В нативный код
из уютного мира Java

Путешествие Туда и Обратно

Иван Углянский
Huawei

Иван Углянский

JVM engineer at Excelsior@Huawei

JUGNsk co-lead

ivan.ugliansky@gmail.com

@dbg_nsk

Какой еще нативный код?

Какой еще нативный код?

```
public class JavaToNative {

   static native void goNative();

   static native void goThere(Callback andBackAgain);

}
```

Какой еще нативный код?

```
public class JavaToNative {

   static native void goNative();

   static native void goThere(Callback andBackAgain);

}
```

Какой еще нативный код?

```
public class JavaToNative {

   static native void goNative();

   static native void goThere(Callback andBackAgain);

}
```

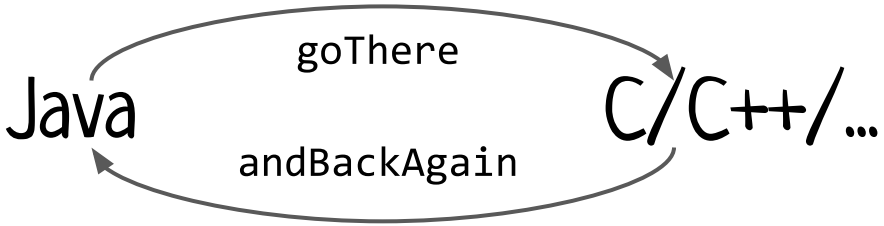
Какой еще нативный код?

```
public class JavaToNative {

   static native void goNative();

   static native void goThere(Callback andBackAgain);

}
```



Но зачем нам нативы?

Но зачем нам нативы?

Java - managed язык

Но зачем нам нативы?

Java - managed язык

- автоматическое управление памятью

Но зачем нам нативы?

Java - managed язык

- автоматическое управление памятью
- безопасные операции

Но зачем нам нативы?

Java - managed язык

- автоматическое управление памятью
- безопасные операции

Java - это...

Но зачем нам нативы?

Java - managed язык

- автоматическое управление памятью
- безопасные операции

А вот нативный код - это...

Но знаете, как это бывает...

Причины звать нативный код из Java

1. Есть отличная библиотека!

   (Но она на C/C++)

Причины звать нативный код из Java

1. Есть отличная библиотека!

   (Но она на C/C++)

   OpenGL, DirectX, Tensorflow, Cuda, OpenCL,

   OpenSSl, Vulkan, Криптография, ...

Причины звать нативный код из Java

1. Есть отличная библиотека!

   (Но она на C/C++)

2. Да мне всего лишь один метод из WinAPI позвать!

Причины звать нативный код из Java

1. Есть отличная библиотека!

   (Но она на C/C++)

2. Да мне всего лишь один метод из WinAPI позвать!

3. Напишу один модуль на C++, все как разгонится!

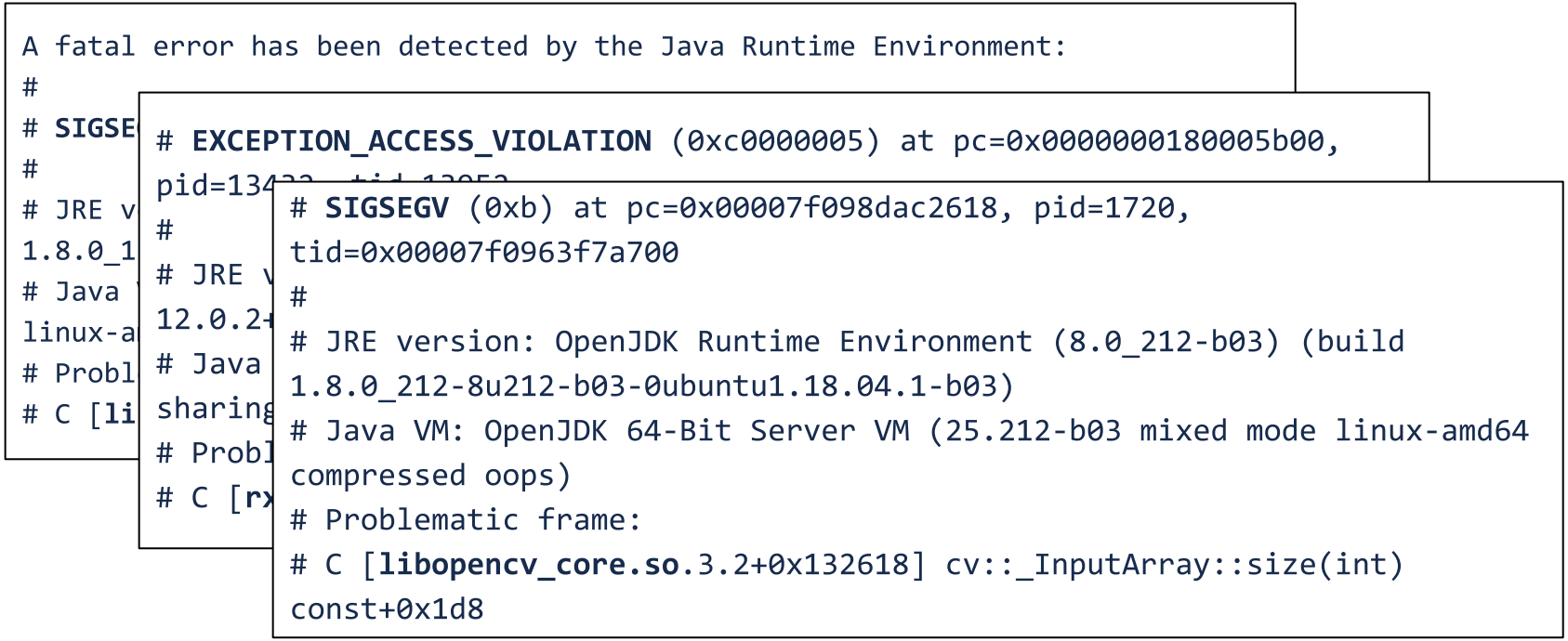
Причины звать нативный код из Java

4. Ничего из этого не нужно, но JVM сама зовет нативный код!

```
A fatal error has been detected by the Java Runtime Environment:
#
# SIGSEGV (0xb) at pc=0x00007f542922359f, pid=25670, tid=0x00007f5444efa700
#
# JRE version: Java(TM) SE Runtime Environment (8.0_151-b12) (build
1.8.0_151-b12)
# Java VM: Java HotSpot(TM) 64-Bit Server VM (25.151-b12 mixed mode
linux-amd64 compressed oops)
# Problematic frame:
# C [libavutil.so.56+0x1159f] av_strstart+0x1f
```

```
A fatal error has been detected by the Java Runtime Environment:
#
# SIGSE
#
# JRE v
1.8.0_1
# Java
linux-a
# Probl
# C [li
```

```
# EXCEPTION_ACCESS_VIOLATION (0xc0000005) at pc=0x0000000180005b00,
pid=13432, tid=13952
#
# JRE version: Java(TM) SE Runtime Environment (12.0.2+10) (build
12.0.2+10)
# Java VM: Java HotSpot(TM) 64-Bit Server VM (12.0.2+10, mixed mode,
sharing, tiered, compressed oops, g1 gc, windows-amd64)
# Problematic frame:
# C [rxtxSerial.dll+0x5b00]
```


A fatal error has been detected by the Java Runtime Environment:
#
SIGSE
#
JRE v
1.8.0_1
Java
linux-a
# Probl
# C [li
# EXCEPTION_ACCESS_VIOLATION (0xc0000005) at pc=0x0000000180005b00,
pid=13
#
JRE v
12.0.2+
Java
sharing
# Probl
# C [rx
# SIGSEGV (0xb) at pc=0x00007f098dac2618, pid=1720,
tid=0x00007f0963f7a700
#
JRE version: OpenJDK Runtime Environment (8.0_212-b03) (build
1.8.0_212-8u212-b03-0ubuntu1.18.04.1-b03)
Java VM: OpenJDK 64-Bit Server VM (25.212-b03 mixed mode linux-amd64
compressed oops)
Problematic frame:
C [libopencv_core.so.3.2+0x132618] cv::_InputArray::size(int)
const+0x1d8

```
A fatal error has been detected by the Java Runtime Environment:
#
# SIGSE
#
# JRE v
1.8.0_1
# Java
linux-a
# Probl
# C [li
```

```
# EXCEPTION_ACCESS_VIOLATION (0xc0000005) at pc=0x0000000180005b00,
pid=13
#
# JRE v
12.0.2+
# Java
sharing
# Probl
# C [rx
```

```
# SIGSEGV (0xb) at pc=0x00007f098dac2618, pid=1720,
tid=0x00007f0963f7a700
#
# JRE versio
1.8.0_212-8u
# Java VM: C
compressed o
# Problemati
# C [liboper
const+0x1d8
```

Будем разбираться

Будем разбираться

1. Почему так много проблем с нативами?

Будем разбираться

1. Почему так много проблем с нативами?
2. Как пройти в Мордор и не получить SIGSEGV?

Как позвать натив?

Как позвать натив?

Где взять нативы?

Как позвать натив?

Как работать с Java из натива?

Где взять нативы?

Как позвать натив?

Где взять нативы?

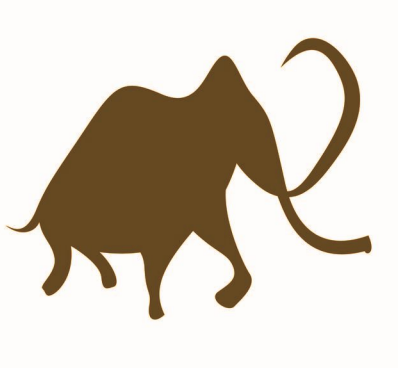
Как работать с Java из натива?

Как себя должен вести GC?

Немного истории

История до нашей эры

1. JDK 1.0 NMI — Native Method Invocation (Sun JVM)
2. Raw Native Interface (RNI) in Microsoft J++ and J/Direct
3. Netscape's JRI (Java Runtime Interface)
4. ...



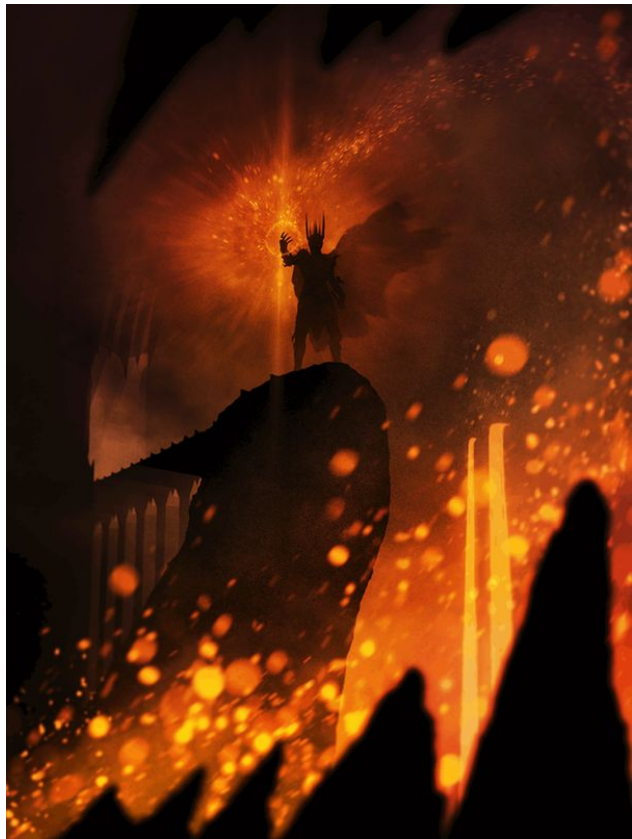
Наша эра

JNI - Java Native Interface

Наша эра

JNI - Java Native Interface

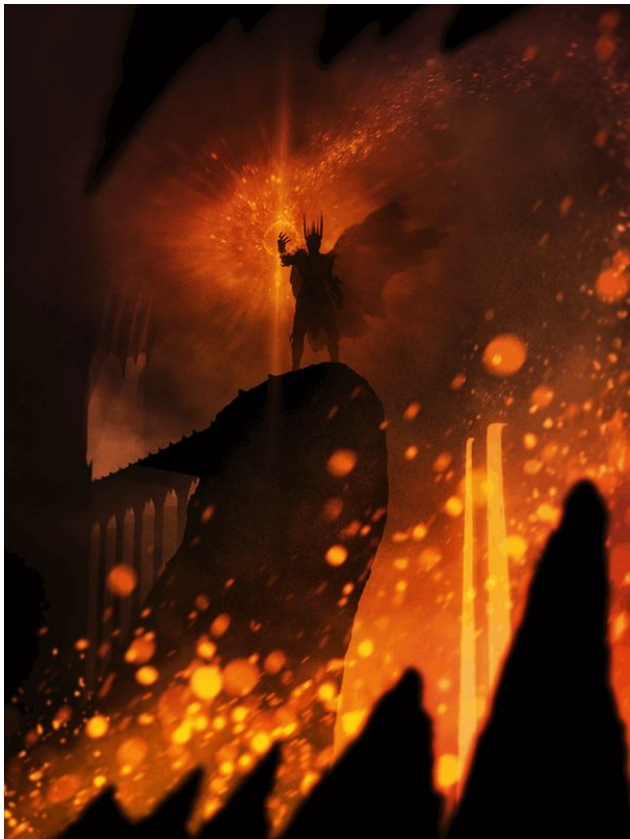
- ✓ Единый интерфейс, чтобы править всеми!



Наша эра

JNI - Java Native Interface

- ✓ Единый интерфейс, чтобы править всеми!
- ✓ Детали реализации скрыты ⇒ JVM нейтрален, GC нейтрален



Как это выглядит со стороны Java?

Как это выглядит со стороны Java?

```
public class JavaToNative {

   static native void goNative();

   static native void goThere(Callback andBackAgain);

   public static void main(String[] args) {
      System.loadLibrary("NativeLib");
      goThere(new Callback("Eagles"));
   }
}
```

Как это выглядит со стороны Java?

```
public class JavaToNative {

   static native void goNative();

   static native void goThere(Callback andBackAgain);

   public static void main(String[] args) {
      System.loadLibrary("NativeLib");
      goThere(new Callback("Eagles"));
   }
}
```

Как это выглядит со стороны Java?

```
public class JavaToNative {

   static native void goNative();

   static native void goThere(Callback andBackAgain);

   public static void main(String[] args) {
      System.loadLibrary("NativeLib");
      goThere(new Callback("Eagles"));
   }
}
```

Как это выглядит со стороны Java?

```
public class JavaToNative {

    stat

    stat

    publ
        S
        {

    }
}
```

```
class Callback {

    private final String transport;

    public Callback(String transport) {
        this.transport = transport;
    }

    public void call() {
        System.out.println("Ok, we are in Shire again!
                           Returned by " + transport);
    }
}
```

Как это выглядит со стороны C?

Как это выглядит со стороны C?

```
public class JavaToNative {

    static native void goThere(Callback andBackAgain);

}
```

Как это выглядит со стороны C?

```
public class JavaToNative {

    static native void goThere(Callback andBackAgain);

}
```

javac JavaToNative.java -h .

Как это выглядит со стороны C?

```
public class JavaToNative {
    static native void goThere(Callback andBackAgain);
}
```

javac JavaToNative.java -h .

JavaToNative.h

```
JNIEXPORT void JNICALL Java_JavaToNative_goThere
 (JNIEnv *, jclass, jobject);
```

Как это выглядит со стороны C?

```
public class JavaToNative {
    static native void goThere(Callback andBackAgain);
}
```

javac JavaToNative.java -h .

JavaToNative.h

```
JNIEXPORT void JNICALL Java_JavaToNative_goThere
 (JNIEnv *, jclass, jobject);
```

Как это выглядит со стороны C?

Примитивные

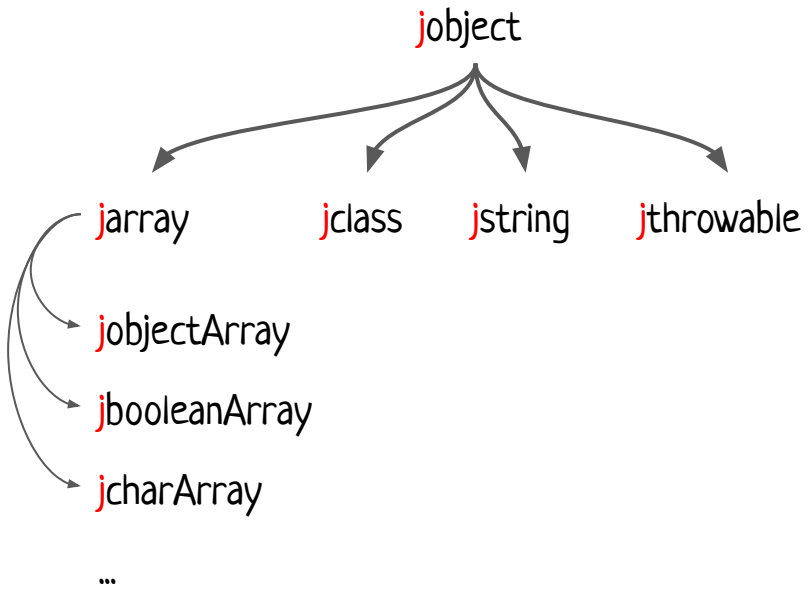
boolean	⟹	jboolean	unsigned 8 bits
byte	⟹	jbyte	signed 8 bits
char	⟹	jchar	unsigned 16 bits
short	⟹	jshort	signed 16 bits
int	⟹	jint	signed 32 bits
long	⟹	jlong	signed 64 bits
float	⟹	jfloat	32 bits
double	⟹	jdouble	64 bits
void	⟹	void	N/A

Как это выглядит со стороны C?

Примитивные

boolean	⟹	jboolean	unsigned 8 bits
byte	⟹	jbyte	signed 8 bits
char	⟹	jchar	unsigned 16 bits
short	⟹	jshort	signed 16 bits
int	⟹	jint	signed 32 bits
long	⟹	jlong	signed 64 bits
float	⟹	jfloat	32 bits
double	⟹	jdouble	64 bits
void	⟹	void	N/A

Ссылочные



Как это выглядит со стороны C?

```
public class JavaToNative {

    static native void goThere(Callback andBackAgain);

}
```

javac JavaToNative.java -h .

JavaToNative.h

```
JNIEXPORT void JNICALL Java_JavaToNative_goThere
  (JNIEnv *, jclass, jobject);
```

Как это выглядит со стороны C?

JNIEnv

Как это выглядит со стороны C?

JNIEnv

- ✓ Указатель на JNINativeInterface (214 функций)

Как это выглядит со стороны C?

JNIEnv

- ✓ Указатель на JNINativeInterface (214 функций)

GetVersion,
DefineClass,
FindClass,
FromReflectedMethod,
FromReflectedField,
ToReflectedMethod,
GetSuperclass,
IsAssignableFrom,
ToReflectedField,
Throw,
ThrowNew,
ExceptionOccurred,
ExceptionDescribe,
ExceptionClear,
FatalError,
PushLocalFrame,
PopLocalFrame,
NewGlobalRef,

DeleteGlobalRef,
DeleteLocalRef,
IsSameObject,
NewLocalRef,
EnsureLocalCapacity,
AllocObject,
NewObject,
NewObjectV,
NewObjectA,
GetObjectClass,
IsInstanceOf,
GetMethodID,
CallObjectMethod,
CallObjectMethodV,
CallObjectMethodA,
CallBooleanMethod,
CallBooleanMethodV,
CallBooleanMethodA,

CallByteMethod,
CallByteMethodV,
CallByteMethodA,
CallCharMethod,
CallCharMethodV,
CallCharMethodA,
CallShortMethod,
CallShortMethodV,
CallShortMethodA,
CallIntMethod,
CallIntMethodV,
CallIntMethodA,
CallLongMethod,
CallLongMethodV,
CallLongMethodA,
CallFloatMethod,
CallFloatMethodV,
CallFloatMethodA,

CallDoubleMethod,
CallDoubleMethodV,
CallDoubleMethodA,
CallVoidMethod,
CallVoidMethodV,
CallVoidMethodA,
CallNonvirtualObjectMethod,
CallNonvirtualObjectMethodV,
CallNonvirtualObjectMethodA,
CallNonvirtualBooleanMethod,
CallNonvirtualBooleanMethodV,
CallNonvirtualBooleanMethodA,
CallNonvirtualByteMethod,
CallNonvirtualByteMethodV,
CallNonvirtualByteMethodA,
CallNonvirtualCharMethod,
CallNonvirtualCharMethodV,
CallNonvirtualCharMethodA,

CallNonvirtualShortMethod,
CallNonvirtualShortMethodV,
CallNonvirtualShortMethodA,
CallNonvirtualIntMethod,
CallNonvirtualIntMethodV,
CallNonvirtualIntMethodA,
CallNonvirtualLongMethod,
CallNonvirtualLongMethodV,
CallNonvirtualLongMethodA,
CallNonvirtualFloatMethod,
CallNonvirtualFloatMethodV,
CallNonvirtualFloatMethodA,
CallNonvirtualDoubleMethod,
CallNonvirtualDoubleMethodV,
CallNonvirtualDoubleMethodA,
CallNonvirtualVoidMethod,
CallNonvirtualVoidMethodV,
CallNonvirtualVoidMethodA,

GetFieldID,
GetObjectField,
GetBooleanField,
GetByteField,
GetCharField,
GetShortField,
GetIntField,
GetLongField,
GetFloatField,
GetDoubleField,
SetObjectField,
SetBooleanField,
SetByteField,
SetCharField,
SetShortField,
SetIntField,
SetLongField,
SetFloatField,

SetDoubleField,
GetStaticMethodID,
CallStaticObjectMethod,
CallStaticObjectMethodV,
CallStaticObjectMethodA,
CallStaticBooleanMethod,
CallStaticBooleanMethodV,
CallStaticBooleanMethodA,
CallStaticByteMethod,
CallStaticByteMethodV,
CallStaticByteMethodA,
CallStaticCharMethod,
CallStaticCharMethodV,
CallStaticCharMethodA,
CallStaticShortMethod,
CallStaticShortMethodV,
CallStaticShortMethodA,
CallStaticIntMethod,

CallStaticIntMethodV,
CallStaticIntMethodA,
CallStaticLongMethod,
CallStaticLongMethodV,
CallStaticLongMethodA,
CallStaticFloatMethod,
CallStaticFloatMethodV,
CallStaticFloatMethodA,
CallStaticDoubleMethod,
CallStaticDoubleMethodV,
CallStaticDoubleMethodA,
CallStaticVoidMethod,
CallStaticVoidMethodV,
CallStaticVoidMethodA,
GetStaticFieldID,
GetStaticObjectField,
GetStaticBooleanField,
...

