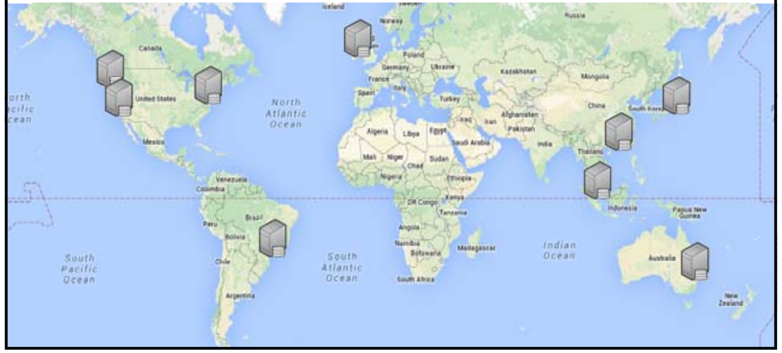
Specification & Complexity of Replicated Objects

Hagit Attiya, Technion

Replicated data stores

Geo-distributed systems driving Google, Facebook, Amazon, etc.



This talk

Theoretical exploration of **highly-available replicated data stores**

- Framework for reasoning
- Results on:
  - Achievable consistency
  - Lower bounds on message size and metadata overhead
- Clarify the landscape

This talk

Theoretical exploration of **highly-available replicated data stores**

Asynchronous message-passing algorithms implementing shared objects

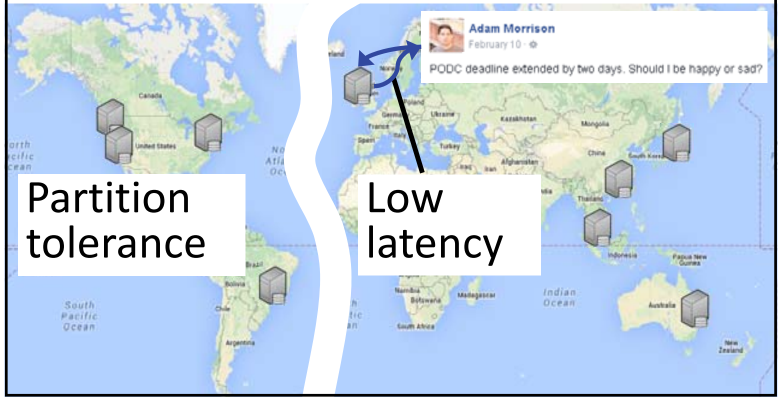
High availability

Respond without communication

Adam Morrison
February 10 ·
PODC deadline extended by two days. Should I be happy or sad?

High availability

Respond without communication



Adam Morrison
February 10 ·
PODC deadline extended by two days. Should I be happy or sad?

Partition tolerance

Low latency

Propagate Updates Later

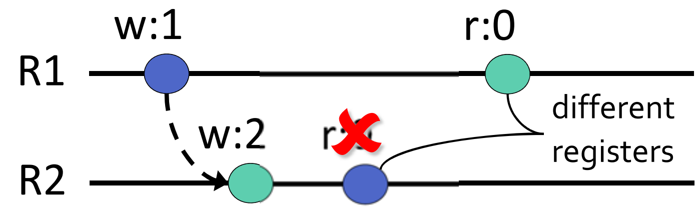
Converge to same state

Adam Morrison
February 10 ·
PODC deadline extended by two days. Should I be happy or sad?

Eventual consistency

CAP Theorem

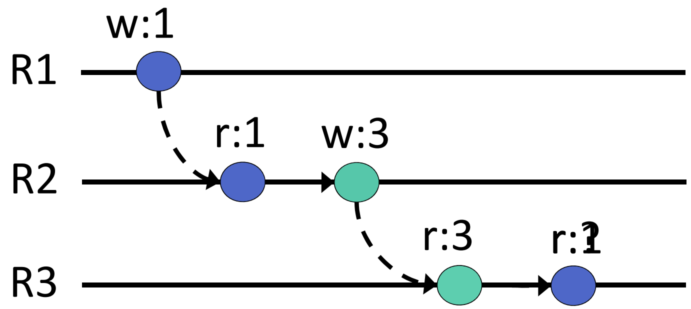
Cannot provide “strong” consistency (linearizability or serializability)



Causal Consistency

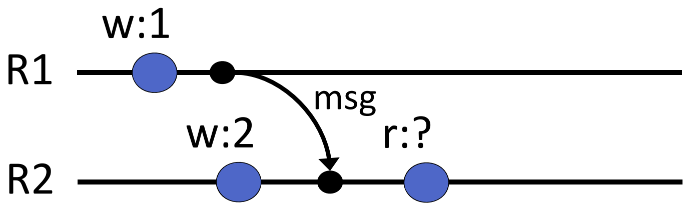
Following [Ahamad, Neiger, Burns, Kohli, Hutto]

If an operation is visible, so are its dependencies:



Exposing Concurrency

How to handle concurrent (conflicting) writes?

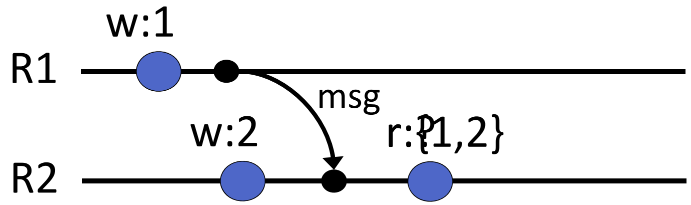


Exposing Concurrency

Practical approach: expose conflicts to user [Dynamo'07]

Multi-valued reg (**MVR**) :

Returns concurrent writes



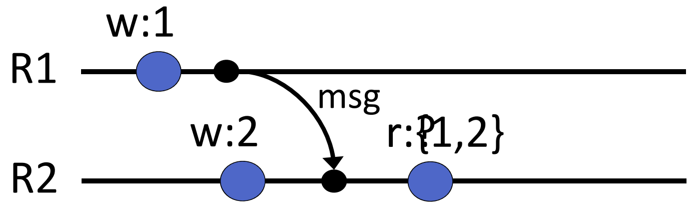
Exposing Concurrency

Mixing low-level & high-level details

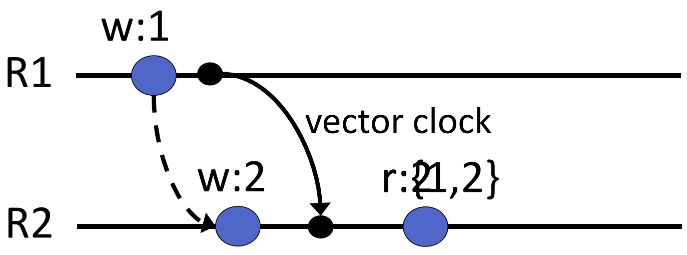
→ - - - - - →

Multi-valued reg (**MVR**) :

Returns concurrent writes



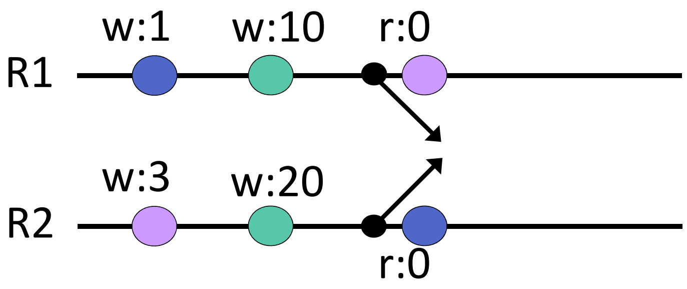
Why Non-Sequential Objects?



