Оптимизация представления сложных Java объектов (проект ObjectLayout)

http://objectlayout.org

Иван Крылов

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Случай

- Код обходит массив с записями, с разным шагом, каждая запись читается и обрабатывается.
- Был код на Си
- Переписали на Java - стало существенно медленнее
- Мы редуцировали до следующего примера
- Demo

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Код Си и Java

- Обходит матрицу сверху вниз
- Размеры элементов с учетом заголовка java-объектов одинаковы
- Необходимое “прогревание” сделано в обоих случаях
- Влияние GC исключено, остается..

©2014 Azul Systems, Inc.

AZUL SYSTEMS

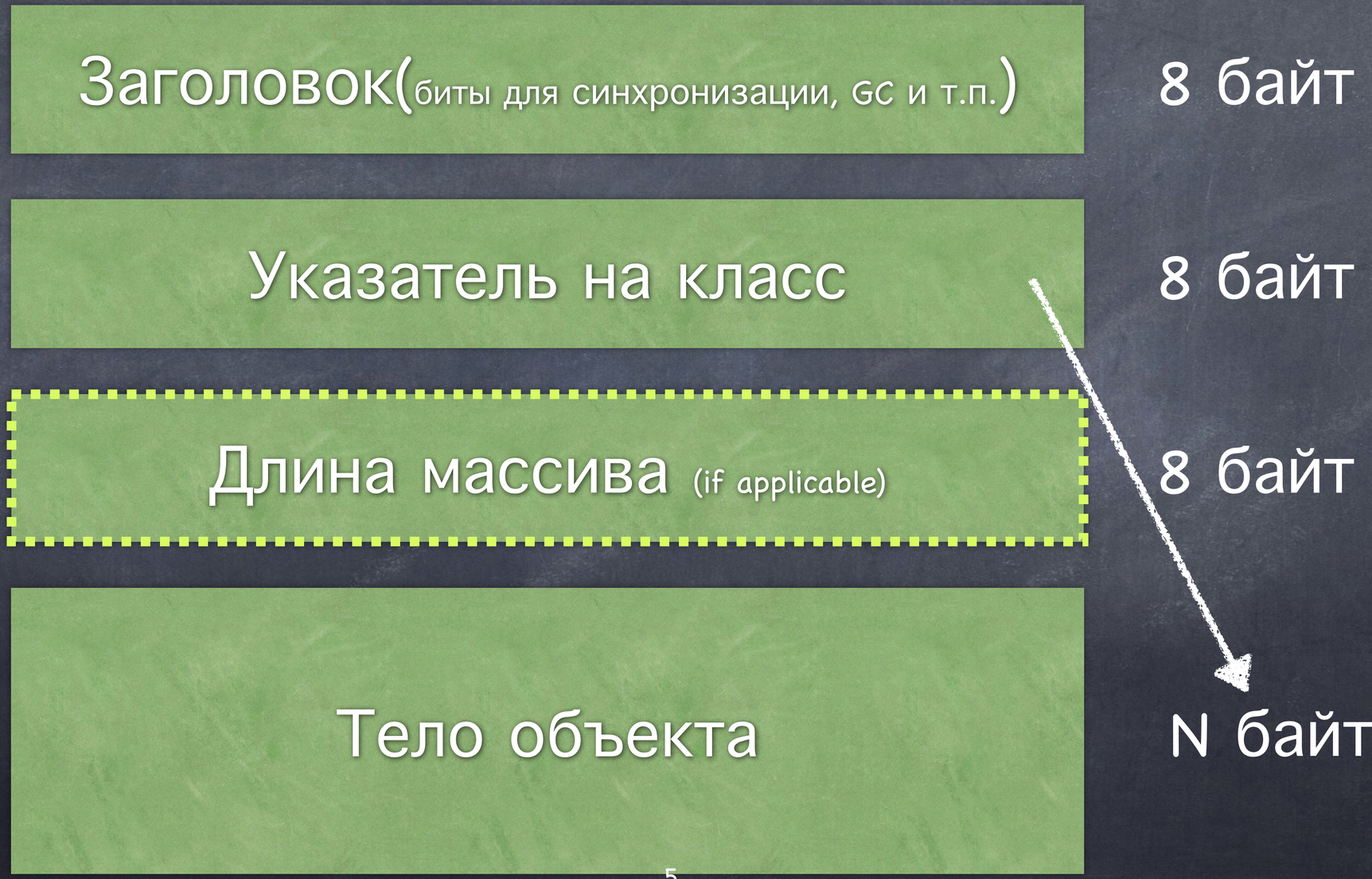
Зачем

- Вам слушать?
  - Понять проблему
  - Услышать возможные решения
  - Готового решения нету
- Мне рассказывать?
  - Мы над этим работаем и собираем отзывы
  - Хотим, чтобы `StructuredArray` попал в стандарт Java N (9+)

©2014 Azul Systems, Inc.

