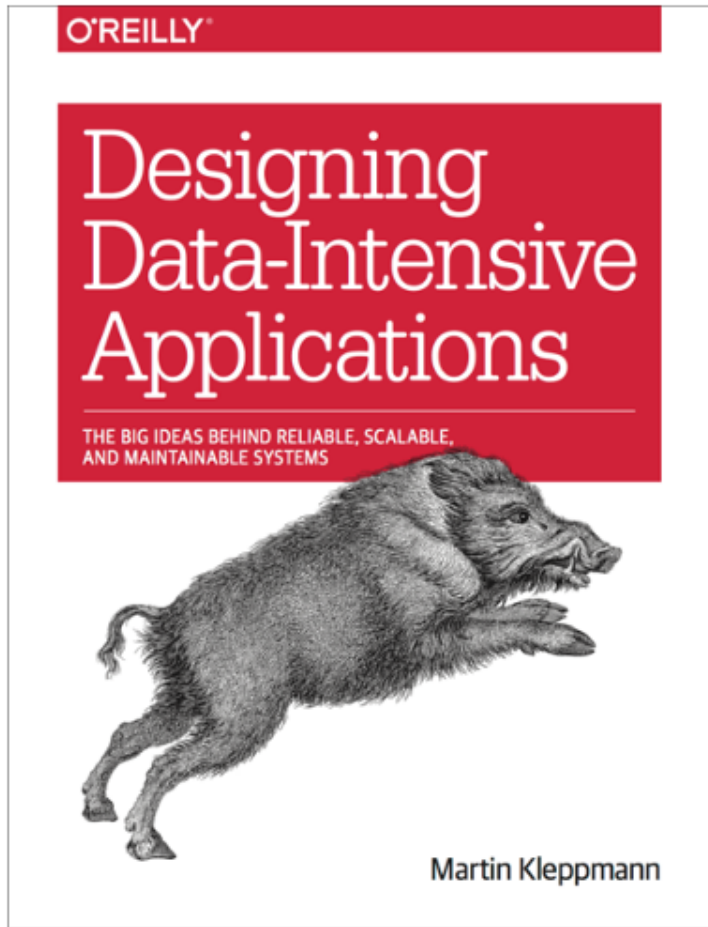


CRDTs:

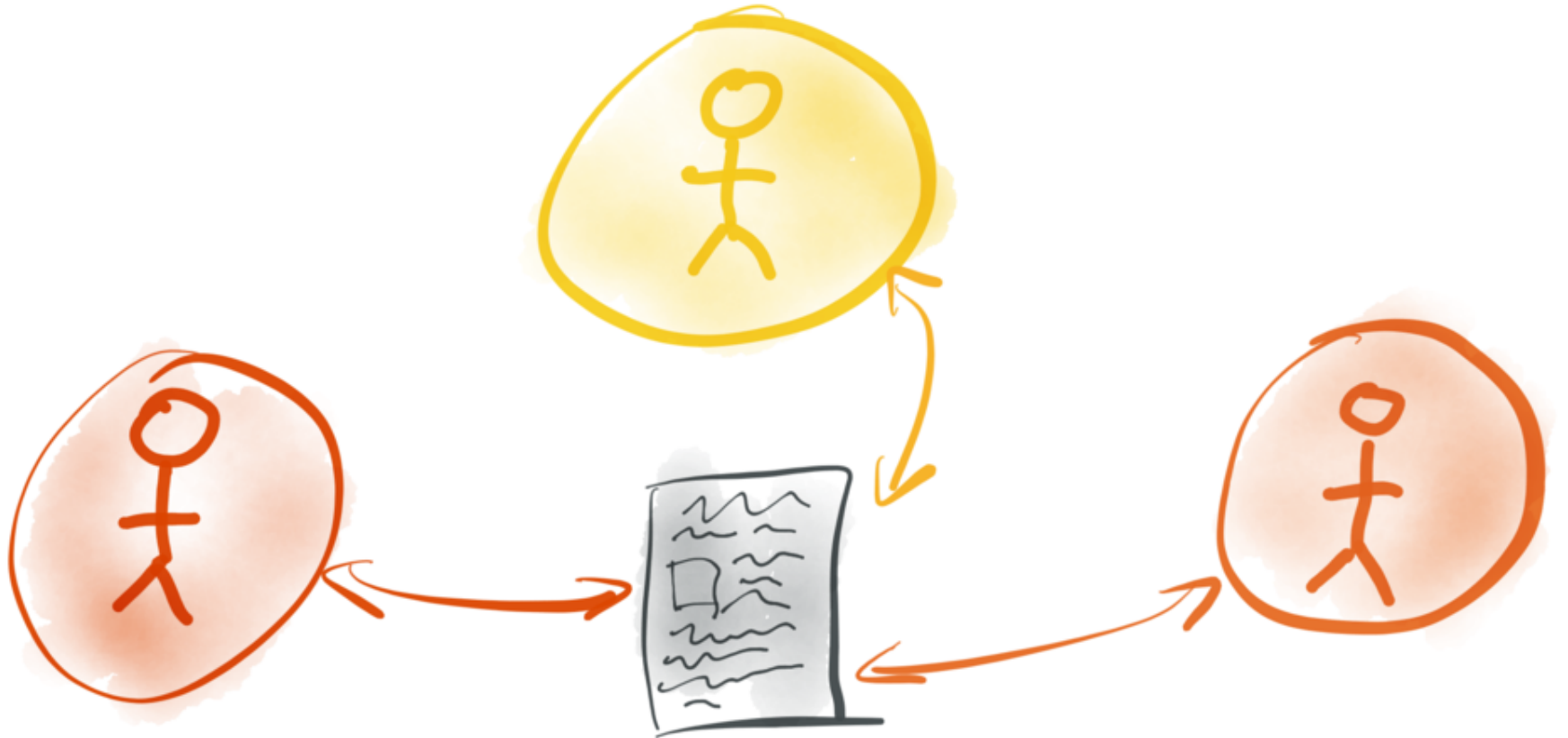
THE HARD PARTS

Martin Kleppmann, University of Cambridge
@martinkl mk428@cst.cam.ac.uk

dataintensive.net



COLLABORATIVE APPLICATIONS



COLLABORATIVE APPLICATIONS



Google Docs



Google Sheets



Office 365



Figma



Example: Text editing



Example: Text editing

insert "World"
after "Hello"



"Hello!"

"Hello World!"

time

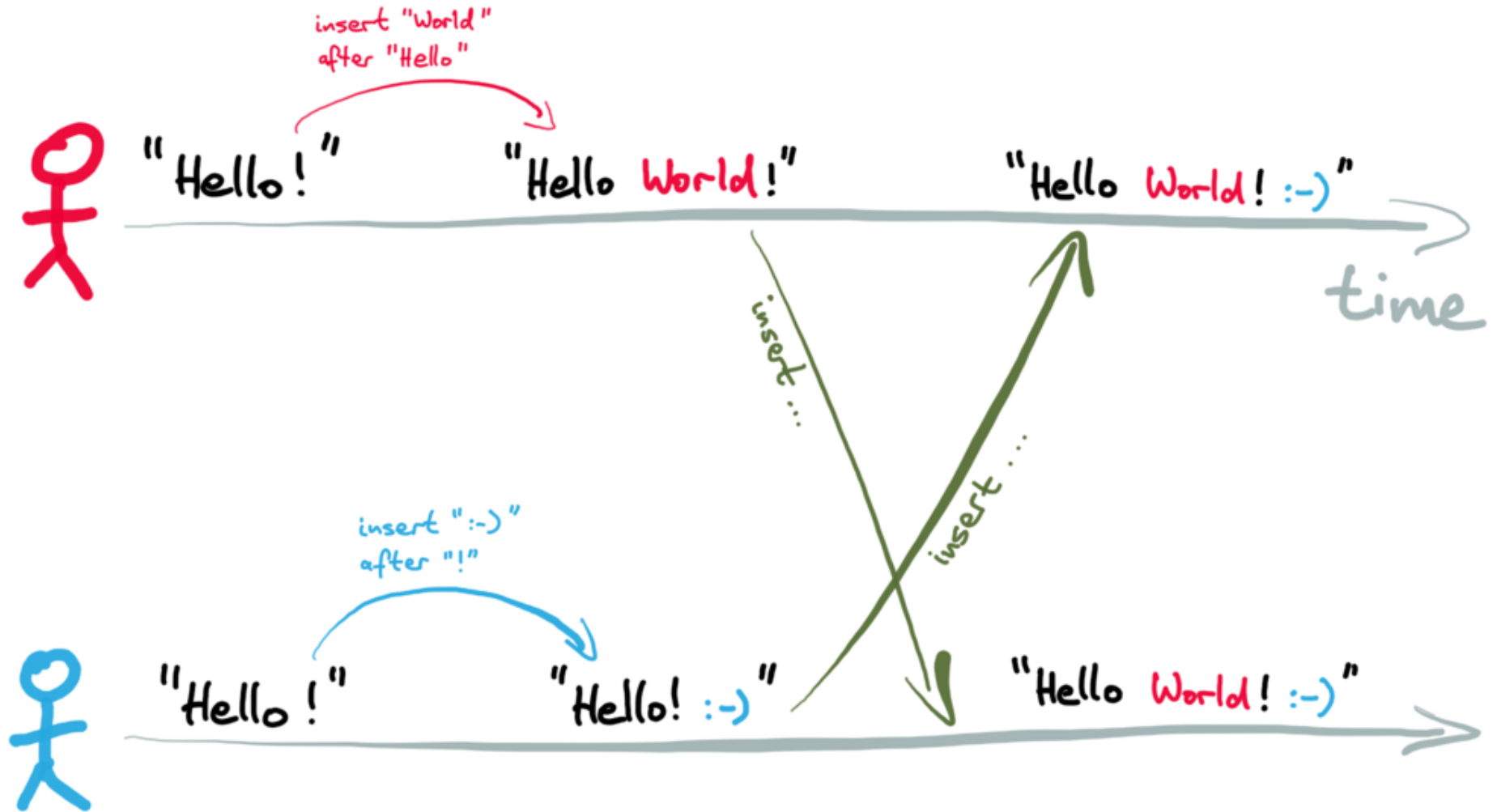
insert ":-)"
after "!"



"Hello!"

"Hello! :-)"

Example: Text editing



Algorithms for convergence

OPERATIONAL TRANSFORMATION (OT)

- e.g. Google Docs,
MS Office Online

1989-2006

CONFLICT-FREE REPLICATED DATA TYPES (CRDTs)

- e.g. Riak, TomTom GPS,
Teletype for Atom, ...

2006 - present

GOOGLE DOCS (NUTSHELL)

H	e	l	o
0	1	2	3

H	e	l	o
0	1	2	3

GOOGLE DOCS (NUTSHELL)

H	e	l	o
---	---	---	---

0 1 2 3

↓ edit

H	e	l	l	o
---	---	---	---	---

0 1 2 3 4

H	e	l	o
---	---	---	---

0 1 2 3

↓ edit

H	e	l	o	!
---	---	---	---	---

0 1 2 3 4

GOOGLE DOCS (NUTSHELL)

H	e	l	o
---	---	---	---

0 1 2 3

↓ edit

H	e	l	l	o
---	---	---	---	---

0 1 2 3 4

insert "l"
at pos 3

H	e	l	o
---	---	---	---

0 1 2 3

↓ edit

H	e	l	o	!
---	---	---	---	---

0 1 2 3 4

insert "!" at
position 4

server

GOOGLE DOCS (NUTSHELL)

H	e	l	o
---	---	---	---

0 1 2 3

↓ edit

H	e	l	l	o
---	---	---	---	---

0 1 2 3 4

insert "l"
at pos 3

server

H	e	l	o
---	---	---	---

0 1 2 3

↓ edit

H	e	l	o	!
---	---	---	---	---

0 1 2 3 4

insert "!" at
position 4

insert "l" at
position 3

H	e	l	l	o	!
---	---	---	---	---	---

0 1 2 3 4 5

GOOGLE DOCS (NUTSHELL)

H	e	l	o
---	---	---	---

0 1 2 3

⇓ edit

H	e	l	l	o
---	---	---	---	---

0 1 2 3 4

insert "l"
at pos 3

server

insert "!"
at position 5 (!)

