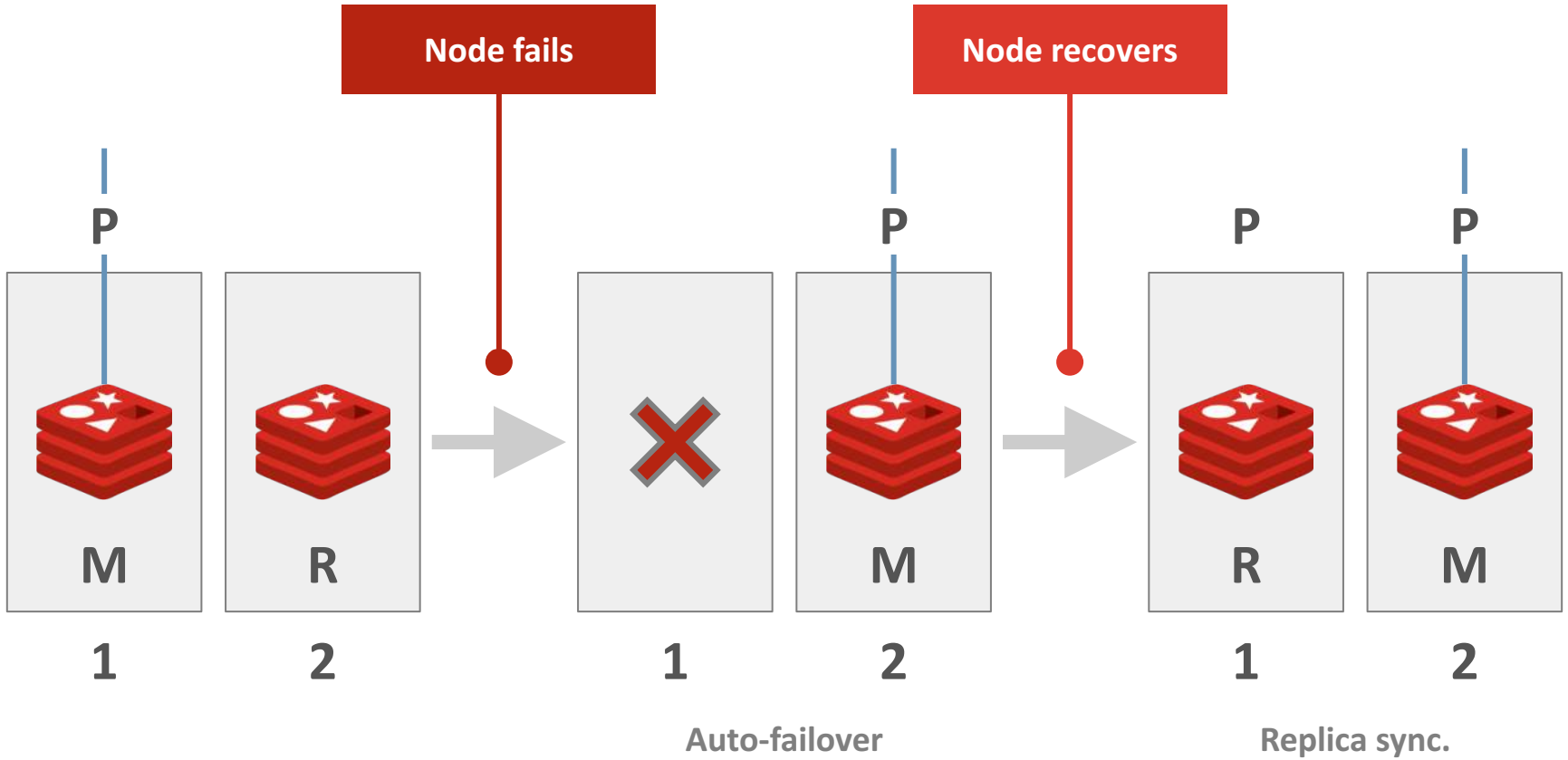
Replica shards can be in the same / different rack / datacenter.

High Availability: Shard Failure



redislabs
HOME OF REDIS

High Availability: Node Failure



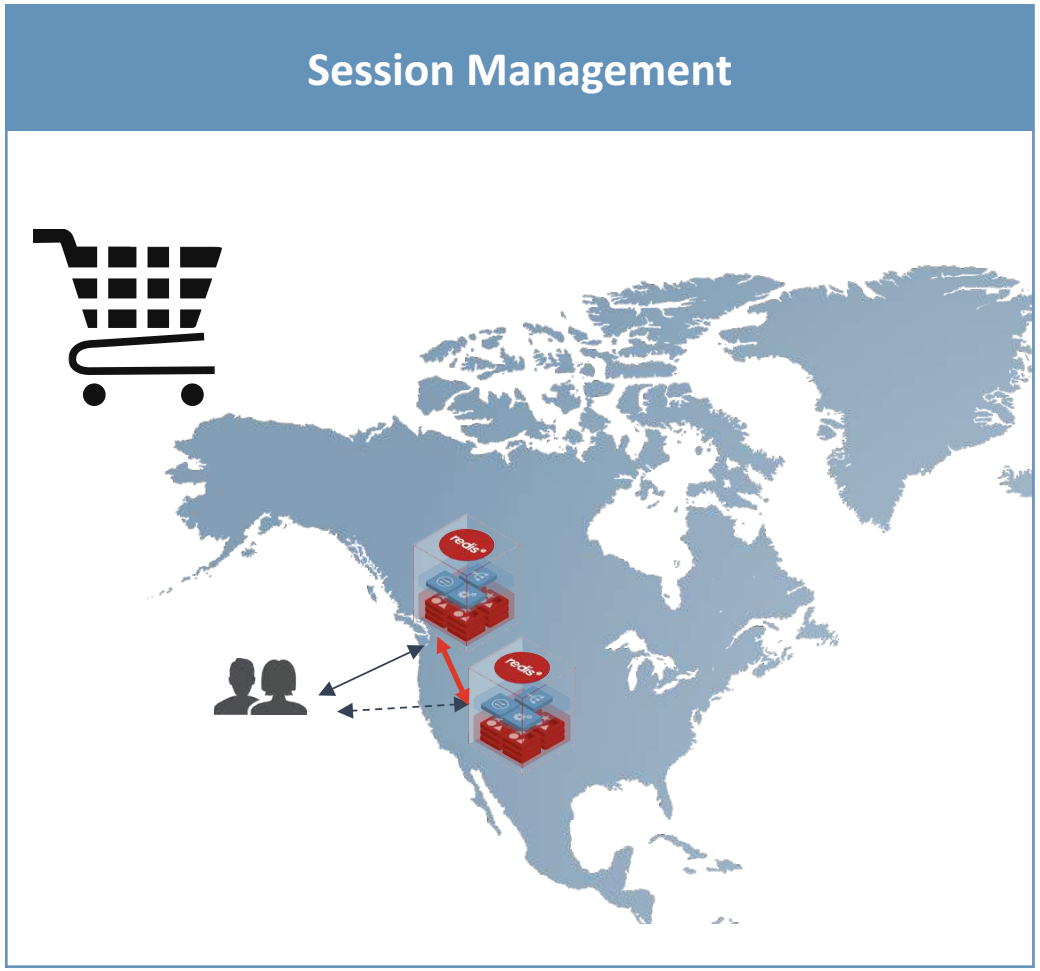
Active-Active & CRDTs

CRDT (Conflict-Free Replicated Data Type)

- Years of academic research
- Based on consensus free protocol
- Strong eventual consistency
- Built to resolve conflicts with complex data types

Why do you need Active-Active in Redis Enterprise?

