Портирование JS на Эльбрус

Unipro JS team

Про что доклад?

- Про Эльбрус

Про что доклад?

- Про Эльбрус
- Про JavaScript
- Про V8
- Про SpiderMonkey

Про что доклад?

- Про Эльбрус
- Про JavaScript
- Про V8
- Про SpiderMonkey
- Про внутреннее устройство движков JS
- Про портирование

Кто мы такие?

- Unipro - 25 лет
- Участвовали в разработке платформы Java с 1996
- Разработчик Java Compatibility Kit
- “Реинкорнатор” Apache Harmony
- Тестируем Dart для Google
- Java под Эльбрус
- JavaScript под Эльбрус 


Эльбрус

Эльбрус

1. VLIW

X86

1. Суперскалярный


<- Ваш код
Intel

<- Ваш код

<- Ваш код

Intel

<- Ваш код

Elbrus


ALU 1
ALU 2
ALU 3
ALU 4

```
function floatMath(a, b) {
    return Math.sqrt(a.x * b.x + a.y * b.y);
}
```

ASM

```
setwd wsz = 0x4
return %ctpr3
```

```
ldd, 0 %dr0, 0x0, %r0
ldd, 2 %dr0, 0x4, %r1
ldd, 3 %dr1, 0x0, %r2
ldd, 5 %dr1, 0x4, %r3
```

```
fmuld, 0 %r0, %r1, %r0
fmuld, 3 %r2, %r3, %r1
```

```
faddd, 0 %r0, %r1, %r0
```

```
fsqrts, 0 %r0, %r0
```

```
ct %ctpr3
```

Эльбрус

1. VLIW
2. Много регистров (192+32+32)

X86

1. Суперскалярный
2. Мало регистров (16 + 16 + 8)

Эльбрус

1. VLIW
2. Много регистров (192+32+32)
3. Явная спекулятивность

X86

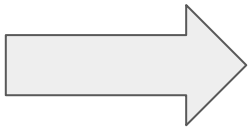
1. Суперскалярный
2. Мало регистров (16 + 16 + 8)
3. Неявная спекулятивность

Спекулятивность

```
function Foo(a) {
  if (a === null) {
    return 0;
  }
  return a.bar;
}
```

Спекулятивность

```
function Foo(a) {
  if (a === null) {
    return 0;
  }
  return a.bar;
}
```



```
function Foo(a) {
  let b = a.bar;
  if (a === null) {
    return 0;
  }
  return b;
}
```

Эльбрус

1. VLIW
2. Много регистров (192+32+32)
3. Явная спекулятивность
4. Условное выполнение ”почти” любых инструкций

X86

1. Суперскалярный
2. Мало регистров (16 + 16 + 8)
3. Неявная спекулятивность
4. CMOV

Условное исполнение

```
function Foo(a) {
  let b = a.bar;
  if (a === null) {
    return 0;
  }
  return b;
}
```

Условное исполнение

```
function Foo(a) {
  let b = a.bar;
  if (a === null) {
    return 0;
  }
  return b;
}
```

```
function Foo(a):
```

```
RET ctpr1
LDD a.bar b
MOV 0 result
CMPEQ a null P1
```

```
MOV b result ? !P1
CT ctpr1
```

Эльбрус

1. VLIW
2. Много регистров (192+32+32)
3. Явная спекулятивность
4. Условное выполнение ”почти” любых инструкций
5. 3 аппаратных стека (2 защищенных)

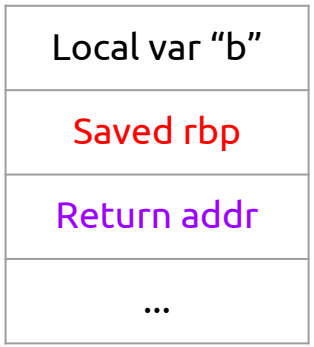
X86

1. Суперскалярный
2. Мало регистров (16 + 16 + 8)
3. Неявная спекулятивность
4. CMOV
5. 1 общедоступный стек

Специфика стеков на Эльбрусе

```
function Foo(a) {
  let b = a.bar;
  ...
  return b;
}
```

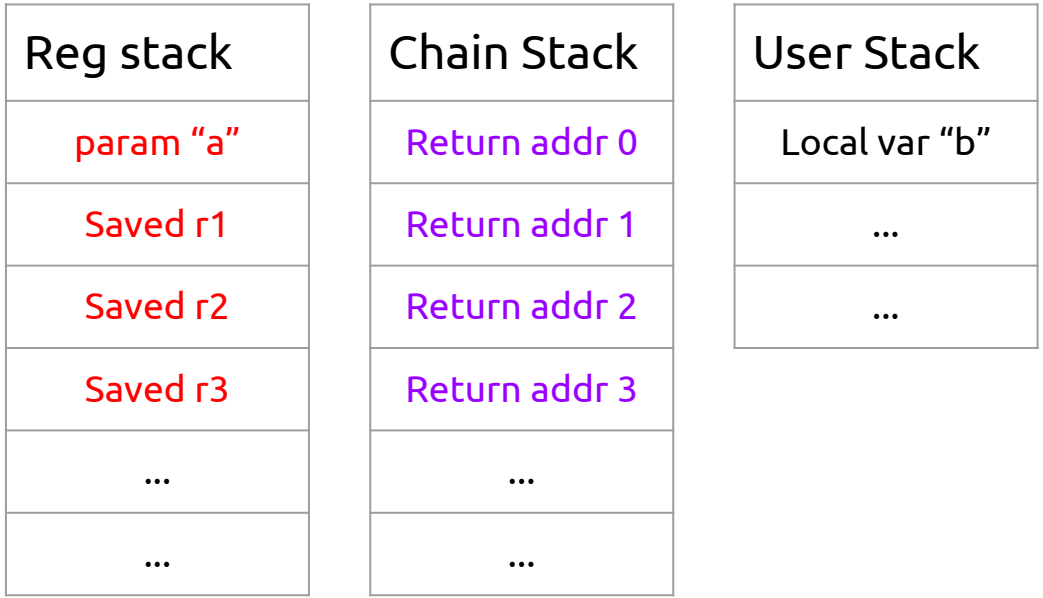
Стек x86-64



Специфика стеков на Эльбрусе

```
function Foo(a) {
  let b = a.bar;
  ...
  return b;
}
```

Стеки Эльбруса

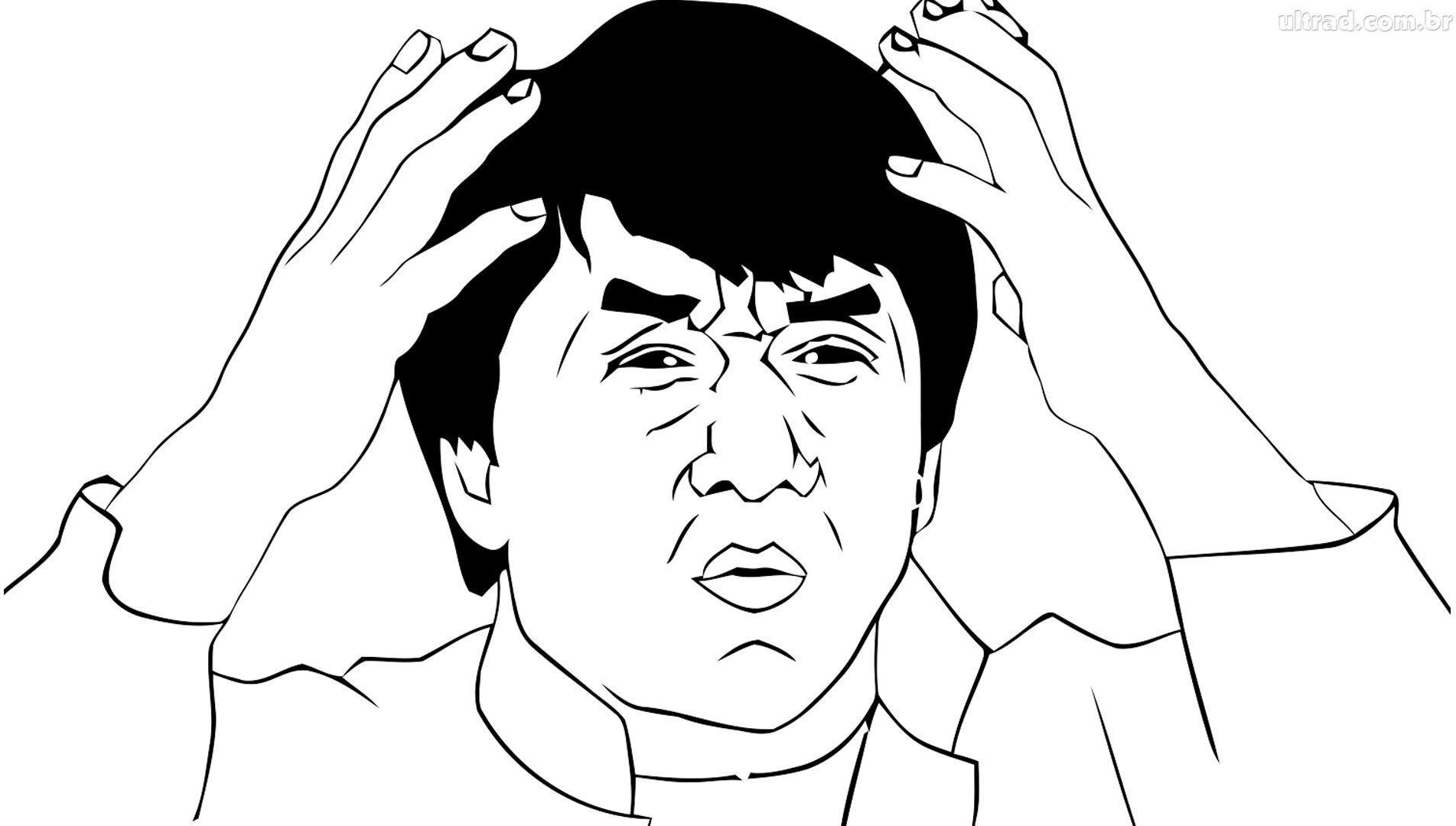


Эльбрус

1. VLIW
2. Много регистров (192+32+32)
3. Явная спекулятивность
4. Условное выполнение ”почти” любых инструкций
5. 3 аппаратных стека (2 защищенных)
6. Нет динамического предсказателя переходов

X86

1. Суперскалярный
2. Мало регистров (16 + 16 + 8)
3. Неявная спекулятивность
4. CMOV
5. 1 общедоступный стек
6. Есть аппаратный предсказатель переходов


ultrad.com.br

Зачем JavaScript на Эльбрусе?

- Импортозамещение

Зачем JavaScript на Эльбрусе?

- Импортозамещение
- Коммерциализация Эльбруса

Зачем JavaScript на Эльбрусе?

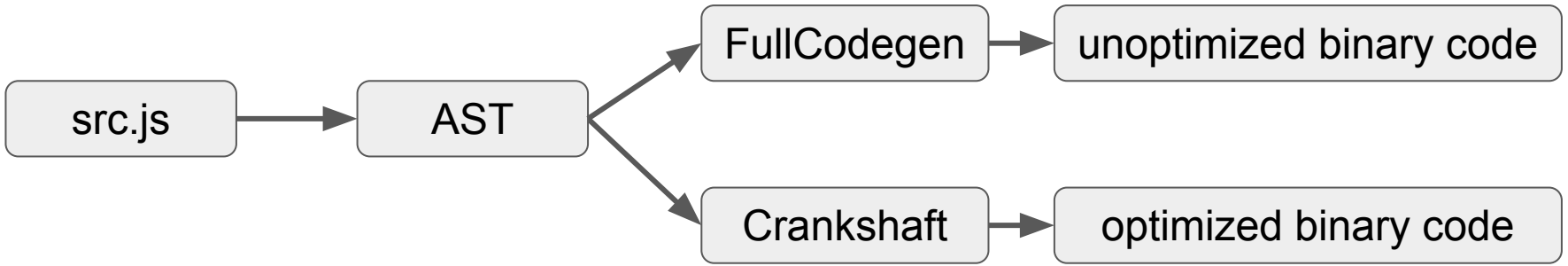
- Импортозамещение
- Коммерциализация Эльбруса
- Node в нефтегазовом секторе [1]

Зачем JavaScript на Эльбрусе?