AZUL SYSTEMS

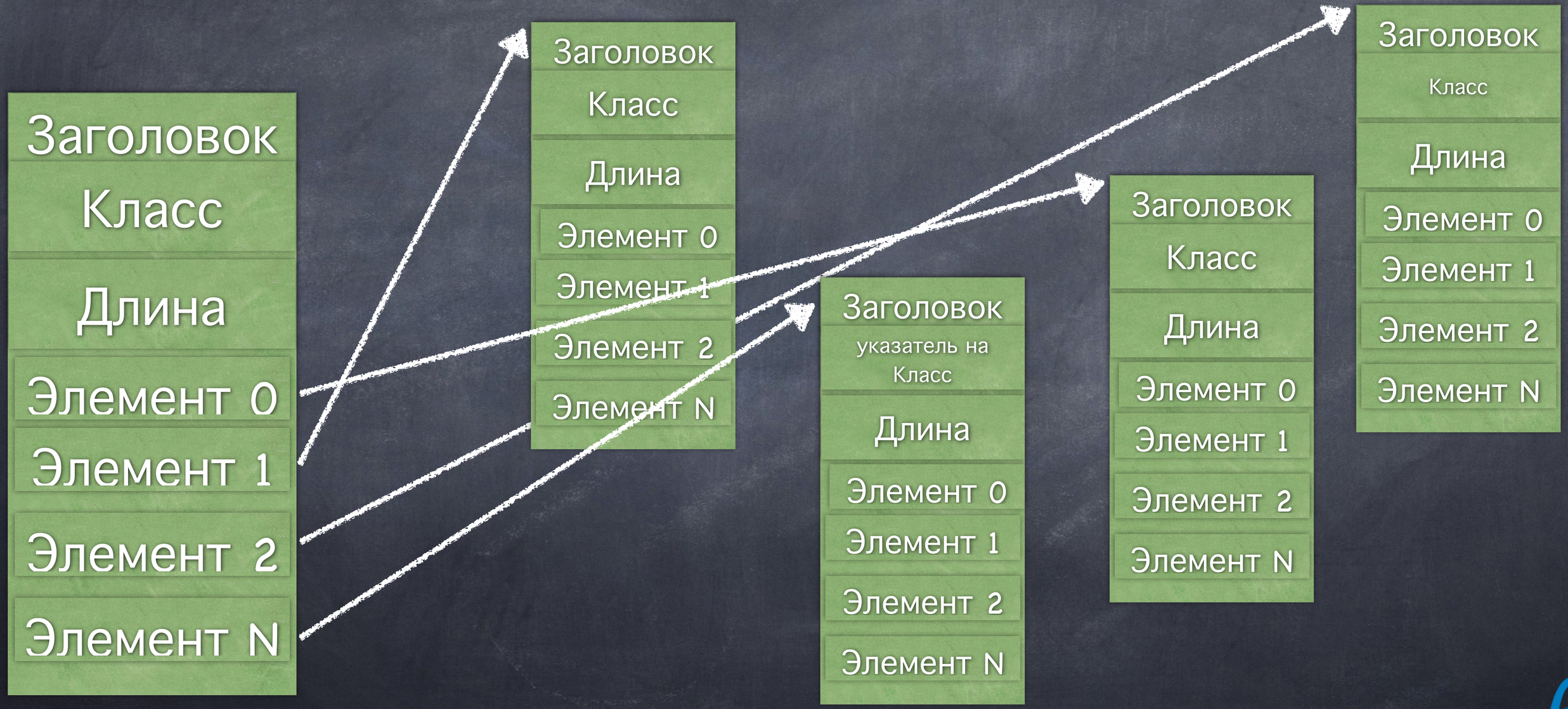
Структура Java-объекта



©2014 Azul Systems, Inc.

