SCALAR

Scala Conference in Central Europe

GFT

Conscious Consistency with Akka Cluster, CRDTs and Distributed Data.

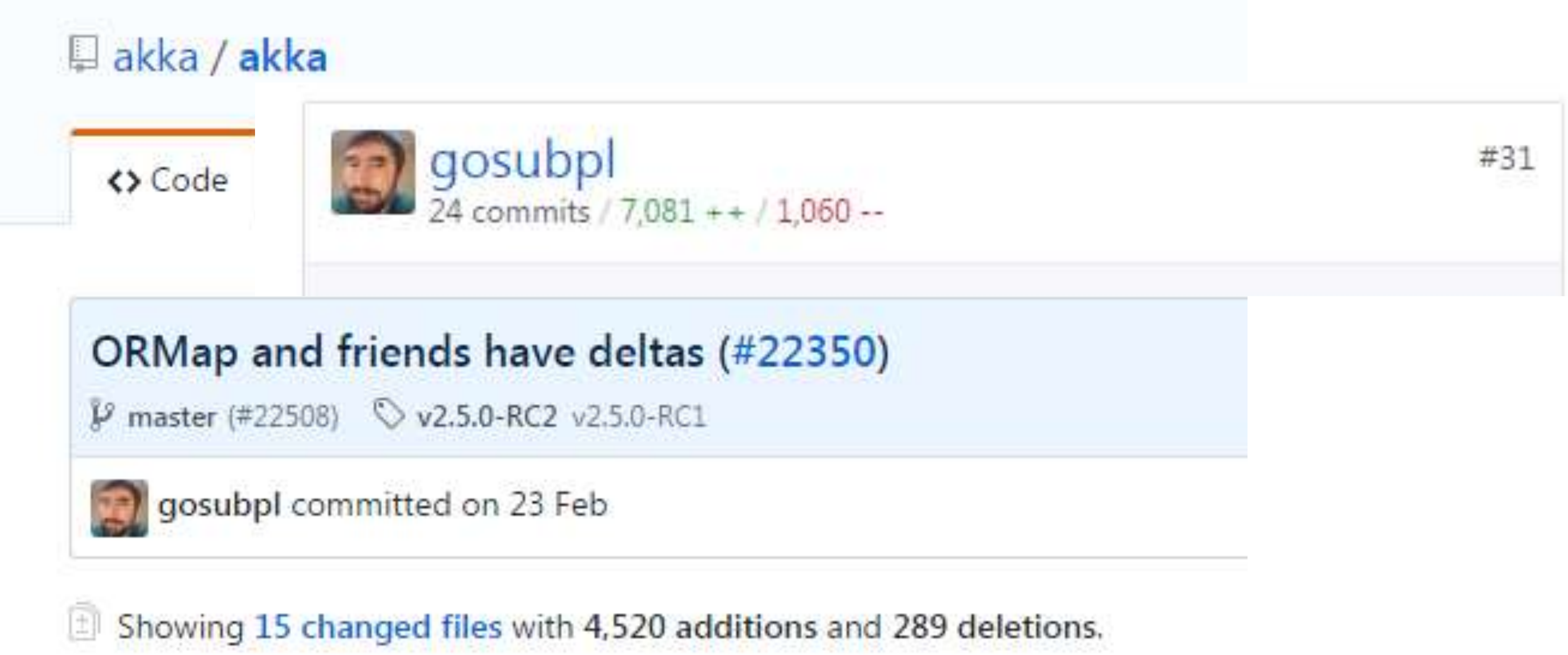
Jan Pustelnik @gosubpl

6 April 2017

GoSub is Vintage ☺

```
DIV10:
SBC #10  ;
Y=INT(X/10)
INY
BCS DIV10
ADC #10+'0'
CPY #"0"
BEQ ONEDIG
PHA  ; IF Y>0 THEN
TYA  ? Y;
JSR PUTCH
PLA  ;  ? X MOD 10;
```

Who is GoSub?



akka / akka

<> Code

gosubpl

24 commits / 7,081 ++ / 1,060 --

#31

ORMap and friends have deltas (#22350)

master (#22508) v2.5.0-RC2 v2.5.0-RC1

gosubpl committed on 23 Feb

Showing 15 changed files with 4,520 additions and 289 deletions.

Roadmap

- CRDTs
  - Why
  - What / How
  - in akka

Why?

- **Why?**

Why?

- CRDTs

Why?

- CRDTs
  - „C”

Why?

- CRDTs
  - „C”
  - ???

Why?

- CRDTs
  - „C”
  - ???
  - In a minute…

Why?

- **CRDTs**
  - **Replicated**

Why?

- CRDTs
  - Replicated
- Cluster, Replication

Why?

- **CRDTs**
  - **Replicated**
- **Cluster, Replication**
  - **Cluster provides very low level primitives – nodes, communication primitives**

Why?

- CRDTs
  - Replicated
- Cluster, Replication
  - Cluster provides very low level primitives – nodes, communication primitives
  - There is a Replicator in the Cluster