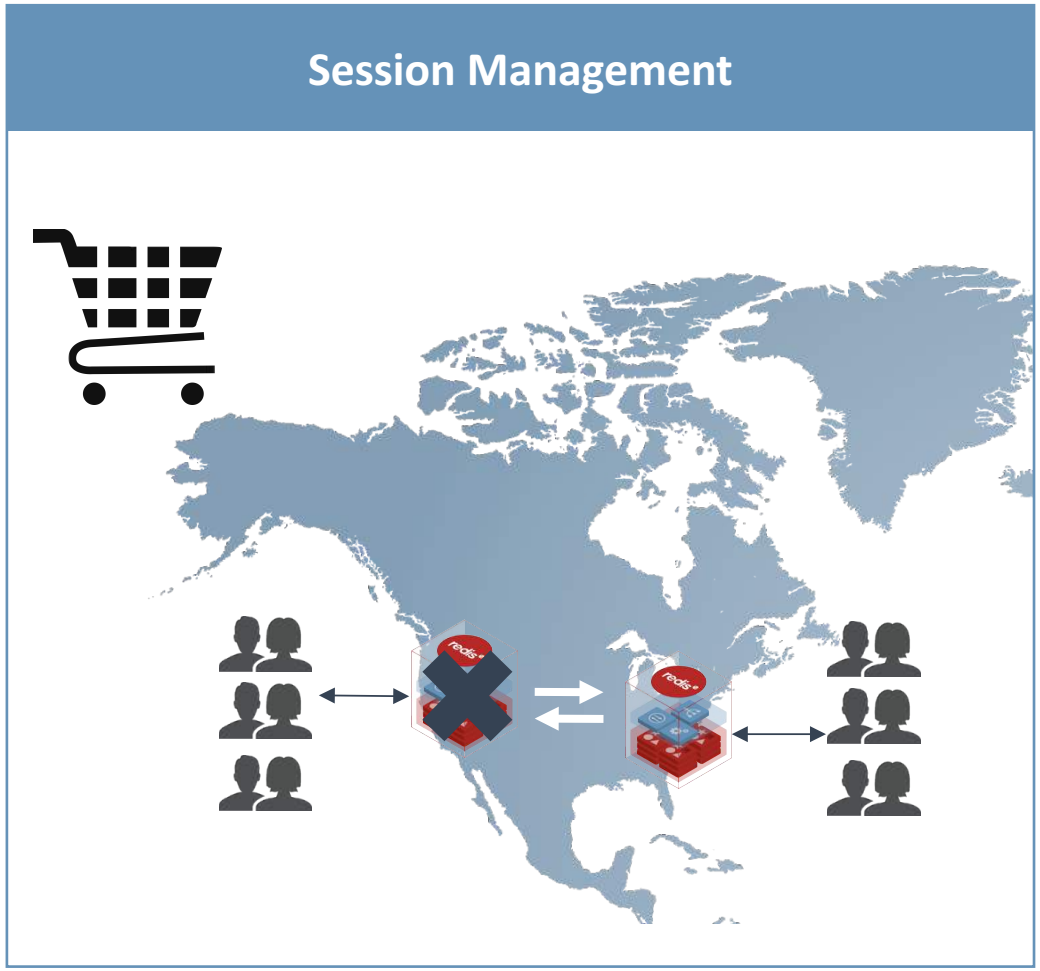
1. Migrating user sessions across data centers



1. Customers get routed from one datacenter to another on the fly or while moving from one location to another
2. All the session states (example: items in shopping carts) need to be exactly the same
3. If routed back, any changes need to be sync-ed between datacenters

Why do you need Active-Active in Redis Enterprise?

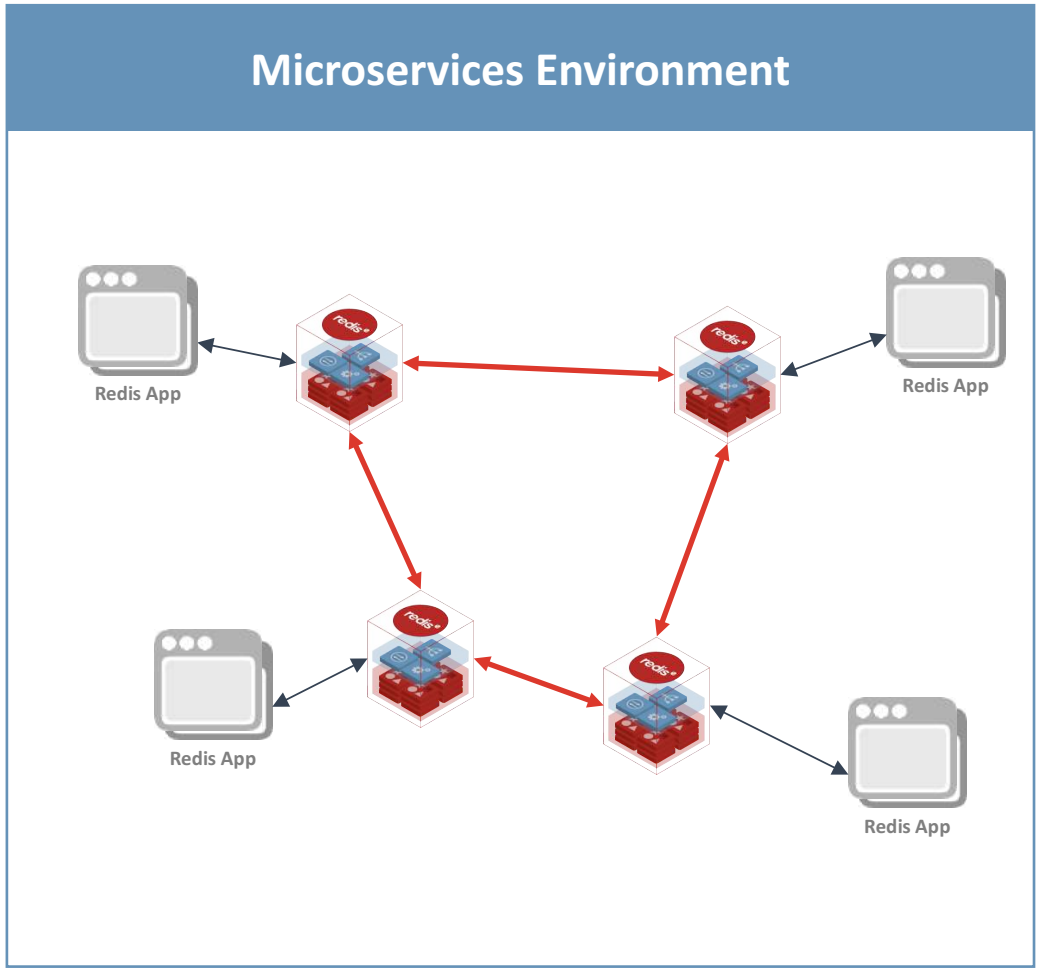
2. Handling node failures



1. Failure in one datacenter, needs sessions to move over to the other data center
2. All the session states (example: items in shopping carts) need to be exactly the same
3. Once restored, any changes need to show up in first datacenter

Why do you need Active-Active in Redis Enterprise?

