Going Native

Foreign Functions on the JVM

Me

- Charles Oliver Nutter
- Red Hat, JRuby
- headius@headius.com
- @headius

JRuby

JRuby Challenges

- Dynamic language
- Multi-encoding strings (see my talk from yesterday)
- Startup time
- <u>POSIX and native library integration</u>

Native Calls in JRuby

- OS-level: filesystem, spawn, stat, tty/pty/fcntl
- Libraries: graphics, crypto, nosql, mq, db
- C ext replacement/transition
- Languages and runtimes: llvm, clang, V8

A Simple Case

- Get the current process PID
- Only hacky, unreliable pure-Java ways
- getpid(2) is a trivial function, let’s use it

Name

getpid, getppid - get process identification

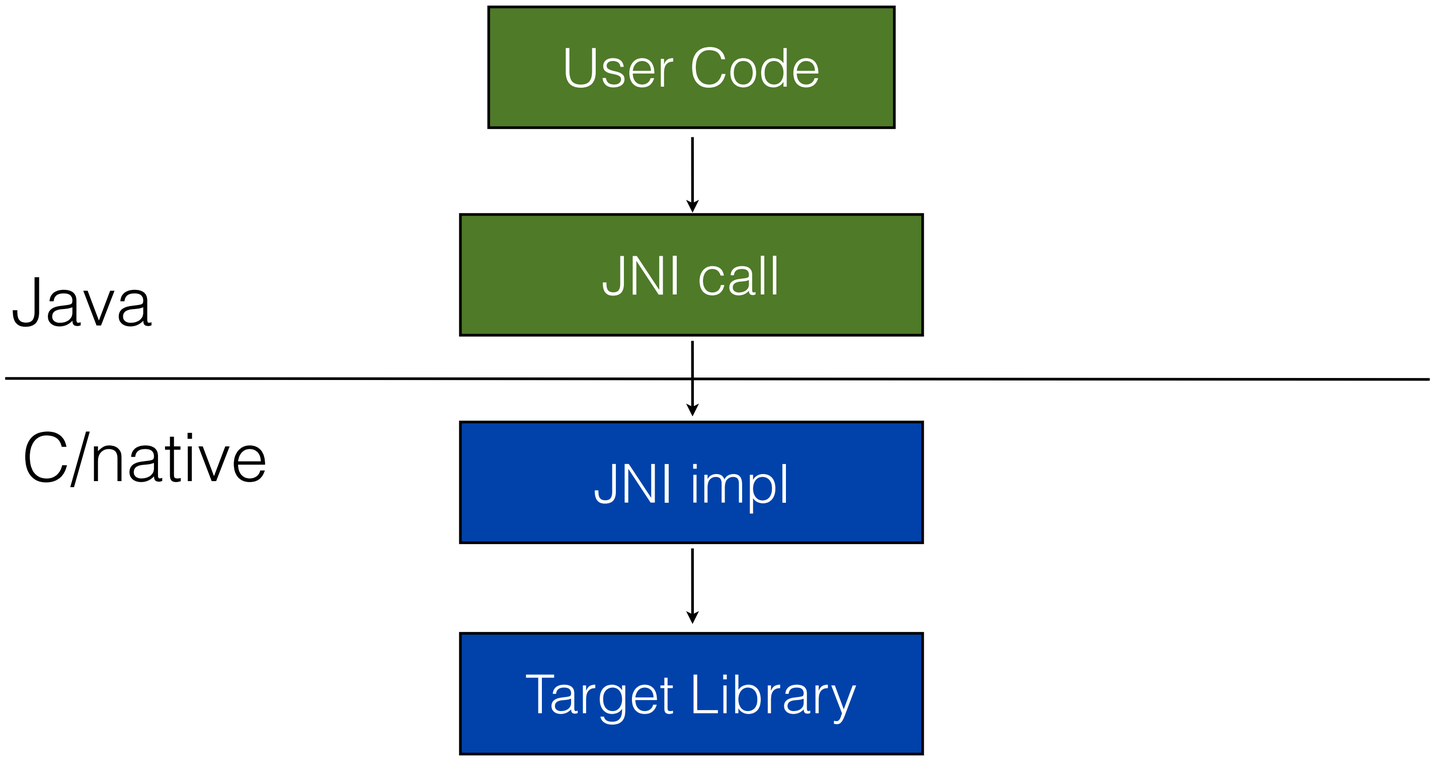
Synopsis

```
#include <sys/types.h>
#include <unistd.h>
pid_t getpid(void);
pid_t getppid(void);
```

Java Native Interface

JNI

- First standardized in Java 1.1
- Minimal changes since then
- "Designed to be hard to use"
  - Mission accomplished!


User Code
JNI call
Java
C/native
JNI impl
Target Library

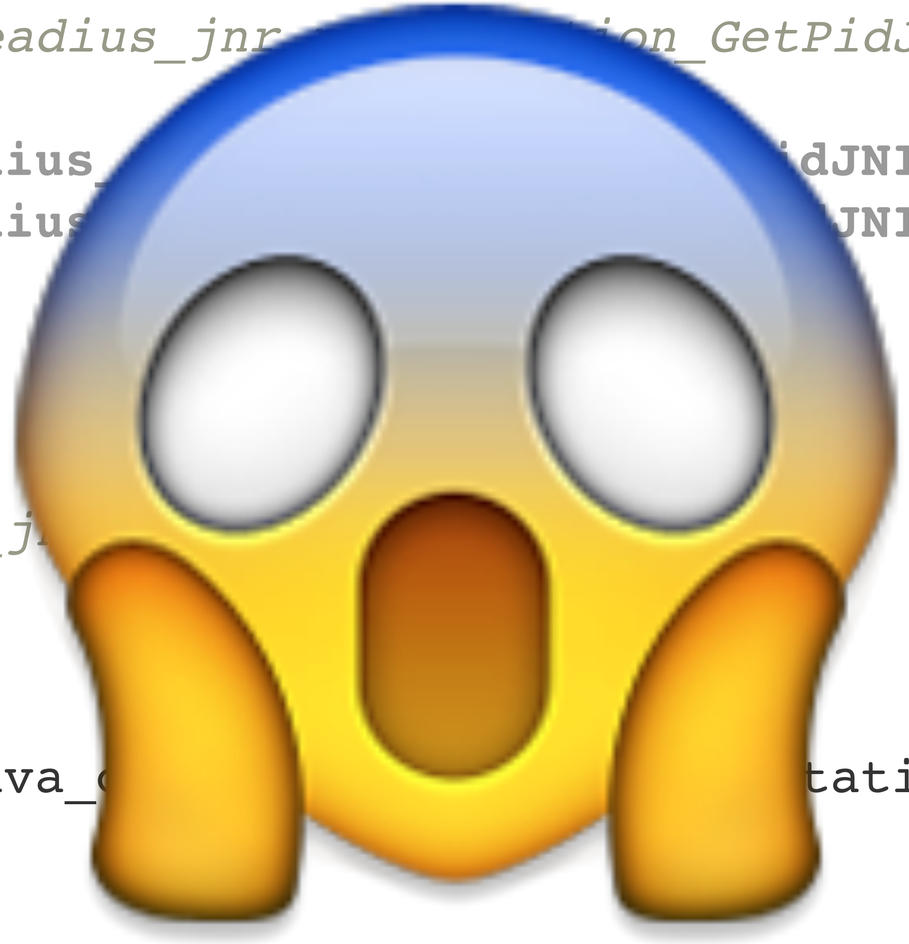
JNI

```
public class GetPidJNI {
    public static native long getpid();

    public static void main( String[] args ) {
        System.out.println(getpid());
    }

    static {
        System.load(System.getProperty("user.dir") + "/getpidjni.dylib");
    }
}
```

```
$ javah GetPidJNI.java
```

```c
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_headius_jnr_presentation_GetPidJNI */

#ifndef _Included_com_headius_jnr_presentation_GetPidJNI
#define _Included_com_headius_jnr_presentation_GetPidJNI
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     com_headius_jnr_presentation_GetPidJNI
 * Method:    getpid
 * Signature: ()J
 */
JNIEXPORT jlong JNICALL Java_com_headius_jnr_1presentation_GetPidJNI_getpid
  (JNIEnv *, jclass);

#ifdef __cplusplus
}
#endif
#endif
```

