React concurrency

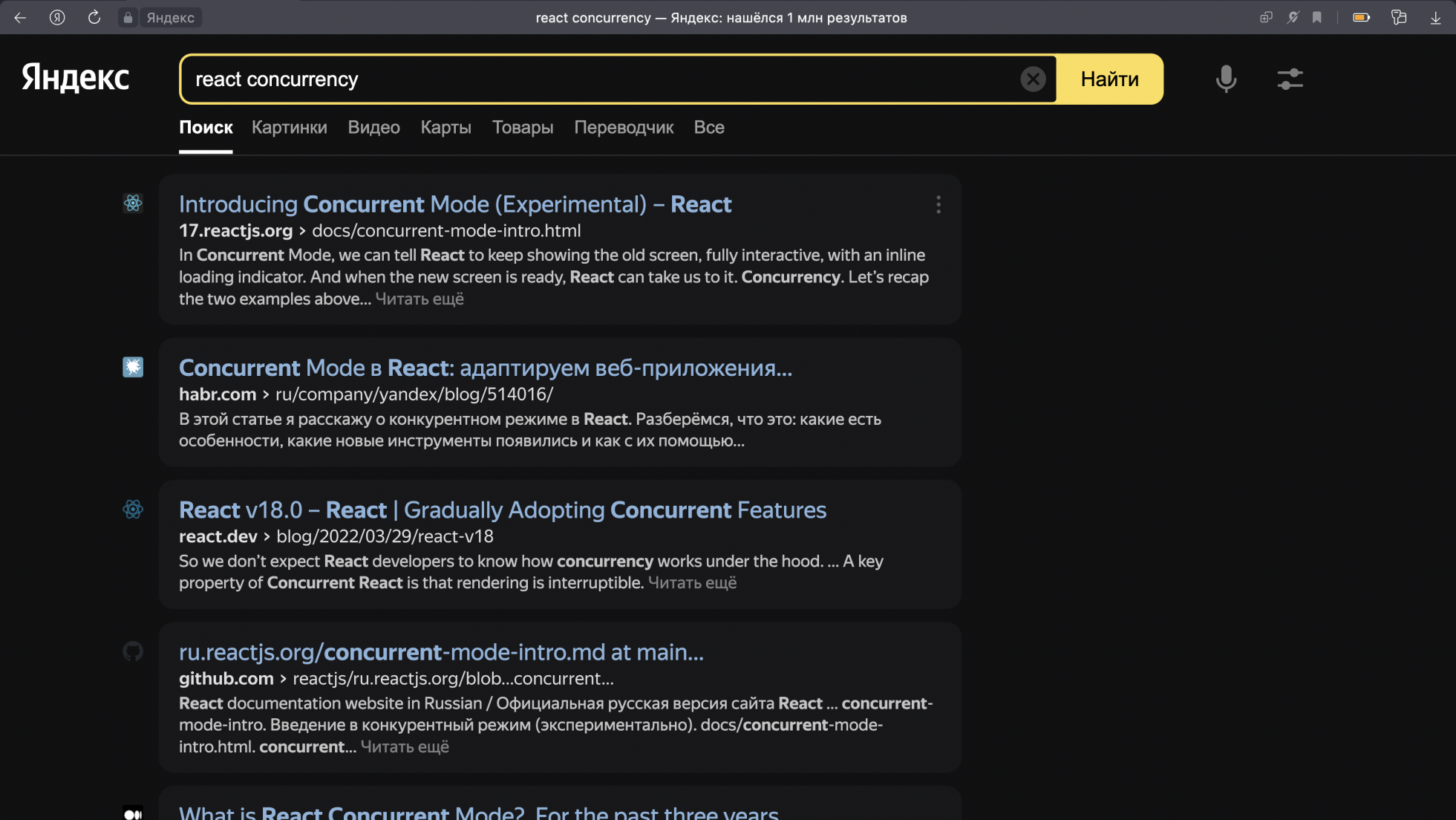
Кто я?

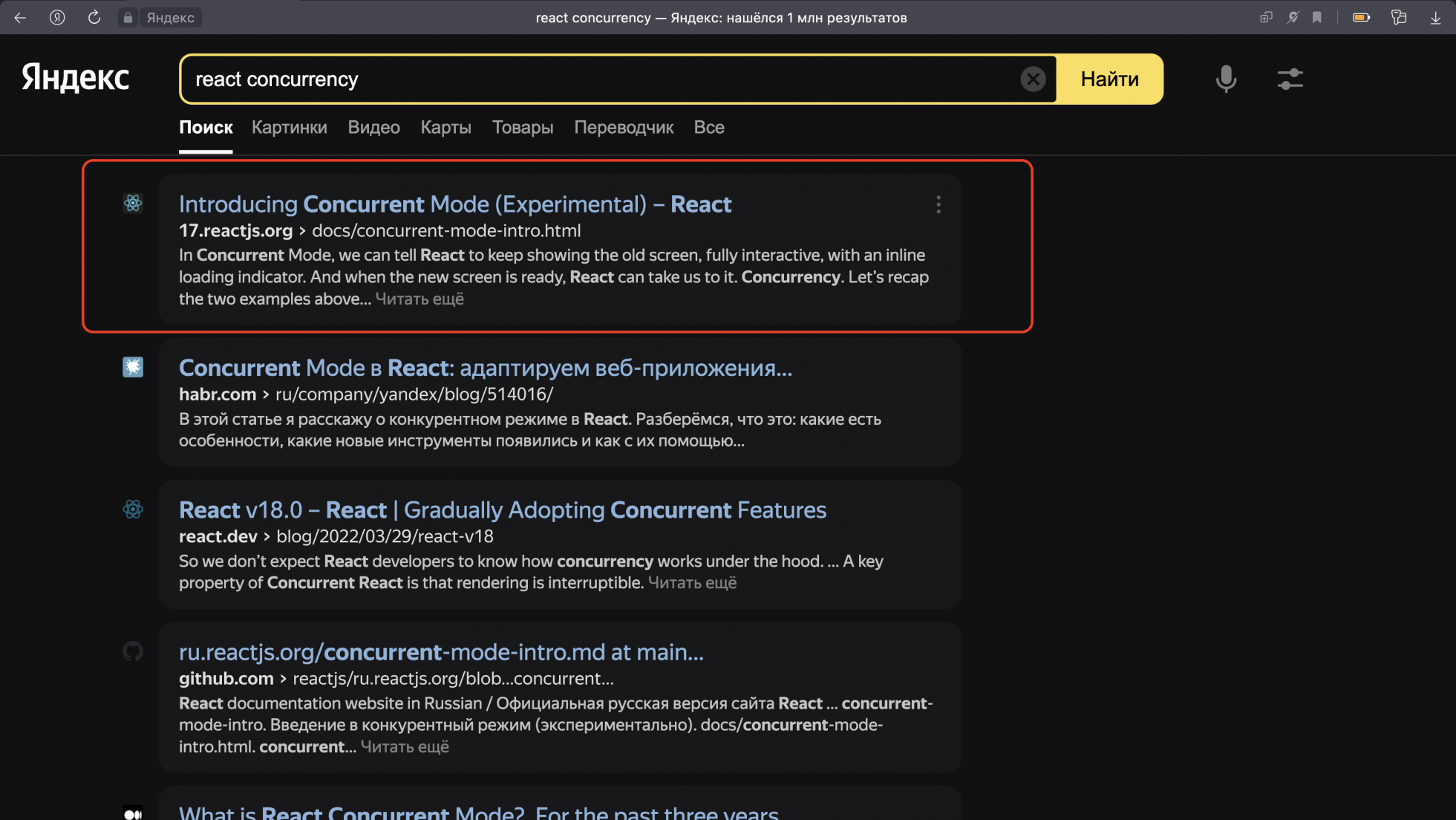
- Дима Грош
- React developer
- Работаю в Wis Software

Disclaimer

Это фронтенд. Это не бэкенд

Сегодня ничего не будет про SSR.


