Bigsets…CRDT sets but BIGGER

Russell Brown, Basho Technologies
russelldb@basho.com

basho

riak

riak KV

riakKV

riakTS

riakS2

This project is funded by the European Union,
7th Research Framework Programme, ICT call 10,
grant agreement n°609551.

4 Sections

1. What are CRDTs (good for)?
2. History of CRDT Sets
3. Sets in Riak
4. Bigger Sets in Riak

Why CRDTs?

Fundamental Trade Off

- Lipton/Sandberg '88
- Attiya/Welch '94
- Gilbert/Lynch '02

Low Latency/Availability:
- Increased Revenue
- User Engagement

Strong Consistency:
- Easier for Programmers
- Less user "surprise"

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

Categories and Subject Descriptors

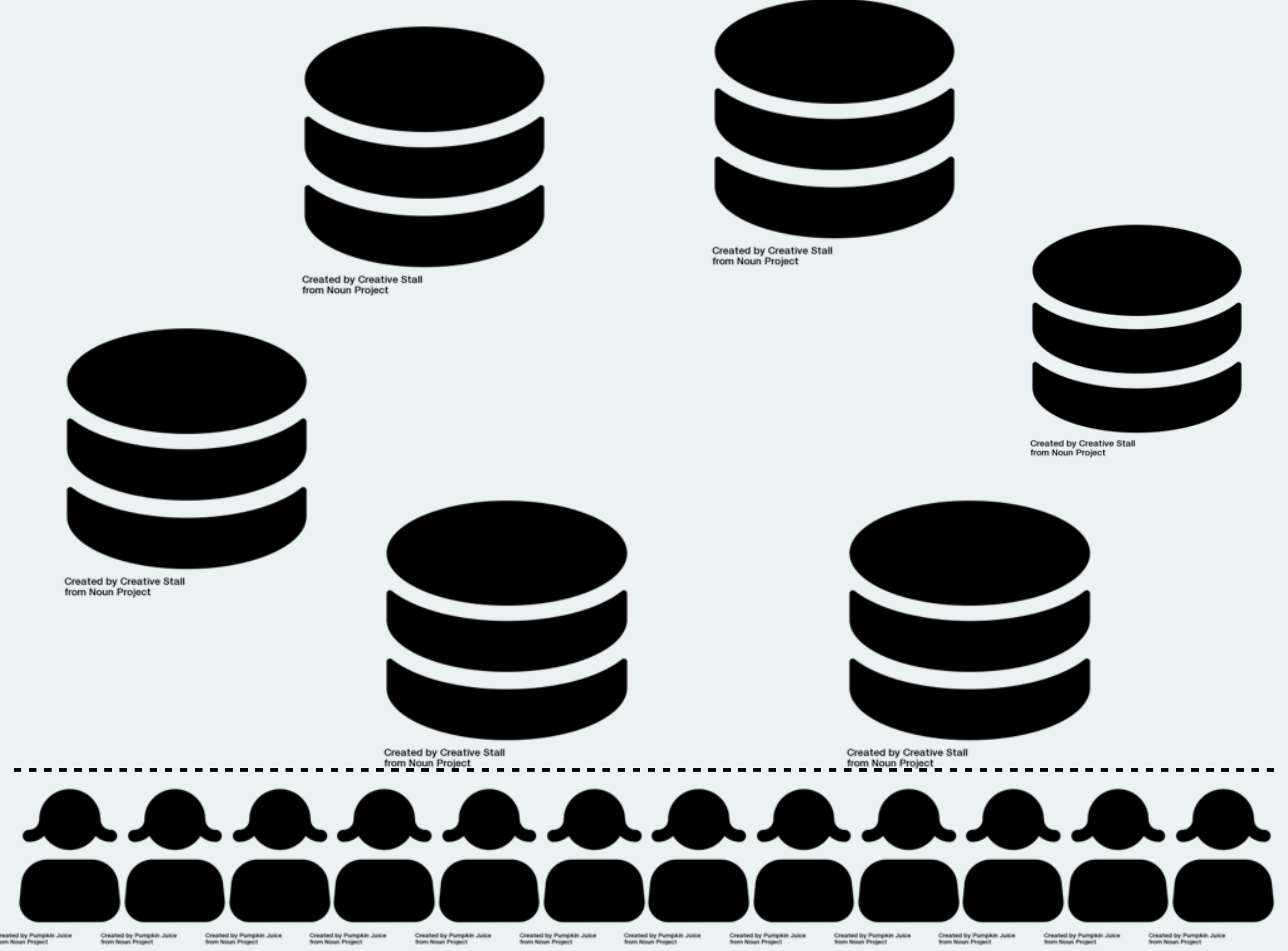
D.4.2 [**Operating Systems**]: Storage Management; D.4.5 [**Operating Systems**]: Reliability; D.4.2 [**Operating Systems**]: Performance;

General Terms

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

Dealing with failures in an infrastructure comprised of millions of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such Amazon's software systems need to be constructed in a manner that treats failure handling as the normal case without impacting availability or performance.

To meet the reliability and scaling needs, Amazon has developed a number of storage technologies, of which the Amazon Simple Storage Service (also available outside of Amazon and known as Amazon S3), is probably the best known. This paper presents the design and implementation of Dynamo, another highly available


Created by Creative Stall from Noun Project
Created by Creative Stall from Noun Project
Created by Creative Stall from Noun Project
Created by Creative Stall from Noun Project
Created by Creative Stall from Noun Project
Created by Creative Stall from Noun Project
reated by Pumpkin Juice om Noun Project
Created by Pumpkin Juice from Noun Project
Created by Pumpkin Juice from Noun Project
Created by Pumpkin Juice from Noun Project
Created by Pumpkin Juice from Noun Project
Created by Pumpkin Juice from Noun Project
Created by Pumpkin Juice from Noun Project
Created by Pumpkin Juice from Noun Project
Created by Pumpkin Juice from Noun Project
Created by Pumpkin Juice from Noun Project
Created by Pumpkin Juice from Noun Project
Created by Pumpkin Juice from Noun Project

2 REPLICAS

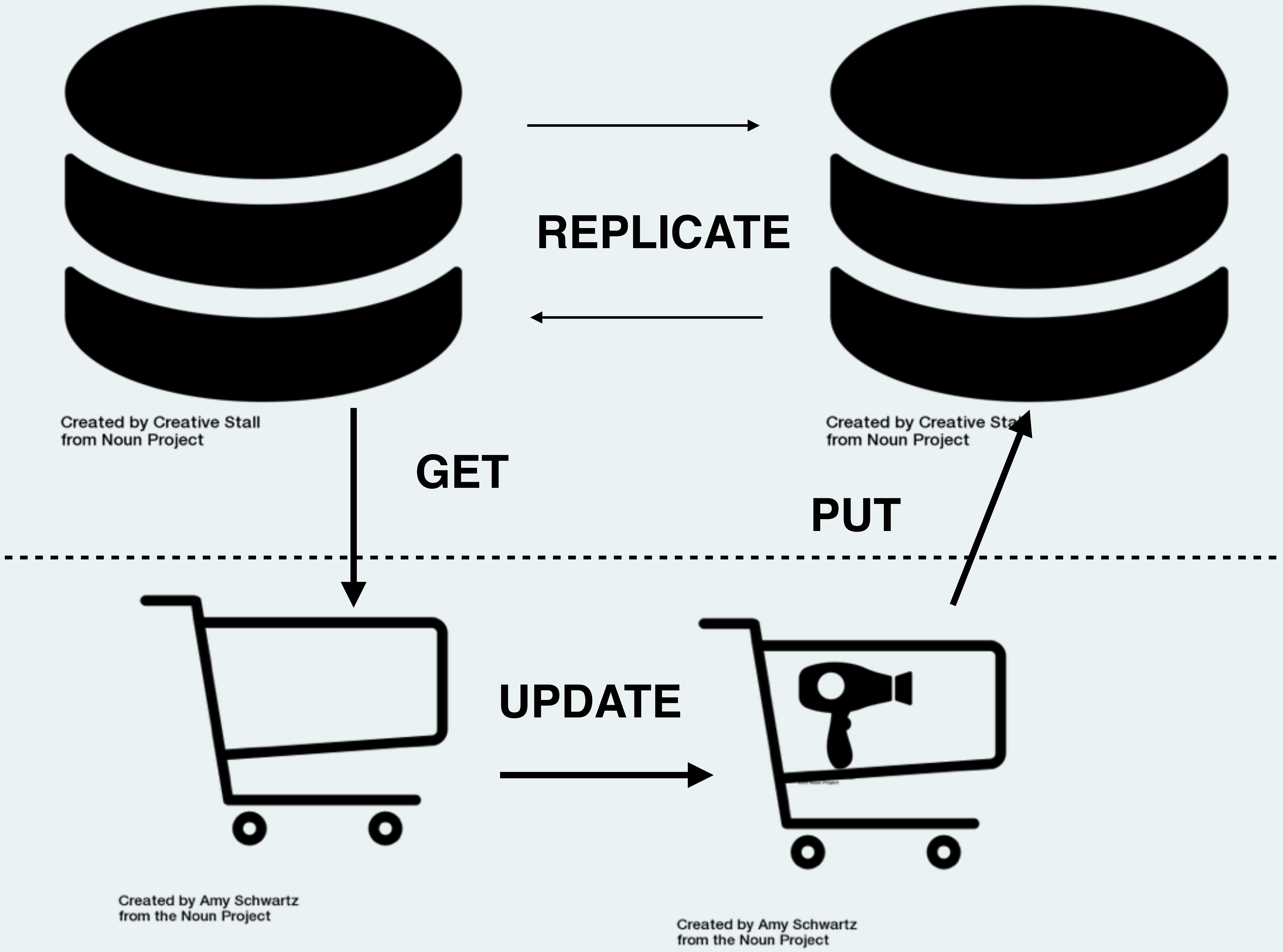
Created by Creative Stall
from Noun Project

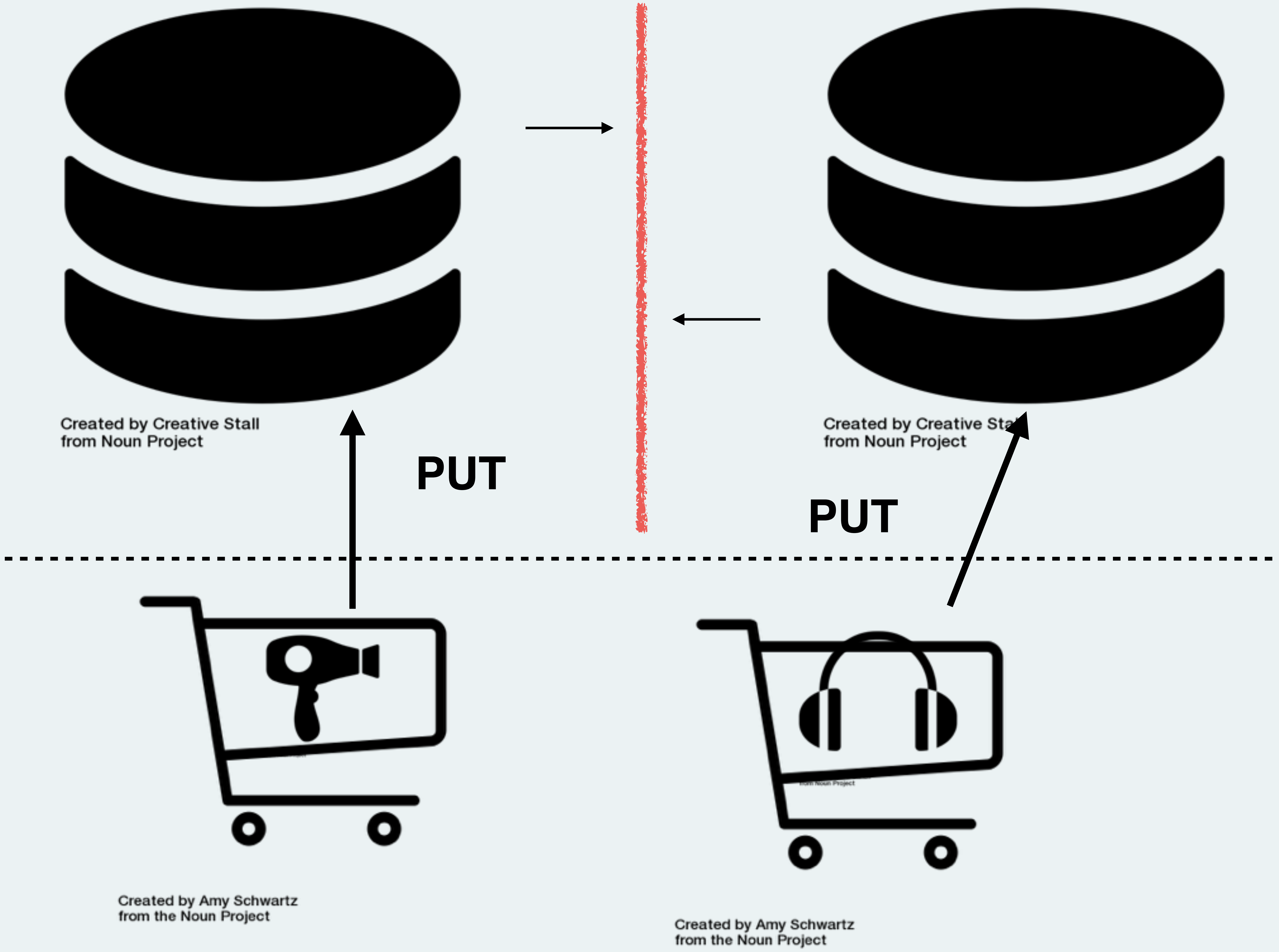
1 KEY

Created by Creative Stall
from Noun Project

---

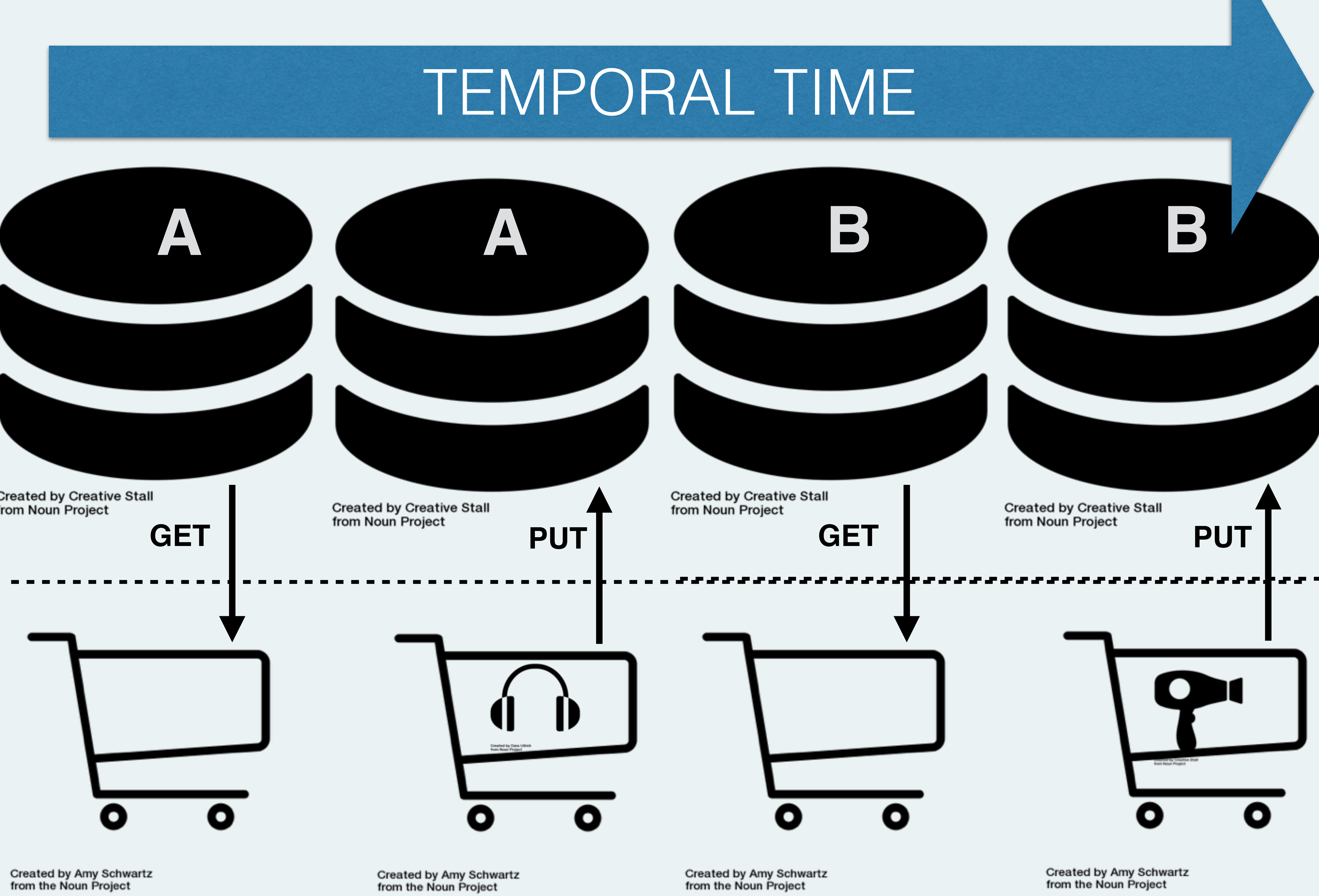
1 CLIENT


REPLICATE
GET
PUT
UPDATE
Created by Creative Stall from Noun Project
Created by Creative Stall from Noun Project
Created by Amy Schwartz from the Noun Project
Created by Amy Schwartz from the Noun Project


Created by Creative Stall
from Noun Project
PUT
Created by Creative Stall
from Noun Project
PUT
Created by Amy Schwartz
from the Noun Project
Created by Amy Schwartz
from the Noun Project

Quorum

Created by Creative Stall from Noun Project
Created by Creative Stall from Noun Project
£
Created by Charlie Bob Gordon from Noun Project

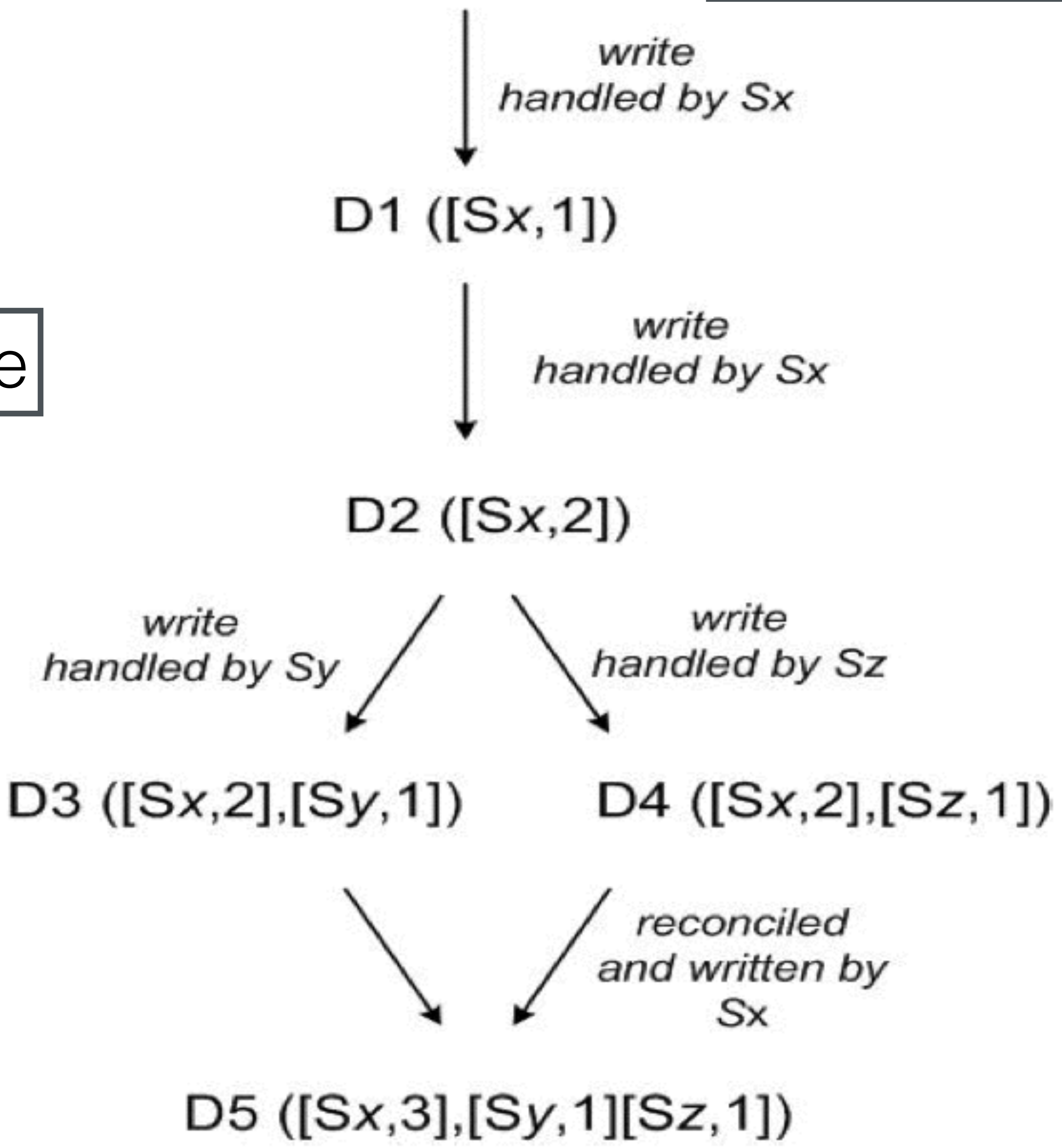

TEMPORAL TIME
A
A
B
B
Created by Creative Stall from Noun Project
Created by Creative Stall from Noun Project
Created by Creative Stall from Noun Project
Created by Creative Stall from Noun Project
GET
PUT
GET
PUT
Created by Amy Schwartz from the Noun Project
Created by Amy Schwartz from the Noun Project
Created by Amy Schwartz from the Noun Project
Created by Amy Schwartz from the Noun Project

Logical Clocks

happens before

concurrent
divergent

convergent

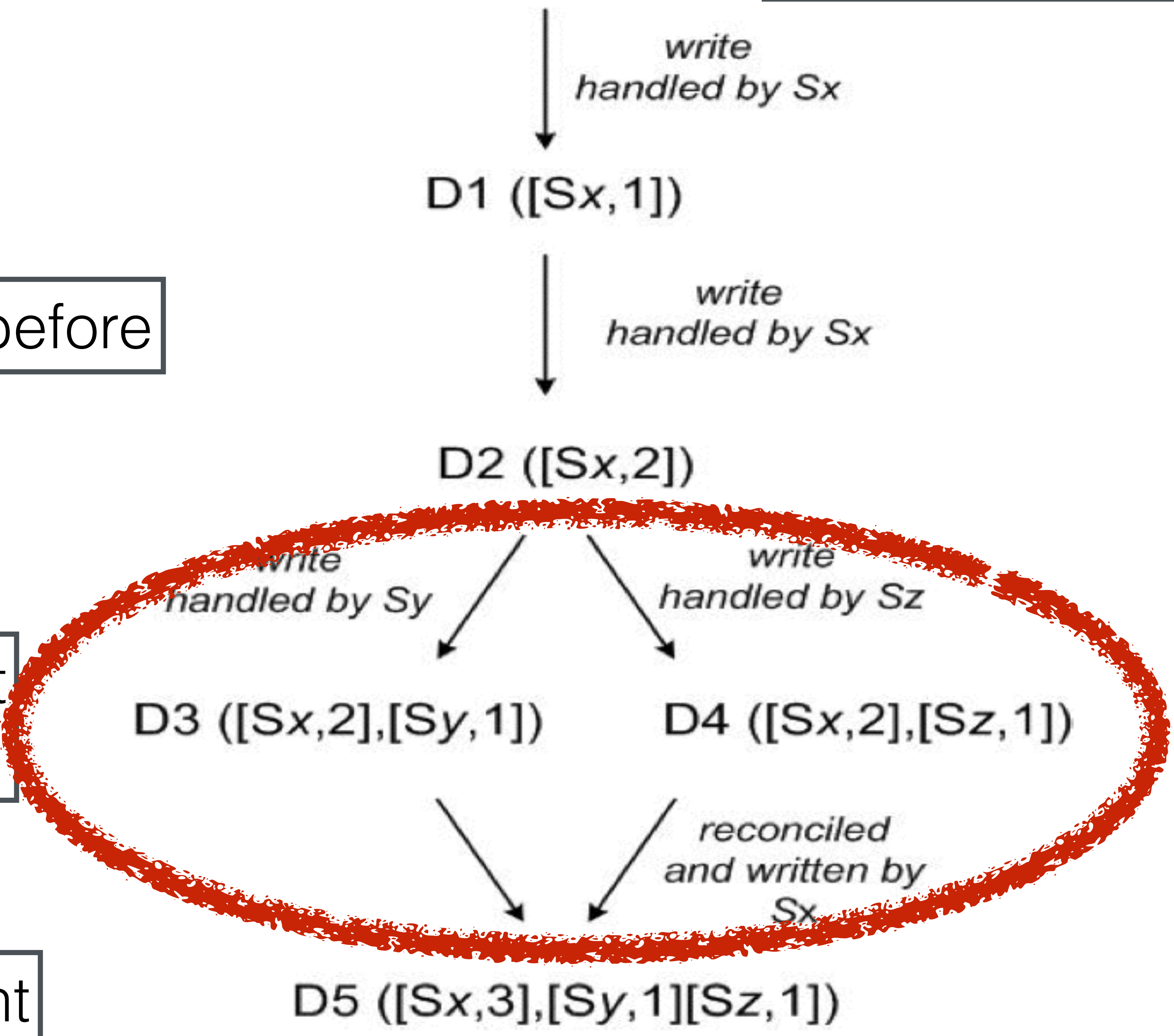


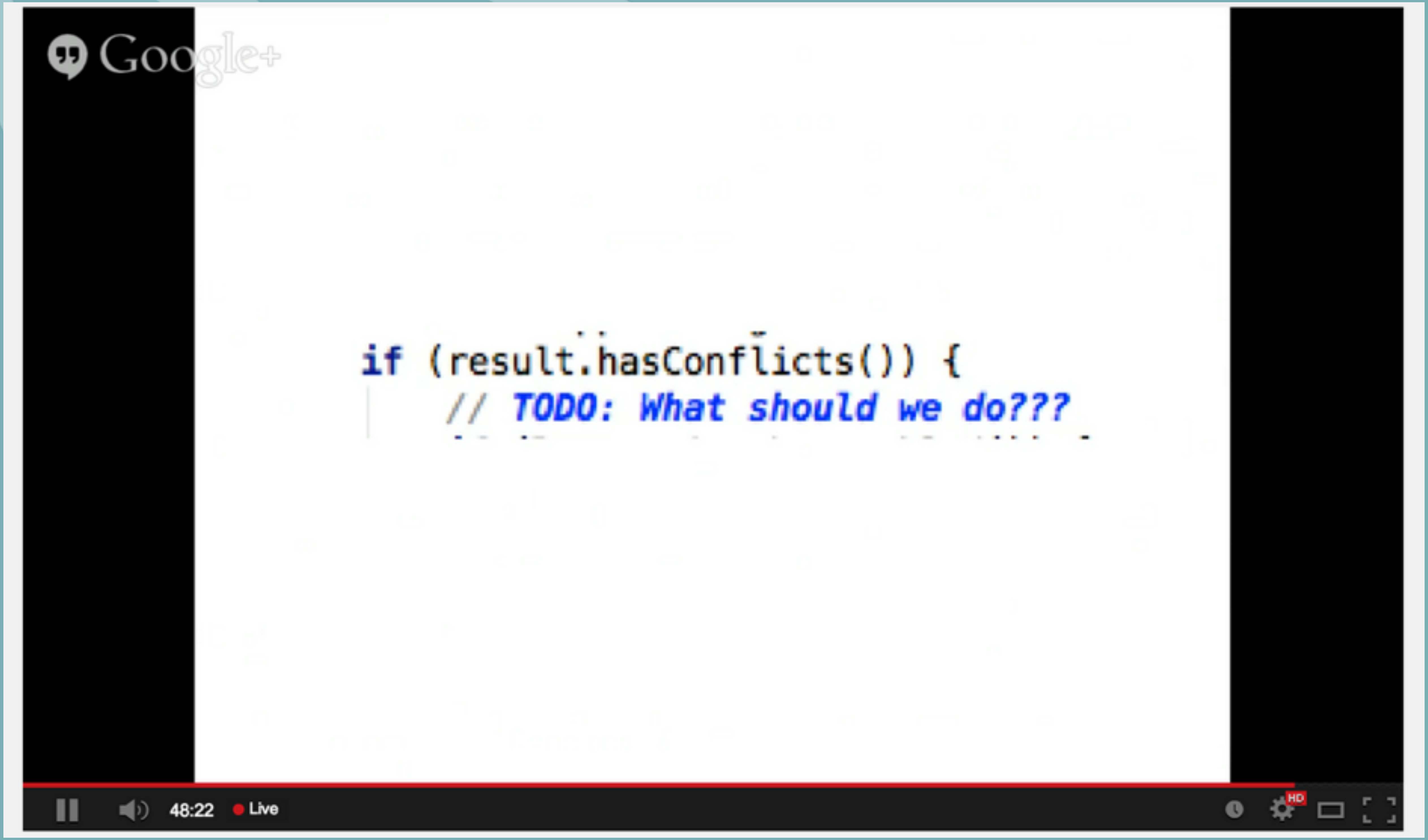
Logical Clocks

happens before

concurrent
divergent

convergent





```
if (result.hasConflicts()) {
    // TODO: What should we do???
```

Timestamp based reconciliation

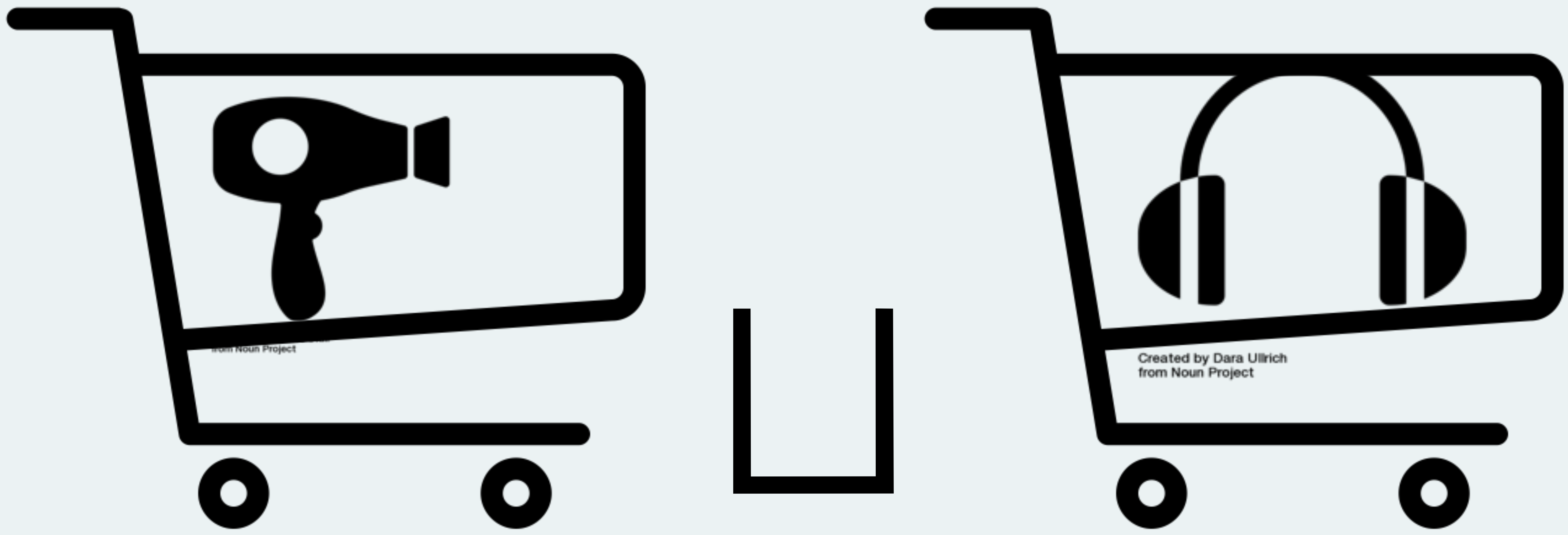
155196119890 > **155196118001**

Created by Amy Schwartz
from the Noun Project

Created by Amy Schwartz
from the Noun Project

Created by Amy Schwartz
from the Noun Project

Business Logic/Semantic Reconciliation



Created by Amy Schwartz
from the Noun Project

Created by Amy Schwartz
from the Noun Project

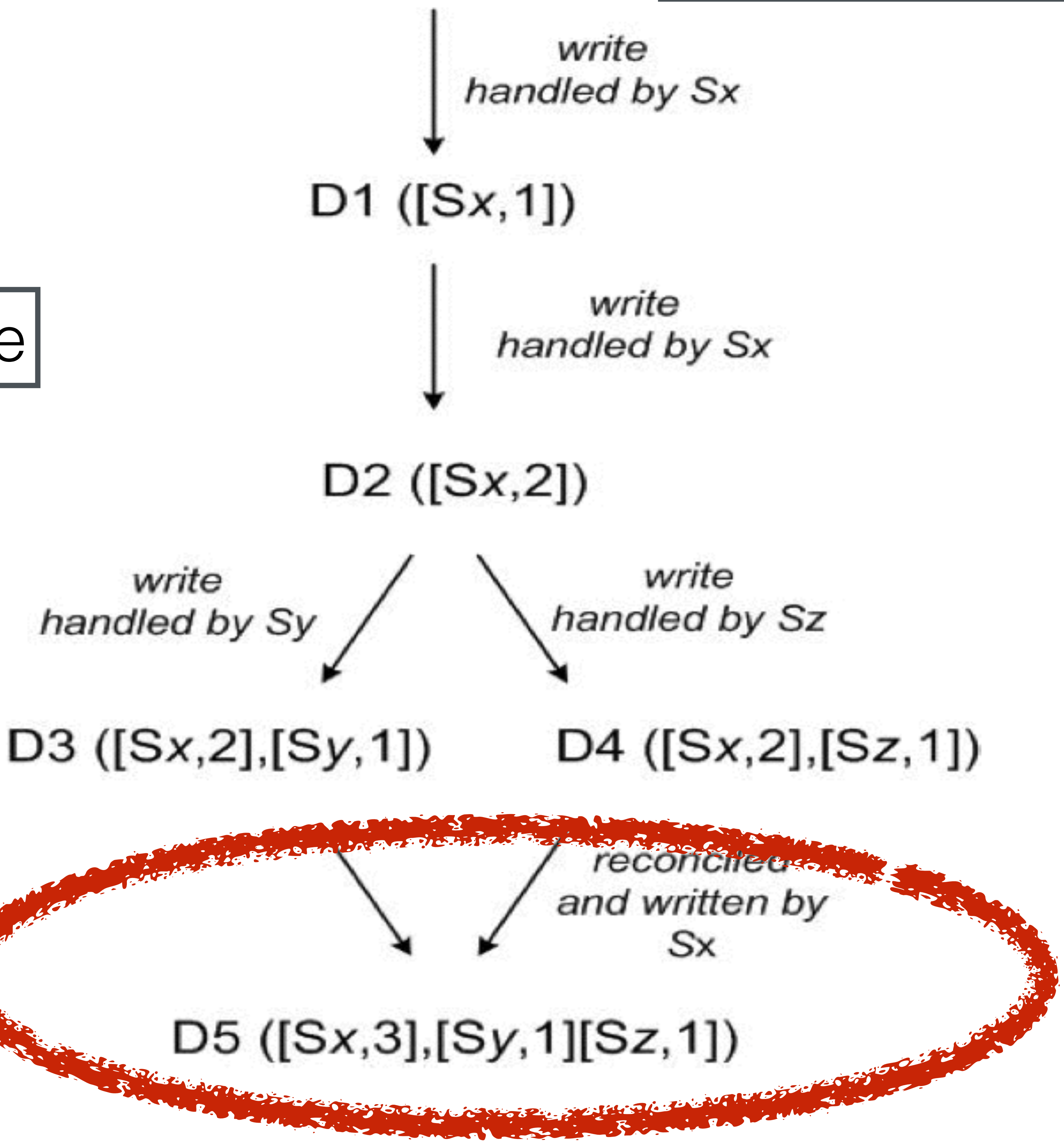
Created by Amy Schwartz
from the Noun Project

Logical Clocks

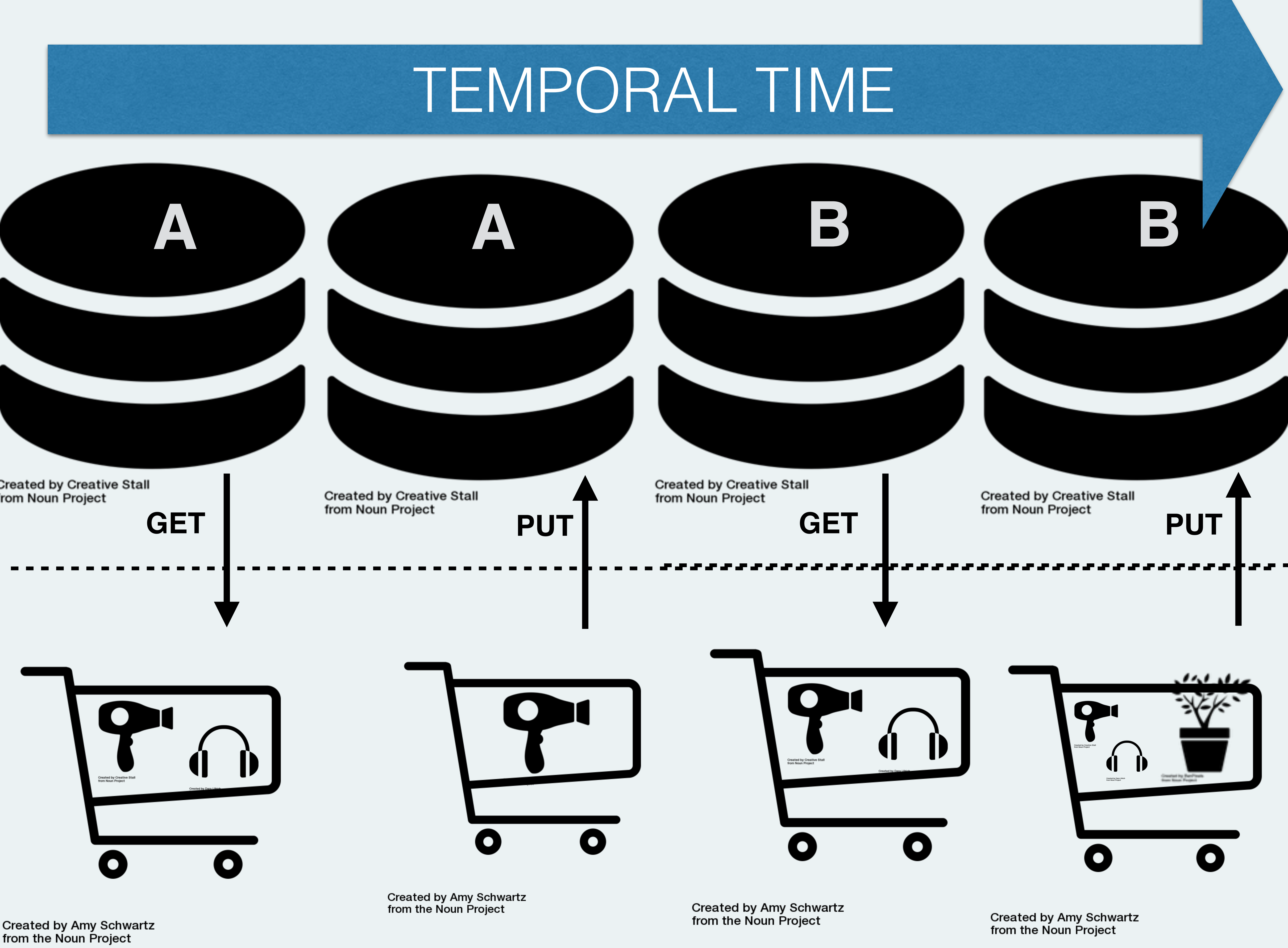
happens before

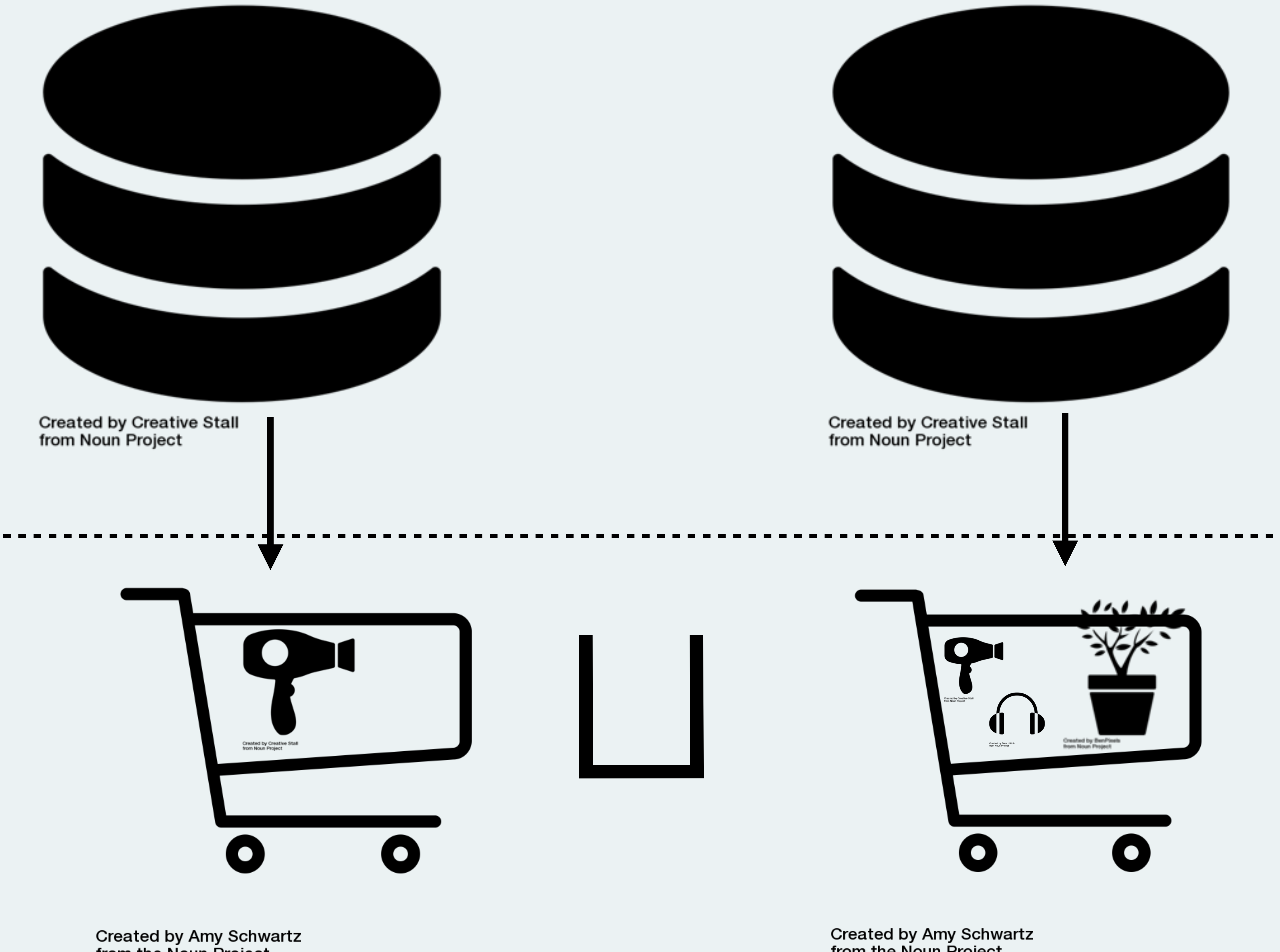
concurrent
~~divergent~~
divergent

convergent



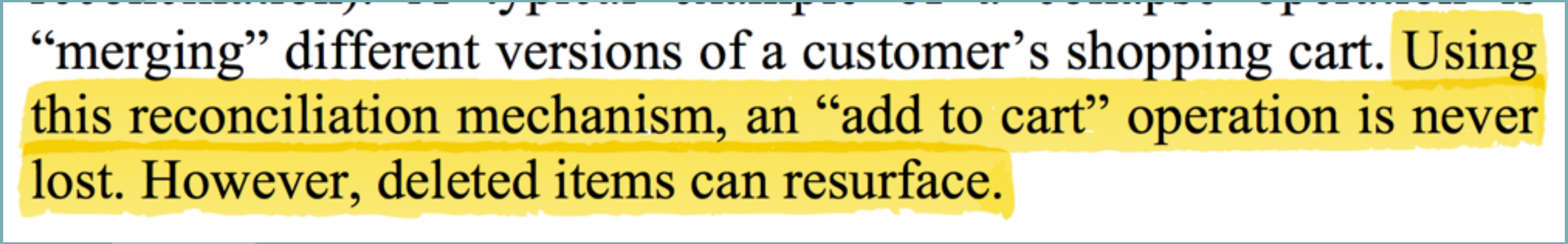
Removes?


TEMPORAL TIME
A
A
B
B
Created by Creative Stall from Noun Project
Created by Creative Stall from Noun Project
Created by Creative Stall from Noun Project
Created by Creative Stall from Noun Project
GET
PUT
GET
PUT
Created by Amy Schwartz from the Noun Project
Created by Amy Schwartz from the Noun Project
Created by Amy Schwartz from the Noun Project
Created by Amy Schwartz from the Noun Project


Created by Creative Stall
from Noun Project
Created by Creative Stall
from Noun Project
Created by Creative Stall
from Noun Project
Created by Creative Stall
from Noun Project
Created by BenPixels
from Noun Project
Created by Amy Schwartz
from the Noun Project
Created by Amy Schwartz
from the Noun Project

Created by Creative Stall from Noun Project
Created by BenPixels from Noun Project
Created by Dara Ullrich from Noun Project

Removes?


"merging" different versions of a customer's shopping cart. Using this reconciliation mechanism, an "add to cart" operation is never lost. However, deleted items can resurface.

Google F1

“Designing applications to cope with concurrency **anomalies** in their data is very **error-prone, time-consuming**, and ultimately ***not worth the performance gains***.”

