In-Memory Computing SUMMIT | NORTH AMERICA 2018

Redis Streams, Functions and Data Structures

Dave Nielsen
Redis Labs

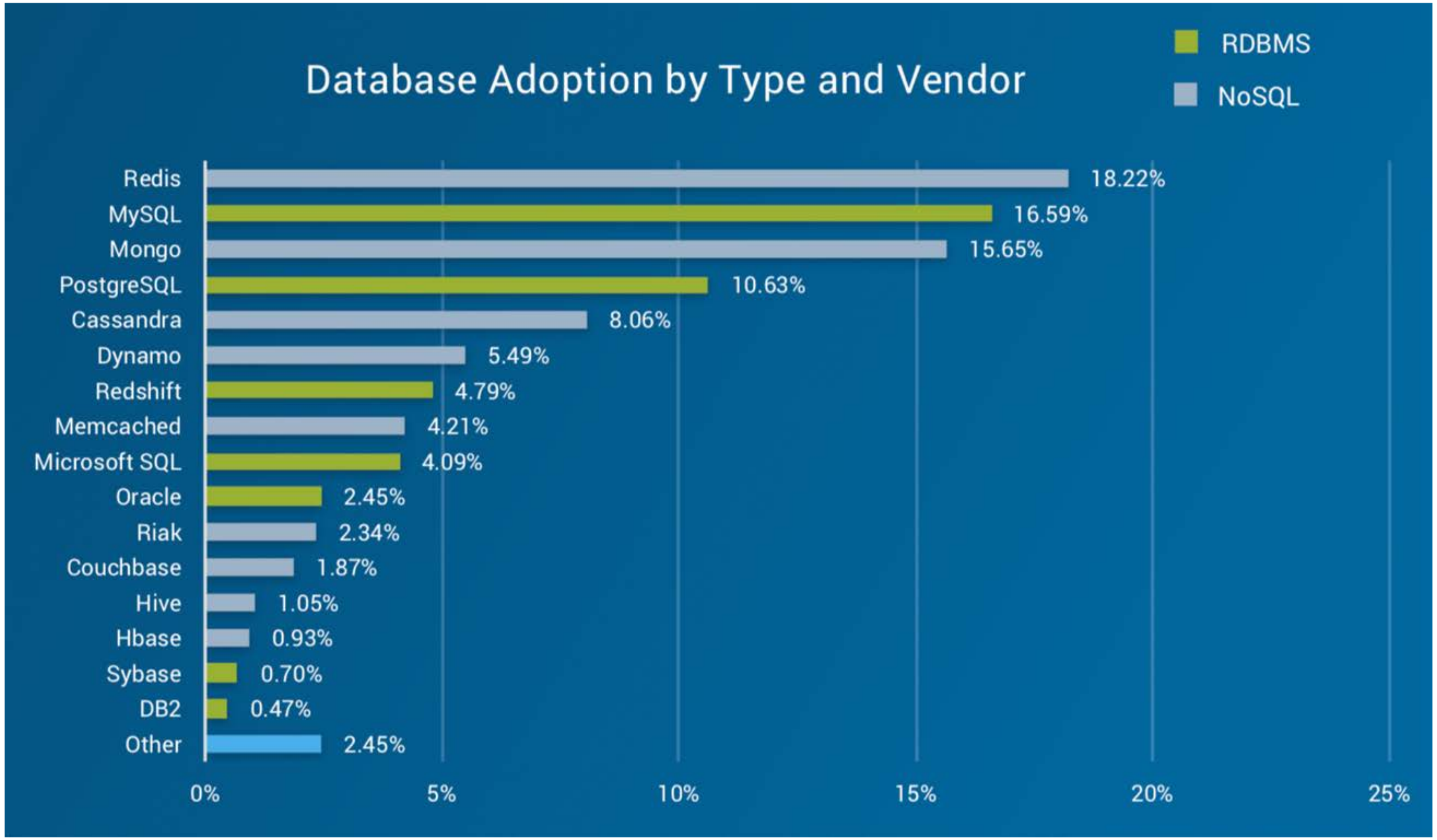
Agenda

About Redis

Use Cases

Redis Streams

"Most Popular Database on AWS" – Sumo Logic 2016 Survey

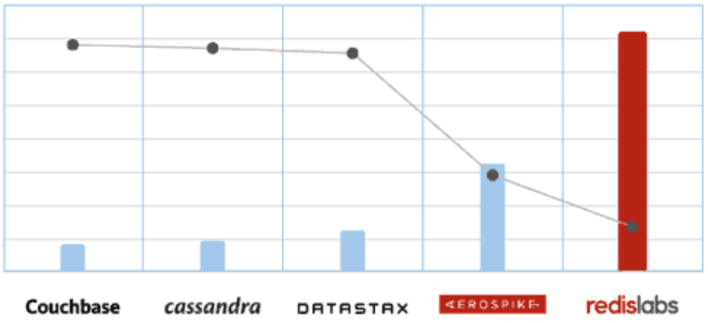


Redis Top Differentiators

1

Performance

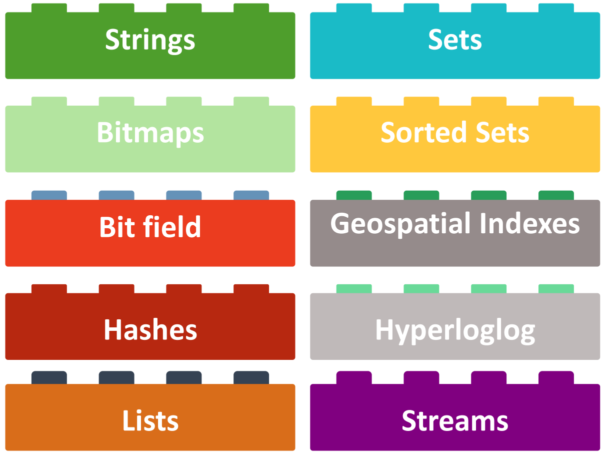
NoSQL Benchmark



2

Simplicity

Redis Data Structures

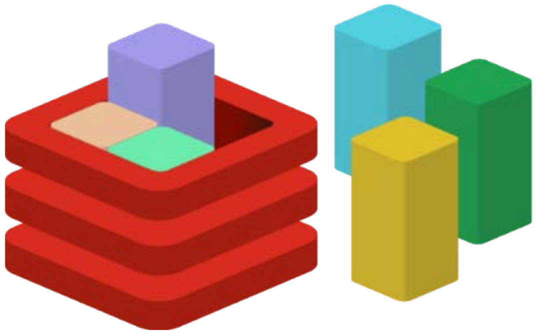


Strings
Sets
Bitmaps
Sorted Sets
Bit field
Geospatial Indexes
Hashes
Hyperloglog
Lists
Streams

3

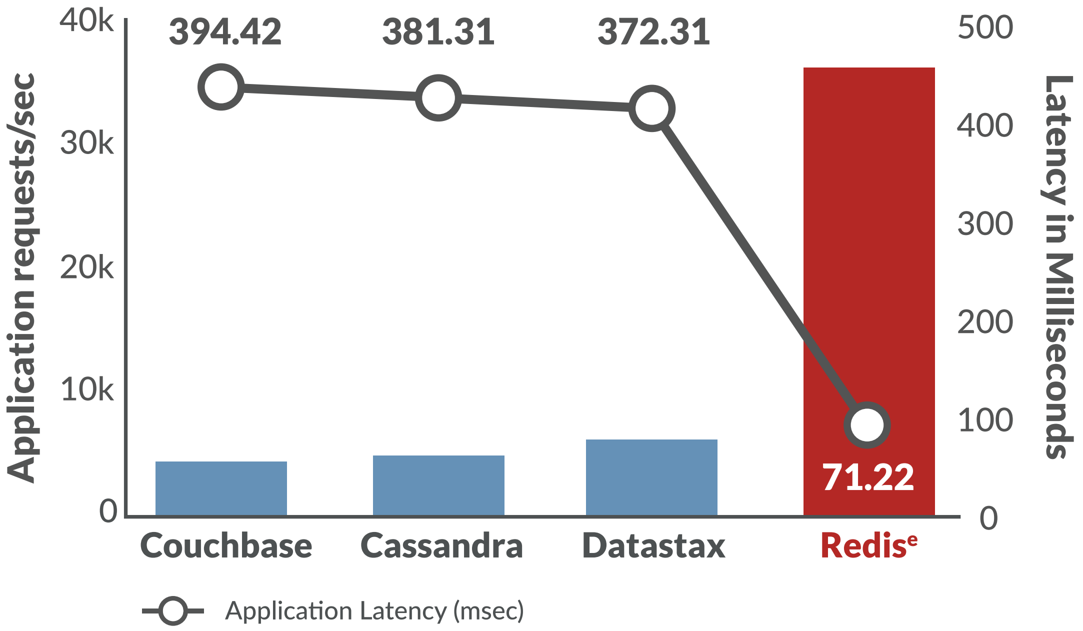
Extensibility

Redis Modules



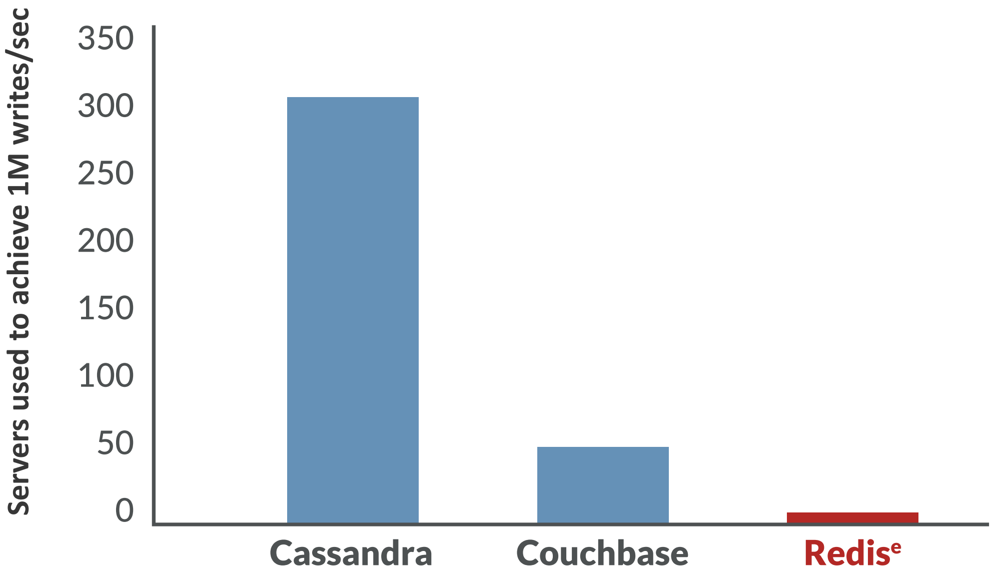
1 Performance: The Most Powerful Database

