JSR 292 Backport
(first year report)

Rémi Forax
JVM Summit'09

Objective

- Support JSR 292 features for jdk5/jdk6 VM
  - Method Handle
  - Invokedynamic
  - ~~Interface injection~~ (not yet !)
- Same speed than jdk7 VM

  At least, less than 2 times slower

How it works ?

- Use ASM (3.2) to
  - replace invokedynamic/MH.invoke by stubs
  - rename java/dyn/* to jsr292/java/dyn/*
  - insert java.lang.Class when caller class is needed
- Transformation during classloading
  - Java agent
- Retransformation at runtime
  - Java agent

jsr292.java.dyn.MethodHandle

- A class that contains 9 methods $invoke$
  - 8 with 0 to 7 parameters
  - 1 with an array of object
- All method handle adapters inherit from MethodHandle and implement these 9 methods
- Borrowed to JRuby pre-invokedynamic design

Example

```
class MHGuarder extends MethodHandle {
  private final MethodHandle test;
  private final MethodHandle target;
  private final MethodHandle fallback;

  @Override
  public Object $invoke$() throws Throwable {
     return ((Boolean)test.$invoke$())?
       target.$invoke$():
       fallback.$invoke$();
  }

  @Override
  public Object $invoke$(Object o1) throws Throwable {
    return ((Boolean)test.$invoke$(o1))?
      target.$invoke$(o1):
      fallback.$invoke$(o1);
    }
```

Stub

- Replace invokedynamic/MH.invoke call
  => method call to a private method (stub)

- Stub code:
  - Box primitive argument
  - Box in an array if more than 7 arguments
  - invokevirtual one of $invoke$(Object, Object, ...)
  - Unbox return value

Method Handle adapters

- All methods of class MethodHandles.Lookup are implemented using reflection
- Boxing/unboxing conversions are no-ops
- Collect (resp. spread) has 2 versions
  - one for array of object
  - one for array of primitives
- Insert only support insert at position 0 otherwise it uses permute before insert.

Perf problem

- Bytecode instrumentation

  => 4 to 8 times slower than jdk7 (with JIT)

  => Add a runtime optimizer (a kind of JIT)

- Optimizer
  - Add counter at each call site
  - Add a way to retransform bytecode at runtime when the counter meet a threshold
  - Deopt if method handle change

Stub code with optimizer

- In stub, record current MH and a counter
- Check if MH change
  - Yes: reset counter, store new MH
    call generic $invoke$
  - No:
- If counter reaches high_threshold
  - Call instrumentation.retransform on class
  - For all callsite.counter greater than low_threshold
    - Replace stub by bytecode blob generated from MH

Generate a code blob

- Optimization:
  - If adapter tree can be generated
  - Walk the method handle adapter tree
  - For each method handle adapter,
    apply a snippet of bytecode (using ASM)
  - Generate a byte array, and replace the stub code with the blob.
- Reuse the constant pool of the old class, just insert new constants

