Бенчмарки и perf-тесты

для JavaScript и TypeScript

Виктор Хомяков, программист

- Виктор Хомяков
- 5 лет в Поиске Яндекса
- 5 лет в команде скорости

Bing observed that an engineer that improves server performance by 10ms (that's 1/30 of the speed that our eyes blink) more than pays for his fully-loaded annual costs. Every millisecond counts.

wpostats.com/tags/server/

О чём НЕ будет доклад

- Как ускорить страницу в X раз
- Какой браузер быстрее
- В чём смысл жизни

О чём будет доклад?

- Как писать бенчмарки
- Как не делать ошибок
- Кратко: performance-тесты, профилирование

Зачем оно вам?

Существующий проект

Хочется:

- Увеличить RPS
- Сэкономить железо
- Ускорить места, где клиенты жалуются

Пример из личного опыта: Яндекс Поиск

- Миграция с jQuery на React
- Оптимизация клиентского кода
- Грузим jQuery+React без потерь в скорости и деньгах

Пример 2: Яндекс Поиск

- Оптимизация серверного кода
- +10% пользователей
  без дополнительного железа

Больше примеров

- PerfPerfPerf 3perf.com/#clients
- WPO Stats wpostats.com
- web.dev web.dev/tags/case-study/

Новый проект

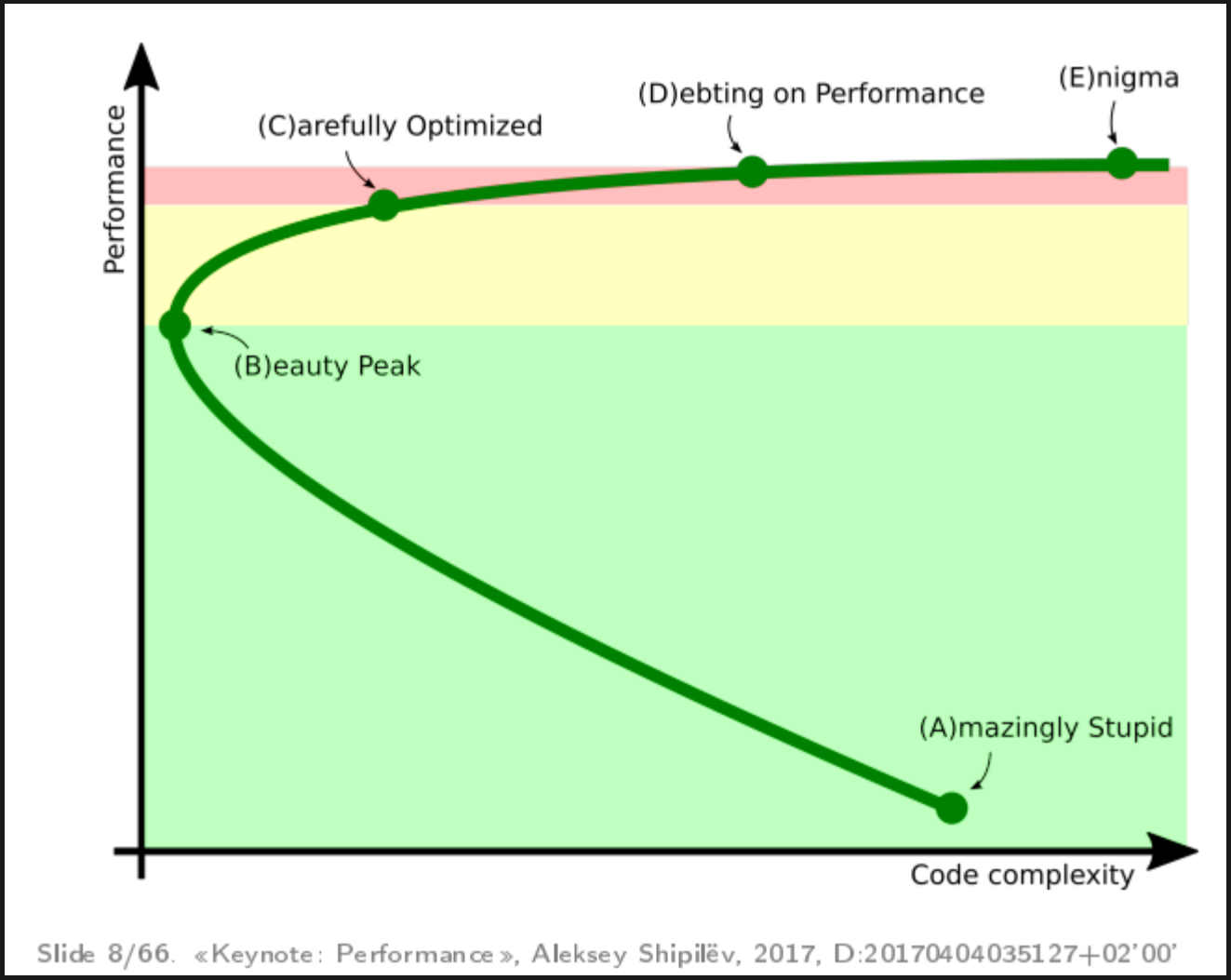
Хочется:

