Falcon

новый JIT компилятор в Zing JVM

Артур Пилипенко
apilipenko@azul.com
@arturpilipenko

AZUL SYSTEMS®

Докладчик

- Артур Пилипенко
- Инженер в Azul Systems
- Разрабатываю виртуальные машины и компиляторы
- До Azul работал в Oracle над Java ME VM

О чем доклад?

- Общеобразовательный доклад про технологию
- Будет много технических деталей! 🤘

AZUL SYSTEMS®

План

- Что такое Falcon?
- Зачем писать новый компилятор?
- Что такое LLVM?
- Как сделать Java компилятор из C++ компилятора?
- Производительность Falcon против Oracle JDK

AZUL SYSTEMS®

Zing JVM

Zing: A better JVM

- Коммерческая серверная JVM
- Low Latency
  - C4 GC, ReadyNow!

AZUL SYSTEMS®

Что такое Falcon?

- Tier2 компилятор в Zing JVM
- Замена компилятору C2
- Построен на базе LLVM

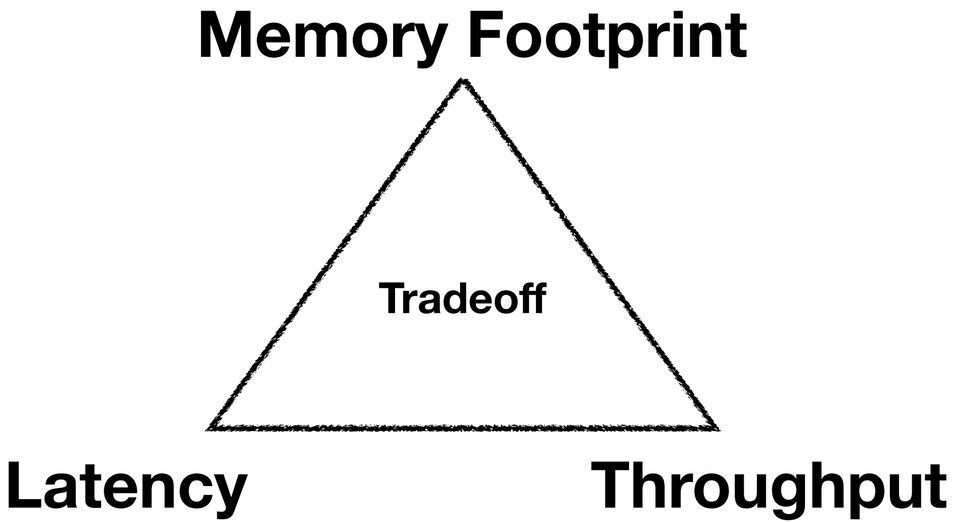
AZUL SYSTEMS®

Хронология разработки

Команда 4-6 человек,
~20 человеко-лет

Зачем писать новый компилятор?

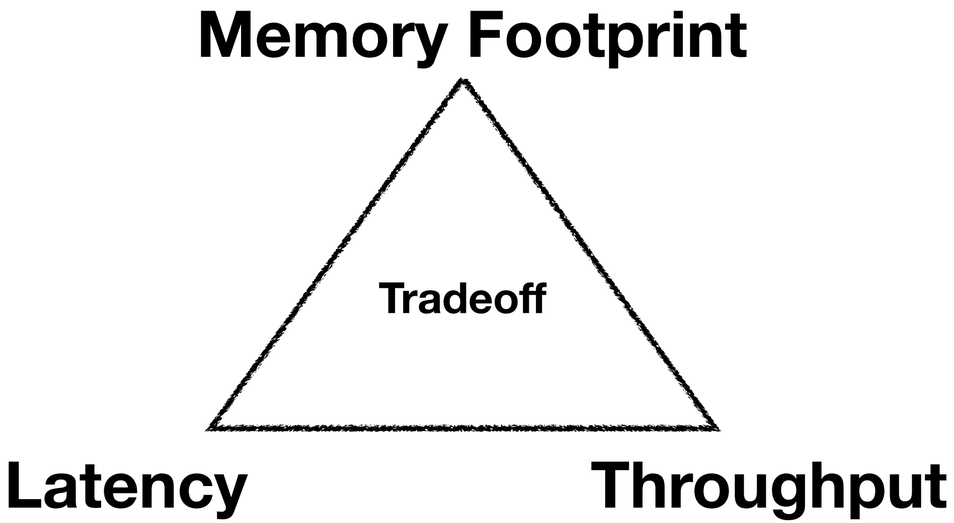
- Zing - Low Latency VM



AZUL SYSTEMS®

Зачем писать новый компилятор?

- Zing - Low Latency VM
- Хотим улучшить throughput



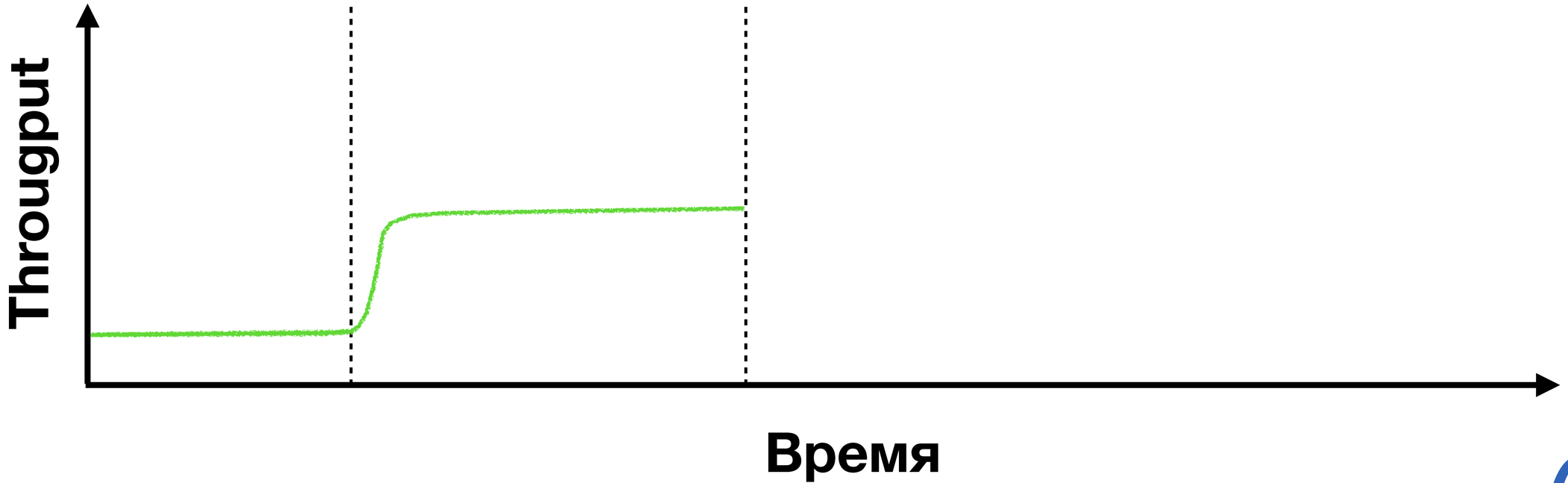
Многоуровневая компиляция

Многоуровневая компиляция

Интерпретатор

Многоуровневая компиляция

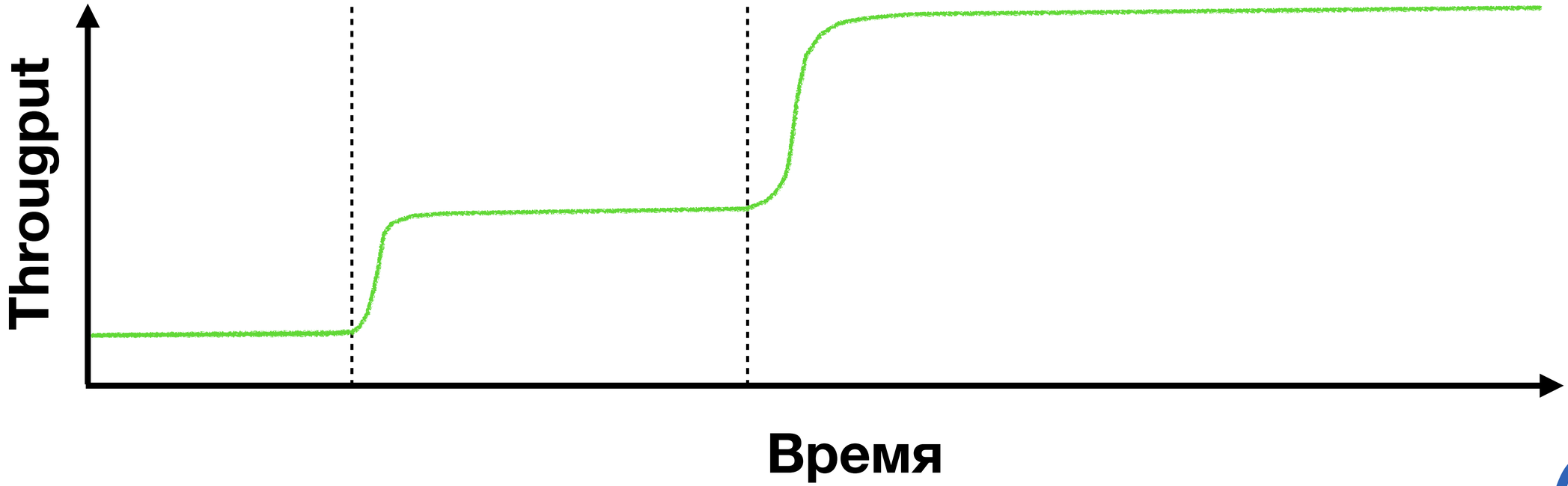
Интерпретатор → C1 компилятор



AZUL SYSTEMS®

Многоуровневая компиляция

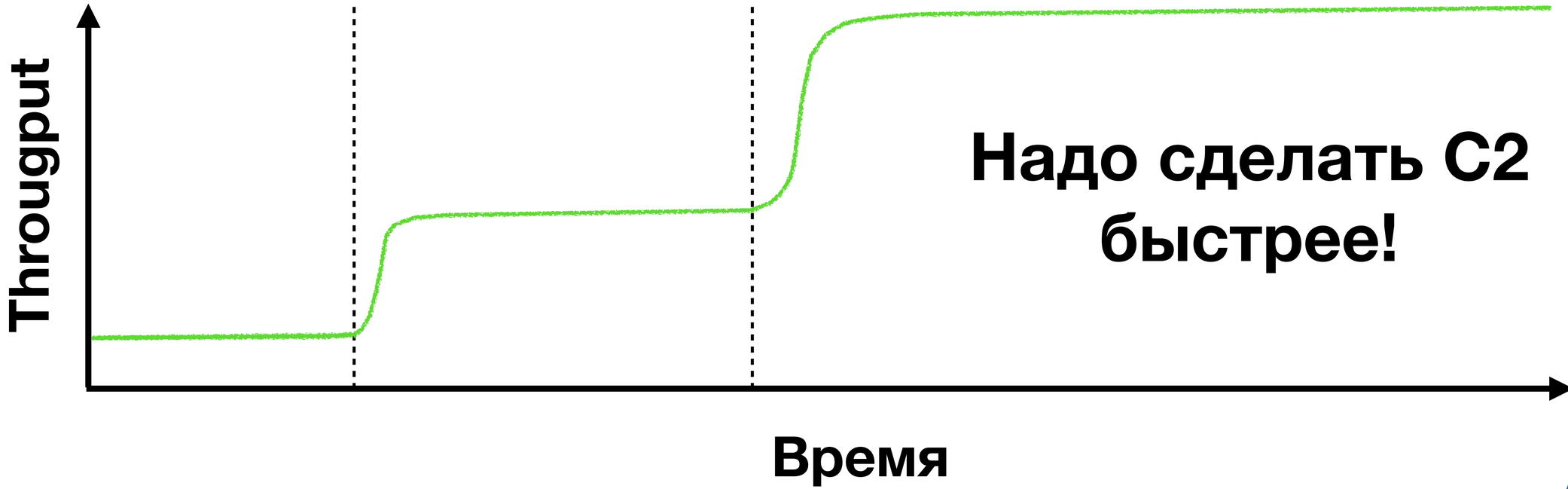
Интерпретатор → C1 компилятор → C2 компилятор



