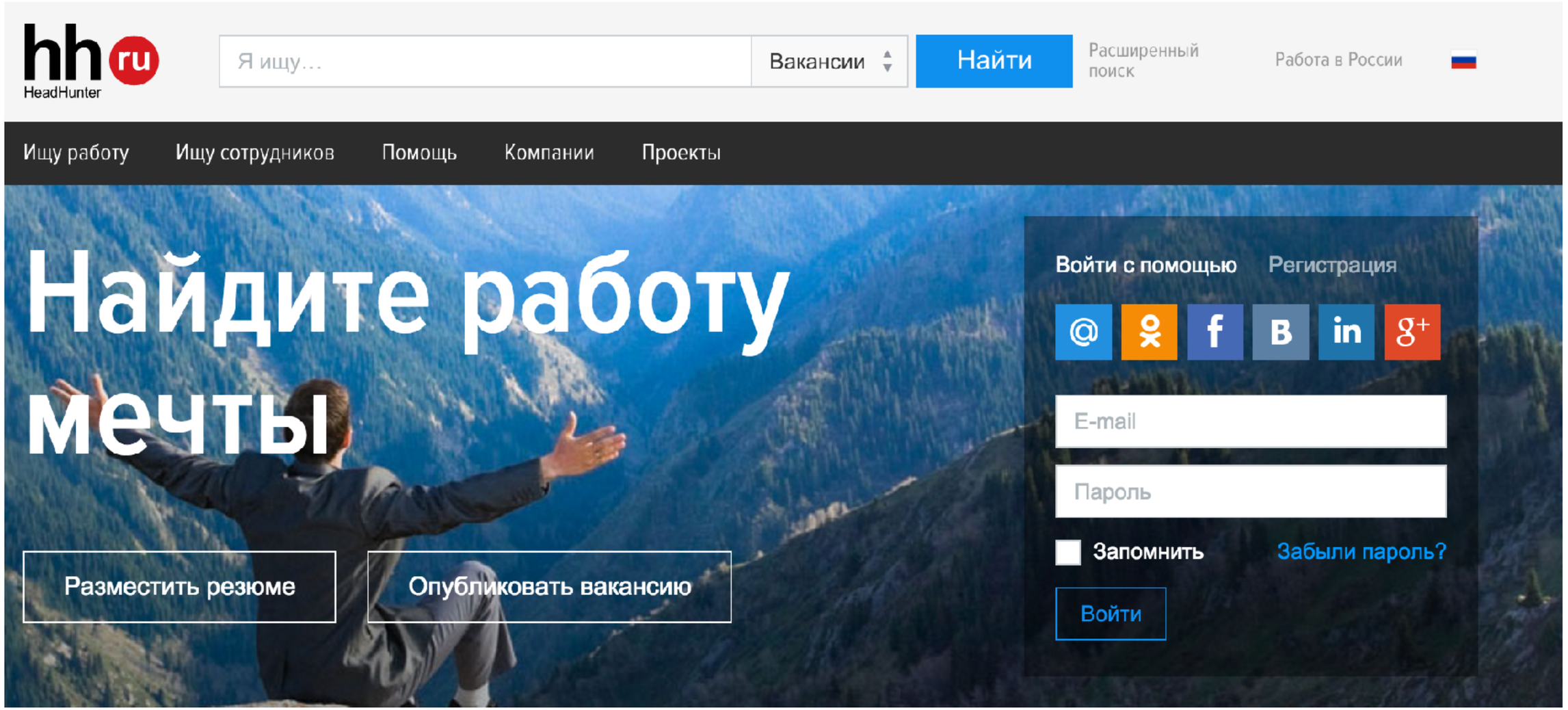
CSS-in-JS — Назад в будущее!

Мостовой Никита
Frontend developer at HeadHunter

HeadHunter


hh ru
HeadHunter
Я ищу…
Вакансии
Найти
Расширенный поиск
Работа в России
Ищу работу
Ищу сотрудников
Помощь
Компании
Проекты
Найдите работу мечты
Разместить резюме
Опубликовать вакансию
Войти с помощью
Регистрация
E-mail
Пароль
Запомнить
Забыли пароль?
Войти


hh
Войти
HeadHunter API
С помощ
соискате
функцио
приложе
Как работа
Банк данных заработных плат
hh Оценка талантов
Войти
Оценка талантов
Оценивайте сотрудников вашей компании или соискателей, у которых нет резюме на hh.ru. Мы сделали все, чтобы работа с сервисом была для вас простой, удобной и безопасной. Просто добавьте информацию о тех, кого вы хотите оценить, отправьте им приглашение на оценку и анализируйте результаты. В сервис включены инструменты оценки от CEB SHL Talent Measurement Solutions.
Подробнее о сервисе


jQuery
write less, do more.


