vs

# кто быстрее?

Виктор Хомяков, программист

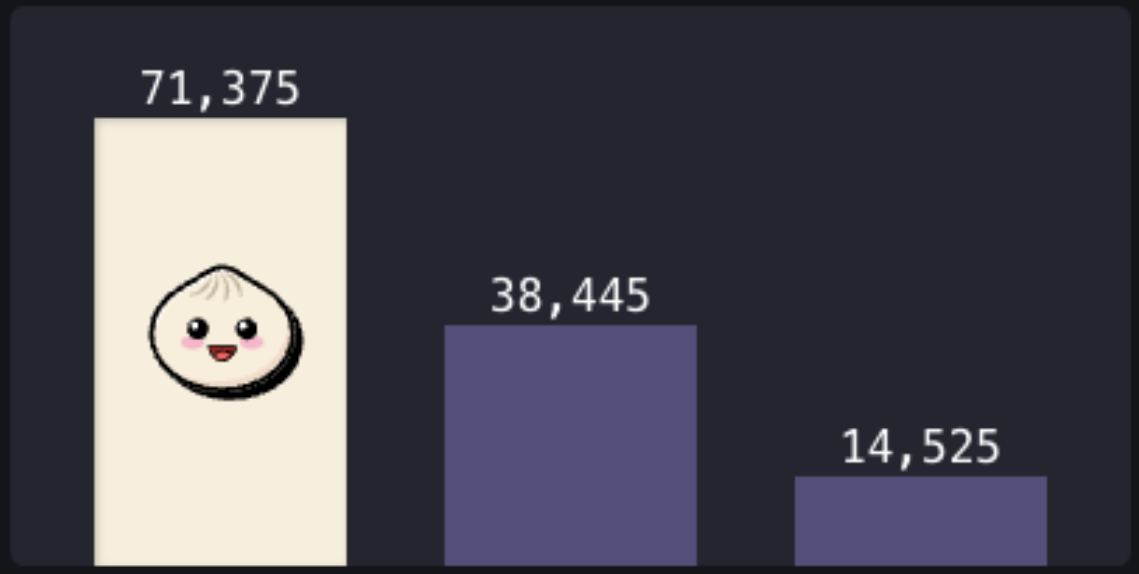
# Введение

- https://bun.sh/
- 08.09.2023 Bun 1.0
- ≈800 bugs, 3 crashes, 1 mem leak
- "improves reliability of fetch()"
- "fixes an 8x perf regression to Bun.serve()"
- 01.04.2024 Bun 1.1

- пакетный менеджер
- рантайм
- бандлер

## Server-side rendering React

HTTP requests per second (Linux x64)




| Bun | Deno.serve() | Node.js |
| --- | --- | --- |
| v1.1 | v1.42.1 | 21.7.1 |

## Node.js:

```
const App = () => (
  <html>
    <body>
      <h1>Hello World</h1>
    </body>
  </html>
);
```

## Deno и Bun:

```
const App = () => (
  <html>
    <body>
      <h1>Hello World</h1>
      <p>This is an example.</p>
    </body>
  </html>
);
```

