AOT компиляция Java

Никита Липский
Excelsior

Давным давно

Давным давно,
Во времена С

Давным давно,

Во времена C,

Все компиляторы

были статические

Пока не появилась..

Пока не появилась

Java

EXCELSIOR

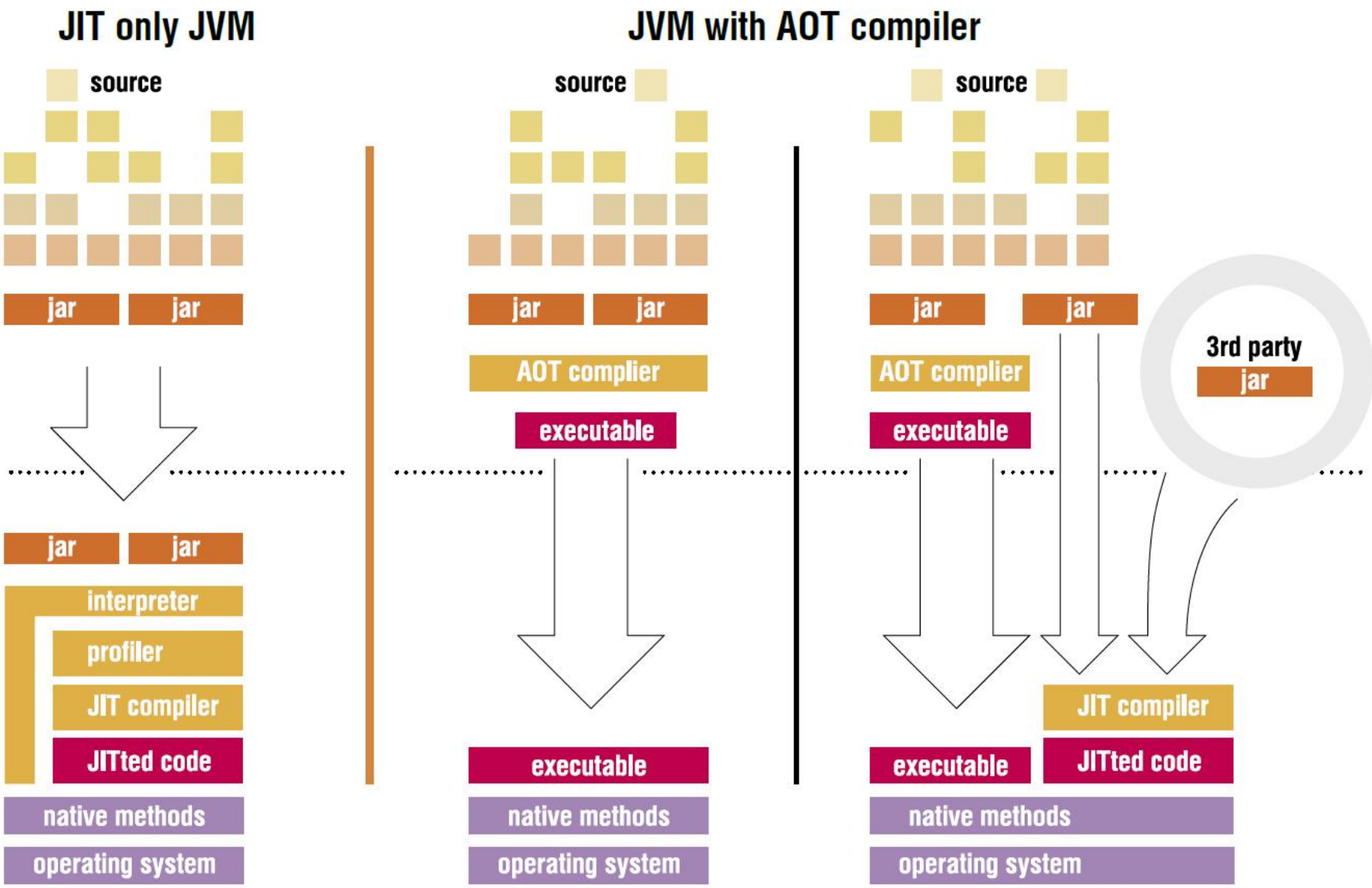
Исполнение Java байткода JVM

- Интерпретация
- Компиляция в машинный код и исполнение на “железе”

EXCELSIOR

Трансляция Java to Native

- Динамическая (Just-In-Time – **JIT**).
  - Трансляция происходит во время исполнения программы
- Статическая (Ahead-Of-Time – **AOT**)
  - Трансляция происходит до исполнения программы


JIT only JVM
source
jar
jar
jar
jar
interpreter
profiler
JIT compiler
JITted code
native methods
operating system
JVM with AOT compiler
source
jar
jar
AOT complier
executable
executable
native methods
operating system
source
jar
jar
AOT complier
executable
3rd party
jar
JIT compiler
JITted code
executable
native methods
operating system

EXCELSIOR

Статическая компиляция Java

- Это вообще возможно?
  - Если да, то при каких условиях и будет ли это все еще называться Java?
- Зачем это нужно?
  - Обфускация vs. статическая компиляция
  - Старт приложения
  - Уменьшение размера дистрибутива Java приложения без зависимостей от Java
  - Производительность: какие преимущества перед JIT компиляцией
  - Почему AOT лучше для Desktop, Embedded, Mobile
  - Есть ли выгода от применения AOT на server side

Java COMPATIBLE

Excelsior JET 10

Enterprise Edition

Accelerate, protect, and deploy your Java™ applications.

Кому нужен AOT для Java

EXCELSIOR

Мифы вокруг статической компиляции Java

EXCELSIOR

Миф 1.
Java - «слишком» динамическая

- Reflection

- Динамическая загрузка

Статическая компиляция невозможна?

EXCELSIOR

Миф 2. AOT компиляция – убийца WORA

WORA: Write Once Run Anywhere

(пиши раз, исполняй везде)

!=

BORA: Build Once?

(собирай раз??? …)

EXCELSIOR

Миф 3. AOT = маленький Exe

”Я получу (маленький) исполняемый файл, который будет работать без JVM (как в C)”

Но:

– Стандартные классы?

– GC, reflection?

