Redis 101

A whirlwind tour of the next big thing in NoSQL data storage

PETER COOPER

http://twitter.com/peterc

http://coder.io/

Whirlwind tour?

No overbearing detail.

A <u>quick</u> whizz-through.

Enough to get you excited
(if Redis is for you.)

Official docs, etc, are an awesome way to continue.

Redis is...

NoSQL

an "advanced key-value store"

by

SALVATORE SANFILIPPO

(@antirez)

NoSQL?

An informal, loosely-defined term for non-relational, structured data storage systems

Like MongoDB, memcached, CouchDB, and Redis

See http://en.wikipedia.org/wiki/Structured_storage for comparisons

memcached

The canonically simple example
a networked “hash in the sky” behind a simple protocol

Keys		**Values**
`page:index.html`	→	`<html><head>[...]`
`user:123:session`	→	`xDrSdEwd4dSlZkEkj+`
`login_count`	→	`“7464”`
`user:100:last_login_time`	→	`“102736485756”`

Everything’s a string (or a “blob”)
Commands just set or get data (mostly)

Take memcached's simplicity,
Add more data types,
Add persistence,
Add more commands,

.. and more™

Redis

Redis Data Types

Strings

Lists

Sets

Sorted/Scored Sets

Hashes

all accessed by a string "key"

Redis Data Examples

Keys		Values		
page:index.html	→	<html><head>[...]	←	String
login_count	→	7464		
users_logged_in_today	→	{ 1, 2, 3, 4, 5 }	←	Set
latest_post_ids	→	[201, 204, 209,..]	←	List

user:123:session	→	time => 10927353
		username => joe
			←	Hash

users_and_scores	→	joe ~ 1.3483 bert ~ 93.4 fred ~ 283.22 chris ~ 23774.17	←	Sorted (scored) Set

Strings

Redis command
line client app

Key

Value

```
./redis-cli SET mystring “hello world”
```

```
./redis-cli GET mystring
```

returns

```
“hello world”
```

Strings

GETSET	INCR
MGET	INCRBY
SETNX	DECR
SETEX	DECRBY
MSET	APPEND
MSETNX	SUBSTR

← Works on strings that appear to be integers. Magic!

http://code.google.com/p/redis/wiki/CommandReference

Expiration

When caching, you don't want things to live forever.

Any item in Redis can be made to expire after or at a certain time.