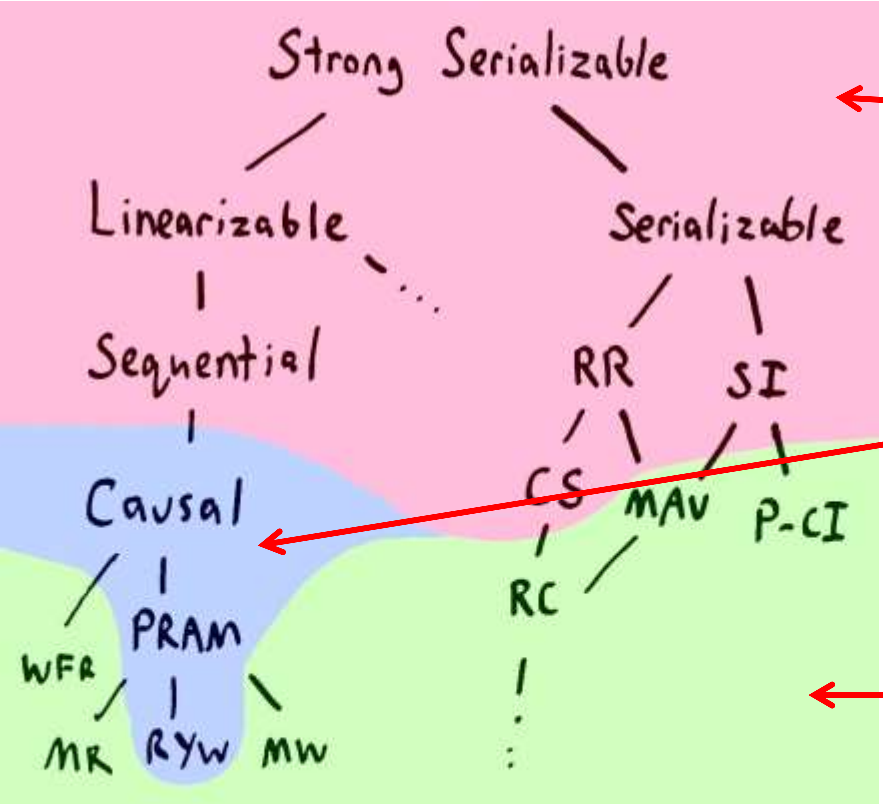
Why?

- CRDTs
  - Data Types

Why?

- CRDTs
  - Data Types
- „Data Types” most obvious here ☺

Why?



Cannot be fully available, global consistency (CP)

HAT – session based consistency, some availability limitations may apply

Cannot be fully consistent (AP)

From Peter Bailis' Highly Available Transactions via https://aphyr.com/posts/313-strong-consistency-models

Why?

- **CRDTs**
  - **Concurrent**
  - **???**

Why?

- CRDTs
  - Concurrent

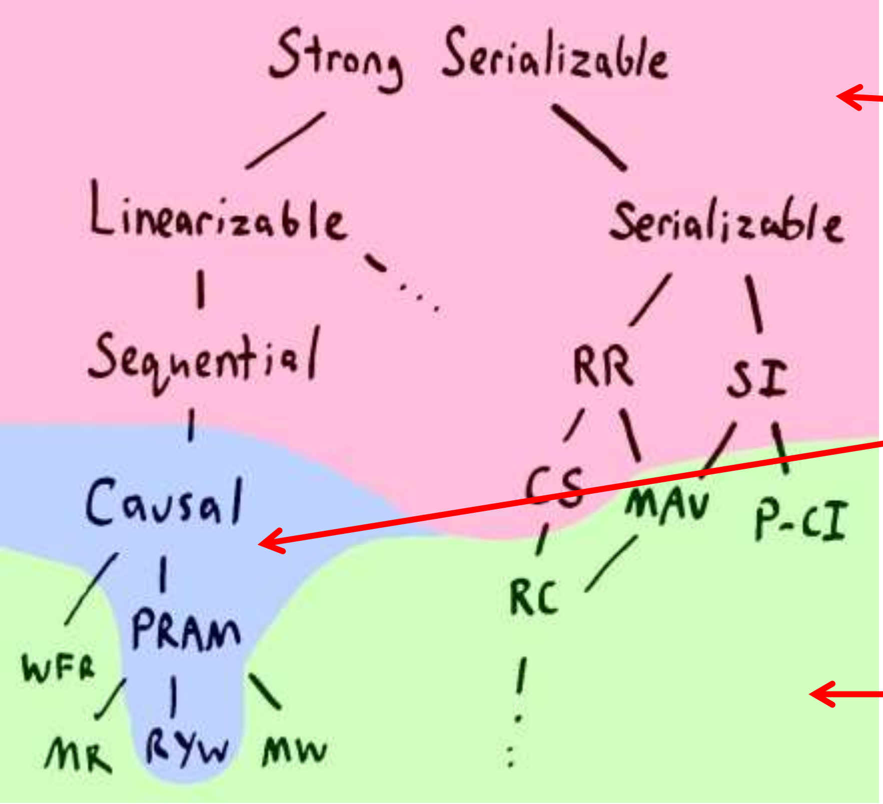
Why?

- **CRDTs**
  - **Concurrent**
    - **Allow concurrent updates on Cluster nodes without requiring synchronisation (AP)**

Why?

- CRDTs
  - Concurrent
    - High Availability
    - High Throughput
    - Great for Shopping Cart!

Why?