Highest Throughput at Lowest Latency in High Volume of Writes Scenario



Benchmarks performed by Avalon Consulting Group

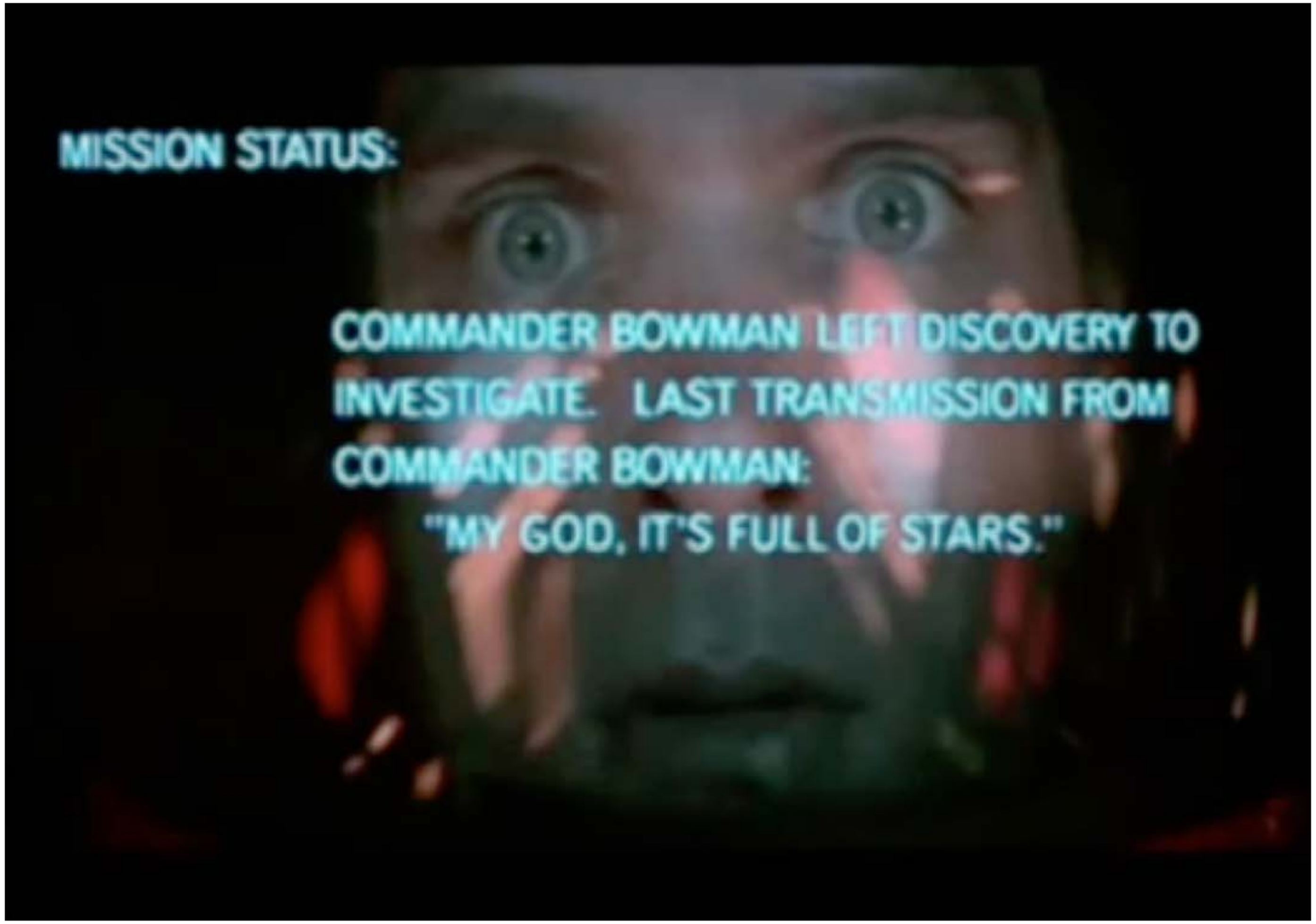
Least Servers Needed to Deliver 1 Million Writes/Sec



	Cassandra	Couchbase	Redis^e
ANNUAL COST	$2,226,216	$371,040	$14,832
COST COMPARED TO REDIS^e	150X	25X	

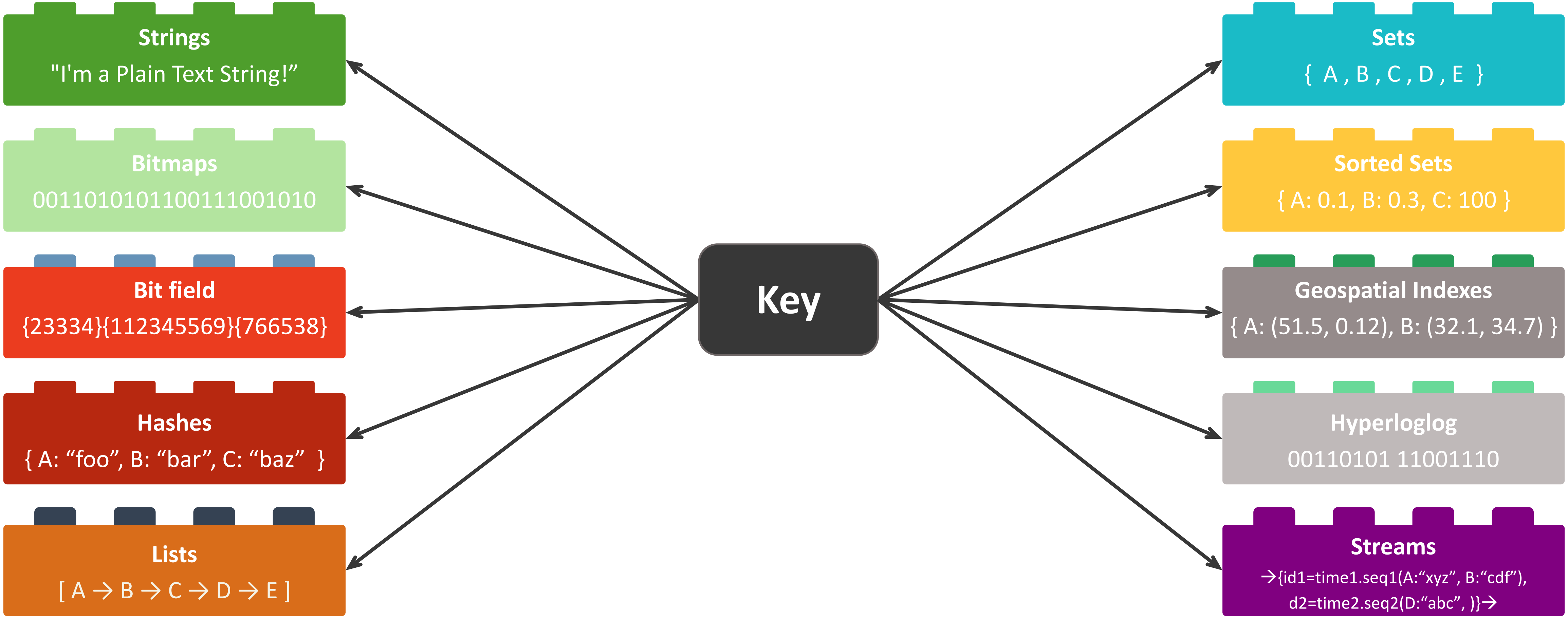
Benchmarks published in the Google blog

2 Simplicity: Data Structures - Redis' Building Blocks


MISSION STATUS:
COMMANDER BOWMAN LEFT DISCOVERY TO INVESTIGATE. LAST TRANSMISSION FROM COMMANDER BOWMAN:
"MY GOD, IT'S FULL OF STARS."

"REDIS IS FULL OF DATA STRUCTURES!"

2 Simplicity: Redis Data Structures – ’Legos’



Strings
"I'm a Plain Text String!”

Bitmaps
0011010101100111001010

Bit field
{23334}{112345569}{766538}

Hashes
{ A: “foo”, B: “bar”, C: “baz” }

Lists
[A → B → C → D → E]

Key

Sets
{ A , B , C , D , E }

Sorted Sets
{ A: 0.1, B: 0.3, C: 100 }

Geospatial Indexes
{ A: (51.5, 0.12), B: (32.1, 34.7) }

Hyperloglog
00110101 11001110

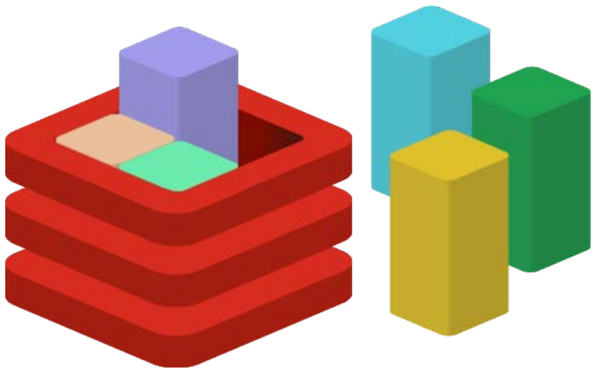
Streams
→{id1=time1.seq1(A:“xyz”, B:“cdf”), d2=time2.seq2(D:“abc”,)}→

”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 Functionality

- RediSearch
- Redis-ML
- Redis Graph
- ReJSON
- Rebloom
- Neural-Redis
- Redis-Cell
- Redis-TDigest
- Redis-Timerseries

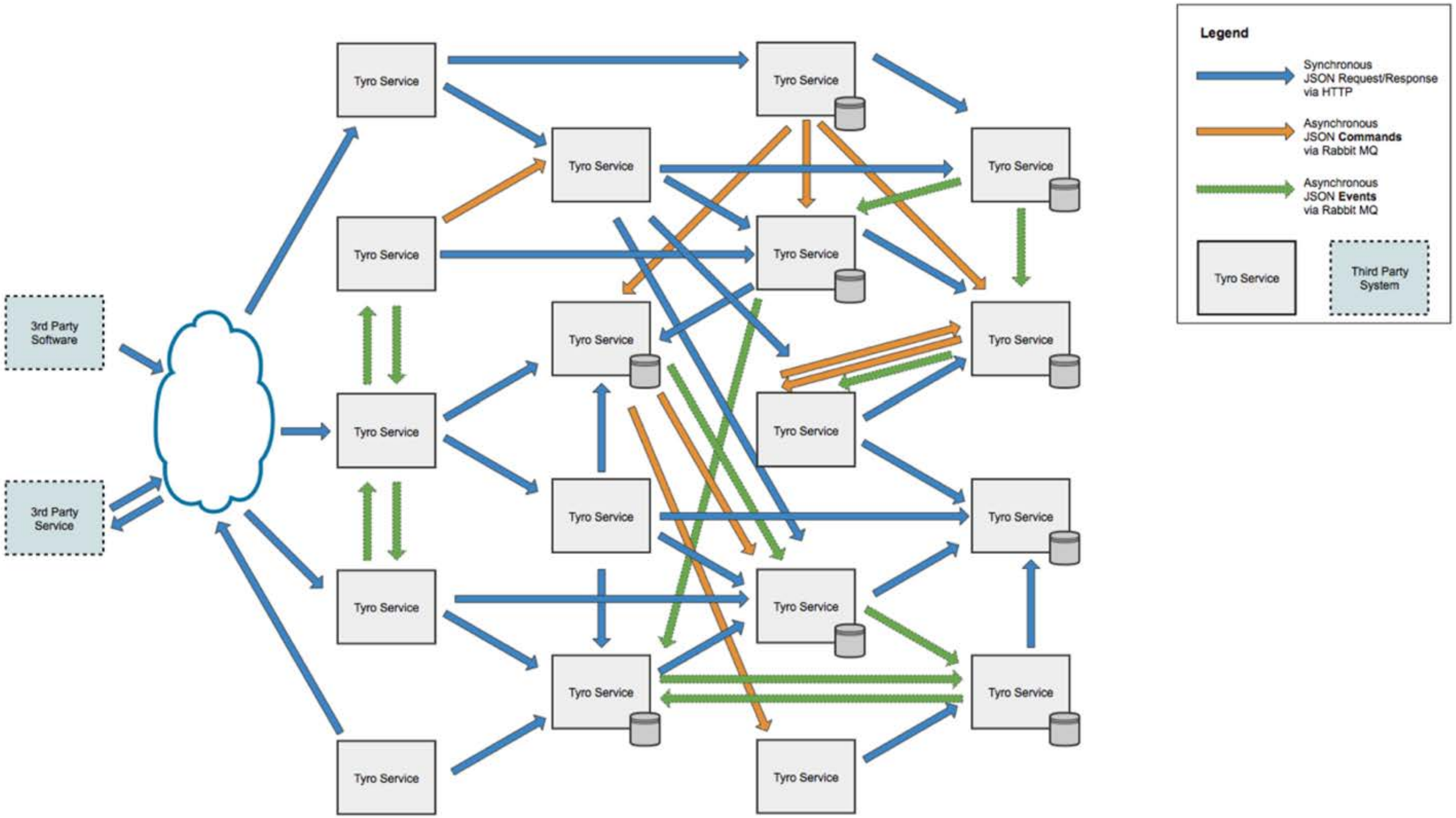
- Redis-Rating
- Redis-Cuckoofilter
- Cthulhu
- Redis Snowflake
- redis-roaring
- Session Gate
- ReDe
- TopK
- countminsketch