BACKBONE.JS


React


Redux

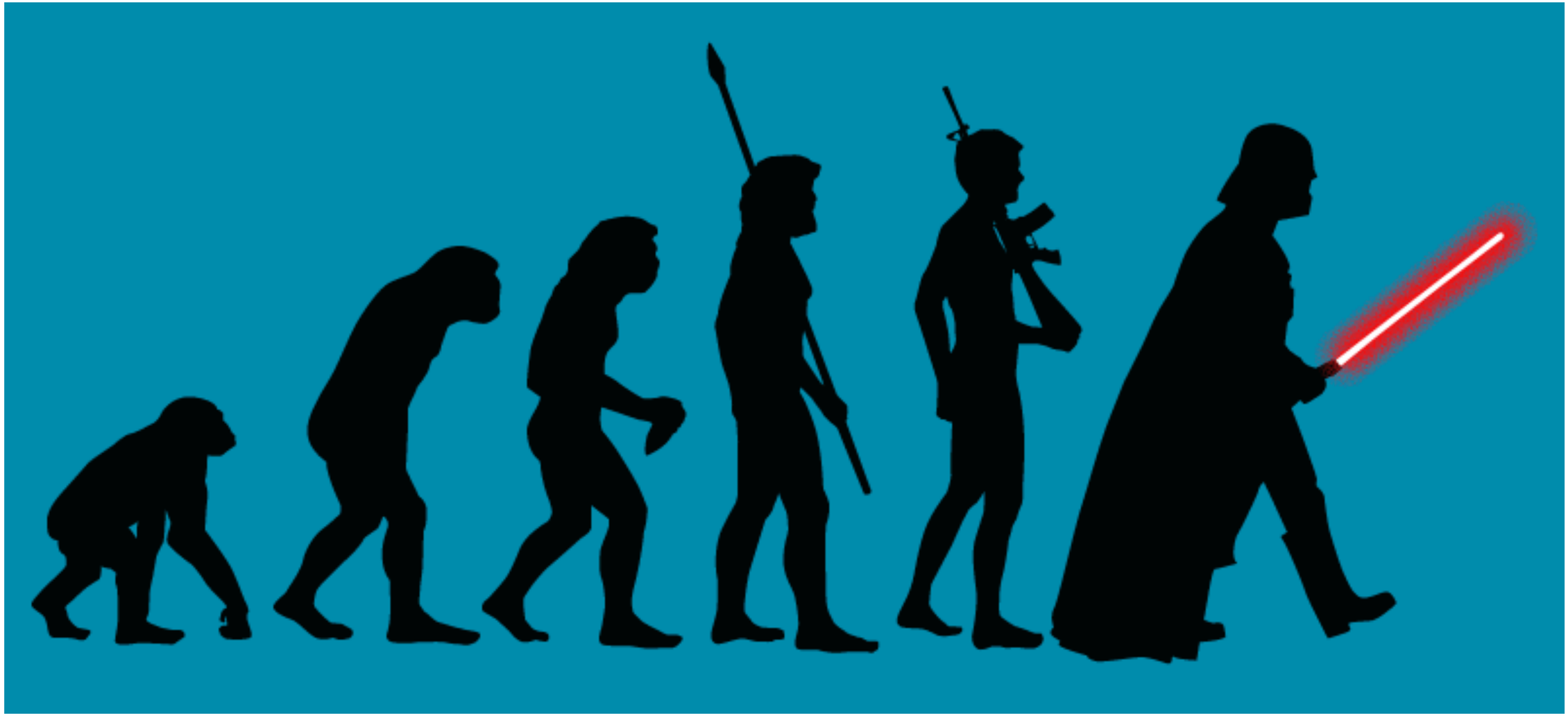

Mastering XML Transformations
XSLT
Hogan.js


python™

Об авторе

- Frontend at HeadHunter
- Работал в командах поиска, мобильного сайта и команде HR-решений
- Twitter: @xnimorz
- xnim.ru

Эволюция фронтенд-разработчика



JavaScript. 1 век до н.э.

```
function calculateTopichHash() {
    window.hashSource = window.employee.field +
window.hash;
    window.murmurhash2();
}

window.murmurhash2 = function() {
    // Работаем с window.hashSource
}
```

JavaScript. Древний мир

```
function calculateTopichHash(employee, vacancy) {
    window.murmurhash2(employee, vacancy);
}
```

```
<script src="/vendor/murmur2.js"></script>
```

JavaScript. Средние века.
Поклонение папе ~~римскому~~ jQuery

```
$(function() {
    $('.test-field').html(hash(window.employee,
window.vacancy));
});
```

```
<script src="/vendor/murmur2.js"></script>
```

JavaScript. Ренессанс

```
(function(MyLib) {
    if (typeof exports === 'object') {
        module.exports = MyLib;
    } else if (typeof define === 'function' &&
define.amd) {
        define(MyLib);
    } else {
        global.MyLib = MyLib;
    }
})(MyLib);
```

JavaScript. Ренессанс

```
define([
    'murmurhash2', 'employee', 'vacancy'
], function(murmurhash2, employee, vacancy) {

    function hash(employee, vacancy) {
        return murmurhash2(employee, vacancy);
    }

    // some code with hash(employee, vacancy)
});
```

Забираем в стандарт

```
import murmurhash2 from 'murmurhash2';
import employee from 'employee';
import vacancy from 'vacancy';

function hash(employee, vacancy) {
    return murmurhash2(employee, vacancy);
}

// some code with hash(employee, vacancy)
```

