

# CSSO — сжимаем CSS

Роман Дворнов  
Avito



- Работаю в [Avito](#)
- Делаю [SPA](#)
- Автор [basis.js](#)
- Мейтнер [CSSO](#)

За любую движуху,  
кроме голодовки ;)

# Оптимизация

## и сжатие CSS



для ускорения  
сайта

# Оптимизация

## и сжатие CSS



для ускорения  
сайта





«Хэппи-энда» не будет

A photograph of three women from the TV show 'The Big Bang Theory'. On the left, Kaley Cuoco (as Penny) has her mouth wide open in surprise. In the center, Melissa Rauch (as Bernadette) wears glasses and a yellow cardigan, looking directly at the camera with a neutral expression. On the right, Mayim Bialik (as Amy) also has her mouth open in surprise, wearing red-rimmed glasses and a blue top. They are sitting on a green couch in a living room setting. A black rectangular box with white text is overlaid on the bottom left of the image.

CSS-минификаторы не нужны!



CSS-минификаторы не нужны!

Шутка... почти

Быстрые  
браузеры



«Тяжелые»  
сайты

Быстрые  
браузеры



«Тяжелые»  
сайты

Много CSS – нужно сжимать

Чем сжимать?



# Герои сегодняшнего дня

# Герои сегодняшнего дня



**ycssmin**

YUI Compressor fork

# Герои сегодняшнего дня



**ycssmin**

YUI Compressor fork

# Герои сегодняшнего дня



ycssmin

YUI Compressor fork

csso

# Герои сегодняшнего дня



**y.cssmin**

YUI Compressor fork

**csso**

# Герои сегодняшнего дня



**ycssmin**

YUI Compressor fork

**csso**

**clean-css**

# Герои сегодняшнего дня



**ycssmin**

YUI Compressor fork

**csso**

**clean-css**

**cssnano**

# Герои сегодняшнего дня



**ycssmin**

YUI Compressor fork

**csso**

**clean-css**

**cssnano**

Минификаторов гораздо больше,  
но либо не популярны  
либо слабо развиваются

Сравниваем



**clean-css 3.4.9****cssnano 3.5.2****csso 2.0.0****bootstrap.css****118 186**

147 427 байт

273 ms

**117 440****1 813 ms****117 756****169 ms****foundation.css****142 667**

200 341 байт

389 ms

**145 030****1 983 ms****144 262****222 ms****normalize.css****1792**

7 707 байт

5 ms

**1824****17 ms****1 831****4 ms****reset.css****758**

1 092 байт

3 ms

**773****13 ms****747****3 ms**[goalsmashers.github.io/css-minification-benchmark/](https://goalsmashers.github.io/css-minification-benchmark/)



Выглядит как-то так...

Библиотеки пишутся

оптимально,

реальный CSS – нет

# Наши цифры

	<b>clean-css 3.4.9</b>	<b>cssnano 3.5.2</b>	<b>csso 2.0.0</b>
--	------------------------	----------------------	-------------------

<b>ActiAgent.ru</b>	<b>430 240</b>	<b>439 024</b>	<b>435 588</b>
602 233 байт (5 мес назад)	1 077 ms	23 270 ms	531 ms

<b>ActiAgent.ru</b>	<b>587 906</b>	<b>604 503</b>	<b>595 834</b>
822 021 байт (сейчас)	1 705 ms	48 550 ms	616 ms

В gzip фактор сжатия 8 (~72Kb)

Результат можно улучшить!



Минификация



Все минификаторы работают похоже

# Базовая минификация

- Удаление
- Замена значений
- Структурная оптимизация

Кажется, минификация CSS –  
это про знание спецификаций



На деле – постоянно что-то вылезит

# Потому что

- Спецификации меняются
- Разная поддержка браузерами
- Баги браузеров
- Хаки

Самое главное  
минификатор не должен  
ломать или чинить CSS



Удаление

# Удаляем

- Пробелы и комментарии (основной выигрыш)
- Правила с неверными селекторами
- Пустые правила
- Неверные декларации
- Неверно расположенные `@import`, `@charset`
- ...

Но нужно учитывать  
особенности спецификаций

# Удаление пробелов

Оригинальный CSS

```
calc(4 * 2em - 10% / 3)
```

Не правильно

```
calc(4*2em-10%/3)
```

Правильно

```
calc(4*2em - 10%/3)
```

# Еще примеры

- Единицы измерения у нулей

`0px` → `0`

- Кавычки

`[attr="name"]` → `[attr=name]`

`url('image.png')` → `url(image.png)`

- ...

# Но всегда есть нюансы...

- $\theta_{px} \rightarrow 0$

МОЖНО

# Но всегда есть нюансы...

- $0px \rightarrow 0$

МОЖНО

- $0deg \rightarrow 0$

НЕЛЬЗЯ, ТАК КАК НЕ ДЛИНА

# Но всегда есть нюансы...

- $0px \rightarrow 0$   
можно
- $0deg \rightarrow 0$   
нельзя, так как не длина
- `flex: 1 0 0px` → `flex: 1 0 0`  
нельзя, сломается в IE



Замена

Замена значений на более  
короткие эквиваленты

# Наиболее интересное: цвет

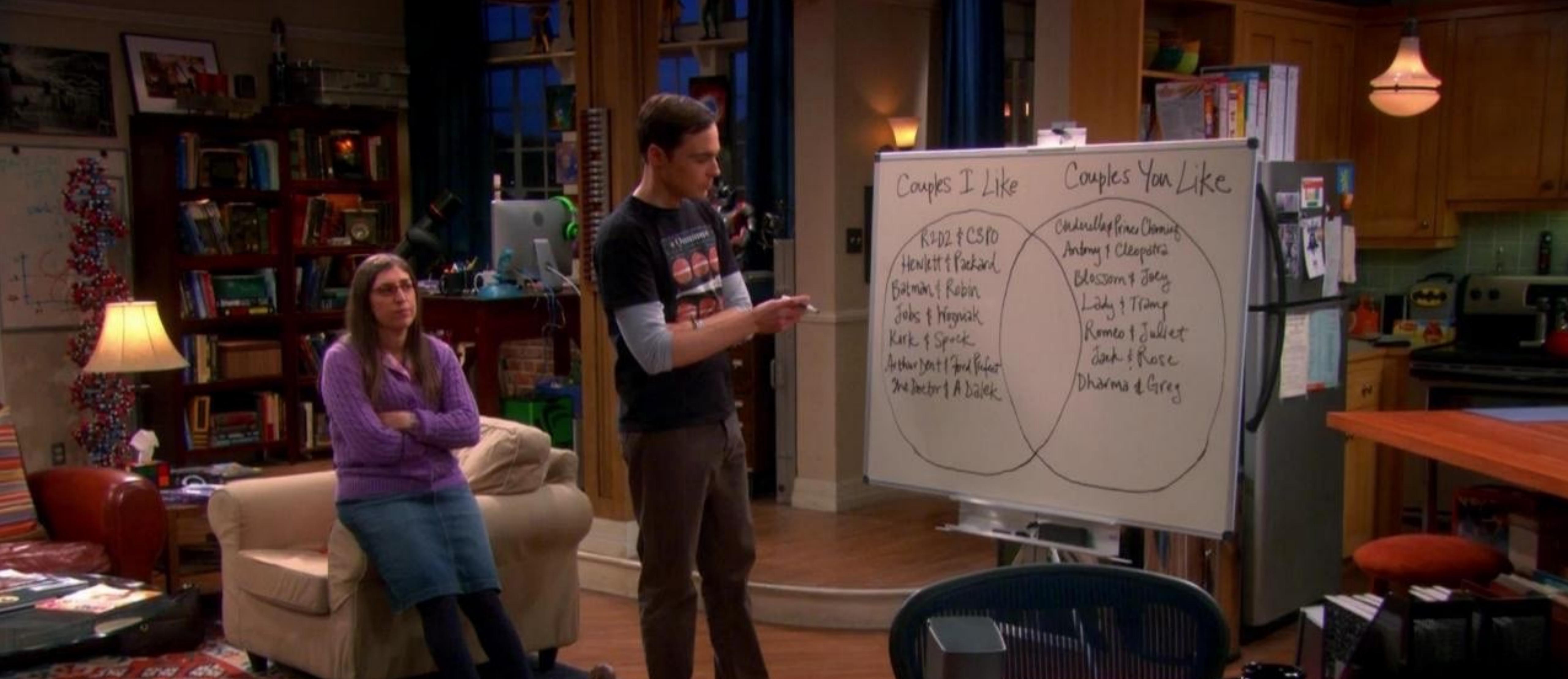
- $\text{hsl} \rightarrow \text{rgb}$ ,  $\text{hsla} \rightarrow \text{rgba}$
- $\text{rgb}(100\%, 0, 0) \rightarrow \text{rgb}(255, 0, 0)$
- $\text{rgba}(a, b, c, 1) \rightarrow \text{rgb}(a, b, c)$
- нормализация:  $\text{rgb}(500, -100, 0) \rightarrow \text{rgb}(255, 0, 0)$
- $\text{rgb}(255, 0, 0) \rightarrow \text{\#ff0000}$
- $\text{\#aabbc} \rightarrow \text{\#abc}$
- $\text{\#ff0000} \rightarrow \text{red}$ ,  $\text{darkslateblue} \rightarrow \text{\#483d8b}$

# Что еще

- Нормализация чисел:  $0.00 \rightarrow 0$  или  $0.123 \rightarrow .123$
- Специфика для свойств
  - `font-weight:bold` → `font-weight:700`
  - `background:none` → `background:0 0`
- `from` → `0%, 100%` → `to` в `@keyframes`
- ...