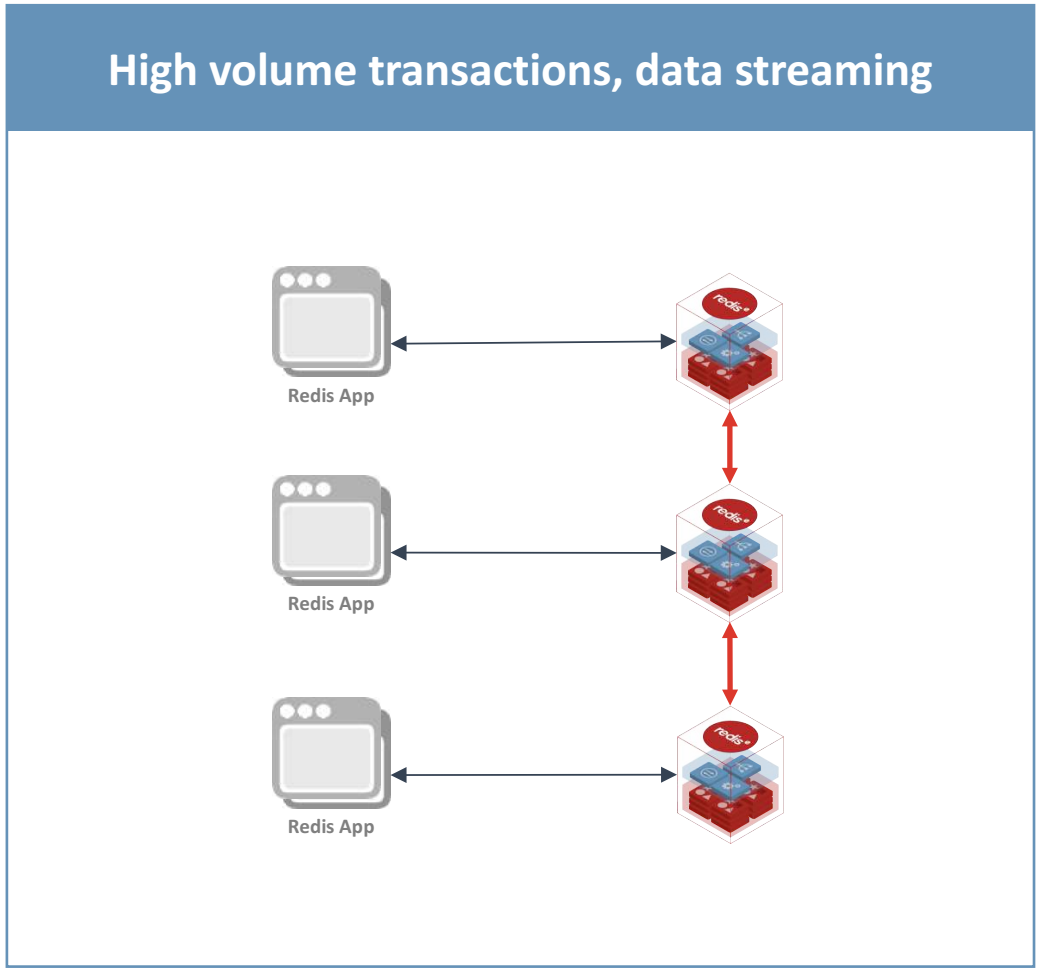
3. Data consolidation



1. In a microservices environment, apps connect to their own databases
2. However, apps share data structures/tables and require data to be consistent

Why do you need Active-Active in Redis Enterprise?

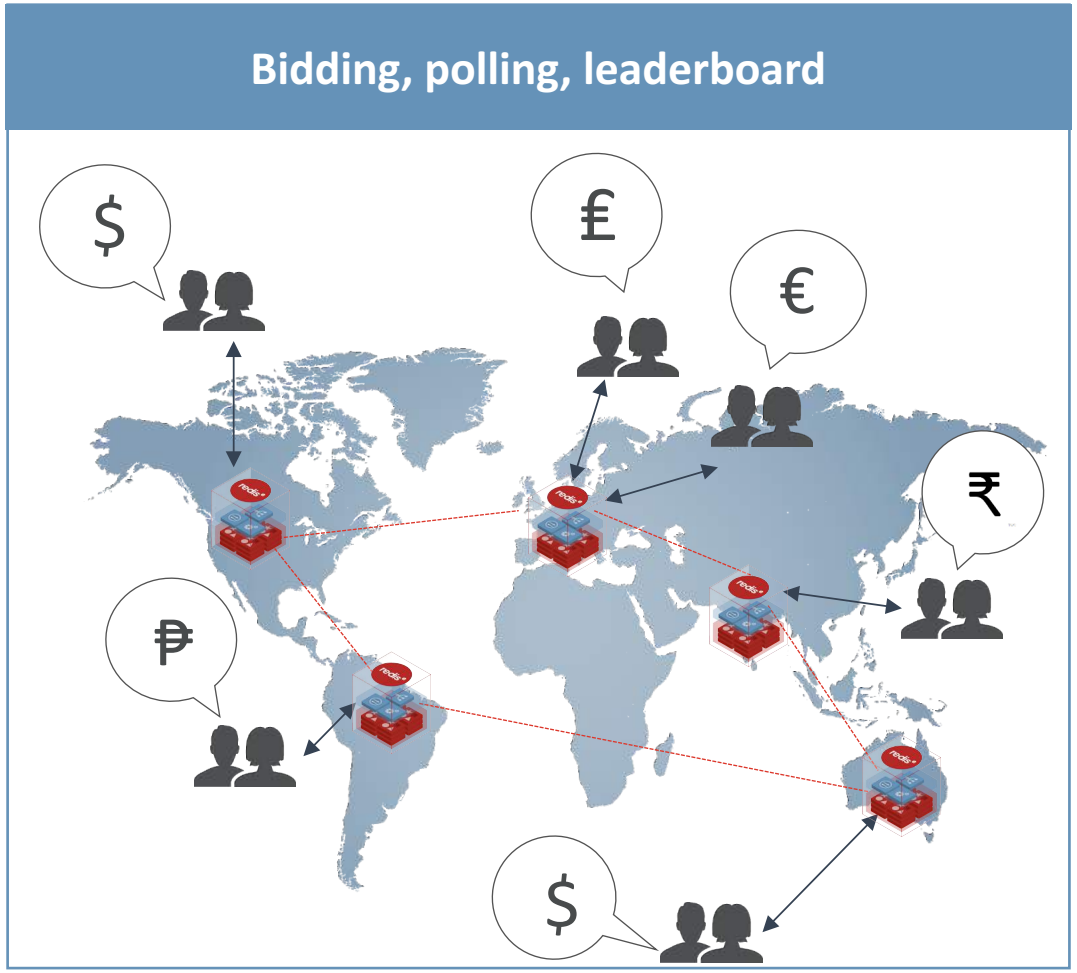
4. Load distribution



1. Need to handle high volume of incoming traffic
2. Distribute the load across multiple servers

Why do you need Active-Active in Redis Enterprise?

5. Delivering local latencies for geographically distributed apps



Geo distributed datacenters

1. High frequency writes/reads in many regions
2. Complex data types, not just key-value
3. Database needs to reflect current position
4. Customer needs simple ways to resolve simultaneous updates – delegate to databases

Redis Enterprise Delivers Strong Eventual Consistency And Causal Consistency

High Performance

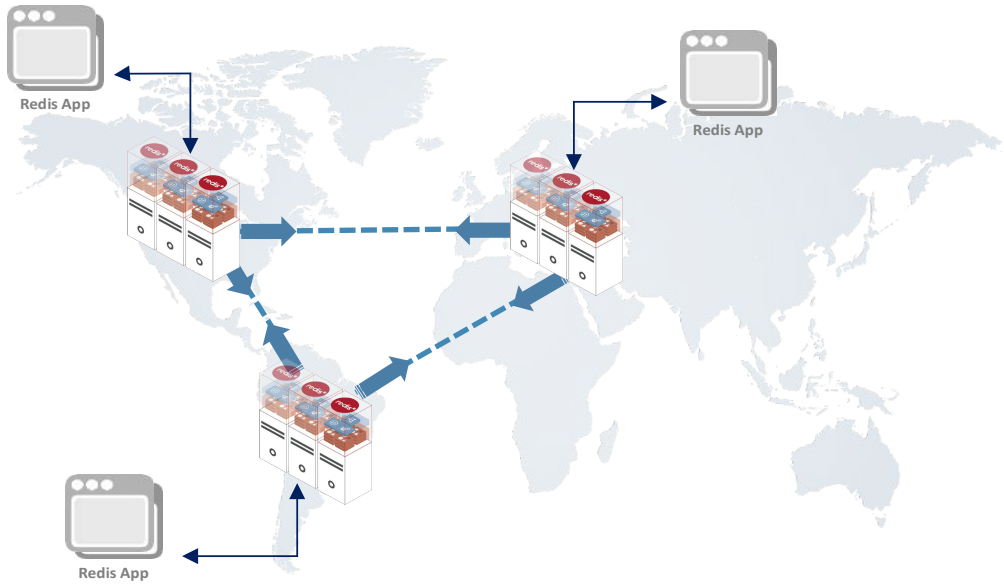
Read and write with low local sub-millisecond latency

Guaranteed data consistency

CRDT based: The datatypes are conflict-free by design.

Simplifies the app design

Develop as if it's a single app in a single geo, we take care of all the rest



redislabs
HOME OF REDIS

Why Does This Matter?

