MethodHandle compilation pipeline

A detailed look at J9’s approach to MethodHandle compilation

Dan Heidinga, J9 Virtual Machine Team Lead
@DanHeidinga

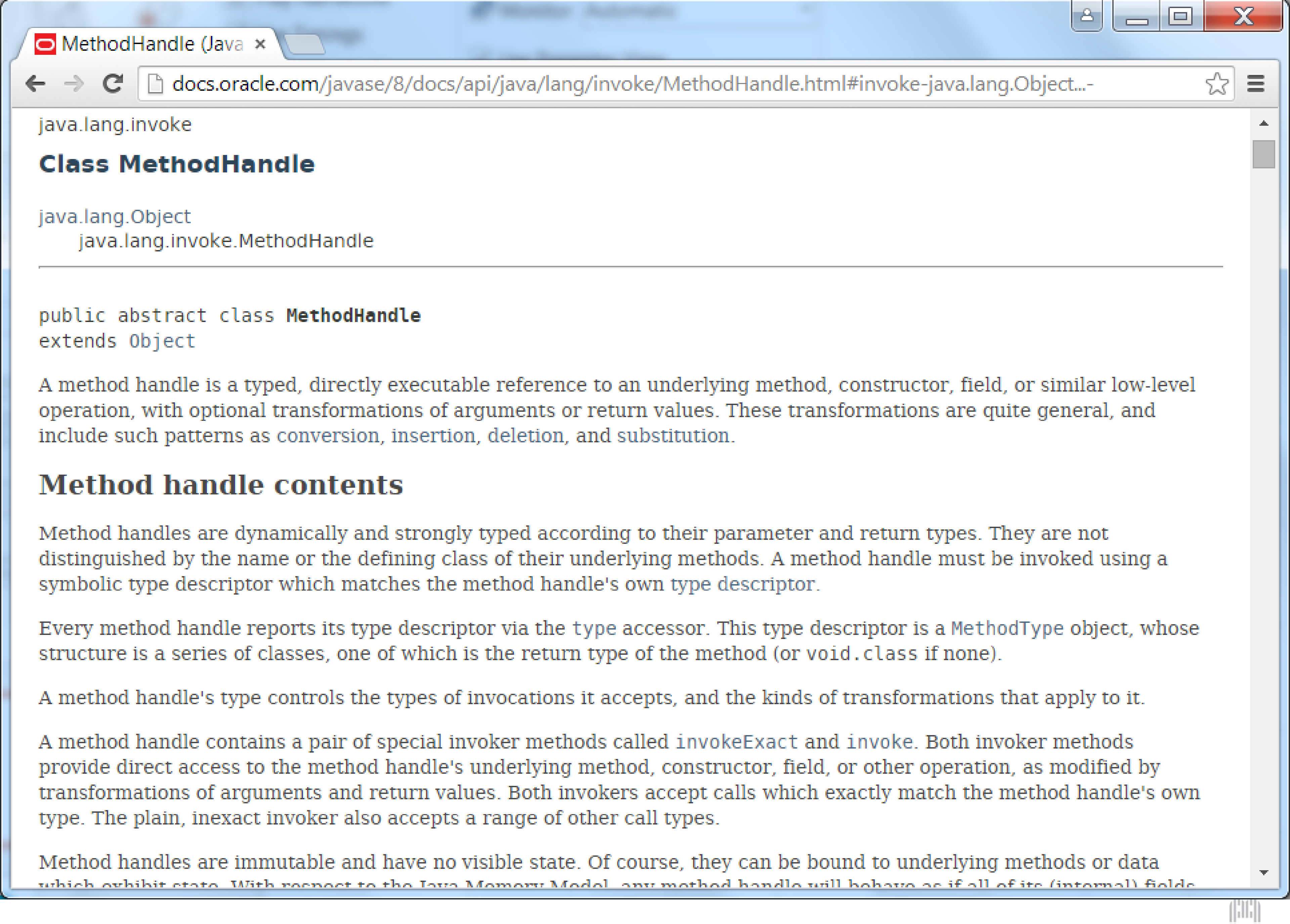
2015-02-02

Important disclaimers

- THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.
- WHILST EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.
- ALL PERFORMANCE DATA INCLUDED IN THIS PRESENTATION HAVE BEEN GATHERED IN A CONTROLLED ENVIRONMENT. YOUR OWN TEST RESULTS MAY VARY BASED ON HARDWARE, SOFTWARE OR INFRASTRUCTURE DIFFERENCES.
- ALL DATA INCLUDED IN THIS PRESENTATION ARE MEANT TO BE USED ONLY AS A GUIDE.
- IN ADDITION, THE INFORMATION CONTAINED IN THIS PRESENTATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM, WITHOUT NOTICE.
- IBM AND ITS AFFILIATED COMPANIES SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.
- NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:
  - CREATING ANY WARRANT OR REPRESENTATION FROM IBM, ITS AFFILIATED COMPANIES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS

Who am I?

- I've been involved with virtual machine development at IBM since 2007 and am now the J9 Virtual Machine Team Lead. J9 is IBM's independent implementation of the JVM.
- I've represented IBM on both the JSR 292 ('invokedynamic') and JSR 335 ('lambda') expert groups and lead J9's implementation of both JSRs.
- I also maintain the bytecode verifier and deal with various other parts of the runtime.


MethodHandle (Java
docs.oracle.com/javase/8/docs/api/java/lang/invoke/MethodHandle.html#invoke-java.lang.Object...-

java.lang.invoke

Class MethodHandle

java.lang.Object
    java.lang.invoke.MethodHandle

```
public abstract class MethodHandle
extends Object
```

