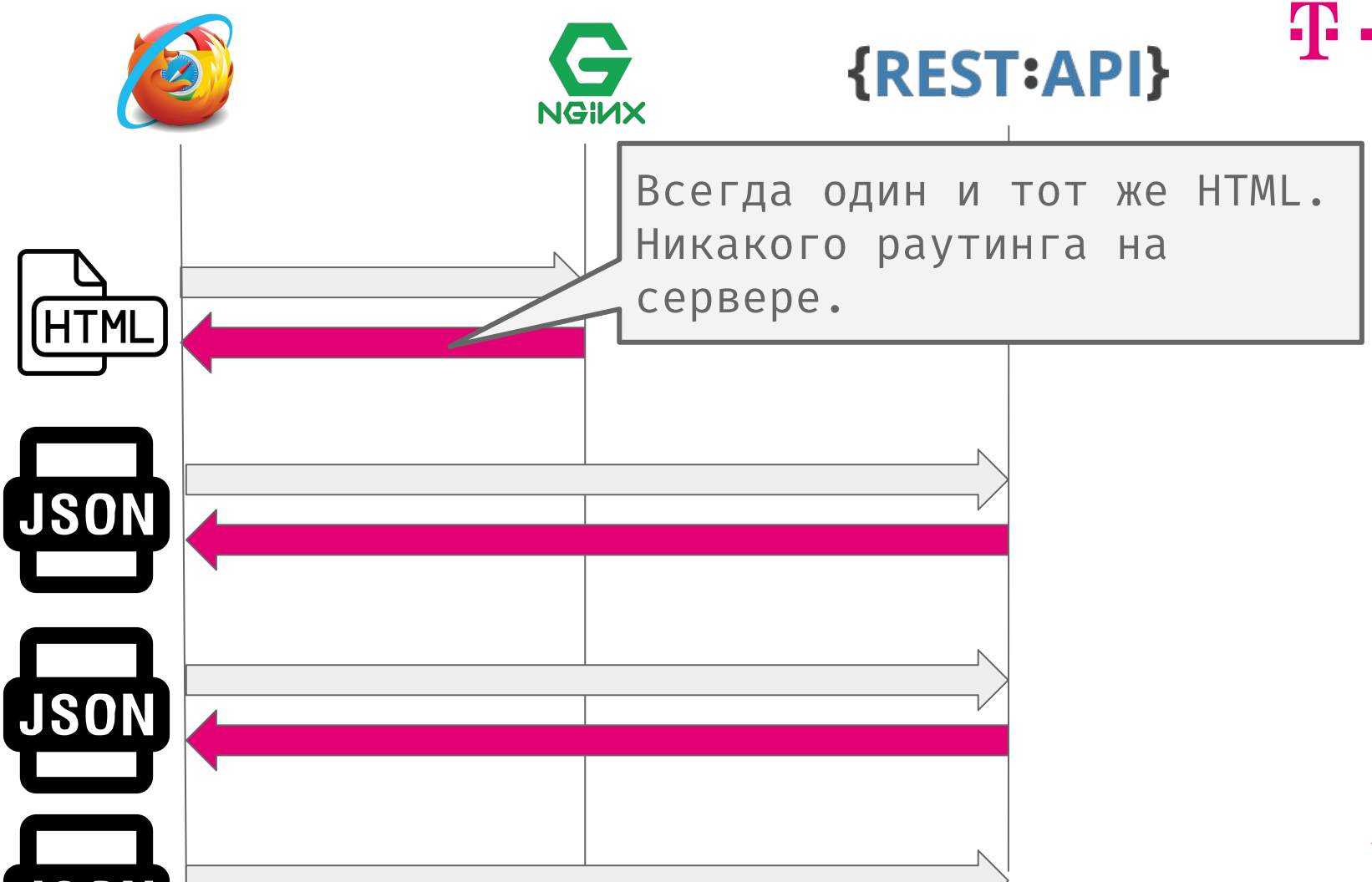
React + SSR
там, где их уже не ждали

Если для вашего случая туториалов не завезли

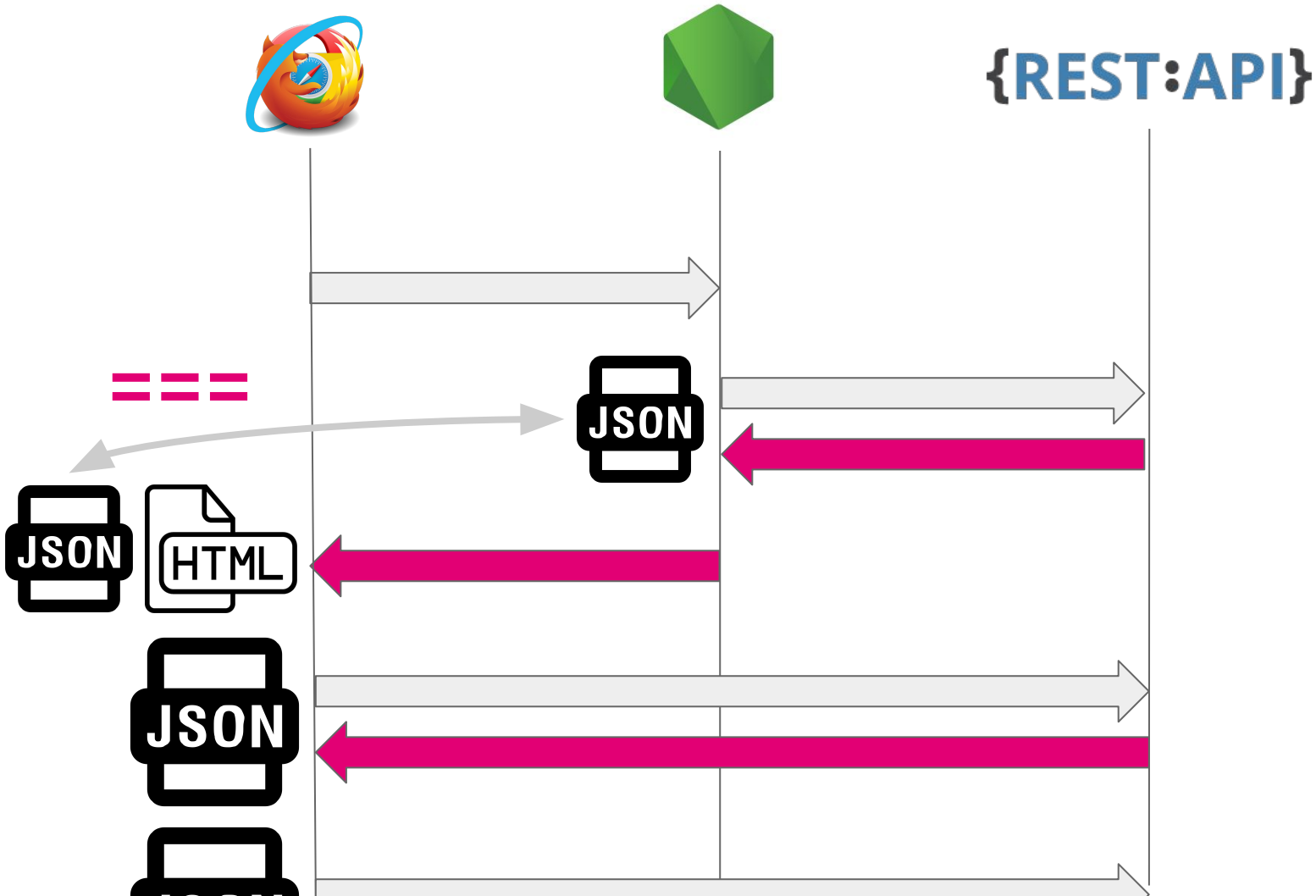
Кейс про использование React + SSR в необычных условиях.


NGINX
{REST:API}
Всегда один и тот же HTML.
Никакого раутинга на
сервере.
HTML
JSON
JSON

Всегда тот же самый компонент без параметров.

```
<script>
   let component = React.createElement(App);
   React.render(
      component,
      document.getElementById('root')
   );
</script>
```

Всегда один тот же самый DOM элемент


{REST:API}
===
JSON
JSON
HTML
JSON

```
server.get('*', async (req, resp) => {
   let props = await getProps(req);
   res.send(
      wrapIntoPageHtml(
         React.renderToString(
            React.createElement(App, props)
         )
      )
   );
});
```

