Implement JSR 292 on Android

Rémi Forax + IG2K IR team
Université Paris-Est Marne-la-Vallée

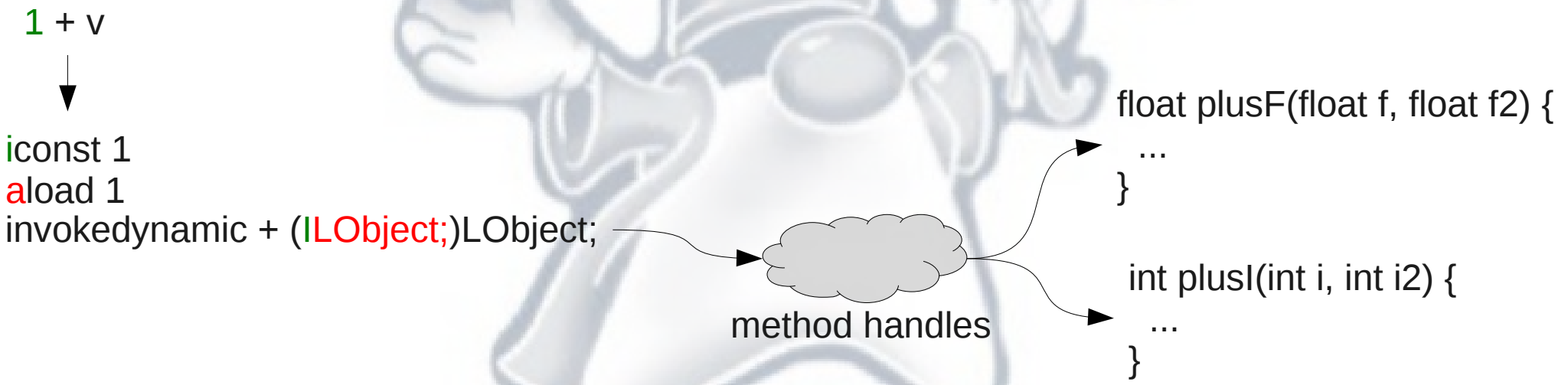
IndyDroid

- Short intro to JSR 292
- An example
- Dalvik
  - dx, dexopt, DEX
  - add method handles
  - add invokedynamic
- Schedule & Future