// а еще в стандарте будет import(), но это уже другая история

На фронте все хорош

CSS. Юрский период

```
<div styles="display: flex; height: 10px">

</div>
```

CSS. 1 век до н.э.

```
.person {
    display: table;
}

.padded {
    padding: 15px;
}

.status {
    display: table-cell;
    color: red;
}
```

JavaScript. 1 век до н.э.

```
function calculateTopichHash() {
    window.hashSource = window.employee.field +
window.hash;
    window.murmurhash2();
}

window.murmurhash2 = function() {
    // Работаем с window.hashSource
}
```

CSS. 1 век до н.э.

```css
.person {
    display: table;
}

.padded {
    padding: 15px;
}

.status {
    display: table-cell;
    color: red;
}
```

```css
.form {
    background-color: #fff;
}

.padded {
    padding: 25px;
}

.status {
    display: block;
    float: right;
}
```

CSS. 1 век до н.э.

Pen

```
.person {
    display: table;
}

.padded {
    padding: 15px;
}

.status {
    display: table-cell;
    color: red;
}
```

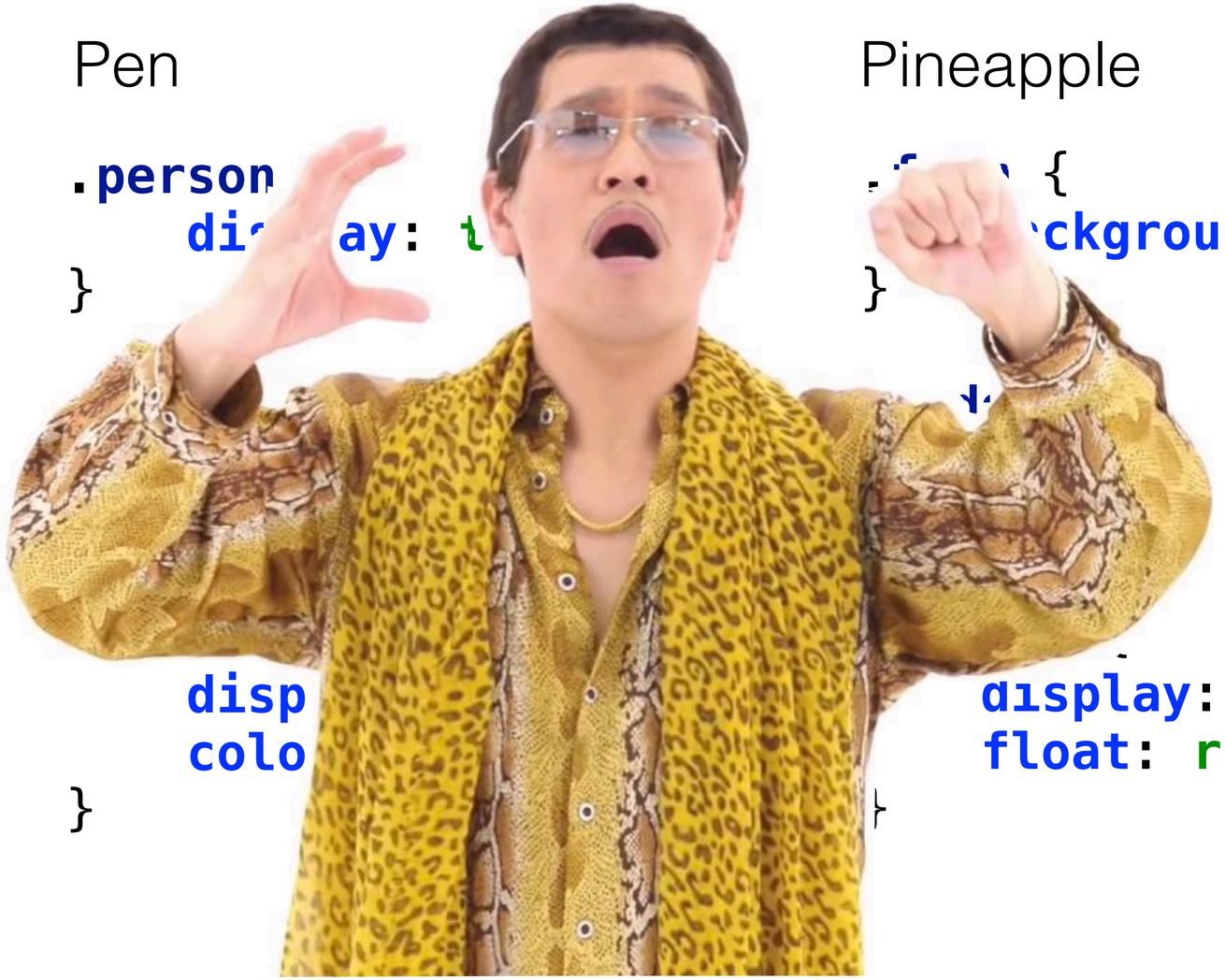
Pineapple

```
.form {
    background-color: #fff;
}

.padded {
    padding: 25px;
}

.status {
    display: block;
    float: right;
}
```

CSS. 1 век до н.э.