EXCELSIOR

Миф 3. AOT = маленький Exe

Тем не менее, используя AOT, можно получить дистрибутив Java приложения без зависимостей от Java значительно меньший, чем размер JRE:

“Java худеет”:

http://www.youtube.com/watch?v=CKWlp8UQQb4

EXCELSIOR

Java «слишком» динамическая

EXCELSIOR

Нестандартные загрузчики классов

- Переопределяют умолчательную логику разрешений ссылок между классами
- Уникальное пространство имен
- Позволяют управлять зависимостями, решая проблемы JAR hell (OSGi, Java Module System)
- Java EE сервера, Eclipse RCP, плагины

EXCELSIOR

Нестандартные загрузчики классов

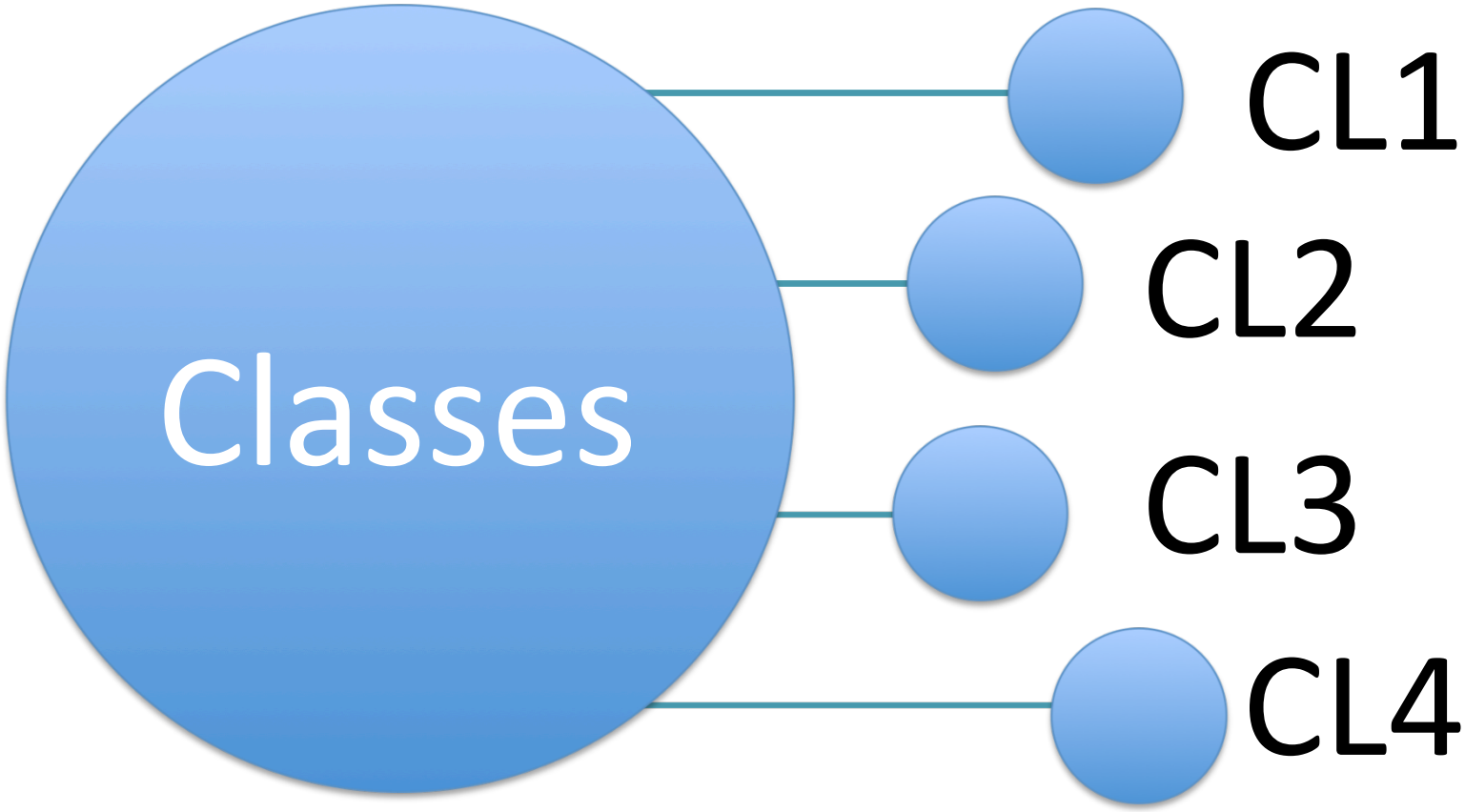
Как компилировать статически?

- Компилировать каждый класс изолировано от других
  - Плохо для производительности

- Изучить логику разрешения ссылок для популярных загрузчиков
  - Не работает для произвольных загрузчиков