Вы вернули тот джейсон
с данными ранее с сервера

```
<script>
   let comp = React.createElement(
      App, getPropsFromJson()
   );
   React.hydrate(
      comp, document.getElementById('root')
   );
</script>
```

Такой подход предполагает, что

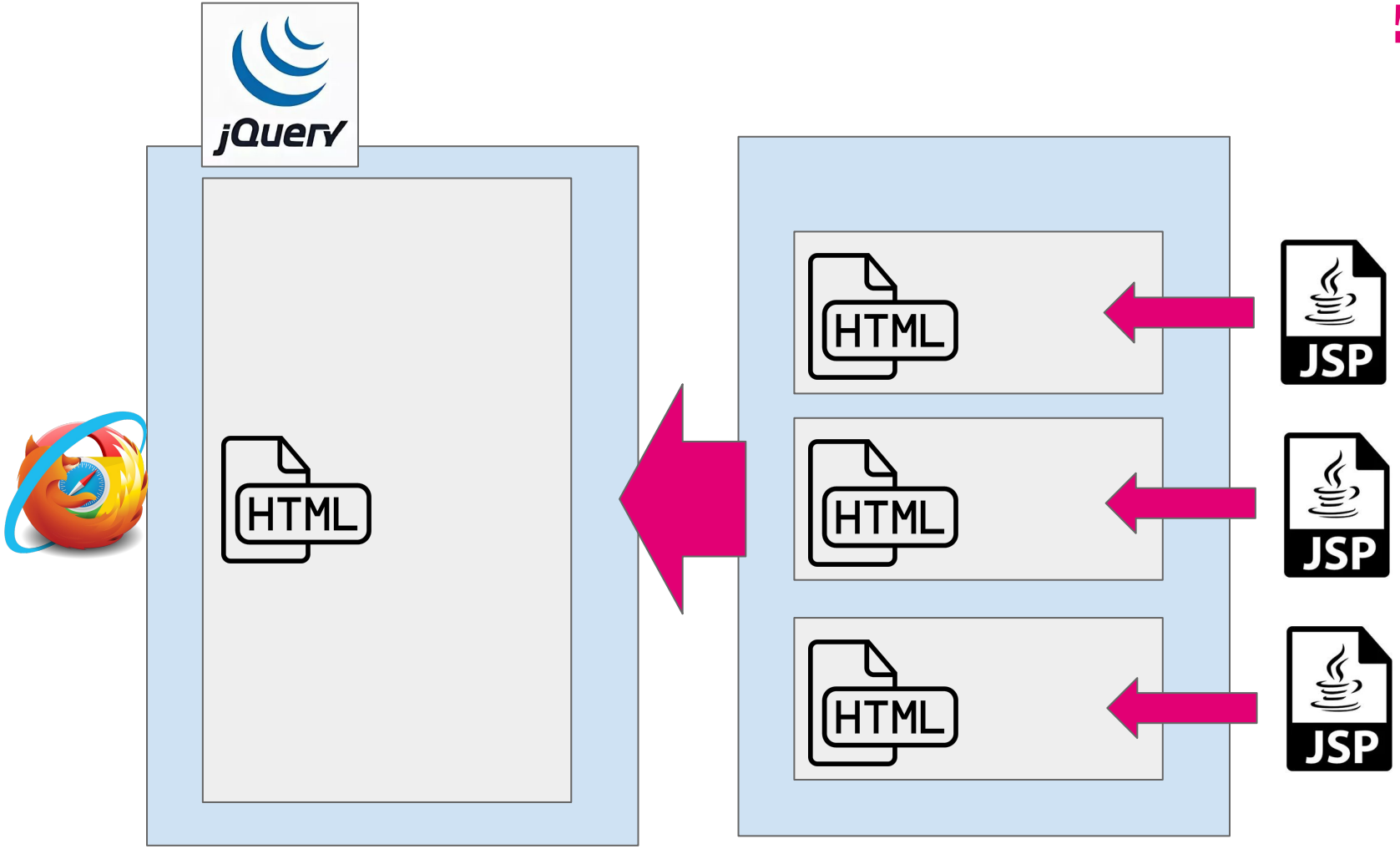
I. За сборку страницы отвечает JS приложение