- Выбрать подходящую библиотеку (Moment, Dayjs etc.)
- Выбрать лучшую реализацию функциональности
- Или показать, что в данном месте это неважно

- Чтобы ускорить, надо измерить
- Чтобы выбрать библиотеку, надо измерить
- Чтобы показать, что скорость не важна — надо измерить

Каких результатов ожидать в вашем проекте

Кривая имени Шипилёва



youtu.be/p2b4JHESEOc?t=331

Пример из точки “А” — ускорение в секундах

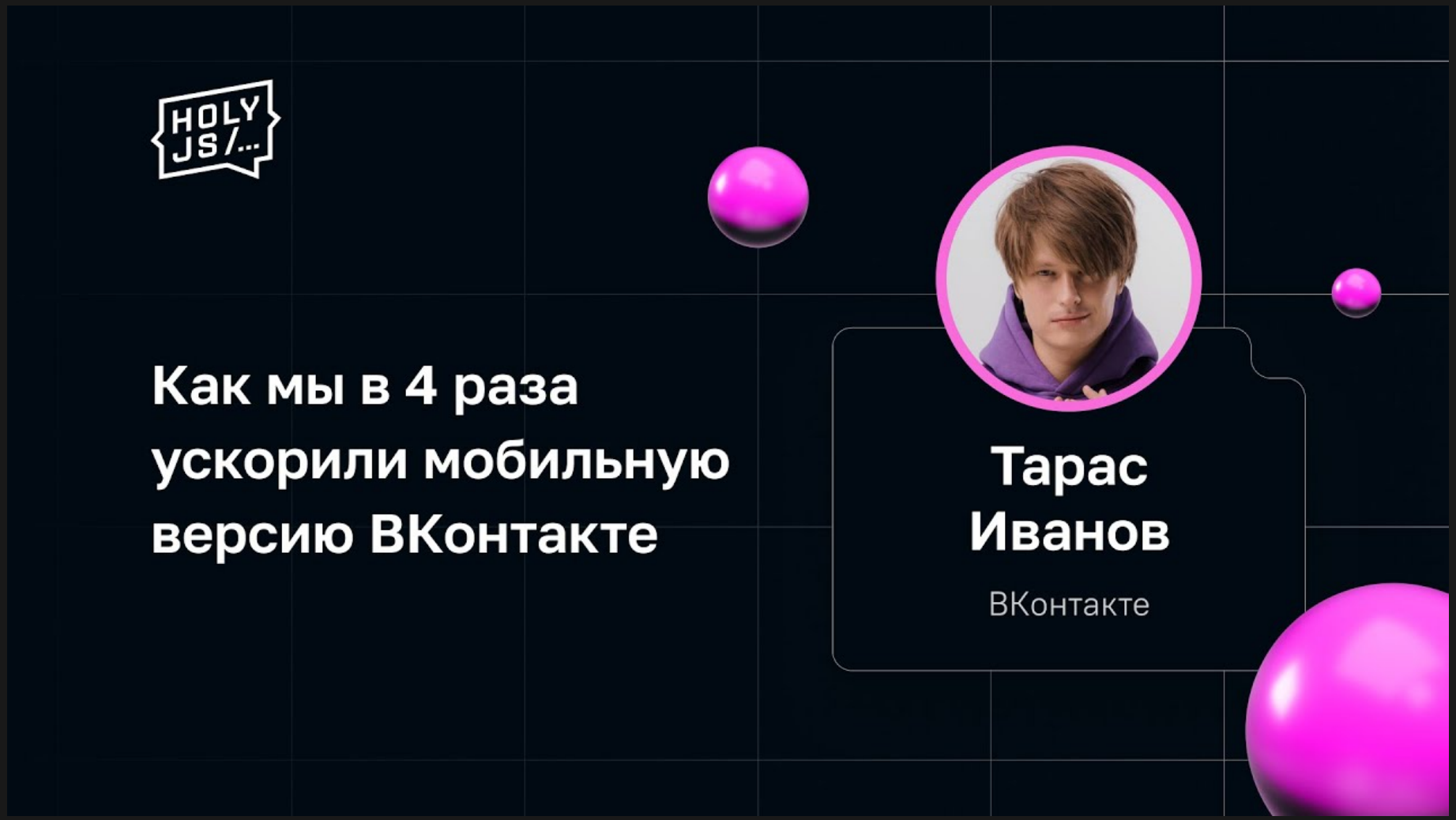
Профилирование Node.js, или Как мы в несколько раз ускорили Практикум

Алексей Яковлев
Яндекс Практикум

FC Frontend Conf 2023

2 и 3 октября 2023, Москва
Старт Хаб на Красном Октябре

Ещё пример из точки “А” — ускорение в секундах



youtu.be/O09Scq9DFss

Пример из точки “С” — десятки миллисекунд

Я.Суеетцсмо?
по разработке интесхжктпг<
Ajay; пу В еп Y<
АлбжнЬйам Нпзди

YfD */

youtu.be/ML1H_MHChIQ

Пример из точки “С” — десятки миллисекунд

Я.Субботник;
по разработке интерфейсов;
Ajax: от А до Х;
Александр Сулима;

YfD */

youtu.be/ML1H_MHChIQ

Бенчмарк: что измеряем?

