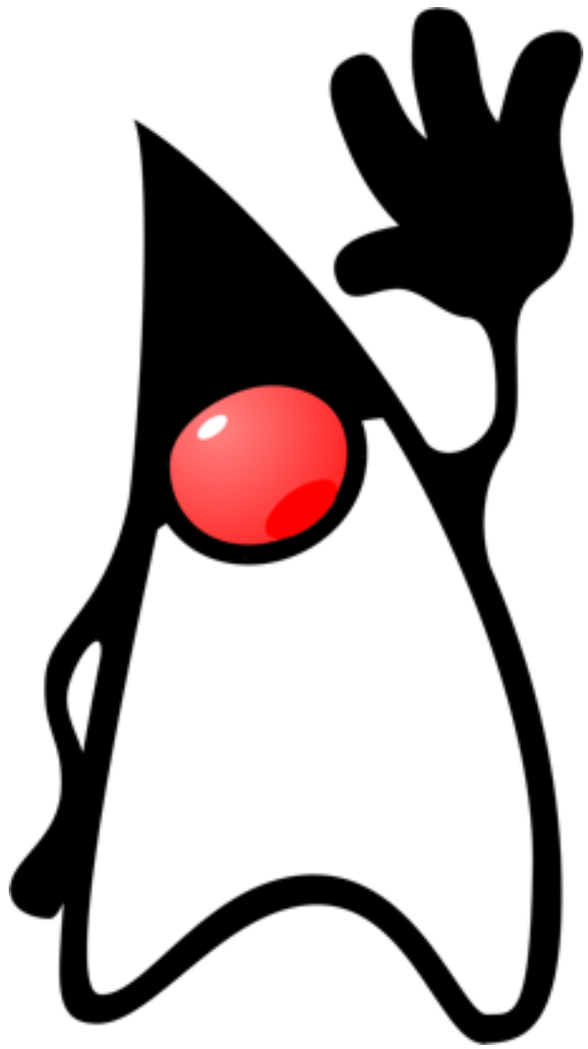
Beyond JVM

Charles Oliver Nutter
@headius

Me


JRuby

+



2001: JRuby is Born

2006: JRuby on Rails

2015: JRuby 9000

```
$ git log --oneline | wc -l
29435
```

Why do this for 9 years?

We like the JVM

Magical black box that runs our code

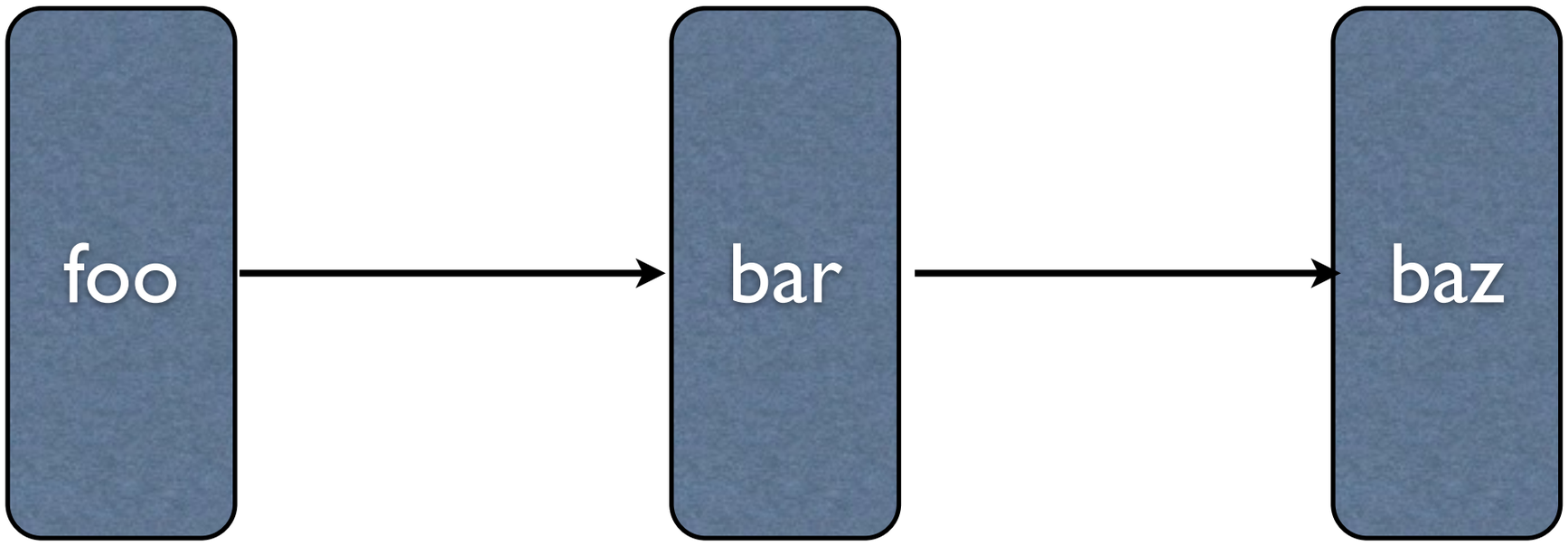
Except when it doesn’t

Ruby == Method Calls

Lots and lots and lots and lots...

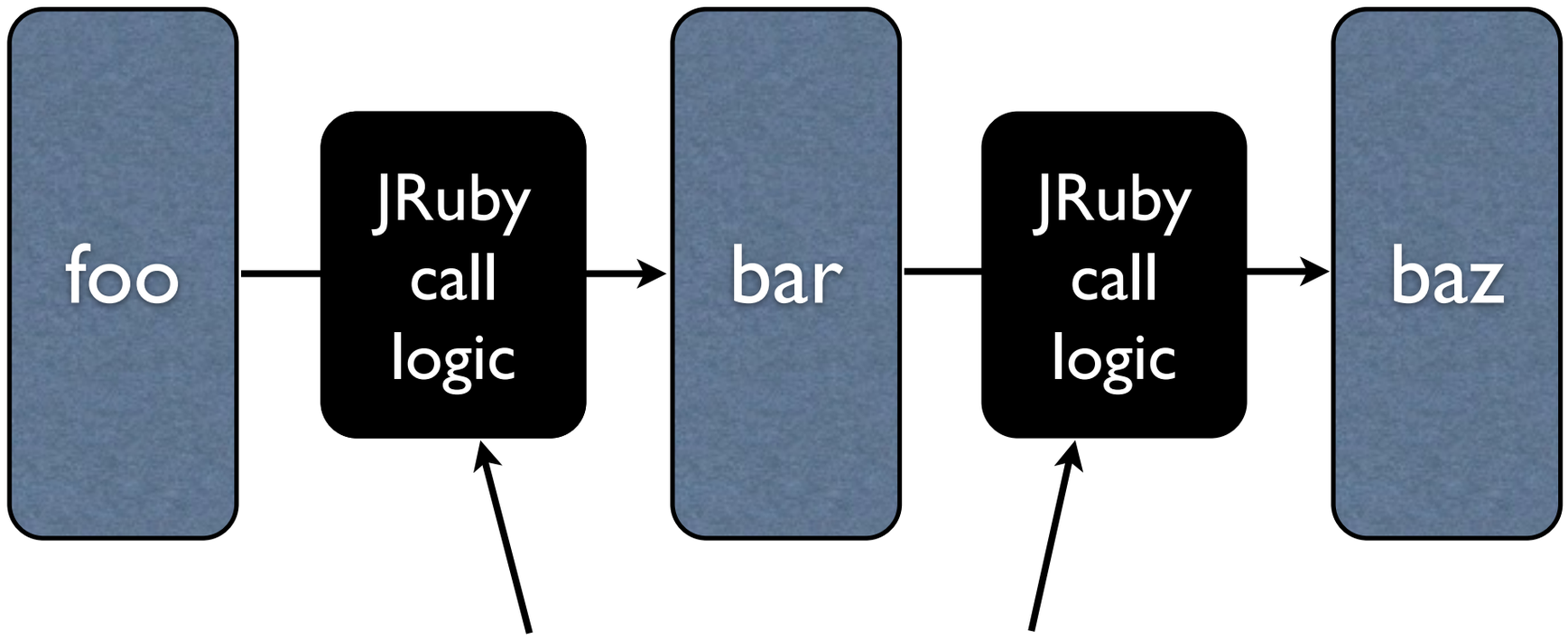
Command	Number of simple calls
jruby -e 1	1k
gem install rails	315k
rails new testapp	606k
rails simple CRUD	16k

```ruby
def foo
  bar
end

def bar
  baz
end

def baz
  # ...
end
```



JRuby on Java 5/6

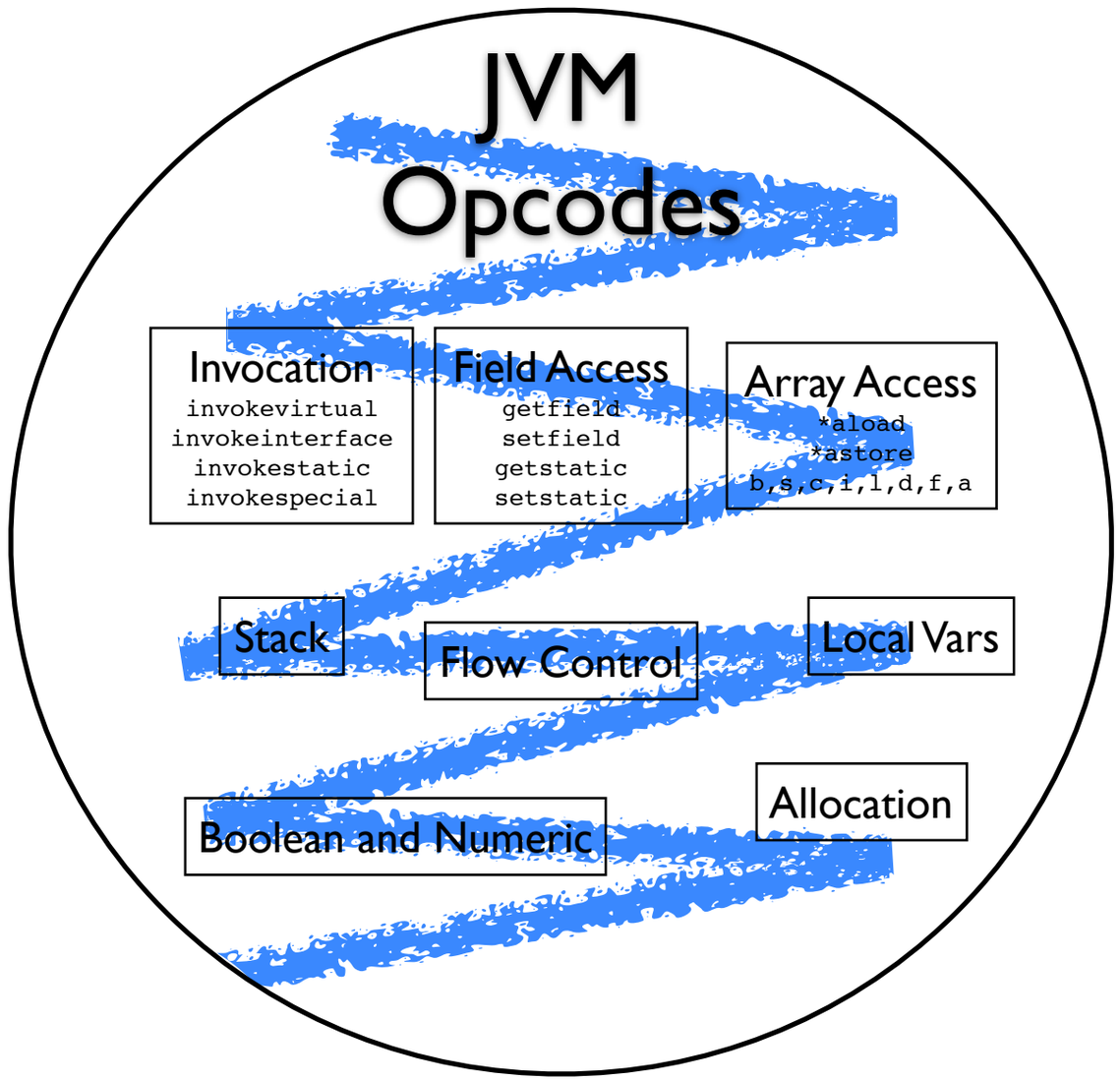
```
def foo
  bar
end

def bar
  baz
end

def baz
  # ...
end
```



Invokedynamic

History

- JVM authors mentioned non-Java languages
- Language authors have targeted JVM
  - Hundreds of JVM languages now
- But JVM was a mismatch for many of them
  - Usually required tricks that defeated JVM optimizations
  - Or required features JDK could not provide


JVM
Opcodes
Invocation
invokevirtual
invokeinterface
invokestatic
invokespecial
Field Access
getfield
setfield
getstatic
setstatic
Array Access
*aload
*astore
b,s,c,i,l,d,f,a
Stack
Flow Control
Local Vars
Boolean and Numeric
Allocation

"In the future, we will consider bounded extensions to the Java Virtual Machine to provide better support for other languages."

