Generating Efficient Code for Lambdas and Function Types

Fredrik Öhrström
Principal Member of Technical Staff
ORACLE JRockit+Hotspot

You Have to Grok the MethodHandle

MethodHandle

- Is an opaque reference to a method
- Embeds a type to verify that a call is safe at runtime
- Can box and unbox arguments at runtime
- Can be acquired using ldc in the bytecode
- A receiver argument can be bound

Example 1 (warning deprecated syntax)

```
public class Test1 {
 static volatile MethodHandle mh = Test1#calc(int,Object);

 public static void main(String... args) throws Throwable {
    String a = mh.<String>invoke(1, (Object)"A");
    Object b = mh.<Object>invoke(2, "B");
    Object c = mh.<Object>invoke((Object)3, 3);
    System.out.println(""+a+","+b+","+c);
 }

 static String calc(int x, Object o) {
    return ""+x+"="+o.hashCode();
 }
}
```

Example 1 (warning deprecated syntax)

```
public class Test1 {
 static volatile MethodHandle mh = Test1#calc(int,Object);

 public static void main(String... args) throws Throwable {
    String a = mh.<String>invoke(1, (Object)"A");
    Object b = mh.<Object>invoke(2, "B");
    Object c = mh.<Object>invoke((Object)3, 3);
    System.out.println(""+a+","+b+","+c);
 }

 static String calc(int x, Object o) {
    return ""+x+"="+o.hashCode();
 }
}
```

Will print 1=65,2=66,3=3

Several Optimization Opportunities

- All arguments match exactly at callsite
- Co/contra variant references, exact primitives
- Inlining boxing code
- And more...

How fast is invokevirtual?

```
mov rax <- [objptr]    // vtable & other meta data
mov rcx <- [rax+1024] // load target from vtable
call rcx
```

How fast is invokevirtual?

```
mov rax <- [objptr]   // vtable & other meta data
mov rcx <- [rax+1024] // load target from vtable
call rcx
```

2 loads

How fast is invokeinterface?

Depends on the implementation:

- JRockit uses a constant time lookup
- Hotspot uses an iteration over the implemented interfaces

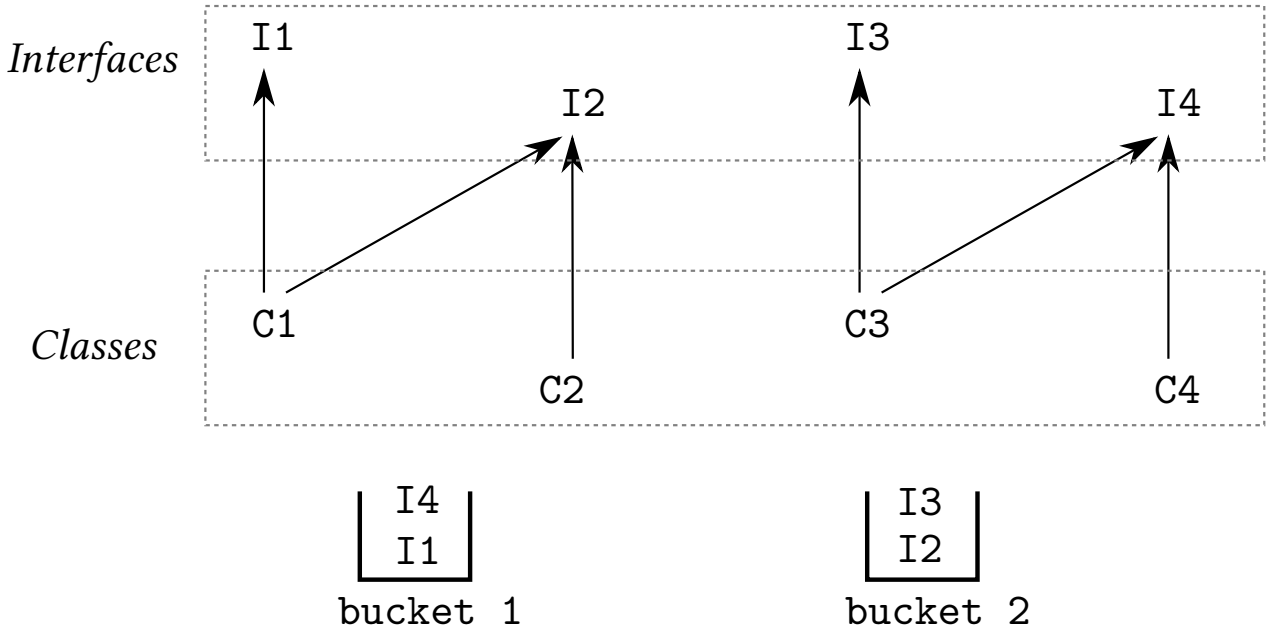
How fast is invokeinterface?

