Experiences building a JVM with LLVM ORC JIT

Markus Böck
University of Cambridge[1]

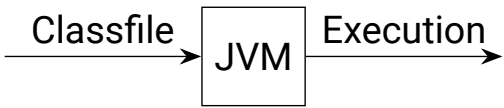
Marton Karolyi
Technical University of Vienna

Thomas Mayerl
ETH Zürich[1]

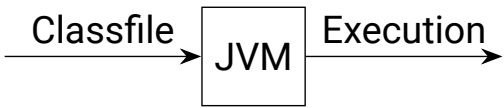
[1] Work performed while at Technical University of Vienna

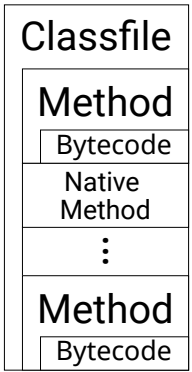
JVM Basics

JVM Basics



JVM Basics


Classfile
JVM
Execution

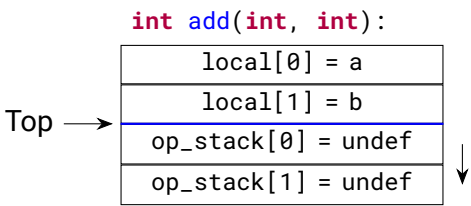

Classfile
Method
Bytecode
Native Method
Method
Bytecode

JVM Bytecode Basics

```
static int add(int a, int b) {
  return a + b;
}
```

→

```
add:(II)I
  iload_0
  iload_1
  iadd
  ireturn
```

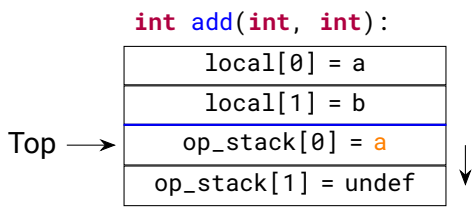


JVM Bytecode Basics

```
static int add(int a, int b) {
  return a + b;
}
```

→

```
add:(II)I
  iload_0
  iload_1
  iadd
  ireturn
```

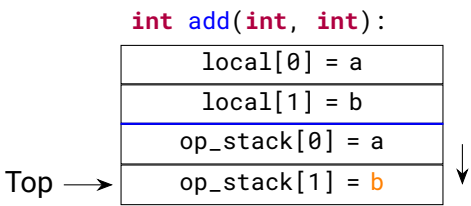


JVM Bytecode Basics

```
static int add(int a, int b) {
  return a + b;
}
```

→

```
add:(II)I
  iload_0
  iload_1
  iadd
  ireturn
```

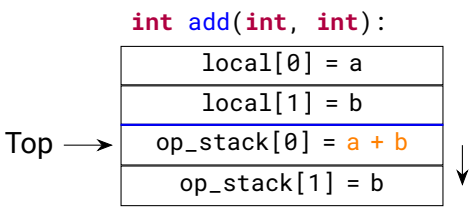


JVM Bytecode Basics

```
static int add(int a, int b) {
  return a + b;
}
```

→

```
add:(II)I
  iload_0
  iload_1
  iadd
  ireturn
```

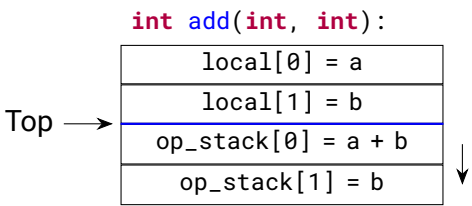


JVM Bytecode Basics

```
static int add(int a, int b) {
  return a + b;
}
```

→

```
add:(II)I
  iload_0
  iload_1
  iadd
  ireturn
```



JVM Bytecode Restrictions

```
foo:(I)I
  iconst_0
  iload_0
  ifeq skip
  fconst_1
skip:
  ireturn
```

JVM Bytecode Restrictions

```
foo:(I)I
  iconst_0
  iload_0
  ifeq skip
  fconst_1
skip:
  ireturn
```

🚫 Flow dependent stack 🚫

JVM Bytecode Restrictions

```
foo:(I)I
  iconst_0
  iload_0
  ifeq skip
  fconst_1
skip:
  ireturn
```

🚫 Flow dependent stack 🚫
⇒ Statically computable
“Top” and stack types

JVM Bytecode Restrictions

```
foo:(I)I
  iconst_0
  iload_0
  ifeq skip
  fconst_1
skip:
  ireturn
```

🚫 Flow dependent stack 🚫

⇒ Statically computable “Top” and stack types

```
foo:(I)I
  iload_0
  ifeq skip
  fconst_1
  fstore_0
skip:
  iload_0
  ireturn
```

JVM Bytecode Restrictions

```
foo:(I)I
  iconst_0
  iload_0
  ifeq skip
  fconst_1
skip:
  ireturn
```

🚫 Flow dependent stack 🚫
⇒ Statically computable
“Top” and stack types

```
foo:(I)I
  iload_0
  ifeq skip
  fconst_1
  fstore_0
skip:
  iload_0
  ireturn
```

🚫 Flow dependent
local variable type 🚫

JVM Bytecode Compilation

```
add:(II)I
  iload_0
  iload_1
  iadd
  ireturn
```

JVM Bytecode Compilation

```
add:(II)I
  iload_0
  iload_1
  iadd
  ireturn
```



```
define i32 @"Test.add:(II)I"(i32 %0, i32 %1) {
  %op0 = alloca ptr
  %op1 = alloca ptr
  %local0 = alloca ptr
  %local1 = alloca ptr
  store i32 %0, ptr %local0
  store i32 %1, ptr %local1
  …
}
```

