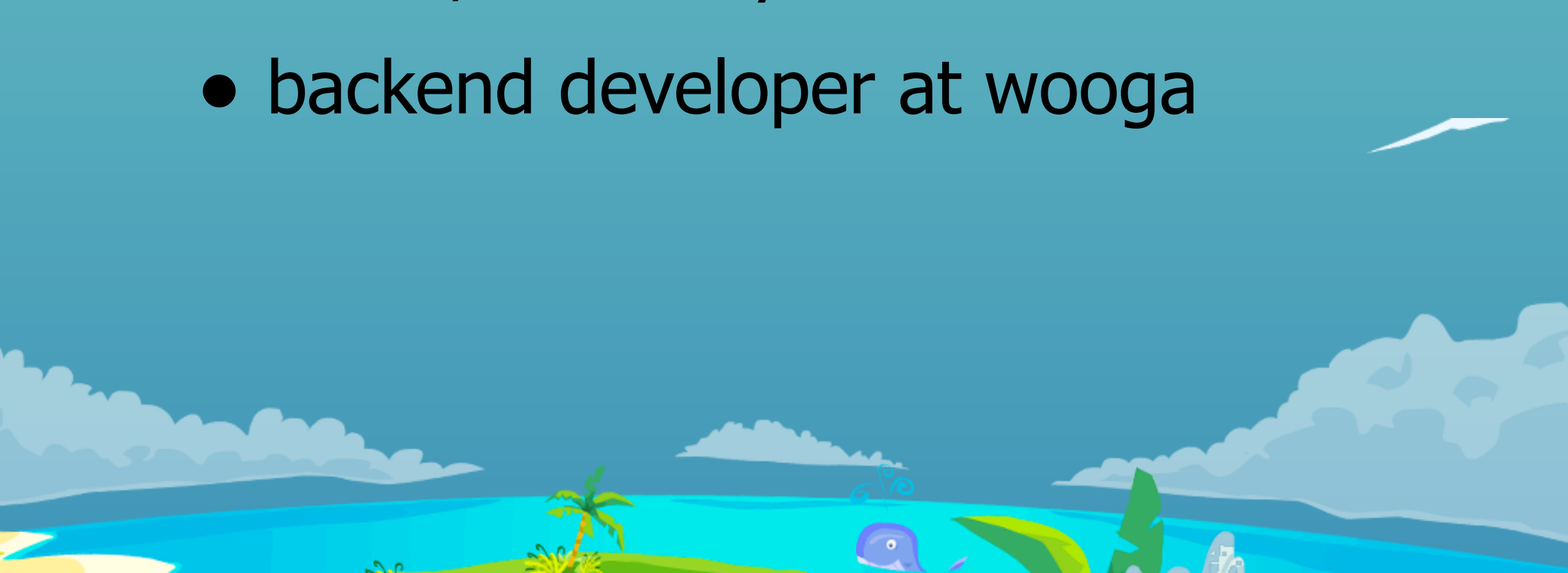
Redis to the Rescue?

O'Reilly MySQL Conference
2011-04-13

Who?

- Tim Lossen / @tlossen
- Berlin, Germany
- backend developer at wooga



wooga – world of gaming

www.wooga.com

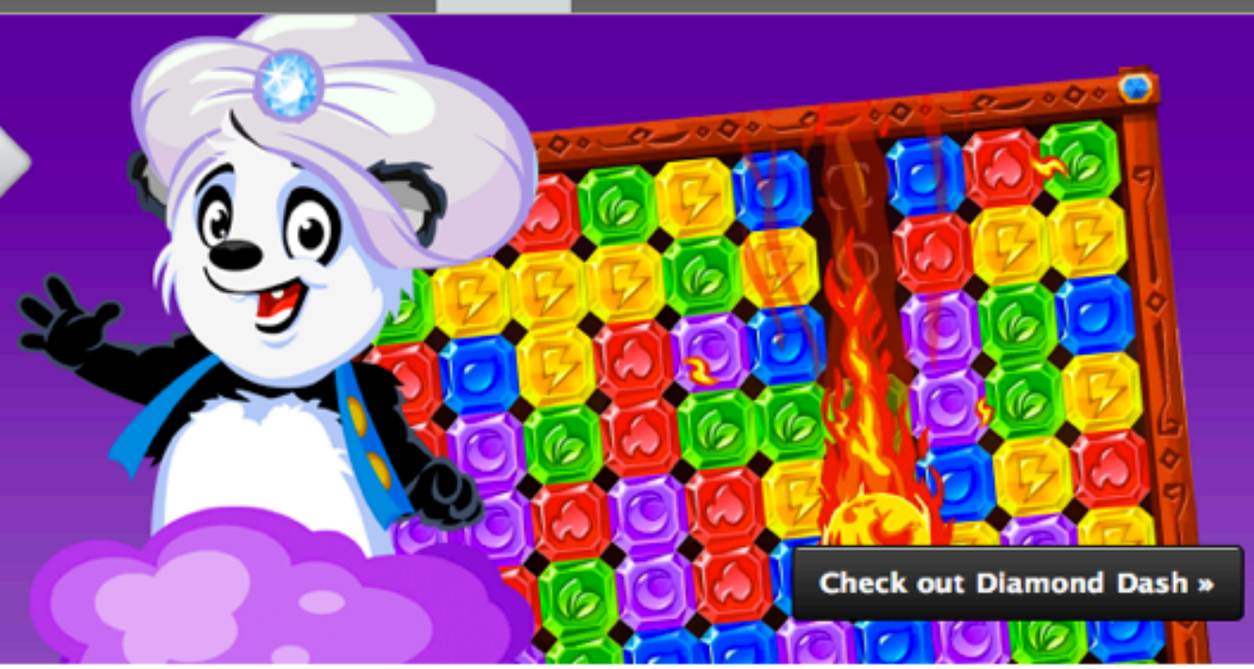
wooga
world of gaming

Home | Games | Jobs | About | Press | Support

Diamond Dash
60 seconds gem rush

wooga is now no. 5 worldwide
over 18 m active users every month

Happy Hospital
Cure cute pets from funny diseases



wooga on Facebook

Like

wooga Due to the slow growth of our games on Facebook we have decided that as of today (2011-04-01) we will completely remove our games Monster World, Bubble Island, Diamond Dash, Happy Hospital and Brain Buddies from Facebook and instead focus all of our efforts on developing games for Myspace

about an hour ago

wooga Peggy Beschnitt sprach heute bei der Munich Gaming über "Dos and Don'ts bei der Bewerbung und dem Vorstellungsgespräch" für Young Game Talents.

20 hours ago

wooga Diamond Dash, wooga's fifth game, launched two weeks ago and already has more then 1,000,000 users: http://goo.gl/ntJsw

Two Weeks, and 1,000,000 Players – wooga's Dash to the Top | wooga

goo.gl

Two numbers that define the wave of explosive success wooga is riding. Diamond Dash, our fifth and newest game, has experienced astonishing growth, capturing

Based in Berlin, wooga is the leading European social games developer.

We are hiring!

Currently, we are searching for:

- Internship Social Media and PR (m/f)
- Game Designer – Bubble Island (m/f) (m/f)
- Software Engineer – Graduate Position (m/f)
- Internship IT Management (m/f)

Redis Intro
Case 1: Monster World
Case 2: Happy Hospital
Discussion

Redis Intro

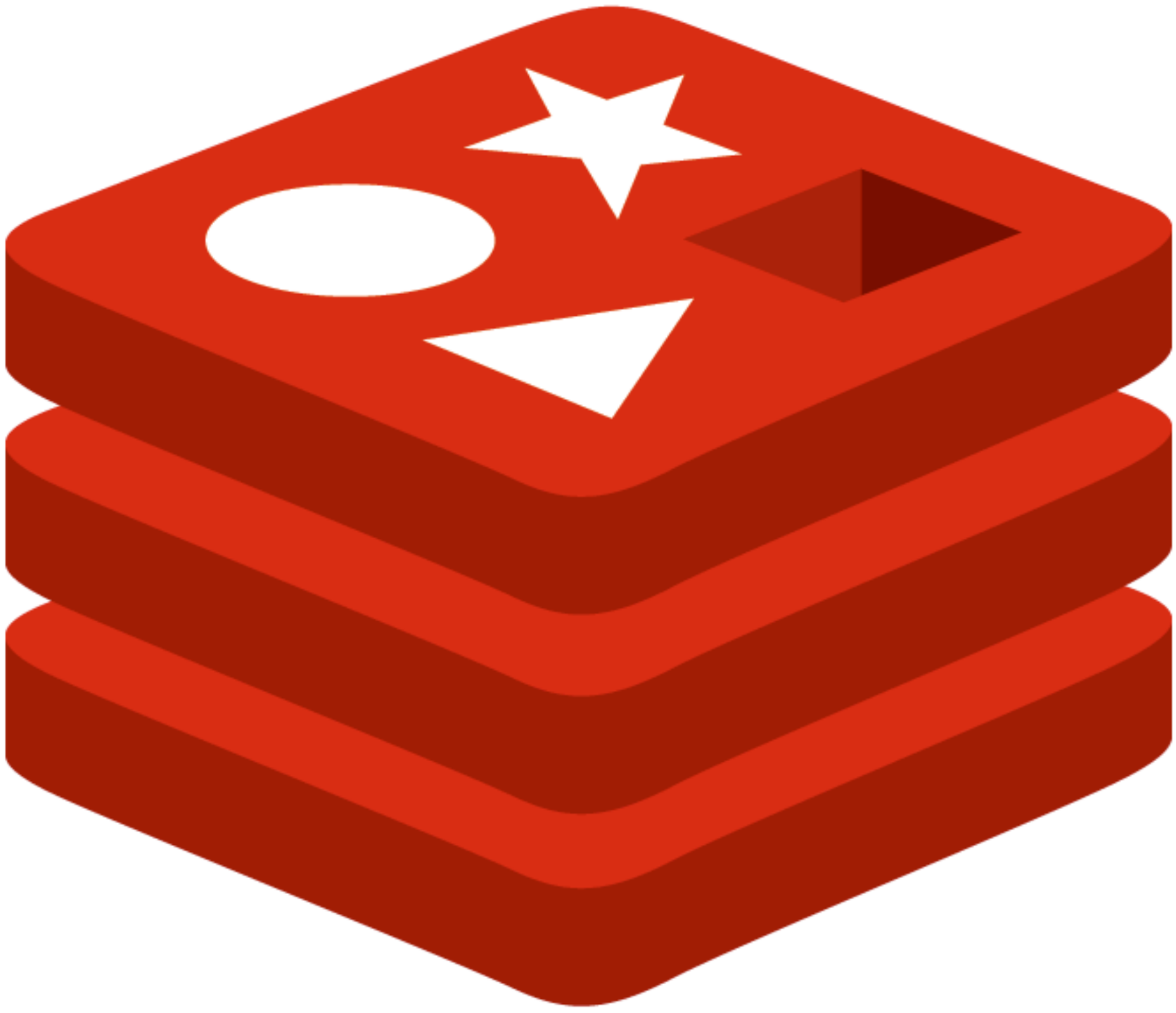
Case 1: Monster World

Case 2: Happy Hospital

Discussion

What?

