

Лекция 8

Современный фронтенд-разработчик

FRONT-END DEVELOPER

THE LIGHTBULB
IS BROKEN



COVERS LIGHTBULB
WITH YELLOW PAINT



TELLS BACKEND THAT
LIGHTBULB WORKS, BUT
ONLY DURING THE DAY



Service Workers

Подробнее тут <https://frontend.tech-mail.ru/slides/s6/#65>

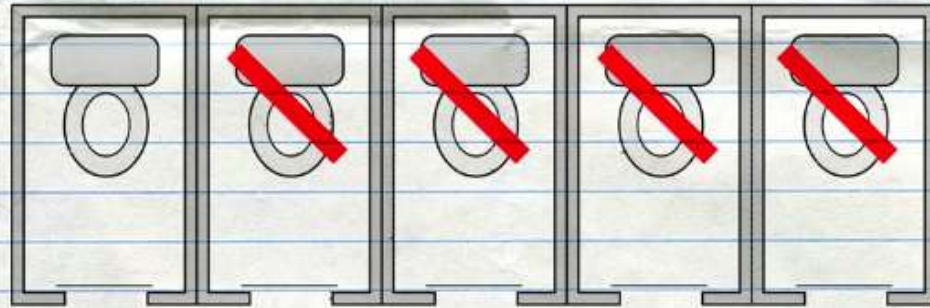
Service Workers

<https://www.youtube.com/watch?v=cmGr0RszHc8>

Building offline-first Progressive Web Apps

Service Workers

Fig 1: Turdo-dynamics



Service Workers

Fig 1: Turdo-dynamics



Что это?

Service Workers — продвинутая технология, которая позволяет получить полный контроль над жизненным циклом приложения. Сервис воркер — это воркер, который:

- работает в выделенном контексте и отдельном потоке
- имеет доступ к Cache Storage (расширенное хранилище данных)
- имеет возможность перехватывать HTTP-запросы, отправляемые страницей
- а так же **может работать даже если само web-приложение или даже браузер не запущены**

Где используется?

- Кэширование данных (до 50% диска)
- **Оффлайн-работа приложения** (прямо вообще без Интернета чтобы было!)
- Фоновая синхронизация данных
- Ответ на запросы от других источников ("проксирование" запросов)
- Улучшение производительности
- Push-нотификации
- Распараллеливание вычислений

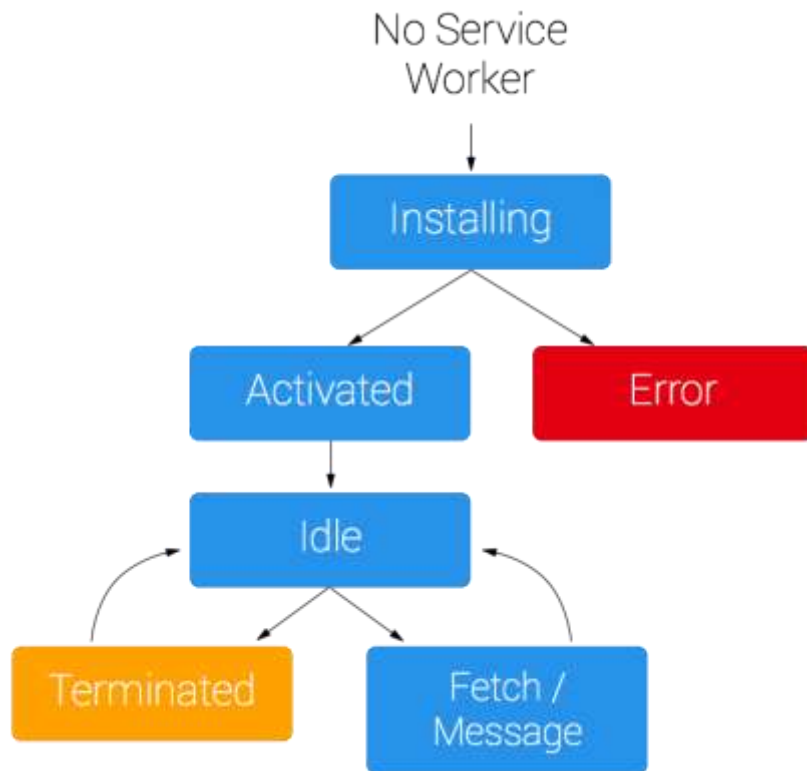
Service Worker

Работает в специальном скоупе **ServiceWorkerGlobalScope**, который не имеет доступа к обычному скоупу с **window**

Имеет несколько событий, на которые можно добавлять обработчики:

```
1 this.addEventListener('install', listener); // SW зарегистрировали
2 this.addEventListener('activate', listener); // SW запустили
3 this.addEventListener('fetch', listener); // SW перехватил запрос
4 this.addEventListener('message', listener); // SW получил сообщение
5 this.addEventListener('push', listener); // SW получил push
6
```

Lifecycle

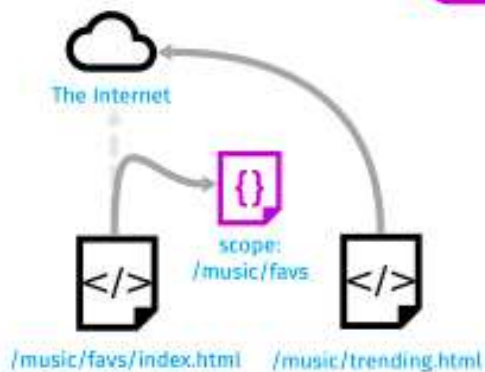


Установка

```
1 navigator.serviceWorker.register('/sw.js', { scope: '/' })
2   .then(function(registration) {
3     // Registration was successful
4     console.log('SW registration OK:', registration);
5   })
6   .catch(function(err) {
7     // registration failed :(
8     console.log('SW registration FAIL:', err);
9   });
10 });
11
```

Важно

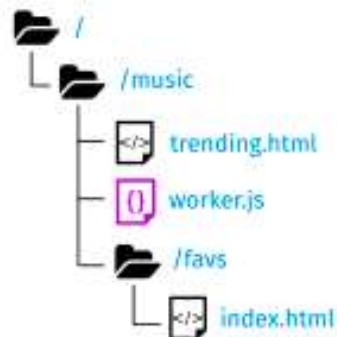
Important!



The service worker will catch requests from the clients **under scope** only.



The service worker must be served over **https**.



The **max scope** for a service worker is the location of the worker.

Service Worker - это файл

```
1 this.addEventListener('install', function (event) {  
2     console.log('Service worker установлен')  
3     event.waitUntil(  
4         // находим Cache-объект с нашим именем  
5         caches.open('MY_CACHE')  
6         .then(function (cache) {  
7             // загружаем в наш cache необходимые файлы  
8             return cache.addAll(['/index.html']);  
9         });  
10    });  
11 });  
12
```

Примерчик

<https://googlechrome.github.io/samples/service-worker/>


PWA

Подробнее тут <https://frontend.tech-mail.ru/slides/s9/#113>

Прогрессивные интернетные приложения

PWA

PWA (Progressive Web Application) - технология в web-разработке, которая визуально и функционально трансформирует сайт в приложение (мобильное приложение в браузере).

 <https://s3.eu-central-1.amazonaws.com/techno>

Выберите операцию ▼

Calc

Reset

Скачиваемые файлы

Поделиться...

Найти на странице

Добавить на главный экран

Полная версия



Настройки

Справка/отзыв



https://s3.eu-central-1.amazonaws.com/technopark-cdn/sample-nativ

2



Добавить на главный экран



Calc

ОТМЕНА

ДОБАВИТЬ

Calc

Reset

13:51

MTS TELE 2 73%



Сбербанк



Музыка



ICQ



Почта Mail..



MAPS.ME



Calc



Telegram



2ГИС



Почта



Радио



Навигатор



Рокетбанк



Аэроэспр.



Мой МТС



ВКонтакте



Slack



Телефон



Сообщения



Chrome



Браузер



Камера

Что нужно сделать, чтобы завелось?

- Service Worker;
- Application shell (оболочка для быстрой загрузки с Service Worker);
- Web App Manifest;
- Push Notifications (включено в Service Worker);
- SSL-сертификат для передачи данных по протоколу HTTPS.

Web App Manifest

```
1 <link rel=manifest href="/manifest.json">  
2
```

Web App Manifest

```
1 {  
2   "name": "Calculator",  
3   "short_name": "Calc",  
4   "lang": "ru-RU",  
5   "start_url": "/index.html",  
6   "display": "fullscreen", // standalone, minimal-ui, browser  
7   "orientation": "landscape", // portrait, any  
8   "background_color": "#0F0848",  
9   "theme_color": "#0F0848",  
10  "related_applications": [ ... ]  
11  "prefer_related_applications": false  
12  "icons": [ ... ]  
13 }  
14
```

Apple Meta Tags

```
1 <link rel=apple-touch-icon href="/imgs/icon-152.png">
2 <meta name=theme-color content=#0F0848>
3
4 <meta name=application-name content="Calc">
5 <meta name=apple-mobile-web-app-title content="Calc">
6
7 <meta name=mobile-web-app-capable content=yes>
8 <meta name=apple-mobile-web-app-capable content=yes>
9 <meta name=apple-mobile-web-app-status-bar-style content=#0F0848>
10
```


Преимущества PWA

PWA совмещает в себе свойства нативного приложения и функции браузера, что имеет свои преимущества:

- PWA поддерживается наиболее популярными ОС: Windows, iOS, Android. При этом загрузить можно на десктоп, смартфон, планшет, терминал в торговом зале;
- обновления добавляются разработчиками удаленно. Пользователи видят изменения и улучшения, но им не требуется скачивать эти обновления самостоятельно;
- PWA индексируется Google и другими поисковыми системами;
- благодаря сценарию Service Worker, который запускается браузером в фоновом режиме, и стратегии кэширования, обеспечивается возможность работы офлайн;
- Frontend отделен от Backend'а. Меньше времени и ресурсов тратится на разработку/переработку дизайна и логики взаимодействия PWA с клиентом;
- PWA можно установить без «Google play» и App Store, а также вопреки запрету устанавливать приложения из неизвестных источников. Лояльно относятся к PWA и антивирусные программы. Одновременно с этим передача данных происходит по протоколу HTTPS, поэтому PWA безопасно;
- с февраля 2019 года PWA можно добавлять в App Store и Google Play, давая пользователю возможность скачать приложение из привычного источника.

Недостатки PWA

Технология PWA не универсальна и имеет ряд недостатков:

- не все устройства и не все операционные системы поддерживают полный набор возможностей PWA;
- невозможно наладить активное участие пользователей iOS (например приложение может хранить локальные данные и файлы размером только до 50 Мбайт, нет доступа к In-App Payments (встроенные платежи) и многим другим сервисам Apple, нет интеграции с Siri), поддержка iOS начинается с версии 11.3;
- работа офлайн ограничена;
- работу PWA может ограничивать неполный доступ к аппаратным компонентам;
- нет достаточной гибкости в отношении «специального» контента для пользователей (например программы лояльности);
- при использовании PWA увеличивается расход заряда батареи мобильного устройства.

Примерчик

<https://www.google.ru/maps/>

Virtual DOM

Подробнее тут <https://frontend.tech-mail.ru/slides/s10/#40>



Что это?

Главная проблема DOM — он никогда не был рассчитан для создания **динамического пользовательского интерфейса**

Virtual DOM — это техника и набор библиотек/алгоритмов, которые позволяют нам улучшить производительность на клиентской стороне, избегая прямой работы с DOM путем создания и работы с абстракцией, имитирующей DOM-дерево

Как работает?