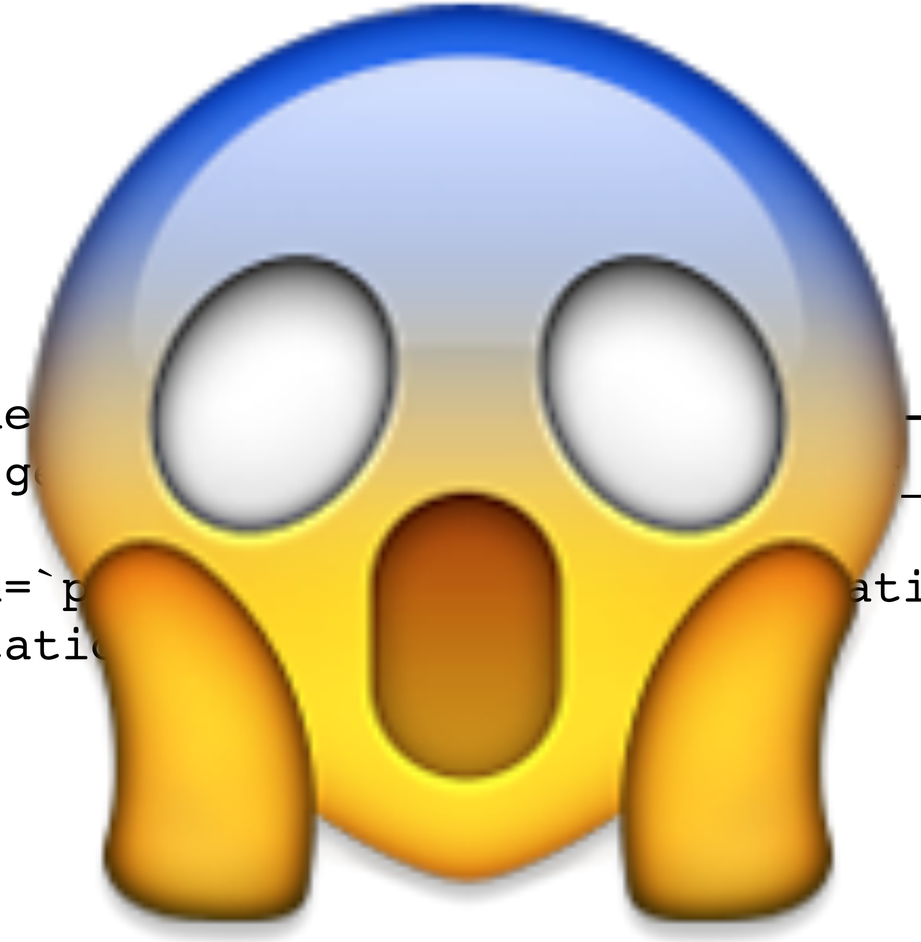
```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class com_headius_jnr                    ion_GetPidJNI */

#ifndef _Included_com_headius_                          idJNI
#define _Included_com_headius                           JNI
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     com_headius_j
 * Method:    getpid
 * Signature: ()J
 */
JNIEXPORT jlong JNICALL Java_                           tation_GetPidJNI_getpid
  (JNIEnv *, jclass);

#ifdef __cplusplus
}
#endif
#endif
```



```
#include "com_headius_jnr_presentation_GetPidJNI.h"

jlong JNICALL
Java_com_headius_jnr_1presentation_GetPidJNI_getpid
  (JNIEnv *env, jclass c) {

  return getpid();
}
```

```
$ gcc -I $JAVA_HOME/include -I $JAVA_HOME/include/darwin -L $JAVA_HOME/jre/lib/ \
    -dynamiclib -ljava -o getpidjni.dylib com_headius_jnr_presentation_GetPidJNI.c

$ java -Djava.library.path=`pwd` -cp target/jnr_presentation-1.0-SNAPSHOT.jar \
   com.headius.jnr_presentation.GetPidJNI
```

```
$ gcc -I $JAVA_HOME/include                                  -L $JAVA_HOME/jre/lib/ \
    -dynamiclib -ljava -o g                                  _presentation_GetPidJNI.c

$ java -Djava.library.path=`p                                ation-1.0-SNAPSHOT.jar \
   com.headius.jnr_presentati
```

