

REPLICATED DATA, FROM PRACTICE TO THEORY

Madhavan Mukund

Chennai Mathematical Institute and UMI RELAX, Chennai, India

Joint work with Gautham Shenoy R, S P Suresh

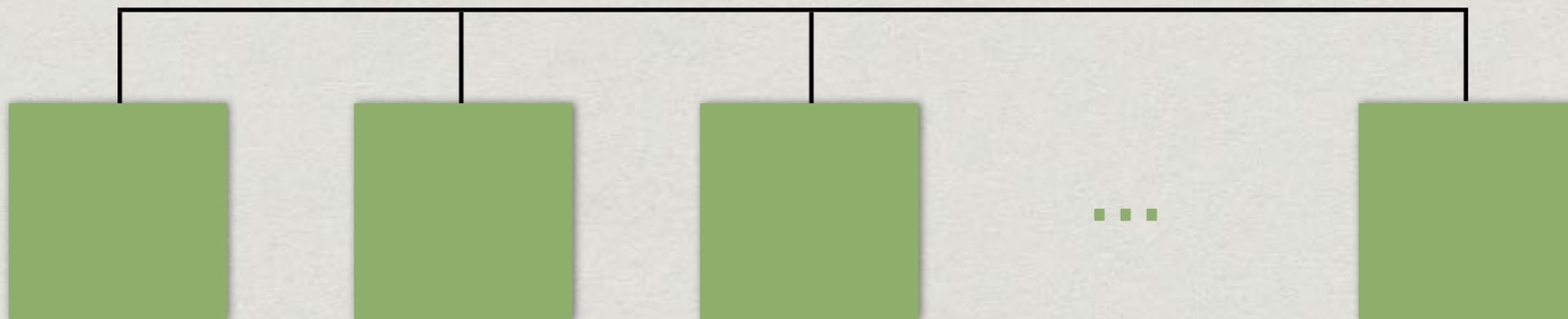
LSV 20th Anniversary Celebrations,

LSV, ENS Paris-Saclay, 10-11 May 2017

Distributed systems

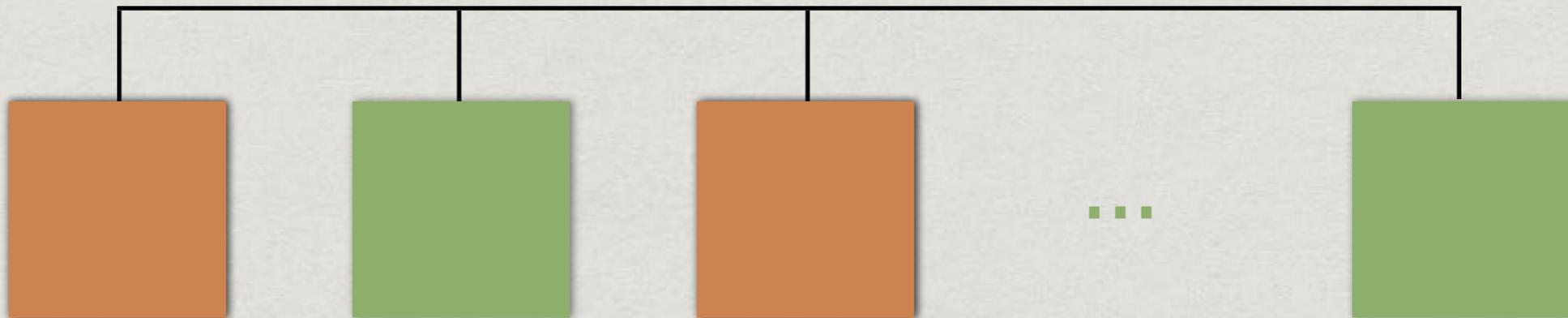
Distributed systems

- * N nodes connected by asynchronous network



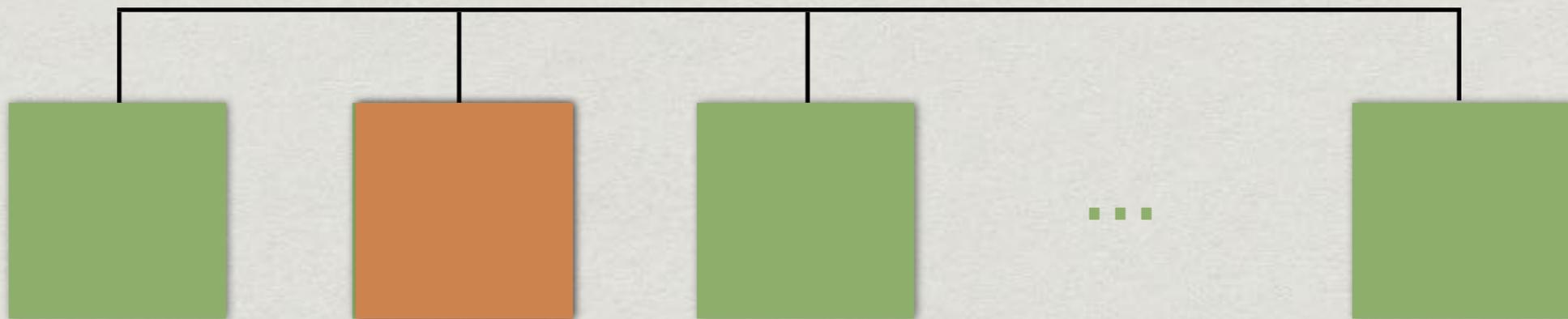
Distributed systems

- * N nodes connected by asynchronous network
- * Nodes may fail and recover infinitely often



Distributed systems

- * N nodes connected by asynchronous network
- * Nodes may fail and recover infinitely often
- * Nodes resume from safe state before failure



Replicated datatypes



Replicated datatypes

- * Each node replicates the data structure



Replicated datatypes

- * Each node replicates the data structure
- * Queries / updates addressed to any replica
 - * Queries are side-effect free
 - * Updates change the state of the data structure



Replicated datatypes ...

- * Typical applications
 - * Amazon shopping carts
 - * Google docs
 - * Facebook “like” counters



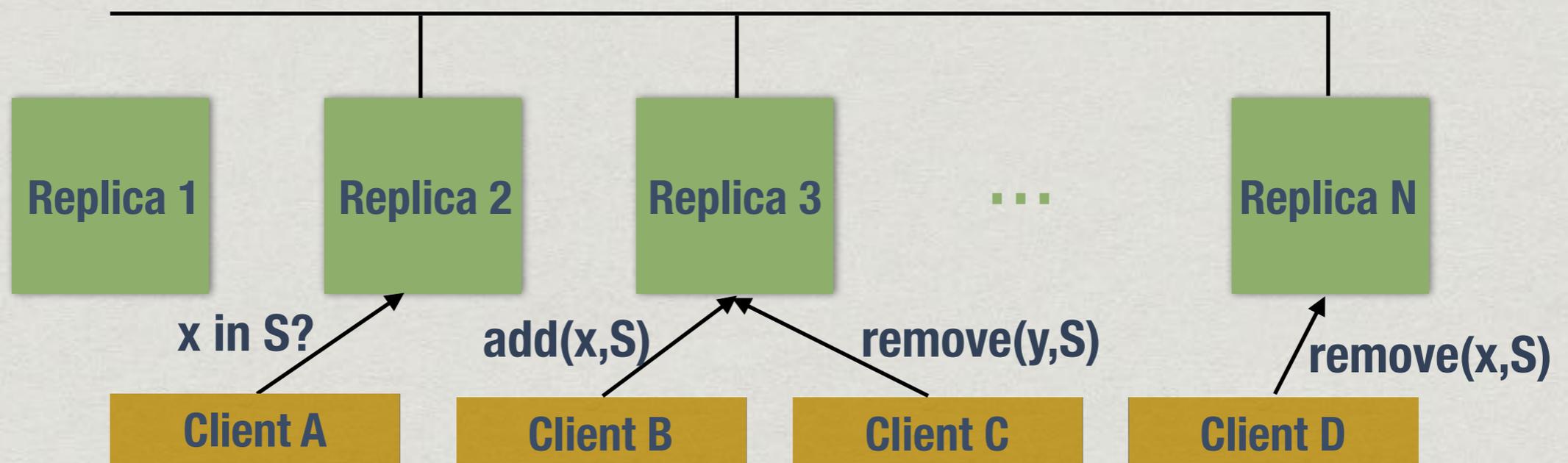
Replicated datatypes ...

- * Typical data structure — Sets
 - * Query : is x a member of S ?
 - * Updates : add x to S , remove x from S



Clients and replicas

- * Clients issue query/update requests
- * Each request is fielded by an individual **source** replica