- key-value-store
- in-memory database
- “data structure server”



Data Types

- strings (integers)

Data Types

- strings (integers)
- lists
- hashes

Data Types

- strings (integers)
- lists
- hashes
- sets
- sorted sets

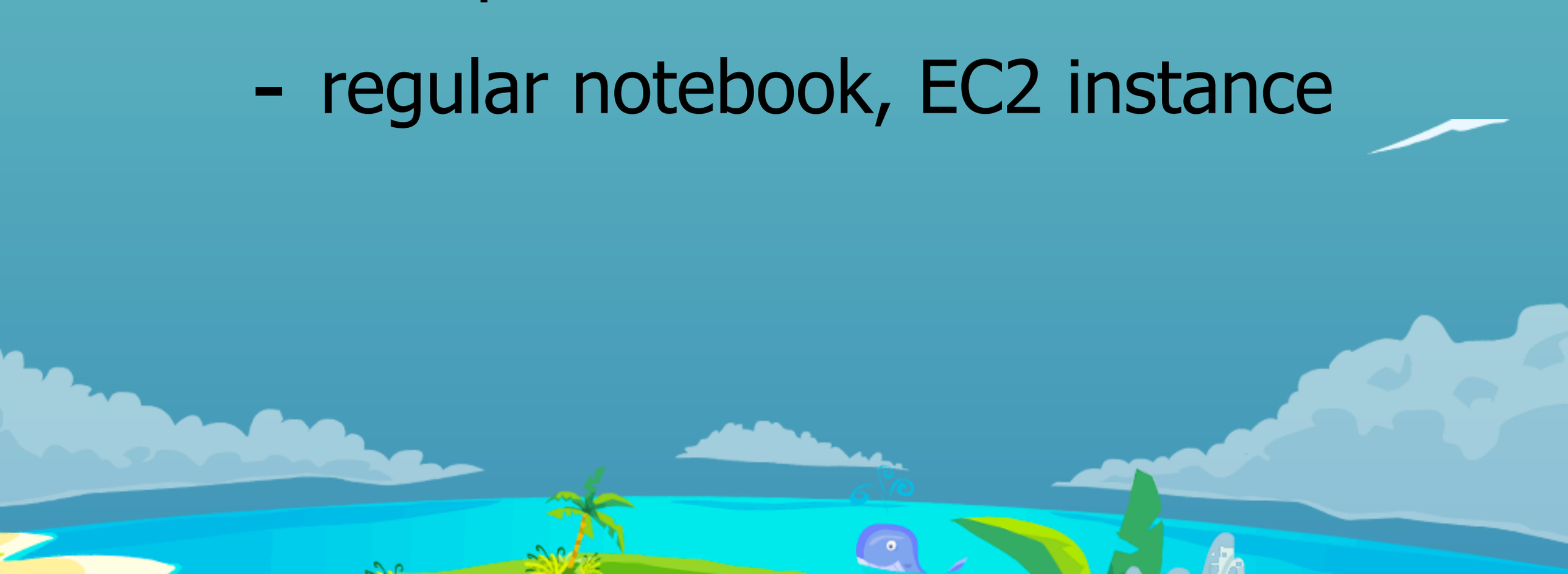

flickr.com/photos/atzu/2645776918

Performance

- same for reads / writes

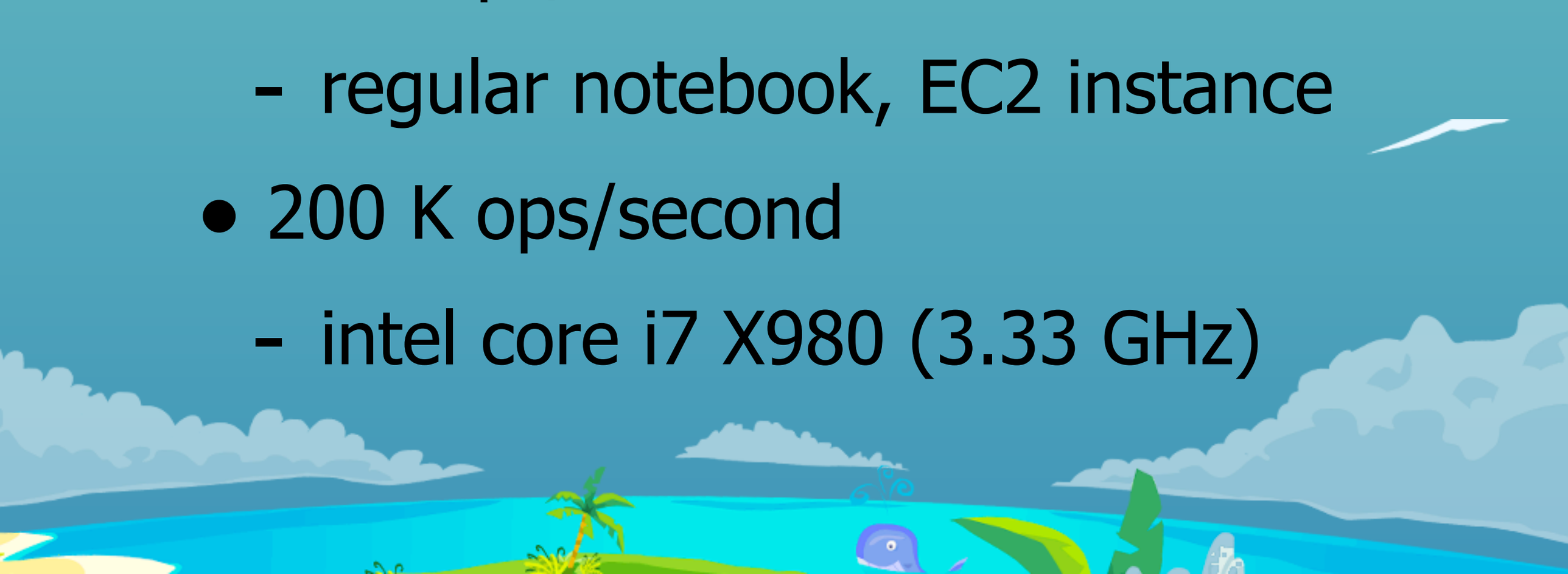
Performance

- same for reads / writes
- 50 K ops/second
  - regular notebook, EC2 instance



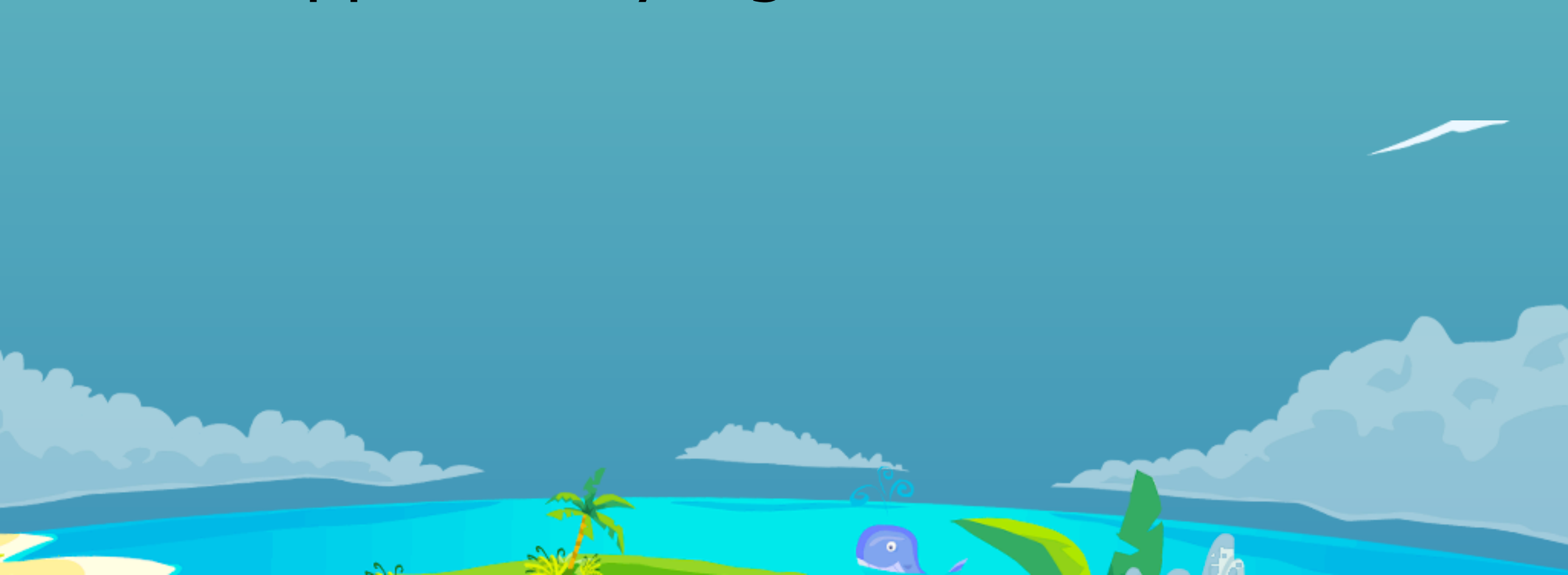
Performance

- same for reads / writes
- 50 K ops/second
  - regular notebook, EC2 instance
- 200 K ops/second
  - intel core i7 X980 (3.33 GHz)



Durability

- snapshots
- append-only log



Other Features

- master-slave replication
- virtual memory
- ...

