ISP RAS

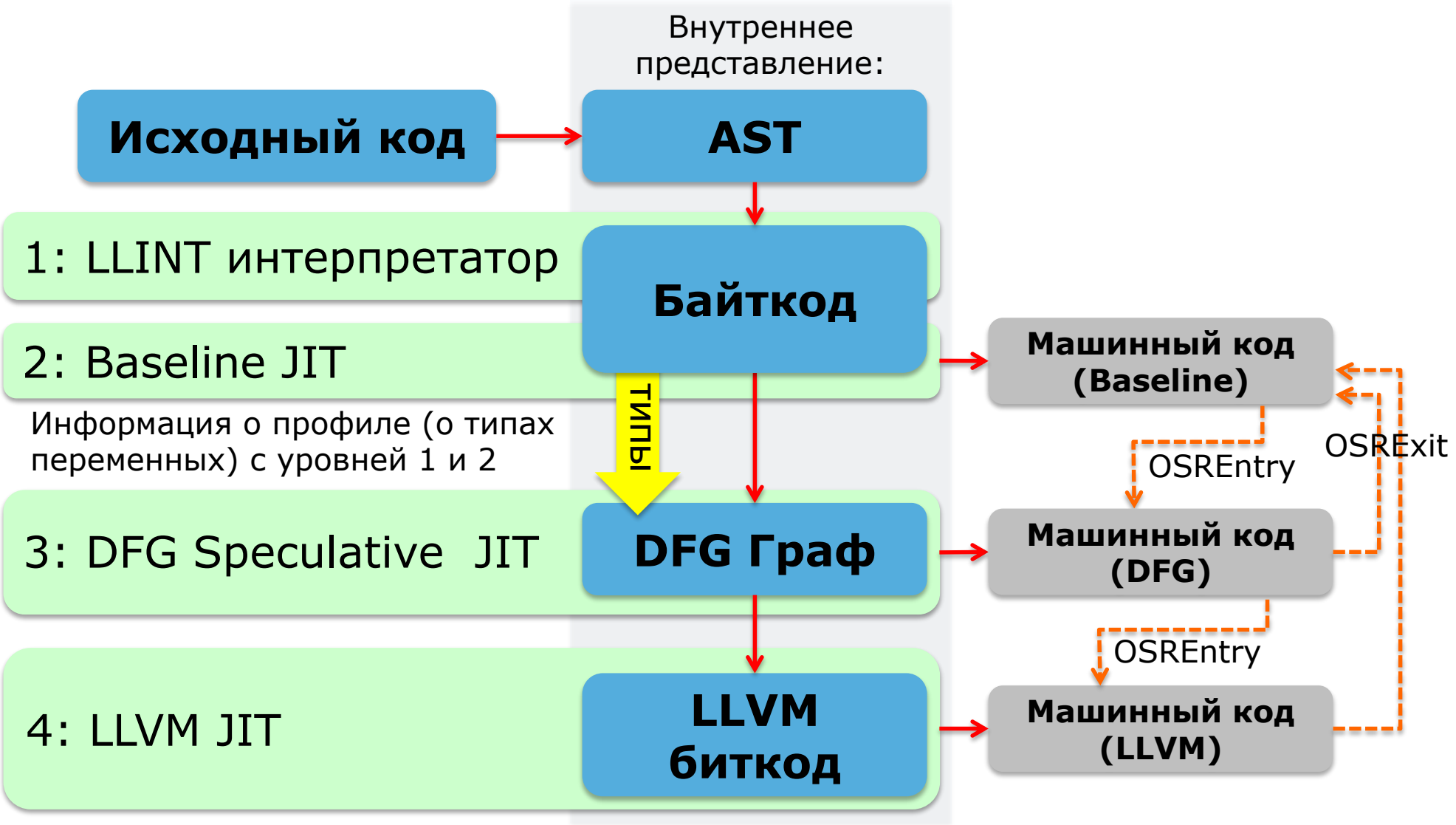
Методы предварительной оптимизации программ на языке JavaScript

Роман Жуйков
zhroma@ispras.ru

Евгений Шарыгин
eugene.sharygin@gmail.com

Открытая конференция по компиляторным технологиям
2 декабря 2015 г.

