Algorithms for collaborative text editing

MARTIN KLEPPMANN

until Dec 2023
TU Munich

from Jan 2024
University of Cambridge

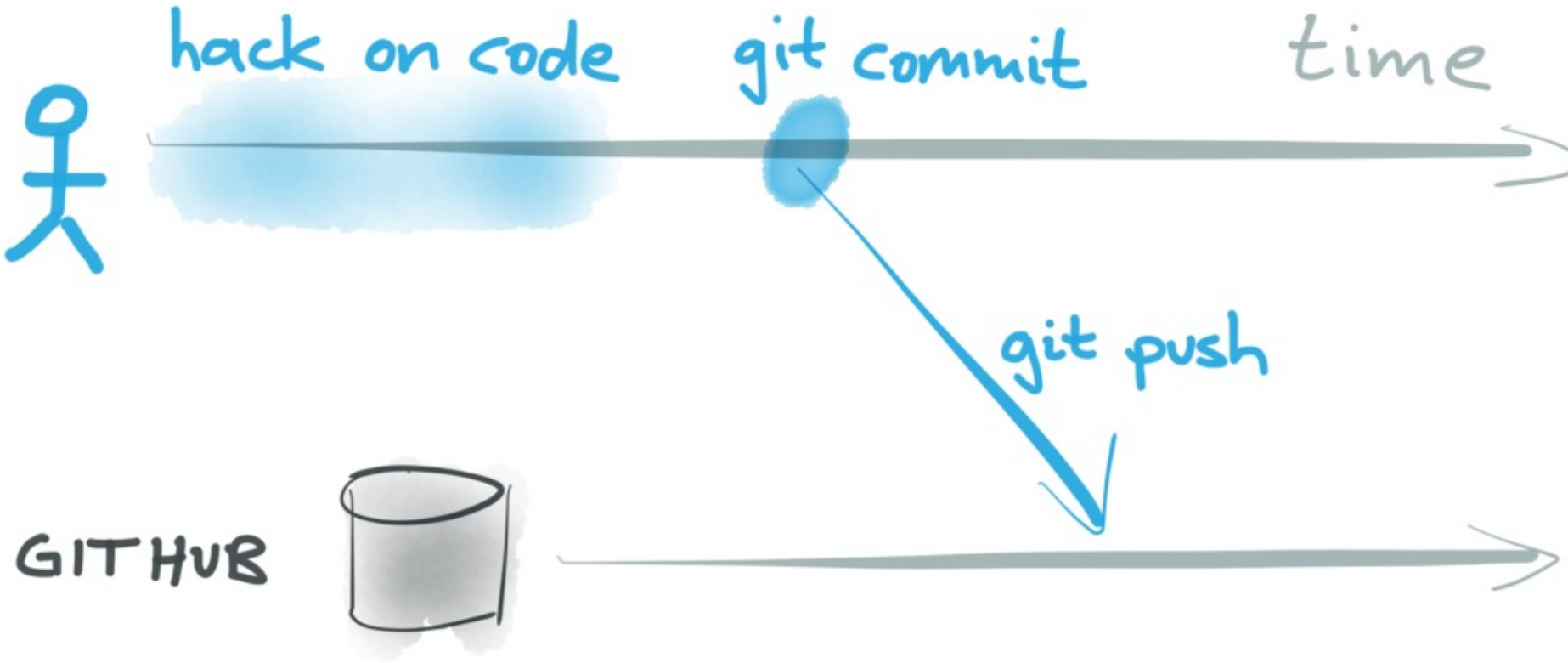
email martin@kleppmann.com
web https://martin.kleppmann.com
bluesky @martin.kleppmann.com
mastodon @martin@nondeterministic.computer

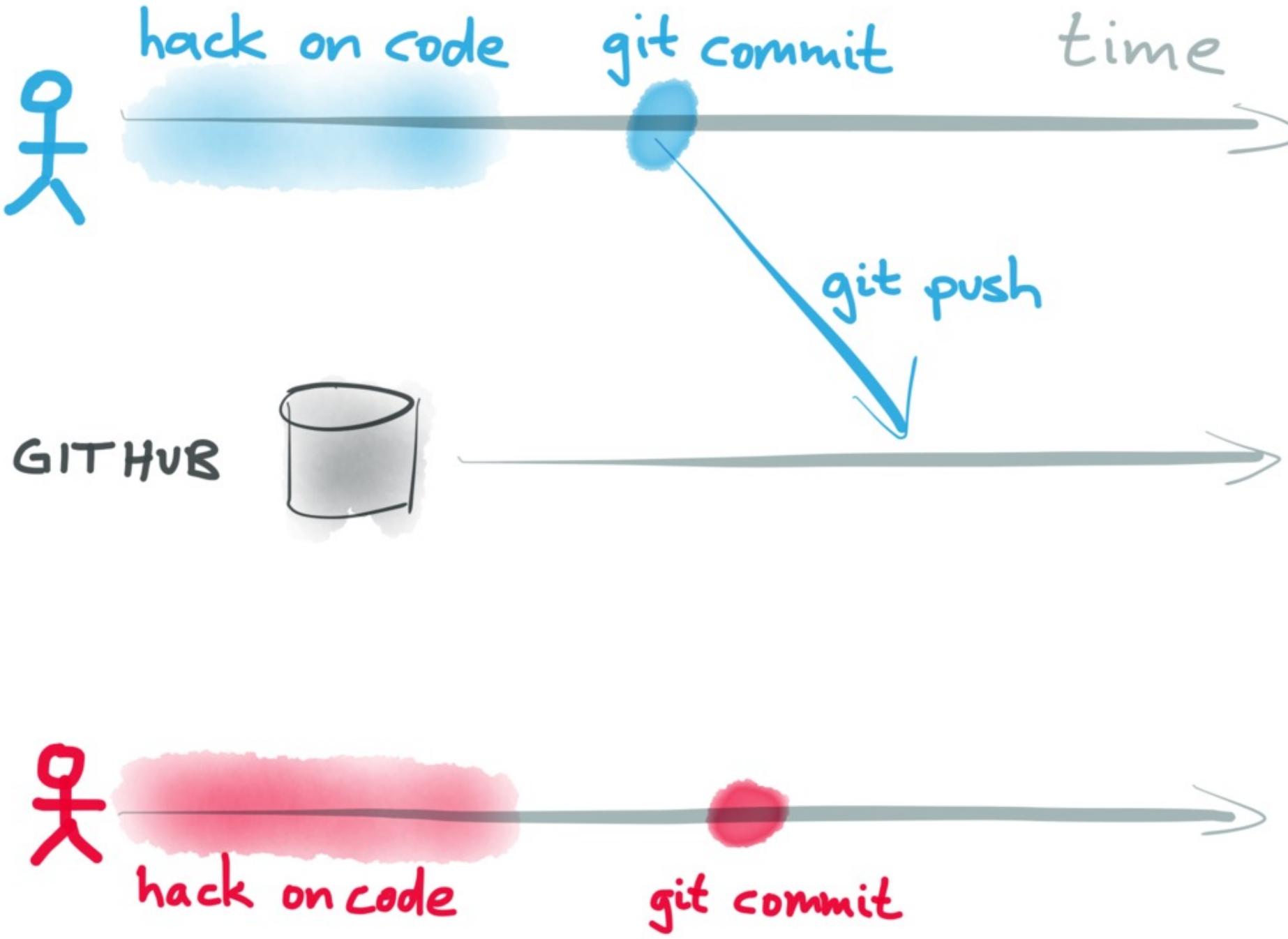
VolkswagenStiftung

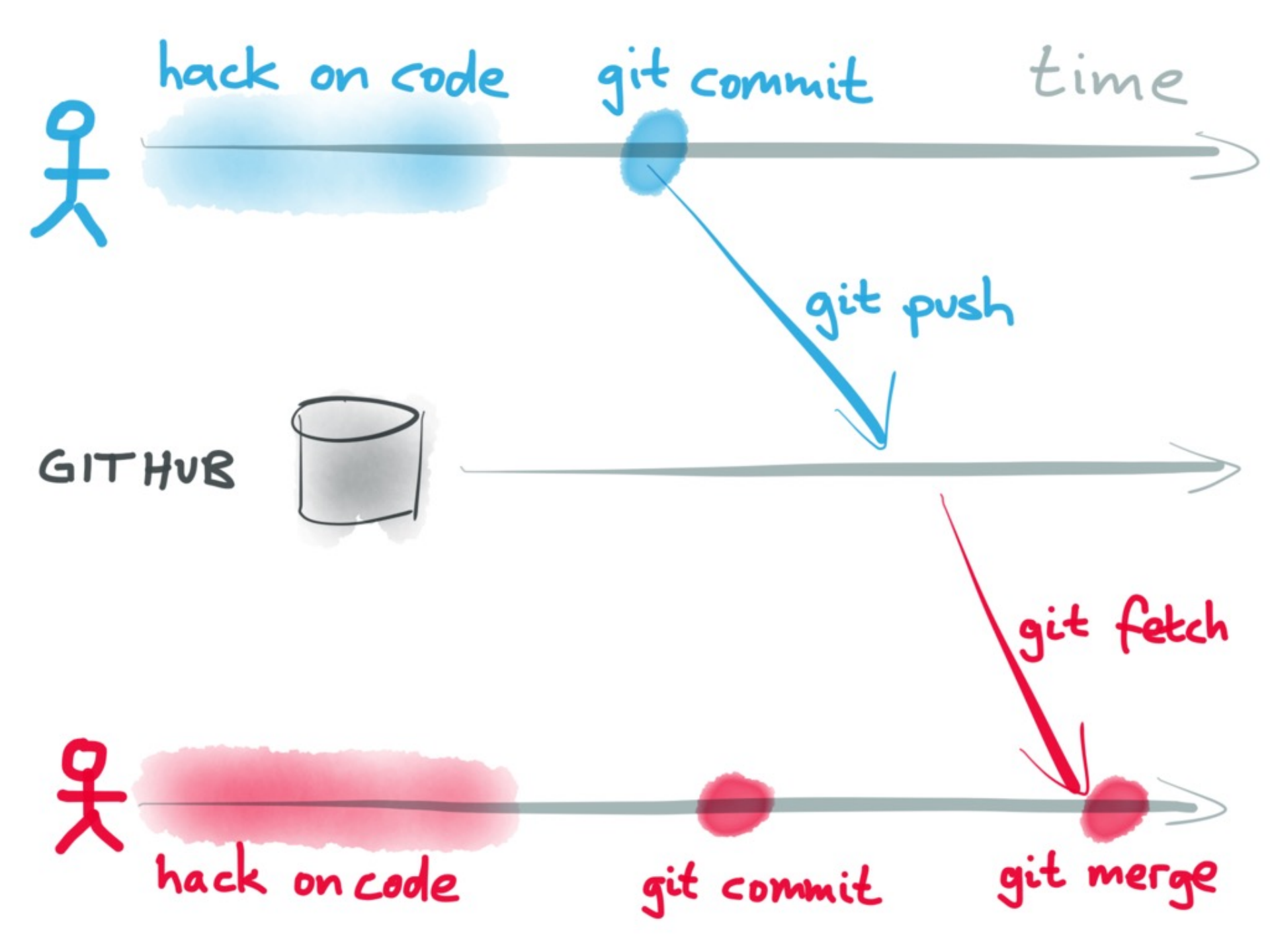
Ink & Switch

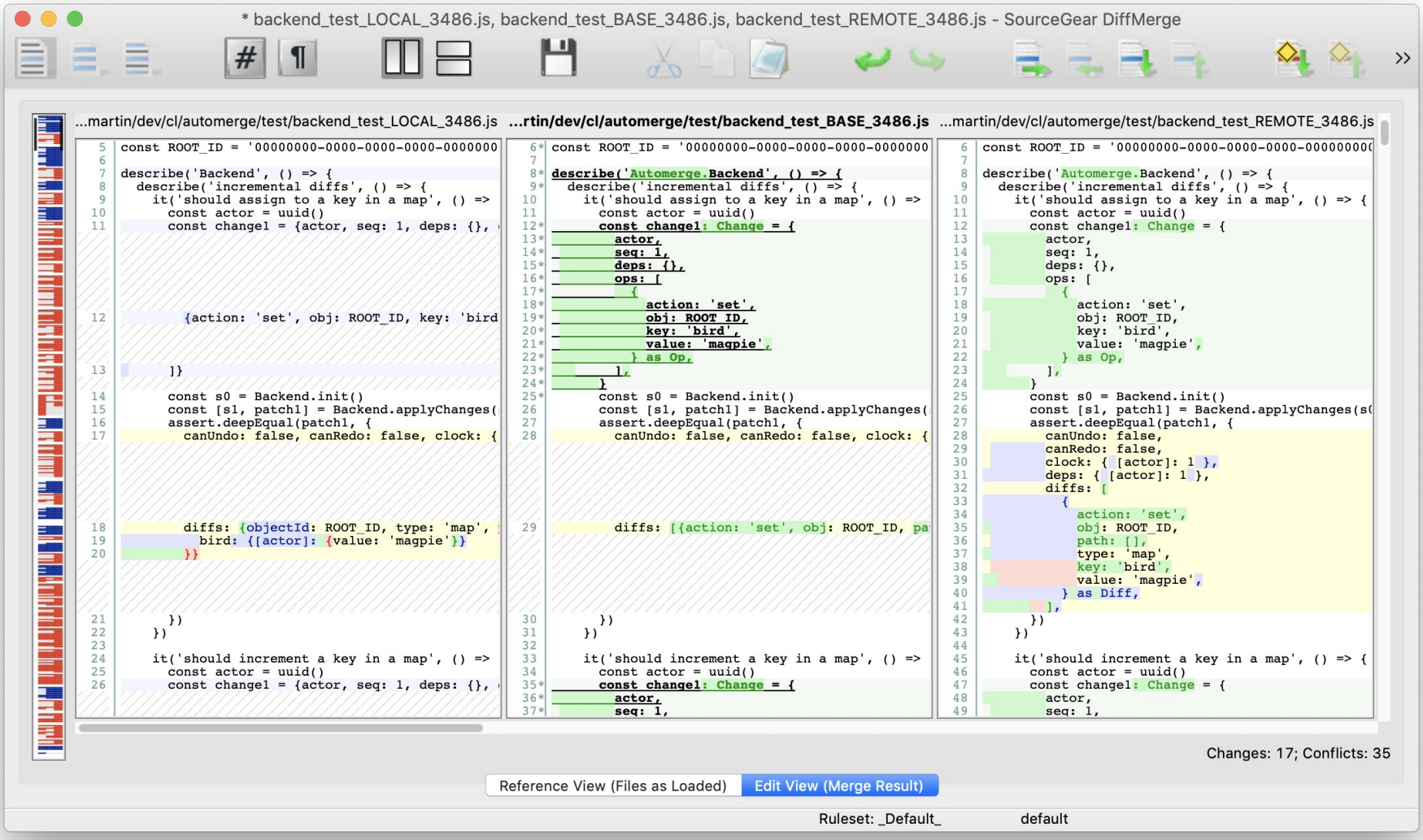

hack on code
time


hack on code
git commit
time

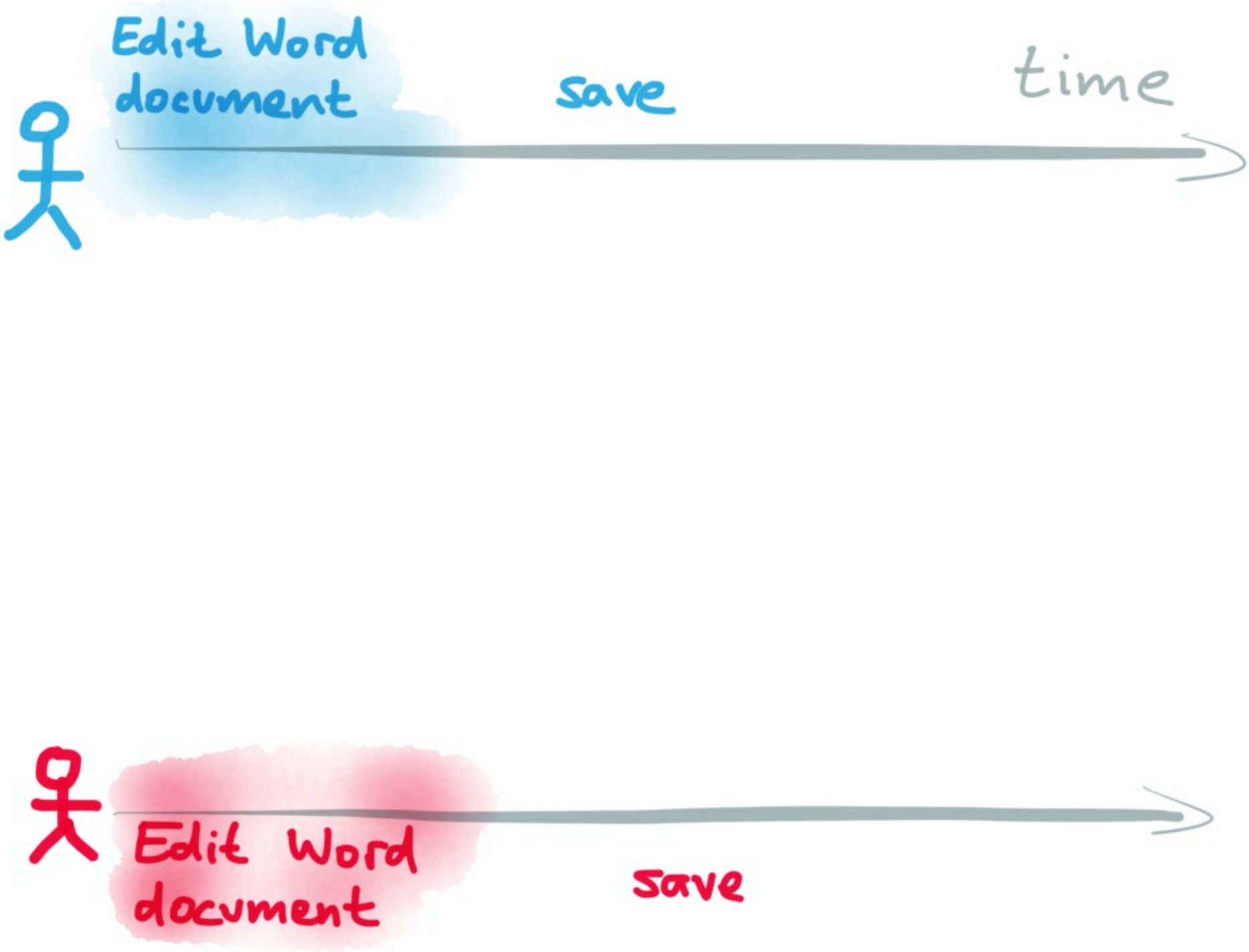

hack on code
git commit
time
git push
GITHUB

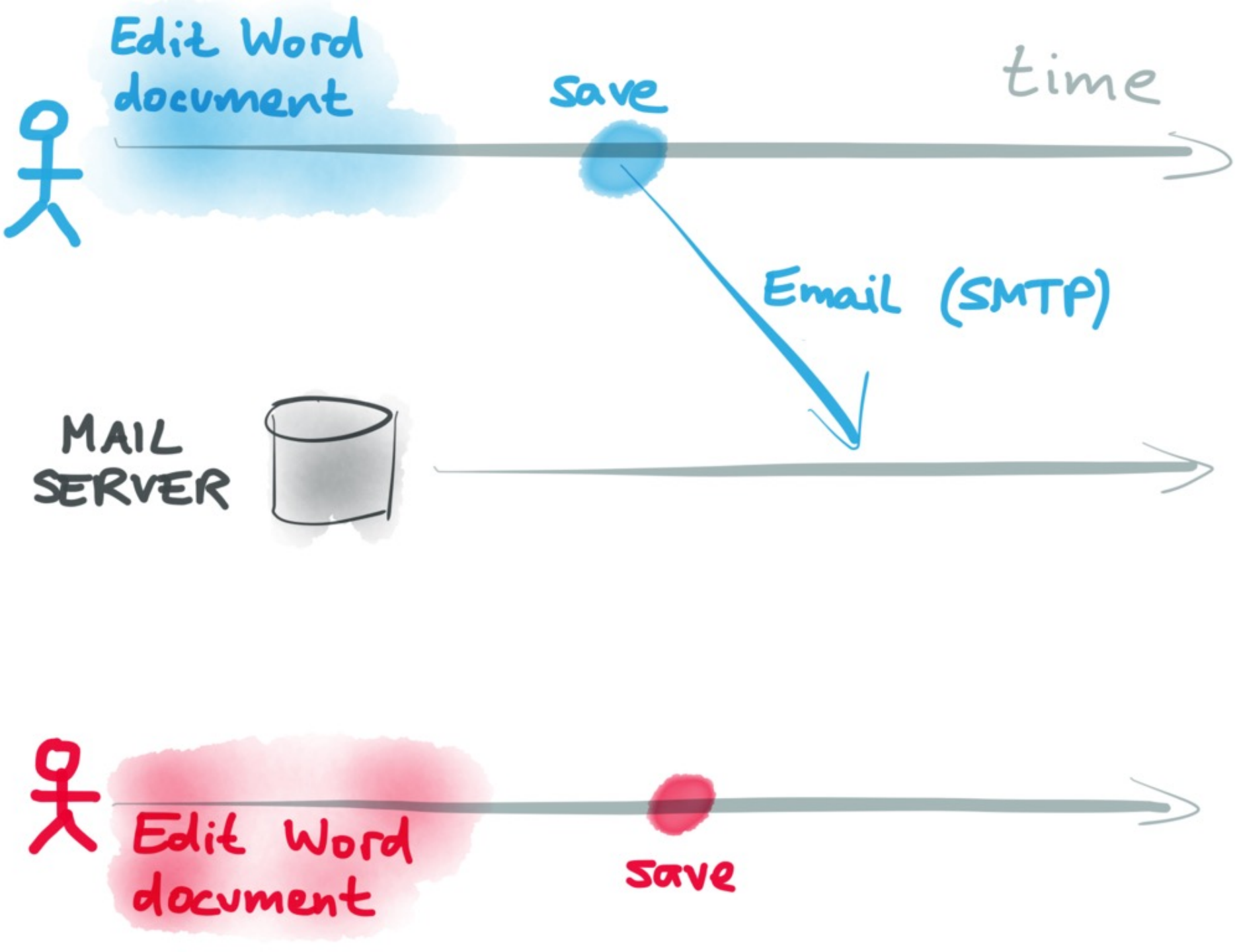

hack on code
git commit
time
git push
GITHUB
hack on code
git commit

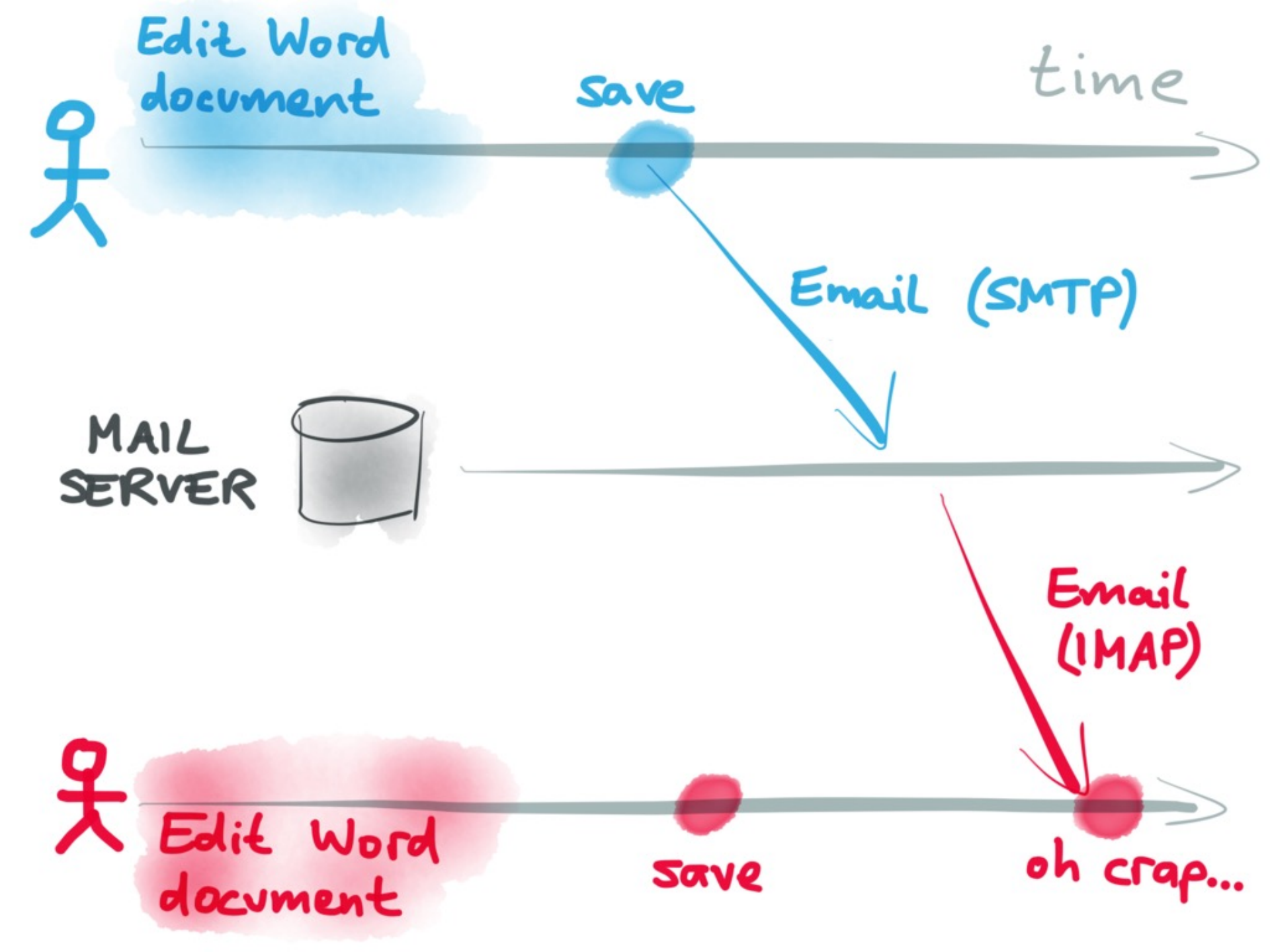

hack on code
git commit
time
git push
GITHUB
git fetch
hack on code
git commit
git merge


* backend_test_LOCAL_3486.js, backend_test_BASE_3486.js, backend_test_REMOTE_3486.js - SourceGear DiffMerge
...martin/dev/cl/automerge/test/backend_test_LOCAL_3486.js
...rtin/dev/cl/automerge/test/backend_test_BASE_3486.js
...martin/dev/cl/automerge/test/backend_test_REMOTE_3486.js
Changes: 17; Conflicts: 35
Reference View (Files as Loaded)
Edit View (Merge Result)
Ruleset: _Default_
default


Edit Word document
save
time


Edit Word document
save
time
Edit Word document
save


Edit Word document
Save
time
Email (SMTP)
MAIL SERVER
Edit Word document
Save


Edit Word document
save
time
Email (SMTP)
MAIL SERVER
Email (IMAP)
Edit Word document
save
oh crap...

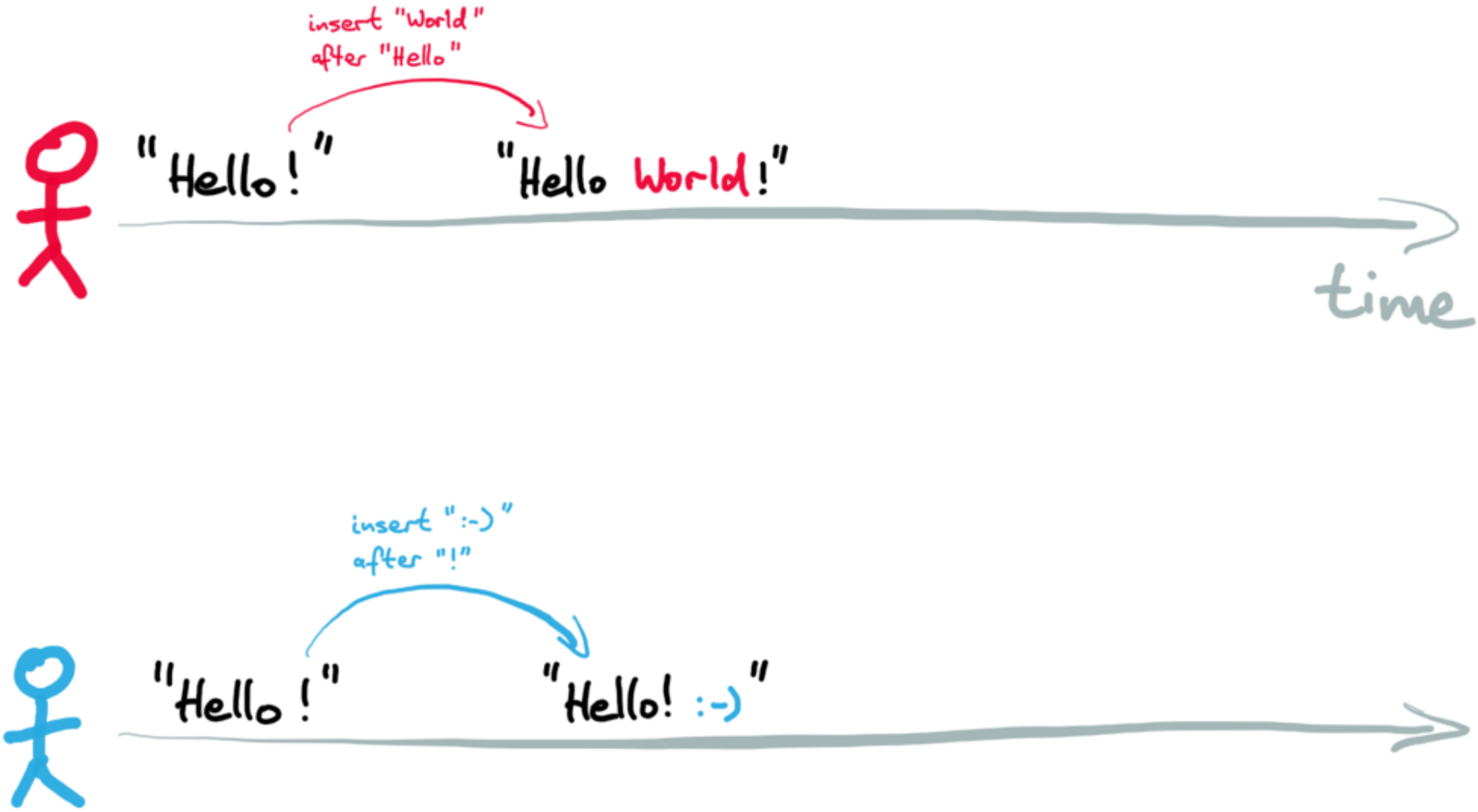
COLLABORATIVE APPLICATIONS

Figma

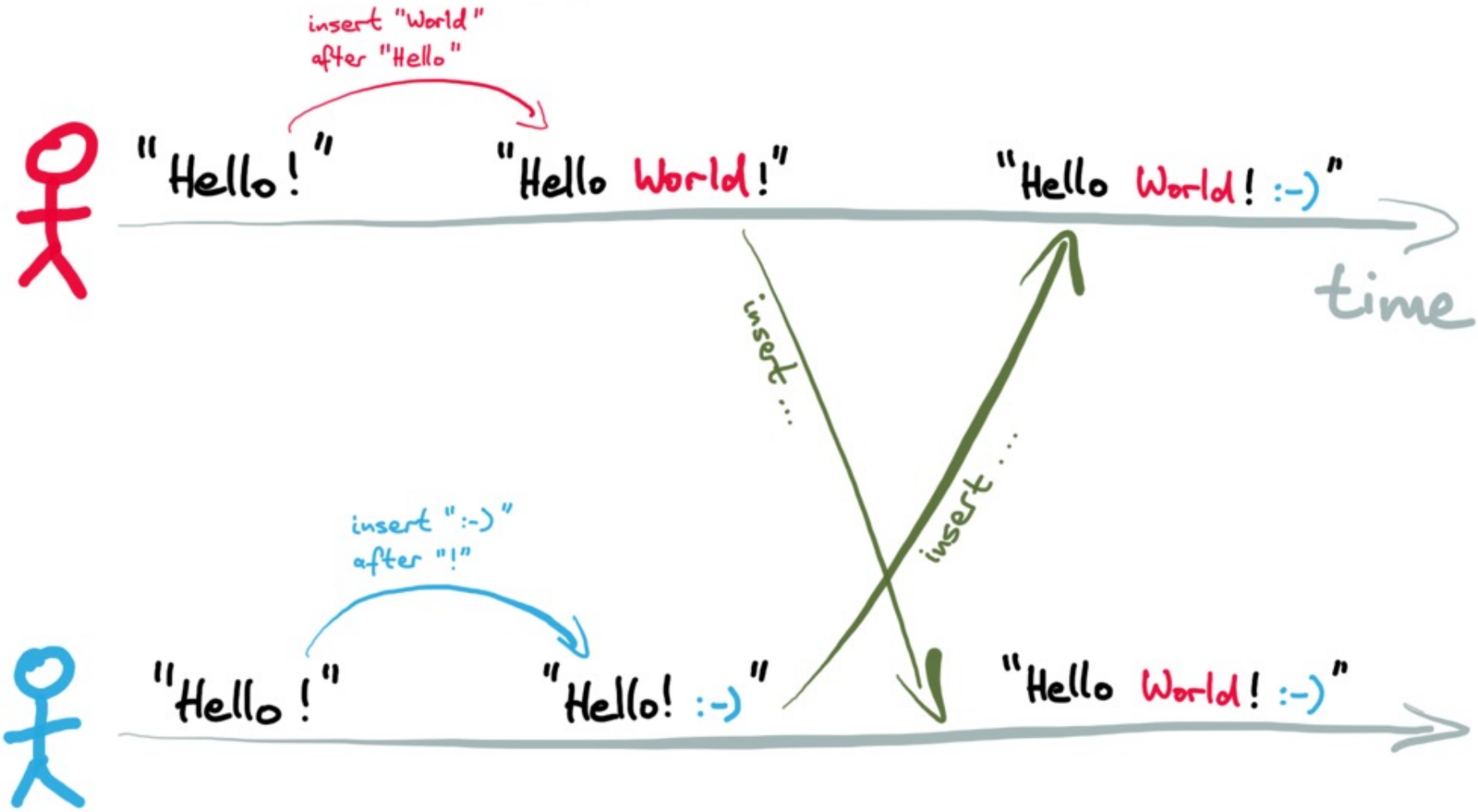
Wide range of domain-specific collaboration software, e.g. for investigative journalism, medical records, data analysis, engineering/CAD, ...

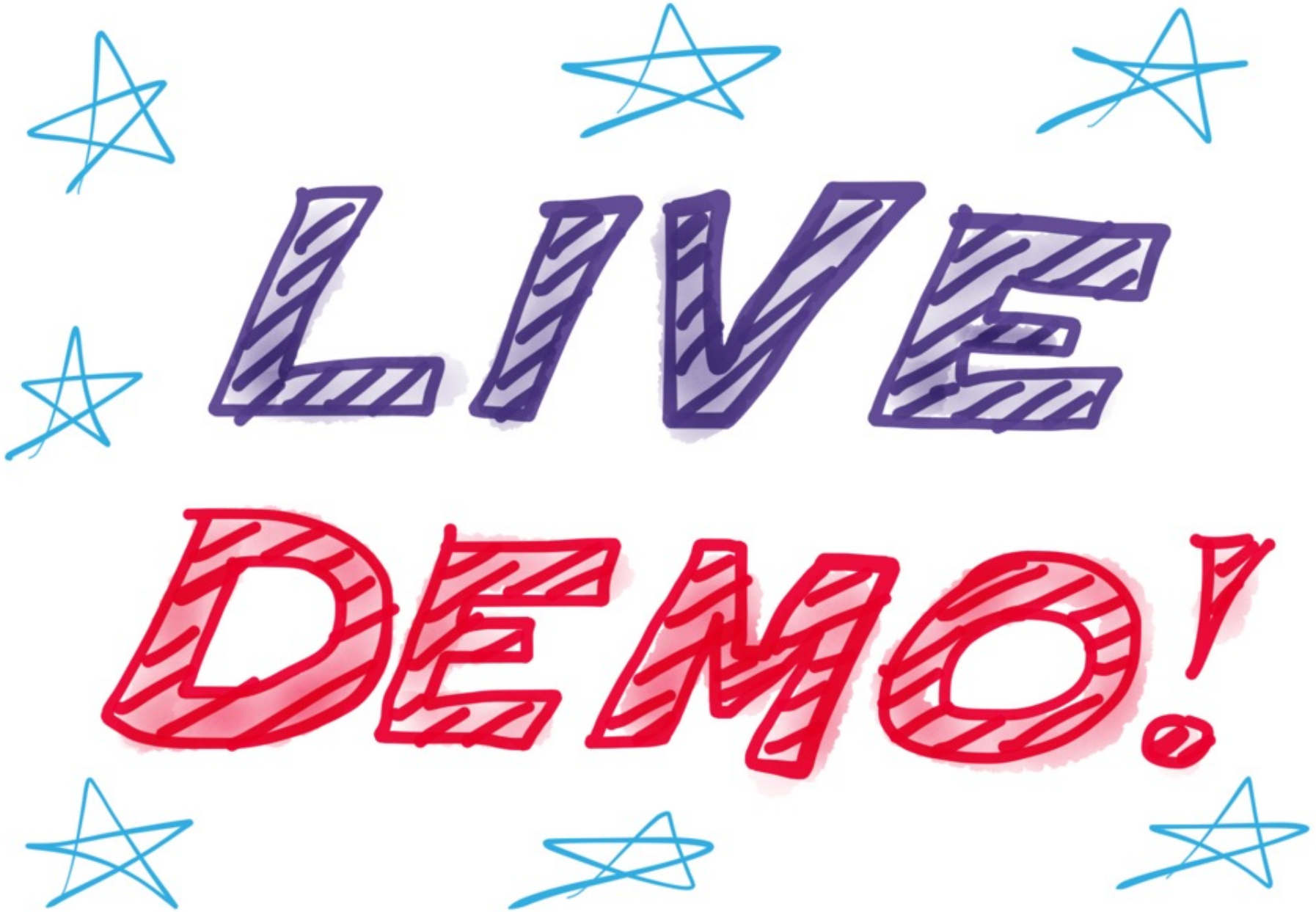
Example: Text editing

Example: Text editing



Example: Text editing




LIVE
DEMO!

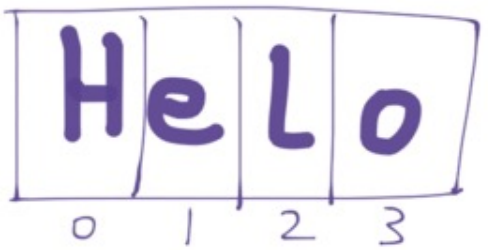
COLLABORATIVE TEXT EDITING • TODAY'S TOPICS

1. Operational transformation
2. Branching and the need for CRDTs
3. Automerge
4. Rich text — Peritext
5. Non-interleaving — Fugue

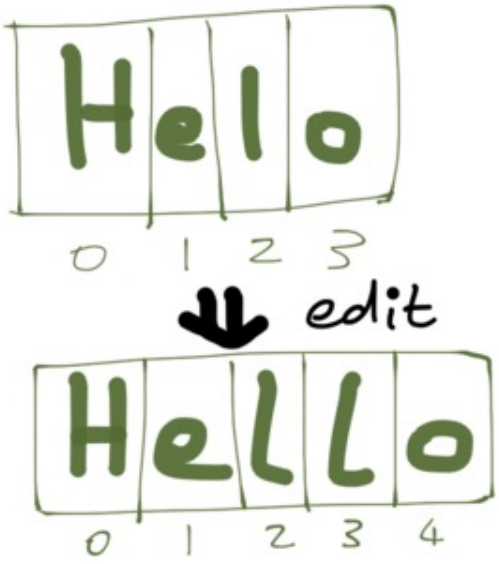
COLLABORATIVE TEXT EDITING • TODAY'S TOPICS

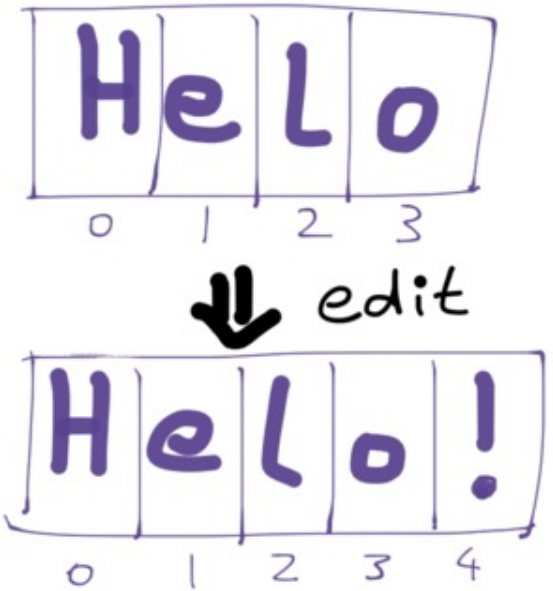
1. Operational transformation
2. Branching and the need for CRDTs
3. Automerge
4. Rich text — Peritext
5. Non-interleaving — Fugue

GOOGLE DOCS (NUTSHELL)

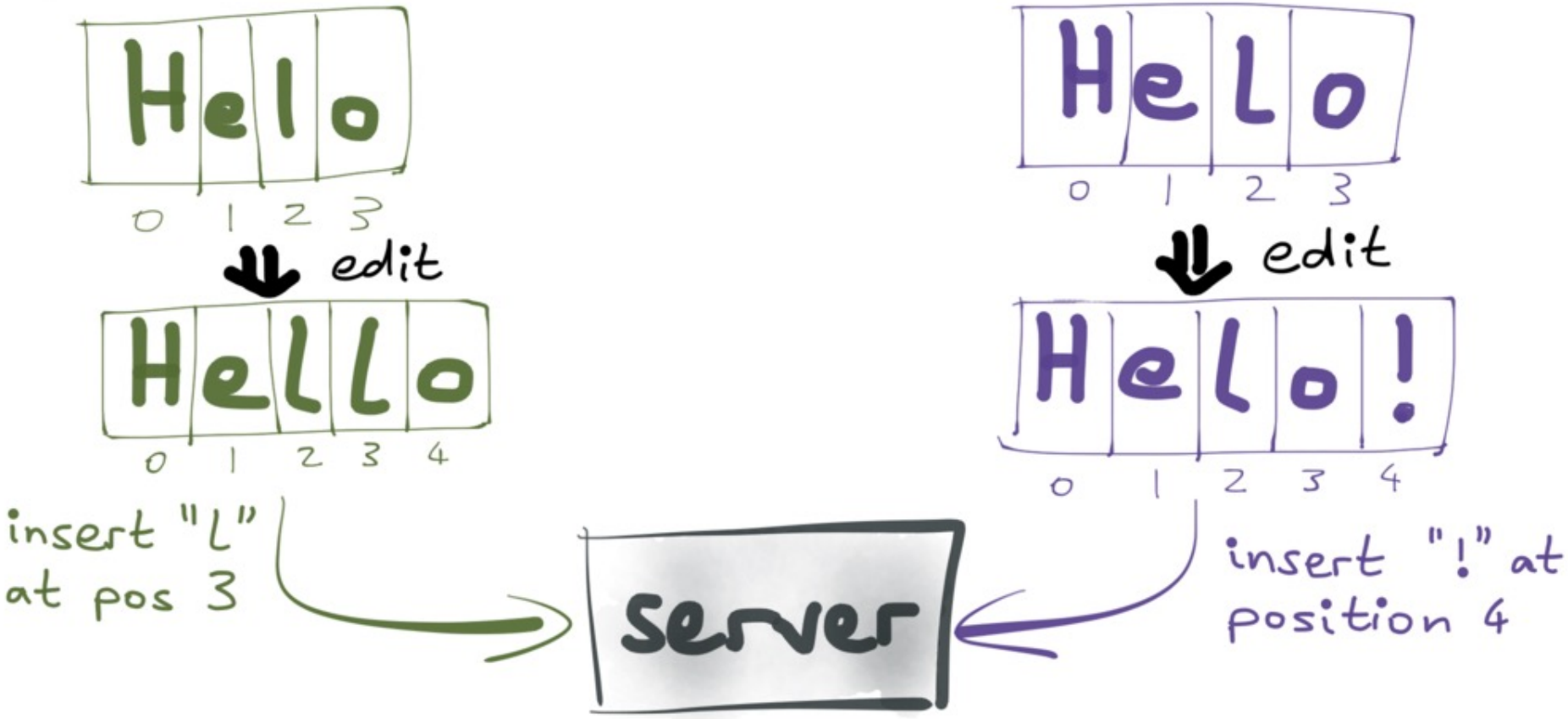


GOOGLE DOCS (NUTSHELL)

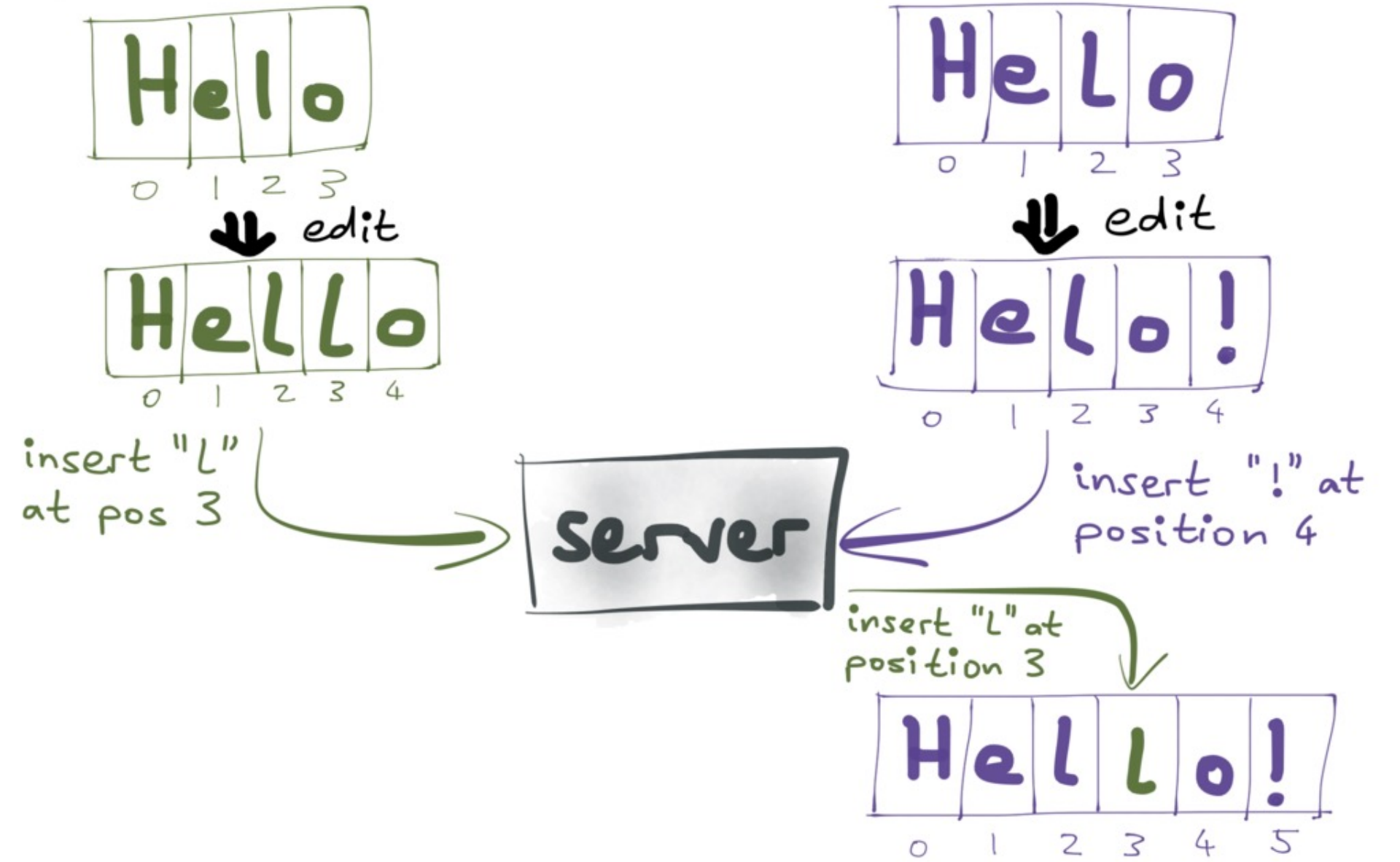




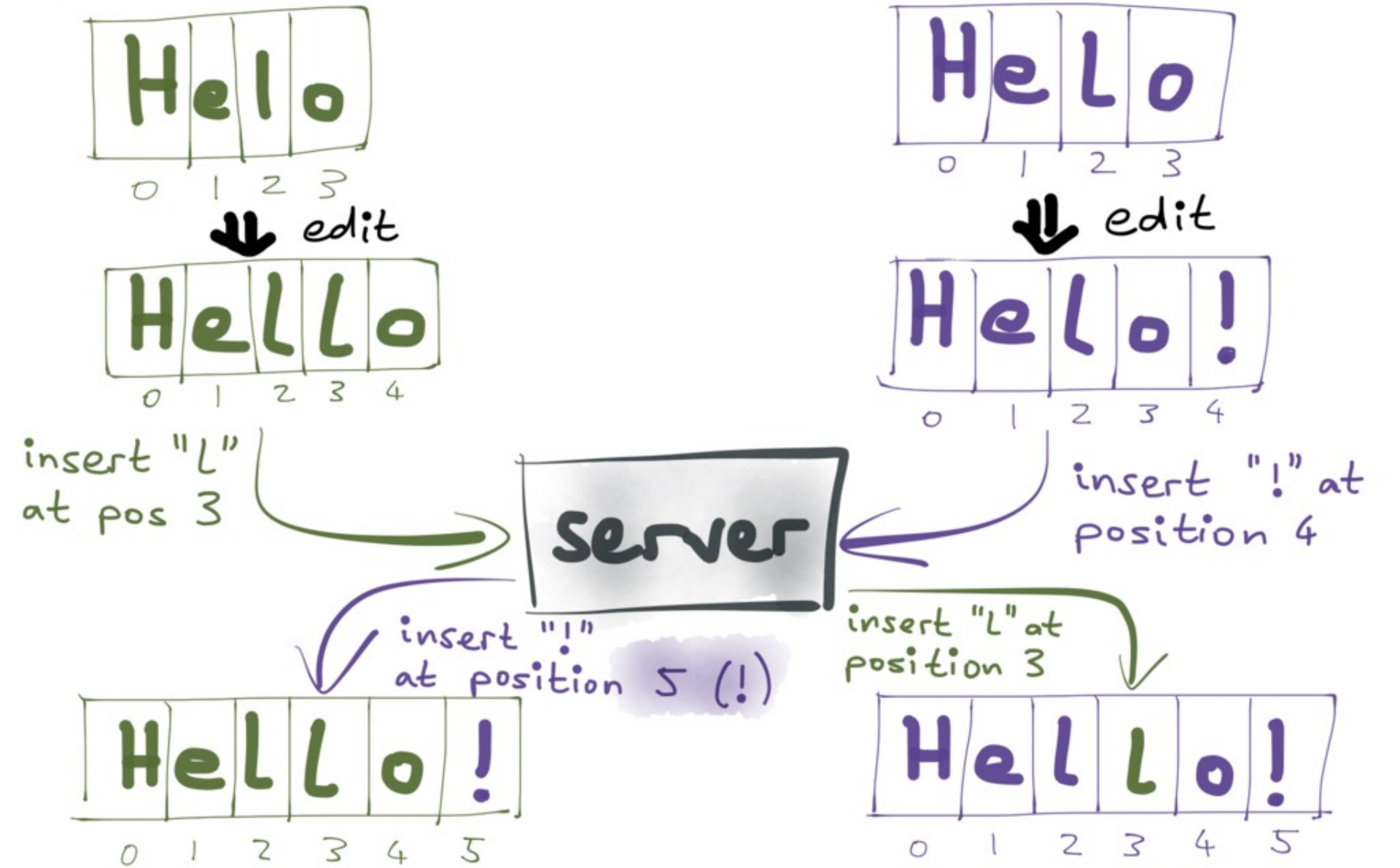
GOOGLE DOCS (NUTSHELL)



