JIT, V8 и ещё чуть-чуть

Как работает JS

- Код загружается
- С ним что-то делается
- С ним что-то делается (2)
- Вжух, всё выполнилось
- PROFIT!

Что творится на самом деле?

- Javascript что-то выполняет
- Это что-то — интерпретатор
- Интерпретируется обычно на ходу
- Основная разница — как же работает тот или иной интерпретатор.

Что должен уметь интерпретатор?

- Выполнять наш код
- Выполнять наш код корректно
- Выполнять наш код быстро
- Выполнять наш код корректно и быстро :-)

Как будем переводить всё

- Наша цель — ассемблер
- Парсим JS
- Строим AST
- Магия
- Оптимизации
- Ура! Ассемблер :-)

Почему оптимизация кода — сложная задача

- Разные процессоры, разные наборы команд
- Разные ОС с разными особенностями
- Проведение разного рода оптимизаций требует разного рода затрат компухтера
- А как вообще оптимизировать? Под скорость? Под размер бинаря? А что насчёт кеша, аллокаций памяти, выравнивания памяти, и т.д.

Сложно, ***, слишкам сложна

- Предугадывание ветвлений
- Оптимизация по стоимости операций
- Девиртуализация
- Инлайнинг
- Data locality optimization
- Выравнивание
- И т.д.

КДПВ — набор регистров

Интерпретатор vs компилятор

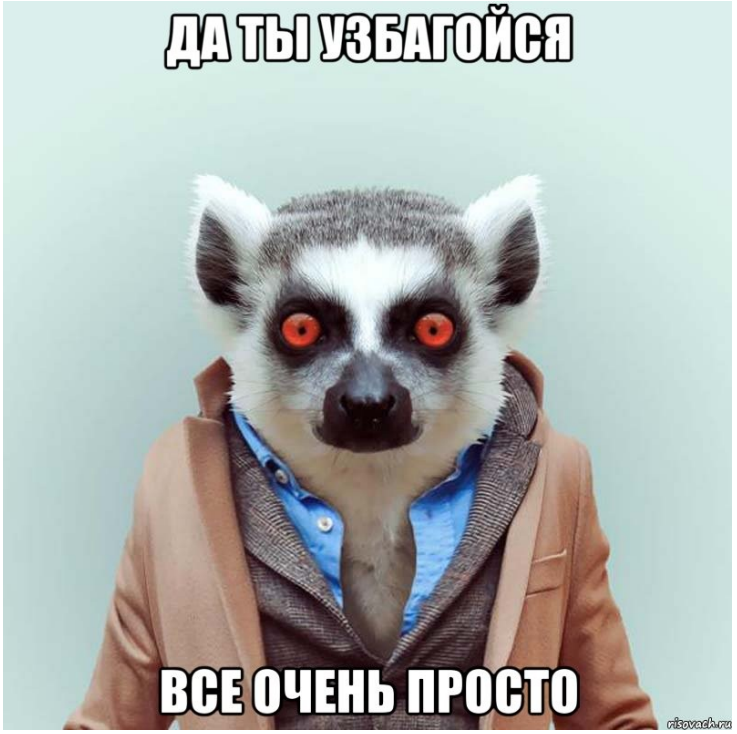
Интерпретатор	Компилятор
Преобразует код на ходу	Преобразует код до его выполнения
Ограничен по времени	Время по большему счёту не волнует
Больше информации о системе, где работает*	Обладает ограниченным набором информации о среде*
Результат работы — выполняющаяся программа	Результат работы — бинарный файл для выполнения

Что выберем?

- Компилятор — вместо загрузки JS нам придётся загружать уже готовый бинарь
- Интепретатор — загружаем код, а он уже сам превратиться в выполняющуюся штуку

Движок — интерпретатор на стероидах

- Выполняет наш код
- Старается оптимизировать его как может
- Отвечает за управление памятью



Какие движки есть?

- V8 — на C++ (православно), от Google
- Rhino — Java, Mozilla
- SpiderMonkey — Mozilla
- JavaScriptCore — Safari
- KJS — поделие от KDE
- Chakra(JScript9) — IE :-)
- Chakra(JS) — Edge
- Nashorn — Oracle
- JerryScript — что-то для IoT

V8 — что за зверь?