Устройство WebKit JavaScriptCore

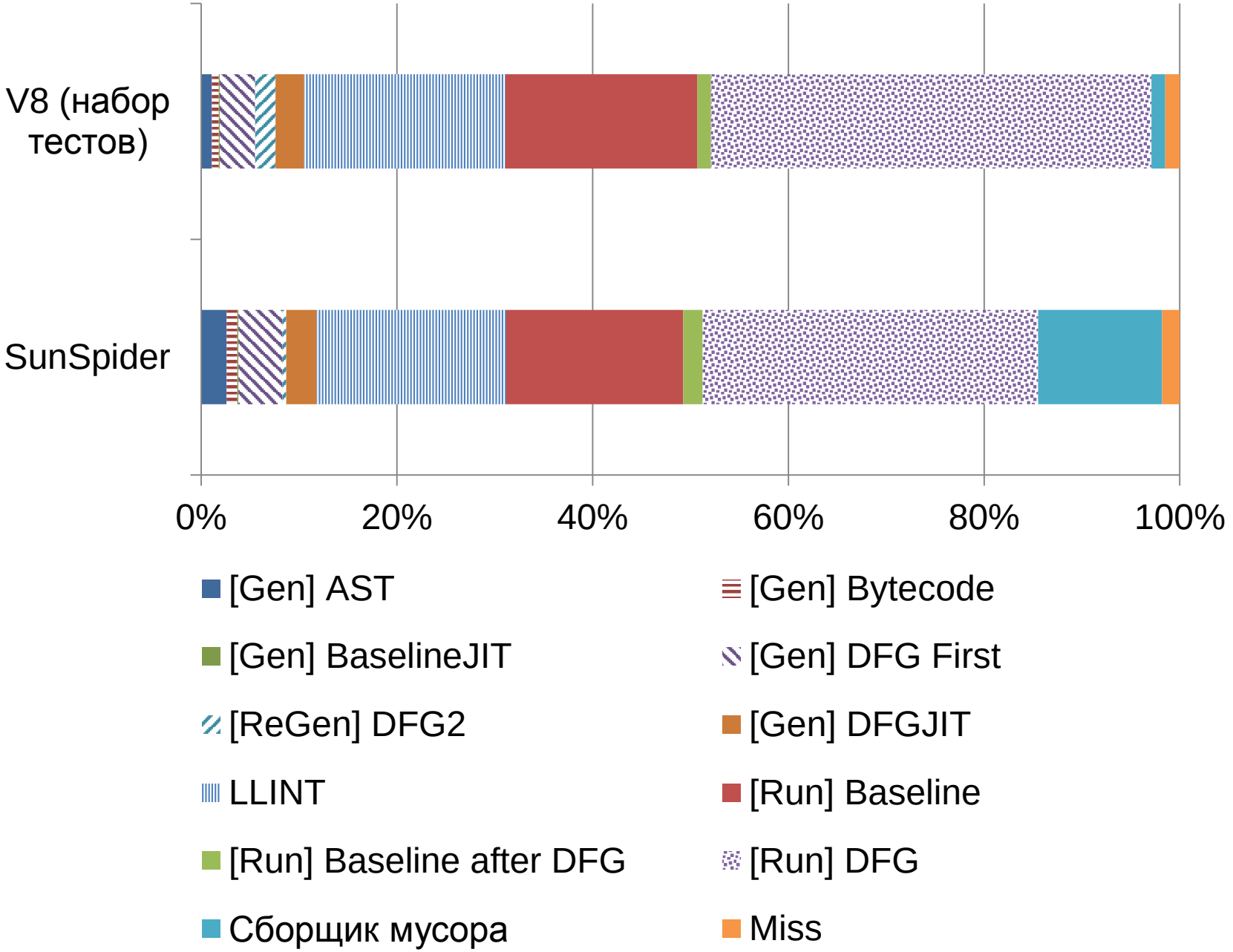


Когда функция становится «горячей», происходит переход **Baseline JIT** → **DFG** → **LLVM** с помощью механизма замены на стеке (OSR)

