Automerge:

A new foundation for collaboration software

MARTIN KLEPPMANN @martinkl
UNIVERSITY OF CAMBRIDGE

UNIVERSITY OF CAMBRIDGE

Ink & Switch

Thank you to my supporters

LEVERHULME TRUST

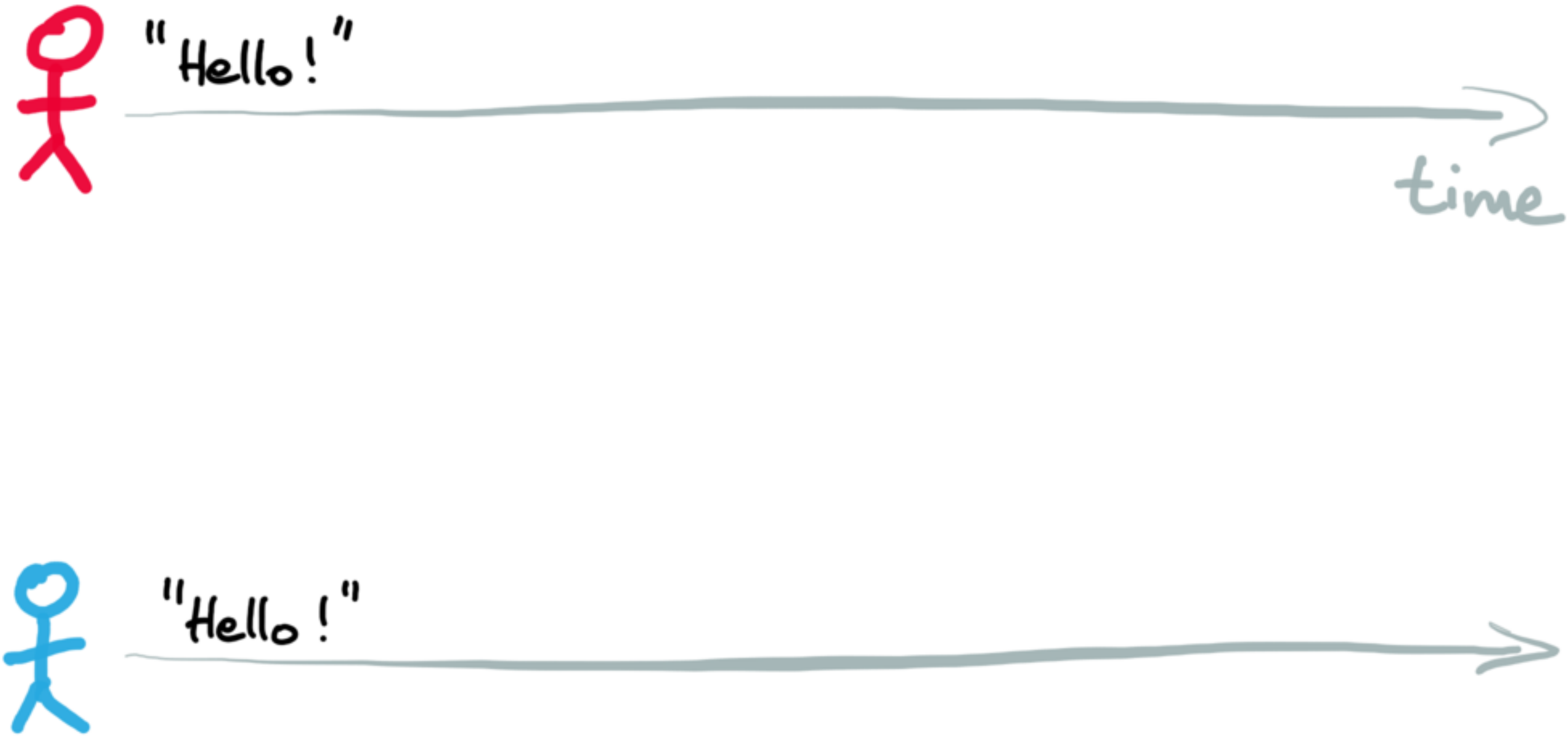
Isaac Newton Trust

NOKIA Bell Labs

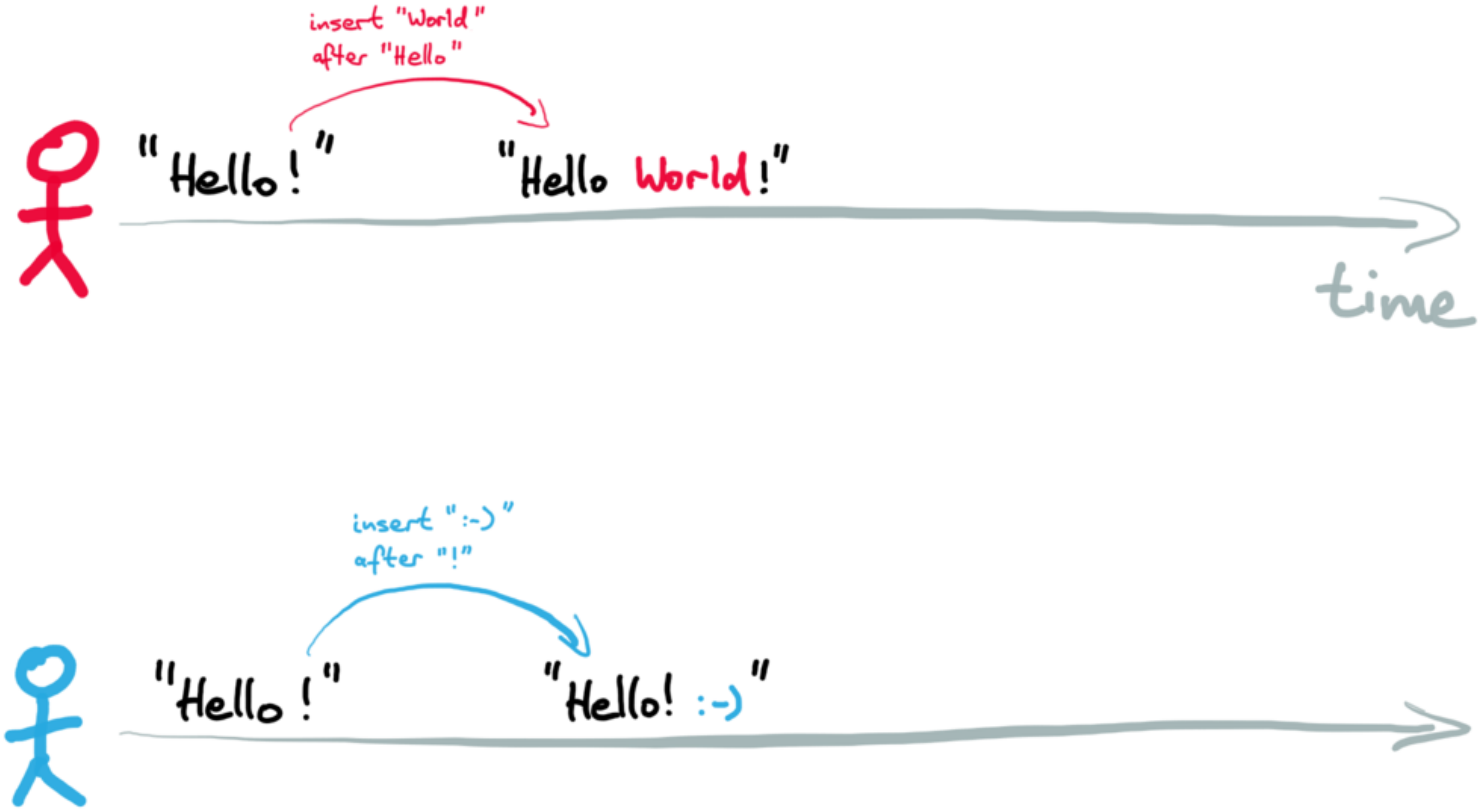
Crowdfunding supporters include: Ably, Adrià Arcarons, Chet Corcos, Macrometa, Mintter, David Pollak, Prisma, RelationalAI, SoftwareMill, Talent Formation Network, Adam Wiggins