AZUL SYSTEMS®

Многоуровневая компиляция

Интерпретатор → C1 компилятор → C2 компилятор



AZUL SYSTEMS®

Компилятор C2

- Старый — компилятору больше 20 лет
  - 20 лет назад компьютеры были медленнее
  - Время компиляции было важнее
  - Стоимость абстракций была высока

Компилятор C2

- Старый — компилятору больше 20 лет
  - 20 лет назад компьютеры были медленнее
  - Время компиляции было важнее
  - Стоимость абстракций была высока
- Сложный — монолитный дизайн

Эволюция C2

- Накоплен технический долг
- Внесение изменений сложно и долго
- Очень мало людей обладают экспертизой в C2

Альтернативы?

Альтернативы

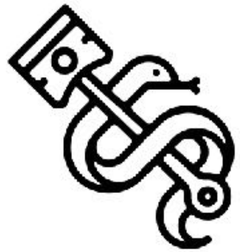
- Писать компилятор с нуля долго и сложно
- Oracle Graal - не даст конкурентного преимущества перед OpenJDK
- LLVM?

“The LLVM Project is a collection of modular and reusable compiler and toolchain technologies”

– *llvm.org*

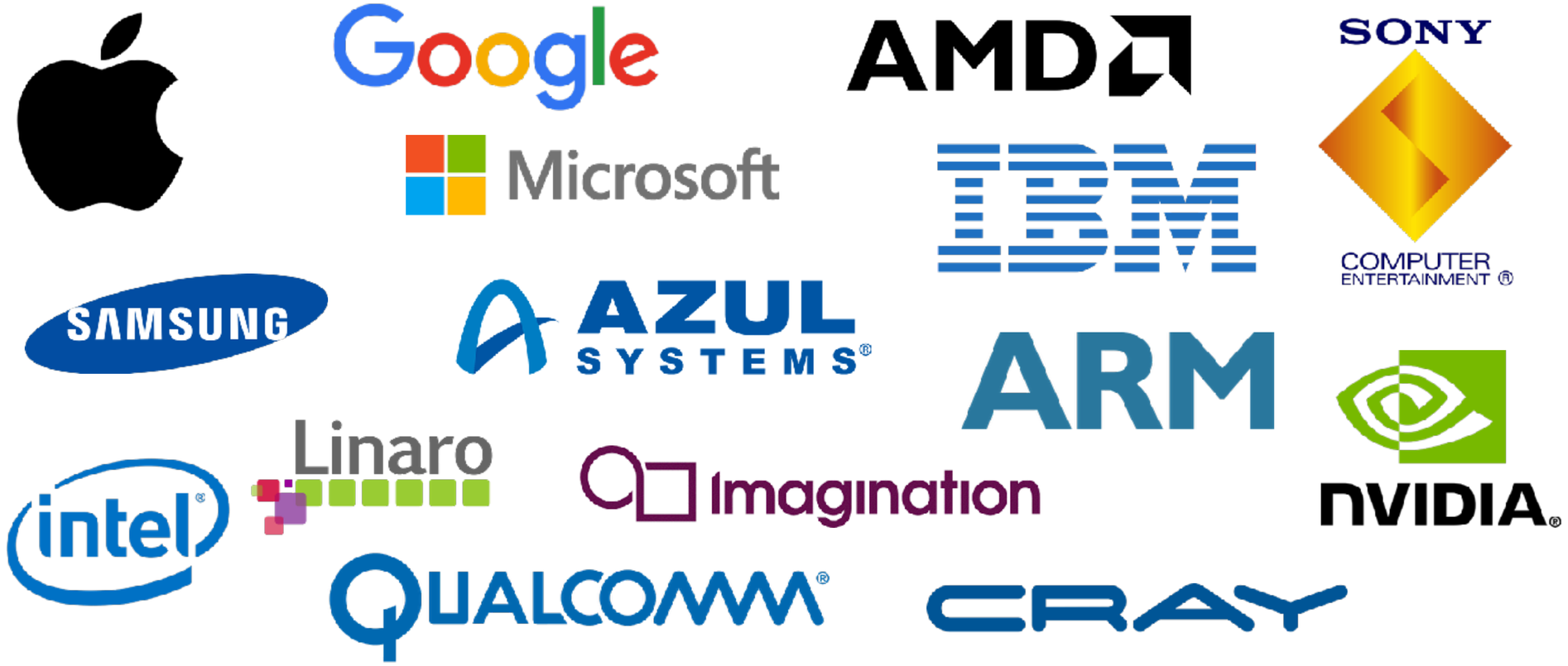
Где используется?

- C/C++/Objective C
- Swift
- Haskell
- Rust
- …

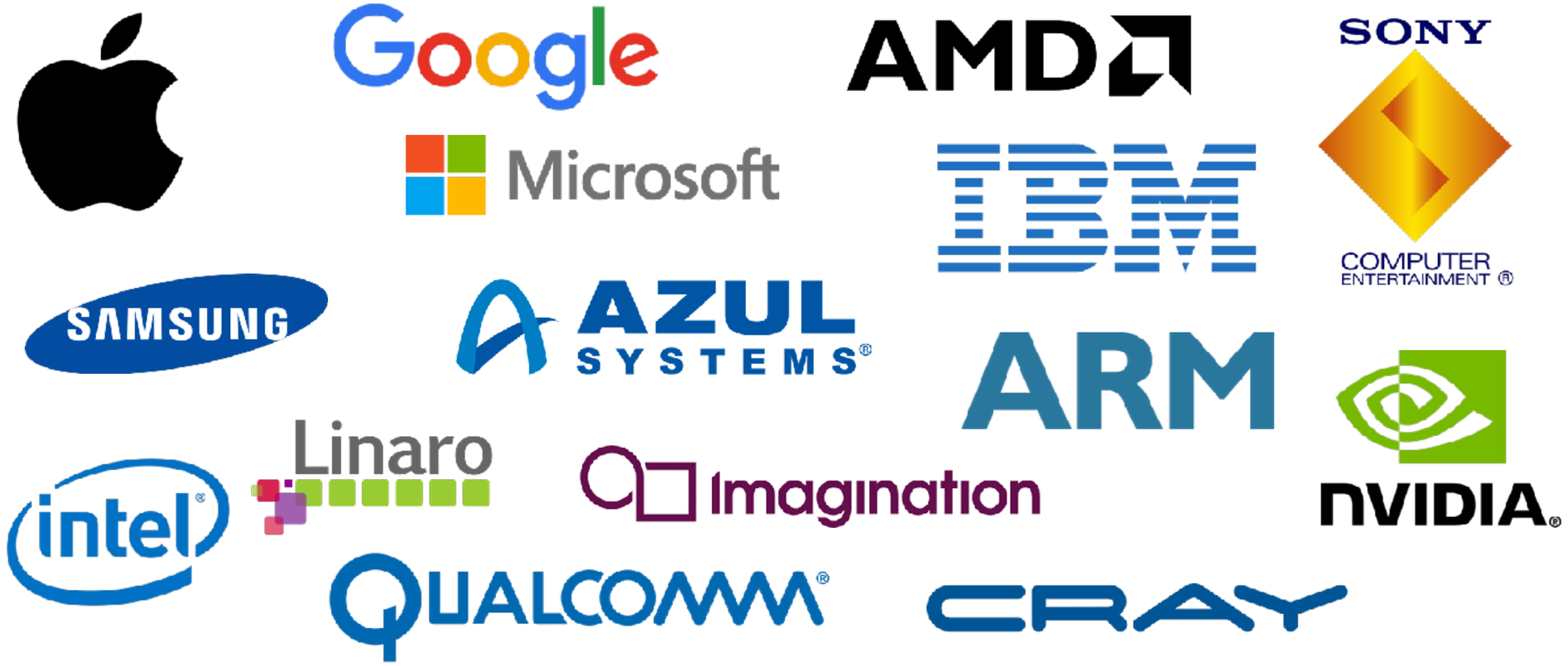






Кто разрабатывает LLVM?



Кто разрабатывает LLVM?



Более 500 разработчиков

AZUL SYSTEMS

LLVM

- Разрабатывается с 2000 года
- Зрелая и стабильная
- Обеспечивает хорошую производительность для C/C++ кода

Модульная архитектура

- LLVM IR — универсальное промежуточное представление
- Анализы, трансформации
- Backend-ы для различных архитектур
- Тулы

