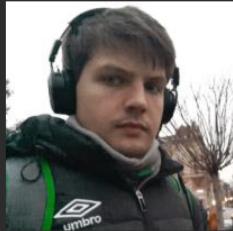




Нужен ли вам React SSR ?



Dmitry Poddubniy
Fullstack developer, Antarasoft

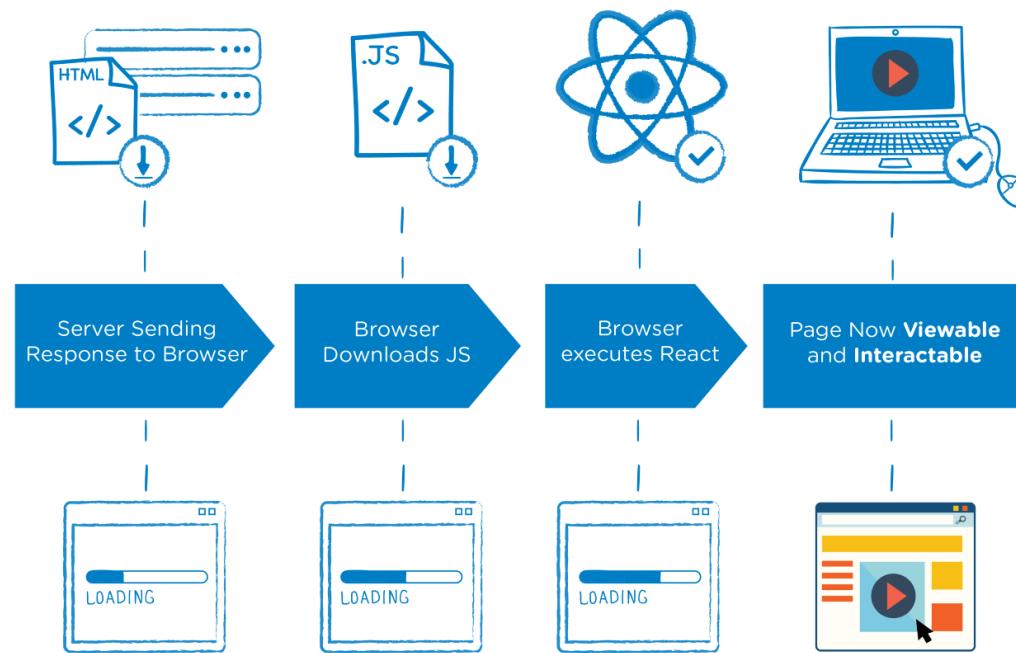
Dmitry Poddubniy a.k.a
@mr47



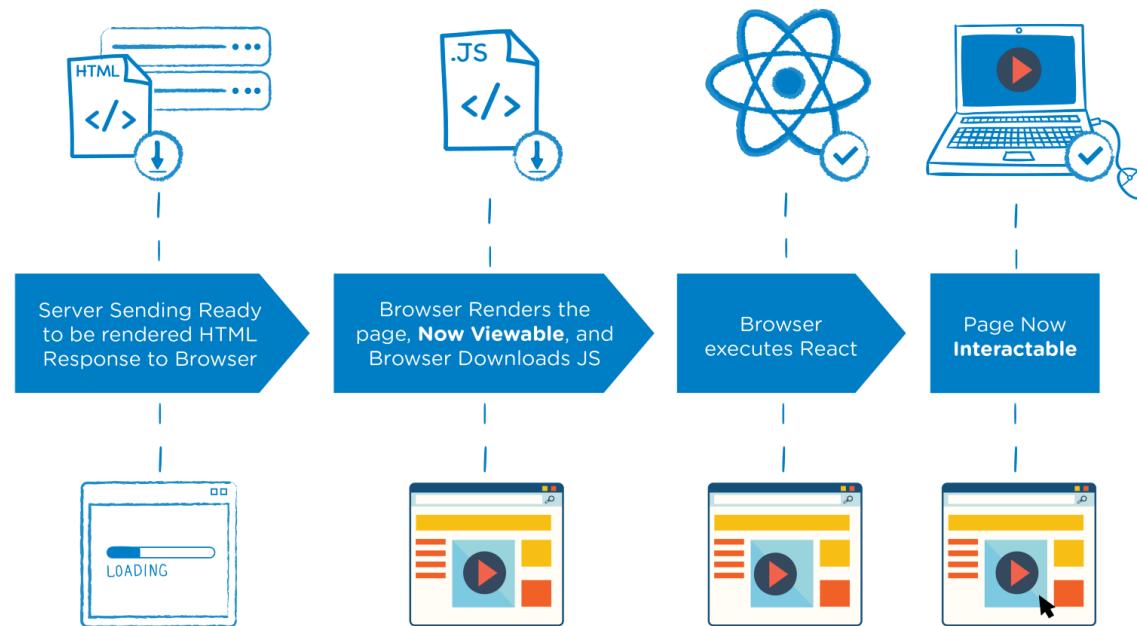
Когда нужно использовать SSR

- SEM, SEO - Baidu, Bing, Yahoo
- Хотим наилучшую производительность
- Убрать "лаг" на начальный рендеринг
- Письма / PDF
- Ваше приложение работает

CSR



SSR



Как же выглядит SSR на самом деле.

Как работает React v15.x SSR, сервер

```
// using Express
import { renderToString } from "react-dom/server"

import MyPage from "./MyPage"

app.get("/", (req, res) => {
  res.write("<!DOCTYPE html><html><body>");
  res.write("<div id='content'>");
  // render app
  res.write( renderToString(<MyPage/>) );
  // finish html
  res.write("</div></body></html>");
  res.end();
});
```

На клиенте

```
import { render } from "react-dom"
import MyPage from "./MyPage"

render(<MyPage/>, document.getElementById("content"));
```

Что на выходе у ReactDOM.renderToString

```
<!-- code -->
<div data-reactroot data-reactid="1" data-react-checksum="-111">
  <div data-reactid="2"></div>
</div>
<!-- code -->
```

Легко описать SSR так :



Какие плюсы и минусы у SSR:

- + Universal / Isomorphic app
- Sync
- Rerender always
- CPU waste
- Memory waste