A method handle is a typed, directly executable reference to an underlying method, constructor, field, or similar low-level operation, with optional transformations of arguments or return values. These transformations are quite general, and include such patterns as conversion, insertion, deletion, and substitution.

Method handle contents

Method handles are dynamically and strongly typed according to their parameter and return types. They are not distinguished by the name or the defining class of their underlying methods. A method handle must be invoked using a symbolic type descriptor which matches the method handle's own type descriptor.

Every method handle reports its type descriptor via the `type` accessor. This type descriptor is a `MethodType` object, whose structure is a series of classes, one of which is the return type of the method (or `void.class` if none).

A method handle's type controls the types of invocations it accepts, and the kinds of transformations that apply to it.

A method handle contains a pair of special invoker methods called `invokeExact` and `invoke`. Both invoker methods provide direct access to the method handle's underlying method, constructor, field, or other operation, as modified by transformations of arguments and return values. Both invokers accept calls which exactly match the method handle's own type. The plain, inexact invoker also accepts a range of other call types.

Method handles are immutable and have no visible state. Of course, they can be bound to underlying methods or data which exhibit state. With respect to the Java Memory Model, any method handle will behave as if all of its (internal) fields

J9’s MethodHandle hierarchy

- Original prototype had 1 class: MethodHandle
  - “kind” field to determine which operation
  - “type” field to hold the MethodType
  - “vmSlot” field to hold the address, offset, vtable or itable index
- Grab bag of data necessary to support field access and method sends
- 2 major problems with this approach