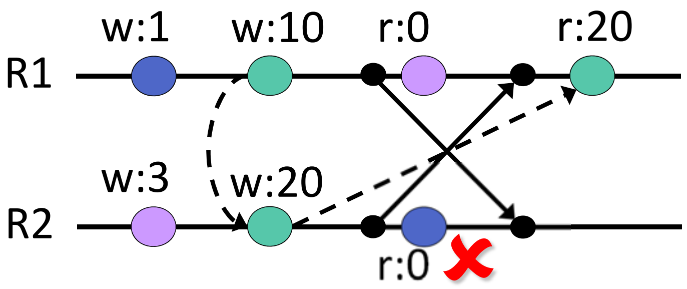
Works for single objects

[Perrin, Mostefaoui, Jard'14]

Causality Exposes Concurrency



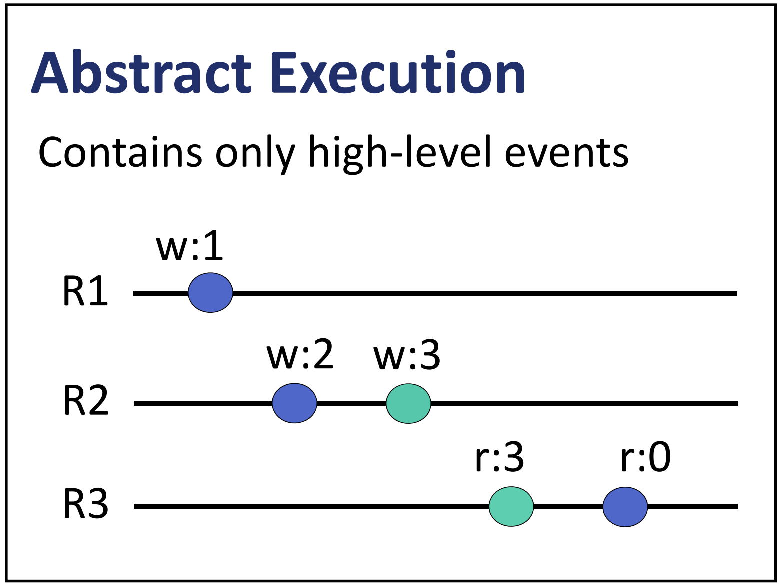
Causality Exposes Concurrency



Avoiding Low-Level Details

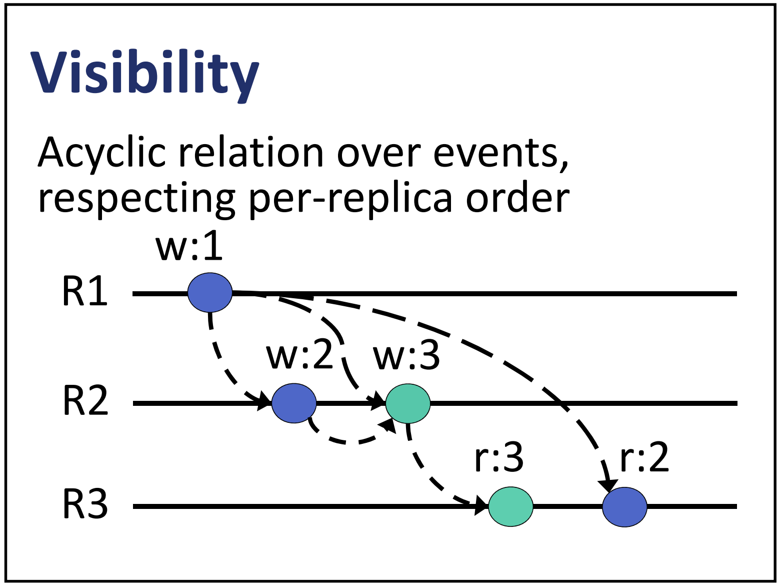
Specify replicated objects using **visibility** in **abstract executions**

[Burckhardt, Gotsman, Yang, Zawirski]



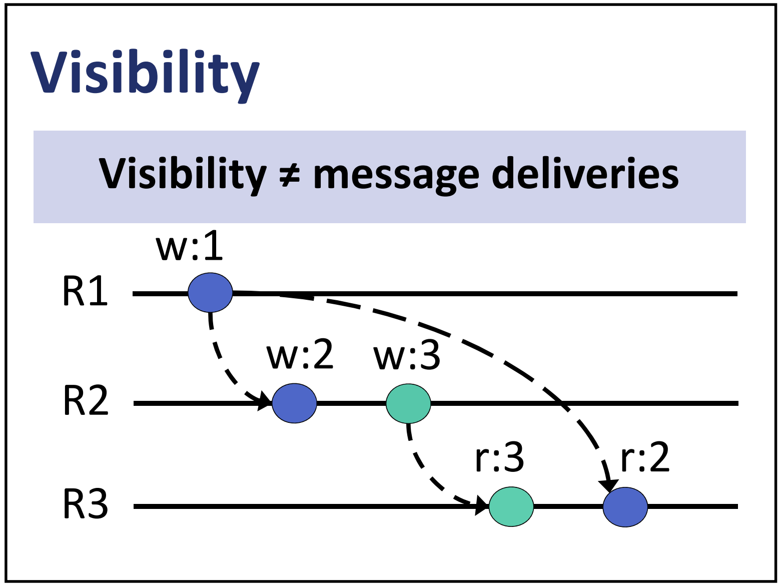
Abstract Execution

Contains only high-level events



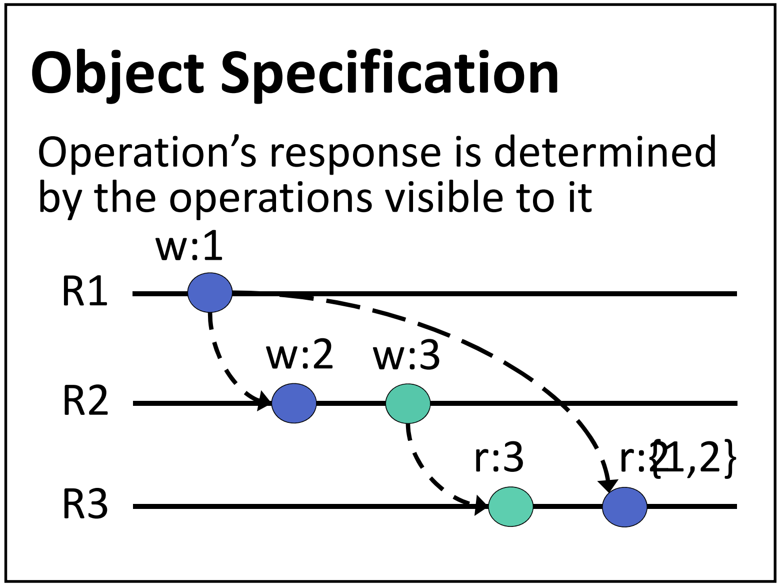
Visibility

Acyclic relation over events, respecting per-replica order



Visibility

Visibility ≠ message deliveries

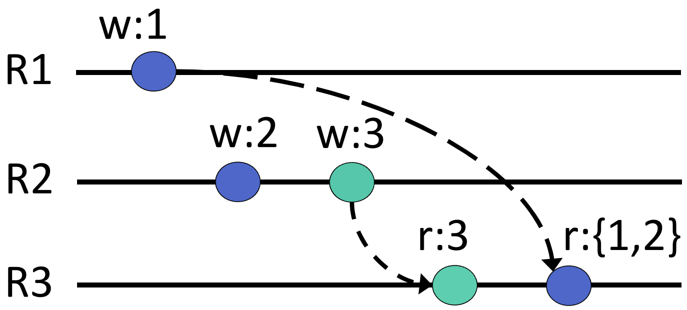


Object Specification

Operation’s response is determined by the operations visible to it

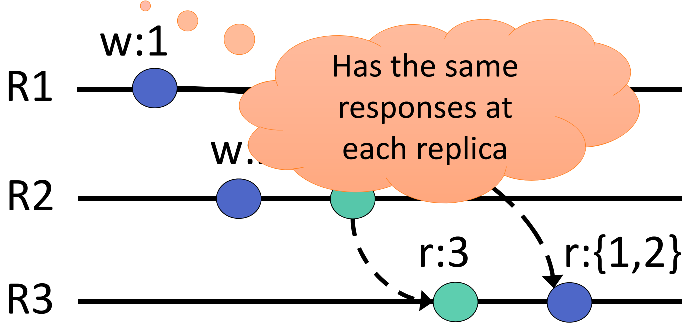
Object Specification

Concrete execution **implements** the object if it **complies** with an abstract object execution