- JVM Specification First Edition (1997), preface

Goals of JSR 292

- A user-definable bytecode
  - Full freedom to define VM behavior
  - Fast method pointers + adapters
  - Caching and invalidation
  - Avoid future modifications

Indy Users

- Languages
  - Grew into it: JRuby, Groovy
    - Grew up with it: Dyn.js and Nashorn (JS)
- Java 8 lambdas
- Java 9 varhandles, specialied generics

How does it work?

Invoke

Invoke

```
// Static
System.currentTimeMillis()
Math.log(1.0)
```

Invoke

```
// Static
System.currentTimeMillis()
Math.log(1.0)

// Virtual
"hello".toUpperCase()
System.out.println()
```

Invoke

```
// Static
System.currentTimeMillis()
Math.log(1.0)

// Virtual
"hello".toUpperCase()
System.out.println()
```

```
// Interface
myList.add("happy happy")
myRunnable.run()
```

Invoke

```
// Static
System.currentTimeMillis()
Math.log(1.0)

// Virtual
"hello".toUpperCase()
System.out.println()
```

```
// Interface
myList.add("happy happy")
myRunnable.run()

// Constructor and super
new ArrayList()
super.equals(other)
```

```
// Static
invokestatic java/lang/System.currentTimeMillis:()J
invokestatic java/lang/Math.log:(D)D

// Virtual
invokevirtual java/lang/String.toUpperCase:()Ljava/lang/String;
invokevirtual java/io/PrintStream.println:()V

// Interface
invokeinterface java/util/List.add:(Ljava/lang/Object;)Z
invokeinterface java/lang/Runnable.add:()V

// Special
invokespecial java/util/ArrayList.<init>:()V
invokespecial java/lang/Object.equals:(java/lang/Object)Z
```

`invokevirtual`

1. Confirm object is of correct type
2. Confirm arguments are of correct type
3. Look up method on Java class
4. Cache method
5. Invoke method

`invokestatic`

1. Confirm arguments are of correct type
2. Look up method on Java class
3. Cache method
4. Invoke method

`invokeinterface`

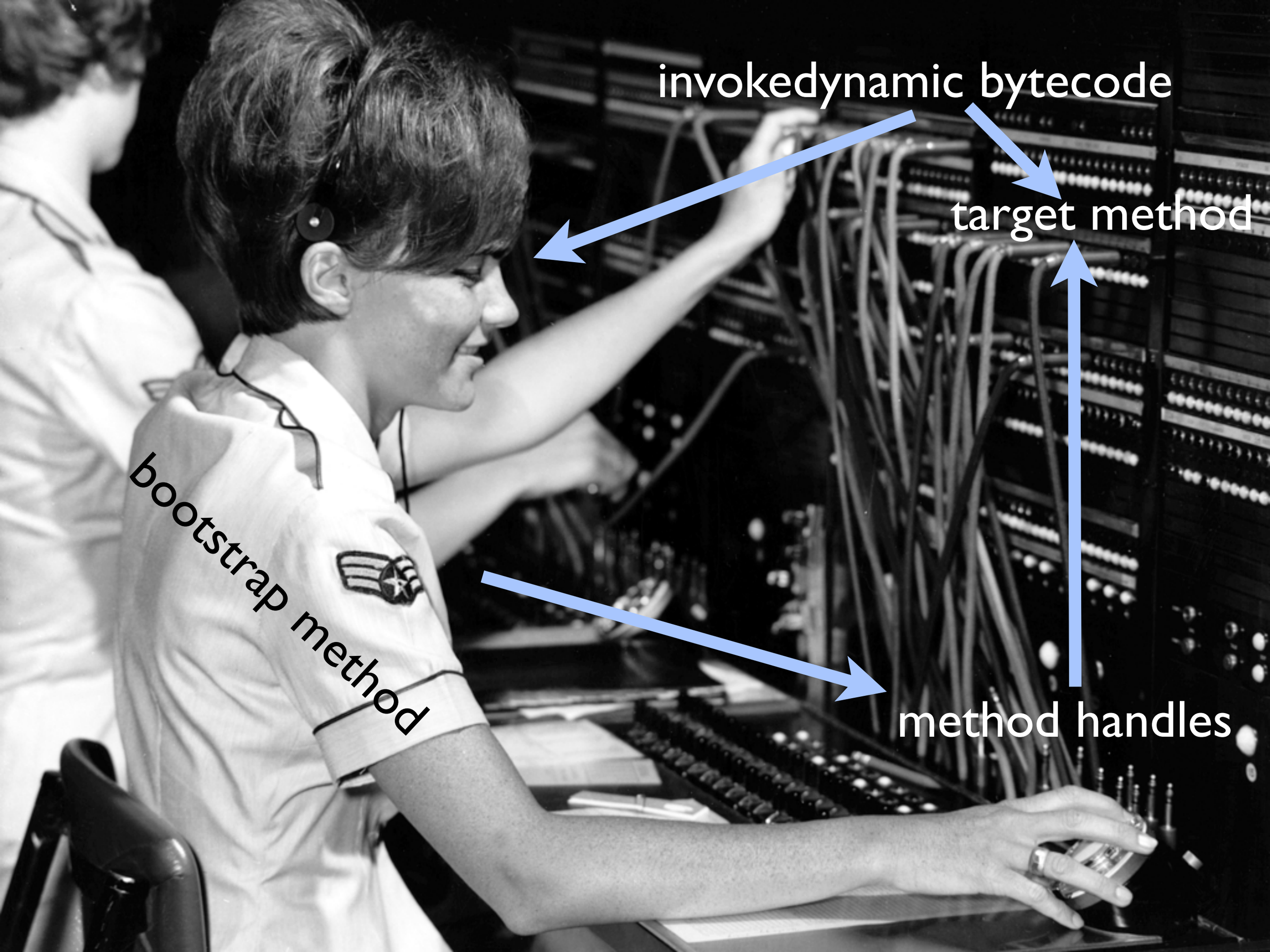
1. Confirm object's type implements interface
2. Confirm arguments are of correct type
3. Look up method on Java class
4. Cache method
5. Invoke method

`invokespecial`

1. Confirm object is of correct type
2. Confirm arguments are of correct type
3. Confirm target method is visible
4. Look up method on Java class
5. Cache method
6. Invoke method

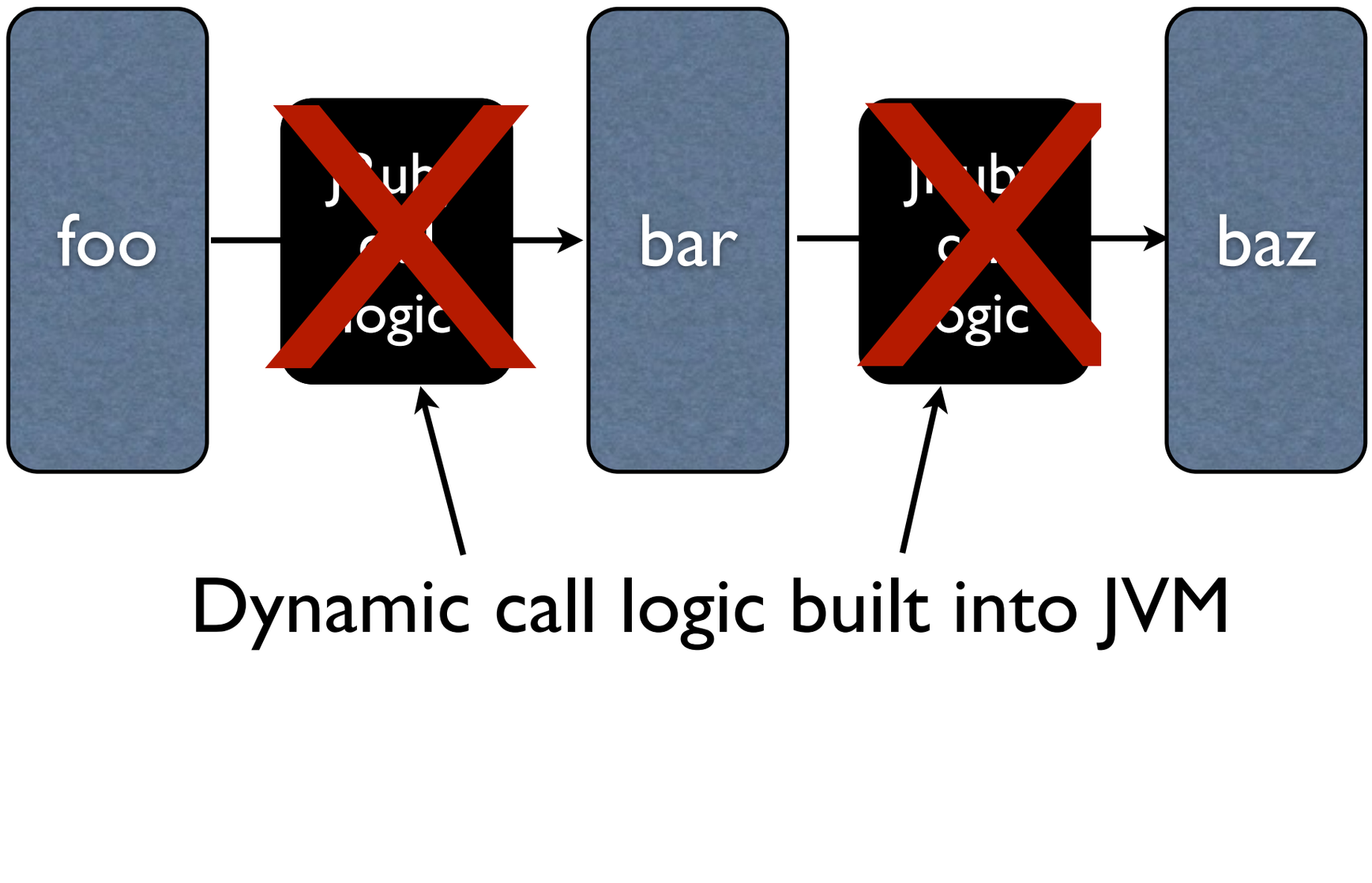
`invokedynamic`

1. Call your bootstrap code
2. Bootstrap wires up a target function
3. Target function invoked directly until you change it