"…writing merge functions was likely to **confuse the hell** out of all our developers and ***slow down development***…"

riak

http://www.infoq.com/articles/key-lessons-learned-from-transition-to-nosql

CRDTs

DATA TYPES That CONVERGE

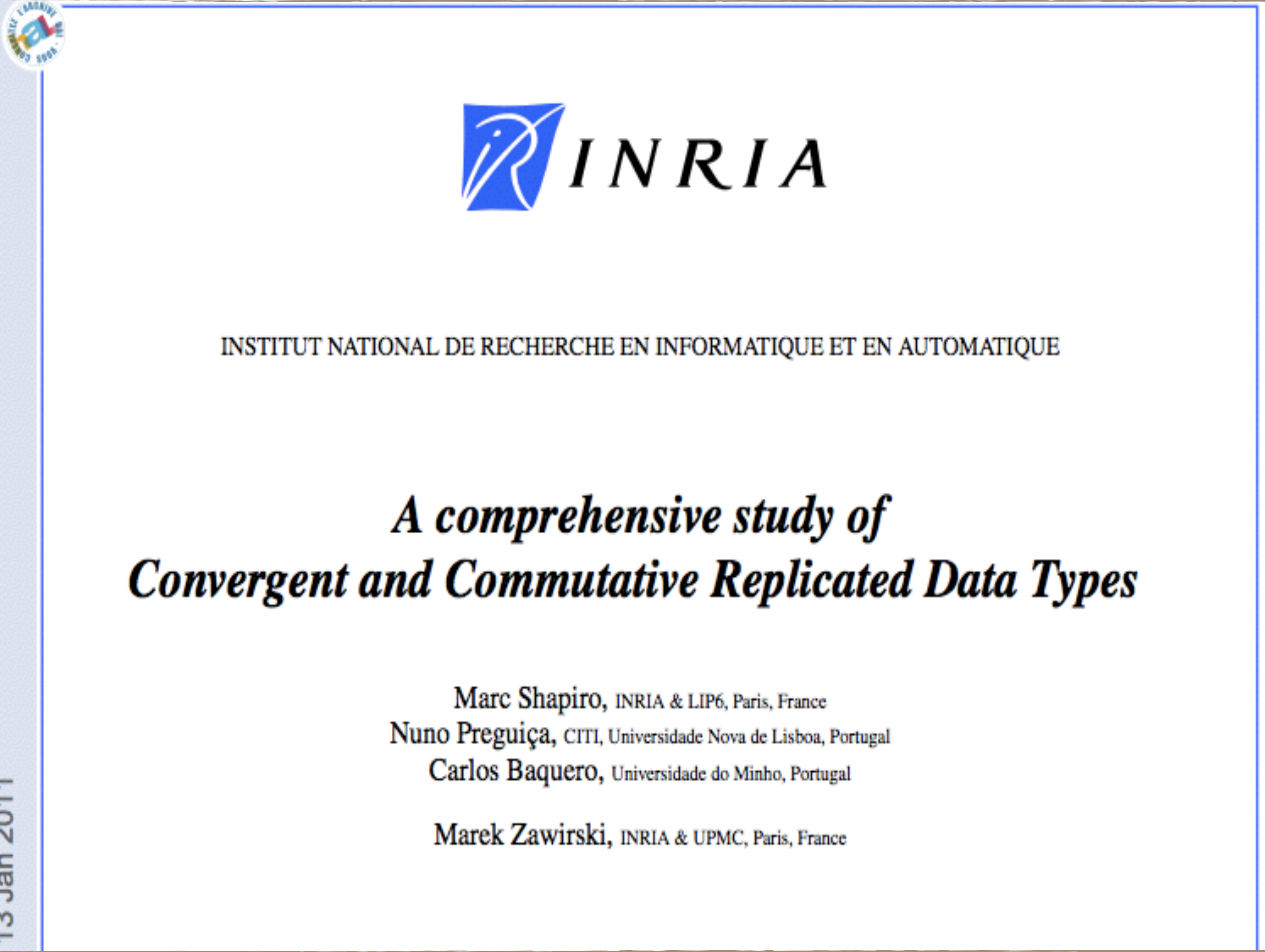
CRDTs

Off the shelf MERGE functions

CRDTs

CRDTs are Data Types (maps/sets/booleans/graphs/etc)

THAT CONVERGE



INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A comprehensive study of Convergent and Commutative Replicated Data Types

Marc Shapiro, INRIA & LIP6, Paris, France
Nuno Preguiça, CITI, Universidade Nova de Lisboa, Portugal
Carlos Baquero, Universidade do Minho, Portugal

Marek Zawirski, INRIA & UPMC, Paris, France

CRDT SETS

"…after some analysis we found that **much of our data could be modelled within sets** so by leveraging CRDT's our developers don't have to worry about writing bespoke merge functions for **95% of carefully selected use cases**…"

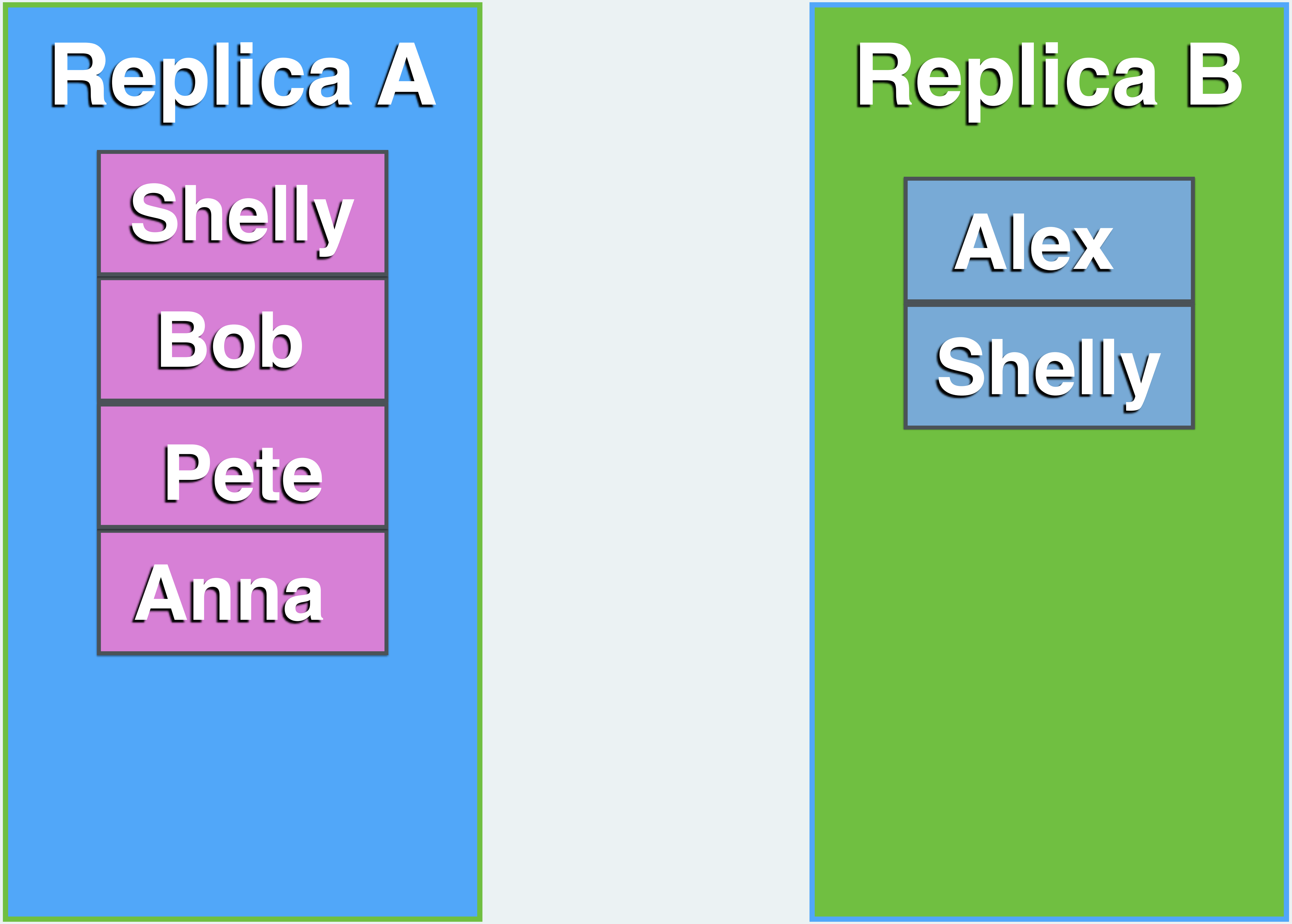
riak

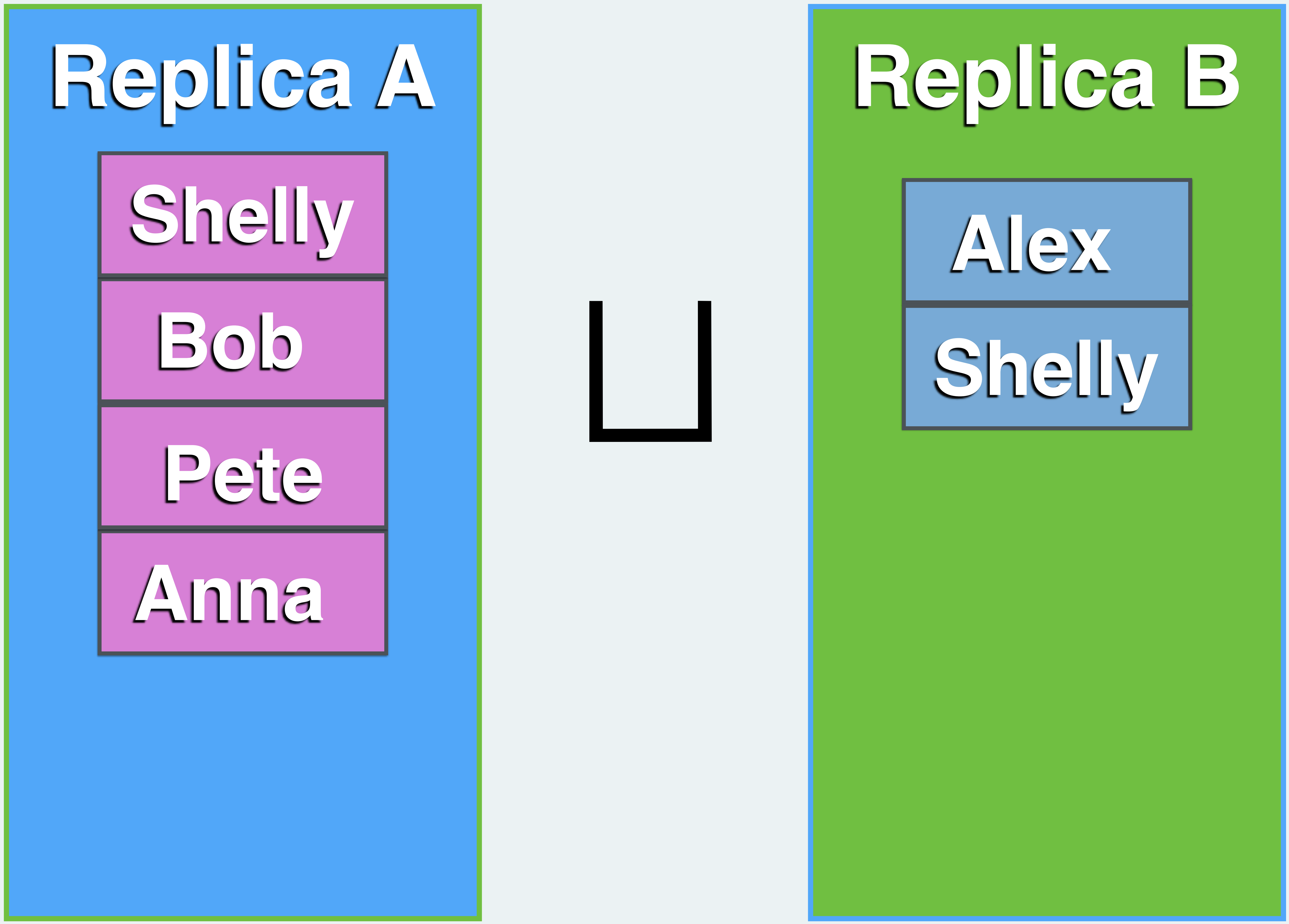
http://www.infoq.com/articles/key-lessons-learned-from-transition-to-nosql

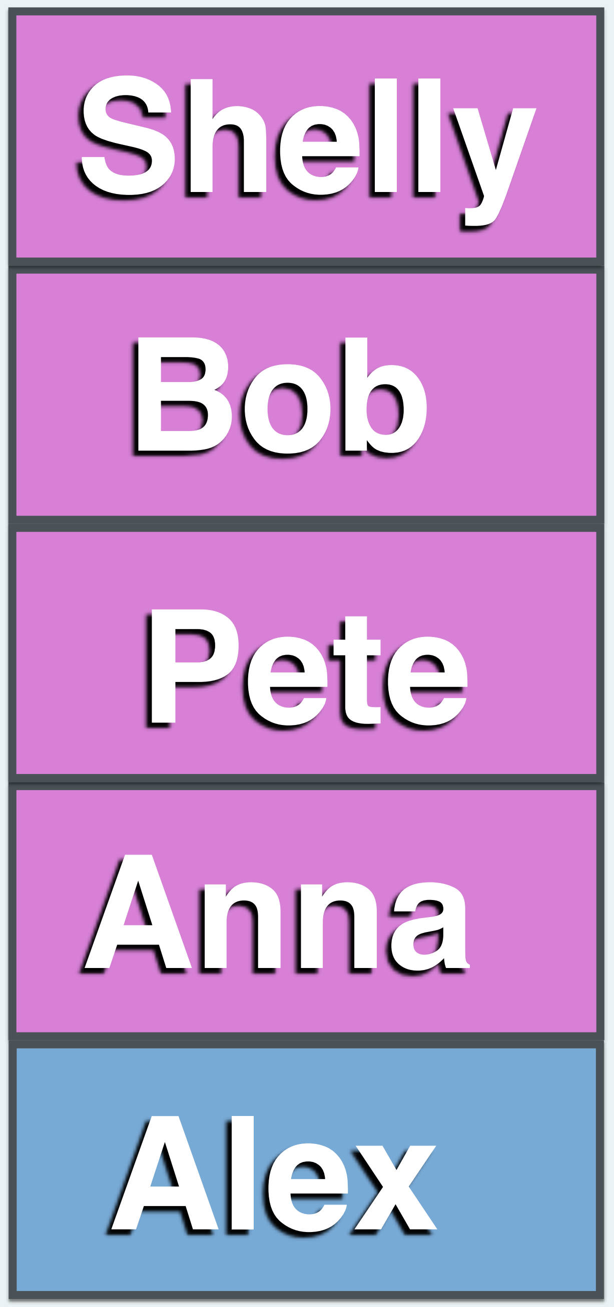
Evolution of a CRDT Set

Evolution of a Set

G-SET


Replica A
Shelly
Bob
Pete
Anna
Replica B
Alex
Shelly


Replica A
Shelly
Bob
Pete
Anna
⊔
Replica B
Alex
Shelly



G-SET

Removes?

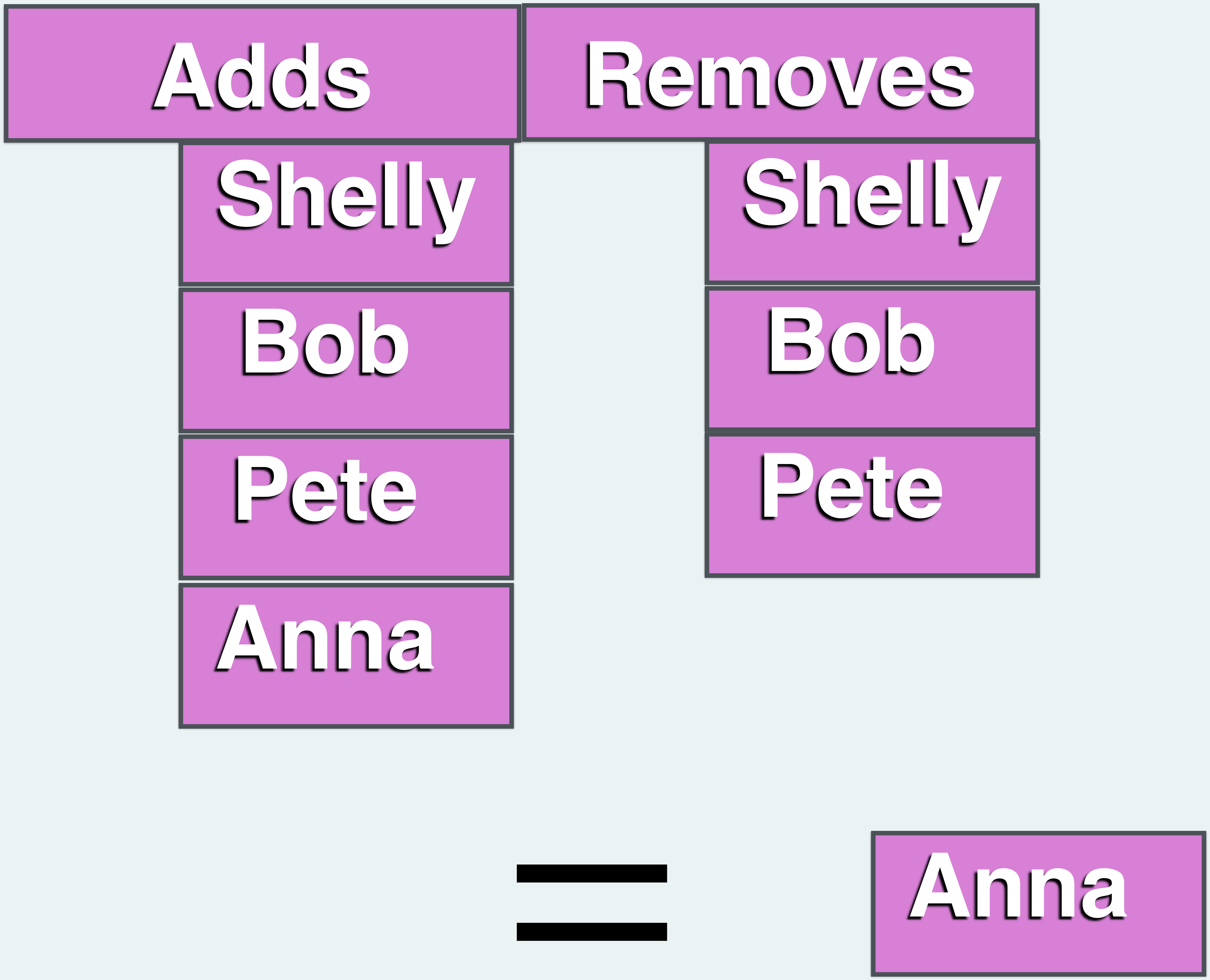
Evolution of a Set

G-SET

2P-SET

Adds	Removes
Shelly	Shelly
Bob	Bob
Pete	Pete
Anna	

2P-SET

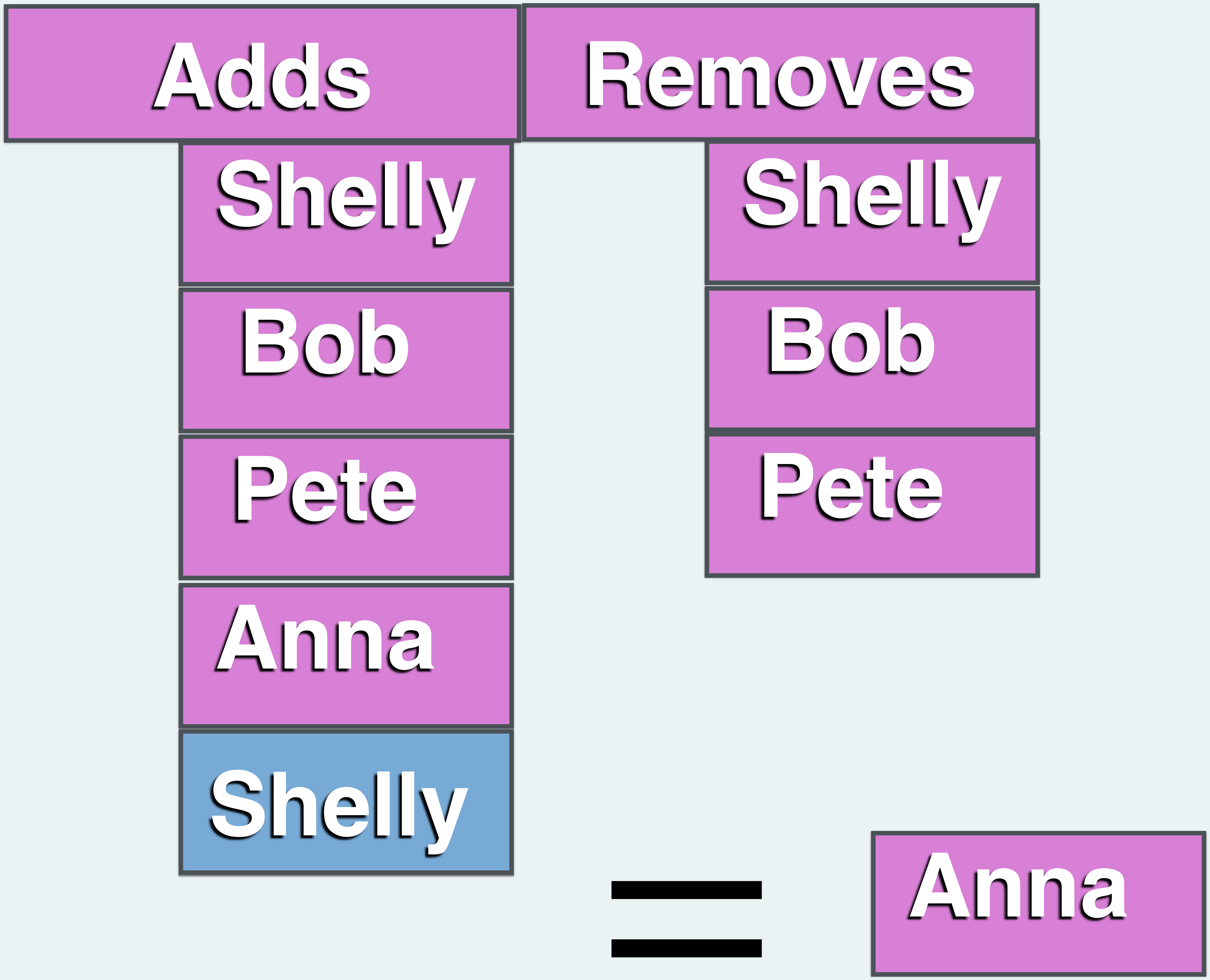

Adds
Shelly
Bob
Pete
Anna
Removes
Shelly
Bob
Pete
=
Anna

Value /= Structure

Adds	Removes
Shelly	Shelly
Bob	Bob
Pete	Pete
Anna	

= Anna

I changed my mind!


Adds
Removes
Shelly
Bob
Pete
Anna
Shelly
Shelly
Bob
Pete
=
Anna

Evolution of a Set

U-SET

Replica A

1	Shelly
2	Bob
3	Pete
4	Anna

$\sqcup$

Replica B

5	Alex
6	Shelly

1,6	Shelly
2	Bob
3	Pete
4	Anna
5	Alex

U-SET

Evolution of a Set

U-SET
OR-SET

Evolution of a Set

~~OR-SET~~

AW-SET

Adds	
1,5	Shelly
2	Bob
3	Pete
4	Anna

Removes	
1	Shelly
2	Bob
3	Pete

AW-SET

Replica A

Adds	
1	Shelly
2	Bob
3	Pete

Removes	
1	Shelly
2	Bob
3	Pete

$\sqcup$

Replica B

Adds	
4	Anna
5	Shelly

Adds	
1,5	Shelly
2	Bob
3	Pete
4	Anna

Removes	
1	Shelly
2	Bob
3	Pete

=

Anna
Shelly

Observed Remove

Semantics

Add Wins

Evolution of a Set

AW-SET

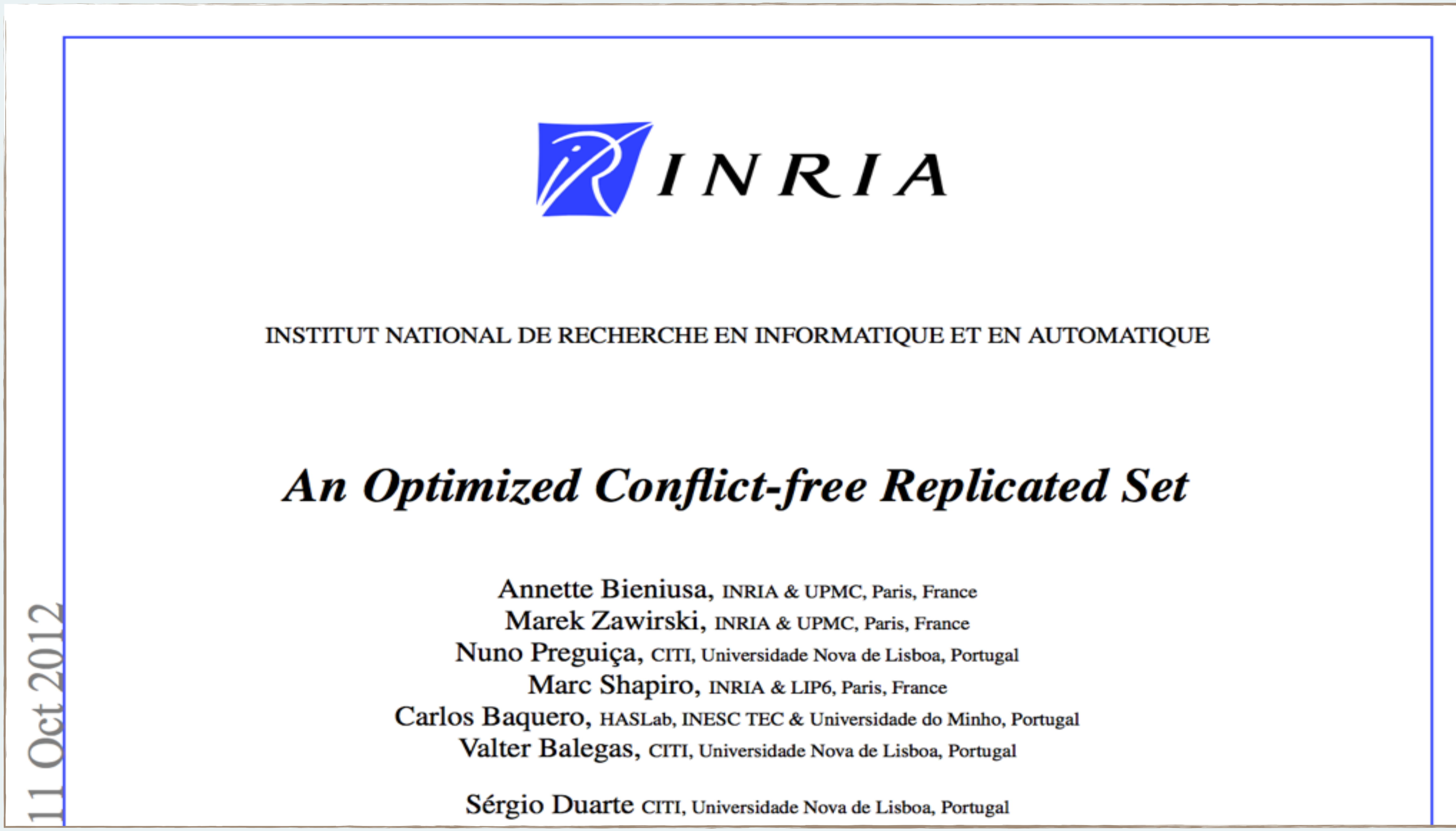
Adds	
1,5	Shelly
2	Bob
3	Pete
4	Anna

Removes	
1,5	Shelly
2	Bob
3	Pete
4	Anna

=

Shelly

[]



INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

An Optimized Conflict-free Replicated Set

Annette Bieniusa, INRIA & UPMC, Paris, France
Marek Zawirski, INRIA & UPMC, Paris, France
Nuno Preguiça, CITI, Universidade Nova de Lisboa, Portugal
Marc Shapiro, INRIA & LIP6, Paris, France
Carlos Baquero, HASLab, INESC TEC & Universidade do Minho, Portugal
Valter Balegas, CITI, Universidade Nova de Lisboa, Portugal

Sérgio Duarte CITI, Universidade Nova de Lisboa, Portugal

11 Oct 2012

Evolution of a Set

OR-SWOT

Evolution of a Set

~~OR-SWOT~~

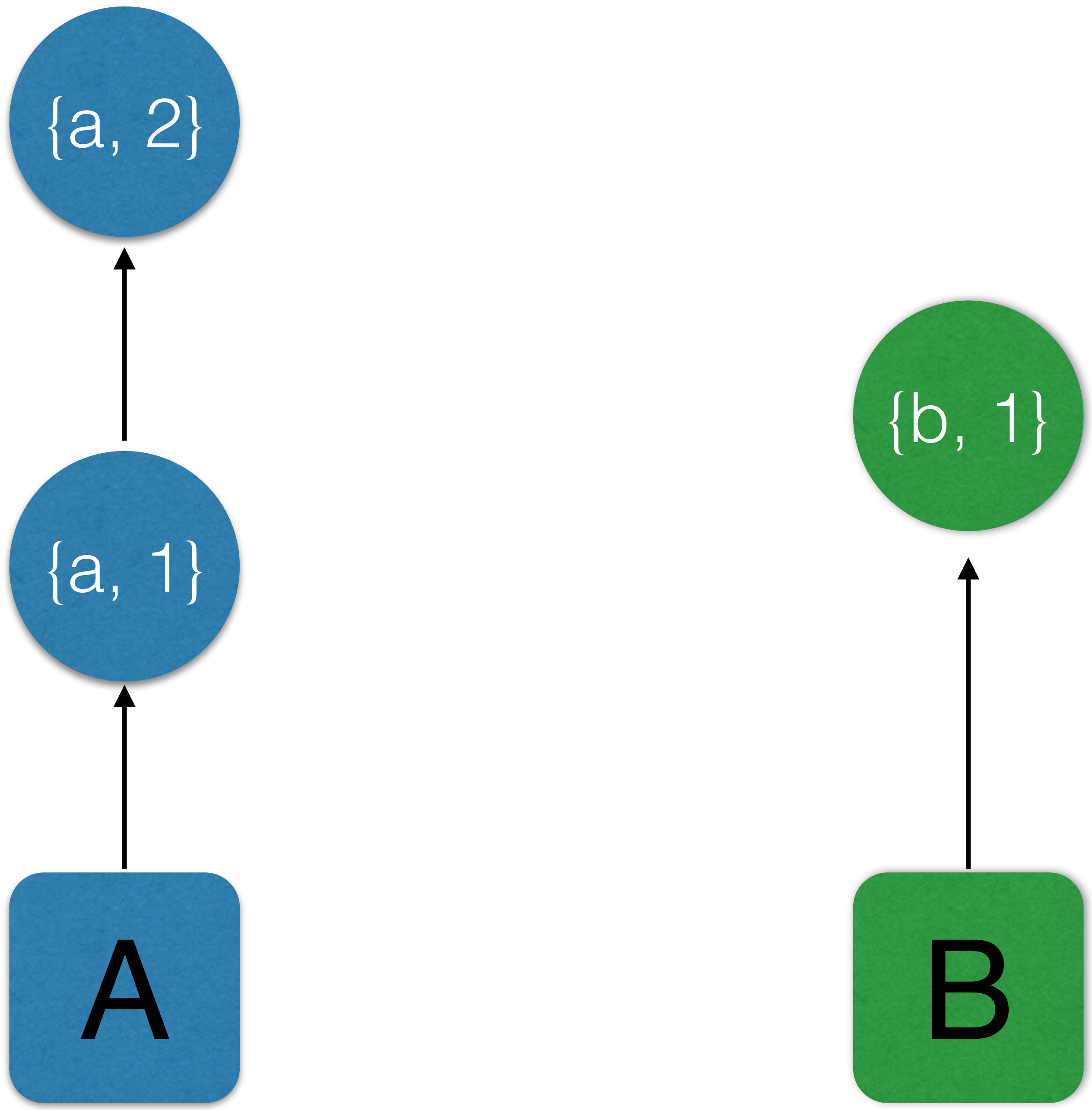
Optimised

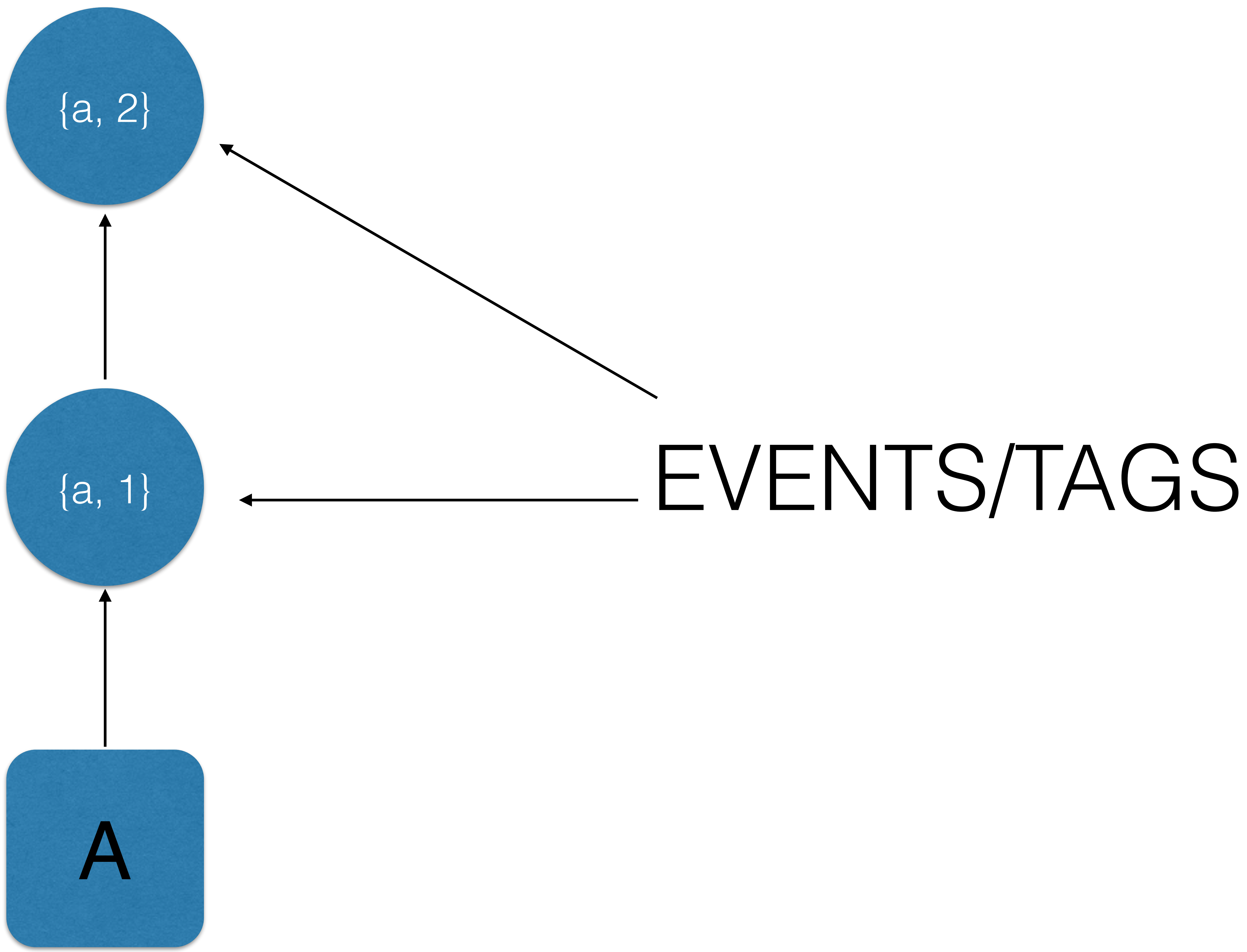
AW-SET

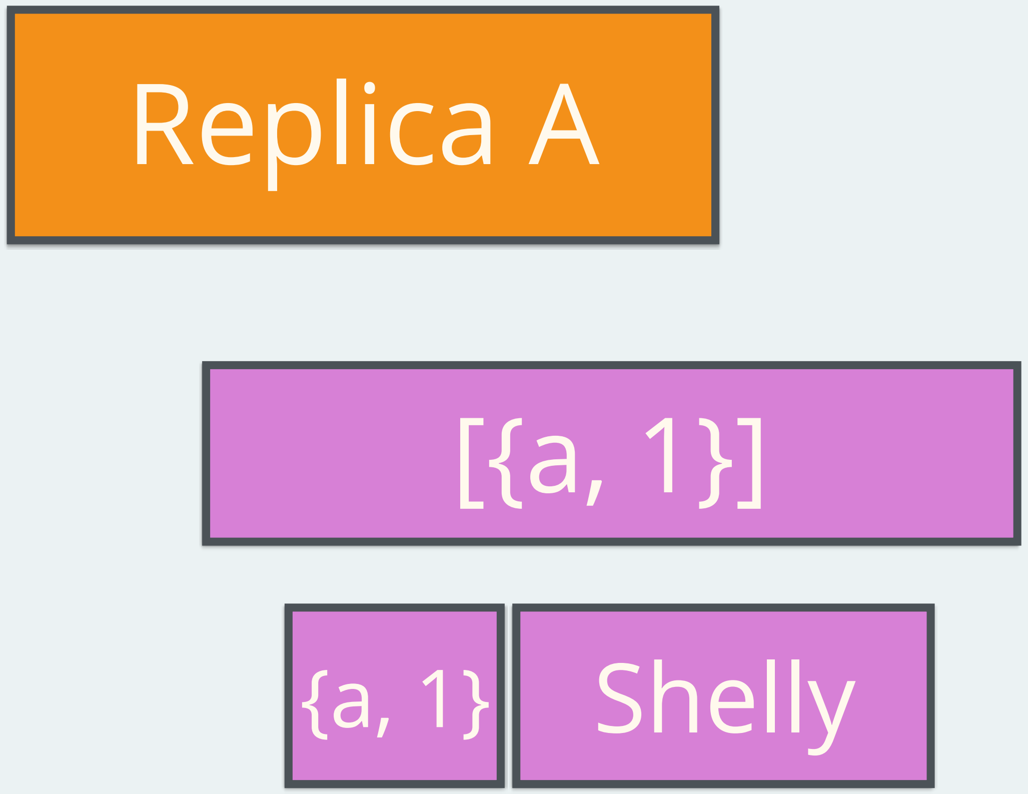
Version Vectors

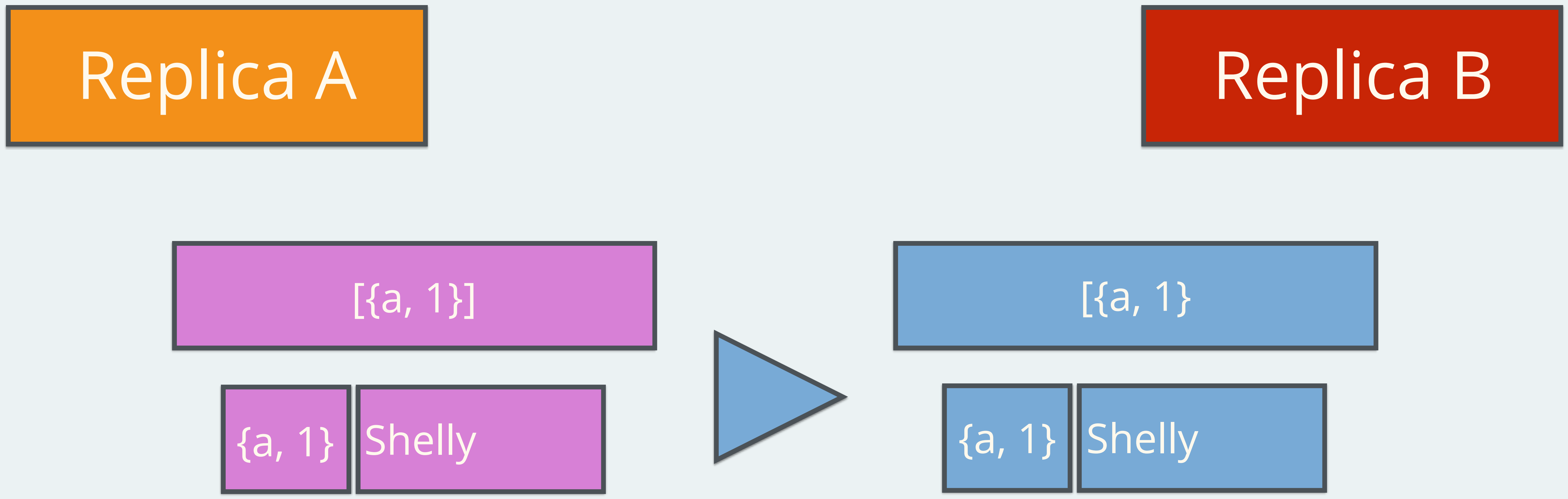
Version Vectors

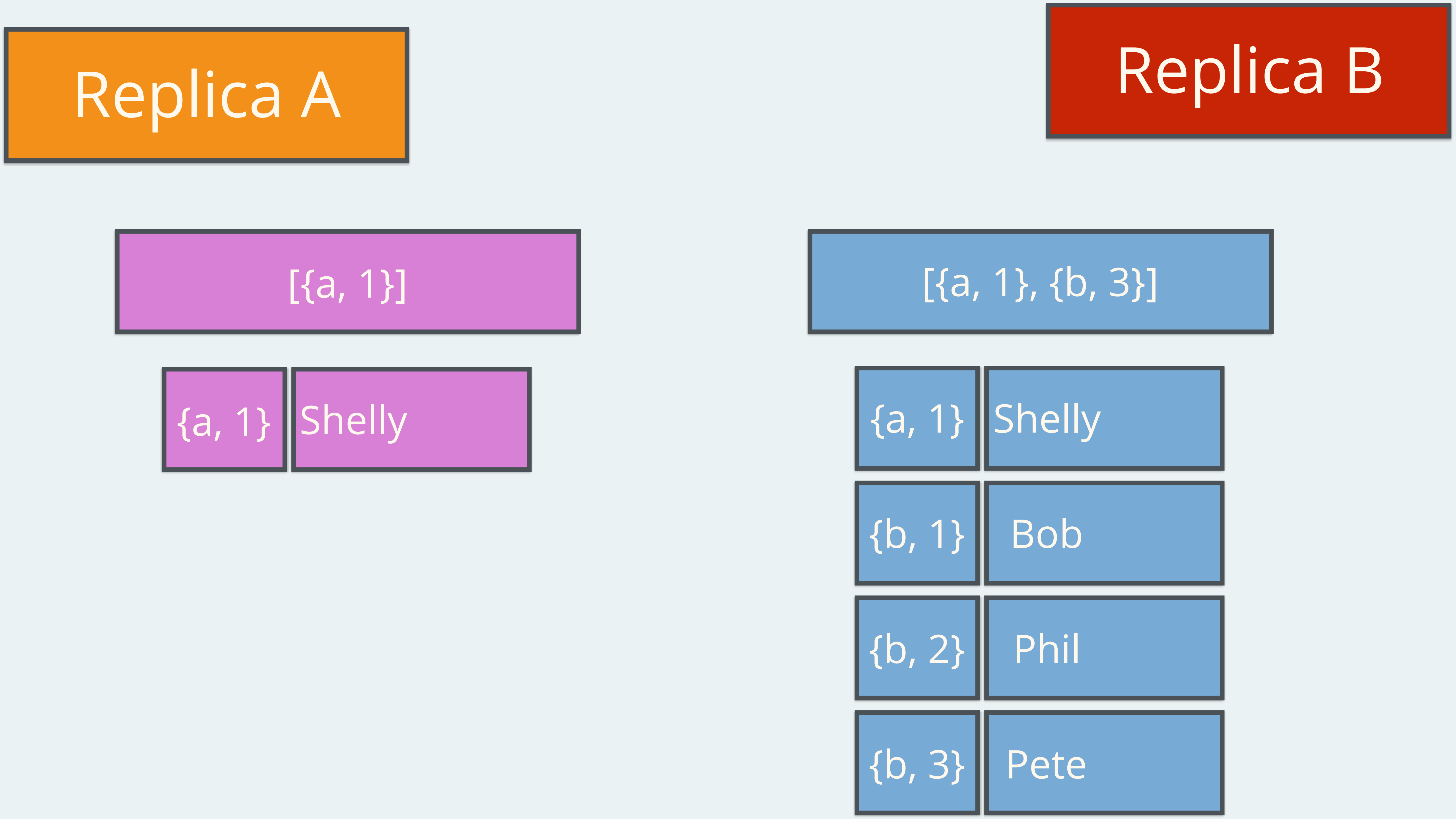
[{a, 2}, {b, 1}]