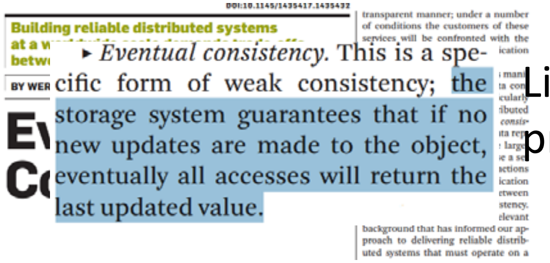
Object Specification

Concrete execution **implements** the object if it **complies** with an abstract object execution



Has the same responses at each replica

Eventual consistency



DOI:10.1145/1435417.1435432

Building reliable distributed systems at a w betw

BY WER

Ev Co

▸ *Eventual consistency.* This is a specific form of weak consistency; the storage system guarantees that if no new updates are made to the object, eventually all accesses will return the last updated value.

transparent manner; under a number of conditions the customers of these services will be confronted with the ication

background that has informed our approach to delivering reliable distributed systems that must operate on a

Liveness property

Eventual consistency

[Burckhardt, Gotsman, Yang, Zawirski]

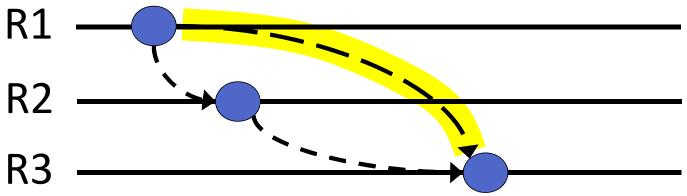
Infinite abstract execution is **eventually consistent** if an operation is invisible to only finitely many operations

Implies Vogels' informal definition

Causal consistency

Consistency model: prefix-closed set of abstract executions

Causal consistency: visibility is transitive



Comparing Consistency Models

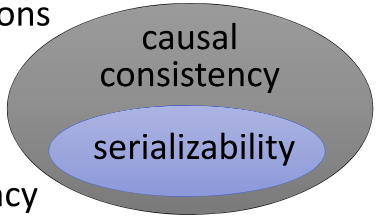
A consistency model is **satisfied** when all concrete executions comply with one of its abstract executions

Fewer abstract executions

⇨ stronger model

Bayou, PRACTI, COPS… satisfy causal consistency

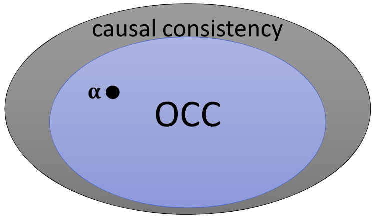
Can we satisfy a stronger model?



Consistency Limit Result

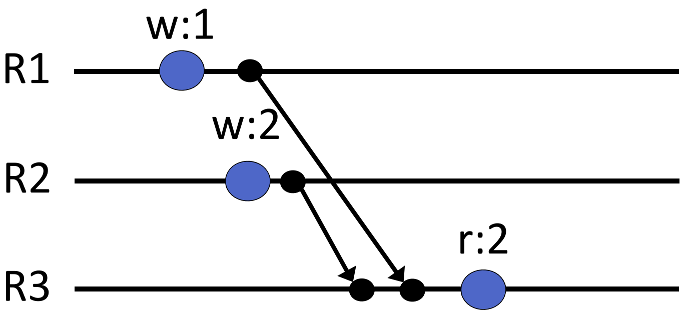
Theorem: Eventually consistent data store D does not satisfy a consistency model stronger than **observable causal consistency (OCC)**

OCC hides concurrency unless user can infer it



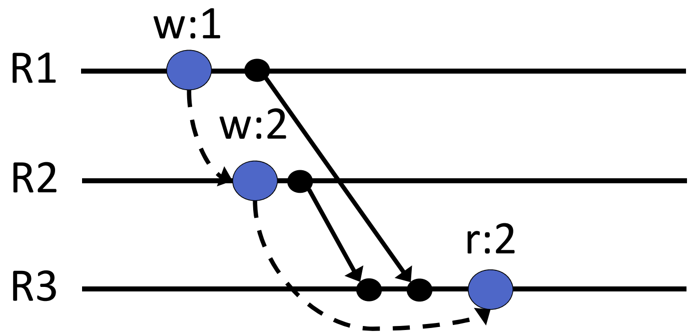
Deriving OCC

Goal: Comply with abstract execution without concurrency



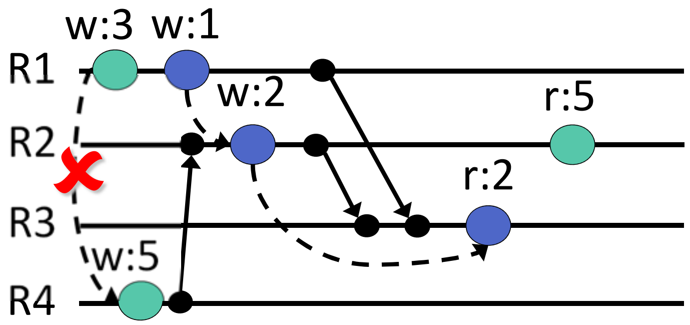
Deriving OCC

Goal: Comply with abstract execution without concurrency.



Deriving OCC

Goal: Comply with abstract execution without concurrency.



	Our Theorem	**CAC Theorem** [Mahajan, Alvisi, Dahlin]
Liveness	Eventual consistency	One-way convergence (stronger)
Strongest consistency	OCC	Causal consistency (weaker)
Tight?	Don't know	Yes*
Assumptions	invisible reads	
	msg sending	real time

Message lower bound

n: # of replicas
s: # of MVRs, each of **~lg k** bits

Theorem: $\Omega(\min\{n, s\} \lg k)$ bit message lower bound for causally & eventually consistent data store

Basically: vector clock if $s \geq n$

What if $s << n$?

Collaborative Text Editing

"Using Google Docs for Collaborative Work"

File Edit View Insert Format Tools Table Help

So just type in your netID@msu.edu and click sign in, which will take you to a new login page. On this page you'll re-enter your netID and add your password this time. Then click login and this will take you to google drive.

1. Create a new google Doc. (It's straightforward
2. Next, to add group members click on the "Share the page. This will prompt you to give your peo
   a. You'll need to know the netID of the othe recommendation would be to ask them MSU.edu and use the people search to
   b. Make sure you add your instructor whe make comments.
   c. and when you are finished click "Share
3. (fade into half-completed document, with collab show you a document with comments so you because this is a really powerful feature in Goo

When you or anyone in the group would like to

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search

Collaborative editing

From Wikipedia, the free encyclopedia

This article is about collaborative writing in a technical or academic sense. For collaborative writing of fiction, see Collaborative fiction.

Not to be confused with simultaneous editing or collaborative real-time editor.