- Цель — ускорить выполнение JS
- Chromium-based браузеры
- День рождения: 3 июля 2008
- Фичи: высокая скорость работы и интересные оптимизации
- Мажорное изменение: версия 5.9

V8: до версии 5.9

- Два интерпретатора: full-codegen и Crankshaft
- Full-codegen: шустрый, но в сильные оптимизации не умеет
- Crankshaft: более медленный, умеющий в более крутые оптимизации с использованием статистики от full-codegen

Немного про потоки

- Главный поток — собственно сам интерпретатор
- Оптимизирующий поток
- Профилирующий поток
- Пул потоков на сборку мусора

Немного про оптимизации

- Inlining
- Машинные оптимизации
- Оптимизации на уровне графа выполнения
- И много чего ещё, что я не знаю

V8: 5.9 и выше

- Интерпретатор Ignition + компилятор TurboFan
- Идеи LLVM дошли и до JS (похвально)

Идея

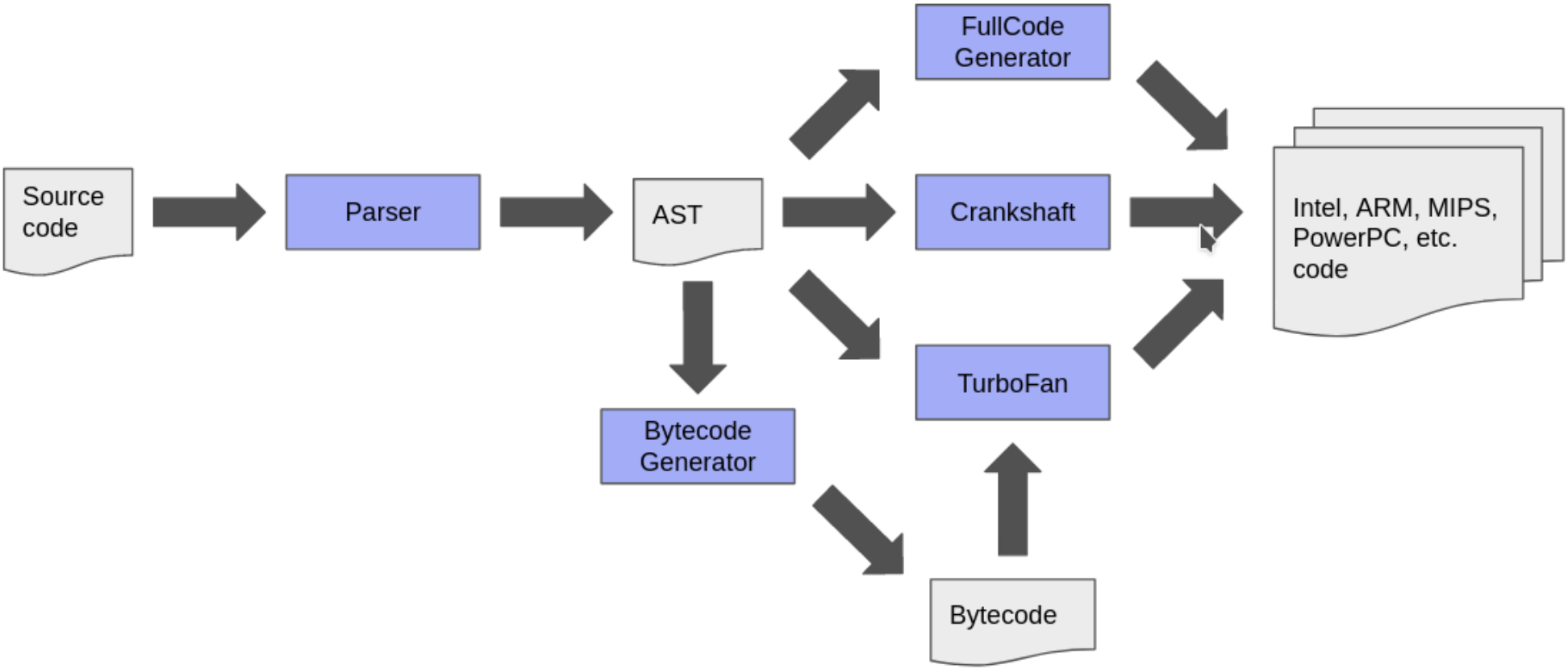
- В V8 старом было очень много hand-written specific кода под каждую платформу. Это плохо
- В новой архитектуре хотим под каждую платформу что-то отдельное
- А переводить будем в какой-нибудь IL
- А потом IL → assembler