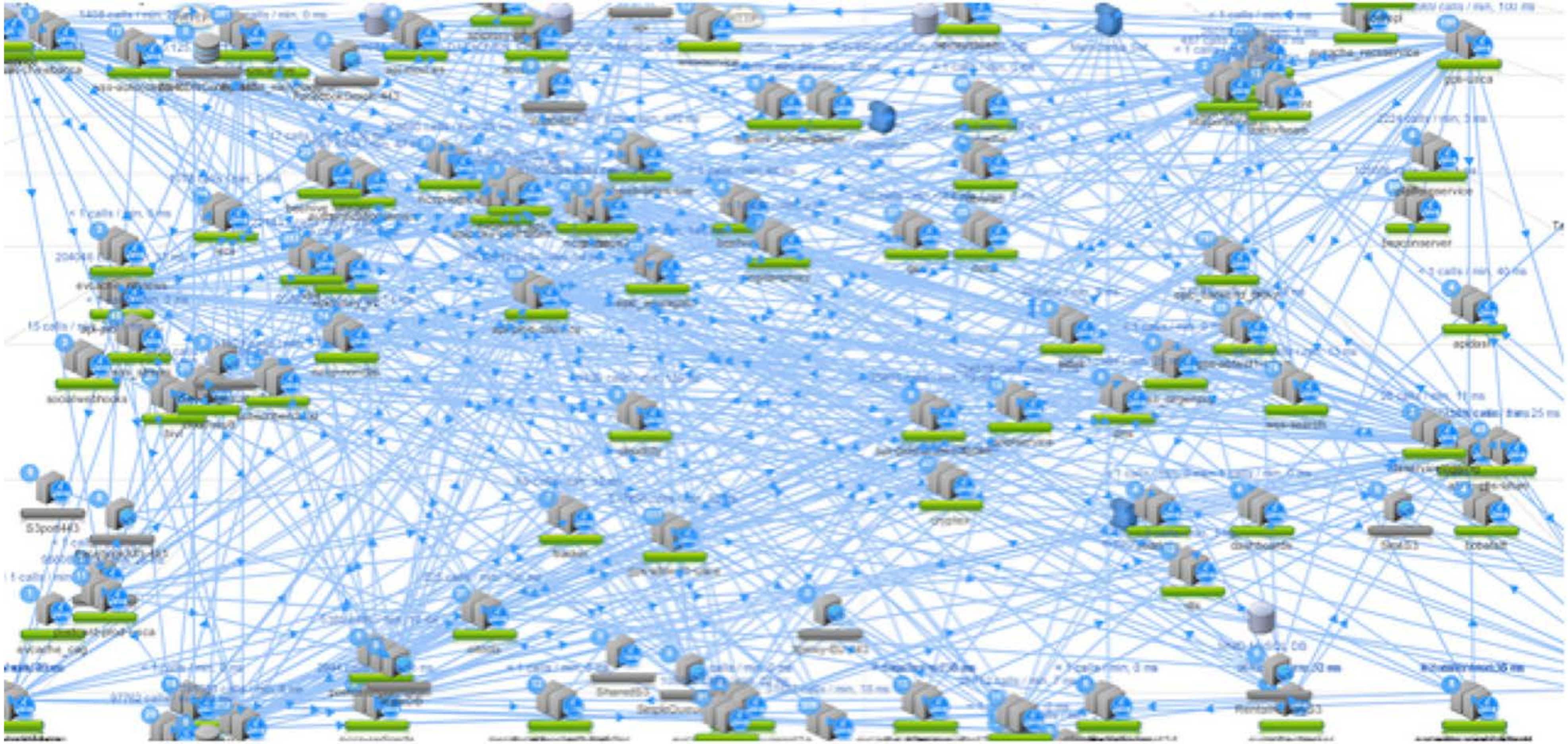
Microservices

Click to add text

SOA vs. Microservices



Microservices at Netflix

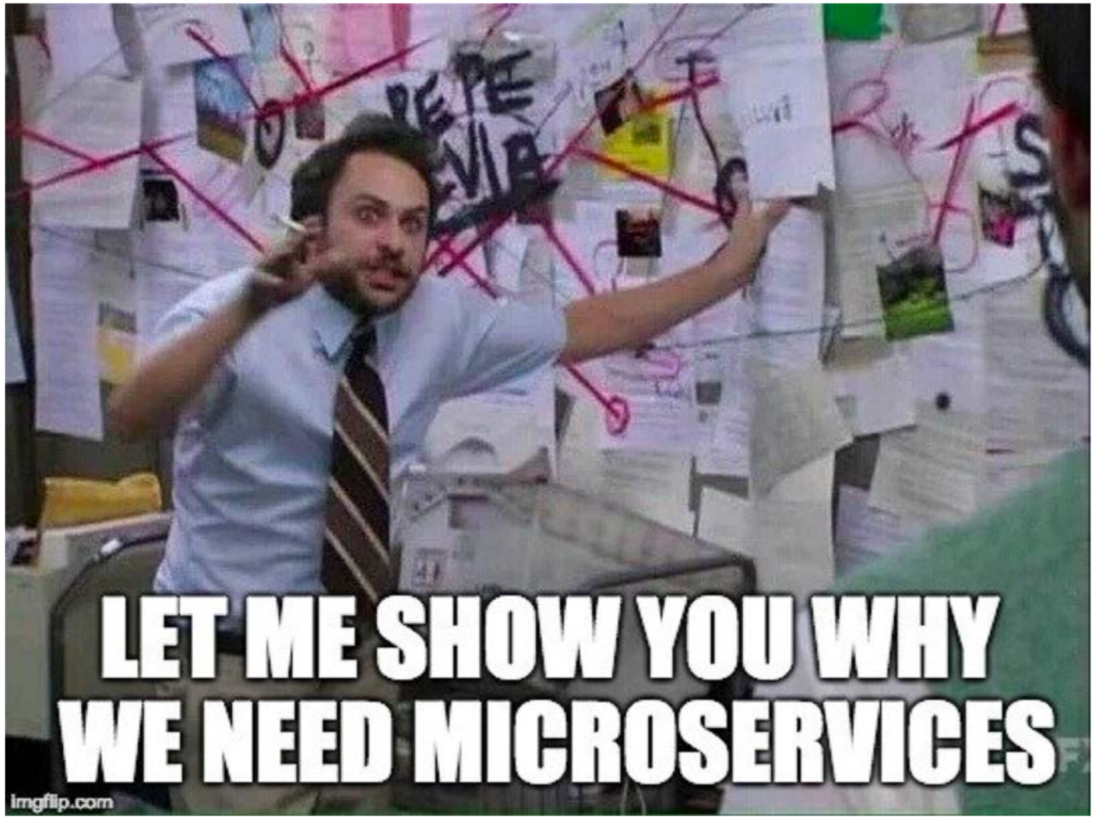


Monolith or Microservices?

Click to add text

Benefits of Microservices

- Microservices are hot. It seems like everyone is using them

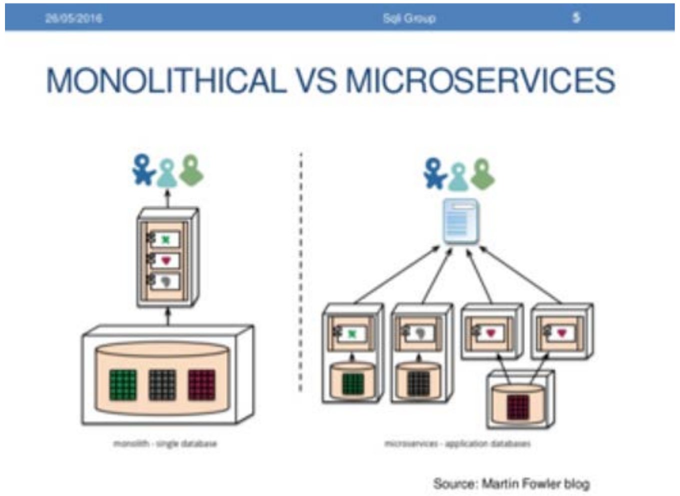


Benefits of Microservices

- Make it perform faster or scale better
- Extend an application's capabilities more easily
- Add new features more quickly and easily
- Improve maintainability
- Reduce vulnerabilities

But, Microservices are Complicated

- A lot more going on that meets the eye.



