Мои данные — это мои данные

или Local-first Decentralized Web

Павел Вострикoв

Community + JS Buddy

– Джуниор су-шеф,
– Co-org @ MoscowJS,
– Bibliography — Reading list for IT-hungry people
– Node.js member & Open Source fan

Web Platform, Node.js services,
product development

Kaspersky

Moscow JS
60

Moscow JS
60

@MOSCOWJS

Ммм, пиво)))

Мы должны форкнуть VS Code!

JS
@BEERJS_MOSCOW
Ммм, пиво)))
Мы должны форкнуть VS Code!

bibliography Public

Pin | Watch 2 | Fork 4 | Star 33

master | 1 Branch | 0 Tags | Go to file t | Add file | Code

vostrik feat: add recommendations from @pepelsbey_ 540999b · 5 years ago 7 Commits

README.md	feat: add recommendations from @pepelsbey_	5 years ago

README

Bibliography

Reading list for IT-hungry people.

Vadim Makeev (@pepelsbey_), from HTML Academy, recommendations:

1. «**You Don't Know JS: ES6 & Beyond**», Kyle Simpson
   - 🇺🇸 [en-US]
   - 🇷🇺 [ru-RU]
2. «**JavaScript: The Good Parts**», Douglas Crockford
   - 🇺🇸 [en-US]
   - 🇷🇺 [ru-RU]
3. «**JavaScript Patterns: Build Better Applications with Coding and Design Patterns**», Stoyan Stefanov
   - 🇺🇸 [en-US]
   - 🇷🇺 [ru-RU]
4. «**Practical SVG**», Chris Coyier
   - 🇺🇸 [en-US]
5. «**Accessibility for Everyone**», Laura Kalbag
   - 🇺🇸 [en-US]

Aleksey Kozyatinskiy (@ak_239), from Netflix, ex. @ChromeDevTools engineer, recommendations:

About

Reading list for IT-hungry people

Readme
Activity
33 stars
2 watching
4 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

@JUNIOR_SOUS_CHEF

#32. Стейк Рибай с перечным соусом

3,1 тыс. просмотров

План на ближайшие 40 минут

What?

Прокачаемся в области UX и архитектуры веб-приложений

План на ближайшие 40 минут

What?

Прокачаемся в области UX и архитектуры веб-приложений

Why?

Затронем то, чего обычно не затрагиваем в текущем production

План на ближайшие 40 минут

What?

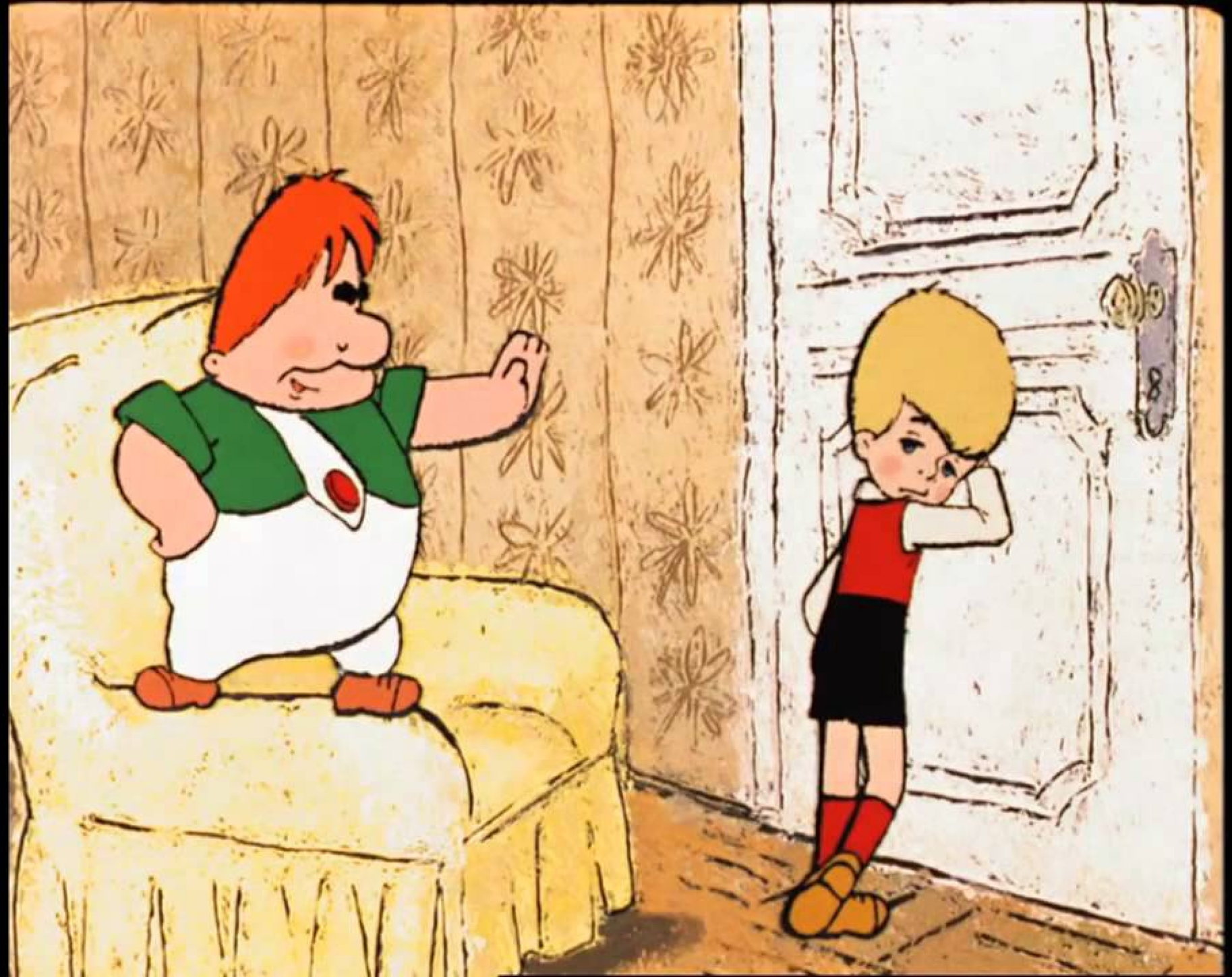
Прокачать нас в области UX и архитектуры веб-приложений

Why?

Затронем то, чего обычно не затрагиваем в текущем production

How?

Познакомимся с новыми подходами
и рассмотрим практические кейсы использования



В чём проблема?

удалили сервер	амнезия
выключили сервер	потеряли доступ к контенту
включили сервер	смотрим как крутится спиннер
отправили данные	смотрим на таргетированную рекламу

Сервер

Простота
инадежность

Внешняя зависимость
и точка отказа

Сервер

Как решать?

Local-first
Decentralized Web

Какие инструменты?

Что нужно
для Local-first Decentralized Web

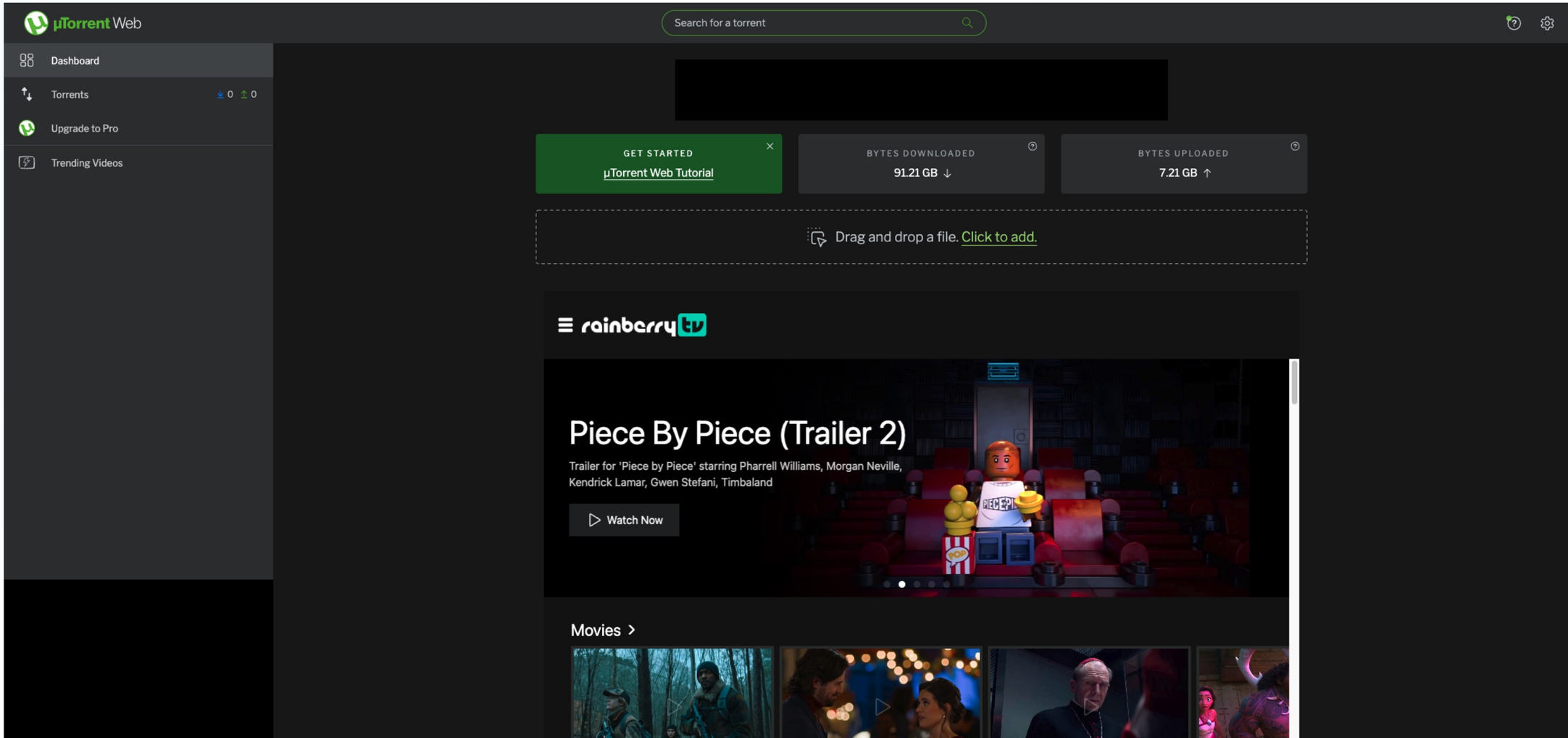
1. Публичные данные: децентрализованные сети (DAT, **IPFS**, WebTorrent)

2. Приватные данные: local p2p (OpenP2P, PeerJS, Browser-Based Peer-to-Peer Network)

3. Browser Javascript и Node.js:
   – во-первых, это нативно
   – во-вторых, нам нравится JavaScript!

uTorrent Web

Real-world examples



WebTorrent

Real-world examples

WebTorrent

Get Started | Docs | FAQ | WebTorrent Desktop NEW | Instant.io | GitHub

Get Started with WebTorrent

WebTorrent is the first torrent client that works in the **browser**. It's easy to get started!

Install

To start using WebTorrent, simply include the `webtorrent.min.js` script on your page.

```
<script src="webtorrent.min.js"></script>
```

This provides a `WebTorrent` function on the `window` object.

Browserify

WebTorrent also works great with browserify, which lets you use node.js style `require()` to organize your browser code, and load packages installed by npm.

```
npm install webtorrent
```

Then use `WebTorrent` like this:

```
const WebTorrent = require('webtorrent')
```

Quick Examples

Downloading a torrent (in the browser)

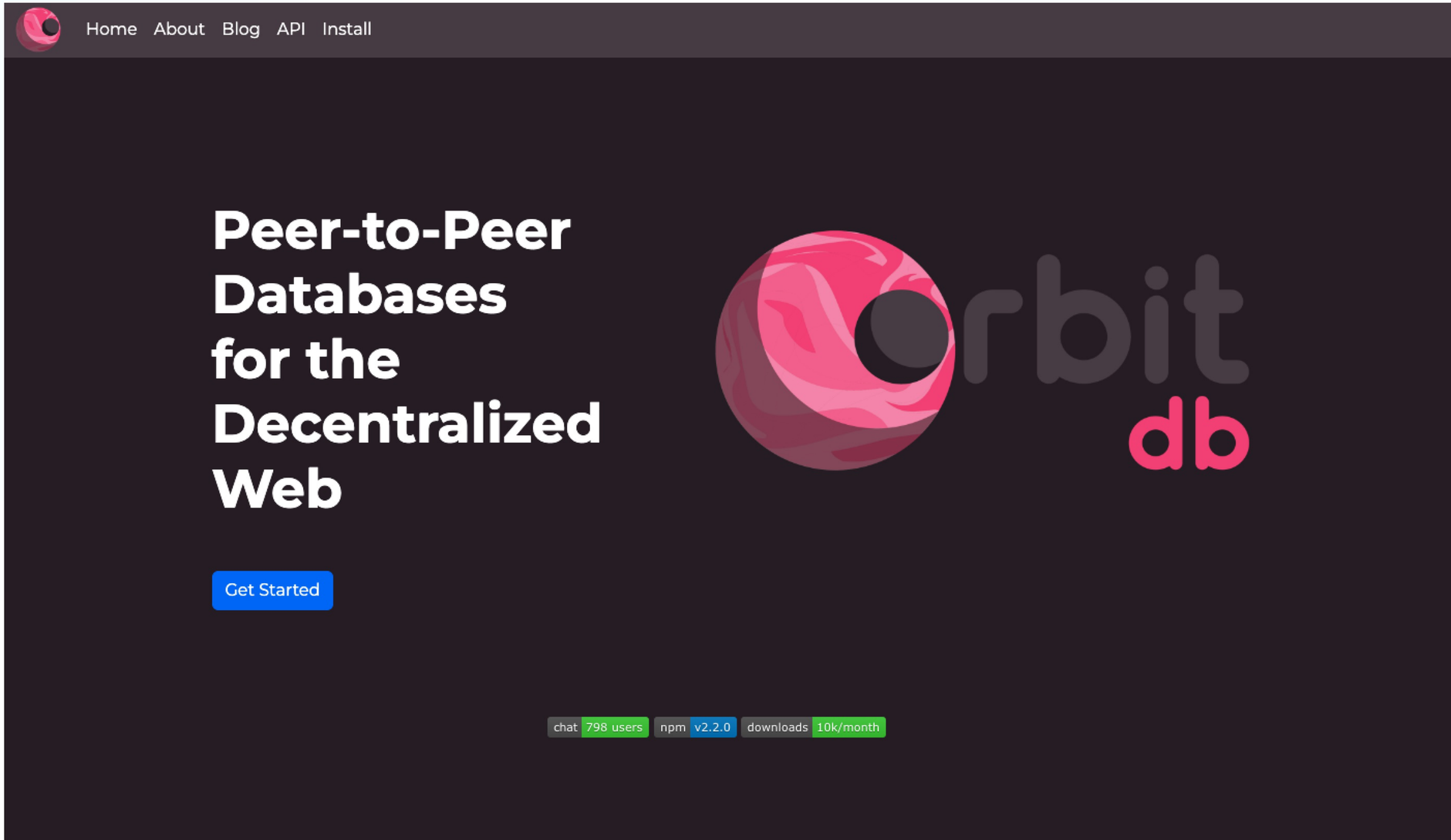
```
const WebTorrent = require('webtorrent')

const client = new WebTorrent()

// Sintel, a free, Creative Commons movie
const torrentId = 'magnet:?
xt=urn:btih:08ada5a7a6183aae1e09d831df6748d566095a10&dn=Sintel&tr=udp%3A%2F%2Fexplod
```

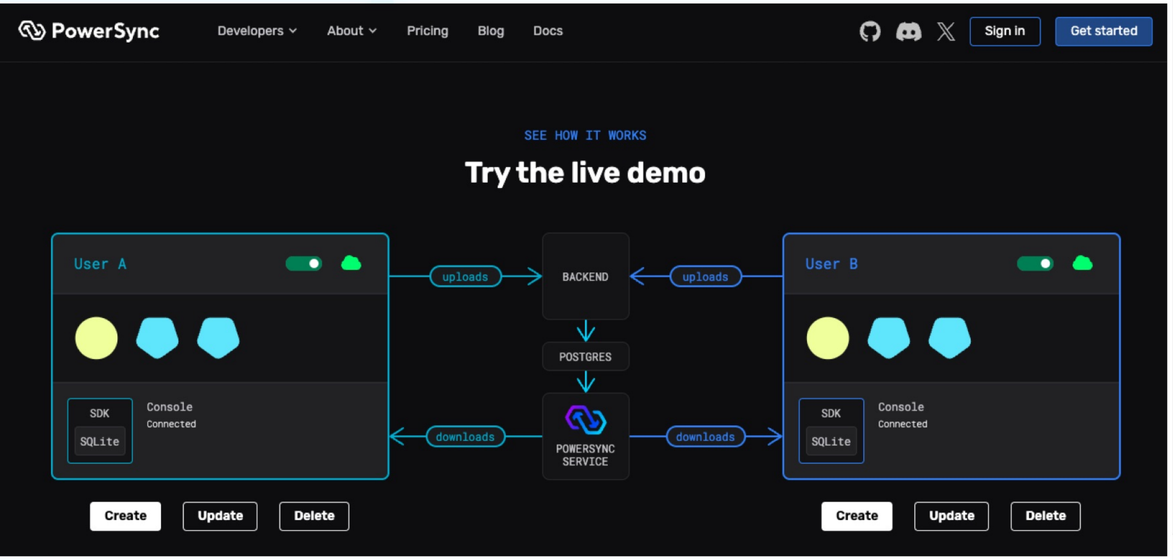
OrbitDB

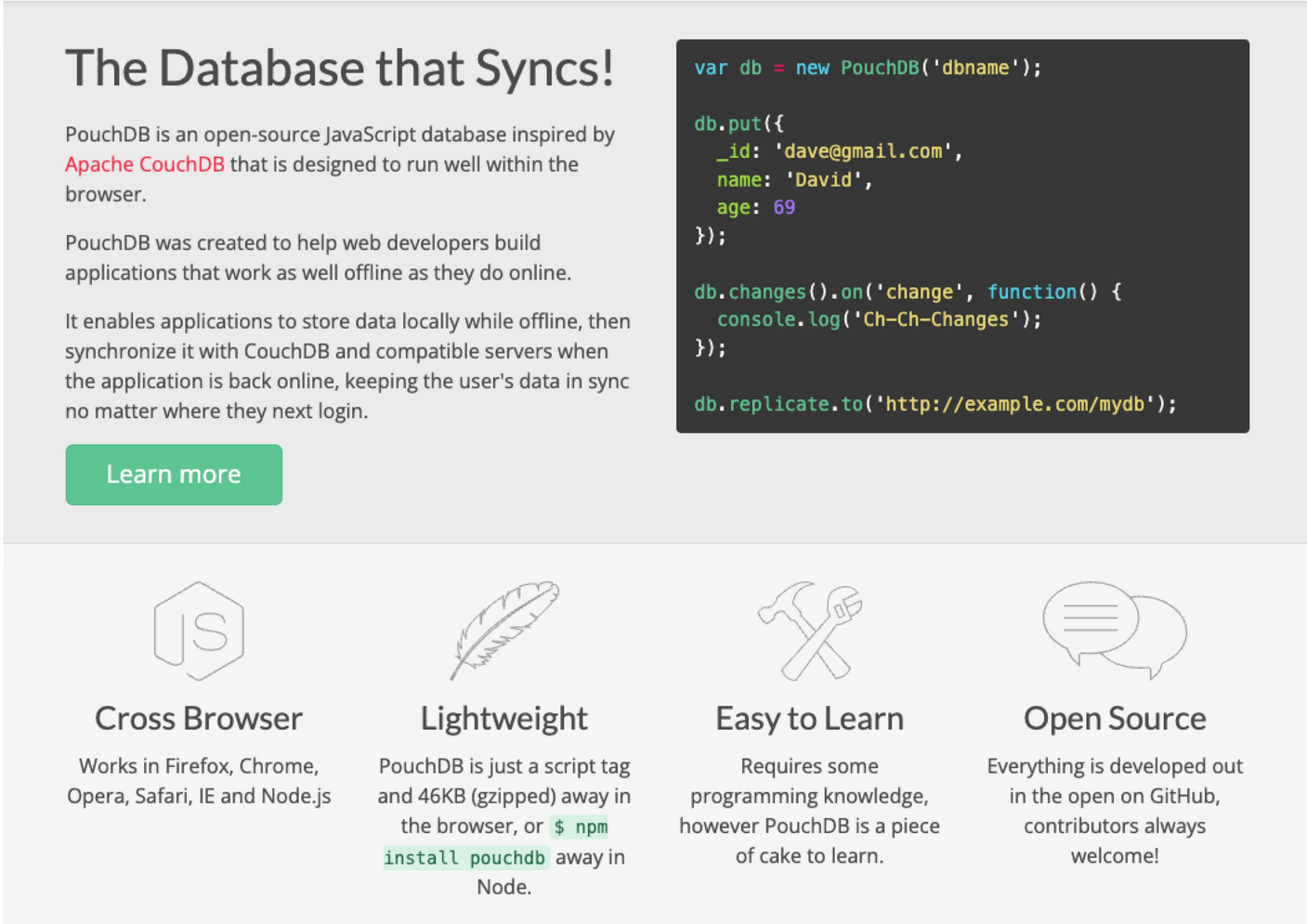
Real-world examples



Local-first Databases

Real-world examples





Local-first. Основная идея

- **Клиент** — основное хранилище данных
- **Сервер** — инструмент репликации и бекапа
- **Синхронизация данных**