Пару слов за Ignition

- На замену full-codegen
- Потребляет меньше RAM
- Его выхлоп отлично используется как entry point для TurboFan компилятора

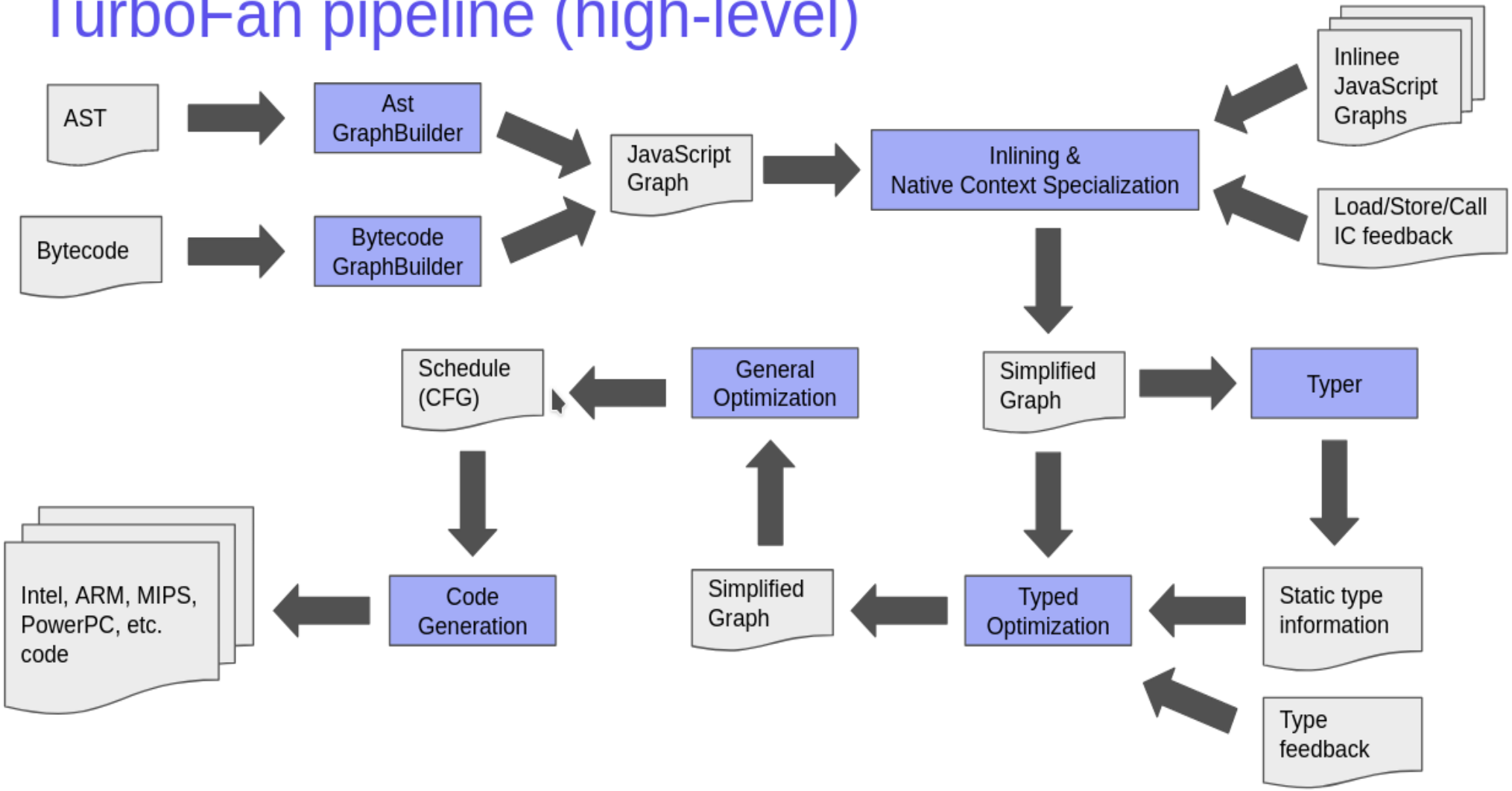
Как-то так :-)

V8 compilation overview



Как-то так :-) (2)