Сравнение уровней JIT

Тест	Ускорение V8-richards, раз		Ускорение набора Browsermark, раз	
	Относительно интерпрета-тора	Относительно предыдущего уровня	Относительно LLINT	Относительно предыдущего уровня
JSC интерпретатор	1.00	-	н/д	-
LLINT	2.22	2.22	1.00	-
Baseline JIT	15.36	6.90	2.50	2.5
DFG JIT	61.43	4.00	4.25	1.7
Тот же код на C	107.50	1.75	н/д	-

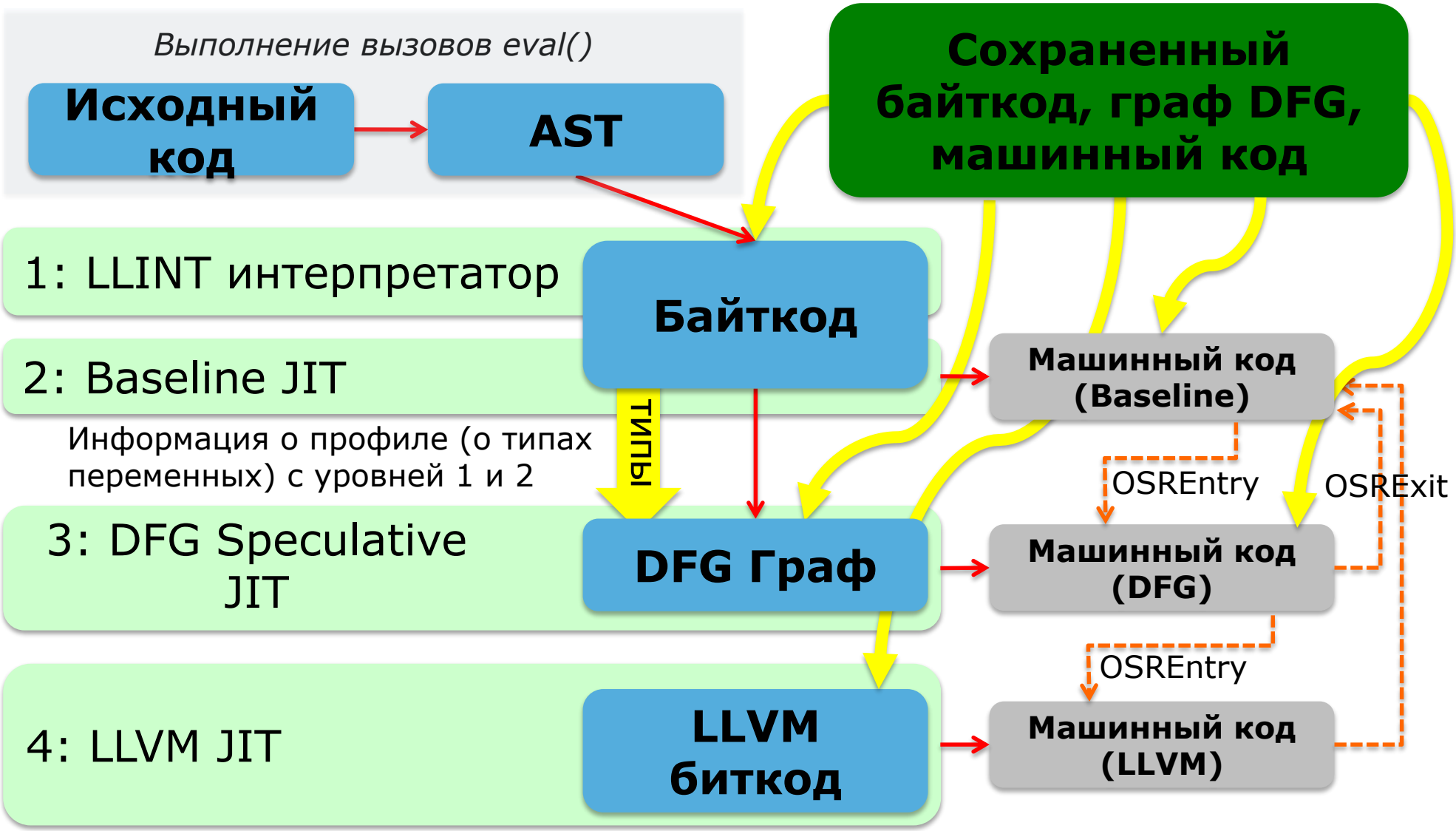
Анализ работы JSC



Возможности AOTC

- Сокращение времени, затрачиваемого на подготовительные операции, такие как работа парсера исходного кода, построение AST, генерация байткода и машинного кода
- Обеспечение раннего перехода выполнения на более оптимизированный уровень JIT-компиляции
- Базовая защита исходного кода за счет распространения только бинарного файла с сохраненной информацией

Архитектура AOTC



Сохранение байткода JSC

Два этапа работы системы:

- Оффлайн фаза, сохранение:
  - парсинг исходного кода
  - генерация байткода (с выполнением или без выполнения кода)
  - сохранение байткода в файл вместе с необходимой информацией
  - добавление таблицы смещений функций в начало файла
- Онлайн фаза, загрузка:
  - чтение файла с байткодом
  - при вызове сохраненной функции ее байткод загружается в готовом виде