- Вместо того, чтобы взаимодействовать с DOM напрямую, мы работаем с его **легковесной копией**
- Мы можем вносить изменения в копию, исходя из наших потребностей, а после этого применять изменения к реальному DOM
- При этом происходит сравнение DOM-дерева с его виртуальной копией, **определяется разница и запускается перерисовка того, что было изменено**

Картиночка

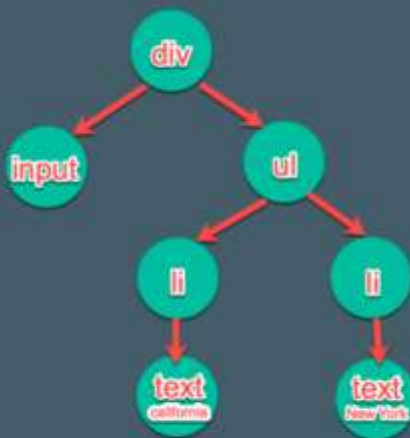
The App



Real DOM

```
<div>
  <input type="text" placeholder="Search">
  <ul>
    <li>California</li>
    <li>New York</li>
  </ul>
</div>
```

DOM Tree



VDOM

```
{
  "nodeName": "div",
  "children": [
    {
      "nodeName": "input",
      "attributes": {
        "type": "text",
        "placeholder": "Search",
        "onChange": ""
      },
      "children": []
    },
    {
      "nodeName": "List",
      "attributes": {
        "items": [
          "California",
          "New York"
        ]
      },
      "children": []
    }
  ]
}
```


Virtual DOM

Такой подход работает быстрее, потому как не включает в себя все тяжеловесные части реального DOM. Но только если мы делаем это правильно.

Есть две проблемы:

- Когда именно делать повторную перерисовку DOM?
- Как это сделать эффективно?

Когда?

Когда данные изменяются и нуждается в обновлении. Есть два варианта узнать, что данные изменились:

- Первый из них — «dirty checking» (грязная проверка) заключается в том, чтобы опрашивать данные через **регулярные промежутки времени** и **рекурсивно проверять все значения** в структуре данных
- Второй вариант — «observable» (наблюдаемый) заключается в наблюдении за изменением состояния. Если ничего не изменилось, мы ничего не делаем. Если изменилось, **мы точно знаем, что нужно обновить**

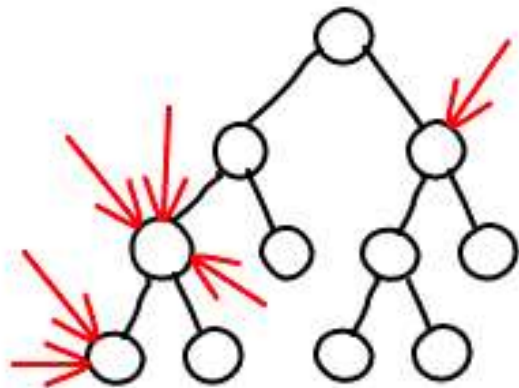
Как?

Что делает этот подход действительно быстрым:

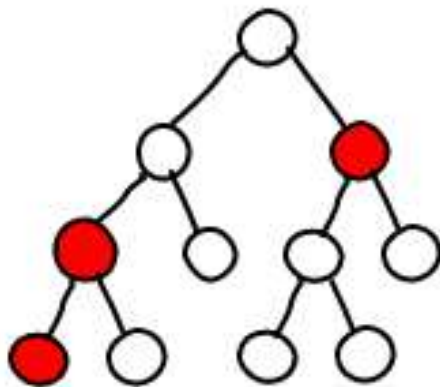
- Эффективные алгоритмы сравнения
- Группировка операций чтения/записи при работе с DOM
- Эффективное обновление под-деревьев

Наглядно

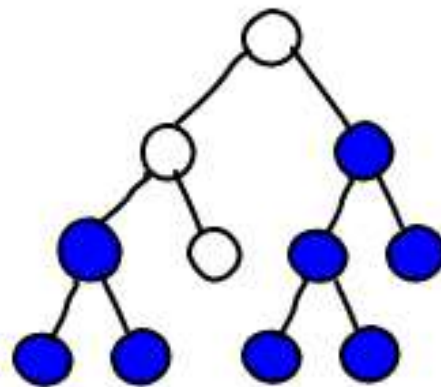
setState



Dirty



Re-rendered



Сложная статейка по VDOM

<https://nickbulljs.medium.com/how-virtual-dom-work-567128ed77e9>

Разбираем как работает Virtual DOM



Что должен уметь VDOM?

- Создавать дерево элементов на странице по шаблону
- Изменять дерево элементов согласно обновленному шаблону

Создание элемента

`vdom.create()`

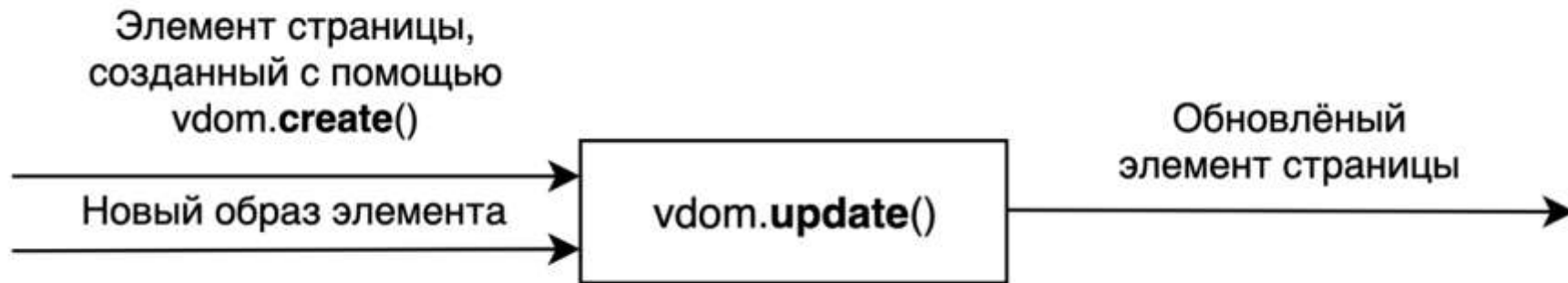


vdom.create() - пример

```
1 const node = vdom.create({  
2   tag: 'DIV',  
3   attrs: {  
4     style: 'background-color: red;',  
5     'tab-index': 0  
6   }  
7 });  
8  
9 document.body.appendChild(node);  
10
```

Обновление элемента

vdom.update()



Что могло измениться?

- Количество и значения атрибутов
- Любой из потомков

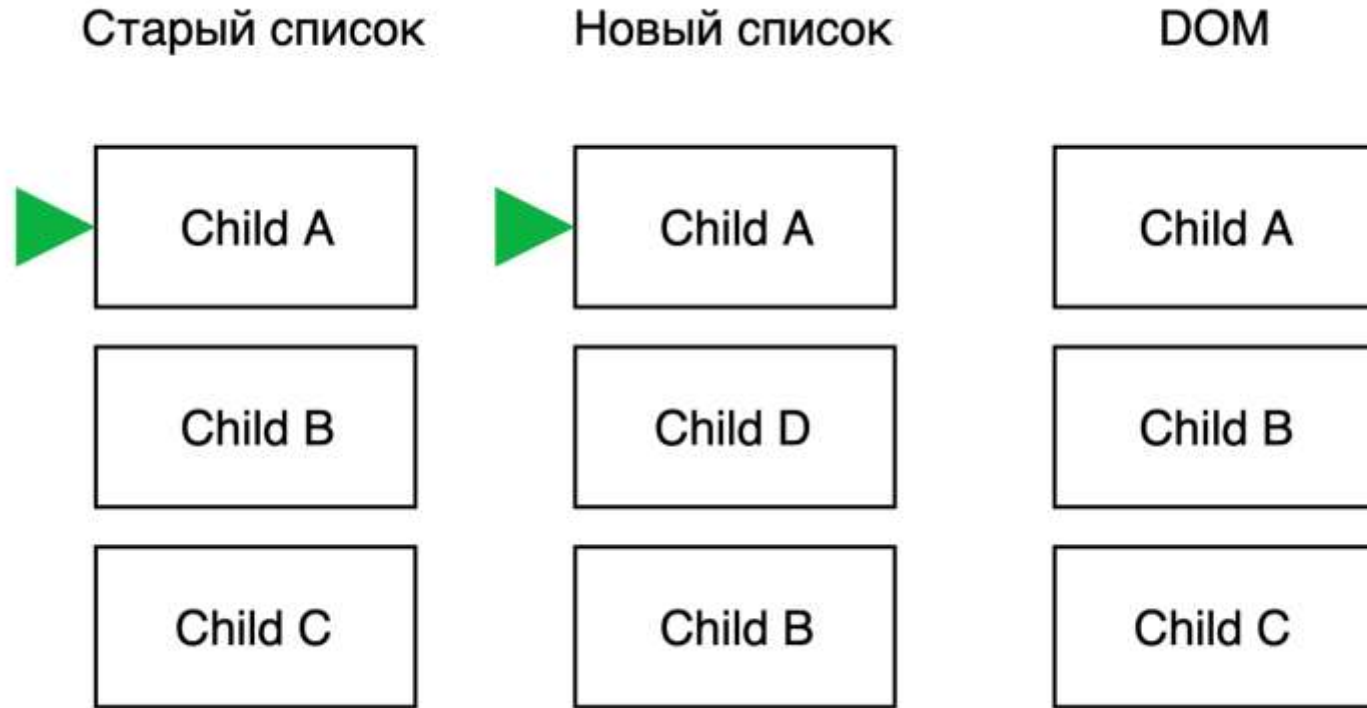
Обновление атрибутов

vdom.update() - пример

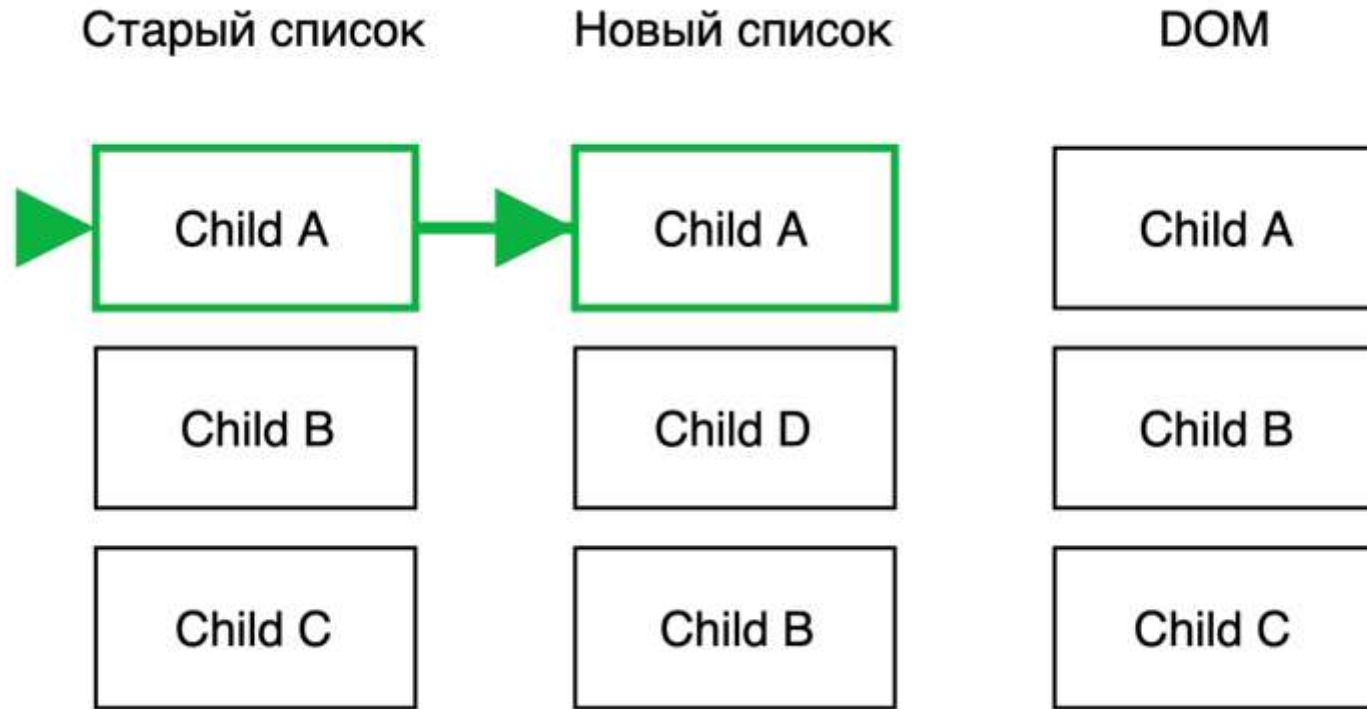
```
1 const node = vdom.create({
2   tag: 'DIV',
3 });
4
5 document.body.appendChild(node);
6
7 vdom.update(node, {
8   tag: 'DIV',
9   attrs: {
10     style: 'background-color: black;',
11   }
12 });
13
```