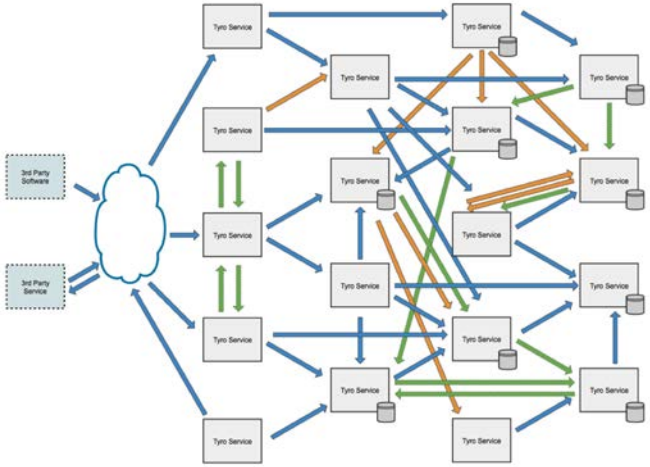
26/05/2016 Sql Group 5

MONOLITHICAL VS MICROSERVICES

monolith - single database

microservices - application databases

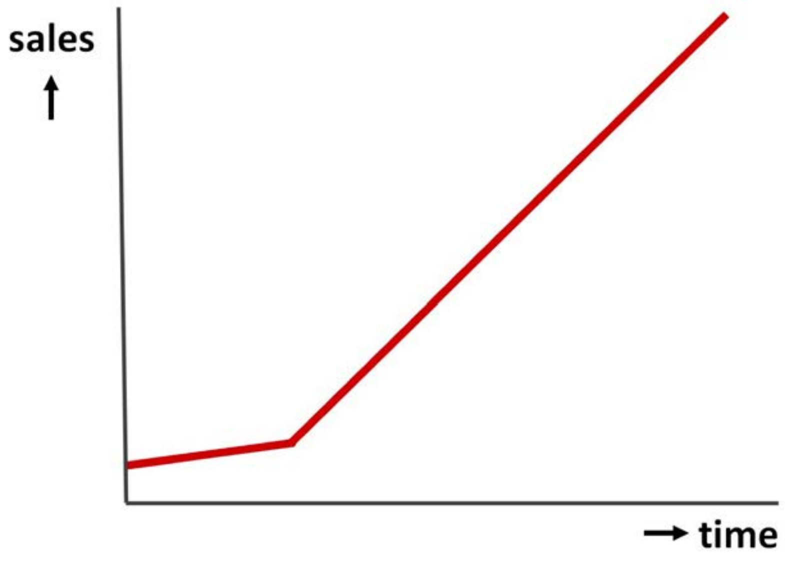
Source: Martin Fowler blog

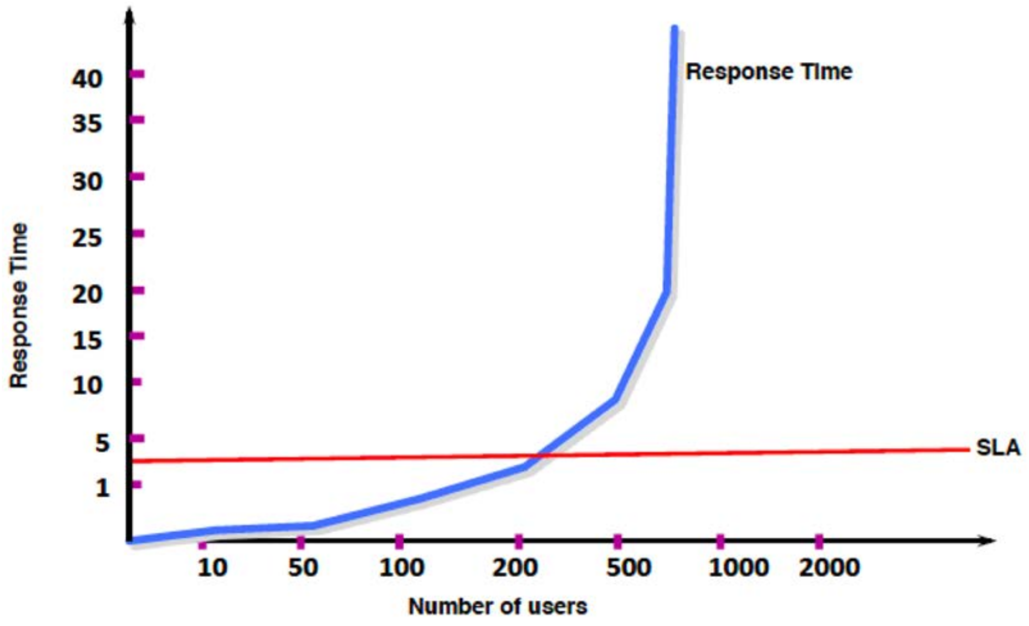


NETFLIX

Be Prepared for Success

- What to do when your app begins to hockey stick
  - Duck tape the parts when they break?
  - Do you rewrite your app with scalability in mind?

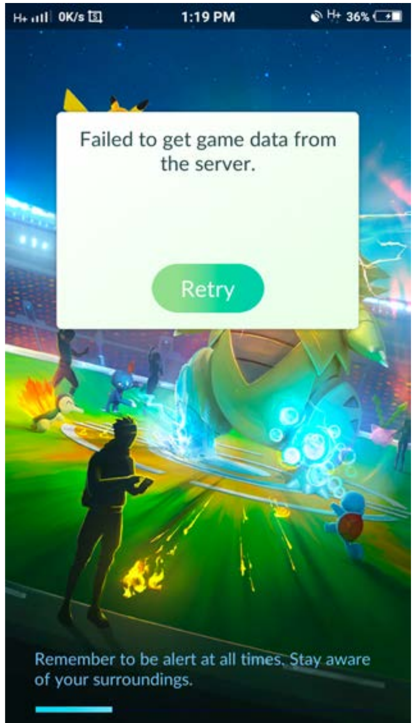




You Can Do Both with Redis & Kubernetes

- Redis became famous by solving web scale data problems
  - Remember the Twitter Fail Whale?
- Kubernetes became famous by solving hockey stick problem
  - Remember Pokemon Go?

Twitter is over capacity.

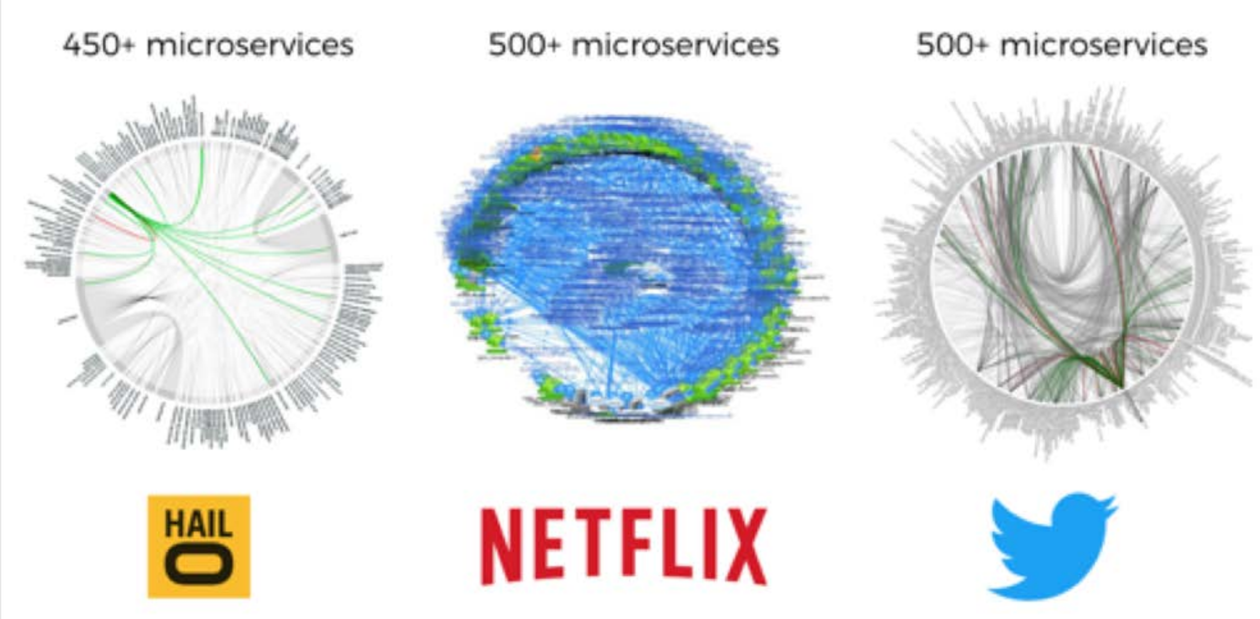


Failed to get game data from the server.

Remember to be alert at all times. Stay aware of your surroundings.

And Scale with Redis and Microservices

- In many cases, Monolith is the right way to start
- Smaller apps and small teams don’t need the overhead and unnecessary complexity of Microservices Architecture
- But when its time to scale, use Redis and Microservices



Use Cases

Click to add text

Use Cases

Top 4

- Cache
- Session Store
- Metering
- Fast Data Ingest

More:

•Primary Database

•Real-time Analytics

•Messaging

•Recommendations

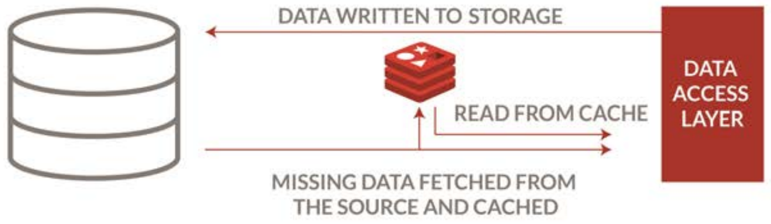
•High-speed Transactions

•Search – RediSearch

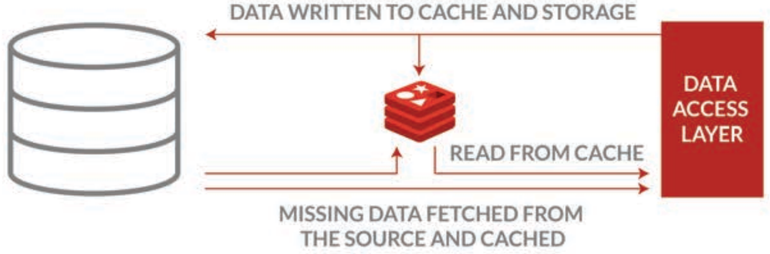
•Geo Spatial Indexing

•Many more …

1. Redis as a Cache



Look-aside cache


DATA WRITTEN TO CACHE AND STORAGE
READ FROM CACHE
DATA ACCESS LAYER
MISSING DATA FETCHED FROM THE SOURCE AND CACHED

Write-through cache

When to use

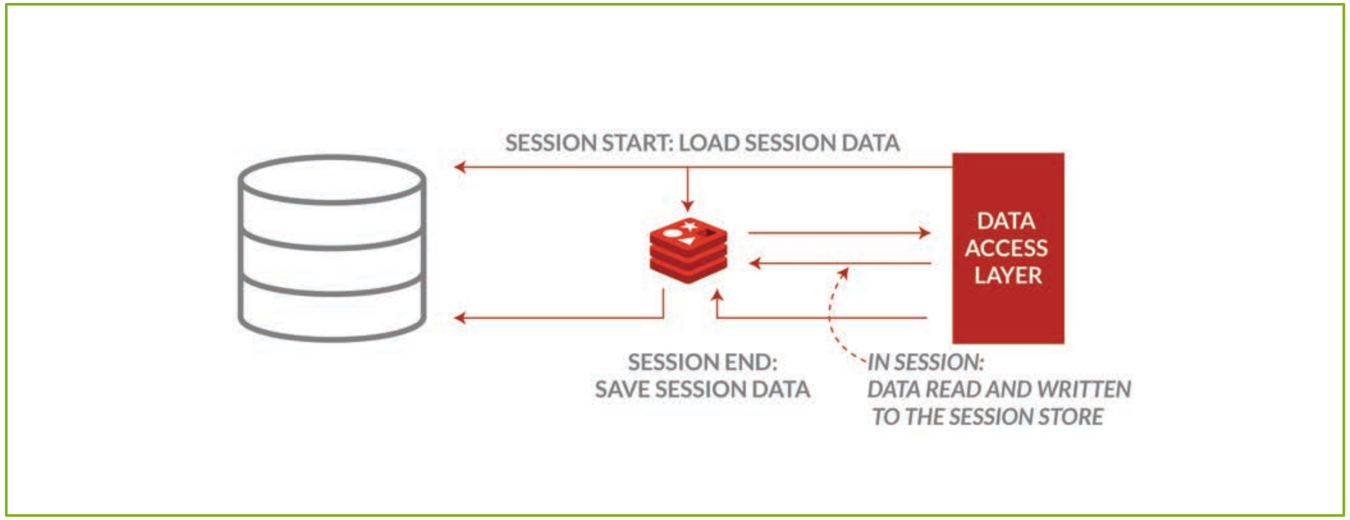
- Frequent reads, infrequent writes
- Data is shared between user sessions

Examples:

- Pictures, documents, videos, statements, reports, etc.