Depends on the implementation:

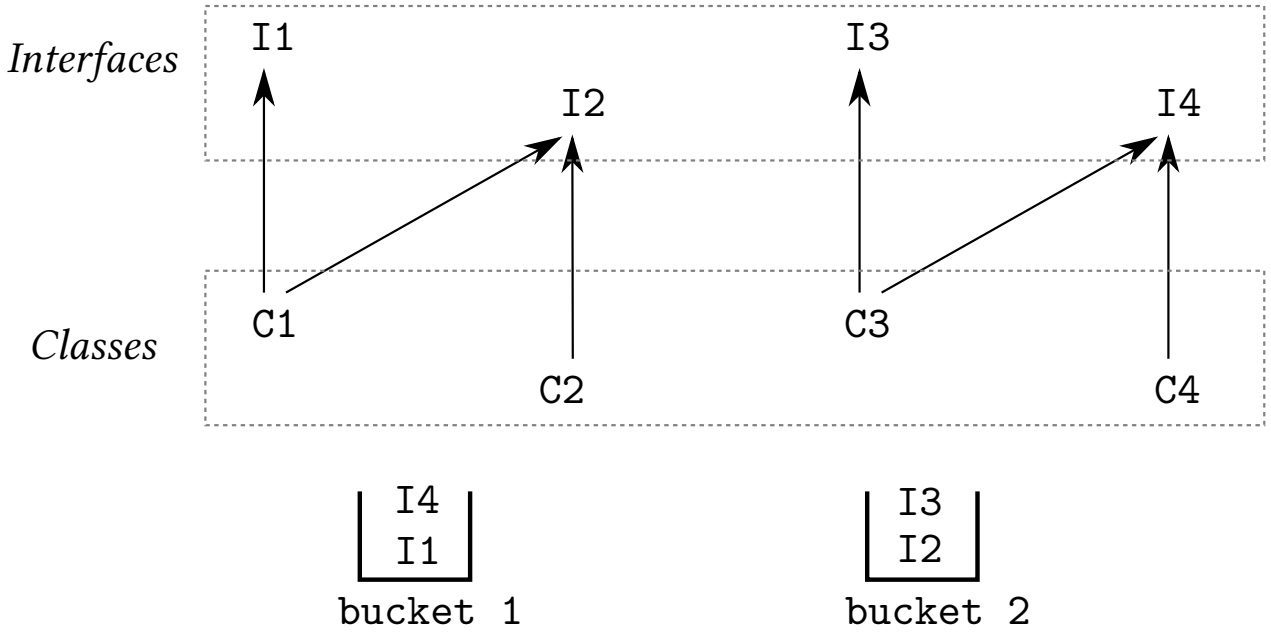
- JRockit uses a constant time lookup
- Hotspot uses an iteration over the implemented interfaces

The invokevirtual solution is infeasible due to excessive memory usage. Some kind of memory-speed tradeoff is needed.