Не сильно эффективно



Структурная оптимизация

# Слияние и перемещение деклараций и правил



Самая сложная и ресурсоемкая  
оптимизация

# Удаление ненужных деклараций

```
.foo {  
color: red;  
color: green;  
}
```



color: red никогда не будет использовано браузером –  
можно удалить

# Внимание! Викторина

Насколько вы хороший минификатор ;)

# Удаление ненужных деклараций

Правильно?

```
.foo {  
color: red;  
color: rgba(..);  
}  
      
```



```
.foo {  
color: rgba(..);  
}
```

# Удаление ненужных деклараций

```
.foo {  
color: red;  
color: rgba(..);  
}
```

Правильно?



```
.foo {  
color: rgba(..);  
}  
НЕВЕРНО
```

В старых браузерах  
не поддерживается `rgba()`

# Перегруппировка

Правильно?

```
.foo {  
    color: red;  
}  
.bar {  
    color: green;  
}  
.qux {  
    color: red;  
}
```



```
.foo, .qux {  
    color: red;  
}  
.bar {  
    color: green;  
}
```

# Перегруппировка

Правильно?

```
.foo {  
    color: red;  
}  
.bar {  
    color: green;  
}  
.qux {  
    color: red;  
}
```

~~.foo, .qux {  
 color: red;  
}  
.bar {  
 color: green;  
}~~

Разный эффект, например:  
`<div class="bar qux">`

# Перегруппировка

Правильно?



```
span {  
    color: red;  
}  
div {  
    color: green;  
}  
ul {  
    color: red;  
}
```

```
span, ul {  
    color: red;  
}  
div {  
    color: green;  
}
```

# Перегруппировка

Правильно?

```
span {  
    color: red;  
}  
  
div {  
    color: green;  
}  
  
ul {  
    color: red;  
}
```



```
span, ul {  
    color: red;  
}  
  
div {  
    color: green;  
}
```

Правильно,  
у элементов одно имя

# Перегруппировка

Правильно?

```
.foo {  
    color: red;  
}  
  
span {  
    color: green;  
}  
  
.bar {  
    color: red;  
}
```



```
.foo, .bar {  
    color: red;  
}  
  
span {  
    color: green;  
}
```

# Перегруппировка

Правильно?

```
.foo {  
    color: red;  
}  
  
span {  
    color: green;  
}  
  
.bar {  
    color: red;  
}
```

```
.foo, .bar {  
    color: red;  
}  
  
span {  
    color: green;  
}
```

Правильно,  
разная специфичность –  
порядок не важен

# Перегруппировка

Правильно?

```
.foo {  
  color: red;  
}  
.bar:not(.baz) {  
  color: red;  
}
```



```
.foo,  
.bar:not(.baz) {  
  color: red;  
}
```

# Перегруппировка

Правильно?

```
.foo {  
  color: red;  
}  
.bar:not(.baz) {  
  color: red;  
}
```



```
.foo,  
.bar:not(.baz) {  
  color: red;  
}
```



Старые браузеры  
не поддерживают :not()

И Т.Д. И Т.П.

# Вынос общих частей

```
.foo {  
    color: red;  
    width: 100px;  
}  
.bar {  
    color: green;  
    width: 100px;  
}
```



```
.foo, .bar {  
    width: 100px;  
}  
.foo {  
    color: red;  
}  
.bar {  
    color: green;  
}
```

Выносить можно в каждом  
случае по-разному

# Вынос общих частей

```
.foo {  
    color: red;  
}  
.bar {  
    color: red;  
    color: rgba(..);  
}
```



```
.foo, .bar {  
    color: red;  
}  
.bar {  
    color: rgba(..);  
}
```

В данном примере можно только **в начало**

# Вынос общих частей

```
.foo {  
    color: rgba(..);  
}  
.bar {  
    color: red;  
    color: rgba(..);  
}
```



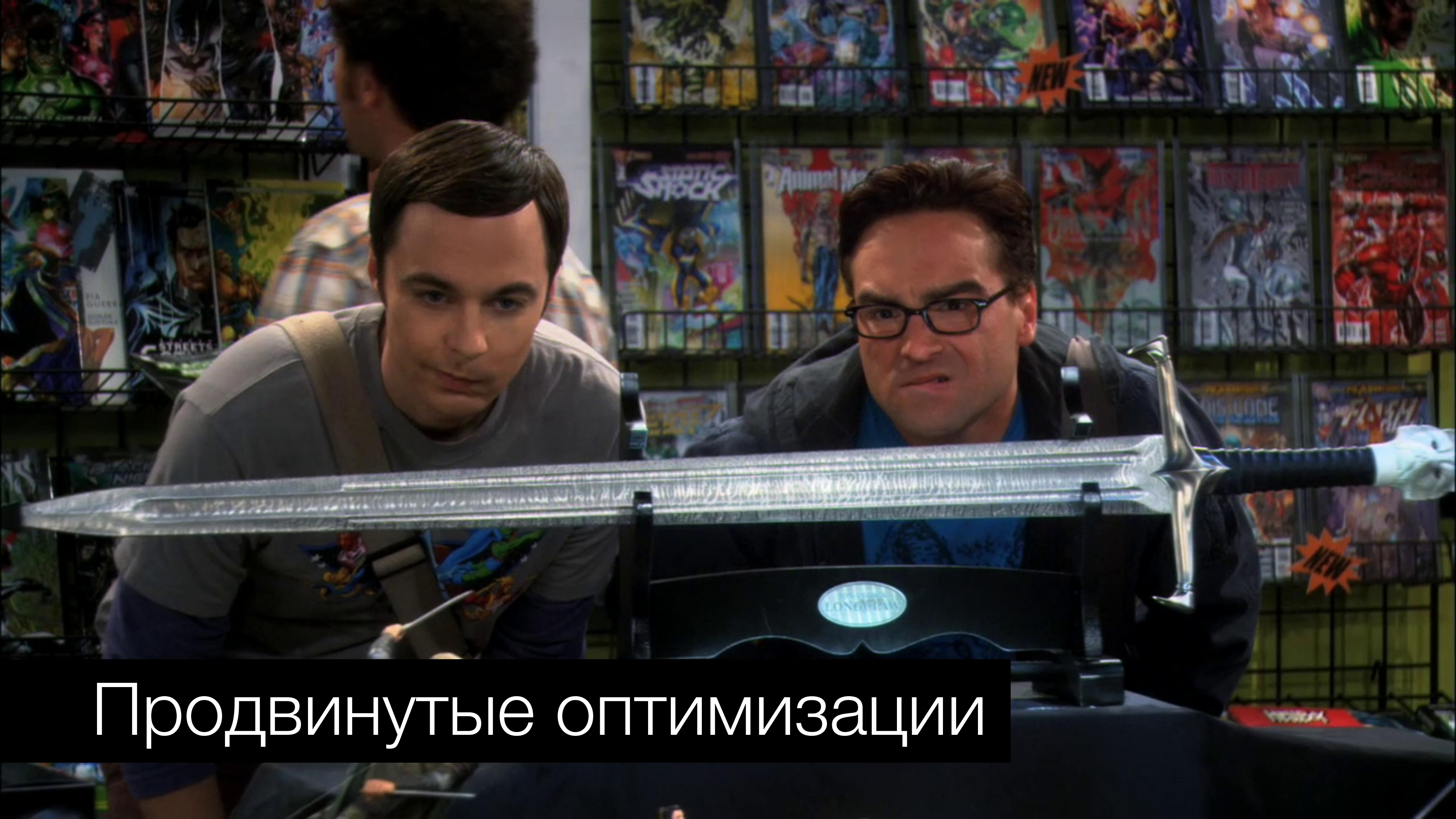
```
.bar {  
    color: red;  
}  
.foo, .bar {  
    color: rgba(..);  
}
```