Яндекс
react concurrency — Яндекс: нашёлся 1 млн результатов
Яндекс
react concurrency
Найти
Поиск
Картинки
Видео
Карты
Товары
Переводчик
Все
Introducing Concurrent Mode (Experimental) – React
17.reactjs.org › docs/concurrent-mode-intro.html
In Concurrent Mode, we can tell React to keep showing the old screen, fully interactive, with an inline loading indicator. And when the new screen is ready, React can take us to it. Concurrency. Let's recap the two examples above... Читать ещё
Concurrent Mode в React: адаптируем веб-приложения...
habr.com › ru/company/yandex/blog/514016/
В этой статье я расскажу о конкурентном режиме в React. Разберёмся, что это: какие есть особенности, какие новые инструменты появились и как с их помощью...
React v18.0 – React | Gradually Adopting Concurrent Features
react.dev › blog/2022/03/29/react-v18
So we don't expect React developers to know how concurrency works under the hood. ... A key property of Concurrent React is that rendering is interruptible. Читать ещё
ru.reactjs.org/concurrent-mode-intro.md at main...
github.com › reactjs/ru.reactjs.org/blob...concurrent...
React documentation website in Russian / Официальная русская версия сайта React ... concurrent-mode-intro. Введение в конкурентный режим (экспериментально). docs/concurrent-mode-intro.html. concurrent... Читать ещё
What is React Concurrent Mode? For the past three years


Яндекс
react concurrency — Яндекс: нашёлся 1 млн результатов
Яндекс
react concurrency
Найти
Поиск
Картинки
Видео
Карты
Товары
Переводчик
Все
Introducing Concurrent Mode (Experimental) – React
17.reactjs.org › docs/concurrent-mode-intro.html
In Concurrent Mode, we can tell React to keep showing the old screen, fully interactive, with an inline loading indicator. And when the new screen is ready, React can take us to it. Concurrency. Let's recap the two examples above... Читать ещё
Concurrent Mode в React: адаптируем веб-приложения...
habr.com › ru/company/yandex/blog/514016/
В этой статье я расскажу о конкурентном режиме в React. Разберёмся, что это: какие есть особенности, какие новые инструменты появились и как с их помощью...
React v18.0 – React | Gradually Adopting Concurrent Features
react.dev › blog/2022/03/29/react-v18
So we don't expect React developers to know how concurrency works under the hood. ... A key property of Concurrent React is that rendering is interruptible. Читать ещё
ru.reactjs.org/concurrent-mode-intro.md at main...
github.com › reactjs/ru.reactjs.org/blob...concurrent...
React documentation website in Russian / Официальная русская версия сайта React ... concurrent-mode-intro. Введение в конкурентный режим (экспериментально). docs/concurrent-mode-intro.html. concurrent... Читать ещё
What is React Concurrent Mode? For the past three years

Introducing Concurrent Mode (Experimental)

Caution:
This page is **somewhat outdated** and only exists for historical purposes.

React 18 was released with support for concurrency. However, **there is no "mode" anymore,** and the new behavior is fully opt-in and only enabled when you use the new features.

For up-to-date high-level information, refer to:

- React 18 Announcement
- Upgrading to React 18
- React Conf 2021 Videos

For details about concurrent APIs in React 18, refer to:

- `React.Suspense` reference
- `React.startTransition` reference
- `React.useTransition` reference
- `React.useDeferredValue` reference

The rest of this page includes content that's stale, broken, or incorrect.

React Concurrency - это новый механизм, представленный в React 18, который позволяет приложениям React более эффективно использовать ресурсы устройства и улучшить пользовательский интерфейс, предоставляя отзывчивость и быстродействие приложениям, даже при выполнении тяжелых вычислительных задач.

Он основан на идее о том, что в приложениях React есть два типа работы: работа в пользовательском интерфейсе и работа вне пользовательского интерфейса. Работа в пользовательском интерфейсе должна быть быстрой и отзывчивой, чтобы приложение чувствовалось плавным и приятным в использовании. Работа вне пользовательского интерфейса может быть более тяжелой и длительной, и может включать в себя вычисления, сетевые запросы или другие задачи, которые не имеют непосредственного отношения к пользовательскому интерфейсу.

React Concurrency позволяет приложениям React запускать работу вне пользовательского интерфейса и возвращаться к работе в пользовательском интерфейсе в любой момент без блокировки пользовательского интерфейса. Он также позволяет приложениям приоритезировать работу в пользовательском интерфейсе и оптимизировать использование ресурсов устройства для достижения максимальной производительности и отзывчивости.

Первая проблема*

Нейминг

Нейминг

1. Async Rendering (Fiber, TimeSlicing)
2. Concurrent react
3. Concurrent mode
4. Concurrent rendering
5. Concurrent features

Проблемы пользователя

Фонд золотых цитат

1. Ничего не работает
2. У меня все сломалось
3. Памагите !

Не отзывчивый интерфейс

Blocking rendering

David de Gea

T. Heaton

N. Bishop

J. Butland

R. Vítek

V. Lindelöf

P. Jones

H. Maguire

Lisandro Martínez

T. Malacia

R. Varane

Diogo Dalot

L. Shaw

A. Wan-Bissaka

B. Williams

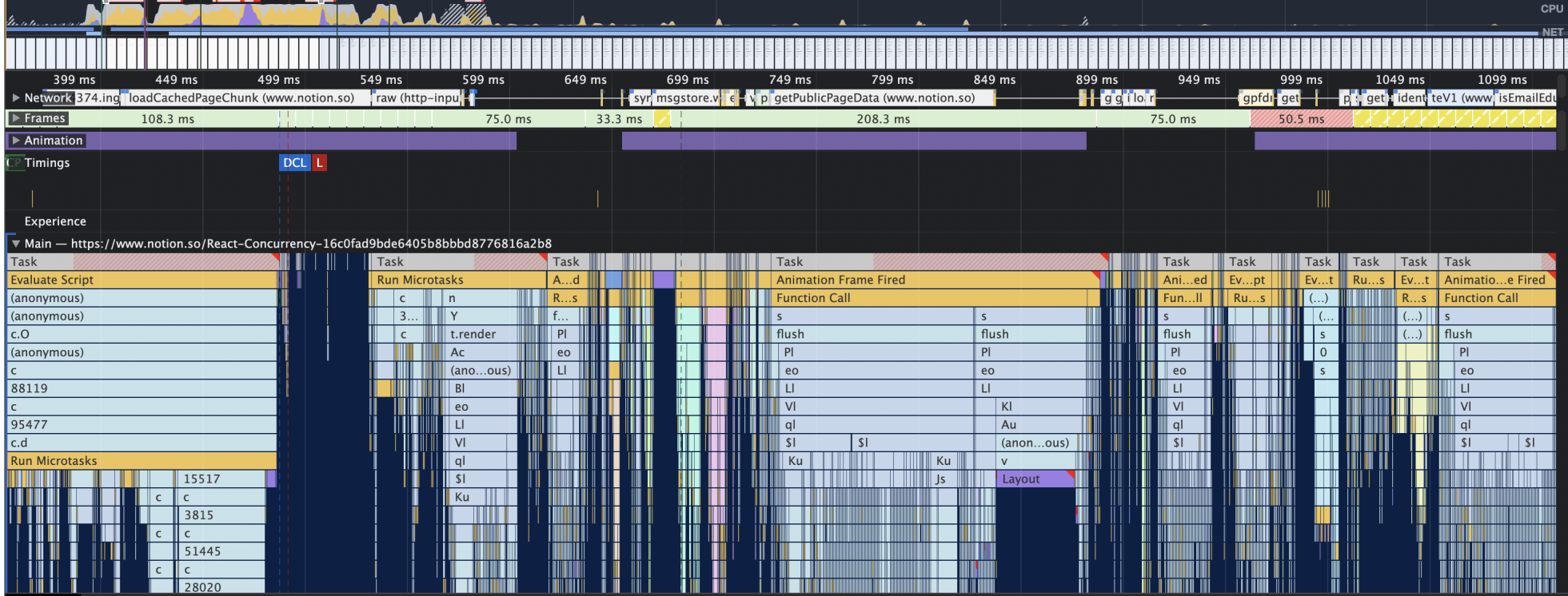
Not selected Player

```ts
const sleep = (ms: number) => {

  const start = performance.now();

  while (performance.now() - start < ms);
};
```

Main thread


CPU
NET
399 ms
449 ms
499 ms
549 ms
599 ms
649 ms
699 ms
749 ms
799 ms
849 ms
899 ms
949 ms
999 ms
1049 ms
1099 ms
Network
loadCachedPageChunk (www.notion.so)
getPublicPageData (www.notion.so)
Frames
108.3 ms
75.0 ms
33.3 ms
208.3 ms
75.0 ms
50.5 ms
Animation
Timings
DCL
L
Experience
Main — https://www.notion.so/React-Concurrency-16c0fad9bde6405b8bbbd8776816a2b8
Task
Evaluate Script
Run Microtasks
Animation Frame Fired
Function Call
(anonymous)
c.O
c.d
flush
Summary
Bottom-Up
Call Tree
Event Log
Range: 353 ms – 1.11 s
758 ms
3 ms Loading
613 ms Scripting
66 ms Rendering
3 ms Painting
33 ms System
40 ms Idle
758 ms Total

Джейк Арчибальд. В цикле - JSConf.Asia

Что выполняется в main thread*

1. Tasks
2. MicroTasks
3. RequestAnimationFrame
4. Style
5. Layout
6. Paint
7. Composite

А давайте все вынесем в другой поток

master 4 branches 4 tags Go to file Add file Code

oskarer Merge pull request #17 from tizmagik/patch-1 ... 8fc5c01 on Jan 11, 2017 65 commits

src	Prevent default events	8 years ago
test	Added up/down button in todo examples	8 years ago
.gitignore	Initial commit	8 years ago
LICENSE	Updated package.json, exported main file	8 years ago
README.md	Update README.md	7 years ago
package.json	Updated package, added a build-demo step	8 years ago

README.md

React Renderer using Web Workers

A React Custom renderer using Web Workers. All the Virtual DOM reconcilliations happen in a WebWorker thread. Only node updates are sent over to the UI thread, result in a much more responsive UI.

