HOLYJS 2026 AUTUMN

V8 для самых маленьких

Что за лев этот турболев?

Александр Зайцев

X5 Tech


ХОРОШО ЧТО ТЫ ТУТ
ЭТА ГРУППА ДЛЯ КРУТЫХ

Александр Зайцев

Разбираюсь, как JavaScript работает внутри V8

Сегодня — на примере двух версий одной функции



01 Две версии одной функции

Три строчки кода → разные формы объектов

Почему потом читать поля дороже

О чём поговорим

02 Что может компилятор

Как функция прогревается и меняет ярус

Сколько разницы остаётся в HTTP-сервисе

03 Немного истории V8

full-codegen и Crankshaft → Ignition и TurboFan

Maglev, Turboshaft и тот самый Turbolev

04 Что проверить у себя

Сколько форм приходит в одно место кода

Команды, замеры и примеры в репозитории



давай!! ДАВАААЙ УРААА
ДАВААЙ ДАААВАААЙ

ДАВАААААЙ

АВААВАВВВВВВАААААААААа

Три строчки — два способа собрать объект

A · поле есть всегда

```
referrer: raw.referrer ?? null,
trialUntil: raw.trialUntil ?? null,
teamId: raw.teamId ?? null,
```

B · поле может отсутствовать

```
...(raw.referrer && { referrer: raw.referrer }),
...(raw.trialUntil && { trialUntil: raw.trialUntil }),
...(raw.teamId && { teamId: raw.teamId }),
```

В этом примере читатель даёт одинаковый результат. null и отсутствие поля — разные вещи.

А время чтения отличается в шесть раз

A

0.12

мс за проход

B

0.79

мс за проход

×6.6

50 000 готовых объектов
Код прогрет
Медианы пяти прогонов

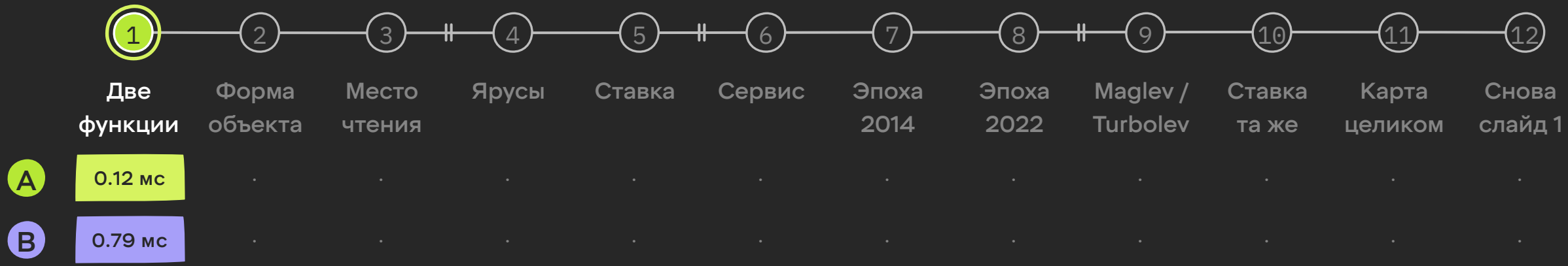
```
A: p50_ms 0.11 · correct true    B: p50_ms 0.76 · correct true
```

Создание объектов в замер не входит. Внизу — один прогон; крупные числа — медианы пяти.

Улика и пока пустая карта

```
$ node --allow-natives-syntax --trace-deopt demo-wrongmap.js ref-b.js | grep "JSFunction score"
[bailout (kind: deopt-eager, reason: wrong map): begin. deoptimizing <JSFunction score>, <Code TURBOFAN_JS>
$ node --allow-natives-syntax --trace-deopt demo-wrongmap.js ref-a.js | grep "JSFunction score"
(пусто)
```

У B — wrong map. У A — пусто. Причину ещё предстоит разобрать.



A: 0.12 мс → ?

B: 0.79 мс → ?

Форма объекта

Посмотрим, что V8 видит в наших двух вариантах.

Что такое форма

hidden class
В трейсе — Map

Внутреннее описание объекта: какие поля есть и в каком порядке они добавлены.

Одинаково собранные объекты могут делить одно описание.

Три необязательных поля → восемь форм

A · ПОЛЯ ЕСТЬ ВСЕГДА

1

referrer, trialUntil, teamId
Значение или null

B · ПОЛЯ ПО УСЛОВИЮ

—	teamId
trialUntil	trialUntil + teamId
referrer	referrer + teamId
referrer + trialUntil	все три

Восемь комбинаций присутствуют в тестовых данных: $2^3 = 8$.


ФУ
покажи

%HaveSameMap: проверяем предположение

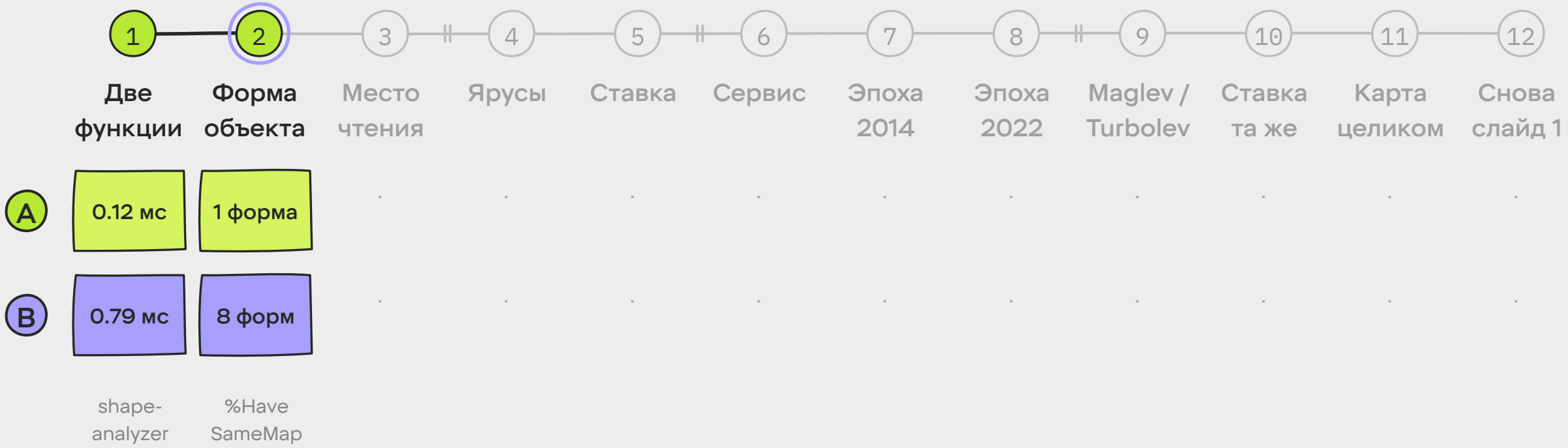
```
$ node --allow-natives-syntax demo-samemap.js
A: true
B: false
форм у B: 8
```

A: true
B: false

Одинаковая ли форма
у двух объектов?

Карта: первая развилка

У A одна форма. У B восемь. Дальше проследим, что из этого следует.



Дальше — кто платит за восемь форм: сборка или чтение.

|| передышки после 3, 5 и 8

За что мы платим

За создание объекта или за чтение его полей?

Дороже обходится именно чтение



Сборка через if
×0.93–1.02

Сборка через спред
×1.5–1.7

Чтение обоих B
×5.1–5.4

В относительно A. 150 прогонов; цену спреда отделяем от цены разнообразия форм.

Место чтения запоминает формы

MONO

Одна форма

Одна проверка,
готовое смещение

1 форма: ×1

POLY

Небольшой набор

Несколько проверок
по короткому списку

4 формы: ×1.3

MEGA

Много форм

Общий механизм
поиска поля

8 форм: ×2.7

Inline cache относится к месту кода. Эти множители — отдельный замер одного поля, Node 24.