https://www.patreon.com/martinkl

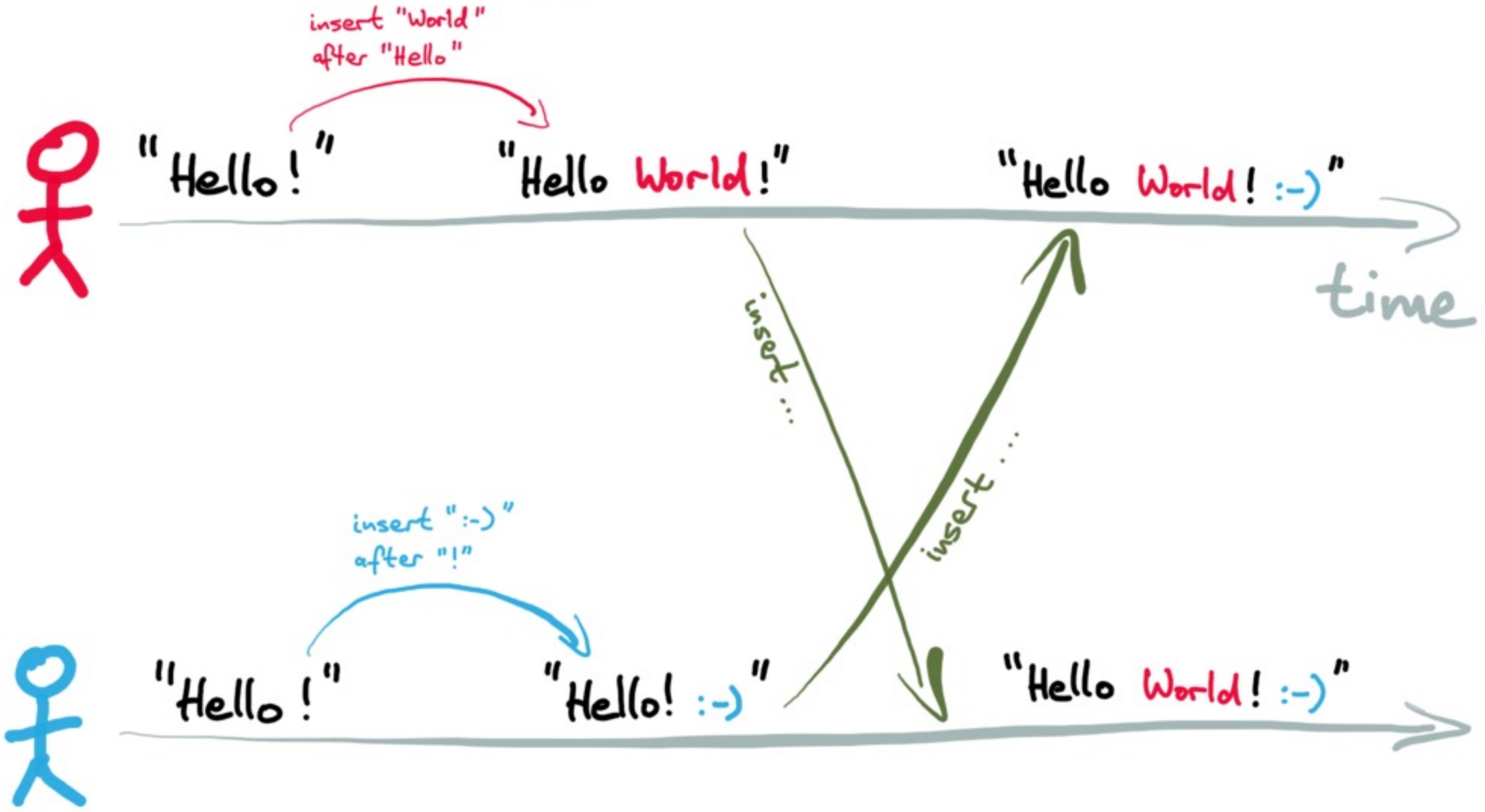
Example: Text editing



Example: Text editing

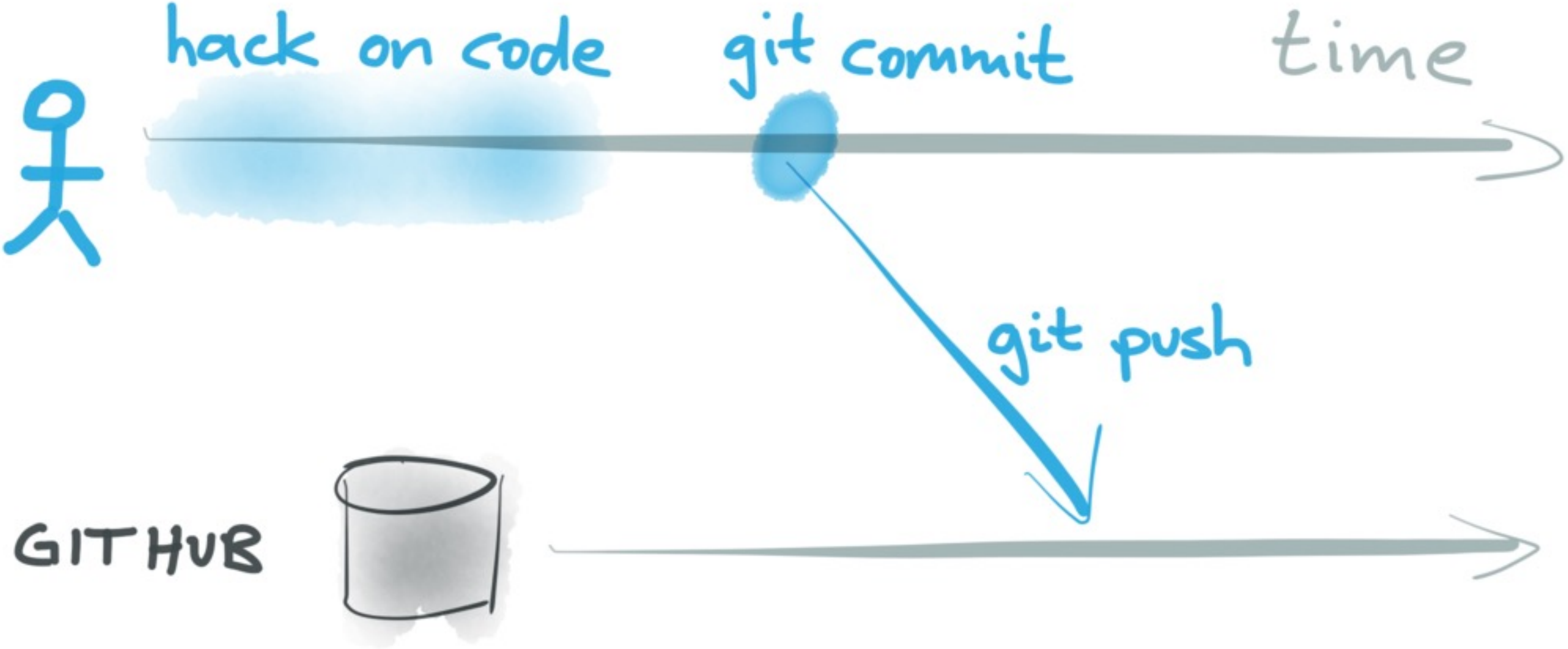


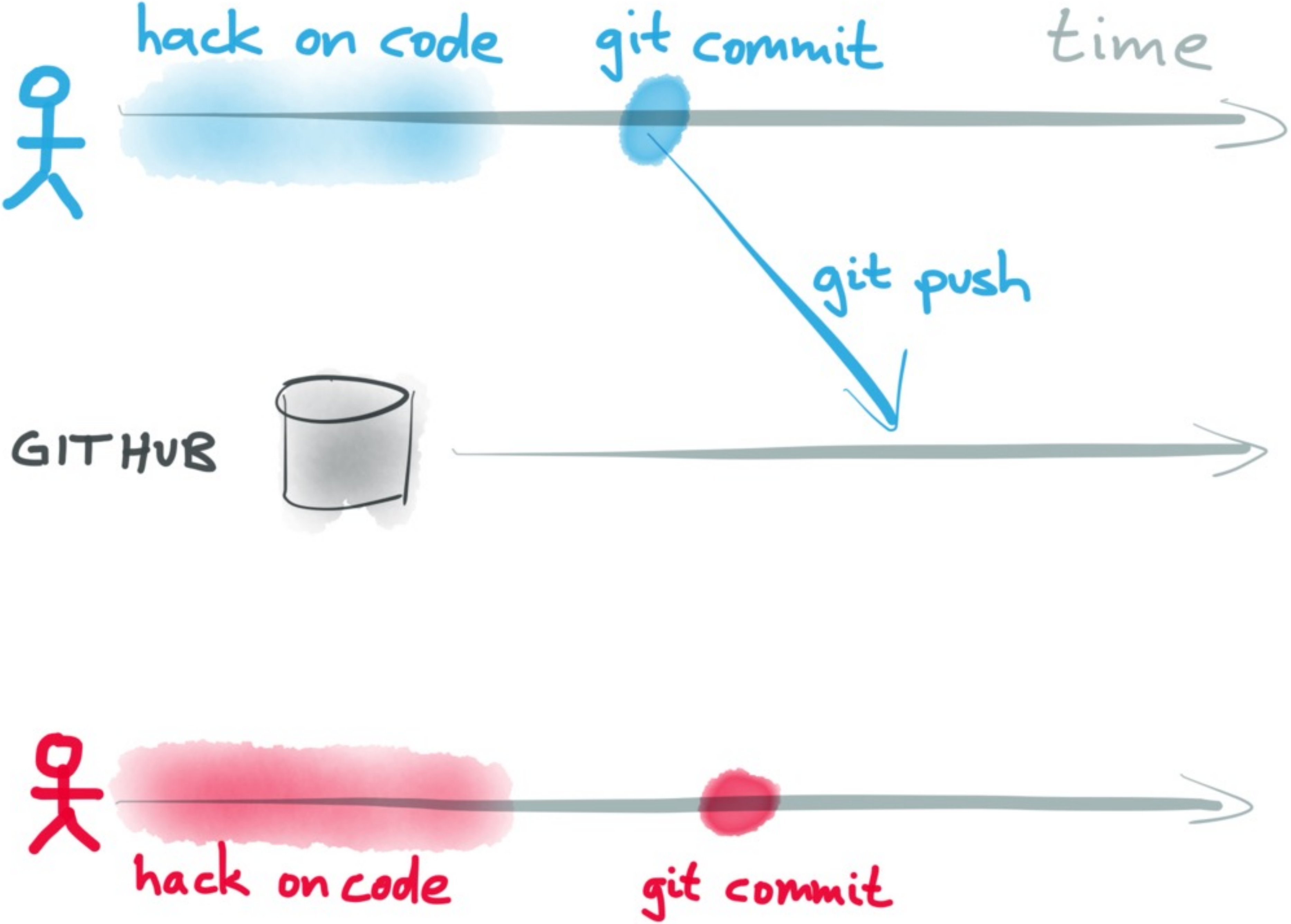
Example: Text editing

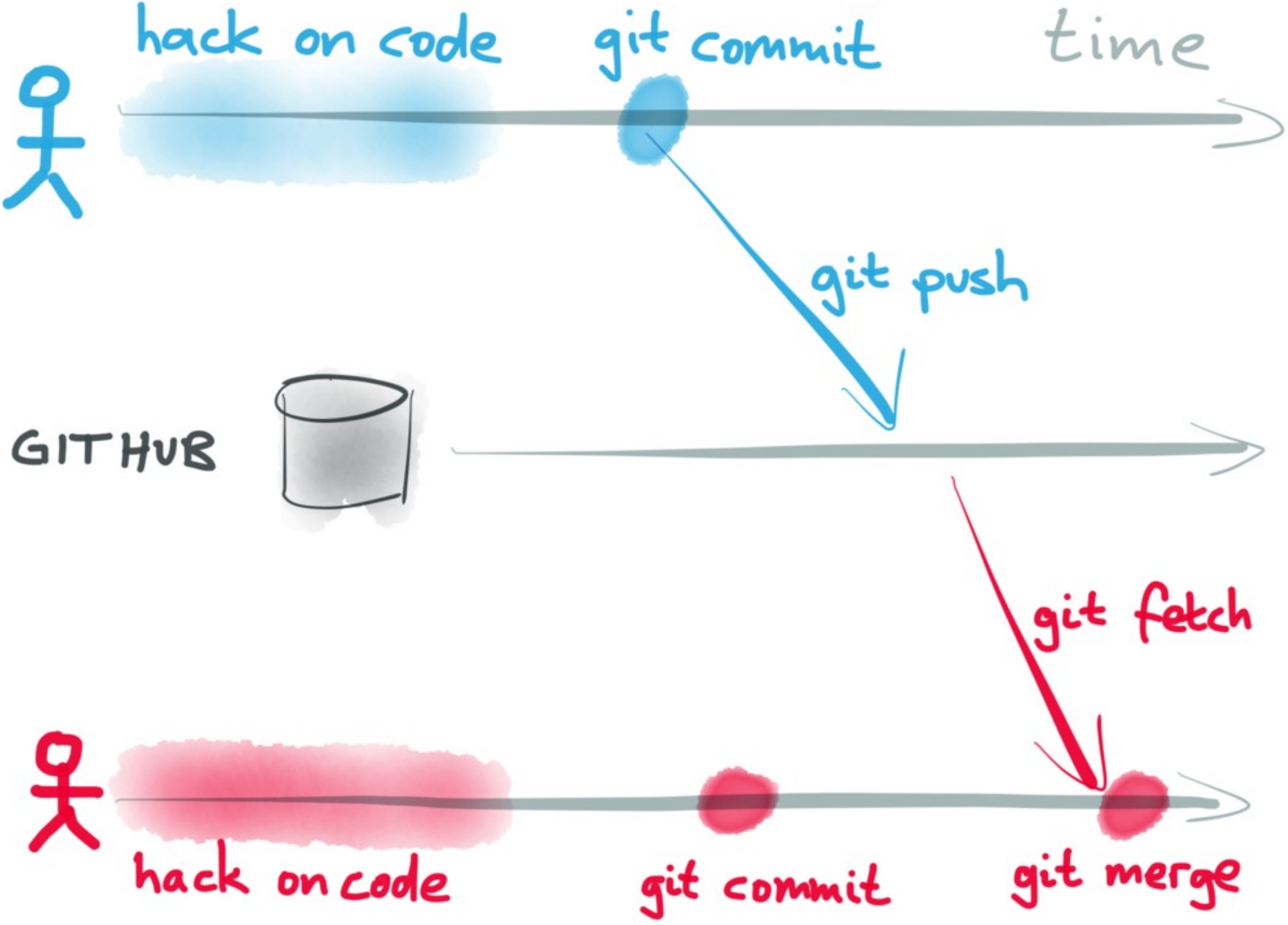


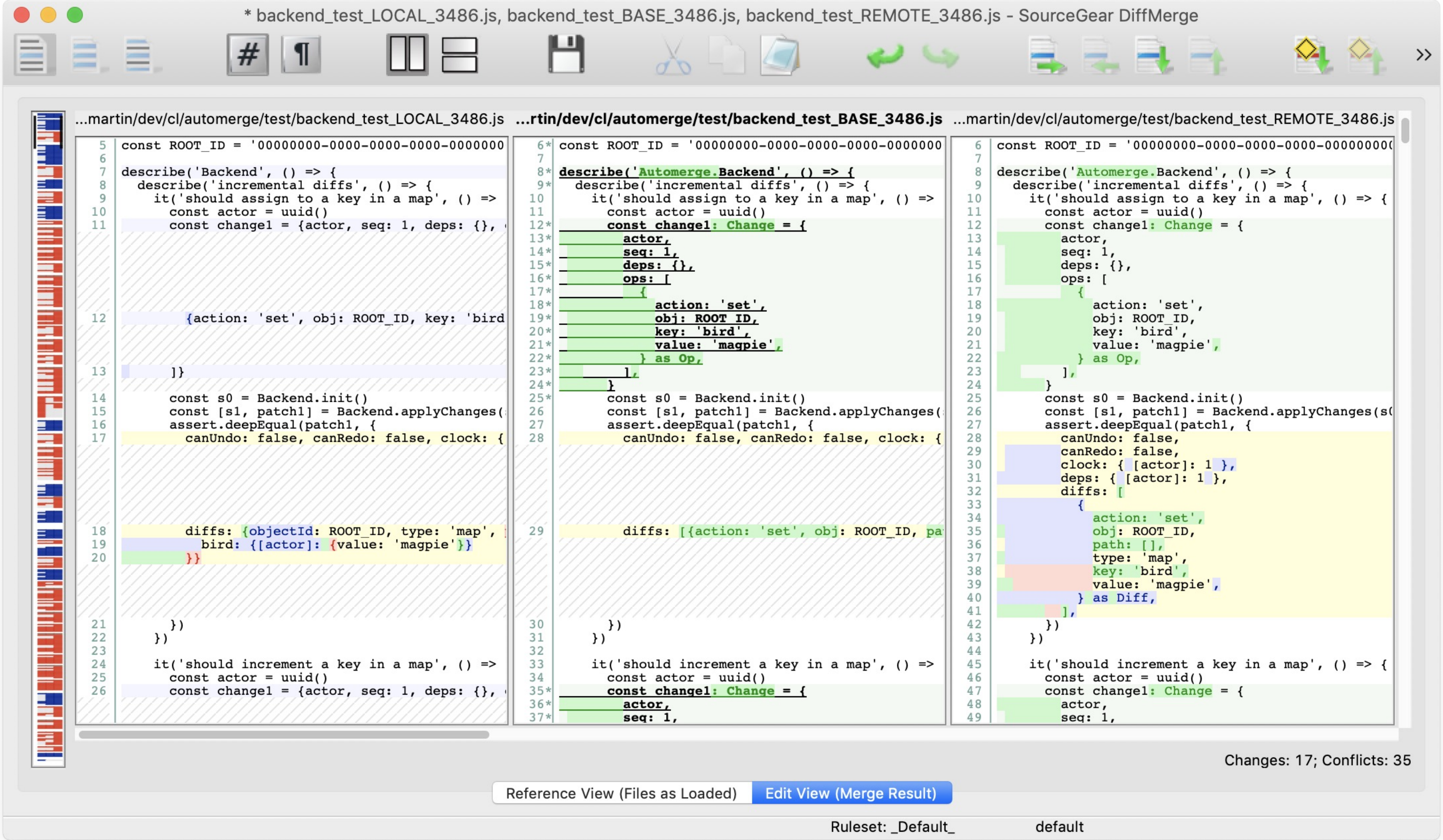
hack on code
time

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

```
const ROOT_ID = '00000000-0000-0000-0000-0000000

describe('Backend', () => {
  describe('incremental diffs', () => {
    it('should assign to a key in a map', () =>
      const actor = uuid()
      const change1 = {actor, seq: 1, deps: {},
        {action: 'set', obj: ROOT_ID, key: 'bird
      ]}

      const s0 = Backend.init()
      const [s1, patch1] = Backend.applyChanges(
      assert.deepEqual(patch1, {
        canUndo: false, canRedo: false, clock: {
        diffs: {objectId: ROOT_ID, type: 'map',
          bird: {[actor]: {value: 'magpie'}}
        }}
      })
    })

    it('should increment a key in a map', () =>
      const actor = uuid()
      const change1 = {actor, seq: 1, deps: {},
```

...rtin/dev/cl/automerge/test/backend_test_BASE_3486.js

```
const ROOT_ID = '00000000-0000-0000-0000-0000000

describe('Automerge.Backend', () => {
  describe('incremental diffs', () => {
    it('should assign to a key in a map', () =>
      const actor = uuid()
      const change1: Change = {
        actor,
        seq: 1,
        deps: {},
        ops: [
          {
            action: 'set',
            obj: ROOT_ID,
            key: 'bird',
            value: 'magpie',
          } as Op,
        ],
      }
      const s0 = Backend.init()
      const [s1, patch1] = Backend.applyChanges(
      assert.deepEqual(patch1, {
        canUndo: false, canRedo: false, clock: {
        diffs: [{action: 'set', obj: ROOT_ID, pa
      })
    })