Overview of JSR 292

Uncouple call site and callee

- invokedynamic
- method handle



+ API for creating/managing method handles

Method Handle

Single type: java.dyn.MethodHandle

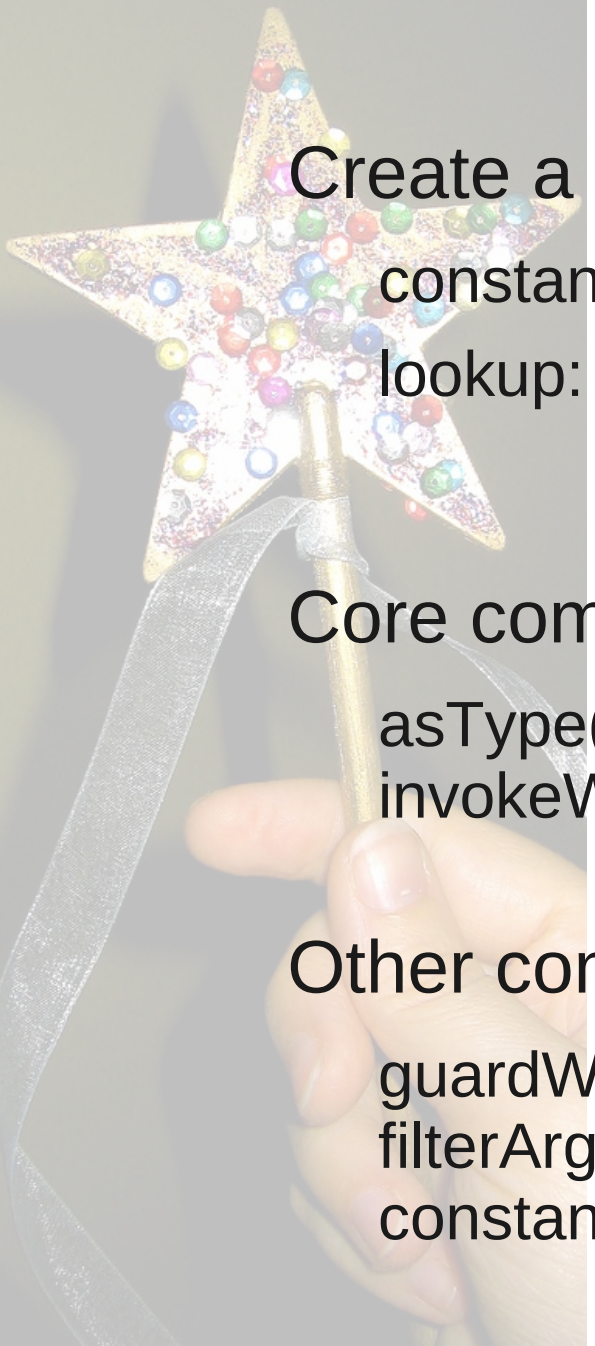
Type safe function pointer

- embodies its type (MethodType)

Callable with 2 polymorphic signature methods

- (int)mh.invokeExact(2, 3.0)
  (int)mh.invokeGeneric(new Integer(2), 3.0)

Managing method handles



Create a Method Handle

constant: String#charAt(int)

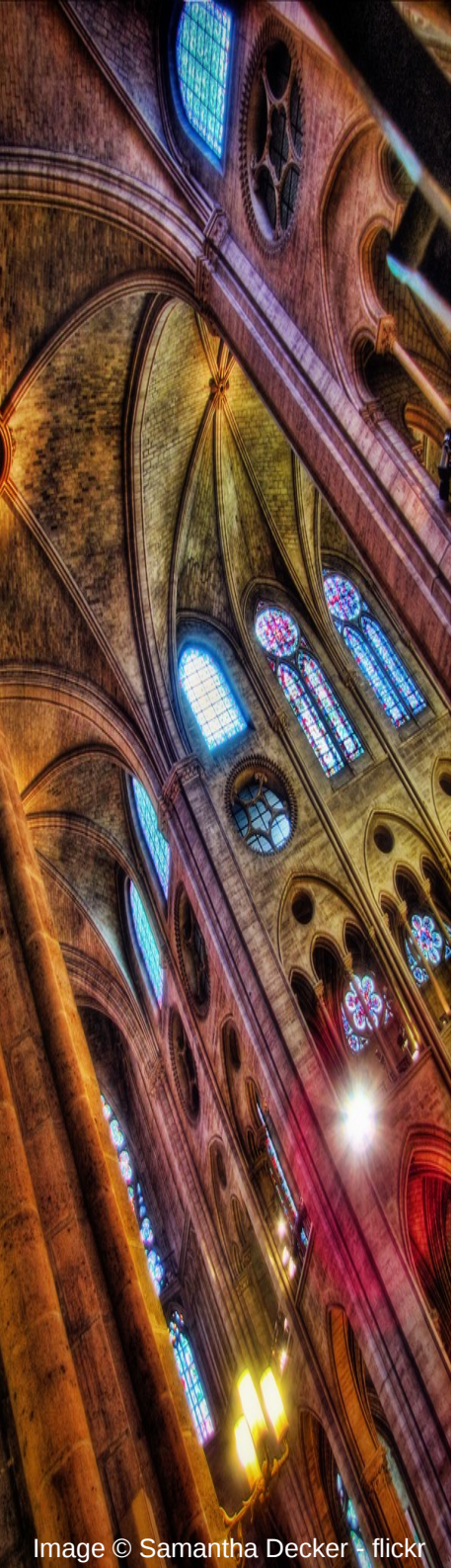
lookup: MHs.lookup().findVirtual(String.class,
"charAt", methodType(char.class, int.class))

Core combinators

asType(), bindTo(), asCollector()/asSpreader(),
invokeWithArguments()

Other combinators

guardWithTest(), dropArguments/insertArguments(),
filterArguments()/filterReturnValue(),
constant(), etc.



Invokedynamic

- First call
  - a bootstrap method does the linkage
    - Each invokedynamic has its own BSM
    - Creates a reified CallSite
      - CallSite contains the target method handle
- Next calls
  - Use the target method handle
- CallSite.setTarget()
  - Ask a relinking !