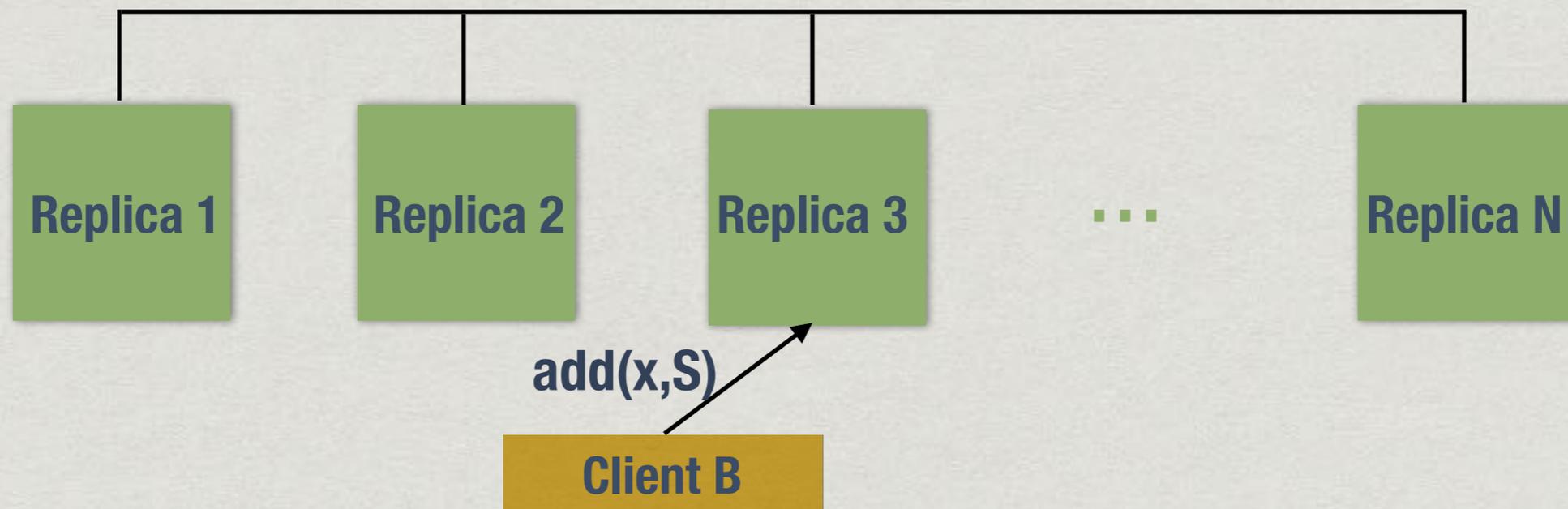


Processing query requests

- * Queries are answered directly by source replica, using local state

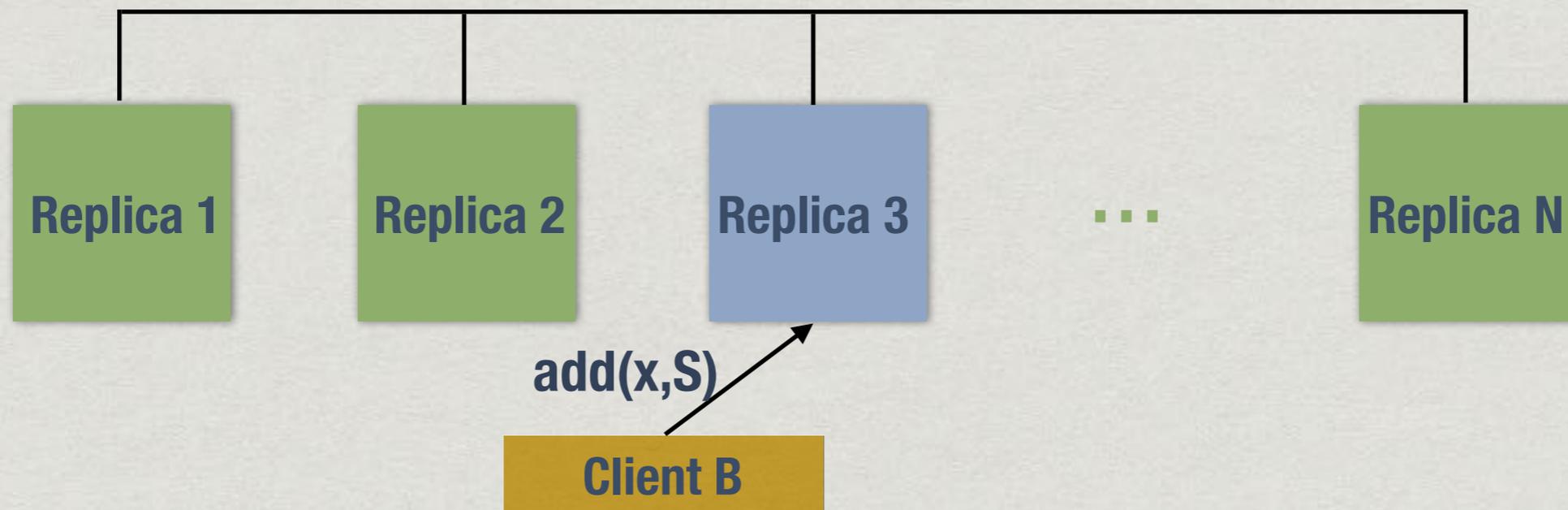


Processing updates



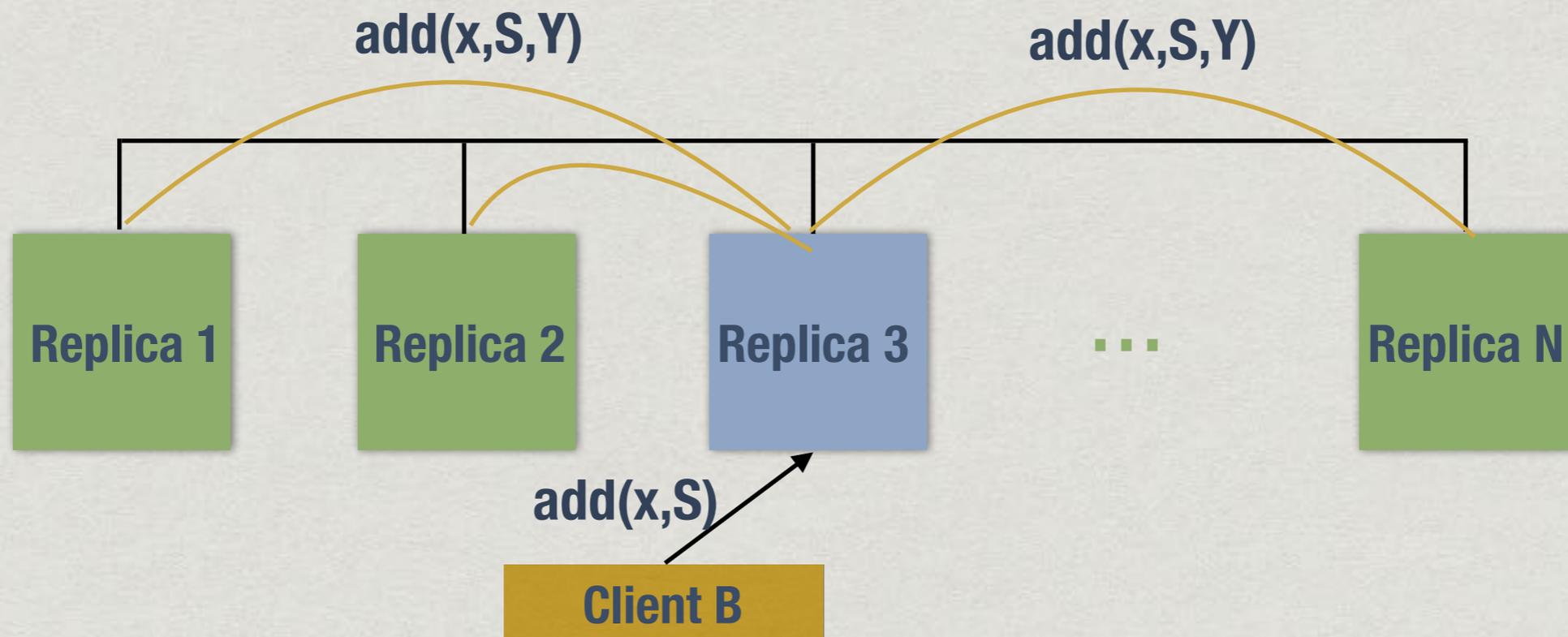
Processing updates

- * Source replica first updates its own state



Processing updates

- * Source replica first updates its own state
- * Propagates update message to other replicas
- * With auxiliary metadata (timestamps etc)



Consistency

- * Queries answered by replicas based on local data
- * What guarantees can we provide about consistency?

The CAP bottleneck

- * **Consistency** All the nodes see the same data at the same time
- * **Availability** Every request receives a status response—success or failure
- * **Partition Tolerance** Resilient to (transient or permanent) failures in connectivity
- * **CAP Theorem [Brewer 2000, Gilbert+Lynch 2002]**
A distributed system can satisfy at most 2 out of 3

The CAP tradeoff

- * Network failures are unavoidable, require partition tolerance
- * Give up consistency or availability
- * Traditional distributed databases: maintain consistency
- * Web services require high availability
- * What is a suitable weakening of consistency?

Strong eventual consistency

- * Replicas may diverge while updates propagate
 - * All messages are reliably delivered
- * Replicas that receive the same set of updates are query equivalent
 - * Concurrent updates need not arrive same order
- * After a period of quiescence, all replicas converge
 - * Does not specify what value they converge to!

Facebook example (2012)

<http://markcathcart.com/2012/03/06/eventually-consistent/>

The image shows a screenshot of a Facebook post and its comments. The post is by Mike Gillespie, asking if anyone else has noticed that the Facebook locator is 'squiffy' (inaccurate) near Inkberrow. The comments are from Mark Cathcart, Mike Gillespie, and Martin Jenkins. The right side of the screenshot shows the right-hand column of the Facebook interface, including a birthday notification for Hugo Garza, a sponsored advertisement for Fab.com outdoor lanterns, and a link to Al Franken's website.

Mike Gillespie Has anyone else noticed that the FB locator is squiffy.... I am no where near Inkberrow....
Like · Comment · Share · 2 hours ago near Inkberrow, England · 🌐

Mark Cathcart are you on a wired network? They get it from the ISP based on the IP address...
2 hours ago · Like

Mike Gillespie I know.... Actually this might interest you... I didn't realise until today that there are actually two sets of recognised gps co ordinates used on the web - OSGB36 and WGS84... and depending on which set you use (given that we use post codes here and zip codes elsewhere) a post code can be as much as 100 metres out.....
about an hour ago · Like

Martin Jenkins and that matters because?
15 minutes ago · Like

Mark Cathcart well its of passing interest because Mike has a business that could benefit from being able to accurately locate properties based on the location of the people looking...
4 minutes ago · Like · 🗨️ 1

Write a comment...

Mike Gillespie commented on his own status: "When you run a business that r..."

Jen Mathe commented on her own status: "Jenni Plane This is probably t..."

Hugo Garza's birthday is today
4 events this week

Sponsored Create an Ad

Outdoor Lanterns
Light up your outdoor space with these wireless lanterns. Join Fab.com and save 25%
159,998 people like Fab.com.

