Decentralised collaborative documents

a.k.a. Building Google Docs without the "Google"

Using IPFS, Y.js a Browser and some JavaScript

David Dias & Pedro Teixeira

Protocol Labs

Warning

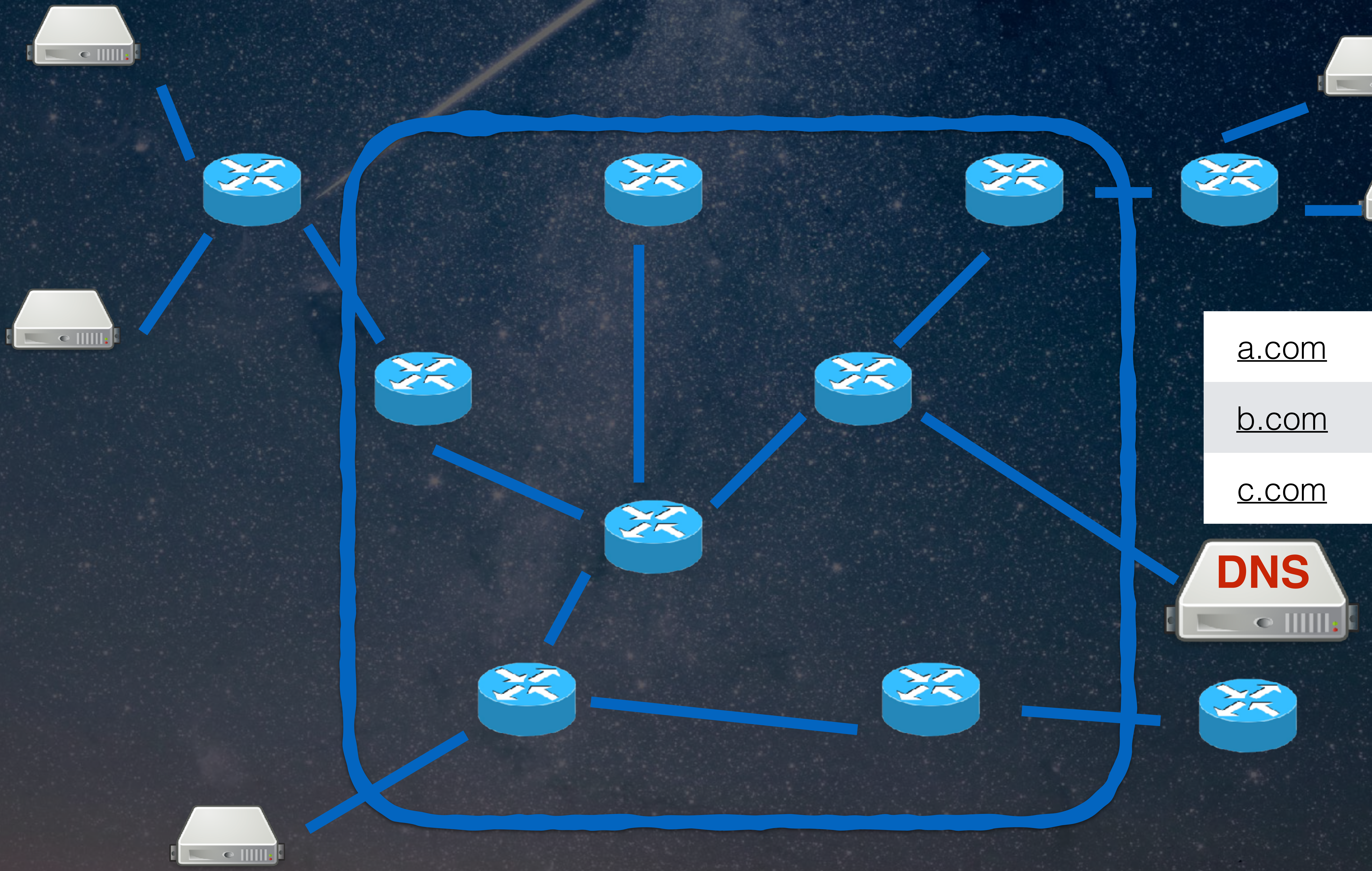
This is going to envolve some coding.
Probably pairing up!

Agenda

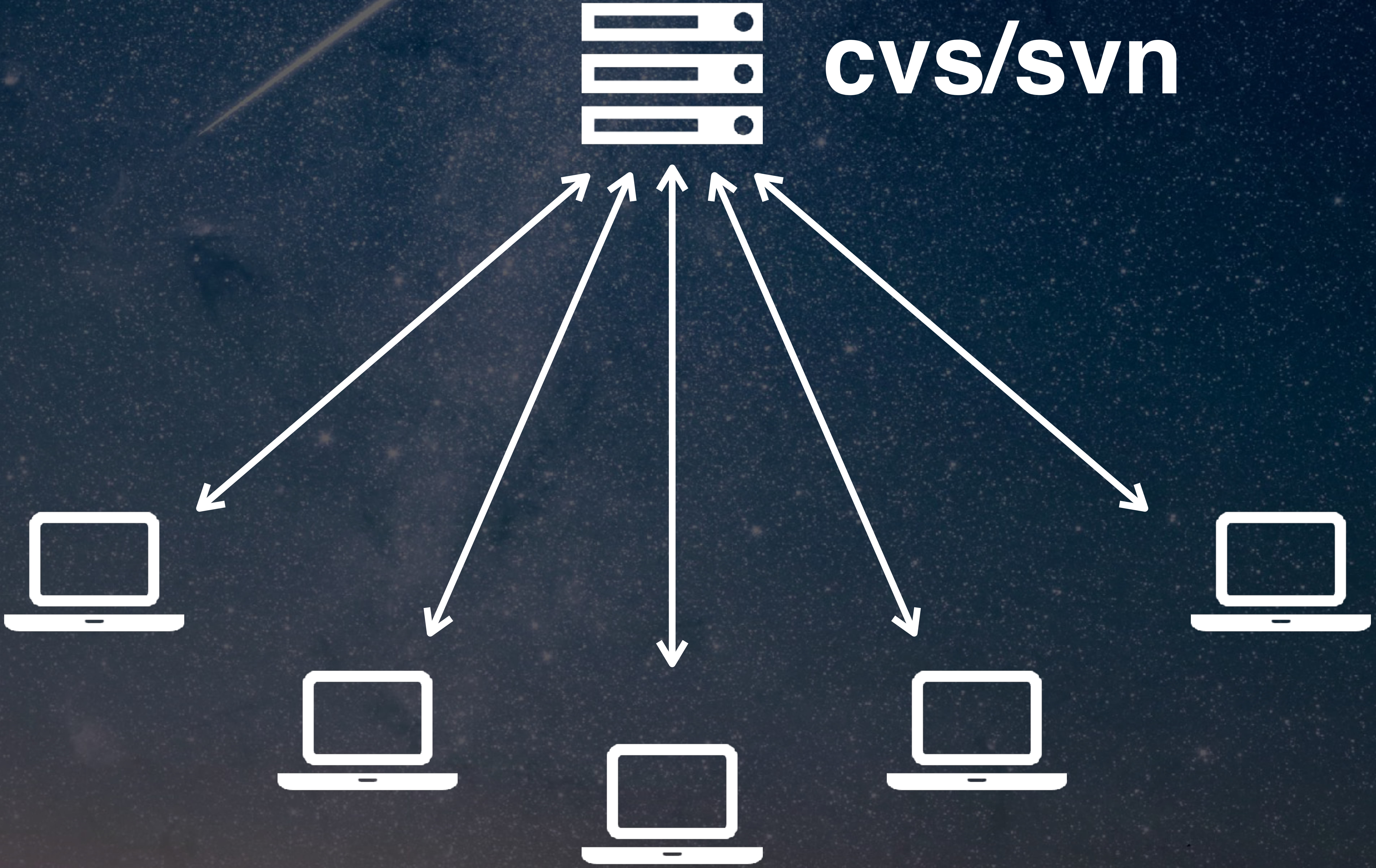
- Grouping
- Motivation for Decentralising Web Apps
- Goal
- Setup
- Code! (30 mins)
- **Discussion:** Decentralised Web Apps
- Quick Announcement

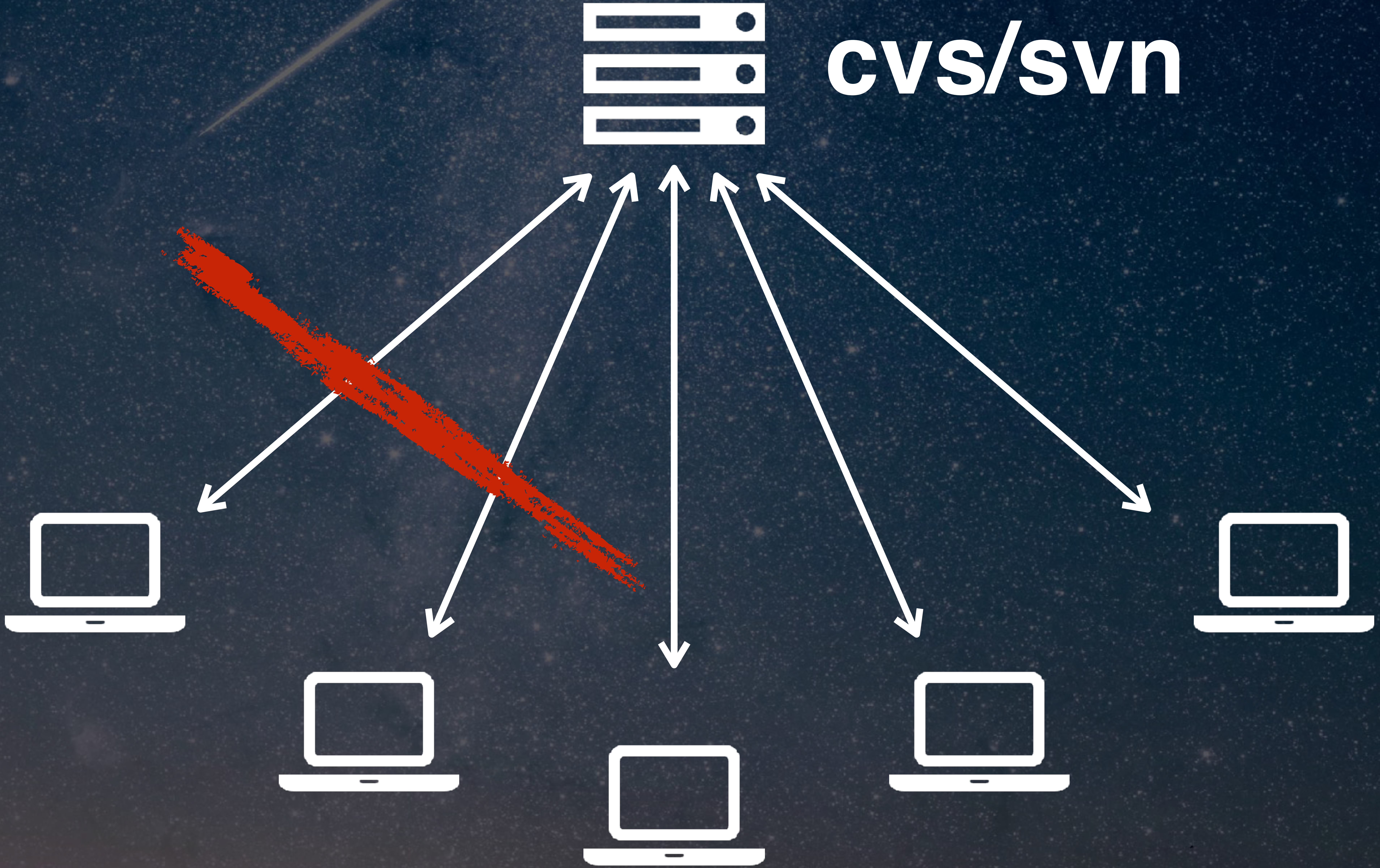
Motivation

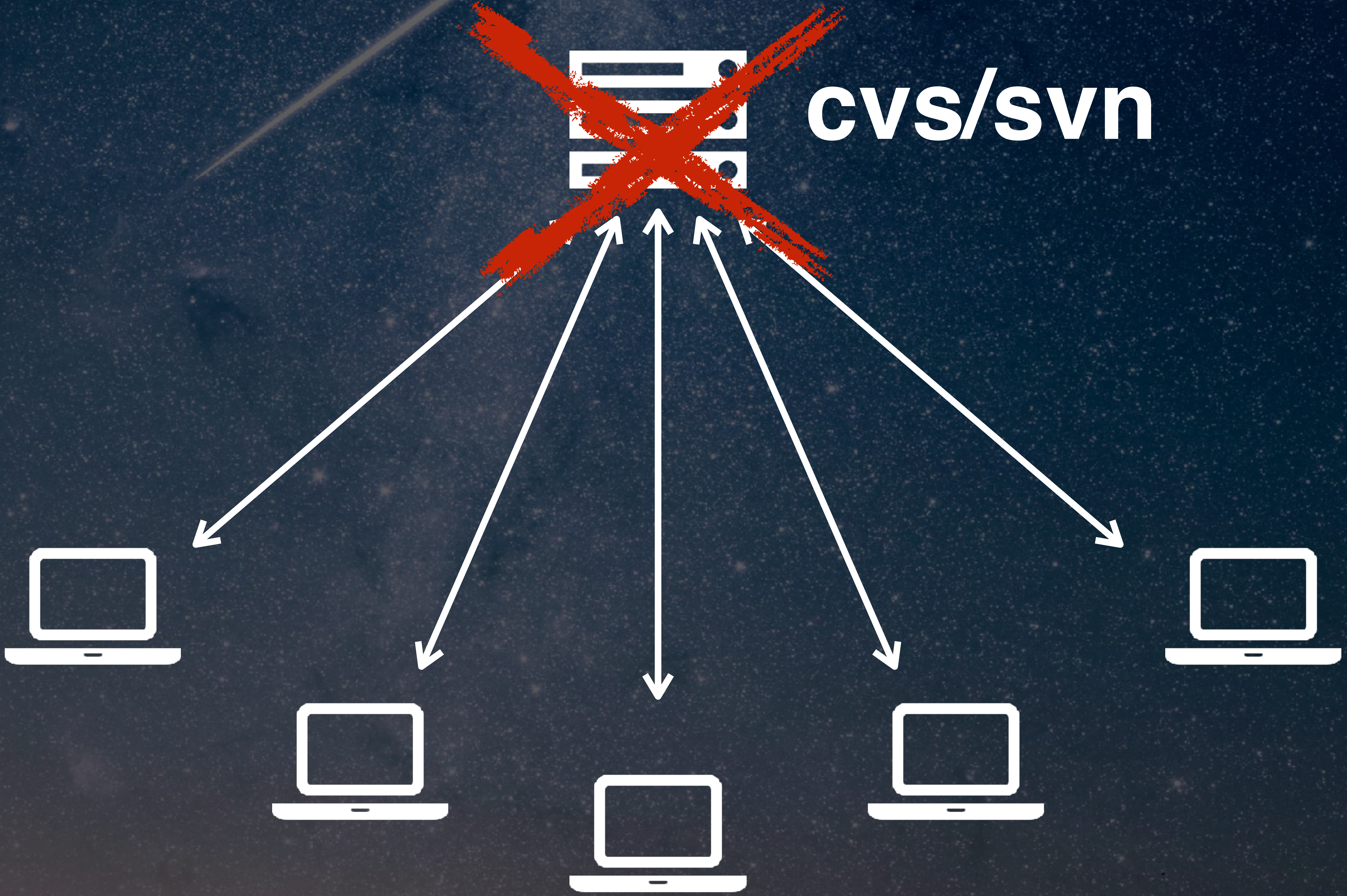
Cloud-based software has taken the web a long way, but lead us into centralisation.