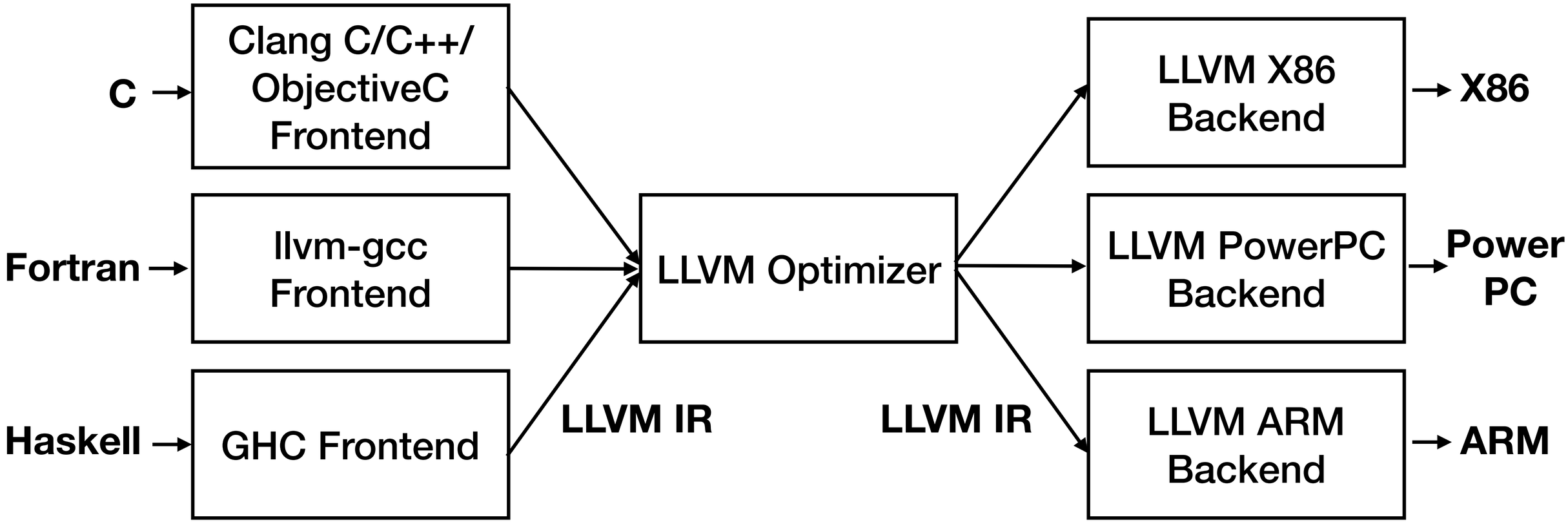
LLVM IR

Универсальный высокоуровневый ассемблер

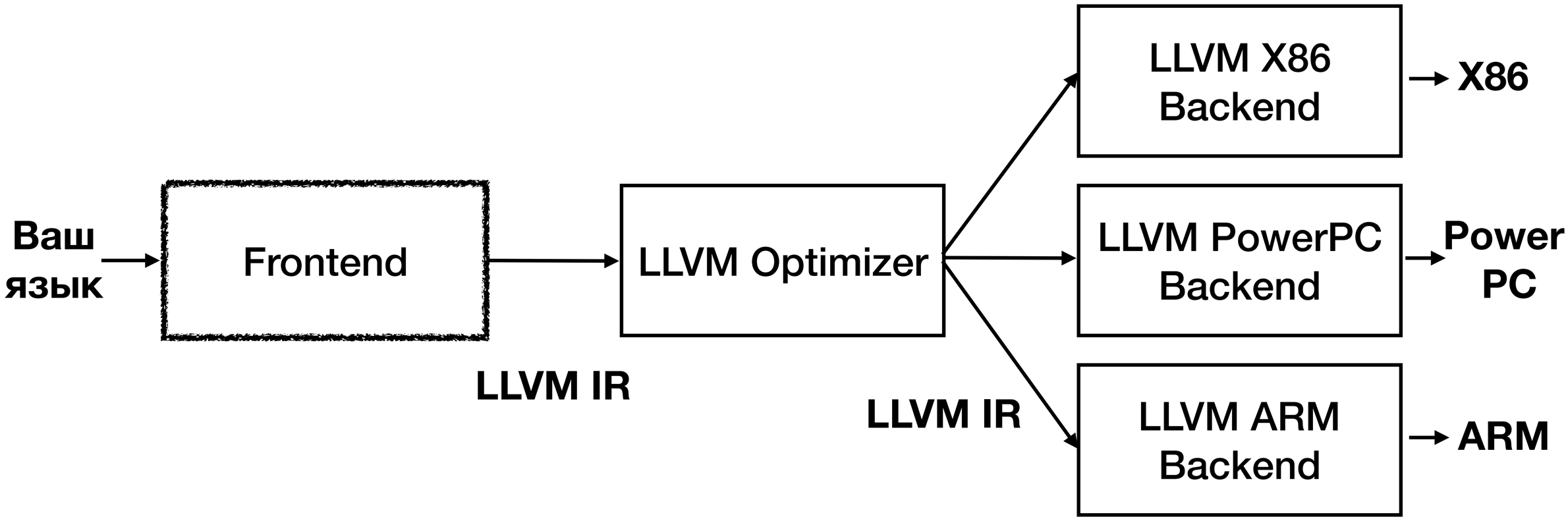
```
int mul_add(int x, int y, int z) {
  return x * y + z;
}

define i32 @mul_add(i32 %x, i32 %y, i32 %z) {
entry:
  %tmp = mul i32 %x, %y
  %tmp2 = add i32 %tmp, %z
  ret i32 %tmp2
}
```

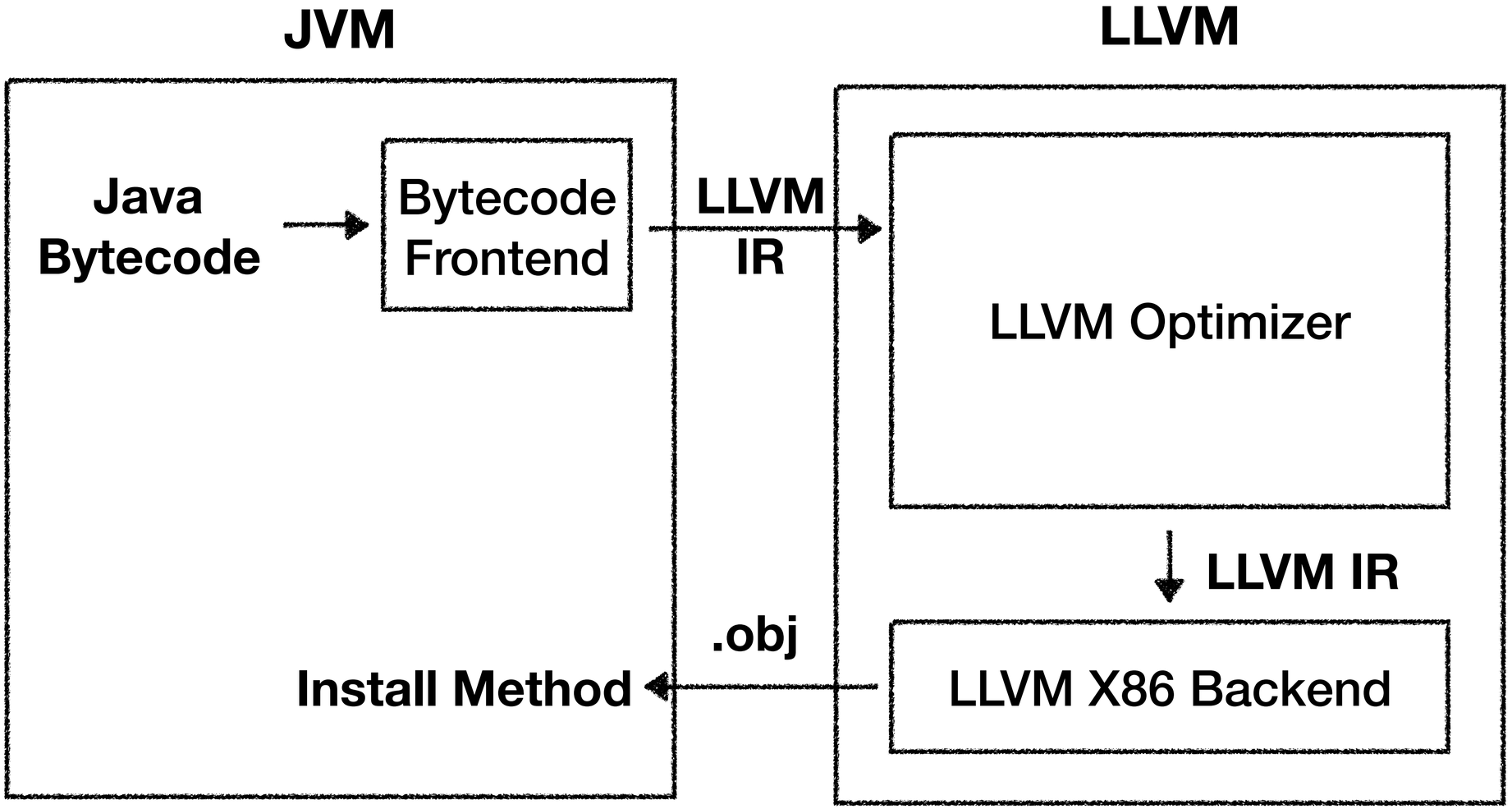
Как написать компилятор?



Как написать компилятор?



Как устроен Falcon?

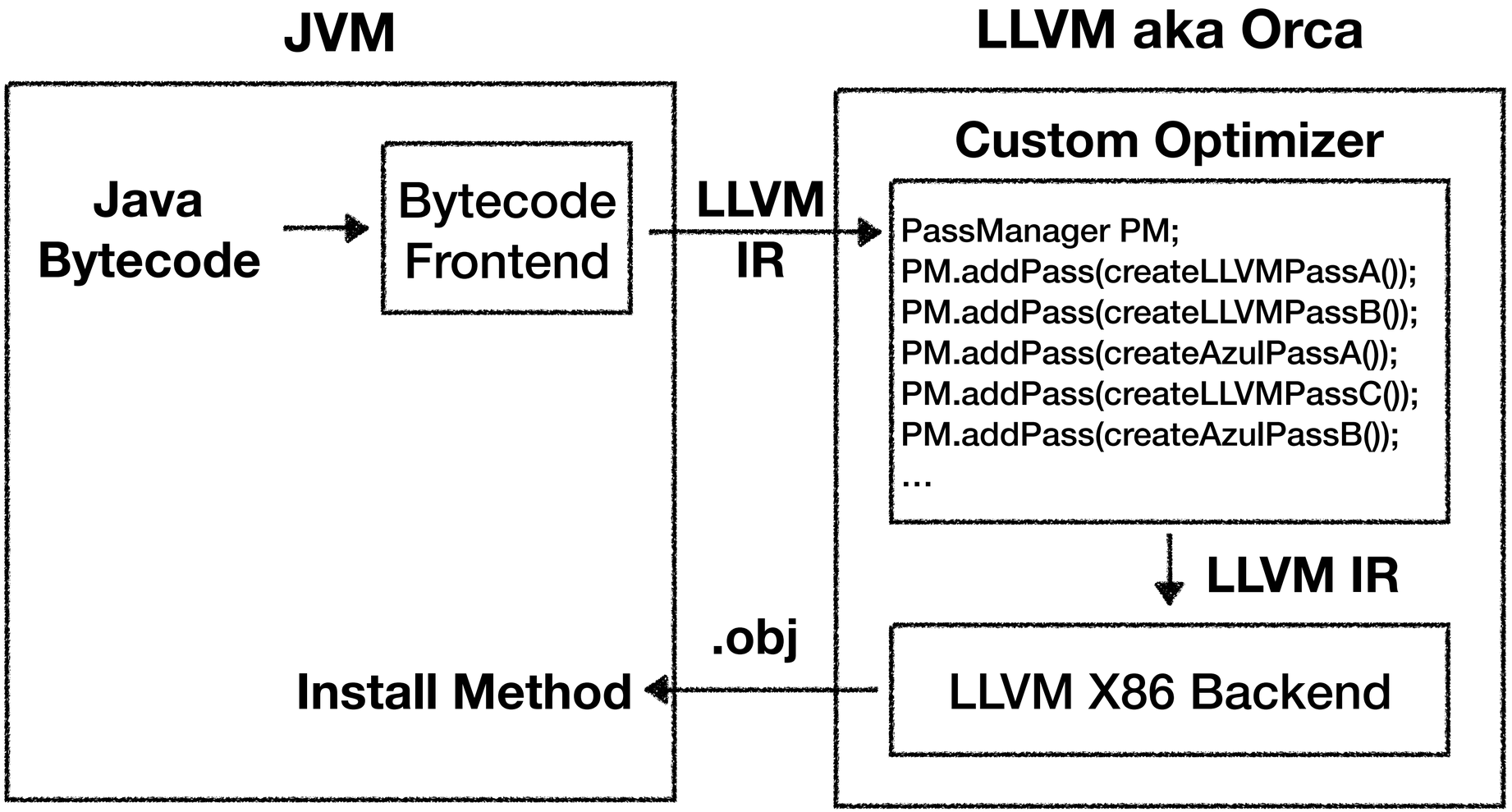


AZUL SYSTEMS®

Orca

- Наш форк LLVM
- Универсальный Java компилятор на базе LLVM
  - Может быть использован с любой другой VM
  - Или вовсе вне VM

Как устроен Falcon?



