redis

devLINK Technical Conference
www.devlink.net

Need Some Cache?

Redis In Depth

Chris Meadows
Architect & Developer
Terenine Technology Solutions
Email: meadoch1@gmail.com
Twitter: @meadoch1

CHATTANOOGA - AUG 17-19, 2011

Agenda

What’s Redis?
How do I make it work?
What’s under the hood?
Q&A

Credit Where Due

Peter Cooper – http://coder.io or peterc (Twitter)

http://redis.io - one of the best OSS documentation sites

What is Redis?

“an…advanced key-value store…” - from redis.io

(aka a NoSQL data store)

NoSQL?

A loose category of non-relational data storage technologies

Other examples
Memcached, MongoDB, RavenDB
CouchDB, Riak, Cassandra

Memcache

Used (largely) as a network based hash/dictionary

Became popular largely because it is simple to use and
fast

Step it up!

<u>Add</u>

Configurable disk persistence options

Multiple native data structure options

Native support for replication

Native support for pub/sub

…more

…and you get

Redis

Reasons to Consider Redis

- Functionally
  - Speed (more on this later)
  - Flexibility
  - Low hardware demand
  - Disk persistence guarantees (if configured)
  - Low development overhead
- “Politically”
  - Price (free – BSD licensing)
  - Actively developed
  - Good community (list, IRC & wiki)
  - Used broadly (GitHub, Twitter, StackOverflow…)
  - Solid Corporate backing by VMWare

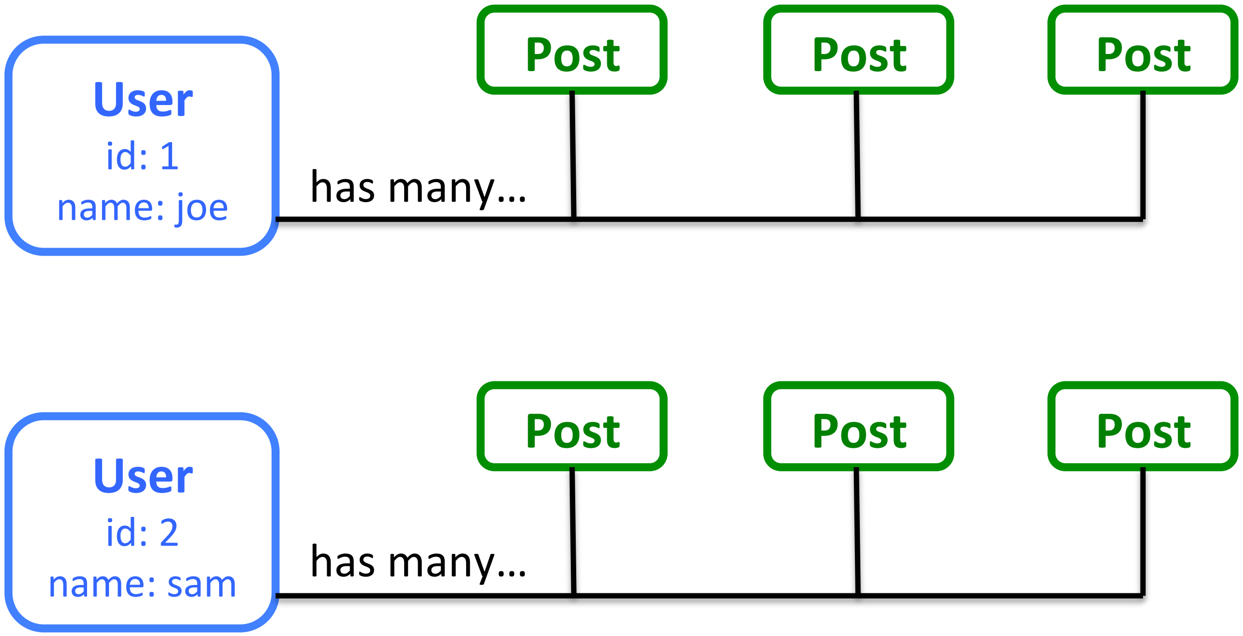
And now for something completely different…

Example Application Usage

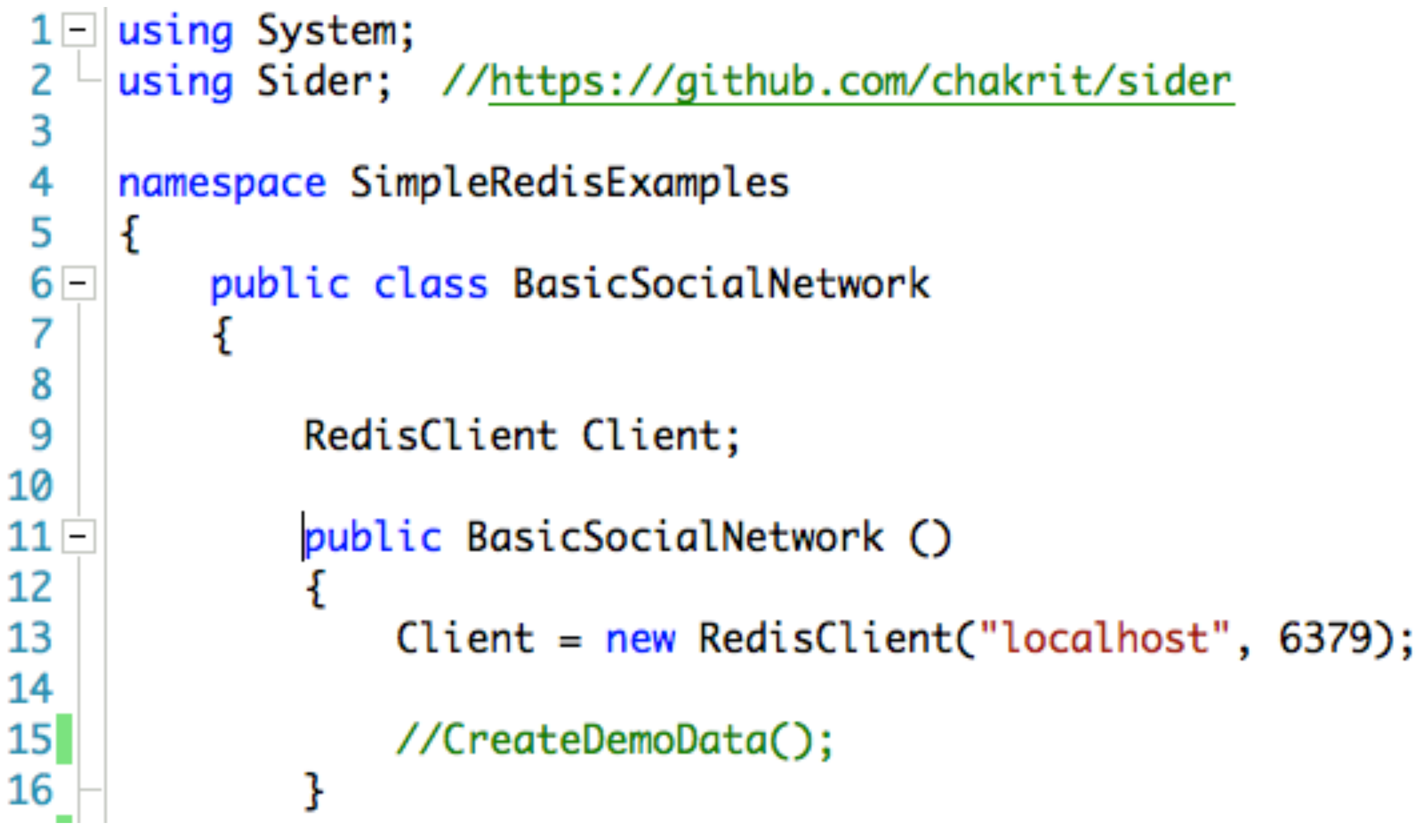
Social Network Storage

Users
have names, can follow others, and be followed

Posts
are things like messages, photos, etc


User
id: 1
name: joe
has many...
Post
Post
Post
User
id: 2
name: sam
has many...
Post
Post
Post

Example Application: Foundation



```
using System;
using Sider;  //https://github.com/chakrit/sider

namespace SimpleRedisExamples
{
    public class BasicSocialNetwork
    {

        RedisClient Client;

        public BasicSocialNetwork ()
        {
            Client = new RedisClient("localhost", 6379);

            //CreateDemoData();
        }
```

Now back to your
regularly scheduled program

How do I make it work?