AZUL SYSTEMS

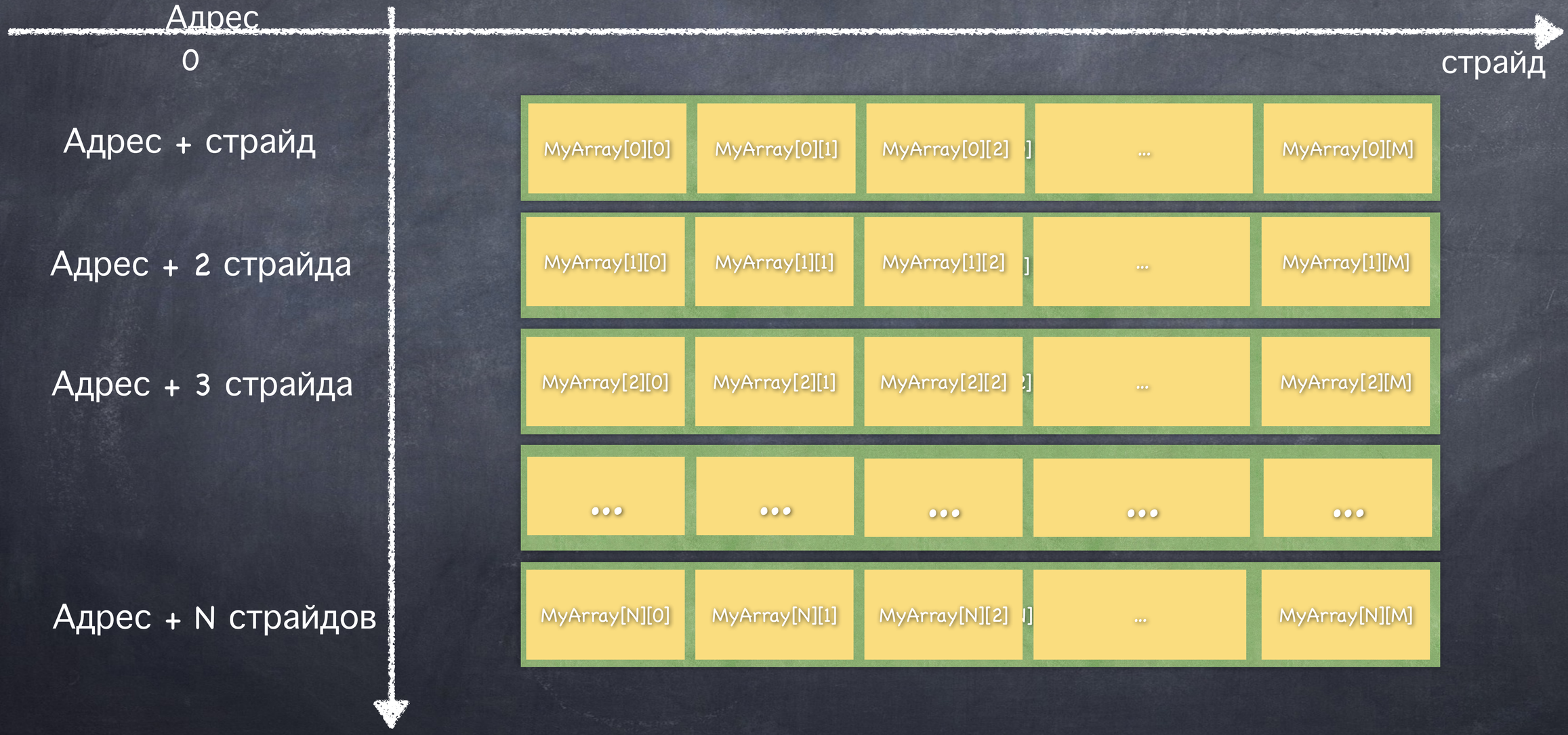
Структура Массива Массивов



©2014 Azul Systems, Inc.

AZUL SYSTEMS

Двумерный Массив в Си



©2014 Azul Systems, Inc.

AZUL SYSTEMS

2 причины, почему Си быстрее

- Адреса всех полей известны заранее:
  - Адреса “внутренних” структур вытекают из адреса “внешней структуры”
  - Нет 2го, 3-го, N-ного уровня индирекции. Чтение поля не ждет результата чтения адреса
- Локальность и последовательность (массивы)
  - обход массива – это доступ к соседним адресам
  - Выравненность + помощь префетчеров

©2014 Azul Systems, Inc.

AZUL SYSTEMS

В чем синтаксическая разница?

- Java `Object[]` против Си `struct foo[]`
- Java: изменяемый* массив объектов с общим базовым классом (*речь о контейнере,а не данных)
- Си: неизменяемый массив одинаковых структур
- Изменяемость контейнера и варьирование размера элементов требует лишних обращений к памяти

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Нужны ли структуры в Java?

- “Нам нужны структуры в Java, чтобы...”.
  - Многим уже пришлось переписать свой Java код с использованием ByteBuffer, и т.п.
- “Но структуры то уже есть. Они называются Объекты”.
  - Нам нужен набор оптимизаций, который даст производительность, сравнимую с Си
- Надо определить рамки, набор ограничений, в которых возможно провести оптимизации

©2014 Azul Systems, Inc.

AZUL SYSTEMS

ObjectLayout: отправная точка

- Придумать и описать категории объектов, которые стоит оптимизировать
- Не хотим изменений в спецификации языка
- Начнем с “простых” классов
- В дальнейшем, ВМ должна распознавать такие классы, оптимизировать представление и аксессоры
- Интринсики влияют только на скорость

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Интересные для ObjectLayout виды структур

©2014 Azul Systems, Inc.

AZUL SYSTEMS

StructuredArray

- массив структур

  struct foo[];

- вложенные структуры

  struct foo { int a; struct bar b; int c; };

- структуры с массивом в качестве последнего поля

  struct packet { int len; char[] body; }

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Массивы структур

Заголовок
Класс
Длина
Элемент 0
Элемент 1
Элемент 2

element[2]

Заголовок
Класс
{
primitiveType field1;
primitiveType field2;
....
primitiveType fieldN;
}

element[0]

Заголовок
Класс
{
primitiveType field1;
primitiveType field2;
....
primitiveType fieldN;
}

element[1]

Заголовок
Класс
{
primitiveType field1;
primitiveType field2;
....
primitiveType fieldN;
}

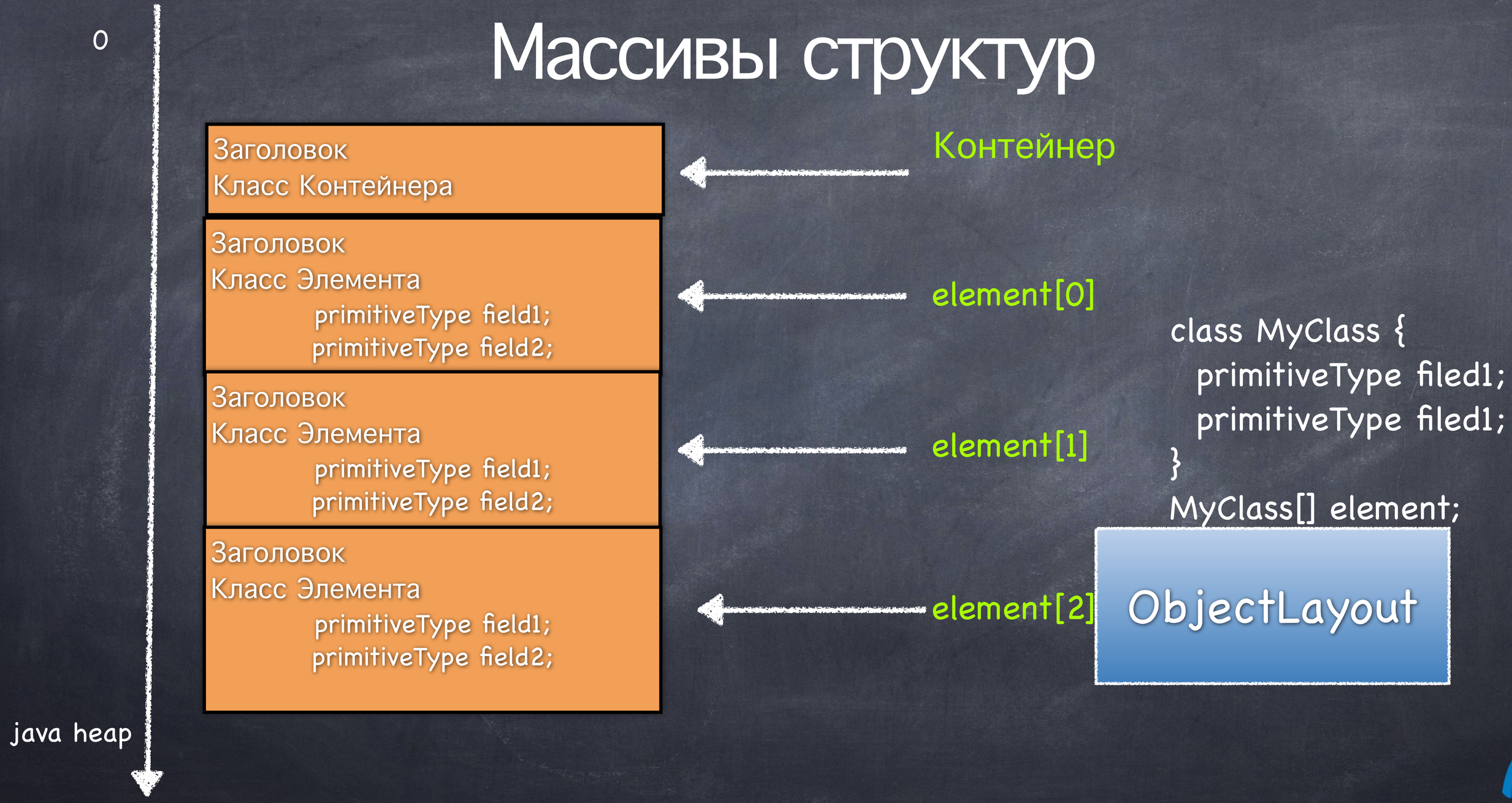
```
MyClass[] element;
class MyClass {
  primitiveType filed1;
  primitiveType filed1;

  ...
}
```