II. Запрос пользователя принимает Node.js сервер

А у нас

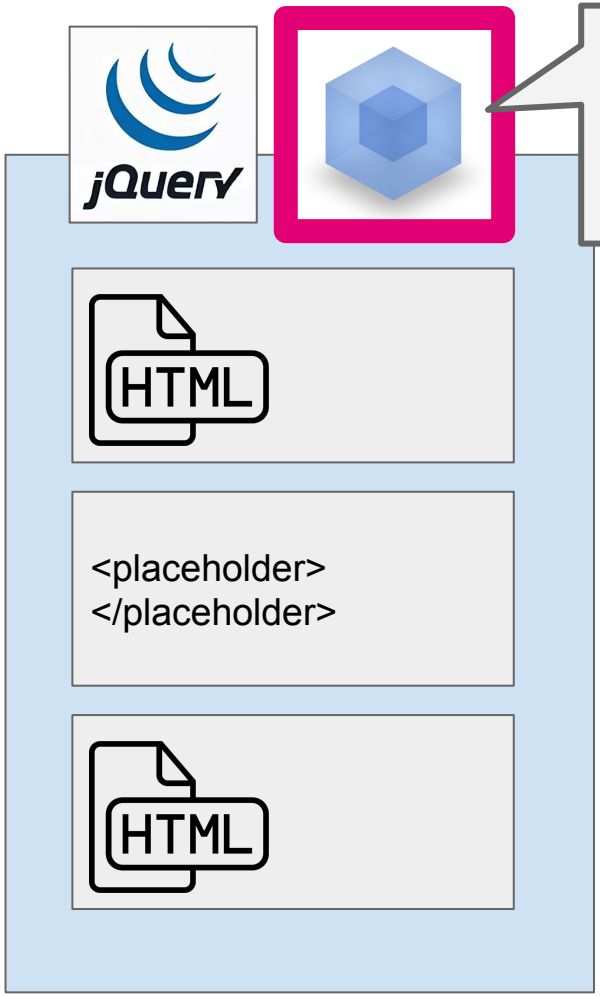
I. За сборку страницы отвечает Oracle ATG, эта **CMS** ничего не знает про JS.

