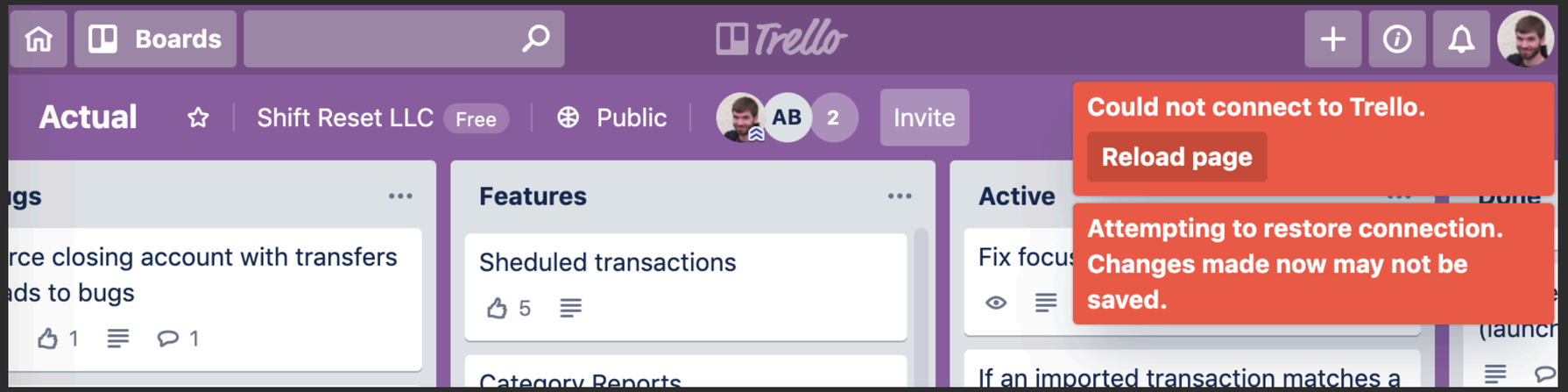
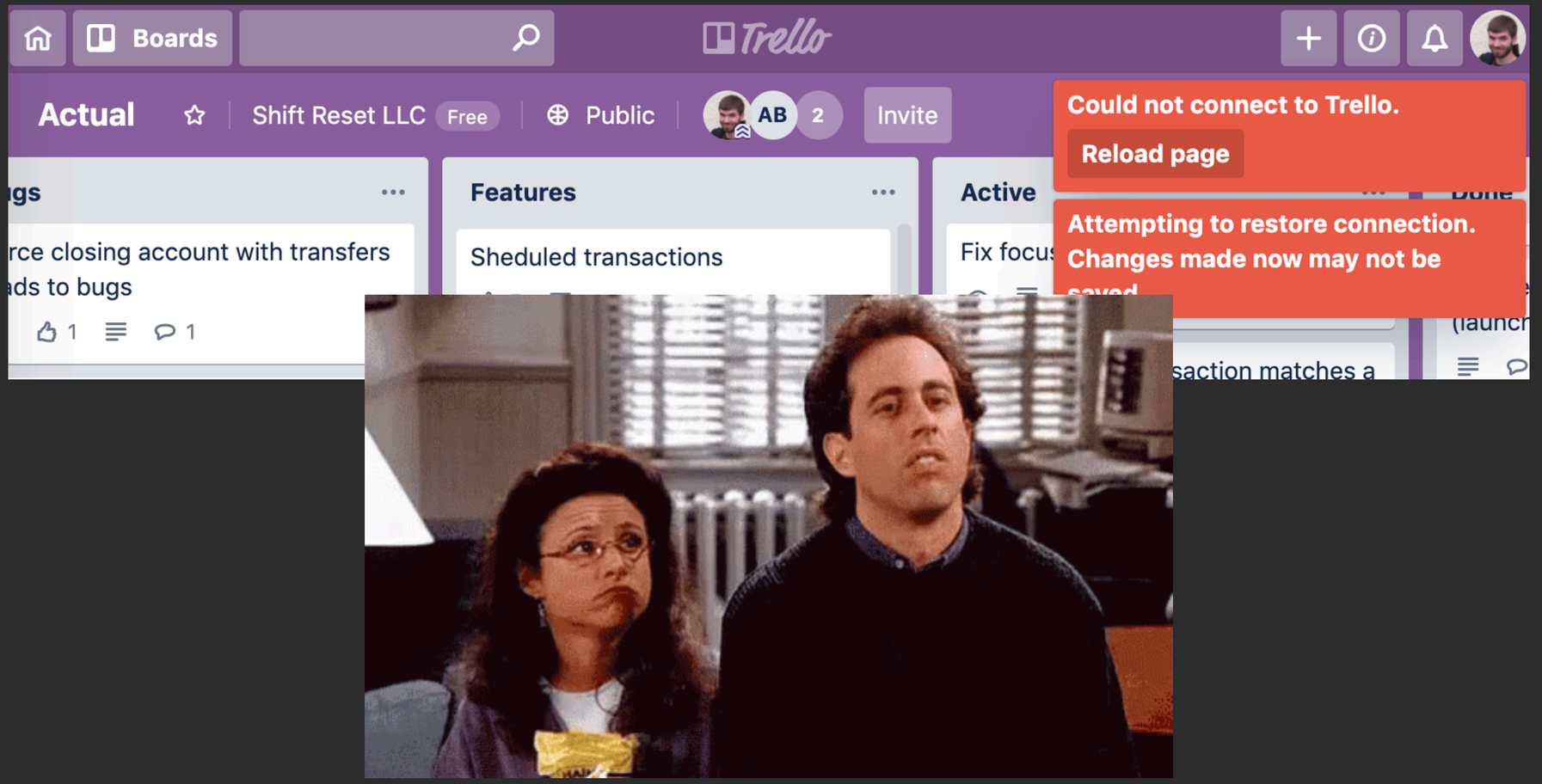
CRDTs for Mortals

Why haven’t “offline-first” apps taken off?

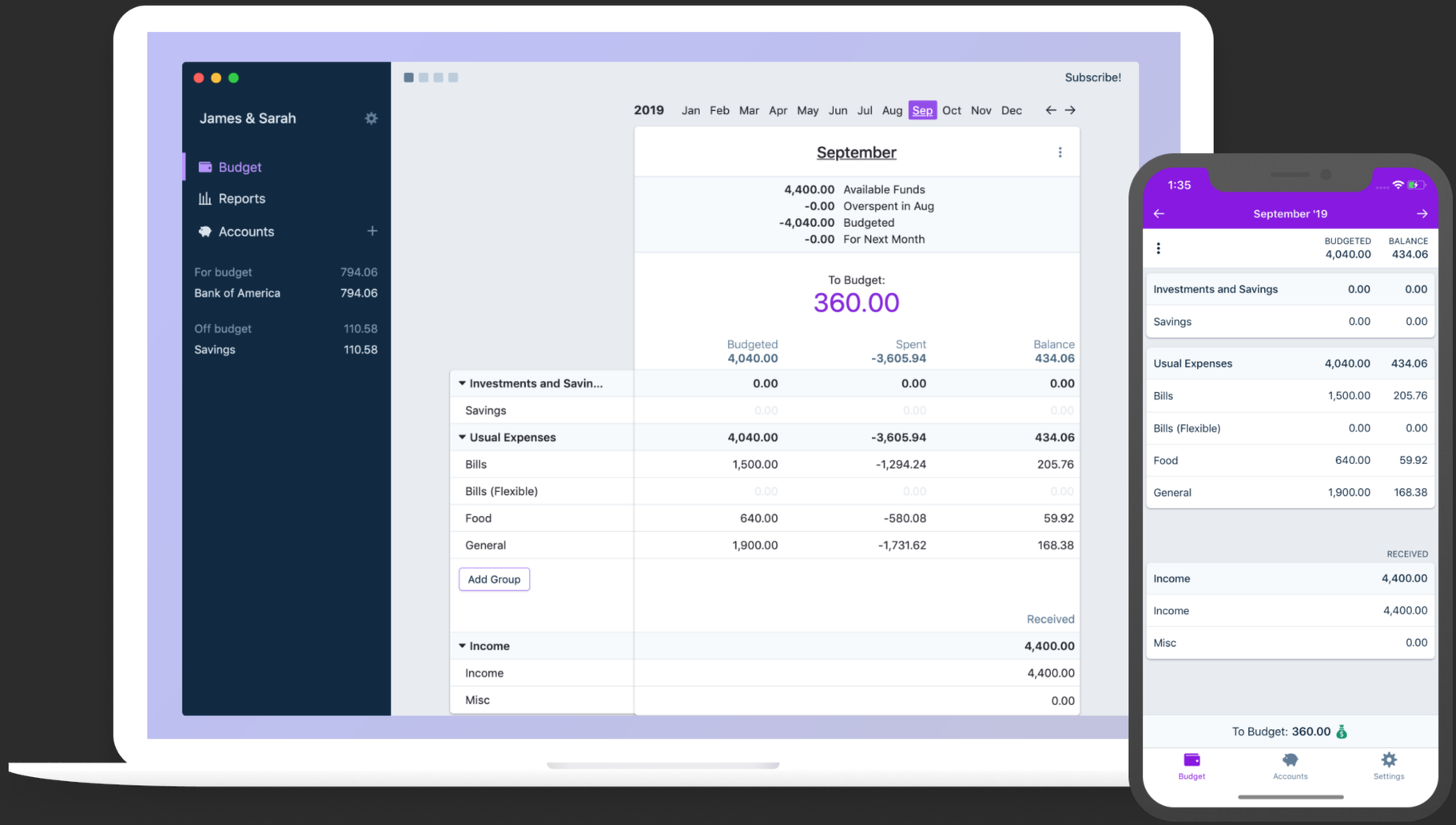
Syncing is hard


Boards
Trello
Actual
Shift Reset LLC
Free
Public
AB
2
Invite
Could not connect to Trello.
Reload page
Attempting to restore connection. Changes made now may not be saved.
Features
Sheduled transactions
5
Category Reports
Active
Fix focus
If an imported transaction matches a
1
1