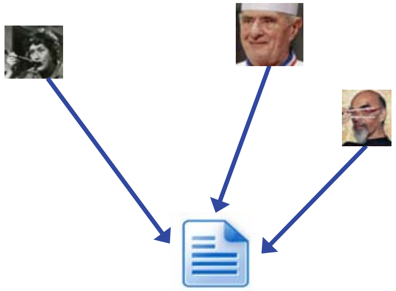
Collaborative editing is the practice of groups producing works together through individual contributions. Effective choices in group awareness, participation, and coordination are critical to successful collaborative writing outcomes.[1]

Contents [hide]
1 Overview
2 See also
3 References
4 Sources
5 External links

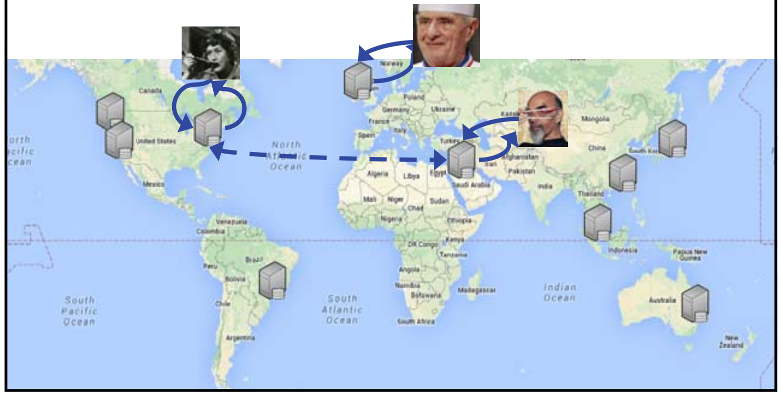
Overview [edit]

Collaborative writing is writing done by more than one person; they may discuss what they are going to write before they start, and discuss what they have written after they finish each draft they write.[2] The writing might be organized by dividing the

Collaborative Text Editing

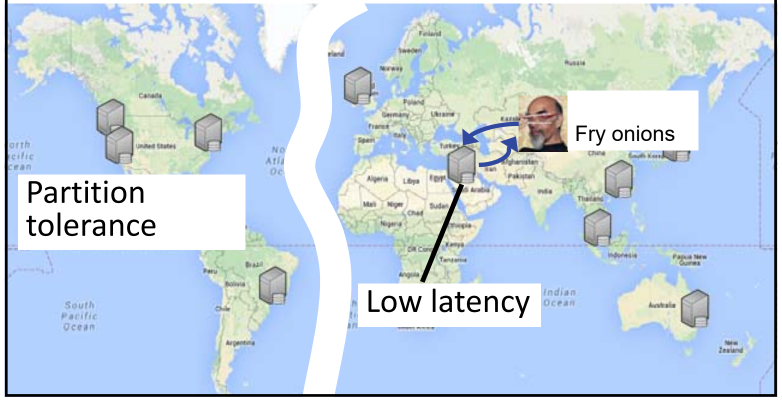


Collaborative Text Editing: Under the Hood



Collaborative Text Editing: Under the Hood

High availability: Respond without communication



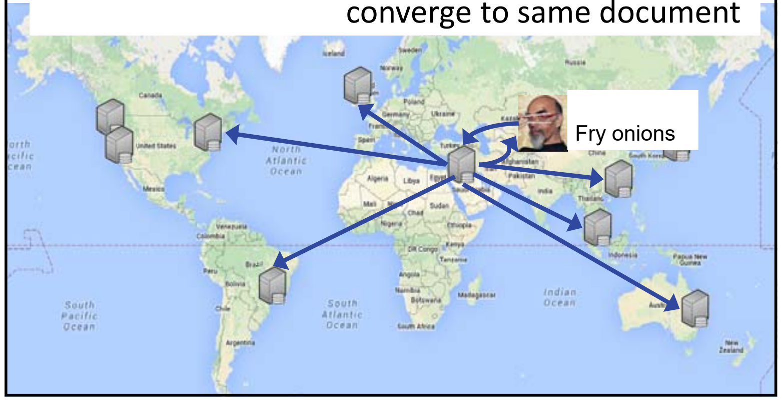
Fry onions

Partition tolerance

Low latency

Collaborative Text Editing: Under the Hood

Eventual consistency: Propagate changes
converge to same document



Fry onions

Replicated Object: List

Basic shared document editing operations:

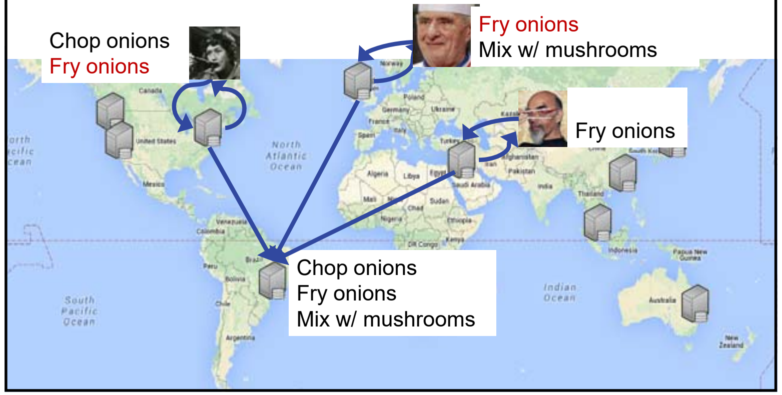
ins(a, pos) del(a) read()

(inserted elements are unique)

Every op returns state of the list:

ins(x, 0) : x ins(a, 1) : xa

Expected List Behavior



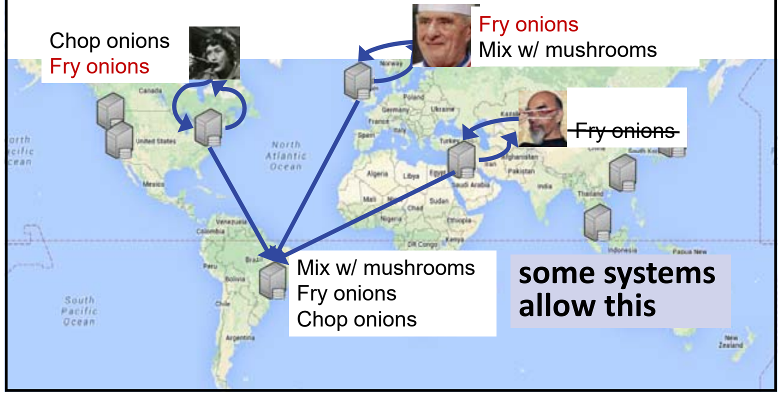
Chop onions
Fry onions

Fry onions
Mix w/ mushrooms

Fry onions

Chop onions
Fry onions
Mix w/ mushrooms

Expected List Behavior



Chop onions
Fry onions

Fry onions
Mix w/ mushrooms

~~Fry onions~~

Mix w/ mushrooms
Fry onions
Chop onions

some systems allow this

List Semantics

Shared document editing operations:

ins(a, pos) del(a) read()

(inserted elements are unique)

Every op returns state of the list

List Semantics

Shared document editing operations:

ins(a, pos) del(a) read()

(inserted elements are unique)

Every op returns state of the list

List of elements, each with previous ins()

but no previous del()

List Semantics

What does **previous** mean**?**

Can’t use messages received (low-level)

Again, use visibility in abstract executions

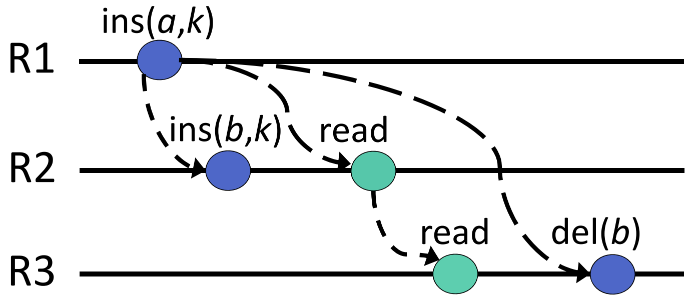
Every op returns state of the list

List of elements, each with previous ins()

but no previous del()