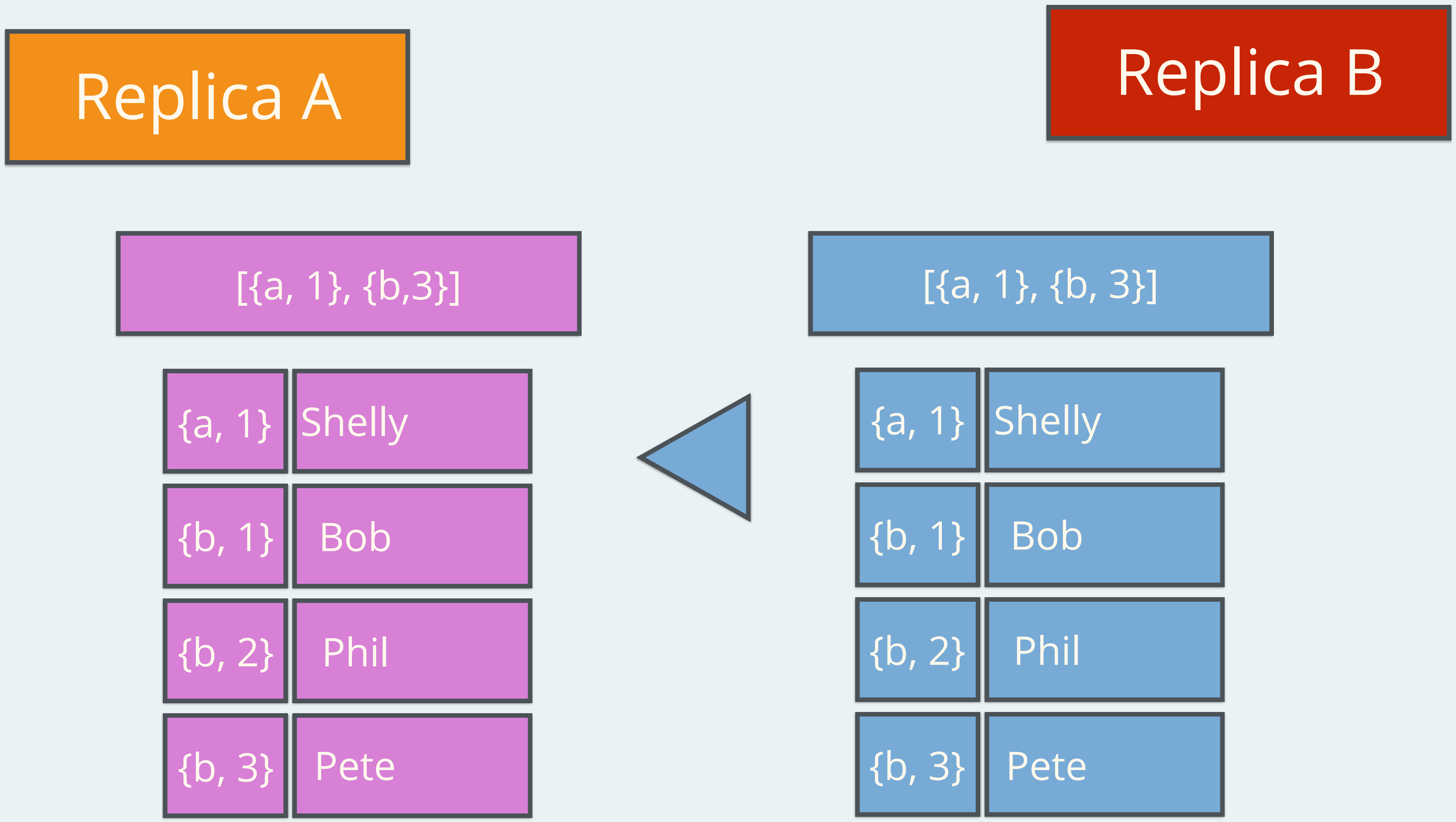


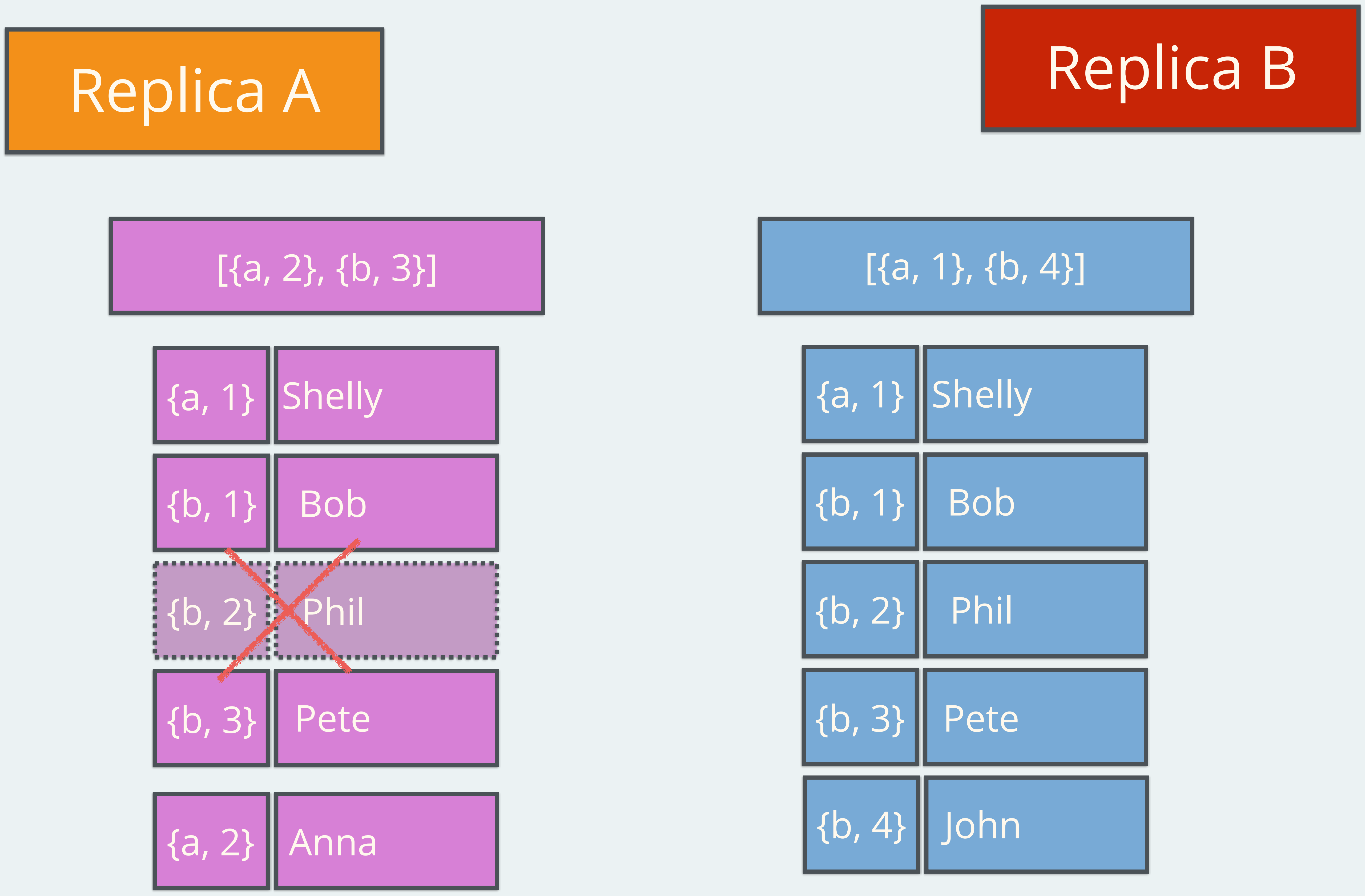

{a, 2}
{a, 1}
A
EVENTS/TAGS

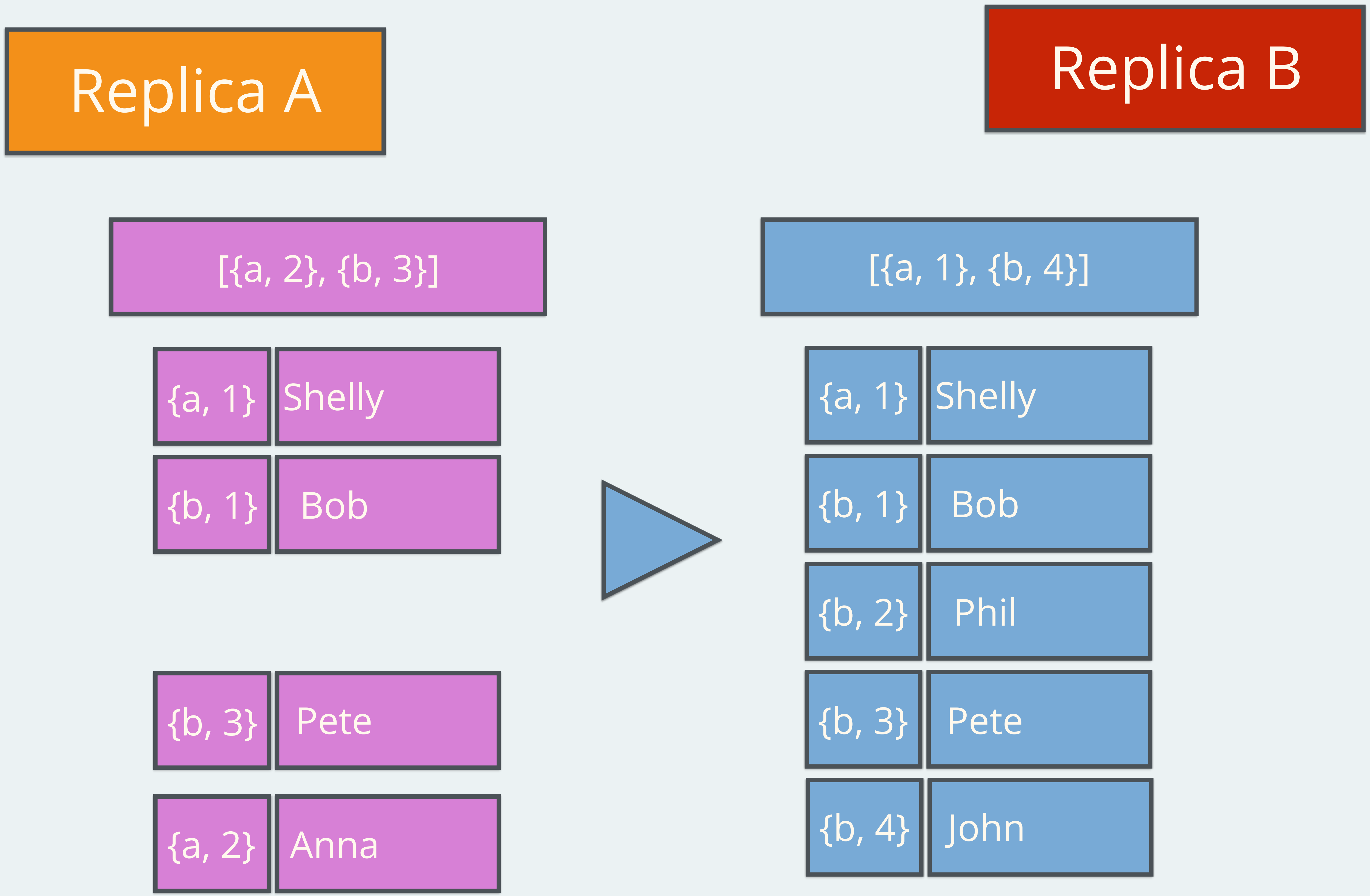

Replica A
[{a, 1}]
{a, 1}
Shelly


Replica A
Replica B
[{a, 1}]
{a, 1}
Shelly
[{a, 1}
{a, 1}
Shelly


Replica A
Replica B
[{a, 1}]
[{a, 1}, {b, 3}]
{a, 1}
Shelly
{a, 1}
Shelly
{b, 1}
Bob
{b, 2}
Phil
{b, 3}
Pete


Replica A
Replica B
[{a, 1}, {b,3}]
[{a, 1}, {b, 3}]
{a, 1} Shelly
{b, 1} Bob
{b, 2} Phil
{b, 3} Pete
{a, 1} Shelly
{b, 1} Bob
{b, 2} Phil
{b, 3} Pete


Replica A
Replica B
[{a, 2}, {b, 3}]
{a, 1} Shelly
{b, 1} Bob
~~{b, 2} Phil~~
{b, 3} Pete
{a, 2} Anna
[{a, 1}, {b, 4}]
{a, 1} Shelly
{b, 1} Bob
{b, 2} Phil
{b, 3} Pete
{b, 4} John


Replica A
Replica B
[{a, 2}, {b, 3}]
{a, 1} Shelly
{b, 1} Bob
{b, 3} Pete
{a, 2} Anna
[{a, 1}, {b, 4}]
{a, 1} Shelly
{b, 1} Bob
{b, 2} Phil
{b, 3} Pete
{b, 4} John

Replica A

[{a, 2}, {b, 3}]

{a, 1}	Shelly
{b, 1}	Bob
{b, 3}	Pete
{a, 2}	Anna

Replica B

[{a, 1}, {b, 4}]

{a, 1}	Shelly
{b, 1}	Bob
{b, 2}	Phil
{b, 3}	Pete
{b, 4}	John

MERGE

Replica A		Replica B		MERGE	
[{a, 2}, {b, 3}]		[{a, 1}, {b, 4}]			
{a, 1}	Shelly	{a, 1}	Shelly	{a, 1}	Shelly
{b, 1}	Bob	{b, 1}	Bob	{b, 1}	Bob
		{b, 2}	Phil		
{b, 3}	Pete	{b, 3}	Pete		
{a, 2}	Anna	{b, 4}	John		

{b, 2} ∈ [{a, 2}, {b, 3}]

Phil

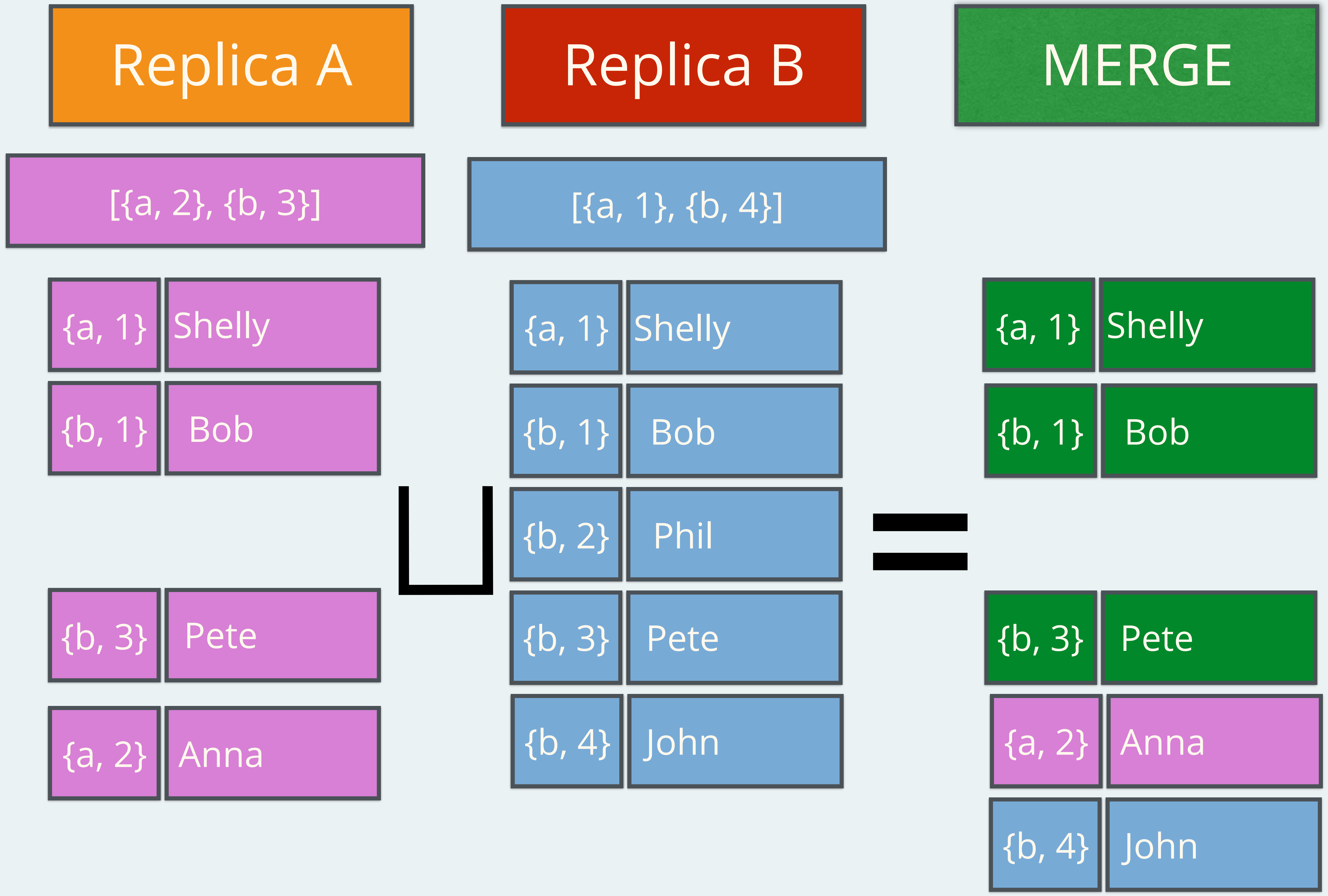
{b, 2} ∈ [{a, 2}, {b, 3}]

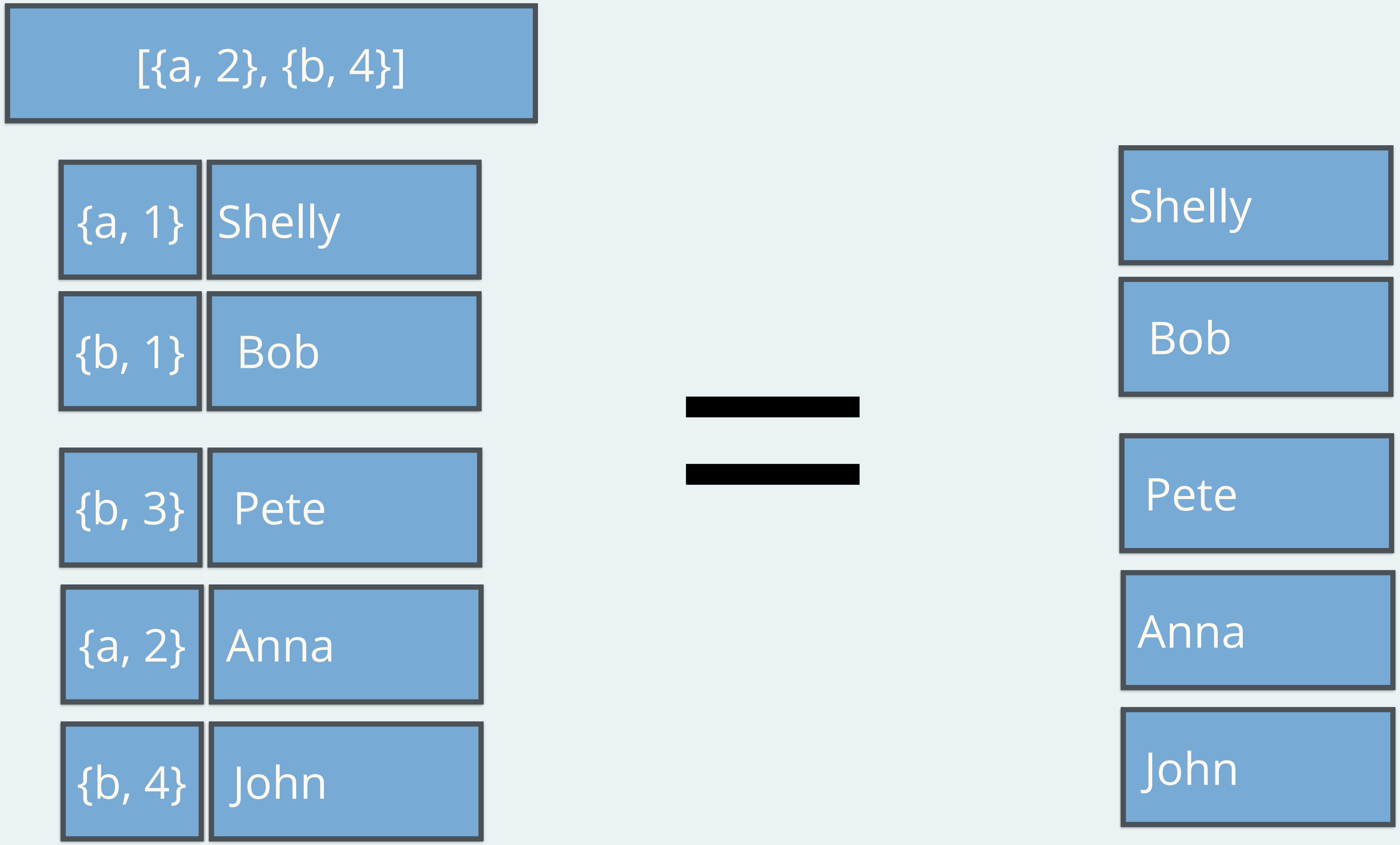
~~Phil~~

Replica A		Replica B		MERGE	
[{a, 2}, {b, 3}]		[{a, 1}, {b, 4}]			
{a, 1}	Shelly	{a, 1}	Shelly	{a, 1}	Shelly
{b, 1}	Bob	{b, 1}	Bob	{b, 1}	Bob
		{b, 2}	Phil		
{b, 3}	Pete	{b, 3}	Pete	{b, 3}	Pete
{a, 2}	Anna	{b, 4}	John		

{a, 2} $\notin$ [{a, 1}, {b, 4}]

Anna

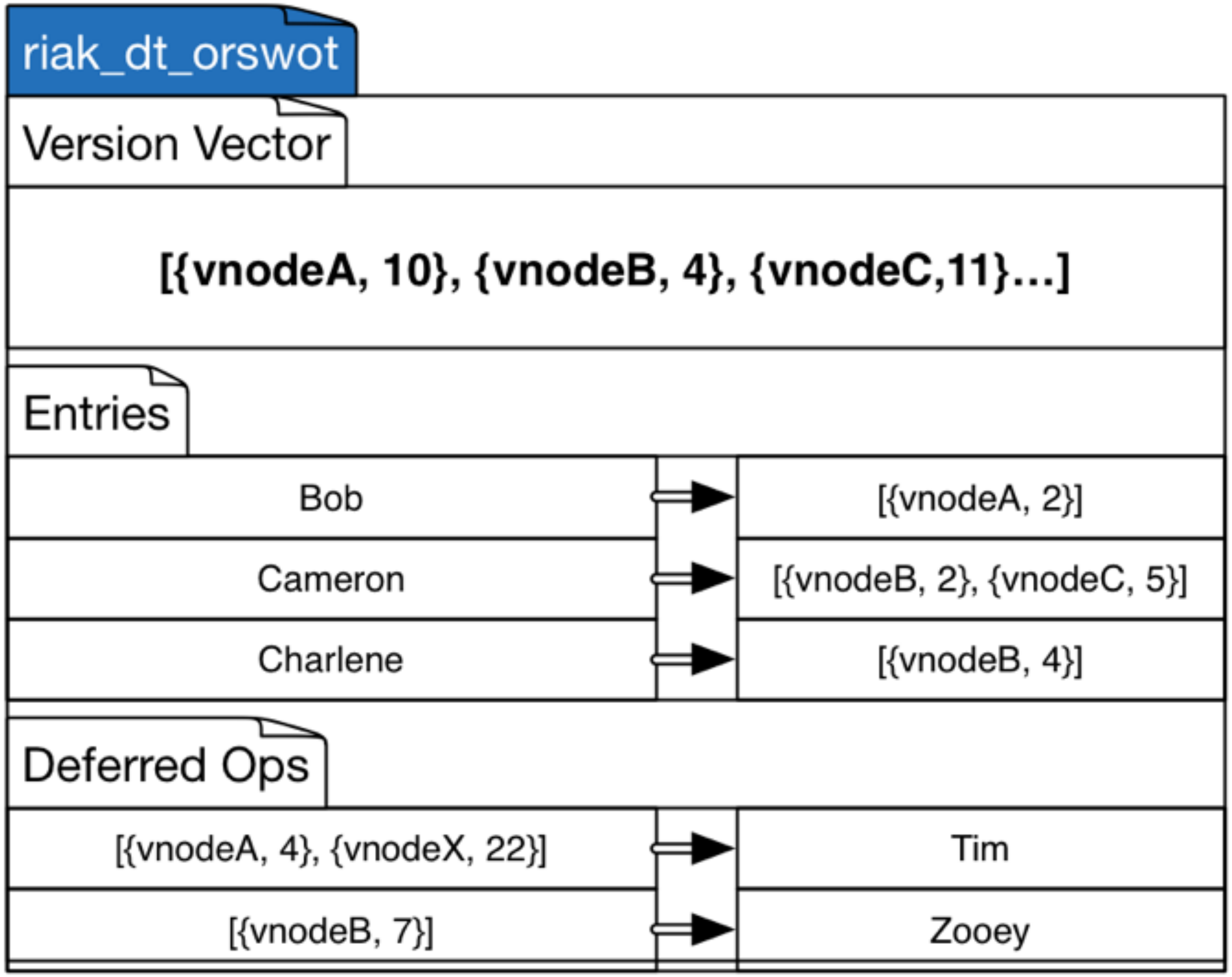

Replica A
Replica B
MERGE
[{a, 2}, {b, 3}]
{a, 1} Shelly
{b, 1} Bob
{b, 3} Pete
{a, 2} Anna
⊔
[{a, 1}, {b, 4}]
{a, 1} Shelly
{b, 1} Bob
{b, 2} Phil
{b, 3} Pete
{b, 4} John
=
{a, 1} Shelly
{b, 1} Bob
{b, 3} Pete
{a, 2} Anna
{b, 4} John


[{a, 2}, {b, 4}]
{a, 1}
Shelly
{b, 1}
Bob
{b, 3}
Pete
{a, 2}
Anna
{b, 4}
John
=
Shelly
Bob
Pete
Anna
John

CRDT Sets

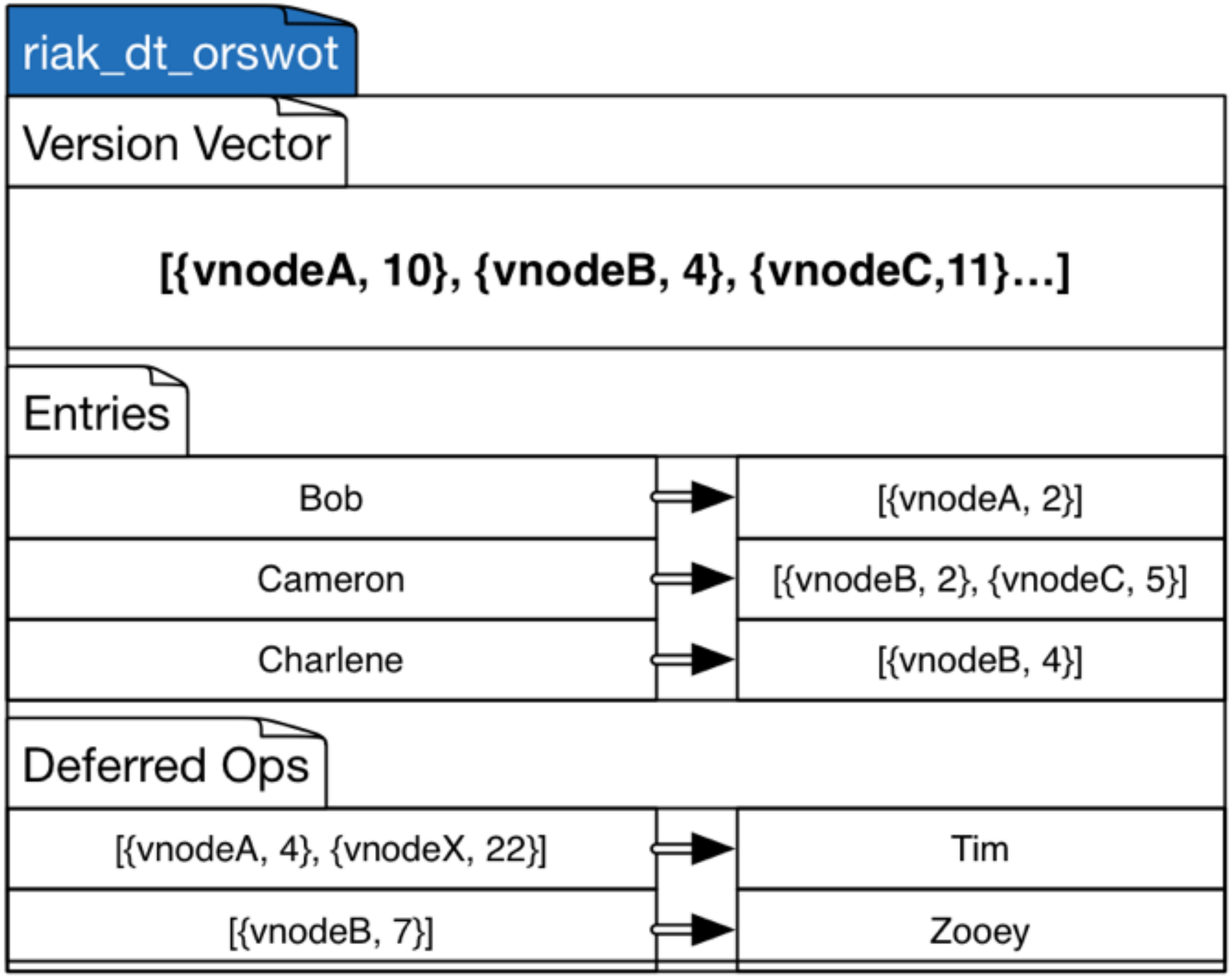
a semantic of “Add-Wins”
via
“Observed Remove”

SETS in RIAK 2.0+

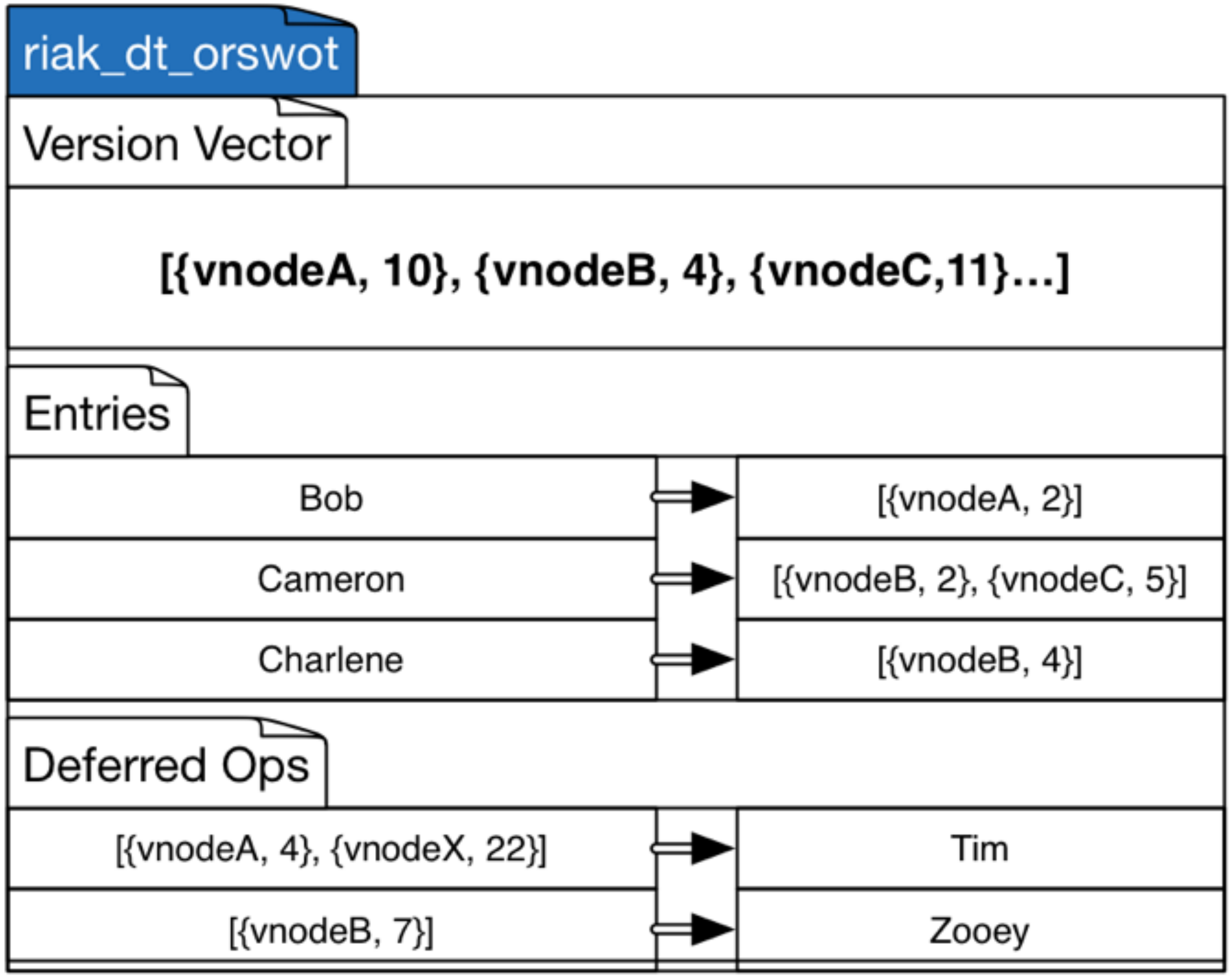


Riak 2.0

riak_dt -> Riak Data Types

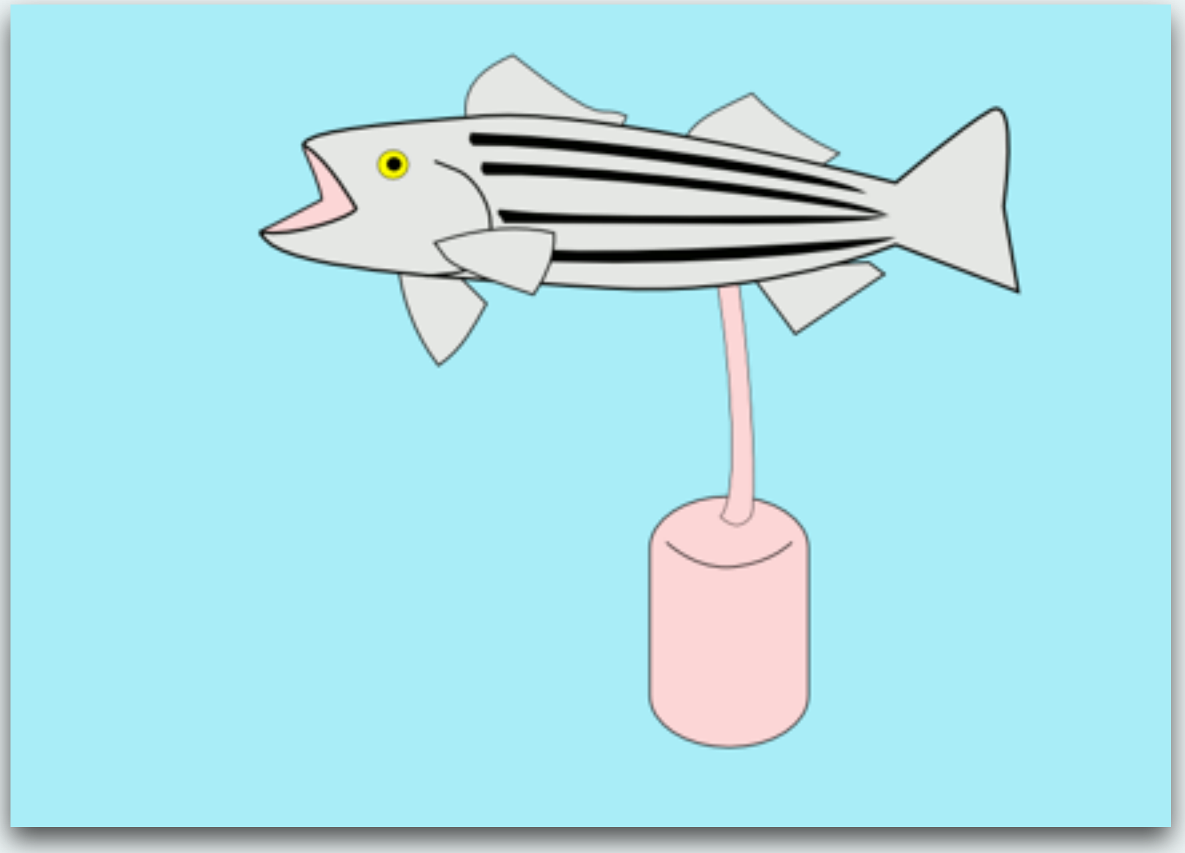


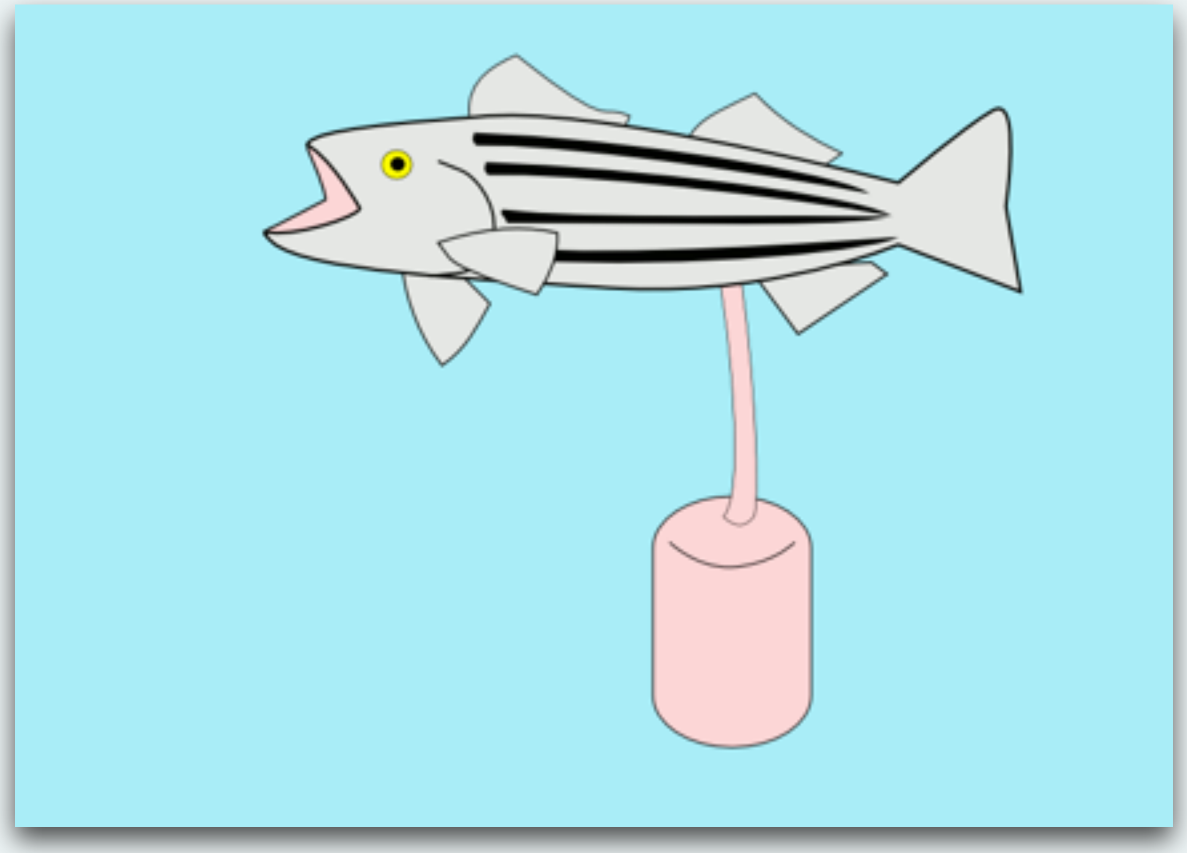
Sets in Riak



An optimized conflict-free replicated set
Annette Bieniusa et al
http://arxiv.org/abs/1210.3368

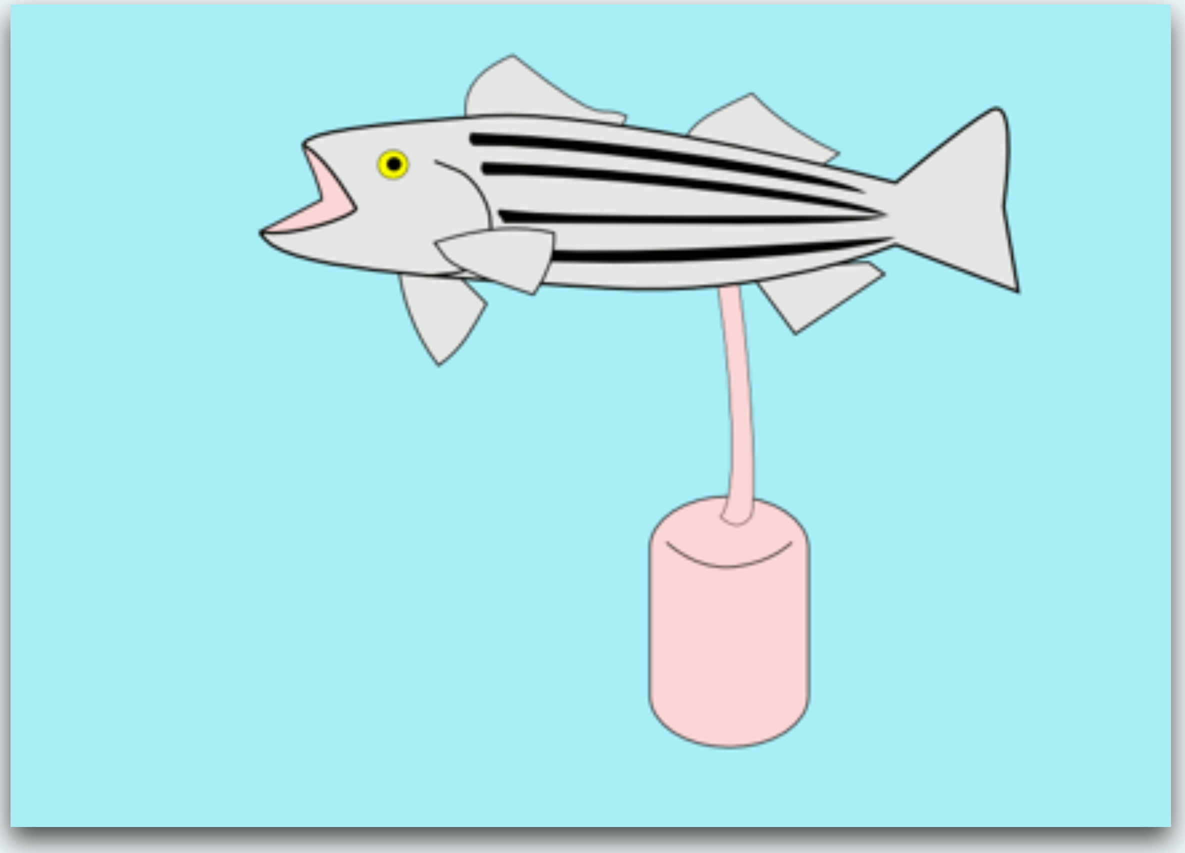
WHO USES THE LIB?

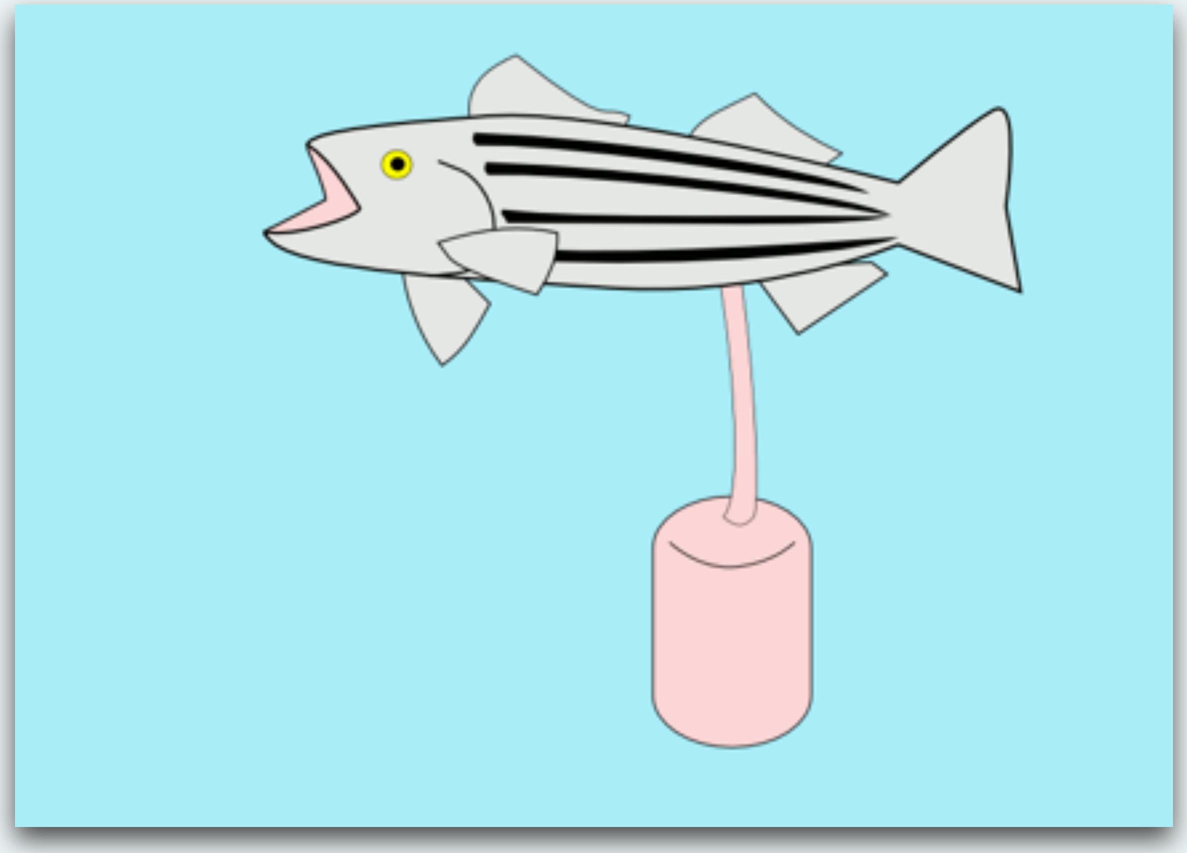

A

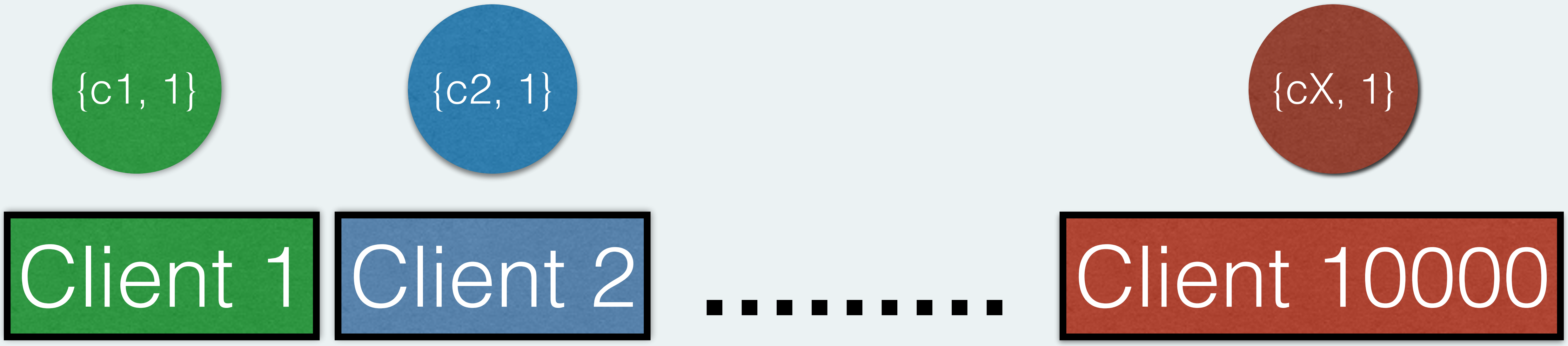

B

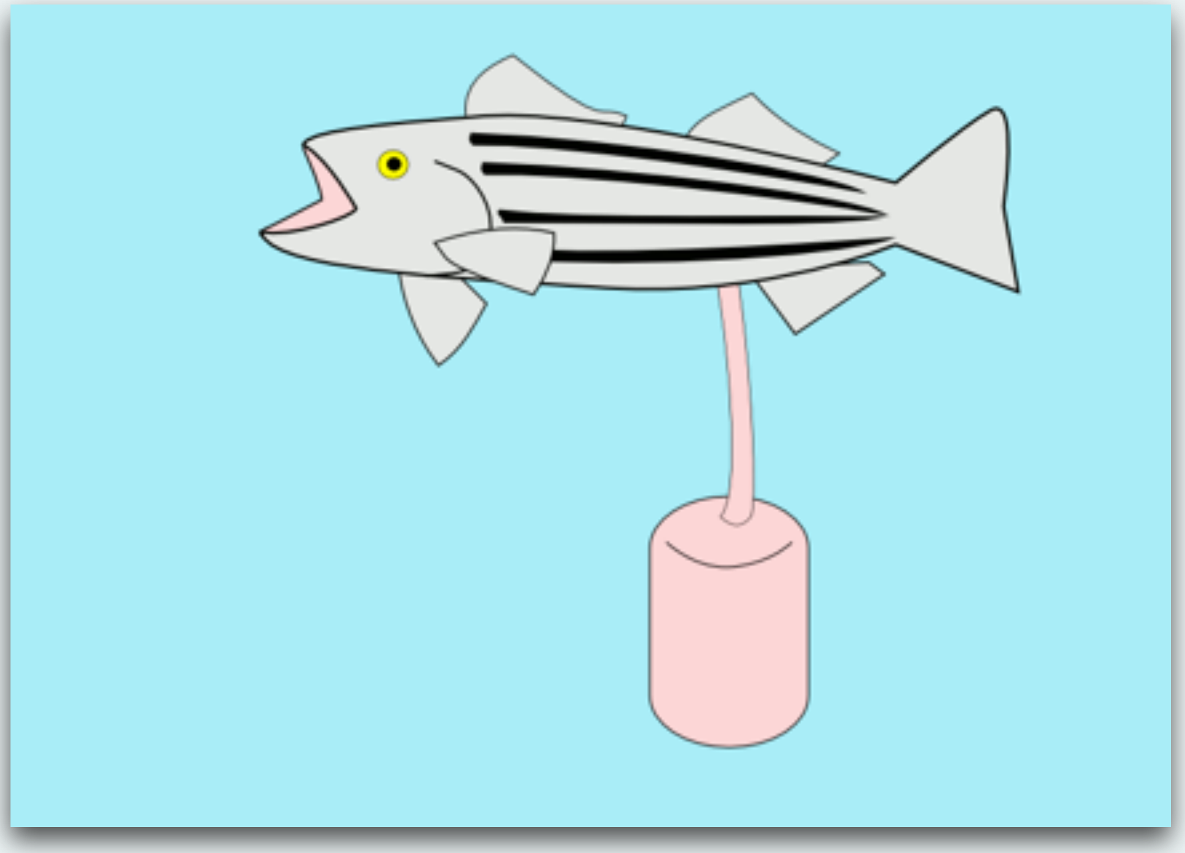
who’s the actor?