Redis Intro

Case 1: Monster World

Case 2: Happy Hospital

Discussion

2M people

MonsterWorld

3388

2

Add Coins

Add WooGoo

9

358

Team wooga

00:58:17

Plants

Items

44

3

2

#57 Sebastian 8 464

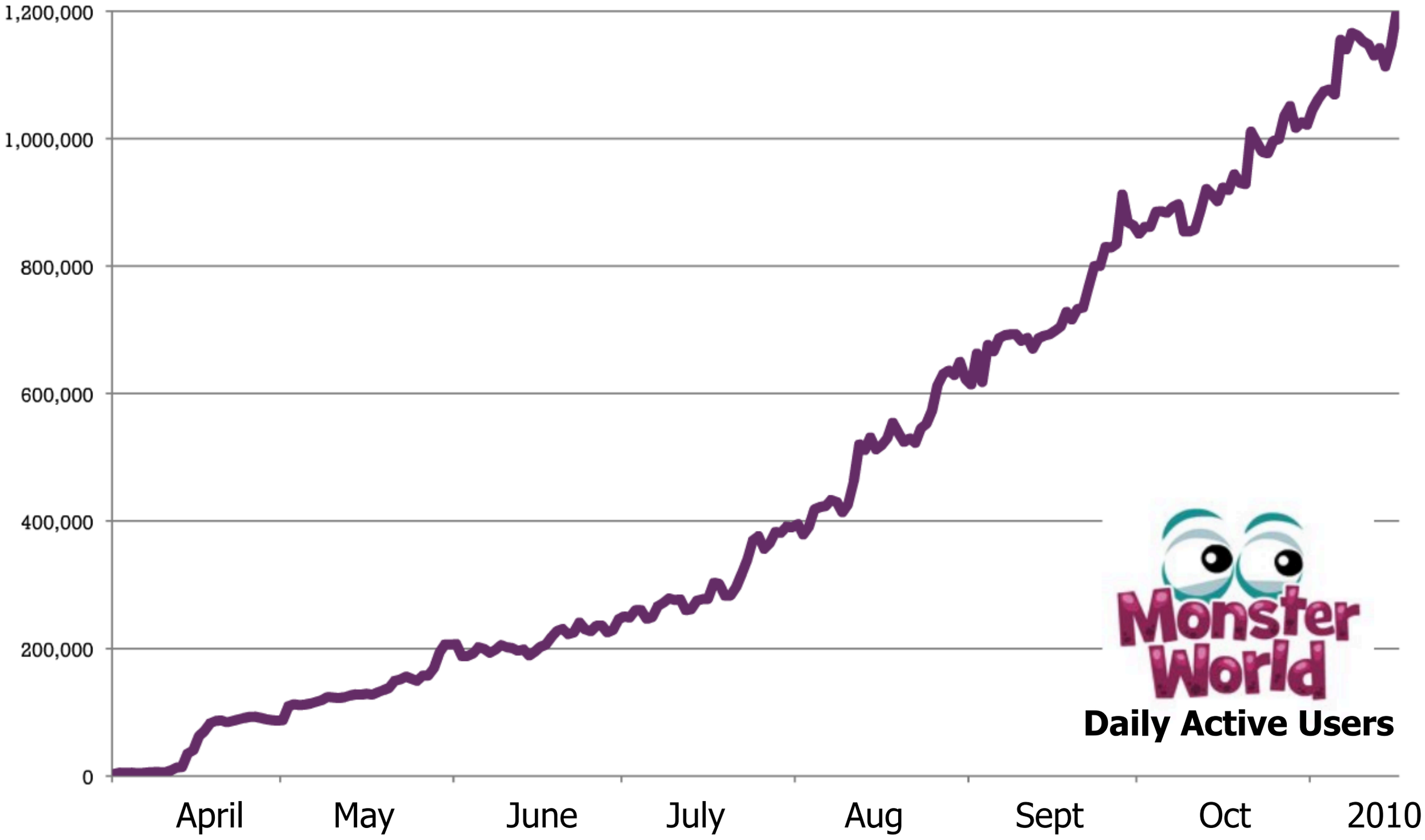
#58 Kim 8 433

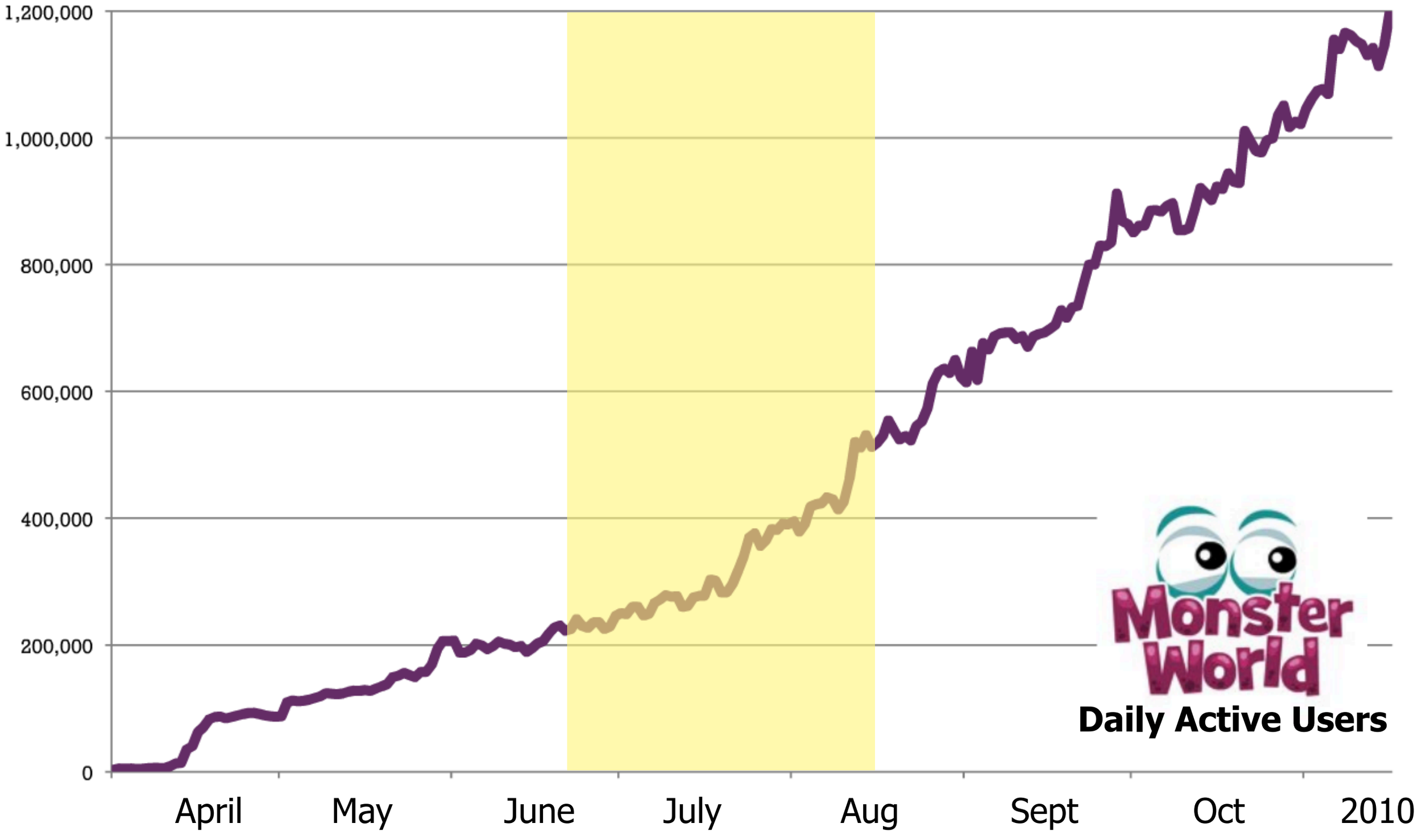
#59 Juha 8 377

#60 Tim 9 358

#61 Steffen 7 357

Invite


1,200,000
1,000,000
800,000
600,000
400,000
200,000
0
April
May
June
July
Aug
Sept
Oct
2010
Monster World
Daily Active Users


1,200,000
1,000,000
800,000
600,000
400,000
200,000
0
April
May
June
July
Aug
Sept
Oct
2010
Monster World
Daily Active Users

Challenge

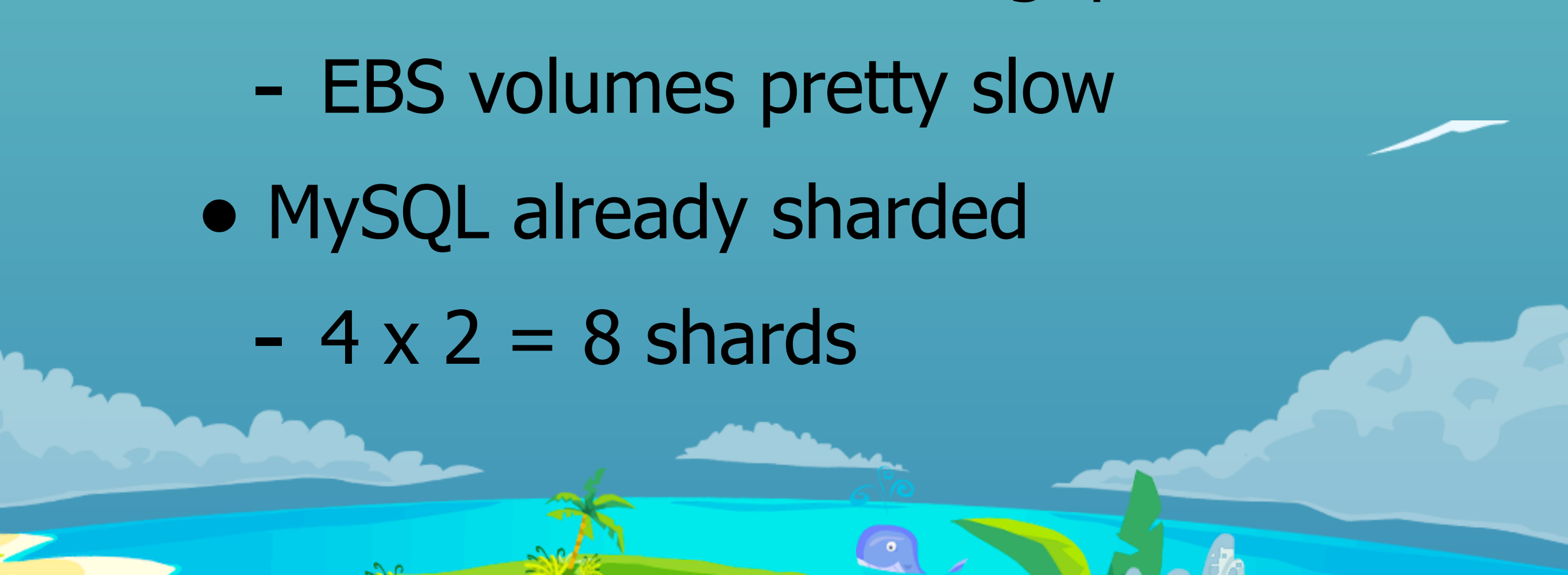
- traffic growing rapidly

Challenge

- traffic growing rapidly
- bottleneck: write throughput
  - EBS volumes pretty slow

Challenge

- traffic growing rapidly
- bottleneck: write throughput
  - EBS volumes pretty slow
- MySQL already sharded
  - 4 x 2 = 8 shards



Idea

- move write-itensive data to Redis

Idea

- move write-itensive data to Redis
- first candidate: inventory
  - integer fields
  - frequently changing

2M people

MonsterWorld

3388

2

Add Coins

Add WooGoo

9

358

Team wooga

00:58:17

Plants

Items

44

3

2

#57 Sebastian 8 464

#58 Kim 8 433

#59 Juha 8 377

#60 Tim 9 358

#61 Steffen 7 357

Invite

Solution

- inventory = Redis hash
  - atomic increment / decrement !

Solution

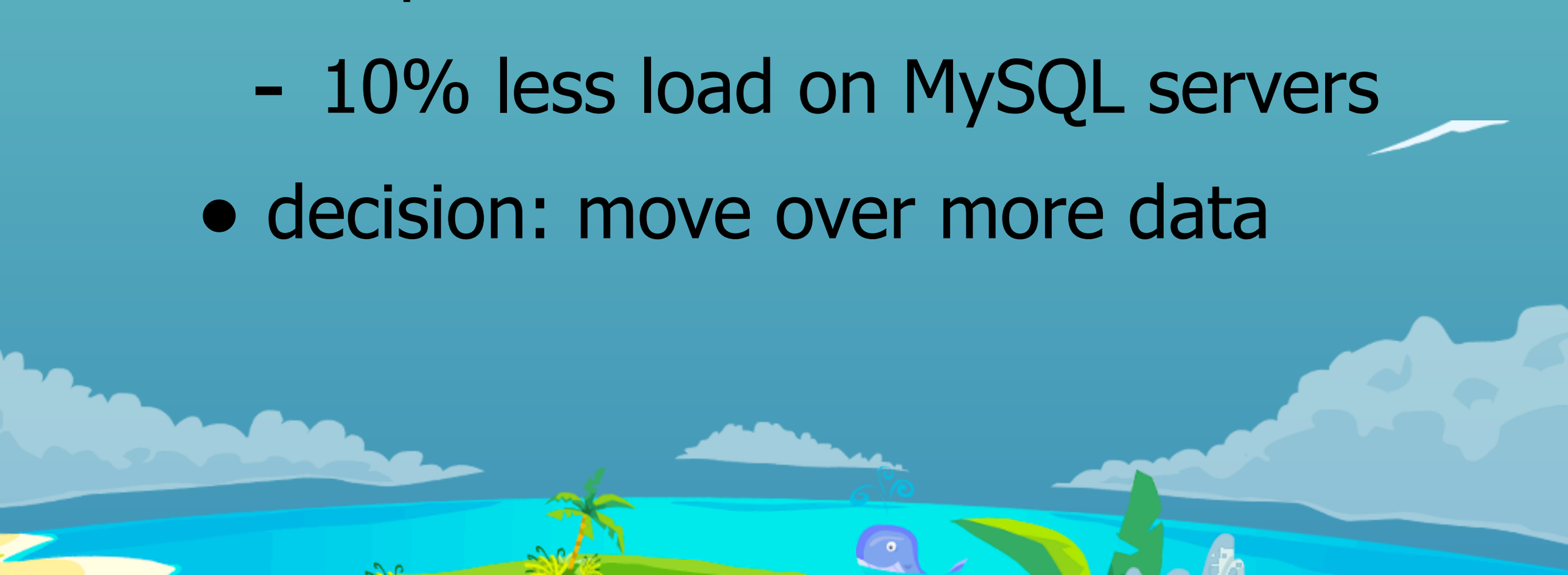
- inventory = Redis hash
  - atomic increment / decrement !
- on-demand migration of users
  - with batch roll-up

Results

- quick win
  - implemented in 2 weeks
  - 10% less load on MySQL servers

Results

- quick win
  - implemented in 2 weeks
  - 10% less load on MySQL servers
- decision: move over more data

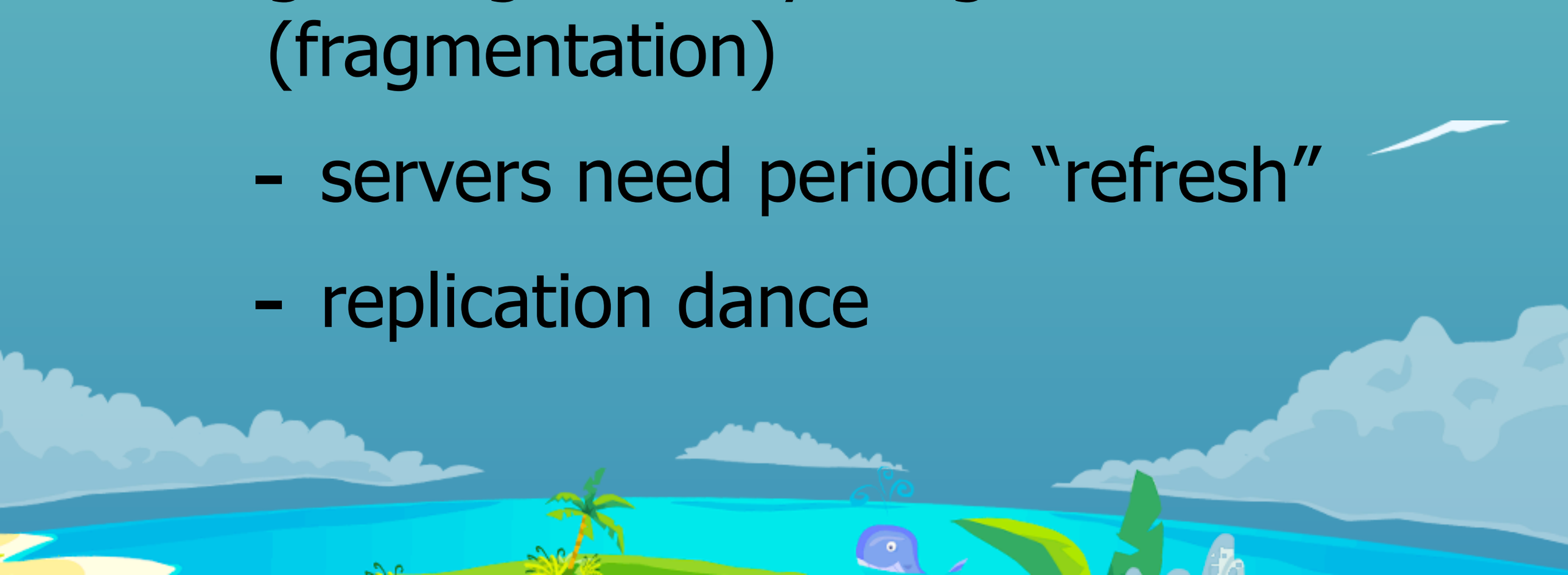


But …

- “honeymoon soon over”

But ...