Стандартное представление

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Массивы структур



©2014 Azul Systems, Inc.

AZUL SYSTEMS

IntrinsicObjectModels

- массив структур

  struct foo[];

- вложенные структуры

  struct foo { int a; struct bar b; int c; };

- структуры с массивом в качестве последнего поля

  struct packet { int len; char[] body; }

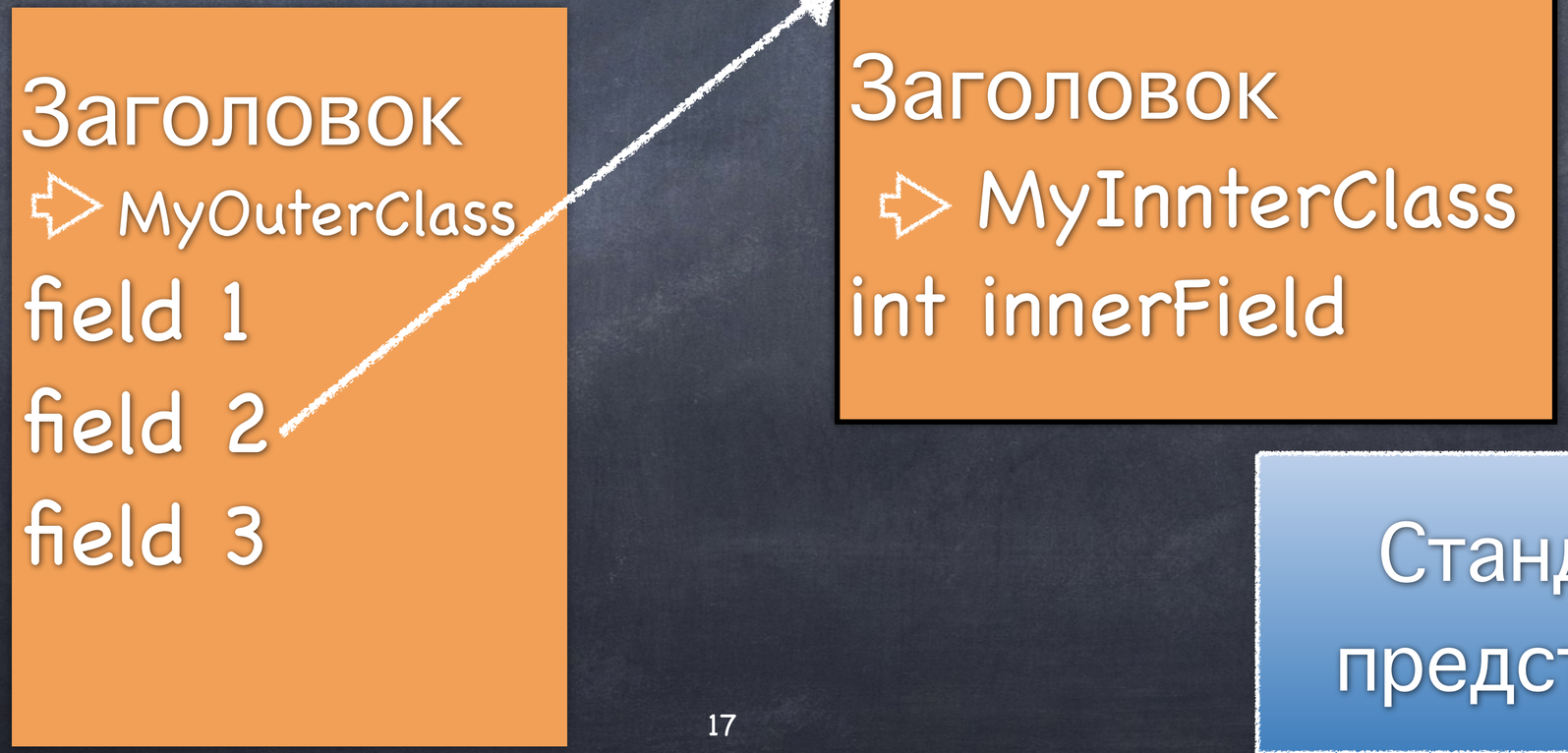
©2014 Azul Systems, Inc.

