Data Modeling for Scale with Riak Data Types

Sean Cribbs
@seancribbs #riak #datatypes

QCon NYC 2014

I work for Basho

We make riak

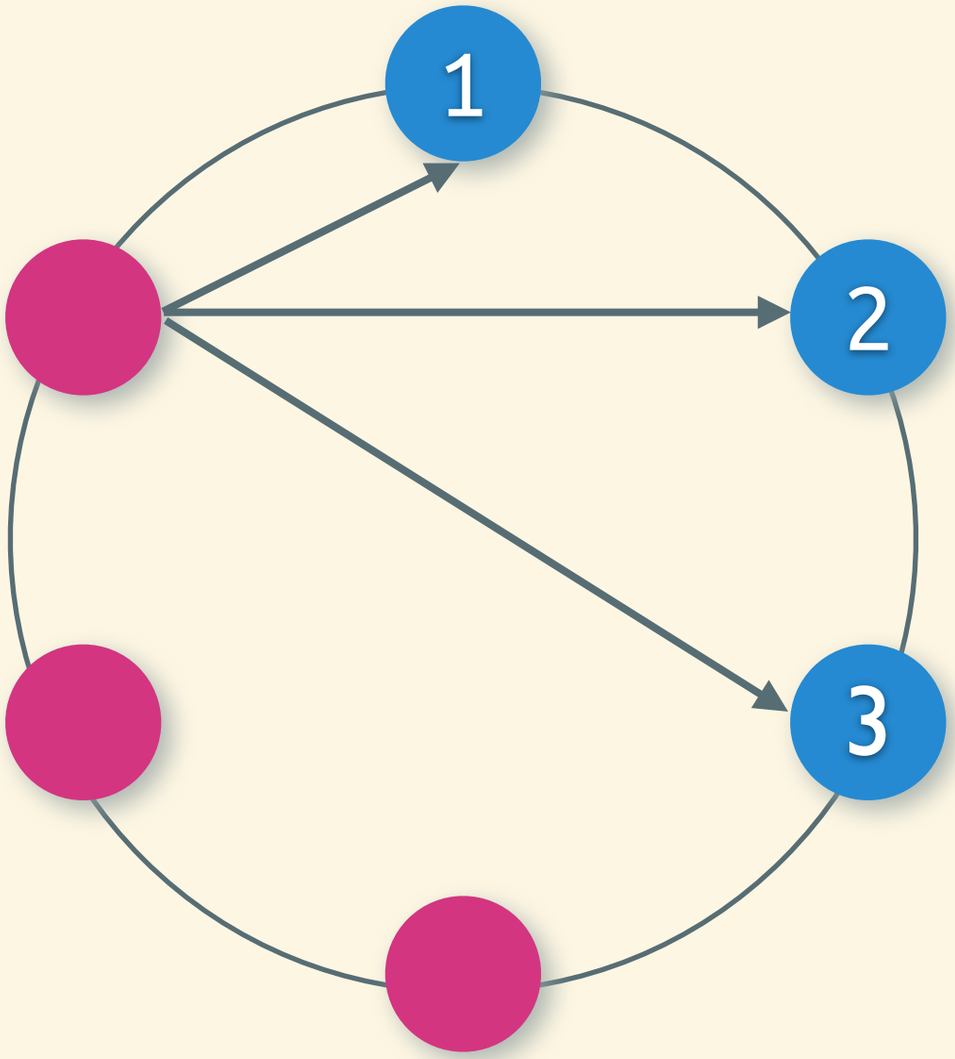
Visit our booth!

