

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

Роман Дворнов

Ostrovok.ru

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

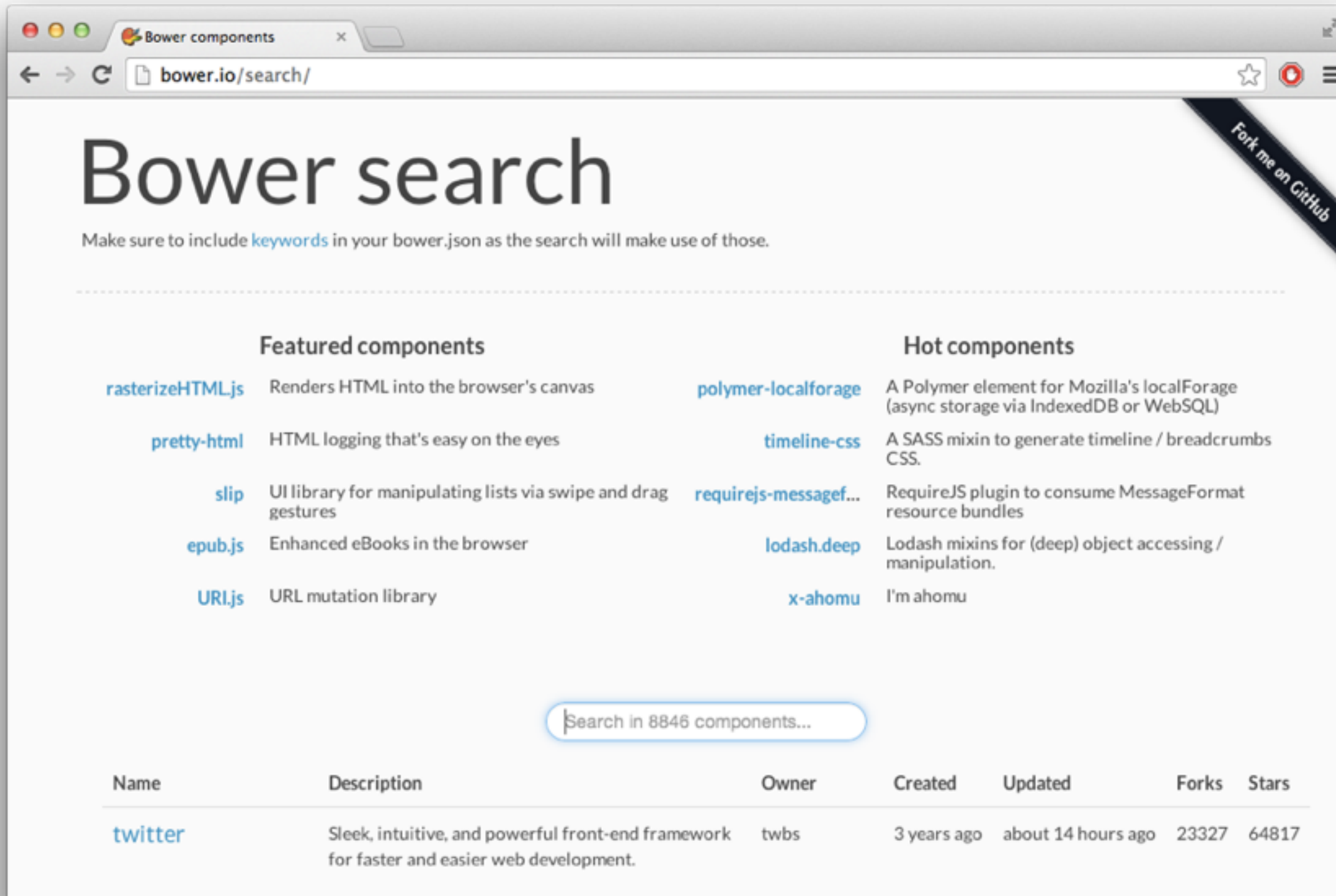


Client-side глазами скептиков...

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"!



ААааа!!!!.. Все пропало!..
Нужно грузить постранично...