Обновление дочерних элементов

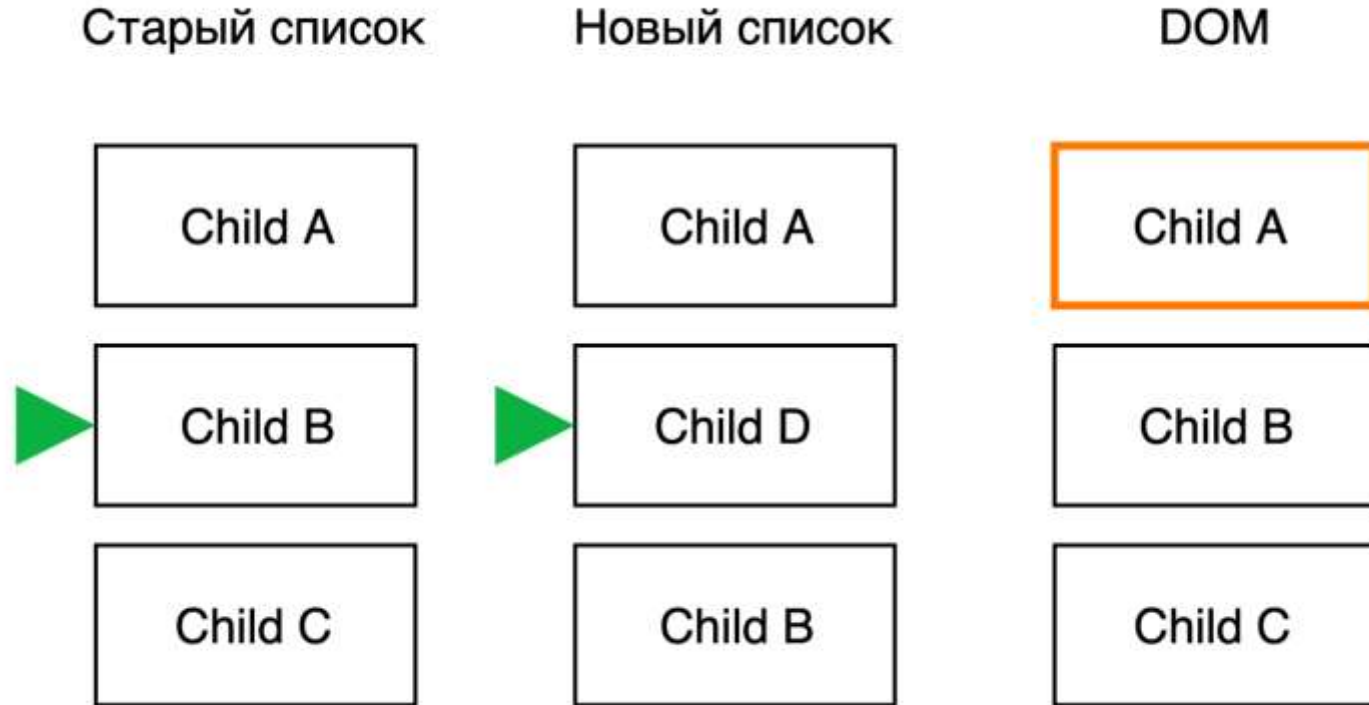
Обновление дочерних элементов



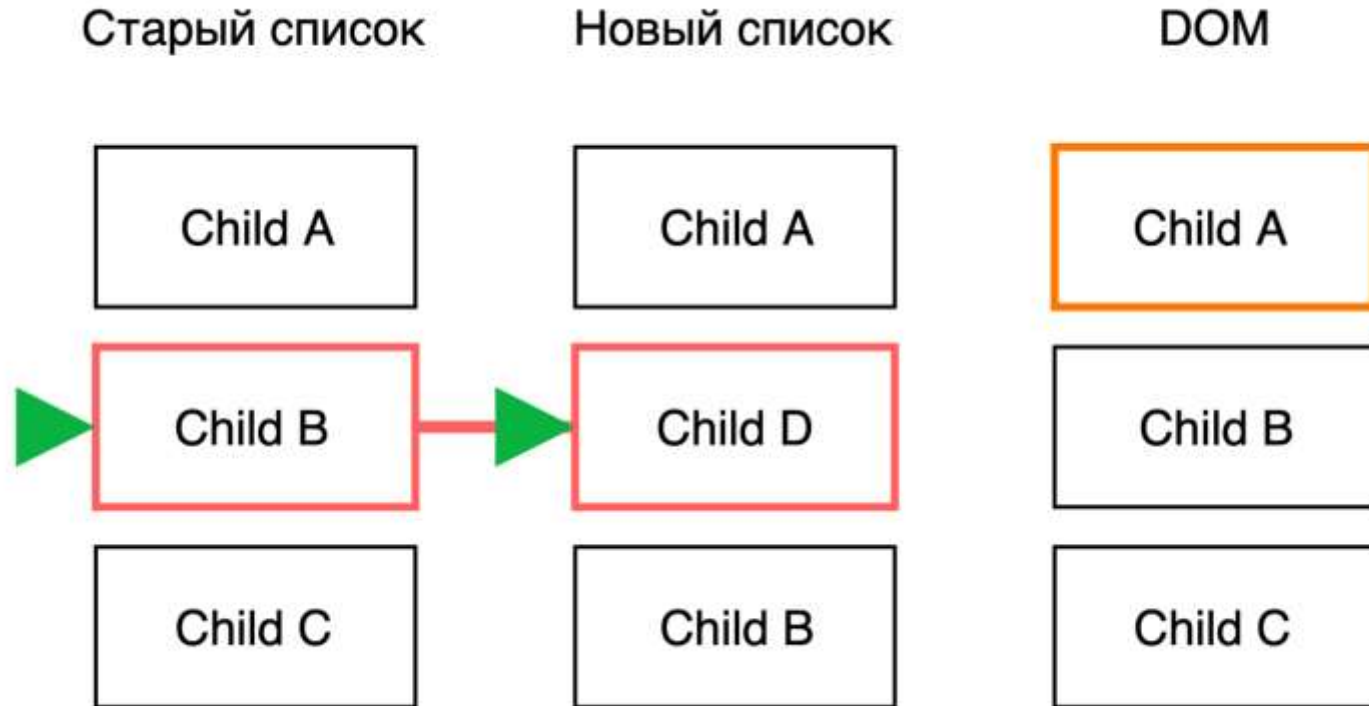
Обновление дочерних элементов



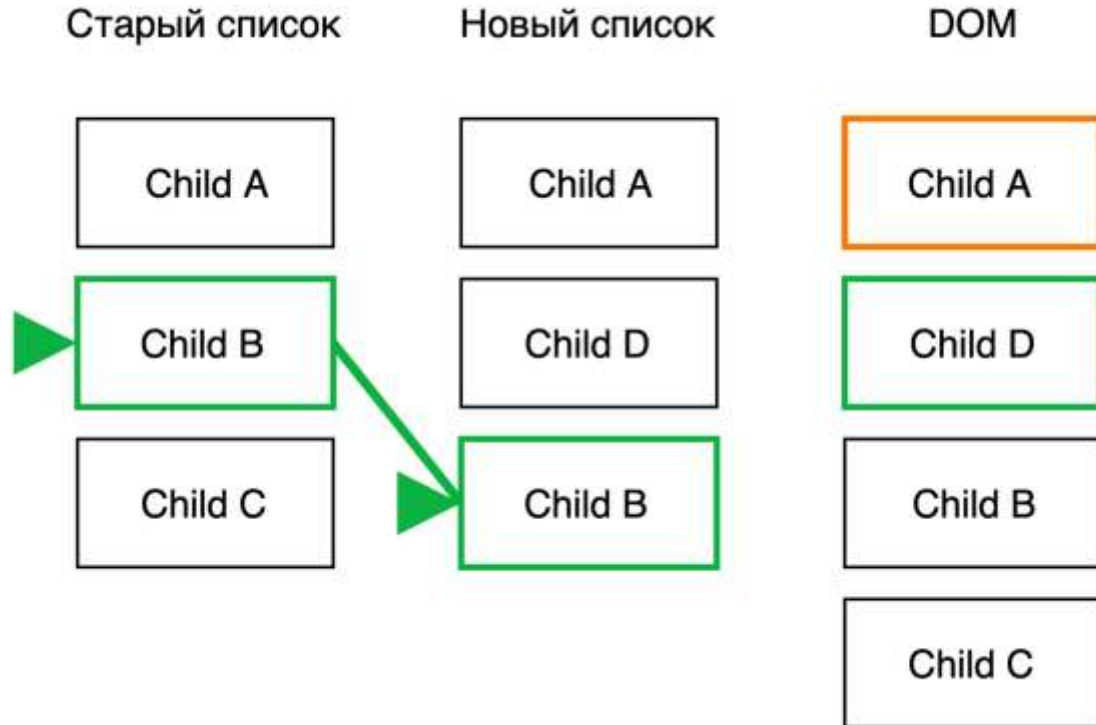
Обновление дочерних элементов



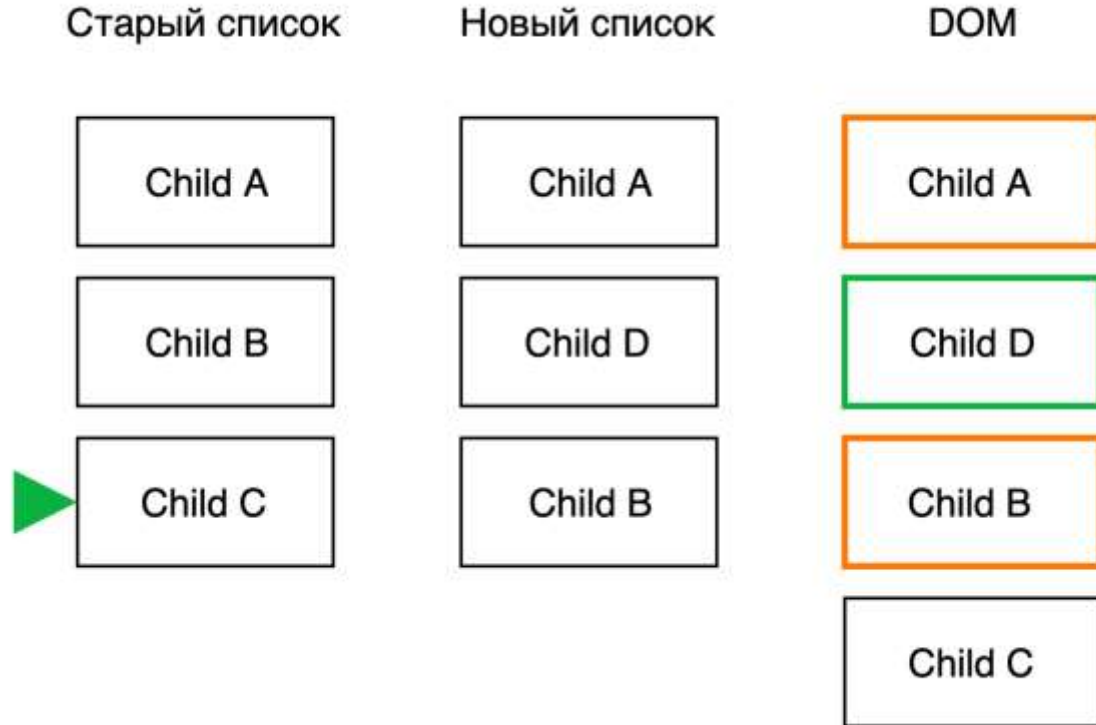
Обновление дочерних элементов



Обновление дочерних элементов



Обновление дочерних элементов



Обновление дочерних элементов

Старый список

Child A

Child B

~~Child C~~

Новый список

Child A

Child D

Child B

DOM

Child A

Child D

Child B

~~Child C~~

В итоге

- 1 создание
- 1 удаление
- 2 обновления

vdom.update() - пример


```
1 const node = vdom.create({
2   tag: 'DIV',
3 });
4
5 document.body.appendChild(node);
6
7 vdom.update(node, {
8   tag: 'DIV',
9   children: [
10    {
11      tag: 'div',
12      attrs: {
13        style: 'backgroun-color: black;',
14      },
15    }
16  ]
17 });
18
```



