REDIS
EVERYWHERE

HELLO WORLD

- **Ricard Clau**, born and grown up in **Barcelona**
- Software engineer at **Hailo** in **London**
- **Symfony2** lover and **PHP** believer
- **Open-source** contributor, sometimes I give talks
- **Twitter** @ricardclau / **Gmail** ricard.clau@gmail.com

WHAT IS REDIS?

- **RE**mote **DI**ctionary **S**erver
- Created in 2009 by Salvatore Sanfilipo (**@antirez**)
- Open source
- Advanced **in-memory** key-value data-structure server
- Part of the NoSQL movement
- Stop thinking **SQL**, think back to **structures**!

TO ME REDIS IS LIKE...

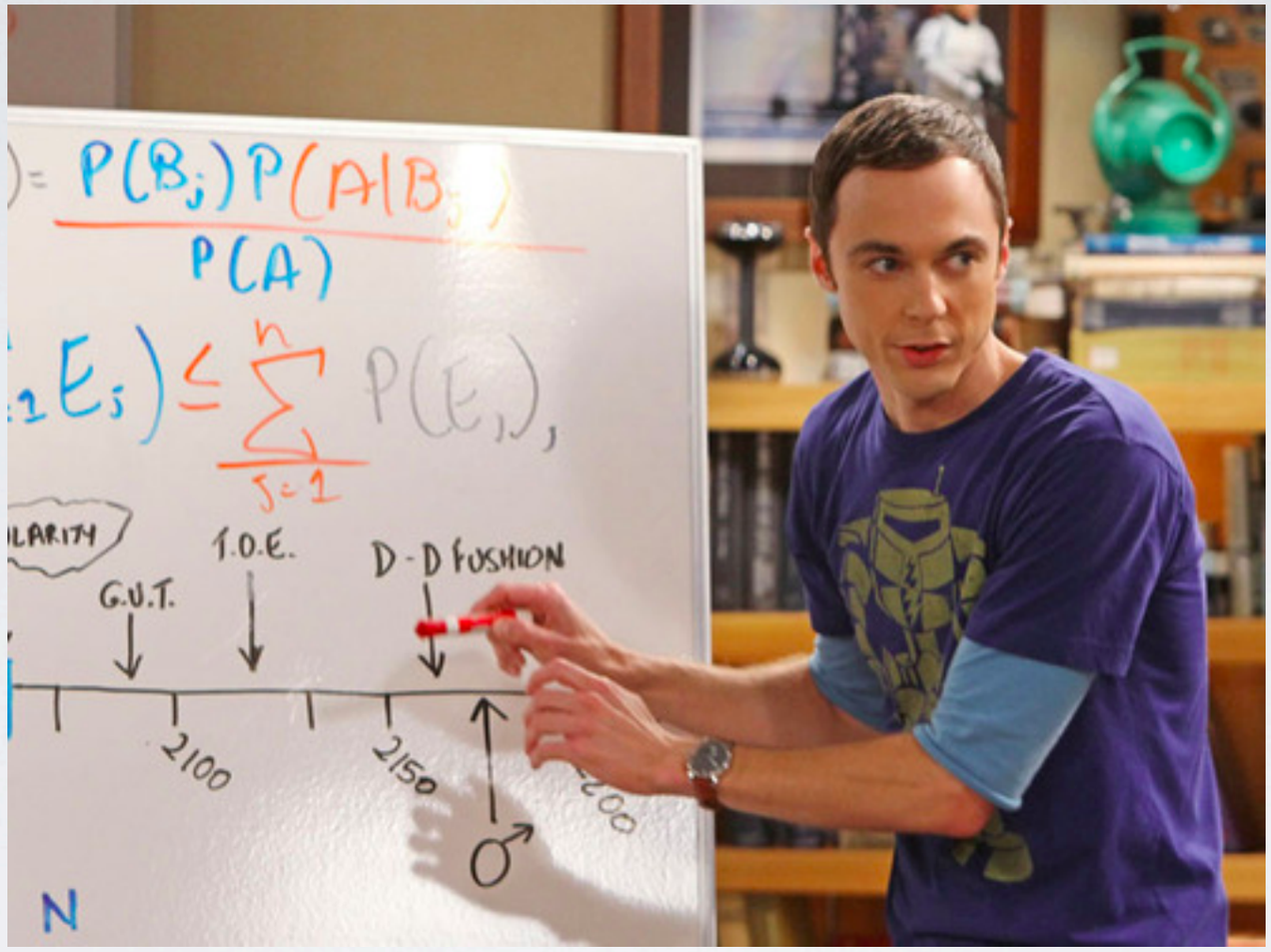
REDIS EVERYWHERE?