Как это выглядит со стороны C?

JNIEnv

- ✓ Указатель на JNINativeInterface (214 функций)

```
NewObject,              CallObjectMethod,
GetObjectClass,         CallStaticObjectMethod

GetObjectField,         Call<PrimType>Method,
SetObjectField,         CallStatic<PrimType>Method

Get<PrimType>Field,     Throw,
Set<PrimType>Field,     ThrowNew,
```

Очень похоже на Reflection!

«мета»-программирование на Java

Как это выглядит со стороны C?

JNIEnv

- ✓ Указатель на JNINativeInterface (214 функций)
- ✓ Только через них можно взаимодействовать с Java

docs.oracle.com/javase/8/docs/technotes/guides/jni/spec/functions.html

Как это выглядит со стороны C?

JavaToNative.c

```
/*
 * Class:     JavaToNative
 * Method:    goThere
 * Signature: (LCallback;)V
 */
JNIEXPORT void JNICALL Java_JavaToNative_goThere(JNIEnv * env, jclass klass,
                                                 jobject andBackAgain) {




}
```

Как это выглядит со стороны C?

JavaToNative.c

```
/*
 * Class:     JavaToNative
 * Method:    goThere
 * Signature: (LCallback;)V
 */
JNIEXPORT void JNICALL Java_JavaToNative_goThere(JNIEnv * env, jclass klass,
                                                 jobject andBackAgain) {
   printf("Ok, we are in Mordor now!\n");



}
```

Как это выглядит со стороны C?

JavaToNative.c

```
/*
 * Class:     JavaToNative
 * Method:    goThere
 * Signature: (LCallback;)V
 */
JNIEXPORT void JNICALL Java_JavaToNative_goThere(JNIEnv * env, jclass klass,
                                                 jobject andBackAgain) {
   printf("Ok, we are in Mordor now!\n");
   jclass cls = (*env)->GetObjectClass(env, andBackAgain);
   jmethodID method = (*env)->GetMethodID(env, cls, "call", "()V");

}
```

Как это выглядит со стороны C?

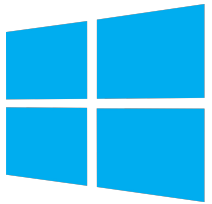
JavaToNative.c

```
/*
 * Class:     JavaToNative
 * Method:    goThere
 * Signature: (LCallback;)V
 */
JNIEXPORT void JNICALL Java_JavaToNative_goThere(JNIEnv * env, jclass klass,
                                                 jobject andBackAgain) {
   printf("Ok, we are in Mordor now!\n");
   jclass cls = (*env)->GetObjectClass(env, andBackAgain);
   jmethodID method = (*env)->GetMethodID(env, cls, "call", "()V");
   (*env)->CallVoidMethod(env, andBackAgain, method);
}
```

Собираем!

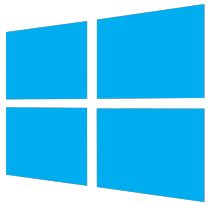
Собираем!

```
gcc -Wall -D_JNI_IMPLEMENTATION_ -Wl,--kill-at -I
%JAVA_HOME%/include -I %JAVA_HOME%/include/win32 -I. -L
%JAVA_HOME%/jre/lib -shared JavaToNative.c -o lib/NativeLib.dll
```



Собираем!

```
gcc -Wall -D_JNI_IMPLEMENTATION_ -Wl,--kill-at -I
%JAVA_HOME%/include -I %JAVA_HOME%/include/win32 -I. -L
%JAVA_HOME%/jre/lib -shared JavaToNative.c -o lib/NativeLib.dll
```

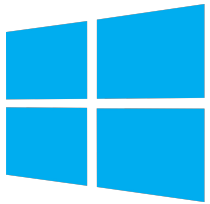


Nokee plugins

- ✓ Кроссплатформенное решение
- ✓ Удобное использование через Gradle
- ✓ nokee.dev

Собираем и запускаем!

```
gcc -Wall -D_JNI_IMPLEMENTATION_ -Wl,--kill-at -I
%JAVA_HOME%/include -I %JAVA_HOME%/include/win32 -I. -L
%JAVA_HOME%/jre/lib -shared JavaToNative.c -o lib/NativeLib.dll
```

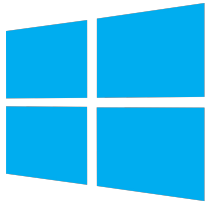


```
public class JavaToNative {
   static native void goThere(Callback andBackAgain);

   public static void main(String[] args) {
      System.loadLibrary("NativeLib");
      goThere(new Callback("Eagles"));
   }
}
```

Собираем и запускаем!

```
gcc -Wall -D_JNI_IMPLEMENTATION_ -Wl,--kill-at -I
%JAVA_HOME%/include -I %JAVA_HOME%/include/win32 -I. -L
%JAVA_HOME%/jre/lib -shared JavaToNative.c -o lib/NativeLib.dll
```



```
public class JavaToNative {
   static native void goThere(Callback andBackAgain);

   public static void main(String[] args) {
      System.loadLibrary("NativeLib");
      goThere(new Callback("Eagles"));
   }
}
```

```
$ java -Djava.library.path=./lib JavaToNative

Ok, we are in Mordor now!
Ok, we are in Shire again! Returned by Eagles
```

Что может пойти не так?

Что может пойти не так?

- ✓ Типовая информация утеряна.
  Добро пожаловать в JavaScript!

```
/*
 * Class:     JavaToNative
 * Method:    goThere
 * Signature: (LCallback;)V
 */
JNIEXPORT void JNICALL Java_JavaToNative_goThere(JNIEnv * env, jclass klass,
                                                 jobject andBackAgain) {
   printf("Ok, we are in Mordor now!\n");
   jclass cls = (*env)->GetObjectClass(env, andBackAgain);
   jmethodID method = (*env)->GetMethodID(env, cls, "call", "()V");
   (*env)->CallVoidMethod(env, andBackAgain, method);
}
```

```
/*
 * Class:     JavaToNative
 * Method:    goThere
 * Signature: (LCallback;Ljava/lang/Object)V
 */
JNIEXPORT void JNICALL Java_JavaToNative_goThere(JNIEnv * env, jclass klass,
                                                 jobject andBackAgain,
                                                 jobject luggage) {
   printf("Ok, we are in Mordor now!\n");
   jclass cls = (*env)->GetObjectClass(env, andBackAgain);
   jmethodID method = (*env)->GetMethodID(env, cls, "call", "()V");
   (*env)->CallVoidMethod(env, luggage, method);
}
```

```
java -Djava.library.path=./lib JavaToNative
```

```
/*
 * Class:     JavaToNative
 * Method:    goThere
 * Signature: (LCallback;Ljava/lang/Object)V
 */
JNIEXPORT void JNICALL Java_JavaToNative_goThere(JNIEnv * env, jclass klass,
                                                 jobject andBackAgain,
                                                 jobject luggage) {
    printf("Ok, we are in Mordor now!\n");
    j
    j
    (
}
```

```
Ok, we are in Mordor now!
# A fatal error has been detected by the Java Runtime Environment:
#
#  EXCEPTION_ACCESS_VIOLATION (0xc0000005) at pc=0x00007ffb8add70e1, pid=14100, tid=12564
#
# JRE version: OpenJDK Runtime Environment AdoptOpenJDK (14.0.1+7) (build 14.0.1+7)
# Java VM: OpenJDK 64-Bit Server VM AdoptOpenJDK (14.0.1+7, mixed mode, sharing, tiered,
compressed oops
, g1 gc, windows-amd64)
# Problematic frame:
# V  [jvm.dll+0x3970e1]
```

Что может пойти не так?

- ✓ Типовая информация утеряна.
  Добро пожаловать в JavaScript!

- ✓ Вызов правильных JNI функций - ваша ответственность

Что может пойти не так?

- ✓ Типовая информация утеряна.
  Добро пожаловать в JavaScript!
- ✓ Вызов правильных JNI функций - ваша ответственность
- ✓ Исключения из Java не пробрасываются (см. ExceptionOccurred, ExceptionClear)


-Xcheck:jni

```
/*
 * Class:     JavaToNative
 * Method:    goThere
 * Signature: (LCallback;Ljava/lang/Object)V
 */
JNIEXPORT void JNICALL Java_JavaToNative_goThere(JNIEnv * env, jclass klass,
                                                 jobject andBackAgain,
                                                 jobject luggage) {
   printf("Ok, we are in Mordor now!\n");
   jclass cls = (*env)->GetObjectClass(env, andBackAgain);
   jmethodID method = (*env)->GetMethodID(env, cls, "call", "()V");
   (*env)->CallVoidMethod(env, luggage, method);
}
```

```
java -Djava.library.path=./lib JavaToNative
```

```
/*
 * Class:     JavaToNative
 * Method:    goThere
 * Signature: (LCallback;Ljava/lang/Object)V
 */
JNIEXPORT void JNICALL Java_JavaToNative_goThere(JNIEnv * env, jclass klass,
                                                 jobject andBackAgain,
                                                 jobject luggage) {
   printf("Ok, we are in Mordor now!\n");
   jclass cls = (*env)->GetObjectClass(env, andBackAgain);
   jmethodID method = (*env)->GetMethodID(env, cls, "call", "()V");
   (*env)->CallVoidMethod(env, luggage, method);
}
```

```
java -Xcheck:jni -Djava.library.path=./lib JavaToNative
```

```
java -Xcheck:jni -Djava.library.path=./lib JavaToNative
```

```
Ok, we are in Mordor now!
FATAL ERROR in native method: Wrong object class or methodID passed to JNI call
        at JavaToNative.goThere(Native Method)
        at JavaToNative.main(JavaToNative.java:9)
```

Что может пойти не так?

- ✓ Типовая информация утеряна.
  Добро пожаловать в JavaScript!
- ✓ Вызов правильных JNI функций - ваша ответственность
- ✓ Исключения из Java не пробрасываются
- ✓ -Xcheck:jni значительно улучшает диагностику простых случаев
  (но это стоит дорого)

Как позвать натив?

Где взять нативы? ✓

Как работать с Java из натива? ✓

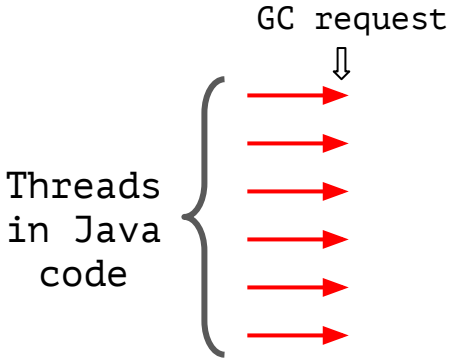
Как себя должен вести GC?

GC в Java коде

- safe-points, в которых "припарковываются" потоки

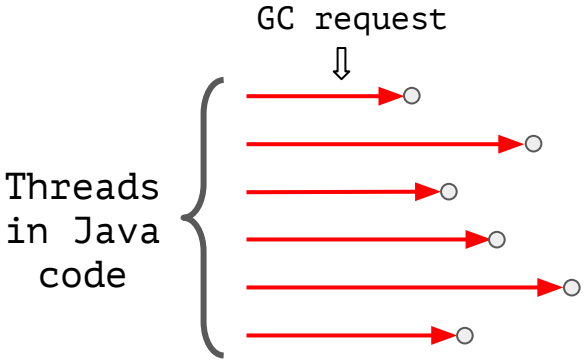
GC в Java коде

- safe-points, в которых "припарковываются" потоки
- GC ждет, пока потоки припаркуются



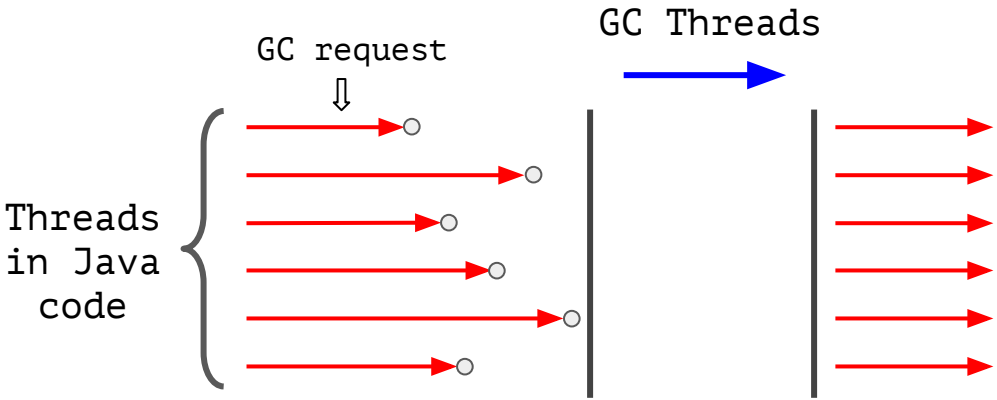
GC в Java коде

- safe-points, в которых "припарковываются" потоки
- GC ждет, пока потоки припаркуются



GC в Java коде

- safe-points, в которых "припарковываются" потоки
- GC ждет, пока потоки припаркуются

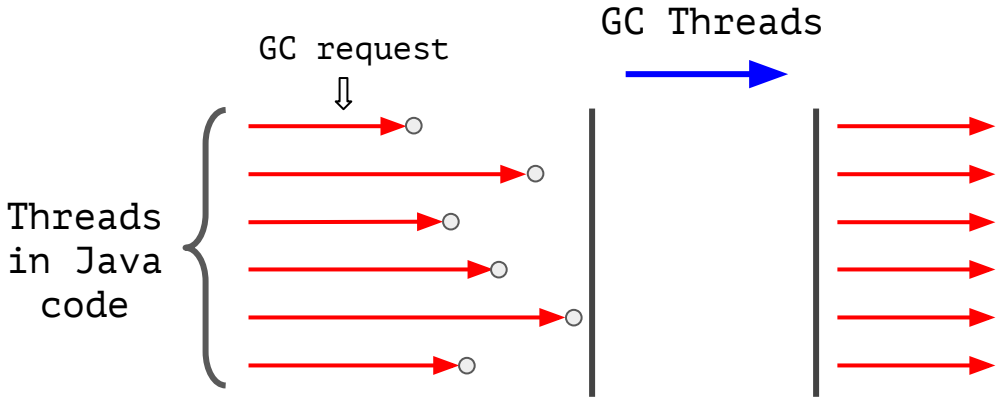


GC в нативном коде

- какие еще safe-points?
- натив продолжает работать

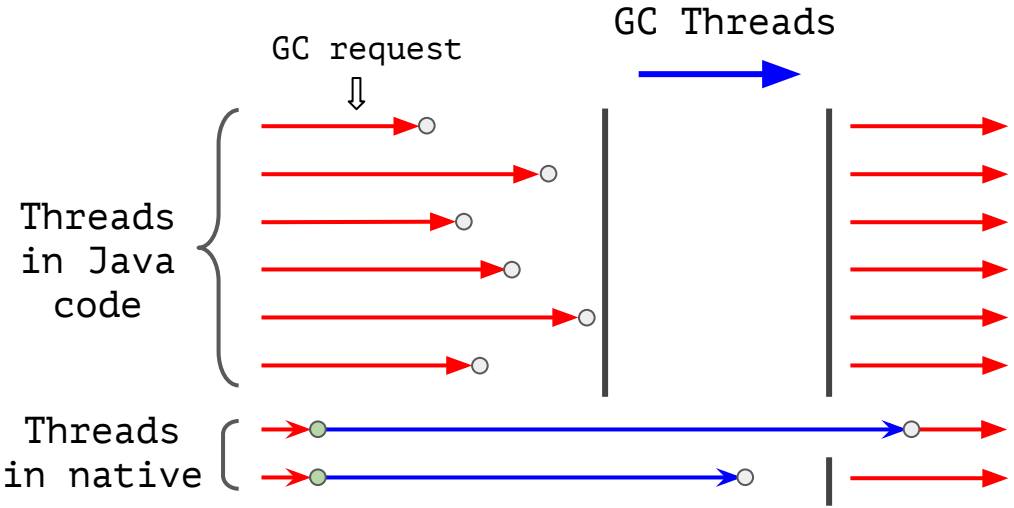
GC в нативном коде

- какие еще safe-points?
- натив продолжает работать



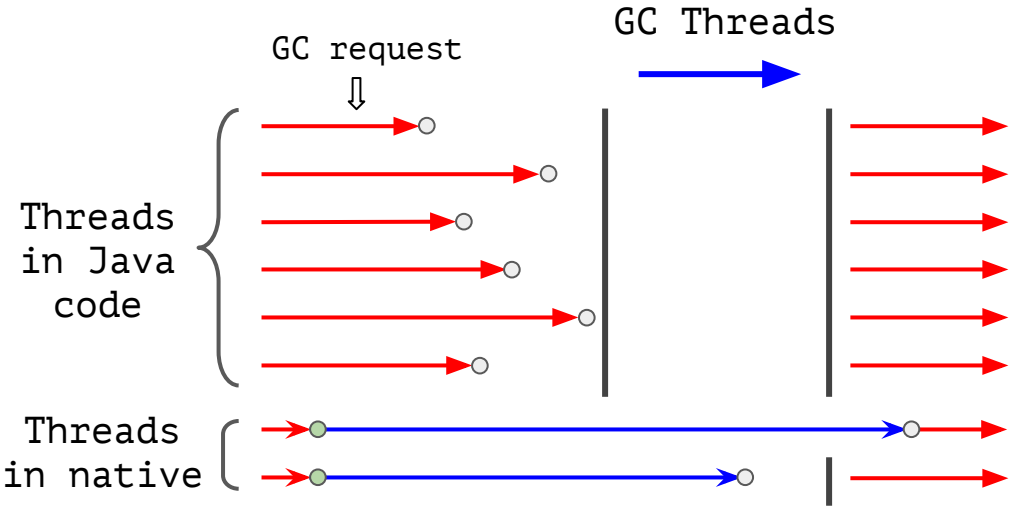
GC в нативном коде

- какие еще safe-points?
- натив продолжает работать



GC в нативном коде

- какие еще safe-points?
- натив продолжает работать
- на входе и выходе из натива синхронизация с GC



Нельзя трогать Java объекты в нативе во время GC!

GC в нативном коде

```java
public class JavaToNative {
    static native void goThere(Callback andBackAgain);
}
```

javac JavaToNative.java -h .

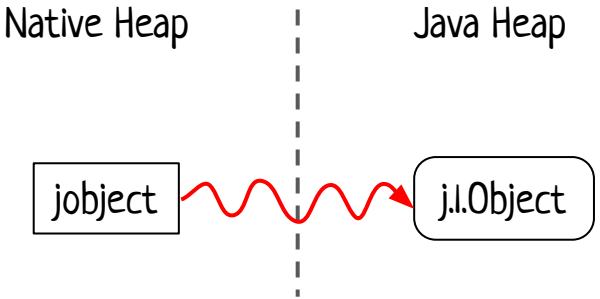
JavaToNative.h

```
JNIEXPORT void JNICALL Java_JavaToNative_goThere
 (JNIEnv *, jclass, jobject);
```

GC в нативном коде

jobject и компания - специальные хендлы, для которых:

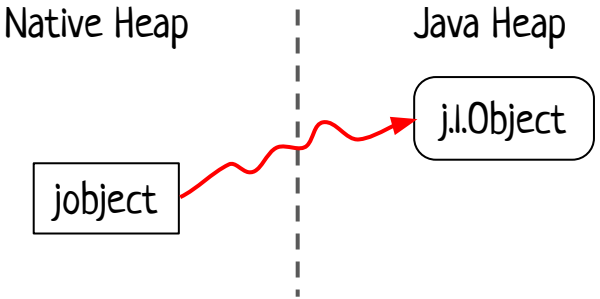
1. JVM поддерживает связь с реальными Java объектами



GC в нативном коде

jobject и компания - специальные хендлы, для которых:

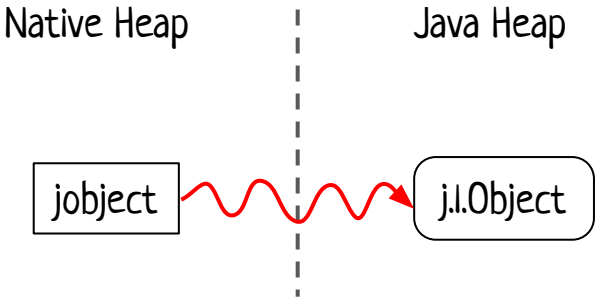
1. JVM поддерживает связь с реальными Java объектами



GC в нативном коде

jobject и компания - специальные хендлы, для которых:

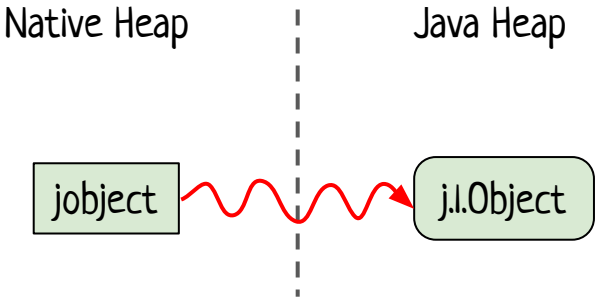
1. JVM поддерживает связь с реальными Java объектами
2. Доступ к реальным Java объектам (через JNI) синхронизирован с GC



GC в нативном коде

jobject и компания - специальные хендлы, для которых:

1. JVM поддерживает связь с реальными Java объектами
2. Доступ к реальным Java объектам (через JNI) синхронизирован с GC
3. Содержимое считается GC-roots!