Но это было в 15 версии в 16й нас ждет...

Render -> Hydrate

```
import { hydrate } from "react-dom"
import MyPage from "./MyPage"

hydrate(<MyPage/>, document.getElementById("content"));
```

Arrays, Strings, Numbers

```
class MyArrayComponent extends React.Component {
  render() {
    return [
      <div key="1">first element</div>,
      <div key="2">second element</div>
    ];
  }
}

class MyStringComponent extends React.Component {
  render() {
    return "hey there";
  }
}

class MyNumberComponent extends React.Component {
  render() {
    return 2;
  }
}
```

Для SSR в React 16.0.0:

- Обратная совместимость
- Render -> Hydrate
- Менее строгое сравнение содержимого
- componentDidCatch не работает
- Порталы тоже не работают им нужен DOM
- Stream API
- Fiber
- Оптимизированы warnings с
`process.env.NODE_ENV !== "production"`

О чём мало кто говорит:

- ComponentDidMount
- Компоненты не поддерживающие SSR
- Canvas / WebGL
- Двойной рендеринг (Тру маунтинг близко)
- Race conditions
- Передача начального состояния на клиент
- Позиционирование / масштабирование элементов
- Начальная анимация
- Fiber + setState transaction

setState:

- setState выполняется не моментально
- setState всегда вызывает rerender
- Fiber: pause
- Fiber: split
- Fiber: rebase
- Fiber: abort

Какие частые проблемы можно встретить.