Redis is **not** a silver bullet... so let´s rephrase that with...

The **amazing** story on how we gradually introduced **Redis**
in a social videogaming company... and it saved us!

AGENDA

- **REDIS** introduction
- **Getting started** with PHP / Symfony2: sessions, caching
- **Recipes** and comparison with other technologies
- **Sharding** data. And some **criticism**!
- **War Stories** from a 7.5M DAU project
- **Final thoughts**



INTRODUCTION TO REDIS

And some demystifications

DATA TYPES

- **Strings** up to 512 Mb
- **Lists** of elements sorted by insertion order (max 2^32 - 1)
- **Sets** unordered collection of elements supporting unions, intersections and differences (max 2^32 - 1)
- **Hashes** key-value pairs (max 2^32 - 1)
- **Sorted sets** automatically ordered by score (max 2^32 - 1)

ONLY IN-MEMORY?

- Your data needs to **fit in your memory**
- It has **configurable persistence**: RDB snapshots, AOF persistence logs, combine both or no persistence at all
- Think like **memcached on steroids** with persistence
- http://redis.io/topics/persistence
- **“Memory is the new disk, disk is the new tape”**

INTERESTING FEATURES

- Master-slave **replication**
- **Pipelining** to improve performance
- **Transactions** (kind of with MULTI ... EXEC)
- **Publisher** / **Subscriber**
- Scripting with **LUA** (~stored procedures)
- **Predictable** performance (check commands complexity)

HARDWARE TIPS

- Redis is **single-threaded**, so a 2 core machine is enough
- If using EC2, maybe **m1.large** (7.5Gb) is the most suitable, but if it is not enough you might consider any m2 memory optimized type
- **Amazon ElastiCache** finally supports Redis!
- **CPU is rarely the bottleneck** with Redis
- Read carefully the doc to maximize performance!

NEW IN REDIS 2.8

- **CONFIG REWRITE** and **IPv6** support
- Improvements in **scripting** and key **expiring** algorithm
- Keyspace notifications via PUB / SUB. (~Triggered events)
- **Partial resynchronization** with slaves
- Better consistency support (allow writes only when N slaves)
- A fully rewritten **Redis Sentinel**

ALSO IN 2.8 (H|Z|S)?SCAN

- Ability to **iterate** the data structures
- SCAN cursor [MATCH pattern] [COUNT count]
- (H|Z|S)SCAN key cursor [MATCH pattern] [COUNT count]
- **Cursor** is kind of a pointer that we send back and forth
- **Count** is NOT the number of elements but the complexity
- When using MATCH some steps in iteration may return nil

GETTING STARTED WITH PHP

It is easier than you may think!

PHP CLIENTS

- **Predis** (PHP) vs **phpredis** (PHP extension)
- Predis is very mature, actively maintained, feature complete, extendable, composer friendly and supports all Redis versions
- Phpredis is faster, but not backwards compatible
- **Network latency** is the biggest performance killer
- http://redis.io/clients

REDIS-CLI AND MONITOR

- **Redis-cli** to connect to a Redis instance
- Experiment with http://redis.io/commands

- With **Monitor** we can watch live activity!

```
ricard :~ redis-cli
redis 127.0.0.1:6379> monitor
OK
1374953660.710455 [0 127.0.0.1:63598] "SELECT" "1"
1374953660.710633 [1 127.0.0.1:63598] "GET" "foo:ft02r16hdm6kg4t
hkif5dps5k1"
1374953660.737992 [1 127.0.0.1:63598] "SETEX" "foo:ft02r16hdm6kg
4thkif5dps5k1" "120" "_sf2_attributes|a:0:{}_sf2_flashes|a:0:{}_
sf2_meta|a:3:{s:1:\"u\";i:1374953660;s:1:\"c\";i:1374953630;s:1:
\"l\";s:1:\"0\";}"
```

SESSIONS, CACHING

- Super-easy to implement with commercial frameworks
- Commands used: **GET** <session-id>, **SETEX** <session-id> <expires> <serialized-data>
- Fast performance and also persistent!
- Caching and temp-data storage work exactly equal
- Be careful with temp-data storage and memory usage!