- Node.js: `renderToPipeableStream()`
- Deno и Bun: `renderToReadableStream()`

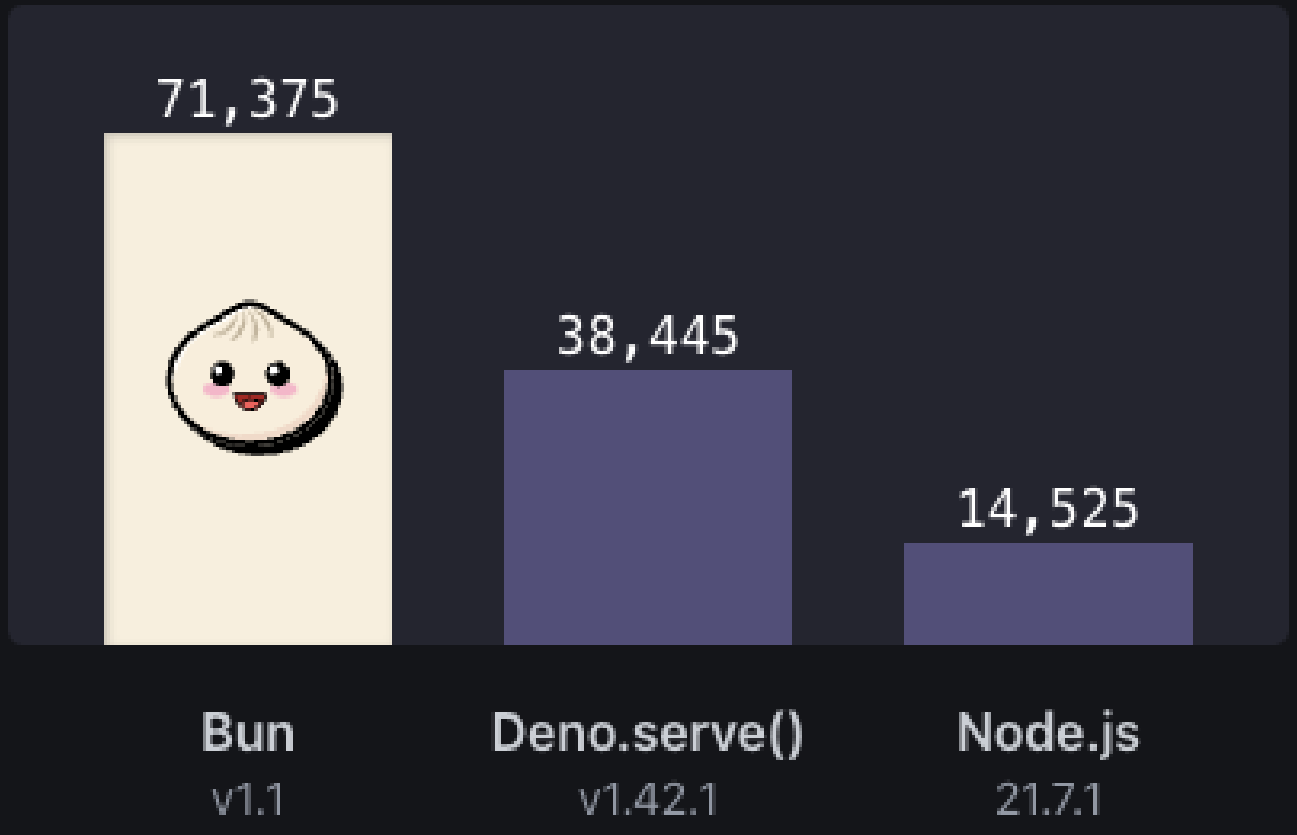
- Node.js: `http.createServer()`,
- Deno и Bun: свои `.serve()`,

- Node.js и Deno: в ответе два заголовка `Content-Type`, `Cache-Control`
- Bun: один заголовок `Content-Type`

- Node: заголовки создаются в JS "на лету" (megamorphic IC, slow)
- Bun, Deno: объект заголовков создан один раз

# Server-side rendering React

HTTP requests per second (Linux x64)



# SSR React

| | RPS | Δ |
|---|---|---|
| Node.js | 14,525 | 1x |
| Bun | 71,375 | 5x |

~~SSR React~~

# Накладные расходы на 1 запрос

| | µs | Δ |
|---|---|---|
| Node.js | 69 | |
| Bun | 14 | ~~5x~~ -55 |

# Бенчмарки из инета

Одни и те же "hello, world"


© "Пятый элемент"

# Клиентские сценарии

- Пакетные менеджеры
- CLI-инструменты (ESLint, precommit etc.)