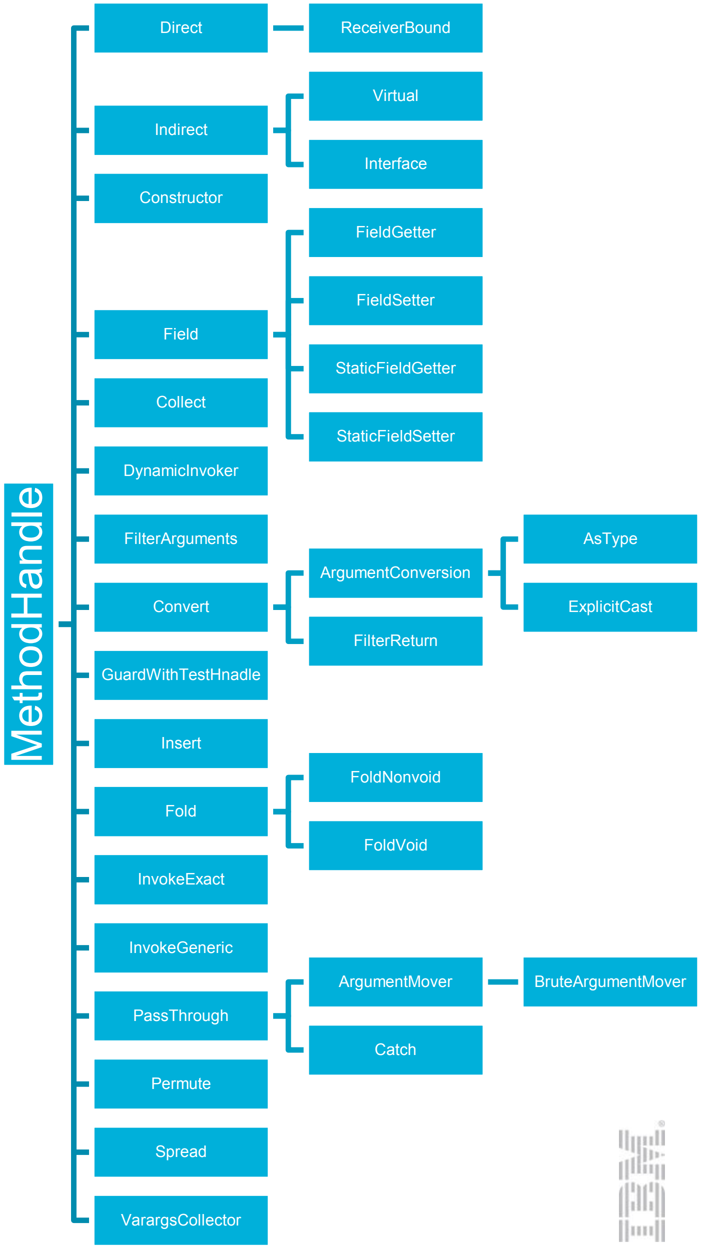
MethodHandle

MethodHandle

http://commons.wikimedia.org/wiki/File:MudBall03.jpg
Public domain.

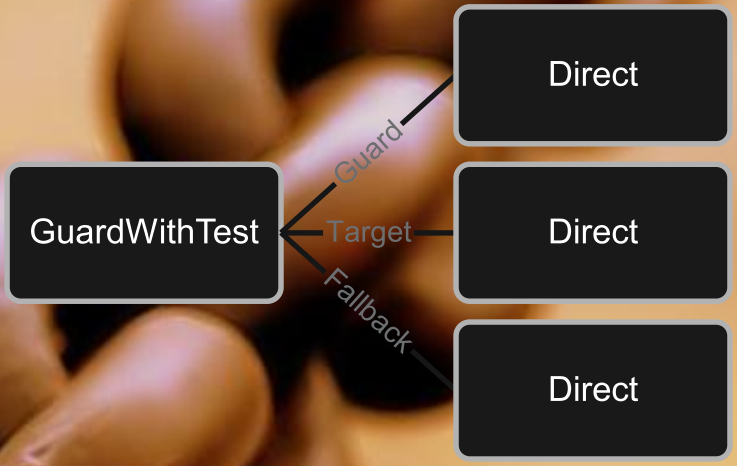
J9’s MethodHandle hierarchy

- Hierarchy that separates each MH kind into its own class
- Each MH subclass describes the data needed by the MH
- JITs look at the class rather than the ‘kind’ instance field
  - Provides a place to put specialized behaviour



"Broad chain closeup" by Toni Lozano - http://www.flickr.com/photos/quiero-un-pantano/176909201.
Licensed under CC BY 2.0 via Wikimedia Commons -
http://commons.wikimedia.org/wiki/File:Broad_chain_closeup.jpg#mediaviewer/File:Broad_chain_closeup.jpg