AZUL SYSTEMS

Вложенные структуры

```
class MyOuterClass{
  int field1;
   MyInnerClass field2;
   long field3;
}
```

```
class MyInnerClass{
 int innerField;
}
```



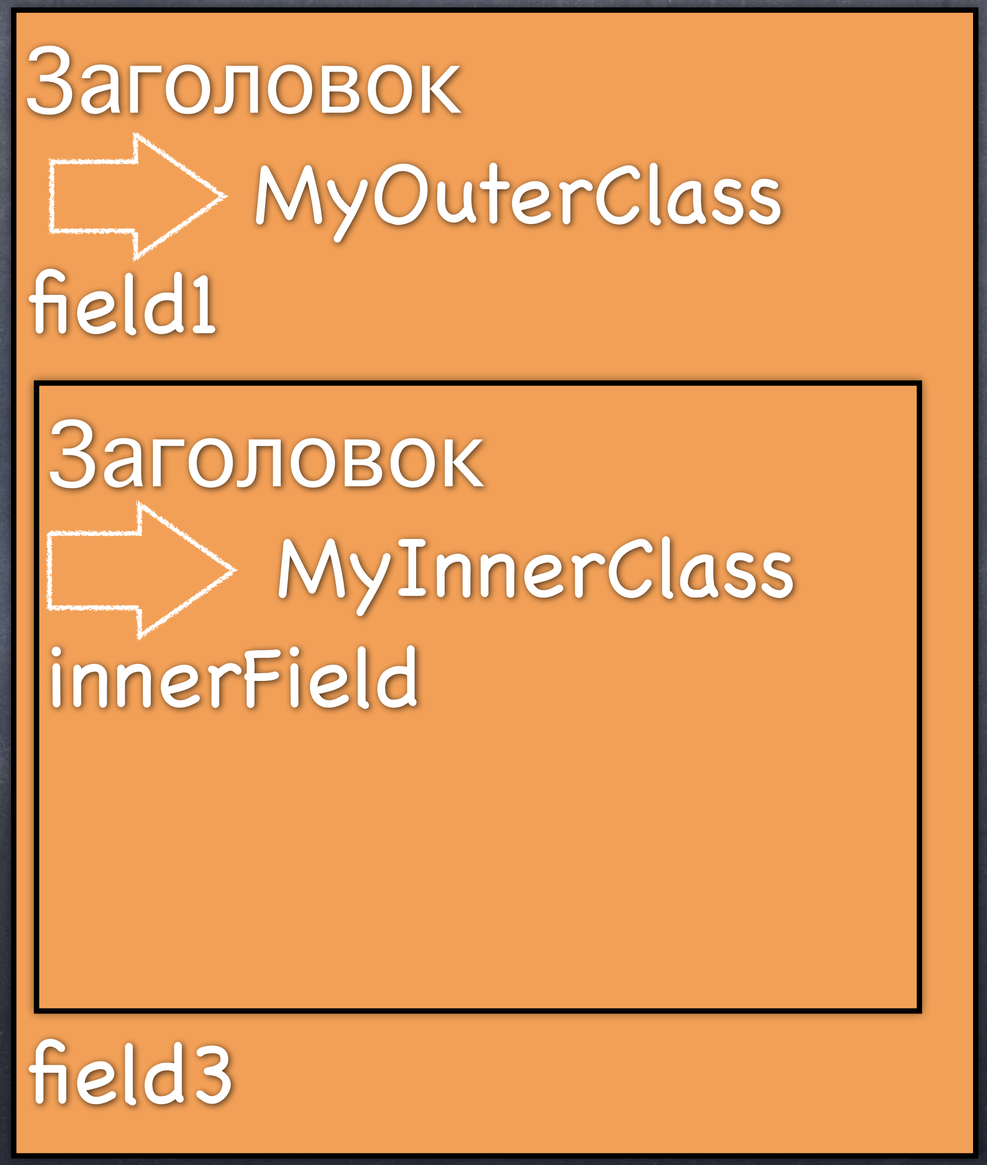
Стандартное представление

©2014 Azul Systems, Inc.

Вложенные структуры

```
class MyOuterClass{
    int field1;
    MyInnerClass field2;
    long field3;
}

class MyInnerClass{
   int innerField;
}
```

ObjectLayout



©2014 Azul Systems, Inc.

AZUL SYSTEMS

PrimitiveArrays

- массив структур

  struct foo[];

- вложенные структуры

  struct foo { int a; struct bar b; int c; };

- структуры с массивом в качестве последнего поля

  struct packet { int len; char[] body; }

©2014 Azul Systems, Inc.

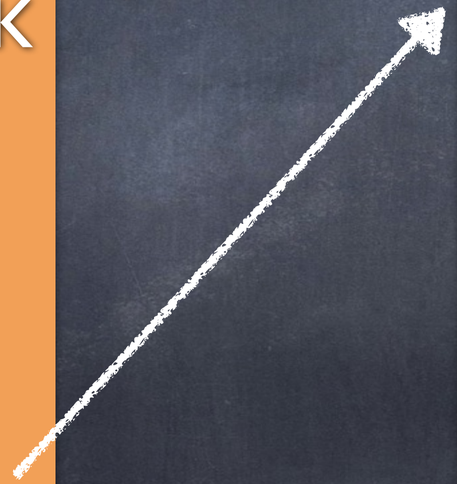
AZUL SYSTEMS

Структура с массивом в конце

```
class MyClass{
  int field1;
  long field2;
  int[] fieldArray;
}
```

Стандартное представление

Заголовок
Класс
field 1
field 2
fieldArray



Заголовок
Класс
Длина
fieldArray[0]
fieldArray[...]