- "honeymoon soon over"
- growing memory usage (fragmentation)
  - servers need periodic "refresh"
  - replication dance

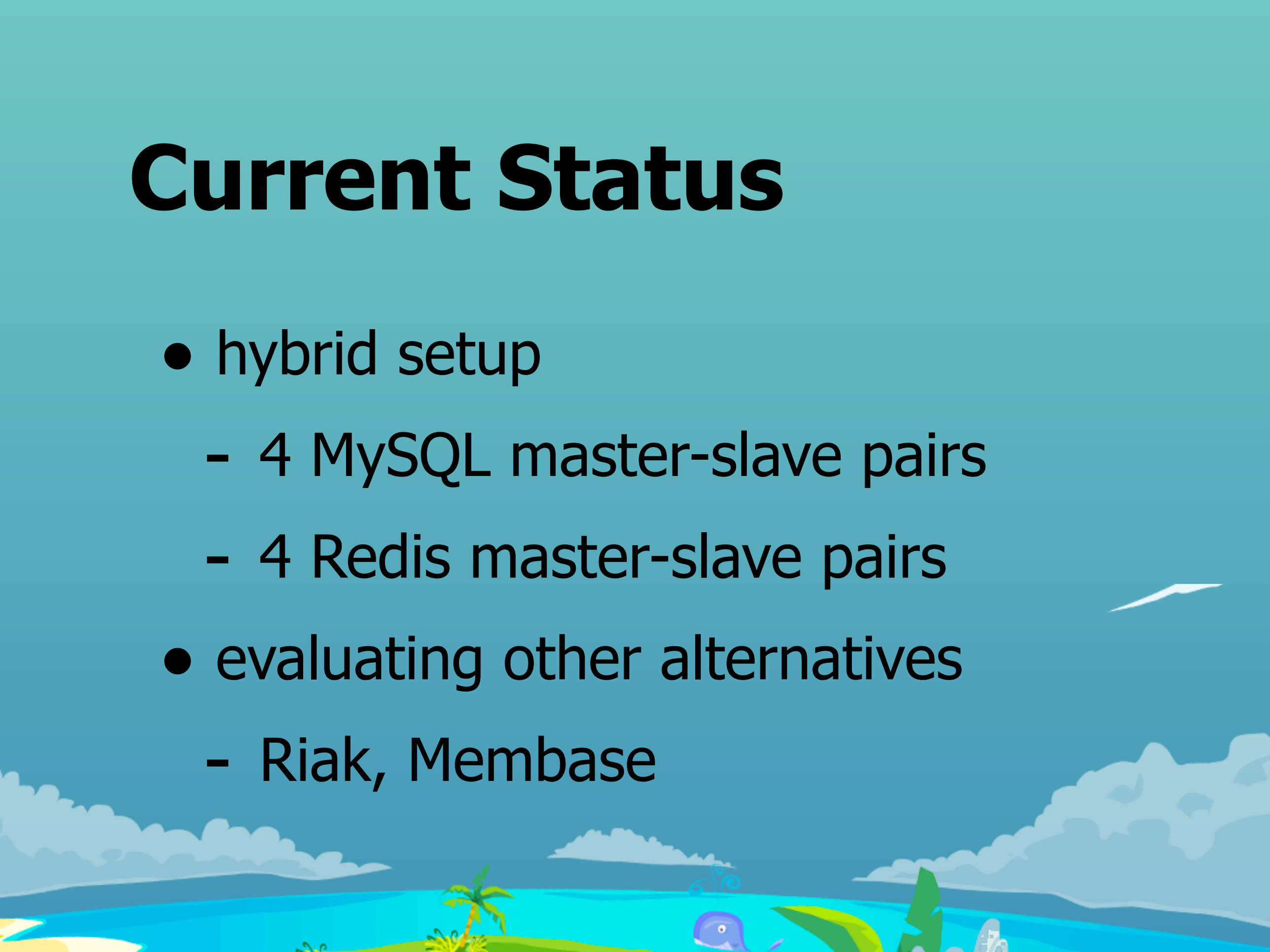


Current Status

- hybrid setup
  - 4 MySQL master-slave pairs
  - 4 Redis master-slave pairs

Current Status

- hybrid setup
  - 4 MySQL master-slave pairs
  - 4 Redis master-slave pairs
- evaluating other alternatives
  - Riak, Membase



Redis Intro

Case 1: Monster World

Case 2: Happy Hospital

Discussion

874

33/14

19

7519

134/200

POWER SLOT

Friends

Pets

Doctors

Inventory

19 7519 Tim

19 7456 Peggy

19 7255 Stefan

19 7133 Alena

Invite friend

18 6479 Anabel

29

10

51

4

17

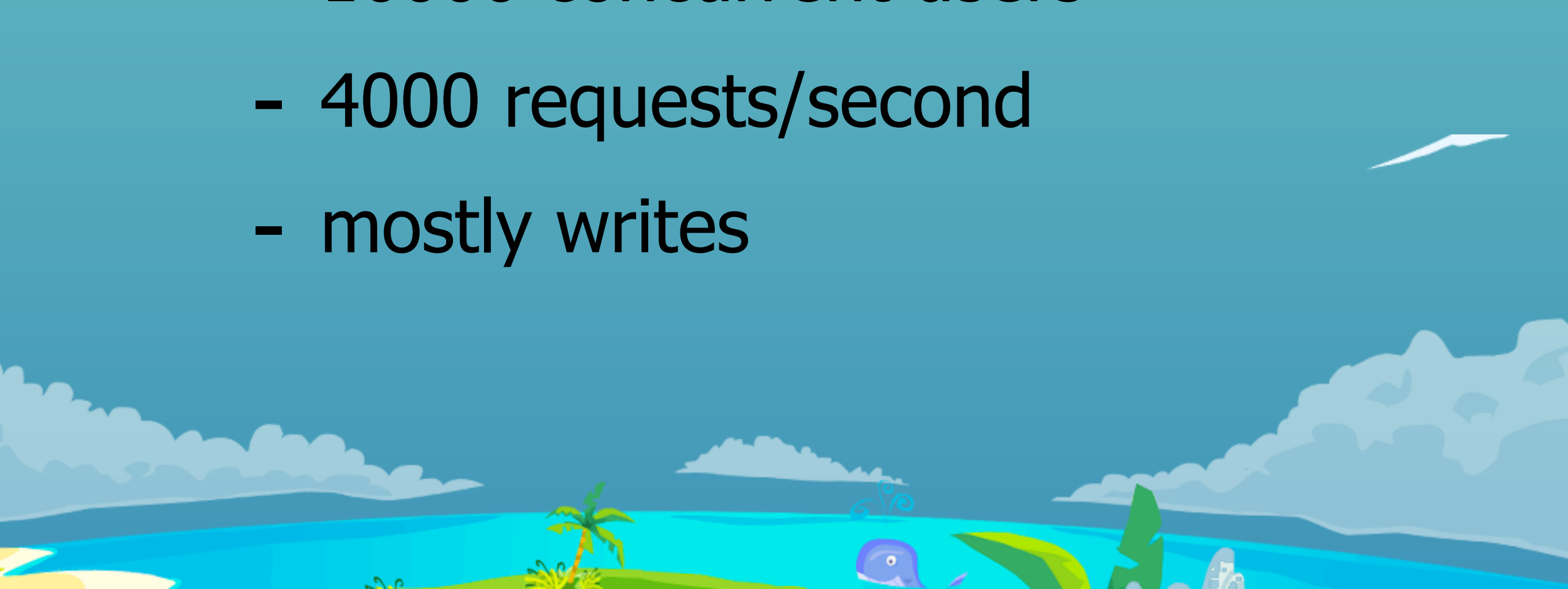
4

1

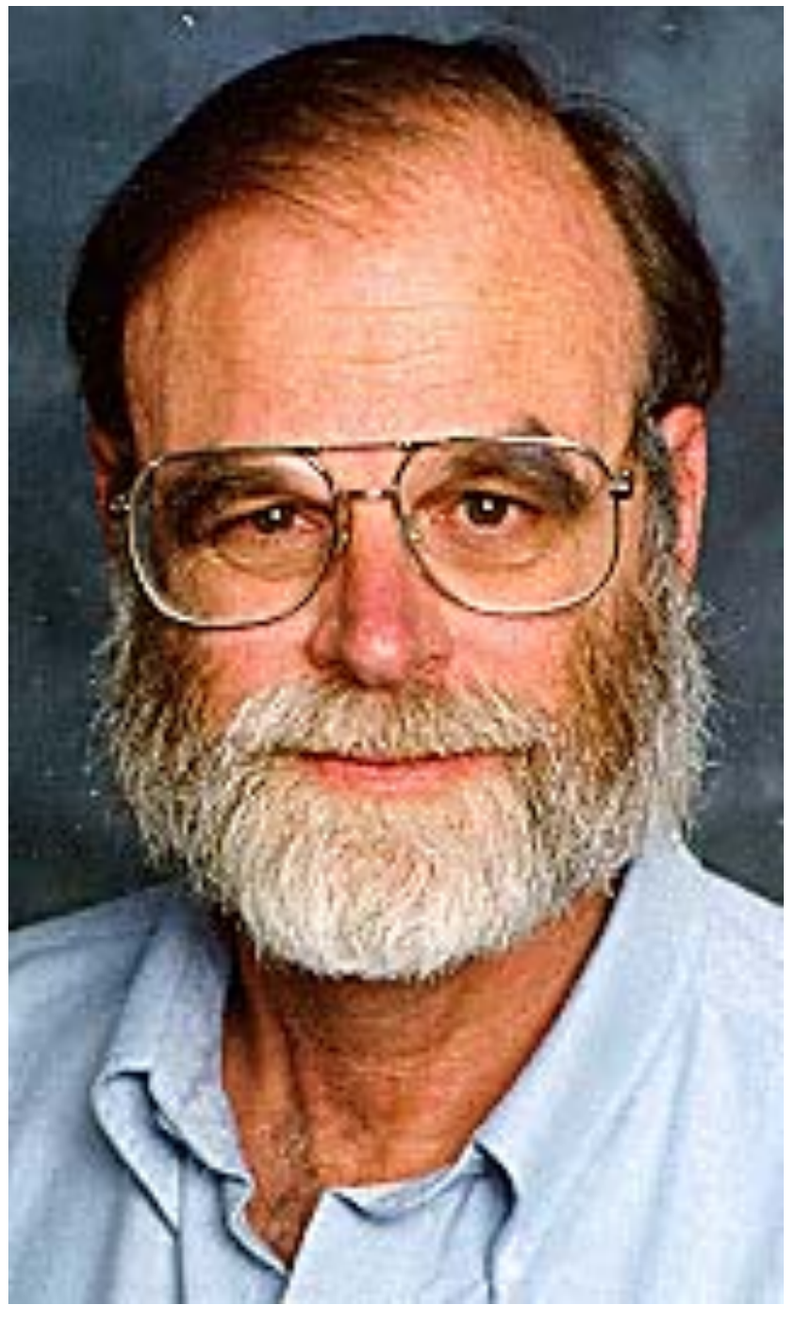
4

Challenge

- expected peak load:
  - 16000 concurrent users
  - 4000 requests/second
  - mostly writes