В данном примере можно только **в конец**

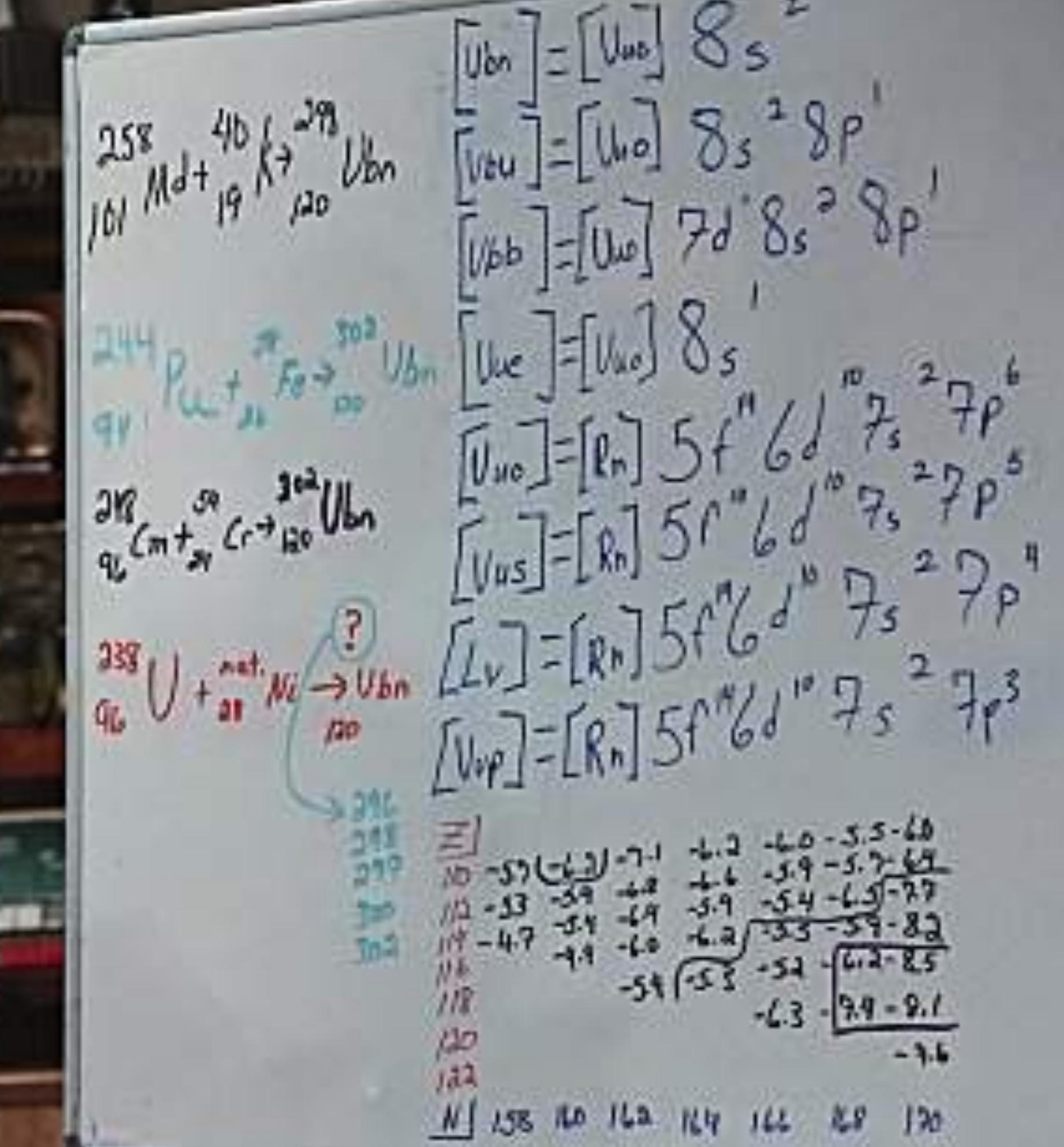
Очень много нюансов,  
нужно знать специфику  
свойств и селекторов

# Базовая оптимизация

- Похожие методы
- Основной выигрыш дает удаление пробелов
- Много хаков
- У всех есть ошибки



Продвинутые оптимизации



Usage data

# Вспомним пример

```
.foo {  
    color: red;  
}  
.bar {  
    color: green;  
}  
.qux {  
    color: red;  
}
```



```
.foo, .qux {  
    color: red;  
}  
.bar {  
    color: green;  
}
```

Трансформация не безопасна,  
так как мы не знаем как  
используется CSS в разметке

Но что, если мы знаем как  
используется?

# Фильтрация

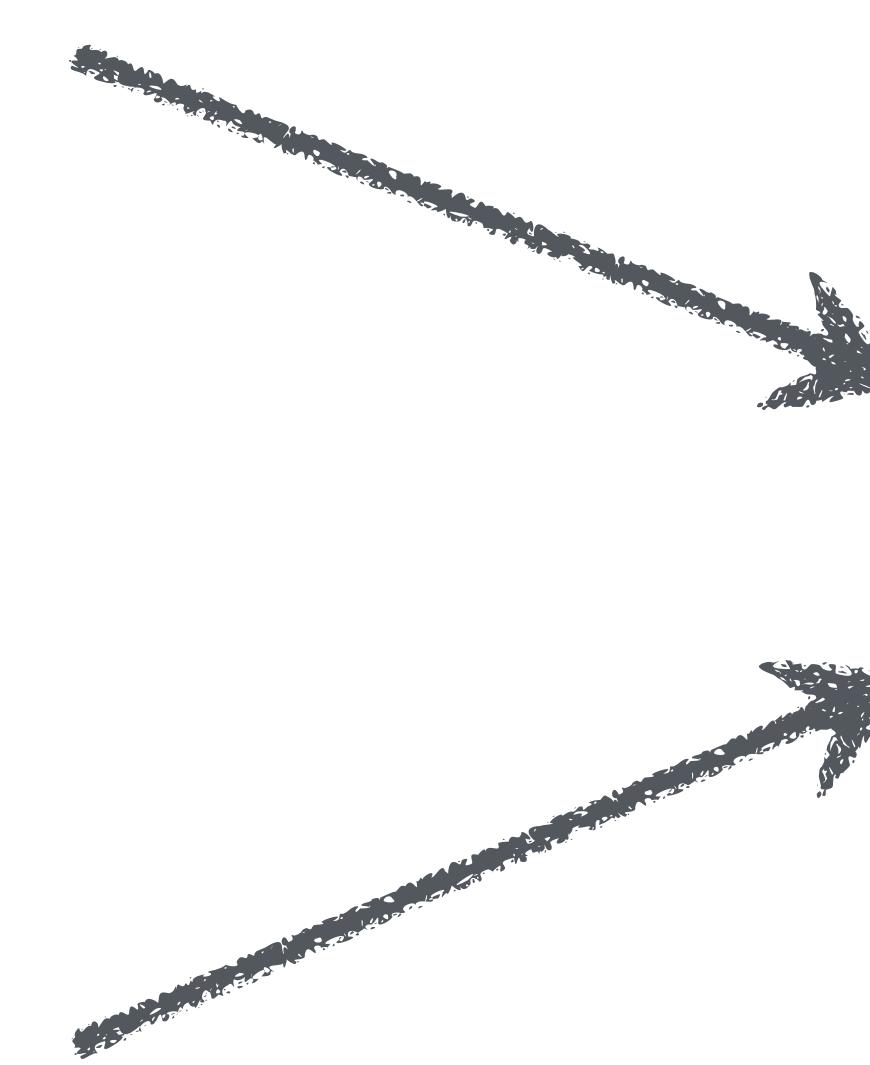
## CSS

```
.foo { color: red }
div.bar { color: green }
ul li, ol li { color: blue }
```

+

## usage.json

```
{
  "classes": ["foo", "bar"],
  "tags": ["ul", "li"]
}
```



## Результат

```
.foo { color: red }
ul li { color: blue }
```

# Scopes

# Пример

```
.module1-foo { background: red; }
```

```
.module1-bar { font-size: 1.5em; background: yellow; }
```

```
.module2-baz { background: red; }
```

```
.module2-qux { font-size: 1.5em; background: yellow; width: 50px; }
```

Нельзя объединить  
.module1-foo и .module2-baz,  
так как между ними .module1-bar

# Пример

```
.module1-foo { background: red; }
```

```
.module1-bar { font-size: 1.5em; background: yellow; }
```

```
.module2-baz { background: red; }
```

```
.module2-qux { font-size: 1.5em; background: yellow; width: 50px; }
```

Минификатор не знает,  
что имена не смешиваются  
и можно безопасно перемещать

# Usage data

```
{  
  "scopes": [  
    ["module1-foo", "module1-bar"],  
    ["module2-baz", "module2-qux"]  
  ]  
}
```

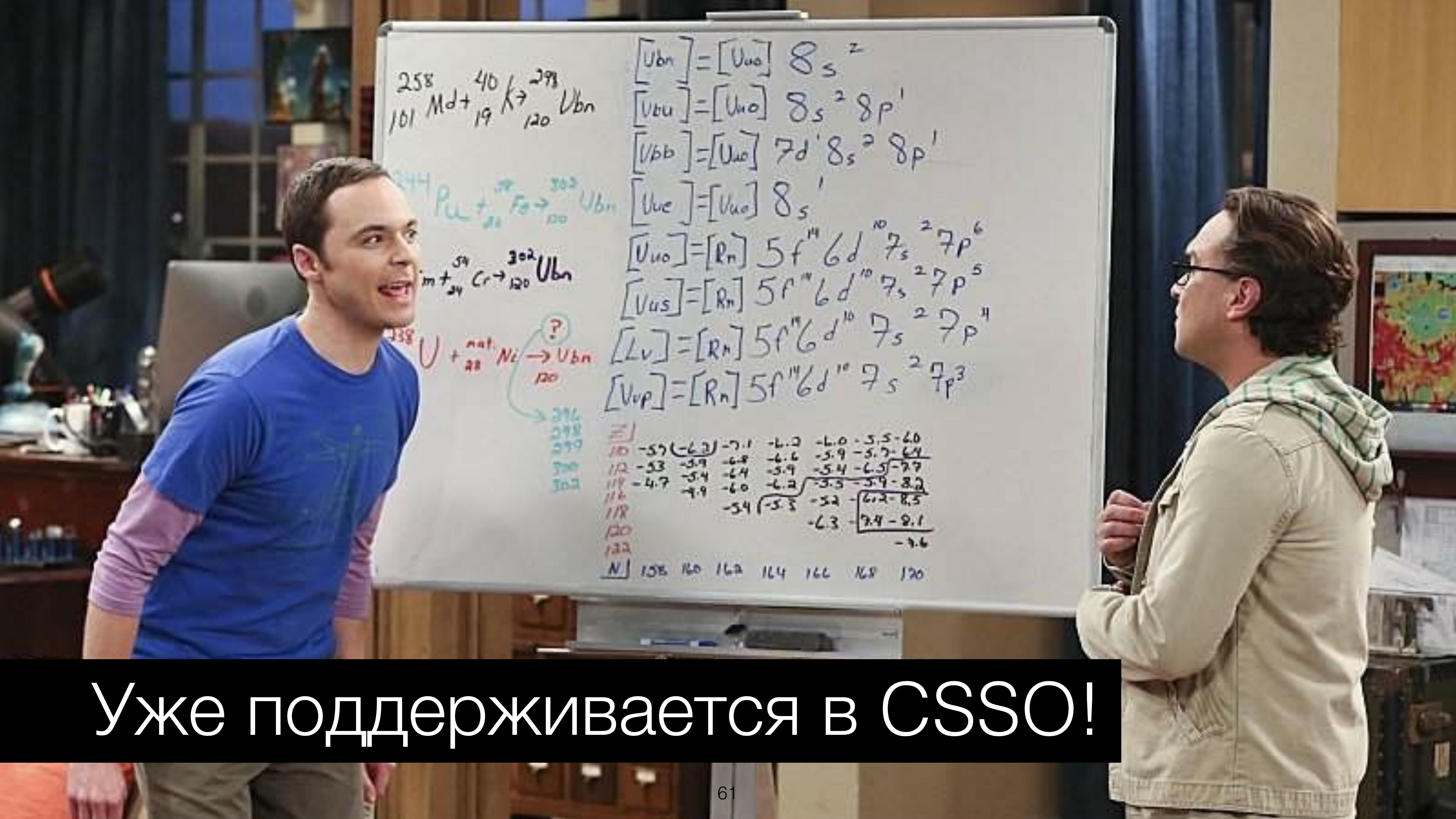
Так мы гарантируем, что классы  
`module1-*` и `module2-*`  
не встречаются на одном элементе