Riak is

Eventually Consistent

key-value + indexes + search + MapReduce

Eventual Consistency



Replicated

Loose coordination

Convergence

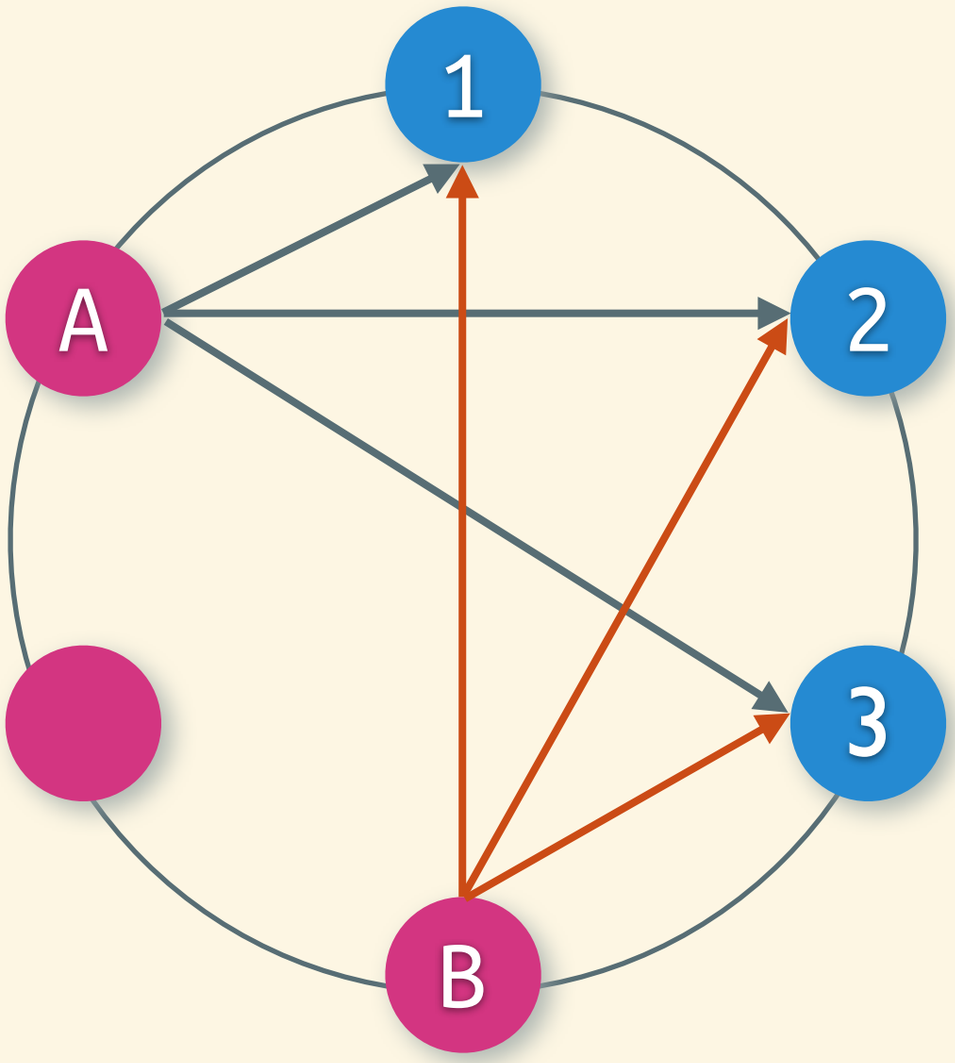
Eventual is Good

✔ Fault-tolerant

✔ Highly available

✔ Low-latency

Consistency?

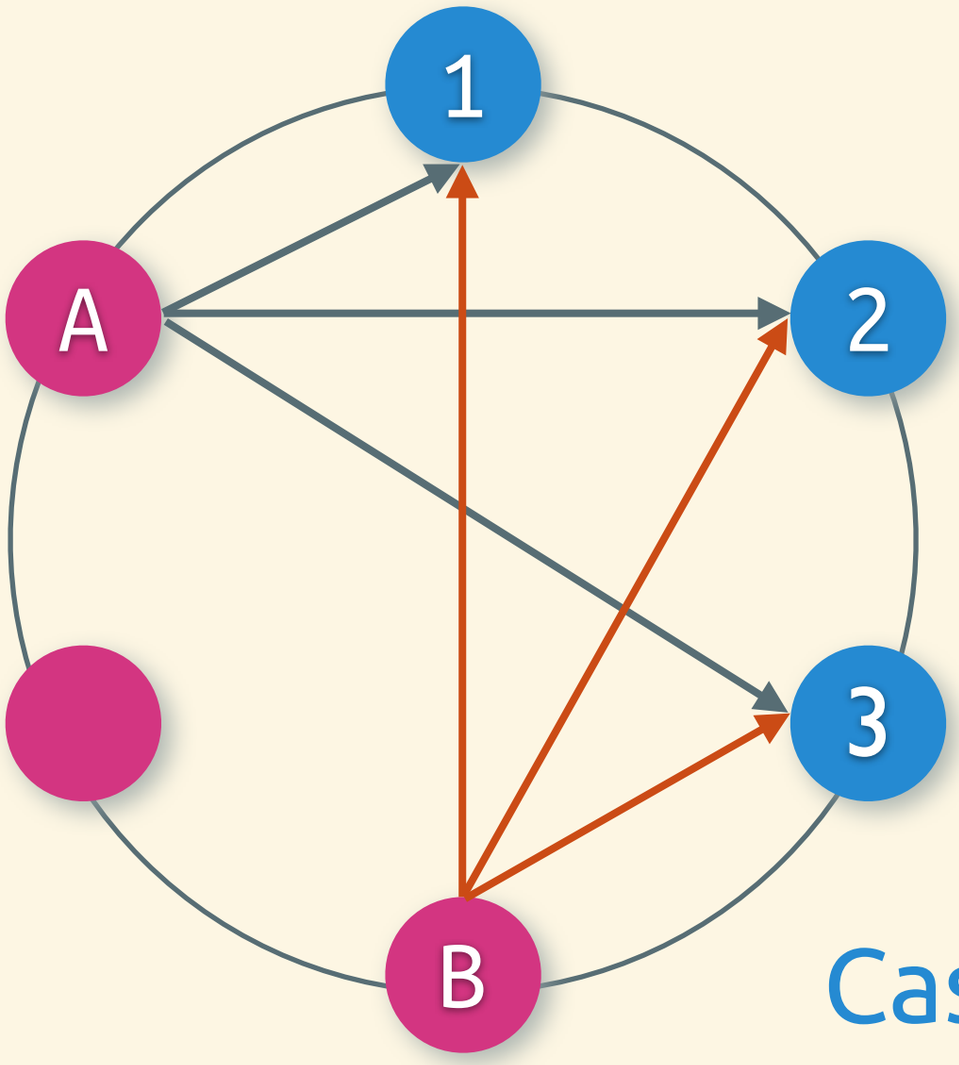


No clear winner!

Throw one out?

Keep both?

Consistency?



