Cloud Native JVM

Cloud Runtime

Владимир Воскресенский
Azul Systems
Distinguished Engineer
vladimir@azul.com

azul

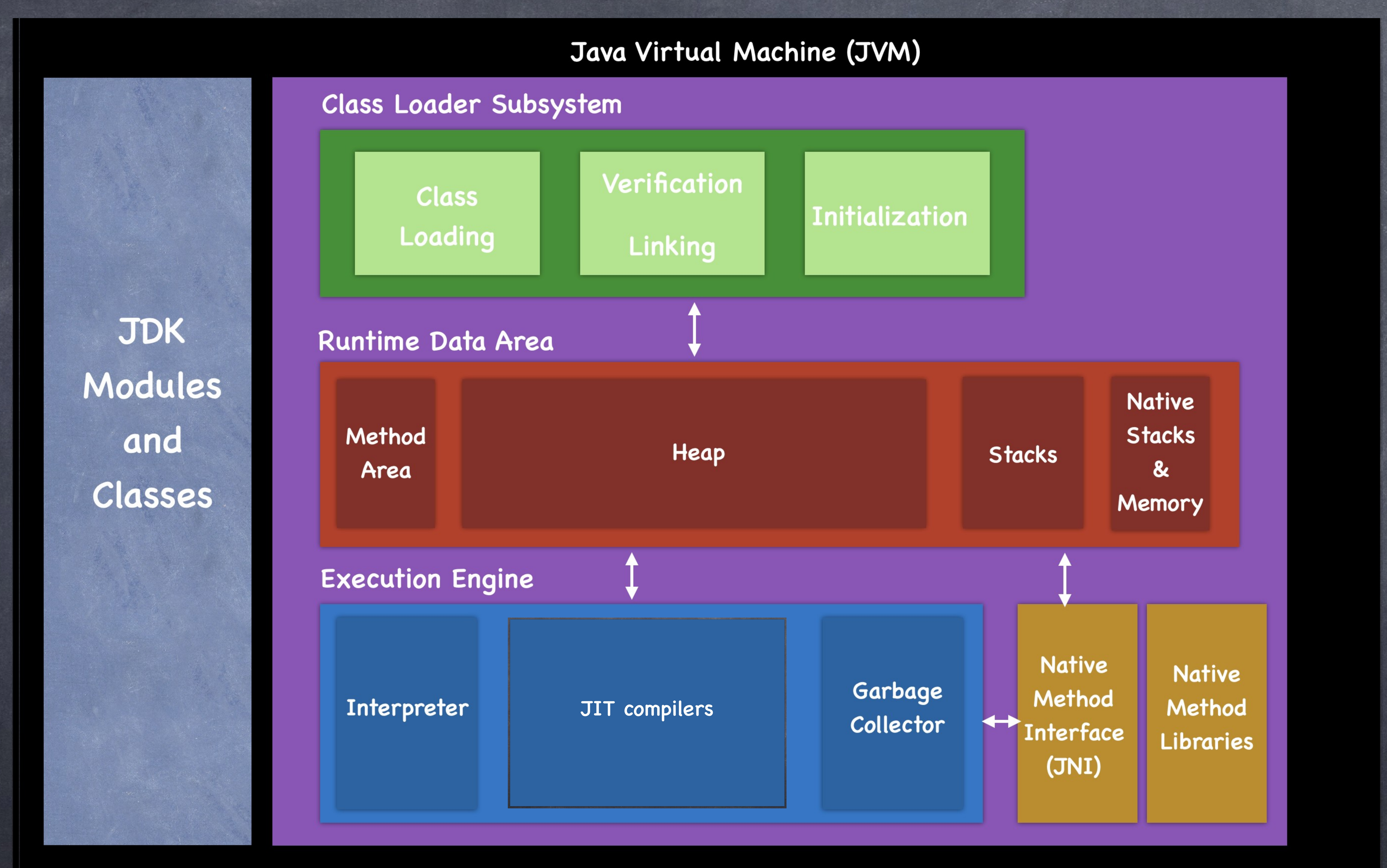
©2023 Azul Systems, Inc.

Cloud Native Compiler

https://www.youtube.com/watch?v=KFgnB8l6p_U

©2023 Azul Systems, Inc.
azul

Устройство JRE



©2023 Azul Systems, Inc.
azul

Java Runtime Environment

JRE = JVM + Библиотеки классов

Java Development Kit

JDK = JRE + инструменты разработки и отладки

©2023 Azul Systems, Inc.
azul

Рантаймы очень популярны

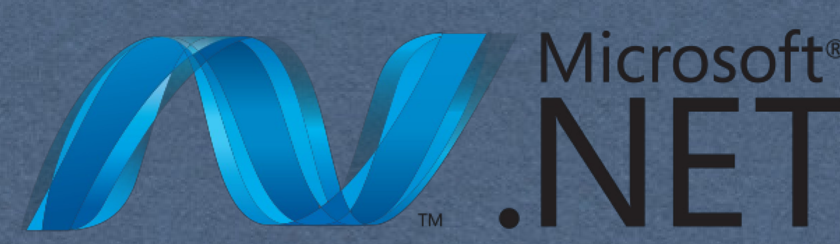
©2023 Azul Systems, Inc.

Kotlin Java

Scala

Clojure

C# F#



Go

PHP

Python

Erlang

JavaScript

Objective-C Swift

Ruby

©2023 Azul Systems, Inc.

azul

Продуктивность? Надежность?

Много экспертов? Стабильность?

Почему?

Скорость разработки? Стоимость разработки? Крупные Экосистемы?

Производительность?

©2023 Azul Systems, Inc.
azul

Облака очень популярны

©2023 Azul Systems, Inc.

Продуктивность? Надежность?

Много экспертов? Стабильность?

Почему?

Скорость разработки? Стоимость разработки? Крупные Экосистемы?

Производительность?

©2023 Azul Systems, Inc.
azul

Cloud Native

©2023 Azul Systems, Inc.

JVM (сейчас)

- Изолирована от других JVM
- Нет “памяти” о прошлых/других JVM запусках
- Полностью полагается лишь на себя
  - Ограничена локальными ресурсами
  - Ограничена своим функционалом
  - Должна выбирать на что тратить ресурсы

©2023 Azul Systems, Inc.

JVM (сейчас)

- В отсутствии “магического облака”
  - Ограниченные вычислительные ресурсы
  - Ограниченное место для хранения данных
  - Ограничена функционально
  - Ограничена умением анализировать и обучаться
  - Ограниченные “знания” об устройстве мира

©2023 Azul Systems, Inc.
azul

Cloud Native JVM

©2023 Azul Systems, Inc.

Cloud Native JVM могла бы...

- Присоединяться к сообществу других JVM
- Использовать и расчитывать на:
  - Внешние ресурсы
  - Внешнюю функциональность
  - Внешний опыт
- Создавать новые (и пополнять старые) знания

©2023 Azul Systems, Inc.
azul

Cloud Native JVM могла бы…

- Имея доступ к “магическому облаку” получить
  - “безграничные” вычислительные ресурсы
  - “безграничные” размеры хранилища данных
  - “безграничные” аналитические возможности
  - “знания”, “опыт”, …

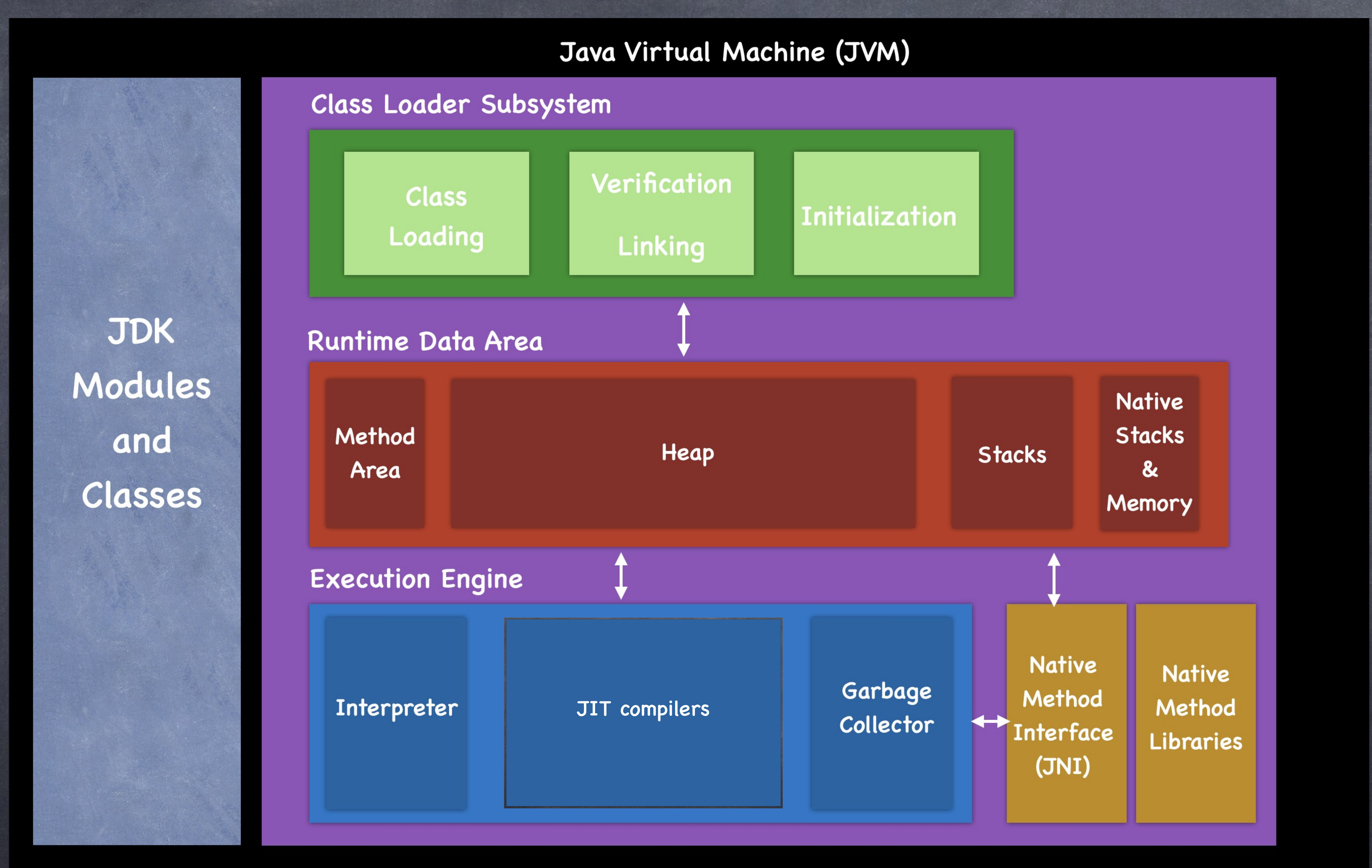
©2023 Azul Systems, Inc.
azul

Cloud Native JVM (определение)

- Обладает всеми свойствами JVM
- Делегирует (аутсорсит) некоторые ключевые функции в облако
- Сообщает в облако опыт и знания, которые могут быть использованы другими JVM, присоединенными к этому облаку

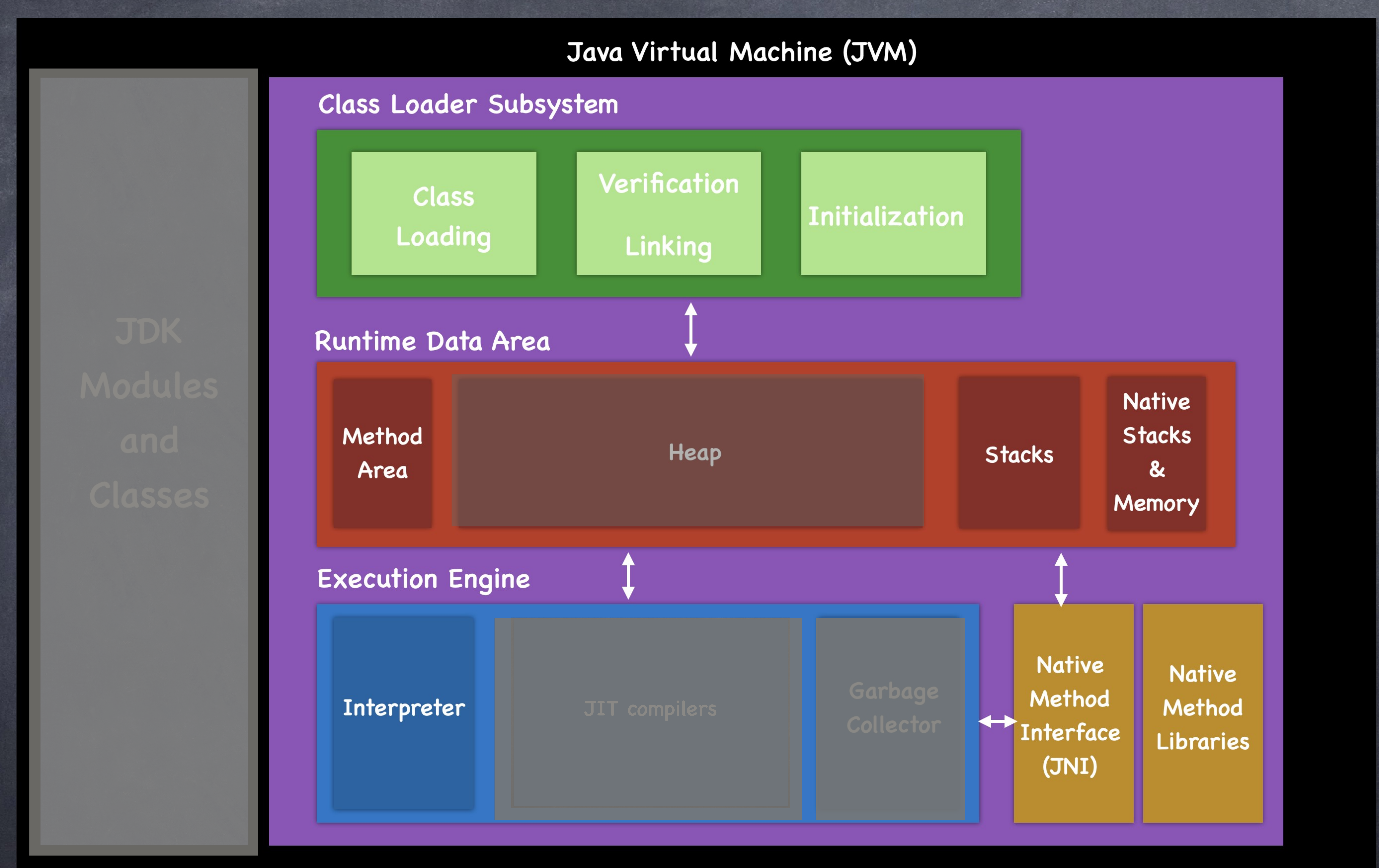
©2023 Azul Systems, Inc.
azul

Java Runtime Environment



©2023 Azul Systems, Inc.
azul

Устройство JVM: Runtime



©2023 Azul Systems, Inc.
azul

JVM Runtime

- Command-Line Argument Processing
- VM Lifecycle
- VM Class Loading, Bytecode Verifier & Format Checker
- Class Data Sharing
- Interpreter
- Compiler Runtime
- Java Exception Handling & VM Fatal Error Handling
- Thread Management & Synchronization
- C++ Heap Management & Java Native Interface (JNI)

https://openjdk.org/groups/hotspot/docs/RuntimeOverview.html

©2023 Azul Systems, Inc.
azul

Знакомство с Runtime’ом

Сотворение вселенной

©2023 Azul Systems, Inc.
azul

Жизненый цикл VM

- Обработка аргументов
- Сотворение вселенной (universe)
- Создание интерпретатора и других подсистем
- Запуск приложения
- Обработка завершения

©2023 Azul Systems, Inc.

Знакомство с Runtime’ом

Classes Lifecycle

©2023 Azul Systems, Inc.
azul

Терминология:
метод *bar* requires класс *Foo*

```
void bar(int param) {
    if (param != 0) {
        Foo a = new Foo(); // Foo must be initialized
        // * * *
    } else {
        // * * *
    }
}
```


©2023 Azul Systems, Inc.

azul

```
class Foo {
  static final LocalDateTime baz = LocalDateTime.now();
}
```

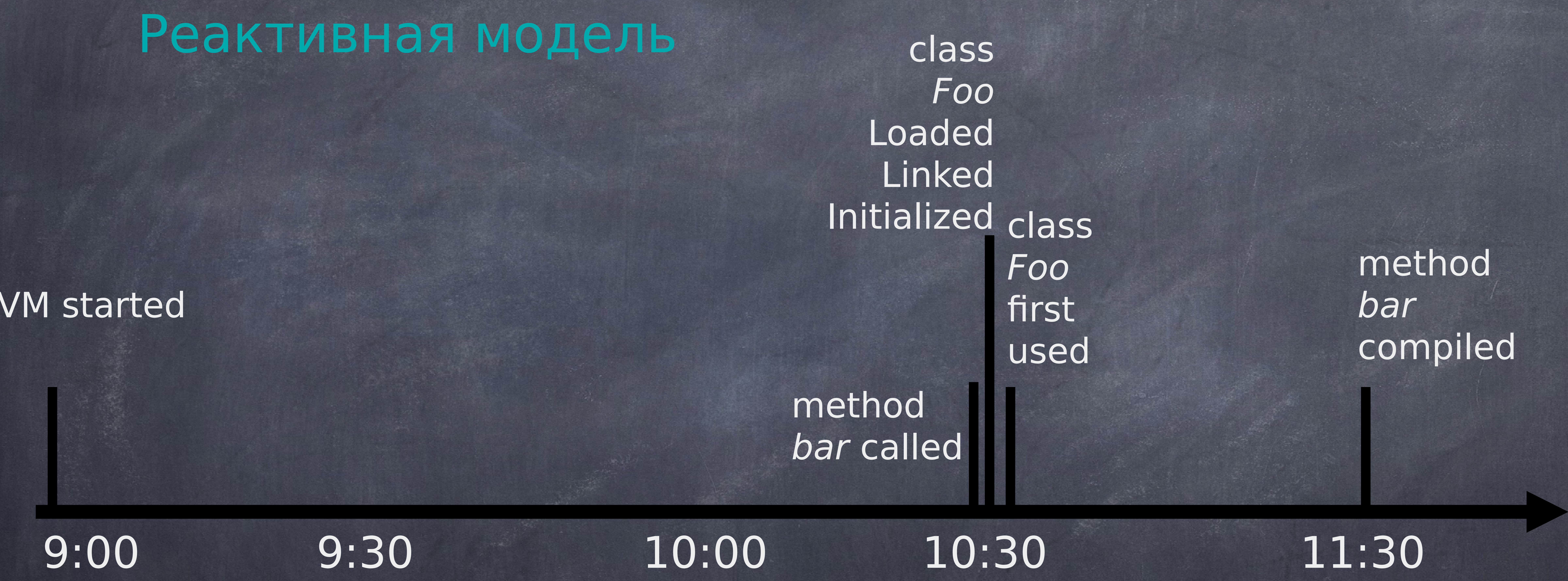
```
class Foo {
  static final java.time.LocalDateTime baz;

  Foo();
    Code:
       0: aload_0
       1: invokespecial #1  // Method java.lang.Object."<init>":()V
       4: return

  static {};
    Code:
       0: invokestatic  #2  // Method j.time.LocalDateTime.now:()Ljava/time/LocalDateTime;
       3: putstatic     #3  // Field baz:Ljava/time/LocalDateTime;
       6: return
}
```


©2023 Azul Systems, Inc.

azul

Жизненый цикл класса Foo и метода bar

Реактивная модель



©2023 Azul Systems, Inc.
azul

Жизненый цикл классов

```
class Foo {
  static final LocalDateTime baz = LocalDateTime.now();
}
```

- VM Class
  - ✓ Load (ClassLoader)
  - ✓ Link (Resolve symbols)
  - ✓ Initialize (static initializer <clinit>)
  - ✓ Unload
- Bytecode Verifier & Format Checker
- Class Data Sharing (JEP 310)

©2023 Azul Systems, Inc.