Garbage Collector
jobject
jobject
jobject

Подводные камни (GC + Natives)

- ✓ Для хендлов реализована альтернативная система управления памятью

Подводные камни (GC + Natives)

- ✓ Для хендлов реализована альтернативная система управления памятью

  Типы хендлов (в коде все выглядит, как jobject & Co):

Подводные камни (GC + Natives)

- ✓ Для хендлов реализована альтернативная система управления памятью

  Типы хендлов (в коде все выглядит, как jobject & Co):

  1. Local Reference

Подводные камни (GC + Natives)

Local Reference:

1. Существует не дольше, чем исполняется натив
2. Аргументы нативов и возвращаемое значение многих JNI функций

Подводные камни (GC + Natives)

Local Reference:

1. Существует не дольше, чем исполняется натив
2. Аргументы нативов и возвращаемое значение многих JNI функций
3. Отличный источник утечек памяти!

Подводные камни (GC + Natives)

```
JNIEXPORT void JNICALL Java_JavaToNative_objectsAllocationTest
                                                (JNIEnv *env, jclass klass) {
    int ready = 0, id = 0;
    jclass cls = (*env)->FindClass(env, "BornInNative");
    jmethodID init = (*env)->GetMethodID(env, cls, "<init>", "(I)V");
    jmethodID check = (*env)->GetMethodID(env, cls, "areYouReady", "()Z");

    while (!ready) {
        jobject obj = (*env)->NewObject(env, cls, init, id++);
        ready = (*env)->CallBooleanMethod(env, obj, check) == JNI_TRUE;
    }
    printf("finally ready after %d objects created!\n", id);
}
```

Подводные камни (GC + Natives)

```
JNIEXPORT void JNICALL Java_JavaToNative_objectsAllocationTest
                                                (JNIEnv *env, jclass klass) {
    int ready = 0, id = 0;
    jclass cls = (*env)->FindClass(env, "BornInNative");
    jmethodID init = (*env)->GetMethodID(env, cls, "<init>", "(I)V");
    jmethodID check = (*env)->GetMethodID(env, cls, "areYouReady", "()Z");

    while (!ready) {
        jobject obj = (*env)->NewObject(env, cls, init, id++);
        ready = (*env)->CallBooleanMethod(env, obj, check) == JNI_TRUE;
    }
    printf("finally ready after %d objects created!\n", id);
}
```

Подводные камни (GC + Natives)

```
JNIEXPORT void JNICALL Java_JavaToNative_objectsAllocationTest
                                                (JNIEnv *env, jclass klass) {
   int ready = 0, id = 0;
   jclass cls = (*env)->FindClass(env, "BornInNative");
   jmethodID init = (*env)->GetMethodID(env, cls, "<init>", "(I)V");
   jmethodID check = (*env)->GetMethodID(env, cls, "areYouReady", "()Z");

   while (!ready) {
       jobject obj = (*env)->NewObject(env, cls, init, id++);
       ready = (*env)->CallBooleanMethod(env, obj, check) == JNI_TRUE;
   }
   printf("finally ready after %d objects created!\n", id);
}
```

Подводные камни (GC + Natives)

```
JNIEXPORT void JNICALL Java_JavaToNative_objectsAllocationTest
                                                (JNIEnv *env, jclass klass) {
    int ready = 0, id = 0;
    jclass cls = (*env)->FindClass(env, "BornInNative");
    jmethodID init = (*env)->GetMethodID(env, cls, "<init>", "(I)V");
    jmethodID check = (*env)->GetMethodID(env, cls, "areYouReady", "()Z");

    while (!ready) {
        jobject obj = (*env)->NewObject(env, cls, init, id++);
        ready = (*env)->CallBooleanMethod(env, obj, check) == JNI_TRUE;
    }
    printf("finally ready after %d objects created!\n", id);
}
```

Подводные камни (GC + Natives)

```
JNIEXPORT void JNICALL Java_JavaToNative_objectsAllocationTest
                                                (JNIEnv *env, jclass klass) {
    int ready = 0, id = 0;
    jclass cls = (*env)->FindClass(env, "BornInNative");
    jmethodID init = (*env)->GetMethodID(env, cls, "<init>", "(I)V");
    jmethodID check = (*env)->GetMethodID(env, cls, "areYouReady", "()Z");

    while (!ready) {
        jobject obj = (*env)->NewObject(env, cls, init, id++);
        if (!obj) {
            printf("allocation attempt %d failed\n", id);
        }
        ready = (*env)->CallBooleanMethod(env, obj, check) == JNI_TRUE;
    }
    printf("finally ready after %d objects created!\n", id);
}
```

Подводные камни (GC + Natives)

```
JNIEXPORT void JNICALL Java_JavaToNative_objectsAllocationTest
                                            (JNIEnv *env, jclass klass) {
   int ready = 0, id = 0;
   jclass cls = (*env)->FindClass(env, "BornInNative");
   jmethodID init = (*env)->GetMethodID(env, cls, "<init>", "(I)V");
   jmethodID check = (*env)->GetMethodID(env, cls, "areYouReady", "()Z");

   while (!ready) {
       jobject obj = (*env)->NewObject(env, cls, init, id++);
       if (!obj) {
           printf("allocation attempt %d failed\n", id);
       }
       ready = (*env)->CallBooleanMethod(env, obj, check) == JNI_TRUE;
   }
   printf("finally ready af
}
```

```
public static void main(String[] args) {
    System.loadLibrary("NativeLib");
    objectsAllocationTest();
}
```

```
java -Xmx1G -Djava.library.path=./lib JavaToNative
```

Подводные камни (GC + Natives)

```
JNIEXPORT void JNICALL Java_JavaToNative_objectsAllocationTest
                                              (JNIEnv *env, jclass klass) {
    int ready = 0, id = 0;
    jclass cls = (*env)->FindClass(env, "BornInNative");
    jmethodID init = (*env)->GetMethodID(env, cls, "<init>", "(I)V");
    jmethodID check = (*env)->GetMethodID(env, cls, "areYouReady", "()Z");

    while (!ready) {
        jobject obj = (*env)->NewObject(env, cls, init, id++);
        if (!obj) {
            printf("allocation attempt %d failed\n", id);
        }
        ready = (*env)->CallBooleanMethod(env, obj, check) == JNI_TRUE;
    }
    printf("finally ready af
}
```

```
public static void main(String[] args) {
    System.loadLibrary("NativeLib");
    objectsAllocationTest();
}
```

```
allocation attempt 67065480 failed
allocation attempt 67065481 failed
allocation attempt 67065482 failed
allocation attempt 67065483 failed
allocation attempt 67065484 failed
allocation attempt 67065485 failed
allocation attempt 67065486 failed
allocation attempt 67065487 failed
allocation attempt 67065488 failed
allocation attempt 67065489 failed
```

PID	CPU	Private bytes
11860	69,23	2,1 GB

```
java -Xmx1G -Djava.library.path=./lib JavaToNative
```

Подводные камни (GC + Natives)

```
JNIEXPORT void JNICALL Java_JavaToNative_objectsAllocationTest
                                                (JNIEnv *env, jclass klass) {
   int ready = 0, id = 0;
   jclass cls = (*env)->FindClass(env, "BornInNative");
   jmethodID init = (*env)->GetMethodID(env, cls, "<init>", "(I)V");
   jmethodID check = (*env)->GetMethodID(env, cls, "areYouReady", "()Z");

   while (!ready) {
       jobject obj = (*env)->NewObject(env, cls, init, id++);
       if (!obj) {
           printf("allocation attempt %d failed\n", id);
       }
       ready = (*env)->CallBooleanMethod(env, obj, check) == JNI_TRUE;
       (*env)->DeleteLocalRef(env, obj);
   }
   printf("finally ready after %d objects created!\n", id);
}
```

Подводные камни (GC + Natives)

- ✓ Для хендлов реализована альтернативная система управления памятью

  Типы хендлов (в коде все выглядит, как jobject & Co):

  1. Local Reference

живут не дольше одного нативного вызова

Подводные камни (GC + Natives)

✓ Для хендлов реализована альтернативная система управления памятью

Типы хендлов (в коде все выглядит, как jobject & Co):

1. Local Reference
2. Global Reference
3. Weak Global Reference

живут не дольше одного нативного вызова

живут, пока их явно не освободят

живут, пока их явно не освободят, но GC может собрать Java объект

Подводные камни (GC + Natives)

- ✓ Особая система управления памятью (в которой легко получить memory leak/dangling pointer)

Подводные камни (GC + Natives)

- ✓ Особая система управления памятью (в которой легко получить memory leak/dangling pointer)
- ✓ Особая обработка массивов и строк: pinning vs copying

```
jint* GetIntArrayElements(JNIEnv* env, jintArray array, jboolean* isCopy);

void ReleaseIntArrayElements(JNIEnv* env, jintArray array, jint* elems, jint mode);
```

Подводные камни (GC + Natives)

- ✓ Особая система управления памятью (в которой легко получить memory leak/dangling pointer)
- ✓ Особая обработка массивов и строк: pinning vs copying

```
jint* GetIntArrayElements(JNIEnv* env, jintArray array, jboolean* isCopy);

void ReleaseIntArrayElements(JNIEnv* env, jintArray array, jint* elems, jint mode);
```

на самом деле
копирование происходит всегда

(для большинства JVM/GC)

Подводные камни (GC + Natives)

- ✓ Особая система управления памятью (в которой легко получить memory leak/dangling pointer)
- ✓ Особая обработка массивов и строк: pinning vs copying
- ✓ Опасные JNI методы Get*Critical (могут сильно помешать работе GC вплоть до зависания JVM)

shipilev.net/jvm/anatomy-quarks/9-jni-critical-gclocker

Производительность нативных вызовов

Производительность нативных вызовов

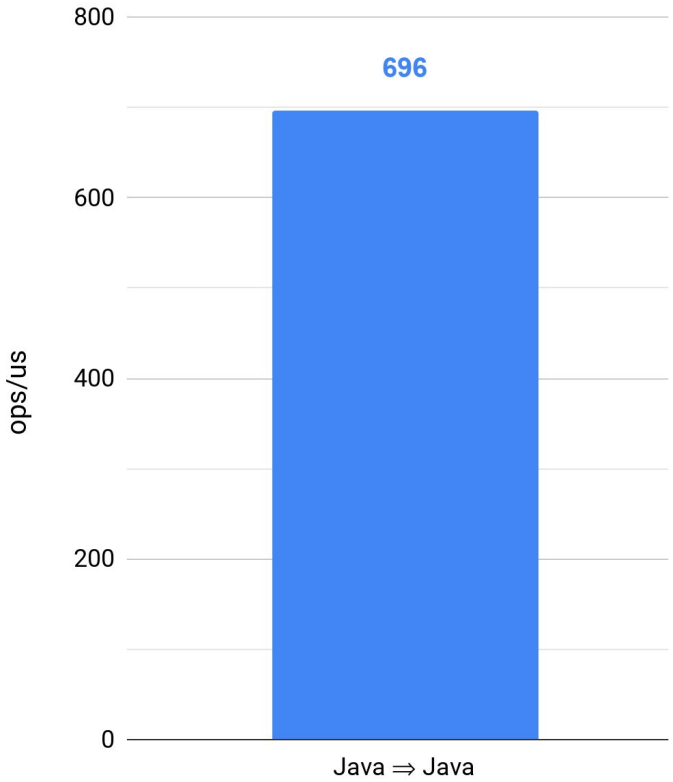
ВСЕ ОЧЕНЬ МЕДЛЕННО

Производительность нативных вызовов

ВСЕ ОЧЕНЬ МЕДЛЕННО

Измерять будем на:
Intel Core i7-7700 @ 3.60 GHz; 16GB RAM; Linux Ubuntu 18.04

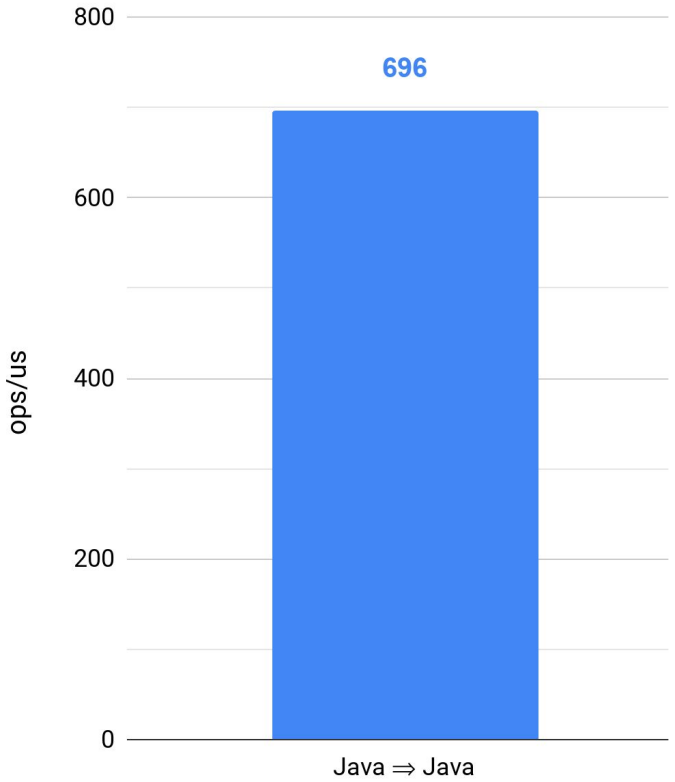
Производительность нативных вызовов



Java ⟹ Java

прямой вызов Java метода
без инлайна

Производительность нативных вызовов



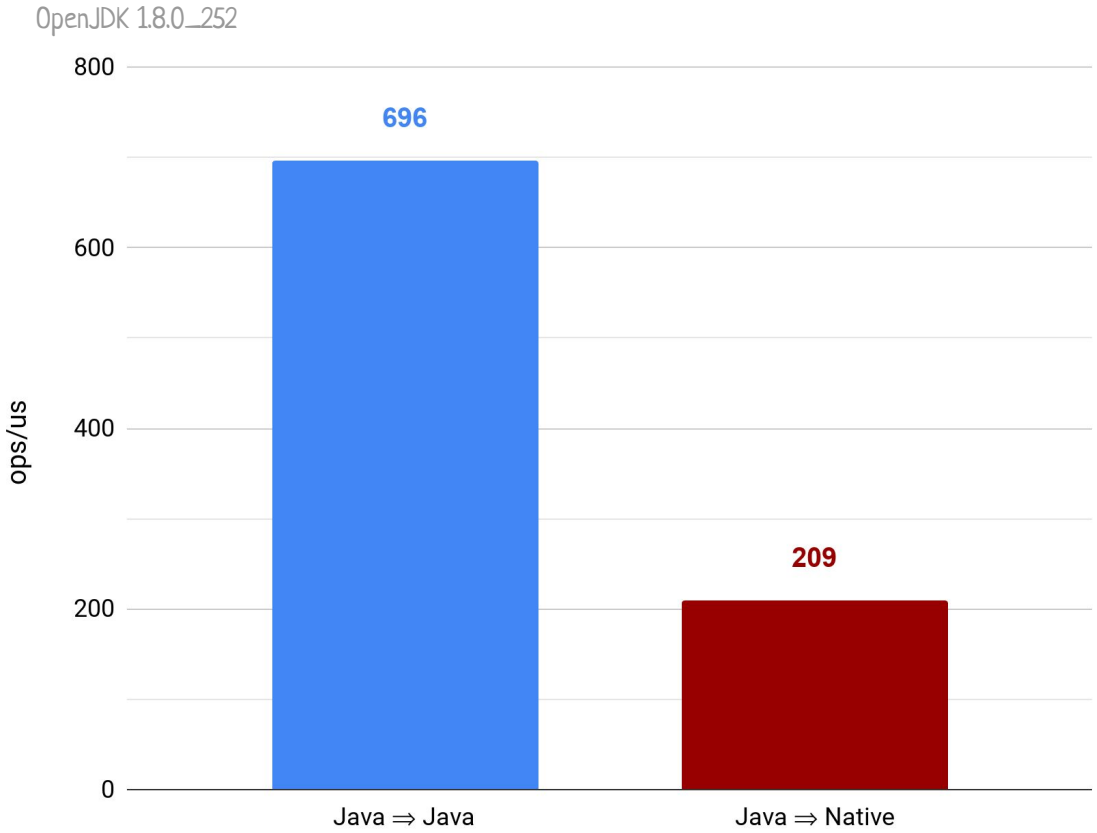
Java ⟹ Java

прямой вызов Java метода
без инлайна

Java ⟹ Native

вызов нативного метода
(без параметров) из Java

Производительность нативных вызовов



Java ⟹ Java

прямой вызов Java метода
без инлайна

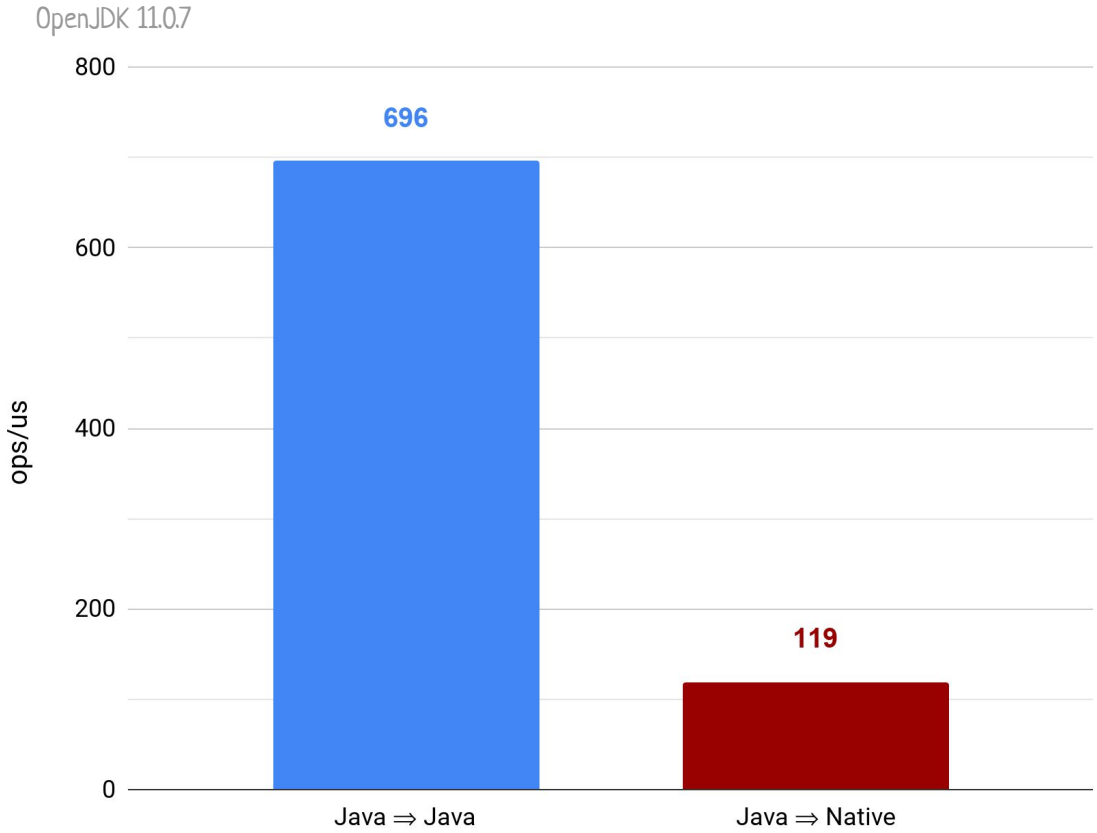
Java ⟹ Native

вызов нативного метода
(без параметров) из Java

Разница в 3.3 раза

На jdk8u252

Производительность нативных вызовов



Java ⟹ Java

прямой вызов Java метода
без инлайна

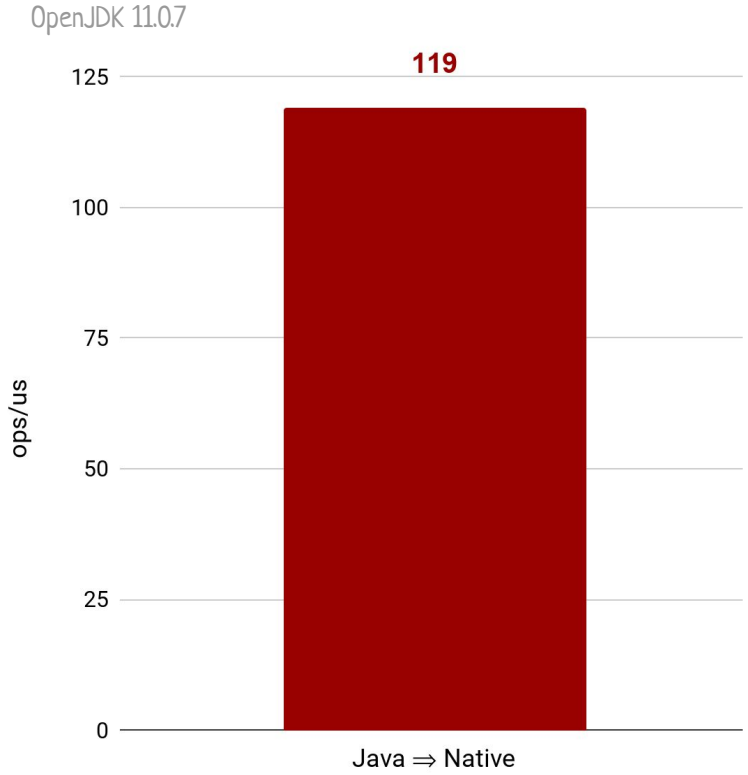
Java ⟹ Native

вызов нативного метода
(без параметров) из Java

Разница в 6 раз

На jdk-11.0.7

Производительность нативных вызовов



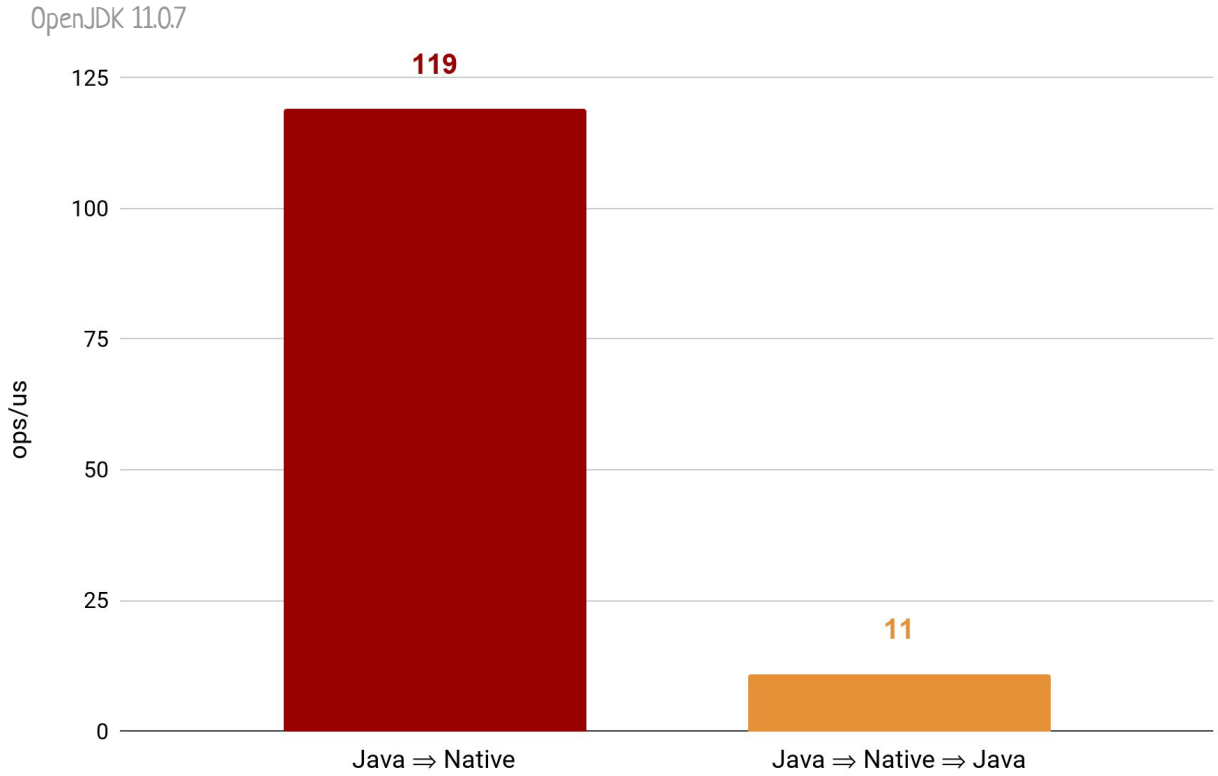
Java ⟹ Native

Вызов нативного метода (без параметров) из Java

Java ⟹ Native ⟹ Java

Вызов нативного метода (без параметров) из Java, а из него вызов Java метода (без параметров)

