real world redis

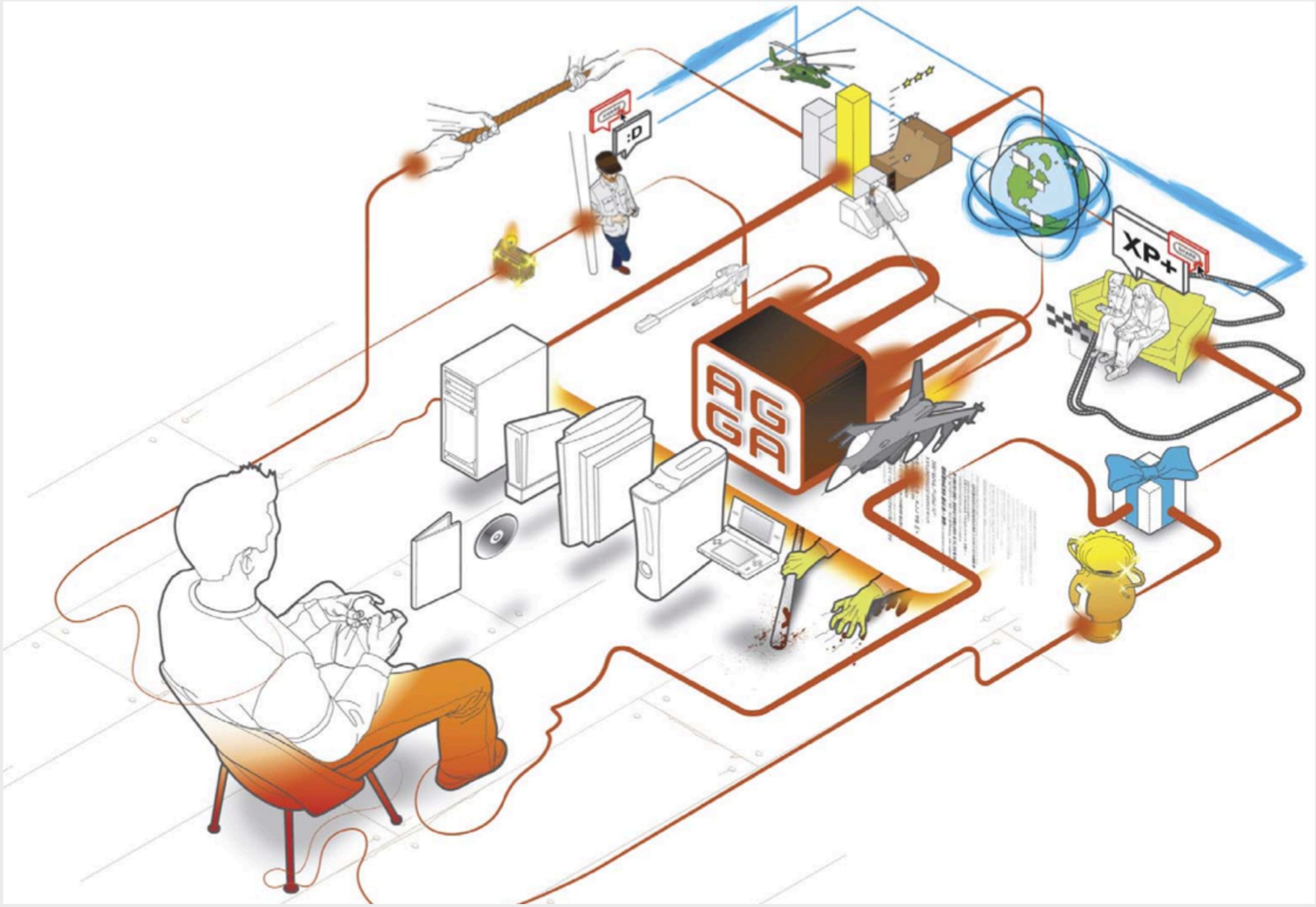
strange loop conference

september 23rd-25th, 2012

david czarnecki

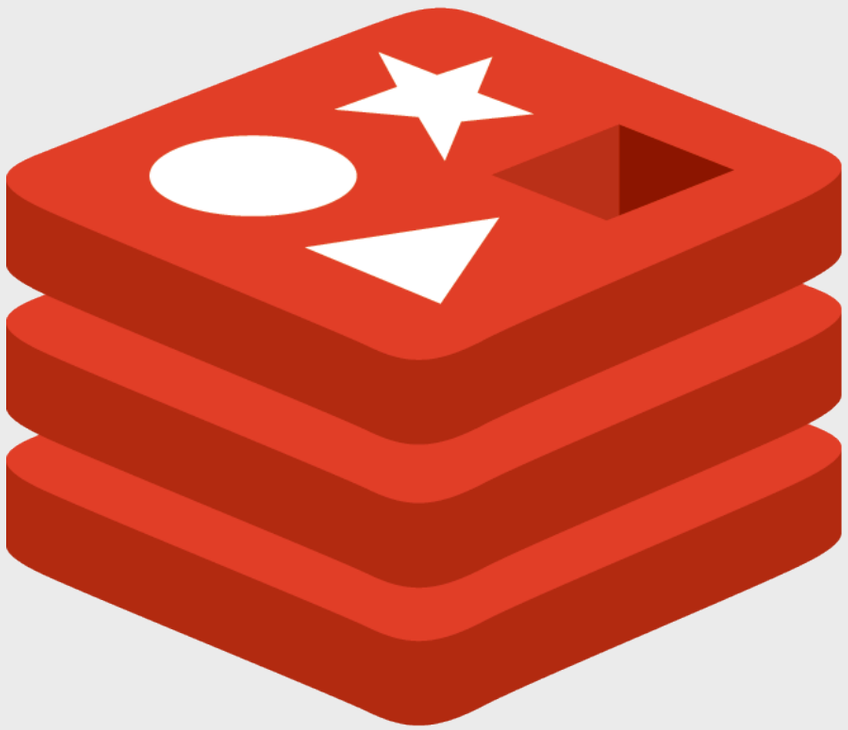
https://speakerdeck.com/u/czarneckid/

@czarneckid



@agoragames

our portfolio



http://redis.io

remote dictionary server

data structure server

it’s badass

string

caching, counters

```
# Simple get and set with redis
require 'redis'

redis = Redis.new
redis.flushdb

redis.set('foo', 'bar')
redis.get('foo')
#  => 'bar'
old_value = redis.getset('foo', 'baz') # atomic operation
# => 'bar'
redis.get('foo')
# => 'baz'
```

```
# Set, get and append with redis
require 'redis'

redis = Redis.new
redis.flushdb

redis.set('key', 'Hello')
redis.get('key')
# => 'Hello'
redis.append('key', ' World')

redis.get('key')
#  => 'Hello World'
```

```
# Set a key with expiration
require 'redis'

redis = Redis.new
redis.flushdb

redis.setex('key', 10, 'Hello')
redis.get('key')
#  => 'Hello'

sleep(10)

redis.get('key')
#  => nil
```

```
# Increment, decrement, increment by and decrement by operations
require 'redis'

redis = Redis.new
redis.flushdb

redis.incr('key')
#  => 1
redis.incrby('key', 10)
#  => 11

redis.decr('key')
#  => 10
redis.decrby('key', 10)
#  => 0
```