Bytecode Verifier

https://www.youtube.com/watch?v=m16AIz1fIFI

©2023 Azul Systems, Inc.
azul

Class Data Sharing (JEP 310)

https://www.youtube.com/watch?v=fqUG1rr-y78

©2023 Azul Systems, Inc.
azul

Знакомство с Runtime’ом

Compiler Runtime

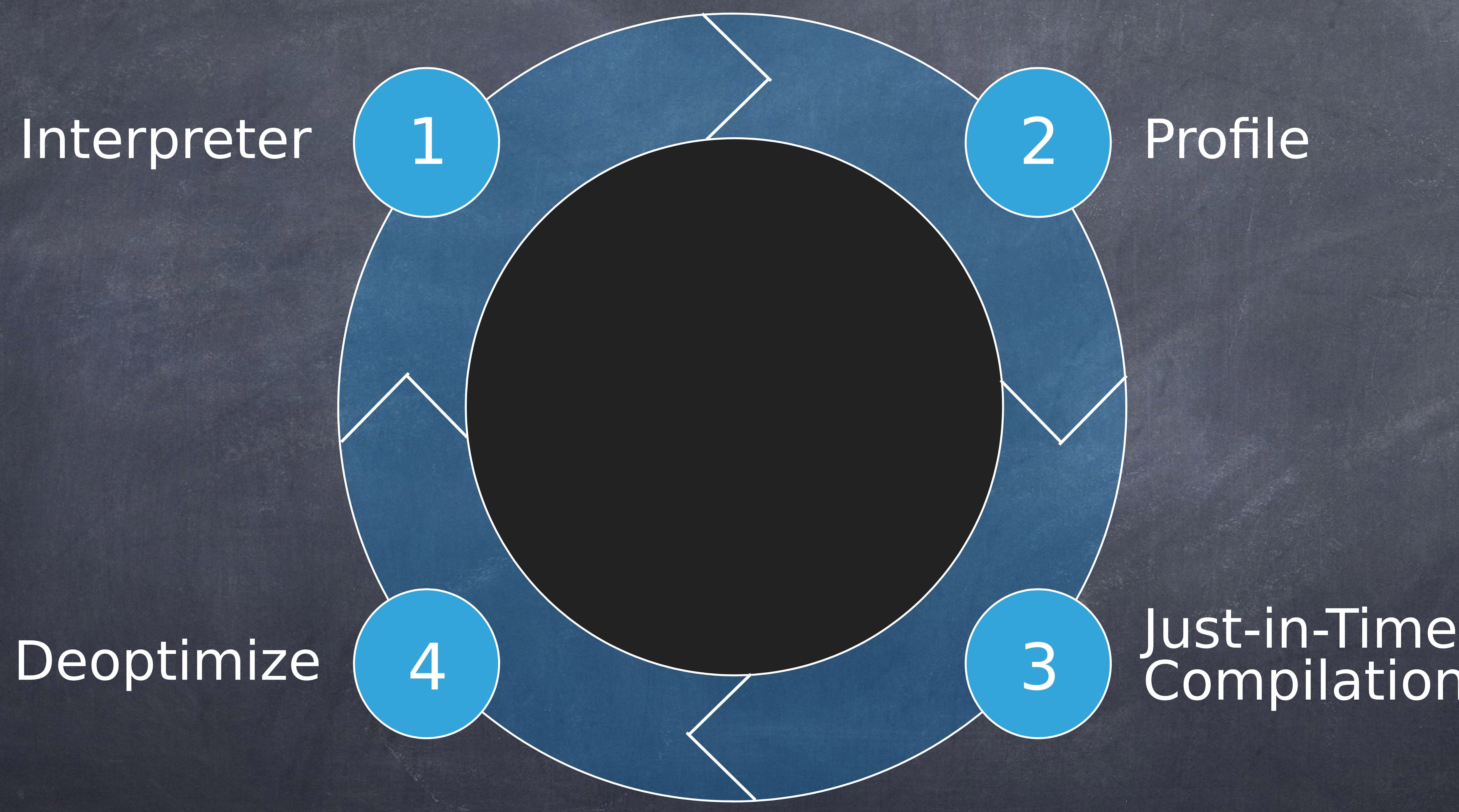
©2023 Azul Systems, Inc.
azul

Compiler Runtime

- Compilation Policy
- Profiling System
- Install/Uninstall JIT-methods
- CodeCache
- De-optimization & code-profile healing

©2023 Azul Systems, Inc.