# На чём измеряю

- MacBook Pro 16, Apple M1 Pro,
  8 производительных ядер на 3.2 ГГц
  2 энергоэффективных ядра на 2.06 ГГц
- MacBook Pro 16, Intel i7,
  6 ядер 2.6 ГГц
- Windows и Windows WSL не измерял

## Чем измеряю: `time`

- встроенная `bash time`
- встроенная `zsh time`
- `/usr/bin/time`

```
time some_command
some_command  45s user 50s system 190% cpu 50 total
```

# Чем измеряю: `hyperfine`

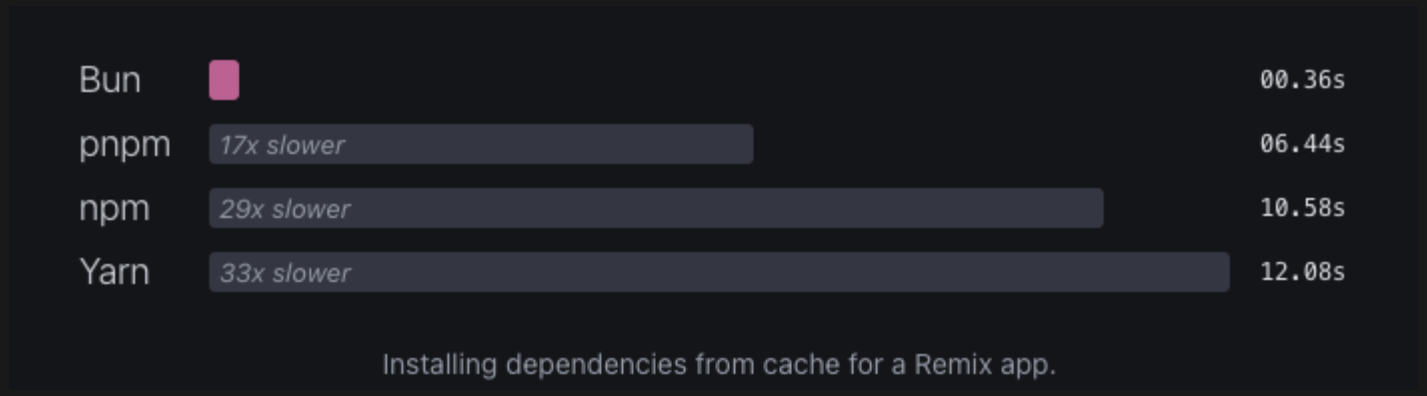
```
brew install hyperfine
```

```
hyperfine --warmup 3 'sleep 1' 'sleep 2'
Benchmark 1: sleep 1
  Time (mean ± σ):      1.006 s ±  0.002 s    [User: 0.001 s
  Range (min … max):    1.002 s …  1.008 s    10 runs

Benchmark 2: sleep 2
  Time (mean ± σ):      2.009 s ±  0.001 s    [User: 0.001 s
  Range (min … max):    2.006 s …  2.010 s    10 runs

Summary
  sleep 1 ran
    2.00 ± 0.00 times faster than sleep 2
```

# Пакетные менеджеры



# 3 сценария

| | **кэш** | **локфайл** | **node_modules** |
|---|---|---|---|
| первая установка | ⛔ | ⛔ | ⛔ |
| переустановка | ✅ | ✅ | ⛔ |
| валидация | ✅ | ✅ | ✅ |

- `everything` уже удалили
- `bloater` старый, не ставится
- свой package.json tinyurl.com/4jut2dj8
  35 dependencies и 48 devDependencies
- ~1660 пакетов, 3100..3800 package.json

# Bun

```
# первая установка
$ bun pm cache rm && time bun install
  2,23s user 7,30s system 47% cpu 20,174 total

# вторая первая установка
$ bun pm cache rm && rm -rf node_modules && time bun install
  1,44s user 13,90s system 103% cpu 14,827 total

# переустановка
$ rm -rf node_modules && time bun install
  0,01s user 2,92s system 79% cpu 3,712 total

# валидация
$ time bun install
  0,02s user 0,06s system 39% cpu 0,217 total
```

```
bun install --backend=copyfile
```

Замедление в сценарии "переустановка"

`--yarn` или `-y`

Дополнительный yarn-совместимый локфайл

Замедления нет

# NPM 8 - 9 - 10

```
# первая установка
$ npm cache clean --force && time npm install
  45,10s user 15,88s system 37% cpu 2:42,19 total

# переустановка
$ rm -rf node_modules && time npm install
  18,55s user 10,31s system 123% cpu 23,404 total

# валидация
$ time npm install
  2,93s user 0,29s system 21% cpu 14,672 total
```