Compressing the Interface Tables

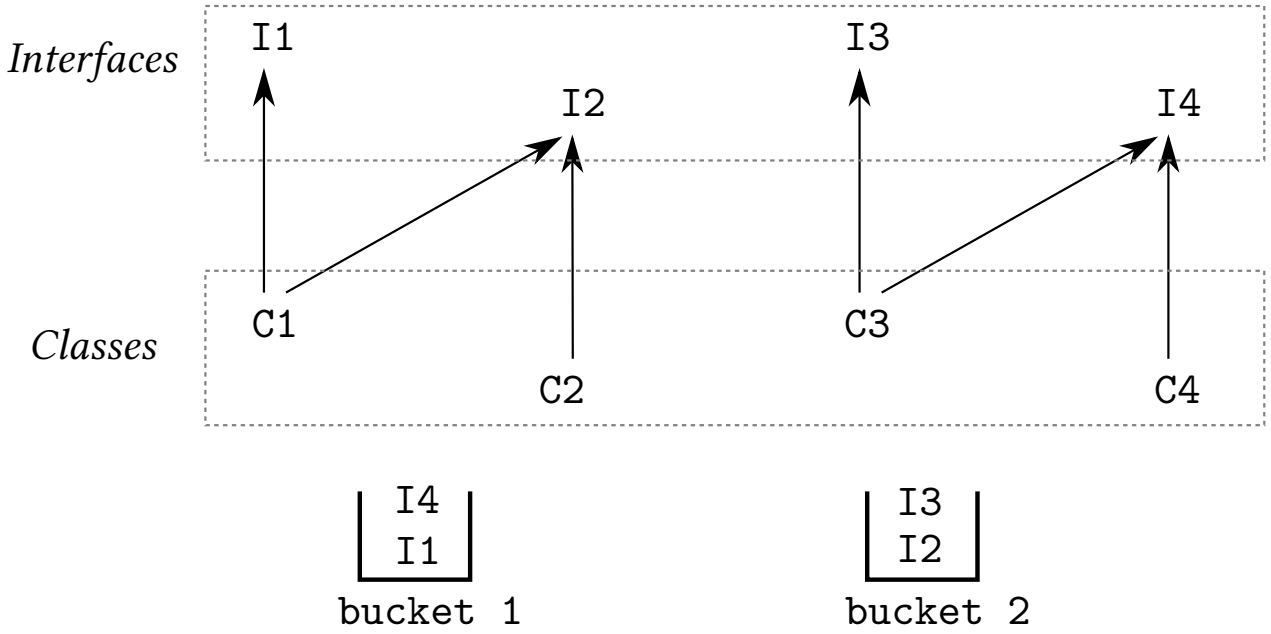


Compressing the Interface Tables



Graph coloring, fast approximation algorithms exist.

Compressing the Interface Tables



Graph coloring, fast approximation algorithms exist.
Can be partially updated until recompression is needed.

Interface instanceof

```
mov rax <- [objptr]     // vtable & other meta data
mov rbx <- [0x123456]   // interface bucket nr
mov rdx <- [rax+32]     // buckets from meta data
mov rcx <- [rdx+rbx*8] // bucket content
cmp rcx, 0x123456       // is instance of?
```

Interface instanceof

```
mov rax <- [objptr]     // vtable & other meta data
mov rbx <- [0x123456]   // interface bucket nr
mov rdx <- [rax+32]     // buckets from meta data
mov rcx <- [rdx+rbx*8] // bucket content
cmp rcx, 0x123456       // is instance of?
```

4 loads needed for instanceof check.