В журнале: 1 → P → N

```
$ node --allow-natives-syntax --log-ic --no-logfile-per-isolate \
    --logfile=b.log shape-analyzer.js ref-b.js > /dev/null
$ grep ^LoadIC b.log | grep ',30,9,' | cut -d, -f1,4-9
LoadIC,30,9,0,1,0x3093ac1963b1,plan
LoadIC,30,9,1,P,0x3093ac195f81,plan
LoadIC,30,9,P,P,0x3093ac196279,plan
LoadIC,30,9,P,P,0x3093ac196011,plan
LoadIC,30,9,P,N,0x3093ac196b49,plan
LoadIC,30,9,N,N,0x3093ac196c91,plan
LoadIC,30,9,N,N,0x3093ac1964b9,plan
LoadIC,30,9,N,N,0x3093ac196159,plan
$ node --allow-natives-syntax --log-ic --no-logfile-per-isolate \
    --logfile=a.log shape-analyzer.js ref-a.js > /dev/null
$ grep ^LoadIC a.log | grep ',30,9,' | cut -d, -f1,4-9
LoadIC,30,9,0,1,0x3c9e6d815ff9,plan
```

1
Одна форма

P
Несколько

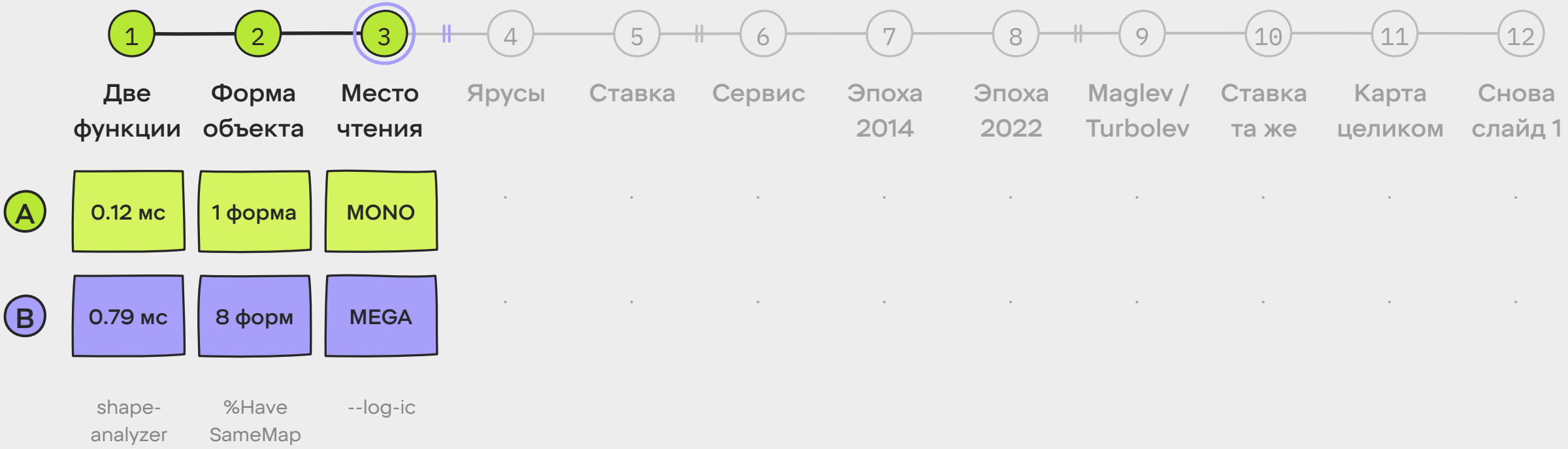
N
Общий случай

А компилятор поможет?

И что будет, если читать эти объекты чаще?

Где мы · форма и чтение

С формой и чтением разобрались.



Дальше — что поверх этого делает компилятор.

|| передышки после 3, 5 и 8

Как функция разгоняется

Одного прогрева достаточно, чтобы B стал быстрым?