Нестандартные загрузчики классов

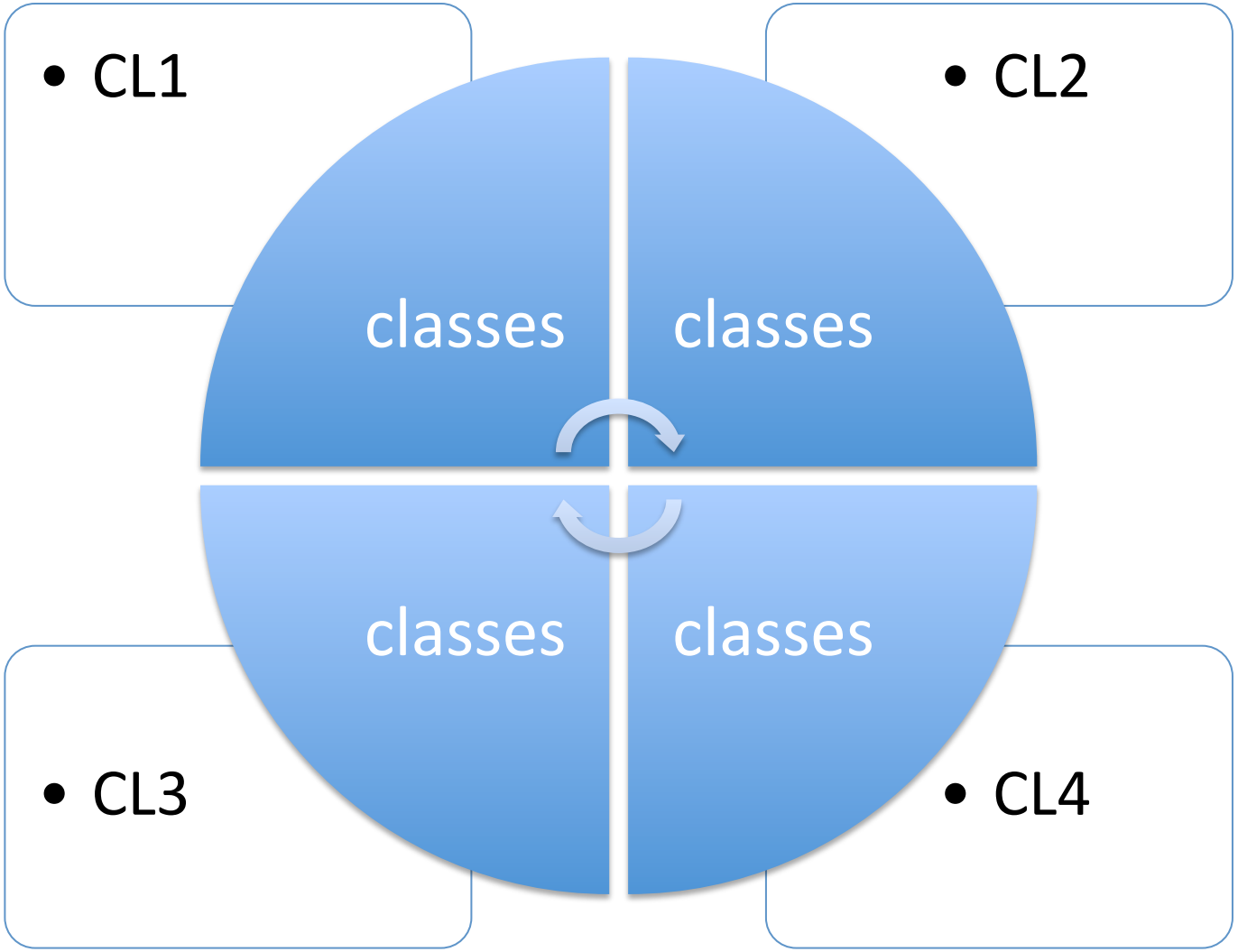
Нестандартные загрузчики классов



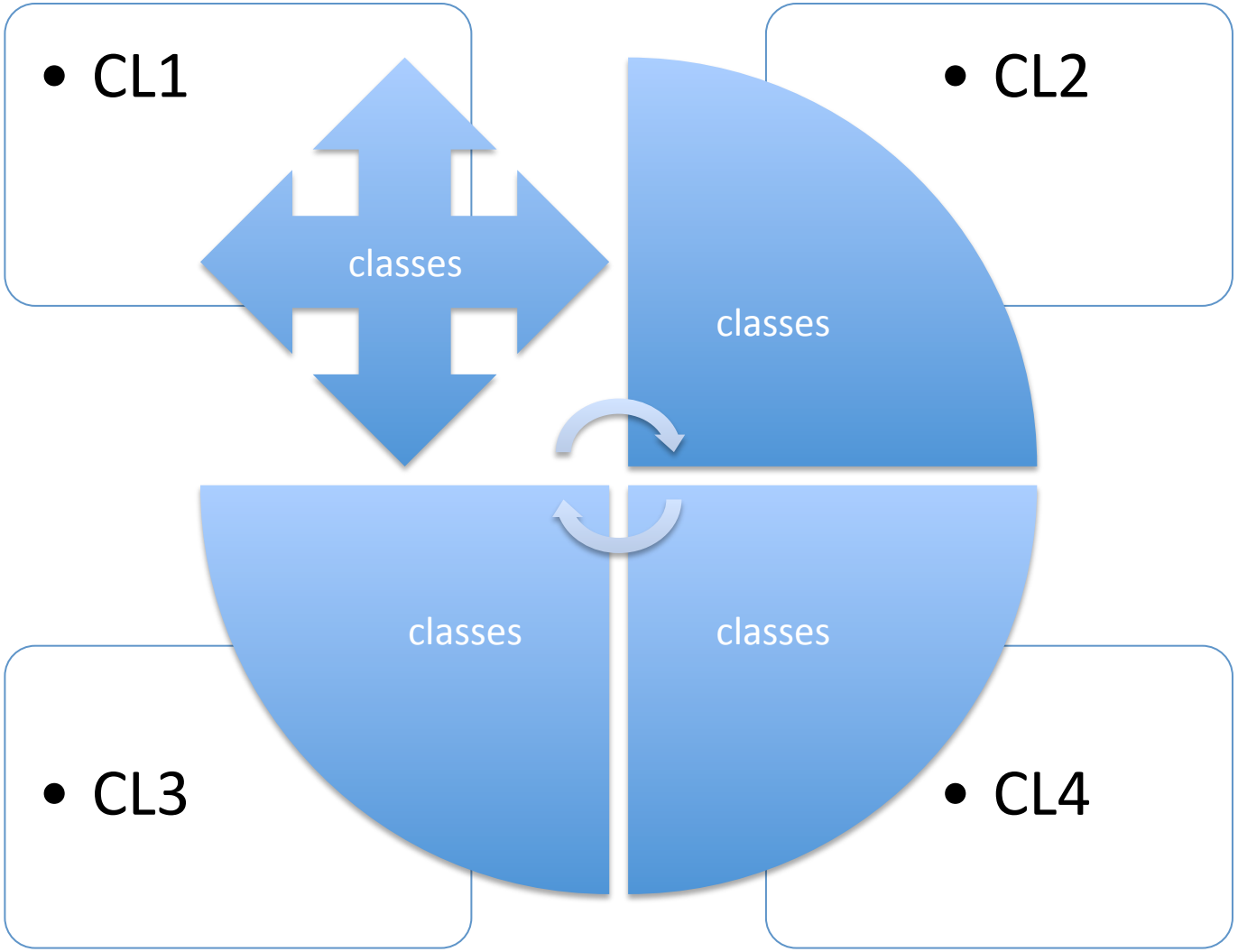
EXCELSIOR

CLi: classloader (загрузчик классов)