MethodHandle chains



GuardWithTest

Guard

Target

Fallback

Direct

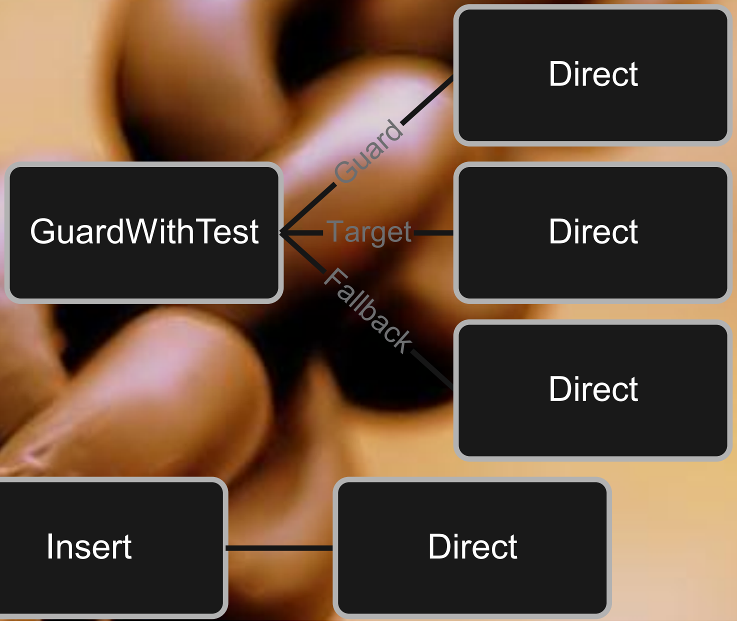
Direct

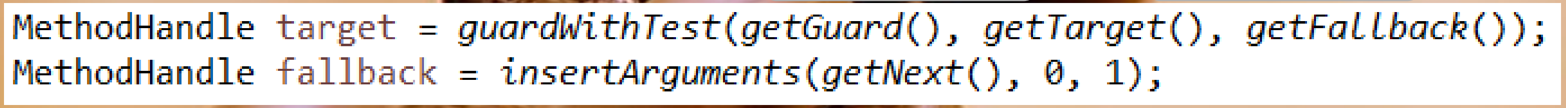
Direct

```
MethodHandle target = guardWithTest(getGuard(), getTarget(), getFallback());
```

"Broad chain closeup" by Toni Lozano - http://www.flickr.com/photos/quiero-un-pantano/176909201. Licensed under CC BY 2.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Broad_chain_closeup.jpg#mediaviewer/File:Broad_chain_closeup.jpg