AZUL SYSTEMS®

VM Callbacks

```
static final Object X = new Integer(42);
bool foo() {
  return X instanceof Number;
}
```


AZUL SYSTEMS®

VM Callbacks

```
static final Object X = new Integer(42);
bool foo() {
  return X instanceof Number;
}
```

```
define i1 @foo() {
  %X = load i8*, i8** <Xaddress>
  %kid = call i32 @get_class(%X)
  %res = call i1 @is_subtype_of(<Number>,%kid)
  ret i1 %res
}
```


AZUL SYSTEMS®

VM Callbacks

```
static final Object X = new Integer(42);
bool foo() {
  return X instanceof Number;
}
```

VM, что лежит по этому адресу?

```
define i1 @foo() {
  %X = load i8*, i8** <Xaddress>
  %kid = call i32 @get_class(%X)
  %res = call i1 @is_subtype_of(<Number>,%kid)
  ret i1 %res
}
```

VM Callbacks

```
static final Object X = new Integer(42);
bool foo() {
  return X instanceof Number;
}
```

VM, а какой у этого объекта тип?

```
define i1 @foo() {
  %X = load i8*, i8** <Xaddress>
  %kid = call i32 @get_class(%X)
  %res = call i1 @is_subtype_of(<Number>,%kid)
  ret i1 %res
}
```


AZUL SYSTEMS®

VM Callbacks

```
static final Object X = new Integer(42);
bool foo() {
  return X instanceof Number;
}
```

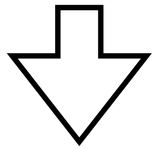
java.lang.Integer

```
define i1 @foo() {
  %X = load i8*, i8** <Xaddress>
  %kid = <Integer>
  %res = call i1 @is_subtype_of(<Number>,%kid)
  ret i1 %res
}
```


AZUL SYSTEMS®

VM Callbacks

```
static final Object X = new Integer(42);
bool foo() {
  return X instanceof Number;
}
```



```
define i1 @foo() {
  %res = call i1
    @is_subtype_of(<Number>, <Integer>)
  ret i1 %res
}
```

VM Callbacks

```
static final Object X = new Integer(42);
bool foo() {
  return X instanceof Number;
}
```

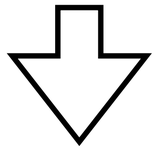
VM, а Integer это подтип Number?

```
define i1 @foo() {
  %res = call i1
    @is_subtype_of(<Number>, <Integer>)
  ret i1 %res
}
```


AZUL SYSTEMS®

VM Callbacks

```
static final Object X = new Integer(42);
bool foo() {
  return X instanceof Number;
}
```



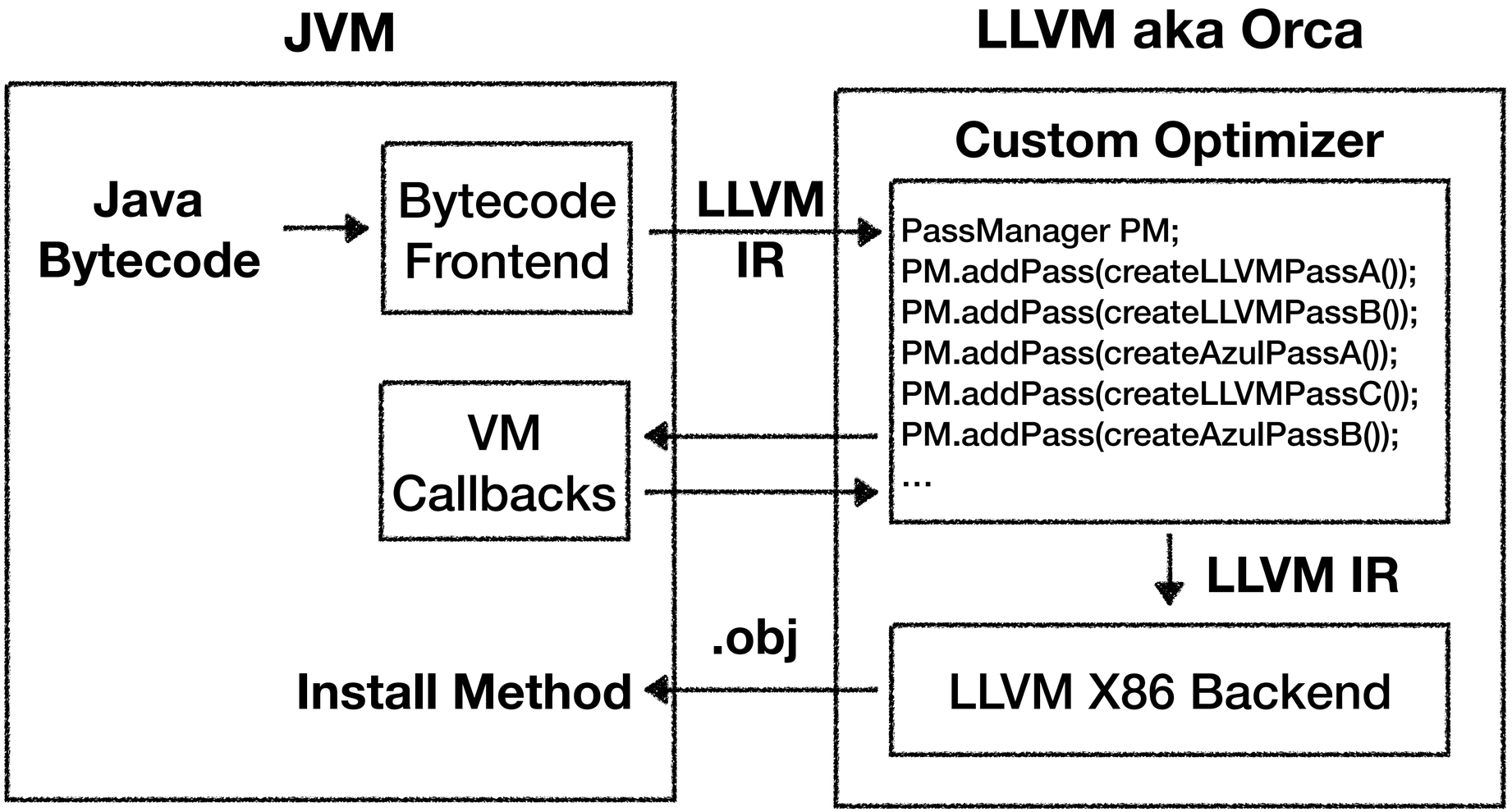
```
define i1 @foo() {
  ret i1 true
}
```


AZUL SYSTEMS®

VM Callbacks

- Является ли A подклассом B?
- Можно ли девиртуализовать вызов?
- Сгенерируй LLVM IR для такого-то метода
- …

Как устроен Falcon?



AZUL SYSTEMS®

Compilation Replay

Debuggability

- Отлаживать JIT сложно
  - Что если падает через 2 часа после запуска?
  - …и то не всегда

Compilation Replay

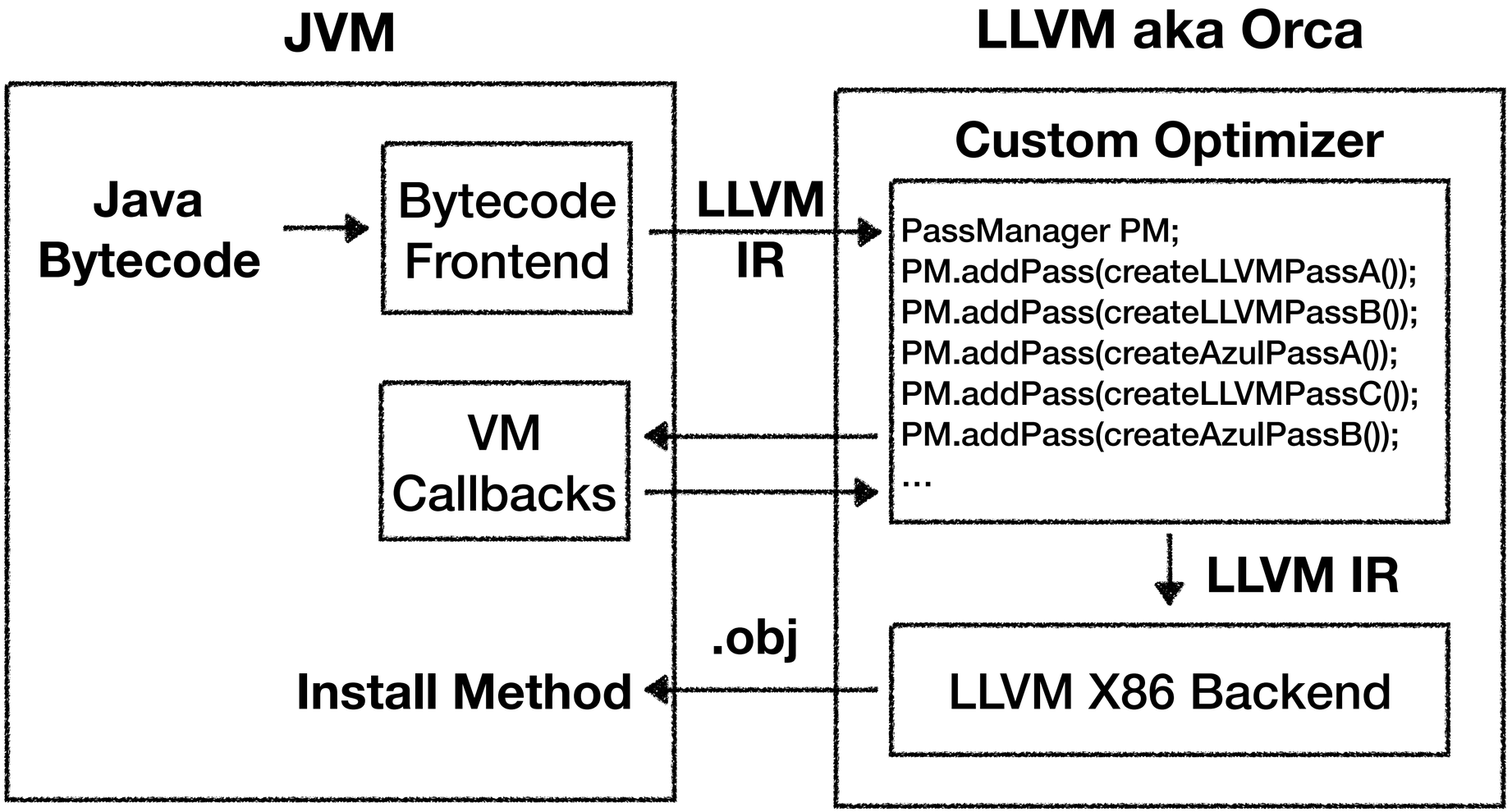
- Компилятор изолирован от VM
- Входные данные для компиляции определены
  - LLVM IR, VM Query Database