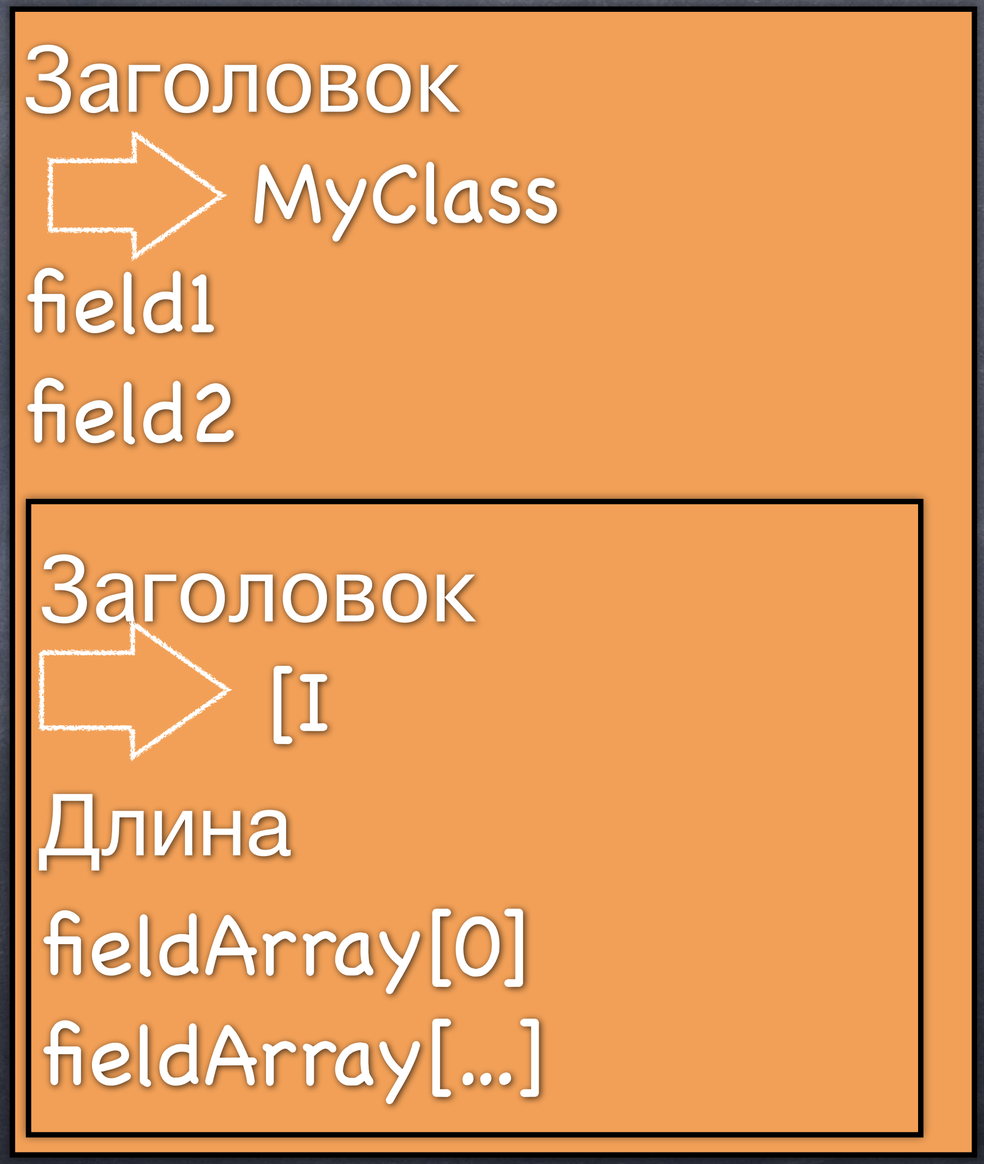
©2014 Azul Systems, Inc.

AZUL SYSTEMS

Структура с массивом в конце

```
class MyClass{
  int field1;
  long field2;
  int[] fieldArray;
}
```

ObjectLayout



©2014 Azul Systems, Inc.

AZUL SYSTEMS

StructuredArray

- массив структур

  struct foo[];

- вложенные структуры

  struct foo { int a; struct bar b; int c; };

- структуры с массивом в качестве последнего поля

  struct packet { int len; char[] body; }

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Решение

- Предлагаемое решение

  StructuredArray<T>: Неизменяемый массив

  [потенциально] изменяемых объектов одного класса (T)

- Поддержка инициализации через “конструктор”
  поддержка get(), но не put()...

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Реализация а-ля java.util.concurrent

- Сначала java.util.concurrent развивался отдельно от JDK
- Смотрели, как сделаны быстрые параллельные операции
- На начальном этапе изменений в язык и JVM* - ноль. Просто набор новых классов
- Пример интринсики: BM распознает AtomicLong CAS -> x86's CMPXCHG
- namespace Java-только так в будущем гарантируется работа unsafe и инстринсик

©2014 Azul Systems, Inc.

AZUL SYSTEMS

StructuredArray<T>

- Коллекция объектов класса T
- Представление-массив: T element = get(index);
- Коллекция неизменна,нельзя замещать эл-ты
- Создание через factory метод:

```
a = StructuredArray.newInstance(SomeClass.class, 100);
```

- Все элементы создаются вместе с контейнером
- Поддержка произвольных конструкторов эл-тов
  - В т.ч. index-specific CtorAndArgs

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Время жизни

- Отбросили подход “элементы сохраняют “живыми” свои контейнеры”.
- т.к. разные объекты (даже в одной коллекции) имеют разное время жизни
- Итого: `StructuredArray` обычная коллекция
  - контейнер жив => живы элементы
  - (некоторые) элементы живы => контейнер жив

*** В целях оптимизации, ВМ может не удалять контейнер, просто чтобы элементы “не расползались” в памяти, но это не должно быть видимо снаружи

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Достоинства такого подхода

- StructuredArray - обычная коллекция объектов
  - Поведение как у других коллекций
  - Если рантайм "знает" о такой коллекции - то может оптимизировать
- Элементы StructuredArray - нормальные объекты
  - Поддерживают обход объектов при GC
  - Содержат поля для синхронизации по объекту
  - Хэш-коды там же и такие же
  - Сторонний код не требует специальных аксессоров
- Для нас - это простой и естественный подход

©2014 Azul Systems, Inc.