Все работает

Но есть одна проблемка

Я ♥ HTML

JSX





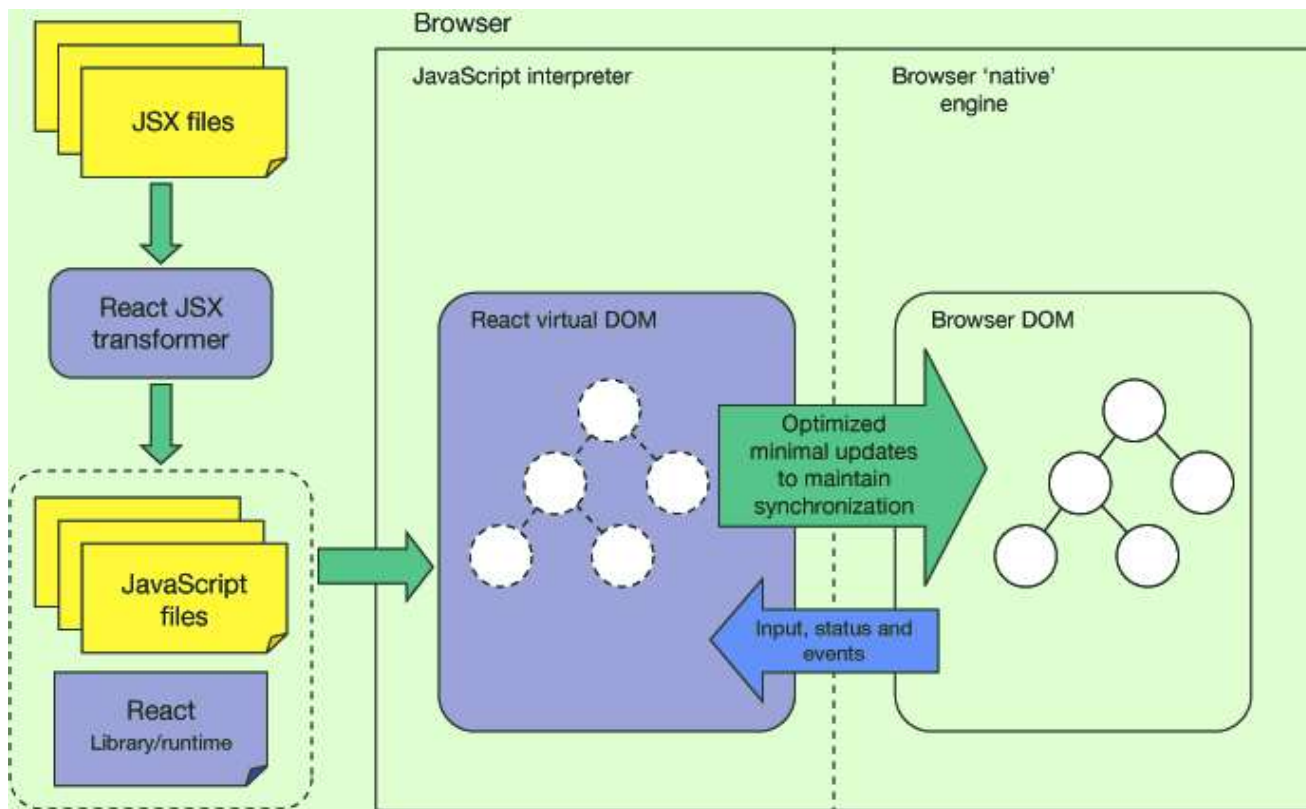
☒ Evaluate Presets: es2015, react, stage-2 ▾ ☐ Line Wrap ☐ Minify (Babili) Babel 6.25.0

```
1 var App = (  
2   <div>  
3     <div></div>  
4     <footer>2017 All rights reserved.</footer>  
5   </div>  
6 );  
7 ReactDOM.render(App, document.getElementById('renderTarget'));
```

```
1 "use strict";  
2  
3 var App = React.createElement(  
4   "div",  
5   null,  
6   React.createElement(  
7     "div",  
8     null,  
9     React.createElement("img", { src: "./img/logo.png" })  
10  ),  
11  React.createElement(  
12    "footer",  
13    null,  
14    "2017 All rights reserved."  
15  )  
16 );  
17 ReactDOM.render(App, document.getElementById('renderTarget'));
```

React is not defined

Как выглядит



React/Vue/Angular/Svelte

Подробнее тут ???

Что такое фреймворки и зачем они
нам нужны?

Об этом вы узнаете в следующем
семестре 😊

Из основного

- Имеют встроенный Virtual DOM или аналоги
- Позволяют писать независимые компоненты
- Умеют работать с событиями
- Имеют решения по работе с состояниями
- Имеют решения по работе с данными (стор)
- Позволяют быстро писать классные приложения



Logo

WordBox

SearchButton

Google Search

I'm Feeling Lucky

LuckyButton

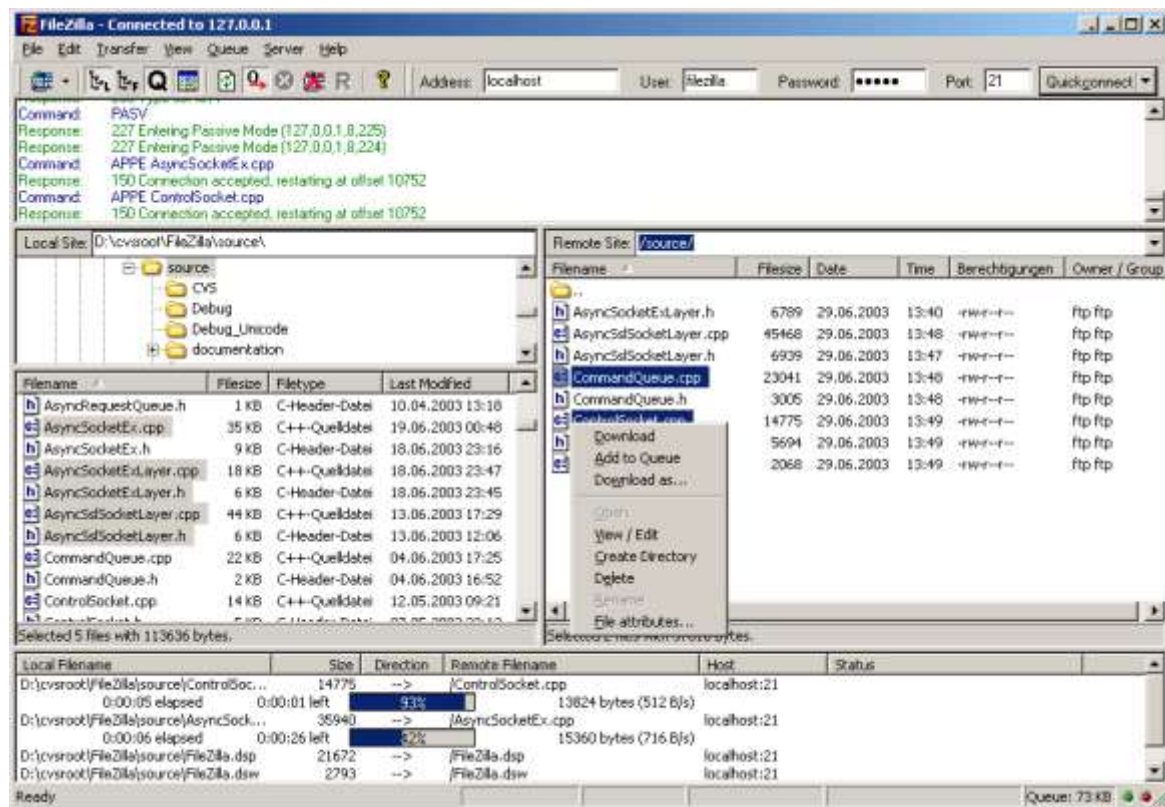
Google.ca offered in: Français

LanguageNote

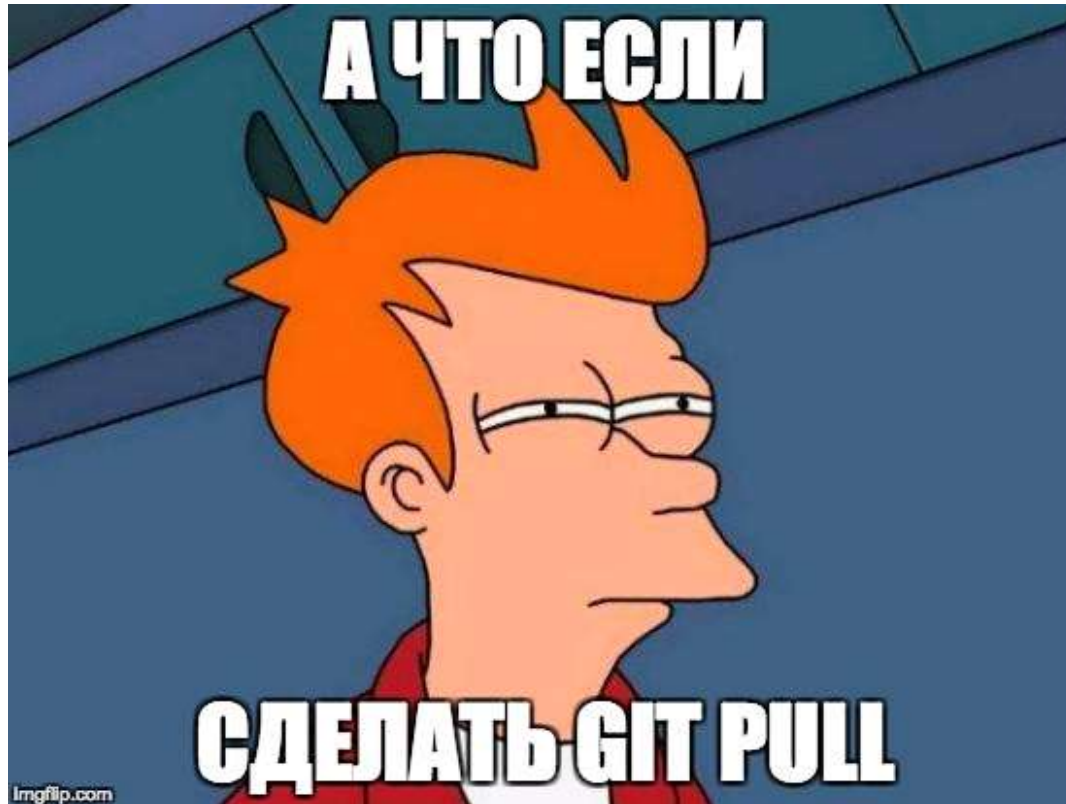
Deploy

Подробнее тут <https://frontend.tech-mail.ru/slides/s11/#32>

Давным давно



Git way



Написали код

Запустили код

Выполнили сборку

Запустили тесты

Получили
отчеты

Отдали на
выкатку



а что тут делает водопад?