redis-store

https://github.com/jodosha/redis-store

hash

store objects

```
# Simple hash get and set with redis
require 'redis'

redis = Redis.new
redis.flushdb

redis.hset('hash', 'foo', 'bar')
redis.hset('hash', 'biz', 'buzz')

redis.hget('hash', 'foo')
#  => 'bar'
redis.hget('hash', 'biz')
#  => 'buzz'
redis.hget('hash', 'unknown')
#  => nil
```

```
# Interact with keys and values from a hash in redis
require 'redis'

redis = Redis.new
redis.flushdb

redis.hset('hash', 'foo', 'bar')
redis.hset('hash', 'biz', 'buzz')

redis.hkeys('hash')
#  => ['foo', 'biz']
redis.hvals('hash')
#  => ['bar', 'buzz']
redis.hgetall('hash')
#  => {'foo'=>'bar', 'biz'=>'buzz'}
```

```
# Information about a hash in redis
require 'redis'

redis = Redis.new
redis.flushdb

redis.hset('hash', 'foo', 'bar')
redis.hset('hash', 'biz', 'buzz')

redis.hlen('hash')
#  => 2
redis.hexists('hash', 'foo')
#  => true
redis.hexists('hash', 'unknown')
#  => false
```

```
# Perform multiple get, set and delete operations on a hash in redis
require 'redis'

redis = Redis.new
redis.flushdb

redis.hmset('hash', 'foo', 'bar', 'biz', 'buzz')
redis.hmget('hash', 'foo', 'biz', 'unknown')
#  => ['bar', 'buzz', nil]
redis.hdel('hash', 'foo')
redis.hgetall('hash')
#  => {'biz'=>'buzz'}
```

ohm

https://github.com/soveran/ohm

list

message passing

```
# Basic list operations in redis
require 'redis'

redis = Redis.new
redis.flushdb

redis.lpush('list', 'bar')
redis.rpush('list', 'baz')
redis.lpush('list', 'foo')

redis.lindex('list', 1)
#  => 'bar'
redis.lrange('list', 0, -1)
#  => ['foo', 'bar', 'baz']
redis.lpop('list')
#  => 'foo'
redis.rpop('list')
#  => 'baz'
```

```
# More list operations in redis
require 'redis'

redis = Redis.new
redis.flushdb

redis.lpush('list', 'bar')
redis.rpush('list', 'baz')
redis.lpush('list', 'foo')

redis.llen('list')
redis.linsert('list', 'before', 'bar', 'foozy')
redis.lrange('list', 0, -1)
#  => ['foo', 'foozy', 'bar', 'baz']
redis.lset('list', 1, 'loozy')
redis.lrange('list', 0, -1)
#  => ['foo', 'loozy', 'bar', 'baz']
```

```
# List pop/push operations in redis
require 'redis'

redis = Redis.new
redis.flushdb

redis.lpush('list', 'bar')
redis.rpush('list', 'baz')
redis.lpush('list', 'foo')

redis.brpoplpush('list', 'another_list', 0)
#  => 'baz'
redis.brpoplpush('list', 'another_list', 0)
#  => 'bar'
redis.lrange('list', 0, -1)
#  => ['foo']
redis.lrange('another_list', 0, -1)
#  => ['bar', 'baz']
```

resque

https://github.com/defunkt/resque

set

tracking, membership

```
# Basic set operations in redis
require 'redis'

redis = Redis.new
redis.flushdb

redis.sadd('users', 'david')
redis.sadd('users', 'waldo')
redis.sadd('users', 'matthew')

redis.scard('users')
#  => 3
redis.sismember('users', 'david')
#  => true
redis.sismember('users', 'john')
#  => false
redis.smembers('users')
#  => ['waldo', 'david', 'matthew']
```

```
# Intersection set operations in redis
require 'redis'

redis = Redis.new
redis.flushdb

redis.sadd('set_1', 'a')
redis.sadd('set_1', 'b')
redis.sadd('set_1', 'c')
redis.sadd('set_2', 'c')
redis.sadd('set_3', 'c')
redis.sadd('set_3', 'd')

redis.sinter('set_1', 'set_2', 'set_3')
# => ['c']
redis.sinterstore('set_4', 'set_1', 'set_2', 'set_3')
redis.smembers('set_4')
# => ['c']
```

```
# Difference set operations in redis
require 'redis'

redis = Redis.new
redis.flushdb

redis.sadd('set_1', 'a')
redis.sadd('set_1', 'b')
redis.sadd('set_1', 'c')
redis.sadd('set_2', 'c')
redis.sadd('set_3', 'c')
redis.sadd('set_3', 'd')

redis.sdiff('set_1', 'set_2', 'set_3')
# => ['a', 'b']
redis.sdiffstore('set_4', 'set_1', 'set_2', 'set_3')
redis.smembers('set_4')
# => ['a', 'b']
```

```
# Union set operations in redis
require 'redis'

redis = Redis.new
redis.flushdb

redis.sadd('set_1', 'a')
redis.sadd('set_1', 'b')
redis.sadd('set_1', 'c')
redis.sadd('set_2', 'c')
redis.sadd('set_3', 'd')
redis.sadd('set_3', 'e')

redis.sunion('set_1', 'set_2', 'set_3')
# => ['c', 'd', 'a', 'b', 'e']
redis.sunionstore('set_4', 'set_1', 'set_2', 'set_3')
redis.smembers('set_4')
# => ['c', 'd', 'a', 'b', 'e']
```