An existing React application can leverage WebWorkers using this library with minimal change. Look at the usage section for details.

Demo

The demo is hosted at http://web-perf.github.io/react-worker-dom/. To run a local version of the demo,

- Clone the repo run `npm install` to install all dependencies.

README.md

React Renderer using Web Workers

A React Custom renderer using Web Workers. All the Virtual DOM reconcilliations happen in a WebWorker thread. Only node updates are sent over to the UI thread, result in a much more responsive UI.

An existing React application can leverage WebWorkers using this library with minimal change. Look at the usage section for details.

web-perf / react-worker-dom Public

Watch 33 Fork 58 Star 921

Code Issues 4 Pull requests 2 Actions Projects Wiki Security Insights

master 4 branches 4 tags Go to file Add file Code

oskarer Merge pull request #17 from tizmagik/patch-1 ... 8fc5c01 on Jan 11, 2017 65 commits

src	Prevent default events	8 years ago
test	Added up/down button in todo examples	8 years ago
.gitignore	Initial commit	8 years ago
LICENSE	Updated package.json, exported main file	8 years ago
README.md	Update README.md	7 years ago
package.json	Updated package, added a build-demo step	8 years ago

README.md

React Renderer using Web Workers

A React Custom renderer using Web Workers. All the Virtual DOM reconcilliations happen in a WebWorker thread. Only node updates are sent over to the UI thread, result in a much more responsive UI.

An existing React application can leverage WebWorkers using this library with minimal change. Look at the usage section for details.

Demo

The demo is hosted at http://web-perf.github.io/react-worker-dom/. To run a local version of the demo,

- Clone the repo run `npm install` to install all dependencies.

About

Experiments to see the advantages of using Web Workers to Render React Virtual DOM

web-perf.github.io/react-worker-dom

Readme

BSD-3-Clause license

921 stars

33 watching

58 forks

Report repository

Releases

4 tags

Packages

No packages published

Used by 9

+ 1

Contributors 5

Demo

The demo is hosted at http://web-perf.github.io/react-worker-dom/. To run a local version of the demo,

- Clone the repo run `npm install` to install all dependencies.
- Build the app using `npm run demo`
- Open `http://localhost:8080/test/dbmonster/` to view the demo app, or `http://localhost:8080/test/todo` for the todo app.
- Tweak the params in the URL to change to use web workers, increase number of components, etc.

React Worker Dom

A ReactJS custom renderer using Web Workers.

React is fast, thanks to the VirtualDOM. Using a diffing algorithm, the browser DOM nodes are manipulated only when there is a state change. This algorithm is computationally expensive. Using webworkers to perform the calculations can make React even faster.

Examples

DBMonster

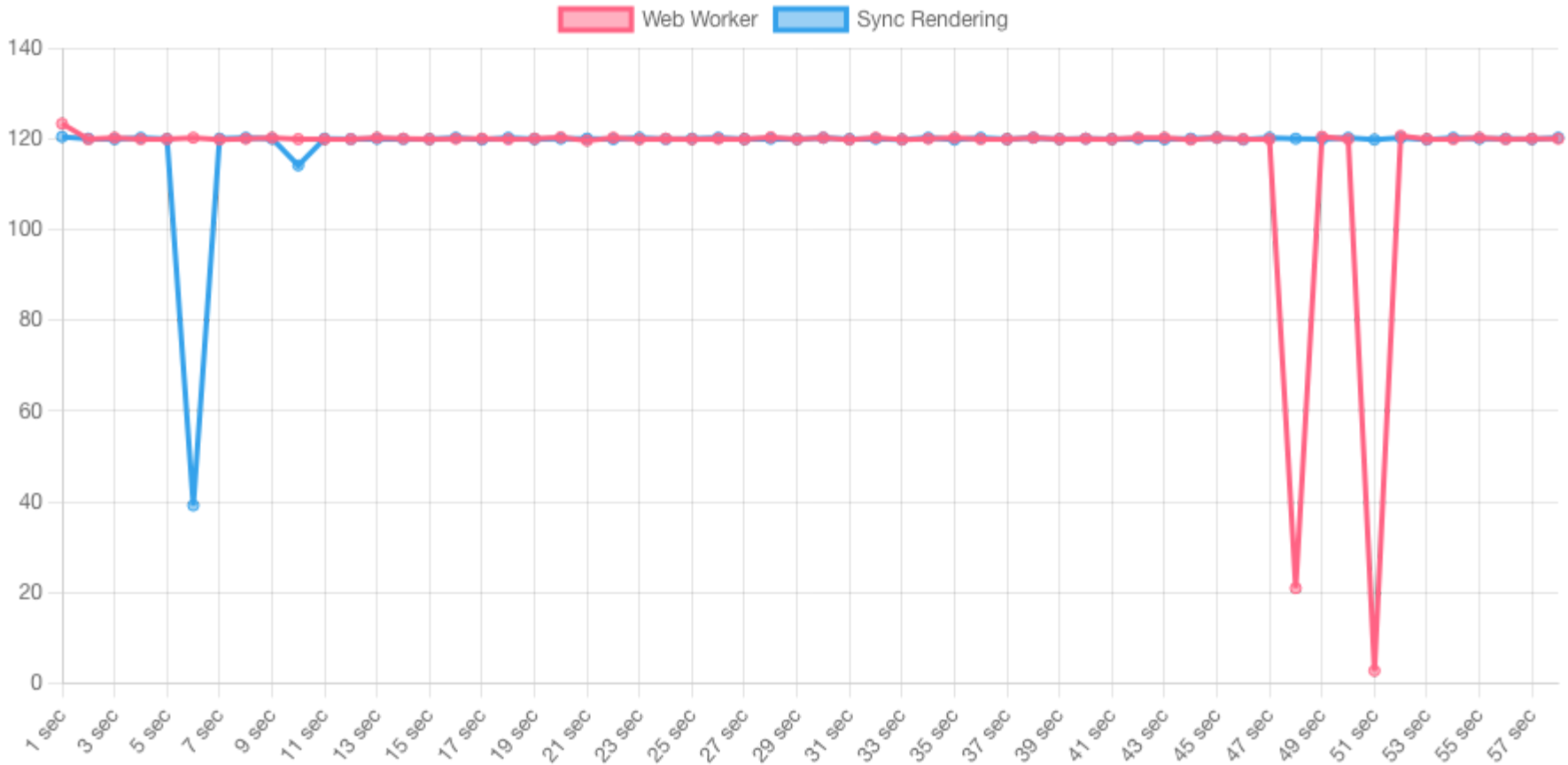
Debuted in a session at ReactConf 2015 and has been used to benchmark many javascript frameworks. It shows an application simulating DB queries.
To see frame rates, open Chrome dev tools, enable *Show fps meter* in Console >