Compiler Runtime



©2023 Azul Systems, Inc.
azul

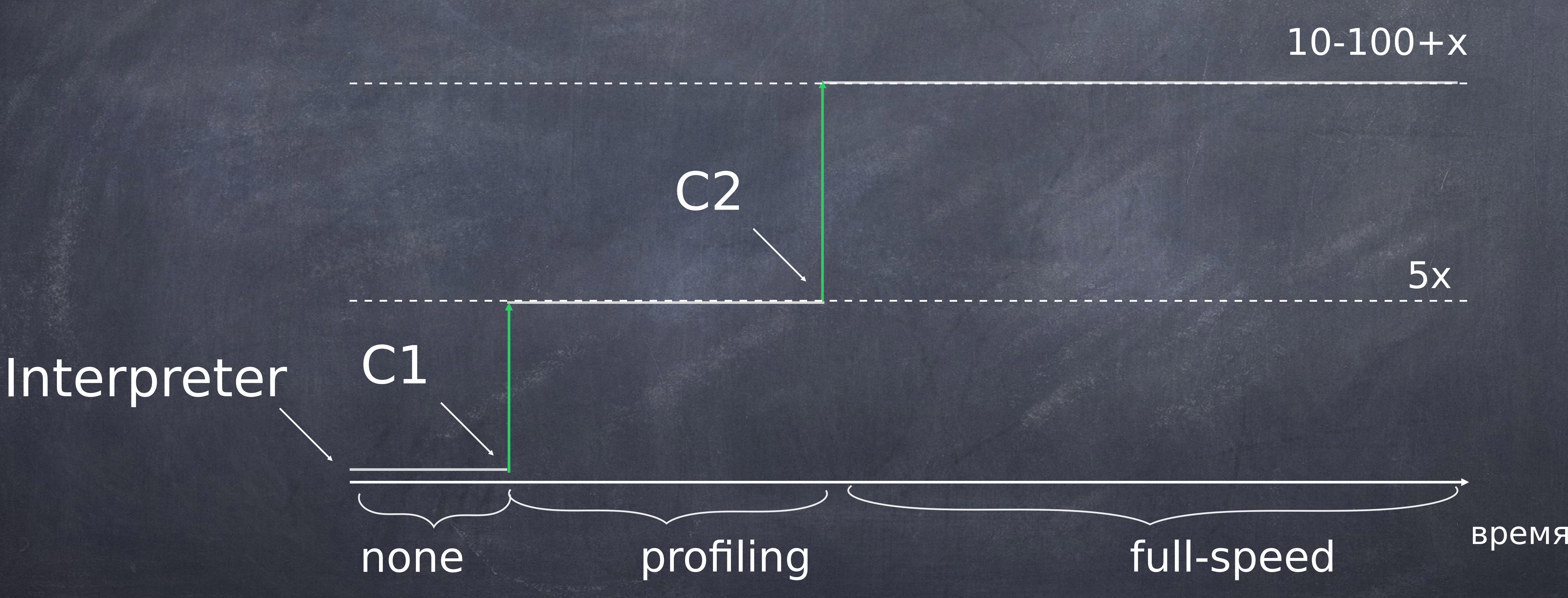
JIT Компиляторы

Стараются перевести байткод в оптимальное машинное представление

JIT compilers

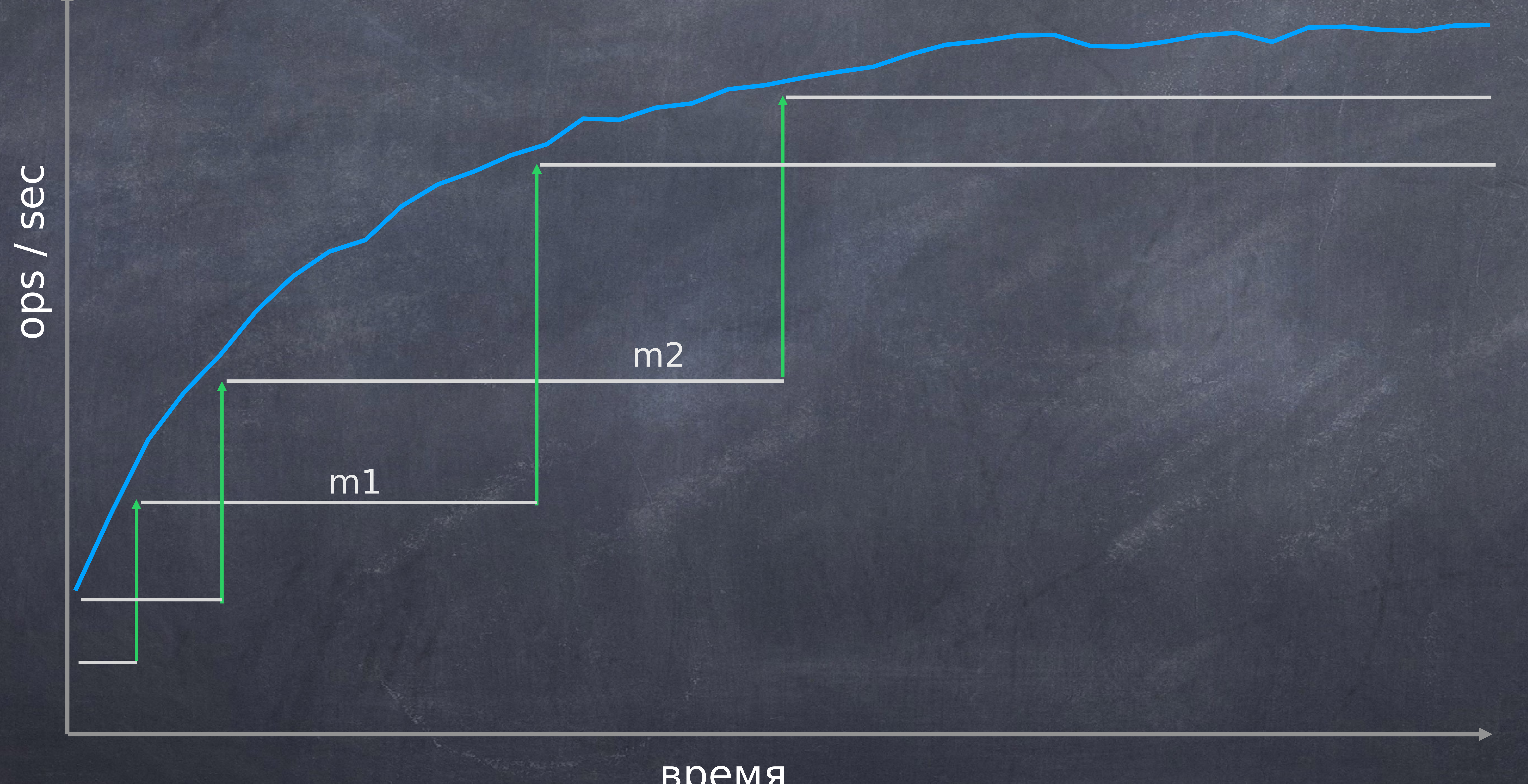
©2023 Azul Systems, Inc.
azul

Реактивная модель жизни одного метода



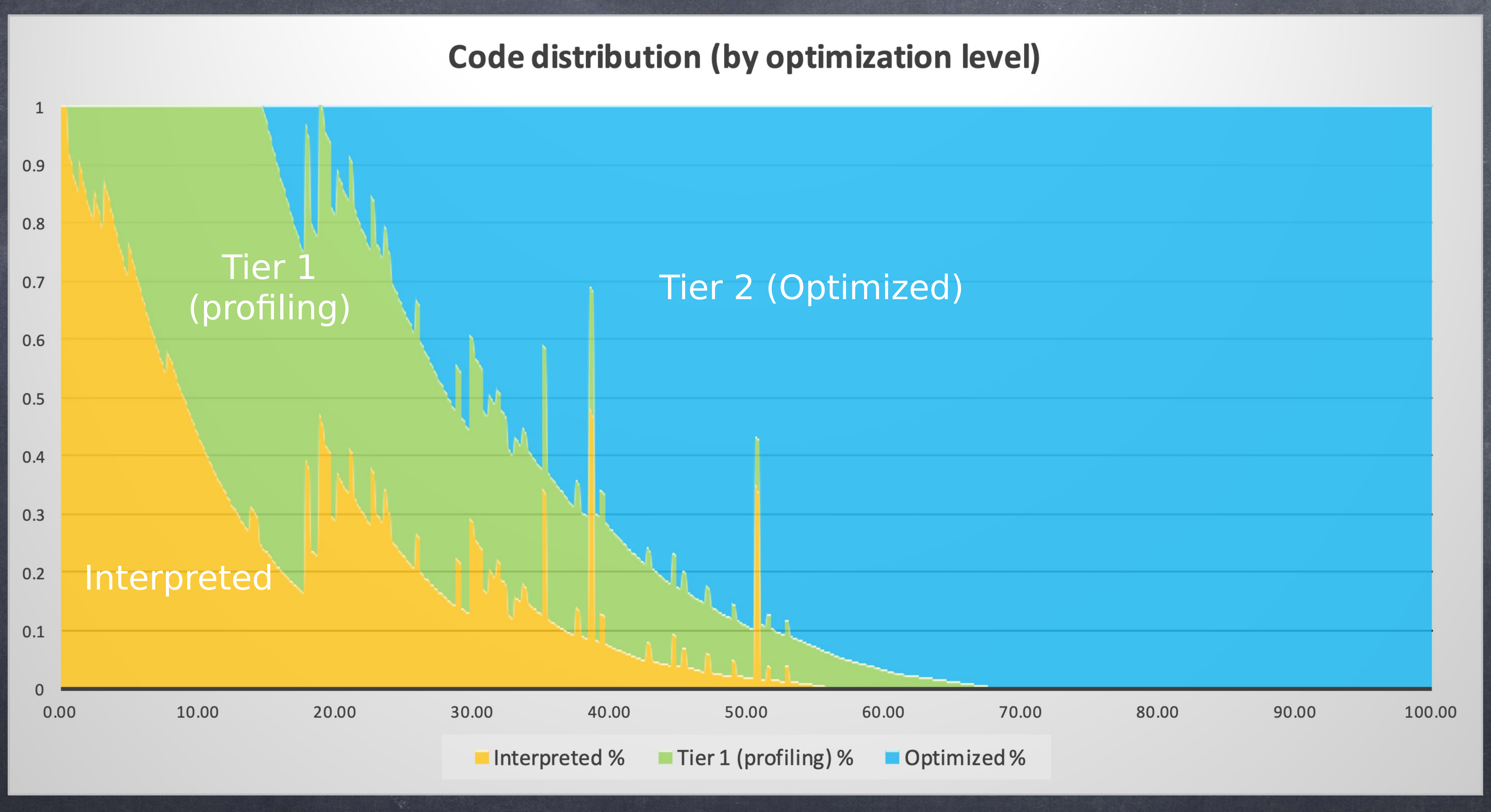
©2023 Azul Systems, Inc.
azul

Реактивная модель прогрева: warmup приложения



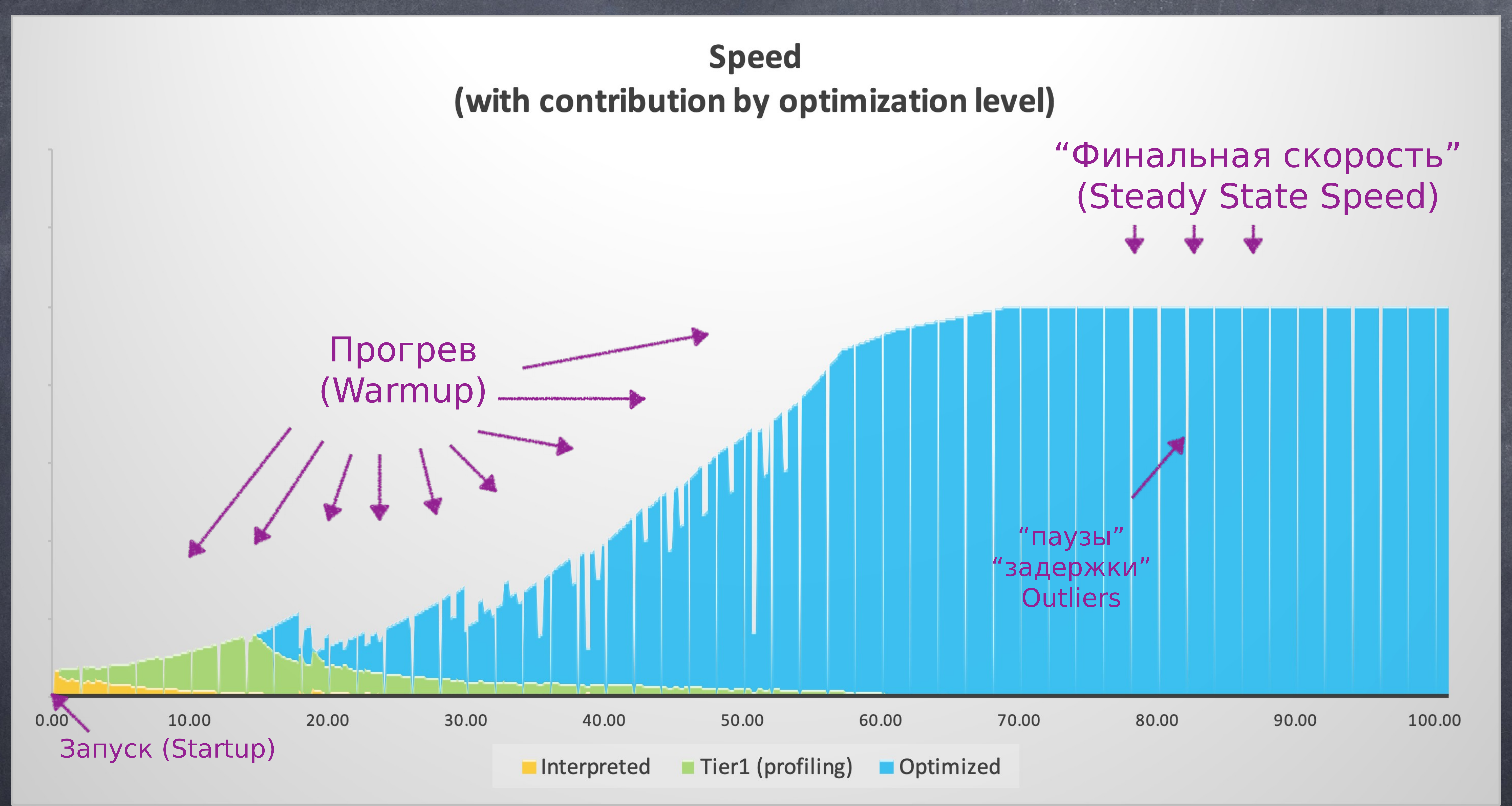
©2023 Azul Systems, Inc.
azul

Внутренности JVM: Распределение кода



©2023 Azul Systems, Inc.
azul

Внутренности JVM:
Фазы работы приложения



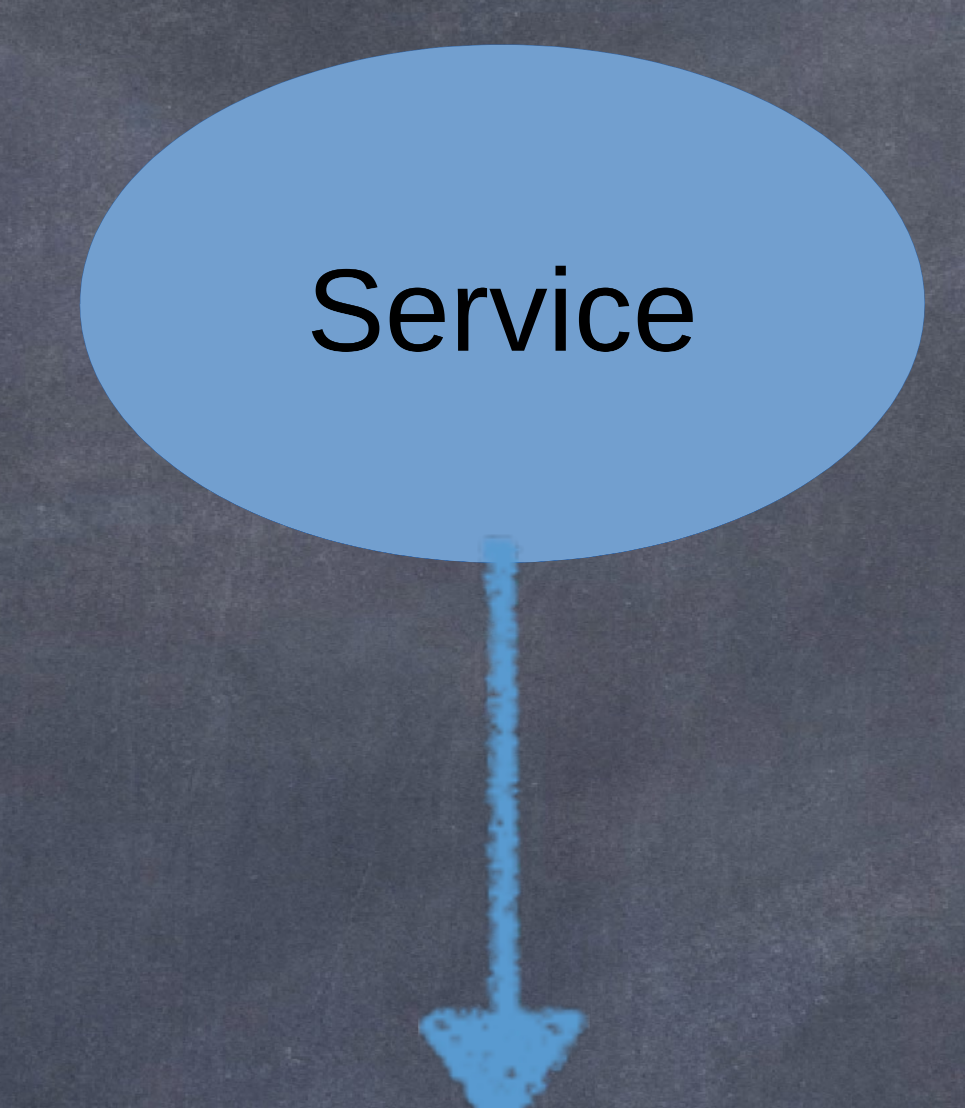
©2023 Azul Systems, Inc.
azul

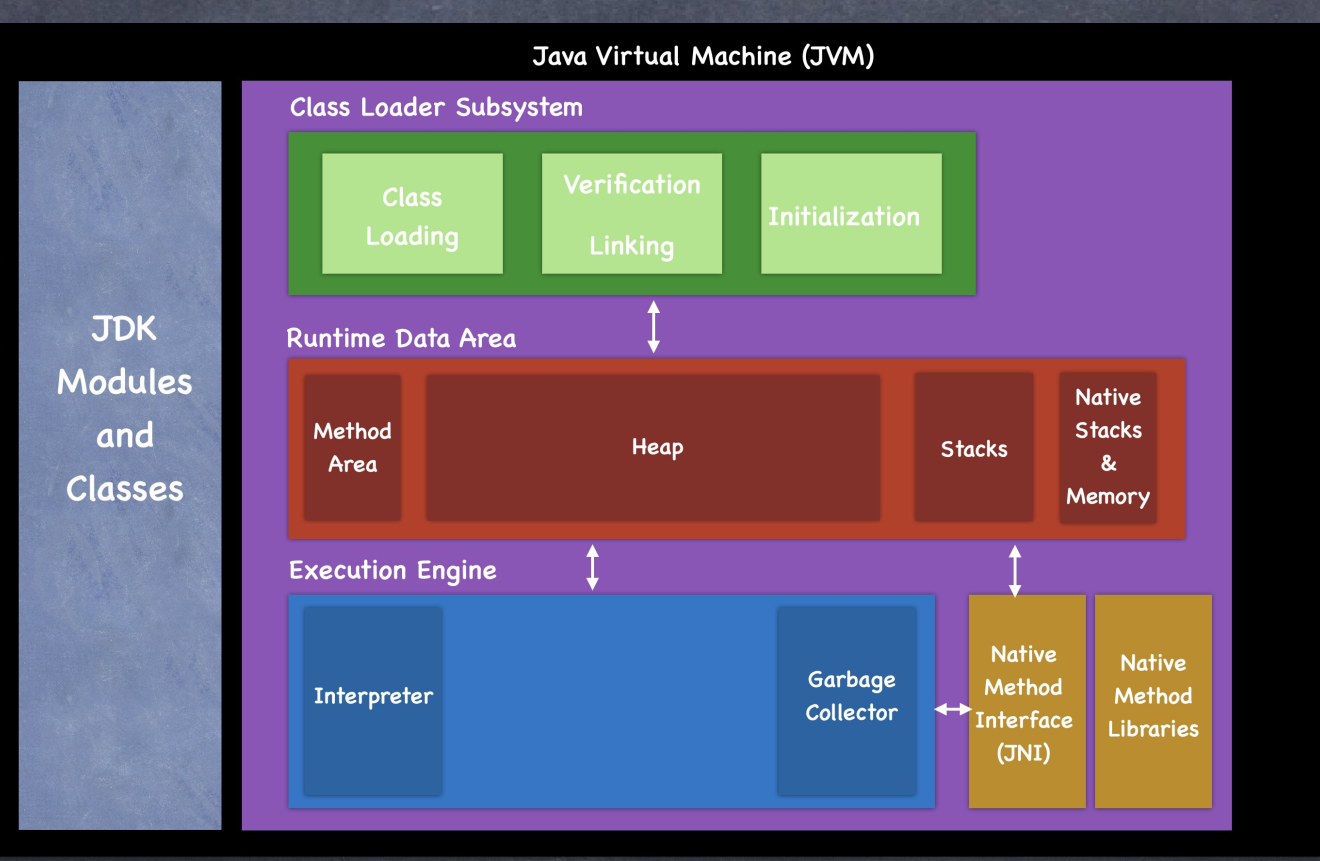
Знакомство с Runtime’ом

Cloud Native Application
warmup

©2023 Azul Systems, Inc.
azul

Cloud Natve Application

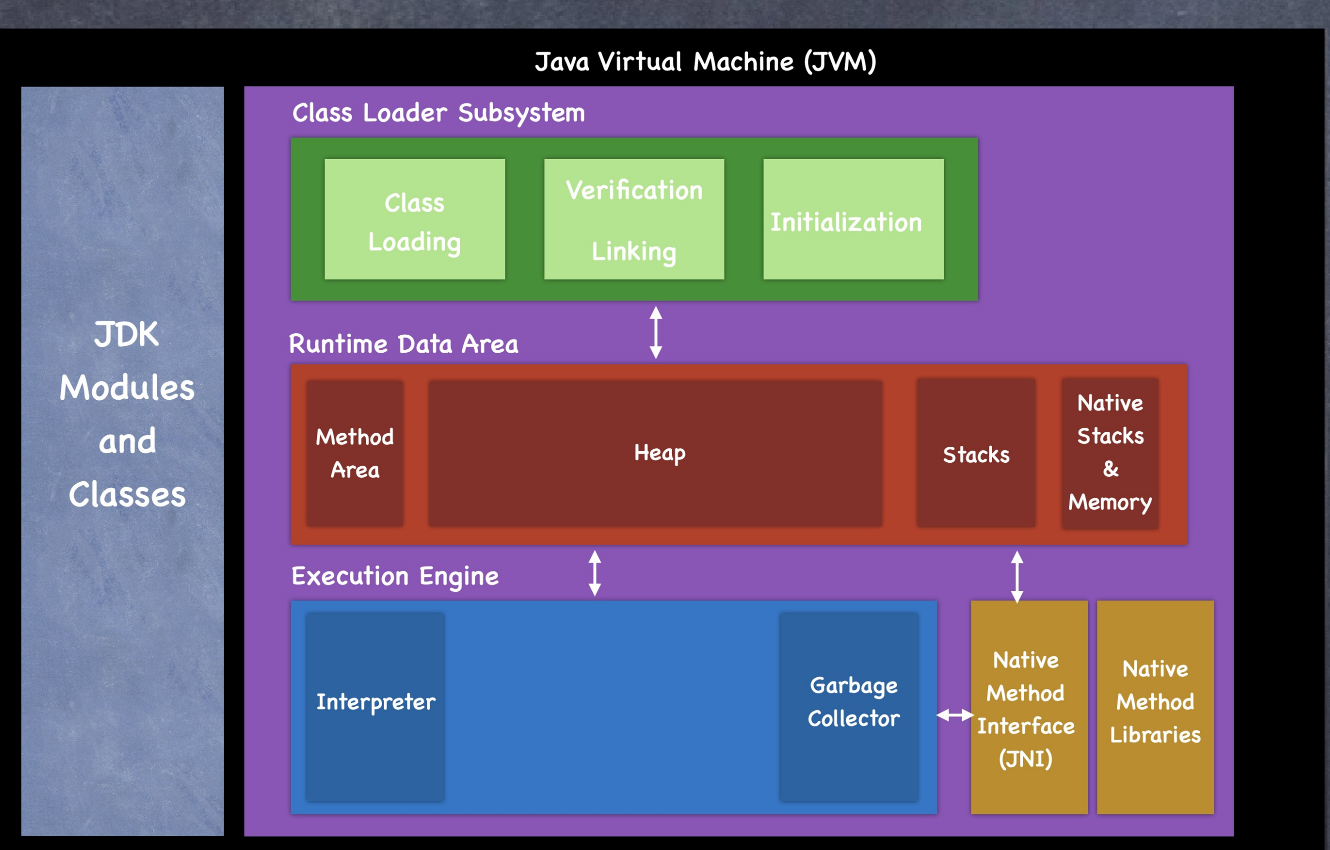


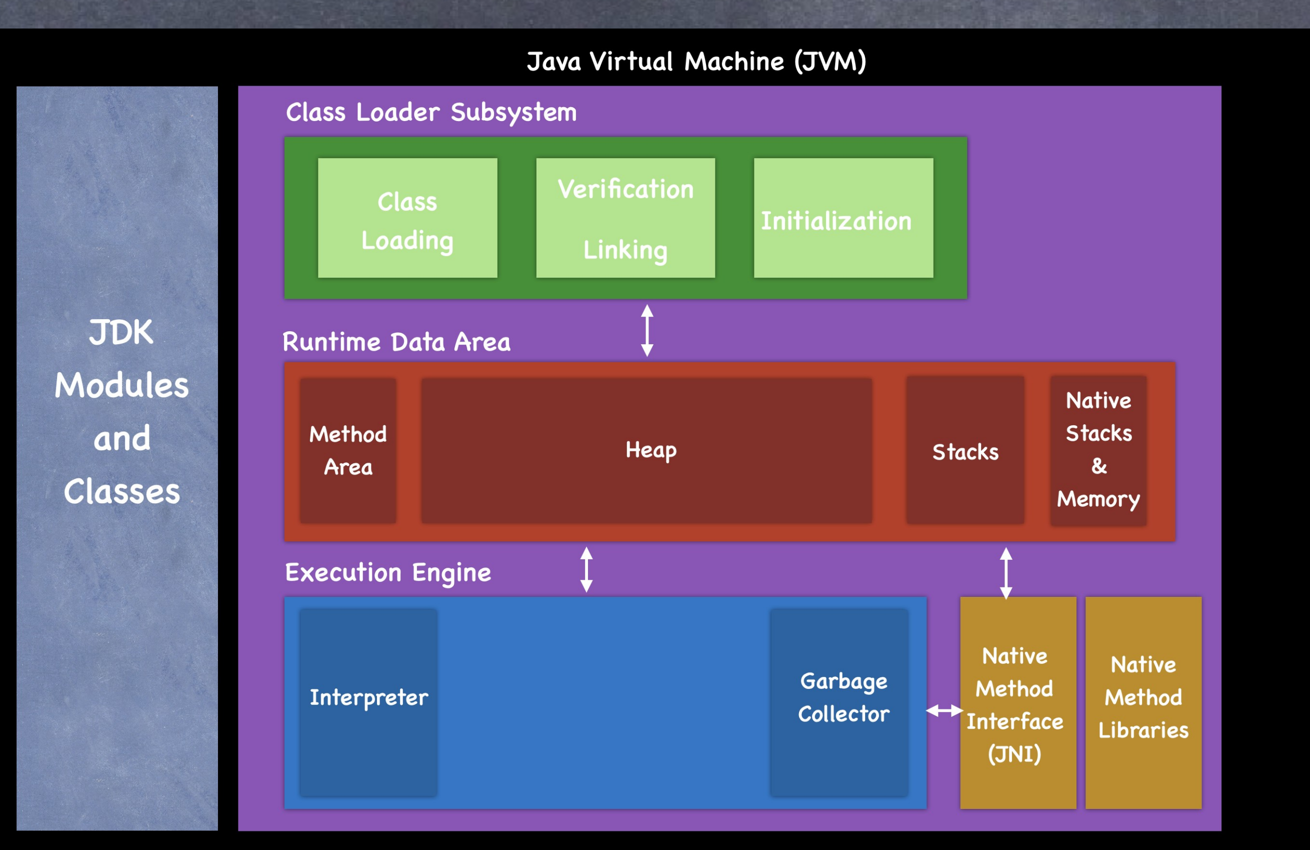


©2023 Azul Systems, Inc.
azul

Scale Up

Service

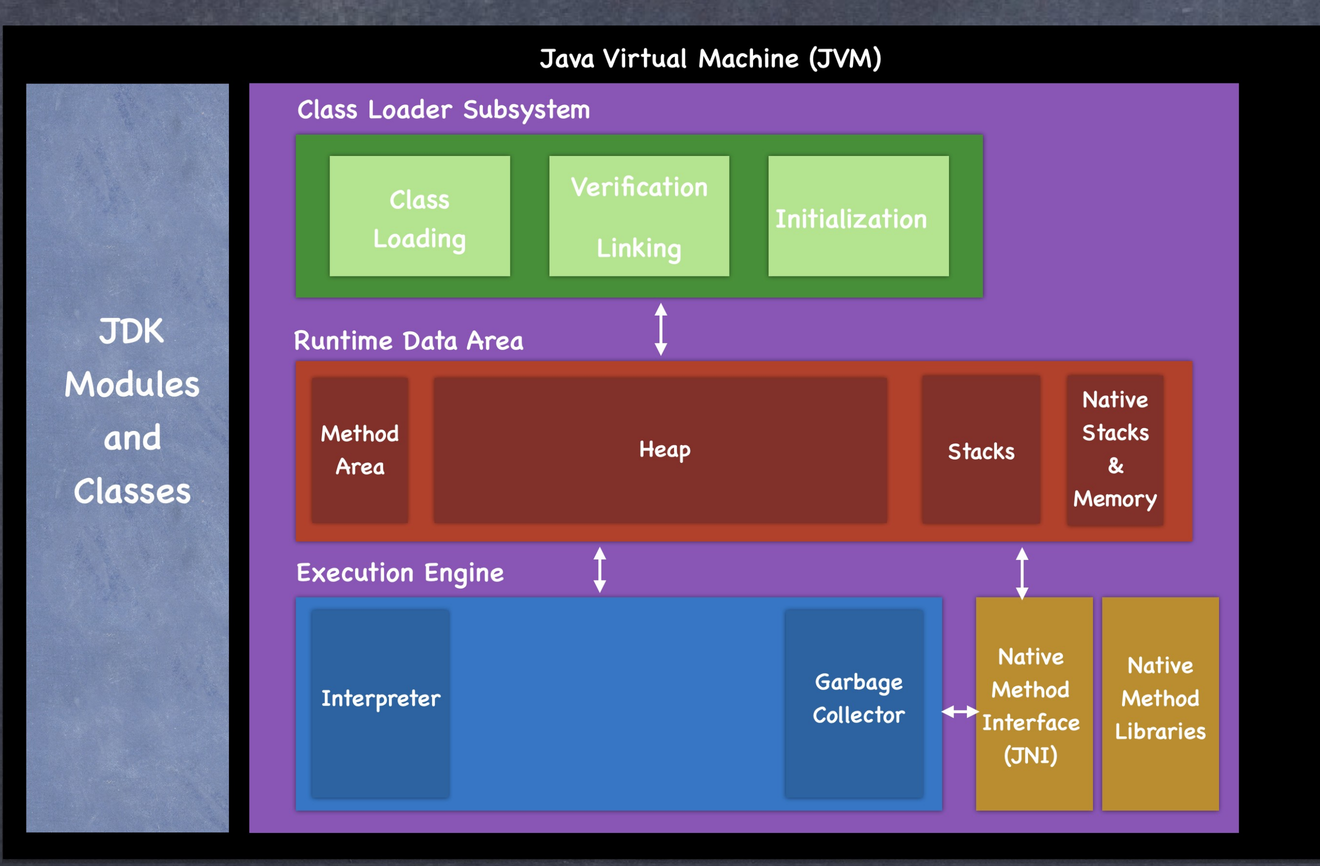


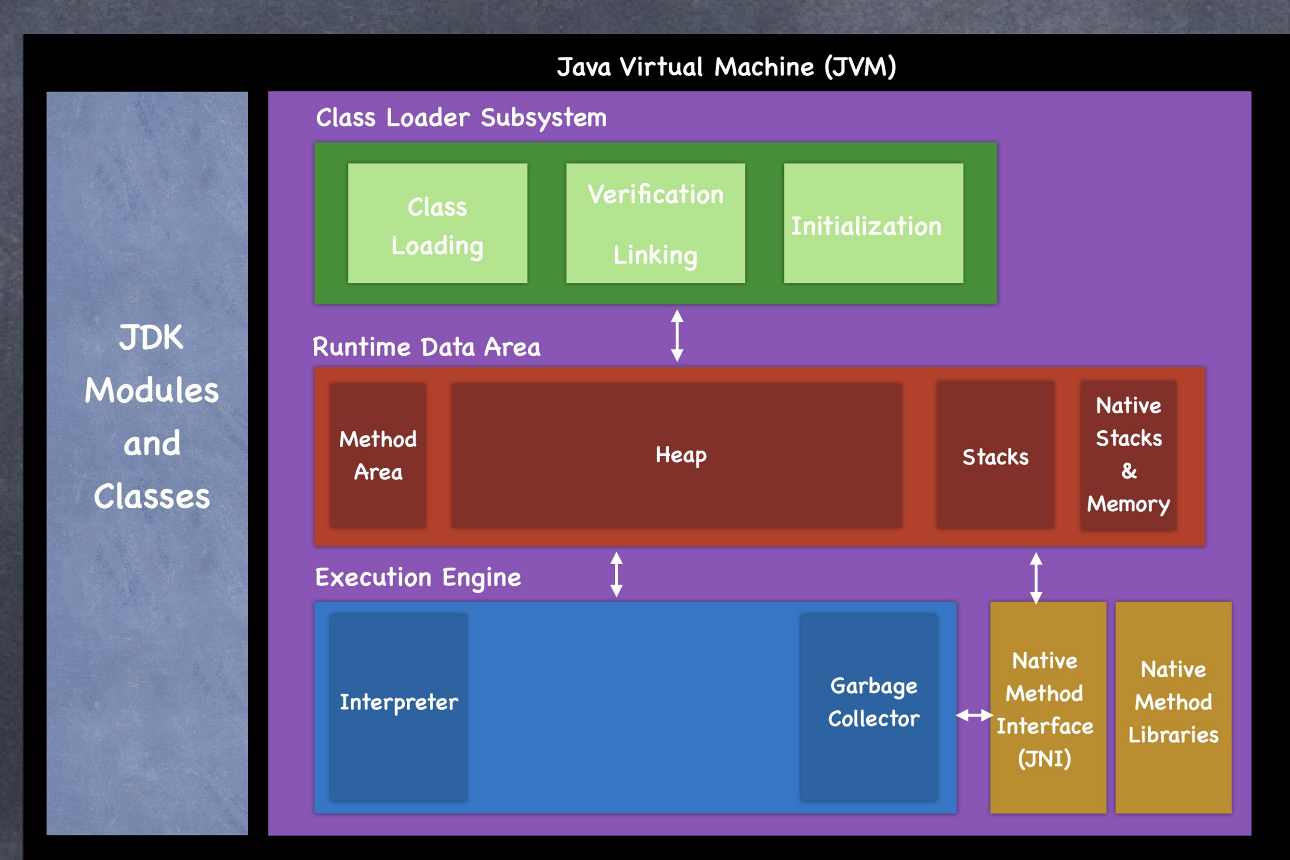


©2023 Azul Systems, Inc.