Local-first. Deep dive

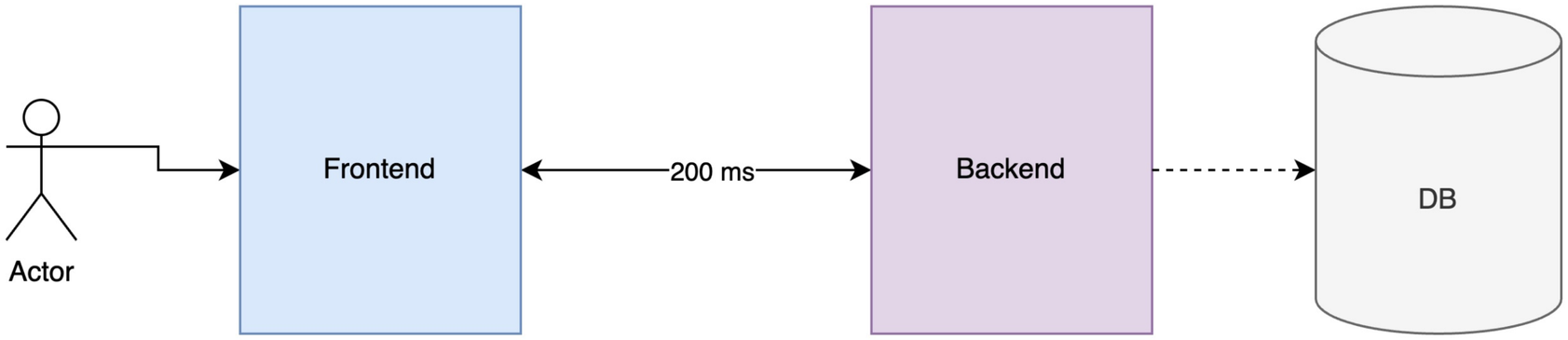
- browser JS !== frontend, **browser JS === application**

Local-first. Deep dive

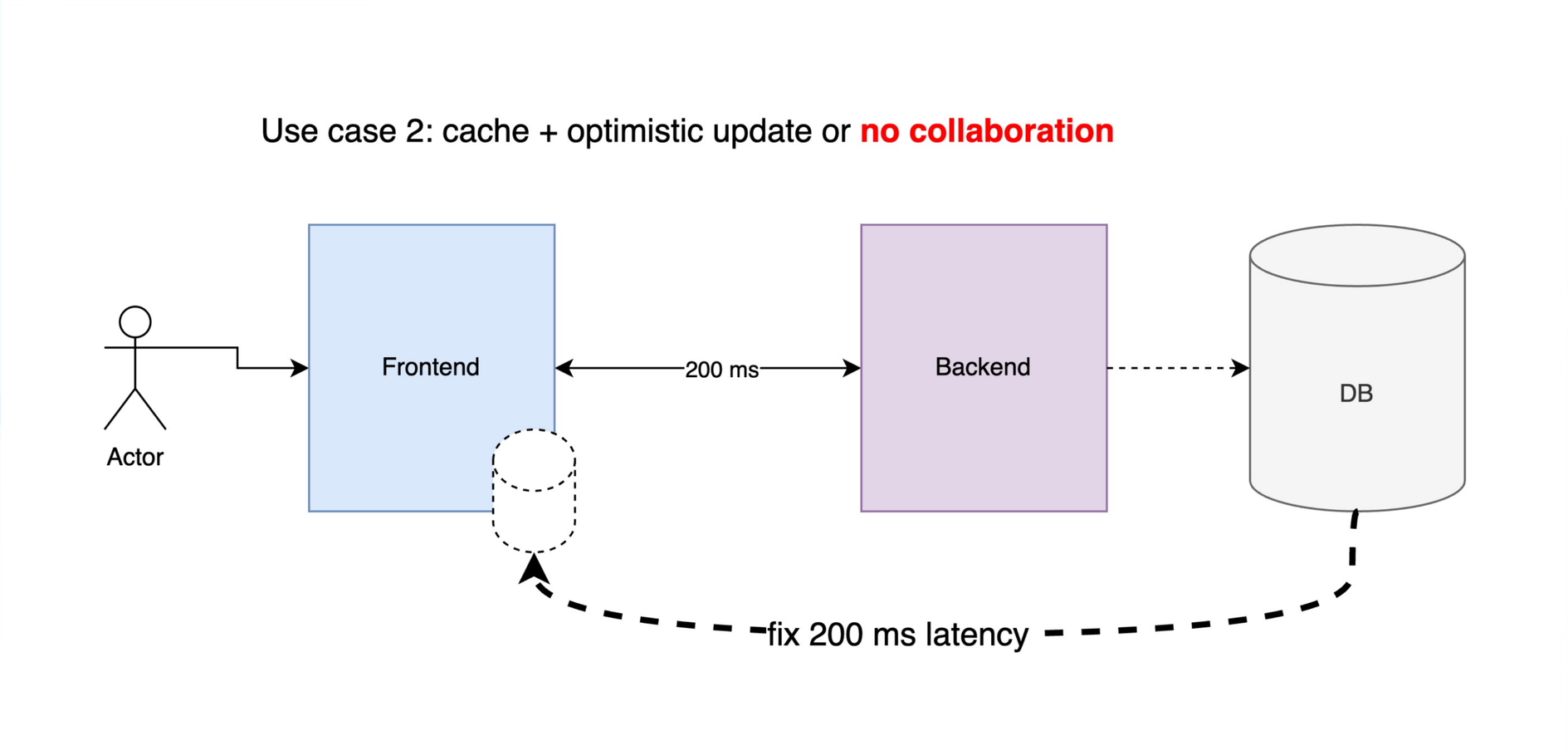
- browser JS !== frontend, browser JS === application
- local-first уже был популярен (CRDT, local cache, optimistic updates)

Local-first. Эволюция 1/5

Use case 1: classic N-tier architecture or **200 ms latency architecture**



Local-first. Эволюция 2/5



Local-first. Decrease TTI

TS useTodos.ts

1-cache-and-optimistic-update > TS useTodos.ts > ...

```
1   import { useState } from 'react';
2
3   type Todo = { id: number; text: string };
4   export const useTodos = () => {
5     const [todos, setTodos] = useState<Todo[]>([]);
6     const [cache, setCache] = useState<Todo[]>([]);
7
8     const addTodo = async (text: string) => {
9       const tempTodo = { id: -1, text };
10      setCache(todos);
11      setTodos([...todos, tempTodo]);
12
13      try {
14        const savedTodo = await new Promise<Todo>((resolve) =>
15          setTimeout(() => resolve({ ...tempTodo, id: Date.now() }), 500)
16        );
17        setTodos((prev) => prev.map((t) => (t.id === -1 ? savedTodo : t)));
18      } catch {
19        setTodos(cache);
20      }
21    };
22
23    return { todos, addTodo };
24  };
25
```

1

2

Local-first. Decrease TTI

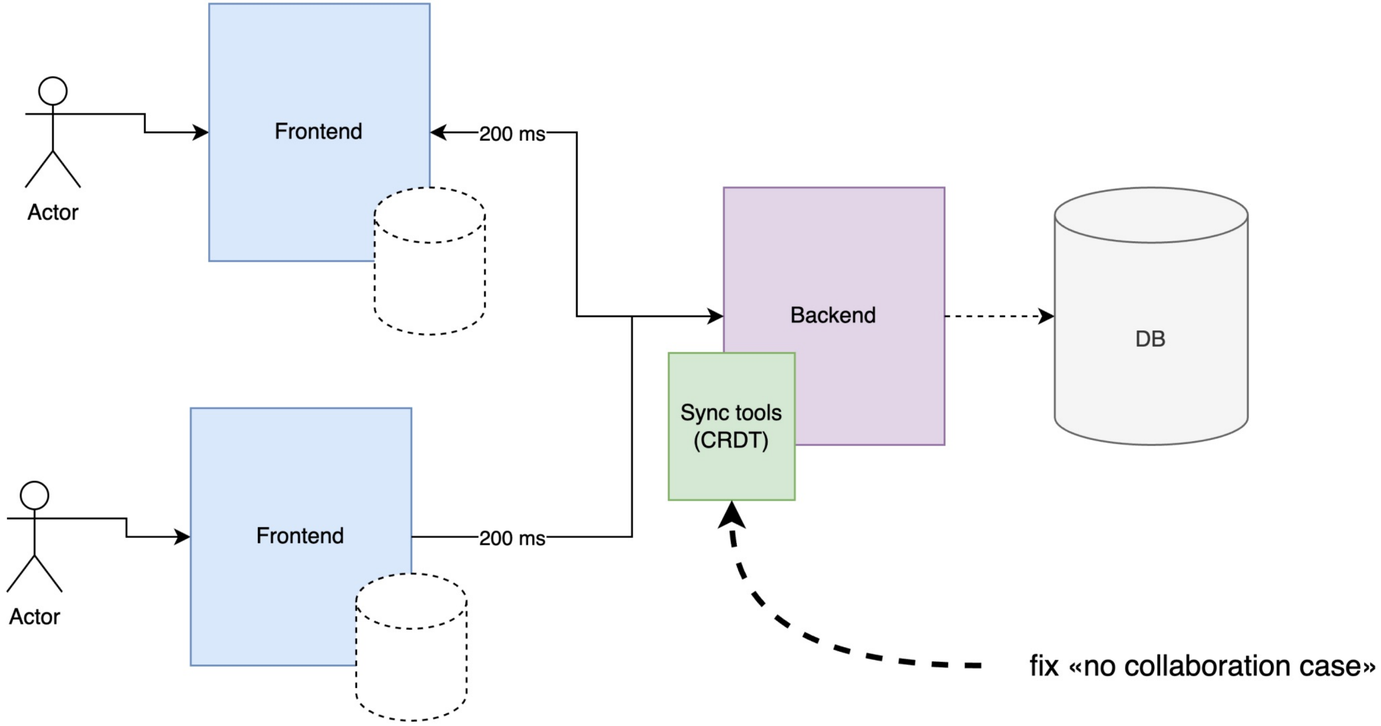
TodoApp.tsx

1-cache-and-optimistic-update > TodoApp.tsx > ...

```
1   import React, { useState } from 'react';
2   import { useTodos } from './useTodos';
3
4   const TodoApp = () => {
5     const { todos, addTodo } = useTodos();
6     const [text, setText] = useState("");
7
8     return (
9       <div>
10        <input value={text} onChange={(e) => setText(e.target.value)} />
11        <button onClick={() => { addTodo(text); setText(""); }}>Add</button>
12        <ul>
13          {todos.map((todo) => (
14            <li key={todo.id}>
15              {todo.text} {todo.id === -1 && "(saving...)"}
16            </li>
17          ))}
18        </ul>
19      </div>
20    );
21  };
22
23  export default TodoApp;
24
```

Local-first. Эволюция 3/5

Use case 3: sync tools, more cache and more optimistic things or **happy debug «no internet connection»**



Local-first. Collaboration

TS useTodosAutomerge.ts

2-sync-clients > TS useTodosAutomerge.ts > ...

```
1   import { useState, useEffect } from 'react';
2   import * as Automerge from 'automerge';
3
4   type Todo = { id: string; text: string; completed: boolean };
5
6   export const useTodosAutomerge = () => {
7     const [todosDoc, setTodosDoc] = useState(Automerge.from<{ todos: Todo[] }>({ todos: [] }));
8     const [ws, setWs] = useState<WebSocket | null>(null);
9
10    // Initialize WebSocket
11  > useEffect(() => {…
23    }, []);
24
25  > const addTodo = (text: string) => {…
35    };
36
37    return {
38      todos: todosDoc.todos,
39      addTodo,
40    };
41  };
42
```

1

2

Local-first. Collaboration

TS useTodosAutomerge.ts ×

2-sync-clients > TS useTodosAutomerge.ts > useTodosAutomerge > useEffect() callback

```
6   export const useTodosAutomerge = () => {
10    // Initialize WebSocket
11    useEffect(() => {
12      const socket = new WebSocket("ws://localhost:8080");
13      setWs(socket);
14
15      socket.onmessage = (event) => {
16        const remoteChanges = new Uint8Array(JSON.parse(event.data));
17        setTodosDoc((doc) => Automerge.applyChanges(doc, [remoteChanges])[0]);
18      };
19
20      return () => {
21        socket.close();
22      };
23    }, []);
```

1

Local-first. Collaboration

TS useTodosAutomerge.ts

2-sync-clients > TS useTodosAutomerge.ts > useTodosAutomerge > useEffect() callback

```
6   export const useTodosAutomerge = () => {
25    const addTodo = (text: string) => {
26      setTodosDoc((doc) => Automerge.change(doc, (draft) => {
27        draft.todos.push({ id: Date.now().toString(), text, completed: false });
28      }));
29
30      // Optimistic update: send changes to server immediately
31      if (ws) {
32        const changes = Automerge.getLastLocalChange(todosDoc);
33        if (changes) ws.send(JSON.stringify(Array.from(changes)));
34      }
35    };
```

1

2

Local-first. Collaboration

TS useTodosAutomerge.ts

2-sync-clients > TS useTodosAutomerge.ts > useTodosAutomerge > useEffect() callback

```
6   export const useTodosAutomerge = () => {
10    // Initialize WebSocket
11    useEffect(() => {
12      const socket = new WebSocket("ws://localhost:8080");
13      setWs(socket);
14
15      socket.onmessage = (event) => {
16        const remoteChanges = new Uint8Array(JSON.parse(event.data));
17        setTodosDoc((doc) => Automerge.applyChanges(doc, [remoteChanges])[0]);
18      };
19
20      return () => {
21        socket.close();
22      };
23    }, []);
24
25    const addTodo = (text: string) => {
26      setTodosDoc((doc) => Automerge.change(doc, (draft) => {
27        draft.todos.push({ id: Date.now().toString(), text, completed: false });
28      }));
29
30      // Optimistic update: send changes to server immediately
31      if (ws) {
32        const changes = Automerge.getLastLocalChange(todosDoc);
33        if (changes) ws.send(JSON.stringify(Array.from(changes)));
34      }
35    };
36
```

1

2

Local-first. Collaboration

TodoApp.tsx

2-sync-clients > TodoApp.tsx > TodoApp > todos.map() callback

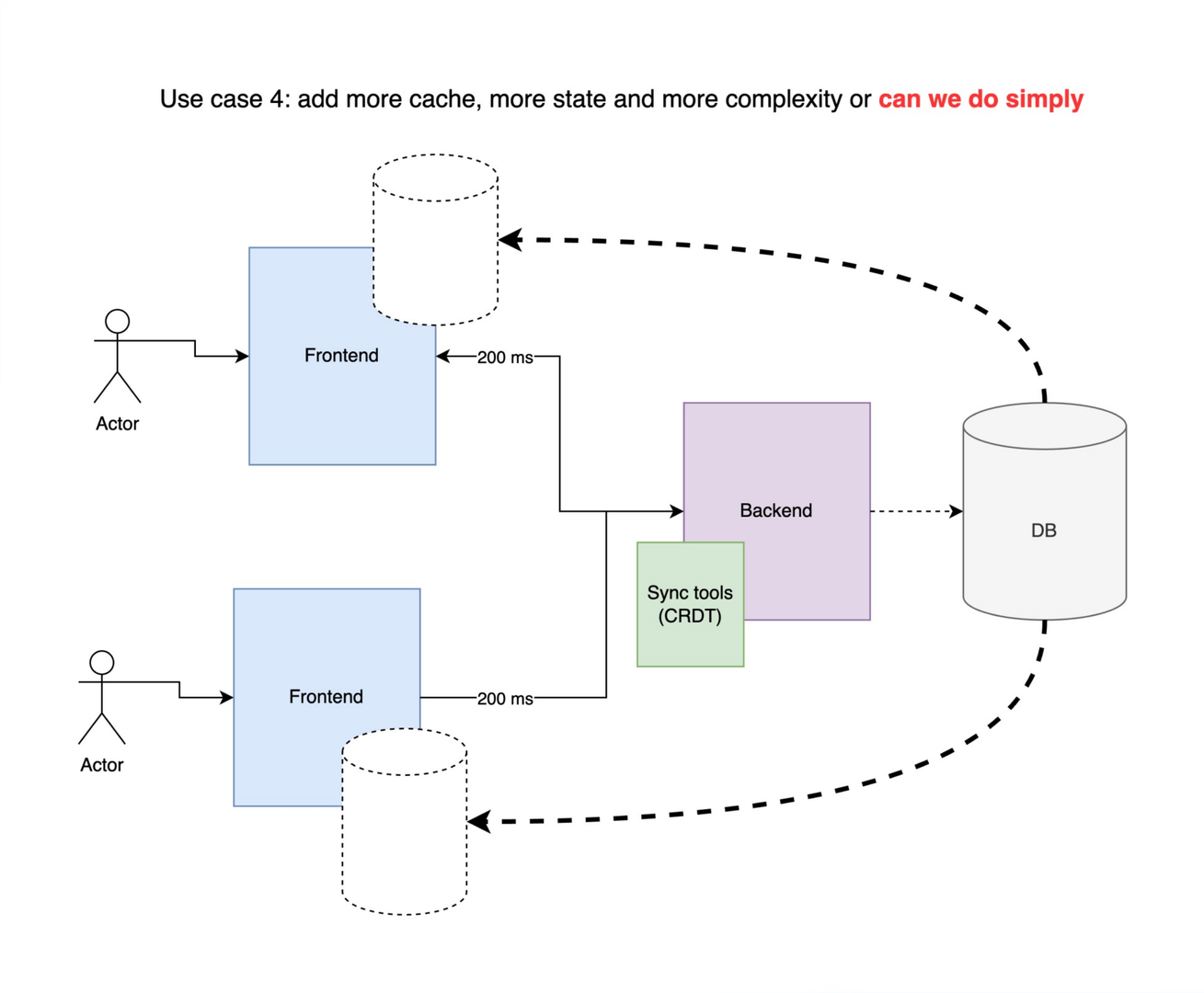
```react/usetodosautomerge/todoapp.tsx
import React, { useState } from 'react';
import { useTodosAutomerge } from './useTodosAutomerge';

const TodoApp = () => {
  const { todos, addTodo, toggleTodo } = useTodosAutomerge();
  const [text, setText] = useState("");

  return (
    <div>
      <input value={text} onChange={(e) => setText(e.target.value)} />
      <button onClick={() => { addTodo(text); setText(""); }}>Add</button>
      <ul>
        {todos.map((todo) => (
          <li key={todo.id}>
            {todo.text} {todo.completed ? "(completed)" : ""}
          </li>
        ))}
      </ul>
    </div>
  );
};

export default TodoApp;
```