Boards
Trello
Actual
Shift Reset LLC
Free
Public
AB
2
Invite
Could not connect to Trello.
Reload page
Attempting to restore connection.
Changes made now may not be
Features
Sheduled transactions
Active
Fix focus
rce closing account with transfers
ds to bugs
1
1
saction matches a

You must admit that local apps are a distributed system


Edit profile
James Long
@jlongster
Finally launched @actualbudget! Created @PrettierCode. Ex-Mozilla.
Chattanooga, TN
jlongster.com
Born June 20, 1984
Joined May 2008



Actual — https://actualbudget.com

available offline

must be fast

focus on privacy

arbitrary queries

... a local app!

My Budget

Budget

Reports

Accounts

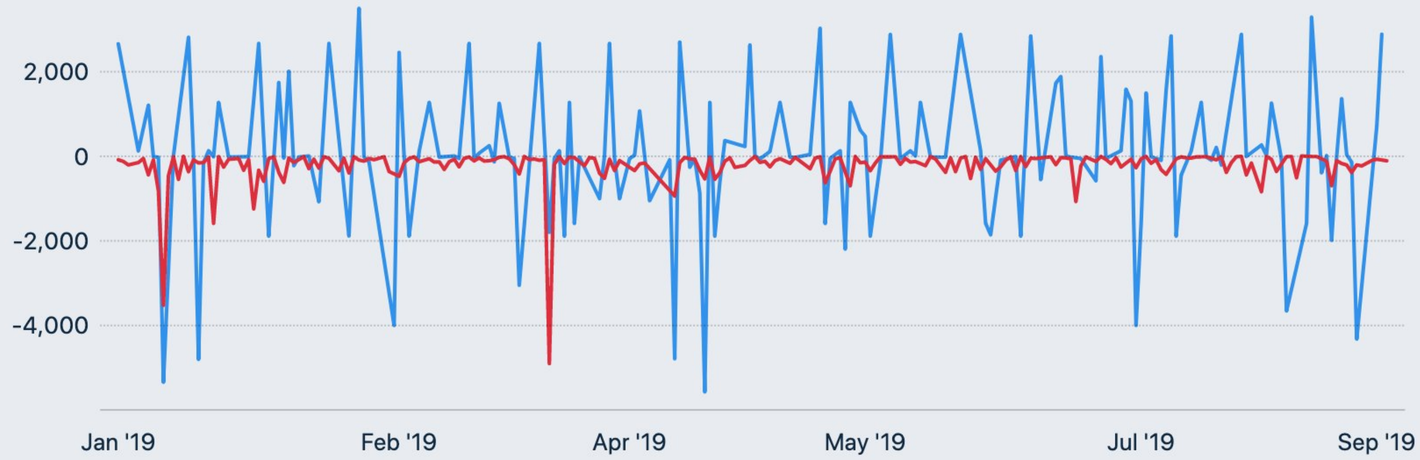
For budget	68,829.89
Chase	0.00
Credit Card	-5,112.23
Test Account	73,942.12
Off budget	-2,523.36
Test Account2	-2,523.36

Closed Accounts...

Sign in

```
let accounts = await query("accounts").get(["id"]);

let result1 = await query("transactions")
  .filter({
    date: { $gt: 20190101 },
    acct: accounts[6].id
  })
  .groupBy({ $substr: ["date", 0, 10] })
  .get([
    "id",
    { amount: { $sum: "amount" } },
    { date: { $substr: ["date", 0, 10] } }
  ]);

let result2 = await query("transactions")
  .filter({
    date: { $gt: 20190101 },
```

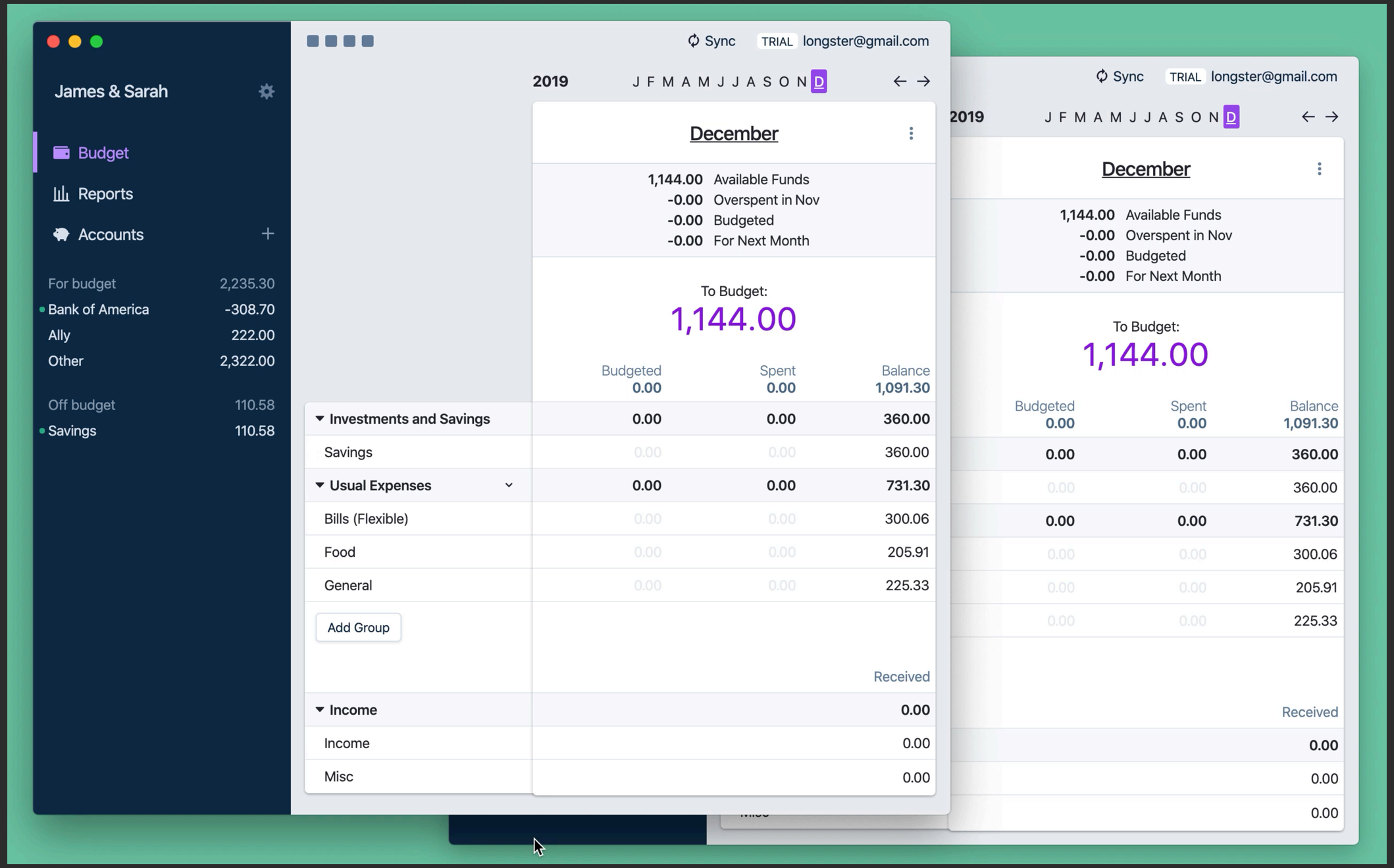
Run Prettier



a mobile app!

uh oh, syncing?