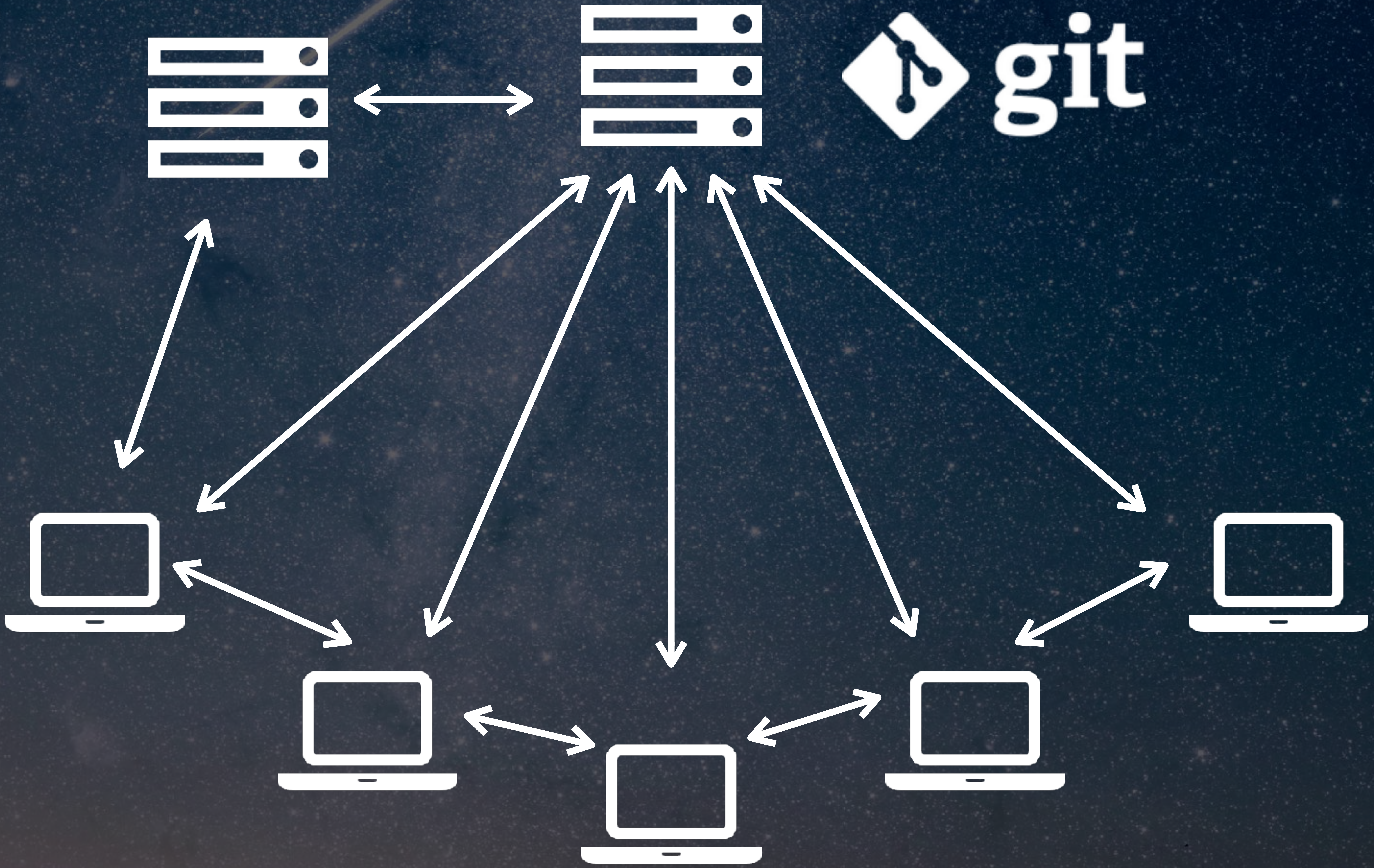


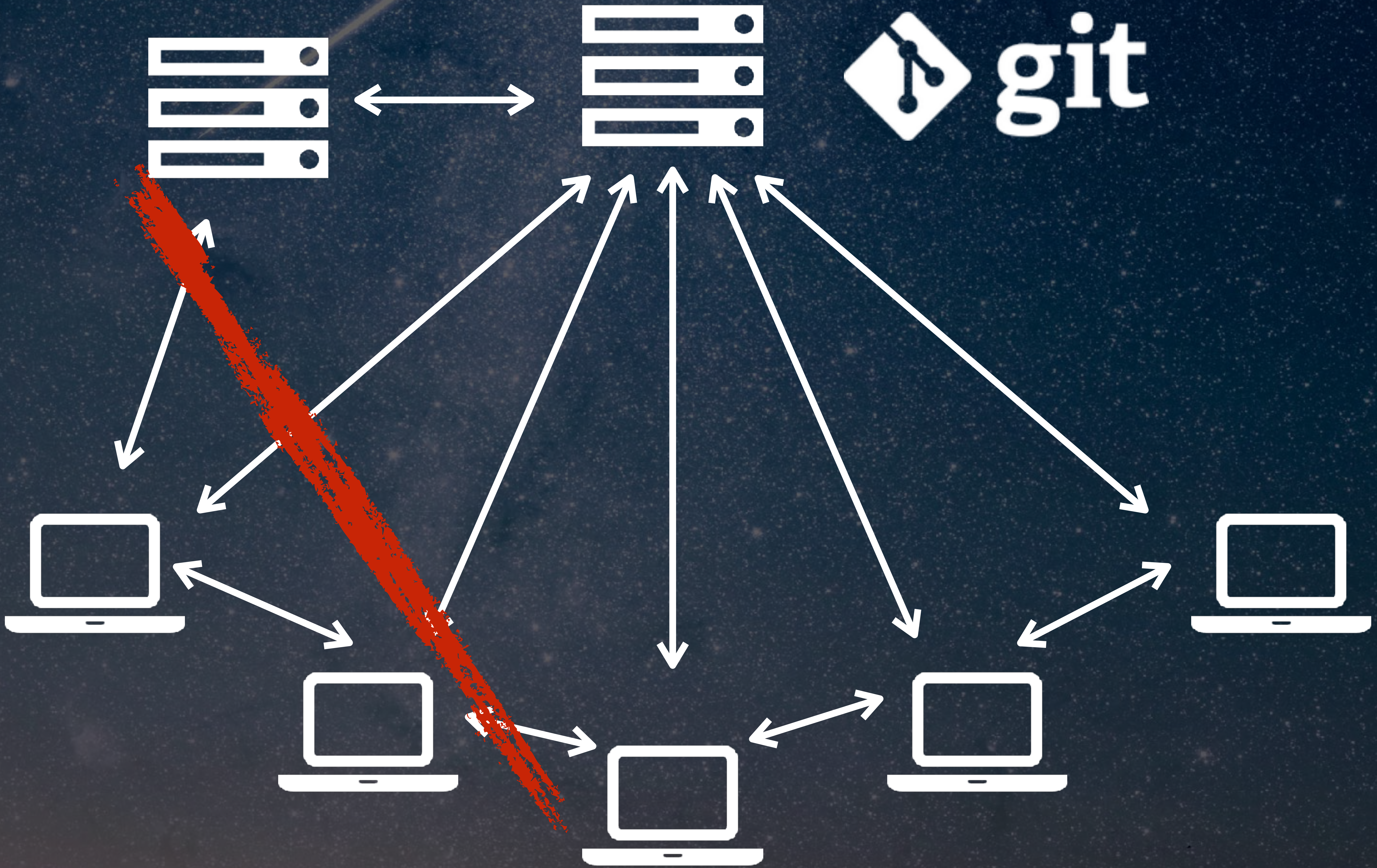
a.com	30.30.30.30
b.com	40.40.40.40
c.com	50.50.50.50