Microsoft STAPLES AMERICAN EXPRESS

2. Redis as a Session Store


SESSION START: LOAD SESSION DATA
DATA ACCESS LAYER
SESSION END: SAVE SESSION DATA
IN SESSION: DATA READ AND WRITTEN TO THE SESSION STORE

When to use

Session based apps with frequent reads *and writes*

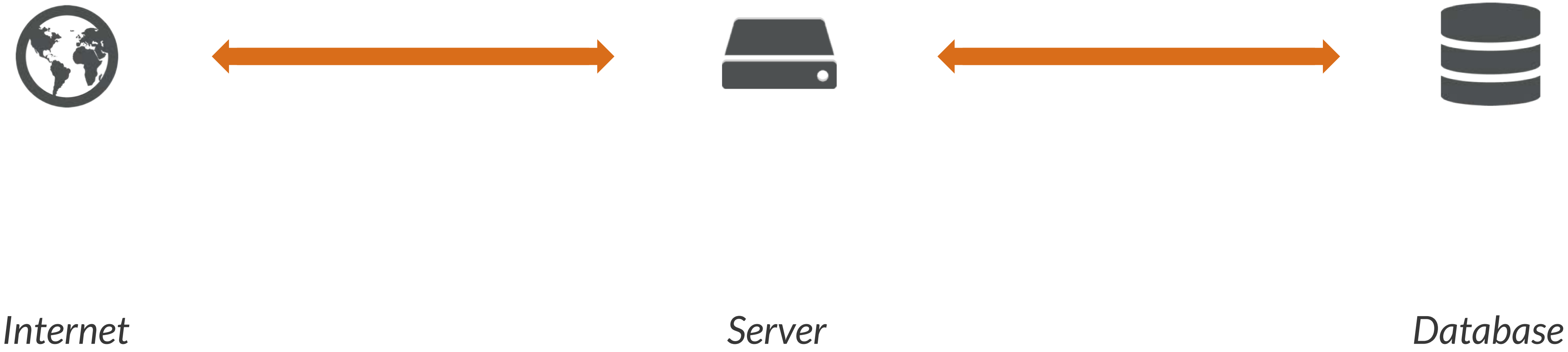
Data is isolated between sessions

Examples:

e-Commerce, gaming, social applications, etc.