BUILDING MY
OWN SYNC ENGINE
SQLite
ME
BUILDING
A PRODUCT
imgflip.com


James & Sarah
Budget
Reports
Accounts
For budget 2,235.30
Bank of America -308.70
Ally 222.00
Other 2,322.00
Off budget 110.58
Savings 110.58
Sync
TRIAL longster@gmail.com
2019
J F M A M J J A S O N D
December
1,144.00 Available Funds
-0.00 Overspent in Nov
-0.00 Budgeted
-0.00 For Next Month
To Budget:
1,144.00
Budgeted 0.00
Spent 0.00
Balance 1,091.30
Investments and Savings 0.00 0.00 360.00
Savings 0.00 0.00 360.00
Usual Expenses 0.00 0.00 731.30
Bills (Flexible) 0.00 0.00 300.06
Food 0.00 0.00 205.91
General 0.00 0.00 225.33
Add Group
Received
Income 0.00
Income 0.00
Misc 0.00

simple data (5-10 MB database)
syncing on top of sqlite

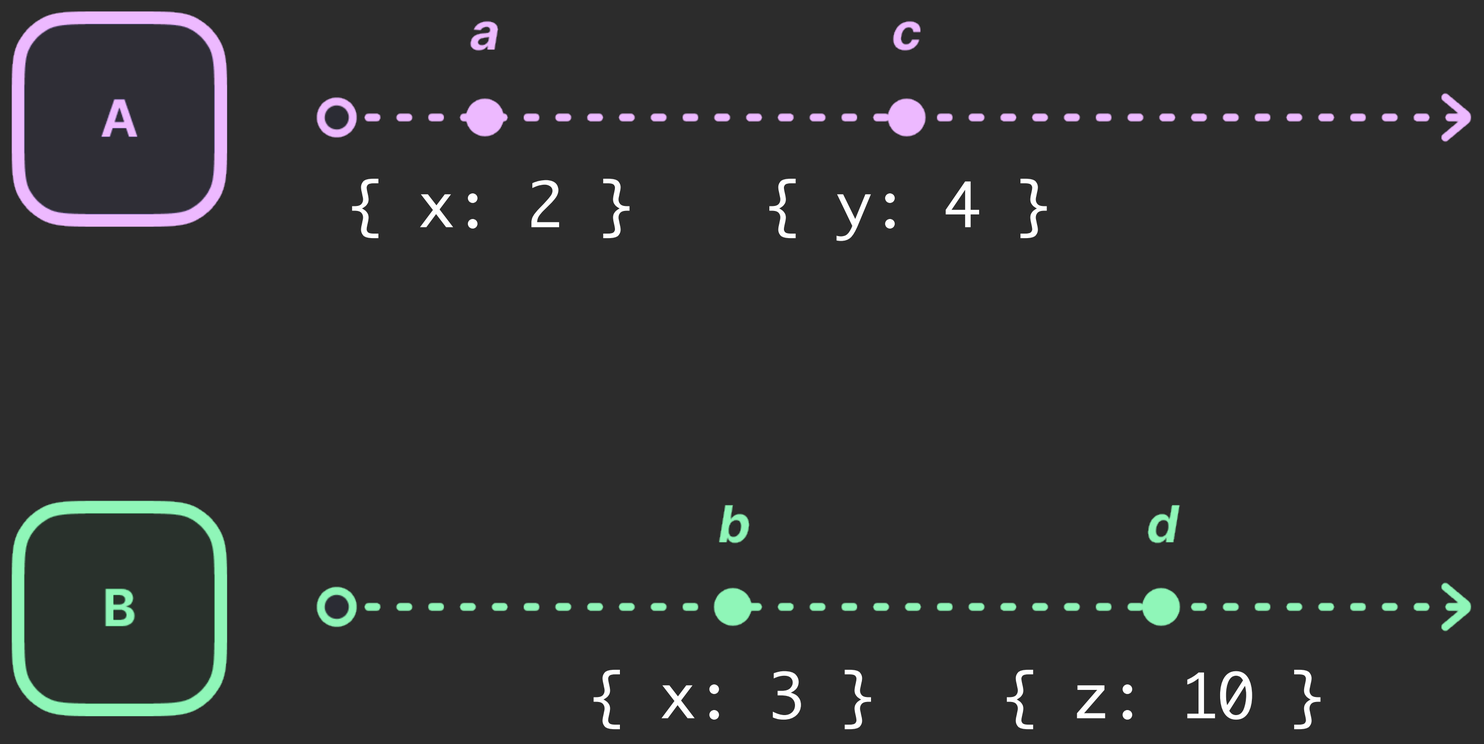
unreliable ordering

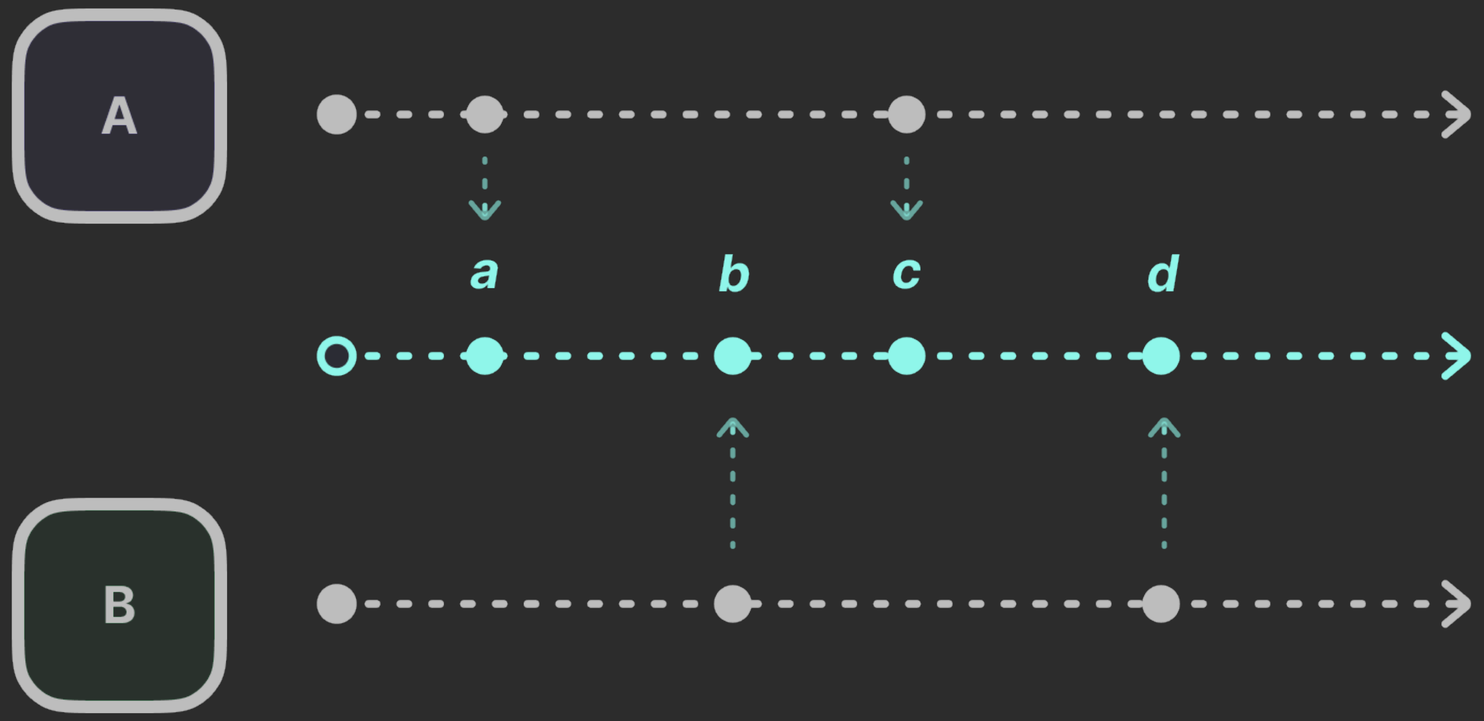
conflicts

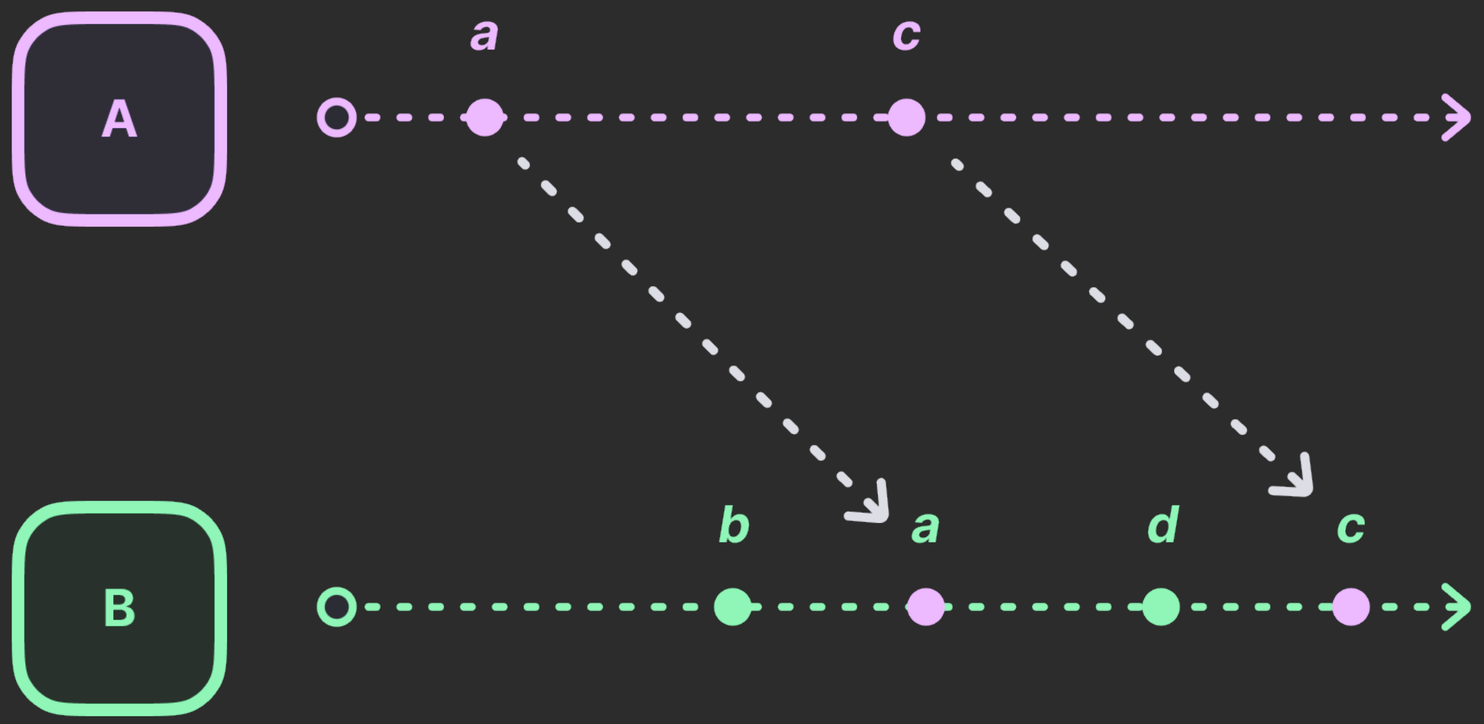
also... needs to work 100% of the time

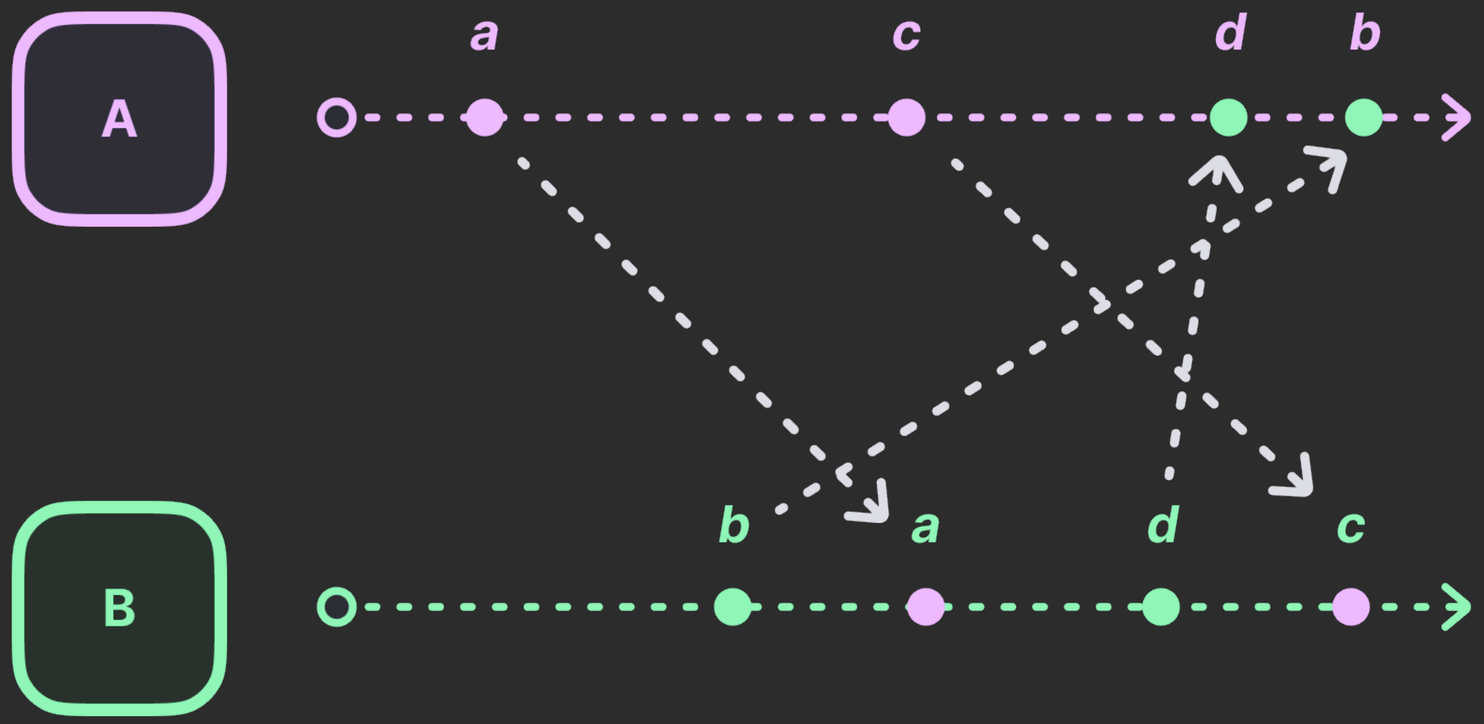
unreliable ordering

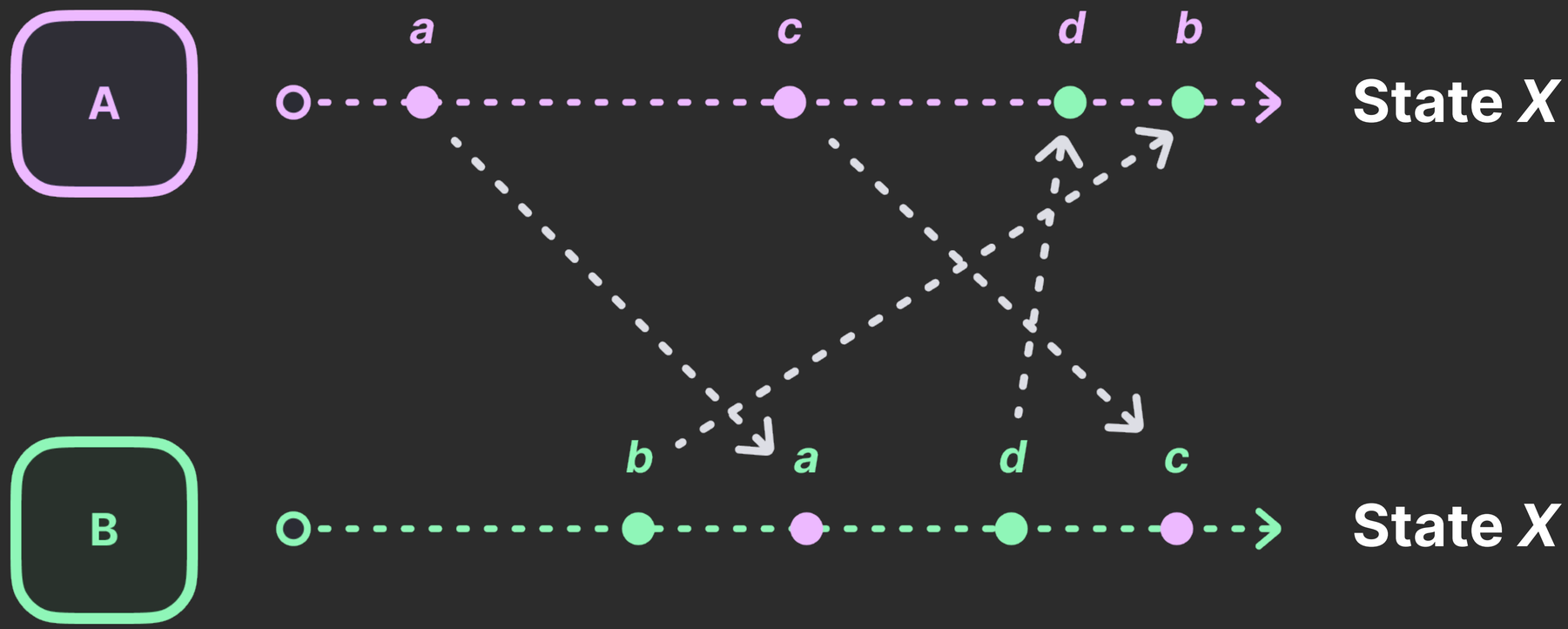
conflicts


A
a
c
{ x: 2 }
{ y: 4 }
B
b
d
{ x: 3 }
{ z: 10 }


A
B
a
b
c
d


A
a
c
B
b
a
d
c

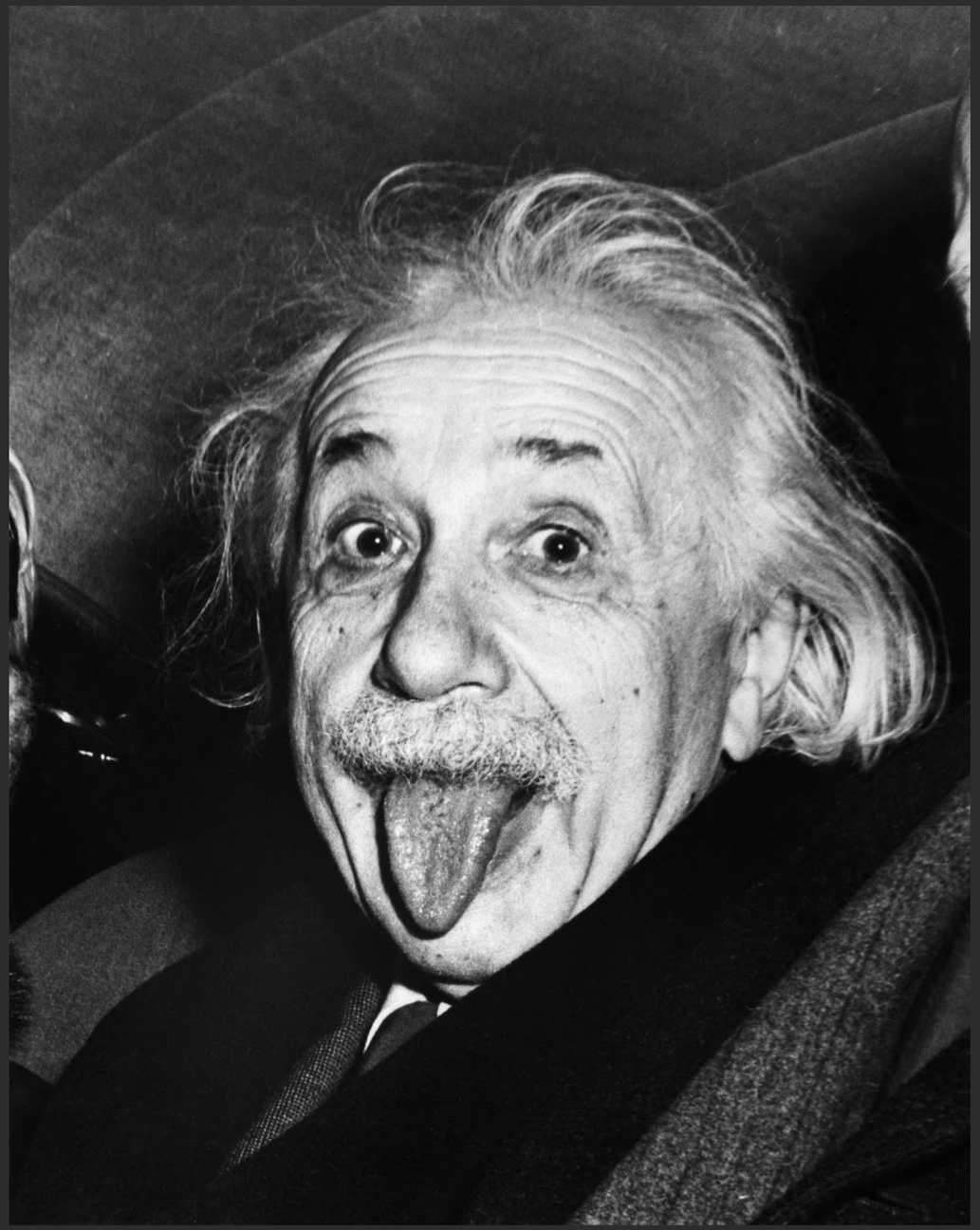

A
a
c
d
b
B
b
a
d
c