Local-first. Эволюция 4/5



Local-first. Airplane mode

TS useTodosAutomerge.ts

3-offline-first > TS useTodosAutomerge.ts > useTodosAutomerge

```
1   import { useState, useEffect } from 'react';
2   import * as Automerge from 'automerge';
3
4   type Todo = { id: string; text: string; completed: boolean };
5
6   const LOCAL_STORAGE_KEY = "todos-automerge-doc";
7
8   export const useTodosAutomerge = () => {
9     const [todosDoc, setTodosDoc] = useState(() => {
10      const savedDoc = localStorage.getItem(LOCAL_STORAGE_KEY);
11      return savedDoc
12        ? Automerge.load<{ todos: Todo[] }>(Uint8Array.from(JSON.parse(savedDoc)))
13        : Automerge.from<{ todos: Todo[] }>({ todos: [] });
14    });
15
16    const [ws, setWs] = useState<WebSocket | null>(null);
17    const [isOnline, setIsOnline] = useState(navigator.onLine);
18
19    // Save to local storage whenever todosDoc changes
20    useEffect(() => {…
22    }, [todosDoc]);
23
24    // Monitor connection status and reconnect when online
25    useEffect(() => {…
34    }, []);
35
36    // Initialize WebSocket connection when online
37    // And sync local changes on reconnect
38    useEffect(() => {…
57    }, [isOnline, ws, todosDoc]);
58
59    const addTodo = (text: string) => {…
70    };
71
72    return { todos: todosDoc.todos, addTodo };
73  };
74
```

1

2

3

4

5

Local-first. Airplane mode

TS useTodosAutomerge.ts ×

3-offline-first > TS useTodosAutomerge.ts > useTodosAutomerge

```
5
6   const LOCAL_STORAGE_KEY = "todos-automerge-doc";
7
8   export const useTodosAutomerge = () => {
9     const [todosDoc, setTodosDoc] = useState(() => {
10      const savedDoc = localStorage.getItem(LOCAL_STORAGE_KEY);
11      return savedDoc
12        ? Automerge.load<{ todos: Todo[] }>(Uint8Array.from(JSON.parse(savedDoc)))
13        : Automerge.from<{ todos: Todo[] }>({ todos: [] });
14    });
15
16    const [ws, setWs] = useState<WebSocket | null>(null);
17    const [isOnline, setIsOnline] = useState(navigator.onLine);
18
```

1

2

Local-first. Airplane mode

useTodosAutomerge.ts

3-offline-first > useTodosAutomerge.ts > useTodosAutomerge > useEffect() callback

```
8   export const useTodosAutomerge = () => {

16    const [ws, setWs] = useState<WebSocket | null>(null);
17    const [isOnline, setIsOnline] = useState(navigator.onLine);
18
19    // Save to local storage whenever todosDoc changes
20    useEffect(() => {
21      localStorage.setItem(
22        LOCAL_STORAGE_KEY,
23        JSON.stringify(Array.from(Automerge.save(todosDoc)))
24      );
25    }, [todosDoc]);
26
```

1

Local-first. Airplane mode

useTodosAutomerge.ts

3-offline-first > useTodosAutomerge.ts > useTodosAutomerge > useEffect() callback

```
8   export const useTodosAutomerge = () => {
27    // Monitor connection status and reconnect when online
28    useEffect(() => {
29      const handleOnline = () => setIsOnline(true);
30      const handleOffline = () => setIsOnline(false);
31      window.addEventListener("online", handleOnline);
32      window.addEventListener("offline", handleOffline);
33      return () => {
34        window.removeEventListener("online", handleOnline);
35        window.removeEventListener("offline", handleOffline);
36      };
37    }, []);
38
```

Local-first. Airplane mode

TS useTodosAutomerge.ts

3-offline-first > TS useTodosAutomerge.ts > useTodosAutomerge

```
8   export const useTodosAutomerge = () => {
39    // Initialize WebSocket connection when online
40    // And sync local changes on reconnect
41    useEffect(() => {
42      if (isOnline && !ws) {
43        // Initialize WebSocket connection when online
44        const socket = new WebSocket("ws://localhost:8080");
45        setWs(socket);
46
47        socket.onopen = () => {
48          // Sync local changes on reconnect
49          const changes = Automerge.getChanges(Automerge.init(), todosDoc);
50          socket.send(JSON.stringify(Array.from(changes)));
51        };
52
53        socket.onmessage = (event) => {
54          const remoteChanges = new Uint8Array(JSON.parse(event.data));
55          setTodosDoc((doc) => Automerge.applyChanges(doc, [remoteChanges])[0]);
56        };
57
58        socket.onclose = () => setWs(null);
59      }
60    }, [isOnline, ws, todosDoc]);
61
```

1

2

Local-first. Airplane mode

TodoApp.tsx

3-offline-first > TodoApp.tsx > ...

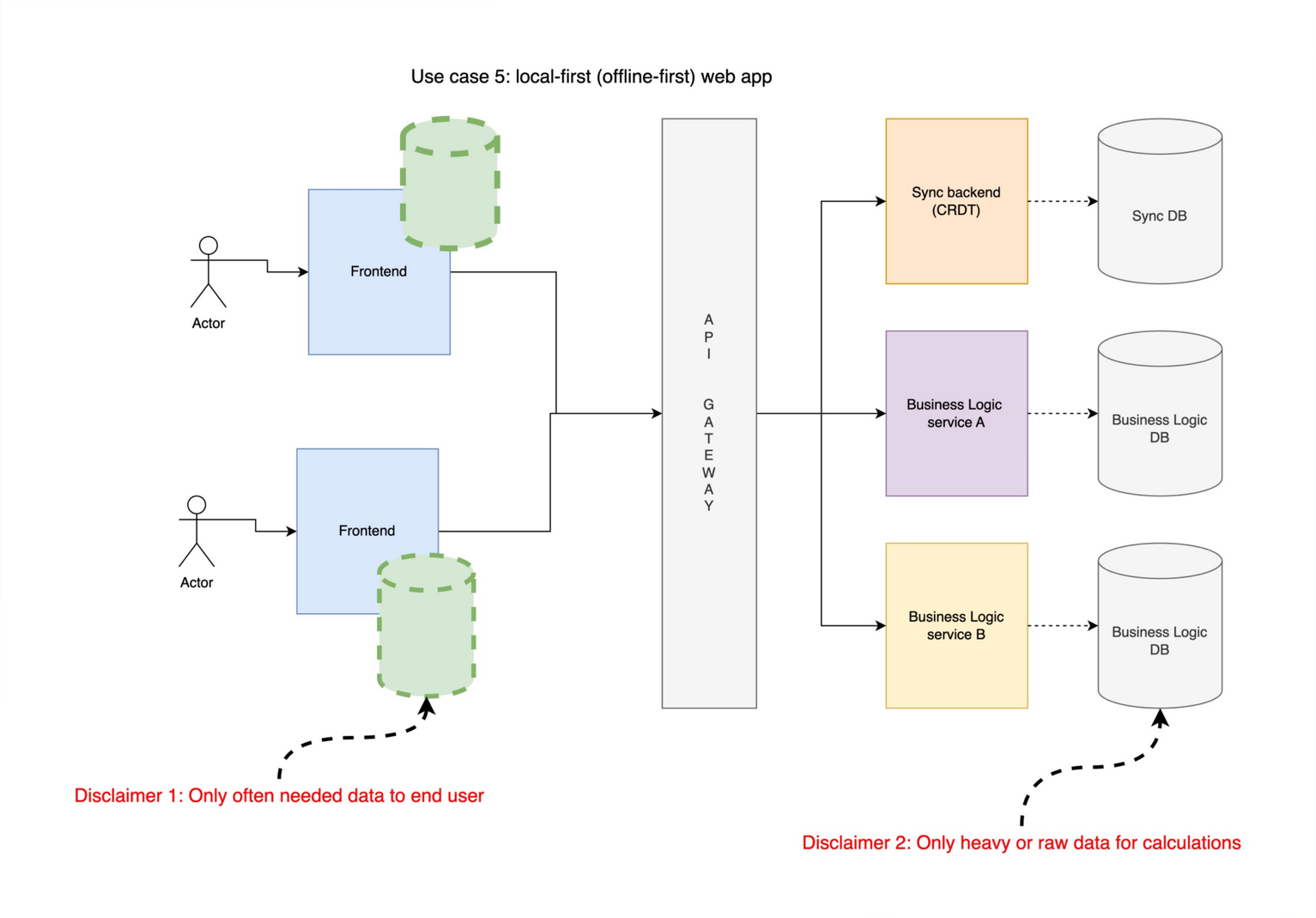
```
/**
 *
 *  NO CHANGES NEEDED
 *
 */
import React, { useState } from 'react';
import { useTodosAutomerge } from './useTodosAutomerge';

const TodoApp = () => {
  const { todos, addTodo, toggleTodo } = useTodosAutomerge();
  const [text, setText] = useState("");

  return (
    <div>…
    </div>
  );
};

export default TodoApp;
```

Local-first. Эволюция 5/5



Local-first. Synced SQLite in browser

TS useTodosAutomerge.ts

4-browser-sqlite > TS useTodosAutomerge.ts > useTodosAutomerge

```
1   import { useState, useEffect } from 'react';
2   import * as Automerge from 'automerge';
3   import initSqlJs from 'sql.js';
4
5   type Todo = { id: string; text: string; completed: boolean };
6
7   export const useTodosAutomerge = () => {
8     const [todosDoc, setTodosDoc] = useState(() => Automerge.from<{ todos: Todo[] }>({ todos: [] }));
9     const [ws, setWs] = useState<WebSocket | null>(null);
10    const [isOnline, setIsOnline] = useState(navigator.onLine);
11    const [db, setDb] = useState<any>(null);
12
13    // init db
14    useEffect(() => {…
39    }, []);
40
41    // Monitor connection status and reconnect when online
42    useEffect(() => {…
52    }, []);
53
54    // Initialize WebSocket connection when online
55    useEffect(() => {…
73    }, [isOnline, ws, todosDoc]);
74
75    const addTodo = (text: string) => {…
97    };
98
99    return { todos: todosDoc.todos, addTodo };
100 };
101
```

1

2

Local-first. Synced SQLite in browser

TS useTodosAutomerge.ts

4-browser-sqlite > TS useTodosAutomerge.ts > useTodosAutomerge

```ts
// Utility functions for IndexedDB storage
// Why we need Indexed DB?
// To persist DB state between sessions
const saveToIndexedDB = (key: string, data: Uint8Array) => {
  const request = indexedDB.open(key, 1);
  request.onsuccess = () => {
    const db = request.result;
    const transaction = db.transaction(key, "readwrite");
    const store = transaction.objectStore(key);
    store.put(data, "todos");
  };
};

const loadFromIndexedDB = (key: string): Promise<Uint8Array | null> => {
  return new Promise((resolve) => {
    const request = indexedDB.open(key, 1);
    request.onsuccess = () => {
      const db = request.result;
      const transaction = db.transaction(key, "readonly");
      const store = transaction.objectStore(key);
      const dataRequest = store.get("todos");
      dataRequest.onsuccess = () => resolve(dataRequest.result ? new Uint8Array(dataRequest.result) : null);
    };
    request.onerror = () => resolve(null);
  });
};
```

1

Local-first. Synced SQLite in browser

TS useTodosAutomerge.ts

useTodosAutomerge.ts > useTodosAutomerge > useEffect() callback > initializeDb > setTodosDoc() callback >

```
7   export const useTodosAutomerge = () => {
13    // init db
14    useEffect(() => {
15      const initializeDb = async () => {
16        const SQL = await initSqlJs({ locateFile: (file) => `https://sql.js.org/dist/${file}` });
17        const db = new SQL.Database();
18        db.run("CREATE TABLE IF NOT EXISTS todos (id TEXT, text TEXT, completed INTEGER)");
19        setDb(db);
20
21        // Load existing todos from IndexedDB
22        const savedData = await loadFromIndexedDB("todos-automerge-db");
23        if (savedData) db.run(savedData);
24
25        // Load todos into Automerge from the database
26        const todos = db.exec("SELECT * FROM todos")[0]?.values || [];
27        setTodosDoc((doc) =>
28          Automerge.change(doc, (draft) => {
29            draft.todos = todos.map(([id, text, completed]: [string, string, number]) => ({
30              id,
31              text,
32              completed: !!completed,
33            }));
34          })
35        );
36      };
37
38      initializeDb();
39    }, []);
40
```

1

2

3

Local-first. Synced SQLite in browser

TodoApp.tsx

4-browser-sqlite > TodoApp.tsx > ...

```
/**
 *
 *  NO CHANGES NEEDED
 *
 */
import React, { useState } from 'react';
import { useTodosAutomerge } from './useTodosAutomerge';

const TodoApp = () => {
  const { todos, addTodo, toggleTodo } = useTodosAutomerge();
  const [text, setText] = useState("");

  return (
    <div>
      <input value={text} onChange={(e) => setText(e.target.value)} />
      <button onClick={() => { addTodo(text); setText(""); }}>Add</button>
      <ul>
        {todos.map((todo) => (
          <li key={todo.id}>
            {todo.text} {todo.completed ? "(completed)" : ""}
          </li>
        ))}
      </ul>
    </div>
  );
};

export default TodoApp;
```

Local-first. Deep dive

- browser JS !== frontend, browser JS === application
- local-first уже был популярен (CRDT, local cache, optimistic updates)
- ренессанс local-first: **reactive storage** in Browser (WASM, SQLite / Postgres in Browser)

Local-first. Deep dive

- browser JS !== frontend, browser JS === application
- local-first уже был популярен (CRDT, local cache, optimistic updates)
- ренессанс local-first: reactive storage in Browser (WASM, SQLite / Postgres in Browser)
- как объяснить бизнесу?

Local-first. Deep dive

- browser JS !== frontend, browser JS === application
- local-first уже был популярен (CRDT, local cache, optimistic updates)
- ренессанс local-first: reactive storage in Browser (WASM, SQLite / Postgres in Browser)
- как объяснить бизнесу?

  local-first => zero latency UX

Local-first. Deep dive

- browser JS !== frontend, browser JS === application
- local-first уже был популярен (CRDT, local cache, optimistic updates)
- ренессанс local-first: reactive storage in Browser (WASM, SQLite / Postgres in Browser)
- как объяснить бизнесу?

  local-first => zero latency UX

  «Хей, Продакт, привет! давай обойдем тех медленных ребят на повороте.»

Акей, local-first — гуд,

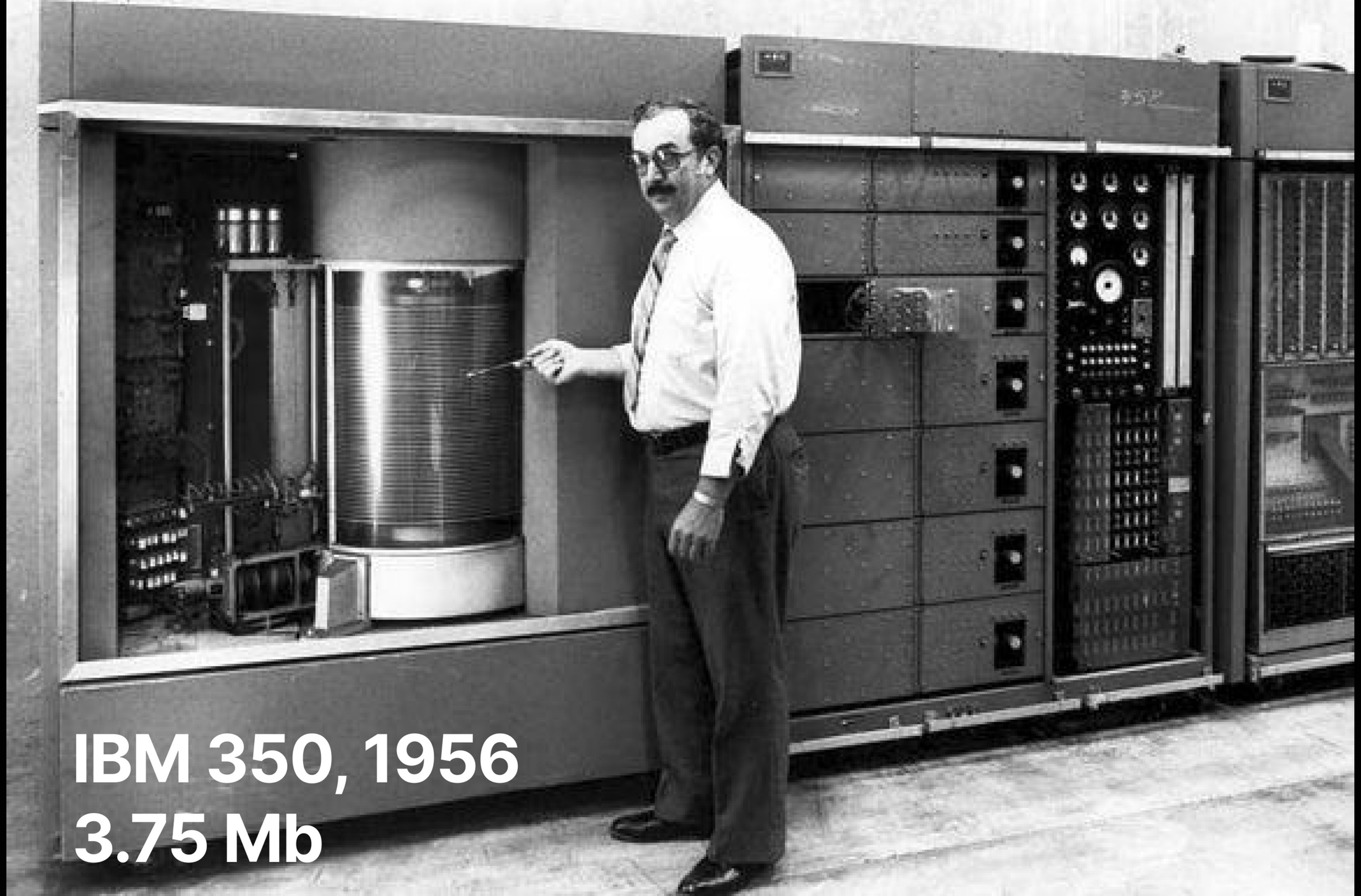
что такое IPFS и зачем?