Join Al Franken
alfranken.com
Republicans think your employer should decide what health care you get. Sign here if you think they

Facebook example (2012)

<http://markcathcart.com/2012/03/06/eventually-consistent/>

The image shows a screenshot of a Facebook post and its comments. The post is by Mike Gillespie, asking if anyone else has noticed that the Facebook locator is 'squiffy' (inaccurate) near Inkberrow, England. The comments are as follows:

- Mark Cathcart:** "are you on a wired network? They get it from the ISP based on the IP address..." (2 hours ago, Liked)
- Mike Gillespie:** "I know.... Actually this might interest you... I didn't realise until today that there are actually two sets of recognised gps co ordinates used on the web - OSGB36 and WGS84... and depending on which set you use (given that we use post codes here and zip codes elsewhere) a post code can be as much as 100 metres out....." (about an hour ago, Liked)
- Martin Jenkins:** "and that matters because?" (15 minutes ago, Liked)
- Mark Cathcart:** "well its of passing interest because Mike has a business that could benefit from being able to accurately locate properties based on the location of the people looking..." (4 minutes ago, Liked, 1 reply)

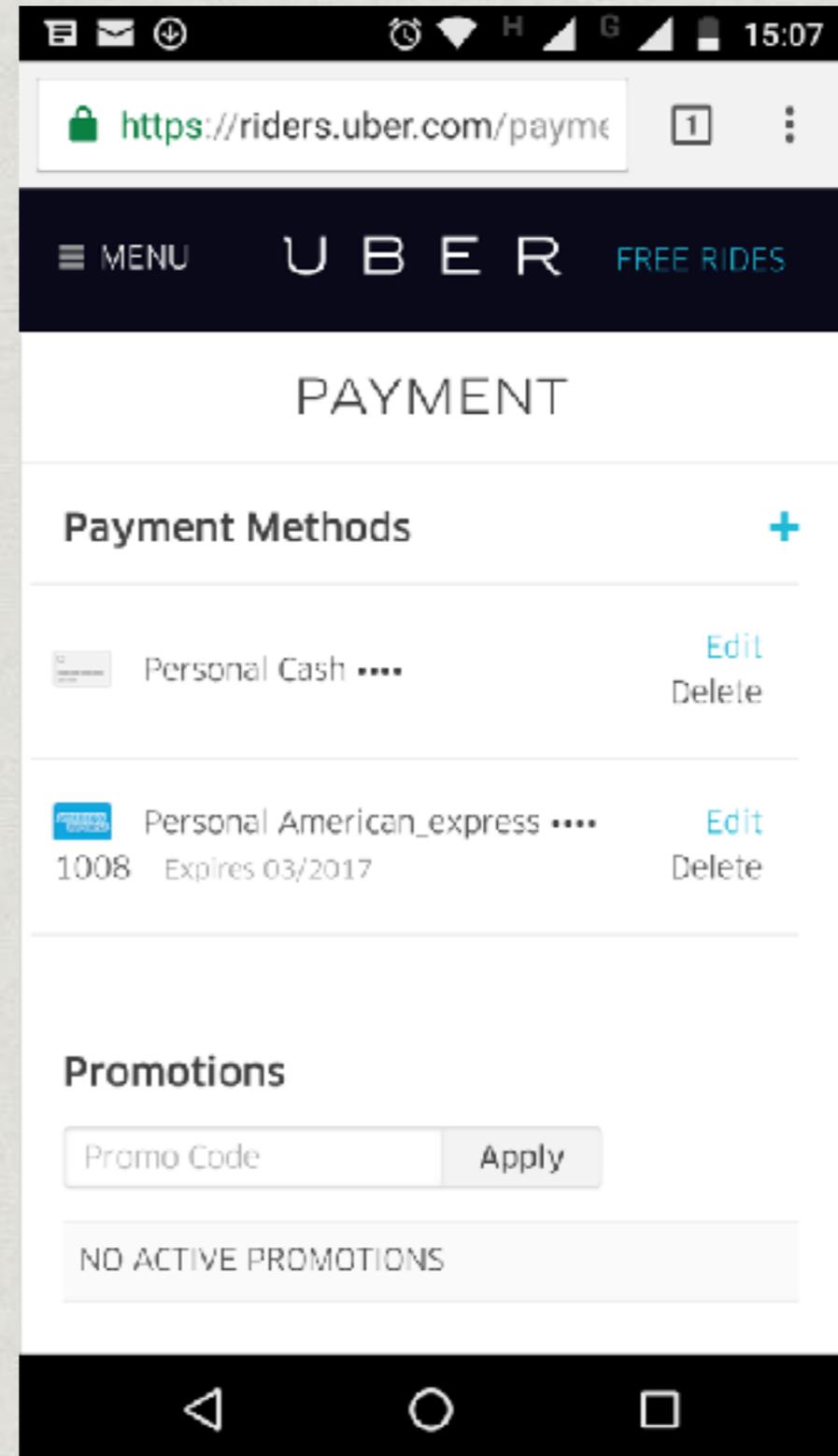
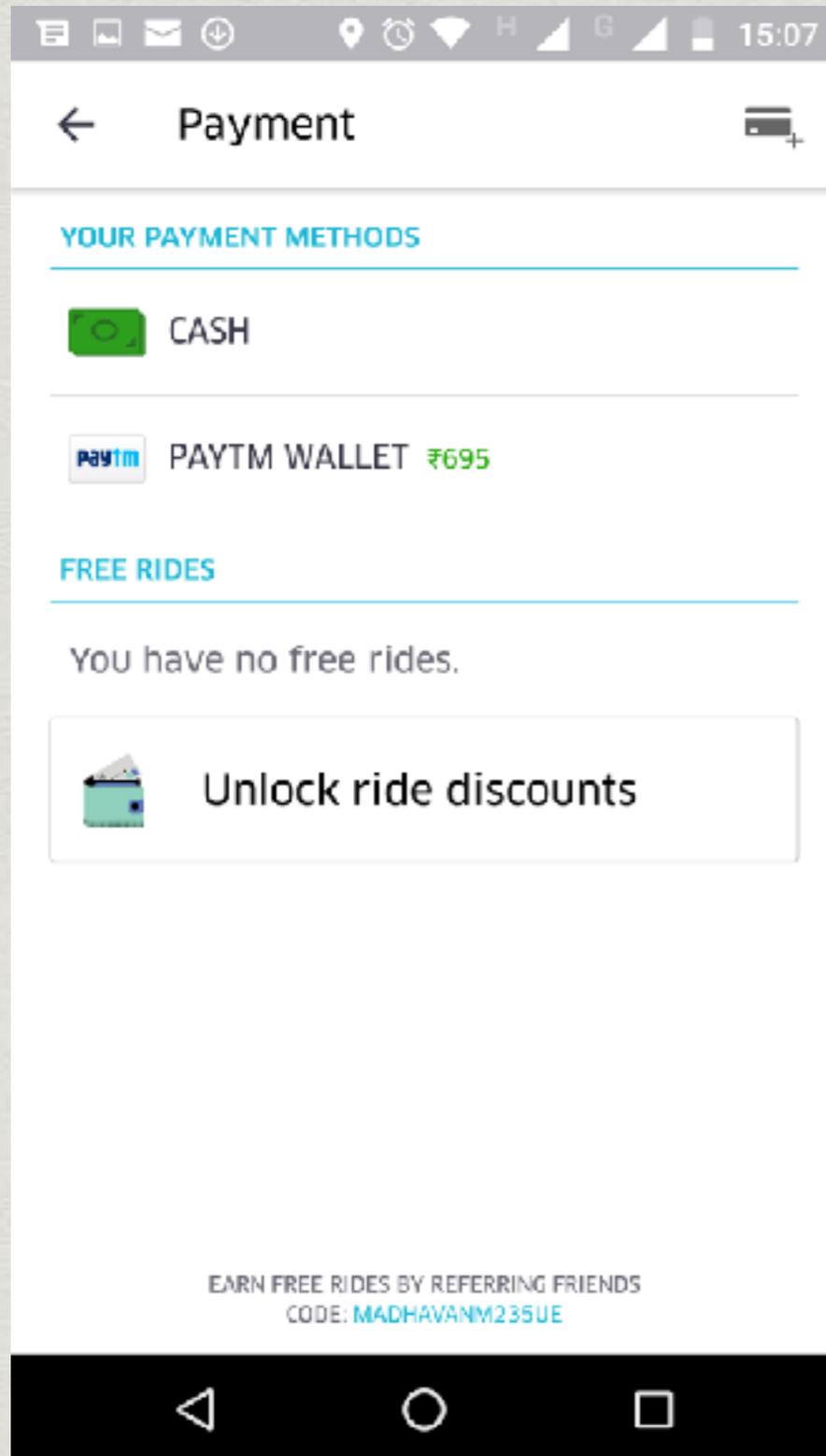
At the bottom of the comments is a text box that says "Write a comment...".

On the right side of the screenshot, there are several other elements:

- A comment by Mike Gillespie on his own status: "When you run a business that r..." (This comment is circled in red).
- A comment by Jen Mathe on her own status: "Jenni Plane This is probably t..."
- A birthday notification for Hugo Garza.
- A notification for 4 events this week.
- A sponsored advertisement for "Outdoor Lanterns" from Fab.com, featuring a photo of colorful lanterns and the text: "Light up your outdoor space with these wireless lanterns. Join Fab.com and save 25%". Below the ad, it says "159,998 people like Fab.com."
- A notification to "Join Al Franken" from alfranken.com, with a photo of Al Franken and the text: "Republicans think your employer should decide what health care you get. Sign here if you think they..."

A red arrow points from the circled comment on the right to the "Write a comment..." box at the bottom of the main post.