Data Structure Types

Type	Key	Value
String	login.html TotalLogins	“<html><head>…” 3949
List	user:1:messages	[1, 2, 3, 4]
Set	user:1:contacts	{100, 107, 304, 238}
Hash	user:1:settings	visibility -> contacts only name -> john
Sorted Set	userActivity	1 -> 20 2 -> 34 3 -> 102

Strings

Interaction from command line

```
       client utility            key          value
             ↓                    ↓             ↓
./redis-cli SET “mykey” 12345
                                 key
                                  ↓               returns value
./redis-cli GET “mykey” → 12345
```

Strings

Work on strings which represent integers

GET	SETNX	INCR
SET	MSETNX	DECR
MGET	GETBIT	INCRBY
MSET	SETBIT	DECRBY
SETRANGE	GETSET	STRLEN
GETRANGE	SETEX	

Unique IDs

INCR next_post_id

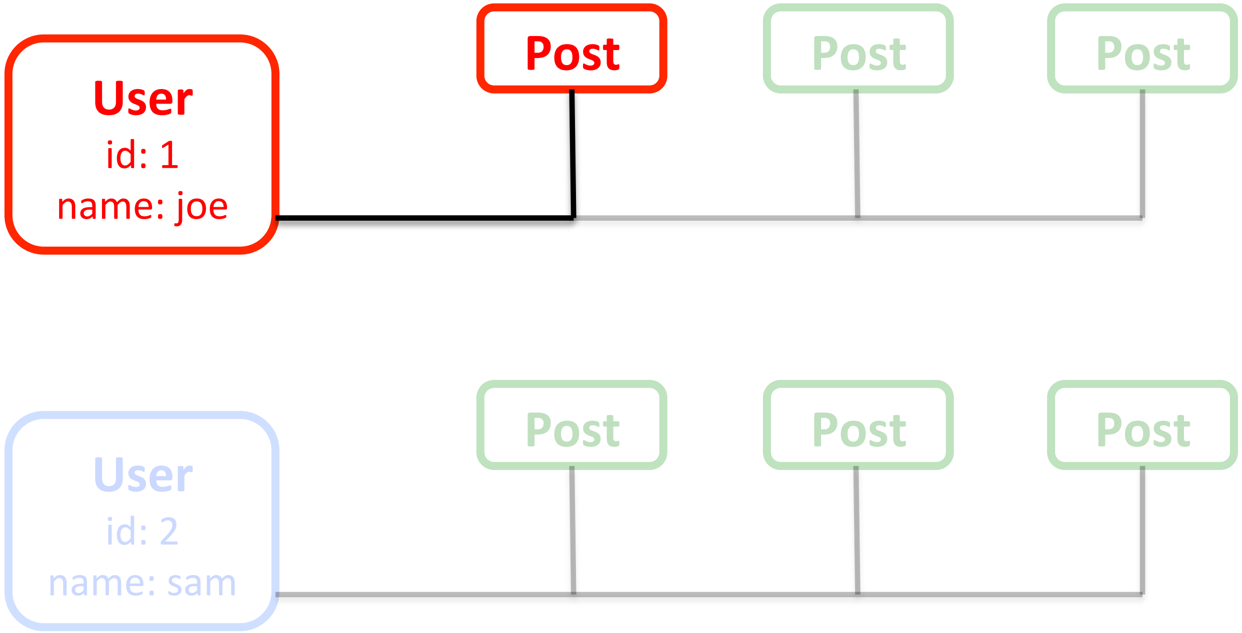
If next_post_id doesn't exist or doesn't contain a number, it will be set to 0, incremented, and 1 will be returned.

returns $\longrightarrow$ 1

INCR next_post_id

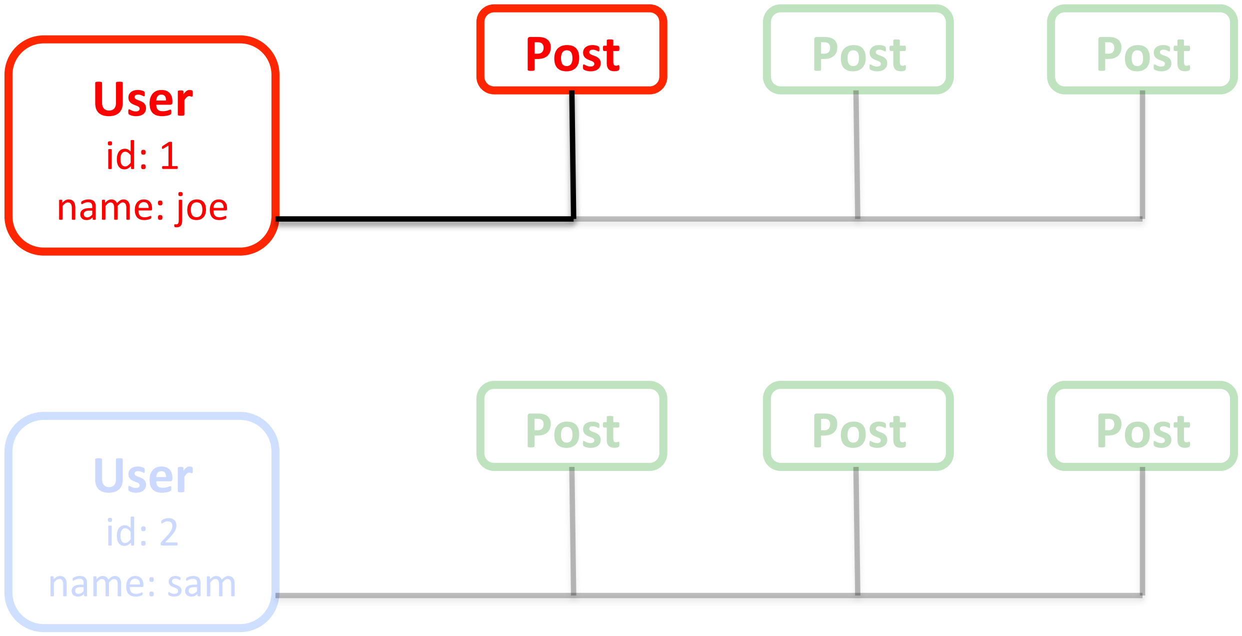
returns $\longrightarrow$ 2

Example Application



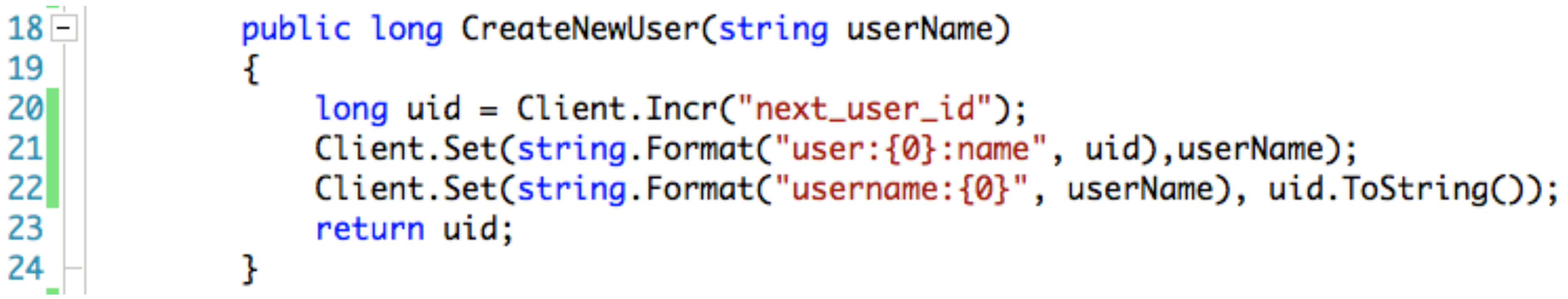
user:1:name →	joe
username:joe →	1
post:1:content →	hello world
post:1:user →	1

