# Method Handle Graph JIT Compilation

**Shijie Xu, David Bremner**
University of New Brunswick, IBM Canada
Faculty of Computer Science
{Sxu3, bremner}@unb.ca

## Motivation

A Method Handle Graph (MHG) is a graph structure that transforms a method invocation at a dynamic call site to a number of target method invocations. Although an MHG, together with the JVM instruction, invokedynamic, resolves 'pain points' (e.g., polluted profiles and failed inlining) when implementing dynamic JVM language implementations, it:

1) introduces a cost of graph traversal to resolve target methods when a dynamic invocation is made; and

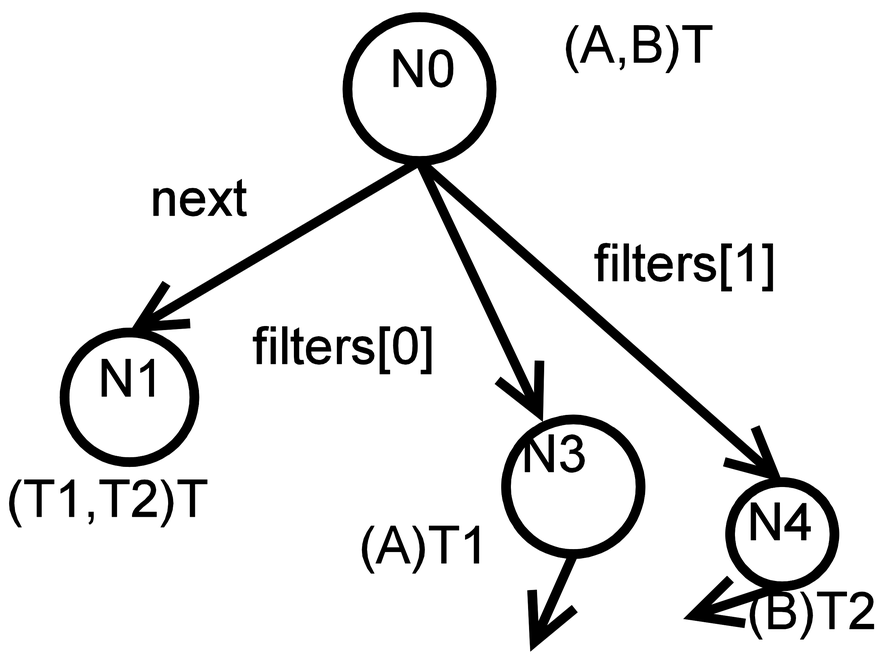
2) adds more overhead to Just-In-Time (JIT) compile a graph into native code due to a number of homogeneous method handle nodes in the MHG.

## Background

An example of *FilterArgument* method handle is:

```
MethodHandle [ ]filters;
MethodHandle  next;
T  filterArguments (A a , B b ) {
        return (T) next.invokeExact (
                ( T1 ) filters[0].invokeExact( a ) ,
                ( T2 ) filters[1].invokeExact( b )) ;
}
```

The corresponding MHG is:




## Solution

MHG JIT is a dynamic compilation that converts an MHG into another equivalent optimized bytecode version. We use inlining to concatenate multiple method handles of a graph into a single bytecode method.

Example for N0->N3 in the *FilterArgument* MHG

```
aload 0
getField filters[0]      //For simplification
aload 1
invokevirtual MH:invokExact(A)T1
astore 3

…
```

```
aload 0
getField filters[0]
aload 1
{
  astore 3; astore 4;
  //bytecodes of N3's target here and re-number variables
  ldc 6acdcb0c-705b-4426
  Invokestatic ConstCache.peek:(String;)Object
  Checkcast T1
  ….
}
astore 5
…
```