Пропускная способность, RPS

- ! Стабильный поток запросов
- ! Примерно одинаковая длительность

Длительность одной операции, s/op

- ⛔ Гранулярность/latency таймера
- ⛔ JIT-компиляция
- ⛔ Шум ОС/GC

Длительность N операций, s

- ⛔ Шум ОС/GC
- ✅ Просто и достаточно точно
- ✅ Стандарт де-факто

Тики процессора (TIC)

✅ Не зависит от ОС/GC/тактовой частоты/etc.

⛔ Плохая поддержка: Chrome/Linux

✅ github.com/stoyan/ticr

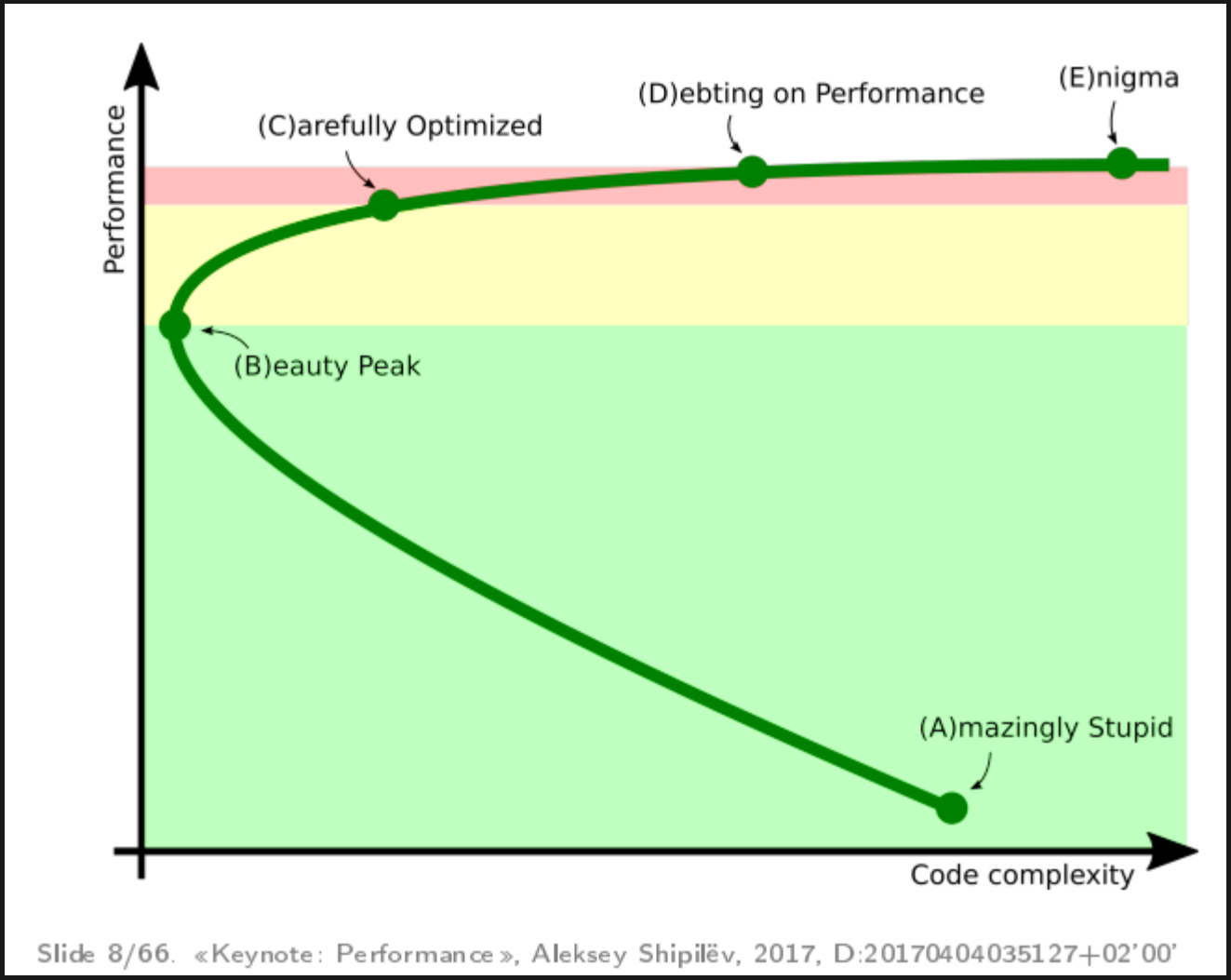
Сравнение скорости реализаций

- ✅ Разное время операций
- ✅ Реализации выполняют N операций
- ✅ Стандарт де-факто