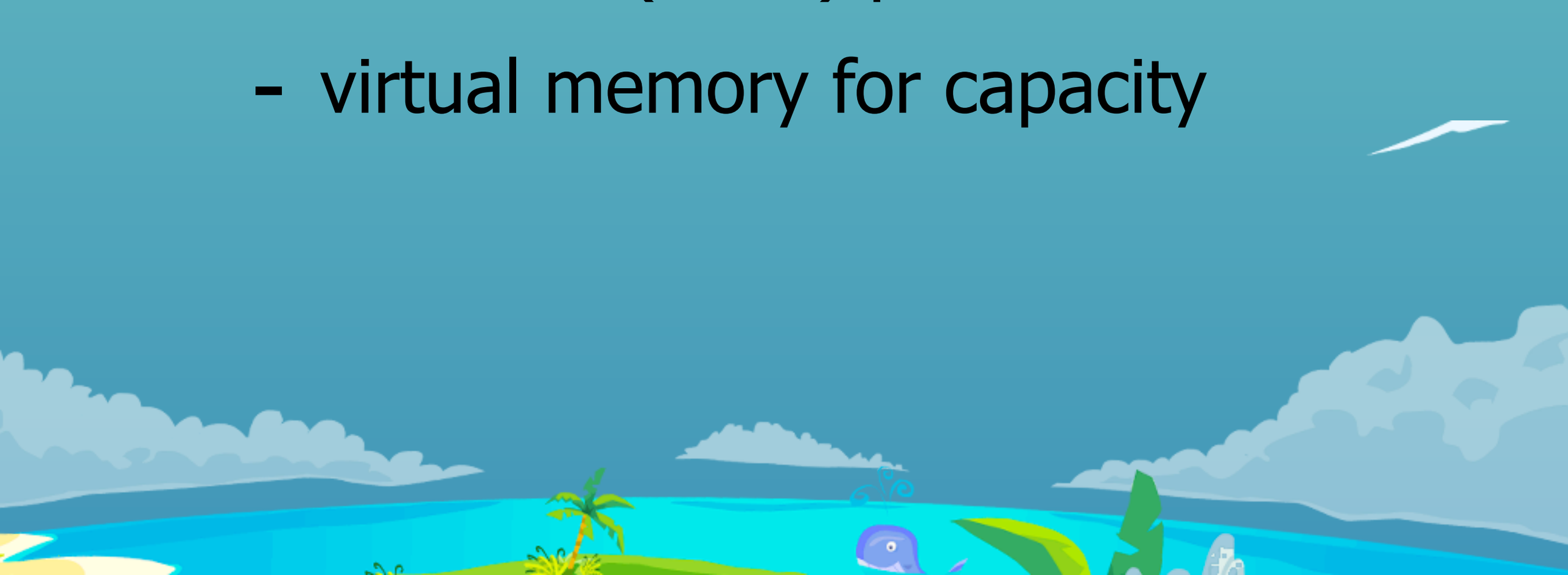


"Memory is the new Disk,
Disk is the new Tape."
— Jim Gray



Idea

- use Redis as main database
  - excellent (write) performance
  - virtual memory for capacity



Idea

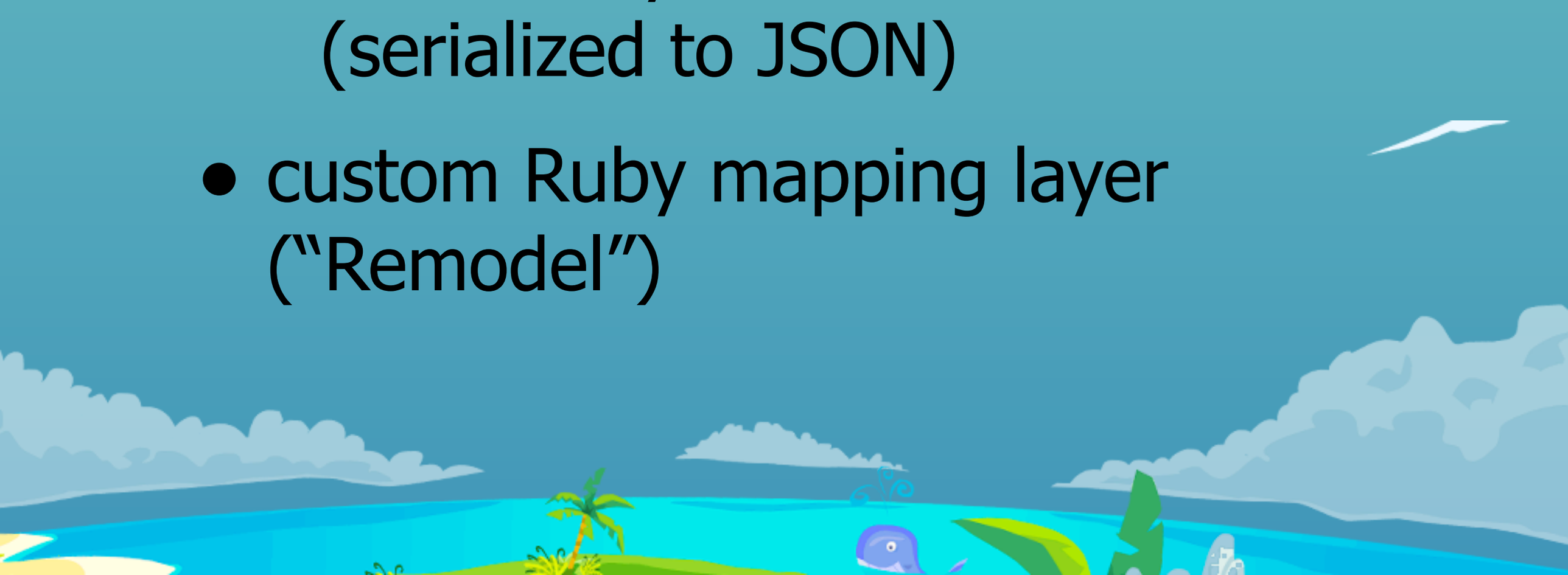
- use Redis as main database
  - excellent (write) performance
  - virtual memory for capacity
- no sharding = simple operations

Data Model

- user = single Redis hash
  - each entity stored in hash field (serialized to JSON)

Data Model

- user = single Redis hash
  - each entity stored in hash field (serialized to JSON)
- custom Ruby mapping layer ("Remodel")



example.rb

```ruby
class User < Remodel::Entity

  has_many :pets, :class => Pet

  property :level, :class => Integer, :default => 1
  property :xp, :class => Integer, :default => 0

end


class Pet < Remodel::Entity

  property :pet_type, :class => String

end
```

1220032045	ul	{“level”: 4, “xp”: 241}
	ul_pets	[“p7”, “p8”]
	p7	{“pet_type”: “Cat”}
	p8	{“pet_type”: “Dog”}
1234599660	ul	{“level”: 1, “xp”: 22}
	ul_pets	[“p3”]
	...	...

1220032045	ul	{“level”: 4, “xp”: 241}
	ul_pets	[“p7”, “p8”]
	p7	{“pet_type”: “Cat”}
	p8	{“pet_type”: “Dog”}
1234599660	ul	{“level”: 1, “xp”: 22}
	ul_pets	[“p3”]
	...	...

1220032045	ul	{“level”: 4, “xp”: 241}
	ul_pets	[“p7”, “p8”]
	p7	{“pet_type”: “Cat”}
	p8	{“pet_type”: “Dog”}
1234599660	ul	{“level”: 1, “xp”: 22}
	ul_pets	[“p3”]
	...	...

1220032045	ul	{“level”: 4, “xp”: 241}
	ul_pets	[“p7”, “p8”]
	p7	{“pet_type”: “Cat”}
	p8	{“pet_type”: “Dog”}
1234599660	ul	{“level”: 1, “xp”: 22}
	ul_pets	[“p3”]
	...	...

1220032045	ul	{“level”: 4, “xp”: 241}
	ul_pets	[“p7”, “p8”]
	p7	{“pet_type”: “Cat”}
	p8	{“pet_type”: “Dog”}
1234599660	ul	{“level”: 1, “xp”: 22}
	ul_pets	[“p3”]
	...	...

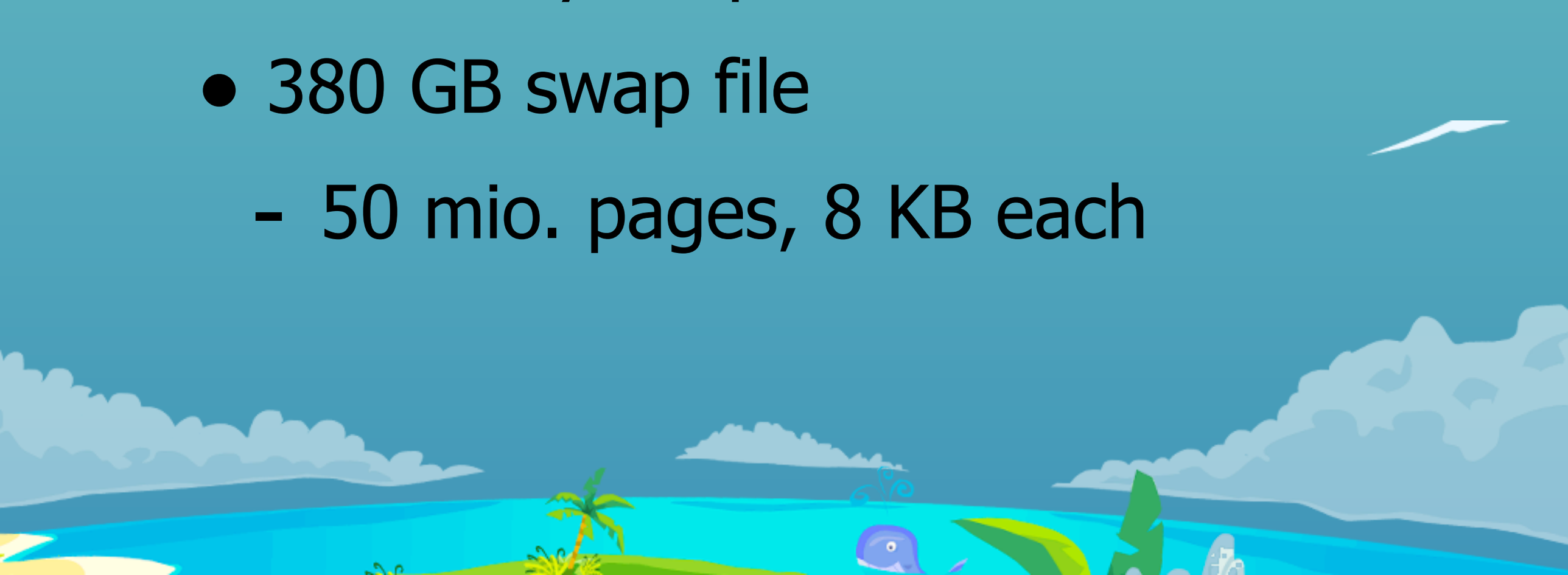
1220032045	ul	{“level”: 4, “xp”: 241}
	ul_pets	[“p7”, “p8”]
	p7	{“pet_type”: “Cat”}
	p8	{“pet_type”: “Dog”}
1234599660	ul	{“level”: 1, “xp”: 22}
	ul_pets	[“p3”]
	...	...