seconds

```
EXPIRE your_key 1234
TTL your_key == 1234
```

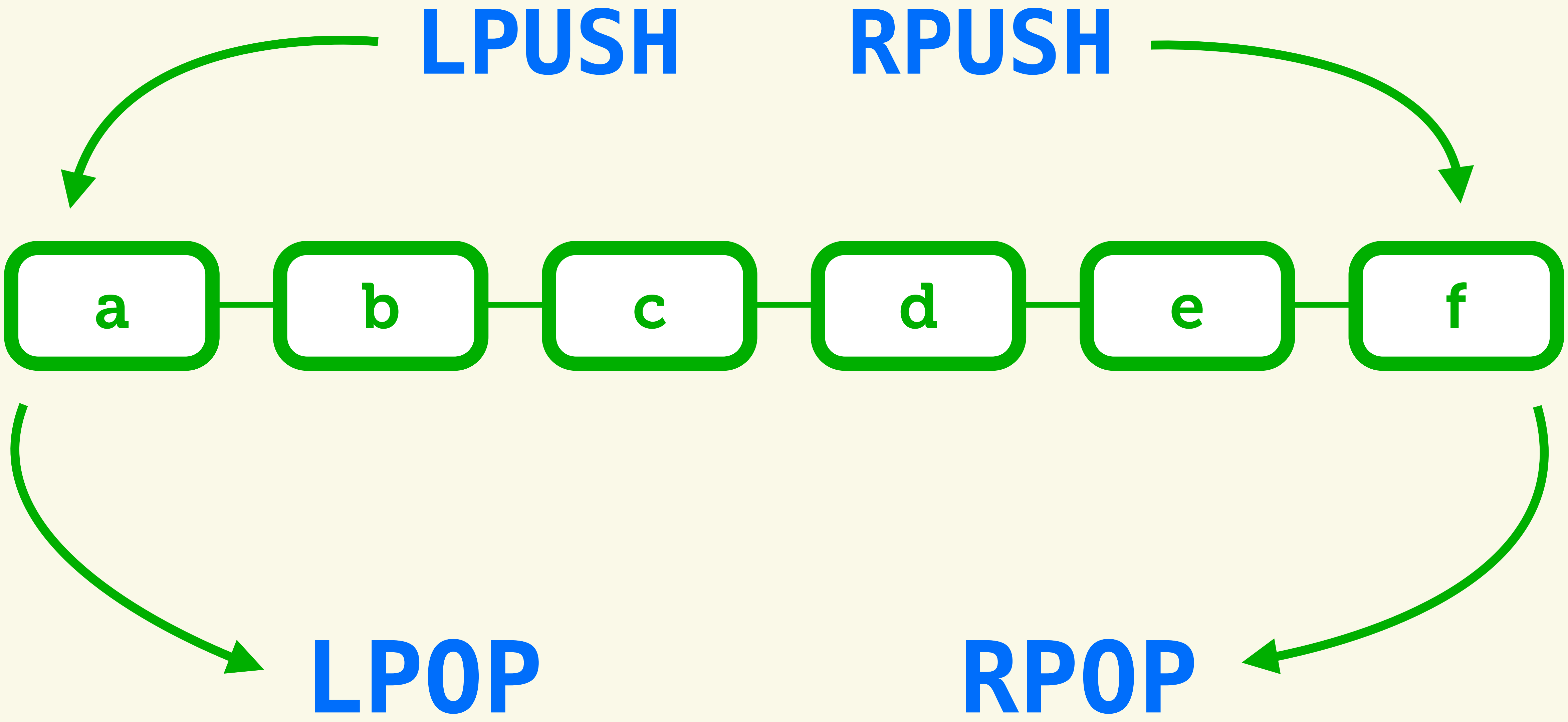
Deleting Keys

You can also delete data at will.

```
DEL your_key
```

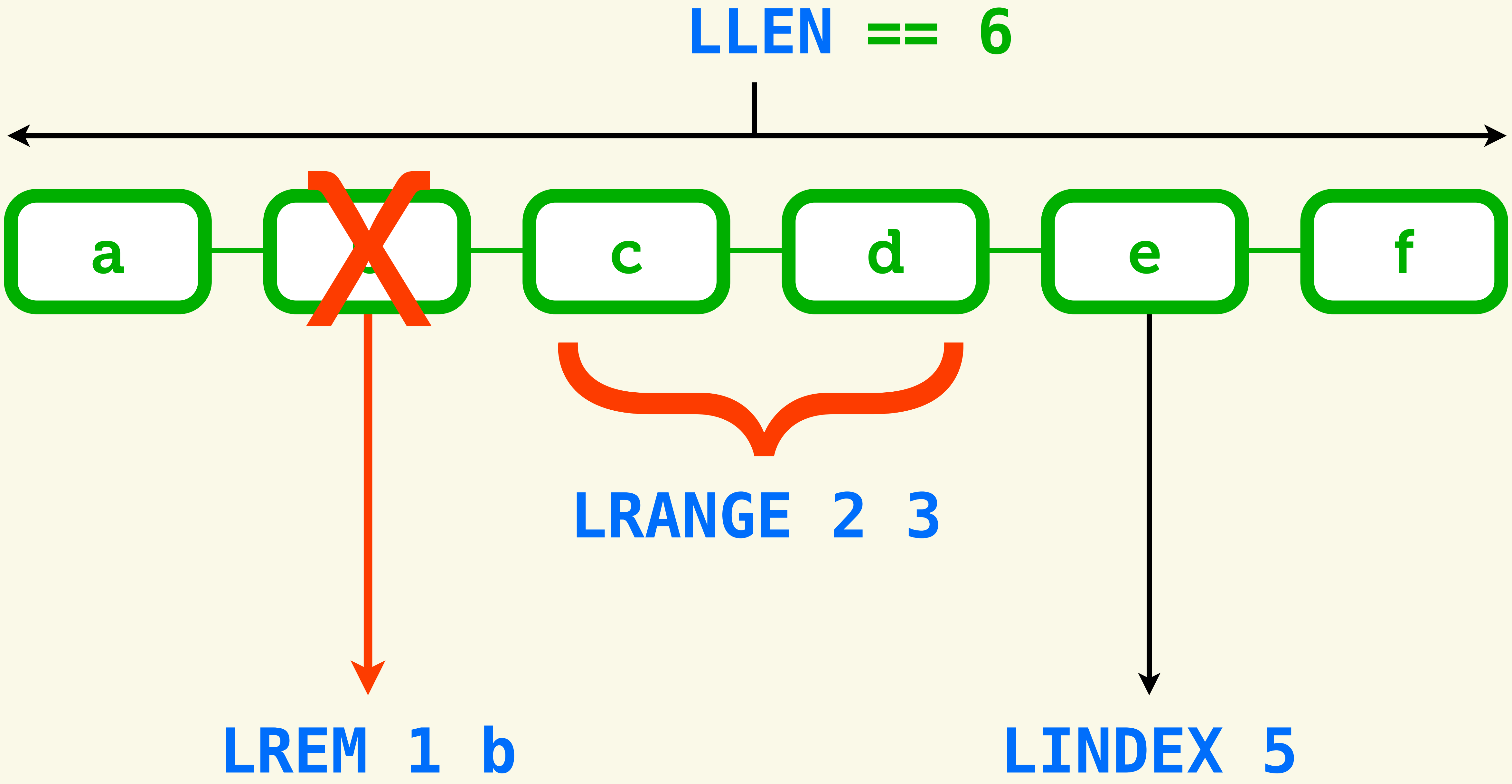
```
EXISTS your_key == 0 (false)
```

Lists



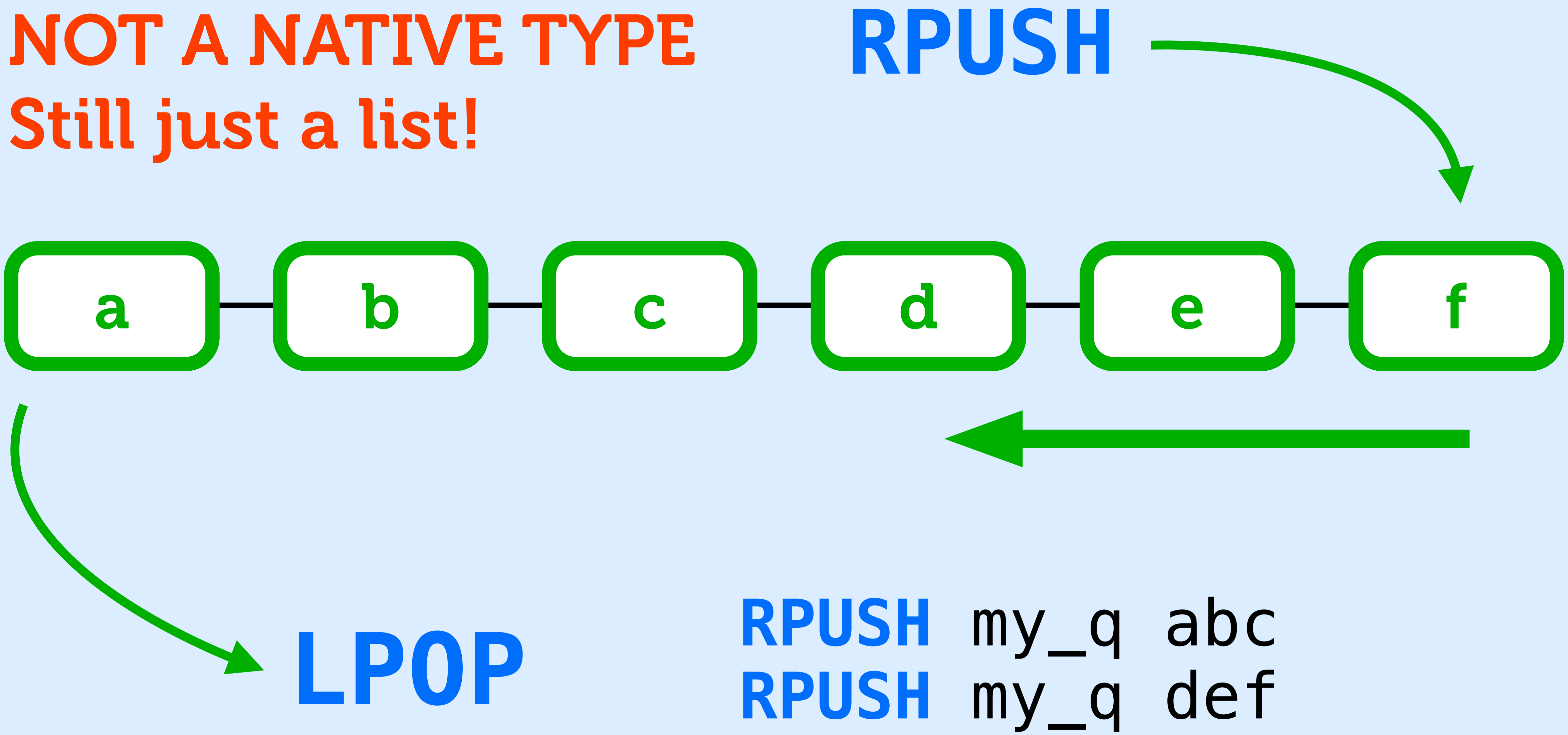
e.g. `RPUSH my_q f`

Lists



Queues

NOT A NATIVE TYPE
Still just a list!