verizon intuit WELLS FARGO

In a simple world

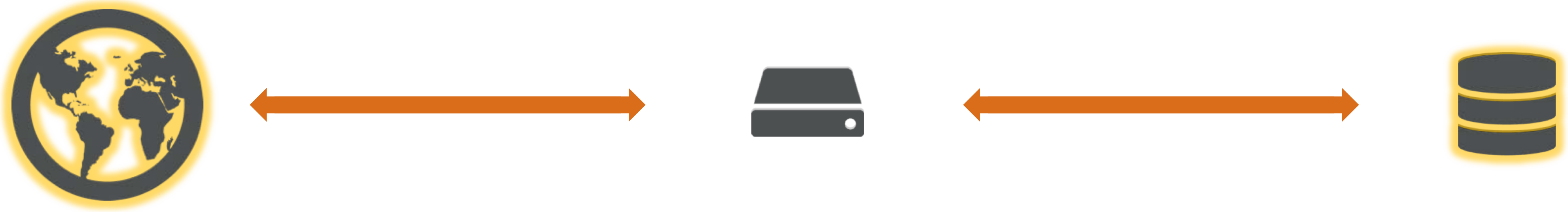


Internet

Server

Database

Good problems



Internet

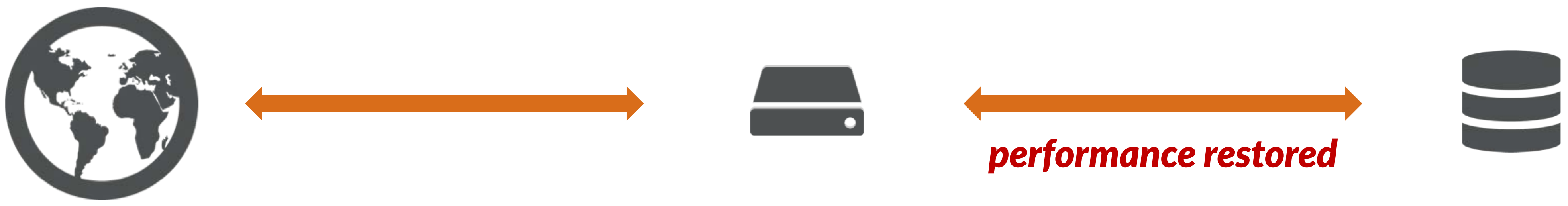
Traffic Grows…

Server

Database

Struggles

Good solution



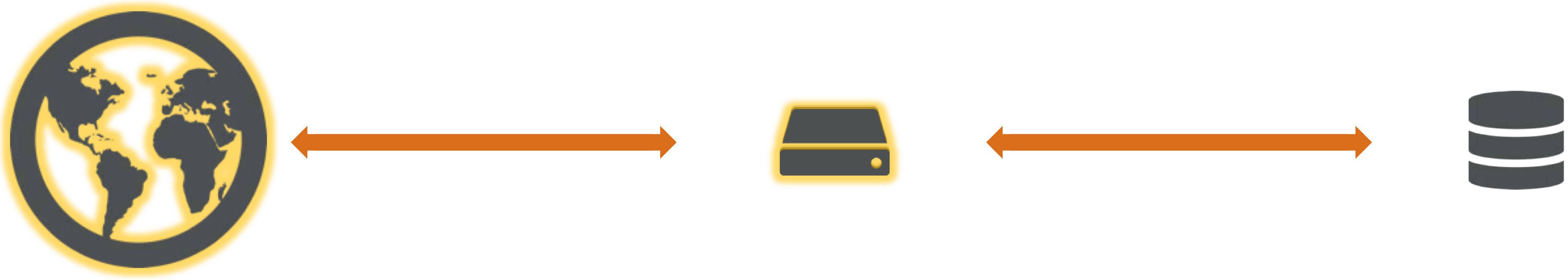
Internet

Server

Session storage on the server

Database

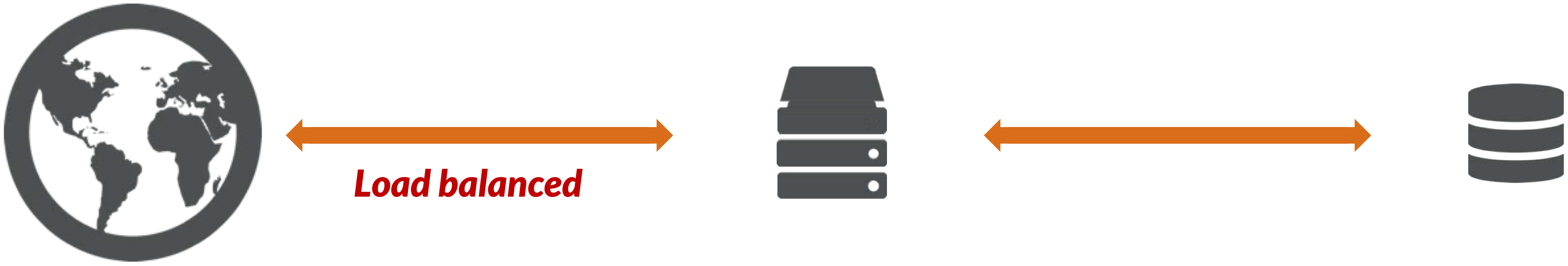
More good problems



Struggling

Session storage on the server

Problematic Solutions



Load balanced

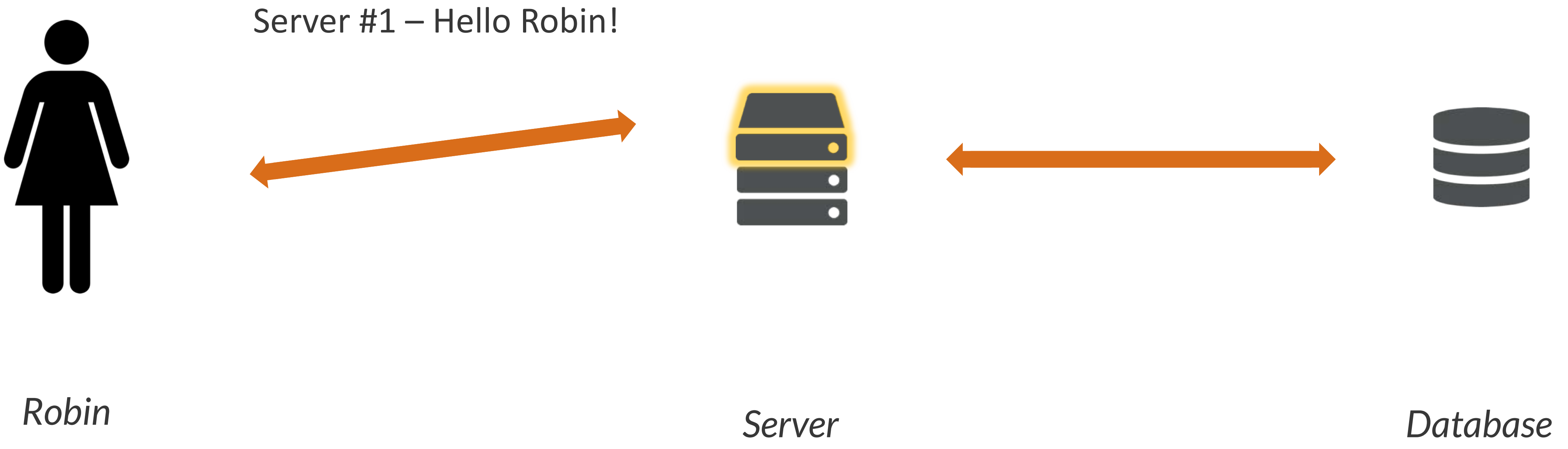
Internet

Server

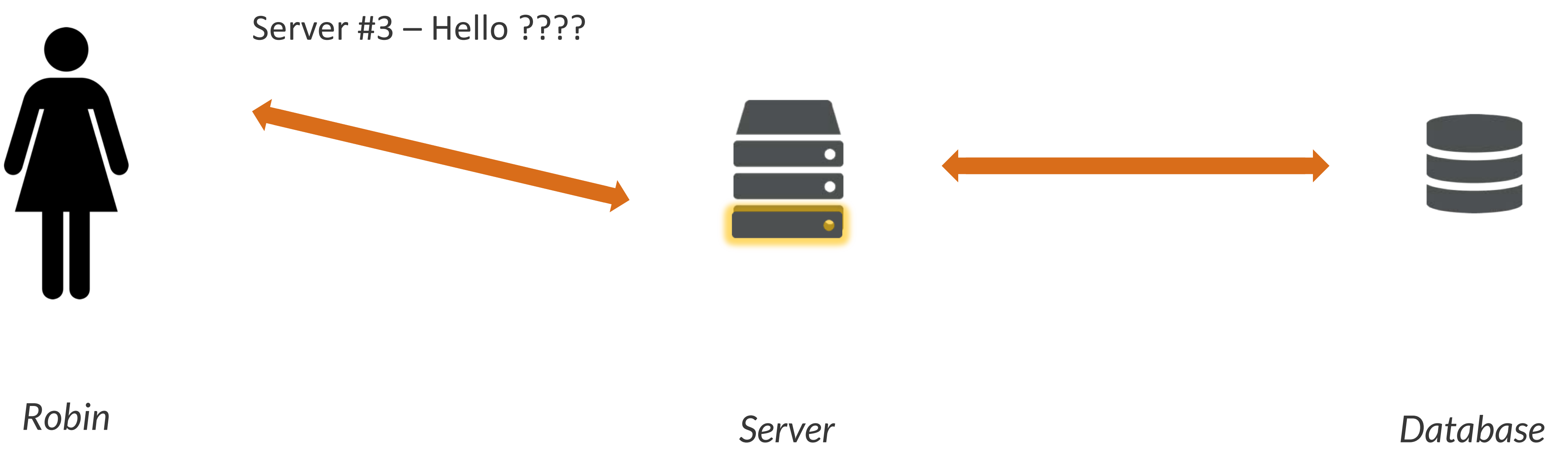
Session storage
on the server

Database

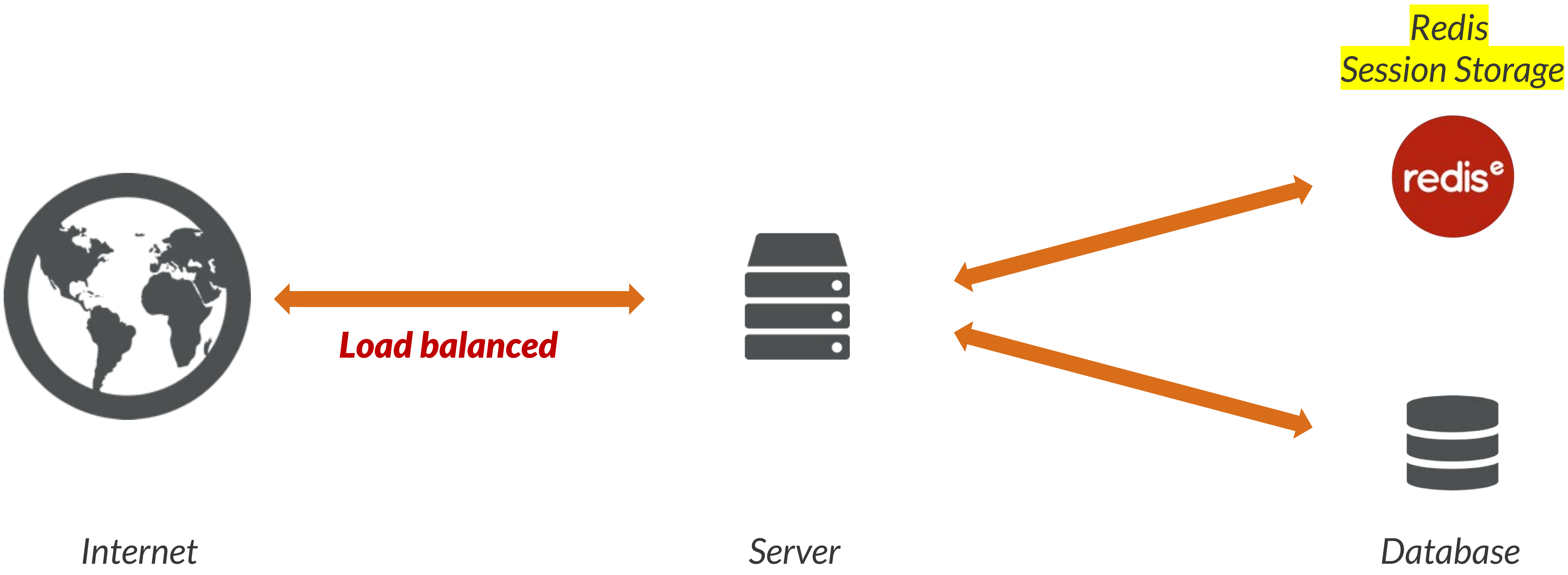
Multiple Servers + On-server Sessions?



Multiple Servers + On-server Sessions?



Better solution



Use Redis Hash For Session Store

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
```

```
DEL usersession:1
```

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

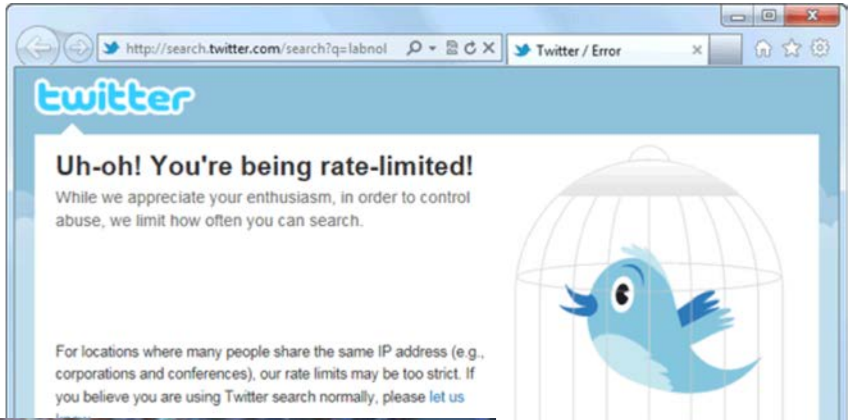
3. Redis for Metering

Use Case: Rate-limiting

Limit the peak load on your legacy database by limiting the number of queries per second to the highest threshold

How Redis helps you?

- Built-in counters
- Time-to-live
- Single-threaded architecture assures serializability



Uh-oh! You're being rate-limited!

While we appreciate your enthusiasm, in order to control abuse, we limit how often you can search.

For locations where many people share the same IP address (e.g., corporations and conferences), our rate limits may be too strict. If you believe you are using Twitter search normally, please let us

4. Redis for Fast Data Ingest

Use Cases:

- Real-time analytics
- IoT
- Log collection, time-series

How Redis helps you?

- Pub/Sub
- List
- Sorted Set

etermax fiserv. verizon

Do more with Redis

- Caching
- Session Store
- Metering
- Fast Data Ingest

- Primary Database
- Real-time Analytics
- Messaging
- Recommendations
- High-speed Transactions
- Search – RediSearch
- Geo Spatial Indexing

It's a Swiss Army Knife for data processing

Managing Leaderboards w/ Redis Sorted Sets

Click to add text

In-Memory Computing SUMMIT NORTH AMERICA 2018

Leaderboard with Sorted Sets Example

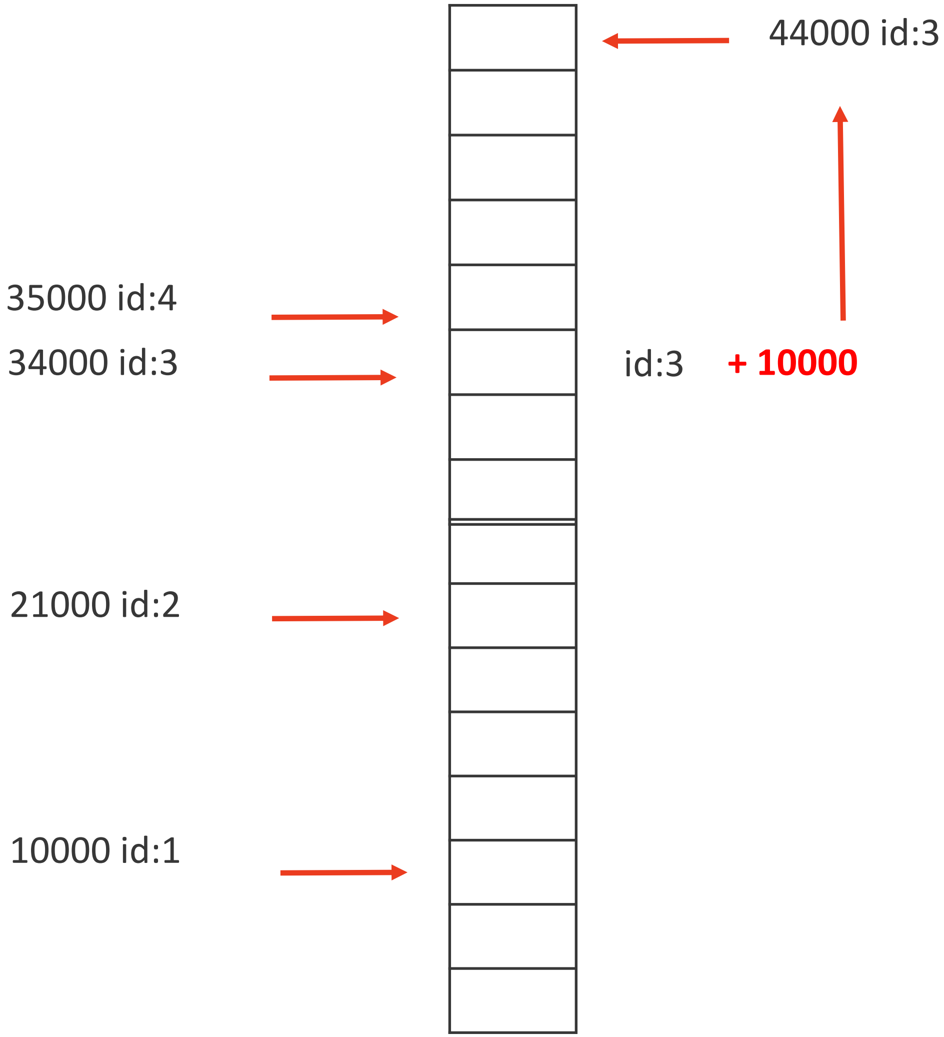
The Problem

- 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