Pen

```
.person
    di  ay: t
}

    disp
    colo
}
```

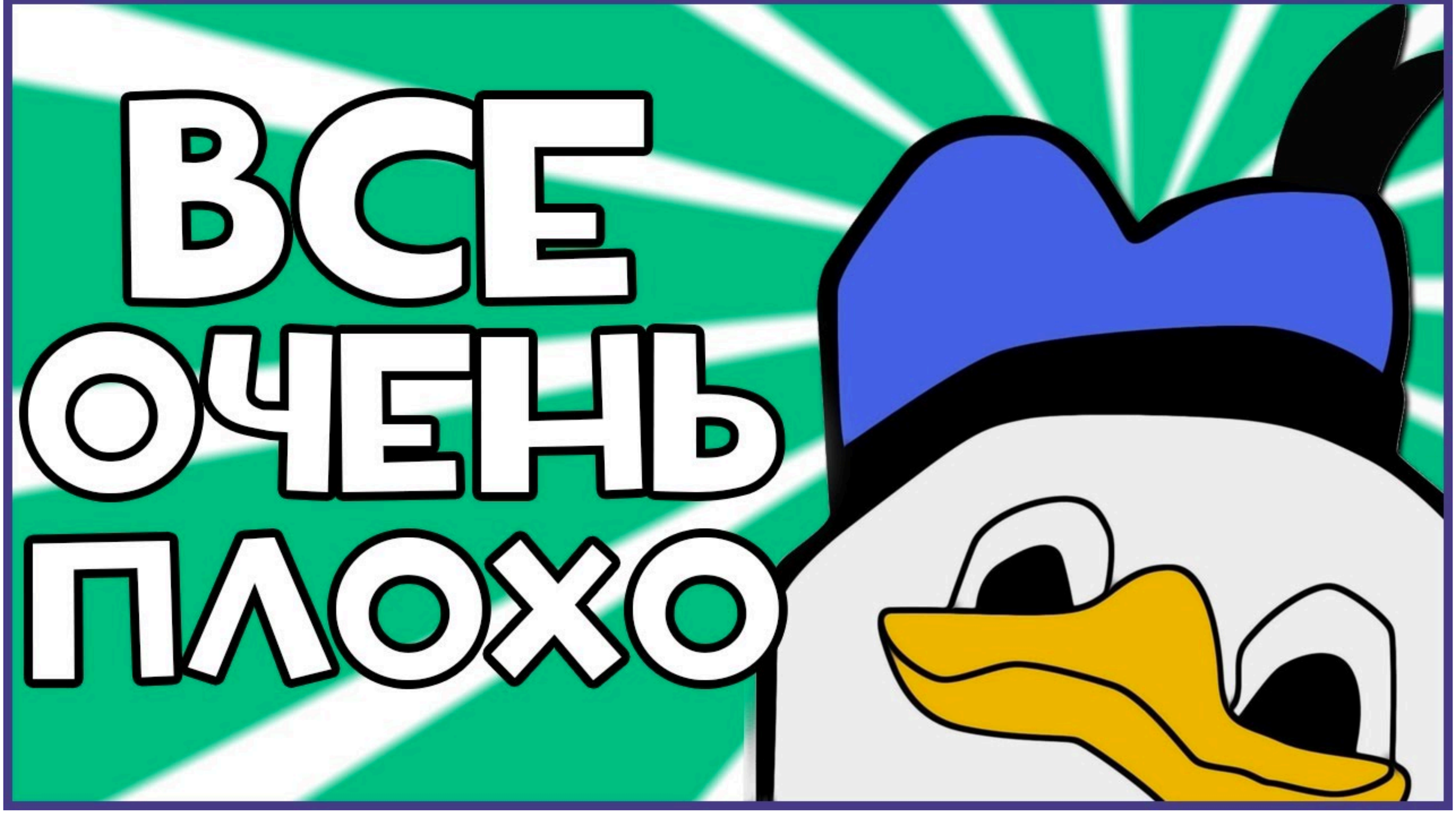


Pineapple

```
.  {
  ckground-color: #fff;
}

  25px;

  display: block;
  float: right;
}
```

Bootstrap

> 600 глобальных переменных!


ВСЕ
ОЧЕНЬ
ПЛОХО

CSS. Костылестроение

- А давайте придумаем OOCSS!
- АНБ позволяет забыть о головной боли с CSS
- Мы вам тут БЭМ принесли!
- У нас SMACCSS!

CSS. Проблемы

- Каскады
- Все пишем в глобальную область видимости — коллизии
- Наследуемые стили
- Браузеров много

БЭМ?