IndyDroid

Short intro to JSR 292

An example

Dalvik

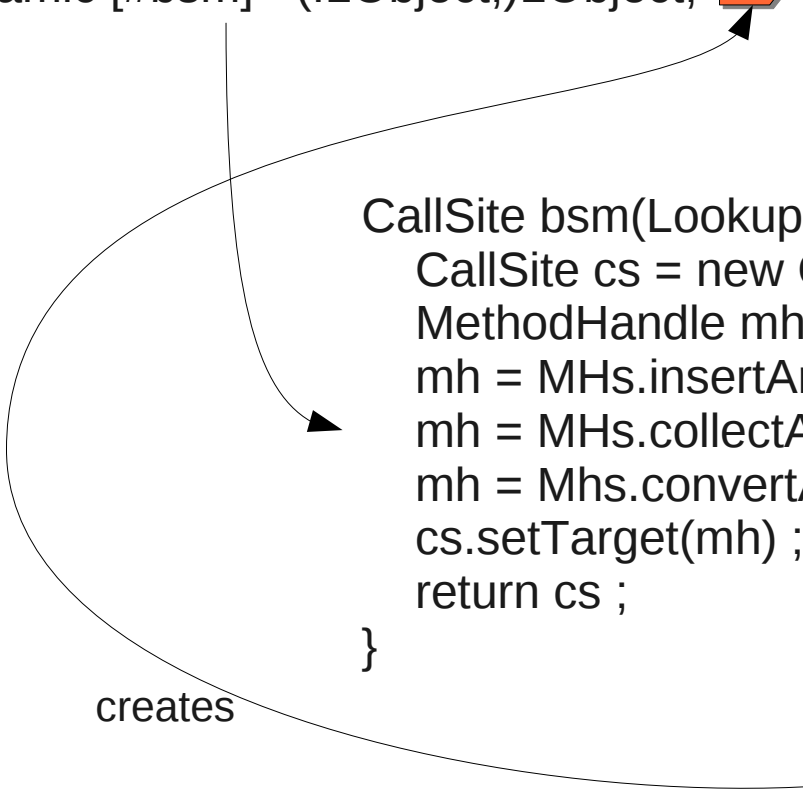
- dx, dexopt, DEX
- add method handles
- add invokedynamic

Future & Conclusion

How to implement an inlining cache ?

The BSM installs a fallback method

```
iconst 1
aload 1
invokedynamic [#bsm] +(ILObject;)LObject;
```



```
CallSite bsm(Lookup lookup, String name, MethodType type) {
    CallSite cs = new CallSite(type);
    MethodHandle mh = #fallback(CallSite, Object[]);
    mh = MHs.insertArguments(mh, 0, cs);
    mh = MHs.collectArguments(mh, type.pCount(), type.generic());
    mh = Mhs.convertArguments(mh, type);
    cs.setTarget(mh) ;
    return cs ;
}
```

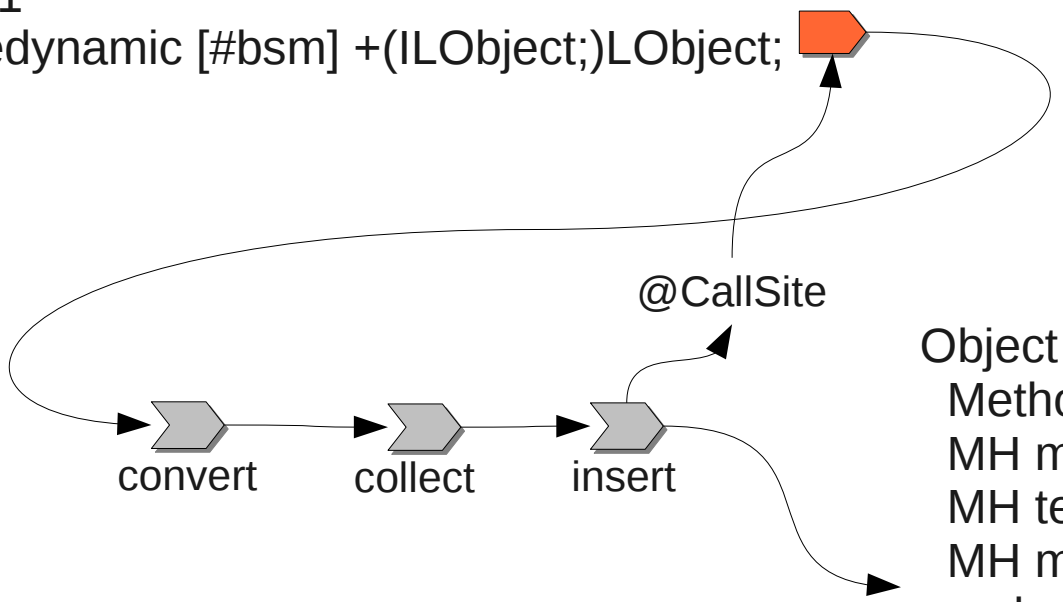
creates


```
Object fallback(CallSite cs, Object[] args) { ... }
```