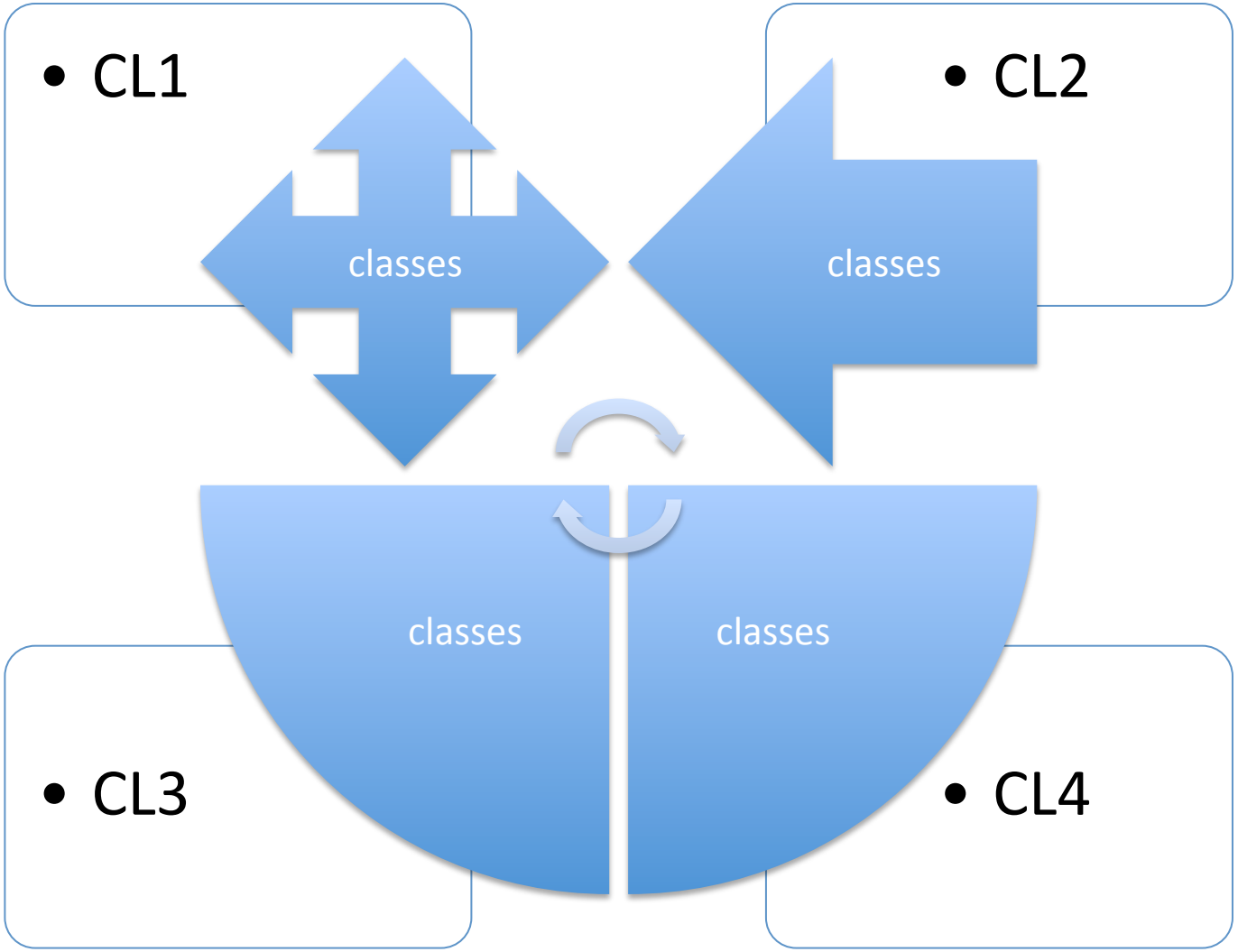
Нестандартные загрузчики классов



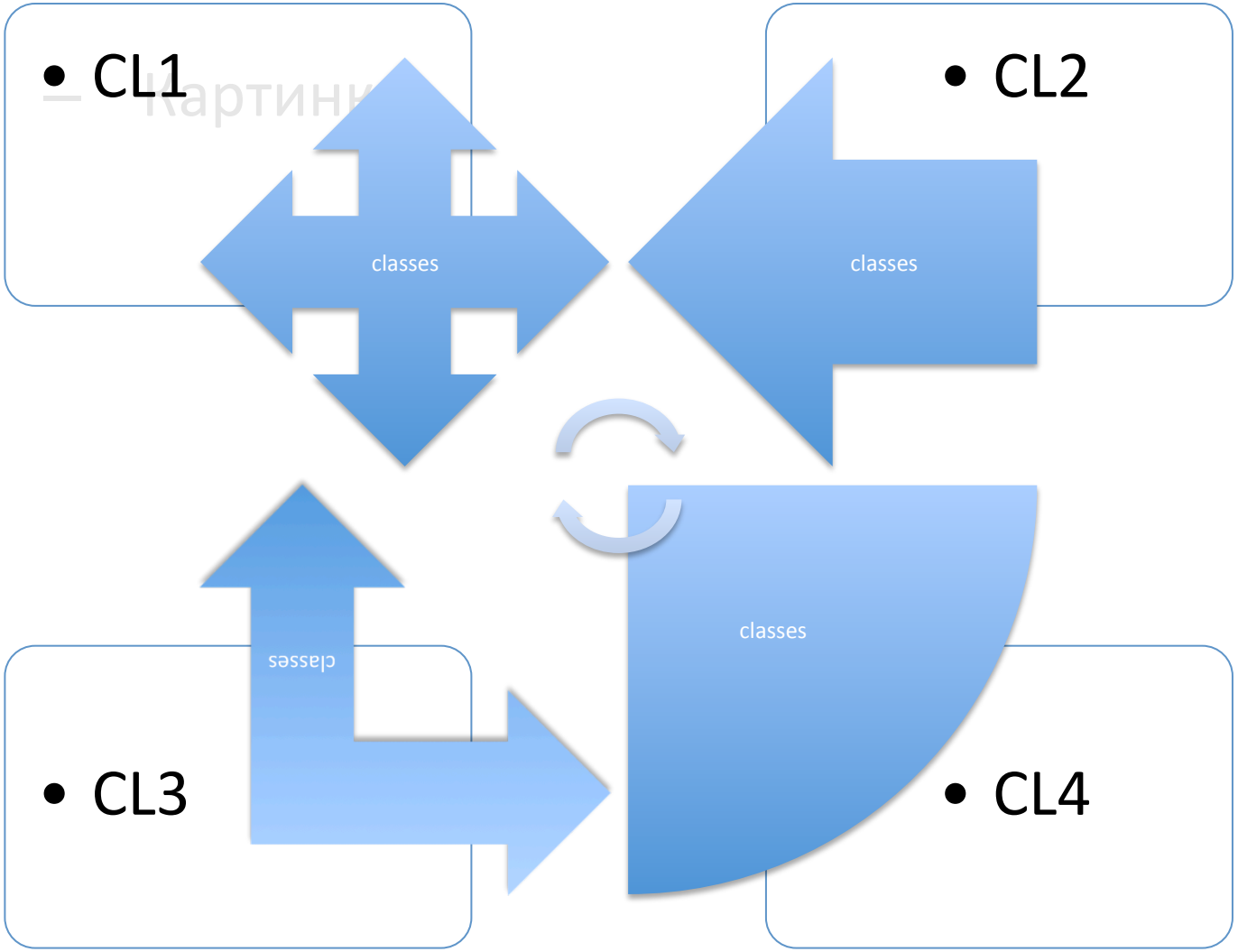
Нестандартные загрузчики классов



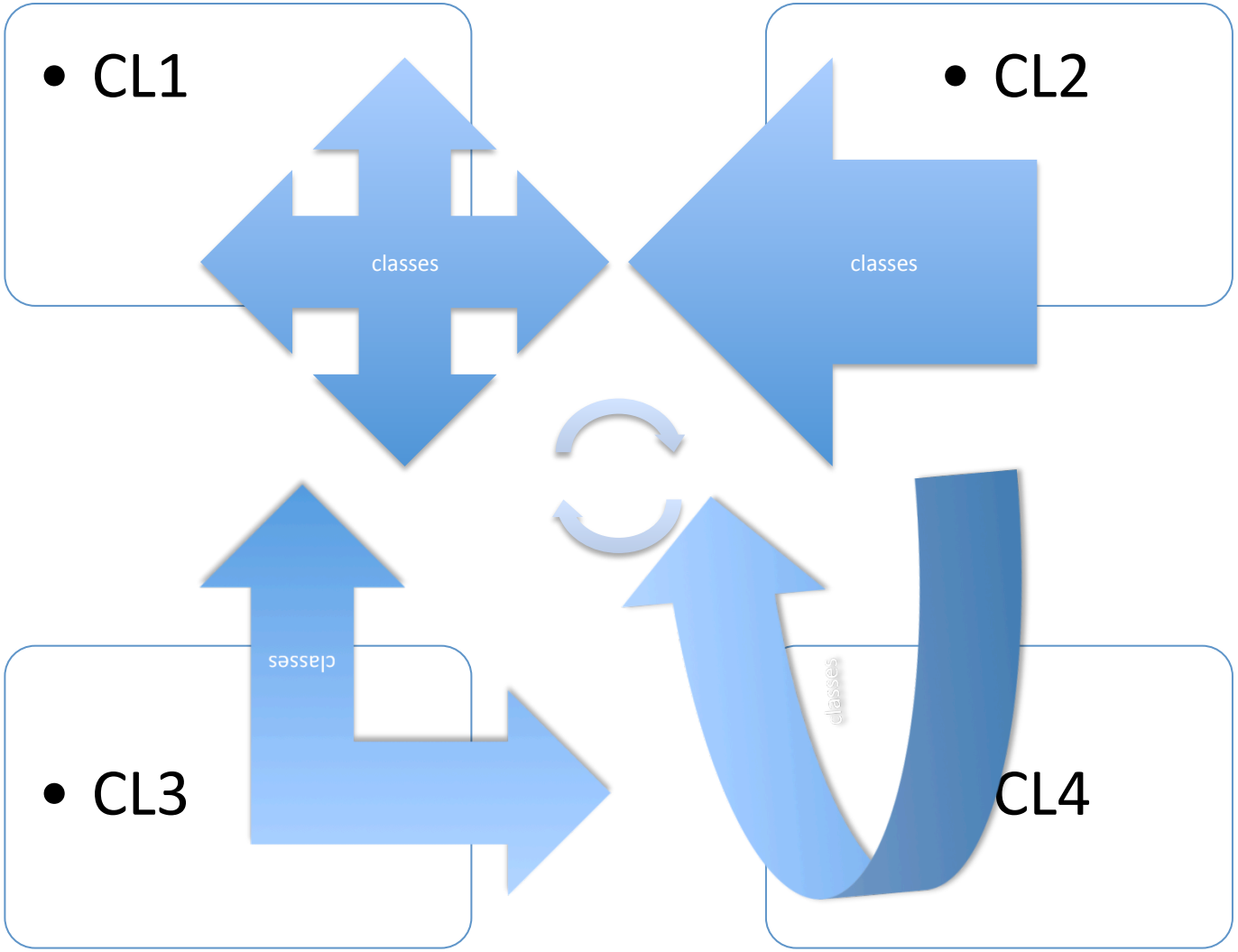
Нестандартные загрузчики классов



Нестандартные загрузчики классов



Нестандартные загрузчики классов



Нестандартные загрузчики классов

Схема поддержки загрузчиков в AOT:

- Компонента разрешения ссылок в AOT компиляторе:
  - Определяет какие загрузчики будут созданы во время исполнения и назначается каждому статический ID
  - Разбивает классы приложения по загрузчикам
  - Разрешает ссылки между классами

EXCELSIOR

Нестандартные загрузчики классов

Схема поддержки загрузчиков в AOT:

- Во время исполнения:
  - По экземпляру загрузчика вычисляется ID
  - Имея статический ID мы можем грузить статически скомпилированные классы:
    - создание экземпляра класса java.lang.Class
    - наполнение его рефлективной информацией
    - загрузка кода осуществляется OS

EXCELSIOR

