Web Standards Days

Не бойся, это всего лишь данные... просто их много

Роман Дворнов

Ostrovok.ru

Новосибирск
Апрель 2014

О себе

- Работаю в Ostrovok.ru
- Автор фреймворка basis.js

Данные

Когда говорят про
данные на client-side,
чаще всего, подразумевают
объекты и их наборы
(модели и коллекции)

Сегодня есть опыт,
разнообразные библиотеки,
быстрые браузеры,
мощные API...

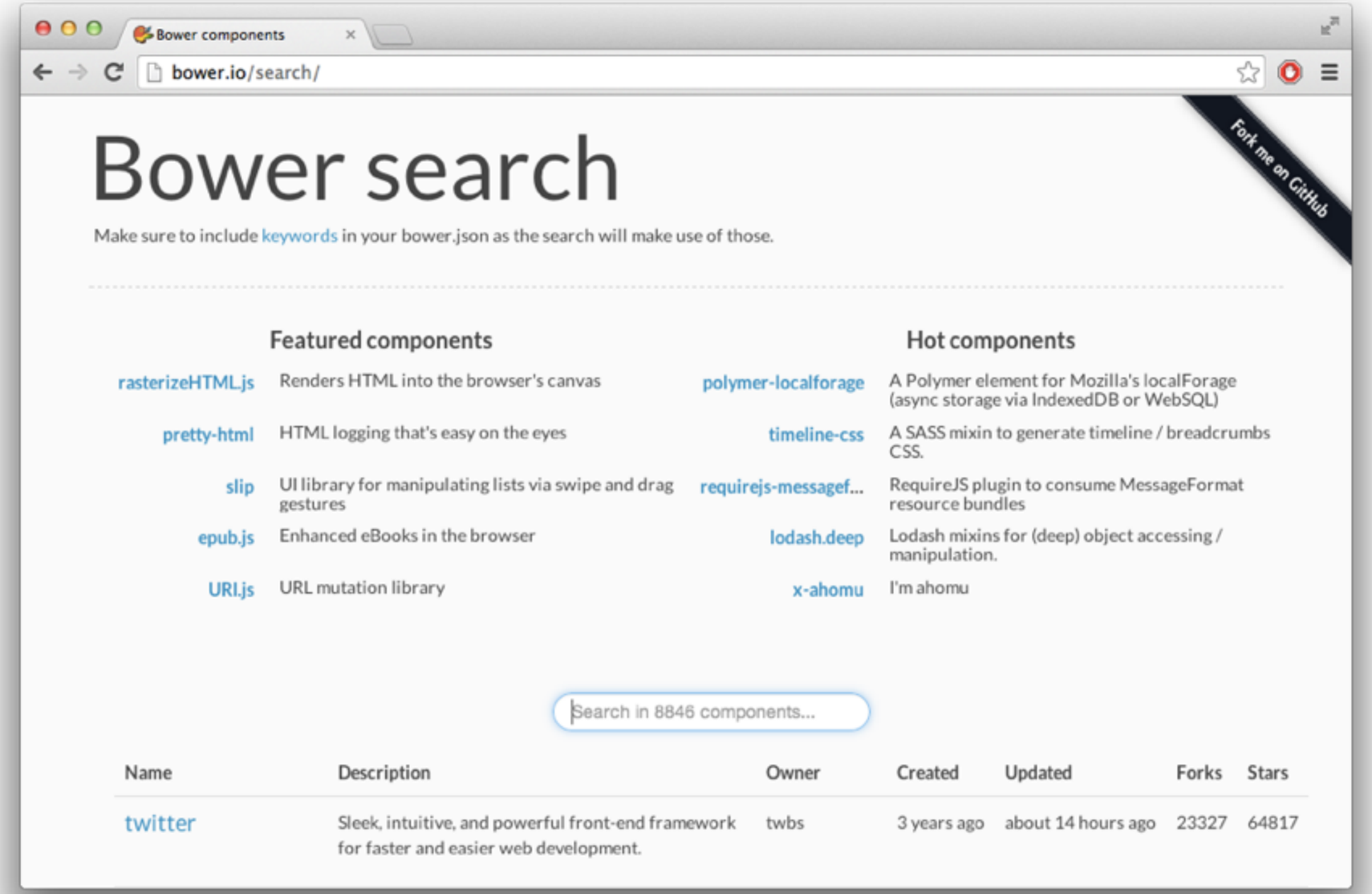
... но многие по-прежнему скептически относятся к сложным вычислениям и системам на client-side

Client-side глазами скептиков...

Все ли так плохо?

Хороший «плохой» пример

Или как заставить браузер страдать



bower.io/search

~10 000 модулей

~2,5 Мб JSON

~10 000 модулей

~2,5 Мб JSON

~8 сек браузер "висит"!

AAaaa!!!!.. Все пропало!...

Нужно грузить постранично...

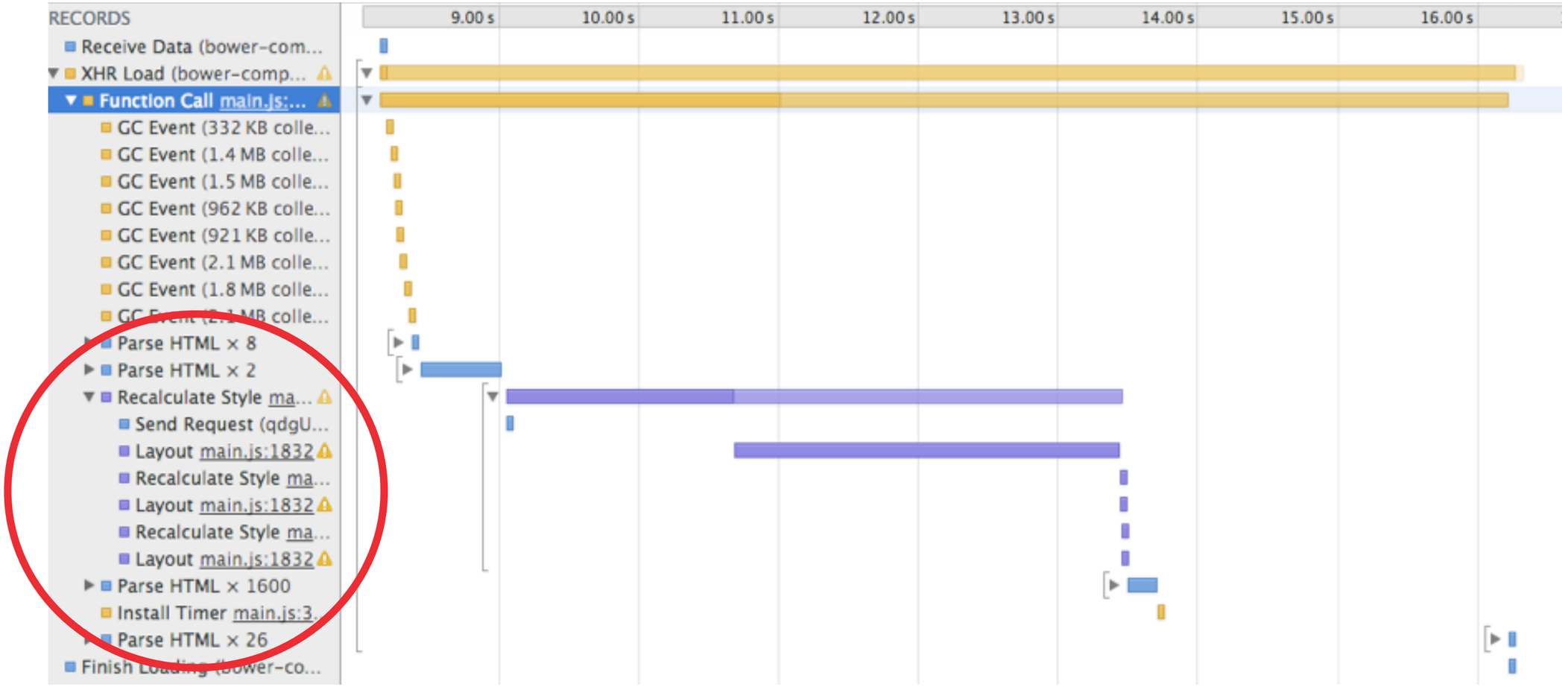
Постойте, не так быстро...