Uber: App vs Webpage



CRDT: Conflict Free Data Types

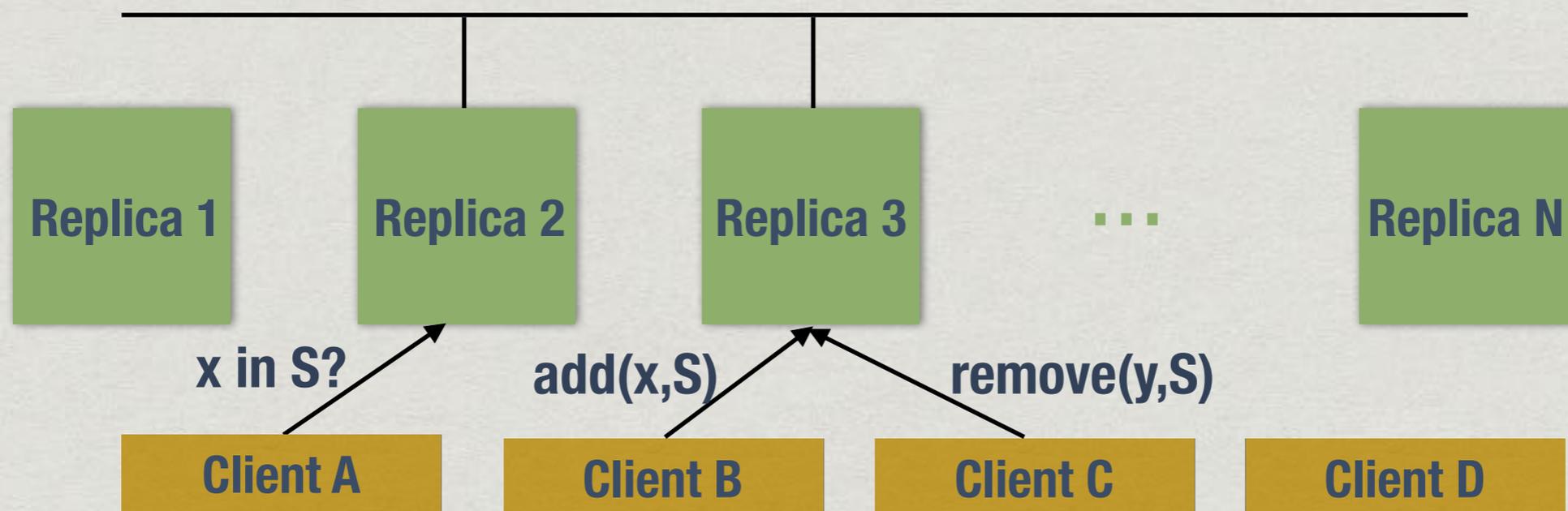
- * Introduced by Shapiro et al 2011
- * Implementations of counters, sets, graphs, ... that satisfy strong eventual consistency by design
- * Maintain auxiliary metadata at each replica
- * Use metadata to reconcile updates
 - * State based: merge entire state
 - * Op based: send summary of operation

A distributed counter

- * N replicas, each maintains a vector C_i of counters
 - * $C_i[k]$ — i 's info about increments received by k
- * Increment at i — update $C_i[i]$
- * Merge j 's info at i : take max of $C_i[k]$ and $C_j[k]$
- * Query counter value at i — sum of all entries in C_i
- * All updates are monotonic

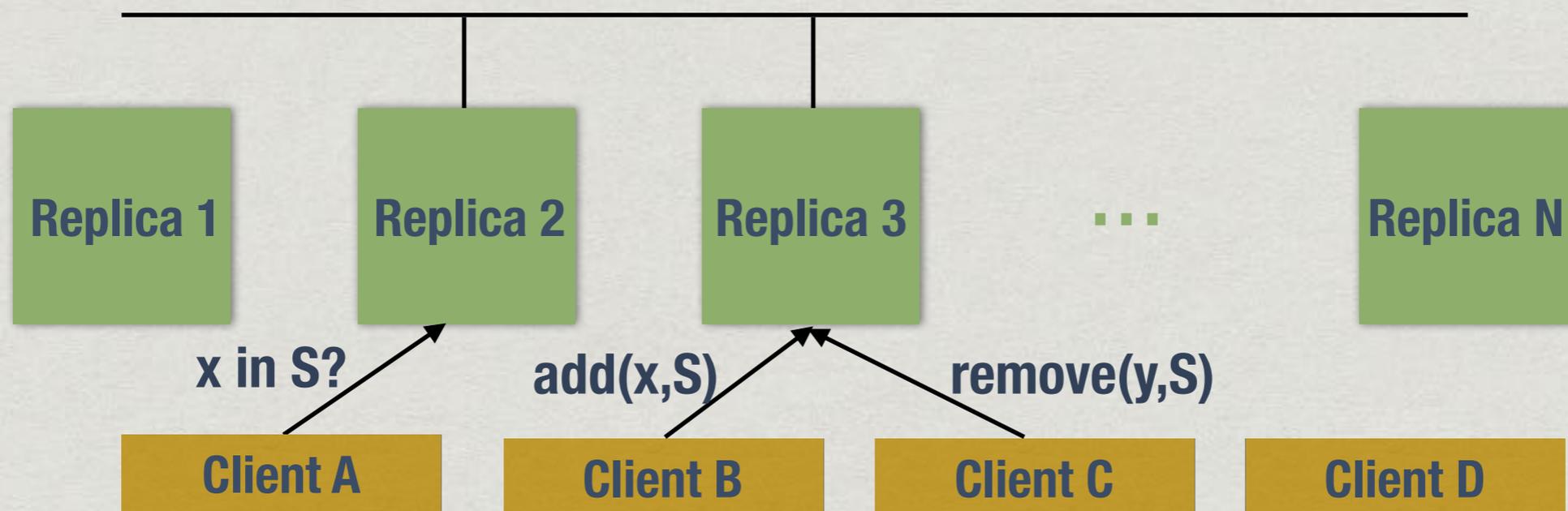
A distributed set

- * $\text{add}(x)$, $\text{remove}(x)$, $\text{member}(x)$
- * $\text{add}(x)$ and $\text{remove}(x)$ are conflicting
- * What is the effect of concurrent operations



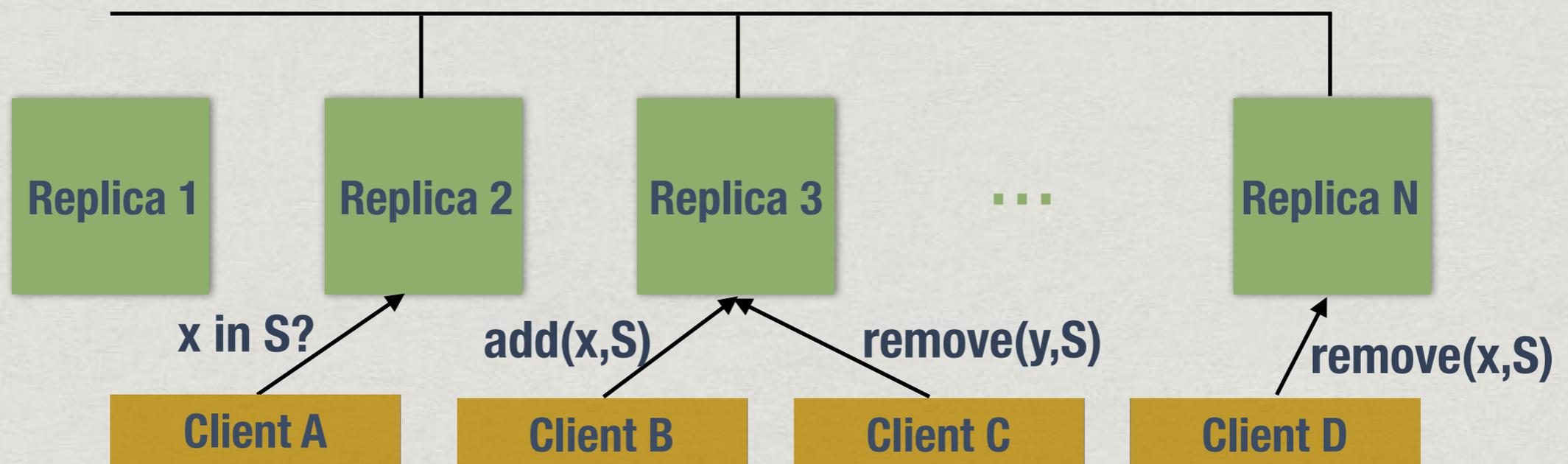
A distributed set

- * Define a conflict resolution policy
 - * Add wins — “Observed-Remove” (OR) set
 - * Question is of consistency, not “correctness”



“Operational” specifications

- * My implementation uses timestamps, ... to detect causality and concurrency
- * If my replica received $\langle \text{add}(x,S), t \rangle$ and $\langle \text{remove}(x,S), t' \rangle$ and t and t' are related by ..., then answer Yes to “ x in S ?”, otherwise No

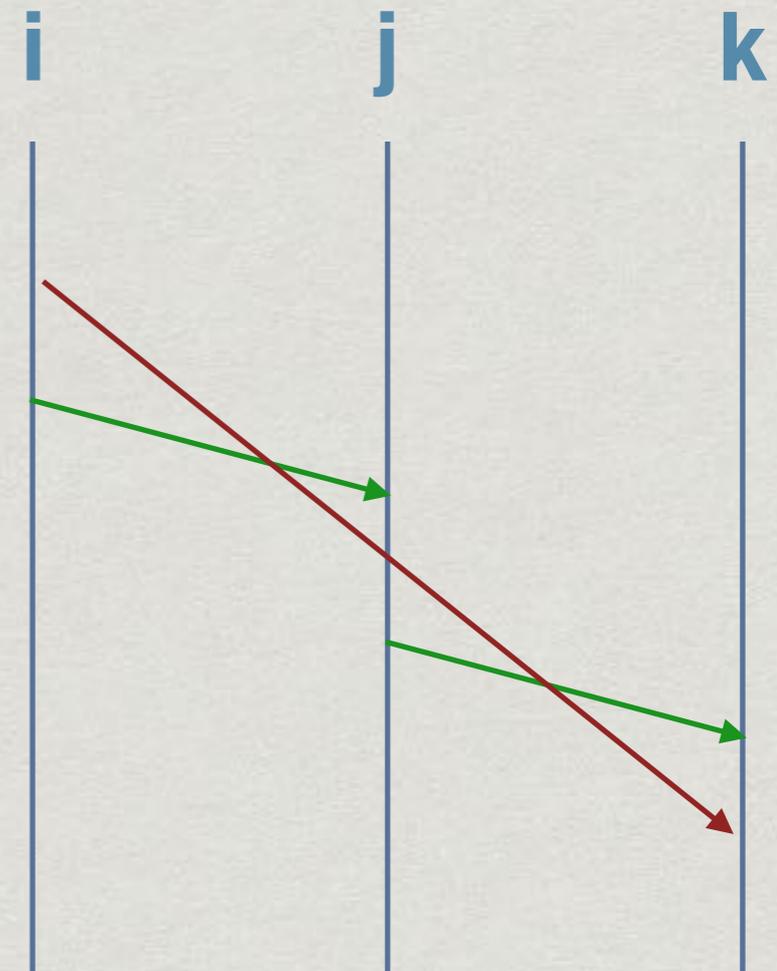


Declarative specification

- * Represent a concurrent computation canonically
 - * Say a labelled partial order
- * Describe effect of a query based on this abstract representation
 - * Reordering of concurrent updates does not matter
 - * Strong eventual consistency is guaranteed

Declarative specification

- * A fully general declarative model is very complicated [Burckhardt et al, POPL 2014]
- * Message delivery relation, visibility relation for updates, ...
- * Simplify — assume causal delivery of messages
- * Can then use partial orders