H	e	l	l	o	!
---	---	---	---	---	---

0 1 2 3 4 5

H	e	l	o
---	---	---	---

0 1 2 3

⇓ edit

H	e	l	o	!
---	---	---	---	---

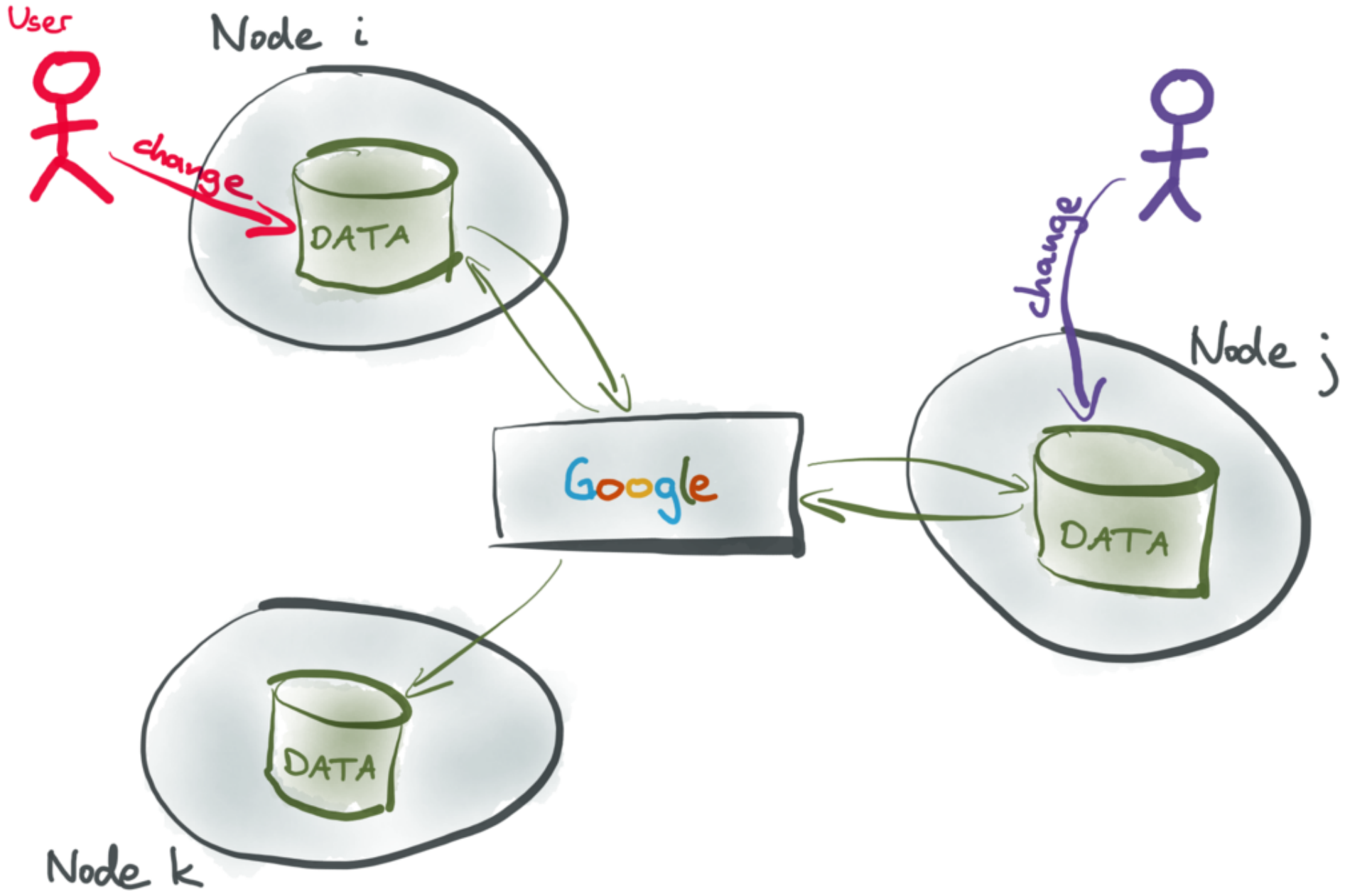
0 1 2 3 4

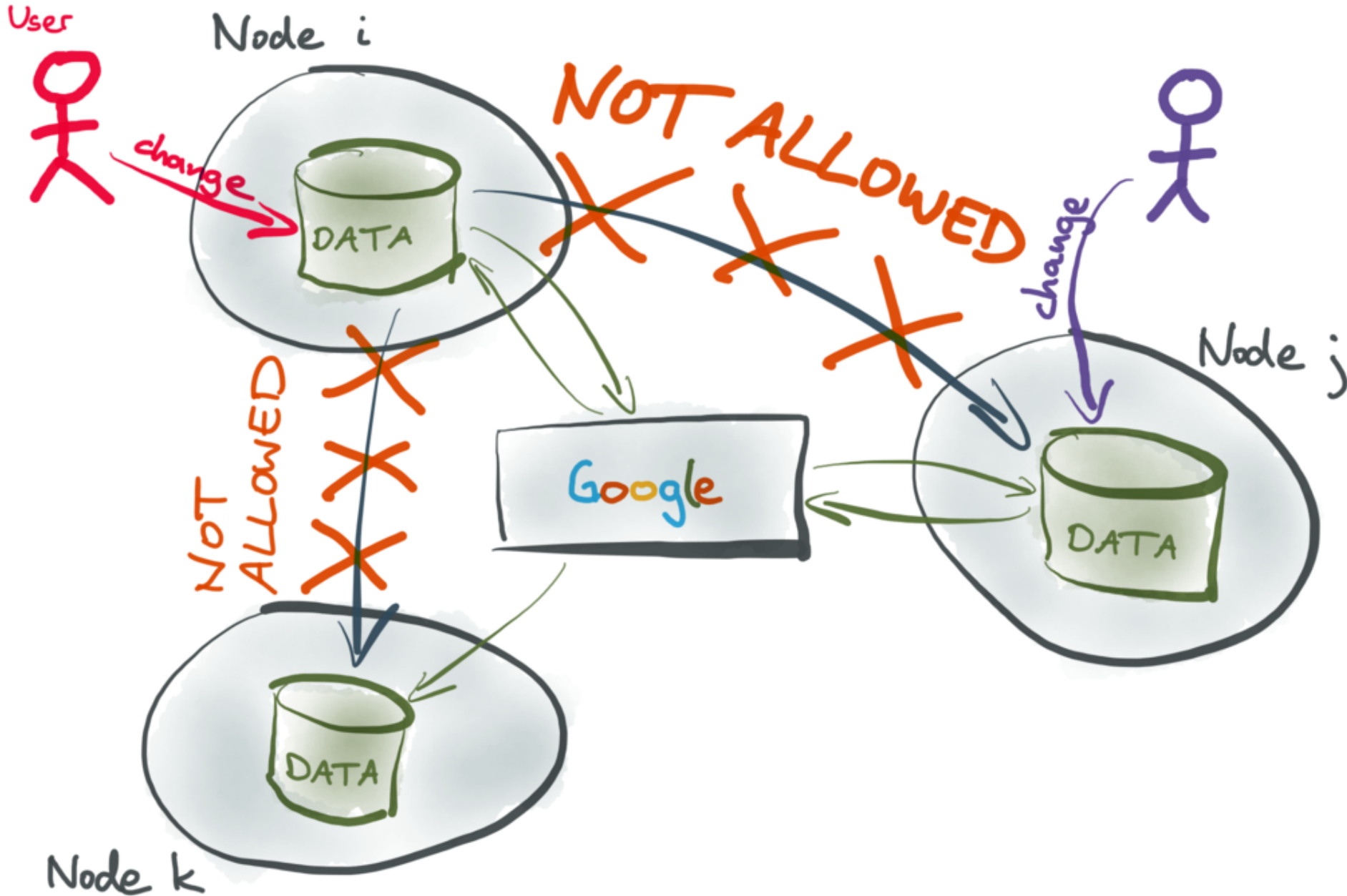
insert "!" at
position 4

insert "l" at
position 3

H	e	l	l	o	!
---	---	---	---	---	---

0 1 2 3 4 5





Algorithms for convergence

OPERATIONAL TRANSFORMATION (OT)

- e.g. Google Docs,
MS Office Online

1989-2006

CONFLICT-FREE REPLICATED DATA TYPES (CRDTs)

- e.g. Riak, TomTom GPS,
Teletype for Atom, ...

2006 - present



Convergence guarantee:

Any two nodes have seen the
same set of operations
(but maybe in a different order!)



They are in the same state

Convergence is good, but...

is the merged state the
state you wanted?

CRDTs are really easy to
implement... badly.

TODAY'S TOPICS

1. Interleaving anomalies in text editing
2. Moving (reordering) list items
3. Moving subtrees of a tree
4. Reducing metadata overhead of CRDTs

(Publications available, see references at the end)

1. Interleaving anomalies in collaborative text editors

(published at PaPoC 2019)

H	e	L	o
---	---	---	---

0.2 0.4 0.6 0.8

⇓ edit

H	e	L	L	o	!
---	---	---	---	---	---

0.2 0.4 0.6 0.7 0.8 0.9



⇓ edit



$$\{ (0.2, A, "H"), (0.4, A, "e"), \\ (0.6, A, "L"), (0.8, A, "o") \}$$

∪

$$\{ (0.7, A, "L"), (0.9, B, "!") \}$$

=

$$\{ (0.2, A, "H"), (0.4, A, "e"), \\ (0.6, A, "L"), (0.7, A, "L"), \\ (0.8, A, "o"), (0.9, B, "!") \}$$

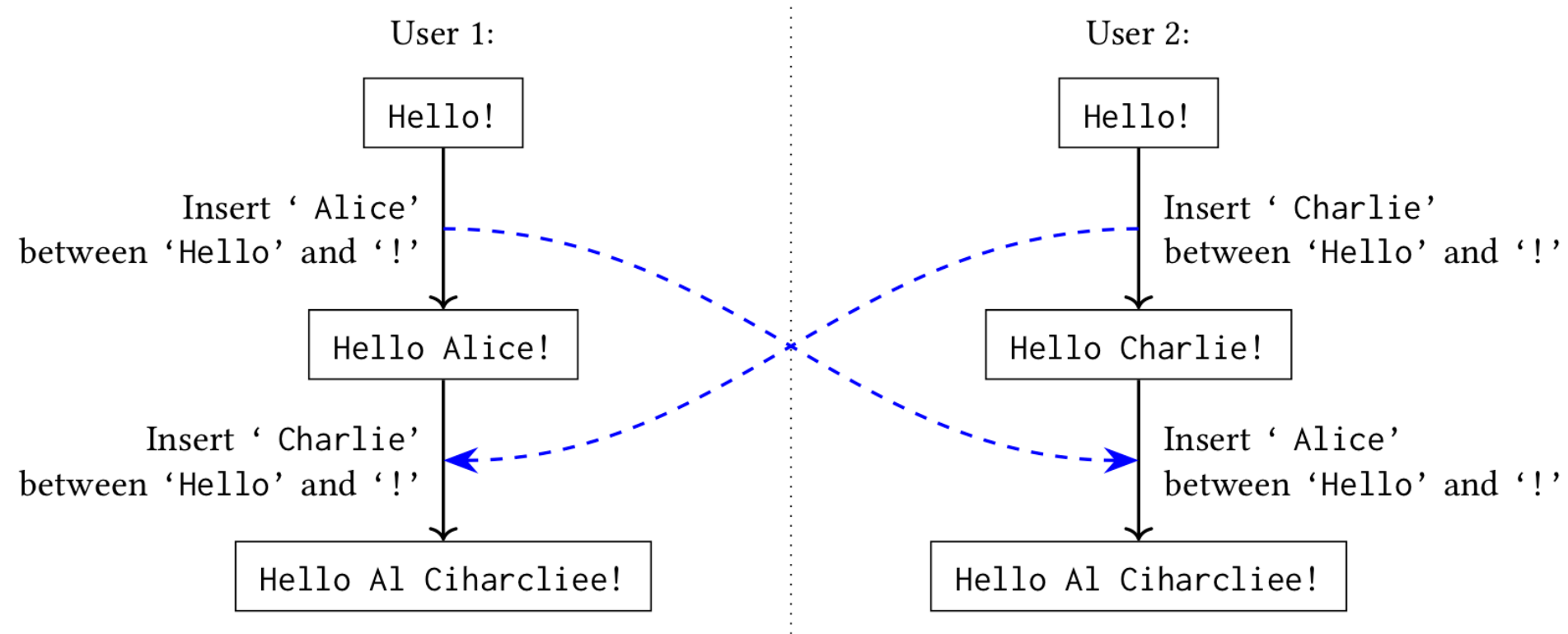


Figure 2: Two concurrent insertions at the same position are interleaved.