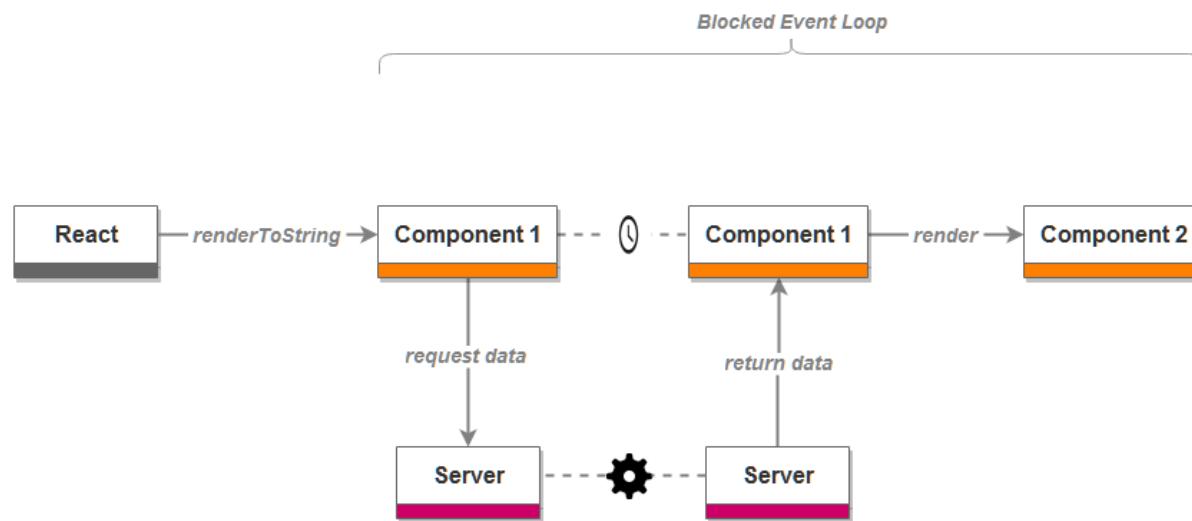
Warning: React
attempted to reuse
markup in a container
but the checksum was
invalid.

- «[console.log](#)»

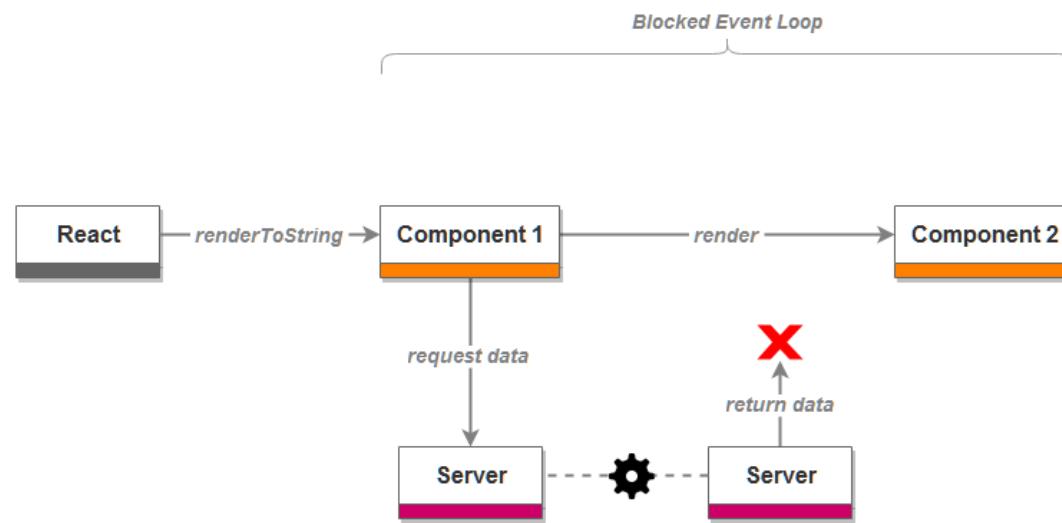
Пример проблемы

```
class Slider extends React.PureComponent{
    constructor(props, context){
        super(props, context);
        this.state = { animated: props.animated || false };
    }
    componentWillMount(){
        setTimeout(
            () => this.setState({ animated: true }),
            400)
    }
    render(){
        const { animated } = this.state;
        const c = animated ? 'my-div--animated' : '';
        return(
            <div className={`my-div ${c}`}>
                {/* ... Some code ...*/}
            </div>
        );
    }
}
```