Decentralized Web. Основная идея

- Сохранность данных при потере сервера
- Приватное и публичное распространение
- Крипто-подпись by design
  - иммутабельность
  - верификация

Storage evolution

- Centralized mainframes

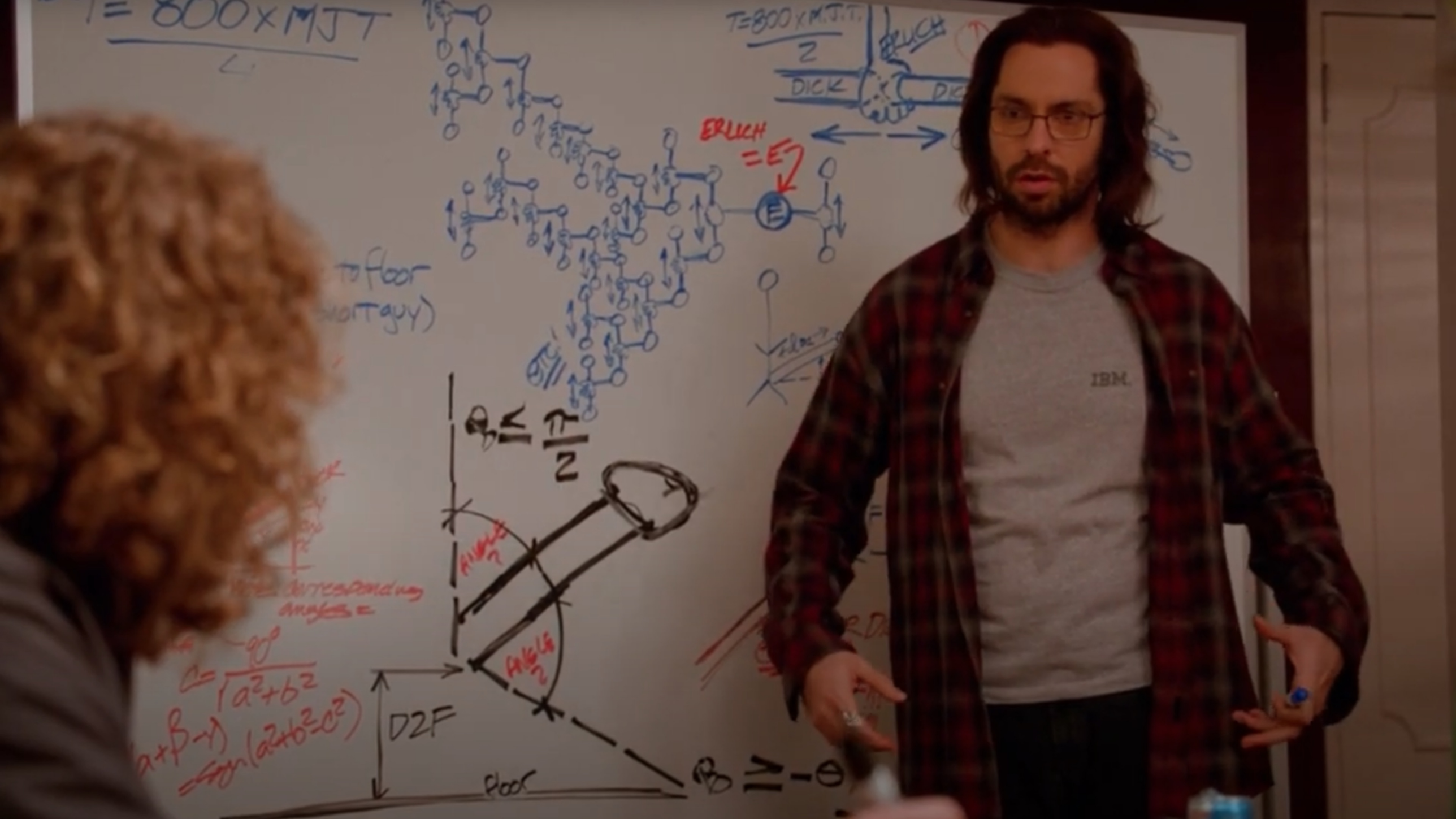

IBM 350, 1956
3.75 Mb

Storage evolution

- Centralized mainframes
- Decentralized portable devices

Storage evolution

- Centralized mainframes
- Decentralized portable devices
- Centralized data centers


T= 800 x MJT
T=800 x M.J.T.
2
DICK
ERUCH
=E
E
to floor
D2F
floor
IBM

Storage evolution

- Centralized mainframes
- Decentralized portable devices
- Centralized data centers
- Decentralized IPFS — InterPlanetary File System

Что такое IPFS?

- IPFS — InterPlanetary File System

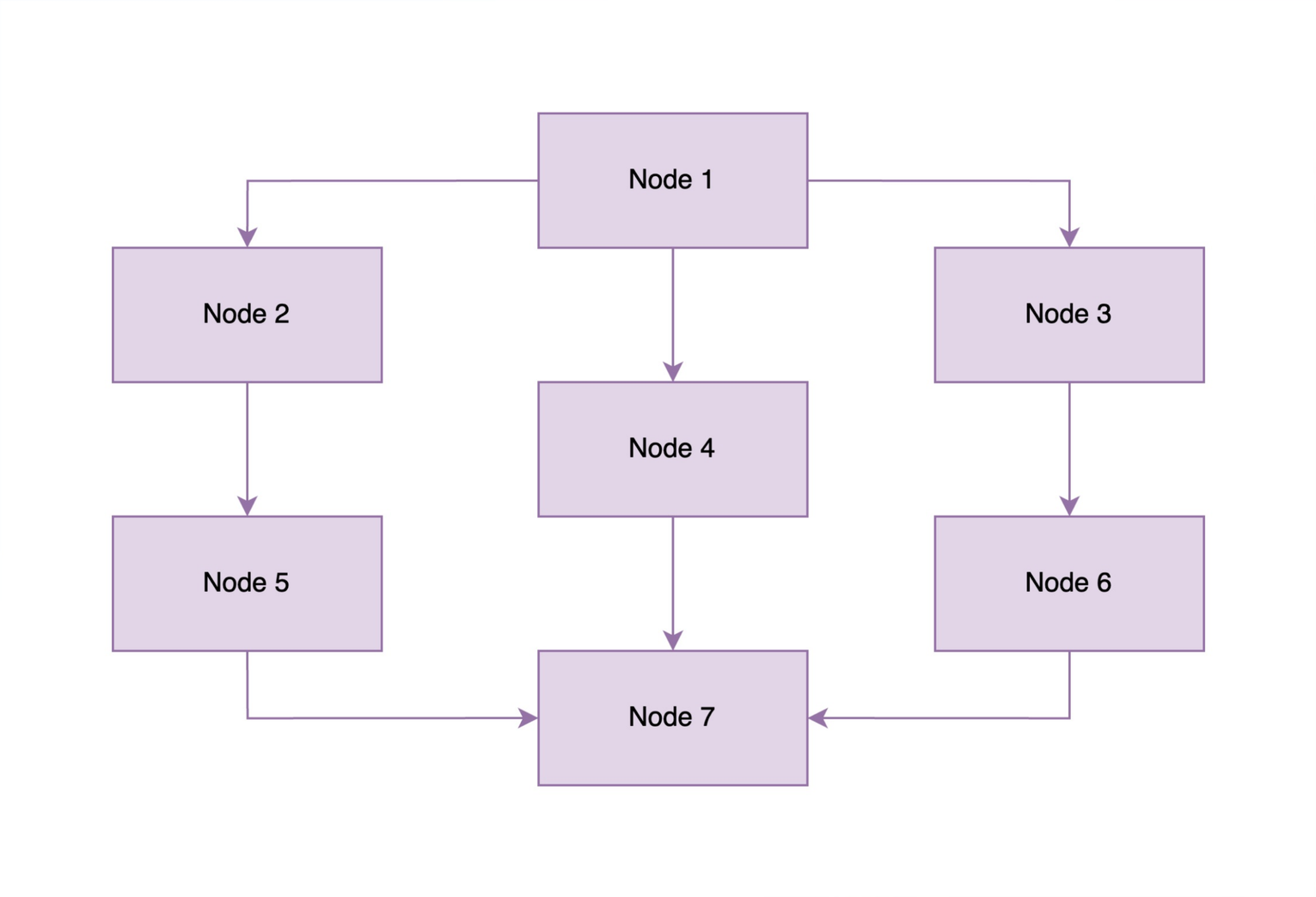
Что такое IPFS?

- IPFS — InterPlanetary File System
- IPFS is an open system to manage data without a central server

Что такое IPFS?

- IPFS — InterPlanetary File System
- IPFS is an open system to manage data without a central server
- Decentralized
- Distributed
- P2P

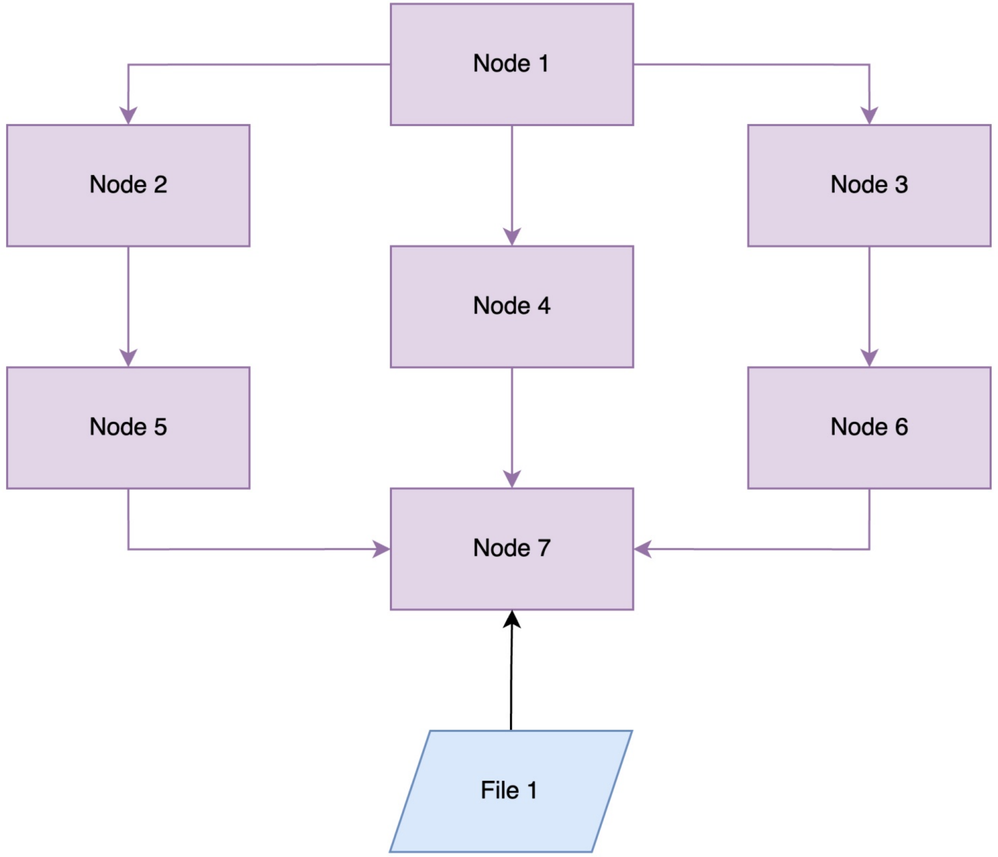
Decentralized, Distributed, P2P



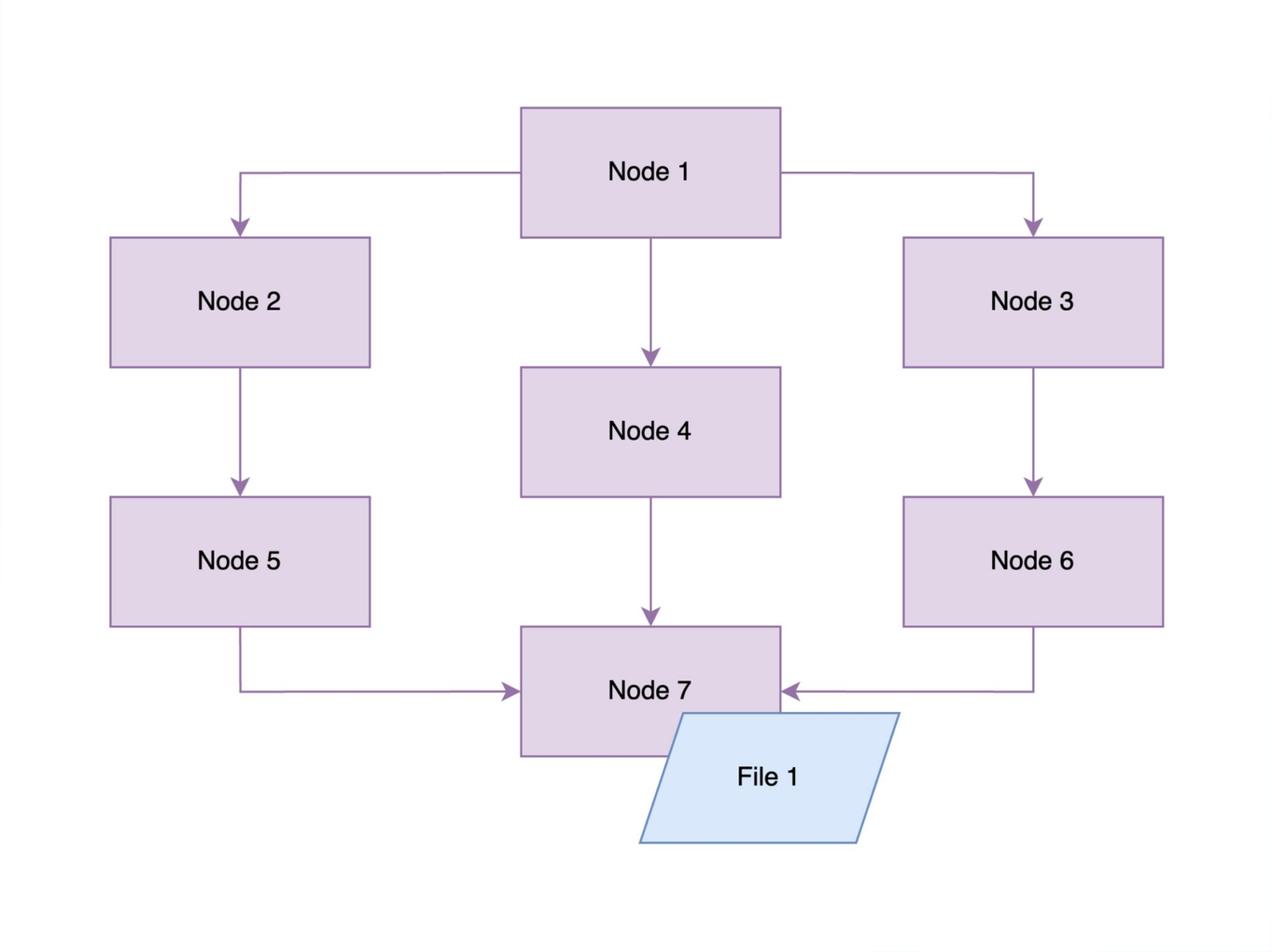
Что такое IPFS?

- IPFS — InterPlanetary File System
- IPFS is an open system to manage data without a central server
- Decentralized
- Distributed
- P2P
- Content addressable
- Immutable
- Distributed Hash Table