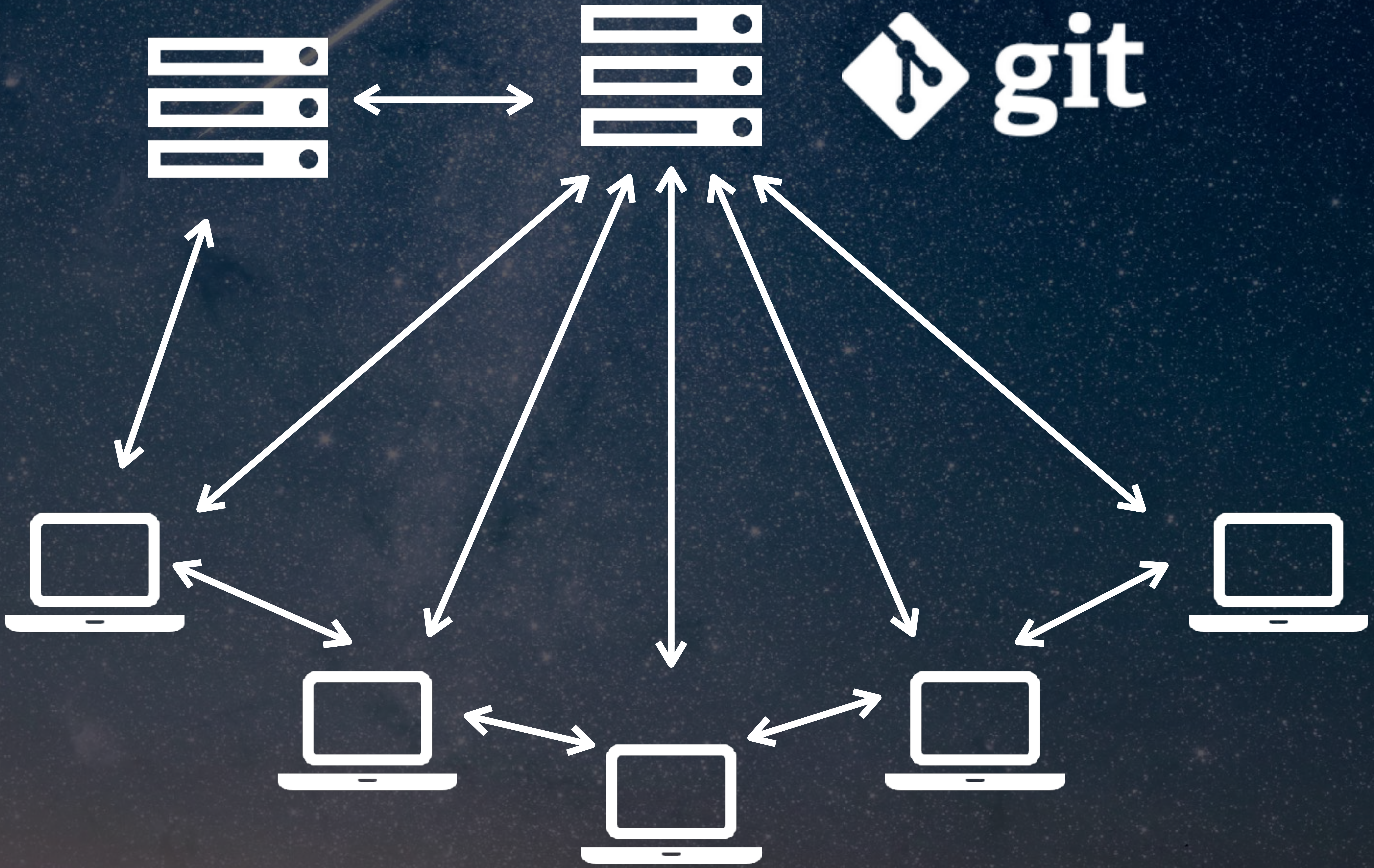

cvs/svn

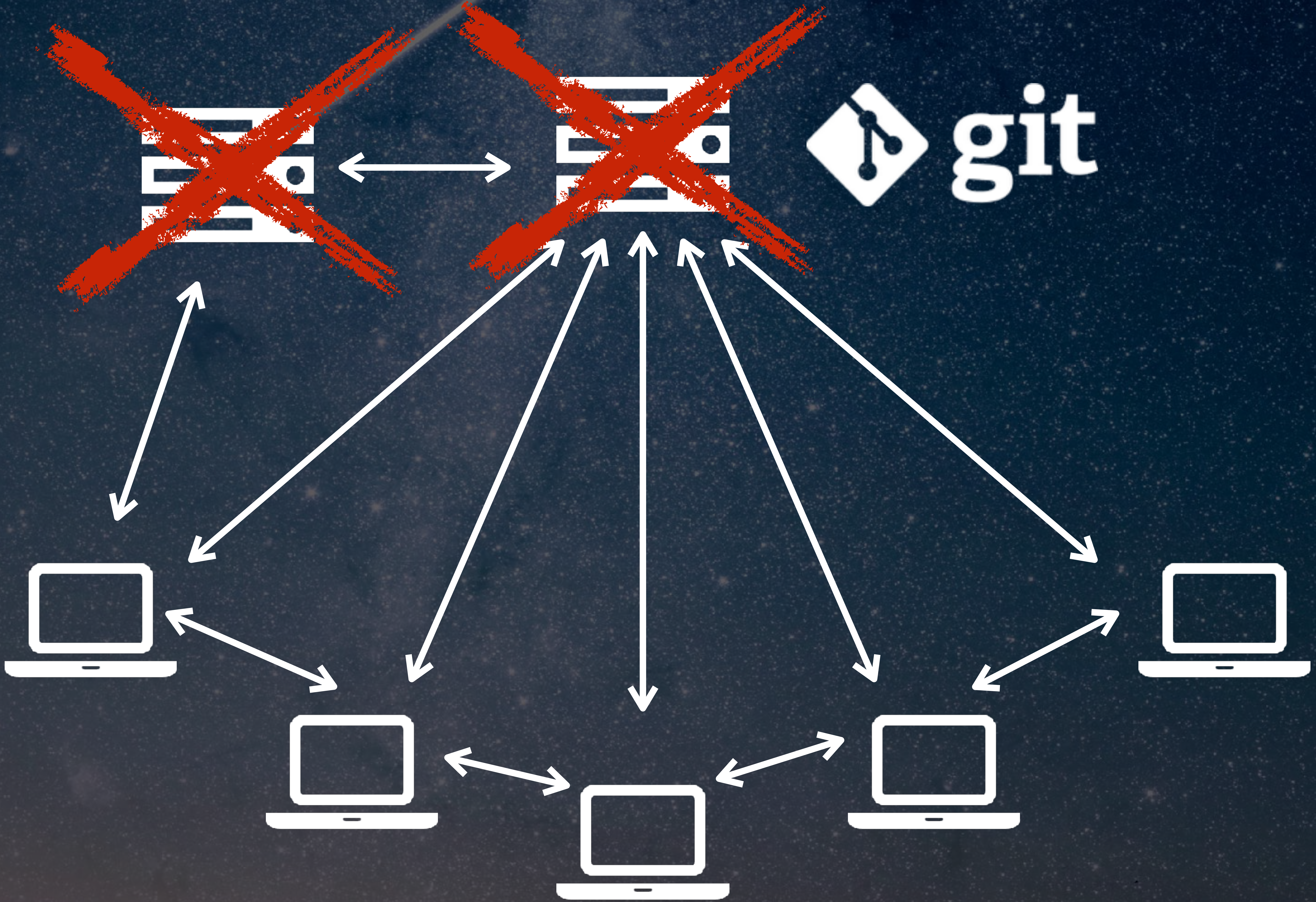

cvs/svn

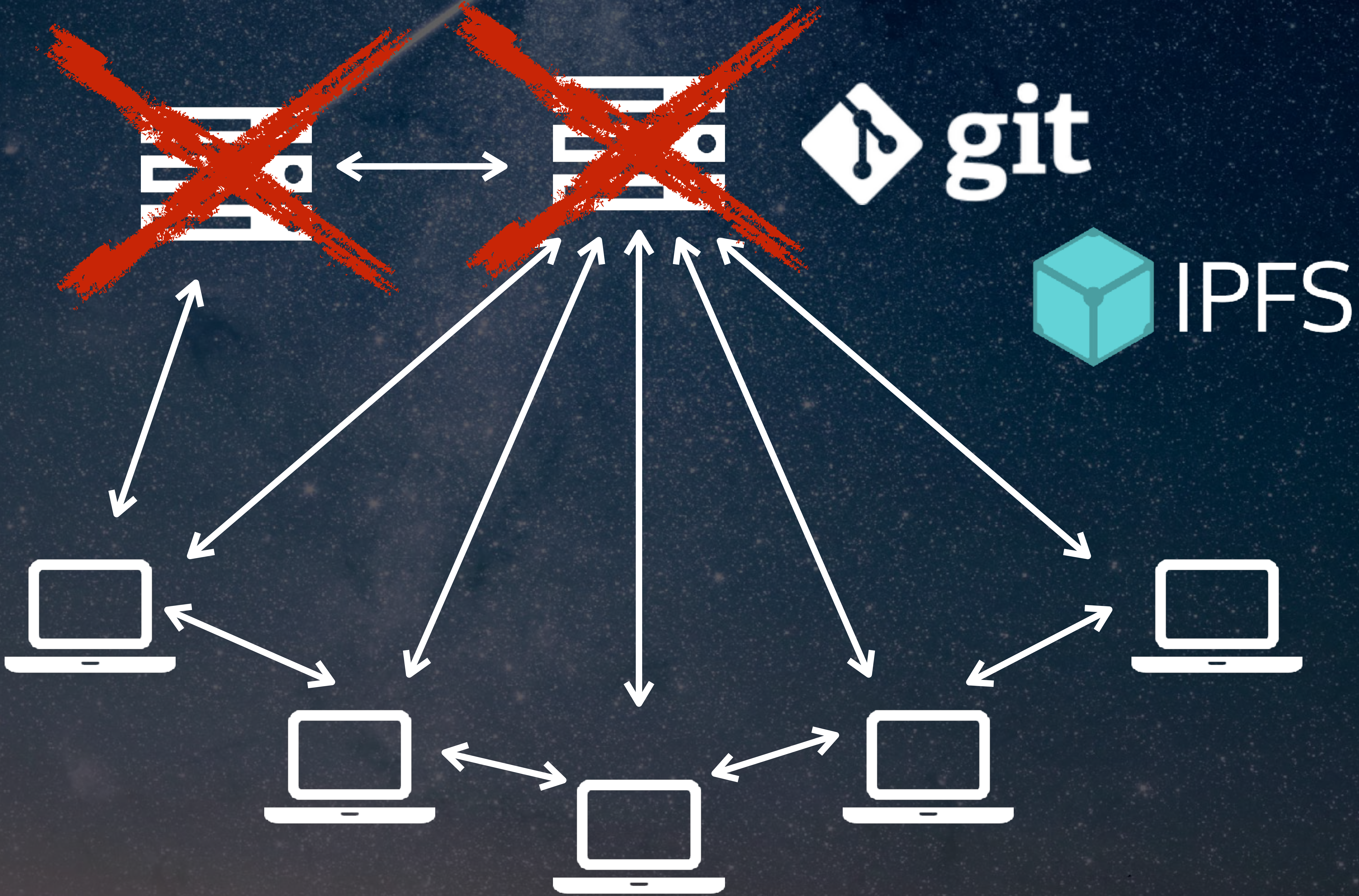

cvs/svn


git


git


git


git


git
IPFS

Goal

Create a
decentralised collaborative web-based real-time
Flipchart application

Materials

- Modern Browser
- JavaScript
- IPFS (js-ipfs)
- CRDT
- Code Editor
- Command line

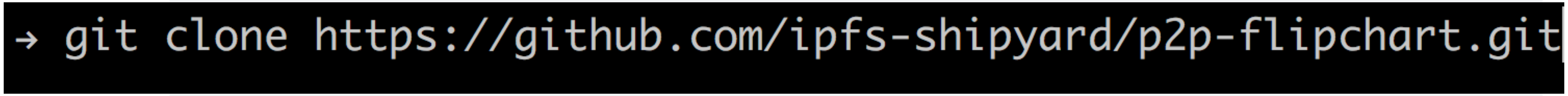
Setup:

Install Node.js

Setup:

Download code

- Through git:



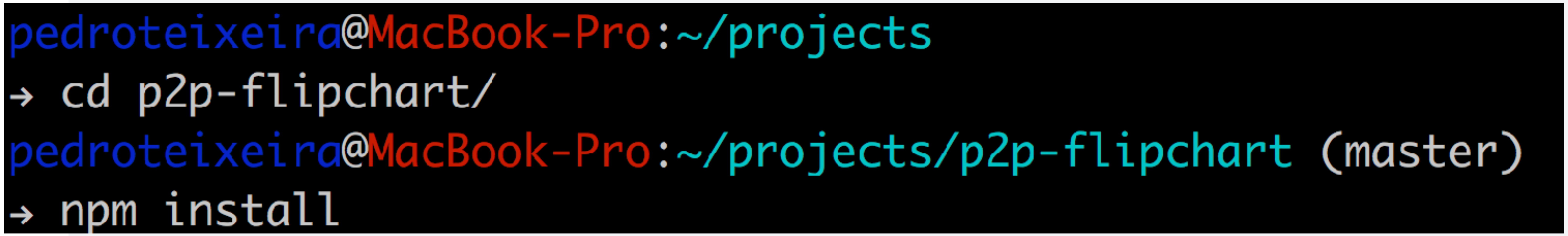
```
→ git clone https://github.com/ipfs-shipyard/p2p-flipchart.git
```

- or download zip file from

http://bit.ly/2xsCJfP

Setup:

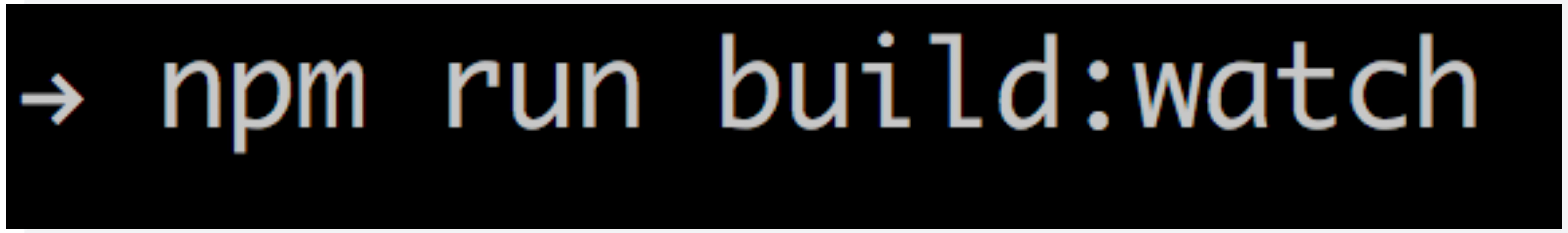
Install dependencies



```
pedroteixeira@MacBook-Pro:~/projects
→ cd p2p-flipchart/
pedroteixeira@MacBook-Pro:~/projects/p2p-flipchart (master)
→ npm install
```

Setup:

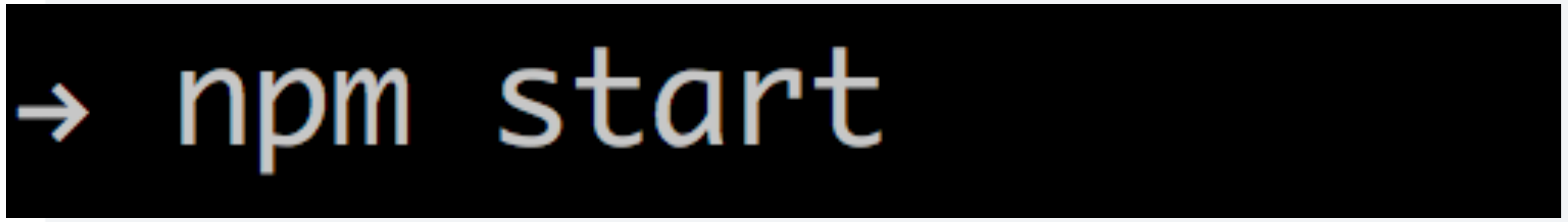
Build and watch



```
→ npm run build:watch
```

Setup:

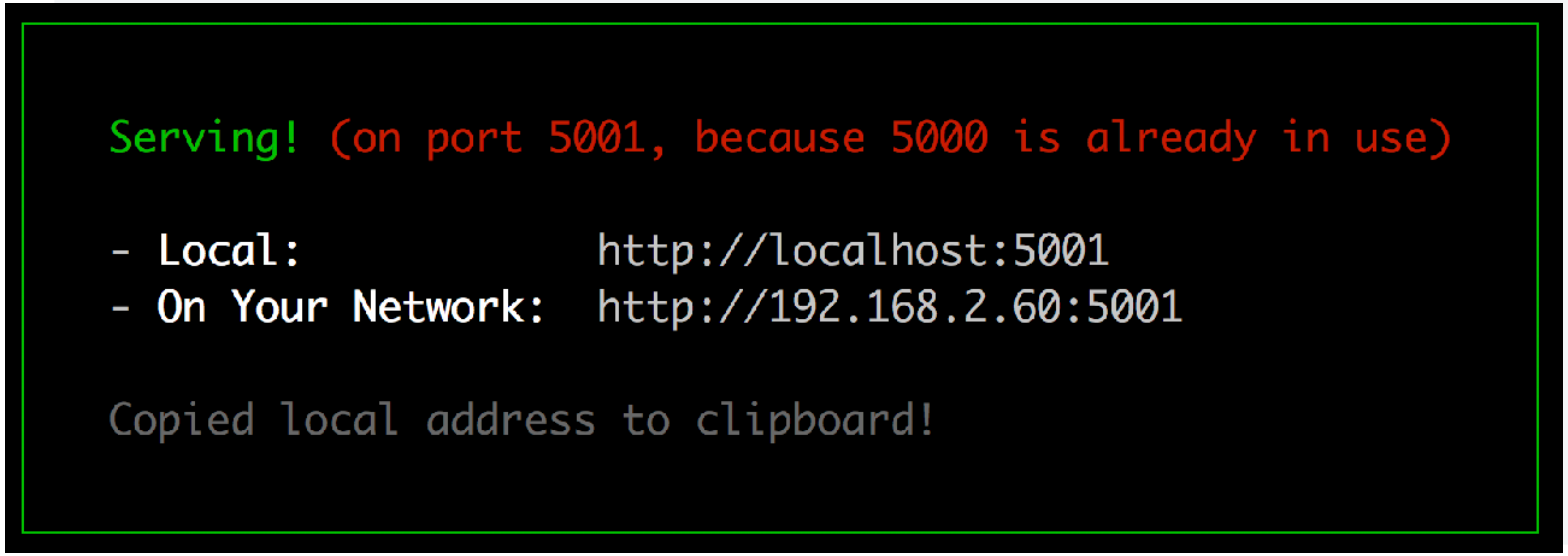
Start HTTP server



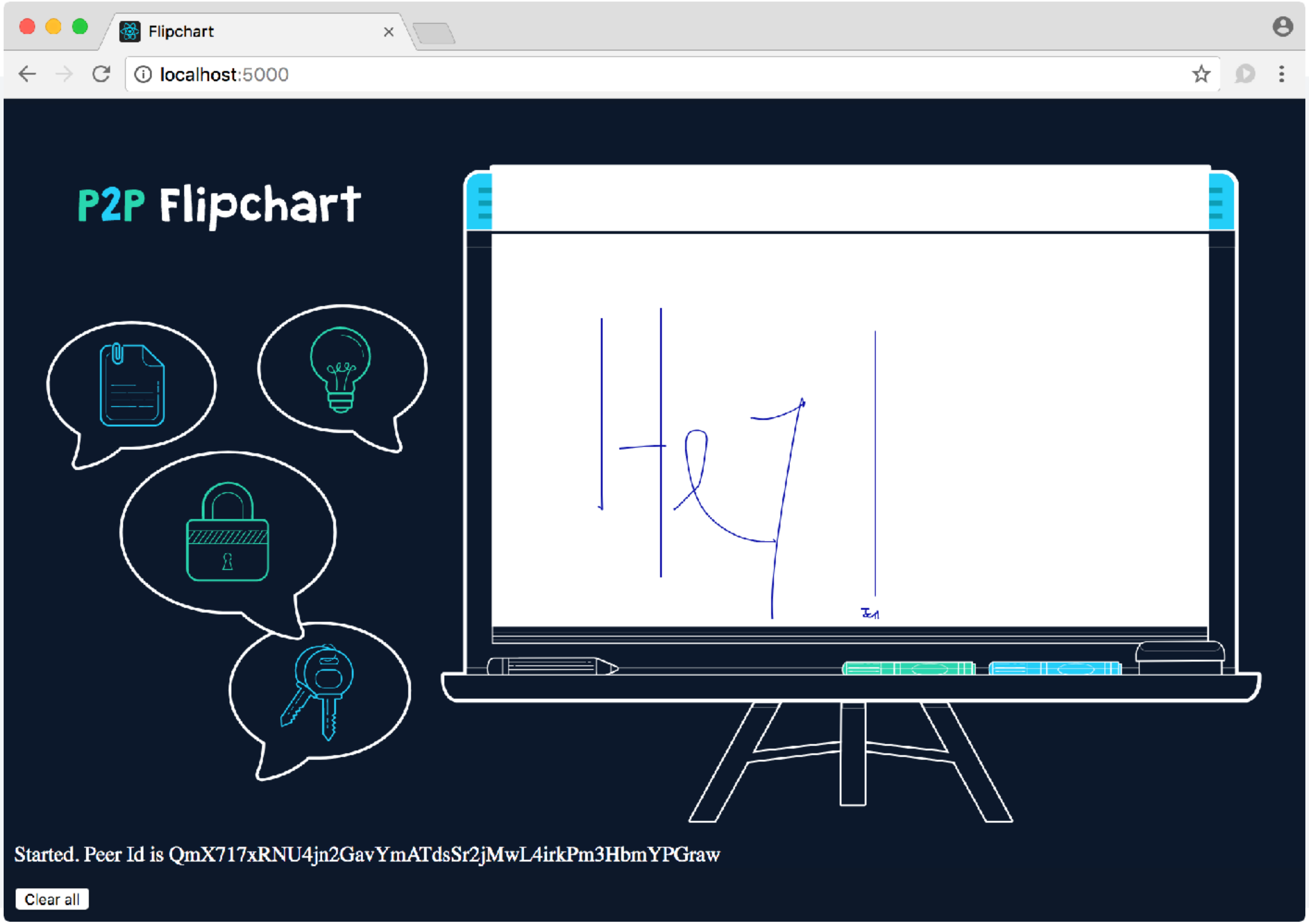
```
→ npm start
```

Setup:

Test flipchart app



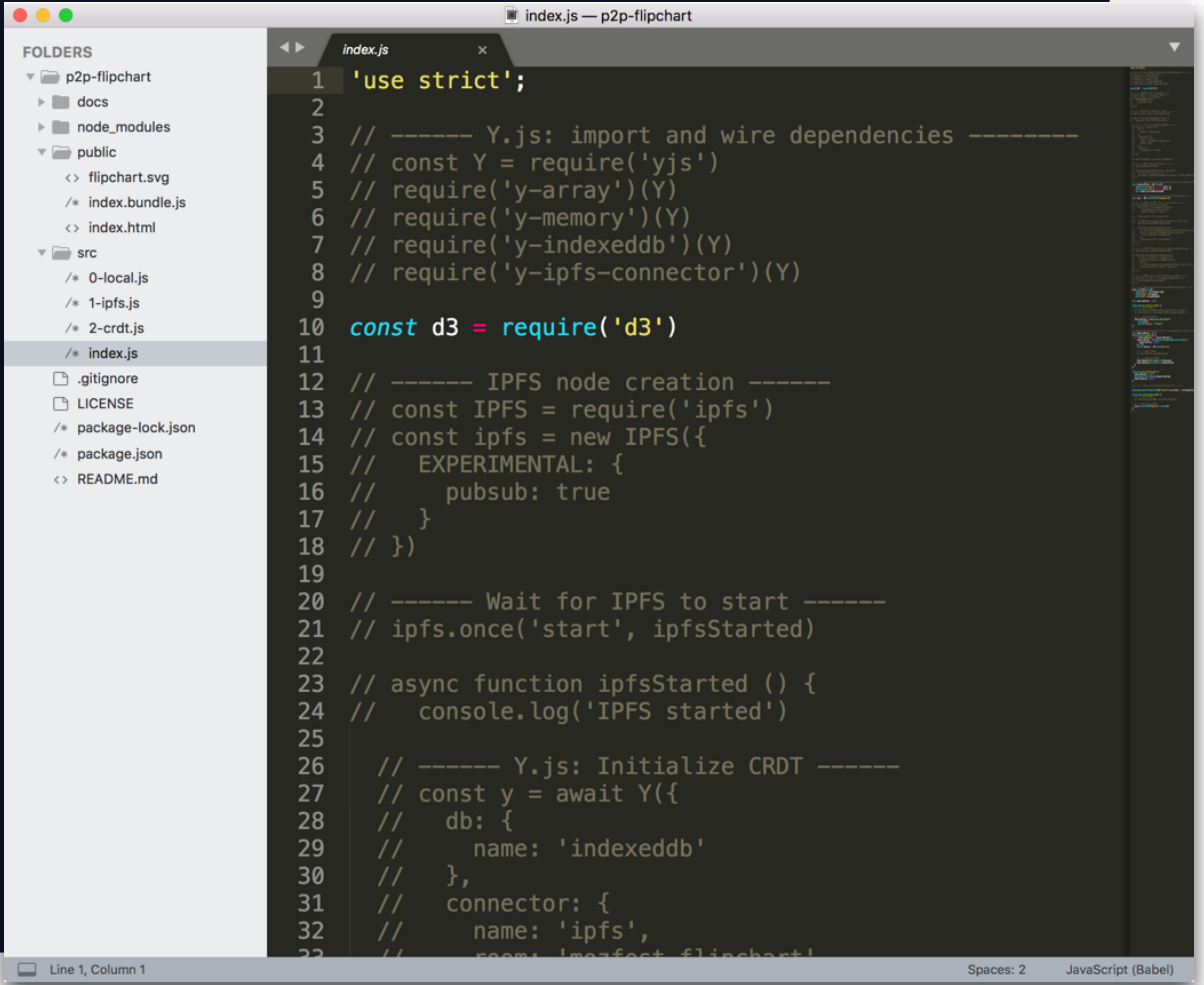
```
Serving! (on port 5001, because 5000 is already in use)

- Local:            http://localhost:5001
- On Your Network:  http://192.168.2.60:5001

Copied local address to clipboard!
```



Started. Peer Id is QmX717xRNU4jn2GavYmATdsSr2jMwL4irkPm3HbmYPGraw

The source code

Open a code editor and take a peek at
`src/index.js`



index.js — p2p-flipchart

```
FOLDERS
▾ p2p-flipchart
  ▸ docs
  ▸ node_modules
  ▾ public
      flipchart.svg
      index.bundle.js
      index.html
  ▾ src
      0-local.js
      1-ipfs.js
      2-crdt.js
      index.js
    .gitignore
    LICENSE
    package-lock.json
    package.json
    README.md
```

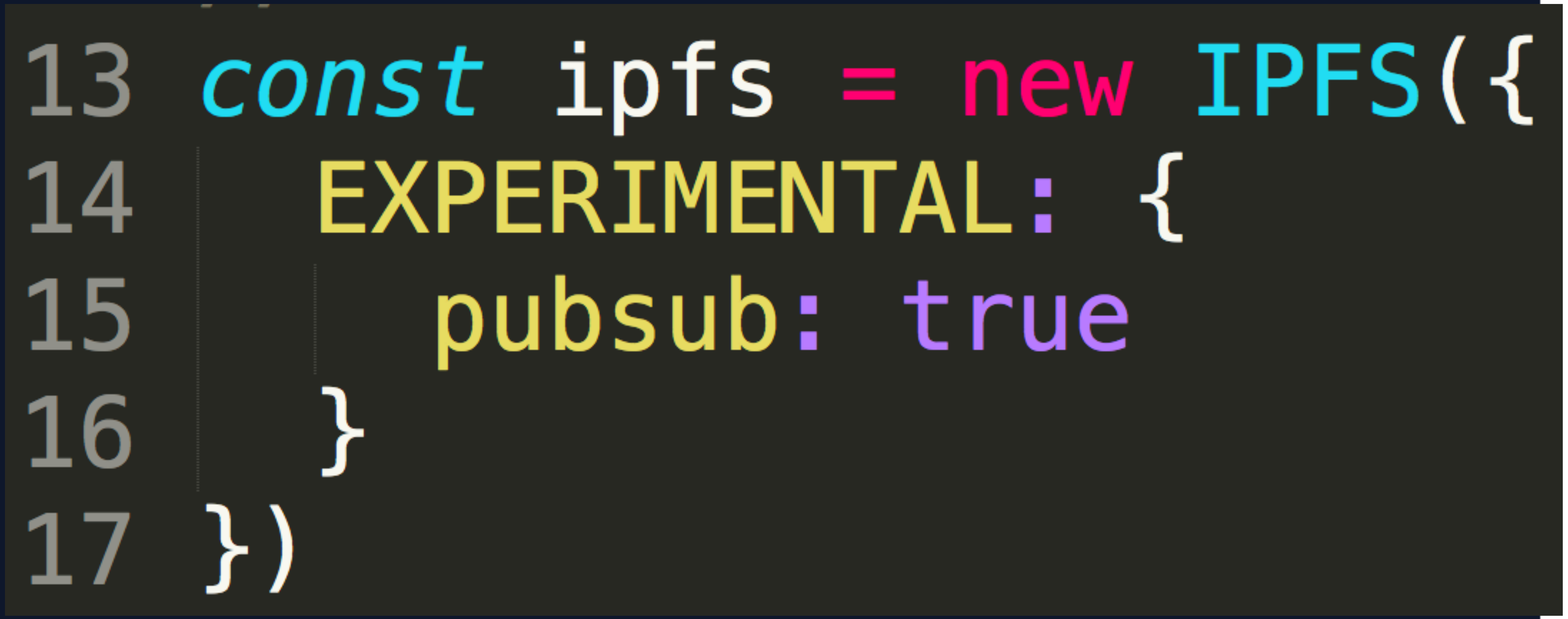
index.js

```javascript
'use strict';

// ------ Y.js: import and wire dependencies --------
// const Y = require('yjs')
// require('y-array')(Y)
// require('y-memory')(Y)
// require('y-indexeddb')(Y)
// require('y-ipfs-connector')(Y)

const d3 = require('d3')

// ------ IPFS node creation ------
// const IPFS = require('ipfs')
// const ipfs = new IPFS({
//   EXPERIMENTAL: {
//     pubsub: true
//   }
// })

// ------ Wait for IPFS to start ------
// ipfs.once('start', ipfsStarted)

// async function ipfsStarted () {
//   console.log('IPFS started')

  // ------ Y.js: Initialize CRDT ------
  // const y = await Y({
  //   db: {
  //     name: 'indexeddb'
  //   },
  //   connector: {
  //     name: 'ipfs',
  //       room: 'mozfest-flipchart',
```

Line 1, Column 1   Spaces: 2   JavaScript (Babel)

Add 1 IPFS

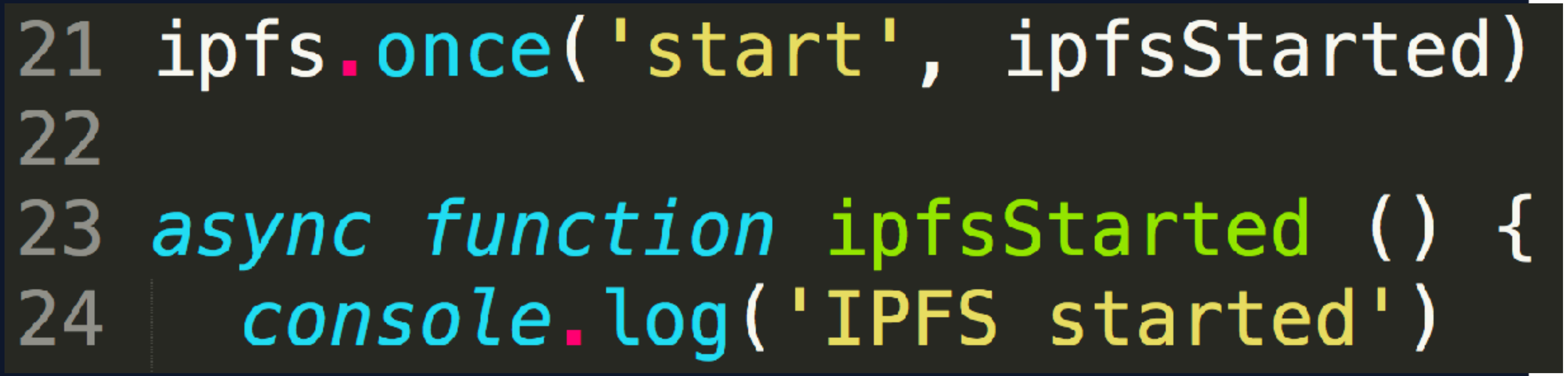
Let’s create an IPFS node inside the browser app.



```
const ipfs = new IPFS({
  EXPERIMENTAL: {
    pubsub: true
  }
})
```

Wait for IPFS to start

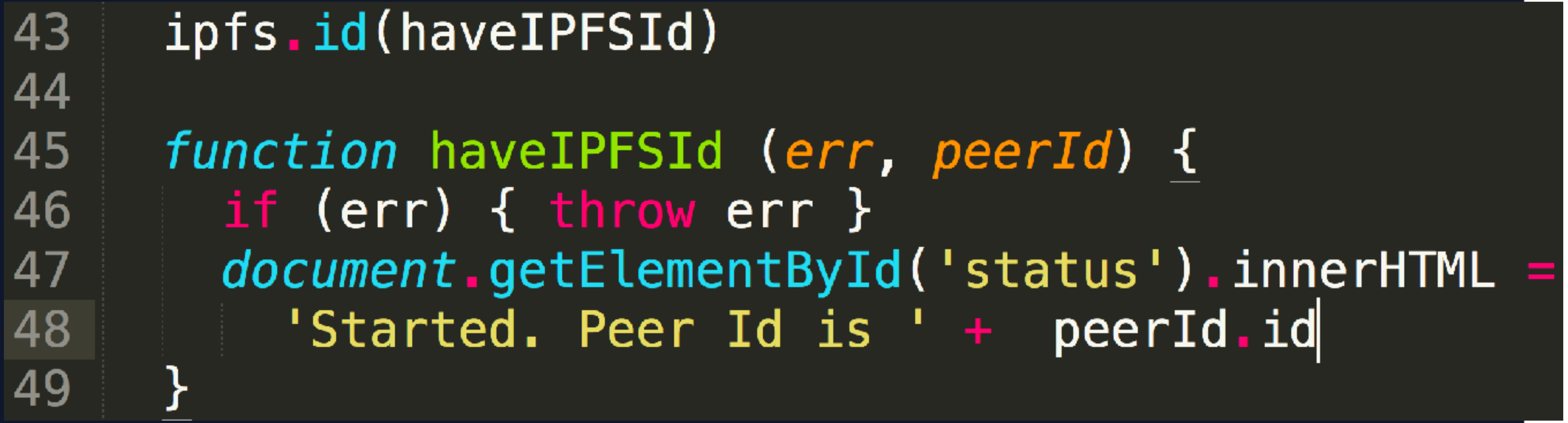
…by handling the IPFS "start" event.