eventual consistency

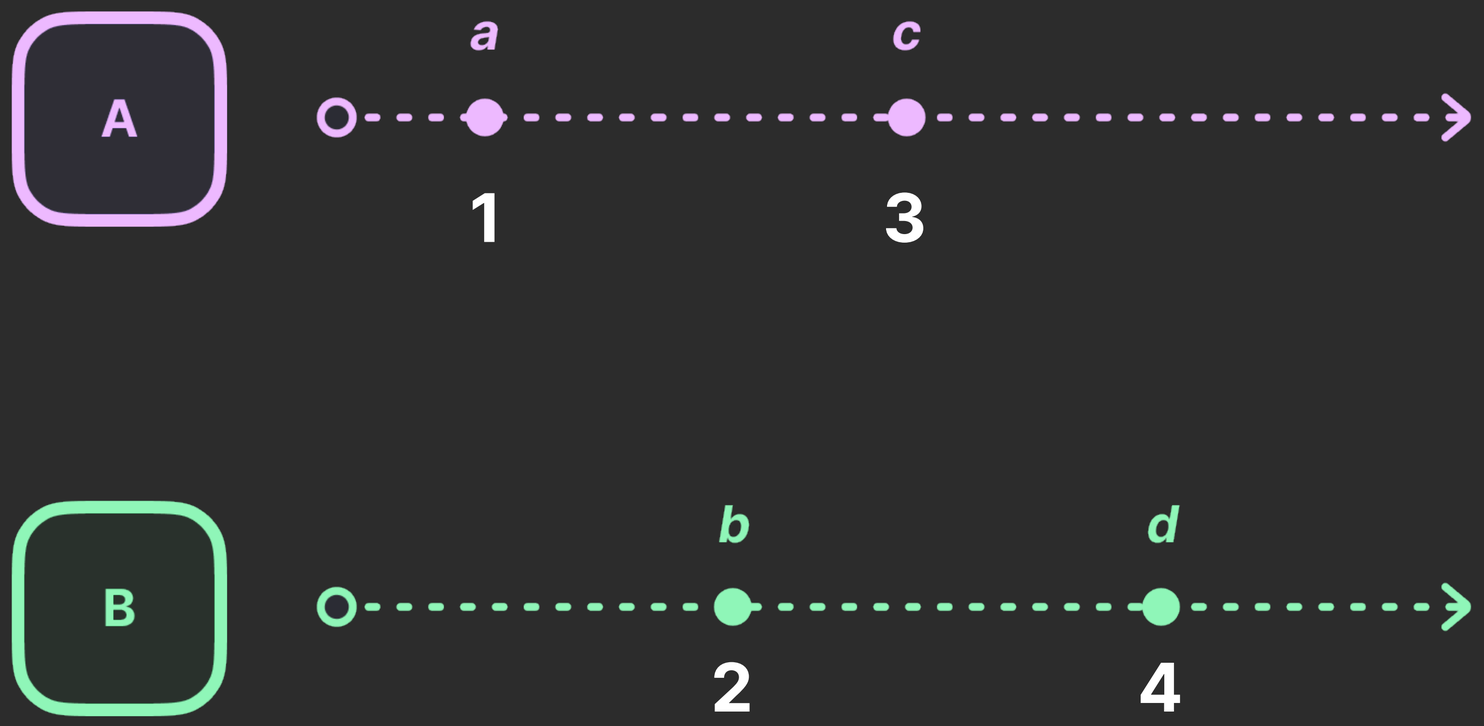
how do we solve unreliable ordering?

we need to assign timestamps



“Time is relative; its only worth depends upon what we do as it is passing.”

- Albert Einstein


A
a
c
1
3
B
b
d
2
4

- vector clock
- hybrid logical clock (HLC) *
- per-device
- assigns “timestamps” to changes

* https://cse.buffalo.edu/tech-reports/2014-04.pdf

```
{ x: 3,
  timestamp: “2019-11-05T15:29:40.273Z-0001-eede1195b7d94dd5" }

{ x: 5,
  timestamp: “2019-11-04T15:35:40.273Z-0001-85b8c0d2bbb57d99" }
```

“Simplicity is a prerequisite for reliability.”