    it('should increment a key in a map', () =>
      const actor = uuid()
      const change1: Change = {
        actor,
        seq: 1,
```

...martin/dev/cl/automerge/test/backend_test_REMOTE_3486.js

```
const ROOT_ID = '00000000-0000-0000-0000-00000000(

describe('Automerge.Backend', () => {
  describe('incremental diffs', () => {
    it('should assign to a key in a map', () => {
      const actor = uuid()
      const change1: Change = {
        actor,
        seq: 1,
        deps: {},
        ops: [
          {
            action: 'set',
            obj: ROOT_ID,
            key: 'bird',
            value: 'magpie',
          } as Op,
        ],
      }
      const s0 = Backend.init()
      const [s1, patch1] = Backend.applyChanges(s(
      assert.deepEqual(patch1, {
        canUndo: false,
        canRedo: false,
        clock: { [actor]: 1 },
        deps: { [actor]: 1 },
        diffs: [
          {
            action: 'set',
            obj: ROOT_ID,
            path: [],
            type: 'map',
            key: 'bird',
            value: 'magpie',
          } as Diff,
        ],
      })
    })

    it('should increment a key in a map', () => {
      const actor = uuid()
      const change1: Change = {
        actor,
        seq: 1,
```

Changes: 17; Conflicts: 35

Ruleset: _Default_ default

COLLABORATIVE APPLICATIONS

Google Docs

Google Sheets

Office 365

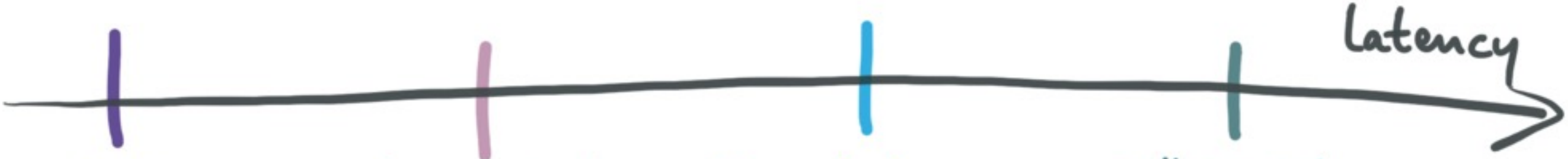
Overleaf

Trello

Figma

Collaboration

Syncing changes between several users



real-time (keystroke by keystroke)

fine-grained (update on clicking ok)

suggest changes, accept/reject

pull requests: asynchronous collaboration

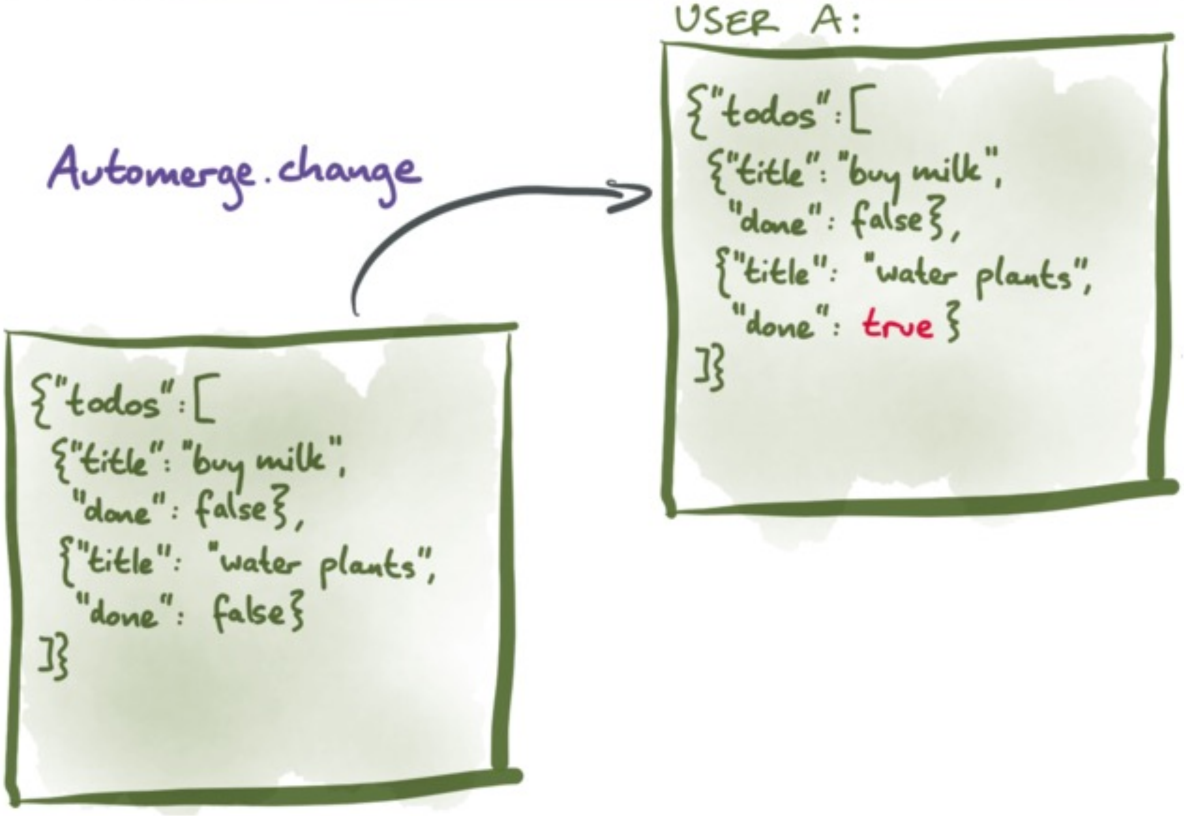
many (100,000s) small changes

few large changes

AUTOMERGE: Branching and merging

```
{"todos":[
 {"title":"buy milk",
  "done": false},
 {"title": "water plants",
  "done": false}
]}
```