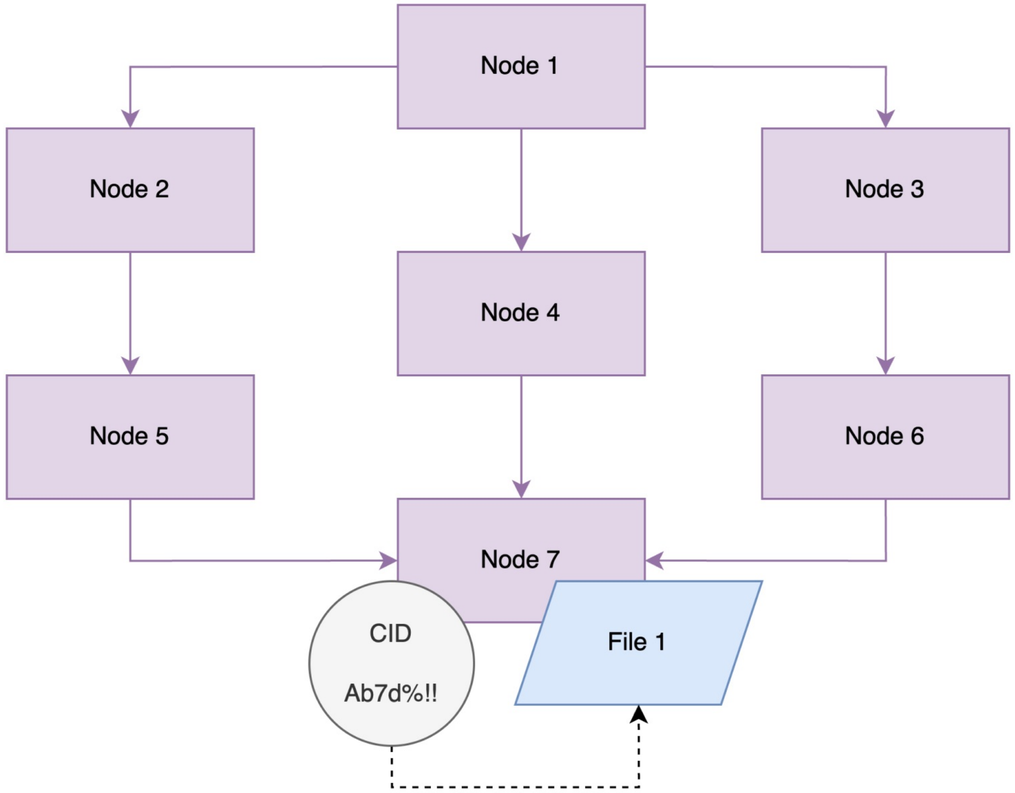
Content addressable, Immutable



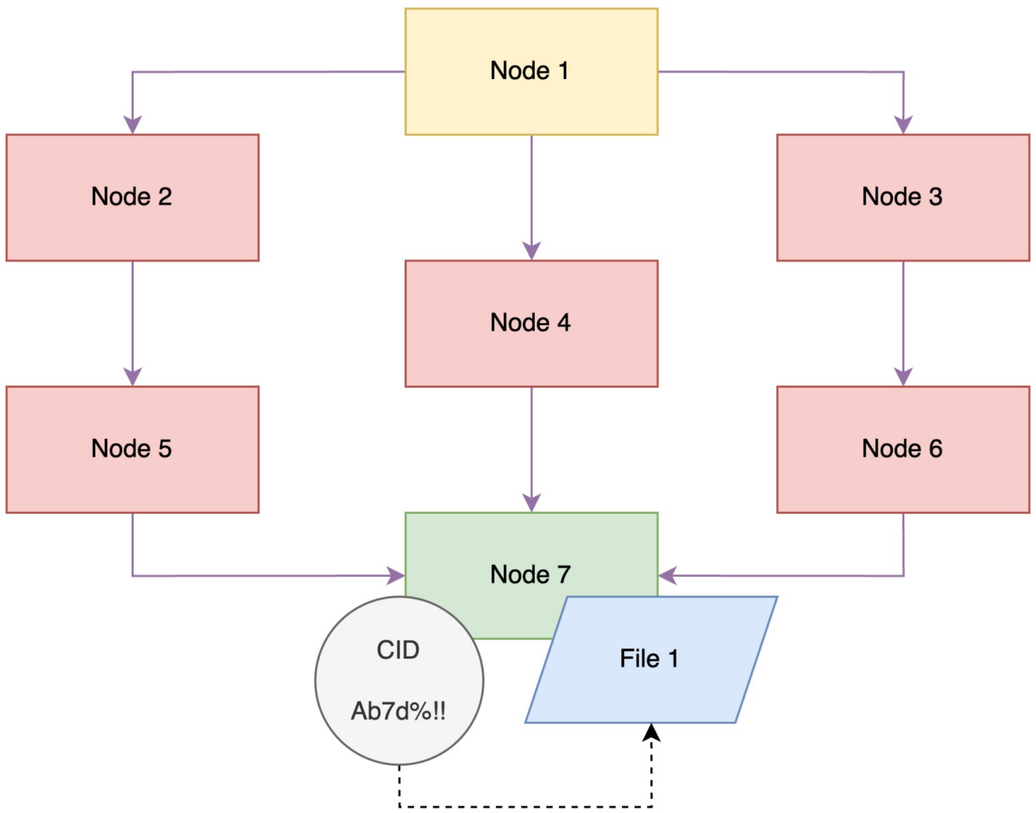
Content addressable, Immutable



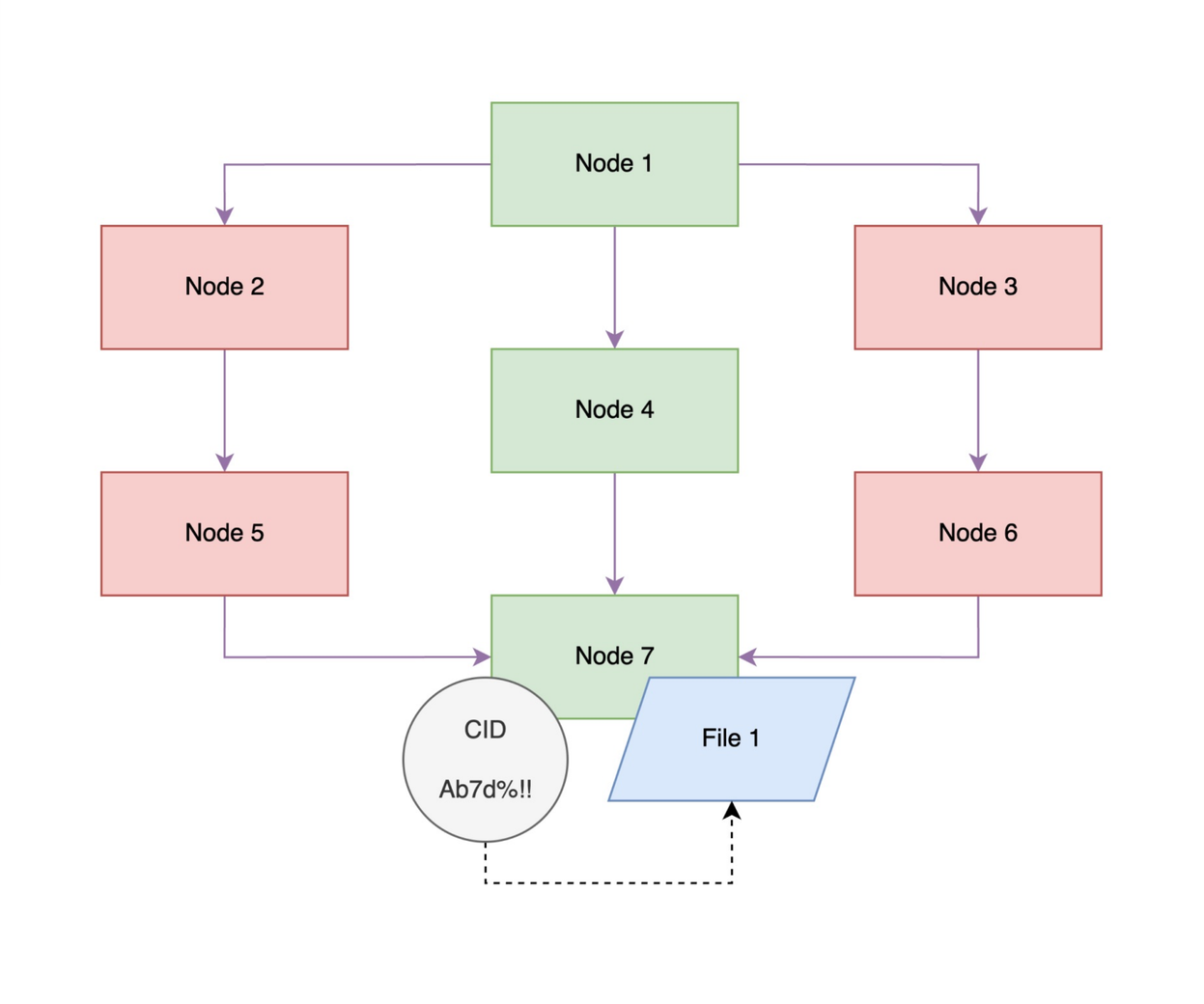
Content addressable, Immutable



Content addressable, Immutable



Content addressable, Immutable



IPFS. Upload npm to Cloudflare GW

TS upload-npm-package-to-ipfs.ts ×

5-ipfs-npm-registry > TS upload-npm-package-to-ipfs.ts > ...

```
1   import { create } from 'ipfs-http-client';
2   import fs from 'fs';
3   import path from 'path';
4
5   // Connect to the IPFS client
6   const ipfs = create({ url: 'https://ipfs.infura.io:5001/api/v0' });
7
8   async function uploadNpmPackageToIPFS(packagePath: string) {
9     try {…
24    } catch (error) {…
26    }
27  }
28
29  // Replace with your package tarball file path
30  const packagePath = './your-package.tgz';
31  uploadNpmPackageToIPFS(packagePath);
32
```

1

2

3

IPFS. Upload npm to Cloudflare GW

TS upload-npm-package-to-ipfs.ts ×

5-ipfs-npm-registry > TS upload-npm-package-to-ipfs.ts > ...

```
8   async function uploadNpmPackageToIPFS(packagePath: string) {
9     try {
10      // Read the package file
11      const packageFile = fs.readFileSync(packagePath);
12      const packageName = path.basename(packagePath);
13
14      // Upload the package file to IPFS
15      const result = await ipfs.add({ path: packageName, content: packageFile });
16
17      // Retrieve the CID for Cloudflare's gateway
18      const ipfsCid = result.cid.toString();
19      console.log(`Uploaded to IPFS with CID: ${ipfsCid}`);
20
21      // Access the package via Cloudflare's IPFS gateway
22      const cloudflareUrl = `https://cloudflare-ipfs.com/ipfs/${ipfsCid}`;
23      console.log(`Access your package at: ${cloudflareUrl}`);
24    } catch (error) {
25      console.error('Failed to upload NPM package to IPFS:', error);
26    }
27  }
28
```

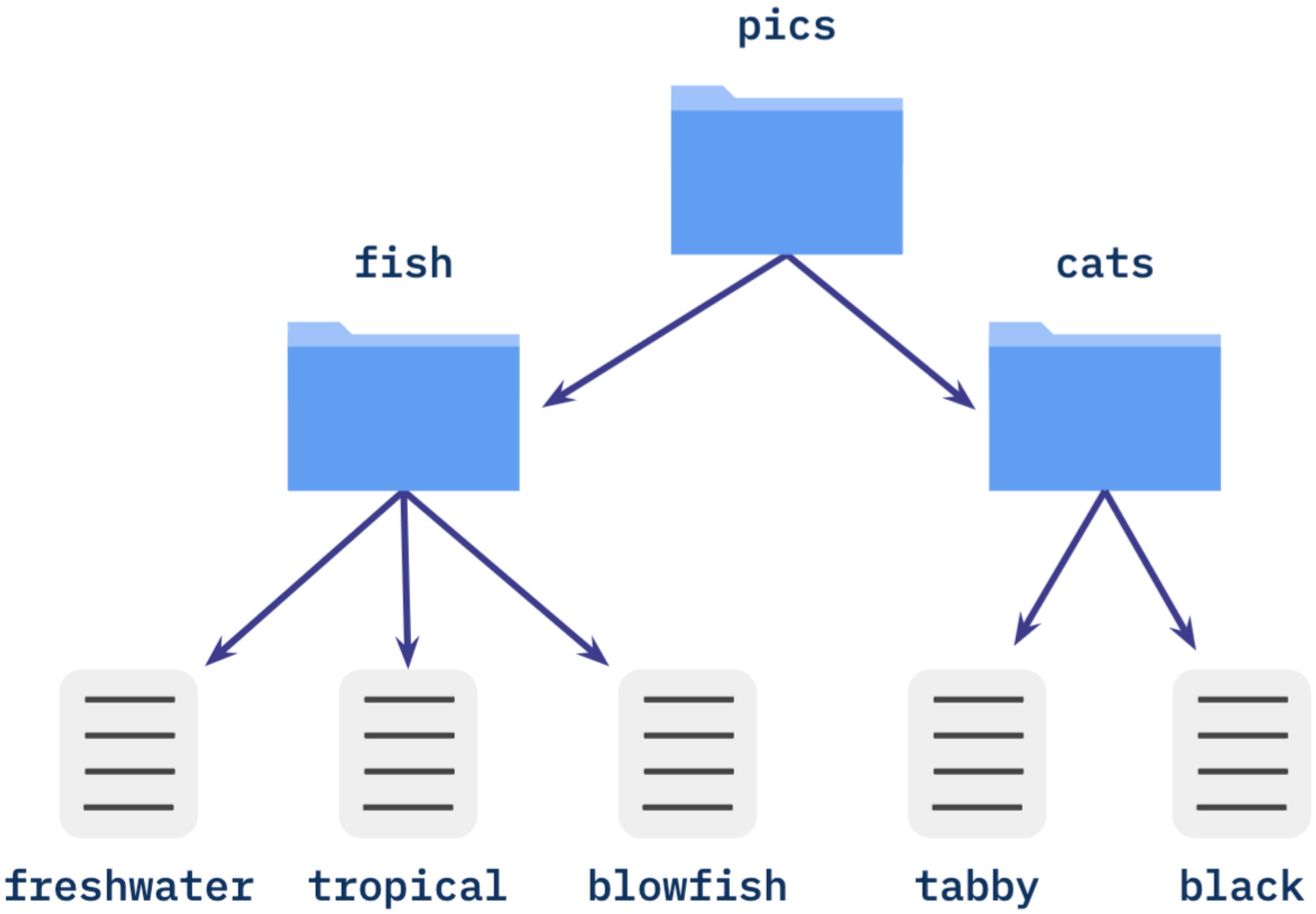
1

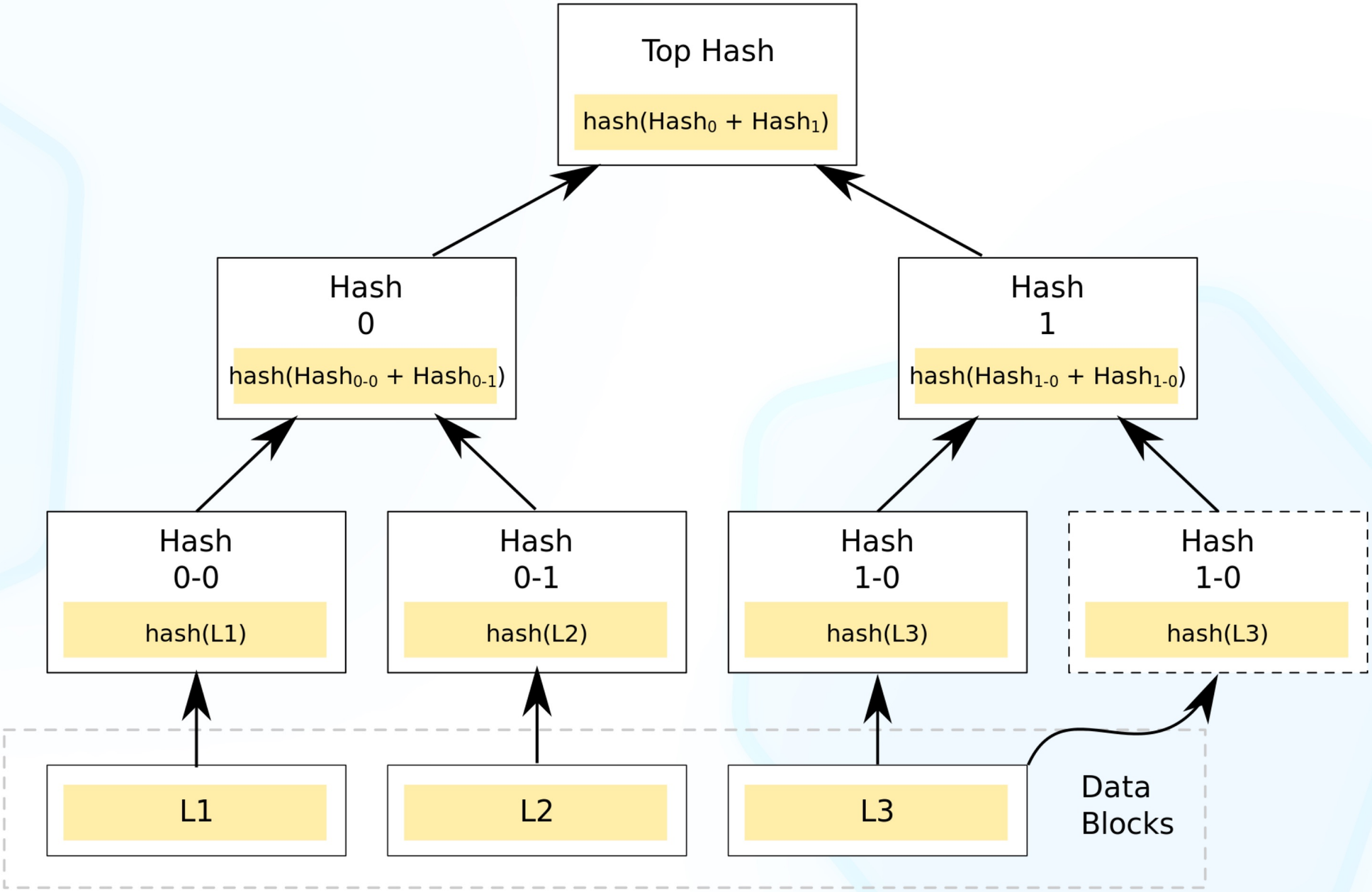
2

3

Что такое IPFS?

- IPFS — InterPlanetary File System
- IPFS is an open system to manage data without a central server
- Decentralized
- Distributed
- P2P
- Content addressable
- Immutable
- Distributed Hash Table
- Merkle Directed Acyclic Graphs


pics
fish
cats
freshwater
tropical
blowfish
tabby
black


Top Hash
hash(Hash0 + Hash1)
Hash
0
hash(Hash0-0 + Hash0-1)
Hash
1
hash(Hash1-0 + Hash1-0)
Hash
0-0
hash(L1)
Hash
0-1
hash(L2)
Hash
1-0
hash(L3)
Hash
1-0
hash(L3)
L1
L2
L3
Data
Blocks

Что такое IPFS?

- IPFS — InterPlanetary File System
- IPFS is an open system to manage data without a central server
- Decentralized
- Distributed
- P2P
- Content addressable
- Immutable
- Distributed Hash Table
- Merkle Directed Acyclic Graphs
- Web 3.0 (decentralized, no censorship and immutable)

IPFS. Store forgotten article with IPFS

TS upload-text-article-to-ipfs.ts ×

6-ipfs-text-article > TS upload-text-article-to-ipfs.ts > ...

```
1   import { create } from 'ipfs-http-client';
2
3   // Connect to the IPFS client (e.g., Infura's public IPFS node)
4   const ipfs = create({ url: 'https://ipfs.infura.io:5001/api/v0' });
5
6 > async function uploadTextArticleToIPFS(title: string, content: string) {…
36  }
37
38  // Example usage
39  const articleTitle = 'My Article Title';
40  const articleContent = 'This is the content of my article. It's now stored on IPFS!';
41  uploadTextArticleToIPFS(articleTitle, articleContent);
42
```

1

2

IPFS. Store forgotten article with IPFS

TS upload-text-article-to-ipfs.ts

6-ipfs-text-article > TS upload-text-article-to-ipfs.ts > ...

```ts
6   async function uploadTextArticleToIPFS(title: string, content: string) {
7     try {
8       // Create an HTML document
9       const htmlContent = `…
24      `;
25
26      // Upload the HTML content to IPFS
27      const result = await ipfs.add(htmlContent);
28      const ipfsCid = result.cid.toString();
29
30      // Access the article via Cloudflare's IPFS gateway
31      const cloudflareUrl = `https://cloudflare-ipfs.com/ipfs/${ipfsCid}`;
32      console.log(`Access your article as a website at: ${cloudflareUrl}`);
33    } catch (error) {
34      console.error('Failed to upload article to IPFS:', error);
35    }
36  }
37
```

1

2

3

IPFS. Store forgotten article with IPFS

https://cloudflare-ipfs.com/ipfs/{CID}

Web 3.0 to Good to be True?