# Результат с usage data

```
.module1-foo, .module2-baz{background:red}  
.module1-bar, .module2-qux{font-size:1.5em;background:#ff0}  
.module2-qux{width:50px}
```

На 29 байт меньше, чем без usage data

Уже поддерживается в CSSO!



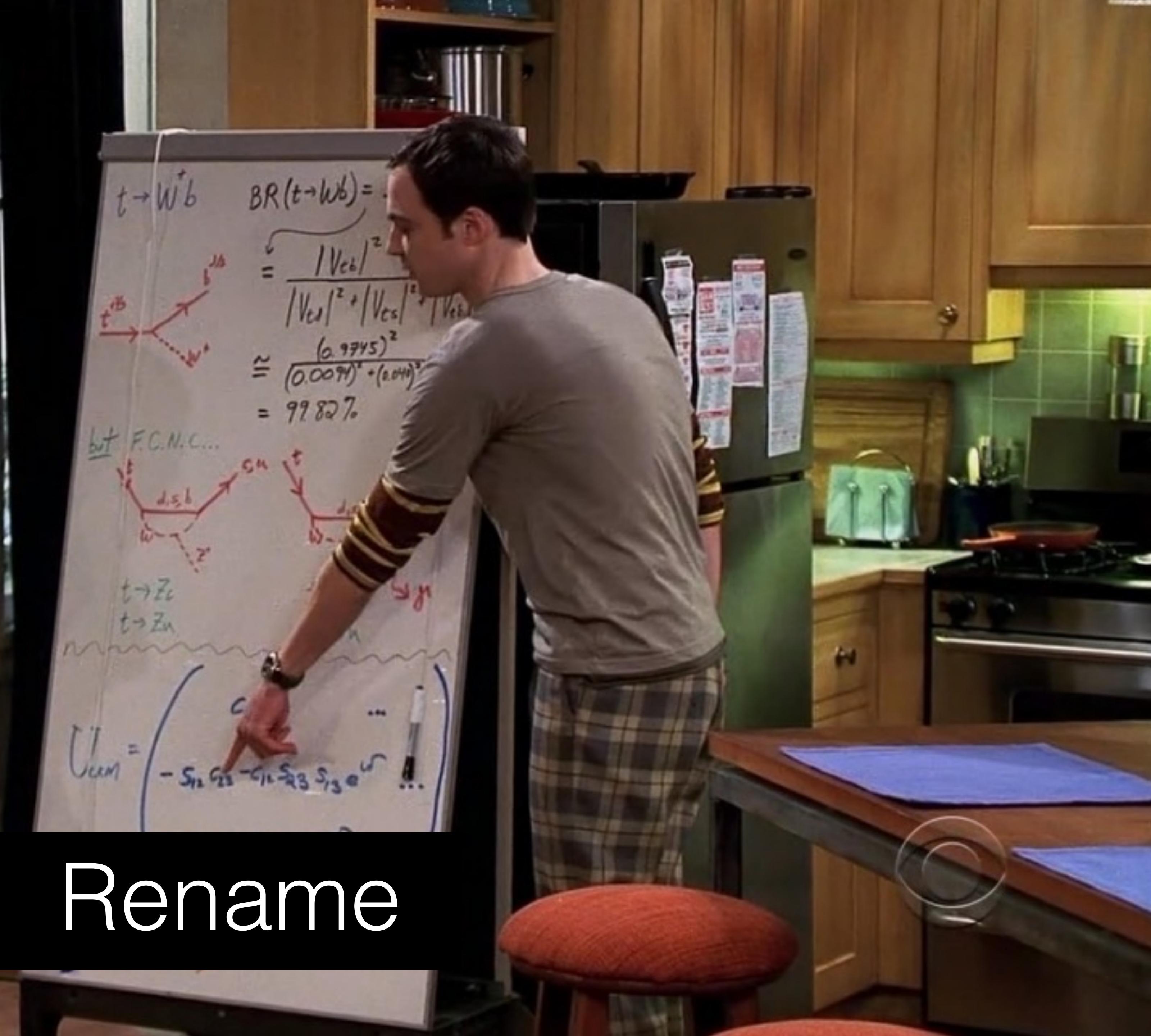
# Что это дало нашему проекту

- 823 Кб Оригинальный CSS
- 596 Кб Базовая минификация
- 437 Кб Минификация с usage data

Улучшение результата на 159Кб (26%)

# Как генерировать usage data?

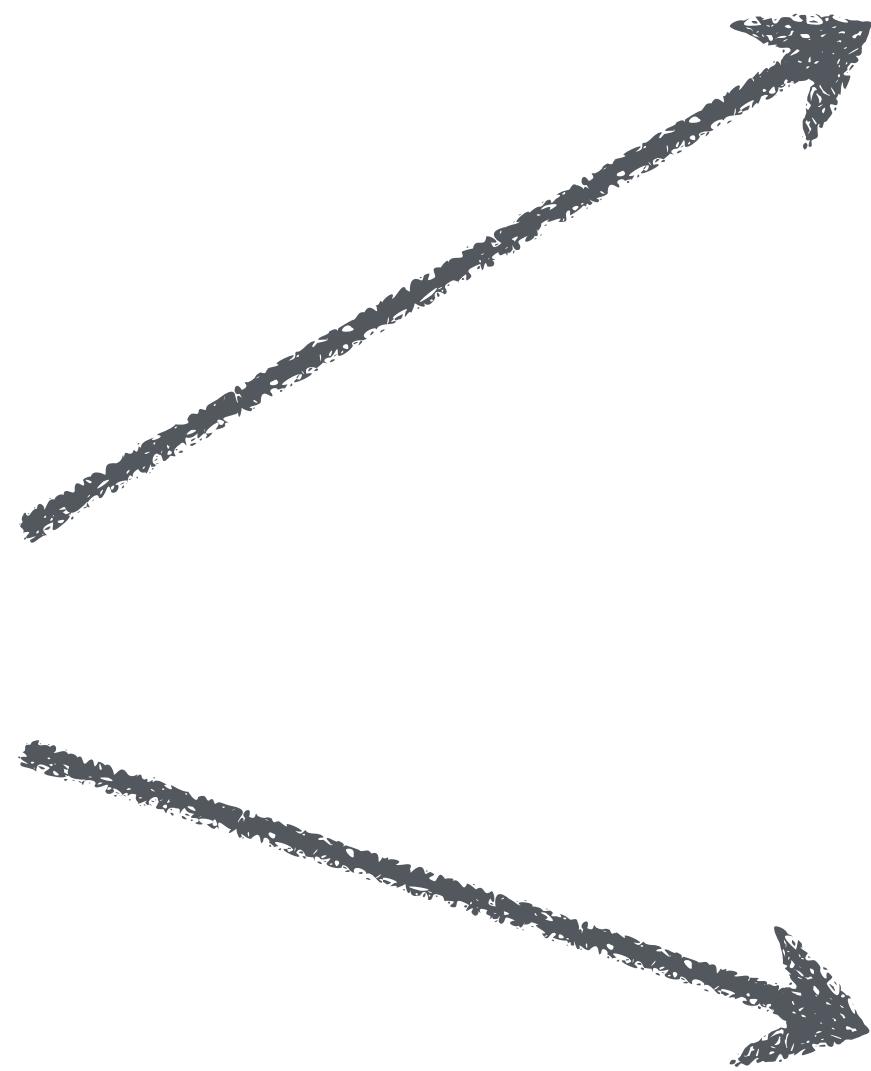
Универсального решения нет –  
все зависит от используемого стека



Rename

## CSS

```
.foo { color: red }  
.foo.bar { color: green }
```



## Результат

```
.a { color: red }  
.a.b { color: green }
```

+

## rename map

```
{  
  "foo": "a",  
  "bar": "b"  
}
```

# Что это дало нашему проекту

- 823 Kb Оригинальный CSS
- 596 Kb Базовая минификация
- 385 Kb Rename (пока вне CSSO)

Улучшение результата на 211Kb (35%)

