Automerge:
A new foundation for collaboration software

MARTIN KLEPPMANN  @martinkl
UNIVERSITY OF CAMBRIDGE

Ink & Switch
Thank you to my supporters

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

- "Hello!"
- "Hello!"

Time
Example: Text editing

- Insert "World" after "Hello"

"Hello !"  "Hello World !"

- Insert ":-)" after "!"

"Hello !"  "Hello! :-)"
Example: Text editing

```
"Hello!"  "Hello World!"  "Hello World! :-)"

insert "World" after "Hello"
insert ":-)" after "!"
```

Time arrow pointing right.
hack on code
time
hack on code   git commit   time
GitHub

- hack on code
- git commit
- git push
- git fetch
- git merge

Time

- hack on code
- git commit

GitHub
```javascript
const ROOT_ID = '00000000-0000-0000-0000-000000000000

describe('Backend', () => {
  describe('incremental diffs', () => {
    it('should assign to a key in a map', () => {
      const actor = uid();
      const changel = {actor, seq: 1, deps: {}},
        
      {action: 'set', obj: ROOT_ID, key: 'bird',
        value: 'maagpie'}
    }
      
      const s0 = Backend.init();
      const [s1, patch1] = Backend.applyChanges;
      assert.deepEqual(patch1, { canUndo: false, canRedo: false, clock: {
        }
      }
      
      diffs: {objectID: ROOT_ID, type: 'map',
        bird: [{key: 'bird',
          value: 'maagpie' }]
    }
      } })
      it('should increment a key in a map', () => {
      const actor = uid();
      const changel = {actor, seq: 1, deps: {}},
        
      {action: 'set', obj: ROOT_ID, key: 'bird',
        value: 'maagpie'}
    }
      
      const s0 = Backend.init();
      const [s1, patch1] = Backend.applyChanges;
      assert.deepEqual(patch1, { canUndo: false, canRedo: false, clock: {
        }
      }
      
      diffs: [{action: 'set', obj: ROOT_ID, pa
        actor: 1}],
      }
      
      it('should increment a key in a map', () => {
      const actor = uid();
      const changel = {actor, seq: 1, deps: {}},
        
      {action: 'set', obj: ROOT_ID, key: 'bird',
        value: 'maagpie'}
    }
      
      const s0 = Backend.init();
      const [s1, patch1] = Backend.applyChanges;
      assert.deepEqual(patch1, { canUndo: false, canRedo: false, clock: {
        }
      }
      
      diffs: [{action: 'set', obj: ROOT_ID, pa
        actor: 1}],
      }
      
      it('should increment a key in a map', () => {
      const actor = uid();
      const changel = {actor, seq: 1, deps: {}},
        
      {action: 'set', obj: ROOT_ID, key: 'bird',
        value: 'maagpie'}
    }
      
      const s0 = Backend.init();
      const [s1, patch1] = Backend.applyChanges;
      assert.deepEqual(patch1, { canUndo: false, canRedo: false, clock: {
        }
      }
      
      diffs: [{action: 'set', obj: ROOT_ID, pa
        actor: 1}],
      }
      
      it('should increment a key in a map', () => {
      const actor = uid();
      const changel = {actor, seq: 1, deps: {}},
        
      {action: 'set', obj: ROOT_ID, key: 'bird',
        value: 'maagpie'}
    }
      
      const s0 = Backend.init();
      const [s1, patch1] = Backend.applyChanges;
      assert.deepEqual(patch1, { canUndo: false, canRedo: false, clock: {
        }
      }
      
      diffs: [{action: 'set', obj: ROOT_ID, pa
        actor: 1}],
      }
      
      it('should increment a key in a map', () => {
      const actor = uid();
      const changel = {actor, seq: 1, deps: {}},
        
      {action: 'set', obj: ROOT_ID, key: 'bird',
        value: 'maagpie'}
    }
      
      const s0 = Backend.init();
      const [s1, patch1] = Backend.applyChanges;
      assert.deepEqual(patch1, { canUndo: false, canRedo: false, clock: {
        }
      }
      
      diffs: [{action: 'set', obj: ROOT_ID, pa
        actor: 1}],
      }
      
      it('should increment a key in a map', () => {
      const actor = uid();
      const changel = {actor, seq: 1, deps: {}},
        
      {action: 'set', obj: ROOT_ID, key: 'bird',
        value: 'maagpie'}
    }
      
      const s0 = Backend.init();
      const [s1, patch1] = Backend.applyChanges;
      assert.deepEqual(patch1, { canUndo: false, canRedo: false, clock: {
        }
      }
      
      diffs: [{action: 'set', obj: ROOT_ID, pa
        actor: 1}],
      }
      
      it('should increment a key in a map', () => {
      const actor = uid();
      const changel = {actor, seq: 1, deps: {}},
        
      {action: 'set', obj: ROOT_ID, key: 'bird',
        value: 'maagpie'}
    }
      
      const s0 = Backend.init();
      const [s1, patch1] = Backend.applyChanges;
      assert.deepEqual(patch1, { canUndo: false, canRedo: false, clock: {
        }
      }
      
      diffs: [{action: 'set', obj: ROOT_ID, pa
        actor: 1}],
      }
```
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

```json
{
  "todos": [
    {
      "title": "buy milk",
      "done": false
    },
    {
      "title": "water plants",
      "done": false
    }
  ]
}
```
AUTOMERGE: Branching and merging