Производительность нативных вызовов



Java ⟹ Native

Вызов нативного метода (без параметров) из Java

Java ⟹ Native ⟹ Java

Вызов нативного метода (без параметров) из Java, а из него вызов Java метода (без параметров)

Разница в 10 раз

Производительность нативных вызовов

ВСЕ ОЧЕНЬ МЕДЛЕННО

...почему?

```
static native void goNative();
```

```
call    0x00007f4882317c70
```

```
static native void goNative();
```

```
mov    DWORD PTR [rsp-0x14000],eax
push   rbp
mov    rbp,rsp
sub    rsp,0x40
movabs r14,0x76d381c90;
mov    QWORD PTR [rsp+0x30],r14
lea    r14,[rsp+0x30]
mov    rsi,r14
movabs r10,0x7f486d4481fe
mov    QWORD PTR [r15+0x1c8],r10
mov    QWORD PTR [r15+0x1c0],rsp
cmp    BYTE PTR [rip+0x154b0527],0x0
je     0x00007f486d448255
push   rsi
movabs rsi,0x7f486b1433f8
mov    rdi,r15
test   esp,0xf
je     0x00007f486d44824f
sub    rsp,0x8
call   0x00007f4882317be0
add    rsp,0x8
jmp    0x00007f486d448254
call   0x00007f4882317be0
```

```
pop    rsi
lea    rdi,[r15+0x1e0]
mov    DWORD PTR [r15+0x258],0x4
call   0x00007f486073a79a
vzeroupper
mov    DWORD PTR [r15+0x258],0x5
mov    ecx,r15d
shr    ecx,0x4
and    ecx,0xffc
movabs r10,0x7f4883580000
mov    DWORD PTR [r10+rcx*1],ecx
cmp    DWORD PTR [rip+0x154bbf5e],0x0
jne    0x00007f486d4482b2
cmp    DWORD PTR [r15+0x30],0x0
je     0x00007f486d4482cb
call   0x00007f4882317be0
add    rsp,0x8
jmp    0x00007f486d448254
call   0x00007f4882317be0
pop    rsi
lea    rdi,[r15+0x1e0]
mov    DWORD PTR [r15+0x258],0x4
call   0x00007f486073a79a
```

```
mov    DWORD PTR [r15+0x258],0x4
call   0x00007f486073a79a
vzeroupper
mov    DWORD PTR [r15+0x258],0x5
mov    ecx,r15d
shr    ecx,0x4
and    ecx,0xffc
movabs r10,0x7f4883580000
mov    DWORD PTR [r10+rcx*1],ecx
cmp    DWORD PTR [rip+0x154bbf5e],0x0
jne    0x00007f486d4482b2
cmp    DWORD PTR [r15+0x30],0x0
je     0x00007f486d4482cb
mov    rdi,r15
mov    r12,rsp
sub    rsp,0x0
and    rsp,0xfffffffffffffff0
call   0x00007f48823b84f0
mov    rsp,r12
xor    r12,r12
mov    DWORD PTR [r15+0x258],0x8
cmp    DWORD PTR [r15+0x284],0x1
je     0x00007f486d448369
```

```
cmp    BYTE PTR [rip+0x154b0456],0x0
je     0x00007f486d448324
movabs rsi,0x7f486b1433f8
mov    rdi,r15
test   esp,0xf
je     0x00007f486d44831f
sub    rsp,0x8
call   0x00007f4882317c70
add    rsp,0x8
jmp    0x00007f486d448324
call   0x00007f4882317c70
movabs r10,0x0
mov    QWORD PTR [r15+0x1c0],r10
movabs r10,0x0
mov    QWORD PTR [r15+0x1c8],r10
mov    rcx,QWORD PTR [r15+0x38]
mov    DWORD PTR [rcx+0x100],0x0
leave
cmp    QWORD PTR [r15+0x8],0x0
jne    0x00007f486d448364
ret
jmp    Stub::forward exception
```

Производительность нативных вызовов

- ✓ State transition (Java -> Native и Native -> Java) очень дорог:
    1. Синхронизация с GC
    2. Завертка параметров в Local References
    3. Обработка результата + exception check
    4. Перепушивание параметров

Производительность нативных вызовов

- ✓ State transition (Java -> Native и Native -> Java) очень дорог:
    1. Синхронизация с GC
    2. Завертка параметров в Local References
    3. Обработка результата + exception check
    4. Перепушивание параметров

- ✓ Никакого инлайнинга!

- ✓ Особенности реализации Hotspot (стабы для перехода Native -> Java)

Takeaways про JNI

1. `javac -h` для генерации .h файлов
2. `nokee.dev` для сборки
3. `-Xcheck:jni` для отлавливания ошибок
4. Осторожнее с `JNI References` и с `JNI Get*Critical`
5. Переход в натив (и возврат в Java) очень дорогой

JNI – хорошо, но больно

А МОЖЕТ НЕ НУЖНО ПИСАТЬ КОД НА C/C++?

Новейшее время

Идея:

- ✓ Весь код писать на Java, а связь с нативом генерировать автоматически

Новейшее время

Идея:

- ✓ Весь код писать на Java, а связь с нативом генерировать автоматически

Реализация:

- ✓ Библиотеки: JNA, JNR, JavaCPP, ...

JNA

(Java Native Access)

```
<dependency>
  <groupId>net.java.dev.jna</groupId>
  <artifactId>jna</artifactId>
  <version>5.5.0</version>
</dependency>
```

JNA

MyNativeLib.c

```
__declspec(dllexport) void __stdcall sayHello(const char* name) {
     printf("Hello %s from native!\n", name);
}
```

JNA

MyNativeLib.c

```
__declspec(dllexport) void __stdcall sayHello(const char* name) {
     printf("Hello %s from native!\n", name);
}
```

TestJNA.java

```
public interface MyNativeLibrary extends Library {
   MyNativeLibrary INSTANCE = (MyNativeLibrary)
                        Native.load("MyNativeLib", MyNativeLibrary.class);
   void sayHello(String name);
}
```

```
MyNativeLibrary.INSTANCE.sayHello("JPoint");
```

JNA

MyNativeLib.c

```
__declspec(dllexport) void __stdcall sayHello(const char* name) {
     printf("Hello %s from native!\n", name);
}
```

TestJNA.java

```
public interface MyNativeLibrary extends Library {
   MyNativeLibrary INSTANCE = (MyNativeLibrary)
                        Native.load("MyNativeLib", MyNativeLibrary.class);
   void sayHello(String name);
}
```

```
MyNativeLibrary.INSTANCE.sayHello("JPoint");
```

JNA

MyNativeLib.c

```
__declspec(dllexport) void __stdcall sayHello(const char* name) {
    printf("Hello %s from native!\n", name);
}
```

TestJNA.java

```
public interface MyNativeLibrary extends Library {
   MyNativeLibrary INSTANCE = (MyNativeLibrary)
                        Native.load("MyNativeLib", MyNativeLibrary.class);
   void sayHello(String name);
}
```

```
$ java -Djava.library.path=./lib TestJNA

Hello JPoint from native!
```

```
MyNativeLibrary.INSTANCE.sayHello("JPoint");
```

JNA

Поддерживается:

- ✓ Передача, возврат по значению
- ✓ Указатели, C-like массивы, C-like строки
- ✓ Указатели на функции
- ✓ Struct & Union
- ✓ varargs

JNA

Самое вкусное:

- ✓ Заготовлены Java описания для многих популярных C библиотек
  - LibC, X11, udev, ...
  - Kernel32, Pdh, Psapi, ...

github.com/java-native-access/jna#jna-platform

JNA

```
WinBase.SYSTEMTIME systemTime = new WinBase.SYSTEMTIME();
Kernel32.INSTANCE.GetSystemTime(systemTime);
System.out.println("System time: " + systemTime);
```

```
System time: 24 мая 2020 г., 14:08:03
```

JNA

```
WinBase.SYSTEMTIME systemTime = new WinBase.SYSTEMTIME();
Kernel32.INSTANCE.GetSystemTime(systemTime);
System.out.println("System time: " + systemTime);
```

```
System time: 24 мая 2020 г., 14:08:03
```

```
Psapi.PERFORMANCE_INFORMATION info = new Psapi.PERFORMANCE_INFORMATION();
Psapi.INSTANCE.GetPerformanceInfo(info, 104);
System.out.println("commited mem = " + info.CommitTotal);
System.out.println("physical mem = " + info.PhysicalTotal);
```

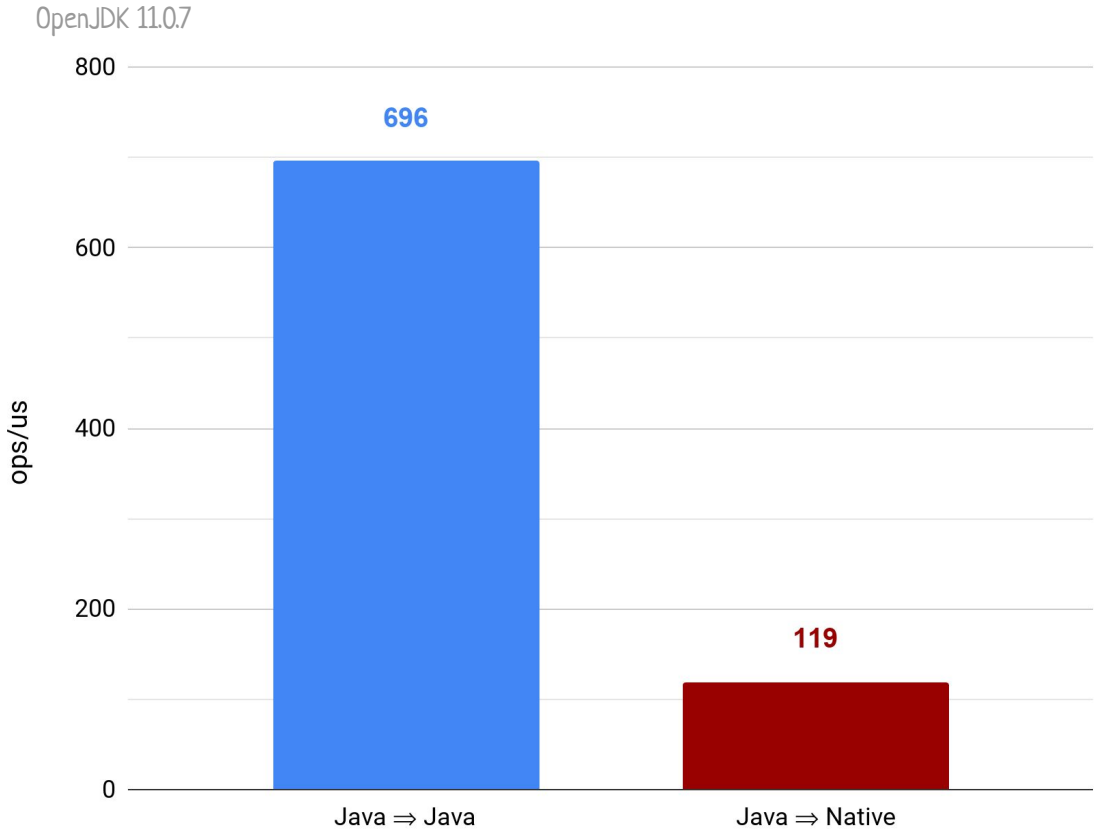
```
commited mem = 3357974
physical mem = 2069169
```

JNA - подводные камни

JNA - подводные камни

ВСЁ ЕЩЁ МЕДЛЕННЕЕ

JNA - подводные камни

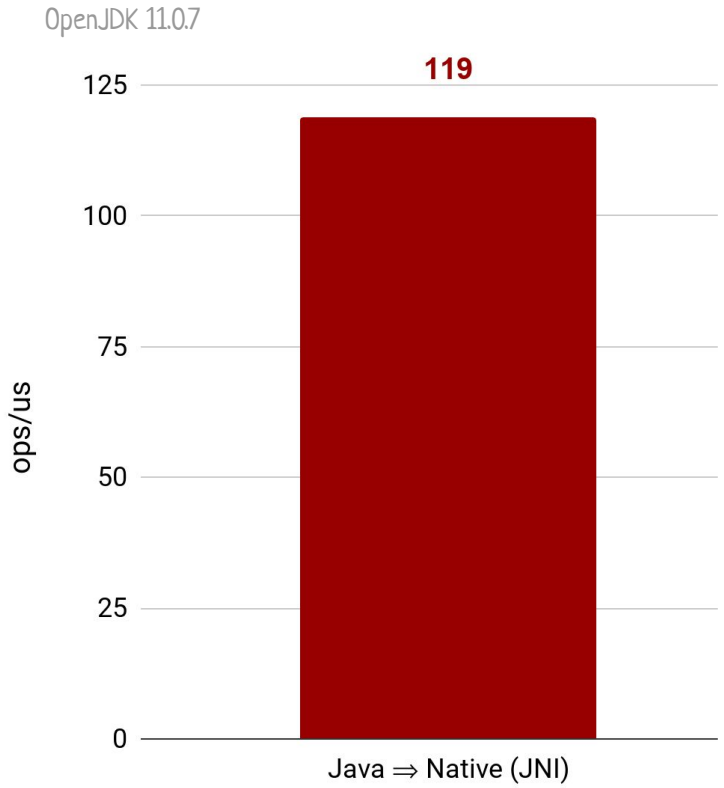


Java ⇒ Java – прямой вызов Java метода без инлайна

Java ⇒ Native – вызов нативного метода (без параметров) из Java

Разница в 6 раз

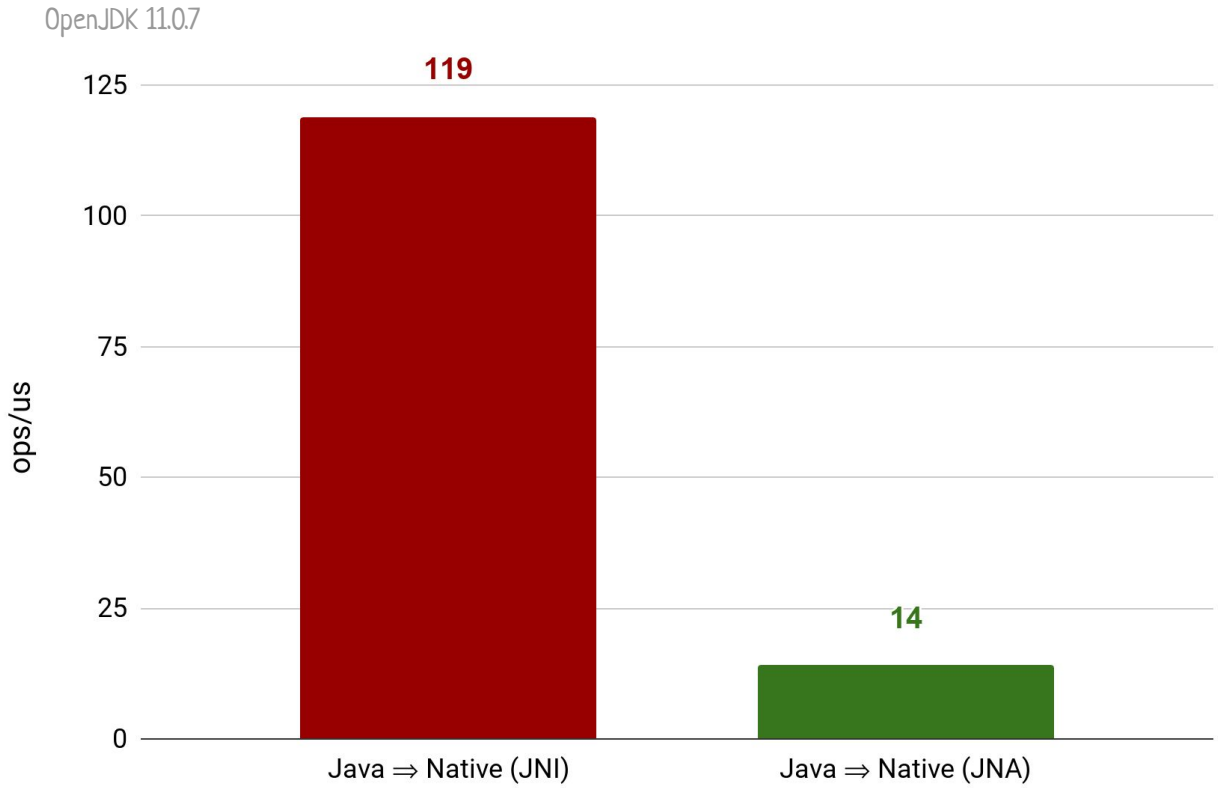
JNA - подводные камни



Java ⇒ Native (JNI) - вызов нативного метода (без параметров) из Java

Java ⇒ Native (JNA) - вызов нативного метода чрез JNA

JNA - подводные камни



Java ⇒ Native (JNI) – вызов нативного метода (без параметров) из Java

Java ⇒ Native (JNA) – вызов нативного метода чрез JNA

Разница в 8.5 раз
(в 50 раз от вызова Java версии)

Производительность JNA

- ✓ Все базируется на JNI, поэтому быстрее быть точно не может

Как позвать натив?

Где взять нативы?

Как работать с Java из натива?

Как себя должен вести GC?

Как позвать натив?

Где взять нативы?

Как работать с Java из натива?

Как себя должен вести GC?

Производительность JNA

- ✓ Все базируется на JNI, поэтому быстрее быть точно не может
- ✓ На стороне Java - Reflection, на стороне натива огромный dispatch

```
MyNativeLibrary.INSTANCE.sayHello("JPoint");
                ⇓
com.sun.jna.Function.invoke(...)
                ⇓
com.sun.jna.Natives.invokeVoid(...)
                ⇓
void __stdcall sayHello(const char* name)
```

JNA - подводные камни

- ✓ JNA мусорит Java обертками вокруг нативых сущностей (`Pointer`, `Memory`, `ByReference`)
  - Нагрузка на GC,
  - Ухудшение производительности,

JNA - подводные камни

- ✓ JNA мусорит Java обертками вокруг нативых сущностей (`Pointer`, `Memory`, `ByReference`)
  - Нагрузка на GC,
  - Ухудшение производительности,
  - И даже некорректное поведение!

```
com.sun.jna
public class Memory
extends Pointer
```

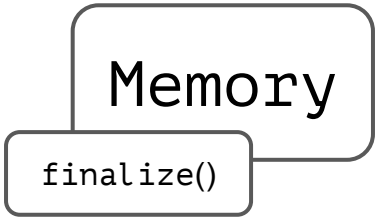
A `Pointer` to memory obtained from the native heap via a call to `malloc`.
In some cases it might be necessary to use memory obtained from `malloc`.
For example, `Memory` helps accomplish the following idiom:

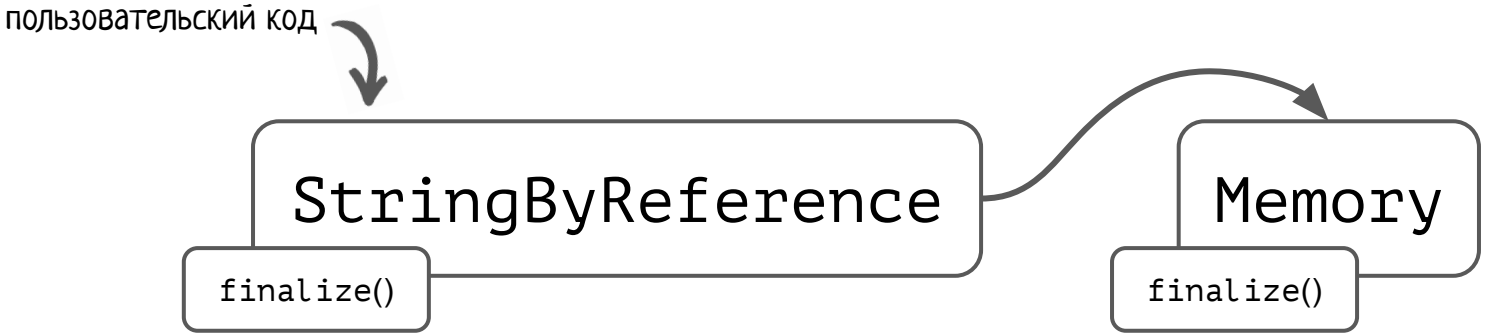
```
void *buf = malloc(BUF_LEN * sizeof(char));
call_some_function(buf);
free(buf);
```

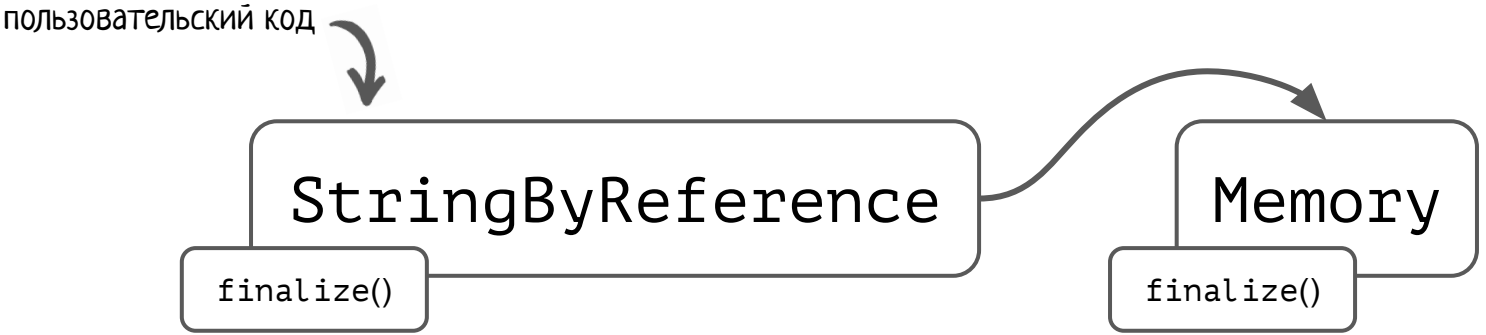
The `finalize` method will free allocated memory when this object is no longer referenced.