azul

Readiness Check: Failed

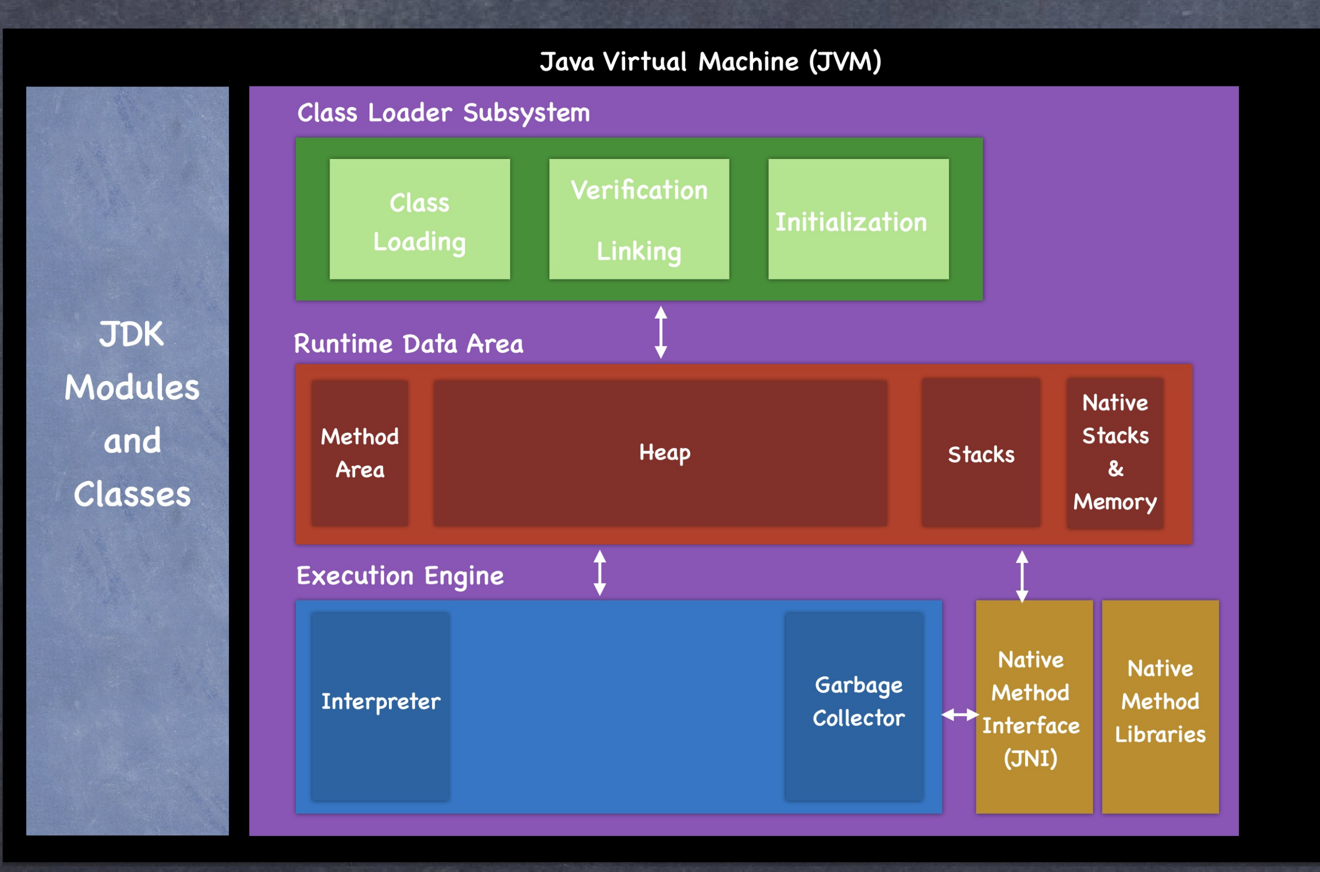


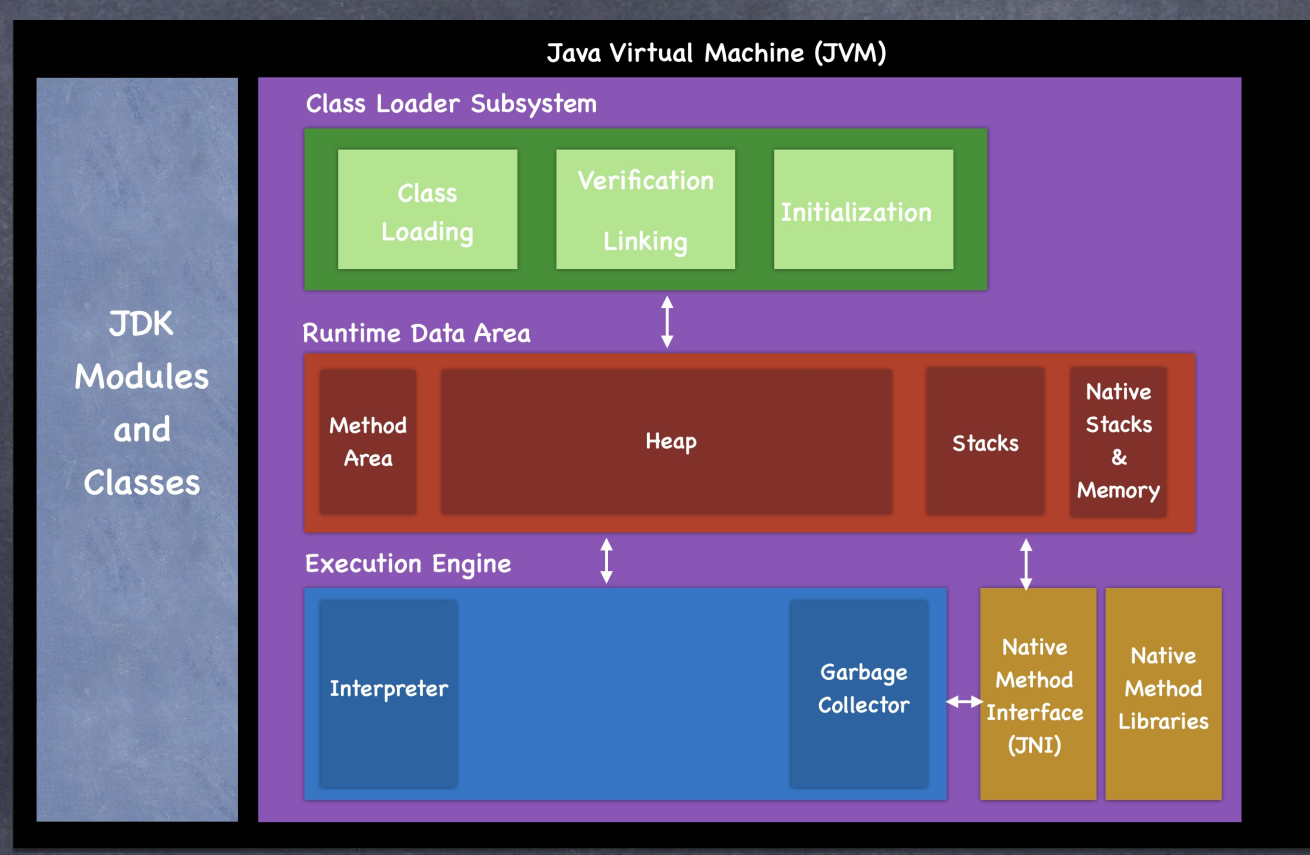


Готов? Нет…
Готов? Еще нет…
Готов? Нет еще…
Готов? Погодите…
Готов? ...

©2023 Azul Systems, Inc.
azul

Readiness Check: Passed

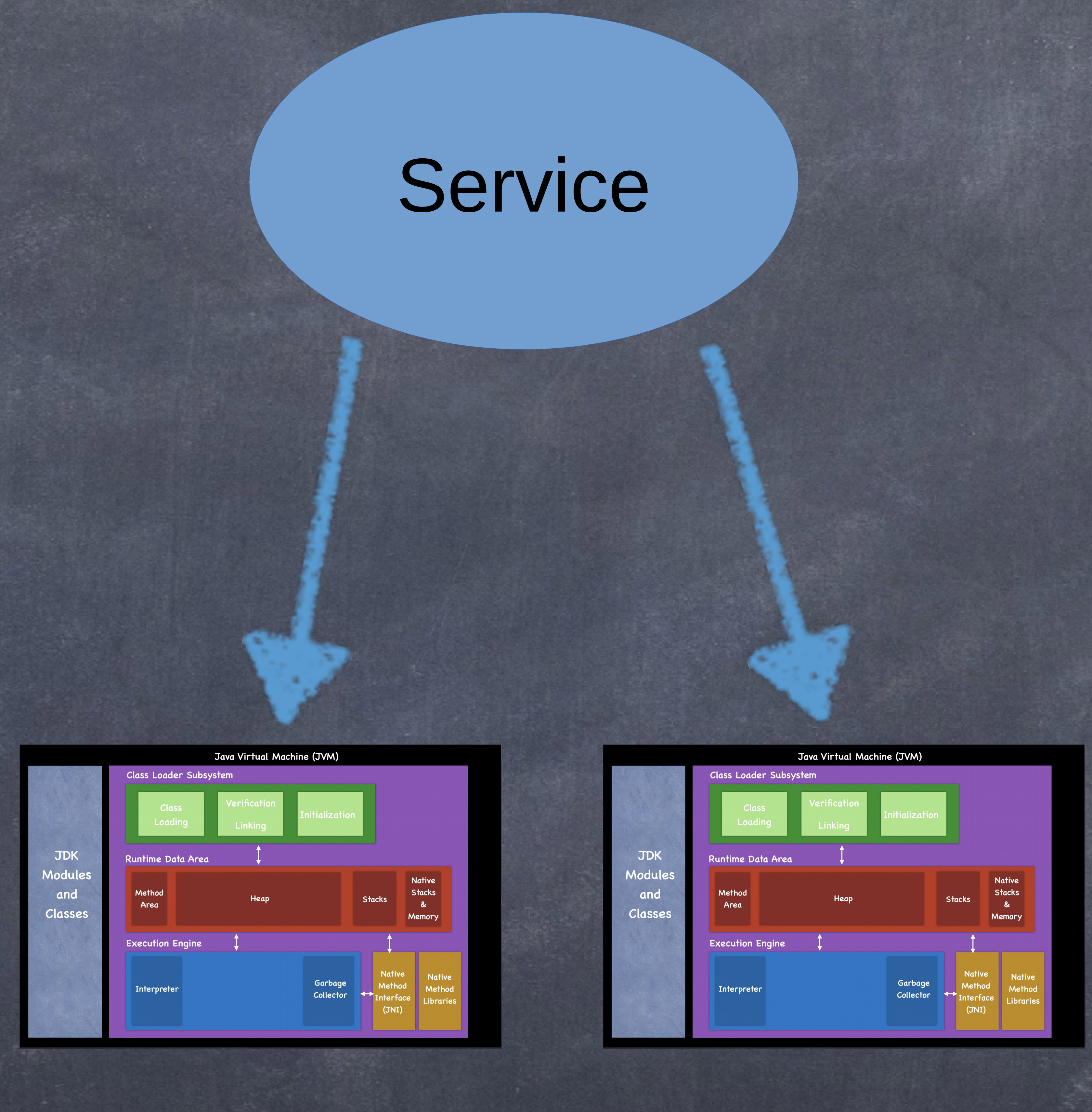




Готов? Нет…
Готов? Еще нет…
Готов? Нет еще…
Готов? Погодите…
Готов? …
…
Готов? Да!

©2023 Azul Systems, Inc.
azul

Перенаправление трафика



Готов? **Да**

©2023 Azul Systems, Inc.
azul

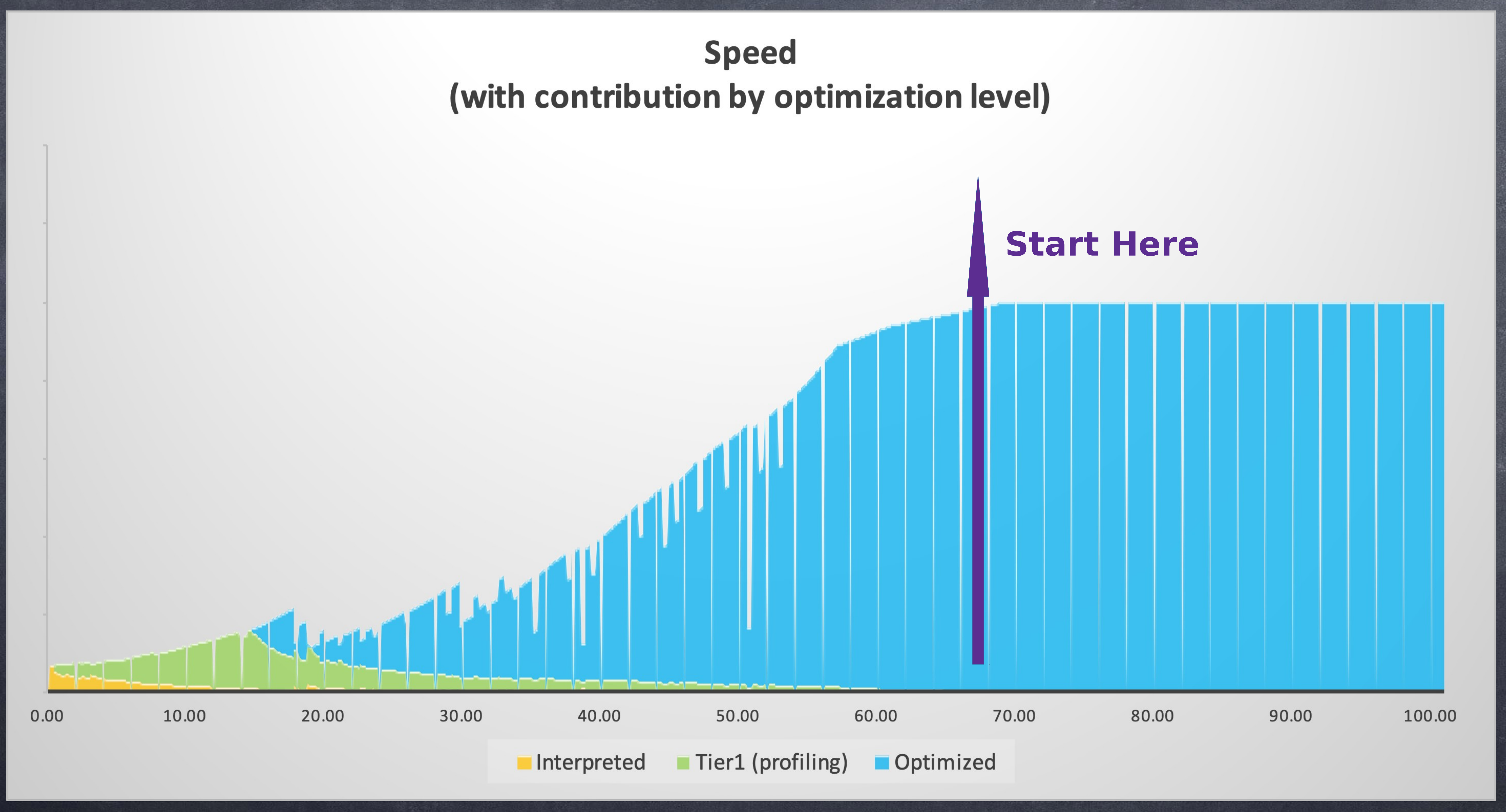
Итого:
Реактивная модель Поведения Java Runtime

©2023 Azul Systems, Inc.
azul

Можно сразу в Steady State?

©2023 Azul Systems, Inc.
azul

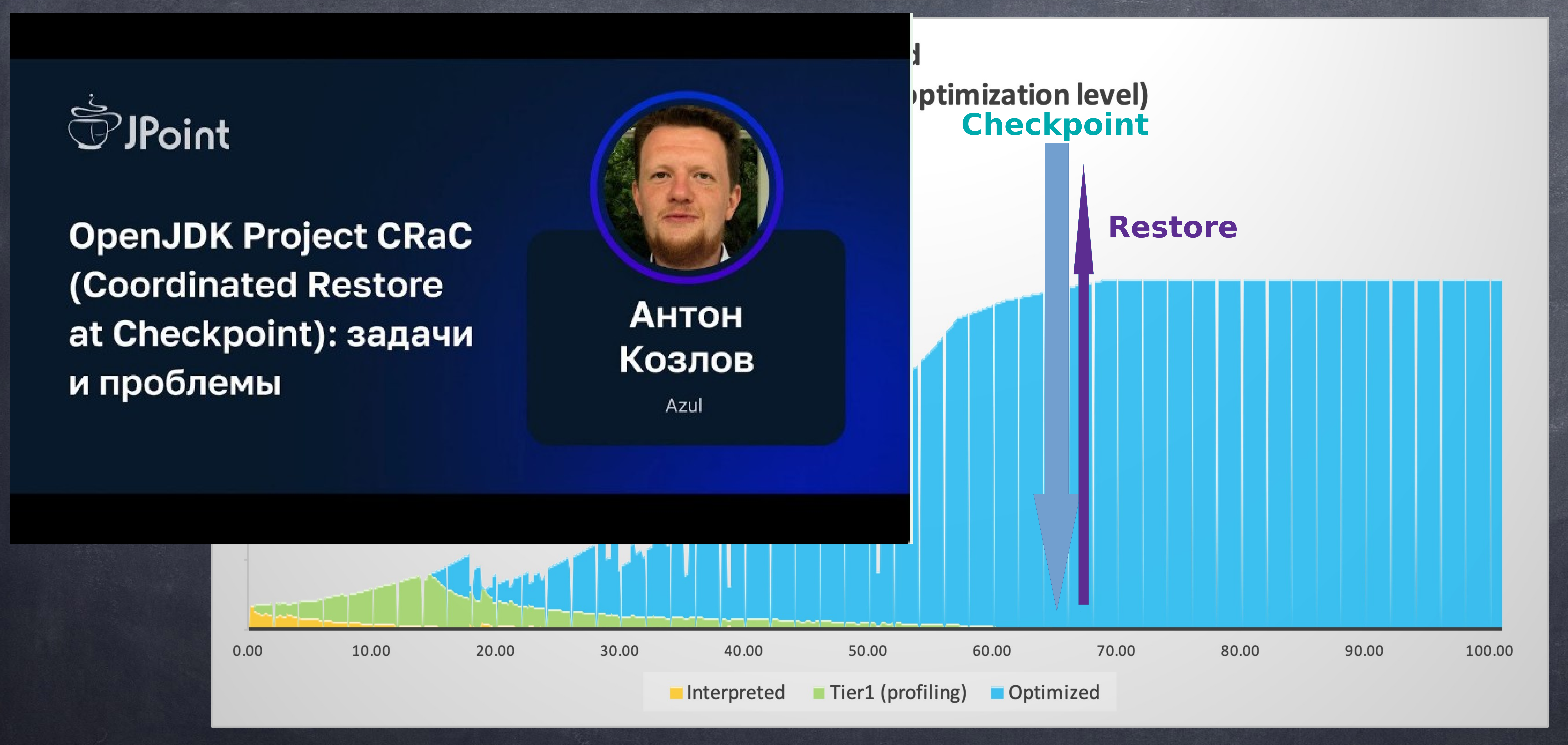
А можно сразу Steady State?



azul

CRaC – Coordinated Restore at Checkpoint

https://openjdk.org/projects/crac/



https://www.youtube.com/watch?v=RLFQj2mPqUM

azul

Реактивная модель
Vs
Проактивная модель

©2023 Azul Systems, Inc.
azul

Терминология:
метод *bar* requires класс *Foo*

```
void bar(int param) {
    if (param != 0) {
        Foo a = new Foo(); // Foo must be initialized
        // * * *
    } else {
        // * * *
    }
}
```


©2023 Azul Systems, Inc.
azul

Инициализация классов (JLS)

JLS for Java 8. 12.4.1. When Initialization Occurs

A class or interface type T will be initialized **immediately before** the first occurrence of any one of the following:

T is a class and an instance of T is created.

A static method declared by T is invoked.

A static field declared by T is assigned.

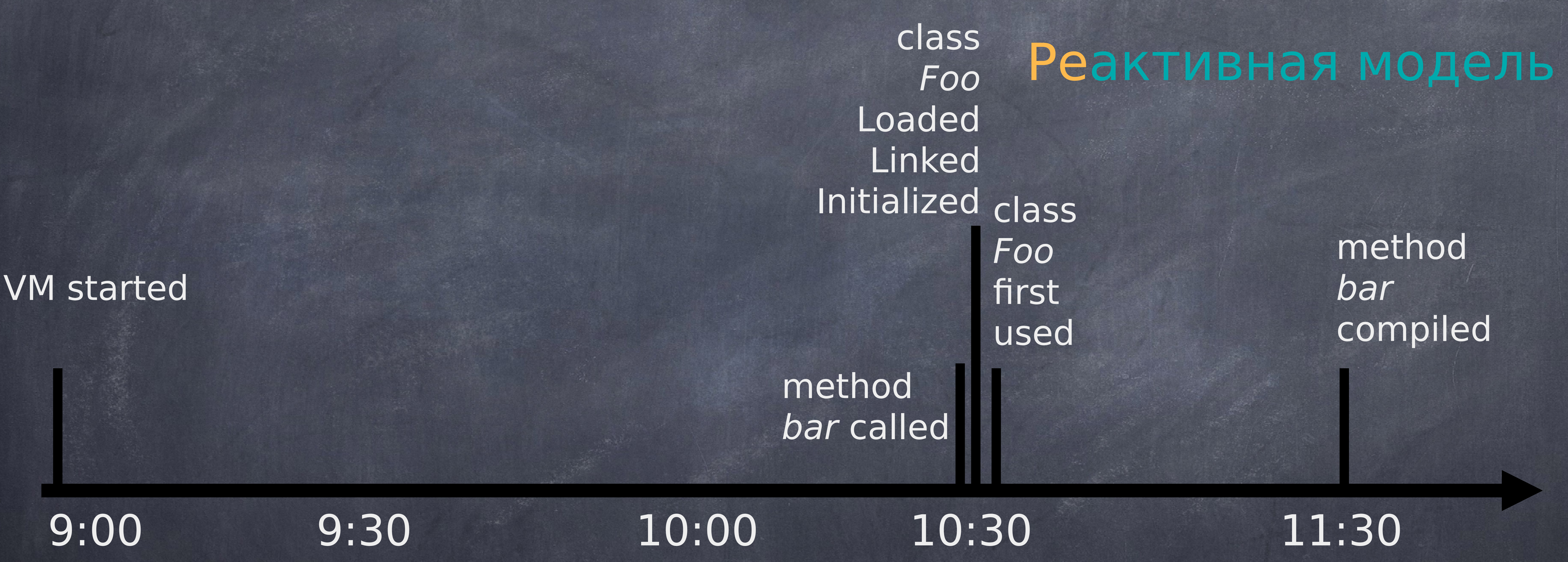
A static field declared by T is used and the field is not a constant variable (§4.12.4).