SYMFONY2 INTEGRATION

- There must be a bundle for that...
- https://github.com/snc/SncRedisBundle
- Out of the box: **Session** management, **Monolog** handler, **Swiftmailer** spooling, **Doctrine** caching
- Full integration with Symfony2 **profiler**

REDIS EVERYWHERE!

```
snc_redis:
    clients:
        leaderboard:
            type: predis
            alias: leaderboard
            dsn: redis://leaderboard.dsn
        lock:
            type: predis
            alias: lock
            dsn:
                - redis://lock-01.dsn
                - redis://lock-02.dsn
                - redis://lock-03.dsn
                - redis://lock-04.dsn
    session:
        type: predis
        alias: session
        dsn: redis://multi.dsn/0
    stats:
        type: predis
        alias: stats
        dsn: redis://multi.dsn/1
```

COOKBOOK

And comparisons with other technologies

REDIS AS A QUEUE

- Using **Lists** and **LPUSH** / **RPOP** instructions
- We can have different languages accessing data
- We used that to send **OpenGraph** requests to Facebook (REALLY slow API) with Python daemons
- ~80k messages/minute processed with 1 m1.large EC2
- If you need some advanced message queuing features check **RabbitMQ** (and Álvaro Videla´s talk) and others

REAL-TIME ANALYTICS

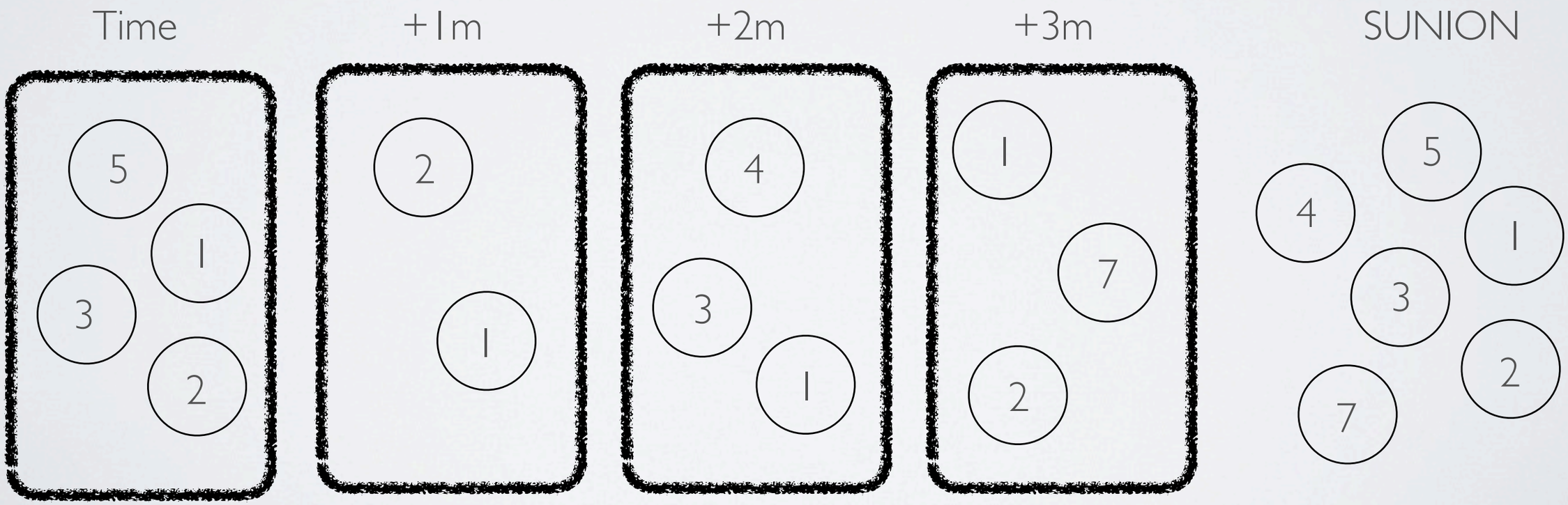
- We can use **hashes** to group many stats under one key
- Most of the times we have counters:
  HINCRBY "stats" <key> <increment>
- **HMGET** "stats" <key1> <key2> ... to retrieve values and
  HMSET "stats" "stat1" <value1> "stat2" <value2> to set them
- **HGETALL** to get the full hash