Особенности движков JS, влияющие на результаты бенчмарка

Измерение времени

В точке “A” не заморачиваемся



Гранулярность таймера: как часто обновляется значение

```
var t1, t2;

t1 = performance.now();
do { t2 = performance.now() } while (t1 === t2);
console.log(t2 - t1);

t1 = Date.now();
do { t2 = Date.now() } while (t1 === t2);
console.log(t2 - t1);
```

Гранулярность, ms

Браузер	performance.now()	Date.now()
Chrome 116 Android, MacOS, Windows	0.1	1
Firefox 116 MacOS, Windows	1	1
Safari 16 MacOS	1	1

jsfiddle.net/f7n6c50o/

Гранулярность в Node.js v16, ms

Метод	Значение
process.hrtime.bigint()	≈ 0.002 – 0.03
performance.now()	≈ 0.004 – 0.02
Date.now()	1

Latency таймера:
сколько времени занимает сам таймер

```
const start = Date.now();
// Имитация замеряемого кода
const time = Date.now() - start;
```

Latency, µs

Браузер	**performance.now()**	**Date.now()**
Chrome 116 Android	0.4	0.07
Chrome 116 MacOS, Windows	0.2	0.05
Firefox 116 MacOS, Windows	0.08	0.1
Safari 16 MacOS	0.05	0.005

jsbench.me/7mljeki2e3/1

Как бороться

- Один вызов таймера на N операций

Сборка мусора (Garbage Collection, GC)

- Major GC — "stop the world"
- Minor GC — всё равно грузит CPU
- $T_{benchmark} \mathrel{+}= T_{GC}$

Как бороться

- `node --trace-gc`
- `node --expose-gc`
- Размер хипа
- Может частый GC и есть проблема?

JIT

V8 (Chrome, Node.js):

- Ignition
- Sparkplug
- TurboFan

v8.dev/blog/sparkplug

SpiderMonkey (Firefox):

- Interpreter
- Baseline Interpreter
- Baseline JIT
- Ion JIT

JavaScriptCore (Safari, Bun):

- Low-Level Interpreter (LLInt)
- Baseline JIT
- Data Flow Graph (DFG) JIT
- Faster than Light (FTL) JIT

TL;DR:

- Первое (холодное) выполнение кода — интерпретатор
- Затем код оптимизируется “на лету” — JIT

Бенчмарк открытия страницы:

- нужно первое выполнение
- JIT мешает

Бенчмарк всего остального:

- интерпретатор мешает
- нужно включить JIT
- но не посреди бенчмарка!

Как бороться

- Прогрев кода (warm-up)

```
const start = performance.now();
for (let i = 0; i < 100_000; i++) {
    // Ваш код
}
const duration = performance.now() - start;
```

```
// Прогрев
for (let i = 0; i < 10_000; i++) {
    // Ваш код
}

// Измерение
const start = performance.now();
for (let i = 0; i < 100_000; i++) {
    // Ваш код
}
const duration = performance.now() - start;
```

Удаление мёртвого кода (Dead Code Elimination, DCE)

```
// Подсчёт числа букв "e" в строке
str.split("e").length - 1;
```

youtu.be/HPFARivHJRY?t=409

900 миллионов op/s в Firefox

```
const start = performance.now();
for (let i = 0; i < 10_000; i++) {
    str.split("e").length - 1;
}
const duration = performance.now() - start;
```

```
const start = performance.now();
for (let i = 0; i < 10_000; i++) {
    // str.split("e").length - 1; // Нигде не используется
}
const duration = performance.now() - start;
```

```
const start = performance.now();
for (let i = 0; i < 10_000; i++) {
    // Пустой цикл
}
const duration = performance.now() - start;
```

```
const start = performance.now();
const duration = performance.now() - start;
```

Как бороться

- Накапливать результат
- Использовать в сайд-эффектах
- Black hole в Java JMH
  github/openjdk/jmh/Blackhole.java

```
const start = performance.now();
for (let i = 0; i < 10_000; i++) {
    str.split("e").length - 1;
}
const duration = performance.now() - start;
```

```
let bh = 0; // black hole
const start = performance.now();
for (let i = 0; i < 10_000; i++) {
    bh += str.split("e").length - 1; // Накапливаем
}
const duration = performance.now() - start;
```

```
let bh = 0; // black hole
const start = performance.now();
for (let i = 0; i < 10_000; i++) {
    bh += str.split("e").length - 1; // Накапливаем
}
const duration = performance.now() - start;
console.log(bh); // Сайд-эффект
```

```
let bh = 0; // black hole
const start = performance.now();
for (let i = 0; i < 10_000; i++) {
    bh += str.split("e").length - 1; // Накапливаем
}
const duration = performance.now() - start;
if (Math.random() >= 1)
    console.log(bh); // Сайд-эффект
```

Варианты сайд-эффектов

```
if (Math.random() >= 1) {
    // Вывод в консоль
    console.log(bh);
}
```

```
if (Math.random() >= 1) {
    // Исключение
    throw new Error(bh);
}
```