T is a top level class (§7.6) and an assert statement (§14.10) lexically nested within T (§8.1.3) is executed.

https://docs.oracle.com/javase/specs/jls/se8/html/jls-12.html#jls-12.4.1

©2023 Azul Systems, Inc.
azul

Жизненый цикл класса Foo и метода bar



©2023 Azul Systems, Inc.
azul

Инициализация классов (JVMS)

JVMS for Java 8. Section 5.5: Initialization

Initialization of a class or interface consists of executing its class or interface initialization method (§2.9). A class or interface C may be initialized only as a result of:

- The execution of any one of the Java Virtual Machine instructions new, getstatic, putstatic, or invokestatic that references C (§new, §getstatic, §putstatic, §invokestatic). These instructions reference a class or interface directly or indirectly through either a field reference or a method reference

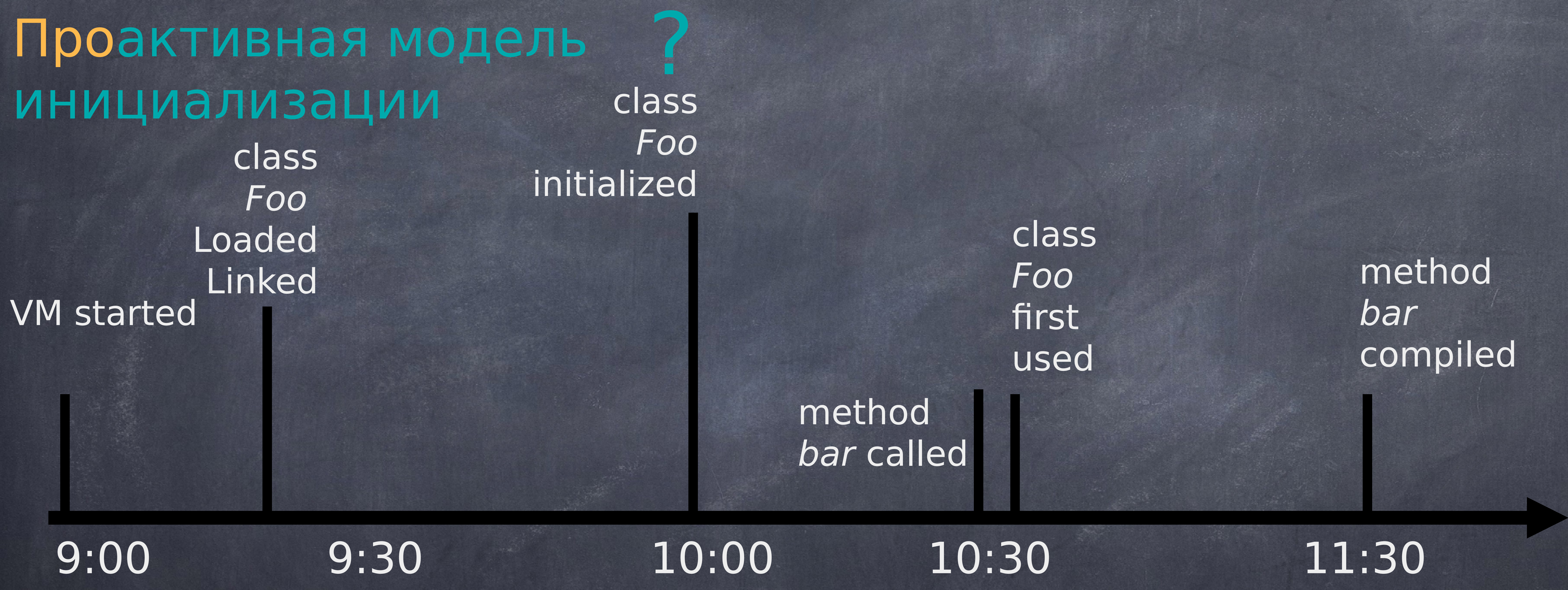
…..

https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-5.html#jvms-5.5

©2023 Azul Systems, Inc.

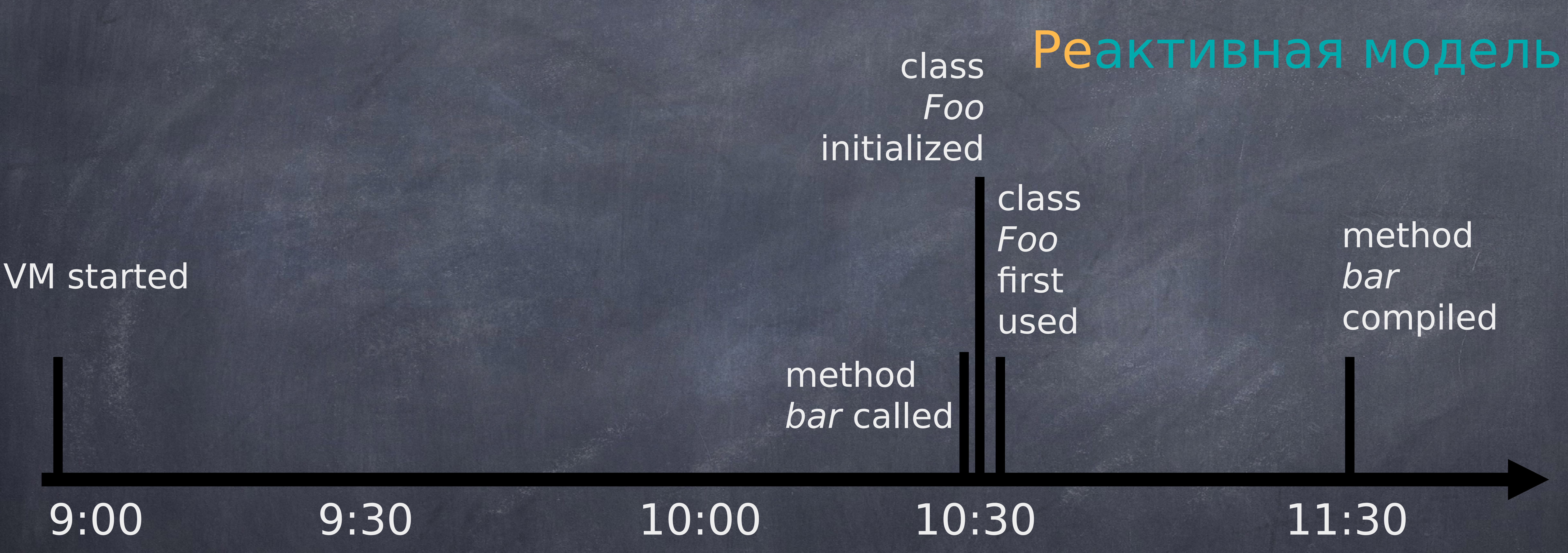
azul

Жизненый цикл класса Foo



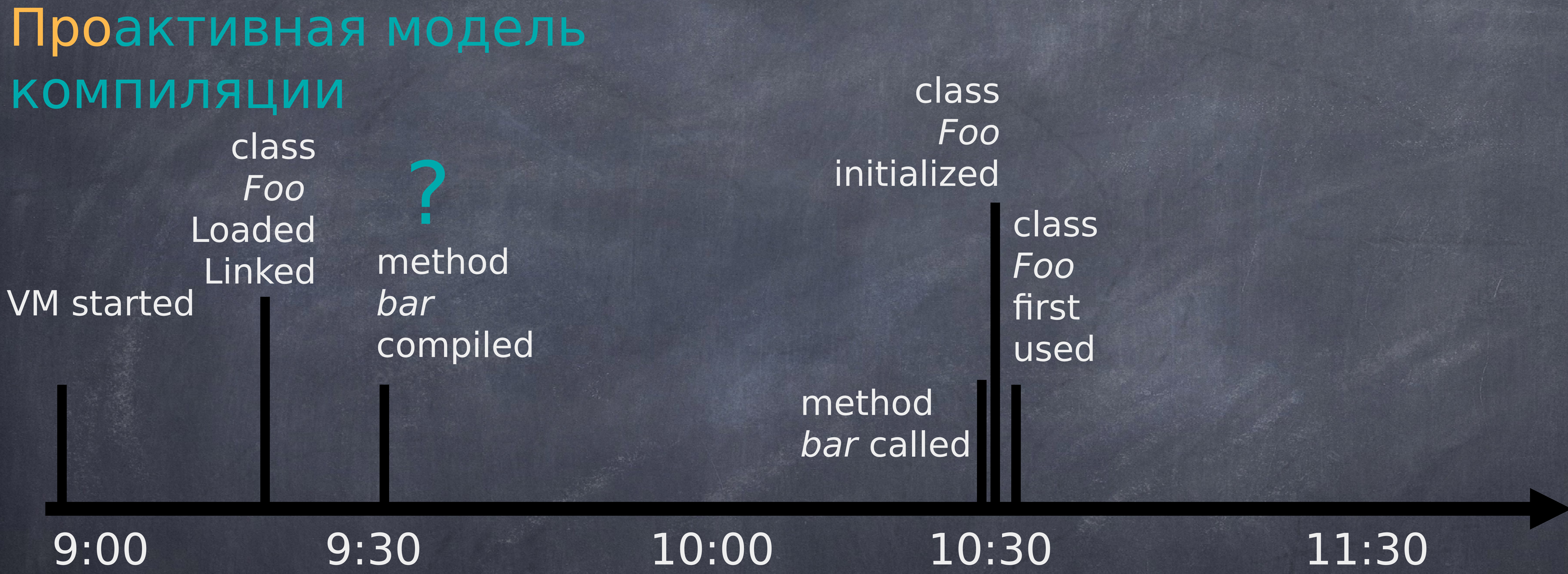
©2023 Azul Systems, Inc.
azul

Жизненый цикл класса Foo и метода bar



©2023 Azul Systems, Inc.
azul

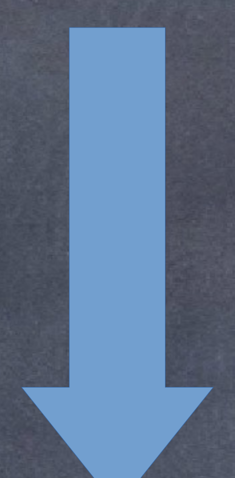
Жизненый цикл метода bar



©2023 Azul Systems, Inc.
azul

Пример: Проверка инициализации класса

```
Foo a = new Foo()
```

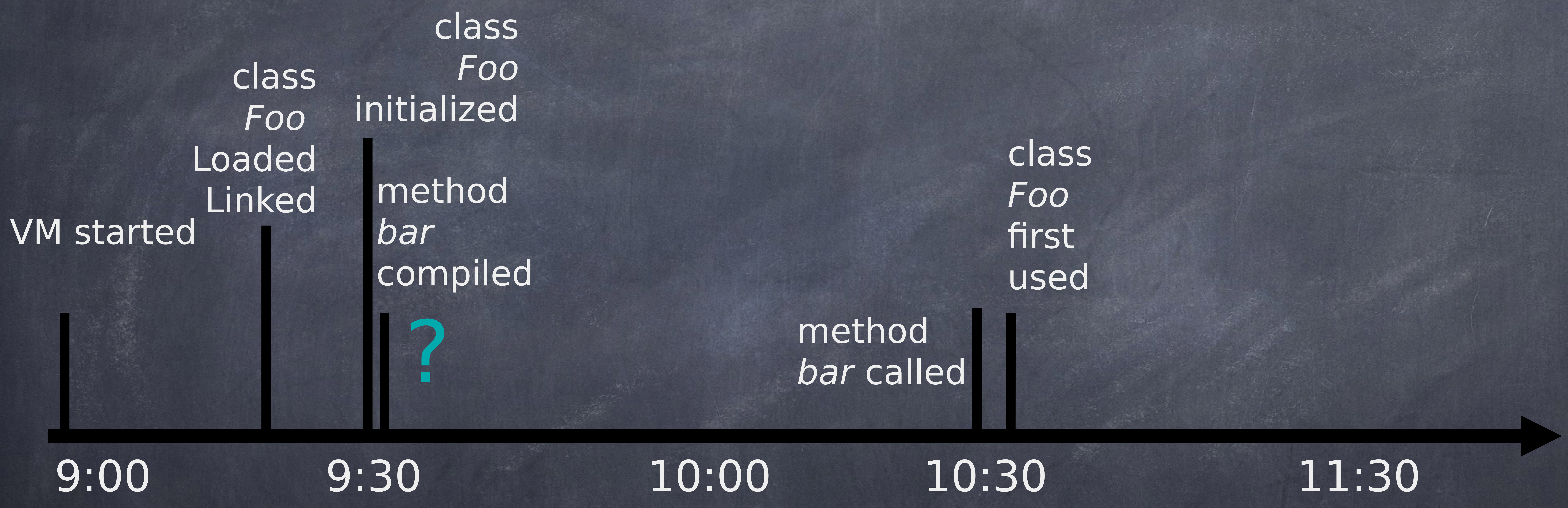


```
if (!vm.is_inititialized(Foo)) {
  vm.init(Foo); // or uncommon_trap(uninitialized)
}
Foo a = new Foo()
```

©2023 Azul Systems, Inc.

Жизненый цикл класса Foo и метода bar

Проактивная модель

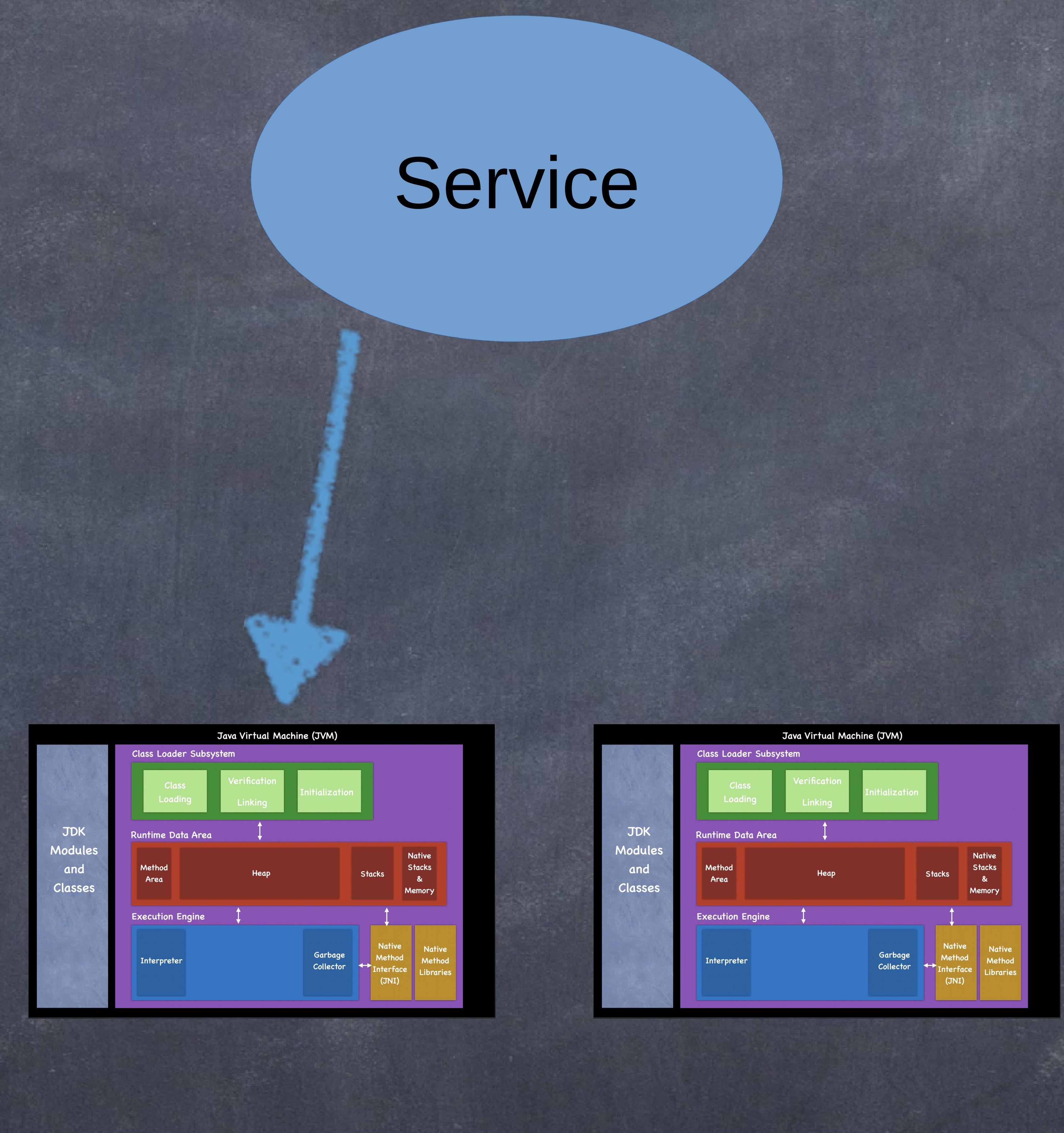


©2023 Azul Systems, Inc.
azul

Что может дать проактивная модель?

©2023 Azul Systems, Inc.
azul

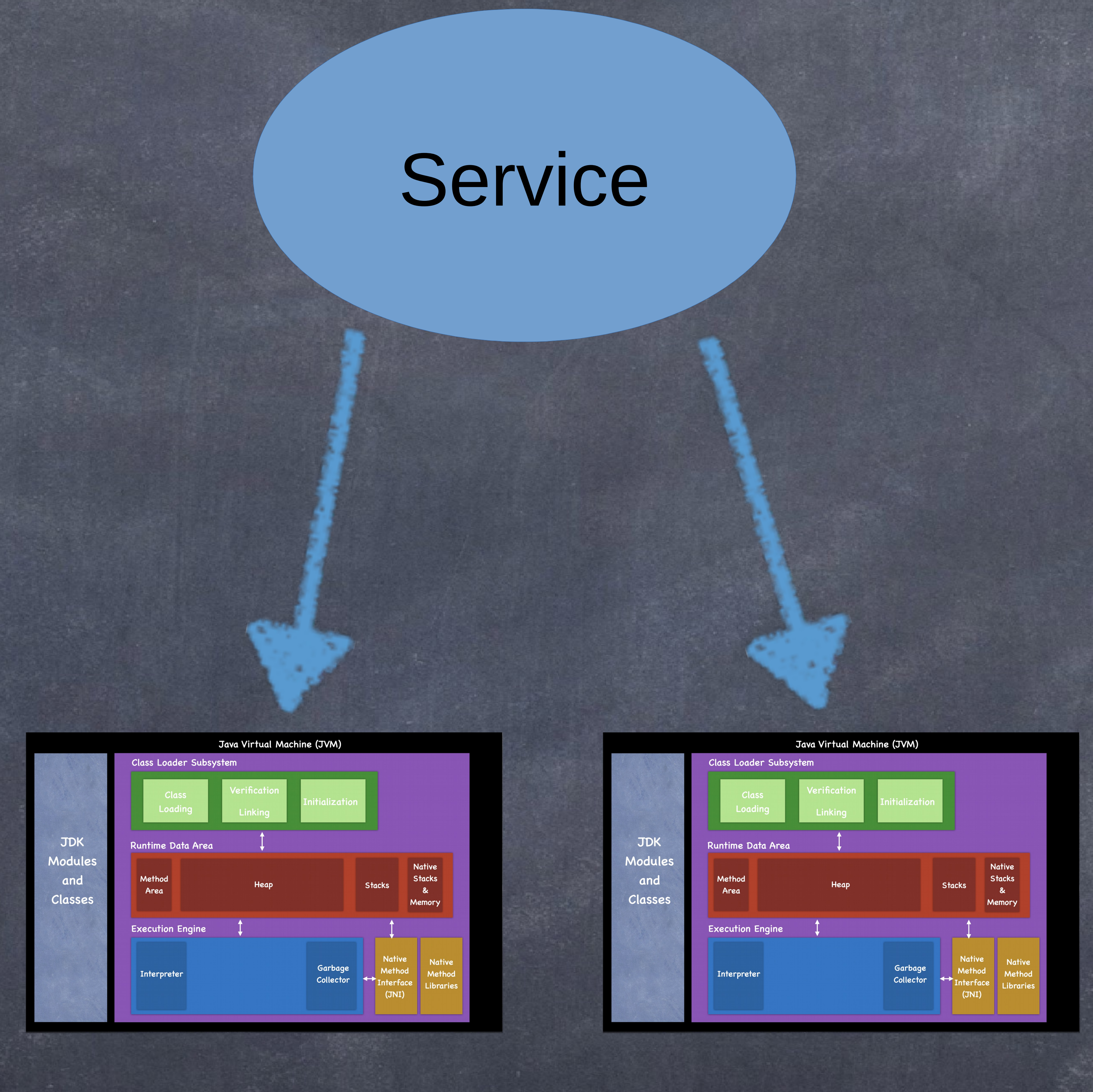
Scale up с проактивным Runtime’ом



Готов?

©2023 Azul Systems, Inc.