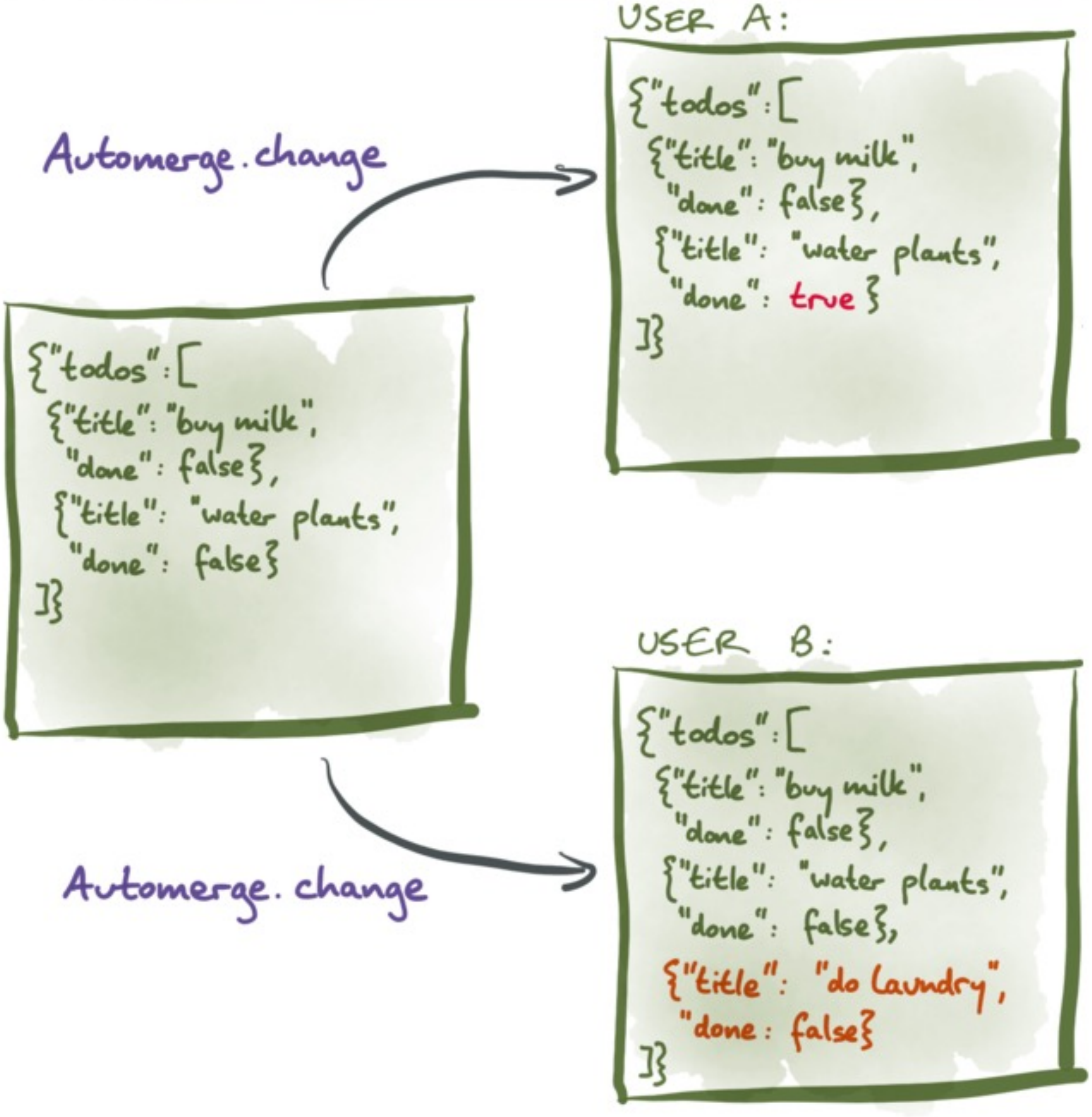
AUTOMERGE: Branching and merging



```
{"todos":[
 {"title":"buy milk",
  "done": false},
 {"title": "water plants",
  "done": false}
]}
```

```
{"todos":[
 {"title":"buy milk",
  "done": false},
 {"title": "water plants",
  "done": true}
]}
```

AUTOMERGE: Branching and merging



```
{"todos":[
 {"title":"buy milk",
  "done": false},
 {"title": "water plants",
  "done": false}
]}
```

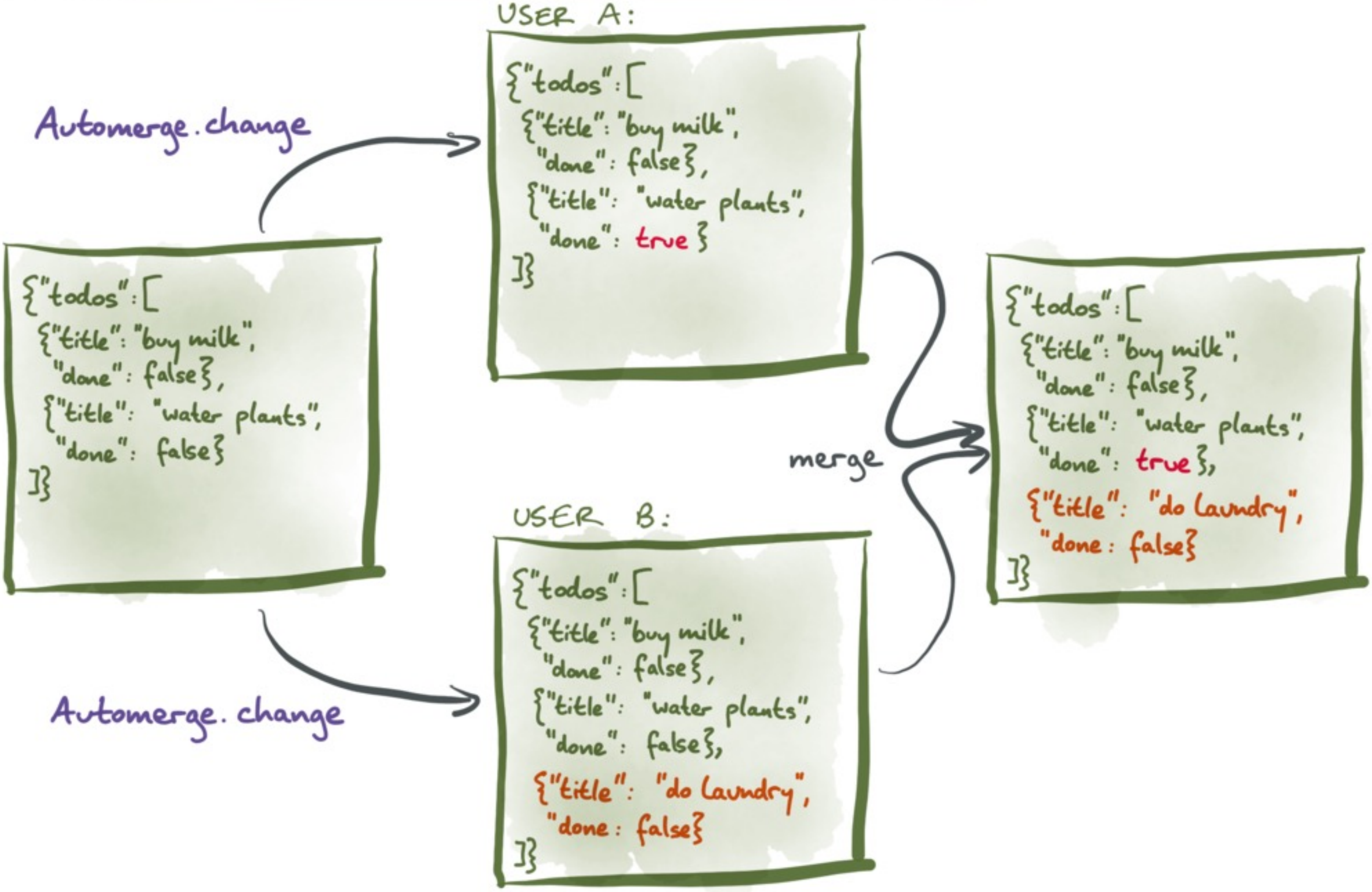
USER A:

```
{"todos":[
 {"title":"buy milk",
  "done": false},
 {"title": "water plants",
  "done": true}
]}
```

USER B:

```
{"todos":[
 {"title":"buy milk",
  "done": false},
 {"title": "water plants",
  "done": false},
 {"title": "do laundry",
  "done : false}
]}
```

AUTOMERGE: Branching and merging



```
{"todos":[
 {"title":"buy milk",
  "done": false},
 {"title": "water plants",
  "done": false}
]}
```

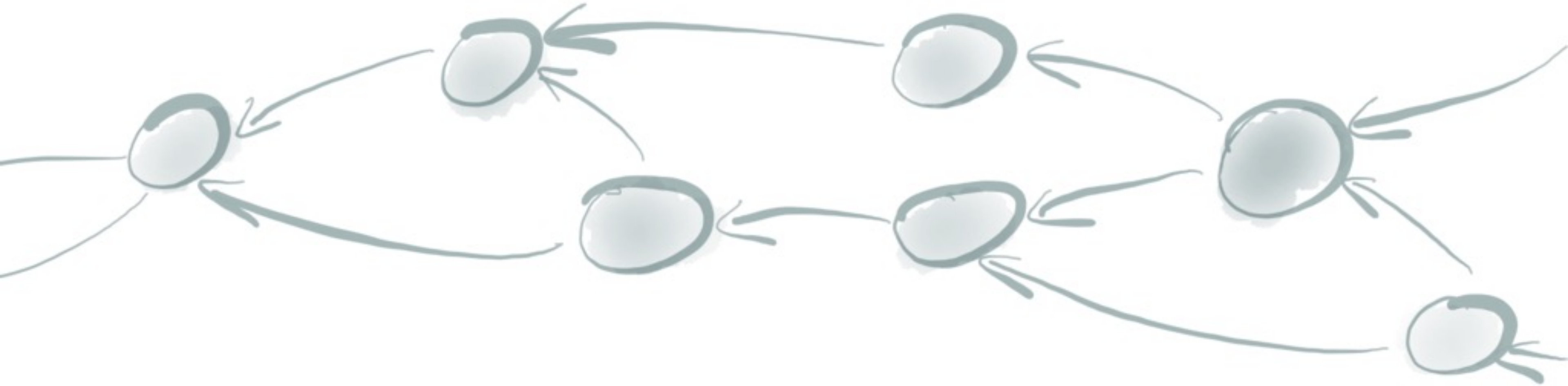
USER A:

```
{"todos":[
 {"title":"buy milk",
  "done": false},
 {"title": "water plants",
  "done": true}
]}
```

USER B:

```
{"todos":[
 {"title":"buy milk",
  "done": false},
 {"title": "water plants",
  "done": false},
 {"title": "do laundry",
  "done : false}
]}
```

```
{"todos":[
 {"title":"buy milk",
  "done": false},
 {"title": "water plants",
  "done": true},
 {"title": "do laundry",
  "done : false}
]}
```



Automerge

https://github.com/automerge/automerge

AUTOMERGE: "Git for your app's data"

```
{"todos": [
    {"title": "buy milk", "done": false},
    {"title": "water plants", "done": false}
]}
```

AUTOMERGE: "Git for your app's data"

```
{"todos": [
    {"title": "buy milk", "done": false},
    {"title": "water plants", "done": false}
]}
```

```
after = Automerge.change(before, "add new item", doc => {
        doc.todos.push({title: "do laundry", done: false});
});
```

AUTOMERGE: "Git for your app's data"

```
{"todos": [
    {"title": "buy milk", "done": false},
    {"title": "water plants", "done": false},
    {"title": "do laundry", "done": false}    ← added
]}
```

```
after = Automerge.change(before, "add new item", doc => {
      doc.todos.push({title: "do laundry", done: false});
});
```

AUTOMERGE: "Git for your app's data"

```
{"todos": [
    {"title": "buy milk", "done": false},
    {"title": "water plants", "done": false},
    {"title": "do laundry", "done": false}
]}
```

added

```
after = Automerge.change(before, "add new item", doc => {
    doc.todos.push({title: "do laundry", done: false});
});
```

append item to list

```
Automerge.change(state, doc => {
    doc.todos.push({title: "do laundry", done: false});
});
```

```
Automerge.change(state, doc => {
    doc.todos.push({title: "do laundry", done: false});
});
```

operation log

```
{op: "makeMap", id: "5a", obj: "1a", elemID: "2a", insert: true}
{op: "assign", id: "6a", obj: "5a", key: "title", value: "do laundry"}
{op: "assign", id: "7a", obj: "5a", key: "done", value: false}
```