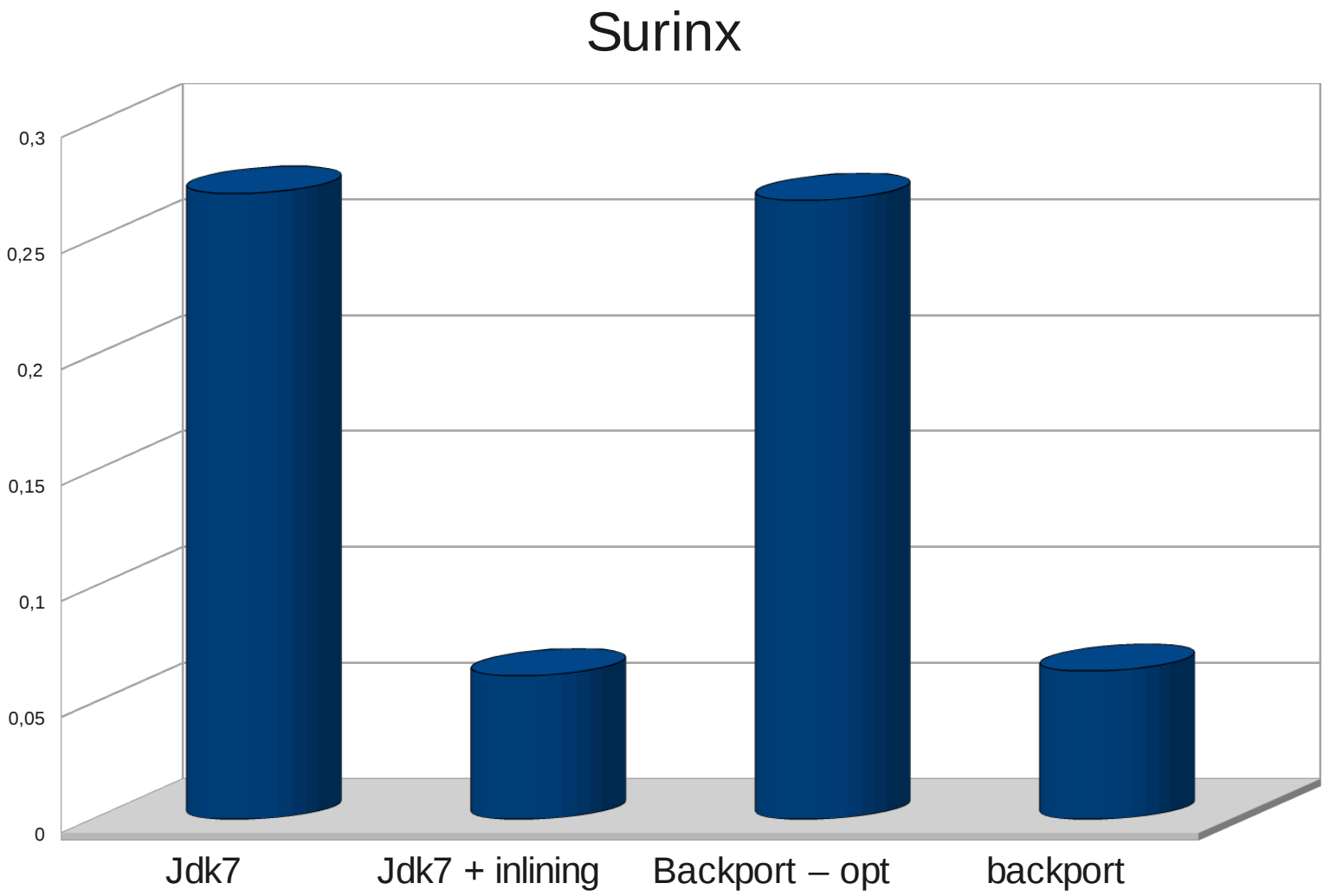
Optimizer calling convention

- Register like convention
  - All parameters are stored in local stack
  - Return value is on the stack
- Great! because:
  - conversions are load/store (with stack slot reuse if possible)
  - permutation too
  - insert is a getstatic/store (or ldc/store)
  - drop is a no-op
  - collect/spread doesn't depend on non boxed parameters