Зачем AOT для Java?

EXCELSIOR

Защита кода от декомпиляции

EXCELSIOR

Защита кода приложения

- Java bytecode легко превращается в исходный код
- Процесс обфускации при активном использовании reflection – трудоемок и трудно поддерживаем
- Даже по обфусцированному коду часто можно понять, что код делает (ссылки на JDK остаются в неизменном виде)

Защита кода приложения

- Машинный код можно только эффективно дизассемблировать
- По дизассемблированному коду не понять структуру приложения (нет имен классов, методов)
- После агрессивной оптимизации машинный код далек от оригинального кода

Время старта приложения

EXCELSIOR

Холодный старт vs теплый

Во второй раз приложение стартует значительно быстрее, чем в первый

- Загрузка кода и данных приложения с диска
- Загрузка системных и сторонних динамических библиотек (dll, so)

EXCELSIOR

Java Quick Start

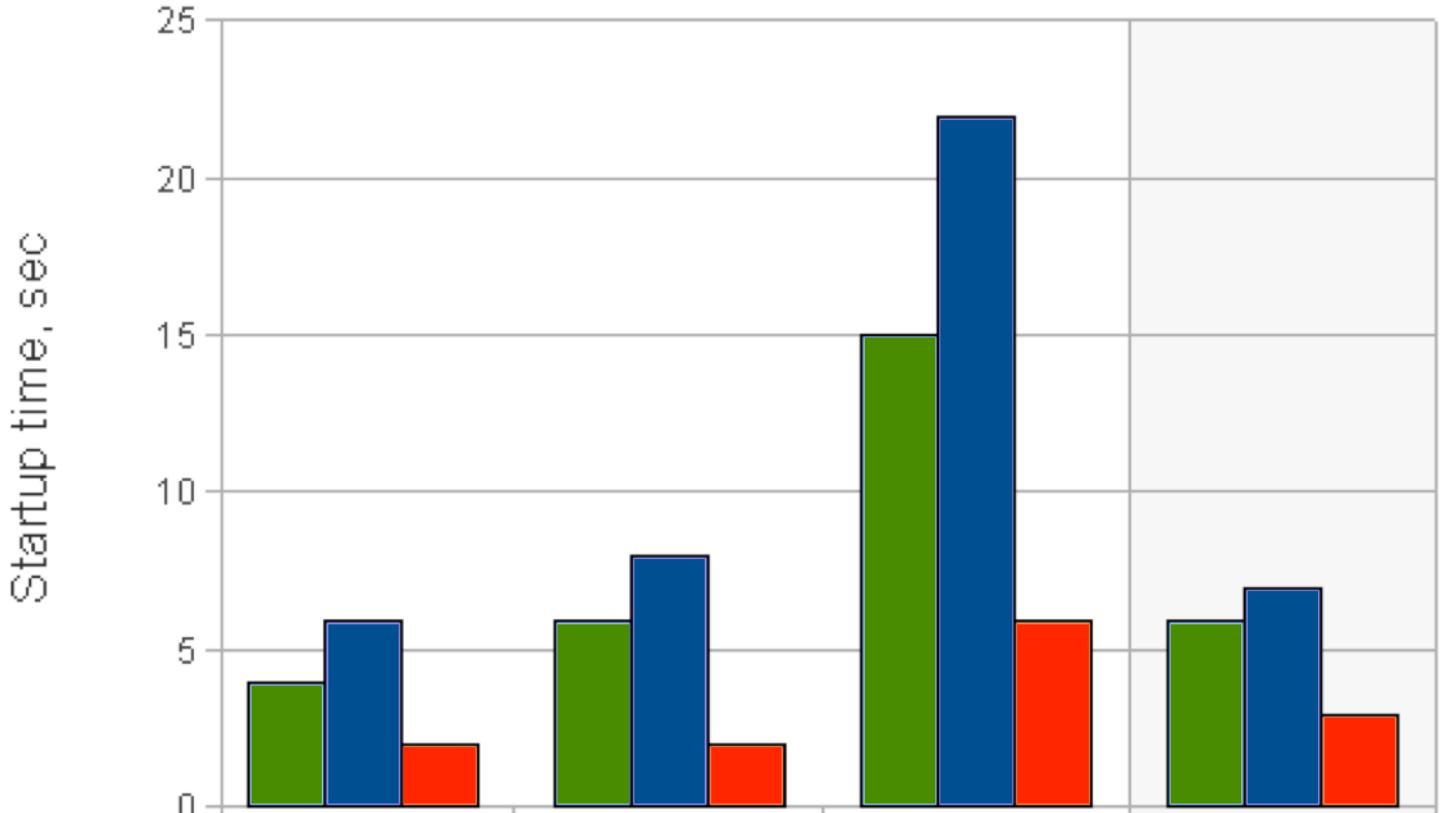
- Java Quick Start
  - Предзагружает rt.jar, динамические библиотеки