rollout

https://github.com/jamesgolick/rollout

sorted set

leaderboards, activity feeds

```
# Basic sorted set operations in redis
require 'redis'

redis = Redis.new
redis.flushdb

redis.zadd('highscores', 100, 'david')
redis.zadd('highscores', 85, 'waldo')
redis.zadd('highscores', 150, 'matthew')

redis.zcard('highscores')
# => 3
redis.zcount('highscores', 80, 110)
# => 2
redis.zrange('highscores', 0, -1, :with_scores => true)
# => [['waldo', 85.0], ['david', 100.0], ['matthew', 150.0]]
redis.zrevrange('highscores', 0, -1, :with_scores => true)
# => [['matthew', 150.0], ['david', 100.0], ['waldo', 85.0]]
```

```
# More sorted set operations in redis
require 'redis'

redis = Redis.new
redis.flushdb

redis.zadd('highscores', 100, 'david')
redis.zadd('highscores', 85, 'waldo')
redis.zadd('highscores', 150, 'matthew')

redis.zrank('highscores', 'matthew')
# => 2
redis.zrevrank('highscores', 'matthew')
#  => 0
redis.zscore('highscores', 'david')
#  => 100.0
redis.zrem('highscores', 'waldo')
```

leaderboard

https://github.com/agoragames/leaderboard

activity_feed

https://github.com/agoragames/activity_feed

amico

https://github.com/agoragames/amico

publish/subscribe

you can

PUBLISH

and

SUBSCRIBE

and

UNSUBSCRIBE

“transactions”

```
redis.multi do
  redis.set ‘foo’, ‘bar’
  redis.set ‘baz’, ‘buzz’
end
```

persistent storage

RDB

point-in-time snapshot

AOF

append-only file

you can also use both

replication

master/slave

redis.conf

slaveof 192.168.1.1 6379

security

“trusted environments”

```
# redis.conf

# require clients to issue AUTH <password> before processing commands
requirepass SECURITYLOLLERSKATE

# rename a command to something that is “unguessable”
rename-command FLUSHALL b840fc02d524045429941cc15f59e41cb7b

# completely kill a command
rename-command FLUSHALL “”
```

by now you're thinking...

zomg rainbow ponies

let's talk redis libraries

i don’t know about you, but...