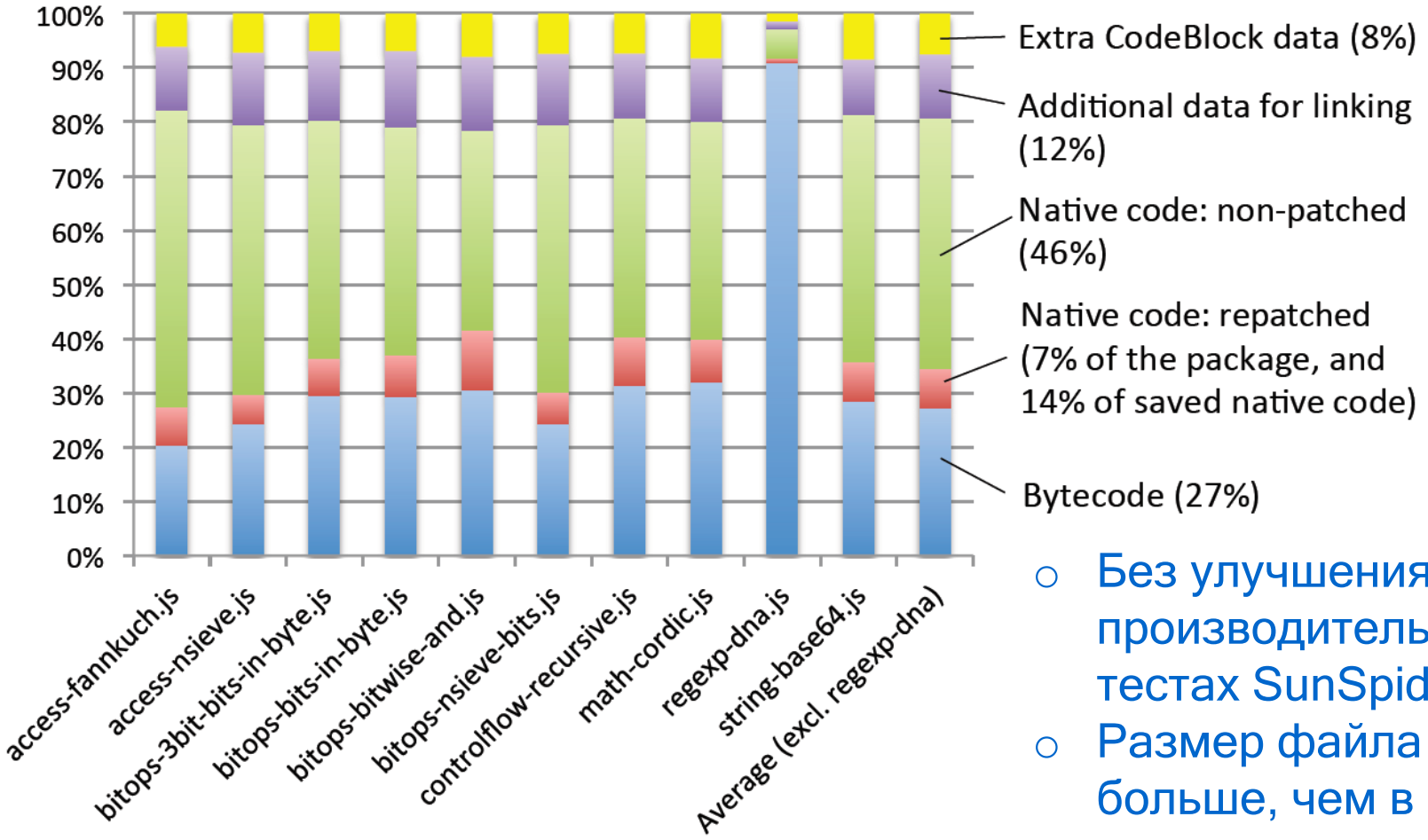
Результаты AOTB

- Полная поддержка стандарта ECMA-262, за исключением обработки операций, явным образом требующих наличия исходного кода
- Вызовы eval() так же поддерживаются, исходный код обрабатывается обычным образом с построением AST и генерацией байткода

Тестовый набор	Полученное ускорение, %	Рост размера файла, раз	
		Оригинальный JavaScript	Сжатые версии файлов
SunSpider	3.3	1.2	1.3
V8-V6	1.0	2.3	4.4
Kraken	16.0	2.0	1.3