Сообщество формирует технологии,

а не стек формирует сообщество

Сообщество формирует технологии,

а не стек формирует сообщество

JS Community может создавать core логику вокруг AI, БД или IPFS :)

IPFS и безопасность

- **Q**: Почему не использовать блокчейн вместо IPFS?
- **A**: Потому что в блокчейне биты — деньги (высокие ресурсы при вычислении). Слишком дорого

IPFS и безопасность

- Q: Почему не использовать блокчейн вместо IPFS?
- A: Потому что в блокчейне биты — деньги (высокие ресурсы при вычислении). Слишком дорого

- **Q**: Почему в публичную IPFS сеть не выложить зашифрованные файлы, реализуя private data storage
- **A**: Потому что криптография дает безопасность на время, пока не вскрыли. А IPFS — immutable система, дает слишком большое временное окно

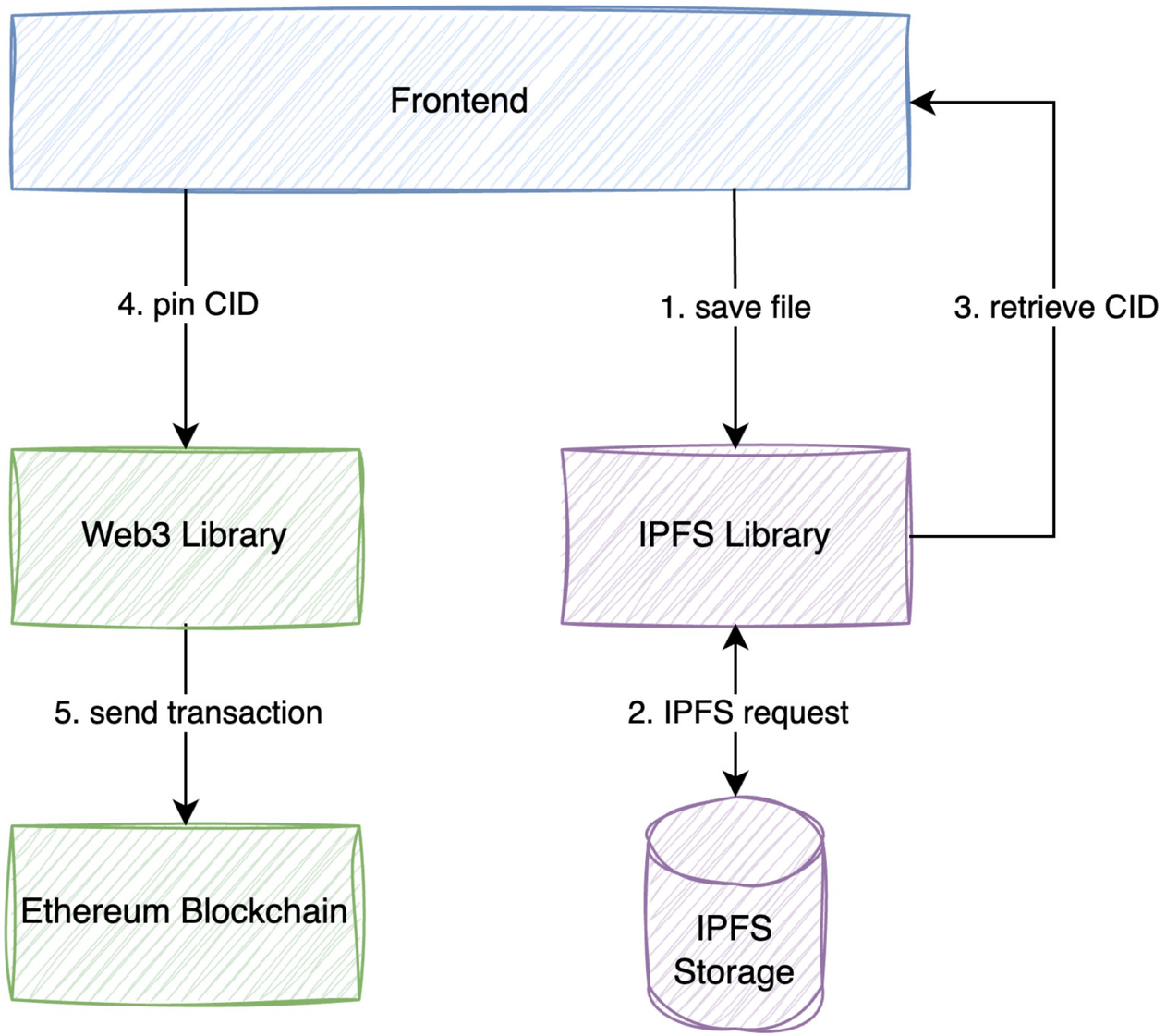
IPFS и безопасность

- Q: Почему не использовать блокчейн вместо IPFS?
- A: Потому что в блокчейне биты — деньги. Слишком дорого

- Q: Почему в публичную IPFS сеть не выложить зашифрованные файлы, реализуя private data storage
- A: Потому что криптография дает безопасность на время, а IPFS дает большое временное окно

- **Правильный вопрос**: как объединить мощь IPFS и блокчейна?
  A: IPFS pinning

IPFS pinning



IPFS. Pinning with Etherium Blockchain

TS pin-cid-to-ethereum.ts ×

7-pin-article-ipfs-cid > TS pin-cid-to-ethereum.ts > ...

```
1   import { ethers } from 'ethers';
2
3   // Configure your provider (Infura, Alchemy, or any Ethereum node provider)
4   const provider = new ethers.providers.InfuraProvider('mainnet', 'YOUR_INFURA_PROJECT_ID');
5
6   // Set up a wallet (make sure to keep the private key secure)
7   const privateKey = 'YOUR_PRIVATE_KEY';
8   const wallet = new ethers.Wallet(privateKey, provider);
9
10  // Function to pin CID to Ethereum
11 > async function pinCidToEthereum(cid: string) {…
30  }
31
32  // Example usage
33  const cid = 'QmYwAPJzv5CZsnAzt8auVZRn2BLyPjcVWx8JrKH5kEG7zn'; // Replace with the actual CID
34  pinCidToEthereum(cid);
35
```

1

2

3

IPFS. Pinning with Etherium Blockchain

TS pin-cid-to-ethereum.ts ×

7-pin-article-ipfs-cid > TS pin-cid-to-ethereum.ts > ...

```ts
// Function to pin CID to Ethereum
async function pinCidToEthereum(cid: string) {
  try {
    // Prepare the transaction
    const tx = {
      to: ethers.constants.AddressZero, // Sending to null address since we're only storing data
      value: ethers.utils.parseEther("0.0"), // No ETH is being transferred, only data
      data: ethers.utils.hexlify(ethers.utils.toUtf8Bytes(cid)) // Convert CID to hex format
    };

    // Send the transaction
    const transactionResponse = await wallet.sendTransaction(tx);
    console.log(`Transaction sent! Hash: ${transactionResponse.hash}`);

    // Wait for the transaction to be confirmed
    const receipt = await transactionResponse.wait();
    console.log(`Transaction confirmed in block ${receipt.blockNumber}`);
  } catch (error) {
    console.error('Failed to pin CID to Ethereum:', error);
  }
}
```

1

2

3

Зачем IPFS для веба?

- Content Addressable
  - Web 2.0
    - https://domain.com/something
    - DNS
    - Mutable content
    - Hosting
  - IPFS
    - CID
    - /ipfs/%CID%
    - Immutable
    - Multiple hosters

Зачем IPFS для веба?

- Content Addressable
- Ready to use infrastructure via gateways
  - IPFS gateway is a web-based service that gets content from an IPFS network (private, or the public swarm backed by Amino DHT), and makes it available via HTTP
    - https://ipfs.io
    - https://developers.cloudflare.com/web3/ipfs-gateway/
    - https://web3.storage/

Зачем IPFS для веба?

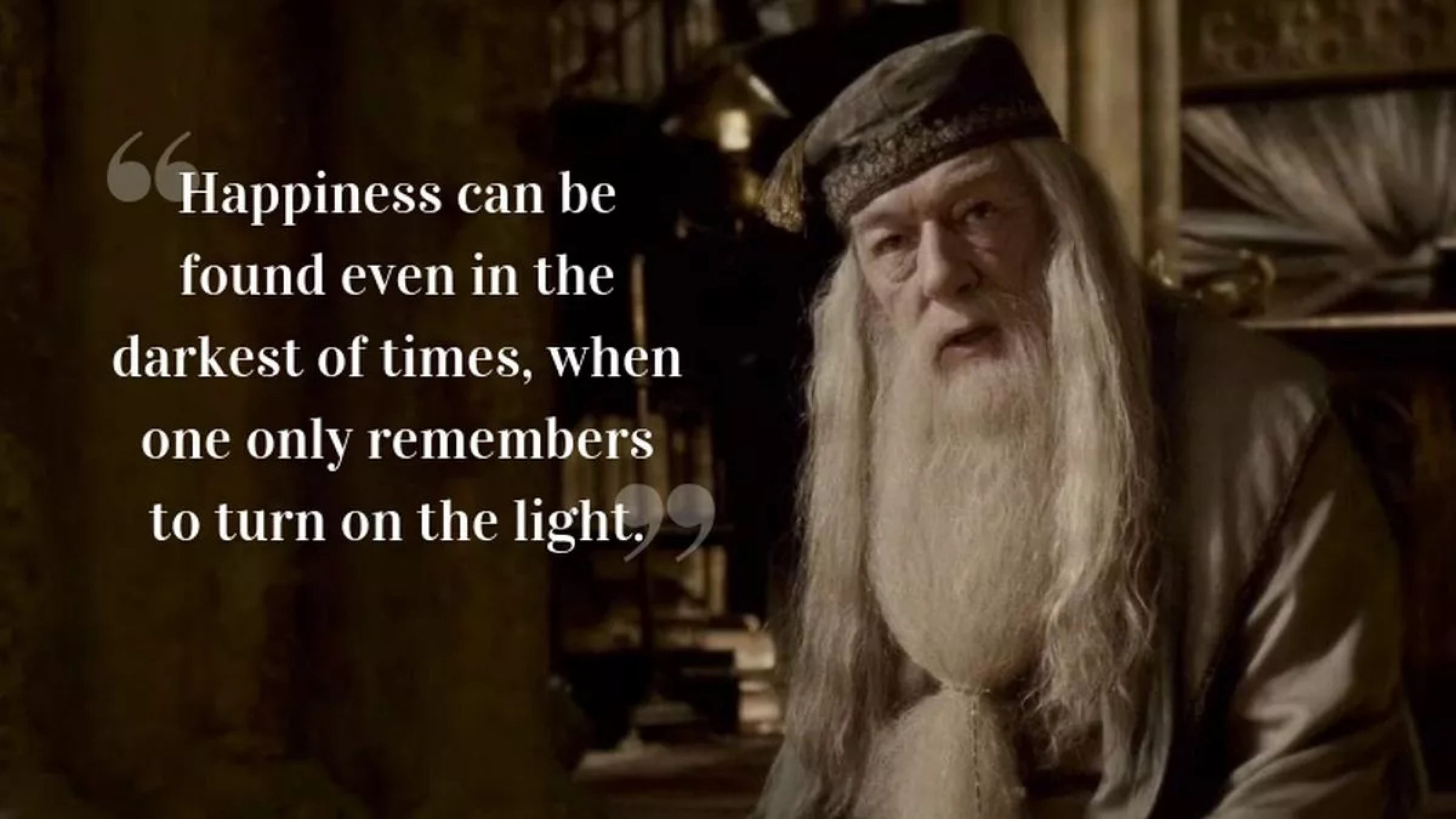
- Content Addressable
- Ready to use infrastructure via gateways
- Open Source by design: developed by community for community
  - вместо объяснений примеры-аналогии:
    - HTTP
    - SSL
    - Linux

Зачем IPFS для веба?

- Content Addressable
- Ready to use infrastructure via gateways
- Open Source by design: developed by community for community
- Use Cases
  - App Hosting
  - Distributed package management
  - High availability public knowledge

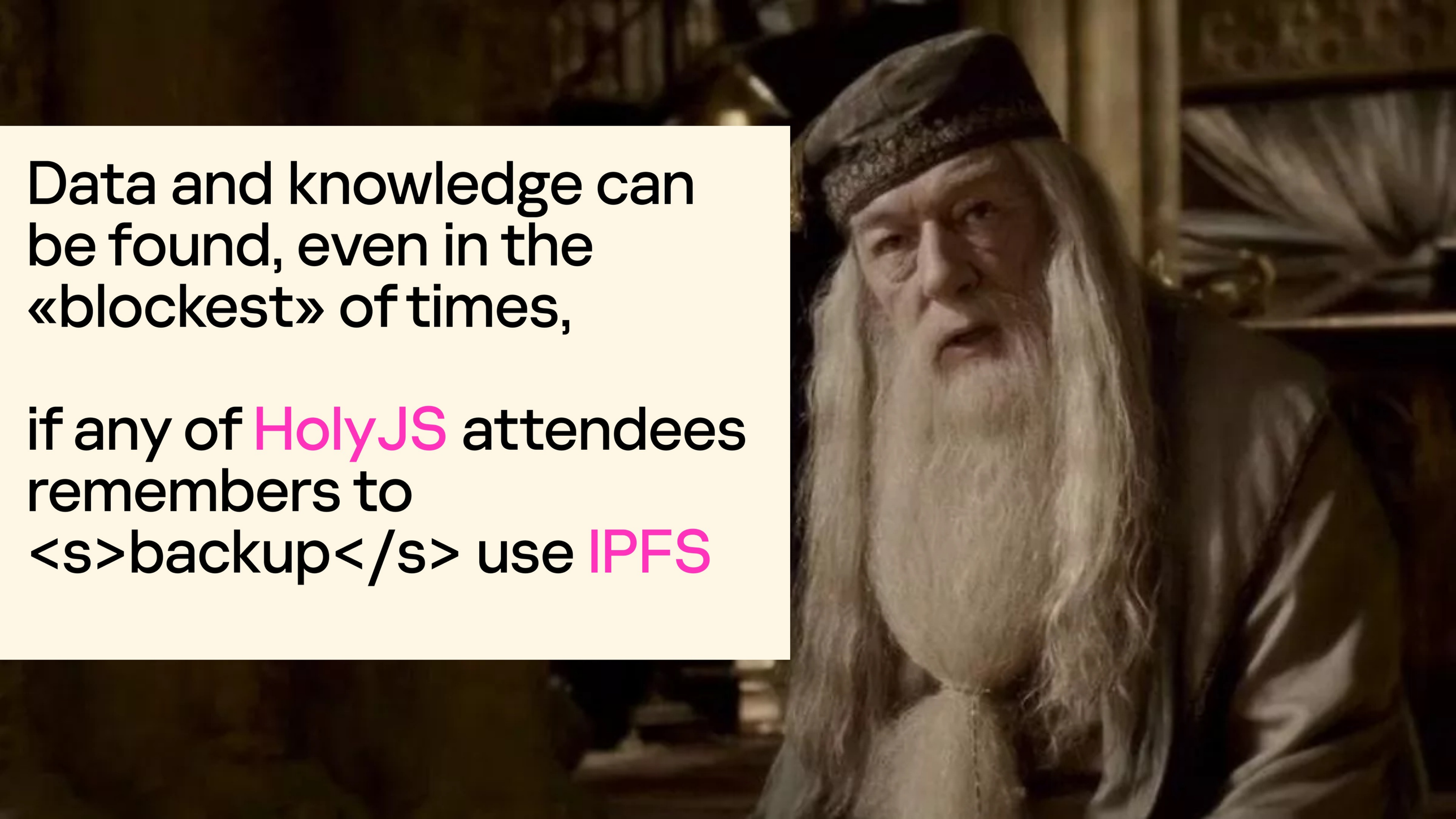
Зачем IPFS для веба?

- Content Addressable
- Ready to use infrastructure via gateways
- Open Source by design: developed by community for community
- Use Cases
  - App Hosting
  - Distributed package management
  - High availability public knowledge
- Архитектурно гарантированная передача знаний


“Happiness can be found even in the darkest of times, when one only remembers to turn on the light.”

Data and knowledge can be found, even in the «blockest» of times,

if any of HolyJS attendees remembers to <s>backup</s> use IPFS



Соберем всё вместе

Local-first and IPFS: Use Cases

- Decrease TTI

Local-first and IPFS: Use Cases

- Decrease TTI
- Collaboration

Local-first and IPFS: Use Cases

- Decrease TTI
- Collaboration
- Offline mode

Local-first and IPFS: Use Cases

- Decrease TTI
- Collaboration
- Offline mode
- Pin with blockchain: verification

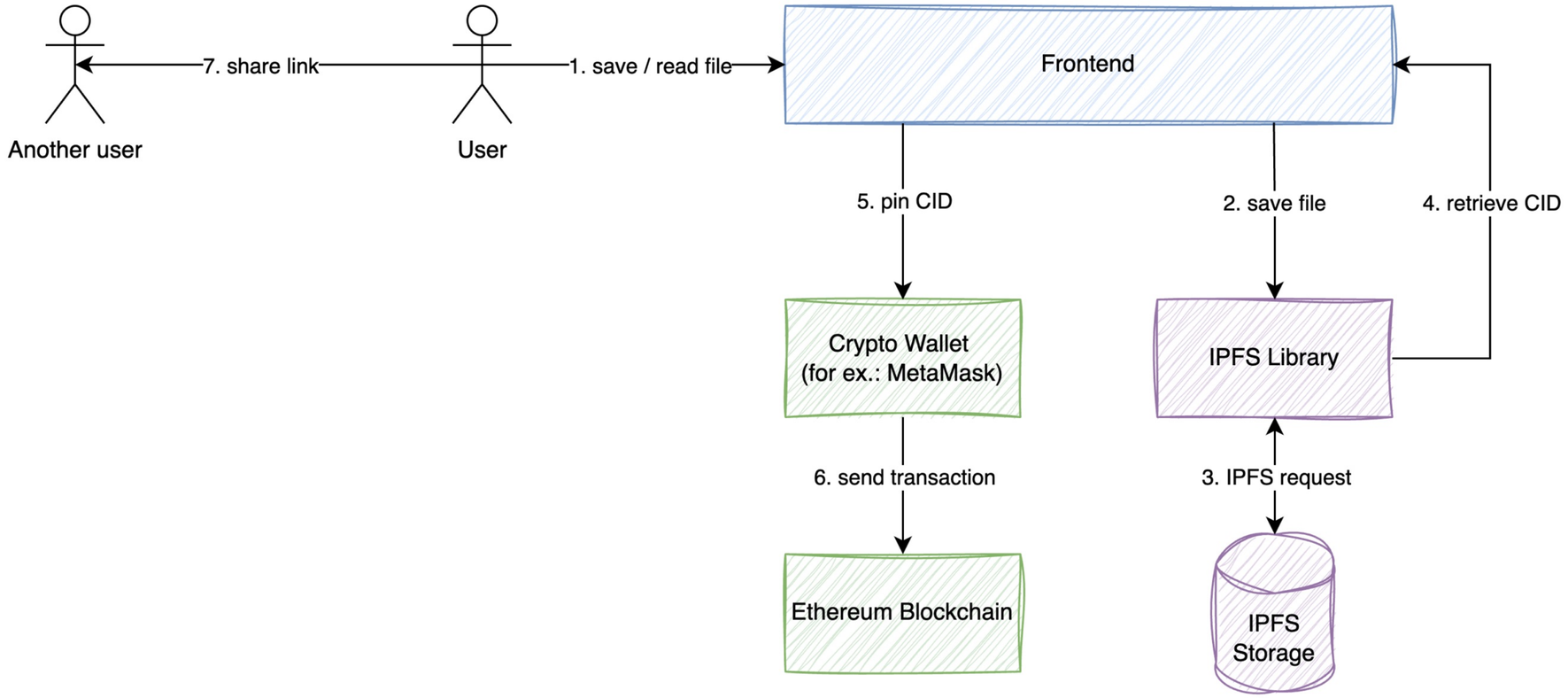
Local-first and IPFS: Use Cases

- Decrease TTI
- Collaboration
- Offline mode
- Pin with blockchain: verification
- Content storage: immutable and decentralized

Local-first and IPFS: Use Cases

- Decrease TTI
- Collaboration
- Offline mode
- Pin with blockchain: verification
- Content storage: immutable and decentralized

Final HLA (High Level Architecture)



Мои данные — это мои данные

Мои данные —
это мои данные,
лежащие в несгораемом сейфе
на устройстве, не
подключенном к сети*

* summary после 5.5 лет обучения ИБ

Мои данные —
это мои данные,
лежащие в несгораемом сейфе
на устройстве, не
подключенном к сети**

* summary после 5.5 лет обучения ИБ

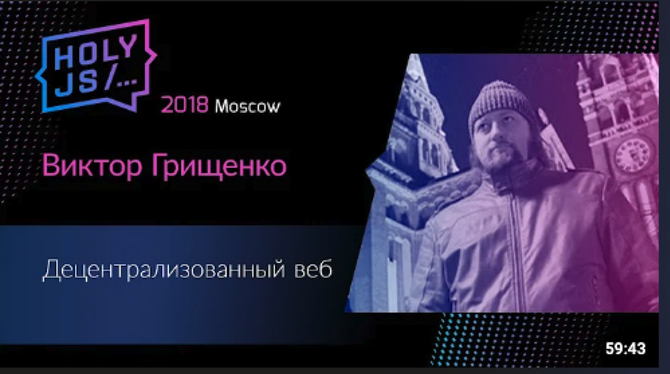
** и то упрут

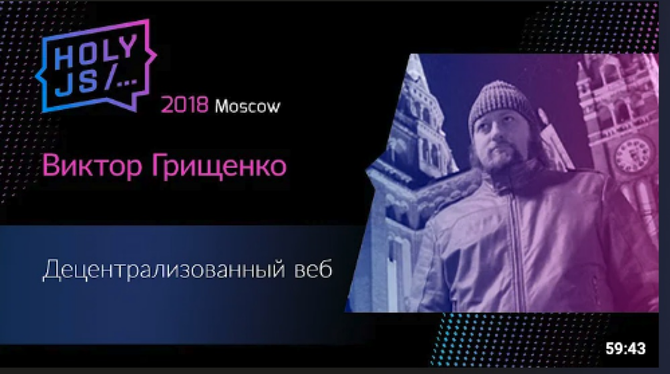
Материалы

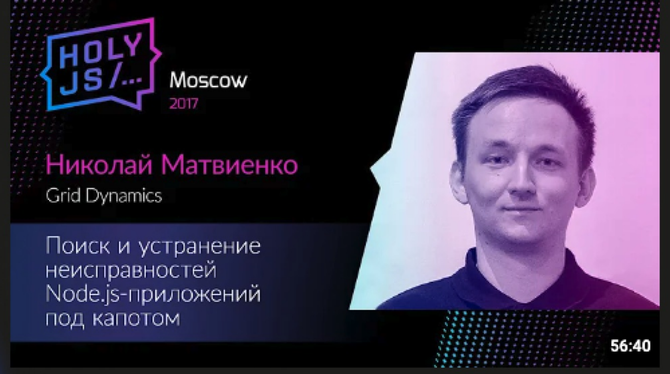
Ссылка на презентацию, все материалы и мои контакты