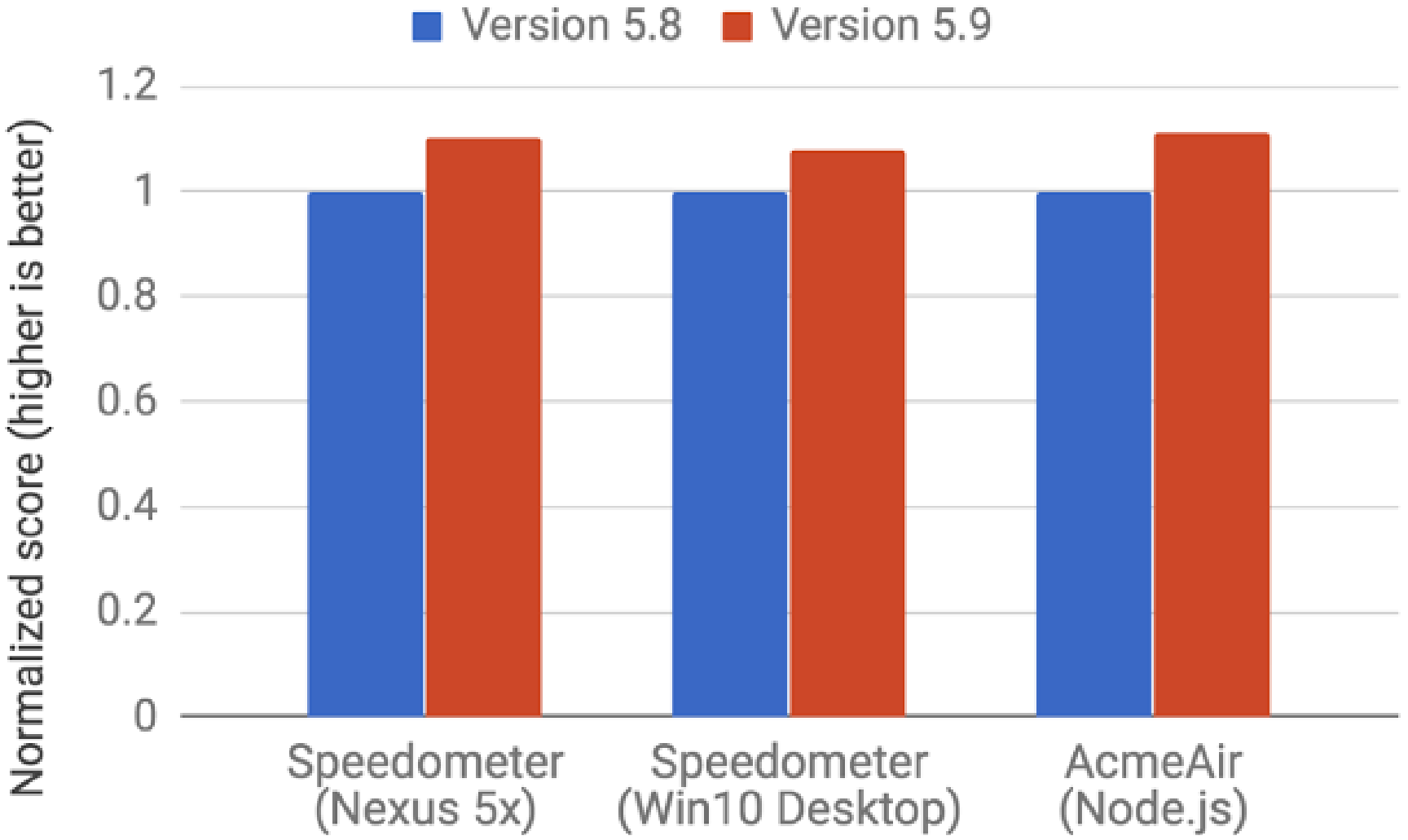
TurboFan pipeline (high-level)



Оптимизации

- Aggresive inlining
- Удаление мёртвых нод на ходу
- Предсказание типов
- Inline allocation
- Control flow-optimization
- Improved register allocation
- More and more optimizations

Сравнение подходов



Внимание!