LEADERBOARDS

- Super-easy with Redis, heavy consuming with other systems
- Each board with a single key, using **sorted sets**
- **ZINCRBY** "rankXXX" <increment> <userId>
- Top N **ZREVRANGE** "rankXXX" 0 N [WITHSCORES]
- We stored **several milions of members** under 1 key

WHO IS ONLINE? (I)

- We can use **SETS** to achieve it!
- One set for every minute, **SADD** <minute> <userId>

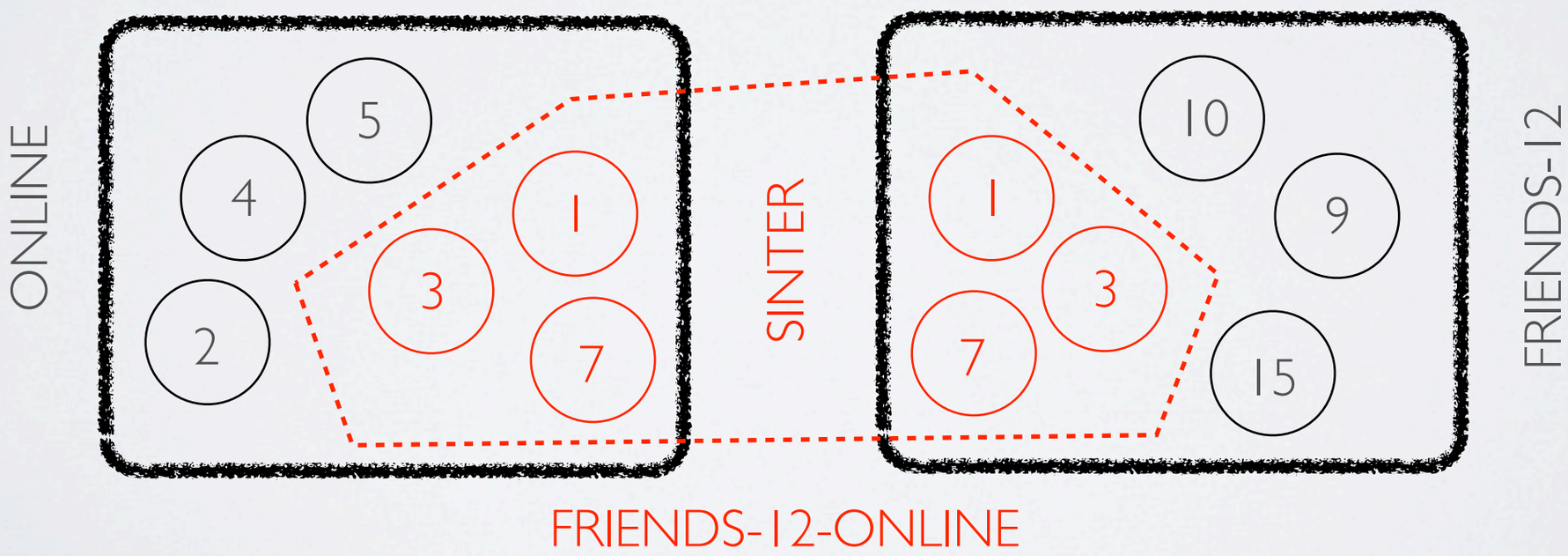


WHO IS ONLINE? (II)

- Online users == everyone active in the last N minutes
- **SUNION** <now> <now-1> <now-2> ...
- **SUNIONSTORE** “online” <now> <now-1> <now-2> ...
- Number of users: **SCARD** “online”
- Obtain online users with **SMEMBERS** “online”

FRIENDS ONLINE?