How to implement an inlining cache ?

The fallback prepends a guard

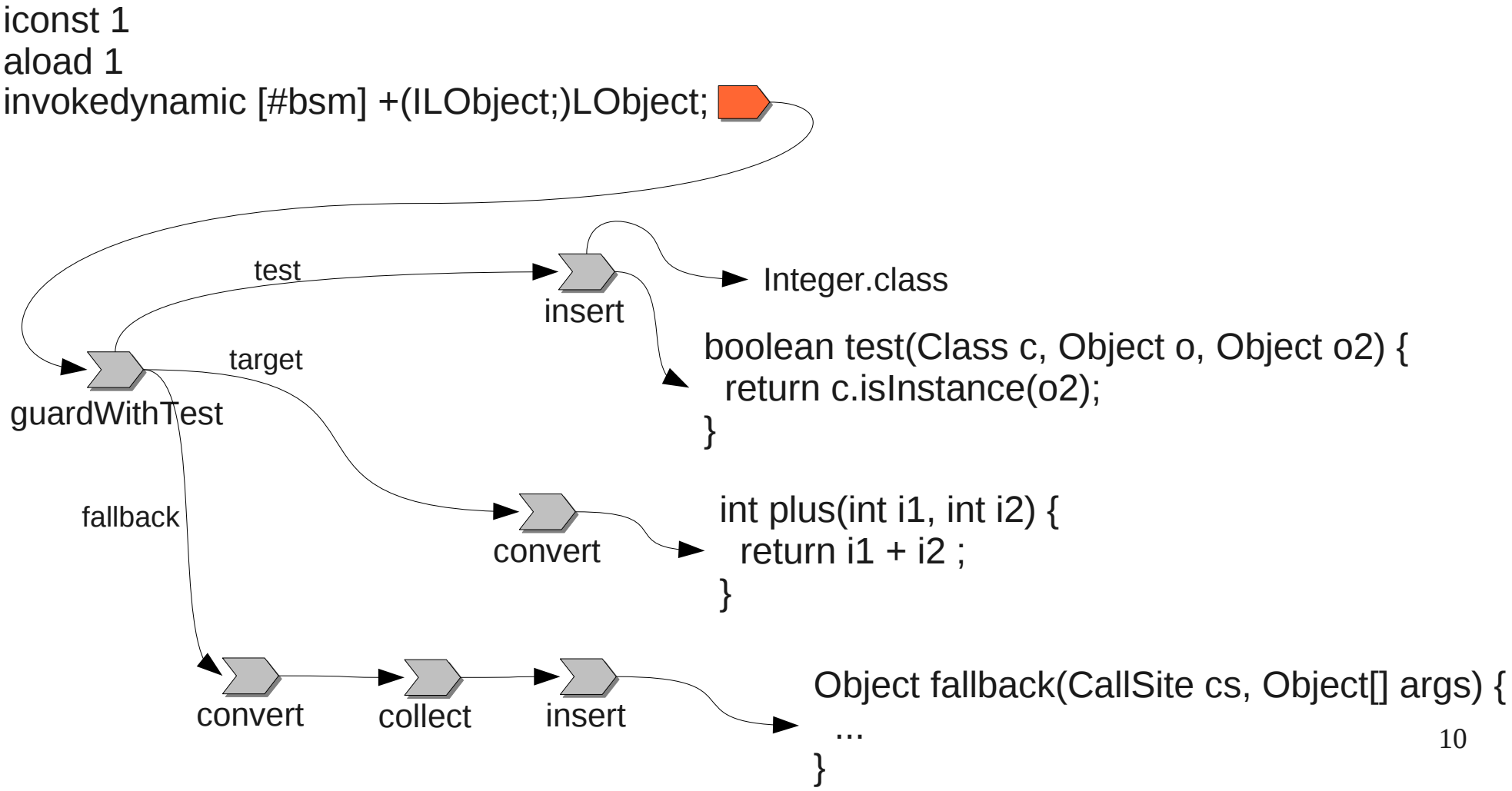
```
iconst 1
aload 1
invokedynamic [#bsm] +(ILObject;)LObject;
```




```
Object fallback(CallSite cs, Object[] args) {
  MethodType type = type(cs, args);
  MH mh = findMH(type, cs.type());
  MH test = findTest(type, mh.type());
  MH mh = MHs.guardWithTest(test,
    mh,
    cs.getTarget());
  cs.setTarget(mh);
  return mh.invokeWithArguments(args);
}
```