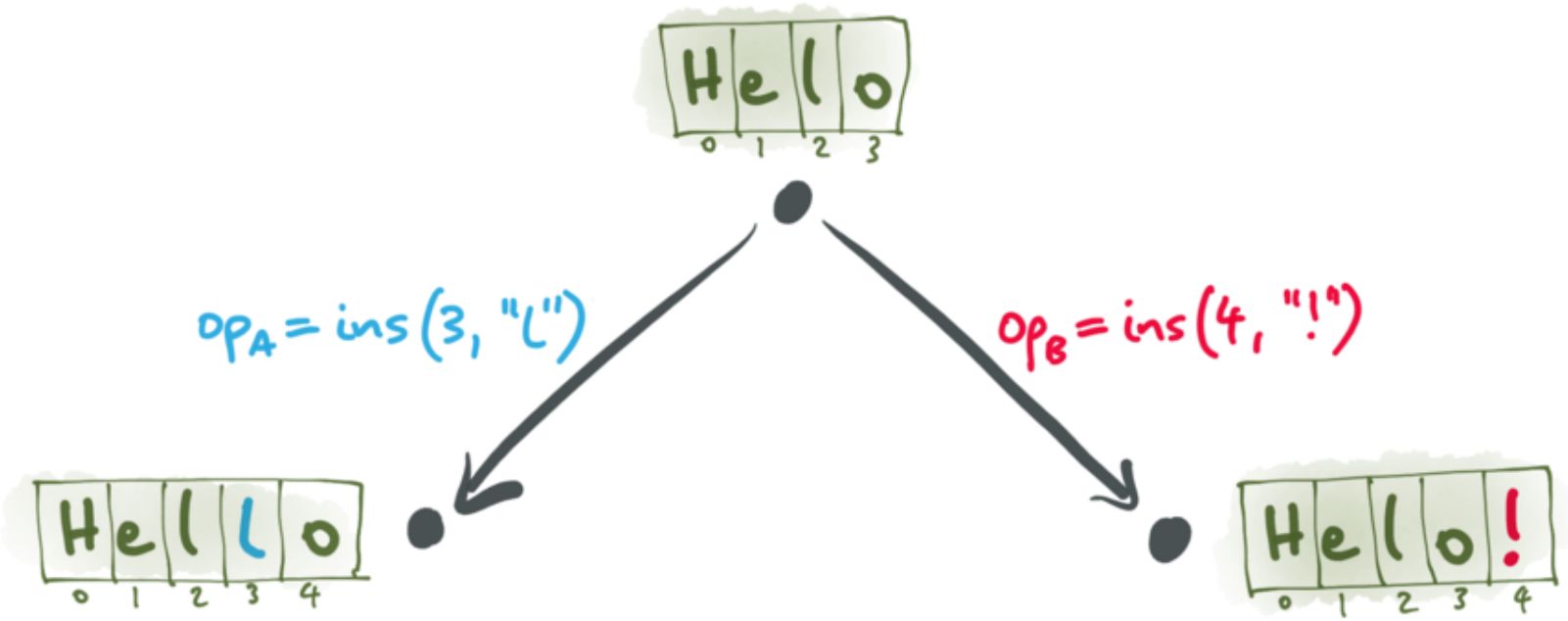
GOOGLE DOCS (NUTSHELL)



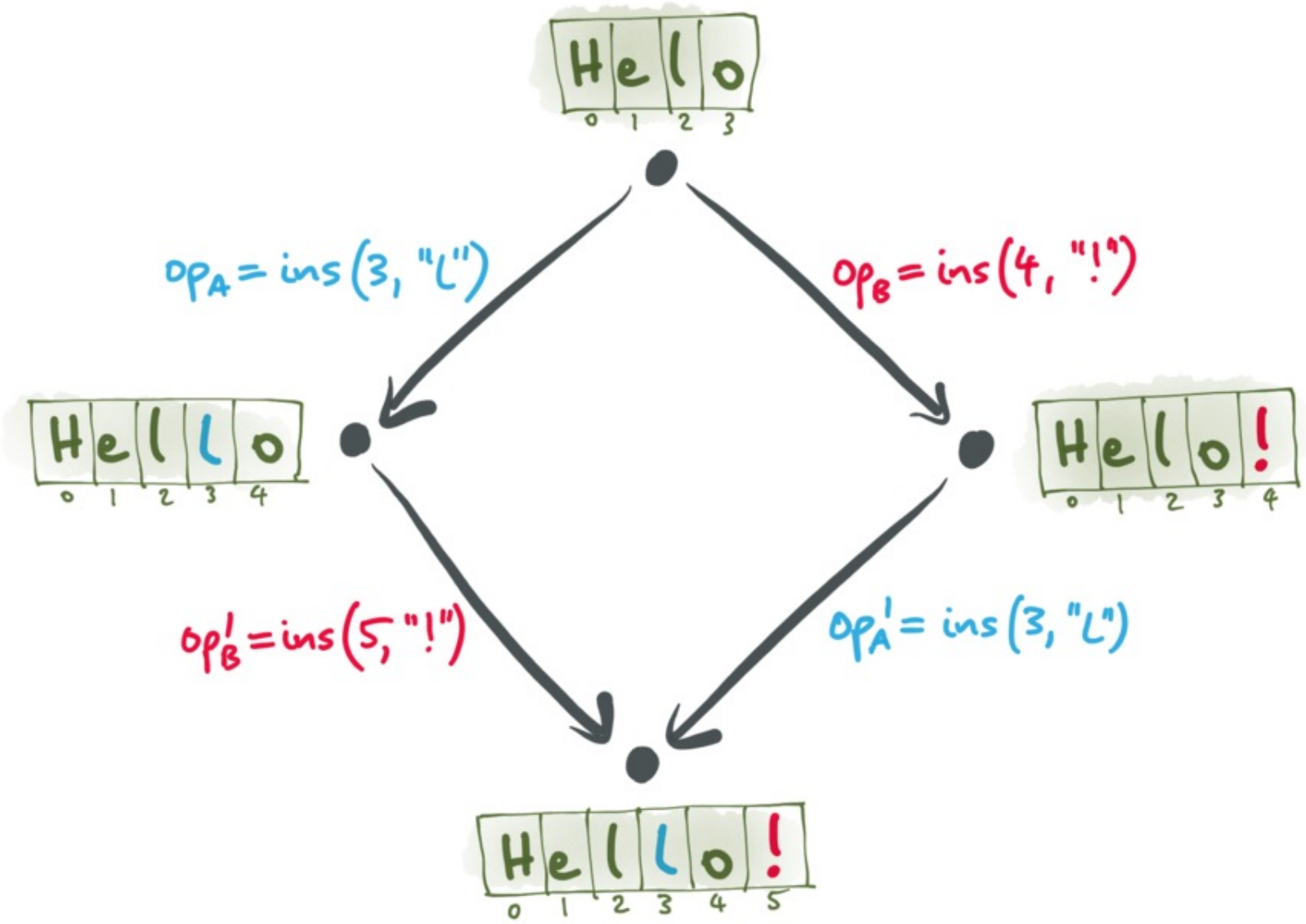
GOOGLE DOCS (NUTSHELL)



Operational Transformation (OT)



Operational Transformation (OT)



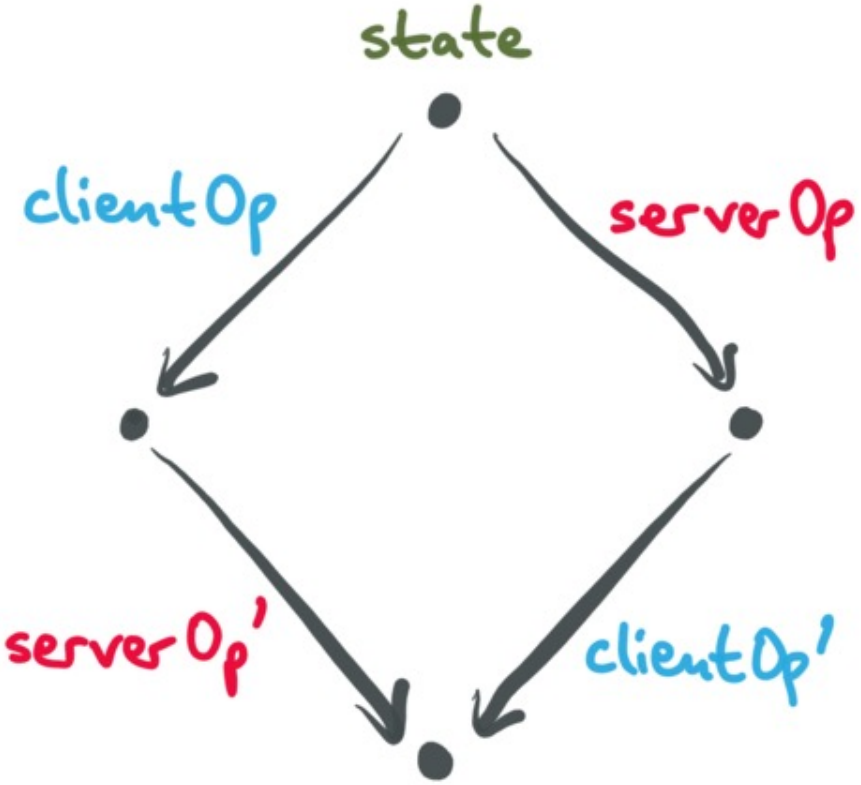
Jupiter: client-server operational transformation (1995)

xform (clientOp, serverOp) = (clientOp', serverOp')

TP1 property:

Let clientOp and serverOp be concurrent operations that both took place in the same state. Let xform(clientOp, serverOp) = (clientOp', serverOp').

Then serverOp'(clientOp(state)) = clientOp'(serverOp(state)).



$$\text{xform}(\text{ins}(i,x), \text{ins}(j,y)) = \begin{cases} (\text{ins}(i,x), \text{ins}(j+1,y)) & \text{if } i < j \\ (\text{ins}(i+1,x), \text{ins}(j,y)) & \text{if } i \geq j \end{cases}$$

$$\text{xform}(\text{ins}(i,x), \text{del}(j)) = \begin{cases} (\text{ins}(i,x), \text{del}(j+1)) & \text{if } i \leq j \\ (\text{ins}(i-1,x), \text{del}(j)) & \text{if } i > j \end{cases}$$

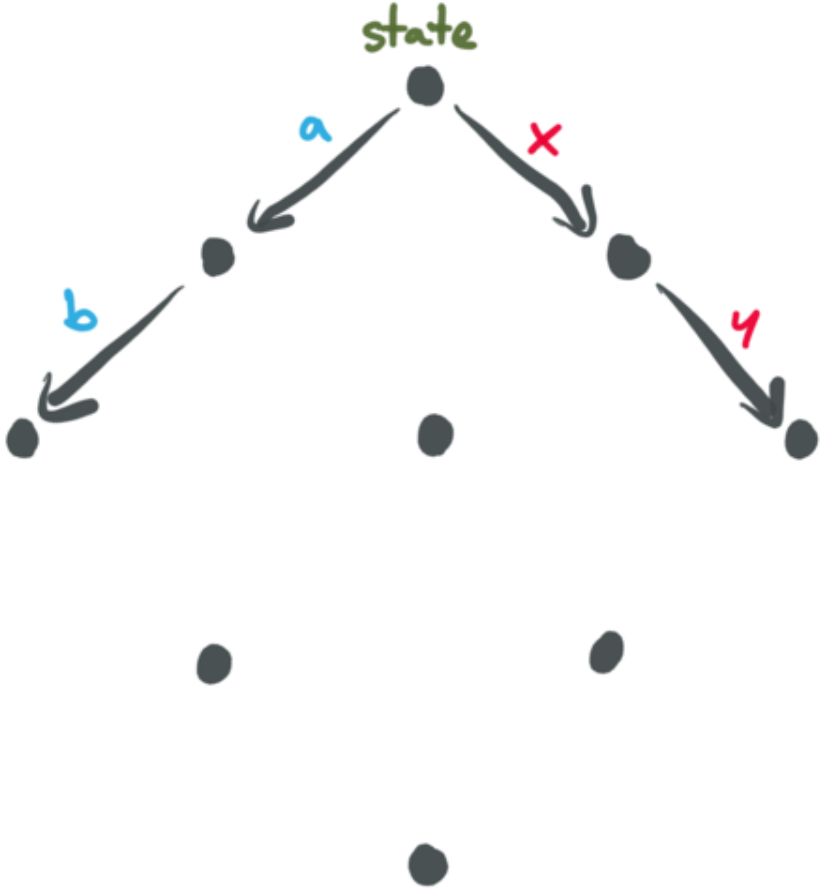
$$\text{xform}(\text{del}(i), \text{ins}(j,y)) = \begin{cases} (\text{del}(i), \text{ins}(j-1,y)) & \text{if } i < j \\ (\text{del}(i+1), \text{ins}(j,y)) & \text{if } i \geq j \end{cases}$$

$$\text{xform}(\text{del}(i), \text{del}(j)) = \begin{cases} (\text{del}(i), \text{del}(j-1)) & \text{if } i < j \\ (\text{del}(i-1), \text{del}(j)) & \text{if } i > j \\ (\text{noop}, \text{noop}) & \text{if } i = j \end{cases}$$

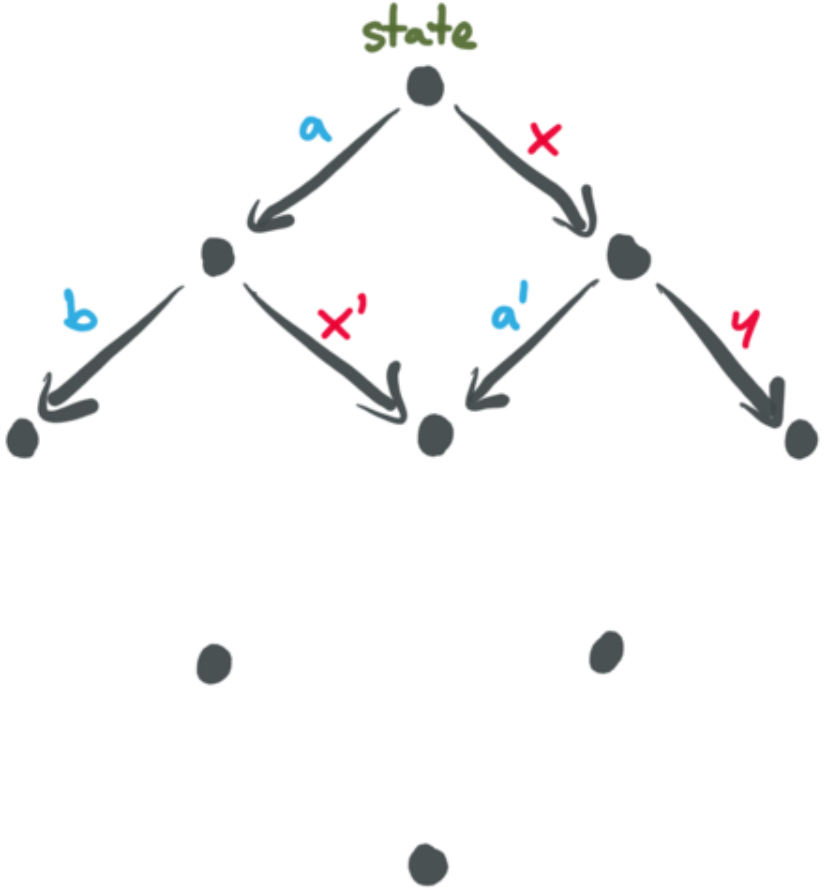
$$\text{xform}(\text{op}, \text{noop}) = (\text{op}, \text{noop})$$

$$\text{xform}(\text{noop}, \text{op}) = (\text{noop}, \text{op})$$

Diverged by more than one op?

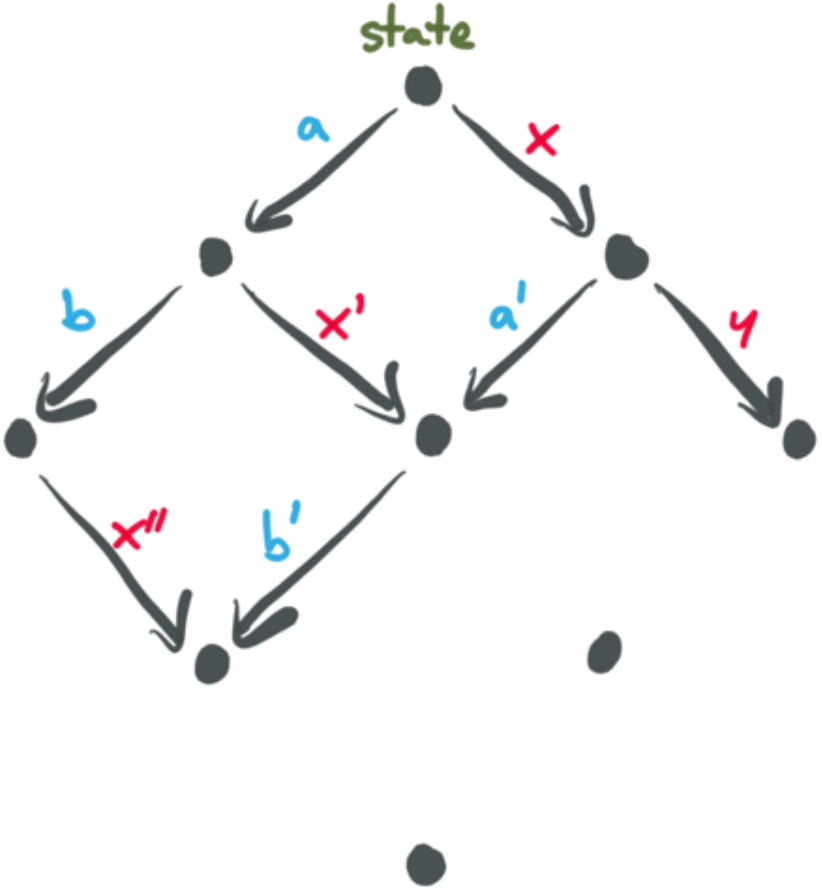


Diverged by more than one op?



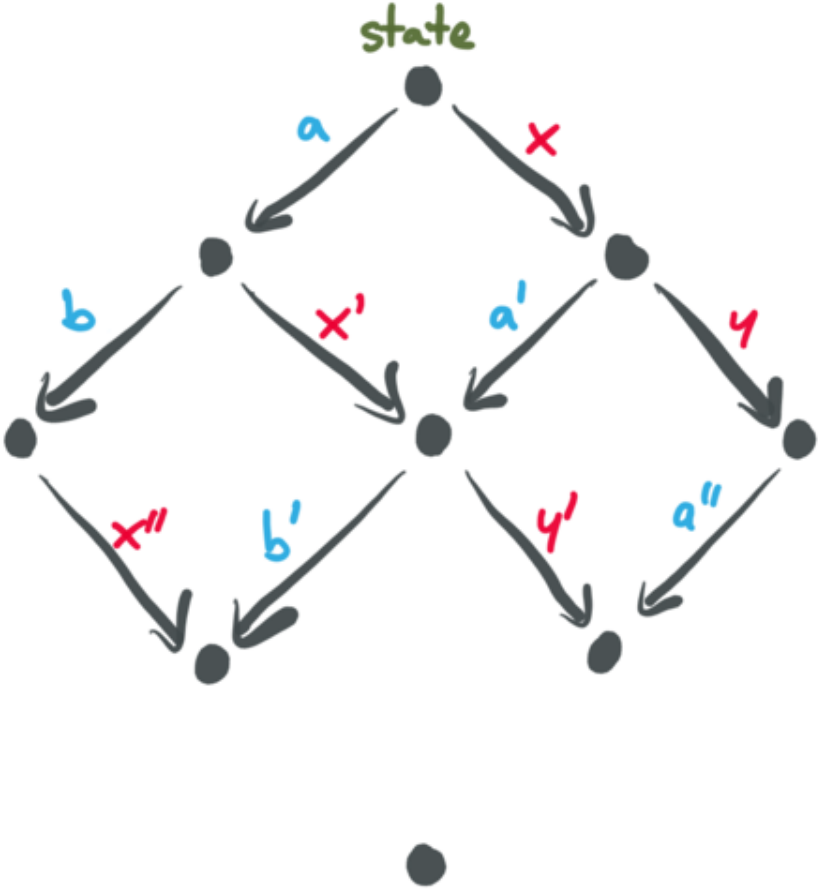
$$xform(a, x) = (a', x')$$

Diverged by more than one op?



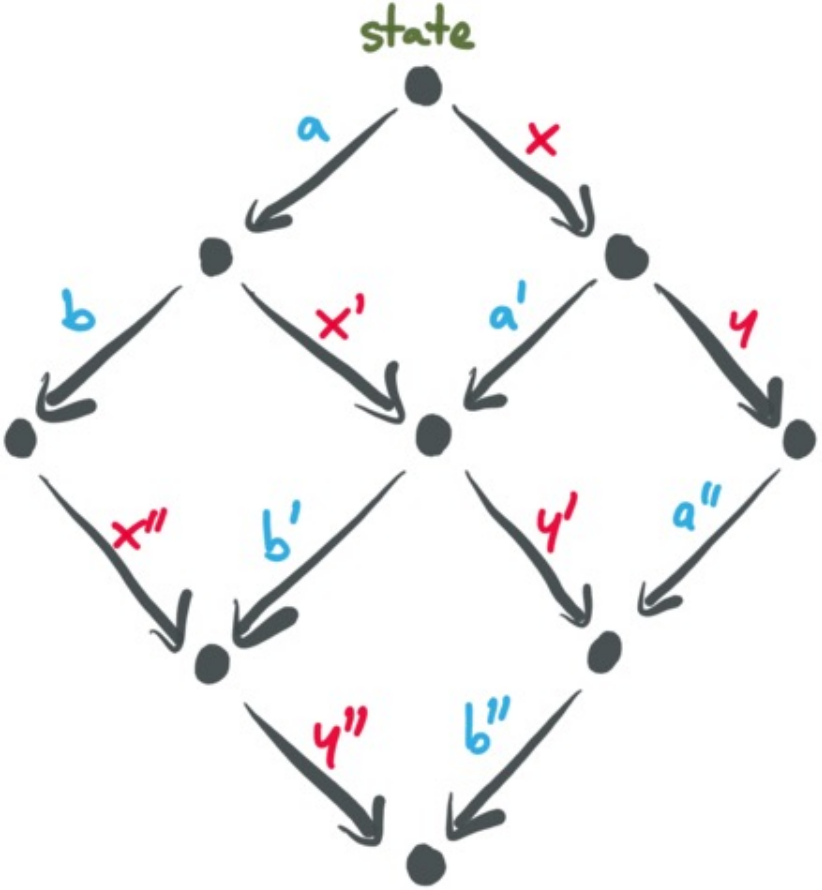
$$xform(a, x) = (a', x')$$
$$xform(b, x') = (b', x'')$$

Diverged by more than one op?



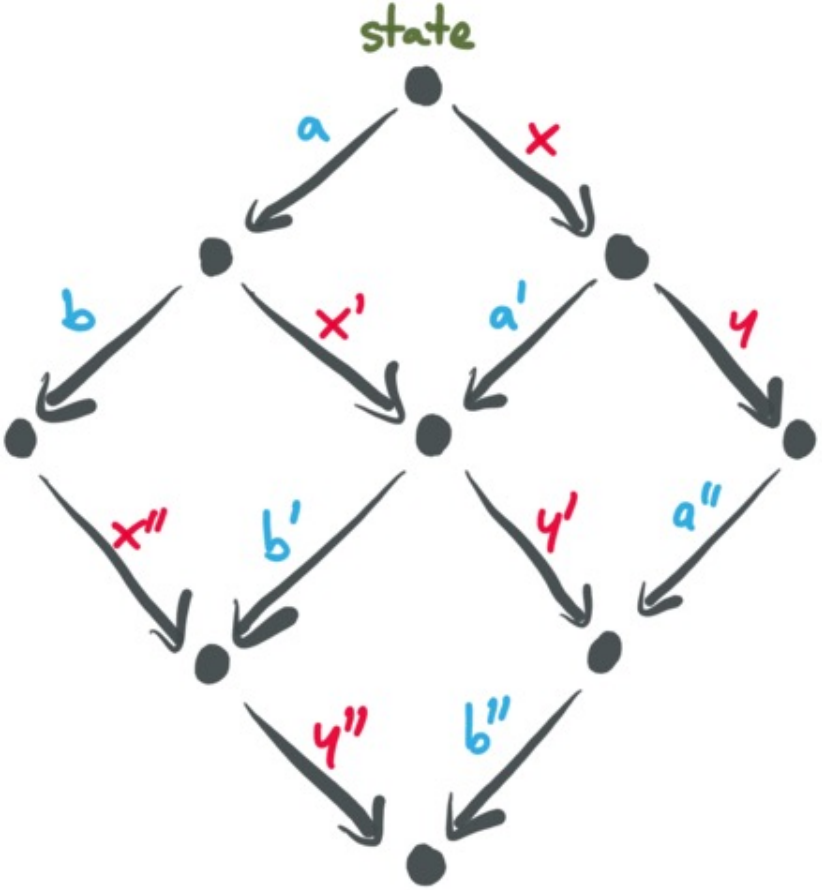
$$xform(a, x) = (a', x')$$
$$xform(b, x') = (b', x'')$$
$$xform(a', y) = (a'', y')$$

Diverged by more than one op?



$$xform(a, x) = (a', x')$$
$$xform(b, x') = (b', x'')$$
$$xform(a', y) = (a'', y')$$
$$xform(b', y') = (b'', y'')$$

Diverged by more than one op?



$xform(a, x) = (a', x')$

$xform(b, x') = (b', x'')$

$xform(a', y) = (a'', y')$

$xform(b', y') = (b'', y'')$

Then

$y''(x''(b(a(state)))) =$

$b''(a''(y(x(state))))$.

Client and server each maintain a queue of operations they have sent to the other party, and that have not yet been acknowledged.

Transform each received op w.r.t. the ops in that queue.

Supporting multiple clients

- Independent instance of the protocol between the server and each client.

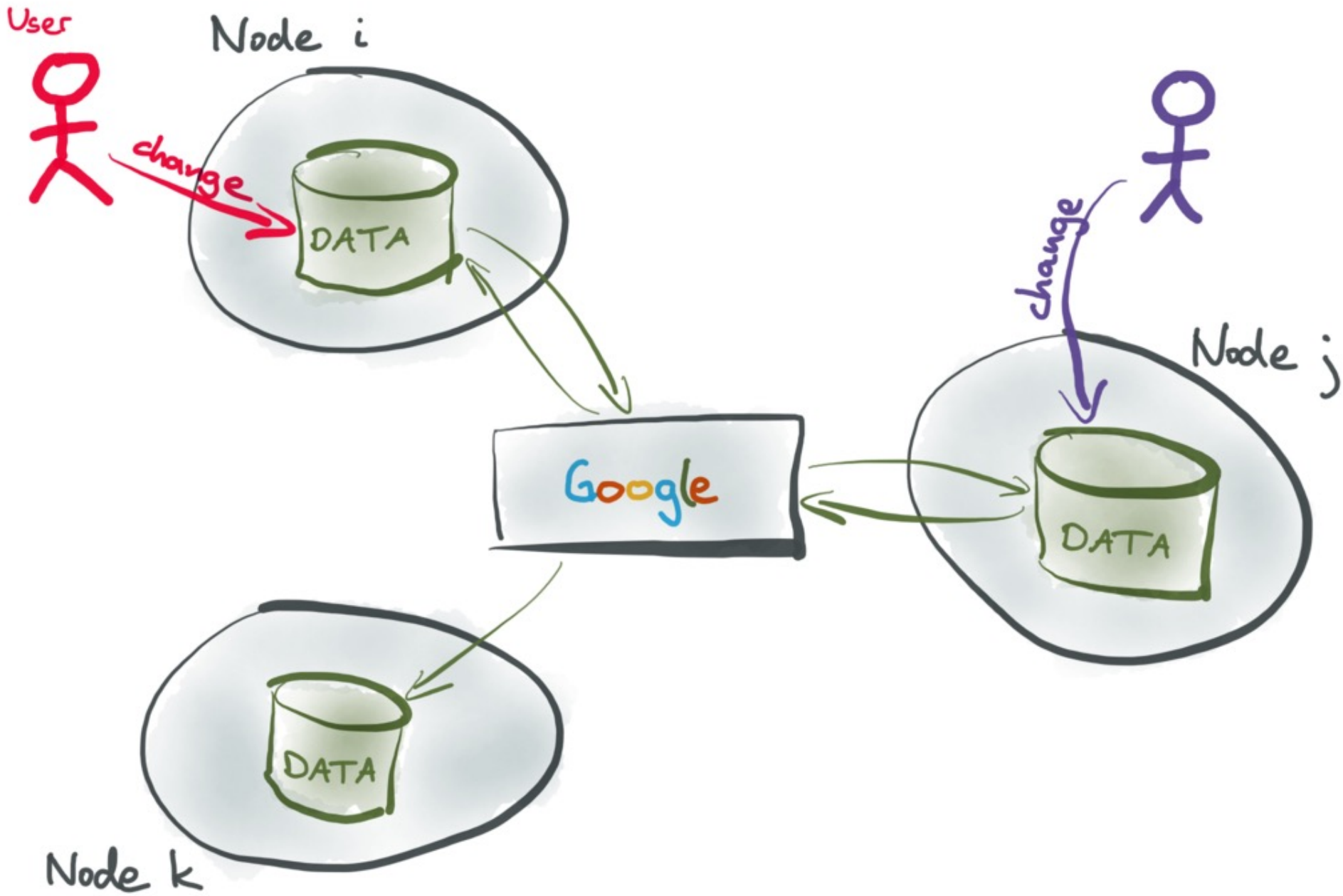
Supporting multiple clients

- Independent instance of the protocol between the server and each client.
- When an operation from one client has been transformed and applied to the server's state, it becomes a server operation from the other clients' point of view.

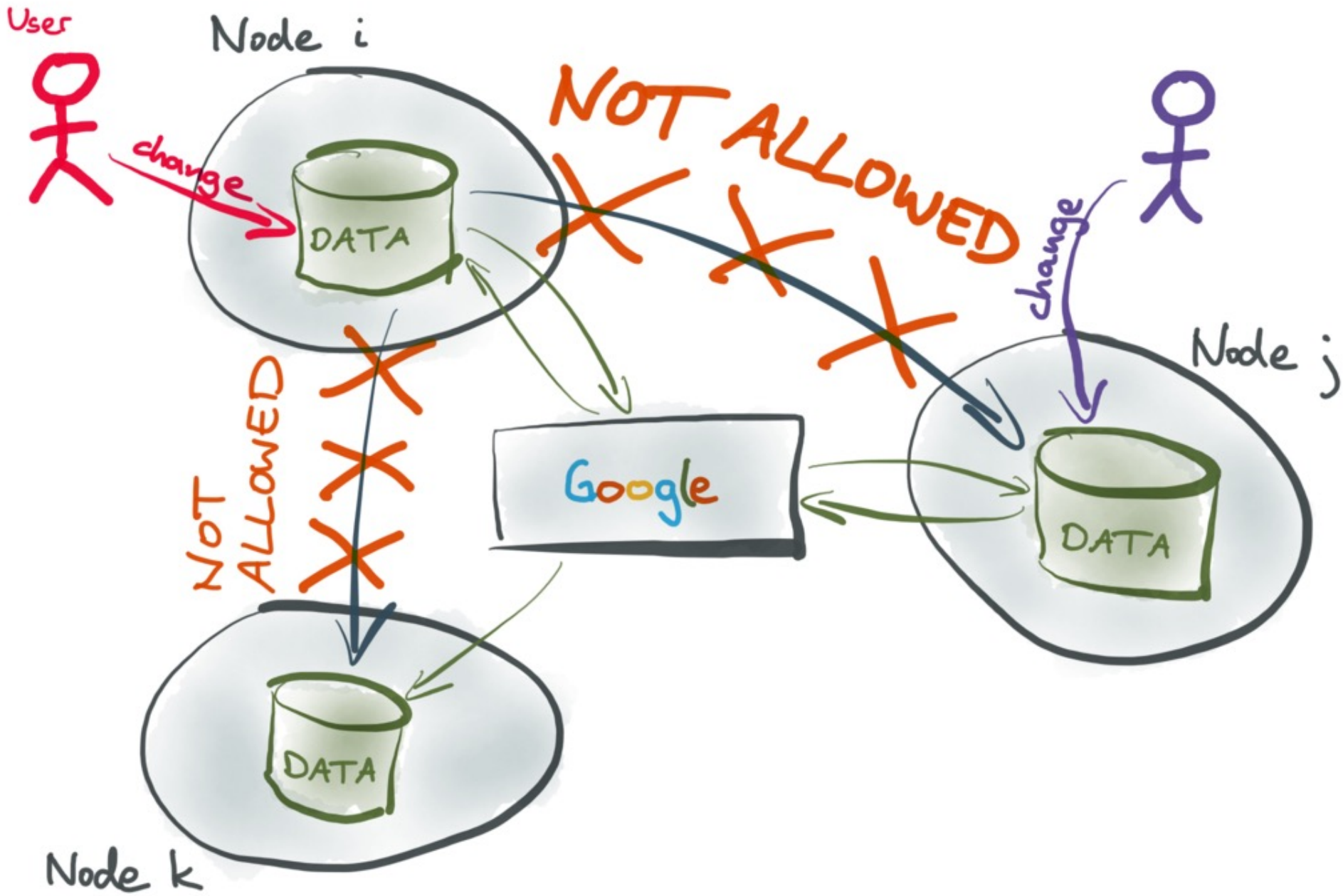
Supporting multiple clients

- Independent instance of the protocol between the server and each client.
- When an operation from one client has been transformed and applied to the server's state, it becomes a server operation from the other clients' point of view.
- The server has a single copy of the document for all clients, and the server applies operations sequentially.
- A server document version is identified by a single integer.

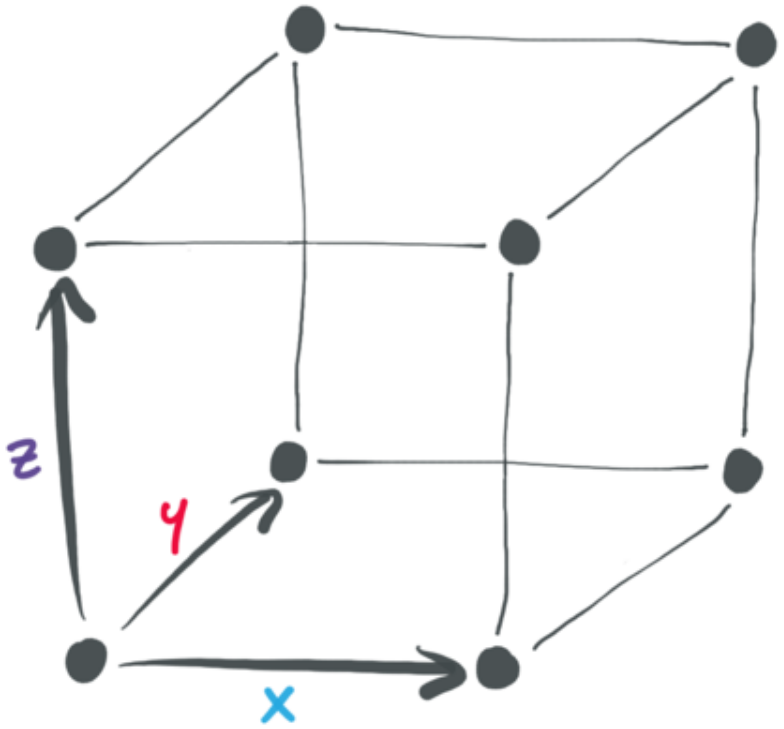
OPERATIONAL TRANSFORMATION IN GOOGLE DOCS.



OPERATIONAL TRANSFORMATION IN GOOGLE DOCS.



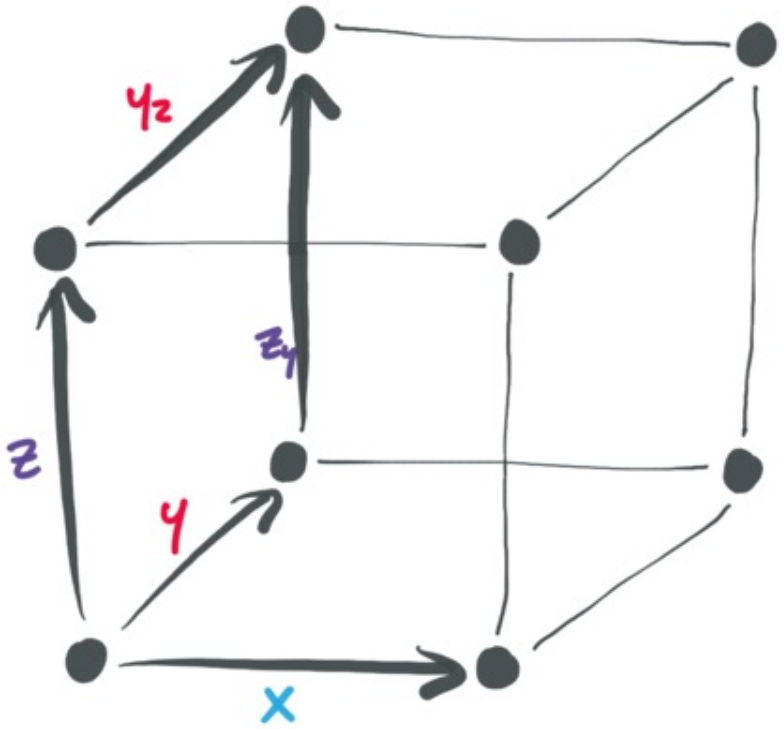
Can we manage without a server?



TP2 property:

Let x, y, z be concurrent ops on the same state

Can we manage without a server?

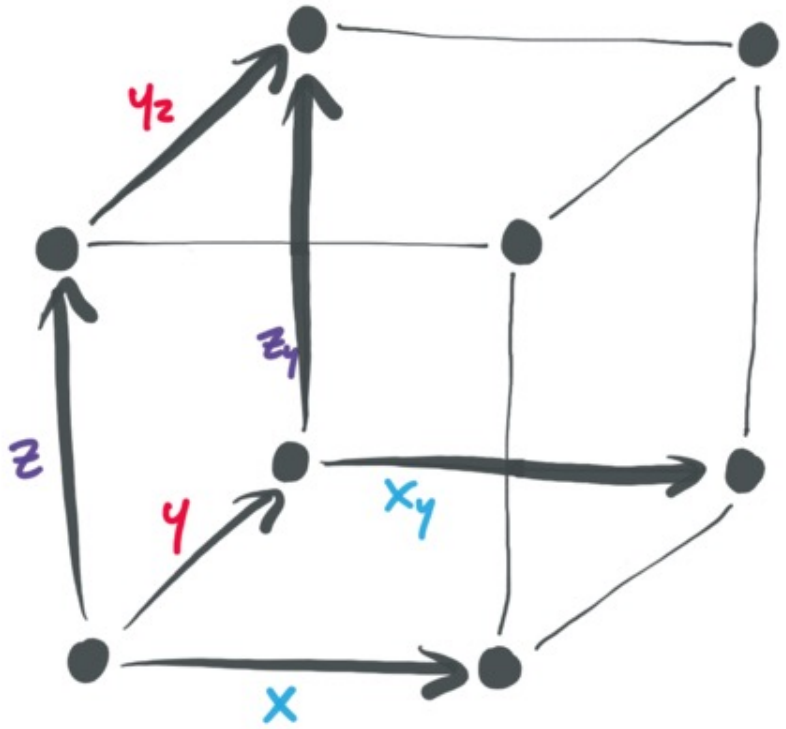


TP2 property:

Let x, y, z be concurrent ops on the same state, and let

$xform(y, z) = (y_2, z_y)$

Can we manage without a server?



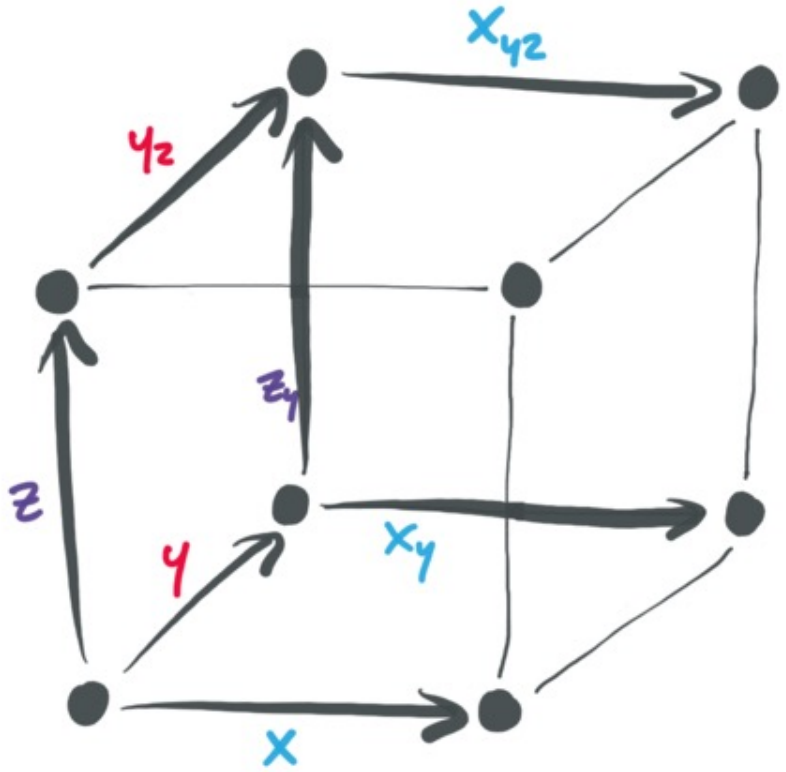
TP2 property:

Let x, y, z be concurrent ops on the same state, and let

$xform(y, z) = (y_z, z_y)$

$xform(x, y) = (x_y, \_)$

Can we manage without a server?



TP2 property:

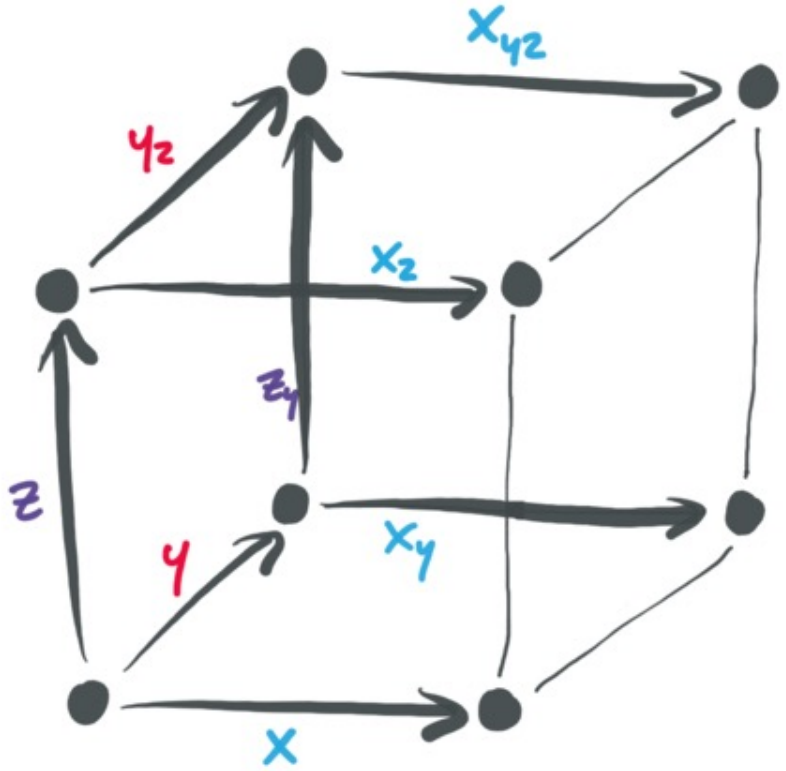
Let x, y, z be concurrent ops on the same state, and let

$xform(y, z) = (y_z, z_y)$

$xform(x, y) = (x_y, \_)$

$xform(x_y, z_y) = (x_{yz}, \_)$

Can we manage without a server?



TP2 property:

Let x, y, z be concurrent ops on the same state, and let

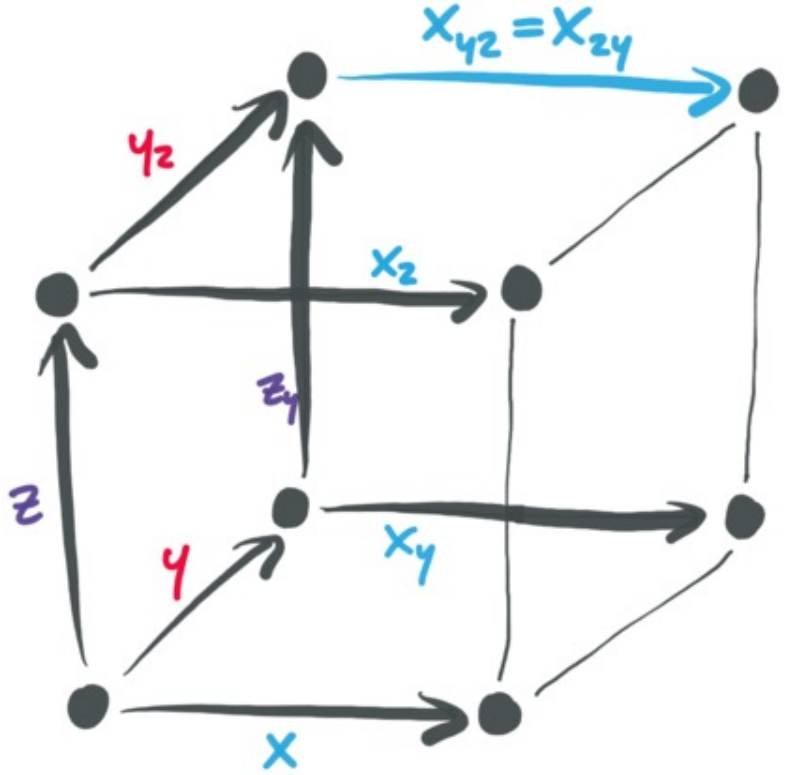
$xform(y, z) = (y_z, z_y)$

$xform(x, y) = (x_y, \_)$

$xform(x_y, z_y) = (x_{yz}, \_)$

$xform(x, z) = (x_z, \_)$

Can we manage without a server?