JVM Bytecode Compilation

```
add:(II)I
  iload_0
  iload_1
  iadd
  ireturn
```

```
define i32 @"Test.add:(II)I"(i32 %0, i32 %1) {
  …
  %7 = load i32, ptr %local0
  store i32 %7, ptr %op0
  %8 = load i32, ptr %local1
  store i32 %8, ptr %op1
  …
}
```

JVM Bytecode Compilation

```
add:(II)I
  iload_0
  iload_1
  iadd
  ireturn
```

```
define i32 @"Test.add:(II)I"(i32 %0, i32 %1) {
  …
  %9 = load i32, ptr %op1
  %10 = load i32, ptr %op0
  %11 = add i32 %10, %9
  store i32 %11, ptr %op0
  …
}
```

JVM Bytecode Compilation

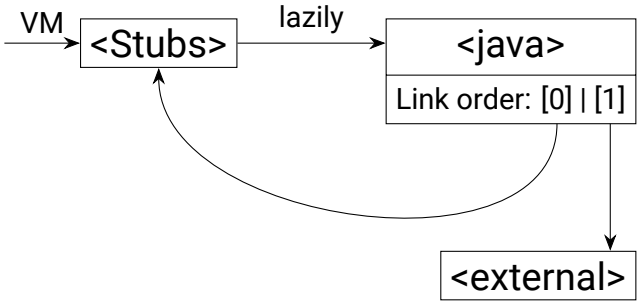
```
add:(II)I
  iload_0
  iload_1
  iadd
  ireturn
```

```
define i32 @"Test.add:(II)I"(i32 %0, i32 %1) {
  …
  %12 = load i32, ptr %op0
  ret i32 %12
}
```

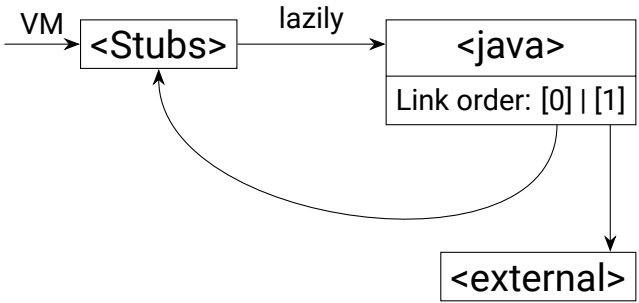
ORC Integration

```
/// Materialization unit to add a JVM Byte code method to the JITLink graph
/// and materializing it once required.
class ByteCodeMaterializationUnit : public MaterializationUnit {
public:
  /// Creates a materialization unit for the given method.
  /// Compilation is done using 'layer'.
  ByteCodeMaterializationUnit(ByteCodeLayer& layer, const Method* method);

  void materialize(std::unique_ptr<MaterializationResponsibility> r) override;
};
…

dylib.define(std::make_unique<ByteCodeMaterializationUnit>(layer, method));
```

ORC Integration - `JITDylibs`

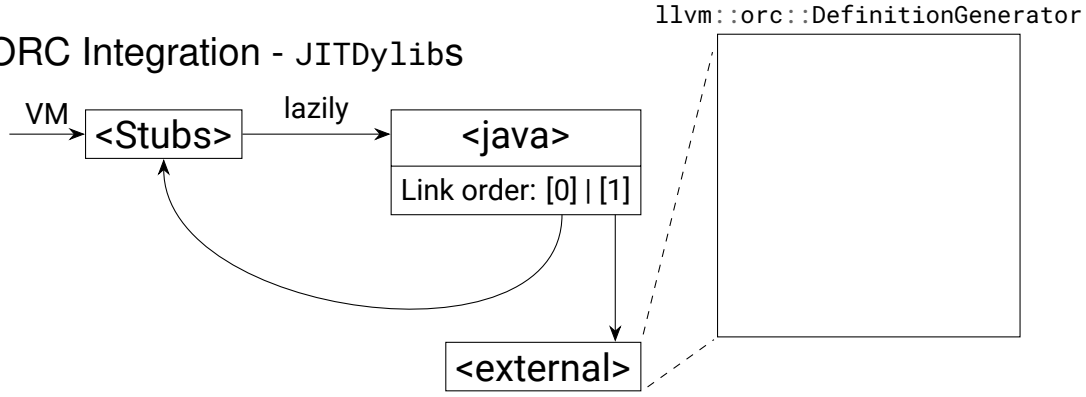


ORC Integration - `JITDylibs`



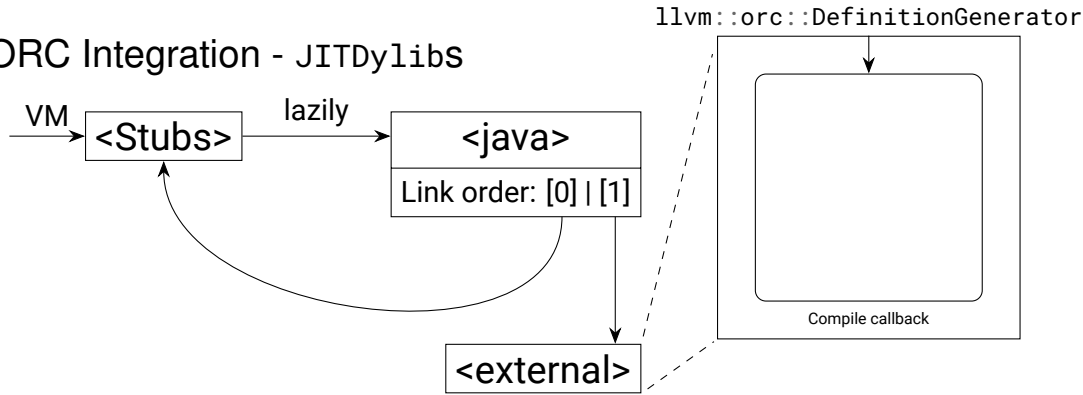

```
java.define(std::make_unique<ByteCodeMaterializationUnit>(layer, method));
stubs.define(lazyReexports(…, java, symbolName(method)));
```