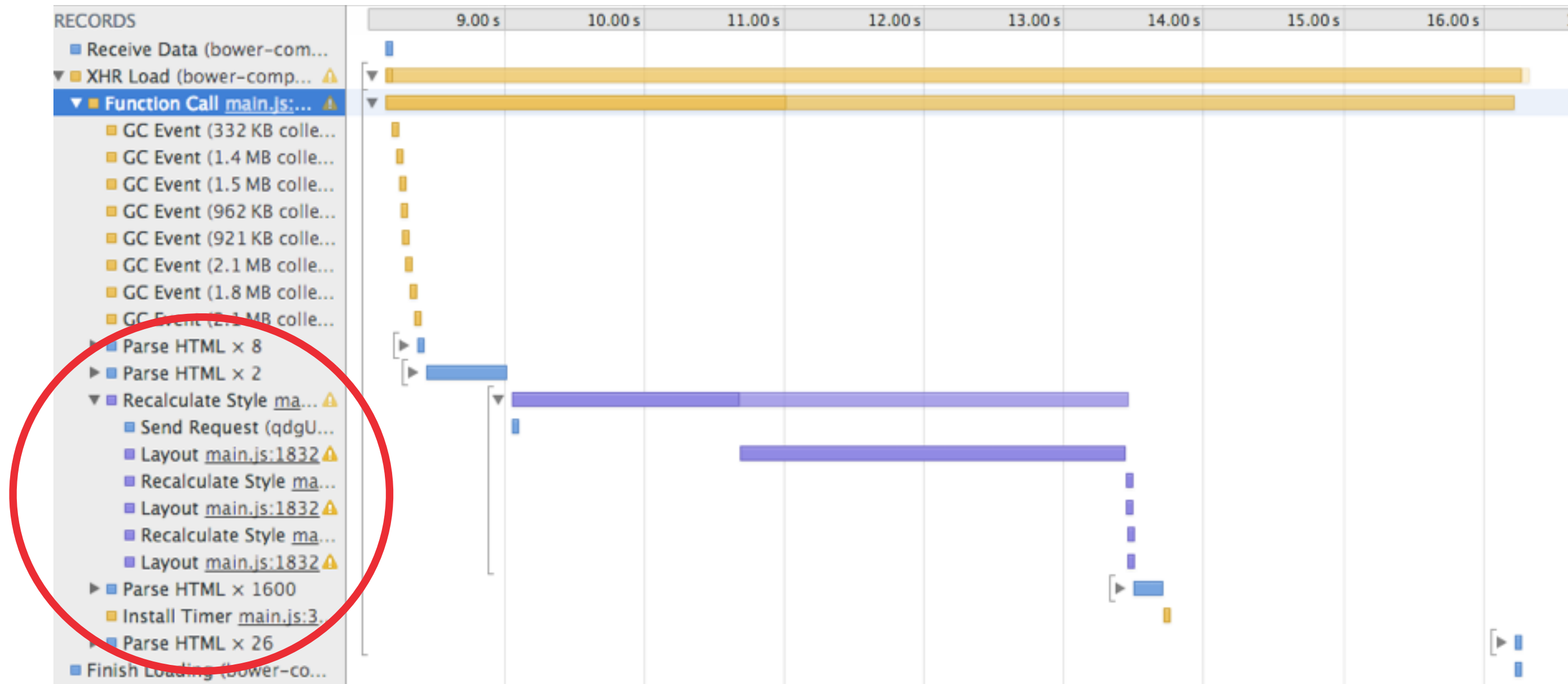
Постойте, не так быстро...

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
- парсинг 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