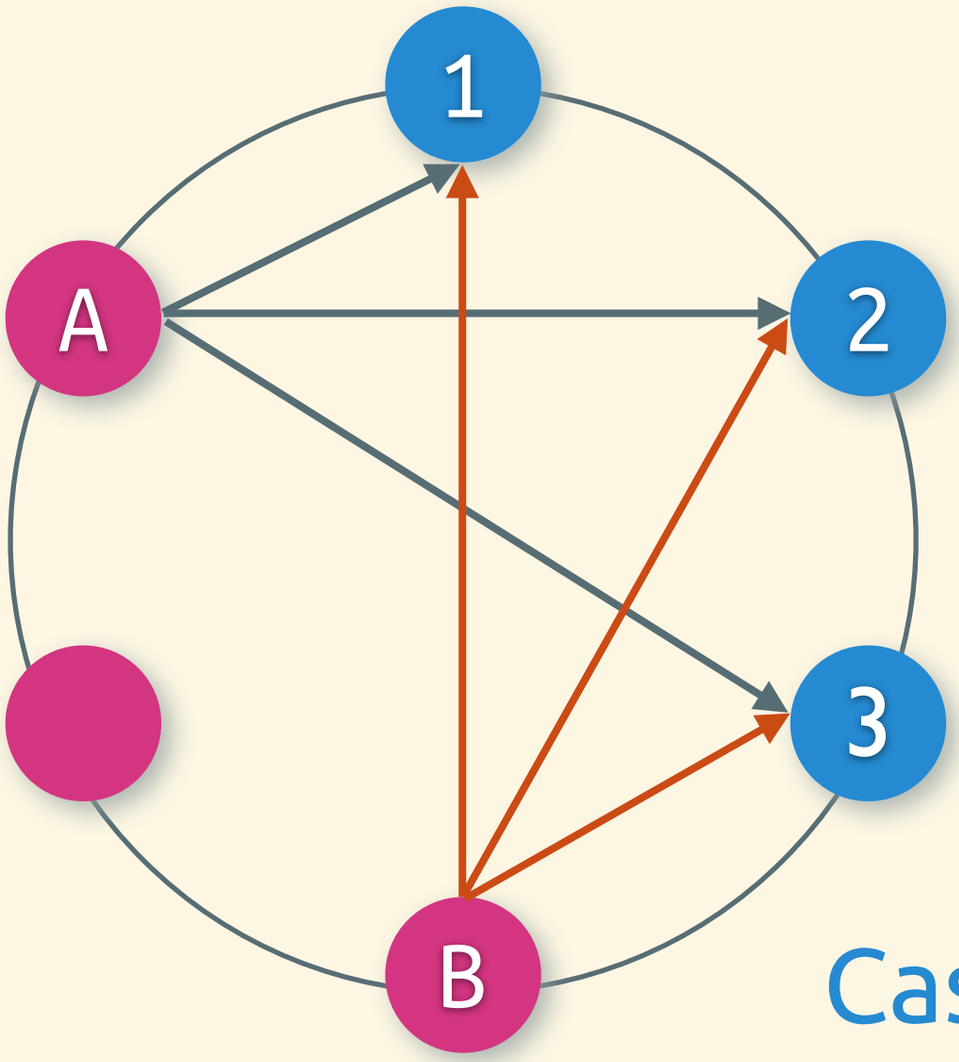
No clear winner!

Throw one out?

Keep both?

Cassandra

Consistency?



No clear winner!

Throw one out?

Keep both?

Cassandra

Riak

Conflicts!

A!

B!

Semantic Resolution

- Your app knows the domain - use business rules to resolve
- Amazon Dynamo’s shopping cart

Semantic Resolution

- Your app knows the domain - use business rules to resolve
- Amazon Dynamo's shopping cart

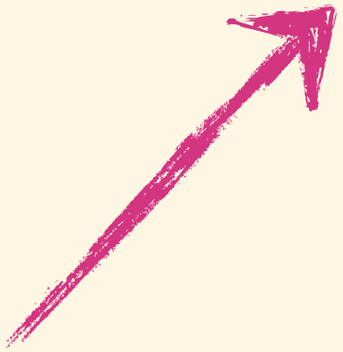
"Ad hoc approaches have proven brittle and error-prone"

Convergent Replicated Data Types

Convergent Replicated Data Types

useful abstractions

Convergent Replicated Data Types



multiple independent copies

useful abstractions

resolves automatically
toward a single value

Convergent Replicated Data Types

multiple independent
copies

useful abstractions

How CRDTs Work

- A partially-ordered **set of values**
- A **merge** function
- An **identity** value
- **Inflation** operations

How CRDTs Work

- A partially-ordered **set of values**
- A **merge** function
- An **identity** value
- **Inflation** operations