invokedynamic bytecode
target method
bootstrap method
method handles

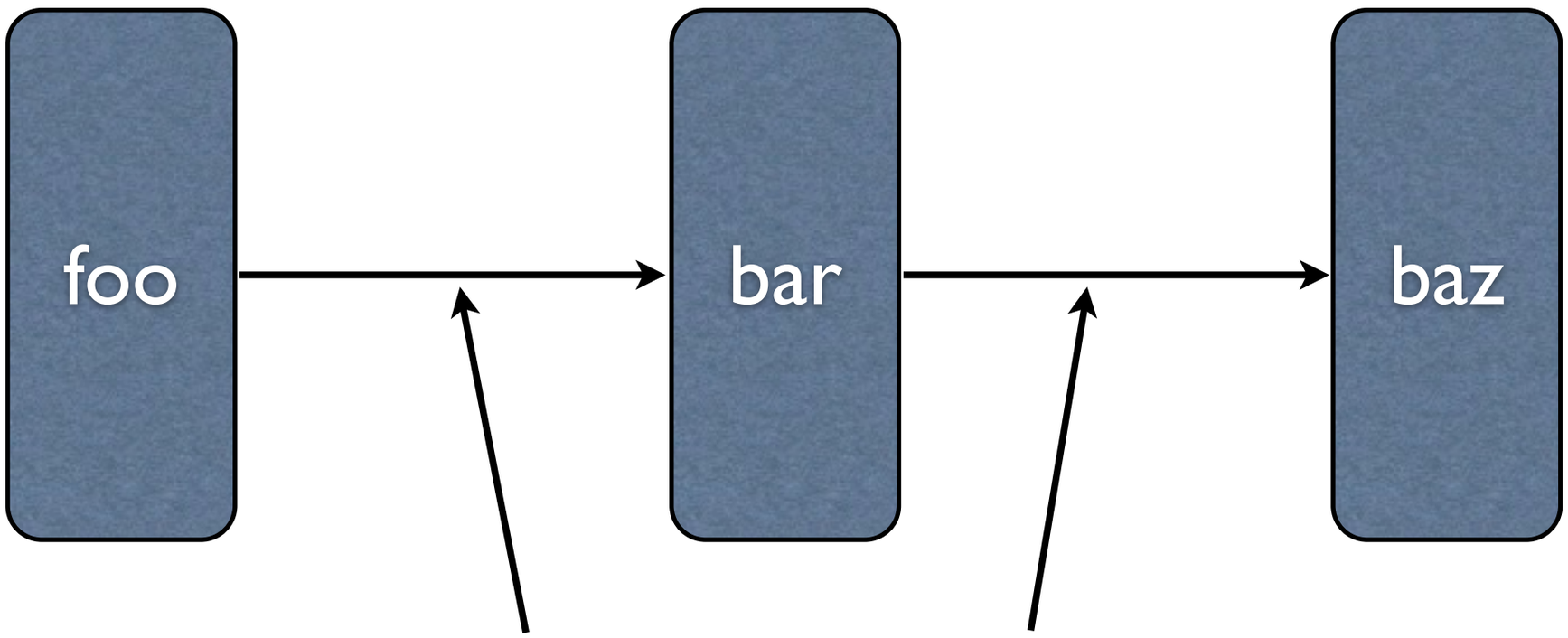
JRuby on Java 7

```
def foo
  bar
end

def bar
  baz
end

def baz
  # ...
end
```



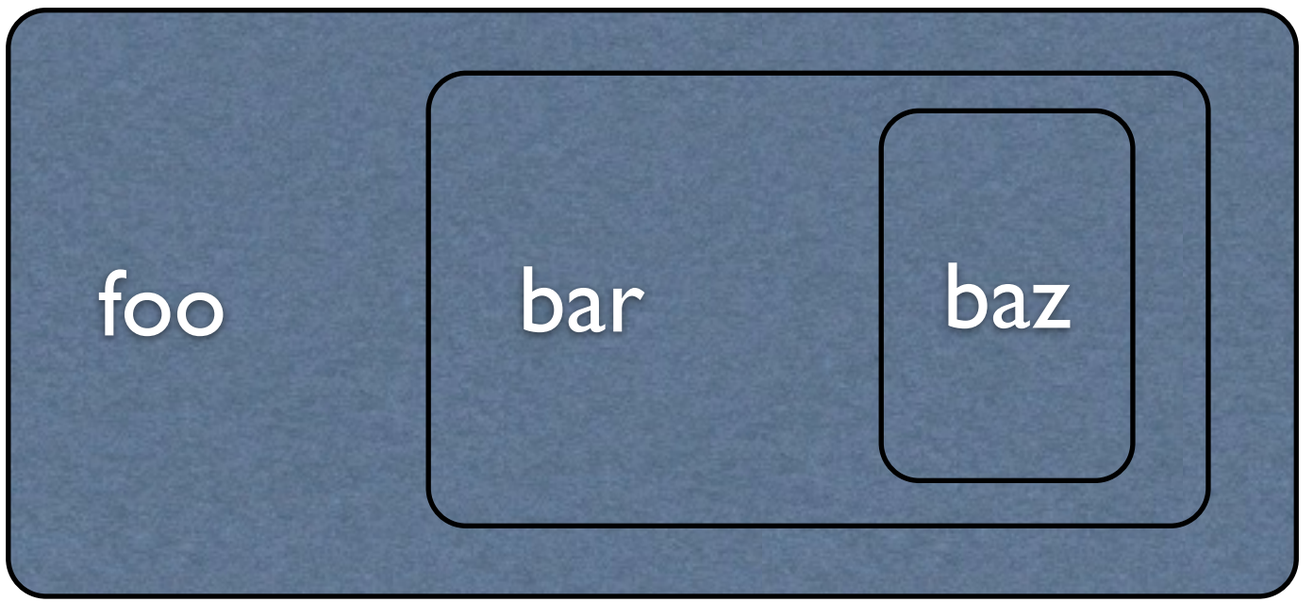
JRuby on Java 7

```
def foo
  bar
end

def bar
  baz
end

def baz
  # ...
end
```



```ruby
def foo
  bar
end

def bar
  baz
end

def baz
  # ...
end
```