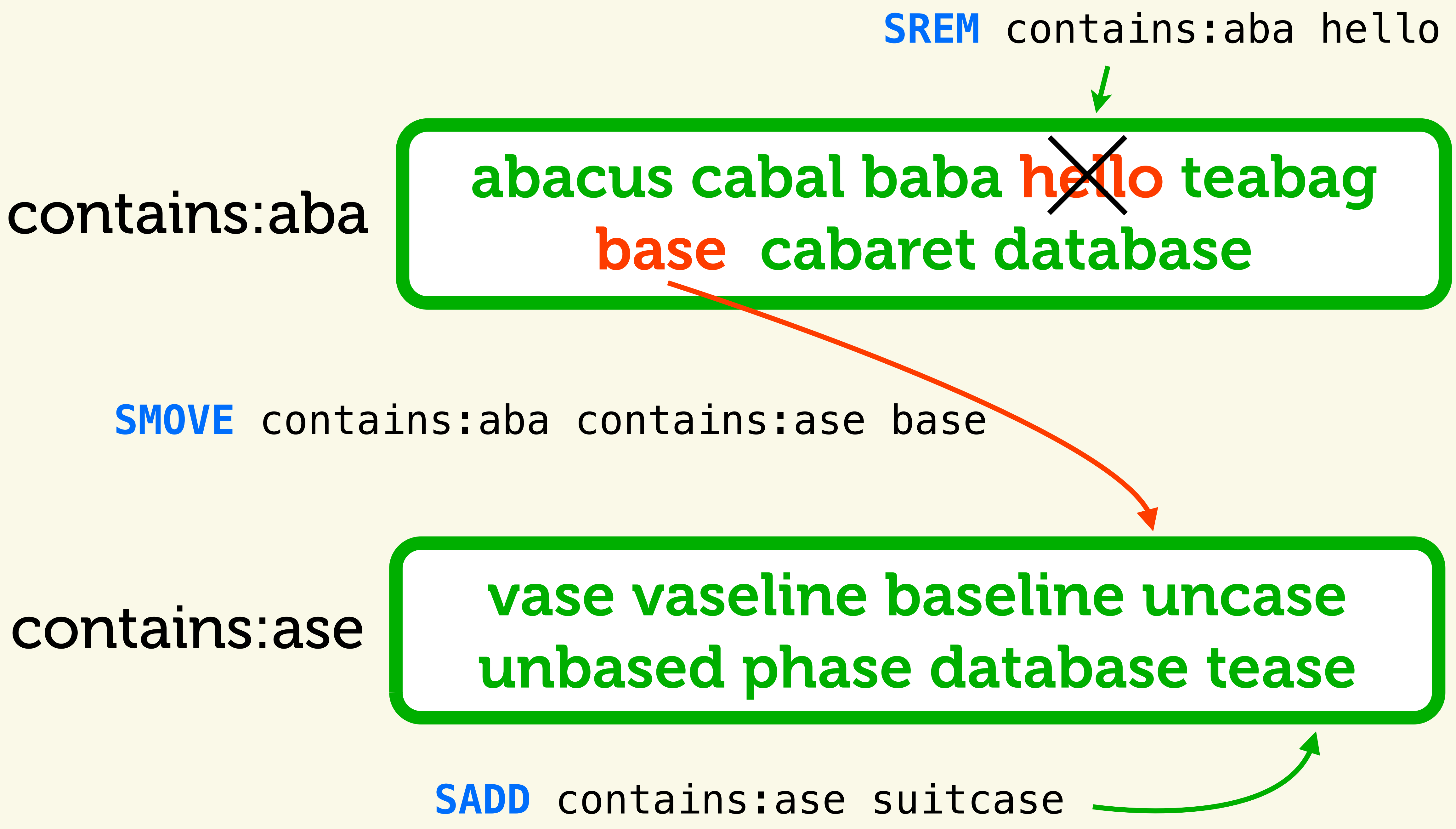


↑

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

```
RPUSH my_q abc
RPUSH my_q def
LPOP my_q        == “abc”
LPOP my_q        == “def”
LPOP my_q        == (nil)
```

Sets



```
SREM contains:aba hello
```

contains:aba
abacus cabal baba ~~hello~~ teabag
base cabaret database

```
SMOVE contains:aba contains:ase base
```

contains:ase
vase vaseline baseline uncase
unbased phase database tease

```
SADD contains:ase suitcase
```

Sets

contains:aba

abacus cabal baba teabag cabaret database

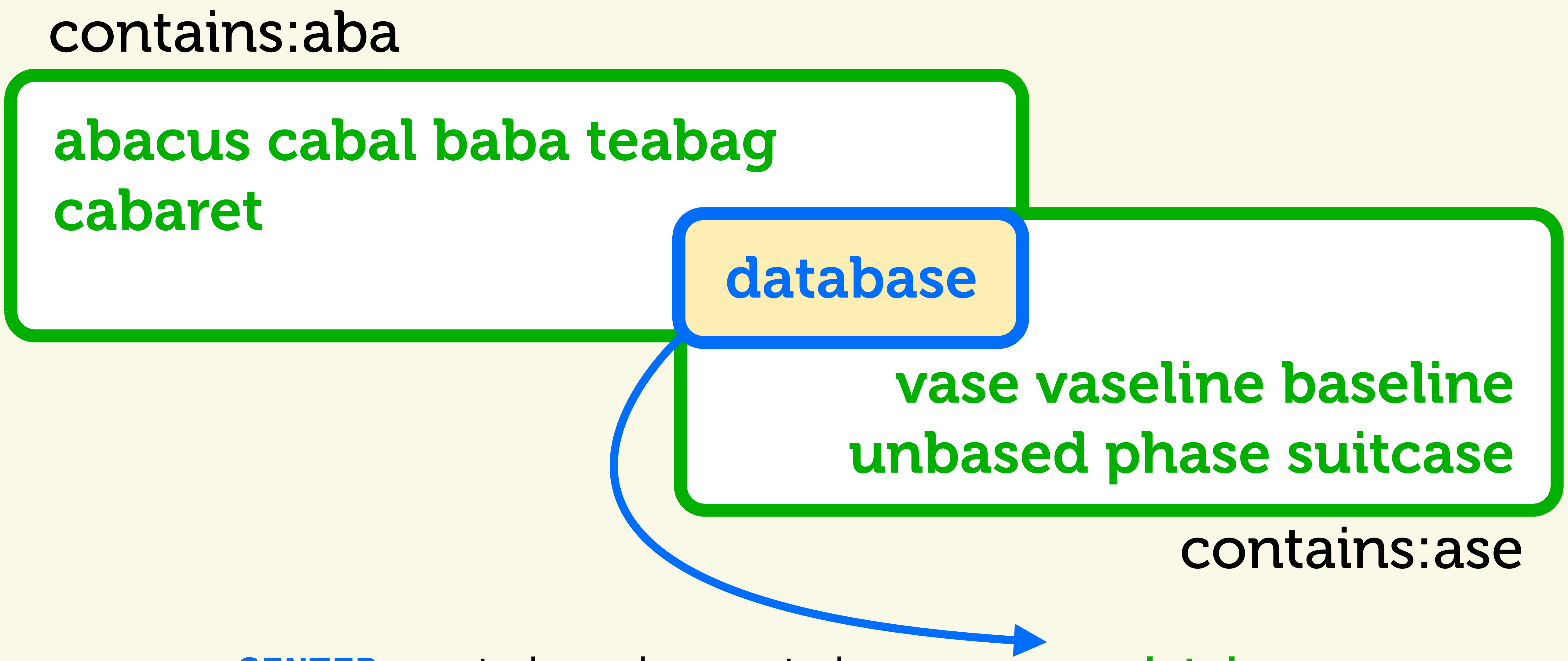
```
SCARD contains:aba                == 6
SISMEMBER contains:aba chips      == 0 (meaning false)
SRANDMEMBER contains:aba          == “teabag”
```