II. Oracle ATG отвечает за обработку запросов пользователей.


jQuery
HTML
HTML
HTML
HTML
JSP
JSP
JSP

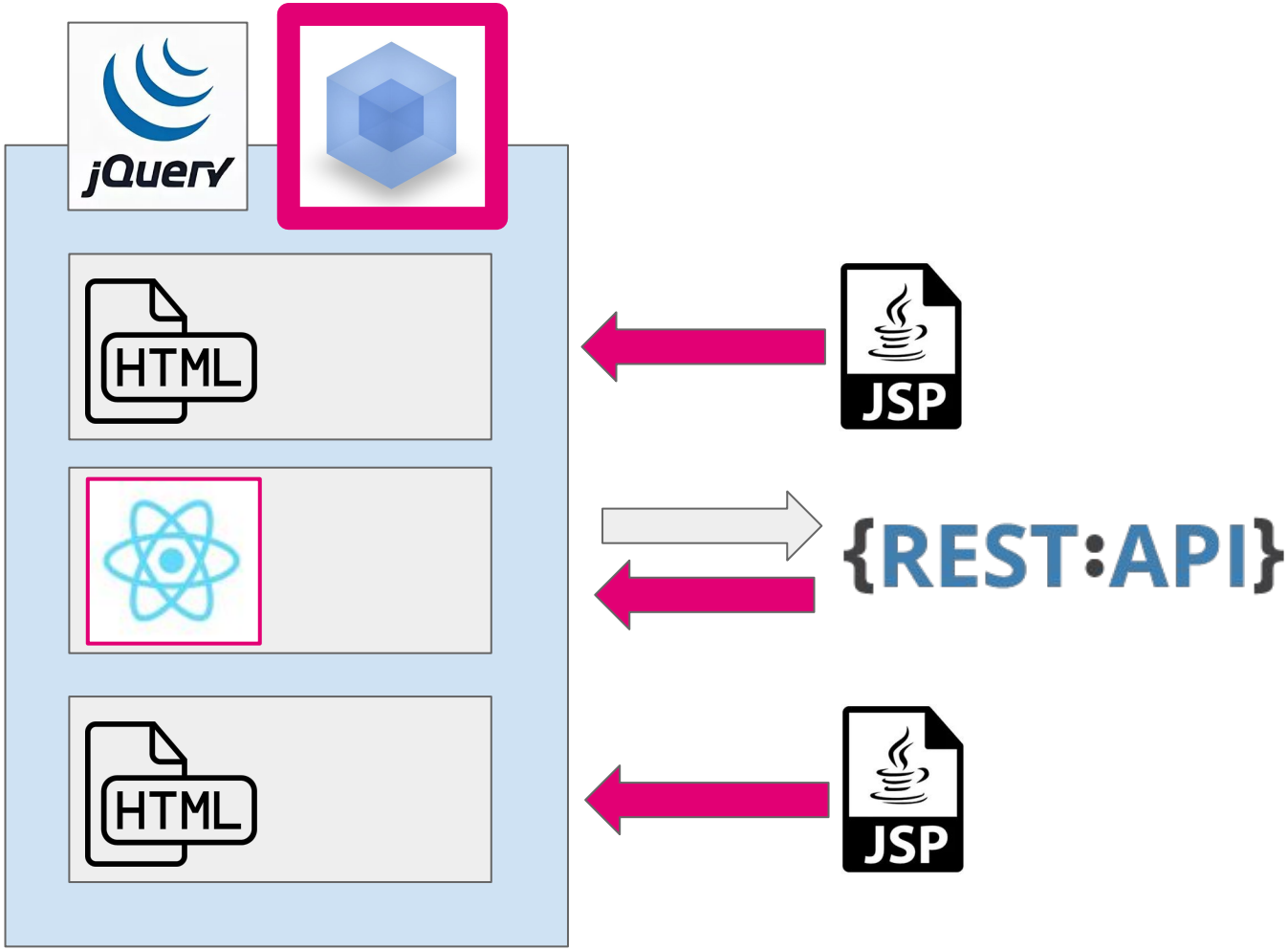
```
<div class="js-view">
    <div class="js-view__veil">
        <div class="spinner"></div>
    </div>
    <script type="application/json">
        {
          "name": "component-to-render-here",
          "props": {...}
        }
    </script>
</div>
```

Подобные блоки мы назвали плейсхолдерами



Мы добавили небольшой скрипт на каждую страницу, чтобы...

I. Найти плейсхолдеры.
II. Вытащить JSON, проверить корректность.
III. Загрузить зависимости.
IV. Загрузить компоненты.
V. Получить данные.
VI. Запустить рендеринг.


jQuery
HTML
HTML
JSP
JSP
{REST:API}

Данные должны быть запрошены до начала рендеринга.

Эту задачу еще до появления Node.js Она требует изменений в root компонентах.

```
{
    "name": "component-to-render-here",
    "props": {
        "key": "value"
    },
    "dataClient": {
        "endpoints": {
            "getData": "endpointURL",
            "submitData": "endpointURL"
        },
        "configuration": {
            "renderingData": [
                "getData"
            ]
        }
    }
}
```

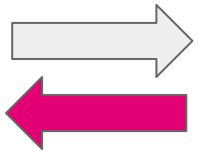
Перечень доступных эндпоинтов.