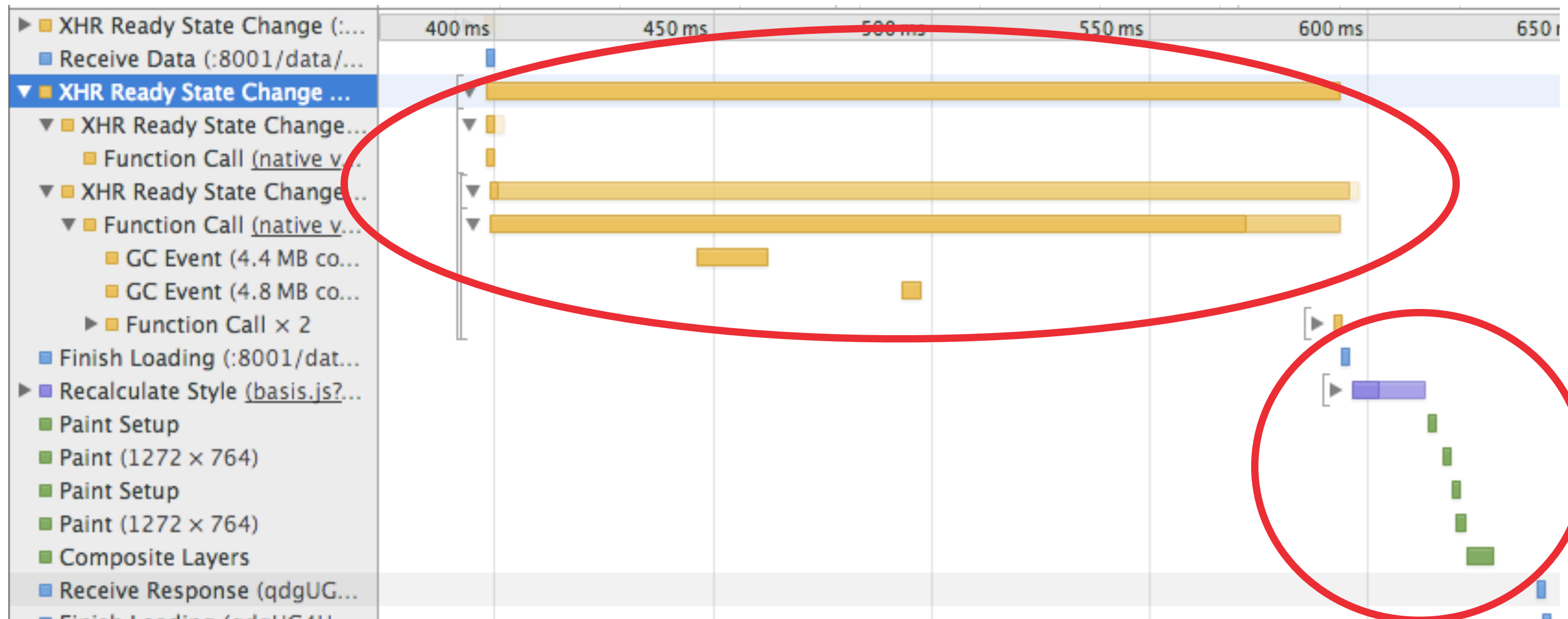
lahmatiy.github.io/bower-search

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

* и можно быстрее

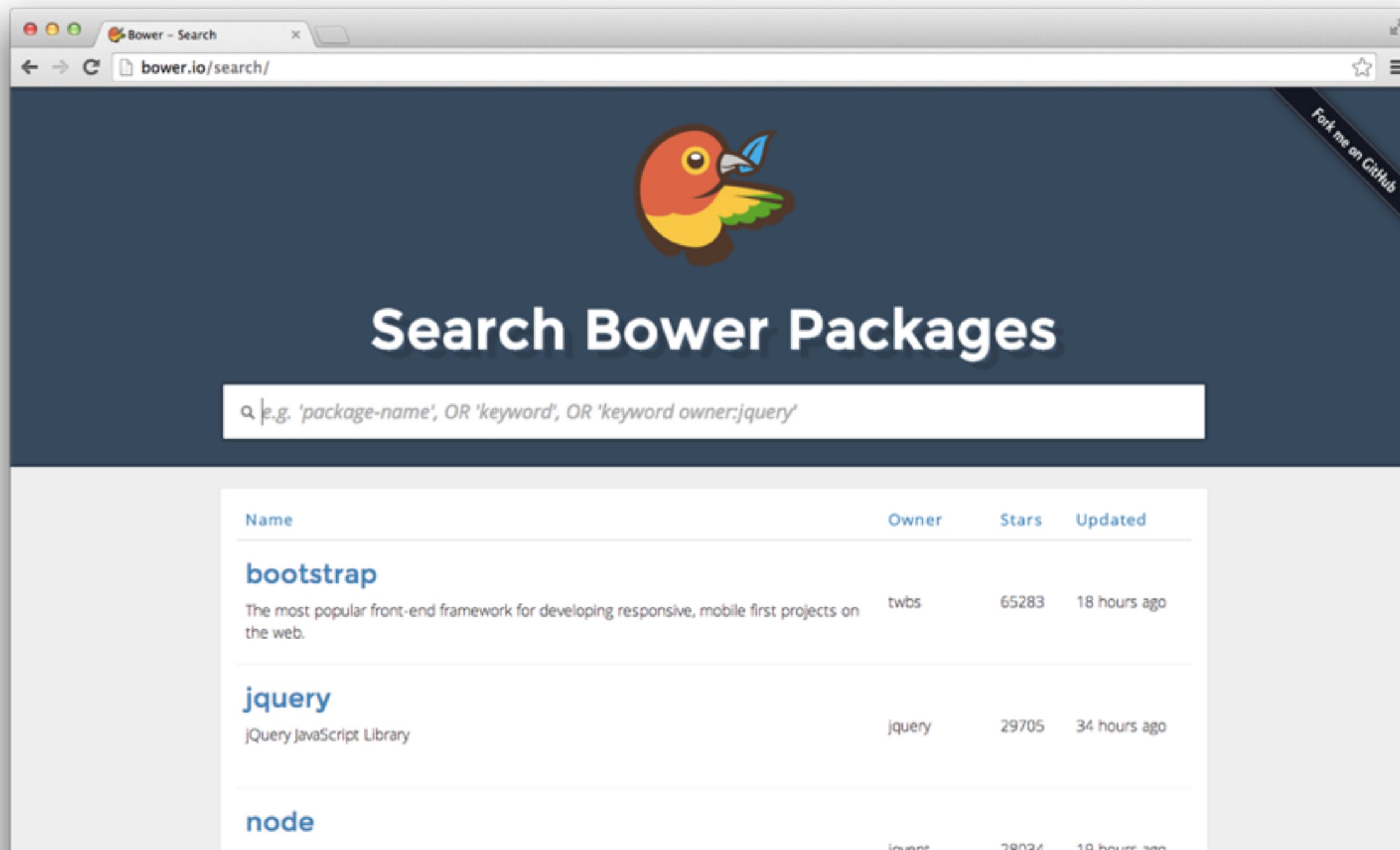
Timeline – Применение данных

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>
```


▼ <tbody class="list">

▼ <tr>

▼ <td class="name">

twitter

</td>

▼ <td class="description" title="Sleek, intuitive, and powerful front-end framework for faster and easier web development.">

▶ <p>...</p>

</td>

<td class="owner">twbs</td>

▶ <td class="created">...</td>

▶ <td class="updated">...</td>

<td class="forks">23346</td>

<td class="stars">64861</td>

<td class="keywords hidden">bootstrap css</td>

</tr>

▶ <tr>...</tr>

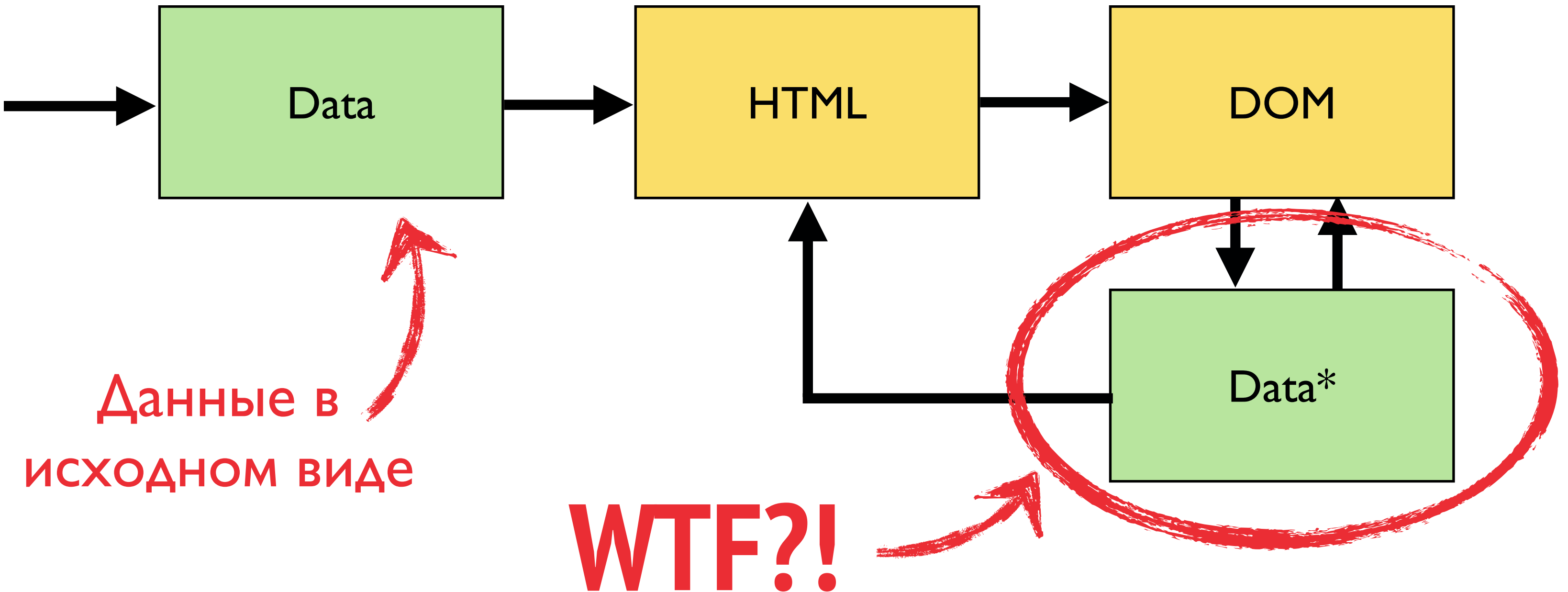
▶ <tr>...</tr>

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



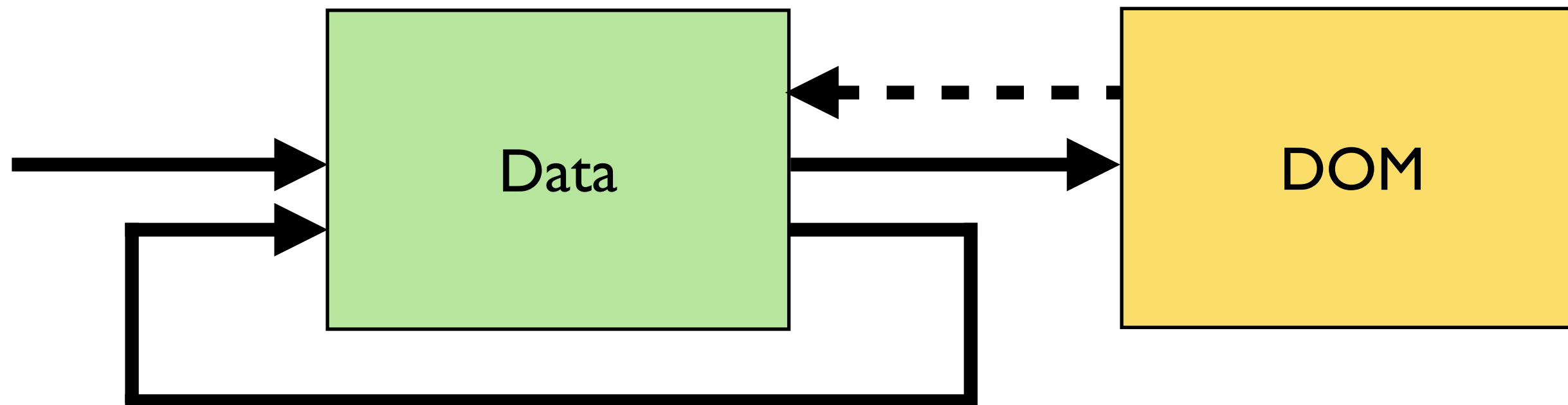
Не используйте
DOM для хранения данных!

А как надо-то?

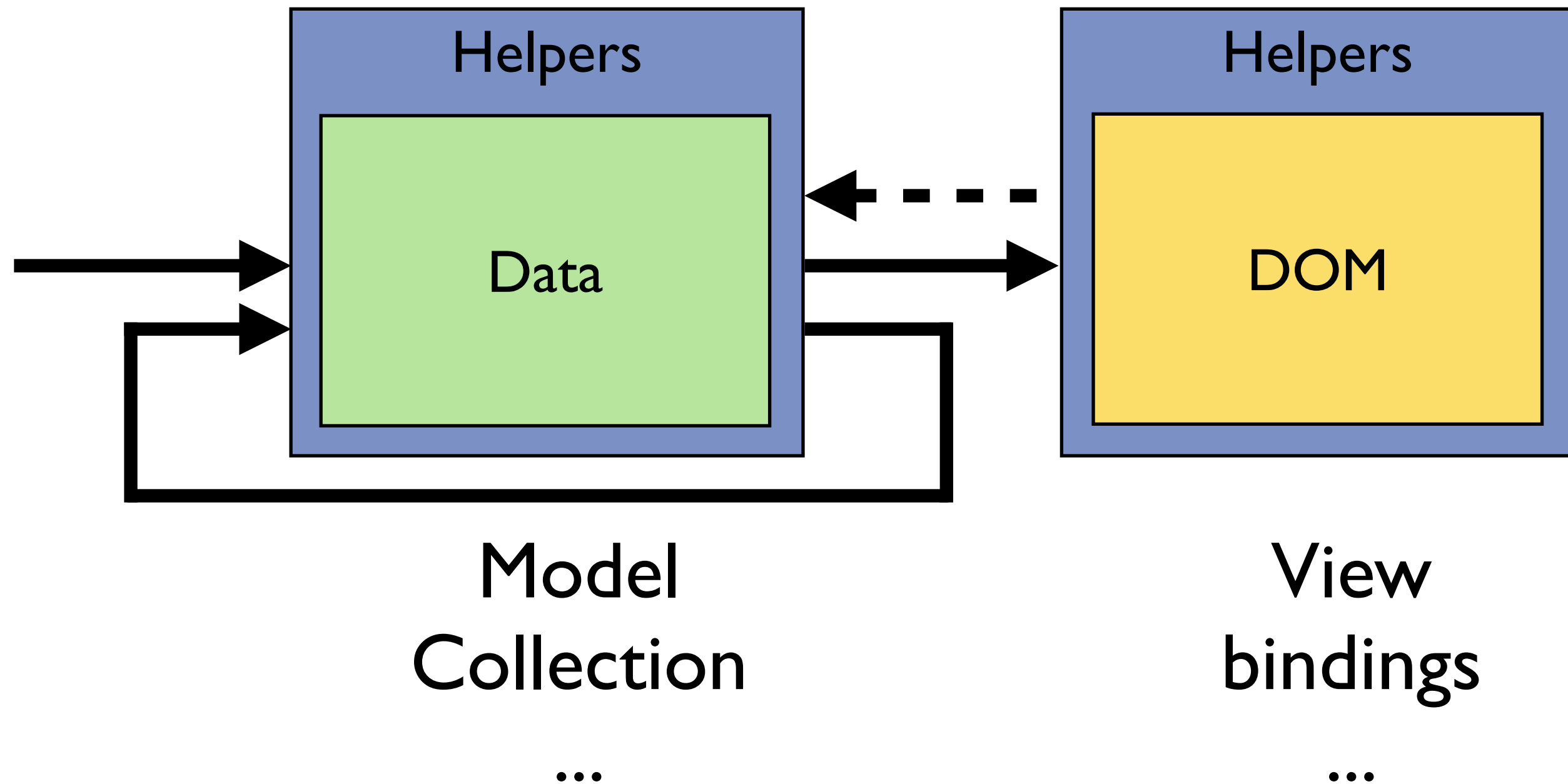
Работа с данными –
вне представлений, в
абстрактном слое

Задача **представлений** –
отражать состояние данных

Идеальный вариант



Идеальный вариант + хелперы



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 $\neq$ 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

Demo blog

Show all posts

1 2 3 4 5 6 7 8 9 10 11 12

[in non et cillum veniam aute culpa tempor culpa](#)

[#5000] 08/04/2014 07:38:57