Почему?

Что делает скрипт

- загружает JSON (массив объектов)
- генерирует 4 небольших списка
- генерирует таблицу на ~10 000 строк
- отдает таблицу компоненту, который разбивает её на страницы и удаляет все строки, кроме первых десяти

Timeline – Что делает браузер



Что делает браузер

- javascript ~2,85 сек
- парсиг html (из шаблонов) ~0,75 сек
- расчет стилей ~1,6 сек
- layout (~200 000 узлов) ~2.8 сек
- *еще несколько секунд асинхронно (по 100) "индексируются" строки для поиска*

А можно быстрее?

Можно!

github.com/lahmatiy/bower-search

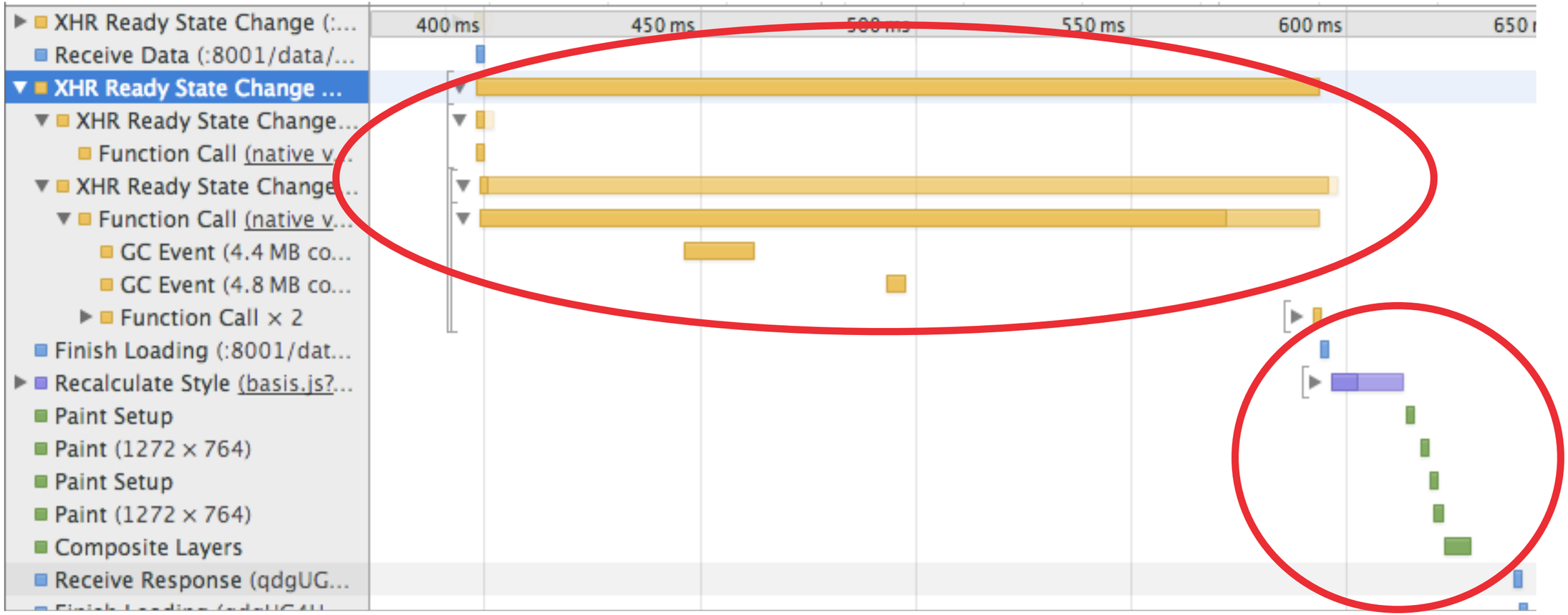
lahmatiy.github.io/bower-search

- генерация представлений ~0,04 сек
- применение данных ~0,15* сек

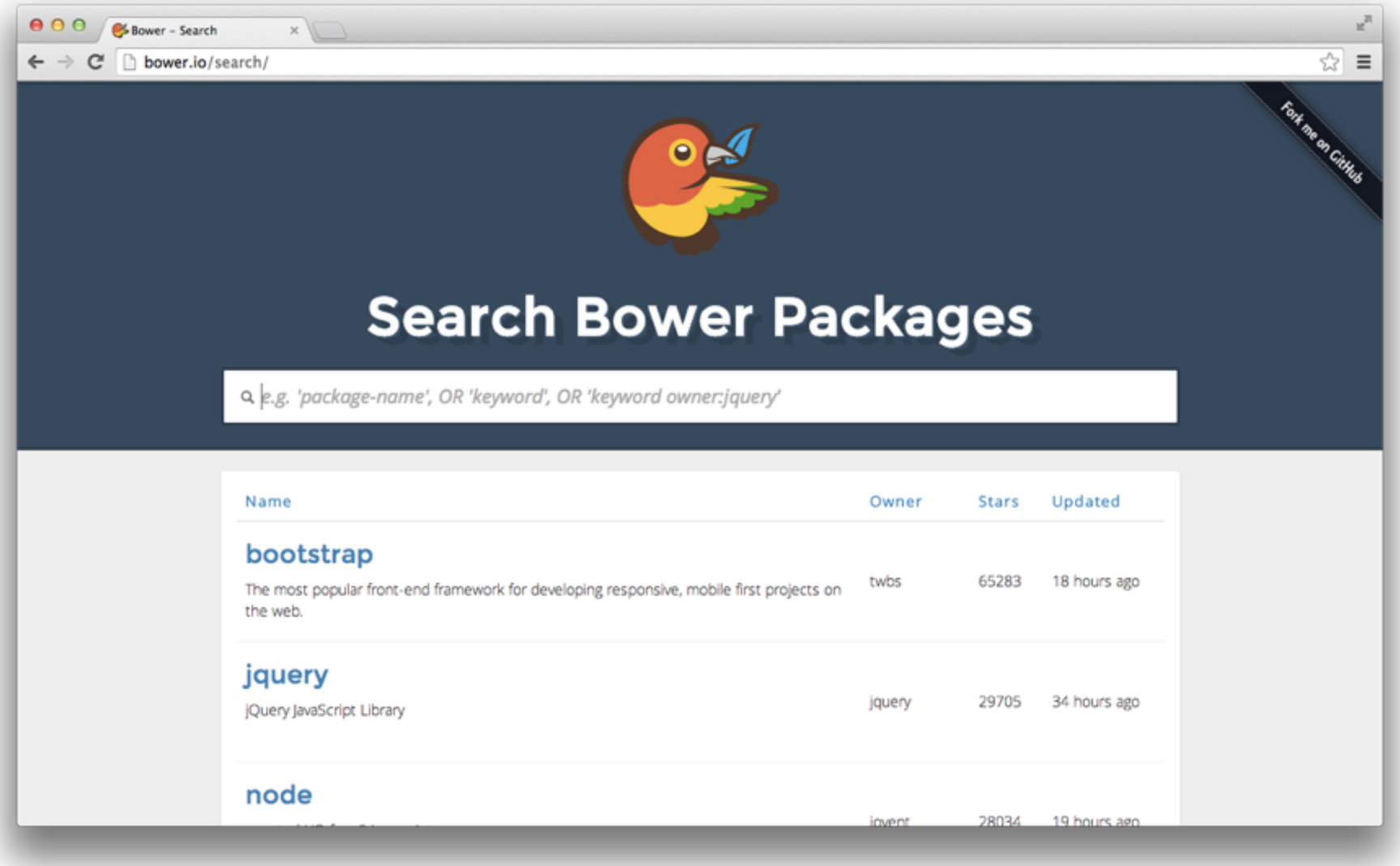
* и можно быстрее

Timeline – Применение данных

только JavaScript – ничего лишнего



Чак одобряет



Хорошие новости:

Переделали – теперь работает быстро

Вывод:

Дело не в количестве данных,
а в неэффективном подходе

Проблема
современного фронтенда

Многие веб-разработчики думают через призму jQuery-like подходов, размышляют про данные через представление (верстку)

Пара примеров

По мотивам bower.io/search

Разметка ради разметки

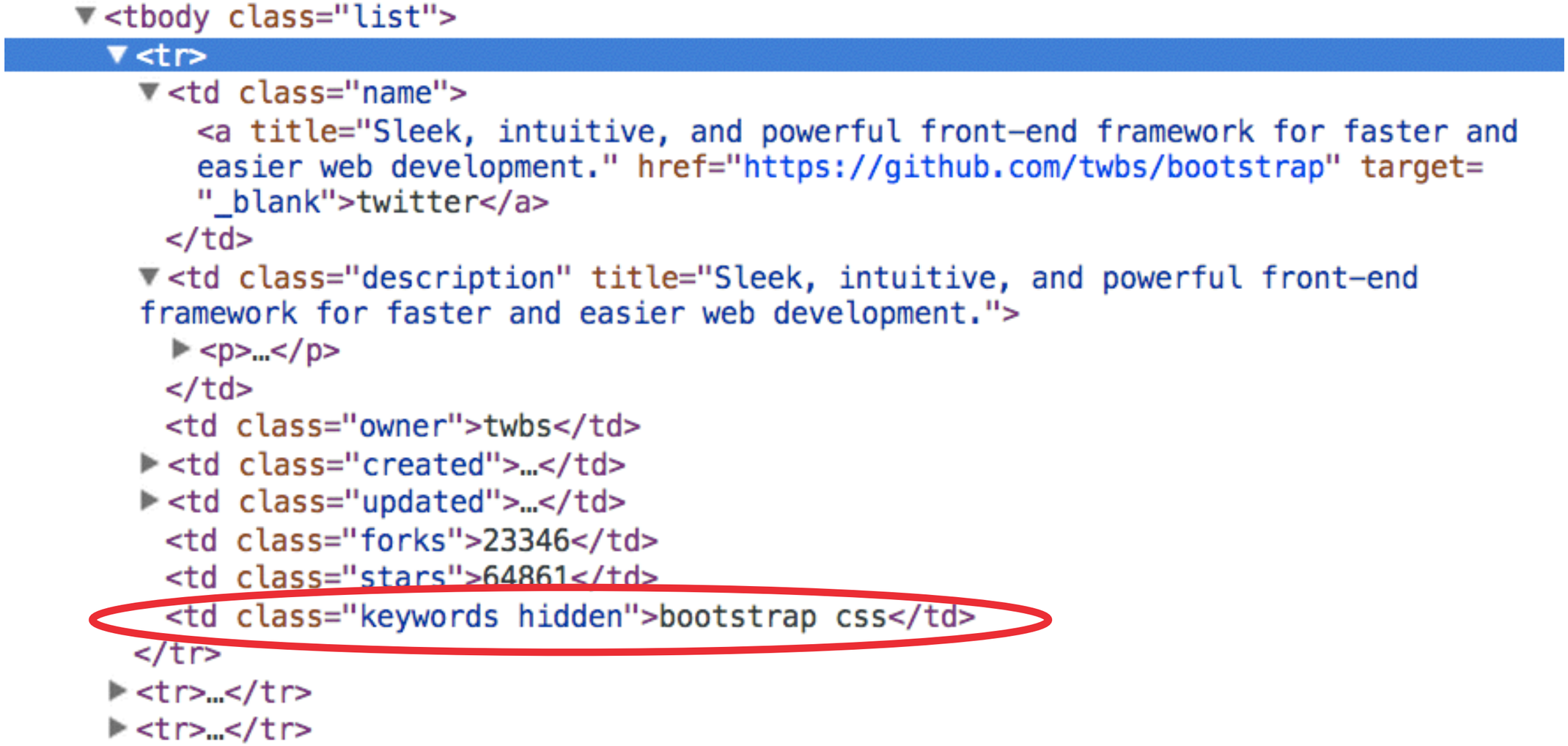
```
▼<td class="created">
   <time datetime="2011-07-29T21:19:00Z">2011-07-29T21:19:00Z</time>
 </td>
```

+

```
$('#components')
  .find('.created time, .updated time')
  .timeago();
```

=

```
▼<td class="created">
  ▼<time datetime="2011-07-29T21:19:00Z" title="2011-07-29T21:19:00Z">
     "3 years ago"
   </time>
 </td>
```



Невидимая ячейка,

ее innerHTML используется для поиска

Получение значений для поиска

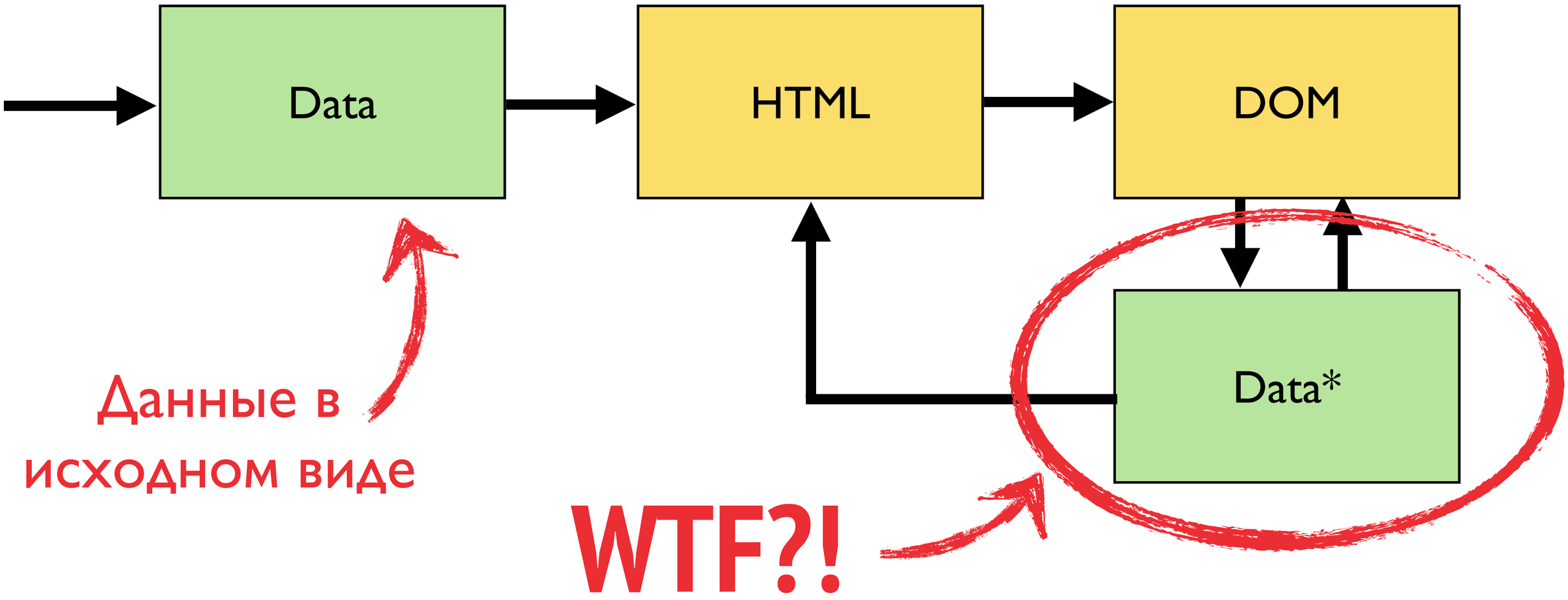
```
Item.prototype.values = function(tr, columns) {
  var values = {};

  for (var i = 0; i < columns.length; i++) {
    var name = columns[i];
    var td = tr.getElementsByClassName(name)[0];

    values[name] = td ? td.innerHTML : '';
  }

  return values;
};
```