те из них, которые нужно вызвать до начала рендеринга

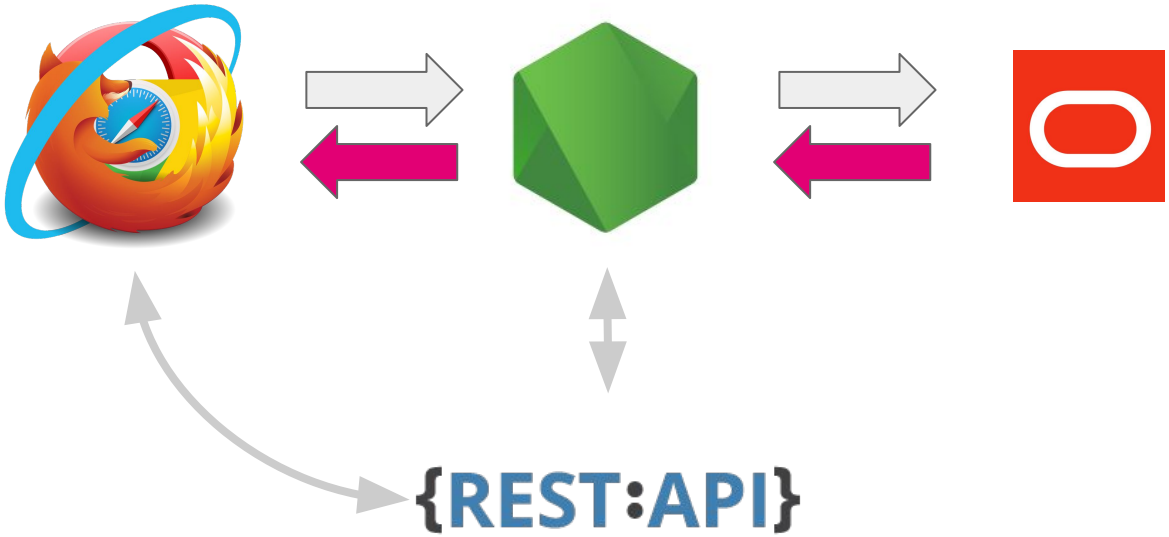
```
class OurComponent extends Component {
    static getRenderingData({dataClient}) {
        return new Promise(res => {
            dataClient.getData
                .get()
                .then(res);
        });
    }
    // ...
}
```

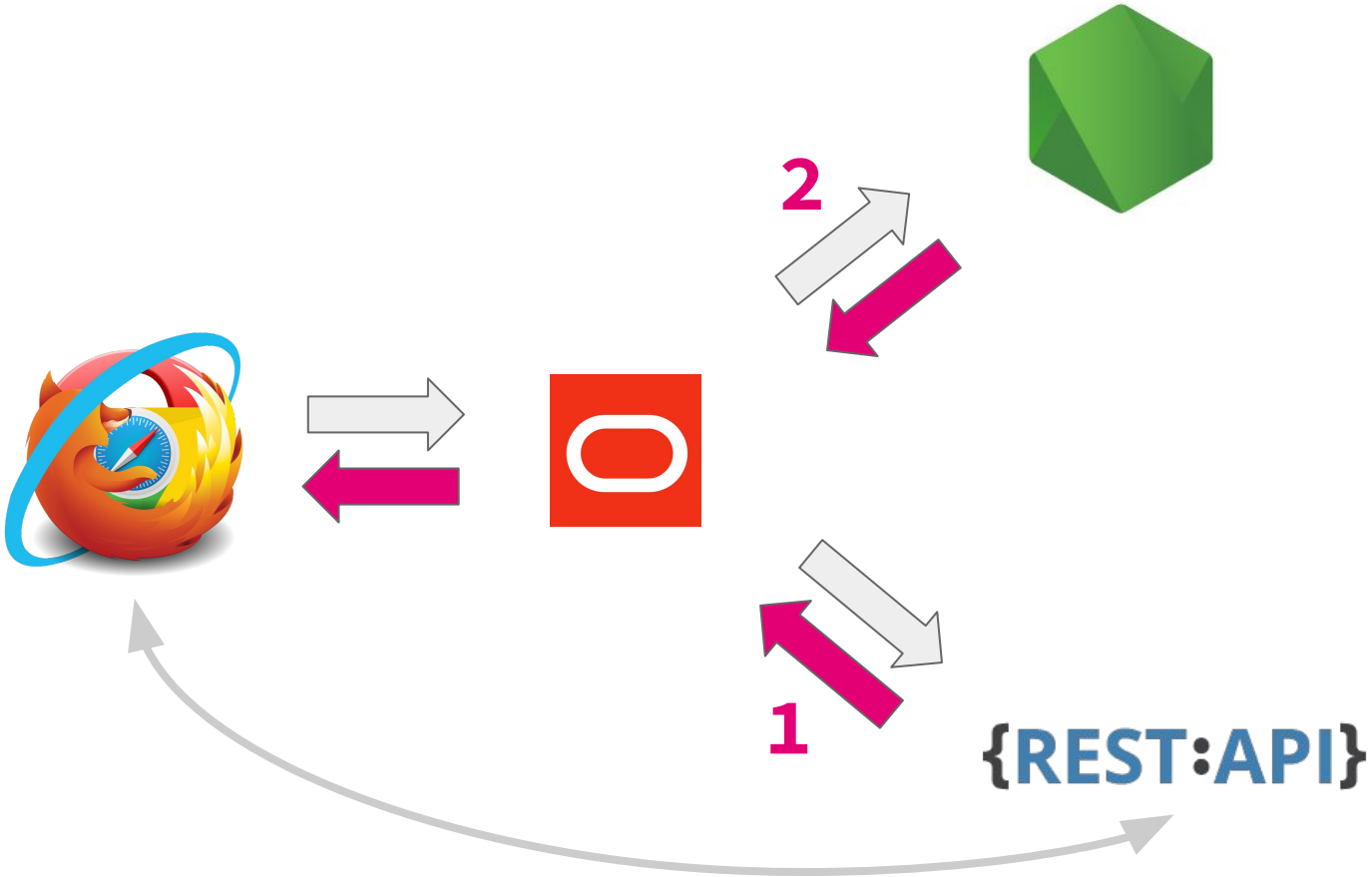
Только в качестве эксперимента можно без node.js