contains:ase

vase vaseline baseline unbased phase database suitcase

```
SMEMBERS contains:ase             == vase, vaseline,
                                     baseline, unbased,
                                     phase, database,
                                     suitcase
```

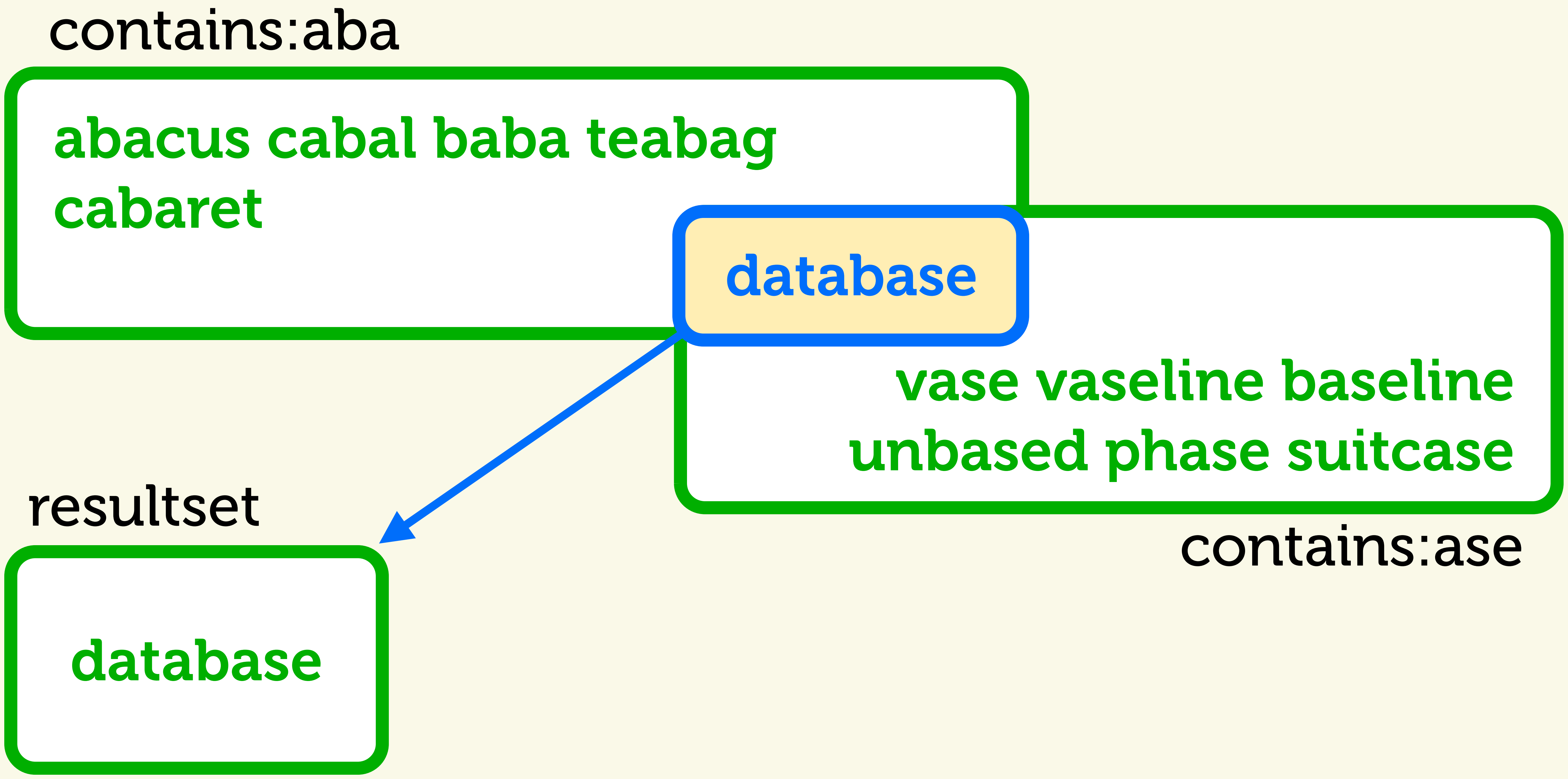
Sets



```
SINTER contains:aba contains:ase    == database
```

This is only a simple example. **SINTER** can take any number of arguments!
SUNION is another command that will join sets together.

Sets



```
SINTERSTORE resultset contains:aba contains:ase
```

`SUNIONSTORE` does the same for set unions.

Sorted Sets?

Sorry - no time!

Basically, like normal sets but each element can have a “rank” or “score” and be returned or sorted by it.

Hashes

product:1

created_at	102374657
product_id	1
name	Twinkies
available	10

```
HSET product:1 created_at 102374657

HSET product:1 product_id 1

HSET product:1 name “Twinkies”

HSET product:1 available 10
```

```
HGET product:1 name         == Twinkies

HLEN product:1              == 4

HKEYS product:1             == created_at, product_id,
                               name, available

HGETALL product:1           == created_at => 102374657
                               product_id => 1
                               [.. etc ..]
```

Also...

HVALS HEXISTS HINCRBY HMGET HMSET

Session Storage

Session 8d3e4
created_at: 102374657
user_id: 1

It's basically a hash

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

OR

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

Then let Redis automatically expire it in 24 hours!