So we can do a two way lookup



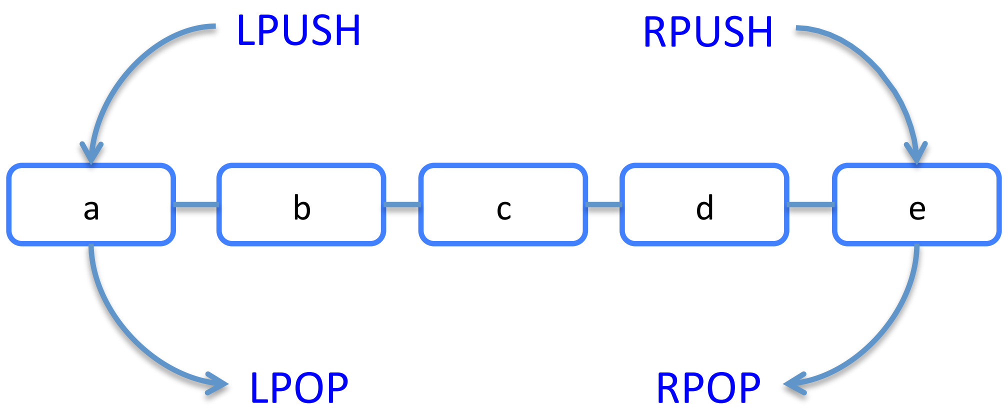
```
SET user:1:name joe
SET username:joe 1

SET post:1:content “hello world”
SET post:1:user 1
```

Example App: Strings



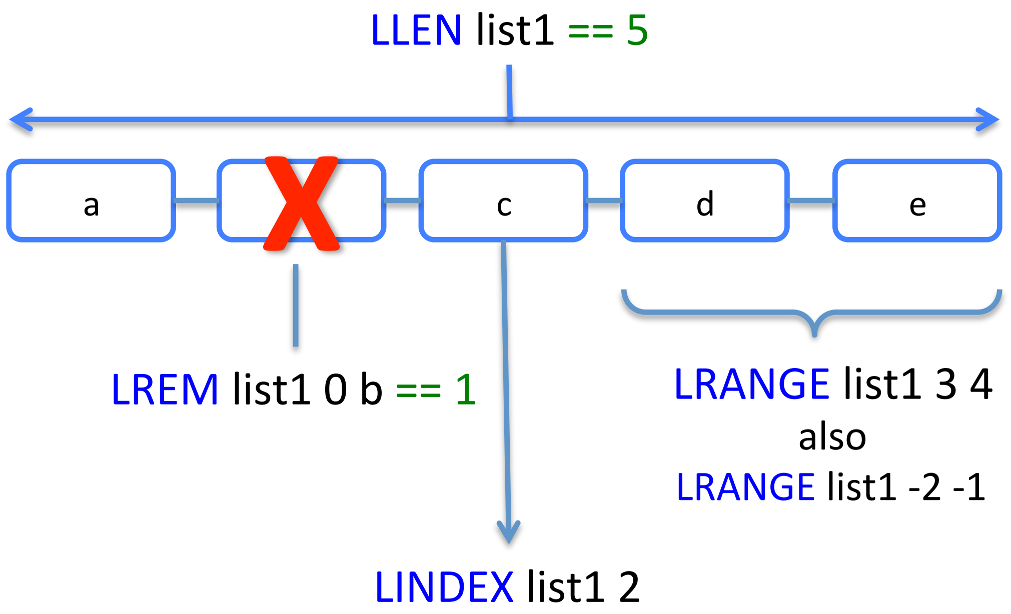
```
public long CreateNewUser(string userName)
{
    long uid = Client.Incr("next_user_id");
    Client.Set(string.Format("user:{0}:name", uid),userName);
    Client.Set(string.Format("username:{0}", userName), uid.ToString());
    return uid;
}
```

Lists

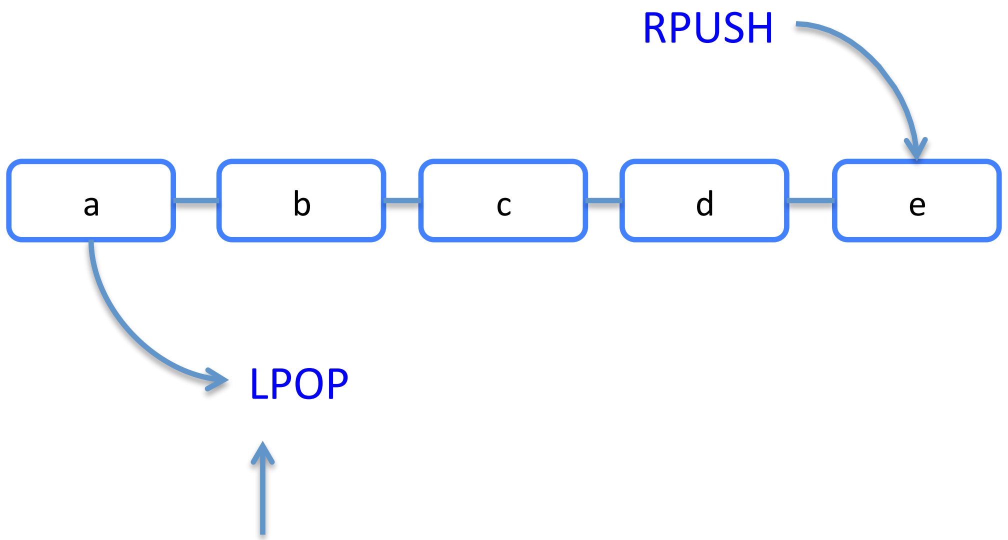


```
RPUSH list1 a -> 1
RPUSH list 1 b -> 2
LPOP list1 -> “a”
RPUSH list1 c -> 2
LPOP list1 -> “b”
LPOP list1 -> “c”
LPOP list1 -> (nil)
```