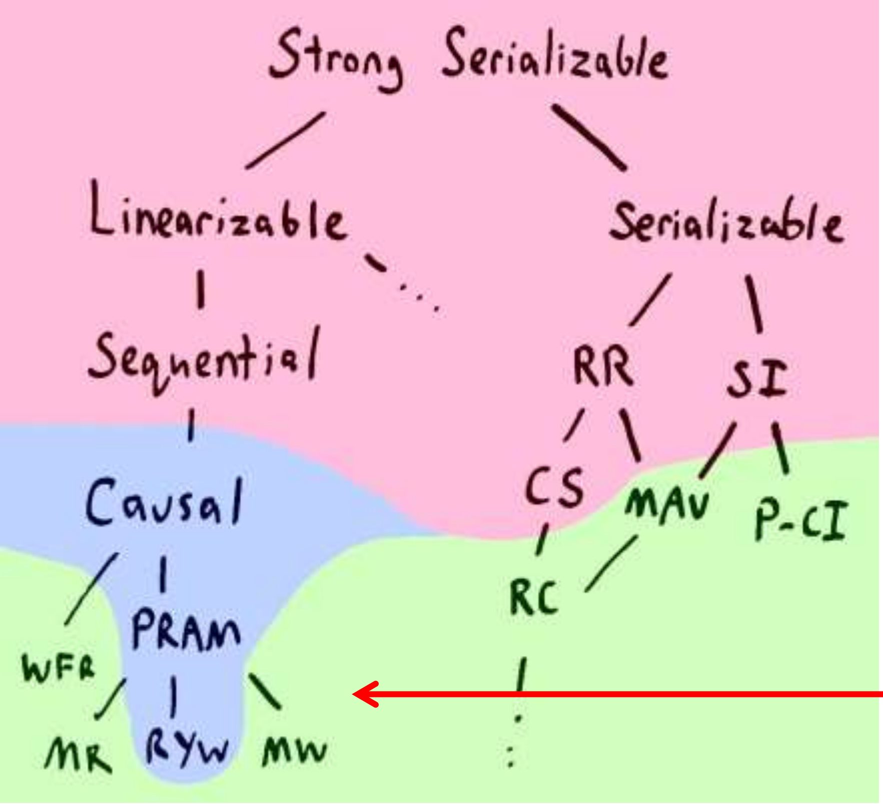


Good Ol’ SQL – „ACID” ☺
CRDTs - NO!

CRDTs ?

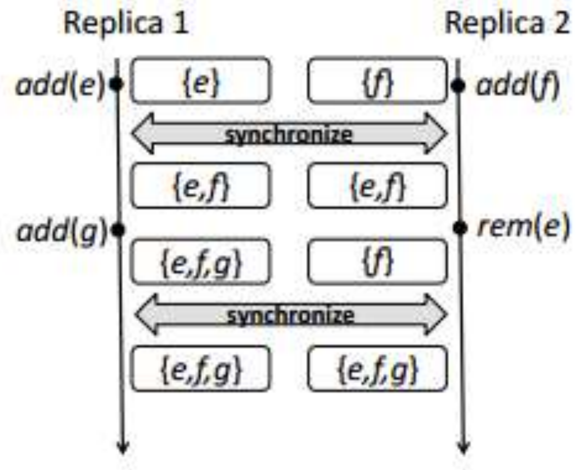
NoSQL! CRDTs ?

Why?



Anomalies! We don’t want to be here!

Why?

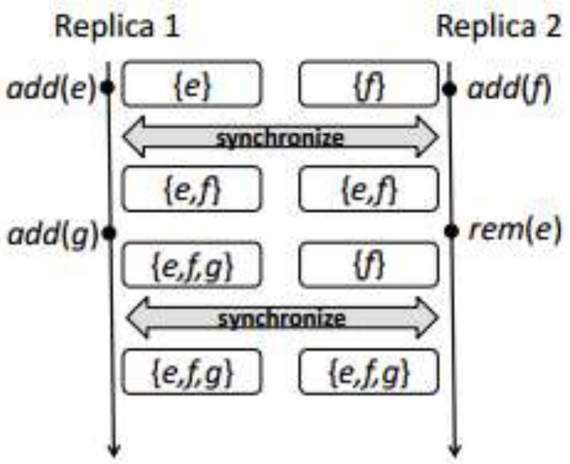


(a) Dynamo shopping cart

An optimized conflict-free replicated set
Bieniusa A., et al.

https://arxiv.org/abs/1210.3368v1

Why?

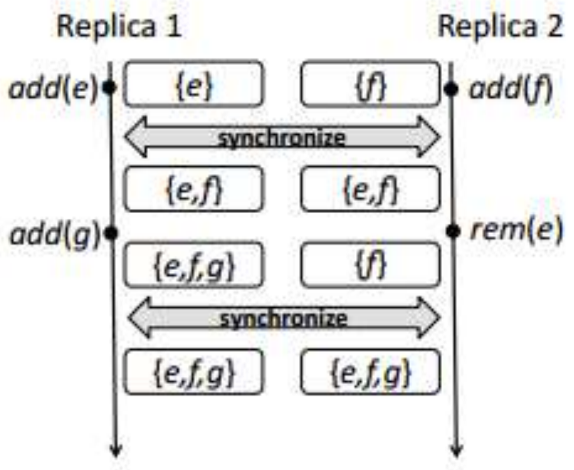


(a) Dynamo shopping cart

Shopping cart service: 99.94% of requests saw 1 version

http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf

Why?

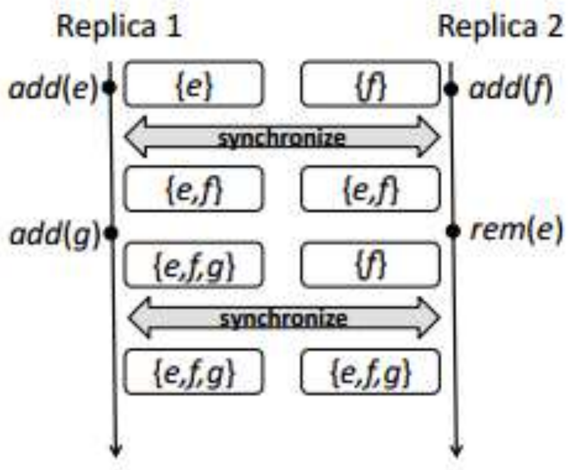


(a) Dynamo shopping cart

Shopping cart service: 99.94% of requests saw 1 version

0.06% saw 2 versions…

Why?