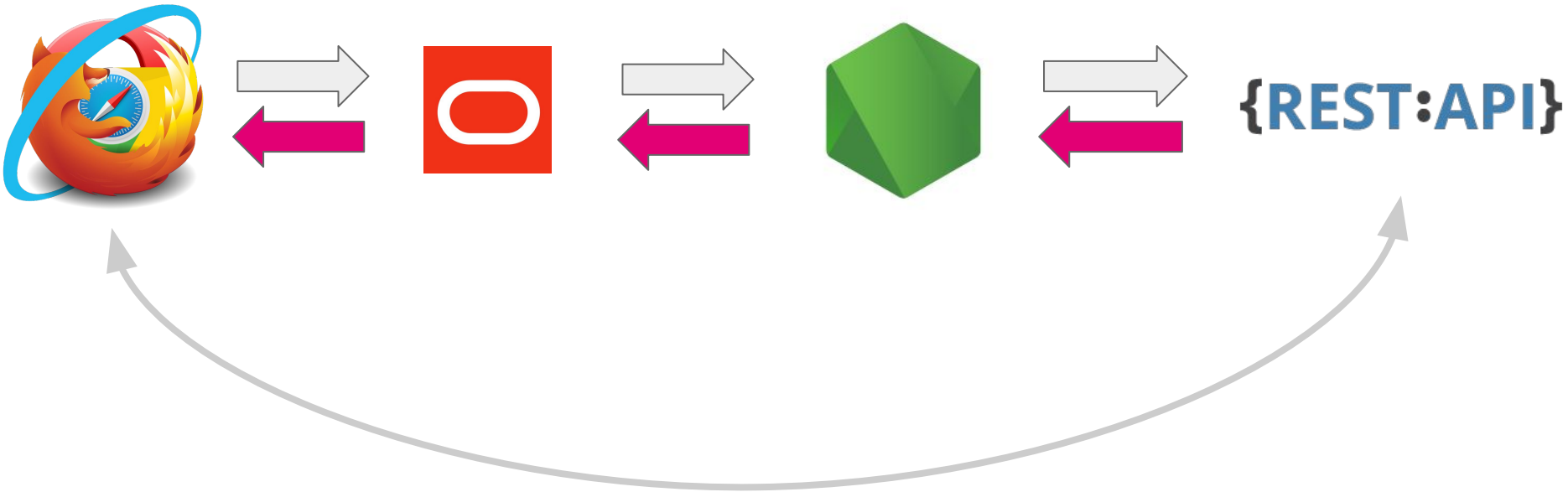
???



+ {REST:API}


{REST:API}


2
1
{REST:API}


{REST:API}

```
<div js="js-view">
    <div class="out-component-markup">
        <!-- all rendered html -->
    </div>
    <script type="application/json">
        {
            "name": "out-component",
            "props": {
                "title": "наш компонент"
                "renderingData": {
                    "//": "самая интересная часть"
                }

            }
        }
    </script>
</div>
```

В таком виде все уже почти работает
Остались только css-modules.