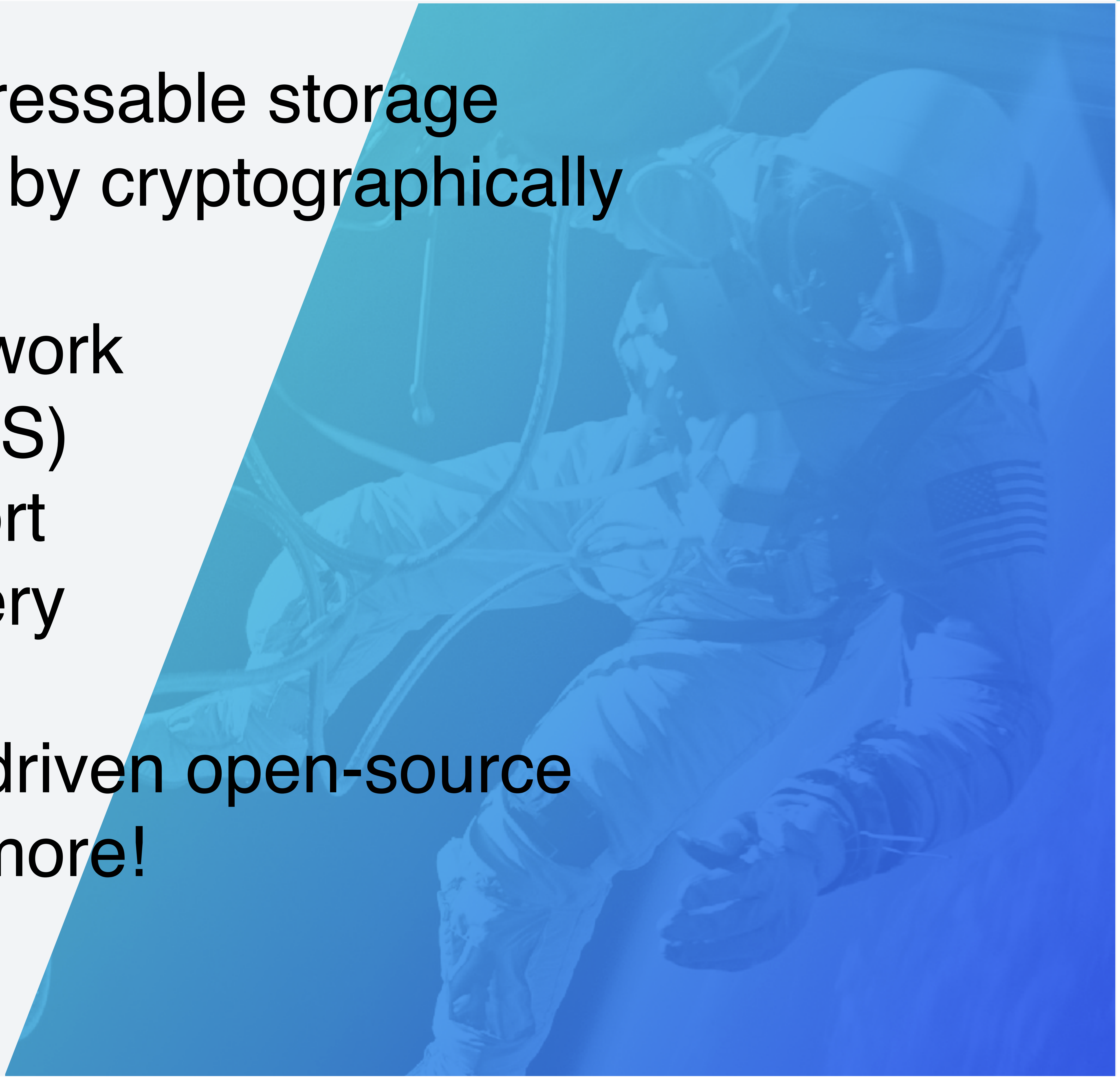
```
ipfs.once('start', ipfsStarted)

async function ipfsStarted () {
  console.log('IPFS started')
```

Print IPFS status

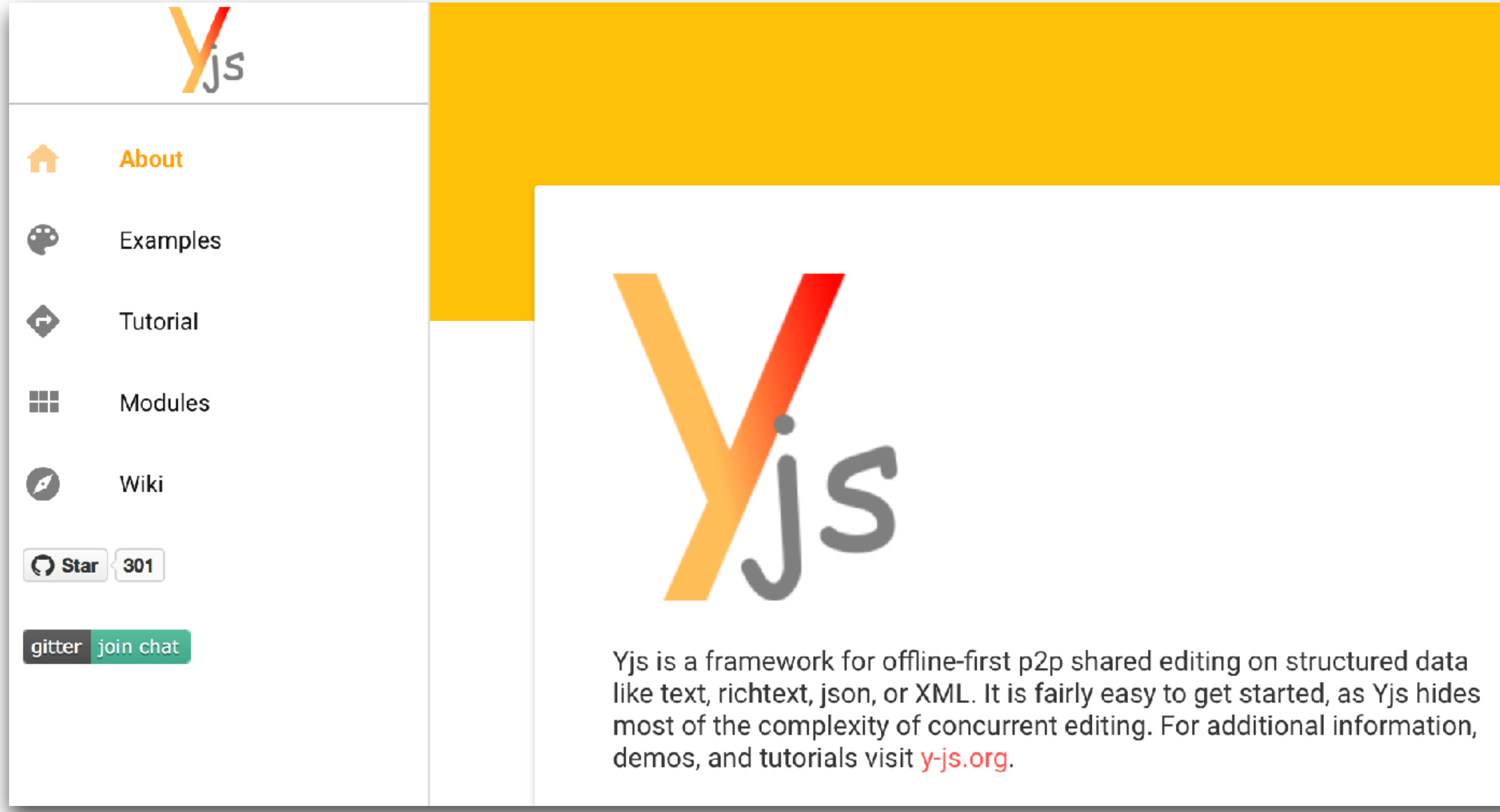
and IPFS Peer ID



```
ipfs.id(haveIPFSId)

function haveIPFSId (err, peerId) {
  if (err) { throw err }
  document.getElementById('status').innerHTML =
    'Started. Peer Id is ' +  peerId.id
}
```

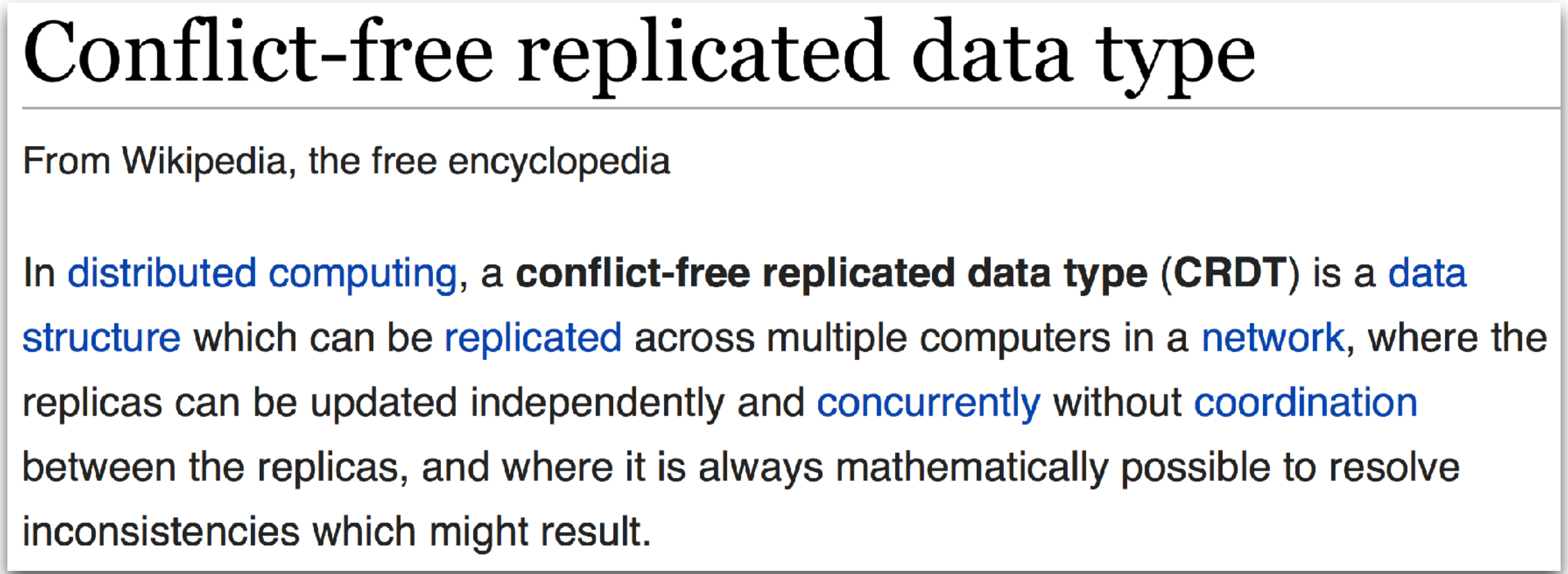


- Content-addressable storage
- DAG formed by cryptographically secure links
- Pub-sub network
- Naming (IPNS)
- Multi-transport
- Multi-discovery
- Multi-*
- Community-driven open-source
- Go, JS and more!
- Future-proof



Yjs is a framework for offline-first p2p shared editing on structured data like text, richtext, json, or XML. It is fairly easy to get started, as Yjs hides most of the complexity of concurrent editing. For additional information, demos, and tutorials visit y-js.org.

Offline-first, real-time, shared editing for data structures


Conflict-free replicated data type

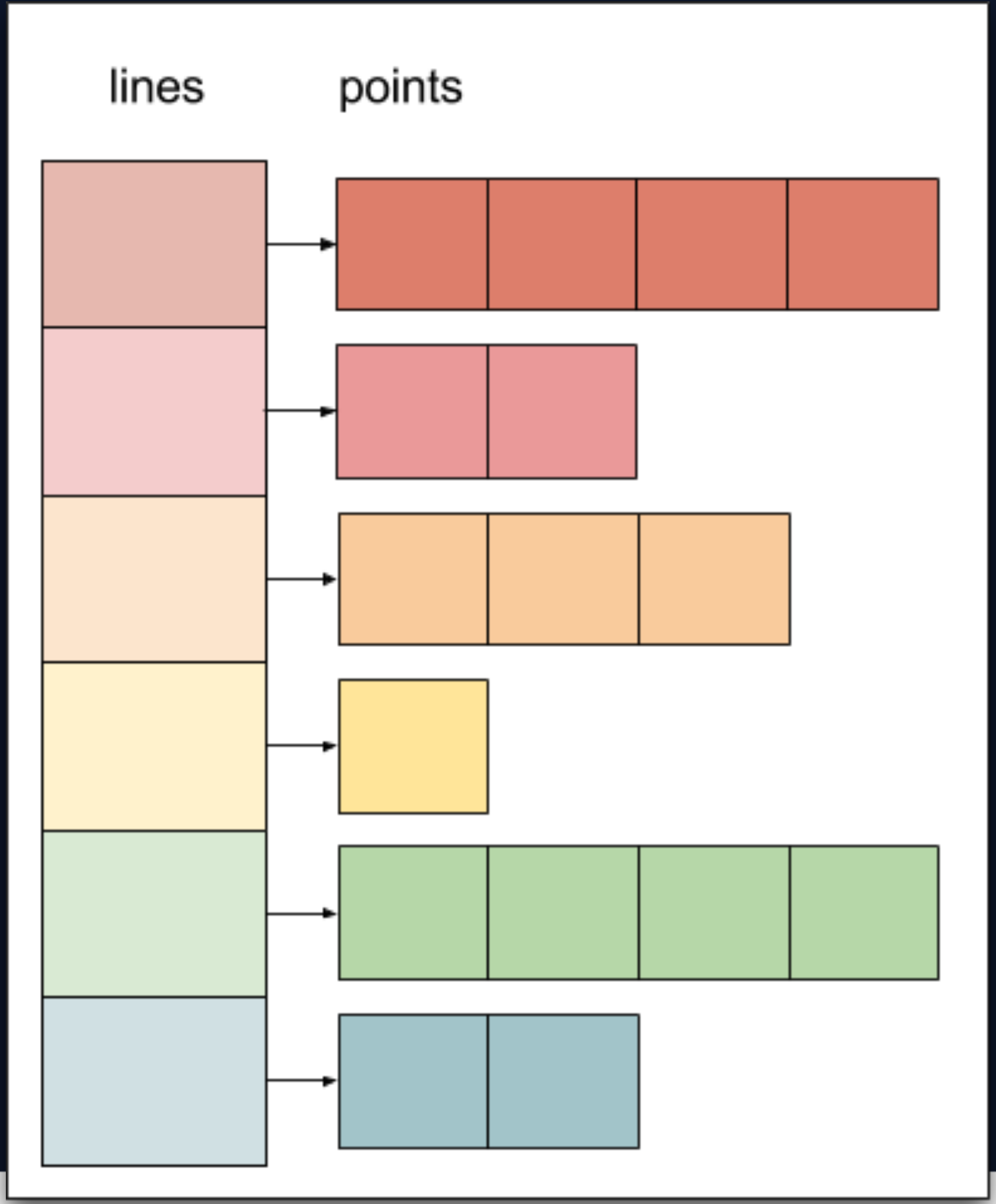
From Wikipedia, the free encyclopedia

In distributed computing, a **conflict-free replicated data type** (**CRDT**) is a data structure which can be replicated across multiple computers in a network, where the replicas can be updated independently and concurrently without coordination between the replicas, and where it is always mathematically possible to resolve inconsistencies which might result.