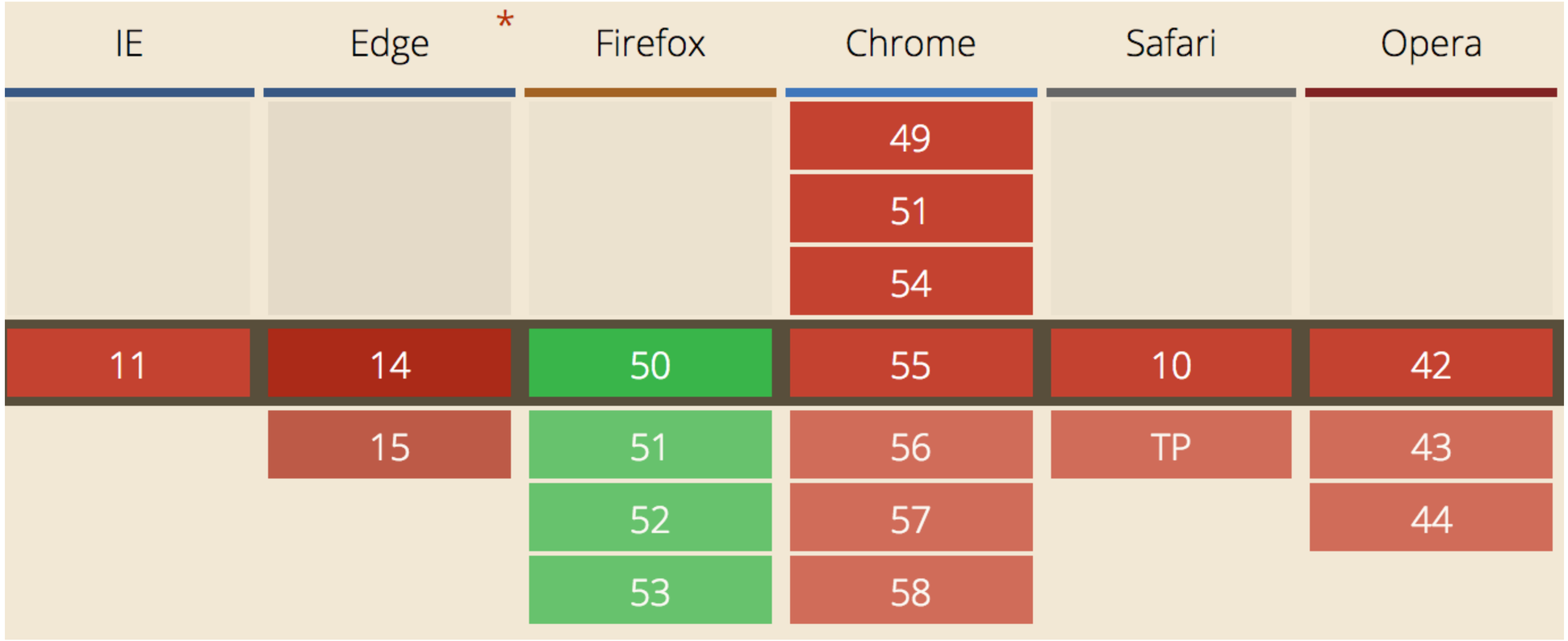
- Спасает частично от коллизий
- Не спасает от наследуемых стилей
- Декларирует использование каскадов

OOCSS/Еще-один-подход

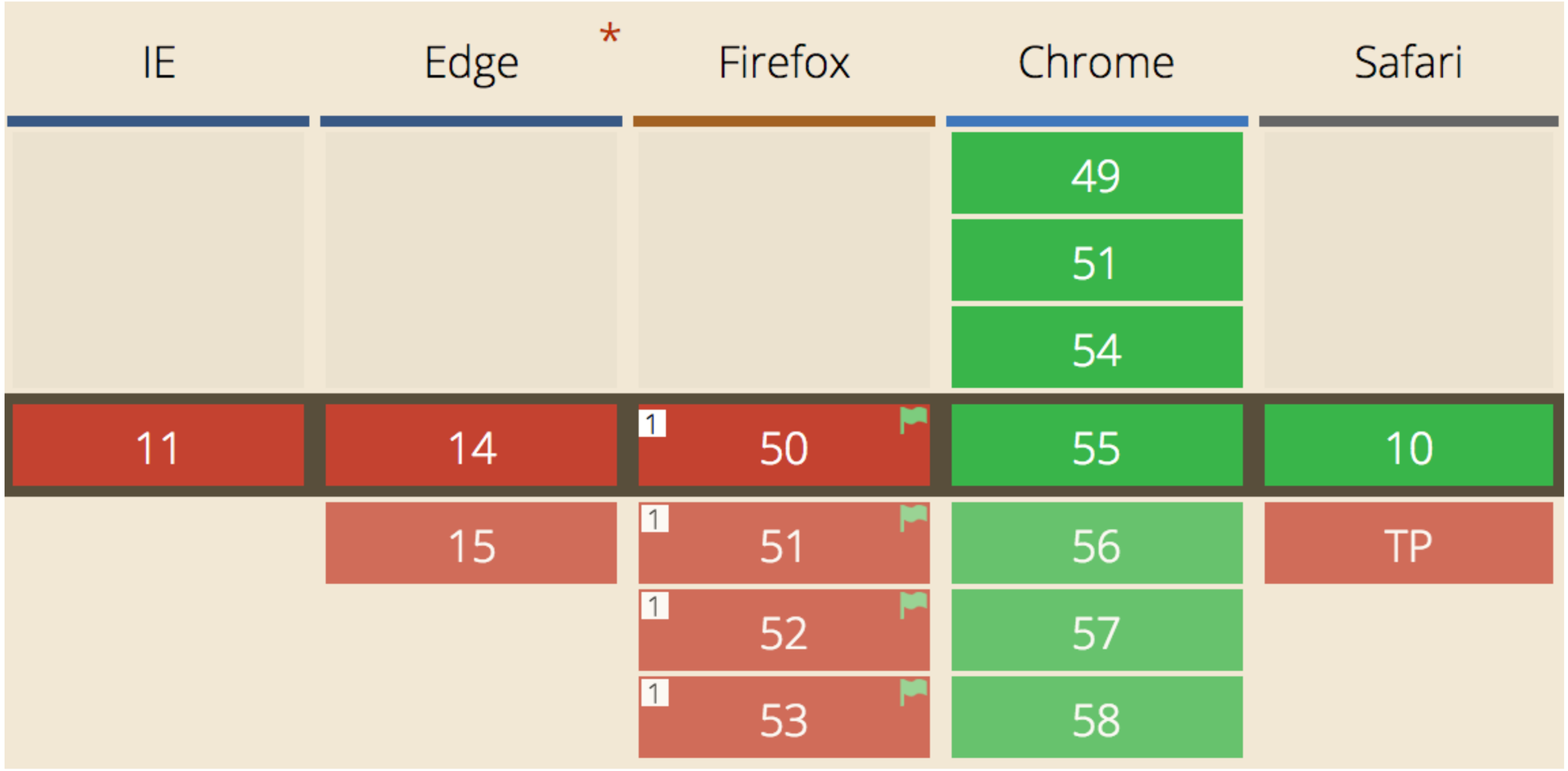
- Спасает частично от коллизий
- Не спасает от наследуемых стилей
- Декларирует использование каскадов