commands like

BRPOPLPUSH

or

ZREVRANGEBYSCORE

make me go <(`^´)>

think of

redis libraries

as

semantic wrappers

let's cover a few in detail

leaderboard

https://github.com/agoragames/leaderboard

leaderboards

aka scoreboards

```
def rank_member_in(leaderboard_name, member, score,
member_data)
  @redis_connection.multi do |transaction|
    transaction.zadd(leaderboard_name, score, member)
    if member_data
      transaction.hmset(member_data_key(leaderboard_name,
member), *member_data.to_a.flatten)
    end
  end
end
```

```
def total_members_in(leaderboard_name)
  @redis_connection.zcard(leaderboard_name)
end

def total_pages_in(leaderboard_name, page_size = nil)
  page_size ||= @page_size.to_f
  (total_members_in(leaderboard_name) / page_size.to_f).ceil
end

def total_members_in_score_range_in(leaderboard_name, min_score,
max_score)
  @redis_connection.zcount(leaderboard_name, min_score, max_score)
end
```

```
def leaders_in(leaderboard_name, current_page, options = {})
  leaderboard_options = DEFAULT_LEADERBOARD_REQUEST_OPTIONS.dup
  leaderboard_options.merge!(options)

  if current_page < 1
    current_page = 1
  end

  page_size = validate_page_size(leaderboard_options[:page_size]) || @page_size

  if current_page > total_pages_in(leaderboard_name, page_size)
    current_page = total_pages_in(leaderboard_name, page_size)
  end

  index_for_redis = current_page - 1

  starting_offset = (index_for_redis * page_size)
  if starting_offset < 0
    starting_offset = 0
  end

  ending_offset = (starting_offset + page_size) - 1

  if @reverse
    raw_leader_data = @redis_connection.zrange(leaderboard_name, starting_offset, ending_offset, :with_scores => false)
  else
    raw_leader_data = @redis_connection.zrevrange(leaderboard_name, starting_offset, ending_offset, :with_scores => false)
  end

  if raw_leader_data
    return ranked_in_list_in(leaderboard_name, raw_leader_data, leaderboard_options)
  else
    return []
  end
end
```

```
def around_me_in(leaderboard_name, member, options = {})
  leaderboard_options = DEFAULT_LEADERBOARD_REQUEST_OPTIONS.dup
  leaderboard_options.merge!(options)

  reverse_rank_for_member = @reverse ?
    @redis_connection.zrank(leaderboard_name, member) :
    @redis_connection.zrevrank(leaderboard_name, member)

  return [] unless reverse_rank_for_member

  page_size = validate_page_size(leaderboard_options[:page_size]) || @page_size

  starting_offset = reverse_rank_for_member - (page_size / 2)
  if starting_offset < 0
    starting_offset = 0
  end

  ending_offset = (starting_offset + page_size) - 1

  raw_leader_data = @reverse ?
    @redis_connection.zrange(leaderboard_name, starting_offset, ending_offset, :with_scores => false) :
    @redis_connection.zrevrange(leaderboard_name, starting_offset, ending_offset, :with_scores => false)

  if raw_leader_data
    return ranked_in_list_in(leaderboard_name, raw_leader_data, leaderboard_options)
  else
    return []
  end
end
```

activity_feed

https://github.com/agoragames/activity_feed

activity feeds

aka timelines

```
def update_item(user_id, item_id, timestamp, aggregate =
ActivityFeed.aggregate)
  feederboard = ActivityFeed.feederboard_for(user_id, false)
  feederboard.rank_member(item_id, timestamp)

  if aggregate
    feederboard = ActivityFeed.feederboard_for(user_id, true)
    feederboard.rank_member(item_id, timestamp)
  end
end
```

```
def remove_item(user_id, item_id)
  feederboard = ActivityFeed.feederboard_for(user_id, false)
  feederboard.remove_member(item_id)
  feederboard = ActivityFeed.feederboard_for(user_id, true)
  feederboard.remove_member(item_id)
end
```

```
def feed(user_id, page, aggregate = ActivityFeed.aggregate)
  feederboard = ActivityFeed.feederboard_for(user_id, aggregate)
  feed = feederboard.leaders(page, :page_size => ActivityFeed.page_size).inject([]) do |
feed_items, feed_item|
    item = if ActivityFeed.item_loader
      ActivityFeed.item_loader.call(feed_item[:member])
    else
      feed_item[:member]
    end

    feed_items << item unless item.nil?
    feed_items
  end

  feed.nil? ? [] : feed
end
```

```
def feed_between_timestamps(user_id, starting_timestamp, ending_timestamp,
aggregate = ActivityFeed.aggregate)
  feederboard = ActivityFeed.feederboard_for(user_id, aggregate)
  feed = feederboard.members_from_score_range(starting_timestamp,
ending_timestamp).inject([]) do |feed_items, feed_item|
    item = if ActivityFeed.item_loader
      ActivityFeed.item_loader.call(feed_item[:member])
    else
      feed_item[:member]
    end

    feed_items << item unless item.nil?
    feed_items
  end

  feed.nil? ? [] : feed
end
```

```
def total_pages_in_feed(user_id, aggregate = ActivityFeed.aggregate,
page_size = ActivityFeed.page_size)
  ActivityFeed.feederboard_for(user_id,
aggregate).total_pages_in(ActivityFeed.feed_key(user_id, aggregate),
page_size)
end

def total_items_in_feed(user_id, aggregate = ActivityFeed.aggregate)
  ActivityFeed.feederboard_for(user_id, aggregate).total_members
end
```

```
def trim_feed(user_id, starting_timestamp, ending_timestamp, aggregate
= ActivityFeed.aggregate)
  ActivityFeed.feederboard_for(user_id,
aggregate).remove_members_in_score_range(starting_timestamp,
ending_timestamp)
end

def expire_feed(user_id, seconds, aggregate = ActivityFeed.aggregate)
  ActivityFeed.redis.expire(ActivityFeed.feed_key(user_id, aggregate),
seconds)
end
```