Promise rejection



Promise rejection

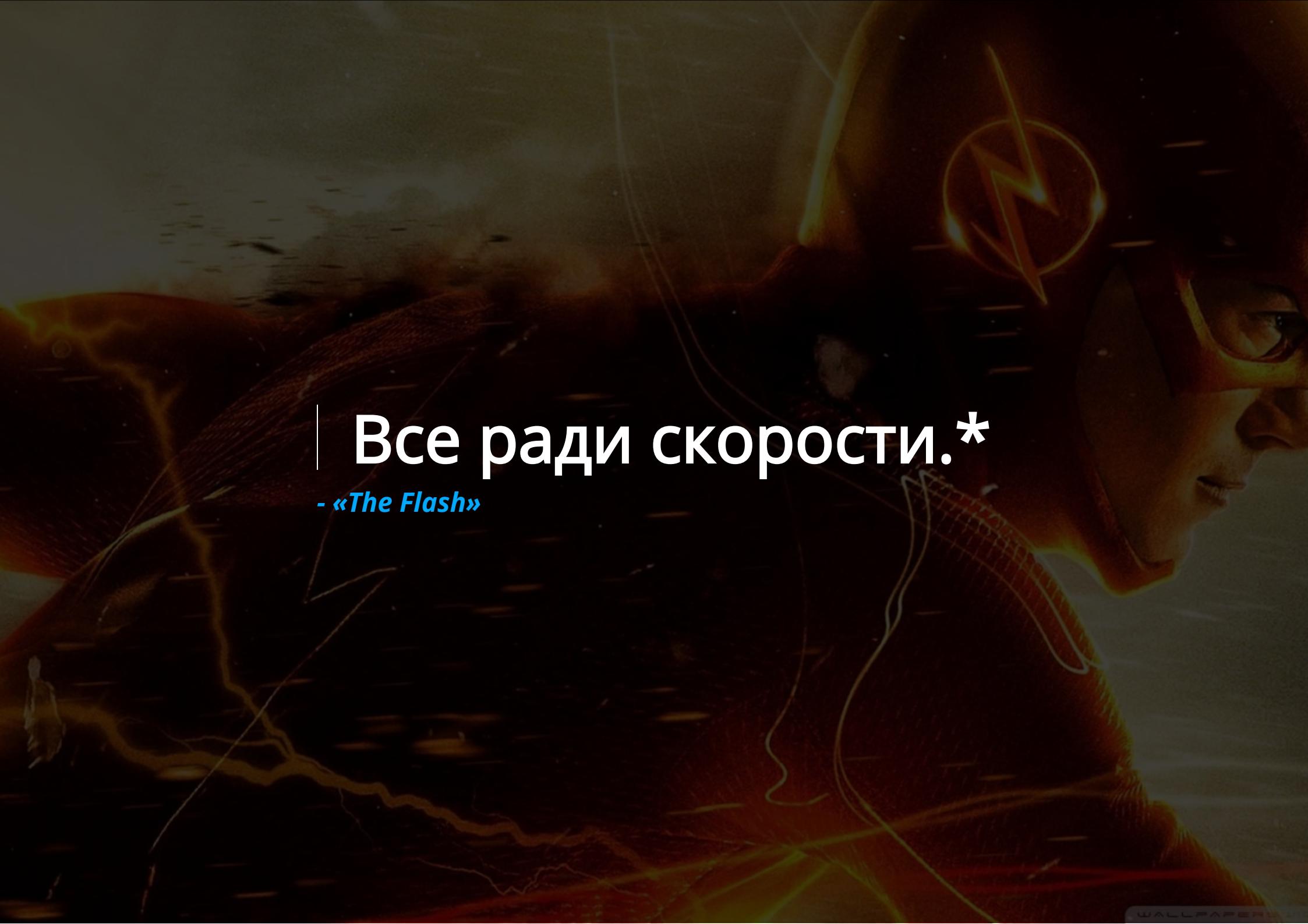


Если вы все еще хотите SSR тогда,



Что же нам поможет:

- redux
- react-router
- no-ssr component
- proxy for ssr components
- react-jobs
- react-server
- react-ssr-optimization
- less components do more*

A dynamic, low-angle shot of The Flash running at high speed. His body is blurred horizontally, suggesting rapid movement. In the upper right corner, his iconic lightning bolt logo is prominently displayed within a circular emblem. The background is dark and filled with streaks of light and energy.