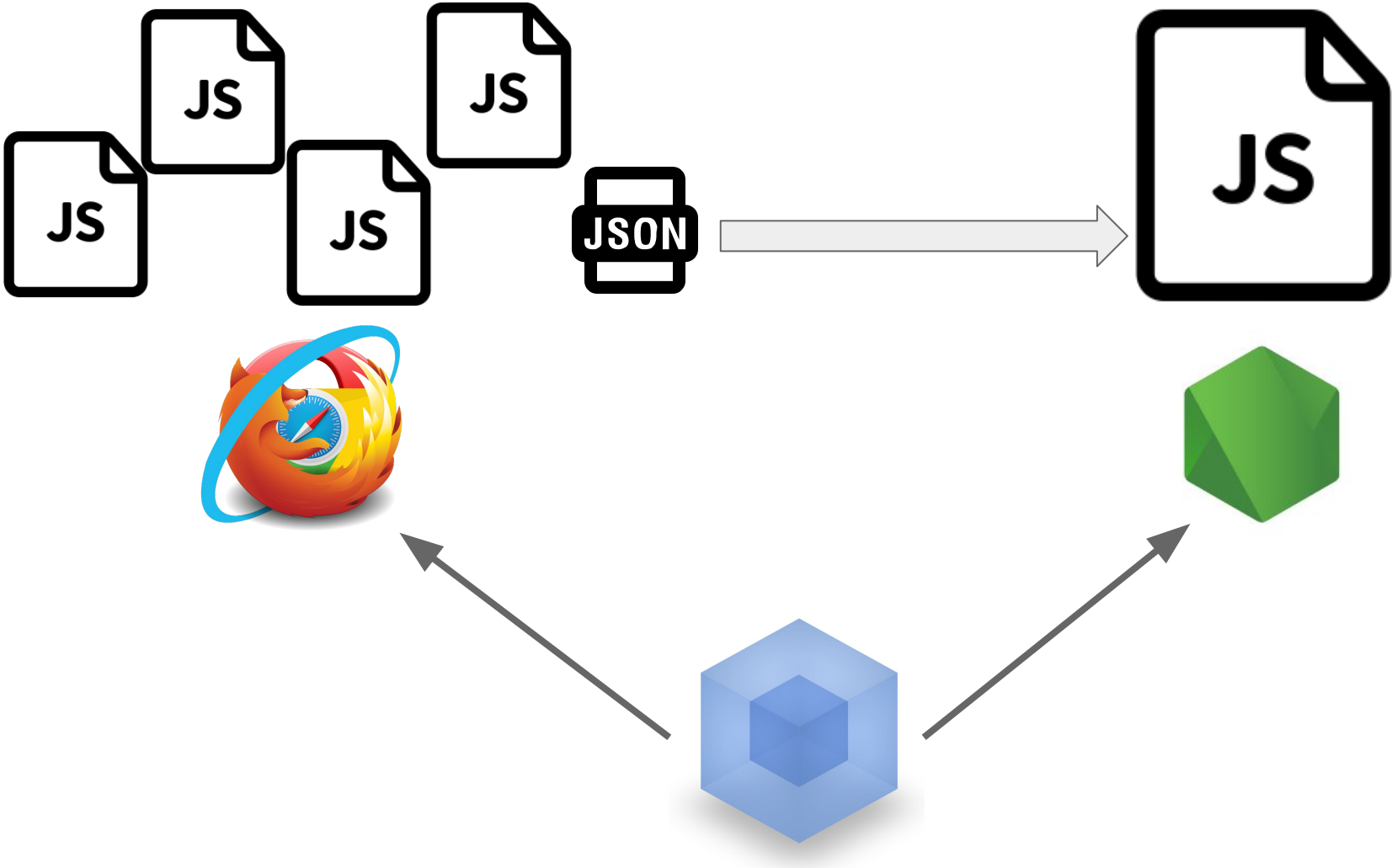
Webpack их подгружает на страницу вместе с кодом. Это слишком поздно.

Есть isomorphic-style-loader. Он позволяет решить проблему, **но**:

- требует изменения кода компонентов
- определяет нужные стили в рантайме

Имена этих файлов
знает только Webpack

```
<link rel="stylesheet" href="/styles/23.css">
<link rel="stylesheet" href="/styles/48.css">
<div js="js-view">
    <!-- наша сервер-сайд рендеренная разметка -->
</div>
```


JS
JS
JS
JS
JSON
JS

Спасибо! Вопросы?

Александр Зонов | @fort_wrong