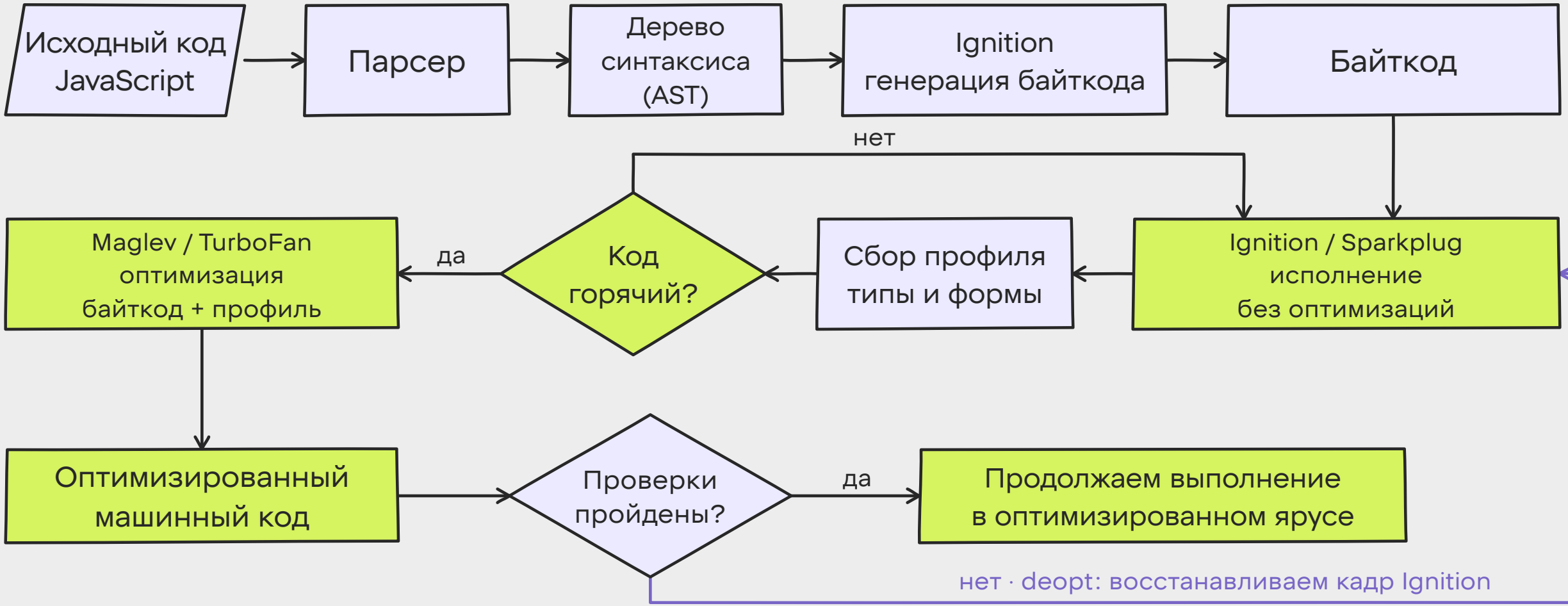
VW МАШИНКА



ВРУМ ВРУМ КРУТАЯ

От исходника до выполнения

Сборка проекта, если нужна, идёт до V8



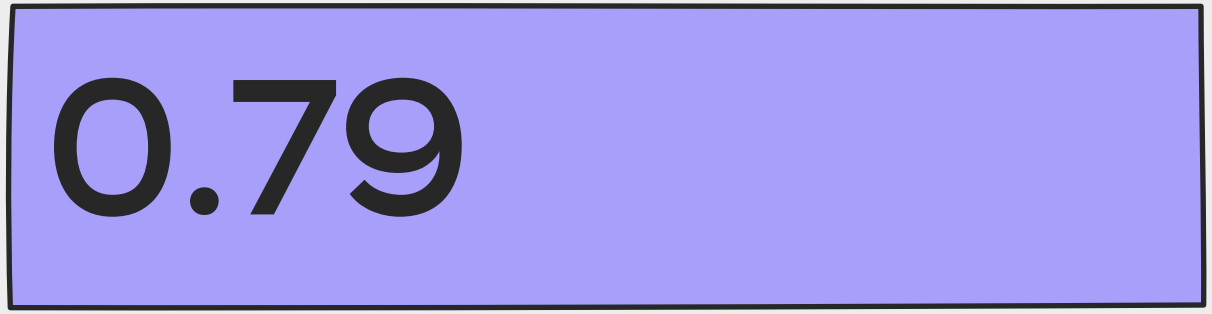
Оба читателя уже оптимизированы

A



мс за проход

B

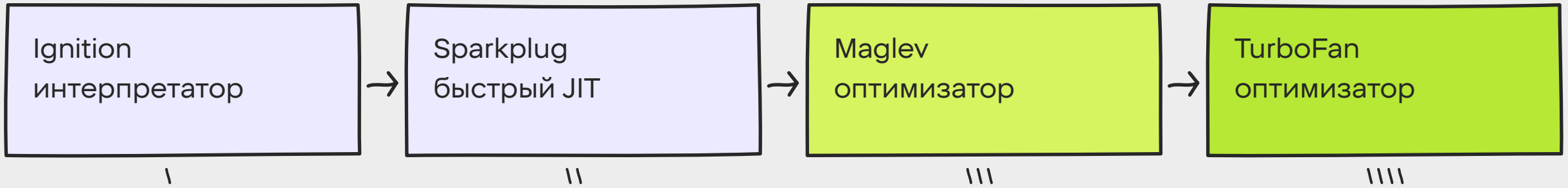


мс за проход

Горячий цикл. Оптимизирующий ярус проверен у обеих версий. Просто подождать недостаточно.


не думайте

Ярус — способ исполнить функцию

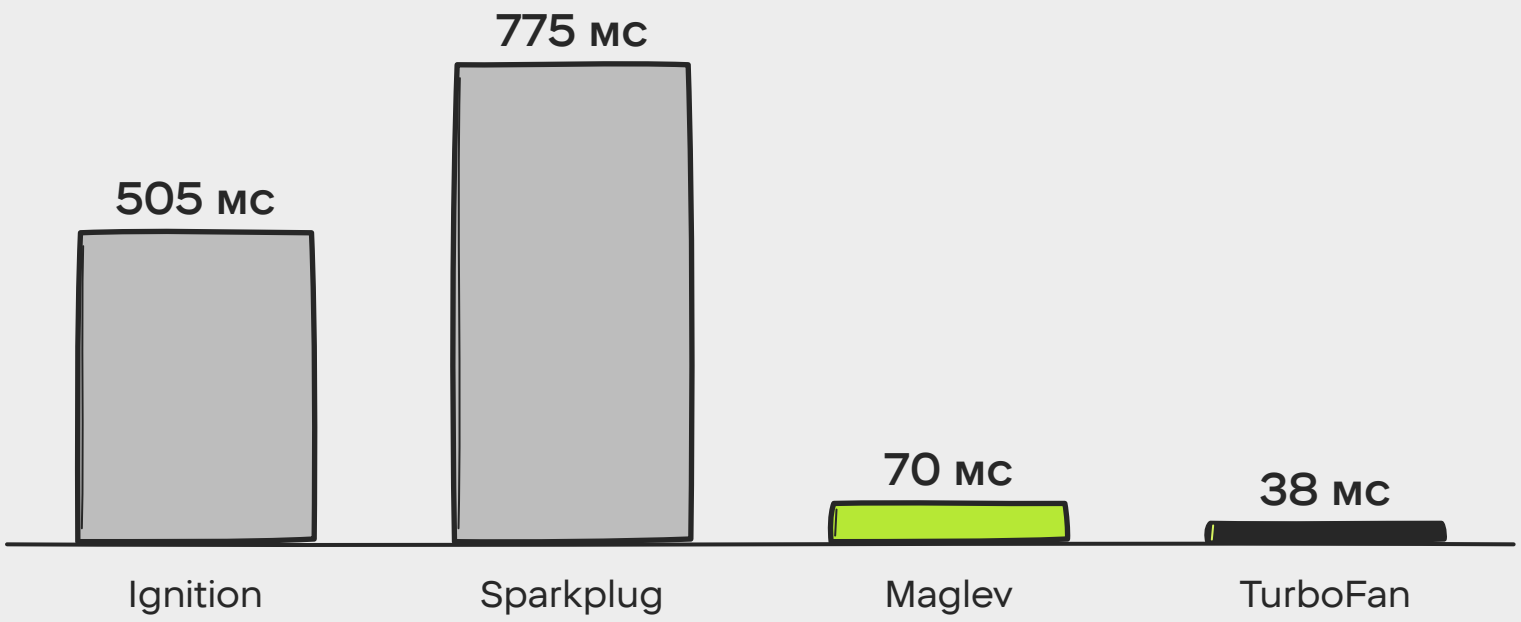


Один вызов → десятки → примерно тысяча → десятки тысяч

```
$ node --print-bytecode --print-bytecode-filter=normalizeUser ref-a.js
  110 S> 0x34e3243c40 @    0 : 85 00 00 29       CreateObjectLiteral [0], [0], #41
  131 E> 0x34e3243c45 @    5 : 33 03 01 01       GetNamedProperty a0, [1], [1]
         0x34e3243c49 @    9 : 3a f9 01 03       DefineNamedOwnProperty r0, [1], [3]
```

Ориентиры из нашего стенда Node 24, не фиксированные пороги для любой функции.

Почему не сразу TurboFan



×13.5

Один код, разные ярусы
--max-opt=0 / 1 / 2 / 3

Оптимизация требует времени и наблюдений

Наш стенд arm64. Sparkplug в этом замере медленнее Ignition; это не универсальная лестница скорости.


думайте

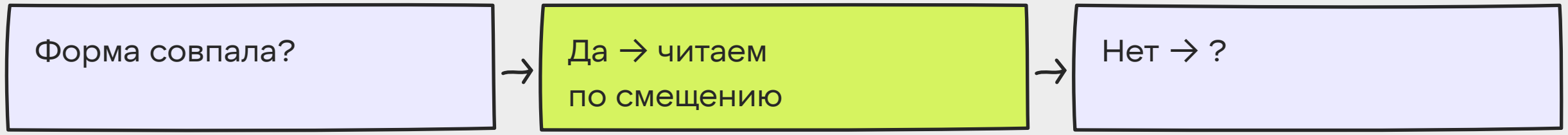
Компилятор опирается на то, что видел

Ignition / Sparkplug

Используют обратную связь
при чтении поля

Maglev / TurboFan

Могут подготовить код
под увиденные формы

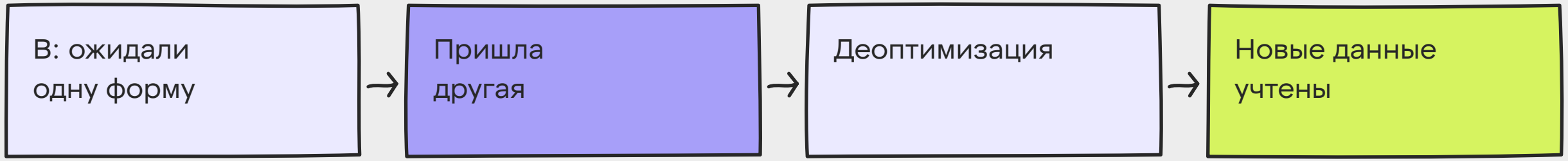


Пришла другая форма

Функцию уже оптимизировали. Что теперь делать с подготовленным кодом?