```
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
```

Redis Streams

Click to add text

In-Memory Computing SUMMIT NORTH AMERICA 2018

Redis Streams

- 1st class Redis citizens
- An abstract data type that is not unlike a log
- Designed with time series data in mind
- Provide some "Kafkaesque" messaging abilities

Why invent yet another Redis thingamajig?

Necessity is the mother of invention

There ain't no such thing as a free lunch

The existing (i.e. lists, sorted sets, PubSub) isn't "good enough" for things like:

- Log-like data patterns
- At-least-once messaging with fan-out

And listpacks, radix trees & reading Kafka :)

The Log is hardly a new thing

A storage abstraction that is:

- Append-only, can be truncated
- A sequence of records ordered by time

A Logical Log is:

- Based on a logical offset, i.e. time (vs. bytes)
- Therefore time range queries
- Made up of in-memory data structures, naturally

Logging streams of semi-structured data

A data stream is a sequence of elements. Consider:

- Real time sensor readings, e.g. particle colliders
- IoT, e.g. the irrigation of avocado groves
- User activity in an application

…

- Messages in distributed systems

A side note about Distributed Systems

“A distributed system in which components located on networked computers communicate and coordinate their actions by passing messages” – Distributed Computing, Wikipedia

Includes: client-server, 3/n-tier, peer to peer, SOA, micro- & nanoservices, FaaS & serverless...

An observation

There are only two hard problems in distributed systems:

2. Exactly-once delivery

1. Guaranteed order of messages

2. Exactly-once delivery

- Mathias Verraes, on Twitter

Refresher on message delivery semantics

Fact #1: you can choose one and only one:

- *At-most-once* delivery, i.e. "shoot and forget"
- *At-least-once* delivery, i.e. explicit ack

Fact #2: *exactly-once* delivery doesn't exist

Observation: order is usually important (duh)

This isn’t exactly a new challenge

Consider the non-exhaustive list at taskqueues.com

- 17 message brokers, including: Apache Kafka, NATS, RabbitMQ and Redis
- 17 queue solutions, including: Celery, Kue, Laravel, Sidekiq, Resque and RQ <- all these use Redis as their backend btw ;)

And that’s without considering protocol-based etc.

So again, why "reinvent hot water"?

Redis (in general and) Streams (in particular) are:

- Everywhere, from the IoT's edge to the cloud
- Blazing fast, massive throughput
- Usable from all(most) languages and platforms

(IoT microcontrollers included)

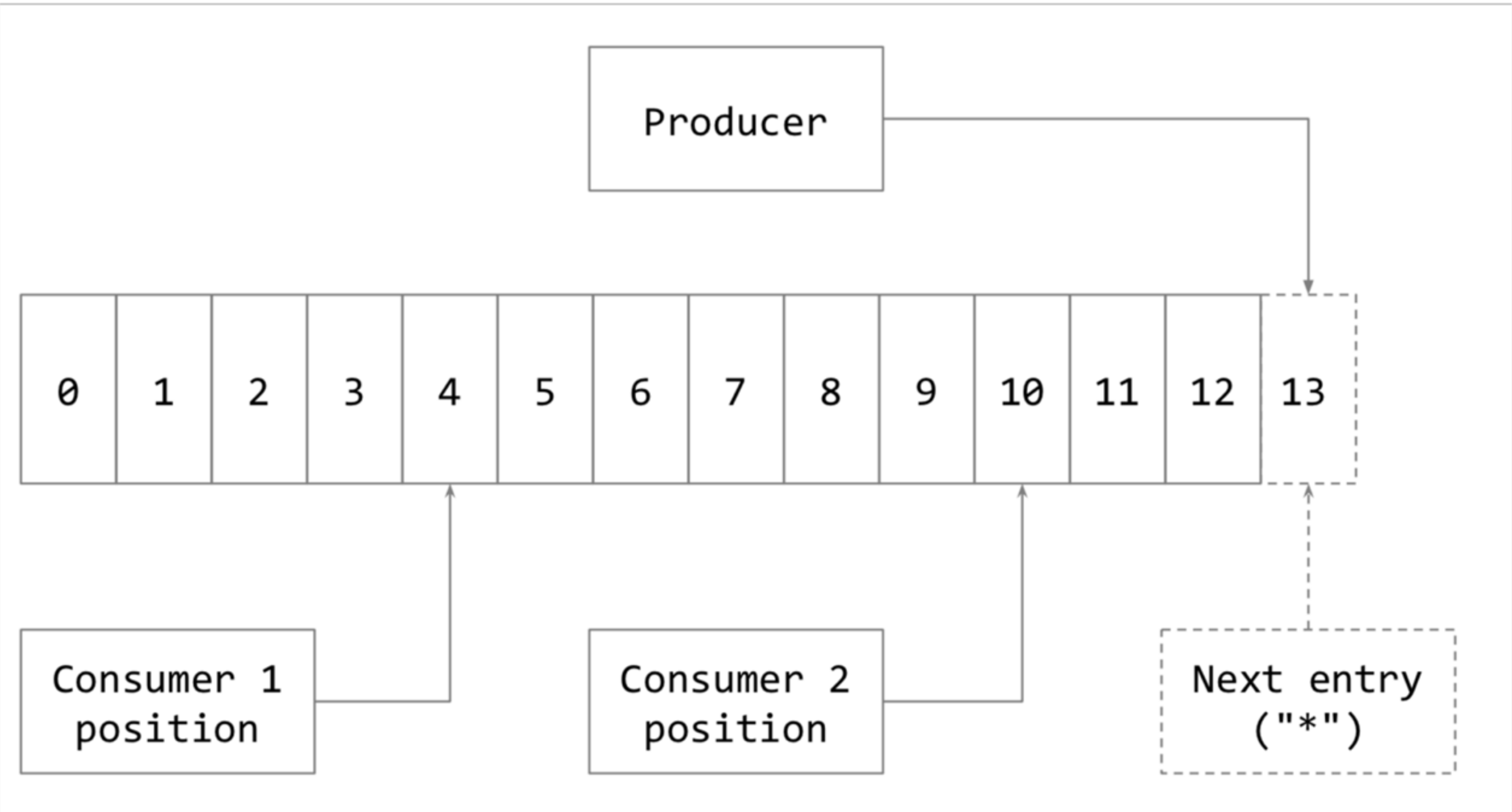
Note: apropos IoT, they are great async buffers

Redis Streams “formalism”

A stream is a sequence of entries (records). It:

- Is "sharded" by key ("topic")
- Has 1+ producers
- Has 0+ consumers
- Can provide *at-most*- or *at-least-once* semantics
- Enables stream processing/real time pipelines (as opposed to batch)

A picture of a stream



Producer

0	1	2	3	4	5	6	7	8	9	10	11	12	13

Consumer 1 position

Consumer 2 position

Next entry ("*")

Entries in the Stream

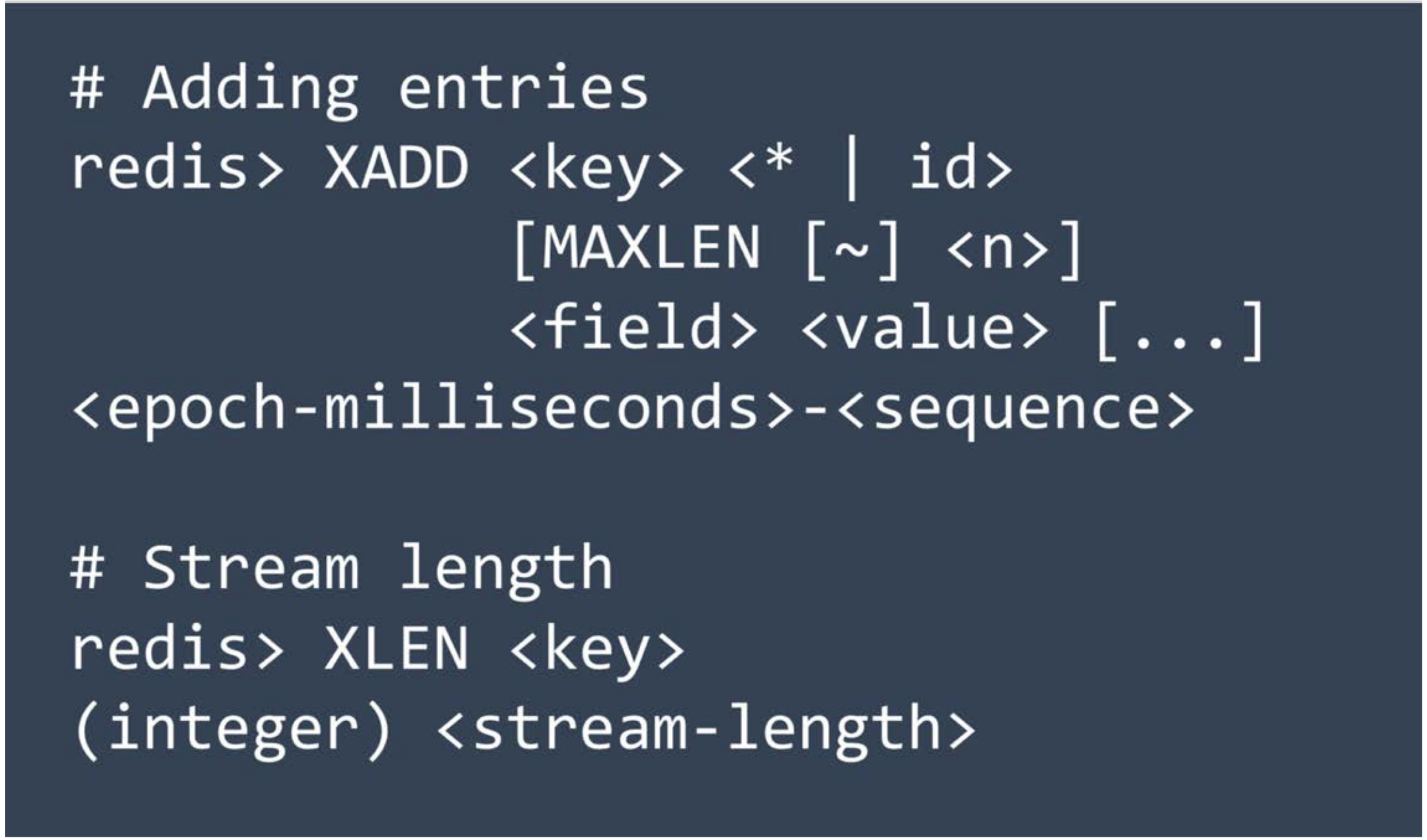
Every entry has a unique ID that is its logical offset. The ID is in following format:

<epoch-milliseconds>-<sequence>

Note: each ID part is a 64-bit unsigned integer

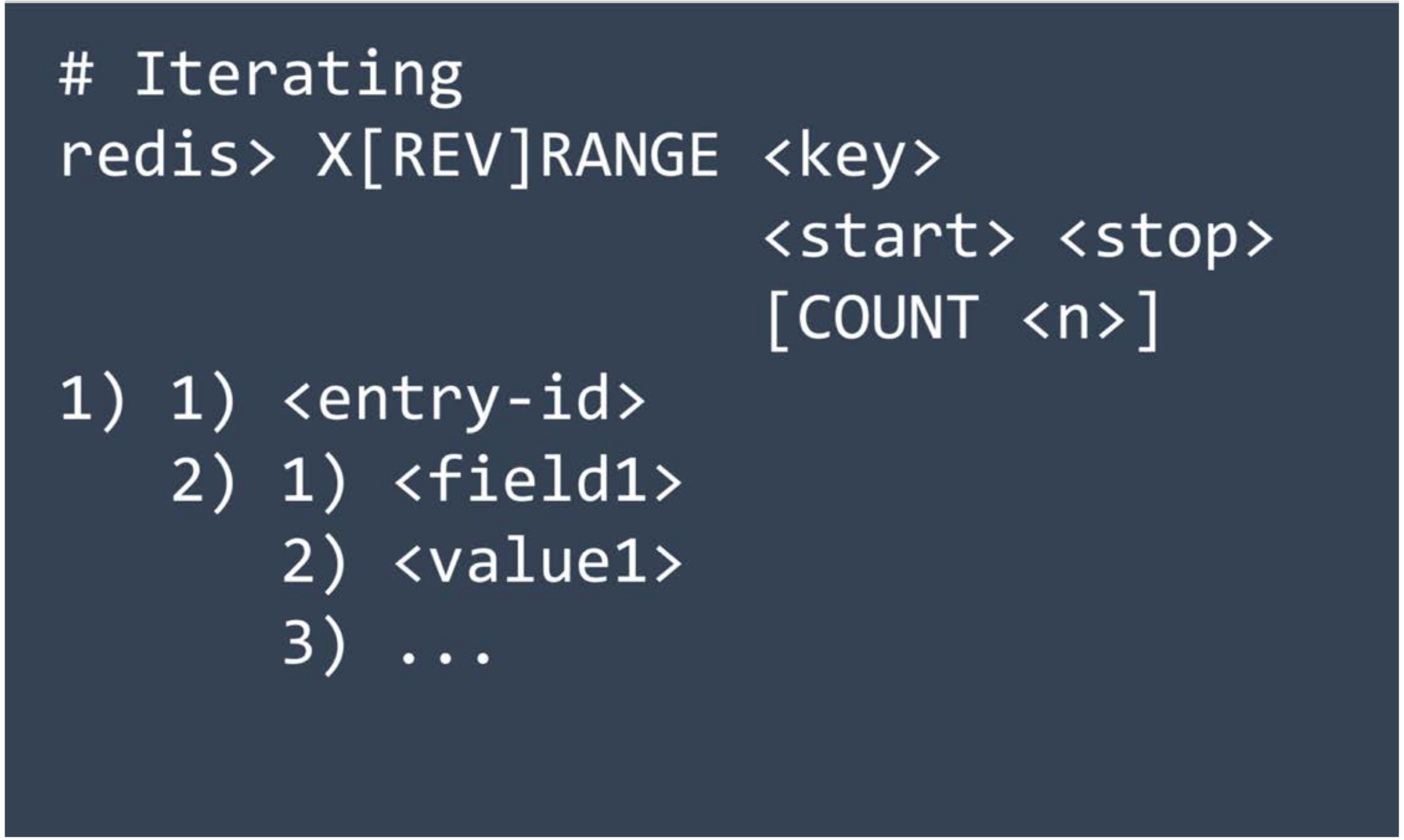
An entry also has one or more ordered field-value pairs, allowing for total abstraction (the empty string is a valid field name, good for time series).

Adding Entries



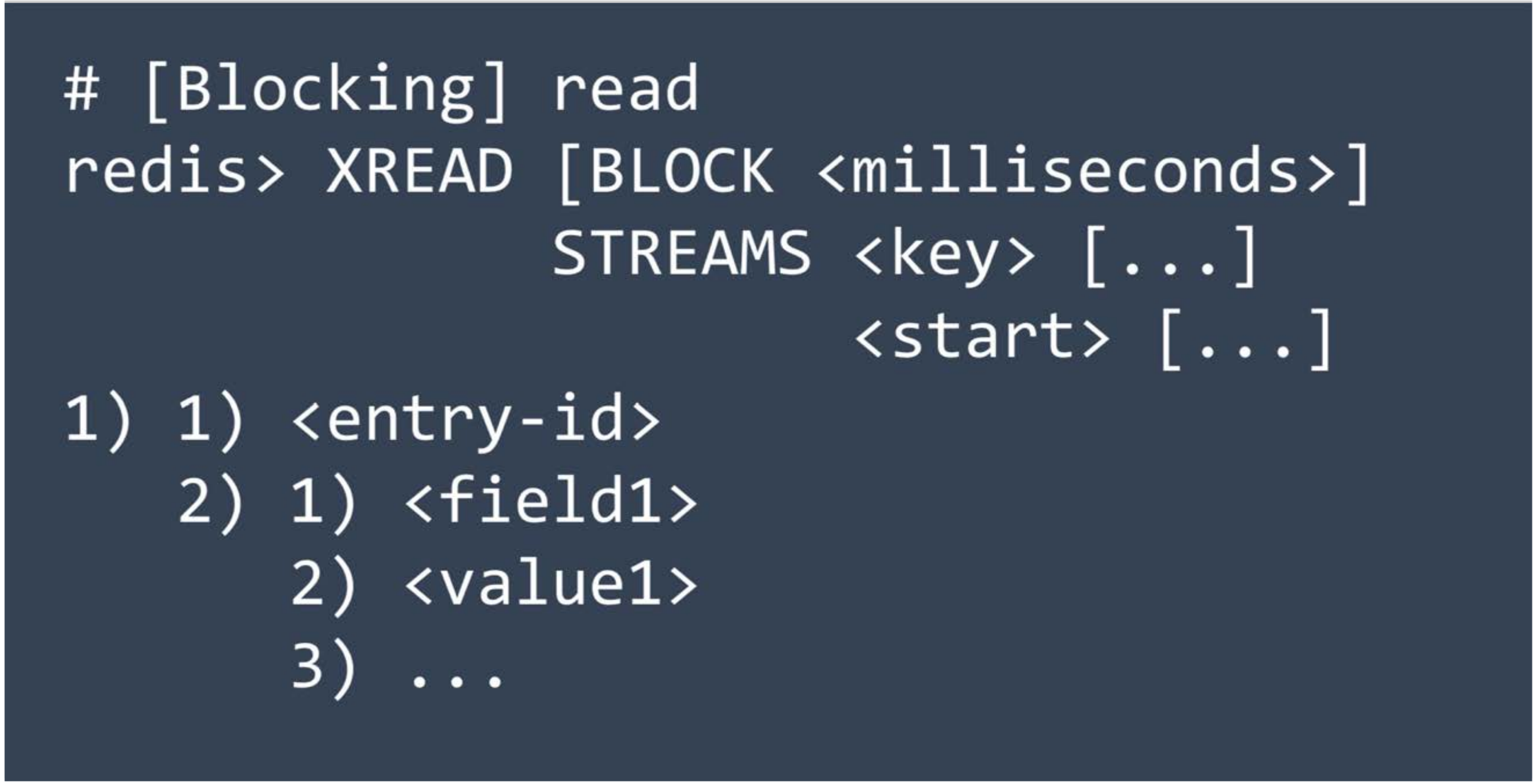
```
# Adding entries
redis> XADD <key> <* | id>
            [MAXLEN [~] <n>]
            <field> <value> [...]
<epoch-milliseconds>-<sequence>