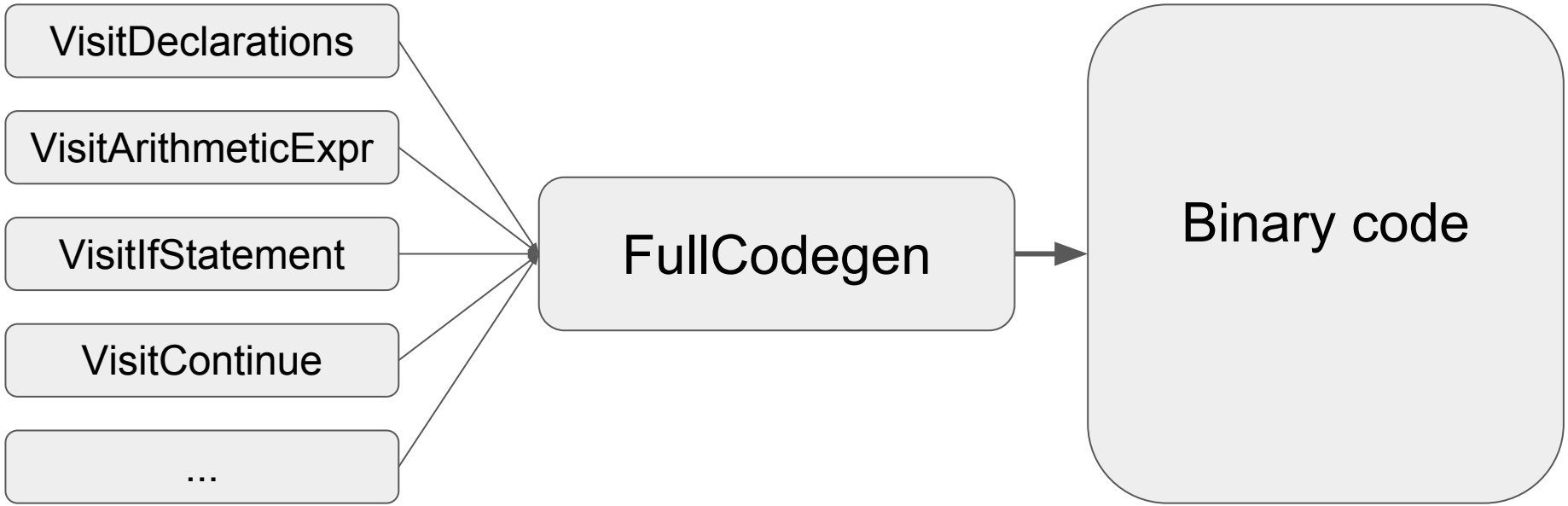
- Импортозамещение
- Коммерциализация Эльбруса
- Node в нефтегазовом секторе [1]
- Развитие архитектуры и умов в России

Реализация

Версия 1.0 RV8



Версия 1.0 RV8



Почему FullCodegen?

Факты:

1. 1 visit* ~**300** строчкам на masm

Почему FullCodegen?

Факты:

1. 1 visit* ~**300** строчкам на masm
2. Не так много переходов как в интерпретаторе

Почему FullCodegen?

Факты:

1. 1 visit* ~**300** строчкам на masm
2. Не так много переходов как в интерпретаторе
3. Очень простой

Почему FullCodegen?

Факты:

1. 1 visit* ~**300** строчкам на masm
2. Не так много переходов как в интерпретаторе
3. Очень простой
4. Но очень много писать...

FullCodegen 1.0

1. Все что можно делается через C++ runtime v8

FullCodegen 1.0

1. Все что можно делается через C++ runtime v8
2. Нет никаких оптимизаций

FullCodegen 1.0

1. Все что можно делается через C++ runtime v8
2. Нет никаких оптимизаций
3. Код просто переписали с интела


LADA 2107
Elbrus inside

FullCodegen 1.1

1. Как можно меньше рантайма

FullCodegen 1.1

1. Как можно меньше рантайма
2. Ручная if-conv

If-conv в дикой природе

```
__ cmpd_e(gr20, gr10, pr6); // pr6 -> NaN -> false
__ cmpd_e(gr18, gr10, pr0); // pr0 -> undefined -> false
__ cmpd_e(gr19, gr10, pr3); // pr3 -> 'null' -> false
// Smis: 0 -> false, all other -> true
__ cmpd_e(gr10, imm_0, pr4); // pr4 -> zero smi
__ CheckSmi(gr10, pr5); // pr5 -> is smi
__ guarantee_new_bundle();

__ ldd(gr20, masm->getFieldOffset(HeapNumber::kValueOffset), gr20);
__ movep(pr0, false, pr0, false, pr2); // undefined || false.val
__ andp(pr4, true, pr5, false, pr5); // non zero smi
__ movep(pr3, false, pr3, false, pr4); // null || zero smi
```

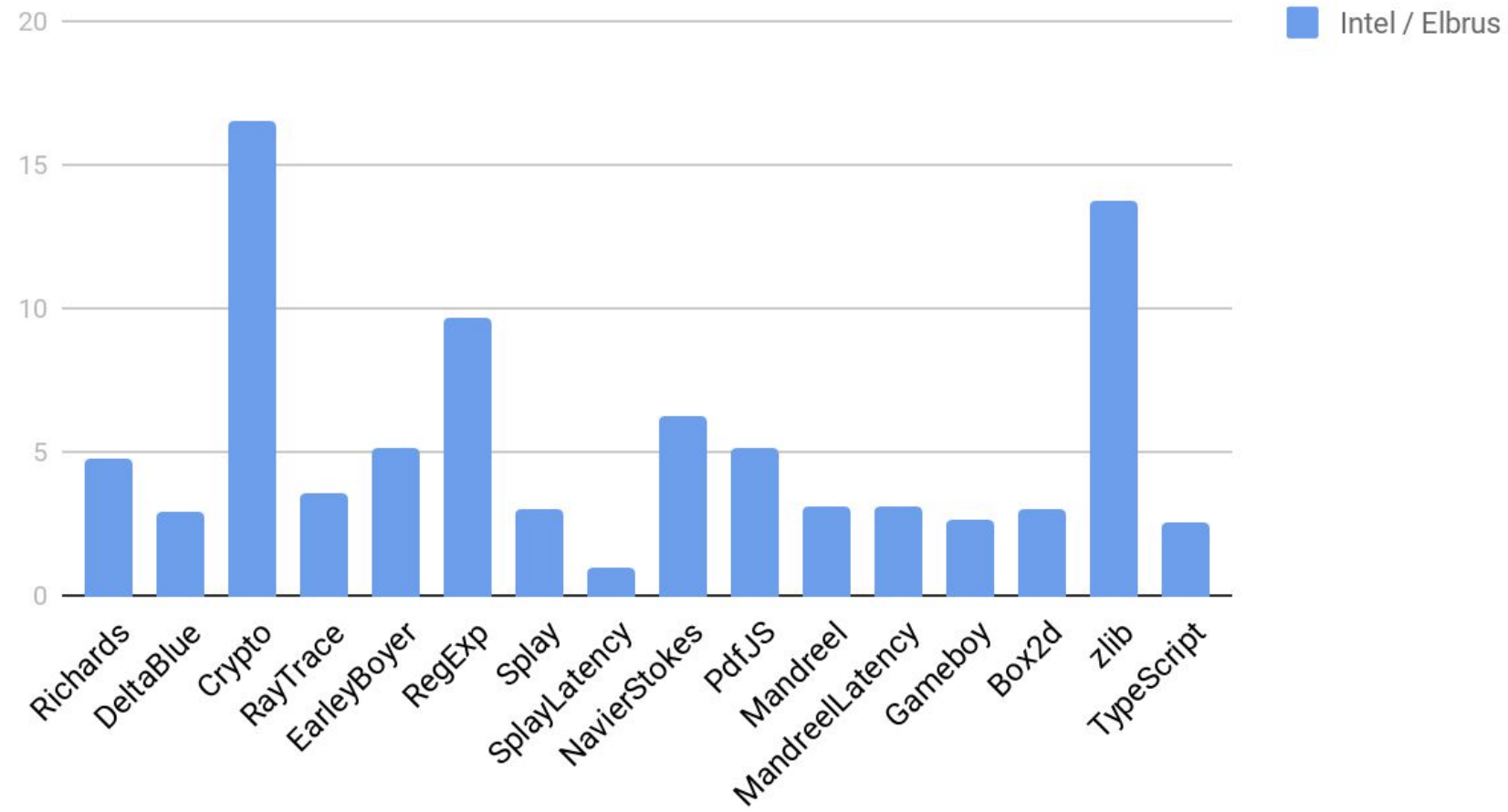
FullCodegen 1.1

1. Как можно меньше рантайма
   a. Включили поддержку ICs
2. Ручная if-conv
3. Код **не просто** переписали с интела
   a. Использовали спекулятивность в стабах
   b. Реализовали fast-path тоже через masm

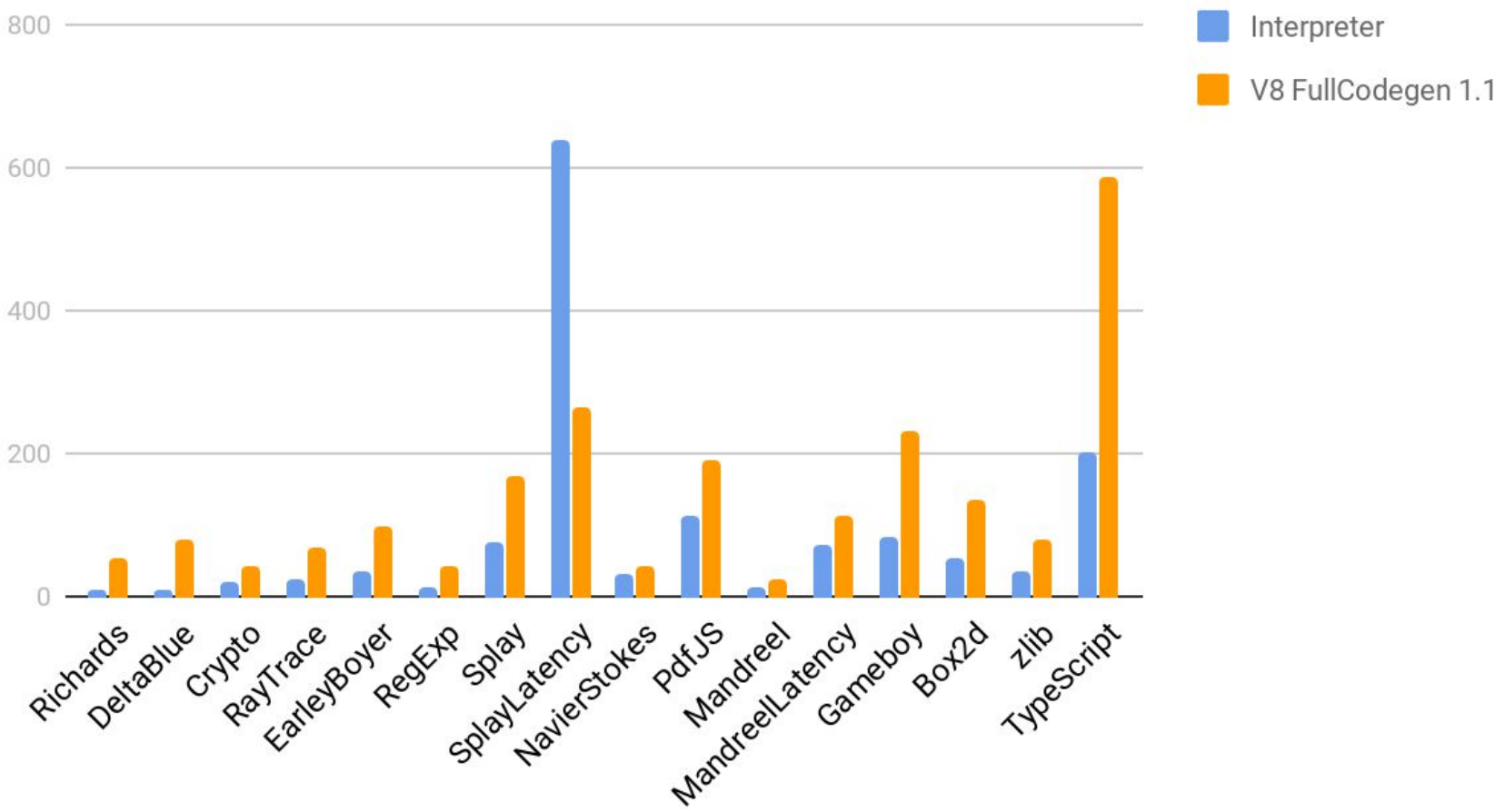
Версия 1.1 RV8 результаты

- Тесты: Google octane
- Эльбрус: E2S 750Mhz, 24Gb.
- Intel: core i7 3.4Ghz, 16Gb.

Octane


Intel / Elbrus
20
15
10
5
0
Richards
DeltaBlue
Crypto
RayTrace
EarleyBoyer
RegExp
Splay
SplayLatency
NavierStokes
PdfJS
Mandreel
MandreelLatency
Gameboy
Box2d
zlib
TypeScript

Octane


800
600
400
200
0
Interpreter
V8 FullCodegen 1.1
Richards
DeltaBlue
Crypto
RayTrace
EarleyBoyer
RegExp
Splay
SplayLatency
NavierStokes
PdfJS
Mandreel
MandreelLatency
Gameboy
Box2d
zlib
TypeScript

Версия 1.1 RV8 результаты

- Проходим Official ECMAScript Conformance Test Suite (test262).
- Производительность в ~**5.2** раза лучше интерпретатора на OS эльбрус на бенчмарке Octane.
- Портировали NodeJS 6.10 как пример использования V8.
- Хуже чем core i7 FCG в ~7 раз.