`npm ci`

Эквивалентна сценарию "переустановка"

```
npm set progress=false
```

Ускорение от 0 до 25%

# PNPM 8

```
# первая установка
$ pnpm store prune && time pnpm install
  16,74s user 27,39s system 90% cpu 48,895 total

# переустановка
$ rm -rf node_modules && time pnpm install
  4,19s user 15,18s system 417% cpu 4,636 total

# валидация
$ time pnpm install
  1,05s user 0,32s system 174% cpu 0,779 total
```

`pnpm --prefer-offline`

Ускорение ~2x в сценарии "первая установка"

# Yarn 1

```
# первая установка
$ yarn cache clean && time yarn
  28,19s user 29,16s system 41% cpu 2:16,58 total

# переустановка
$ rm -rf node_modules && time yarn
  7,35s user 19,86s system 185% cpu 14,662 total

# валидация
$ time yarn
  0,49s user 0,04s system 168% cpu 0,314 total
```

| Пакетный менеджер | Первая установка, с | Переустановка, с | Валидация, с | node_modules, МБ | Лок файл, кБ |
|---|---|---|---|---|---|
| Bun | ❄️ 50<br>🔥 14 | 1,5-4 | 0,2 | 1100 ❗ | 680 |
| PNPM prefer-offline | ❄️ 50<br>🔥 24 | 4,5 | 0,8 | 487 | 456 |
| Yarn | ≈ 140 | 15 | 0,3 | 618 | 570 |
| NPM progress false | ❄️ 160<br>🔥 30 | 22 | 2 | 515 | 1680 |

## Выводы

- TODO Windows и Windows WSL
- Первая установка в Bun быстрее

## Проблемы Bun

- node_modules в 2 раза больше
- бинарный локфайл
- локфайлы PNPM и Yarn компактнее

# Проблемы PNPM

## Скорость сравнима с Bun за счёт 4x CPU

```
# Первая установка
bun pm cache rm && time bun install
  2,23s user 7,30s system 47% cpu 20,174 total
pnpm store prune && time pnpm install --prefer-offline
  14,68s user 36,59s system 214% cpu 23,857 total

# Переустановка
rm -rf node_modules && time bun install
  0,01s user 2,92s system 79% cpu 3,712 total
rm -rf node_modules && time pnpm install
  4,19s user 15,18s system 417% cpu 4,636 total
```

# Скорость старта минимального приложения

```
console.log('Hello World')
```

```
hyperfine --warmup 20 \
  'bun hello-world.js' \
  'node hello-world.js'
```

# Результат на MacBook M1:

```
Benchmark 1: bun hello-world.js
  Time (mean ± σ):       8.6 ms ±   0.6 ms    [User: 5.1 ms,
  Range (min … max):     7.4 ms …  13.1 ms    284 runs

Benchmark 2: node hello-world.js
  Time (mean ± σ):      20.1 ms ±   0.7 ms    [User: 16.0 ms
  Range (min … max):    19.1 ms …  23.2 ms    126 runs
```

# Результат на MacBook i7:

```
Benchmark 1: bun hello-world.js
  Time (mean ± σ):      21.5 ms ±   1.8 ms    [User: 13.2 ms
  Range (min … max):    16.0 ms …  28.3 ms    140 runs

Benchmark 2: node hello-world.js
  Time (mean ± σ):      45.5 ms ±   4.3 ms    [User: 34.7 ms
  Range (min … max):    37.6 ms …  53.0 ms    69 runs
```

# Скорость старта с зависимостями

```
require("@nestjs/axios");
// ...
require("lodash");
// ...
require("rxjs");
// ...
console.log("Loaded");
```

- Скорость старта Bun и Node.js
- Зависимость от структуры node_modules

```
$ hyperfine --warmup 3 \
  'bun modules-bun/index.js' \
  'bun modules-npm/index.js' \
  'bun modules-pnpm/index.js' \
  'bun modules-yarn/index.js' \
  'node modules-bun/index.js' \
  'node modules-npm/index.js' \
  'node modules-pnpm/index.js' \
  'node modules-yarn/index.js'
```