Compilation Replay

- Компилятор изолирован от VM
- Входные данные для компиляции определены
  - LLVM IR, VM Query Database
- Можем запустить компиляцию отдельно от VM

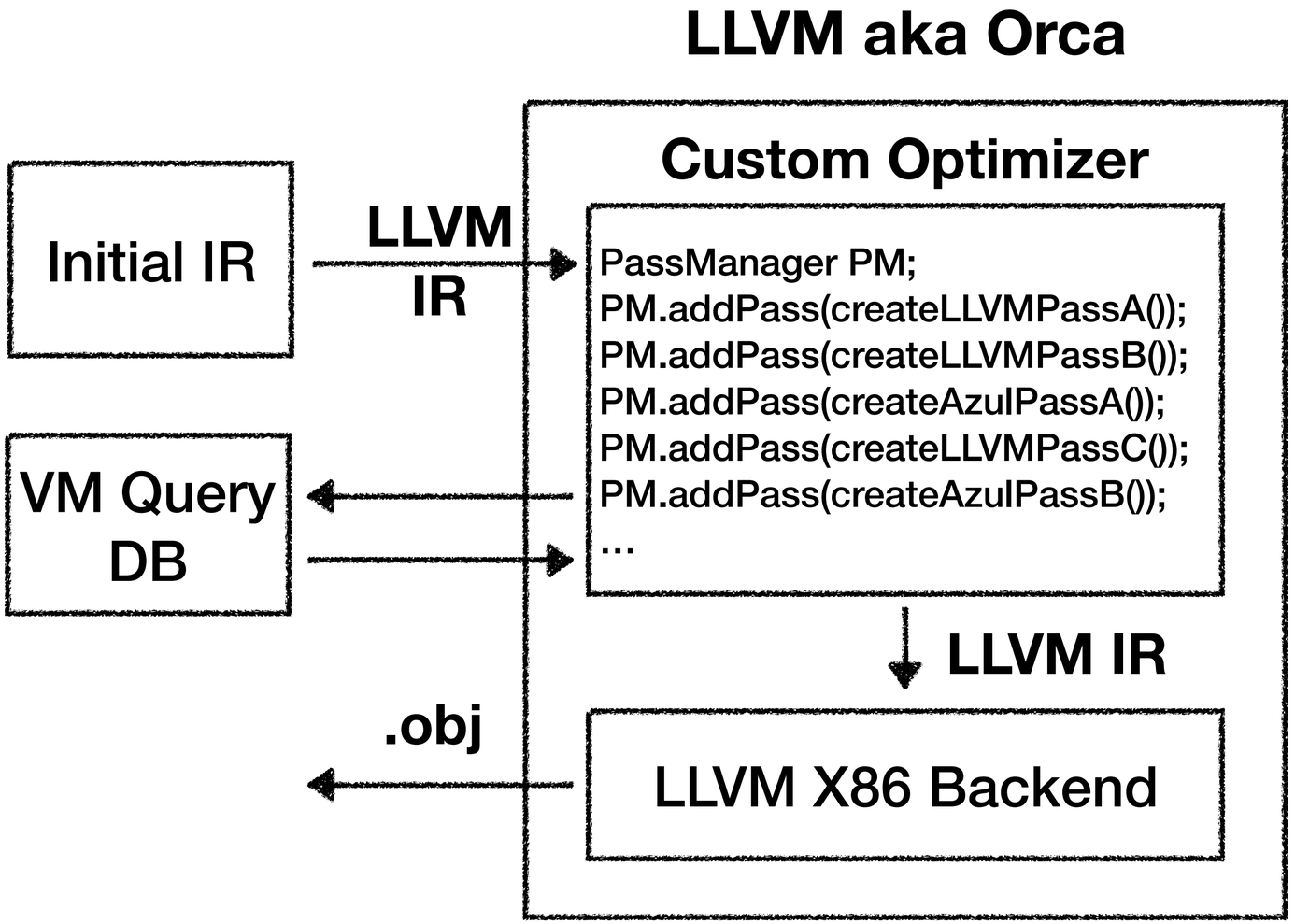
AZUL SYSTEMS®

Compilation Replay



AZUL SYSTEMS®

Compilation Replay



AZUL SYSTEMS®

Зачем нужен replay?

- Тесты
- Разработка новой функциональности
- Отладка крэшей

Как извлечь replay из VM?

- Сохраняем replay N последних компиляции в памяти
- В случае крэша легко извлекается из core dump
- Экономит часы дебага

LLVM Bugpoint

- Что если компилятор падает на очень большом методе?

LLVM Bugpoint

- Что если компилятор падает на очень большом методе?
- LLVM Bugpoint — тул для автоматического упрощения проблемных компиляций

Как сделать JIT компилятор для Java из компилятора для C++?

LLVM и managed языки

- LLVM используется в основном для статических компиляторов
  - Clang — основной пользователь

LLVM и managed языки

- LLVM используется в основном для статических компиляторов
  - Clang — основной пользователь
- Много экспериментальных проектов использующих в качестве JIT для managed языков
  - Shark — эксперимент с LLVM в Open JDK

AZUL SYSTEMS®

Java JIT TODO:

- Precise Garbage Collection
- Спекулятивные оптимизации и деоптимизации
- Java оптимизации

Java JIT TODO:

- Precise Garbage Collection
- Спекулятивные оптимизации и деоптимизации
- Java оптимизации

Garbage Collection

- Обходит граф объектов из корневых ссылок
  - Глобалы, локалы, временные значения на стеке исполнения
- Достижимые объекты — живые объекты

Garbage Collection

- Обходит граф объектов из корневых ссылок
  - Глобалы, локалы, временные значения на стеке исполнения
- Достижимые объекты — живые объекты
- Опционально перемещает живые объекты

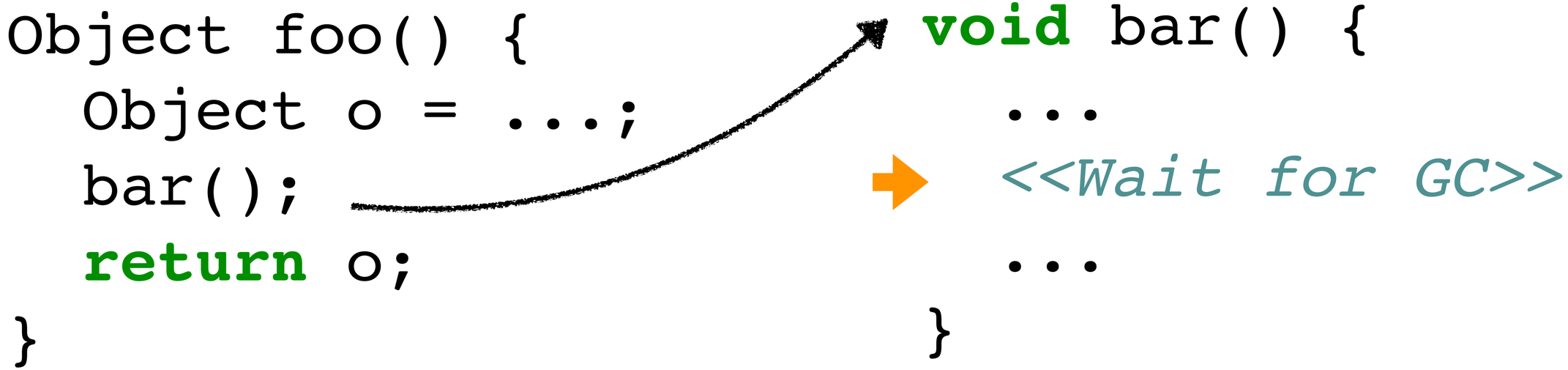
Взаимодействие с GC

- Как узнать где именно лежат ссылки на стеке исполнения?
  - Стек, регистры

Взаимодействие с GC

- Как узнать где именно лежат ссылки на стеке исполнения?
  - Стек, регистры
- GC safepoints — точки в которых GC может обойти стек
- GC может попросить thread остановиться на safepoint