Lists

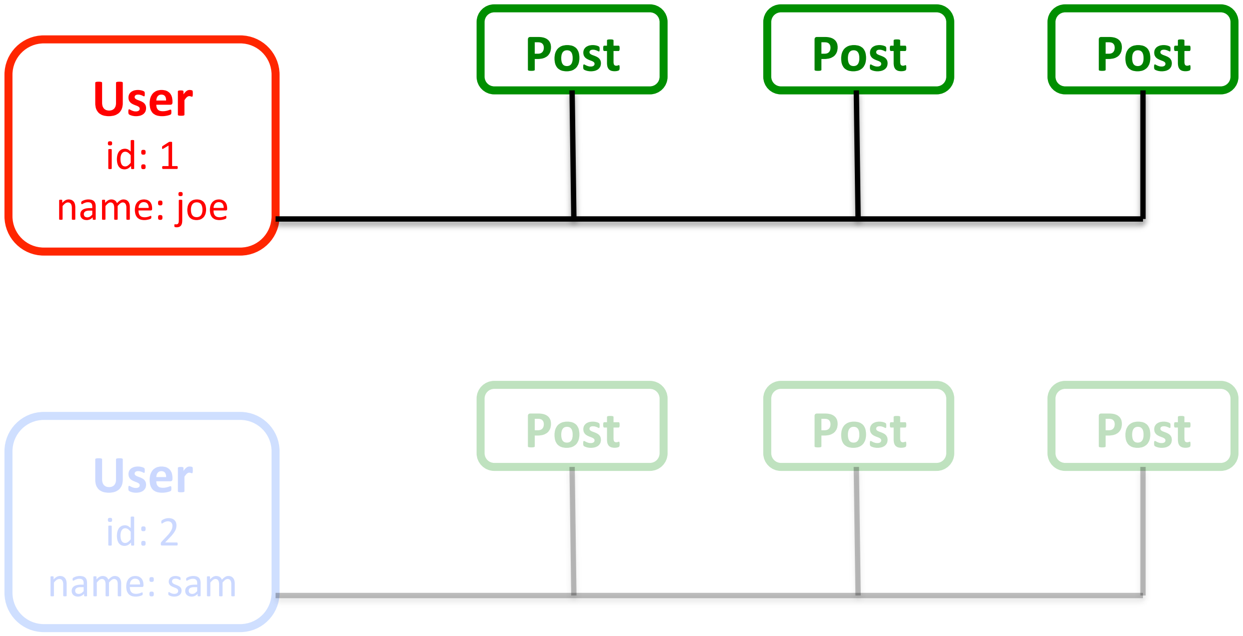

LLEN list1 == 5
a
X
c
d
e
LREM list1 0 b == 1
LRANGE list1 3 4
also
LRANGE list1 -2 -1
LINDEX list1 2

List Use Case
Queue

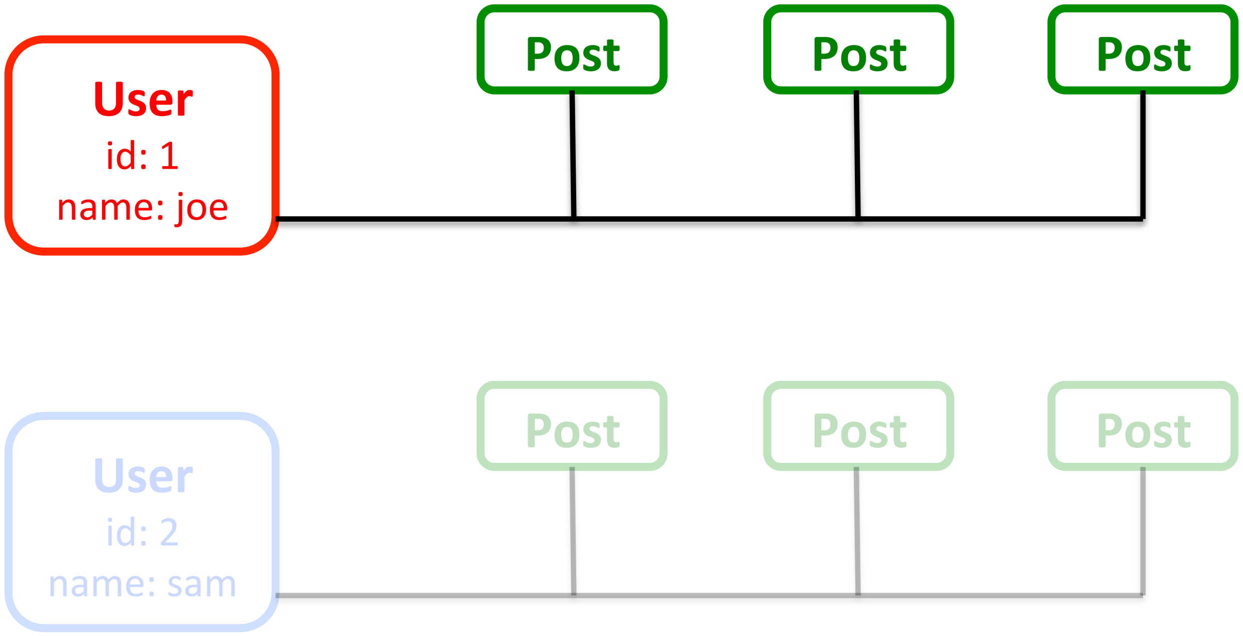


Or BLPOP to block (wait) until something can be popped

```
RPUSH the_q a
…
RPUSH the_q e
LPOP the_q
LPOP the_q
LPOP the_q
```



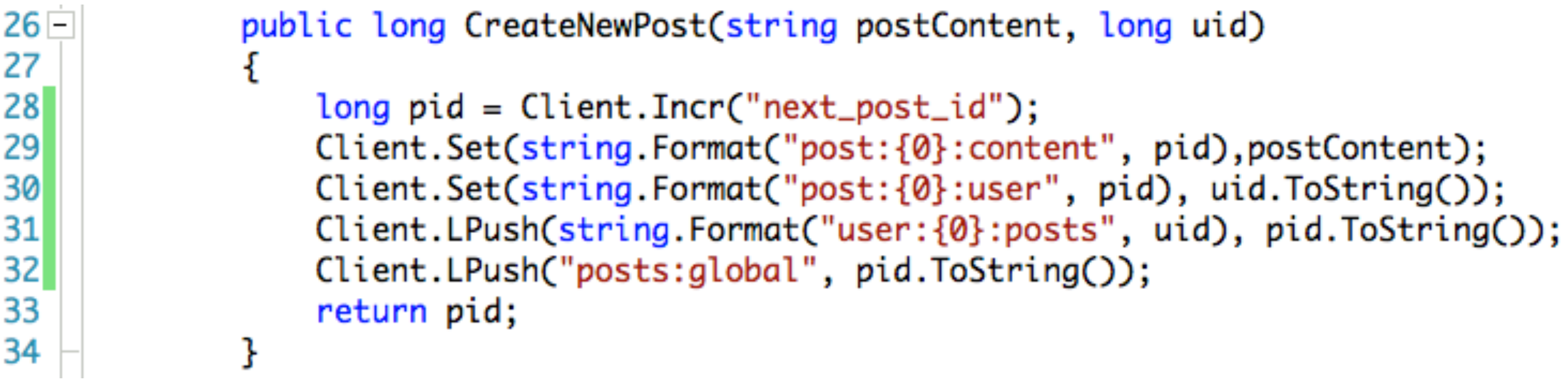
user:1:posts $\longrightarrow$ [3, 2, 1]



user:1:posts $\longrightarrow$ [3, 2, 1]

```
LPUSH user:1:posts 1
LPUSH user:1:posts 2
LPUSH user:1:posts 3
```

Example App: List



```
public long CreateNewPost(string postContent, long uid)
{
    long pid = Client.Incr("next_post_id");
    Client.Set(string.Format("post:{0}:content", pid),postContent);
    Client.Set(string.Format("post:{0}:user", pid), uid.ToString());
    Client.LPush(string.Format("user:{0}:posts", uid), pid.ToString());
    Client.LPush("posts:global", pid.ToString());
    return pid;
}
```

Sets

Holds an unordered group of items
accessed by a string key

contains:aba	abacus cabal baba hello teabag base cabaret database
contains:ase	vase decease baseline case database phase