Scale up с проактивным Runtime’ом



Готов? **Да**

©2023 Azul Systems, Inc.
azul

Может ли JVM проактивно инициализировать Foo?

class Foo {
 int baz;
}

```
class Foo {
  int baz;

  Foo();
    Code:
       0: aload_0
       1: invokespecial #1// Method j.l.Object.”<init>”:()V
       4: return
}
```


azul

А этот Foo?

```
class Foo {
  final static Integer baz = new Integer(42);
}
```

```
class Foo {
  static final java.lang.Integer baz;

  Foo();
    Code:
       0: aload_0
       1: invokespecial #1 // Method j.l.Object.”<init>”:()V
       4: return

  static {};
    Code:
       0: new           #2 // class j.l.Integer
       3: dup
       4: bipush        42
       6: invokespecial #3 // Method j.l.Integer.”<init>”:(I)V
       9: putstatic     #4 // Field baz:Ljava/lang/Integer;
      12: return
}
```


azul

A теперь?

```
class Foo {
  static final LocalDateTime baz = LocalDateTime.now();
}
```

Проактивная модель

class *Foo* initialized

VM started

method *bar* compiled

?

method *bar* called

class *Foo* first used

9:00

9:30

10:00

10:30

11:30

©2023 Azul Systems, Inc.
azul

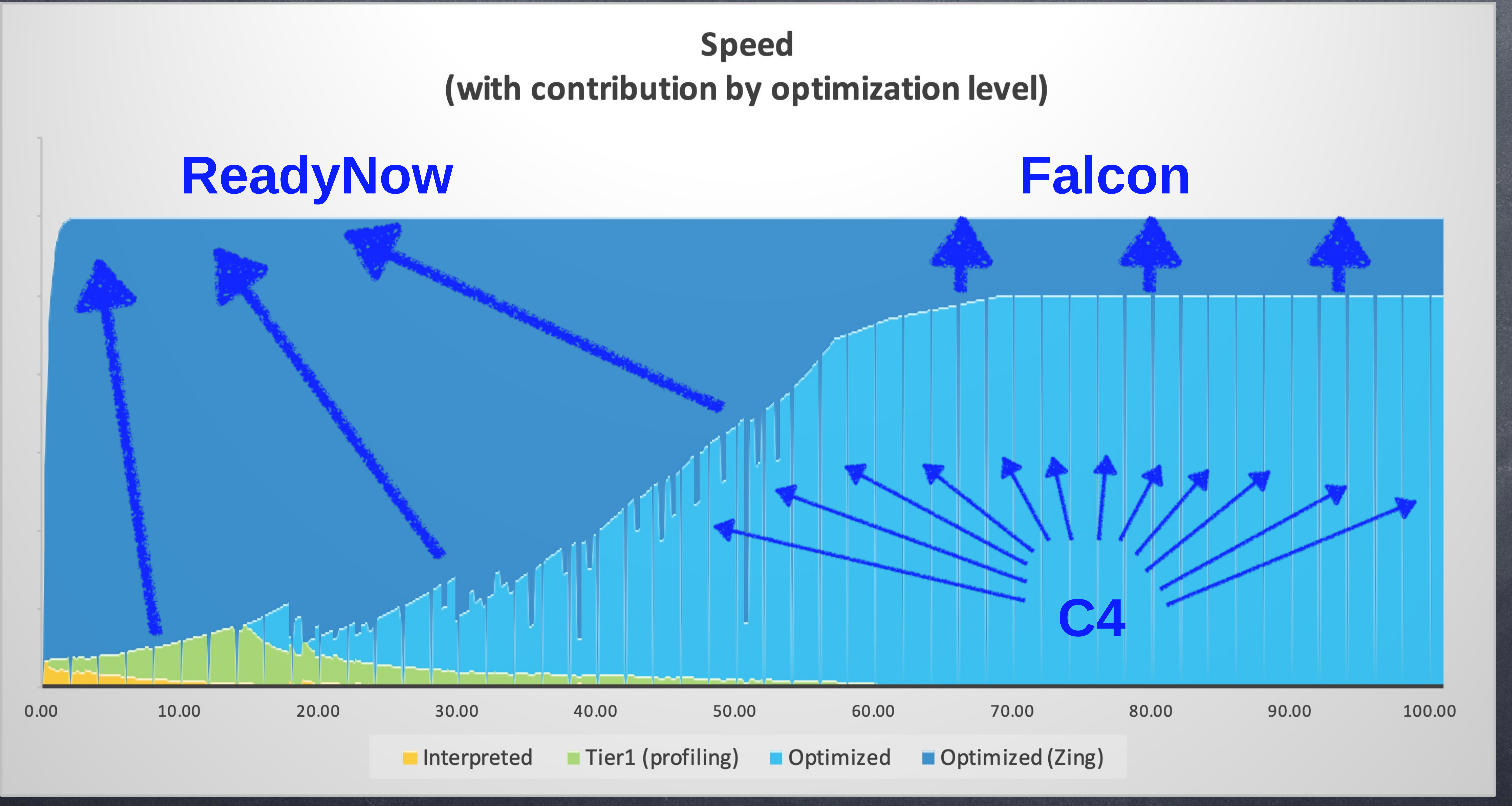
А этот “простенький” Enum?

```
public enum SimpleEnum {
  One
};
```

```
  static {
    Code:
       0: new           #4                  // class SimpleEnum
       3: dup
       4: ldc           #7                  // String One
       6: iconst_0
       7: invokespecial #8                  // Method “<init>”:(Lj.l.String;I)V
      10: putstatic     #9                  // Field One:LSimpleEnum;
      13: iconst_1
      14: anewarray     #4                  // class SimpleEnum
      17: dup
      18: iconst_0
      19: getstatic     #9                  // Field One:LSimpleEnum;
      22: aastore
      23: putstatic     #1                  // Field $VALUES:[LSimpleEnum;
      26: return
}
```


©2023 Azul Systems, Inc.

JWarmup JEP (https://openjdk.org/jeps/8203832)
Azul ReadyNow!



azul

Знакомство с Runtime’ом

И снова Compiler-Runtime: спекуляции и деоптимизации

©2023 Azul Systems, Inc.
azul

“Героические” (спекулятивные) оптимизации

“секрет” скорости Java

©2023 Azul Systems, Inc.

Пример: Мертвый код

```
static final void hotMethod() {
  if ( thing == null )
    System.out.print("always");
  else
    System.out.print("never");
}
```

```
static final void hotMethod() {
  if ( thing == null )
    System.out.print("always");
  else
    uncommon_trap(:unreached);
}
```


```
<bc code='199' bci='3'/>
<branch target_bci='17'
  taken='0' not_taken='5800'
  cnt='5800' prob='never'/>
<uncommon_trap
  bci='3' reason='unstable_if'
  action=‘reinterpret'
  comment='taken never'/>
```

```
<uncommon_trap thread='7171' stamp=‘5.104’
  compile_id='29' compiler='C2' level=‘4'
  reason='unstable_if' action='reinterpret' >
  <jvms
    method=‘…Unreached hotMethod ()V’
    bci=‘3’…/>
</uncommon_trap>
```


©2023 Azul Systems, Inc.

azul

25+%

И еще больше в циклах

©2023 Azul Systems, Inc.

Пример: Проверка инициализации класса

MyClass.getStatic()   new MyClass()

```
if (!vm.is_init(MyClass)) {
  vm.init(MyClass);
}
MyClass.getStatic()
```

↓

MyClass.getStatic()

Увеличение размера 20%

Замедление

или упущенные

Оптимизации на 5-10%

http://joeduffyblog.com/2015/12/19/safe-native-code/

©2023 Azul Systems, Inc.

“Героические” (спекулятивные) оптимизации

de-optimization

©2023 Azul Systems, Inc.
azul

Переход в Interpreter: однократно

```
static final void hotMethod() {
  if ( thing == null )
    System.out.print("always");
  else
    uncommon_trap(:unreached);
}
```

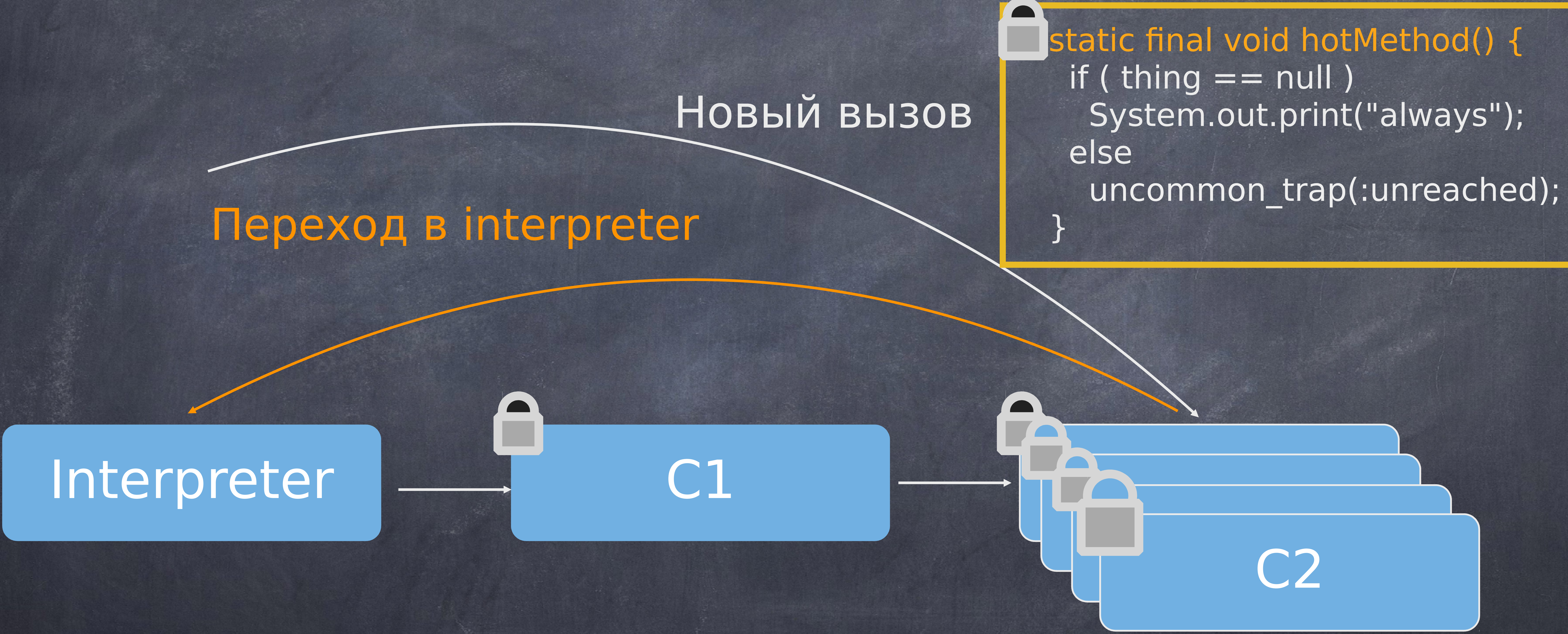
Interpreter → C1 → C2

uncommon trap +
Переход в interpreter

©2023 Azul Systems, Inc.

azul

Переход в Interpreter: Всех новых вызовов



©2023 Azul Systems, Inc.

azul

Переход в Interpreter:
Всех попыток исполнения

Новые вызовы

Возврат в метод

Переход в interpreter новых вызовов

Interpreter

C1

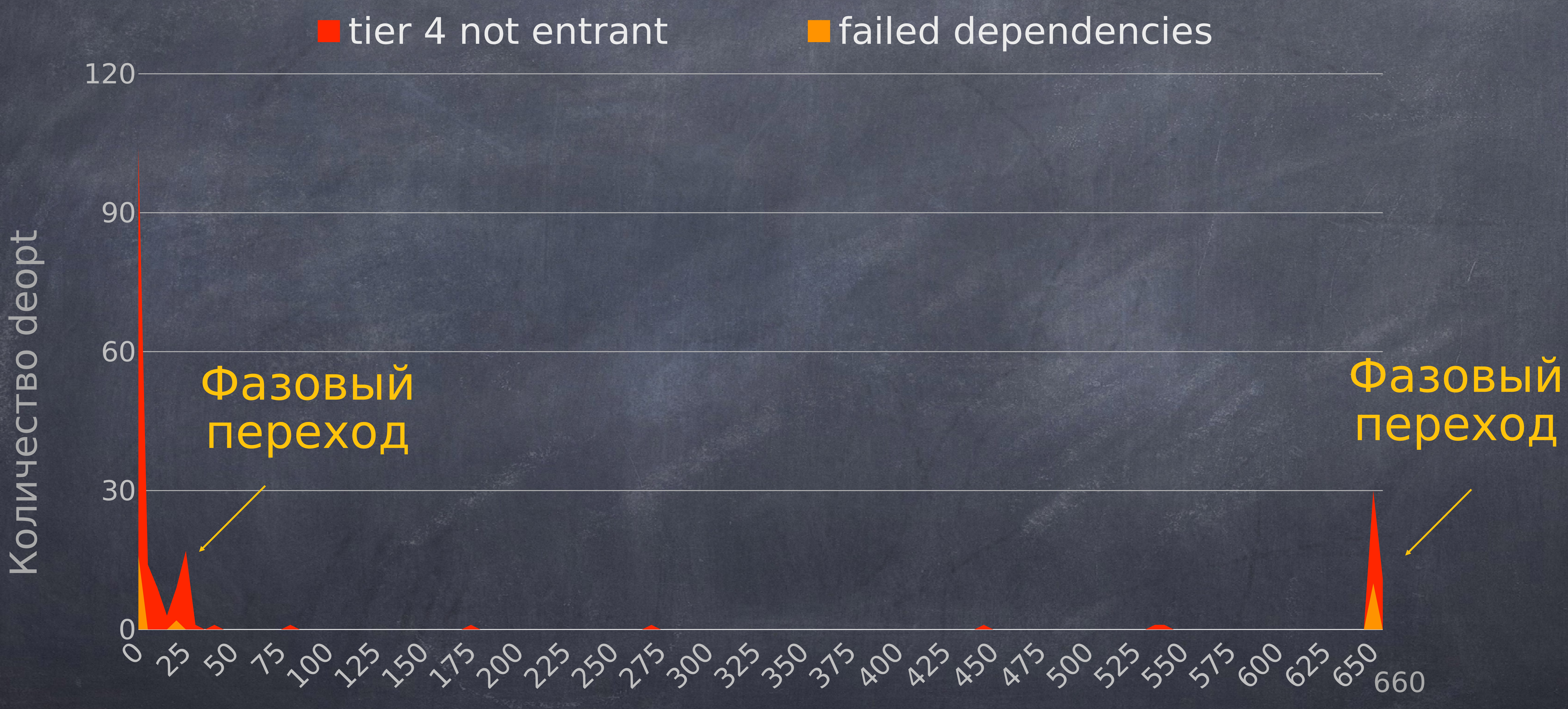
C2

переход в interpreter после return!

©2023 Azul Systems, Inc.

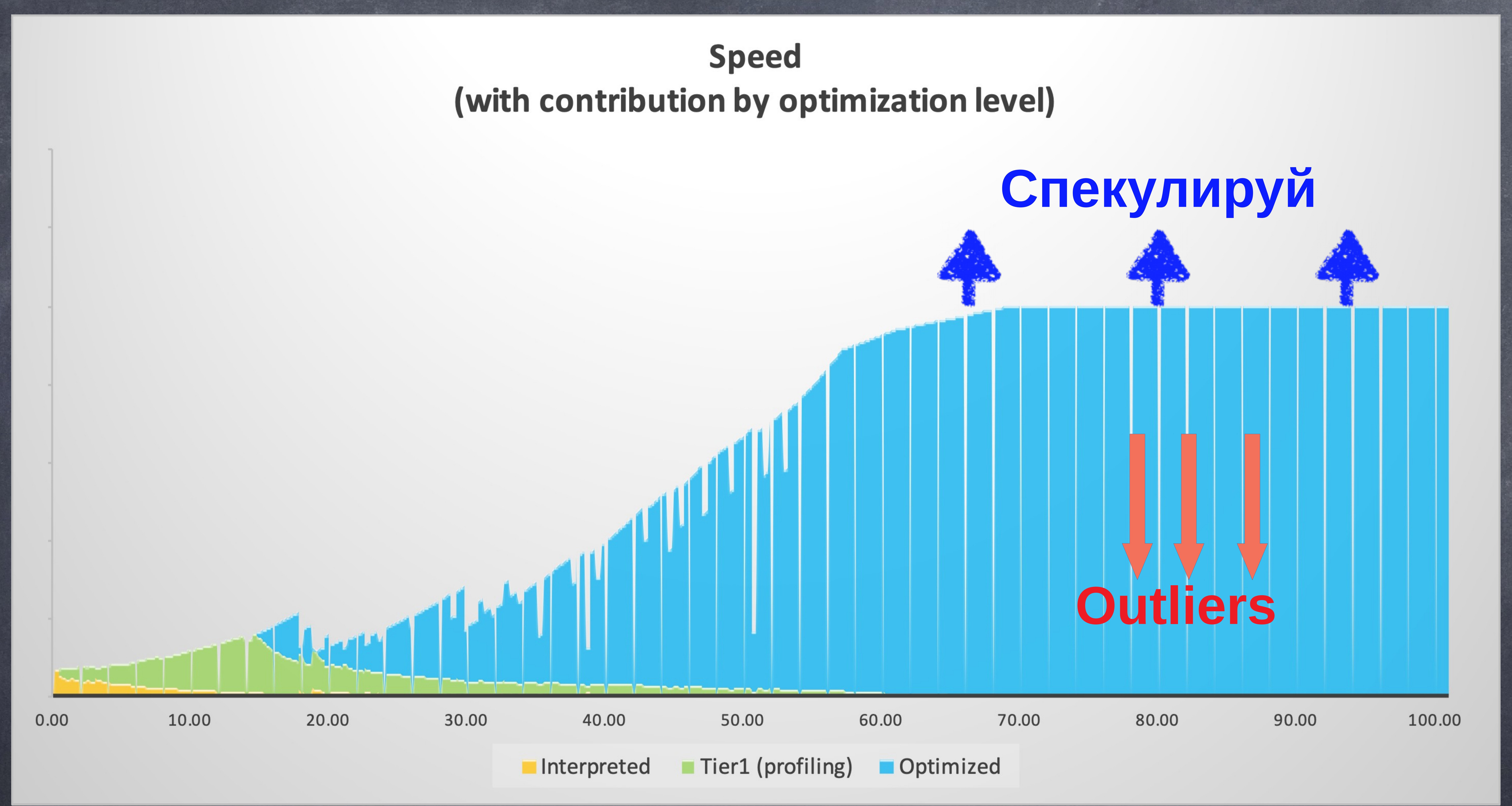
azul

Нарушение спекулятивных предположений



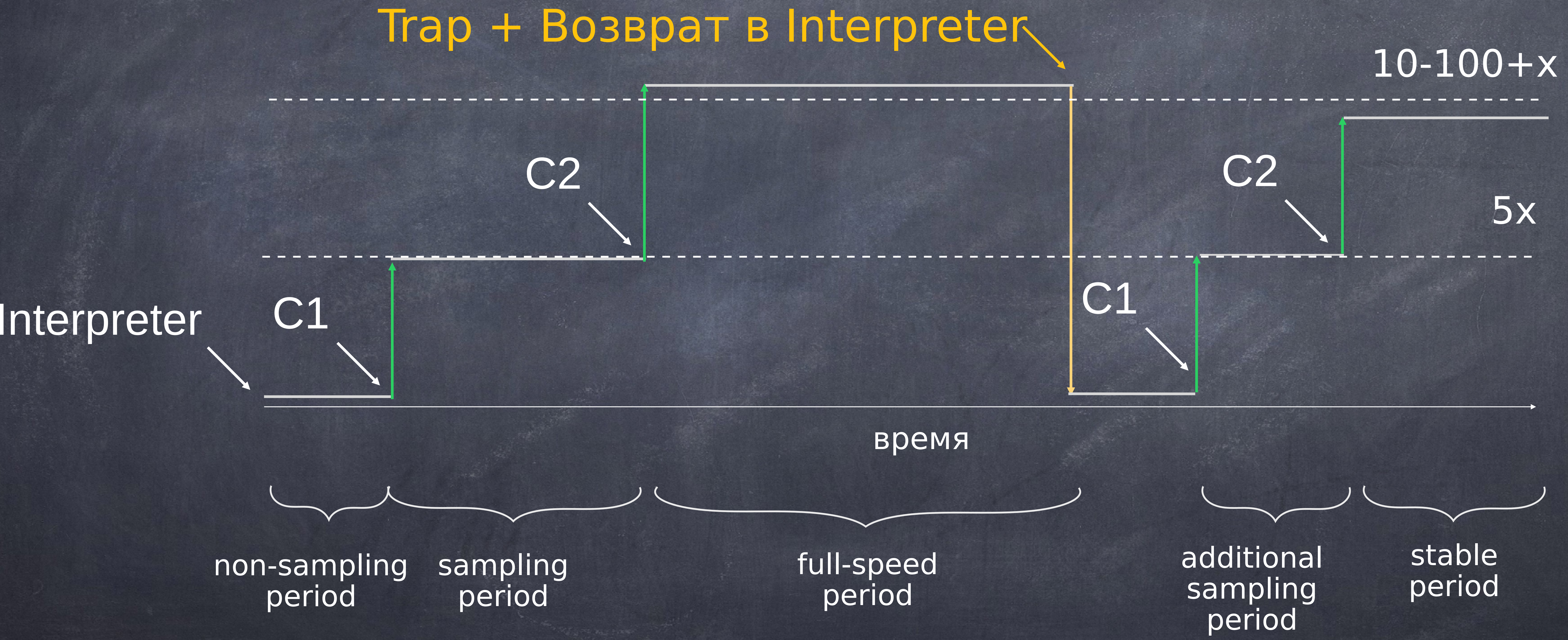
azul

©2023 Azul Systems, Inc.