Thread на Safepoint



GC должен иметь возможность обойти стек

GC Relocations

```
Object foo() {
  Object o = ...;
  bar();
  return o;
}
```

Компилятор должен знать, что о может быть перемещен

AZUL SYSTEMS®

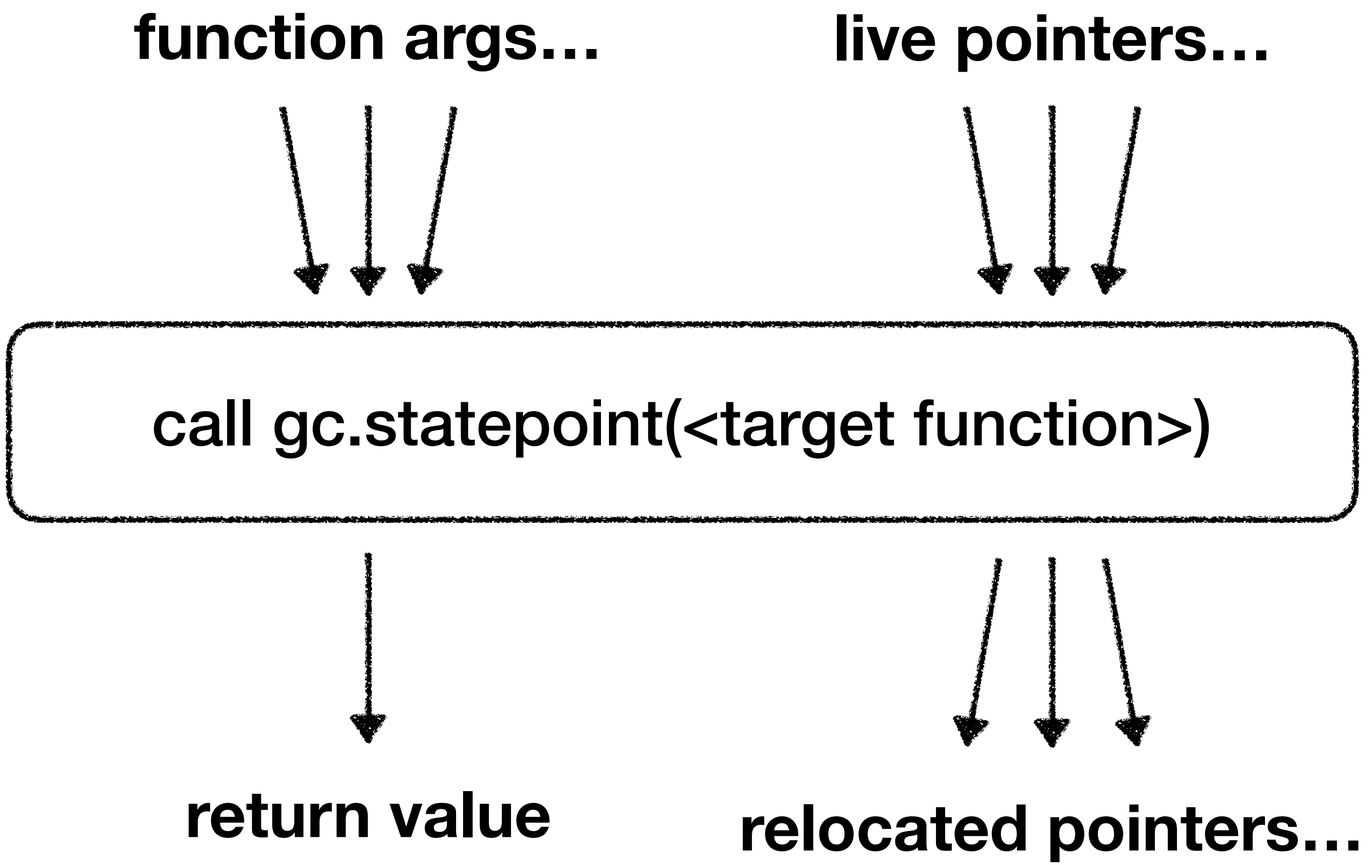
GC Relocations

```
Object foo() {
  Object o = ...;
  _o = safepoint(bar(), o);
  return _o;
}
```

Компилятор должен знать, что
o может быть перемещен

AZUL SYSTEMS®

Statepoint



AZUL SYSTEMS®

Statepoint Intrinsic

http://llvm.org/docs/Statepoints.html

```
declare token
  @llvm.experimental.gc.statepoint(func_type <target>,
                   i64 <#call args>, ... (call parameters),
                   i64 <#deopt args>, ... (deopt parameters),
                   ... (gc parameters))
```


AZUL SYSTEMS®

Statepoint Intrinsic

http://llvm.org/docs/Statepoints.html

```
declare token
  @llvm.experimental.gc.statepoint(func_type <target>,
                  i64 <#call args>, ... (call parameters),
                  i64 <#deopt args>, ... (deopt parameters),
                  ... (gc parameters))
```

Компилируется в вызов <target> + Stack Map

GC обходит стек используя Stack Map

Safepoints и оптимизации

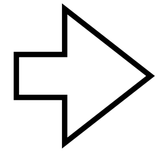
```
void foo(Object o) {
  if (o instanceof String)
    return;
  _o = safepoint(bar(), o);
  if (_o instanceof String)
    return;
  ...
}
```

Оптимизатор не знает, что такое _o

AZUL SYSTEMS®

Safepoints и оптимизации

```
void foo(Object o) {
  if (o instanceof String)
    return;
  bar();
  if (o instanceof String)
    return;
  ...
}
```



```
void foo(Object o) {
  if (o instanceof String)
    return;
  bar();
  ...
}
```

Не используем явное представление для оптимизаций

AZUL SYSTEMS®

GC Pointers

- LLVM может конвертировать pointer <=> int
- Битовое представление GC pointer меняется при релокации объектов

GC Pointers

```
if (obj instanceof String) {
   char[] p = ((String)obj).value;
   ...
} else if (obj instanceof Long) {
   long v = ((Long)obj).value;
   ...
}
```

GC Pointers

```
if (%obj instanceof String) {
   %p = load i8*, i8* %obj+12
   ...
} else if (%obj instanceof Long) {
   %v = load i64, i64 %obj+12
   ...
}
```


AZUL SYSTEMS®

GC Pointers

```
%v = load i64, i64 %obj+12
if (%obj instanceof String) {
   %p = inttoptr i64 %v to i8*
   ...
} else if (%obj instanceof Long) {
   ...
}
```

GC Pointers

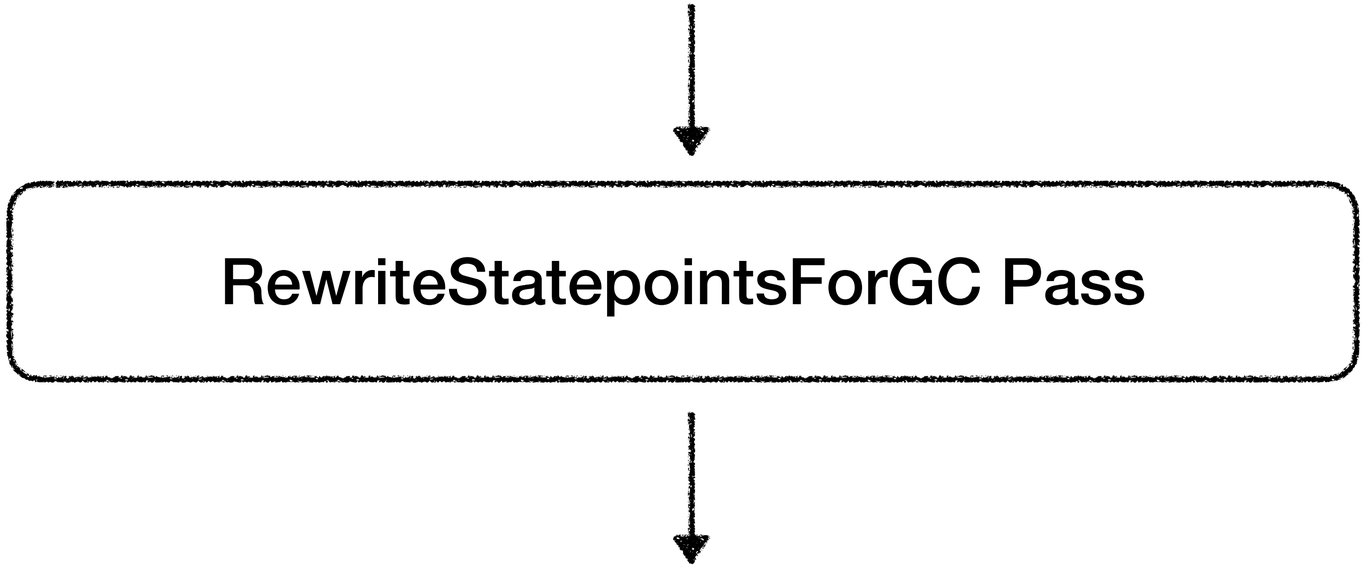
```
%v = load i64, i64 %obj+12
!!!Safepoint!!!
if (%obj instanceof String) {
   %p = inttoptr i64 %v to i8*
   ...
} else if (%obj instanceof Long) {
   ...
}
```

Non-integral Pointers

- Новый тип указателей — non-integral pointer (aka strong GC pointer)
- Битовое представление не определено
- Оптимизации не могут конвертировать non-integral pointer <=> int

RS4GC

**Неявное представление релокаций,
non-integral pointers**



**Явное представление релокаций,
integral pointers**

AZUL SYSTEMS®

Optimization Pipeline

AZUL SYSTEMS®

Java JIT TODO:

✓ Precise Garbage Collection

- Спекулятивные оптимизации и деоптимизации
- Java оптимизации

Java JIT TODO:

✓ Precise Garbage Collection

- Спекулятивные оптимизации и деоптимизации
- Java оптимизации

Uncommon Traps

Пример спекулятивной оптимизации

```
if (cond)
   foo();
else
   bar();
```

Uncommon Traps

Профилируем…

```
if (cond)
  foo();    // 0
else
  bar();    // 10000
```


AZUL SYSTEMS®

Uncommon Traps

Профилируем…

```
if (cond)
  foo();    // 0
else
  bar();    // 10000
```

foo никогда не исполнялся!

Uncommon Traps

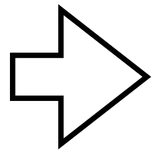
Не будем компилировать foo!

```
if (cond)
  foo();
else
  bar();
```

Uncommon Traps

Не будем компилировать foo!

```
if (cond)
  foo();
else
  bar();
```



```
if (cond)
  <uncommon_trap>

bar();
```


AZUL SYSTEMS®

Uncommon Traps

Если условие случится, исполнение продолжится в интерпретаторе

```
if (cond)
   <uncommon_trap>

bar();
```

Uncommon Traps

Как выразить uncommon trap в LLVM?

Uncommon Traps

Как выразить uncommon trap в LLVM?

```
declare type
  @llvm.experimental.deoptimize(...)
```

Деоптимизации

```
final int f = 5;
boolean foo() {
   int f_ = f;
   bar();
   return f == _f;
}
```

Деоптимизации

```
final int f = 5;
boolean foo() {
   int f_ = f;
   bar();
   return f == _f;
}
```

Предположим, что final поле не изменяется

Деоптимизации

```
final int f = 5;
boolean foo() {
   int f_ = f;
   bar();
   return f == _f;
}
```

Предположим, что final поле не изменяется

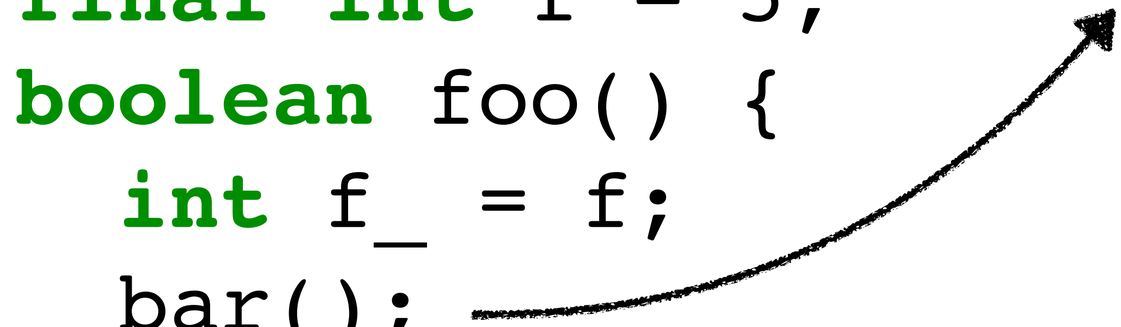
Деоптимизации

```
final int f = 5;
boolean foo() {
   int f_ = f;
   bar();
   return true;
}
```

Предположим, что final поле не изменяется

Деоптимизации

```
final int f = 5;
boolean foo() {
  int f_ = f;
  bar();
  return true;
}
```

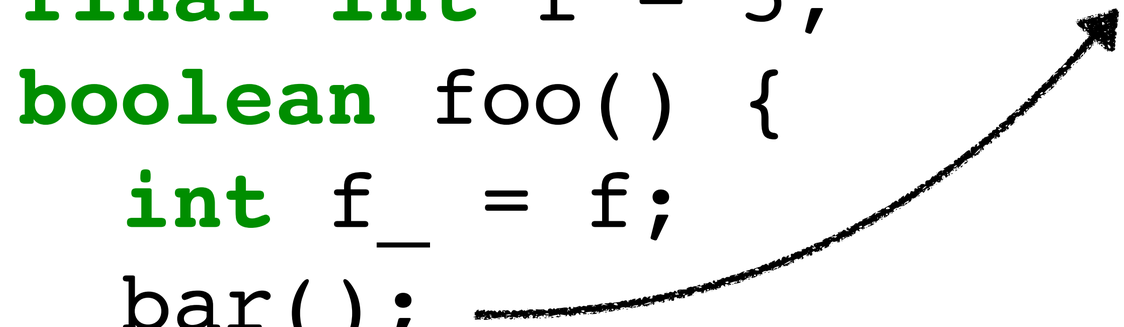


```
void bar() {
  // abuse field
  // finality!
  f = 1;
}
```

Куда возвращаться?