User A:

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

Automerger change:

```json
{ "todos": [
    { "title": "buy milk", "done": false },
    { "title": "water plants", "done": false }
]}
```
AUTOMERGE: Branching and merging

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}]
}
```

Automerger change

Automerger change
AUTOMERGE: Branching and merging

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

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"

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

let after = Automerge.change(before, "add new item", doc => {
  doc.todos.push({
    title: "do laundry", done: false
  });
});
```
AUTOMERGE: "Git for your app's data"

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

```javascript
after = Automerge.change(before, "add new item", doc => {
  doc.todos.push({
    title: "do laundry", done: false
  });
});
```
AUTOMERGE: “Git for your app’s data”

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

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}
```javascript
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, ...])`
Automerger: change (state, doc => { doc.todos.push({title: "do laundry", done: false}); });
{title: "Water plants", done: false}

{op: "makeMap", id: "1a"}


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

{op: "makeMap", id: "1a"}
{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: "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: "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

\[ \text{doc. todos[0]. deadline} = "2021-07-10" \]

\[ \text{doc. todos[0]. deadline} = "2021-07-14" \]

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

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

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

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

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

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

MANUAL CONFLICT RESOLUTION

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

merge

getConflicts(doc.todos[0], "deadline")

= {
  5a: "2021-07-10",
  5b: "2021-07-14"
}
TIME TRAVEL

Automerge.getHistory(state)
```javascript
const automerge = require('automerge');

const history = automerge.getHistory(state);