Exercitation In Tempor Quis Ex In Deserunt Id Ut Exercitation Dolore Pariatur Magna Ea Pariatur Ea Magna Qui Ea Proident Ex Aliqua Dolore Amet Eu Excepteur Officia Ex Culpa Officia Nostrud Excepteur Ad Reprehenderit Mollit Mollit Duis Exercitation Eiusmod Adipisicing Qui Aliqua Eu Id Est Aliquip Voluptate Aliquip Ex Ut Dolore Esse Incidunt Ex Esse Nostrud Nostrud Ut Aliqua In Aliqua Ea

Category: [javascript](#)

Tags: [in](#), [ut](#)

[ipsum pariatur enim elit in](#)

[#4999] 08/04/2014 01:28:11

Sint Labore Proident Eiusmod Esse Non Ea Cillum Reprehenderit Eiusmod Dolore Nulla Ipsum Culpa Pariatur Ad In In Nostrud Voluptate Laboris Sint Aliqua Deserunt Quis Ut Esse Ut Ut Deserunt Ut Dolore Sit Adipisicing Nostrud

Category: [browser](#)

Tags: [duis](#), [do](#)

[exercitation dolor](#)

[#4998] 07/04/2014 15:07:06

Lorem Aliqua Laborum Tempor Do Ad Velit Pariatur Nulla Laboris Magna Non Sunt Fugiat Anim Occaecat Elit Labore Excepteur Consequat Laboris Aliqua Veniam Amet Laboris Fugiat Dolor Ut Consectetur Minim Dolor Est Sit Occaecat Tempor Sint Nostrud Deserunt Elit Non Pariatur Eiusmod Consequat Laborum Cillum Adipisicing Consectetur Labore Dolore Laborum Eu Do Aliquip Irure Duis Occaecat Dolor Eu Irure Et Sed Ut Ea Non Aliquip Ut Lorem Adipisicing Quis Enim Occaecat Labore Aute Culpa Dolor Nulla Dolore Cupidatat Occaecat Sed Pariatur Occaecat

Category: [framework](#)

Tags: [deserunt](#)

Tags cloud

Duis deserunt nisi voluptate
adipisicing pariatur culpa enim
Excepteur aliquip incididunt quis
laboris sit irure proident ullamco anim
sed occaecat cupidatat minim ea