AOT быстрее?

- Машинный код “толще” Java bytecode
- Загрузка кода с диска занимает больше времени, чем его начальное исполнение

AOT быстрее!

- Код исполняемый на старте – в начало исполняемого файла
- Можно предзагружать стартовый сегмент последовательным чтением



	FeedDemon	FeedReader	RSSOwl (JRE)	RSSOwl (optim)
Cold (AC)	4	6	15	6
Cold (battery)	6	8	22	7
Warm	2	2	6	3

EXCELSIOR

Производительность

EXCELSIOR

Миф 4. AOT быстрее

“Скомпилировав Java статически мы получим скорость приложения как в C, C быстрее Java, следовательно -- AOT быстрее”

EXCELSIOR

Миф 5. JIT быстрее

“Эффективно оптимизировать Java приложения можно только на основе динамического профиля исполнения”

EXCELSIOR

Виды оптимизаций

- Протяжка констант
- Удаление избыточного кода
- Удаление общих подвыражений
- Открытая подстановка
- Специализация методов
- Развертывание циклов
- Версионирование циклов
- Вынос инвариантов
- Удаление хвостовой рекурсии
- Девиртуализация вызовов
- Аллокация объектов на стэке и их взрыв
- Удаление проверок времени исполнения
- Удаление избыточной синхронизации
- Оптимальная выборка кода и свертка шаблонов
- Планировка инструкций
- Оптимальное распределение регистров

EXCELSIOR

Java – ООП

- Много методов
- Методы маленькие (get/set)
- Методы по умолчанию ВИРТУАЛЬНЫЕ

Девиртуализация вызовов

- Предусловие дальнейшей открытой подстановки (inline)

- Анализ иерархии классов
  - метод не перегружается – невиртуальный
- Типовый анализ
  - `new T().foo(); //вызов foo() невиртуальный`
- Inline caches

EXCELSIOR

Анализ иерархии классов (CHA)

Идея: метод не перегружается – невиртуальный

```
A a;
…
a.foo();
```

→

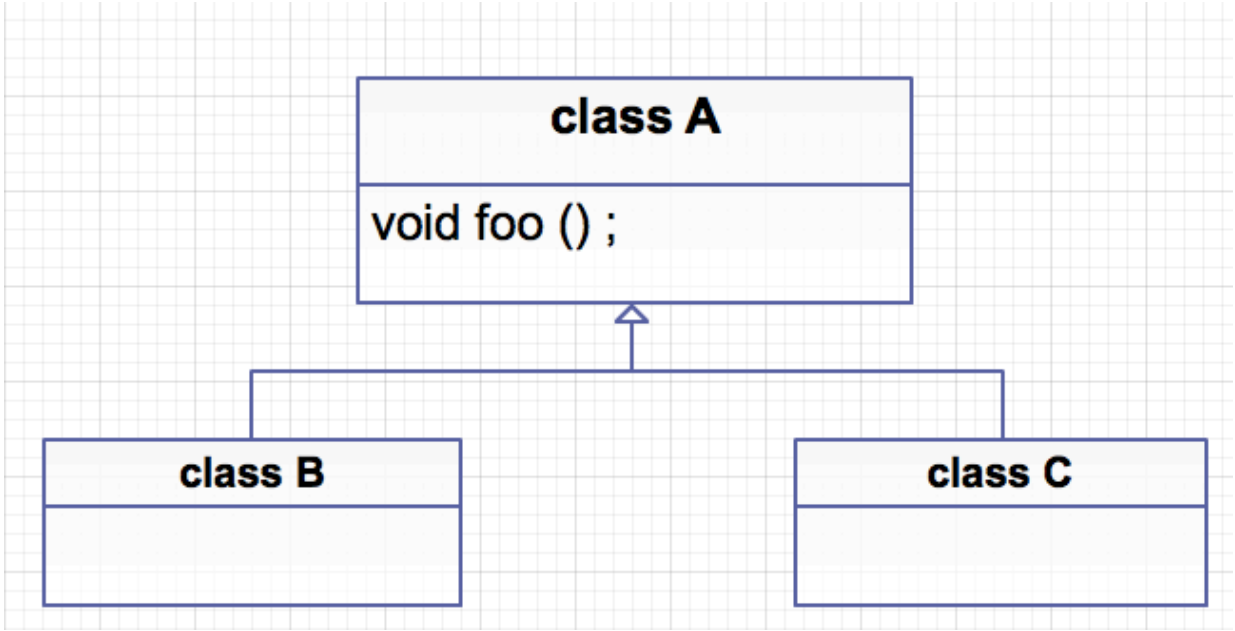
```
if (RT_Type(a) in CHA) {
    inlined body of foo()
} else {
    a.foo();
}
```

EXCELSIOR

Типовый анализ

Идея:

```
new A().foo(); //невиртуальный вызов
```



EXCELSIOR

Типовый анализ

```
A a = b? new B() : new C();
a.foo();
```

Если `foo()` в `B` и `C` один и тоже, то

`a.foo()` невиртуальный вызов

(можно инлайнить)

EXCELSIOR

Типовый анализ

```
A a = b? bar() : baz();
…
a.foo();
```

Если `bar()` возвращает только `new B`,
а `baz()` экземпляры `new C`, то

`a.foo()` – опять невиртуальный вызов
(можно инлайнить)

EXCELSIOR

Типовый анализ

Откуда мы знаем, что возвращает `bar()` и `baz()`?