В общем случае



Фактически, DOM используется как хранилище данных

И это самое медленное,
что можно придумать

Цифры говорят сами за себя

8,0 с **vs.** 0,2 с

40 : 1

Не используйте

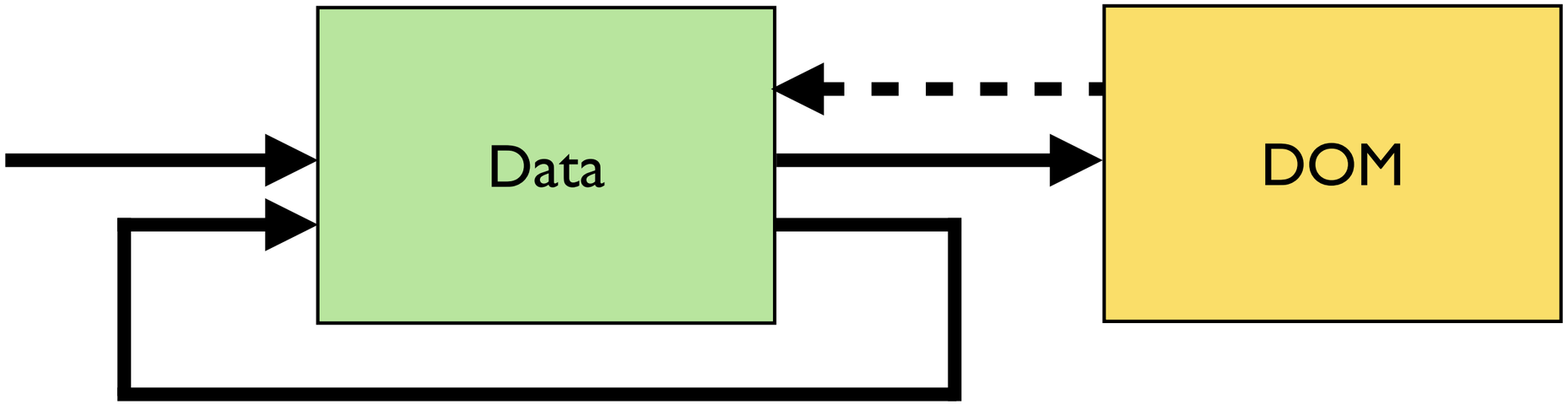
DOM для хранения данных!

А как надо-то?

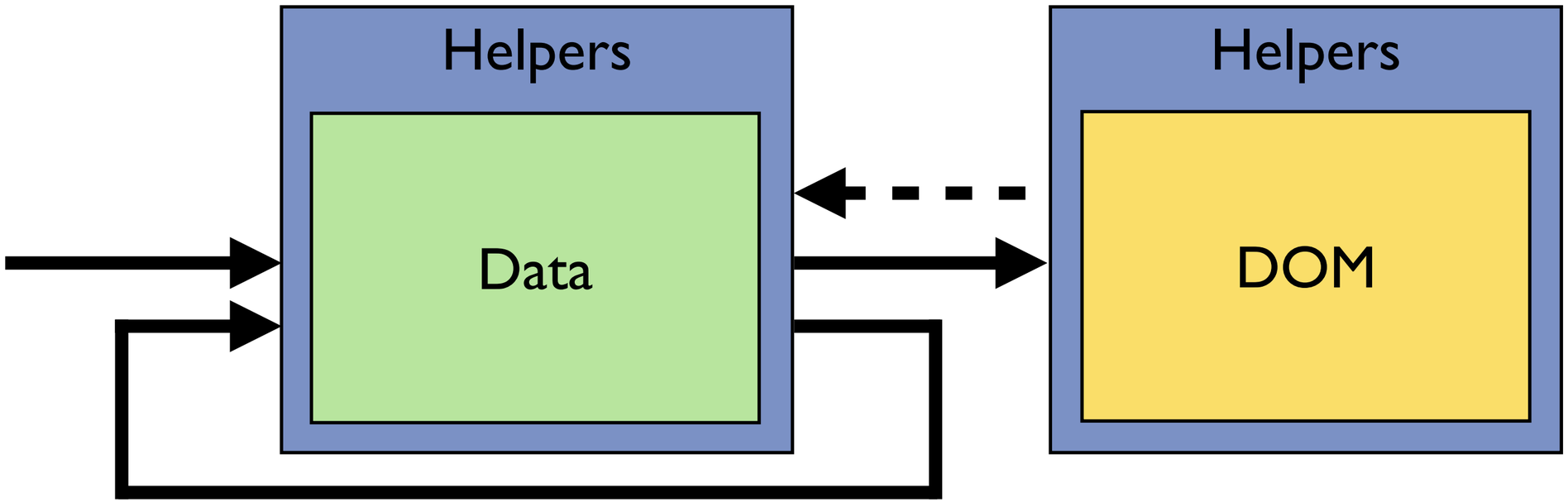
Работа с данными –
вне представлений, в
абстрактном слое

Задача представлений –
отражать состояние данных

Идеальный вариант



Идеальный вариант + хелперы



Model
Collection
...

View
bindings
...

Хелперы упрощают разработку,
но добавляют overhead

Быстрая работа с DOM

Наиболее перспективное направление
DOM-based templates

Фреймворки: React, Ractive, Meteor, Basis.js...

Доклад «Как построить DOM»

tinyurl.com/build-dom

Быстрая работа с данными

Практически никто об этом не думает :(

Якобы основная проблема –
рендеринг

Модель ≠ представление

Данных может быть
гораздо больше,
чем представлений

bower.io/search

10 000+ модулей

кастомные выборки, мгновенная фильтрация и сортировка

без участия сервера

Демо блог – 5 000 постов

показываем
постранично

Имея все посты
генерируем
облако тегов
и архив
на клиенте

tinyurl.com/basis-blog

Каковы пределы?

С каким количеством моделей
мы можем работать?

1 000

10 000

100 000

1 000 000

???

Конечные ресурсы:
память и время

Память

Все чего-то стоит

- массив: 16bytes
  - +12bytes + 4bytes/item (для литералов)
  - +72bytes + 4-8bytes/item (если растить)
- объект: 12bytes + 4bytes/property
- замыкание: 36bytes
- контекст: 24bytes + 4bytes/var

* стоимость в V8 (Google Chrome)

Данные

bower.io/search

File

~2,5 Мб

Object

~9,0 Мб

Создание моделей

10 000 экземпляров, Кб

Решение	0 полей	10 полей
new basis.data.Object()	240	240
basis.entity.Type()	440	840
new Backbone.Model()	920	1 480
Ember.Object.create()	1 040	1 600

Разные задачи, разные решения

basis.data.Object

дешево и сердито

- Произвольные поля

basis.entity.Entity

дороже, но с плюшками

- Строгий набор полей
- Вычисляемые поля
- Индекс
- Нормализация значений
- Defaults
- Rollback
- ...

Event listeners

10 000 экземпляров, Кб

Фреймворк	1 событие	2 события	3 события
Basis	240	240	240
Backbone	1 520	2 860	3 840
Ember	5 480	6 520	7 560

Итого

10 000 экземпляров, Кб

Решение	10 полей, 1 listener	overhead
basis.data	480	5 %
basis.entity	1 080	12 %
Backbone	3 000	33 %
Ember	7 080	79 %

А если пойти дальше...

Интерполяция

10 полей и 1 listener, Мб

Решение	1 000	10 000	100 000
basis.data	0, 05	0,5	5
basis.entity	0,1	1	10
Backbone	0,3	3	30
Ember	0,7	7	70

Вывод №1

О памяти, можно не заботиться когда меньше 10 000 моделей

Вывод №2

Но при больших количествах объектов, расход памяти является серьезной проблемой

Меньше overhead –
больше полезной нагрузки

Время

Парсинг данных

bower.io/search

JSON.parse

~22 мс

Создание моделей

10 000 экземпляров, мс

Фреймворк	0 полей	3 поля	10 полей	20 полей
basis.data	2	2	2	2
basis.entity	~~22~~ 6	~~36~~ 14	~~105~~ 22	~~183~~ 35
Backbone.Model	66	123	238	489
Ember.Object	73	128	201	355

Event listeners

10 000 экземпляров, мс

Решение	1 событие	2 события	3 события
Basis	~ 0	~ 0	~ 0
Backbone	20	29	38
Ember	49	68	89

Итого

10 000 экземпляров, мс

Решение	10 полей, 1 listener	overhead
basis.data	2	9 %
basis.entity	22	100 %
Backbone	248	1127 %
Ember	250	1136 %

А если пойти дальше...