Todo Sample

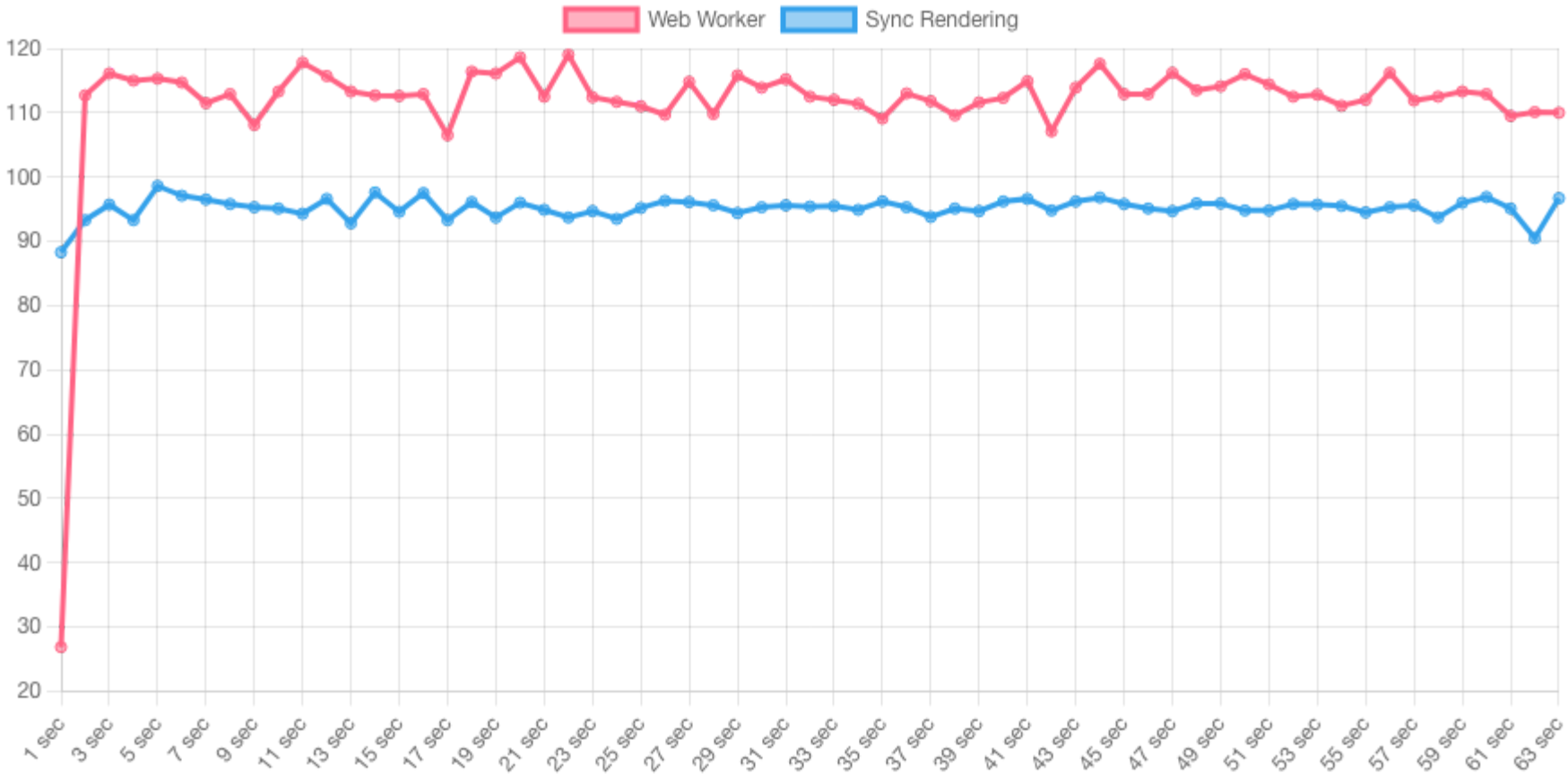
The canonical app that every frontend framework uses as a demo. Shows how events are passed from the UI thread to the worker, and how DOM manipulations as sent back to the UI thread from the worker thread.

cluster1slave	4	11.81	11.54	5.94	1.94	-
cluster2	5	14.97	12.00	7.20	4.96	0.40
cluster2slave	3	11.54	8.60	0.73	-	-
cluster3	1	8.06	-	-	-	-
cluster3slave	8	14.63	12.73	9.22	8.81	8.48
cluster4	3	13.20	10.26	0.16	-	-
cluster4slave	5	13.96	11.04	9.44	7.91	7.37
cluster5	6	11.32	10.22	6.58	5.32	4.28
cluster5slave	9	14.82	13.45	13.39	13.31	12.06
cluster6	6	11.86	10.96	7.51	6.40	6.30
cluster6slave	5	13.36	6.58	4.39	3.28	0.39
cluster7	6	14.81	12.74	11.56	4.49	4.31
cluster7slave	6	14.66	12.25	12.13	11.73	11.18
cluster8	8	14.28	11.75	10.64	8.97	4.94
cluster8slave	3	11.78	8.37	0.35	-	-

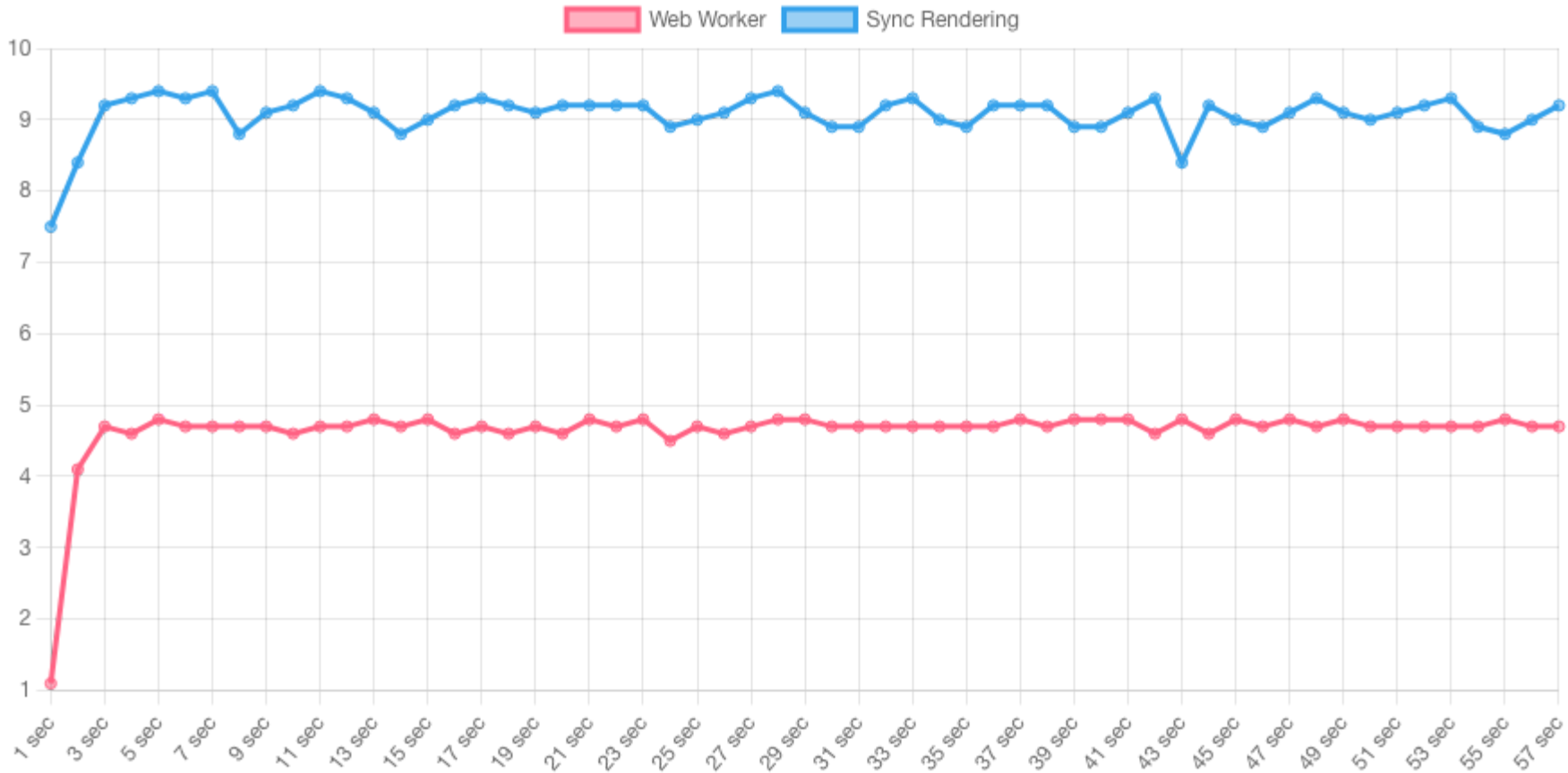
fps при 10 элементах



fps при 100 элементах



fps при 1000 элементах



David de Gea

T. Heaton

N. Bishop

J. Butland

V. Lindelöf

P. Jones

H. Maguire

Lisandro Martínez

T. Malacia

R. Varane

Diogo Dalot

L. Shaw

A. Wan-Bissaka

B. Williams

R. Bennett

Bruno Fernandes

C. Eriksen

M. Sabitzer

Fred

Casemiro

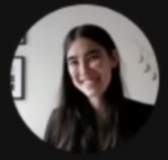
F. Pellistri

D. van de Beek

Not selected Player

Почему не взлетело ?

1. Веб-воркер не имеет доступа к DOM браузера
2. Нужно создавать еще одну обертку над Synthetic events
3. Не нужно при маленьком числе node
4. Не спасает при очень большом числе node



Sophie Alpert
@sophiebits

Replying to @code_punkt

> offloading reconciliation and vdom things to a web worker

We haven't built this because we don't believe it would help perf in most cases, but our upcoming async rendering features will hopefully help here.

2:18 PM · May 18, 2018 · Twitter Web Client

2016

Async Rendering

впервые ввели понятие асинхронного рендеринга

```
const [selectedPlayer, setSelectedPlayer] = useState<IPlayer | null>(null);

const handlePlayerClick = (player: IPlayer) => {

  setTimeout(() => {

    setSelectedPlayer(player);

  }, 0);
};
```

Async Rendering

David de Gea

T. Heaton

N. Bishop

J. Butland

R. Vítek

V. Lindelöf

P. Jones

H. Maguire

Lisandro Martínez

T. Malacia

R. Varane

Diogo Dalot

L. Shaw

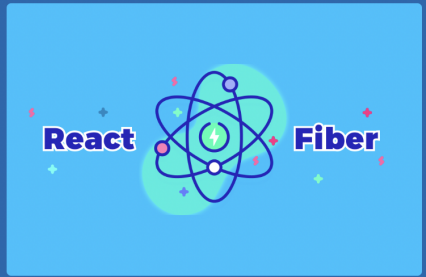
A. Wan-Bissaka

B. Williams

Not selected Player

React Fiber

Inside Fiber: in-depth overview of the new reconciliation algorithm in React



Dive deep into React's new architecture called Fiber and learn about two main phases of the new reconciliation algorithm. We'll take a detailed look at how React updates state and props and processes children.