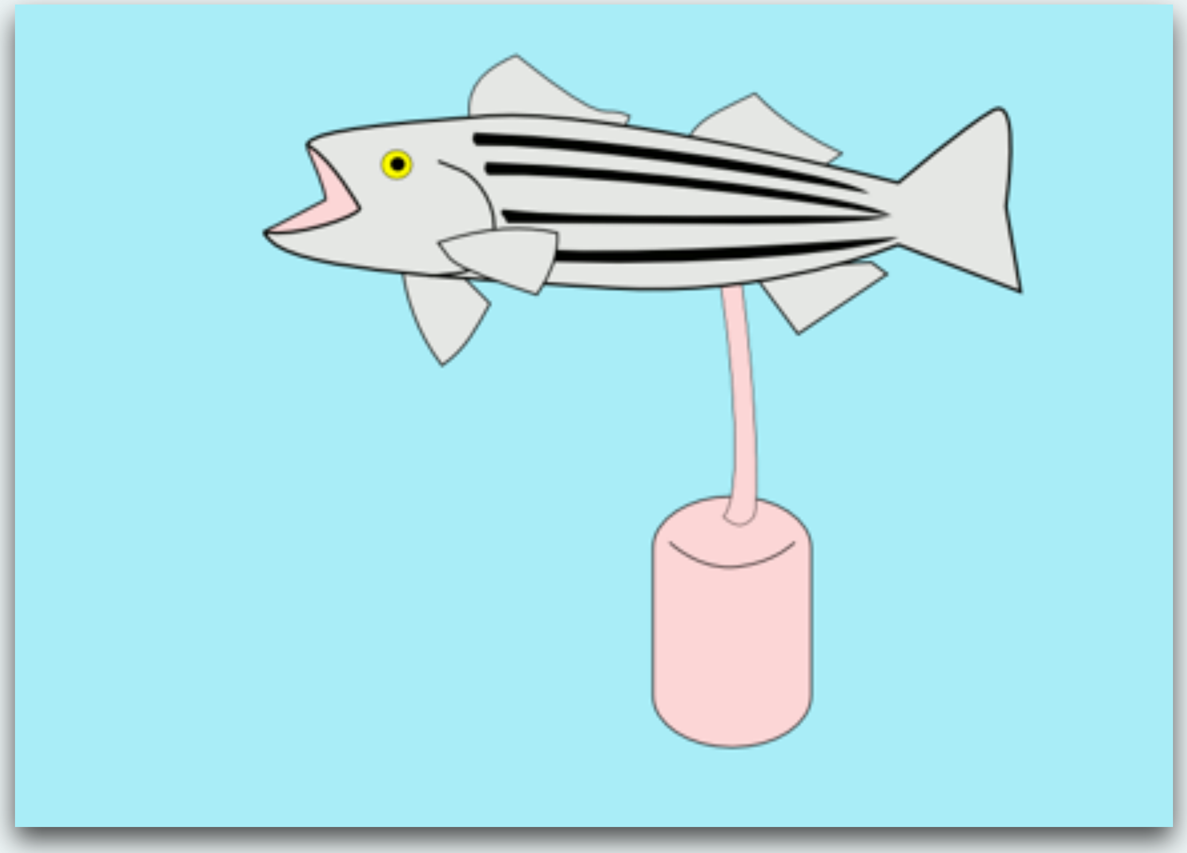
Client 1 Client 2 .... Client 10000

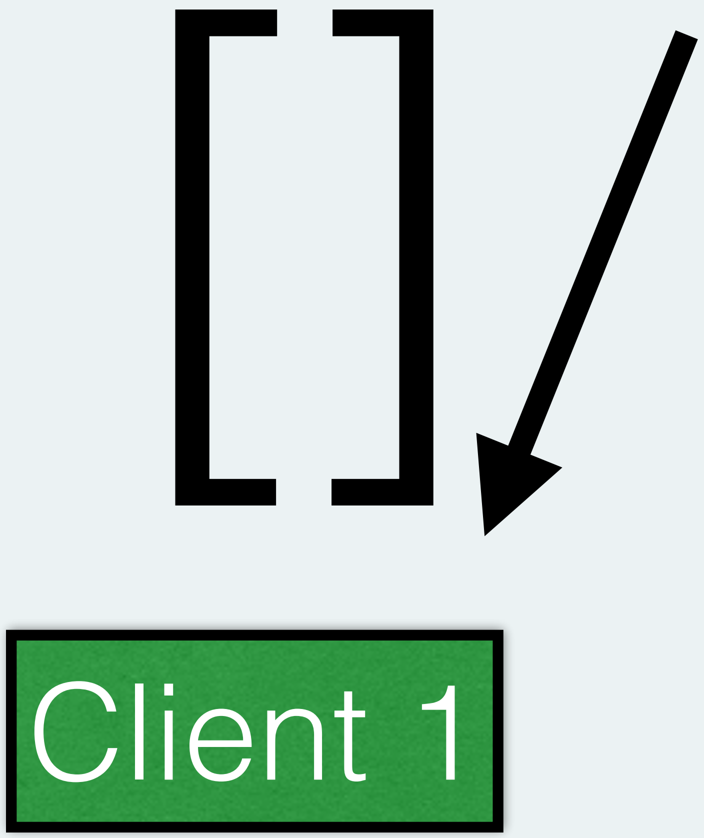

A

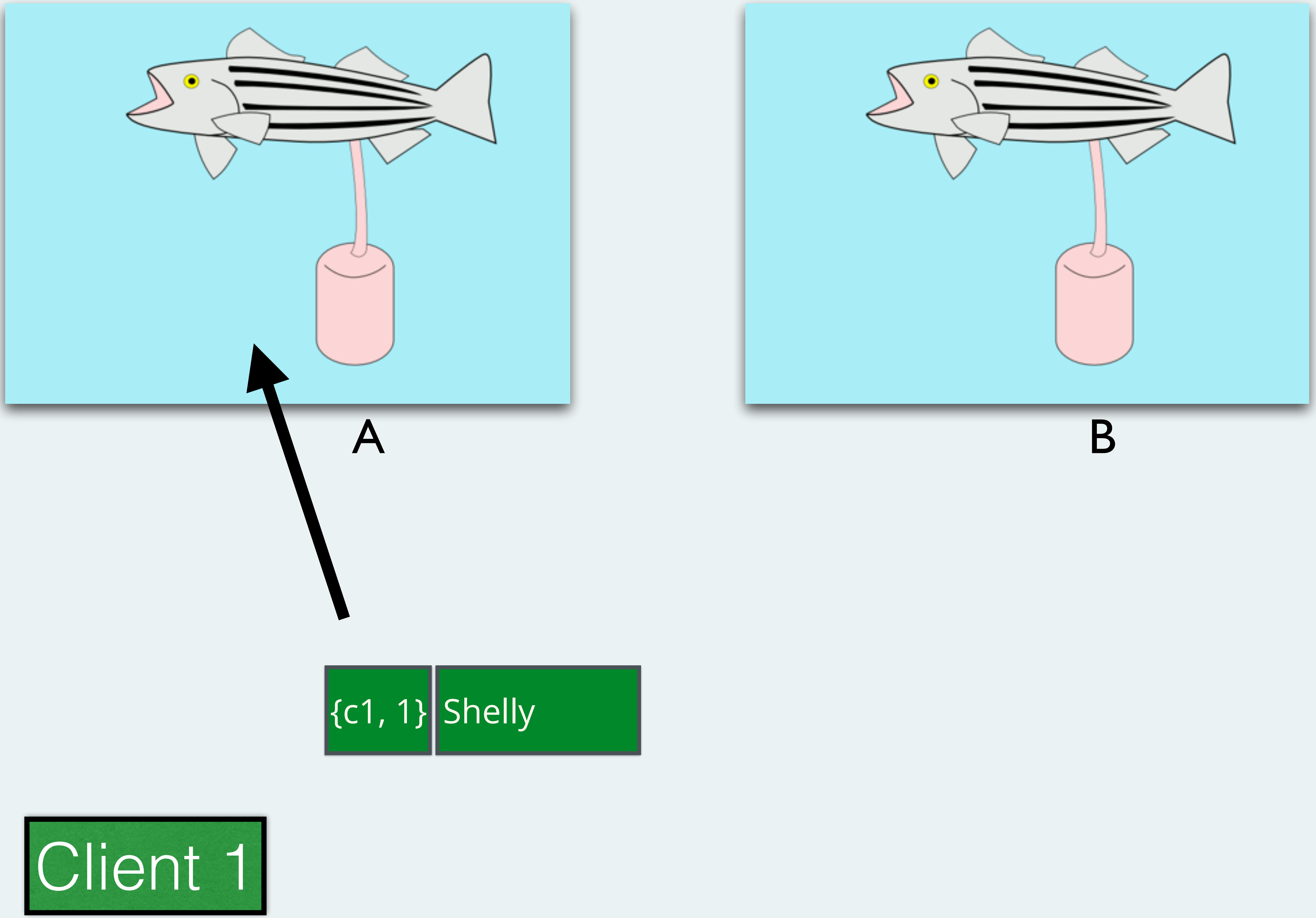

B

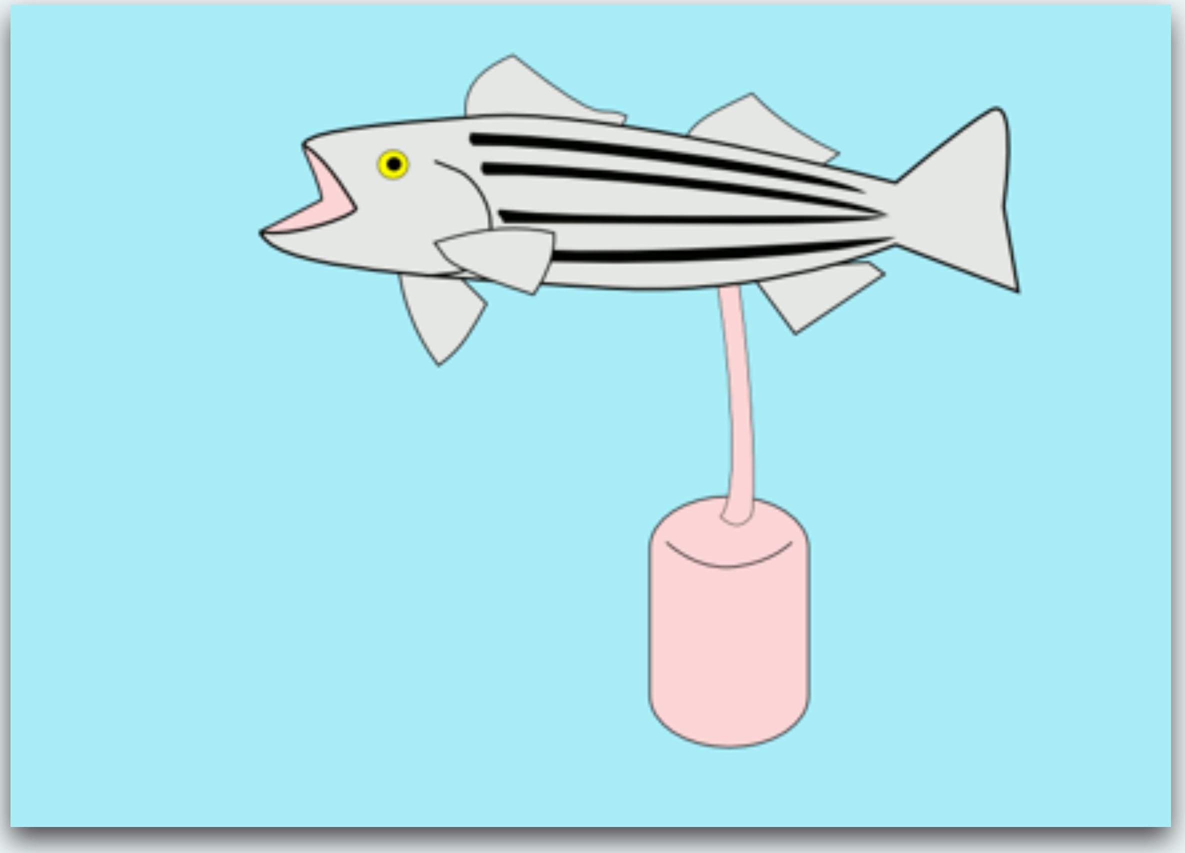


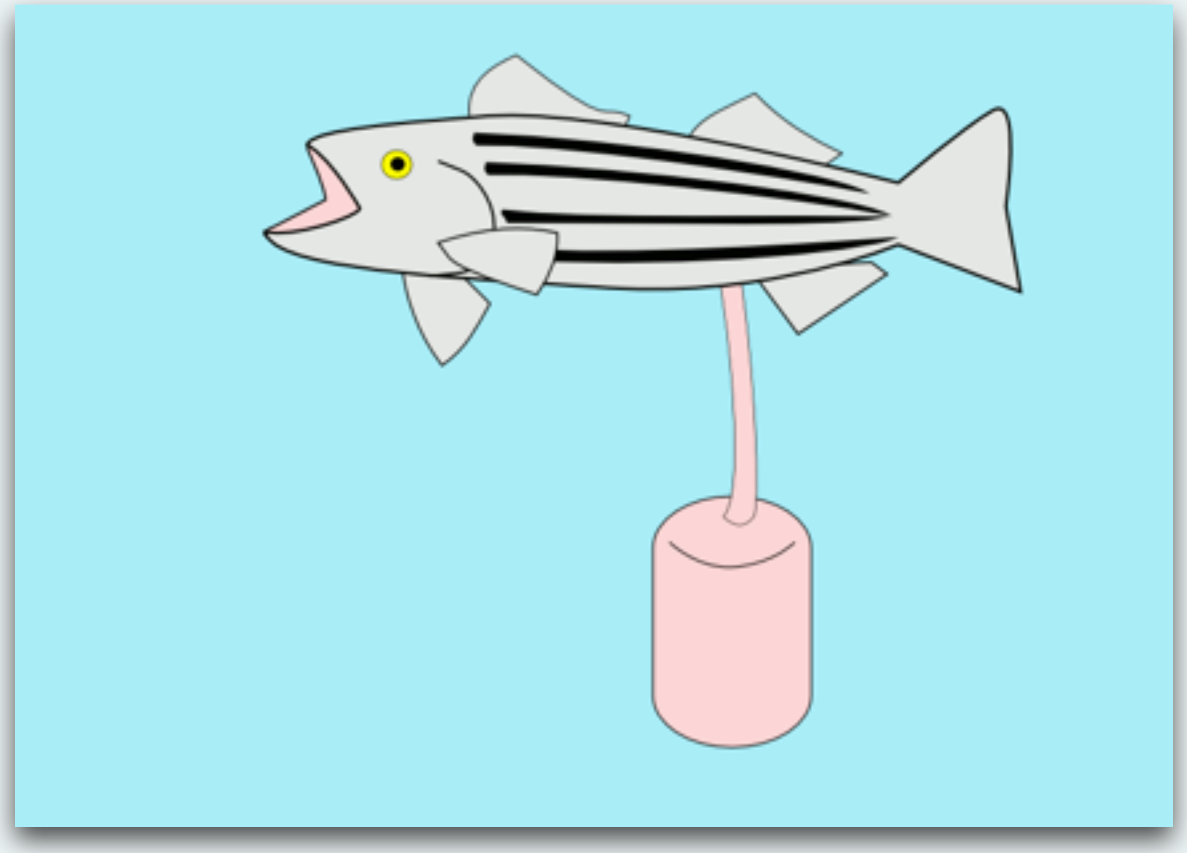

A

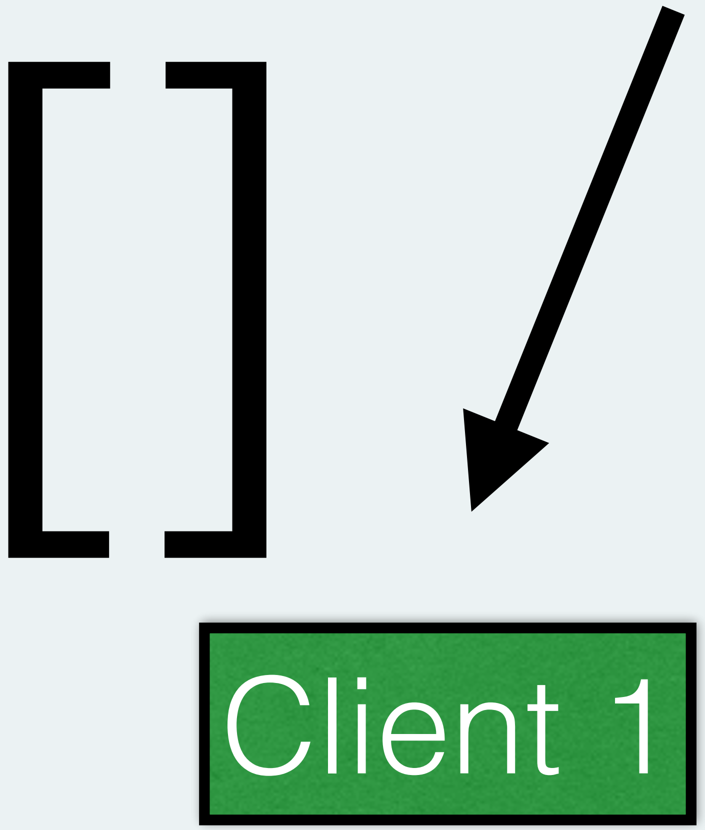

B

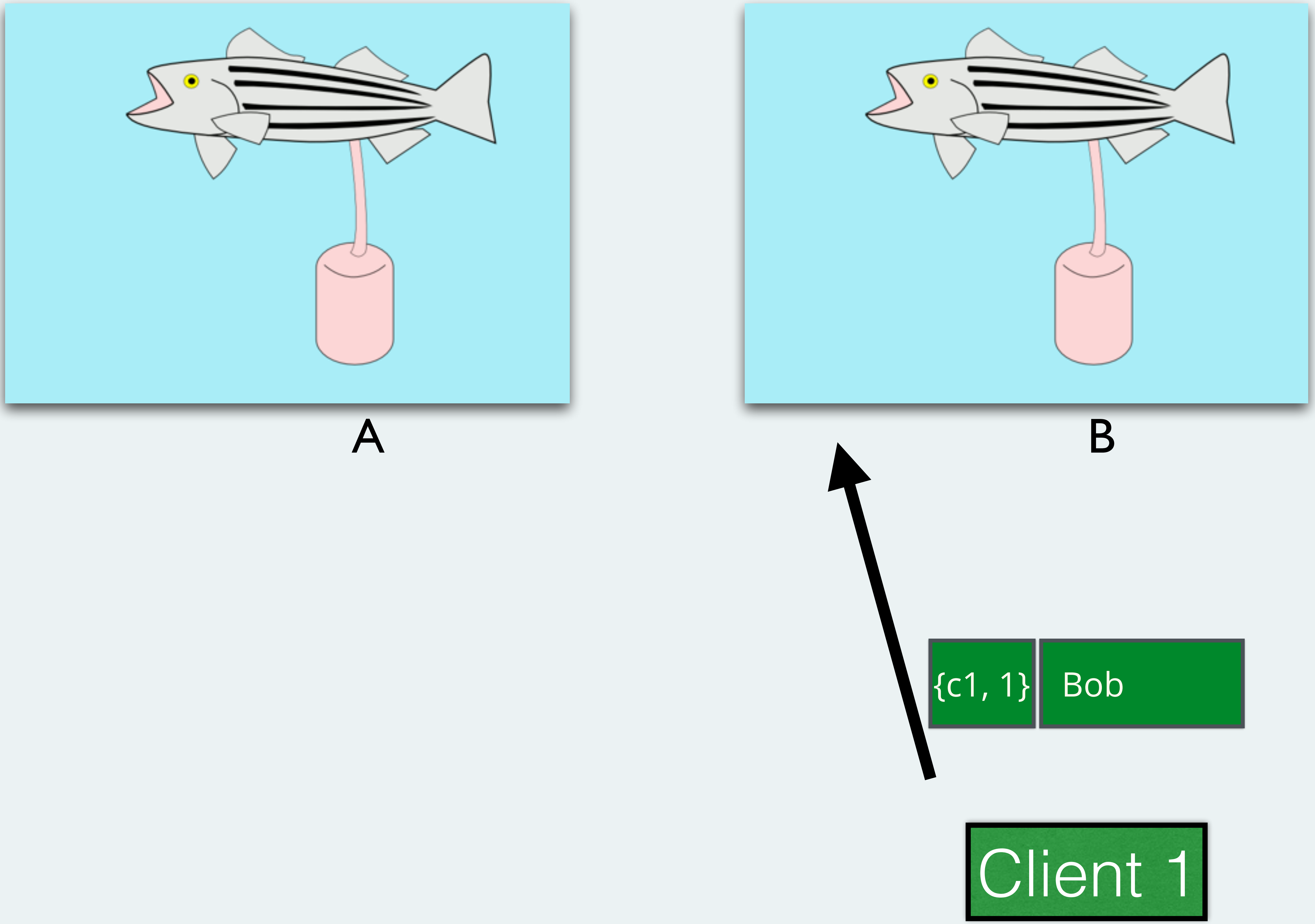



A
B
{c1, 1}
Shelly
Client 1

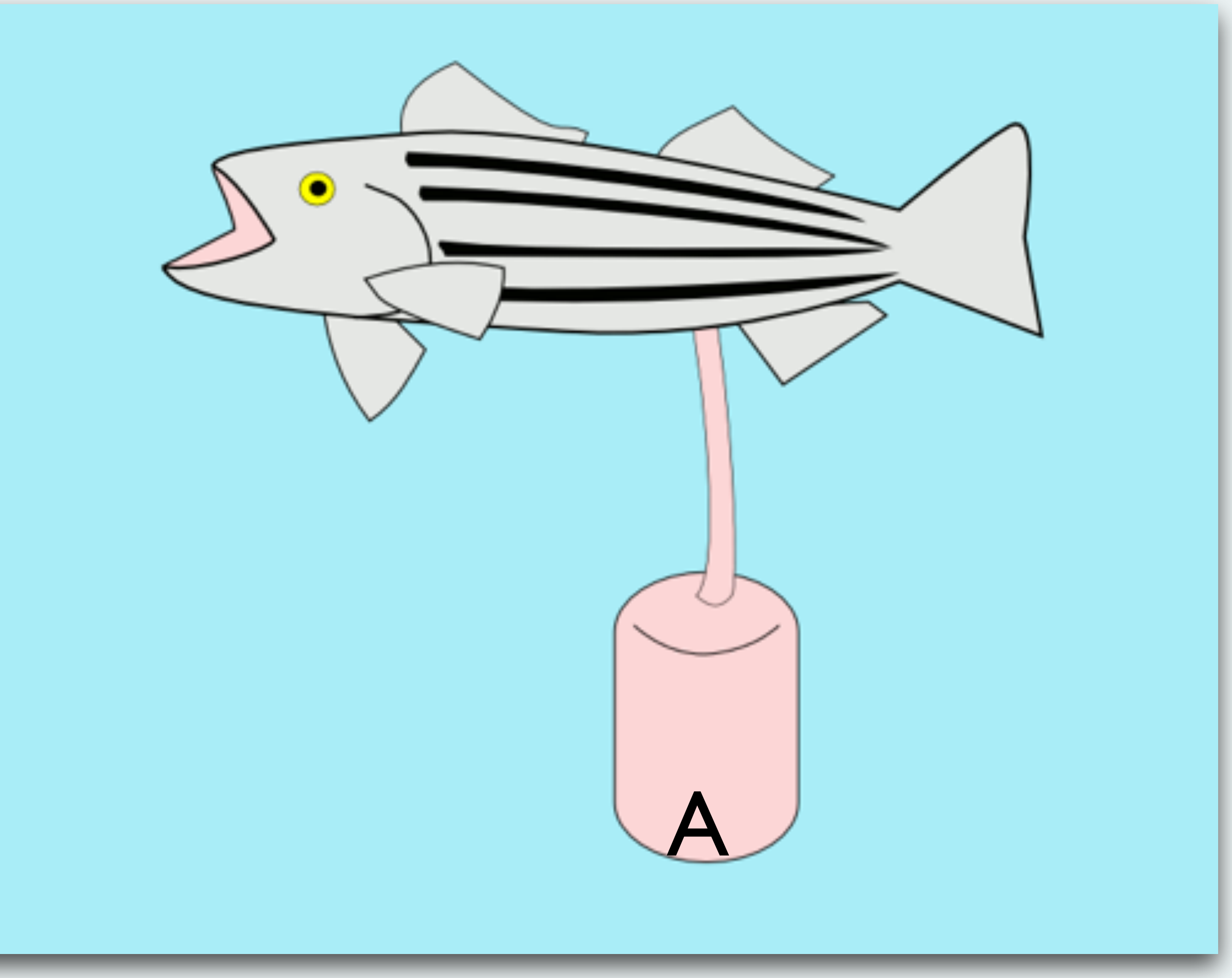

A

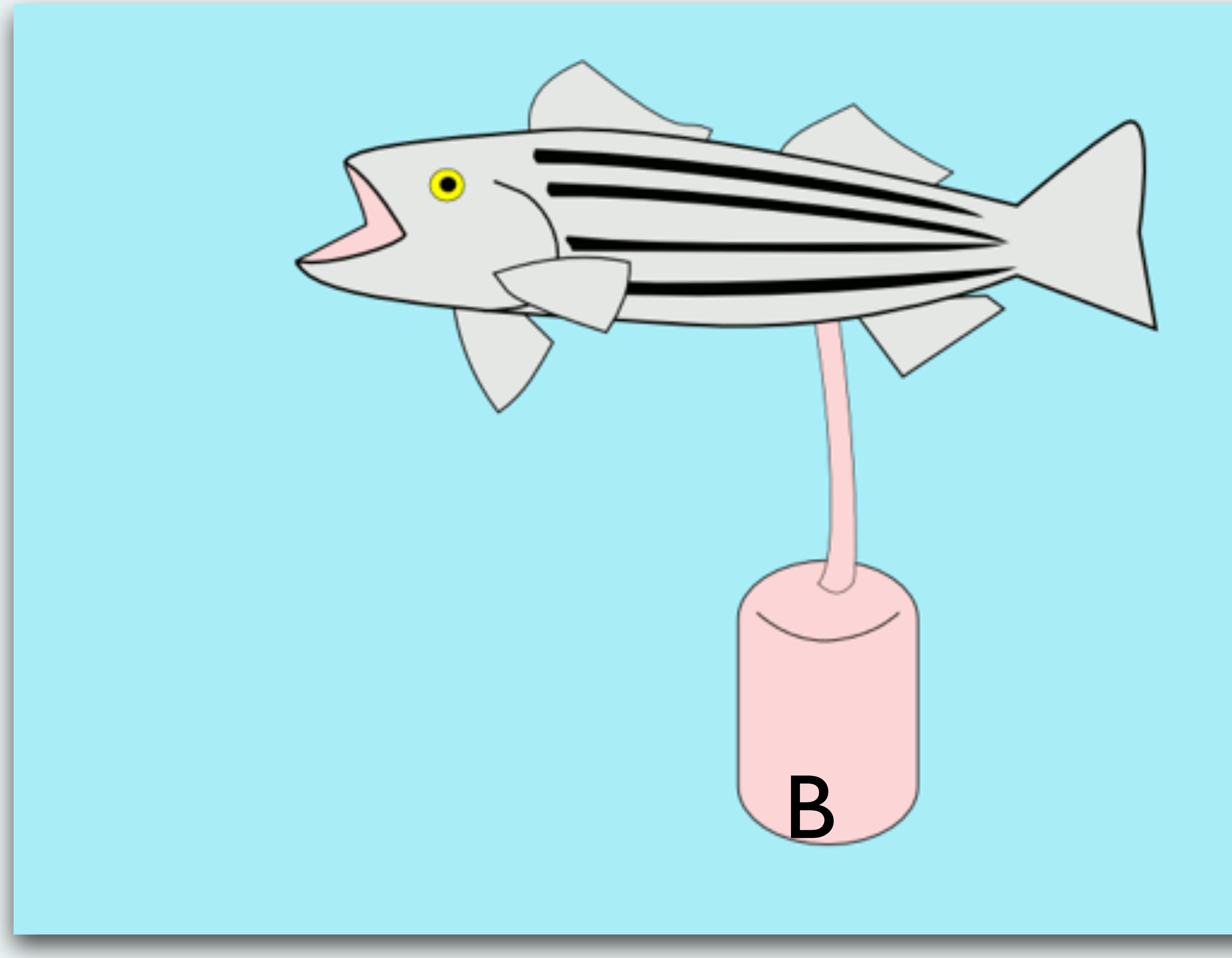

B




A
B
{c1, 1}
Bob
Client 1

Read
Your
Own
Writes

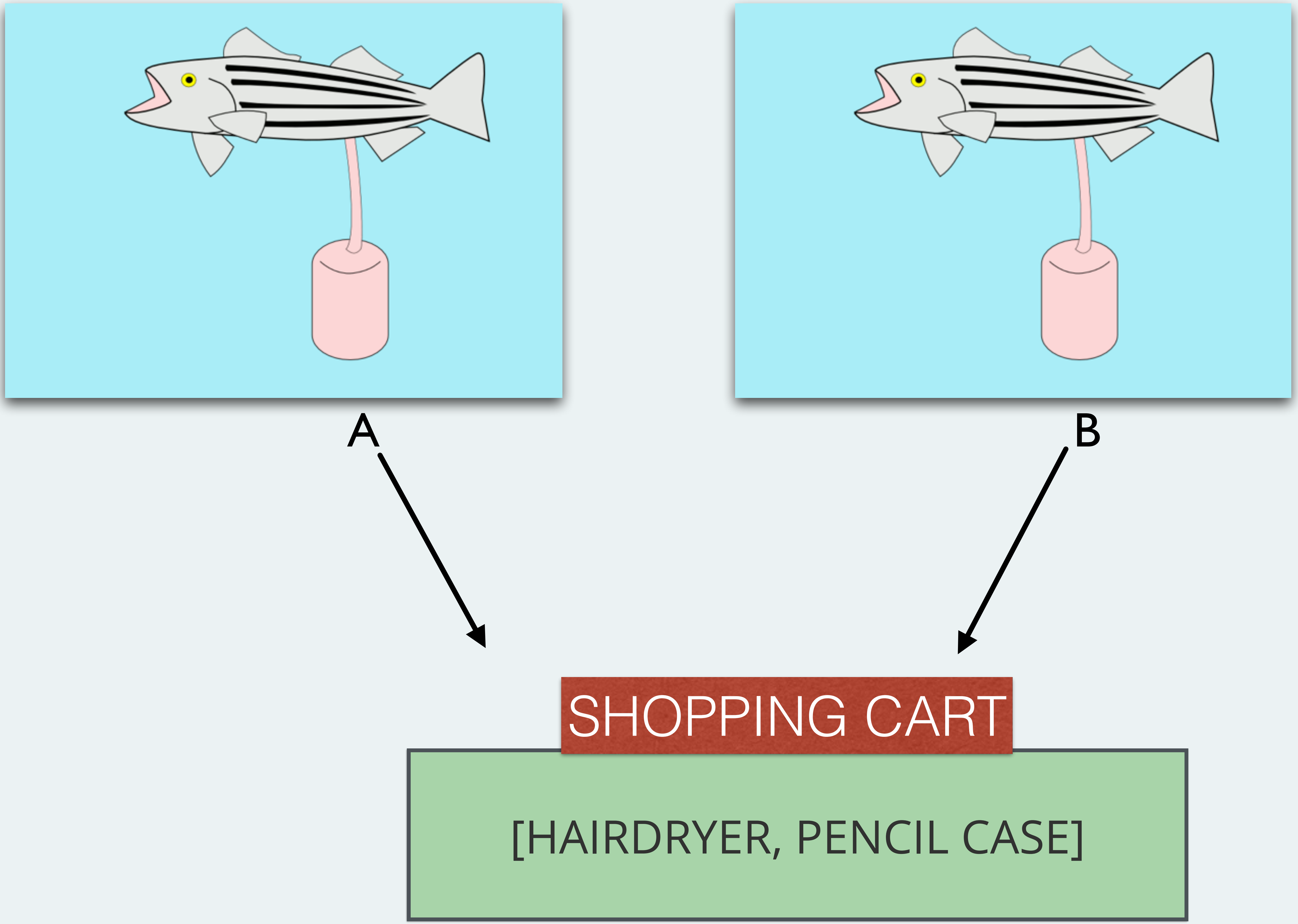


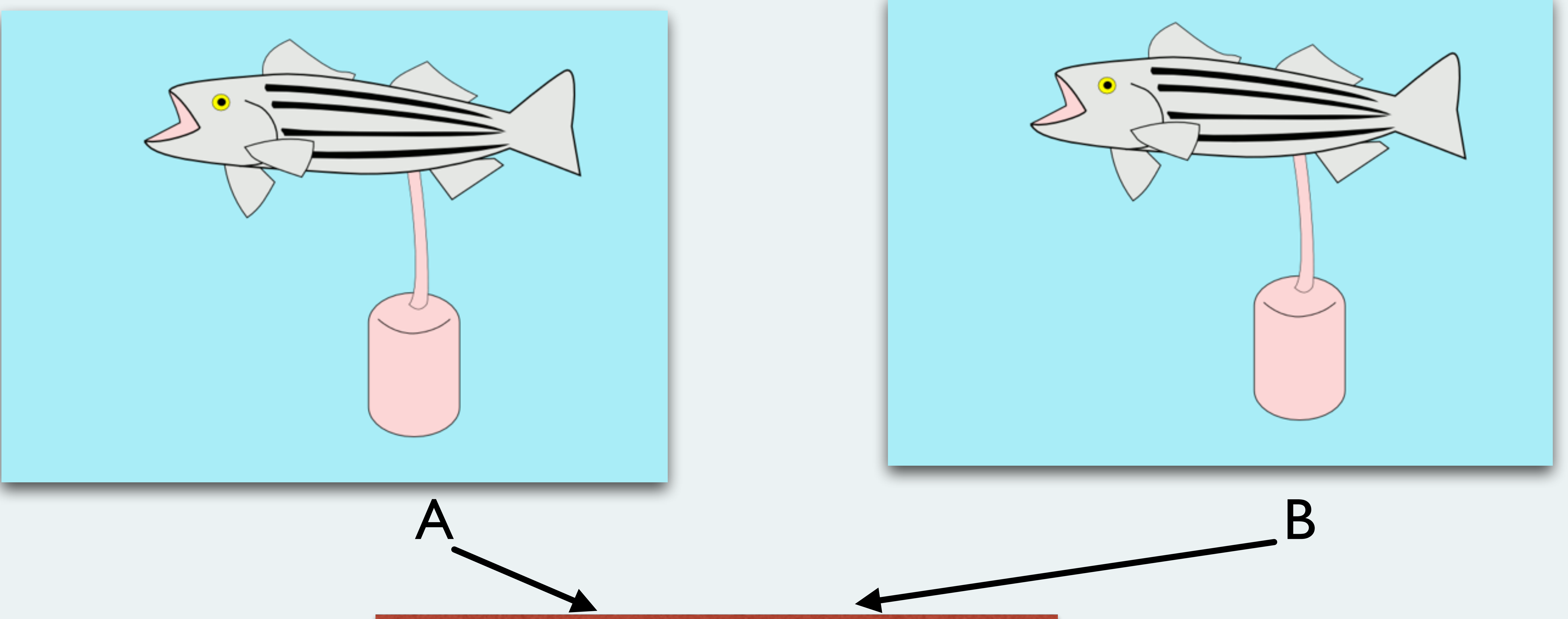


REPLICAS

ACTORS

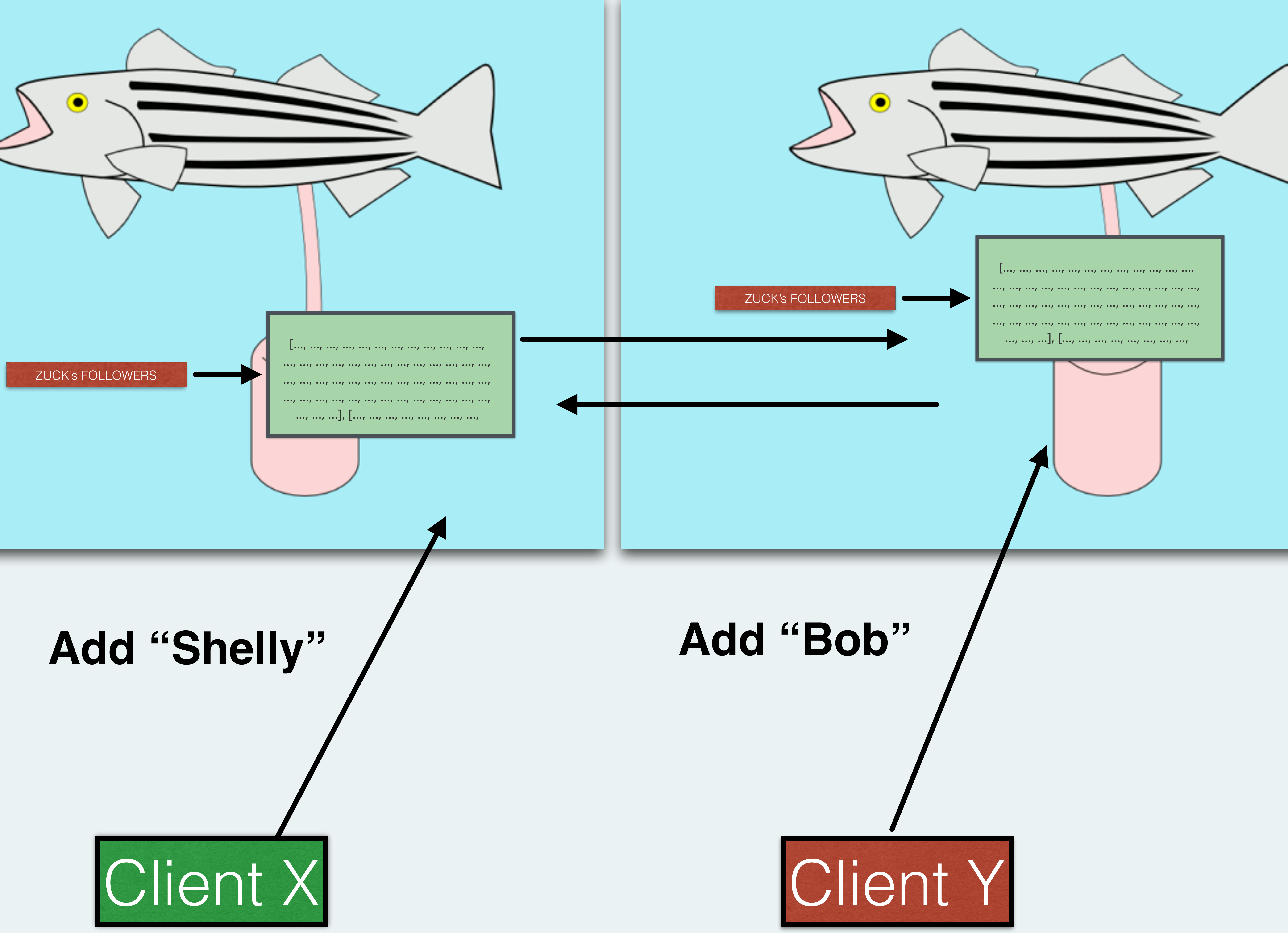
HOW TO USE THE LIB?

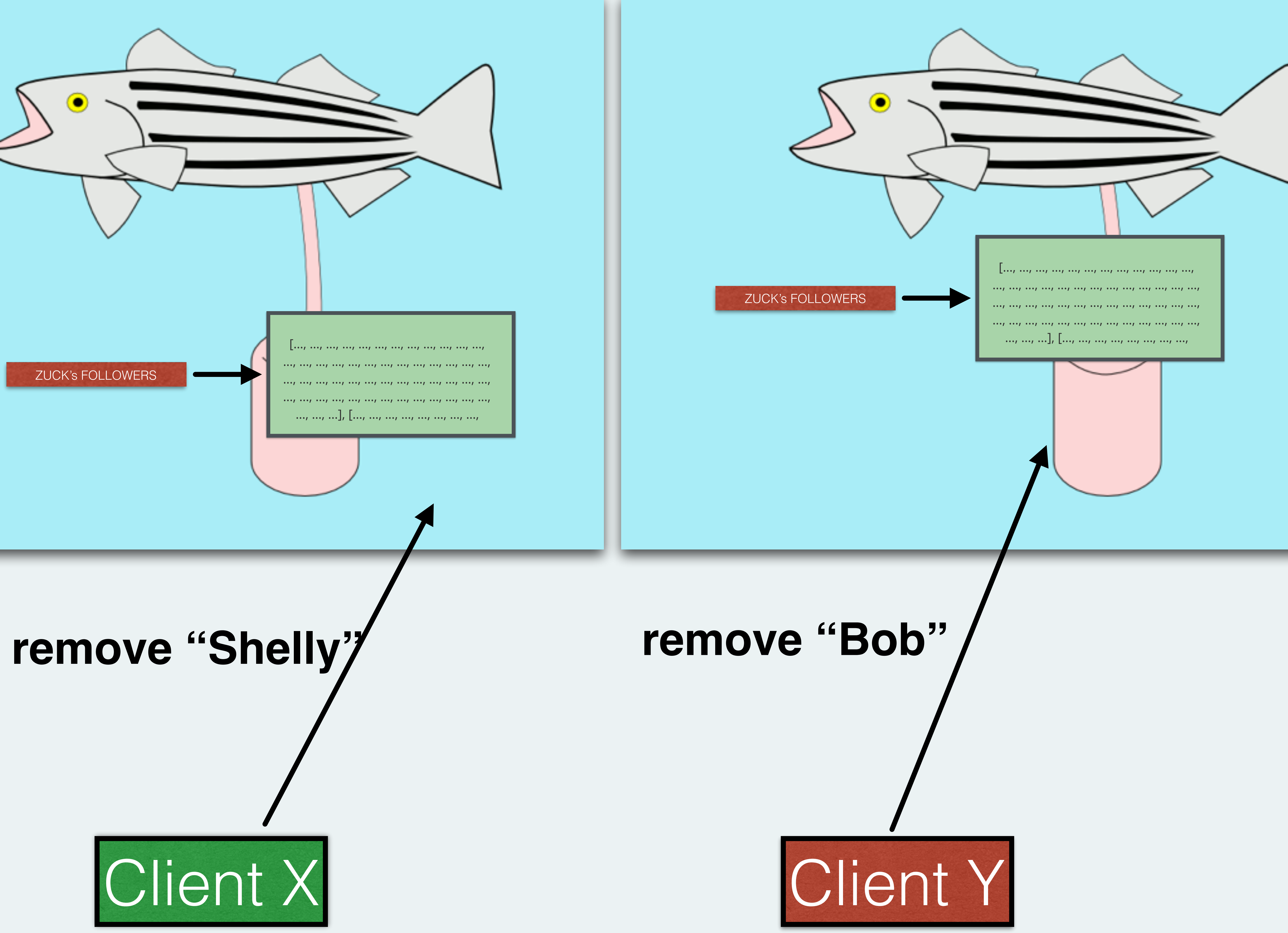

A
B
SHOPPING CART
[HAIRDRYER, PENCIL CASE]



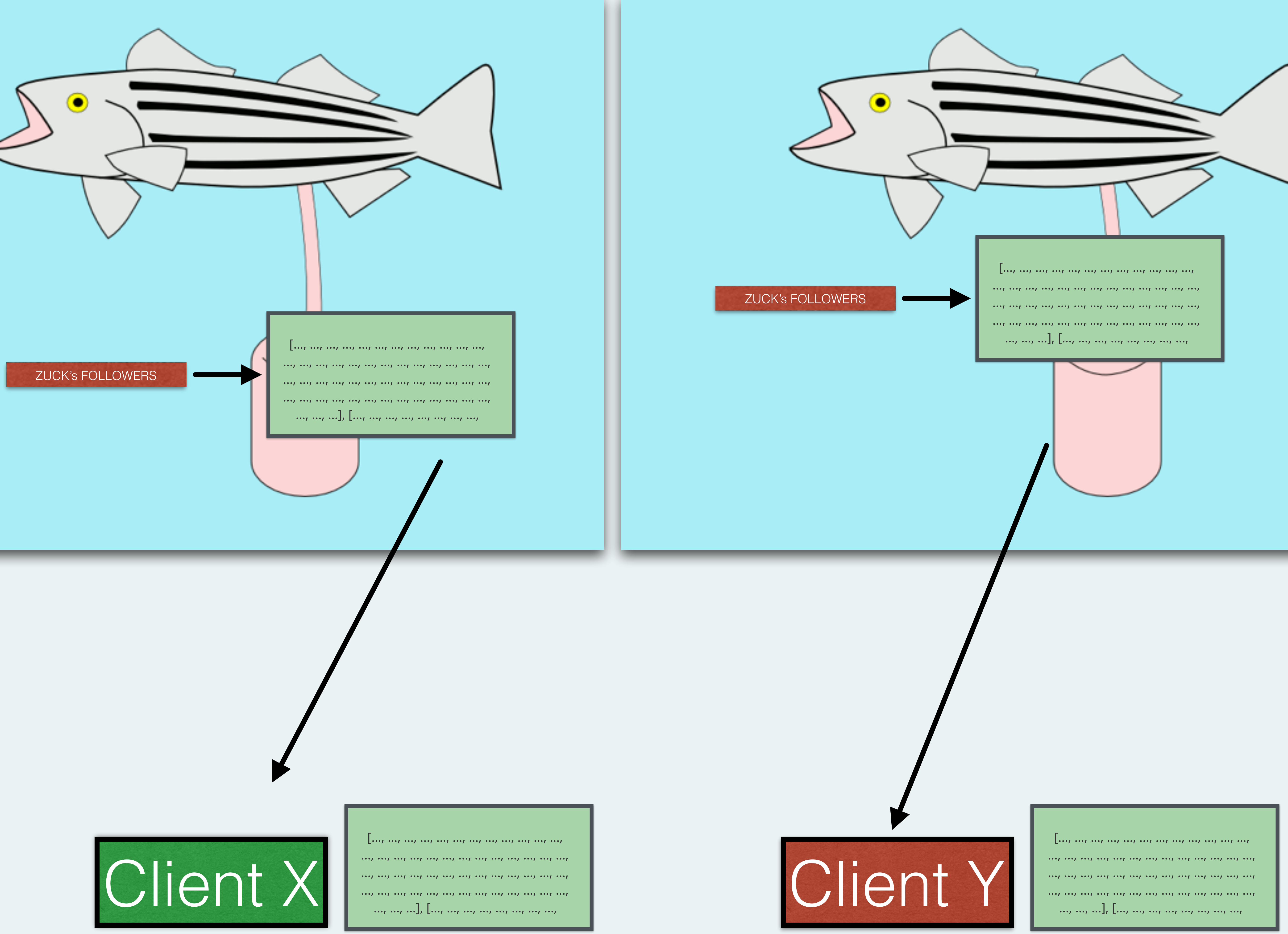
ZUCK's FOLLOWERS?