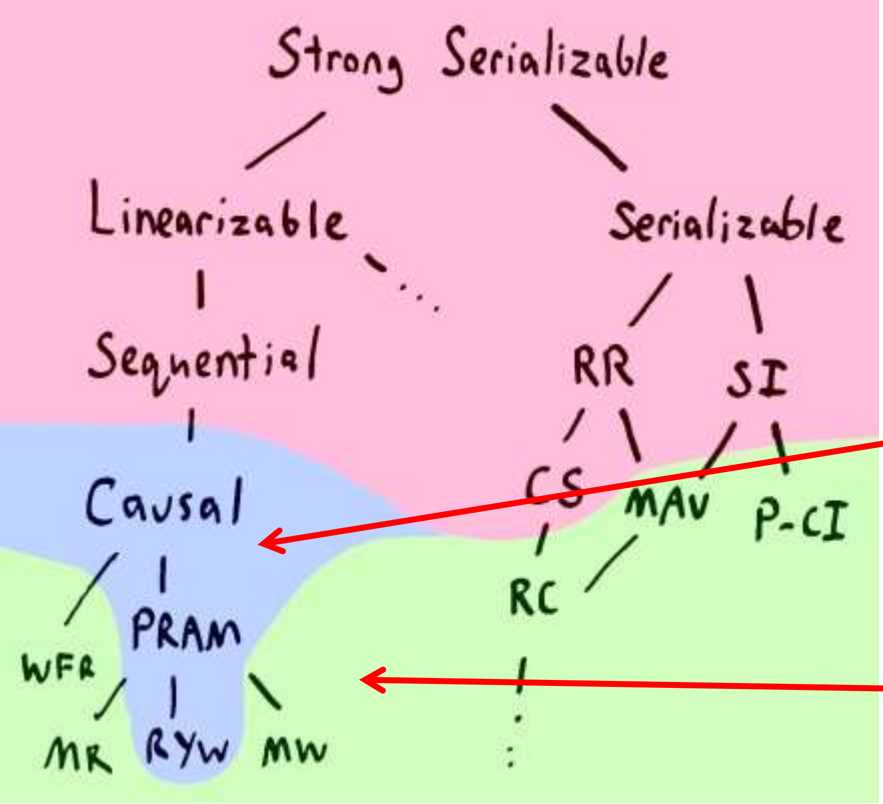


(a) Dynamo shopping cart

Shopping cart service: 99.94% of requests saw 1 version

That makes 6 out of 10 thousand cases where we ship a book that hasn't been paid for…

Why?



CRDTs!

Anomalies! We don’t want to be here!

Why?

- CRDTs
  - Convergent
  - Conflict-free

Why?

- CRDTs
  - Convergent
    - Any two replicas will converge to the same state after updates have stopped

Why?

- CRDTs
  - Conflict-free
    - State change is incremental, no historical state is changed – a (potential) conflict once resolved stays resolved

What / How?

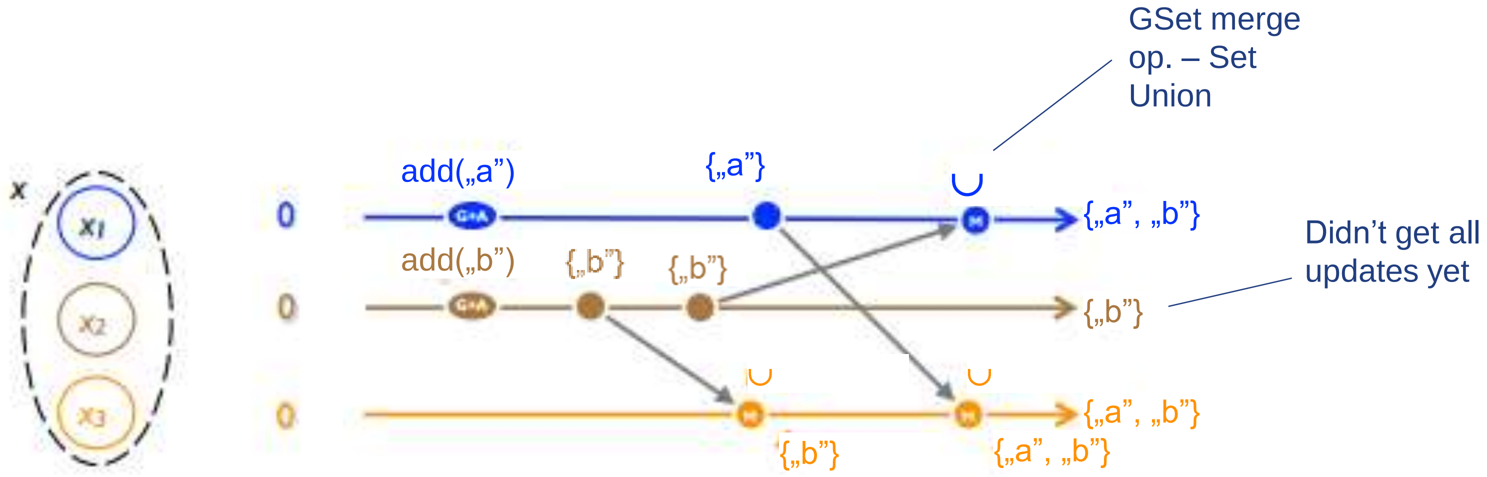
- **What/How?**

What/How?

- GSet – Grow-only Set
  - API – only 2 operations:
    - add(element)
    - elements()
  - No removals!

What/How?

GSet – Timeline



GSet merge op. – Set Union

Didn’t get all updates yet

Based on *A comprehensive study of Convergent and Commutative Replicated Data Types* – Shapiro et al. https://hal.inria.fr/inria-00555588/document

What/How?

- GSet – Grow-only Set
  - API – only 2 operations:
    - add(element)
    - elements()
  - No removals!
  - Update is merged with existing set with standard Set Union operation

What/How?