ORC Integration - `JITDylibs`



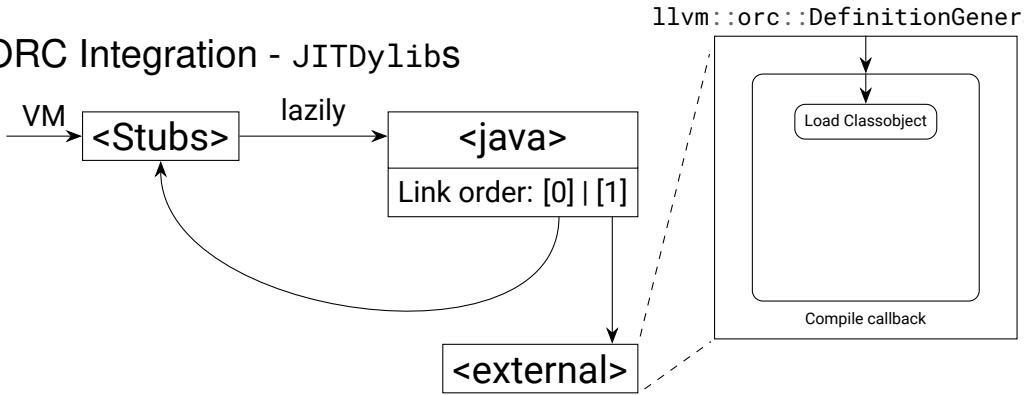

```
java.define(std::make_unique<ByteCodeMaterializationUnit>(layer, method));
stubs.define(lazyReexports(…, java, symbolName(method)));
```

ORC Integration - `JITDylibs`



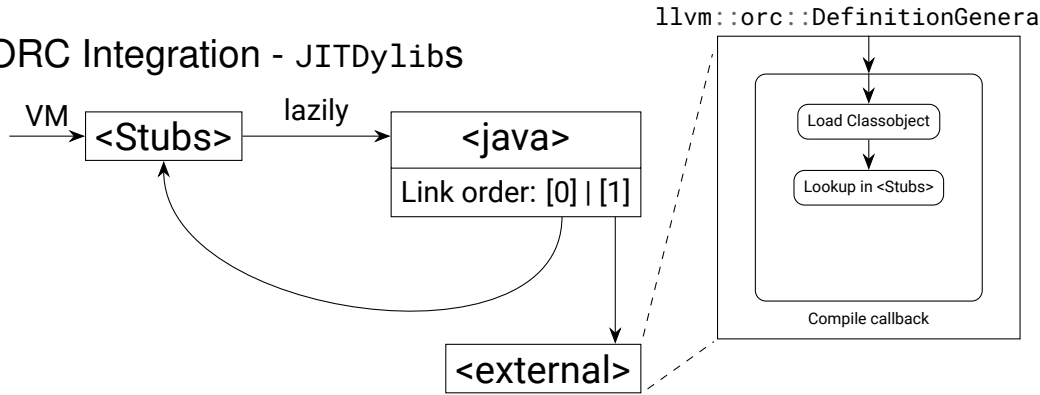

```
java.define(std::make_unique<ByteCodeMaterializationUnit>(layer, method));
stubs.define(lazyReexports(…, java, symbolName(method)));
```

ORC Integration - `JITDylibs`



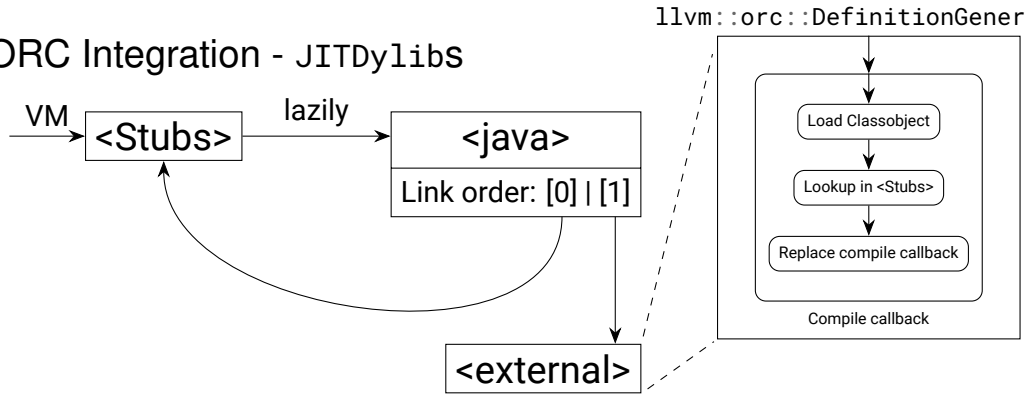

```
java.define(std::make_unique<ByteCodeMaterializationUnit>(layer, method));
stubs.define(lazyReexports(…, java, symbolName(method)));
```