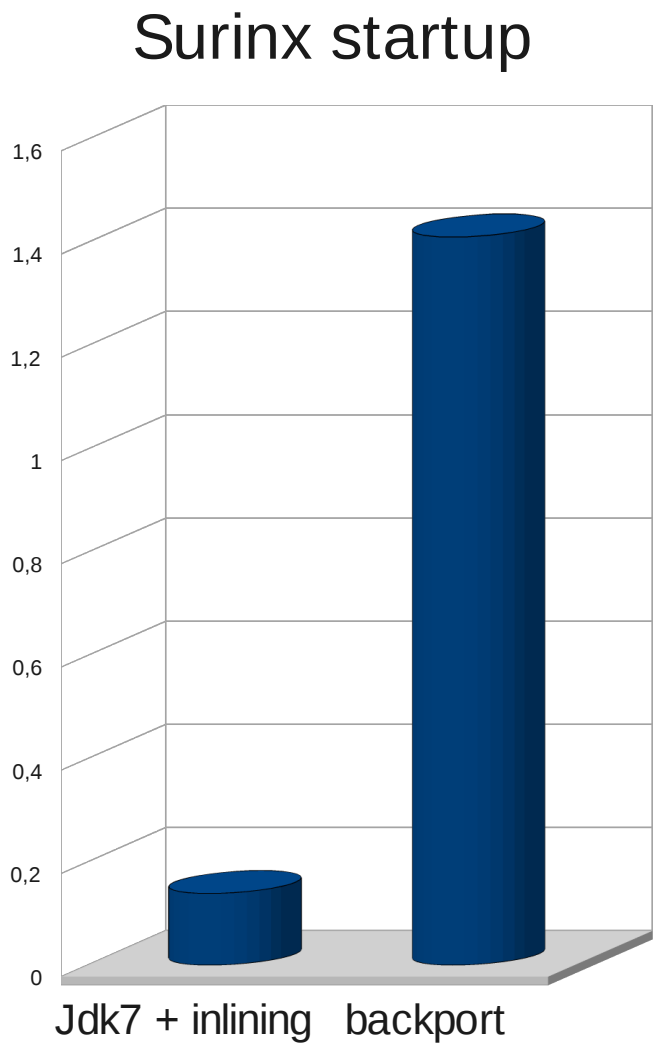
Defeat Optimization

- No optimization if
  - Direct method handle not visible from call site
  - Bound object class not visible from call site
- Optimizer try to be smarter :
  - Use a sub-class for bound object
    and insert a cast
  - Use method of the super-type if overriden
    (not fully implemented)