- Fast Time to Market
  - Simpler to develop geo distributed apps
  - Results in high performance, consistent and highly responsive apps
- Easier to Deploy and Maintain
  - Database does all the heavy lifting
  - Simpler code means easier to change
  - Simpler architectures are easier to implement and manage
- Future Proof
  - Based on the latest thinking in computer science (CRDT)
  - Complex datatypes included in conflict resolution

Solutions that benefit from Redis Enterprise with Active-Active support

Fraud Mitigation

- Geo Distributed Event Tracking: *Sets Gathering Geo Distributed Events*

Social Engagement Apps

- Encoding Social Engagement: *Distributed Counters for "Likes", "Shares", "Retweets"*

Collaboration Apps

- Constructing Smart Timelines: *Merged Lists Ordering Posts*
- Instant Messaging & Conversation Tracking: *Merged Lists Ordering Conversations*

Geo Distributed Trading/Bidding

- Auctions, Bids/Asks: *Lists/Sorted Sets tracking Bids and Asks*

Dashboards & Scoreboards

- Tracking Geo Distributed Scoreboards: *Sorted Sets tracking ordered scores*

Real-time Metering Apps

- Tracking Usage/Consumption: *Sets/Lists Tracking Consumption Events*

And more…..

Questions?

redislabs
HOME OF REDIS

redislabs
HOME OF REDIS

Lua in Redis

Lua overview

- Redis has an embedded sandboxed Lua v5.1 engine.
- User scripts can be sent for execution by the server.
- When a script runs, it blocks the server and is atomic.
- Scripts have full access to the data.

What is Lua

https://www.lua.org/about.html

“Lua is a powerful, efficient, lightweight, embeddable scripting language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description.

Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed ...

TL;DR it is a scripting language

Lua's Hello World

```
-- src: http://rosettacode.org/
print "Hello, World!"
```

- Single line comments begin with a double dash
- Single argument function calls need no parenthesis
- "This is" a string, and also 'this one'
- Whitespace is the statement separator/terminator; semicolon optional

Running one-off scripts

```
redis> EVAL
       "return('I <' .. tostring(3) .. ' Lua')" 0
"I <3 Lua"
```

- EVAL expects the raw script as input - it is dev command
- The functions `tostring` and `tonumber` are usually implicitly called
- Remember the exotic string concatenation operator `..`
- The script can return a value with the `return` statement

Cached scripts

Redis caches the bytecode of **every script** it executes in order to avoid re-compiling it in subsequent calls.

Cached scripts offer two major performance gains:

- Sent to the server only* once
- Parsed to bytecode only* once

* More accurately is "on every SCRIPT LOAD"

Important: keep an eye on the script cache's size, it may explode.

Loading and running cached scripts

```
redis> SCRIPT LOAD "return 21 * 2.01"
       "935b7b8d888c7affc0bac8a49e63933c915b883f"
redis> EVALSHA
       935b7b8d888c7affc0bac8a49e63933c915b883f 0
(integer) 42
redis> EVALSHA nosuchscriptsha1 0
(error) NOSCRIPT No matching script. Please use
EVAL.
```

- SCRIPT LOAD returns the sha1sum of the script
- EVALSHA expects the sha1sum of the loaded script

```
# KEYS and ARGV are prepopulated indexed tables
$ redis-cli EVAL
  "return { ARGV[1], ARGV[2], ARGV[3] }"
  0 foo bar baz
1) "foo"
2) "bar"
3) "baz"
$ redis-cli EVAL
  "return { KEYS[1], { ARGV[1], ARGV[2] } }"
  1 foo bar baz
1) "foo"
2) 1) "bar"
   2) "baz"
```

Use Case: pure functions, better transactions and composable commands

Lua scripts are intended to be pure functions.

If you must, store a state in a key in the database, but always explicitly pass its name.

Lua scripts, being atomic and quite powerful, are often a nicer alternative to using WATCH/MULTI/EXEC/DISCARD and retry. And most times also run faster.

Put differently: Lua lets **compose** commands using the existing API.

```
redis> HSET person fname salvatore lname sanfilippo
(integer) 1

redis> SCRIPT LOAD  "return ARGV[1] .. ', ' ..
redis.call('HGET',KEYS[1],'fname') .. ' ' ..
redis.call('HGET',KEYS[1],'lname')"
"e6b12e9a33824c5a614fa7fc0e516b5465e1bfc0”

redis> EVALSHA e6b12e9a33824c5a614fa7fc0e516b5465e1bfc0
1 person howdy
"howdy, salvatore sanfilippo"
```

redislabs
HOME OF REDIS

Redis Modules

Sample a few modules in a few minutes

- ReJSON
- RediSearch
- Redis Graph

ReJSON

What is JSON

- JavaScript Object Notion
  - Data interchange format
  - Lightweight
  - Human Readable
  - Simple to Parse
  - Simple to Generate
- ECMA Standard
- Common web development format

```
{
    “userId”: 0,
    “firstName”: “Salvatore”,
    “lastName”: “Sanfilippo”,
    “userName”: ”antirez”
}
```

Storing JSON in Redis

Serialized as Redis String

- GET/SET operations
- O(N) access
- Client deserialization
- No in-place updates

Deserialized into a Hash

- Decode: HMSET
- Encode: HMGET
- O(1) Access
- Client deserialization
- No in-place updates

ReJSON Module

- JSON as a native Redis Data Type
- Keys map to JSON values
  - Scalars
  - Objects
  - Arrays
  - Nested or Not
- Stored as a document tree structure
- Path access to JSON elements
- Atomic, in-place updates

```
redis> JSON.SET json:scalar . '"Hello JSON!"'
OK
redis> JSON.SET json:object . '{"userId": 3001, "firstName": "Salvatore"}'
OK

redis> JSON.GET json:scalar
"Hello JSON!"
redis> JSON.GET json:object
{"userId":3001,"firstName":"Salvatore"}
redis> JSON.GET json:object .userId
3001

redis> JSON.SET json:object .userId 2001
OK
Redis> JSON.GET json:object
{"userId":2001,"firstName":"Salvatore"}
```

redislabs
HOME OF REDIS

ReJSON Commands

General	`JSON.DEL, JSON.GET, JSON.MGET,JSON.SET,JSON.TYPE`
Numbers	`JSON.NUMINCRBY, JSON.NUMMULTBY`
Strings	`JSON.STRAPPEND, JSON.STRLEN`
Objects	`JSON.OBJKEYS, JSON.OBJLEN`
Arrays	`JSON.ARRAPPEND, JSON.ARRINDEX, JSON.ARRINSERT, JSON.ARRLEN, JSON.ARRPOP, JSON.ARRTRIM, JSON.RESP`