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

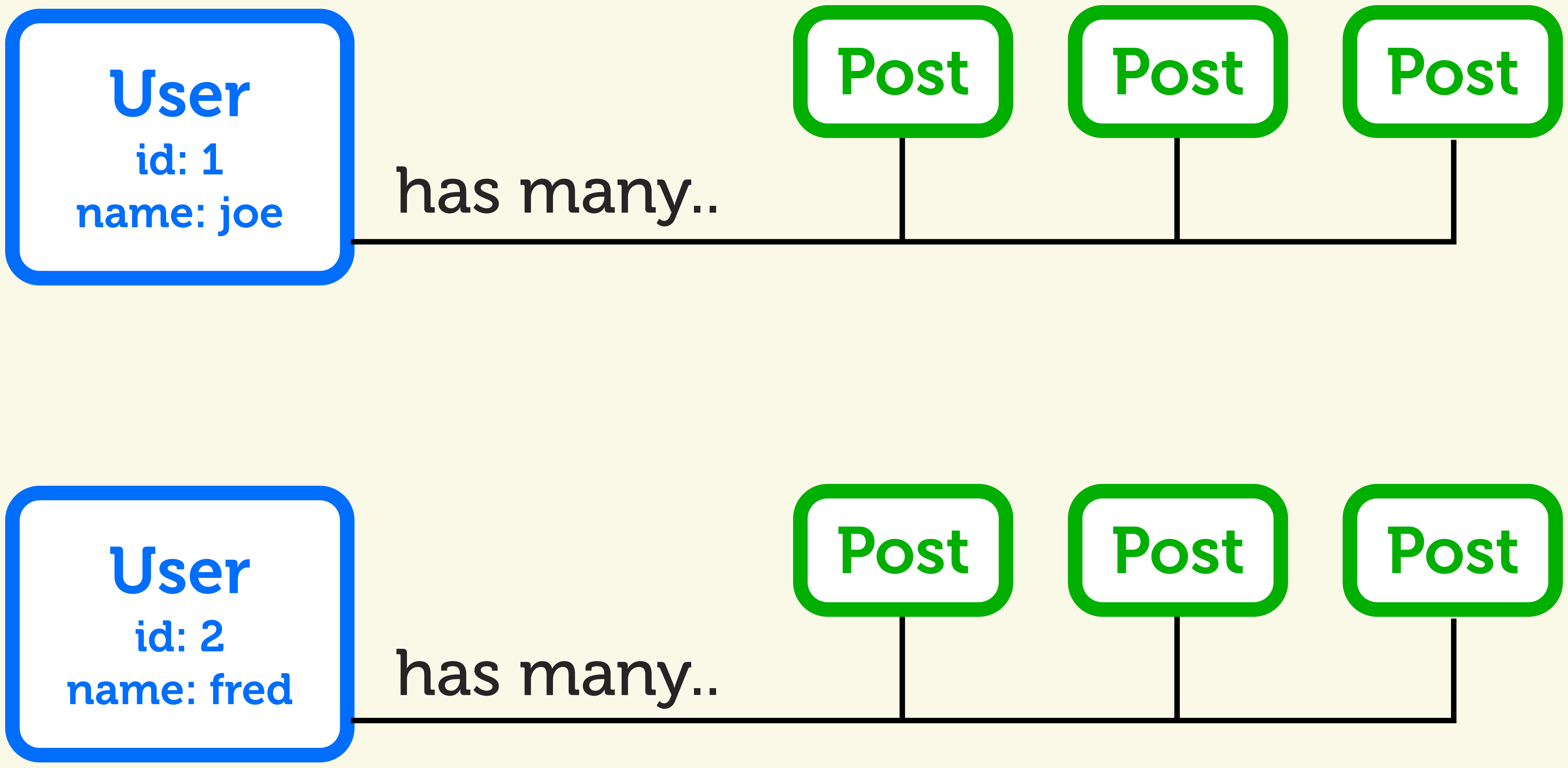
Redis Social Network

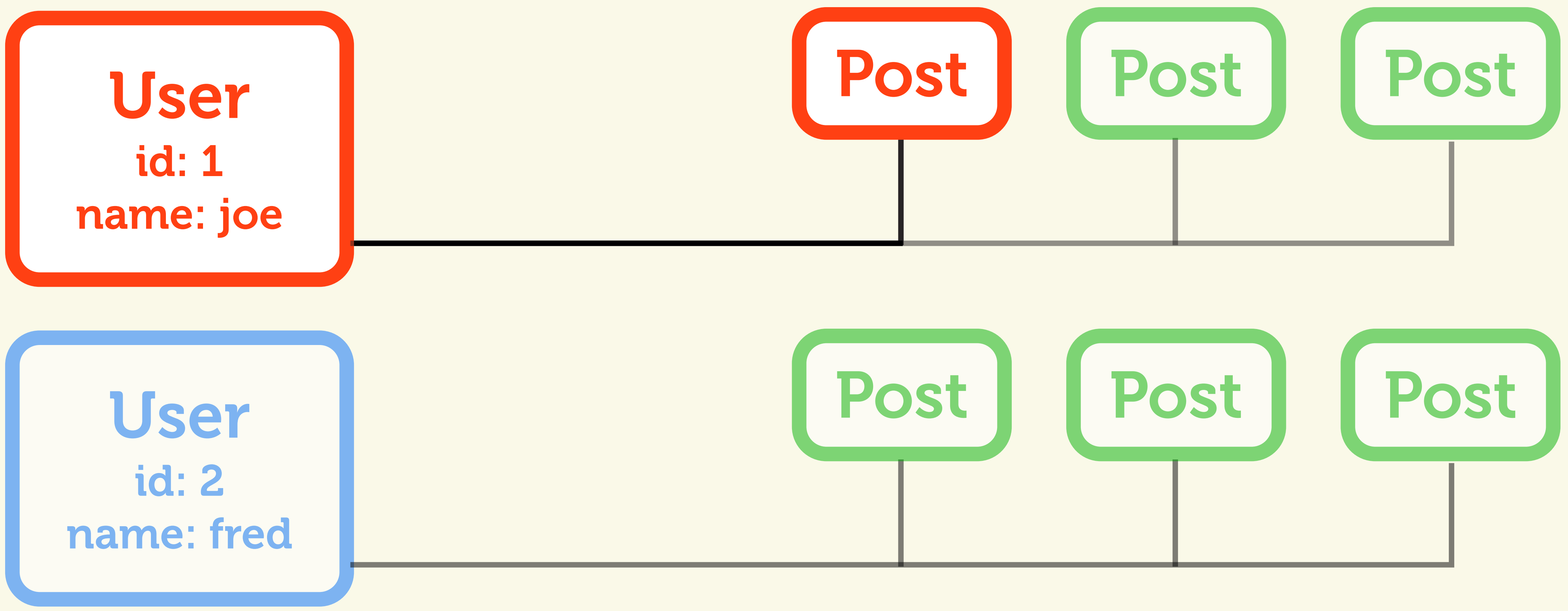
Users

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: fred
has many..
Post
Post
Post