Вот это и есть wrong map

A · поле осталось, внутри null → форма та же



`reason: wrong map`

Фильтруем лог по читателю score(). В normalizeUser могут быть свои деоптимизации.

Ноль деоптов — ещё не значит быстро

0.79 мс

Стабильная цена общего чтения

В отдельном демо

Сначала одна форма, затем другая.
Один откат и новая оптимизация.

В первом бенчмарке

Сразу все восемь форм.
Ноль откатов, но те же ×6.6.


ПОМНИТЕ ЕГО?
ЭТО ОН

Теперь лог можно прочитать

```
reason: wrong map
```

map

Внутренняя форма объекта

wrong

Не совпала с ожидаемой

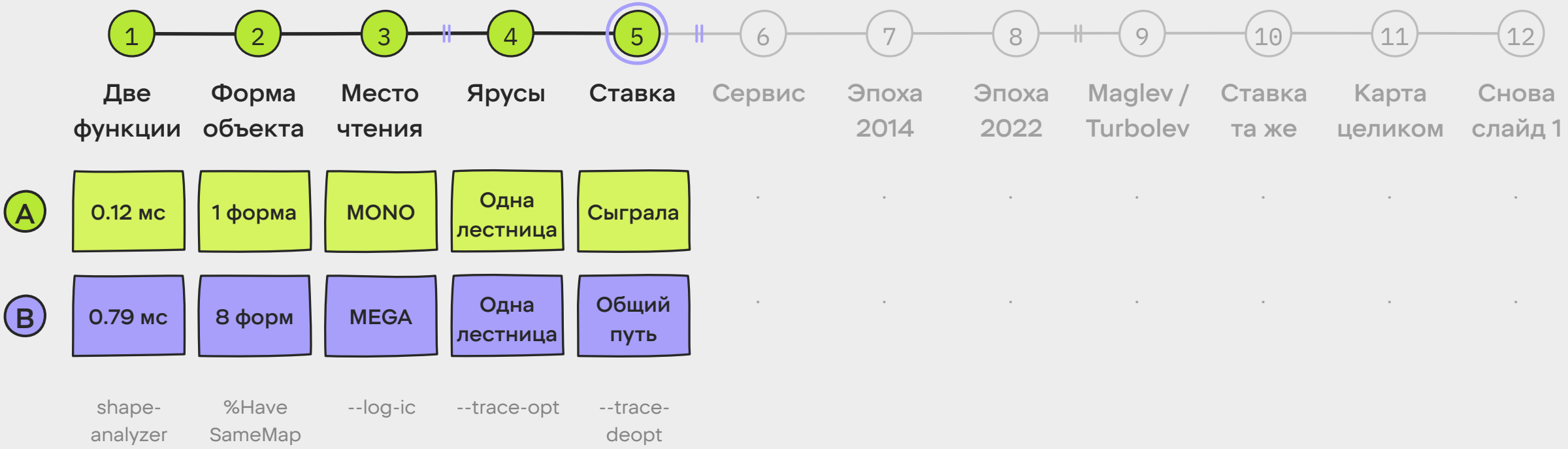
2008

Формы и IC уже были в первом V8



Где мы · путь объекта

Формы, чтение и компиляцию связали. Теперь — в приложение.



Дальше — HTTP, потом три эпохи V8.

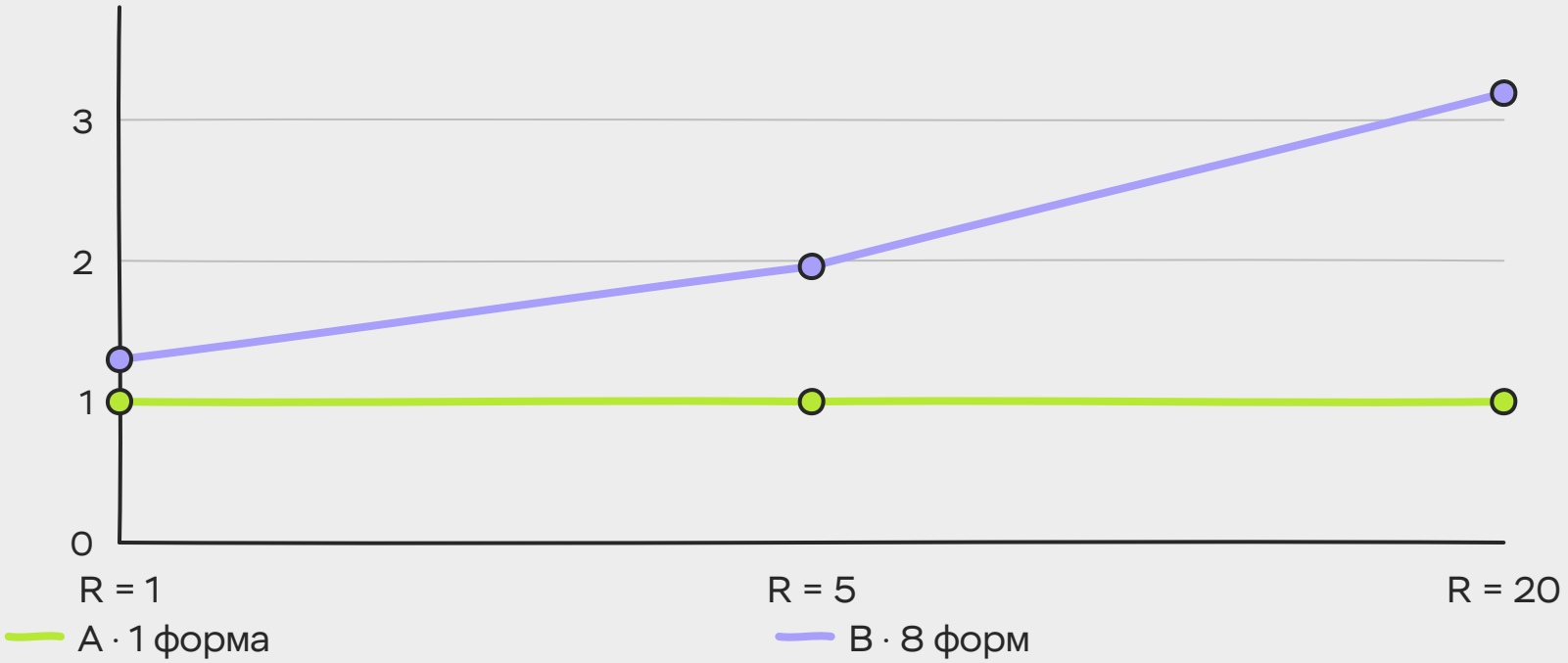
|| передышки после 3, 5 и 8


Хмели сумели
и ты сможешь

А что в сервисе?

Добавим парсинг, обработку данных и ответ по HTTP.

Чем чаще читаем, тем заметнее разница



R = 1 +29–31%

R = 5 ×1.95–1.98

R = 20 ×3.1–3.3

200k NDJSON: parse → normalize → consume × R → serialize. В этом замере базы нет.

Не спешим переписывать все объекты

Почти не читаем

R ≈ 0: разница меньше

if: +6–10%
спред: +13–18%

Оценка по стадиям

Ждём базу

Доли миллисекунды CPU
могут потеряться
на фоне ожидания

Зависит от профиля

Маленькая пачка

500 записей, R = 1

0.17 / 0.22 мс

Проценты заметные,
абсолютная разница мала

Сначала профиль приложения и реальные чтения. Потом решение, нужно ли что-то менять.

Как мерили — и где легко ошибиться

Одинаковые условия

Docker, одна версия Node
Отдельный процесс
Прогрев, повторы, медианы
Контрольные суммы

Что портит замер

Общий процесс для кейсов
Трейс без фильтра читателя
Эмуляция другой архитектуры
Помехи на старте

Проверка масштаба

Чтение готовых объектов
↓
Пайплайн обработки
↓
HTTP под нагрузкой

Резерв по методике. Микробенчмарк и HTTP отвечают на разные вопросы.