- Edsger W. Dijkstra

```
// clock.js
let _clock = null;

function setClock(clock) {
  _clock = clock;
}

function getClock() {
  return _clock;
}

function makeClock(timestamp, merkle = {}) {
  return { timestamp: MutableTimestamp.from(timestamp), merkle };
}

function serializeClock(clock) {
  return JSON.stringify({
    timestamp: clock.timestamp.toString(),
    merkle: clock.merkle
  });
}

function deserializeClock(clock) {
  const data = JSON.parse(clock);
  return {
    timestamp: Timestamp.from(Timestamp.parse(data.timestamp)),
    merkle: data.merkle
  };
}

function makeClientId() {
  return uuidv4()
    .replace(/-/g, '')
    .slice(-16);
}


// timestamp.js
var config = {
  // Maximum physical clock drift allowed, in ms
  maxDrift: 60000
};

class Timestamp {
  constructor(millis, counter, node) {
    this._state = {
      millis: millis,
      counter: counter,
      node: node
    };
  }

  valueOf() {
    return this.toString();
  }

  toString() {
    return [
      new Date(this.millis()).toISOString(),
      (
        '0000' +
        this.counter()
          .toString(16)
          .toUpperCase()
      ).slice(-4),
      ('0000000000000000' + this.node()).slice(-16)
    ].join('-');
  }

  millis() {
    return this._state.millis;
  }

  counter() {
    return this._state.counter;
  }

  node() {
    return this._state.node;
  }

  hash() {
    return murmurhash.v3(this.toString());
  }
}
```

```
class MutableTimestamp extends Timestamp {
  setMillis(n) {
    this._state.millis = n;
  }

  setCounter(n) {
    this._state.counter = n;
  }

  setNode(n) {
    this._state.node = n;
  }
}

MutableTimestamp.from = timestamp => {
  return new MutableTimestamp(
    timestamp.millis(),
    timestamp.counter(),
    timestamp.node()
  );
};

// Timestamp generator initialization
// * sets the node ID to an arbitrary value
// * useful for mocking/unit testing
Timestamp.init = function(options = {}) {
  if (options.maxDrift) {
    config.maxDrift = options.maxDrift;
  }
};

/**
 * Timestamp send. Generates a unique, monotonic timestamp suitable
 * for transmission to another system in string format
 */
Timestamp.send = function(clock) {
  // Retrieve the local wall time
  var phys = Date.now();

  // Unpack the clock.timestamp logical time and counter
  var lOld = clock.timestamp.millis();
  var cOld = clock.timestamp.counter();

  // Calculate the next logical time and counter
  // * ensure that the logical time never goes backward
  // * increment the counter if phys time does not advance
  var lNew = Math.max(lOld, phys);
  var cNew = lOld === lNew ? cOld + 1 : 0;

  // Check the result for drift and counter overflow
  if (lNew - phys > config.maxDrift) {
    throw new Timestamp.ClockDriftError(lNew, phys, config.maxDrift);
  }
  if (cNew > 65535) {
    throw new Timestamp.OverflowError();
  }

  // Repack the logical time/counter
  clock.timestamp.setMillis(lNew);
  clock.timestamp.setCounter(cNew);

  return new Timestamp(
    clock.timestamp.millis(),
    clock.timestamp.counter(),
    clock.timestamp.node()
  );
};

// Timestamp receive. Parses and merges a timestamp from a remote
// system with the local timeglobal uniqueness and monotonicity are
// preserved
Timestamp.recv = function(clock, msg) {
  var phys = Date.now();

  // Unpack the message wall time/counter
  var lMsg = msg.millis();
  var cMsg = msg.counter();

  // Assert the node id and remote clock drift
  if (msg.node() === clock.timestamp.node()) {
    throw new Timestamp.DuplicateNodeError(clock.timestamp.node());
  }
  if (lMsg - phys > config.maxDrift) {
    throw new Timestamp.ClockDriftError();
  }

  // Unpack the clock.timestamp logical time and counter
  var lOld = clock.timestamp.millis();
  var cOld = clock.timestamp.counter();

  // Calculate the next logical time and counter.
  // Ensure that the logical time never goes backward;
  // * if all logical clocks are equal, increment the max counter,
  // * if max = old > message, increment local counter,
  // * if max = messsage > old, increment message counter,
  // * otherwise, clocks are monotonic, reset counter
  var lNew = Math.max(Math.max(lOld, phys), lMsg);
  var cNew =
    lNew === lOld && lNew === lMsg
      ? Math.max(cOld, cMsg) + 1
      : lNew === lOld
      ? cOld + 1
      : lNew === lMsg
      ? cMsg + 1
      : 0;

  // Check the result for drift and counter overflow
  if (lNew - phys > config.maxDrift) {
    throw new Timestamp.ClockDriftError();
  }
  if (cNew > 65535) {
    throw new Timestamp.OverflowError();
  }

  // Repack the logical time/counter
  clock.timestamp.setMillis(lNew);
  clock.timestamp.setCounter(cNew);

  return new Timestamp(
    clock.timestamp.millis(),
    clock.timestamp.counter(),
    clock.timestamp.node()
  );
};

/**
 * Converts a fixed-length string timestamp to the structured value
 */
Timestamp.parse = function(timestamp) {
  if (typeof timestamp === 'string') {
    var parts = timestamp.split('-');
    if (parts && parts.length === 5) {
      var millis = Date.parse(parts.slice(0, 3).join('-')).valueOf();
      var counter = parseInt(parts[3], 16);
      var node = parts[4];
      if (!isNaN(millis) && !isNaN(counter))
        return new Timestamp(millis, counter, node);
    }
  }
  return null;
};

Timestamp.since = isoString => {
  return isoString + '-0000-0000000000000000';
};

Timestamp.DuplicateNodeError = class extends Error {
  constructor(node) {
    super();
    this.type = 'DuplicateNodeError';
    this.message = 'duplicate node identifier ' + node;
  }
};

Timestamp.ClockDriftError = class extends Error {
  constructor(...args) {
    super();
    this.type = 'ClockDriftError';
    this.message = ['maximum clock drift exceeded'].concat(args).join(' ');
  }
};

Timestamp.OverflowError = class extends Error {
  constructor() {
    super();
    this.type = 'OverflowError';
    this.message = 'timestamp counter overflow';
  }
};

setClock(makeClock(new Timestamp(0, 0, makeClientId())));
```