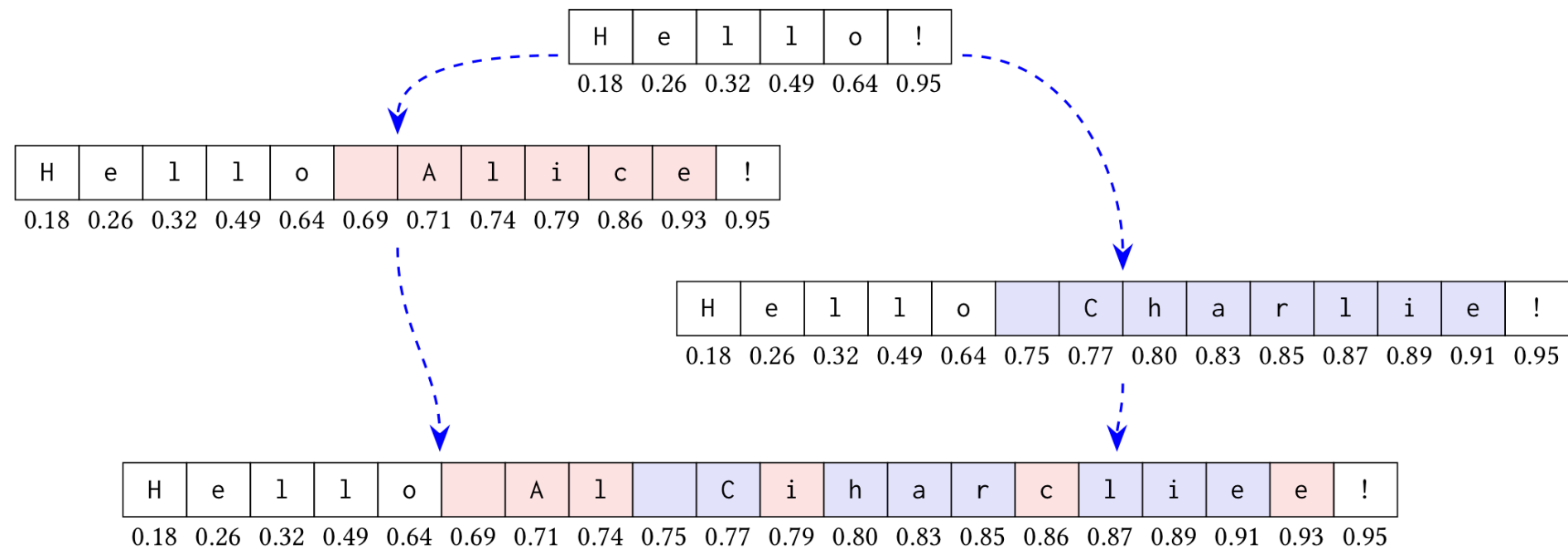


Figure 3: Interleaving due to character positions taken from a dense identifier set, e.g. the rational numbers \mathbb{Q} .

Text editing CRDTs

Logoot

(Weiss, Urso &
Molli 2009/2010)

LSEQ

(Nédelec, Molli, Mostefaoui &
Desmontils, 2013)

RGA

(Roh, Jeon, Kim &
Lee, 2011)

Treedoc

(Preguiça, Marquès,
Shapiro & Letia, 2009)

WOOT

(Oster, Urso, Molli &
Imine, 2006)

Astrong

(Attiya, Burckhardt, Gotsman,
Morrison, Yang & Zawirski, 2016)

Text editing CRDTs

Logoot

(Weiss, Urso &
Molli 2009/2010)

interleaving 😞

LSEQ

(Nédelec, Molli, Mostefaoui &
Desmontils, 2013)

interleaving 😞

RGA

(Roh, Jeon, Kim &
Lee, 2011)

lesser anomaly 😐

Treedoc

(Preguiça, Marquès,
Shapiro & Letia, 2009)

no interleaving 😊
(conjectured)

WOOT

(Oster, Urso, Molli &
Imine, 2006)

no interleaving 😊
(conjectured)

Astrong

(Attiya, Burckhardt, Gotsman,
Morrison, Yang & Zawirski, 2016)

interleaving 😞

RGA:

a lesser interleaving
anomaly

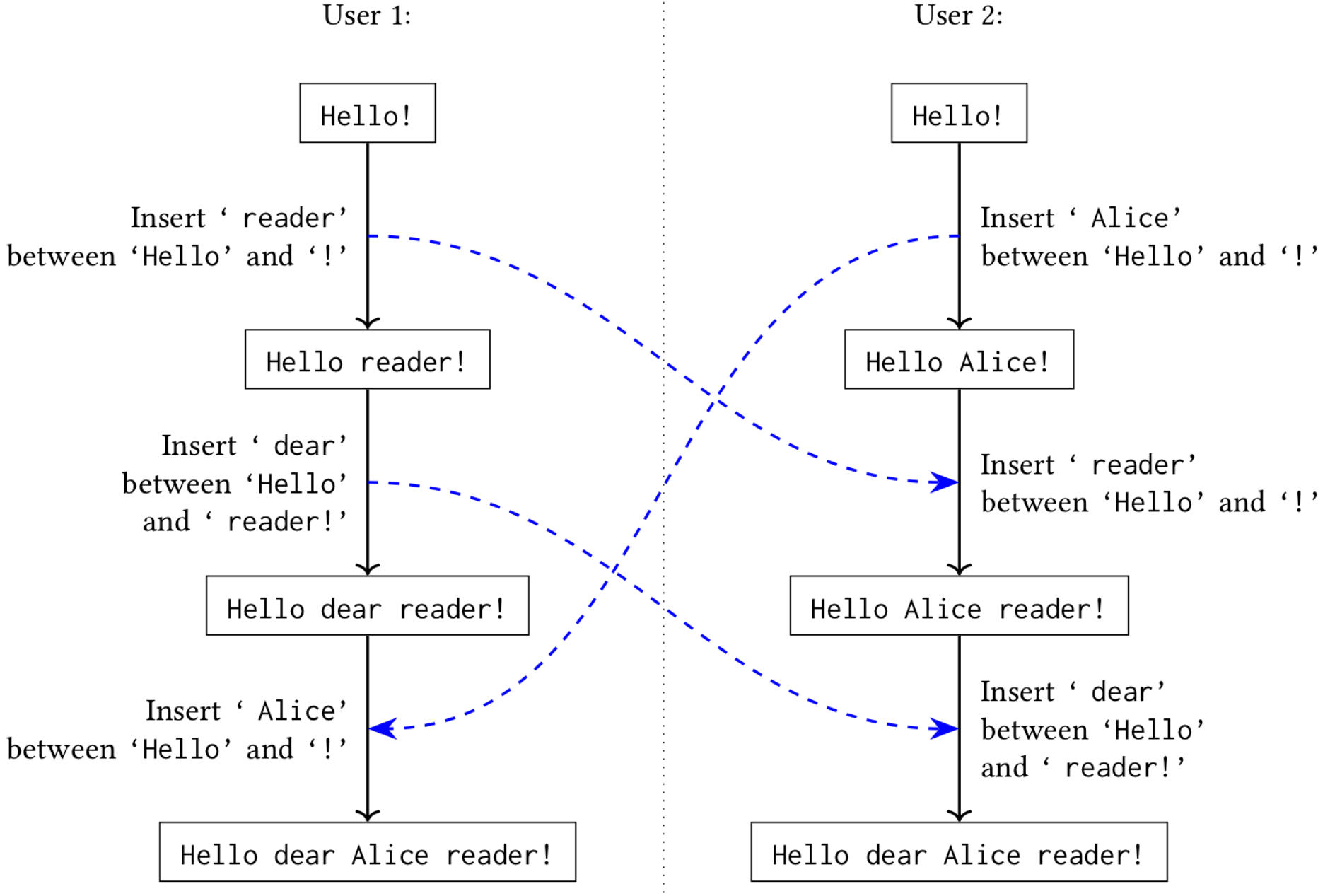
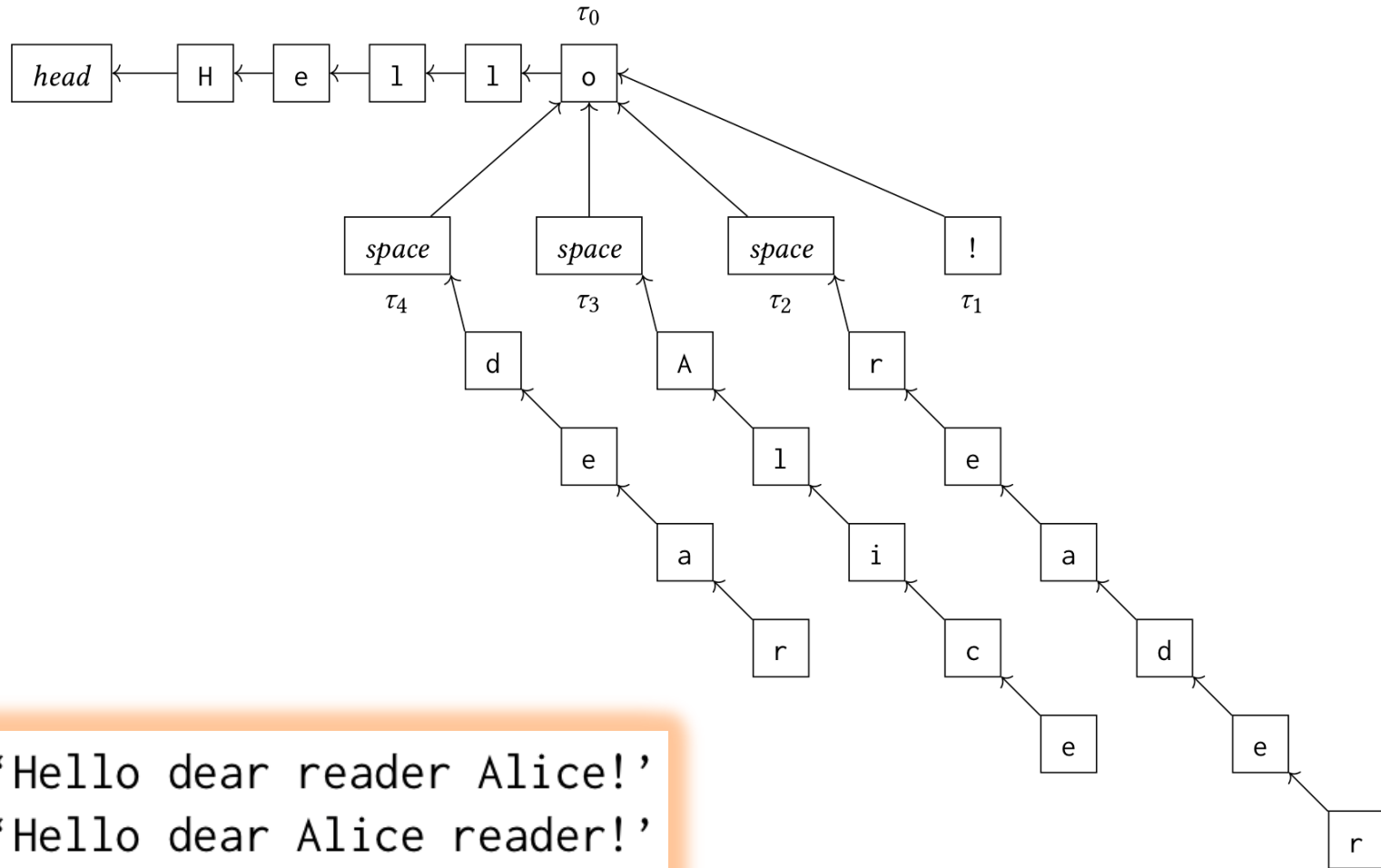
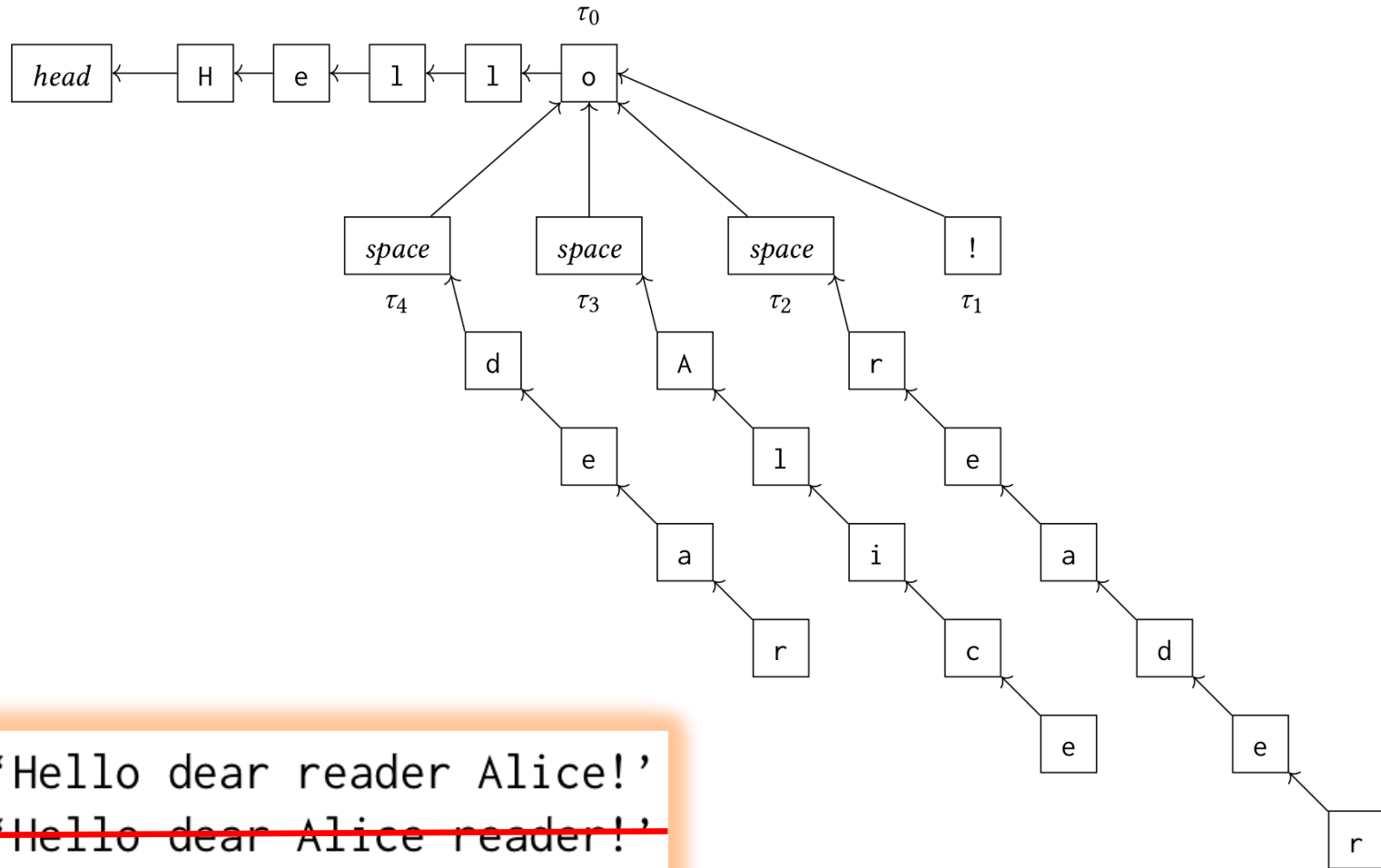