V8 в 2014

Четырёх ярусов ещё нет. Посмотрим, зачем потом меняли обе ступени.

Формы и inline cache старше V8

1984

Smalltalk-80
Inline cache

1989 → 1991

Self
Скрытые классы
Несколько форм на место

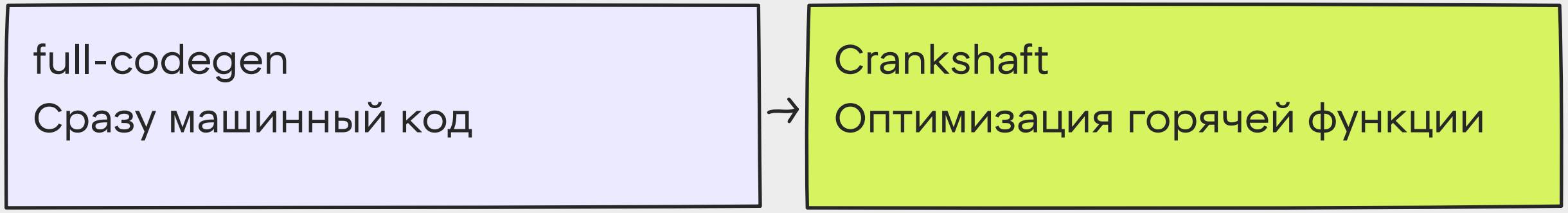
2008

Первый V8
Формы и IC уже есть

Сначала научились запоминать объекты, с которыми работает код. Потом усложняли оптимизацию.



Две ступени: full-codegen и Crankshaft



Обратная связь уже есть.
Crankshaft использует её, чтобы оптимизировать код под формы.

Зачем заменили обе ступени

Меньше памяти

Машинный код хранится
для каждой вызванной функции.
Байткод компактнее.

Больше возможностей

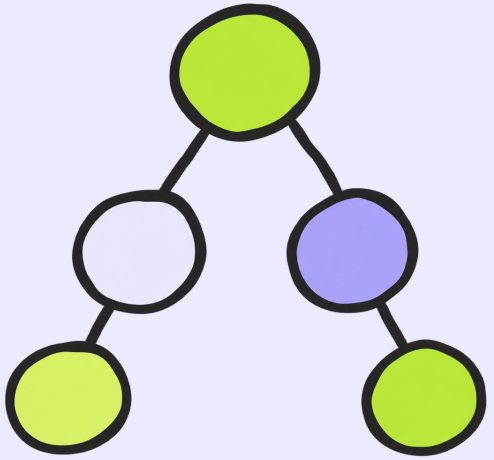
Crankshaft не оптимизировал
часть функций, например
с try/catch.

2017: Ignition + TurboFan



Байткод и новый граф

Обратная связь осталась. Поменялись её хранение и работа компилятора.



Обратная связь переехала в вектор

Раньше

Записи встроены
в машинный код full-codegen

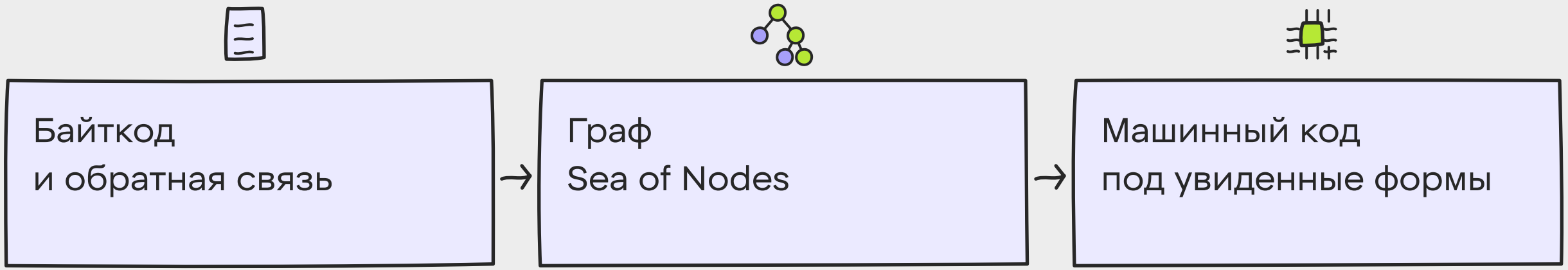
Потом

Отдельный feedback vector.
Место чтения ссылается на слот.



Перенос шёл постепенно: чтения полей — 2015, записи — 2016.

TurboFan строит Sea of Nodes



Зачем граф

У операций есть зависимости.
Можно убирать лишнее
и переставлять допустимое.

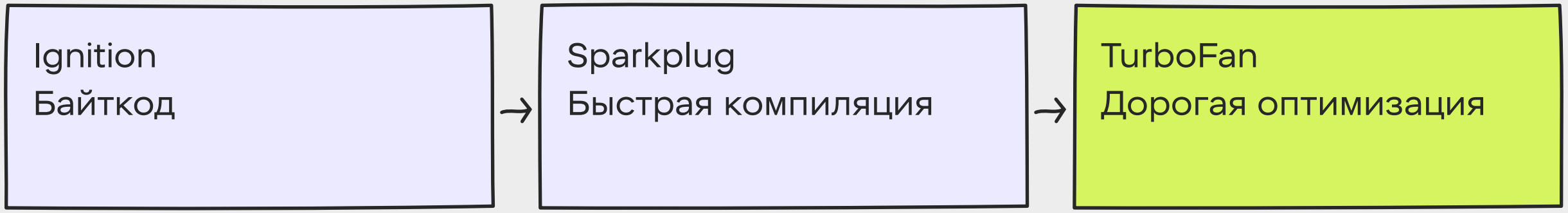
Откуда идея

Cliff Click, 1995; HotSpot.
В V8: 2014 → 2015 → 2017.

НАДЕЮСЬ ВЫ ПОНЯЛИ



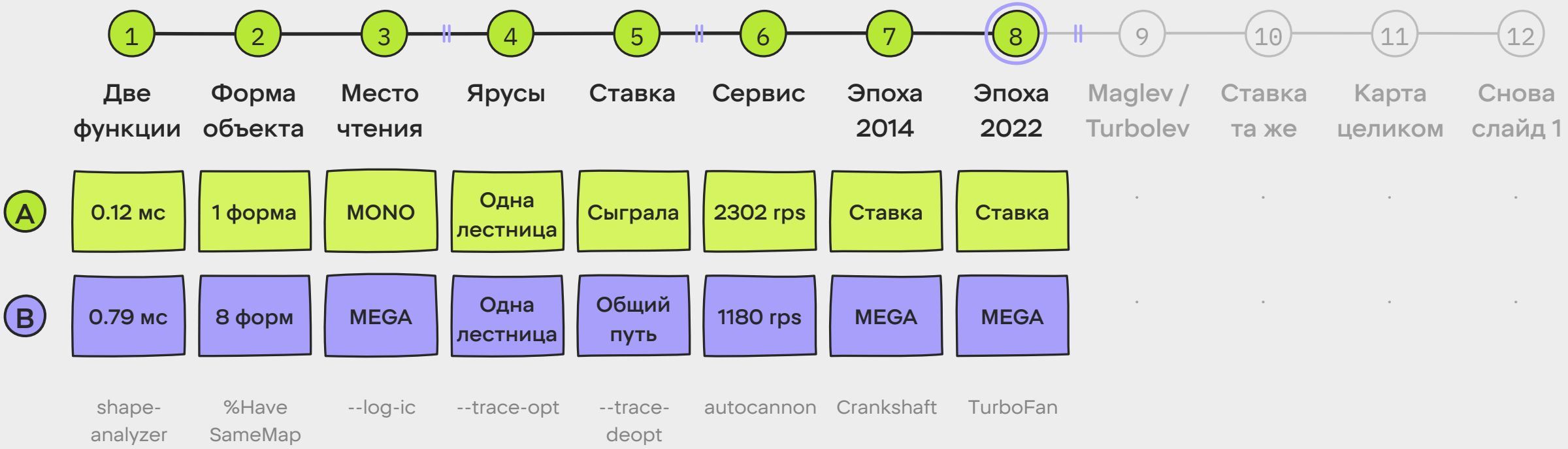
Между Ignition и TurboFan — Sparkplug



Функция уже выполняется как машинный код,
а за самую дорогую оптимизацию ещё не платили.