- If we store user´s friends in a **SET** with key friends-<user-id>
- Friends online: **SINTERSTORE** "friends-<userid>-online" "online" "friends-<userid>"



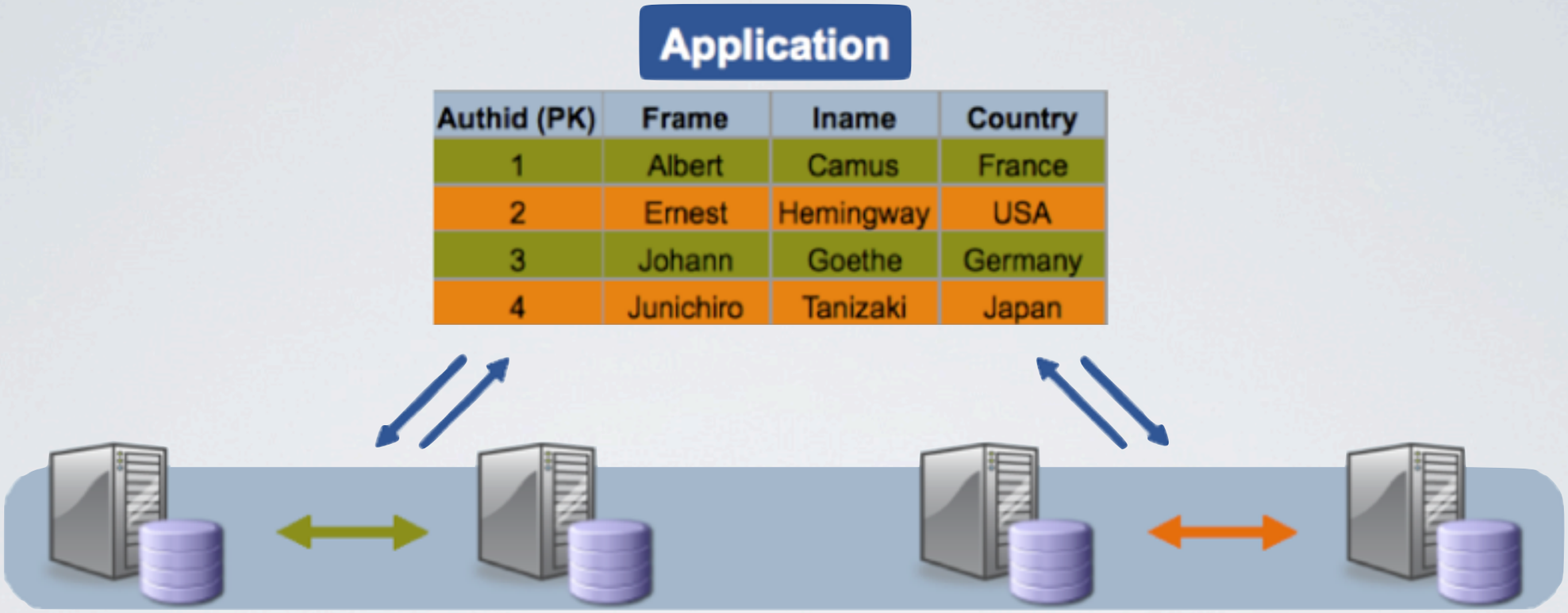
- Imagine how you would do it without Redis!

DISTRIBUTED LOCKING (I)

- Problem in heavy-write applications: 2 almost concurrent requests trying to update same records
- SELECT FOR UPDATE? Easy to create deadlocks!

DISTRIBUTED LOCKING (II)

- Locking can be solved with Redis with Exclusive SET
- **SET <lock> 1 NX EX <seconds>**
- And keep retrying until a specific timeout
- 8 m1.xlarge instances to lock 7.5M Daily Active Users
- Alternatives: Apache **Zookeeper**, perhaps better for scaling or if we need different locking patterns



Authid (PK)	Frame	Iname	Country
1	Albert	Camus	France
3	Johann	Goethe	Germany

Authid (PK)	Frame	Iname	Country
2	Ernest	Hemingway	USA
4	Junichiro	Tanizaki	Japan

SHARDING DATA

Hopefully, you get there at some point
Resharding: a painful experience

NEEDS TO FIT IN MEMORY

- **Redis Cluster** will be available in 3.0 (delayed many times)
- Proxy servers: **Twemproxy** (also for memcached)
- Client-side partitioning (supported in many clients)
- Sharding **strategies** (**range** vs **hash** partitioning)
- **Presharding** can help scaling horizontally
- Memory is cheaper every day but this is a real need!