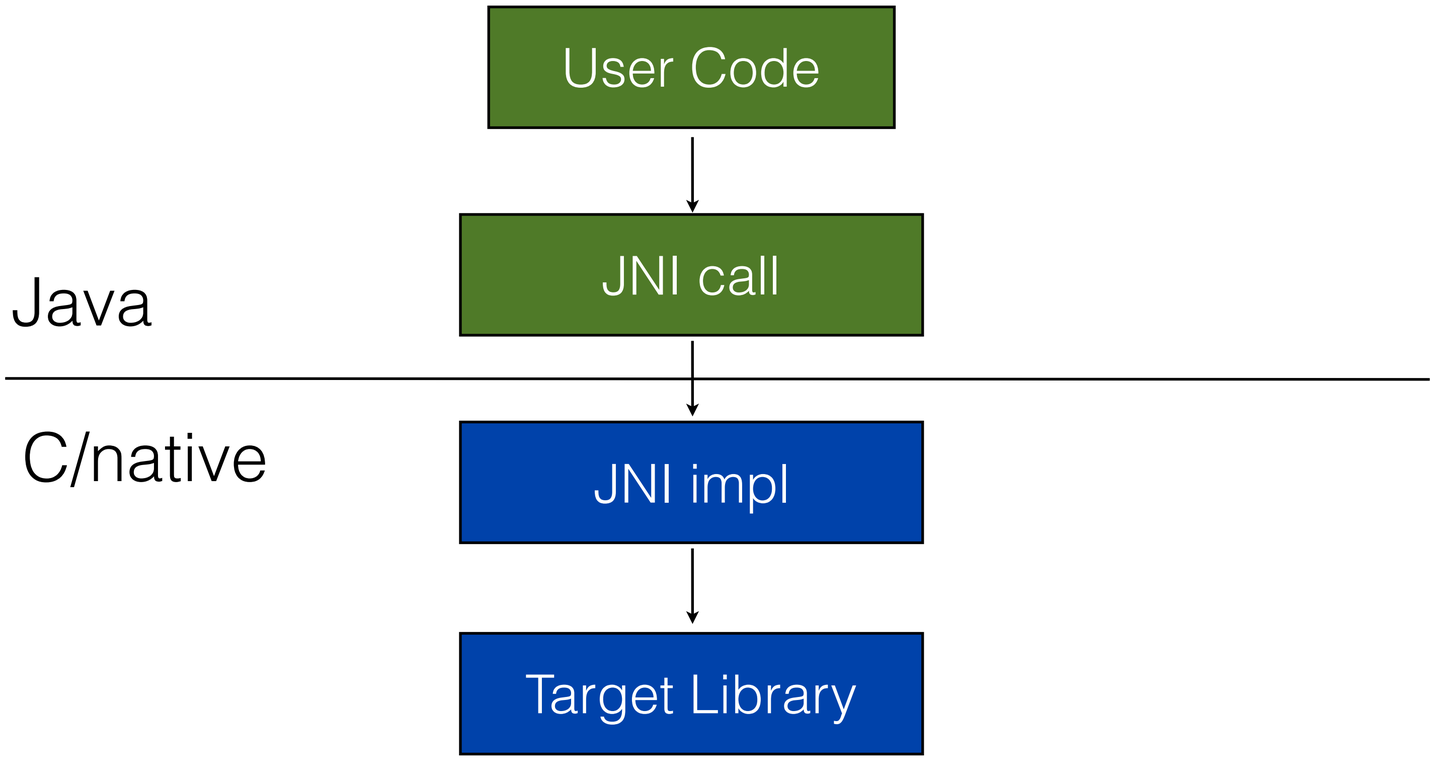


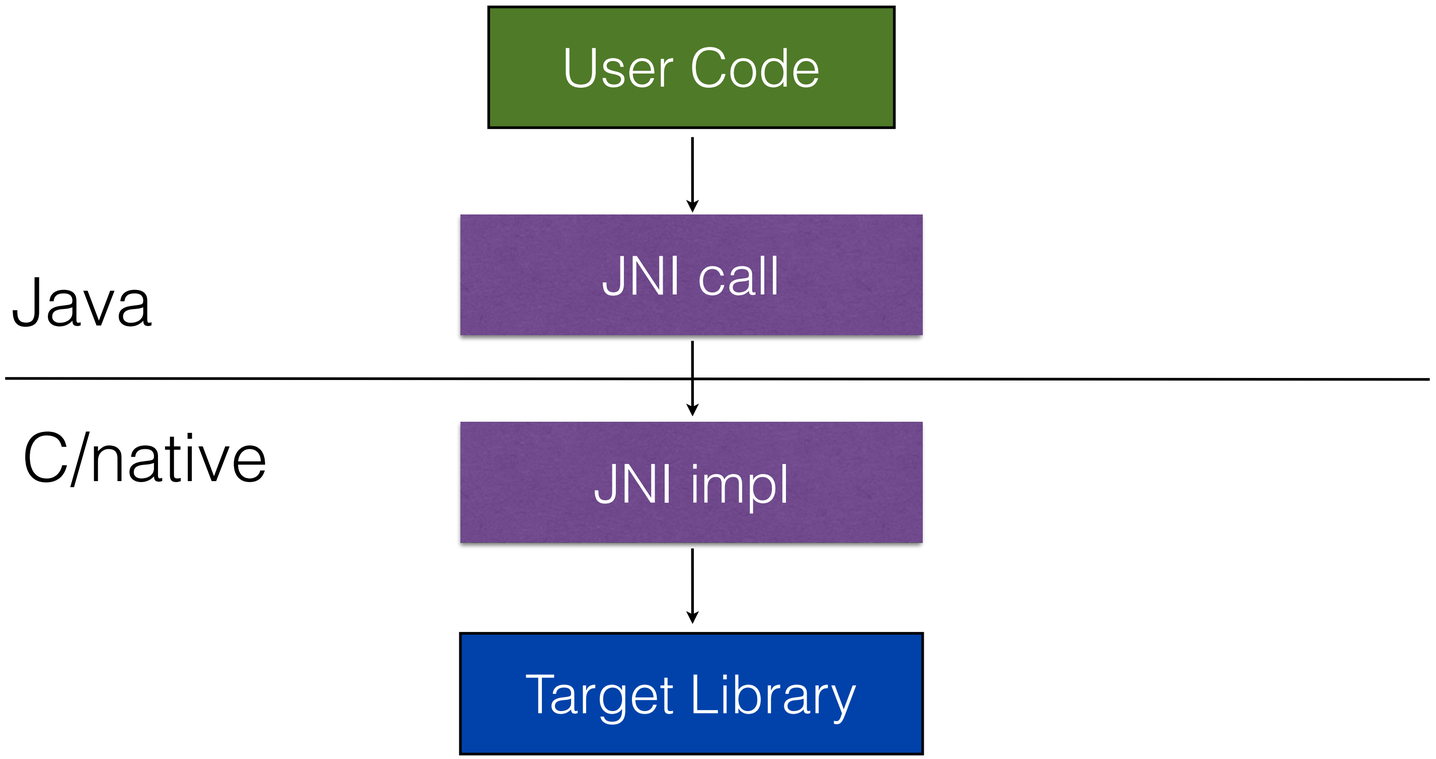
Why is it so bad?

- C requires a different skillset
  - We are Java developers, not C hackers
- Building shared libs across platforms
  - I just want to call a function!
- JNI API is extensive and arcane
  - VERY easy to make mistakes

But We Need It!

- NIO, NIO.2
  - Native IO, symlinks, FS-walking,
- Unmanaged memory
- Selectable stdio, process IO
- Low-level or other sockets (UNIX, ICMP, ...)
- New APIs (graphics, crypto, OS, ...)


User Code
JNI call
Java
C/native
JNI impl
Target Library


User Code
JNI call
Java
C/native
JNI impl
Target Library

There must be a better way!