- GSet – Grow-only Set
  - Update is merged with existing set with standard Set Union operation

What/How?

- GSet – Grow-only Set
  - Update is merged with existing set with standard Set Union operation
  - Wait a second, what's an update?

What/How?

- GSet – Grow-only Set
  - Update is merged with existing set with standard Set Union operation
  - Wait a second, what's an update?
  - Update contains the full state of GSet (all elements!)

What/How?

- GSet – Grow-only Set
  - Update is merged with existing set with standard Set Union operation
  - Wait a second, what's a merge?

What/How?

- GSet – Grow-only Set
  - Update is merged with existing set with standard Set Union operation
  - Wait a second, what's a merge?
  - Merge is Set Union, which is commutative, idempotent and associative.

What/How?

- GSet – Grow-only Set
  - This are the consequences of GSet being a monotonic join semilattice object
  - Which means that its state cannot decrease and merge results in calculating Least Upper Bound of source states

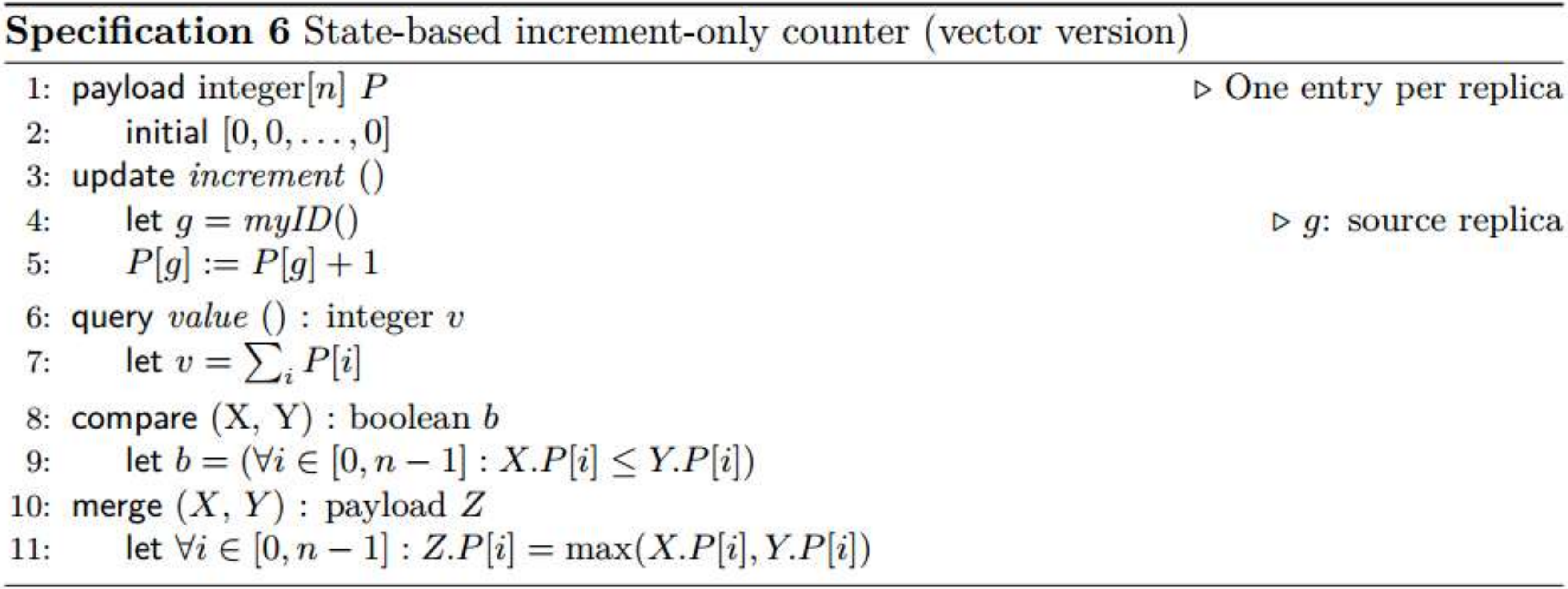
What/How?

- In our case of GSet the Least Upper Bound is the smallest set containing all updates, i.e. it is Set Union of all updates ☺
- Least Upper Bound operation can generaly be shown to be commutative, idempotent and associative

What/How?

- **GCounter – Grow-only Counter**
  - **Two operations:**
    - **Increment(x)**
    - **GetValue**
  - **Merge requires identification of node and doing max over all nodes**

What/How ?

