React concurrency

Disclaimer

Это фронтенд. Это не бэкенд

Сегодня ничего не будет про SSR.

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

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

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.
- 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

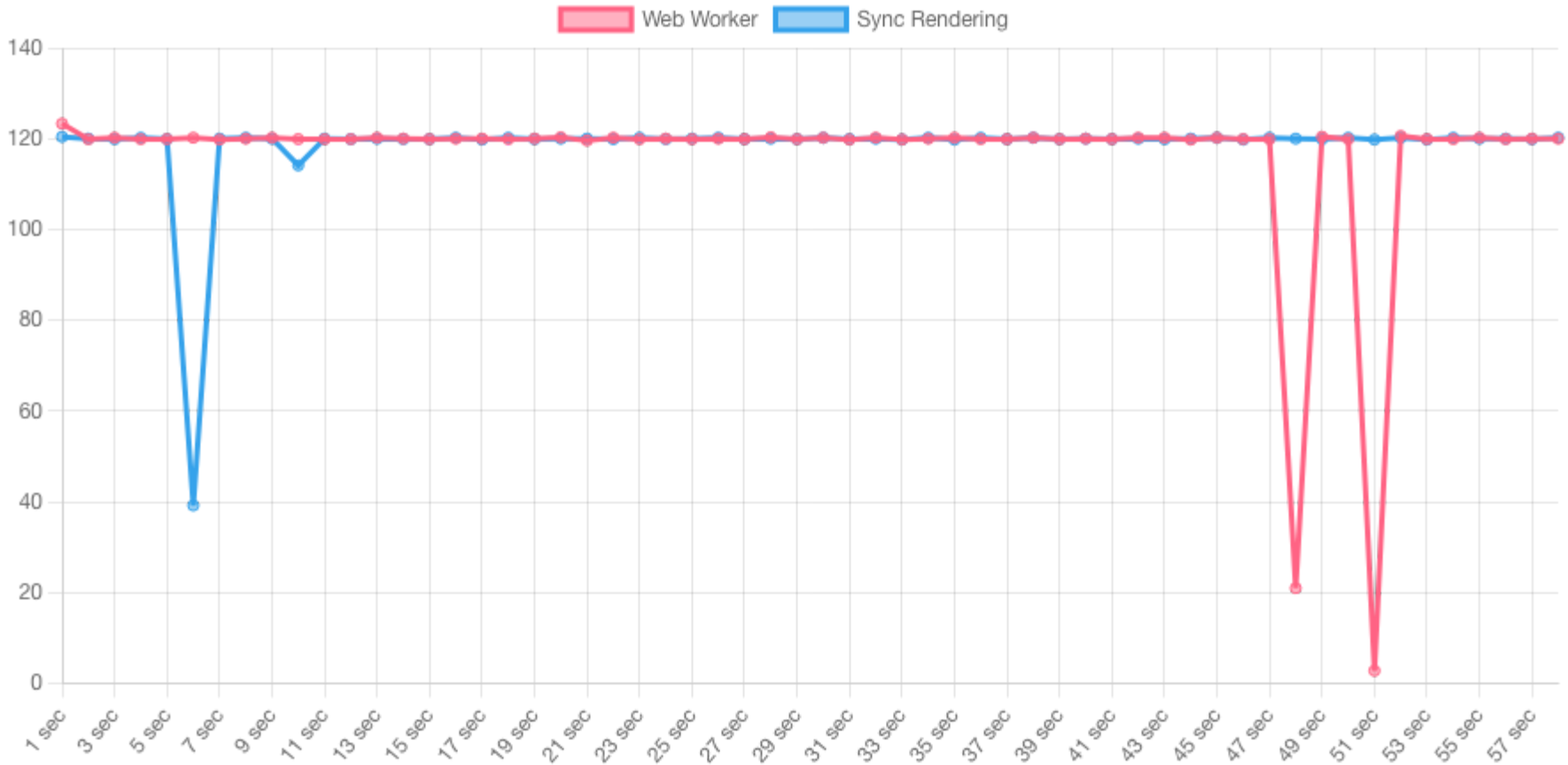
Debuted in a session at ReactConf 2015 and has been used to benchmark many javascript frameworks. It shows an application simulating DB queries.
To see frame rates, open Chrome dev tools, enable *Show fps meter* in Console >