©2023 Azul Systems, Inc.
azul

Уточненная модель жизни одного метода



https://blog.takipi.com/java-on-steroids-5-super-useful-jit-optimization-techniques/

©2023 Azul Systems, Inc.

azul

Реактивная модель
Vs
Проактивная модель

©2023 Azul Systems, Inc.
azul

Outliers и проактивная модель

Владимир Воскресенский

Azul Systems

Медленная Java?
Проблемы производительности,
которые не списать на GC

https://www.youtube.com/watch?v=AMGSVpOCQs8

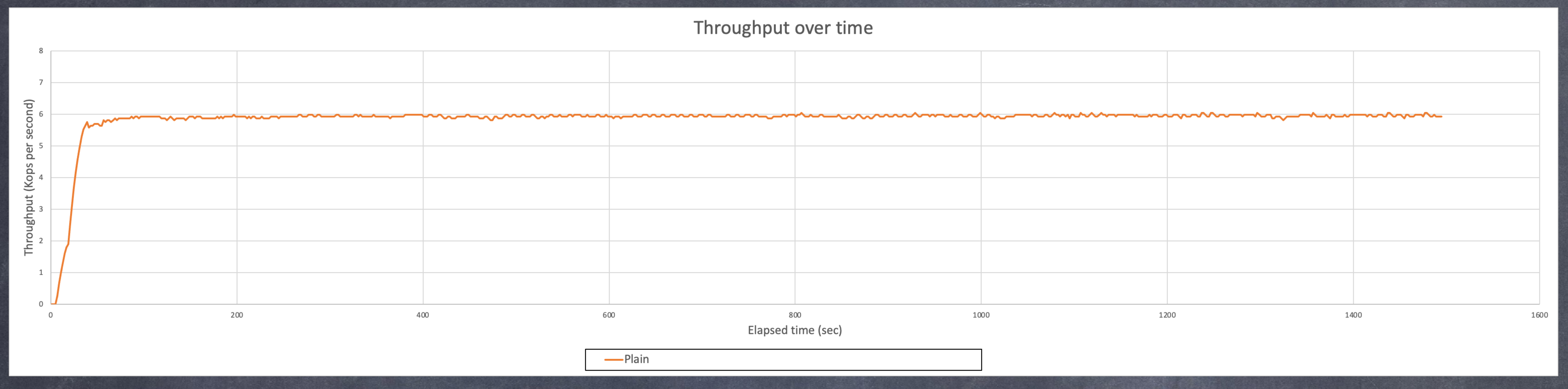
azul

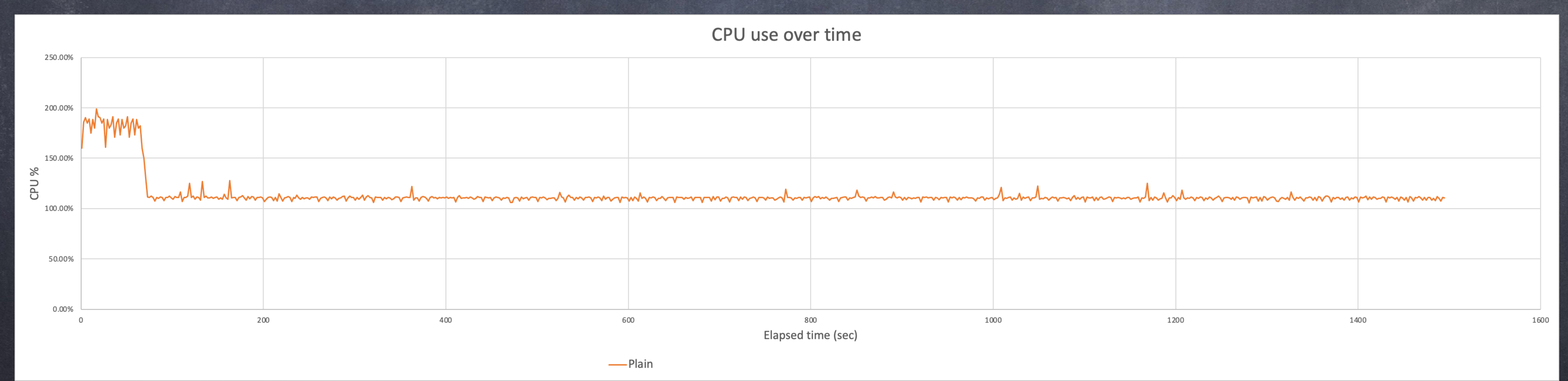
Аутсорсинг

Outsourcing

©2023 Azul Systems, Inc.
azul

Скорость & CPU по времени (2-vcore контейнер)

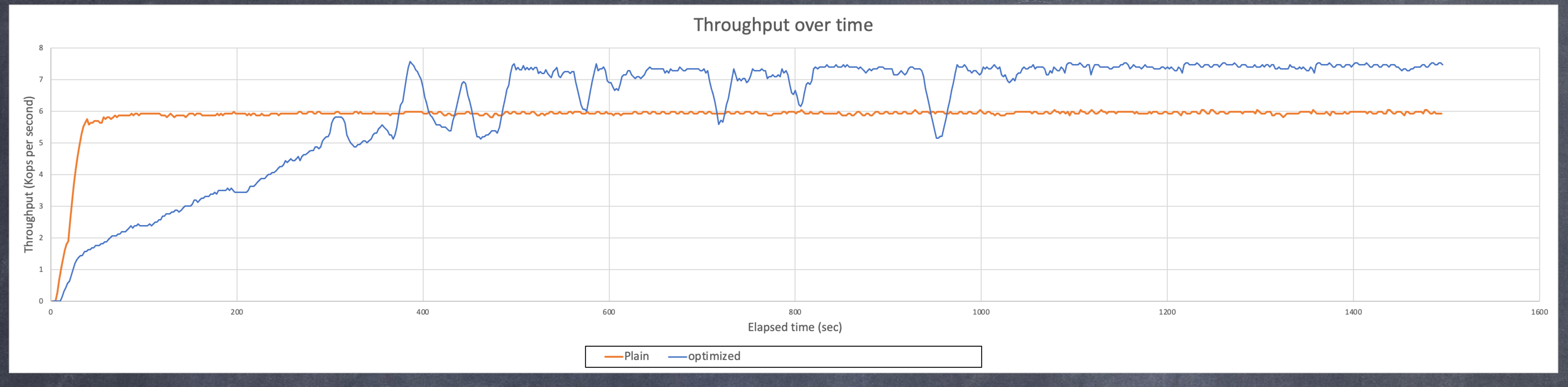


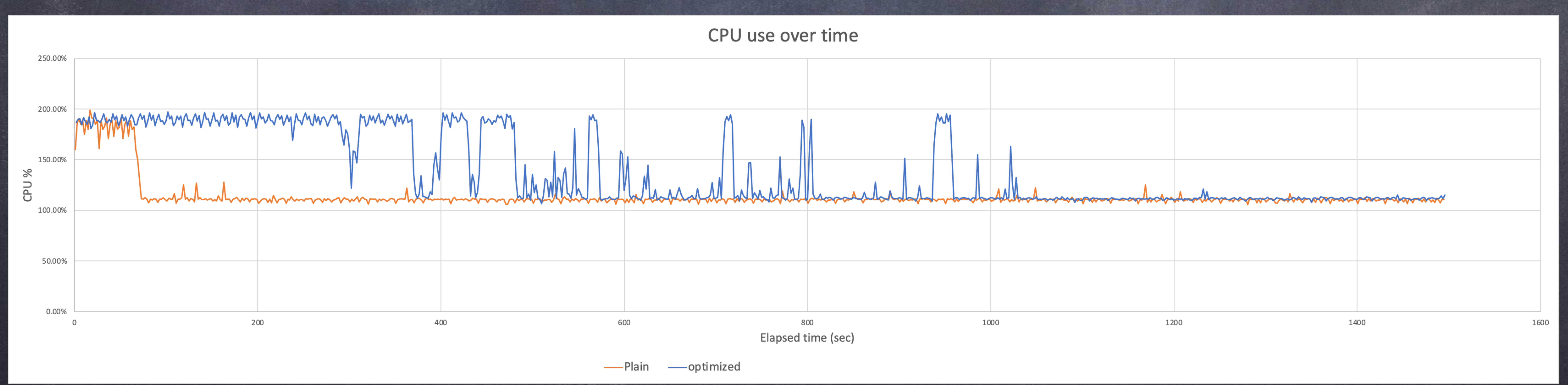


©2023 Azul Systems, Inc.

azul

Скорость & CPU по времени (2-vcore контейнер)





©2023 Azul Systems, Inc.
azul

Простые наблюдения

- В большинстве случаев код приложения:
  - Выполняется множество раз
  - Выполняется на разных устройствах
  - Выполняет одинаковые задачи в разных запусках и на разных устройства
  - Имеет ограниченное множество различных сценариев исполнения
- Это верно даже для часто обновляющихся приложений…
- Когда JVM делает лично для себя оптимизацию и использует её однократно, она “теряется” при перезапуске

©2023 Azul Systems, Inc.
azul

Оптимизации можно переиспользовать…

Даже *спекулятивные* оптимизации…

©2023 Azul Systems, Inc.
azul

Код+Assumptions

Optimized Code

Address	Code	Opcode
0x3000fe50	pushq %rax	0xfff0
0x3000fe52	cmpl $0, %gs:104	0x65833c25680000000
0x3000fe5b	jne 127 ; ABS: 0x3000fedc	0x757f
0x3000fe5d	movl 8(%rsi), %ecx // NPE-> 0x3000fef5	0x8b4e08
0x3000fe60	testq %rcx, %rcx	0x4885c9
0x3000fe63	je 115 ; ABS: 0x3000fed8	0x7473
0x3000fe65	cmpl $7, %ecx	0x83f907
0x3000fe68	ja 20 ; ABS: 0x3000fe7e	0x7714
0x3000fe6a	xorl %edx, %edx	0x31d2
0x3000fe6c	xorl %eax, %eax	0x31c0
0x3000fe6e	nop	0x6690
0x3000fe70	addl 12(%rsi,%rdx,4), %eax	0x0344960c
0x3000fe74	incq %rdx	0x48ffc2
0x3000fe77	cmpq %rcx, %rdx	0x4839ca
0x3000fe7a	jl -12 ; ABS: 0x3000fe70	0x7cf4
0x3000fe7c	popq %rcx	0x59
0x3000fe7d	retq	0xc3
0x3000fe7e	movl %ecx, %r8d	0x4189c8
0x3000fe81	andl $7, %r8d	0x4183e007
0x3000fe85	movq %rcx, %rdx	0x4889ca
0x3000fe88	subq %r8, %rdx	0x4c29c2
0x3000fe8b	je -35 ; ABS: 0x3000fe6a	0x74dd
[illegible]x3000fe8d	leaq 28(%rsi), %rax	0x488d461c
[illegible]000fe91	pxor %xmm0, %xmm0	0x660fefc0
0[illegible]0fe95	movq %rdx, %rdi	0x4889d7
0x[illegible]fe98	pxor %xmm1, %xmm1	0x660fefc9
0x3[illegible]fe9c	nopl (%rax)	0x0f1f4000
0x3[illegible]fea0	movdqu -16(%rax), %xmm2	0xf30f6f50f0
0x3[illegible]fea5	movdqu (%rax), %xmm3	0xf30f6f18
0x[illegible]0fea9	paddd %xmm2, %xmm0	0x660ffec2
[illegible]00fead	paddd %xmm3, %xmm1	0x660ffecb
[illegible]3000feb1	addq $32, %rax	0x4883c020
0x3000feb5	addq $-8, %rdi	0x4883c7f8
0x3000feb9	jne -27 ; ABS: 0x3000fea0	0x75e5
0x3000febb	paddd %xmm0, %xmm1	0x660ffec8
0x3000febf	pshufd $78, %xmm1, %xmm0	0x660f70c14e
0x3000fec4	paddd %xmm1, %xmm0	0x660ffec1
0x3000fec8	phaddd %xmm0, %xmm0	0x660f3802c0
0x3000fecd	movd %xmm0, %eax	0x660f7ec0
0x3000fed1	testl %r8d, %r8d	0x4585c0
0x3000fed4	jne -102 ; ABS: 0x3000fe70	0x759a
0x3000fed6	jmp -92 ; ABS: 0x3000fe7c	0xeba4
0x3000fed8	xorl %eax, %eax	0x31c0
0x3000feda	popq %rcx	0x59
0x3000fedb	retq	0xc3
0x3000fedc	movq %rsi, (%rsp)	0x48893424
0x3000fee0	movabsq $805334400, %rax	0x48b8806d003000000000
0x3000feea	callq *%rax	0xffd0
0x3000feec	movq (%rsp), %rsi	0x488b3424
0x3000fef0	jmp -152 ; ABS: 0x3000fe5d	0xe968ffffff
0x3000fef5	movabsq $805319872, %rax	0x48b8c034003000000000
0x3000feff	movl $7, %edi	0xbf07000000
0x3000ff04	callq *%rax	0xffd0
0x3000ff06	addq $-8, %rsp	0x4883c4f8
0x3000ff0a	jmp -50575 ; ABS: 0x30003980 = StubRoutines::deoptimize	0xe9713affff
0x3000ff0f	int3	0xcc

Cloud Native Compiler

Assumptions

Only one implementor of Animal.getColor() exists

Assertions are disabled

Bar has no subclasses

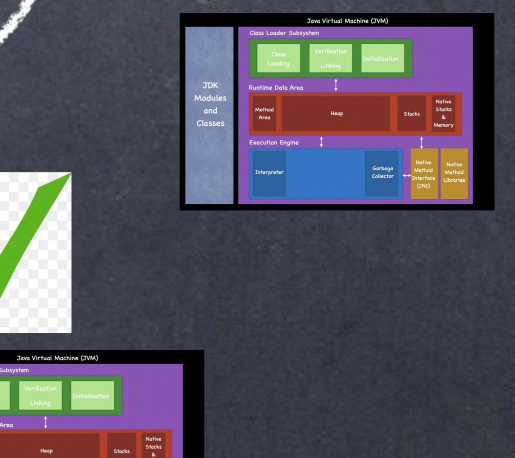
Today is Tuesday

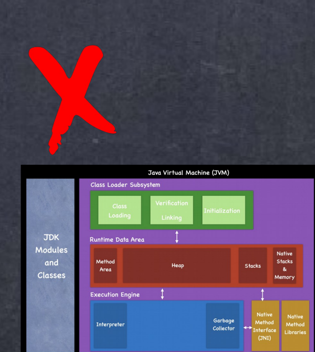
FastDoof.buf is truly final

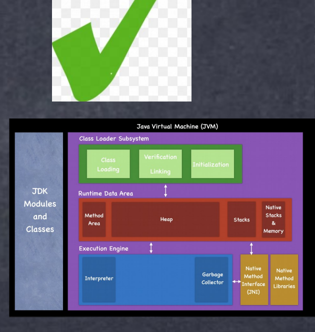
Locale.default() is ENGLISH

Longest String seen so far is <128KB

The actual code for SomeUtil.computeStuff() is {....} and it's checksum is 0x651712365



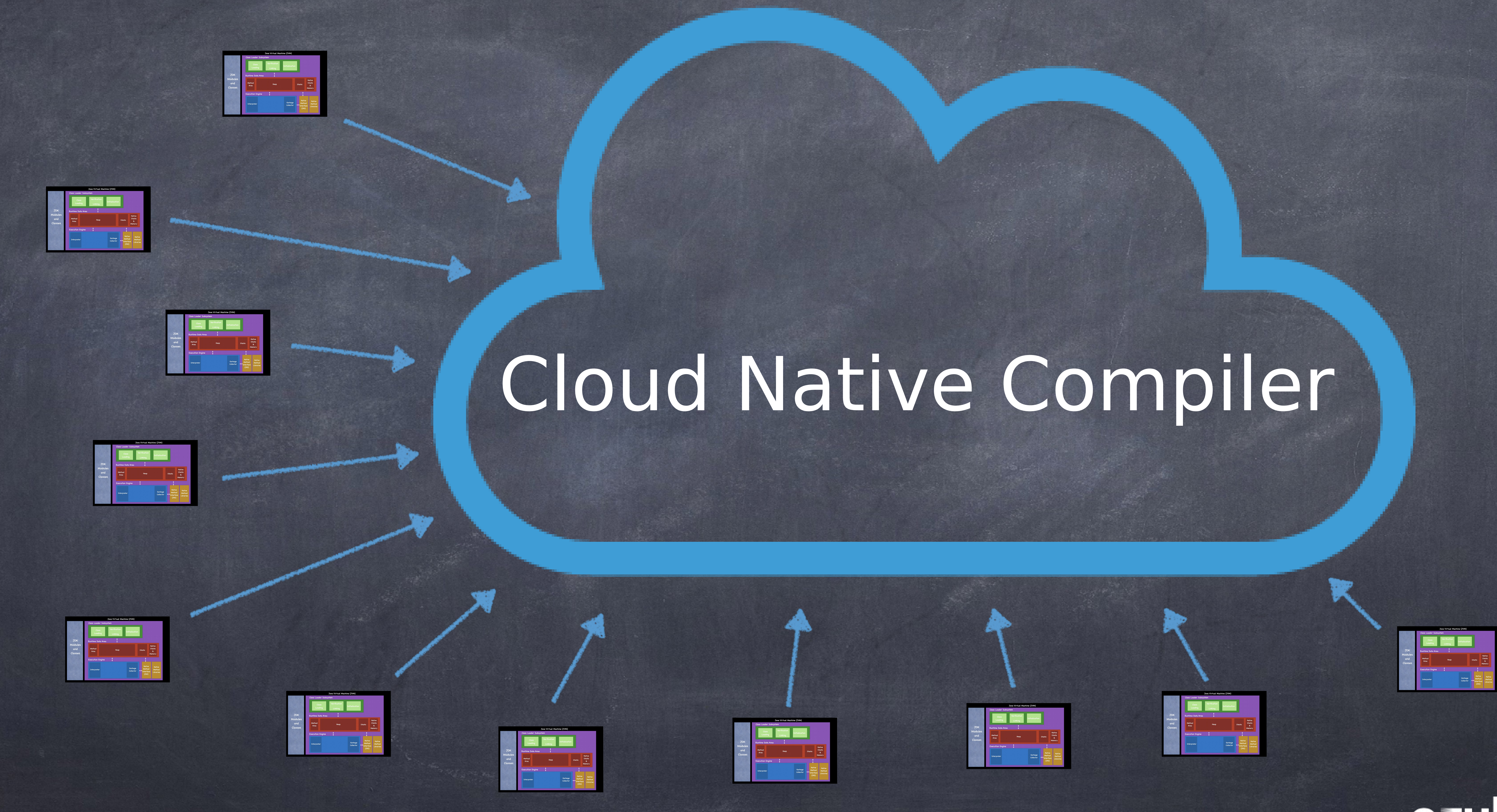




©2023 Azul Systems, Inc.

azul

JIT как Cloud Native ресурс

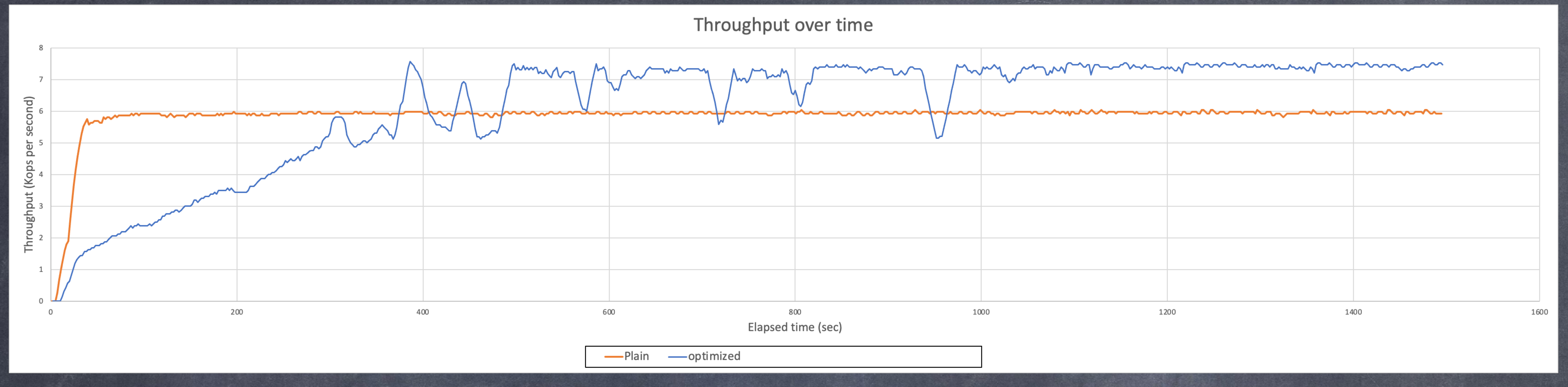


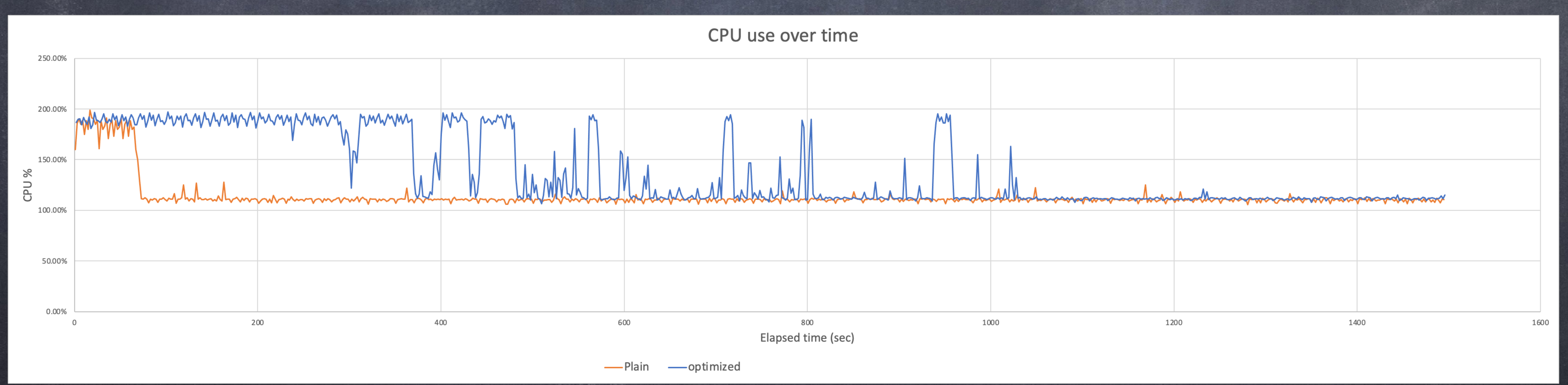
©2022 Azul Systems, Inc.
azul

Что даёт?