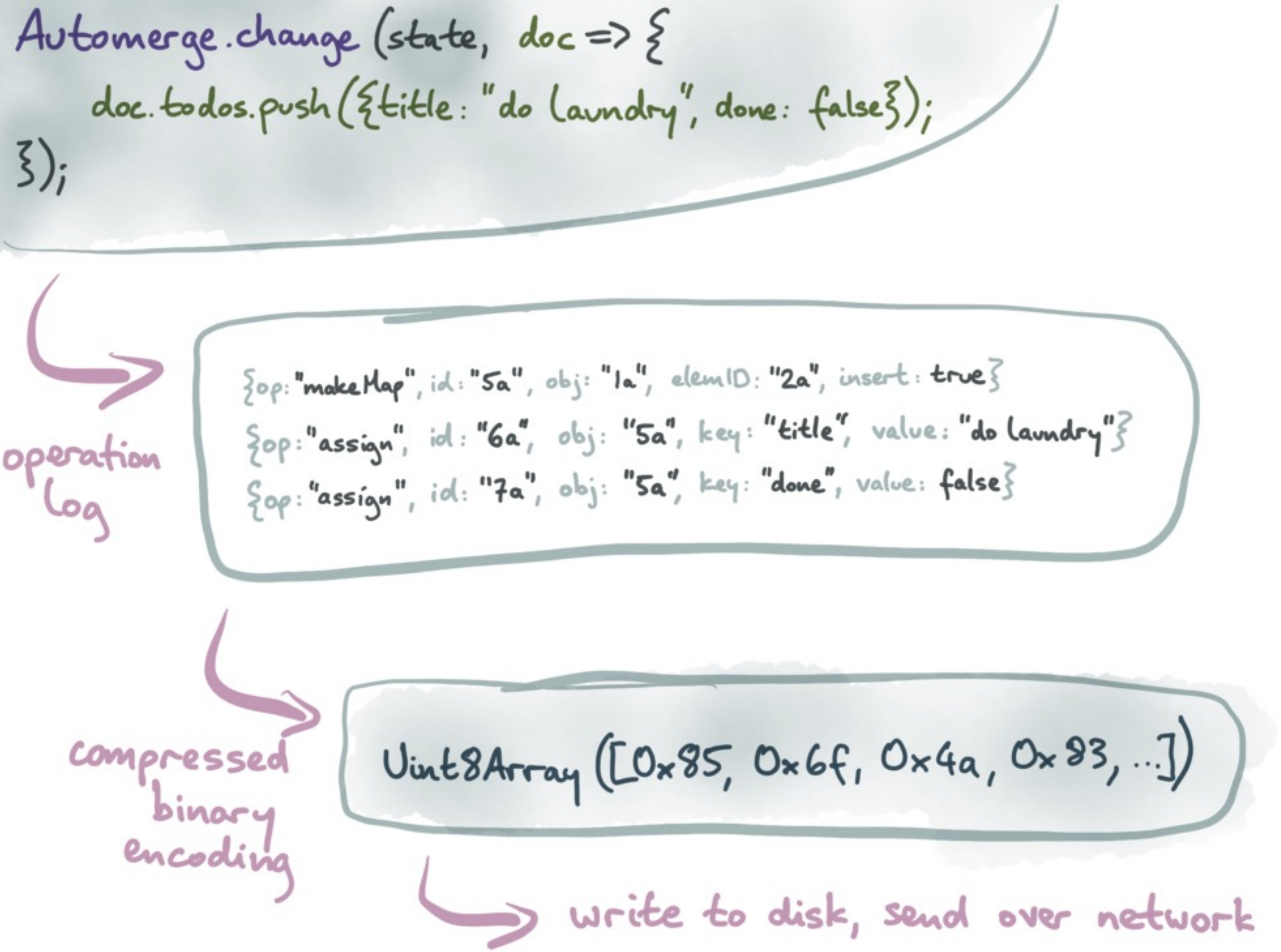
```
Automerge.change(state, doc => {
    doc.todos.push({title: "do laundry", done: false});
});
```

operation log

```
{op: "makeMap", id: "5a", obj: "1a", elemID: "2a", insert: true}
{op: "assign", id: "6a", obj: "5a", key: "title", value: "do laundry"}
{op: "assign", id: "7a", obj: "5a", key: "done", value: false}
```

compressed binary encoding

```
Uint8Array([0x85, 0x6f, 0x4a, 0x83, ...])
```



```
Automerge.change(state, doc => {
    doc.todos.push({title: "do laundry", done: false});
});
```

operation log

```
{op: "makeMap", id: "5a", obj: "1a", elemID: "2a", insert: true}
{op: "assign", id: "6a", obj: "5a", key: "title", value: "do laundry"}
{op: "assign", id: "7a", obj: "5a", key: "done", value: false}
```

compressed binary encoding

```
Uint8Array([0x85, 0x6f, 0x4a, 0x83, ...])
```

write to disk, send over network

{title: "Water plants", done: false}

```
{op: "makeMap", id: "1a"}
{op: "assign", id: "2a", obj: "1a", key: "title", value: "Water plants", overwrites: []}
{op: "assign", id: "3a", obj: "1a", key: "done", value: false, overwrites: []}
```

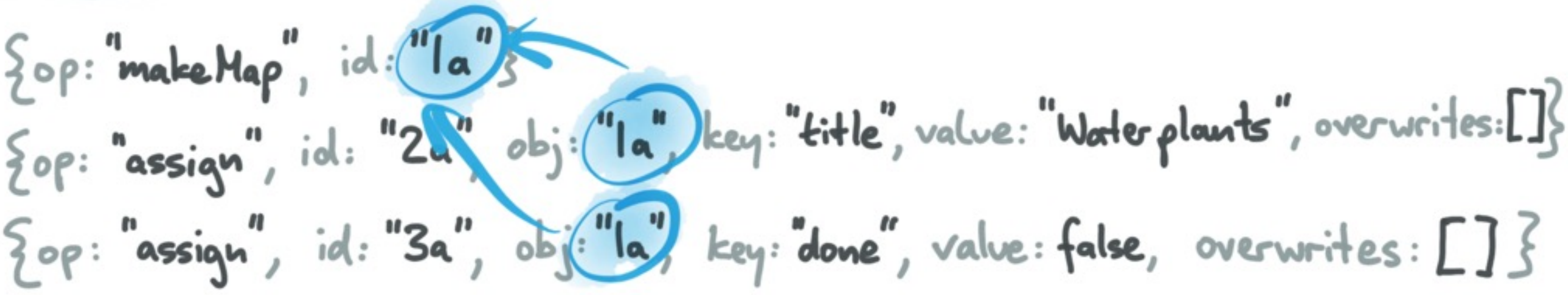
```
{title: "Water plants", done: false}
```

```
{op: "makeMap", id: "1a"}
{op: "assign", id: "2a", obj: "1a", key: "title", value: "Water plants", overwrites: []}
{op: "assign", id: "3a", obj: "1a", key: "done", value: false, overwrites: []}
```

```
doc.todos[0].done = true
```

```
{op: "assign", id: "4a", obj: "1a", key: "done", value: true, overwrites: ["3a"]}
```

```
{title: "Water plants", done: false}
```



```
{op: "makeMap", id: "1a"}
{op: "assign", id: "2a", obj: "1a", key: "title", value: "Water plants", overwrites: []}
{op: "assign", id: "3a", obj: "1a", key: "done", value: false, overwrites: []}
```

```
doc.todos[0].done = true
```

```
{op: "assign", id: "4a", obj: "1a", key: "done", value: true, overwrites: ["3a"]}
```

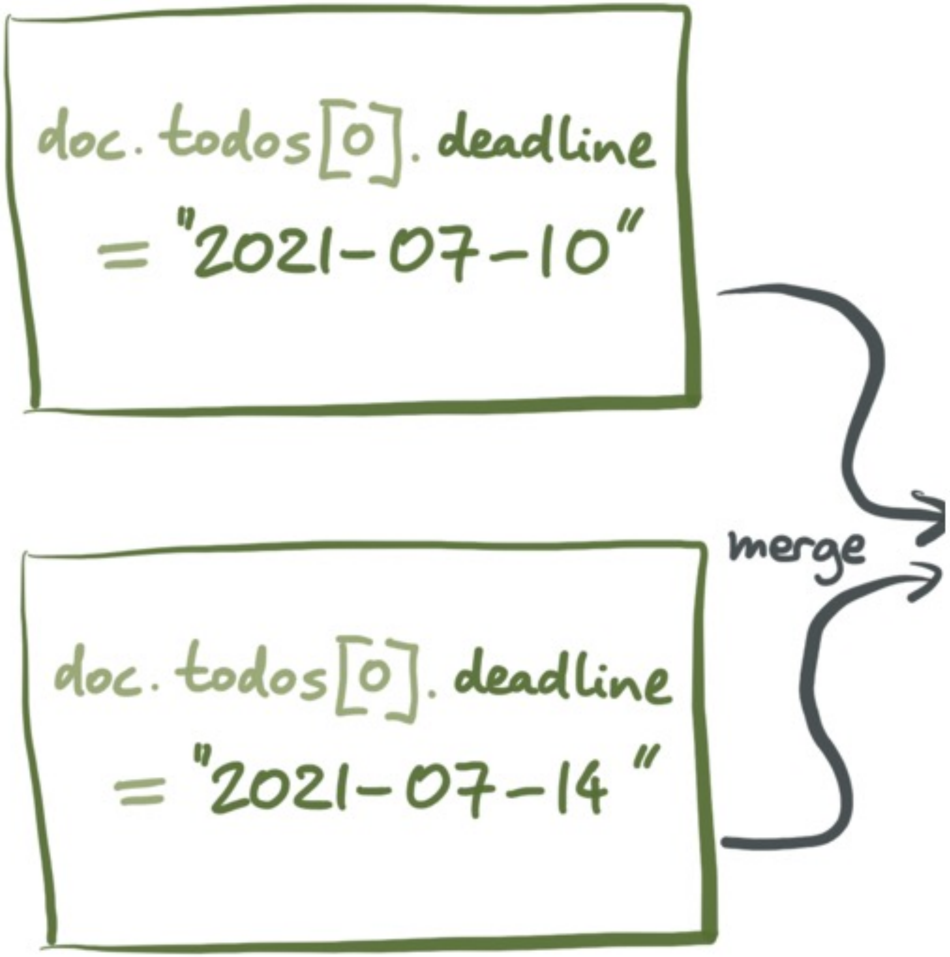
```
{title: "Water plants", done: false}
```

```
{op: "makeMap", id: "1a"}
{op: "assign", id: "2a", obj: "1a", key: "title", value: "Water plants", overwrites: []}
{op: "assign", id: "3a", obj: "1a", key: "done", value: false, overwrites: []}
```

```
doc.todos[0].done = true
```

```
{op: "assign", id: "4a", obj: "1a", key: "done", value: true, overwrites: ["3a"]}
```

MANUAL CONFLICT RESOLUTION



```
doc.todos[0].deadline
  = "2021-07-10"
```

```
doc.todos[0].deadline
  = "2021-07-14"
```

```
doc.todos[0].deadline = "2021-07-10"
```

```
{op: "assign", id: "5a", obj: "1a", key: "deadline", value: "2021-07-10",
                                                       overwrites: []}
```

```
doc.todos[0].deadline = "2021-07-14"
```

```
{op: "assign", id: "5b", obj: "1a", key: "deadline", value: "2021-07-14",
                                                       overwrites: []}
```