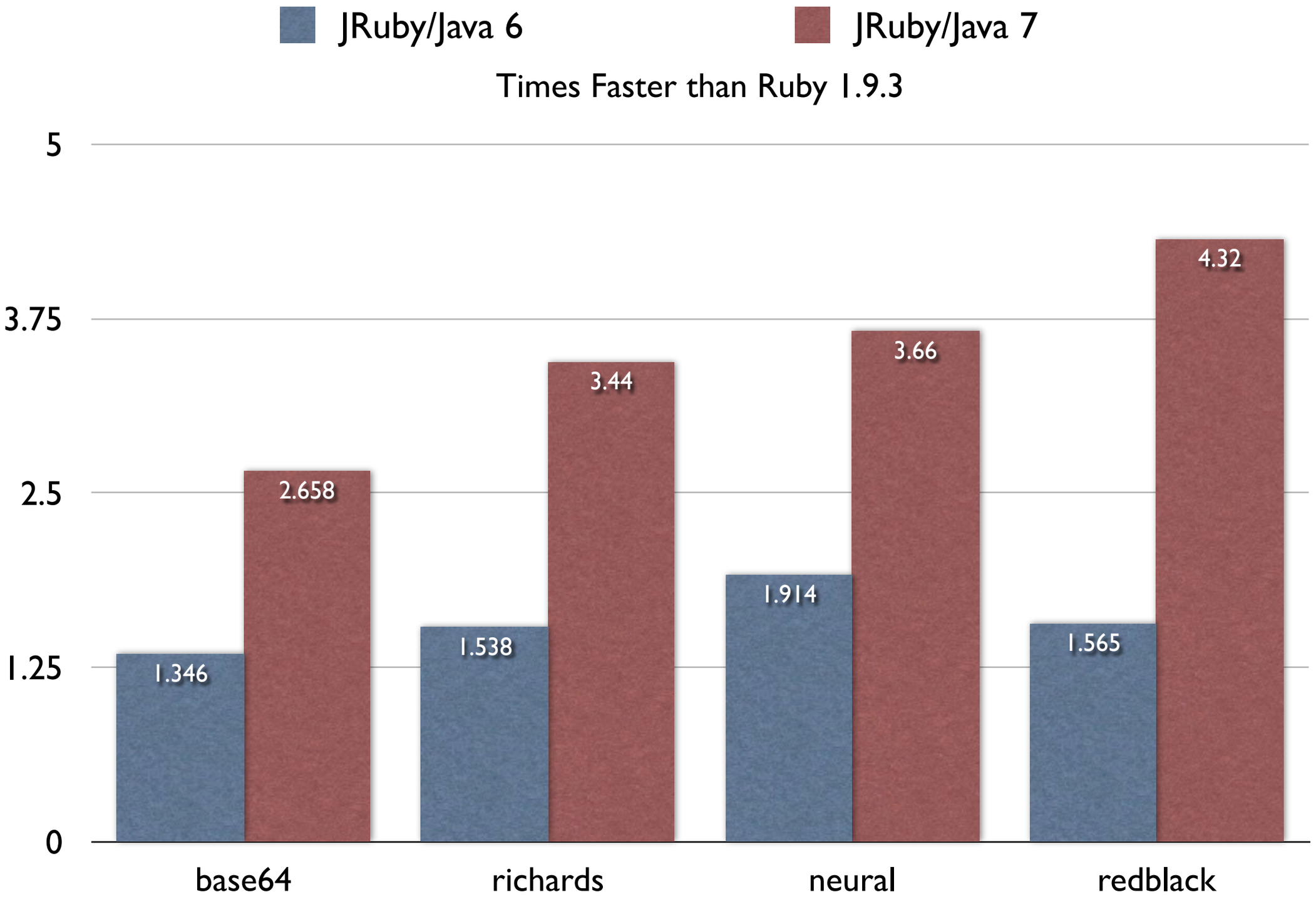
JRuby on Java 7

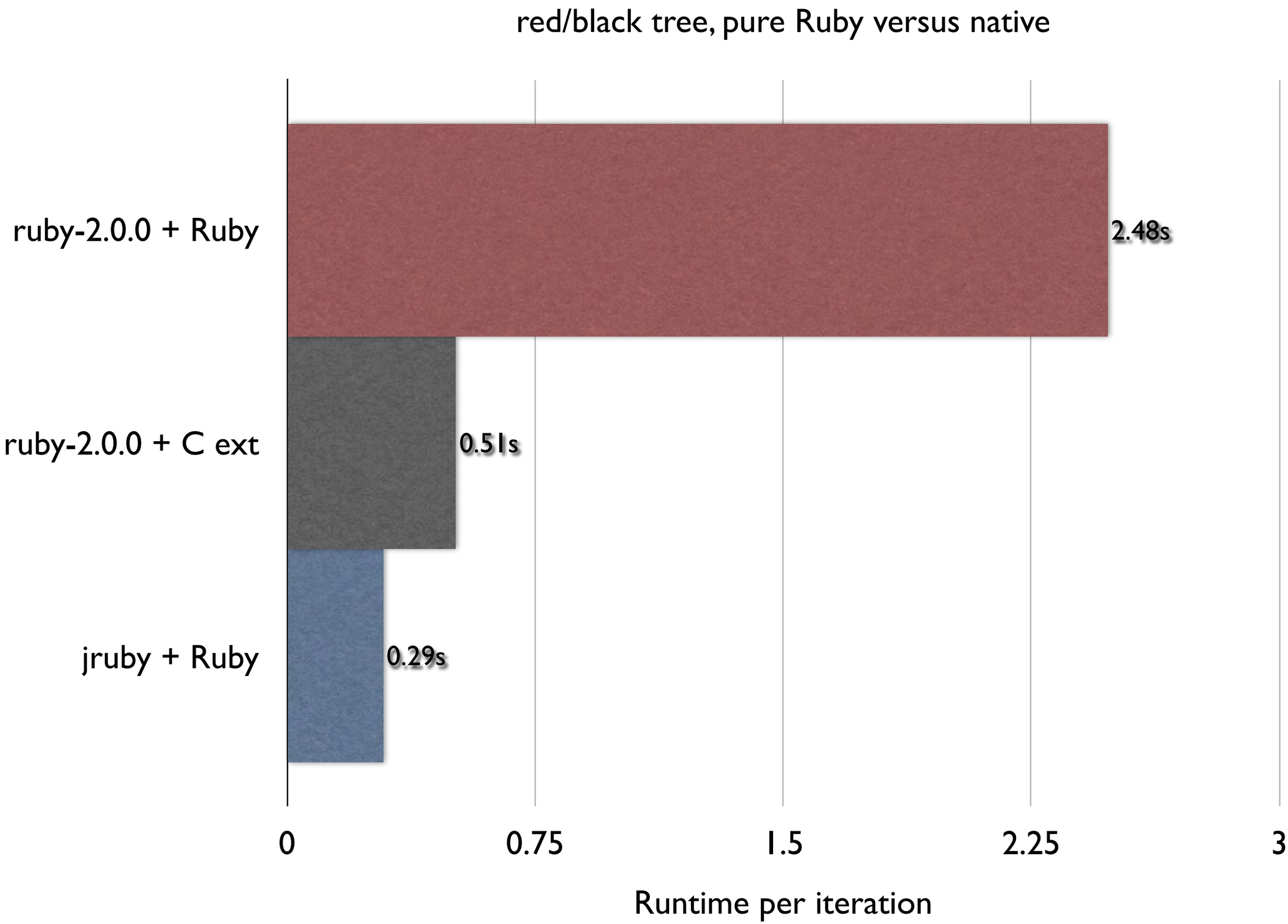


Optimizations (like inlining) can happen!

Indy in JRuby

- Method dispatch
  - Look up method, cache, invalidate if class changes or a new type is seen
- Constant lookup
  - Retrieve constant, cache, invalidate if redefined


JRuby/Java 6
JRuby/Java 7
Times Faster than Ruby 1.9.3
5
3.75
2.5
1.25
0
1.346
2.658
1.538
3.44
1.914
3.66
1.565
4.32
base64
richards
neural
redblack


red/black tree, pure Ruby versus native
ruby-2.0.0 + Ruby
2.48s
ruby-2.0.0 + C ext
0.51s
jruby + Ruby
0.29s
0
0.75
1.5
2.25
3
Runtime per iteration

Nashorn

- Built on indy from day 1
  - Speculative optimization
    - Fall back on fail and reoptimize
    - Performance comparable to V8
- Could become an indy language framework

Lambdas (8)

- Long-awaited closure support for Java
- Similar to inner classes, but…
  - Lazily generate function class
  - Single object constructed, cached
  - All shared state passed on call stack

```
Collections.sort(input, (a,b)->Integer.compare(a.length(), b.length()));
```

```
0: aload_0
1: invokedynamic #36,  0
6: invokestatic  #37          // Comparator.sort:(…)
9: return
```

```
 0: aload_0
 1: invokevirtual #53     // String.length:()I
 4: aload_1
 5: invokevirtual #53     // String.length:()I
 8: invokestatic  #54     // Integer.compare:(II)I
11: ireturn
```

Generic Specialization (9)

- ArrayList<int> that actually uses int
- Indy call site requests int-specialization
- int version of class generated lazily
- Working prototypes now in Valhalla

Ruby == UNIX

Unfortunately, Java == Java everywhere

```
$ ruby -e "p Errno.constants”

[:ENOTRECOVERABLE, :ENAMETOOLONG, :ENOLCK, :ENOSYS, :ENOTE
MPTY, :ELOOP, :FFI, :EWOULDBLOCK, :ENOMSG, :EIDRM, :ENOSTR
, :ENODATA, :ETIME, :ENOSR, :EREMOTE, :ENOLINK, :EPROTO, :
EMULTIHOP, :EBADMSG, :EOVERFLOW, :EILSEQ, :EUSERS, :ENOTSO
CK, :EDESTADDRREQ, :EMSGSIZE, :EPROTOTYPE, :ENOPROTOOPT, :
EPROTONOSUPPORT, :ESOCKTNOSUPPORT, :EOPNOTSUPP, :EPFNOSUPP
ORT, :Mapping, :EPERM, :EAFNOSUPPORT, :EADDRINUSE, :ENOENT
, :EADDRNOTAVAIL, :ESRCH, :ENETDOWN, :EINTR, :ENETUNREACH,
:EIO, :ENETRESET, :ENXIO, :ECONNABORTED, :E2BIG, :ECONNRES
ET, :ENOEXEC, :ENOBUFS, :EBADF, :EISCONN, :ECHILD, :ENOTCO
NN, :EAGAIN, :ESHUTDOWN, :ENOMEM, :ETOOMANYREFS, :EACCES,
:ETIMEDOUT, :EFAULT, :ECONNREFUSED, :ENOTBLK, :EHOSTDOWN,
:EBUSY, :EHOSTUNREACH, :EEXIST, :EALREADY, :EXDEV, :EINPRO
GRESS, :ENODEV, :ESTALE, :ENOTDIR, :EDQUOT, :EISDIR, :EBAD
RPC, :EINVAL, :ERPCMISMATCH, :ENFILE, :EPROGUNAVAIL, :EMFI
LE, :EPROGMISMATCH, :ENOTTY, :EPROCUNAVAIL, :ETXTBSY, :EFT
YPE, :EFBIG, :EAUTH, :ENOSPC, :ENEEDAUTH, :ESPIPE, :ECANCE
```

```
$ ruby -retc -e "p Etc.public_methods.sort -
Module.public_methods”

[:endgrent, :endpwent, :getgrent, :getgrgid,
:getgrnam, :getlogin, :getpwent, :getpwnam, :
getpwuid, :group, :initialize, :passwd, :setg
rent, :setpwent]
```

Native Interop

JVM World

????

Native World

"If non-Java programmers find some library useful and easy to access, it should be similarly accessible to Java programmers."

- John Rose

NAME

getpid, getppid - get process identification

SYNOPSIS

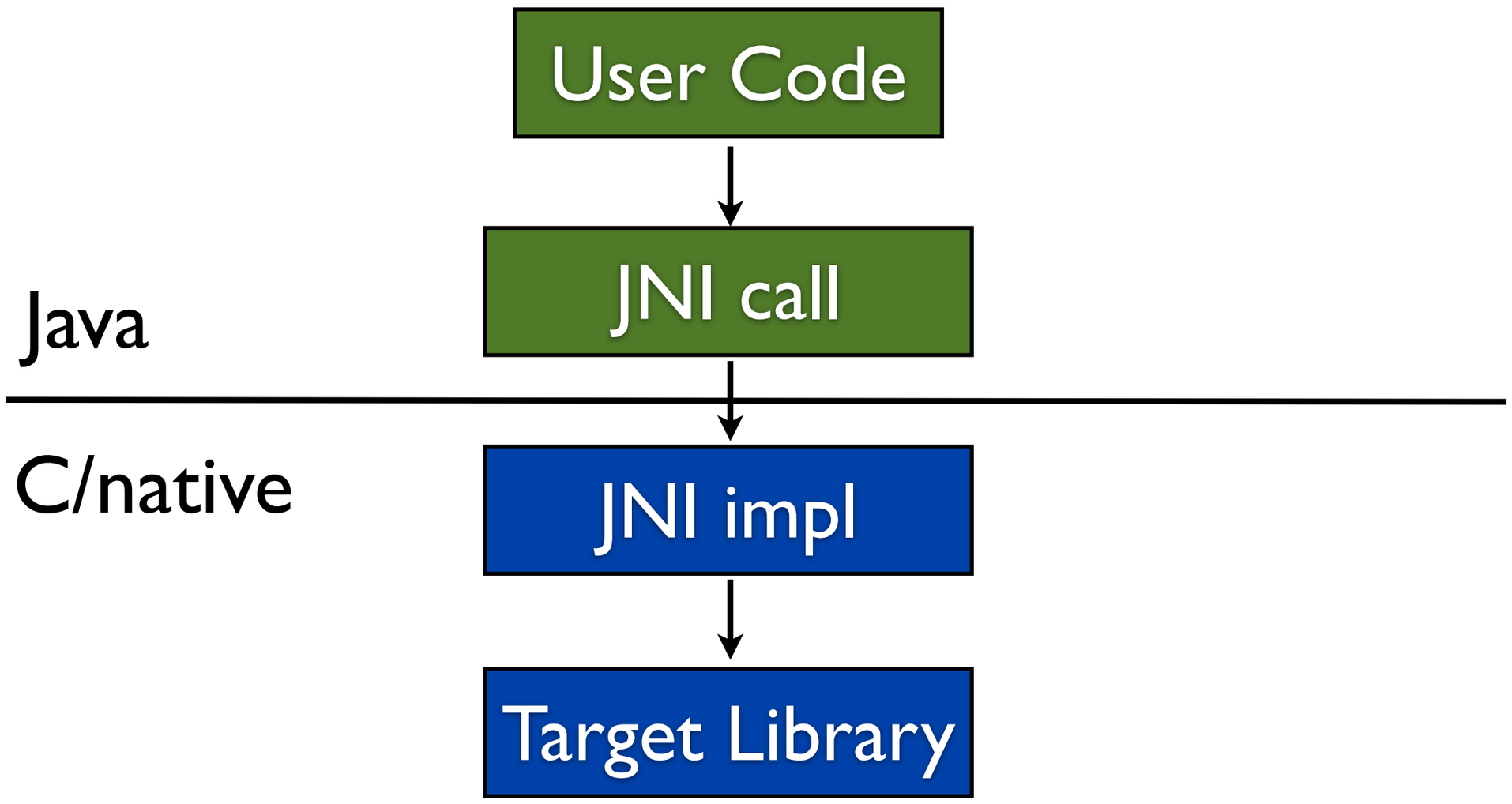
```
#include <sys/types.h>
#include <unistd.h>

pid_t getpid(void);
pid_t getppid(void);
```

DESCRIPTION

getpid() returns the process ID of the current process. (This is often used by routines that generate unique temporary filenames.)

getppid() returns the process ID of the parent of the current process.