~~unreliable ordering~~

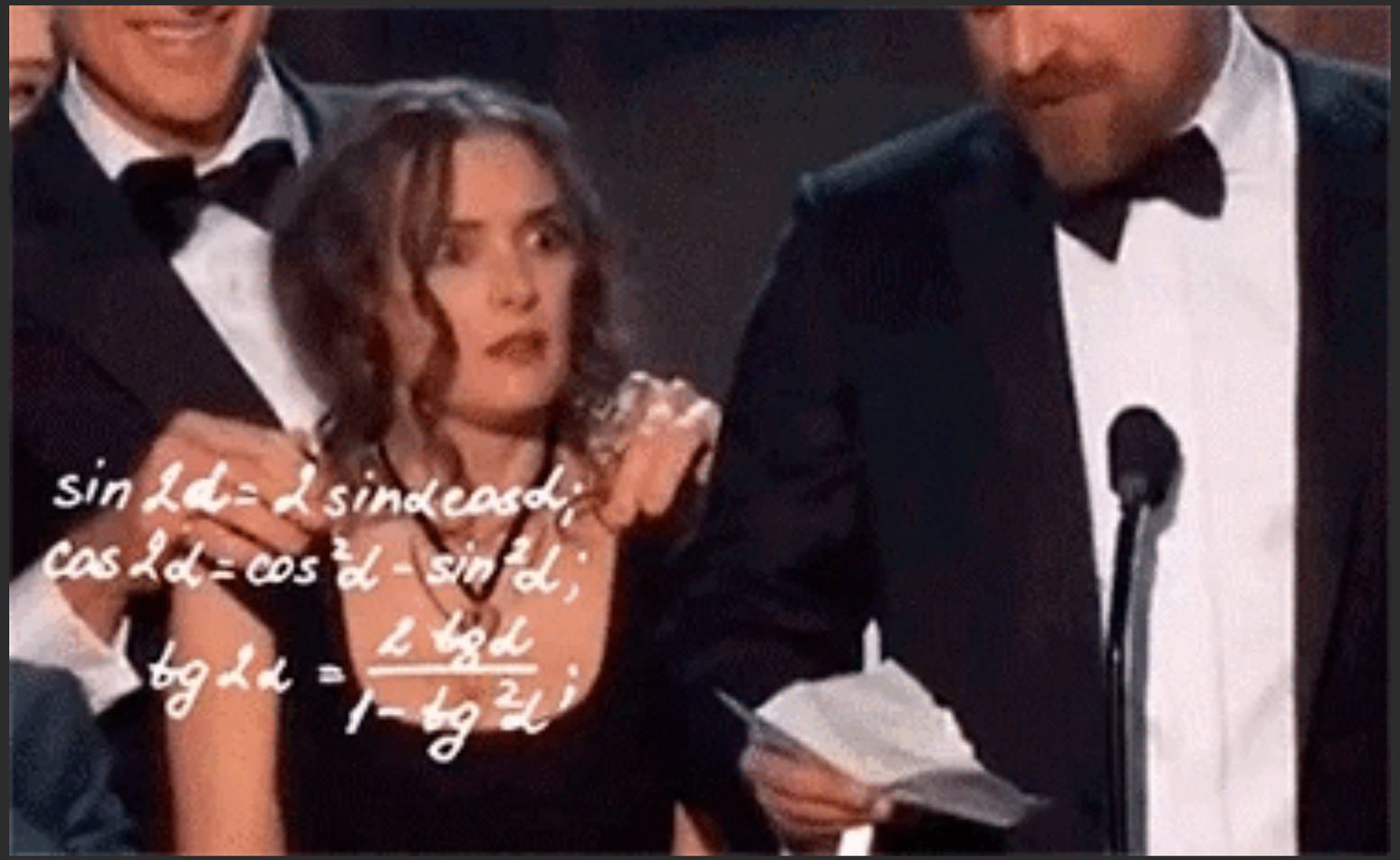
conflicts

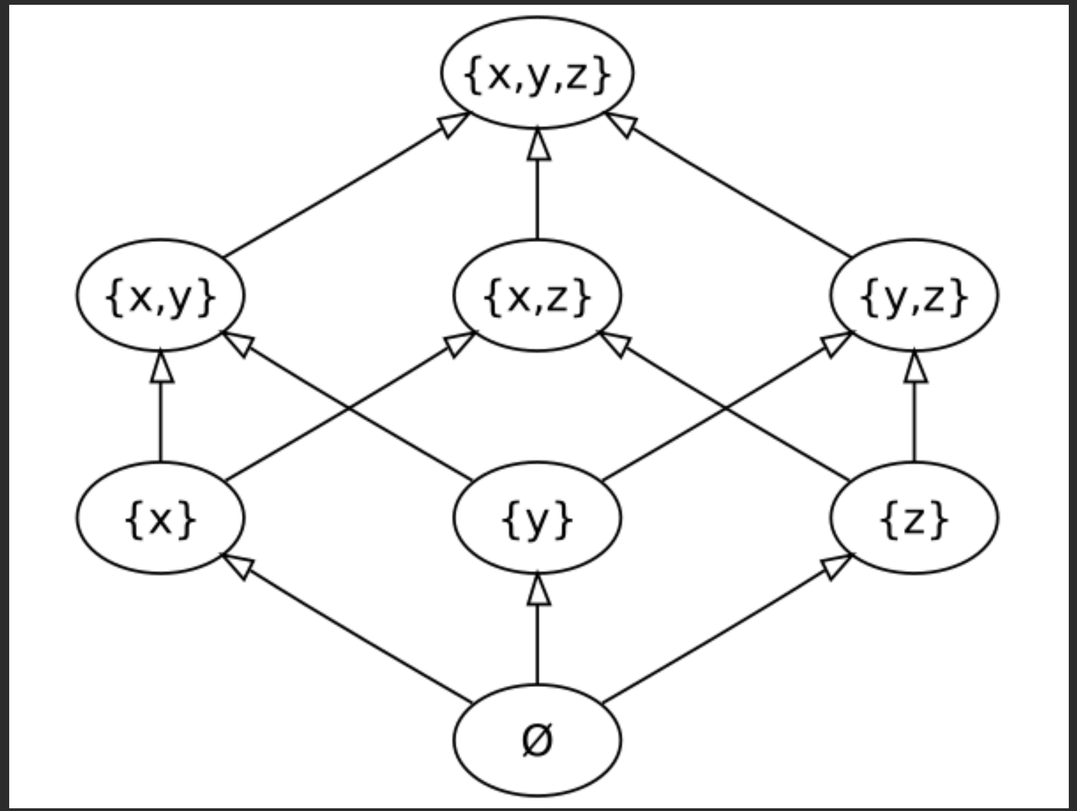
manual conflict resolution


Did you just tell me to go fuck myself?
I believe I did, Bob.

~~manual conflict resolution~~

CRDTs


$\sin 2\alpha = 2\sin\alpha\cos\alpha;$
$\cos 2\alpha = \cos^2\alpha - \sin^2\alpha;$
$\operatorname{tg} 2\alpha = \frac{2\operatorname{tg}\alpha}{1-\operatorname{tg}^2\alpha};$



partially ordered monoid in the category of endofunctors with a least upper bound

conflict-free replicated data types

G-Counter

PN-Counter

G-Set

2P-Set

LWW-Element-Set

OR-Set

ORSWOT (???)

and more...

conflict-free replicated data types

commutative

2 + 3 = 3 + 2 = 5

idempotent

f(x)
f(x) f(x)
f(x) f(x) f(x)

```
{ x: 300,
  timestamp: “2019-11-05T15:29:40.273Z-0000-eede1195b7d94dd5” }

{ y: 73,
  timestamp: “2019-11-02T15:35:32.743Z-0000-85b8c0d2bbb57d99" }

{ x: 8,
  timestamp: “2019-11-02T15:35:32.743Z-0001-85b8c0d2bbb57d99" }

{ z: 114,
  timestamp: “2019-11-02T15:35:32.743Z-0002-85b8c0d2bbb57d99" }
```

{}

```
{ x: 300,
  timestamp: “2019-11-05T15:29:40.273Z-0000-eede1195b7d94dd5” }

{ y: 73,
  timestamp: “2019-11-02T15:35:32.743Z-0000-85b8c0d2bbb57d99" }

{ x: 8,
  timestamp: “2019-11-02T15:35:32.743Z-0001-85b8c0d2bbb57d99" }

{ z: 114,
  timestamp: “2019-11-02T15:35:32.743Z-0002-85b8c0d2bbb57d99" }
```

{}

LWW-Map

Map → Last-Write-Wins-Map (LWW-Map)

Set → Grow-Only Set (G-Set)

```
{ id: “0aead5b3-203e-475f-b3f5-1ab9ace69620”,
  timestamp: “2019-11-05T15:29:40.273Z-0000-eede1195b7d94dd5” }

{ id: “0aead5b3-203e-475f-b3f5-1ab9ace69620”,
  timestamp: “2019-11-02T15:35:32.743Z-0000-85b8c0d2bbb57d99" }

{ id: “e5b4c695-a632-4cec-a646-d61b32b2351f”,
  timestamp: “2019-11-02T15:35:32.743Z-0001-85b8c0d2bbb57d99" }
```

```
(“0aead5b3-203e-475f-b3f5-1ab9ace69620”,
  “e5b4c695-a632-4cec-a646-d61b32b2351f”)
```

G-Set

How to take basic relational data and turn it into CRDTs?

SQLite table ⟶ G-Set of LWW-Maps

A new table: `messages_crdt`

id	timestamp	dataset	row	column	value
1	...	accounts	e29d69a6-148e-4072-8513-f7bca511b013	name	S:Checking
2	...	accounts	e29d69a6-148e-4072-8513-f7bca511b013	type	S:checking
3	...	accounts	e29d69a6-148e-4072-8513-f7bca511b013	offbudget	N:0
4	...	accounts	e29d69a6-148e-4072-8513-f7bca511b013	closed	N:0
5	...	payees	503189fc-efc1-479d-8b35-6a0daf91737d	name	S:
6	...	payees	503189fc-efc1-479d-8b35-6a0daf91737d	transfer_a	S:e29d69a6

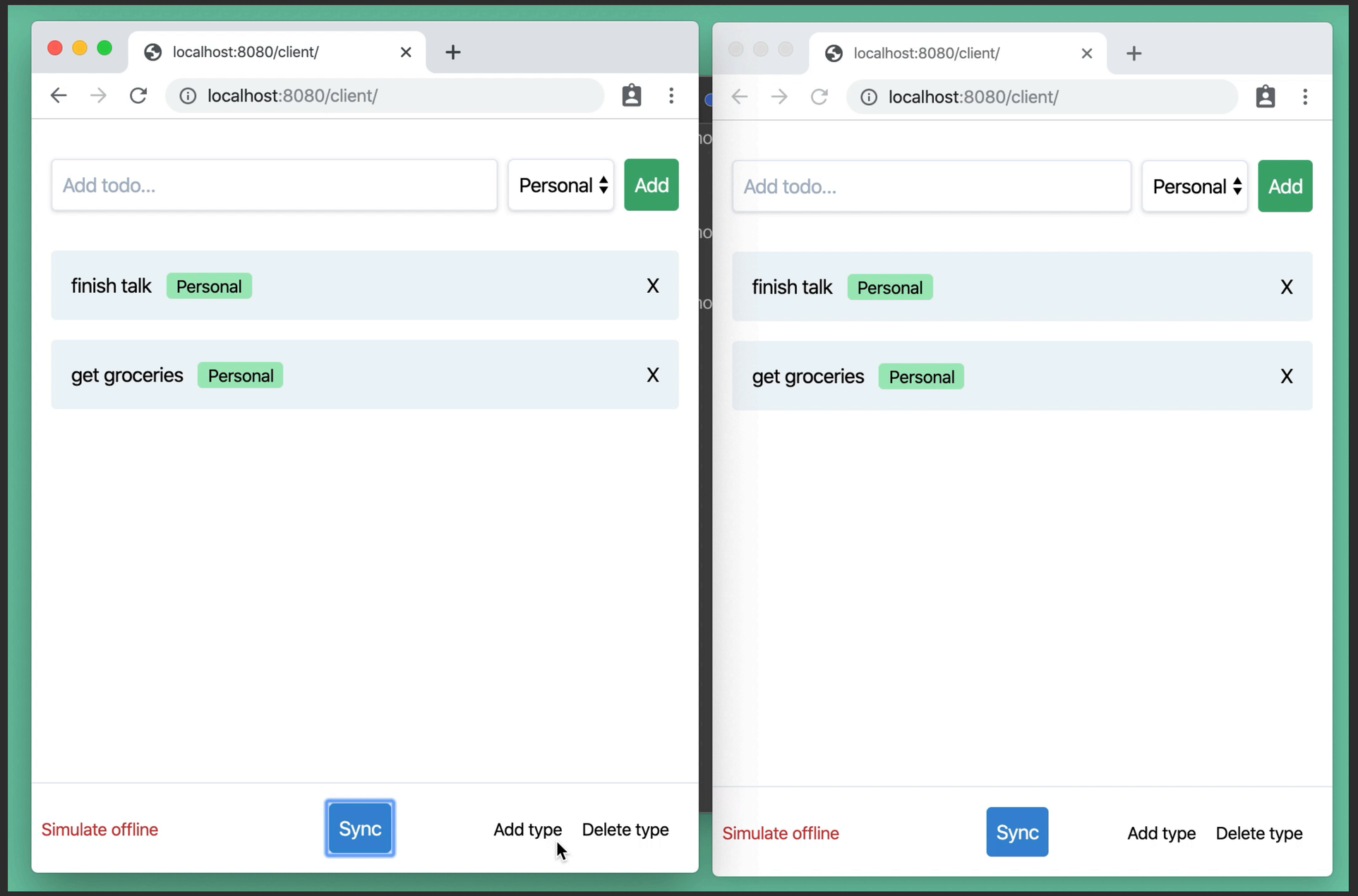
```
update("transactions", {
  id: "30127b2e-f74c-4a19-af65-debfb7a6a55b",
  name: "Kroger",
  amount: 450
})

// becomes

{
  dataset: "transactions",
  row: "30127b2e-f74c-4a19-af65-debfb7a6a55b",
  column: "name",
  value: "Kroger",
  timestamp: "2019-11-02T15:35:32.743Z-0000-85b8c0d2bbb57d99"
}
{
  dataset: "transactions",
  row: "30127b2e-f74c-4a19-af65-debfb7a6a55b",
  column: "amount",
  value: 450,
  timestamp: "2019-11-02T15:35:32.743Z-0001-85b8c0d2bbb57d99"
}
```

```
delete("transactions", "30127b2e-f74c-4a19-af65-debfb7a6a55b")

// becomes

{
  dataset: "transactions",
  row: "30127b2e-f74c-4a19-af65-debfb7a6a55b",
  column: "tombstone",
  value: 1,
  timestamp: "2019-11-02T15:35:32.743Z-0000-85b8c0d2bbb57d99"
}
```