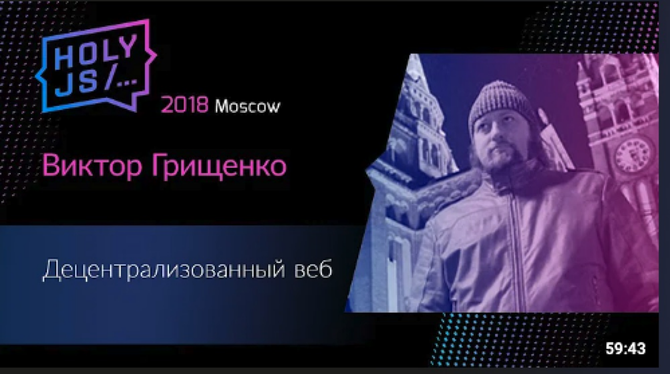
Post Scriptum

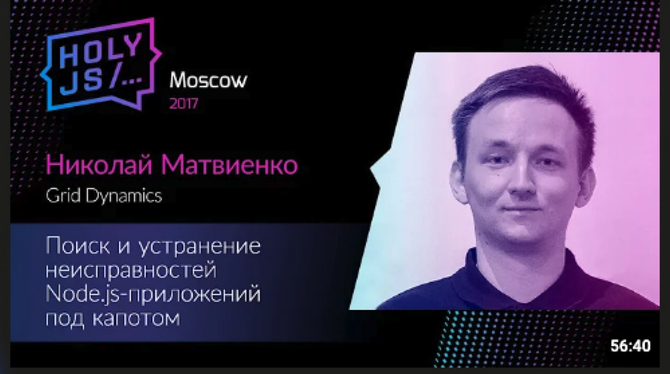
Douglas
Crockford
SPEAKER

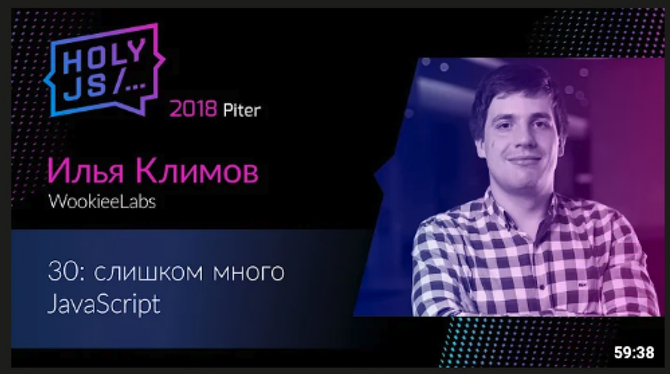

HOLY JS/...
2018 Moscow
Виктор Грищенко
Децентрализованный веб
59:43

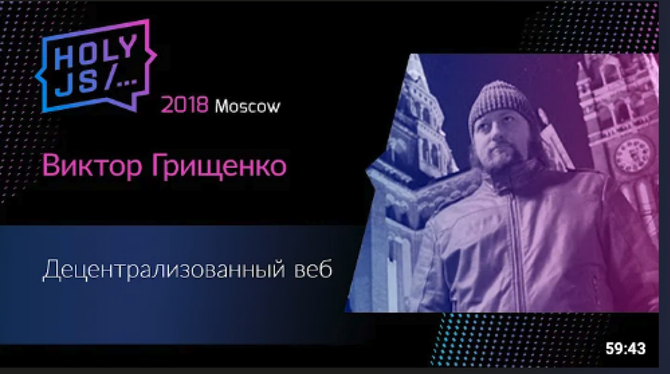

HOLY JS/...
2018 Moscow
Виктор Грищенко
Децентрализованный веб
59:43

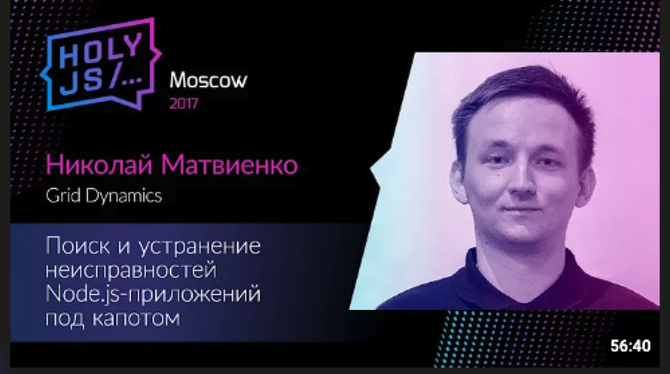

HOLY JS/...
Moscow
2017
Николай Матвиенко
Grid Dynamics
Поиск и устранение неисправностей Node.js-приложений под капотом
56:40

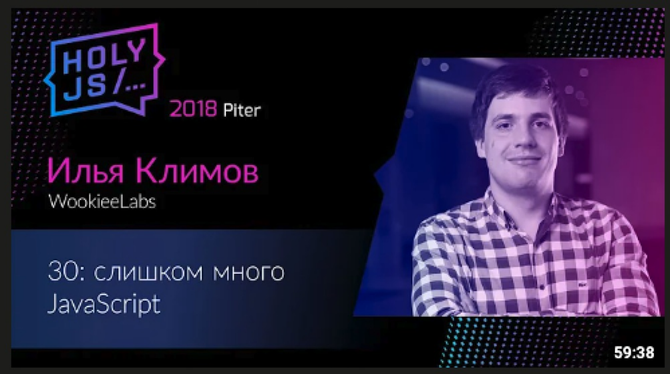

HOLY JS/...
2018 Moscow
Виктор Грищенко
Децентрализованный веб
59:43


HOLY JS/...
Moscow
2017
Николай Матвиенко
Grid Dynamics
Поиск и устранение неисправностей Node.js-приложений под капотом
56:40

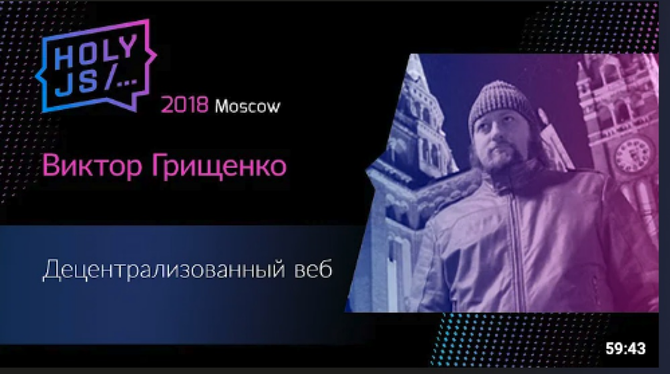

HOLY JS/...
2018 Piter
Илья Климов
WookieeLabs
30: слишком много JavaScript
59:38

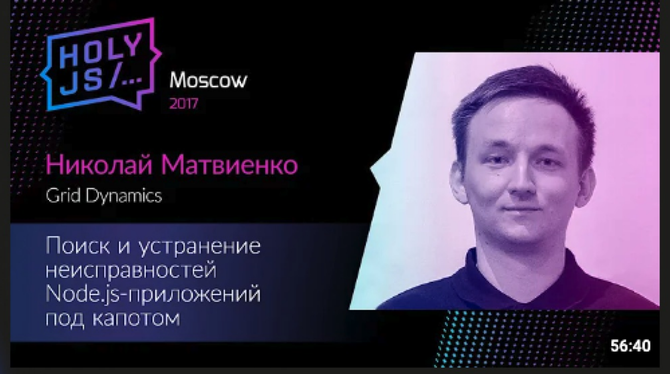

HOLY JS/...
2018 Moscow
Виктор Грищенко
Децентрализованный веб
59:43

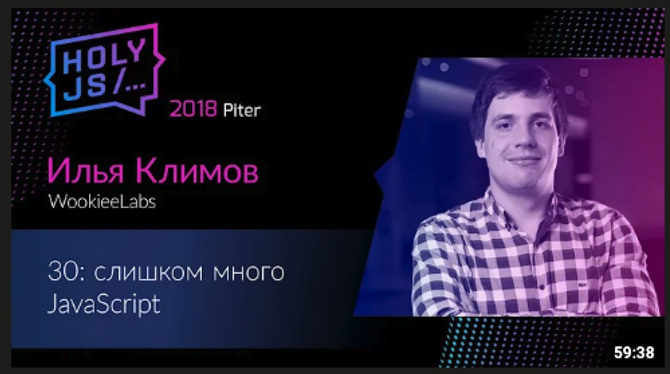

HOLY JS/...
Moscow
2017
Николай Матвиенко
Grid Dynamics
Поиск и устранение неисправностей Node.js-приложений под капотом
56:40


HOLY JS/...
2018 Piter
Илья Климов
WookieeLabs
30: слишком много JavaScript
59:38

HOLY JS/...
2019 MOSCOW
Андрей Оконечников
Component-Driven
Resilient Component Libraries with React
1:00:51

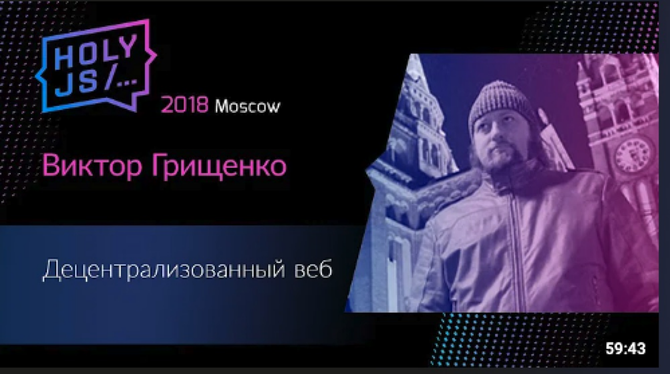

HOLY JS/...
2018 Moscow
Виктор Грищенко
Децентрализованный веб
59:43

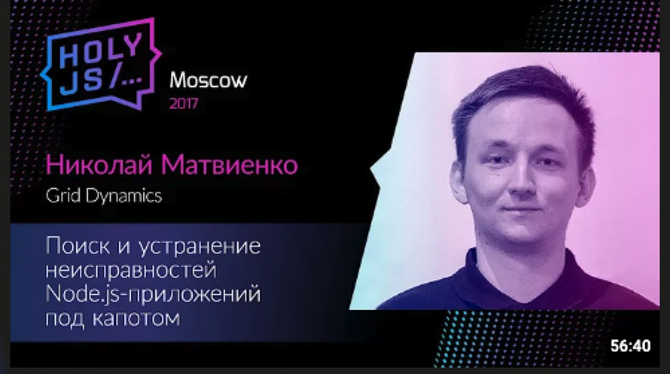

HOLY JS/...
Moscow
2017
Николай Матвиенко
Grid Dynamics
Поиск и устранение неисправностей Node.js-приложений под капотом
56:40

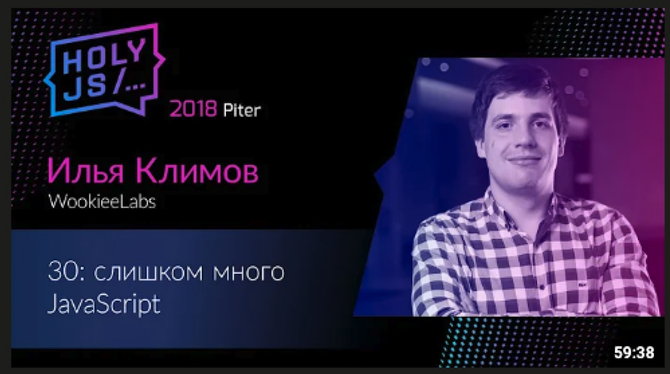

HOLY JS/...
2018 Piter
Илья Климов
WookieeLabs
30: слишком много JavaScript
59:38

HOLY JS/...
2019 MOSCOW
Андрей Оконечников
Component-Driven
Resilient Component Libraries with React
1:00:51

HOLY JS/...
2019 MOSCOW
Guillermo Rauch
ZEIT
Client Rendering, server Rendering, pre Rendering. The full spectrum of JS website and application performance delivery
52:05


HOLY JS/...
2018 Moscow
Виктор Грищенко
Децентрализованный веб
59:43

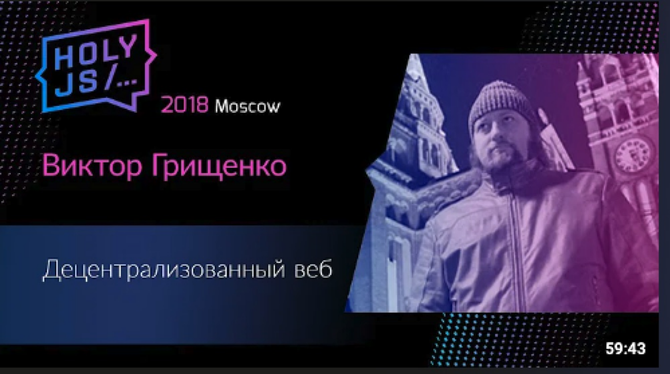

HOLY JS/...
Moscow
2017
Николай Матвиенко
Grid Dynamics
Поиск и устранение неисправностей Node.js-приложений под капотом
56:40

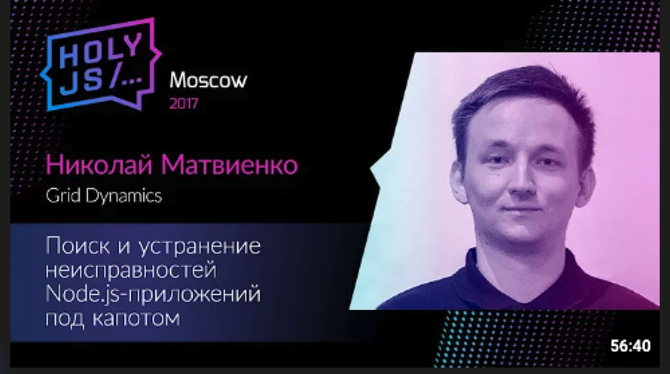

HOLY JS/...
2018 Piter
Илья Климов
WookieeLabs
30: слишком много JavaScript
59:38

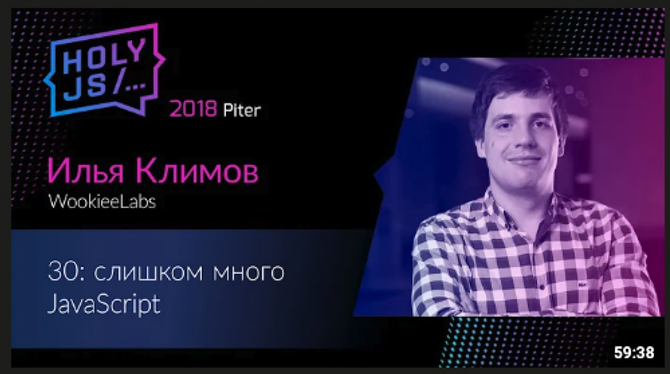
HOLY JS/...
2019 MOSCOW
Андрей Оконечников
Component-Driven
Resilient Component Libraries with React
1:00:51

HOLY JS/...
2019 MOSCOW
Guillermo Rauch
ZEIT
Client Rendering, server Rendering, pre Rendering. The full spectrum of JS website and application performance delivery
52:05

HOLY JS/...
2019 Piter
Андрей Ситник
Evil Martians
Продвижение опенсорс-проектов
42:39


HOLY JS/...
2018 Moscow
Виктор Грищенко
Децентрализованный веб
59:43

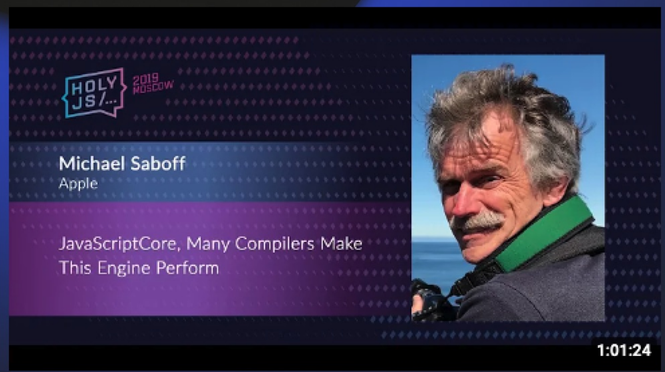

HOLY JS/...
Moscow
2017
Николай Матвиенко
Grid Dynamics
Поиск и устранение неисправностей Node.js-приложений под капотом
56:40

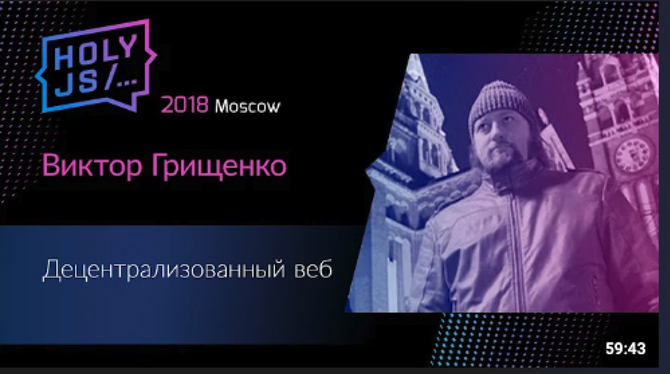

HOLY JS/...
2018 Piter
Илья Климов
WookieeLabs
30: слишком много JavaScript
59:38

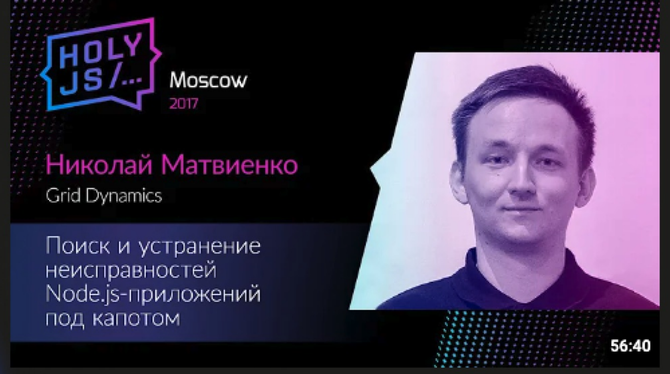
HOLY JS/...
2019 MOSCOW
Андрей Оконечников
Component-Driven
Resilient Component Libraries with React
1:00:51