Где мы · две эпохи позади

Сервис посмотрели. По истории прошли 2014 и 2022.



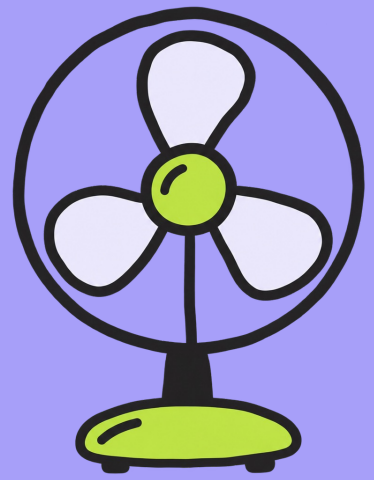
Дальше — Maglev, Turbolev и карта целиком.

|| передышки после 3, 5 и 8

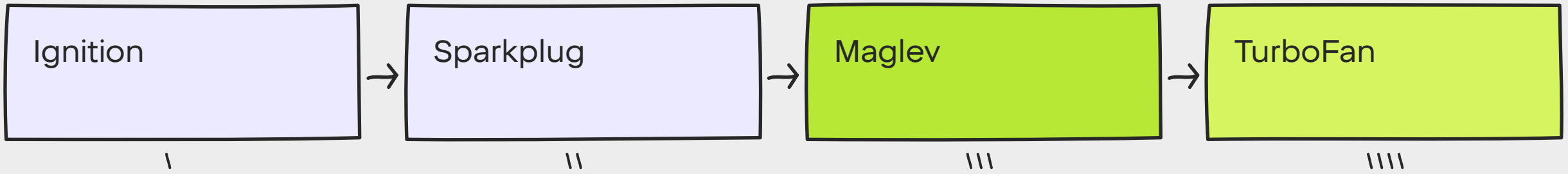

БЕРЕШ
И делаеш

Maglev и Turbolev

Между быстрой компиляцией и дорогой оптимизацией осталось место.



Maglev: оптимизированный код раньше



Свой граф

Использует ту же
обратную связь и формы

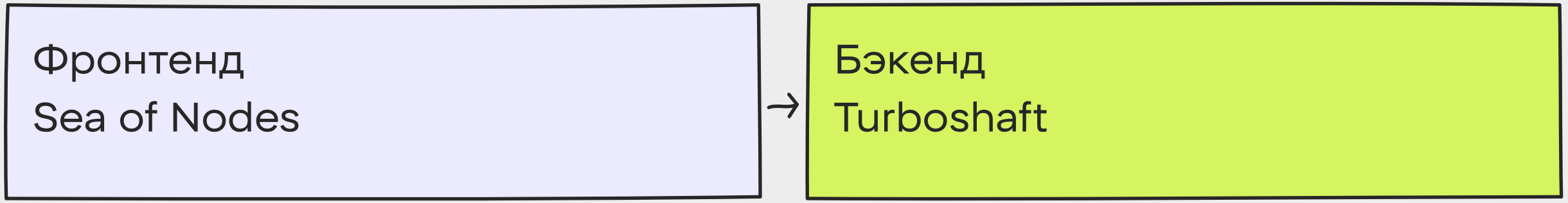
Дешевле компиляция

В нашем замере —
примерно к 1000-му вызову

Версия имеет значение

На нашем Node 24 включён.
В Node 22 с 22.9 — выключен.

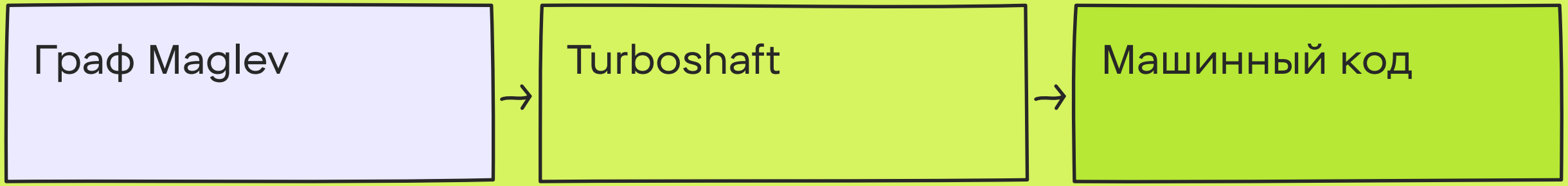
В TurboFan поменяли заднюю часть



~×2

Речь об ускорении компиляции.
Не о двукратном ускорении любого JavaScript.

Turbolev: Maglev + Turboshaft

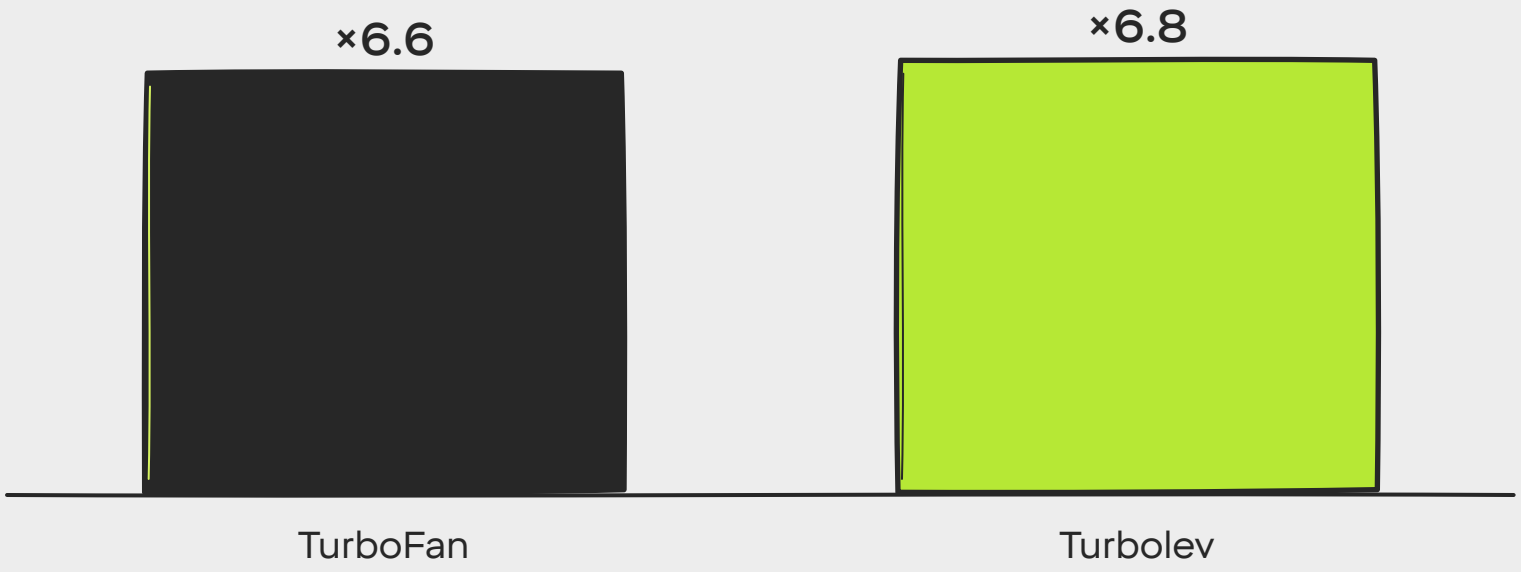


Замена верхнего яруса, не пятая ступень

`--turbolev`

Флаг в нашем стенде. Дата перехода по умолчанию не заявлена.