See Also: `Pointer`

jna

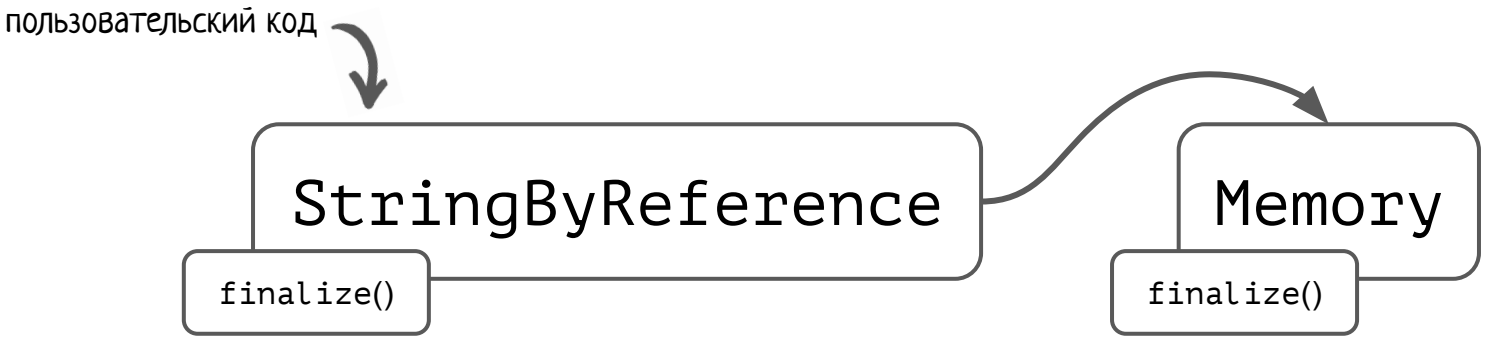

Memory
finalize()


пользовательский код
StringByReference
finalize()
Memory
finalize()




```
if (addr != null) {
    GlobalFree(addr.getPointer(0));
}
```

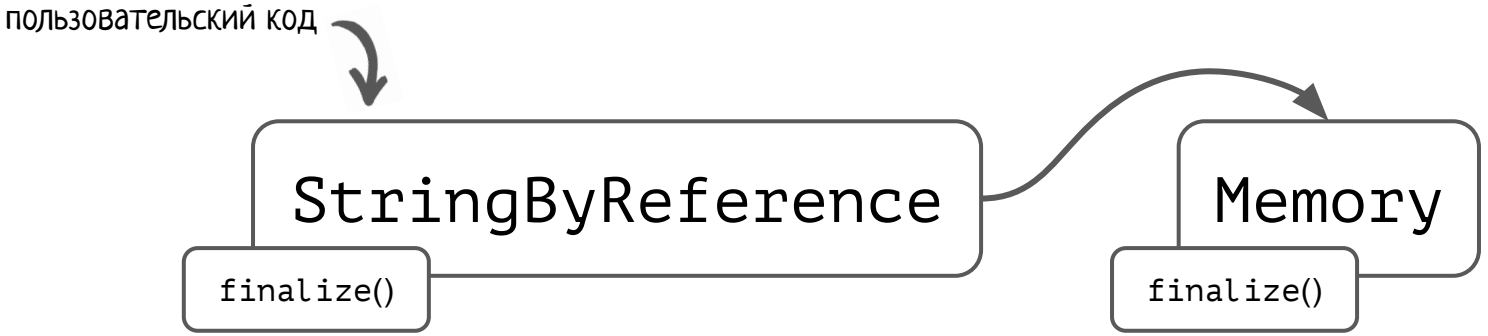
```
free(addr);
addr = null;
```




```
if (addr != null) {
    GlobalFree(addr.getPointer(0));
}
```

```
free(addr);
addr = null;
```

Чей финализатор вызовется первым?

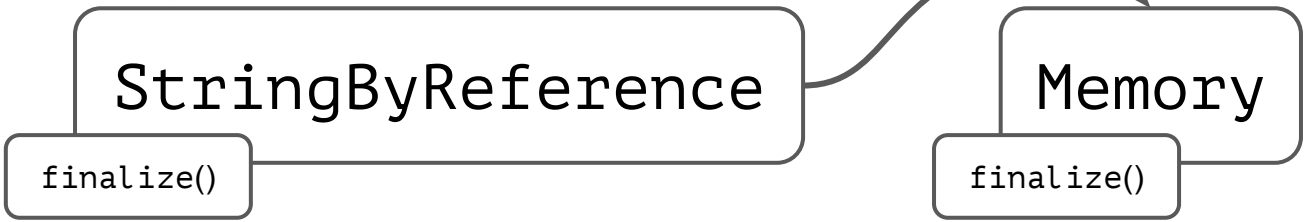



```
if (addr != null) {
    GlobalFree(addr.getPointer(0));
}
```

```
free(addr);
addr = null;
```

случился первым

пользовательский код



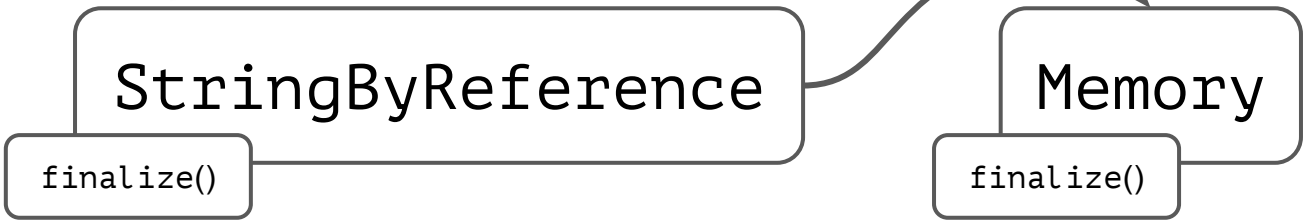

```
if (addr != null) {
    GlobalFree(addr.getPointer(0));
}
```

утечка памяти!

```
free(addr);
addr = null;
```

случился первым

пользовательский код

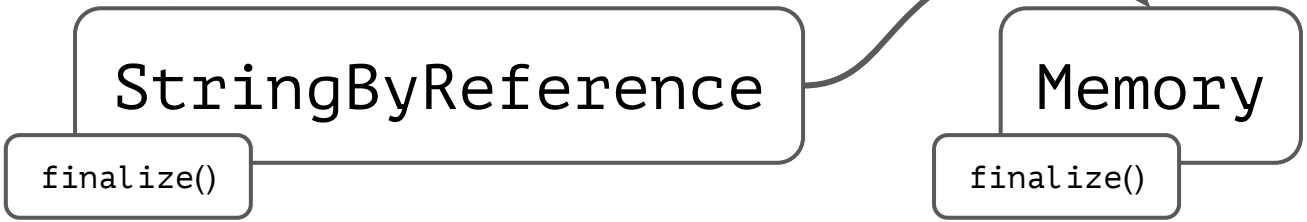



```
if (addr != null) {
    GlobalFree(addr.getPointer(0));
}
```

```
free(addr);
addr = null;
```

случился первым И в addr лежит мусор...

пользовательский код




```
if (addr != null) {
    GlobalFree(addr.getPointer(0));
}
```

```
free(addr);
addr = null;
```

случился первым И в addr лежит мусор...

получаем спорадичный развал из-за вызова GlobalFree от битой памяти!

JNA - подводные камни

- ✓ JNA мусорит Java обертками вокруг нативых сущностей (`Pointer`, `Memory`, `ByReference`)
  - Нагрузка на GC,
  - Ухудшение производительности,
  - И даже некорректное поведение!
- ✓ Недетерминированное поведение и спорадичные развалы JVM

Что же выбрать?

Что же выбрать?

	JNI	JNA
Удобство использования	✗	✓

Что же выбрать?

	JNI	JNA
Удобство использования	✗	✓
Производительность	✓	✗

Что же выбрать?

	JNI	JNA
Удобство использования	✗	✓
Производительность	✓	✗
Надежность	✗	✗

Что же выбрать?

	JNI	JNA
Удобство использования	✗	✓
Производительность	✓	✗
Надежность	✗	✗
Заготовки для библиотек	✗	✓

Что же выбрать?

	JNI	JNA
Удобство использования	✗	✓
Производительность	✓	✗
Надежность	✗	✗
Заготовки для библиотек	✗	✓
Документированность	✓	✓

Что же выбрать?

	JNI	JNA
Удобство использования	✗	✓
Производительность	✓	✗
Надежность	✗	✗
Заготовки для библиотек	✗	✓
Документированность	✓	✓
Работает с C++	✗	✗

JNR

(Java Native Runtime)

```
<dependency>
  <groupId>com.github.jnr</groupId>
  <artifactId>jnr-ffi</artifactId>
  <version>2.1.14</version>
</dependency>
```

JNR

MyNativeLib.c

```
__attribute((visibility("default"))) void sayHello(const char* name) {
     printf("Hello %s from native!\n", name);
}
```

JNR

MyNativeLib.c

```
__attribute((visibility("default"))) void sayHello(const char* name) {
     printf("Hello %s from native!\n", name);
}
```

TestJNR.java

```
public static interface MyNativeLib {
   void sayHello(String s);
}
```

```
MyNativeLib myLib = LibraryLoader.create(MyNativeLib.class).load("MyNativeLib");
myLib.sayHello("JPoint");
```

JNR

MyNativeLib.c

```
__attribute((visibility("default"))) void sayHello(const char* name) {
     printf("Hello %s from native!\n", name);
}
```

TestJNR.java

```
public static interface MyNativeLib {
   void sayHello(String s);
}
```

```
MyNativeLib myLib = LibraryLoader.create(MyNativeLib.class).load("MyNativeLib");
myLib.sayHello("JPoint");
```

```
$ java -Djava.library.path=./lib TestJNR

Hello JPoint from native!
```

JNR

MyNativeLib.c

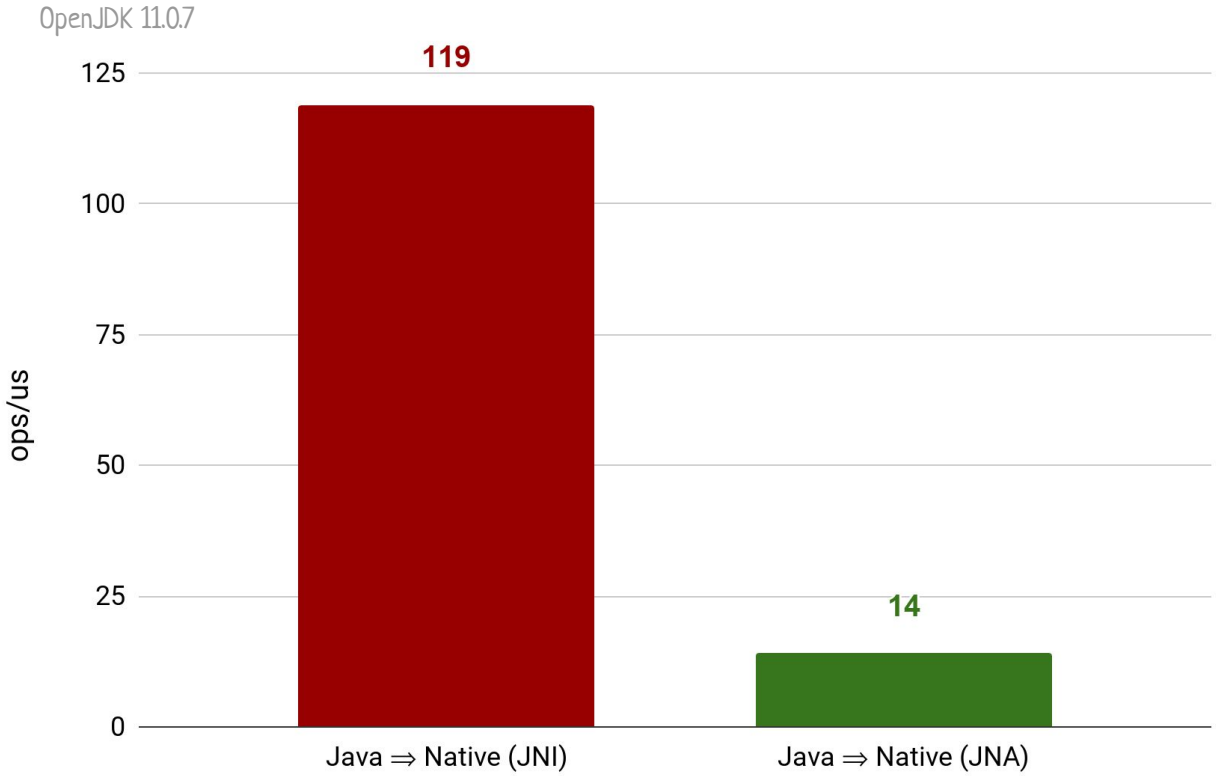
```
__attribute((visibility("default"))) void sayHello(const char* name) {
     printf("Hello %s from native!\n", name);
}
```

TestJNR.java

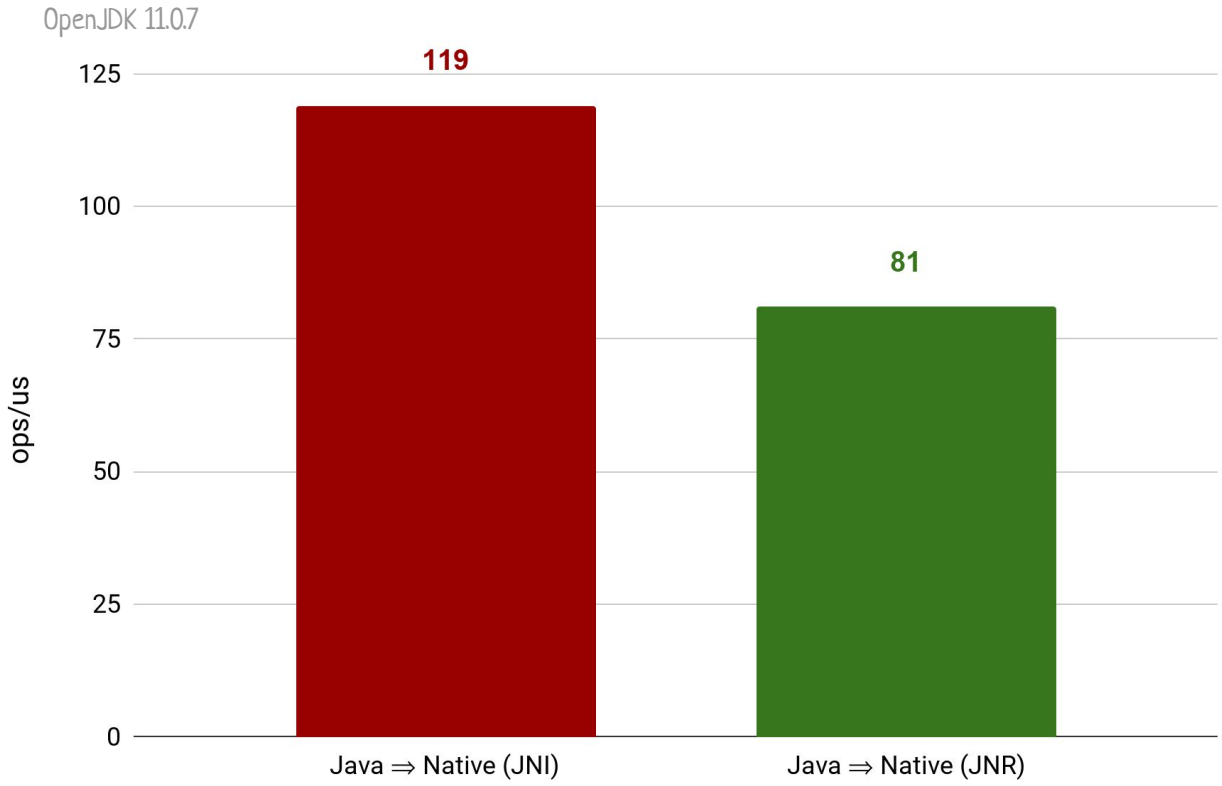
```
public static interface MyNativeLib {
   void sayHello(String s);
}
```

```
MyNativeLib myLib = LibraryLoader.cr
myLib.sayHello("JPoint");
```

JNR - производительность



JNR - производительность

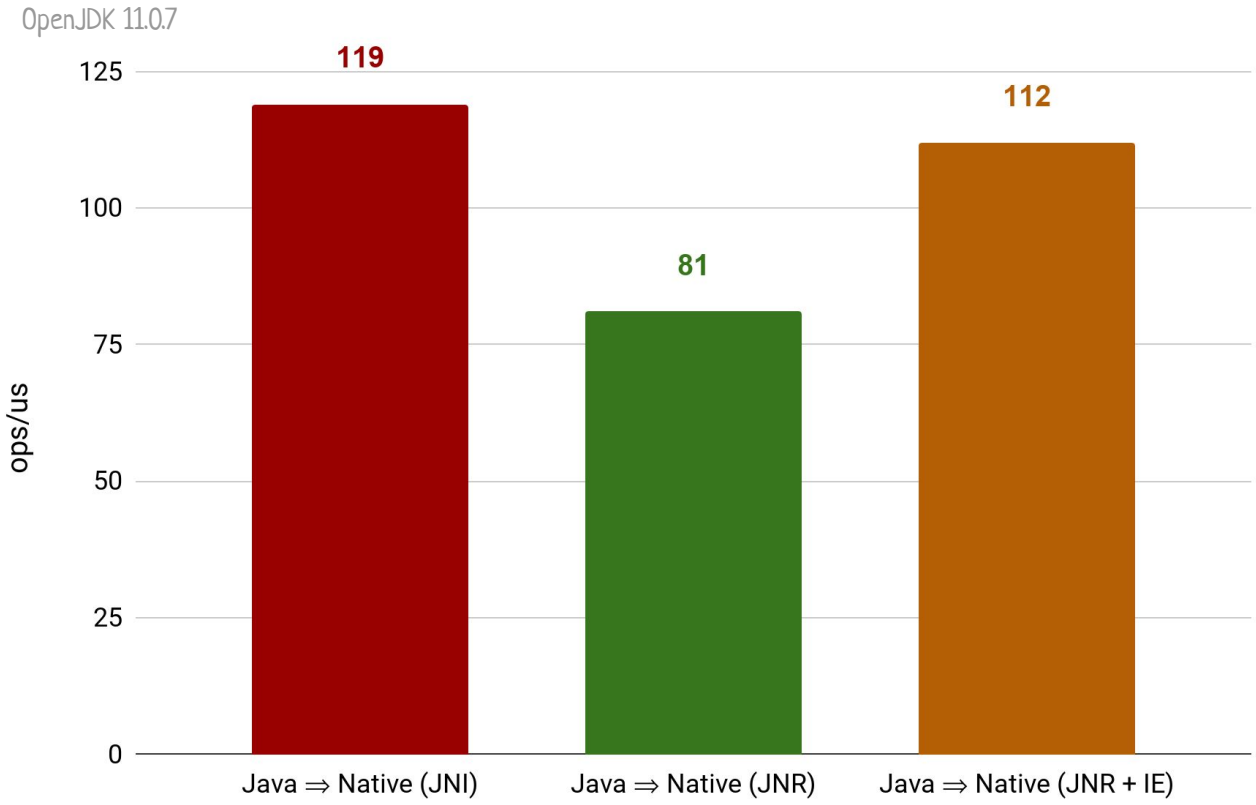


JNR - производительность

TestJNR.java

```
public static interface MyNativeLib {
   @IgnoreError
    void sayHello(String s);
}
```

JNR - производительность



Как позвать натив?

Где взять нативы?

Как работать с Java из натива?

Как себя должен вести GC?

Производительность JNR

- ✓ Все базируется на JNI, поэтому быстрее быть точно не может

Производительность JNR

- ✓ Все базируется на JNI, поэтому быстрее быть точно не может
- ✓ На стороне Java - сгенерированный байткод, на стороне натива сгенерированный асм!

```
myLib.sayHello("JPoint");
⇓
bytecode wrapper (inlined)
⇓
asm wrapper (super lightweight)
⇓
void sayHello(const char* name)
```

JNR:

- ✓ используется в JRuby и активно развивается
- ✓ есть готовая обвязка для posix, unisocket, signal...