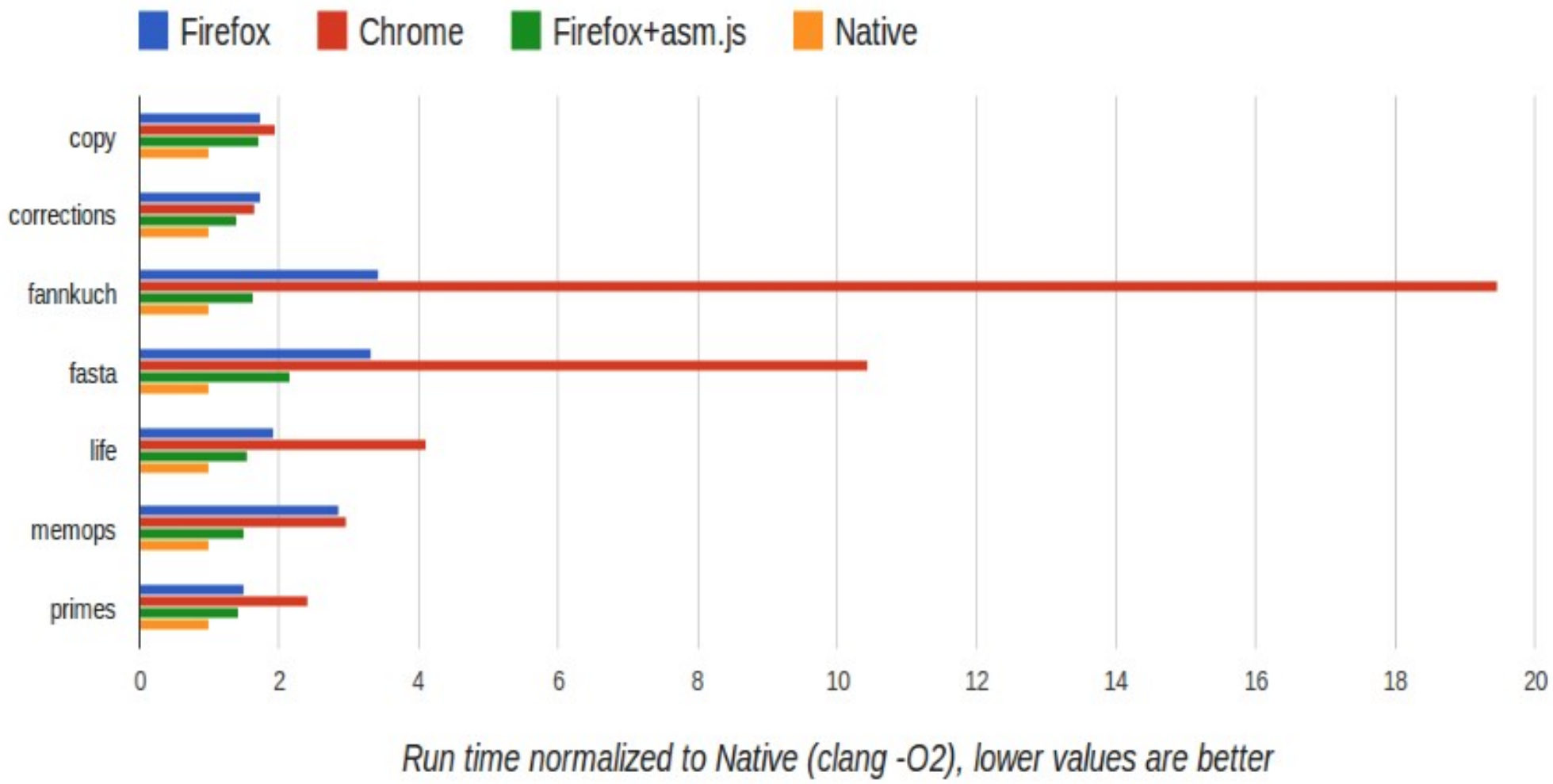
Спасибо за внимание!

(как на лекции БЖЧ)