Inline Cache (IC)

Если данные для бенчмарка придумываем “на ходу”

- Однотипные объекты
- Одинаковый hidden class
- IC работает лучше

Как бороться

- Тип, порядок и размер полей как в проде
- Дамп данных из прода
- N записей — рандомизировать порядок

Дамп данных в Node.js

```
const fs = require('fs');

let uid = 1;

function myFunc(data) {
    const name = `data-${Date.now()}-${uid++}.json`;
    fs.writeFile(name, JSON.stringify(data));
    // остальной код ...
}
```

Реализация массивов в V8

- FixedArray + Packed elements
- FixedArray + Holey elements
- NumberDictionary + Dictionary elements

v8.dev/blog/elements-kinds

Лимит FixedArray

```
kMaxFastArrayLength = 32 * 1024 * 1024;
```

github.com/v8/src/objects/js-array.h

```
$ node --allow-natives-syntax
Welcome to Node.js v16.14.2.

>
```

```
$ node --allow-natives-syntax
Welcome to Node.js v16.14.2.

> %DebugPrint([])
```

```
$ node --allow-natives-syntax
Welcome to Node.js v16.14.2.

> %DebugPrint([])
DebugPrint: [JSArray]
 - ...
 - elements: <FixedArray[0]> [PACKED_SMI_ELEMENTS]
 - length: 0
```

```
> %DebugPrint([1,,3])
```

```
> %DebugPrint([1,,3])
DebugPrint: [JSArray]
 - ...
 - elements: <FixedArray[3]> [HOLEY_SMI_ELEMENTS (COW)]
 - length: 3
```

```
> %DebugPrint(new Array(32 * 1024 * 1024 + 1))
```

```
> %DebugPrint(new Array(32 * 1024 * 1024 + 1))
DebugPrint: [JSArray]
 - ...
 - elements: <NumberDictionary[16]> [DICTIONARY_ELEMENTS]
 - length: 33554433
```

Как бороться

- Понять, какие массивы в проде
- Использовать в бенчмарке такие же

Реализация строк в V8, SM, JSC

- ConsString
- SlicedString
- "обычные" строки

Сколько памяти в Chrome займут строки?

```
const s1 = new Array(1000000).join("x");
const s2 = "x".repeat(1000000);
const s3 = "x".repeat(1000000).toLowerCase();
```

Сколько памяти в Chrome займут строки?

```
const s1 = new Array(1000000).join("x");      //    ≈500 b
const s2 = "x".repeat(1000000);
const s3 = "x".repeat(1000000).toLowerCase();
```

Сколько памяти в Chrome займут строки?

```
const s1 = new Array(1000000).join("x");       //    ≈500 b
const s2 = "x".repeat(1000000);                //    ≈500 b
const s3 = "x".repeat(1000000).toLowerCase();
```

Сколько памяти в Chrome займут строки?

```
const s1 = new Array(1000000).join("x");       //    ≈500 b
const s2 = "x".repeat(1000000);                //    ≈500 b
const s3 = "x".repeat(1000000).toLowerCase(); // 1000012 b
```

```
const s1 = new Array(1000000).join("x");       //    ≈500 b
const s2 = "x".repeat(1000000);                //    ≈500 b
const s3 = "x".repeat(1000000).toLowerCase(); // 1000012 b
```

```
> %DebugPrint(s1)
...
 - type: CONS_ONE_BYTE_STRING_TYPE

> %DebugPrint(s2)
...
 - type: CONS_ONE_BYTE_STRING_TYPE

> %DebugPrint(s3)
...
 - type: ONE_BYTE_STRING_TYPE
```

```
const s1 = new Array(1000000).join("x"); // ≈500 b
```

```
> %DebugPrint(s1)
...
 - type: CONS_ONE_BYTE_STRING_TYPE
```

```
s1.indexOf("a"); // теперь s1 занимает 1000012 b
```

```
> %DebugPrint(s1)
...
 - type: ONE_BYTE_STRING_TYPE
```

- indexOf("something")
- slice(1)
- substring(1)
- toLowerCase()/toUpperCase()