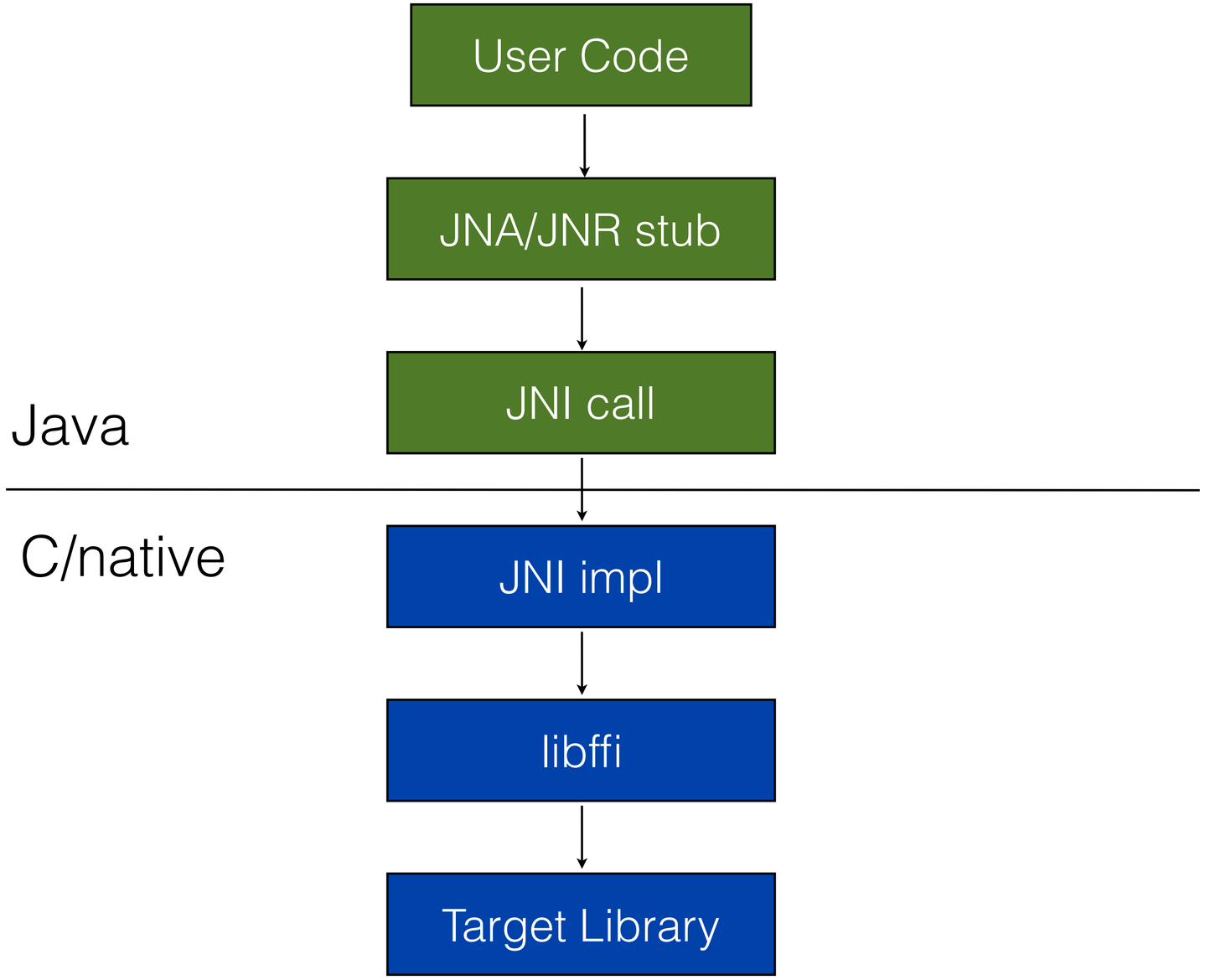
Java Native Runtime

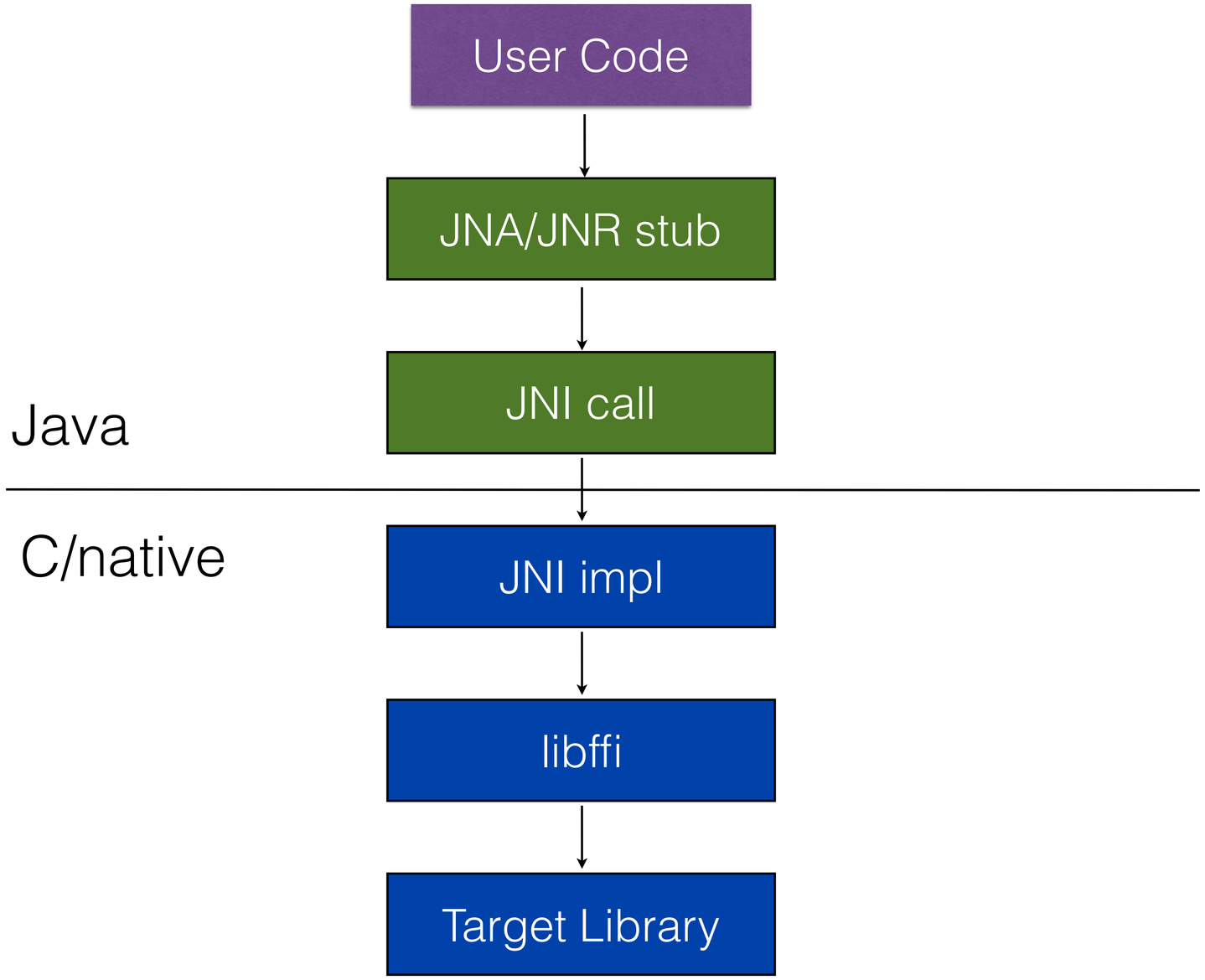
https://github.com/jnr

Java Native Runtime

- **Java** API
- for calling **Native** code
- supported by a rich **Runtime** library
- You may be familiar with **JNA**
- https://github.com/jnr