MethodHandle chains

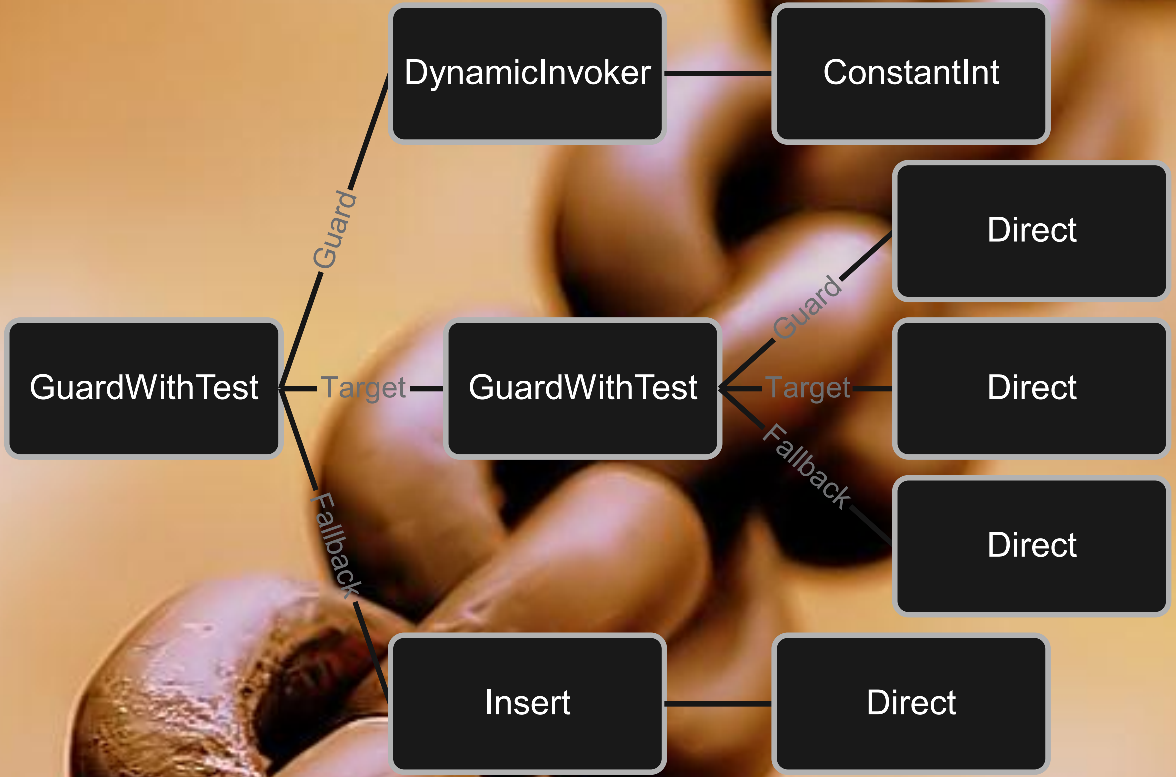


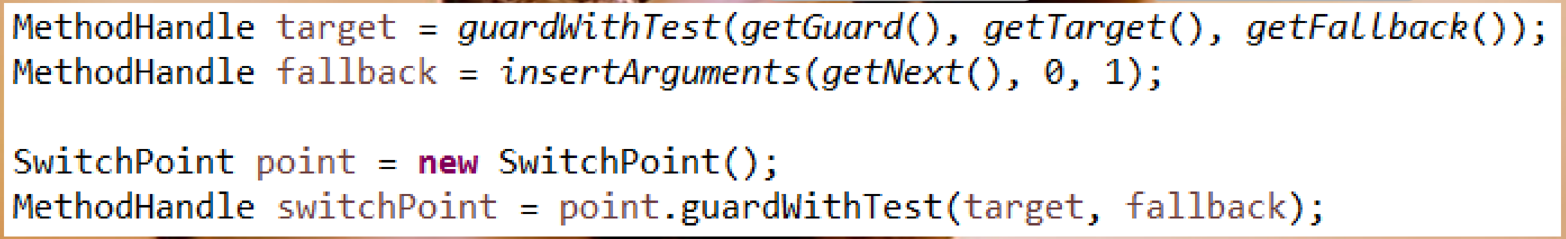


```
MethodHandle target = guardWithTest(getGuard(), getTarget(), getFallback());
MethodHandle fallback = insertArguments(getNext(), 0, 1);
```

"Broad chain closeup" by Toni Lozano - http://www.flickr.com/photos/quiero-un-pantano/176909201. Licensed under CC BY 2.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Broad_chain_closeup.jpg#mediaviewer/File:Broad_chain_closeup.jpg