Включили --turbolev. Что с числами?



Без флага
0.12 / 0.79 мс
×6.6

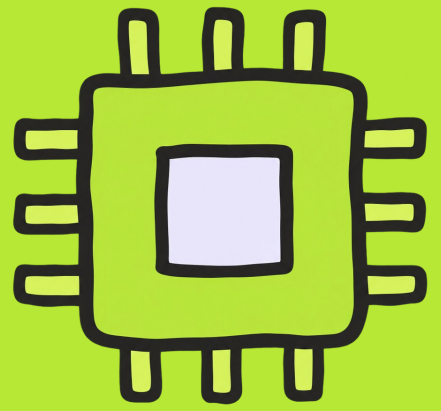
С флагом
0.14 / 0.95 мс
×6.8

```
Один прогон: A 0.13 мс · B 0.88 мс · correct true
```

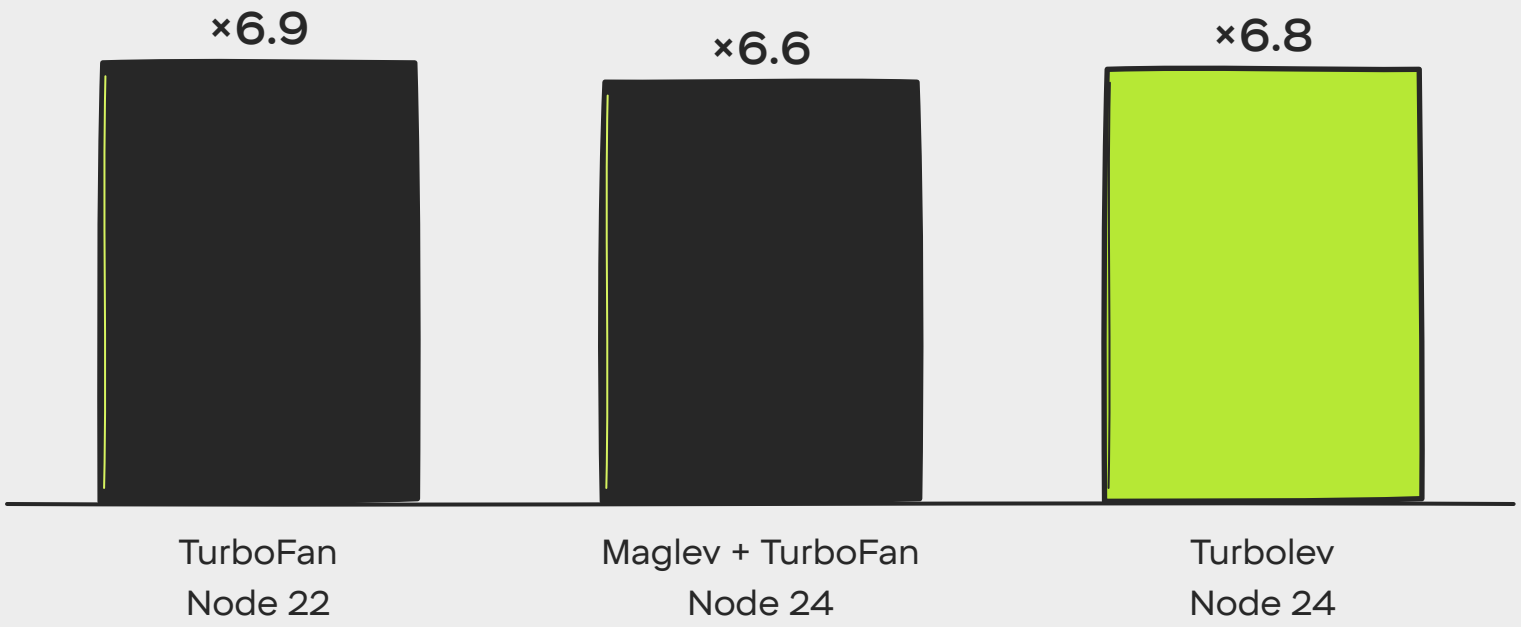
Медианы пяти прогонов. Оба варианта чуть медленнее; разница между A и B сохранилась.

Для нашего примера

Названия компиляторов поменялись. А что поменялось в замере?