Todo Sample

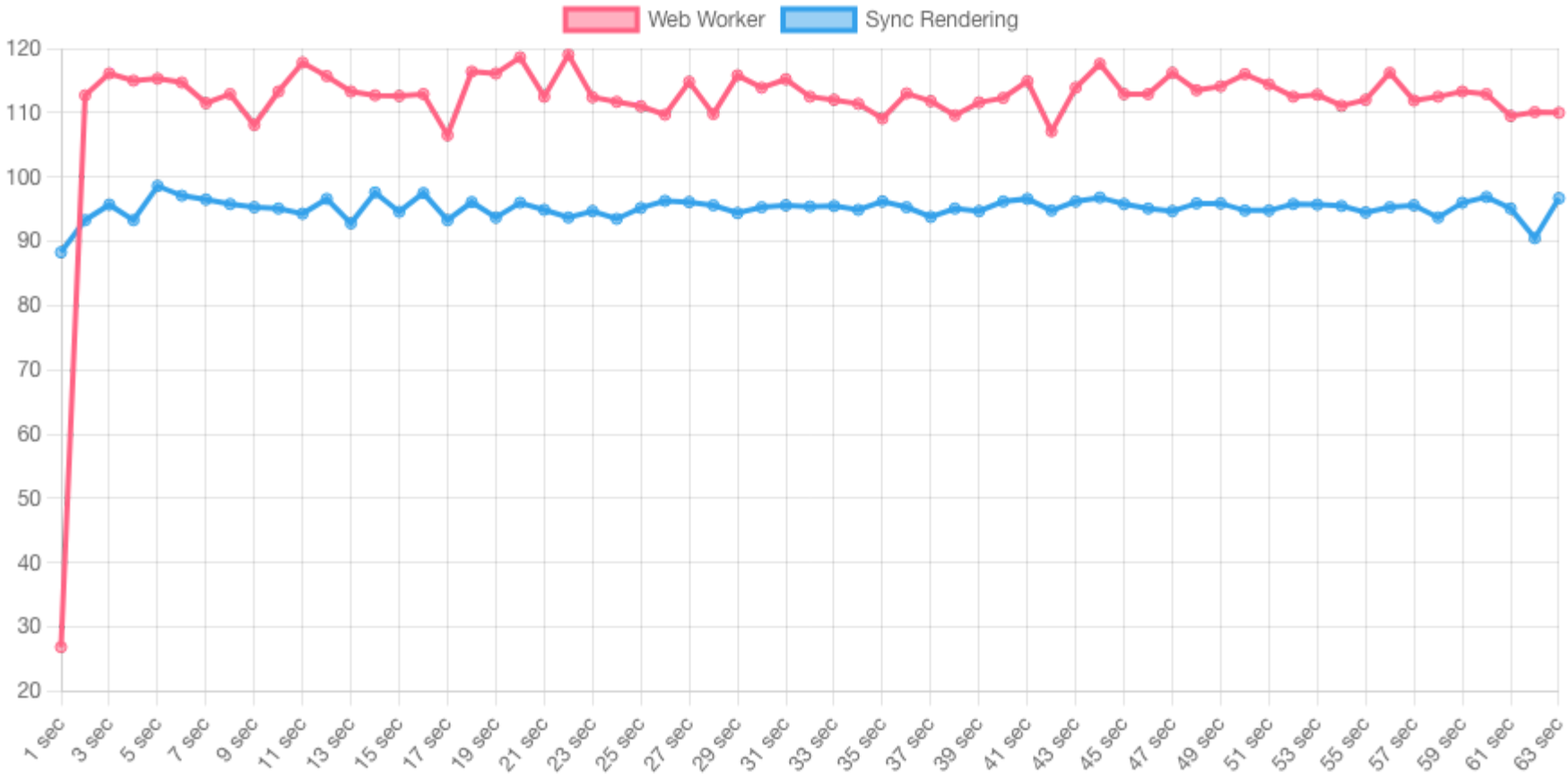
The canonical app that every frontend framework uses as a demo. Shows how events are passed from the UI thread to the worker, and how DOM manipulations as sent back to the UI thread from the worker thread.

cluster1slave	5	12.97	11.50	11.10	5.98	4.69
cluster2	4	11.83	9.29	5.62	3.99	-
cluster2slave	1	1.45	-	-	-	-
cluster3	2	12.78	4.07	-	-	-
cluster3slave	8	14.55	9.87	9.73	9.43	9.11
cluster4	7	12.02	11.69	9.20	7.31	5.01
cluster4slave	2	8.29	7.48	-	-	-
cluster5	1	10.16	-	-	-	-
cluster5slave	3	10.65	8.33	4.06	-	-
cluster6	6	14.32	10.28	8.30	5.98	4.68
cluster6slave	8	11.80	9.99	8.30	8.02	5.24
cluster7	6	12.69	12.06	9.13	7.58	6.83
cluster7slave	8	14.56	11.95	10.37	8.56	6.37
cluster8	3	12.89	7.52	1.44	-	-
cluster8slave	5	10.67	6.40	4.50	3.96	0.30