Interface instanceof

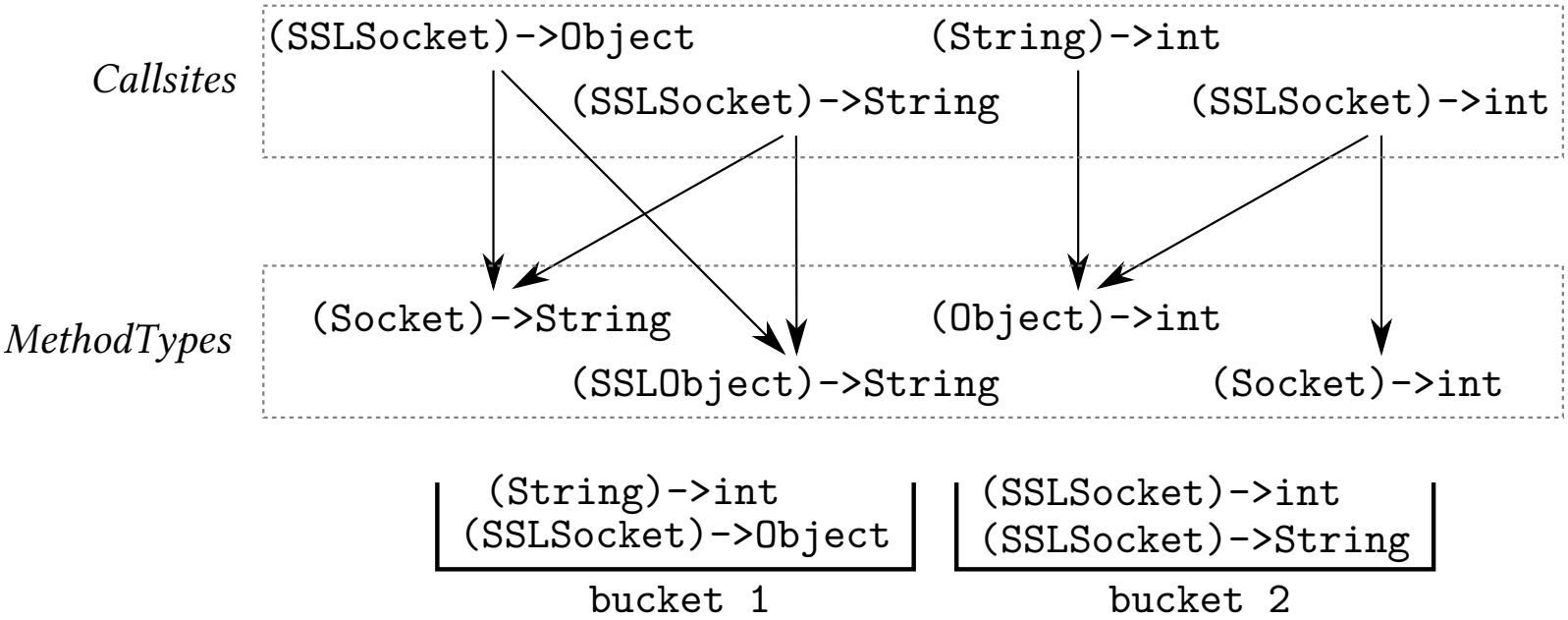
```
mov rax <- [objptr]     // vtable & other meta data
mov rbx <- [0x123456]   // interface bucket nr
mov rdx <- [rax+32]     // buckets from meta data
mov rcx <- [rdx+rbx*8] // bucket content
cmp rcx, 0x123456       // is instance of?
```

4 loads needed for instanceof check.

An additional 1-2 loads to actually perform call.

I.e. 5-6 loads to perform a constant time interface call.

Compressing the Co/Contra Variance Lookup Tables



MethodHandle invoke

```
mov rax <- [mhptr+16]   // load method type
mov r10 <- [mhptr+24]   // target (ndx/addr bou)
cmp rax, 0x654321       // exact match?
je  direct_call_ok
mov rbx <- [0x123456]   // call site type bucket nr
mov rdx <- [rax+32]     // buckets from method type
mov rcx <- [rdx+rbx*8] // bucket content
cmp rcx, 0x123456       // invokable by?
je direct_call_ok
...generic case...
```

MethodHandle invoke

```
mov rax <- [mhptr+16]   // load method type
mov r10 <- [mhptr+24]   // target (ndx/addr bou)
cmp rax, 0x654321       // exact match?
je  direct_call_ok
mov rbx <- [0x123456]   // call site type bucket nr
mov rdx <- [rax+32]     // buckets from method type
mov rcx <- [rdx+rbx*8] // bucket content
cmp rcx, 0x123456       // invokable by?
je direct_call_ok
...generic case...
```

2 loads(sta,exa) 3 loads(sta,exa,bou)

MethodHandle invoke

```
mov rax <- [mhptr+16]   // load method type
mov r10 <- [mhptr+24]   // target (ndx/addr bou)
cmp rax, 0x654321       // exact match?
je  direct_call_ok
mov rbx <- [0x123456]   // call site type bucket nr
mov rdx <- [rax+32]     // buckets from method type
mov rcx <- [rdx+rbx*8] // bucket content
cmp rcx, 0x123456       // invokable by?
je direct_call_ok
...generic case...
```

2 loads(sta,exa) 3 loads(sta,exa,bou)
4 loads(virt,exa) 5 loads(sta,var)