# Все вместе

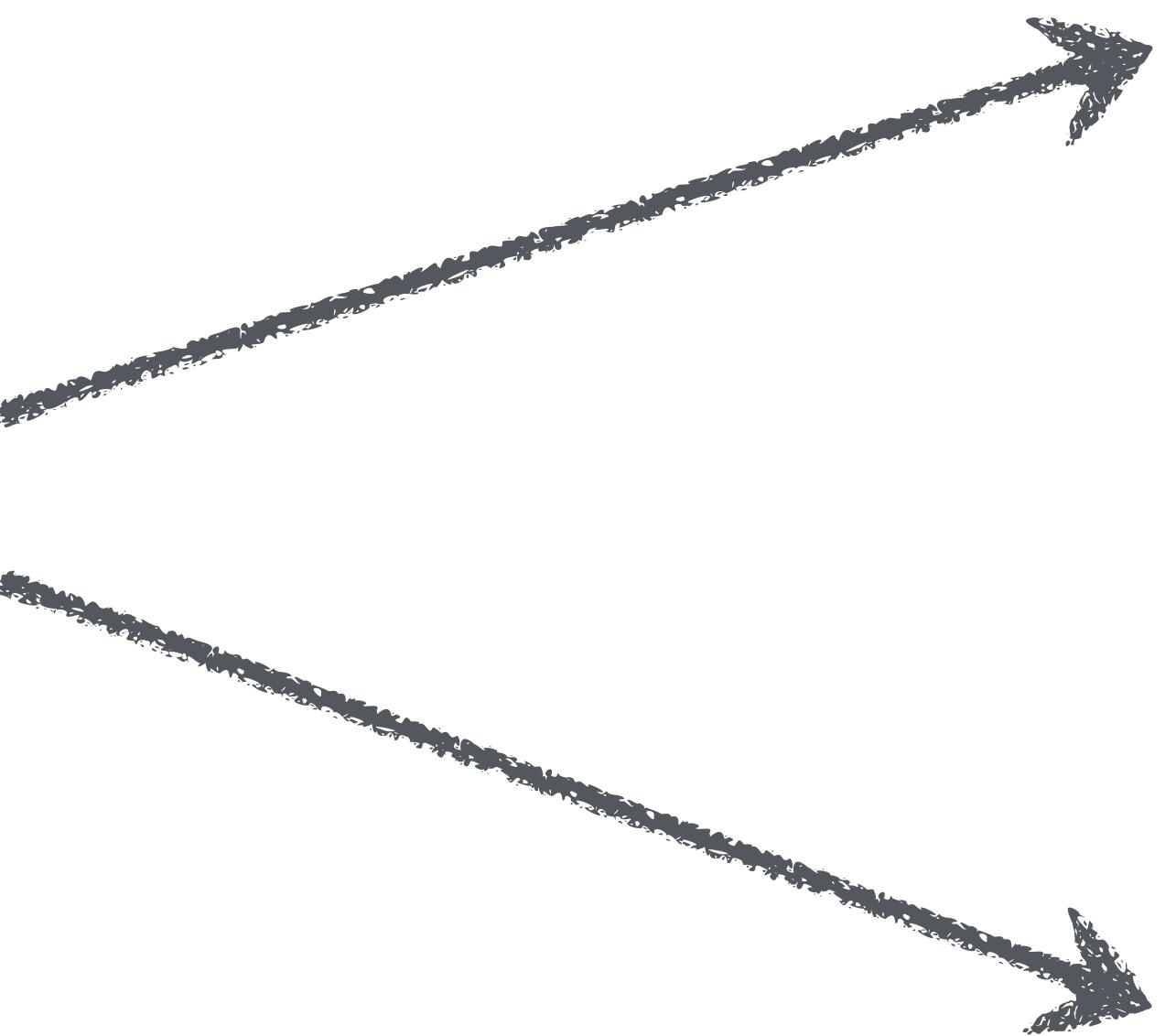
- 823 Kb Оригинальный CSS
- 596 Kb Базовая минификация
- 232 Kb Rename + Usage data

Улучшение результата на 364Kb (61%)

Должно ли это быть  
в минификаторе?

## CSS

```
.foo,  
.bar {  
    color: red;  
}  
  
.foo:hover,  
.bar:hover {  
    color: green  
}
```



## Результат

```
.a { color: red }  
.a:hover { color: green }
```

+

## rename map

```
{  
    "foo": "a",  
    "bar": "a"  
}
```

Кто будет писать такой CSS?

*Usage data!*

# CSS

```
.foo {  
  color: red;  
}  
.foo:hover {  
  color: green  
}  
  
.bar {  
  color: red;  
}  
.bar:hover {  
  color: green  
}
```