Figure 4: The lesser interleaving anomaly that can occur with RGA.



- (1) 'Hello dear reader Alice!'
- (2) 'Hello dear Alice reader!'
- (3) 'Hello Alice dear reader!'



- (1) 'Hello dear reader Alice!'
- ~~(2) 'Hello dear Alice reader!'~~
- (3) 'Hello Alice dear reader!'

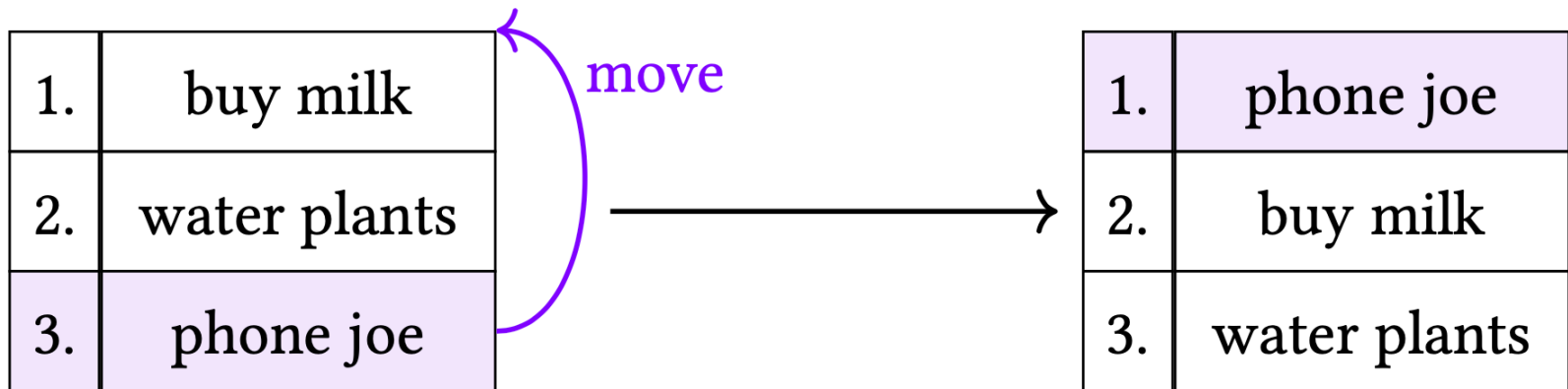
TODAY'S TOPICS

1. Interleaving anomalies in text editing
2. Moving (reordering) list items
3. Moving subtrees of a tree
4. Reducing metadata overhead of CRDTs

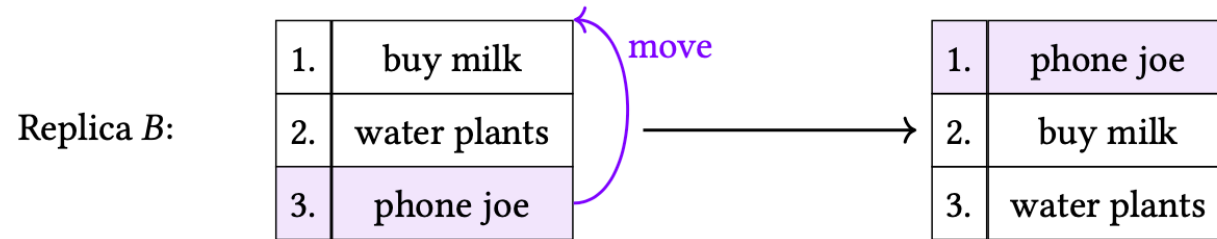
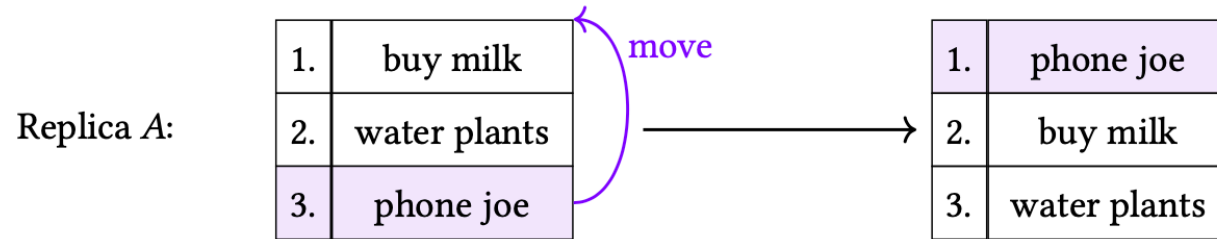
(Publications available, see references at the end)

2. Moving elements in List CRDTs

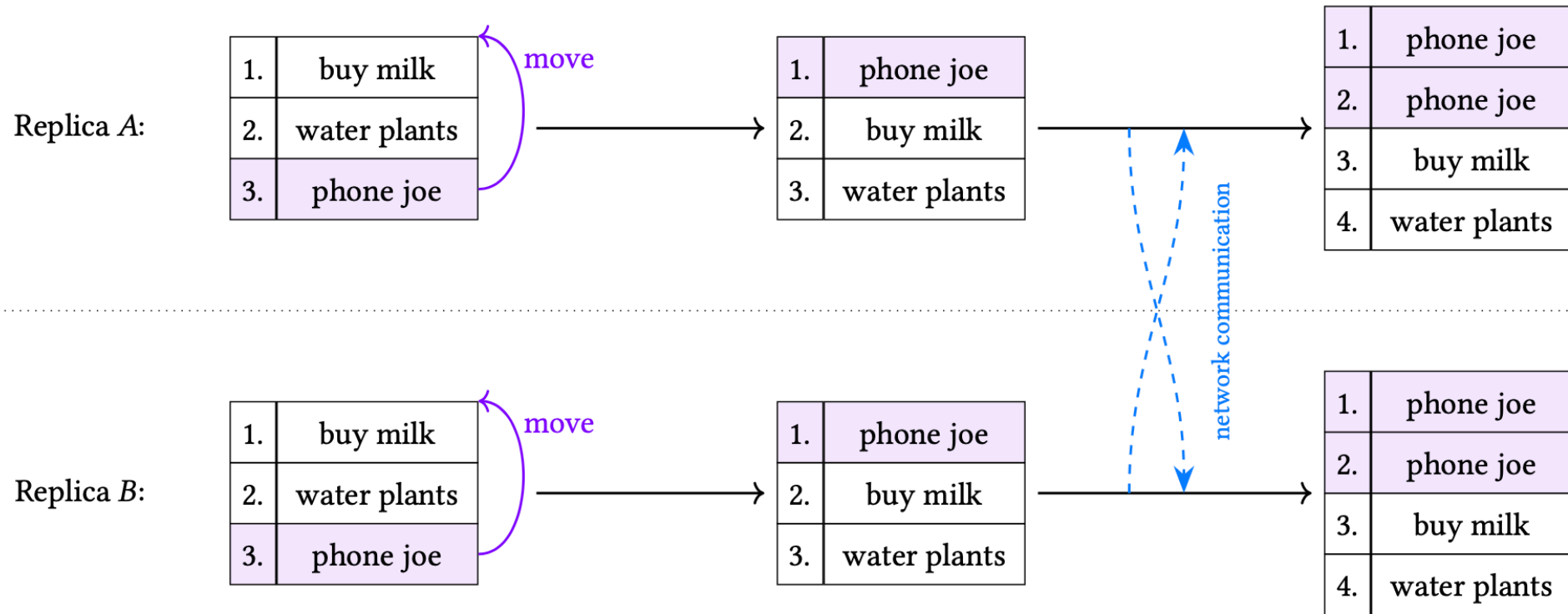
(published at PaPoC 2020)



List (sequence, array) CRDTs: WOOT, Treedoc, Logoot, RGA, Causal Trees, LSEQ, ...