JNI


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 ) {
        getpid();
    }

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

JNI

```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
#endif
```

JNI

```
#include "com_headius_jnr_presentation_GetPidJNI.h"

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

  return getpid();
}
```

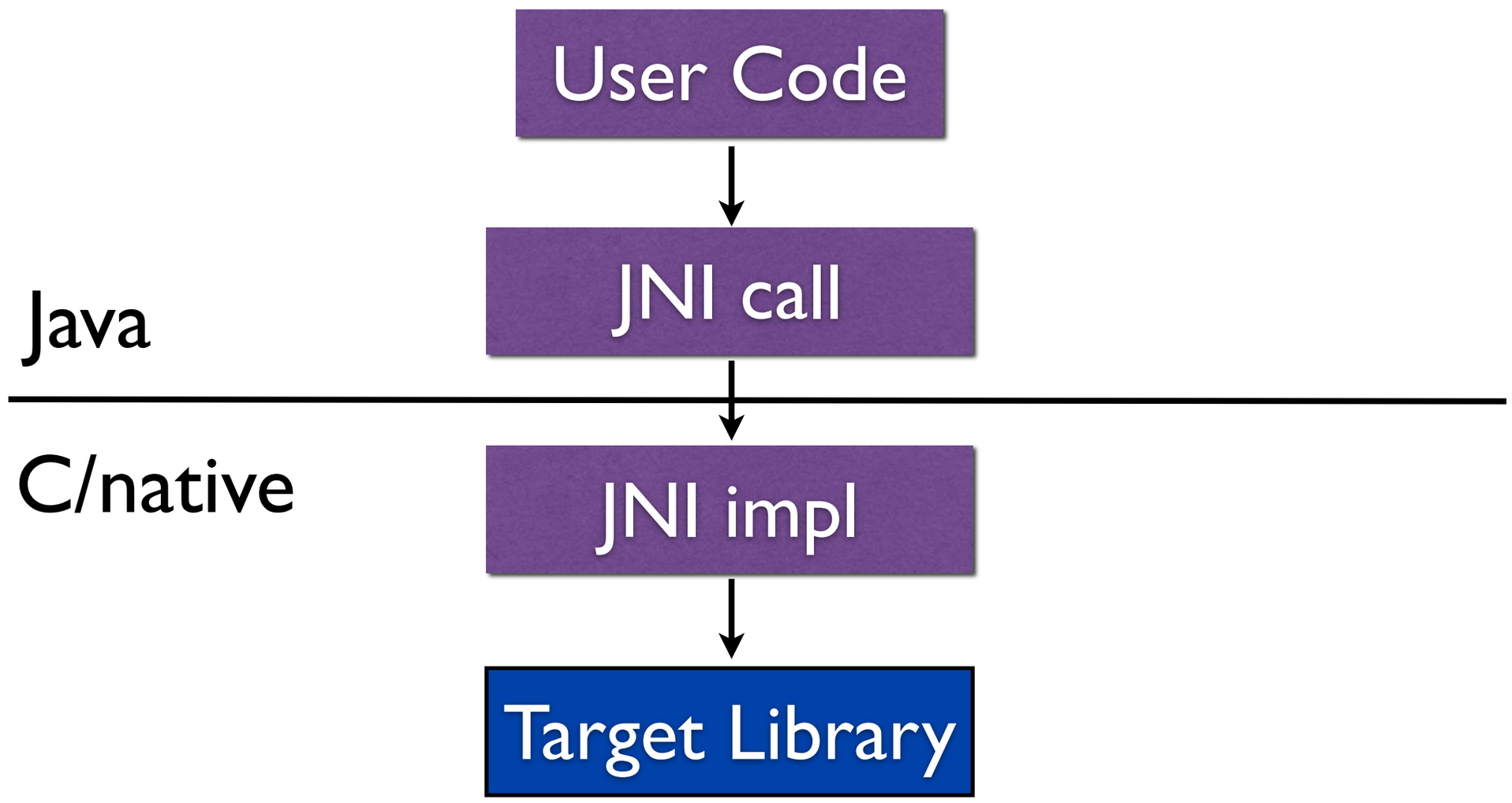
JNI

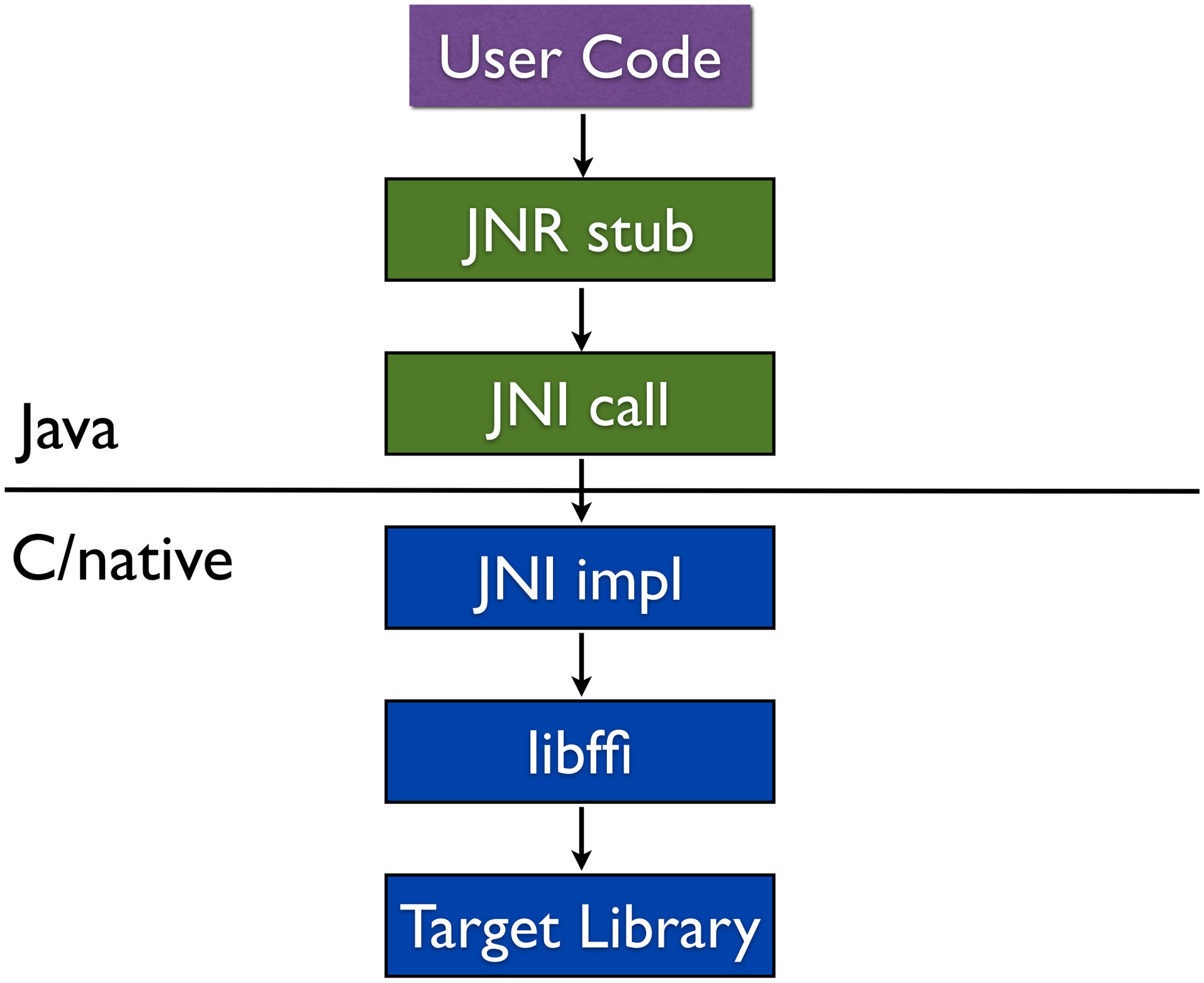
```
$ gcc -I $JAVA_HOME/include -I
$JAVA_HOME/include/darwin -L
$JAVA_HOME/jre/lib/ -dynamiclib -ljava
-o getpidjni.dylib
com_headius_jnr_presentation_GetPidJNI.
c

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

There Must Be A Better Way

Java Native Runtime

- **Java** API
- for calling **Native** code
- supported by a rich **Runtime** library
- You may be familiar with **JNA**
- **F**oreign **F**unction **I**nterface (FFI)
- https://github.com/jnr

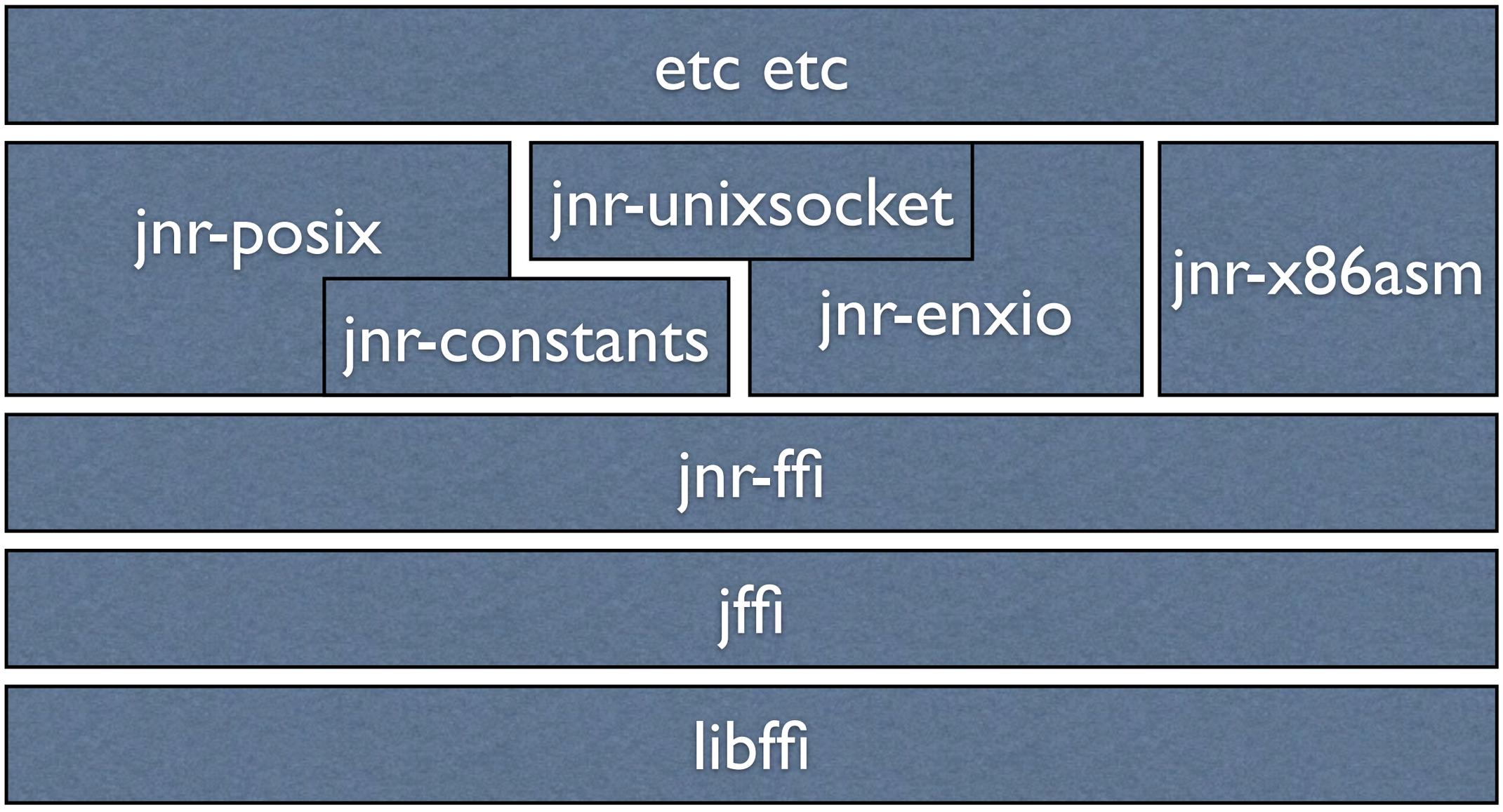

User Code
JNI call
Java
C/native
JNI impl
Target Library


User Code
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



Platforms

- Darwin (OS X): universal (+ppc?)
- Linux: i386, x86_64, arm, ppc, ppc64, ppc64le, s390x
- Windows: i386, x86_64
- FreeBSD, OpenBSD: i386, x86_64
- SunOS: i386, x86_64, sparc, sparcv9
- AIX: ppc
- OpenVMS, AS/400: builds out there somewhere
- If your platform isn't here, contribute a build

jnr-ffi

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

jnr-ffi

```
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();
    }
}
```

jnr-posix

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

```
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);
```

```
POSIX posix = POSIXFactory.getPOSIX(
                new MyPOSIXHandler(this),
                isNativeEnabled);

int pid = posix.getpid();
```

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