A Java API for binding native libraries and native memory

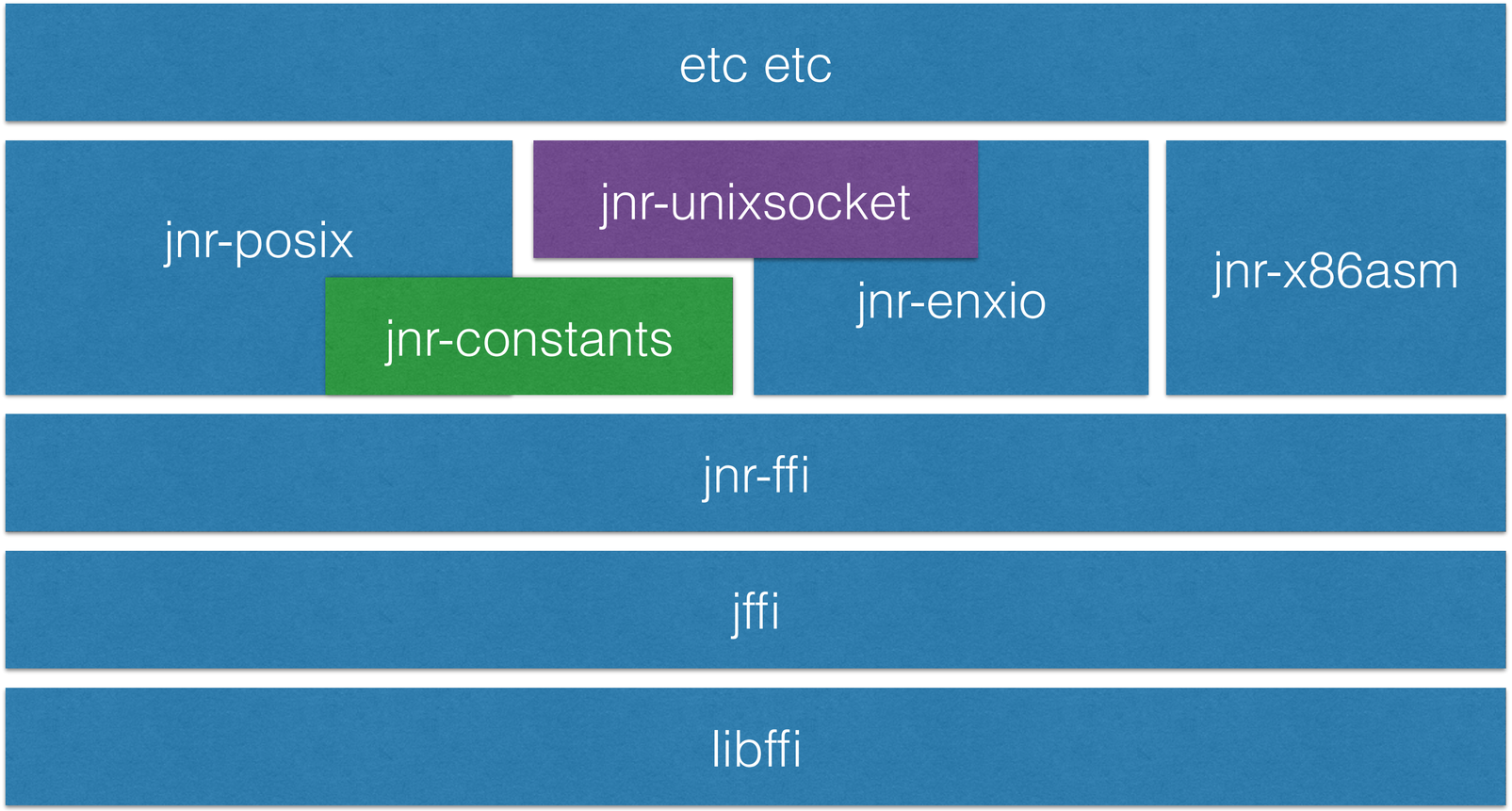

User Code
JNA/JNR stub
JNI call
Java
C/native
JNI impl
libffi
Target Library


User Code
JNA/JNR stub
JNI call
Java
C/native
JNI impl
libffi
Target Library

JNR

```
import jnr.ffi.LibraryLoader;
import jnr.ffi.annotations.IgnoreError;

public class GetPidJNRExample {
    public interface GetPid {
        long getpid();
    }

    public static void main( String[] args ) {
        GetPid getpid = LibraryLoader
                    .create(GetPid.class)
                    .load("c");

        getpid.getpid();
    }
}
```

Layered Runtime



jffi

jffi

- Java Foreign Function Interface
- libffi-based
- Low-level...not the API you're looking for
- Broad platform support
- https://github.com/jnr/jffi

JFFI Platforms

- Darwin (OS X): universal (+ppc?)
- Linux: i386, x86_64, arm, ppc, ppc64, s390x
- Windows: i386, x86_64
- FreeBSD, OpenBSD: i386, x86_64
- SunOS: i386, x86_64, sparc, sparcv9
- AIX: ppc
- ARM, AArch64
- OpenVMS, AS/400: builds out there somewhere
- If your platform isn't here, contribute a build

JNR-FFI

jnr-ffi

- User-oriented API
- Roughly equivalent to what JNA gives you
- Functions, structs, callbacks, memory
- https://github.com/jnr/jnr-ffi
- https://github.com/jnr/jnr-ffi-examples

Why Not JNA?

- Not low-level enough to build other APIs
- Preprocessor constants?
- Standard API sets out of the box
- C callbacks?
- Performance?!?

```java
public static final class Timeval extends Struct {
    public final time_t tv_sec = new time_t();
    public final SignedLong tv_usec = new SignedLong();

    public Timeval(Runtime runtime) {
        super(runtime);
    }
}
```

```
public interface LibC  {
    public int gettimeofday(@Out @Transient Timeval tv, Pointer unused);
}
```

```java
public static void main( String[] args ) {
    LibC libc = LibraryLoader.create(LibC.class).load("c");

    Runtime runtime = Runtime.getRuntime(libc);
    Timeval tv = new Timeval(runtime);

    libc.gettimeofday(tv, null);

    System.out.println("gettimeofday tv.tv_sec=" + tv.tv_sec.get()
                                + " tv.tv_usec=" + tv.tv_usec.get());
}
```

Support Libraries

jnr-constants

jnr-constants

- Preprocessor constants (#define)
- Generation tools
- Several platforms, families built in
- https://github.com/jnr/jnr-constants

Provided Constants

- AddressFamily.java
- ConstantResolver.java
- Errno.java
- Fcntl.java
- INAddr.java
- IPProto.java
- NameInfo.java
- OpenFlags.java
- PRIO.java
- ProtocolFamily.java
- RLIM.java
- RLIMIT.java
- Shutdown.java
- Signal.java
- Sock.java
- SocketLevel.java
- SocketOption.java
- Sysconf.java
- TCP.java
- WaitFlags.java

```
SOCKET(2)          BSD System Calls Manual         SOCKET(2)

NAME
     socket -- create an endpoint for communication

SYNOPSIS
     #include <sys/socket.h>

     int
     socket(int domain, int type, int protocol);
```

```
SOCKET(2)          BSD System Calls Manual          SOCKET(2)

NAME
     socket -- create an endpoint for communication

SYNOPSIS
     #include <sys/socket.h>

     int
     socket(int domain, int type, int protocol);
```

domain

PF_LOCAL	Host-internal protocols, formerly called PF_UNIX,
PF_UNIX	Host-internal protocols, deprecated, use PF_LOCAL,
PF_INET	Internet version 4 protocols,
PF_ROUTE	Internal Routing protocol,
PF_KEY	Internal key-management function,
PF_INET6	Internet version 6 protocols,
PF_SYSTEM	System domain,
PF_NDRV	Raw access to network device

type

```
SOCK_STREAM     streaming sockets
SOCK_DGRAM      datagram sockets
SOCK_RAW        raw, for internal protocols
SOCK_SEQPACKET  reliable connection-based datagrams
```

```java
// WARNING: This file is autogenerated. DO NOT EDIT!
// Generated Tue Feb 24 09:44:06 +1000 2009
package jnr.constants.platform.linux;
public enum Sock implements jnr.constants.Constant {
SOCK_STREAM(1),
SOCK_DGRAM(2),
SOCK_RAW(3),
SOCK_RDM(4),
SOCK_SEQPACKET(5);
// SOCK_MAXADDRLEN not defined
private final int value;
private Sock(int value) { this.value = value; }
public static final long MIN_VALUE = 1;
public static final long MAX_VALUE = 5;

public final int value() { return value; }
public final int intValue() { return value; }
public final long longValue() { return value; }
}
```

jnr-posix

- Pre-bound set of POSIX functions
- Mostly driven by what JRuby, Jython use
- Goal: 100% of POSIX bound to Java
- Bonus: partial pure-Java backend

```
public int chmod(String string, int i);
public int chown(String string, int i, int i1);
public int execv(String string, String[] strings);
public int execve(String string, String[] strings, String[] strings1);
public int fork();
public int seteuid(int i);
public int getgid();
public String getlogin();
public int getpgid();
public int getpgid(int i);
public int getpgrp();
public int getpid();
public int getppid();
public Passwd getpwent();
public Passwd getpwuid(int i);
public Passwd getpwnam(String string);
public Group getgrgid(int i);
public Group getgrnam(String string);
public int getuid();
public boolean isatty(FileDescriptor fd);
public int kill(int i, int i1);
public int symlink(String string, String string1);
public int link(String string, String string1);
public String readlink(String string) throws IOException;
public String getenv(String string);
public int setenv(String string, String string1, int i);
public int unsetenv(String string);
public int getpriority(int i, int i1);
public int setpriority(int i, int i1, int i2);
public int setuid(int i);
public FileStat stat(String string);
public int stat(String string, FileStat fs);
public int umask(int i);
public Times times();
public int utimes(String string, long[] longs, long[] longs1);
public int waitpid(int i, int[] ints, int i1);
public int wait(int[] ints);
public int errno();
public void errno(int i);
public int posix_spawnp(String string, List<? extends SpawnFileAction>
list, List<? extends CharSequence> list1, List<? extends CharSequence>
list2);
```