MethodHandle invoke

```
mov rax <- [mhptr+16]   // load method type
mov r10 <- [mhptr+24]   // target (ndx/addr bou)
cmp rax, 0x654321       // exact match?
je  direct_call_ok
mov rbx <- [0x123456]   // call site type bucket nr
mov rdx <- [rax+32]     // buckets from method type
mov rcx <- [rdx+rbx*8] // bucket content
cmp rcx, 0x123456       // invokable by?
je direct_call_ok
...generic case...
```

2 loads(sta,exa) 3 loads(sta,exa,bou)
4 loads(virt,exa) 5 loads(sta,var)
6 loads(sta,var,bou) 7 loads(virt,var)

The Generic Case (static target, 64bit hardware)

n_loads=(6+4*instanceof+2*unboxings+(boxings>0?2:0))
n_writes=(1+2*boxings+(boxings>0?1:0))

```
Object c = mh.<Object>invoke((Integer)3, 3);

Callsite  (Integer,int)->Object
targeting (int,Object)->String
```

The Generic Case (static target, 64bit hardware)

n_loads=(6+4*instanceof+2*unboxings+(boxings>0?2:0))
n_writes=(1+2*boxings+(boxings>0?1:0))

```
Object c = mh.<Object>invoke((Integer)3, 3);

Callsite  (Integer,int)->Object
targeting (int,Object)->String
```

10 loads and 4 writes (assuming enough space in TLA)

The Generic Case (static target, 64bit hardware)

n_loads=(6+4*instanceof+2*unboxings+(boxings>0?2:0))
n_writes=(1+2*boxings+(boxings>0?1:0))

```
Object c = mh.<Object>invoke((Integer)3, 3);

Callsite  (Integer,int)->Object
targeting (int,Object)->String
```

10 loads and 4 writes (assuming enough space in TLA)
Fun fact: It can be faster to box than to cast!

Sounds good, but!

Unfortunately, these are all premature optimizations!

Sounds good, but!

Unfortunately, these are all premature optimizations!
In any reasonable and optimized code the target will be inlined!

Sounds good, but!

Unfortunately, these are all premature optimizations!
In any reasonable and optimized code the target will be inlined!
Hey! I thought that MethodHandles and inlining were orthogonal?

Sounds good, but!

Unfortunately, these are all premature optimizations!
In any reasonable and optimized code the target will be inlined!
Hey! I thought that MethodHandles and inlining were orthogonal?
Not at all, in fact the primary use case is to inline!

Example 2 (warning deprecated syntax)

```
public class Test2 {
 static int test() throws Throwable {
   MethodHandle mh = Test2#calc(int,Object);
   int a = mh.<int>invoke(1, (Object)"A");
   int b = mh.<int>invoke(2, "B");
   Integer c = mh.<Integer>invoke((Integer)3, 3);
   return a+b+c;
 }

 static int calc(int x, Object o) {
   return x+o.hashCode();
 }
}
```

Example 2 (warning deprecated syntax)

```
public class Test2 {
 static int test() throws Throwable {
   MethodHandle mh = Test2#calc(int,Object);
   int a = mh.<int>invoke(1, (Object)"A");
   int b = mh.<int>invoke(2, "B");
   Integer c = mh.<Integer>invoke((Integer)3, 3);
   return a+b+c;
 }

 static int calc(int x, Object o) {
   return x+o.hashCode();
 }
}
```

Demo!

Example 2 (warning deprecated syntax)

```
public class Test2 {
 static int test() throws Throwable {
   MethodHandle mh = Test2#calc(int,Object);
   int a = mh.<int>invoke(1, (Object)"A");
   int b = mh.<int>invoke(2, "B");
   Integer c = mh.<Integer>invoke((Integer)3, 3);
   return a+b+c;
 }

 static int calc(int x, Object o) {
   return x+o.hashCode();
 }
}
```

Demo!

Optimized into a single return 140 instruction!