JNR:

- ✓ используется в JRuby и активно развивается
- ✓ есть готовая обвязка для posix, unisocket, signal...

Подводные камни:

- ✓ асм завертки генерируются не для всех платформ. На Windows их нет вообще
- ✓ документации почти нет

Что же выбрать?

	JNI	JNA
Удобство использования	✗	✓
Производительность	✓	✗
Надежность	✗	✗
Заготовки для библиотек	✗	✓
Документированность	✓	✓
Работает с C++	✗	✗

Что же выбрать?

	JNI	JNA	JNR
Удобство использования	✗	✓	✓
Производительность	✓	✗	✓
Надежность	✗	✗	?
Заготовки для библиотек	✗	✓	OK
Документированность	✓	✓	✗
Работает с C++	✗	✗	✗

JavaCPP

```
<dependency>
  <groupId>org.bytedeco</groupId>
  <artifactId>javacpp</artifactId>
  <version>1.5.3</version>
</dependency>
```

JavaCPP

MyCPPLib.cpp

```
#include <iostream>

inline void sayHelloFromCPP(std::string name) {

   std::cout << "Hello " << name << " from cpp code!" << std::endl;

}
```

JavaCPP

MyCPPLib.cpp

```
#include <iostream>

inline void sayHelloFromCPP(std::string name) {

   std::cout << "Hello " << name << " from cpp code!" << std::endl;

}
```

JavaCppTest.java

```
@Platform(include = "MyCPPLib.cpp")
public static class MyCPPLib {

   static { Loader.load(); }

   static native void sayHelloFromCPP(@StdString String name);
}

MyCPPLib.sayHelloFromCPP("JPoint");
```

JavaCPP

MyCPPLib.cpp

```cpp
#include <iostream>

inline void sayHelloFromCPP(std::string name) {

   std::cout << "Hello " << name << " from cpp code!" << std::endl;

}
```

JavaCppTest.java

```java
@Platform(include = "MyCPPLib.cpp")
public static class MyCPPLib {

   static { Loader.load(); }

   static native void sayHelloFrom

}

MyCPPLib.sayHelloFromCPP("JPoint");
```

```
$ java -Djava.library.path=./lib JavaCppTest

Hello JPoint from cpp code!
```

JavaCPP

```
#include <iostream>
inline void sayHelloFromCPP(std::string name) {
   std::cout << "Hello " << name
             << " from cpp code!" << std::endl;
}
```

```
@Platform(include = "MyCPPLib.cpp")
public static class MyCPPLib {

   static { Loader.load(); }

   static native void sayHelloFromCPP
                        (@StdString String name);
}

MyCPPLib.sayHelloFromCPP("JPoint");
```

```
JNIEXPORT void JNICALL Java_MyCPPLib_sayHelloFromCPP
 (JNIEnv *, jclass, jstring);
```

сгенерированные JNI адаптеры

компилируются в отдельную нативную библиотеку при сборке

JavaCPP

- ✓ Позволяет из Java работать с C++ сущностями:
  - виртуальные функции и перегруженные операторы
  - string, vector, map
  - указатели, в т.ч. на функции
  - деструкторы и конструкторы
  - smart-pointers
  - шаблоны!

JavaCPP

- ✓ Позволяет из Java работать с C++ сущностями

- ✓ Гибкая генерация Java оберток по C++ библиотеке
- ✓ Огромное количество заготовленных оберток для популярных библиотек

  github.com/bytedeco/javacpp-presets

JavaCPP

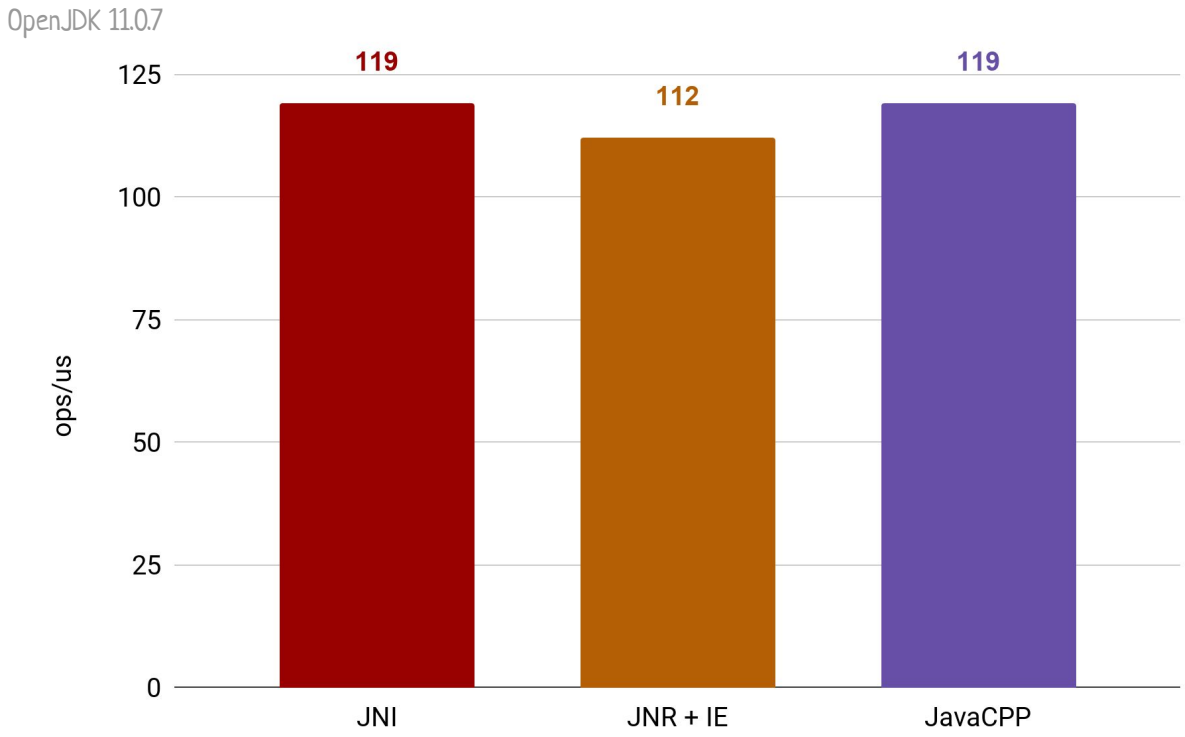
- ✓ Позволяет из Java работать с C++ сущностями
- ✓ Гибкая генерация Java оберток по C++ библиотеке
- ✓ Огромное количество заготовленных оберток для популярных библиотек
  github.com/bytedeco/javacpp-presets
- ✓ Сборка через Maven или Gradle, включая C/C++ часть!

А что там с производительностью?

JavaCPP - подводные камни

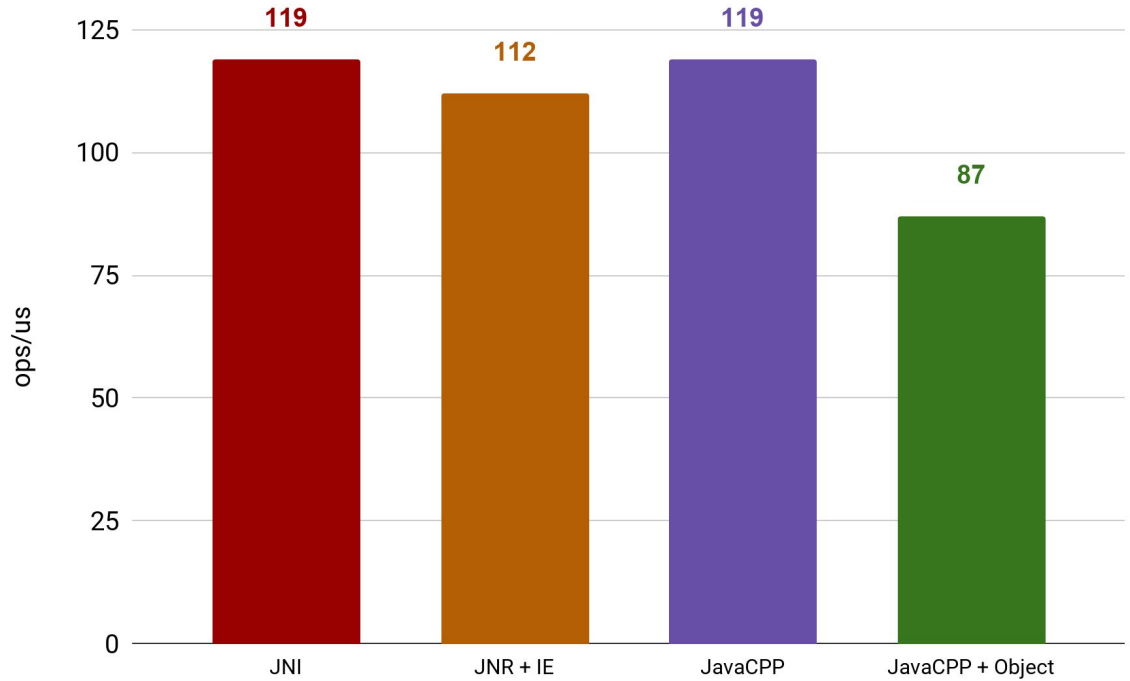
- ✓ Все базируется на JNI, поэтому быстрее быть точно не может

JavaCPP - подводные камни



JavaCPP - подводные камни

OpenJDK 11.0.7



но стоит заиспользовать C++ объекты в Java коде..

JavaCPP - подводные камни

- ✓ Использование Java версий C++ объектов просаживает производительность

JavaCPP - подводные камни

- ✓ Использование Java версий C++ объектов просаживает производительность

- ✓ Вместо деструкторов и smart pointers - PhantomReference + ReferenceQueue

⇓

недетерменированность исполнения и веселые баги! (привет, JNA)

Что же выбрать?

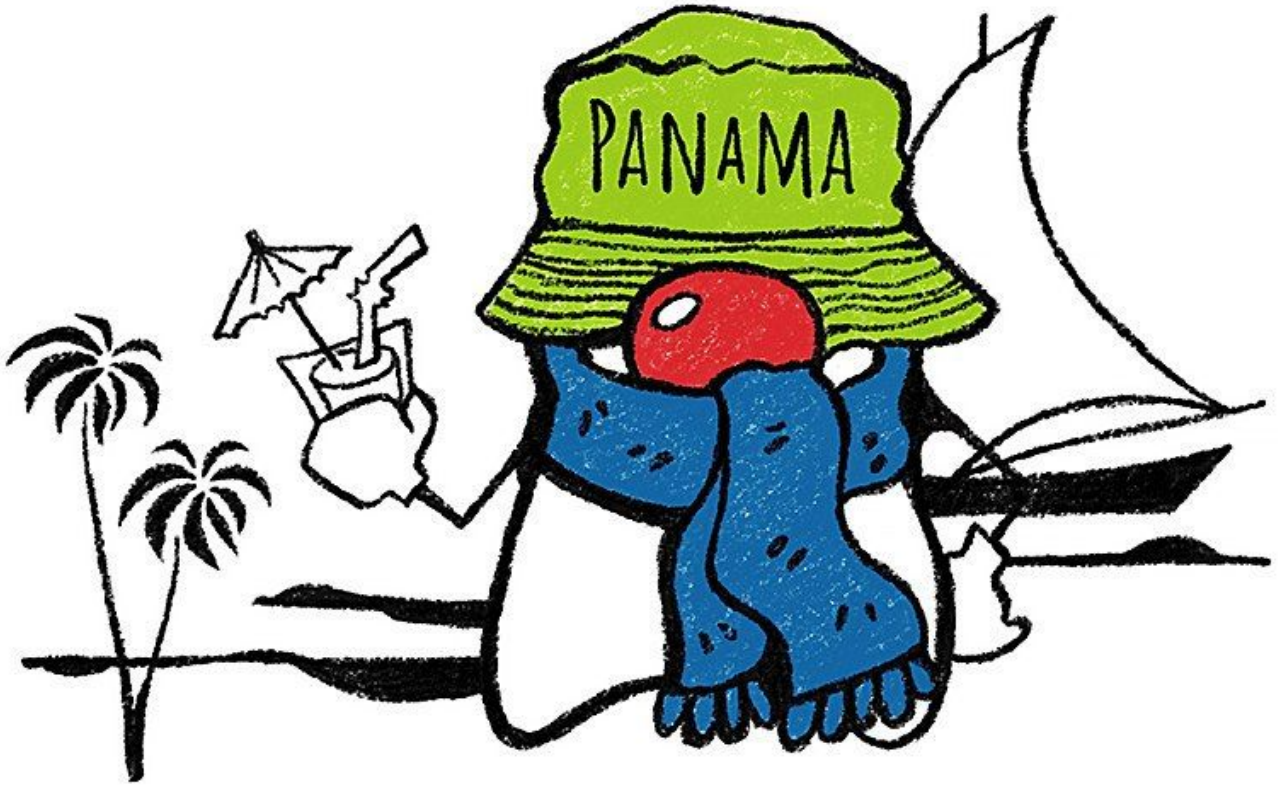
	JNI	JNA	JNR	JavaCPP
Удобство использования	✗	✓	✓	
Производительность	✓	✗	✓	
Надежность	✗	✗	?	
Заготовки для библиотек	✗	✓	OK	
Документированность	✓	✓	✗	
Работает с C++	✗	✗	✗	

Takeaways про библиотеки

	JNI	JNA	JNR	JavaCPP
Удобство использования	✗	✓	✓	✓
Производительность	✓	✗	✓	OK
Надежность	✗	✗	?	OK
Заготовки для библиотек	✗	✓	OK	👍
Документированность	✓	✓	✗	✓
Работает с C++	✗	✗	✗	✓

Что день готовит
нам грядущий?

Project Panama



Project Panama

- ✓ Что: мега-проект в JDK (since 2014)
- ✓ Зачем: легкость использования C/C++ библиотек и нативного кода из Java
- ✓ Как: вместо написания кода на C/C++ расширить возможности Java

Project Panama

- ✓ Memory Access API (JEP 370, JEP 383)
  - Новый API для работы с нативной памятью
  - Потенциальная замена ByteBuffer
  - Incubator-модуль в Java 14

Project Panama

- ✓ Memory Access API (JEP 370, JEP 383)

- ✓ Новый Foreign Function Interface (JEP 191)
  - Native Method Handles
  - MemorySegment from Memory Access API
  - jextract для генерации интерфейсов

Project Panama

panamatest.c

```c
void test(void) {
    printf("Hello from Panama!\n");
}

void testUpcall(void (*upcall)(void)) {
    upcall();
}
```

panamatest.h

```c
void test(void);
void testUpcall(void (*upcall)(void));
```

Project Panama

panamatest.c

```c
void test(void) {
    printf("Hello from Panama!\n");
}

void testUpcall(void (*upcall)(void)) {
    upcall();
}
```

panamatest.h

```c
void test(void);
void testUpcall(void (*upcall)(void));
```

jextract

```
C_bool.class
Cchar.class
Cdouble.class
Cfloat.class
Cint.class
Clong.class
Clong_double.class
Clong_long.class
Cpointer.class
Cshort.class
Cstring.class
panamatest_h.class
panamatest_h$testUpcall$upcall.clas
RuntimeHelper.class
RuntimeHelper$VarargsInvoker.class
panamatest_h$constants.class
```

Project Panama

panamatest_h.java

```
package org.sample;

public final class panamatest_h {

    public static MethodHandle test$MH() {
        return panamatest_h$constants.test$MH();
    }
    public static void test () {
        try {
            panamatest_h$constants.test$MH().invokeExact();
        } catch (Throwable ex) {
            throw new AssertionError(ex);
        }
    }
    ...
}
```

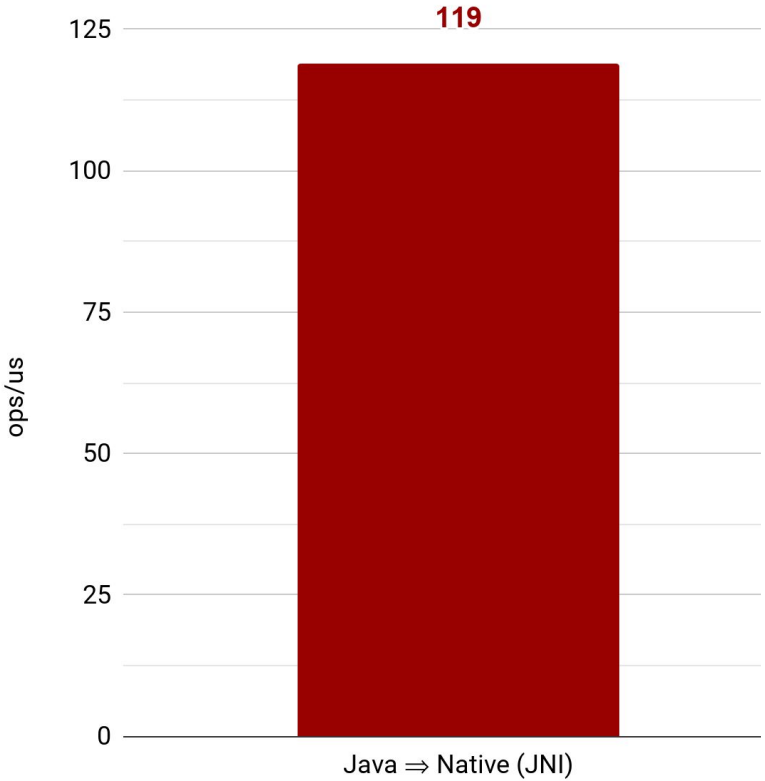
А что там с производительностью?

Project Panama

- ✓ Улучшения производительности Java ⇒ Native и Native ⇒ Java вызовов через MethodHandles

WORK IN PROGRESS!

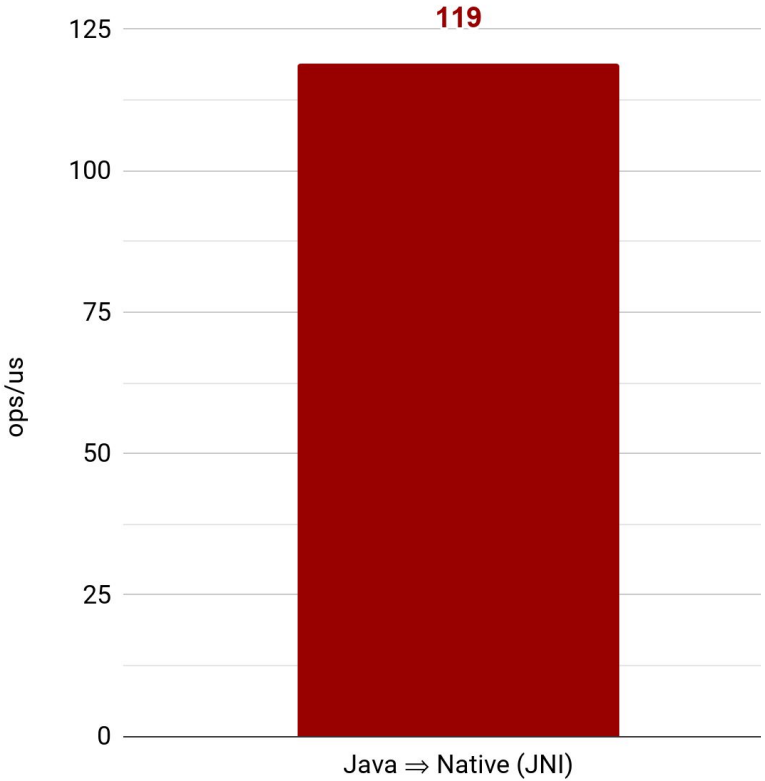
Project Panama



Java ⇒ Native (JNI) – вызов нативного метода (без параметров) из Java

OpenJDK 16-internal+0-adhoc.user.panama-foreign-CallIntrinsicsFinal2

Project Panama

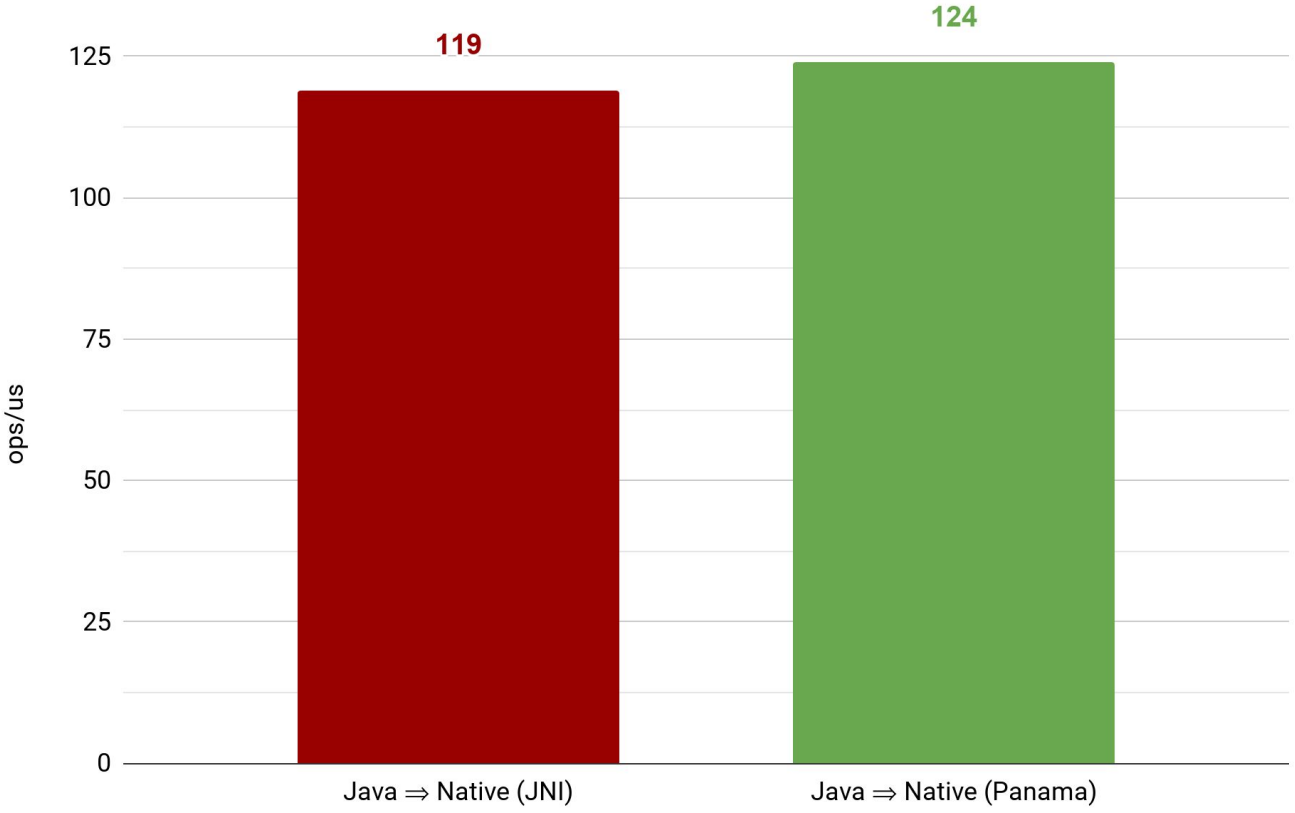


Java ⇒ Native (JNI) – вызов нативного метода (без параметров) из Java

Java ⇒ Native (Panama) – вызов нативного метода (без параметров) из Java через MethodHandle

OpenJDK 16-internal+0-adhoc.user.panama-foreign-CallIntrinsicsFinal2

Project Panama



OpenJDK 16-internal+0-adhoc.user.panama-foreign-CallIntrinsicsFinal2

Java ⇒ Native (JNI) – вызов нативного метода (без параметров) из Java

Java ⇒ Native (Panama) – вызов нативного метода (без параметров) из Java через MethodHandle

+4%

Project Panama

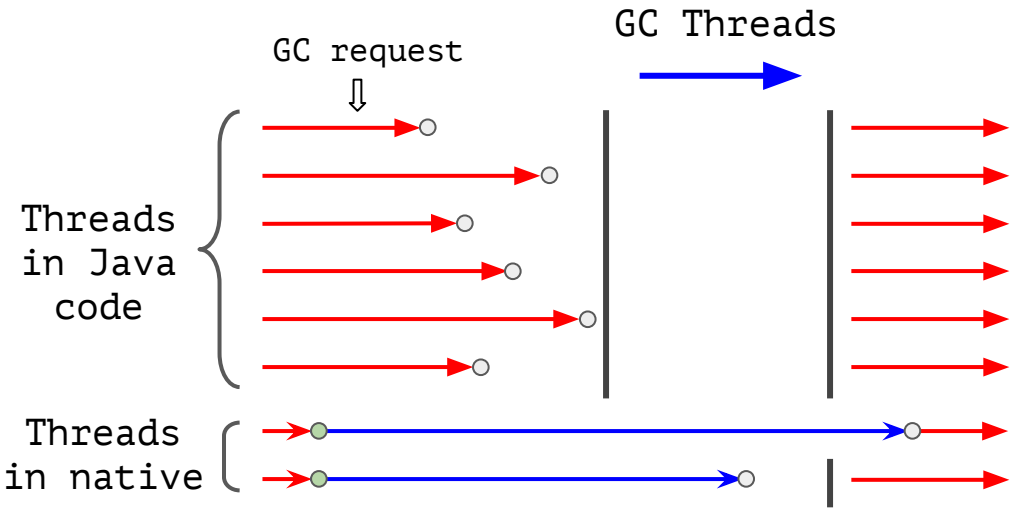
- ✓ Улучшения производительности Java ⇒ Native и Native ⇒ Java вызовов через MethodHandles

  WORK IN PROGRESS!