Интерполяция

10 полей и 1 listener, мс

Решение	1 000	10 000	100 000
basis.data	~ 0	2	20
basis.entity	2	22	220
Backbone	25	248	2 480
Ember	25	250	2 500

Это базовое время –
быстрее не поедет

Вывод №3

Задачи можно решать по-разному, но не все решения хорошо масштабируются

Вывод №4

Возможно создавать быстрые и дешевые интерфейсы к данным

Зачем нужны такие оценки?

Чтобы понимать с каким количеством мы можем работать, не причиняя браузеру страдания

Есть задачи, где требуется работать с большим количеством моделей

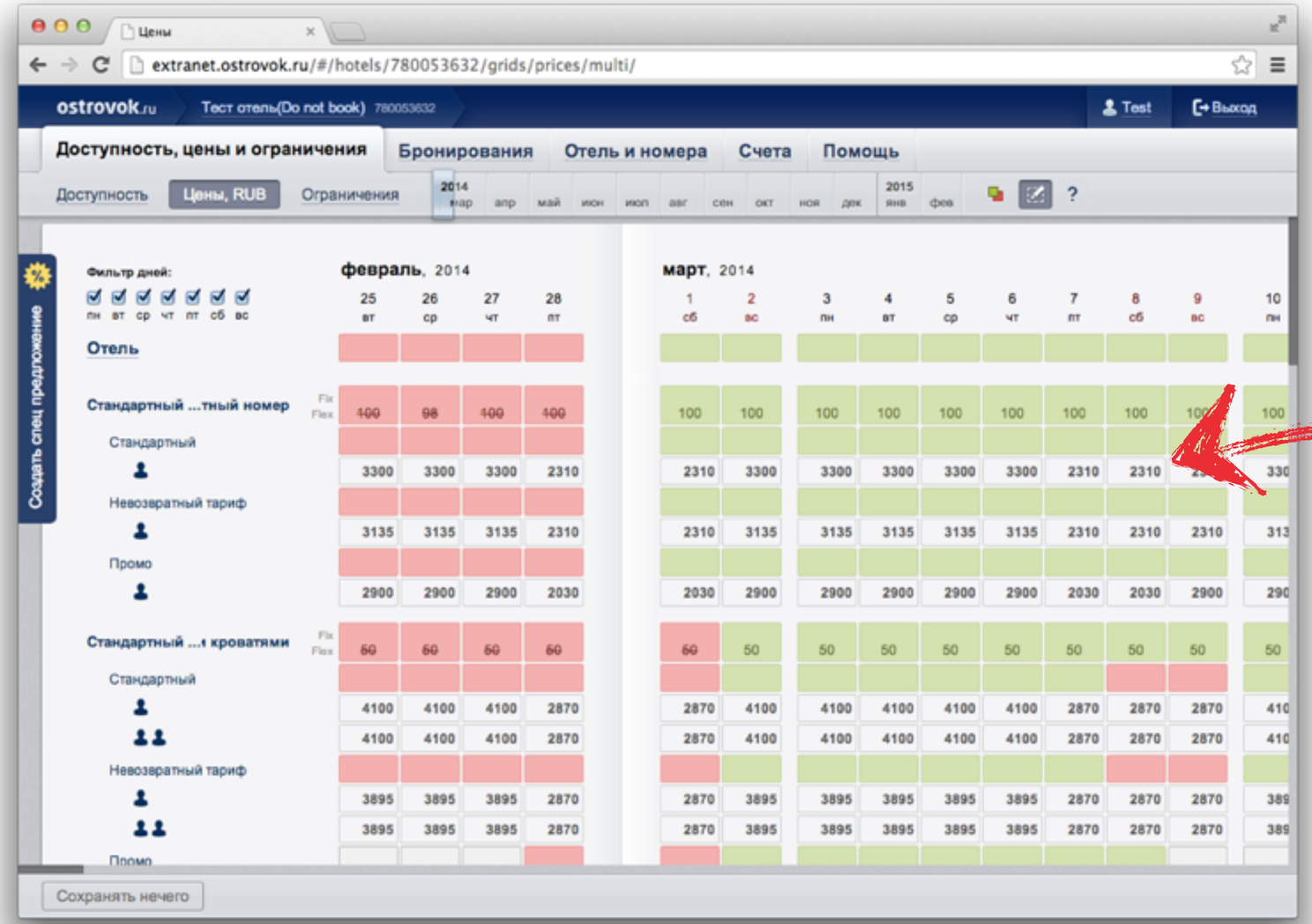
Пример:

Rates & Availability View

Rates & Availability View

Кастомизированный Excel со сложной логикой, специальными функциями и возможностью редактирования

Клиент для отелей



Rates & Availability View

Масштабы бедствия

Структура отеля

до 300 строк

(и растет)

×

Даты

до 730 колонок

(1-2 года)

до 219 000 ячеек

Как это работает сейчас

- По структуре отеля генерируем HTML, вставляем в документ
- document.getElementById() → элемент ячейки
- Запрашиваем данные – месяц/запрос
- Данные – транформируем, создаем модели, настраиваем связи с другими моделями
- Добавляем интерактив – обработчики событий и

Знакомо, не правда ли?

В цифрах

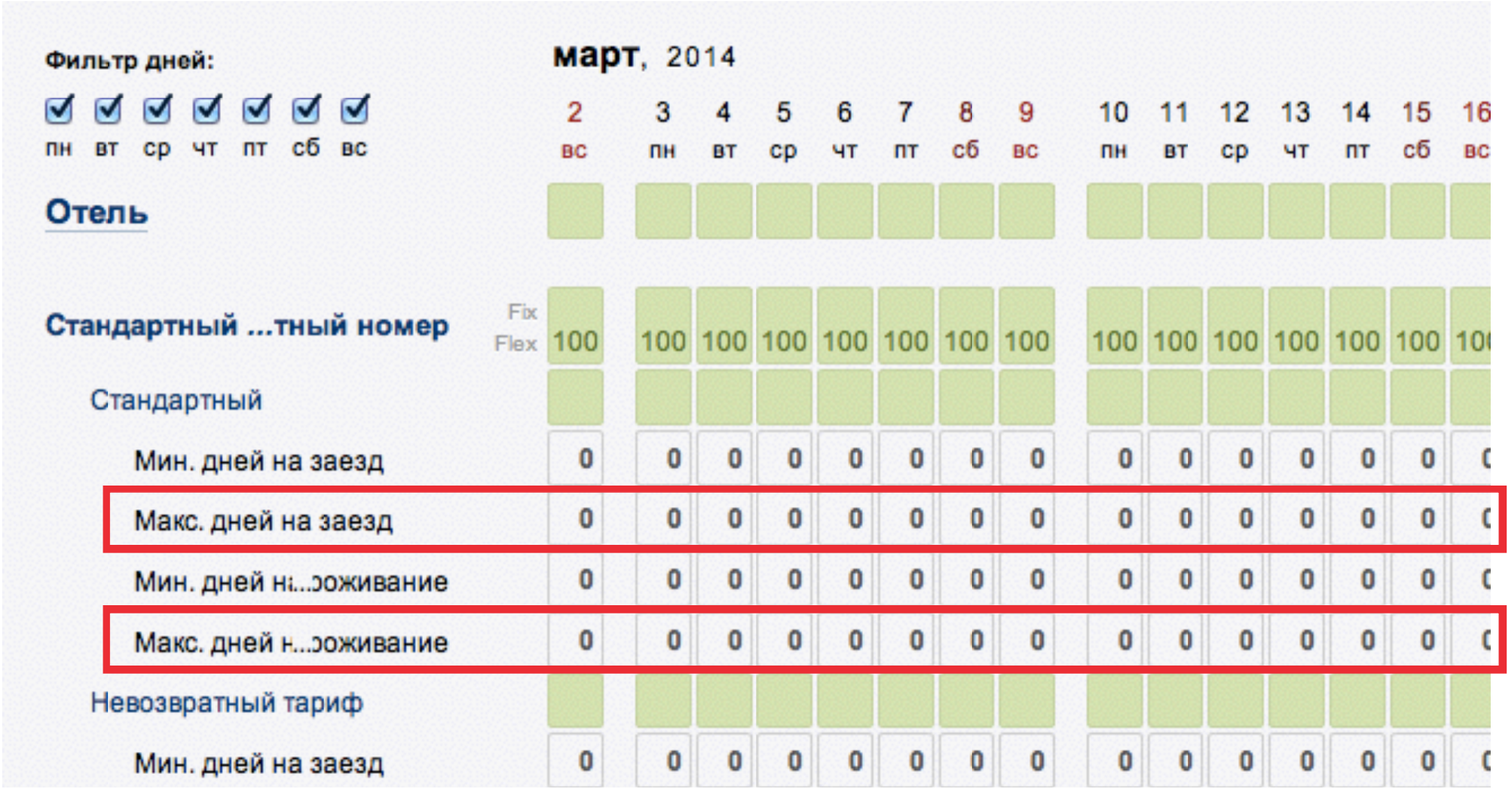
- ~5 000 строк vanilla JavaScript, минимум jQuery
- ~3,5 с чистое время генерации таблицы (в худшем случае десятки секунд)
- ~50 Мб памяти (в худшем случае 100-300 Мб)