Integration With Generics

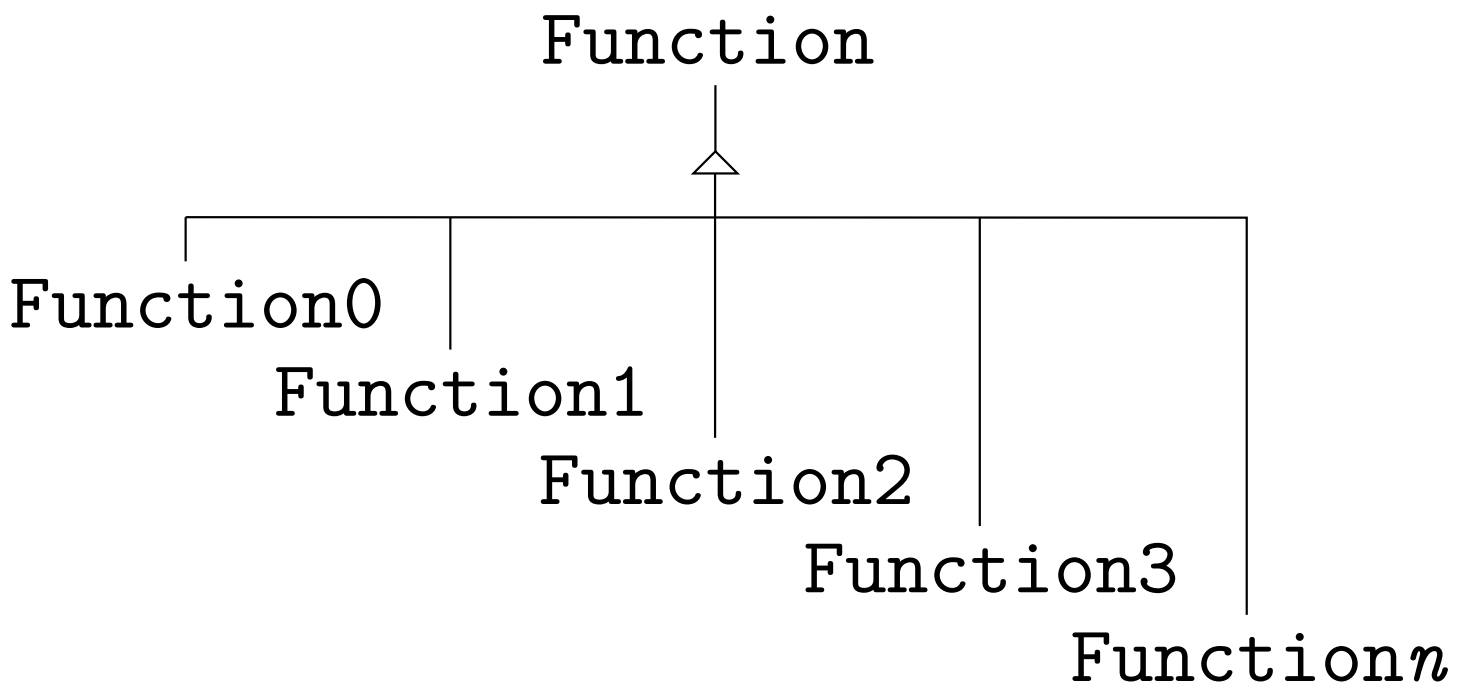
The gordian knot that needs to be untangled is how to integrate with generics and achive the same performance as if we had access to primitives.