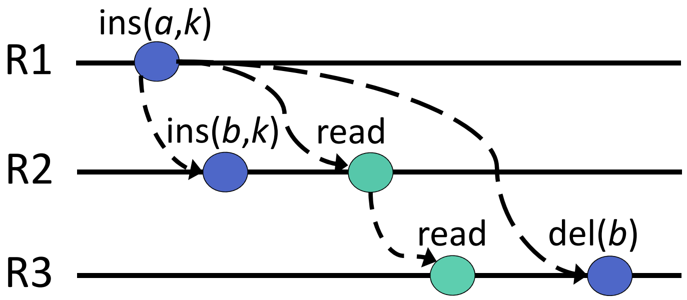
Implementing a List Object

Every concrete execution complies with an abstract list execution



Implementing a List Object

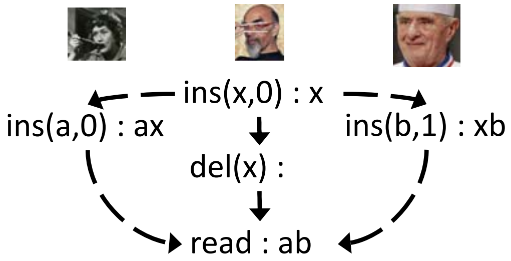
Each operation returns **ordered** list of elements with visible ins() but no visible del()



Strong List Specification

Strong list order: ∃ irreflexive relation that's transitive & total on **all inserted elements**

Intuition: remembers deleted elements



ins(x,0) : x

ins(a,0) : ax

ins(b,1) : xb

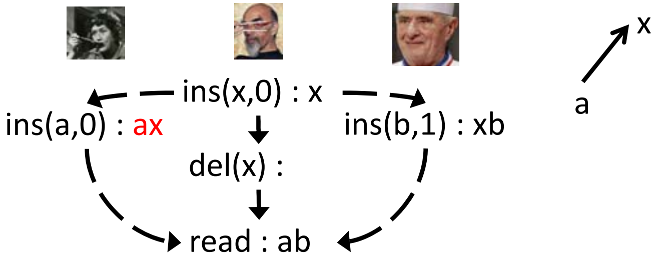
del(x) :

read : ab

Strong List Specification

Strong list order: ∃ irreflexive relation that's transitive & total on **all inserted elements**

Intuition: remembers deleted elements



ins(x,0) : x

ins(a,0) : ax

ins(b,1) : xb

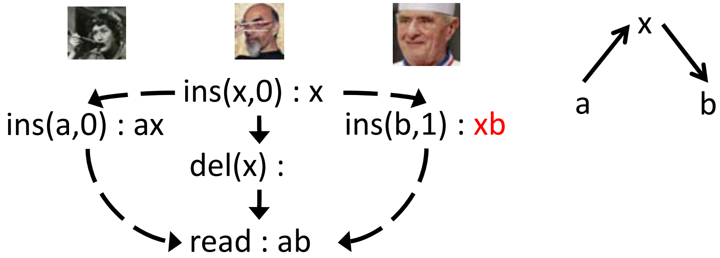
del(x) :

read : ab

Strong List Specification

Strong list order: ∃ irreflexive relation that's transitive & total on **all inserted elements**

Intuition: remembers deleted elements



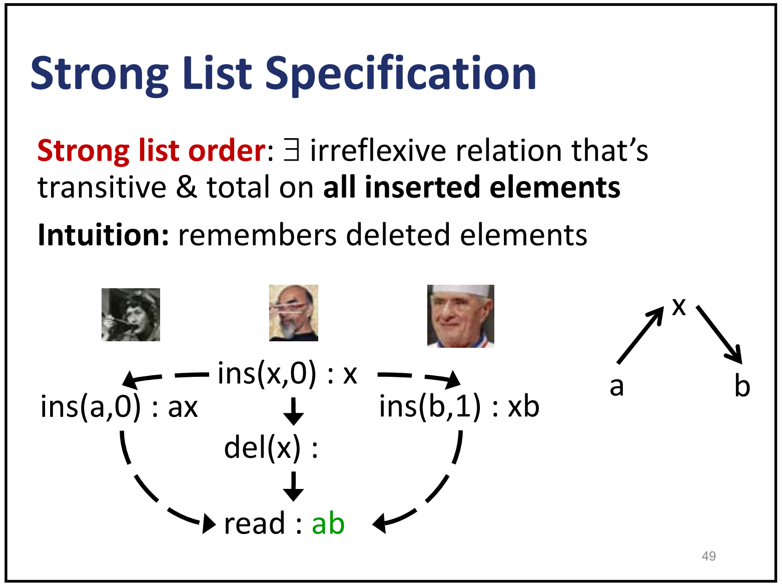
ins(x,0) : x

ins(a,0) : ax

ins(b,1) : xb

del(x) :

read : ab

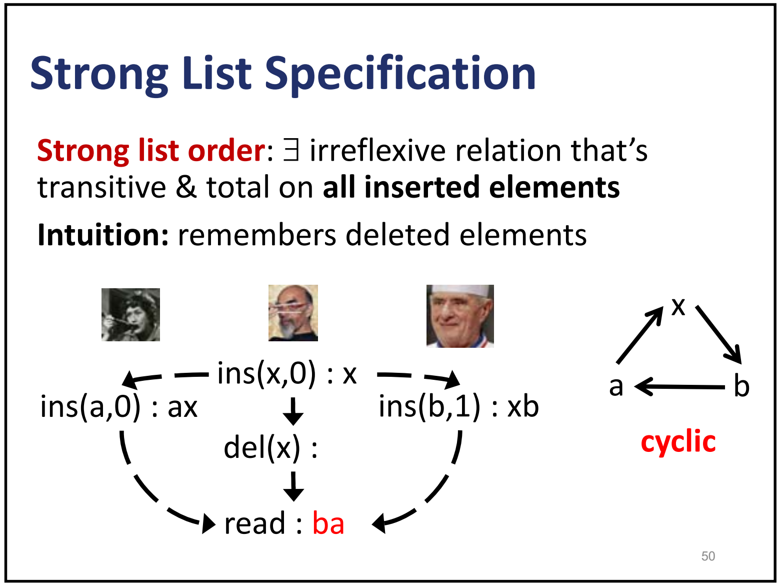


Strong List Specification

Strong list order: ∃ irreflexive relation that’s transitive & total on **all inserted elements**

Intuition: remembers deleted elements

ins(x,0) : x
ins(a,0) : ax
del(x) :
ins(b,1) : xb
read : ab
x
a
b

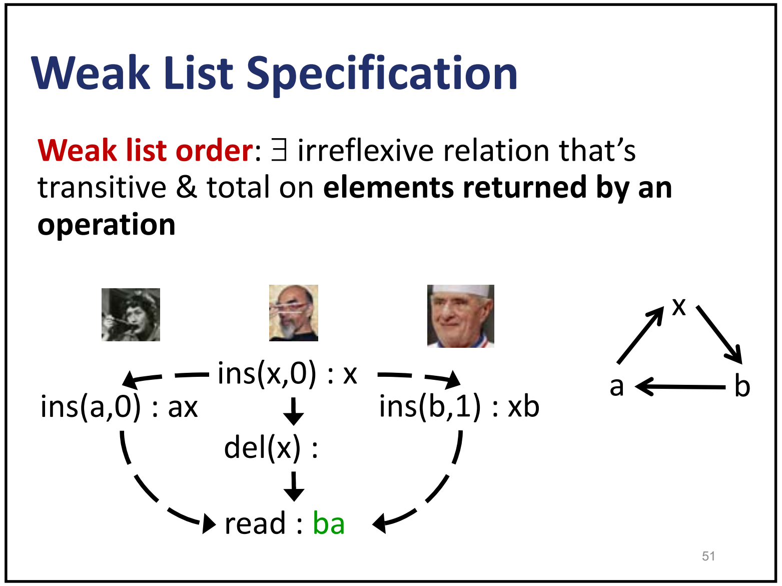


Strong List Specification

Strong list order: ∃ irreflexive relation that’s transitive & total on **all inserted elements**