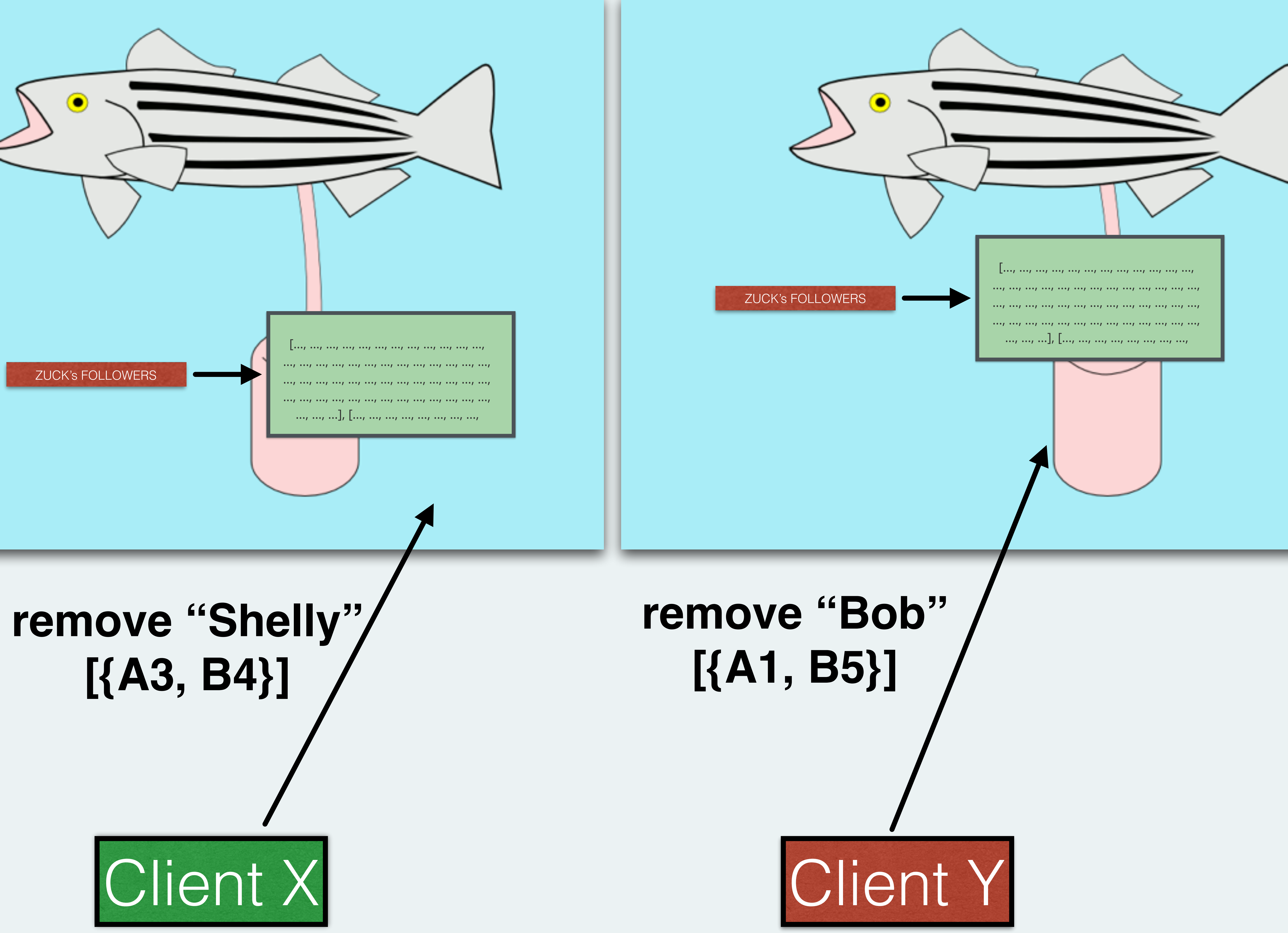
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ...], [...,


ZUCK's FOLLOWERS
[..., ..., ..., ..., ...], [..., ...]
ZUCK's FOLLOWERS
[..., ..., ..., ..., ...], [..., ...]
Add "Shelly"
Add "Bob"
Client X
Client Y


ZUCK's FOLLOWERS
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,
ZUCK's FOLLOWERS
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,
remove "Shelly"
remove "Bob"
Client X
Client Y

Observed Remove

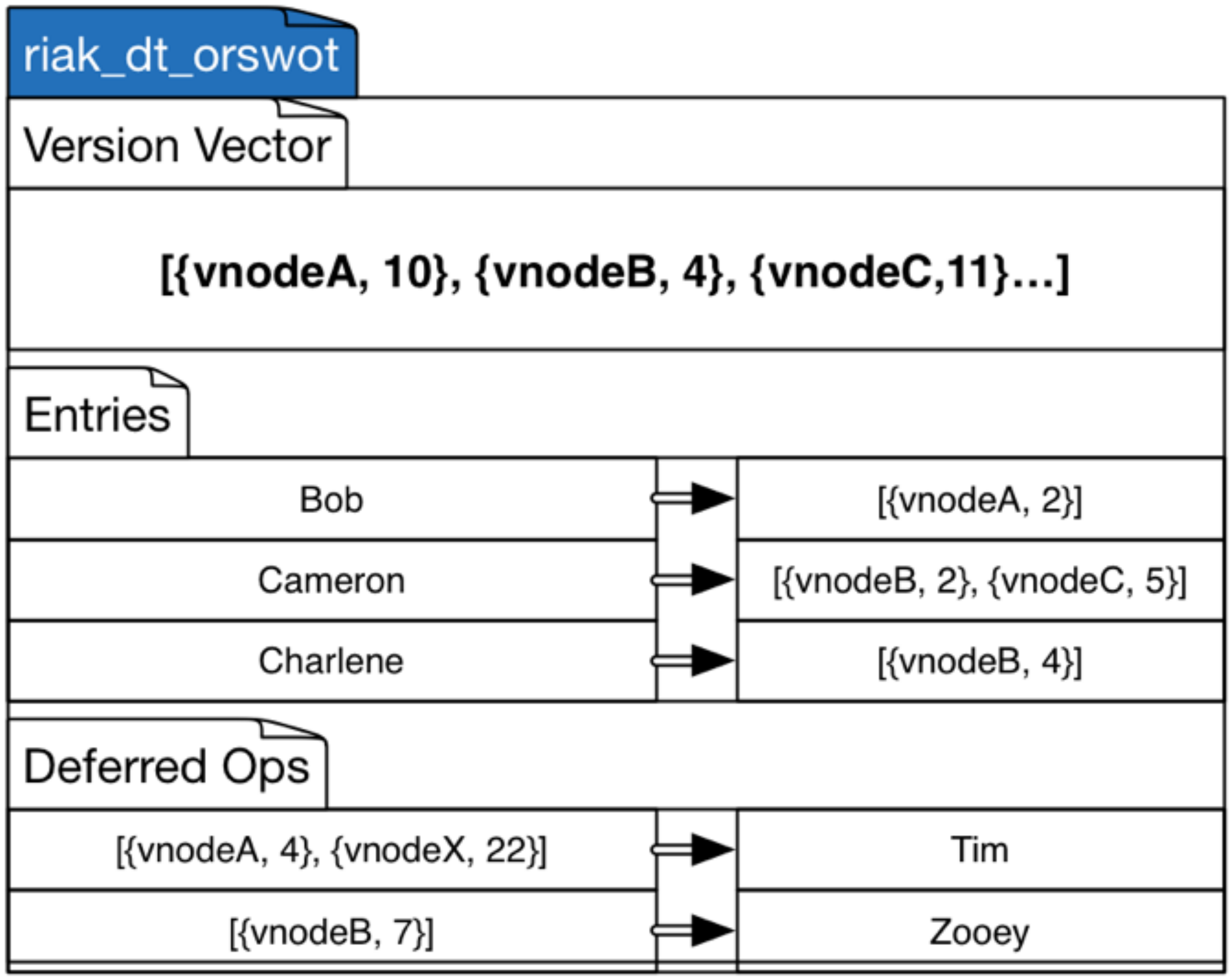

ZUCK's FOLLOWERS
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,
ZUCK's FOLLOWERS
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,
Client X
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,
Client Y
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,


ZUCK's FOLLOWERS
ZUCK's FOLLOWERS
remove "Shelly"
[{A3, B4}]
remove "Bob"
[{A1, B5}]
Client X
Client Y

Sets in Riak

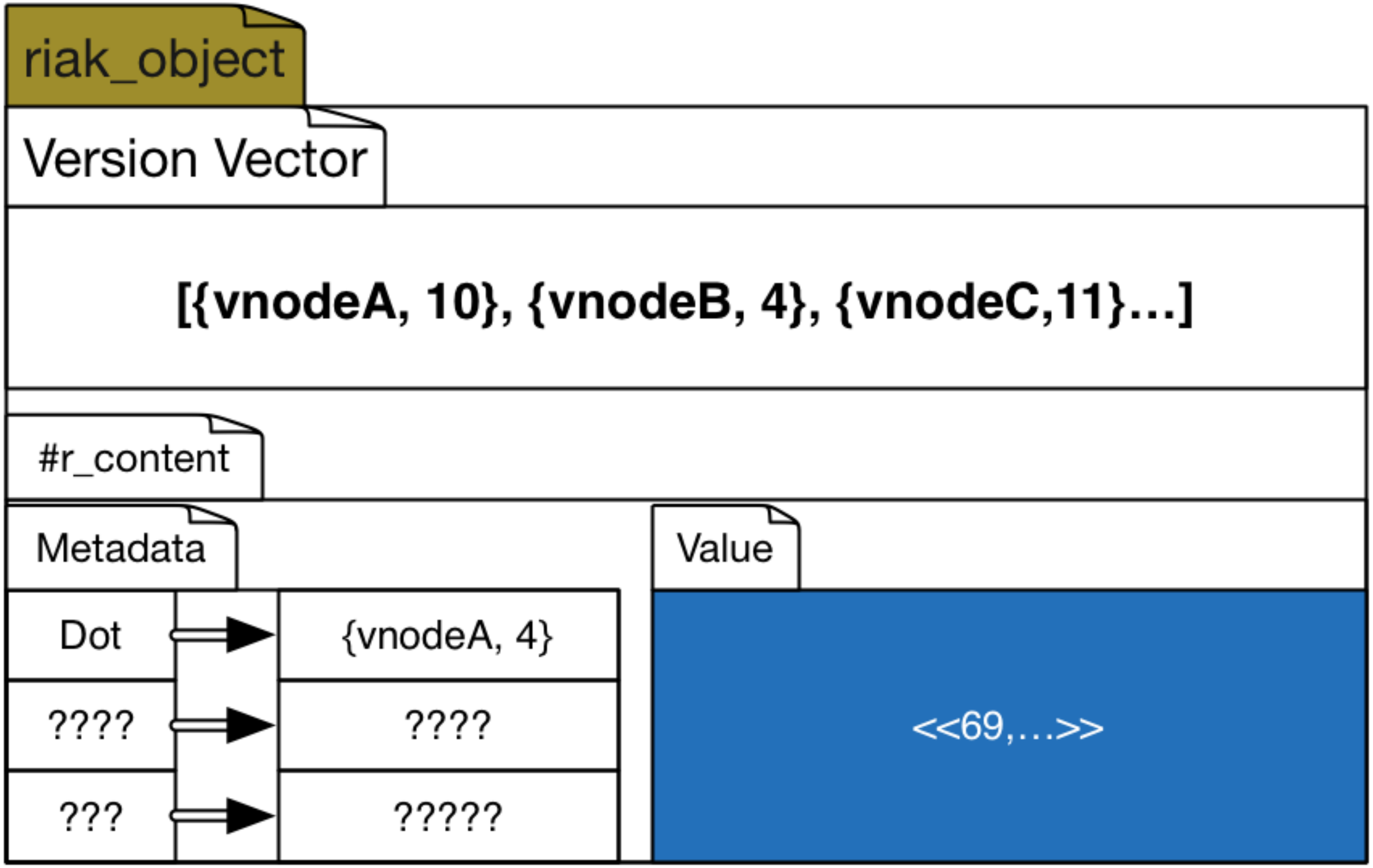
- Operation Based API
  - With causal Context for removes!
- Vnode As Actor/Replica
  - Action-at-a-distance
- Full state replication

Sets in Riak

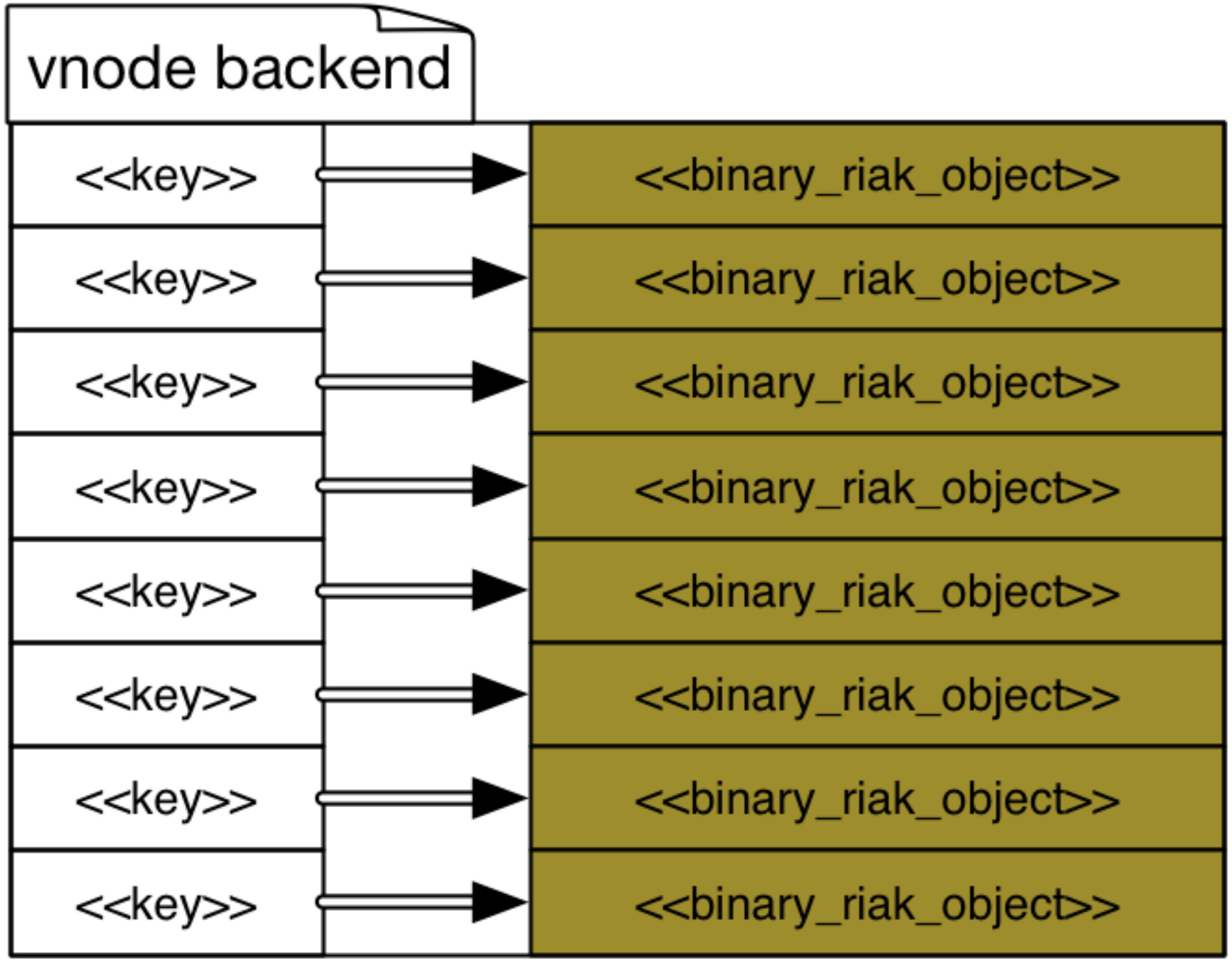


An optimized conflict-free replicated set
Annette Bieniusa et al
http://arxiv.org/abs/1210.3368

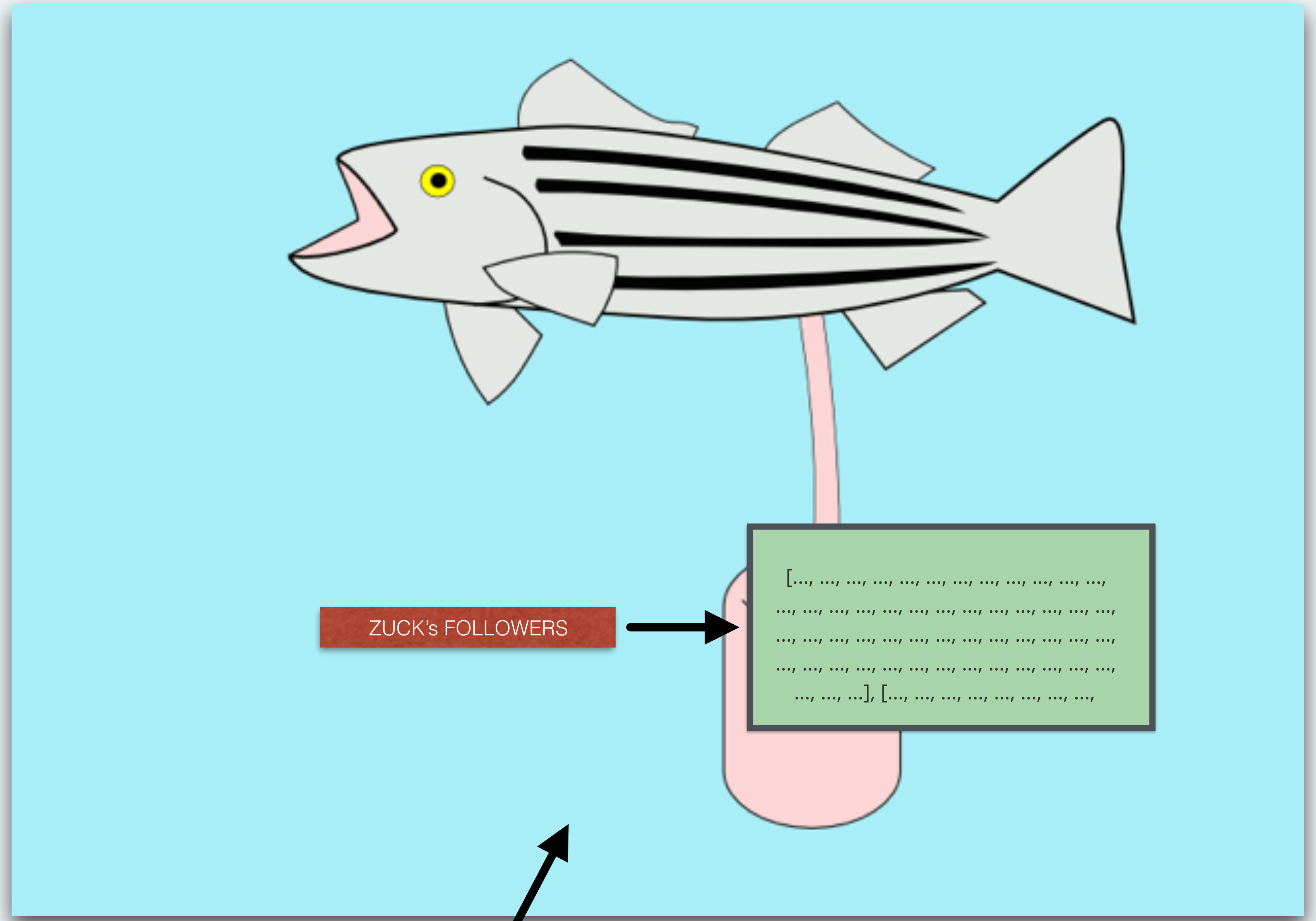
Sets in Riak


riak_object
Version Vector
[{vnodeA, 10}, {vnodeB, 4}, {vnodeC,11}…]
#r_content
Metadata
Dot
{vnodeA, 4}
????
????
???
?????
Value
<<69,…>>

Sets in Riak

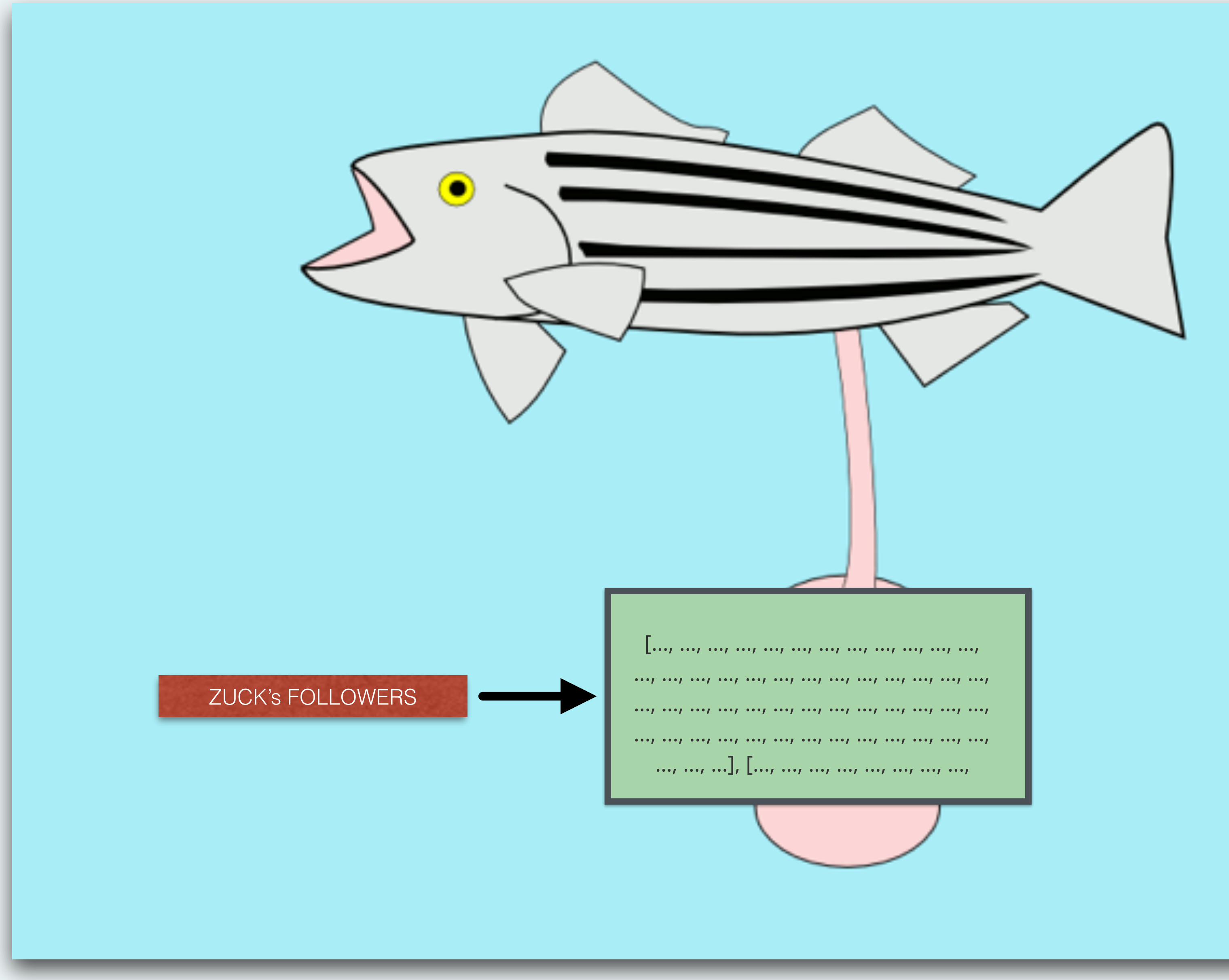


Sets in Riak



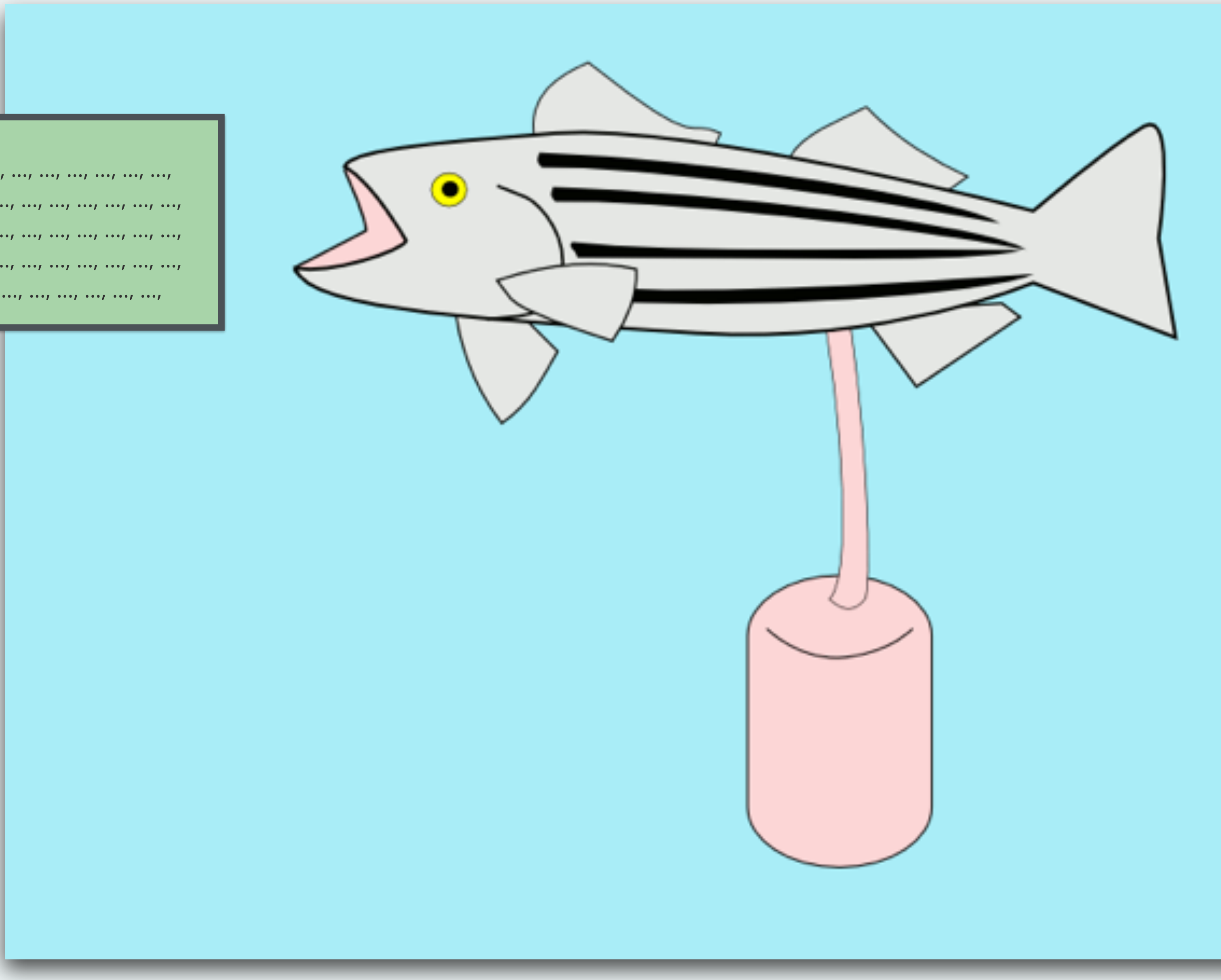
Add "Shelly"

Client X


ZUCK's FOLLOWERS
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,

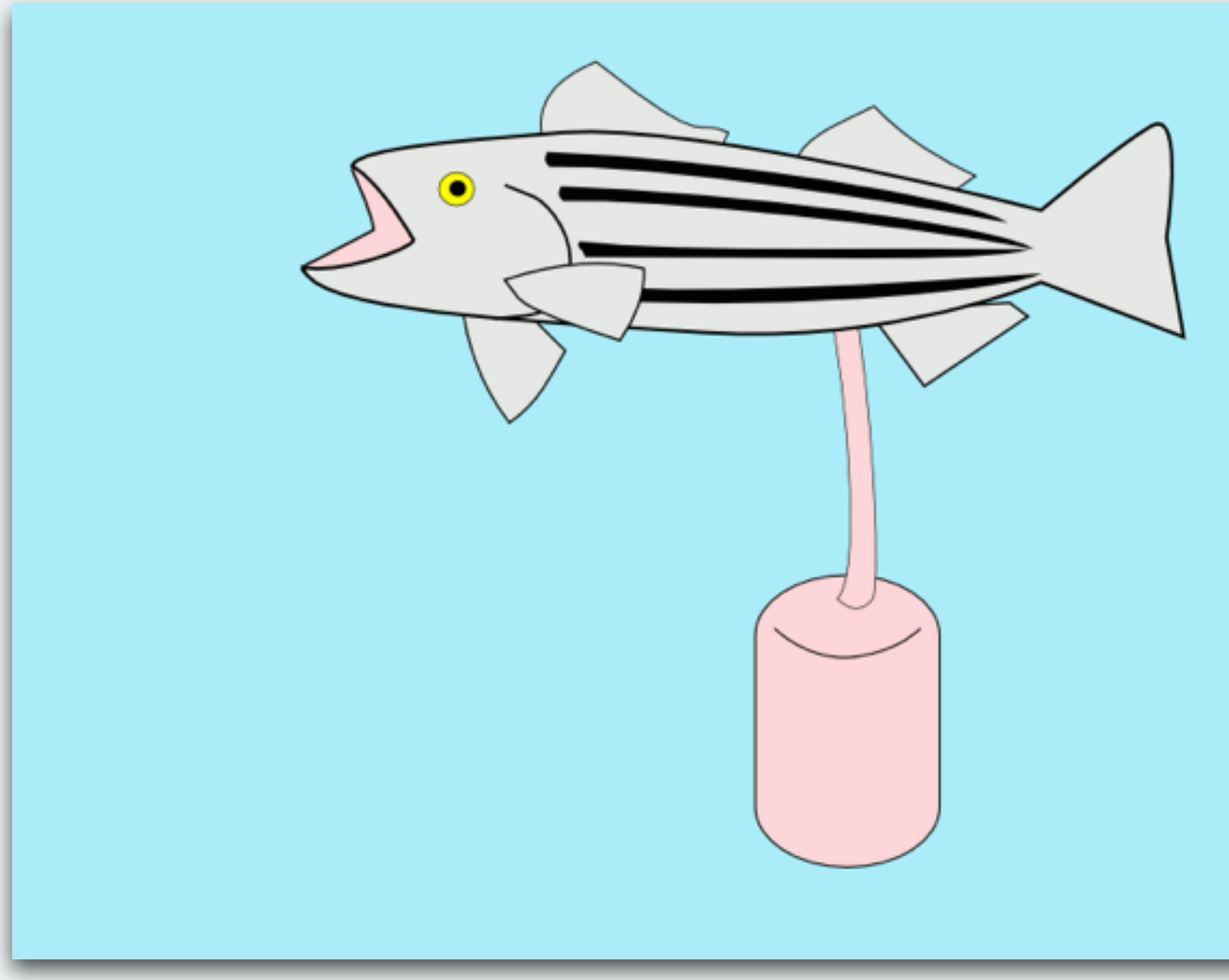
ZUCK's FOLLOWERS →

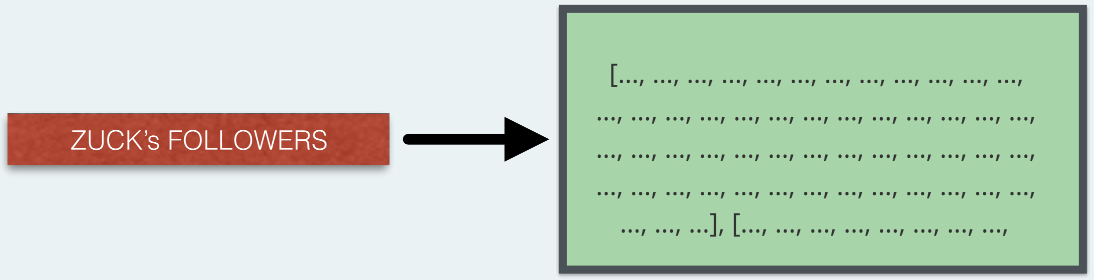
```
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,
```

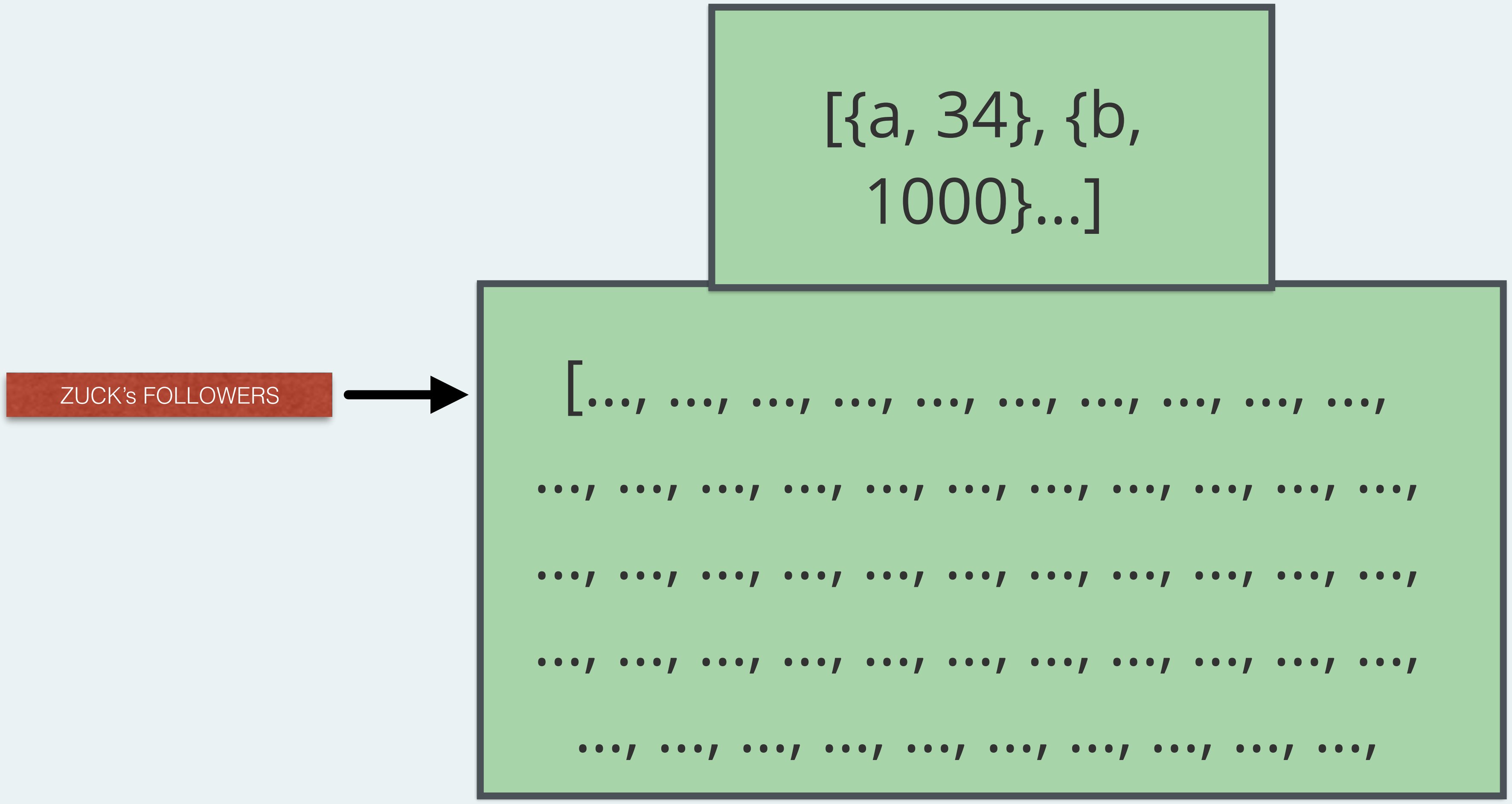


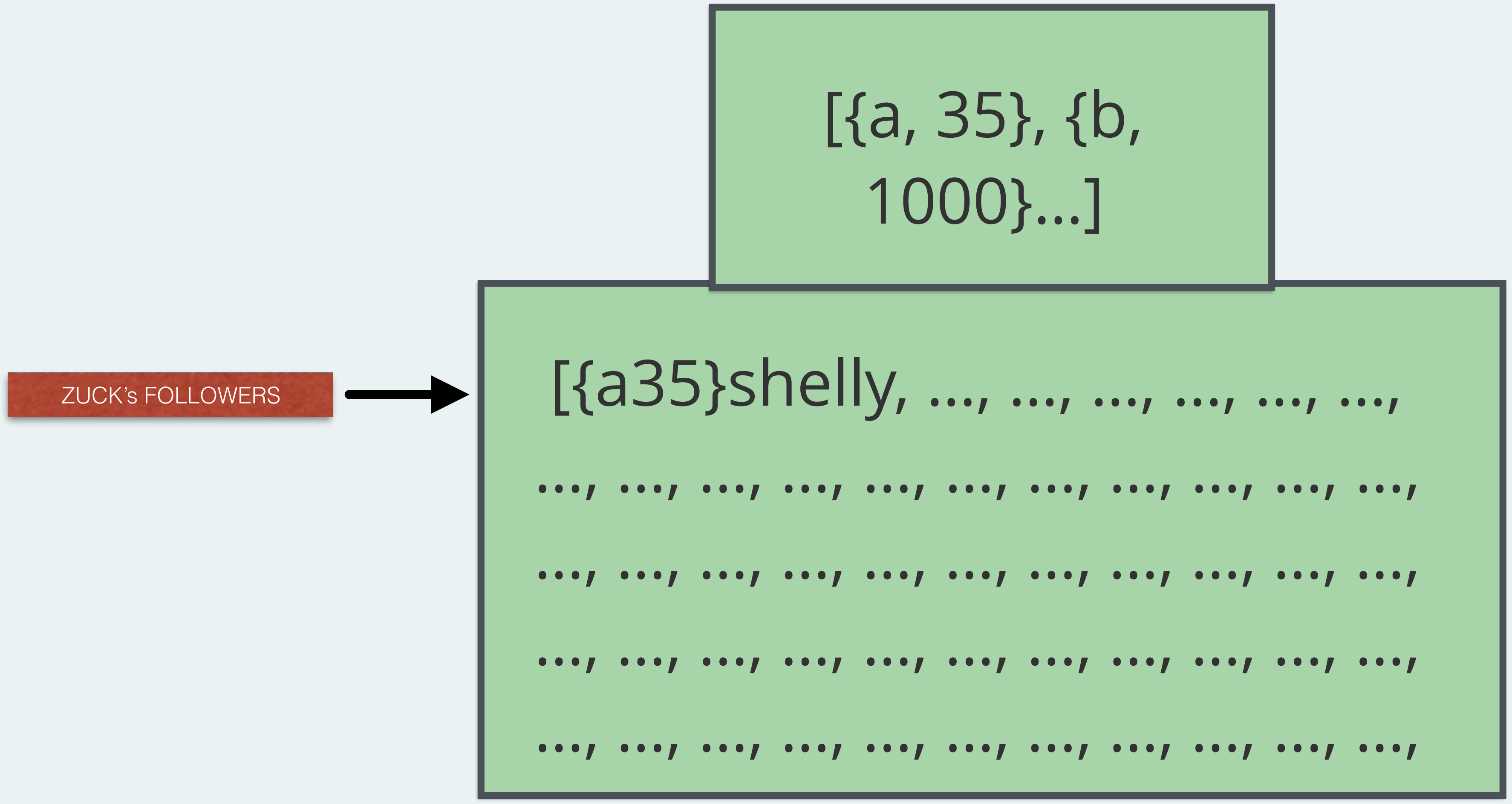
ZUCK's FOLLOWERS →

```
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,
```



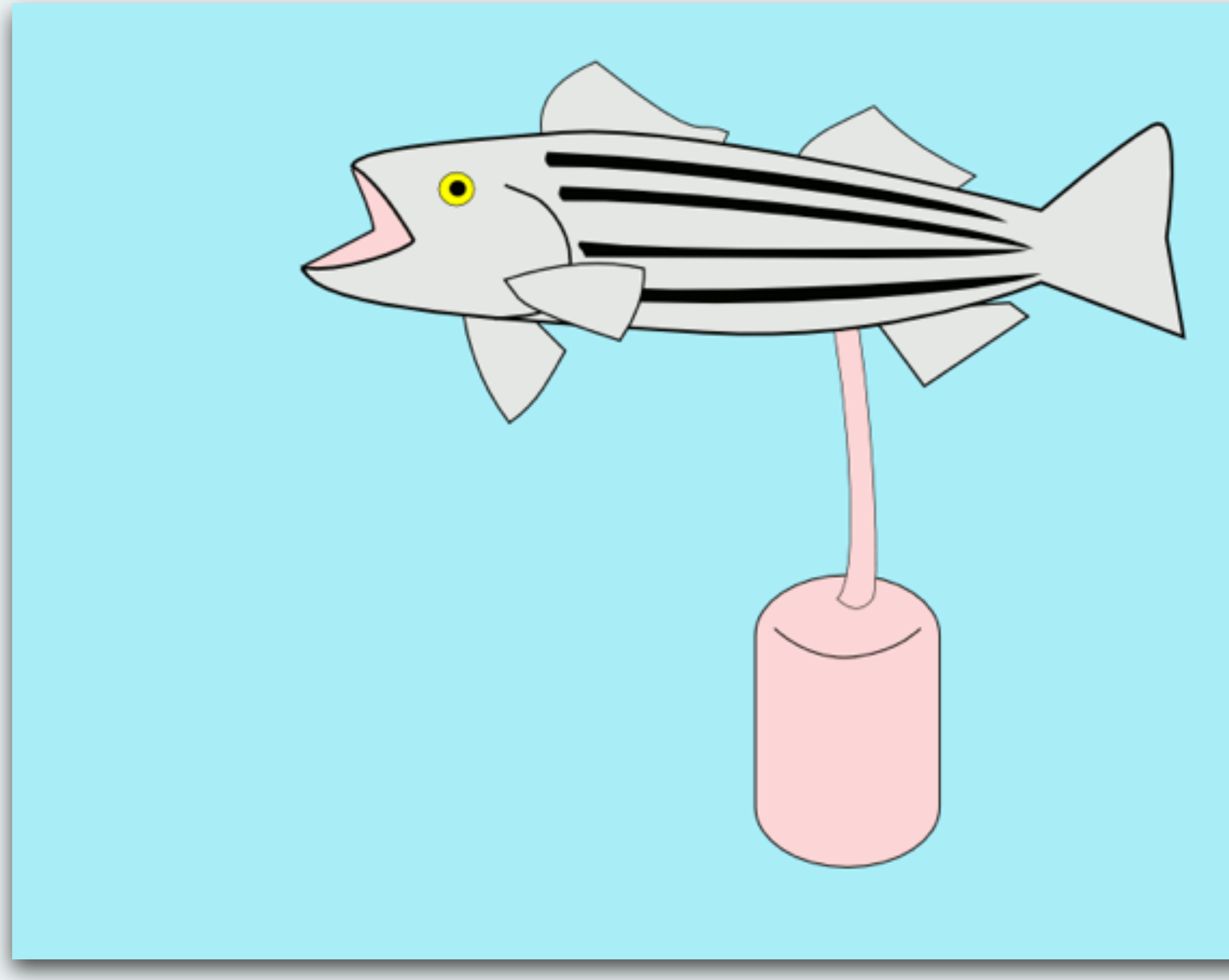

ZUCK's FOLLOWERS
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,


[{a, 34}, {b, 1000}...]
ZUCK's FOLLOWERS
[..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ...,


[{a, 35}, {b, 1000}…]
ZUCK's FOLLOWERS
[{a35}shelly, …, …, …, …, …, …,
…, …, …, …, …, …, …, …, …, …, …,
…, …, …, …, …, …, …, …, …, …, …,
…, …, …, …, …, …, …, …, …, …, …,
…, …, …, …, …, …, …, …, …, …, …,

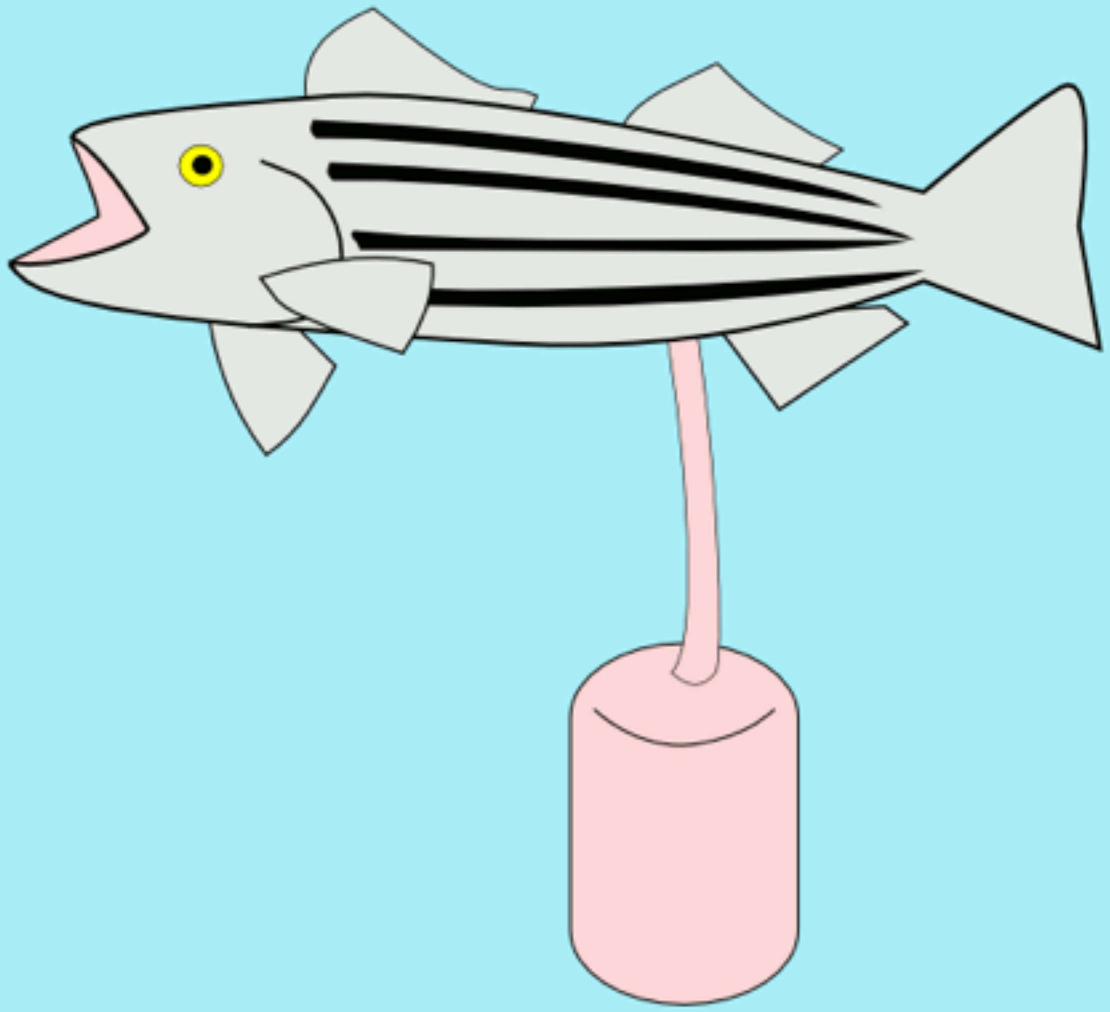
ZUCK's FOLLOWERS →

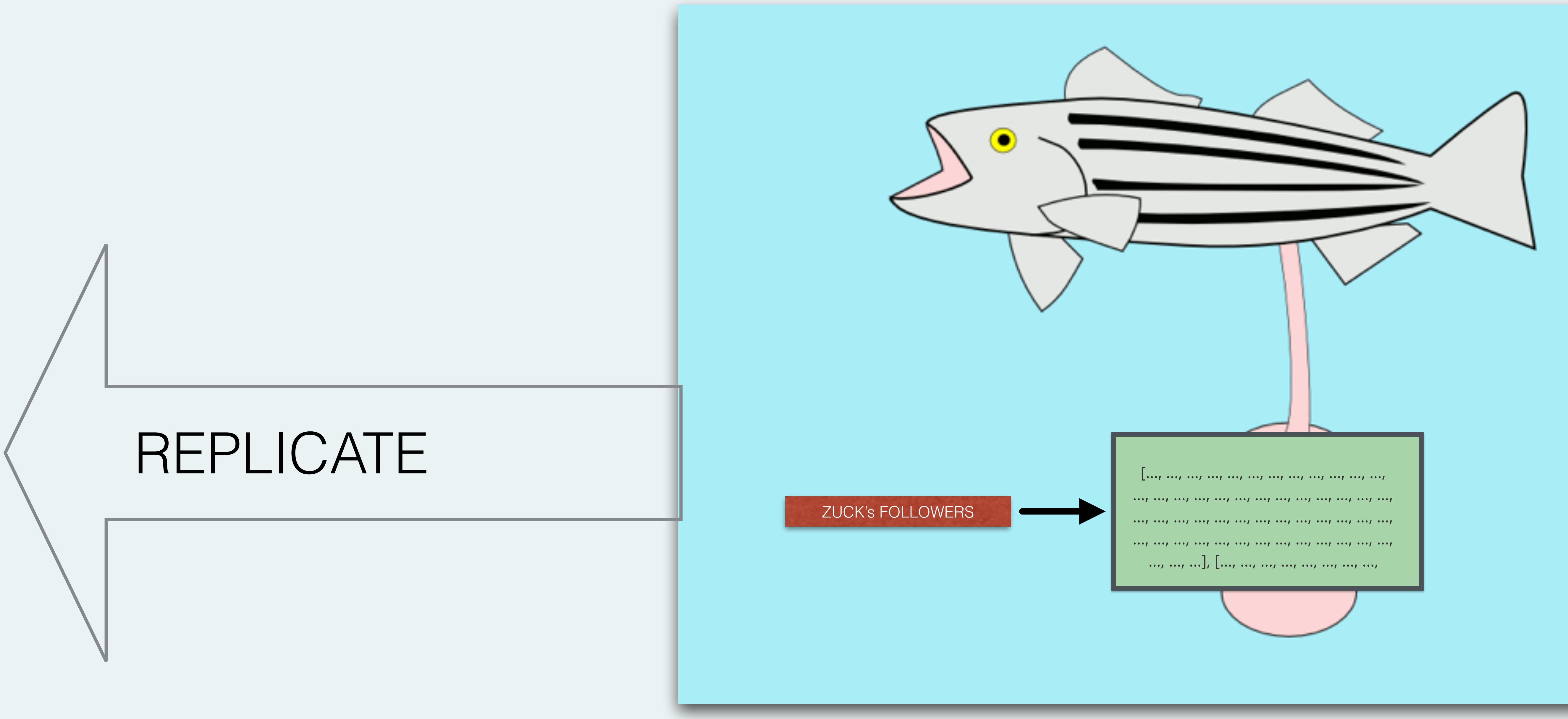
```
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,
```