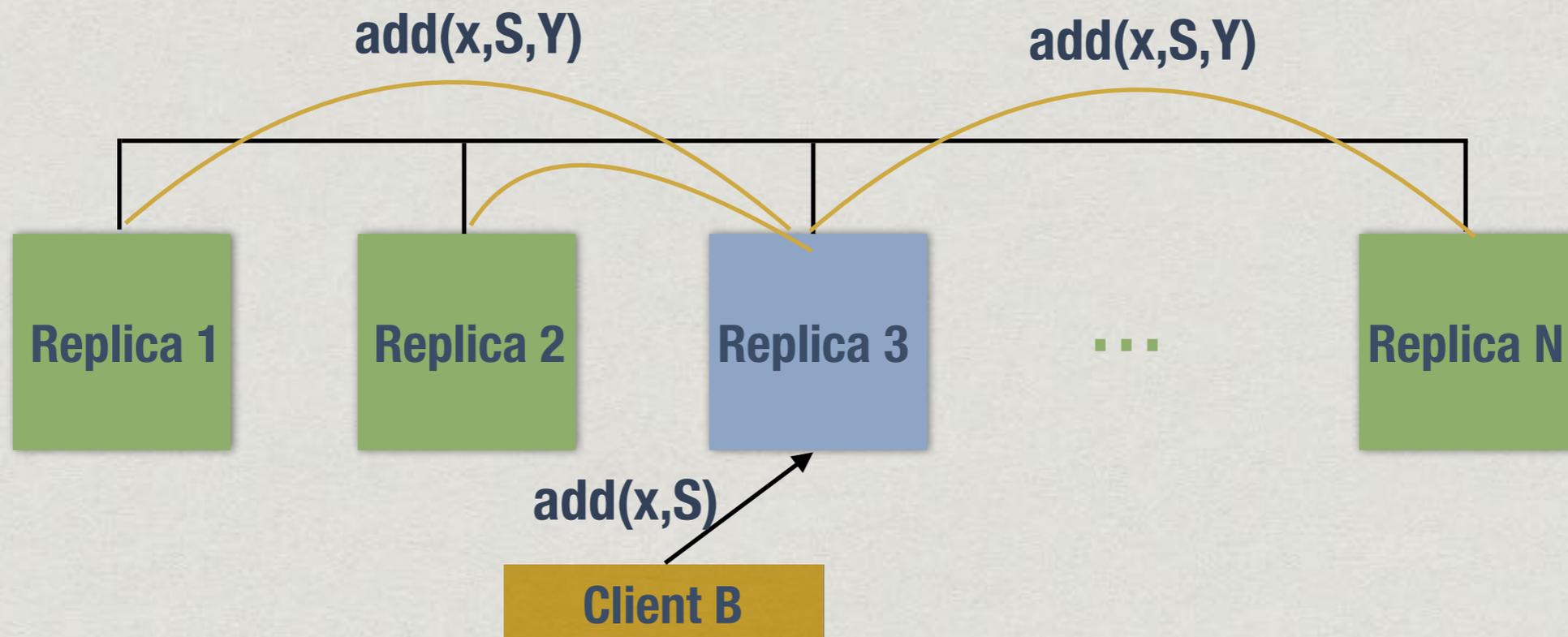
Violation of
causal delivery

CRDTs

- * Conflict-free Replicated Data Type: $D = (V, Q, U)$
 - * V — underlying universe of values
 - * Q — query operations
 - * U — update operations
- * For instance, for OR-sets,
 $Q = \{\text{member-of}\}$, $U = \{\text{add, remove}\}$

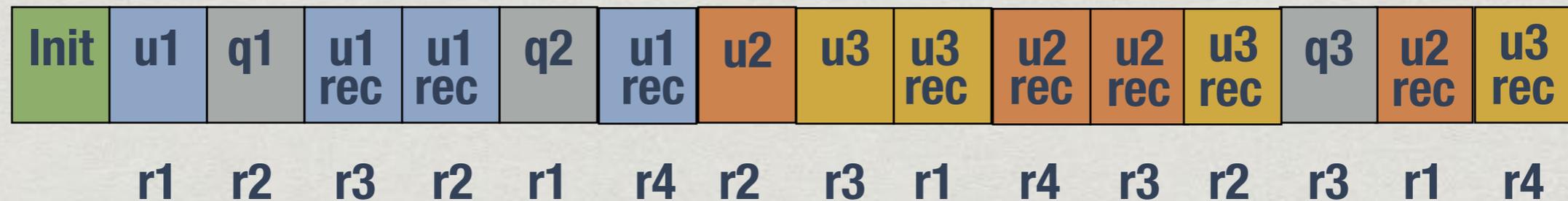
Runs of CRDTs

- * Recall that each update is
 - * locally applied at source replica,
 - * followed by N-1 messages to other replicas



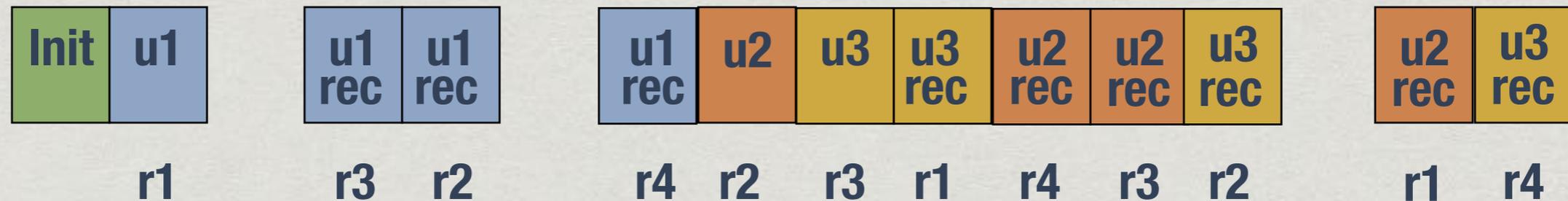
Runs of CRDTs ...

- * Sequence of query, update and receive operations



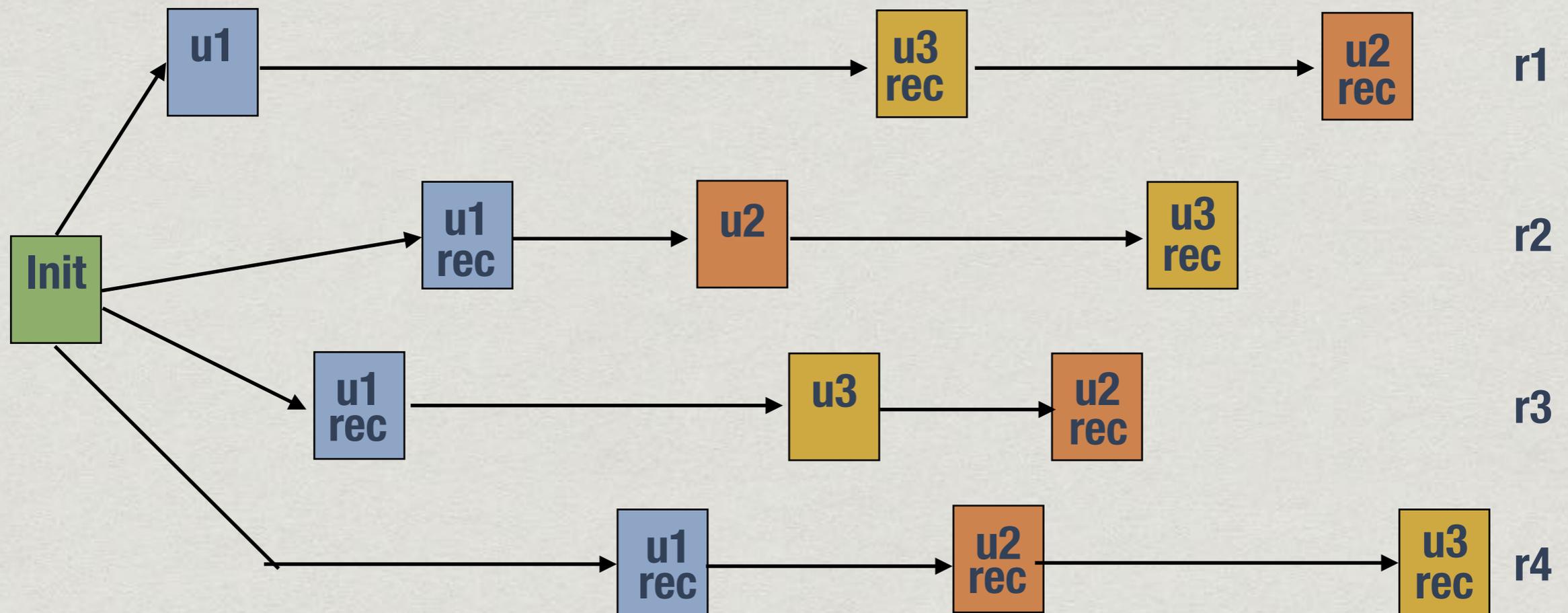
Runs of CRDTs ...