Intuition: remembers deleted elements

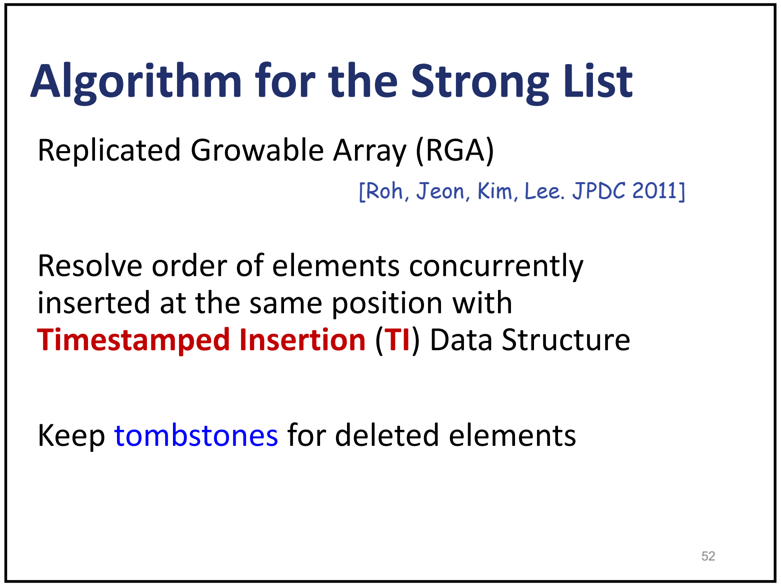
ins(x,0) : x
ins(a,0) : ax
del(x) :
ins(b,1) : xb
read : ba
x
a
b
cyclic



Weak List Specification

Weak list order: ∃ irreflexive relation that’s transitive & total on **elements returned by an operation**

ins(x,0) : x
ins(a,0) : ax
del(x) :
ins(b,1) : xb
read : ba
x
a
b



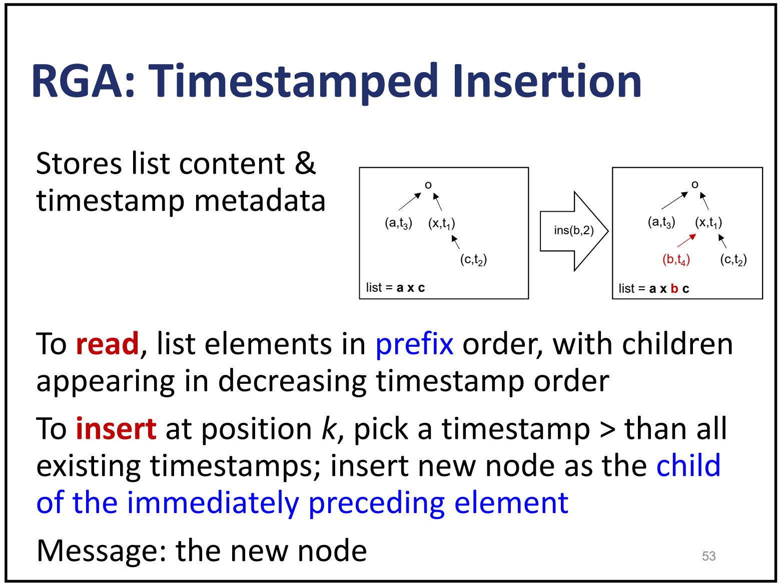
Algorithm for the Strong List

Replicated Growable Array (RGA)

[Roh, Jeon, Kim, Lee. JPDC 2011]

Resolve order of elements concurrently inserted at the same position with **Timestamped Insertion** (**TI**) Data Structure

Keep tombstones for deleted elements



RGA: Timestamped Insertion

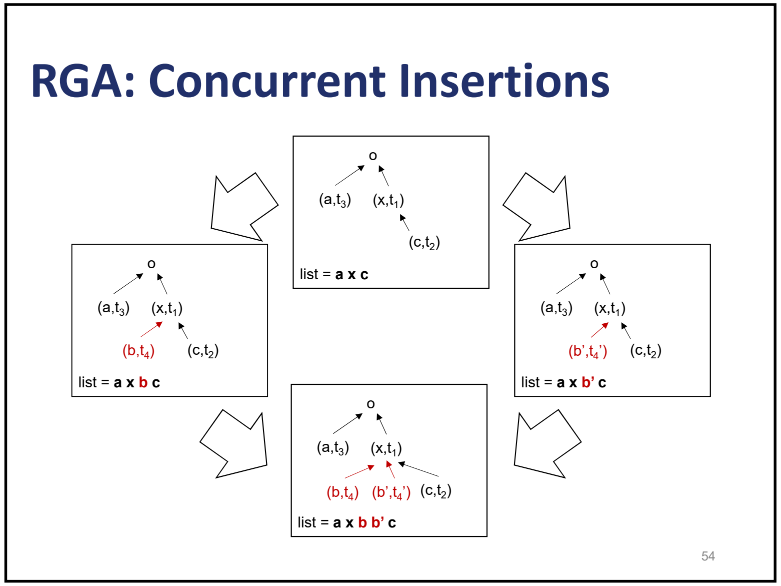
Stores list content & timestamp metadata

To **read**, list elements in prefix order, with children appearing in decreasing timestamp order

To **insert** at position *k*, pick a timestamp > than all existing timestamps; insert new node as the child of the immediately preceding element

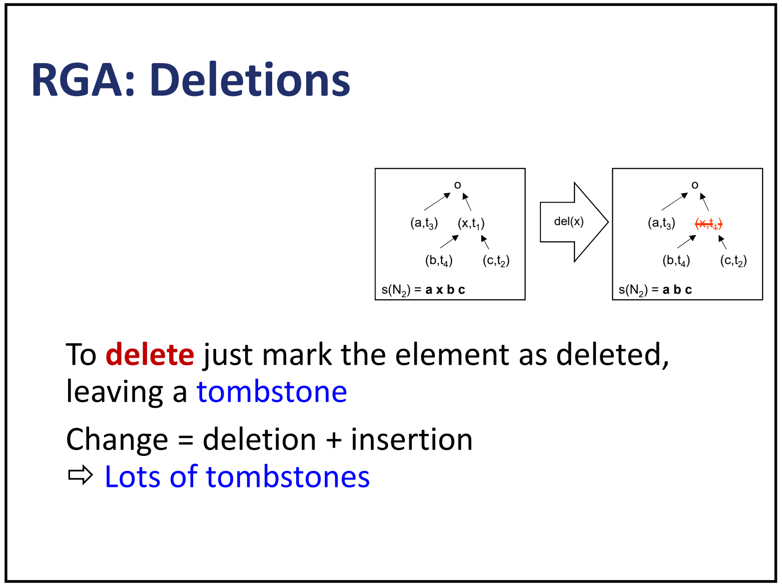
Message: the new node

53



RGA: Concurrent Insertions

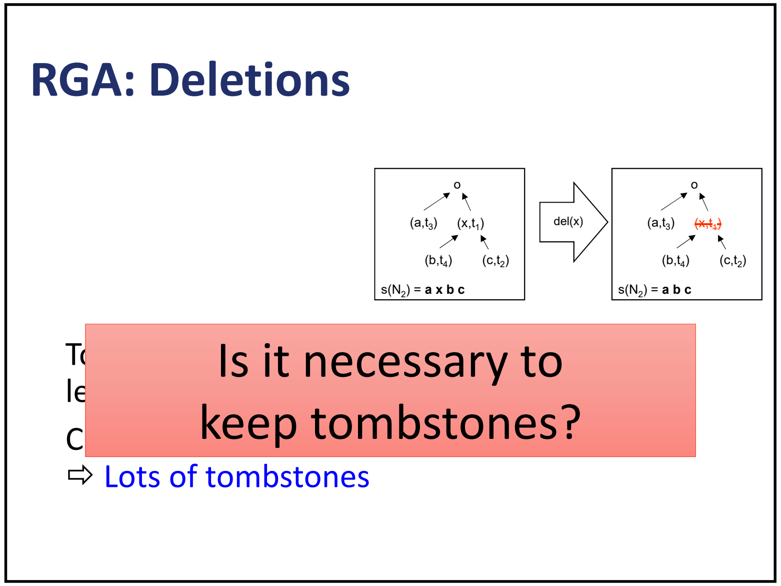
54



RGA: Deletions

To **delete** just mark the element as deleted, leaving a tombstone

Change = deletion + insertion
⇨ Lots of tombstones



RGA: Deletions

To

le

C

⇨ Lots of tombstones

Is it necessary to keep tombstones?