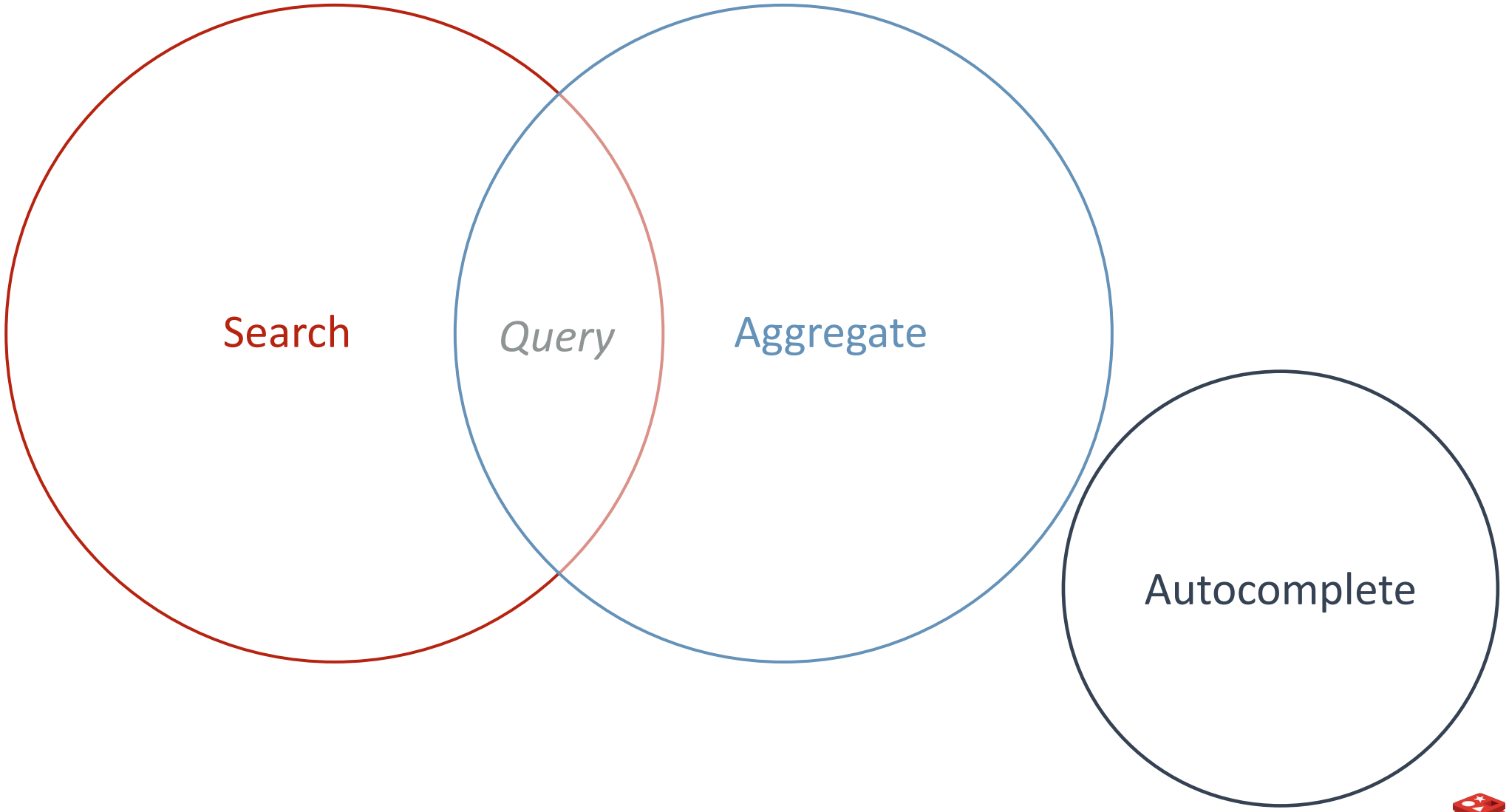
RediSearch

RediSearch can be used for two [primary] things:

Full Text Search

Secondary Index

RediSearch can do three things.



Data Lifecycle in RediSearch / Search and Aggregate

- Create a schema using four types
  - Text
  - Numeric
  - Tag
  - Geospatial
- Add Documents in Real Time
  - Directly
  - From Hash
  - Index only
- Search & Aggregate
- Delete documents as needed
- Drop the whole index

Search & Query

Query Language

- Goals
  - Intentionally not SQL
  - But familiar
  - Exposable to end-users
- Simple
  - No knowledge of data/structure needed
- Powerful
  - With knowledge, zero in on data

Query Syntax

ford truck

Query Syntax – more advanced

AND / OR / NOT / Exact Phrase / Geospatial / Tags / Prefix / Number Ranges / Optional Terms

& more

And combine them all into one query:

```
(chev*|ford) -explorer ~truck @year:[2001 2011] @location:[74 40 100 km] @condition:{ good | verygood }
```

Full-text Search

- Stop words:
  - ”a fox in the woods” -> “fox woods”
- Stemming:
  - Query “going” -> find “going” ”go” “gone”
- Slop:
  - Query: “glass pitcher”, slop 2 -> “glass gallon beer pitcher”
- With or without content:
  - Query: “To be or not to be” -> Hamlet (without the whole play)

  Matched text highlight/summary:
  - Query – “To be or not to be” -> Hamlet. To be, or not to be- that is the question

Full-text Search

- **Synonyms**
  - Query “Bob” -> Find documents with “Robert”
- **Query Spell Check**
  - ”a fxo in the woods” -> Did you mean “a **fox** in the woods”
- **Phonetic Search**
  - “John Smith” -> “Jon Smyth”

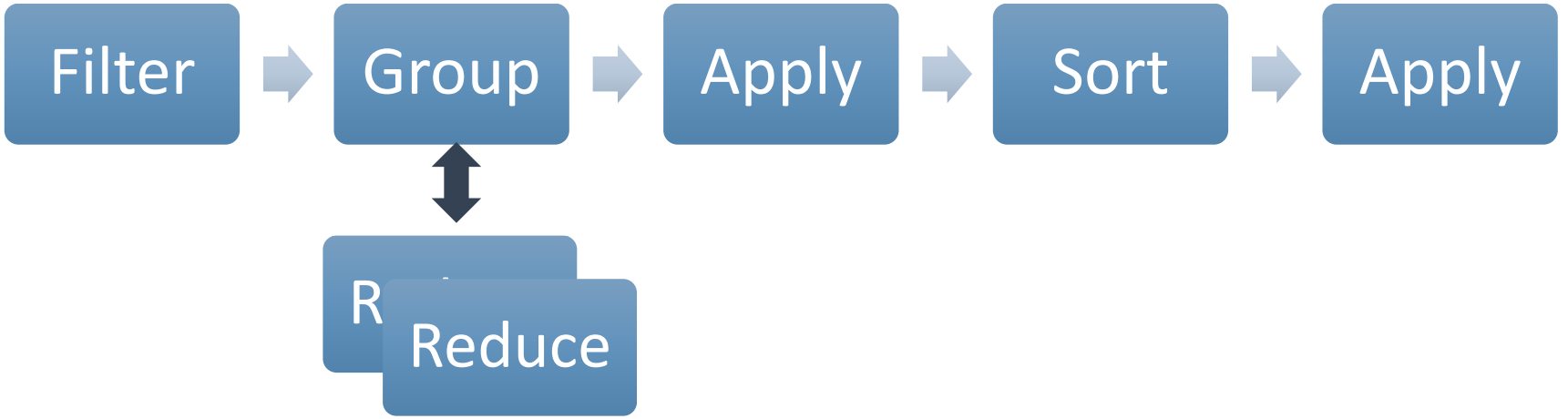
Scoring, Weights, and Sorting

- Each field can have a weight which influences the rank in the returned result
- Each document can have a score to influence rank
- Built-in Scoring Functions
  - Default: TF-IDF / term frequency–inverse document frequency
    - Variant: DOCNORM
    - Variant: BM25
  - DISMAX (Solr's default)
  - DOCSCORE
  - HAMMING for binary payloads
- Fields can be independently sortable, which trumps any in-built scorcer

Aggregations

Aggregations

- Processes and transforms
- Same query language as search
- Can group, sort and apply transformations
- Follows pipeline of composable actions:



Grouping & Applications

- Reducers:
  - COUNT
  - COUNT_DISTINCT
  - COUNT_DISTINCTISH
  - SUM
  - MIN
  - MAX
  - AVG
  - STDDEV
  - QUANTILE
  - TOLIST
  - FIRST_VALUE
  - RANDOM_SAMPLE

- Manipulate
  - Strings
    - `substr(upper('hello world'),0,3)` -> "HEL"
  - Numbers w/ Arithmetic
    - `sqrt(log(foo) * floor(@bar/baz)) + (3^@qaz % 6)`
  - Timestamp to Calendar
    - `timefmt(@mytimestamp, "%b %d %Y %H:%M:%S")` -> Feb 24 2018 00:05:48

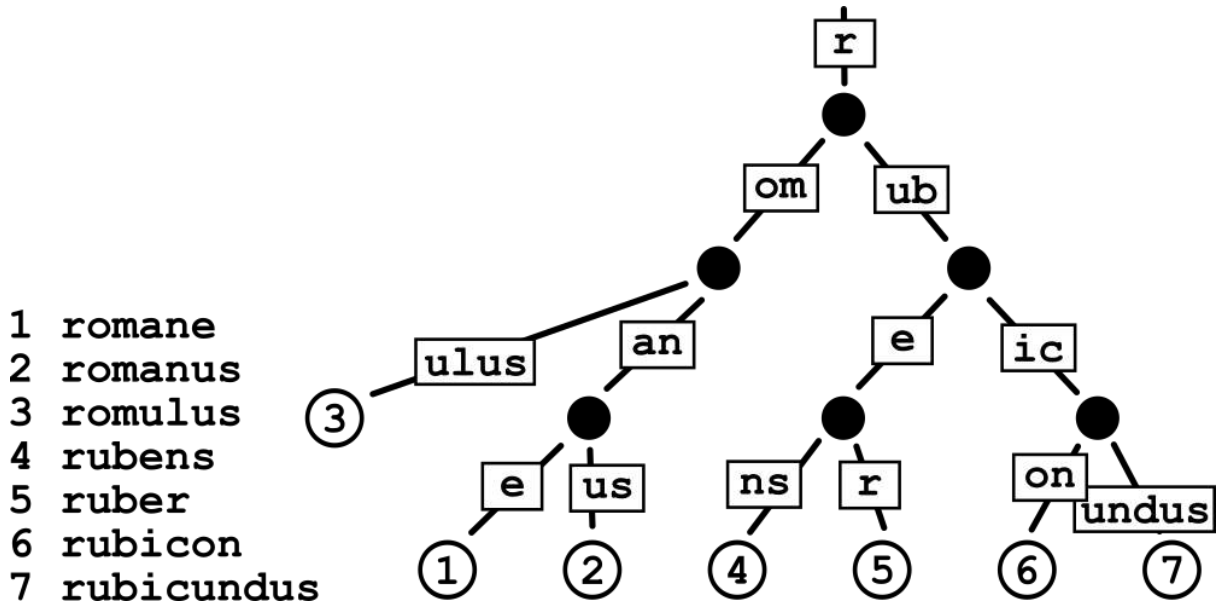
RediSearch in Action: FT.AGGREGATE

```
FT.AGGREGATE shipments "@box_area:[300 +inf]”
      APPLY "year(@shipment_timestamp / 1000)" AS shipment_year
      GROUPBY 1 @shipment_year REDUCE COUNT 0 AS shipment_count
      SORTBY 2 @shipment_count DESC
      LIMIT 0 3
      APPLY "format(\"%sk+ Shipments\",floor(@shipment_count /
1000))" AS shipment_count
```