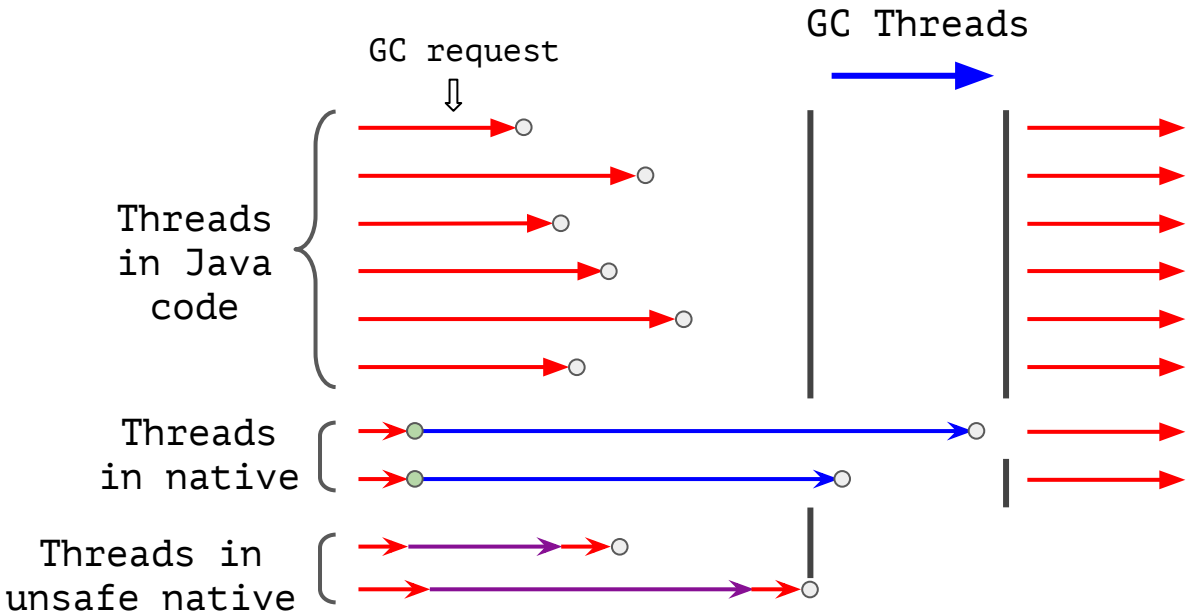
- ✓ Unsafe нативные методы, без синхронизации с GC

  WORK IN PROGRESS!

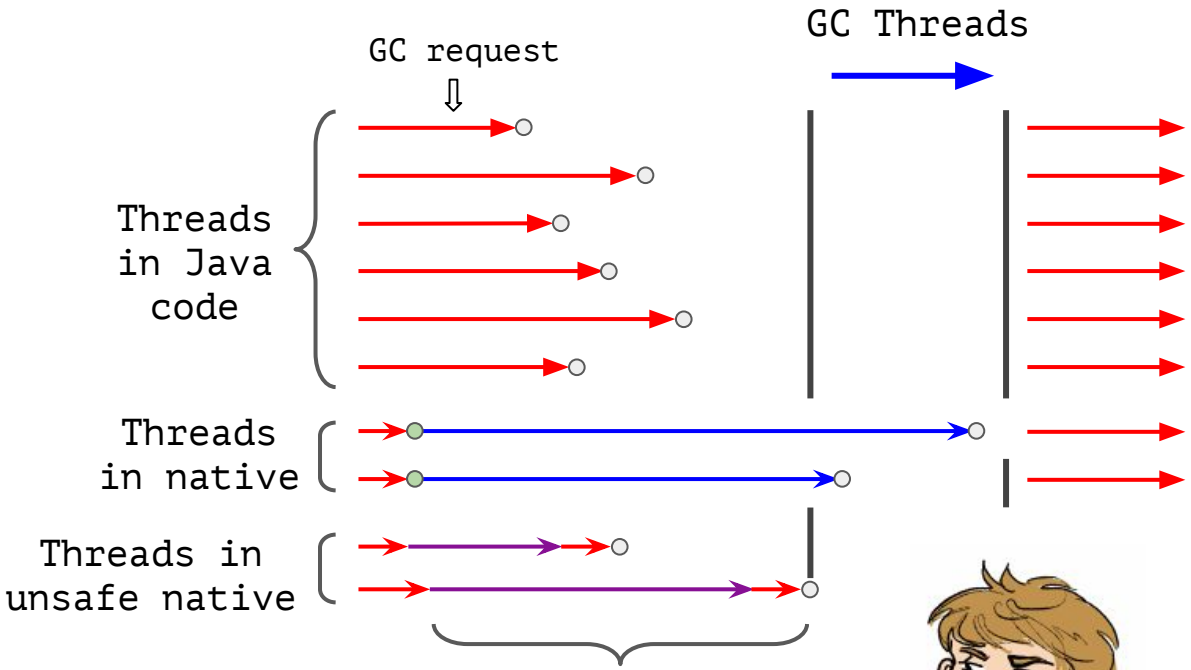
Project Panama (подводные камни)



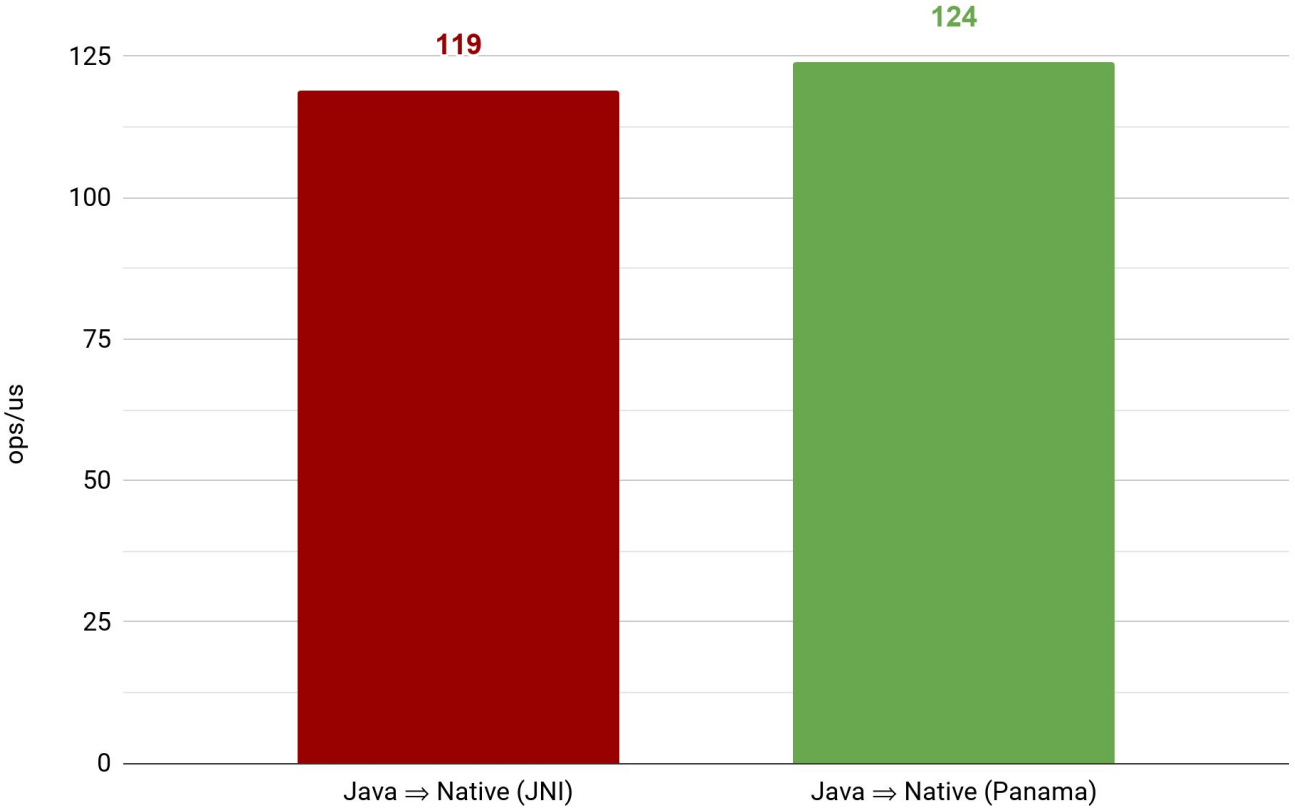
Project Panama (подводные камни)



Project Panama (подводные камни)



Project Panama



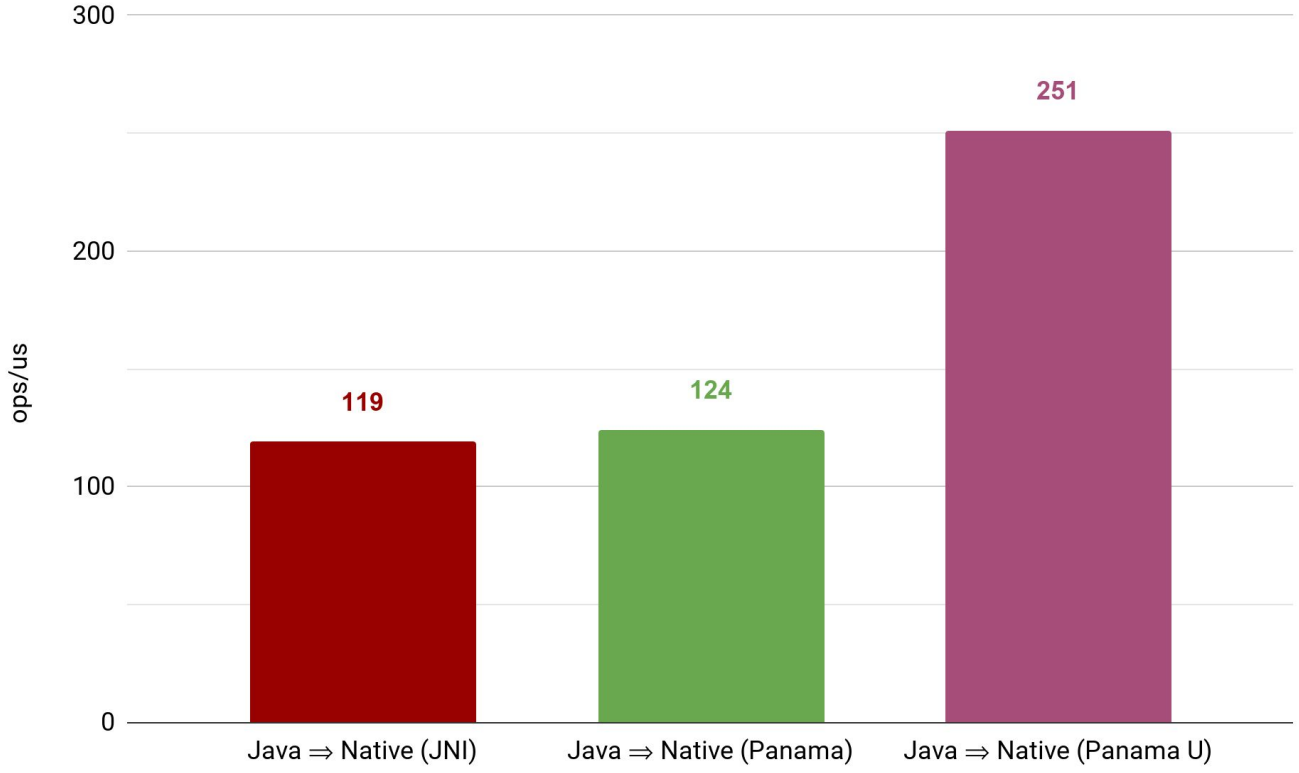
OpenJDK 16-internal+0-adhoc.user.panama-foreign-CallIntrinsicsFinal2

Java ⇒ Native (JNI) – вызов нативного метода (без параметров) из Java

Java ⇒ Native (Panama) – вызов нативного метода (без параметров) из Java через MethodHandle

+4%

Project Panama



OpenJDK 16-internal+0-adhoc.user.panama-foreign-CallIntrinsicsFinal2

Java ⇒ Native (Panama) – вызов нативного метода (без параметров) из Java через MethodHandle без синхронизации с GC

Project Panama

- ✓ Unsafe natives - опасный механизм, но дает уникальную производительность
- ✓ По умолчанию вызовы безопасные
- ✓ Очень экспериментальная фича!
  (замеры на версии от 25.06.20)

Project Panama

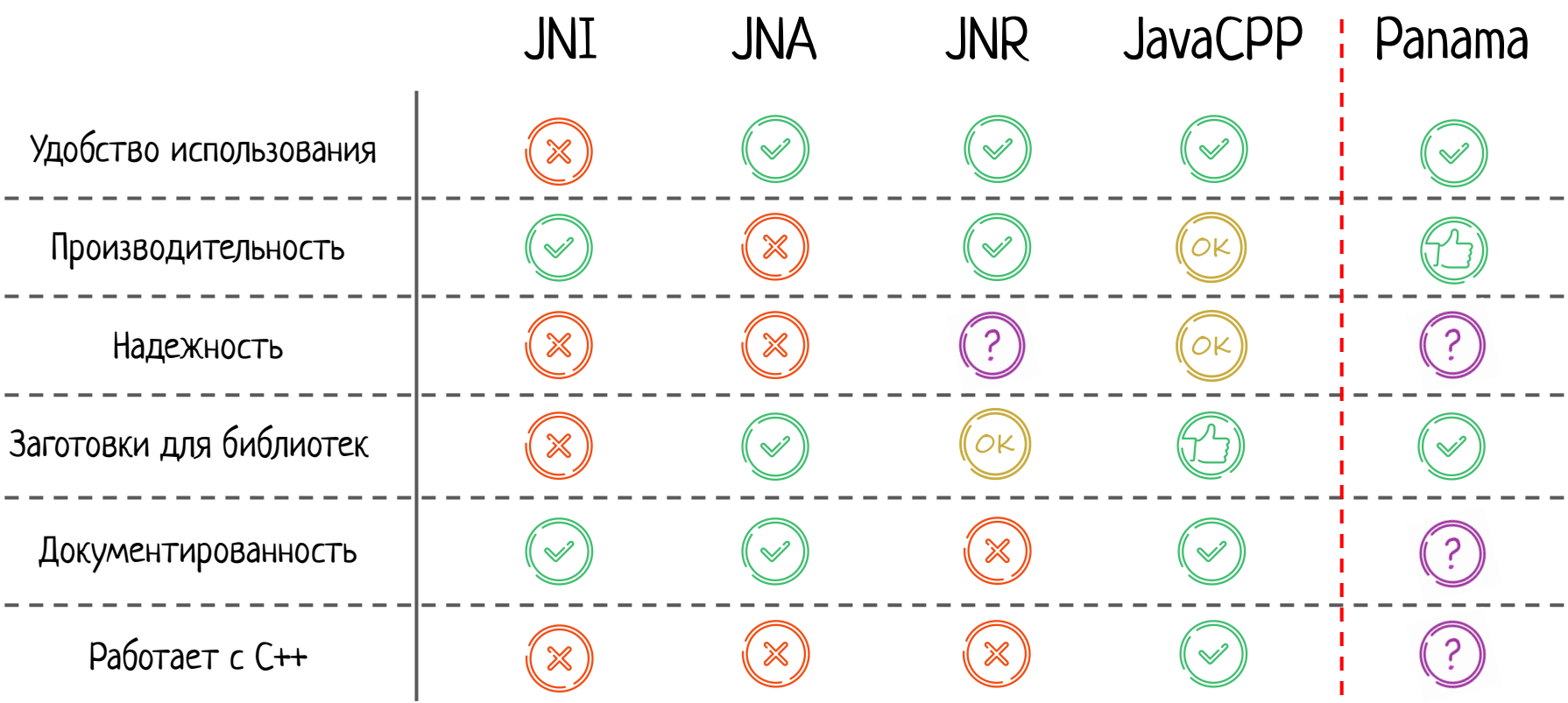
youtu.be/4vHMmLqF09Y

youtu.be/0y6_RDga-fk

Что же выбрать?

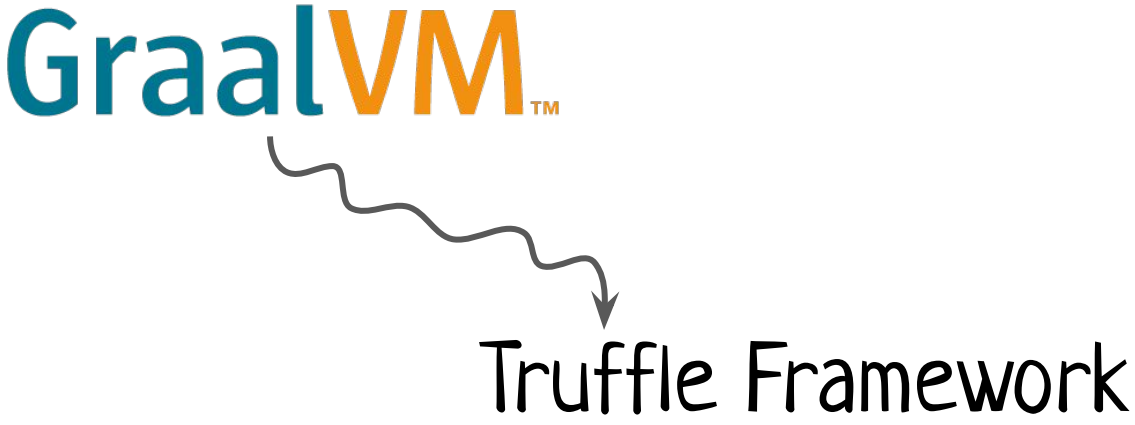
	JNI	JNA	JNR	JavaCPP
Удобство использования	✗	✓	✓	✓
Производительность	✓	✗	✓	OK
Надежность	✗	✗	?	OK
Заготовки для библиотек	✗	✓	OK	👍
Документированность	✓	✓	✗	✓
Работает с C++	✗	✗	✗	✓

Что же выбрать?



А может полетим в Мордор на орлах?