# Длительность старта, мс

| | **bun** | **npm** | **pnpm** | **yarn** |
|---|---|---|---|---|
| bun M1 | 835 | 838 | 885 | 847 |
| node M1 | 577 | 585 | 656 | 577 |
| bun i7 | 1289 | 1280 | 1394 | 1293 |
| node i7 | 1144 | 1193 | 1290 | 1171 |

# Скорость старта и выполнения сложного приложения

- приложение на JS с плагинами
- читает много файлов (IO)
- обрабатывает файлы (CPU)
- проверка проекта в ESLint

# Небольшой проект

```
find . \( -type f -name "*.ts" -o -name "*.tsx" \) | wc -l
     270
```

```
hyperfine --warmup 3 --ignore-failure \
  'npx eslint --ext .ts,.tsx .' \
  'bunx eslint --ext .ts,.tsx .'
```

# ESLint 270 files TS+TSX

| | **Длительность, с** |
|---|---|
| bun M1 | 6.776 ± 0.046 |
| node M1 | 6.896 ± 0.047 |
| bun i7 | 14.517 ± 0.075 |
| node i7 | 15.208 ± 0.278 |

# Большой проект

- десятки тысяч файлов
- миллионы строк кода

```
find . \( -type f -name "*.ts" -o -name "*.tsx" \) | wc -l
   31828
```

```
time node node_modules/eslint/bin/eslint.js ./src/
  612,15s user 15,75s system 122% cpu 8:31,52 total

time bun node_modules/eslint/bin/eslint.js ./src/
  455,24s user 19,13s system 116% cpu 6:47,51 total
```

- 10-100 файлов: bun быстрее ~0.1-0.7 с ~1-5%
- 30000 файлов: bun быстрее ~100 с ~20%
- RAM: разницы нет, ~1 ГБ
- CPU: разницы нет, ~100% (одно ядро)
- На сервере/в облаке требования к железу не поменяются

# Скорость парсинга лог-файла

- Статья на Хабре tinyurl.com/2hjxbhza
- 4 варианта кода
- Эталонный файл лога на 6 МБ

# Медиана времени парсинга, мс

| Вариант | Bun M1 | Node M1 | Bun i7 | Node i7 | Bun AMD | Node AMD |
|---|---|---|---|---|---|---|
| только чтение | 9 | 20 | 16 | 28 | 16 | 31 |
| `String.slice` `String.split` | 102 | 174 | 165 | 311 | 199 | 436 |
| `RegExp.match` `Array.reduce` | ! 81 | 76 | ! 144 | 130 | 177 | 171 |
| `RegExp.match` `for ()` | 35 | 71 | 62 | 119 | 73 | 160 |
| state machine | 18 | 53 | 30 | 84 | 33 | 110 |

# Серверные сценарии

- Скорость React SSR

# Кратко про нагрузочное тестирование

Основные цели:

- RPS разладки (RPS breakpoint)
- какой RPS выдержим на данном железе?
- сколько железа надо для заданного RPS?