1-99999

100000-199999

id % 4 == 0

id % 4 == 1

200000-299999

300000-399999

id % 4 == 2

id % 4 == 3

Easy to predict scale
Load not properly distributed

Load distributed ok
Fairly easy to reshard

RANGE PARTITIONING

May work in some cases, not a silver bullet

hash(key) % 4 == 0

hash(key) % 4 == 1

hash(key) % 4 == 2

hash(key) % 4 == 3

HASH

CRC16
CRC32
MD5
SHA1
MURMUR3

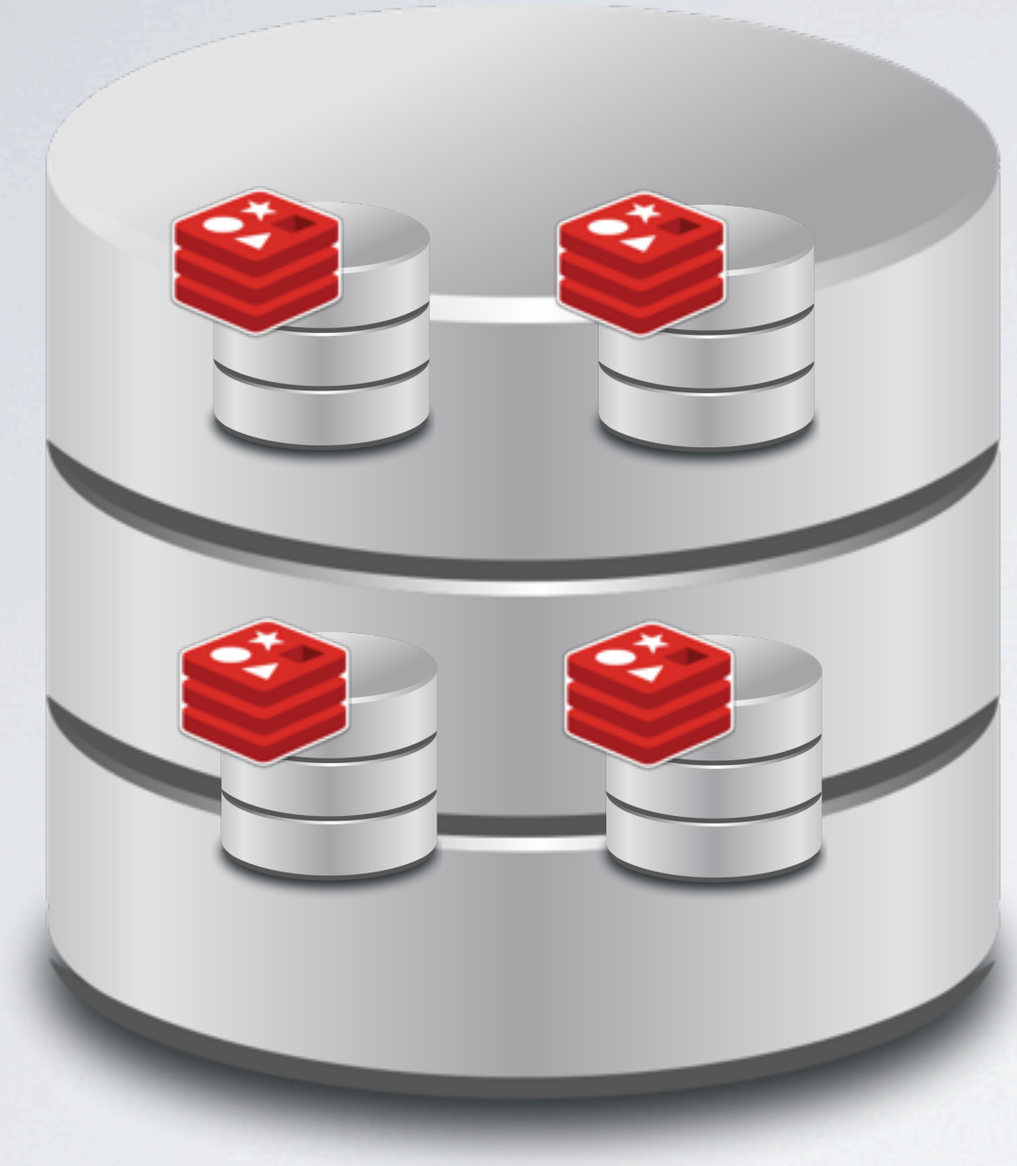
DIST

MODULUS
KETAMA

Distribution also depending on hash algorithm
Complicated resharding

HASH PARTITIONING

Better distribution... but still issues
If few keys, it also becomes useless



Many Redis instances in one initial server

When needed, just split it into more servers and you don´t need resharding!!!

Be careful configuring memory limits

PRESHARDING

Maybe overengineered

CRITICISM IN THE NET

Sometimes unfair... sometimes they have a point

VS OTHER DATABASES

- **Cassandra** or **RIAK** reshard automatically (at a big cost)
- Complicated to fully automate **failovers** (Sentinel problems)
- Severe issues when slaves get out of sync, getting better
- Tricky to inspect big sets and hashes prior to 2.8
- All NoSQL have issues, check **aphyr.com** "Call me maybe"

ABOUT REDIS CLUSTER

- **CAP** Theorem: **C**onsistency, **A**vailability, **P**artition tolerance
- Purists blame it to not properly respect the principles
- CAP theorem is... a theorem and it comes with a **cost**
- The goal is to provide some consistency and some failure resistance without sacrificing other practical qualities
- We could **debate** for hours around that point!

WAR STORIES

https://apps.facebook.com/dragoncity/

(~7.5M DAU Facebook Game)

PROGRESSIVE USAGE

- DragonCity had **~2M DAU** when we introduced Redis
- First usages: **Queues** PHP-Python and **Temporary** storage
- Introduced **Locking** with Redis -> End of deadlocks
- Used for temporary contests involving **rankings**
- **Session tracking, cheat detection** and many others
- About **7.5M DAU** and similar amount of servers

REDIS IS SUPER-FAST

- Your **network gets killed** much before Redis is down
- Always increase your **ulimit** values in Redis servers
- We had Gigabit network connectivity between machines but it was not enough to handle the data sent and retrieved
- Redis was just at 20% of CPU
- **Do some heavy-load tests** before launching!