GraalVM™


GraalVM™
Truffle Framework

Truffle Framework

GraalVM LLVM Runtime

GraalVM LLVM Runtime

- ✓ C/C++ код компилируется в LLVM bitcode

GraalVM LLVM Runtime

✓ C/C++ код компилируется в LLVM bitcode

native.c

```
void goNative() {
    printf("Hello from LLVM!\n");
}
```

$LLVM_TOOLCHAIN/clang -c native.c -o native.bc

native.bc

GraalVM LLVM Runtime

- ✓ C/C++ код компилируется в LLVM bitcode
- ✓ LLVM bitcode интерпретируется с агрессивной специализацией

TestPolyglot.java

```
import org.graalvm.polyglot.*;

...

Context polyglot = Context.newBuilder().allowAllAccess(true).build();

File file = new File("native.bc");


Source source = Source.newBuilder("llvm", file).build();

Value cpart = polyglot.eval(source);


cpart.getMember("goNative").executeVoid();
```

TestPolyglot.java

```
import org.graalvm.polyglot.*;

...

Context polyglot = Context.newBuilder().allowAllAccess(true).build();

File file = new File("native.bc");


Source source = Source.newBuilder("llvm", file).build();

Value cpart = polyglot.eval(source);


cpart.getMember("goNative").executeVoid();
```

```
$ $GRAAL_VM_HOME/javac TestPolyglot.java
$ $GRAAL_VM_HOME/java  TestPolyglot

Hello from LLVM!
```

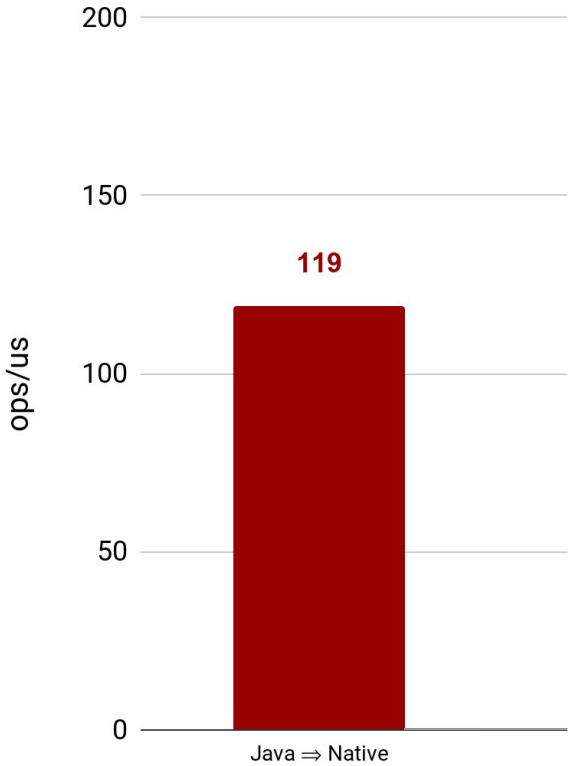
GraalVM LLVM Runtime

- ✓ C/C++ код компилируется в LLVM bitcode
- ✓ LLVM bitcode интерпретируется с агрессивной специализацией
- ✓ Единое внутреннее представление для native и Java кода
- ✓ Native код под нашим контролем!

GraalVM LLVM Runtime

А что там с производительностью?

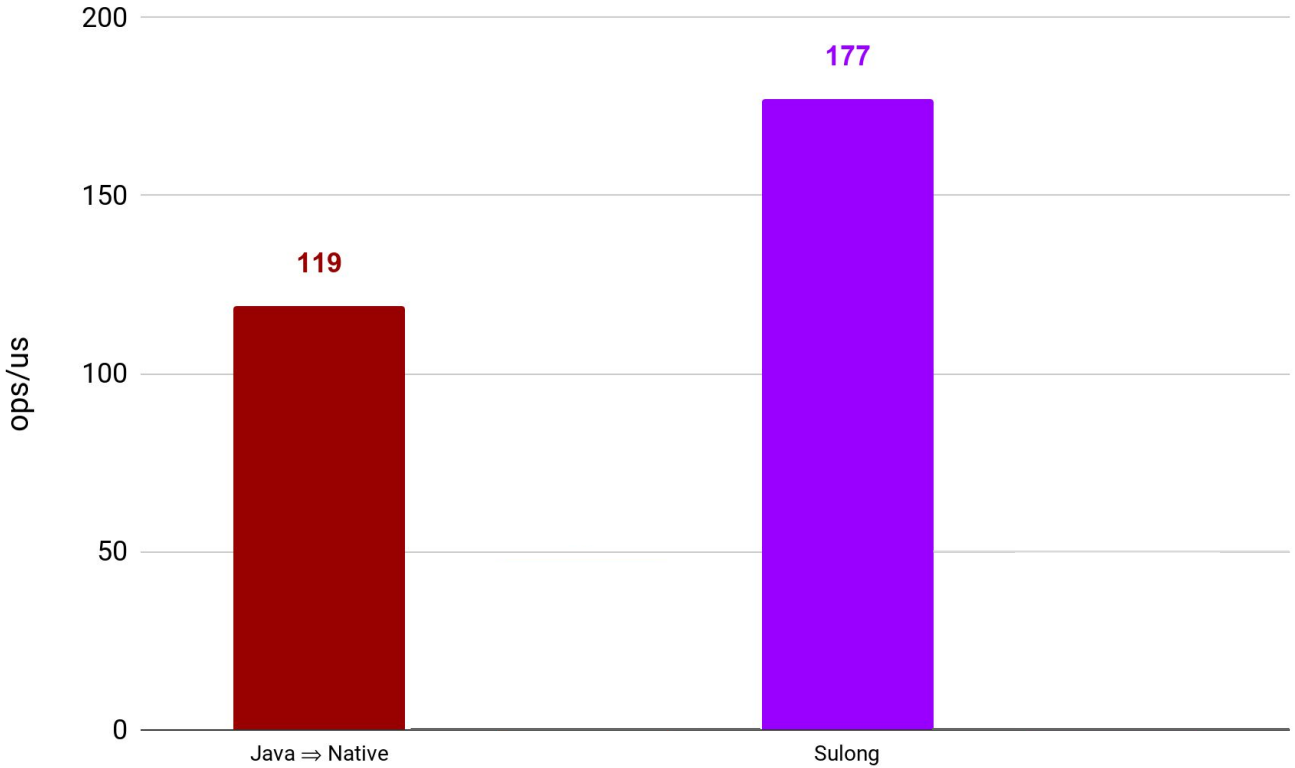
GraalVM LLVM Runtime



Java ⇒ Native – вызов пустого нативного метода (без параметров) из Java

OpenJDK GraalVM CE 20.1.0 (build 11.0.7+10-jvmci-20.1-b02)

GraalVM LLVM Runtime



Java ⇒ Native – вызов пустого нативного метода (без параметров) из Java

Sulong – вызов пустого метода на C (без параметров) через Sulong

OpenJDK GraalVM CE 20.1.0 (build 11.0.7+10-jvmci-20.1-b02)

GraalVM LLVM Runtime

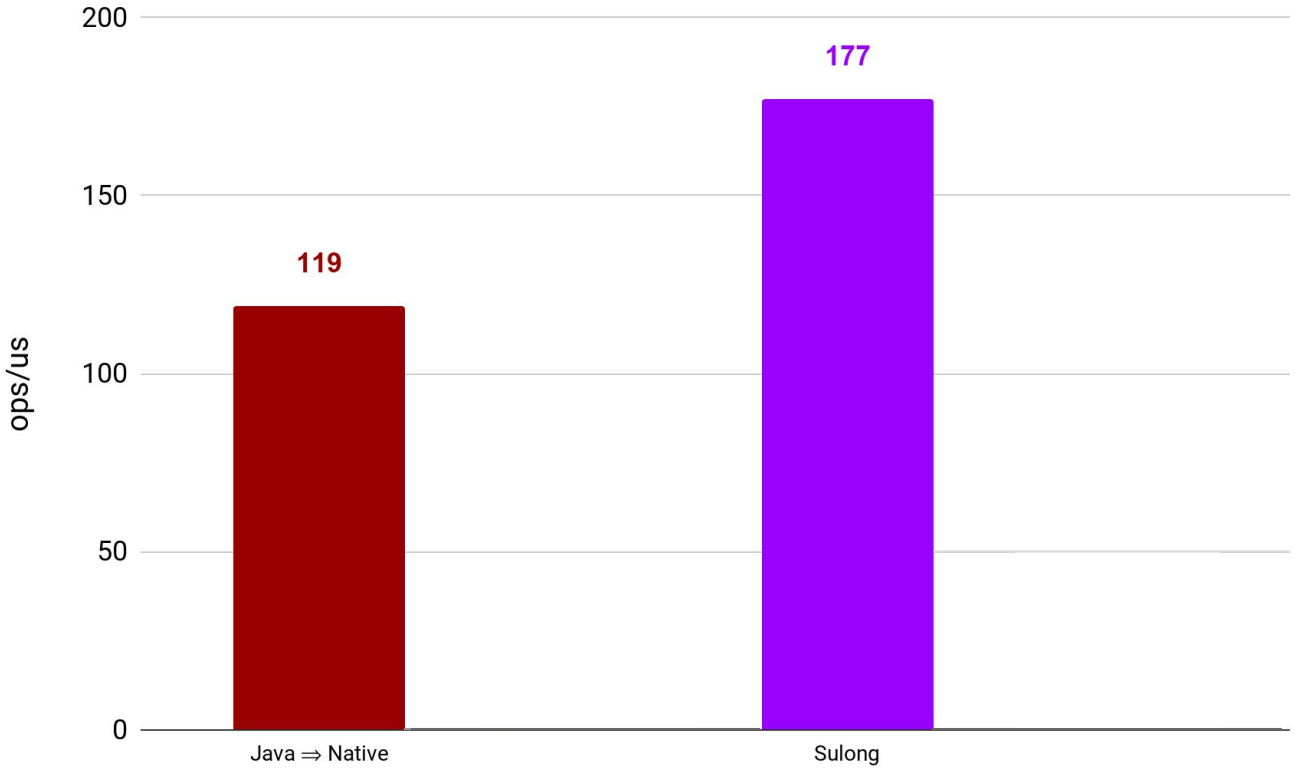
native.c

```
int nativeWithSomeWork(int k) {
    int prev = 0;
    int curr = 1;
    for (int i = 0; i < k; i++) {
        int summ = prev + curr;
        prev = curr;
        curr = summ;
    }
    return curr;
}
```

native.bc

$LLVM_TOOLCHAIN/clang -c native.c -o native.bc

GraalVM LLVM Runtime

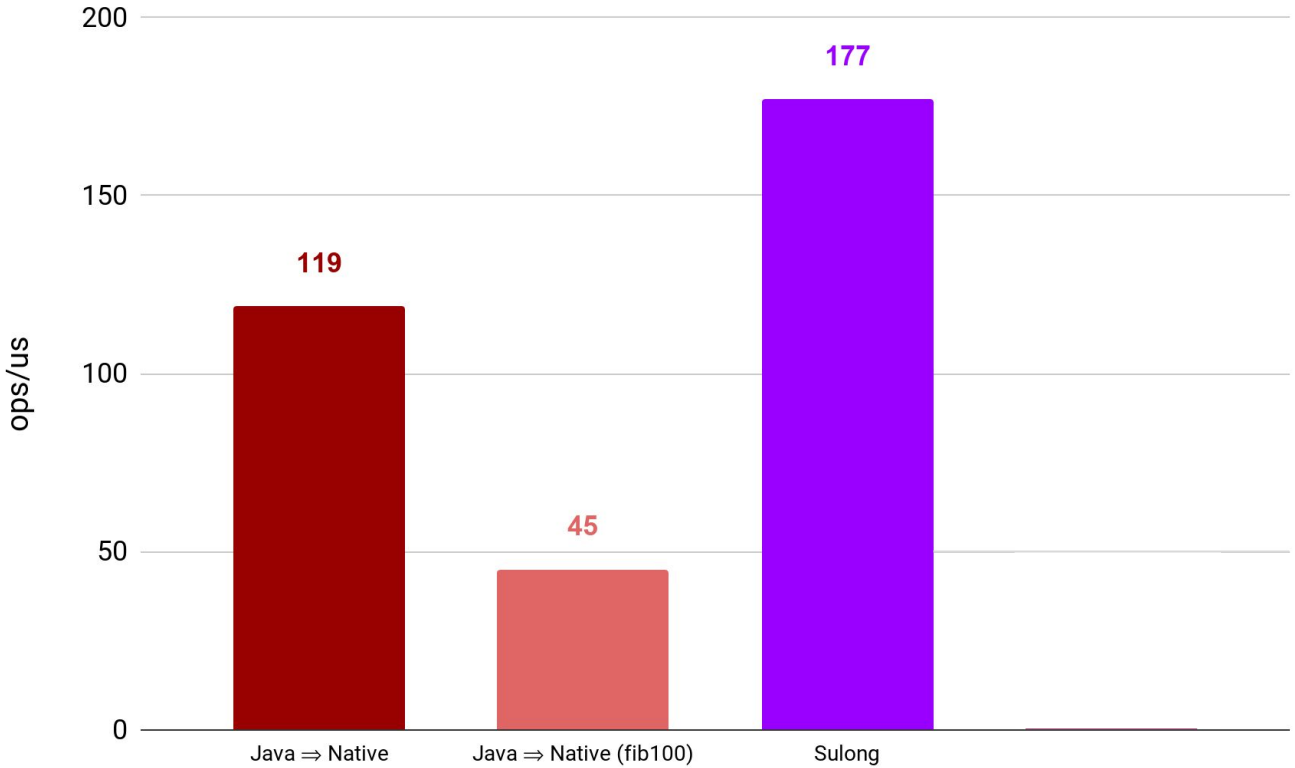


Java ⇒ Native – вызов пустого нативного метода (без параметров) из Java

Sulong – вызов пустого метода на C (без параметров) через Sulong

OpenJDK GraalVM CE 20.1.0 (build 11.0.7+10-jvmci-20.1-b02)

GraalVM LLVM Runtime



Java ⇒ Native – вызов пустого нативного метода (без параметров) из Java

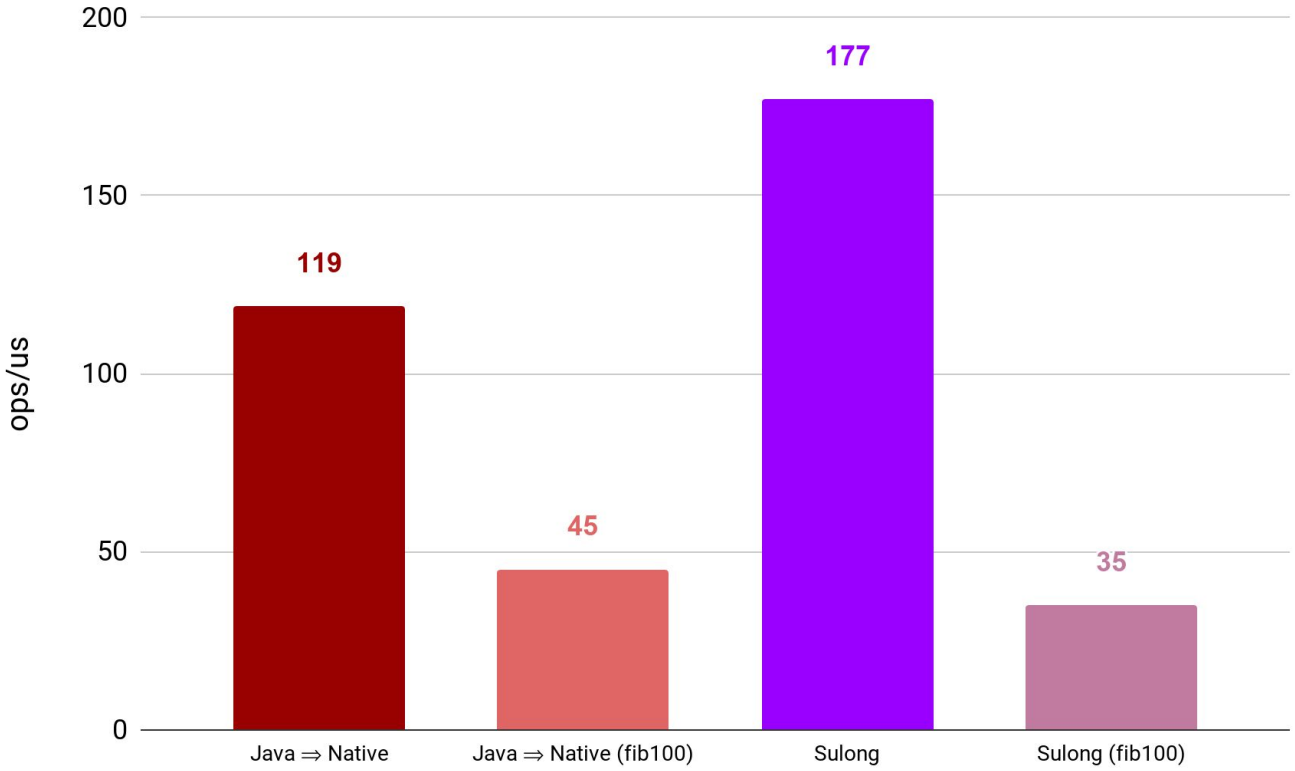
Sulong – вызов пустого метода на C (без параметров) через Sulong

(fib100) – вычисление 100-ого числа Фибоначчи в нативе

↓x2.5

OpenJDK GraalVM CE 20.1.0 (build 11.0.7+10-jvmci-20.1-b02)

GraalVM LLVM Runtime



Java ⇒ Native – вызов пустого нативного метода (без параметров) из Java

Sulong – вызов пустого метода на C (без параметров) через Sulong

(fib100) – вычисление 100-ого числа Фибоначчи в нативе

↓x2.5 vs ↓x5

OpenJDK GraalVM CE 20.1.0 (build 11.0.7+10-jvmci-20.1-b02)

GraalVM LLVM Runtime

- ✓ Чем больше нативного кода, тем хуже производительность
- ✓ Sulong - work in progress
- ✓ Sulong нужно очень долго “прогревать”

Что же выбрать?

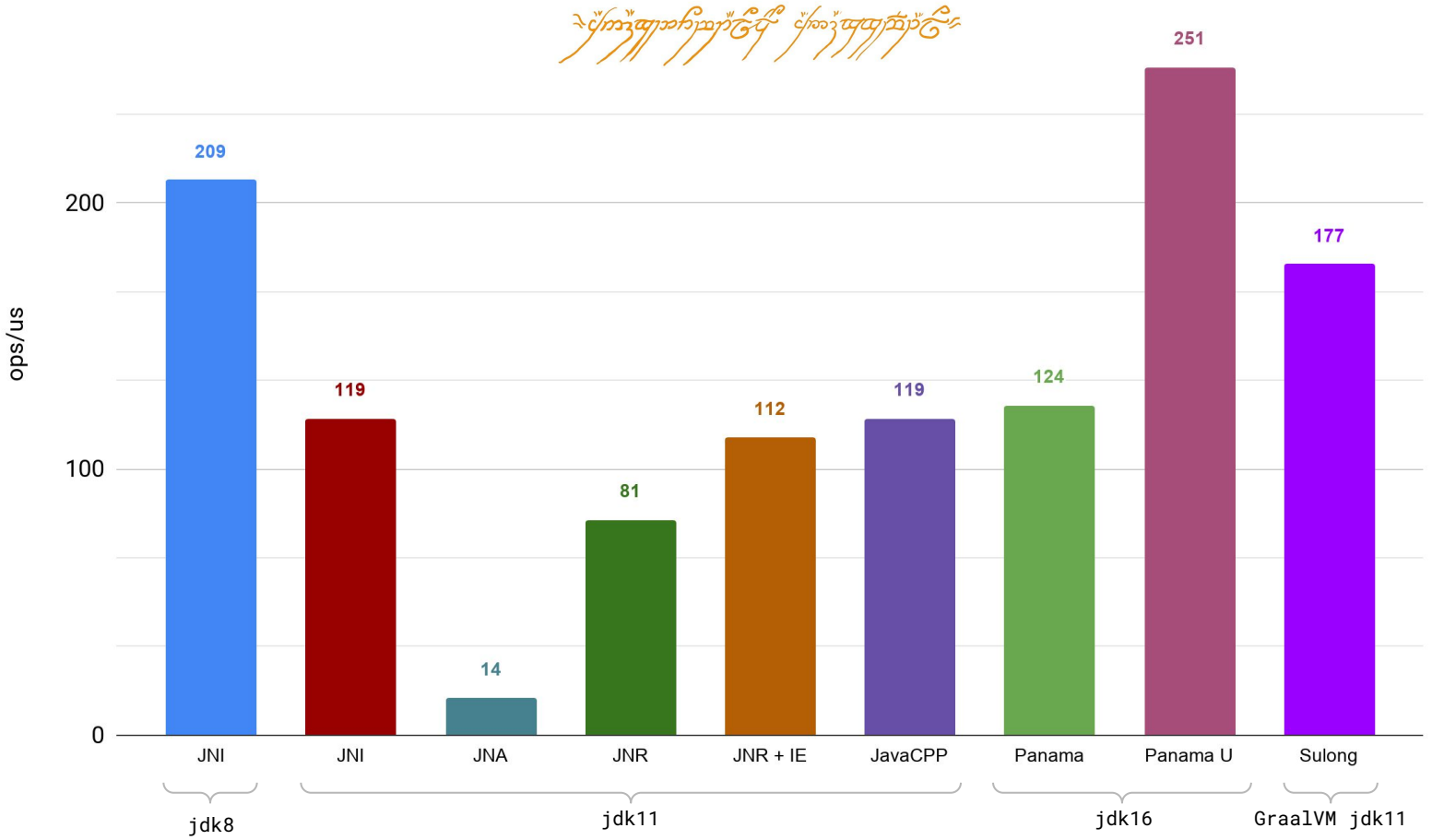
	JNI	JNA	JNR	JavaCPP	Panama
Удобство использования	✗	✓	✓	✓	✓
Производительность	✓	✗	✓	OK	👍
Надежность	✗	✗	?	OK	?
Заготовки для библиотек	✗	✓	OK	👍	✓
Документированность	✓	✓	✗	✓	?
Работает с C++	✗	✗	✗	✓	?

Финальное сравнение

	JNI	JNA	JNR	JavaCPP	Panama	Sulong
Удобство использования	✗	✓	✓	✓	✓	✓
Производительность	✓	✗	✓	OK	👍	OK
Надежность	✗	✗	?	OK	?	✓
Заготовки для библиотек	✗	✓	OK	👍	✓	?
Документированность	✓	✓	✗	✓	?	OK
Работает с C++	✗	✗	✗	✓	?	✓

Заключение

- ✓ Старайтесь не писать нативный код (там горные тролли)
- ✓ Знайте свой ~~путь в Мордор~~ фреймворк для вызова нативного кода
- ✓ Native ⇔ managed переходы в Java - все еще открытый вопрос!



Почему jdk8 такая быстрая: bugs.openjdk.java.net/browse/JDK-8187809

Q & A


ivan.ugliansky@gmail.com


@dbg_nsk

@jugnsk

бенчмарки здесь: github.com/ugliansky/jpoint2020