©2023 Azul Systems, Inc.

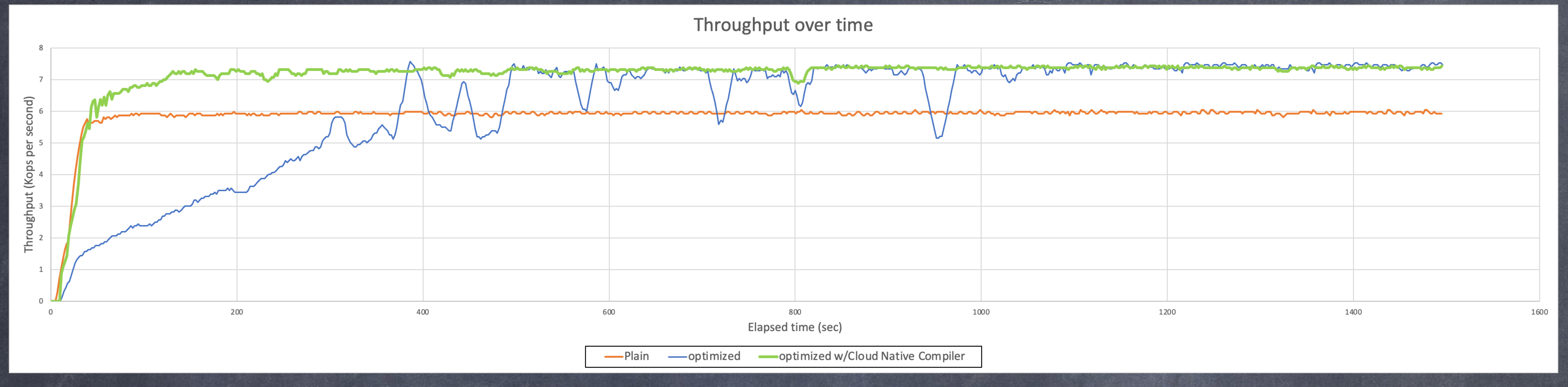
Скорость & CPU по времени (2-vcore контейнер)

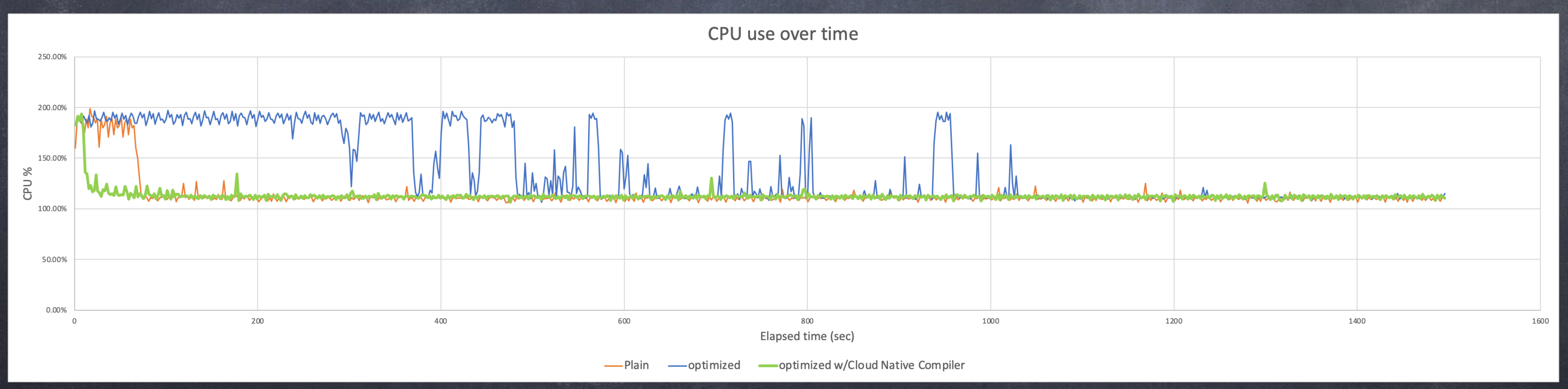




©2023 Azul Systems, Inc.
azul

Скорость & CPU по времени (2-vcore контейнер)

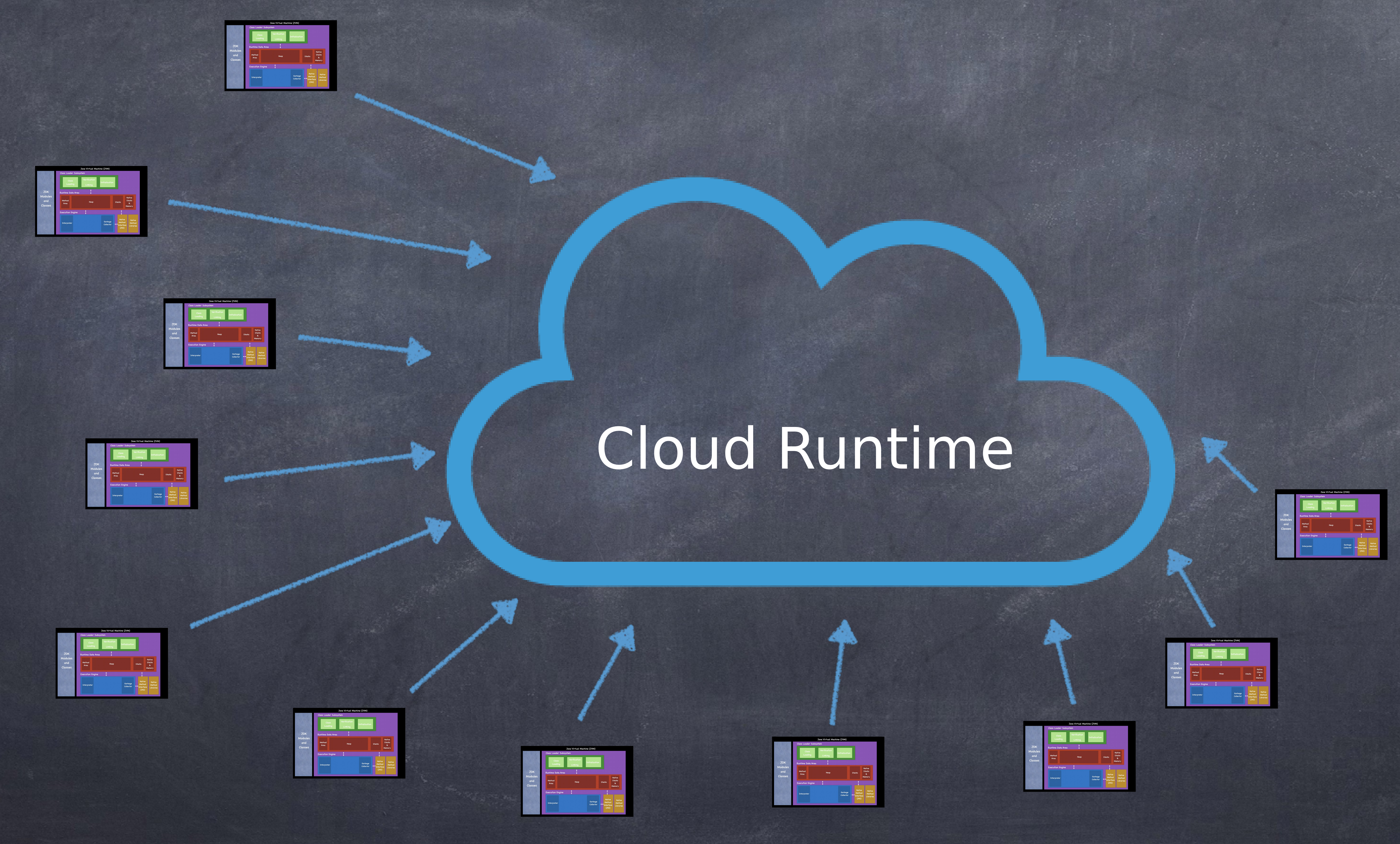




©2023 Azul Systems, Inc.

azul

Используются совместно и делятся между всеми…



©2022 Azul Systems, Inc.
azul

Реактивная модель

+

Проактивная модель

©2023 Azul Systems, Inc.
azul

VM клиенты “**реактивно**” сообщают в Cloud Runtime:

- Какие классы и когда загружают, линкуют, инициализируют
- Какие методы и когда первый раз вызываются и когда становятся горячими
- Какие зависимости у спекулятивных компиляций и от каких состояний VM они зависят
- При каких событиях спекуляции нарушаются

©2022 Azul Systems, Inc.
azul

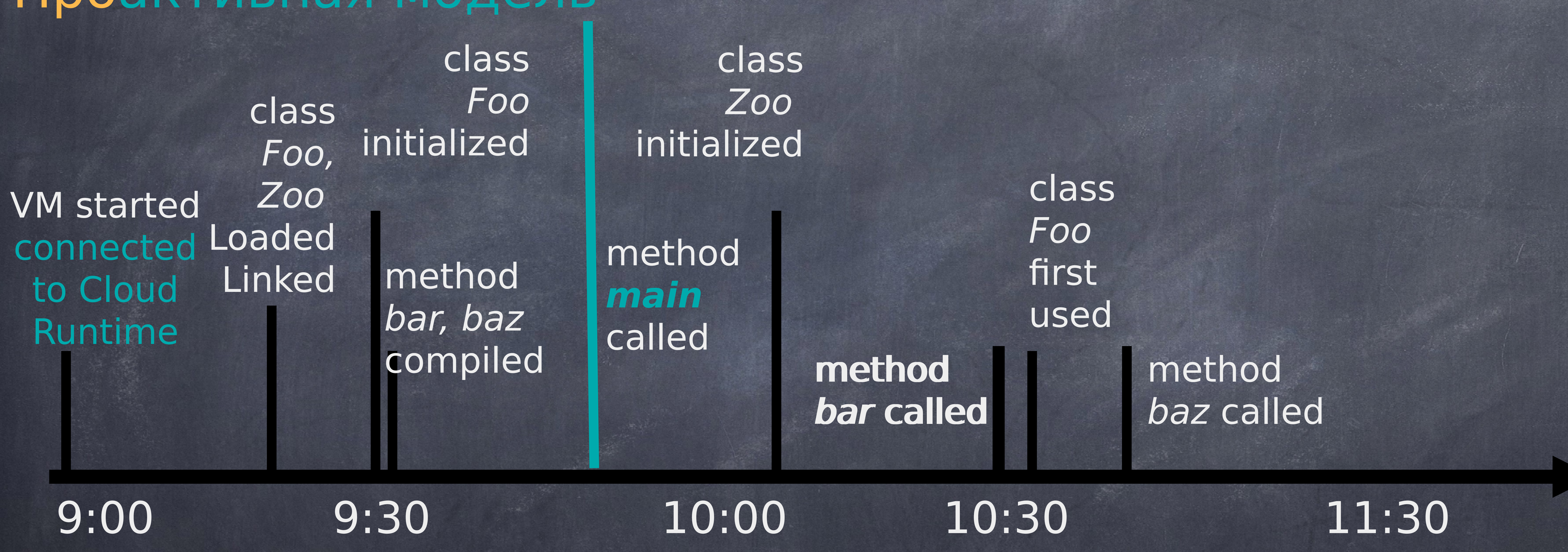
Cloud Runtime “проактивно” советует VM клиентам:

- Из какого образа стартовать (restore) в для устранения warmup фазы
- Какие классы и когда проактивно загружать, линковать, инициализирать
- Какие методы и когда проактивно компилировать и инсталлировать
- К каким фазовым переходам готовиться и при каких условиях

©2022 Azul Systems, Inc.

Жизненый цикл Cloud Native JVM

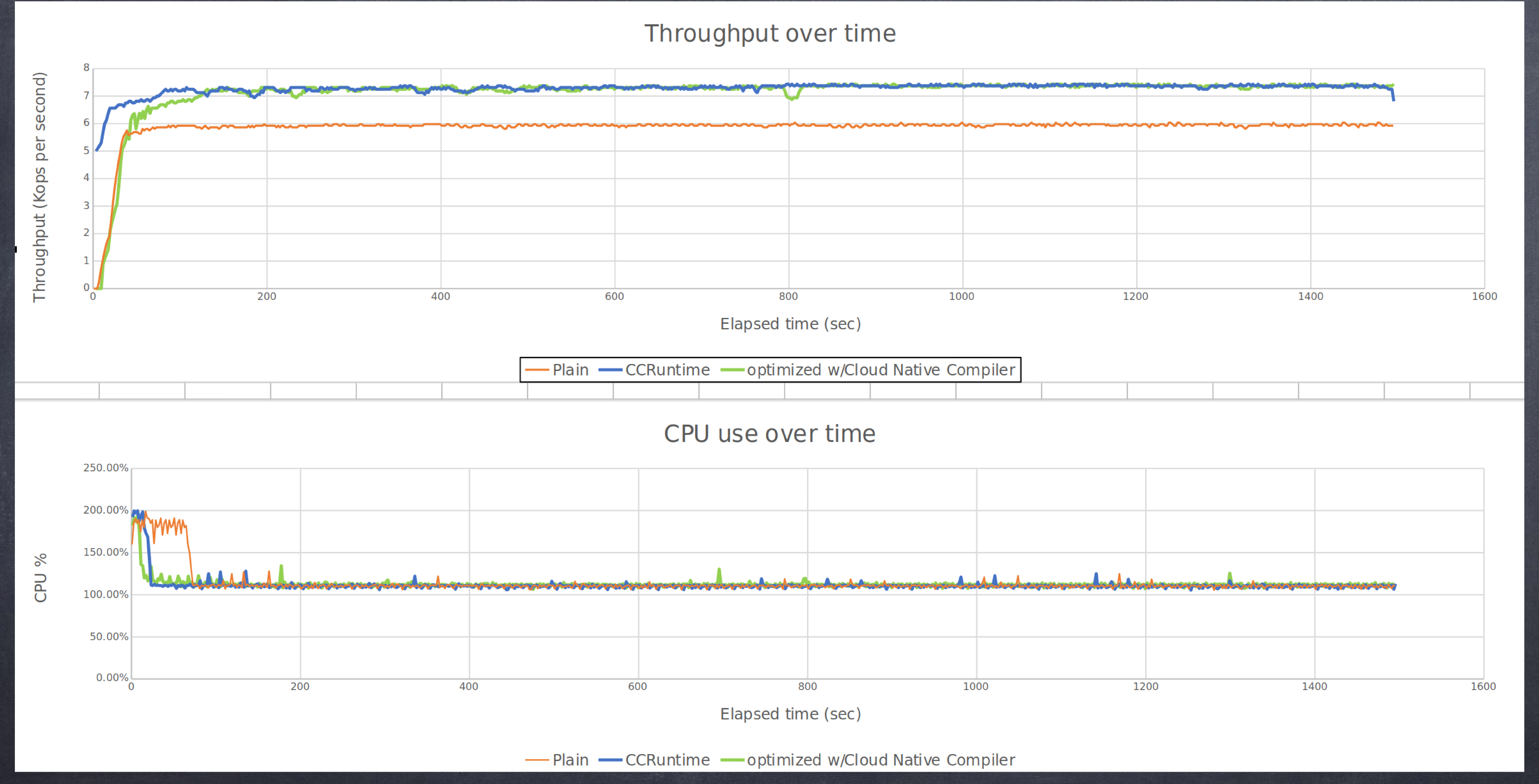
Проактивная модель



©2023 Azul Systems, Inc.

azul

Скорость & CPU по времени (2-vcore контейнер)



©2023 Azul Systems, Inc.

azul

Итого...

©2023 Azul Systems, Inc.

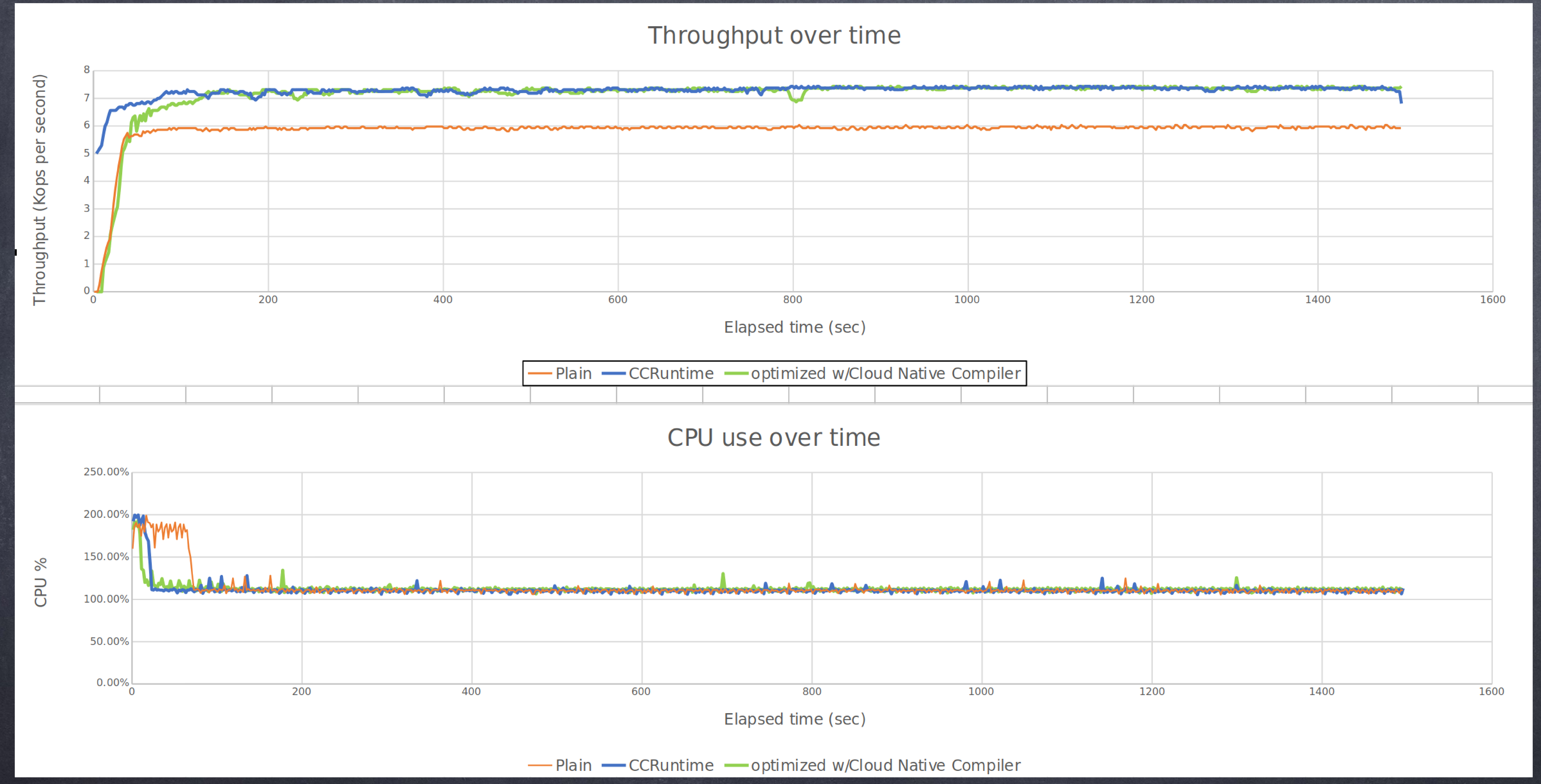
Реактивная модель

+

Проактивная модель

©2023 Azul Systems, Inc.
azul

Cloud Native Runtime == Быстрее, Выше, Сильнее



©2023 Azul Systems, Inc.

azul

Cloud Native Runtime == Эффективнее

©2023 Azul Systems, Inc.
azul

Cloud Native Runtime уже близко….

©2023 Azul Systems, Inc.
azul

Cloud Native JVM

Cloud Runtime

Владимир Воскресенский
Azul Systems
Distinguished Engineer
vladimir@azul.com

Q&A

azul

©2023 Azul Systems, Inc.