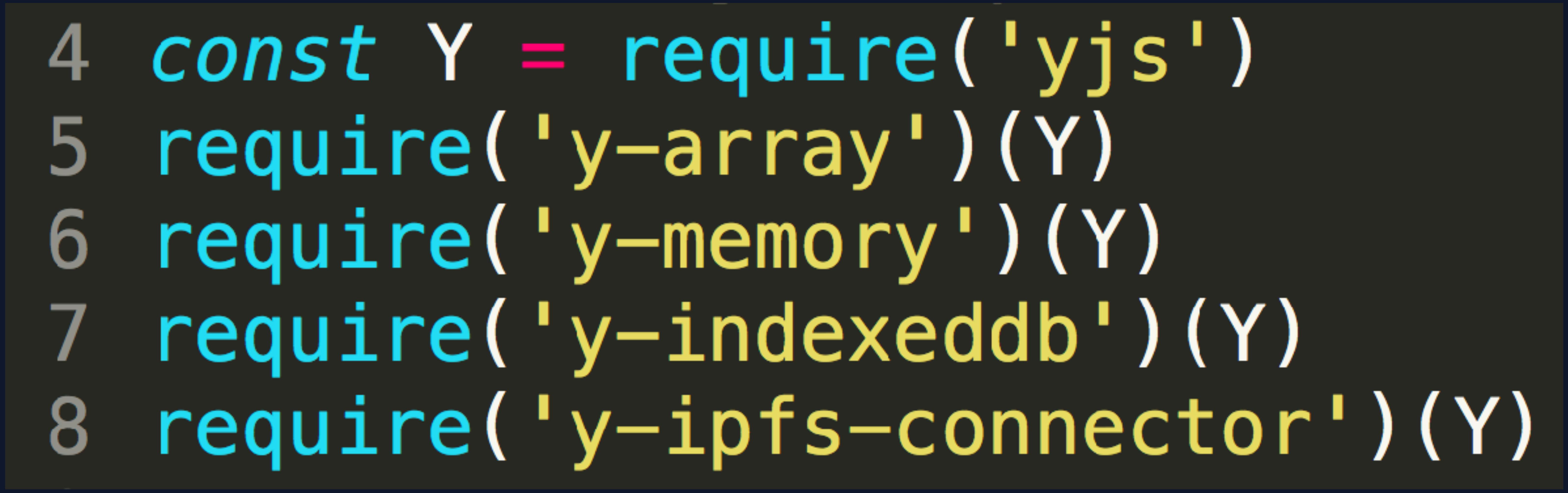
Add 1 CRDT

Let’s replace direct manipulation of the line array with an array of lines.

Each line is (still) an array of points.



Y.js step 1

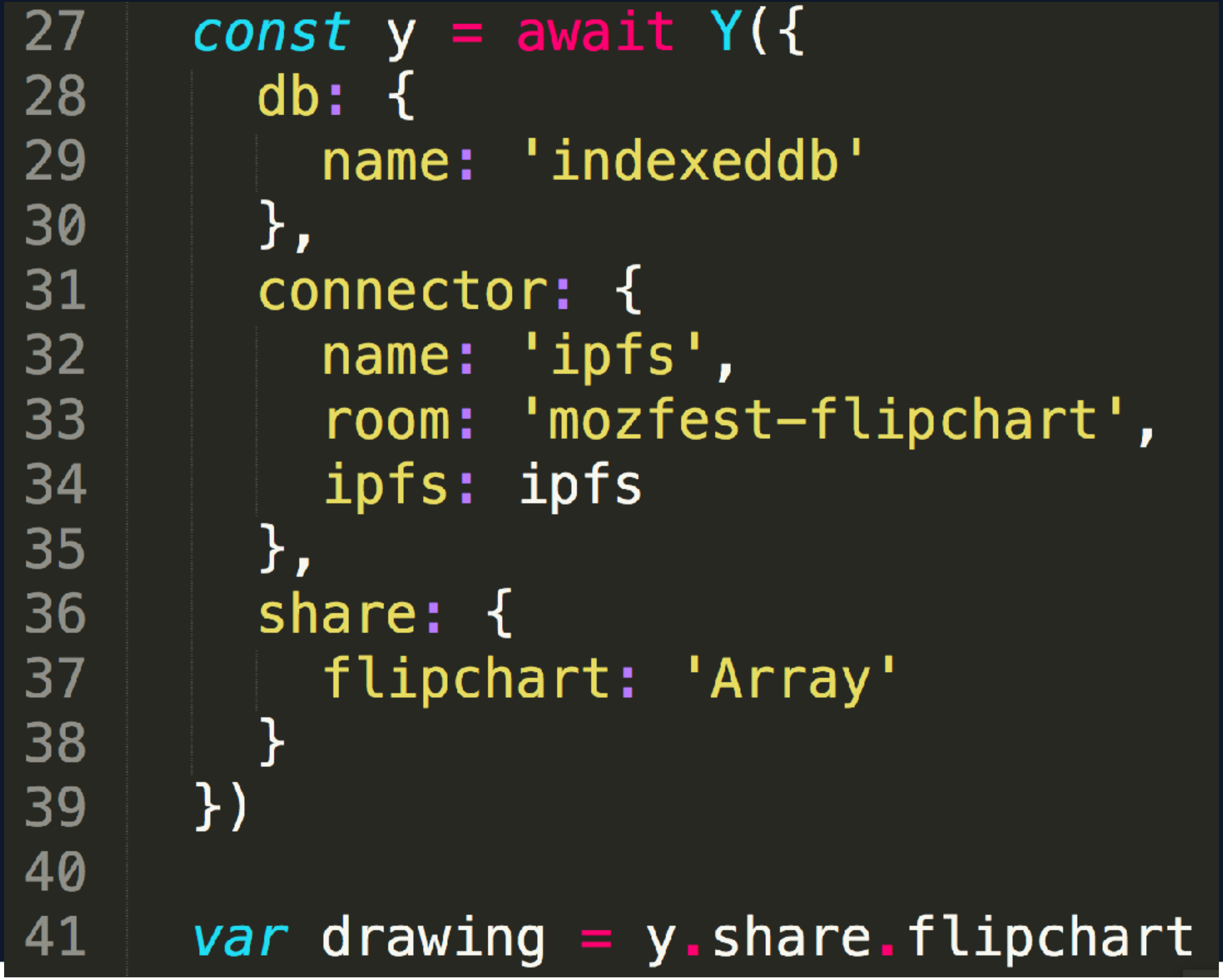


```
const Y = require('yjs')
require('y-array')(Y)
require('y-memory')(Y)
require('y-indexeddb')(Y)
require('y-ipfs-connector')(Y)
```

Import Y.js dependencies

Y.js step 2

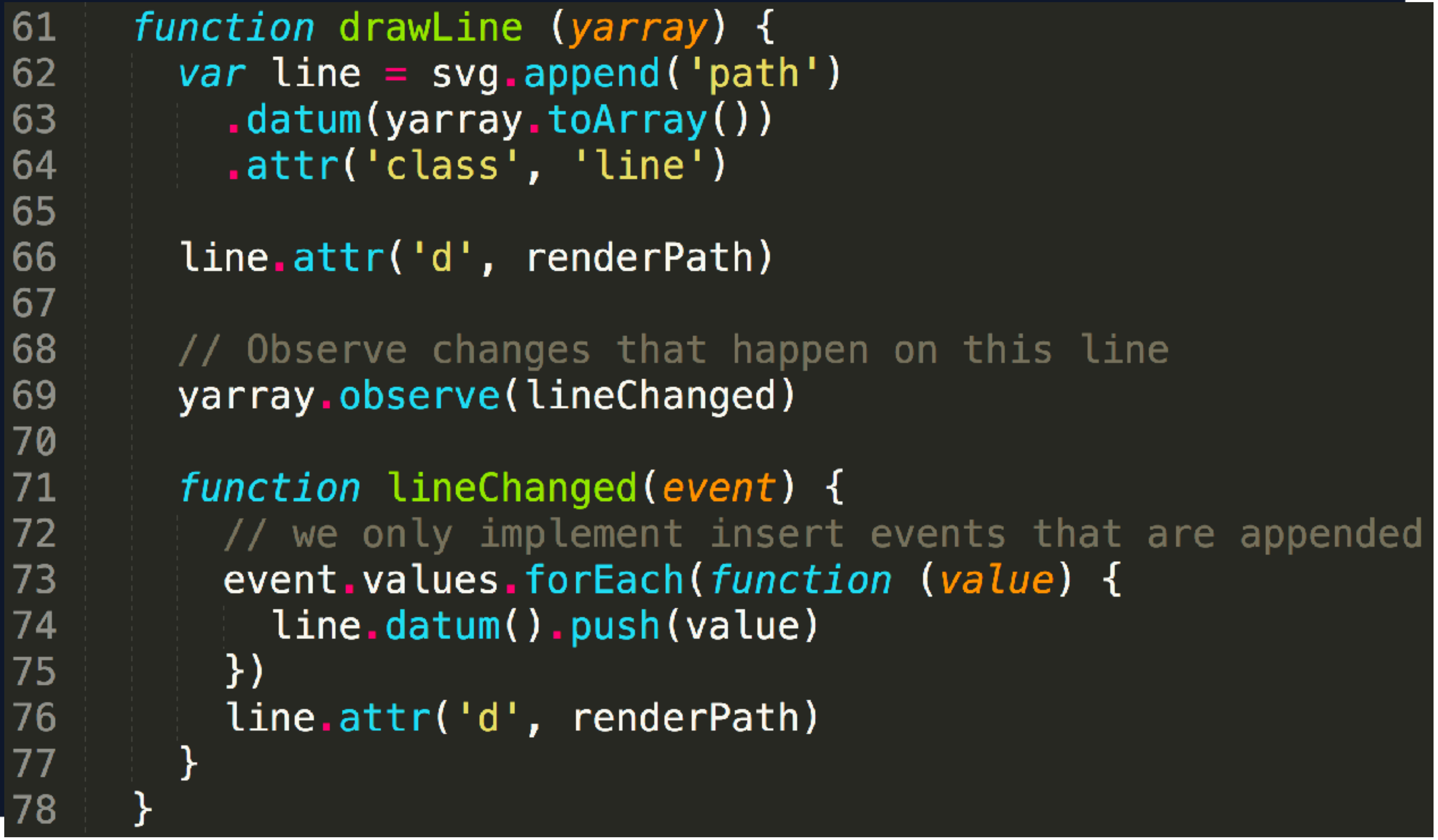
Initialise Y.js



```
const y = await Y({
  db: {
    name: 'indexeddb'
  },
  connector: {
    name: 'ipfs',
    room: 'mozfest-flipchart',
    ipfs: ipfs
  },
  share: {
    flipchart: 'Array'
  }
})

var drawing = y.share.flipchart
```

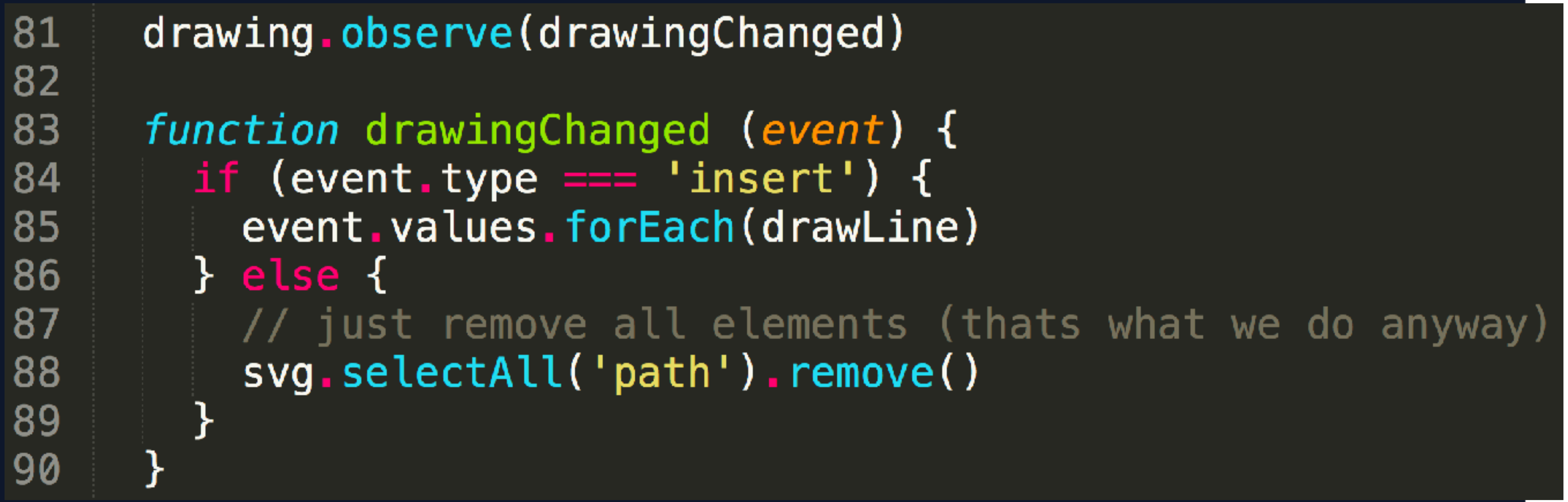
Y.js step 3

Routine to draw a line from a Y.js array



```javascript
function drawLine (yarray) {
  var line = svg.append('path')
    .datum(yarray.toArray())
    .attr('class', 'line')

  line.attr('d', renderPath)

  // Observe changes that happen on this line
  yarray.observe(lineChanged)

  function lineChanged(event) {
    // we only implement insert events that are appended
    event.values.forEach(function (value) {
      line.datum().push(value)
    })
    line.attr('d', renderPath)
  }
}
```

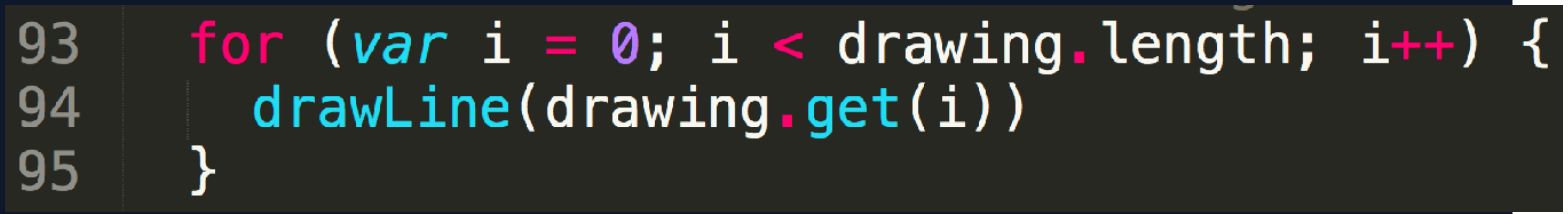
Y.js step 4



```
  drawing.observe(drawingChanged)

  function drawingChanged (event) {
    if (event.type === 'insert') {
      event.values.forEach(drawLine)
    } else {
      // just remove all elements (thats what we do anyway)
      svg.selectAll('path').remove()
    }
  }
```

Listen for changes in drawing and act.