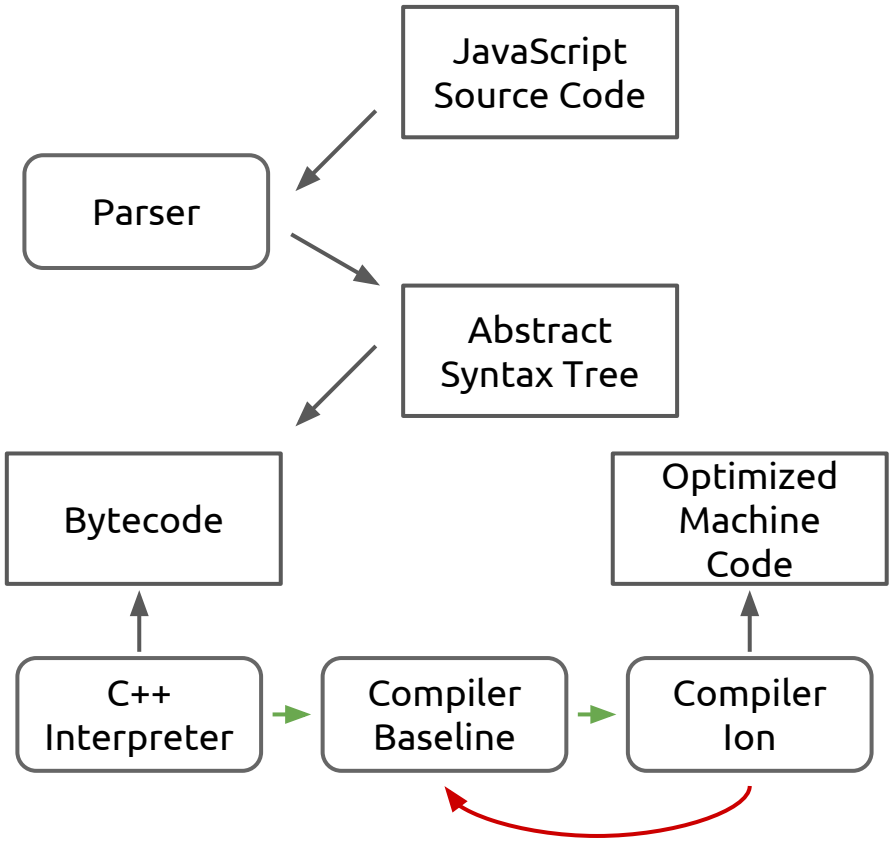

Эх, FullCodegen удался

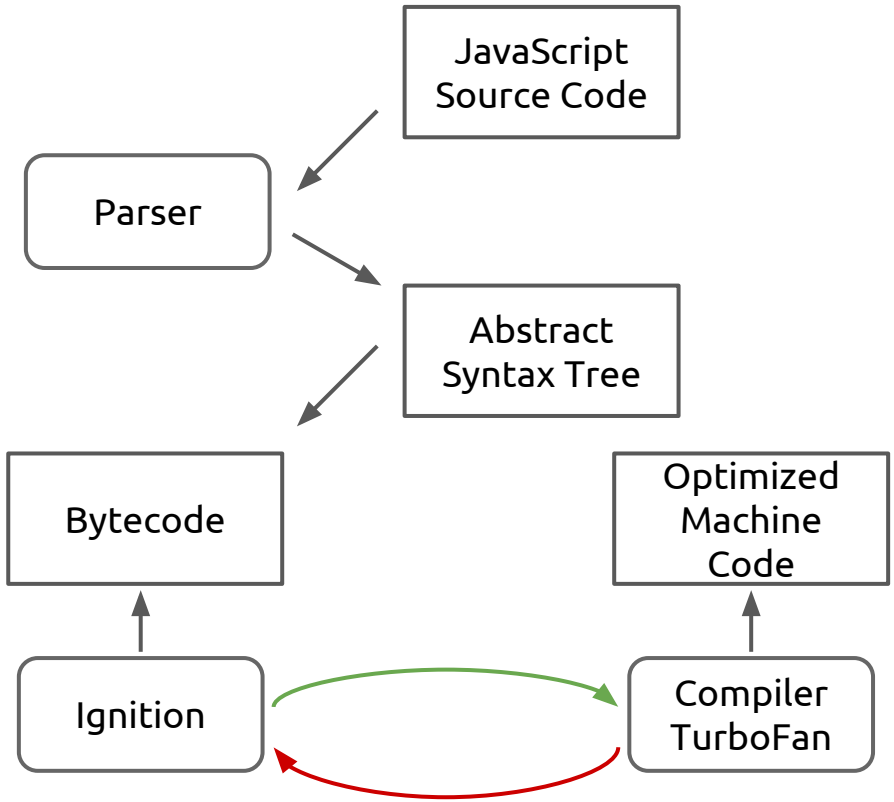
Tuesday, August 23, 2016

Firing up the Ignition Interpreter

For the first time, Ignition and TurboFan are used universally and exclusively for JavaScript execution in V8 5.9. Furthermore, starting with 5.9, Full-codegen and Crankshaft, the technologies that served V8 well since 2010, are no longer used in V8 for JavaScript execution, since they no longer are able to keep pace with new JavaScript language features and the optimizations those features require. We plan to remove them completely very soon. That means

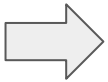
Firefox®


JavaScript Source Code
Parser
Abstract Syntax Tree
Bytecode
Optimized Machine Code
C++ Interpreter
Compiler Baseline
Compiler Ion

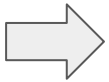

JavaScript Source Code
Parser
Abstract Syntax Tree
Bytecode
Optimized Machine Code
Ignition
Compiler TurboFan

Что такое Baseline?

```
function foo() {
    var b = 5;
    return b + 3;
}
```



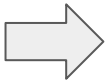
int8
setlocal
pop
getlocal
int8
add
return



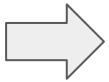
```
frame.push(INT32);

storeValue(frame.addressOfLocal(local), R0);

frame.pop();

frame.pushLocal(GET_LOCALNO(pc));

frame.push(Int32Value(GET_INT8(pc)));

Call_ICBinaryArith_Fallback
frame.push(R0);

frame.popValue(JSReturnOperand);
```

Что такое Baseline?

```
function foo() {
    var b = 5;
    return b + 3;
}
```



int8
setlocal
pop
getlocal
int8
add
return



frame.push(INT32);

storeValue(frame.addressOfLocal(local), R0);

frame.pop();

frame.pushLocal(GET_LOCALNO(pc));

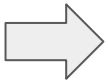
frame.push(Int32Value(GET_INT8(pc)));

Call_ICBinaryArith_Fallback
frame.push(R0);

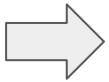
frame.popValue(JSReturnOperand);

Что такое Baseline?

```
function foo() {
    var b = 5;
    return b + 3;
}
```



int8
setlocal
pop
getlocal
int8
add
return



frame.push(INT32);

storeValue(frame.addressOfLocal(local), R0);

frame.pop();

frame.pushLocal(GET_LOCALNO(pc));

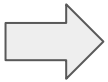
frame.push(Int32Value(GET_INT8(pc)));

Call_ICBinaryArith_Fallback
frame.push(R0);

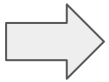
frame.popValue(JSReturnOperand);

Что такое Baseline?

```
function foo() {
    var b = 5;
    return b + 3;
}
```



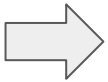
```
int8
setlocal
pop
getlocal
int8
add
return
```



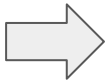
```
frame.push(INT32);

storeValue(frame.addressOfLocal(local), R0);

frame.pop();

frame.pushLocal(GET_LOCALNO(pc));

frame.push(Int32Value(GET_INT8(pc)));

Call_ICBinaryArith_Fallback
frame.push(R0);

frame.popValue(JSReturnOperand);
```

Что такое Baseline?

```
function foo() {
    var b = 5;
    return b + 3;
}
```



int8
setlocal
pop
getlocal
int8
add
return



frame.push(INT32);

storeValue(**frame**.addressOfLocal(local), R0);

frame.pop();

frame.pushLocal(GET_LOCALNO(pc));

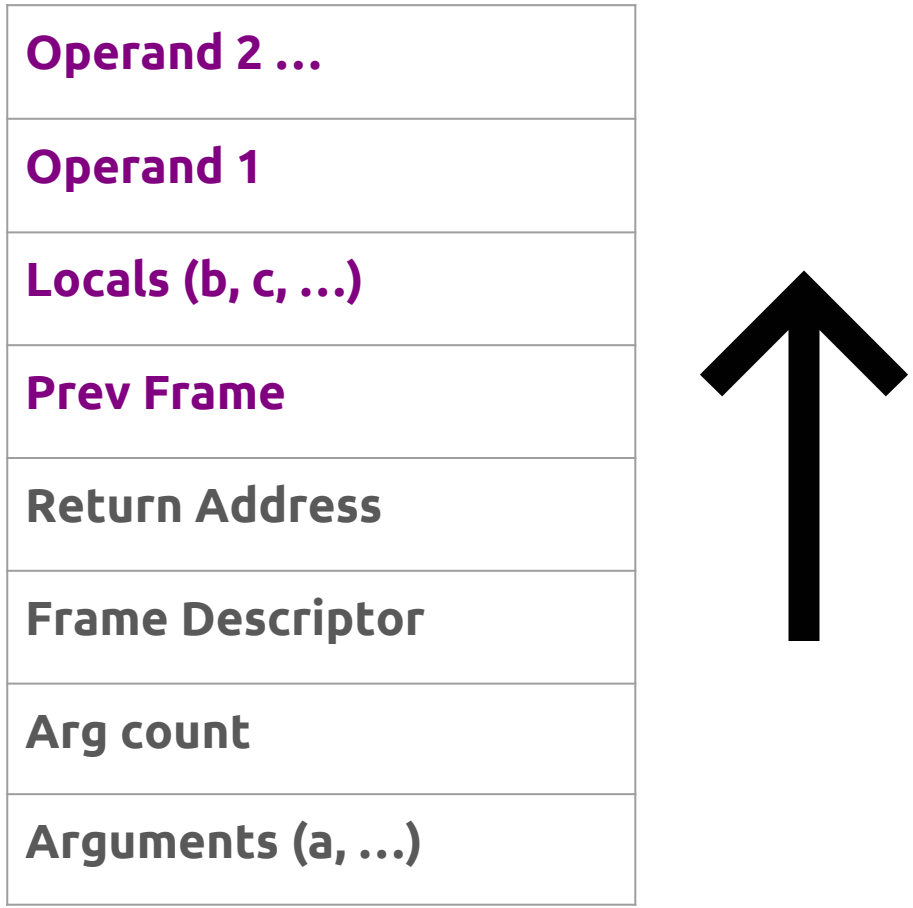
frame.push(Int32Value(GET_INT8(pc)));

Call_ICBinaryArith_Fallback
frame.push(R0);

frame.popValue(JSReturnOperand);

Frames

```
function Foo(a) {
  let b = a.bar();
  var c;
  ...
  return b;
}
```



Baseline

Достоинства

- Портируем ассемблер, получаем рабочий компилятор
- Удобно отлаживать проблемы
- Любой стаб можно переписать

Недостатки

- Линейный код
- Особо не по оптимизируешь

Портирование Baseline. С чего начать?

Trampolines - архитектурно зависимые стабы.