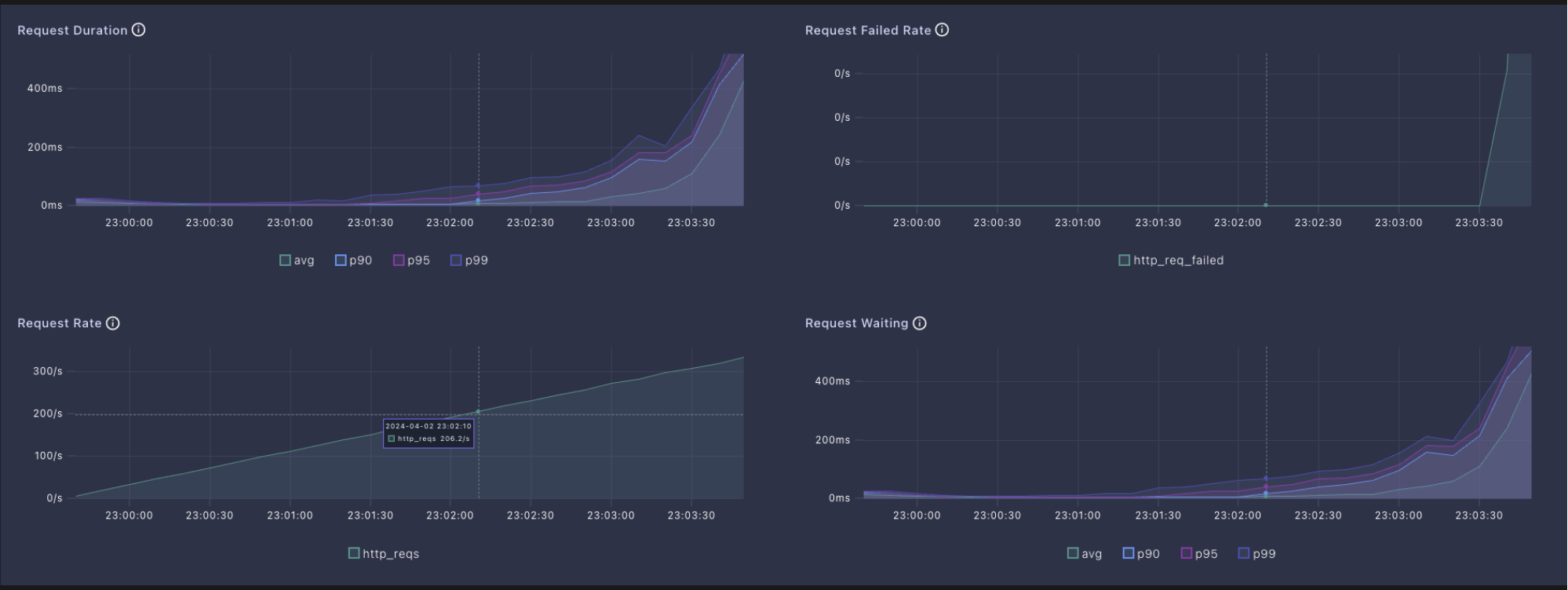
Дополнительные цели:

- как RPS влияет на TTFB, TTLB, ELL, WebVitals
- как стабильно сервис держит нагрузку
- есть ли утечки под нагрузкой
- как выглядит сервис при перегрузке (fallback)

# Чем измеряю

| | |
|---|---|
| npm i -g autocannon | Matteo Collina |
| k6 k6.io | Grafana |

# k6 run script.js --out web-dashboard


Request Duration
400ms
200ms
0ms
23:00:00
23:00:30
23:01:00
23:01:30
23:02:00
23:02:30
23:03:00
23:03:30
avg
p90
p95
p99
Request Failed Rate
0/s
0/s
0/s
0/s
23:00:00
23:00:30
23:01:00
23:01:30
23:02:00
23:02:30
23:03:00
23:03:30
http_req_failed
Request Rate
300/s
200/s
100/s
0/s
2024-04-02 23:02:10
http_reqs 206.2/s
23:00:00
23:00:30
23:01:00
23:01:30
23:02:00
23:02:30
23:03:00
23:03:30
http_reqs
Request Waiting
400ms
200ms
0ms
23:00:00
23:00:30
23:01:00
23:01:30
23:02:00
23:02:30
23:03:00
23:03:30
avg
p90
p95
p99

# На чём измеряю

- MacBook Pro 16, Apple M1 Pro,
  8 производительных ядер на 3.2 ГГц
  2 энергоэффективных ядра на 2.06 ГГц
- MacBook Pro 16, Intel i7,
  6 ядер 2.6 ГГц
- виртуалка Ubuntu Jammy 2022,
  4 ядра на AMD EPYC 7713 64-Core

# Скорость React SSR

Ещё раз вспомним бенчмарк с сайта Bun

- bun 71,375 RPS
- node 14,525 RPS
- разный код
- разные `React.render*()`
- разные заголовки