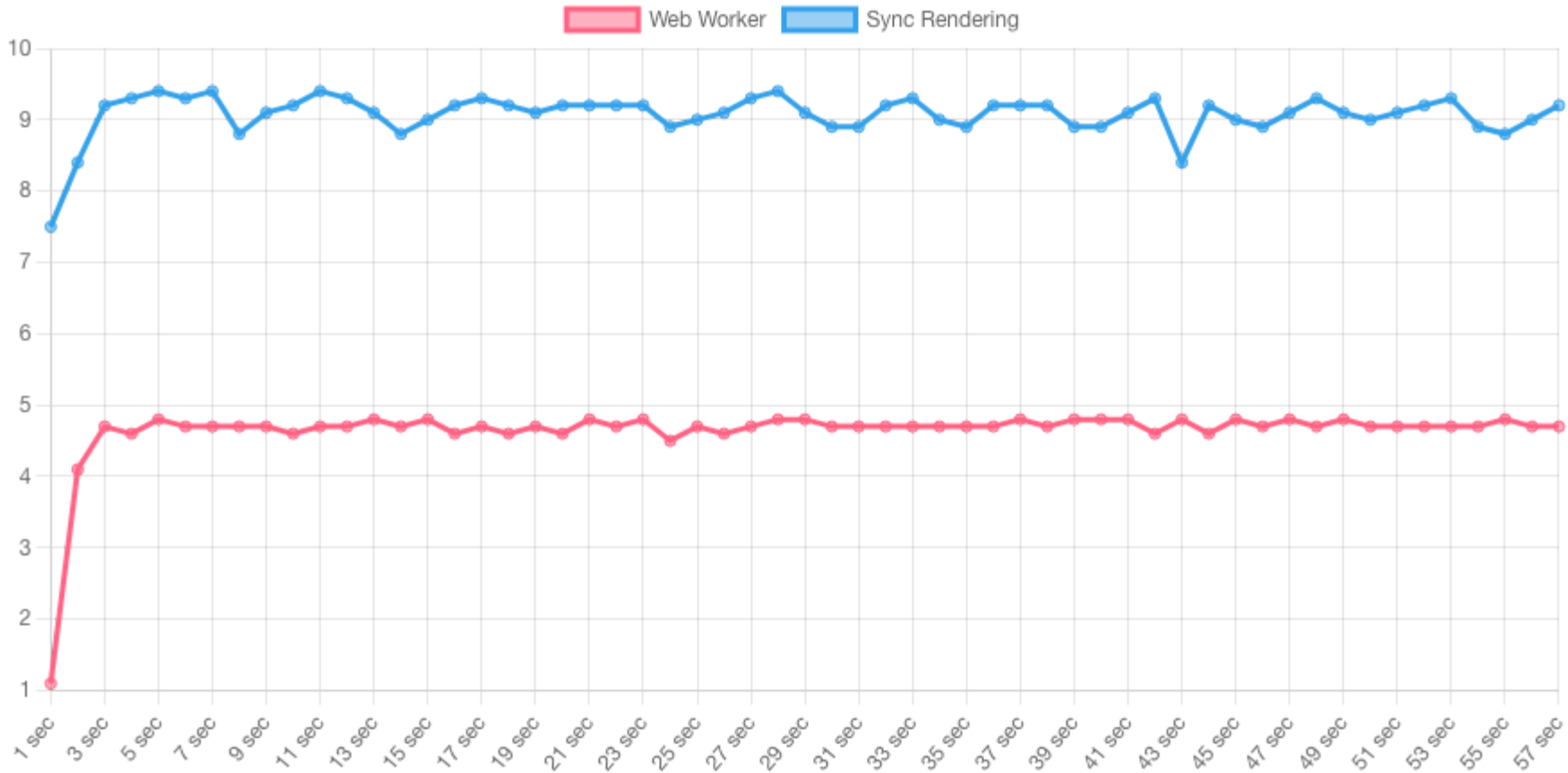
fps при 10 элементах



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

```
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

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.

А в чем разница ?

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

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

“Concurrent mode” -> Concurrent features

Transition features

```
import {
  startTransition,
    useTransition,
} from 'react';

const Component = () => {

  const [state, setState] = useState()

  const [isPending, startTransition] = useTransition();

  const onClick = (newState) => {

    startTransition(() => {

      setState(newState);
    });
  }
}
```

```
import {
  startTransition,
    useTransition,
} from 'react';

const Component = () => {

  const [state, setState] = useState()

  const [isPending, startTransition] = useTransition();

  const onClick = (newState) => {

    startTransition(() => {

      setState(newState);
    });
  }
}
```

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

Transitions are interruption/interruptible*

```
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

useDeferredValue()

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

А что у других ?