Как бороться

- Понять, какие строки в проде
- Использовать в бенчмарке такие же

Особенности движков: recap

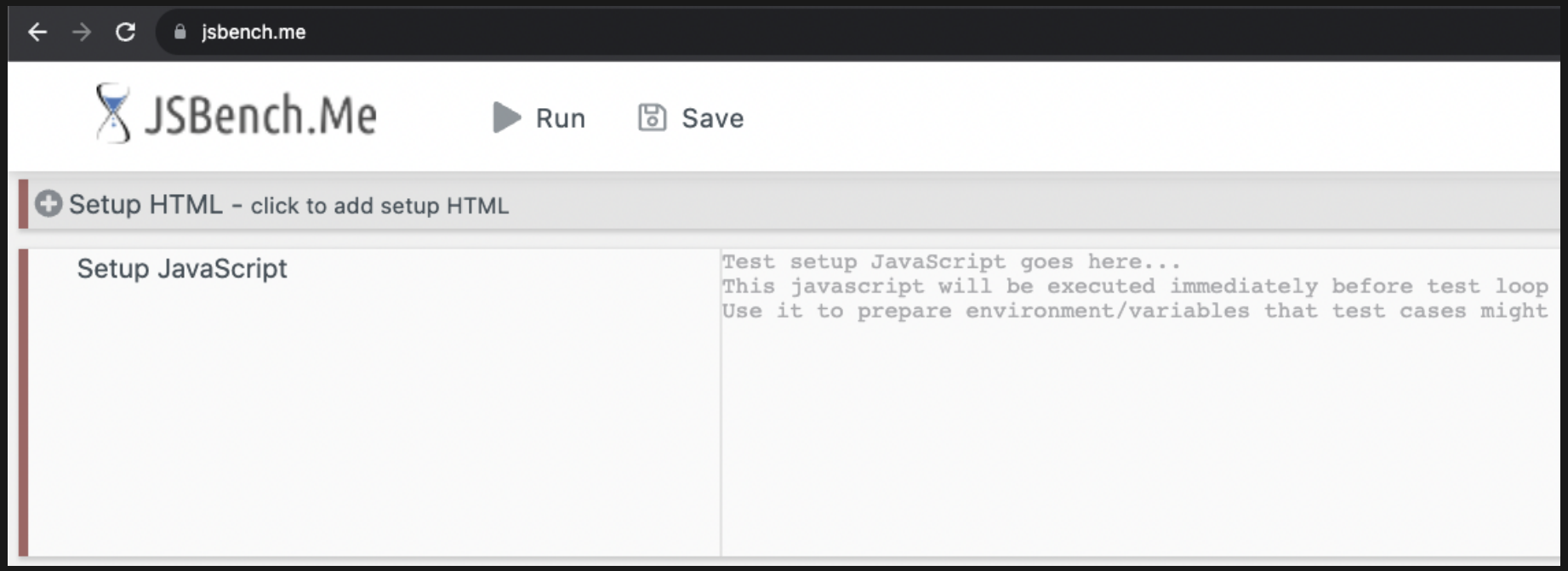
- Гранулярность измерения времени
- Сборка мусора
- JIT и прогрев кода
- Black Hole и удаление мёртвого кода
- Форма объектов и Inline Cache
- Реализации массивов и строк