ORC Integration - `JITDylibs`



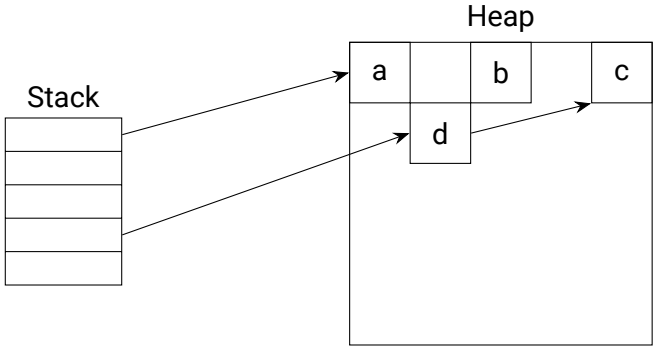

```
java.define(std::make_unique<ByteCodeMaterializationUnit>(layer, method));
stubs.define(lazyReexports(…, java, symbolName(method)));
```

ORC Integration - `JITDylibs`

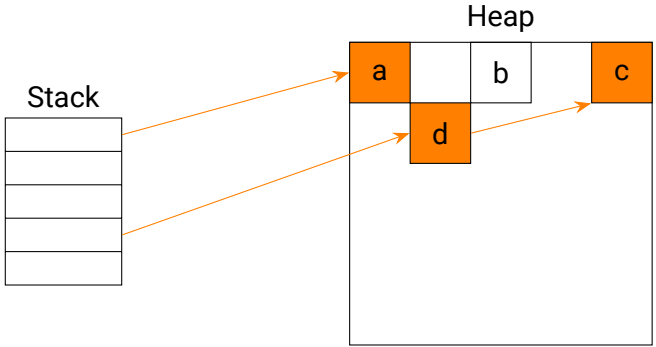



```
java.define(std::make_unique<ByteCodeMaterializationUnit>(layer, method));
stubs.define(lazyReexports(…, java, symbolName(method)));
```