- * Ignore query operations
- * Associate a unique event with each update and receive operation



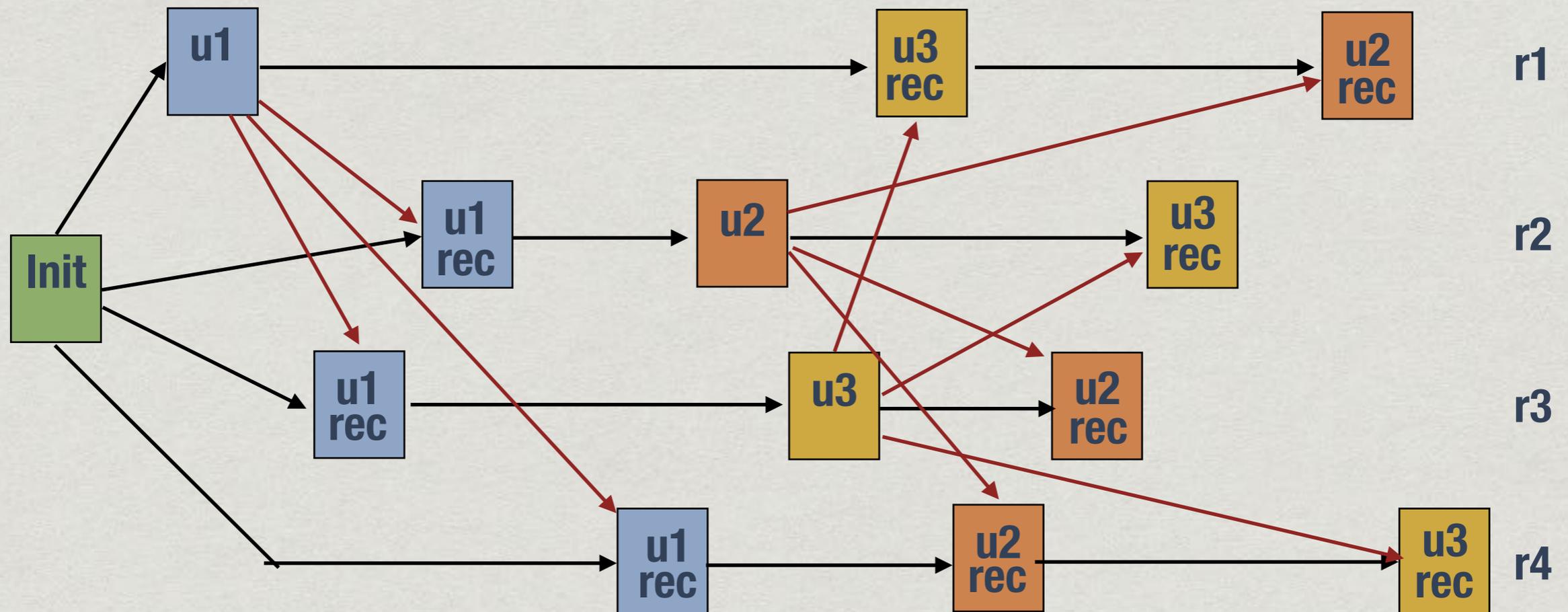
Runs of CRDTs ...

- * Replica order: total order of each replica's events



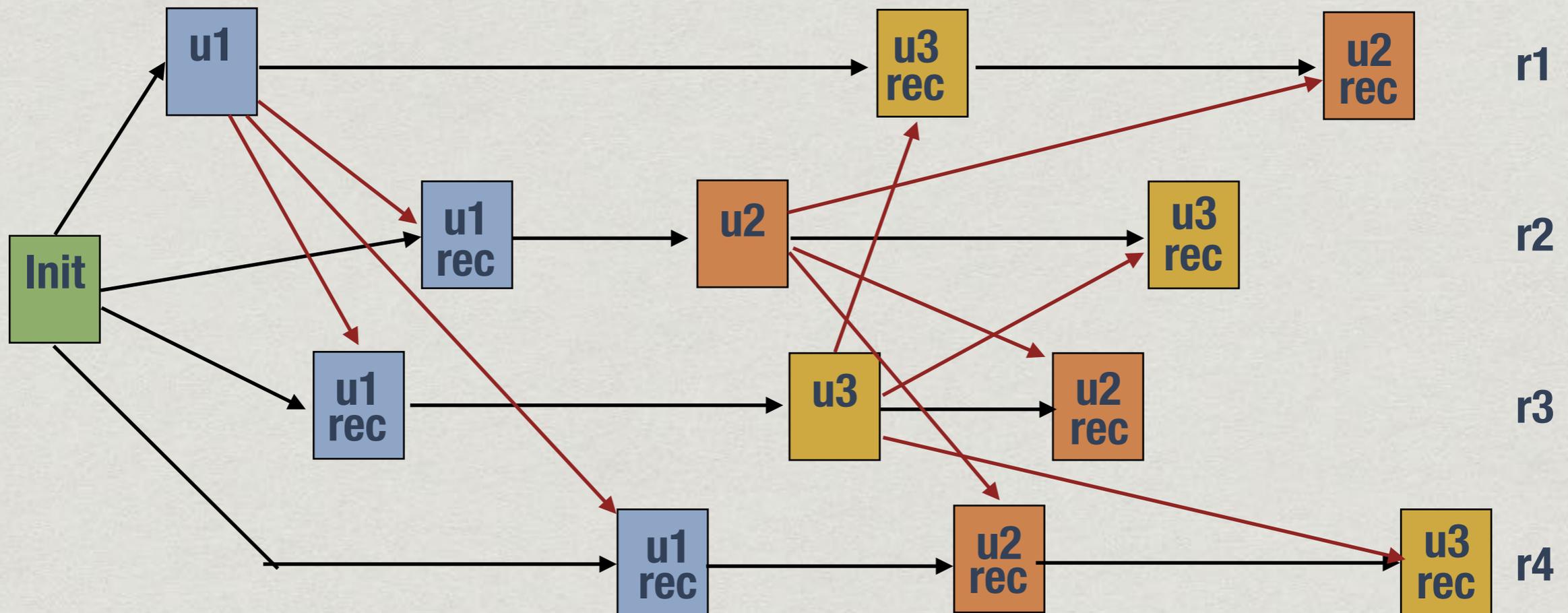
Runs of CRDTs ...

- * Delivery order: match receives to updates



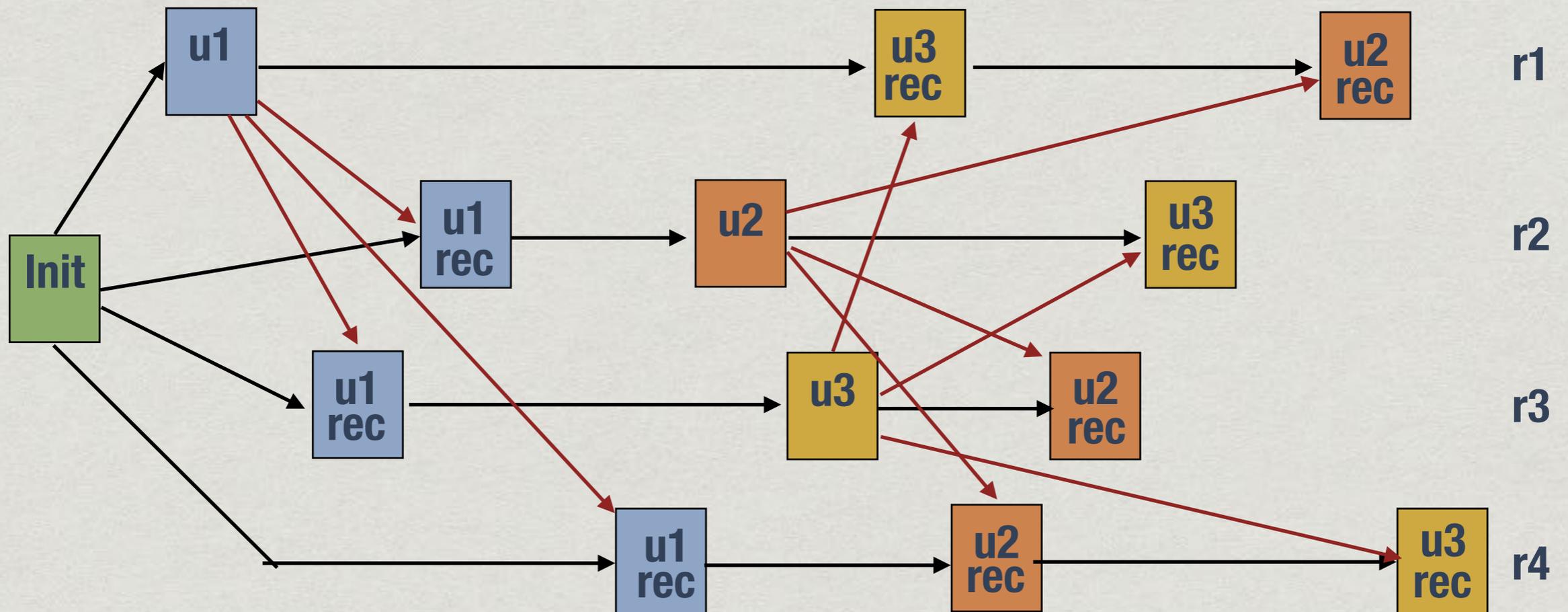
Runs of CRDTs ...

- * Happened before order on updates: Replica + Delivery
 - * Need not be transitive
 - * Causal delivery of messages makes it transitive



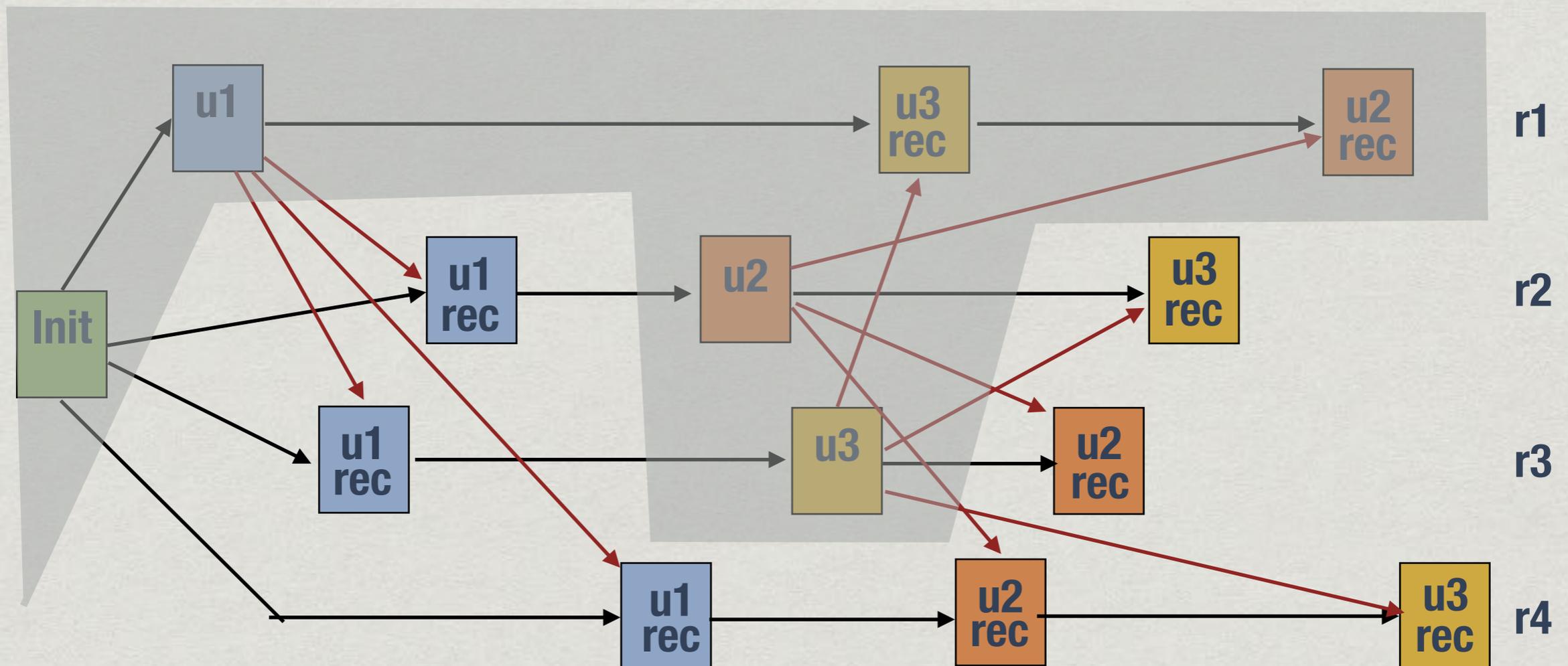
Runs of CRDTs ...