Sets

Holds an unordered group of items
accessed by a string key

contains:aba

abacus cabal baba hello teabag
base cabaret database

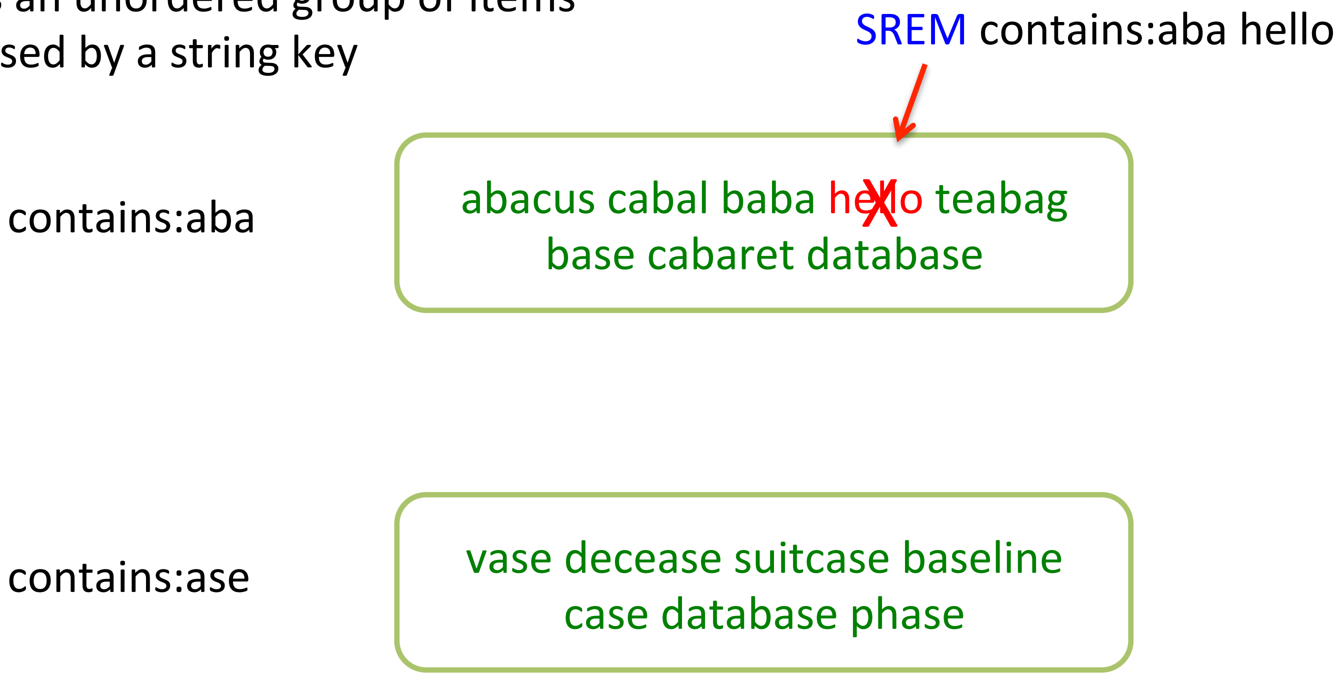
contains:ase

vase decease baseline case
database phase

SADD contains:ase suitcase

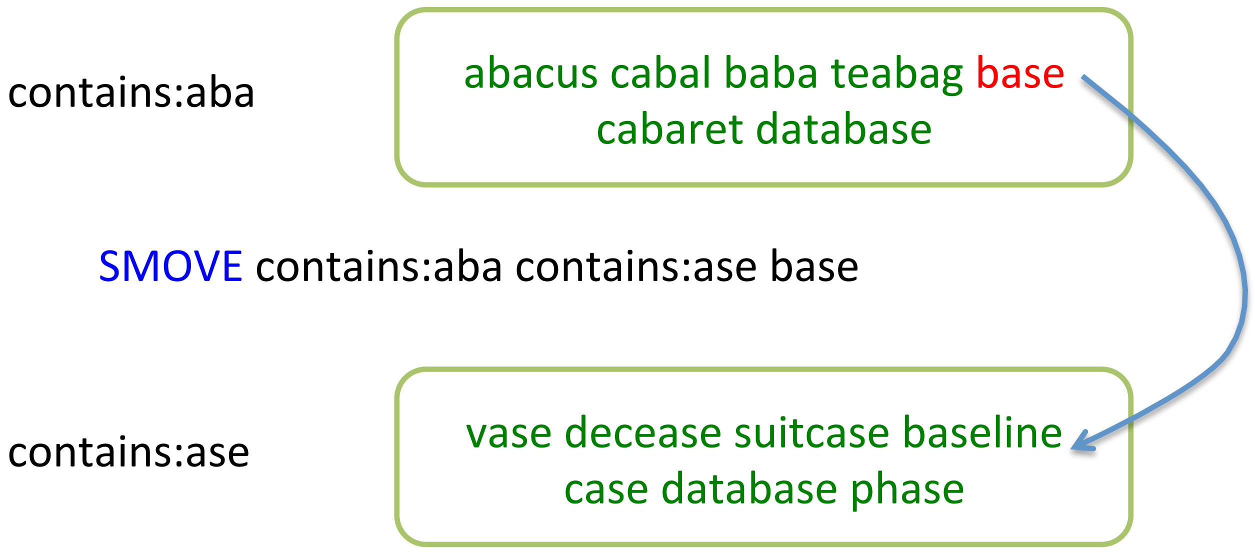
Sets

Holds an unordered group of items
accessed by a string key



Sets

Holds an unordered group of items
accessed by a string key



SMOVE contains:aba contains:ase base

Sets

contains:aba

abacus cabal baba teabag cabaret database

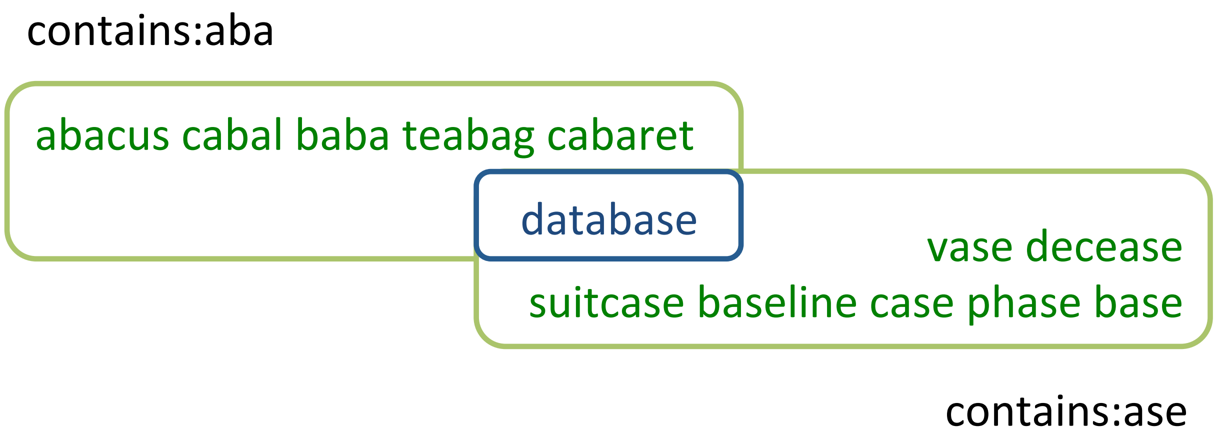
SCARD contains:aba	== 6
SISMEMBER contains:aba chips	== 0 (meaning false)
SRANDMEMBER contains:aba	== “teabag”

contains:ase

vase decease suitcase baseline case database phase base

SMEMBERS contains:ase	== vase, decease, suitcase, baseline, case, database phase, base