Время

Написали код

Запустили код

Continuous Integration

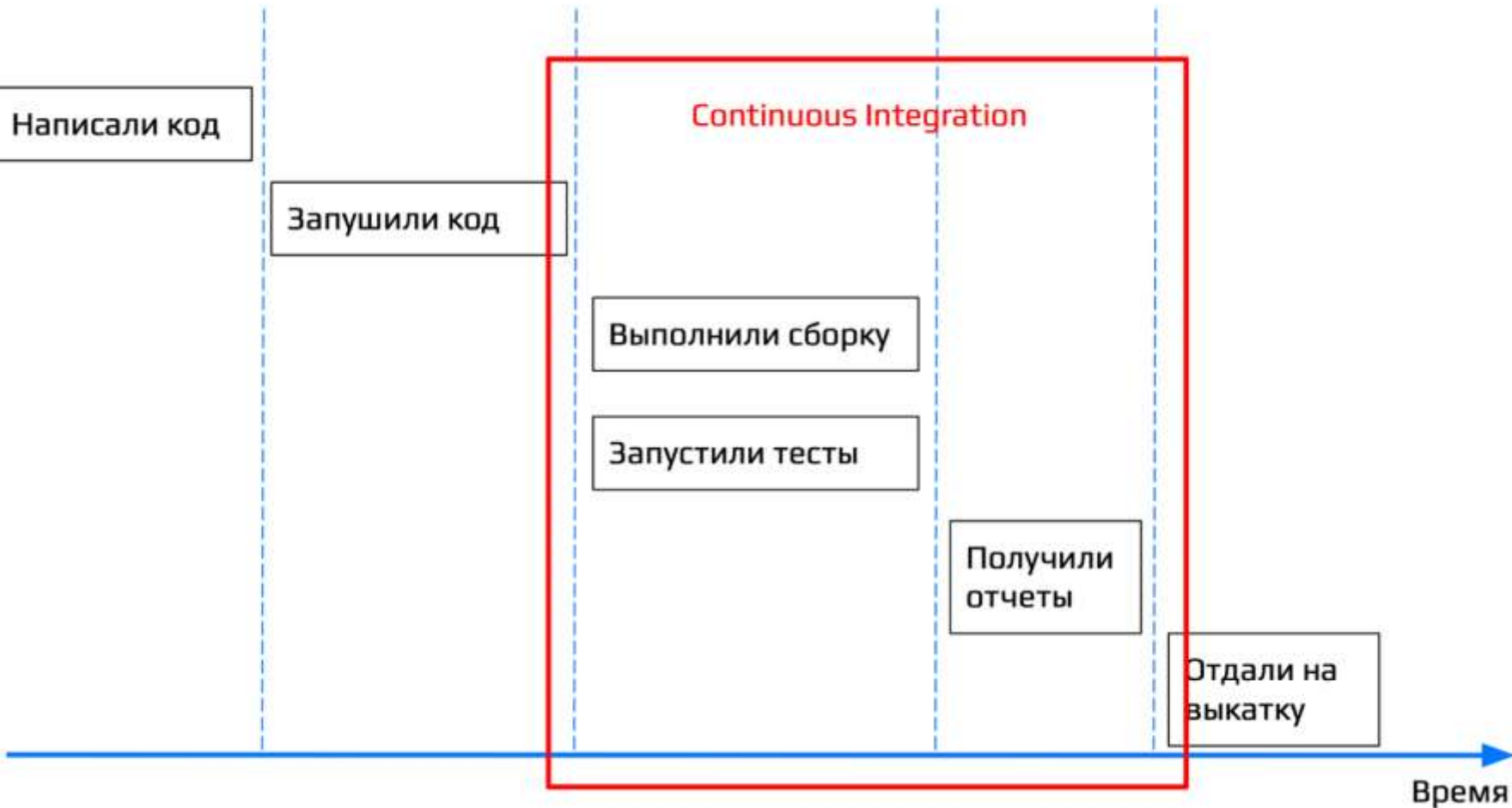
Выполнили сборку

Запустили тесты

Получили
отчеты

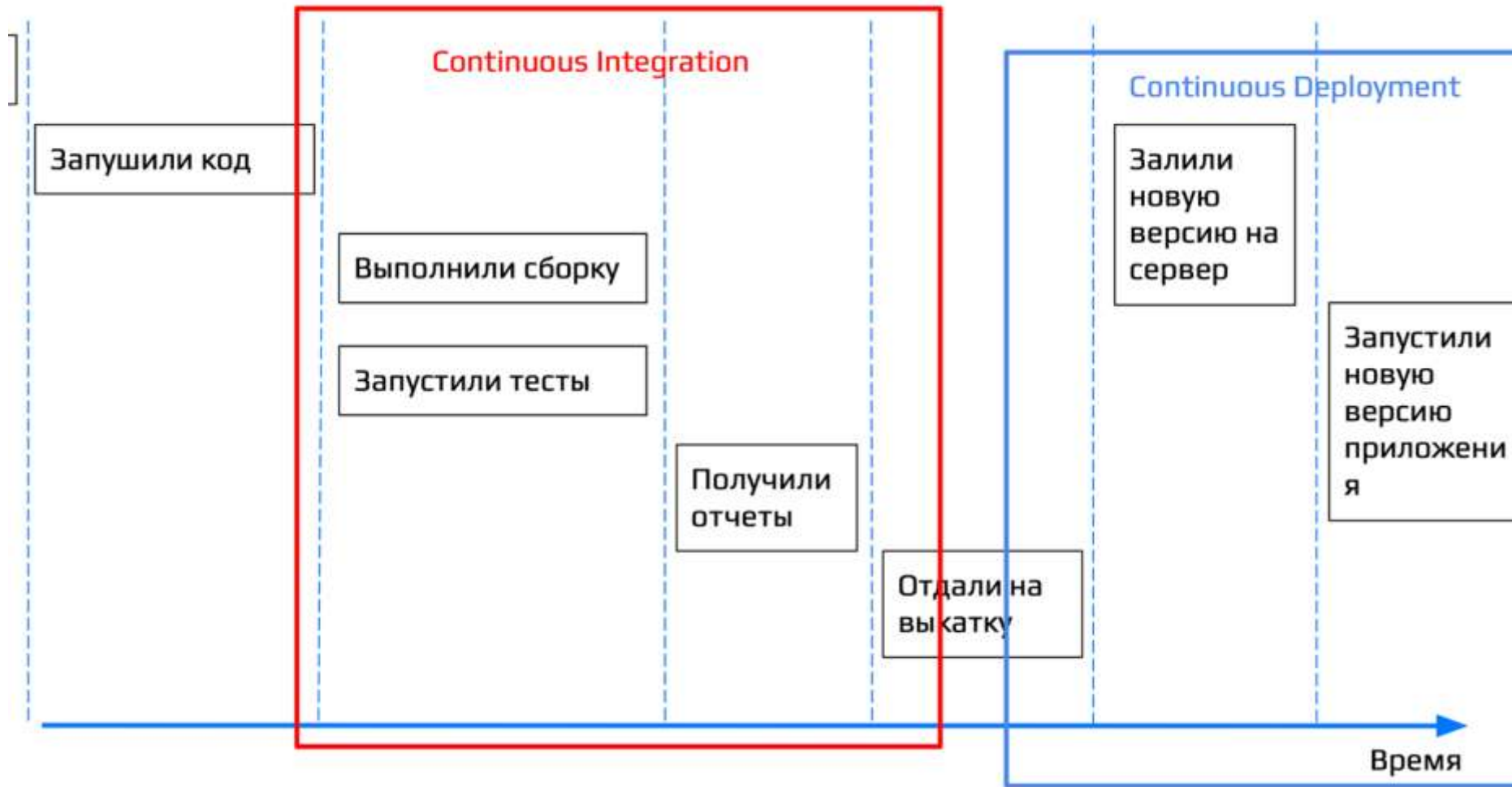
Отдали на
выкатку

Время



Continuous Integration

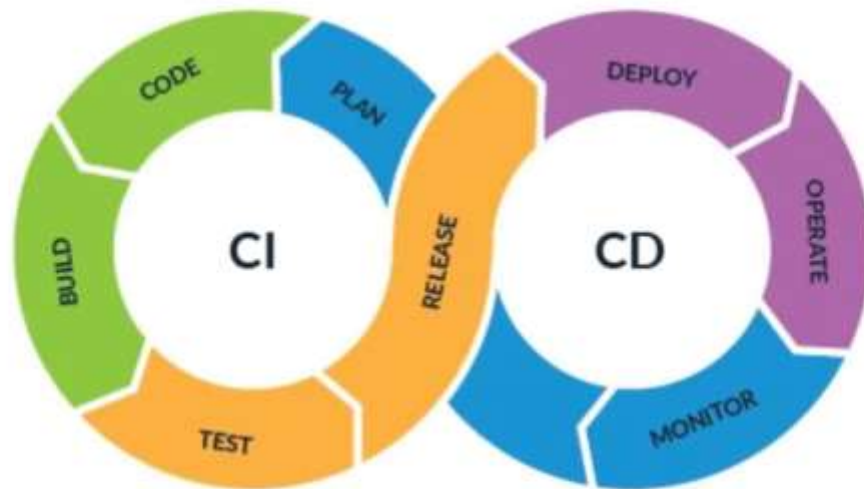
Это практика разработки программного обеспечения, которая заключается в выполнении частых автоматизированных сборок проекта для скорейшего выявления и решения интеграционных проблем.



Continuous Deployment

Это подход к разработке программного обеспечения, при котором функциональные возможности программного обеспечения часто предоставляются посредством автоматизированного развертывания.