amico

https://github.com/agoragames/amico

relationships

aka friendships

```
def follow(from_id, to_id, scope = Amico.default_scope_key)
  return if from_id == to_id
  return if blocked?(to_id, from_id, scope)
  return if Amico.pending_follow && pending?(from_id, to_id, scope)

  if Amico.pending_follow
    Amico.redis.multi do |transaction|
      transaction.zadd('#{Amico.namespace}:#{Amico.pending_key}:#{scope}:#{to_id}',
Time.now.to_i, from_id)
      transaction.zadd('#{Amico.namespace}:#{Amico.pending_with_key}:#{scope}:#{from_id}',
Time.now.to_i, to_id)
    end
  else
    add_following_followers_reciprocated(from_id, to_id, scope)
  end
end
```

```
def add_following_followers_reciprocated(from_id, to_id, scope)
  Amico.redis.multi do
    Amico.redis.zadd('#{Amico.namespace}:#{Amico.following_key}:#{scope}:#{from_id}', Time.now.to_i,
to_id)
    Amico.redis.zadd('#{Amico.namespace}:#{Amico.followers_key}:#{scope}:#{to_id}', Time.now.to_i,
from_id)
    Amico.redis.zrem('#{Amico.namespace}:#{Amico.pending_key}:#{scope}:#{to_id}', from_id)
    Amico.redis.zrem('#{Amico.namespace}:#{Amico.pending_with_key}:#{scope}:#{from_id}', to_id)
  end

  if reciprocated?(from_id, to_id)
    Amico.redis.multi do
      Amico.redis.zadd('#{Amico.namespace}:#{Amico.reciprocated_key}:#{scope}:#{from_id}', Time.now.to_i,
to_id)
      Amico.redis.zadd('#{Amico.namespace}:#{Amico.reciprocated_key}:#{scope}:#{to_id}', Time.now.to_i,
from_id)
    end
  end
end
```

```
def unfollow(from_id, to_id, scope = Amico.default_scope_key)
  return if from_id == to_id

  Amico.redis.multi do
    Amico.redis.zrem('#{Amico.namespace}:#{Amico.following_key}:#{scope}:#{from_id}', to_id)
    Amico.redis.zrem('#{Amico.namespace}:#{Amico.followers_key}:#{scope}:#{to_id}', from_id)
    Amico.redis.zrem('#{Amico.namespace}:#{Amico.reciprocated_key}:#{scope}:#{from_id}', to_id)
    Amico.redis.zrem('#{Amico.namespace}:#{Amico.reciprocated_key}:#{scope}:#{to_id}', from_id)
    Amico.redis.zrem('#{Amico.namespace}:#{Amico.pending_key}:#{scope}:#{to_id}', from_id)
    Amico.redis.zrem('#{Amico.namespace}:#{Amico.pending_with_key}:#{scope}:#{from_id}', to_id)
  end
end
```

```
def block(from_id, to_id, scope = Amico.default_scope_key)
  return if from_id == to_id

  Amico.redis.multi do
    Amico.redis.zrem('#{Amico.namespace}:#{Amico.following_key}:#{scope}:#{from_id}', to_id)
    Amico.redis.zrem('#{Amico.namespace}:#{Amico.following_key}:#{scope}:#{to_id}', from_id)
    Amico.redis.zrem('#{Amico.namespace}:#{Amico.followers_key}:#{scope}:#{to_id}', from_id)
    Amico.redis.zrem('#{Amico.namespace}:#{Amico.followers_key}:#{scope}:#{from_id}', to_id)
    Amico.redis.zrem('#{Amico.namespace}:#{Amico.reciprocated_key}:#{scope}:#{from_id}', to_id)
    Amico.redis.zrem('#{Amico.namespace}:#{Amico.reciprocated_key}:#{scope}:#{to_id}', from_id)
    Amico.redis.zrem('#{Amico.namespace}:#{Amico.pending_key}:#{scope}:#{from_id}', to_id)
    Amico.redis.zrem('#{Amico.namespace}:#{Amico.pending_with_key}:#{scope}:#{to_id}', from_id)
    Amico.redis.zadd('#{Amico.namespace}:#{Amico.blocked_key}:#{scope}:#{from_id}', Time.now.to_i, to_id)
    Amico.redis.zadd('#{Amico.namespace}:#{Amico.blocked_by_key}:#{scope}:#{to_id}', Time.now.to_i,
from_id)
  end
end
```