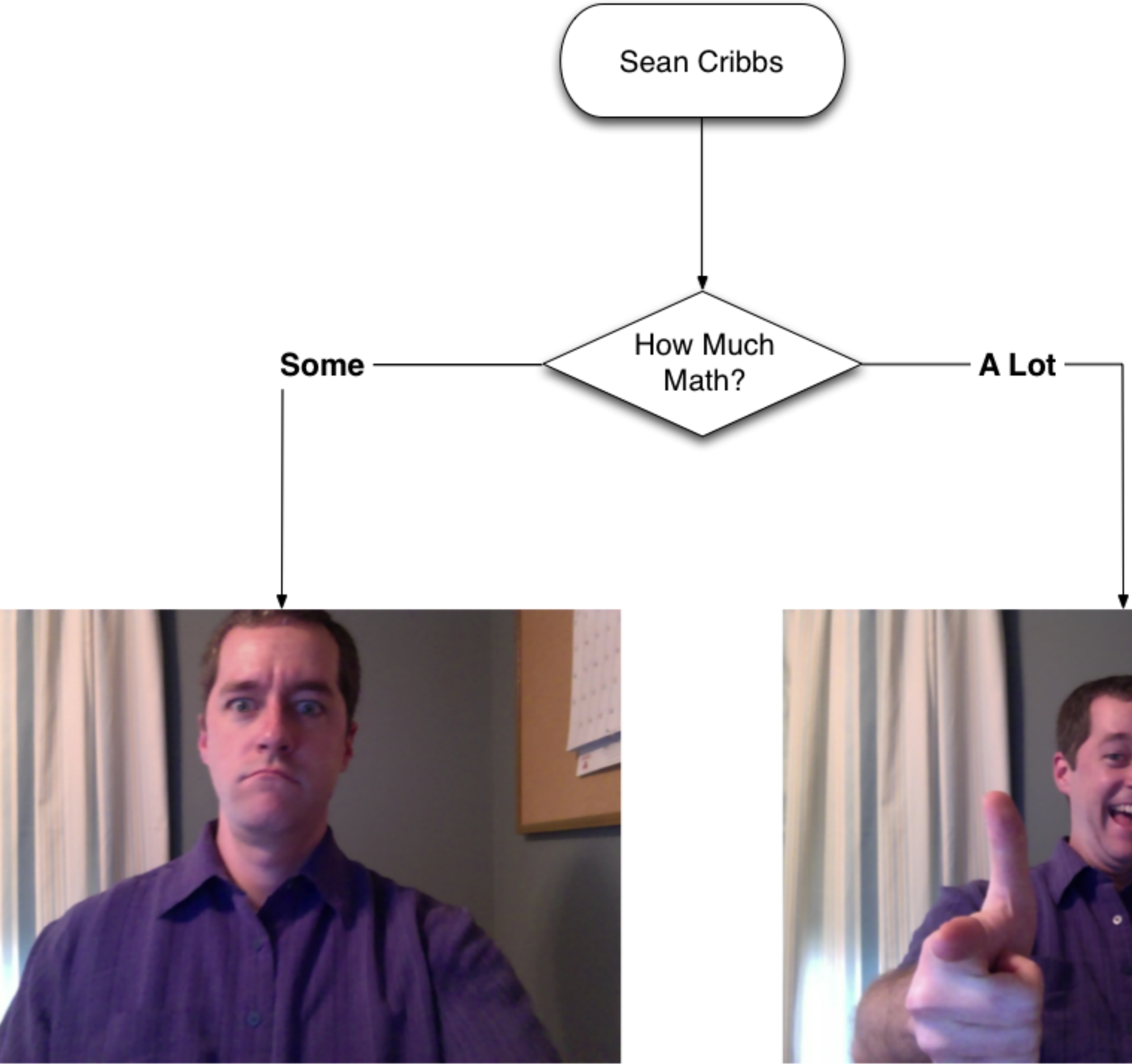
What CRDTs Enable

- Consistency **without coordination**
- **Fluent**, rich interaction with data

SyncFree

This research is supported in part by European FP7 project 609 551 SyncFree http://syncfree.lip6.fr/ (2013--2016).

by @joedevivo



Forget CRDTs

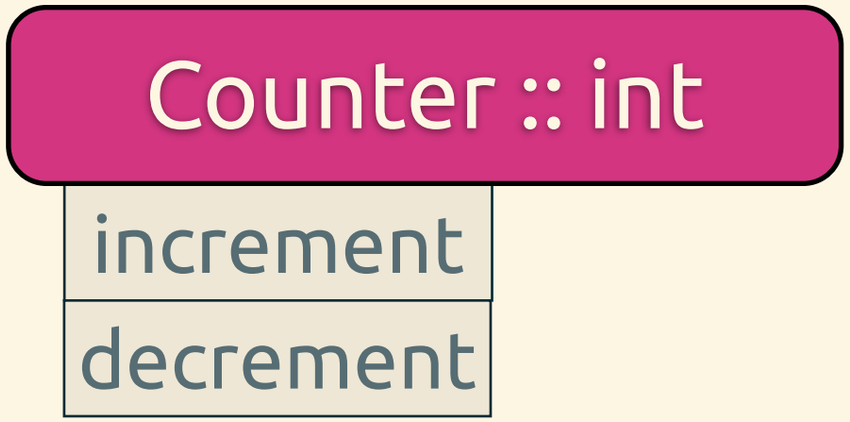
Do Data Modeling

Data Modeling for Riak

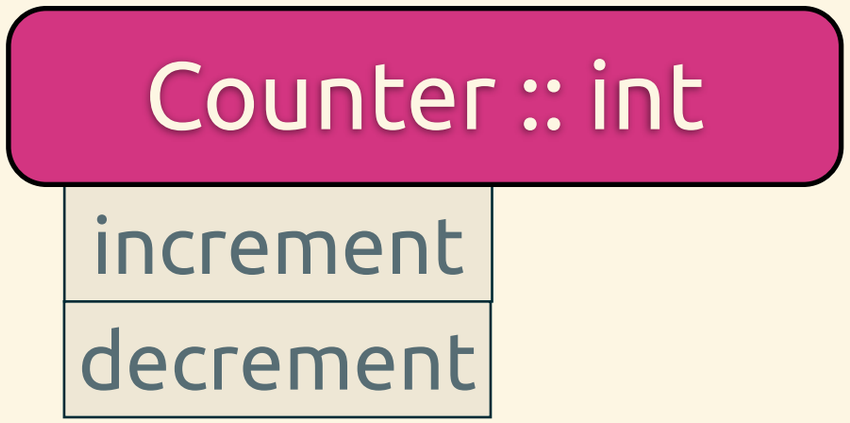
- Identify needs for both **read** and **write**
- Design around **key as index**
- **Denormalize** relationships if possible
- Weigh **data size** against **coherence**