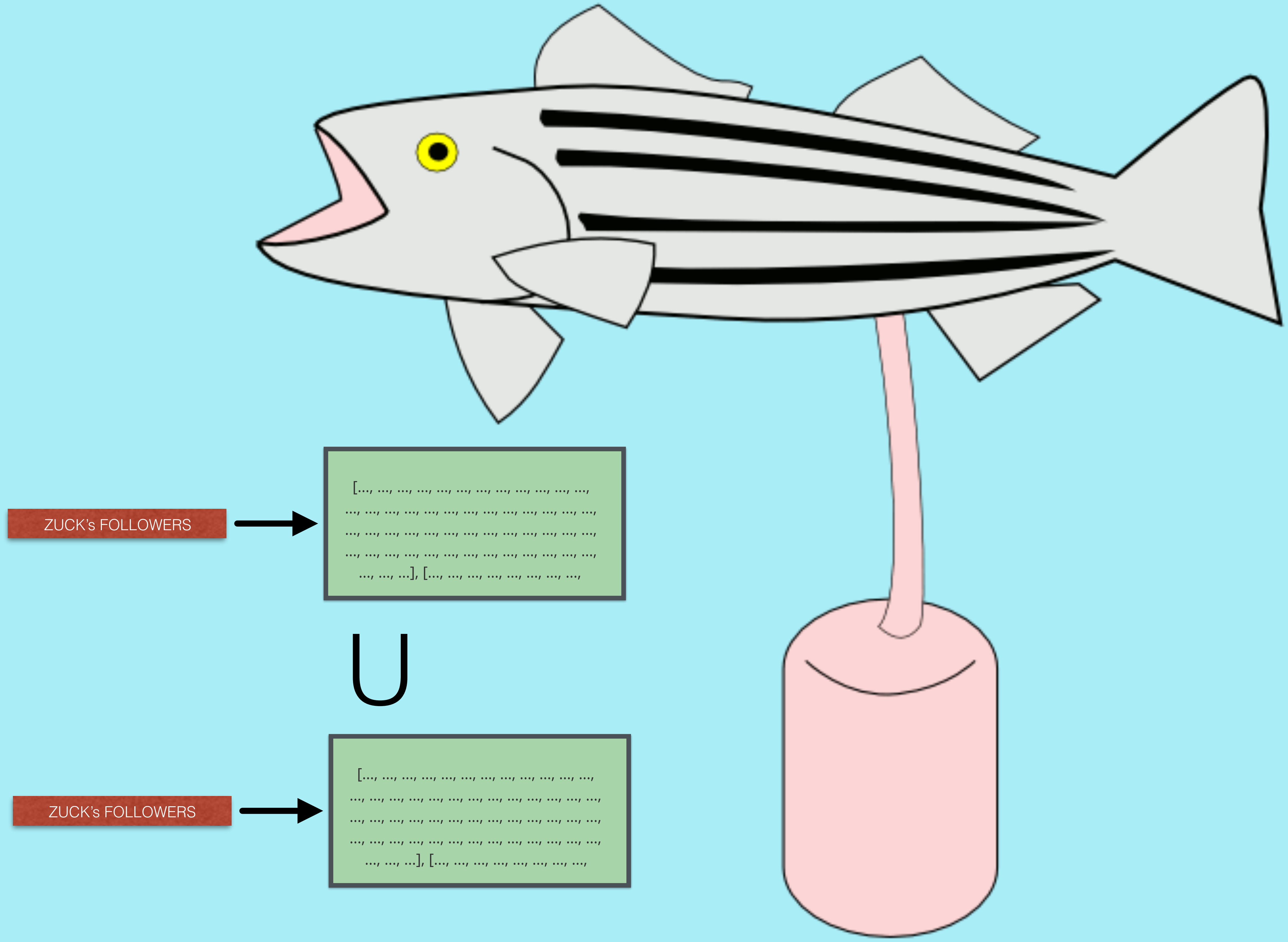


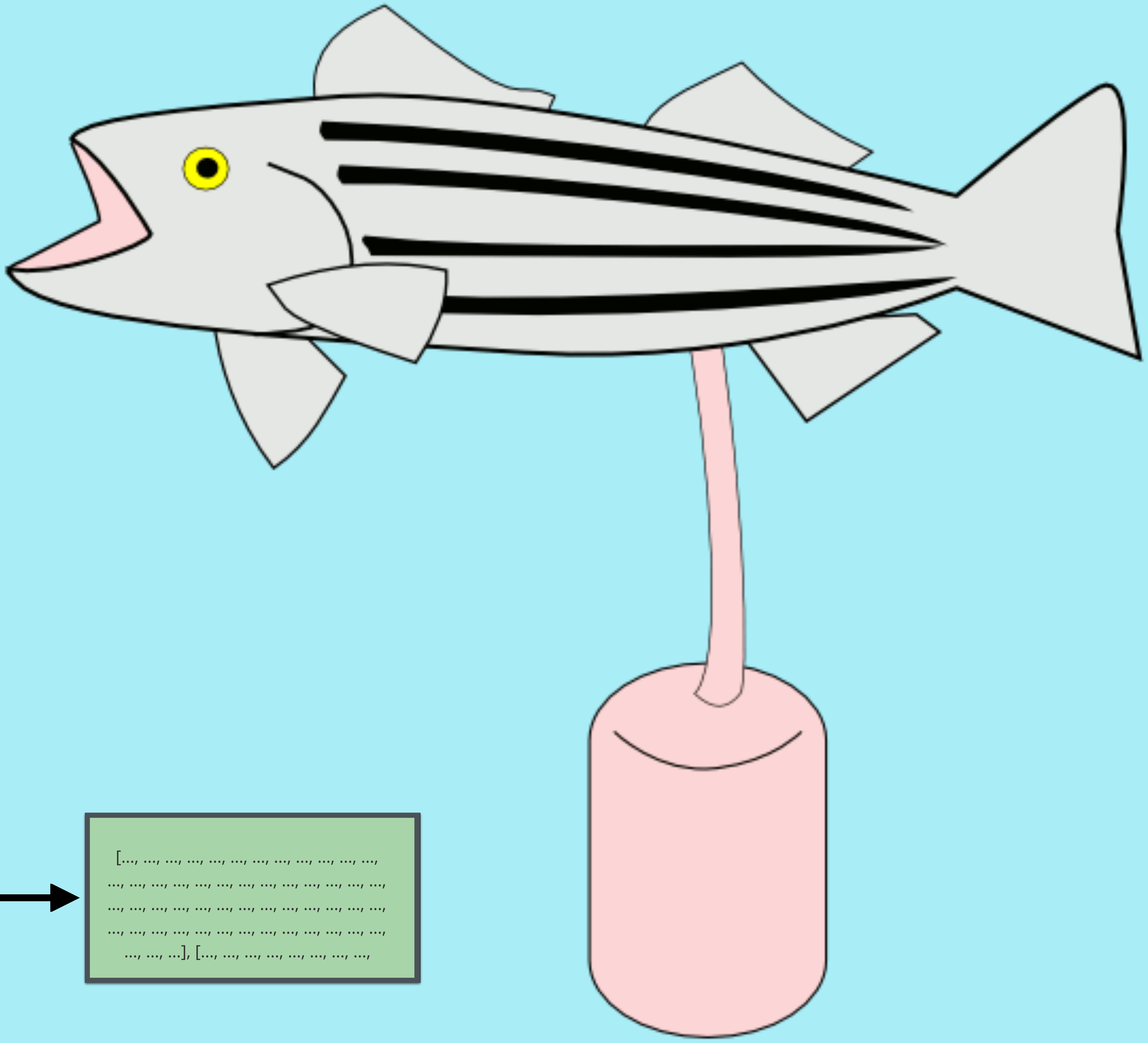
ZUCK's FOLLOWERS

```
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,
```



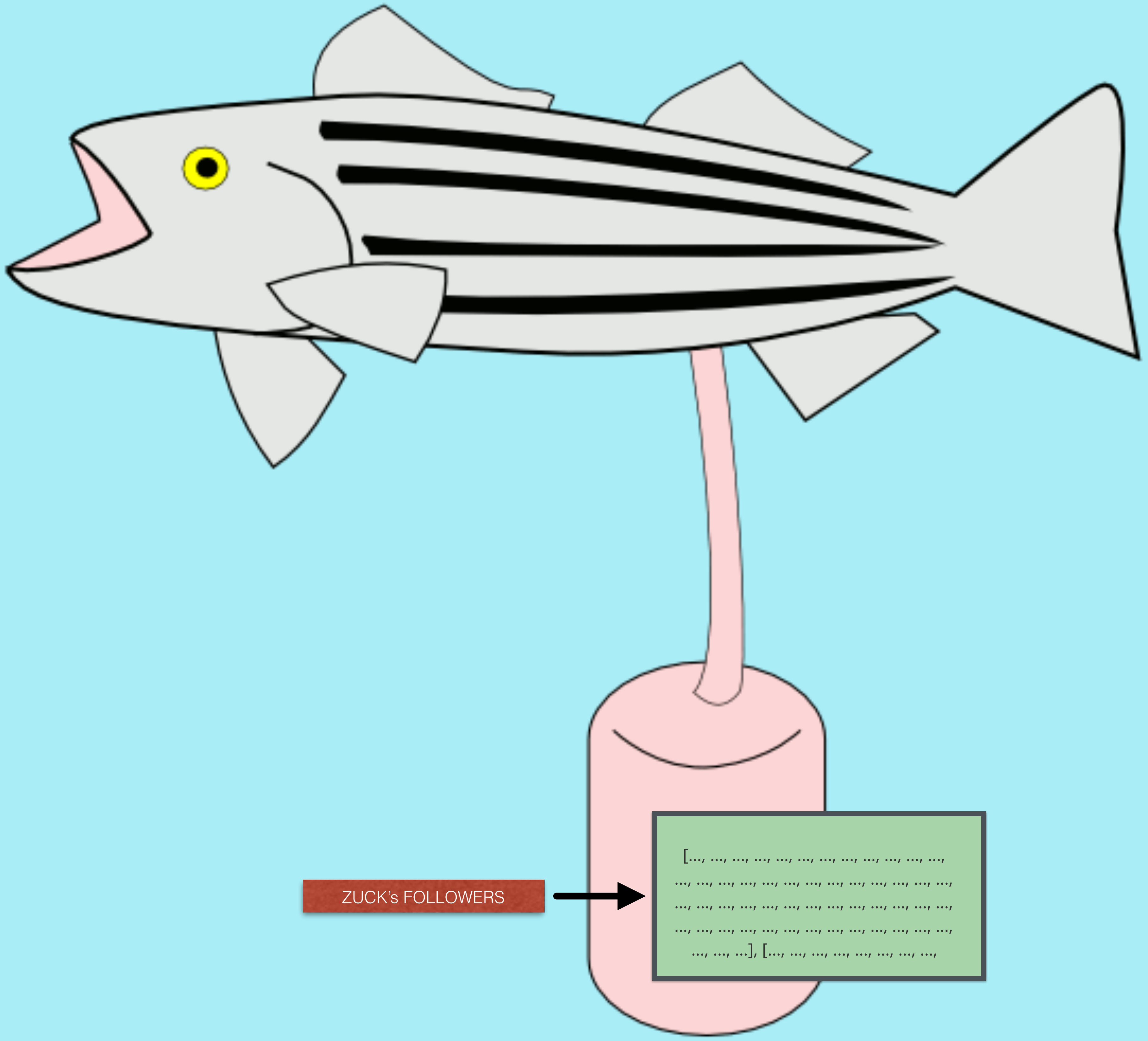

REPLICATE
ZUCK's FOLLOWERS
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,


ZUCK's FOLLOWERS
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,
∪
ZUCK's FOLLOWERS
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,



ZUCK's FOLLOWERS

```
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,
```


ZUCK's FOLLOWERS

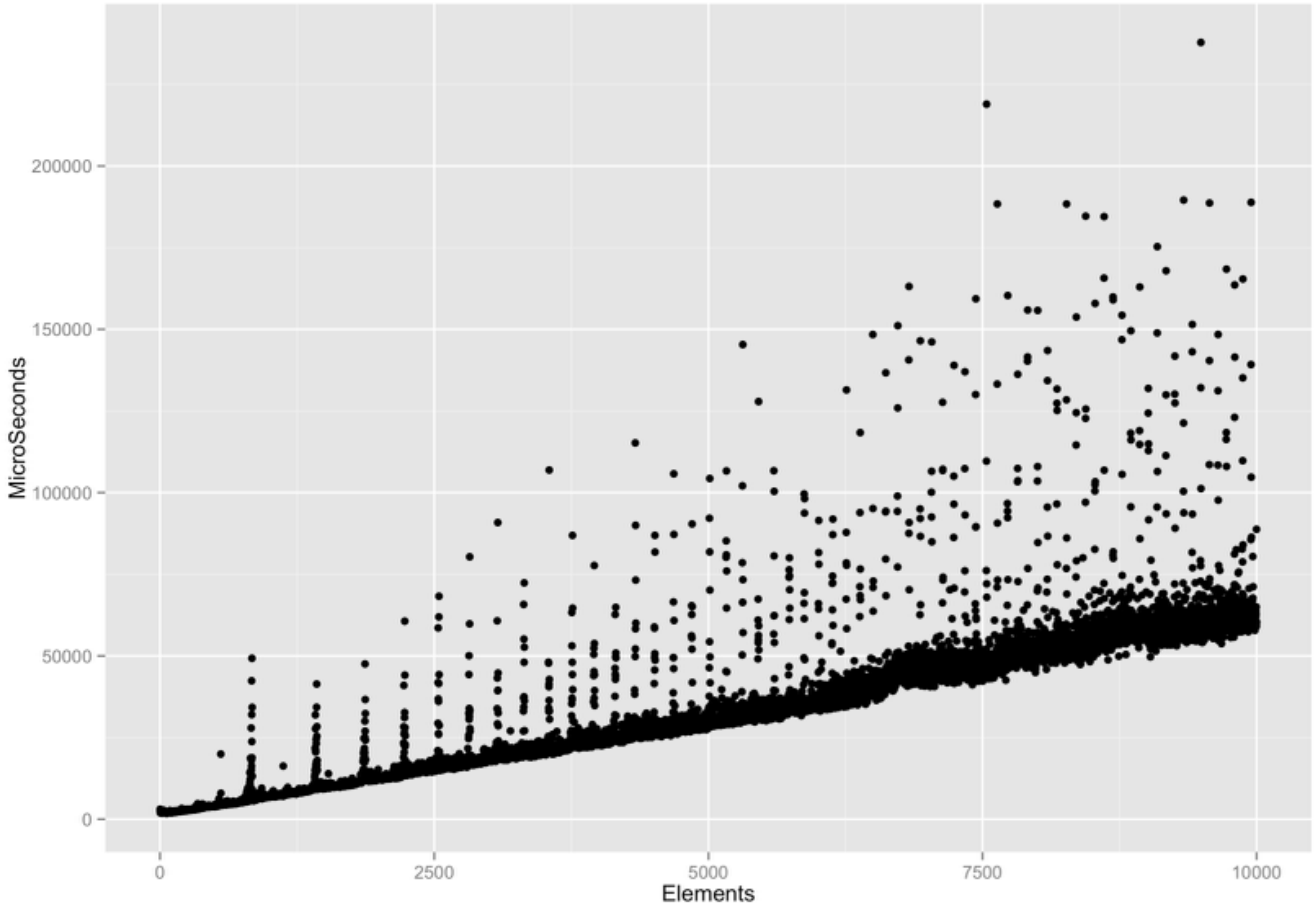
```
[..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ..., ...,
..., ..., ...], [..., ..., ..., ..., ..., ..., ..., ...,
```

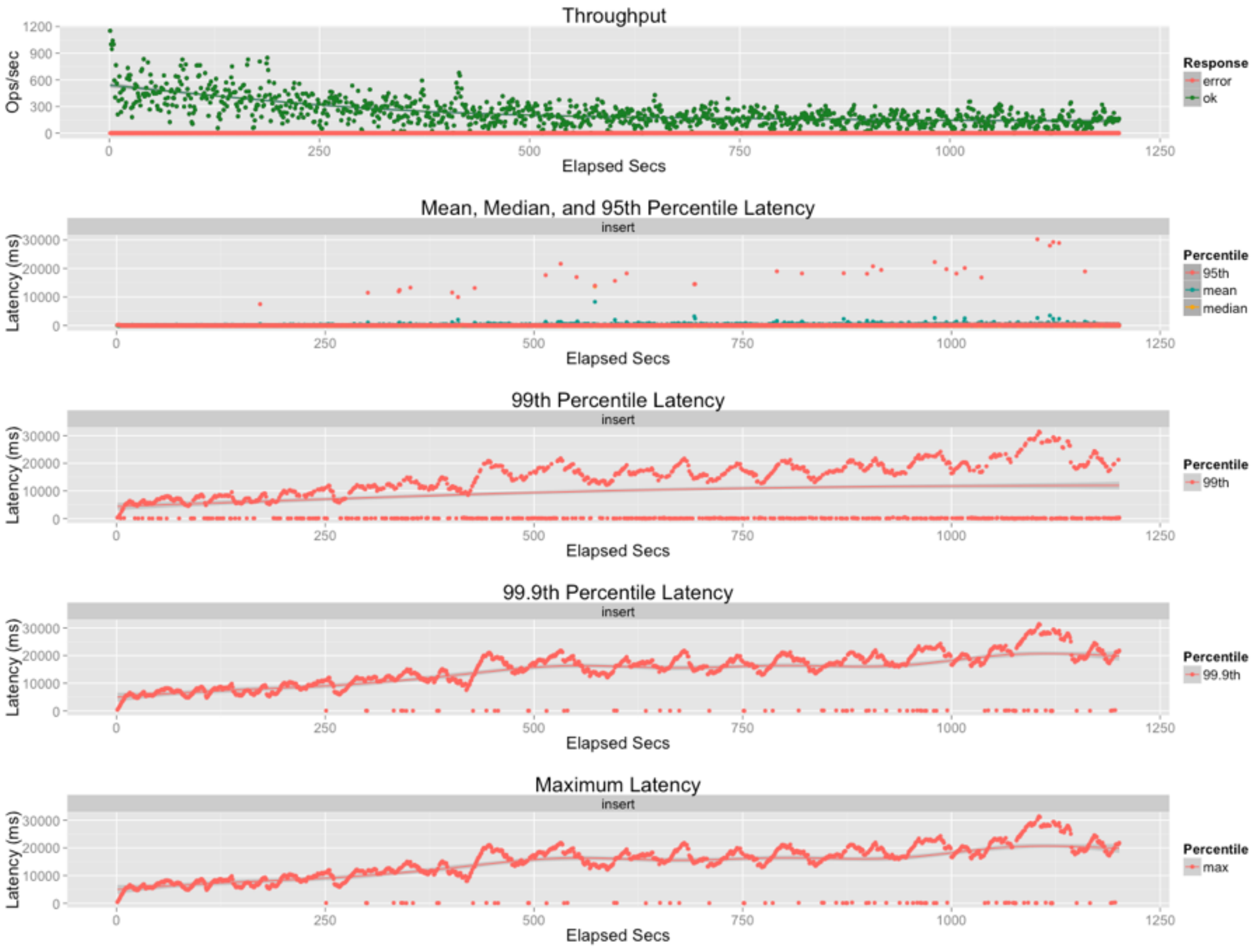
Problem?

- 1key -> 1 Set
  - Poor Write speed
  - Can’t have “big” sets

Every time we change the set we read and write the whole set!

Sets in Riak


MicroSeconds
200000
150000
100000
50000
0
0
2500
5000
7500
10000
Elements



10k sets, 100k elements, 50 workers - write

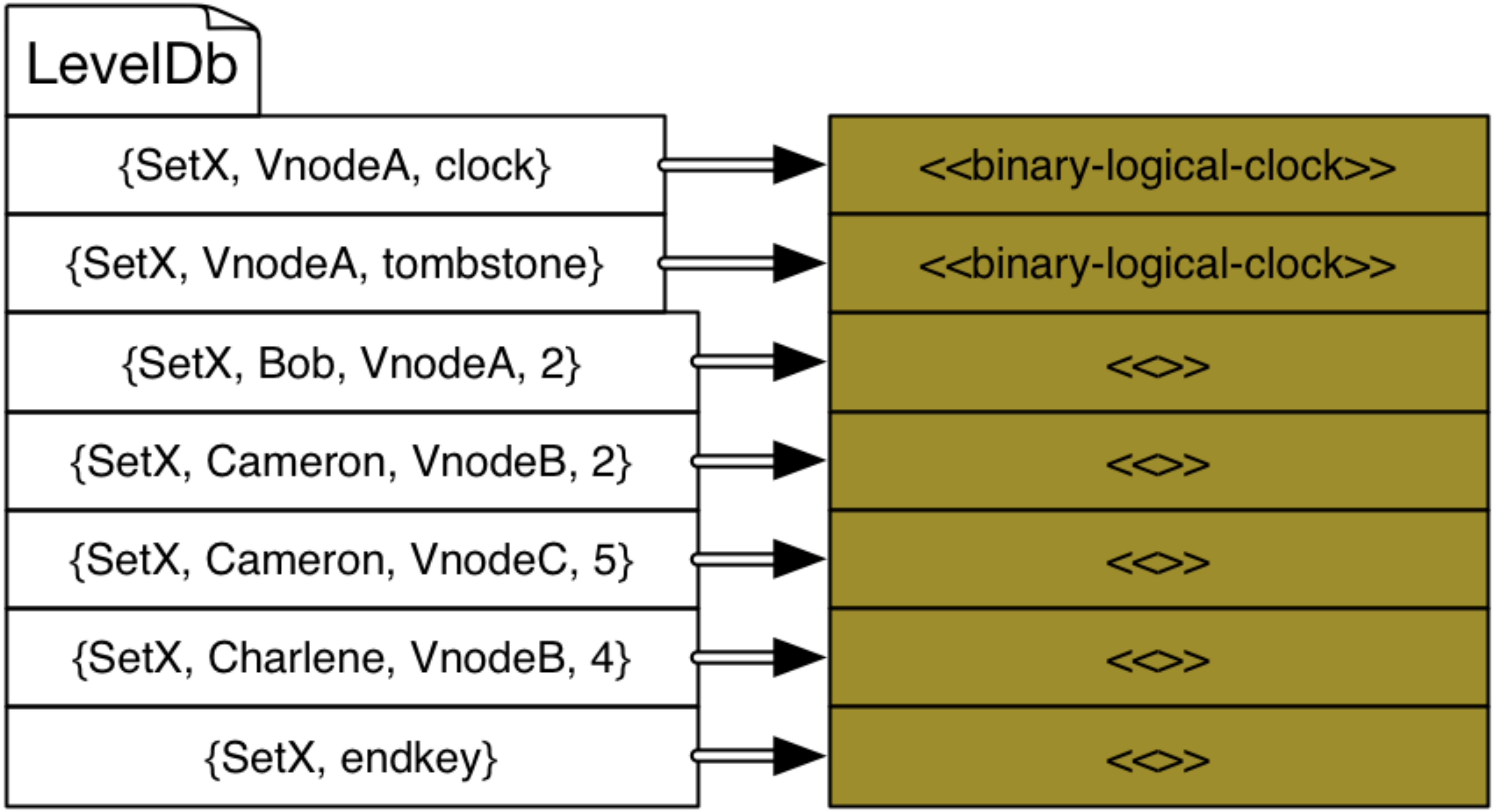
Sets in Riak

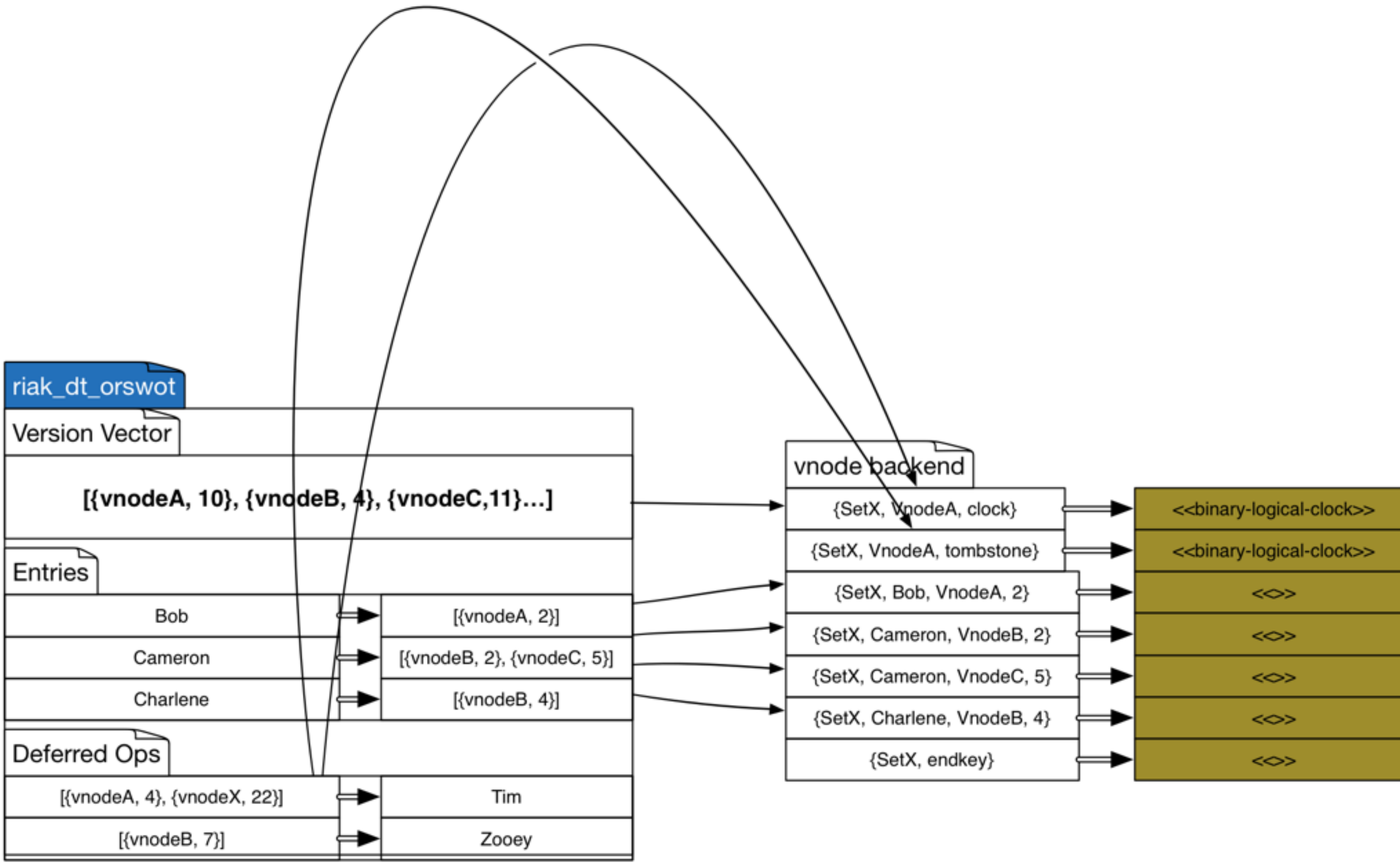
Small : riak object

1MB limit

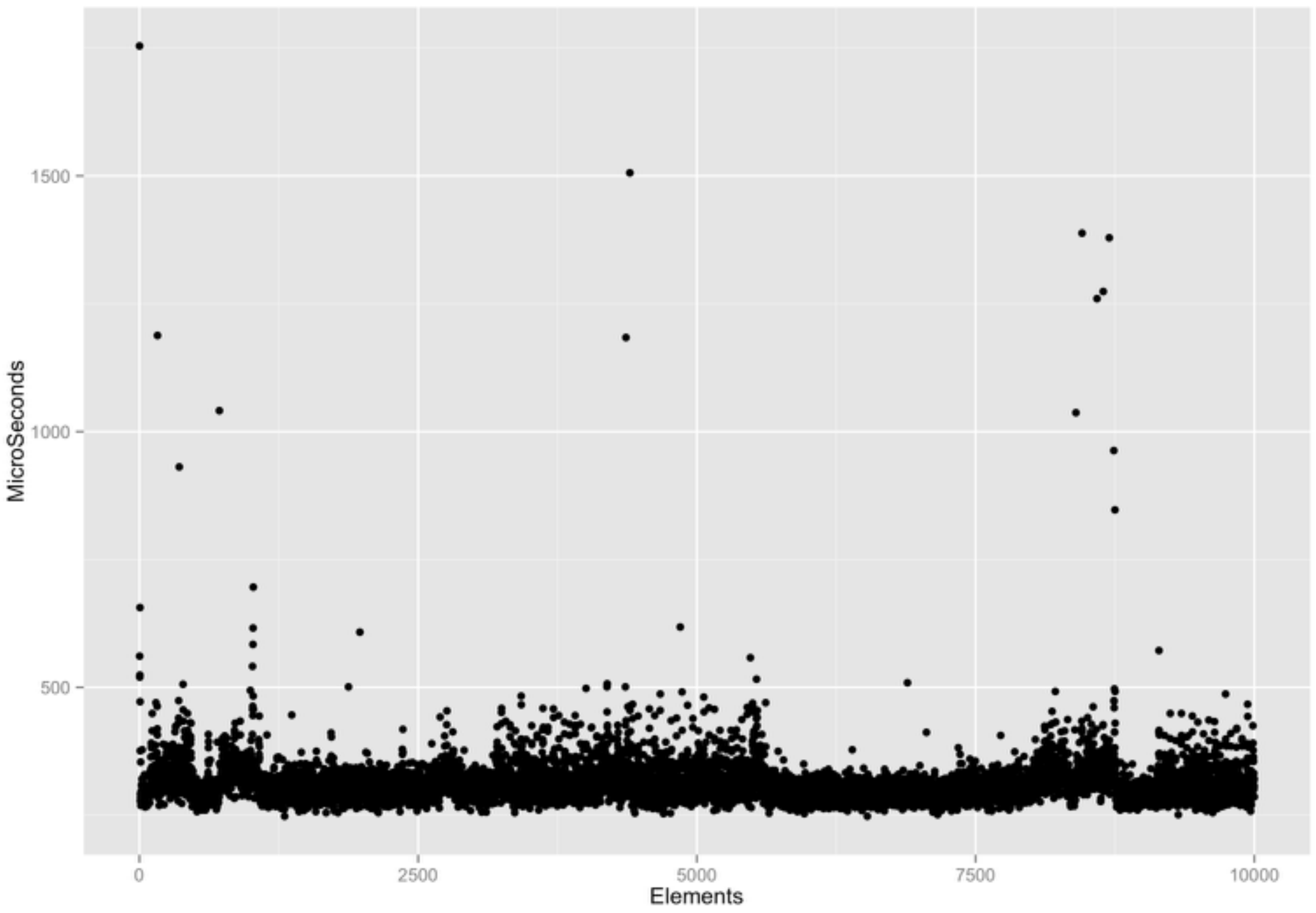
Bigsets:
Make writes faster
and
sets bigger

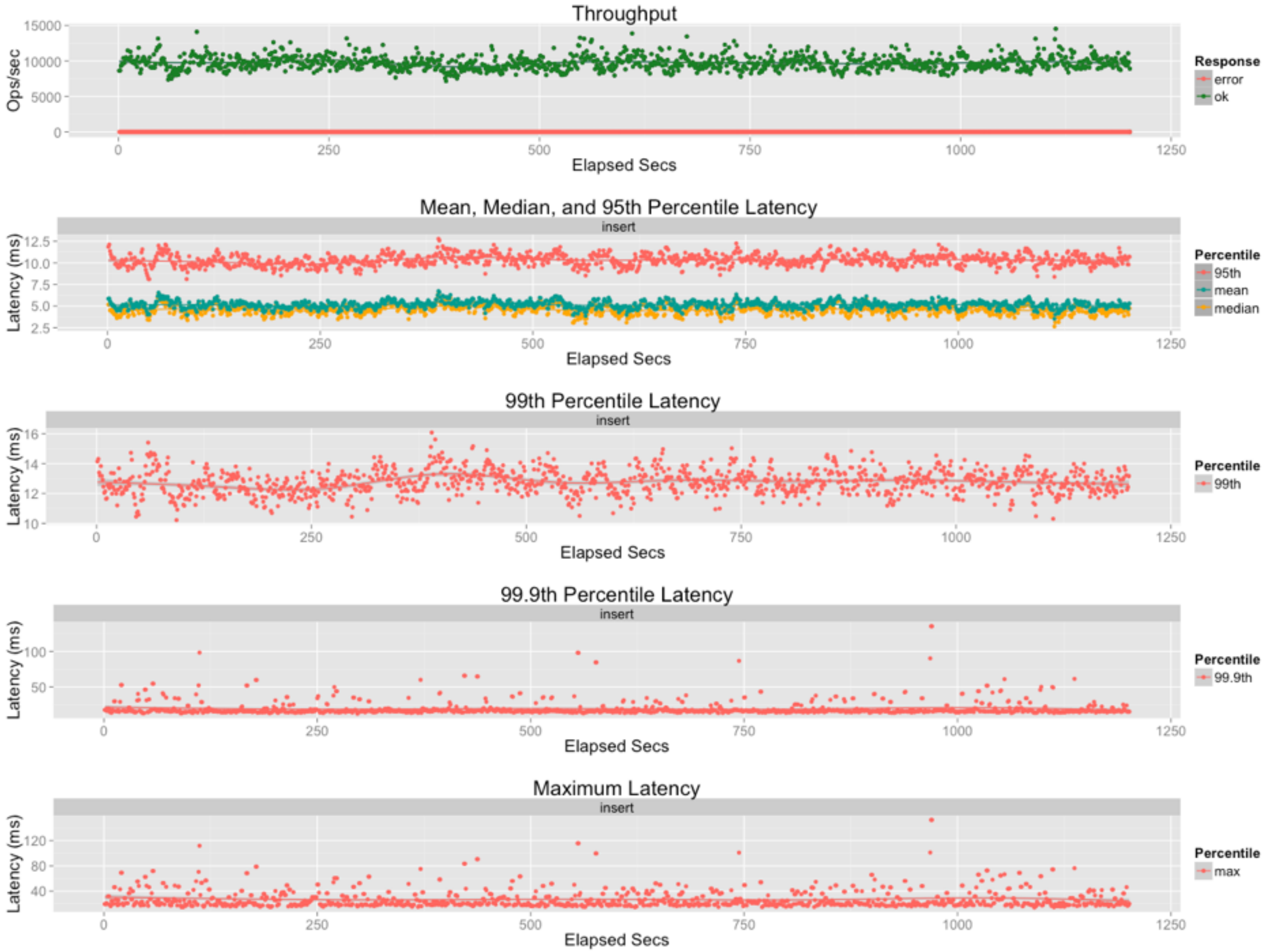
Bigset Design: Overview



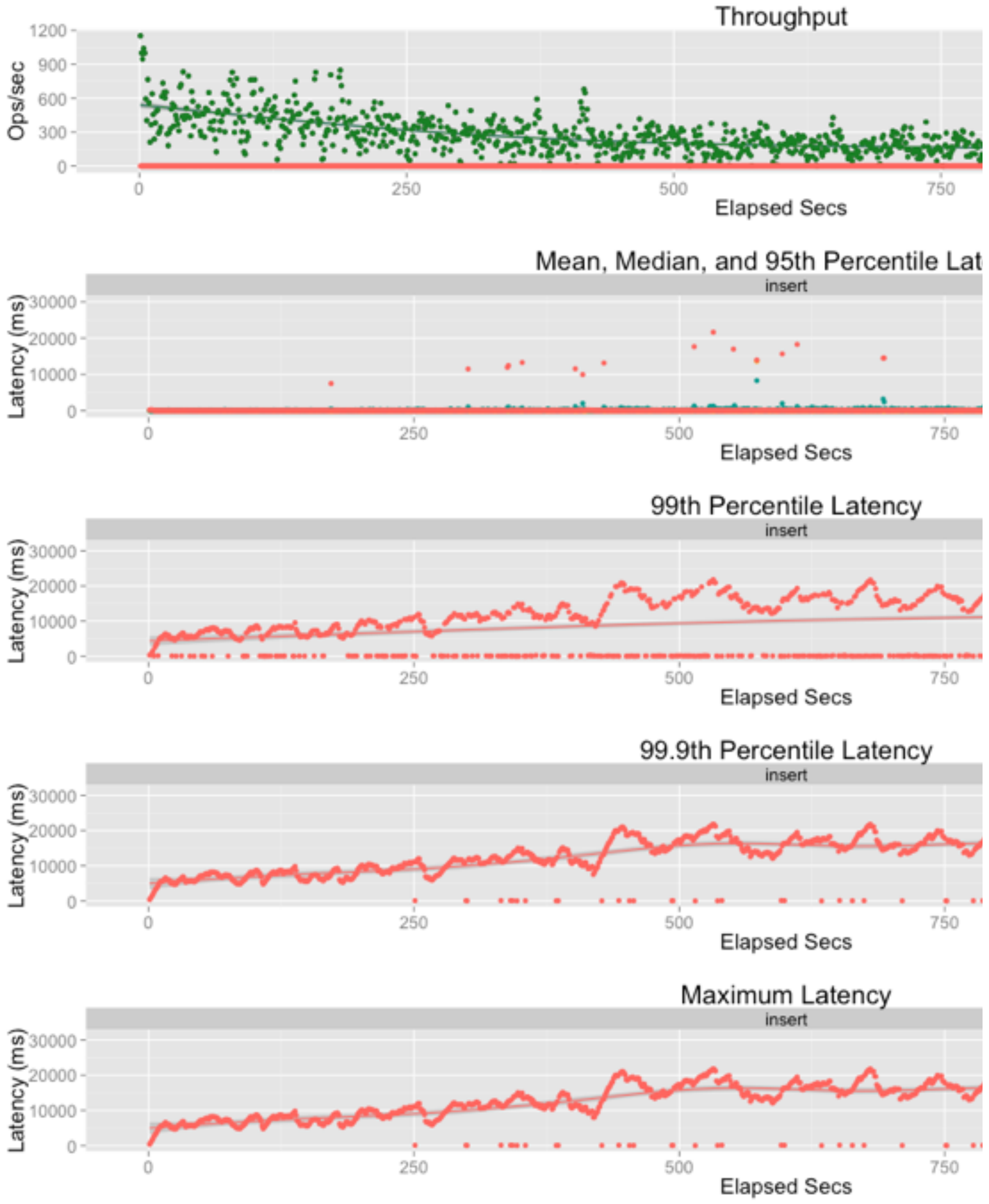

riak_dt_orswot
Version Vector
[{vnodeA, 10}, {vnodeB, 4}, {vnodeC,11}…]
Entries
Bob
[{vnodeA, 2}]
Cameron
[{vnodeB, 2}, {vnodeC, 5}]
Charlene
[{vnodeB, 4}]
Deferred Ops
[{vnodeA, 4}, {vnodeX, 22}]
Tim
[{vnodeB, 7}]
Zooey
vnode backend
{SetX, VnodeA, clock}
{SetX, VnodeA, tombstone}
{SetX, Bob, VnodeA, 2}
{SetX, Cameron, VnodeB, 2}
{SetX, Cameron, VnodeC, 5}
{SetX, Charlene, VnodeB, 4}
{SetX, endkey}
<<binary-logical-clock>>
<<binary-logical-clock>>
<<>>
<<>>
<<>>
<<>>
<<>>