- * Local view of a replica
- * Whatever is visible below its maximal event



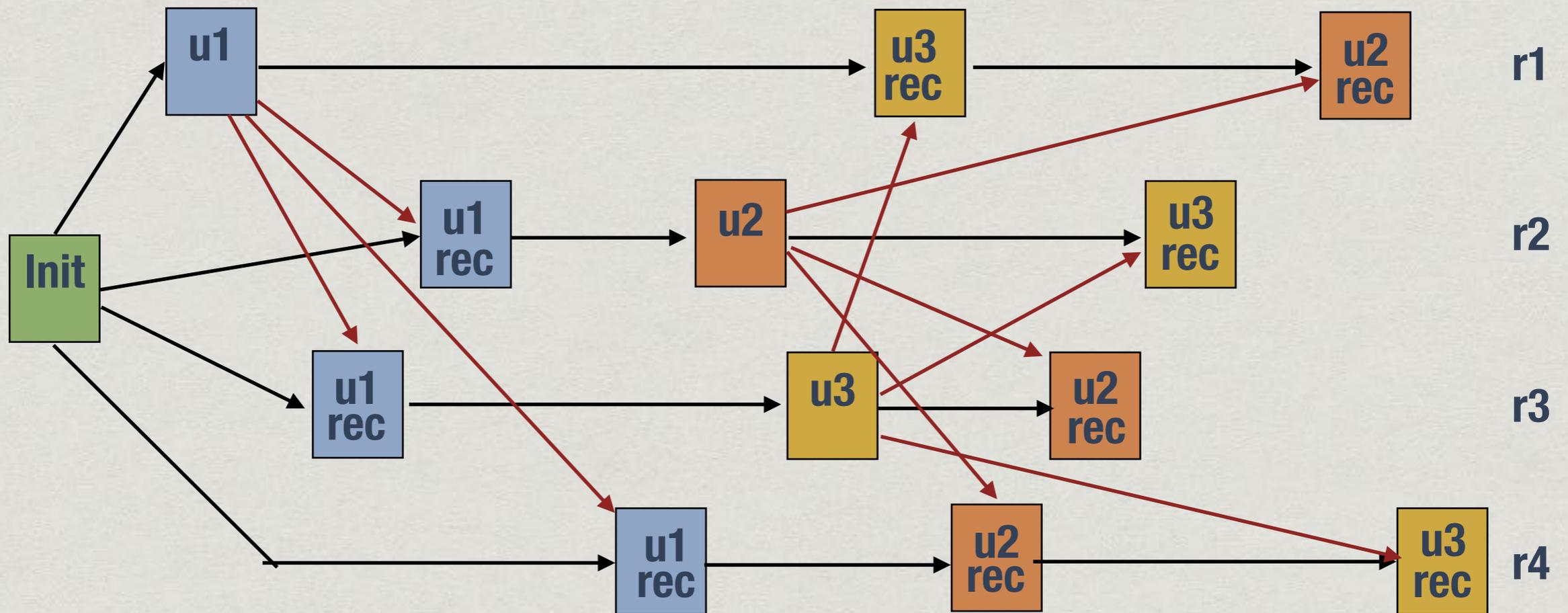
Runs of CRDTs ...

- * Local view of a replica
- * Whatever is visible below its maximal event



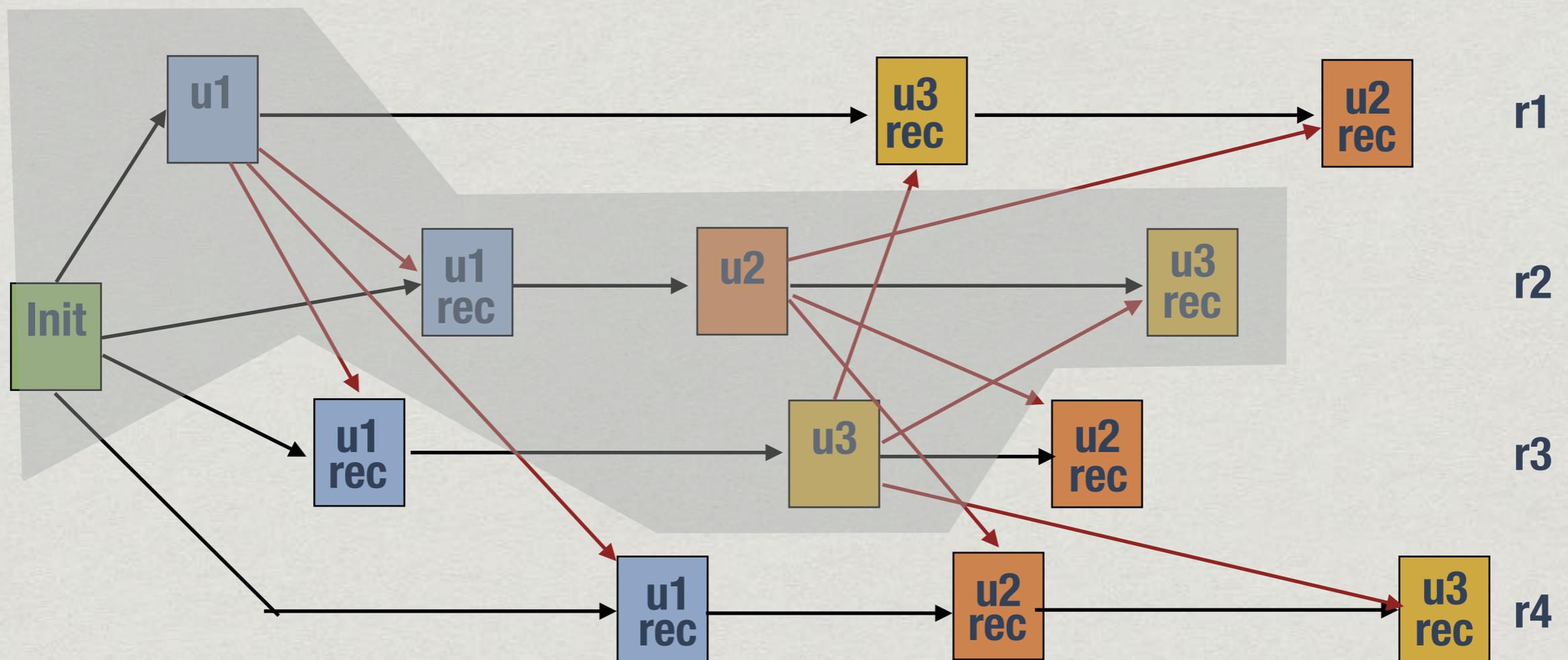
Runs of CRDTs ...

- * Local view of a replica
- * Whatever is visible below its maximal event



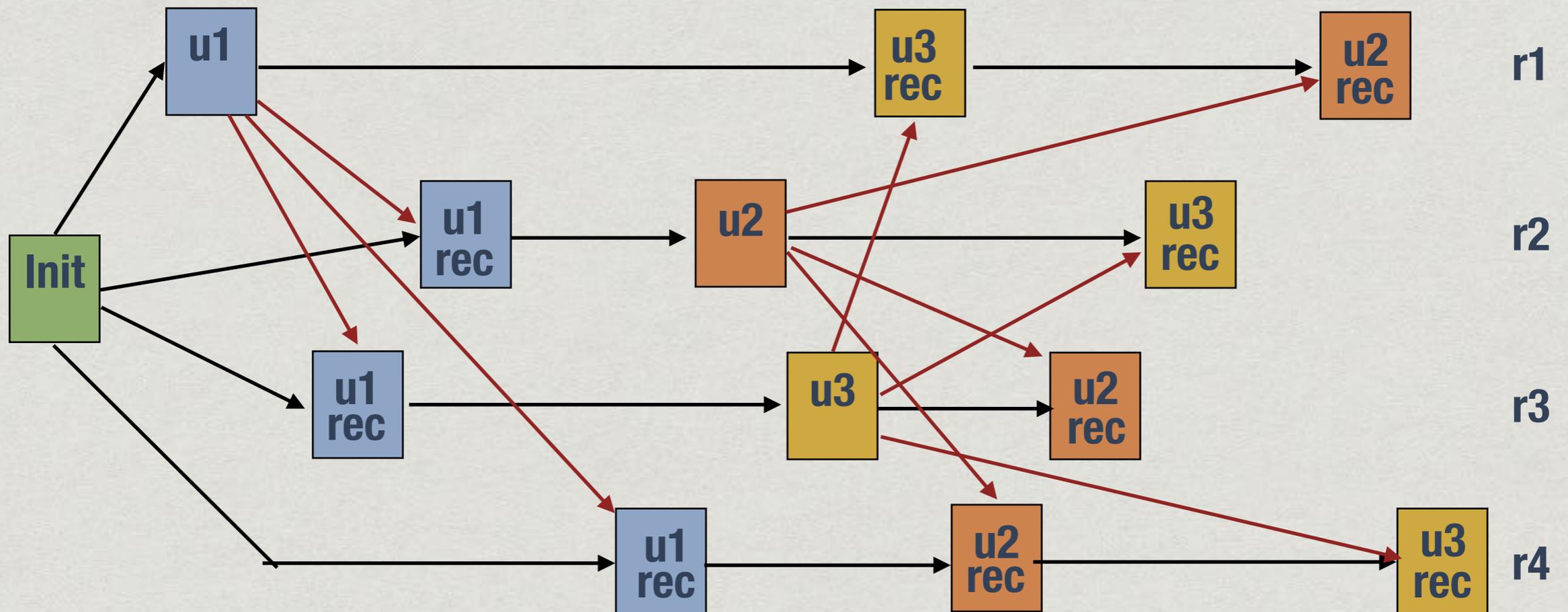
Runs of CRDTs ...