```
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;
}
```

jnr-unixsocket

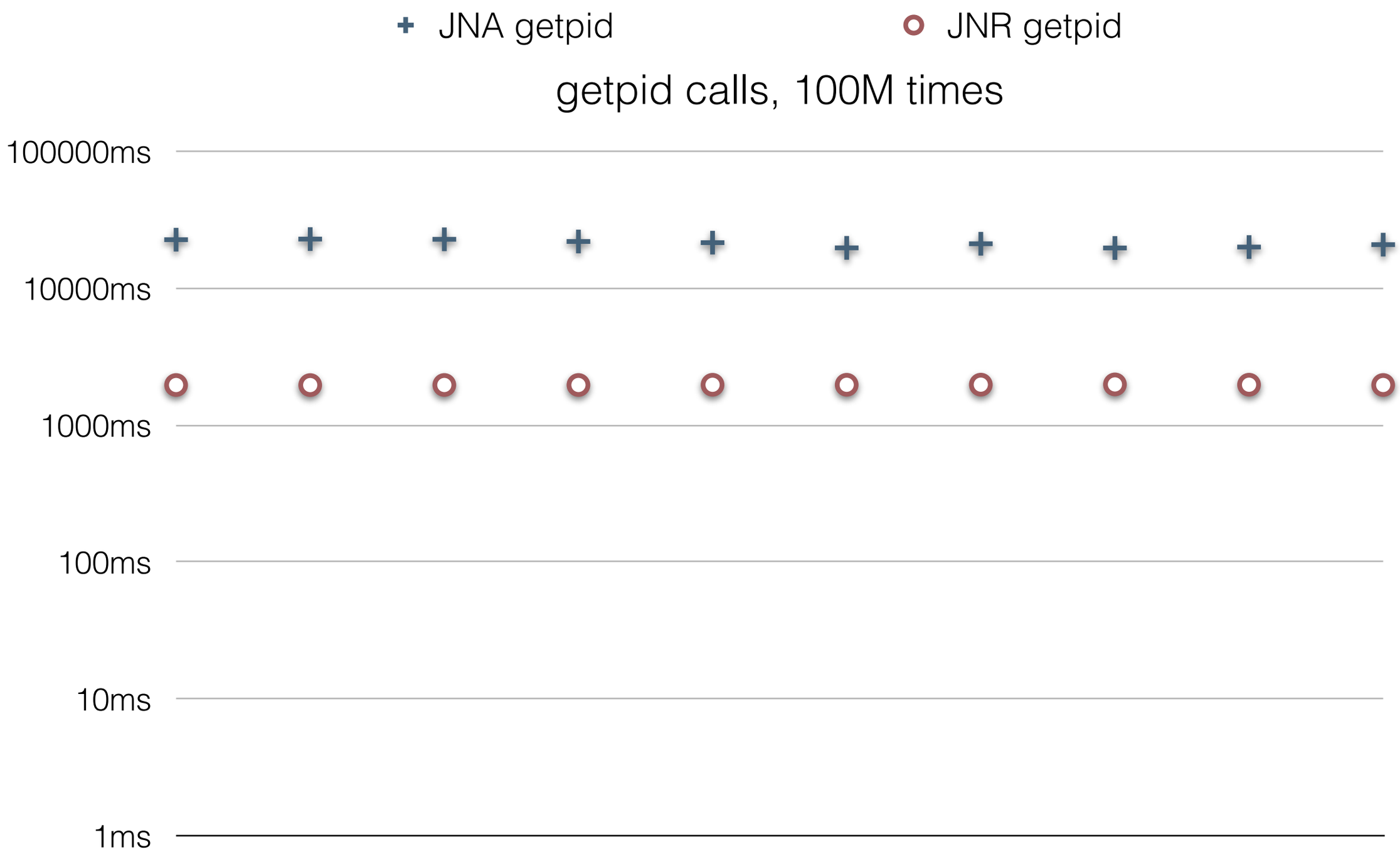
- UNIX sockets for NIO
- Built atop jnr-enxio
- Fully selectable, etc
- https://github.com/jnr/jnr-unixsocket

What Else?

- 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, ...)

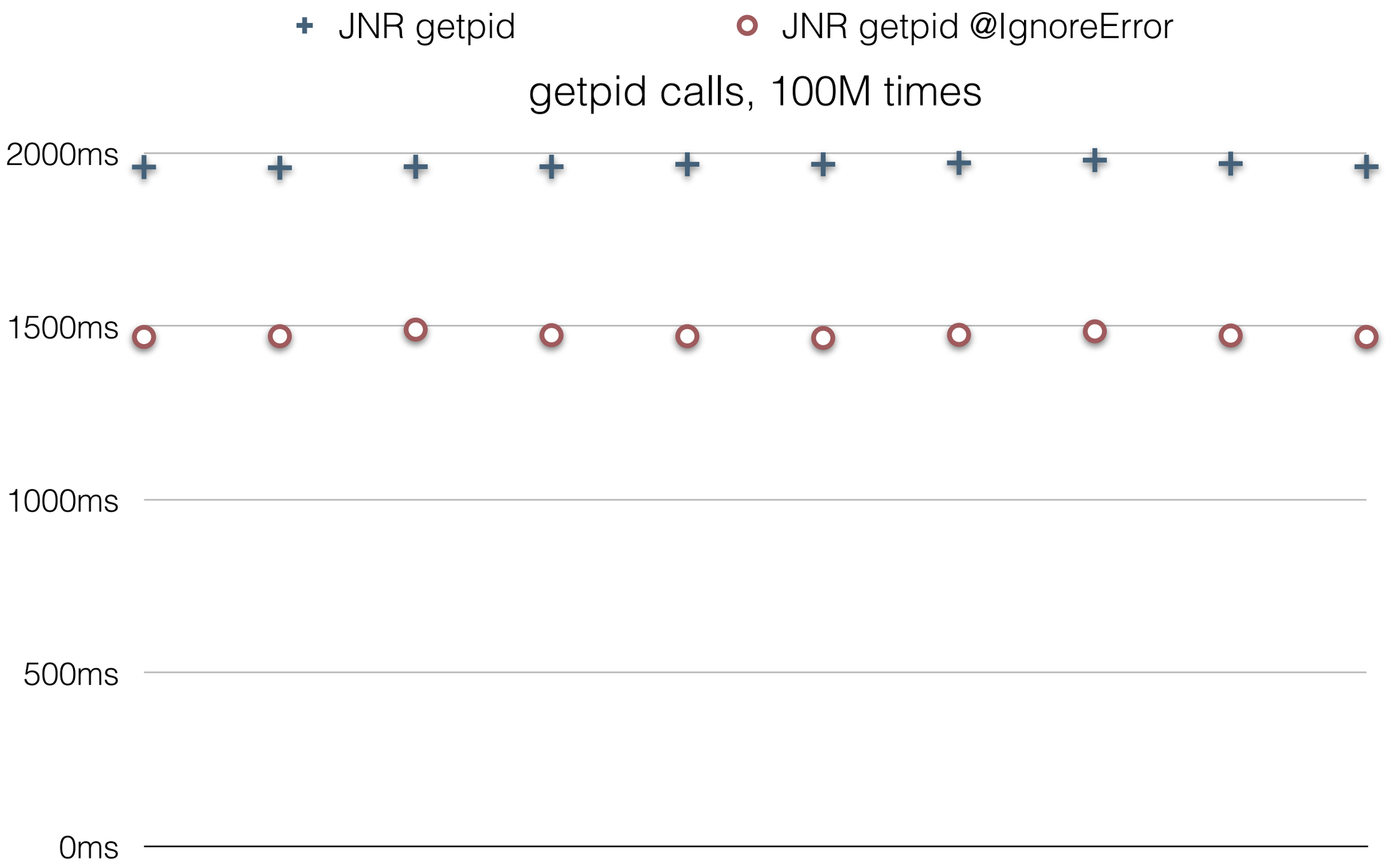
Performance

- Generated code leading to JNI call
- Generated assembly version of native part
- jnr-x86asm: Generate and link ASM
- Used internally by jnr
- https://github.com/jnr/jnr-x86asm

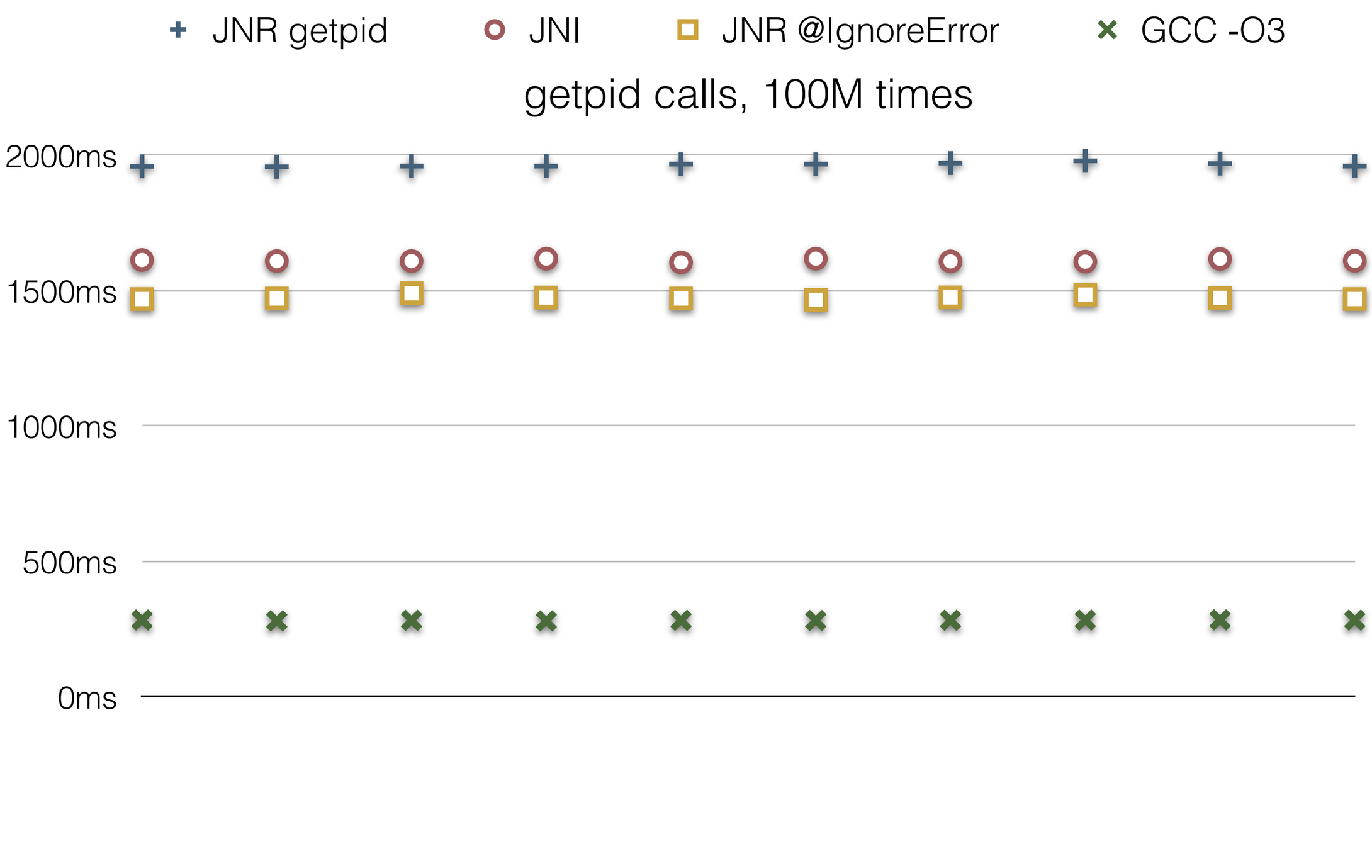

JNA getpid
JNR getpid
getpid calls, 100M times
100000ms
10000ms
1000ms
100ms
10ms
1ms

@IgnoreError