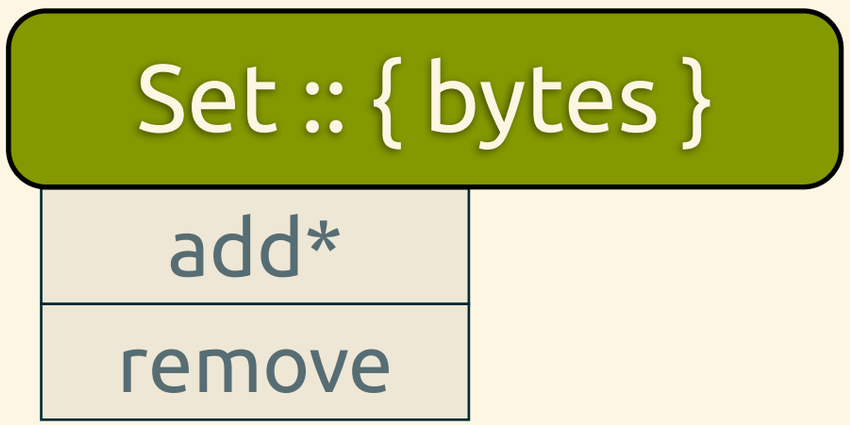
Riak Data Types

Riak Data Types



Riak Data Types





Riak Data Types

Map :: bytes → DT

- update*
- remove

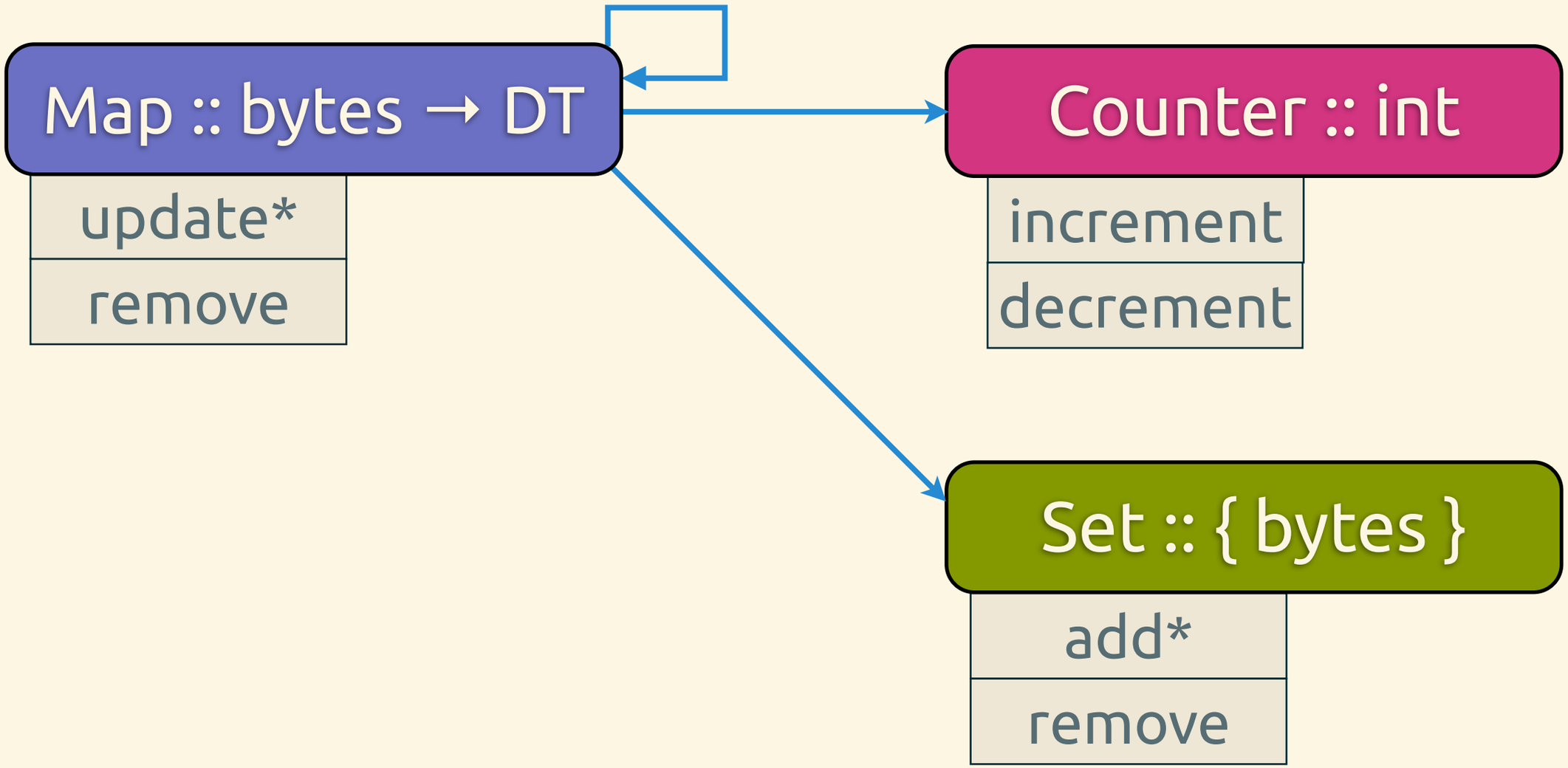
Counter :: int

- increment
- decrement

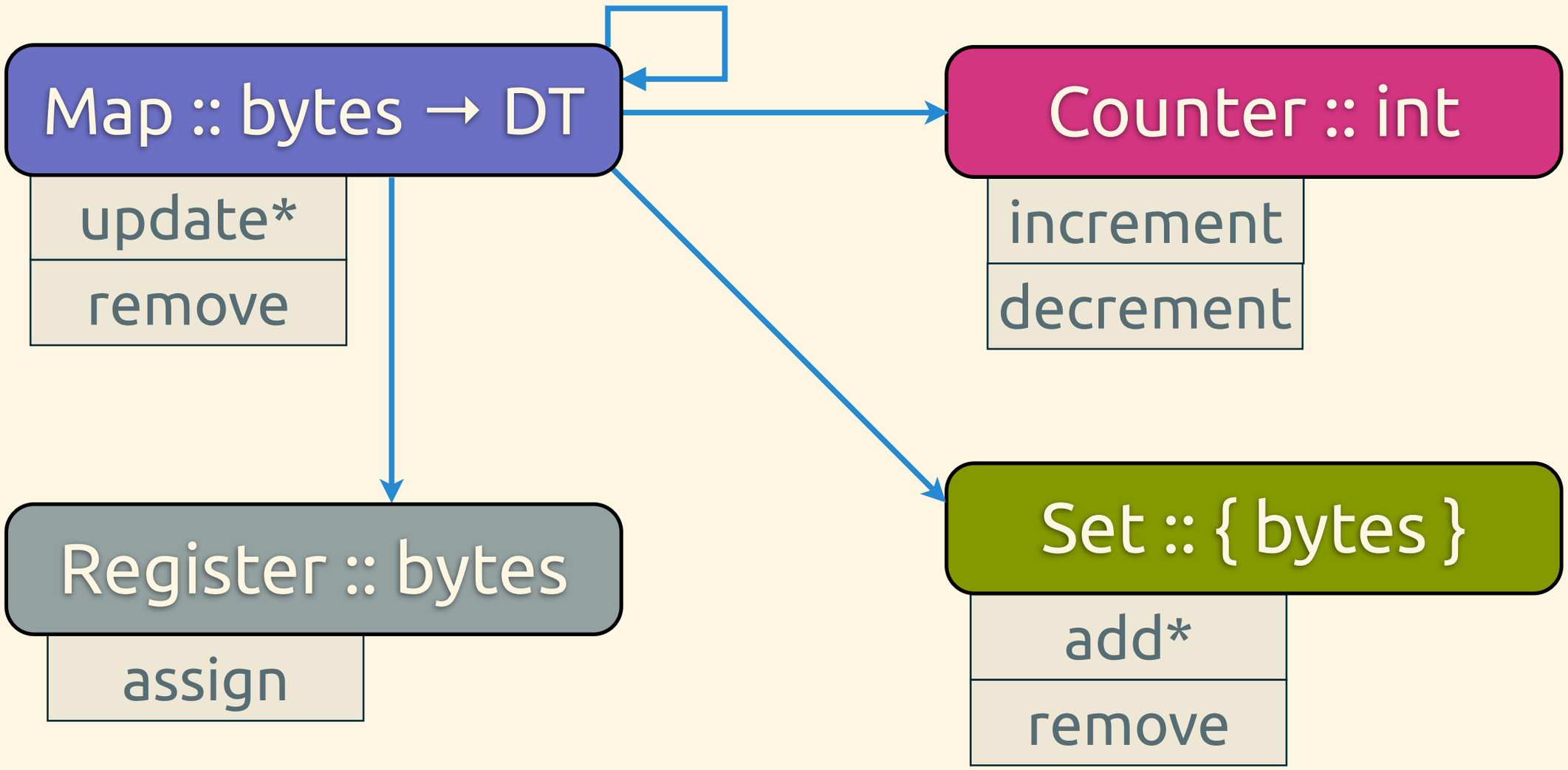
Set :: { bytes }

- add*
- remove

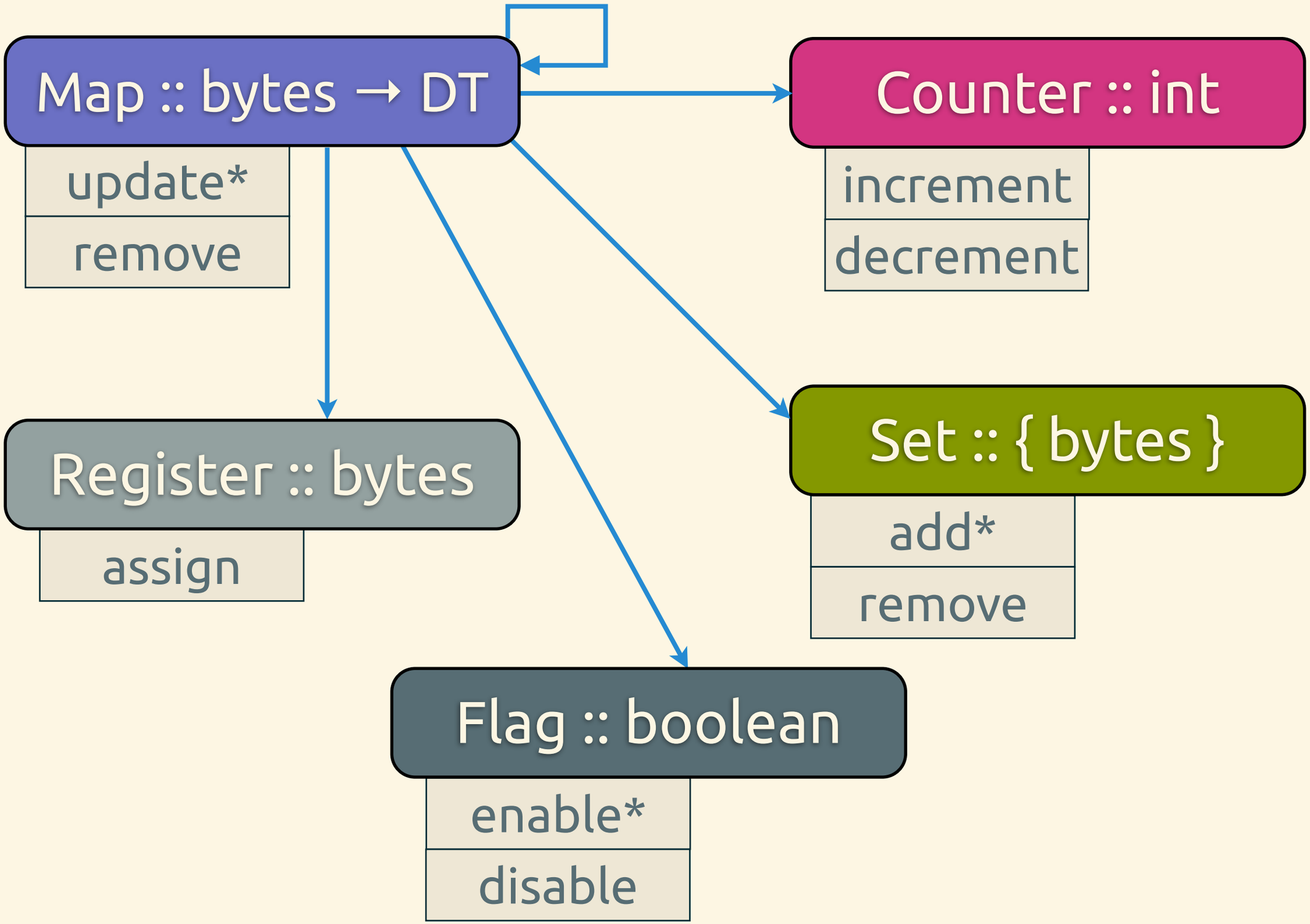
Riak Data Types



Riak Data Types



Riak Data Types



MAD DATA