Глобальный анализ

- Глобальный анализ – анализирует все методы программы, вычисляя про каждый метод полезную информацию

- Результаты глобального анализа используются для оптимизации конкретного метода

EXCELSIOR

Аллокация объектов на стэке

- Все Java объекты создаются в динамической памяти – Java heap (куча)
- Большинство объектов временные
- Хочется их размещать на стэке метода

- Escape анализ (анализ утечек) – определяет утекает ли объект в разделяемую память.

Пример

```
void foo() {
    for (Object o: getCollection()) {
        doSomething(o);
    }
}
```

EXCELSIOR

Пример

```
void foo() {
    Iterator iter = getCollection().iterator();
    while (iter.hasNext()) {
        Object o = iter.next();
        doSomething(o);
    }
}
```

EXCELSIOR

Пример

```
void foo() {
    ArrayList list = getCollection();
    Iterator iter = list.iterator();
    while (iter.hasNext()) {
        Object o = iter.next();
        doSomething(o);
    }
}
```

EXCELSIOR

Пример

```
void foo() {
    ArrayList list = getCollection();
    ArrayList.ListItr iter = new ListItr(list);
    while (iter.hasNext()) {
        Object o = iter.next();
        doSomething(o);
    }
}
```

EXCELSIOR

Пример

```
void foo() {
    ArrayList list = getCollection();
    ArrayList.ListItr iter = onStack ListItr();
    iter.this$0 = list;
    iter.cursor = 0;
    iter.size = list.elemData.length;
    while (iter.hasNext()) {
        Object o = iter.next();
        doSomething(o);
    }
}
```

EXCELSIOR

Пример

```
void foo() {
    ArrayList list = getCollection();
    ArrayList.ListItr iter = onStack ListItr(list);
    iter.this$0 = list;
    iter.cursor = 0;
    iter.size = list.elemData.length;
    while (iter.cursor < iter.size) {
        int index = iter.cursor++;
        Object o = iter.this$0.elemData[index];
        doSomething(o);
    }
}
```

EXCELSIOR

Пример

```
void foo() {
    ArrayList list = getCollection();
    int cursor = 0;
    int size = list.elemData.length;
    while (cursor < size) {
        Object o = list.elemData[cursor++];
        doSomething(o);
    }
}
```

EXCELSIOR

Пример

```
void foo() {
    ArrayList list = getCollection();

    int size = list.elemData.length;
    for (int i = 0; i < size; i++) {
        doSomething(list.elemData[i]);
    }
}
```

EXCELSIOR

Анализ и оптимизации

- Часто бывают довольно сложными
- Требуют итеративного пересчета
- Глобальный анализ зависит от ВСЕЙ программы.

EXCELSIOR

Анализ и оптимизации

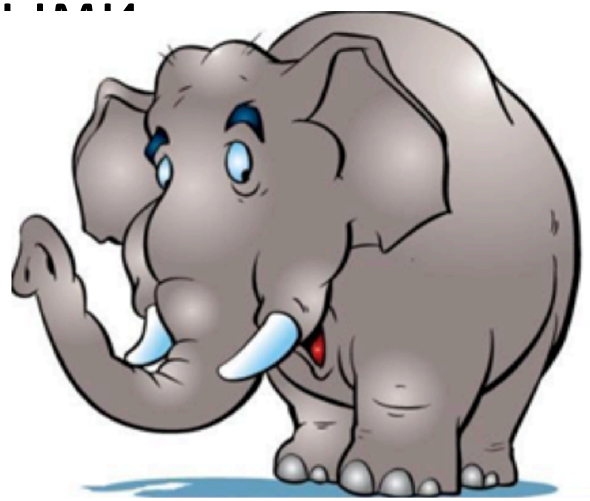
- Часто бывают довольно сложными
- Требуют итеративного пересчета
- Глобальный анализ зависит от ВСЕЙ программы.

Все ли это может
себе позволить JIT?

JIT

Анализ и оптимизации

- Часто бывают довольно сло
- Требуют итеративного пере
- Глобальный анализ зависи



Все ли это может себе позволить JIT?

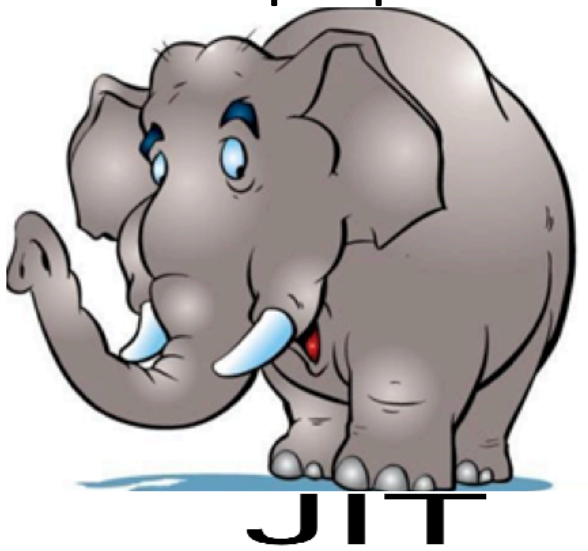
JIT

EXCELSIOR

Анализ и оптимизации

- Часто бывают довольно сложными
- Требуют итеративного пересчета
- Глобальный анализ зависит от ВСЕЙ программы.

Все ли это может себе позволить JIT?



EXCELSIOR

Динамические оптимизации

- Профилировка и селективная компиляция
- Открытая подстановка на основе профиля исполнения
- Оптимизация трасс
  исполнения
- Выбор оптимальных инструкций

EXCELSIOR

Горячий код vs теплый

- А что будет, если у приложения нет ярко выраженного горячего кода?

- Долгий прогрев, результаты прогрева не используются при дальнейших стартах приложения

EXCELSIOR

Динамические оптимизации

Может ли статический компилятор использовать динамический профиль исполнения?

EXCELSIOR

Server side

- Процессорное время в
  облаках – дорого

- Через некоторое время работы сервера – профиль исполнения стабилизируется

- Почему этот профиль не передать статическому компилятору?