```
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



Project Panama

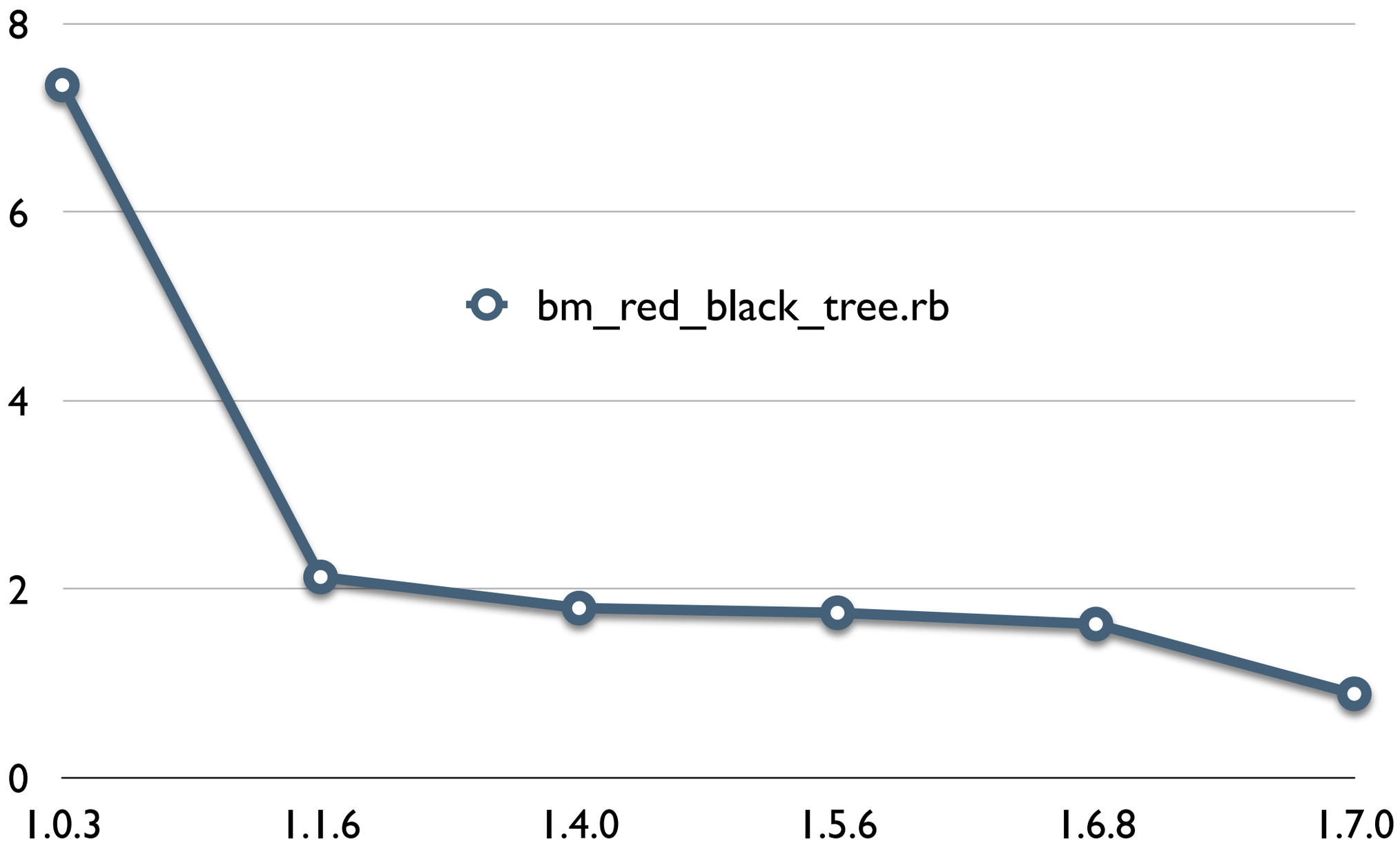
- JEP-191: FFI for the JVM
  - Native interop at platform level
    - JIT intelligence
    - Security
- Native method handles
- Native memory layout and manipulation

Ruby == Objects

Lots and lots and lots…

Allocation is the root of all evil

JRuby Over Time



-J-verbose:gc

8.1s

[GC (Allocation Failure) 223608K->96408K(330752K), 0.0159780 secs]
[GC (Allocation Failure) 208920K->100792K(335168K), 0.0157550 secs]
[GC (Allocation Failure) 213304K->105144K(332160K), 0.0181010 secs]
[GC (Allocation Failure) 205112K->108920K(334400K), 0.0187580 secs]
[GC (Allocation Failure) 208888K->112712K(329152K), 0.0154440 secs]
+28 more

8.4s

[GC (Allocation Failure) 313780K->199892K(339072K), 0.0142010 secs]
[GC (Allocation Failure) 318420K->204420K(331520K), 0.0175690 secs]
[GC (Allocation Failure) 306948K->208316K(335680K), 0.0188120 secs]
[Full GC (Ergonomics) 208316K->54991K(352256K), 0.2709750 secs]
[GC (Allocation Failure) 157519K->58959K(349248K), 0.0120840 secs]
+28 more

Around 1.8GB/s

-J-verbose:gc

[GC (Allocation Failure) 155729K->39697K(207296K), 0.0072730 secs]

0.963s

[GC (Allocation Failure) 160785K->40657K(208320K), 0.0108620 secs]

0.968s

[GC (Allocation Failure) 161745K->41649K(210112K), 0.0083760 secs]

0.968s

[GC (Allocation Failure) 166193K->39729K(210688K), 0.0070670 secs]

0.99s

Hard to Fix

- Closures/blocks
  - Need to put local vars in heap structure
- Numerics
  - All call paths require references
- Transient data structures
  - [foo,bar,baz].map(&:to_s).sort.first

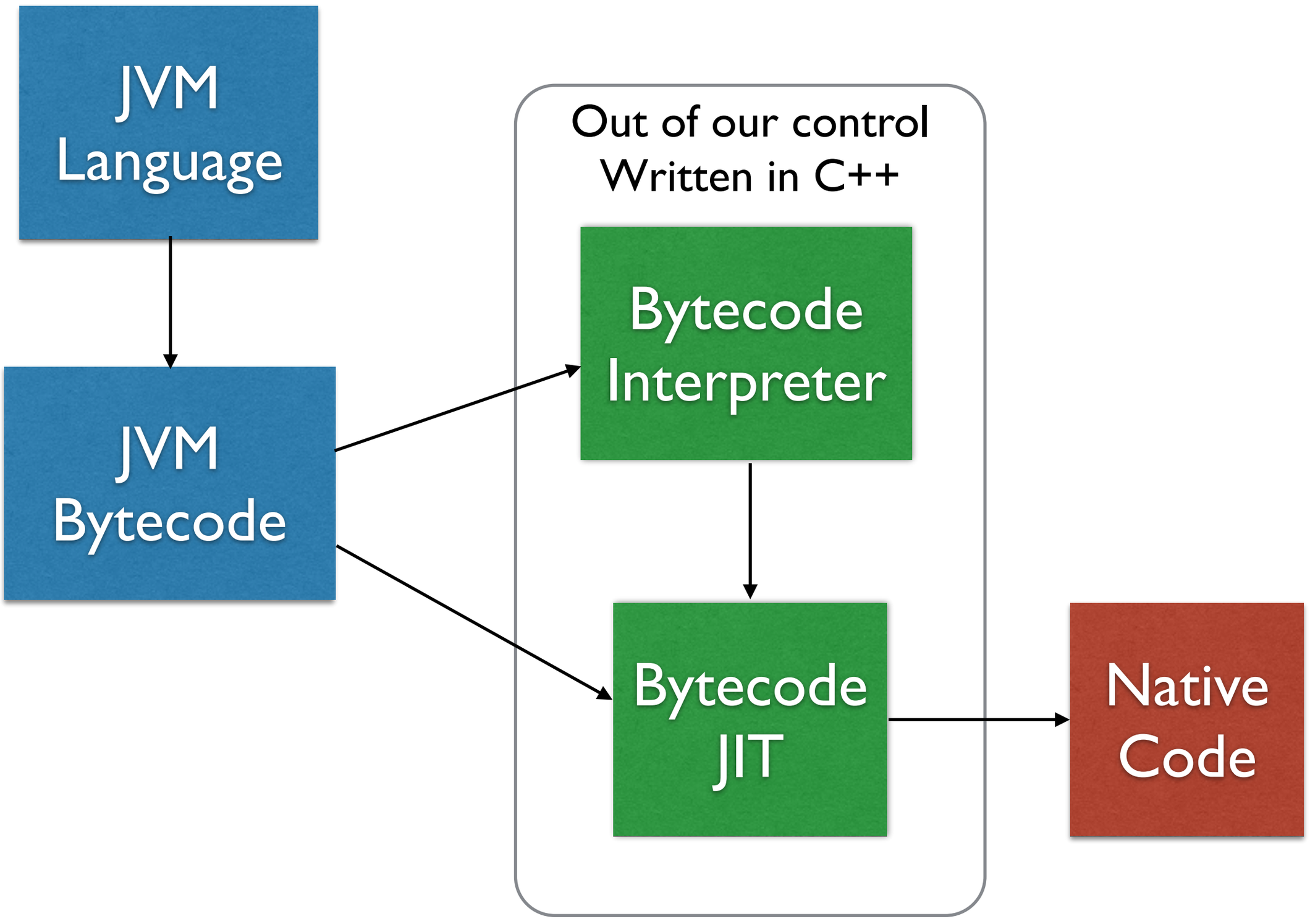
Hard to Fix

- Closures/blocks
  - Need to put local vars in heap structure
- Numerics
  - All call paths require references
- Transient data structures
  - [foo,bar,baz].lazy.map(&:to_s).sort.first

Help Me, JVM

- OpenJDK/Hotspot escape analysis
  - If an object never leaves a piece of code, and is never visible across threads, modify the code to just use the object contents.
- Fails to work in 99% of our cases
  - Improvements on the way, we hope