MethodHandle chains

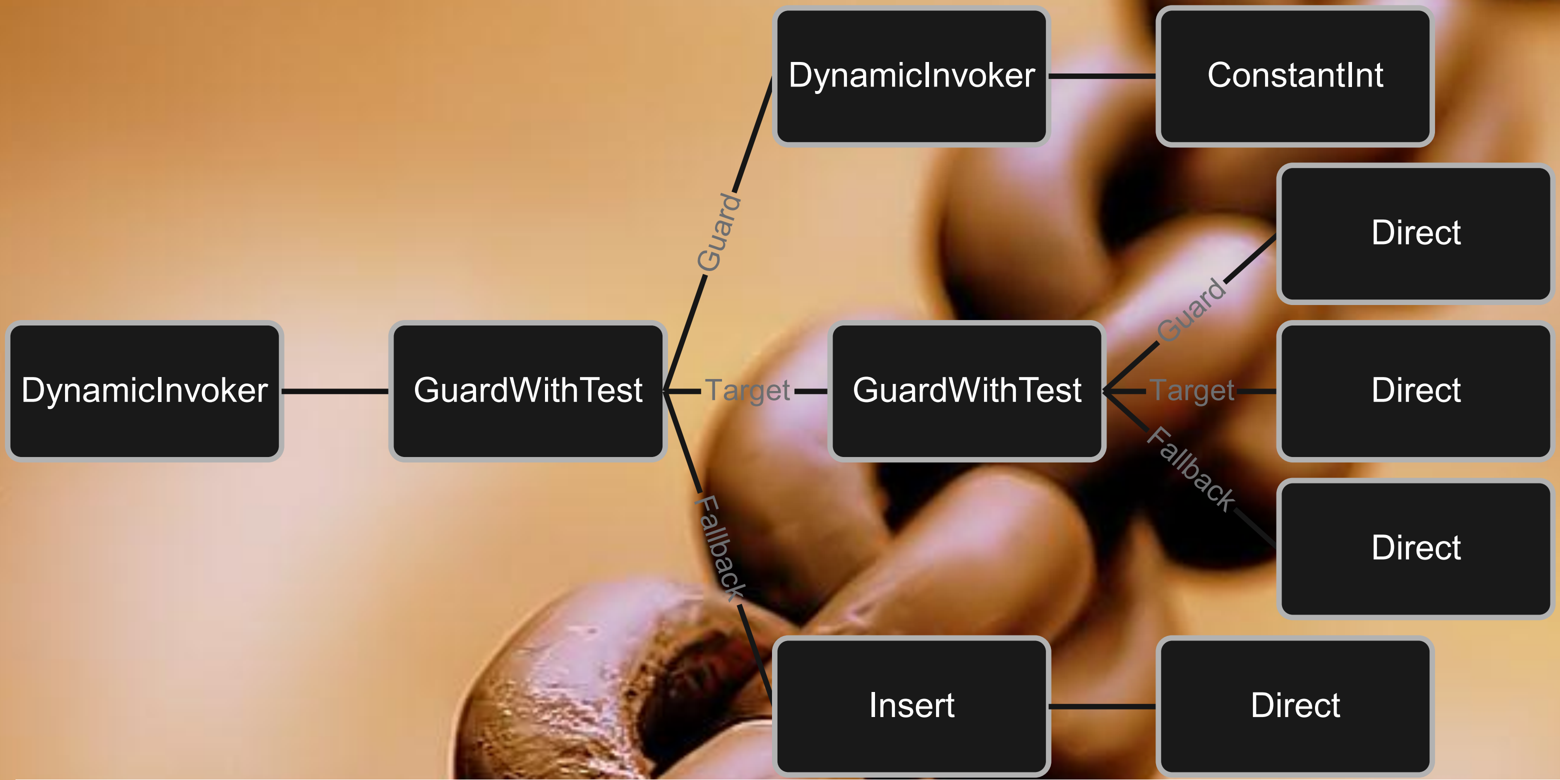


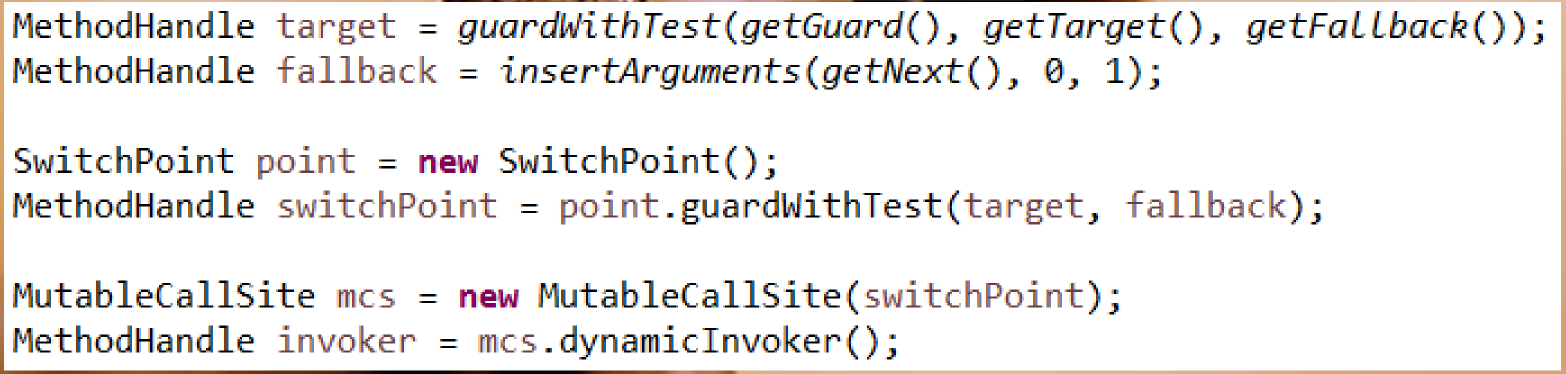


```
MethodHandle target = guardWithTest(getGuard(), getTarget(), getFallback());
MethodHandle fallback = insertArguments(getNext(), 0, 1);

SwitchPoint point = new SwitchPoint();
MethodHandle switchPoint = point.guardWithTest(target, fallback);
```

"Broad chain closeup" by Toni Lozano - http://www.flickr.com/photos/quiero-un-pantano/176909201. Licensed under CC BY 2.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Broad_chain_closeup.jpg#mediaviewer/File:Broad_chain_closeup.jpg

MethodHandle chains