How to implement an inlining cache ?

The tree is stable until a new class is discovered



A great book to stick your nose in!

Inlining Cache & Code Patching

FOR

DUMMIES®

become a friend of the VM

A Reference for the Rest of Us!®

By JSR 292

IndyDroid

- Short intro to JSR 292
- An example
- Dalvik
  - dx, dexopt, DEX
  - add method handles
  - add invokedynamic
- Future & Conclusion

Dalvik

Not that fast VM but easy to port

- One template interpreter by platform
- A small JIT

J~~ava bytecode~~, DEX

- Register based + clever encoding tricks
- 1 constant pool by type (String, NameAndType, etc)

Bytecode changes

5 new instructions:

need to add 3 new constant pools.

8 new opcodes:

LDC MethodType	reuse proto cp
LDC MethodHandle	new cp
invokeexact (mh.invokeExact)	
invokegeneric (mh.invokeGeneric)	
invokedynamic	...

Invodynamic in DEX

invokedynamic [#bsm, "name"] plus(1, 2.0) (ID)I

invokedynamic instruction

Constants splits in 3 parts :

- nameAndType foo (ID)I
- bsm #bsm
- bsm constants ["name"]

+ arguments registers

2 new constant pools

InvokeDynamic [NameAndType, MethodHandle, BSM]

BSM (variable length) pairs of cp type/cp index

IndyDroid

Short intro to JSR 292

An example

Dalvik

- dx, dexopt, DEX
- add method handles
- add invokedynamic

Future & Conclusion

Anatomy of a Method Handle

C function pointer + method type (safety) + invocation mode

Data structure :

2 words object header

type : MethodType (interned in Java code)