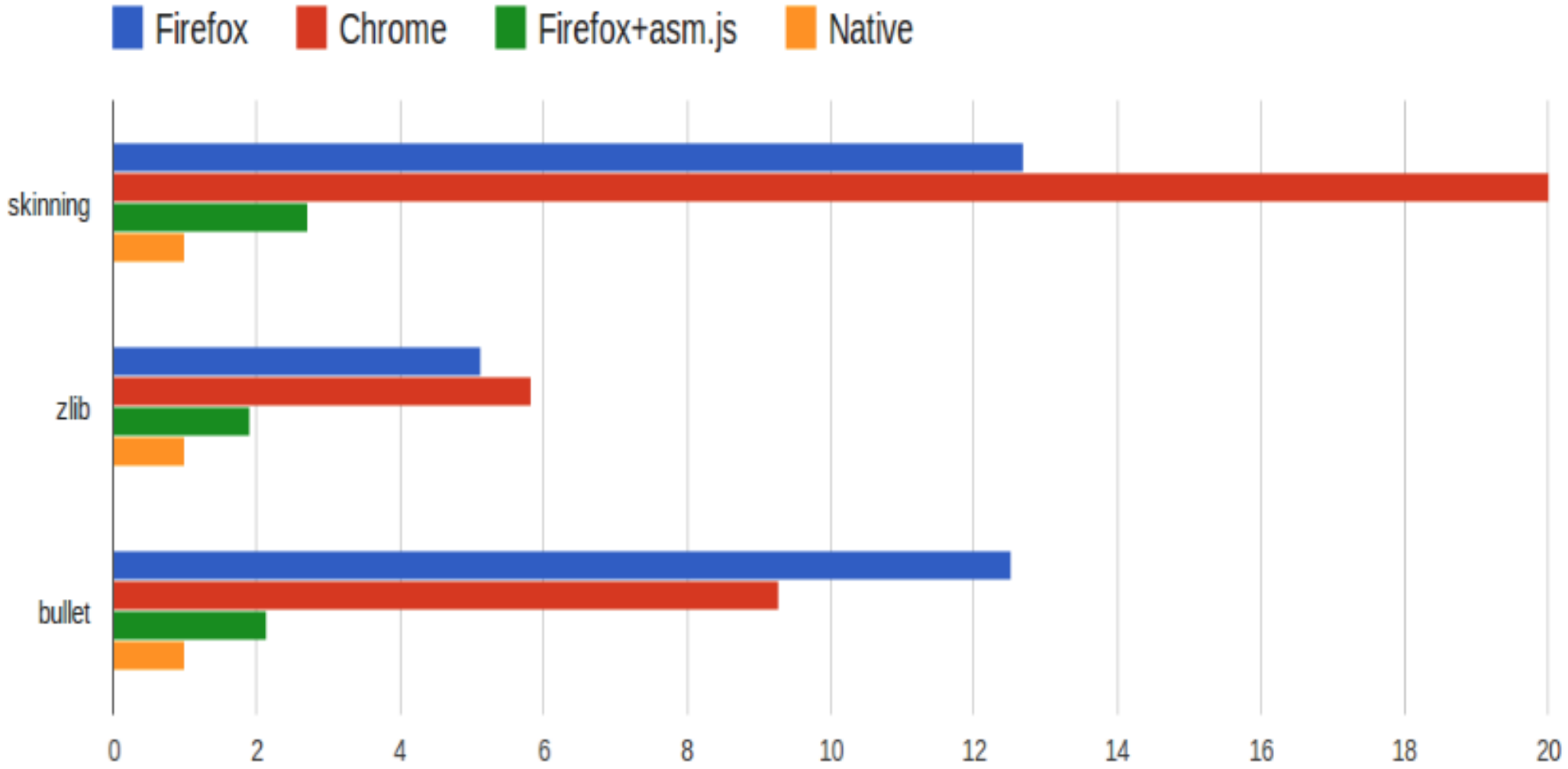
ASM.js

- Создатель: Mozilla
- Подмножество JS
- Есть информация о типах => больше оптимизаций
- Ничего нового нам не надо — это просто подмножество

Микробенч


Firefox
Chrome
Firefox+asm.js
Native
copy
corrections
fannkuch
fasta
life
memops
primes
0
2
4
6
8
10
12
14
16
18
20
Run time normalized to Native (clang -O2), lower values are better

Real-life bench



Native client

- Создатель: Google
- Песочница для выполнения нативного кода
- Ограничения: кол-во платформ (x86, x64, ARM)

Emscripten/Duetto/etc.

- Трансляторы кода из чего-то быстрого в JS
- На выходе — не читаемая человеком каша (но нам это и не надо)
- Работает довольно шустро
- Проблема: не так просто взять и мапить C++ на JS, ОЧЕНЬ много ограничений

WebAssembly

- Свежий проект
- Цель: создать переносной бинарный формат для компиляции
- Поддерживаемые языки: C, C++, Rust
- Стадия: в разработке
- Может ли в нативность: да (ограничено)