# Stream length
redis> XLEN <key>
(integer) <stream-length>
```

Iterating



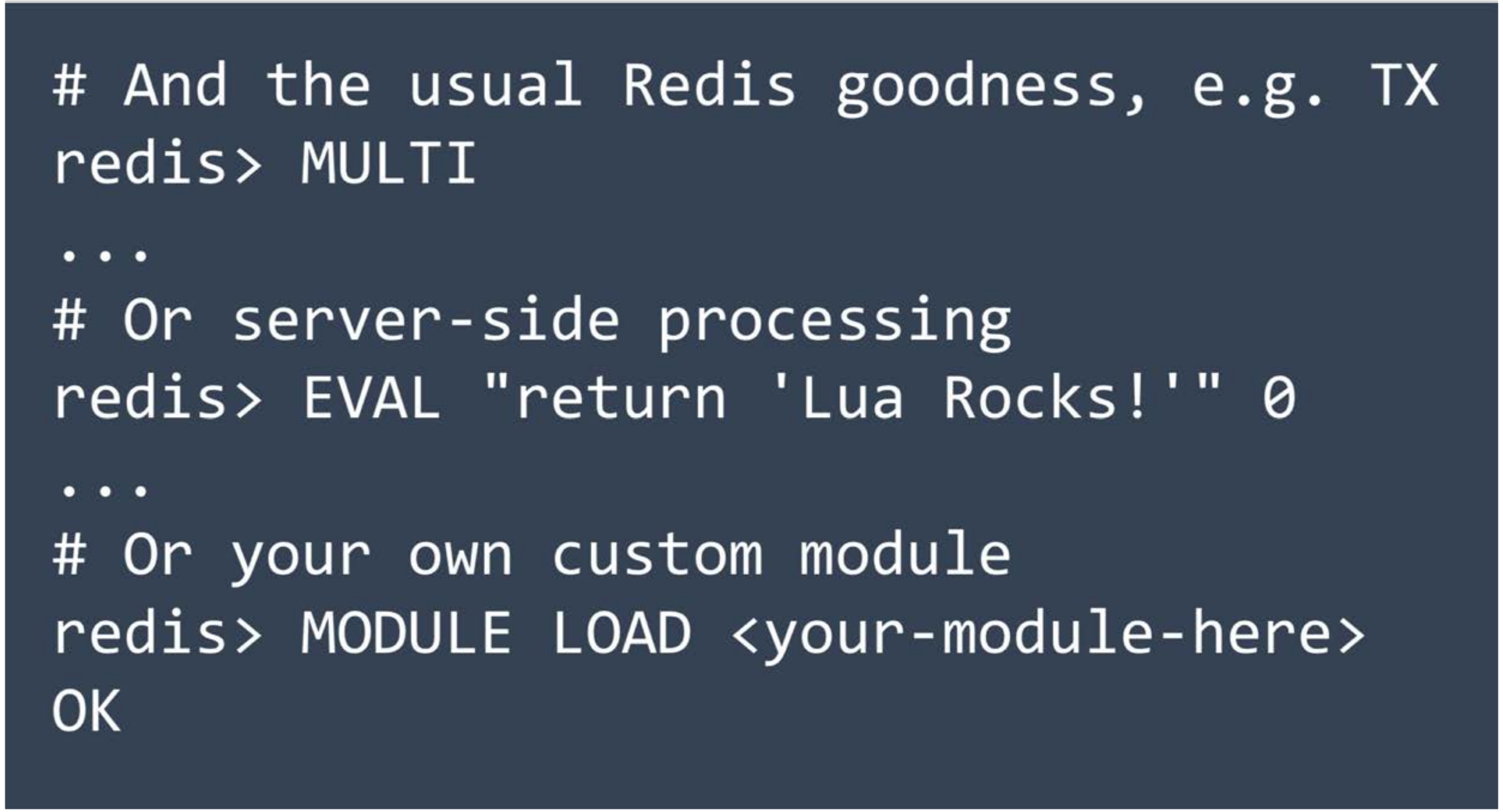
```
# Iterating
redis> X[REV]RANGE <key>
                   <start> <stop>
                   [COUNT <n>]
1) 1) <entry-id>
   2) 1) <field1>
      2) <value1>
      3) ...
```

Blocking Read



```
# [Blocking] read
redis> XREAD [BLOCK <milliseconds>]
             STREAMS <key> [...]
                     <start> [...]
1) 1) <entry-id>
   2) 1) <field1>
      2) <value1>
      3) ...
```

Multi



```
# And the usual Redis goodness, e.g. TX
redis> MULTI
...
# Or server-side processing
redis> EVAL "return 'Lua Rocks!'" 0
...
# Or your own custom module
redis> MODULE LOAD <your-module-here>
OK
```

The problem with scaling consumers

A consumer of a stream gets all entries in order, and will eventually become a bottleneck.

Possible workarounds:

- Add a "type" field to each record - that's dumb
- Shard the stream to multiple keys - meh
- Have the consumer dispatch entries as jobs in queues … GOTO 10

Consumer Groups

” … allow multiple consumers to cooperate in processing messages arriving in a stream, so that each consumers in a given group takes a subset of the messages. “

Shifts the complexity of recovering from consumer failures and group management to the Redis server

Group orientation

We are here :)

- Groups are named and are explicitly (!) created:

  XGROUP CREATE temps agg $

- Consumers are also named, and each gets only a subset of the stream:

  XREAD-GROUP GROUP agg CONSUMER escher-01 STREAMS temps >

- XACK/NOACK in XREAD, XCLAIM, XPENDING

Redis Streams status

- Expected to be GA within a month or so (est. Oct 2018)

Try it yourself

From your browser: https://try.redis.io

Or download it: https://redis.io/download

Or clone it: https://github.com/antirez/redis

Or dockerize it: docker run -it redis

Or try Redis Enterprise by https://redislabs.com

Questions

Dave Nielsen

dave@redislabs.com

@davenielsen