+ usage data



# Результат

```
.foo,  
.bar {  
  color: red;  
}  
.foo:hover,  
.bar:hover {  
  color: green  
}
```



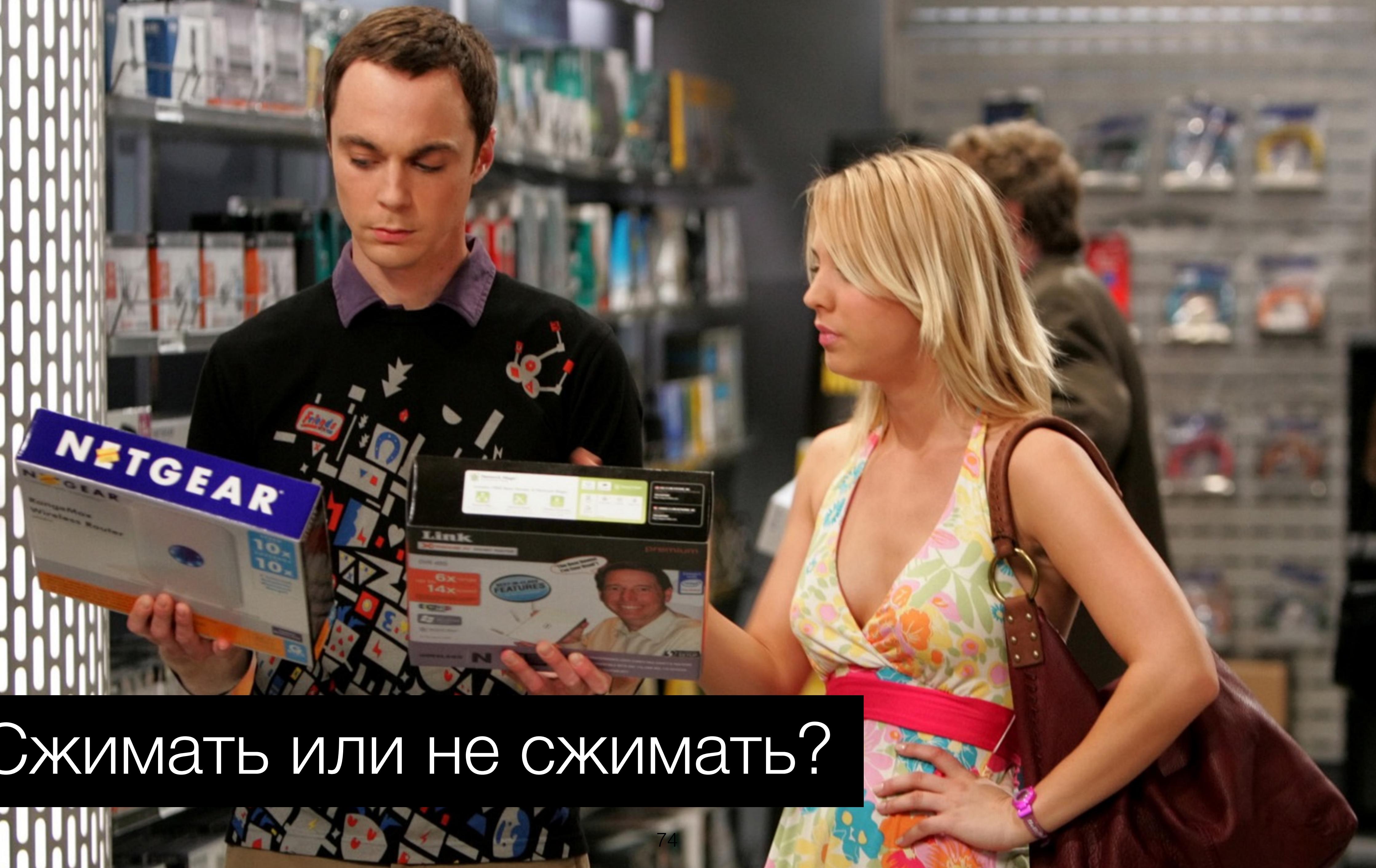
В разработке – скоро в CSSO

# Что это дало нашему проекту

## предварительные оценки

- ~10 Кб дополнительного выигрыша (~3-4%)
- 1431 удаленный селектор из 6904

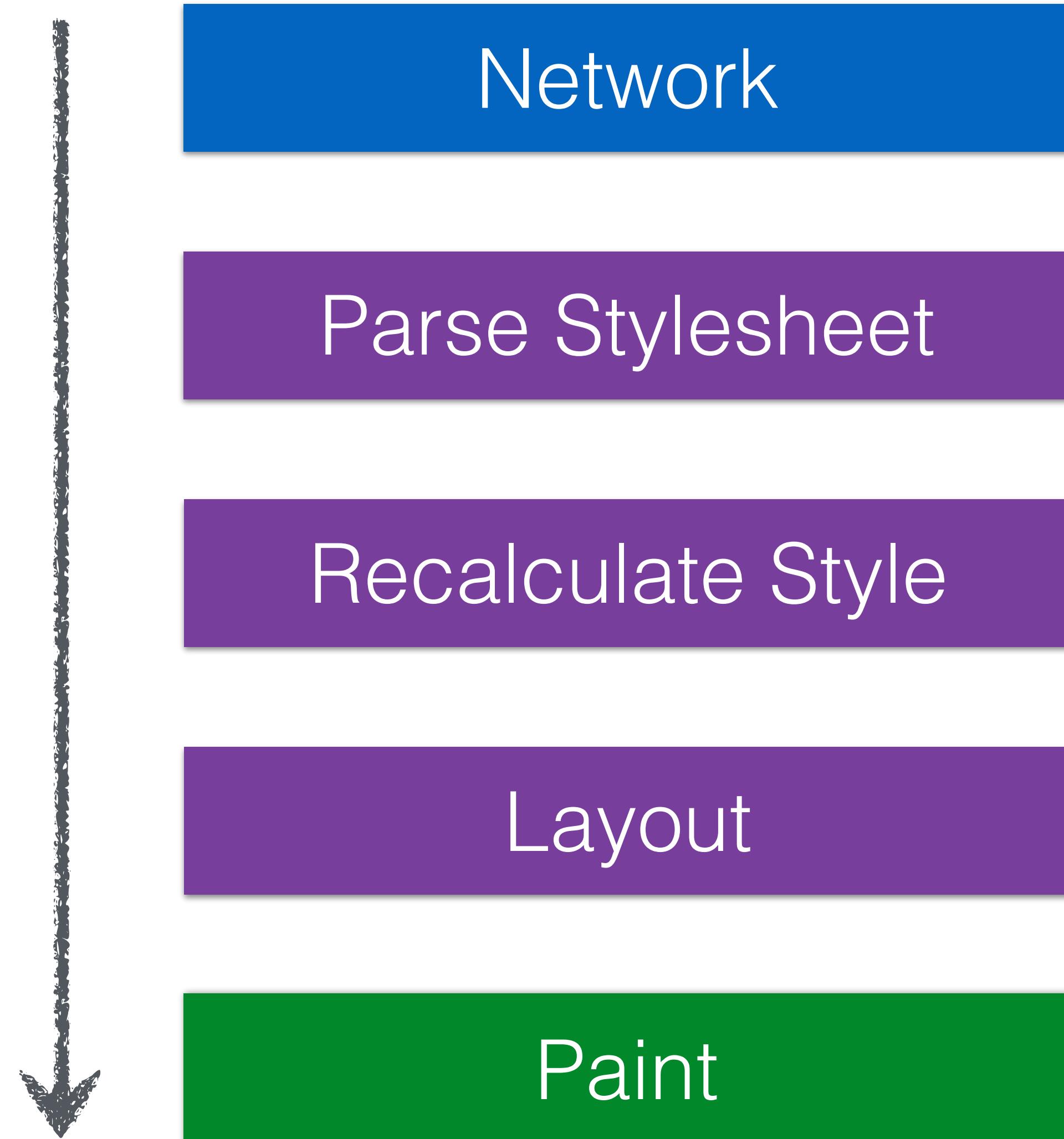
Селекторов стало на ~20% меньше



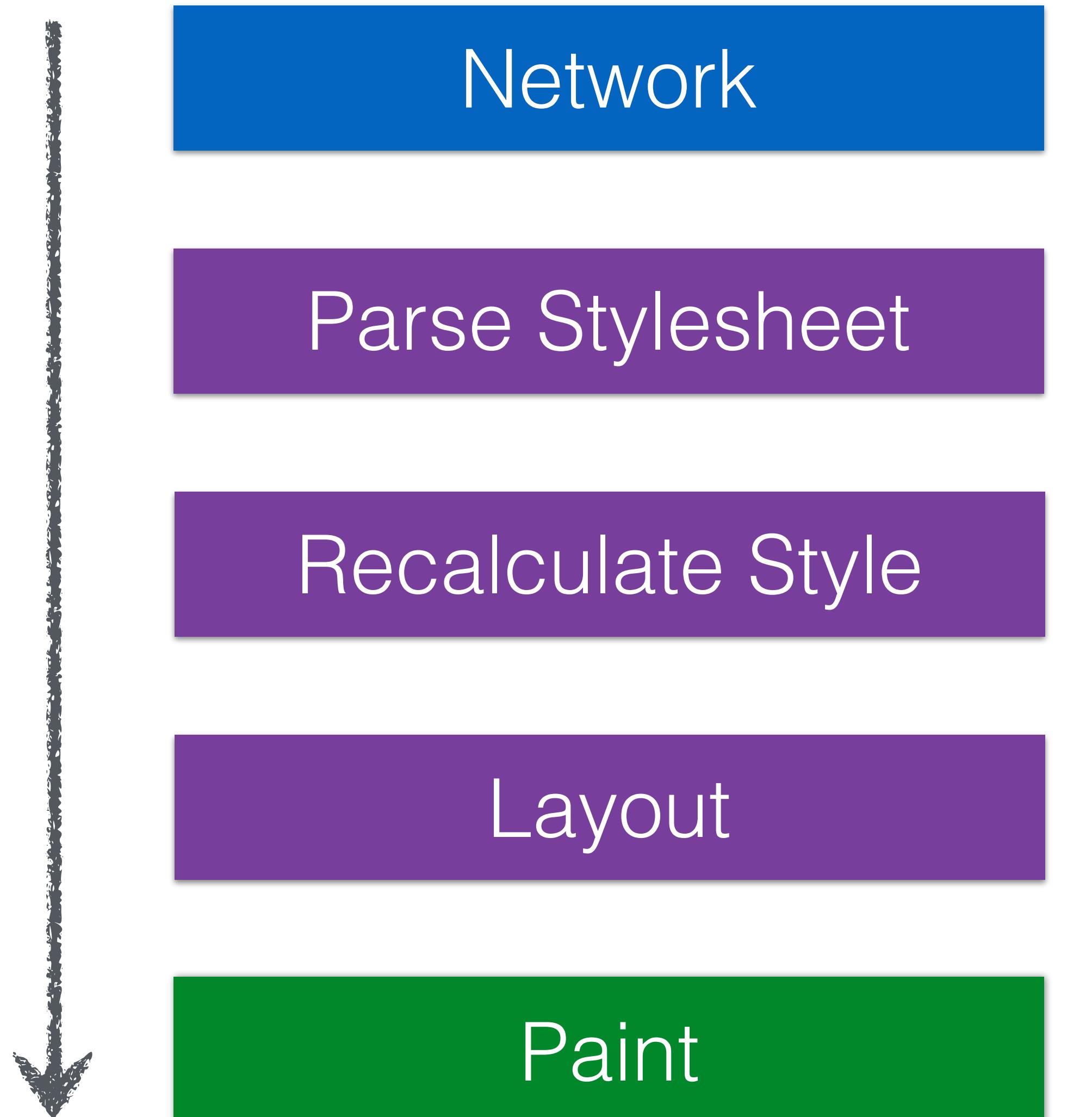
Сжимать или не сжимать?