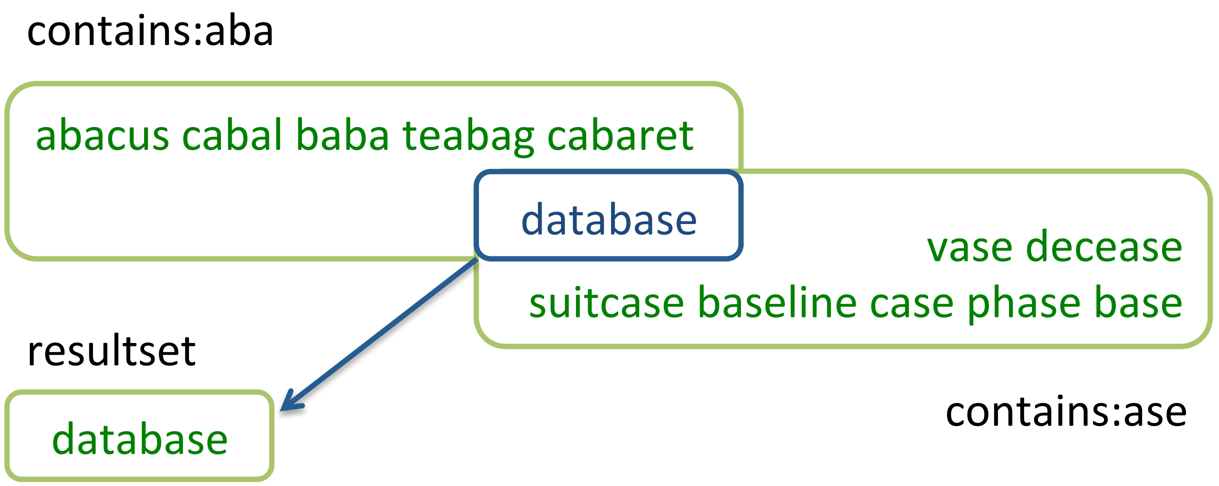
Sets


contains:aba
abacus cabal baba teabag cabaret
database
vase decease
suitcase baseline case phase base
contains:ase

SINTER contains:aba contains:ase == database

This is a very simple example. SINTER can take any number of arguments.
SUNION will join the sets together (not repeating duplicates).
SDIFF will return elements not in common between sets.

Sets


contains:aba
abacus cabal baba teabag cabaret
database
vase decease
suitcase baseline case phase base
contains:ase
resultset
database

SINTERSTORE resultset contains:aba contains:ase

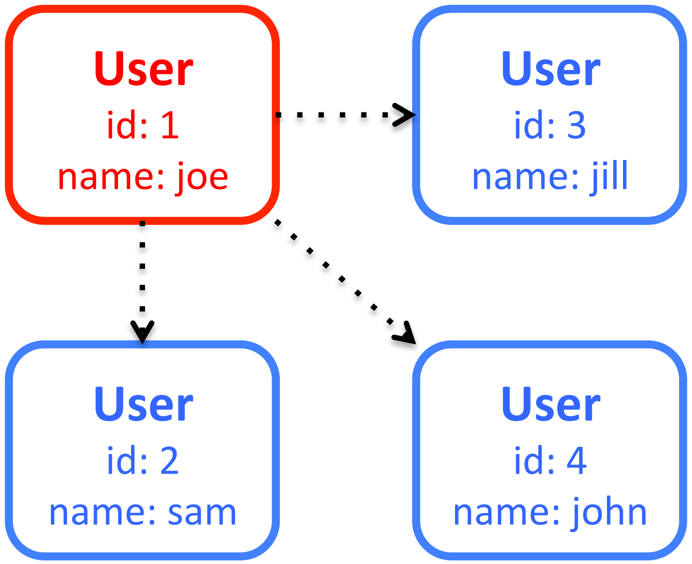
SUNIONSTORE does the same for set unions.
SDIFFSTORE does the same for set diffs.

Set Use Cases

Tags

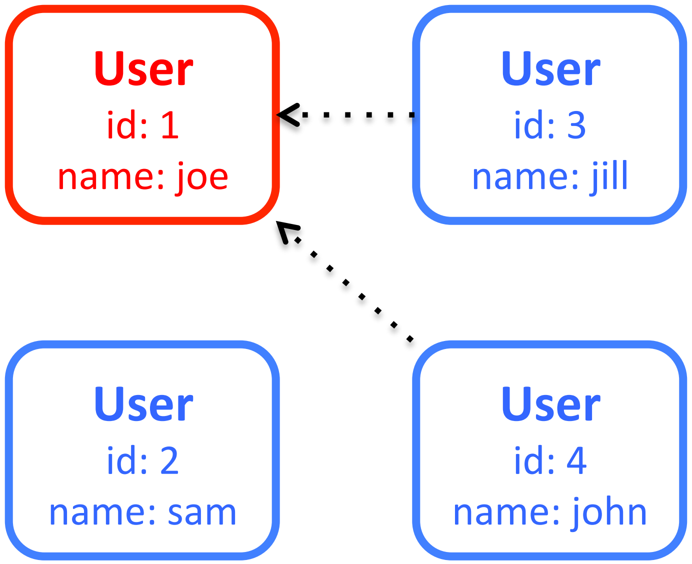
Attributes

Property Bag



user:1:follows $\longrightarrow$ {2, 3, 4}

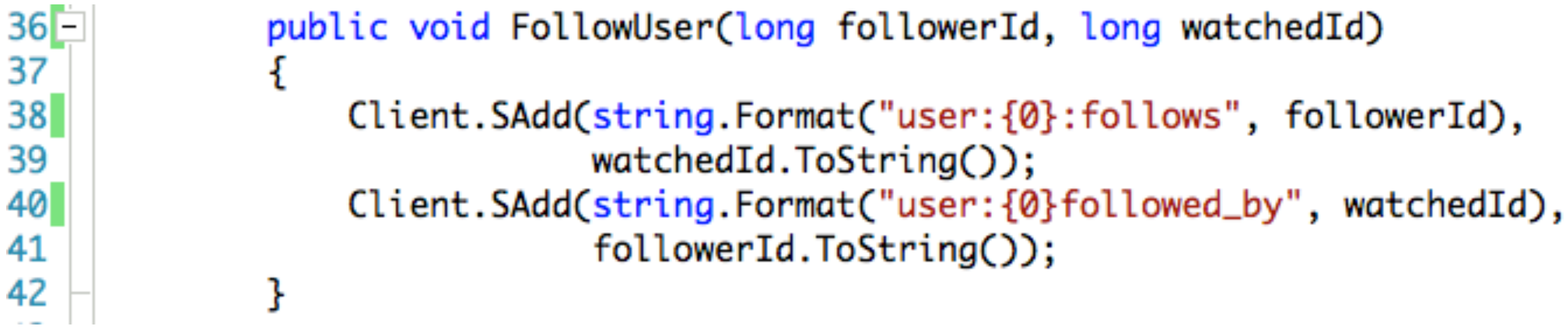
```
SADD user:1:follows 2
SADD user:1:follows 3
SADD user:1:follows 4
```



user:1:followed_by $\longrightarrow$ {2, 3}

```
SADD user:1:followed_by 2
SADD user:1:followed_by 3
```

Example App: Set



```
public void FollowUser(long followerId, long watchedId)
{
    Client.SAdd(string.Format("user:{0}:follows", followerId),
                watchedId.ToString());
    Client.SAdd(string.Format("user:{0}followed_by", watchedId),
                followerId.ToString());
}
```

Sorted Sets

Sorry – not enough time!

Work similarly to sets but each element has a “rank” or “score” associated with it to be used for sort order.