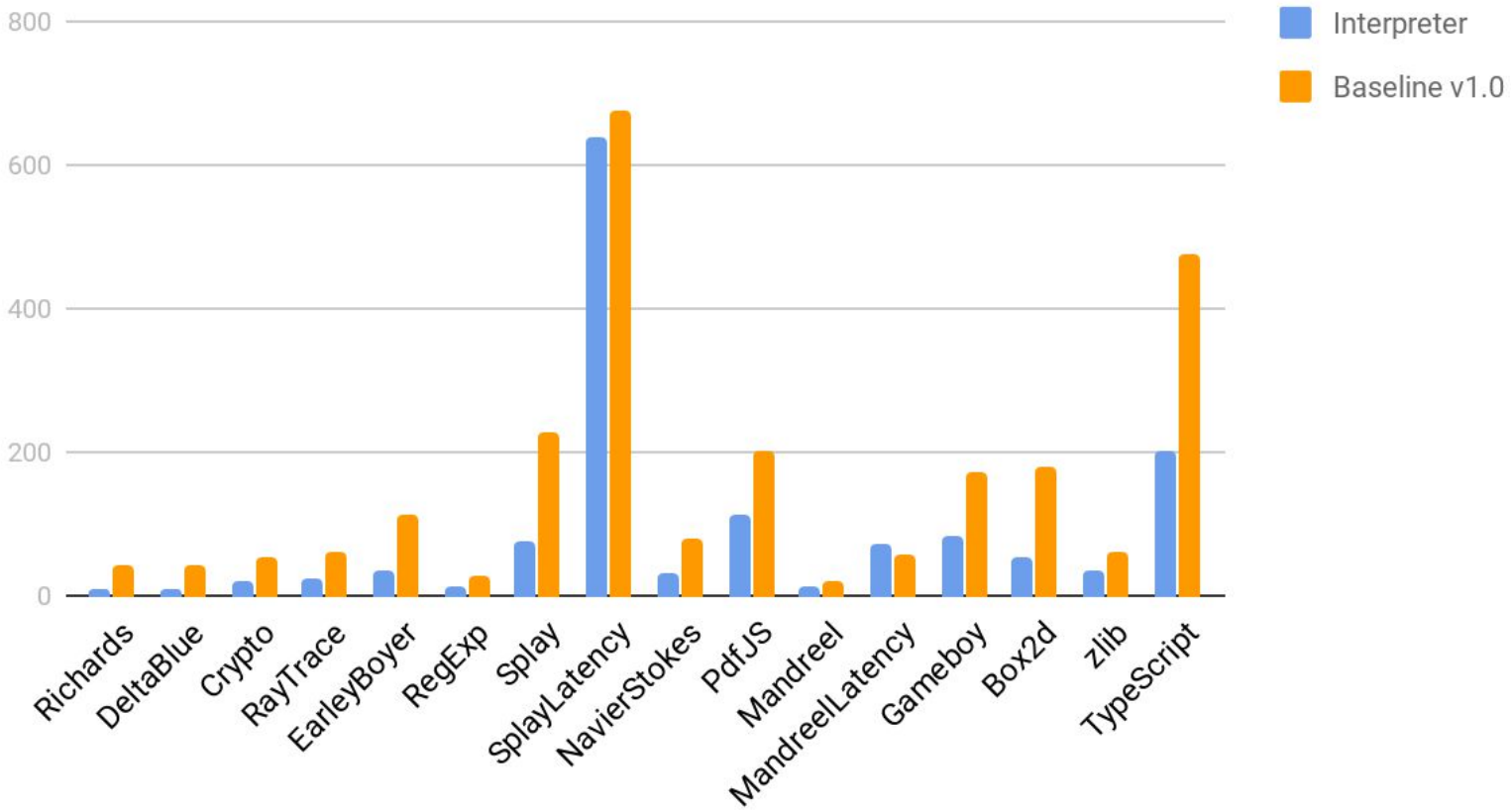
- **EnterJit** - обертка для вызова нативного кода из рантайма
- **VMWrapper** - обертка для вызова рантайма из нативного кода

Пролог метода

- Создание фрейма
- Выделение стека

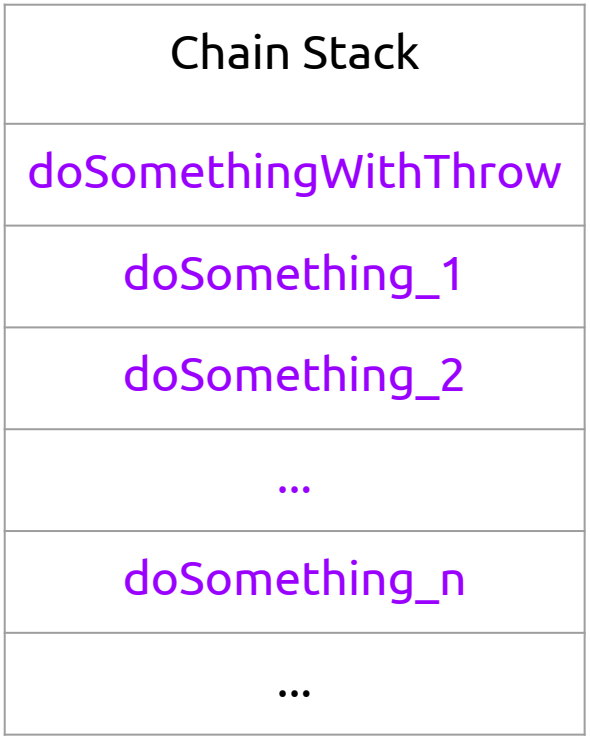

Немного шлифовки

Octane


800
600
400
200
0
Interpreter
Baseline v1.0
Richards
DeltaBlue
Crypto
RayTrace
EarleyBoyer
RegExp
Splay
SplayLatency
NavierStokes
PdfJS
Mandreel
MandreelLatency
Gameboy
Box2d
zlib
TypeScript

Обработка исключений

```
function doSomethingWithThrow() {
    if (mistake()) {
        throw "Mistake";
    } else {
        doSomethingElse();
    }
}

function doSomething_1() {
    doSomethingWithThrow();
}
.........

try {
    doSomething_n();
}
catch (exception) {
    ......
}
```



Обработка исключений

Throw - вызов рантайма.

VMWrapper при обработке возвращаемого значения совершает прыжок в trampoline, вызывающий обработчик исключений.

В обработчике:

- итерируемся по фреймам на стеке
- находим обработчик

Обработка исключений

X86

1) Подмена адреса возврата

Эльбрус

1) Подсчет сколько вызовов отмотать

Обработка исключений

X86

1) Подмена адреса возврата

Эльбрус

1) Подсчет сколько вызовов отмотать
2) Системный вызов для получения стека вызовов

Обработка исключений

X86

1) Подмена адреса возврата

Эльбрус

1) Подсчет сколько вызовов отмотать
2) Системный вызов для получения стека вызовов
3) Замена адресов на адрес стаба делающего возврат

Обработка исключений

X86

1) Подмена адреса возврата

Эльбрус

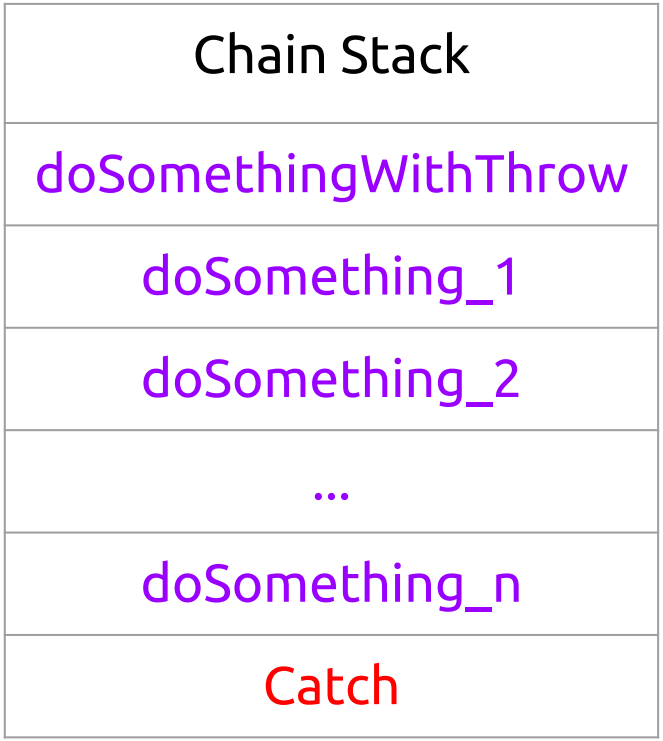
1) Подсчет сколько вызовов отмотать
2) Системный вызов для получения стека вызовов
3) Замена адресов на адрес стаба делающего возврат
4) Системный вызов для перезаписи стека вызовов

```
function doSomethingWithThrow() {
    If (mistake()) {
        throw “Mistake”;
    } else {
        doSomethingElse();
    }
}

function doSomething_1() {
    doSomethingWithThrow();
}

.........
try {
    doSomething_n();
}
catch (exception) {
    ……...
}
```

X86

```
function doSomethingWithThrow() {
    If (mistake()) {
        throw “Mistake”;
    } else {
        doSomethingElse();
    }
}

function doSomething_1() {
    doSomethingWithThrow();
}

.........
try {
    doSomething_n();
}
catch (exception) {
    ..........
}
```

Эльбрус



```
function doSomethingWithThrow() {
    If (mistake()) {
        throw “Mistake”;
    } else {
        doSomethingElse();
    }
}

function doSomething_1() {
    doSomethingWithThrow();
}

.........
try {
    doSomething_n();
}
catch (exception) {
    ..........
}
```

Эльбрус

Chain Stack
doSomethingWithThrow
doSomething_1
doSomething_2
...
doSomething_n
Catch

```
function doSomethingWithThrow() {
    If (mistake()) {
        w “Mistake”;
    } e
        ethingElse();
    }
}

func
        1() {
        row();
}

.....
try
    {
}
catch (exception) {
    .....
}
```

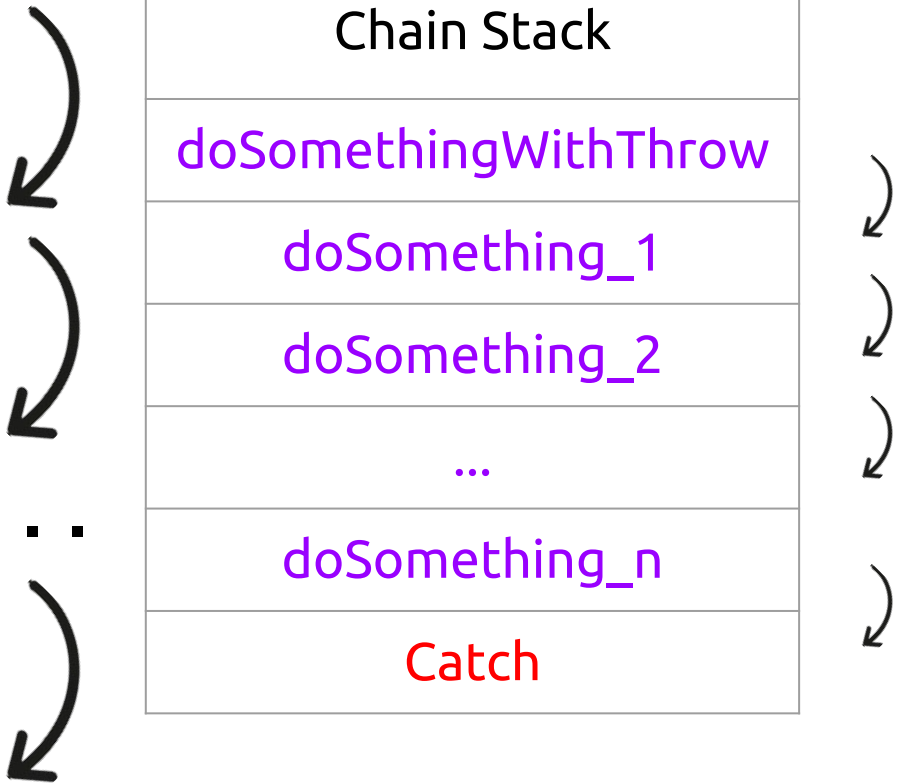
. . .

Эльбрус

Chain Stack
doSomethingWithThrow
doSomething_1
doSomething_2
...
doSomething_n
Catch

```
function doSomethingWithThrow() {
    If (mistake()) {
        throw “Mistake”;
    } else {
        doSomethingElse();
    }
}

function doSomething_1() {
    doSomethingWithThrow();
}

.........
try {
    doSomething_n();
}
catch (exception) {
    ..........
}
```

Эльбрус



Простейшие оптимизации Baseline

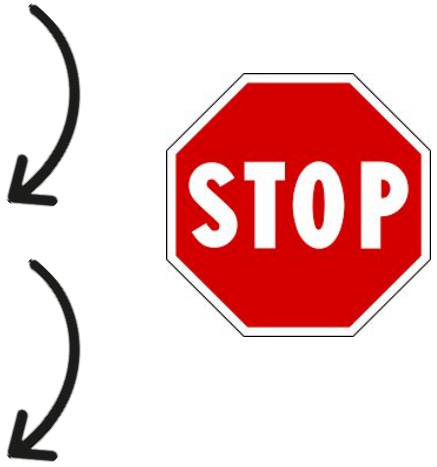
- Переписывание инлайн кэшей
- Ручная расстановка задержек

Расстановка задержек

```
ldd, 0 %dr0, 0x0, %r0
ldd, 2 %dr0, 0x4, %r1
ldd, 3 %dr1, 0x0, %r2
ldd, 5 %dr1, 0x4, %r3
```

```
fmuld, 0 %r0, %r1, %r0
fmuld, 3 %r2, %r3, %r1
```

```
faddd, 0 %r0, %r1, %r0
```



4 + 8 тактов

Расстановка задержек

```
ldd, 0 %dr0, 0x0, %r0
ldd, 2 %dr0, 0x4, %r1
ldd, 3 %dr1, 0x0, %r2
ldd, 5 %dr1, 0x4, %r3
nop, 2
```