Integration With Generics

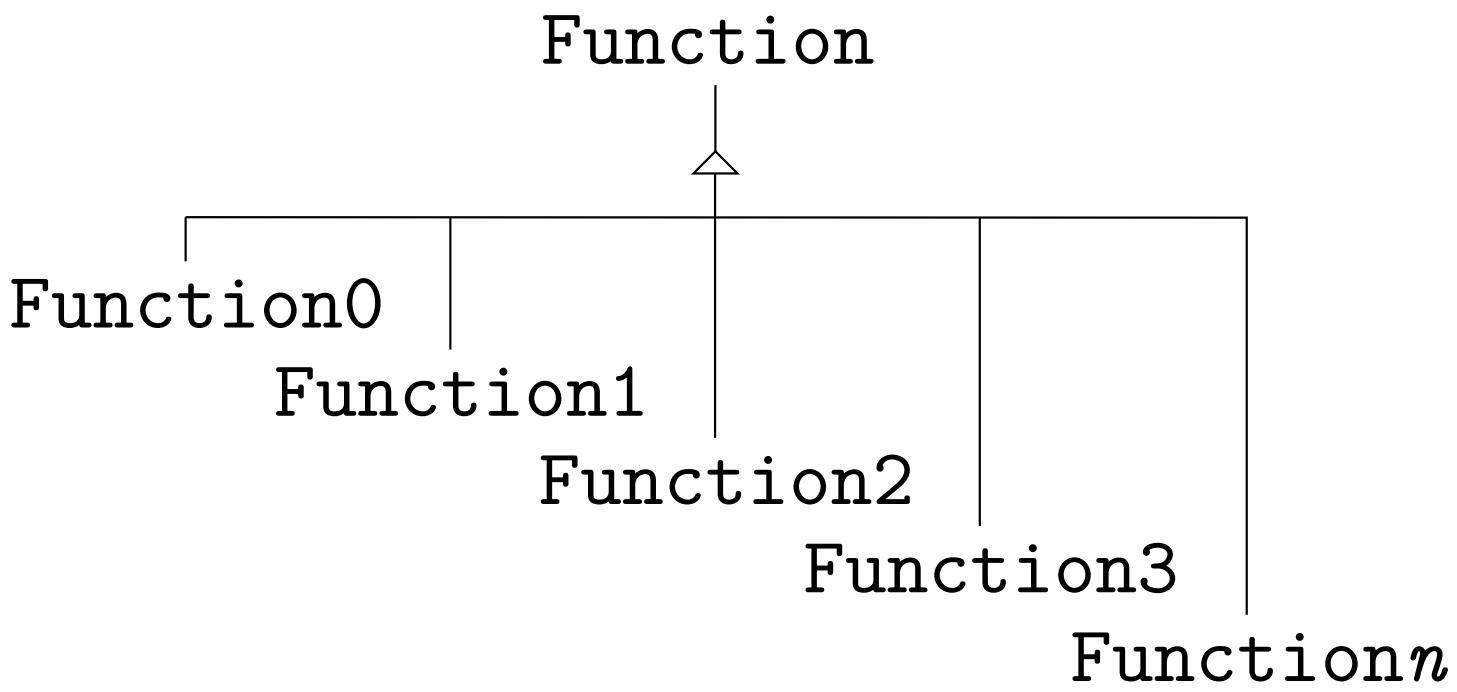
The gordian knot that needs to be untangled is how to integrate with generics and achive the same performance as if we had access to primitives.
For example: the BGGA implementation has its own system to generate interfaces for each permutation of objects and primitives. The exact type of the objects are then controlled by generics, but the primitives are locked in place.

Rename the MethodHandle into Function!

Specialize The Function Object



Specialize The Function Object



This is only necessary to workaround the lack of variadic generics.

Example

```
public final class Function1<R,E,A> extends Function
{
  R invoke(A a) throws E { return null; };
}
```

Transform The Source

```
Function()->void alfa;
```

Transform The Source

```
Function()->void alfa;
```

```
Function0<? extends Void, ? extends Nothing> alfa;
```

Transform The Source

```
Function()->void alfa;

Function0<? extends Void, ? extends Nothing> alfa;

Function(int,String)->Object beta;
```

Transform The Source

```
Function()->void alfa;

Function0<? extends Void, ? extends Nothing> alfa;

Function(int,String)->Object beta;

Function2<? extends Object, ? extends Nothing,
          ? super Integer, ? super String>     beta;
```

Transform The Source

```
Function()->void alfa;

Function0<? extends Void, ? extends Nothing> alfa;

Function(int,String)->Object beta;

Function2<? extends Object, ? extends Nothing,
          ? super Integer, ? super String>     beta;

Function(String)-NumberFormatException->long gamma;
```

Transform The Source

```
Function()->void alfa;

Function0<? extends Void, ? extends Nothing> alfa;

Function(int,String)->Object beta;

Function2<? extends Object, ? extends Nothing,
          ? super Integer, ? super String>     beta;

Function(String)-NumberFormatException->long gamma;

Function1<? extends Long, ? extends NumberFormatException,
          ? super String> gamma;
```

The Gordian Cut!

At the callsite build a Function invoke and replace the wrapper class with its primitive:

```
alfa.invoke();
```

The Gordian Cut!