user:1:name $\longrightarrow$ joe

username:joe $\longrightarrow$ 1 $\longleftarrow$ So we can do a two way reference

post:1:content $\rightarrow$ hello world

post:1:user $\longrightarrow$ 1 $\longleftarrow$ Ditto

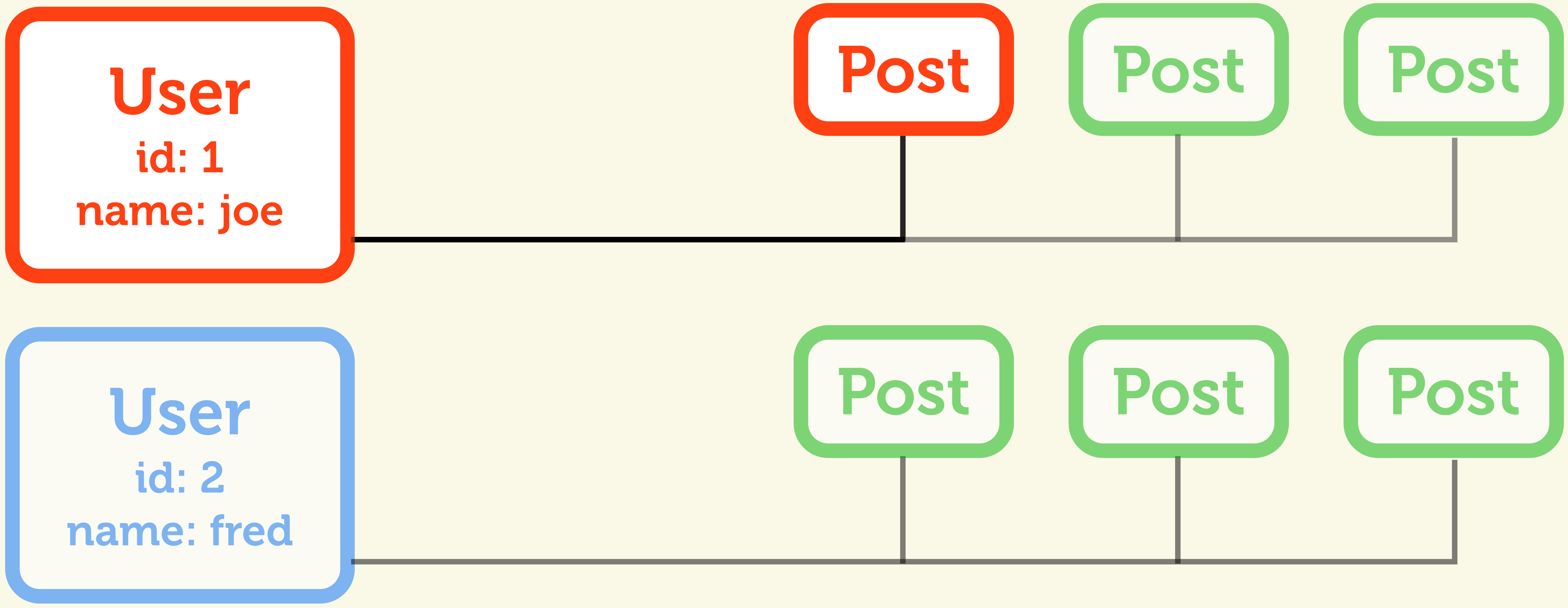
Building unique key names with colons like

`user:1:name`

is just a

convention

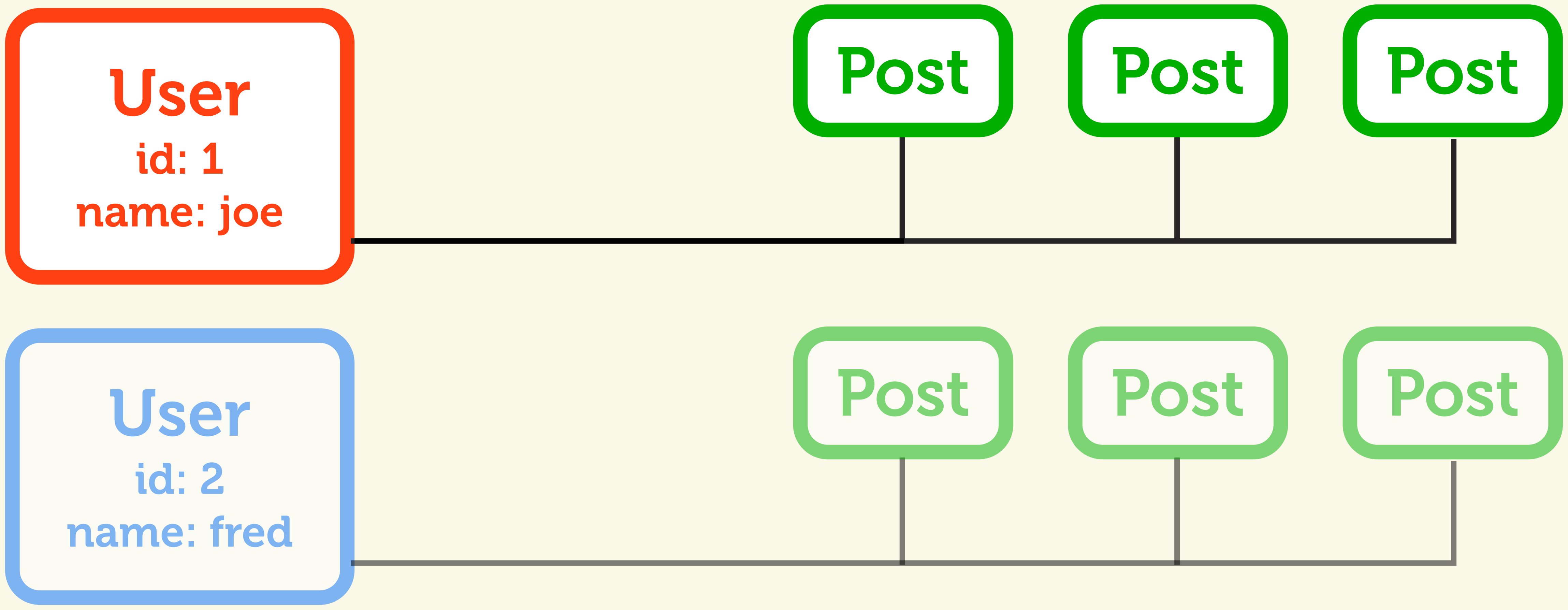
Any string will dooooo.....