2 такта

```
fmuld, 0 %r0, %r1, %r0
fmuld, 3 %r2, %r3, %r1
nop, 5
```

5 тактов

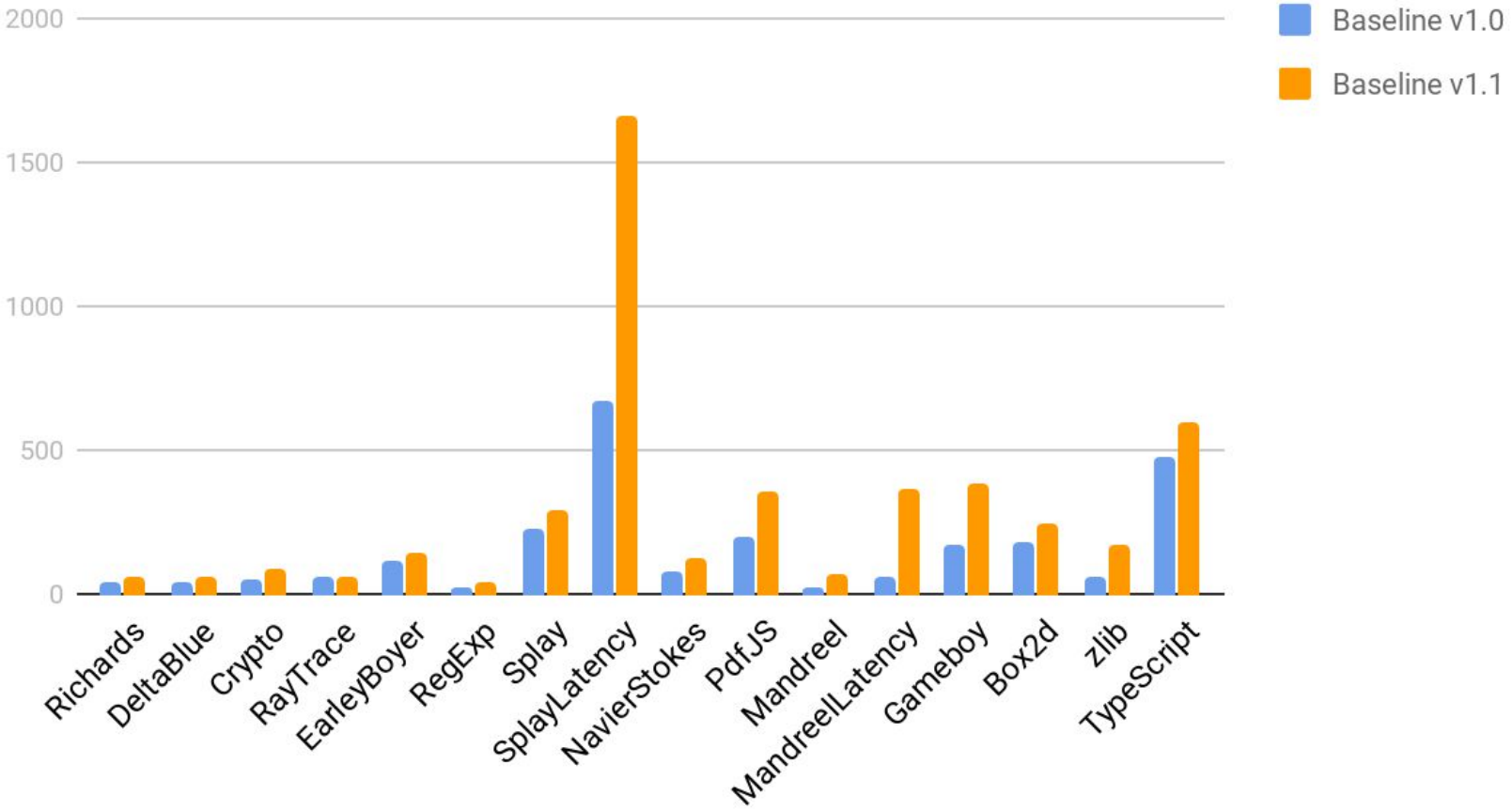
```
faddd, 0 %r0, %r1, %r0
```

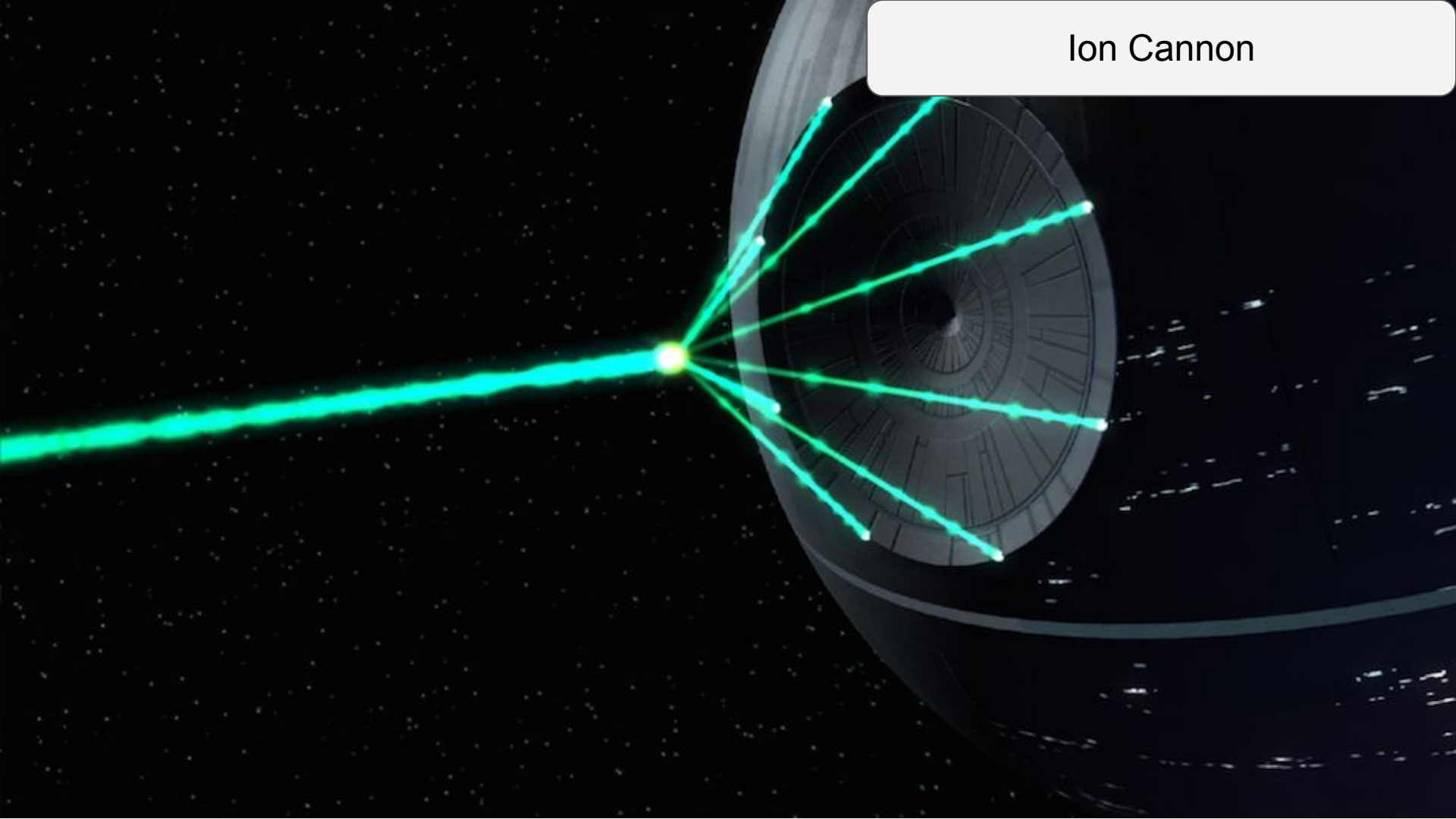
Простейшие оптимизации Baseline

- Ручное переписывание инлайн кэшей
- ~~Ручная расстановка задержек~~
- Автоматическая расстановка задержек
- Вынос подготовок переходов

Что получили?

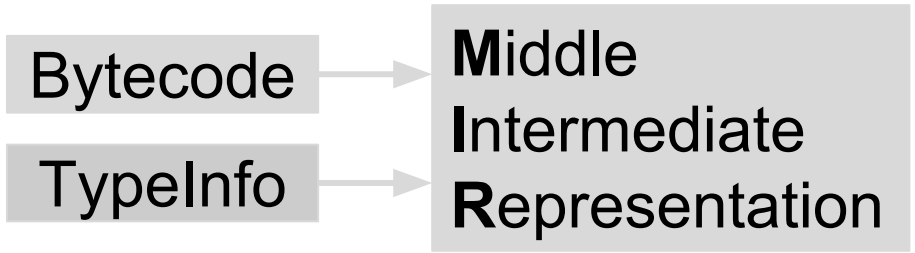
Octane


2000
1500
1000
500
0
Baseline v1.0
Baseline v1.1
Richards
DeltaBlue
Crypto
RayTrace
EarleyBoyer
RegExp
Splay
SplayLatency
NavierStokes
PdfJS
Mandreel
MandreelLatency
Gameboy
Box2d
zlib
TypeScript