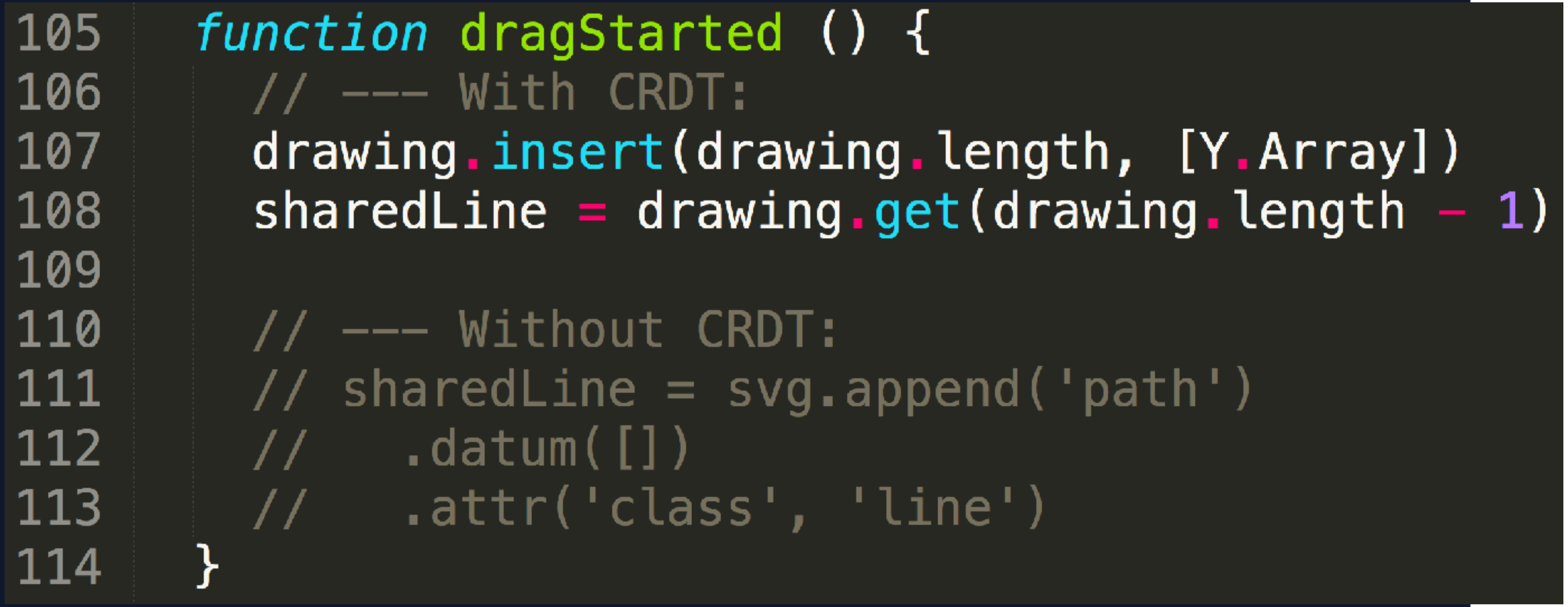
Y.js step 5



```
for (var i = 0; i < drawing.length; i++) {
  drawLine(drawing.get(i))
}
```

Draw pre-existing lines

Y.js step 6

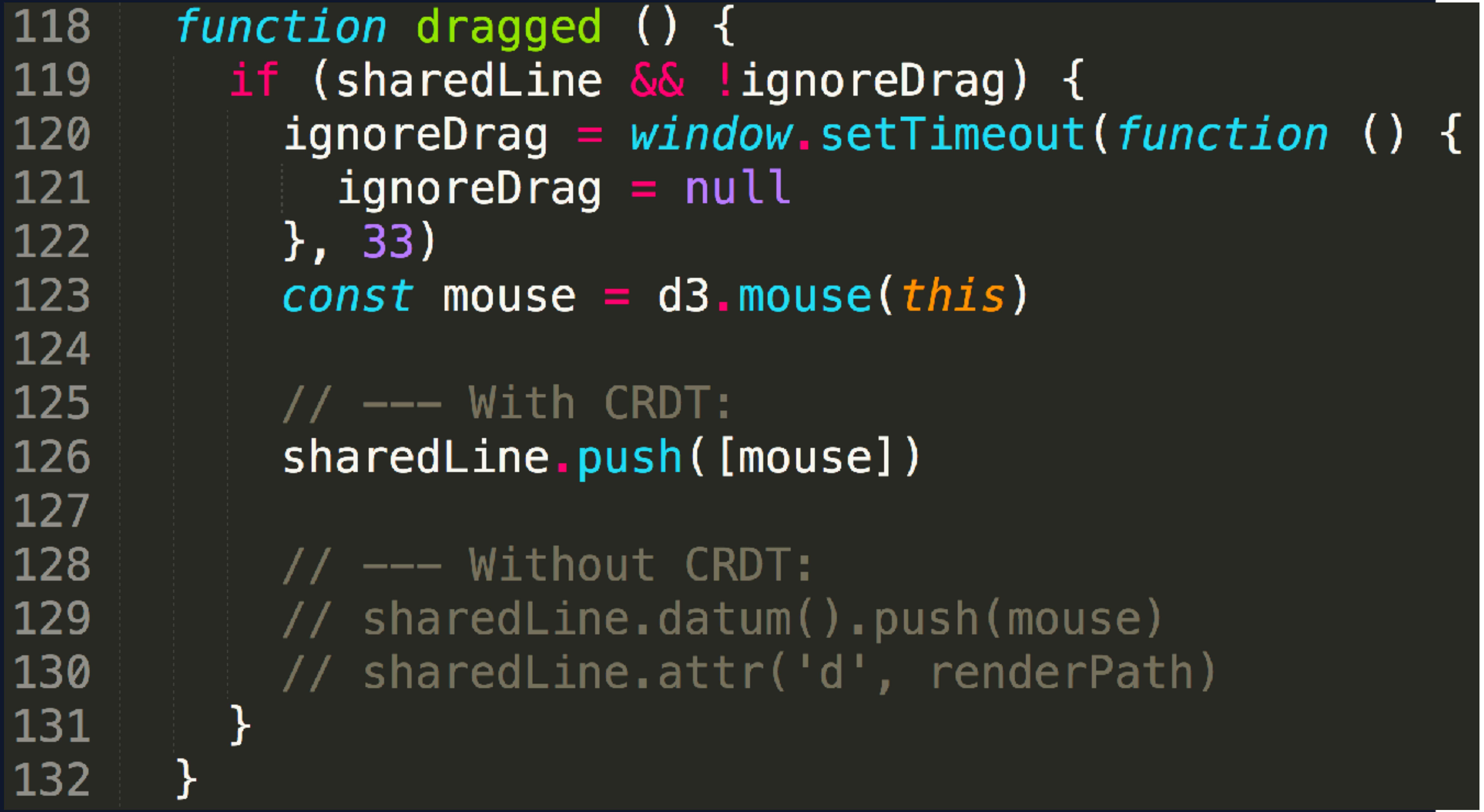


```javascript
function dragStarted () {
  // --- With CRDT:
  drawing.insert(drawing.length, [Y.Array])
  sharedLine = drawing.get(drawing.length - 1)

  // --- Without CRDT:
  // sharedLine = svg.append('path')
  //   .datum([])
  //   .attr('class', 'line')
}
```

Create new line on drag

Y.js step 7

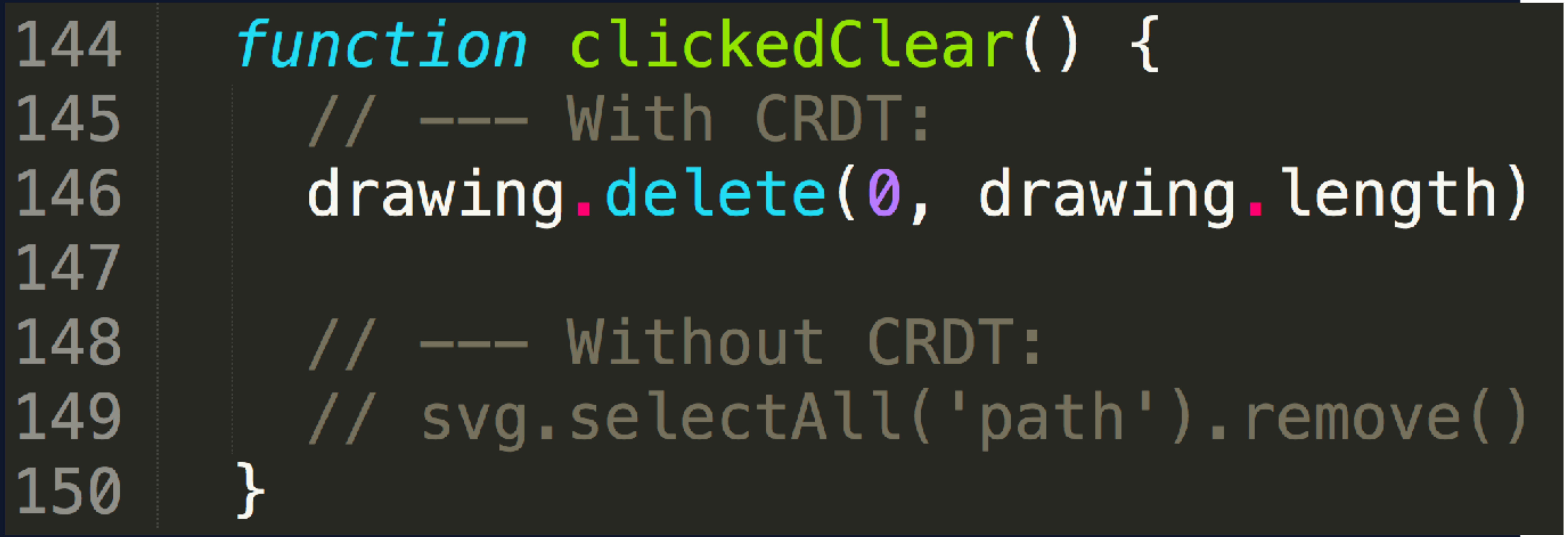
When user drags, add points to current line



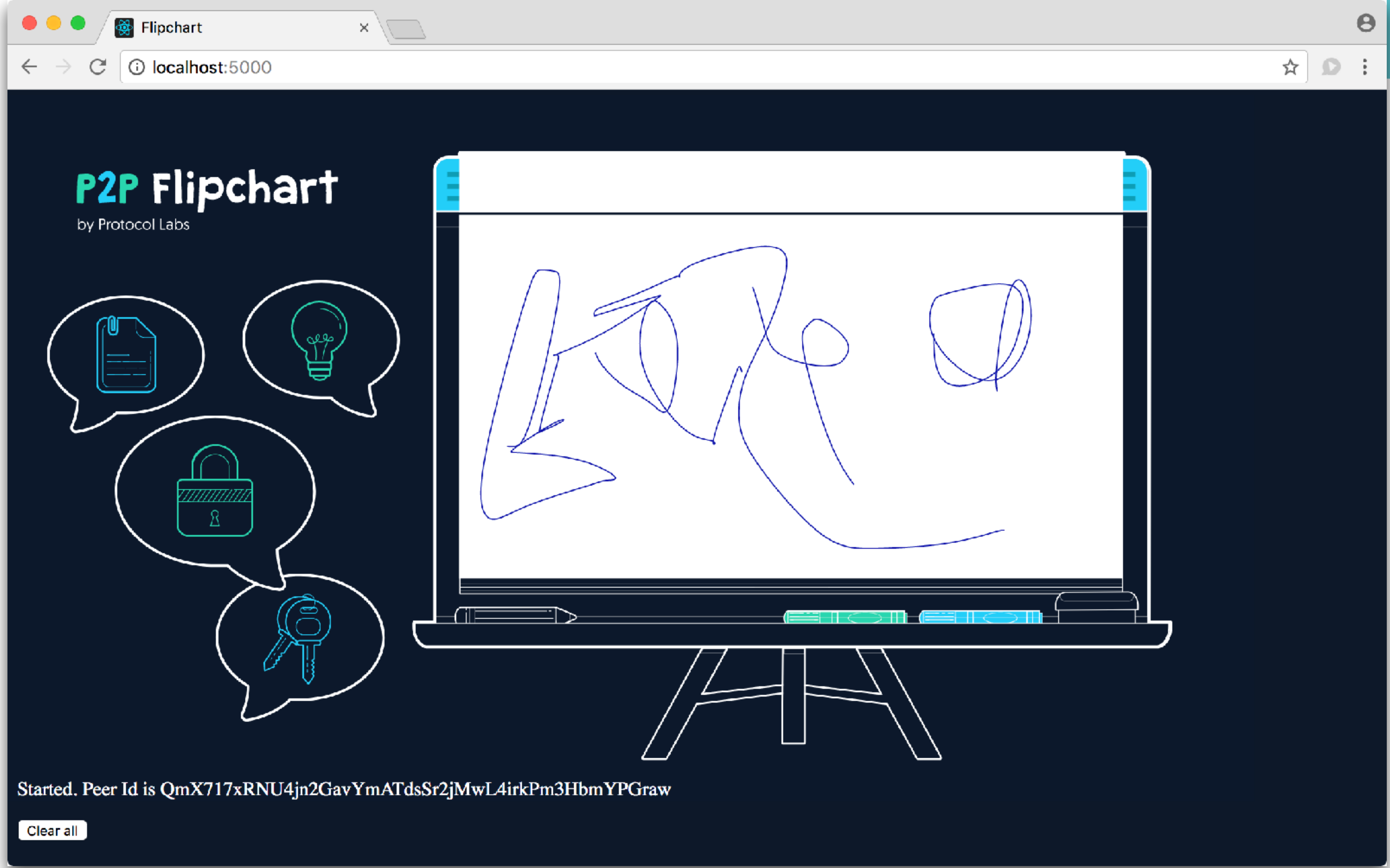
```
function dragged () {
  if (sharedLine && !ignoreDrag) {
    ignoreDrag = window.setTimeout(function () {
      ignoreDrag = null
    }, 33)
    const mouse = d3.mouse(this)

    // --- With CRDT:
    sharedLine.push([mouse])

    // --- Without CRDT:
    // sharedLine.datum().push(mouse)
    // sharedLine.attr('d', renderPath)
  }
}
```