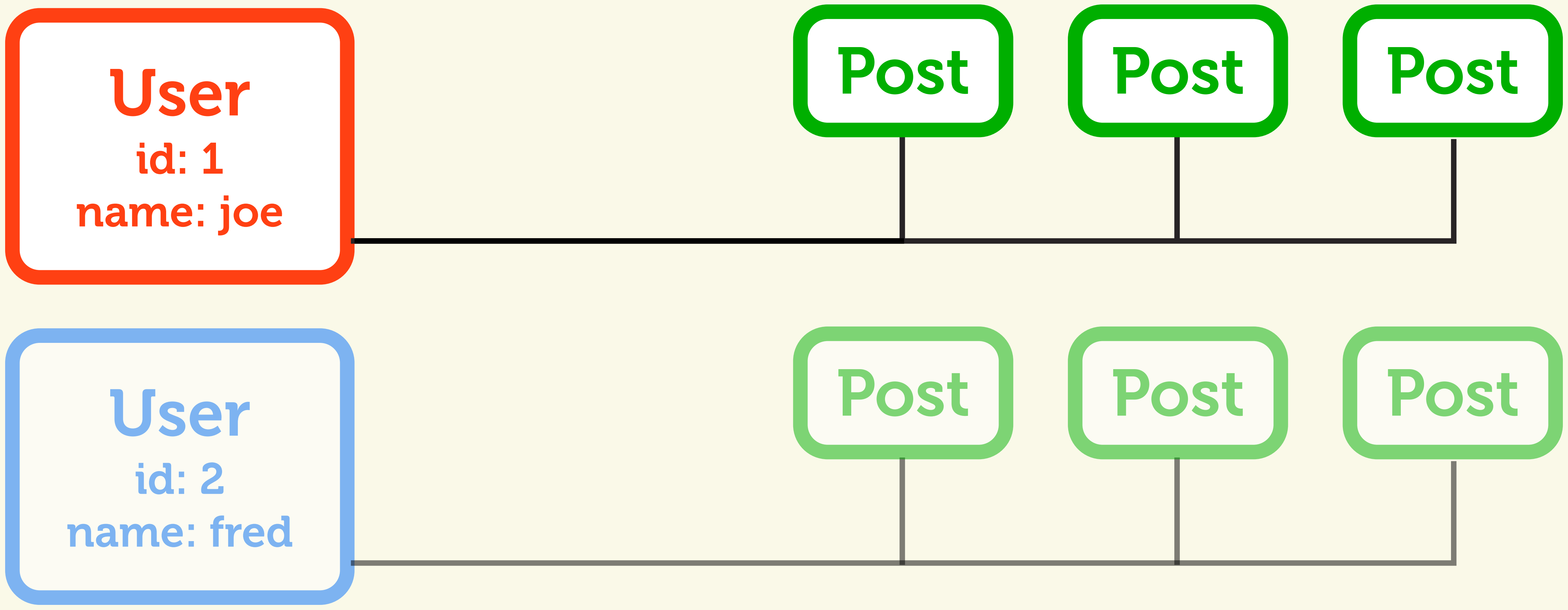
```
set user:1:name joe
set username:joe 1

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

↑

Remember, SET and GET are used for string values



$$\texttt{user:1:posts} \longrightarrow \texttt{[3, 2, 1]} \leftarrow \text{List}$$

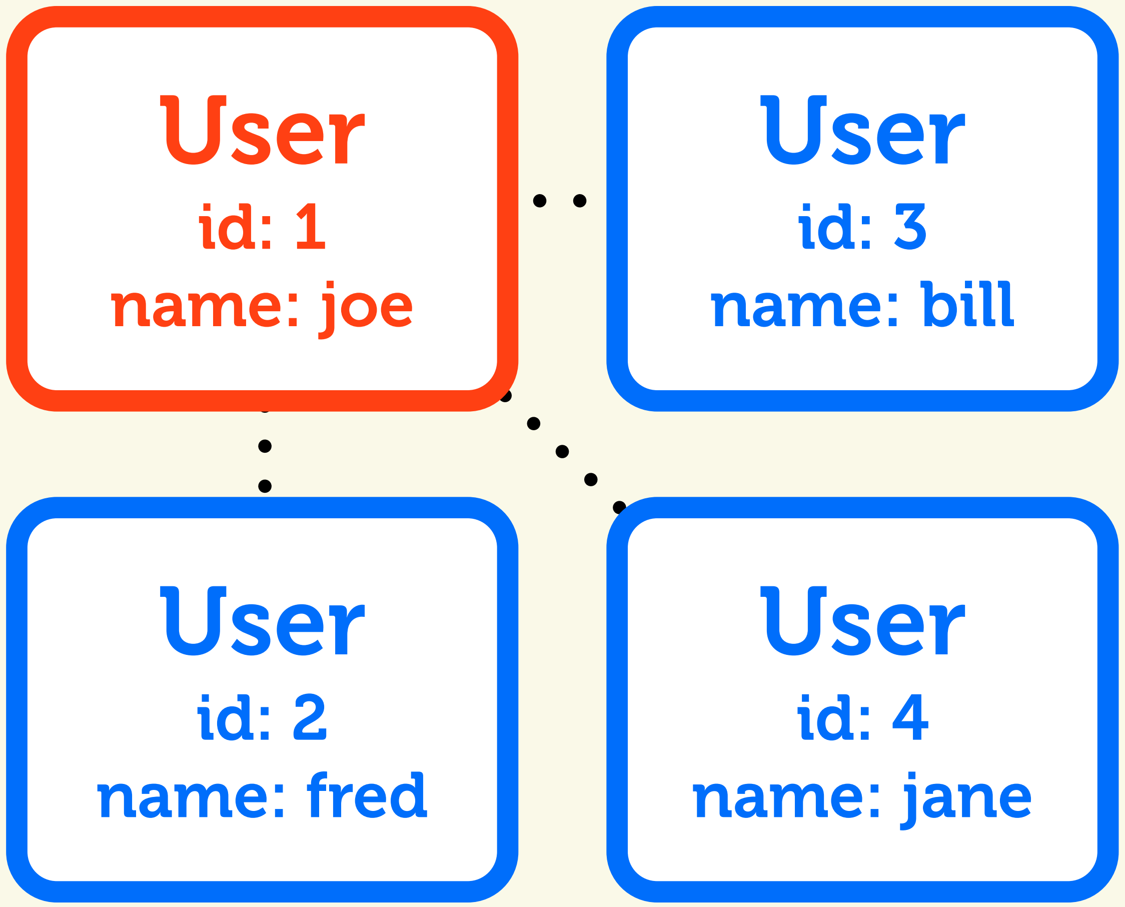


$$\texttt{user:1:posts} \longrightarrow \texttt{[3, 2, 1]}$$

```
lpush user:1:posts 1
lpush user:1:posts 2
lpush user:1:posts 3
```

↑

LPUSH and RPUSH add items to the start or end of a list