Ion Cannon

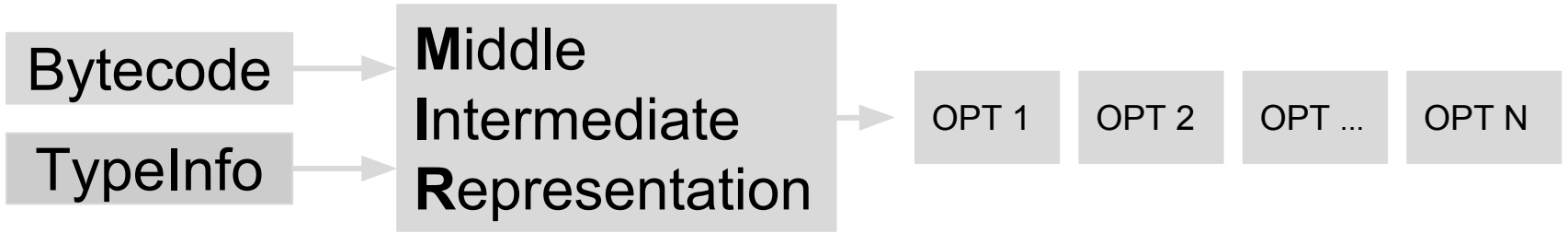
Ion первый оптимизирующий

Устройство и архитектура



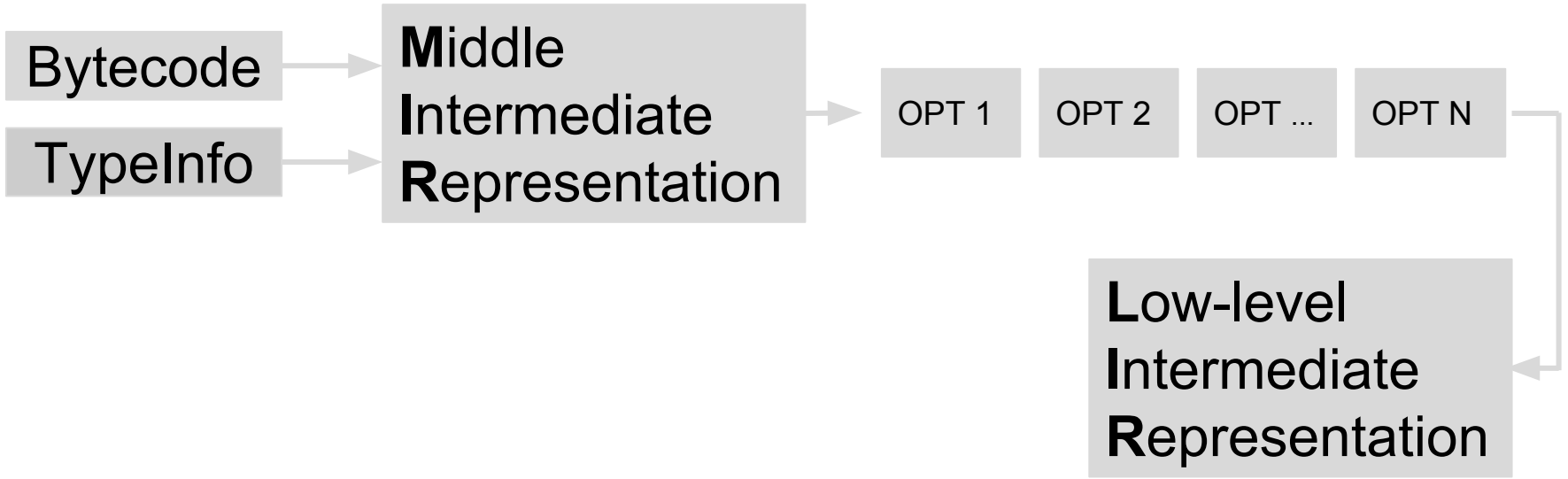
Ion первый оптимизирующий

Устройство и архитектура



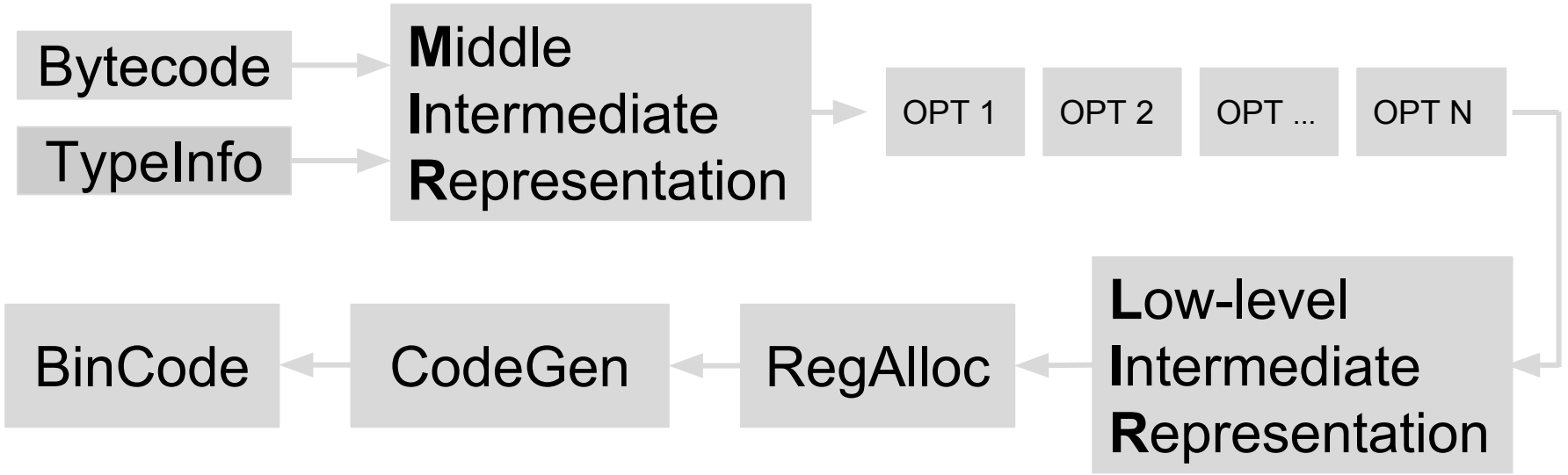
Ion первый оптимизирующий

Устройство и архитектура



Ion первый оптимизирующий

Устройство и архитектура



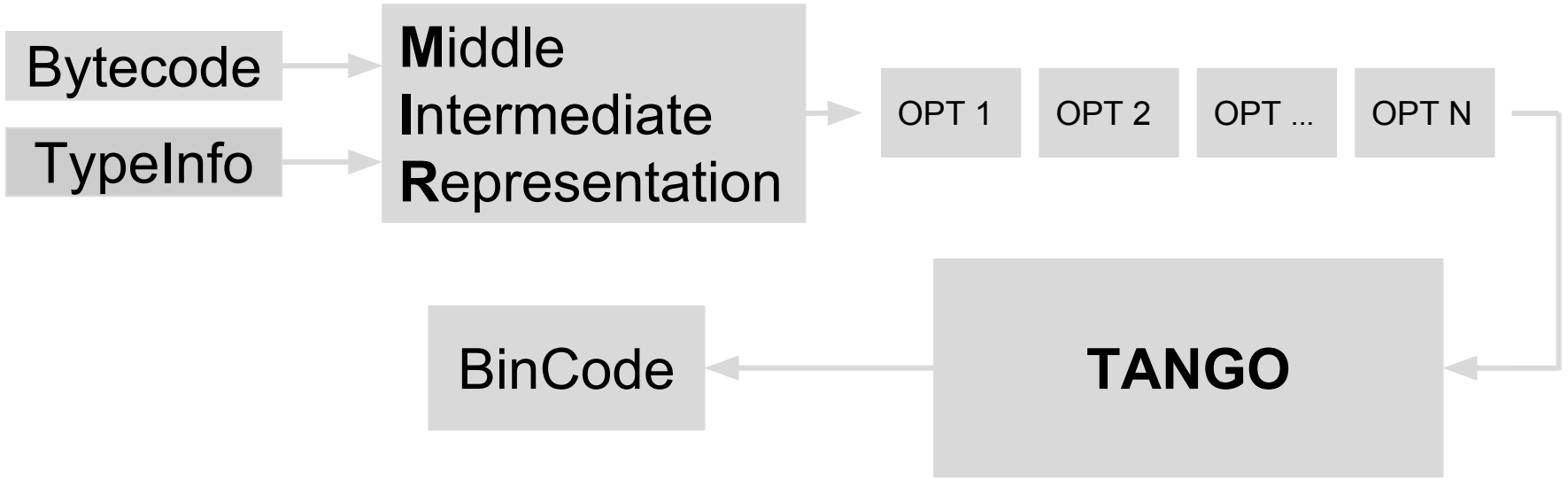
Ion первый оптимизирующий

- Нужен свой планировщик команд
- Свой регистр аллокатор
- Свой LIR
- Свой Code-generator

Что будем делать?

Ion первый оптимизирующий

Устройство и архитектура



Преимущества использования Tango

- Планировщик команд
- Регистр аллокатор
- Низкоуровневые оптимизации (if-conv, loop-unrolling …)

Мысли по поводу селектора

- Сложно поддерживать напрямую отображающийся в Tango
- Код перевода из MIR в Tango должен быть понимаемым и высокоуровневым

Решение по поводу селектора

- Писать свой мини язык для перевода
- HolyJit: A New Hope [3]

Ion деоптимизации

```
function foo(a, b) {
    return a + b;
}

function doSomeStuff(obj) {
    for (let i = 0; i < 1100; ++i) {
      print(foo(obj, obj));
    }
}

doSomeStuff("HolyJS");
doSomeStuff({n: 10});
```

Ion деоптимизации

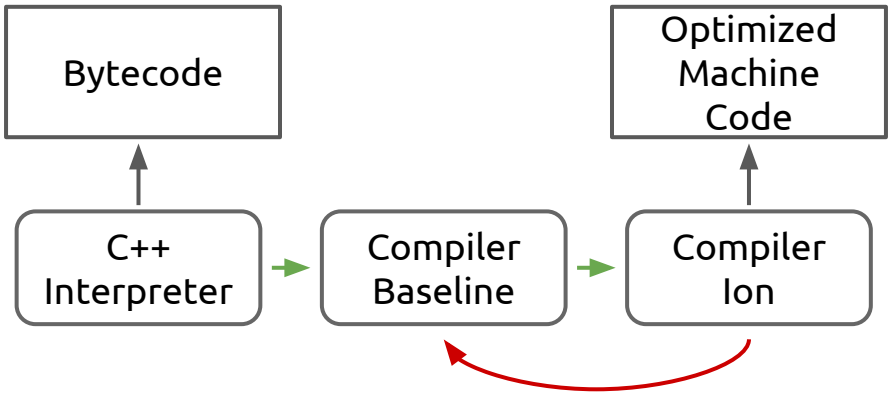
```
function foo(a, b) {
    return a + b;
}

function doSomeStuff(obj) {
    for (let i = 0; i < 1100; ++i) {
      print(foo(obj, obj));
    }
}

doSomeStuff("HolyJS");
doSomeStuff({n: 10});
```

IONFLAGS=bailouts,scripts

--ion-offthread-compile=off

```
HolyJSHolyJS
HolyJSHolyJS
HolyJSHolyJS
HolyJSHolyJS
HolyJSHolyJS
HolyJSHolyJS
HolyJSHolyJS
HolyJSHolyJS
HolyJSHolyJS
HolyJSHolyJS
HolyJSHolyJS
HolyJSHolyJS
[IonScripts] Compiling script deopt.js:5 (7f22b59921c0) (warmup-counter=1100, level=Optimization_Normal)
[IonScripts] Inlining script deopt.js:1 (7f22b5992258)
HolyJSHolyJS
HolyJSHolyJS
HolyJSHolyJS
[IonBailouts] Took bailout! Snapshot offset: 0
[IonBailouts]  bailing from bytecode: nop, MIR: start [0], LIR: phi [0]
[object Object][object Object]
[object Object][object Object]
[object Object][object Object]
[object Object][object Object]
[object Object][object Object]
[object Object][object Object]
[object Object][object Object]
[object Object][object Object]
[object Object][object Object]
[object Object][object Object]
```