Y.js step 8

When user clears, remove all lines from CRDT

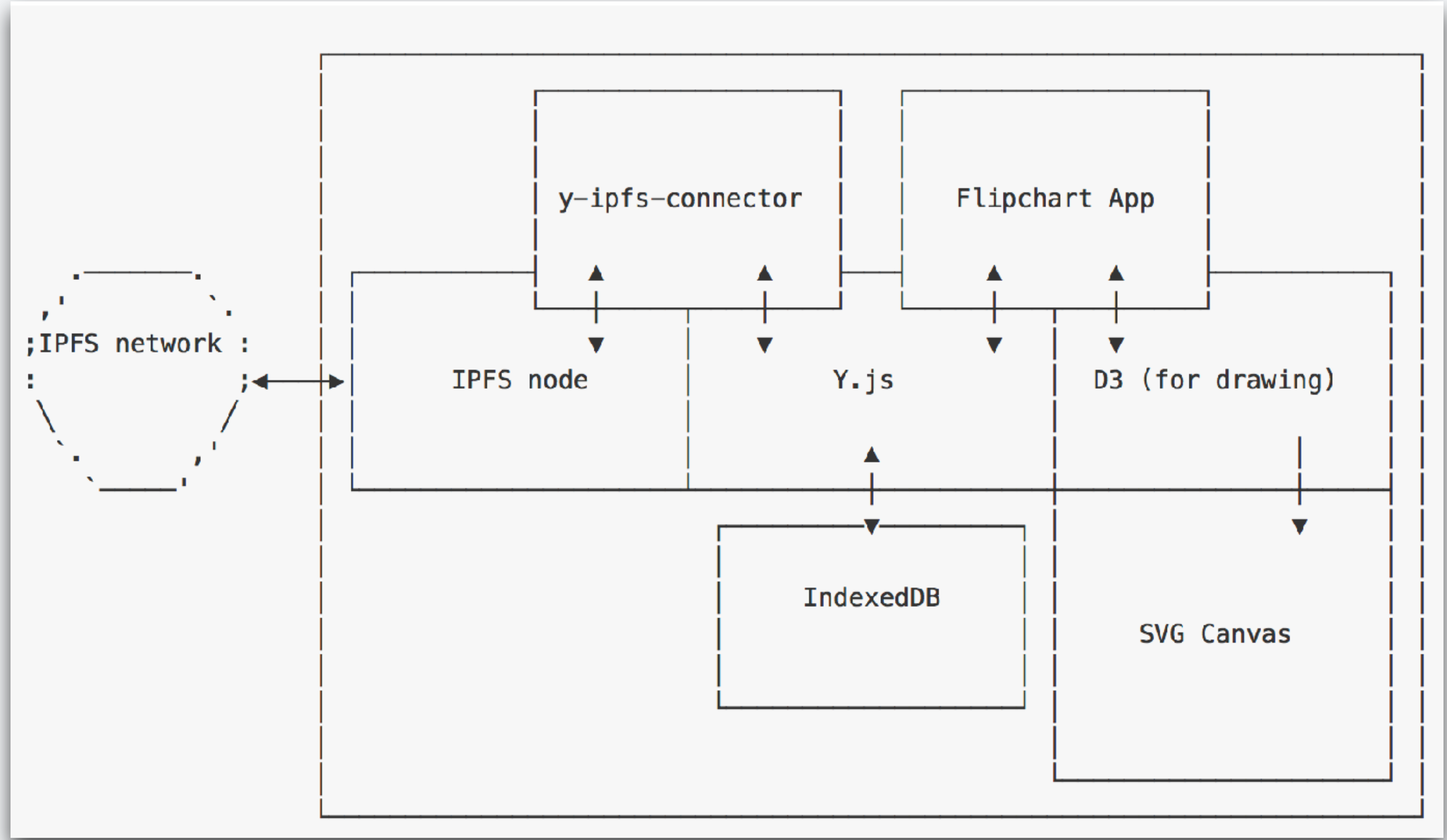


```
function clickedClear() {
  // --- With CRDT:
  drawing.delete(0, drawing.length)

  // --- Without CRDT:
  // svg.selectAll('path').remove()
}
```



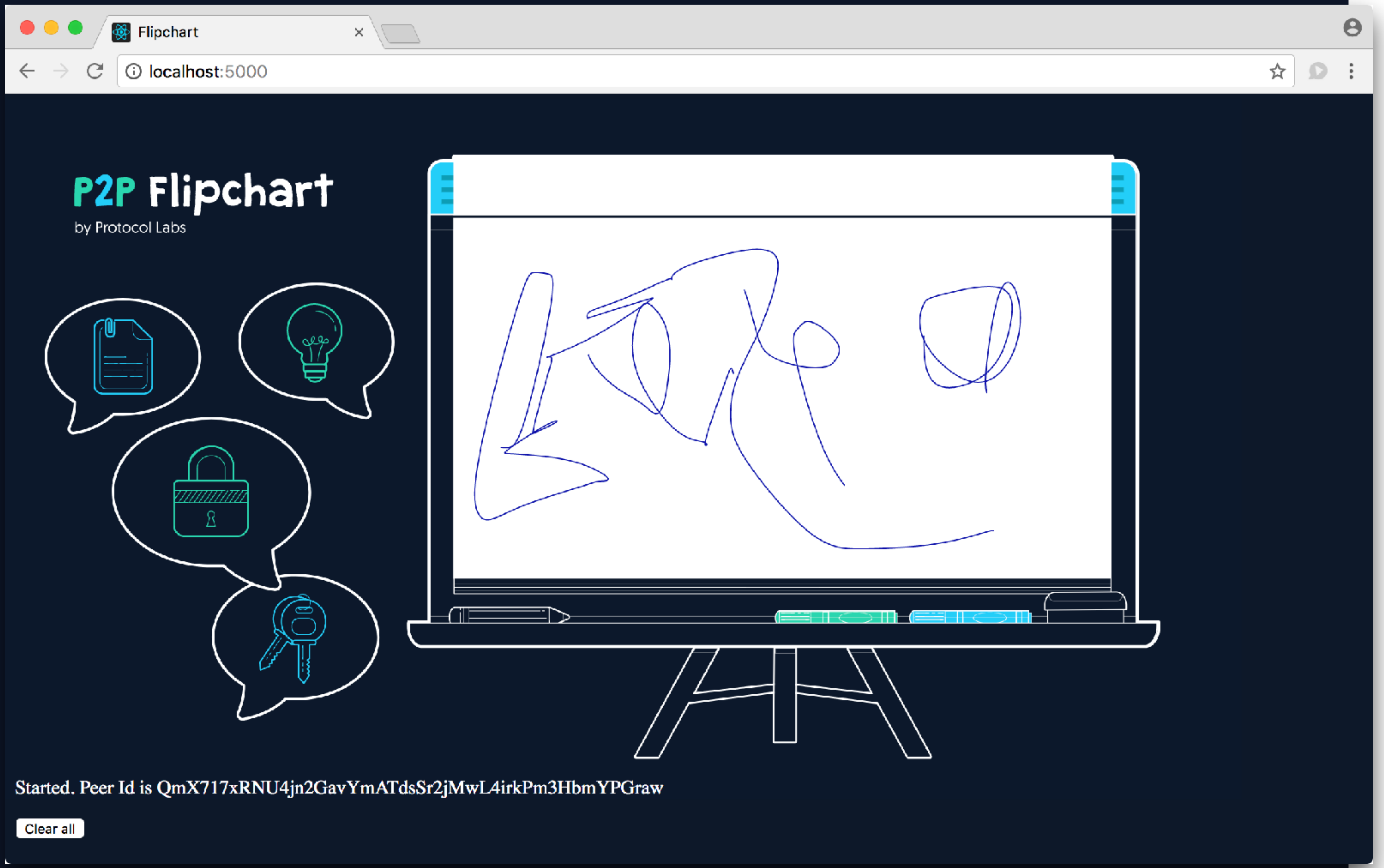
Started. Peer Id is QmX717xRNU4jn2GavYmATdsSr2jMwL4irkPm3HbmYPGraw

Under the hood



Homework Challenge

- Allow user to take a snapshot
- Save the snapshot to IPFS

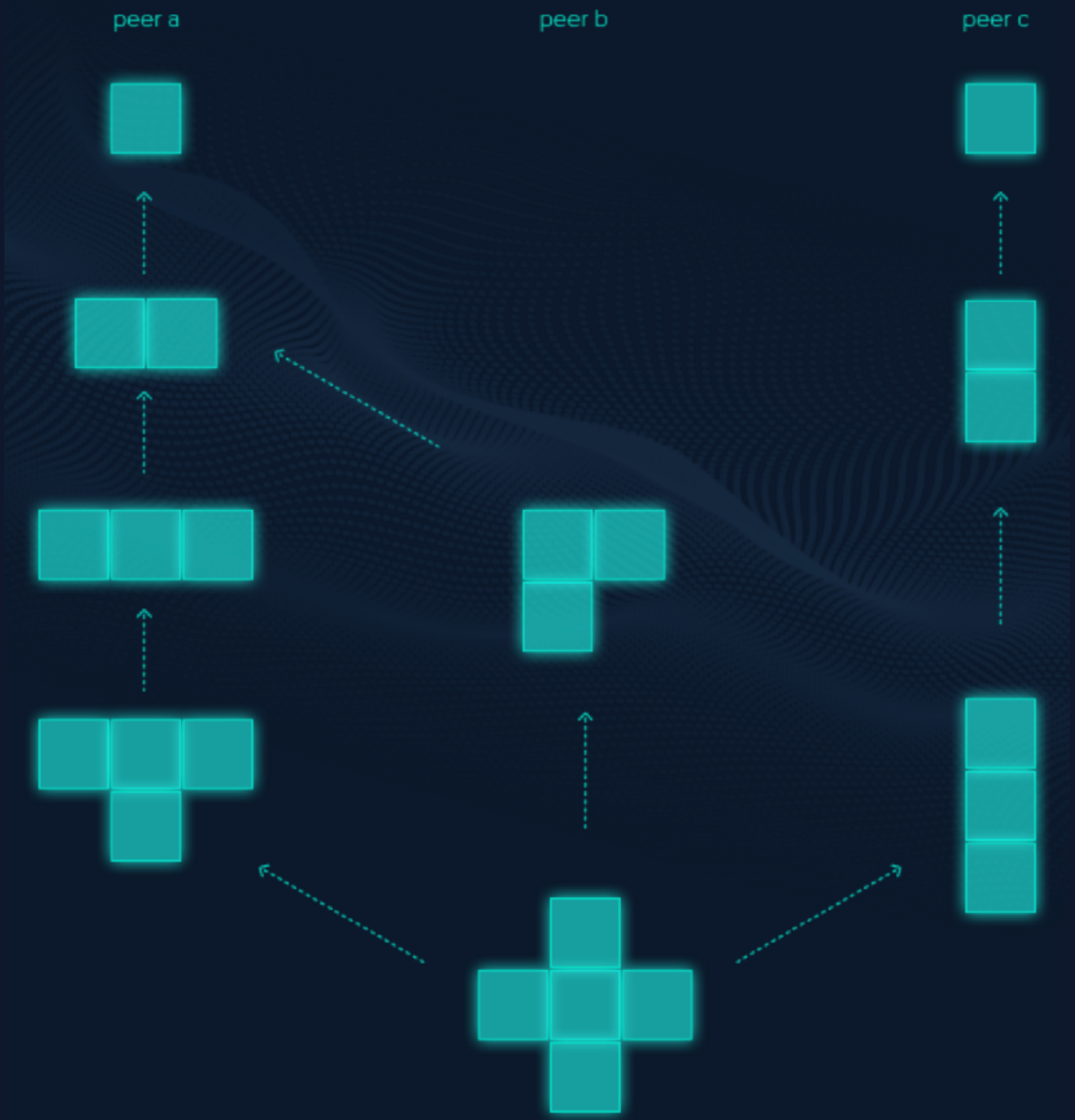


Started. Peer Id is QmX717xRNU4jn2GavYmATdsSr2jMwL4irkPm3HbmYPGraw

CRDTs

CRDTs will be the building block of decentralised collaborative applications.

This type of data structures allows building conflict-free offline-first reliable decentralised apps.



Discussion

CRDTs allow you to build a decentralised collaborative store.

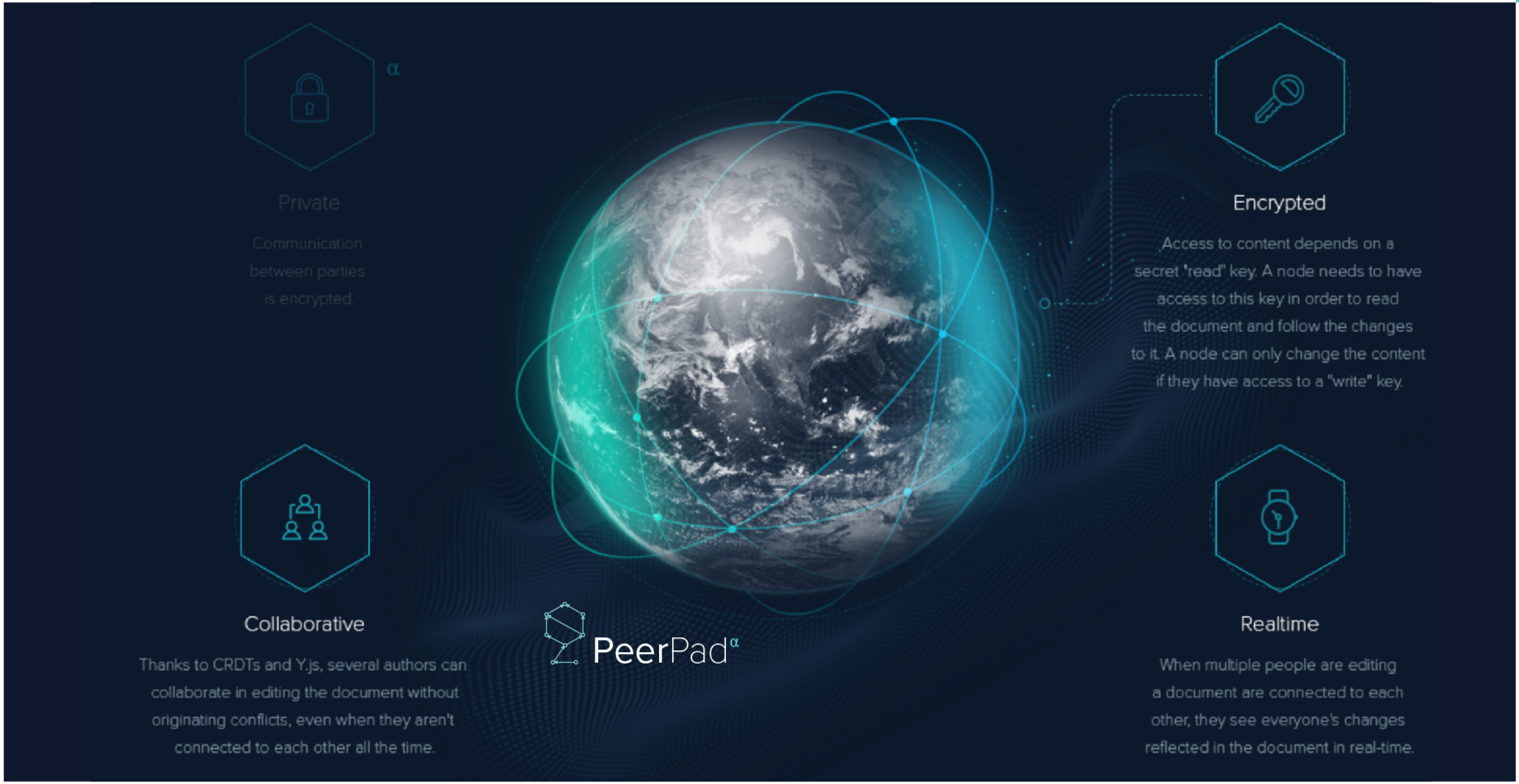
What other building blocks will be necessary to build DApps?

- Identity
- Authentication
- Access Control
- Privacy
- Files
- non-local Key/Object store
- Messaging
- … ?

PeerPad

PeerPad is a real-time collaborative editing tool, powered by IPFS and CRDTs.

https://peerpad.net



α

Private

Communication between parties is encrypted.

Encrypted

Access to content depends on a secret "read" key. A node needs to have access to this key in order to read the document and follow the changes to it. A node can only change the content if they have access to a "write" key.

PeerPadα

Collaborative

Thanks to CRDTs and Y.js, several authors can collaborate in editing the document without originating conflicts, even when they aren't connected to each other all the time.

Realtime

When multiple people are editing a document are connected to each other, they see everyone's changes reflected in the document in real-time.