TP2 property:

Let x, y, z be concurrent ops on the same state, and let

$xform(y, z) = (y_2, z_y)$

$xform(x, y) = (x_y, \_)$

$xform(x_y, z_y) = (x_{y2}, \_)$

$xform(x, z) = (x_2, \_)$

$xform(x_2, y_2) = (x_{2y}, \_)$

Then $x_{y2} = x_{2y}$.

FAILURES OF OPERATIONAL TRANSFORM

dOPT
Ellis & Gibbs
1989

adOPTed
Ressel et al.
1996

IMOR
Imine et al.
2003

Jupiter
Nichols et al.
1995

SOCT2
Suleiman et al.
1997

SDT
Li & Li
2004

SOCT3/4
Vidot et al.
2000

TTF
Oster et al.
2006

FAILURES OF OPERATIONAL TRANSFORM

~~dOPT~~
~~Ellis & Gibbs~~
1989
wrong

adOPTed
Ressel et al.
1996

IMOR
Imine et al.
2003

Jupiter
Nichols et al.
1995

SOCT2
Suleiman et al.
1997

SDT
Li & Li
2004

SOCT3/4
Vidot et al.
2000

TTF
Oster et al.
2006

FAILURES OF OPERATIONAL TRANSFORM

~~dOPT~~
~~Ellis & Gibbs~~
1989
Wrong

~~adOPTed~~
~~Ressel et al.~~
1996
Wrong

IMOR
Imine et al.
2003

Jupiter
Nichols et al.
1995

SOCT2
Suleiman et al.
1997

SDT
Li & Li
2004

SOCT3/4
Vidot et al.
2000

TTF
Oster et al.
2006

FAILURES OF OPERATIONAL TRANSFORM

~~dOPT~~
~~Ellis & Gibbs~~
~~1989~~
Wrong

~~adOPTed~~
~~Ressel et al.~~
~~1996~~
Wrong

IMOR
Imine et al.
2003

Jupiter
Nichols et al.
1995

~~SOCT2~~
~~Suleiman et al.~~
~~1997~~
Wrong

SDT
Li & Li
2004

SOCT3/4
Vidot et al.
2000

TTF
Oster et al.
2006

FAILURES OF OPERATIONAL TRANSFORM

~~dOPT~~
~~Ellis & Gibbs~~
~~1989~~
Wrong

~~adOPTed~~
~~Ressel et al.~~
~~1996~~
Wrong

~~IMOR~~
~~Imine et al.~~
~~2003~~
Wrong

Jupiter
Nichols et al.
1995

~~SOCT2~~
~~Suleiman et al.~~
~~1997~~
Wrong

SDT
Li & Li
2004

SOCT3/4
Vidot et al.
2000

TTF
Oster et al.
2006

FAILURES OF OPERATIONAL TRANSFORM

~~dOPT~~
~~Ellis & Gibbs~~
~~1989~~
~~Wrong~~

~~adOPTed~~
~~Ressel et al.~~
~~1996~~
~~Wrong~~

~~IMOR~~
~~Imine et al.~~
~~2003~~
~~Wrong~~

Jupiter
Nichols et al.
1995

~~SOCT2~~
~~Suleiman et al.~~
~~1997~~
~~Wrong~~

~~SDT~~
~~Li & Li~~
~~2006~~
~~Wrong~~

SOCT3/4
Vidot et al.
2000

TTF
Oster et al.
2006

FAILURES OF OPERATIONAL TRANSFORM

dOPT
Ellis & Gibbs
1989
Wrong

adOPTed
Ressel et al.
1996
Wrong

IMOR
Imine et al.
2003
Wrong

Jupiter
Nichols et al.
1995
require central server

SOCT2
Suleiman et al.
1997
Wrong

SDT
Li & Li
2006
Wrong

SOCT3/4
Vidot et al.
2000

TTF
Oster et al.
2006

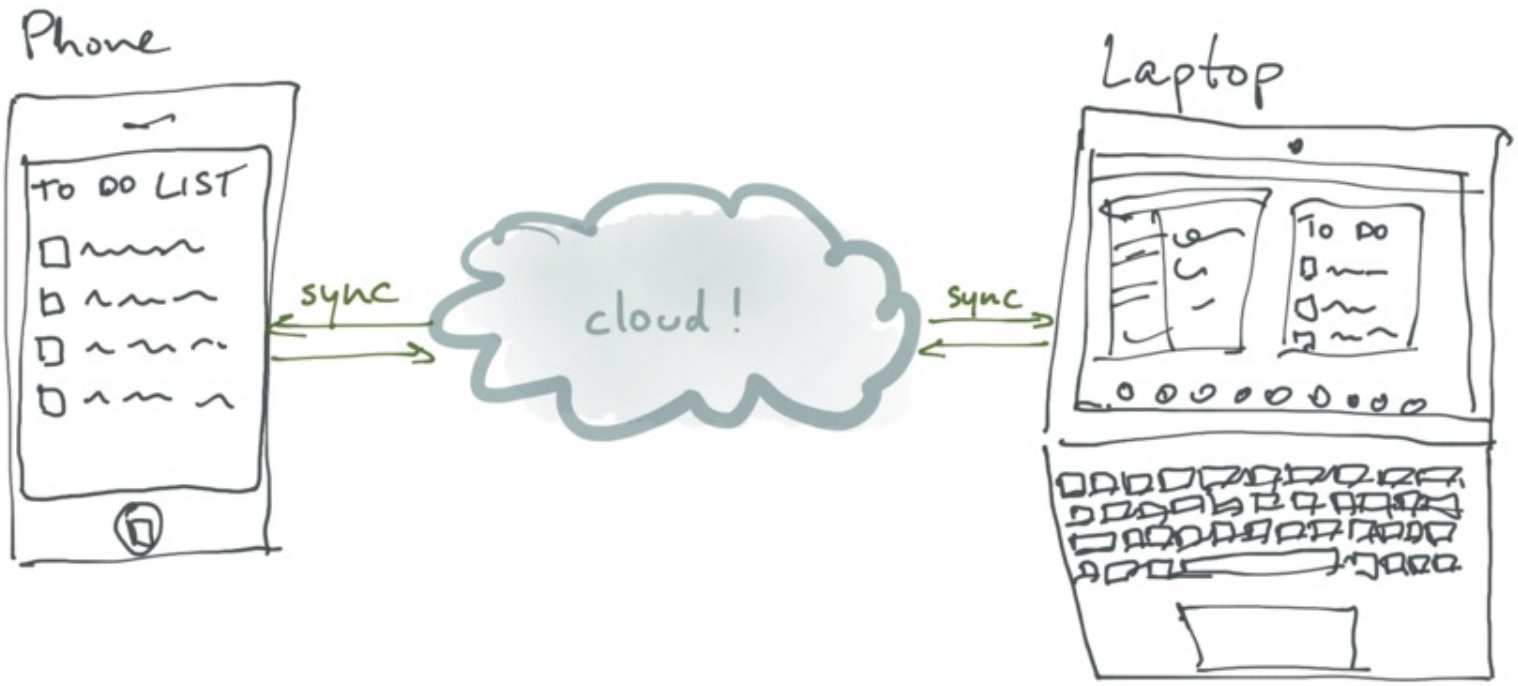
COLLABORATIVE TEXT EDITING • TODAY'S TOPICS

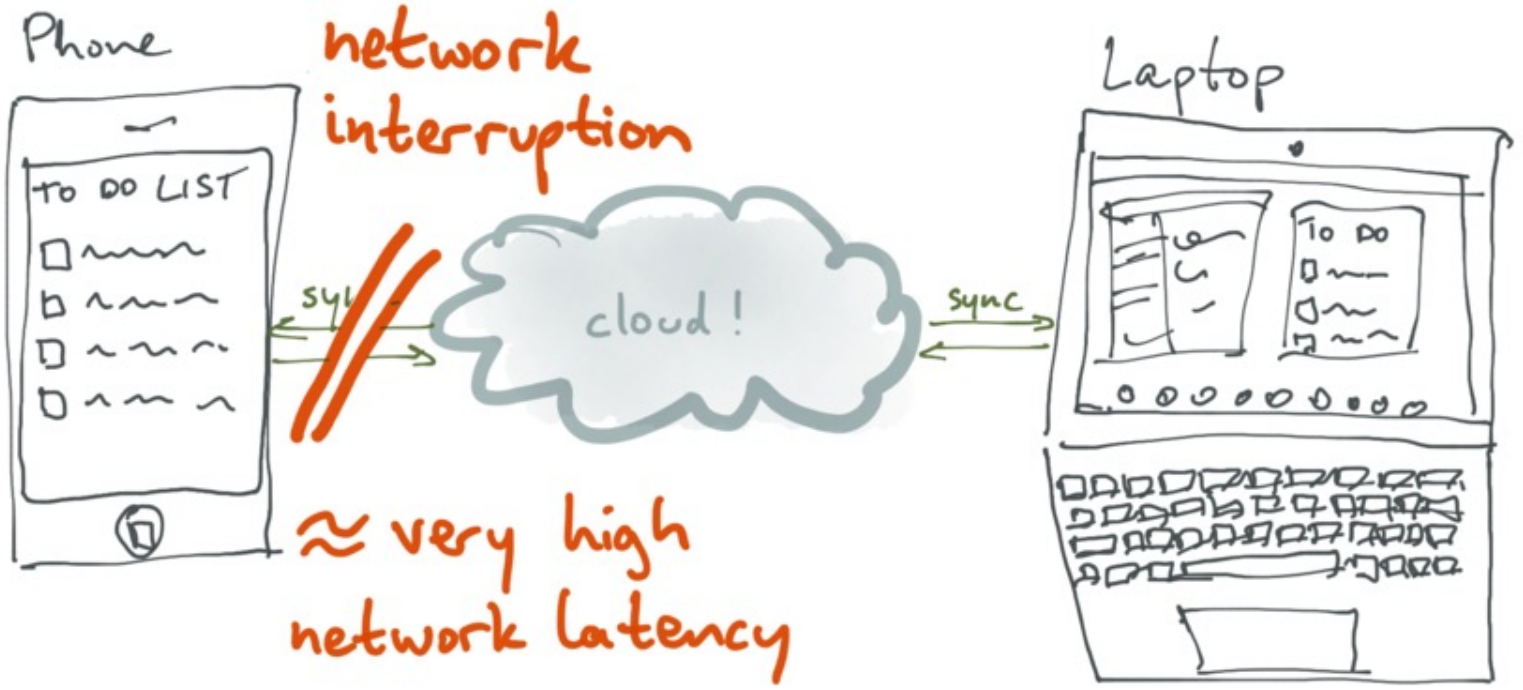
1. Operational transformation
2. Branching and the need for CRDTs
3. Automerge
4. Rich text — Peritext
5. Non-interleaving — Fugue

Convergence guarantee:

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

They are in the same state

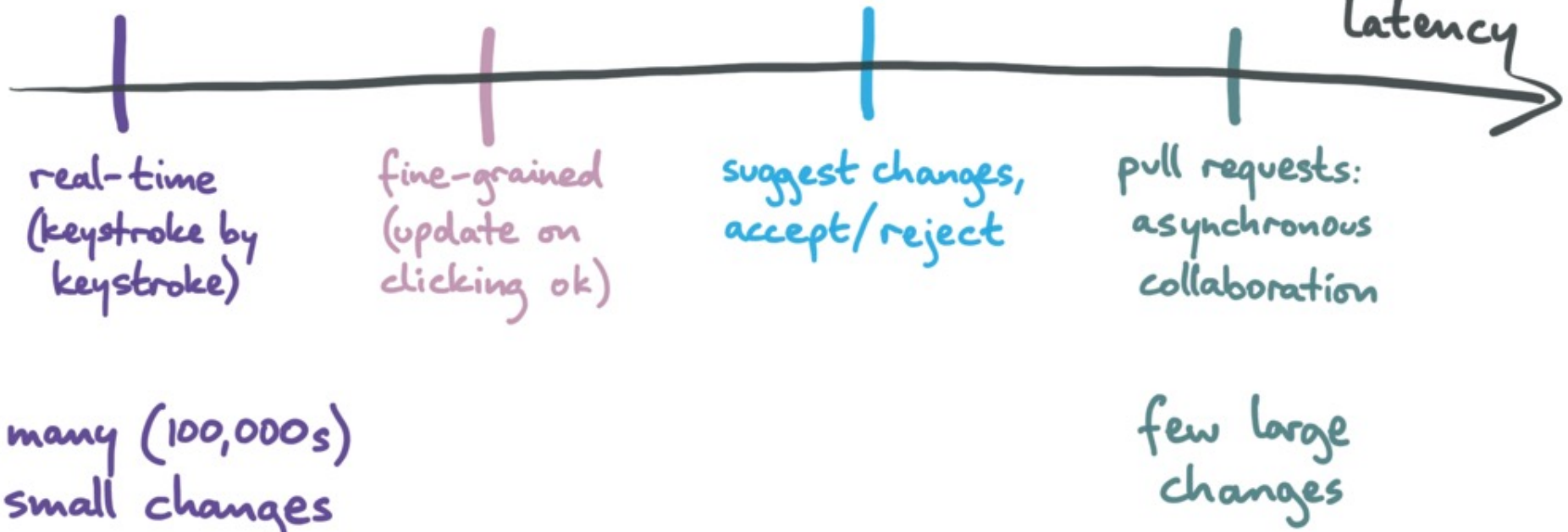

Phone
TO DO LIST
sync
cloud !
sync
Laptop
To DO



Many situations require working offline: remote rural areas (agriculture, fieldwork), train in tunnel, on aeroplane, coffee shop wifi that cuts you off until you buy another drink, military, etc.

Collaboration

Syncing changes between several users



Google Docs demo document Saved to Drive

File Edit View Insert Format Tools Extensions Help

Share Sign in

100% Normal text Arial 11

Version control is not just for code

Use of version control systems (e.g. using Git) is ubiquitous among software developers. However, VCSes don't work well for "binary" files, such as powerpoint decks, spreadsheets, word docs, or vector graphics files – even though they are important to many professions. They need version control for the same reasons as why software developers need it: reviewing what your colleague changed yesterday; tentatively proposing changes to ask others' opinion; or experimenting with a new approach without disrupting others working on other parts of the file.

Microsoft Word's and Google Docs's suggestion mode are not really version control: they are visually messy, they offer no way of making a private branch, and it's difficult to review what has changed after a change has been accepted. Without version control, people use files named "Article revised v5 final (MK edits) FINAL.docx".

https://www.inkandswitch.com/upwelling/

Upwelling

Combining real-time collaboration with version control for writers.

Ink & Switch

Karissa Rae McKelvey

Scott Jenson

Eileen Wagner

Blaine Cook

Martin Kleppmann

March 2023

Collaboration features of existing software do not sufficiently meet the needs of professional non-fiction writers. Real-time collaboration in cloud applications like Google Docs can create stress when writers feel watched by their co-authors, and file-based collaboration can introduce difficulty with versioning and merging edits from different co-authors. Further, in environments where accuracy is crucial, like newsrooms, existing tools make reviewing changes unnecessarily difficult.

In the Upwelling project we have built an experimental editor that aims to satisfy the needs of professional writers and editors. It allows co-authors to collaborate in real time when they wish to, but it also supports work on private drafts that can be shared and merged only when their authors are ready. By combining elements of real-time collaboration with ideas from version control systems, Upwelling supports writers in maintaining their creative privacy and editors in ensuring accurate results.

Version control is not just for code

Use of version control systems (e.g. using Git) is ubiquitous among software developers. However, VCSes don't work well for "binary" files, such as powerpoint decks, spreadsheets, word docs, or vector graphics files – even though they are important to many professions. They need version control for the same reasons as why software developers need it: reviewing what your colleague changed yesterday; tentatively proposing changes to ask others' opinion; or experimenting with a new approach without disrupting others working on other parts of the file.

Microsoft Word's and Google Docs's suggestion mode are not really version control: they are visually messy, they offer no way of making a private branch, and it's difficult to review what has changed after a change has been accepted. Without version control, people use files named "Article revised v5 final (MK edits) FINAL.docx".


HeLo
0.2
0.4
0.6
0.8


edit


HeLLo!
0.2
0.4
0.6
0.7
0.8
0.9

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

$$\cup$$

edit

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

$$=$$

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

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

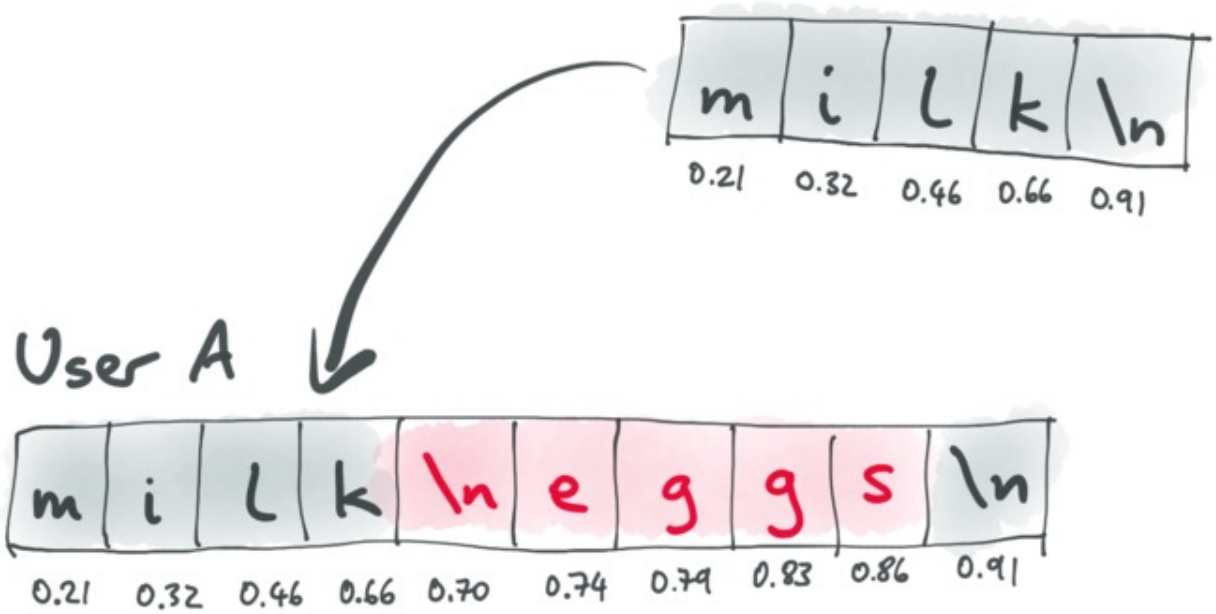
actually a path through a tree...

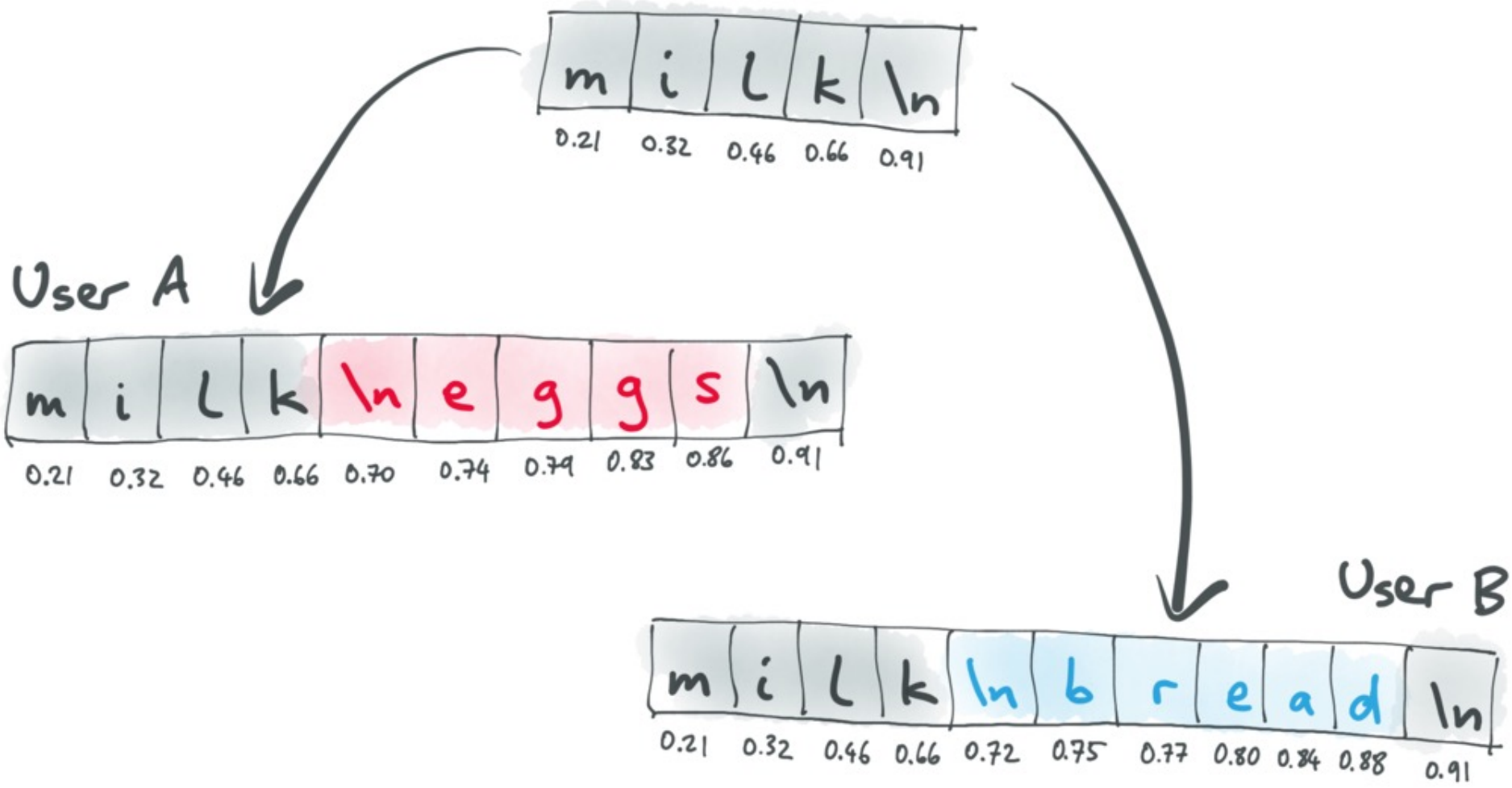
a few dozen bytes?

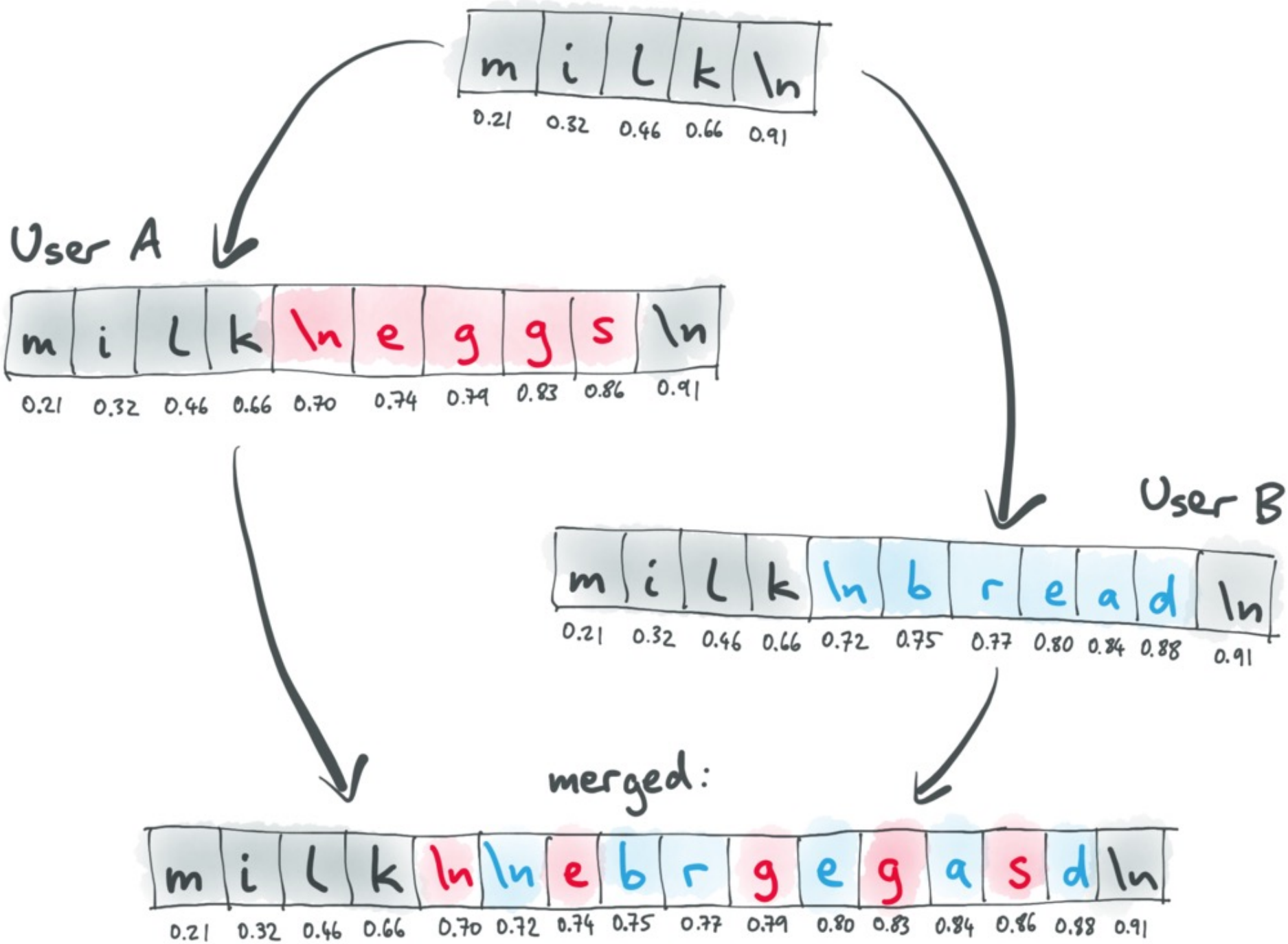
a UUID...
16 bytes binary/
36 bytes hex?

the actual ASCII/ Unicode character:
1 byte


m
i
l
k
\n
0.21
0.32
0.46
0.66
0.91


m i l k \n
0.21 0.32 0.46 0.66 0.91
User A
m i l k \n e g g s \n
0.21 0.32 0.46 0.66 0.70 0.74 0.79 0.83 0.86 0.91


m i l k \n
0.21 0.32 0.46 0.66 0.91
User A
m i l k \n e g g s \n
0.21 0.32 0.46 0.66 0.70 0.74 0.79 0.83 0.86 0.91
User B
m i l k \n b r e a d \n
0.21 0.32 0.46 0.66 0.72 0.75 0.77 0.80 0.84 0.88 0.91

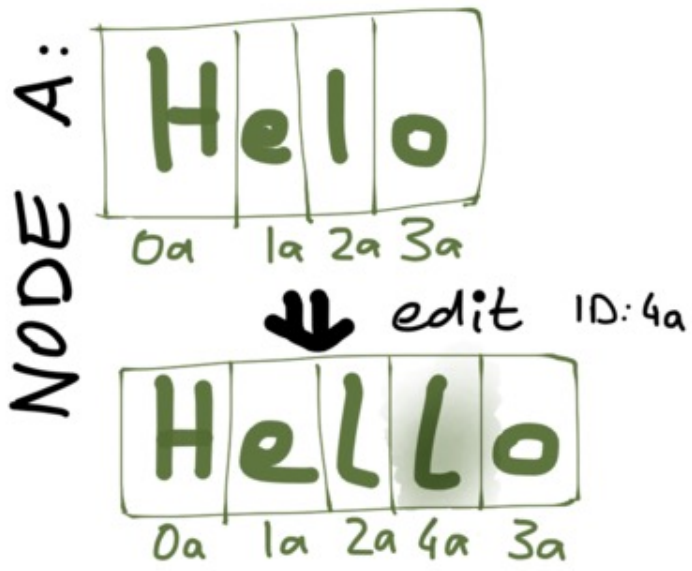

m i l k \n
0.21 0.32 0.46 0.66 0.91
User A
m i l k \n e g g s \n
0.21 0.32 0.46 0.66 0.70 0.74 0.79 0.83 0.86 0.91
User B
m i l k \n b r e a d \n
0.21 0.32 0.46 0.66 0.72 0.75 0.77 0.80 0.84 0.88 0.91
merged:
m i l k \n \n e b r g e g a s d \n
0.21 0.32 0.46 0.66 0.70 0.72 0.74 0.75 0.77 0.79 0.80 0.83 0.84 0.86 0.88 0.91

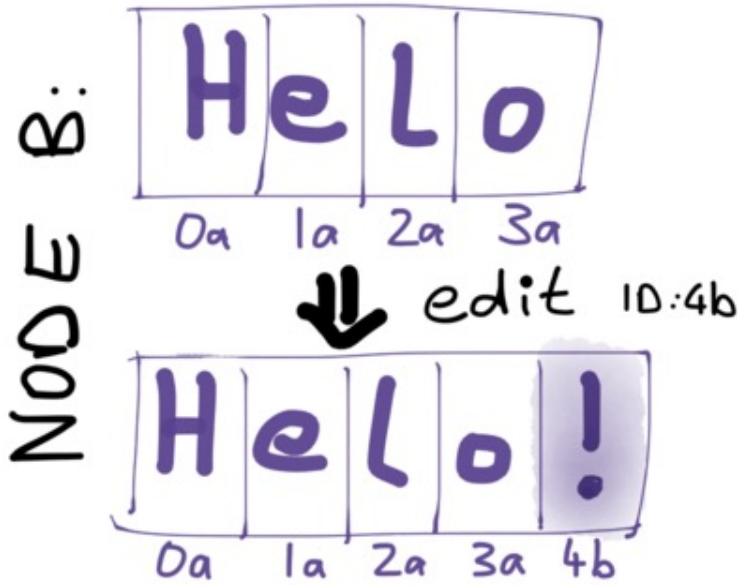
ORDERED LIST CRDT (NUTSHELL)

NODE A:

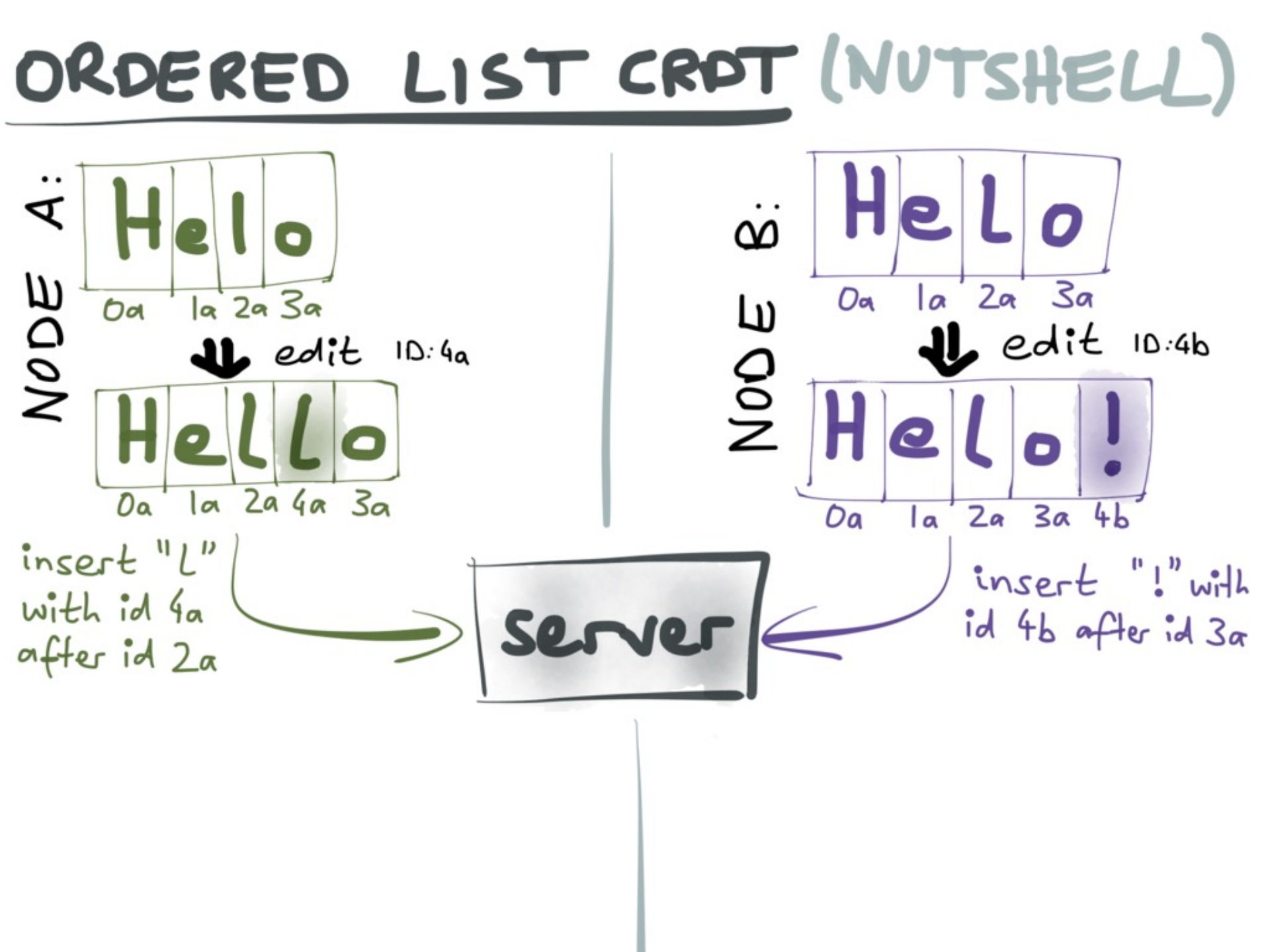
NODE B:

ORDERED LIST CRDT (NUTSHELL)

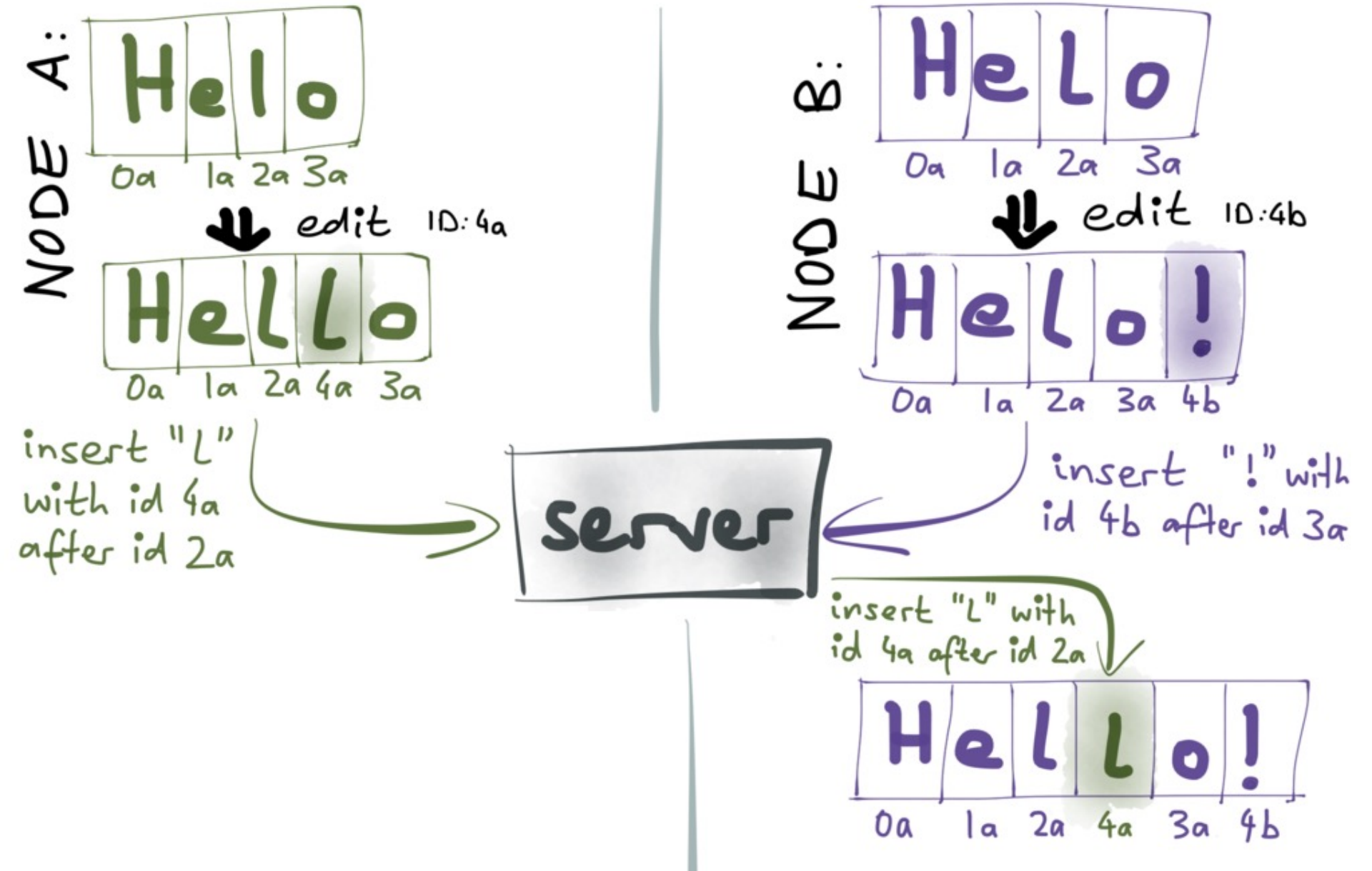




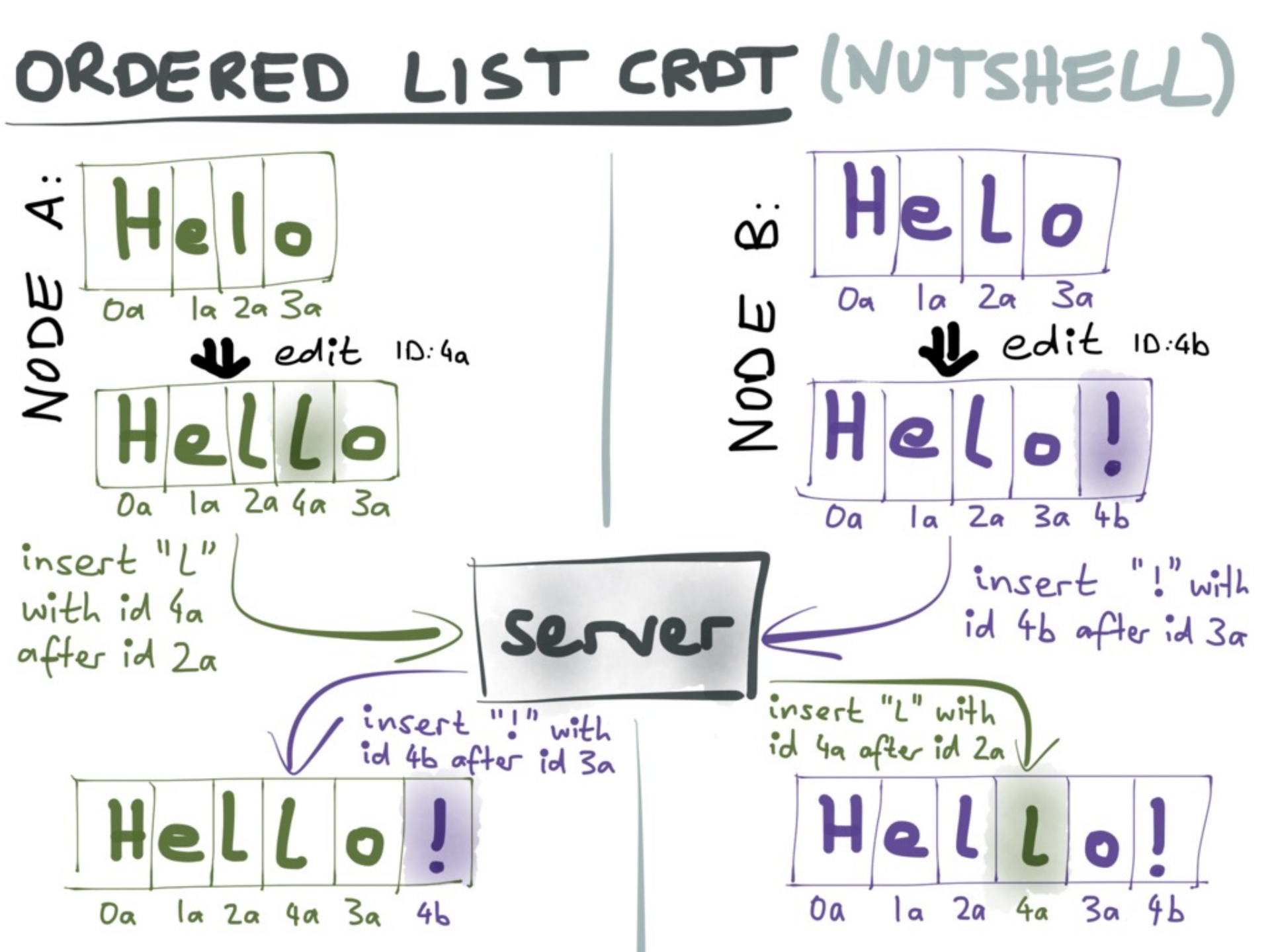
ORDERED LIST CRDT (NUTSHELL)



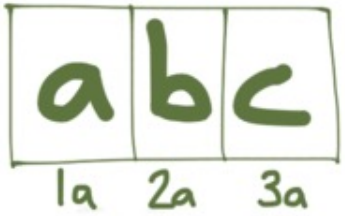
ORDERED LIST CRDT (NUTSHELL)

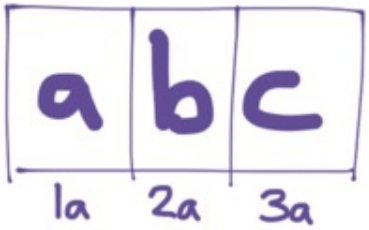


ORDERED LIST CRDT (NUTSHELL)