# Node: 242 байта, response body 46 байт

```
HTTP/1.1 200 OK
Content-Type: text/html
Cache-Control: no-transform
Date: Wed, 01 Apr 2024 12:34:56 GMT
Connection: keep-alive
Keep-Alive: timeout=5
Transfer-Encoding: chunked

<html><body><h1>Hello World</h1></body></html>
```

# Bun: 202 байта, response body 72 байта

```
HTTP/1.1 200 OK
Content-Type: text/html
Cache-Control: no-transform
Date: Wed, 01 Apr 2024 12:34:56 GMT
Content-Length: 72

<html><body><h1>Hello World</h1><p>This is an example.</p></
```

Делаем максимально одинаково

- компонент React App
- `renderToString()`
- размер заголовков 189 байт
- размер ответа 261 байт

```
Bun.serve({
  fetch(req) {
    const body = renderToString(createElement(App));
    return new Response(body, {
      headers: {
        "Content-Type": "text/html",
        "Cache-Control": "no-transform",
        "Ballast": "abcdefgqwertyuiopasdfghjkl;zxcvbnm,./123
      },
    });
  },
});
```

```
const server = http.createServer((req, res) => {
  res.setHeader("Content-Type", "text/html");
  res.setHeader("Cache-Control", "no-transform");
  res.setHeader("Ballast", "a");
  const body = renderToString(createElement(App));
  res.end(body, "utf8", undefined);
});
```

```
require("uWebSockets.js")
  .App()
  .get("/*", (res, req) => {
    res
      .writeStatus("200 OK")
      .writeHeader("Content-Type", "text/html")
      .writeHeader("Cache-Control", "no-transform")
      .writeHeader("Ballast", "qwertyuiopasdfghjkl;zxcvbnm,.
      .end(renderToString(createElement(App)));
  })
```

```
NODE_ENV=production bun server.bun-serve.js
NODE_ENV=production bun server.http.js

NODE_ENV=production node server.http.js
NODE_ENV=production node server.uws.js
```

# RPS на Mac i7

| | autocannon RPS | autocannon CPU |
|---|---|---|
| Bun serve | 25-35k | 100% |
| Bun http | 25-29k | 110% |
| Node http | 19-21k | 100% |
| Node uWS | 25-35k | 100% |

# RPS на Mac M1

| | autocannon RPS | autocannon CPU | k6 RPS br. |
|---|---|---|---|
| Bun serve | 89-95k | 80% | 53k |
| Bun http | 65-75k | 90% | 36k |
| Node http | 50-55k | 110% | 30k |
| Node uWS | 92-98k | 75% | 47k |

# RPS на Ubuntu AMD EPYC

| | autocannon RPS | autocannon CPU | k6 RPS br. |
|---|---|---|---|
| Bun serve | 48-50k | 40% | 10k |
| Bun http | 32-36k | 60% | 10k |
| Node http | 22-35k | 100% | 9k |
| Node uWS | 48-50k | 40% | 9k |

| | Фреймворк | RPS cluster 64 |
|---|---|---|
| node | uwebsockets 20.30 | 167_023 |
| bun | 0http-bun 1 | 121_294 |
| go 1.22 | gin 1.9 | 104_787 |
| node | 0http 3.5 | 79_149 |
| node | fastify 4.26 | 62_417 |
| java 11 | spring 3.2 | 49_924 |
| node | express 4.19 | 24_423 |
| php 8.3 | laravel 11.2 | 2_635 |
| python 3.12 | flask 3 | 1_777 |
| python 3.12 | django 5 | 1_515 |