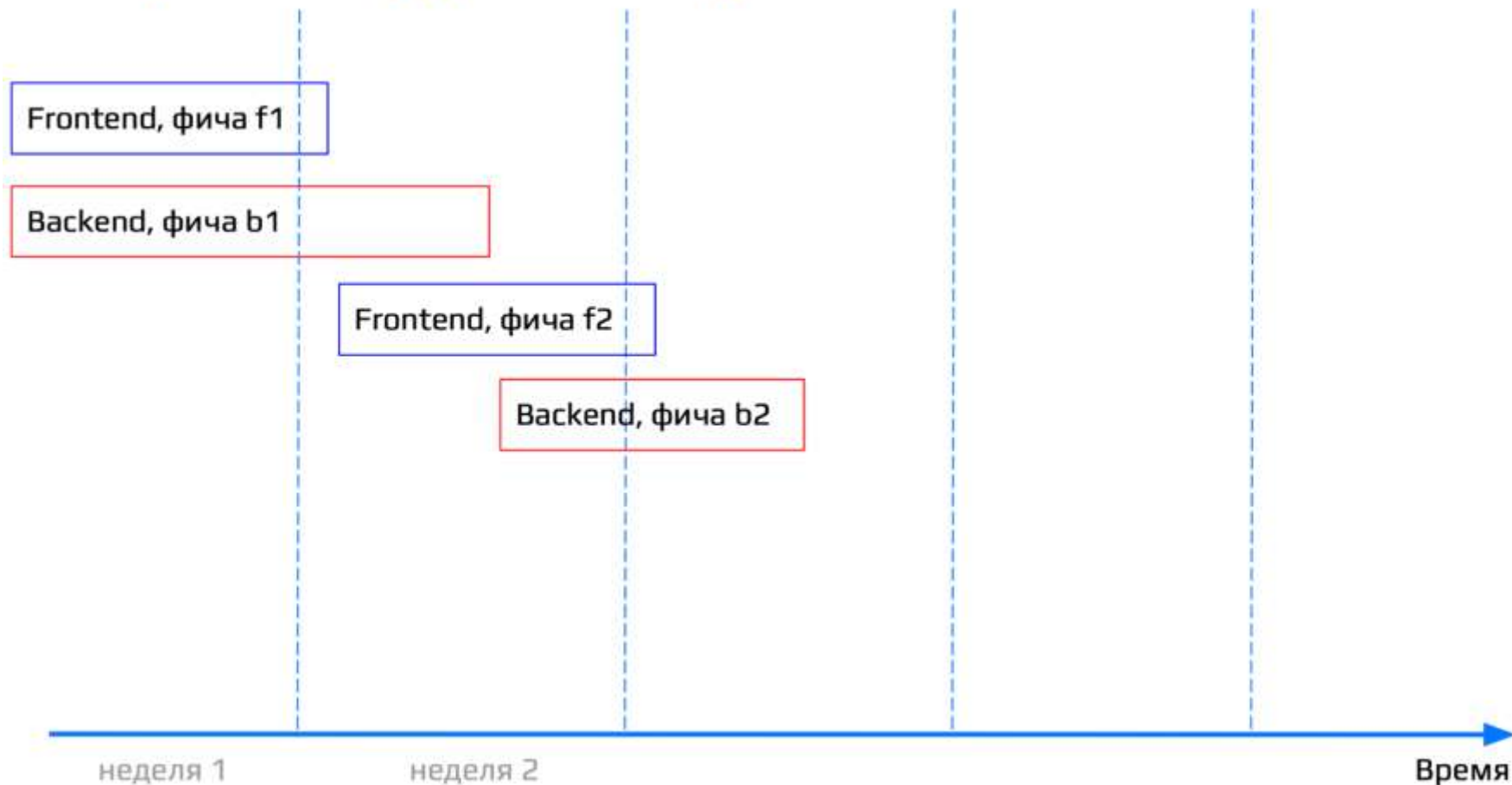
CI и CD – братья близнецы



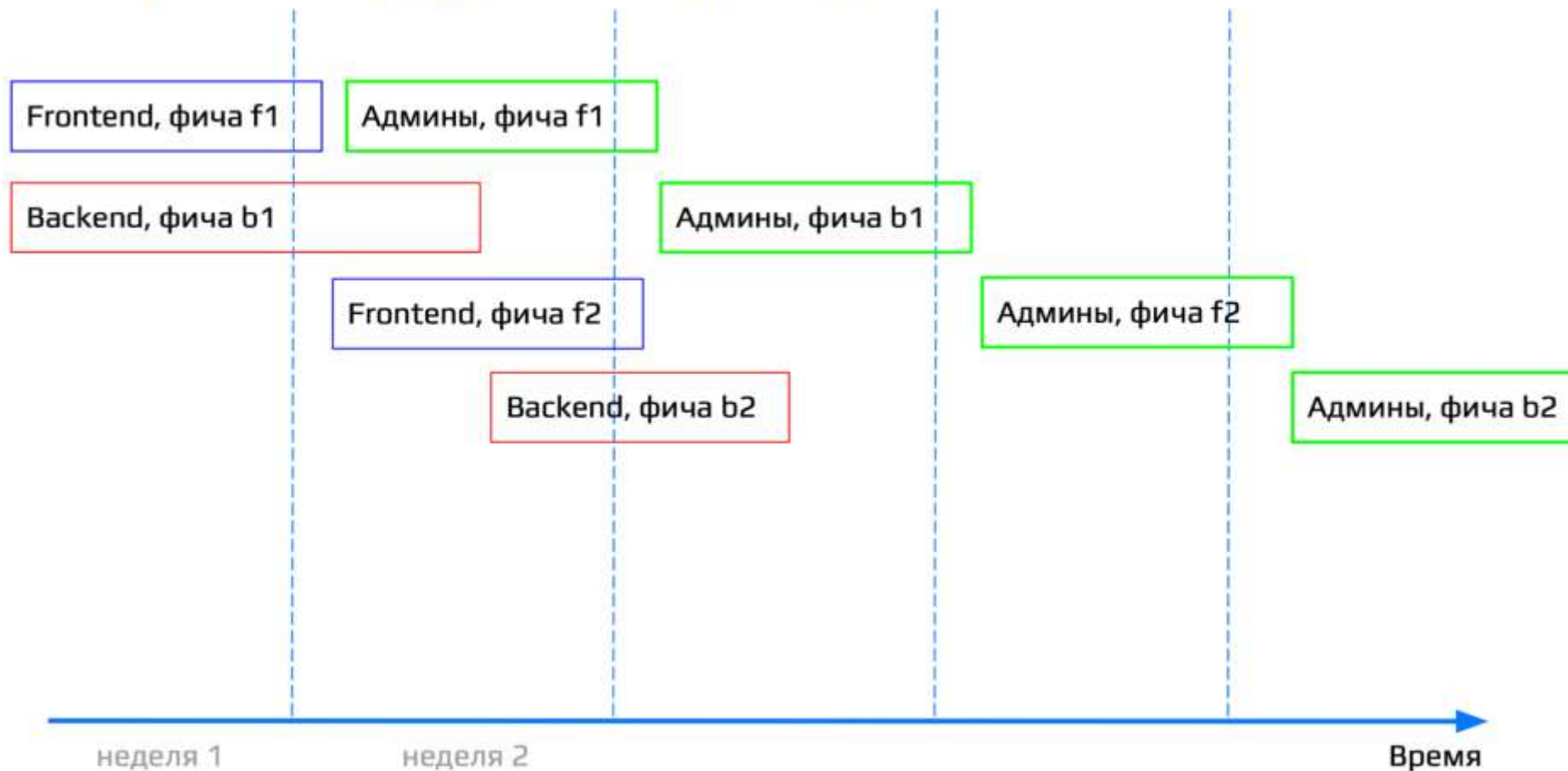
*CI – Continuous Integration

*CD – Continuous Deployment

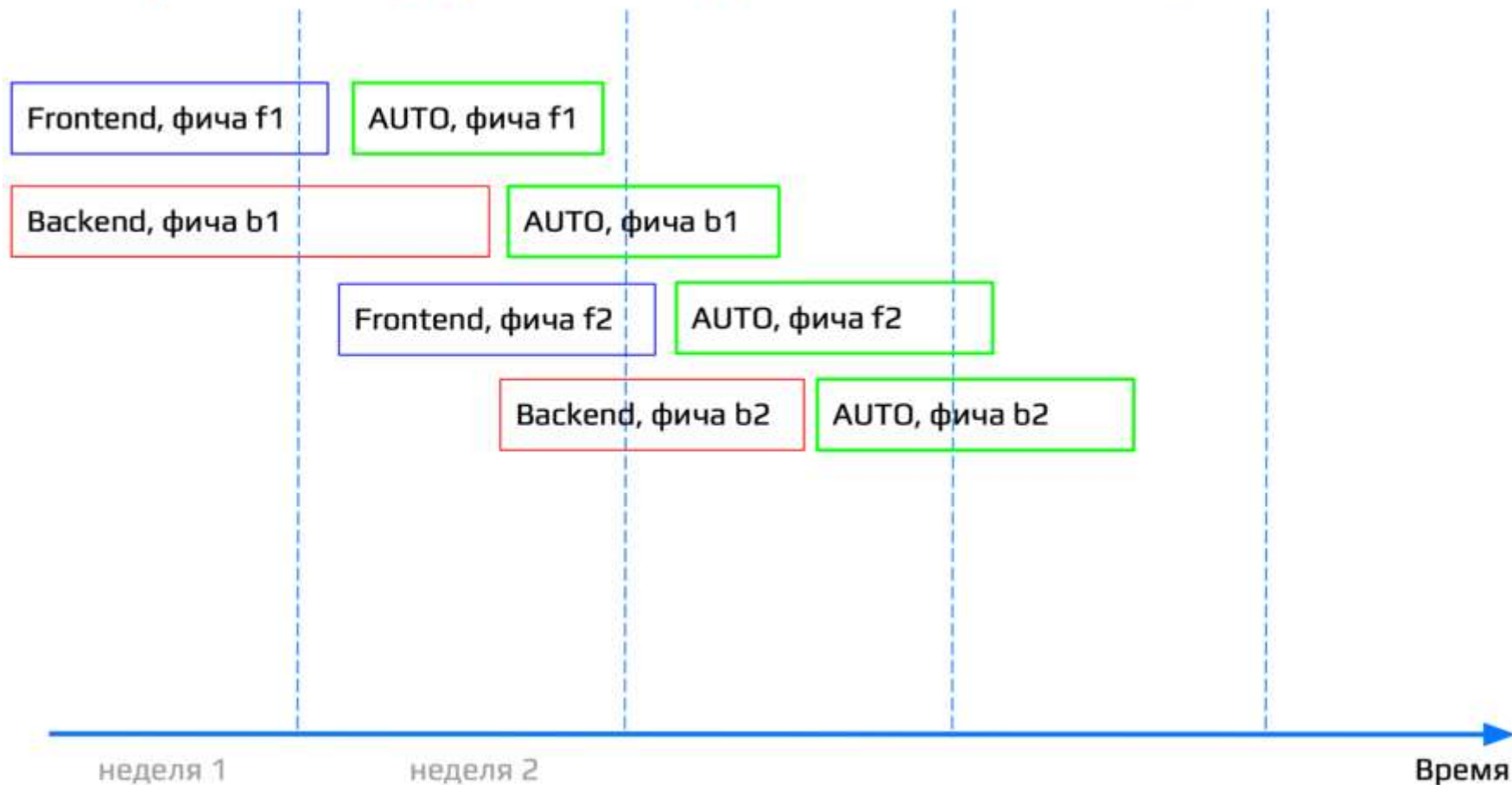
Разработка двух команд



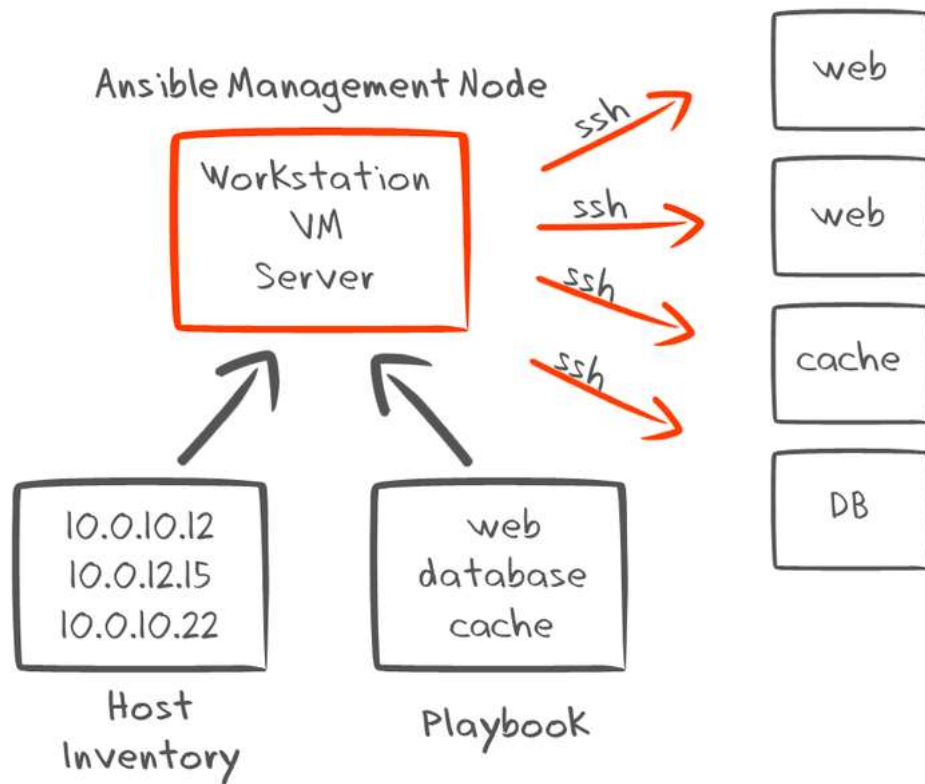
Разработка двух команд + админы



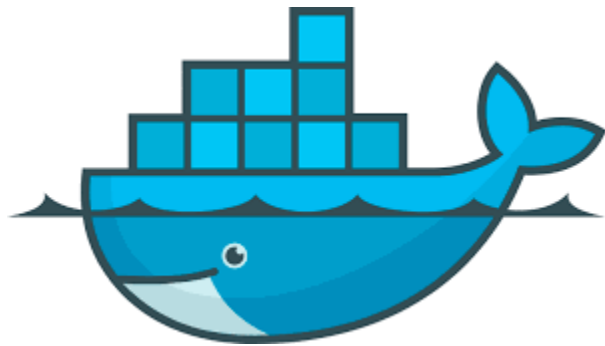
Разработка двух команд + автоматизация



Управляем конфигурацией



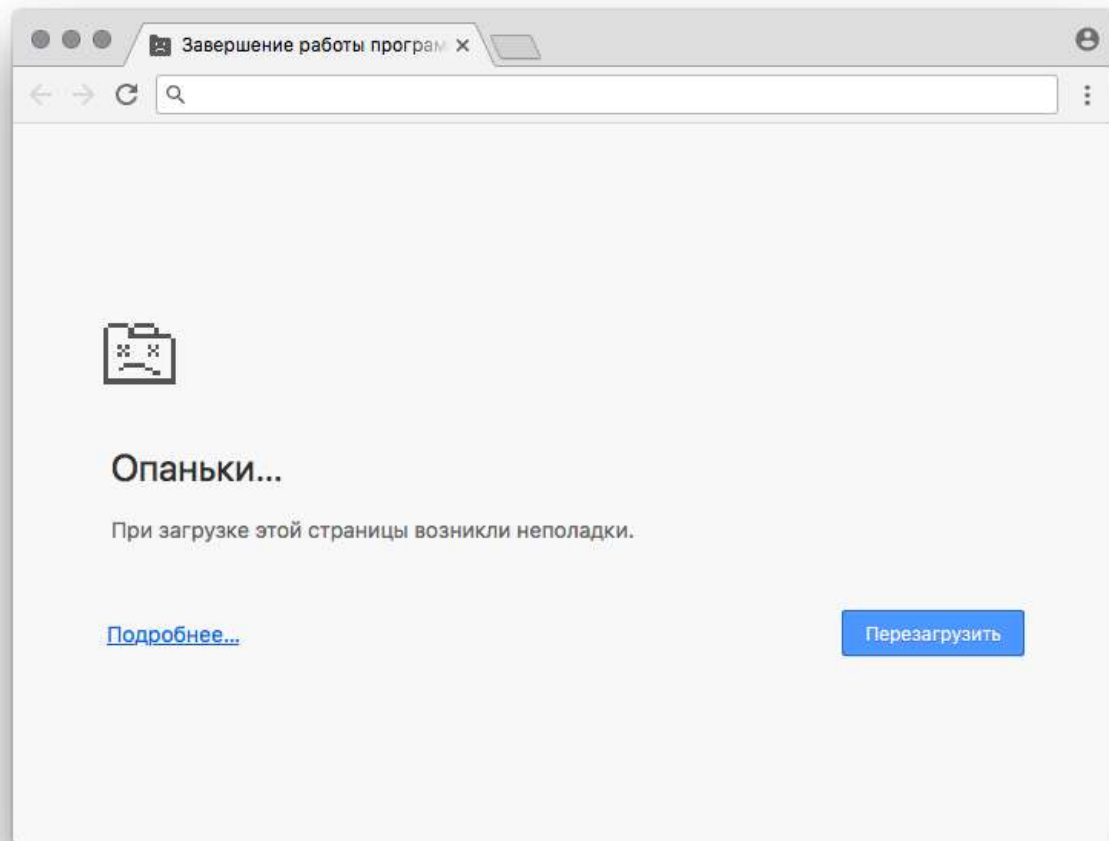
Docker way



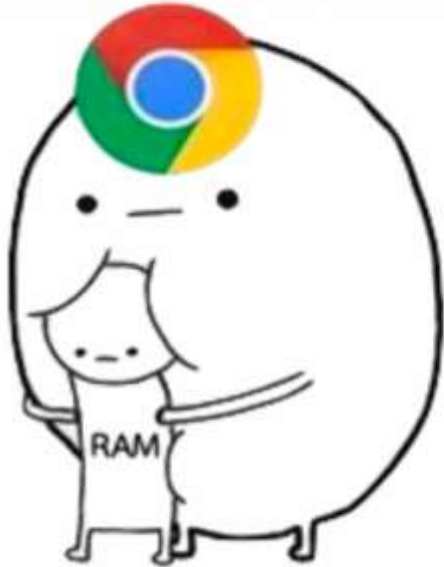
- 🚚 Каждый компонент системы в отдельном контейнере
- ⚙️ Контейнеры содержат в себе всю конфигурацию
- 🌐 Образы хранятся в registry
- 📅 1 Образы версионятся

Производительность

Подробнее тут <https://frontend.tech-mail.ru/slides/s11>



RAM

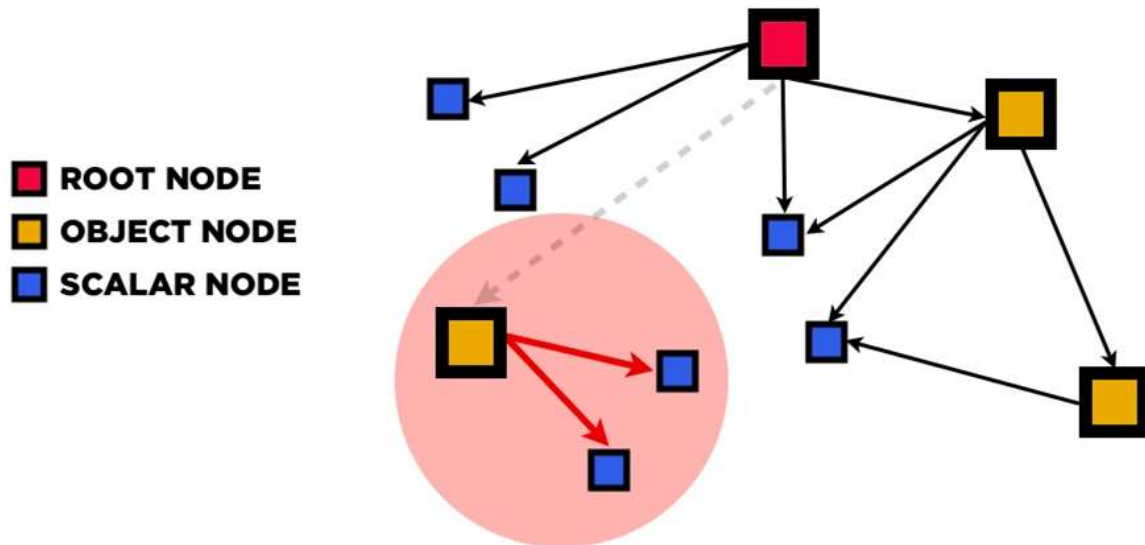


Управление памятью в JavaScript

JavaScript — это язык с **автоматическим управлением памятью**. В языке JavaScript память выделяется автоматически при создании объектов и освобождается тоже автоматически в процессе **сборки мусора**

JAVASCRIPT MEMORY MANAGEMENT

MEMORY VALUE GRAPH



Flagged for elimination in next GC collection cycle

Как может закончиться память?

- **При создании очень большого количества объектов** — например, при слишком агрессивном кешировании чего-нибудь, или вследствие плохого написания кода приложения
- **При создании и работе с большими объектами** — например, работа с медиафайлами (изображениями, видео)
- Вследствие **утечек памяти**

Утечка памяти

Утечка памяти — ситуация, когда память занимается объектами, которые больше не нужны приложению, но которые **не могут быть освобождены автоматически** из-за несовершенства алгоритмов сборки мусора

Чаще всего причиной утечек памяти являются т.н. **нежелательные ссылки** — ссылки, достижимые из корня, но ссылающиеся на фрагменты памяти, которые точно никогда больше не понадобятся

Ссылки на удаленные элементы

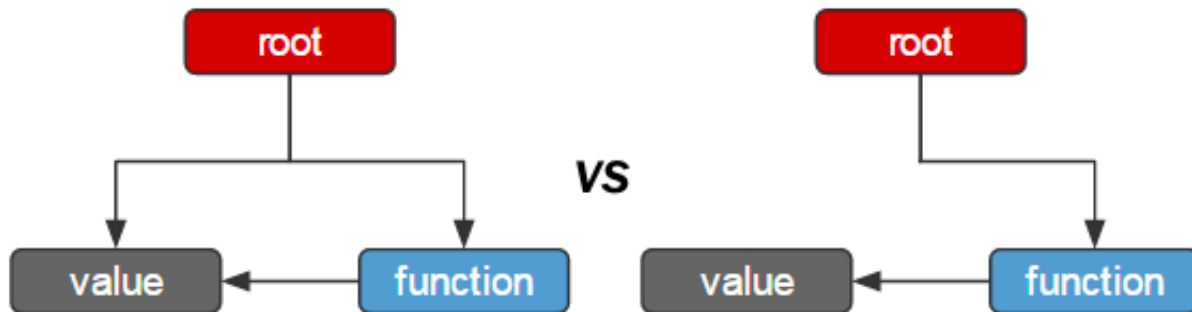
```