# На что влияет минификация?

# Как CSS превращается в картинку



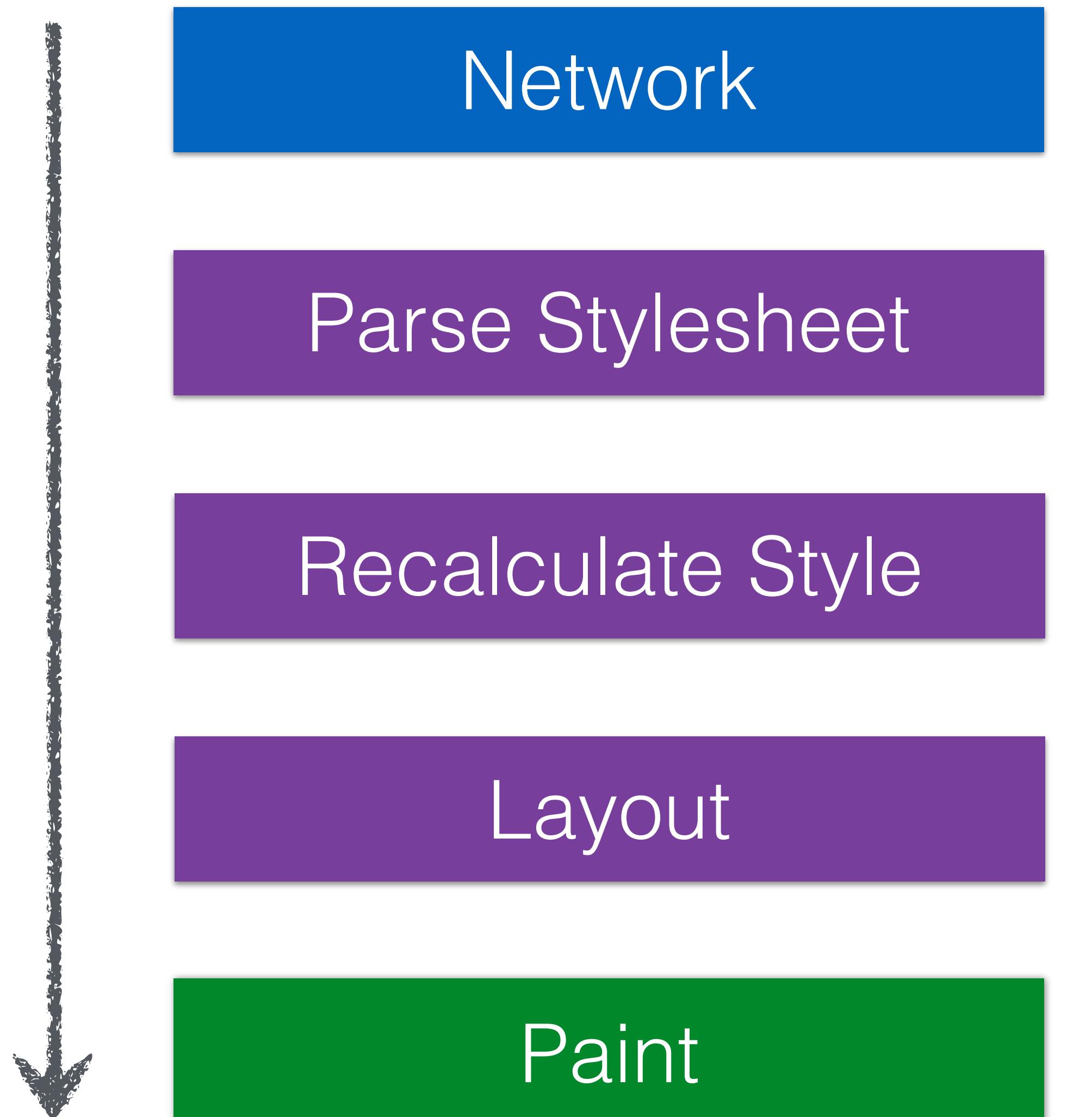
# Влияние характеристик CSS на скорость



Влияют **количественные** характеристики CSS  
(размер, кол-во селекторов и т.д.)

Влияют **качественные** характеристики CSS  
(сложность расчетов и отрисовки)

# Автоматизация улучшений



Компрессия может  
оказать положительный  
эффект

Пока нет предпосылок,  
что задача решается

# Network

Network



Решение: gzip, SDCH ...

Parse Stylesheet

Имеет эффект только  
для холодной загрузки

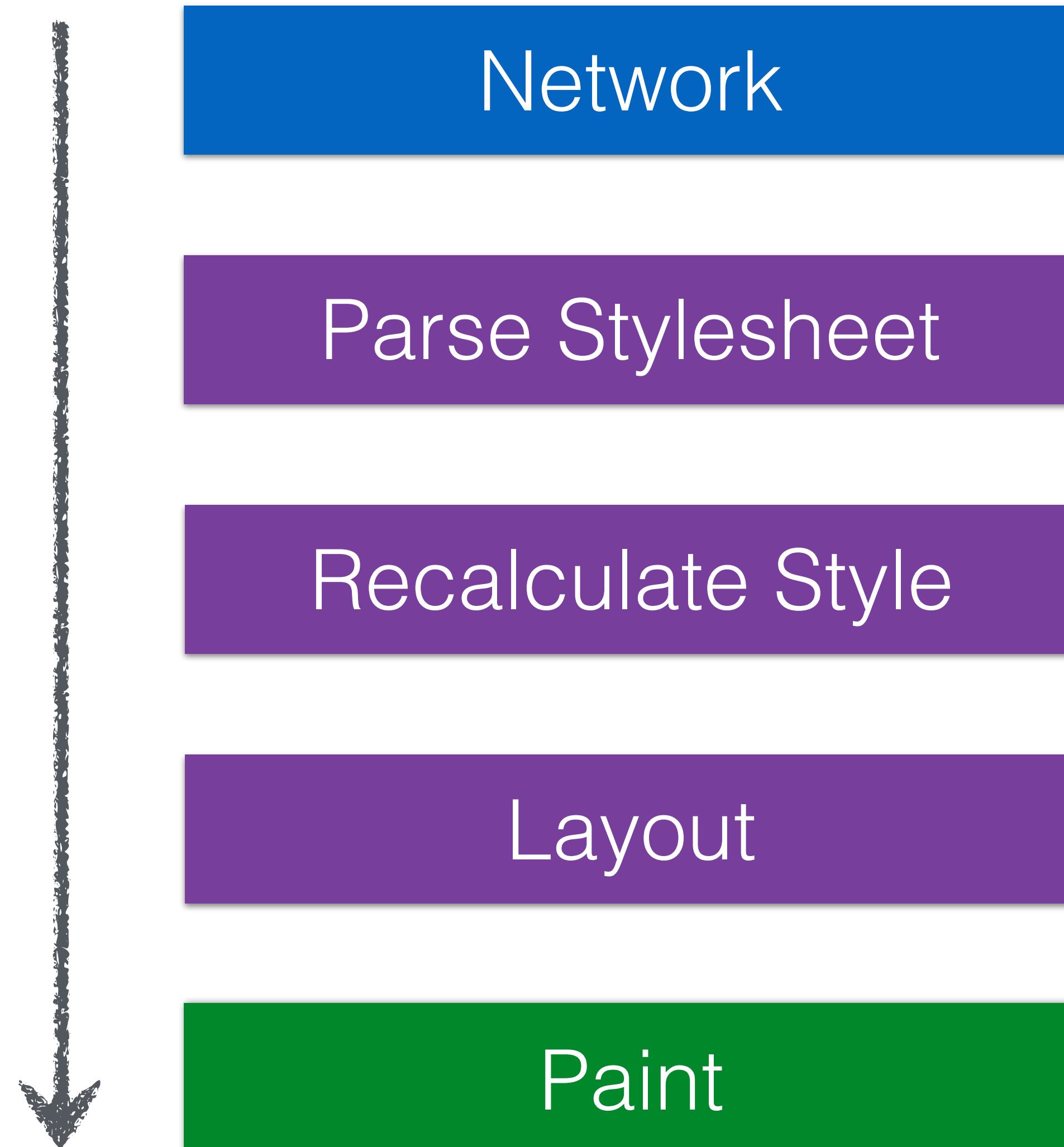
Recalculate Style

Неоптимизированный CSS  
сжимается лучше

Layout

Paint

# Parse Stylesheet



## Решение: сжатие CSS



Тут gzip уже не играет роли, выполняется всегда на старте + мутация DOM

Меньше текста – меньше парсить

# Наколеночные тест

Влияние разного уровня сжатия на время парсинга  
(Chrome на MacBook Air)

Без сжатия 823 Kb – 35ms

Базовое сжатие 596 Kb – 29ms

Rename 385 Kb – 24ms

Rename + usage data 232 Kb – 22ms

Размер уменьшился в ~4 раза, но время лишь на ~50%

## Наколеночные тесты

Ранее, на других устройствах, были получены более обнадеживающие цифры при улучшении сжатия

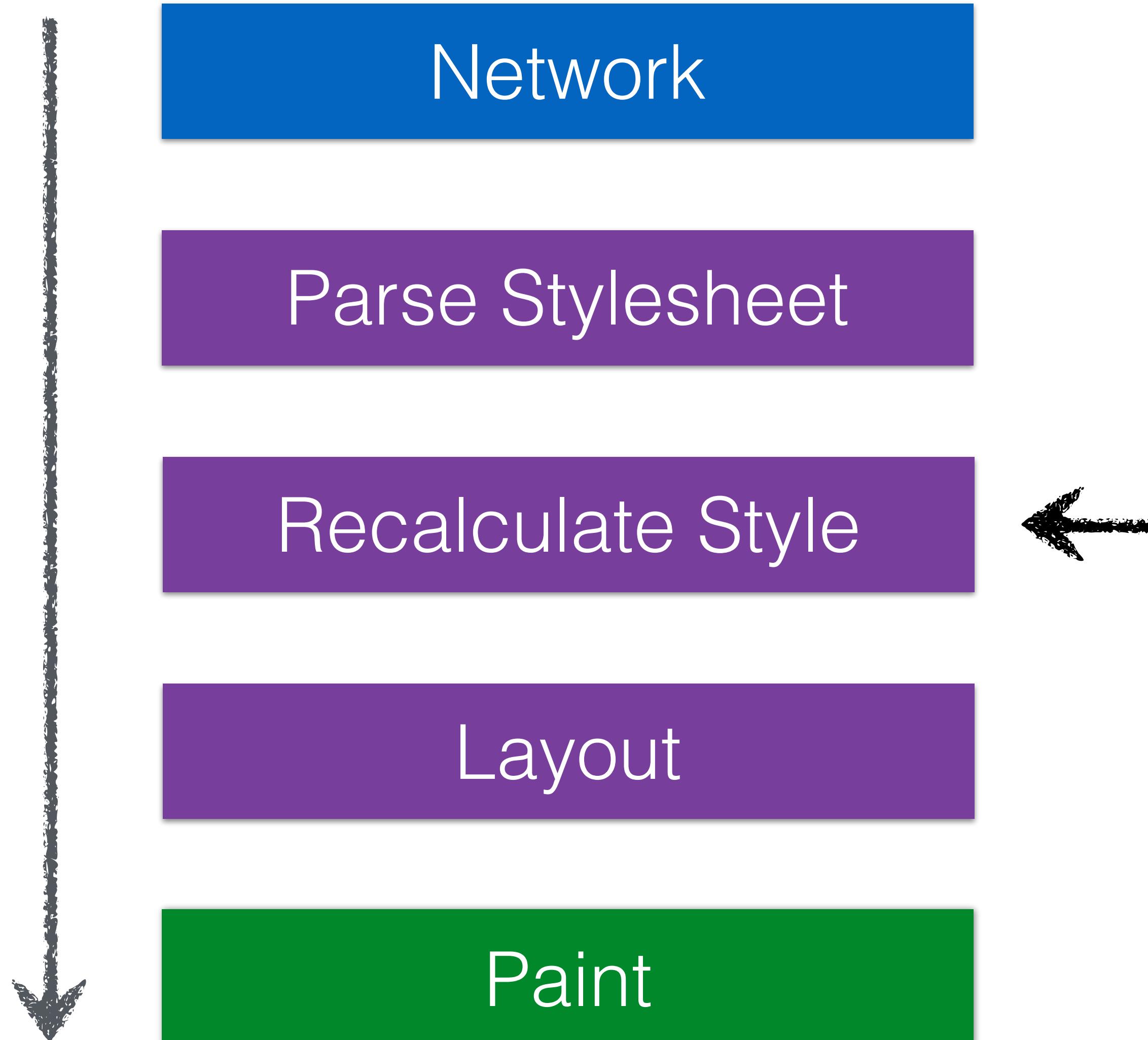
CSS 316Kb ~~+ usage data~~ → 215Kb (-39.5%)

Win10 Desktop 19ms → 11ms

Nexus 5X 68ms → 44ms

Samsung Galaxy Note 2 158ms → 108ms

# Parse Stylesheet



Решение: [rename](#) и др.