jnr-enxio

jnr-enxio

- Extended Native X-platform IO
- NIO-compatible JNR-backed IO library
  - Read, write, select (kqueue, epoll, etc)
  - Low-level fcntl control
- https://github.com/jnr/jnr-enxio

```java
public class NativeSocketChannel
        extends AbstractSelectableChannel
        implements ByteChannel, NativeSelectableChannel {
    public NativeSocketChannel(int fd);
    public NativeSocketChannel(int fd, int ops);
    public final int validOps();
    public final int getFD();
    public int read(ByteBuffer dst) throws IOException;
    public int write(ByteBuffer src) throws IOException
    public void shutdownInput() throws IOException;
    public void shutdownOutput() throws IOException;
}
```

```
int[] sockets = { -1, -1 };
Native.socketpair(ProtocolFamily.PF_UNIX, Sock.SOCK_STREAM, 0, sockets);
return new NativeSocketChannel[] {
        new NativeSocketChannel(sockets[0]),
        new NativeSocketChannel(sockets[1])
};
```

jnr-unixsocket

jnr-unixsocket

- UNIX sockets for NIO
- Built atop jnr-enxio
- Fully selectable, etc
- https://github.com/jnr/jnr-unixsocket

```
int[] sockets = { -1, -1 };
Native.socketpair(ProtocolFamily.PF_UNIX, Sock.SOCK_STREAM, 0, sockets);
return new UnixSocketChannel[] {
        new UnixSocketChannel(sockets[0], State.CONNECTED, true),
        new UnixSocketChannel(sockets[1], State.CONNECTED, true) };
```

```java
String PATH = "foobarbaz";

UnixSocketAddress a = new UnixSocketAddress(PATH);
UnixSocketChannel ch = UnixSocketChannel.open();

ch.bind(a);
```

jnr-process

jnr-process

- java.lang.Process/Builder lookalike
- Built atop enxio channels and posix_spawn
- Finally, real, interactive subprocesses!
- https://github.com/jnr/jnr-process

```java
public SelectableChannel getOut() {
    return out;
}

public SelectableChannel getIn() {
    return in;
}

public SelectableChannel getErr() {
    return err;
}

public long getPid() {
    return pid;
}
```