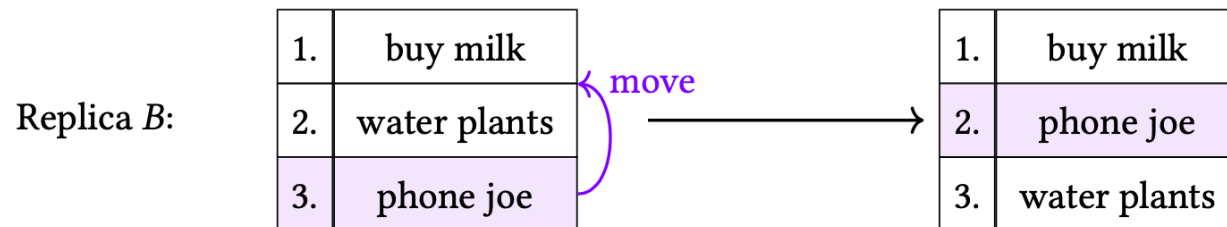
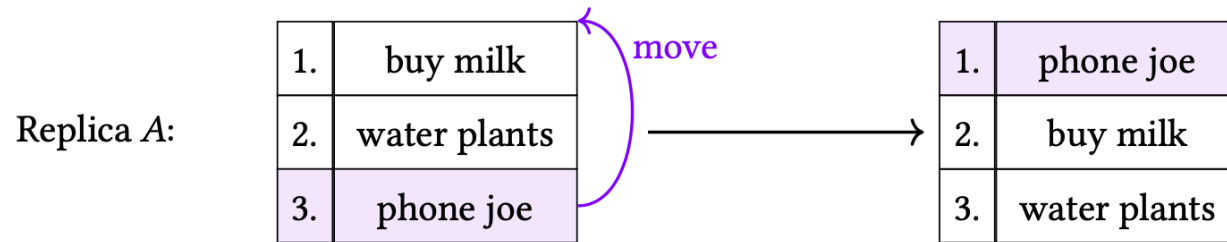


List (sequence, array) CRDTs: WOOT, Treedoc, Logoot, RGA, Causal Trees, LSEQ, ...

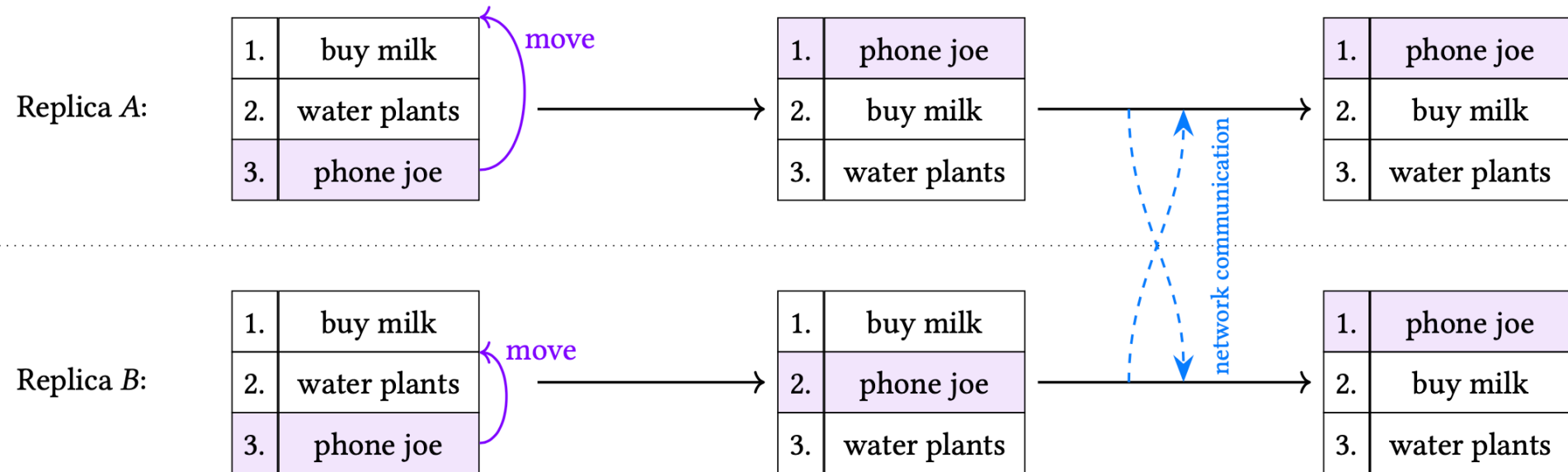


Emulating “move” as “delete-and-reinsert”:
concurrent moves of the same item
→ duplication 🤔 🤔 🤔

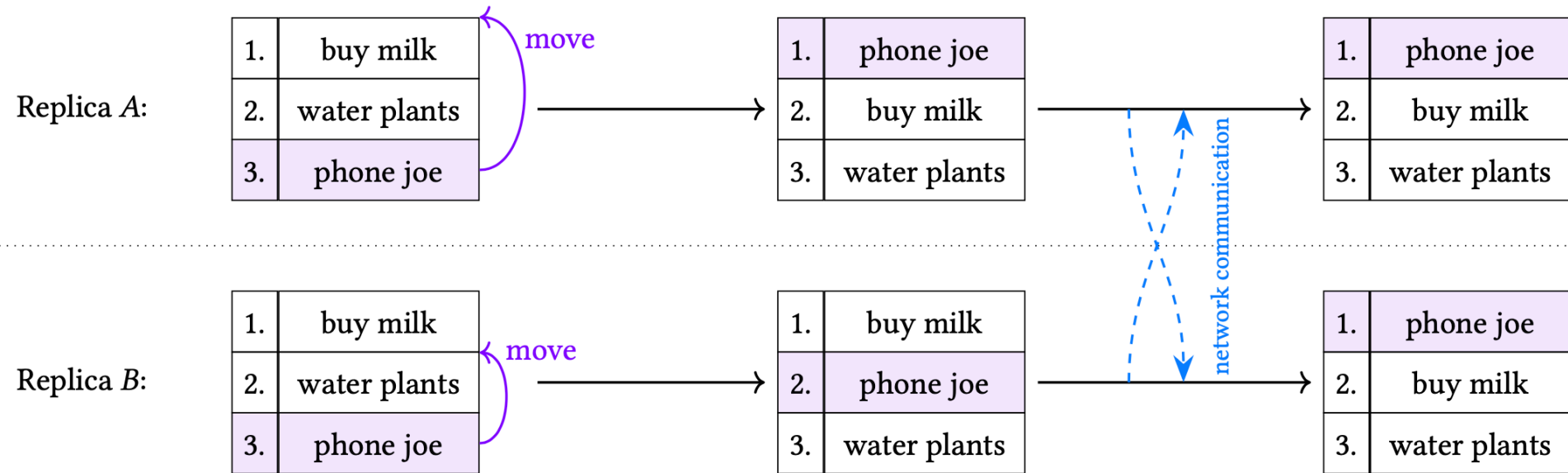
Concurrent move of the same item to different positions — what should happen?



Converge to **one** of the destination positions
(pick one arbitrarily but deterministically)



“pick one arbitrarily”
 = last-writer wins register!



$pos_{\text{phone joe}} := \text{“head of the list”}$

merge

$pos_{\text{phone joe}} == \text{“head of the list”}$

$pos_{\text{phone joe}} := \text{“after buy milk”}$

List CRDT with move operation

$pos_{\text{phone joe}} := \text{"after buy milk"}$

need one register per list item

need a stable way of referencing list positions

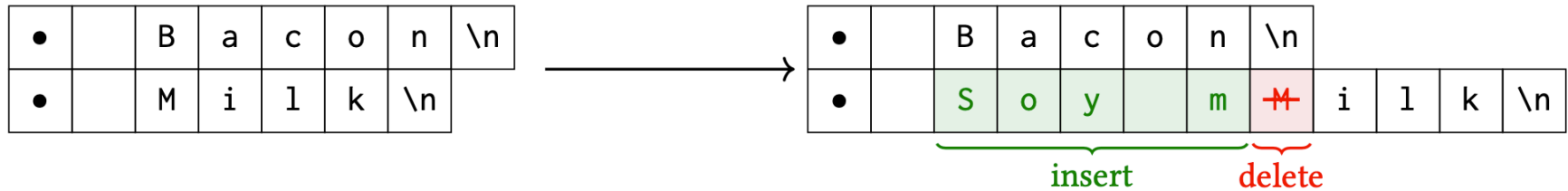
$state = \text{AWSet}(\{ (v_1, \text{LWWRegister}(p_1)), \\ (v_2, \text{LWWRegister}(p_2)), \\ \dots \})$

Treedoc: path through binary tree
Logoot: list of (integer, replicaID) pairs
RGA: s4vector
Causal Trees: logical timestamp
etc...

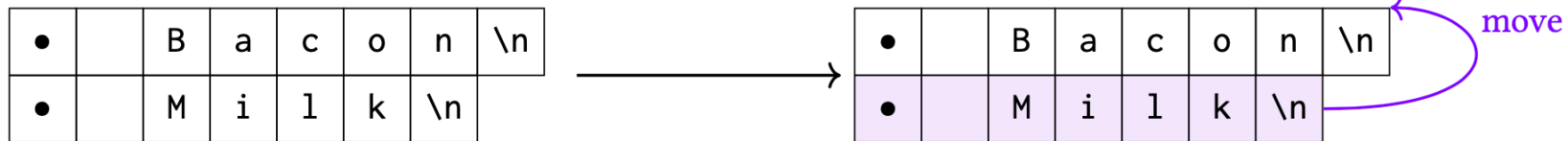
Composition of any list CRDT + AWSet +
LWWRegister = another CRDT 😎 🙌

Moving ranges of elements

Replica A:

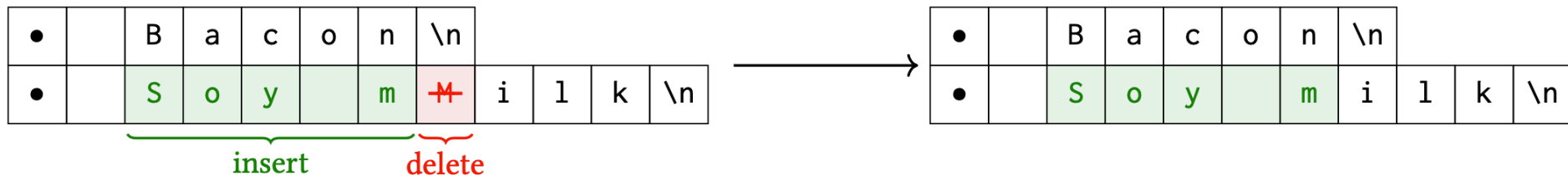


Replica B:

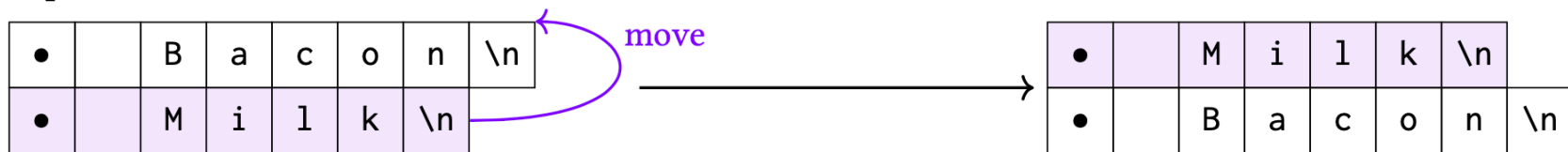


Moving ranges of elements

Replica A:



Replica B:



Desired outcome

Replica A:

•		B	a	c	o	n	\n					
•		S	o	y		m	M	i	l	k	\n	

insert delete

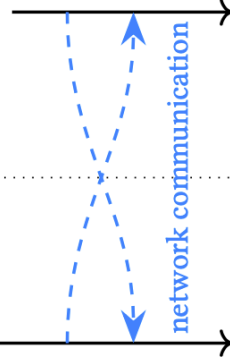
•		S	o	y		m	i	l	k	\n		
•		B	a	c	o	n	\n					

Replica B:

•		B	a	c	o	n	\n					
•		M	i	l	k	\n						

move

•		S	o	y		m	i	l	k	\n		
•		B	a	c	o	n	\n					



Actual outcome



Replica A:

•		B	a	c	o	n	\n						
•		S	o	y		m	+	i	l	k	\n		

insert delete

•		M	i	l	k	\n							
•		B	a	c	o	n	\n						
		S	o	y		m							

Replica B:

•		B	a	c	o	n	\n						
•		M	i	l	k	\n							

move

•		M	i	l	k	\n							
•		B	a	c	o	n	\n						
		S	o	y		m							

Fixing this: an open problem!