AZUL SYSTEMS

StructuredArray<T> еще плюсы

- 8-байтные индексы элементов массива (давно пора, уже 2014й)
- Многомерность
  - вложенные StructuredArrays
- От StructuredArray можно наследовать
- StructuredArray - конструкторы запрещены
  - Создание через factory методы

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Оптимизированный рантайм

- Концепция: “внутренний” и “внешний” объект
  - И тот и другой - обычные объекты
  - Рантайм может перейти от “внутреннего” к “внешнему”
  - При GC перемещение “внутреннего” объекта при живом “внешнем” влечет перемещение всего контейнера
- Несложная реализация для всех GC в Hotspot (и нашего C4, и, вероятно, всех других)

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Оптимизированный рантайм

- Польза от последовательного размещения в памяти
  - HW префетчеры помогут
  - Даже не потребует оптимизаций компилятора
- Арифметика адреса поля потребует изменений в компиляторе
  - e = (T) (a + a.bodySize + (index * a.elementSize));
  - elementSize и bodySize - это не константы
  - Оптимизации не сложнее CHA & inline-cache

©2014 Azul Systems, Inc.

AZUL SYSTEMS

IntrinsicObjectModels

- массив структур

  struct foo[];

- вложенные структуры

  struct foo { int a; struct bar b; int c; };

- структуры с массивом в качестве последнего поля

  struct packet { int len; char[] body; }

©2014 Azul Systems, Inc.

AZUL SYSTEMS

IntrinsicObjectModels

- Вложенные значит располагаются прямо тут, а не по ссылке
- Factory метод по прежнему создает “все и сразу”

```
class SomeClass {
  int a;
  final SomeOtherClass o =
    IntrinsifiedObjects.createIntrinsicObject(SomeOtherClass.class);
}
```

- Предстоит придумать, как защититься от записи через reflection

©2014 Azul Systems, Inc.

AZUL SYSTEMS

PrimitiveArrays

- массив структур

  struct foo[];

- вложенные структуры

  struct foo { int a; struct bar b; int c; };

- структуры с массивом в качестве последнего поля

  struct packet { int len; char[] body; }

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Структуры с массивами в конце: специальный wrapper-класс

- Бонус - возможность наследоваться от массива
- ObjectLayout - по wrapper-классу для каждого из элементарных массивов, типа PrimitiveLongArray, PrimitiveDoubleArray, и т.п.
- + Массив референсов ReferenceArray<T>
- От StructuredArray<T> можно наследоваться, что будет означать “структура с массивом структур в конце”

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Оптимизация представления объектов в ВМ

- Для чего: только для улучшения производительности
- **Мы не рассматриваем в качестве целей**:
  - Экономия памяти
  - off-heap объекты
  - константные объекты

} Важные задачи, но не касаются данной темы

©2014 Azul Systems, Inc.