Scoped CSS

Удален из спецификации



ShadowDOM


IE
Edge *
Firefox
Chrome
Safari
49
51
54
11
14
1 50
55
10
15
1 51
56
TP
1 52
57
1 53
58

Polymer

- Решает проблемы коллизий
- Непопулярный, по крайней мере в России в продакшене


Что делать?
pikabu.ru

Не бежать за технологиями, а решать задачи.

Решает задачу? Используем.

Создаёт проблему и решает задачу? Думаем.


НАМ НУЖНО РАЗГРАНИЧЕНИЕ СТИЛЕЙ
БЭМ
OOCSS
А МОЖЕТ БЫТЬ CSS-MODULES ИЛИ CSS-IN-JS?
risovach.ru

А теперь серьезно.

CSS-модули и CSS-in-JS достаточно близкие по духу идеи. Вопрос в деталях

CSS-модули

- Хеширование для разграничения переменных в глобальной области видимости.
- Коллизии решаются "во время компиляции"
- Псевдолокальная область видимости
- БЭМ не нужен
- Стили динамически не меняются

Реализации

- CSS-loader — с помощью параметра можно включить CSS-модули
- PostCSS-Modules
- React-CSS-Modules

Как работает?

```
import './card-styles.less';

class Card extends Component {
    render() {
        return (
            <div className='card'>
                <div className='card__contacts'>
                    {this.renderContacts()}
                </div>
            </div>
        );
    }
}
```

Как работает?

```
import css from './card-styles.less';

class Card extends Component {
    render() {
        return (
            <div className={css.card}>
                <div className={css.cardContacts}>
                    {this.renderContacts()}
                </div>
            </div>
        );
    }
}
```

Как работает?

```
import css from './card-styles.less';

class Card extends Component {
    render() {
        return (
            <div className={css.card}>
                <div className={css.cardContacts}>
                    {this.renderContacts()}
                </div>
            </div>
        );
    }
}
```

Как работает?

- На клиент отправляются уже захешированные непересекающиеся правила
- В JS заводятся переменные, которые и использует приложение
- Любые реализации работают максимально приближенно друг к другу

Какие проблемы решает?

- Глобальное именование
- Коллизии
- Каскады (некоторые реализации)

Где полезно

- Большие сайты
- Отдельные UI-компоненты в npm
- Библиотеки UI-компонентов

Что если объединить CSS и JS

Идея

```
import { StyleSheet, css } from 'aphrodite/no-important';
const styles = StyleSheet.create({
    card: {
        display: 'block',
        '@media print': {
            width: 240,
        },
    }, // и т.д.
});
class Card extends Component {
    render() {
        return (
            <div className={css(styles.card)}>
                <div className={css(styles.contacts)}>
                    {this.renderContacts()}
                </div>
            </div>
        );
    }
}
```

Идея

```
import { StyleSheet, css } from 'aphrodite/no-important';
const styles = StyleSheet.create({
    card: {
        display: 'block',
        '@media print': {
            width: 240,
        },
    }, // и т.д.
});
class Card extends Component {
    render() {
        return (
            <div className={css(styles.card)}>
                <div className={css(styles.contacts)}>
                    {this.renderContacts()}
                </div>
            </div>
        );
    }
}
```

CSS-in-JS

- Клиент + сервер
- Покрывают весь цикл создания CSS от написания до вставки в DOM

Чем отличаются CSS-in-JS решения

- Ленивость
- Полное или частичное кеширование
- Дробление стилей на отдельные правила
- Композиция классов на одном элементе

Подборка CSS-in-JS

Features

How to read the table

More crosses doesn't mean "better", it depends on your needs. For example, if a package supports the css file extraction you can run the autoprefixing at build time.