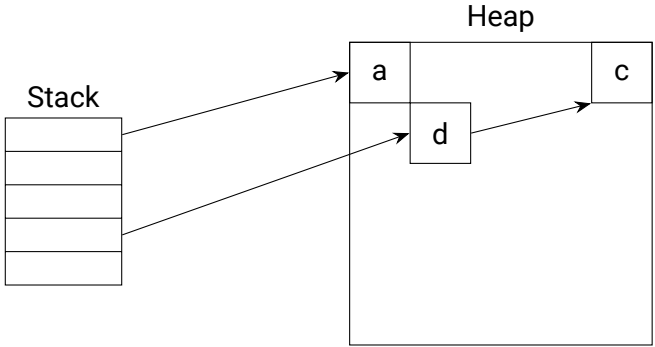
Relocating garbage collection - Mark



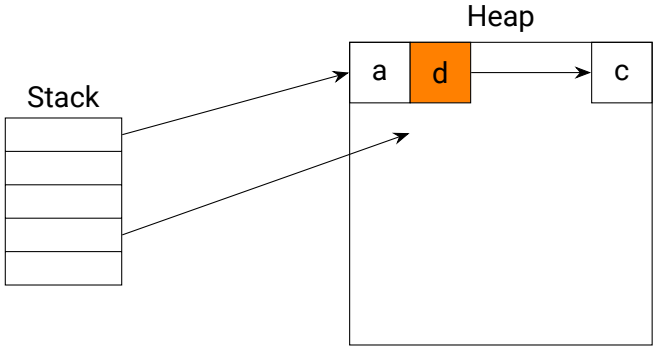
Relocating garbage collection - Mark



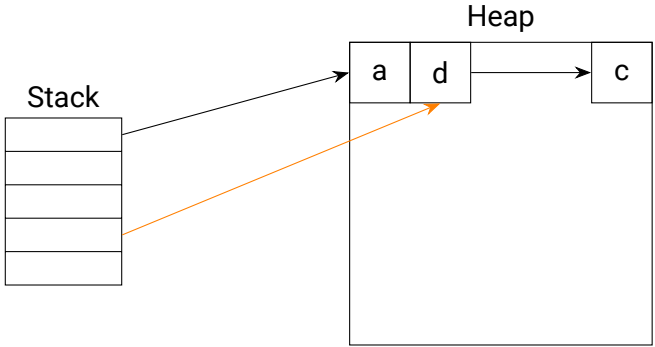
Relocating garbage collection - Sweep



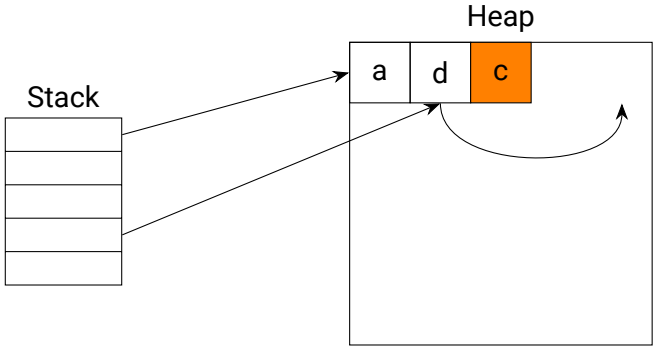
Relocating garbage collection - Sweep



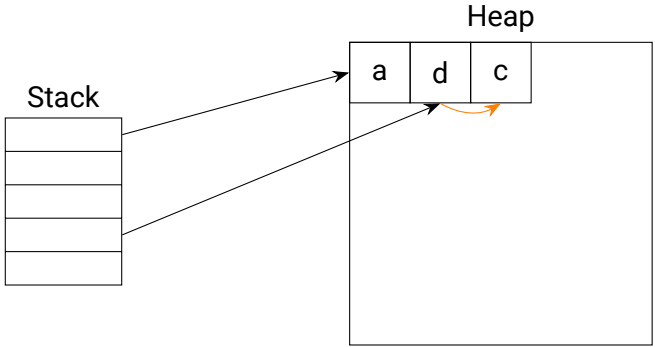
Relocating garbage collection - Sweep



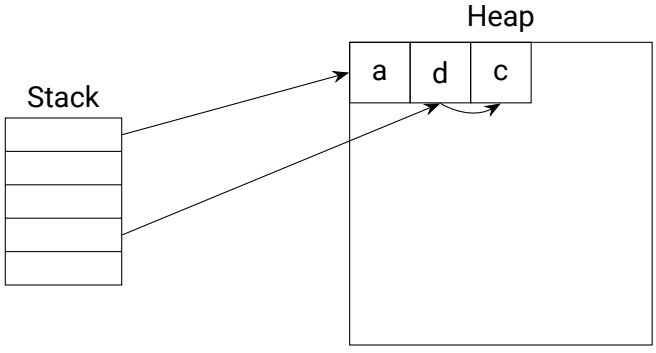
Relocating garbage collection - Sweep



Relocating garbage collection - Sweep



Relocating garbage collection - Sweep



Requirements:

- Find stack references
- Mutate stack references

Statepoints

1. Reference = `ptr addrspace(1)`

Statepoints

1. Reference = `ptr addrspace(1)`
2. Add `define i32 @"Test.add:(II)I"(i32 %0, i32 %1) gc "coreclr"`

Statepoints

1. Reference = `ptr addrspace(1)`
2. Add `define i32 @"Test.add:(II)I"(i32 %0, i32 %1) gc "coreclr"`
3. Schedule:

```
passBuilder.registerOptimizerLastEPCallback(
  [&](ModulePassManager& modulePassManager, OptimizationLevel)
  {
      modulePassManager.addPass(RewriteStatepointsForGC{});
  });
```

Statepoints - `RewriteStatepointsForGC`

Before:

```
define void @"java/lang/Class.<clinit>:()V"() gc "coreclr" {
  %alive = reference
  …
  call void @"String$CaseInsensitiveComparator.<init>"(%9)
  store ptr addrspace(1) %alive, ptr %loc
  …
}
```

Statepoints - `RewriteStatepointsForGC`

Before:

```
define void @"java/lang/Class.<clinit>:()V"() gc "coreclr" {
  %alive = reference
    …
    call void @"String$CaseInsensitiveComparator.<init>"(%9)
    store ptr addrspace(1) %alive, ptr %loc
    …
}
```

Statepoints - `RewriteStatepointsForGC`

After:

```
define void @"java/lang/String.<clinit>:()V"() gc "coreclr" {
  %alive = reference
  …
  %token = call token @llvm.experimental.gc.statepoint(…
    @"String$CaseInsensitiveComparator.<init>", %9, …)
      [ "gc-live"(ptr addrspace(1) %alive) ]
  %alive_re = call @llvm.experimental.gc.relocate.p1(token %token, …)
  store ptr addrspace(1) %alive_re, ptr %loc
  …
}
```

Statepoints - `RewriteStatepointsForGC`

After:

```
define void @"java/lang/String.<clinit>:()V"() gc "coreclr" {
  %alive = reference
  …
  %token = call token @llvm.experimental.gc.statepoint(…
    @"String$CaseInsensitiveComparator.<init>", %9, …)
      [ "gc-live"(ptr addrspace(1) %alive) ]
  %alive_re = call @llvm.experimental.gc.relocate.p1(token %token, …)
  store ptr addrspace(1) %alive_re, ptr %loc
  …
}
```

Statepoints - `RewriteStatepointsForGC`

After:

```
define void @"java/lang/String.<clinit>:()V"() gc "coreclr" {
  %alive = reference
  …
  %token = call token @llvm.experimental.gc.statepoint(…
    @"String$CaseInsensitiveComparator.<init>", %9, …)
      [ "gc-live"(ptr addrspace(1) %alive) ]
  %alive_re = call @llvm.experimental.gc.relocate.p1(token %token, …)
  store ptr addrspace(1) %alive_re, ptr %loc
  …
}
```

Stackmap in `JITLink`

4. Read Stackmap:

```
/// JIT link plugin for extracting the LLVM generated stack map section
/// out of materialized objects and notifying the GC about newly added
/// entries.
class StackMapRegistrationPlugin : public ObjectLinkingLayer::Plugin {
public:
  …
  void modifyPassConfig(MaterializationResponsibility&, LinkGraph&,
    PassConfiguration& config) override;
```

Consuming in GC

```
@"String$CaseInsensitiveComparator.<init>", %9,
  [ "gc-live"(ptr addrspace(1) %alive) ]
```

Consuming in GC

```
@"String$CaseInsensitiveComparator.<init>", %9,
  [ "gc-live"(ptr addrspace(1) %alive) ]
```

```
switch (kind) {
  case Constant:
    return m_union.constant;
```

Consuming in GC

```
@"String$CaseInsensitiveComparator.<init>", %9,
  [ "gc-live"(ptr addrspace(1) %alive) ]
```

```
switch (kind) {
  case Constant:
    return m_union.constant;
  case Register:
    return unw_get_reg(cursor, m_union.registerNumber);
```

Consuming in GC

```
@"String$CaseInsensitiveComparator.<init>", %9,
  [ "gc-live"(ptr addrspace(1) %alive) ]
```

```
switch (kind) {
  case Constant:
    return m_union.constant;
  case Register:
    return unw_get_reg(cursor, m_union.registerNumber);
  case Direct:
    return unw_get_reg(cursor, m_union.registerNumber) + m_union.offset;
```