Are Tombstones Necessary?

Some algorithms don't have them:

- Treedoc [Preguiça, Marqués, Shapiro, Letia. ICDCS 2009]
  Logoot [Weiss, Urso, Molli. ICDCS 2009]

Element position = sequence of edge labels on the path from the root of the tree

Label stays the same after nodes are deleted

- Operational transformations (OT)
  Log updates; transform them locally

Are Tombstones Necessary?

Some algorithms don't have them:

- Treedoc [Preguiça, Marqués, Shapir
  Logoot [Weis

El sequence of edge labels on the rom the root of the tree

Label stays the same after nodes are deleted

Still a lot of **metadata**

$$\text{Metadata overhead} = \frac{\text{size of list state}}{\text{size of observable list}}$$

Metadata Lower Bound

There is an execution with *D* deletions, in which a replica has $\Omega(D)$-bit metadata overhead

✓ Even with **causal** atomic broadcast

✓ Even for **weak** specification

✗ Only for **push-based** protocols

– a replica sends updates to other replicas & merges updates from other replicas into its state as soon as possible

Metadata Lower Bound

There is an execution with *D* deletions, in which a replica has $\Omega(D)$-bit metadata overhead

Execution in which the list at some replica is "*" but replica's state is $\Omega(D)$ bits

Proof Technique

There are 2^D such strings ⇨ for some w, size of replica state after α_w is $\Omega(D)$ bits

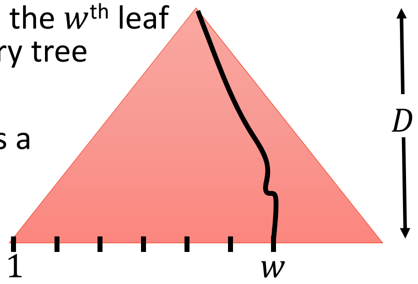
$\forall D$-bit string w, construct an execution α_w s.t.:

- ✓ A replica performs D deletions and receives no messages
- ✓ After α_w, the list at the replica is “*”
- ✓ w can be decoded from the state of the replica
- ✓ The replica has no pending messages

Encoding w

Encode the path to the w^{th} leaf of a complete binary tree

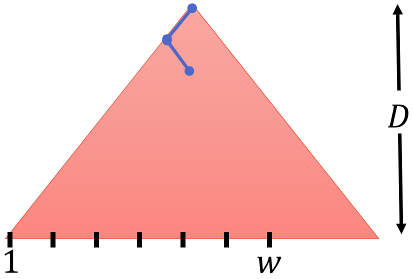
Considering w+1 as a number in $[1..2^D]$



- ✓ After α_w, the list at the replica is “*”
- ✓ w can be decoded from the state of the replica

Example: encoding $w = 01$

$[_0]_0$
send m_1
$[_1]_1[_0]_0$
send m_2
$[_1]_1[_2]_2[_0]_0$
send m_3
$[_1]_1[_2{*}]_2[_0]_0$
send m_4
~~$[_1]_1[_2{*}]_2[_0]_0$~~
send m_5



Output: encoding replica state, σ

Decoding w from σ (strong spec)

$[_0]_0$
send m_1
$[_1]_1[_0]_0$
send m_2
$[_1]_1[_2]_2[_0]_0$
send m_3
$[_1]_1[_2{*}]_2[_0]_0$
send m_4
~~$[_1]_1[_2{*}]_2[_0]_0$~~
send m_5

Reconstruct α_w iteratively

Decoding w from σ (strong spec)

R1

Reconstruct α_w iteratively

We know first step in α_w

Decoding w from σ (strong spec)

R1

$[_0]_0$

send m_1

Reconstruct α_w iteratively

We know first step in α_w

Decoding w from σ (strong spec)

Reconstruct α_w iteratively

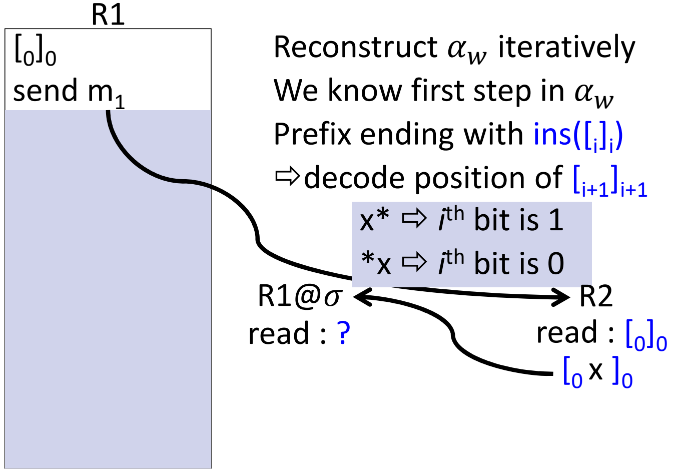
We know first step in α_w

Prefix ending with ins($[_i]_i$)

⇨decode position of $[_{i+1}]_{i+1}$

x* ⇨ ith bit is 1

*x ⇨ ith bit is 0



Decoding w from σ (strong spec)

Reconstruct α_w iteratively

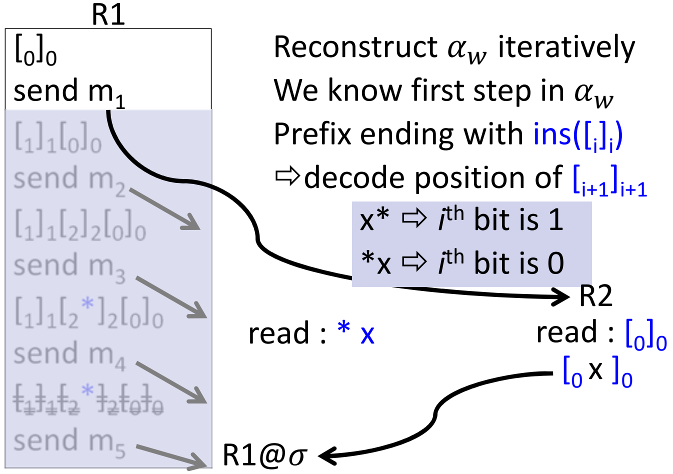
We know first step in α_w

Prefix ending with ins($[_i]_i$)

⇨decode position of $[_{i+1}]_{i+1}$

x* ⇨ ith bit is 1

*x ⇨ ith bit is 0



Decoding w from σ (strong spec)

R1

$[_0]_0$
send m_1
$[_1]_1[_0]_0$

Reconstruct α_w iteratively
We know first step in α_w
Prefix ending with ins($[_i]_i$)
⇨decode position of $[_{i+1}]_{i+1}$
x* ⇨ i^{th} bit is 1
*x ⇨ i^{th} bit is 0

R1@σ
read : * x

R2
read : $[_0]_0$
$[_0\, x\,]_0$

Decoding w from σ (strong spec)