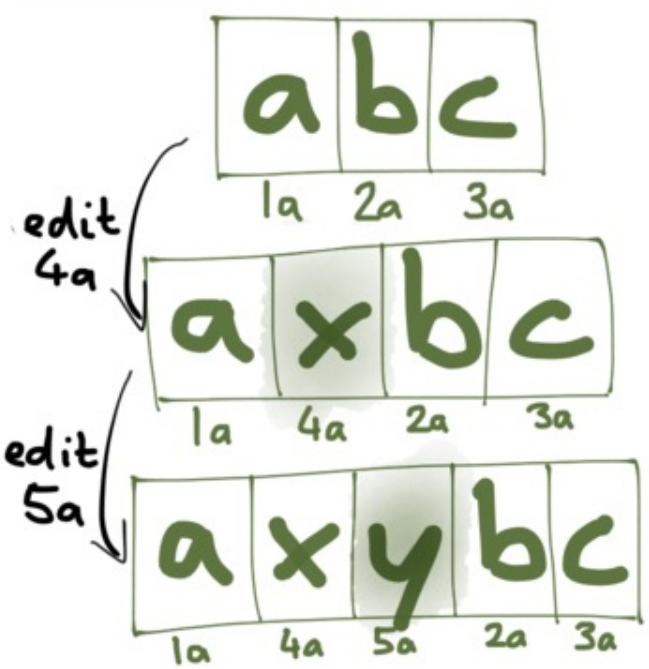


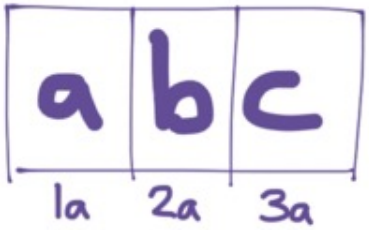
INSERTING IN THE SAME PLACE



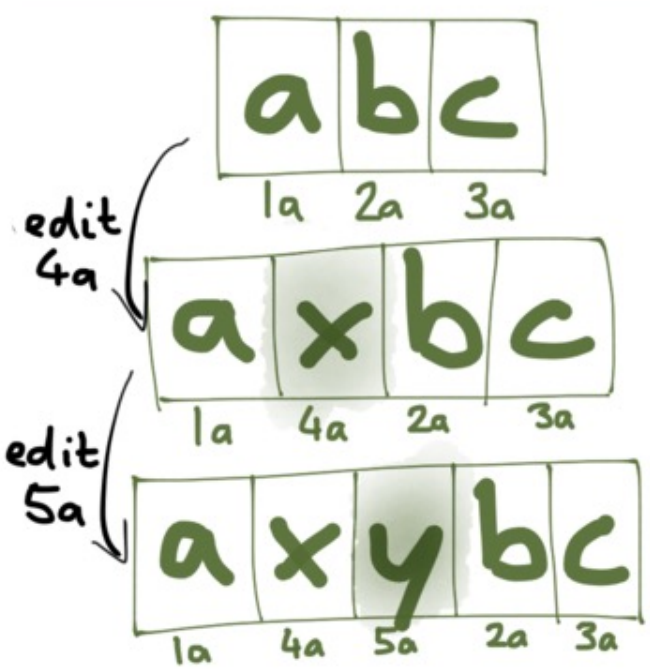


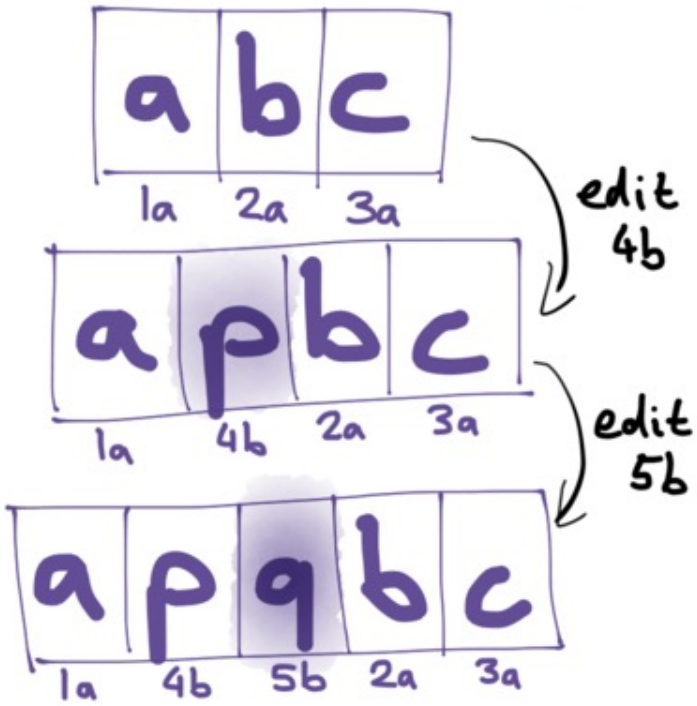
INSERTING IN THE SAME PLACE



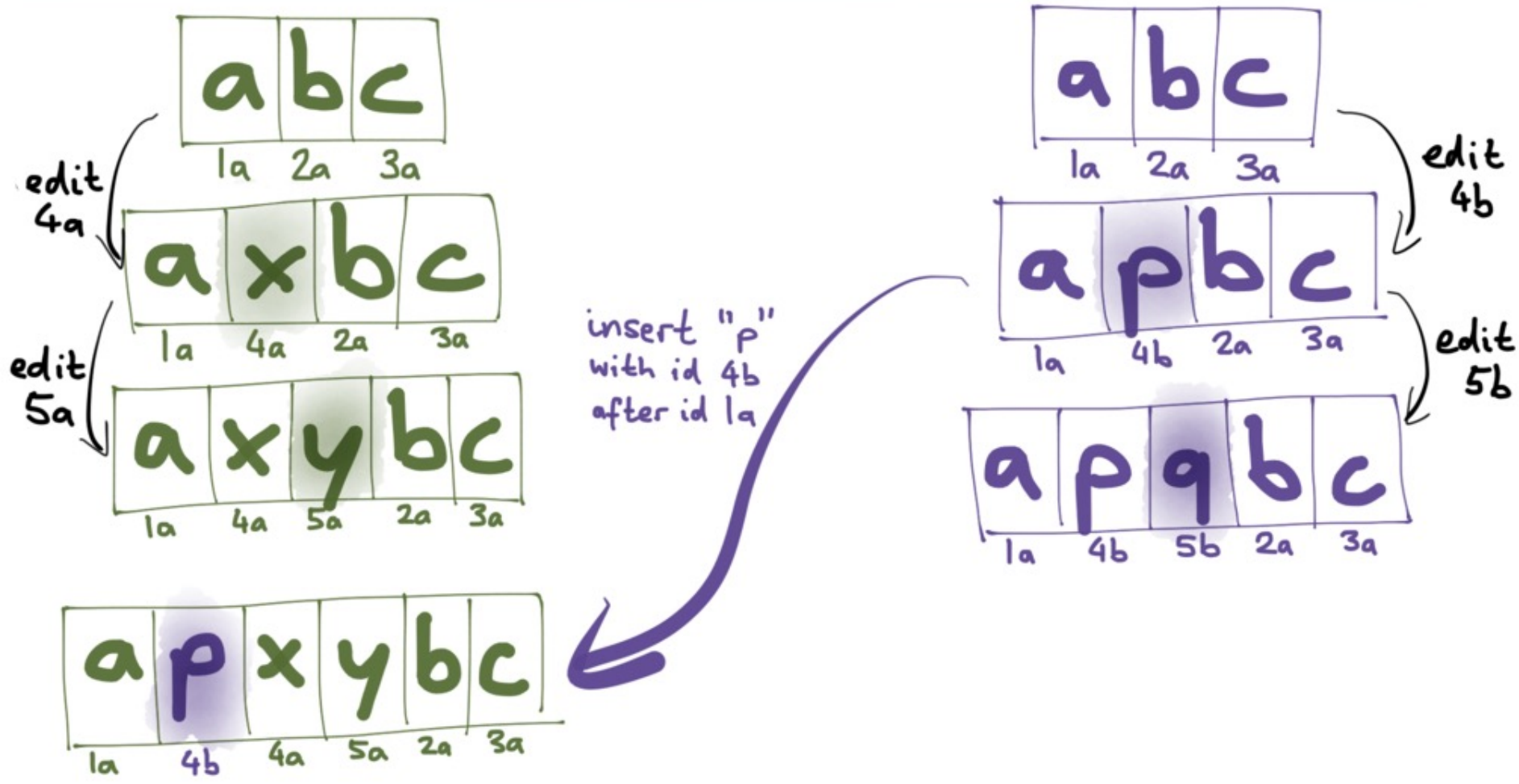


INSERTING IN THE SAME PLACE

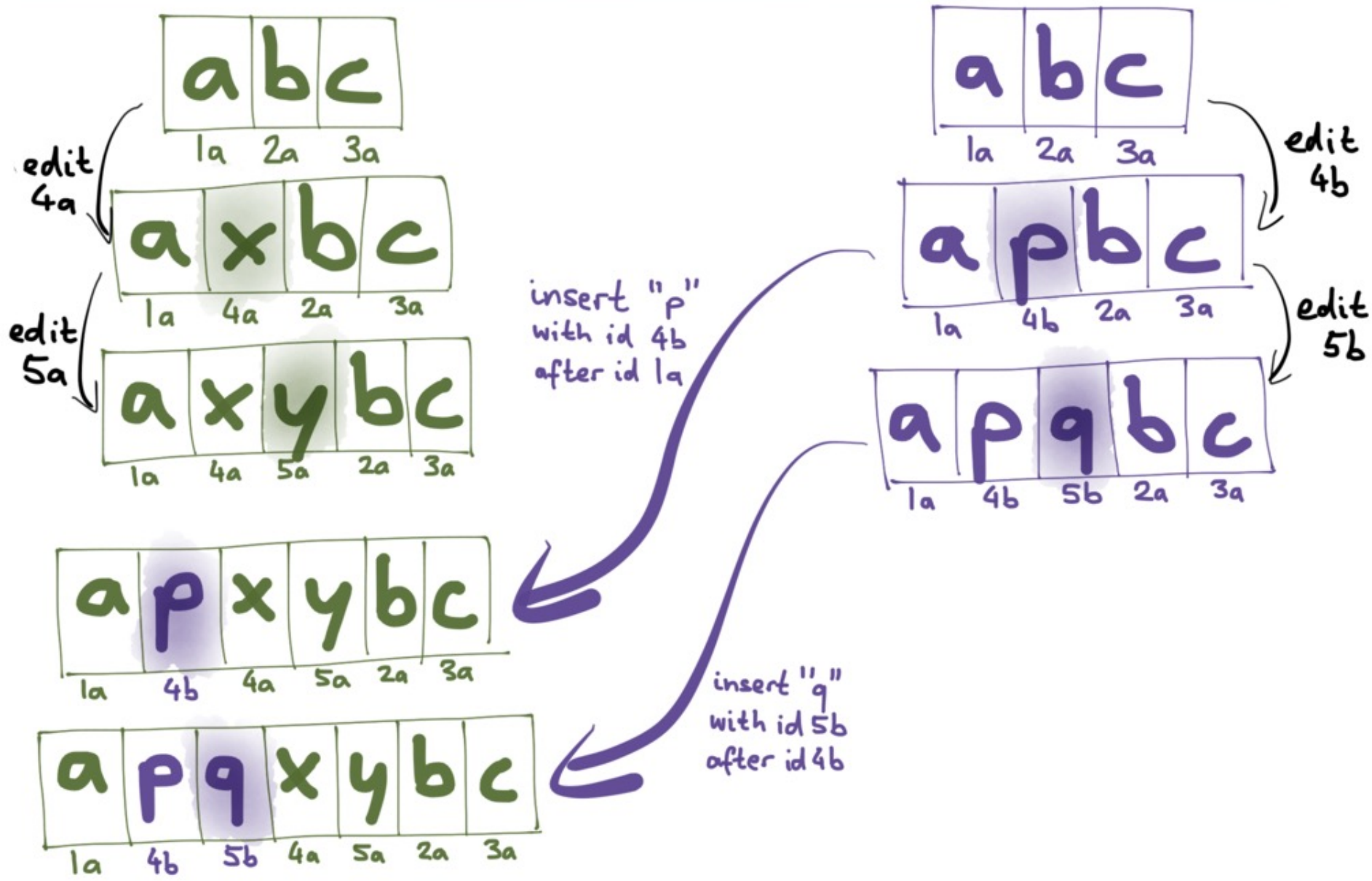




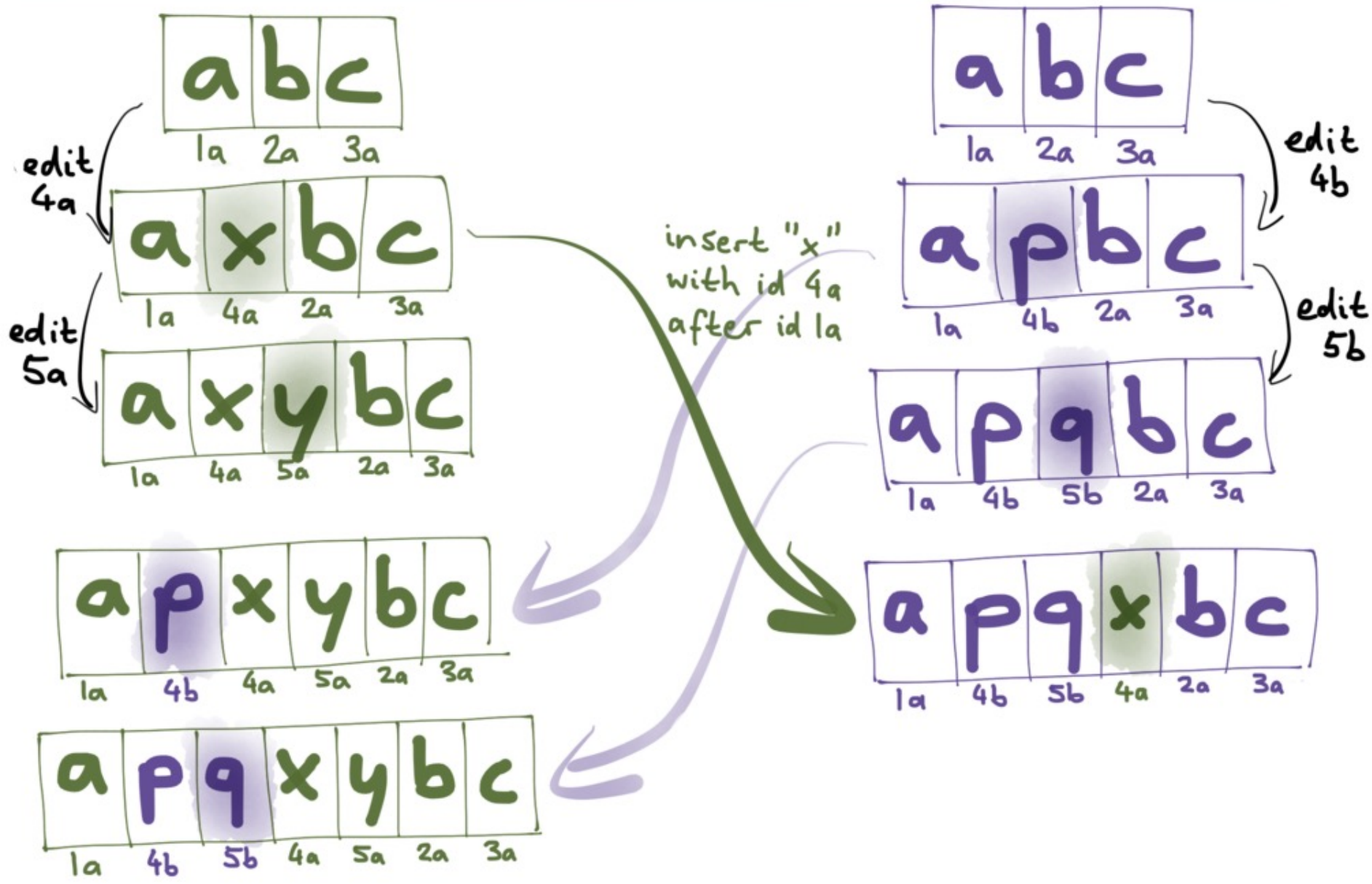
INSERTING IN THE SAME PLACE



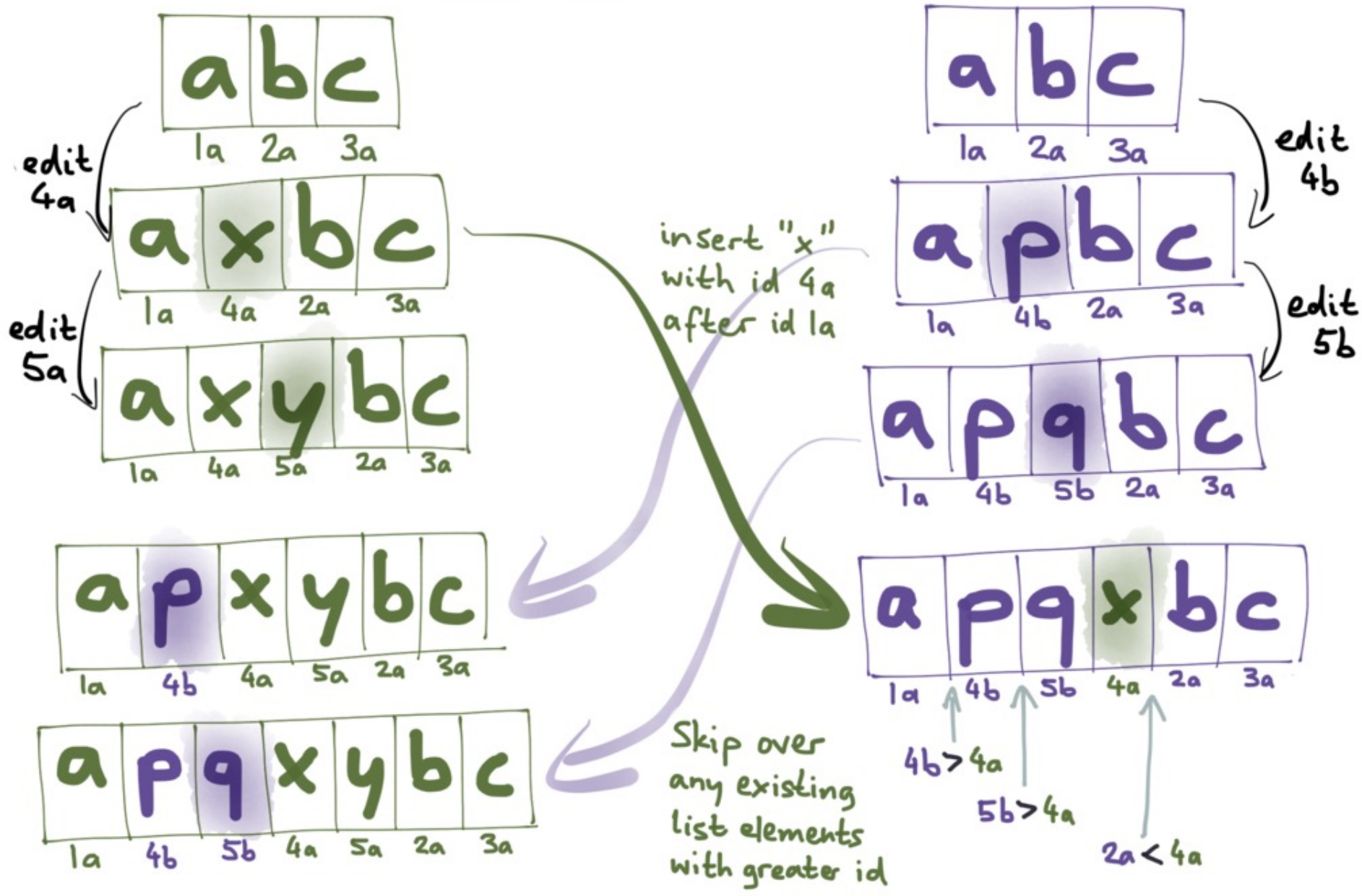
INSERTING IN THE SAME PLACE



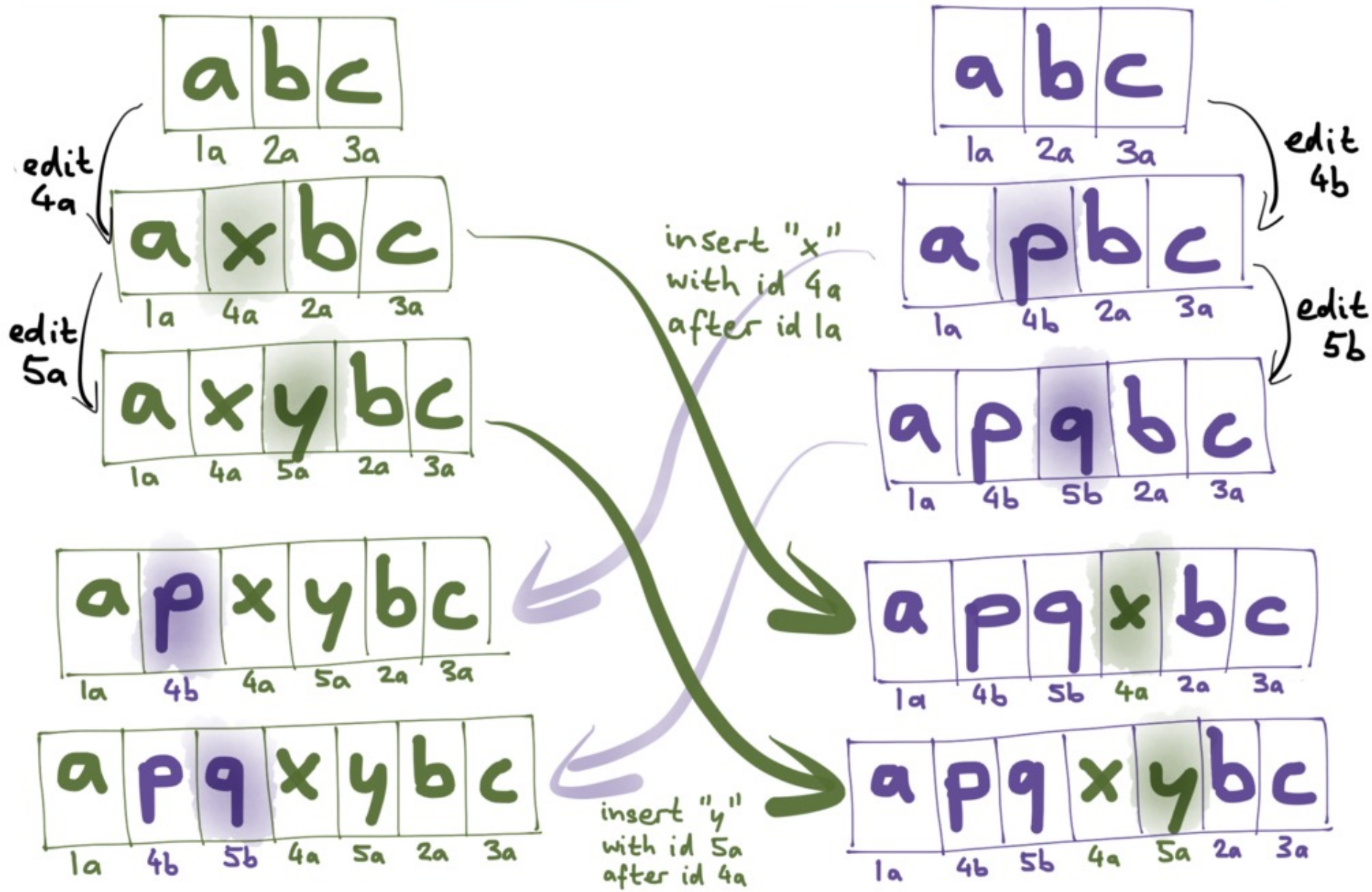
INSERTING IN THE SAME PLACE



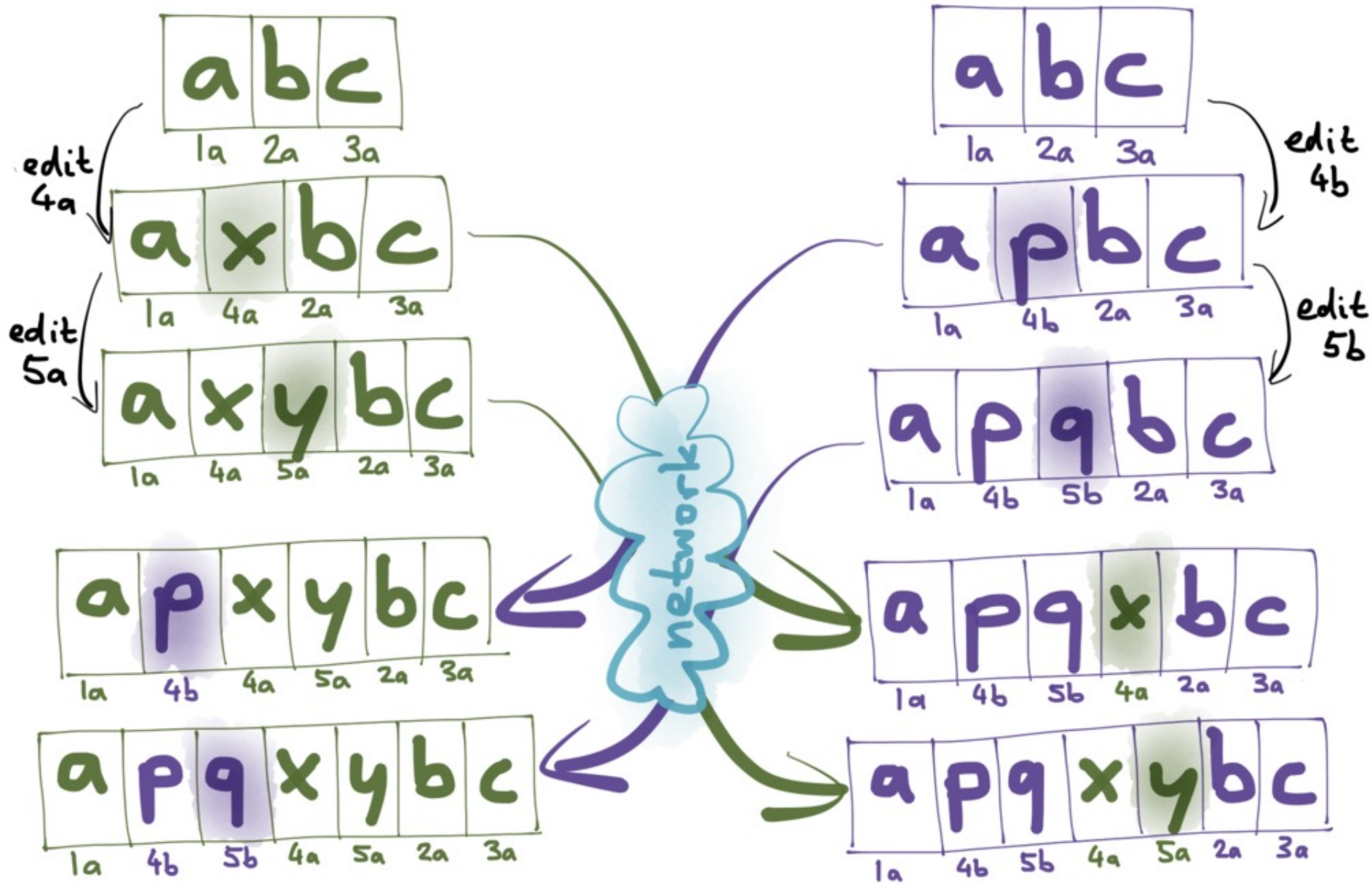
INSERTING IN THE SAME PLACE



INSERTING IN THE SAME PLACE



INSERTING IN THE SAME PLACE

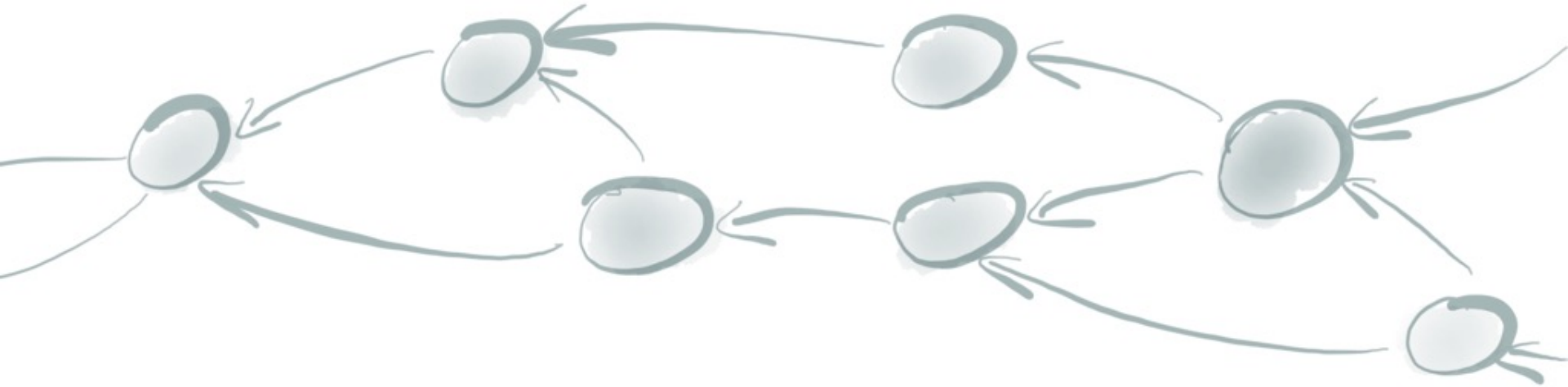


	OT	CRDT
Real-time collaboration		
Offline support		
Branching/merging		
Arbitrary JSON		
Rich text		
Edit history		
Metadata overhead		
Decentralisation		

	OT	CRDT
Real-time collaboration		
Offline support		
Branching/merging		
Arbitrary JSON		
Rich text		
Edit history		 Automerge turning these into
Metadata overhead		 
Decentralisation		

COLLABORATIVE TEXT EDITING • TODAY'S TOPICS

1. Operational transformation
2. Branching and the need for CRDTs
3. Automerge
4. Rich text — Peritext
5. Non-interleaving — Fugue



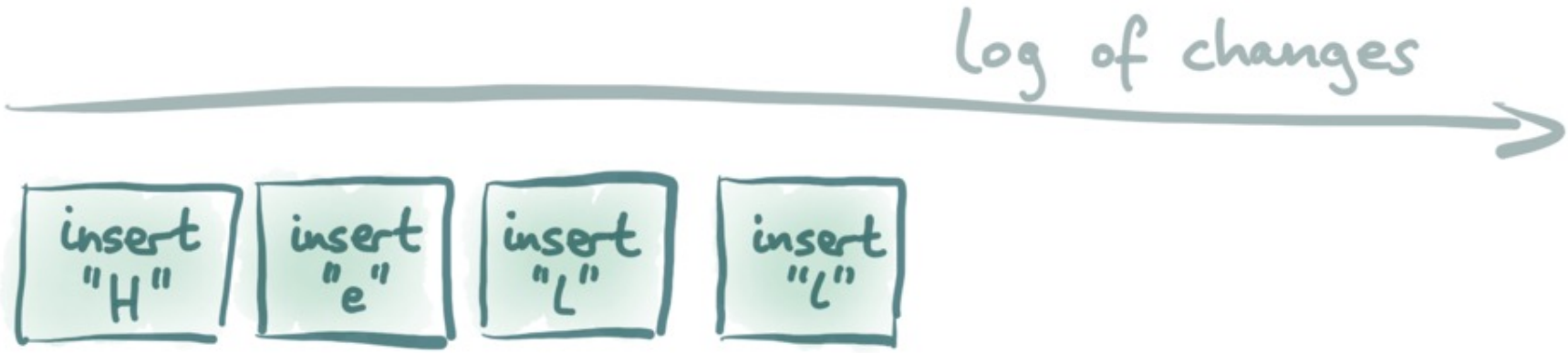
Automerge

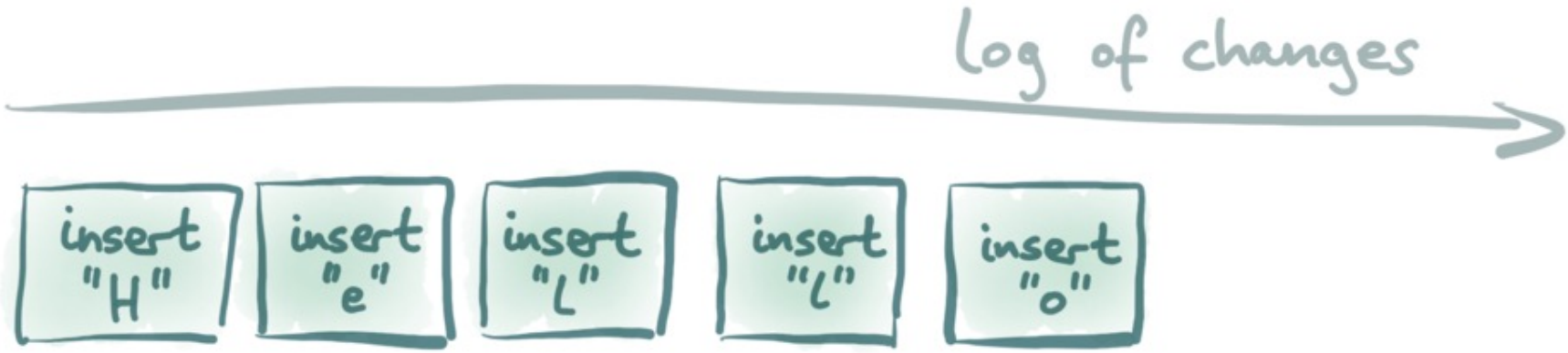
https://github.com/automerge/automerge

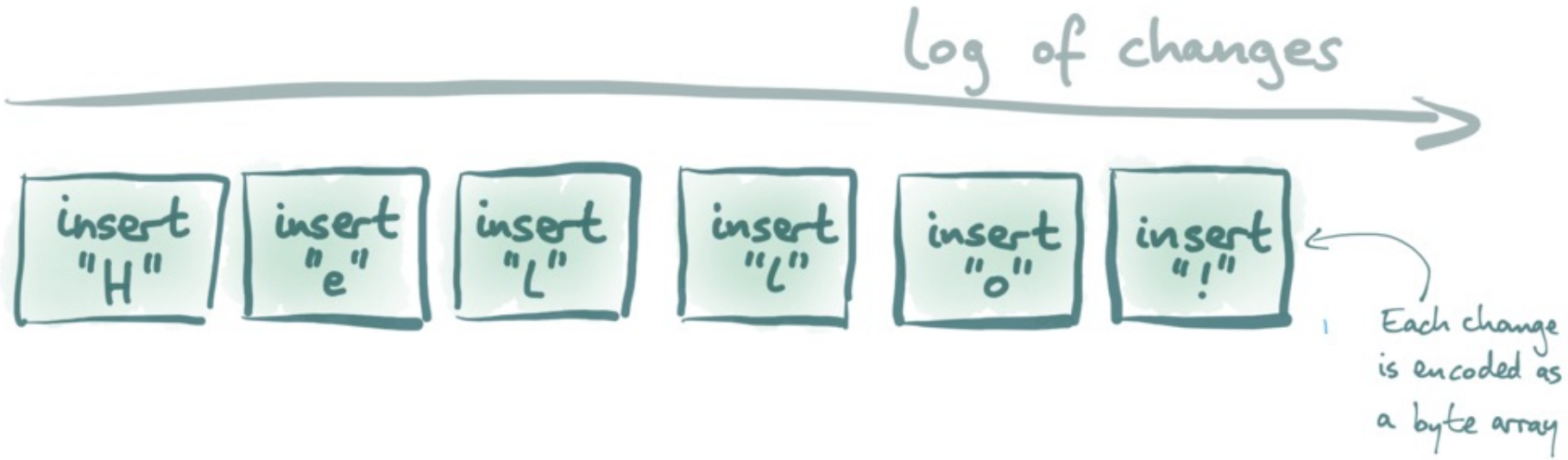

log of changes
insert
"H"

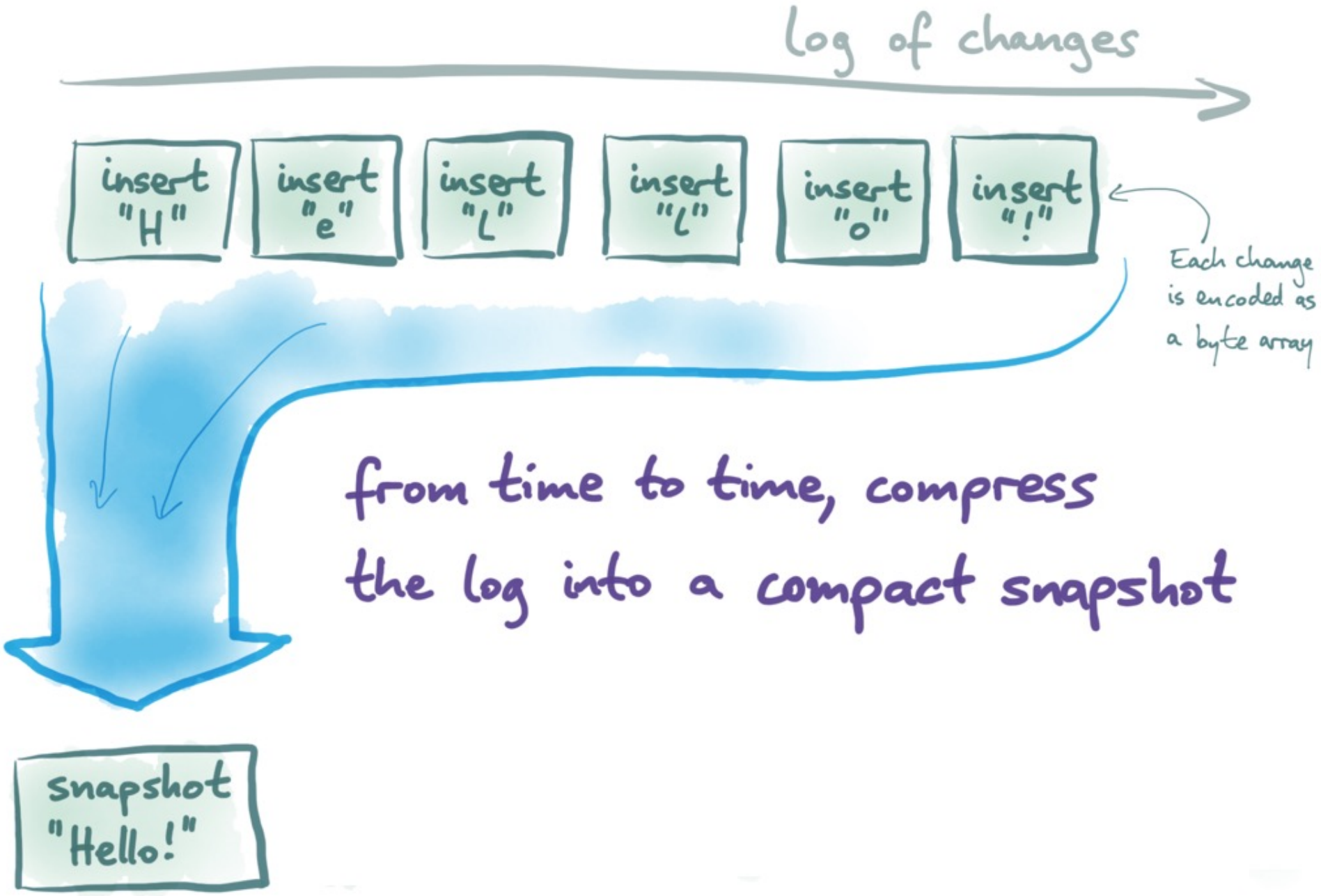

log of changes
insert "H"
insert "e"

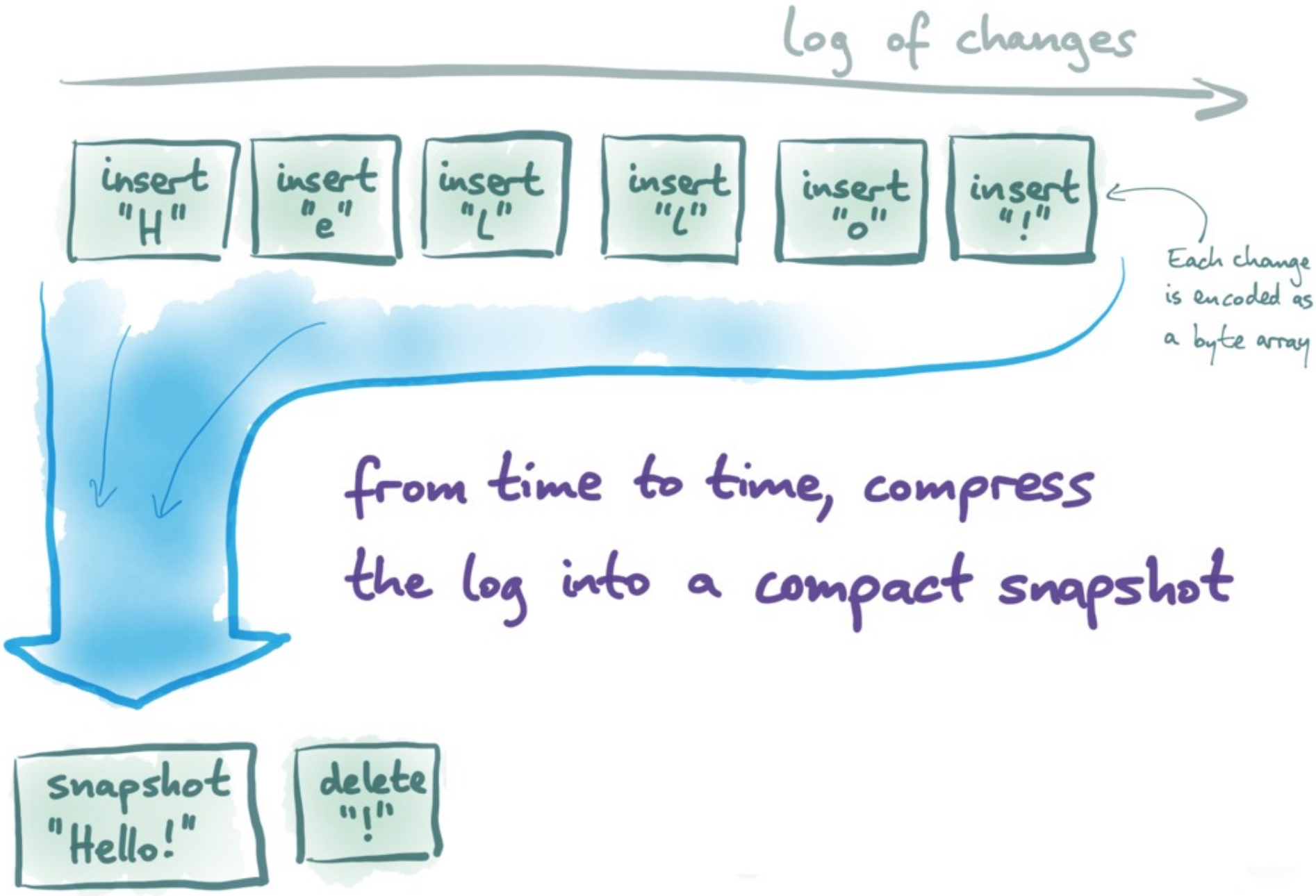

log of changes
insert "H"
insert "e"
insert "L"

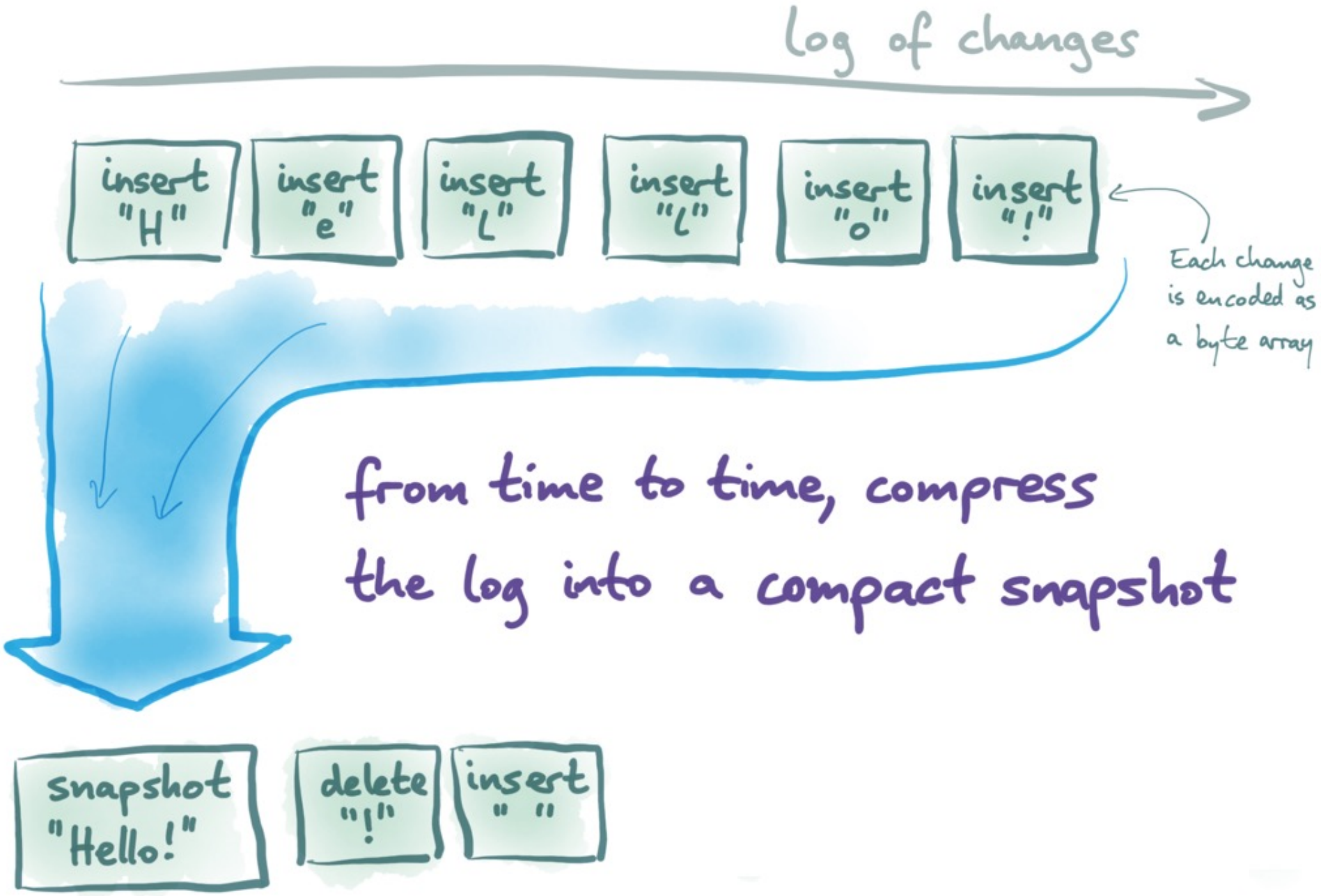

log of changes
insert "H"
insert "e"
insert "l"
insert "l"

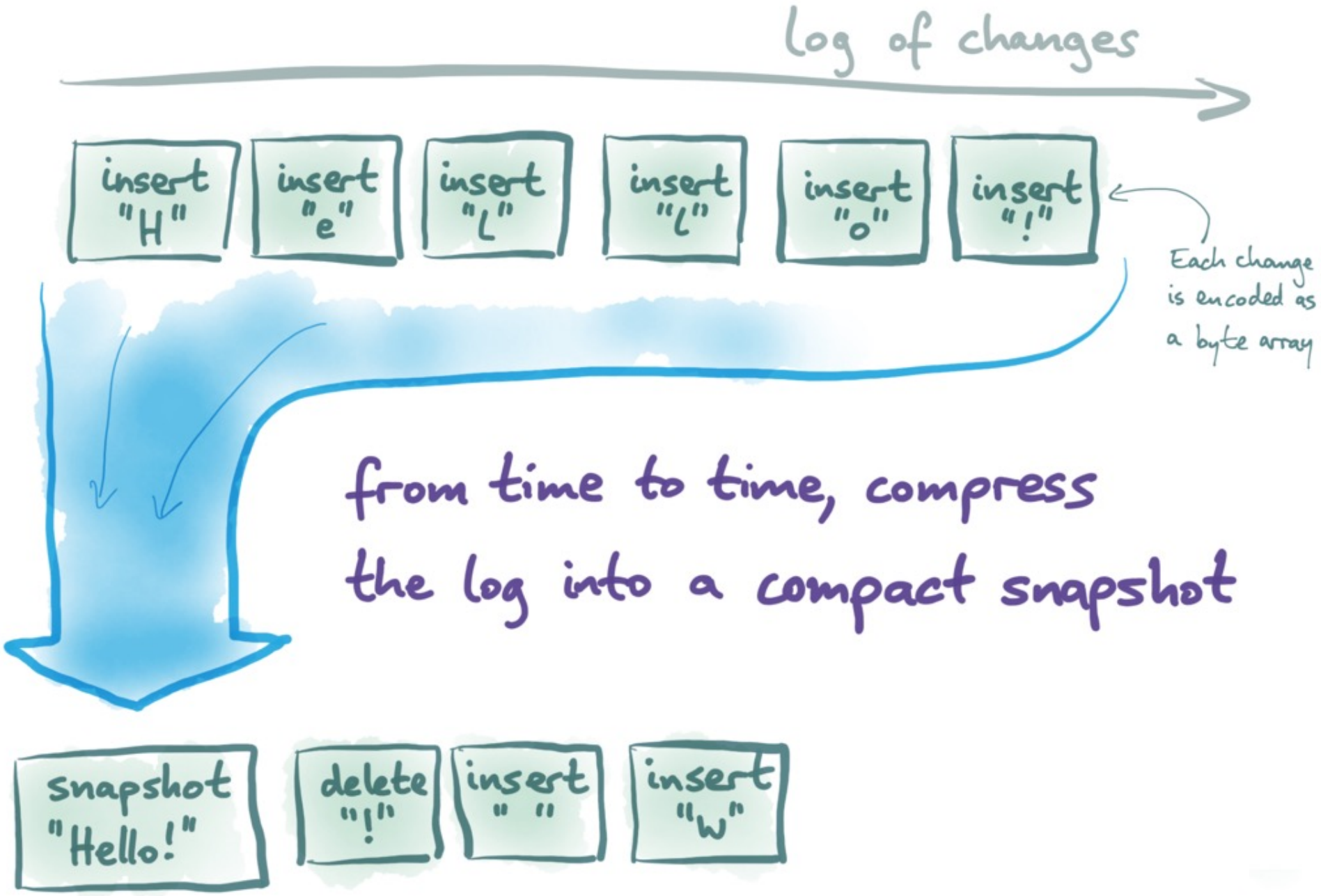

log of changes
insert "H"
insert "e"
insert "l"
insert "l"
insert "o"

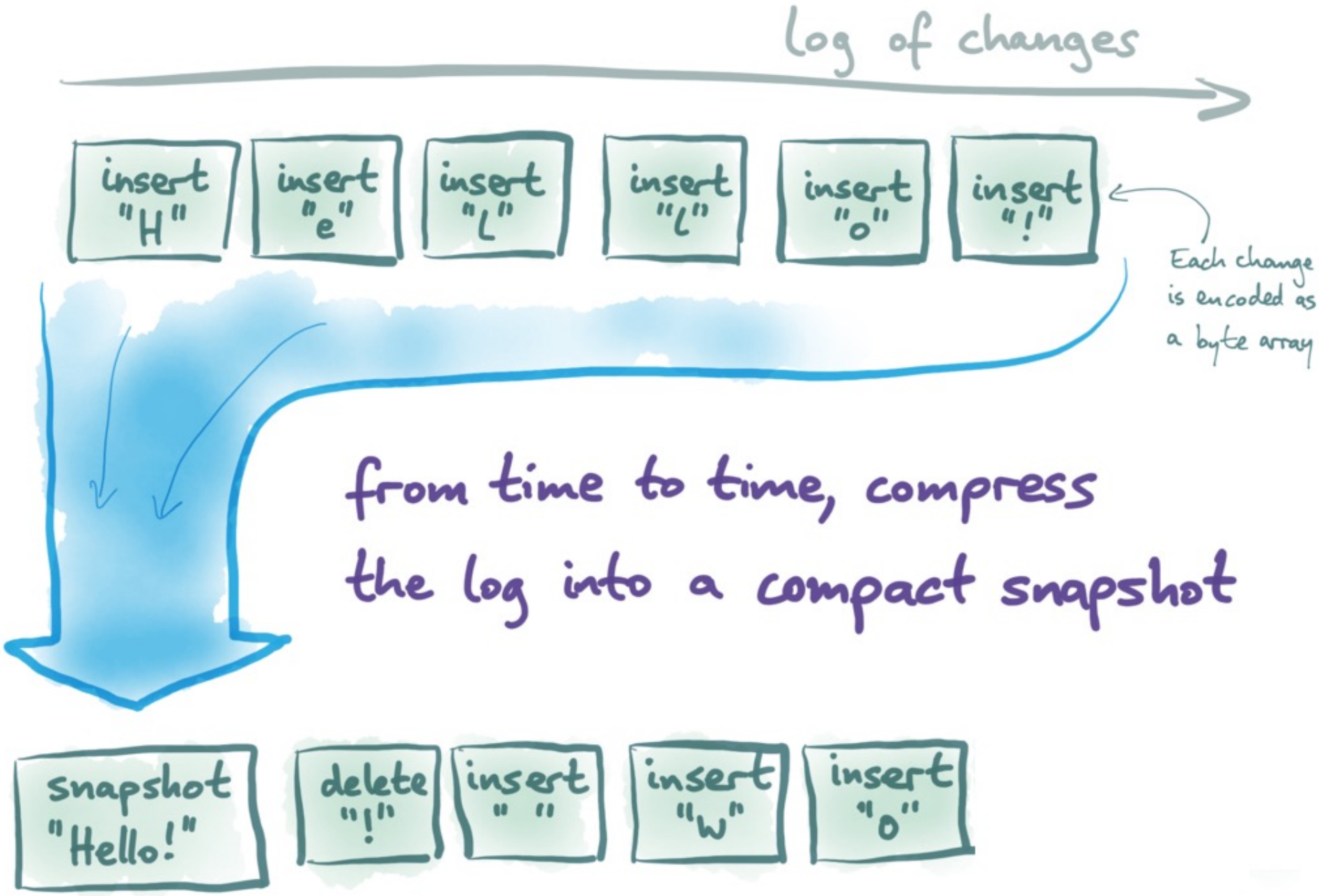

log of changes
insert "H"
insert "e"
insert "l"
insert "l"
insert "o"
insert "!"
Each change is encoded as a byte array

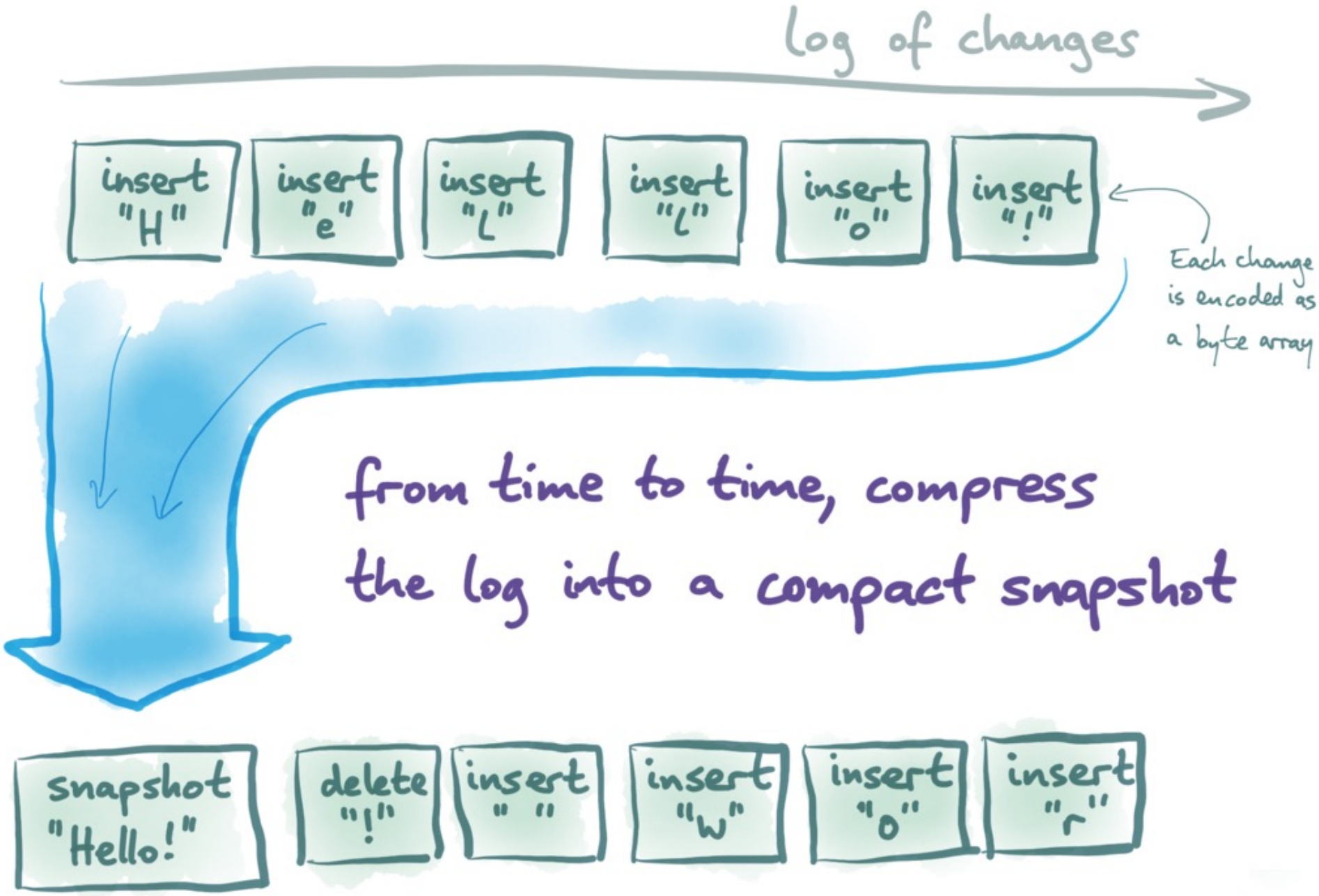

log of changes
insert "H"
insert "e"
insert "l"
insert "l"
insert "o"
insert "!"
Each change is encoded as a byte array
from time to time, compress the log into a compact snapshot
snapshot "Hello!"