React | React-Fiber | Reconciliation

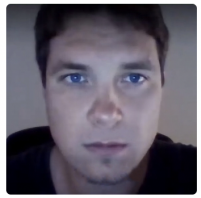
15 March 2020 | 19 min read | 0 comments

React is a JavaScript library for building user interfaces. At its core lies the mechanism that tracks changes in a component state and projects the updated state to the screen. In React we know this process as **reconciliation**. We call the `setState` method and the framework checks if the state or props have changed and re-renders a component on UI.

React's docs provide a good high-level overview of the mechanism: the role of React elements, lifecycle methods and the `render` method, and the diffing algorithm applied to a component's children. The tree of immutable React elements returned from the `render` method is commonly known as the "virtual DOM". That term helped explain React to people early on, but it also caused confusion and isn't used in the React documentation anymore. In this article I'll stick to calling it a tree of React elements.

Besides the tree of React elements, the framework has always had a tree of internal instances (components, DOM nodes etc.) used to keep the state.Starting from version 16, React rolled out a new implementation of that internal instances tree and the algorithm that manages it code-named **Fiber**. To learn about the advantages which the Fiber architecture brings check out The how and why on

ABOUT THE AUTHOR



Max Koretskyi

Principal Engineer at kawa.ai.. Founder indepth.dev. Big fan of software engineering, Web Platform & JavaScript. Man of Science & Philosophy.

Что сделал Fiber ?

1. React выполняет работу в два основных этапа: render и commit.
2. Результатом фазы является дерево узлов Fiber, отмеченных побочными эффектами.
3. Работа на первом этапе render выполняется асинхронно.
4. Напротив, следующая фаза commit всегда синхронна.


2016
Async Rendering
впервые ввели понятие асинхронного рендеринга
2018
Concurrent React
переименовали в Concurrent React

А в чем разница ?

1. Асинхронность - слишком широкое понятие
2. Происходит конкуренция между компонентами
3. У нас есть приоритеты, а не четкий цикл


2016
Async Rendering
впервые ввели понятие асинхронного рендеринга
2018
Concurrent React
переименовали в Concurrent React
2019
Concurrent Mode
появилась возможность попробовать на практике фичи

Concurrent rendering

David de Gea

T. Heaton

N. Bishop

J. Butland

R. Vítek

V. Lindelöf

P. Jones

H. Maguire

Lisandro Martínez

T. Malacia

R. Varane

Diogo Dalot

L. Shaw

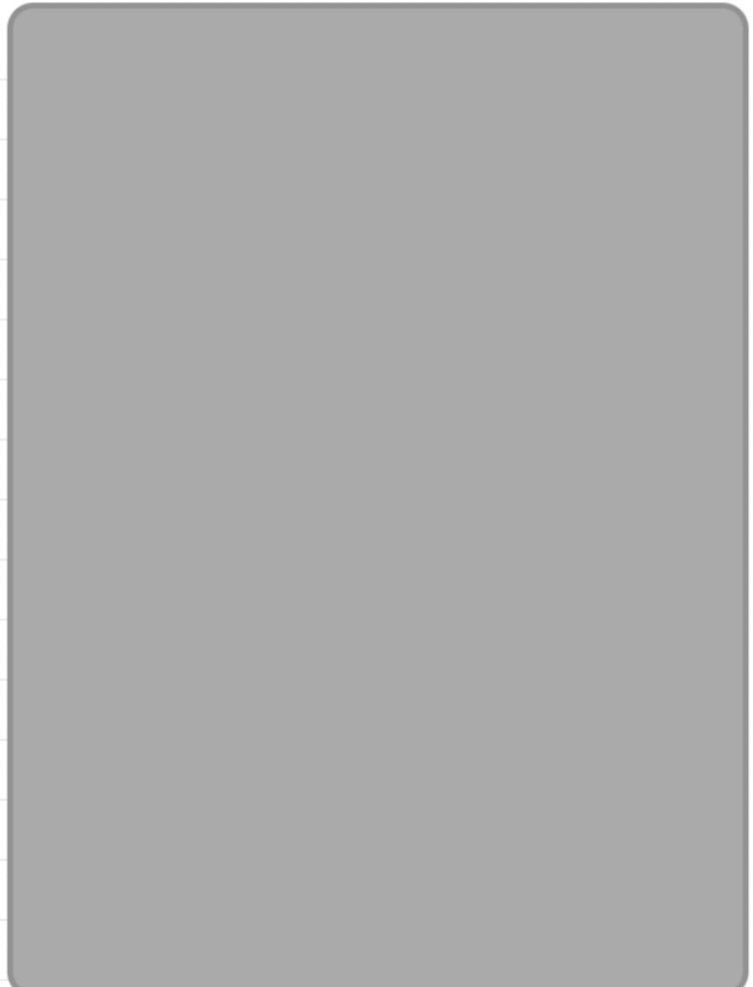
A. Wan-Bissaka

B. Williams

Not selected Player

Feature Comparison

	Legacy Mode
String Refs	✅
Legacy Context	✅
findDOMNode	✅
Suspense	✅
SuspenseList	🚫
Suspense SSR + Hydration	🚫
Progressive Hydration	🚫
Selective Hydration	🚫
Cooperative Multitasking	🚫
Automatic batching of multiple setStates	🚫*
Priority-based Rendering	🚫
Interruptible Prerendering	🚫
useTransition	🚫
useDeferredValue	🚫
Suspense Reveal “Train”	🚫



Feature Comparison

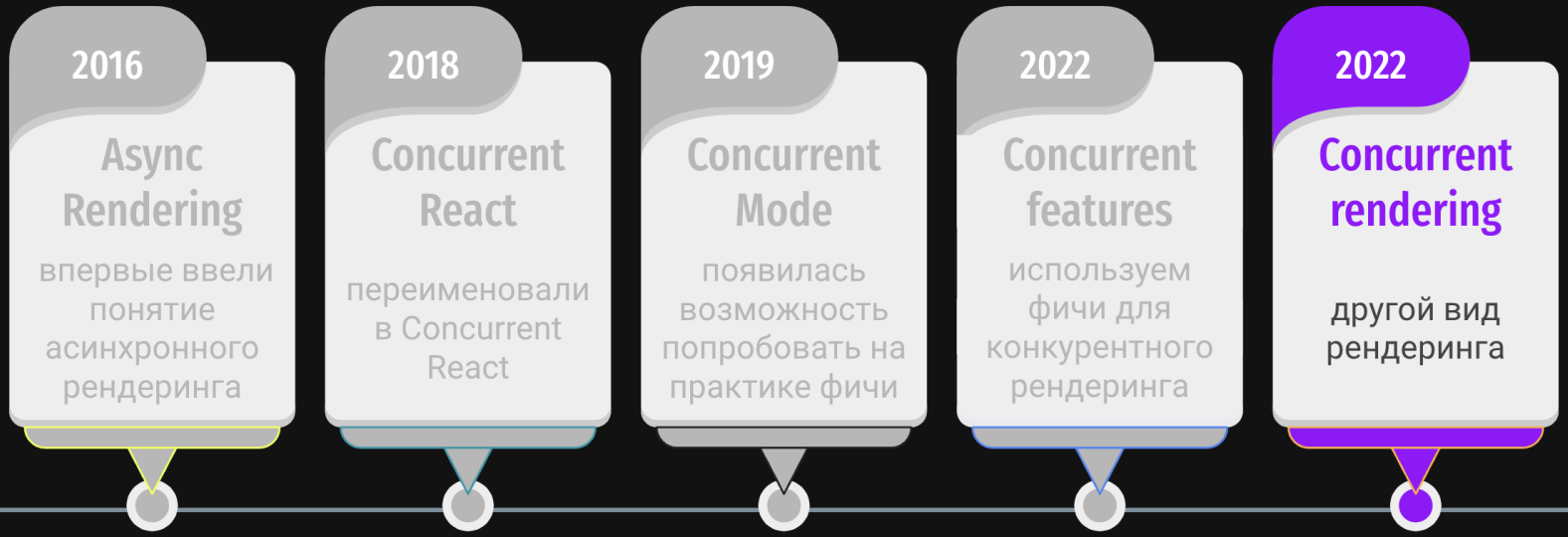
	Legacy Mode		Concurrent Mode
String Refs	✅		🚫**
Legacy Context	✅		🚫**
findDOMNode	✅		🚫**
Suspense	✅		✅
SuspenseList	🚫		✅
Suspense SSR + Hydration	🚫		✅
Progressive Hydration	🚫		✅
Selective Hydration	🚫		✅
Cooperative Multitasking	🚫		✅
Automatic batching of multiple setStates	🚫*		✅
Priority-based Rendering	🚫		✅
Interruptible Prerendering	🚫		✅
useTransition	🚫		✅
useDeferredValue	🚫		✅
Suspense Reveal “Train”	🚫		✅