Уменьшение кол-ва  
селекторов, их сложности



Пока гипотезы, цифр нет,  
но будут как только фича  
появится в CSSO ;)

# Сжимать или не сжимать?

# Сжимать или не сжимать?

Да! Хуже не будет

Хотя эффект – предмет для исследований

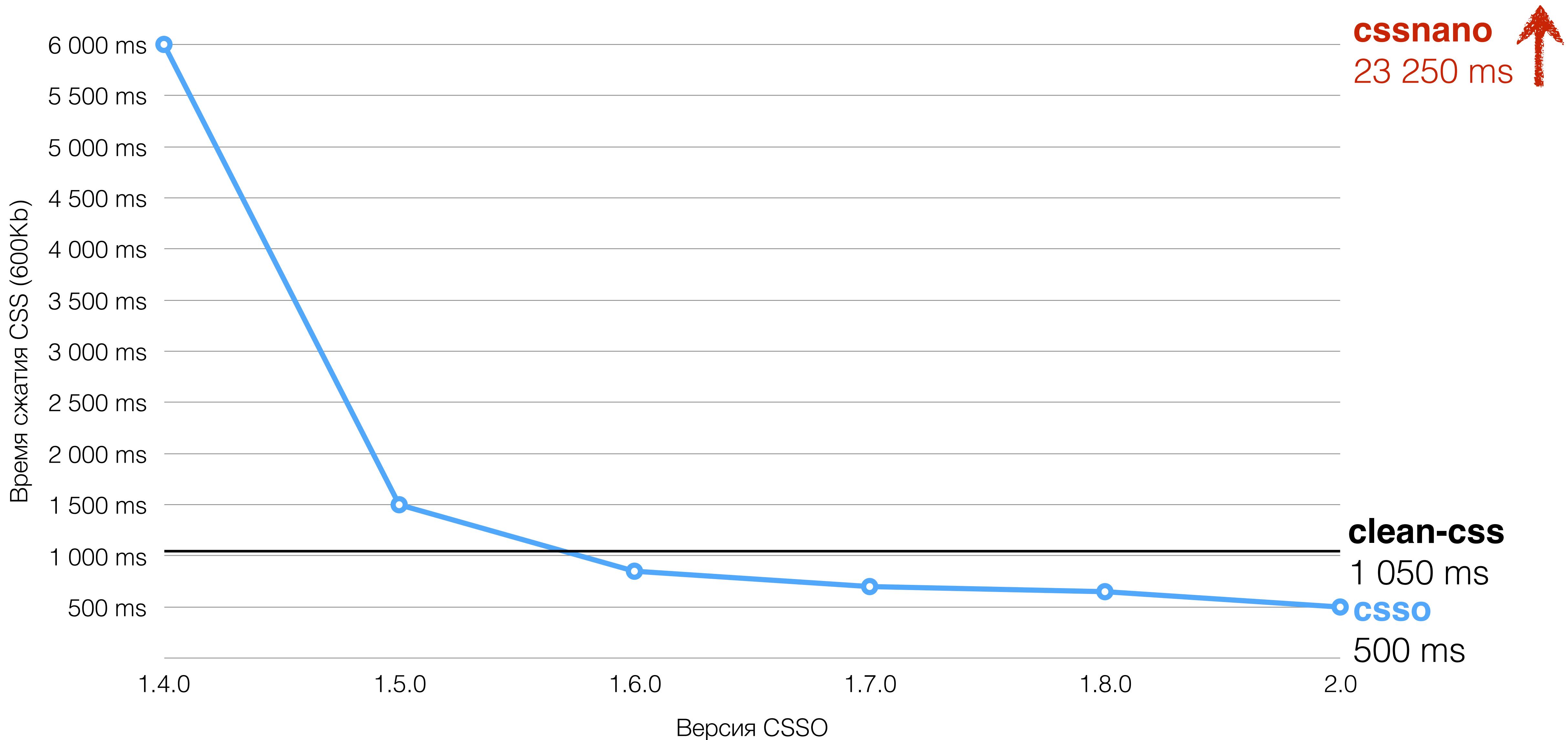


CSSO – новая жизнь

# Что изменилось

- В 10+ раз быстрее
- В 8+ раз меньше потребление памяти
- Исправлена большая часть проблем и багов
- Улучшена кодовая база и API
- Больше скачиваний и звезд на GitHub ;)

# Изменение по скорости



# postcss-css

Плагин для PostCSS, альтернатива cssnano

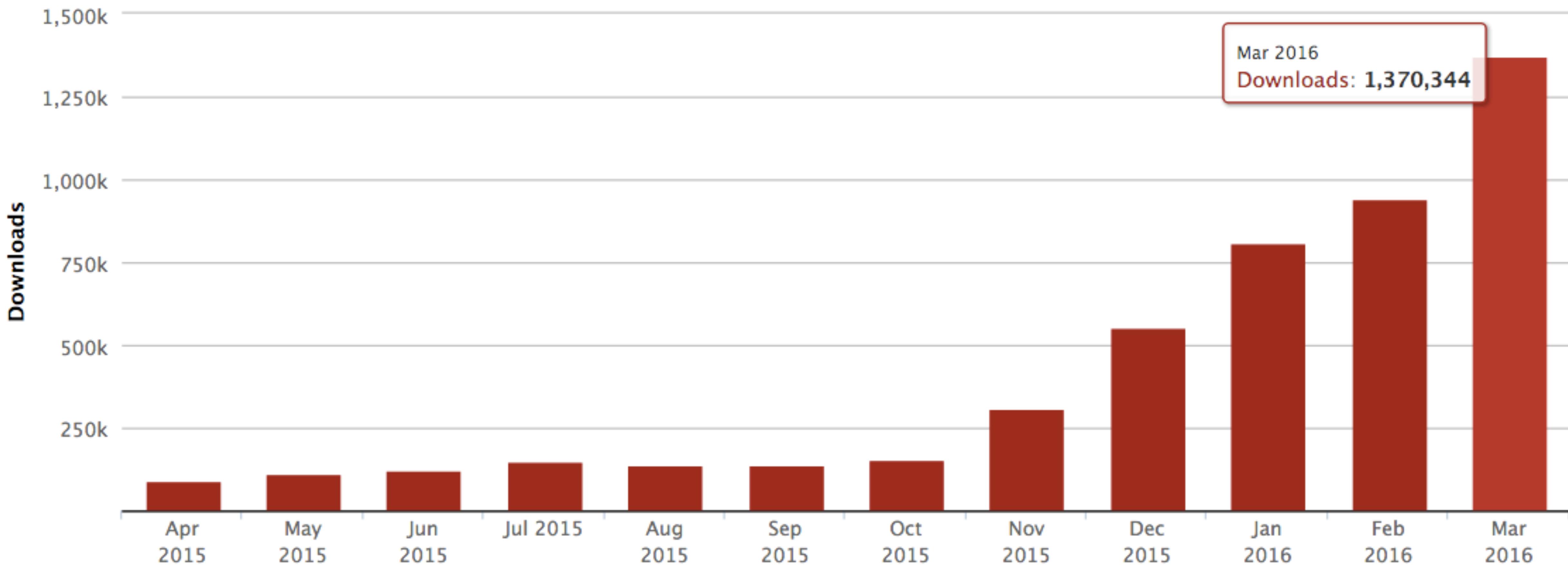
Работает почти также быстро как CSSO отдельно

Под капотом конвертация AST

[github.com/lahmatiy/postcss-css](https://github.com/lahmatiy/postcss-css)

# 1 300 000+ скачиваний в месяц

х9 с октября 2015



# Новое

- Source Maps
- Usage data
- Лучше поддержка "новых" частей в CSS
- Лучше сообщения об ошибках
- Поддержка stdin
- Новый формат AST

Планы





Главная цель – лучший минификатор

# Coming soon

- Новые оптимизации и алгоритмы: быстрее и правильно
- Учет поддерживаемых браузеров
- Семейства свойств и сортировка деклараций
- Нормализация имен и переименование
- Понимание структуры shorthand-значений
- Применение статистики

В заключении





Любите CSS, читайте спеки



Используйте CSSO :)



Все новое в Твиттере [@cssoptimizer](https://twitter.com/cssoptimizer)

Вопросы?



Роман Дворнов

@rdvornov

[github.com/css/csso](https://github.com/css/csso)

rdvornov@gmail.com