Все ради скорости.*

- «*The Flash*»

Почему Redux:

- Pure js (для нас SSR из коробки)
- Не используются Observers
- Reducers синхронные
- Состояние стора легко передать на клиент
- Восстановление состояния крайне просто
- Лучшая интеграция с react-router
- Отсутствуют сайд эффекты

Решение для data fetch (promises hell) - react-jobs

```
export default withJob({
  work: (props) => fetch(`/cat/${props.catId}`).then(r => r.json()),
  LoadingComponent: (props) => <div>Fetching cat...</div>,
  ErrorComponent: ({ error }) => <div>Opps! {error.message}</div>
})(Cat)
```

React-server:

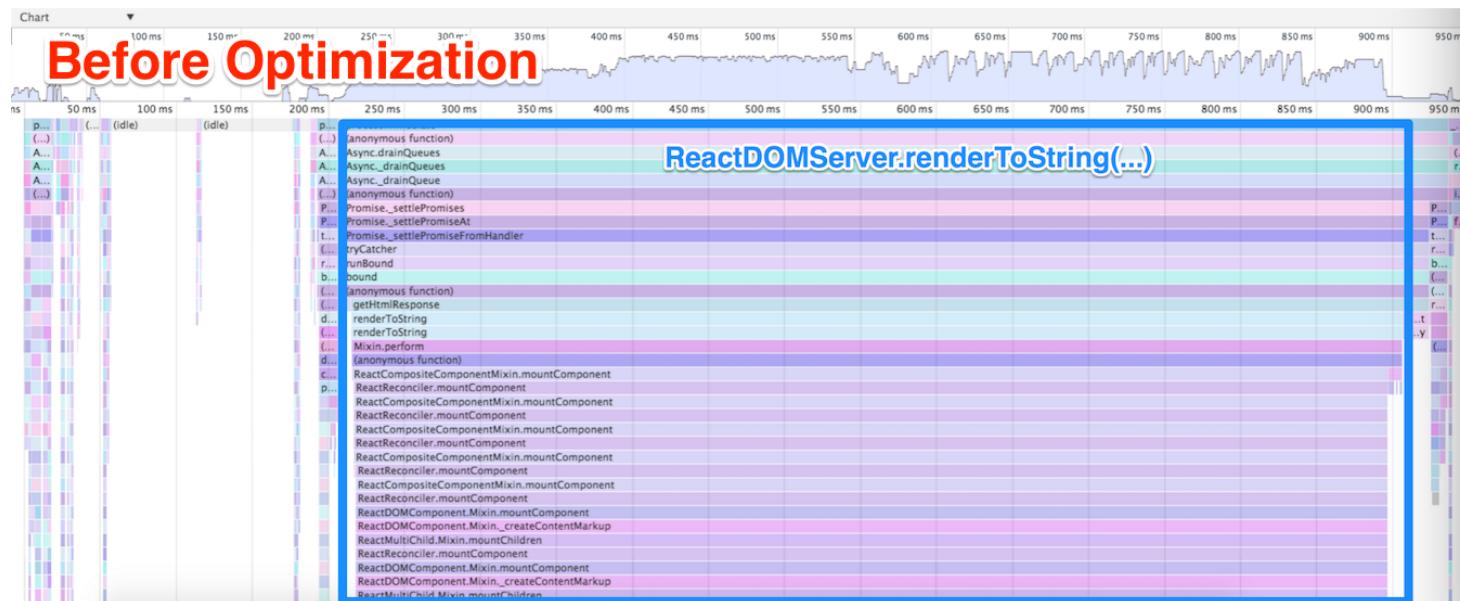
- Параллельные запросы к серверу
- Сжатие данных для клиента
- Потоковая передача HTML
- Маунтинг
- Page-to-page переходы
- Рендеринг быстрый даже если backend медленный