- * Local view of a replica
- * Whatever is visible below its maximal event



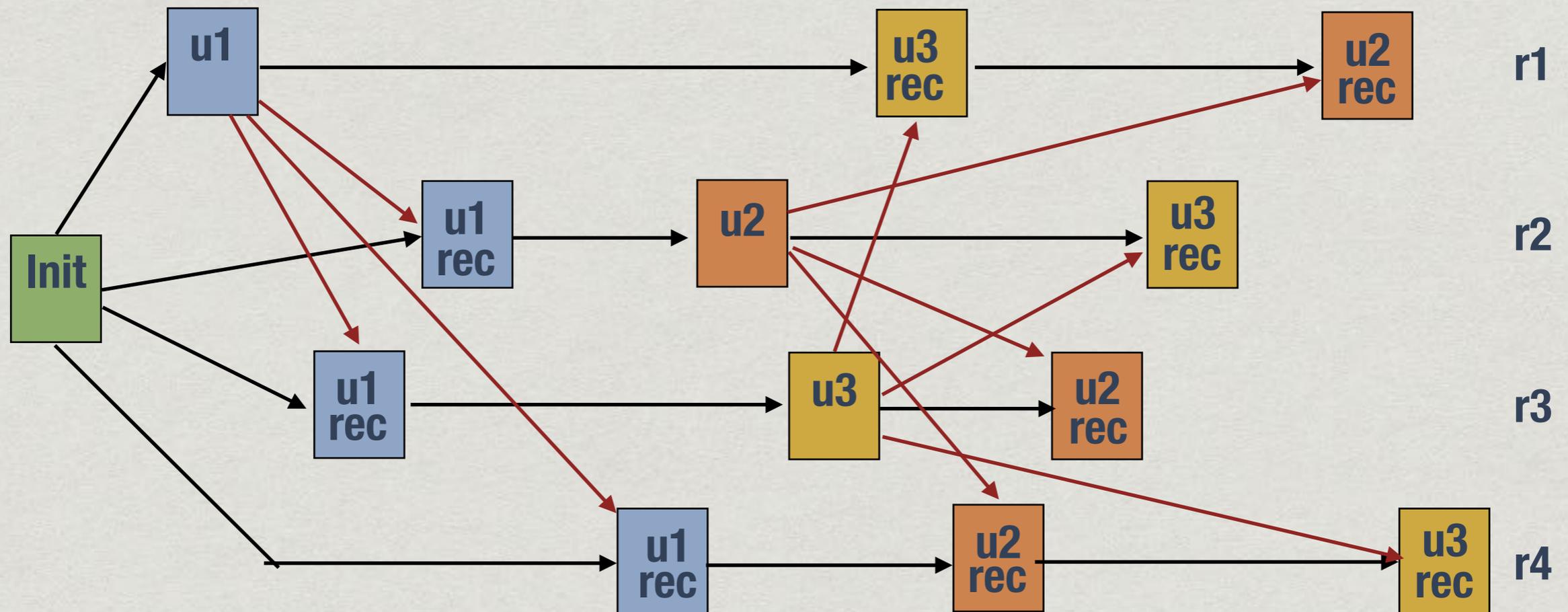
Runs of CRDTs ...

- * Local view of a replica
- * Whatever is visible below its maximal event



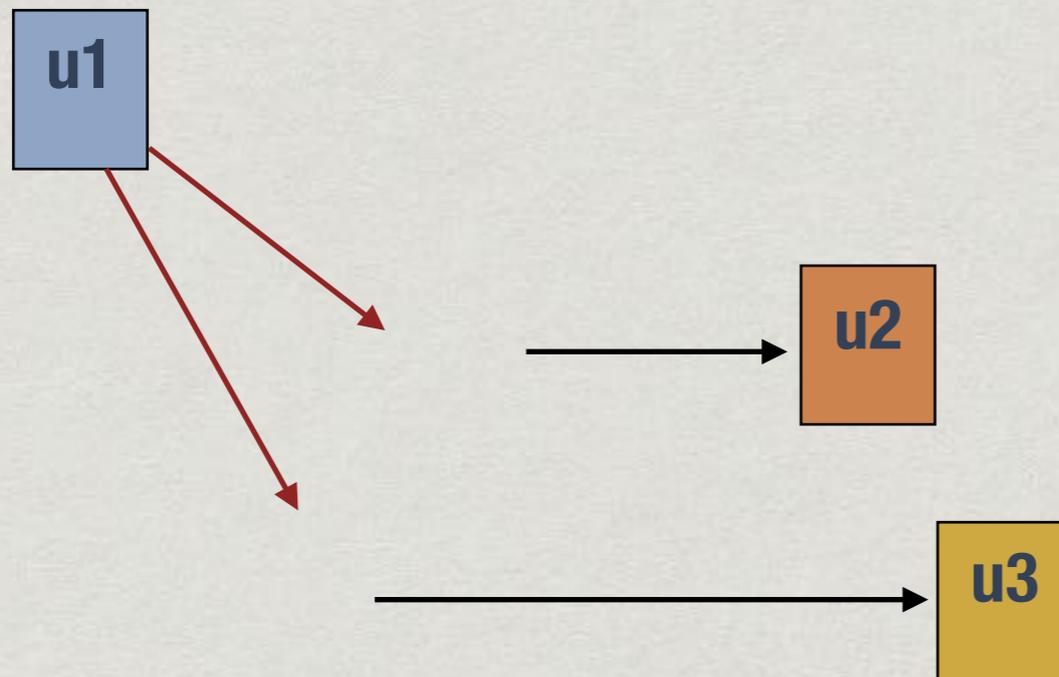
Runs of CRDTs ...

- * Even if updates are received locally in different orders, “happened before” on updates is the same



Runs of CRDTs ...

- * Even if updates are received locally in different orders, “happened before” on updates is the same



Declarative specification

- * Define queries in terms of partial order of updates in local view
- * For example: add wins in an OR-set
 - * Report “x in S” to be true if some maximal update is $\text{add}(x,S)$
 - * Concurrent $\text{add}(x,S)$, $\text{remove}(x,S)$ will both be maximal

Bounded past

- * Typically do not need entire local view to answer a query
- * Membership in OR-sets requires only maximal update for each element
 - * N events per element

Verification

- * Given a CRDT $D = (V, Q, U)$, does every run of D agree with the declarative specification?
- * Strategy
 - * Build a reference implementation from declarative specification
 - * Compare the behaviour of D with reference implementation

Finite-state implementations

- * Assume universe is bounded
- * Can use distributed timestamping to build a sophisticated distributed reference implementation [VMCAI 2015]
- * Asynchronous automata theory
- * Requires bounded concurrency for timestamps to be bounded

Global implementation

- * A simpler global implementation suffices for verification [ATVA 2015]
- * Each update event is labelled by the source replica with an integer (will be bounded later)
- * Maintain sequence of updates applied at each replica
 - * either local update from client
 - * or remote update received from another replica

Later Appearance Record

- * Each replica's history is an LAR of updates
 - * $(u_1, l_1) (u_2, l_2) \dots (u_k, l_k)$
 - * u_j has details about update: source replica, arguments
 - * l_j is label tagged to u_j by source replica
- * Labels are consistent across LARs — (u_i, l) in r_1 and (u_j, l) in r_2 denote same update event
- * Maintain LAR for each replica

Causality and concurrency

- * Suppose r_3 receives (u, l) from r_1 and (u', l') from r_2
 - * If (u, l) is causally before (u', l') , (u, l) must appear in r_2 's LAR before (u', l')
 - * If (u, l) is not causally before (u', l') and (u', l') is not causally before (u, l) , they must have been concurrent
- * Can recover partial order and answer queries according to declarative specification

Pruning LARs

- * Only need to keep latest updates in each local view
- * If (u,l) generated by r is not latest for any other replica, remove all copies of (u,l)
- * To prune LARs, maintain a global table keeping track of which updates are pending (not yet delivered to all replicas)
- * Labels of pruned events can be safely reused

Outcome

- * Simple global reference implementation that conforms to declarative specification of CRDT
- * Reference implementation is bounded if we make suitable assumptions about operating environment
 - * Bounded universe
 - * Bounded message delivery delays

Verification strategy

- * Counter Example Guided Abstraction Refinement (CEGAR)
 - * Build a finite-state abstraction of given CRDT
 - * Compute synchronous product with reference implementation
 - * If an incompatible state is reached, trace out corresponding bad run in CRDT
 - * If we find a bad run, we have found a bug
 - * If not, refine abstraction and repeat

One size does not fit all

- * Traditional strong consistency requires coordination across replicas
- * Eventual consistency provides very weak guarantees

A football match



- * Goals for A and B are recorded separately
- * Strong consistency: 3-4
- * Eventual consistency: 2-0, 2-3, 3-2, ...
- * Prefix: 1-0, 1-1, 1-2, 2-2, 3-2, 3-3, 3-4
- * Monotonic: Can't read 1-2 after 3-2

Multiple requirements

- * Shopping cart can be eventually consistent
- * Checkout requires synchronisation
- * Mobile wallet requires read-my-own writes
- * Same data store may support different consistency models across operations

An abstract model

- * Causal consistency plus Petri net like tokens
[Gotsman et al, POPL 2016]
- * Tokens model conflict, force synchronization
- * Proof rule to check correctness of a specification
- * Extend to an effective verification procedure?

The nature of replicas

- * Should all replicas service all operations?
- * Synchronized operations propagated via “core” servers
- * How to model an underground data vault?

In practice ...

- * Publicly available libraries like Riak for eventually consistent data structures
 - * Useful for high scores in online games etc
 - * Difficult to program “critical” operations
- * Distributed data stores like Redis
 - * No clear consistency guarantees

Speculation and apologies

- * “Sorry, the price for your booking has changed”
- * Systems speculate and then apologise
- * What is a good formal model?

Summary

- * Distributed data stores are increasingly in use
- * We have formal models for CRDTs, eventual consistency
- * The real world is more complicated
 - * Technology remains far ahead of theory
- * Good theoretical foundations can validate and clarify empirical advances