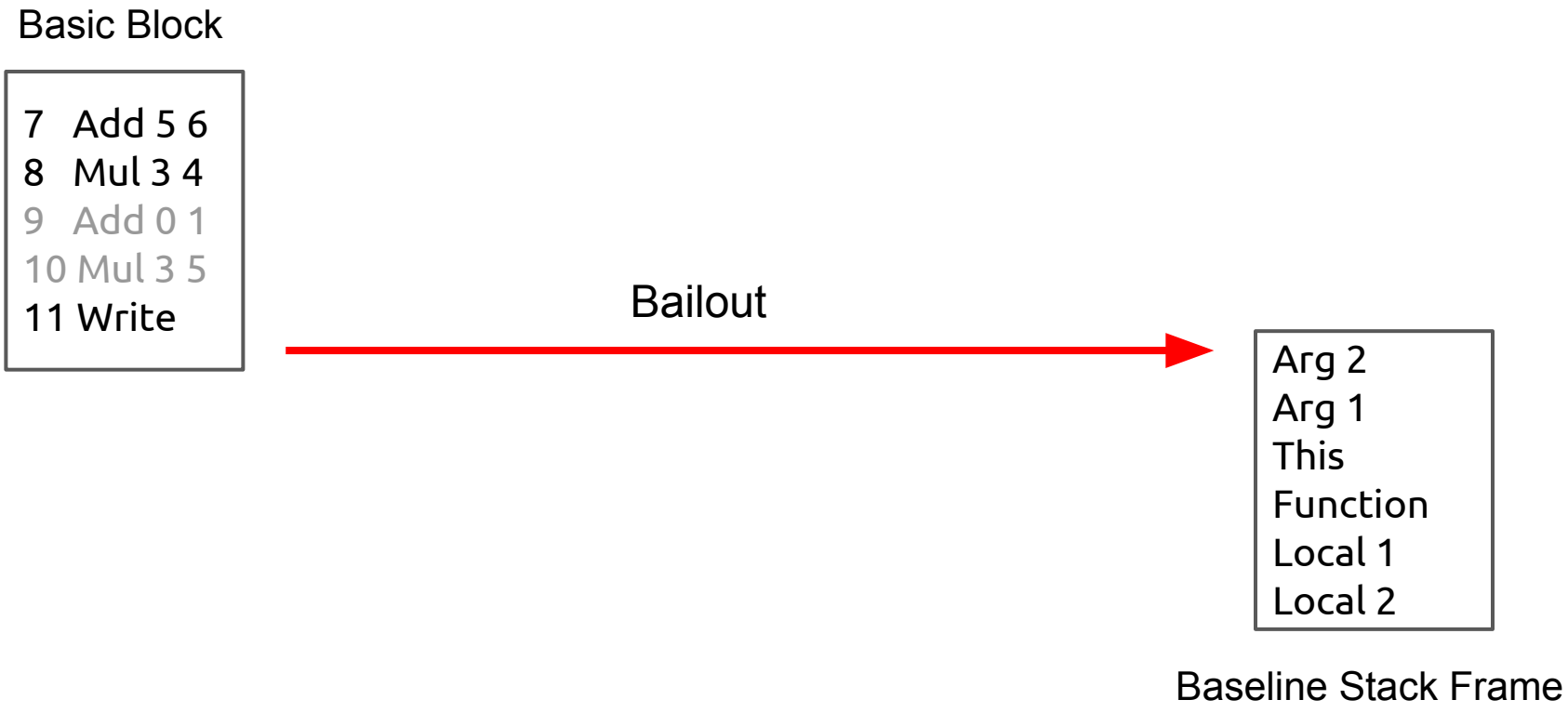
Ion деоптимизации



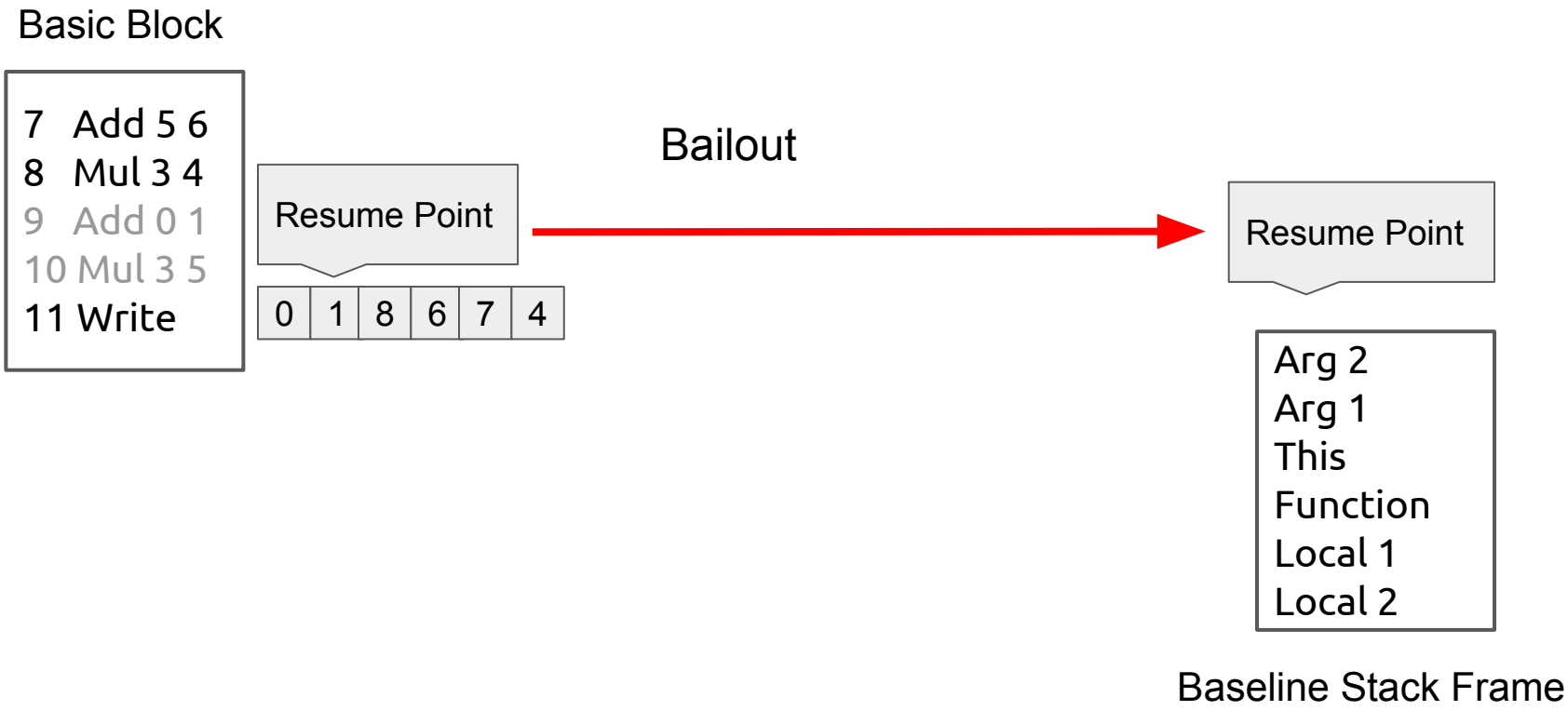
Ion деоптимизации

```
function doSomeStuff(obj) {
  for (let i = 0; i < 1100; ++i) {
     If (!(obj instanceof String))
         // bailout!
    print(foo_only_str(obj, obj));
  }
}
```