Autocomplete

Autocomplete/Suggestions

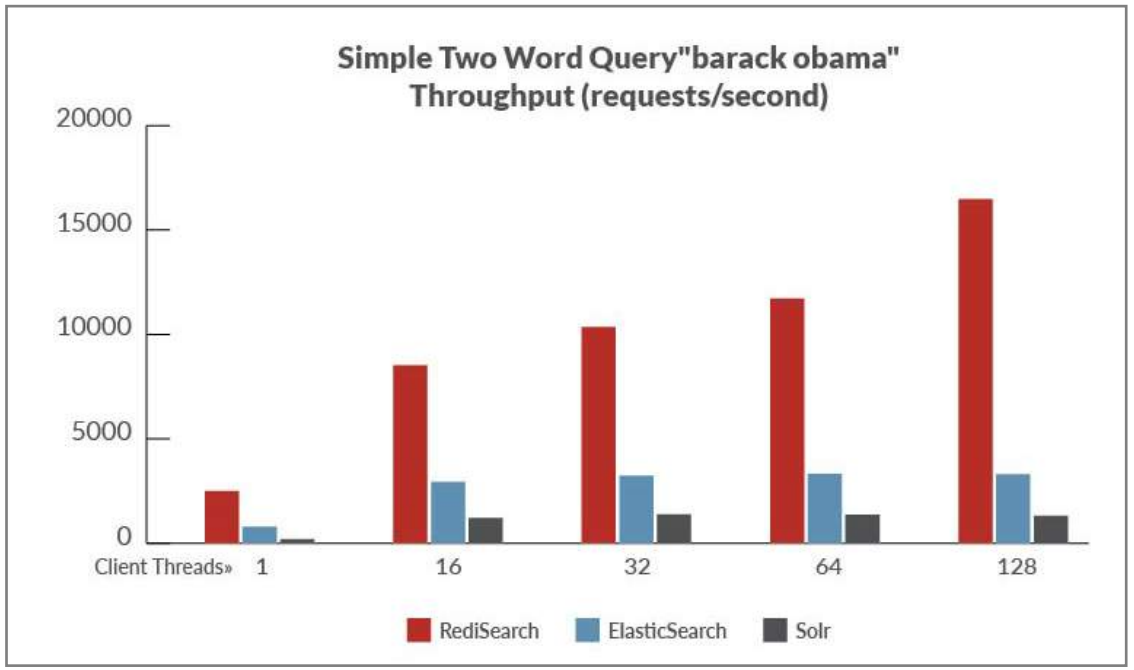
- In the module, but separate storage
- Radix tree-based, optimized for real-time, as-you-type completions
- Simple API
  - Add a suggestion (FT.SUGADD)
  - Get a suggestion (FT.SUGGET)
  - Delete a suggestion (FT.SUGDEL)
- Specify or increment “score” of each item to create custom sortings

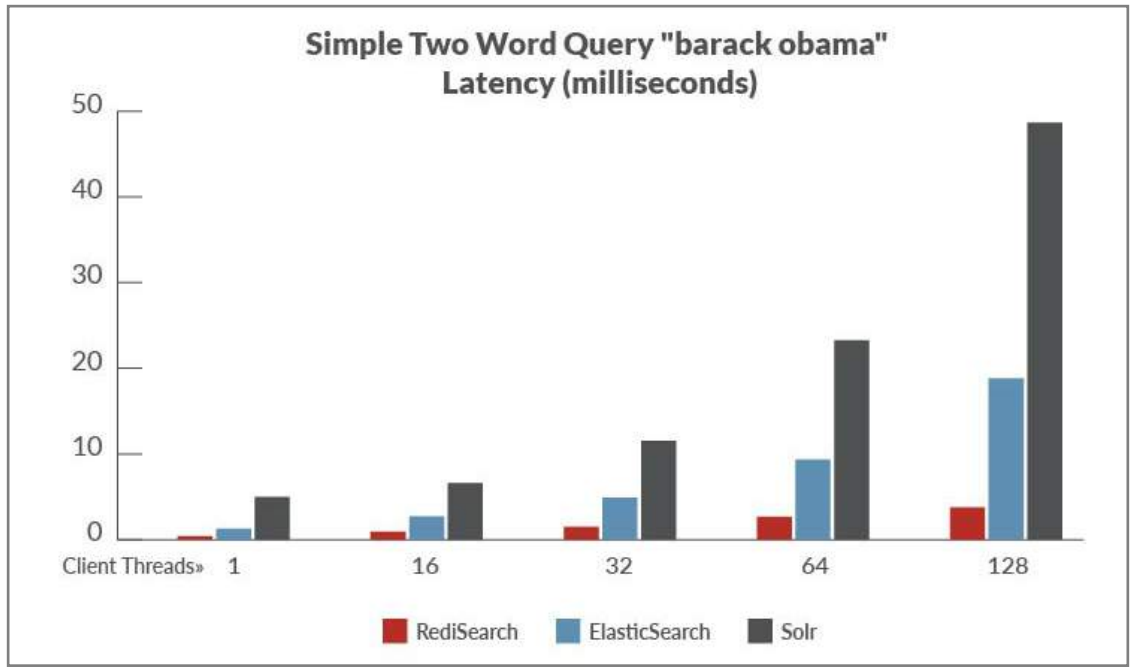


5x

Better Throughput and Latency than Elasticsearch

- ✓ Multiple language support
- ✓ Document and field scoring
- ✓ Numeric Filtering
- ✓ Stemming
- ✓ Auto-suggest
- ✓ Filtering by property