Сигналы

Сигналы – это реактивные примитивы для управления состоянием приложения.

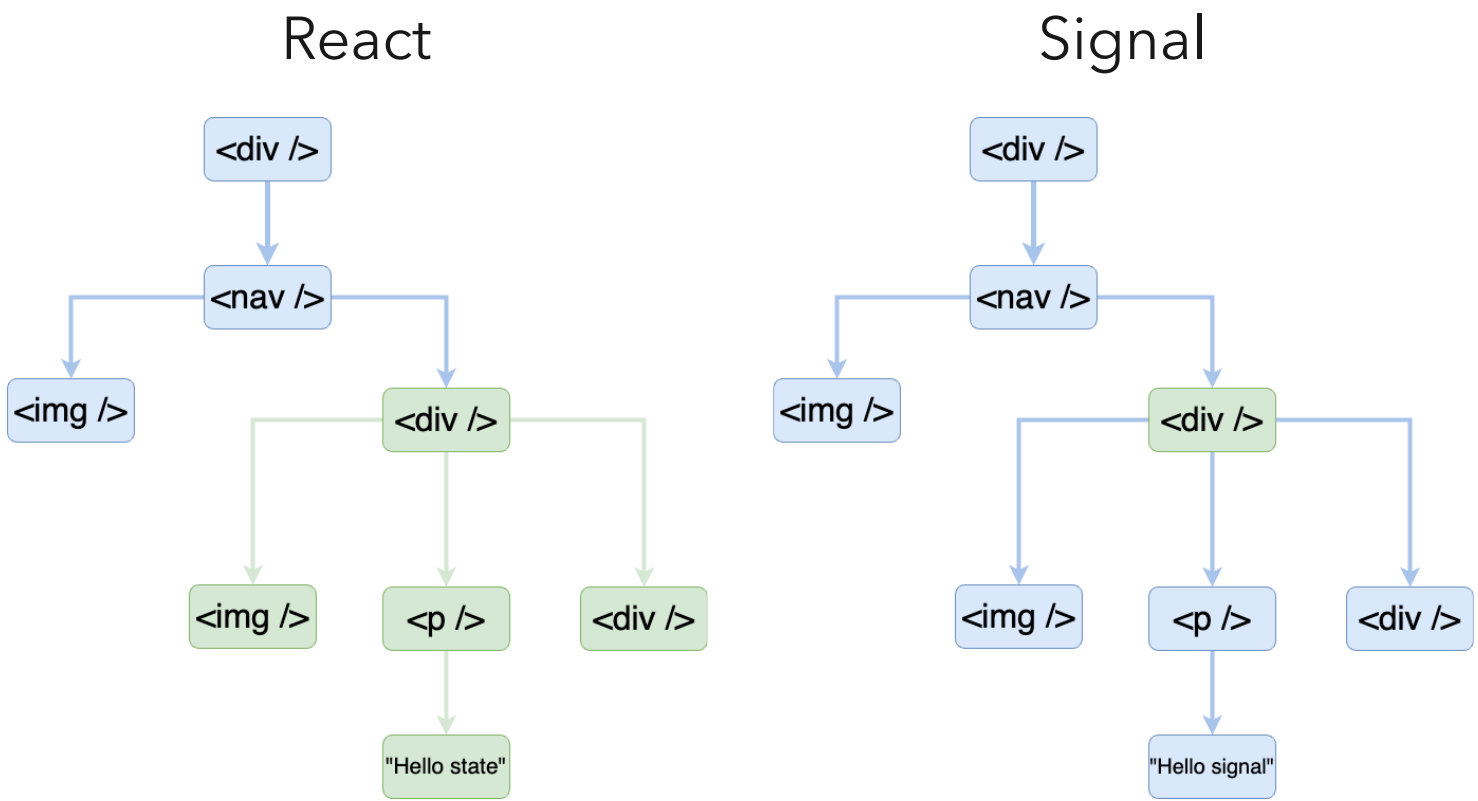
```
// react
const [state, setState] = useState(0);
// state -> value
// setState -> setter

// qwik
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 />

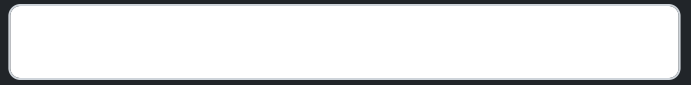
"Hello signal"

Кто использует ?

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

Blocked RxJS Solution

Hello from Angular (blocked thread example)



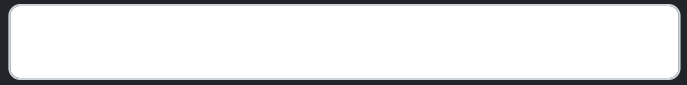
```
export class BlockedComponent implements OnInit {
  public dataControl = new FormControl('');