1 // закешировали ссылки
2 window.elements = {
3     button: document.querySelector('button#inc'),
4     cell: document.querySelector('.js-super-table td.js-super-cell'),
5 };
6
7 // do stuff
8
9 // элементы будут удалены из DOM, но останутся в памяти
10 document.removeChild(window.elements.button);
11 document.removeChild(document.querySelector('.js-super-table'));
12
```

Замыкания, колбеки

```
1 const trigger = document.getElementById('trigger');
2 const elementToRemove = document.getElementById('remove');
3
4 trigger.addEventListener('click', function () {
5     document.removeChild(elementToRemove);
6 });
7
```

Замыкания, колбеки - чиним

```
1 const trigger = document.getElementById('trigger');  
2 trigger.addEventListener('click', function () {  
3   const elementToRemove = document.getElementById('remove');  
4   document.removeChild(elementToRemove);  
5 });  
6
```



Инструменты отладки

- **window.performance.memory** - объект **MemoryInfo**
- Вкладка **Performance**
- Вкладка **Memory Profiler**
- Вкладка **Performance monitor**

Как работают браузеры

- Загрузка ресурсов страницы — тело документа, файлы скриптов и стилей
- Парсинг HTML, построение DOM-дерева документа
- Парсинг CSS, построение CSSOM
- Выполнение JavaScript-кода
- **(re)calculating styles** — расчет всех стилей, применяемых к элементам
- **layout (иначе, reflow)** элементов страницы — расчет параметров элементов документа (ширина и высота элемента, его положение на странице)
- **(re)paint elements** — рендер изображения элементов документа
- **compositing of layers** — сведение всех слоев в единое изображение в правильном порядке
- **go to item 4**

Оптимизация JS

Сегодня большинство устройств обновляют свои экраны 60 раз в секунду. Каждый из этих кадров может длиться чуть более 16 мс ($1 \text{ секунда} / 60 = 16,66 \text{ мс}$). В реальности же браузеру нужно выполнить и еще кое-какие действия, потому непрерывная работа JS должна занимать не более 10 мс