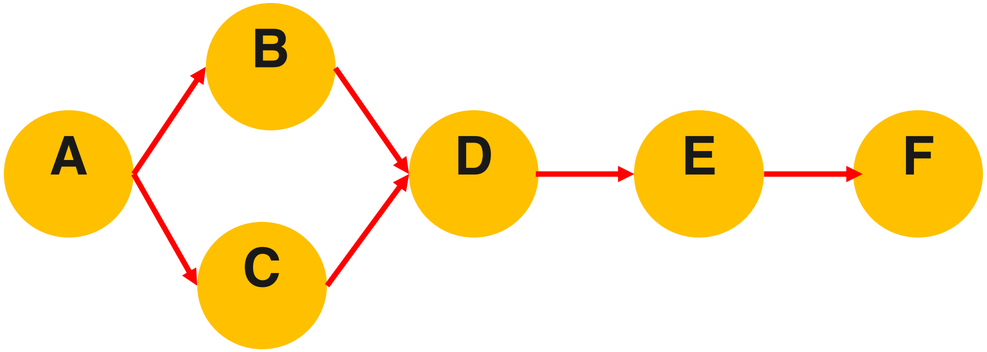




```
MethodHandle target = guardWithTest(getGuard(), getTarget(), getFallback());
MethodHandle fallback = insertArguments(getNext(), 0, 1);

SwitchPoint point = new SwitchPoint();
MethodHandle switchPoint = point.guardWithTest(target, fallback);

MutableCallSite mcs = new MutableCallSite(switchPoint);
MethodHandle invoker = mcs.dynamicInvoker();
```

"Broad chain closeup" by Toni Lozano - http://www.flickr.com/photos/quiero-un-pantano/176909201. Licensed under CC BY 2.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Broad_chain_closeup.jpg#mediaviewer/File:Broad_chain_closeup.jpg