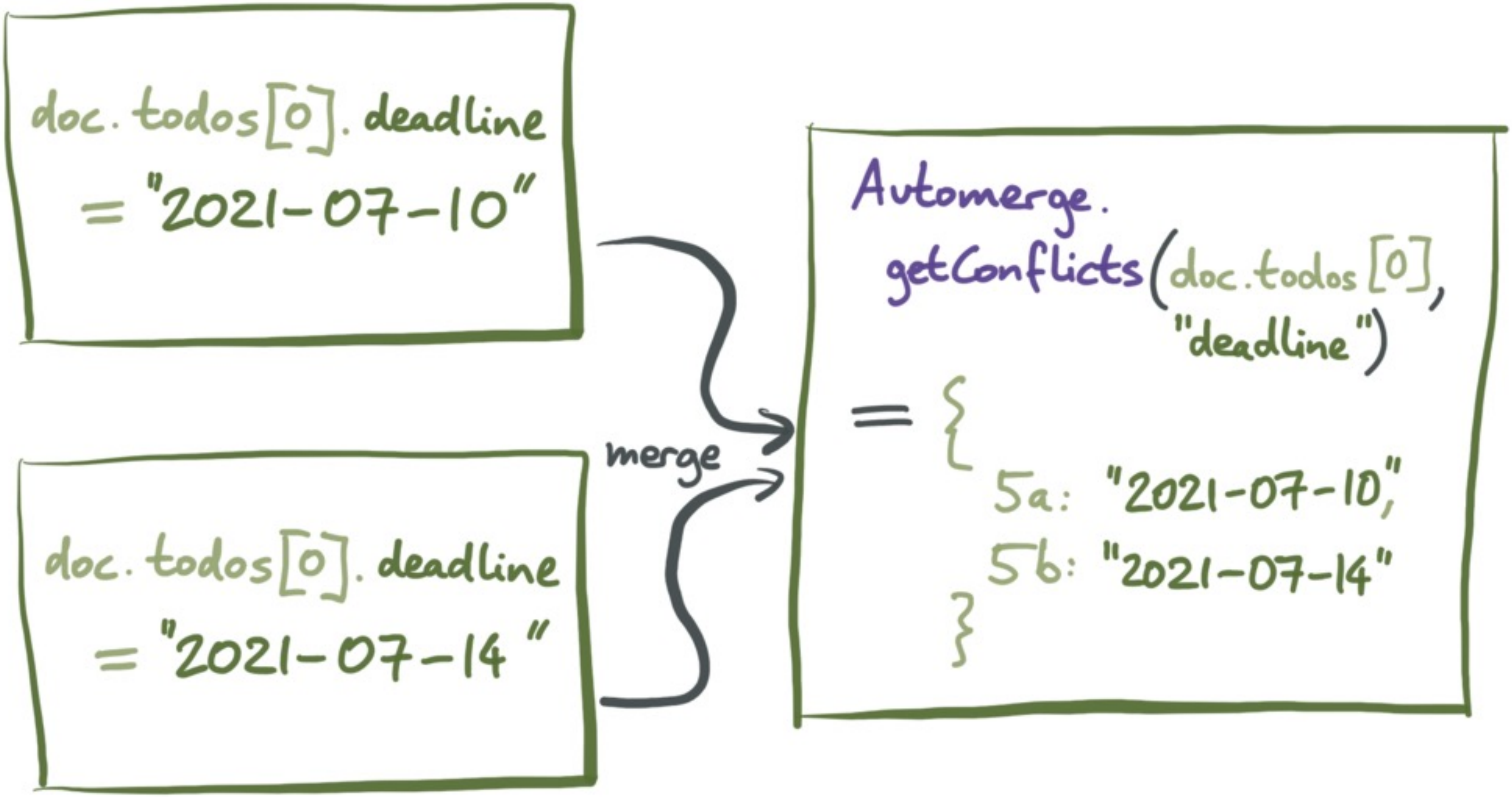
```
doc.todos[0].deadline = "2021-07-10"
```

```
{op: "assign", id: "5a", obj: "1a", key: "deadline", value: "2021-07-10",
                                                     overwrites: []}
```

```
doc.todos[0].deadline = "2021-07-14"
```

```
{op: "assign", id: "5b", obj: "1a", key: "deadline", value: "2021-07-14",
                                                     overwrites: []}
```

MANUAL CONFLICT RESOLUTION



```
doc.todos[0].deadline
  = "2021-07-10"
```

```
doc.todos[0].deadline
  = "2021-07-14 "
```

```
Automerge.
  getConflicts(doc.todos[0],
               "deadline")
= {
    5a: "2021-07-10",
    5b: "2021-07-14"
  }
```

TIME TRAVEL

Automerge.getHistory(state)

TIME TRAVEL

```
Automerge.getHistory(state)

⇒ [ {change: {message: "Add todo item", ...},
       snapshot: {todos: [{title: "Buy milk", ...}, ...]}},

    {change: {message: "Mark item as done", ...},
       snapshot: {todos: [{title: "Buy milk", ...}, ...]}},
    ...
  ]
```

```
{op: "makeMap", id: "1a"}
{op: "assign", id: "2a", obj: "1a", key: "title", value: "Water plants", overwrites: []}
{op: "assign", id: "3a", obj: "1a", key: "done", value: false, overwrites: []}
{op: "assign", id: "4a", obj: "1a", key: "done", value: true, overwrites: ["3a"]}
```

```
{op: "makeMap", id: "1a"}
{op: "assign", id: "2a", obj: "1a", key: "title", value: "Water plants", overwrites: []}
{op: "assign", id: "3a", obj: "1a", key: "done", value: false, overwrites: []}
{op: "assign", id: "4a", obj: "1a", key: "done", value: true, overwrites: ["3a"]}
```

convert into table of ops

op	id	obj	key	value	overwritten by
makeMap	1a	root	todo		{}
assign	3a	1a	done	false	{4a}
assign	4a	1a	done	true	{}
assign	2a	1a	title	"Water plants"	{}

Sort order: 1. by object ID → 2. if map: lexicographic by key → 3. by operation ID

1. by object ID → 2. if list: order of list elements → 3. by operation ID

op	id	obj	key	value	overwritten by
makeMap	1a	root	todo		{}
assign	3a	1a	done	false	{4a}
assign	4a	1a	done	true	{}
assign	2a	1a	title	"Water plants"	{}

Identify document version by version vector

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

op	id	obj	key	value	overwritten by
makeMap	1a	root	todo		$\{\}$
assign	3a	1a	done	false	$\{4a\}$
assign	4a	1a	done	true	$\{\}$
assign	2a	1a	title	"Water plants"	$\{\}$

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

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

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!

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)

COLUMNAR ENCODING (simplified)

operation ID		reference element ID		inserted character		deleted by op ID	
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 op ID	
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 op ID	
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 op ID	
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)

COLUMNAR ENCODING (simplified)

operation ID		reference element ID		inserted character		deleted by op ID	
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"	—	—

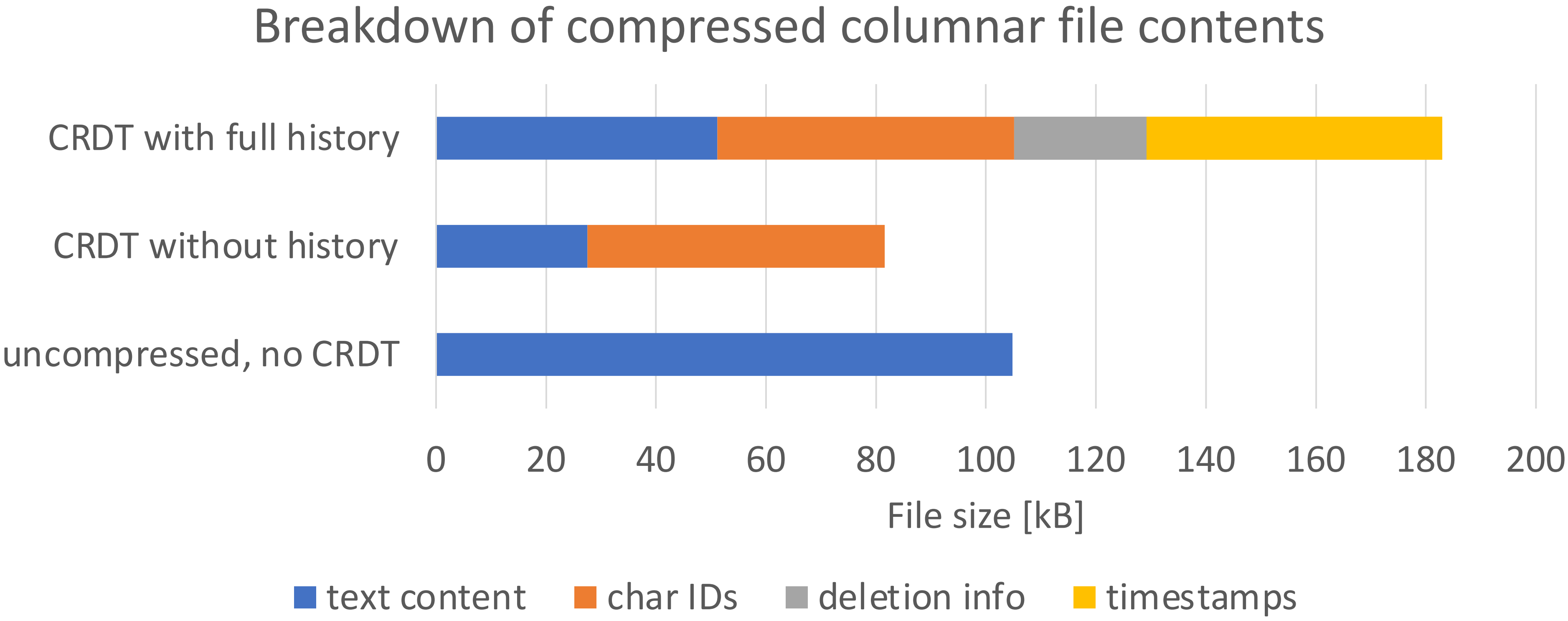
Plus some additional metadata (e.g. timestamp and range of opID counter values for each change)

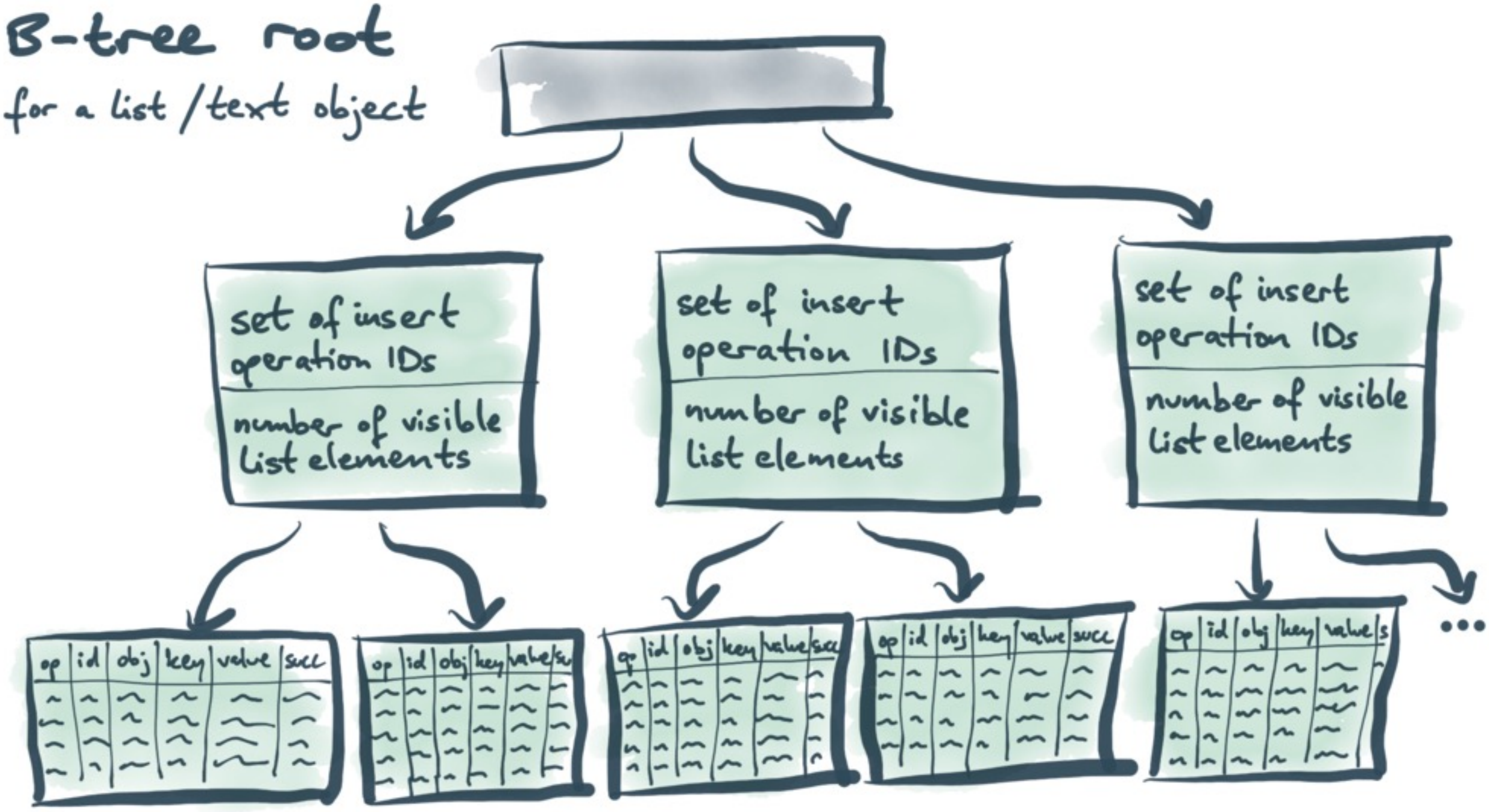
⟹ can reconstruct any past document state