- Использование функции **requestAnimationFrame**
- Вынесение сложных вычисления, обработки больших объёмом данных в WebWorker'ы
- Асинхронная обработка данных по частям
- Оптимизация запуска **тяжёлых** функций с помощью **Throttling** и **Debouncing**
- Профилирование JS кода с помощью инструментов разработчика

Throttling и Debouncing

Throttling — декорирование функции при котором она будет выполняться не чаще одного раза в указанный период, даже если она будет вызвана много раз в течение этого периода. Т.е. все промежуточные вызовы будут игнорироваться

Debouncing — декоратор позволяет превратить несколько вызовов функции в течение определенного времени в один вызов, причем задержка начинает заново отсчитываться с каждой новой попыткой вызова

Оптимизация вычисления стилей

- Сокращение количества элементов в документе
- Сокращение сложности макетов
- Снижение сложности селекторов, уход от использования каскада стилей в сторону **методологий, основанных на классах**
- Профилирование и отладка с помощью инструментов разработчика

Приблизительно 50 % времени, которое тратится на вычисление стиля элемента, уходит на сопоставление селекторов, а вторую половину времени занимает построение `RenderStyle` (представления стиля) на основе сопоставленных правил

Оптимизация перерасчета макета

- Избегание перерасчета макета, потому что зачастую, он выполняется для всего документа целиком
- Уменьшение количества элементов в документе
- Использование новых и более производительных способов вёрстки макета, например, использование **flexbox-layout'ов** вместо моделей макетов на основе **float**
- Избегание **принудительных синхронных reflow всего документа**
- Профилирование и отладка с помощью инструментов разработчика

Оптимизация перерисовок элементов

- Изменение любого свойства, кроме transform и opacity, вызывает перерисовку
- Сокращение области прорисовки путем размещения элементов на отдельных слоях и оптимизации анимации
- Использование более простых css-свойств для стилизации элементов, избегание использования **затратных css-свойств**
- Профилирование и отладка с помощью инструментов разработчика

Оптимизация компоновки слоев

- Управление количеством слоев
- Вынесение анимируемых элементов на новые слои с помощью свойств **will-change, transform...**
- Программирование анимаций с использованием правильных CSS-свойств: **transform, opacity**
- Профилирование и отладка с помощью инструментов разработчика

Резюме FE разработчика в 2023

Подробнее тут <https://frontend.tech-mail.ru/slides/s12/>

Теория

Теория

- Знать про протоколы WEB'a: HTTP/1 (HTTP/1.1), HTTP/2, HTTP/3, HTTPS, WebSocket
- Знание JavaScript — причём, полезно знать как es6+, так и es5. TypeScript
- Знание возможностей браузеров (HTML5, CSS), уметь в *кроссбраузерность*, понимать, как браузеры работают *внутри*
- Вопросы, связанные с безопасностью web-приложений (что такое XSS, как работают cookie, что такое CORS...)
- Понимать вопросы, связанные с инфраструктурой, деплоем статики
- Иметь представление о UX
- Иметь представление о server-side
- Всё, что связано с программированием в целом

Инструменты

Инструменты

- Инструменты разработки: Node.js
- Инструменты сборки: Webpack, grunt, gulp
- Инструменты работы с CSS: PostCSS
- Модульные системы: ES6-модули, AMD, UMD, commonjs
- DevTools браузеров

Фреймворки

Фреймворки

- React*
- Angular
- VueJS
- Svelte

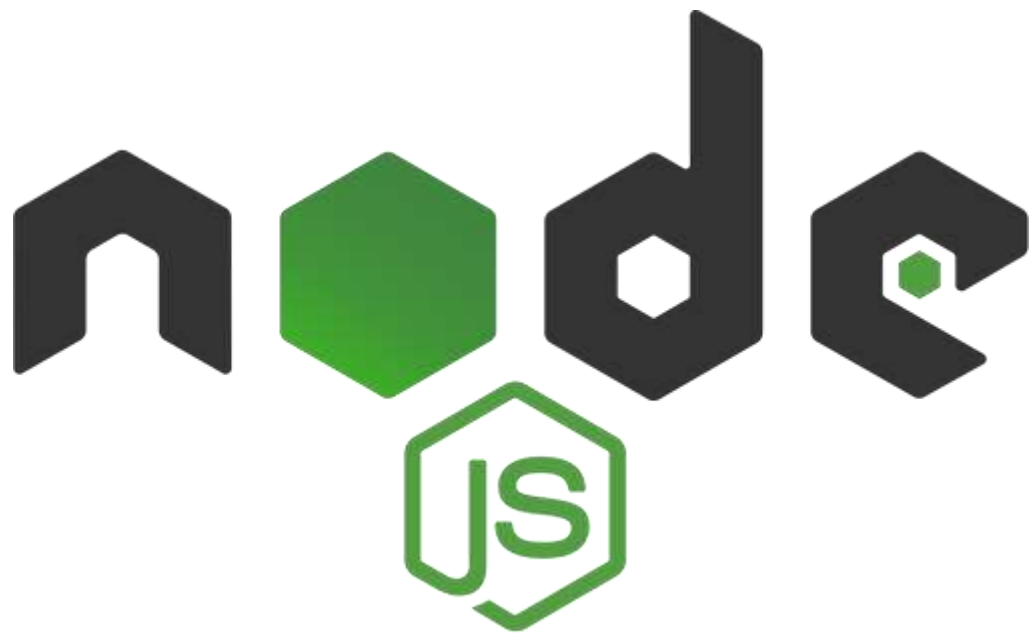
Как развиваться?

Как развиваться?

- Много практики — пробуйте новое, старайтесь развиваться всесторонне
- Читайте статьи — например на Habrahabr, Medium, Learn Javascript

Frontend - это не только браузер

Server-side



Node.JS

- **Локальные и консольные утилиты**
 - Сборщики, трансляторы, компиляторы
 - Скриптинг, CLI, генерация документации, тесты
- **Серверы**
 - API, dev-сервера, бекенды для SPA
 - Highload & real-time системы: игры, чаты
 - Заплаты в узкие места уже готовых систем
- **Клиентские приложения**
 - Кравлеры, сборщики данных, анализаторы логов, статистики
 - Desktop приложения (nw.js, Electron)
- **Железо**
 - Работа на микроконтроллерах (Tessel, Espruino...)



MongoDB

```
{  
  _id: <ObjectId>,  
  username: "123xyz",  
  contact: {  
    phone: "123-456-7890",  
    email: "xyz@example.com"  
  },  
  access: {  
    level: 5,  
    group: "dev"  
  }  
}
```



Embedded sub-
document



Embedded sub-
document

Desktop

Chromium

Chromium — веб-браузер с открытым исходным кодом, разрабатываемый сообществом The Chromium Authors, компанией Google и некоторыми другими компаниями



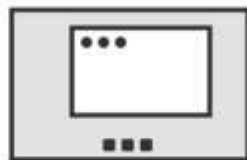
Chromium
for making
web pages

+



Node.js
for filesystems
and networks

+



Native APIs
for three
systems

=



ELECTRON

Electron

Electron — фреймворк, разработанный **GitHub**. Позволяет разрабатывать нативные графические приложения для операционных систем с помощью веб-технологий, комбинируя возможности **Node.js** для работы с **back-end** и браузера **Chromium**.

Популярность

- Atom
- Slack
- Visual Studio Code
- GitHub Desktop
- Hyper
- Discord
- Spotify

Mobile

1st
Gen

Platform Proprietary



Objective-C / Swift



Android Java



Windows .NET

Native UX
High performance
Multi-platform
Unified codebase
Hardware & platform access

2nd
Gen

Hybrid HTML & JavaScript Frameworks



PhoneGap



CORDOVA



ionic

Native UX
High performance
Multi-platform
Unified codebase
*Hardware & platform access

3rd
Gen

Cross- Platform Native



Xamarin

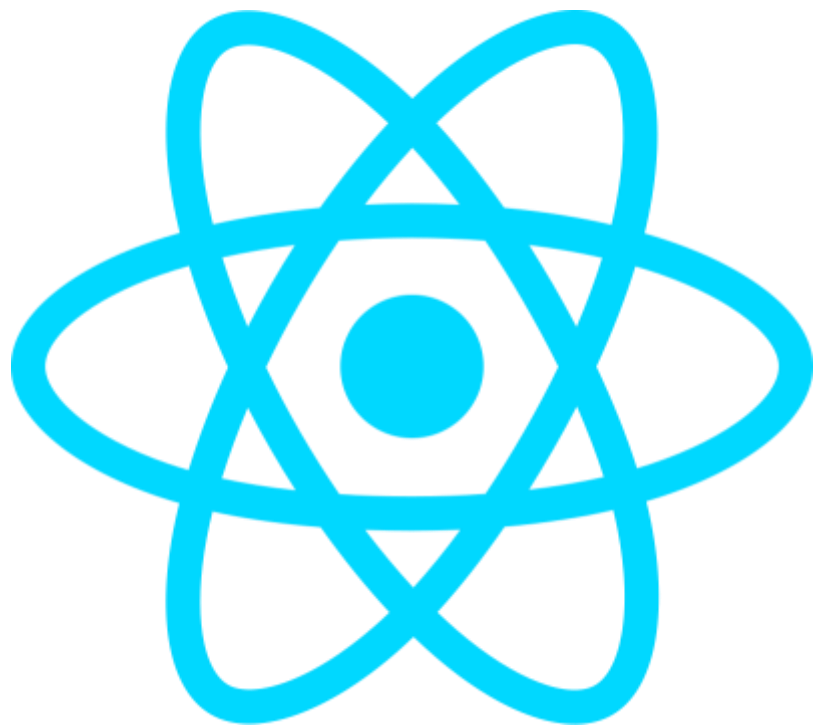


React Native



NativeScript

Native UX
High performance
Multi-platform
*Unified codebase
Hardware & platform access



React Native

React Native — фреймворк для построения нативных мобильных приложений с использованием React

Пример

```
1 import React, { Component } from 'react';
2 import { Text, View, ProgressBar } from 'react-native';
3 import { TheNativeComponent } from './your-native-code';
4 class SomeComponent extends Component {
5     render() {
6         return (<View>
7             <Text>Loading</Text>
8             <ProgressBar />
9             <TheNativeComponent />
10         </View>)
11     }
12 }
13
```

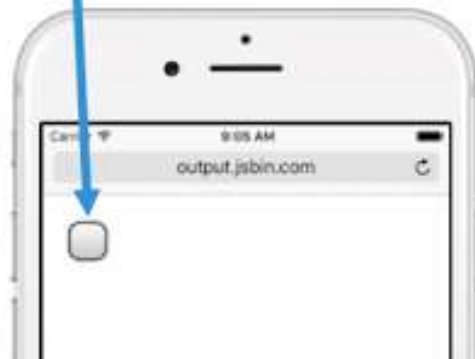


```
@Component({  
  selector: "checkbox"  
  templateUrl: "checkbox.html"  
});
```



<switch>

<input type="checkbox">



Популярность

- Facebook
- Instagram
- Airbnb
- Walmart
- Sberbank

OS & SmartTV

OS & SmartTV

WebOS — веб-приложение, организующее платформу (операционную среду с набором готовых функций API) для выполнения других веб-приложений

Tizen — открытая операционная система на базе ядра Linux, предназначенная для широкого круга устройств, включая смартфоны, интернет-планшеты, компьютеры, автомобильные информационно-развлекательные системы, «умные» телевизоры и цифровые камеры

Chrome OS — ОС от компании Google. Главной особенностью является доминирование веб-приложений над обычными функциями ОС

**Я, представляющий
себя в мире IT:**



**Я, оказавшись в
мире IT:**