Hashes

product:1

Key	Value
created_at	102374657
product_id	1
name	Yo-yos
available	10

```
HSET product:1 created_at 102374657
HSET product:1 product_id 1
HSET product:1 name “Yo-yos”
HSET product:1 available 10
```

```
HGET product:1 name        ==    Yo-yos
HLEN product:1             ==    4
HKEYS product:1            ==    created_at, product_id,
                                 name, available
HGETALL product:1          ==    created_at => 102374657
                                 product_id => 1
                                 …
```

And more…

HVALS HEXISTS HINCRBY HMGET HMSET

Hash Use Case

Session Storage

Session	**8d3e4**
created_at	102374657
user_id	1

Is essentially a hash

```
HSET session:8d3e4 created_at 102374657
HSET session:8d3e4 user_id 1
```

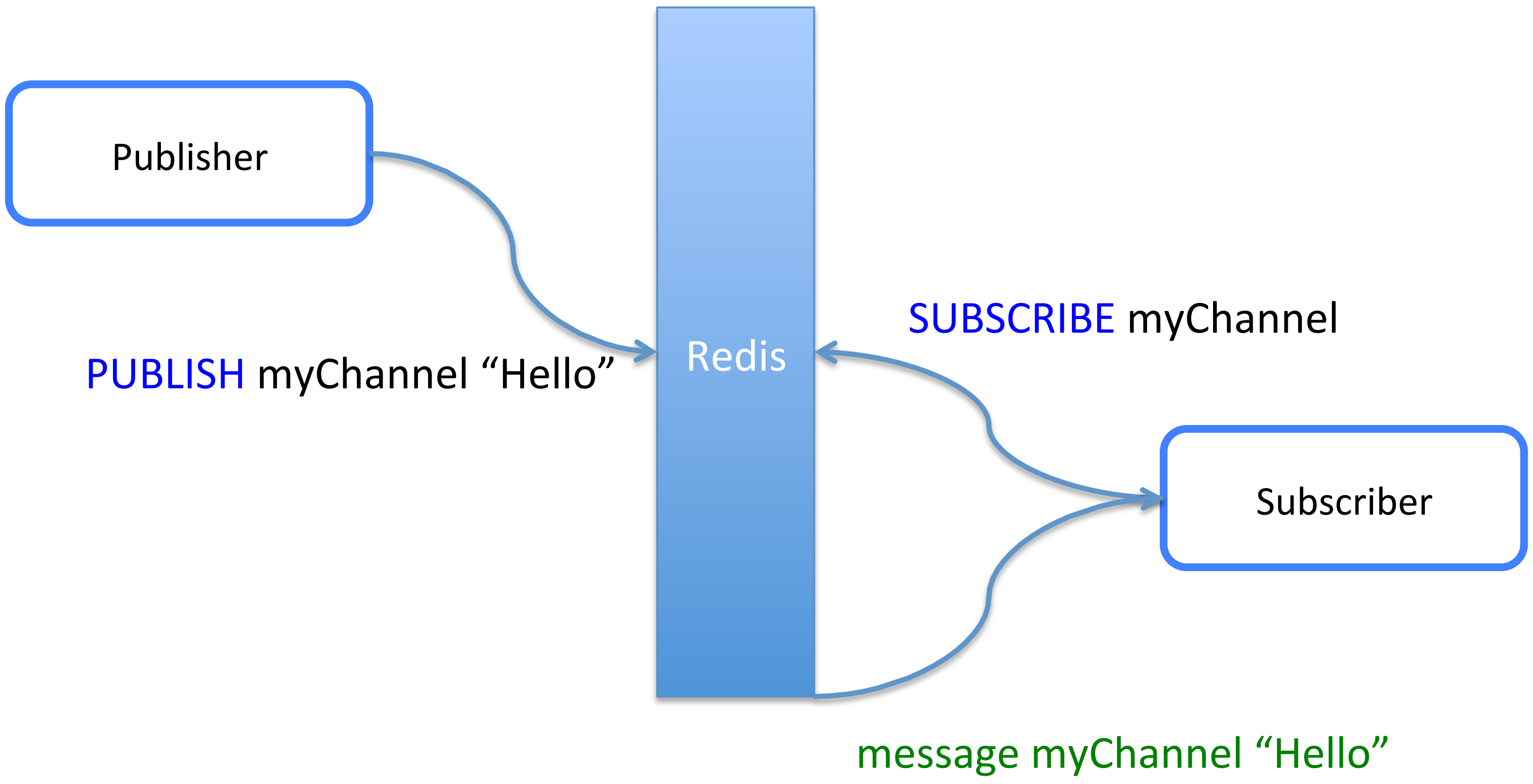
OR

```
HMSET session:8d3e4 created_at 102374657 session:8d3e4 user_id 1
```

For session time out let Redis automatically expire it in 24 hours

```
EXPIRE session:8d3e4 86400
```

Publish/Subscribe



Also PSUBSCRIBE supports wildcards

Enough Commands?

Redis also has commands for
Key Manipulation
Transactions
Connection Settings
Server Management

124 commands in all
(and counting)

What’s Under the Hood?

Written in ANSI C (~20k lines)

Runs on most POSIX systems

Direct binary access for most languages

Event loop processing model

Installation

- Runs on POSIX systems
  - i.e. Linux, Unix, Solaris, OSX
  - Windows: Cygwin or fork of source
    - Not officially supported
- Options to get it running
  - Install from source (uses Make)
  - Install using package manager
    - Apt-get, homebrew and other packages available
- Once installed
  - Redis-server
  - Redis-cli

Configuration Options

General server settings – port, timeouts, log levels, etc
Snapshotting – when to save to disk, where
Replication – master address, password
Security – password (optional), command renaming
Limits – clients, memory
Append only mode – yes/no, fsync behavior
Slow log – parameters
Virtual Memory – yes/no, settings
Memory optimization settings – structure compression
  parameters

Performance

Highly dependent on hardware, configuration, and operation complexity

Also impacted by client library and language

For basic ops (GET, SET, LPUSH, RPOP, HSET, HGET, etc)

Low end – 2000-5000 ops/sec
High end – 120,000+ ops/sec

Performance Example 1

On this machine
MacBook Pro i7 with 8Gb memory

Redis benchmark (3 byte packets)
MSET 36900.37 rps (99.89% <= 2ms)
SET 54644.81 rps (98.78% <= 1ms)
GET 56179.77 rps (99.24% <= 1ms)
INCR 54054.05 rps (98.47% <=1ms)
LPUSH 55248.62 rps (99.93% <= 2ms)
LPOP 56497.18 rps (99.77% <= 1ms)
SADD 54945.05 rps (99.83% <= 2ms)
SPOP 55555.55 rps (100% <= 1ms)

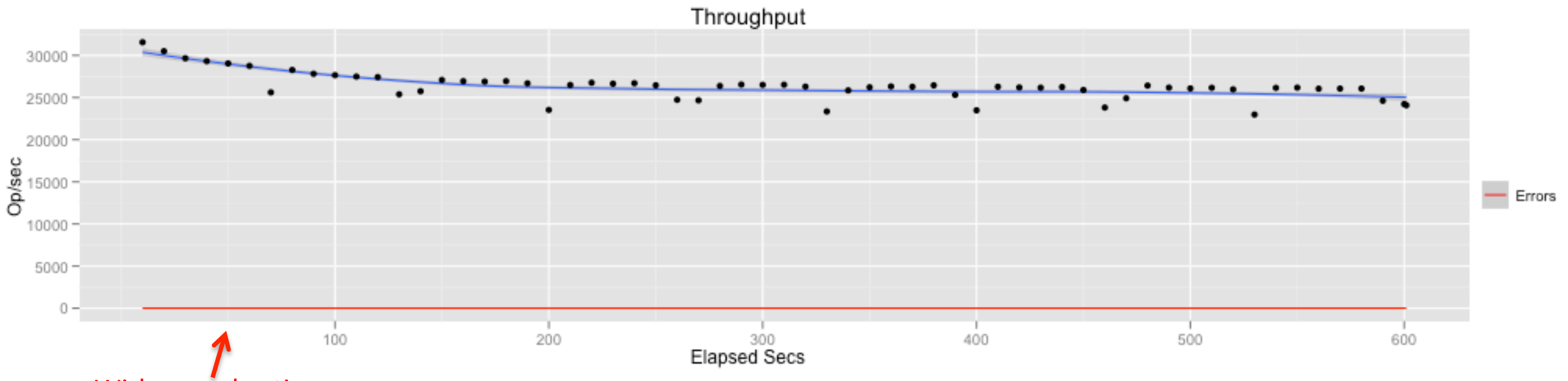
Avg 53k
rps

Performance Example 2

On this machine
MacBook Pro i7 with 8Gb memory

Erlang client using Basho Bench
30 clients, 30000 keys
Simultaneous (random) k/v, hash, & list access
½ single integer payload, ½ 2705 byte string payload