```
user:1:follows ⟶ {2, 3, 4}
```

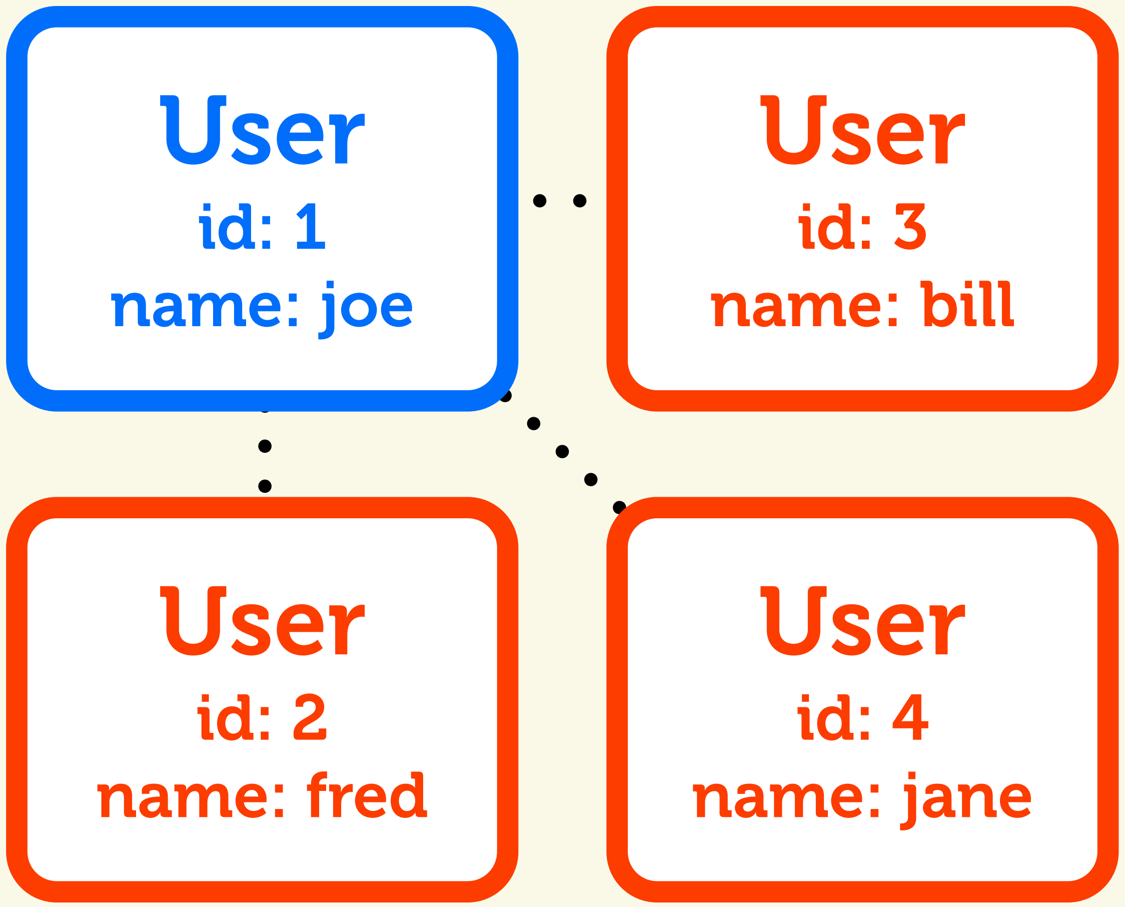
← Set

Order not important

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

↑

SADD and SREM add or remove elements to/from a set



You might want to track the relationship in the opposite direction too. Just create another set!

```
user:1:followed_by ⟶ {3}
```

```
sadd user:1:followed_by 3
```

A Simple Social Network

Keys	Values	
user:1:name	joe	
user:2:name	fred	
username:joe	1	
username:fred	2	
user:1:follows	{2,3,4}	← Set
user:2:follows	{1}	
user:1:followed_by	{2}	
user:2:followed_by	{1}	
post:1:content	“Hello world”	
post:1:user	2	
post:2:content	“Blah blah”	
post:2:user	1	
user:1:posts	[2,3,4]	← List
user:2:posts	[1,5,6]	

Simplified from the earlier graphs due to lack of space :-)

Unique IDs

```
INCR next_post_id
```

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

returns → **1** → `post:1:etc`

```
INCR next_post_id
```

INCR increments the element by 1 and returns the new value. Great for unique IDs!

returns → **2**

or `next_user_id`!

Creating a new user

```
INCR next_user_id   returns ⟶ [uid]
SET user:[uid]:name [username]
SET username:[username] [id]
```

Creating a new post

```
INCR next_post_id   returns ⟶ [pid]
SET post:[pid]:content [content]
SET post:[pid]:user [pid]
LPUSH user:[uid]:posts [pid]
LPUSH posts:global [pid]
```

SORT MONITOR SUBSCRIBE

ZCARD PUBLISH

SLAVEOF

RENAME SAVE

SELECT

Enough commands!

I haven't covered them all though..

On to softer **issues.**

Atomicity

Redis is single threaded
No locking necessary

In other words, commands like INCR won’t tread on each other’s toes coming from multiple clients simultaneously!

Redis Factoids

BSD licensed (free, open)

Sponsored by VMware

Written in ANSI C

Good community (list, IRC & wiki)

Works on all POSIX-compliant UNIXes

An unofficial Windows/Cygwin build is available

Installation

Download a tarball or clone the git repo

Run make

redis-server and redis-cli are ready to roll

(You can make a config file later, if you want.)

http://code.google.com/p/redis/

Performance

Depends a lot on configuration and operation complexity.

Common range from 5000 to 120,000 rps for basic ops GET/SET/LPUSH/LPOP, etc.
(ultra low end to high end hardware)

Performance

redis-benchmark tool on a CentOS virtual machine on a 2009 iMac

GET: 28011 rps

SET: 36101 rps

INCR: 36496 rps

LPUSH: 38759 rps

LPOP: 38610 rps

$\}$ **average ~36000**

And that's with **1024 byte** payloads!

Persistence

Dump data to disk after certain conditions are met. Or manually. ← SAVE and BGSAVE commands

AND/OR

An append only log file

(which can be optimized/rebuilt automatically)

↖ but you need to set this up in a config file

Language Support

Ruby, Python, PHP, Erlang,
Tcl, Perl, Lua, Java, Scala,
Clojure, C#, C/C++,
JavaScript/Node.js, Haskell,
IO, Go

i.e. anything actually worth using

Missed a lot, so where next!?

Google "Redis"
the official site is great

http://coder.io/tag/redis
for news and articles

P.S. I'm writing a Redis book a little like this presentation.
E-mail peter@peterc.org to be put on an announce list!