Feature Comparison

	Legacy Mode	Blocking Mode	Concurrent Mode
String Refs	✅	🚫**	🚫**
Legacy Context	✅	🚫**	🚫**
findDOMNode	✅	🚫**	🚫**
Suspense	✅	✅	✅
SuspenseList	🚫	✅	✅
Suspense SSR + Hydration	🚫	✅	✅
Progressive Hydration	🚫	✅	✅
Selective Hydration	🚫	🚫	✅
Cooperative Multitasking	🚫	🚫	✅
Automatic batching of multiple setStates	🚫*	✅	✅
Priority-based Rendering	🚫	🚫	✅
Interruptible Prerendering	🚫	🚫	✅
useTransition	🚫	🚫	✅
useDeferredValue	🚫	🚫	✅
Suspense Reveal “Train”	🚫	🚫	✅

“Concurrent mode” -> Concurrent features


2016
Async Rendering
впервые ввели понятие асинхронного рендеринга
2018
Concurrent React
переименовали в Concurrent React
2019
Concurrent Mode
появилась возможность попробовать на практике фичи
2022
Concurrent features
используем фичи для конкурентного рендеринга


2016
Async Rendering
впервые ввели понятие асинхронного рендеринга
2018
Concurrent React
переименовали в Concurrent React
2019
Concurrent Mode
появилась возможность попробовать на практике фичи
2022
Concurrent features
используем фичи для конкурентного рендеринга
2022
Concurrent rendering
другой вид рендеринга

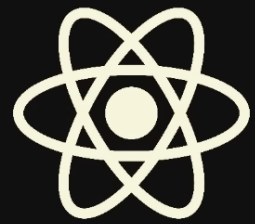
Concurrent features

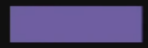
Transitions

1. useTransition
2. startTransition
3. useDeferredValue

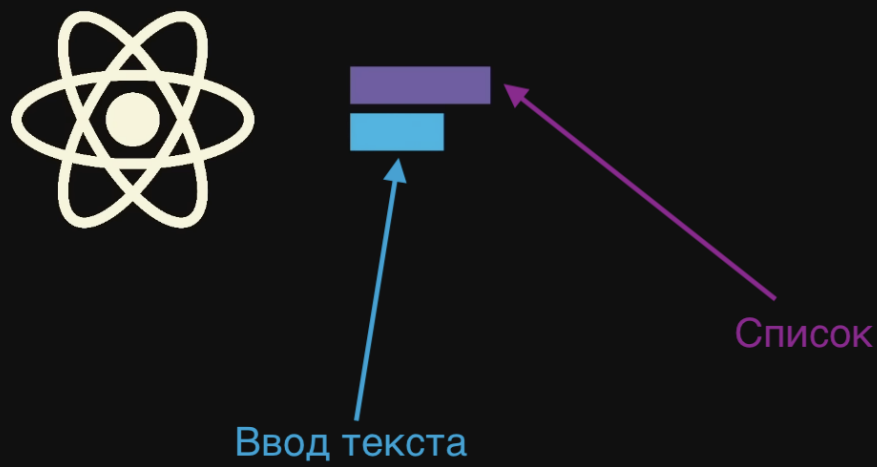
Other

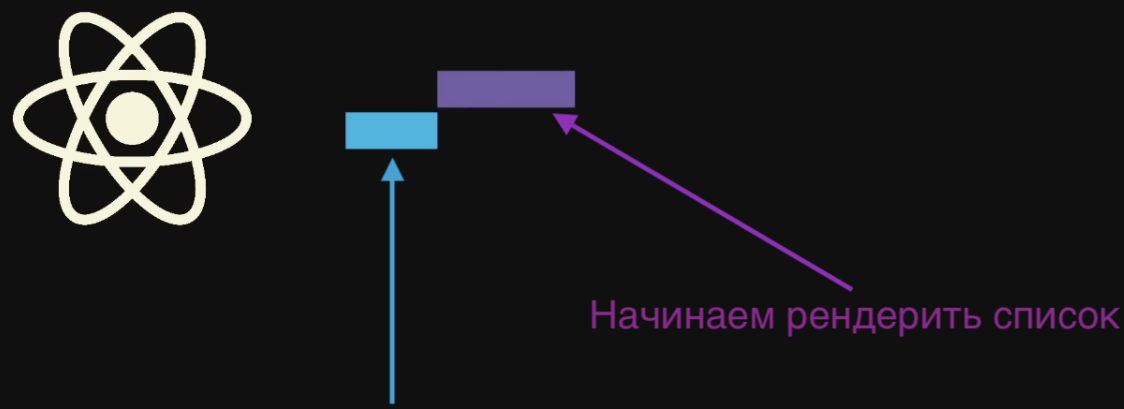
1. Automatic Batching
2. Suspense
3. Client and Server Rendering APIs