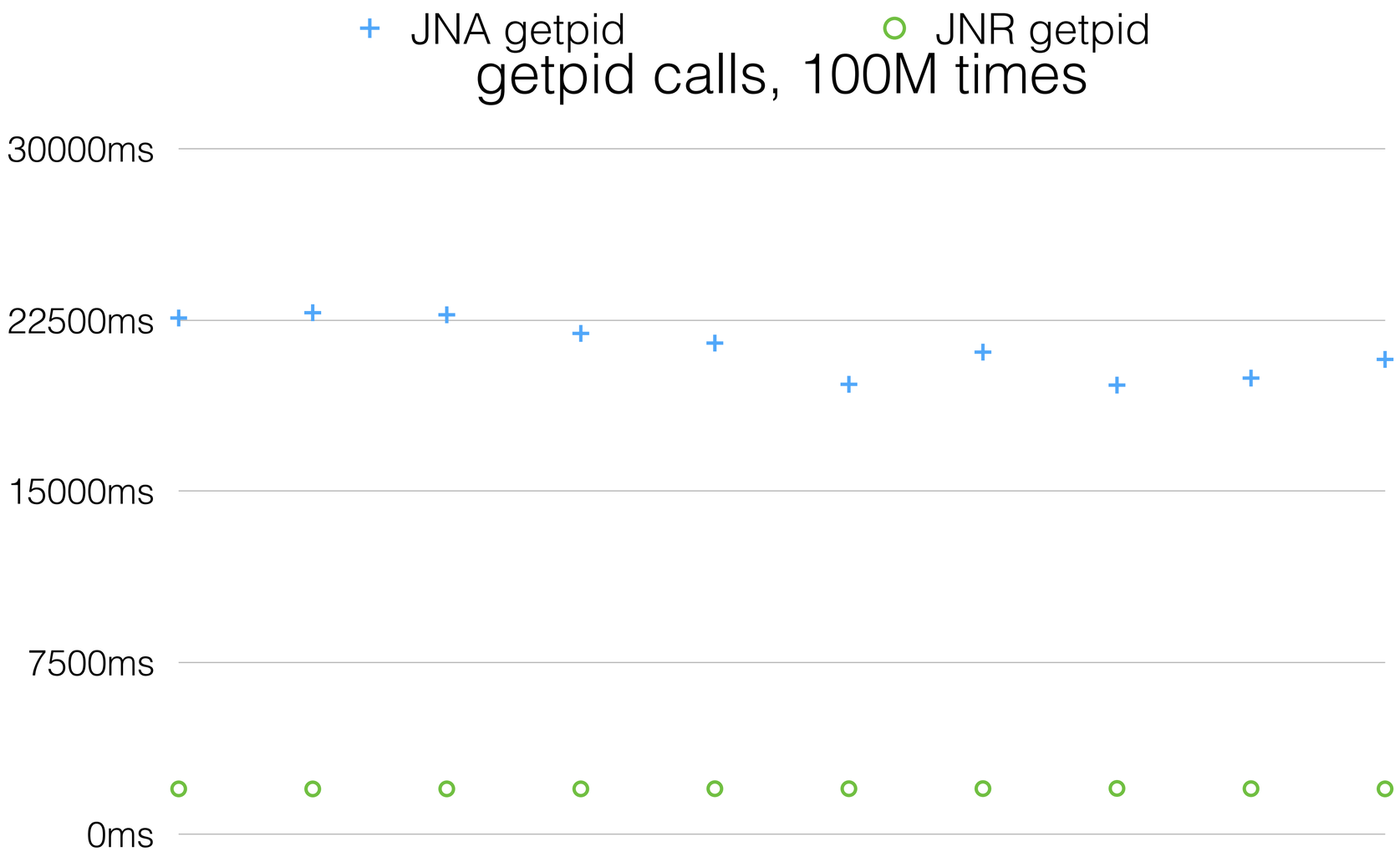
Performance

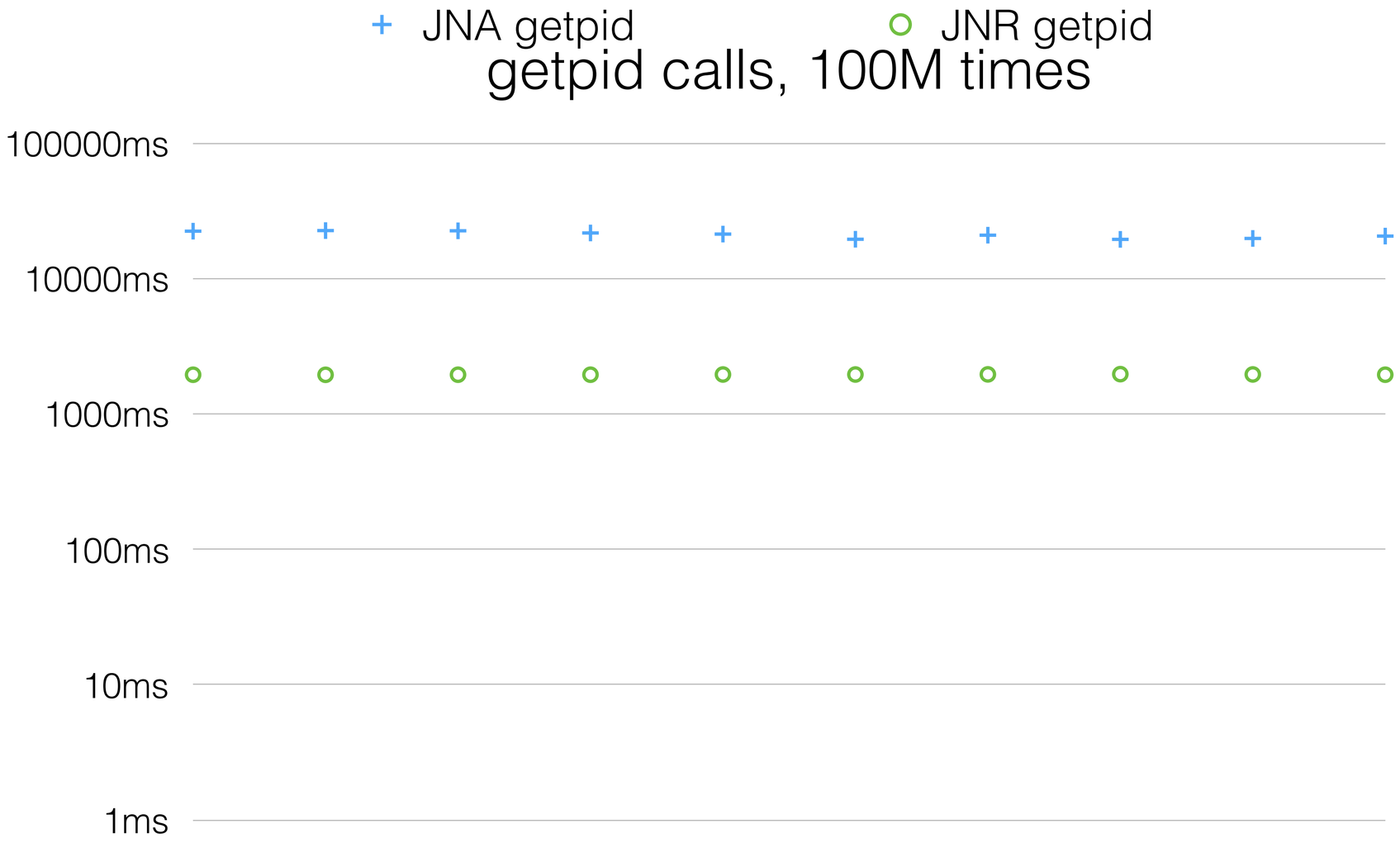
Trying Really Hard...

- Generated bytecode stubs
- Invokedynamic binding (in JRuby)
- Generated JNI stubs
- Best case: 2-3 hops to target function

jnr-x86asm

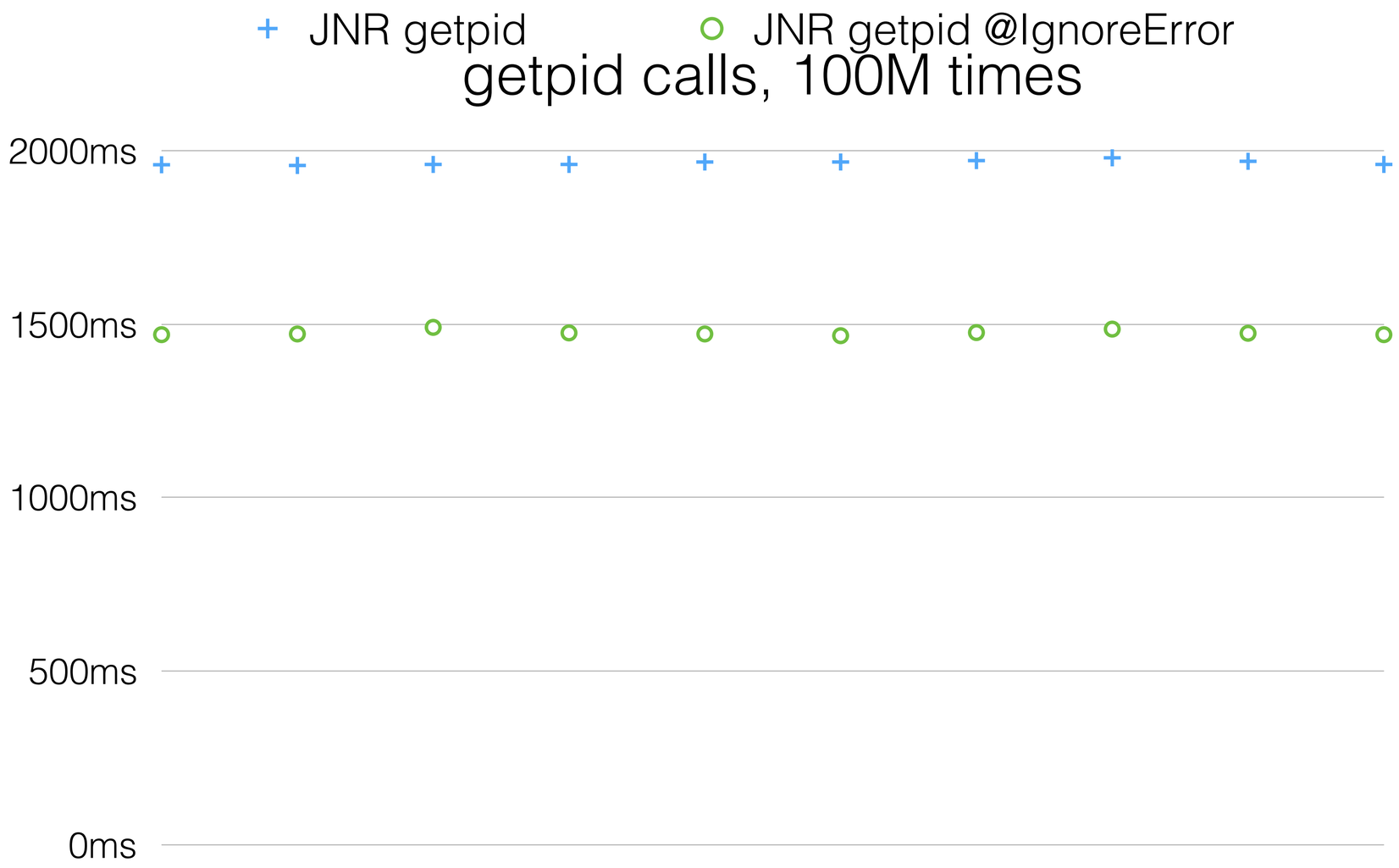
- Generate and link ASM via JNI
- Used internally by jnr-ffi
- https://github.com/jnr/jnr-x86asm


JNA getpid
JNR getpid
getpid calls, 100M times
30000ms
22500ms
15000ms
7500ms
0ms