Package	Version	Automatic Vendor Prefixing	Pseudo Classes	Media Queries	Styles As Object Literals	Extract CSS File
aphrodite	0.1.2	x	x	x	x	x
babel-plugin-css-in-js	1.2.2	x	x	x	x	x
bloody-react-styled	3.0.0		x	x		
classy	0.3.0		x	x	x	
csjs	1.0.0		x	x		
css-constructor	0.1.1	x	x	x		
css-loader	0.15.6		x	x		x
css-ns	1.0.0		x	x		x
cssobj	0.2.21	x	x	x	x	
cssx-loader	3.8.0	x	x	x		x

Поиск решения в проект

Рассматриваемые CSS-in-JS библиотеки или как я выбирал либу в проект

- Aphrodite
- JSS
- Styled-Components
- Babel-plugin-css-in-js
- Styletron

Aphrodite

```
import { StyleSheet, css } from 'aphrodite/no-important';
const styles = StyleSheet.create({
    card: {
        display: 'block',
        '@media print': {
            width: 240,
        },
    }, // и т.д.
});
class Card extends Component {
    render() {
        return (
            <div className={css(styles.card)}>
                <div className={css(styles.contacts)}>
                    {this.renderContacts()}
                </div>
            </div>
        );
    }
}
```

Добавление данных в стор

Aphrodite

```
import { StyleSheet, css } from 'aphrodite/no-important';
const styles = StyleSheet.create({
    card: {
        display: 'block',
        '@media print': {
            width: 240,
        },
    }, // и т.д.
});
class Card extends Component {
    render() {
        return (
            <div className={css(styles.card)}>
                <div className={css(styles.contacts)}>
                    {this.renderContacts()}
                </div>
            </div>
        );
    }
}
```

Вставка в stylesheet

Aphrodite

```
import { StyleSheet, css } from 'aphrodite/no-important';
const styles = StyleSheet.create({
    a: {height: 100}
    card: {
        height: 200,
        '@media print': {
            width: 240,
        },
    }, // и т.д.
});
class Card extends Component {
    render() {
        return (
            <div className={css(styles.card, styles.a)}>
                <div className={css(styles.contacts)}>
                    {this.renderContacts()}
                </div>
            </div>
        );
    }
}
```

Чему будет равна высота?

Aphrodite

Плюсы:

- Удобный и простой синтаксис
- Серверный рендеринг
- Рендеринг через stylesheet.ruleset
- Удобно дебажить

Aphrodite

Минусы:

- Не всегда предсказуемое поведение (если переходим из CSS-модули или обычного CSS)
- Не всегда удобно дебажить композицию классов (ИМХО)
- По умолчанию добавляет везде !important
- Вставка стилей в DOM через setTimeout

Styled-Components

```
import styled from 'styled-components';
const Title = styled.h1`font-size: 1.5em;`;

const Wrapper = styled.section`padding: 4em;`;

export default (
    <Wrapper>
        <Title>Мой заголовок</Title>
    </Wrapper>
);
```

Styled-Components

- Сложно писать стили без плагинов, которые будут автоматизировать преобразование стилей в строку.

Babel-plugin-css-in-js

```
const styles = cssInJS({
    card: {
        height: '100%',
    },
});
class Card extends Component {
    render() {
        return (
            <div className={styles.card}>
                {this.renderCard()}
            </div>
        );

    }
}
```

Babel-plugin-css-in-js

Плюсы:

- Преобразует стили сразу (+ может на лету)
- Создает CSS + JS на выходе

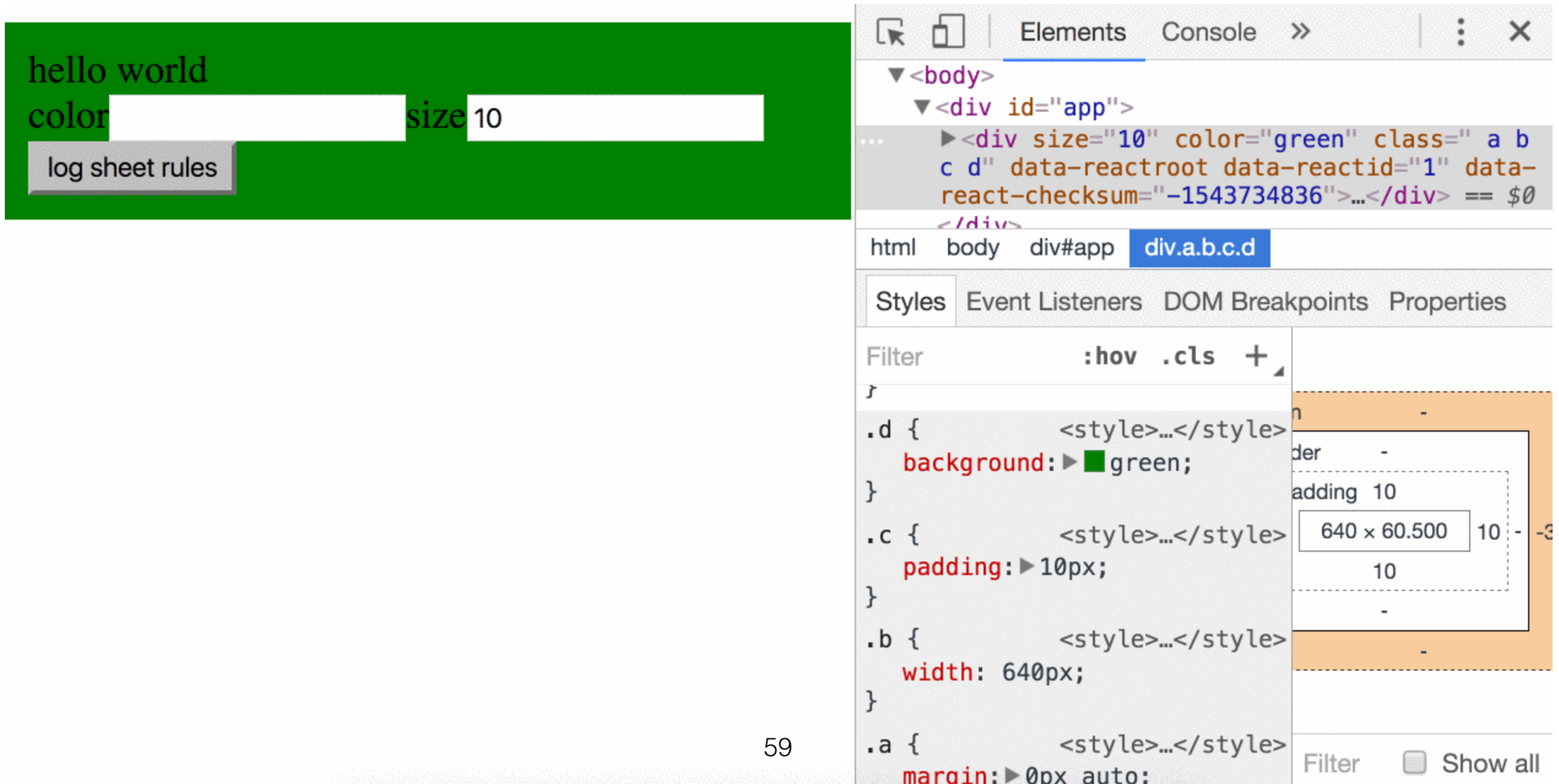
Минусы:

- "Неявная" работа
- Были проблемы с сорсмапами
- Система плагинов не развита

Styletron

```
import {styled} from 'styletron-react';

const Card = styled('div', {
    height: '100%',
});

<Card>Hello World</Card>
```

Styletron


hello world
color
size 10
log sheet rules
Elements
Console
<body>

<div size="10" color="green" class=" a b c d" data-reactroot data-reactid="1" data-react-checksum="-1543734836">… == $0
html
body
div#app
div.a.b.c.d
Styles
Event Listeners
DOM Breakpoints
Properties
Filter
:hov
.cls
.d {
<style>…</style>
background: green;
}
.c {
<style>…</style>
padding: 10px;
}
.b {
<style>…</style>
width: 640px;
}
.a {
<style>…</style>
margin: 0px auto;
adding 10
640 × 60.500
10
10
Filter
Show all

Styletron

Преимущество:

- Высокая скорость работы

Недостаток:

- Сложно дебажить CSS

JSS

```
import jss from 'jss';

const { classes } = jss.createStyleSheet({
    card: {
        height: 100,
    },
}).attach();

class Card extends Component {
    render() {
        return (
            <div className={classes.card}>
                {this.renderContacts()}
            </div>
        );
    }
}
```

JSS

```
import jss from 'jss';

const { classes } = jss.createStyleSheet({
    card: {
        height: 100,
    },
}).attach();

class Card extends Component {
    render() {
        return (
            <div className={classes.card}>
                {this.renderContacts()}
            </div>
        );
    }
}
```

JSS

Плюсы:

- Развитая система плагинов
- Удобный синтаксис
- Нет setTimeout

JSS

Минусы:

- На attach создается отдельный тег style

Что решаем?

- Каскады (частично)
- Коллизии (аналогично CSS-модулям, но в рантайме)
- Глобальное именование

Добавляем плагины

- Наследование
- Каскады

Где полезно

- Веб-приложения (SPA)
- Отдельные UI-компоненты в npm
- Библиотеки UI-компонентов

За бортом доклада

- CXS
- Aphrodisiac (JSS в синтаксисе aphrodite)
- и еще много и много других решений

Если ты создаешь свой CSS-in-JS

- Напиши статью разоблачающую другие библиотеки
- Приведи сравнение в производительности, если у тебя круче
- Если не круче, объясни, почему нужно использовать твою либу

Если ты создаешь свой CSS-in-JS

Итого

- CSS-in-JS — шаг не только в интеграции JS и CSS, но и очередная попытка решить "проблемы" CSS
- Внедрять подобное решение в продакшн — после оценки его и понимания плюсов и минусов
- Нужен стандарт (с использованием веб-компонентов)

Что почитать

- Кристофер Шедоу
- Сравнение CSS-in-JS решений

Спасибо за внимание!