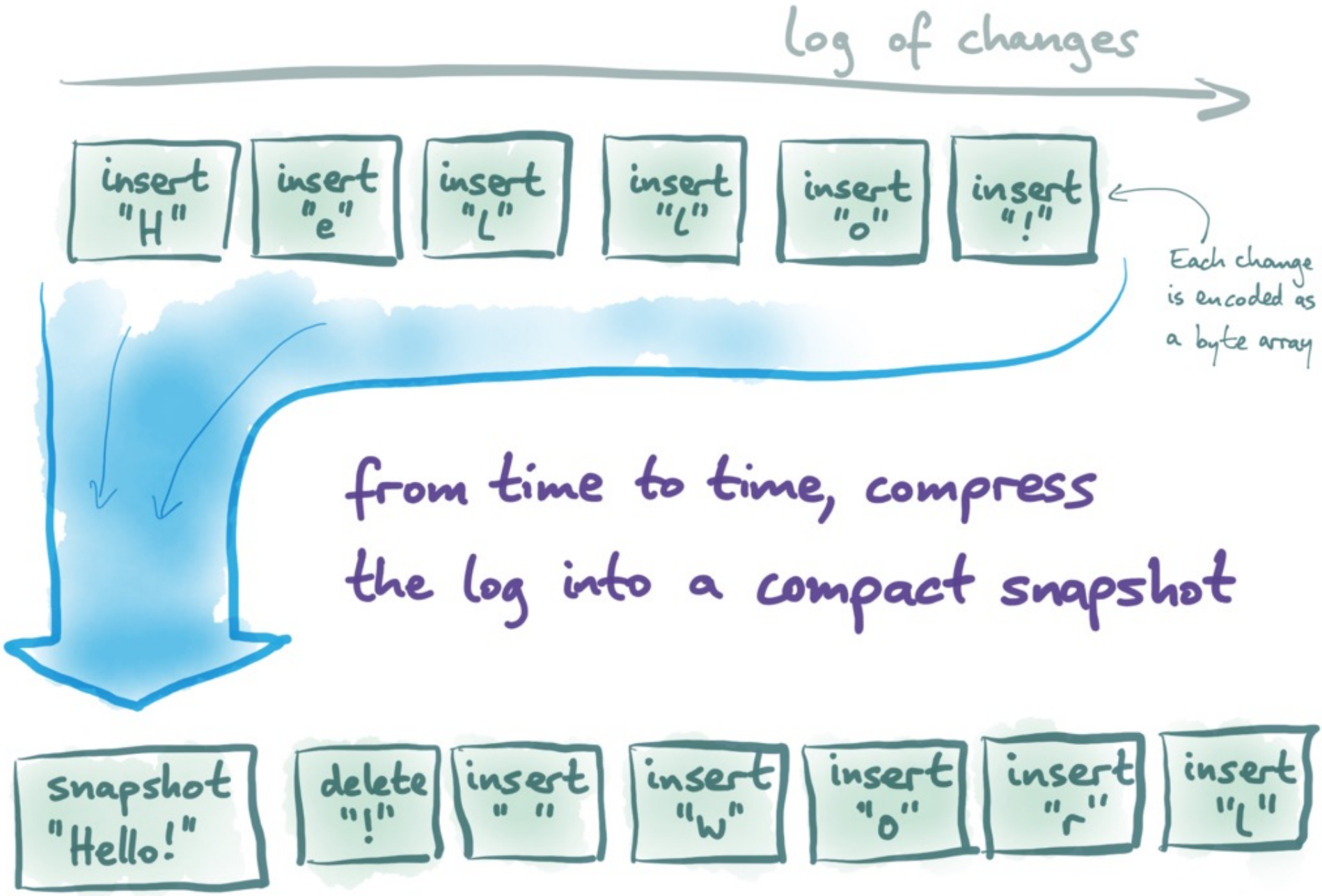

log of changes
insert "H"
insert "e"
insert "l"
insert "l"
insert "o"
insert "!"
Each change is encoded as a byte array
from time to time, compress the log into a compact snapshot
snapshot "Hello!"
delete "!"


log of changes
insert "H"
insert "e"
insert "l"
insert "l"
insert "o"
insert "!"
Each change is encoded as a byte array
from time to time, compress the log into a compact snapshot
snapshot "Hello!"
delete "!"
insert " "


log of changes
insert "H"
insert "e"
insert "l"
insert "l"
insert "o"
insert "!"
Each change is encoded as a byte array
from time to time, compress the log into a compact snapshot
snapshot "Hello!"
delete "!"
insert " "
insert "w"


log of changes
insert "H"
insert "e"
insert "l"
insert "l"
insert "o"
insert "!"
Each change is encoded as a byte array
from time to time, compress the log into a compact snapshot
snapshot "Hello!"
delete "!"
insert " "
insert "w"
insert "o"


log of changes
insert "H"
insert "e"
insert "l"
insert "l"
insert "o"
insert "!"
Each change is encoded as a byte array
from time to time, compress the log into a compact snapshot
snapshot "Hello!"
delete "!"
insert " "
insert "w"
insert "o"
insert "r"


log of changes
insert "H"
insert "e"
insert "l"
insert "l"
insert "o"
insert "!"
Each change is encoded as a byte array
from time to time, compress the log into a compact snapshot
snapshot "Hello!"
delete "!"
insert " "
insert "w"
insert "o"
insert "r"
insert "l"

Automerge compression benchmark

Test document: keystroke-by-keystroke editing trace of a text file (LaTeX source of a research paper)

- 182,315 single-character insertions
- 77,463 single-character deletions

Final text (uncompressed, no edit history): 105 kB

Automerge compression benchmark

Test document: keystroke-by-keystroke editing trace of a text file (LaTeX source of a research paper)

- 182,315 single-character insertions
- 77,463 single-character deletions

Final text (uncompressed, no edit history): 105 kB

Full edit history (columnar compression): 184 kB

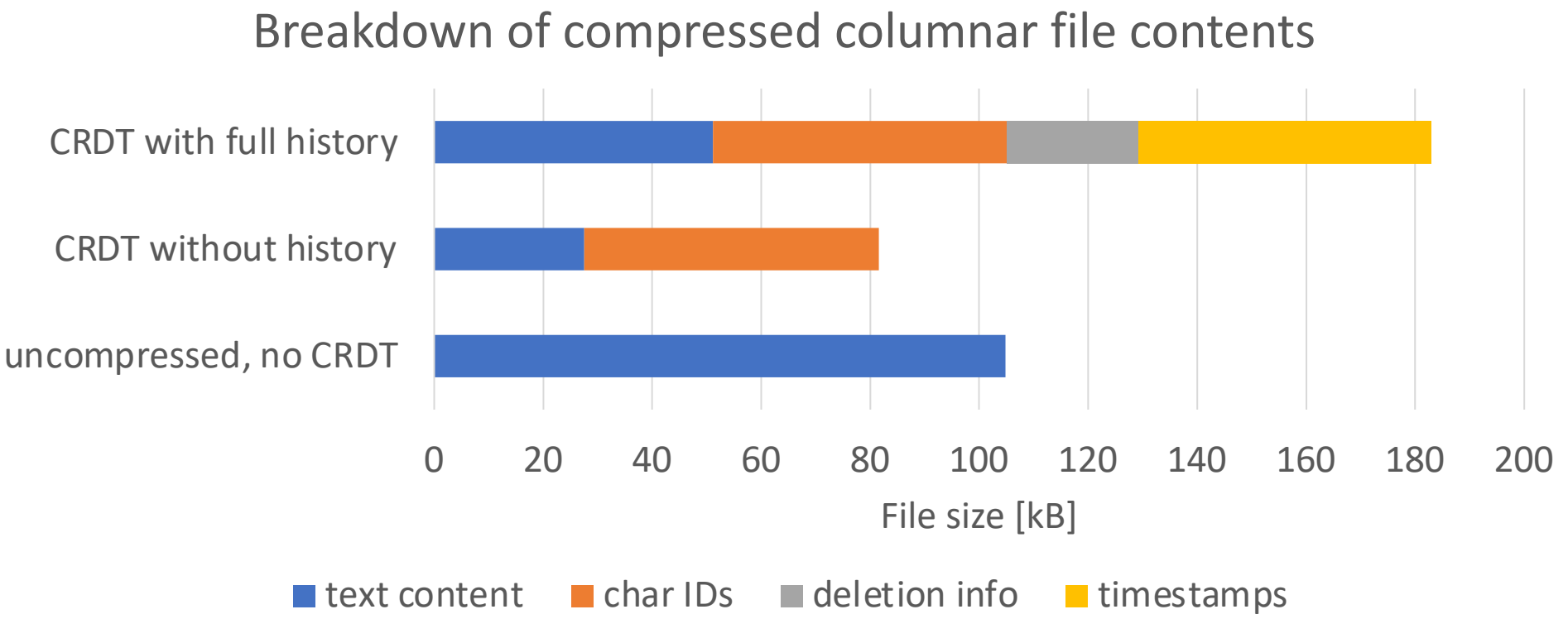
(0.7 bytes/operation!)

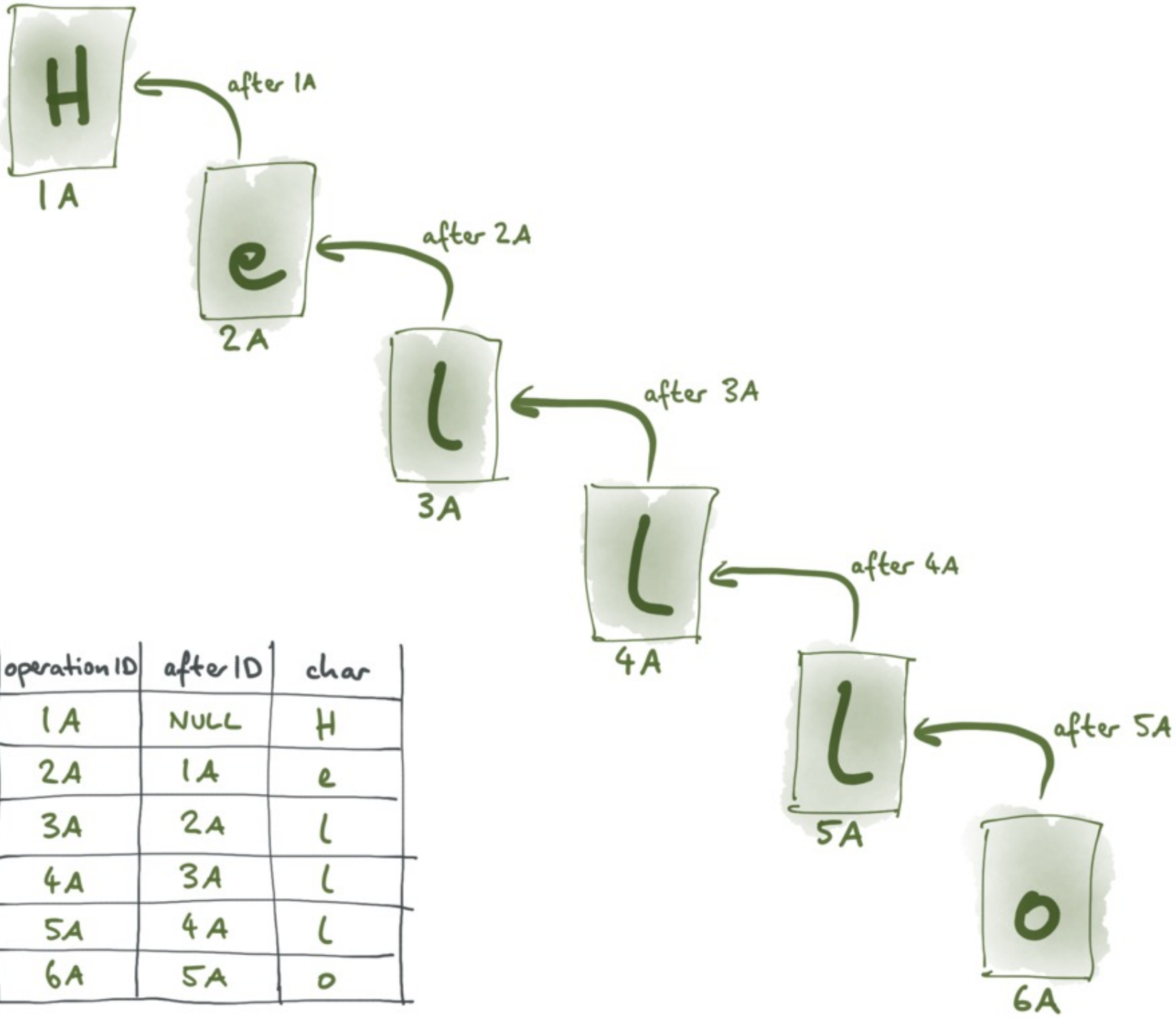
Can look at any past document version, diffing, branching, merging ...

Automerge compression benchmark

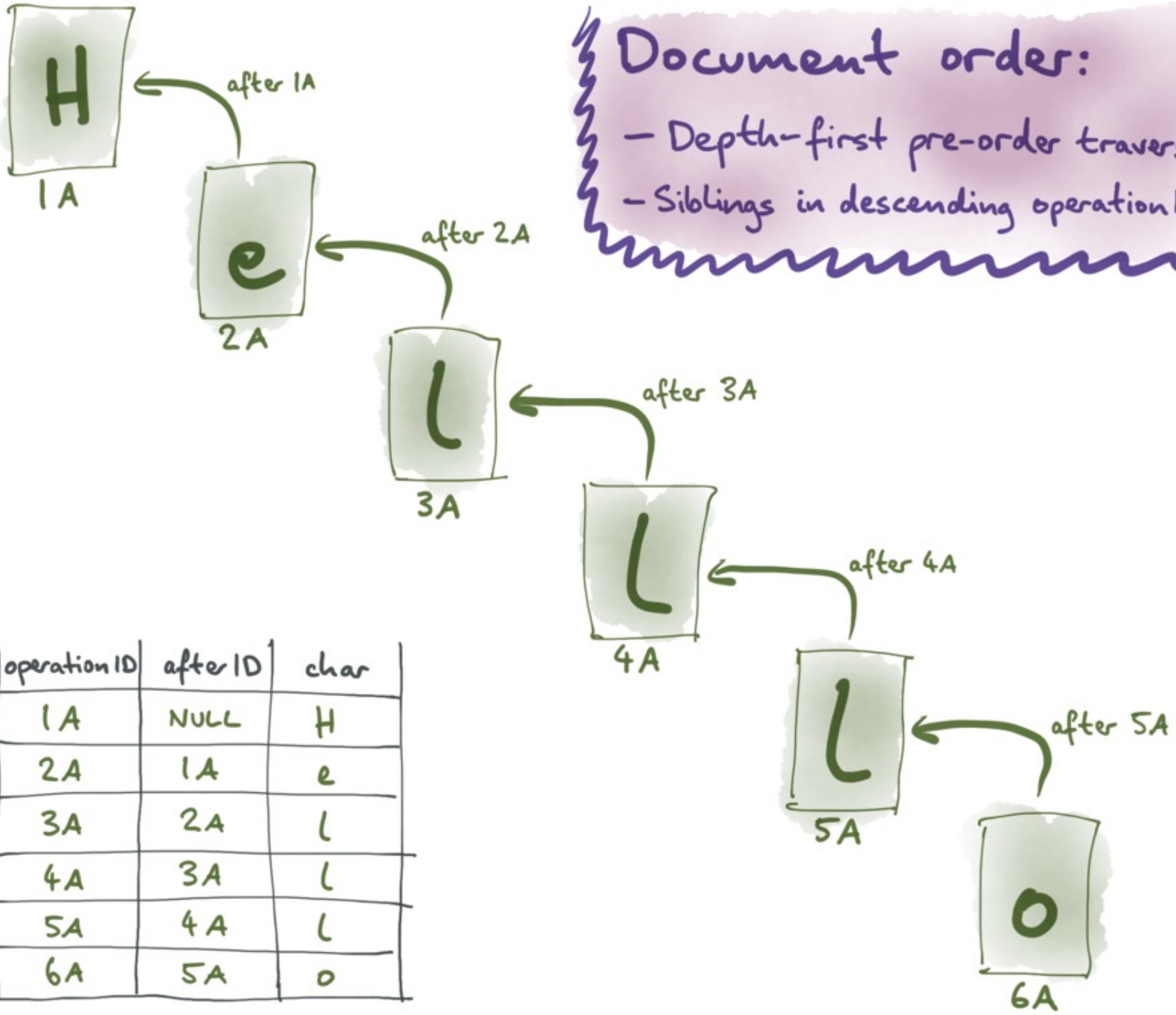
Benchmark data: keystroke-by-keystroke editing trace of a text file (LaTeX source of a research paper) containing 182,315 single-character insertions and 77,463 single-character deletions, timestamped with 1-second granularity.

As individual changes: 33.7 MB (130 bytes/operation)
As compressed document with full edit history: 184 kB (0.7 bytes/operation)





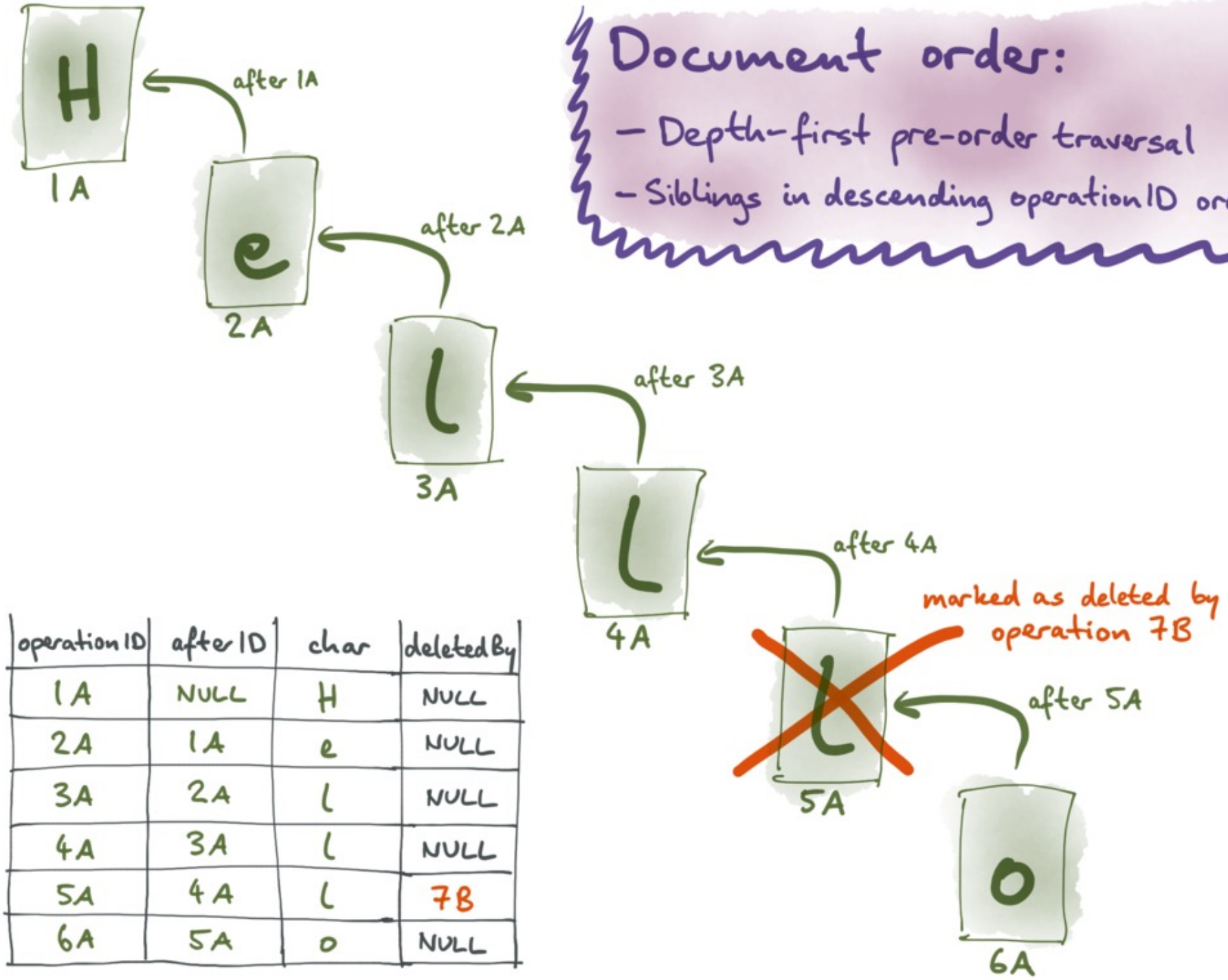
operation ID	after ID	char
1A	NULL	H
2A	1A	e
3A	2A	l
4A	3A	l
5A	4A	l
6A	5A	o



Document order:

- Depth-first pre-order traversal
- Siblings in descending operationID order

operationID	afterID	char
1A	NULL	H
2A	1A	e
3A	2A	l
4A	3A	l
5A	4A	l
6A	5A	o



operationID	afterID	char	deletedBy
1A	NULL	H	NULL
2A	1A	e	NULL
3A	2A	l	NULL
4A	3A	l	NULL
5A	4A	l	7B
6A	5A	o	NULL

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

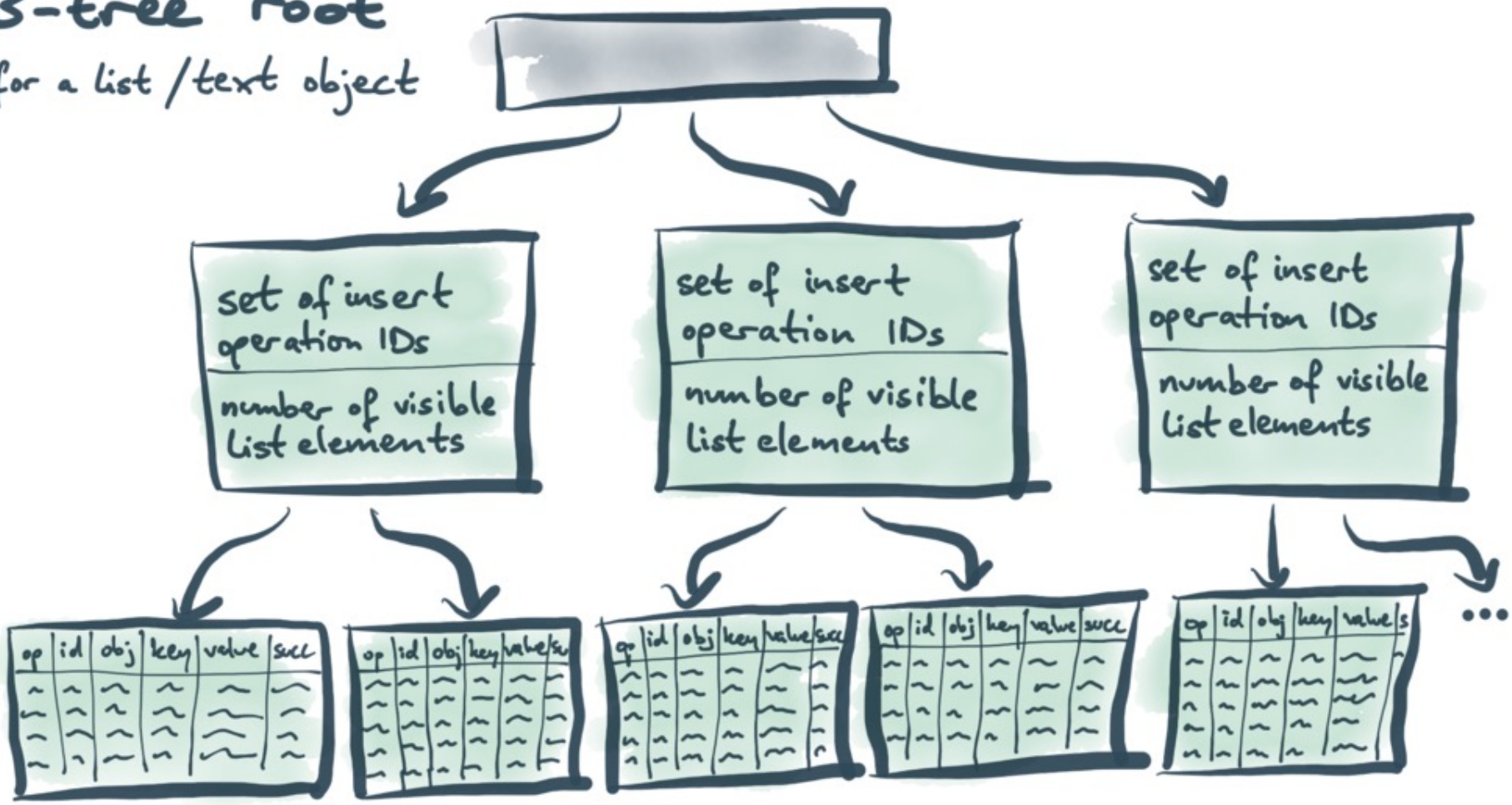
COLUMNAR ENCODING (simplified)

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 → "Helllo" (6 bytes)
(use length column to separate again)

B-tree root

for a list / text object



Local insert: find insertion position by index, counting only visible elements, to translate index into opID of last insert at that index

Remote insert: find subtree containing position ID, add op to B-tree, compute index of inserted element based on number of preceding visible elements

Identify document version by version vector

e.g. $V = \{a:3, b:4\}$

Visibility rule: operation with $ID = (ctr_{ID}, node_{ID})$ and $overwrittenBy = \{(ctr_1, node_1), (ctr_2, node_2), \ldots\}$ is visible at document version V iff $ctr_{ID} \leq V[node_{ID}]$ and

$\nexists (ctr_i, node_i) \in overwrittenBy.\ ctr_i \leq V[node_i]$.

Like MVCC in databases with snapshot isolation!

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"	—	—

Identify document version by version vector

e.g. $V = \{a:3, b:4\}$

Visibility rule: operation with $ID = (ctr_{ID}, node_{ID})$ and $overwrittenBy = \{(ctr_1, node_1), (ctr_2, node_2), \dots\}$ is visible at document version V iff $ctr_{ID} \leq V[node_{ID}]$ and

$\nexists (ctr_i, node_i) \in overwrittenBy.\ ctr_i \leq V[node_i]$.

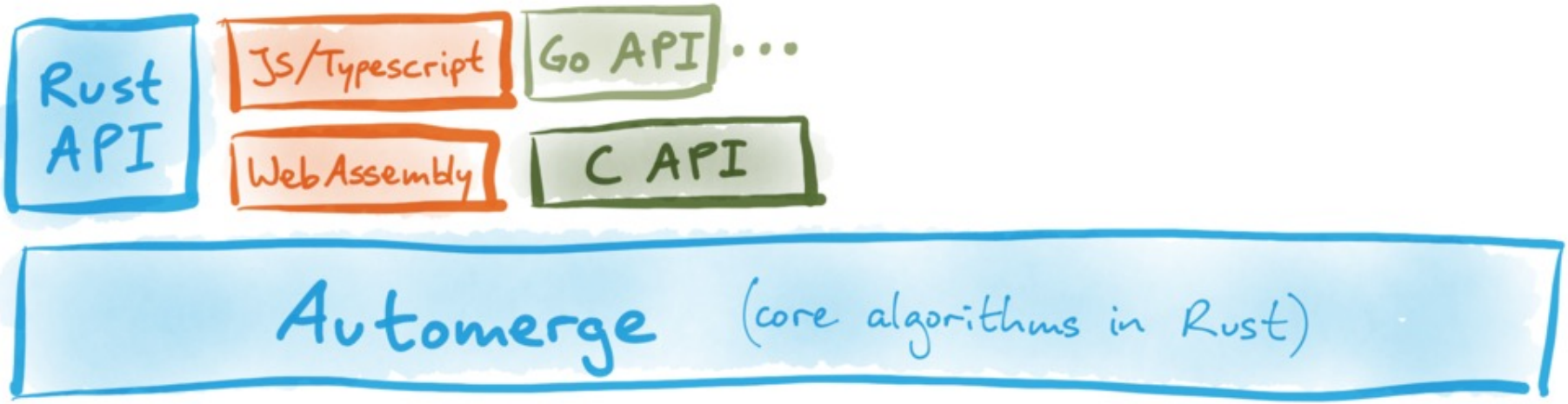
Like MVCC in databases with snapshot isolation!

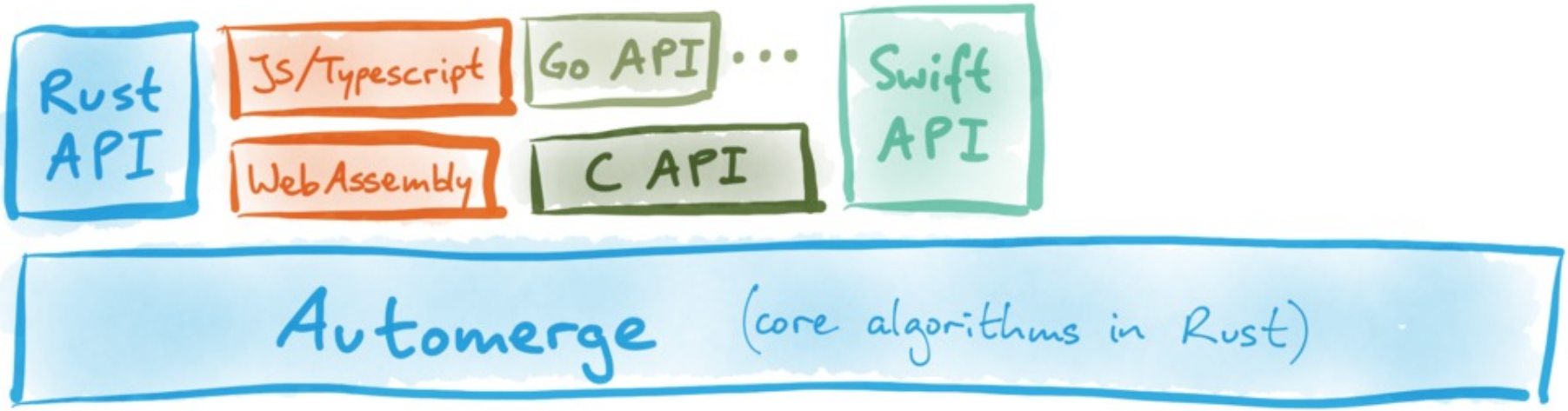
For a given key / list element:

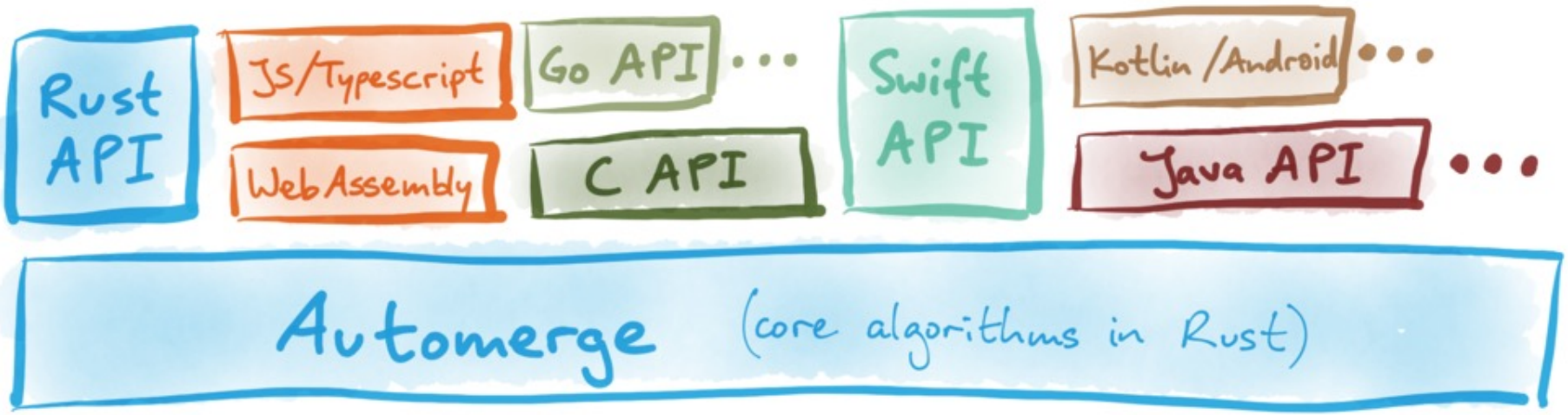
- no ops visible ⇒ deleted
- one op visible ⇒ current value
- multiple ops visible ⇒ conflict (concurrent assignment)

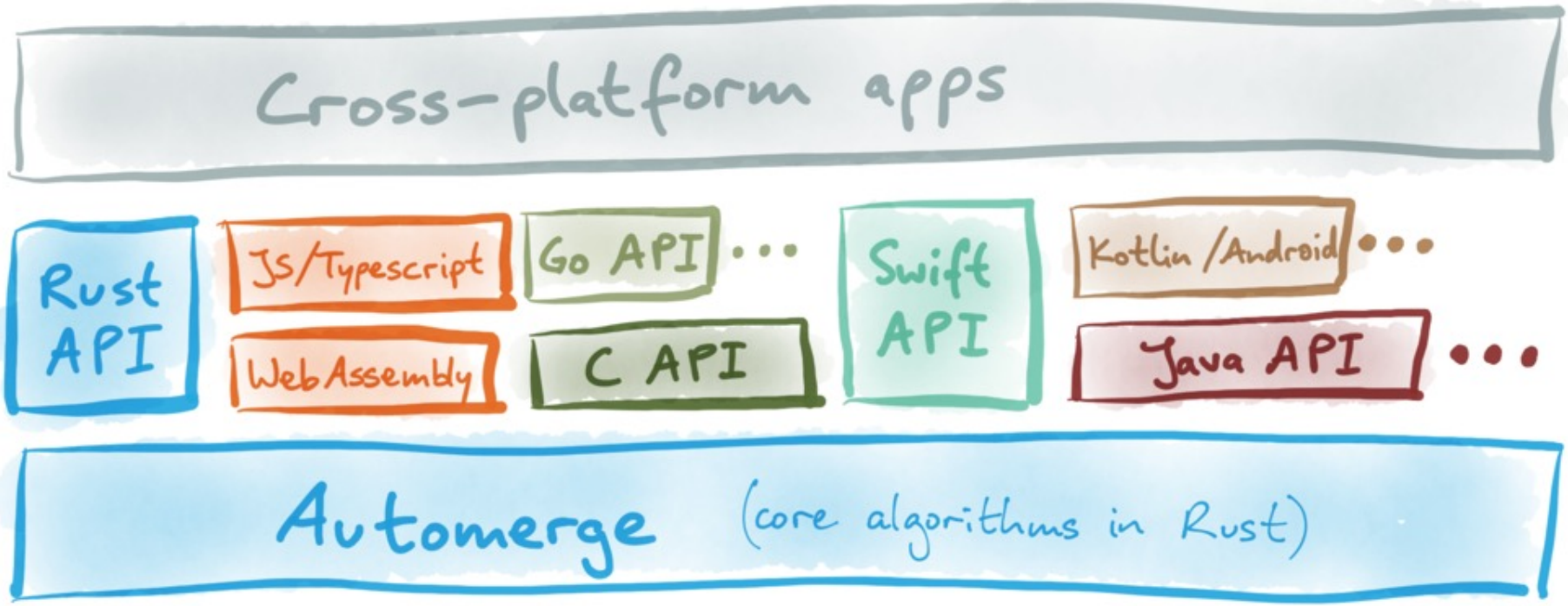

Rust
API
Automerge (core algorithms in Rust)

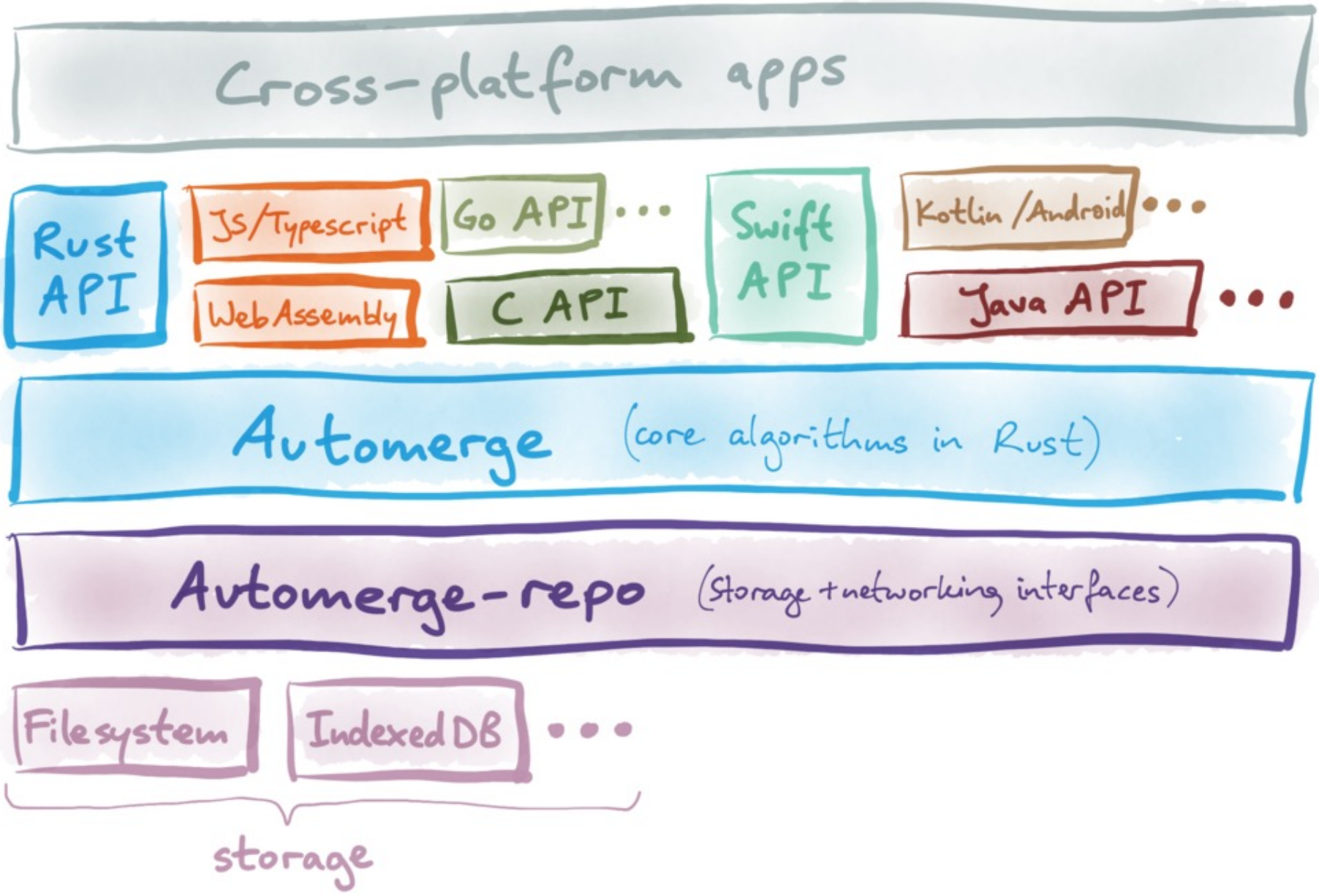

Rust API
JS/Typescript
WebAssembly
Automerge (core algorithms in Rust)

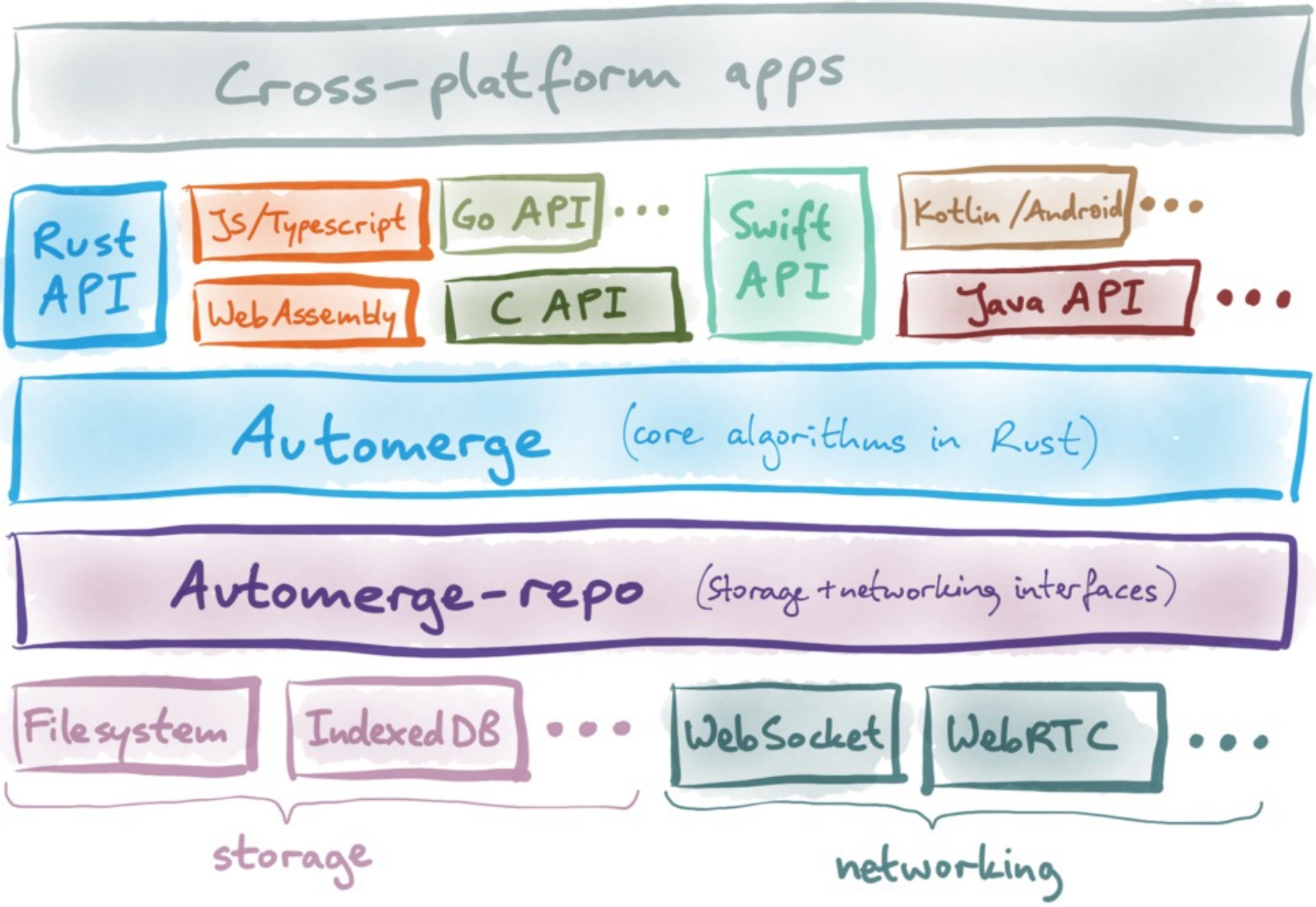

Rust API
JS/Typescript
Go API
...
WebAssembly
C API
Automerge
(core algorithms in Rust)


Rust API
JS/Typescript
Web Assembly
Go API
...
C API
Swift API
Automerge
(core algorithms in Rust)


Rust API
JS/Typescript
Web Assembly
Go API
C API
Swift API
Kotlin /Android
Java API
Automerge
(core algorithms in Rust)


Cross-platform apps
Rust API
JS/Typescript
WebAssembly
Go API
C API
Swift API
Kotlin/Android
Java API
Automerge
(core algorithms in Rust)


Cross-platform apps
Rust API
JS/Typescript
Web Assembly
Go API
C API
Swift API
Kotlin /Android
Java API
Automerge
(core algorithms in Rust)
Automerge-repo
(Storage + networking interfaces)
Filesystem
IndexedDB
storage


Cross-platform apps
Rust API
JS/Typescript
Web Assembly
Go API
C API
Swift API
Kotlin /Android
Java API
Automerge
(core algorithms in Rust)
Automerge-repo
(Storage + networking interfaces)
Filesystem
IndexedDB
WebSocket
WebRTC
storage
networking

A brief history of Automerge

Version 0.1 – 0.14 (2017 – 2020)

Research prototype in JavaScript

A brief history of Automerge

Version 0.1 – 0.14 (2017–2020)

Research prototype in JavaScript

Version 1.0 (2021)

Compressed data format, sync protocol

A brief history of Automerge

Version 0.1 – 0.14 (2017–2020)

Research prototype in JavaScript

Version 1.0 (2021)

Compressed data format, sync protocol

Version 2.0 (2022)

Moved to Rust, performance, production ready, commercial support

A brief history of Automerge

Version 0.1 – 0.14 (2017 – 2020)

Research prototype in JavaScript

Version 1.0 (2021)

Compressed data format, sync protocol

Version 2.0 (2022)

Moved to Rust, performance, production ready, commercial support

Version 2.1 (2023)

Rich text, automerge-repo, performance

CLOUD SOFTWARE	LOCAL-FIRST SOFTWARE
Server returns the current state, other copies are just caches	Every device has state → Multiple versions explicit → Propagating & merging versions

CLOUD SOFTWARE

Server returns the current state, other copies are just caches

Version management + history is an afterthought

LOCAL-FIRST SOFTWARE

Every device has state

→ Multiple versions explicit

→ Propagating & merging versions

OPPORTUNITY:

Build version control into apps at a fundamental level

CLOUD SOFTWARE

Server returns the current state, other copies are just caches

Version management + history is an afterthought

LOCAL-FIRST SOFTWARE

Every device has state

→ Multiple versions explicit

→ Propagating & merging versions

OPPORTUNITY:

Build version control into apps at a fundamental level

Bring Git-like branching/merging (async collaboration) to non-software developers

COLLABORATIVE TEXT EDITING • TODAY'S TOPICS

1. Operational transformation
2. Branching and the need for CRDTs
3. Automerge
4. Rich text — Peritext
5. Non-interleaving — Fugue

Peritext: A CRDT for Collaborative Rich Text Editing

GEOFFREY LITT, MIT CSAIL, USA
SARAH LIM, UC Berkeley, USA
MARTIN KLEPPMANN, University of Cambridge, United Kingdom
PETER VAN HARDENBERG, Ink & Switch, USA

Conflict-Free Replicated Data Types (CRDTs) support decentralized collaborative editing of shared data, enabling peer-to-peer sharing and flexible branching and merging workflows. While there is extensive work on CRDTs for plain text, much less is known about CRDTs for rich text with formatting. No algorithms have been published, and existing open-source implementations do not always preserve user intent.