  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

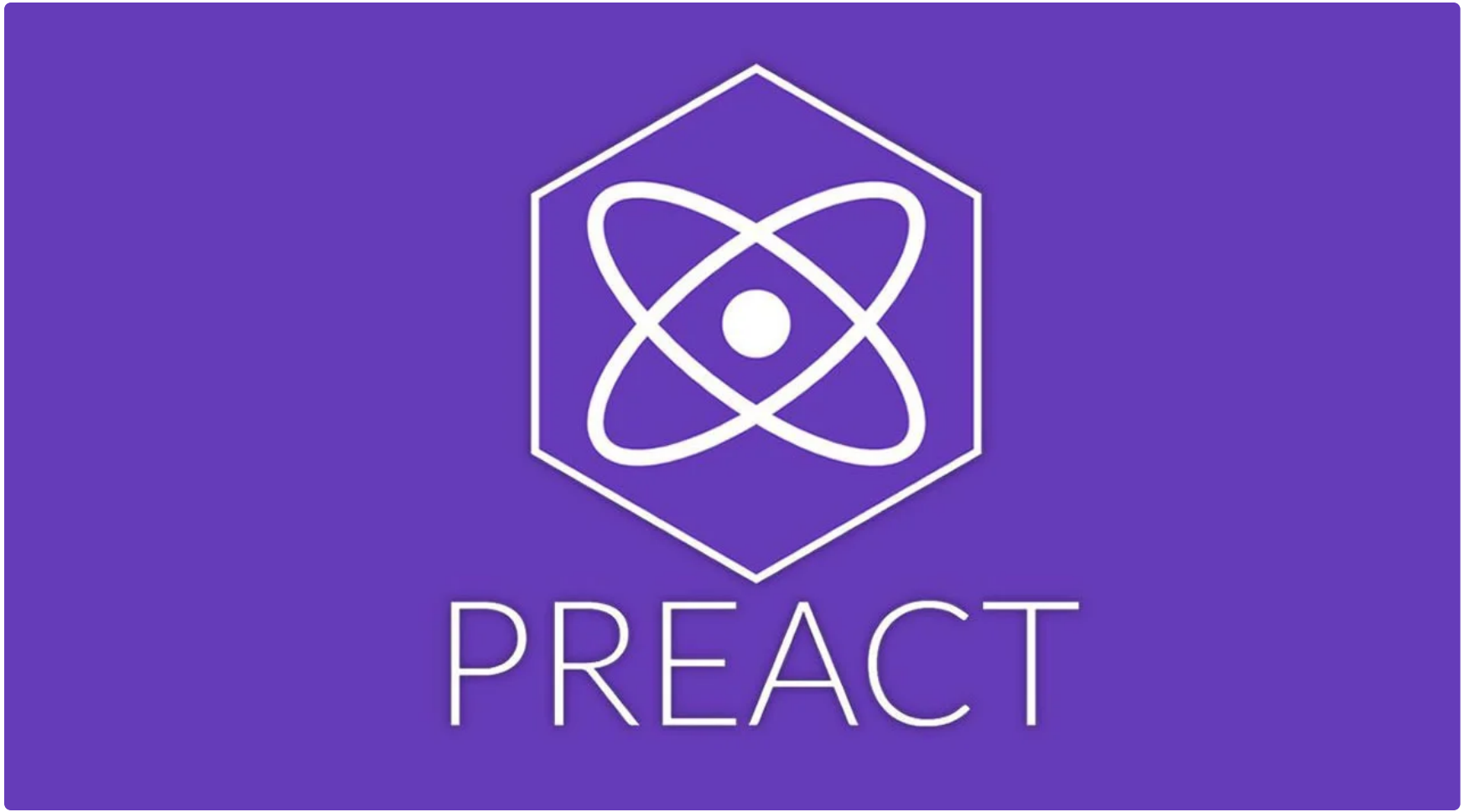
16 87 10 60 38 19 6 90 71 74 60 24 18 72 94

```
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 !