history.map(change => {
  console.log(change.message);
  return {
    snapshot: state.todos.map(todo => {
      if (todo.title === change.message) {
        return { done: true };
      }
      return todo;
    })
  }
})
```

{op: "makeMap", id: "1a"}
{op: "assign", id: "3a", obj: "1a", key: "done", value: false, overwrites: []}
{op: "assign", id: "4a", obj: "1a", key: "done", value: true, overwrites: ["3a"]}
<table>
<thead>
<tr>
<th>op</th>
<th>id</th>
<th>obj</th>
<th>key</th>
<th>value</th>
<th>overwritten by</th>
</tr>
</thead>
<tbody>
<tr>
<td>makeMap</td>
<td>la</td>
<td>root</td>
<td>todo</td>
<td></td>
<td>{3a}</td>
</tr>
<tr>
<td>assign</td>
<td>3a</td>
<td>la</td>
<td>done</td>
<td>false</td>
<td>{4a}</td>
</tr>
<tr>
<td>assign</td>
<td>4a</td>
<td>la</td>
<td>done</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>assign</td>
<td>2a</td>
<td>la</td>
<td>title</td>
<td>&quot;Water plants&quot;</td>
<td>{3a}</td>
</tr>
</tbody>
</table>

Sort order: 1. by object ID, 2. if map: lexicographic by key, 3. if list: order of list elements, 3. by operation ID
<table>
<thead>
<tr>
<th>op</th>
<th>id</th>
<th>obj</th>
<th>key</th>
<th>value</th>
<th>overwritten by</th>
</tr>
</thead>
<tbody>
<tr>
<td>makeMap</td>
<td>1a</td>
<td>root</td>
<td>todo</td>
<td></td>
<td>33</td>
</tr>
<tr>
<td>assign</td>
<td>3a</td>
<td>1a</td>
<td>done</td>
<td>false</td>
<td>4a3</td>
</tr>
<tr>
<td>assign</td>
<td>4a</td>
<td>1a</td>
<td>done</td>
<td>true</td>
<td>33</td>
</tr>
<tr>
<td>assign</td>
<td>2a</td>
<td>1a</td>
<td>title</td>
<td>&quot;Water plants&quot;</td>
<td>33</td>
</tr>
</tbody>
</table>

Identify document version by version vector
e.g. $V = \{a: 3, b: 4\}$
<table>
<thead>
<tr>
<th>op</th>
<th>id</th>
<th>obj</th>
<th>key</th>
<th>value</th>
<th>overwritten by</th>
</tr>
</thead>
<tbody>
<tr>
<td>makeMap</td>
<td>1a</td>
<td>root</td>
<td>todo</td>
<td></td>
<td>≡{3}</td>
</tr>
<tr>
<td>assign</td>
<td>3a</td>
<td>1a</td>
<td>done</td>
<td>false</td>
<td>≡{4a}</td>
</tr>
<tr>
<td>assign</td>
<td>4a</td>
<td>1a</td>
<td>done</td>
<td>true</td>
<td>≡{3}</td>
</tr>
<tr>
<td>assign</td>
<td>2a</td>
<td>1a</td>
<td>title</td>
<td>&quot;Water plants&quot;</td>
<td>≡{3}</td>
</tr>
</tbody>
</table>

Identify document version by version vector

\[V = \{a: 3, b: 4\}\]

**Visibility rule:** operation with ID = \((\text{ctr}_{10}, \text{node}_{10})\) and overwrittenBy = \{\((\text{ctr}_i, \text{node}_i), (\text{ctr}_z, \text{node}_z), \ldots\}\} is visible at document version \(V\) iff \(\text{ctr}_{10} \leq V[\text{node}_{10}]\) and

\(\forall(\text{ctr}_i, \text{node}_i) \in \text{overwrittenBy}. \text{ctr}_i \leq V[\text{node}_i].\)
Identify document version by version vector
e.g. \( V = \{ a: 3, b: 4 \} \)

Visibility rule: operation with ID = \( (\text{ctr}_{10}, \text{node}_{10}) \) and overwrittenBy = \( \{ (\text{ctr}_i, \text{node}_i), (\text{ctr}_2, \text{node}_2), \ldots \} \) is visible at document version \( V \) iff \( \text{ctr}_{10} \leq V[\text{node}_{10}] \) and

\( \forall (\text{ctr}_i, \text{node}_i) \in \text{overwrittenBy}. \ \text{ctr}_i \leq V[\text{node}_i]. \)

Like MVCC in databases with snapshot isolation!
Identify document version by version vector

\[ V = \{a: 3, b: 4\} \]

Visibility rule: operation with \( ID = (ctr_{10}, node_{10}) \) and 
overwrittenBy = \( \{ (ctr_1, node_1), (ctr_2, node_2), \ldots \} \) is visible at document
version \( V \) iff \( ctr_{10} \leq V[node_{10}] \) and

\( \forall (ctr_i, node_i) \in \text{overwrittenBy}. \ ctr_i \leq V[node_i]. \)

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)

<table>
<thead>
<tr>
<th>Operation ID</th>
<th>Operation</th>
<th>Reference Element ID</th>
<th>Inserted Character</th>
<th>Deleted by Op ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>counter</td>
<td>counter</td>
<td>length</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>-</td>
<td>1</td>
<td>&quot;H&quot;</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>&quot;e&quot;</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>&quot;l&quot;</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>3</td>
<td>1</td>
<td>&quot;l&quot;</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>4</td>
<td>1</td>
<td>&quot;l&quot;</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>5</td>
<td>1</td>
<td>&quot;o&quot;</td>
</tr>
</tbody>
</table>

Note: The table shows the operation ID, counter, actor, and the inserted character for each row. The deleted by op ID column indicates if any characters were deleted.
**Columnar Encoding (simplified)**

<table>
<thead>
<tr>
<th>Operation ID</th>
<th>Reference Element ID</th>
<th>Inserted Character</th>
<th>Deleted by OpID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>counter</td>
<td>actor</td>
<td>length</td>
<td>UTF-8</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>1</td>
<td>&quot;H&quot;</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>1</td>
<td>&quot;e&quot;</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>1</td>
<td>&quot;l&quot;</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>1</td>
<td>&quot;l&quot;</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>1</td>
<td>&quot;l&quot;</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>1</td>
<td>&quot;o&quot;</td>
</tr>
</tbody>
</table>

1, 2, 3, 4, 5, 6<br>
delta-encode to 1, 1, 1, 1, 1, 1<br>run-length encode to (6, 1)<br>LEB128 encodes this in 2 bytes
## Columnar Encoding (simplified)

<table>
<thead>
<tr>
<th>Operation ID</th>
<th>Reference Element ID</th>
<th>Inserted Character</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Counter</td>
<td>Actor</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>A</td>
</tr>
</tbody>
</table>

- Make a lookup table: \{"A": 0, "B": 13\}
- \(0, 0, 0, 0, 0, 0, 0\)
- Run-length encode to \((6, 0)\)
- LEB128 encodes in 2 bytes
## Columnar Encoding (simplified)

<table>
<thead>
<tr>
<th>Operation ID</th>
<th>Reference Element ID</th>
<th>Inserted Character</th>
<th>Deleted by Op ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>counter</td>
<td>actor</td>
<td>counter</td>
</tr>
<tr>
<td>1</td>
<td>counter</td>
<td>actor</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>counter</td>
<td>actor</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>counter</td>
<td>actor</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>counter</td>
<td>actor</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>counter</td>
<td>actor</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>counter</td>
<td>actor</td>
<td>5</td>
</tr>
</tbody>
</table>

Just concatenate the UTF-8 byte sequences → "Hello" (6 bytes)

(use length column to separate again)
<table>
<thead>
<tr>
<th>operation ID</th>
<th>reference element ID</th>
<th>inserted character</th>
<th>deleted by opID</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter</td>
<td>actor</td>
<td>length</td>
<td>UTF-8</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>-</td>
<td>&quot;H&quot;</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>1</td>
<td>&quot;e&quot;</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>1</td>
<td>&quot;l&quot;</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>1</td>
<td>&quot;l&quot;</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>1</td>
<td>&quot;l&quot;</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>1</td>
<td>&quot;o&quot;</td>
</tr>
</tbody>
</table>

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)

Breakdown of compressed columnar file contents

- **CRDT with full history**
- **CRDT without history**
- **uncompressed, no CRDT**

## File size [kB]

- **text content**
- **char IDs**
- **deletion info**
- **timestamps**
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

TypeScript
JavaScript API

Automerge
JavaScript
IMPLEMENTATION