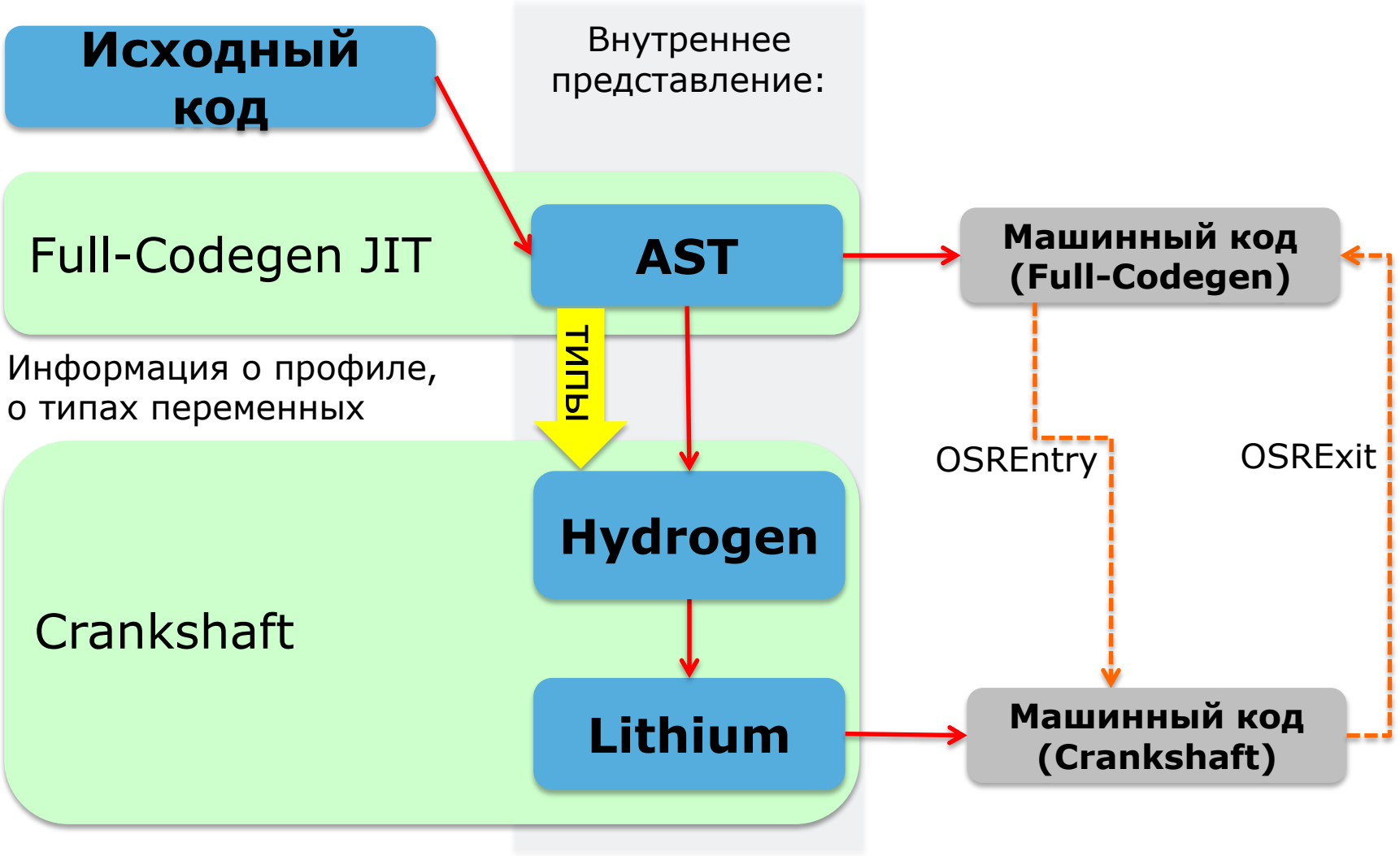
Сохранение BaselineJIT кода

- В AOTB была добавлена инфраструктура для сохранения и загрузки неоптимизированного машинного кода
- Множество адресов требует линковки



- Без улучшения производительности на тестах SunSpider и v8-v6
- Размер файла в 2.5 – 5 раз больше, чем в случае AOTB

Архитектура V8



Сохранение Lithium

- Аналогично байткоду JavaScriptCore, для V8 Lithium реализовано сохранение в файл и загрузка из файла, однако полученный файл лишь дополняет исходный JavaScript
- Сохранение необходимой информации с уровня Hydrogen путем создания дополнительного уровня между Hydrogen и Lithium
- Встраивание сохранения и загрузки в процессы виртуальной машины, в том числе корректность работы при деоптимизации (OSR Exit)

Результаты сохранения Lithium

- Возможно достижение 20-кратного ускорения на искусственных тестах
- До 30% ускорения для asm.js версии теста Tramp3d.cpp
- До 15% (0.1 – 0.2 сек) ускорения на тестовых приложениях Box2d, Bullet и ZLib с сайта arewefastyet.com
- Не получено ускорения на наборах SunSpider, Octane, Kraken

Заключение

- Предложен общий метод применения предварительной компиляции в виртуальных машинах с использованием многоуровневой JIT-компиляции
- Выполнена реализация в двух виртуальных машинах JavaScript
  - **В JavaScriptCore (WebKit)**
    - Сохранение и загрузка байткода и неоптимизированного машинного кода, с полной интеграцией в WebKit и поддержкой стандарта ECMA-262
    - 3% ускорения на тестах SunSpider, 16% на тестах Kraken
  - **В движке V8 (Chrome)**
    - Сохранение и загрузка оптимизированного машинно-зависимого внутреннего представления Lithium
    - Улучшение производительности asm.js тестов
    - Проект будет опубликован на Github до конца 2015 года