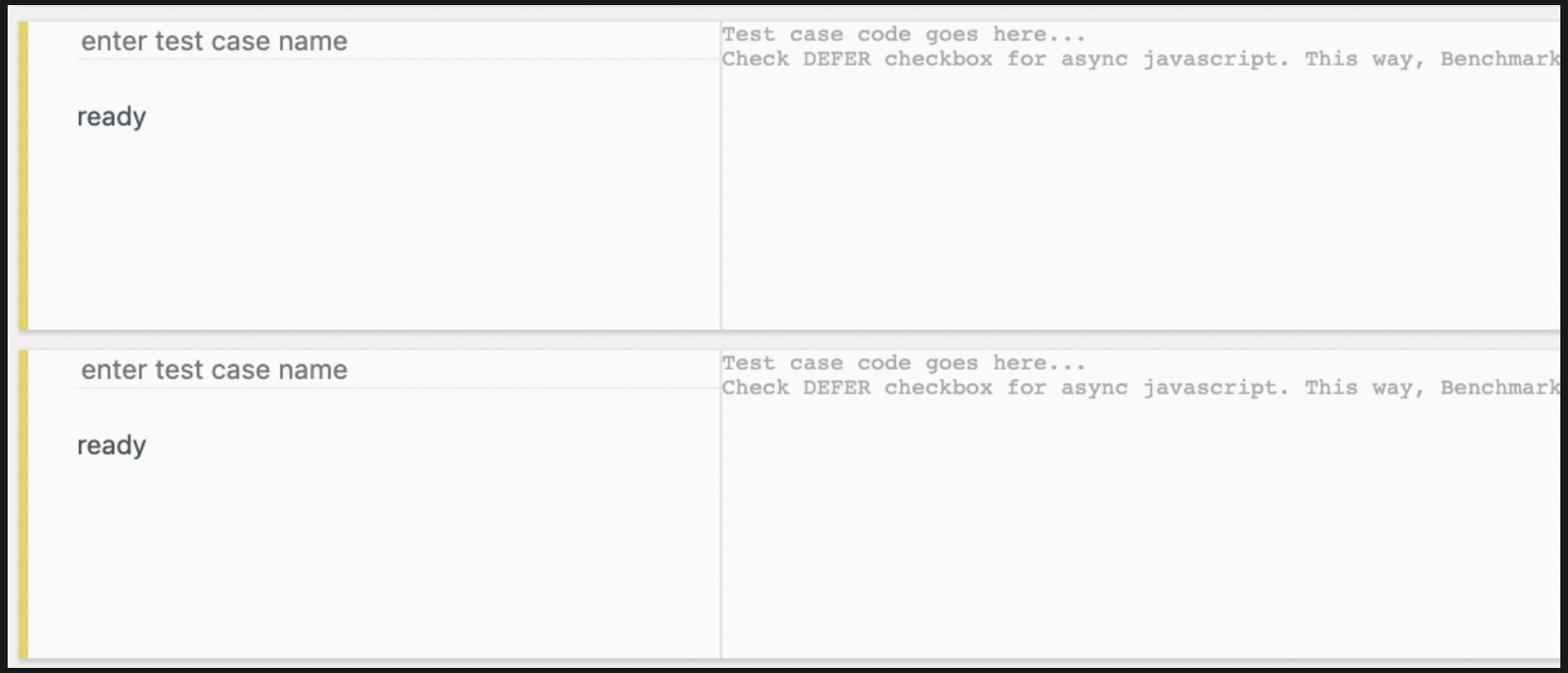
Бенчмарки

Бенчмарк браузерного кода

Код работает только в браузере

- jsperf.app
- jsbench.me


jsbench.me
JSBench.Me
Run
Save
Setup HTML - click to add setup HTML
Setup JavaScript
Test setup JavaScript goes here...
This javascript will be executed immediately before test loop
Use it to prepare environment/variables that test cases might


enter test case name
Test case code goes here...
Check DEFER checkbox for async javascript. This way, Benchmark
ready
enter test case name
Test case code goes here...
Check DEFER checkbox for async javascript. This way, Benchmark
ready

victor-homyakov-1

ready

```
bh += fizzbuzz1(N).length;
```

victor-homyakov-2

ready

```
bh += fizzbuzz2(N).length;
```


Test Case - click to add another test case
teardown JavaScript
if (Math.random() > 1) {
console.log(bh);
}
Output (DOM) - click to monitor output (DOM) while test is running
RUN

victor-homyakov-2

```
bh += fizzbuzz2(N).length;
```

finished

13 ops/s ± 5.06%
57.35 % slower

victor-homyakov-3

```
bh += fizzbuzz3(N).length;
```

finished

30 ops/s ± 2.77%
Fastest

Бенчмарк изоморфного кода

Код работает в браузере и Node.js

Если

- не важны Safari/Firefox
- не работаем с DOM/Canvas/Fetch/etc.

```
import * as React from 'react';
import {renderToString} from 'react-dom/server';
```

```
// Запускать с NODE_ENV=production
import * as React from 'react';
import {renderToString} from 'react-dom/server';
```

```
// Запускать как угодно
const React = require('react/cjs/react.production.min.js');
const {renderToString} = require(
    'react-dom/cjs/react-dom-server.node.production.min.js'
);
```

```
const data = require('./data-dump.json');

const SAMPLE_COUNT = 100;
const SAMPLES = [];
for (let i = 0; i < SAMPLE_COUNT; i++)
    SAMPLES.push(data[randomInt(0, data.length - 1)]);
```

```
for (let i = 0; i < data.length; i++) {
    if (i < SAMPLE_COUNT) {
        SAMPLES.push(data[i]);
    } else {
        // Random integer between 0 and i (inclusive)
        const r = randomInt(0, i);
        if (r < SAMPLE_COUNT) SAMPLES[r] = data[i];
    }
}
```

ru.wikipedia.org/wiki/Reservoir_sampling

```
import {Button1, Button2} from './buttons';

let bh = 0;

function button1() {
    for (const props of SAMPLES) {
        bh += renderToString(<Button1 ...props />).length;
    }
}

// function button2 – аналогично
```

```
import * as Benchmark from 'benchmark';
```

npmjs.com/package/benchmark

```
new Benchmark.Suite()
    .add('Button1', button1)
    .add('Button2', button2)
    .on('cycle', e => {
        // Выводим результат
        console.log(String(e.target));
        // Сайд-эффекты
        if (Math.random() > 1)
            console.log(bh);
    })
    .run({ async: true });
```

```
describe('Бенчмарк рендера', () => {
    it('Button1 vs Button2', done => {
        new Benchmark.Suite()
            // ...
            .on('complete', () => done()) // Завершаем тест
            .run({ async: true });
    }, 120000); // Время для выполнения бенчмарка
});
```

```
describe('Performance-тест', () => {
    it('Должен проходить', () => {
        bh = 0;
        // ... Здесь должен быть прогрев

        const start = performance.now();
        for (let i = 0; i < N; i++) {
            button1();
        }
        const time = performance.now() - start;

        // ... Здесь проверки
    });
});
```

```
describe('Performance-тест', () => {
    it('Должен проходить', () => {
        // ...

        // Время рендера одного компонента <1ms
        expect(time / N).toBeLessThan(1);
        // Sanity check: работали минимум секунду
        expect(time).toBeGreaterThan(1000);

        // Отрендеренный HTML занимает 100–200 байт
        expect(bh / N).toBeGreaterThan(100);
        expect(bh / N).toBeLessThan(200);
    });
});
```

```
const times = [];
do {
    const start = performance.now();
    for (let i = 0; i < 1_000; i++) {
        button1();
    }
    times.push(performance.now() - start);
} while (marginOfError(times) > MAX_ERROR);
```