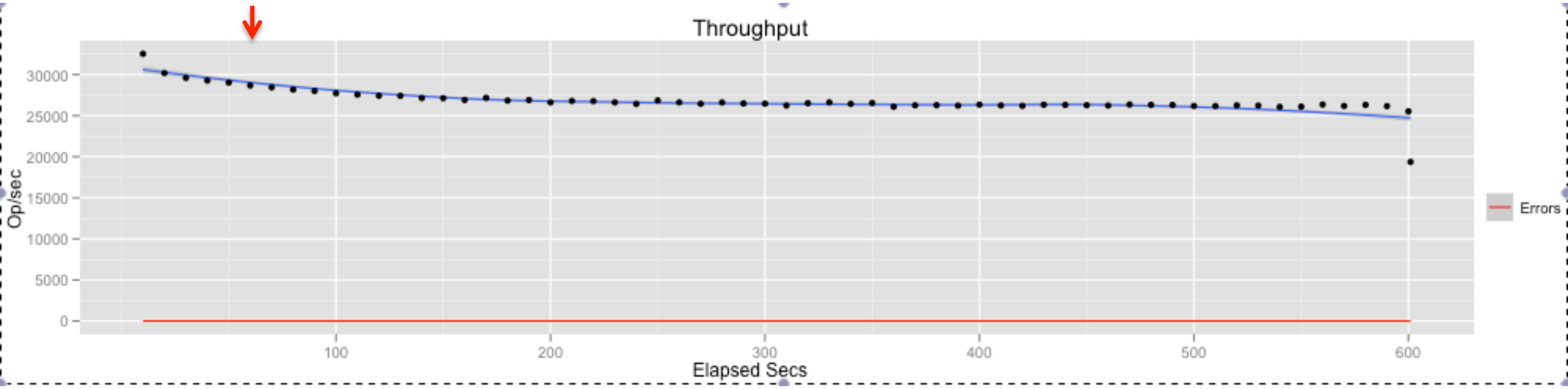
Performance Example 2



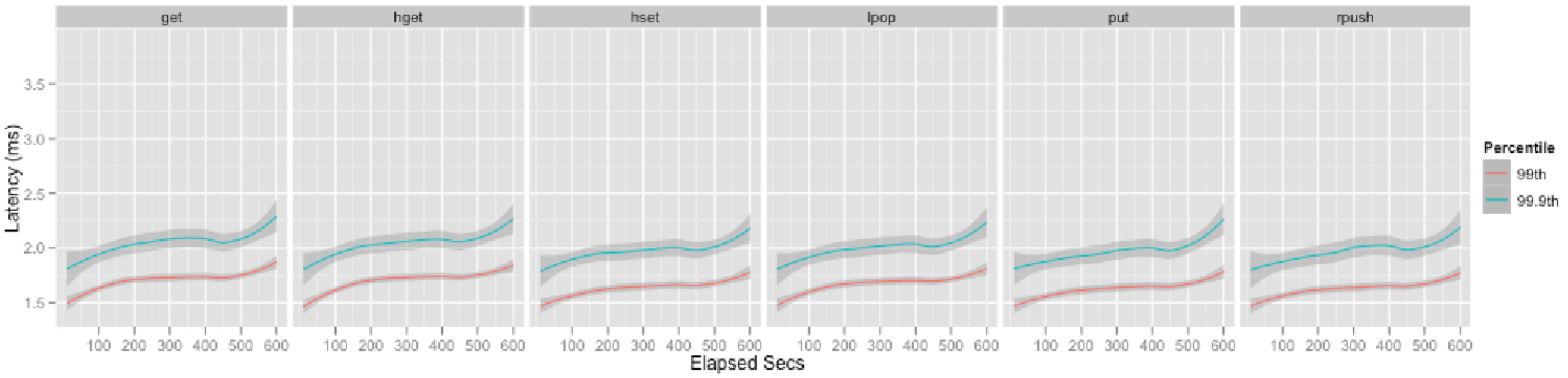
With snapshotting on

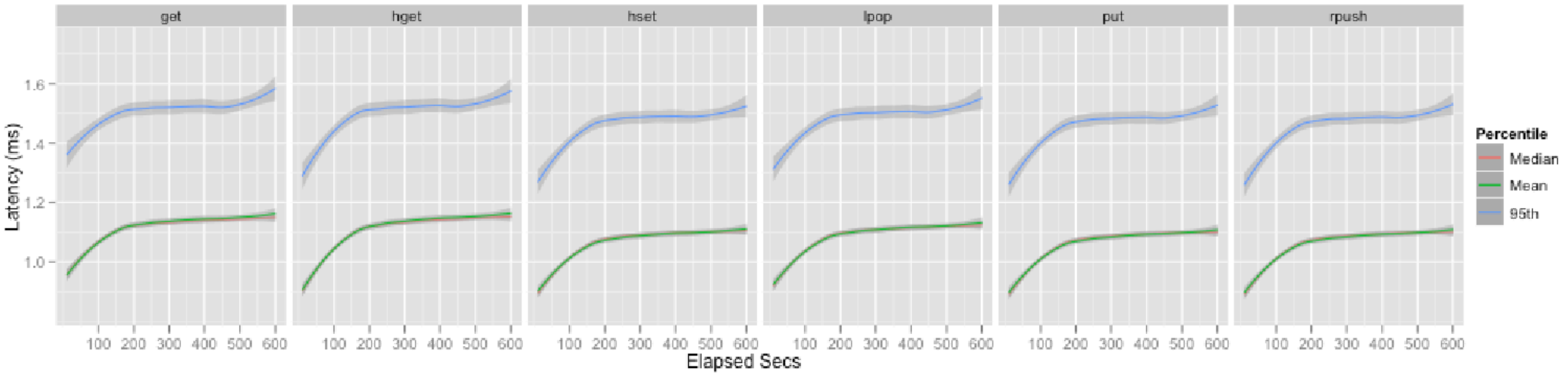
Average between 30,000-25,000 ops/sec across 10 minutes

With snapshotting off



Performance Example 2





Latencies average between <1-1.1ms
99.9% of all latencies < 2.0-2.3 ms

Languages

Most

(worth using and some that aren't)

C, C#, C++, Clojure, Common Lisp, Erlang, Go, Haskell, Io, Java, Lua, Node.js, Objective-C, Perl, PHP, Python, Ruby, Scala, and more...

http://redis.io/clients

More info

We just scratched the surface

Dig into redis.io for more depth

Google "redis"

I'd be glad to talk

Chris Meadows
meadoch1@gmail.com
@meadoch1 (Twitter)

https://github.com/meadoch1/RedisIntro