Foreign Function & Memory API (JEP 454)
Project Panama

Inner Workings of the FFI API in the JVM

FOSDEM'25 - **FREE JAVA DEVROOM**

MARTIN DOERR

Java and native code

native library:

```
time.so
```

with function:

```
void get_time(char* timeString, int len);
```

JNI ("Java Native Interface") C layer

```c
typedef void (*get_time_func_type)(char* timeString, int len);
get_time_func_type get_time_func = NULL;

JNIEXPORT jstring JNICALL Java_TestJNI_getTime(JNIEnv *env, jclass cls) {
  if (get_time_func == NULL) {
    void* dll_handle = dlopen("./time.so", RTLD_LAZY);
    if (dll_handle == NULL) {
      (*env)->ThrowNew(env, (*env)->FindClass(env, "java/lang/Exception"), "Couldn't open time.so.");
      return NULL;
    }
    get_time_func = (get_time_func_type) dlsym(dll_handle, "get_time");
    if (get_time_func == NULL) {
      (*env)->ThrowNew(env, (*env)->FindClass(env, "java/lang/Exception"), "Couldn't find get_time in time.so.");
      return NULL;
    }
  }

  char buf[100];
  get_time_func(buf, sizeof(buf));
  return (*env)->NewStringUTF(env, buf);
}
```

JNI ("Java Native Interface") Java layer

```
public class TestJNI {
    static {
        final String dir = System.getProperty("user.dir");
        System.load(dir + "/JNI.so");
    }

    public static native String getTime();

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

Java FFI ("Foreign Function Interface") Downcall

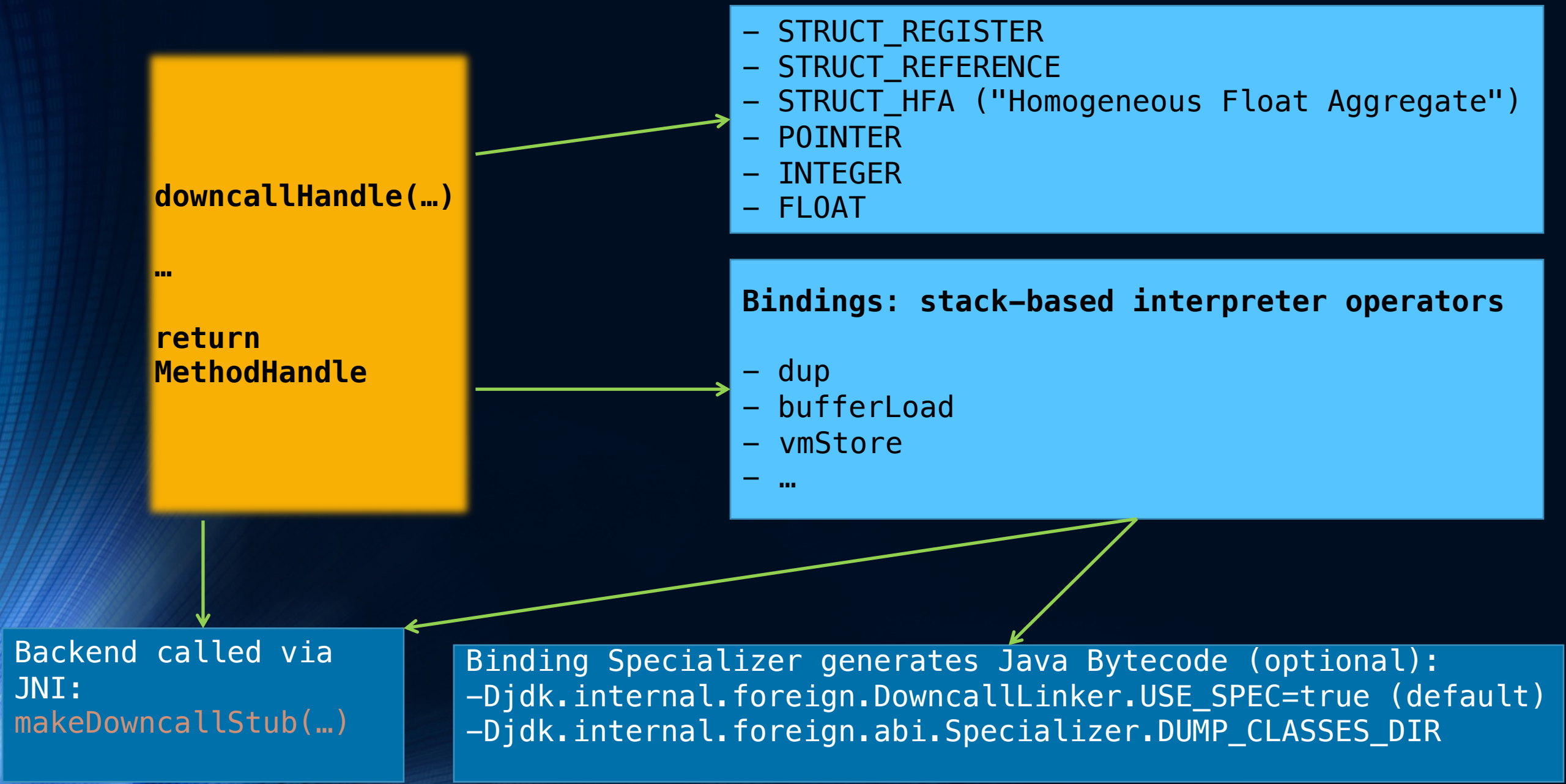
```
public class TestFFI {
    static {
        final String dir = System.getProperty("user.dir");
        System.load(dir + "/time.so");
    }