laborum eiusmod non officia sint
qui ad commodo elit Lorem aute
consequat amet eu esse sunt labore
cillum exercitation magna et
dolor ex veniam tempor
est reprehenderit nostrud
mollit dolor nulla velit ipsum
fugiat id aliqua ut consectetur do
ut in

Archive

▼ 2014

[April](#) (19)

[March](#) (81)

[February](#) (69)

[January](#) (81)

► 2013

► 2012

► 2011

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 0000 0000

???

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 M6



Object
~9,0 M6

Создание моделей

10 000 экземпляров, Кб

Решение	0 полей	10 полей
<code>new basis.data.Object()</code>	240	240
<code>basis.entity.Type()</code>	440	840
<code>new Backbone.Model()</code>	920	1 480
<code>Ember.Object.create()</code>	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

The screenshot displays the 'Островок.ру' (Ostrovok.ru) website interface for hotel booking. The browser address bar shows the URL: `extranet.ostrovok.ru/#/hotels/780053632/grids/prices/multi/`. The page title is 'Цены' (Prices). The main navigation bar includes tabs for 'Доступность, цены и ограничения' (Availability, prices and restrictions), 'Бронирования' (Bookings), 'Отель и номера' (Hotel and rooms), 'Счета' (Accounts), and 'Помощь' (Help). Below this, there are sub-tabs for 'Доступность' (Availability), 'Цены, RUB' (Prices, RUB), and 'Ограничения' (Restrictions). The current view is 'Цены, RUB'.