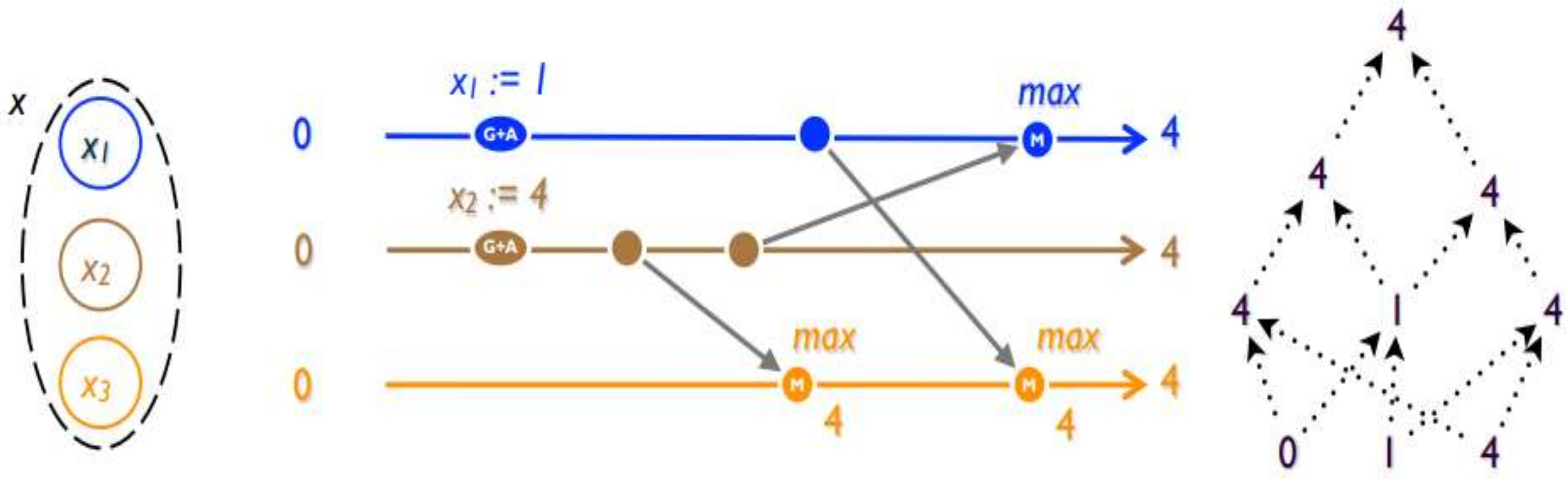


Specification 6 State-based increment-only counter (vector version)

```
payload integer[$n$] $P$                                   ▷ One entry per replica
    initial $[0, 0, \ldots, 0]$
update $increment$ ()
    let $g = myID()$                                       ▷ $g$: source replica
    $P[g] := P[g] + 1$
query $value$ () : integer $v$
    let $v = \sum_i P[i]$
compare (X, Y) : boolean $b$
    let $b = (\forall i \in [0, n-1] : X.P[i] \leq Y.P[i])$
merge $(X, Y)$ : payload $Z$
    let $\forall i \in [0, n-1] : Z.P[i] = \max(X.P[i], Y.P[i])$
```

From *A comprehensive study of Convergent and Commutative Replicated Data Types* – Shapiro et al.
https://hal.inria.fr/inria-00555588/document

What/How ?



From *A comprehensive study of Convergent and Commutative Replicated Data Types* – Shapiro et al.
https://hal.inria.fr/inria-00555588/document

What/How?

- PNCounter – Positive-Negative Counter
  - Three operations:
    - Increment(x)
    - Decrement(x)
    - GetValue

What/How?

- **PNCounter – Positive-Negative Counter**
  - **Cannot have global condition (e.g. PNCounter > 0)**
  - **Simple case – add 5 on one node, subtract (4 or 5) on another**

What/How?

- **PNCounter – Positive-Negative Counter**
  - **We can prevent decreasing if state on current node <= 0**
  - **But that means I cannot decrease even though I (globally) know that this should be posible**

What/How?

- **PNCounter – Positive-Negative Counter**
  - **One solution – system of „credits" / escrow transactions, etc.**

What/How?

- **PNCounter – Positive-Negative Counter**
  - **Another clever solution: When decrement is not possible because violating > 0 on local node, synchronise the cluster; but only then!**

What/How?

- **ORSet – Observed-Remove Set**
  - **Set that allows for removal of elements that have been observed at the current node**
  - **Given causal support (version vectors) for merge operation and conflict resolution protocol that prefers additions over removals this is a CRDT**

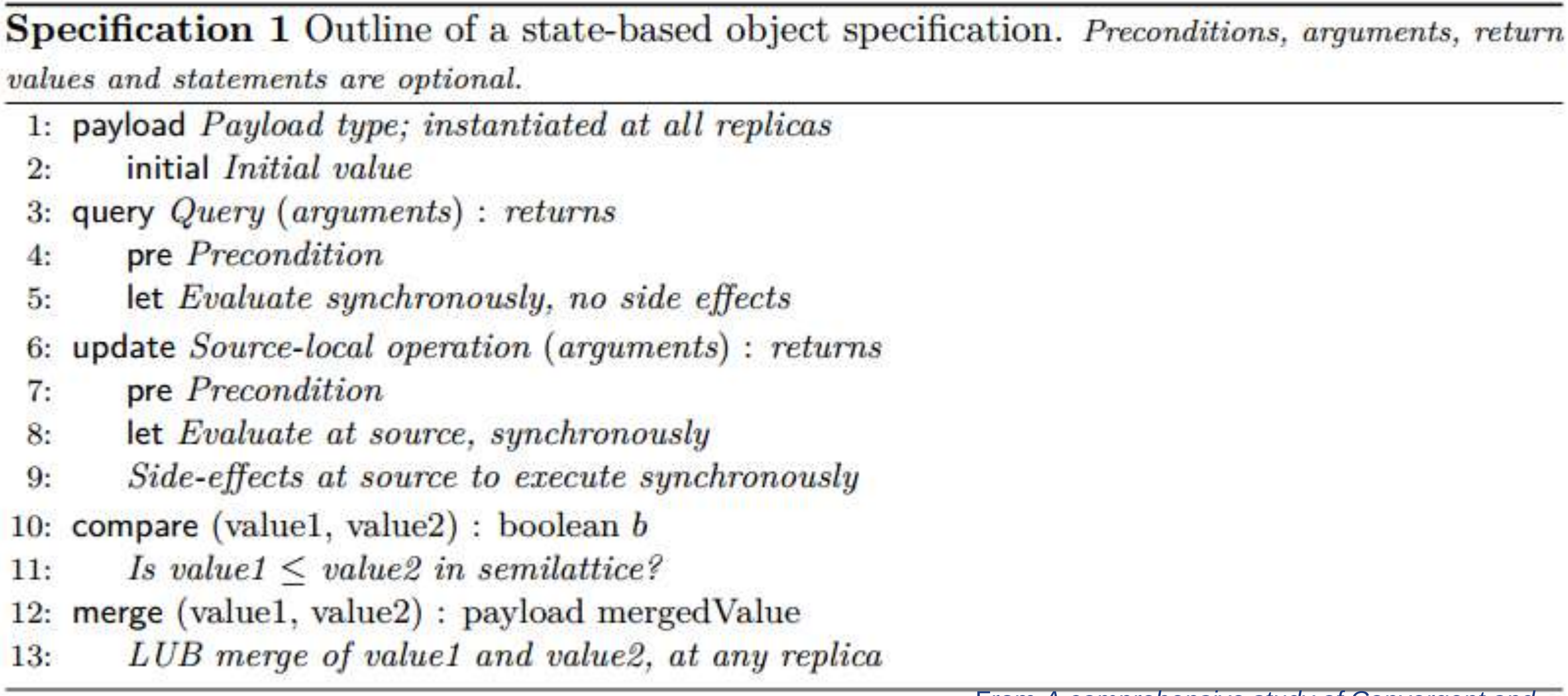
What/How?