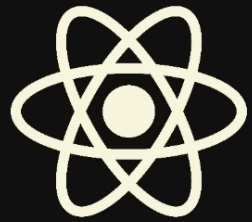


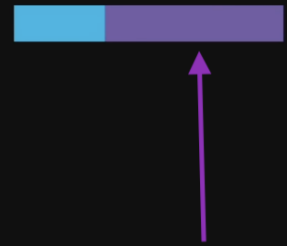


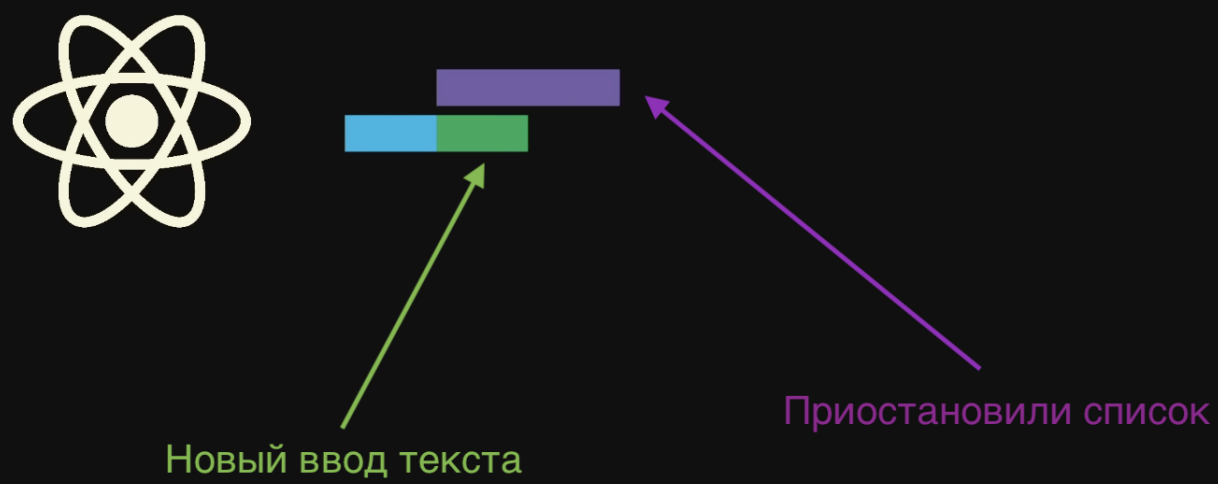
Рендеринг списка

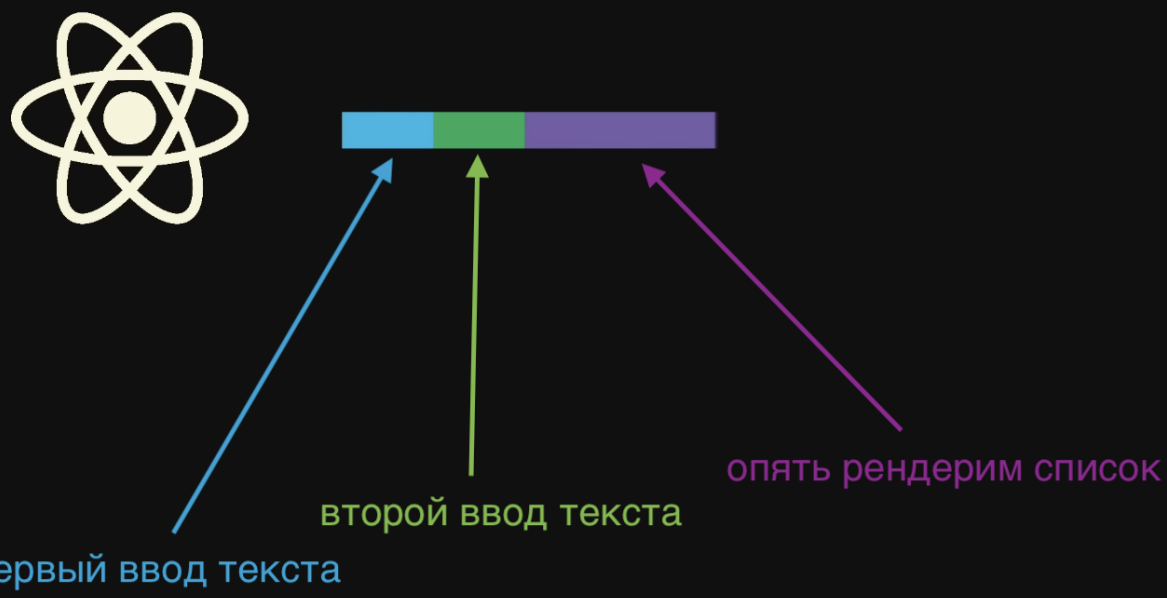

Список
Ввод текста

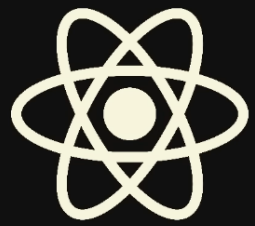

Начинаем рендерить список
Завершили ввод текста

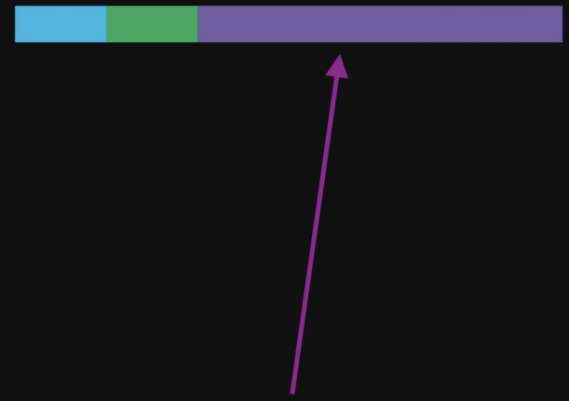



Опять рендерим список


Новый ввод текста
Приостановили список


опять рендерим список
второй ввод текста
Первый ввод текста




Завершили отрисовку списка

Update

```
export const Update = () => {
  const [count, setCount] = useState(0);
  const render = useRef(0);

  const handleIncrement = () => {
    setCount((prev) => prev + 1)
  };

  const handleSame = () => {
    setCount(count);
  };

  render.current += 1;

  return (
    <div>
          <p>Counter {count}</p>
          <button onClick={handleIncrement}>Handle increment</button>
          <button onClick={handleSame}>Handle same</button>
          <p>Rerender Update {render.current}</p>
    </div>
    );
};
```

Counter: 0

Handle increment

Handle same

Rerender: 1

Категории Update

1. High priority (Urgent) Updates
2. Low priority (Non-Urgent) Updates

High priority (Urgent) updates

1. useState
2. useReducer
3. useSyncExternalStore

Low priority (Non-Urgent) updates

1. useTransition
2. startTransition
3. useDeferredValue

```
export const ConcurrentRendering = () => {

  const handlePlayerClick = (player) => {

    // high priority (urgent update)
    setHighPriorityPlayer(player);

    // low priority (non-urgent update)
    startTransition(() => {
      setLowPriorityPlayer(player);
    });
  };

  return (
    <div>
          {playersArray.map((player) => (
                <button onClick={handlePlayerClick}>
                    {player.name}
                </button>
          ))}
          <Statistics id={lowPriorityPlayer.id} />
        </div>
    );
};
```

Concurrent rendering

Clear

High Priority End:
player: "undefined" delayed player: "undefined"

High Priority Start:
player: "undefined" delayed player: "undefined"

High Priority End:
player: "undefined" delayed player: "undefined"

Low Priority Start:
player: "undefined" delayed player: "undefined"

Low Priority End:
player: "undefined" delayed player: "undefined"

David de Gea

T. Heaton

N. Bishop

J. Butland

R. Vítek

V. Lindelöf

P. Jones

H. Maguire

Lisandro Martínez

T. Malacia

R. Varane

Diogo Dalot

L. Shaw

A. Wan-Bissaka

B. Williams

Not selected Player

Transitions

1. Высокопроиоитетная таска прирывает низкоприорит кт ную
2. Если компонент начала рендерится, то он должен дойти до return
3. Компонент нельзя прервать
4. Низкоприоритетные таски могут возвращать свой предыдущий стейт

Низкоприоритетные таски могут возвращать свой предыдущий стейт

packages -> react-reconciler -> src ->
ReactFiberHooks.js

А что у других ?

Сигналы

Сигналы – это реактивные примитивы для управления состоянием приложения.

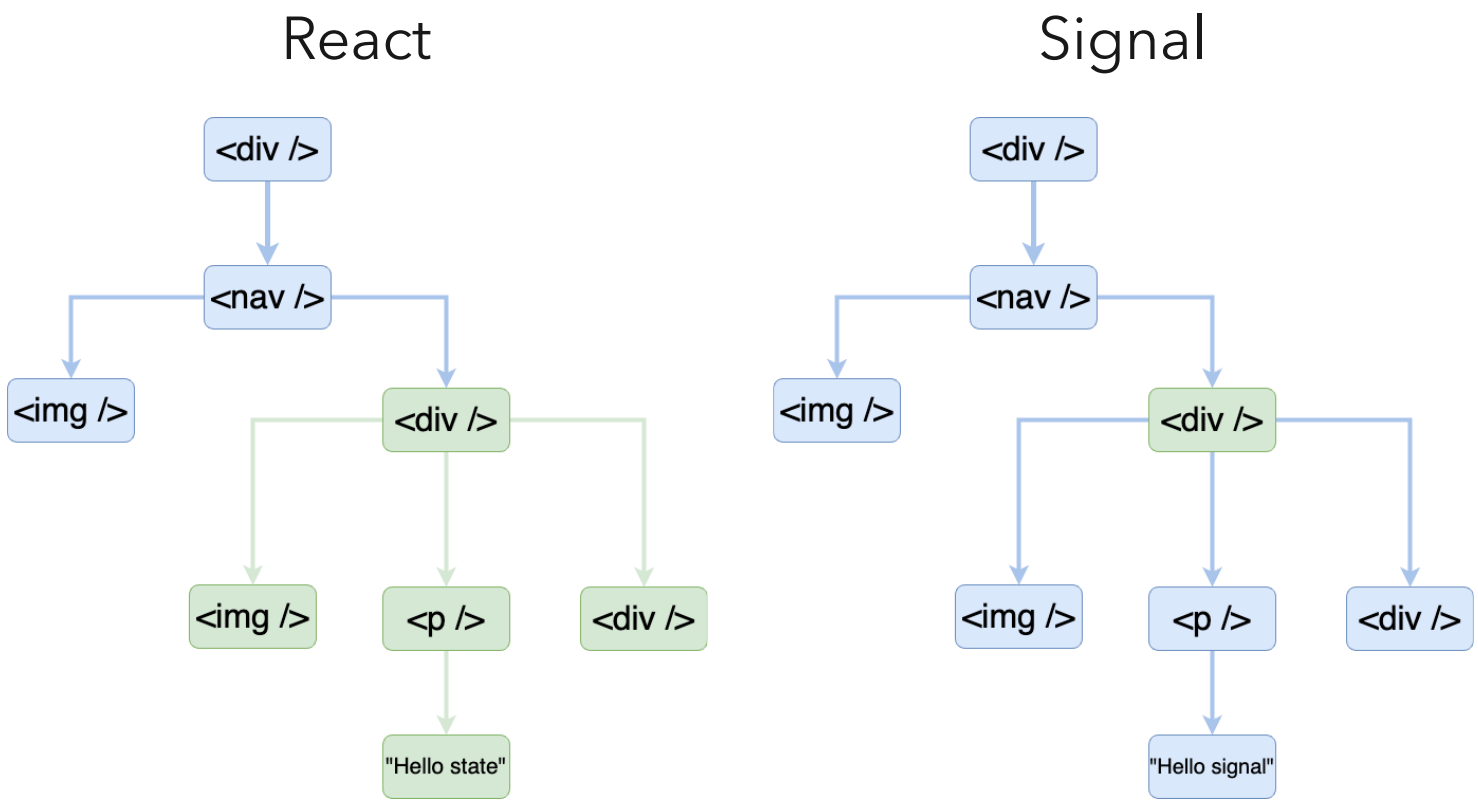
```
// react
const [state, setState] = useState(0);
// state -> value
// setState -> setter

// solidJs
const [signal, setSignal] = createSignal(0);
// signal -> getter
// setSignal -> setter

// preact
import { signal } from "@preact/signals-core";

const counter = signal(0);

// counter.value -> get value
// counter.value = 1 -> set value
```


React

<nav />
<img />

<img />
<p />

"Hello state"
Signal

<nav />
<img />

<img />
<p />

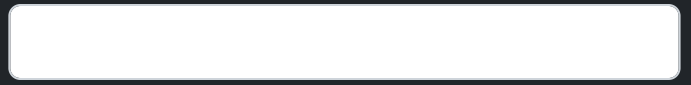
"Hello signal"

Кто использует ?