    static final int array_size = 100;
    static final SequenceLayout array_layout = MemoryLayout.sequenceLayout(array_size, JAVA_BYTE);
    static final FunctionDescriptor get_time_fd = FunctionDescriptor.ofVoid(ADDRESS, JAVA_INT);
    static final MemorySegment get_time_sym = SymbolLookup.loaderLookup().find("get_time").orElseThrow();
    static final MethodHandle get_time_mh = Linker.nativeLinker().downcallHandle(get_time_sym, get_time_fd);

    public static void main(String args[]) throws Throwable {
        try (Arena arena = Arena.ofConfined()) {
            MemorySegment s = arena.allocate(array_layout);
            get_time_mh.invokeExact(s, array_size);
            System.out.println(s.getString(0));
        }
    }
}
```

Linker.nativeLinker()

CPU	*OS*
x86_64	UNIX (Linux / MacOS)
	Windows
aarch64	Linux
	MacOS
	Windows
ppc64	Linux little endian (ABIv2)
	Linux big endian
	AIX
riscv	Linux
s390	Linux
others	supported by libffi

Created by Copilot (AI)

Linker.nativeLinker().downcallHandle(...)



The backend

can be traced with debug build:

```
java --enable-native-access=ALL-UNNAMED -Xlog:foreign+downcall=trace TestFFI
```

```
Decoding CodeBlob, name: nep_invoker_blob, at  [0x00007f989c589ae0, 0x00007f989c589bc8]  232 bytes
  0x00007f989c589ae0:   push   %rbp                <= Stack frame
  0x00007f989c589ae1:   mov    %rsp,%rbp
 ;; { thread java2native                          <= Change thread state to "in native"
  0x00007f989c589ae4:   vzeroupper
  0x00007f989c589ae7:   mov    %rbp,0x3f0(%r15)
  0x00007f989c589aee:   movabs $0x7f989c589ae4,%r10
  0x00007f989c589af8:   mov    %r10,0x3e8(%r15)
  0x00007f989c589aff:   mov    %rsp,0x3e0(%r15)
  0x00007f989c589b06:   movl   $0x4,0x48c(%r15)
 ;; } thread java2native
 ;; { argument shuffle                            <= Setup arguments registers
  0x00007f989c589b11:   mov    %rsi,%r10
  0x00007f989c589b14:   mov    %rcx,%rsi
  0x00007f989c589b17:   mov    %rdx,%rdi
 ;; } argument shuffle
  0x00007f989c589b1a:   callq  *%r10               <= Call the native function
 ;; { thread native2java                          <= Change thread state to "in Java"
  0x00007f989c589b1d:   vzeroupper
  0x00007f989c589b20:   movl   $0x5,0x48c(%r15)
  0x00007f989c589b2b:   lock addl $0x0,-0x40(%rsp)  <= Memory barrier (not with -XX:+UseSystemMemoryBarrier)
  0x00007f989c589b31:   cmp    0x28(%r15),%rbp
  0x00007f989c589b35:   ja     0x00007f989c589b88
  0x00007f989c589b3b:   cmpl   $0x0,0x488(%r15)
  0x00007f989c589b43:   jne    0x00007f989c589b88
  0x00007f989c589b49:   movl   $0x8,0x48c(%r15)
 ;; reguard stack check
  0x00007f989c589b54:   cmpl   $0x2,0x520(%r15)
...
 ;; } thread native2java
  0x00007f989c589b86:   leaveq                     <= Remove frame and return
  0x00007f989c589b87:   retq
```

Java critical version - Warning: Use with care!