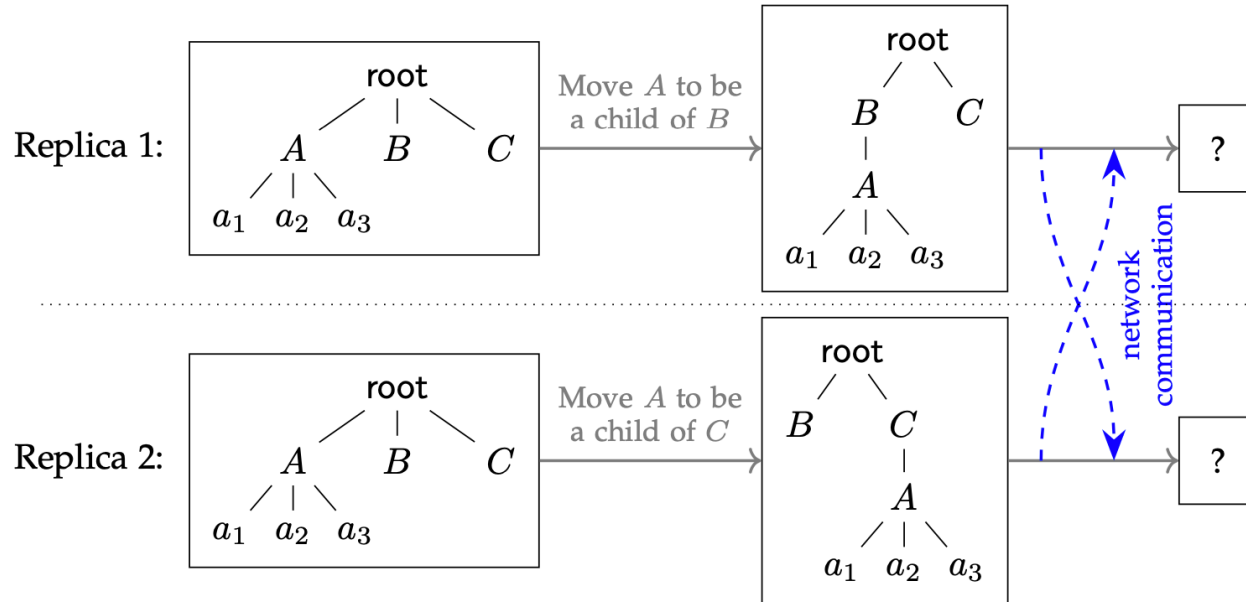
TODAY'S TOPICS

1. Interleaving anomalies in text editing
2. Moving (reordering) list items
3. Moving subtrees of a tree
4. Reducing metadata overhead of CRDTs

(Publications available, see references at the end)

3. A move operation for tree CRDTs

Concurrent moves of same node



Concurrent moves of same node

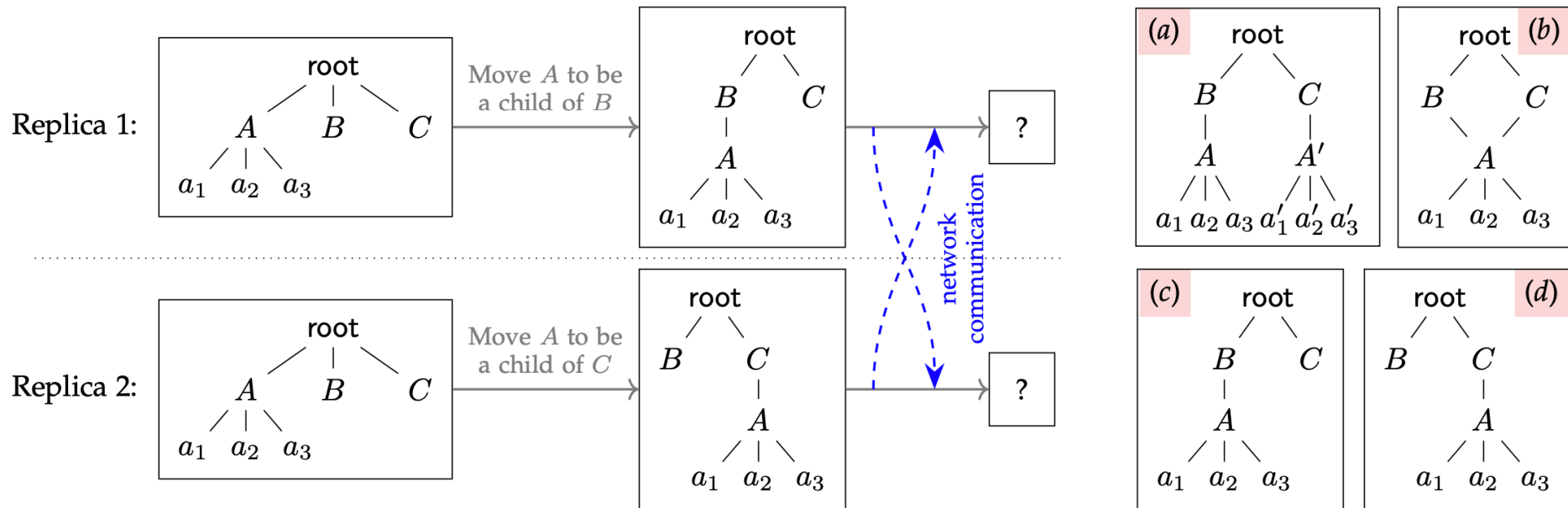


Fig. 1. Replica 1 moves A to be a child of B , while concurrently replica 2 moves the same node A to be a child of C . Boxes (a) to (d) show possible outcomes after the replicas have communicated and merged their states.

Do try this at home!

```
mkdir a
```

```
mkdir a/b
```

```
mv a a/b
```

Directory being moved is an ancestor of the destination.

Do try this at home!

```
mkdir a
```

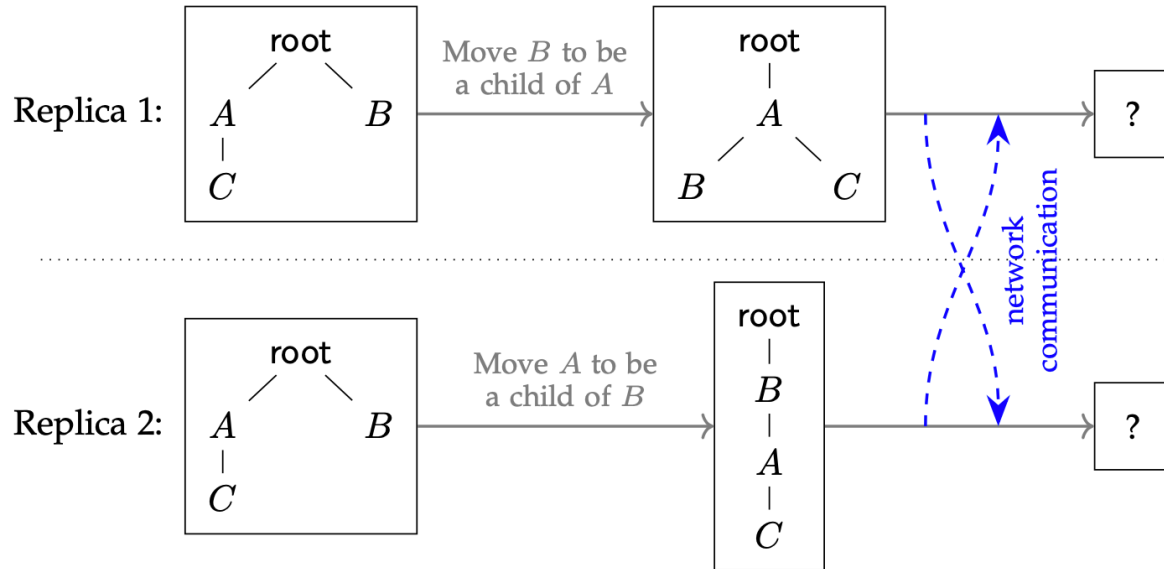
```
mkdir a/b
```

```
mv a a/b
```

Directory being moved is an ancestor of the destination.

"mv: rename a to a/b/a:
Invalid argument"

Moving A into B, and B into A



Moving A into B, and B into A

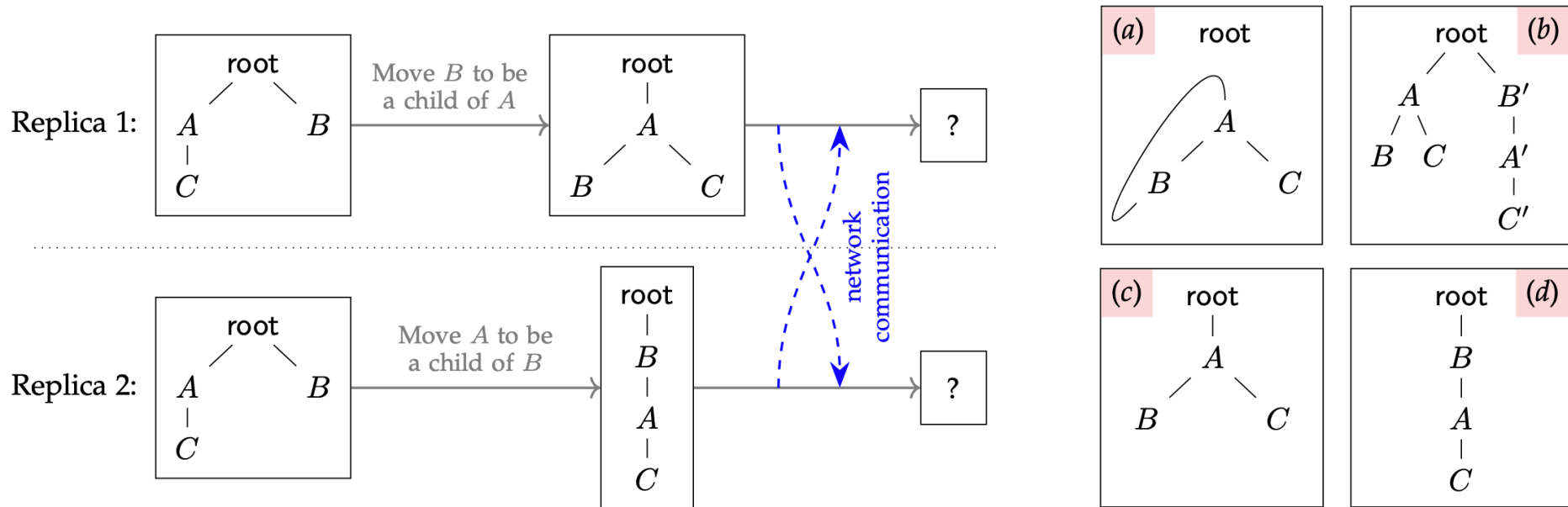


Fig. 2. Initially, nodes *A* and *B* are siblings. Replica 1 moves *B* to be a child of *A*, while concurrently replica 2 moves *A* to be a child of *B*. Boxes (a) to (d) show possible outcomes after the replicas have communicated and merged their states.

Backup and Sync

Can't sync 2 files:

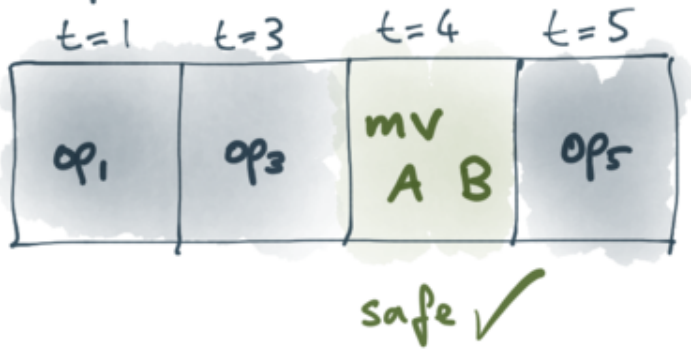
Can't remove files with changes pending. Try again after sync is complete.

📁 Download Error - My Drive/experiment/folder-b

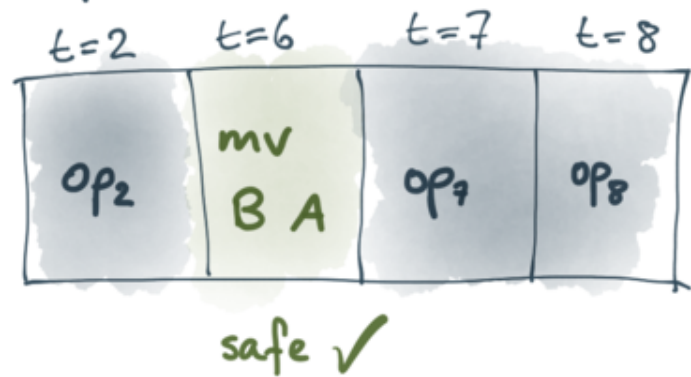
An unknown error occurred.