At the callsite build a Function invoke and replace the wrapper class with its primitive:

```
alfa.invoke();
((Function)alfa).<void>invoke();
```

The Gordian Cut!

At the callsite build a Function invoke and replace the wrapper class with its primitive:

```
alfa.invoke();
((Function)alfa).<void>invoke();
beta.invoke(4,"Hello");
```

The Gordian Cut!

At the callsite build a Function invoke and replace the wrapper class with its primitive:

```
alfa.invoke();
((Function)alfa).<void>invoke();
beta.invoke(4,"Hello");
((Function)beta).<Object>invoke((int)4, (String)"Hello");
```

The Gordian Cut!

At the callsite build a Function invoke and replace the wrapper class with its primitive:

```
alfa.invoke();
((Function)alfa).<void>invoke();
beta.invoke(4,"Hello");
((Function)beta).<Object>invoke((int)4, (String)"Hello");
long x = gamma.invoke("4711");
```

The Gordian Cut!

At the callsite build a Function invoke and replace the wrapper class with its primitive:

```
alfa.invoke();
((Function)alfa).<void>invoke();
beta.invoke(4,"Hello");
((Function)beta).<Object>invoke((int)4, (String)"Hello");
long x = gamma.invoke("4711");
long x = ((Function)gamma).<long>invoke((String)"4711");
```

The Gordian Cut!

At the callsite build a Function invoke and replace the wrapper class with its primitive:

```
alfa.invoke();
((Function)alfa).<void>invoke();
beta.invoke(4,"Hello");
((Function)beta).<Object>invoke((int)4, (String)"Hello");
long x = gamma.invoke("4711");
long x = ((Function)gamma).<long>invoke((String)"4711");
```

I.e. we cannot generate callsites with explicit Integer and Float, but nobody will care because the JVM will fix this for us!

The Gordian Cut!

At the callsite build a Function invoke and replace the wrapper class with its primitive:

```
alfa.invoke();
((Function)alfa).<void>invoke();
beta.invoke(4,"Hello");
((Function)beta).<Object>invoke((int)4, (String)"Hello");
long x = gamma.invoke("4711");
long x = ((Function)gamma).<long>invoke((String)"4711");
```

I.e. we cannot generate callsites with explicit Integer and Float, but nobody will care because the JVM will fix this for us!

As long as the callsite uses int and the target uses int, then the invocation will equivalent in speed to an interface call or faster. It does not matter that the specification has taken a round trip through generics.

Example 4 (Warning proposed syntax)

```
static <R,A,B> Function(B,A)->R swap(Function(A,B)->R f) {
  return a,b -> f.invoke(b,a);
}

int test(int a, int b) {
  Function(int,int)->int f = x,y -> x-y;
  return swap(f).invoke(a,b);
}
```

Example 4 (Warning proposed syntax)

```
static <R,A,B> Function(B,A)->R swap(Function(A,B)->R f) {
  return a,b -> f.invoke(b,a);
}

int test(int a, int b) {
  Function(int,int)->int f = x,y -> x-y;
  return swap(f).invoke(a,b);
}
```

test should compile into the equivalent of

```
int test(int a, int b) { return b-a; }
```

Example 3 (After transform)

```
static class C1{MethodHandle v1;C1(MethodHandle v){v1=v;}
Object f1(Object a, Object b) throws Throwable
{ return v1.<Object>invoke(b,a); } }

static<R,E,A,B>Function2<R,E,B,A>swap(Function2<R,E,A,B>f)
throws Throwable { C1 c1 = new C1(f);
  return MethodHandles.asF2(c1#f1(Object,Object));
}

static int f2(int x, int y) { return x-y; }

static int test(int a, int b) throws Throwable {
 Function2<Integer,Nothing,Integer,Integer> f =
 MethodHandles.asF2(Test3#f2(int,int));
 return ((MethodHandle)swap(f)).<int>invoke((int)a,(int)b);
}
```

Result

Test optimized into a single subtraction instruction!

Conclusions

- We can achieve high performance function types without reifying generics.
- It can be compatible with the reified generics of the future.

Workshop starters

- How do we implement the function type carrier, is-a or has-a?
- How do we express varags in function types?
- Bring your lambda!