- **ORSet – Details beyond capacity of this presentation**
  - **Read:** https://arxiv.org/pdf/1210.3368v1.pdf

An Optimized Conflict-free Replicated Set

Bieniusa, A. et al.

What/How?

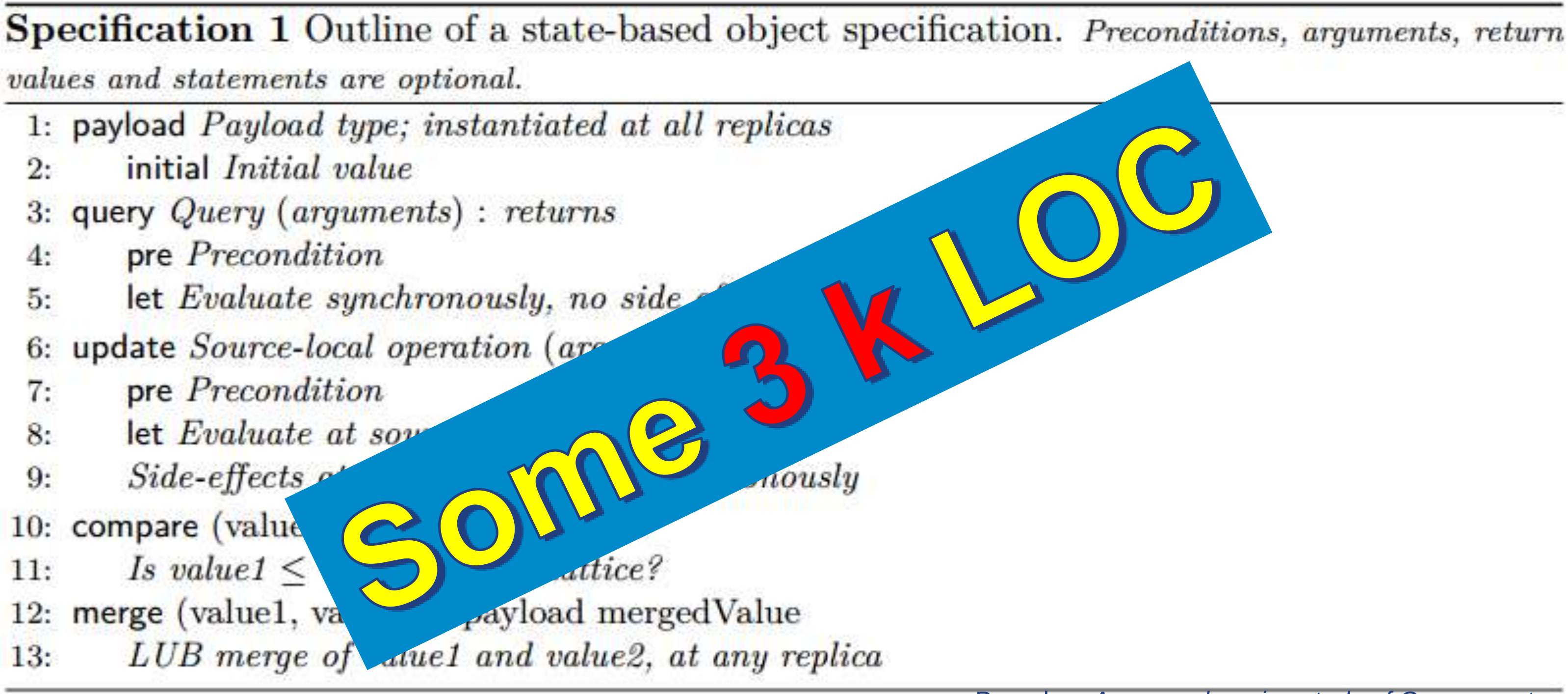


Specification 1 Outline of a state-based object specification. *Preconditions, arguments, return values and statements are optional.*

```
payload Payload type; instantiated at all replicas
    initial Initial value
query Query (arguments) : returns
    pre Precondition
    let Evaluate synchronously, no side effects
update Source-local operation (arguments) : returns
    pre Precondition
    let Evaluate at source, synchronously
    Side-effects at source to execute synchronously
compare (value1, value2) : boolean b
    Is value1 $\leq$ value2 in semilattice?
merge (value1, value2) : payload mergedValue
    LUB merge of value1 and value2, at any replica
```

From *A comprehensive study of Convergent and Commutative Replicated Data Types* – Shapiro et al.
https://hal.inria.fr/inria-00555588/document

What/How?