Consuming in GC

```
@"String$CaseInsensitiveComparator.<init>", %9,
  [ "gc-live"(ptr addrspace(1) %alive) ]
```

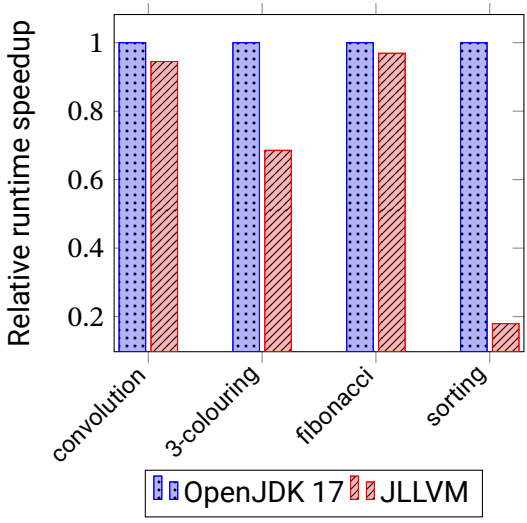
```
switch (kind) {
  case Constant:
    return m_union.constant;
  case Register:
    return unw_get_reg(cursor, m_union.registerNumber);
  case Direct:
    return unw_get_reg(cursor, m_union.registerNumber) + m_union.offset;
  case Indirect:
    uinptr_t result;
    auto* ptr = unw_get_reg(cursor, m_union.registerNumber) + m_union.offset;
    std::memcpy(&result, ptr, m_union.size);
    return result;
}
```

```
markus@EuroLLVM2024:~$ time jllvm HelloWorld.class
```

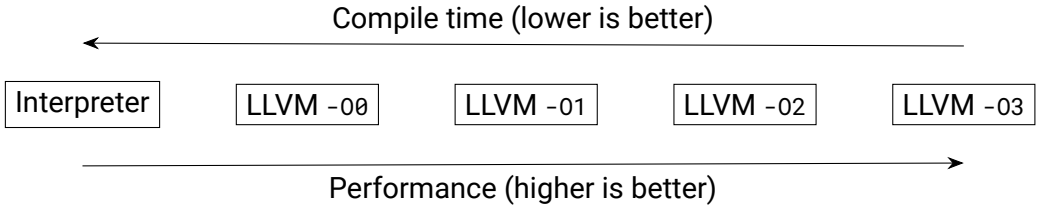
```
markus@EuroLLVM2024:~$ time jllvm HelloWorld.class
Hello World

real    0m1.947s
user    0m1.894s
sys     0m0.042s
```

```
markus@EuroLLVM2024:~$ time jllvm HelloWorld.class
Hello World

real    0m1.947s
user    0m1.894s
sys     0m0.042s
markus@EuroLLVM2024:~$ time java HelloWorld
Hello World

real    0m0.020s
user    0m0.011s
sys     0m0.011s
```

```
markus@EuroLLVM2024:~$ time jllvm HelloWorld.class
Hello World

real    0m1.947s
user    0m1.894s
sys     0m0.042s
markus@EuroLLVM2024:~$ time java HelloWorld
Hello World

real    0m0.020s
user    0m0.011s
sys     0m0.011s

markus@EuroLLVM2024:~$ jllvm HelloWorld.class -Xdebug 2>&1 \
  | grep "Emitting LLVM IR" \
  | wc -l
462
```


Relative runtime speedup
1
0.8
0.6
0.4
0.2
convolution
3-colouring
fibonacci
sorting
OpenJDK 17
JLLVM

Multi-tier VMs



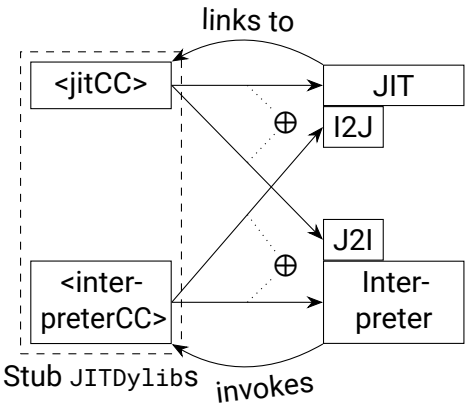
Multi-tier VMs

Compile time (lower is better)

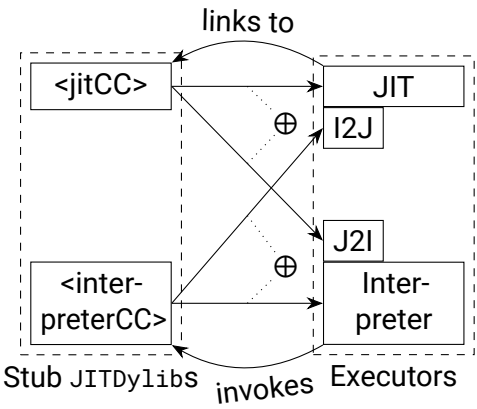
Interpreter | LLVM -O0 | LLVM -O1 | LLVM -O2 | LLVM -O3

Performance (higher is better)

Multi-tier in ORC



Multi-tier in ORC




```
/// Abstract interface for all classes
/// capable of executing Java methods.
class Executor {
public:
  /// Registers a method within the executor,
  /// making it available in the dylibs
  /// returned by 'getJITCCDylib' and
  /// 'getIntCCDylib'.
  virtual void add(const Method& method) = 0;

  virtual JITDylib& getJITCCDylib() = 0;

  virtual JITDylib& getIntCCDylib() = 0;
};
```