Одна из быстрых реализаций с таким ПОДХОДОМ

Время и память
зависят от размеров
и количества данных

Новый функционал

Например, не так давно +2 строки/тариф



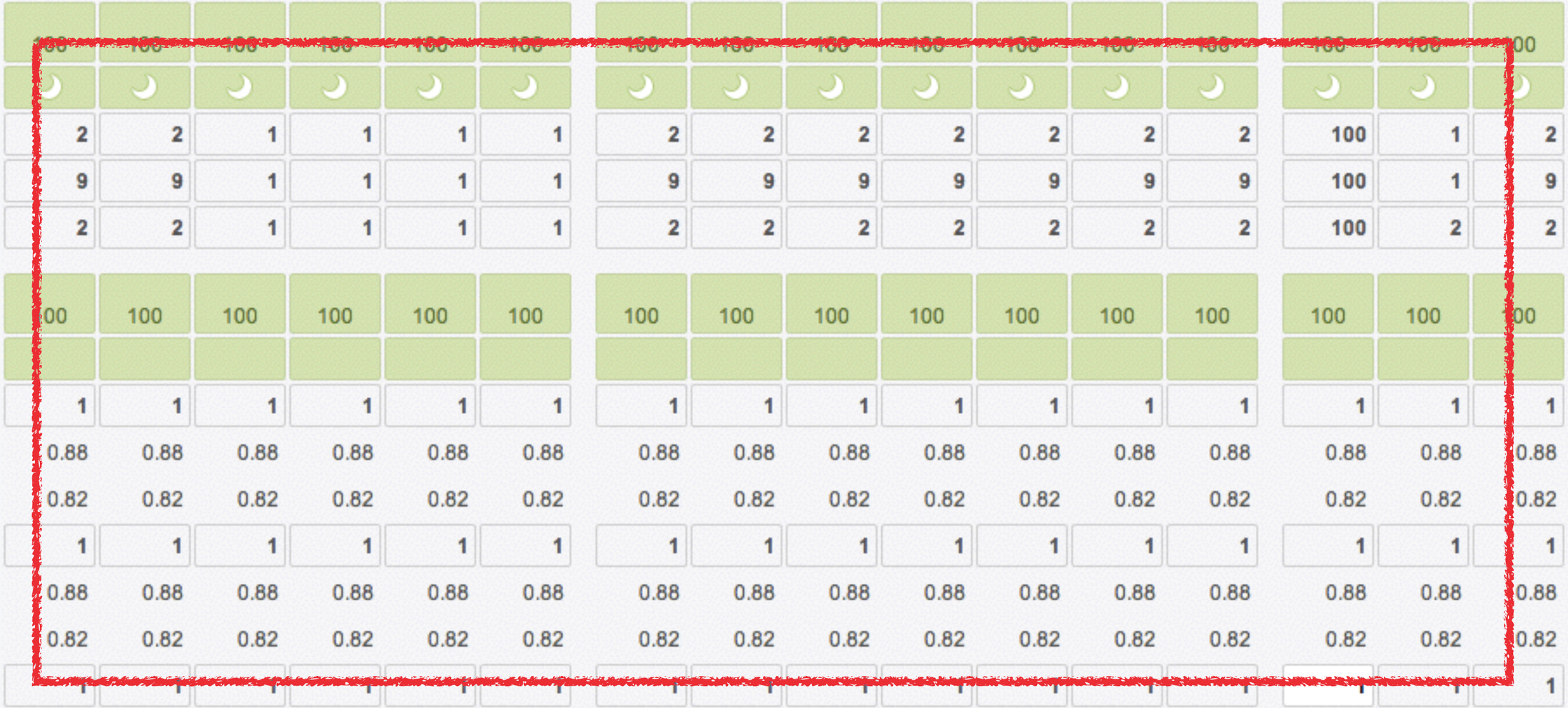
Средний отель

- 25 тарифов × 2 новые строки
- = 50 новых строк
- = ~18 000 новых ячеек (для года)
- = время увеличилось с 3 сек до 3,6 сек (на 16%)

Нужно чтобы было быстрее...

Новая реализация:

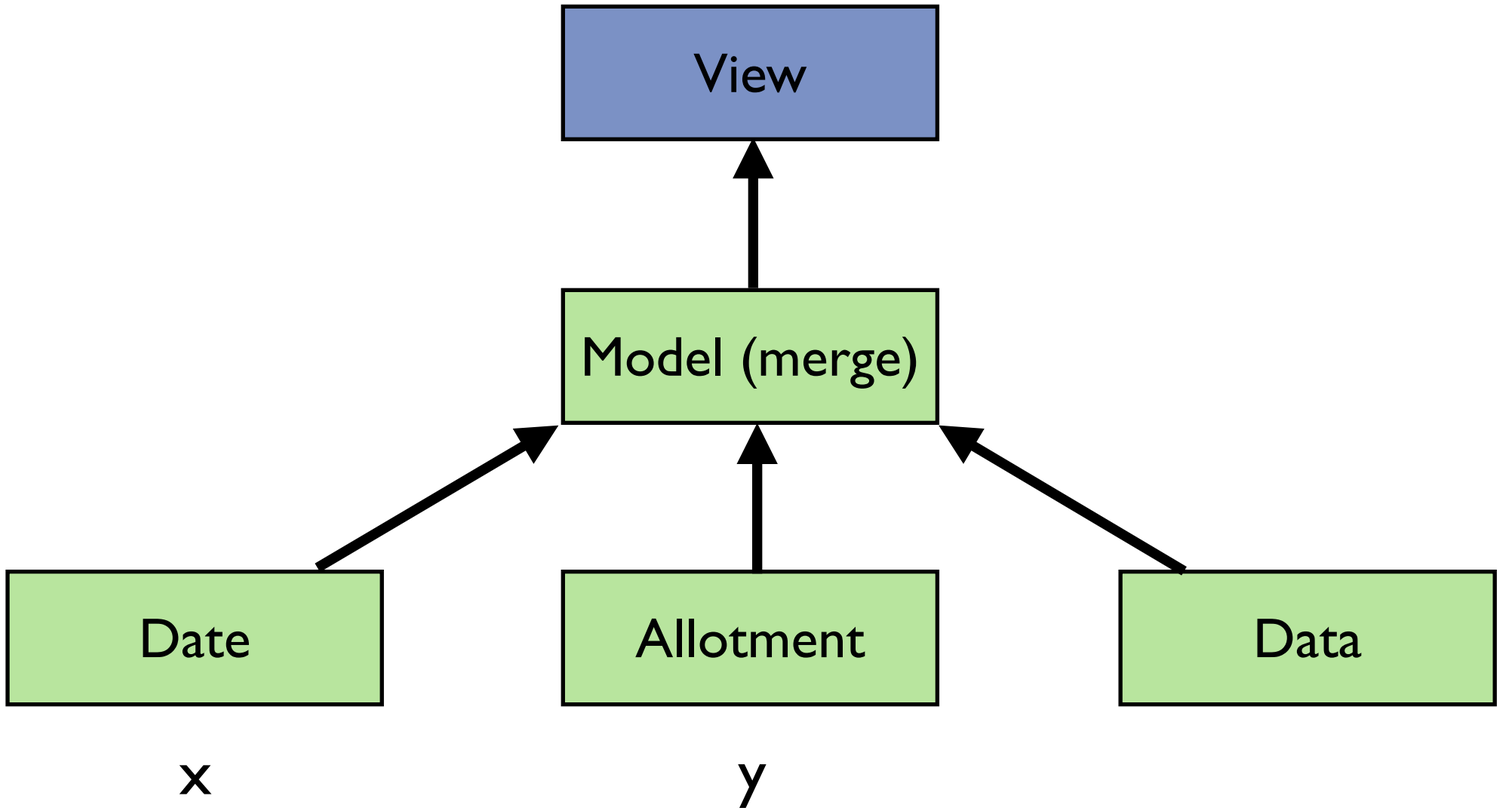
Динамический вьюпорт



2	2	1	1	1	1	2	2	2	2	2	2	2	100	1	2
9	9	1	1	1	1	9	9	9	9	9	9	9	100	1	9
2	2	1	1	1	1	2	2	2	2	2	2	2	100	2	2
100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88
0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88	0.88
0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Рисуем только то,
что попадает в зону видимости

Видимая ячейка



В видимой области
от 200 до 2000 ячеек

Повышенные требования

- [ ] Быстрое создание представлений
- [ ] Быстрая вставка и удаление представлений
- [ ] Переиспользование представлений
- [ ] Быстрое создание моделей
- [ ] Ленивое создание и рассчеты
- [ ] Малое потребление памяти

Повышенные требования

- [x] Быстрое создание представлений
- [ ] Быстрая вставка и удаление представлений
- [x] Переиспользование представлений
- [x] Быстрое создание моделей
- [ ] Ленивое создание и рассчеты
- [x] Малое потребление памяти

Ленивая обработка данных

В месяце от 1 000 до 10 000 ячеек

Немедленное создание

- JSON.parse
- создание моделей
- расчеты
- GC

x12 ~3,0 сек

x24 ~6,0 сек

40-250мс/месяц

Отложенное создание

- JSON.parse
- добавление в кеш

x12 ~0,3 сек

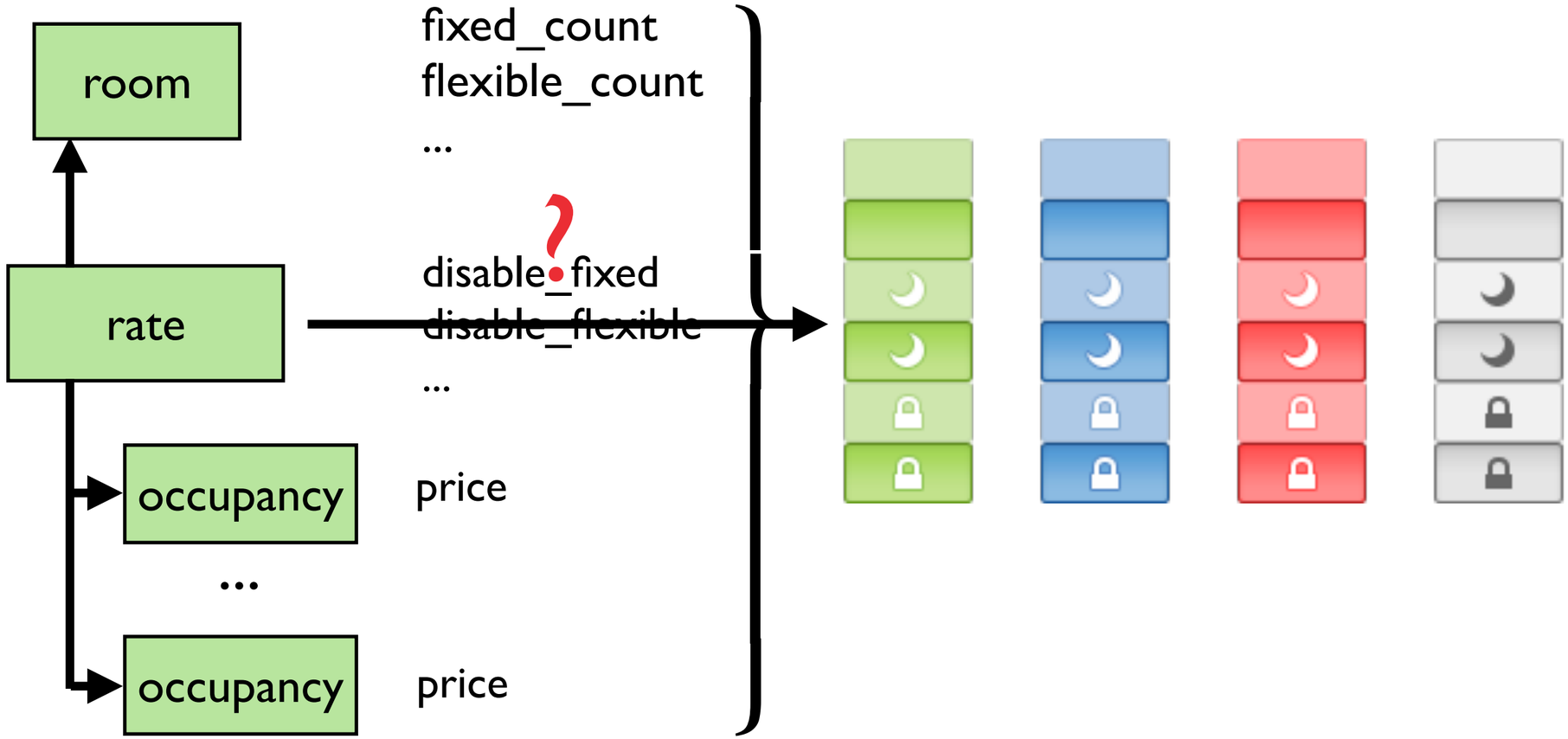
x24 ~0,6 сек

5-25мс/месяц

Ленивое создание моделей

Создаются только те модели,
что попадают во вьюпорт и те,
от которых зависит вид
отображаемых ячеек

Расчет стиля ячейки



Старт

Время:

~1 секунда

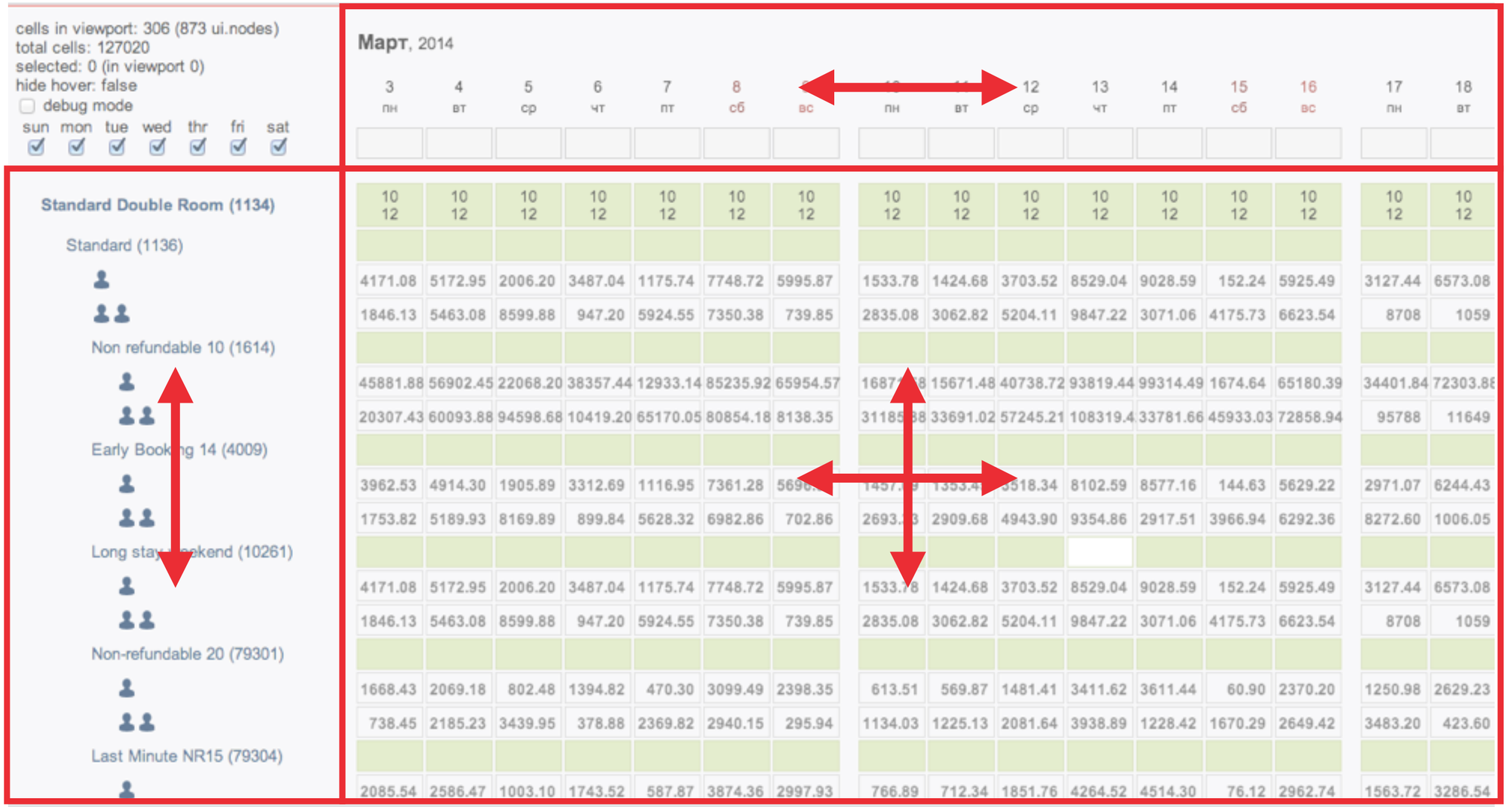
без учета сетевых издержек

Память:

6-10 Мб

с постепенным увеличением
при смещении вьюпорта

Scroll


cells in viewport: 306 (873 ui.nodes)
total cells: 127020
selected: 0 (in viewport 0)
hide hover: false
debug mode
sun mon tue wed thr fri sat
Март, 2014
3 пн
4 вт
5 ср
6 чт
7 пт
8 сб
вс
12 ср
13 чт
14 пт
15 сб
16 вс
17 пн
18 вт
Standard Double Room (1134)
Standard (1136)
Non refundable 10 (1614)
Early Booking 14 (4009)
Long stay weekend (10261)
Non-refundable 20 (79301)
Last Minute NR15 (79304)

3 вьюпорта

При сдвиге вычисляется дельта,
какие ячейки нужно удалить
и какие добавить

В среднем
при сдвиге вьюпорта

удаляется
20-250 ячеек

добавляется
20-250 ячеек

А могут замениться и все ячейки вьюпорта

Основные операции

- Создание моделей
- Создание представлений ячеек
- Вставка и удаление ячеек
- Рассчеты

} Нужно уклыдываться в 16мс

Scroll

Сейчас: 30-40 FPS

Цель: 50-60 FPS

Кажется,

вполне ДОСТИЖИМО

Demo

Бонус трек

пример простой реализации

tinyurl.com/table-scroll

Заключение

Не использовать

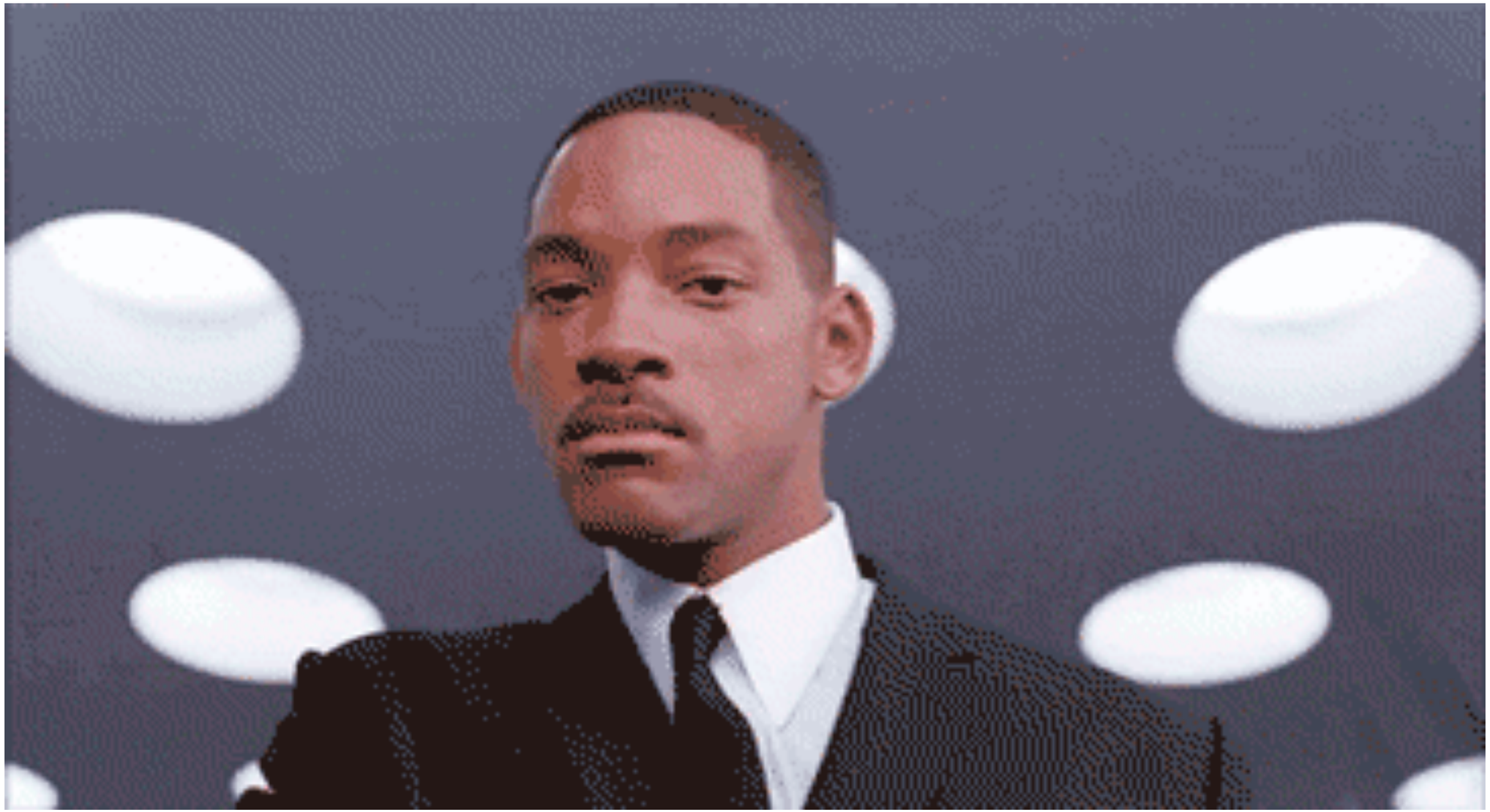
DOM для хранения данных

Модель ≠ представление

Данных может быть
гораздо больше

Возможно делать
дешевые интерфейсы к данным

Можно работать
с сотнями тысяч моделей
на client-side



Главное, делать это аккуратно ;)

Вопросы?

Роман Дворнов
@rdvornov
rdvornov@gmail.com

basis.js
basisjs.com
github.com/basisjs