R1

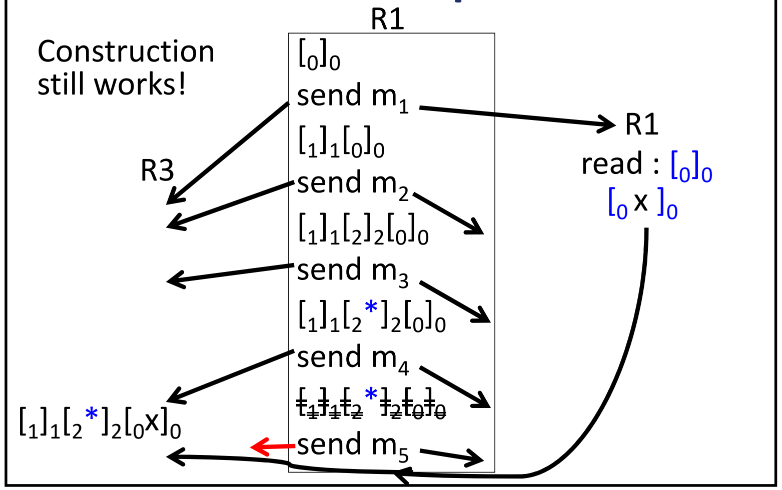
$[_0]_0$
send m_1
$[_1]_1[_0]_0$
send m_2

Reconstruct α_w iteratively
We know first step in α_w
Prefix ending with ins($[_i]_i$)
⇨decode position of $[_{i+1}]_{i+1}$
x* ⇨ i^{th} bit is 1
*x ⇨ i^{th} bit is 0

R1@σ
read : * x

R2
read : $[_0]_0$
$[_0\, x\,]_0$

Extension to weak spec



Construction still works!

R1

$[_0]_0$
send m_1
$[_1]_1[_0]_0$
send m_2
$[_1]_1[_2]_2[_0]_0$
send m_3
$[_1]_1[_2{*}]_2[_0]_0$
send m_4
~~$[_1]_1[_2{*}]_2[_0]_0$~~
send m_5

R3

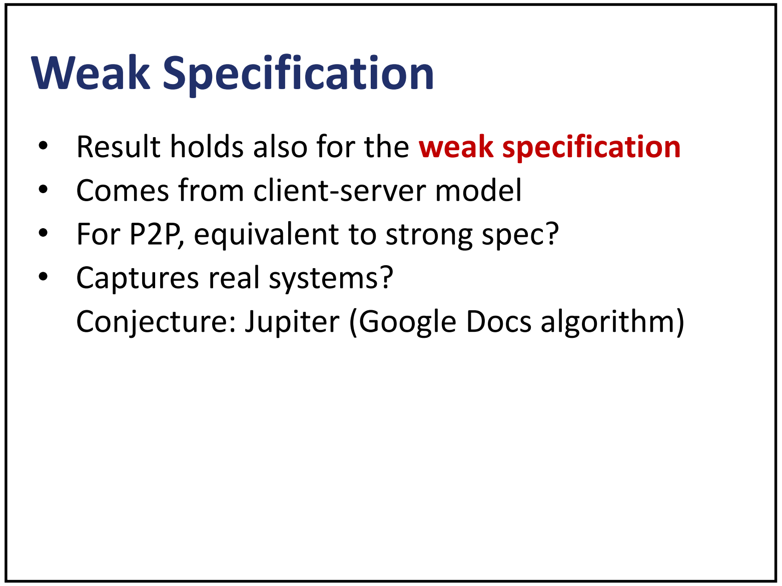
$[_1]_1[_2{*}]_2[_0x]_0$

R1
read : $[_0]_0$
$[_0\, x\,]_0$

Extensions

- Result holds for **client-server** model
  - Proof's execution satisfies **atomic broadcast**: All replicas receive messages in same order
  - Replicas can maintain server's state
  - Encoding replica receives no messages = Server is in its initial state

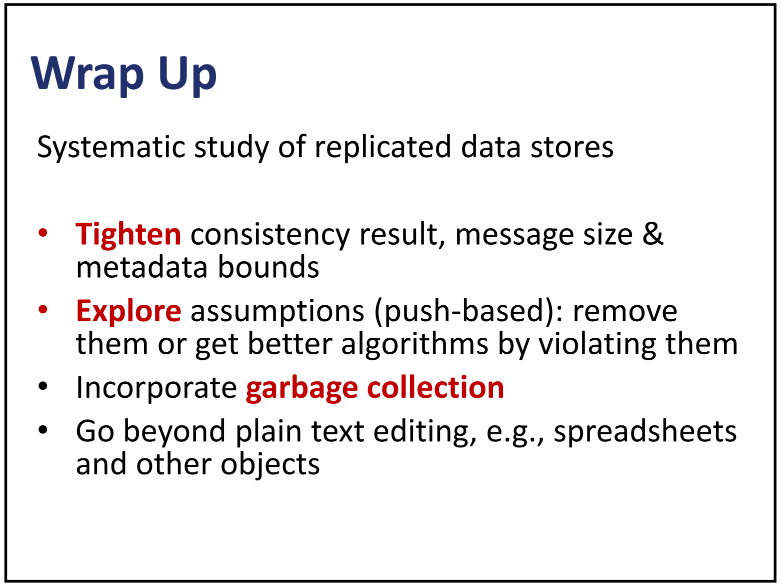
⇨ $\Omega(D)$-bit metadata overhead **for clients**



Weak Specification

- Result holds also for the **weak specification**
- Comes from client-server model
- For P2P, equivalent to strong spec?
- Captures real systems?

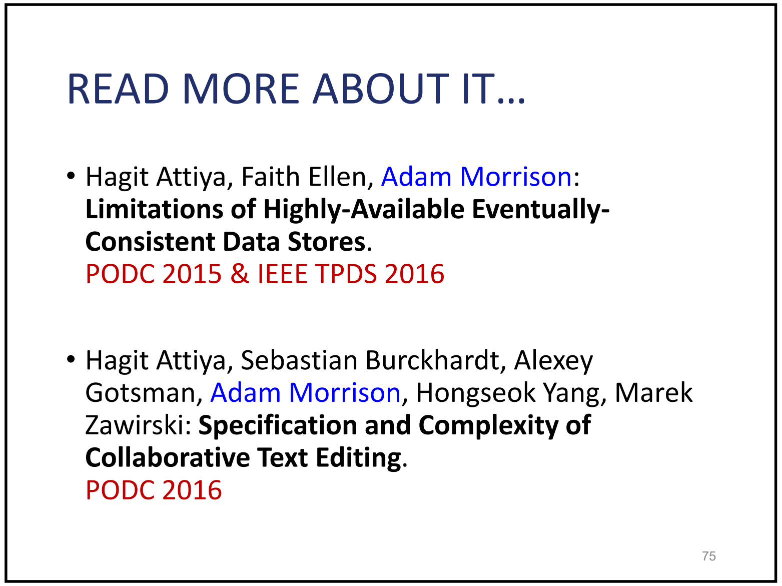
  Conjecture: Jupiter (Google Docs algorithm)



Wrap Up

Systematic study of replicated data stores

- **Tighten** consistency result, message size & metadata bounds
- **Explore** assumptions (push-based): remove them or get better algorithms by violating them
- Incorporate **garbage collection**
- Go beyond plain text editing, e.g., spreadsheets and other objects



READ MORE ABOUT IT…

- Hagit Attiya, Faith Ellen, Adam Morrison: **Limitations of Highly-Available Eventually-Consistent Data Stores**. PODC 2015 & IEEE TPDS 2016

- Hagit Attiya, Sebastian Burckhardt, Alexey Gotsman, Adam Morrison, Hongseok Yang, Marek Zawirski: **Specification and Complexity of Collaborative Text Editing**. PODC 2016

75