Tier-up triggers

Tier-up triggers

1. Method Invocation

```
int64_t interpreterEntry(Method* method, int64_t* arguments) {
  if (method->incrementInvocationCounter() >= m_invocationThreshold)
  {
      m_runtime.changeExecutor(method, m_virtualMachine.getJIT());
      return method->callInterpreterCC(arguments);
  }
  return interpreterLoop(method, arguments);
}
```

Tier-up triggers

2. Hot loop

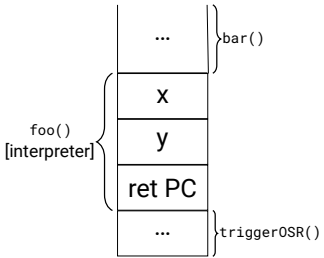
```
match(result, [&](SetPC setPc) {
    // Backedge.
    if (setPc.newPC < offset) {
        backEdgeCounter++;
        if (backEdgeCounter == m_backEdgeThreshold)
            escapeToJIT();
    }
    curr = ByteCodeIterator(codeArray.data(), setPc.newPC);
    …
```

Tier-up triggers

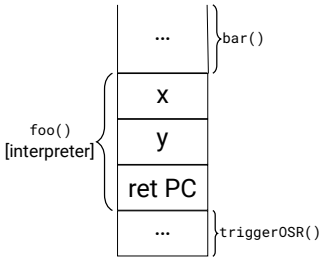
2. Hot loop

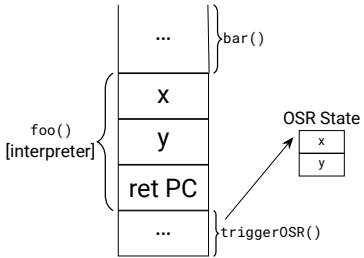
```
match(result, [&](SetPC setPc) {
    // Backedge.
    if (setPc.newPC < offset) {
        backEdgeCounter++;
        if (backEdgeCounter == m_backEdgeThreshold)
            escapeToJIT();
    }
    curr = ByteCodeIterator(codeArray.data(), setPc.newPC);
    …
```

On-Stack replacement

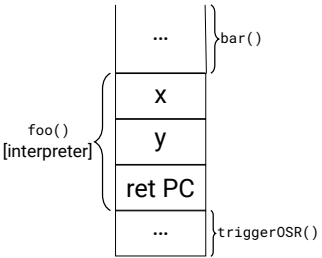


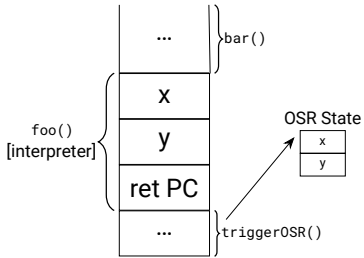
On-Stack replacement

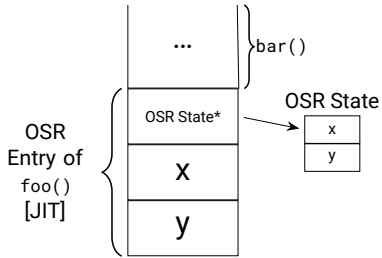




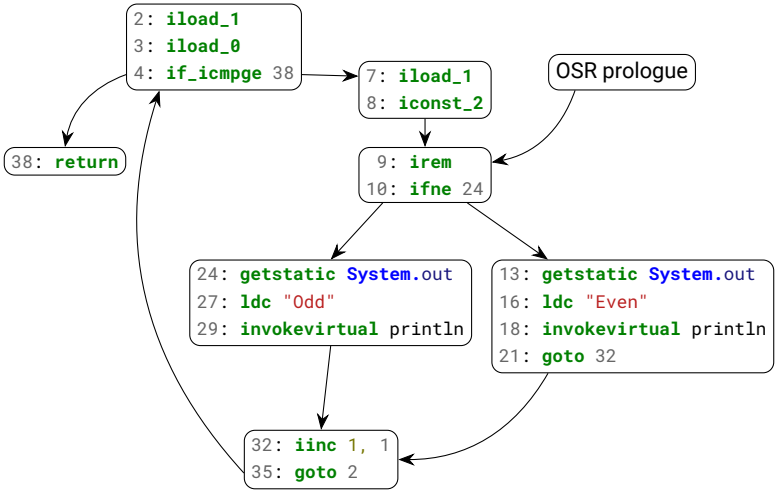
On-Stack replacement





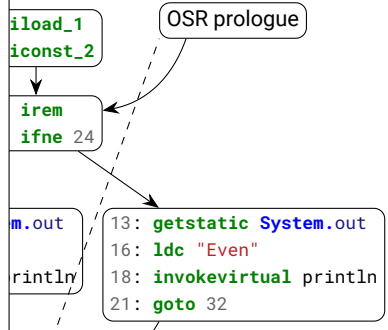


Compiled OSR Entry



Compi

```
define void @"Test.oddOrEven:(II)V$9"(ptr %0)
  %1 = alloca ptr
  %2 = alloca ptr
  %3 = alloca ptr
  %4 = alloca ptr
  %5 = alloca ptr
  %6 = alloca ptr
  %7 = getelementptr i64, ptr %0, i32 0
  %8 = load ptr addrspace(1), ptr %7
  store ptr addrspace(1) %8, ptr %3
  %9 = getelementptr i64, ptr %0, i32 1
  %10 = load i32, ptr %9
  store i32 %10, ptr %4
  %11 = getelementptr i64, ptr %0, i32 2
  %12 = load i32, ptr %11
  store i32 %12, ptr %5
  %13 = getelementptr i64, ptr %0, i32 3
  %14 = load i32, ptr %13
  store i32 %14, ptr %6
  %15 = getelementptr i64, ptr %0, i32 4
  %16 = load i32, ptr %15
  store i32 %16, ptr %1
  %17 = getelementptr i64, ptr %0, i32 5
  %18 = load i32, ptr %17
  store i32 %18, ptr %2
  call void @jllvm_osr_frame_delete(ptr %0)
```