Virtual Memory

- server: 24 GB RAM, 500 GB disk
  - can only keep “hot data” in RAM

Virtual Memory

- server: 24 GB RAM, 500 GB disk
  - can only keep “hot data” in RAM
- 380 GB swap file
  - 50 mio. pages, 8 KB each



Dec 2010: Crisis

- memory usage growing fast

Dec 2010: Crisis

- memory usage growing fast
- cannot take snapshots any more
  - cannot start new slaves

Dec 2010: Crisis

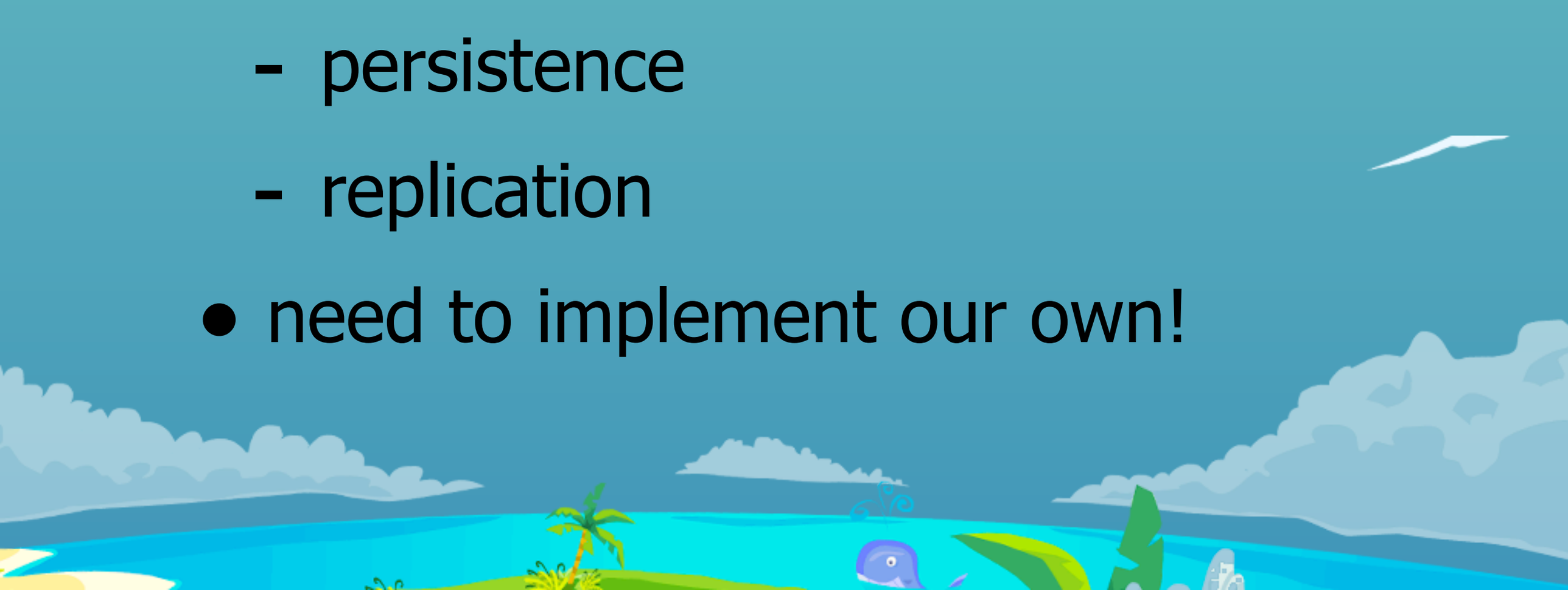
- memory usage growing fast
- cannot take snapshots any more
  - cannot start new slaves
- random crashes

Analysis

- Redis virtual memory not compatible with:
  - persistence
  - replication

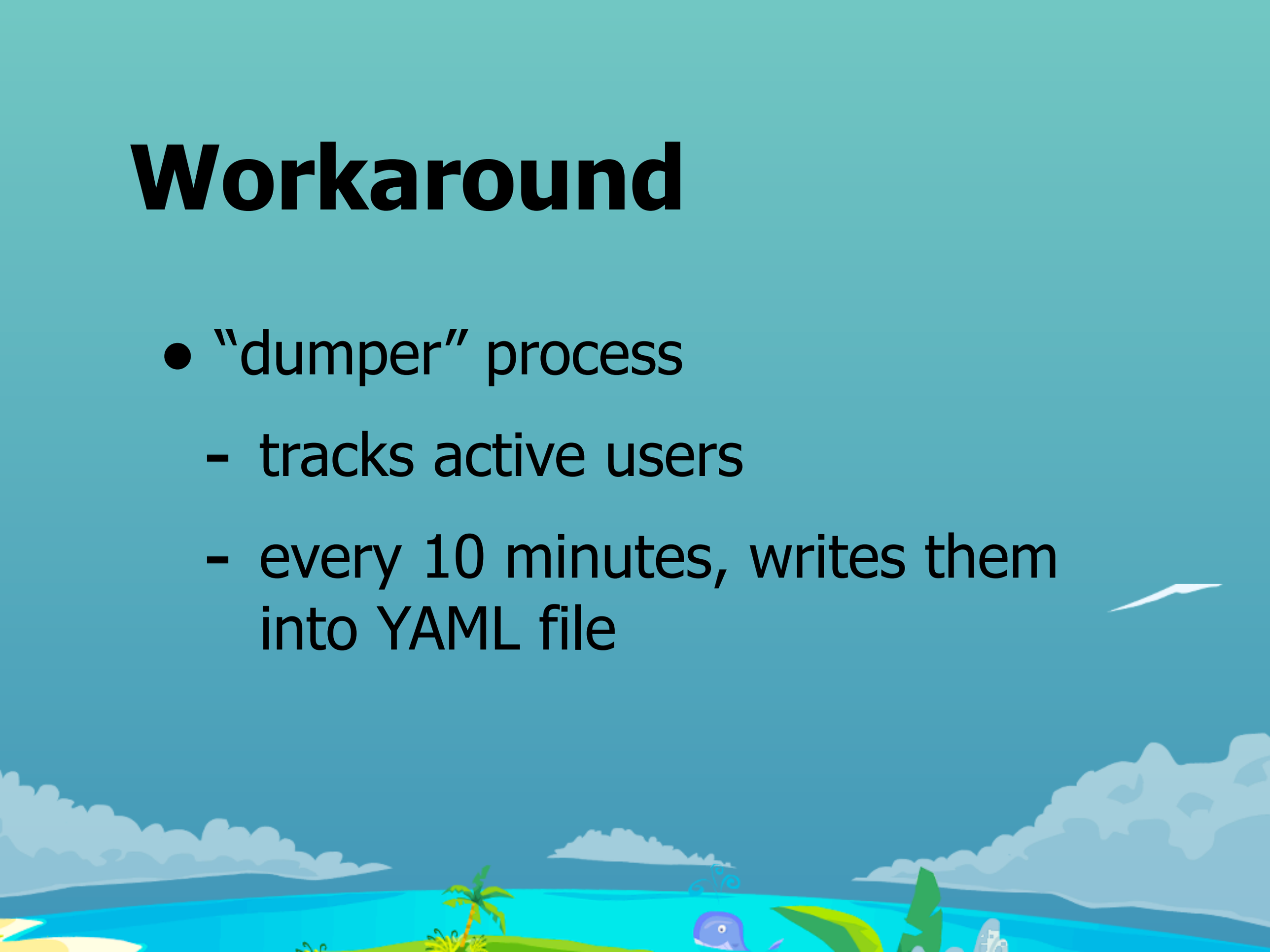
Analysis

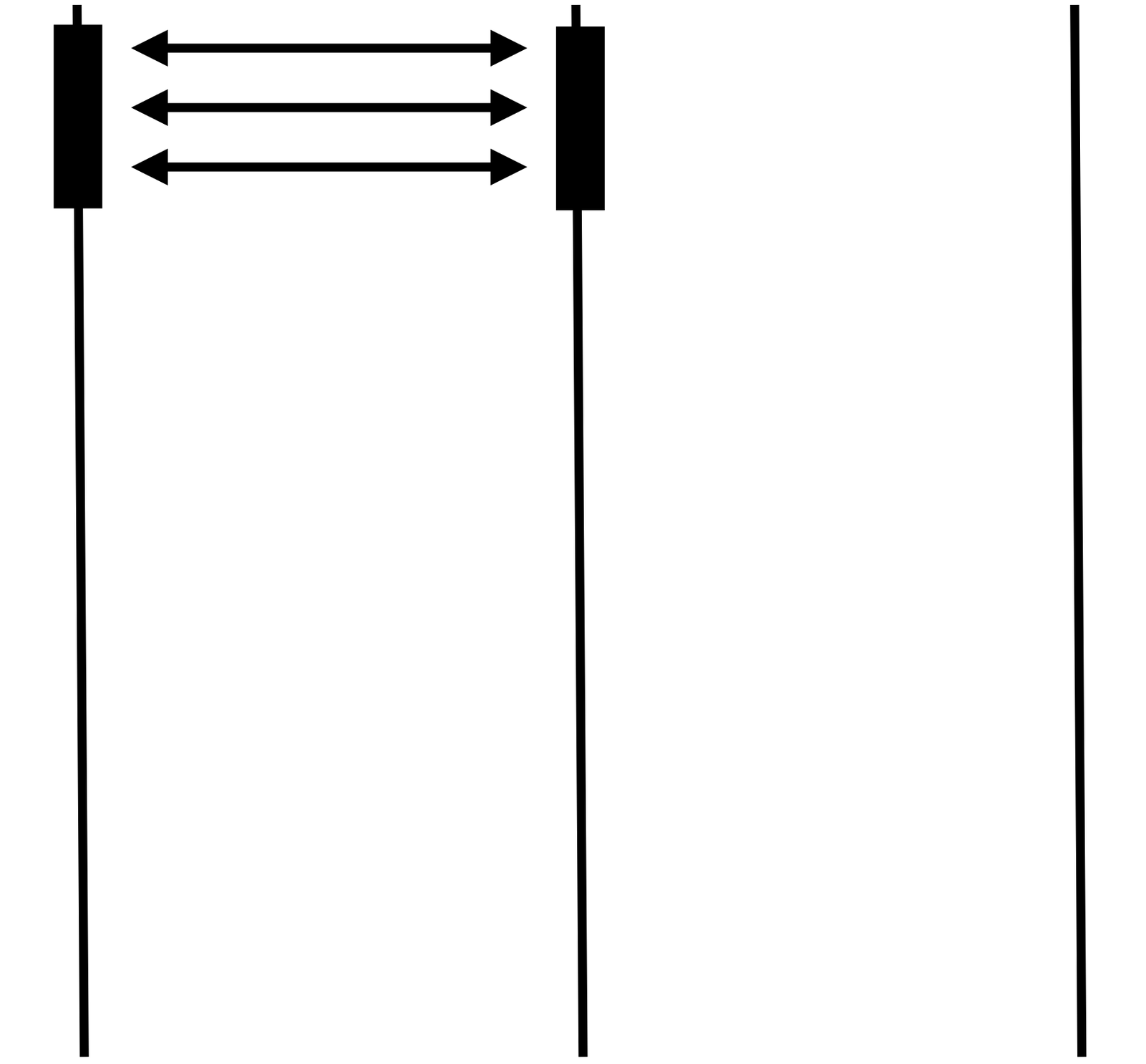
- Redis virtual memory not compatible with:
  - persistence
  - replication
- need to implement our own!