```
"backend"

TypeScript
JavaScript API

"frontend"

Automerger
JavaScript
```
IMPLEMENTATION

"backend"

- TypeScript
- JavaScript
- Automerge

"frontend"

- JavaScript API
- WebAssembly
- Automerge
- Rust
IMPLEMENTATION

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
IMPLEMENTATION

"frontend"
- TypeScript
- JavaScript API
- Automerge JavaScript

"backend"
- Python API
- Swift API
- WebAssembly
- Py03
- C bindings
- Automerge Rust
IMPLEMENTATION

"frontend"
TypeScript
JavaScript API
Python API
Swift API
Rust API

"backend"
Automerge JavaScript
WebAssembly
Py03
C bindings
Automerge Rust

IMPLEMENTATION

Multi-platform apps

```
<table>
<thead>
<tr>
<th>Frontend</th>
<th>Backend</th>
</tr>
</thead>
<tbody>
<tr>
<td>TypeScript</td>
<td>Automerge JavaScript</td>
</tr>
<tr>
<td>JavaScript API</td>
<td>WebAssembly</td>
</tr>
<tr>
<td>Python API</td>
<td>PyO3</td>
</tr>
<tr>
<td>Swift API</td>
<td>C bindings</td>
</tr>
<tr>
<td>Rust API</td>
<td>Automerge Rust</td>
</tr>
</tbody>
</table>
```
IMPLEMENTATION

Multi-platform apps

TypeScript  JavaScript API  Python API  Swift API  Rust API

Automerger  JavaScript

WebAssembly  PyO3  C bindings

Automerger  Rust

Storage + networking
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!