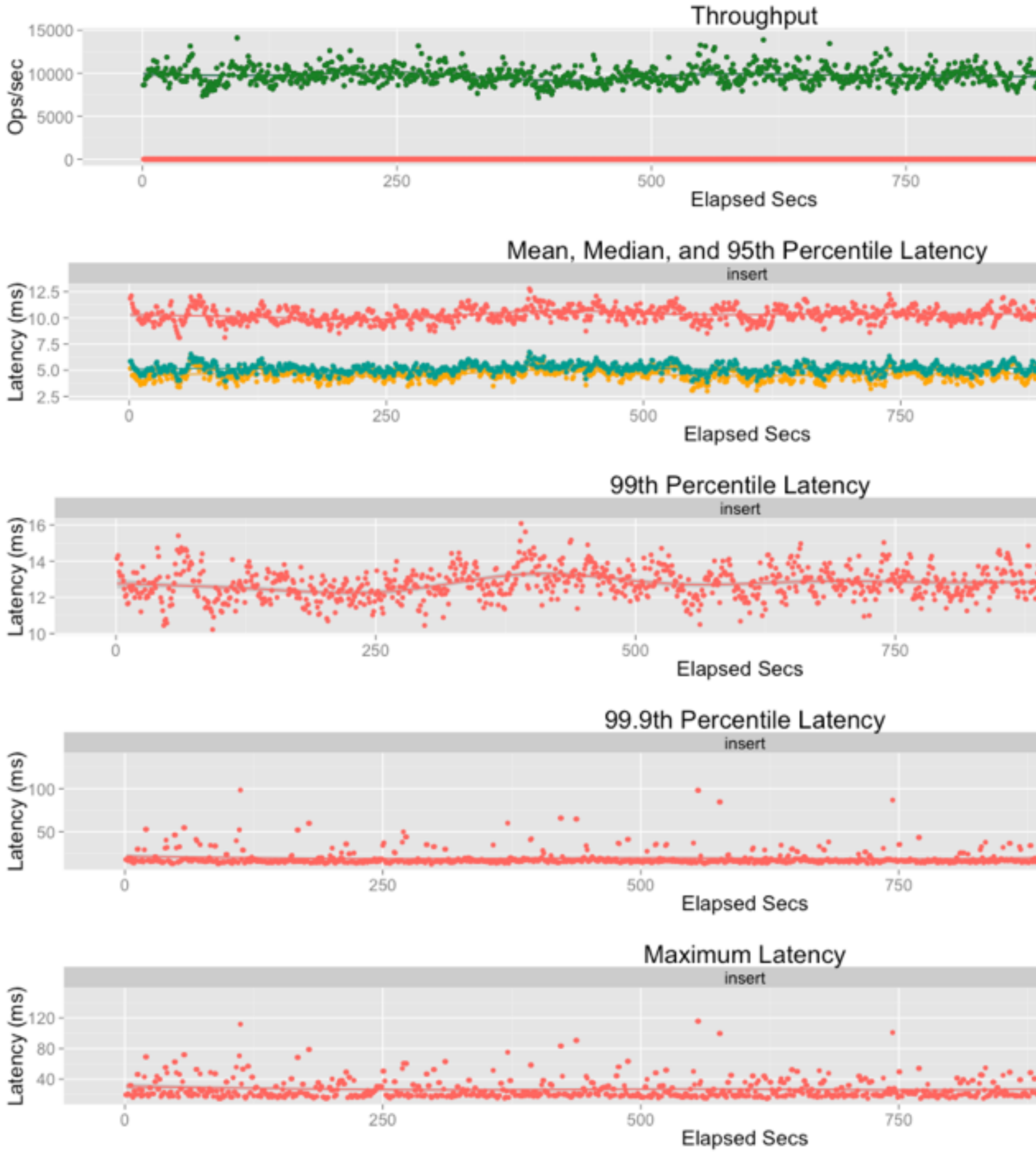
Initial Results



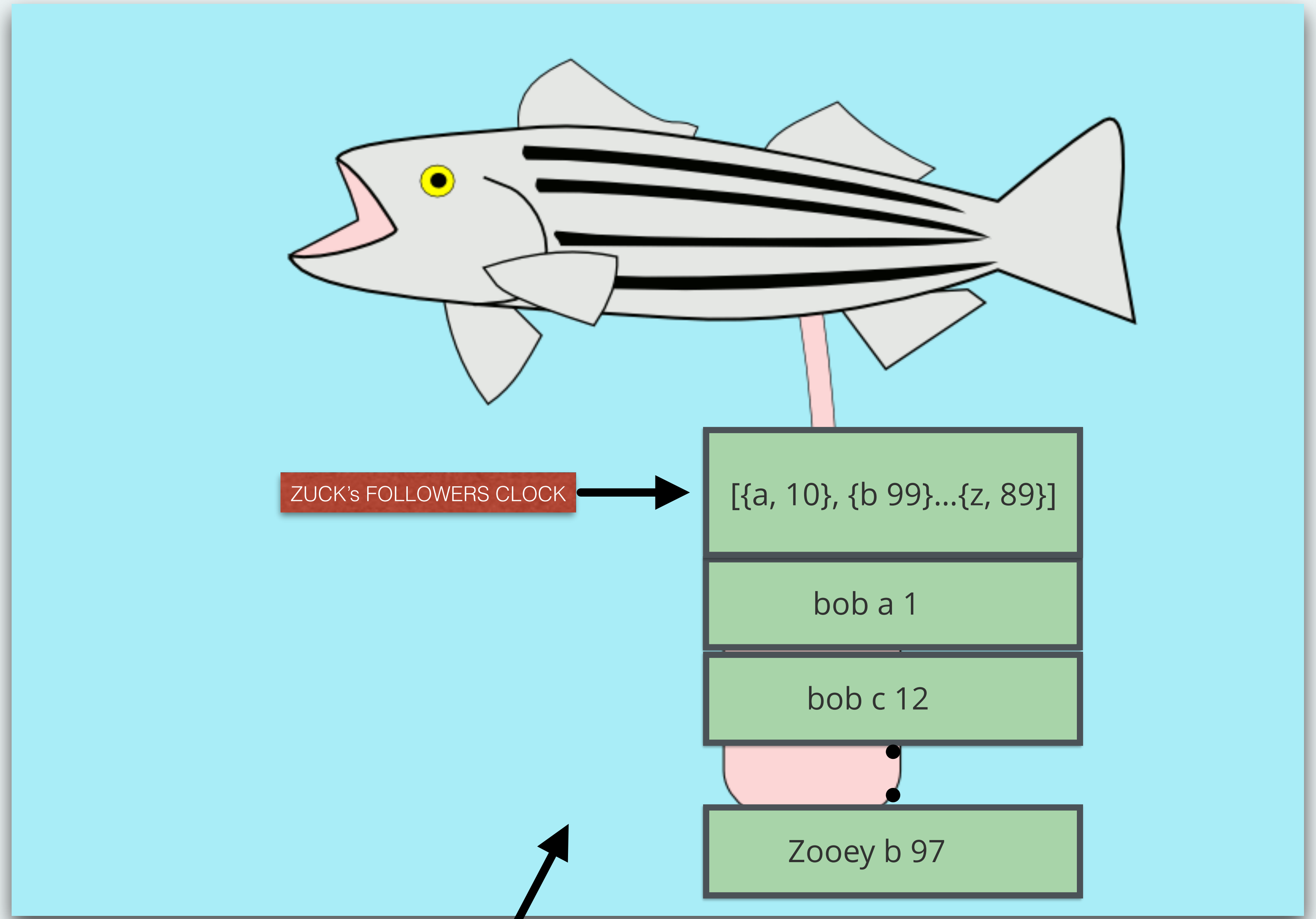


10k sets, 100k elements, 50 workers - write

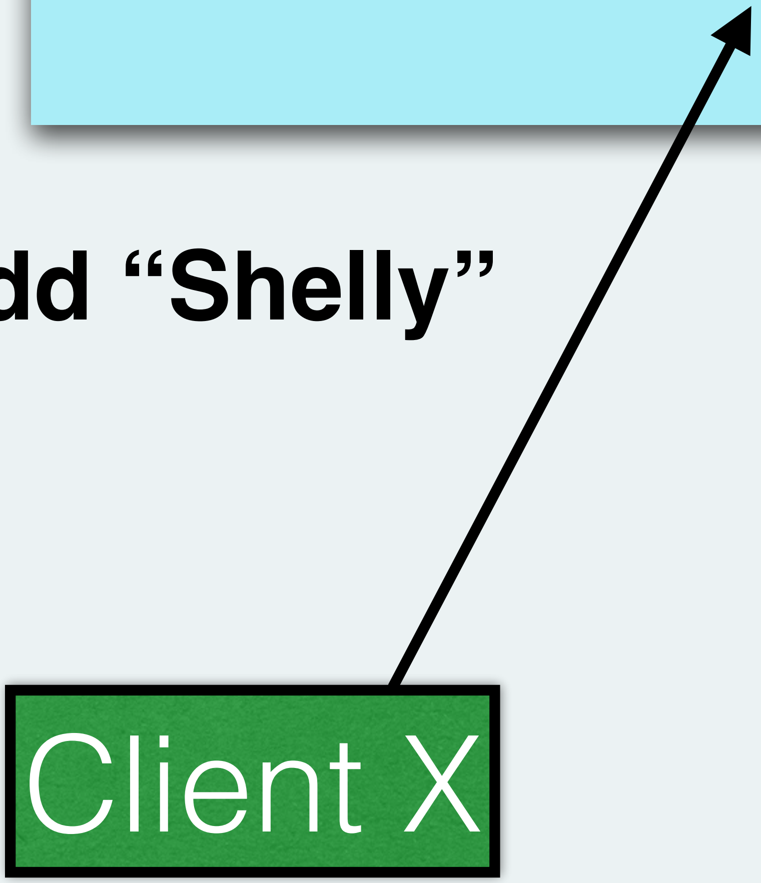


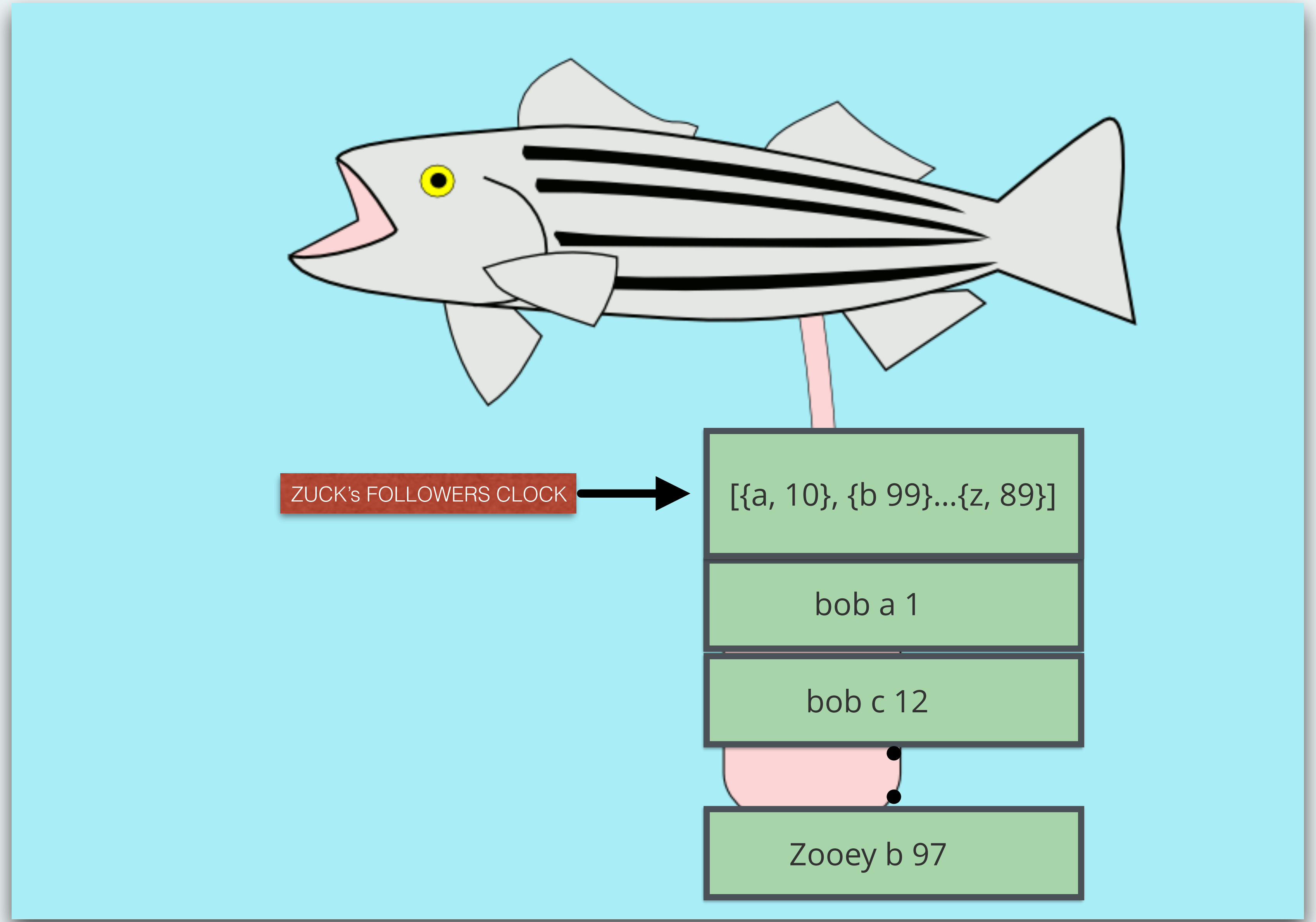


10k sets, 100k elements, 50 workers - write

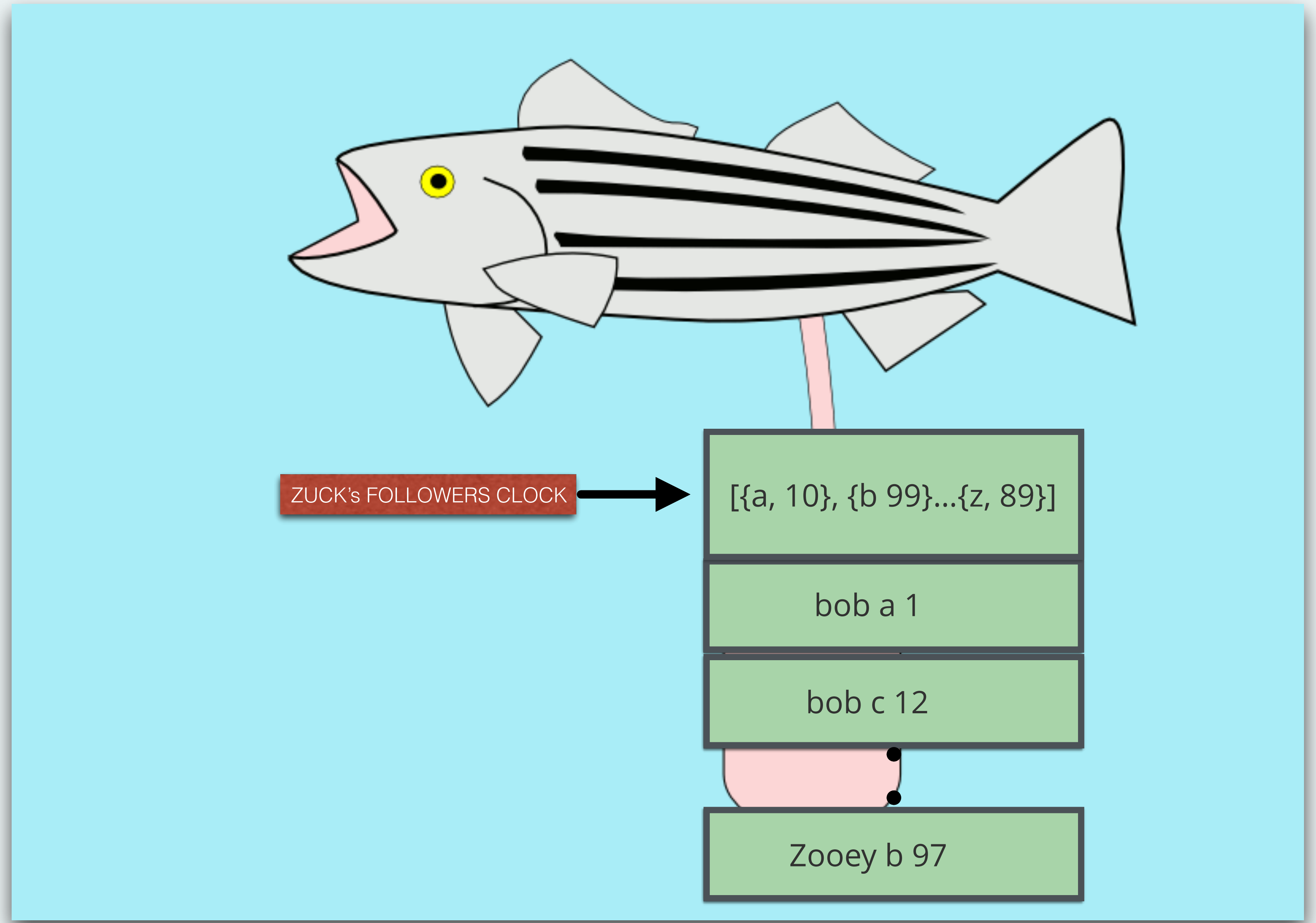

ZUCK's FOLLOWERS CLOCK
[{a, 10}, {b 99}...{z, 89}]
bob a 1
bob c 12
Zooey b 97

Add "Shelly"


Client X

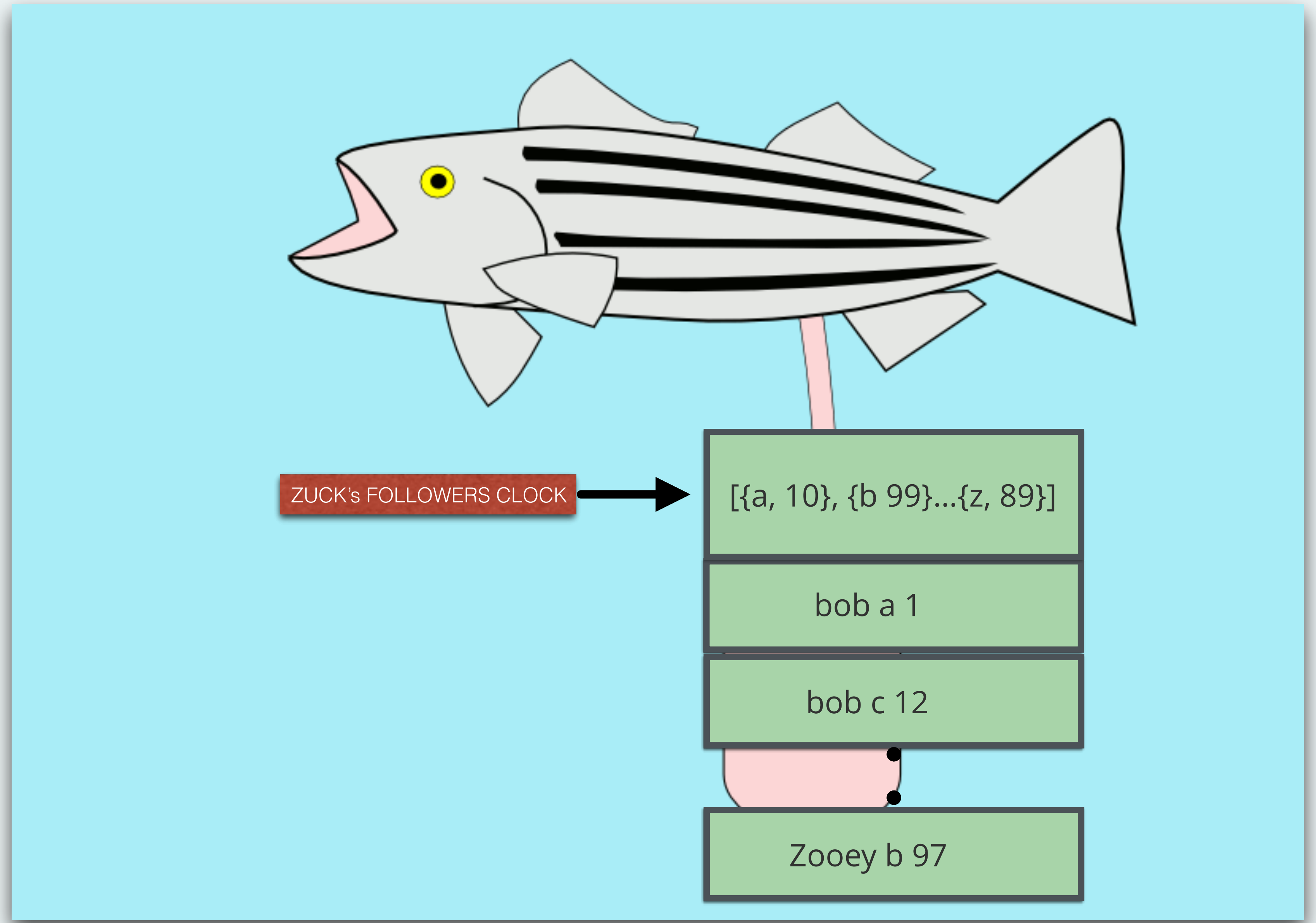

ZUCK's FOLLOWERS CLOCK
[{a, 10}, {b 99}...{z, 89}]
bob a 1
bob c 12
Zooey b 97

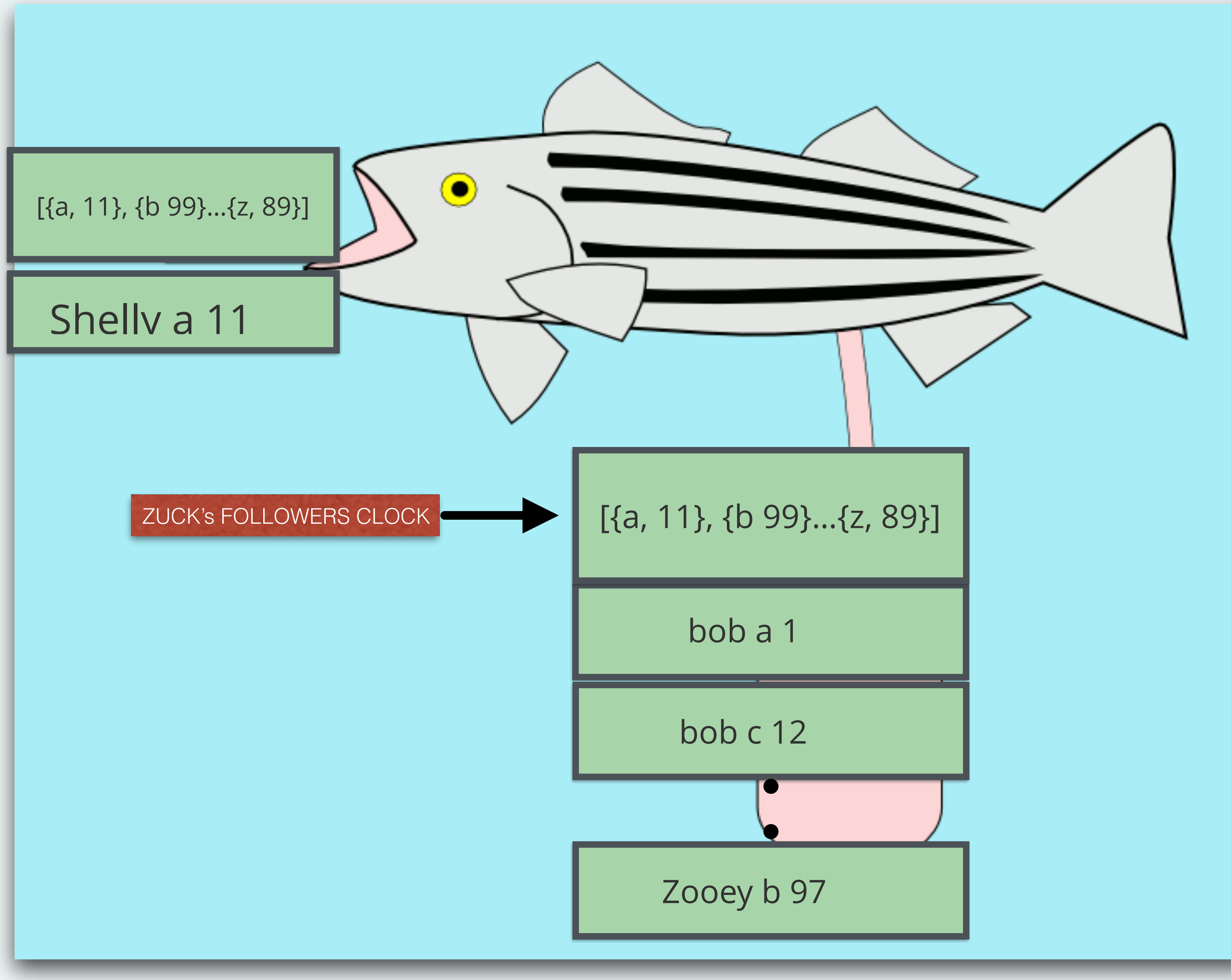
[{a, 10}, {b 99}...{z, 89}]

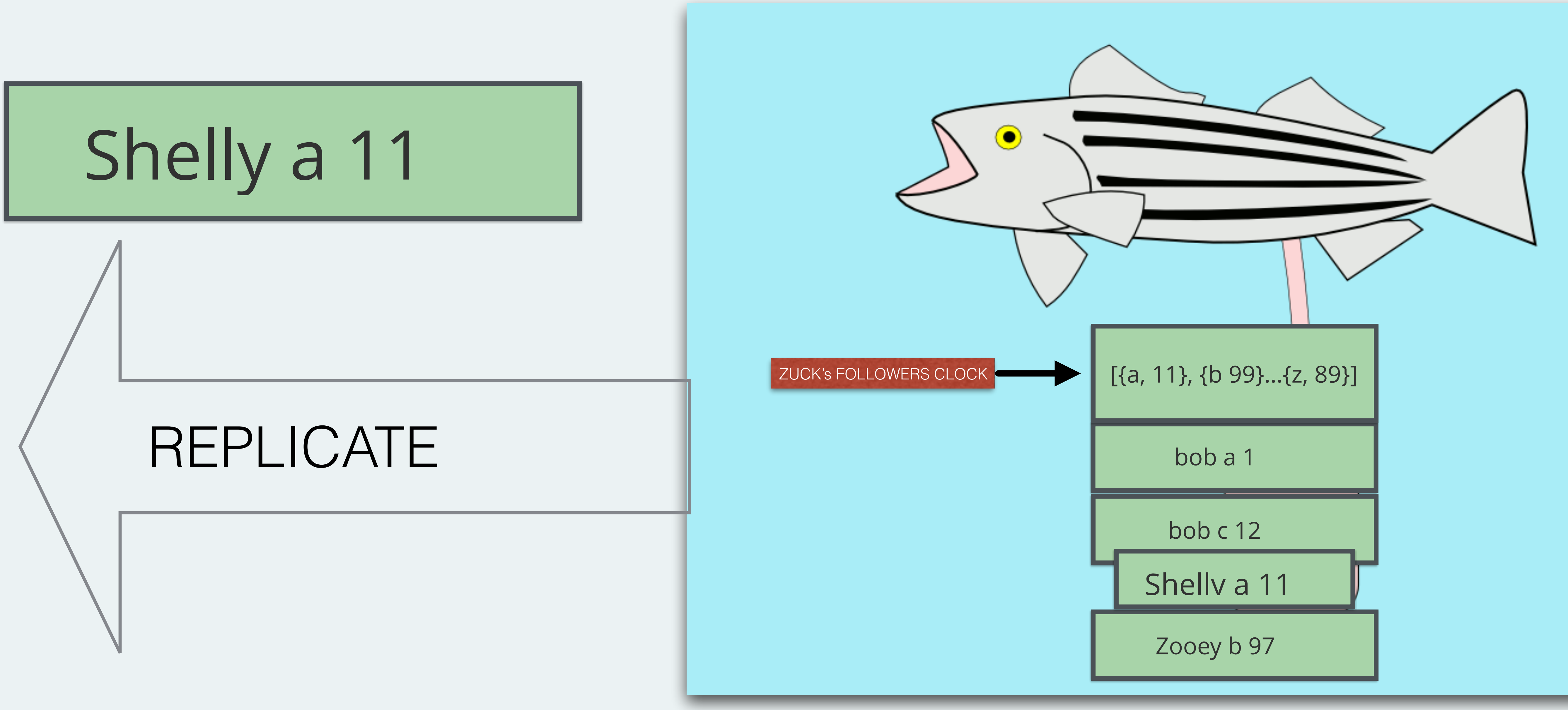


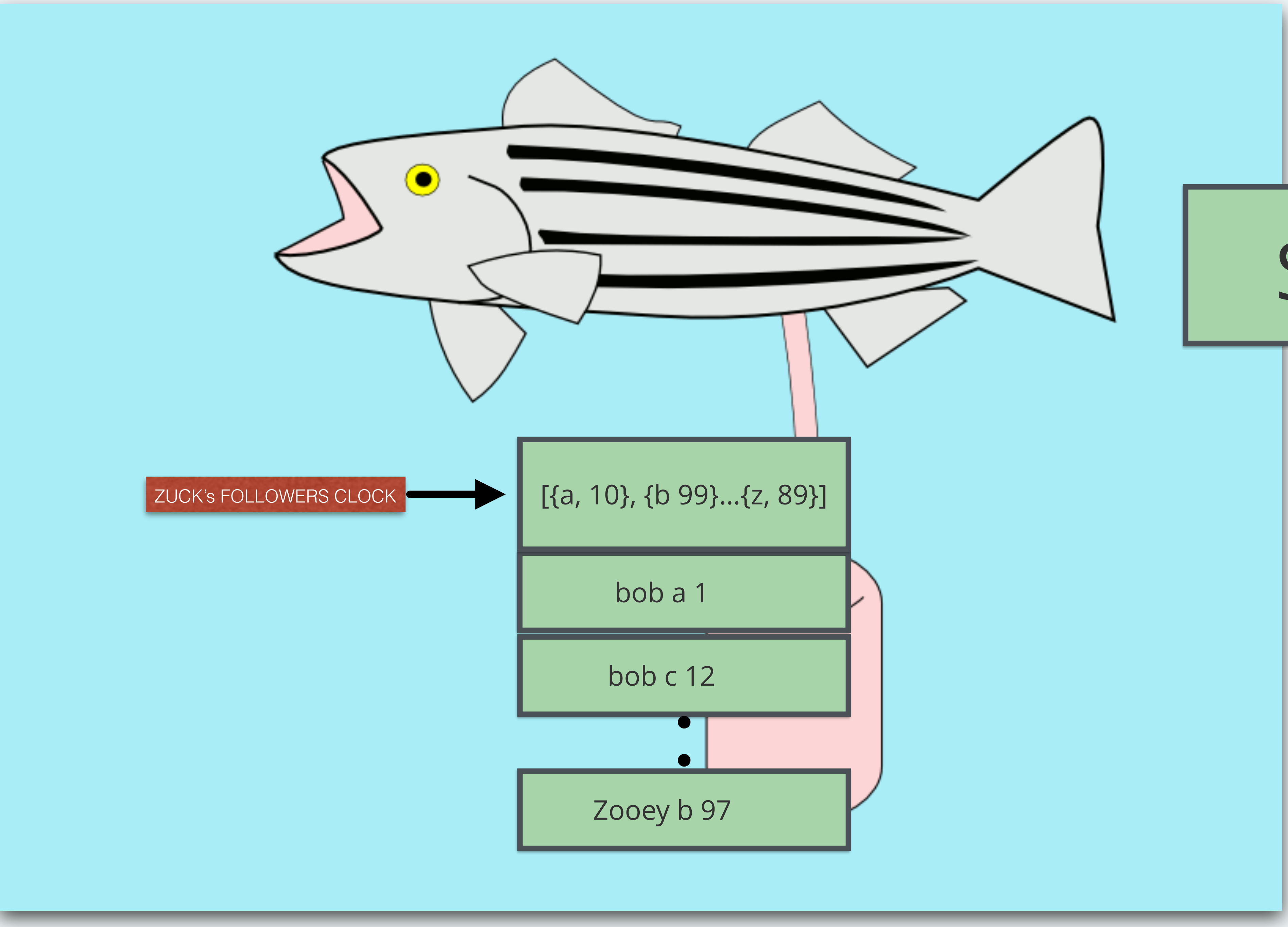
[{a, 11}, {b 99}...{z, 89}]

Shelly a 11

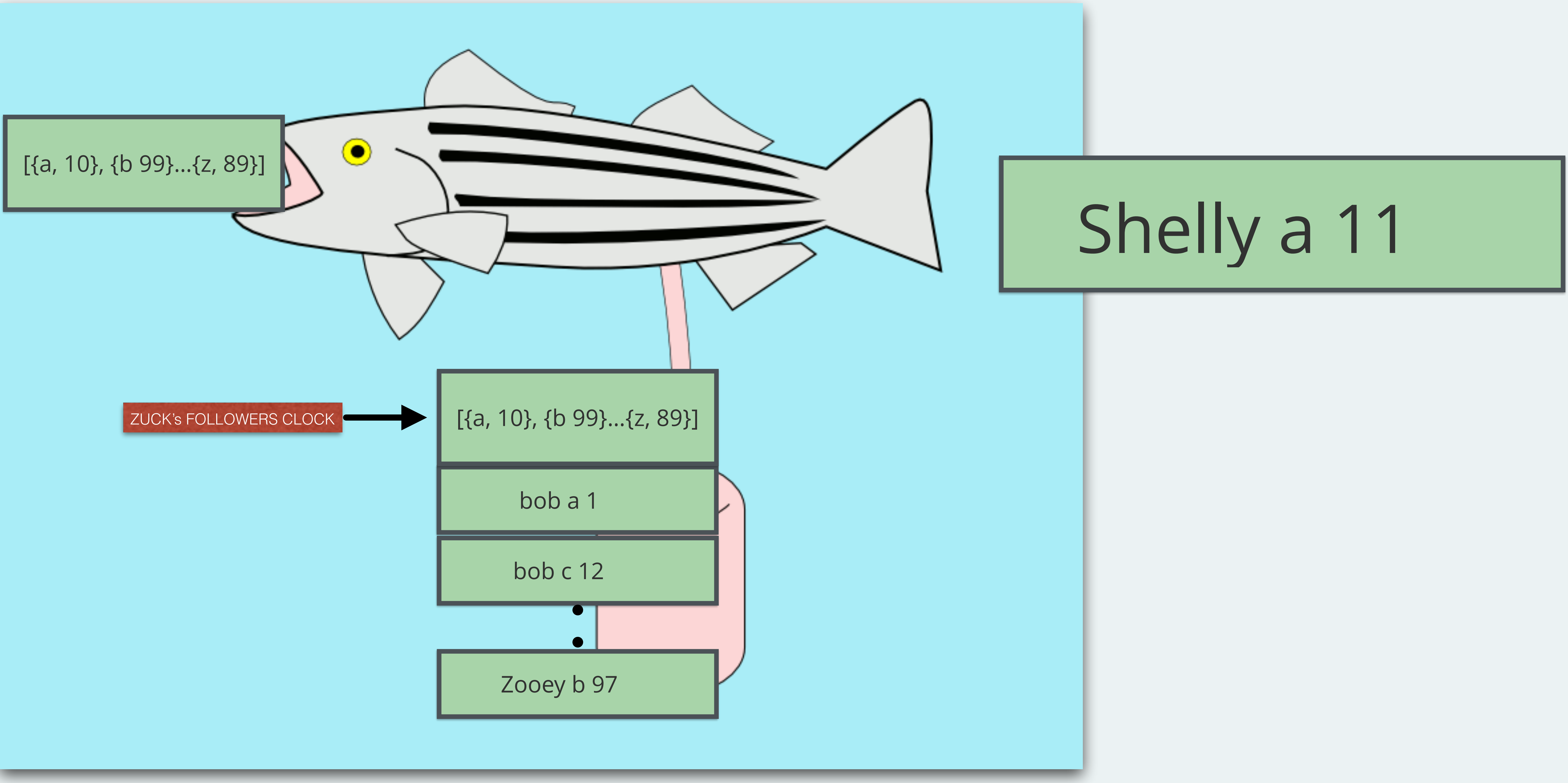


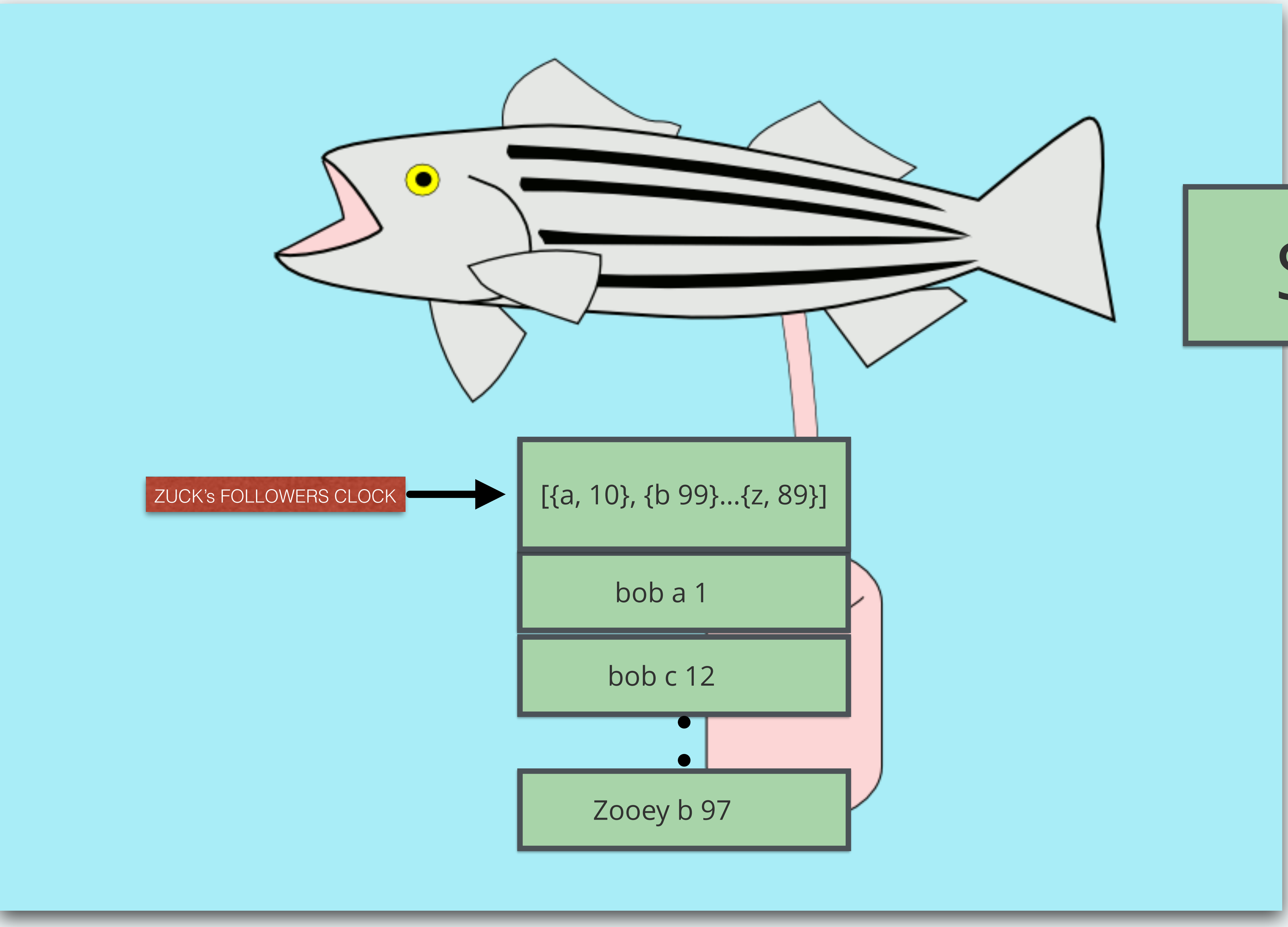

[{a, 11}, {b 99}...{z, 89}]
Shelly a 11
ZUCK's FOLLOWERS CLOCK
[{a, 11}, {b 99}...{z, 89}]
bob a 1
bob c 12
Zooey b 97


Shelly a 11
REPLICATE
ZUCK's FOLLOWERS CLOCK
[{a, 11}, {b 99}...{z, 89}]
bob a 1
bob c 12
Shelly a 11
Zooey b 97



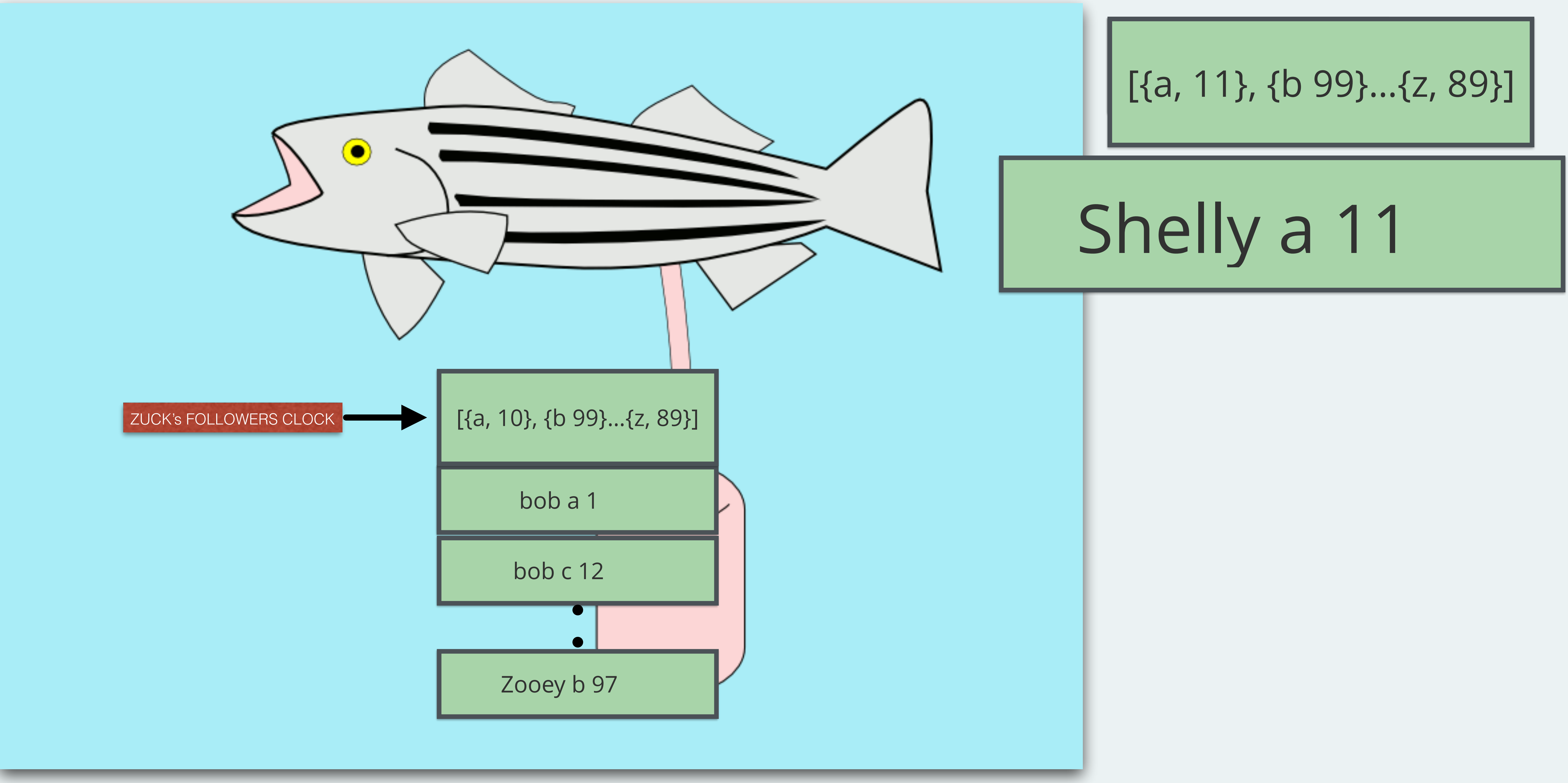
Shelly a 11


[{a, 10}, {b 99}...{z, 89}]
Shelly a 11
ZUCK's FOLLOWERS CLOCK
[{a, 10}, {b 99}...{z, 89}]
bob a 1
bob c 12
Zooey b 97

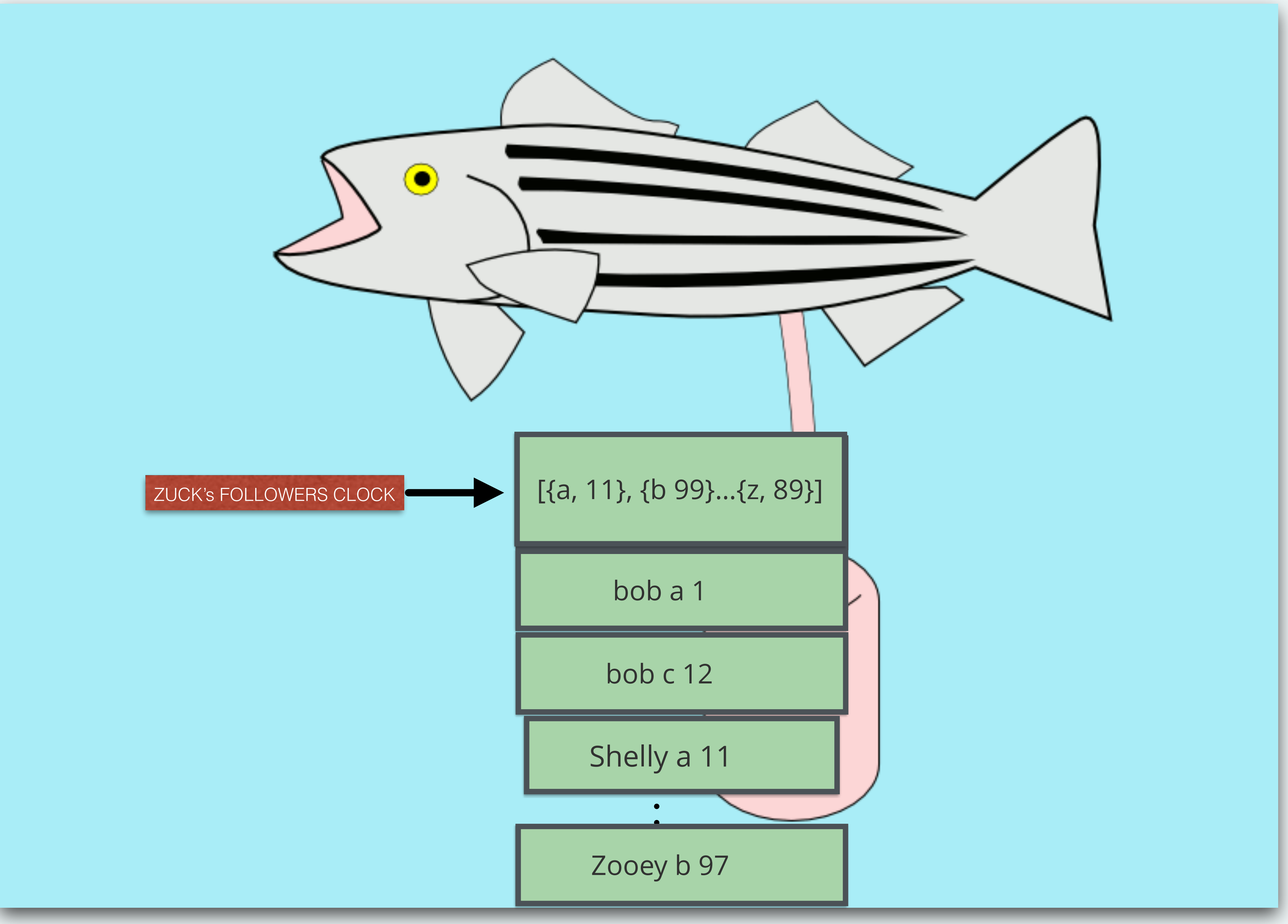


[{a, 10}, {b 99}...{z, 89}]

Shelly a 11



[{a, 11}, {b 99}...{z, 89}]
Shelly a 11
ZUCK's FOLLOWERS CLOCK
[{a, 10}, {b 99}...{z, 89}]
bob a 1
bob c 12
Zooey b 97


ZUCK's FOLLOWERS CLOCK
[{a, 11}, {b 99}...{z, 89}]
bob a 1
bob c 12
Shelly a 11
Zooey b 97

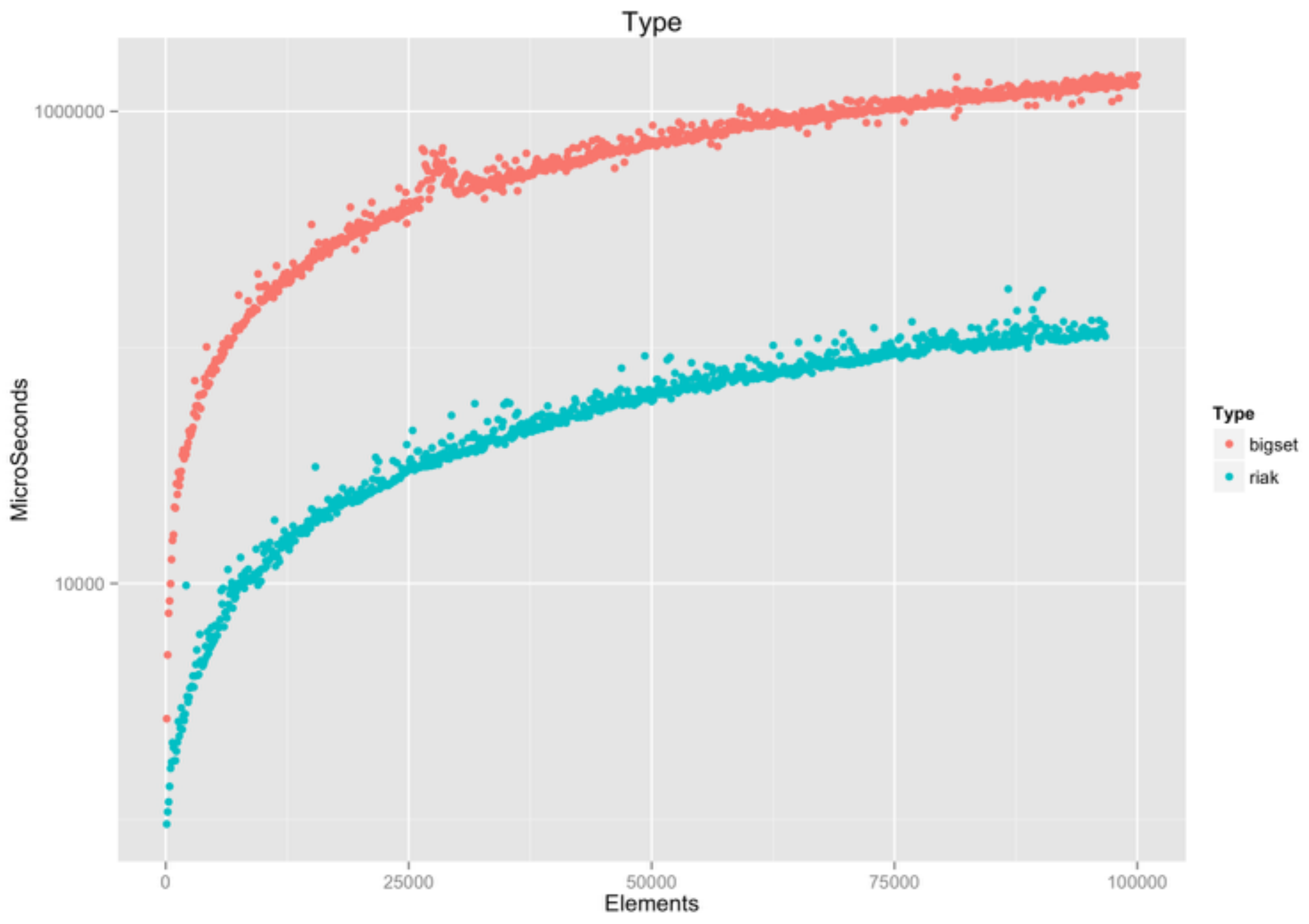
THAT’S IT!