Counters

Ad Network

- **Impressions** - when someone sees an ad
- **Click-through** - when someone clicks on an ad
- Hourly rollups `ad-metrics/<campaign>/<type>-<hour>`

Ad Network

```
$ riak-admin bucket-type create ad-metrics \
  '{"props":{"datatype":"counter"}}'
ad-metrics created

$ riak-admin bucket-type activate ad-metrics
ad-metrics has been activated

$ riak-admin bucket-type list
ad-metrics (active)
```

Ad Network

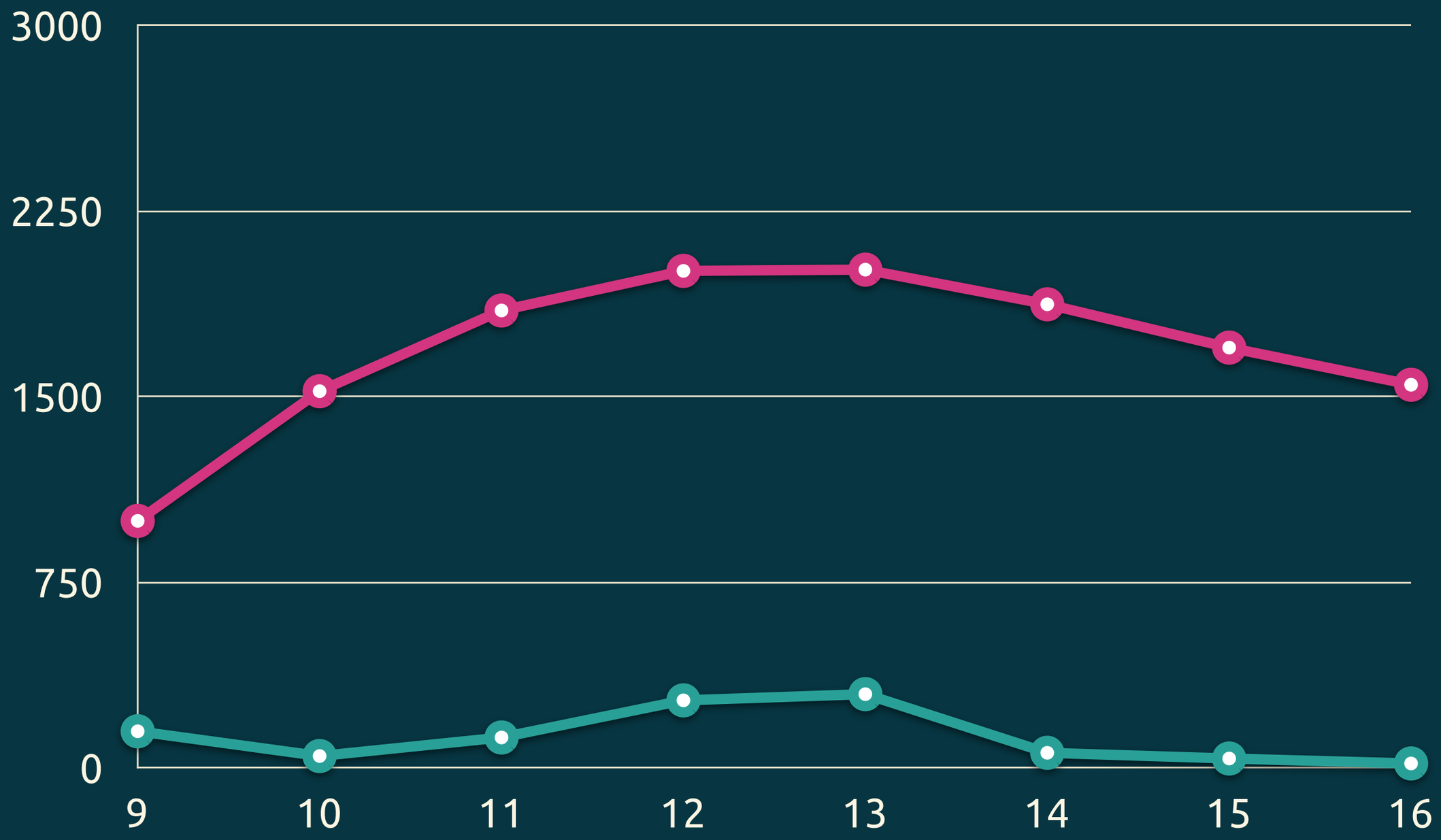
```
from riak import RiakClient
from rogersads import RIAK_CONFIG
from time import strftime

client = RiakClient(**RIAK_CONFIG)

metrics = client.bucket_type('ad-metrics')

def record_metric(campaign, metric_type):
    key = metric_type + strftime('-%Y%m%d-%H')
    counter = metrics.bucket(campaign).new(key)
    counter.increment()
    counter.store()
```