In this paper, we describe a model of intent preservation in rich text editing, developed through a series of concurrent editing scenarios. We then describe Peritext, a CRDT algorithm for rich text that satisfies the criteria of our model. The key idea is to store formatting spans alongside the plaintext character sequence, linked to a stable identifier for the first and last character of each span, and then to derive the final formatted text from these spans in a deterministic way that ensures concurrent operations commute.

We have prototyped our algorithm in TypeScript, validated it using randomized property-based testing, and integrated it with an editor UI. We also prove that our algorithm ensures convergence, and demonstrate its causality preservation and intention preservation properties.

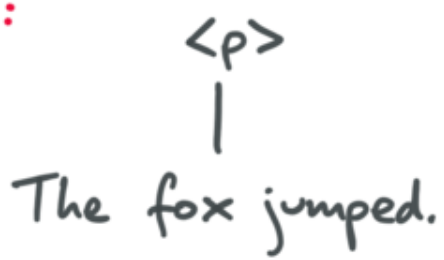
CCS Concepts: • **Human-centered computing** → **Asynchronous editors**; • **Information systems** → Version management.

Additional Key Words and Phrases: collaborative editing, asynchronous collaboration, rich text, Conflict-free Replicated Data Types

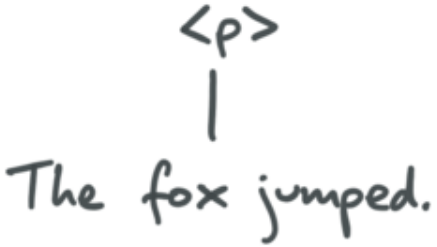
ACM Reference Format:
Geoffrey Litt, Sarah Lim, Martin Kleppmann, and Peter van Hardenberg. 2022. Peritext: A CRDT for Collaborative Rich Text Editing. *Proc. ACM Hum.-Comput. Interact.* 6, CSCW2, Article 531 (November 2022), 35 pages. https://doi.org/10.1145/3555644

Rich text as a tree?

A:



B:

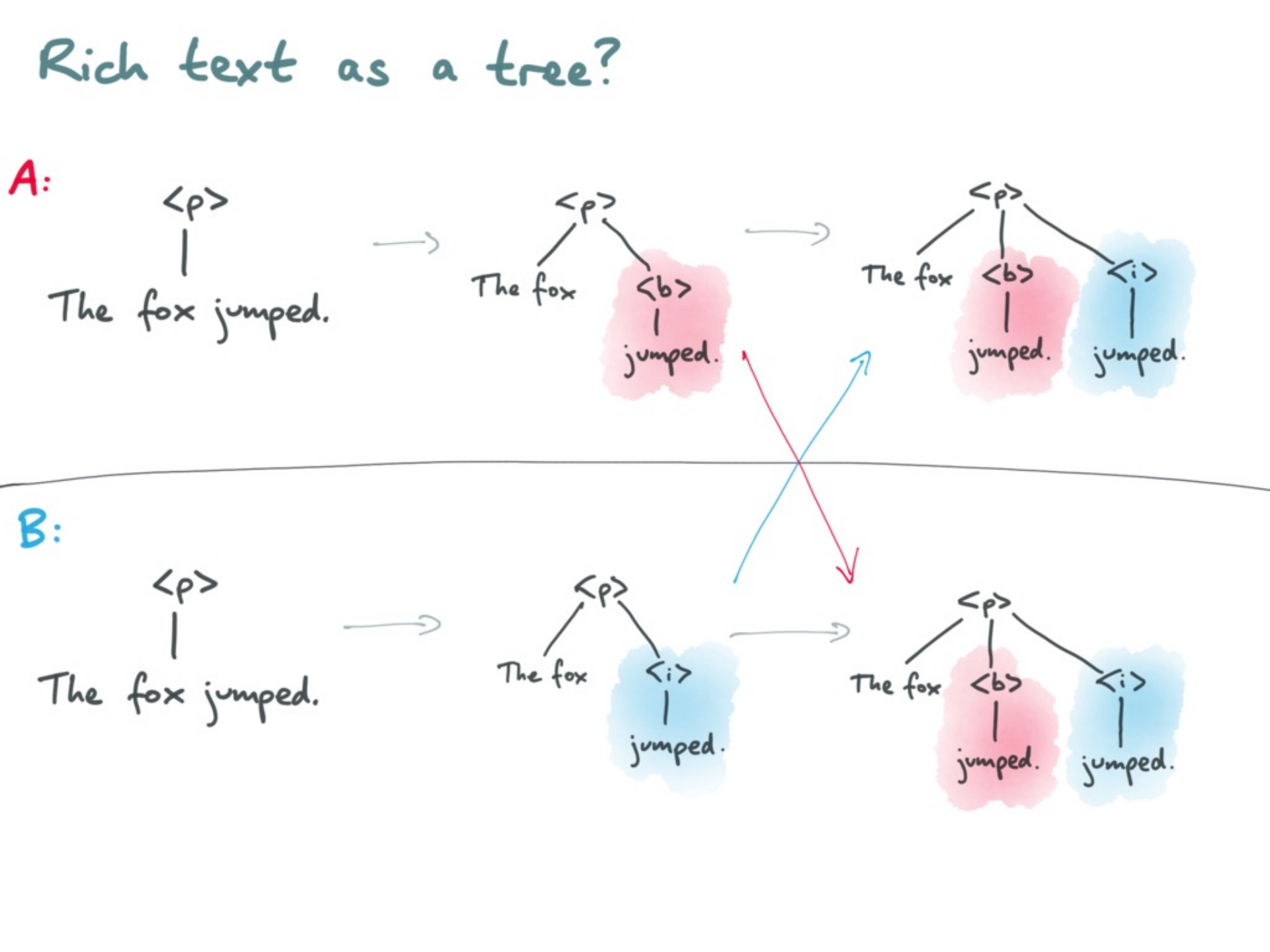


Rich text as a tree?

A:

B:

Rich text as a tree?



Rich text as markup?

A:

 The fox

jumped

over the dog.

B:

 The fox

jumped

over the dog.

Rich text as markup?

A:

 The fox jumped over the dog.

unbold everything →

The fox jumped over the dog.

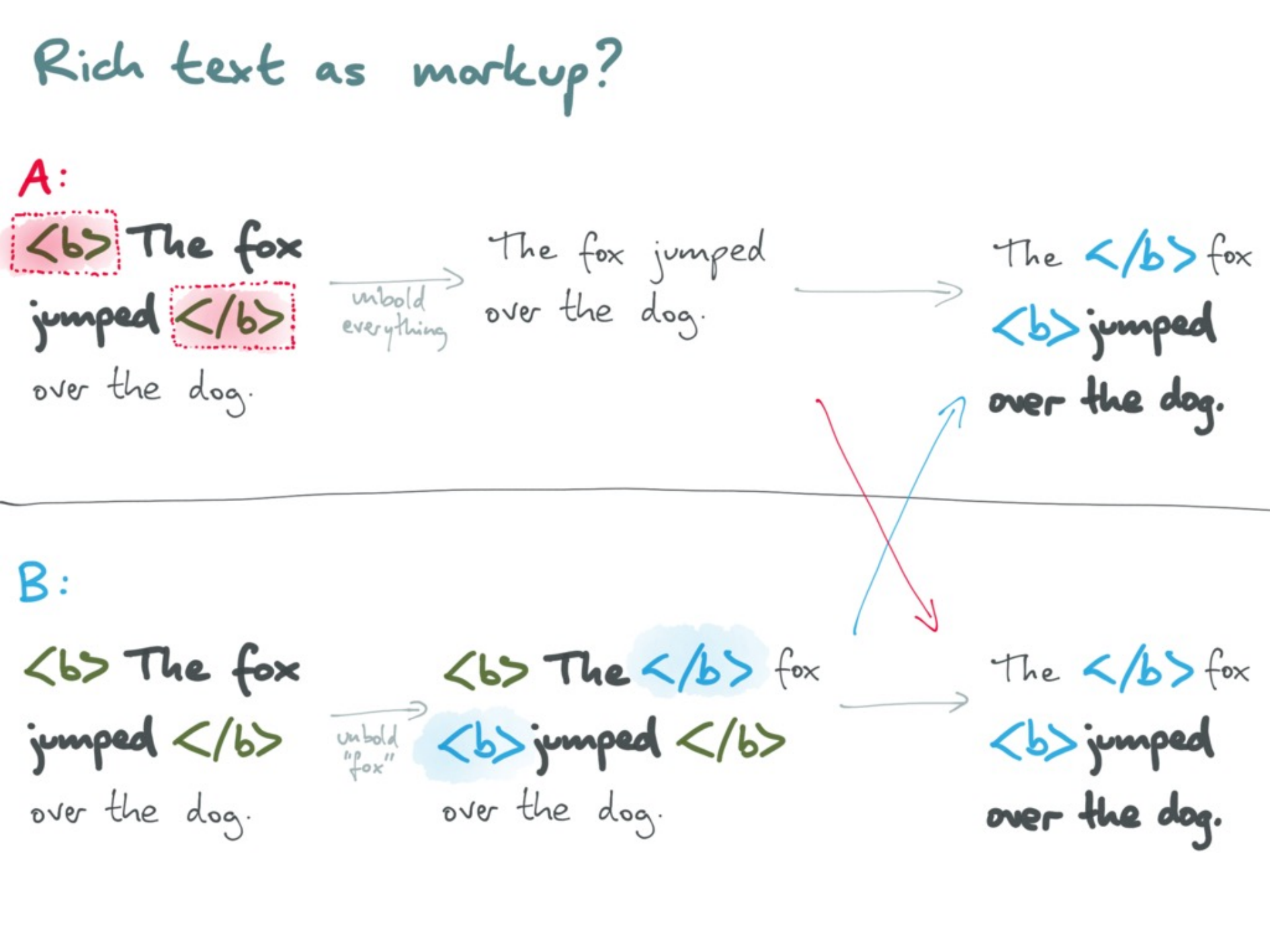
B:

 The fox jumped over the dog.

unbold "fox" →

 The fox jumped over the dog.

Rich text as markup?



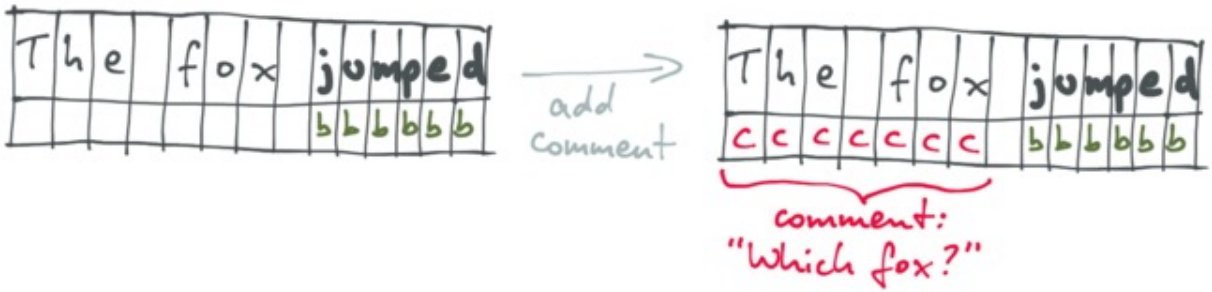
Rich text as per-character properties?

A:

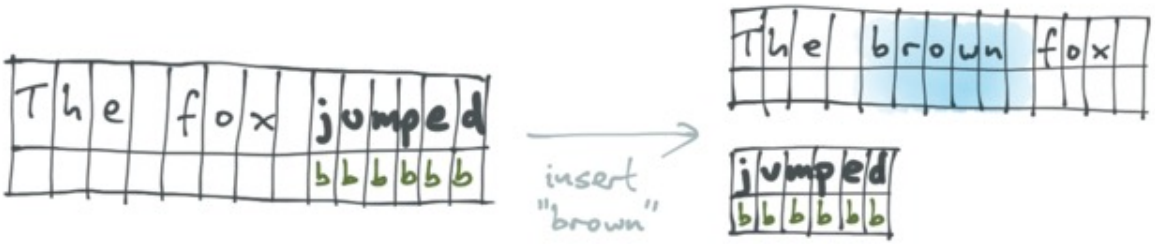
B:

Rich text as per-character properties?

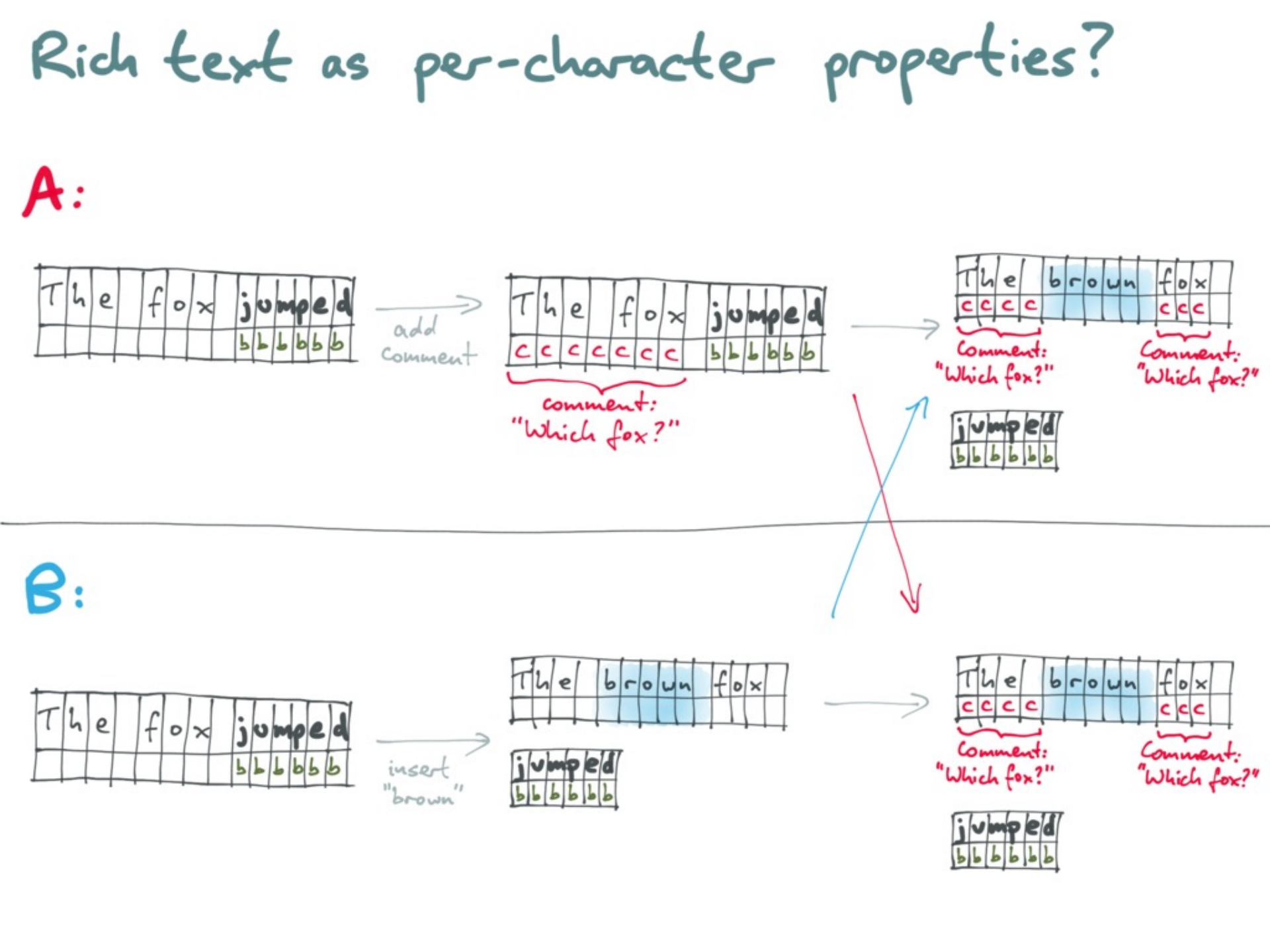
A:



B:



Rich text as per-character properties?



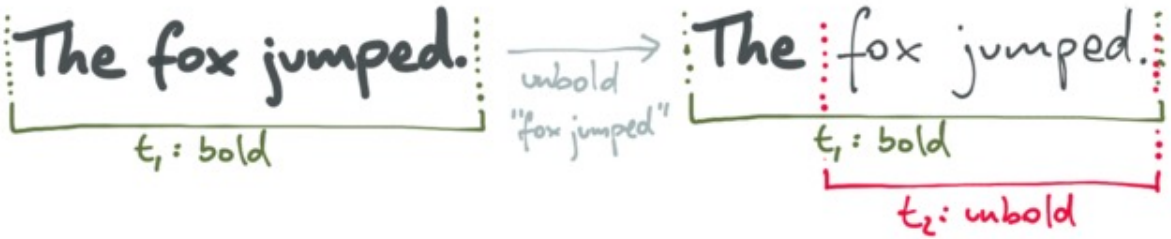
Rich text as text with annotation spans

A:

B:

Rich text as text with annotation spans

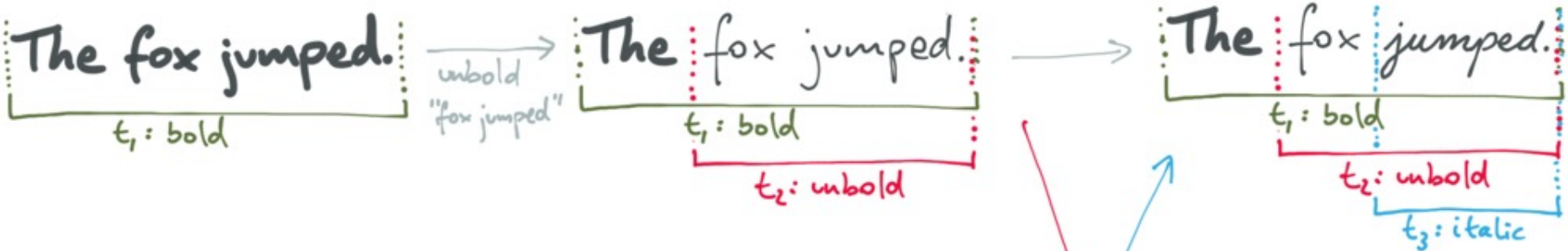
A:



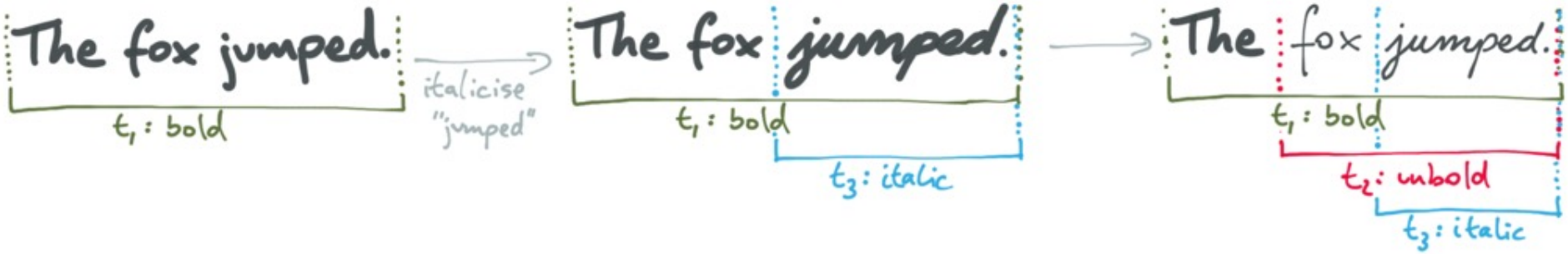
B:

Rich text as text with annotation spans

A:



B:



COLLABORATIVE TEXT EDITING • TODAY'S TOPICS

1. Operational transformation
2. Branching and the need for CRDTs
3. Automerge
4. Rich text — Peritext
5. Non-interleaving — Fugue

work in progress; previous version at https://arxiv.org/abs/2305.00583

The Art of the Fugue: Minimizing Interleaving in Collaborative Text Editing

Matthew Weidner and Martin Kleppmann

Abstract—Most existing algorithms for replicated lists, which are widely used in collaborative text editors, suffer from a problem: when two users concurrently insert text at the same position in the document, the merged outcome may interleave the inserted text passages, resulting in corrupted and potentially unreadable text. The problem has gone unnoticed for decades, and it affects both CRDTs and Operational Transformation. This paper presents Fugue, an algorithm that guarantees maximal non-interleaving, our new correctness property for replicated lists. We also implement Fugue and demonstrate that it offers performance comparable to state-of-the-art CRDT libraries for text editing.

Index Terms—Distributed data structures, replica consistency, collaborative text editing, Conflict-free Replicated Data Types (CRDTs), operational transformation.

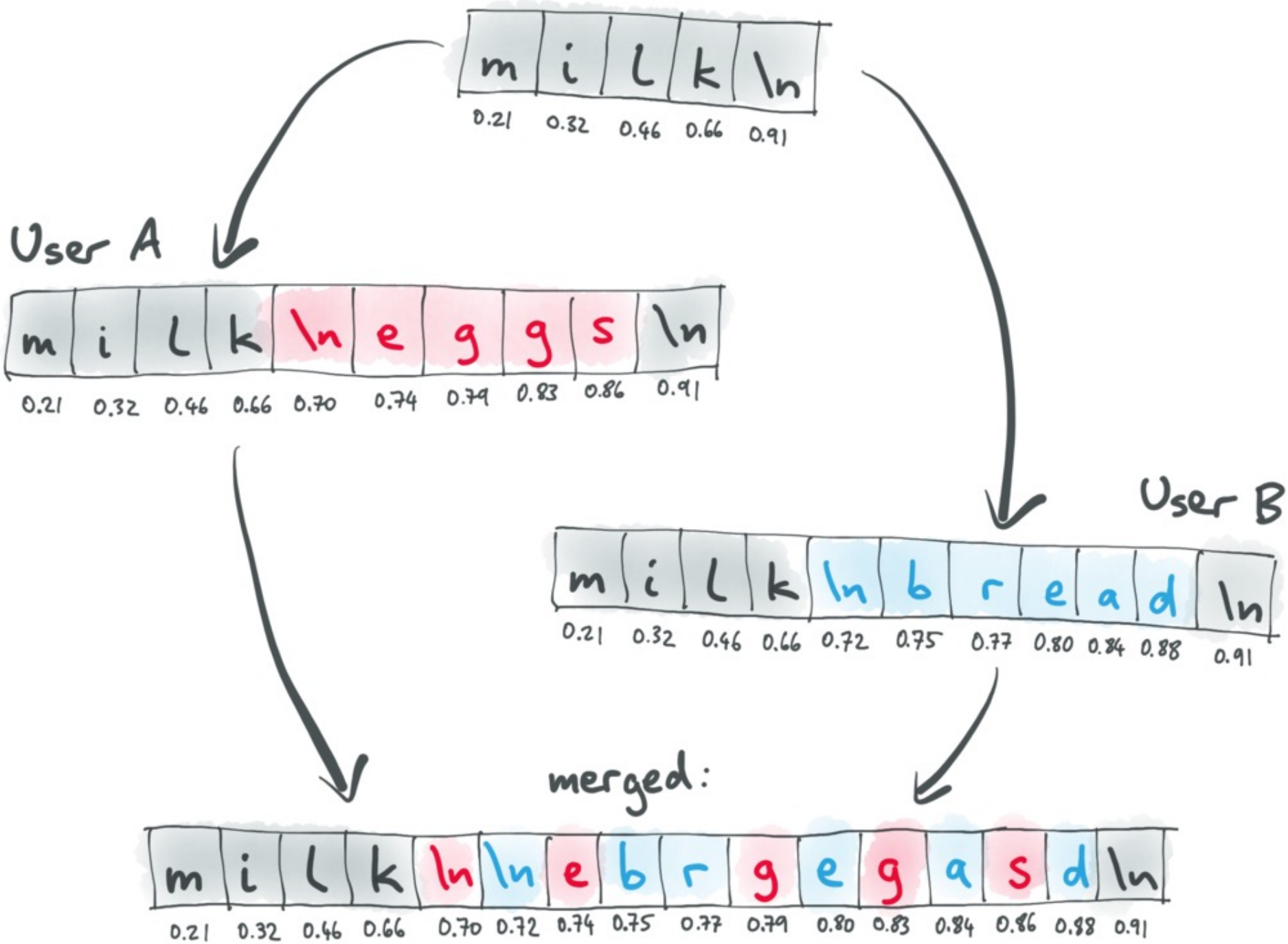
1 Introduction

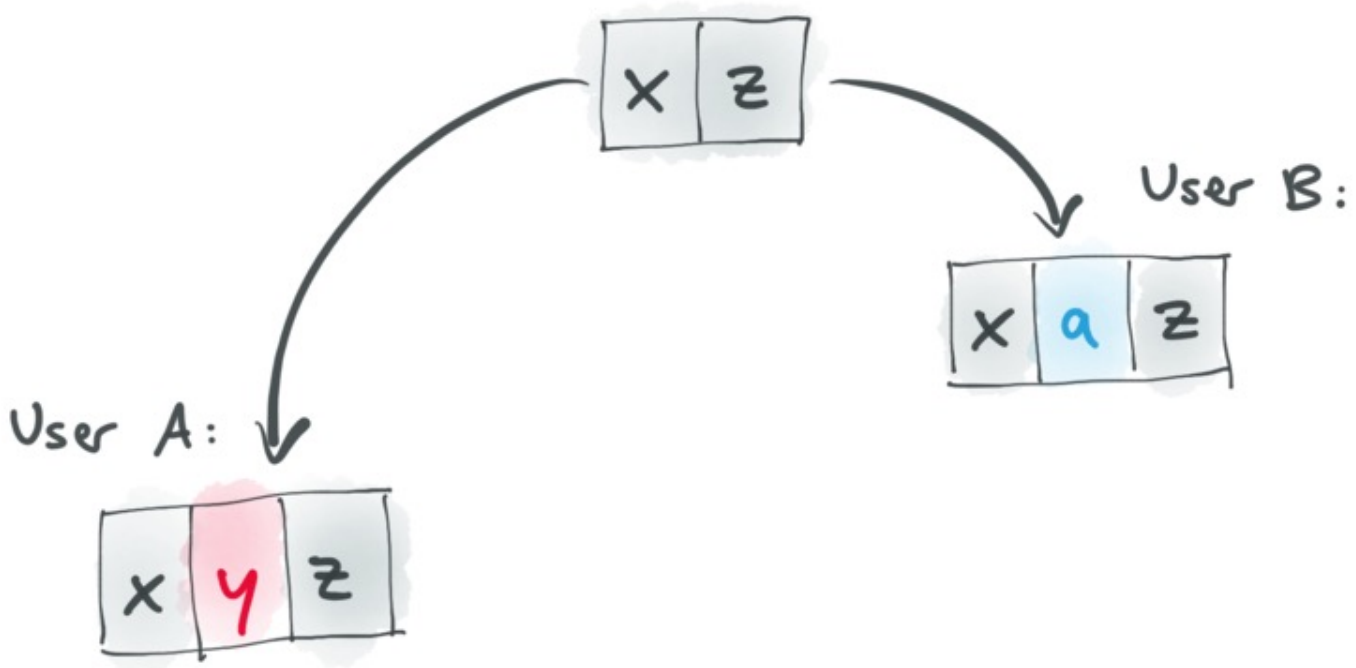
Collaborative text editors such as Google Docs allow several users to concurrently modify a document, while ensuring that all users' copies of the document converge to the same state. This type of software is implemented as a replicated list of characters; a replicated list object is a distributed data structure whose semantics can be formally specified [1].

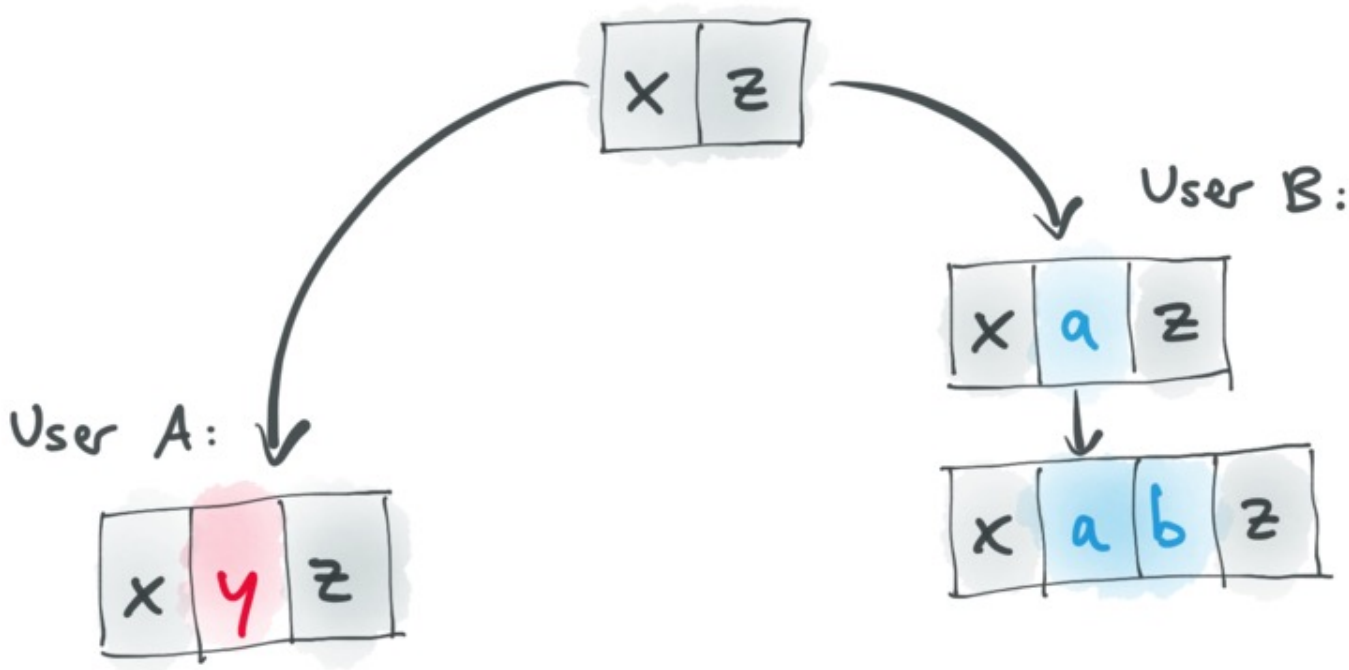
Even though algorithms for replicated lists and collaborative text editing have been studied for over three decades [2], [3], [4], a formal specification of the required behavior of a replicated list only appeared as recently as 2016 [1]. We argue that this specification is incomplete. In Section 2 we show that there is an additional correctness property that is important in practice, but which has been overlooked by almost all prior research on this topic: *non-interleaving*. Informally stated, this property requires that when sections of text are composed independently from

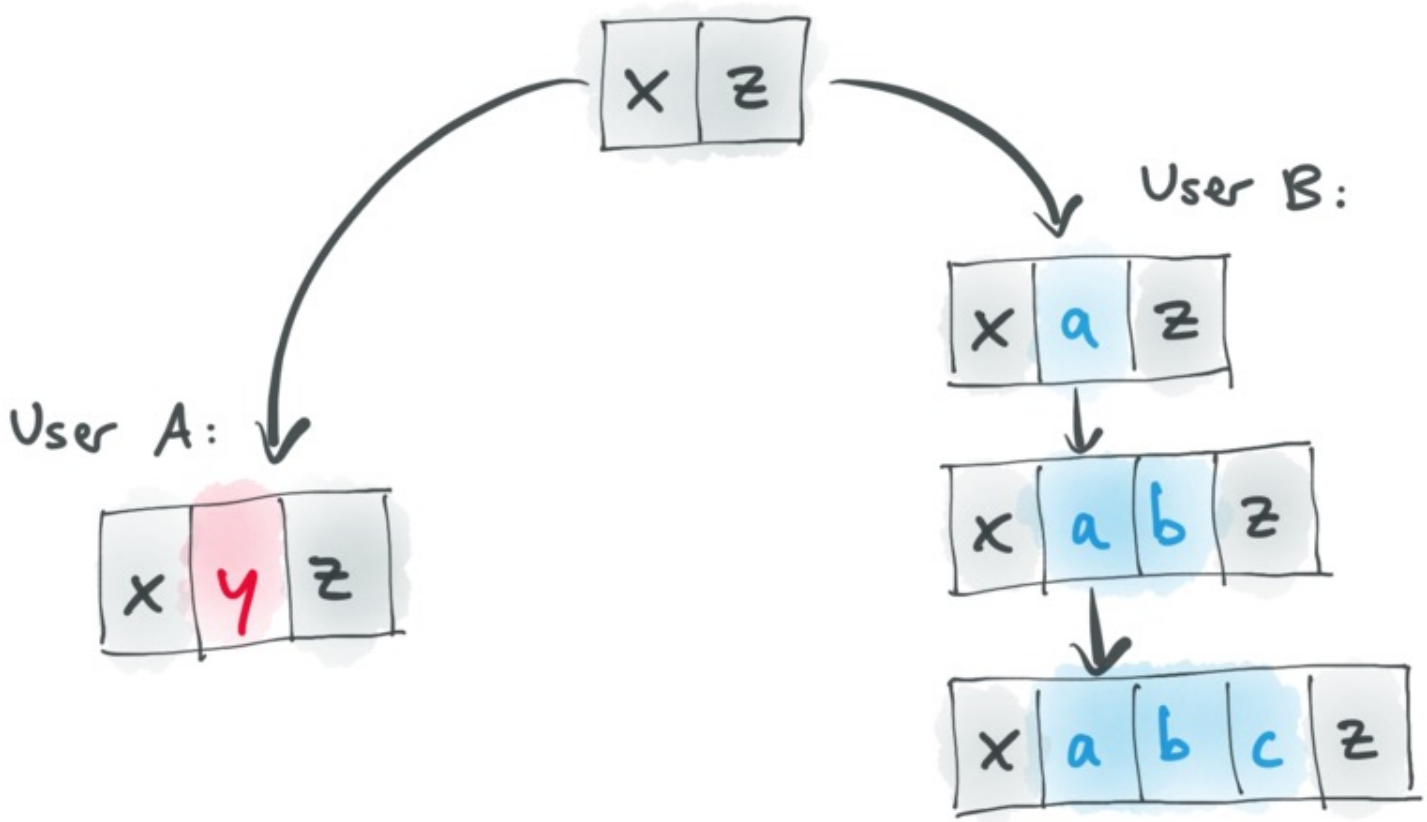
proposed algorithm in that paper does not converge (Section 2.4).

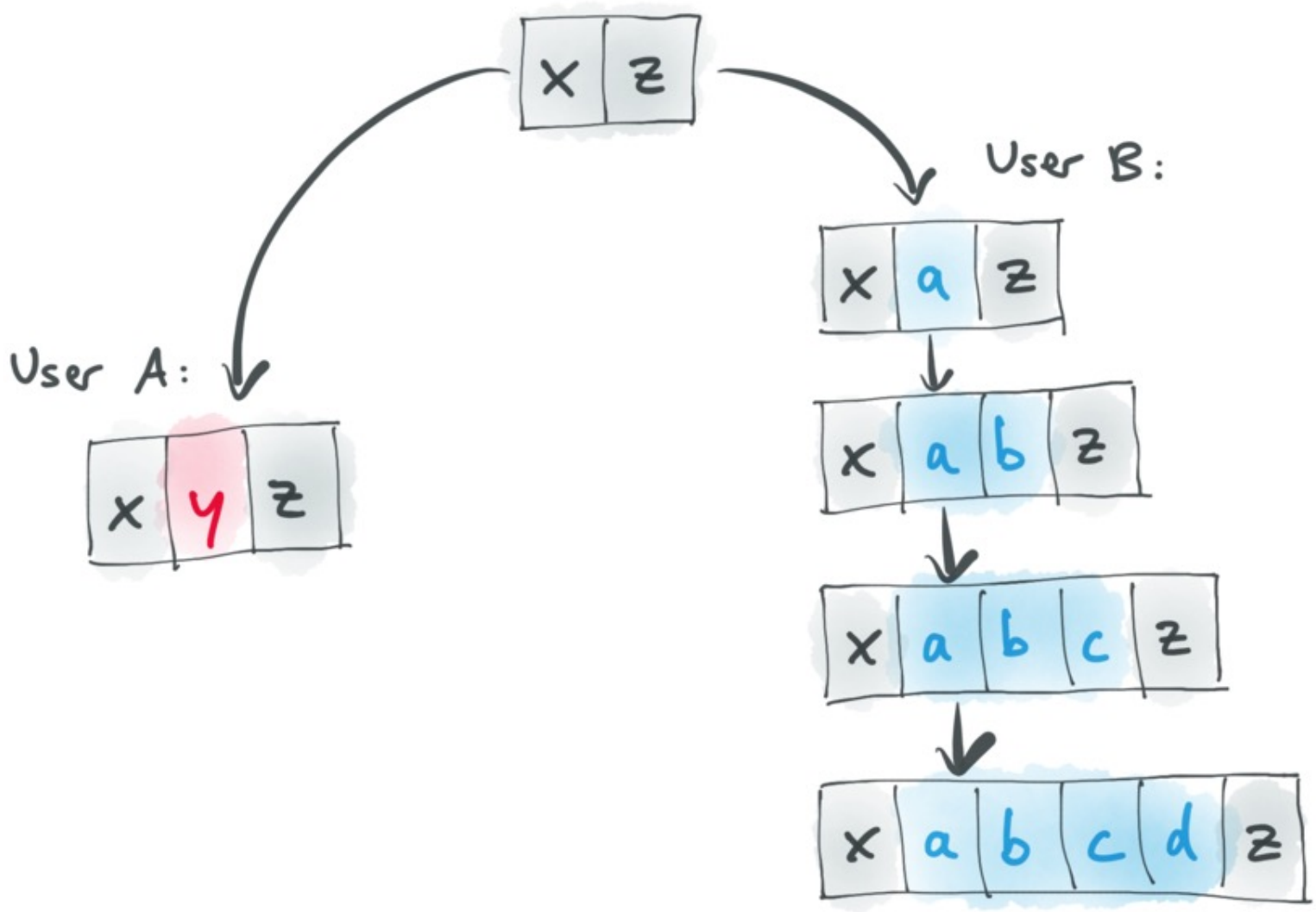
- We extend the formal specification of replicated lists by Attiya et al. [1] with a new property, which we call *maximal non-interleaving* (Section 5.1). This definition is subtle: we show that an alternative, simpler definition is also impossible to satisfy (Section 5.1).
- We introduce *Fugue*, a CRDT algorithm for replicated lists that (to our knowledge) is the first algorithm proven to satisfy maximal non-interleaving (Section 3).
- We provide an optimized open source implementation of Fugue, and show that it achieves memory, network, and CPU performance comparable to the state-of-the-art Yjs library on a realistic text-editing trace (Section 4).

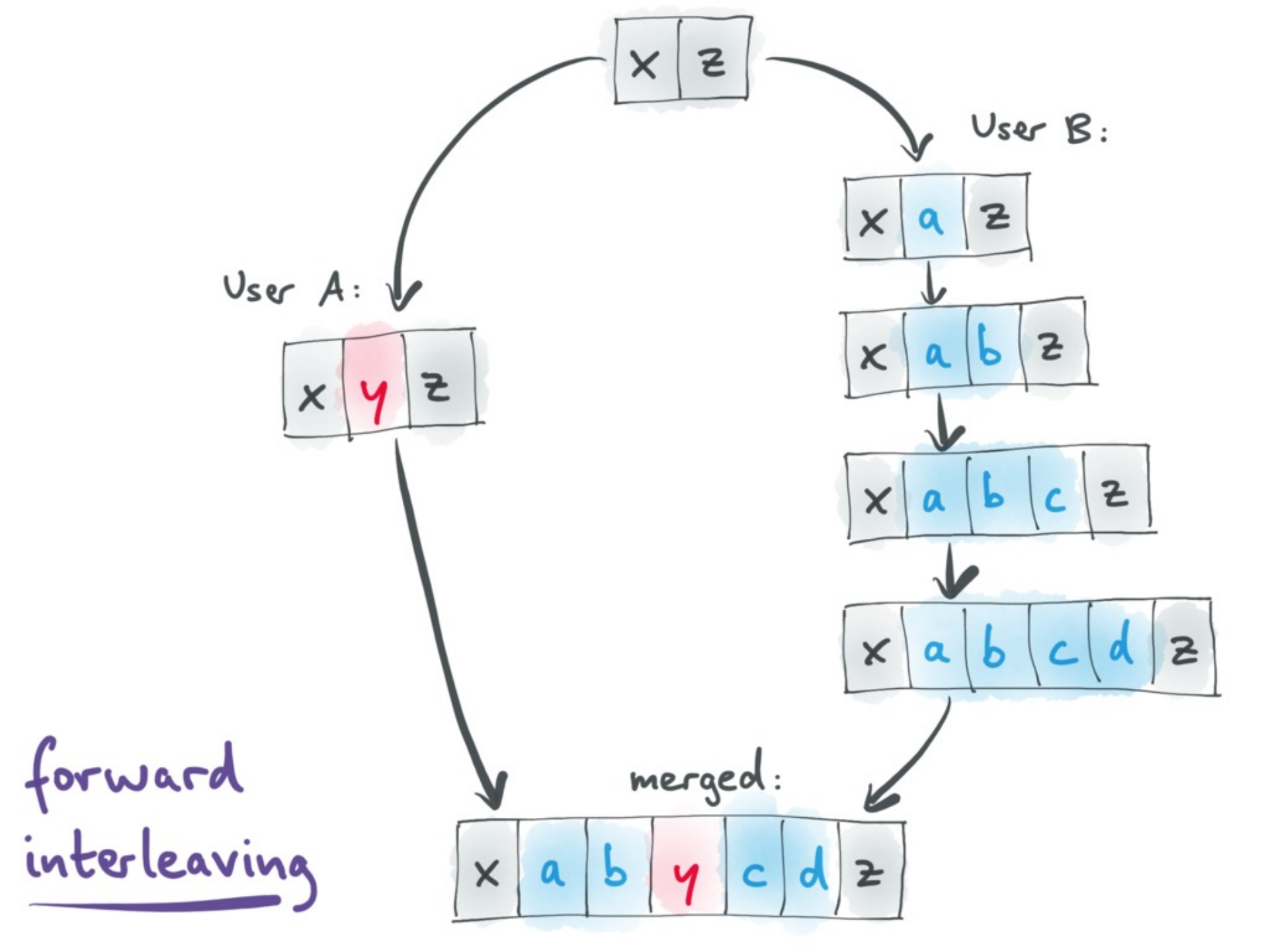

m i l k \n
0.21 0.32 0.46 0.66 0.91
User A
m i l k \n e g g s \n
0.21 0.32 0.46 0.66 0.70 0.74 0.79 0.83 0.86 0.91
User B
m i l k \n b r e a d \n
0.21 0.32 0.46 0.66 0.72 0.75 0.77 0.80 0.84 0.88 0.91
merged:
m i l k \n \n e b r g e g a s d \n
0.21 0.32 0.46 0.66 0.70 0.72 0.74 0.75 0.77 0.79 0.80 0.83 0.84 0.86 0.88 0.91