Три конфигурации — близкий результат

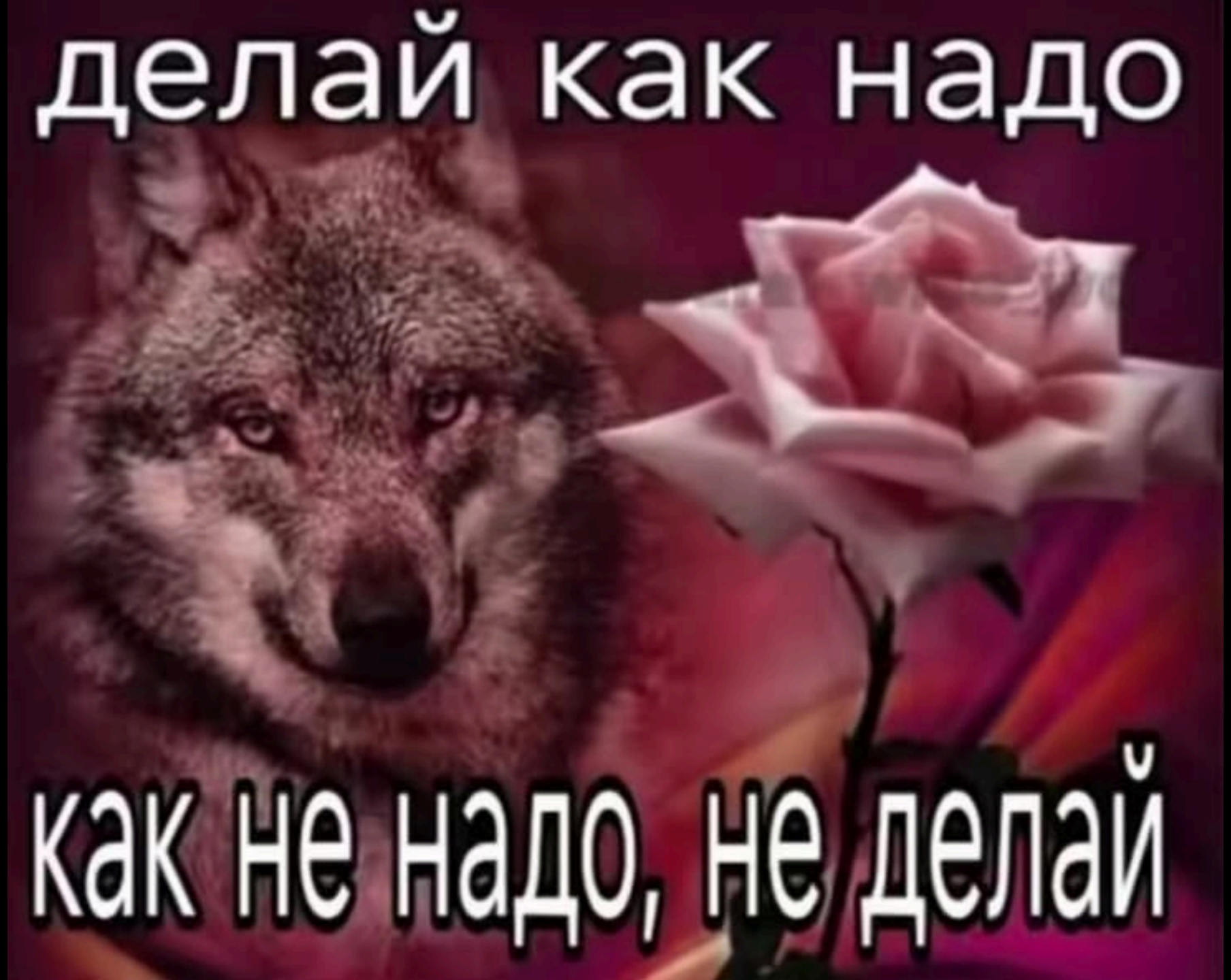


Node 22 ×6.9

Node 24 ×6.6

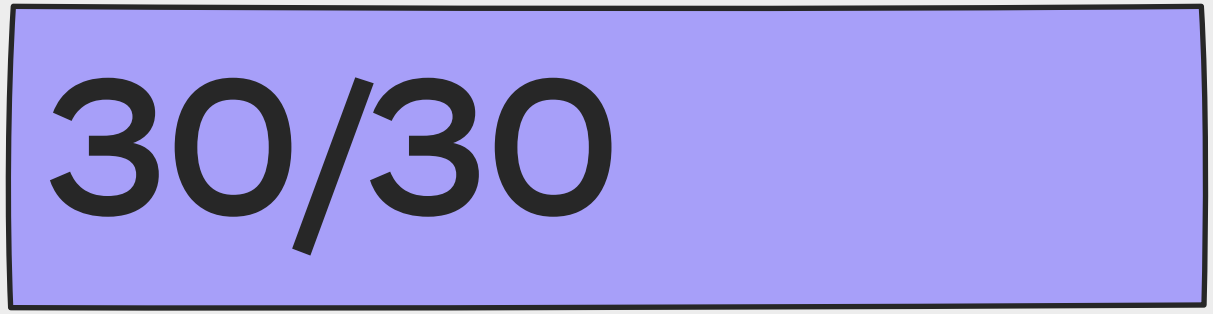
+ turbolev ×6.8

Docker arm64. Crankshaft 2014 не измерялся: историческая часть объясняет механизм.


делай как надо
как не надо, не делай

Такое требование можно объяснить агенту

БЕЗ ПРАВИЛ



8 форм · MEGA

С ЧЕТЫРЬМЯ ПРАВИЛАМИ



1 форма · MONO

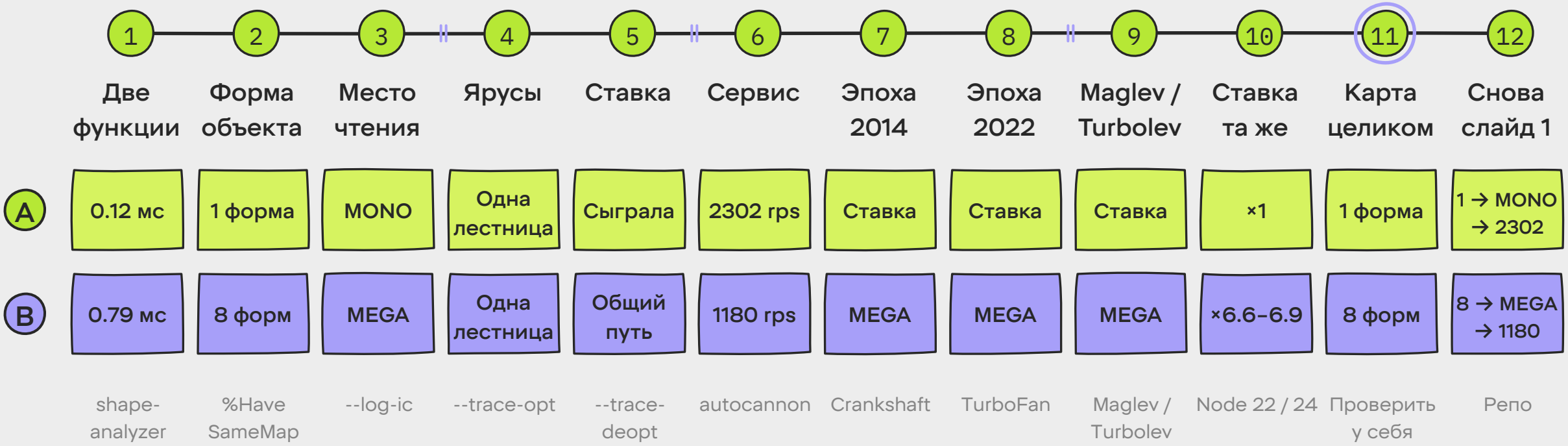
Все 60 прошли проверки. Прямая инструкция с примером; измеряем полученный JS, не скорость агента.

Проверяем у себя

Соберём весь путь и команды, которыми его можно посмотреть.

Карта целиком

Те же A и B — от трёх строк кода до сервиса и истории V8.



Форма задаётся при сборке. Ярусы меняются; различие A/B остаётся.

2014 — механизм, не замер. tsc / esbuild отдельно не измерялись.

Четыре вопроса — четыре ответа

1 **Где расходятся две функции**

В объектах: у A одна форма, у B восемь. Потом эти поля по-разному читаются.

2 **Что может компилятор — и что остаётся в сервисе**

Он опирается на увиденные формы. В нашем примере: ×6.6 на чтении, ×1.95 в HTTP.

3 **Что менялось в V8**

Ярусы и графы менялись. Формы и обратная связь всё ещё нужны оптимизатору.

4 **Как проверить у себя**

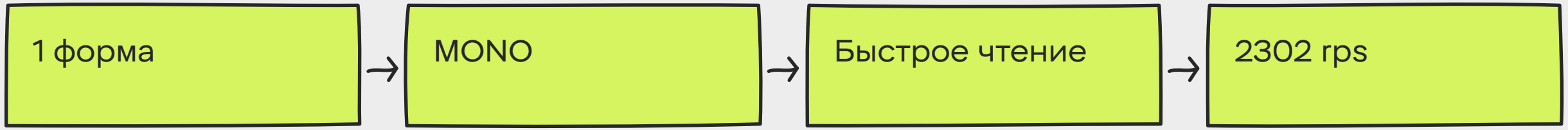
%HaveSameMap, --log-ic и замер своего приложения. Команды — на следующем слайде.

Вернёмся к началу

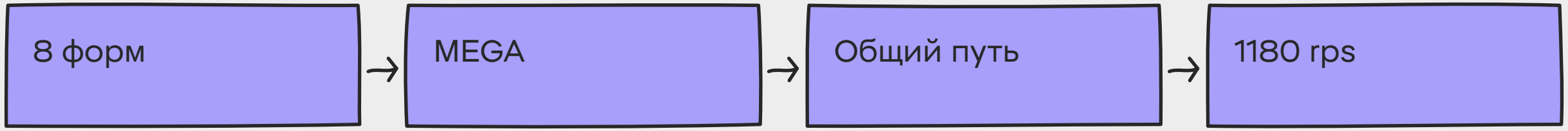
Теперь эти две функции можно прочитать по-другому.

Те же три строчки. Теперь понятно почему

```
A  referrer: raw.referrer ?? null
```




```
B  ...(raw.referrer && { referrer: raw.referrer })
```





Спасибо!

Александр Зайцев

Примеры и замеры

github.com/Yoshimuro/
issues-memory-research



Три команды для проверки

```
node --allow-natives-syntax \
  lab/shape-analyzer.js lab/ref-b.js
```

Формы объектов

```
node --log-ic --no-logfile-per-isolate \
  --logfile=ic.log app.js
```

Журнал мест чтения

```
./docker/run.sh 24 'bash docker/verify.sh my-run \
  && node docker/report.js docker/out/my-run'
```

Все замеры доклада