redislabs
HOME OF REDIS

Redis Graph

Components of a property graph database

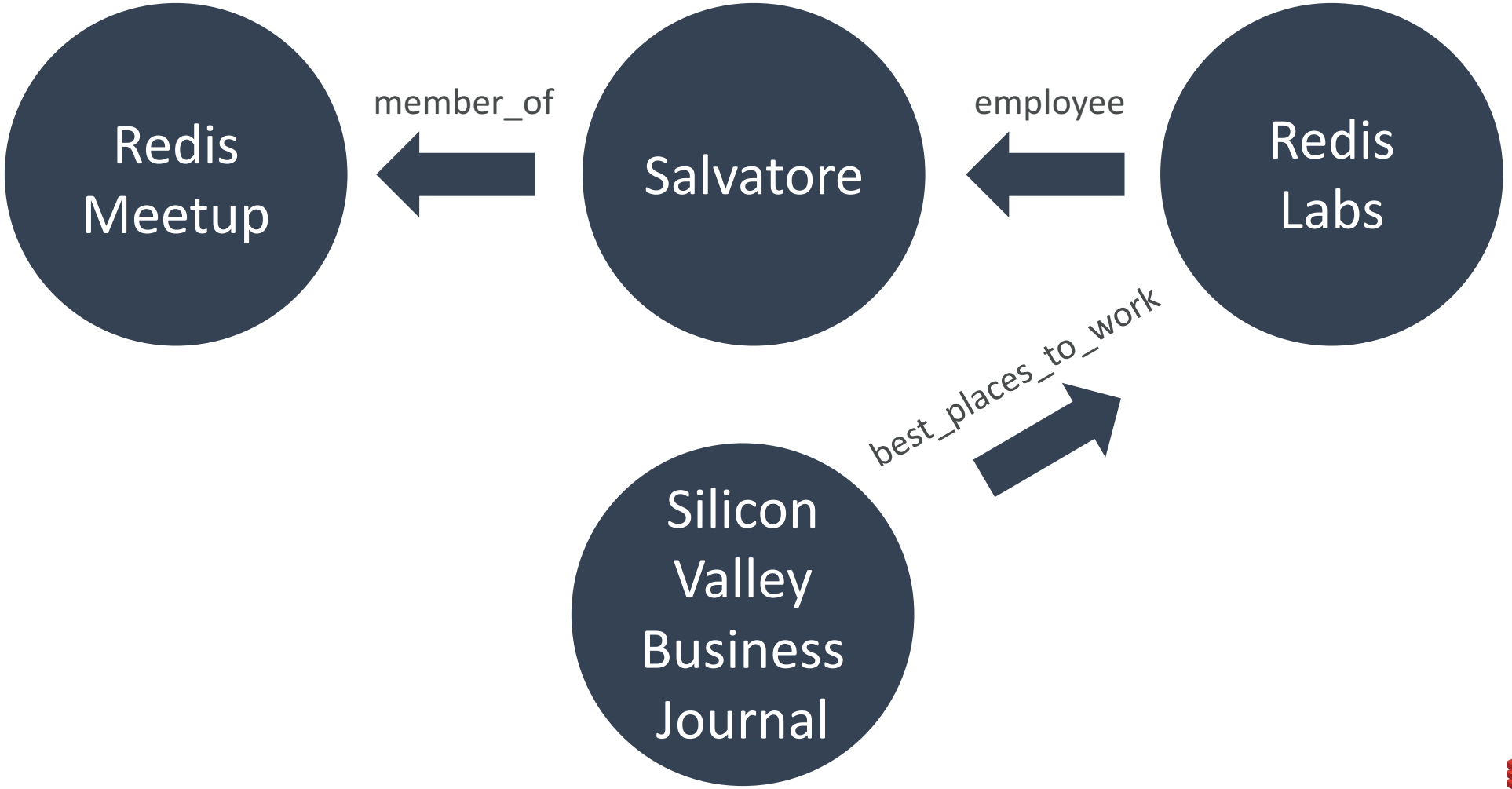
Nodes

- represents items/entities
- properties
- document/records

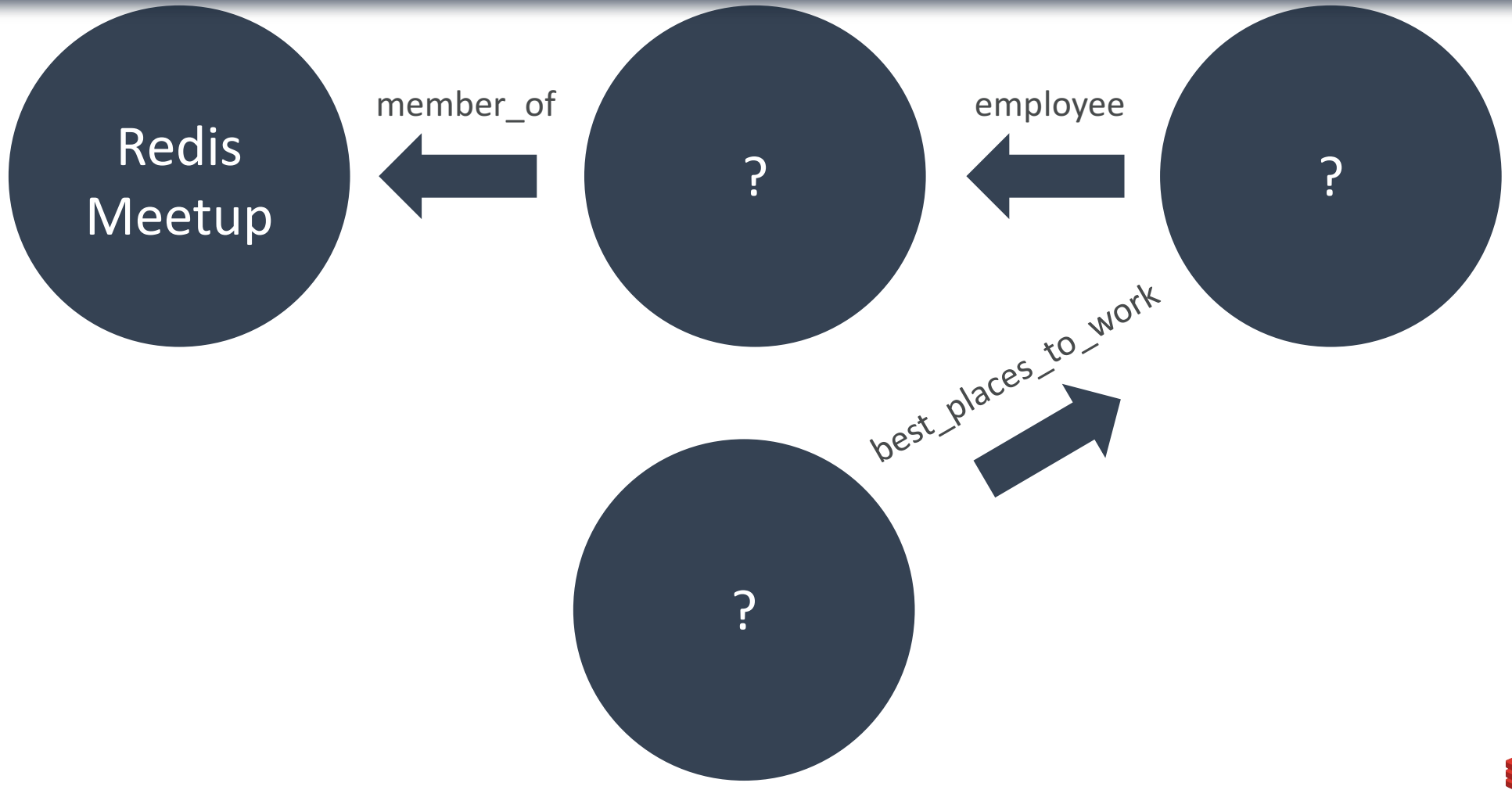
Relationships

- AKA graphs, edges
- attributes (connections)
- labeled
- adhoc (unlike relational databases)

Graph Relationship Example

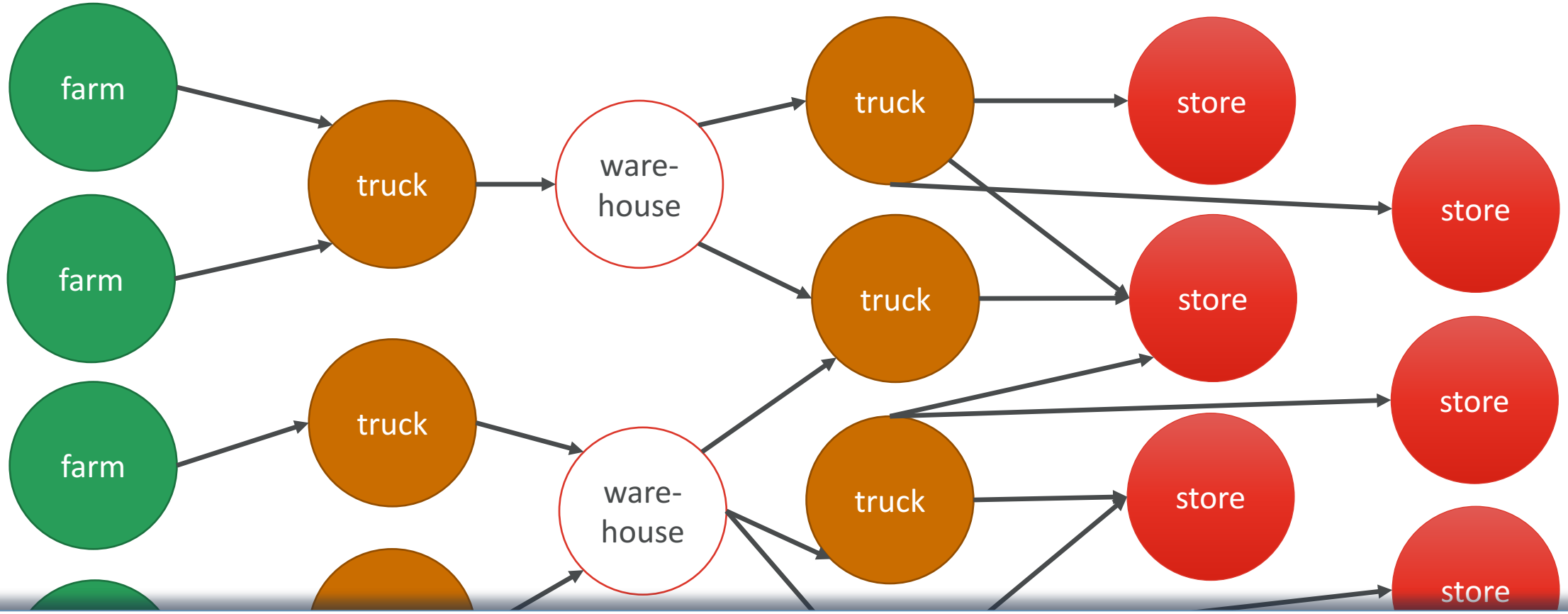


```
(:meetup { title : “RedisMeetup” }) <-[:memberOf]- (:person)<-
  [:employee]-(:company)<-[:bestPlaceToWork]-(:publication)
```