SURPRISES WITH MEMORY

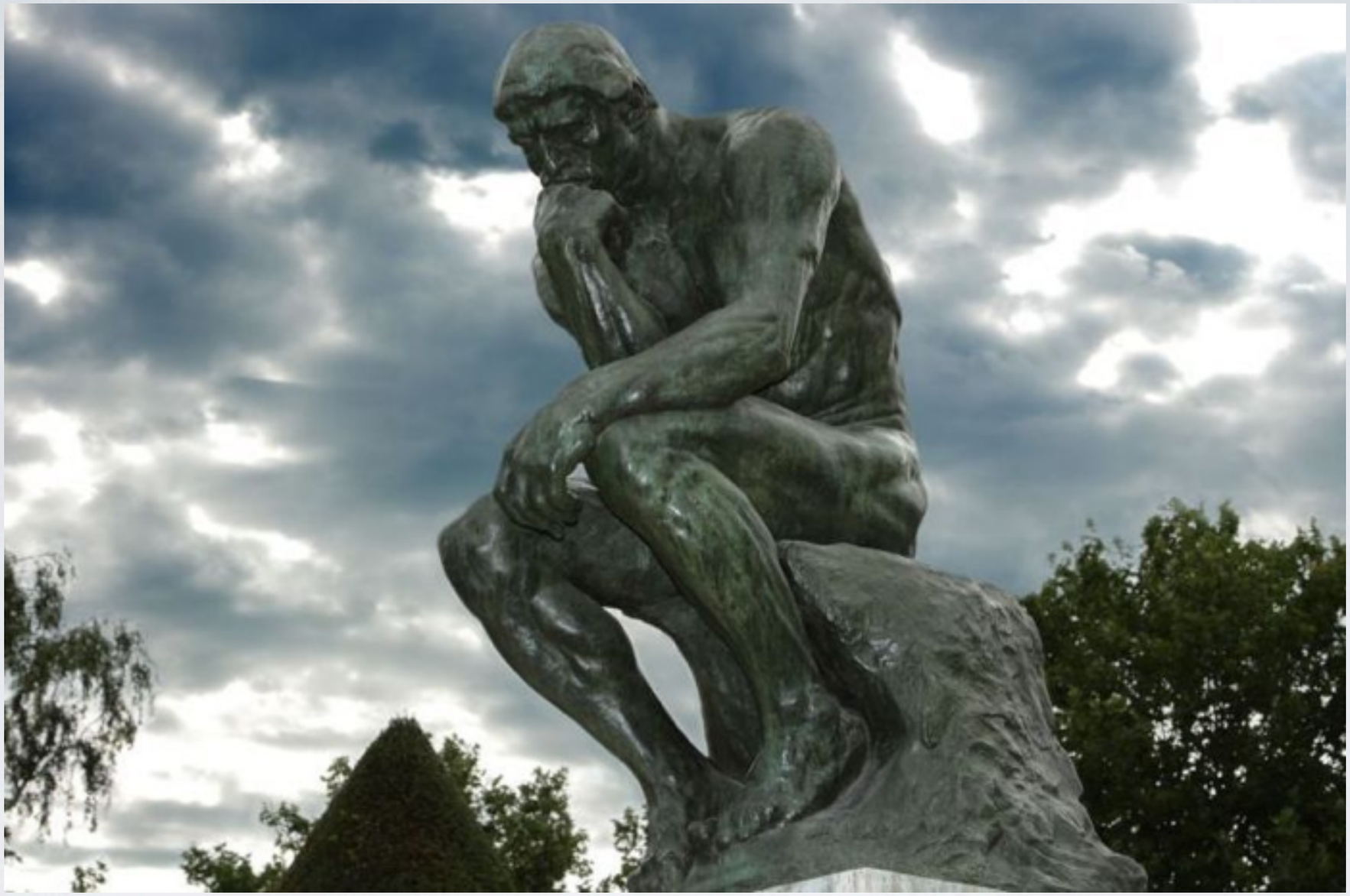
- Redis documentation: "**Sets of binary-safe strings**" so, we changed values in sets from being just numbers to number + :<char> to add some info to values

- We expected ~20% memory increase but in was 500%

- https://github.com/antirez/redis/blob/unstable/src/t_set.c#L55

- If value can be represented as long, it is stored more efficiently

- http://alonso-vidales.blogspot.co.uk/2013/06/not-all-redis-values-are-strings.html



FINAL THOUGHTS

When and when not to consider Redis

REDIS IS PERFECT FOR...

- Intensive **read-write** data applications
- **Temporary** stored data
- Data that **fits in memory**
- Problems that fit **Redis built-in data types**
- **Predictability** in performance needed (all commands complexity documented)

BUT IS NOT SUITABLE WHEN..

- Big data sets, archive data
- Relational data (**RDBMS are absolutely fine** to scale)
- We don´t know how we will **access data**
- **Reporting** applications (no where clauses)
- It gets tricky for distributed applications (replication, clustering)
- ALWAYS **choose the right tool for the job**!

SPECIAL THANKS

- Ronny López (@ronnylt) & Alonso Vidales (@alonsovidales)
- All backend and systems engineers at @socialpoint
- Of course, to all of you for being here today!
- Check our open-source project **Redtrine**, PR welcome: https://github.com/redtrine/redtrine

QUESTIONS?

- **Twitter:** @ricardclau
- **E-mail:** ricard.clau@gmail.com
- **Github:** https://github.com/ricardclau
- **Blog** about PHP and Symfony2: http://www.ricardclau.com
- Please, rate this talk at https://joind.in/talk/view/10531