Surinx Fib Execution Time



Class loading/Retransforming time

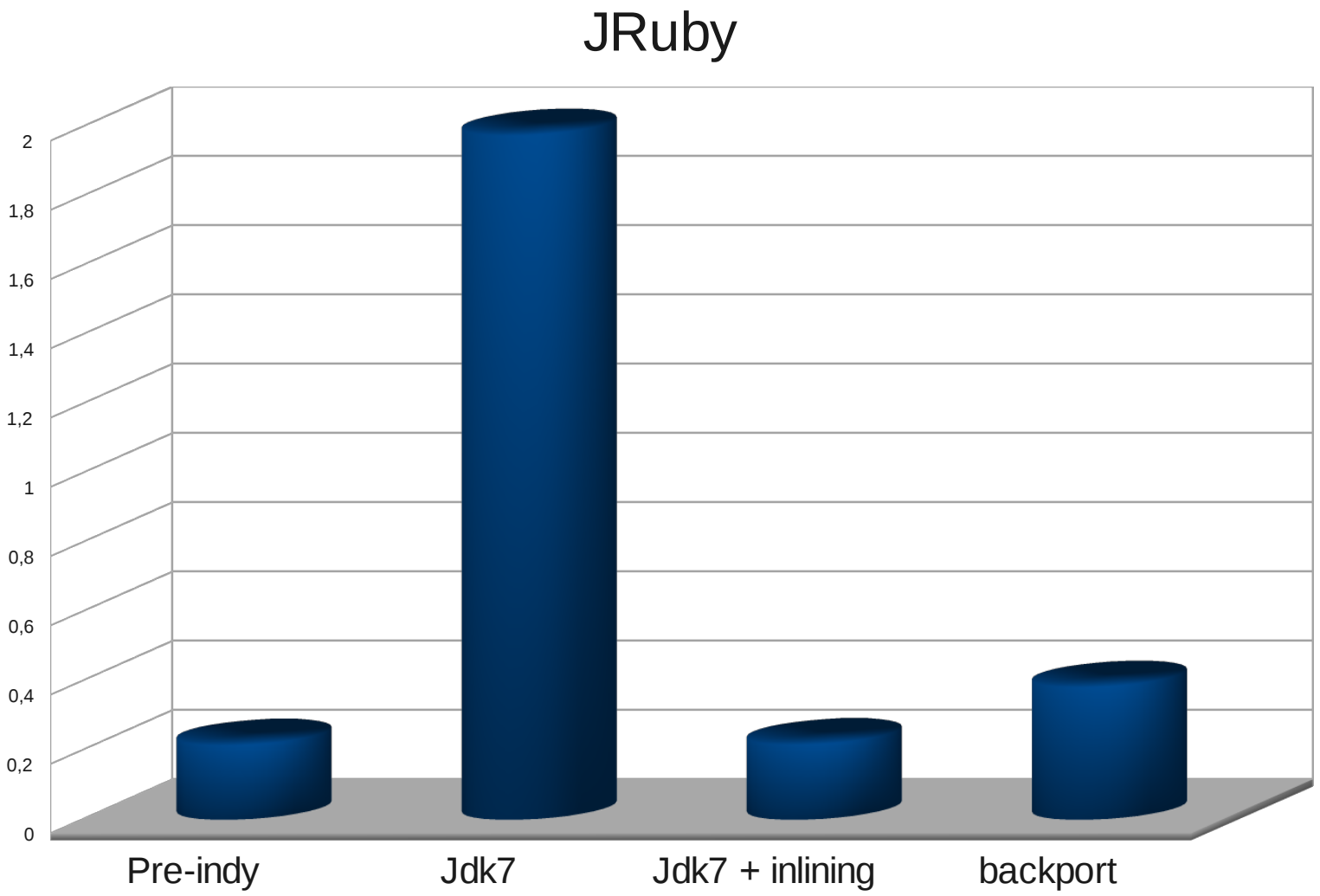


Weaving during class loading

+

Bytecode generation during retransformation

JRuby Fib Execution Time



Todos

- Fix bugs

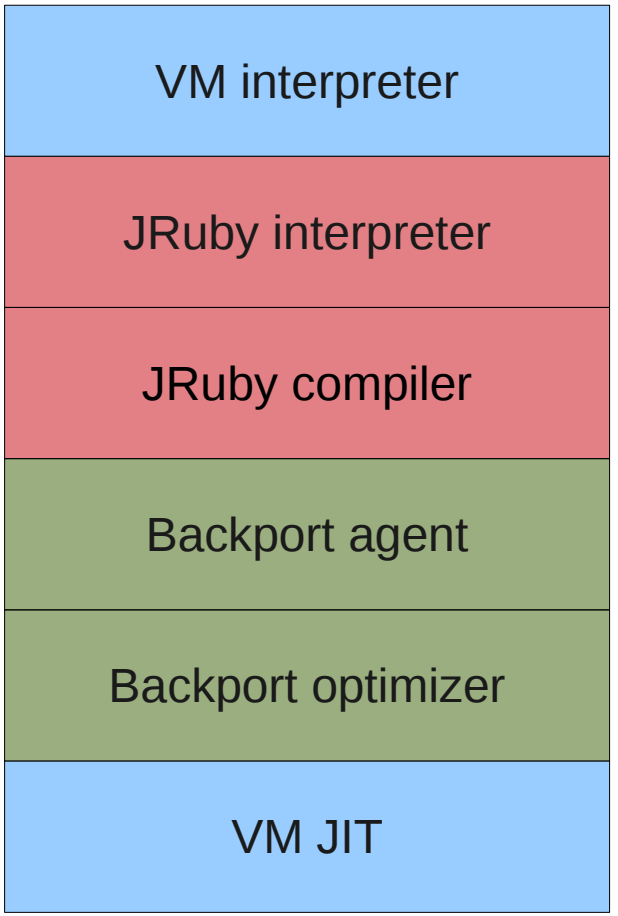
- Threaded retransformation (like hotspot)
- Put stubs in an helper class
- Deoptimization of MethodHandle.invoke should be changed
- Mixed register/stack convention in the optimizer (avoid load/cast/store)
- Add tranpolines at callee side

Backport dirty secret

- MethodHandle.invoke doesn't check the method handle signature
- Why ?
  - Allow to call a method handle with subtypes and the return type to be a supertype
    like a normal Java call
  - It doesn't do any conversions like proposed by Fredrik Ohrstrom
    (require changes in Lookup and in the optimizer)

JRuby + backport

- Stack of interpreter/optimizers



JRuby + backport

- Use online weaver at end of JRuby compiler