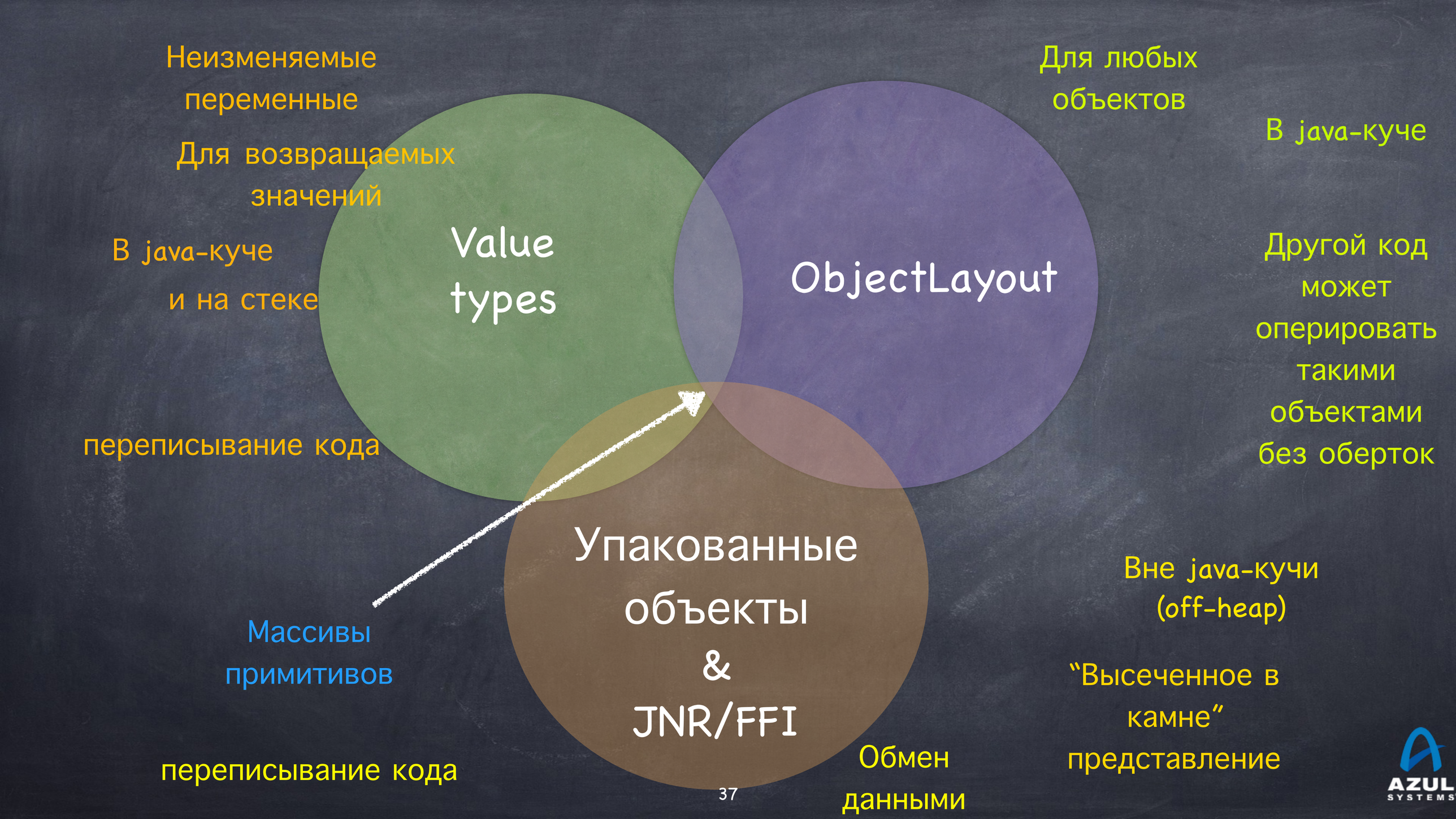
AZUL SYSTEMS

Другие подходы (?)

- Value types (JDK9+)
  - http://cr.openjdk.java.net/~jrose/values/values.html
  - Приходите на доклад Владимира Иванова “Future of Java: 9 and beyond”. Вторник 17:15
- IBM’s Packed Objects - Упакованные объекты
  - http://www.oracle.com/technetwork/java/jvmls2013sciam-2013525.pdf

©2014 Azul Systems, Inc.

AZUL SYSTEMS



AZUL SYSTEMS

Состояние проекта

- Синтаксис и базовая реализация на Гитхабе.
- StructuredArray - интерфейс зафиксировался.
  IntrinsicObjectModels & PrimitiveArrays - еще меняется
- Реализация интринсиков для Zing и OpenJDK на подходе
- Что дальше: проект OpenJDK, JEP...
- Цель: ObjectLayout в Java SE (9?)
  - Работающая реализация для всех JDKs (pure java)

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Пару слайдов об Azul Systems

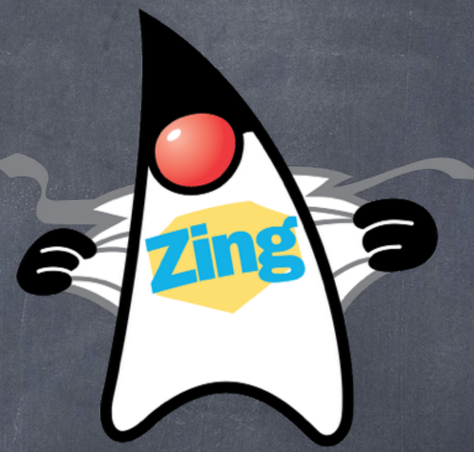
- Мы делаем масштабируемую среду исполнения Java программ
- С 2002-года
- 3 поколения SMP Multi-core машин (Vega)
- **Сегодняшний Zing: чисто софтверное решение для x86**
- Zing известен за то, что он
  - Low Latency
  - Предсказуемый (GC)
  - Поддержка больших значений XMX

©2013 Azul Systems, Inc.

AZUL SYSTEMS

Azul это 2 ключевых продукта

- Zing: лучшая ВМ-ка по ряду параметров (x86, только linux)
- Zulu: Продуктизированный OpenJDK с платной поддержкой
- Долгосрочная поддержка для версий Java SE 6, 7, 8
- Zulu: Бесплатная & Open Source, мы отдаем свои изменения обратно в OpenJDK
- Zulu: Windows, Linux, Mac OS X, Azure & EC2



©2013 Azul Systems, Inc.

AZUL SYSTEMS

Заключение

- Новый пакет: org.ObjectLayout.*
  - Но хотим перевести в namespace java.* ко времени Java SE (9?)
- Работает "без допиливания" на Java 6, 7, 8, 9, ...
  - Нового синтаксиса - 0 , не требует новых байткодов
  - JVM должен "знать" о таких хитрых объектах
- Оптимизированные JRE будут заметно ускорять код
  - Достаточно просто, несколько точечных изменений в JVM
  - Мы надеемся увидеть все "ускорения" в OpenJDK (9?)
  - Zing получит* эти оптимизации для Java 6, 7, 8, 9, ...

  * - это лишь в планах

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Спасибо

http://www.azulsystems.com

http://objectlayout.org

https://github.com/ObjectLayout/ObjectLayout

@JohnWings

AZUL SYSTEMS

Backup slides

AZUL SYSTEMS

Словарик

- Интринсики (Intrinsics) - java методы, которые JVM подменяет на внутренние функции
- –> POJO - Plain Old Java Object (след. слайд)
- Factory method
- Layout - представление в памяти
- StructuredArray & ObjectLayout
- Wrapper-class - класс-оболочка
- Аксессоры, геттеры, сеттеры
- Префетч(ер) - механизм спекулятивной загрузки данных по адресу

©2014 Azul Systems, Inc.

AZUL SYSTEMS

Простые классы (POJO)

- Не являются реализациями интерфейсов
- Прямые наследники `Object class`
- Поля являются `private`
- Пустой конструктор
- Стандартные геттеры-сеттеры полей

©2014 Azul Systems, Inc.

AZUL SYSTEMS