1. Solid JS
2. Vue
3. Qwik
4. Angular
5. Preact

Blocked RxJS Solution

Hello from Angular (blocked thread example)



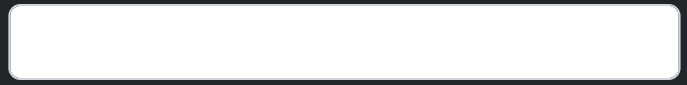
```
export class BlockedComponent implements OnInit {
  public dataControl = new FormControl('');

  private _longRequest$ = of({})
      .pipe(
          map(() => sleep(1000)),
          map(() => []),
      );

  public ngOnInit(): void {
    this.dataOptions$ = this.dataControl.valueChanges
            .pipe(
                startWith(''),
                map((value) => this._filter(value)),
                switchMap(() => this._longRequest$),
            );
  }

}
```

Hello from Angular (rxjs solution)



```
export class RxJsSolutionComponent implements OnInit {
  public dataControl = new FormControl(null);

  public todos$ = this.control.valueChanges
    .pipe(
      // map(() => []),
      switchMap(() => this._longRequest$())
    );

  private _longRequest$(): Observable<ITodo[]> {
    return this._http.get('https://jsonplaceholder.typicode.com/todos')
              .pipe(
                tap(() => sleep(1000)),
                map((todos) => todos),
              );
  }

}
```

Hello from Angular with signals!

[ENTER]
START SLEEP
END SLEEP
=== END LONG REQUEST

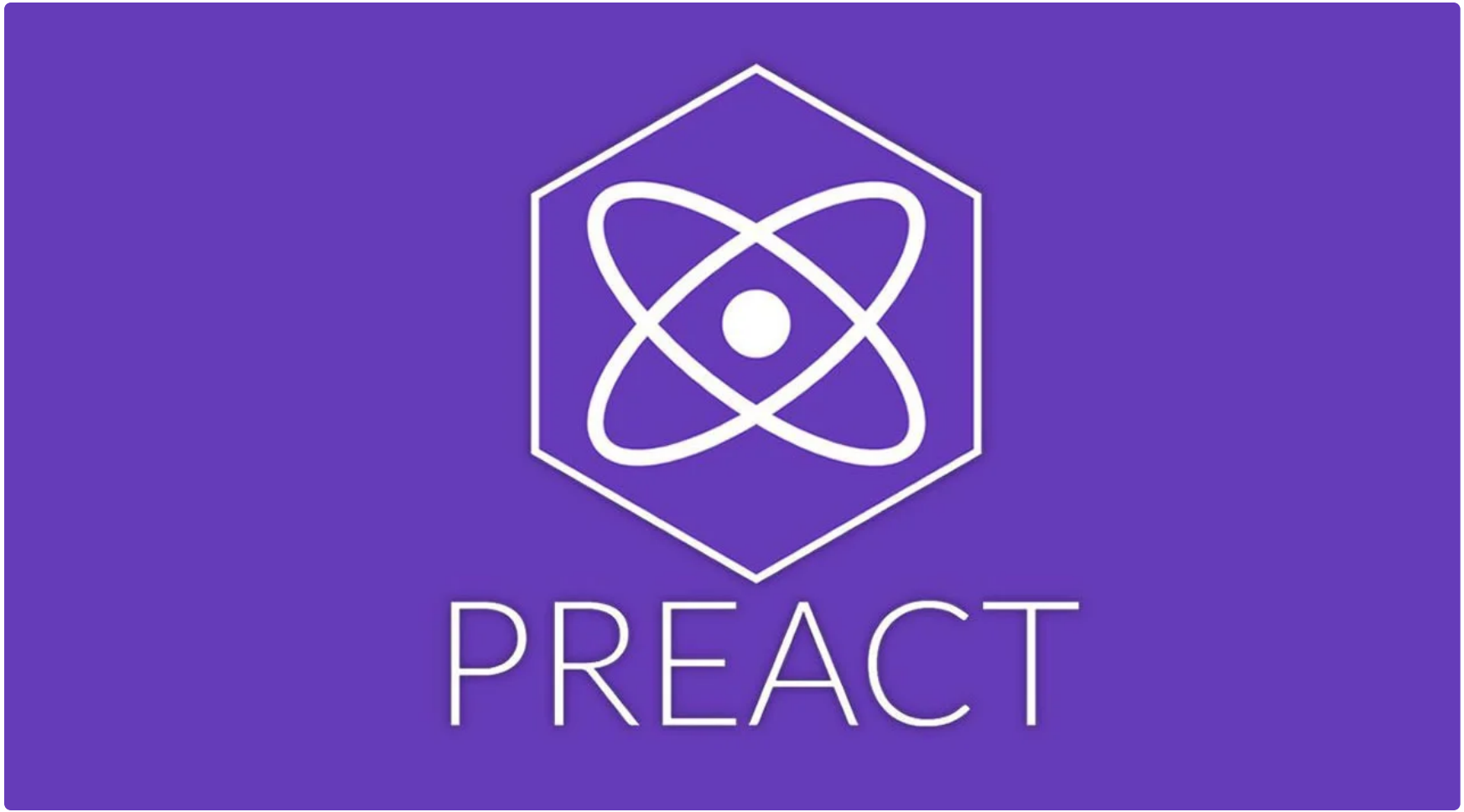
60 45 57 82 80 24 22 81 50 95 20 42 91 96 41

```
export class SignalsSolutionComponent implements OnInit {
  public readonly name = signal(null);
  public message = computed(() => `Hello ${this.name()}!`);

  public readonly todos$ = toObservable(this.name)
      .pipe(
        switchMap(() => this._longRequest$()),
      );

  private _longRequest$(): Observable<ITodo[]> {
    return this._http.get('https://jsonplaceholder.typicode.com/todos')
      .pipe(
        tap(() => sleep(1000)),
        map((todos) => todos),
      );
  }

}
```


PREACT

Async Blocking

Luxurious Plastic Soap
Bespoke Frozen Shoes
Luxurious Wooden Chair
Rustic Wooden Keyboard
Luxurious Wooden Pants
Oriental Fresh Towels
Awesome Soft Mouse
Elegant Bronze Salad
Generic Plastic Shirt
Bespoke Fresh Gloves
Handmade Cotton Car
Incredible Plastic Sausages
Luxurious Steel Bacon
Tasty Wooden Pizza
Handmade Concrete Cheese
Practical Metal Fish
Unbranded Rubber Fish
Gorgeous Metal Salad
Sleek Bronze Chicken
Luxurious Plastic Shirt
Modern Plastic Chicken
Small Steel Salad
Intelligent Granite Chips

```
import { signal } from '@preact/signals';

export const input = signal('');

export const Autocomplete = () => {
  const onInput = (event) => {
    const { value } = event.target;

    input.value = value;
  };

  return (
    <input value={input.value} onInput={onInput} />
    );
}
```

```
import { useComputed } from '@preact/signals';

import { list } from './const';
import { input } from './Autocomplete';

export const List = () => {
  const filteredList = useComputed(() => list.map((el) => el.name.includes(input.value)));

  sleep(2000);

  return (
    <div>
          {filteredList.value.map((el, index) => (
            <span key={el.id}>{el.name}</span>
          ))}
    </div>
    );
}
```

Async Blocking

Luxurious Plastic Soap
Bespoke Frozen Shoes
Luxurious Wooden Chair
Rustic Wooden Keyboard
Luxurious Wooden Pants
Oriental Fresh Towels
Awesome Soft Mouse
Elegant Bronze Salad
Generic Plastic Shirt
Bespoke Fresh Gloves
Handmade Cotton Car
Incredible Plastic Sausages
Luxurious Steel Bacon
Tasty Wooden Pizza
Handmade Concrete Cheese
Practical Metal Fish
Unbranded Rubber Fish
Gorgeous Metal Salad
Sleek Bronze Chicken
Luxurious Plastic Shirt
Modern Plastic Chicken
Small Steel Salad
Intelligent Granite Chips

```
import { signal } from '@preact/signals';

export const input = signal('');

export const Autocomplete = () => {
  const onInput = (event) => {
        setTimeout(() => {
            const { value } = event.target;
            input.value = value;
        }, 0);
  };


  return (
    <input value={input.value} onInput={onInput} />
  );
}
```

Вывод

1. Ничего не работает
2. У меня все сломалось
3. Памагите !

Спасибо за внимание !