After the initial 50 samples, tachometer will continue taking samples until there is a clear statistically significant difference between all benchmarks, for up to 3 minutes.

github.com/google/tachometer

Бенчмарк серверного кода

```
// my-func.benchmark.js
const myFunc = require('./my-func');
const data = require('./data-123456789.json');
console.log(myFunc(data[0]));
```

```
// my-func.benchmark.js
const Benchmark = require('benchmark');
const myFunc = require('./my-func');
const data = require('./data-123456789.json');

const SAMPLES = makeSample(data); // Reservoir sampling
let bh = 0; // Black hole

console.log('Warm up...');
for (let i = 0; i < 10; i++) {
    for (const sample of SAMPLES) {
        bh += myFunc(sample);
    }
}
```

Дальнейшие пути

- Бенчмарк, измерение длительности N операций
- Бенчмарк, сравнение скорости двух реализаций
- Написание performance-теста
- Профилирование

Бенчмарк:
измерение длительности N операций

```
console.log('Measure...');
const repeats = 1_000_000;
const start = process.hrtime().bigint();
for (let i = 0; i < repeats; i++) {
    for (const sample of SAMPLES) {
        bh += myFunc(sample);
    }
}
const timeNs = process.hrtime().bigint() - start;

const timeMs = timeNs / 1e6;
const timeUs = timeNs / 1e3;
console.log(
    'Time:', timeMs.toFixed(2), 'ms',
    'Speed:', (timeUs / repeats).toFixed(2), 'µs/operation'
);
```

Бенчмарк:
сравнение скорости двух реализаций

Сравнение скорости двух реализаций:

- Одинаковая сложность
- Разная сложность + фикс. размер

```
const myFunc = require('./myFunc');
```

```
const myFunc = require('./myFunc');

const myFuncOptimized = function(data) {
    // Копипастим реализацию myFunc
};
```

```
const myFunc = require('./myFunc');

const myFuncOptimized = function(data) {
    // Копипастим реализацию myFunc
};

// Sanity check
for (const d of data) {
    assert.deepStrictEqual(myFunc(d), myFuncOptimized(d));
}
```

```
const Benchmark = require('benchmark');
const suite = new Benchmark.Suite();
suite
    .add('myFunc', function() {
        for (const sample of SAMPLES)
            bh += myFunc(sample);
    })
    .add('myFuncOptimized', function() {
        for (const sample of SAMPLES)
            bh += myFuncOptimized(sample);
    })
```

```
for (let i = 1; i <= 2; i++) {
    suite
        .add(`myFunc ${i}`, function() {
            // ...
        })
        .add(`myFuncOptimized ${i}`, function() {
            // ...
        })
}
```

```
suite
    .on('cycle', function(event) {
        console.log(String(event.target));
    })
    .on('complete', function() {
        console.log(
            'Fastest is ' +
            this.filter('fastest').map('name')
        );
    })
    .on('error', function(e) {
        console.log(e);
    })
    .run({ 'async': true });
```

1. Вносим изменения в myFuncOptimized
2. Запускаем бенчмарк
3. Сравниваем скорости
4. Оставляем/откатываем изменение
5. GOTO 1

Бенчмарки: recap

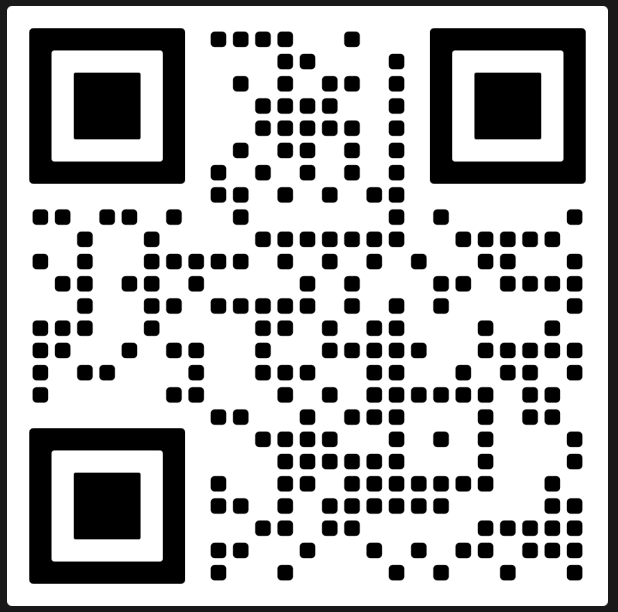
- Браузерный код — jsperf.app, jsbench.me
- Изоморфный/серверный код — node.js
- Бенчмарк → perf-тест
- Бенчмарк → профилирование

Резюме

Как писать хорошие бенчмарки

- Имеют ли смысл эти числа?
- Почему этот код работает так быстро?
- Почему этот код не может работать быстрее?
- Где в моём бенчмарке ошибка?

Вопросы?



https://tinyurl.com/5tjsj3c6