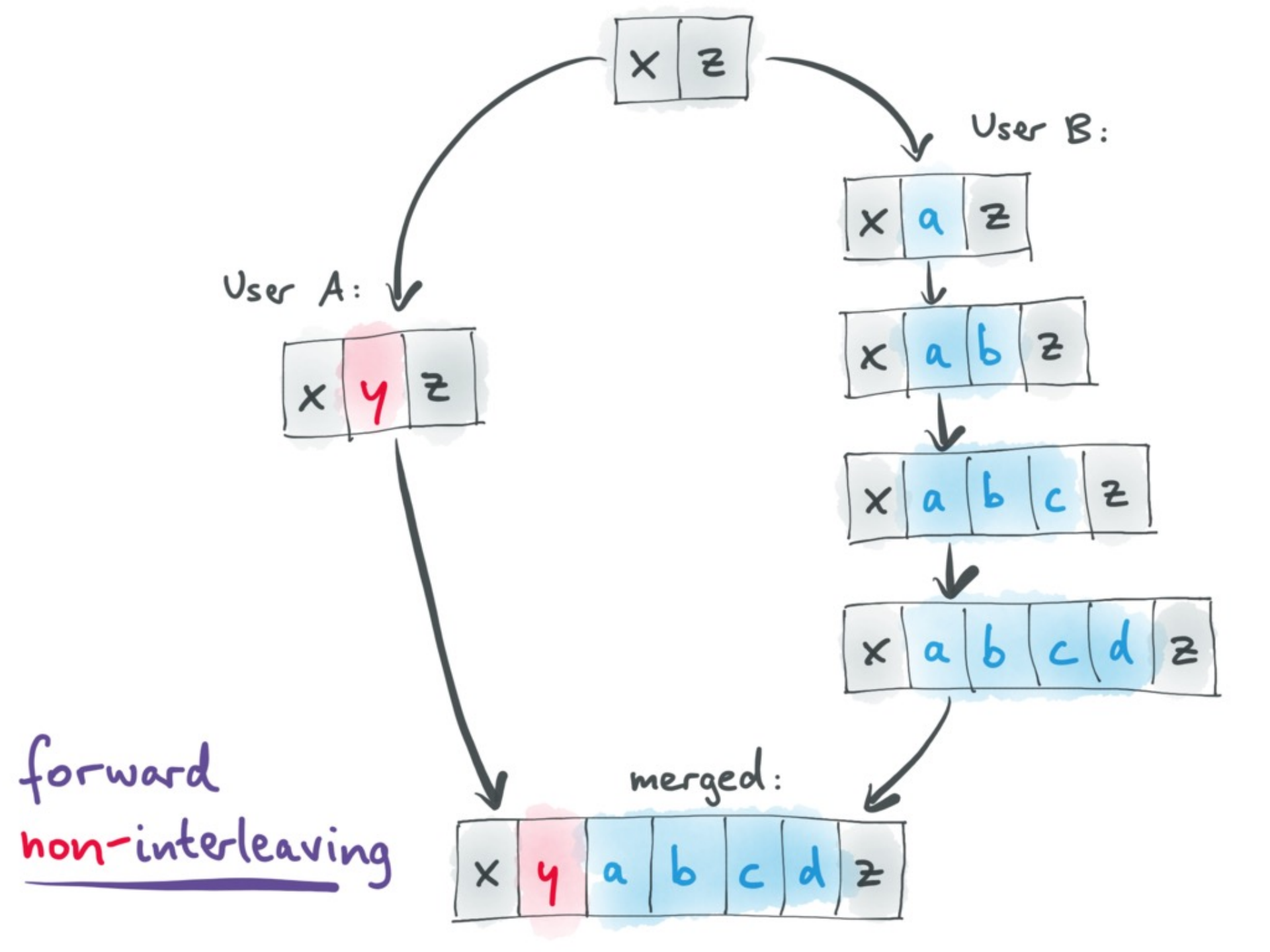

x z
User B:
x a z
User A:
x y z

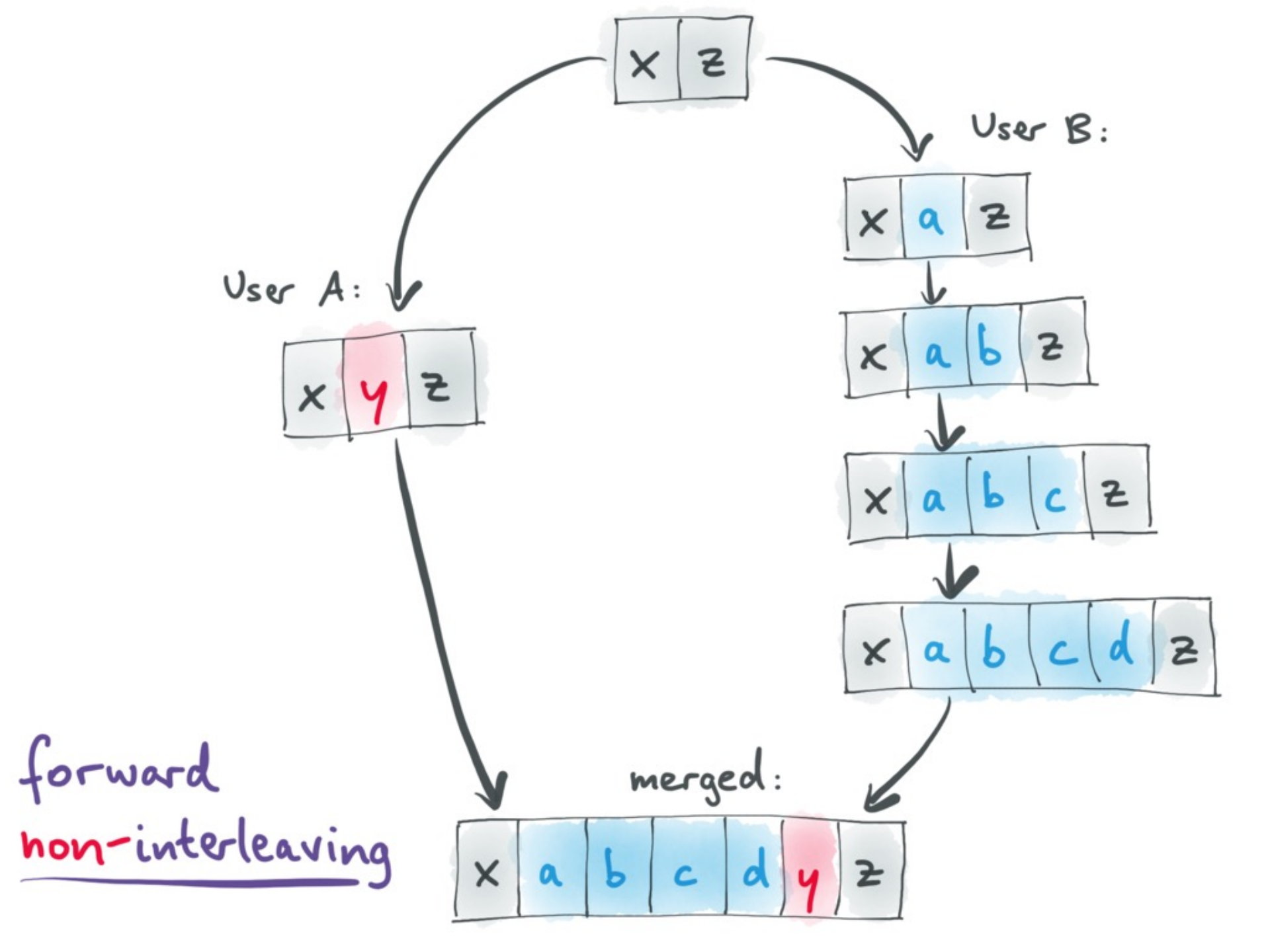

x z
User B:
x a z
x a b z
User A:
x y z


x z
User A:
x y z
User B:
x a z
x a b z
x a b c z

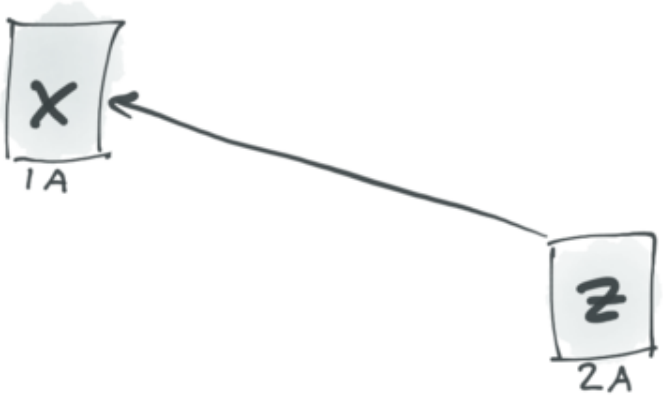

x z
User A:
x y z
User B:
x a z
x a b z
x a b c z
x a b c d z


x z
User B:
x a z
User A:
x y z
x a b z
x a b c z
x a b c d z
merged:
x a b y c d z
forward
interleaving

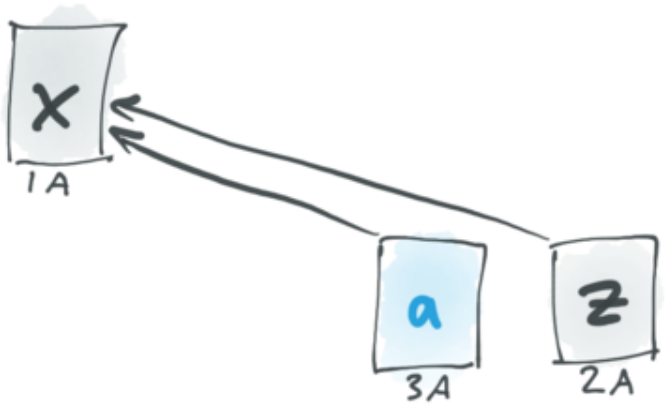

x z
User B:
x a z
User A:
x y z
x a b z
x a b c z
x a b c d z
merged:
x y a b c d z
forward
non-interleaving


x z
User B:
x a z
User A:
x y z
x a b z
x a b c z
x a b c d z
merged:
x a b c d y z
forward
non-interleaving

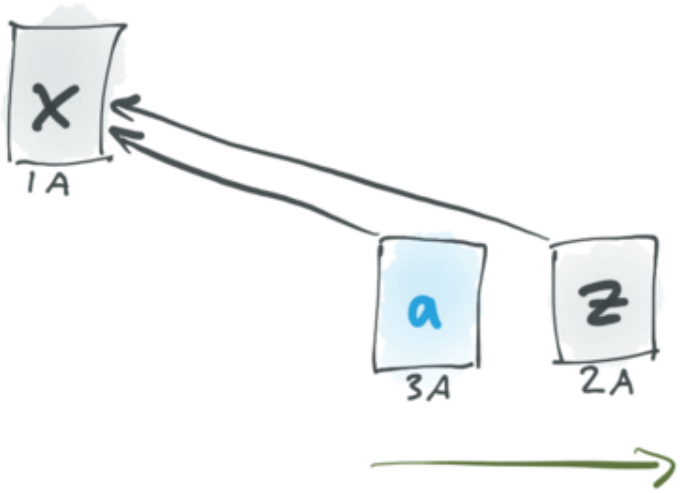
RGA is forward non-interleaving



RGA is forward non-interleaving

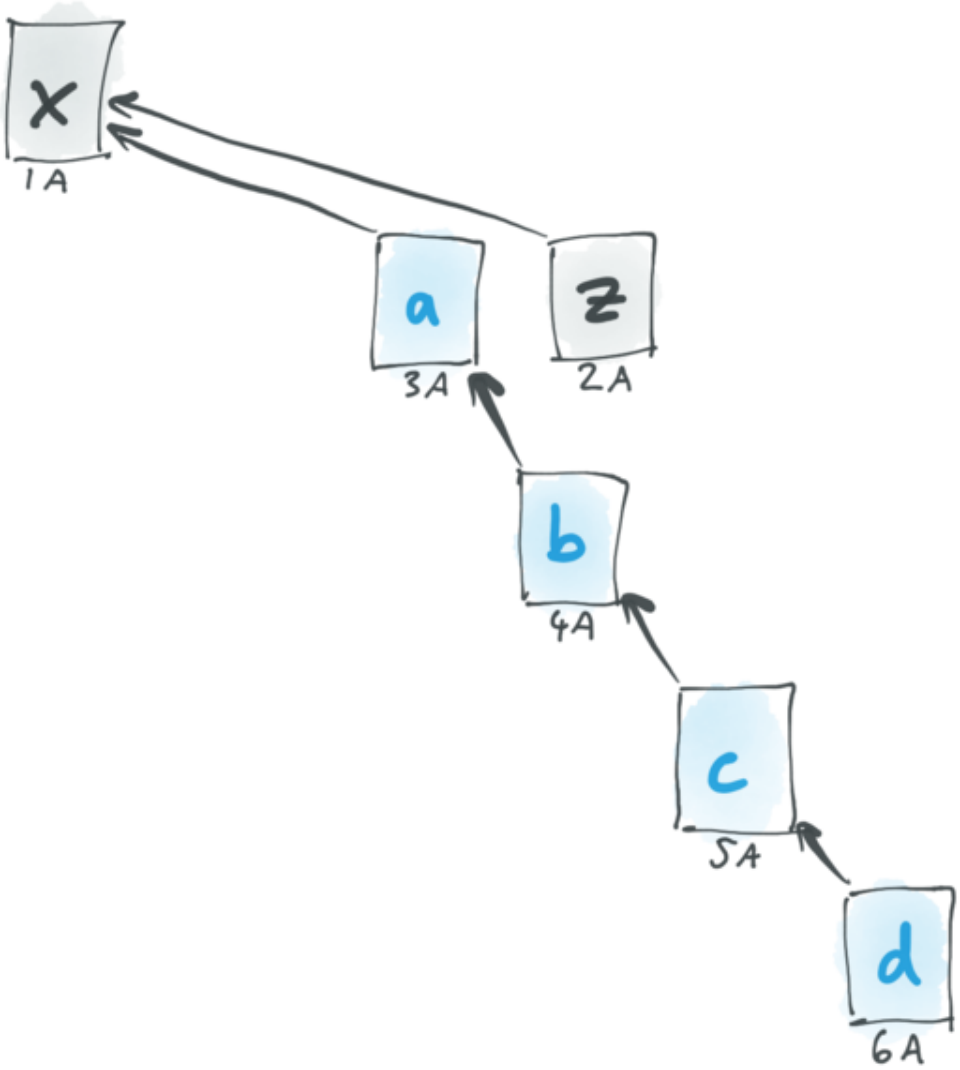


RGA is forward non-interleaving

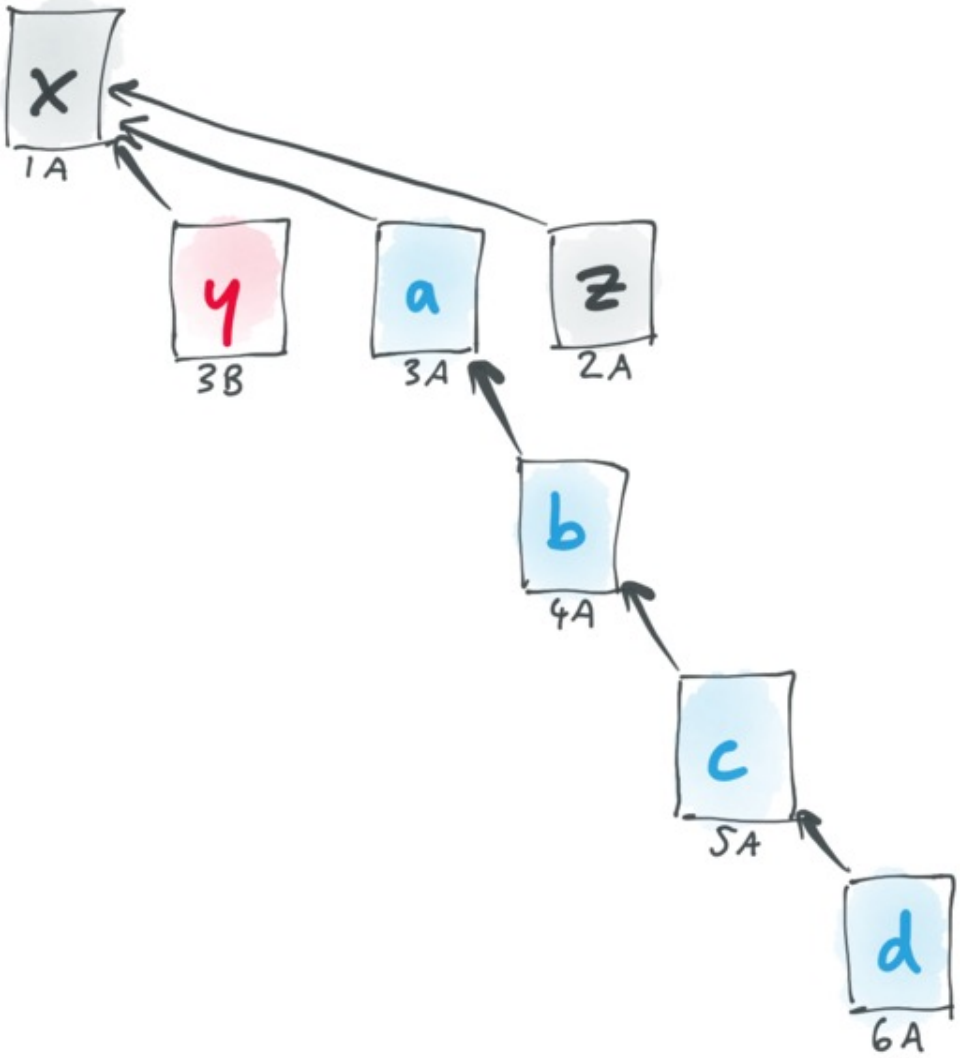


Visit siblings in descending timestamp order

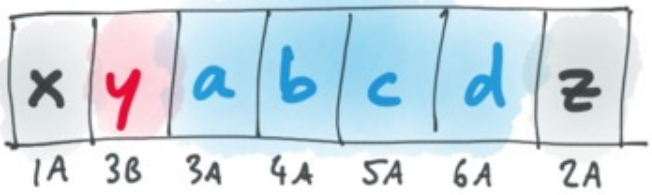
RGA is forward non-interleaving



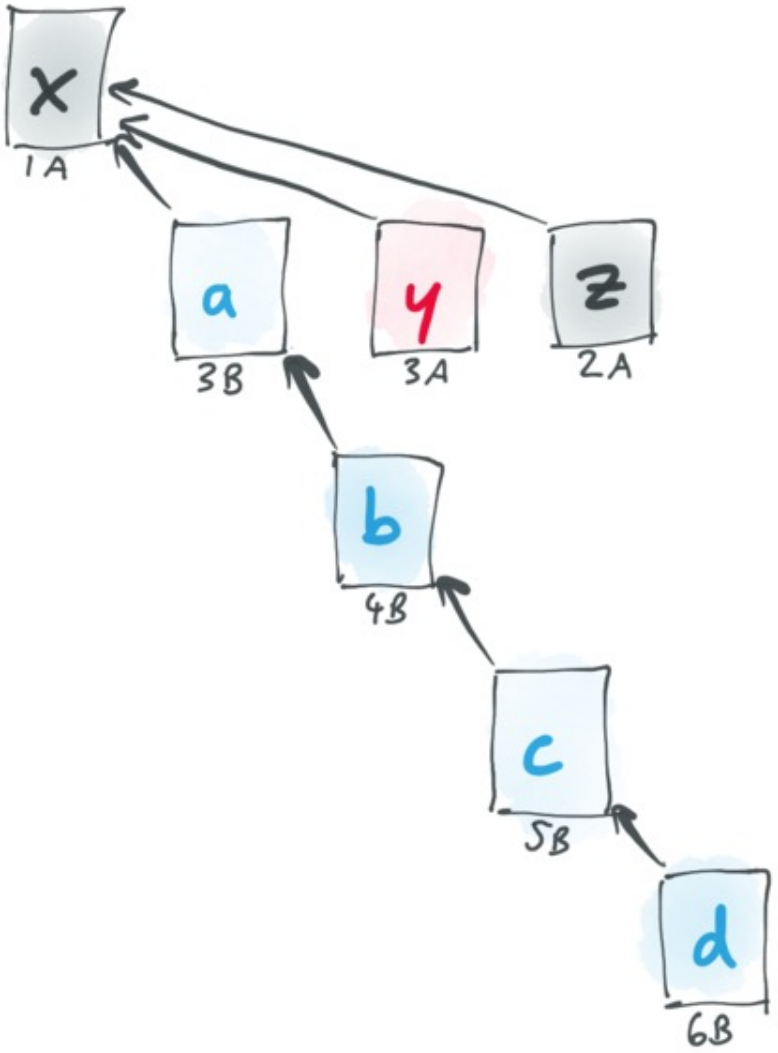
RGA is forward non-interleaving



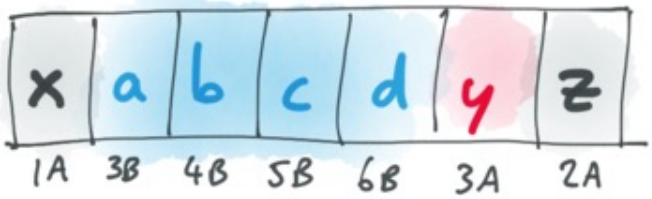
Depth-first traversal order:

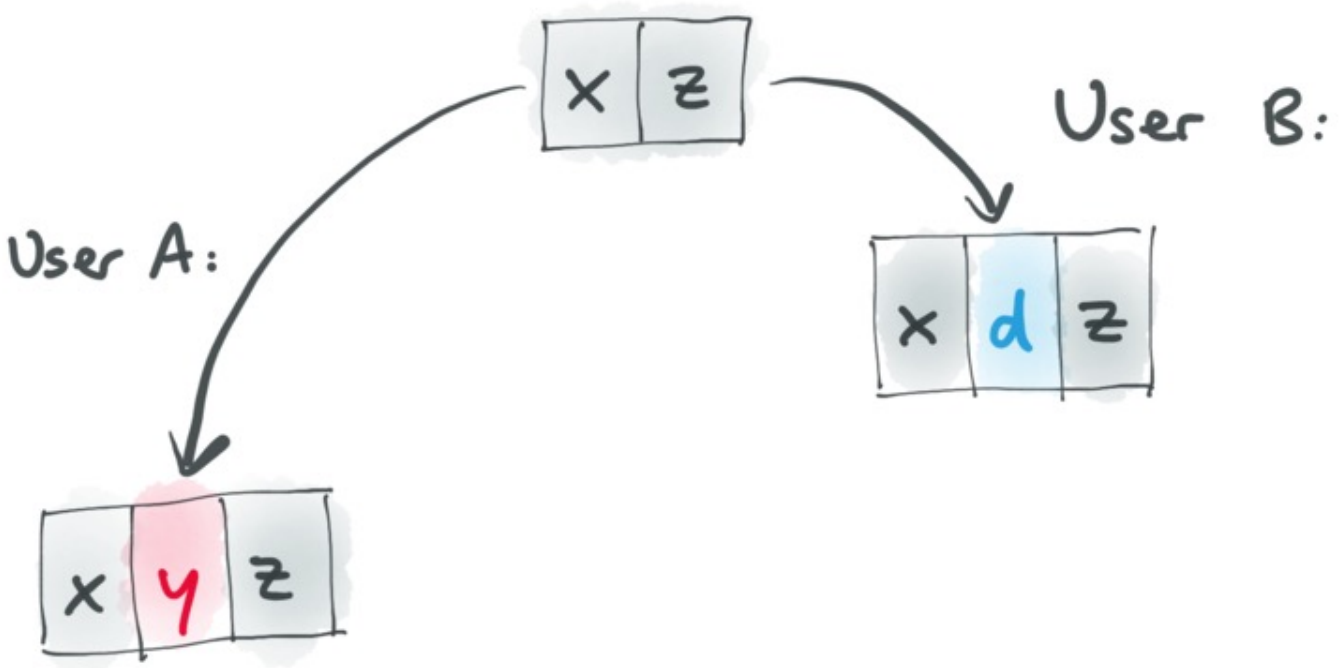


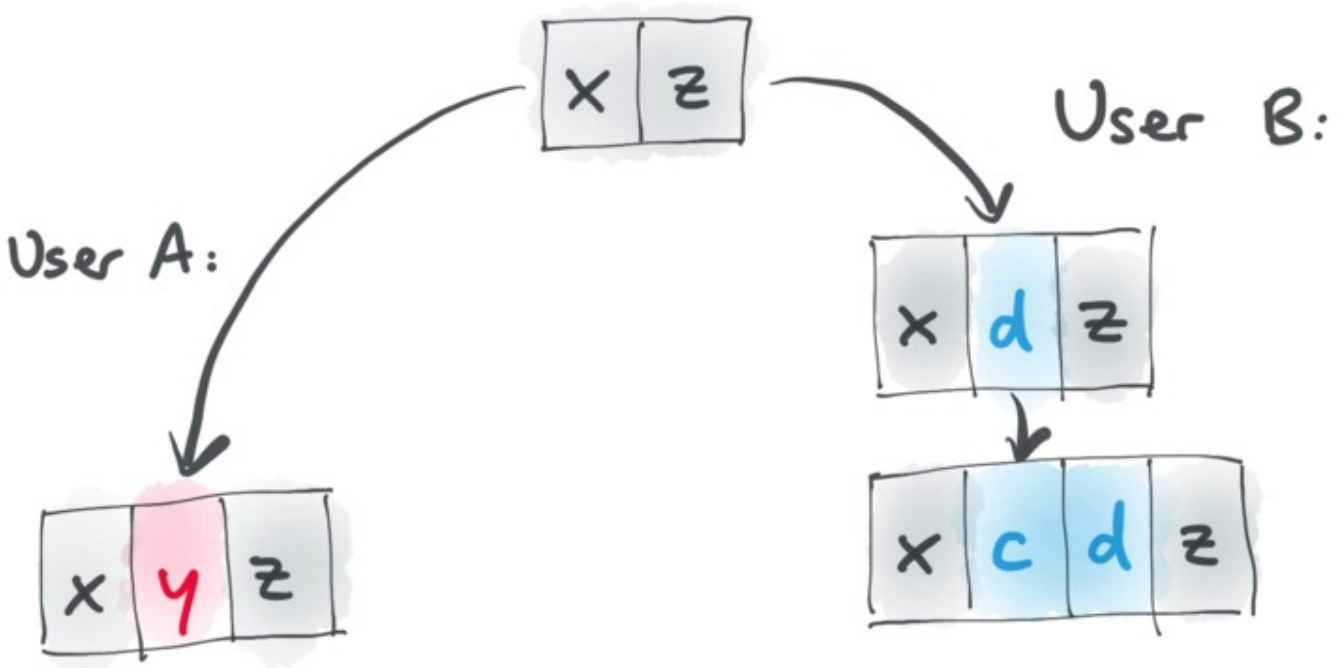
RGA is forward non-interleaving

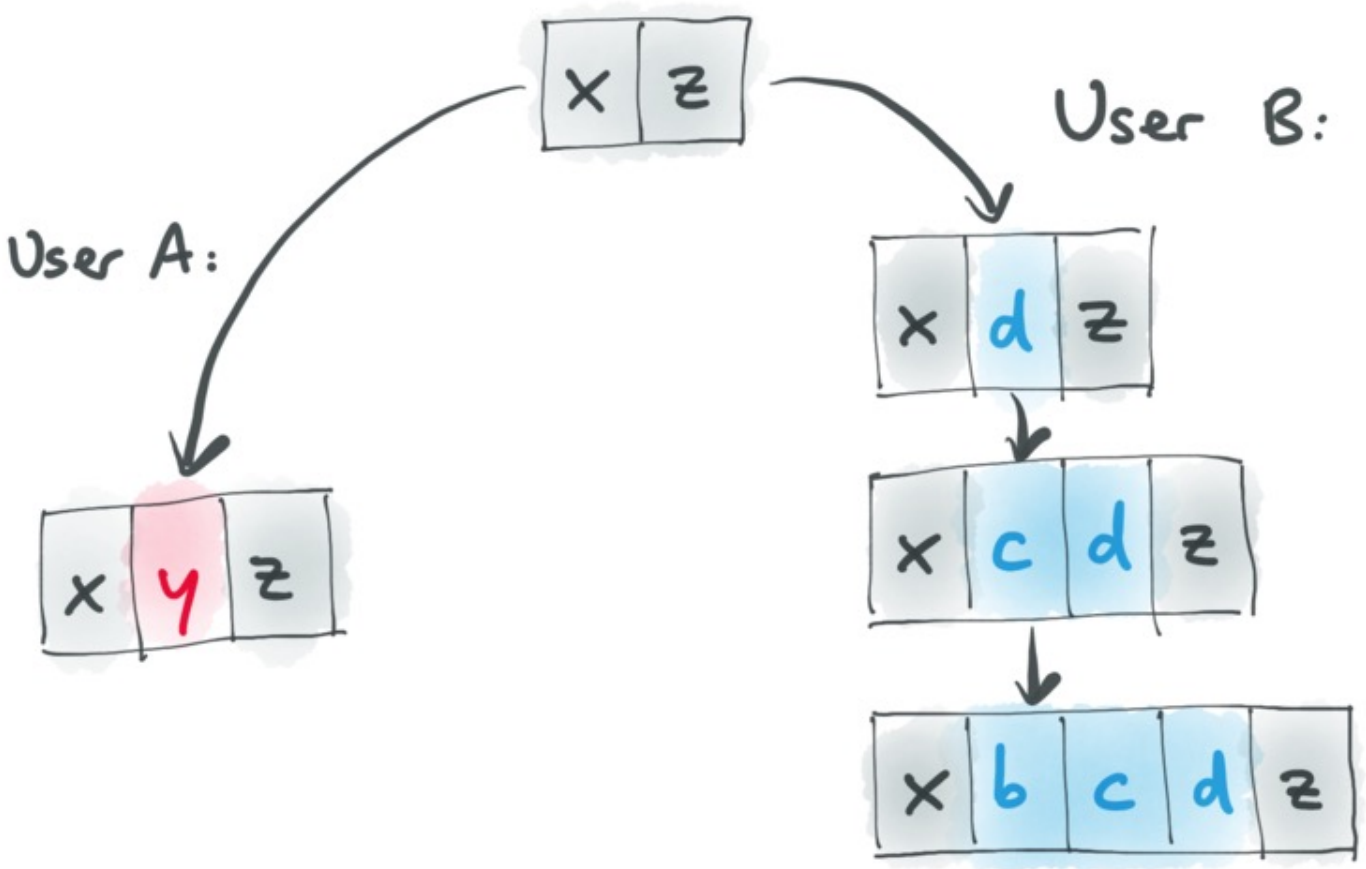


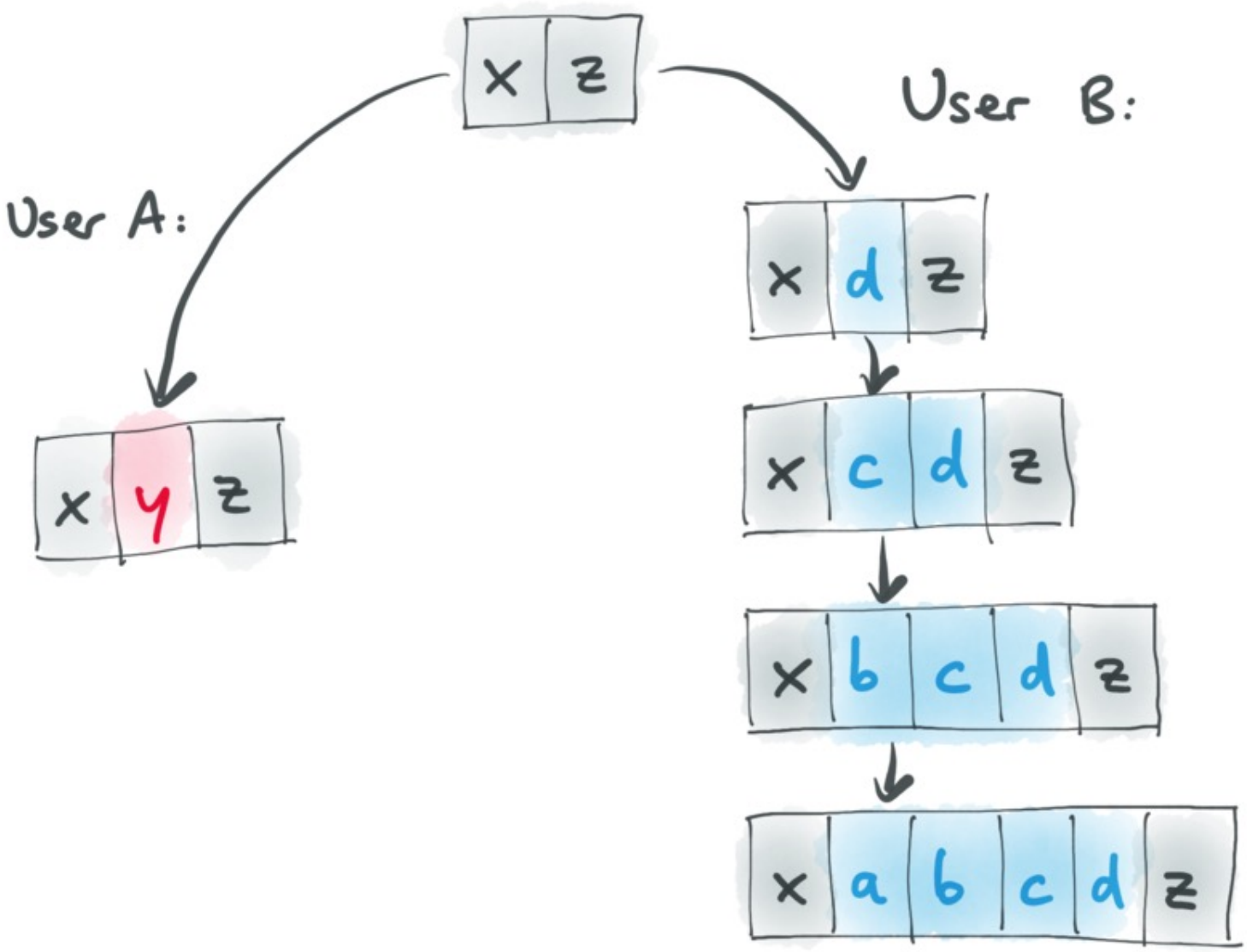
Depth-first traversal order:

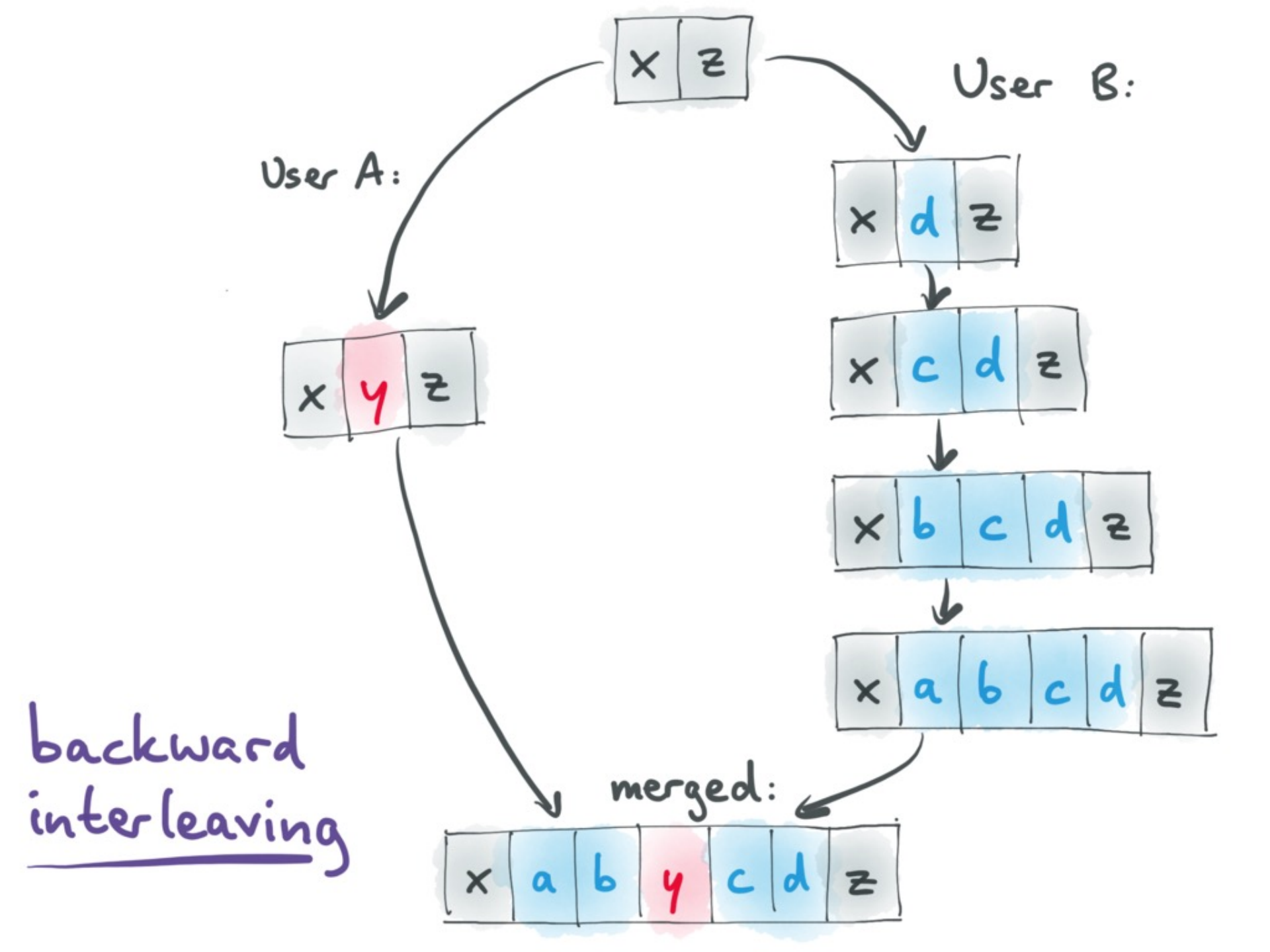


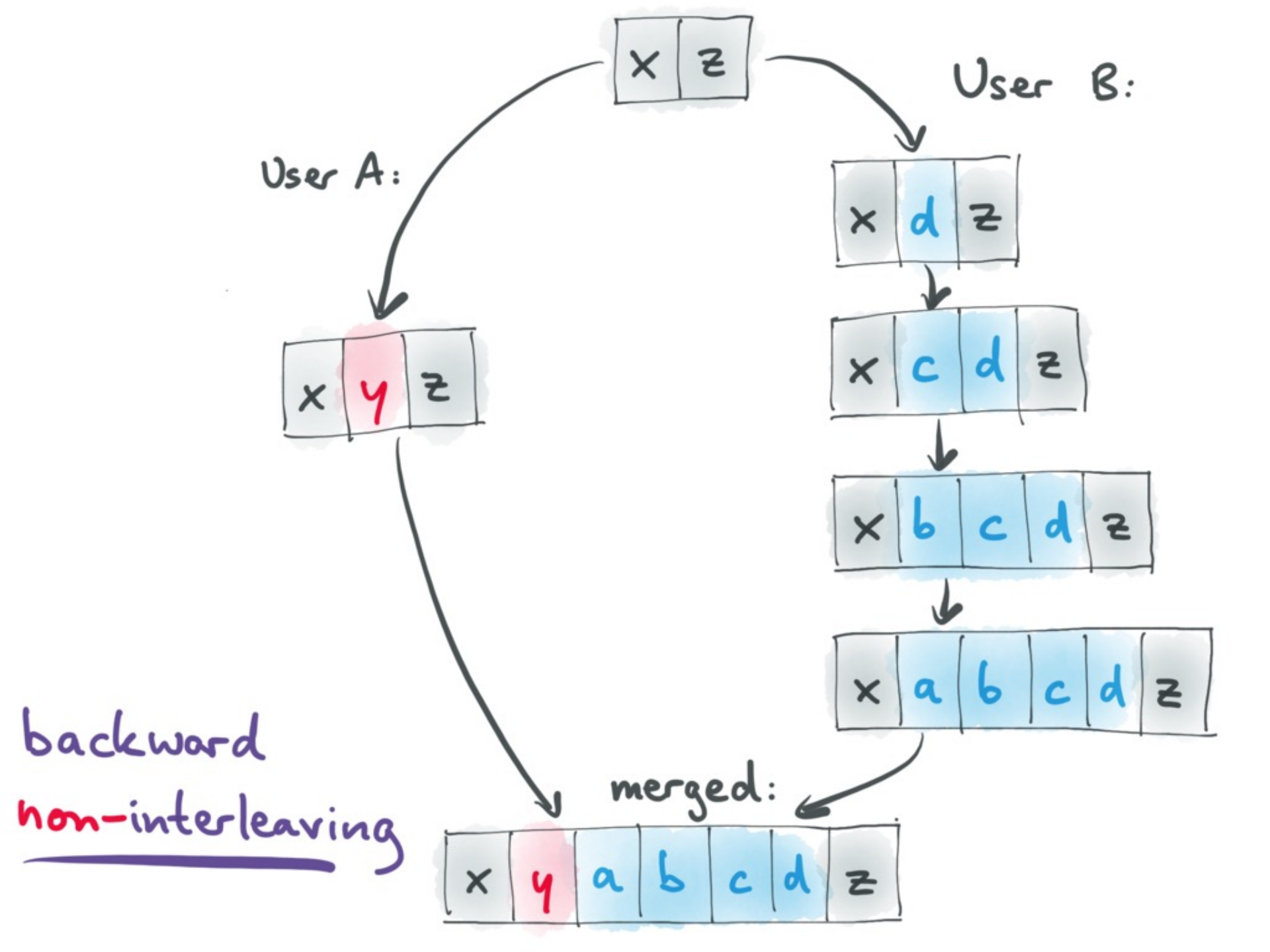

x z
User B:
x d z
User A:
x y z

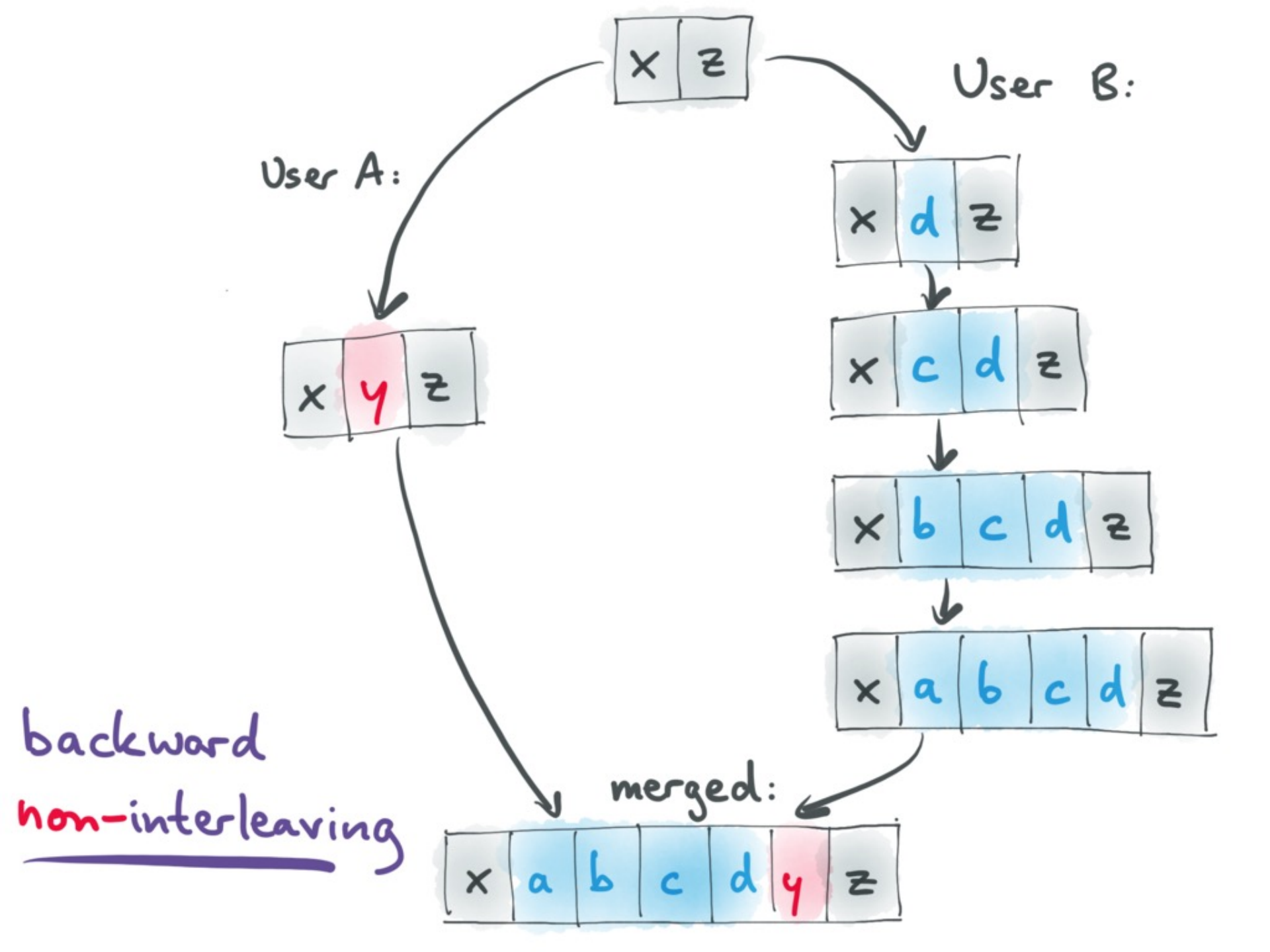

x z
User A:
x y z
User B:
x d z
x c d z


x z
User A:
x y z
User B:
x d z
x c d z
x b c d z

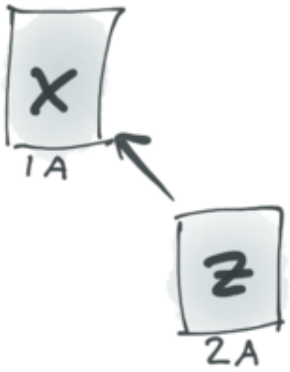

x z
User A:
x y z
User B:
x d z
x c d z
x b c d z
x a b c d z


x z
User A:
User B:
x y z
x d z
x c d z
x b c d z
x a b c d z
merged:
x a b y c d z
backward interleaving

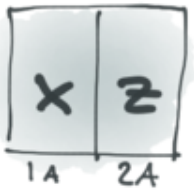

x z
User A:
User B:
x y z
x d z
x c d z
x b c d z
x a b c d z
merged:
x y a b c d z
backward
non-interleaving


x z
User A:
User B:
x y z
x d z
x c d z
x b c d z
x a b c d z
merged:
x a b c d y z
backward
non-interleaving

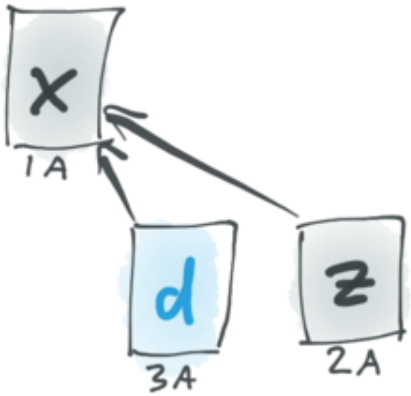
Backward interleaving in RGA



Depth-first traversal order:

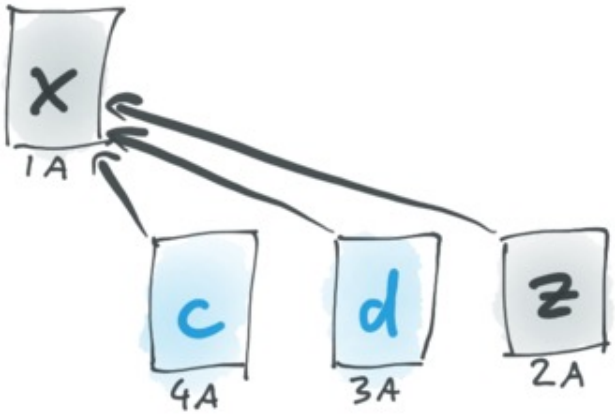


Backward interleaving in RGA

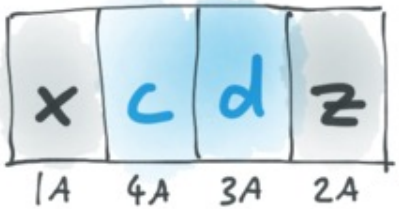


Depth-first traversal order:

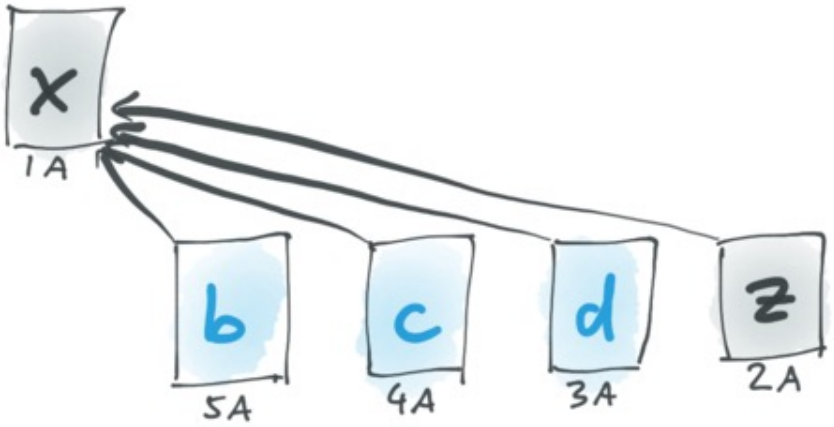
Backward interleaving in RGA



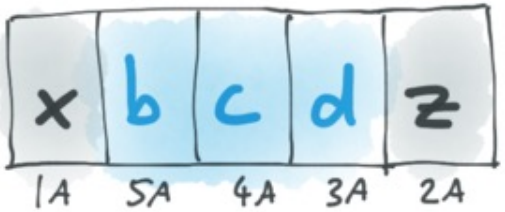
Depth-first traversal order:



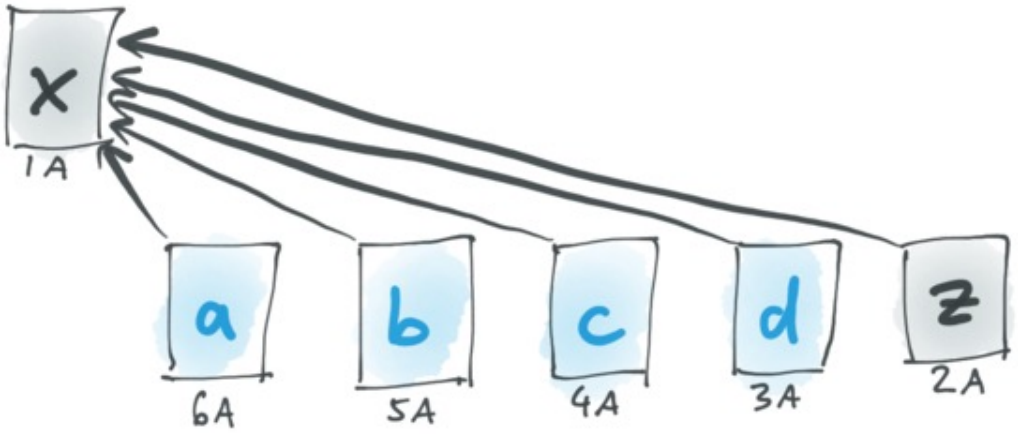
Backward interleaving in RGA



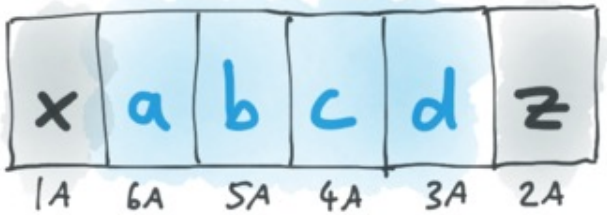
Depth-first traversal order:



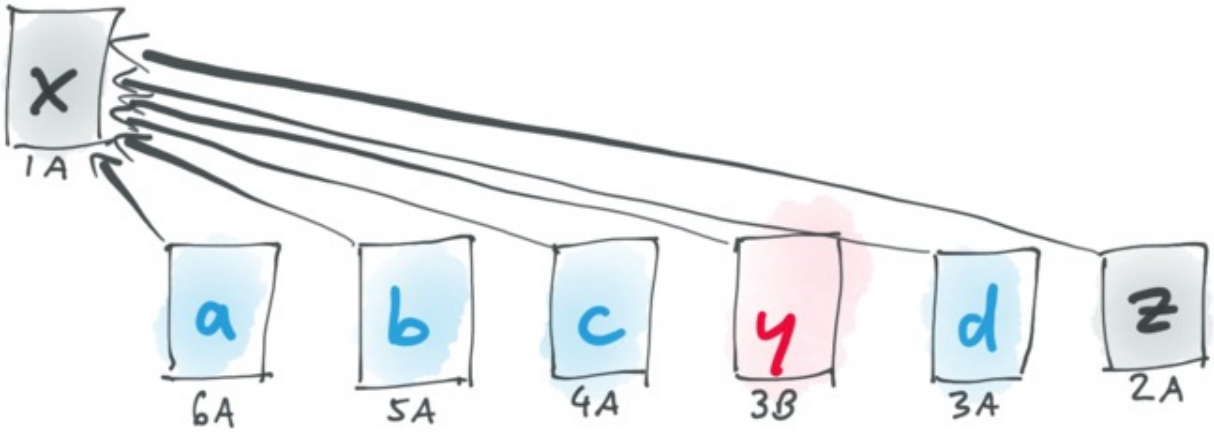
Backward interleaving in RGA



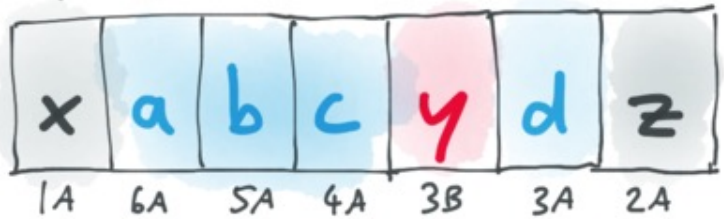
Depth-first traversal order:



Backward interleaving in RGA

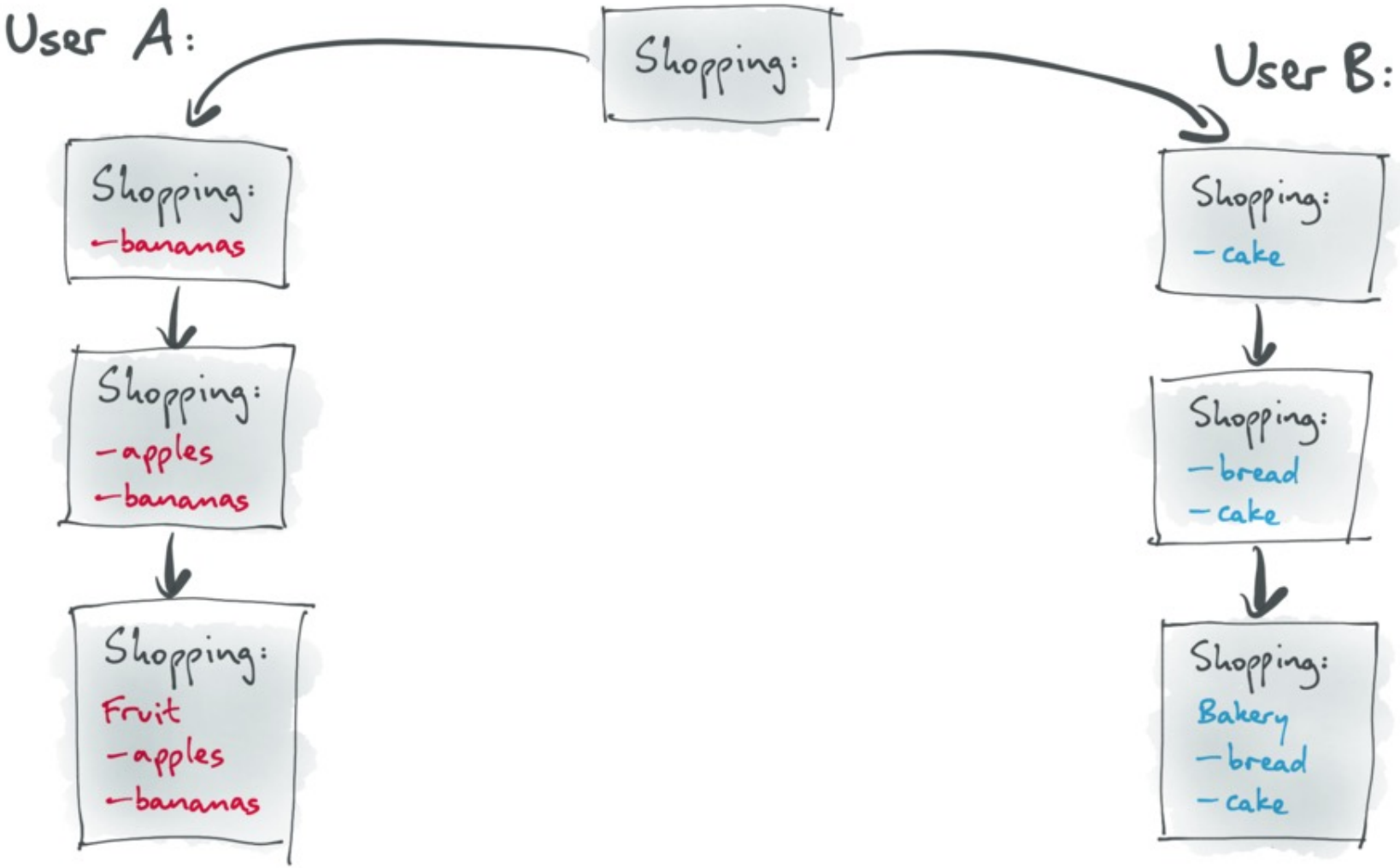


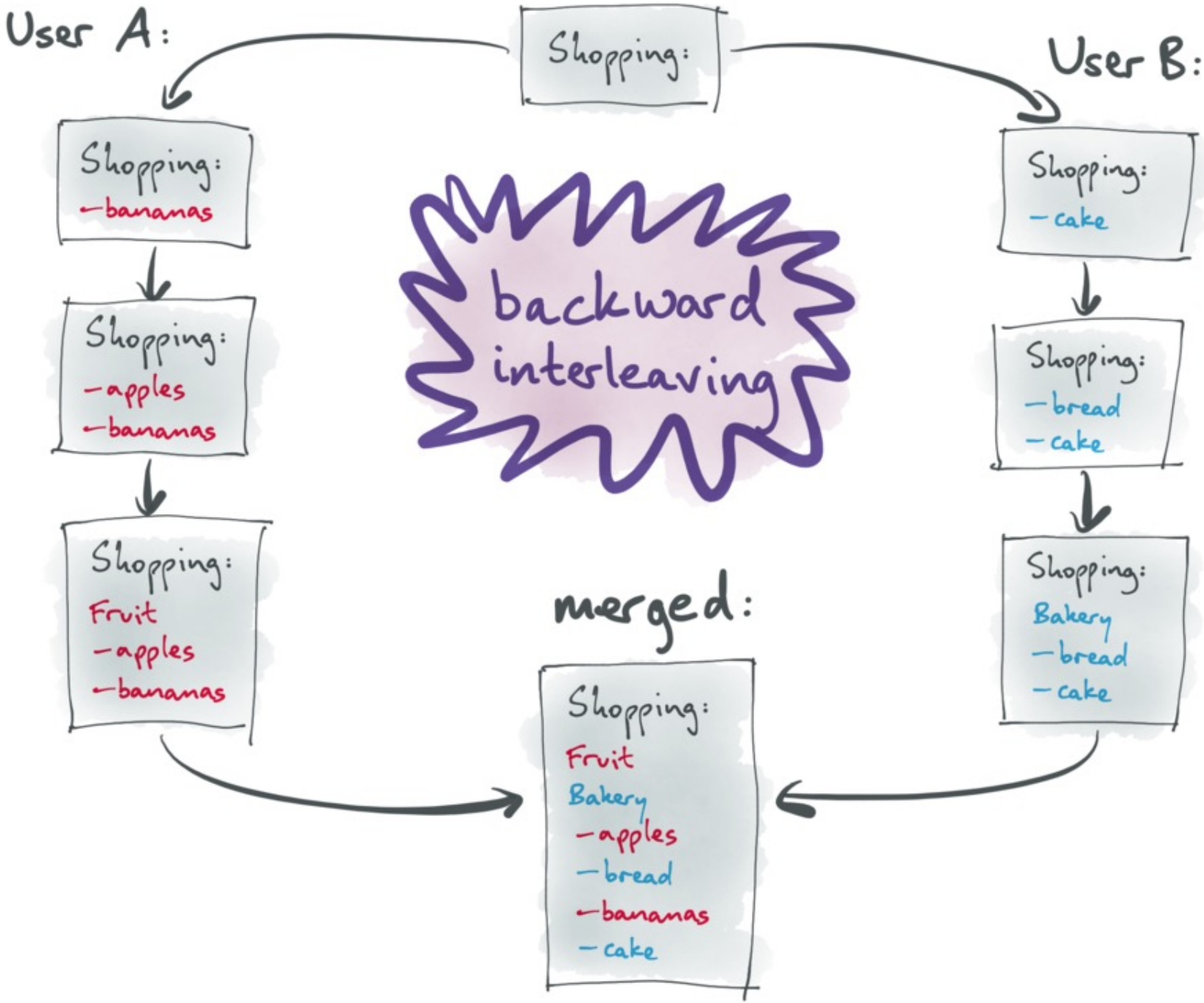
Depth-first traversal order:

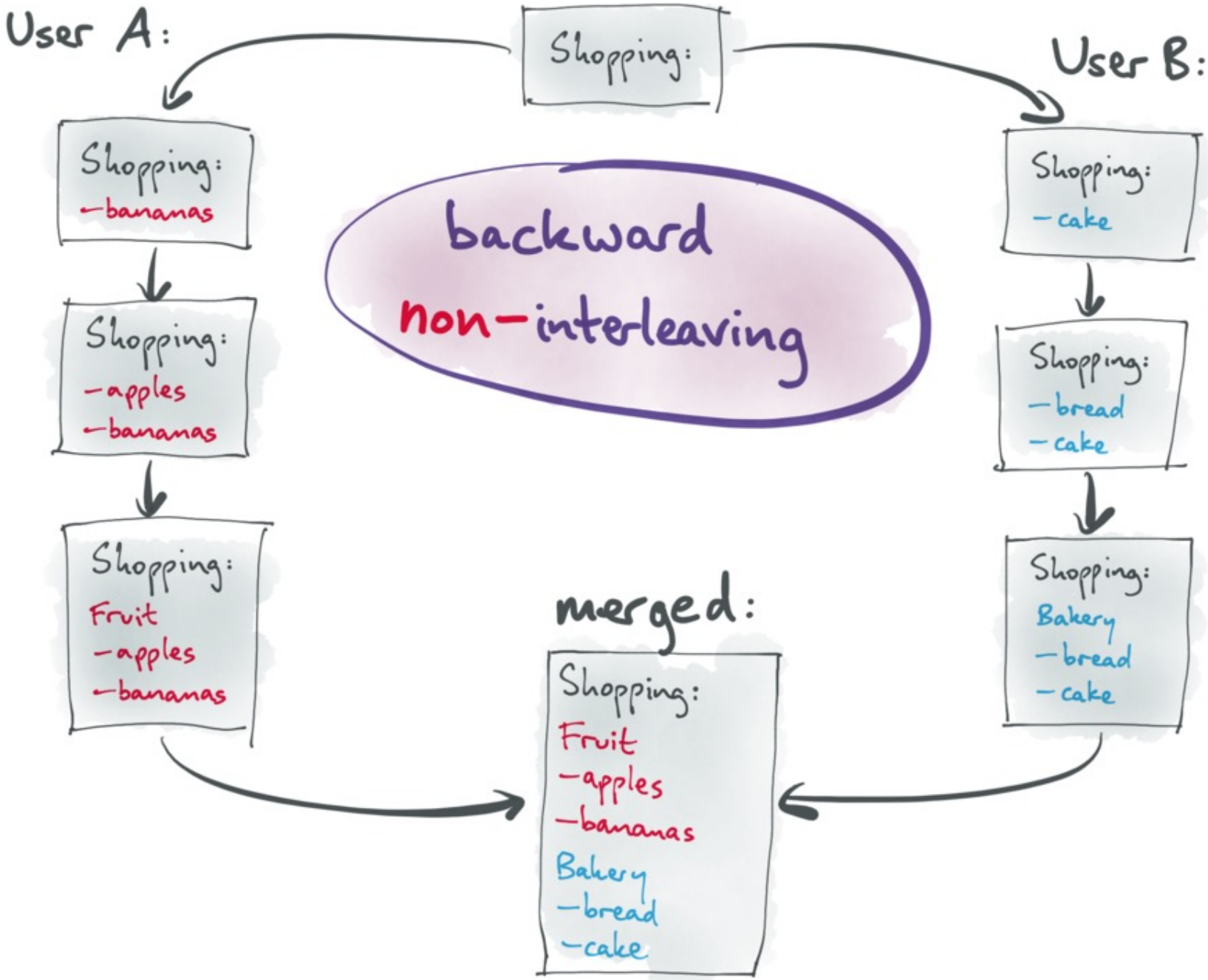



User A:
Shopping:
Shopping:
User B:
Shopping:
-bananas
Shopping:
- cake


User A:
Shopping:
User B:
Shopping:
-bananas
Shopping:
-cake
Shopping:
-apples
-bananas
Shopping:
-bread
-cake


User A:
Shopping:
User B:
Shopping:
-bananas
Shopping:
-apples
-bananas
Shopping:
Fruit
-apples
-bananas
Shopping:
-cake
Shopping:
-bread
-cake
Shopping:
Bakery
-bread
-cake


User A:
Shopping:
User B:
Shopping:
-bananas
Shopping:
-cake
backward interleaving
Shopping:
-apples
-bananas
Shopping:
-bread
-cake
Shopping:
Fruit
-apples
-bananas
merged:
Shopping:
Bakery
-bread
-cake
Shopping:
Fruit
Bakery
-apples
-bread
-bananas
-cake


User A:
Shopping:
User B:
Shopping:
-bananas
Shopping:
-cake
backward
non-interleaving
Shopping:
-apples
-bananas
Shopping:
-bread
-cake
Shopping:
Fruit
-apples
-bananas
Shopping:
Bakery
-bread
-cake
merged:
Shopping:
Fruit
-apples
-bananas
Bakery
-bread
-cake

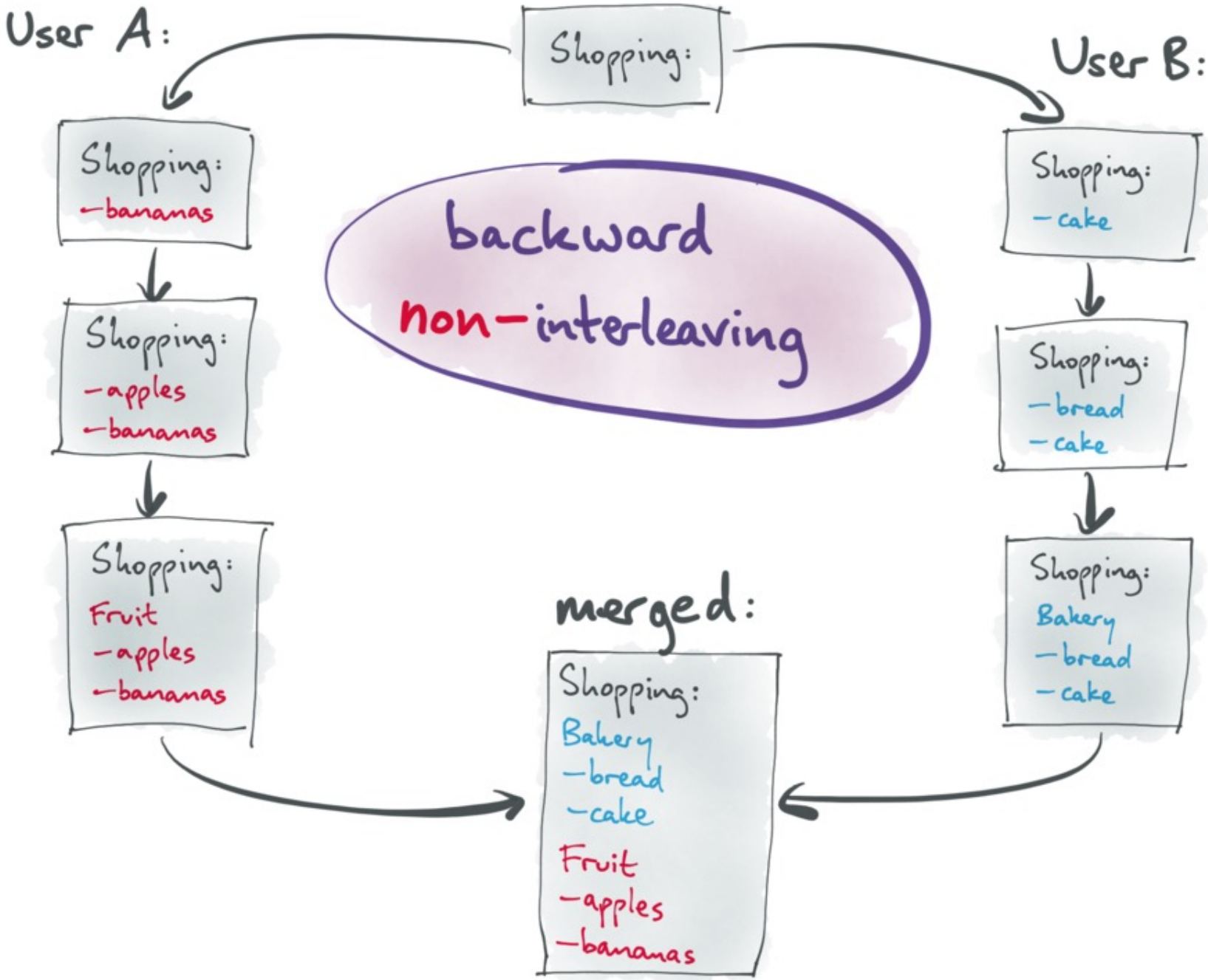

User A:
Shopping:
User B:
Shopping:
-bananas
Shopping:
-cake
backward
non-interleaving
Shopping:
-apples
-bananas
Shopping:
-bread
-cake
Shopping:
Fruit
-apples
-bananas
Shopping:
Bakery
-bread
-cake
merged:
Shopping:
Bakery
-bread
-cake
Fruit
-apples
-bananas

Table 1 Various algorithms' susceptibility to interleaving anomalies. Key: ● = interleaving can occur; ○ = we have not been able to find examples of interleaving; ○✓ = proven not to interleave; ↯ = algorithm may incorrectly reorder characters. Examples of anomalies appear in Appendix A.

Family	Algorithm	forward interleaving (one replica)	forward interleaving (multi-replica)	backward interleaving (one replica)	backward interleaving (multi-replica)
OT	adOPTed [25]	●	●	○	●
	Jupiter [18]	●	●	○	○
	GOT [31]	●	●	↯	↯
	SOCT2 [29]	●	●	●	●
	TTF [20]	●	●	○	●
CRDT	WOOT [22]	●	●	○	○
	Logoot [35]	●	●	●	●
	LSEQ [17]	●	●	●	●
	Treedoc [24]	●	●	●	●
	RGA [26]	○✓	○✓	●	●
	Yjs [10]	○✓	○✓	○	●
	Fugue (this work)	○✓	○✓	○✓	○✓

RGA is non-interleaving for forward insertions, but interleaves backward insertions.

Could swap forward/backward by inserting before ID, not after.

RGA is non-interleaving for forward insertions, but interleaves backward insertions.

Could swap forward/backward by inserting before ID, not after.

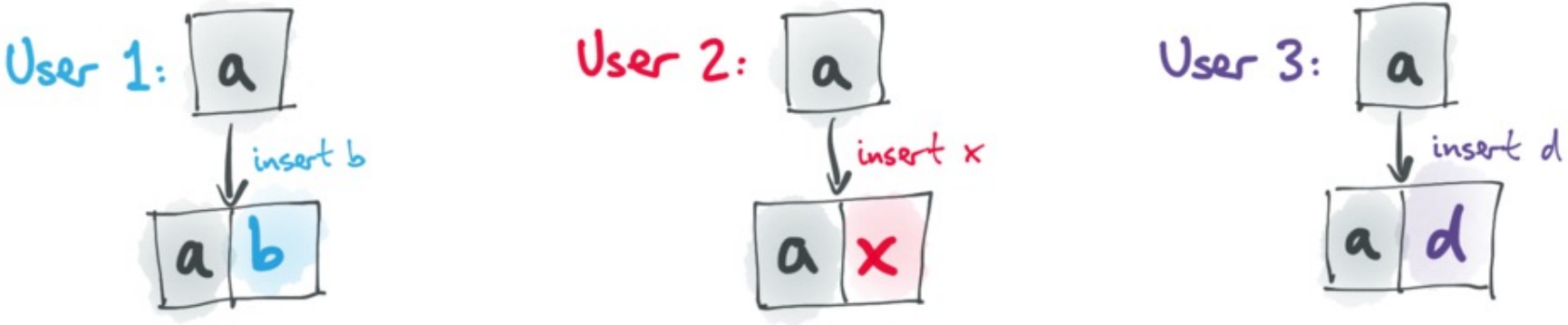
Can we make an algorithm that is non-interleaving for both forward and backward insertions?

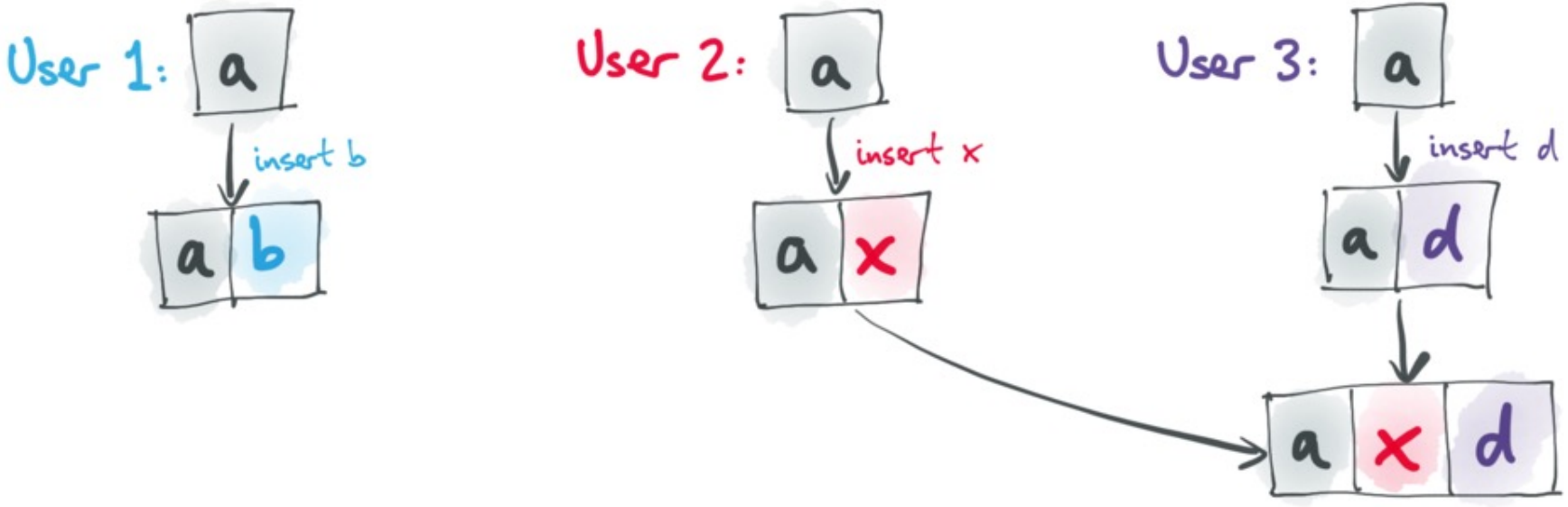
RGA is non-interleaving for forward insertions, but interleaves backward insertions.

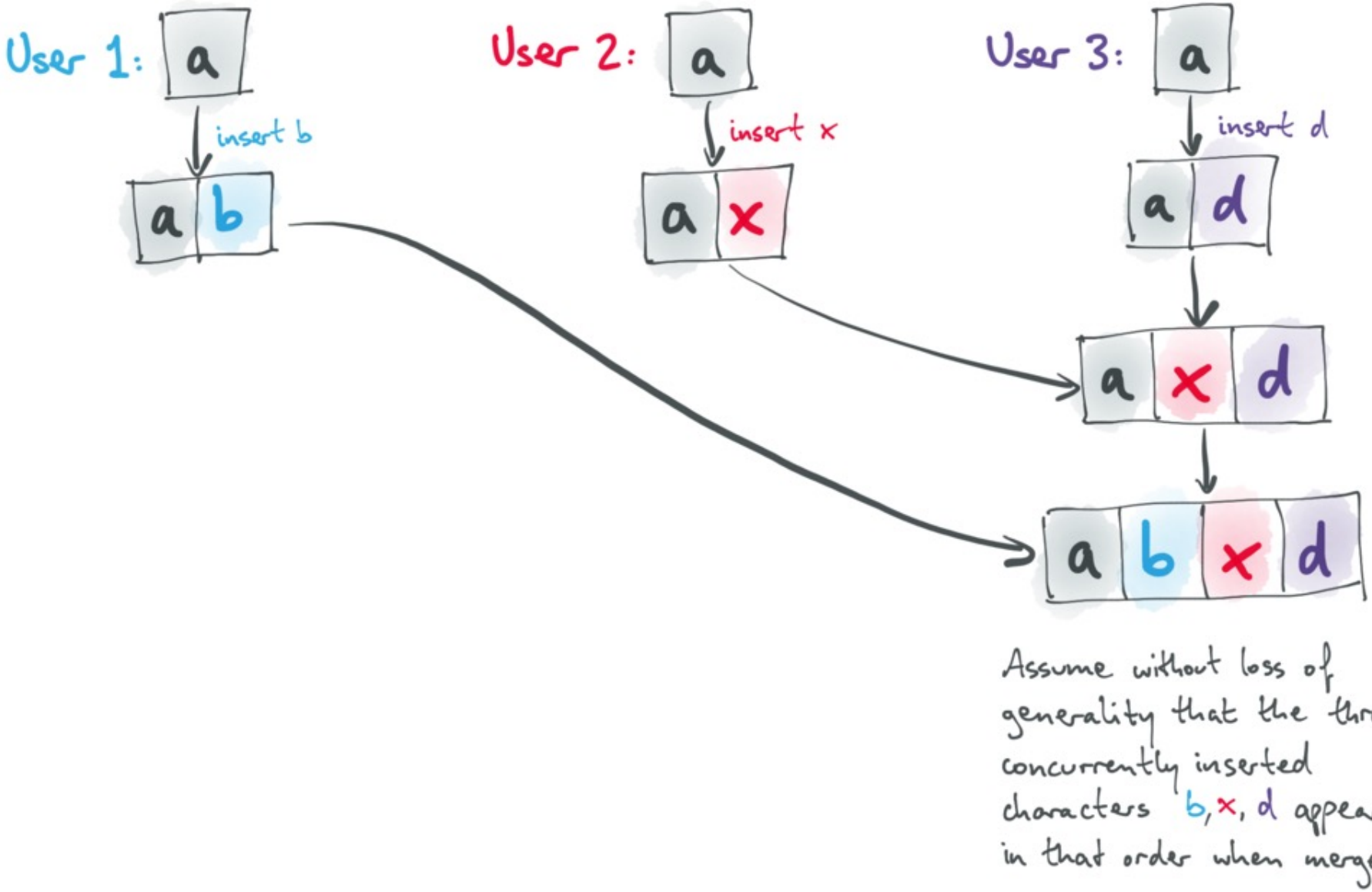
Could swap forward/backward by inserting before ID, not after.

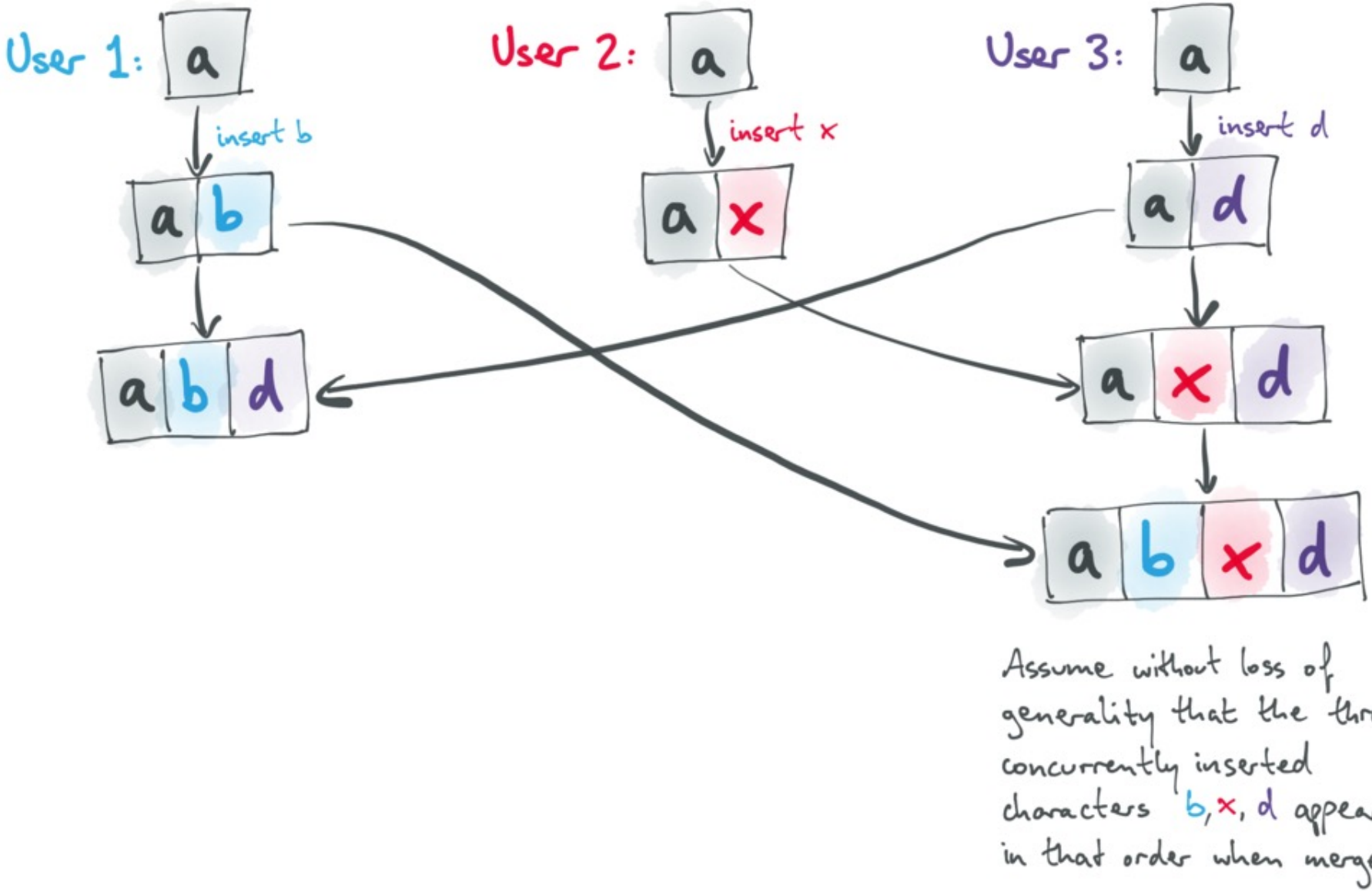
Can we make an algorithm that is non-interleaving for both forward and backward insertions?

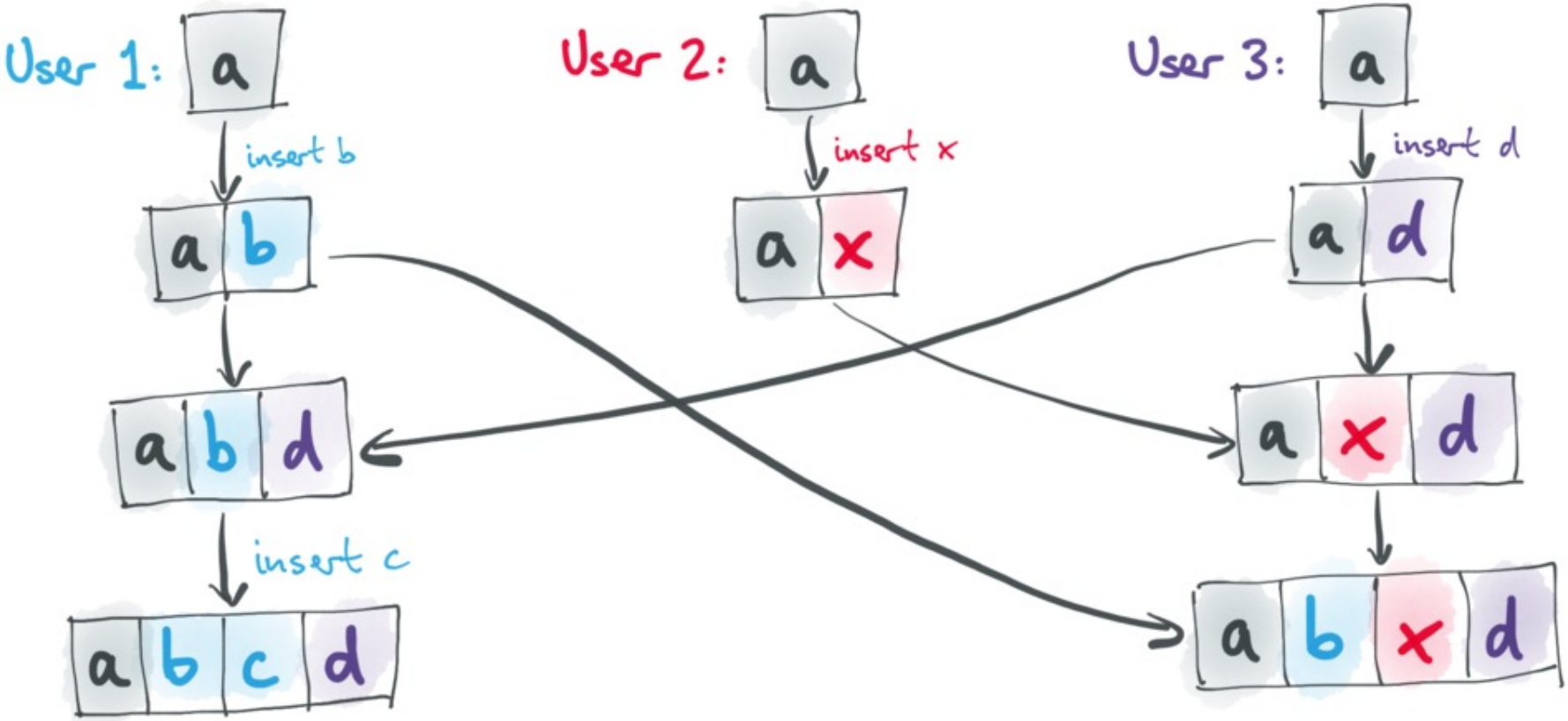
It is impossible for an algorithm to guarantee both forward and backward non-interleaving!


User 1:
a
insert b
a b
User 2:
a
insert x
a x
User 3:
a
insert d
a d

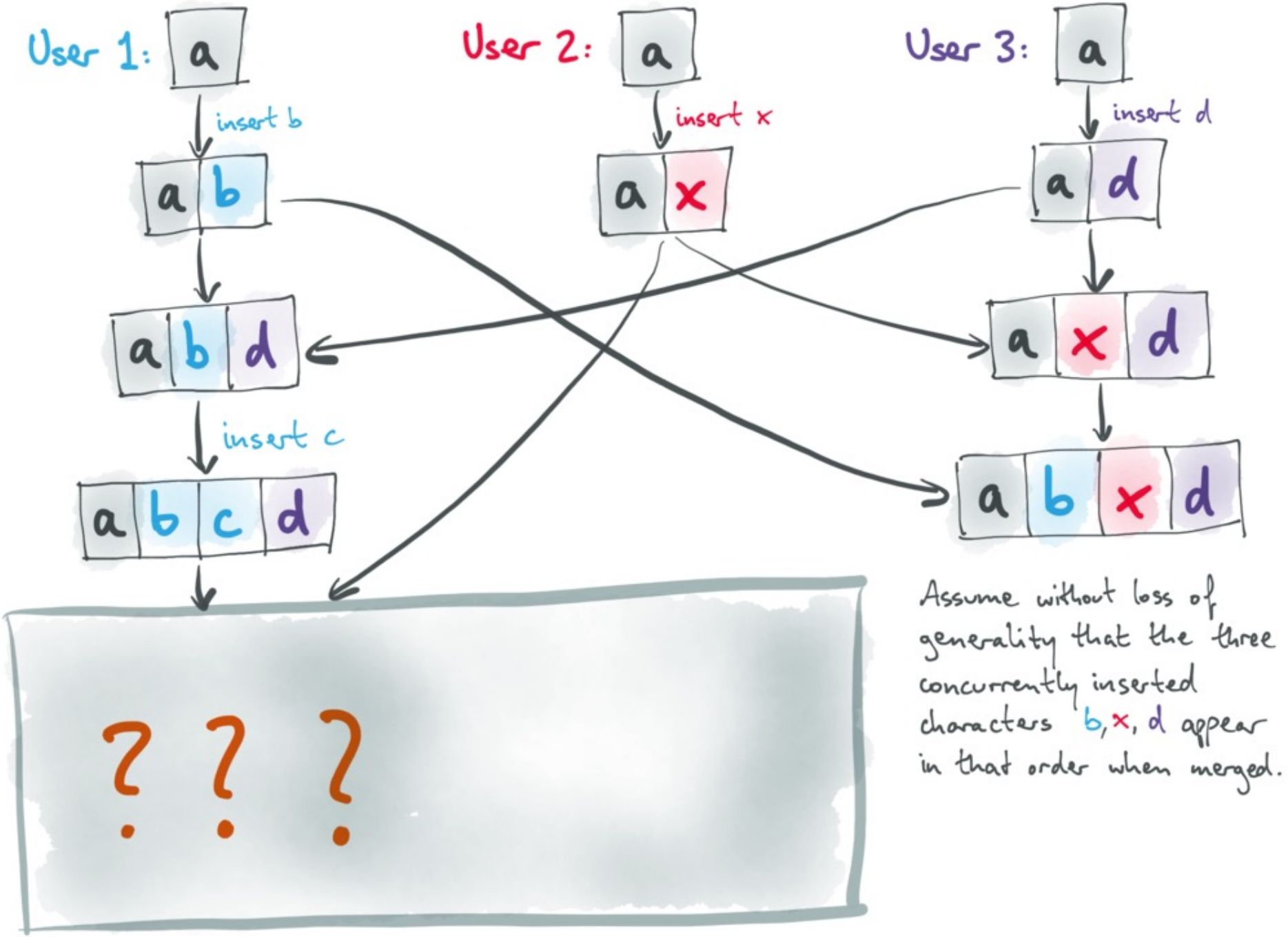

User 1:
a
insert b
a b
User 2:
a
insert x
a x
User 3:
a
insert d
a d
a x d

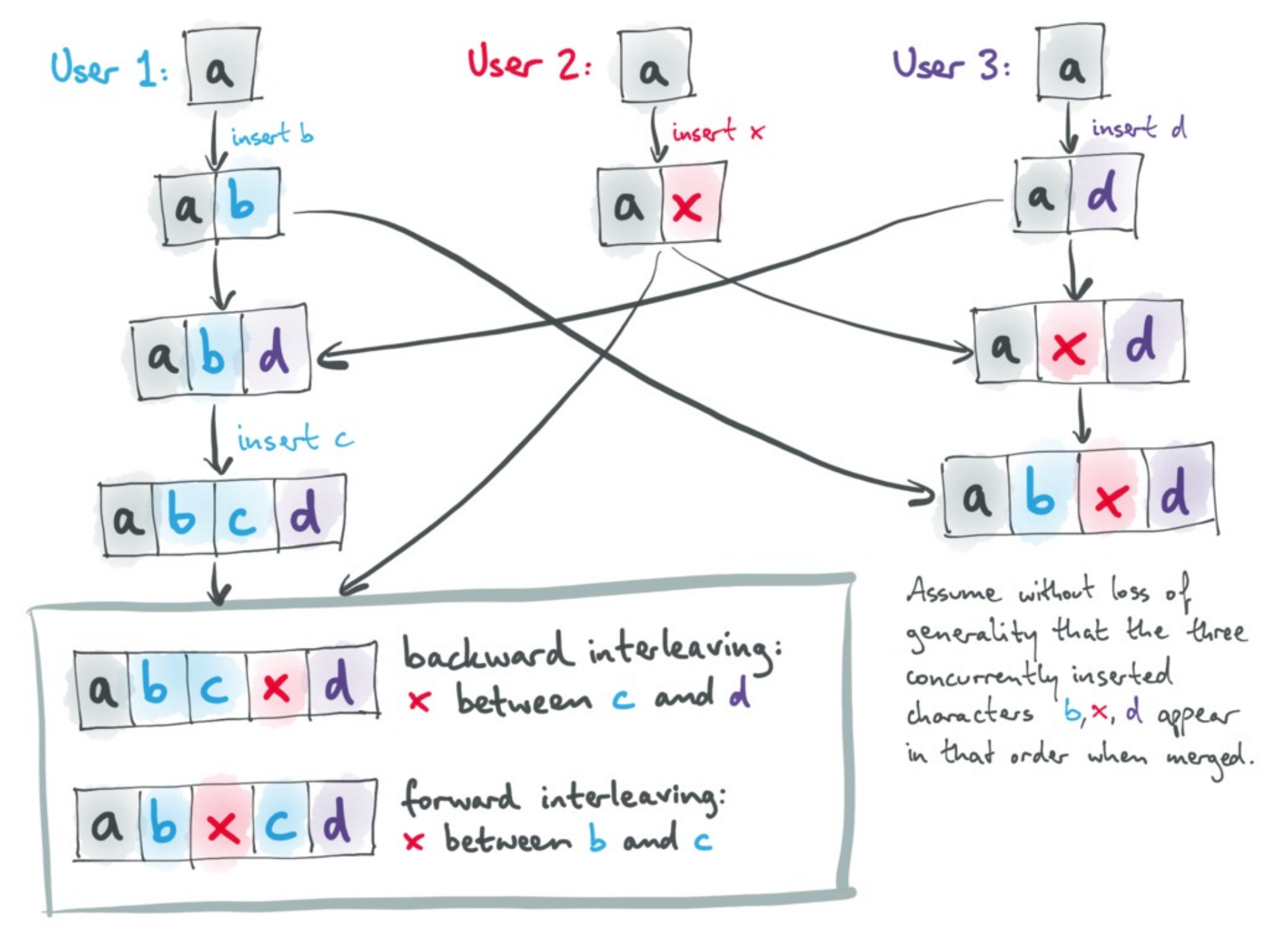

User 1:
a
insert b
a b
User 2:
a
insert x
a x
User 3:
a
insert d
a d
a x d
a b x d
Assume without loss of generality that the three concurrently inserted characters b, x, d appear in that order when merged.


User 1:
a
insert b
a b
a b d
User 2:
a
insert x
a x
User 3:
a
insert d
a d
a x d
a b x d
Assume without loss of generality that the three concurrently inserted characters b, x, d appear in that order when merged.



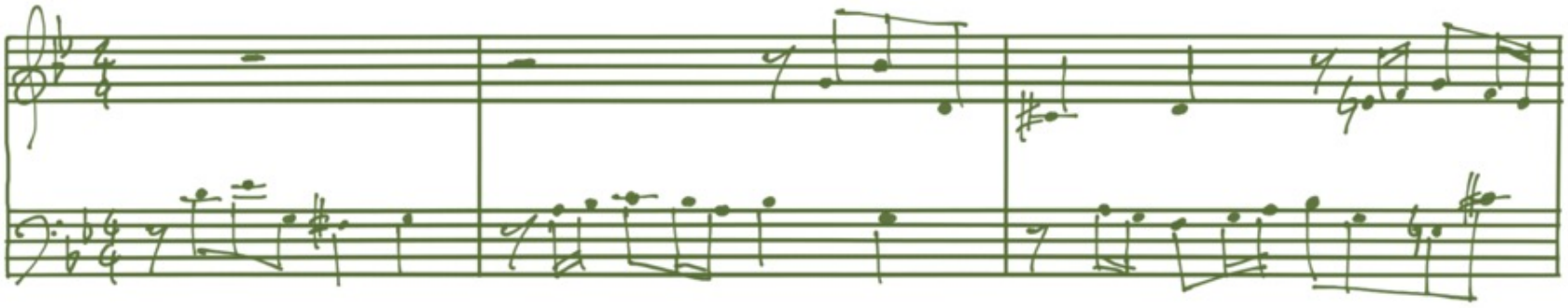
Assume without loss of generality that the three concurrently inserted characters b, x, d appear in that order when merged.


User 1:
a
insert b
a b
a b d
insert c
a b c d
User 2:
a
insert x
a x
User 3:
a
insert d
a d
a x d
a b x d
? ? ?
Assume without loss of generality that the three concurrently inserted characters b, x, d appear in that order when merged.

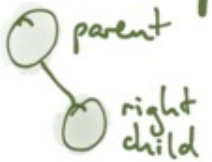

User 1:
a
insert b
a b
a b d
insert c
a b c d
User 2:
a
insert x
a x
a x d
a b x d
User 3:
a
insert d
a d
a b c x d
backward interleaving:
x between c and d
a b x c d
forward interleaving:
x between b and c
Assume without loss of generality that the three concurrently inserted characters b, x, d appear in that order when merged.

Fugue

A MAXIMALLY NON-INTERLEAVING TEXT CRDT



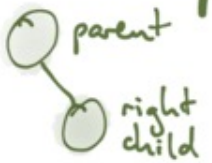
THE FUGUE ALGORITHM : INTUITION

- A tree where every node is either a right child 

or a left child 

(note: not a binary tree — a node may have multiple left and multiple right children)

THE FUGUE ALGORITHM : INTUITION

- A tree where every node is either a right child 

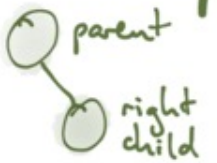
or a left child 

(note: not a binary tree — a node may have multiple left and multiple right children)

- **Depth-first in-order traversal**

(first visit all left subtrees, then the node itself, then all right subtrees.
visit same-side siblings in ID/timestamp order.)

THE FUGUE ALGORITHM : INTUITION

- A tree where every node is either a right child 

or a left child 

(note: not a binary tree — a node may have multiple left and multiple right children)

- **Depth-first in-order traversal**

(first visit all left subtrees, then the node itself, then all right subtrees.
visit same-side siblings in ID/timestamp order.)

- To insert at index i:
  - if character at index i-1 has no right children, make the new character a right child of the character at index i-1
  - otherwise, make new character a left child of character at index i

References

- M. Weidner, J. Gentle, M. Kleppmann. The Art of the Fugue: Minimizing Interleaving in Collaborative Text Editing. Preprint, 2023. https://arxiv.org/abs/2305.00583
- K.R. McKelvey, S. Jenson, E. Wagner, B. Cook, M. Kleppmann. Upwelling: Combining real-time collaboration with version control for writers. Ink & Switch, 2023. https://www.inkandswitch.com/upwelling/
- G. Litt, S. Lim, M. Kleppmann, P. van Hardenberg. Peritext: A CRDT for Collaborative Rich Text Editing. CSCW 2022. doi:10.1145/3555644
- M. Kleppmann, A. Wiggins, P. van Hardenberg, M. McGranaghan. Local-first software: You own your data, in spite of the cloud. Onward! 2019. doi:10.1145/3359591.3359737
- G. Oster, P. Urso, P. Molli, A. Imine. Proving correctness of transformation functions in collaborative editing systems. INRIA Tech Report, RR-5795, 2005. https://hal.inria.fr/inria-00071213/
- M. Ressel, D. Nitsche-Ruhland, R. Gunzenhäuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. CSCW 1996. doi:10.1145/240080.240305
- D.A. Nichols, P. Curtis, M. Dixon, J. Lamping. High-latency, low-bandwidth windowing in the Jupiter collaboration system. UIST 1995. doi:10.1145/215585.215706

Project suggestions

- For SOCT2 we have shown interleaving in an implementation of the algorithms (https://github.com/ept/insert-interleaving) but not fully explained how it happens. Can you provide an explanation?
- There are various CRDT implementations besides Automerge (e.g. https://github.com/yjs/yjs, https://github.com/josephg/diamond-types). Please pick one of these, and explain in detail how their algorithm works (you will need to read the source).
- CRDT implementations differ in their support for version control use cases (e.g. diffing two document versions). Can you prototype branching and diffing with one or more of these libraries mentioned above, test the performance, and report which issues you run into?
- Upwelling demonstrates one possible design for integrating version control into a rich text editor. Can you come up with a design for version control in another type of app, such as a spreadsheet, or presentation software?