Truffle and Graal

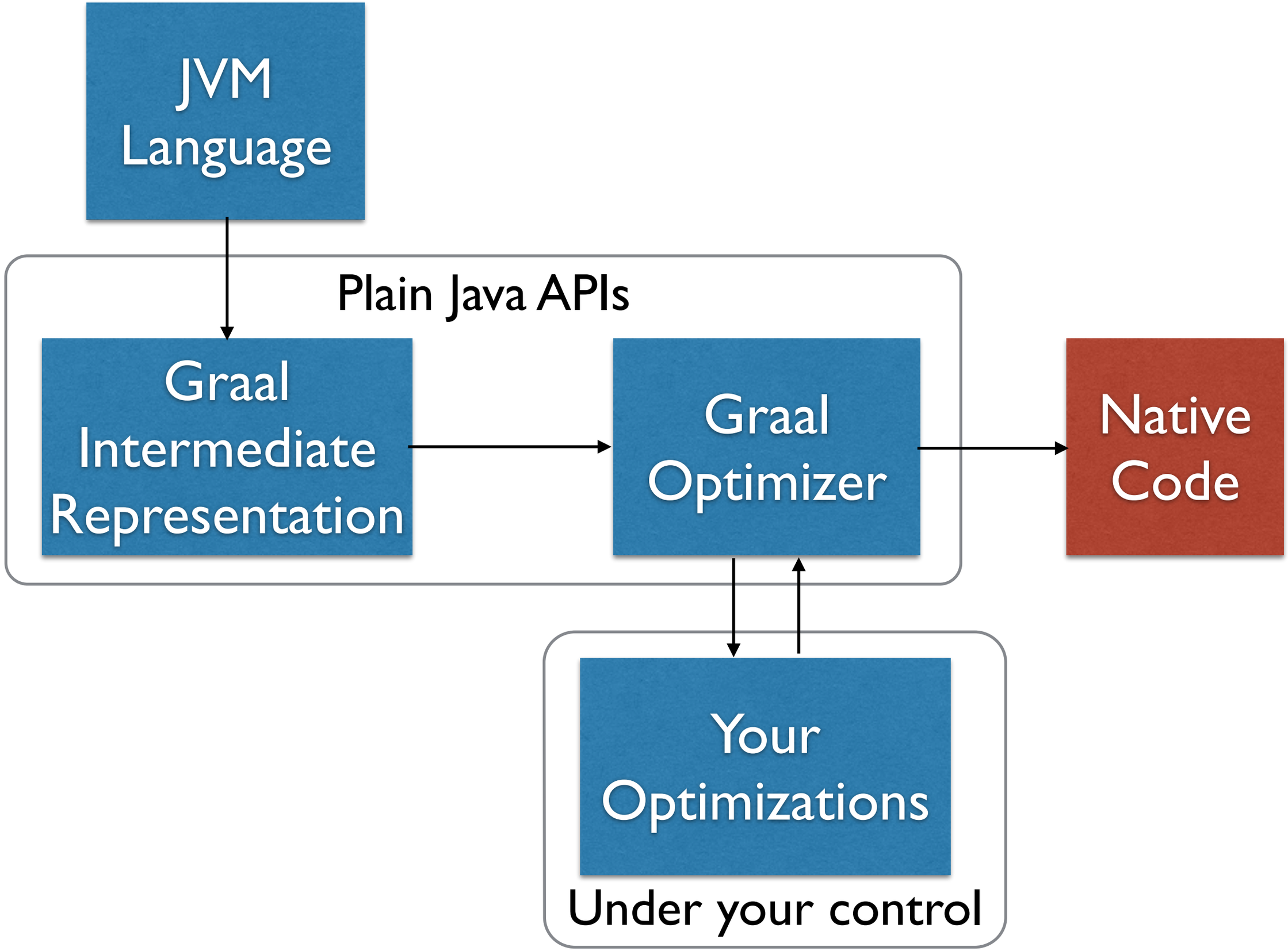

JVM Language
JVM Bytecode
Out of our control
Written in C++
Bytecode Interpreter
Bytecode JIT
Native Code

What If…

- The JVM's JIT optimizer were written in Java
- You could customize how the JIT works for your language or library
- JITed code could directly make native calls

Graal

- A 100% Java-based JIT framework
  - Grew out of the 100% Java "Maxine" JVM
  - Emits assembly or HotSpot IR
  - Directly control code generation
  - Route around JVM bytecode
  - http://openjdk.java.net/projects/graal/


JVM Language
Plain Java APIs
Graal Intermediate Representation
Graal Optimizer
Native Code
Your Optimizations
Under your control

However…

- Not everyone is a compiler writer
- Graal's IR is low-level and nontrivial
- Need to understand JVM internals
- Need some understanding of CPU

The Dream

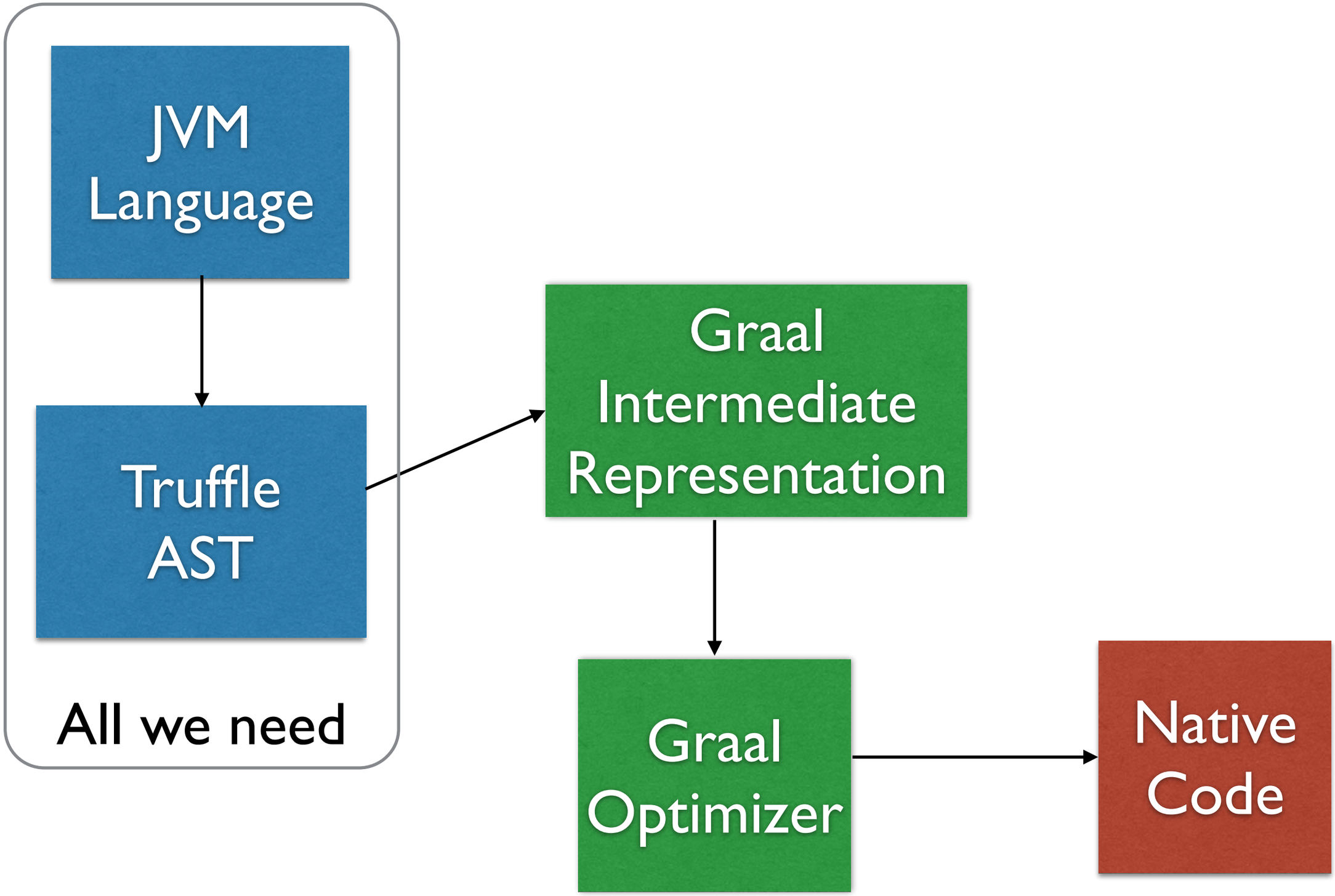
- Design your language
- ???
- PROFIT

What We Want

- Design your language
- Write an interpreter
- PROFIT

Truffle

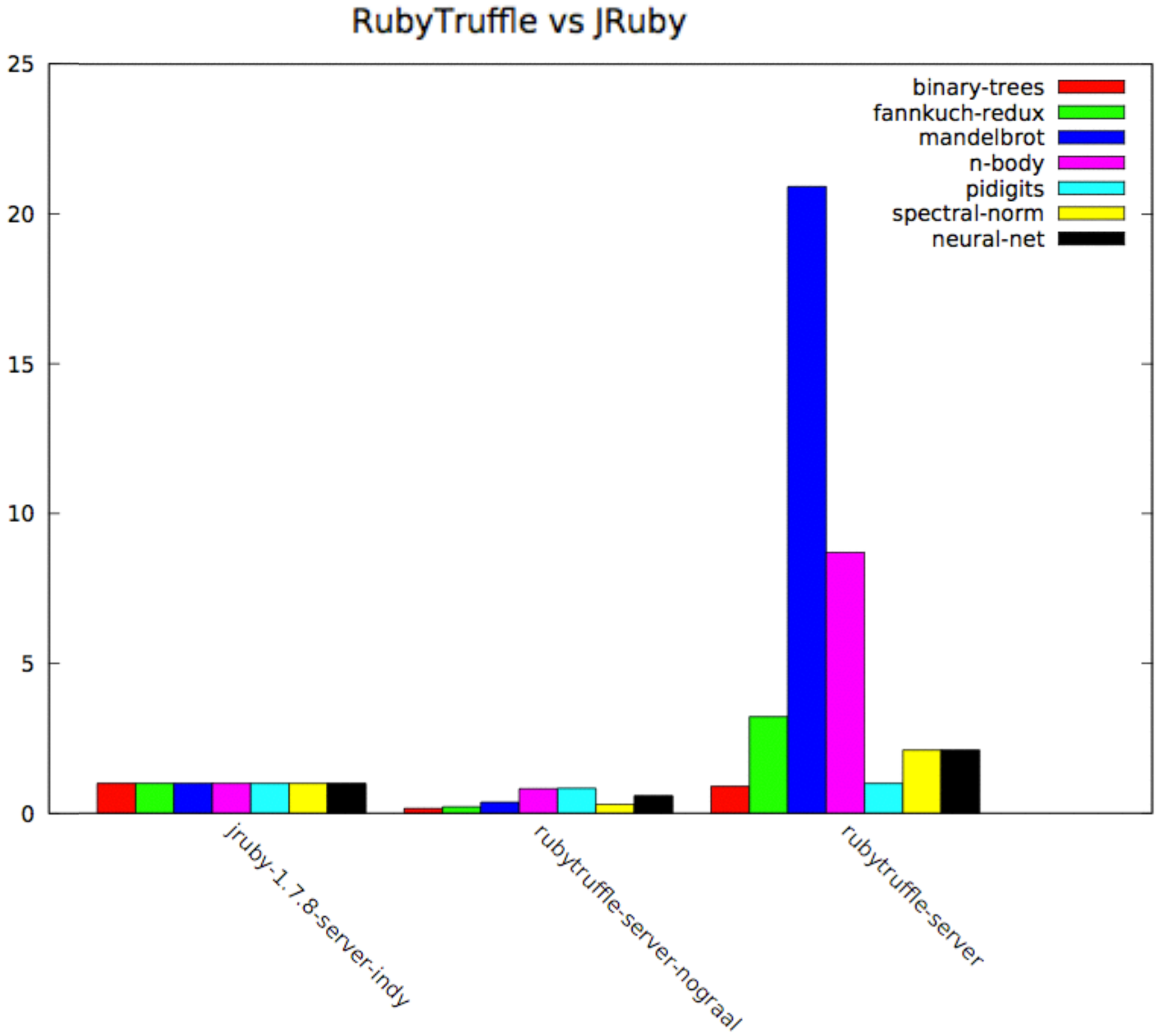
- Language framework built on Graal
- Designed to fulfill the dream
  - Implement interpreter
  - Truffle feeds that to backend
  - No compiler expertise needed
- https://wiki.openjdk.java.net/display/Graal/Truffle+FAQ+and+Guidelines


JVM
Language
Truffle
AST
All we need
Graal
Intermediate
Representation
Graal
Optimizer
Native
Code

Chris Seaton
@ChrisGSeaton

Follow

JVM languages will never be the same: RubyTruffle faster than @JRuby and @TopazProject; as complete as Topaz; simpler than either. #JVMLS


RubyTruffle vs JRuby
25
20
15
10
5
0
binary-trees
fannkuch-redux
mandelbrot
n-body
pidigits
spectral-norm
neural-net
jruby-1.7.8-server-indy
rubytruffle-server-nograal
rubytruffle-server

The Payoff

- RubyTruffle now part of JRuby
  - Released in JRuby 9k
  - Many languages in the works
  - May become part of OpenJDK releases?
  - Forcing Hotspot folks to push harder

What Have We Learned?

The JVM has its problems, but we can fix them.

This is all open source…you can help

Nothing is impossible.

Thank you!

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