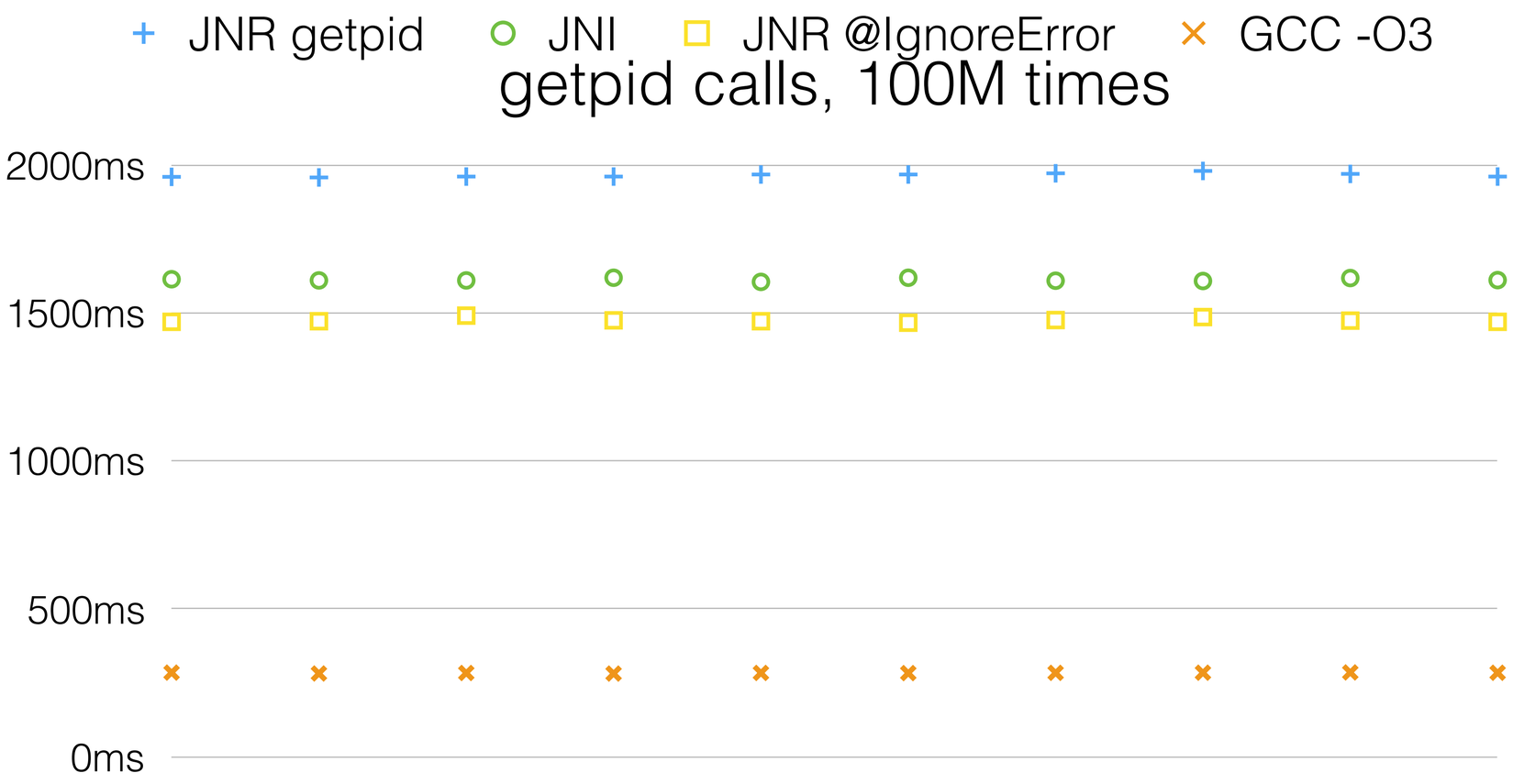

JNA getpid
JNR getpid
getpid calls, 100M times
100000ms
10000ms
1000ms
100ms
10ms
1ms

Ignore errno

```
import jnr.ffi.LibraryLoader;
import jnr.ffi.annotations.IgnoreError;

public class GetPidJNRExample {
    public interface GetPid {
        @IgnoreError
        long getpid();
    }

    public static void main( String[] args ) {
        GetPid getpid = LibraryLoader
                    .create(GetPid.class)
                    .load("c");

        getpid.getpid();
    }
}
```


JNR getpid
JNR getpid @IgnoreError
getpid calls, 100M times
2000ms
1500ms
1000ms
500ms
0ms

But There's More to Do



```
mov    0x8(%rbx),%r11d    ; implicit exception: dispatches to 0x000000010e31f0f8
cmp    $0x2232118f,%r11d  ;   {oop('.../GetPidJNRExample$GetPid$jnr$ffi$0')}
jne    0x000000010e31f0f8 ;*aload_0
                          ; - GetPidJNRExample::benchGetPid@12 (line 26)
mov    %rbx,%r10          ;*invokeinterface getpid
                          ; - GetPidJNRExample::benchGetPid@13 (line 26)
jmp    0x000000010e31f049
...
mov    %r10,0x8(%rsp)
mov    %r13,(%rsp)        ;*aload_0
                          ; - GetPidJNRExample::benchGetPid@12 (line 26)
mov    %r10,%rsi
xchg   %ax,%ax
callq  0x000000010e2cfc60 ; OopMap{[8]=Oop off=156}
                          ;*invokeinterface getpid
                          ; - GetPidJNRExample::benchGetPid@13 (line 26)
                          ;   {optimized virtual_call}
```

```
mov    0x8(%rbx),%r11d    ; implicit exception: dispatches to 0x000000010e31f0f8
cmp    $0x2232118f,%r11d  ;   {oop('.../GetPidJNRExample$GetPid$jnr$ffi$0')}
jne    0x000000010e31f0f8 ;*aload_0
                          ; - GetPidJNRExample::benchGetPid@12 (line 26)
mov    %rbx,%r10          ;*invokeinterface getpid
                          ; - GetPidJNRExample::benchGetPid@13 (line 26)
jmp    0x000000010e31f049
...
mov    %r10,0x8(%rsp)
mov    %r13,(%rsp)        ;*aload_0
                          ; - GetPidJNRExample::benchGetPid@12 (line 26)
mov    %r10,%rsi
xchg   %ax,%ax
callq  0x000000010e2cfc60 ; OopMap{[8]=Oop off=156}
                          ;*invokeinterface getpid
                          ; - GetPidJNRExample::benchGetPid@13 (line 26)
                          ;   {optimized virtual_call}
```

```
{0x000000014ba85810} 'invokeI0' '(JJ)I' in 'com/kenai/jffi/Foreign')}
 0x000000011192233f: mov    %rsp,%rbp
                     ;*invokestatic linkToStatic
                     ; - java.lang.invoke.LambdaForm$DMH/731395981::invokeStatic_JJ_I@13
                     ; - java.lang.invoke.LambdaForm$BMH/1757676444::reinvoke@32
                     ; - java.lang.invoke.LambdaForm$MH/892529689::collect@5
                     ; - java.lang.invoke.LambdaForm$DMH/1058025095::invokeSpecial_LLLL_L@16
                     ; - java.lang.invoke.LambdaForm$MH/152134087::guard@50
                     ; - java.lang.invoke.LambdaForm$DMH/1058025095::invokeSpecial_LLLL_L@16
                     ; - java.lang.invoke.LambdaForm$MH/1580893732::guard@50
                     ; - java.lang.invoke.LambdaForm$MH/1781256139::linkToCallSite@14
                     ; - getpid_bench::method__1$RUBY$go@9 (line 13)
```

```
callq  <getpid address>   ; - libSystem.B.dylib
                          ;*invokeinterface getpid
                          ; - GetPidJNRExample::benchGetPid@13 (line 26)
                          ;   {optimized virtual_call}
```

We Need JVM Help

- Standard FFI API in JDK
- JIT intelligence
  - Drop JNI overhead where possible
  - Bind native call directly at call site
- Security policies, segv protection, etc

Project Panama

- Coming soon to a JVM near you!
- Code generation
- Native memory access
- JIT intelligence
- Владимир Иванов

Thank You!

- Charles Oliver Nutter
- headius@headius.com
- @headius
- http://blog.headius.com