📁 Upload Error - /Users/martin/Google Drive/experiment/folder-a

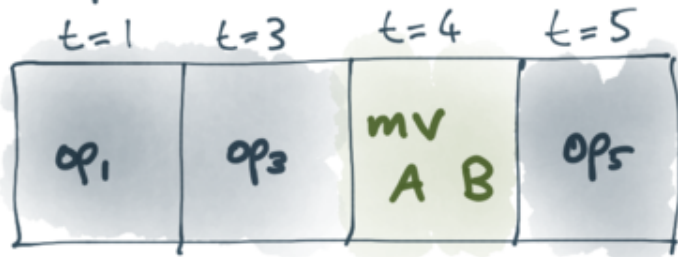
Replica 1:



Replica 2:



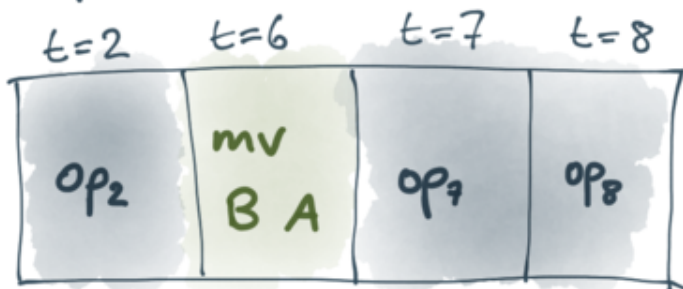
Replica 1:



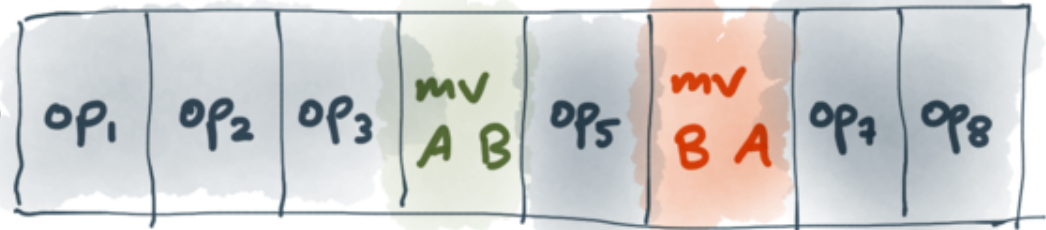
safe ✓

merge operation
sequences in
timestamp order

Replica 2:



safe ✓



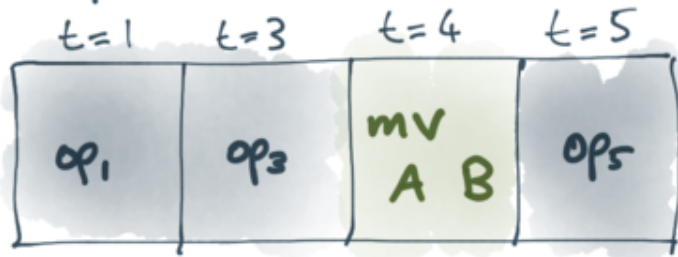
safe ✓

unsafe



skip this operation!

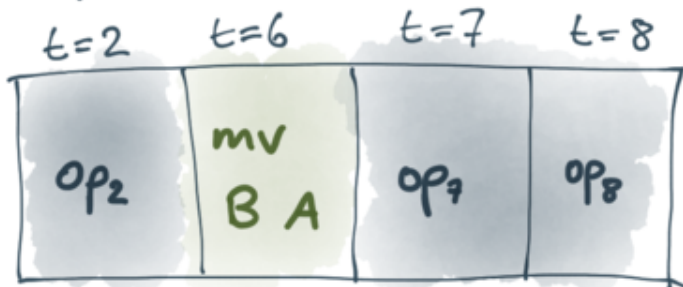
Replica 1:



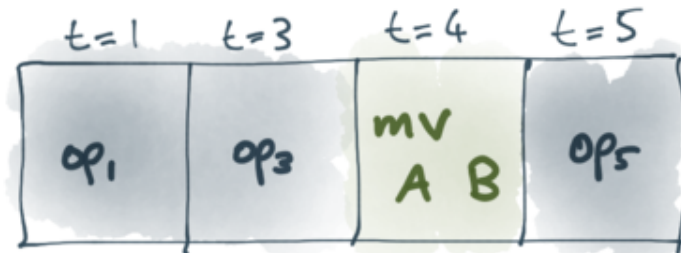
safe ✓

merge operation
sequences in
timestamp order

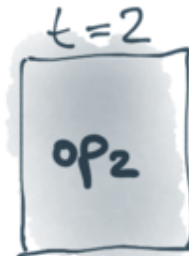
Replica 2:



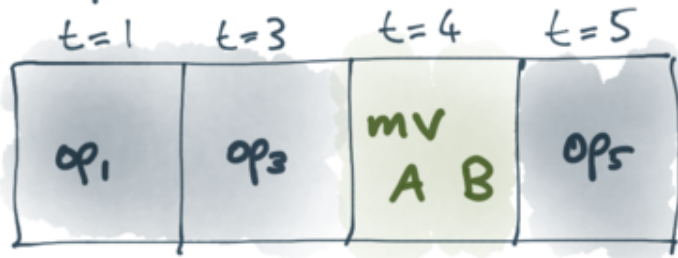
safe ✓



insert
operation
here



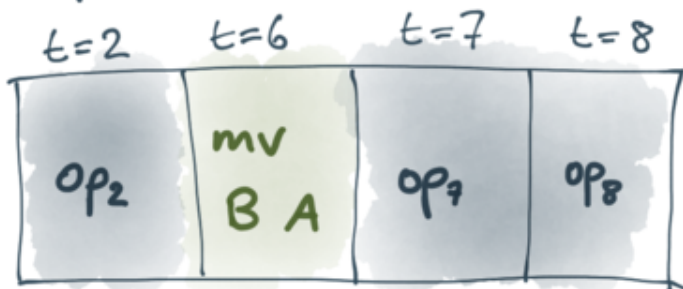
Replica 1:



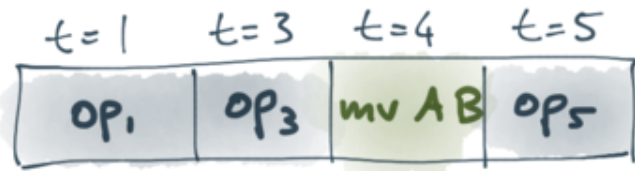
safe ✓

merge operation sequences in timestamp order

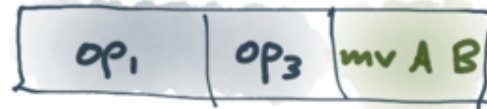
Replica 2:



safe ✓



undo op₅



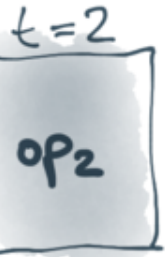
undo "mv A B"



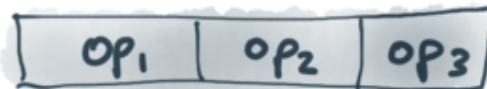
undo op₃



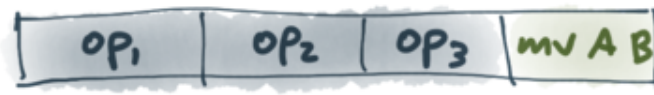
apply op₂



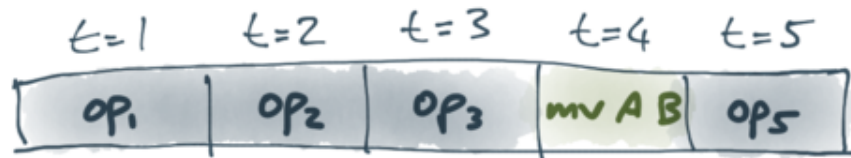
redo op₃



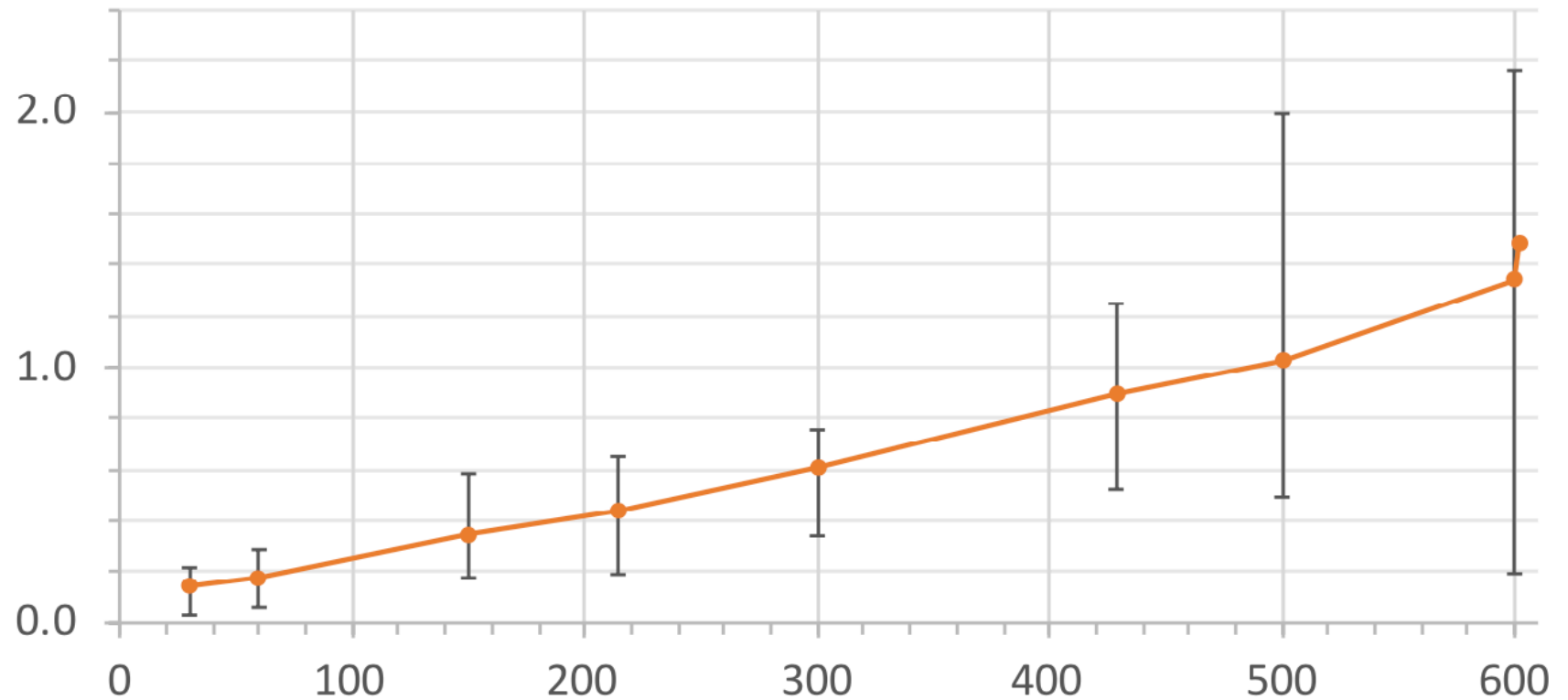
redo "mv A B"



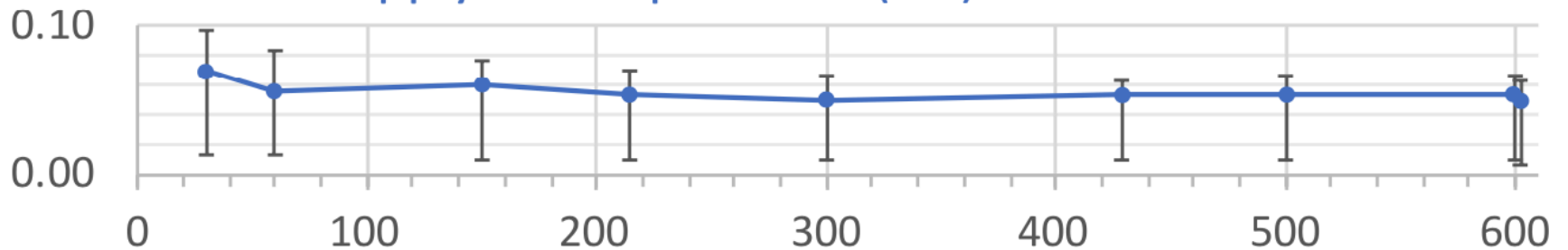
redo op₅



Time to apply remote operation (ms)