Other features

Ensuring consistency with a merkle tree

End-to-end encryption



Live: https://crdt.jlongster.com/

Source: https://github.com/jlongster/crdt-example-app

```
let _messages = [];
let _data = {
  todos: [],
  todoTypes: [],
  todoTypeMapping: []
};

function insert(table, row) {
  let id = uuidv4();
  let fields = Object.keys(row);

  sendMessages(
    fields.map(k => {
      return {
        dataset: table,
        row: row.id || id,
        column: k,
        value: row[k],
        timestamp: Timestamp.send(getClock()).toString()
      };
    })
  );

  return id;
}

function update(table, params) {
  let fields = Object.keys(params).filter(k => k !== 'id');

  sendMessages(
    fields.map(k => {
      return {
        dataset: table,
        row: params.id,
        column: k,
        value: params[k],
        timestamp: Timestamp.send(getClock()).toString()
      };
    })
  );
}

function delete_(table, id) {
  sendMessages([
    {
      dataset: table,
      row: id,
      column: 'tombstone',
      value: 1,
      timestamp: Timestamp.send(getClock()).toString()
    }
  ]);
}

function getTodos() {
  let todos = _data.todos
    .filter(todo => todo.tombstone !== 1)
    .map(todo => ({
      ...todo,
      type: todo.type ? getTodoType(todo.type) : null
    }));

  todos.sort((t1, t2) => {
    if (t1.order < t2.order) {
      return 1;
    } else if (t1.order > t2.order) {
      return -1;
    }
    return 0;
  });

  return todos;
}

function getTodoType(id) {
  // Go through the mapping table, which is a layer of indirection. In
  // SQL you could think of doing a LEFT JOIN onto this table and
  // using the id from the mapping table instead of the raw id
  let mapping = _data.todoTypeMapping.find(m => m.id === id);
  let type =
    mapping && _data.todoTypes.find(type => type.id === mapping.targetId);
  return type && type.tombstone !== 1 ? type : null;
}

function getNumTodos() {
  return _data.todos.length;
}

function getTodoTypes() {
  return _data.todoTypes.filter(todoType => todoType.tombstone !== 1);
}

function insertTodoType({ name, color }) {
  let id = insert('todoTypes', { name, color });

  // Create an entry in the mapping table that points it to itself
  insert('todoTypeMapping', { id, targetId: id });
}

function deleteTodoType(id, targetId) {
  if (targetId) {
    // We need to update all the pointers the point to the type that
    // we are deleting and point it to the new type. This already
    // includes the type we are deleting (when created, it creates a
    // mapping to itself)
    for (let mapping of _data.todoTypeMapping) {
      if (mapping.targetId === id) {
        update('todoTypeMapping', { id: mapping.id, targetId });
      }
    }
  }

  delete_('todoTypes', id);
}

setClock(makeClock(new Timestamp(0, 0, makeClientId())));

let _onSync = null;
let _syncEnabled = true;

function setSyncingEnabled(flag) {
  _syncEnabled = flag;
}

async function post(data) {
  let res = await fetch('https://crdt.jlongster.com/server/sync', {
    method: 'POST',
    body: JSON.stringify(data),
    headers: {
      'Content-Type': 'application/json'
    }
  });
  res = await res.json();

  if (res.status !== 'ok') {
    throw new Error('API error: ' + res.reason);
  }
  return res.data;
}

function apply(msg) {
  let table = _data[msg.dataset];
  if (!table) {
    throw new Error('Unknown dataset: ' + msg.dataset);
  }

  let row = table.find(row => row.id === msg.row);
  if (!row) {
    table.push({ id: msg.row, [msg.column]: msg.value });
  } else {
    row[msg.column] = msg.value;
  }
}

function compareMessages(messages) {
  let existingMessages = new Map();

  // This could be optimized, but keeping it simple for now. Need to
  // find the latest message that exists for the dataset/row/column
  // for each incoming message, so sort it first

  let sortedMessages = [..._messages].sort((m1, m2) => {
    if (m1.timestamp < m2.timestamp) {
      return 1;
    } else if (m1.timestamp > m2.timestamp) {
      return -1;
    }
    return 0;
  });

  messages.forEach(msg1 => {
    let existingMsg = sortedMessages.find(
      msg2 =>
        msg1.dataset === msg2.dataset &&
        msg1.row === msg2.row &&
        msg1.column === msg2.column
    );
    existingMessages.set(msg1, existingMsg);
  });

  return existingMessages;
}

function applyMessages(messages) {
  let existingMessages = compareMessages(messages);
  let clock = getClock();

  messages.forEach(msg => {
    let existingMsg = existingMessages.get(msg);

    if (!existingMsg || existingMsg.timestamp < msg.timestamp) {
      apply(msg);
    }

    if (!existingMsg || existingMsg.timestamp !== msg.timestamp) {
      clock.merkle = merkle.insert(
        clock.merkle,
        Timestamp.parse(msg.timestamp)
      );
      _messages.push(msg);
    }
  });

  _onSync && _onSync();
}

function sendMessages(messages) {
  applyMessages(messages);
  sync(messages);
}

function receiveMessages(messages) {
  messages.forEach(msg =>
    Timestamp.recv(getClock(), Timestamp.parse(msg.timestamp))
  );

  applyMessages(messages);
}

function onSync(func) {
  _onSync = func;
}

async function sync(initialMessages = [], since = null) {
  if (!_syncEnabled) {
    return;
  }

  let messages = initialMessages;

  if (since) {
    let timestamp = new Timestamp(since, 0, '0').toString();
    messages = _messages.filter(msg => msg.timestamp >= timestamp);
  }

  let result = await post({
    group_id: 'my-group',
    client_id: getClock().timestamp.node(),
    messages,
    merkle: getClock().merkle
  });

  receiveMessages(result.messages);

  let diffTime = merkle.diff(result.merkle, getClock().merkle);
  if (diffTime) {
    if (since && since === diffTime) {
      throw new Error(
        'A bug happened while syncing and the client ' +
          'was unable to get in sync with the server. ' +
          "This is an internal error that shouldn't happen"
      );
    }

    return sync([], diffTime);
  }
}
```

server

132 lines of JS

client

639 lines of JS

65 tweets!

only dependencies: `uuid` `murmurhash`

Conclusion

Local apps have superior UX. They are super fast, no latency, and work offline

We've got to start simplifying our solutions

Clocks (particularly HLCs) and CRDTs are an elegant solution to distributed apps

- Actual: https://actualbudget.com/
- Hybrid logical clocks: https://cse.buffalo.edu/tech-reports/2014-04.pdf
- CRDTs: https://bit.ly/2DMk0AD
- Demo app:
  - Live: https://crdt.jlongster.com/
  - Source: https://github.com/jlongster/crdt-example-app