Solving Problems with Graphs

Supply chains and contamination




```
(:farm)-[:shipped]->(:truck)-[:shipped]->(:warehouse)-
[:shipped]->(:truck)-[:shipped]->(:store { name: "#26" })
```

How we do graph

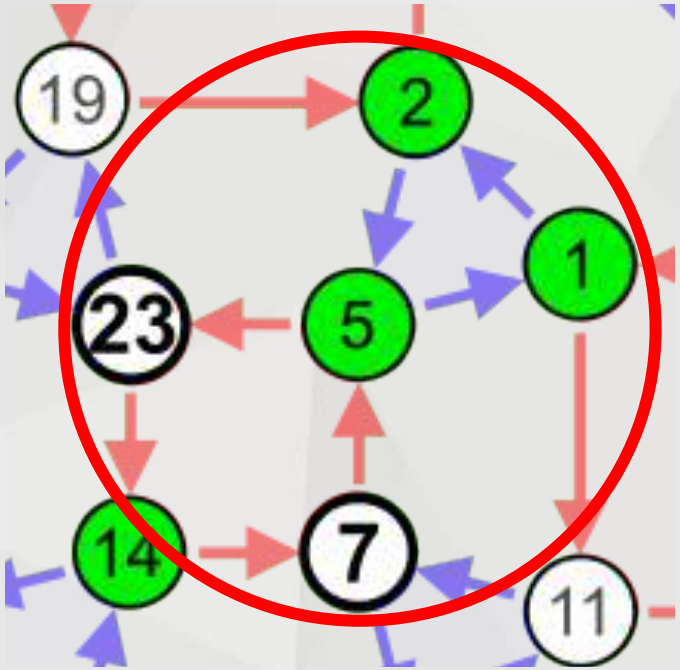
The problem with graph

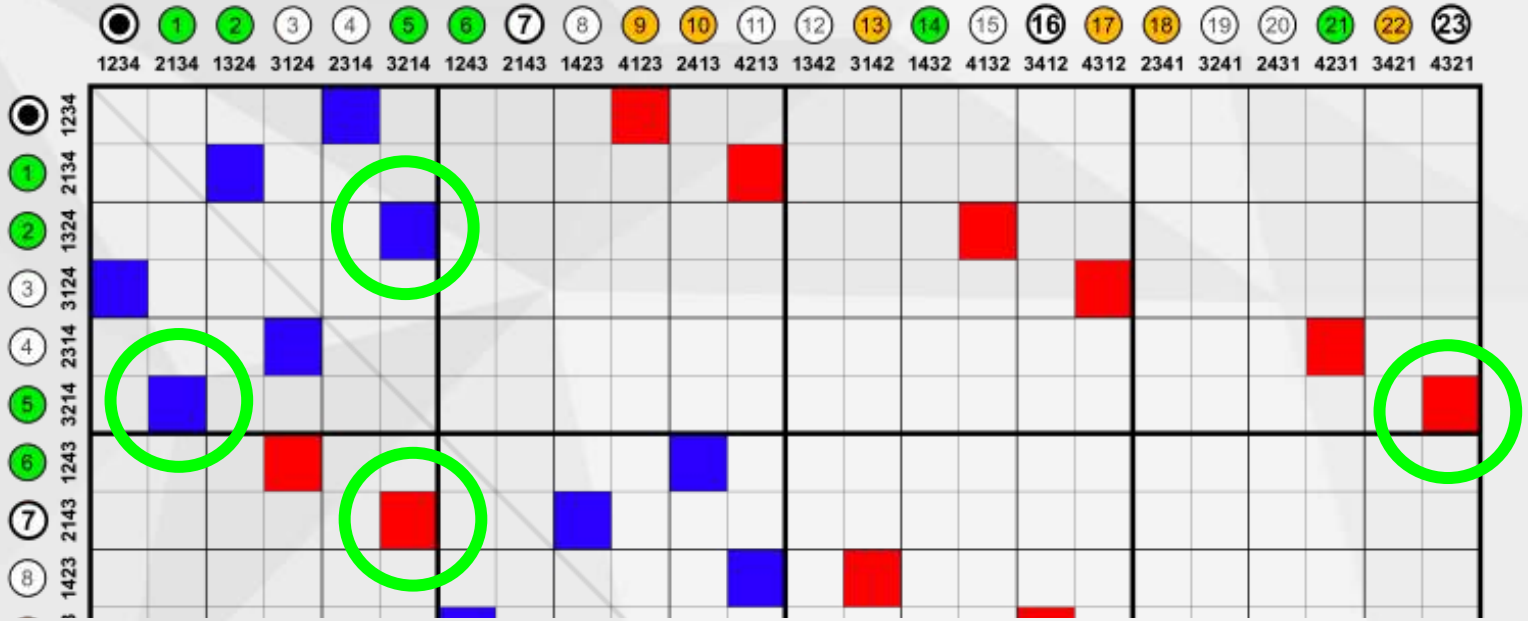
- Represented on top of other databases. Adding capabilities to existing system but….
  - Tabular – No [efficient] way to index
  - Documents – Documents are nodes and relationships are indexes

- Formal Graphs – Effective, but restricted
  - Adjacency lists – Better but many o(n) operations

Graph is useful but hard.

How it works: Binary Adjacency Matrix





Problem: It’s huge.

- 2 Billion users.
- Each user has average 338 friends

$$\frac{2{,}000{,}000{,}000 \times 338}{2{,}000{,}000{,}000^2}$$

0.000000169% utilisation

2 Part Solution Solution

Sparse Matrix

. . 8 .
. 5 2 .
. . . .

Convert binary matrix to a sparse matrix for storage compactness

GraphBLAS

- Highly optimized matrix operations over sparse matrix
- All the basics of linear algebra
  - Linear algebra can express the graph operations

Performance

	Neo4j	Redis Graph	Redis Graph Improvement
Create the graph (1,791,489 Nodes, 28,511,807 Edges)	5:30 min	2:30 min	**2.2x faster**
Return a node	2,464 ms	70 ms	**35x faster**
Find direct neighbours	2,960 ms	55 ms	**49x faster**
Who's connected	2,496 ms	70 ms	**35x faster**
Two hops away (left given)	2,816 ms	62 ms	**45x faster**
Two hops away (right given)	2,710 ms	720 ms	**3.8x faster**
Memory Consumption	3,000 mb	880 mb	**3.4x smaller**

redislabs
HOME OF REDIS

What's next?

redisconf19

April 1, 2019 Training

April 2-3, 2019 Sessions

San Francisco

redisconf.com

The Redis user community is one of the strongest in the world, with widespread affinity and adoption by developers worldwide

As a Redis Star, you will have access to:

- Knowledge Center
- Relationship Forging
- Rewards & recognition

To learn more and join the Redis Stars, visit:

redislabs.com/community/redis-stars/