The interface shows two calendar grids for February and March 2014. The left grid is for February 2014, and the right grid is for March 2014. Each grid displays rates for different room types and tariffs. A red arrow points to the 'Standard' room type in the March grid, specifically to the cell for the 9th of March, which shows a rate of 2310.

On the left side, there is a sidebar with a gear icon and the text 'Создать спец предложение' (Create special offer). Below this, there are filters for 'Фильтр дней:' (Filter days) and 'Отель' (Hotel). The filters include checkboxes for days of the week (пн, вт, ср, чт, пт, сб, вс) and icons for room types (one person, two people, etc.).

The main content area is divided into two sections: 'февраль, 2014' (February, 2014) and 'март, 2014' (March, 2014). Each section contains a table of rates for different room types and tariffs. The table for February 2014 shows rates for the 25th, 26th, 27th, and 28th. The table for March 2014 shows rates for the 1st through 10th. The rates are displayed in a grid format, with columns for the date and rows for the room type and tariff.

At the bottom of the page, there is a button labeled 'Сохранять нечего' (Nothing to save).

Rates &
Availability
View

Масштабы бедствия

Структура отеля

Даты

до **300** строк
(и растет)

×

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 – трансформируем, 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

Фильтр дней:

☒

☒

☒

☒

☒

☒

☒

пн вт ср чт пт сб вс

ОТЕЛЬ

Стандартный ...тный номер

Стандартный

Мин. дней на заезд

Макс. дней на заезд