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)

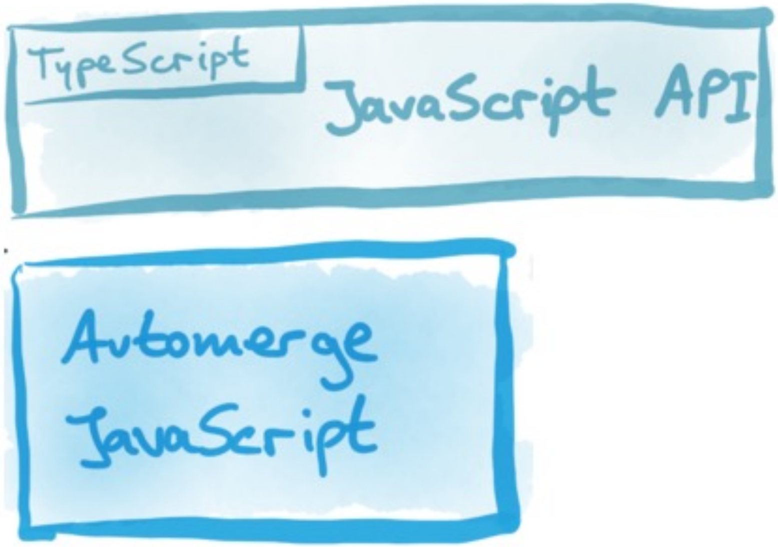



B-tree root
for a list / text object
set of insert operation IDs
number of visible list elements
set of insert operation IDs
number of visible list elements
set of insert operation IDs
number of visible list elements
op | id | obj | key | value | succ
op | id | obj | key | value | su
op | id | obj | key | value | succ
op | id | obj | key | value | succ
op | id | obj | key | value | s
...

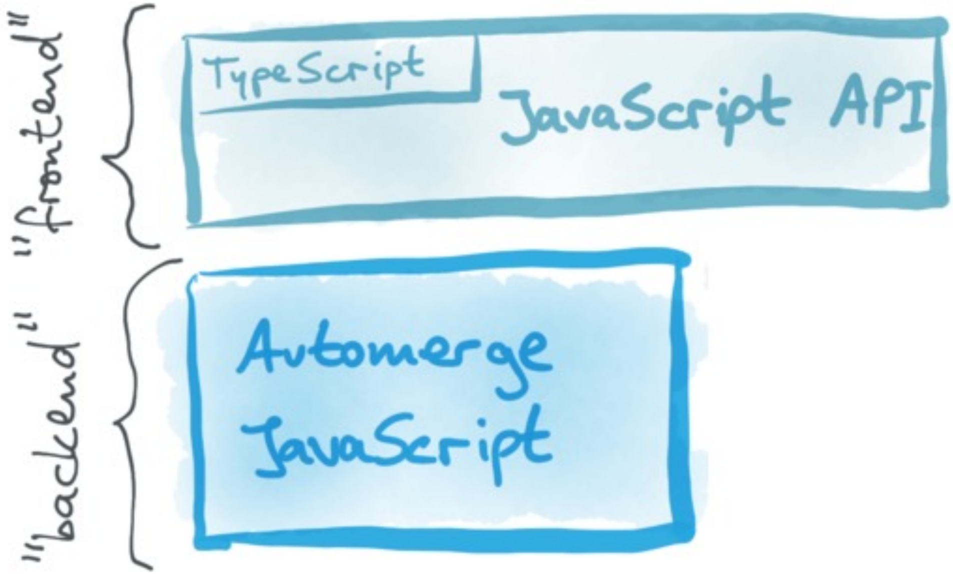
Local insert: find insertion position by index, counting only visible elements,
to translate index into opID of last insert at that index