```
    static final MethodHandle get_time_mh = Linker.nativeLinker().downcallHandle(get_time_sym, get_time_fd,
        Linker.Option.critical(true));

    public static void main(String args[]) throws Throwable {
        byte[] array = new byte[array_size];
        get_time_mh.invokeExact(MemorySegment.ofArray(array), array_size);
        System.out.println(new String(array));
    }
```

Warning: Do not use for anything which may block!

Native Entry Point invoker blob as critical version

```
Decoding CodeBlob, name: nep_invoker_blob, at  [0x00007f772486c6e0, 0x00007f772486c6f8]  24 bytes
  0x00007f772486c6e0:   push   %rbp                    <= Stack frame
  0x00007f772486c6e1:   mov    %rsp,%rbp
  0x00007f772486c6e4:   add    %rcx,%rdx               <= Compute array start address
 ;; { argument shuffle                                <= Setup arguments registers
  0x00007f772486c6e7:   mov    %rsi,%r10
  0x00007f772486c6ea:   mov    %r8,%rsi
  0x00007f772486c6ed:   mov    %rdx,%rdi
 ;; } argument shuffle
  0x00007f772486c6f0:   callq  *%r10                   <= Call the native function
  0x00007f772486c6f3:   leaveq                         <= Remove frame and return
  0x00007f772486c6f4:   retq
```

Capture Call State

```
static final MethodHandle get_time_mh = Linker.nativeLinker().downcallHandle(get_time_sym, get_time_fd,
    Linker.Option.captureCallState("errno"));
static final VarHandle errnoHandle = Linker.Option.captureStateLayout().varHandle(
    MemoryLayout.PathElement.groupElement("errno"));

public static void main(String args[]) throws Throwable {
    try (Arena arena = Arena.ofConfined()) {
        MemorySegment s = arena.allocate(array_layout);
        MemorySegment capturedState = arena.allocate(Linker.Option.captureStateLayout());
        get_time_mh.invokeExact(capturedState, s, array_size);
        System.out.println(s.getString(0) + " errno: " + (int) errnoHandle.get(capturedState, 0L));
    }
}
```

Native Entry Point invoker blob: Capture Call State

```
 0x00007f288458a0a2:   callq  *%r10
;; { save thread local
 0x00007f288458a0a5:   mov    (%rsp),%rdi
 0x00007f288458a0a9:   mov    $0x4,%esi
 0x00007f288458a0ae:   vzeroupper
 0x00007f288458a0b1:   mov    %rsp,%r12
 0x00007f288458a0b4:   sub    $0x0,%rsp
 0x00007f288458a0b8:   and    $0xfffffffffffffff0,%rsp
 0x00007f288458a0bc:   callq  0x00007f2893ae0eb0 = DowncallLinker::capture_state(int*, int)
 0x00007f288458a0c1:   mov    %r12,%rsp
 0x00007f288458a0c4:   xor    %r12,%r12
;; } save thread local
```

Passing structures

```
struct S_IDF { int p0; double p1; float p2; };
```

```
static final StructLayout S_IDFLayout_with_padding= MemoryLayout.structLayout(
    C_INT.withName("p0"),
    MemoryLayout.paddingLayout(4), // AIX: only with #pragma align (natural)
    C_DOUBLE.withName("p1"),
    C_FLOAT.withName("p2"),
    MemoryLayout.paddingLayout(4)
).withName("S_IDF");
```

Upcall: native to Java

```
MemorySegment stub = Linker.nativeLinker().upcallStub(
    MethodHandles.lookup().findStatic(Test4BAlignedDouble.class, "S_IDF_fun", mt),
    fdpass_S_IDF, arena);
s = (MemorySegment) mhpass_S_IDF_fun.invokeExact((SegmentAllocator) arena, stub, s);
```

Ways to call native functions

- **JNI**
  - Critical functions: (available until JDK17, but deprecated) `-XX:+CriticalJNINatives`
- **FFI**
  - Critical functions: available since JDK22 as Linker Option
- Project **nalim**: https://github.com/apangin/nalim
  - Only for critical functions
  - Uses JVMCI ("Java Virtual Machine Compiler Interface")
  - Use at your own risk!

PPC64 porting challenges

- int passed as long
- float uses double format when passed in register
- Very complicated HFA ("Homogenous Float Aggregate")
- 3 different ABIs (Linux LE, Linux BE, AIX)
- Big endian platforms need to shift small values in registers

Panama can be used free of charge!

https://openjdk.org/projects/panama/

Martin

TheRealMDoerr

Follow

JVM developer at SAP. OpenJDK Committer/Reviewer, PowerPC/AIX Port Project Lead and JCP EC member.

https://sapmachine.io

Download