Deoptimizing JIT frames

```
int i = 5;
try {
  i = foo();
} catch (Exception ignored) {
  return i;
}
return 0;
```

Deoptimizing JIT frames

```
int i = 5;
try {
  i = foo();
} catch (Exception ignored) {
  return i;
}
return 0;
```

```
define i32 @"Test.bar:()I"() gc "coreclr"
  …
  %5 = load i32, ptr %local0
  %6 = call i32 @"Test.foo:()I"()
    [ "deopt"(i16 2, i16 2, i32 %5) ]
  store i32 0, ptr %op0
  %7 = load i32, ptr %op0
  ret i32 %7
}
```

Deoptimizing JIT frames

```
int i = 5;
try {
  i = foo();
} catch (Exception ignored) {
  return i;
}
return 0;
```

```
define i32 @"Test.bar:()I"() gc "coreclr"
  …
  %5 = load i32, ptr %local0
  %6 = call i32 @"Test.foo:()I"()
    [ "deopt"(i16 2, i16 2, i32 %5) ]
  store i32 0, ptr %op0
  %7 = load i32, ptr %op0
  ret i32 %7
}
```

Deoptimizing JIT frames

```
int i = 5;
try {
  i = foo();
} catch (Exception ignored) {
  return i;
}
return 0;
```

```
define i32 @"Test.bar:()I"() gc "coreclr"
  …
  %5 = load i32, ptr %local0
  %6 = call i32 @"Test.foo:()I"()
    [ "deopt"(i16 2, i16 2, i32 %5) ]
  store i32 0, ptr %op0
  %7 = load i32, ptr %op0
  ret i32 %7
}
```

Installing an OSR entry

```
std::uintptr_t nextStack = nextFrame.getIntegerRegister(UNW_REG_SP);
if constexpr (returnAddressOnStack)
    nextStack += (stackGrowsDown ? -1 : 1) * sizeof(void (*)());

nextFrame.setIntegerRegister(UNW_REG_IP, functionPointer);
nextFrame.setIntegerRegister(UNW_REG_SP, nextStack);

// Set the function arguments.
for (std::size_t i = 0; i < arguments.size(); i++)
    nextFrame.setIntegerRegister(argRegisterNumbers[i], arguments[i]);
```

Installing an OSR entry

```
_Unwind_ForcedUnwind(…,
  +[](…, _Unwind_Context* context, void* stopPc) {
      std::uintptr_t pc = _Unwind_GetIP(context);
      if (pc != reinterpret_cast<std::uintptr_t>(stopPc))
          return _URC_NO_REASON;

      unw_cursor_t cursor = exception->frame.m_cursor;
      _Unwind_DeleteException(exception);

      unw_resume(&cursor);
  }, getProgramCounter());
```

Conclusion

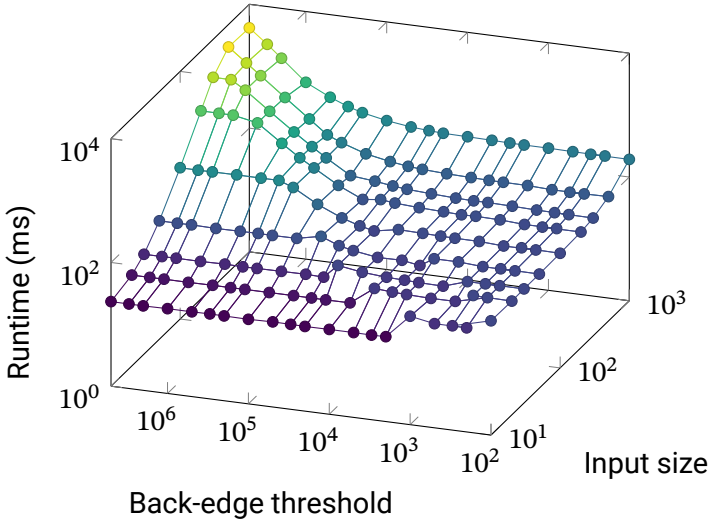
- Java SE 17 Virtual Machine Specification with OpenJDK 17 Class Library
- ORCv2 architecture
- Relocating garbage collector
- Two execution tiers:
  - Interpreter
  - `-O3` LLVM JIT
- Tier-up on method entry and in hot-loops
- On-Stack replacement
- Open source at `https://github.com/JLLVM/JLLVM`!

Thank you!

Stackmap structure

```
type Stackmap = {
  Entries: Entry[],
}

type Entry = {
  ProgramCounter: u64,
  DeoptOperands: Location[],
}

type Location = Register
              | Direct
              | Indirect
              | Constant
```

```
type Register = {
  RegisterNumber: int
}

type Direct = {
  FPRegisterNumber: int,
  Offset: i32,
}

type Indirect = {
  FPRegisterNumber: int,
  Offset: i32,
  Size: int,
}
```


Runtime (ms)
10^4
10^2
10^0
10^6
10^5
10^4
10^3
10^2
Back-edge threshold
10^3
10^2
10^1
Input size