Specification 1 Outline of a state-based object specification. *Preconditions, arguments, return values and statements are optional.*

```
payload Payload type; instantiated at all replicas
    initial Initial value
query Query (arguments) : returns
    pre Precondition
    let Evaluate synchronously, no side
update Source-local operation (ar
    pre Precondition
    let Evaluate at sou
    Side-effects a                    nously
compare (value
    Is value1 $\leq$              attice?
merge (value1, va         payload mergedValue
    LUB merge of    lue1 and value2, at any replica
```

Some 3 k LOC

Based on *A comprehensive study of Convergent and Commutative Replicated Data Types* – Shapiro et al. https://hal.inria.fr/inria-00555588/document

In Akka

- **You need top-level key to be able to access your objects:**

```
val DataKey = ORSetKey[String]("key")
```

In Akka

- **Get**
- **To retrieve the current value of a data you send Replicator.Get message to the Replicator. You supply a consistency level which has the following meaning:**
  - **ReadLocal** the value will only be read from the local replica
  - **ReadFrom(n)** the value will be read and merged from n replicas, including the local replica
  - **ReadMajority** the value will be read and merged from a majority of replicas, i.e. at least **N/2 + 1** replicas, where N is the number of nodes in the cluster (or cluster role group)
  - **ReadAll** the value will be read and merged from all nodes in the cluster (or all nodes in the cluster role group)

In Akka

```
val replicator = DistributedData(system).replicator
val Counter1Key = PNCounterKey("counter1")
val Set1Key = GSetKey[String]("set1")
val Set2Key = ORSetKey[String]("set2")

replicator ! Get(Counter1Key, ReadLocal)

val readFrom3 = ReadFrom(n = 3, timeout = 1.second)
replicator ! Get(Set1Key, readFrom3)

val readMajority = ReadMajority(timeout = 5.seconds)
replicator ! Get(Set2Key, readMajority)
```

In Akka

```
case g @ GetSuccess(Set1Key, req) =>
        val elements = g.get(Set1Key).elements
case GetFailure(Set1Key, req) =>
        // read from 3 nodes failed within 1.second
case NotFound(Set1Key, req) => // key set1 does not exist
```

In Akka

- **Update**
- **To modify and replicate a data value you send a Replicator.Update message to the local Replicator.**
- You supply a write consistency level which has the following meaning:
  - **WriteLocal** the value will immediately only be written to the local replica, and later disseminated with gossip
  - **WriteTo(n)** the value will immediately be written to at least n replicas, including the local replica
  - **WriteMajority** the value will immediately be written to a majority of replicas, i.e. at least **N/2 + 1** replicas, where N is the number of nodes in the cluster (or cluster role group)
  - **WriteAll** the value will immediately be written to all nodes in the cluster (or all nodes in the cluster role group)

In Akka

```
val Counter1Key = PNCounterKey("counter1")
val Set1Key = GSetKey[String]("set1")
val Set2Key = ORSetKey[String]("set2")

replicator ! Update(Counter1Key, PNCounter(), WriteLocal)(_ + 1)

val writeTo3 = WriteTo(n = 3, timeout = 1.second)
replicator ! Update(Set1Key, GSet.empty[String], writeTo3)(_ + "hello")

val writeMajority = WriteMajority(timeout = 5.seconds)
replicator ! Update(Set2Key, ORSet.empty[String], writeMajority)(_ + "hello")
```

In Akka – Removing From Shopping Cart

```
def receiveRemoveItem: Receive = {
case cmd @ RemoveItem(productId) =>
        // Try to fetch latest from a majority of nodes first, since ORMap
        // remove must have seen the item to be able to remove it.
        replicator ! Get(DataKey, readMajority, Some(cmd))

case GetSuccess(DataKey, Some(RemoveItem(productId))) =>
        replicator ! Update(DataKey, LWWMap(), writeMajority, None) {
        _ - productId
}
case NotFound(DataKey, Some(RemoveItem(productId))) =>
// nothing to remove
}
```

In Akka – GSet

```
val s0 = GSet.empty[String]
val s1 = s0 + "a"
val s2 = s1 + "b" + "c"
if (s2.contains("a"))
          println(s2.elements) // a, b, c
```

In Akka – Gcounter/PNCounter

```
implicit val node = Cluster(system)
val c0 = PNCounter.empty
val c1 = c0 + 1
val c2 = c1 + 7
val c3: PNCounter = c2 - 2
println(c3.value) // 6
```

Note – needs implicit node!

In Akka – ORSet

```
implicit val node = Cluster(system)
val s0 = ORSet.empty[String]
val s1 = s0 + "a"
val s2 = s1 + "b"
val s3 = s2 - "a"
println(s3.elements) // b
```

In Akka – ORMultiMap

```
implicit val node = Cluster(system)
val m0 = ORMultiMap.empty[String, Int]
val m1 = m0 + ("a" -> Set(1, 2, 3))
val m2 = m1.addBinding("a", 4)
val m3 = m2.removeBinding("a", 2)
val m4 = m3.addBinding("b", 1)
println(m4.entries)
```

Akka – docs!

- Docs:
  - http://doc.akka.io/docs/akka/2.5/scala/distributed-data.html
- Interesting Specs from akka/akka
  - https://github.com/akka/akka/tree/master/akka-distributed-data/src/multi-jvm/scala/akka/cluster/ddata
- Interesting Implementations from akka/akka-samples (incl. ShoppingCart)
  - https://github.com/akka/akka-samples/tree/master/akka-sample-distributed-data-scala/src/main/scala/sample/distributeddata
- Intersting Specs from akka/akka-samples
  - https://github.com/akka/akka-samples/tree/master/akka-sample-distributed-data-scala/src/multi-jvm/scala/sample/distributeddata

Shopping Cart, ftw!

Demo //FIXME

That’s it…

Thank you! Questions? Answers?

https://en.wikipedia.org/wiki/Banner_Mania

私はカピバスも好きです