Мин. дней н...оживание

Макс. дней н...оживание

Невозвратный тариф

Мин. дней на заезд

март, 2014

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
вс	пн	вт	ср	чт	пт	сб	вс	пн	вт	ср	чт	пт	сб	вс
100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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...

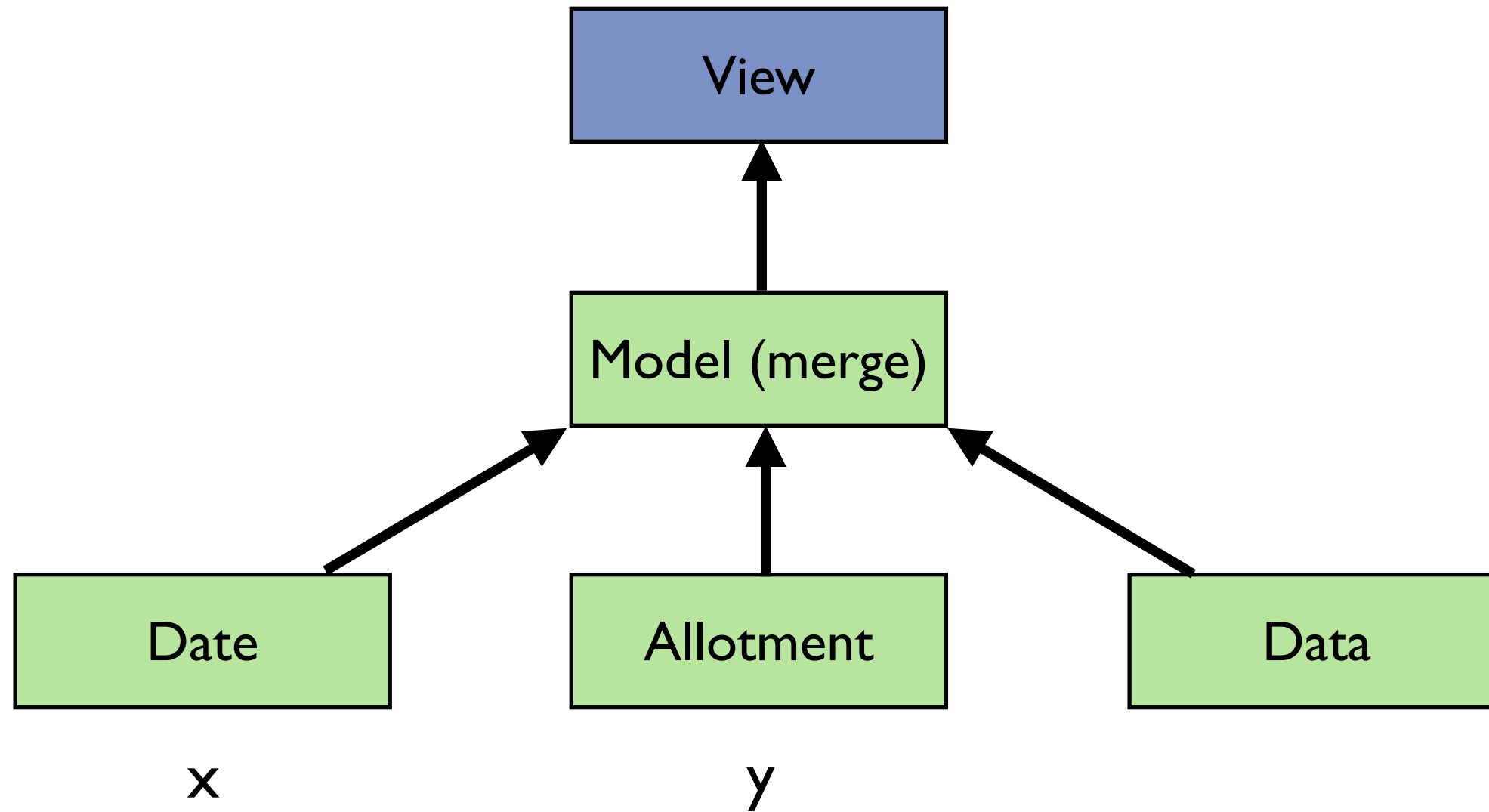
Новая реализация:

Динамический **вьюпорт**

100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
☾	☾	☾	☾	☾	☾	☾	☾	☾	☾	☾	☾	☾	☾	☾	☾
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 расчеты
- ☐ Малое потребление памяти

Повышенные требования

- ☒ Быстрое создание представлений
- ☐ Быстрая вставка и удаление представлений
- ☒ Переиспользование представлений
- ☒ Быстрое создание моделей
- ☐ Ленивое создание и расчеты
- ☒ Малое потребление памяти

Ленивая обработка данных

В месяце от 1 000 до 10 000 ячеек

Немедленное создание

- JSON.parse
 - создание моделей
 - расчеты
x12 ~3,0 сек
 - GC
x24 ~6,0 сек
- 40-250мс/месяц

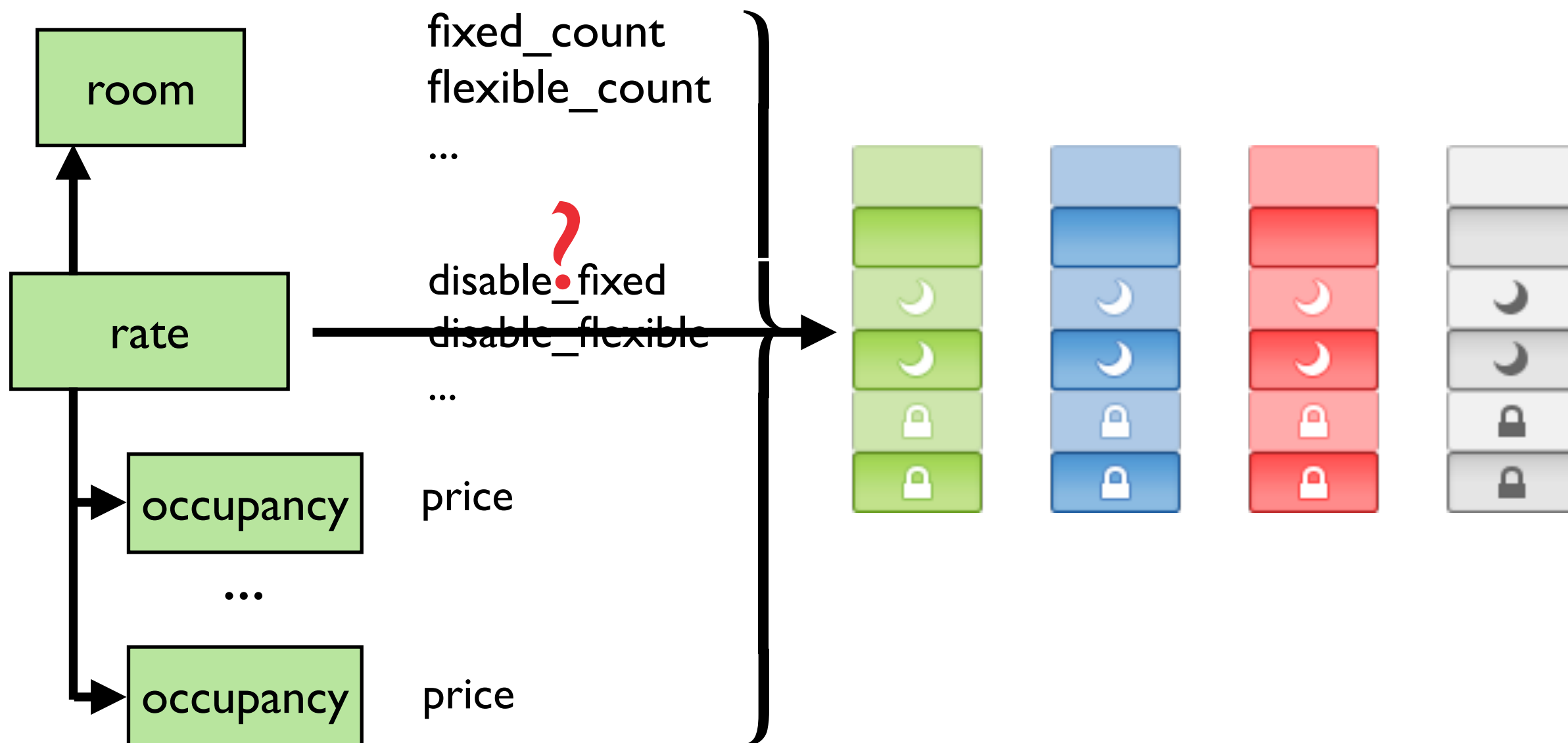
Отложенное создание

- 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	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
пн	вт	ср	чт	пт	сб	вс	пн	вт	ср	чт	пт	сб	вс	пн	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)

10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
4171.08	5172.95	2006.20	3487.04	1175.74	7748.72	5995.87	1533.78	1424.68	3703.52	8529.04	9028.59	152.24	5925.49	3127.44	6573.08