React-ssr-optimization

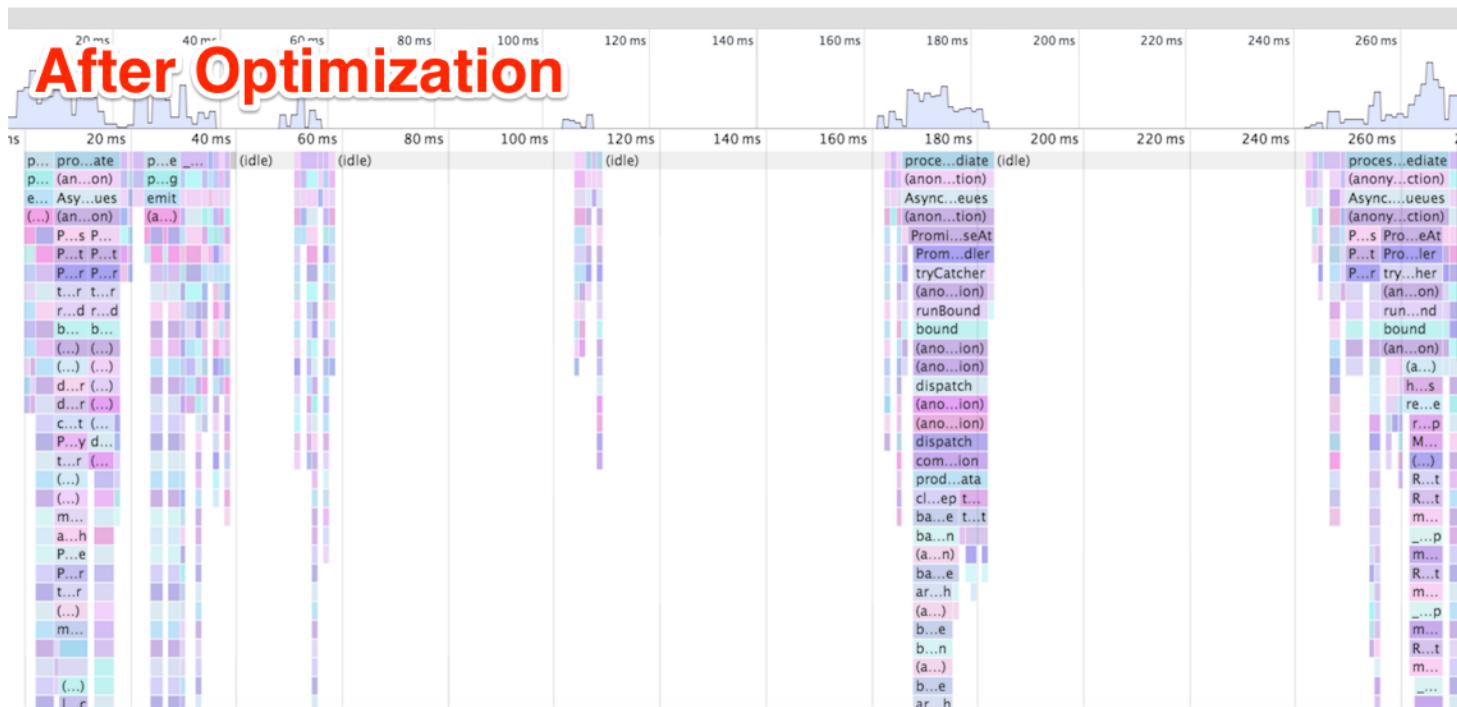
```
const component = (props) => (
  <div className="product">
    <p className="description">{props.product.description}</p>
    <p className="price">Цена: ${props.selected.price}</p>
    <button
      disabled={props.inventory > 0 ? '' : 'disabled'}
    >
      {props.inventory ? 'Купить' : 'Нет в наличии'}
    </button>
  </div>
);
```

```
const component = () => (
  <div className="product">
    <p className="description">${product_description}</p>
    <p className="price">Цена: ${selected_price}</p>
    <button
      disabled={this.props.inventory > 0 ? '' : 'disabled'}
    >
      {this.props.inventory ? 'Купить' : 'Нет в наличии'}
    </button>
  </div>
);
```

До оптимизации



После оптимизации





Альтернативы:

- CSR (client side rendering)
- Next.js (static generator)
- react-snapshot (static pre-renderer)
- no-ssr component (частичное использование)

React's server-side rendering can become a performance bottleneck for pages requiring many virtual DOM nodes.

- *Walmart Labs*

УВИДИМСЯ В БУДУЩЕМ!



Ссылки:

- mr47.in
- github.com/mr47/kharkivjs-2017
- react-server
- no-ssr component
- react-jobs
- react-ssr-optimization
- Fiber