Time to apply local operation (ms)



Move operations per second

struct MoveOp {

Timestamp time; ← globally unique (e.g. Lamport timestamp)

NodeID parent; ← destination of move

Metadata meta; ← e.g. filename within parent directory

NodeID child; ← subtree being moved

}

struct MoveOp {

Timestamp time; ← globally unique (e.g. Lamport timestamp)

NodeID parent; ← destination of move

Metadata meta; ← e.g. filename within parent directory

NodeID child; ← subtree being moved

}

struct LogEntry {

Timestamp time; ← from the move operation

Option <NodeID> oldParent; } empty if child was previously
Option <Metadata> oldMeta; not in the tree

NodeID newParent;

Metadata newMeta;

NodeID child;

} from the move operation

}

Tree is set of (parent, meta, child) triples.

$$\text{ancestor}(a, b) \equiv \left(\exists m. (a, m, b) \in \text{tree} \right) \text{ OR } \left(\exists c, m. (a, m, c) \in \text{tree} \text{ AND } \text{ancestor}(c, b) \right)$$

Tree is set of (parent, meta, child) triples.

$$\text{ancestor}(a, b) \equiv \left(\exists m. (a, m, b) \in \text{tree} \right) \text{ OR } \left(\exists c, m. (a, m, c) \in \text{tree} \text{ AND } \text{ancestor}(c, b) \right)$$

doMove (move, tree) =

if ancestor(move.child, move.parent)

OR move.child = move.parent

then do nothing

else tree' = $\{ (p, m, c) \in \text{tree} \mid c \neq \text{move.child} \}$
 $\cup \{ (\text{move.parent}, \text{move.meta}, \text{move.child}) \}$

Theorems:

- every tree node has a unique parent

$$\forall p_1, p_2, m_1, m_2, c. (p_1, m_1, c) \in \text{tree} \wedge (p_2, m_2, c) \in \text{tree} \Rightarrow p_1 = p_2 \wedge m_1 = m_2$$

- the tree contains no cycles

$$\nexists a. \text{ancestor}(a, a)$$

- it's a CRDT

$$\text{applyOps}(\text{ops}_1) = \text{applyOps}(\text{ops}_2) \quad \text{if } \text{ops}_1 \text{ is a permutation of } \text{ops}_2$$

i.e. applying operations is commutative

TODAY'S TOPICS

1. Interleaving anomalies in text editing
2. Moving (reordering) list items
3. Moving subtrees of a tree
4. Reducing metadata overhead of CRDTs

(Publications available, see references at the end)

4. Reducing metadata overhead

Making CRDTs more efficient

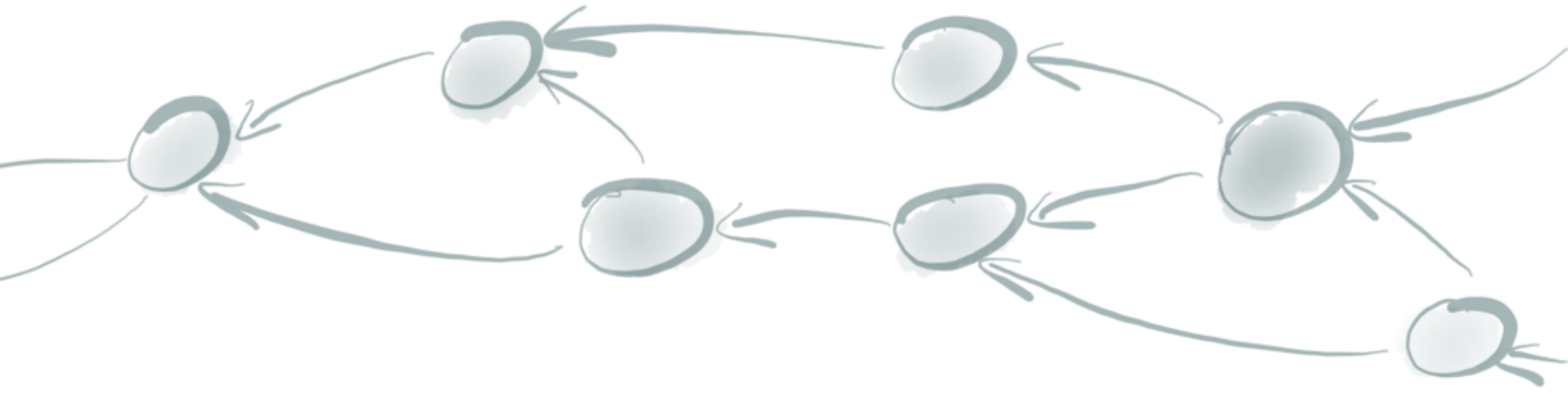


$\{ (0.2, A, "H"), (0.4, A, "e"),$
 $(0.6, A, "L"), (0.7, A, "L"),$
 $(0.8, A, "o"), (0.9, B, "!") \}$

actually a path
through a tree...
a few dozen bytes?

a UUID...
16 bytes binary /
36 bytes hex?

the actual ASCII /
Unicode character:
1 byte




Automerge

Ink & Switch

<https://github.com/automerge/automerge>

Compressing CRDT metadata in Automerge

Benchmark data: keystroke-by-keystroke editing trace of a text file (LaTeX source of a research paper) containing 182,315 single-character insertions, 77,463 single-character deletions, and 102,049 cursor movements.

	File size	File size (gzipped)	
Full document history, JSON format	146,406,415 bytes	6,132,895 bytes	 211 x
Full document history, custom binary format	695,298 bytes	302,067 bytes	
Document history with cursor movements omitted	570,992 bytes	214,889 bytes	
CRDT document with editing history omitted	228,153 bytes	114,821 bytes	
CRDT document with tombstones removed	154,418 bytes	63,249 bytes	
Baseline: plain text with no CRDT metadata	104,852 bytes	27,569 bytes	

EFFICIENTLY STORING EDIT HISTORY

- Store set of all insertion operations
(If char is deleted, mark it as such)
- Each operation has ID: Lamport timestamp
(pair of counter and node ID / "actor ID")
- Each insertion references ID of predecessor
(like in RGA)
- Store operations in document order
(The above is for text, but it easily generalises to JSON trees)

COLUMNAR ENCODING (simplified)

operation ID		reference element ID		inserted character		deleted by opID	
counter	actor	counter	actor	length	UTF-8	counter	actor
1	A	—	—	1	"H"	—	—
2	A	1	A	1	"e"	—	—
3	A	2	A	1	"l"	—	—
4	A	3	A	1	"l"	—	—
5	A	4	A	1	"l"	7	B
6	A	5	A	1	"o"	—	—

COLUMNAR ENCODING (simplified)

operation ID		reference element ID		inserted character		deleted by opID	
counter	actor	counter	actor	length	UTF-8	counter	actor
1	A	—	—	1	"H"	—	—
2	A	1	A	1	"e"	—	—
3	A	2	A	1	"l"	—	—
4	A	3	A	1	"l"	—	—
5	A	4	A	1	"l"	7	B
6	A	5	A	1	"o"	—	—

→ 1, 2, 3, 4, 5, 6

delta-encode to 1, 1, 1, 1, 1, 1

run-length encode to (6, 1)

LEB128 encodes this in 2 bytes

COLUMNAR ENCODING (simplified)

operation ID		reference element ID		inserted character		deleted by opID	
counter	actor	counter	actor	length	UTF-8	counter	actor
1	A	—	—	1	"H"	—	—
2	A	1	A	1	"e"	—	—
3	A	2	A	1	"l"	—	—
4	A	3	A	1	"l"	—	—
5	A	4	A	1	"l"	7	B
6	A	5	A	1	"o"	—	—

→ make a lookup table: $\{"A": 0, "B": 1\}$
→ 0, 0, 0, 0, 0, 0
→ run-length encode to (6, 0)
→ LEB128 encodes in 2 bytes

COLUMNAR ENCODING (simplified)

operation ID		reference element ID		inserted character		deleted by opID	
counter	actor	counter	actor	length	UTF-8	counter	actor
1	A	—	—	1	"H"	—	—
2	A	1	A	1	"e"	—	—
3	A	2	A	1	"l"	—	—
4	A	3	A	1	"l"	—	—
5	A	4	A	1	"l"	7	B
6	A	5	A	1	"o"	—	—

just concatenate the UTF-8
byte sequences → "Hello" (6 bytes)
(use length column to separate again)

COLUMNAR ENCODING (simplified)

operation ID		reference element ID		inserted character		deleted by opID	
counter	actor	counter	actor	length	UTF-8	counter	actor
1	A	—	—	1	"H"	—	—
2	A	1	A	1	"e"	—	—
3	A	2	A	1	"l"	—	—
4	A	3	A	1	"l"	—	—
5	A	4	A	1	"l"	7	B
6	A	5	A	1	"o"	—	—

Plus some additional metadata (e.g. timestamp and range of opID counter values for each change)
⇒ can reconstruct any past document state

CRDTs are really easy to **implement badly**:

1. Interleaving anomalies in text editing
2. Moving (reordering) list items
3. Moving subtrees of a tree
4. Reducing metadata overhead of CRDTs

... but research is making progress! 😊

Text editing CRDTs:

Logoot: Stéphane Weiss, Pascal Urso, and Pascal Molli: “Logoot: A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks,” ICDCS 2009.

LSEQ: Brice Nédelec, Pascal Molli, Achour Mostefaoui, and Emmanuel Desmontils: “LSEQ: an Adaptive Structure for Sequences in Distributed Collaborative Editing,” DocEng 2013.

RGA: Hyun-Gul Roh, Myeongjae Jeon, Jin-Soo Kim, and Joonwon Lee: “Replicated abstract data types: Building blocks for collaborative applications,” Journal of Parallel and Distributed Computing, 71(3):354–368, 2011.

Treedoc: Nuno Preguiça, Joan Manuel Marques, Marc Shapiro, and Mihai Letia: “A Commutative Replicated Data Type for Cooperative Editing,” ICDCS 2009.

WOOT: Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine: “Data consistency for P2P collaborative editing,” CSCW 2006.

A_{strong}: Hagit Attiya, Sebastian Burckhardt, Alexey Gotsman, Adam Morrison, Hongseok Yang, and Marek Zawirski: “Specification and Complexity of Collaborative Text Editing,” PODC 2016.

More details in these related publications:

Interleaving anomaly: Martin Kleppmann, Victor B. F. Gomes, Dominic P. Mulligan, and Alastair R. Beresford: “Interleaving anomalies in collaborative text editors”. PaPoC 2019.

Proof of no interleaving in RGA: Martin Kleppmann, Victor B F Gomes, Dominic P Mulligan, and Alastair R Beresford: “OpSets: Sequential Specifications for Replicated Datatypes,” <https://arxiv.org/abs/1805.04263>, May 2018.

Moving list items: Martin Kleppmann: “Moving Elements in List CRDTs”. PaPoC 2020.

Move operation in CRDT trees: Martin Kleppmann, Dominic P. Mulligan, Victor B. F. Gomes, and Alastair R. Beresford: “A highly-available move operation for replicated trees and distributed filesystems”. Preprint, <https://martin.kleppmann.com/papers/move-op.pdf>

Reducing metadata overhead: Martin Kleppmann: “Experiment: columnar data encoding for Automerge”, 2019. <https://github.com/automerge/automerge-perf/blob/master/columnar/README.md>

Local-first software: Martin Kleppmann, Adam Wiggins, Peter van Hardenberg, and Mark McGranaghan: “Local-first software: You own your data, in spite of the cloud”. Onward! 2019. <https://www.inkandswitch.com/local-first.html>

Thanks!

- Martin's email: mk428@cl.cam.ac.uk
- Martin on Twitter: <https://twitter.com/martinkl>
- Martin's book: <https://dataintensive.net/>
- CRDT resources: <https://crdt.tech/>
- Automerge: <https://github.com/automerge/automerge>

Thank you to these organisations for supporting this work!



Ink & Switch

LEVERHULME
TRUST