https://tinyurl.com/25874kym

# Throughput vs Latency

**Throughput: $T_{io} + T_{framework} + T_{code}$**

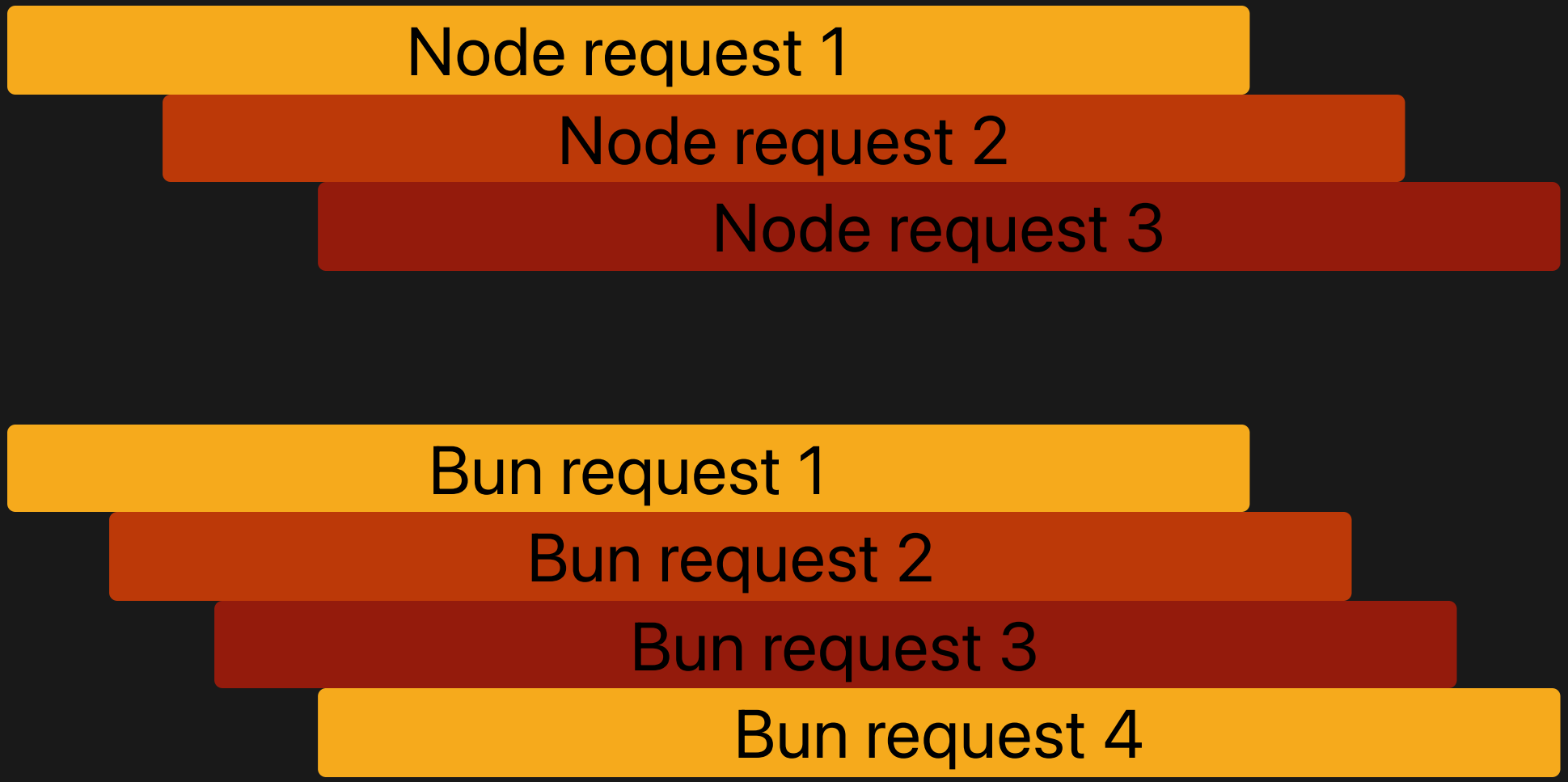
**Latency: $T_{code} + T_{backend}$**

$T_{io}$ - сеть, файловая система

$T_{framework}$ в Node на JS

$T_{framework} + T_{code}$ ~ CPU

# Throughput vs Latency



# Заключение

- бенчмарки сложно писать
- бенчмарки сложно интерпретировать
- обещанного 5x преимущества не видно
- переход на Bun может дать замедление
- проверяйте на своём реальном проекте

# Вопросы?

https://tinyurl.com/yc396s2w