1846.13	5463.08	8599.88	947.20	5924.55	7350.38	739.85	2835.08	3062.82	5204.11	9847.22	3071.06	4175.73	6623.54	8708	1059
45881.88	56902.45	22068.20	38357.44	12933.14	85235.92	65954.57	16871.18	15671.48	40738.72	93819.44	99314.49	1674.64	65180.39	34401.84	72303.86
20307.43	60093.88	94598.68	10419.20	65170.05	80854.18	8138.35	31185.88	33691.02	57245.21	108319.4	33781.66	45933.03	72858.94	95788	11649
3962.53	4914.30	1905.89	3312.69	1116.95	7361.28	5690.7	1457.19	1353.4	3518.34	8102.59	8577.16	144.63	5629.22	2971.07	6244.43
1753.82	5189.93	8169.89	899.84	5628.32	6982.86	702.86	2693.13	2909.68	4943.90	9354.86	2917.51	3966.94	6292.36	8272.60	1006.05
4171.08	5172.95	2006.20	3487.04	1175.74	7748.72	5995.87	1533.78	1424.68	3703.52	8529.04	9028.59	152.24	5925.49	3127.44	6573.08
1846.13	5463.08	8599.88	947.20	5924.55	7350.38	739.85	2835.08	3062.82	5204.11	9847.22	3071.06	4175.73	6623.54	8708	1059
1668.43	2069.18	802.48	1394.82	470.30	3099.49	2398.35	613.51	569.87	1481.41	3411.62	3611.44	60.90	2370.20	1250.98	2629.23
738.45	2185.23	3439.95	378.88	2369.82	2940.15	295.94	1134.03	1225.13	2081.64	3938.89	1228.42	1670.29	2649.42	3483.20	423.60
2085.54	2586.47	1003.10	1743.52	587.87	3874.36	2997.93	766.89	712.34	1851.76	4264.52	4514.30	76.12	2962.74	1563.72	3286.54

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
- Расчеты

Нужно
укладываться
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

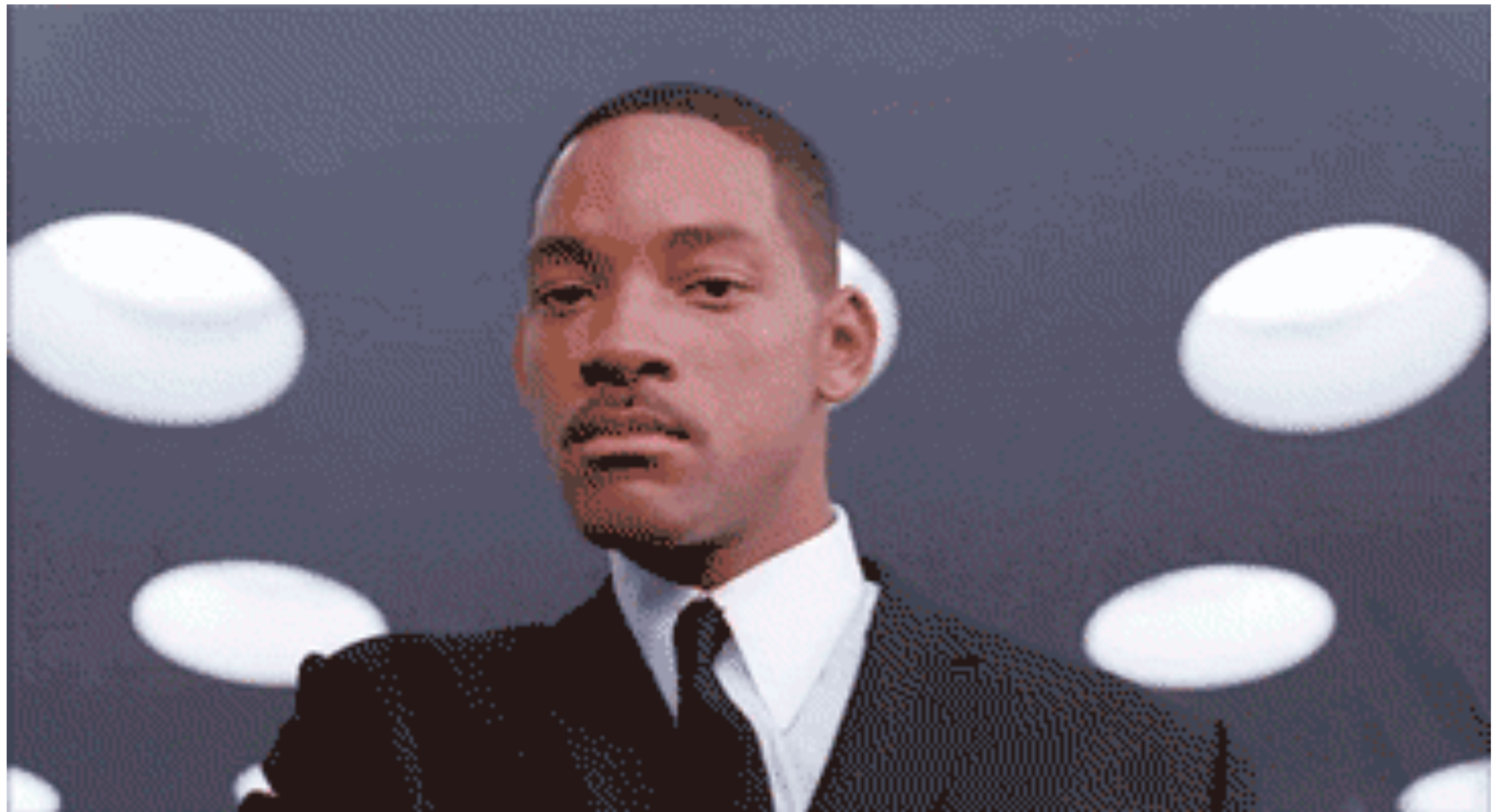
Не использовать
DOM для хранения данных

Модель $\neq$ 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