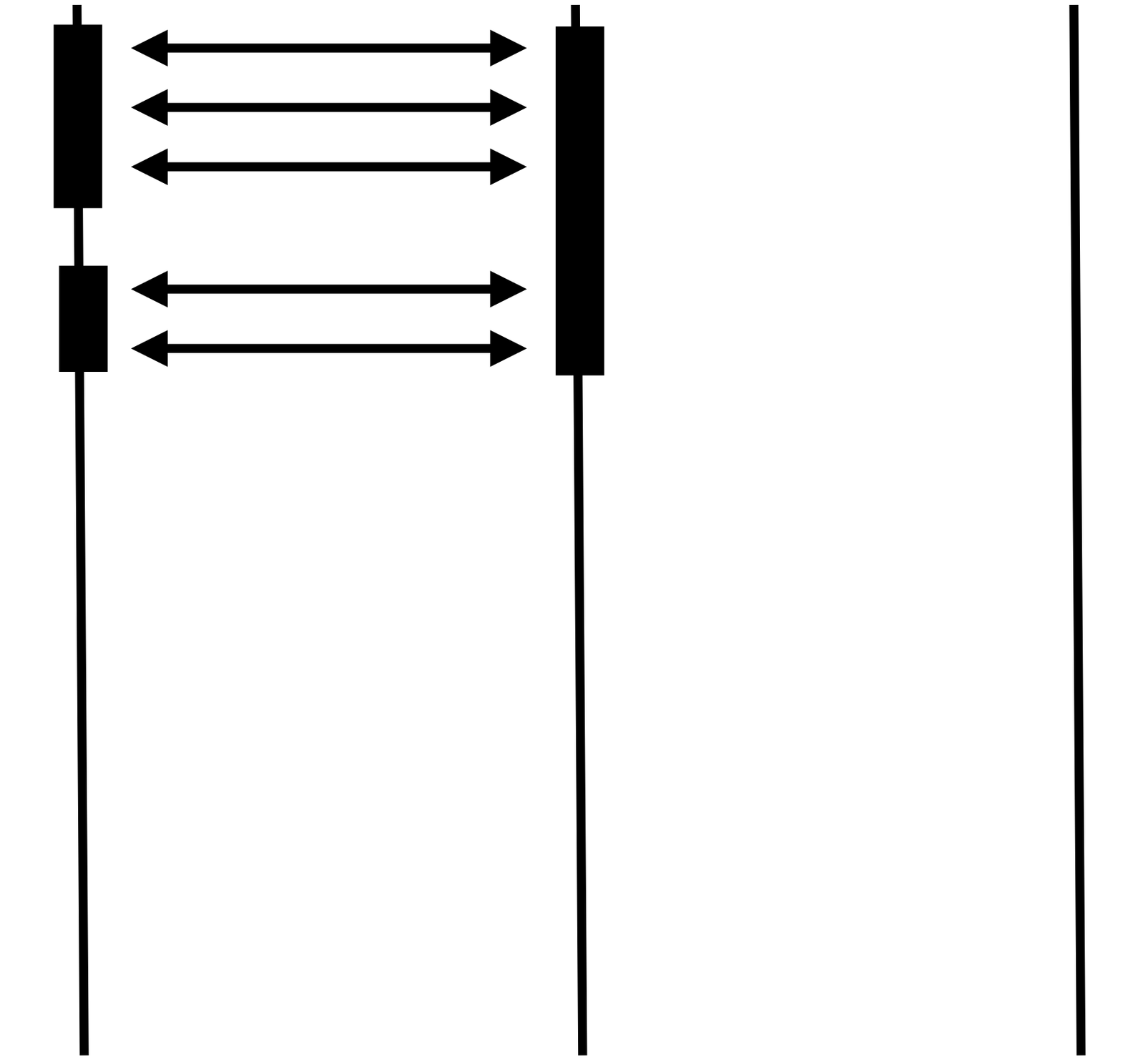


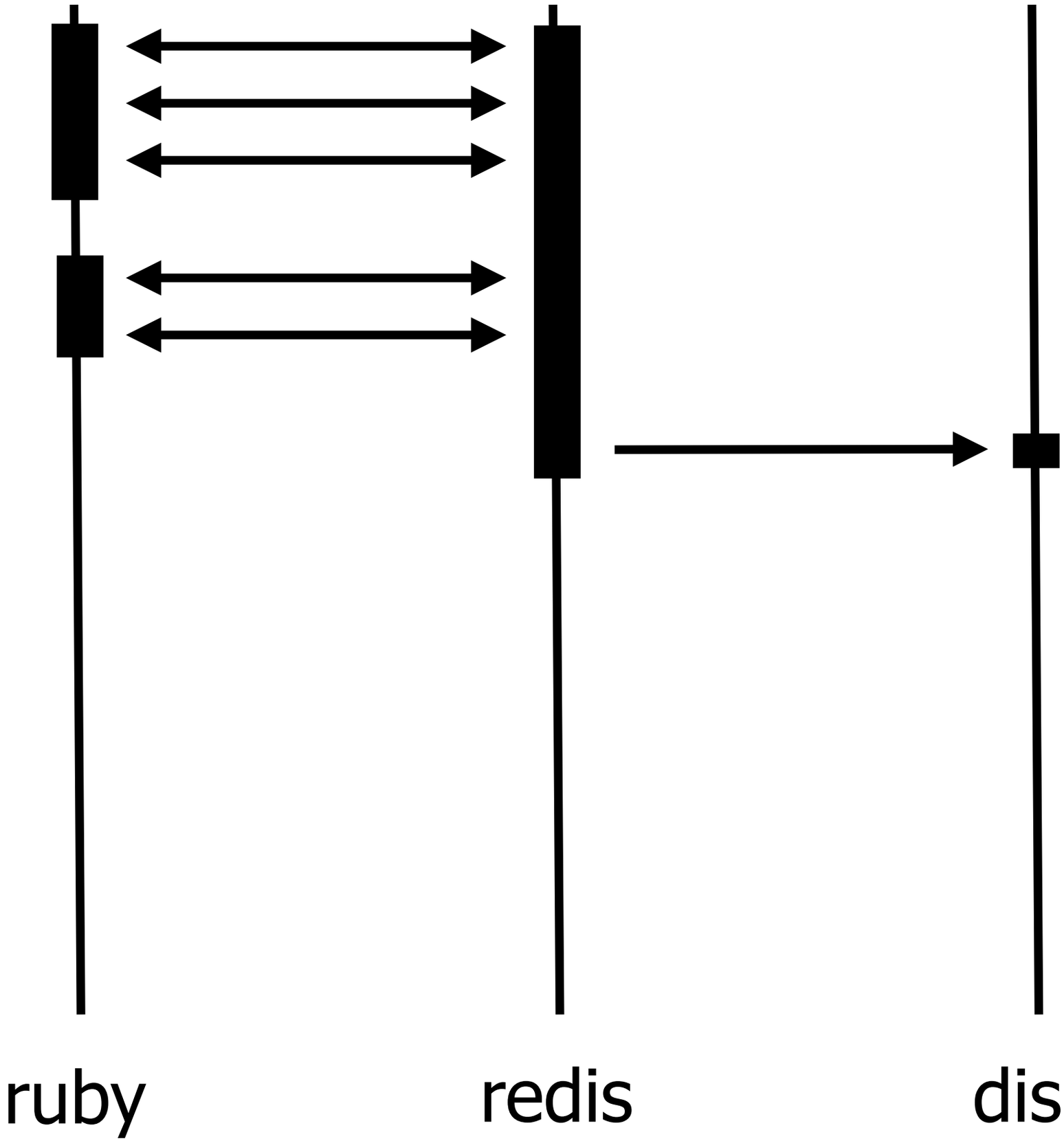
Workaround

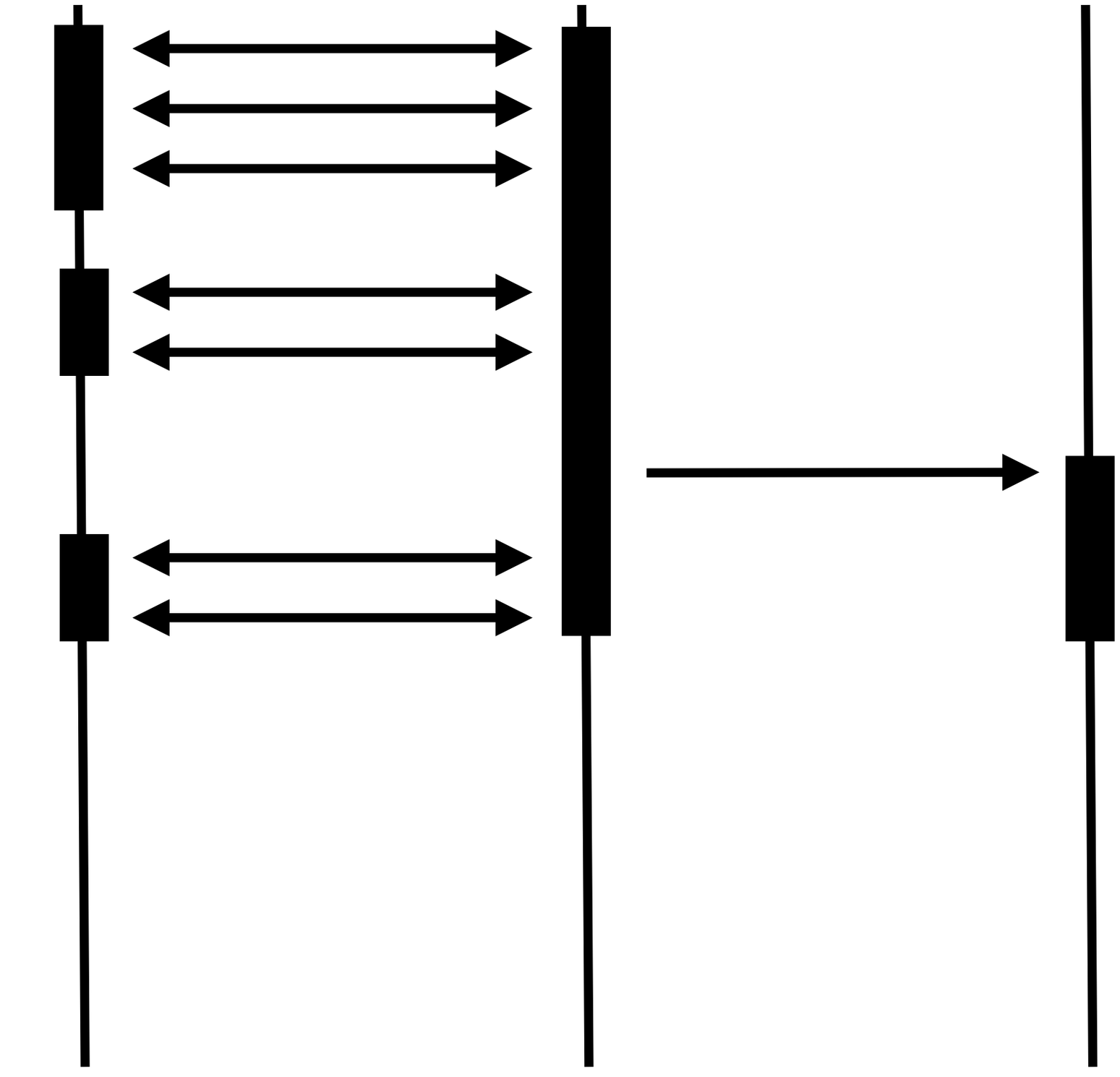
- "dumper" process
  - tracks active users
  - every 10 minutes, writes them into YAML file