Деоптимизации

```
final int f = 5;
boolean foo() {
  int f_ = f;
  bar();
  return true;
}
```

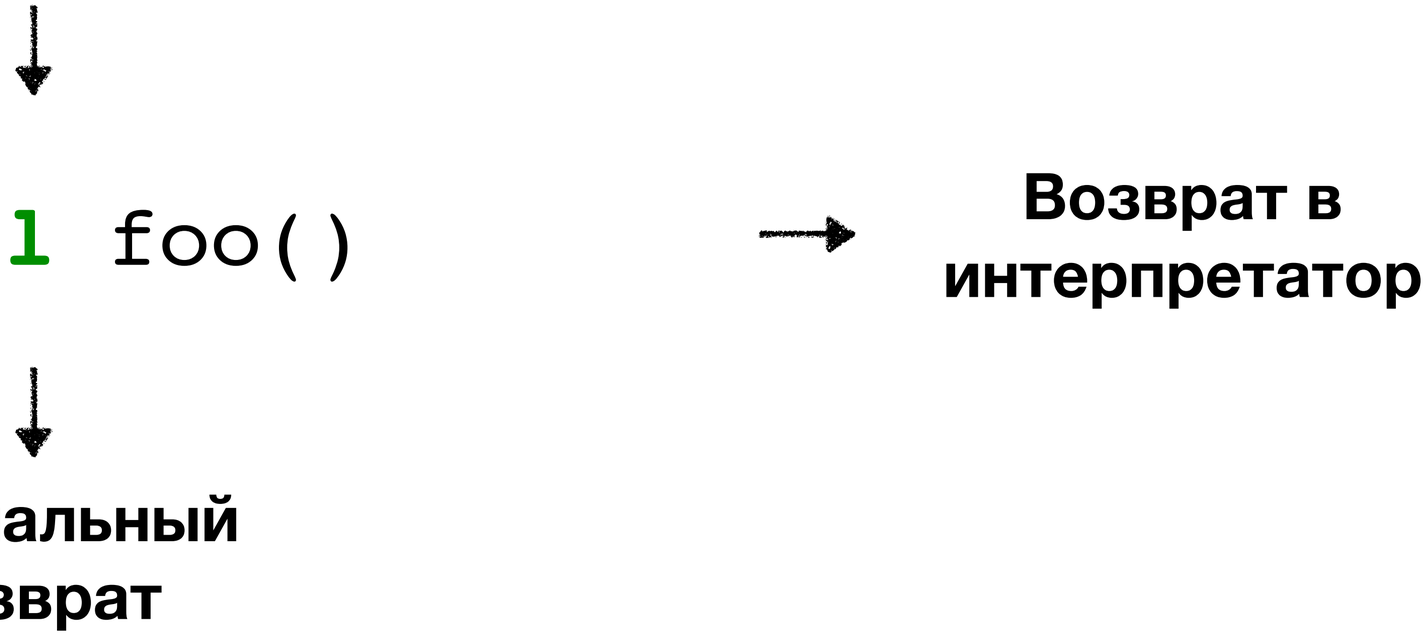


```
void bar() {
  // abuse field
  // finality!
  f = 1;
}
```

В интерпретатор!

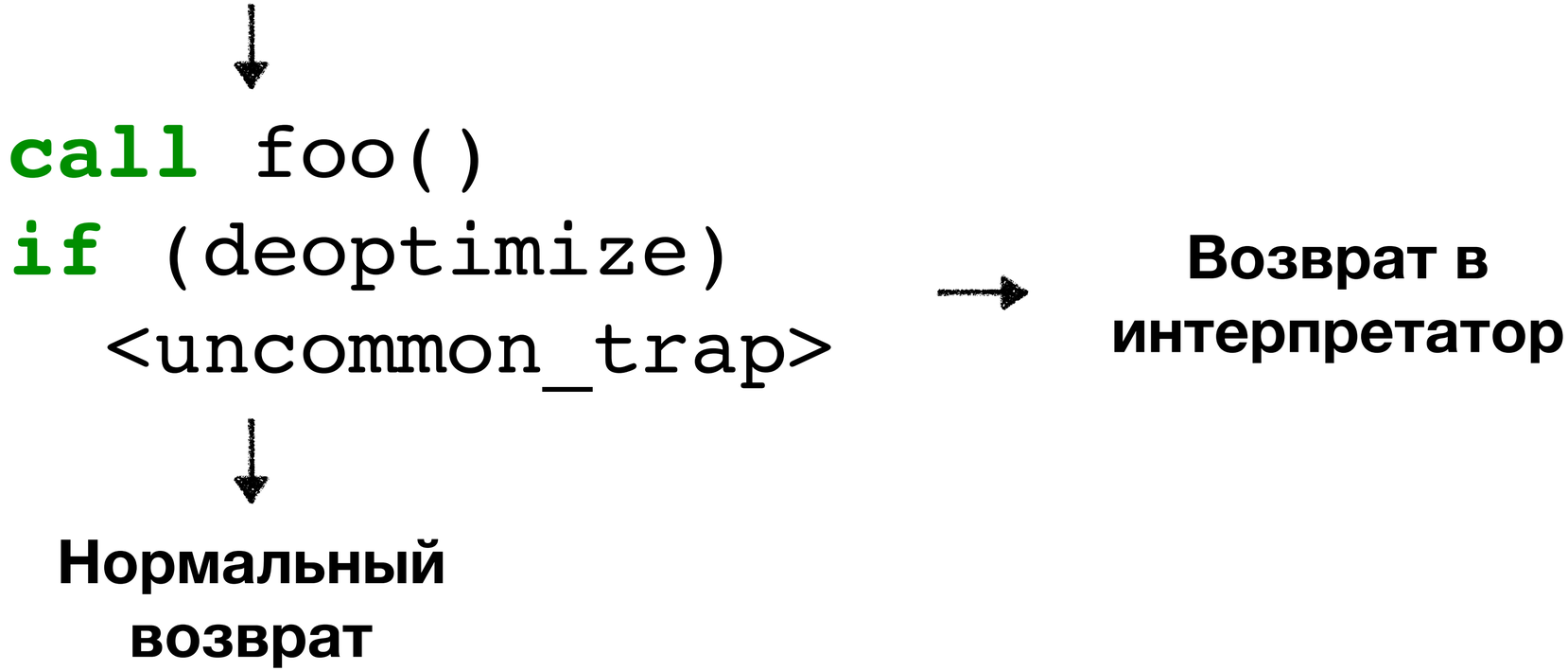
Возврат из вызова

У любого вызова есть два продолжения



Возврат из вызова

У любого вызова есть два продолжения



Interpreter State

```
boolean foo() {
  int f_ = f;
  bar();
  if (deoptimize)
    <uncommon_trap>
  return true;
}
```

```
 0: iconst_1
 1: istore_1
 2: aload_0
 3: invokevirtual bar()
 6: iconst_1
 7: iload_1
 8: if_icmpne 15
11: iconst_1
12: goto 16
15: iconst_0
16: ireturn
```

Interpreter State

```
boolean foo() {
  int f_ = f;
  bar();
  if (deoptimize)
    <uncommon_trap>
    [bci=6, local_1=f_]
  return true;
}
```

```
 0: iconst_1
 1: istore_1
 2: aload_0
 3: invokevirtual bar()
 6: iconst_1
 7: iload_1
 8: if_icmpne 15
11: iconst_1
12: goto 16
15: iconst_0
16: ireturn
```

Interpreter State

```
boolean foo() {
  int f_ = f;
  bar();
  if (deoptimize)
    <uncommon_trap>
    [bci=6, local_1=f_]
  return true;
}
```



```
 0: iconst_1
 1: istore_1
 2: aload_0
 3: invokevirtual bar()
 6: iconst_1
 7: iload_1
 8: if_icmpne 15
11: iconst_1
12: goto 16
15: iconst_0
16: ireturn
```

Interpreter State

```
boolean foo() {
  int f_ = f;
  bar();
  if (deoptimize)
    <uncommon_trap>
    [bci=6, local_1=f_]
  return true;
}
```

```
 0: iconst_1
 1: istore_1
 2: aload_0
 3: invokevirtual bar()
 6: iconst_1
 7: iload_1
 8: if_icmpne 15
11: iconst_1
12: goto 16
15: iconst_0
16: ireturn
```

Interpreter State

- Состояние интерпретатора:
  - BCI, локалы, стек, мониторы
- Runtime должен знать, где лежат соответствующие значения

Interpreter State

Как выразить в LLVM IR?