method : *Method

call_mode : special|static|virtual|interface|direct + a varargs bit

Method Handle

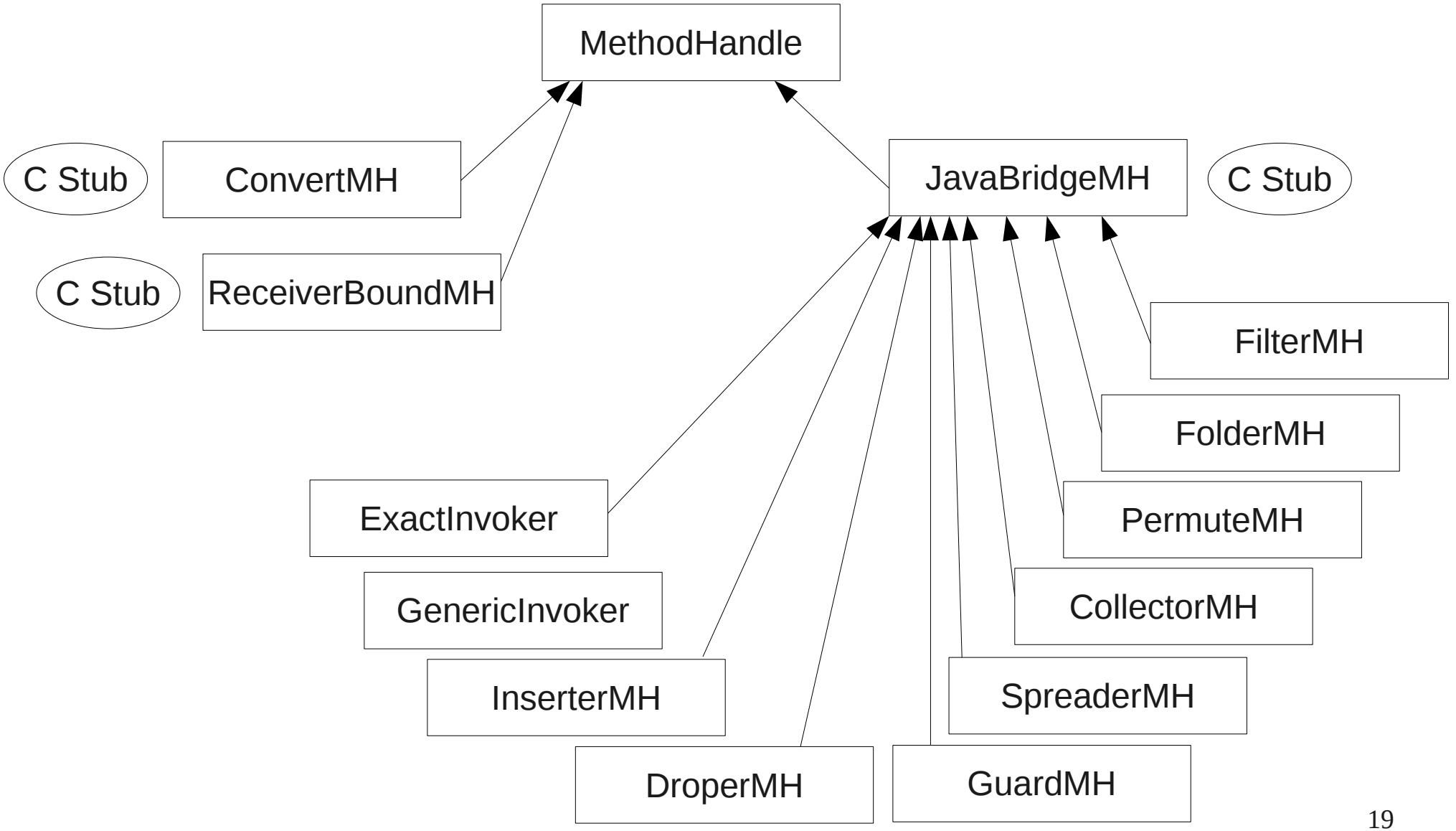
Several of kind Method*:

- direct/static java call
  - ex : identity: One identity() by primitive + Object
- vtable/itable slot
- native
  - VM function in C / JNI
  - javaBridge

Several subclasses :

- C stub subclasses + javaBridge subclasses

Method Handle Hierarchy



The idea is to move MHs from right to left

JavaBridge

Special C Stub:

```
if (paramCount<10)
  invokegeneric bridge_table[paramCount]
else
  invokegeneric_varags bridge_invoke_array
```

```
abstract class JavaBridgeMH extends MethodHandle {
  Object invoke() { ... }
  Object invoke(Object o1) { ... }
  Object invoke(Object o1, Object o2) { ... }
  Object invoke(Object o1, Object o2, Object o3) { ... }
  ...
  Object invoke(Object... vargs) { ... }
}
```

MethodHandle Invocation

invokeExact

check methodType => call stub
else WrongMethodTypeException

invokeGeneric

check methodType => call stub
else check not varargs => convert types + call stub
else => varargs wrap + convert types + call stub
else WrongMethodTypeException

invokeWithArguments (native)

asSpread() + invokeGeneric

IndyDroid

Short intro to JSR 292

An example

Dalvik

- dx, dexopt, DEX
- add method handles
- add invokedynamic

Future & Conclusion

CallSite & Invokedynamic

Invokedynamic

```
if (target[id] == null)
    CAS(&target[id], null, initBSM(...))
invokeexact target[id](args...)
```

Side table of target method handles

- each invokedynamic has a class local index
- load target is a volatile read !
- bootstrap method initialization:
  - call BSM with invokegeneric
  - inject class + target index into CallSite for setTarget,
    - MutableCallSite (normal write), VolatileCallSite (volatile write)
  - ends with a CAS(null, newTarget)

Schedule & Future

Where we are ?

- dx, dexopt, DEX — we are here :(
- add method handles
- add invokedynamic

basic interpreter targeted for mid-march

JIT ??

- Method handle reflection to JIT IR ?
- perf counters ? & Guards ?