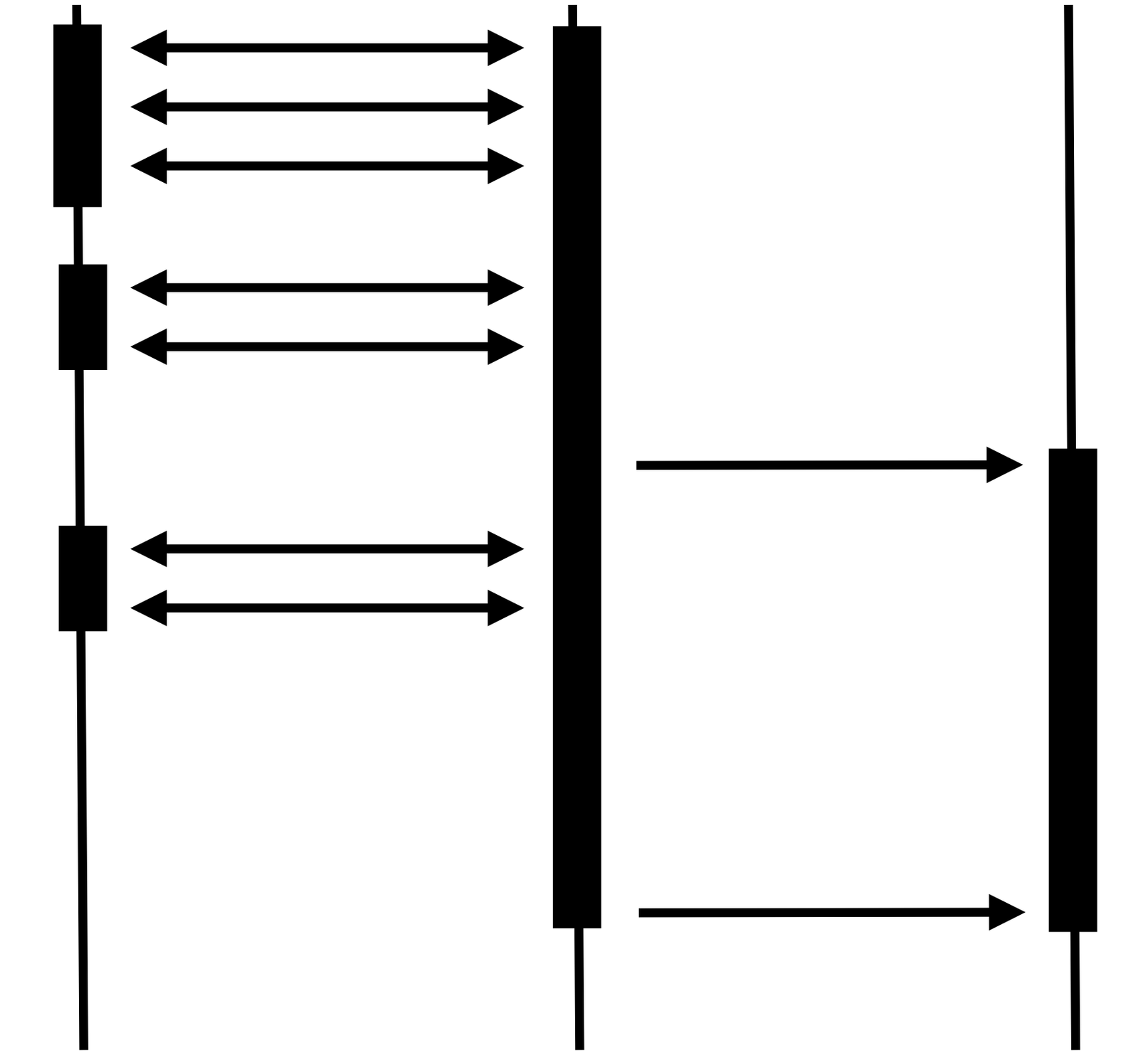



ruby
redis
disk


ruby
redis
disk

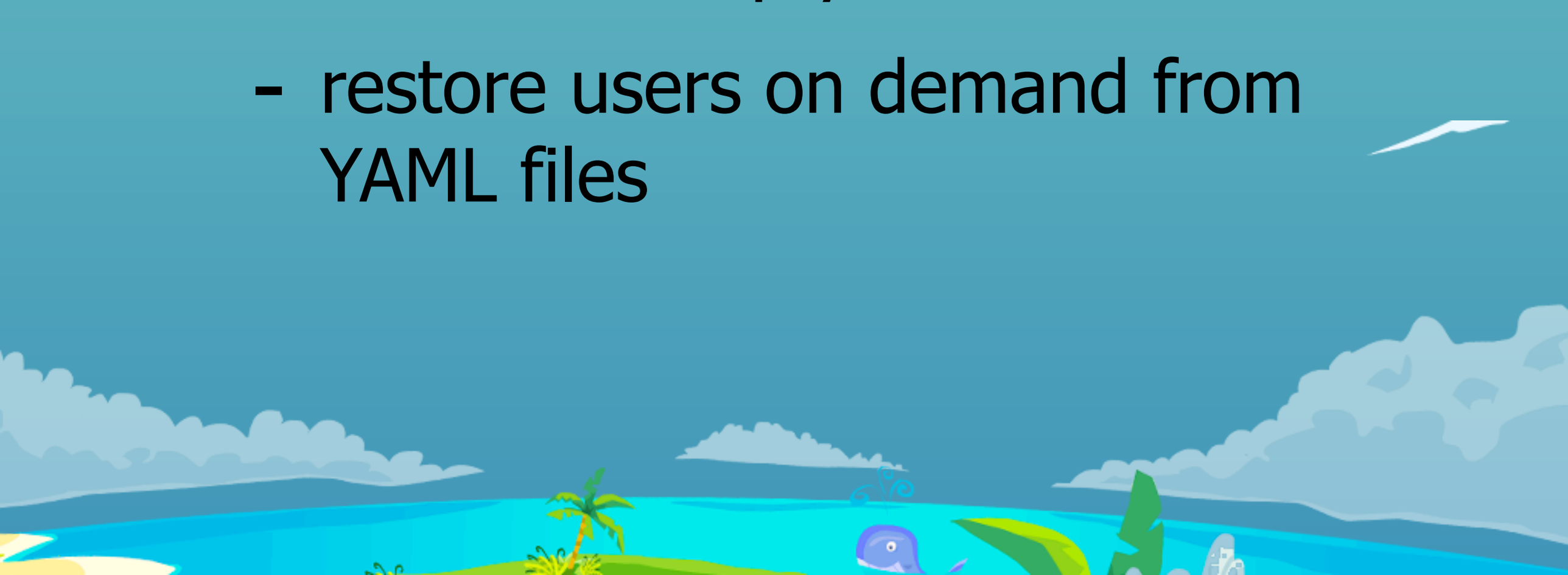

ruby
redis
disk


ruby
redis
disk


ruby
redis
disk

Workaround

- in case of Redis crash
  - start with empty database
  - restore users on demand from YAML files



Real Solution

- Redis "diskstore"
  - keeps all data on disk
  - swaps data into memory as needed

Real Solution

- Redis "diskstore"
  - keeps all data on disk
  - swaps data into memory as needed
- under development (expected Q2)

```
uptime_in_days:102
total_commands_processed:54428739517
vm_stats_swappin_count:73139647
vm_stats_swappout_count:77343129
db2:keys=4296618,expires=0
```

Results

- average response time: **10 ms**

Results

- average response time: **10 ms**
- peak traffic:
  - 1500 requests/second
  - 15000 Redis ops/second

Current Status

- very happy with setup
  - simple, robust, fast
  - easy to operate
- still lots of spare capacity

Redis Intro
Case 1: Monster World
Case 2: Happy Hospital

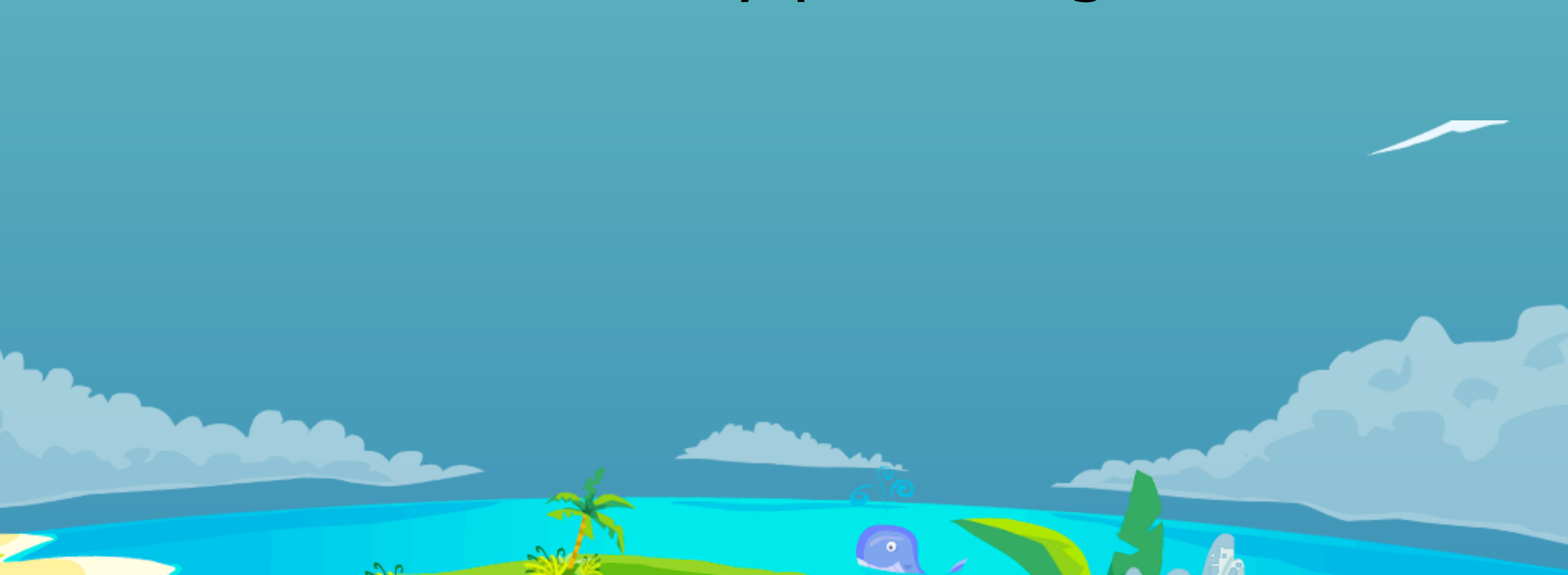
Discussion

Advantages

- order-of-magnitude performance improvement
  - removes main bottleneck
  - enables simple architecture

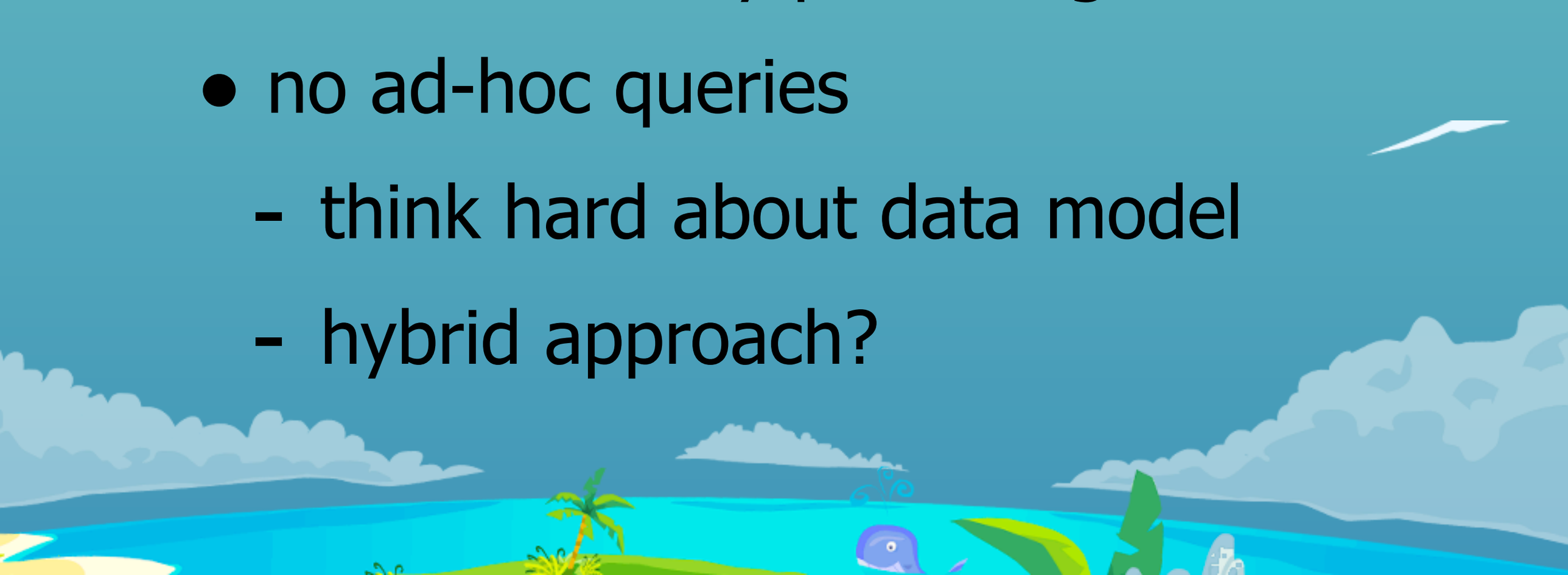
Disadvantages

- main challenge: durability
  - diskstore very promising



Disadvantages

- main challenge: durability
  - diskstore very promising
- no ad-hoc queries
  - think hard about data model
  - hybrid approach?



Conclusion

- Ruby + Redis = killer combo

Q & A

redis.io
github.com/tlossen/remodel
wooga.com/jobs