Не кодом единым …

- Кроме кода приложения, есть еще код JVM
  - Управление памятью
  - Сборка мусора
  - Потоки и синхронизация
  - Обработка исключительных ситуаций
  - …
- Кроме Java кода, есть сторонний код
  - Native методы + нативные библиотеки (в т.ч. ОС)

EXCELSIOR

Embedded

- Чем слабее железо, тем дороже динамическая компиляция

- Встроенные системы часто не обладают теми же вычислительными мощностями, что и настольные компьютеры/сервера

Mobile

- JVM on Mobile:

EXCELSIOR

Mobile

- JVM on Mobile:
  - Платформенные классы

EXCELSIOR

Mobile

- JVM on Mobile:
  - Платформенные классы
  - GC и Memory Manager

EXCELSIOR

Mobile

- JVM on Mobile:
  - Платформенные классы
  - GC и Memory Manager
  - Reflection

EXCELSIOR

Mobile

- JVM on Mobile:
  - Платформенные классы
  - GC и Memory Manager
  - Reflection
  - JIT

EXCELSIOR

Mobile

- JVM on Mobile:
  - Платформенные классы
  - GC и Memory Manager
  - Reflection
  - JIT

Что не хватает?

EXCELSIOR

Mobile

- JVM on Mobile:
  - Платформенные классы
  - GC и Memory Manager
  - Reflection
  - JIT

Зарядка!

EXCELSIOR

Mobile

- У беспроводных устройств есть БАТАРЕЙКА
- Стоит ли ее тратить на динамическую компиляцию?

EXCELSIOR

iOS

- Политика распространения приложений на iOS запрещает любую динамическую загрузку кода

- Для Java возможен либо интерпретатор, либо AOT

AOT компиляторы для Java

GCJ

Excelsior JET

Android ART

RoboVM

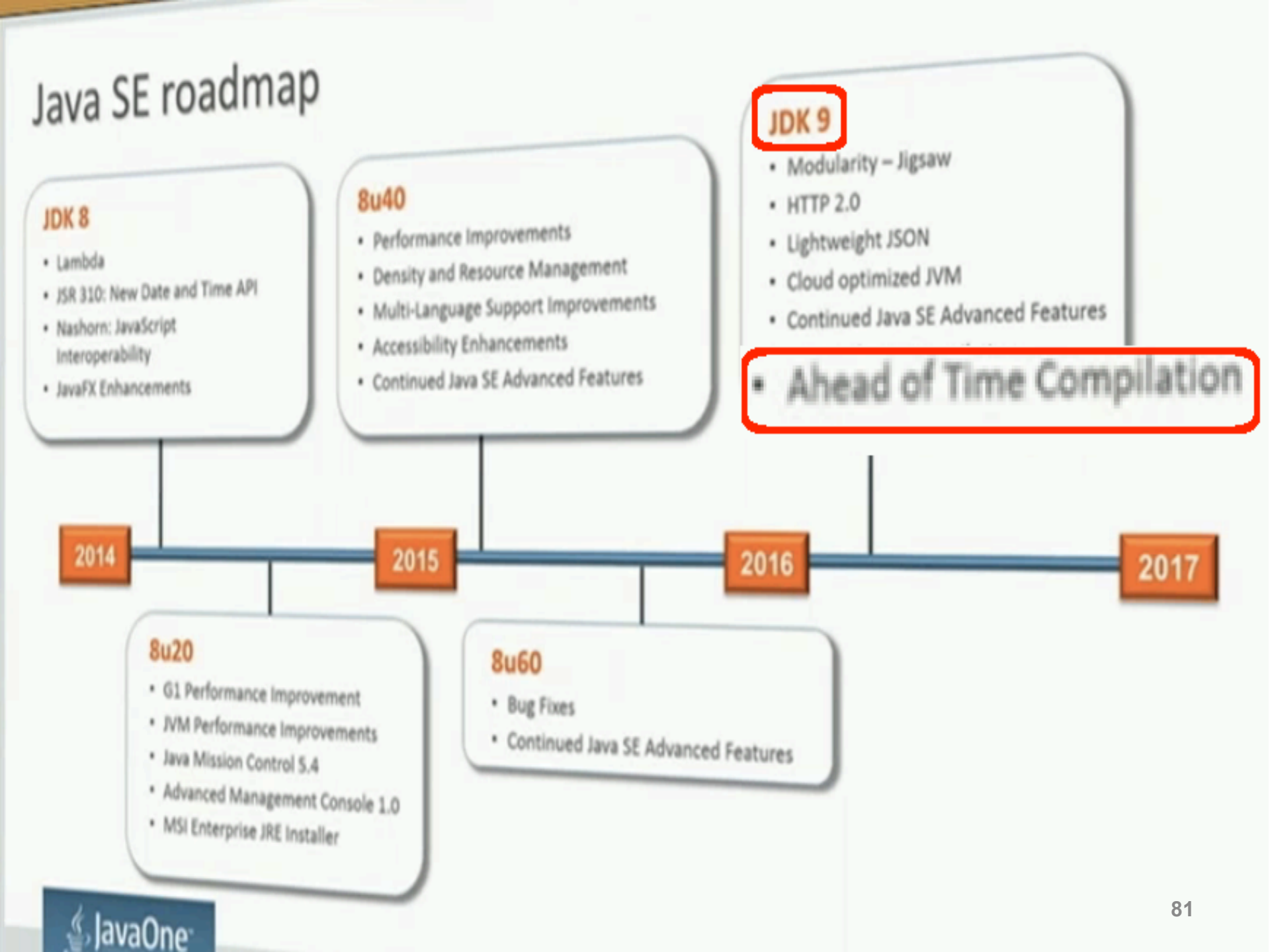
Avian

CodeNameOne

IBM J9

Java ME

EXCELSIOR


Java SE roadmap
JDK 8
• Lambda
• JSR 310: New Date and Time API
• Nashorn: JavaScript Interoperability
• JavaFX Enhancements
8u40
• Performance Improvements
• Density and Resource Management
• Multi-Language Support Improvements
• Accessibility Enhancements
• Continued Java SE Advanced Features
JDK 9
• Modularity – Jigsaw
• HTTP 2.0
• Lightweight JSON
• Cloud optimized JVM
• Continued Java SE Advanced Features
• Ahead of Time Compilation
2014
2015
2016
2017
8u20
• G1 Performance Improvement
• JVM Performance Improvements
• Java Mission Control 5.4
• Advanced Management Console 1.0
• MSI Enterprise JRE Installer
8u60
• Bug Fixes
• Continued Java SE Advanced Features
JavaOne

Итоги

Статическая компиляция Java

- возможна
  - с сохранением всех возможностей Java
  - даже в присутствии нестандартных загрузчиков классов
- полезна
  - для защиты кода
  - быстрого старта
  - лучшего UX
  - улучшения производительности
  - экономии батарейки, памяти, диска

EXCELSIOR

Вопросы и ответы

Никита Липский,
Excelsior

nlipsky@excelsior-usa.com
twitter: @pjBooms

EXCELSIOR

Ускорение приложений