Interpreter State

Как выразить в LLVM IR?

```
call void @f(i32 %arg)
  [ "deopt"(i32 0, i8* %a, i32* null) ]
```


AZUL SYSTEMS®

Operand Bundles

http://llvm.org/docs/LangRef.html#operand-bundles

- Механизм для добавления произвольного набора значений к вызову
- Используем Deopt Bundle для описания interpreter state

Deopt Bundle Lowering

Как runtime узнает, где лежат значения?

```
call void @f(i32 %arg)
  [ "deopt"(i32 0, i8* %a, i32* null) ]
```

Deopt Bundle Lowering

```
call void @f(i32 %arg)
  [ "deopt"(i32 0, i8* %a, i32* null) ]
```

RewriteStatepointsForGC

```
call token
  @llvm.experimental.gc.statepoint(void @f(i32),
                   i64 1, i32 %arg,
                   i64 3, i32 0, i8* %a, i32* null,
                   ... (gc parameters))
```

Deopt Bundle Lowering

```
call token
  @llvm.experimental.gc.statepoint(void @f(i32),
                   i64 1, i32 %arg,
                   i64 3, i32 0, i8* %a, i32* null,
                   ... (gc parameters))
```

Генерируется Stack Map

Runtime использует Stack Map, чтобы извлечь значения

Java JIT TODO:

✓ Precise Garbage Collection

✓ Спекулятивные оптимизации и деоптимизации

• Java оптимизации

Java JIT TODO:

✓ Precise Garbage Collection

✓ Спекулятивные оптимизации и деоптимизации

- Java оптимизации

Java семантика

- Высокоуровневая семантика нужна для оптимизаций
  - Например, девиртуализация

Java семантика

- Высокоуровневая семантика нужна для оптимизаций
  - Например, девиртуализация
- LLVM IR — низкоуровневое представление
- LLVM ничего не знает про Java семантику

High Level Embedded IR

- Высокоуровневое представление поверх LLVM IR
- Атрибуты/метаданные для аннотации типов
- Абстракции — high level интринсики
  - Функции с семантикой известной оптимизатору

High Level Embedded IR
Пример

```
int foo(Object o) {
  if (o instanceof Integer)
    if (o instanceof Number)
      return o.hashCode();
  return 0;
}
```

High Level Embedded IR
Пример

```
int foo(i8* “java-type-class”=“Object” o) {
  o_class = @get_class(o);
  if (@is_subtype_of(Integer, o_class))
    if (@is_subtype_of(Number, o_class)) {
      target =
        @resolve_virtual(o, hashCode);
      return target(o);
    }
  return 0;
}
```


AZUL SYSTEMS®

High Level Embedded IR
Пример

Атрибуты

```
int foo(i8* “java-type-class”=“Object” o) {
  o_class = @get_class(o);
  if (@is_subtype_of(Integer, o_class))
    if (@is_subtype_of(Number, o_class)) {
      target =
        @resolve_virtual(o, hashCode);
      return target(o);
    }
  return 0;
}
```

High Level Embedded IR
Пример

Абстракции

```
int foo(i8* “java-type-class”=“Object” o) {
  o_class = @get_class(o);
  if (@is_subtype_of(Integer, o_class))
    if (@is_subtype_of(Number, o_class)) {
      target =
        @resolve_virtual(o, hashCode);
      return target(o);
    }
  return 0;
}
```

High Level Embedded IR Оптимизации

```
int foo(i8* “java-type-class”=“Object” o) {
  o_class = @get_class(o);
  if (@is_subtype_of(Integer, o_class))
    if (@is_subtype_of(Number, o_class)) {
      target =
        @resolve_virtual(o, hashCode);
      return target(o);
    }
  return 0;
}
```

o_class после проверки Integer!

AZUL SYSTEMS®

High Level Embedded IR Оптимизации

```
int foo(i8* “java-type-class”=“Object” o) {
  o_class = @get_class(o);
  if (@is_subtype_of(Integer, o_class))
    if (@is_subtype_of(Number, o_class)) {
      target =
        @resolve_virtual(o, hashCode);
      return target(o);
    }
  return 0;
}
```

Integer это подкласс Number

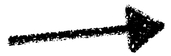
AZUL SYSTEMS®

High Level Embedded IR Оптимизации

```
int foo(i8* “java-type-class”=“Object” o) {
  o_class = @get_class(o);
  if (@is_subtype_of(Integer, o_class)) {
    target =
      @resolve_virtual(o, hashCode);
    return target(o);
  }
  return 0;
}
```

High Level Embedded IR Оптимизации

resolve_virtual для Integer Integer::hashCode



```
int foo(i8* “java-type-class”=“Object” o) {
  o_class = @get_class(o);
  if (@is_subtype_of(Integer, o_class)) {
    target =
      @resolve_virtual(o, hashCode);
    return target(o);
  }
  return 0;
}
```


AZUL SYSTEMS®

High Level Embedded IR Оптимизации

```
int foo(i8* “java-type-class”=“Object” o) {
  o_class = @get_class(o);
  if (@is_subtype_of(Integer, o_class))
    return Integer::hashCode(o);
  return 0;
}
```

Abstraction Lowering

- VM предоставляет реализации абстракций
- Реализации инлайнятся после высокоуровневых оптимизаций
- После этого возможны более низкоуровневые оптимизации

Java JIT TODO:

✓ Precise Garbage Collection

✓ Спекулятивные оптимизации и деоптимизации

✓ Java оптимизации

LLVM Upstream

LLVM Upstream

- Большая часть наших изменений в upstream
  - Инфраструктура для GC
  - Инфраструктура для спекулятивных оптимизаций
  - Улучшения оптимизаций

LLVM Upstream

- Работая с LLVM upstream мы получаем
  - Design/Code Review
  - Дополнительное тестирование
- Наша инфраструктура используется в других компиляторах

LLVM Upstream

- Регулярный merge из upstream
- Используем ToT upstream
- Регулярно получаем багфиксы, новые оптимизации, новые CPU архитектуры

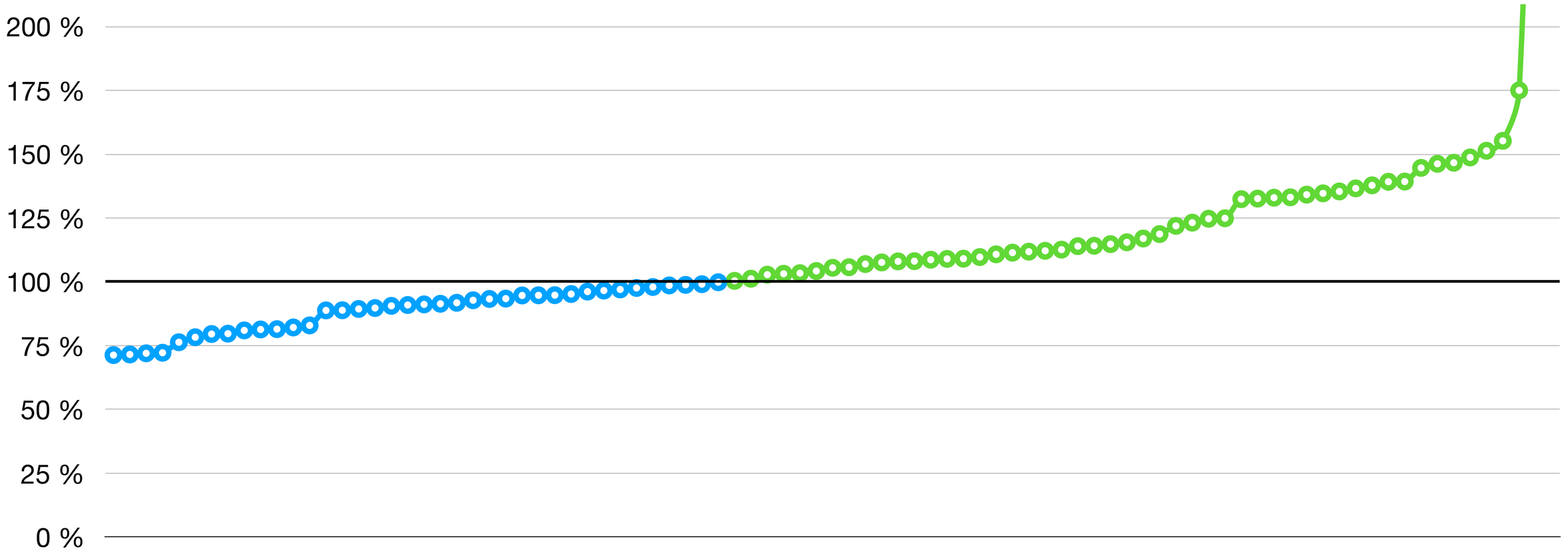
Пример

```
public int intsMin() {
   int min = Integer.MAX_VALUE;
   for (int i = 0; i < a_I.length; i++) {
     min = min > a_I[i] ? a_I[i] : min;
   }
   return min;
}
```

За счет векторизации в 4 раза быстрее, чем OpenJDK

Falcon vs C2

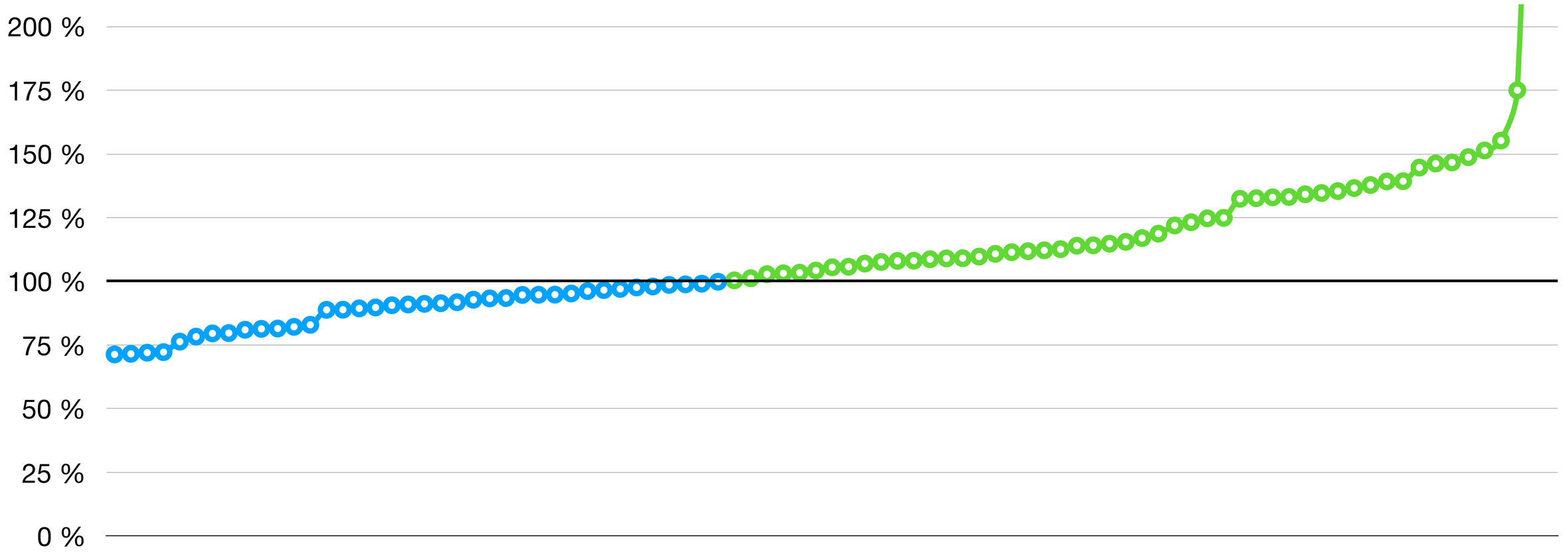
Zing 17.08 vs Oracle 8u121



SPECjvm + Dacapo + application benchmarks
skylake + haswell

AZUL SYSTEMS®

Zing 17.08 vs Oracle 8u121



119% geomean, 57% win rate

AZUL SYSTEMS®

Falcon vs C2
Стабильность

- Мутационное тестирование, на 100 000 тестов
  - Falcon — 1-2 падение
  - OpenJDK C2 — ~10 падений

Заключение

- C2 сложно развивать и поддерживать
- Решили построить новый компилятор на базе LLVM
- Проделали много инфраструктурной работы в LLVM upstream

Заключение
Что мы получили?

- Стабильность
- Производительность
- Расширяемость
- Удобство отладки

Вопросы?

Как попробовать?

http://www.azul.com/zingtrial/