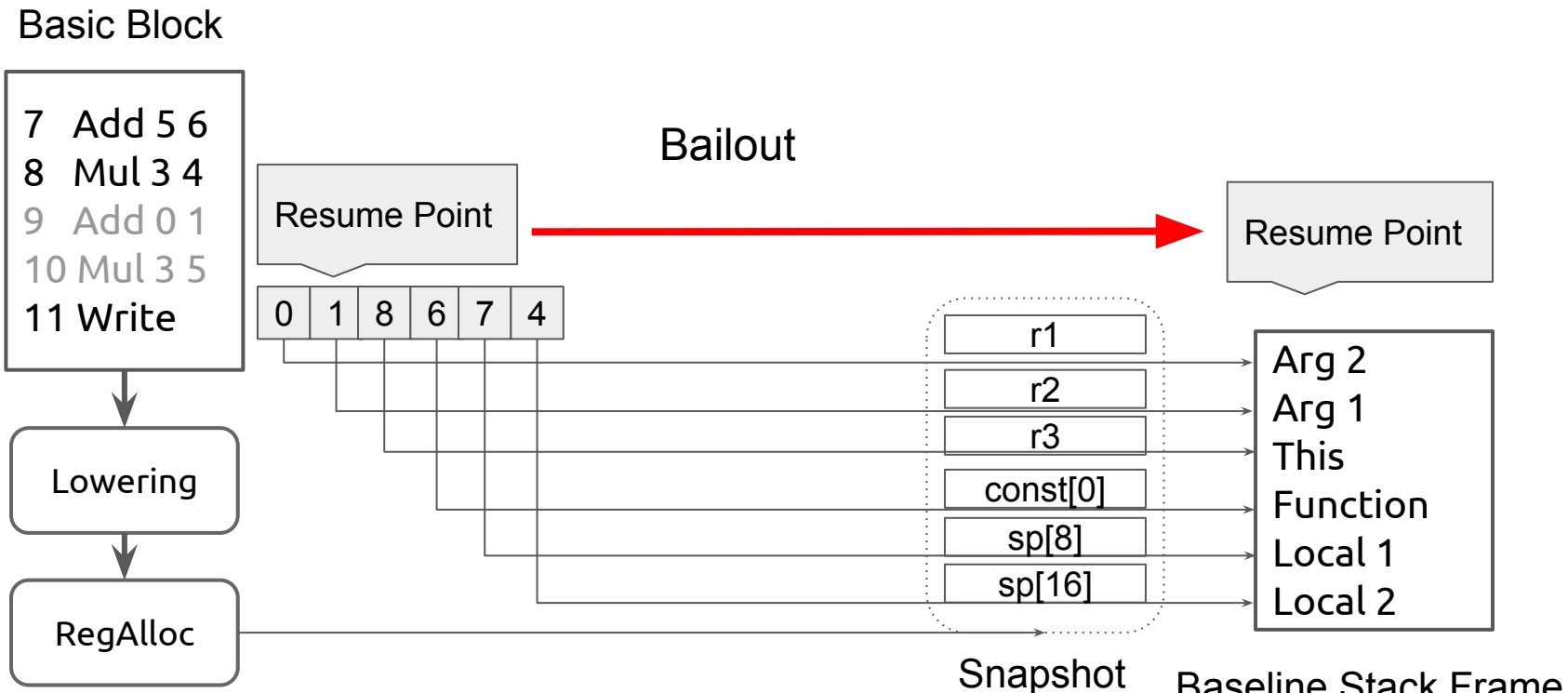
Ion деоптимизации



Ion деоптимизации



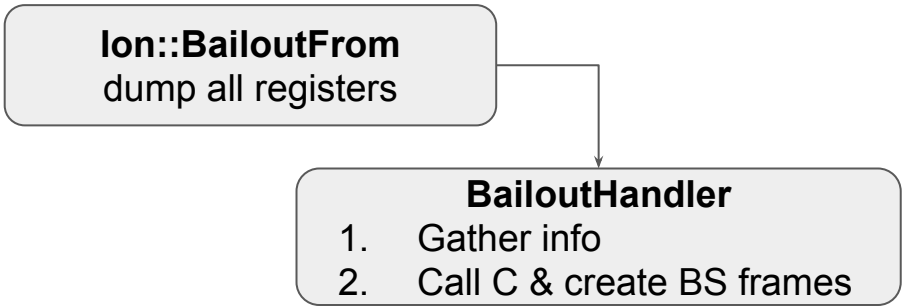
Ion деоптимизации



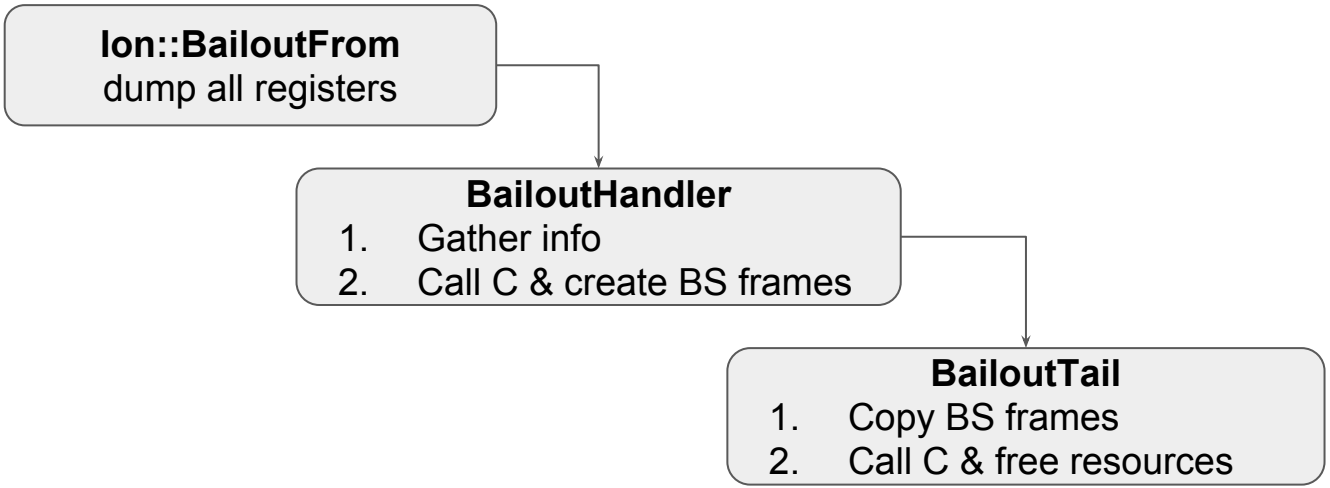
Деоптимизация в x86

Ion::BailoutFrom
dump all registers

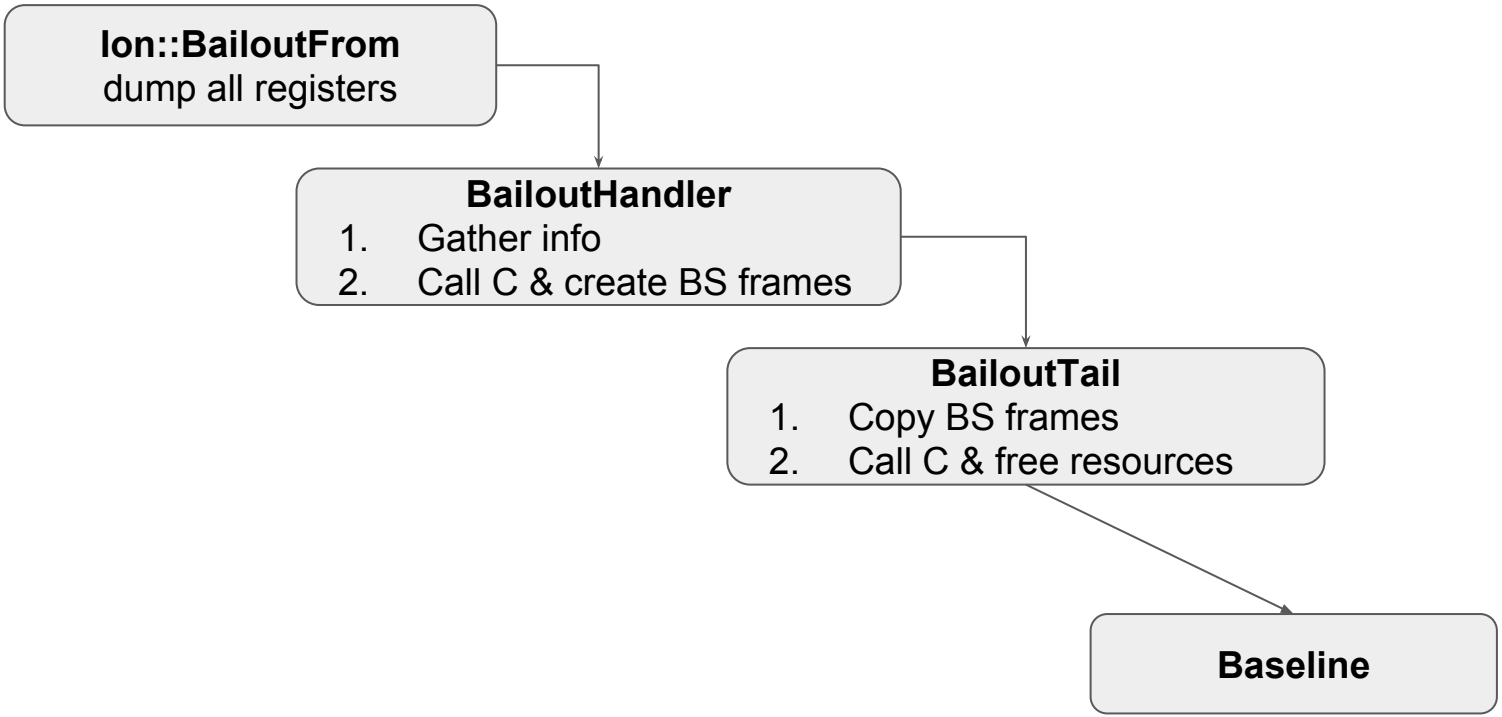
Деоптимизация в x86



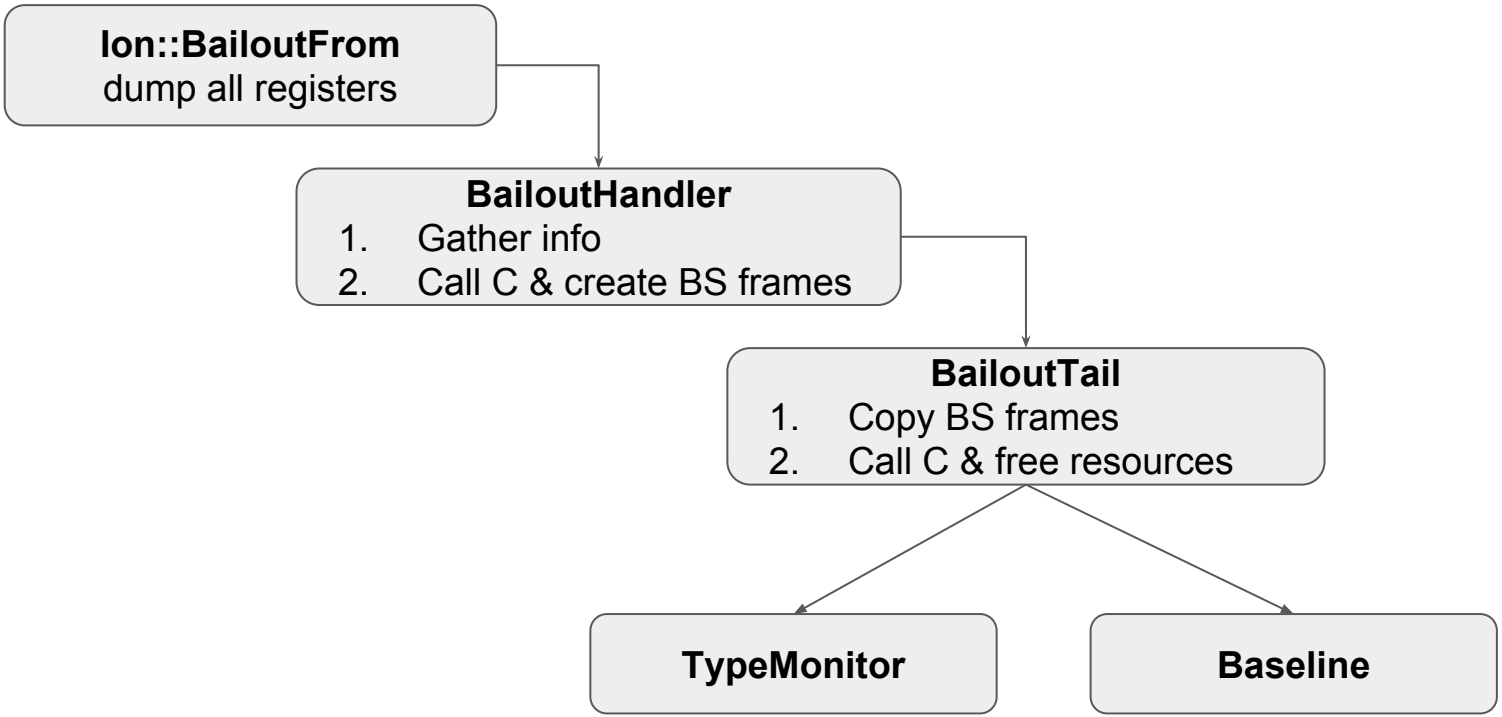
Деоптимизация в x86



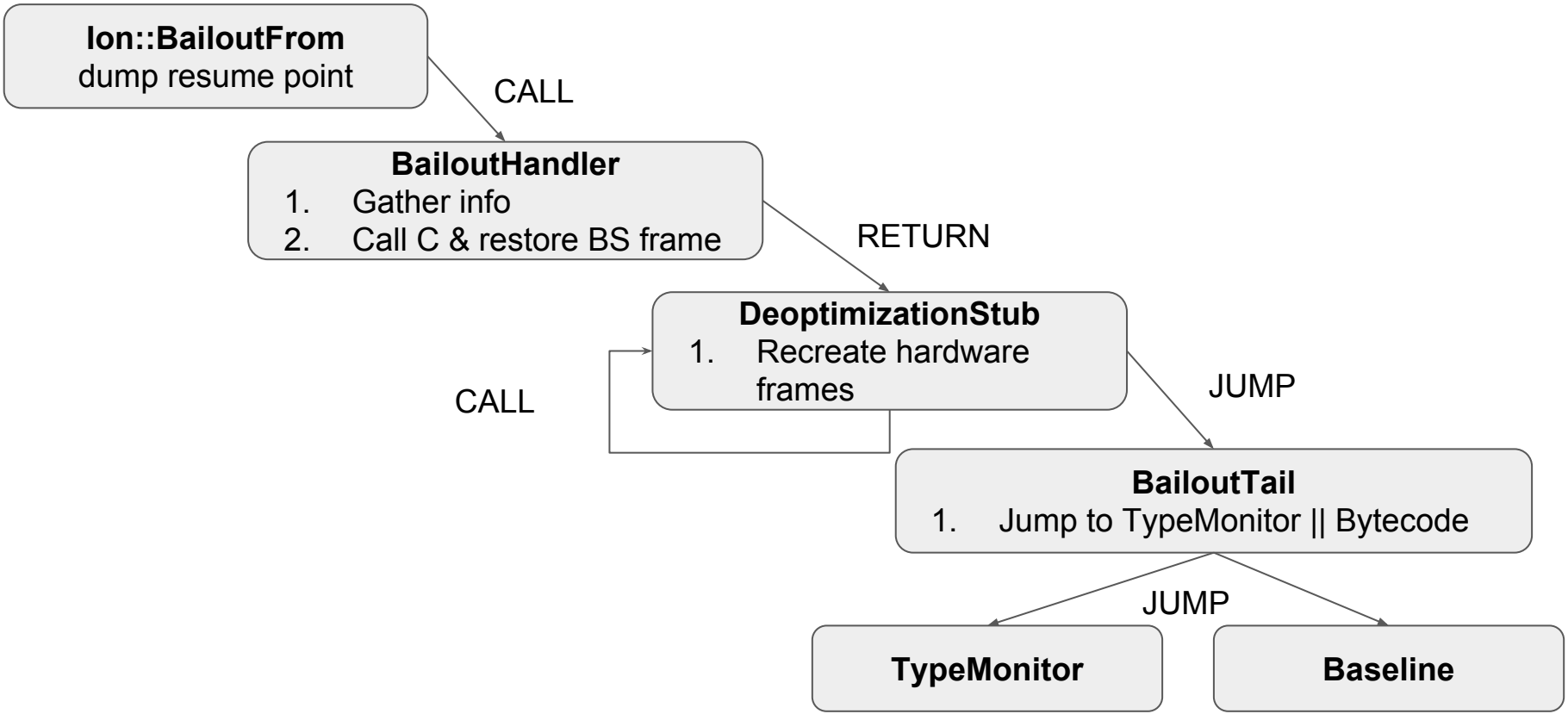
Деоптимизация в x86



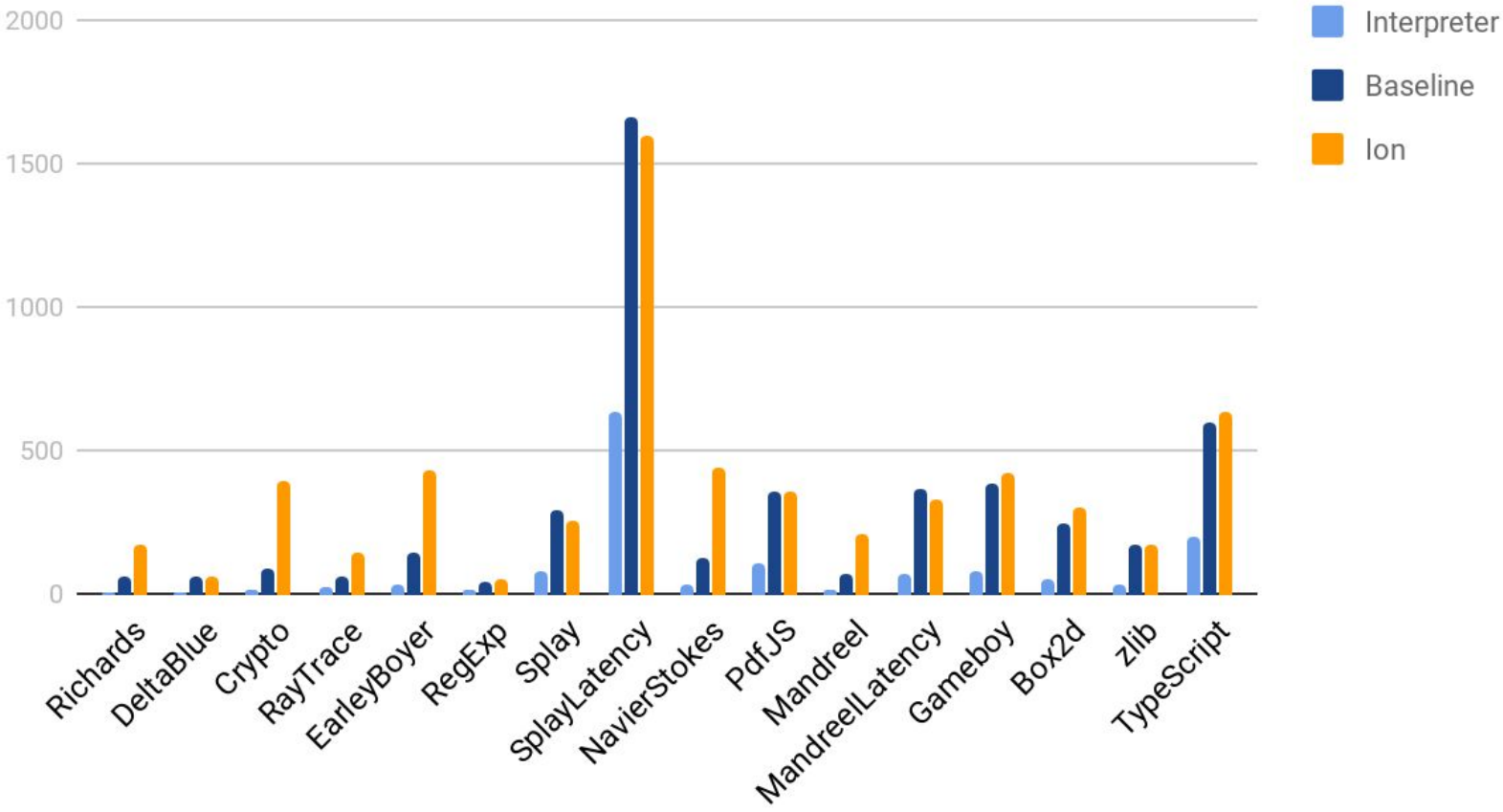
Деоптимизация в x86



Наша переработанная схема



Octane


2000
1500
1000
500
0
Interpreter
Baseline
Ion
Richards
DeltaBlue
Crypto
RayTrace
EarleyBoyer
RegExp
Splay
SplayLatency
NavierStokes
PdfJS
Mandreel
MandreelLatency
Gameboy
Box2d
zlib
TypeScript

Что вы узнали сегодня

1. SpiderMonkey, V8, Node есть на Эльбрусе
2. Портирование может быть выполнено небольшой командой
3. Подводные камни портирования в виде деоптимизации и медленной работы стабов

Спасибо за внимание

1. Ануфриенко Андрей
2. Ануфриенко Владимир
3. Бежецков Дмитрий
4. Сябро Максим
5. Тлеукенов Тимур
6. Исаченко Владимир
7. Матвеев Андрей


HOLY JS
JAVASCRIPT-КОНФЕРЕНЦИИ
МЦСТ
ЭЛЬБРУС
SAMSUNG

Ссылки

1. https://habrahabr.ru/company/redsys/blog/337730/
2. http://channel9.msdn.com/Shows/Going+Deep/Expert-to-Expert-Erik-Meijer-and-Lars-Bak-Inside-V8-A-Javascript-Virtual-Machine
3. https://blog.mozilla.org/javascript/2017/10/20/holyjit-a-new-hope/

	Эльбрус-2С+	Эльбрус-4С	Эльбрус-8С	Эльбрус-16С
Год выпуска	2011	2014	2015-2018 (доработки)	2018 (план)
Тактовая частота	500 МГц	800 Мгц	1300 МГц	1500 Мгц
Разрядность	хз	32/64 бит	64 бит	64/128 бит
К-во ядер	2	4	8	8/16
Кэш первого уровня	64 Кб	128 Кб	—	—
Кэш второго уровня	1 Мб	8 Мб	4 Мб	4 Мб
Кэш третьего уровня	—	—	16 Мб	16 Мб
Поддержка ОЗУ	DDR2-800	3 x DDR3-1600	4 x DDR3-1600	4 x DDR4-2400
Техпроцесс	90 нм	65 нм	28 нм	28 нм (или 16)
Потребление энергии	25 Вт	45 Вт	75-100 Вт	60-90 Вт