Ad Network



Sets

PartyOn

- **RSVPs** - guest lists
- **Connections** - friends lists per-user
- **Likes** - expressing interest

PartyOn

```
$ riak-admin bucket-type create partyon-sets \
  '{"props":{"datatype":"set"}}'
partyon-sets created

$ riak-admin bucket-type activate partyon-sets
partyon-sets has been activated

$ riak-admin bucket-type list
partyon-sets (active)
```

PartyOn

- **RSVPs**
  `partyon-sets/rsvps/<eventid>`
- **Connections**
  `partyon-sets/friends/<userid>`
- **Likes**
  `partyon-sets/likes/<eventid>`

PartyOn

```
from riak.datatypes import Set

sets = client.bucket_type('partyon-sets')

rsvps = sets.bucket('rsvps')
friends = sets.bucket('friends')
likes = sets.bucket('likes')
```

PartyOn

```
def rsvp_get(event):
    return rsvps.get(event) # Returns a Set

def rsvp_add(event, user):
    guests = rsvps.new(event)
    guests.add(user)
    guests.store(return_body=True)
    return guests.context

def rsvp_remove(event, user, context):
    guests = Set(rsvps, event, context=context)
    guests.remove(user)
    guests.store()
```

Maps
(and the rest)

GameNet

- **User profiles** - demographic data
  `users/profiles/<userid>`
- **Achievements** - trophies per game
  `users/trophies/<userid>`
- **Game state** - progress and stats
  `users/<gameid>/<userid>`

GameNet

```
$ riak-admin bucket-type create users \
  '{"props":{"datatype":"map"}}'
users created

$ riak-admin bucket-type activate users
users has been activated

$ riak-admin bucket-type list
users (active)
```

GameNet

```
users = client.bucket_type('users')

def update_profile(user, fields):
    profile = users.bucket('profiles').get(user)
    for field in fields:
        if field in USER_FLAGS:
            if fields[field]:
                profile.flags[field].enable()
            else:
                profile.flags[field].disable()
        else:
            value = fields[field]
            profile.registers[field].assign(value)

    profile.store()
```

GameNet

```
def add_trophy(user, game, trophy):
    trophies = users.bucket('trophies').get(user)
    trophies.sets[game].add(trophy)
    trophies.store()

def get_trophies(user, game):
    trophies = users.bucket('trophies').get(user)
    return trophies.sets[game].value
```

GameNet

```
def build_structure(user, game, structure, gold,
                    wood, stone):
    gamestate = users.bucket(game).get(user)
    gamestate.sets['structures'].add(structure)
    gamestate.counters['gold'].decrement(gold)
    gamestate.counters['wood'].decrement(wood)
    gamestate.counters['stone'].decrement(stone)
    gamestate.store(return_body=True)
    return gamestate.value
```

GameNet

```
client.create_search_index('asteroids')
users.bucket('asteroids').set_property('search_index',
                                       'asteroids')
```

```
def find_asteroids_opponents(min_score=0):
    query = "score_counter:[{} to *]".format(min_score)
    results = client.fulltext_search(
                    'asteroids', query,
                   fl=['userid_register',
                       'score_counter'])
    return results['docs']
```

Benefits

- **Richer interactions**, familiar types
- Write **mutations**, not state
- No **merge function** to write
- Same **reliability** and **predictability** of vanilla Riak

Caveats

- **Value size** still matters
- Updates **not idempotent** (yet)
- Cross-key atomicity **not possible** (yet)

Future

- Riak 2.0 due out this summer - betas available now!
- Richer querying, lighter storage requirements, more types

C.R.D.T.

TO RESIST BUGS
AND WRITE LOSS.