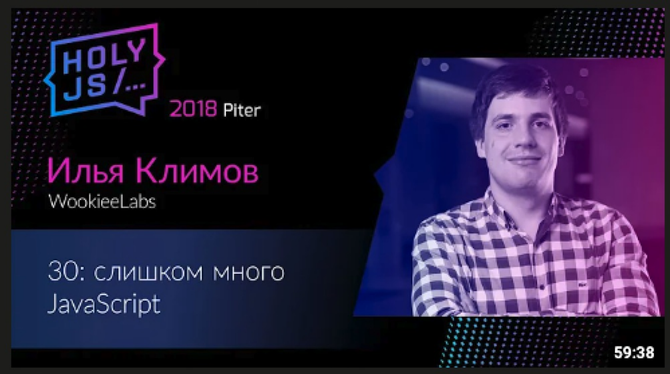
HOLY JS/...
2019 MOSCOW
Guillermo Rauch
ZEIT
Client Rendering, server Rendering, pre Rendering. The full spectrum of JS website and application performance delivery
52:05

HOLY JS/...
2019 Piter
Андрей Ситник
Evil Martians
Продвижение опенсорс-проектов
42:39

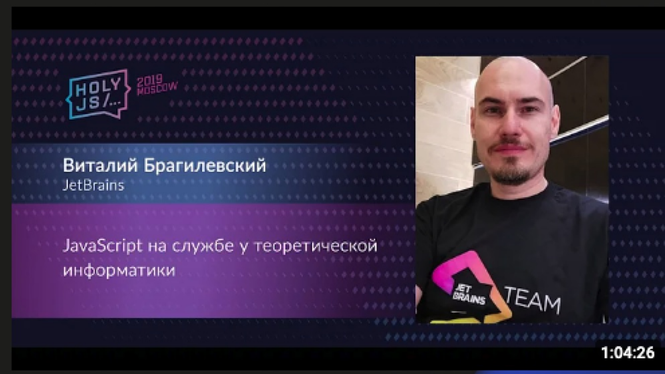

HOLY JS/...
2019 MOSCOW
Michael Saboff
Apple
JavaScriptCore, Many Compilers Make This Engine Perform
1:01:24


HOLY JS/...
2018 Moscow
Виктор Грищенко
Децентрализованный веб
59:43


HOLY JS/...
Moscow
2017
Николай Матвиенко
Grid Dynamics
Поиск и устранение неисправностей Node.js-приложений под капотом
56:40

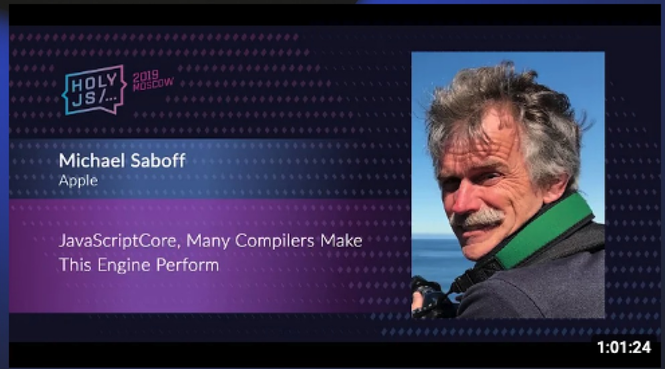

HOLY JS/...
2018 Piter
Илья Климов
WookieeLabs
30: слишком много JavaScript
59:38

HOLY JS/...
2019 MOSCOW
Андрей Оконечников
Component-Driven
Resilient Component Libraries with React
1:00:51

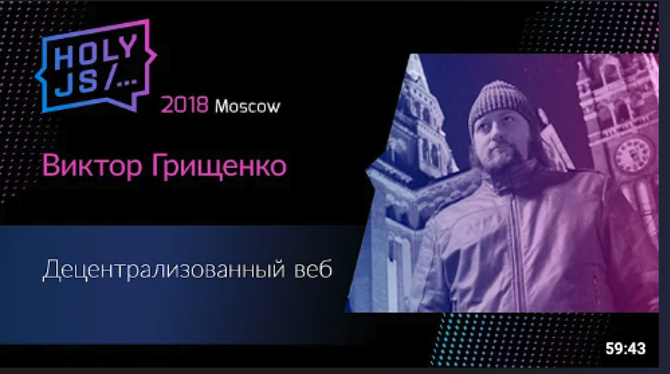

HOLY JS/...
2019 MOSCOW
Виталий Брагилевский
JetBrains
JavaScript на службе у теоретической информатики
TEAM
1:04:26

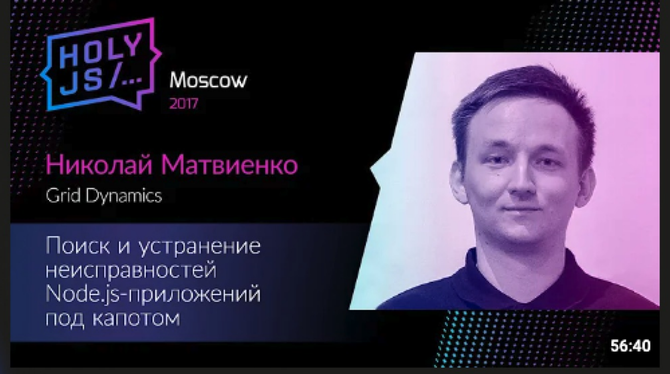
HOLY JS/...
2019 MOSCOW
Guillermo Rauch
ZEIT
Client Rendering, server Rendering, pre Rendering. The full spectrum of JS website and application performance delivery
52:05

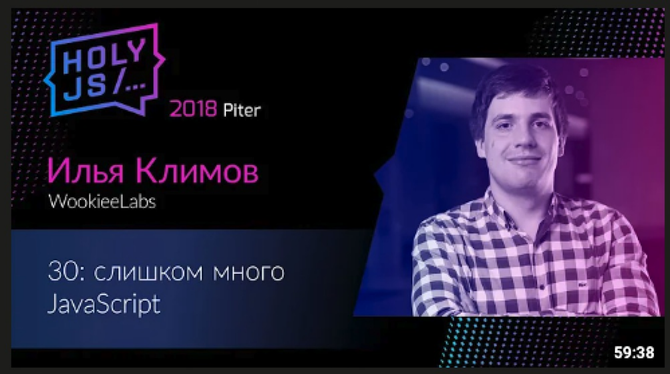
HOLY JS/...
2019 Piter
Андрей Ситник
Evil Martians
Продвижение опенсорс-проектов
42:39


HOLY JS/...
2019 MOSCOW
Michael Saboff
Apple
JavaScriptCore, Many Compilers Make This Engine Perform
1:01:24

Да никак, конечно. Просто максимум можете умным человеком стать.
Как все рассказанное может помочь мне как джаваскриптеру?

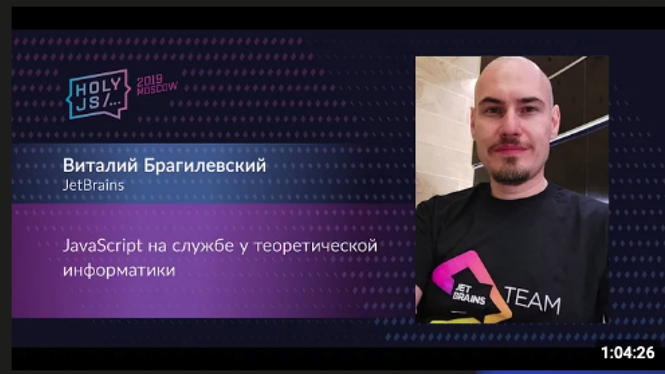

HOLY JS/...
2018 Moscow
Виктор Грищенко
Децентрализованный веб
59:43


HOLY JS/...
Moscow
2017
Николай Матвиенко
Grid Dynamics
Поиск и устранение неисправностей Node.js-приложений под капотом
56:40


HOLY JS/...
2018 Piter
Илья Климов
WookieeLabs
30: слишком много JavaScript
59:38

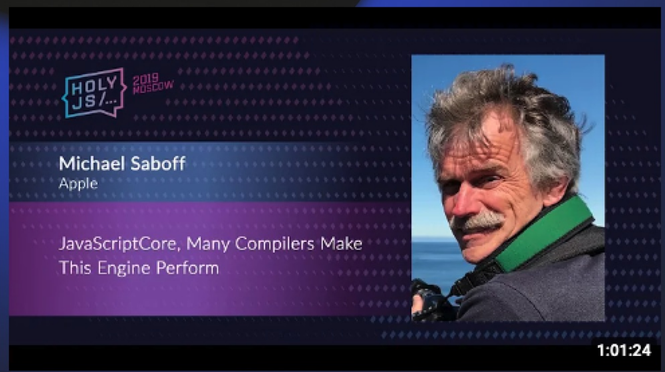
HOLY JS/...
2019 MOSCOW
Андрей Оконечников
Component-Driven
Resilient Component Libraries with React
1:00:51

36
1:06:31


HOLY JS/...
2019 MOSCOW
Виталий Брагилевский
JetBrains
JavaScript на службе у теоретической информатики
JET BRAINS
TEAM
1:04:26

HOLY JS/...
2019 MOSCOW
Guillermo Rauch
ZEIT
Client Rendering, server Rendering, pre Rendering. The full spectrum of JS website and application performance delivery
52:05

HOLY JS/...
2019 Piter
Андрей Ситник
Evil Martians
Продвижение опенсорс-проектов
42:39


HOLY JS/...
2019 MOSCOW
Michael Saboff
Apple
JavaScriptCore, Many Compilers Make This Engine Perform
1:01:24

Таймлайн работы: 2010–2024–...

- 28 ноября 2010 первый день работы на первой работе веб-программистом

Таймлайн работы: 2010–2024–...

- 28 ноября 2010 первый день работы на первой работе веб-программистом

- ...
- приходит мысль «почему бы не работать по 10–12–14–16 часов в день» что будет плохого-то? просто буду много успевать
- ...

Таймлайн работы: 2010–2024–...

- 28 ноября 2010 первый день работы на первой работе веб-программистом
- ...
- приходит мысль «почему бы не работать по 10–12–14–16 часов в день»
  что будет плохого-то? просто буду много успевать
- ...
- 15 марта 2024 на минуту у меня останавливалось дыхание
  - две скорых
  - две больницы
  - жить буду

Таймлайн работы: 2010–2024–...

- 28 ноября 2010 первый день работы на первой работе веб-программистом
- ...
- приходит мысль «почему бы не работать по 10–12–14–16 часов в день»
  что будет плохого-то? просто буду много успевать
- ...
- 15 марта 2024 на минуту у меня останавливалось дыхание
  - две скорых
  - две больницы
  - жить буду

- 15 апреля 2024 выступаю на Холи про фиче-флаги
  - https://www.youtube.com/watch?v=pv3Z_oa5AnU

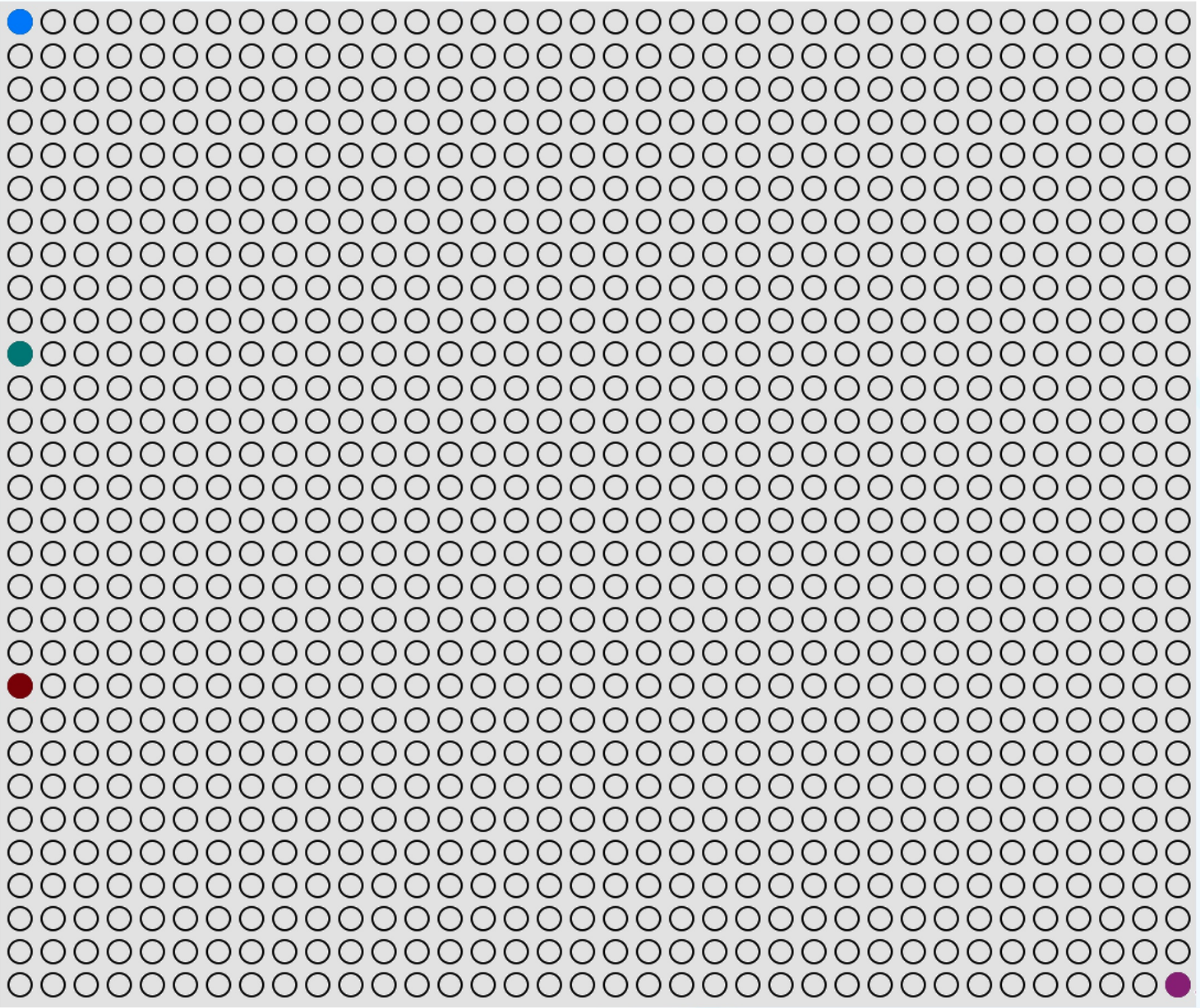
Таймлайн работы: 2010–2024–...

- 28 ноября 2010 первый день работы на первой работе веб-программистом
- ...
- приходит мысль «почему бы не работать по 10–12–14–16 часов в день» что будет плохого-то? просто буду много успевать
- ...
- 15 марта 2024 на минуту у меня останавливалось дыхание
  - две скорых
  - две больницы
  - жить буду
- 15 апреля 2024 выступаю на Холи про фиче-флаги
  - https://www.youtube.com/watch?v=pv3Z_oa5AnU
- 16 апреля 2024 операция

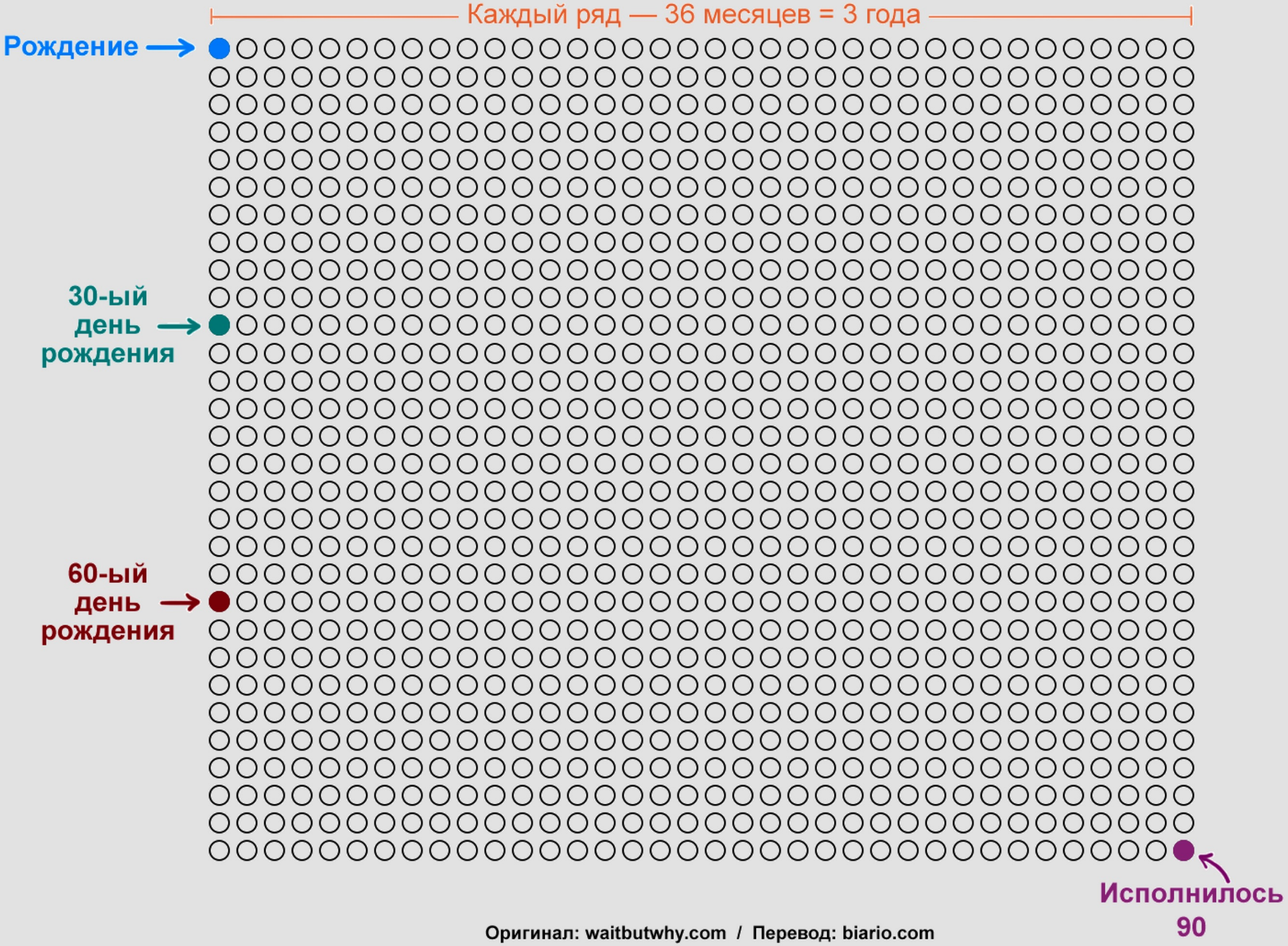
Таймлайн работы: 2010–2024–...

- 28 ноября 2010 первый день работы на первой работе веб-программистом
- ...
- приходит мысль «почему бы не работать по 10–12–14–16 часов в день»
  что будет плохого-то? просто буду много успевать
- ...
- 15 марта 2024 на минуту у меня останавливалось дыхание
  - две скорых
  - две больницы
  - жить буду
- 15 апреля 2024 выступаю на Холи про фиче-флаги
  - https://www.youtube.com/watch?v=pv3Z_oa5AnU
- 16 апреля 2024 операция
- 14 ноября 2024 я стою перед вами
  – на 100 кг меньше
  – с лучшим качеством жизни и здоровья
  – работая не более 8 часов

Фото с 15.04.2024



Жизнь 90-летнего человека по месяцам



Оригинал: waitbutwhy.com / Перевод: biario.com

Сегодня

@JUNIOR_SOUS_CHEF

Holy Holy Holy

Instrumental Sax

Сергей Карпов

Материалы

Ссылка на презентацию, все материалы и мои контакты

Спасибо!