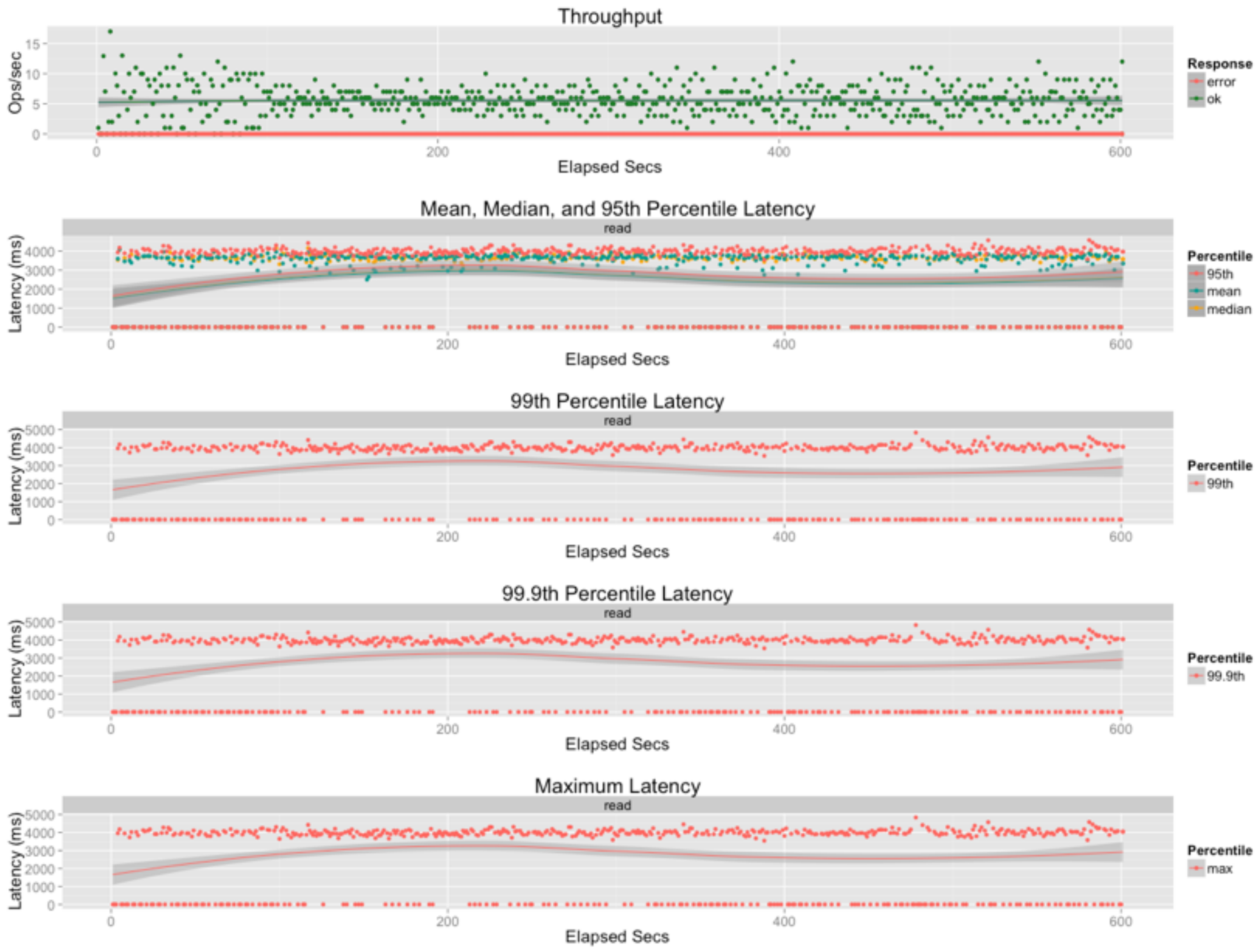
THAT'S IT?

- Reads!
- Version Vector!
- Hand-Off!
- AAE!

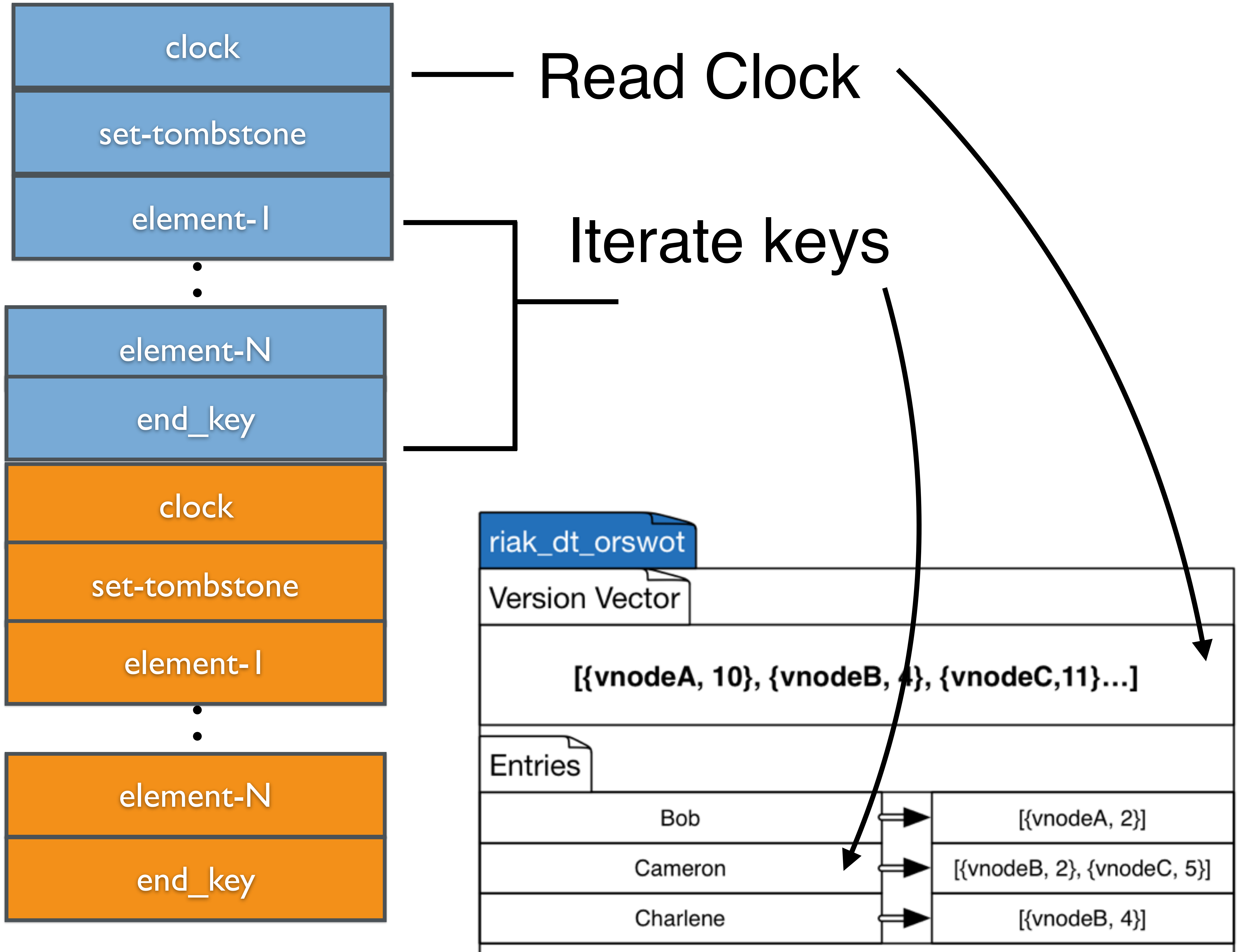
Reads?

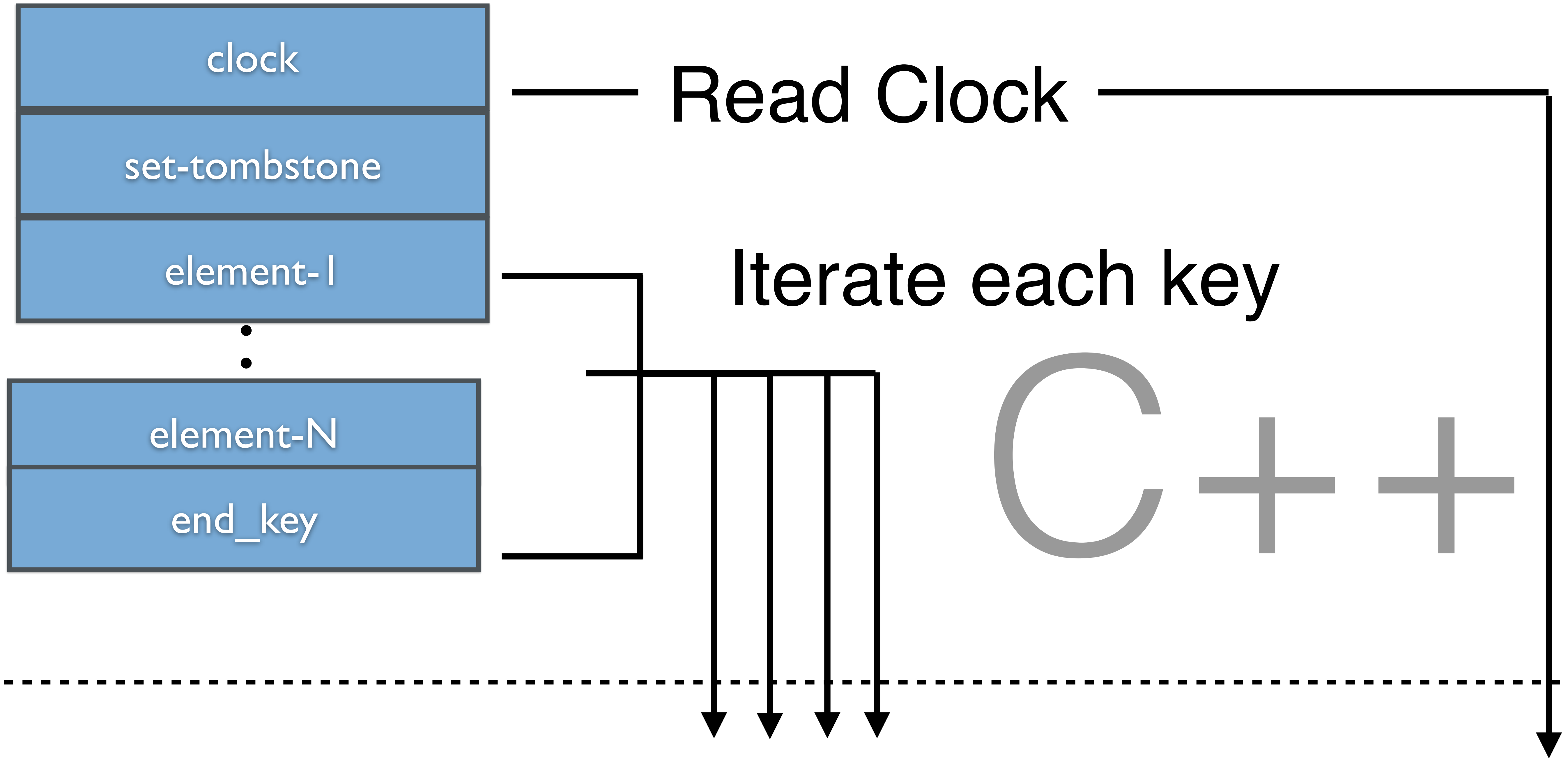
Initial Read Results



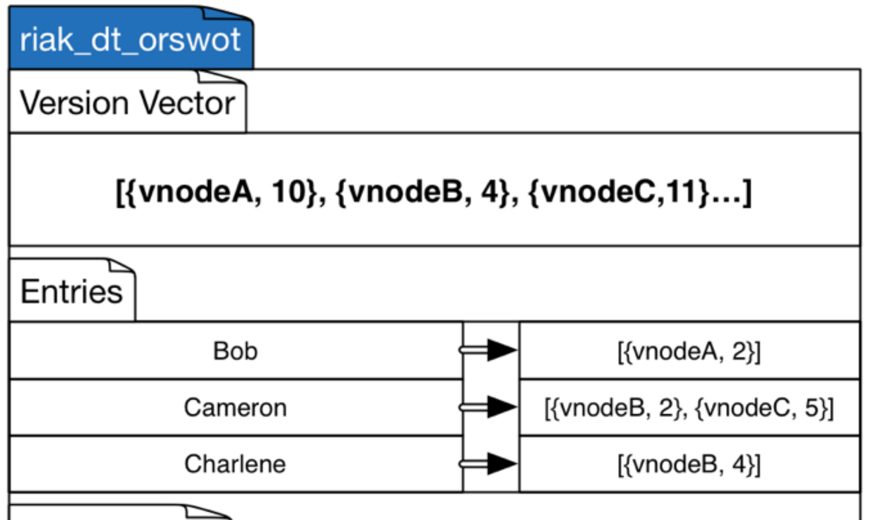


10k sets, 100k elements, 20 workers - read


clock
set-tombstone
element-1
element-N
end_key
clock
set-tombstone
element-1
element-N
end_key
Read Clock
Iterate keys
riak_dt_orswot
Version Vector
[{vnodeA, 10}, {vnodeB, 4}, {vnodeC,11}...]
Entries
Bob
[{vnodeA, 2}]
Cameron
[{vnodeB, 2}, {vnodeC, 5}]
Charlene
[{vnodeB, 4}]

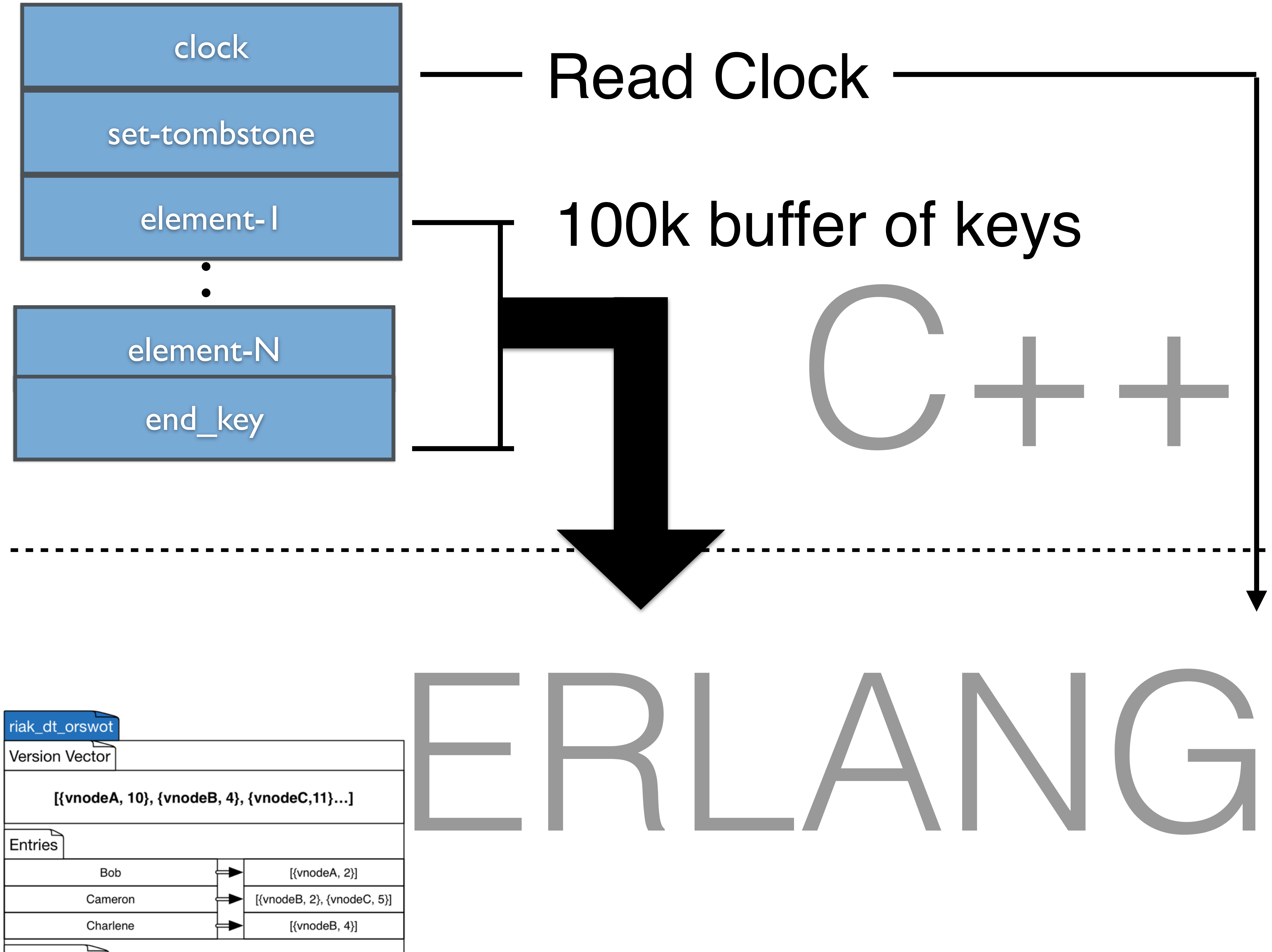


ERLANG



[{vnodeA, 10}, {vnodeB, 4}, {vnodeC,11}...]

Bob	[{vnodeA, 2}]
Cameron	[{vnodeB, 2}, {vnodeC, 5}]
Charlene	[{vnodeB, 4}]

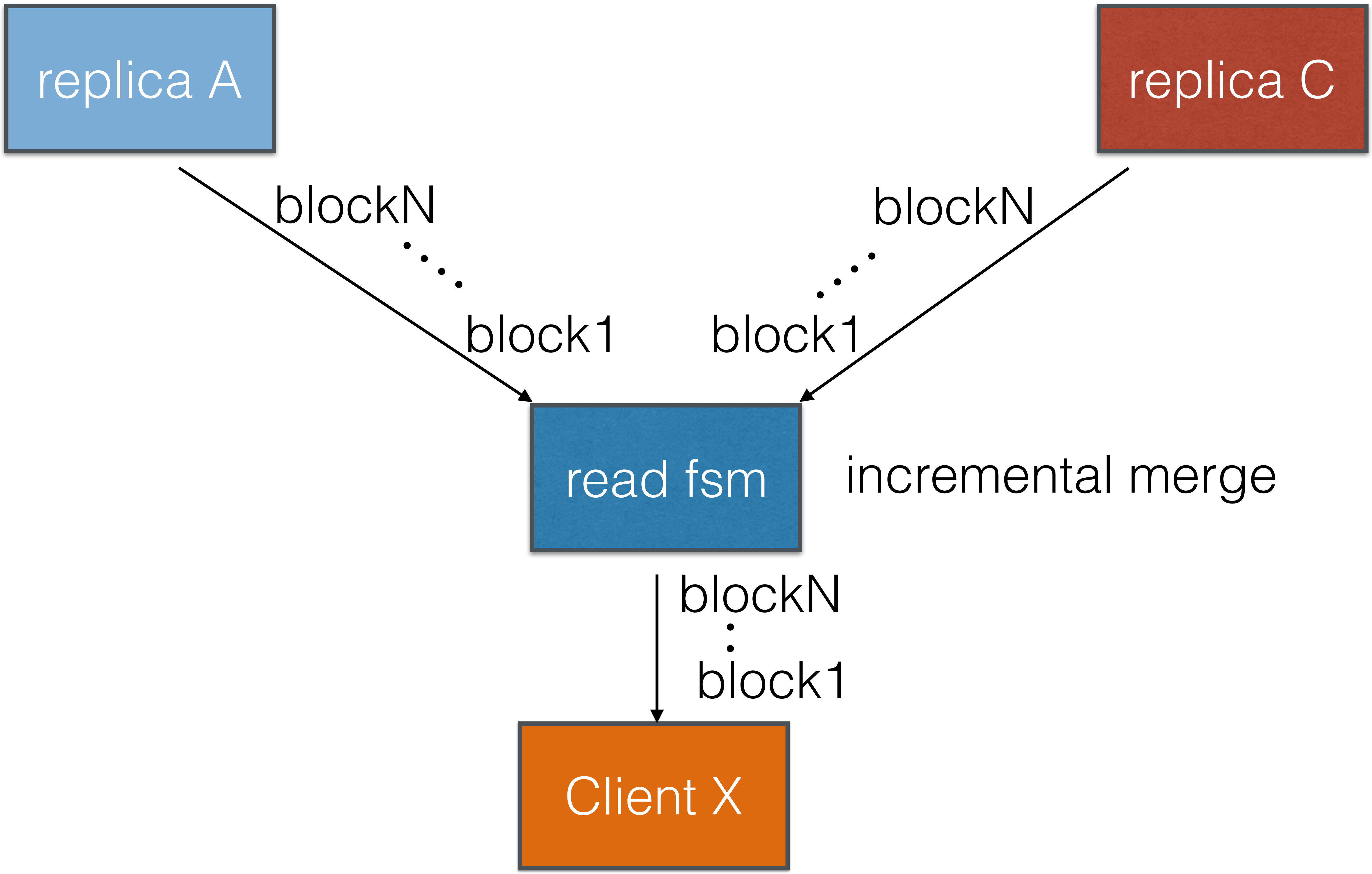

clock
set-tombstone
element-1
element-N
end_key
Read Clock
100k buffer of keys
C++
ERLANG
riak_dt_orswot
Version Vector
[{vnodeA, 10}, {vnodeB, 4}, {vnodeC,11}...]
Entries
Bob
[{vnodeA, 2}]
Cameron
[{vnodeB, 2}, {vnodeC, 5}]
Charlene
[{vnodeB, 4}]

```
<<Set, \0, $c, \0, Actor, \0>>

<<Set, \0, $e, \0, Element-1, \0,
Actor, Cnt:/64/big-unsigned-
integer>>

⋮

<<Set, \0, $e, \0, Element-1, \0,
Actor, Cnt:/64/big-unsigned-
integer>>

<<Set, \0, %z, \0, \0>>
```

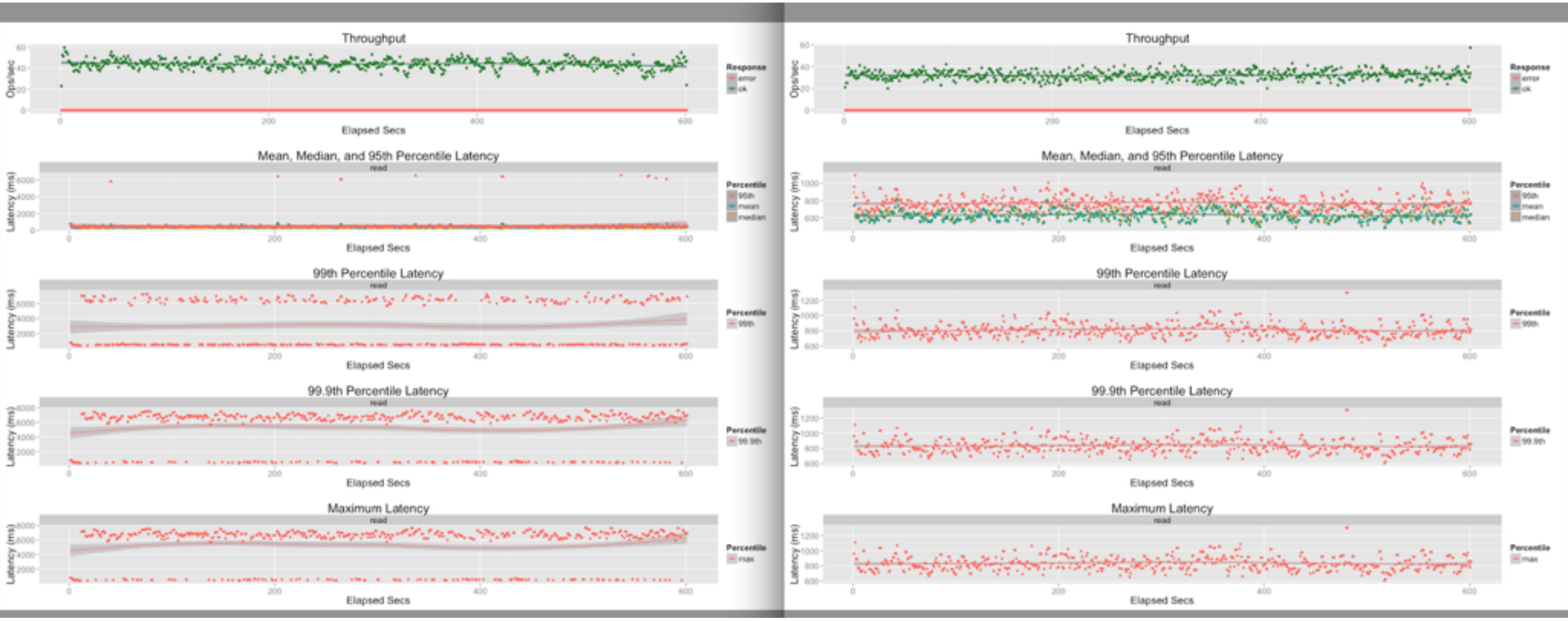
No Sext

No T2B

Bigset Design: read

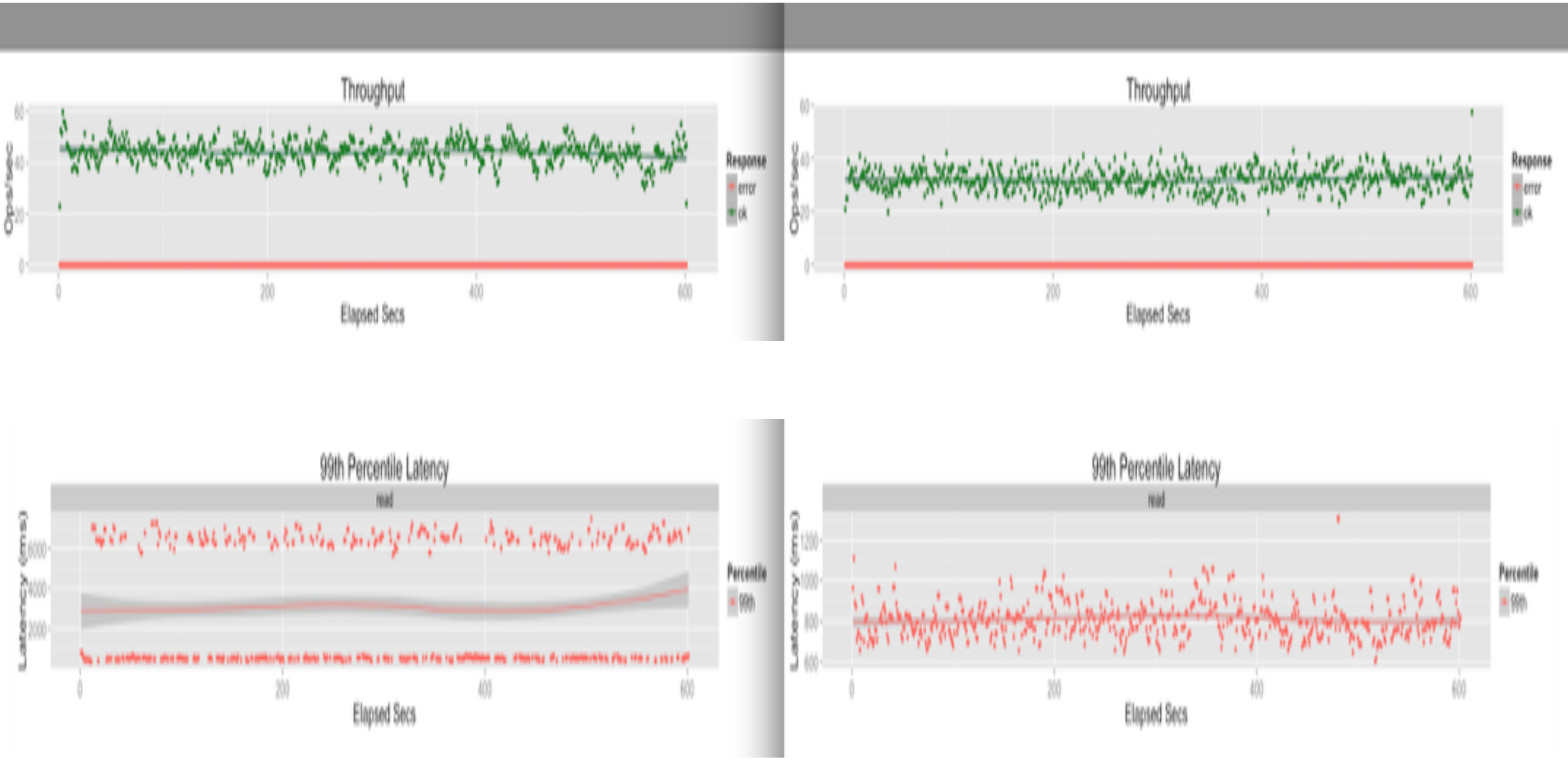


Reads Today



10k sets, 100k elements, 20 workers - read

Reads Today



10k sets, 100k elements, 20 workers - read

Full Set Read or Queries?

Why read the whole set?

‘Cos you HAVE TO!

Full Set Read or Queries?

Why read the whole set?

Full Set Read or Queries?

Why read the whole set?

‘Cos you HAVE TO!

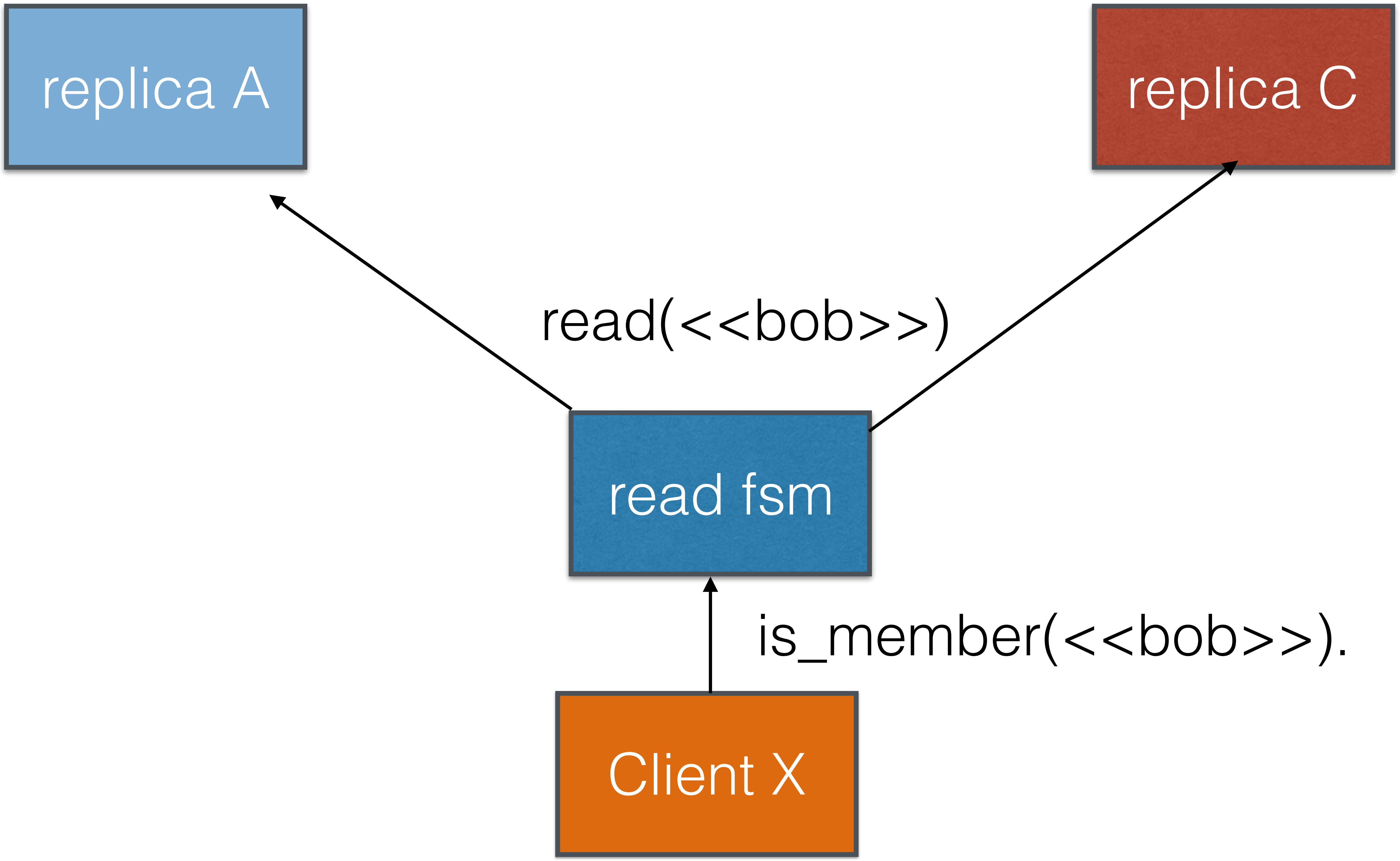
Bigset Queries

- Subset
    - Is Member?
- Range queries SORTED!
- Pagination

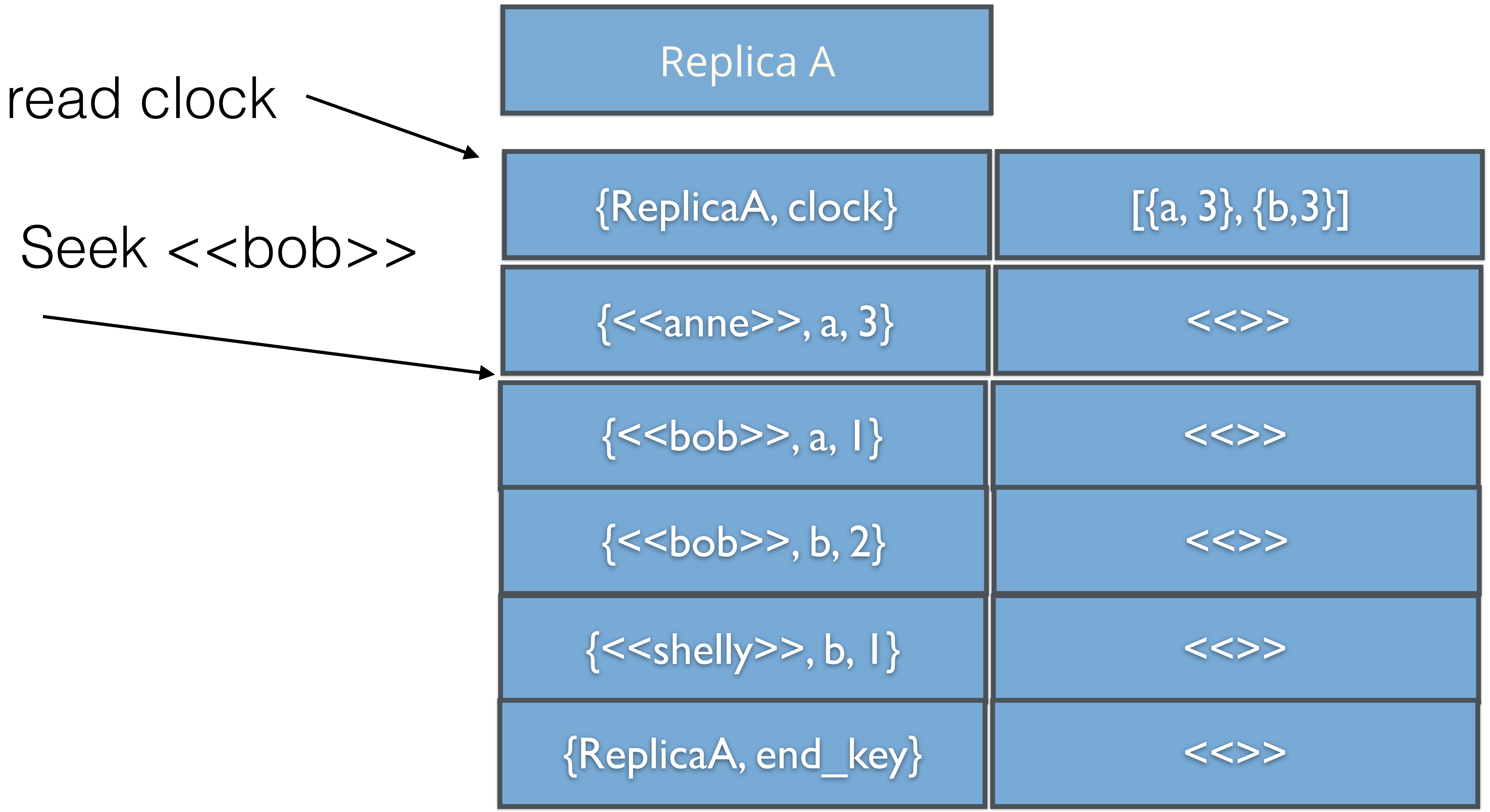
Removes

- Observed-Remove - context
- Requires _some kind_ of read
  - cheap membership check

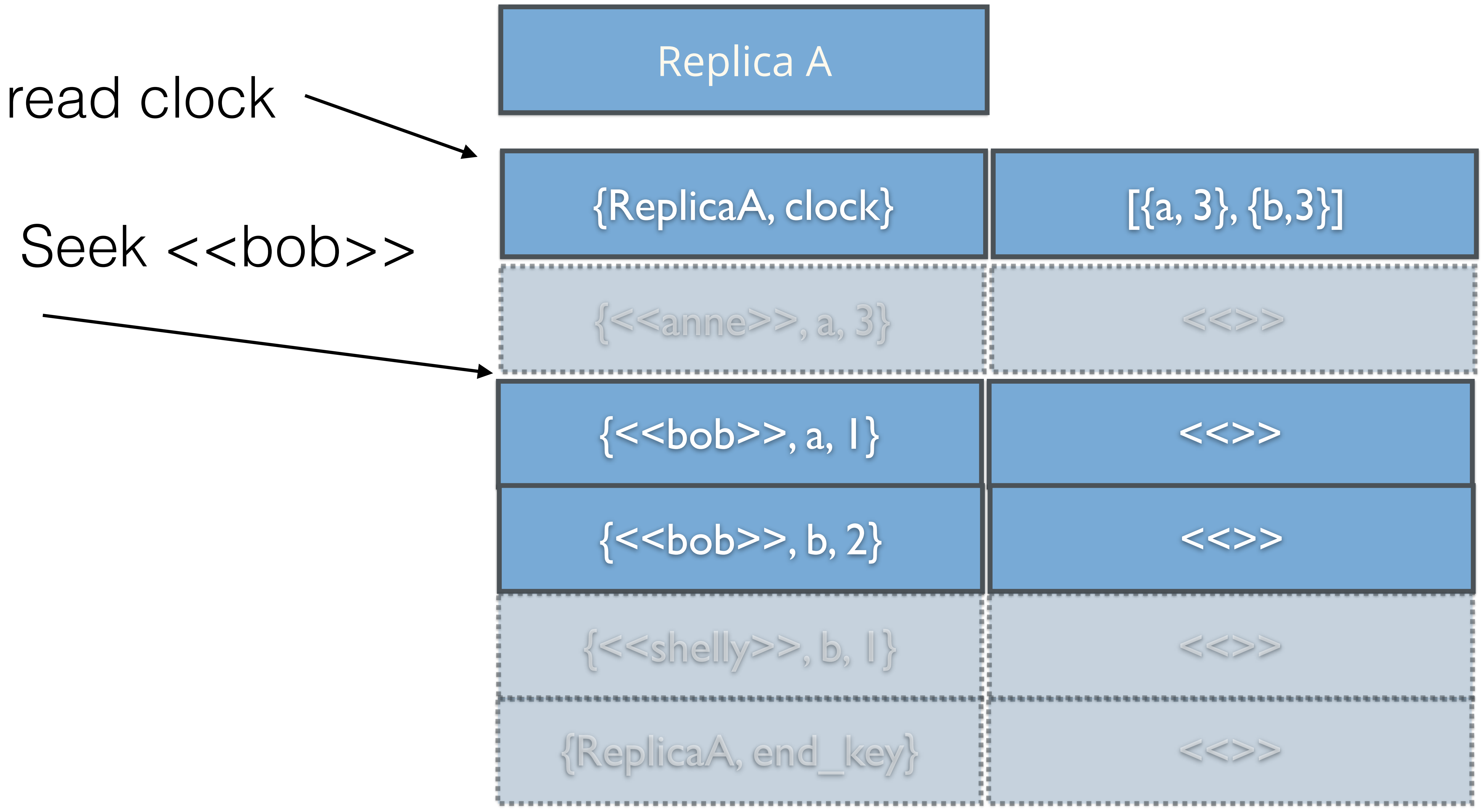
Is Member(X)



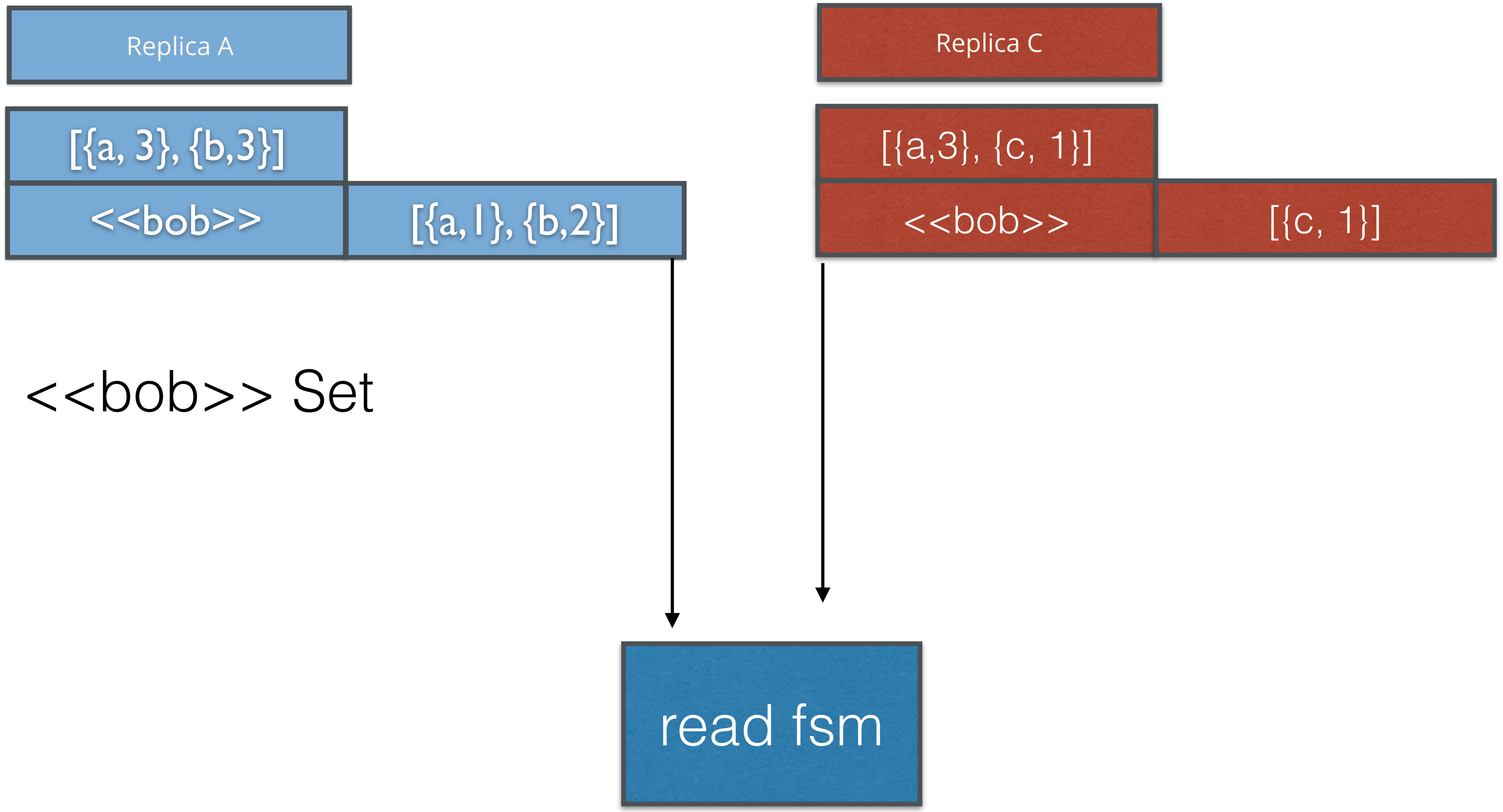
Is Member(X)


read clock
Seek <<bob>>
Replica A
{ReplicaA, clock}
[{a, 3}, {b,3}]
{<<anne>>, a, 3}
<<>>
{<<bob>>, a, 1}
<<>>
{<<bob>>, b, 2}
<<>>
{<<shelly>>, b, 1}
<<>>
{ReplicaA, end_key}
<<>>

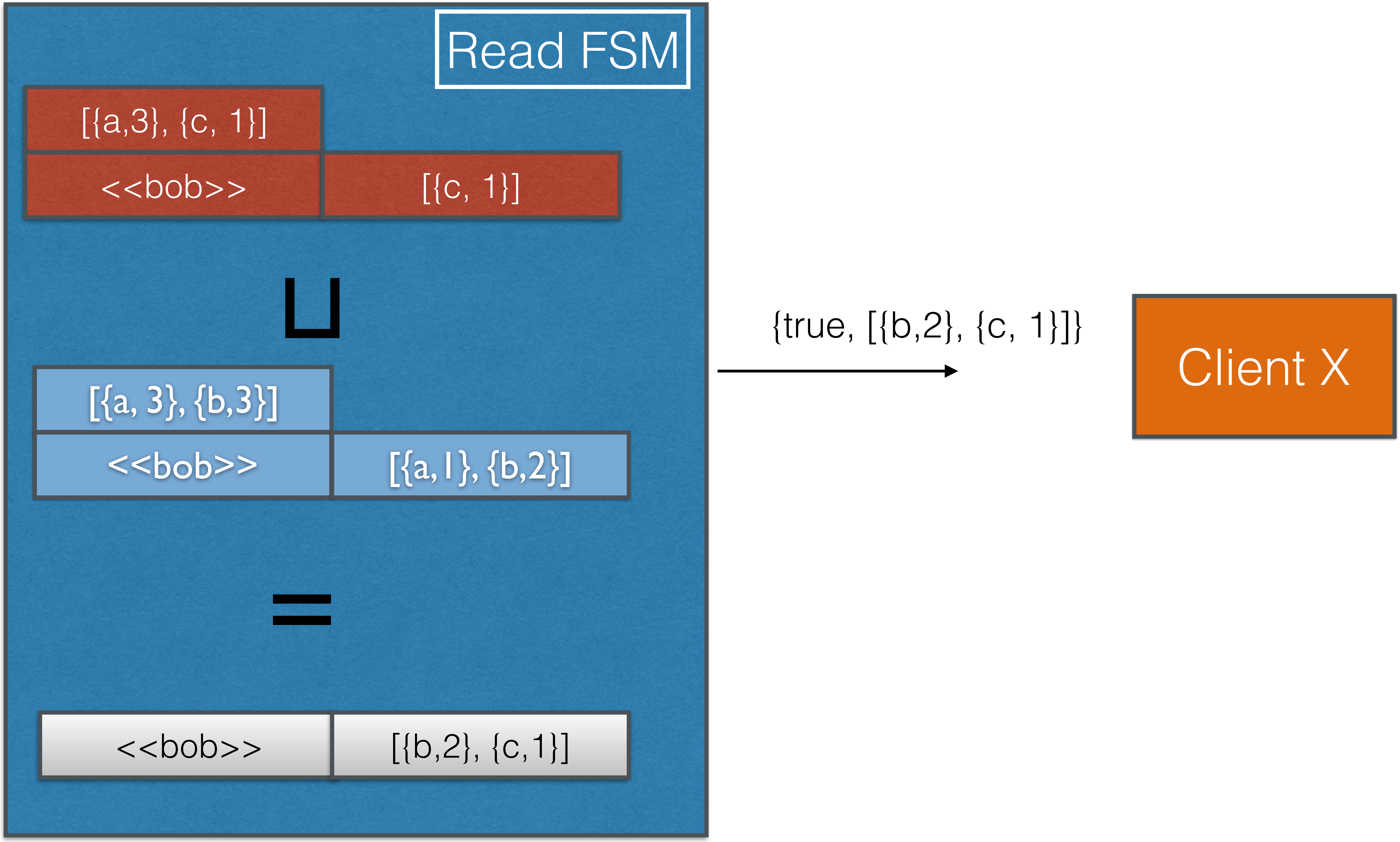
Is Member(X)


read clock
Seek <<bob>>
Replica A
{ReplicaA, clock}
[{a, 3}, {b,3}]
{<<anne>>, a, 3}
<<>>
{<<bob>>, a, 1}
<<>>
{<<bob>>, b, 2}
<<>>
{<<shelly>>, b, 1}
<<>>
{ReplicaA, end_key}
<<>>

Is Member(X)



Is Member(X)



Next?

- Other “Big” Types - Maps
- Quorum Read Secondary Indexes
- Big Sets of Maps - Tables
- Joins? SQL?

Summary

- CRDTs make eventual consistency easier on developers
- There exists an Optimised Add-Wins Set…
- It takes more more than a lib
- A little engineering goes a long way

Bigset Paper
https://arxiv.org/abs/1605.06424

THANK YOU!

WE'RE HIRING!

UK Client Service Engineer

Developer Advocate EMEA

bashojobs.theresumator.com

VISIT OUR STAND

Experience our Riak TS demo and be entered to win a Scalextric set!

Get your invitation to our IoT Riak TS Roadshow

basho