why not use the set operations?

again, think of

redis libraries

as

semantic wrappers

now i'm all (^o^)/

performance

is it web scale? ;)

redis commands give time complexity in big-O notation

and that's awesome, but...

what about some real numbers?

Ruby 1.8.7

Time to rank 10 million people in a leaderboard (sequential scores): 794.1574201583
Time to rank 10 million people in a leaderboard (random scores): 849.301838159561
Average time to retrieve an arbitrary page from the leaderboard (50,000 requests):
0.00165219999999999

Ruby 1.9.3

Time to rank 10 million people in a leaderboard (sequential scores): 651.057383
Time to rank 10 million people in a leaderboard (random scores): 719.157958
Average time to retrieve an arbitrary page from the leaderboard (50,000 requests):
0.00107919999999996

Ruby 1.9.3

Time to rank 10 million people in a leaderboard (sequential scores): 651.057383
Time to rank 10 million people in a leaderboard (random scores): 719.157958
Average time to retrieve an arbitrary page from the leaderboard (50,000 requests):
0.0010791999999999996

Ruby 1.9.3 and hiredis driver

Time to rank 10 million people in a leaderboard (sequential scores): 472.544572
Time to rank 10 million people in a leaderboard (random scores): 549.911350
Average time to retrieve an arbitrary page from the leaderboard (50,000 requests):
0.0003803999999999928

costco coding

“a desk of cheez-its”

```
# ranking 1,000,000 members in a leaderboard individually

insert_time = Benchmark.measure do
  1.upto(1000000) do |index|
    highscore_lb.rank_member('member_#{index}', index)
  end
end
 => 29.340000  15.050000  44.390000 ( 81.673507)
```

```
# ranking 1,000,000 members in a leaderboard in a multi/exec

member_data = []
 => []
1.upto(1000000) do |index|
  member_data << 'member_#{index}'
  member_data << index
end
 => 1
insert_time = Benchmark.measure do
  highscore_lb.rank_members(member_data)
end
 =>  22.390000   6.380000  28.770000 ( 31.144027)
```

failover

let’s go to the zoo

redis_failover

https://github.com/ryanlecompte/redis_failover

redis-sentinel

http://redis.io/topics/sentinel

MONITORING

and

FAILOVER

and

RedisFailover::Client

scripting

extensible redis

redis 2.6 w/ lua scripting

scripts are cached

scripts are atomic

real world redis

strange loop conference

september 23rd-25th, 2012

david czarnecki

https://speakerdeck.com/u/czarneckid/