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

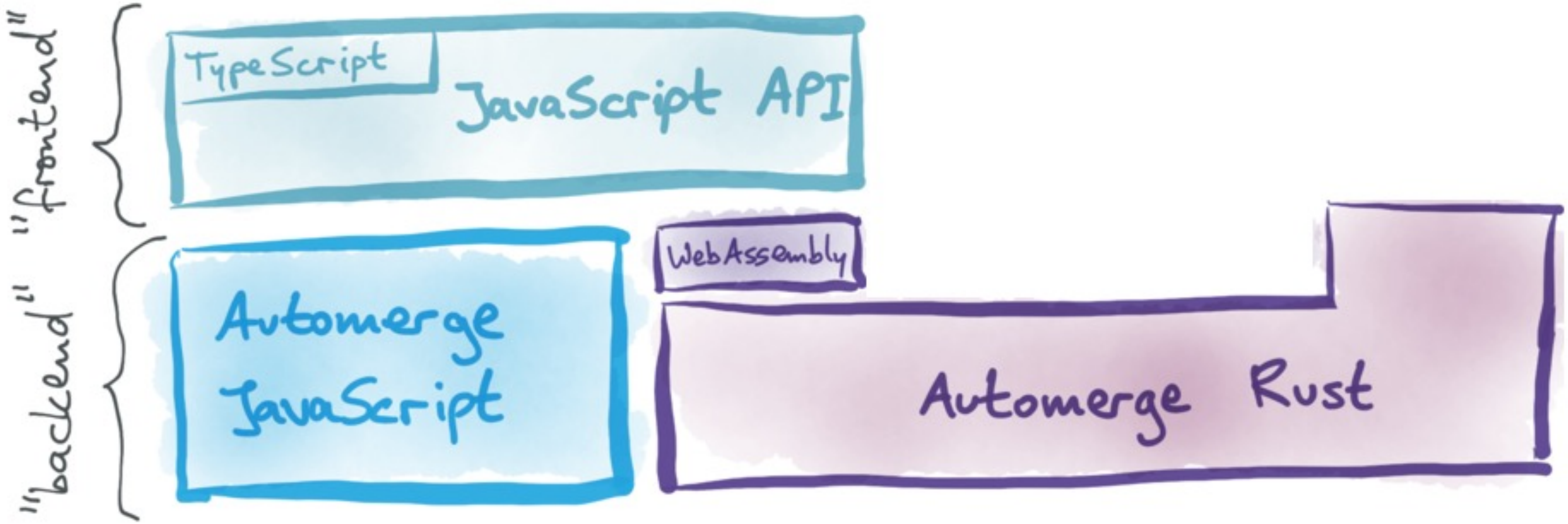
IMPLEMENTATION



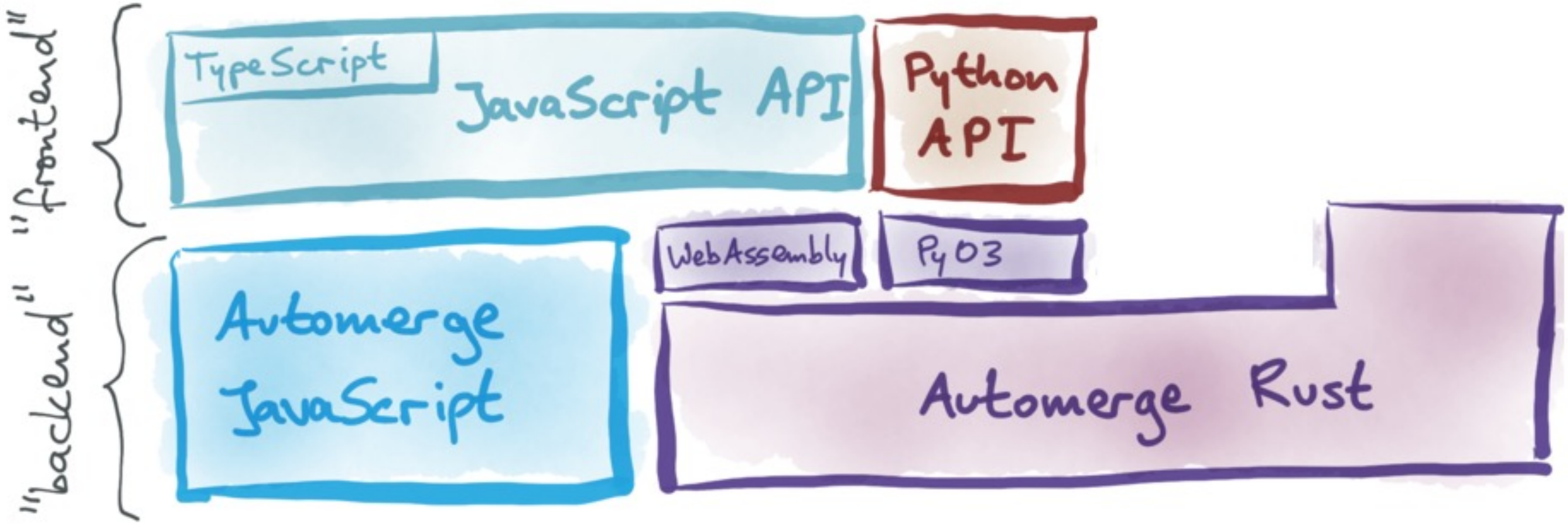
IMPLEMENTATION



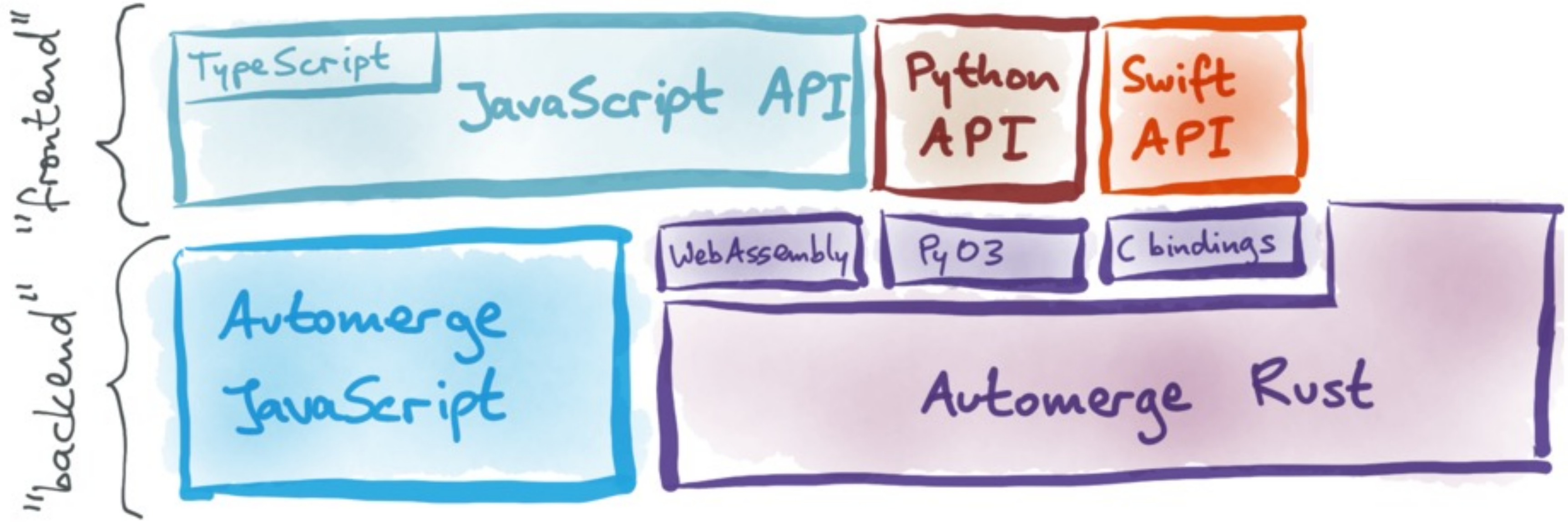
IMPLEMENTATION



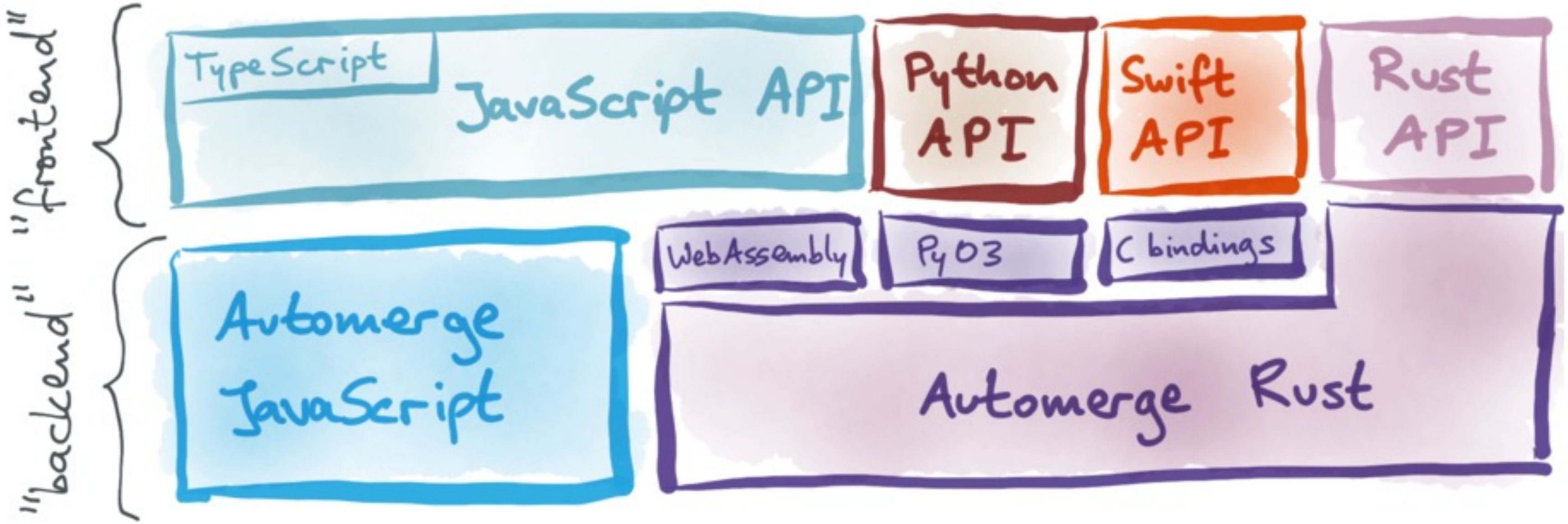
IMPLEMENTATION



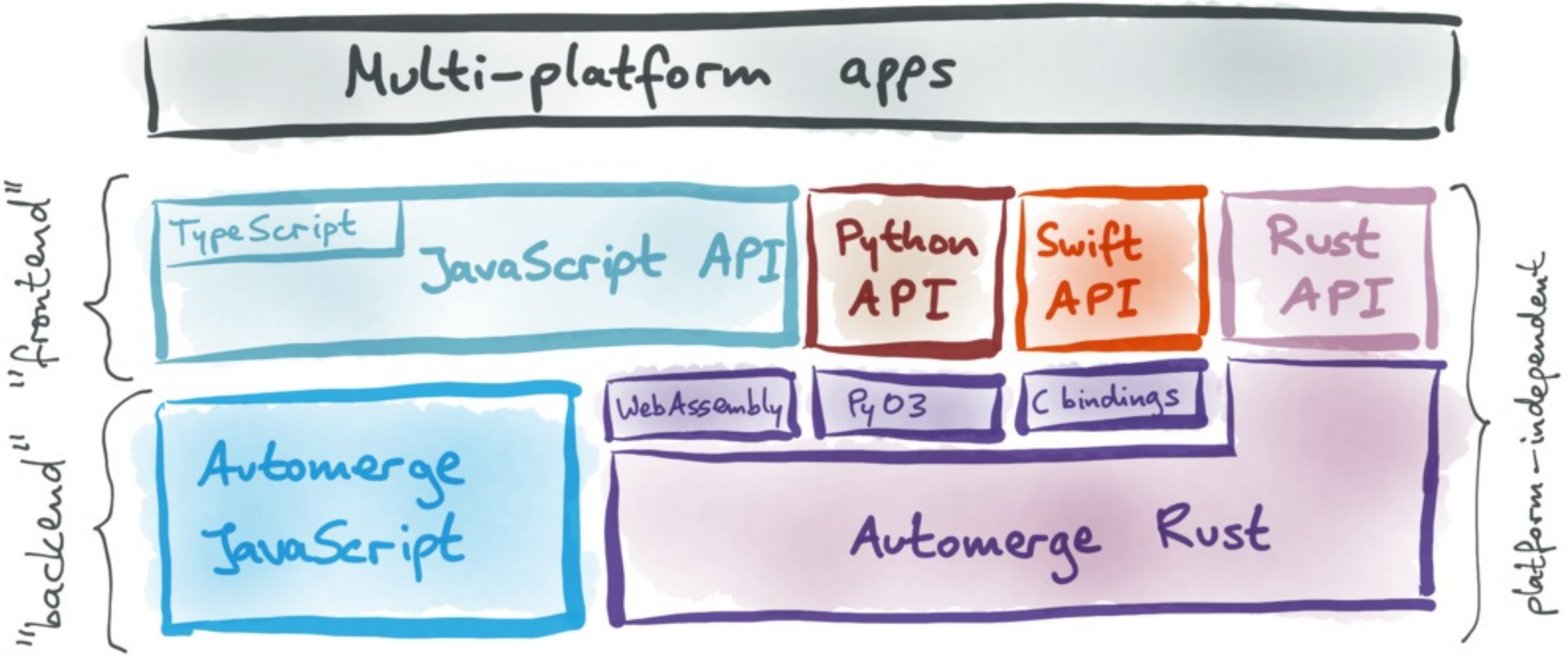
IMPLEMENTATION



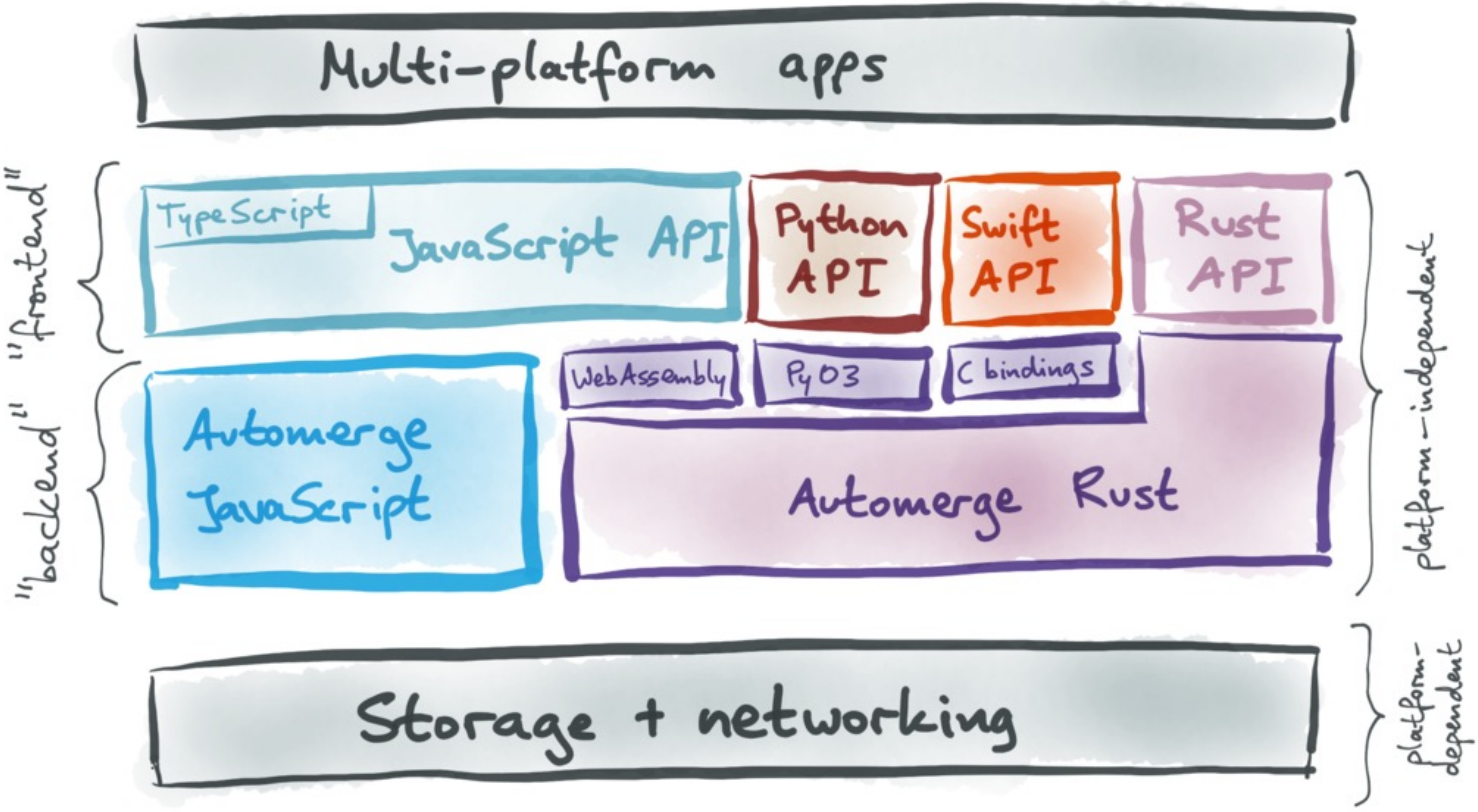
IMPLEMENTATION



IMPLEMENTATION



IMPLEMENTATION



Resources

Automerge	https://automerge.org/
Publications	https://martin.kleppmann.com/#publications
Email	martin@kleppmann.com
Twitter	@martinkl
Book	http://dataintensive.net/
Support me	https://www.patreon.com/martinkl

Huge thanks to my collaborators and the Automerge community and contributors, especially Peter van Hardenberg, Orion Henry, Alex Good, Andrew Jeffery, Rae McKelvey, Herb Caudill, & others!

LEVERHULME TRUST
Isaac Newton Trust
Ink & Switch