(Mostly) tail recursive MH interpreter



1	**1**	**1**	**2**
2	2	2	3
3	3	3	1
4	4	D	E
A	C		

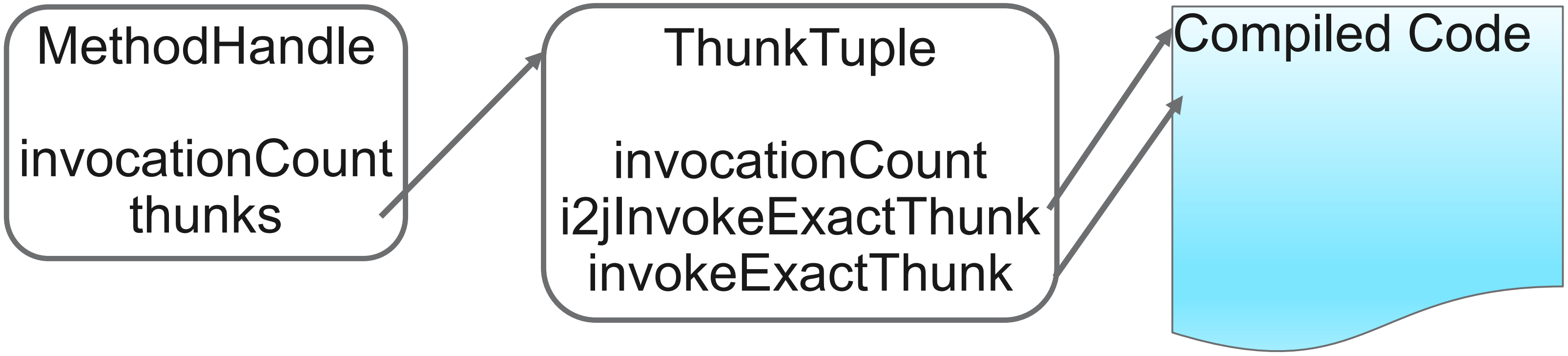
• • •

ThunkTuples

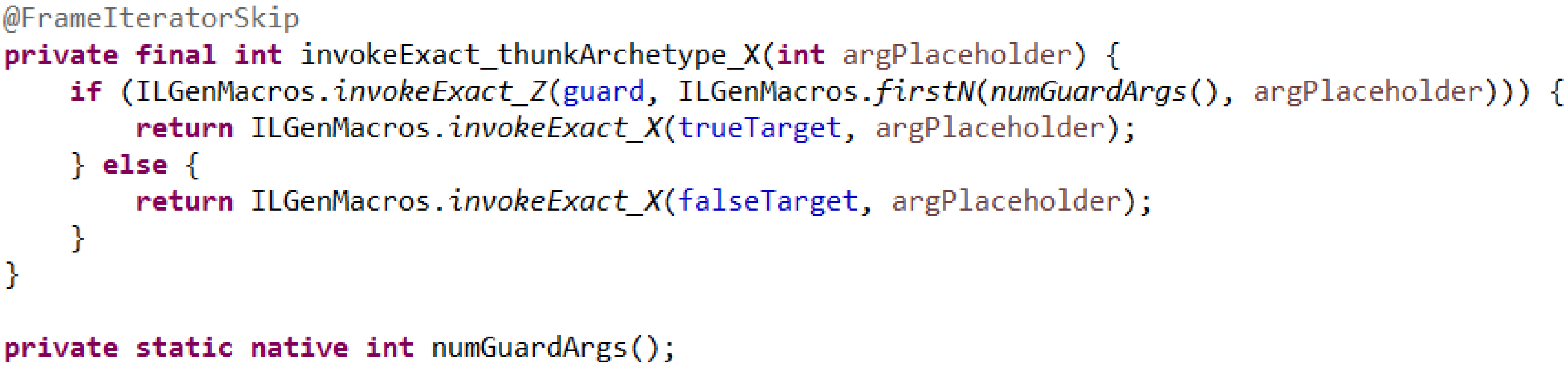
- Every MethodHandle has a ThunkTuple.

- ThunkTuples hold onto the compiled code for the MethodHandle
  - i2jInvokeExactThunk: interpreter to JIT entrypoint
  - invokeExactThunk: JIT to JIT entrypoint

- Each ThunkTuple is generated from a bytecode template for the MethodHandle subclass



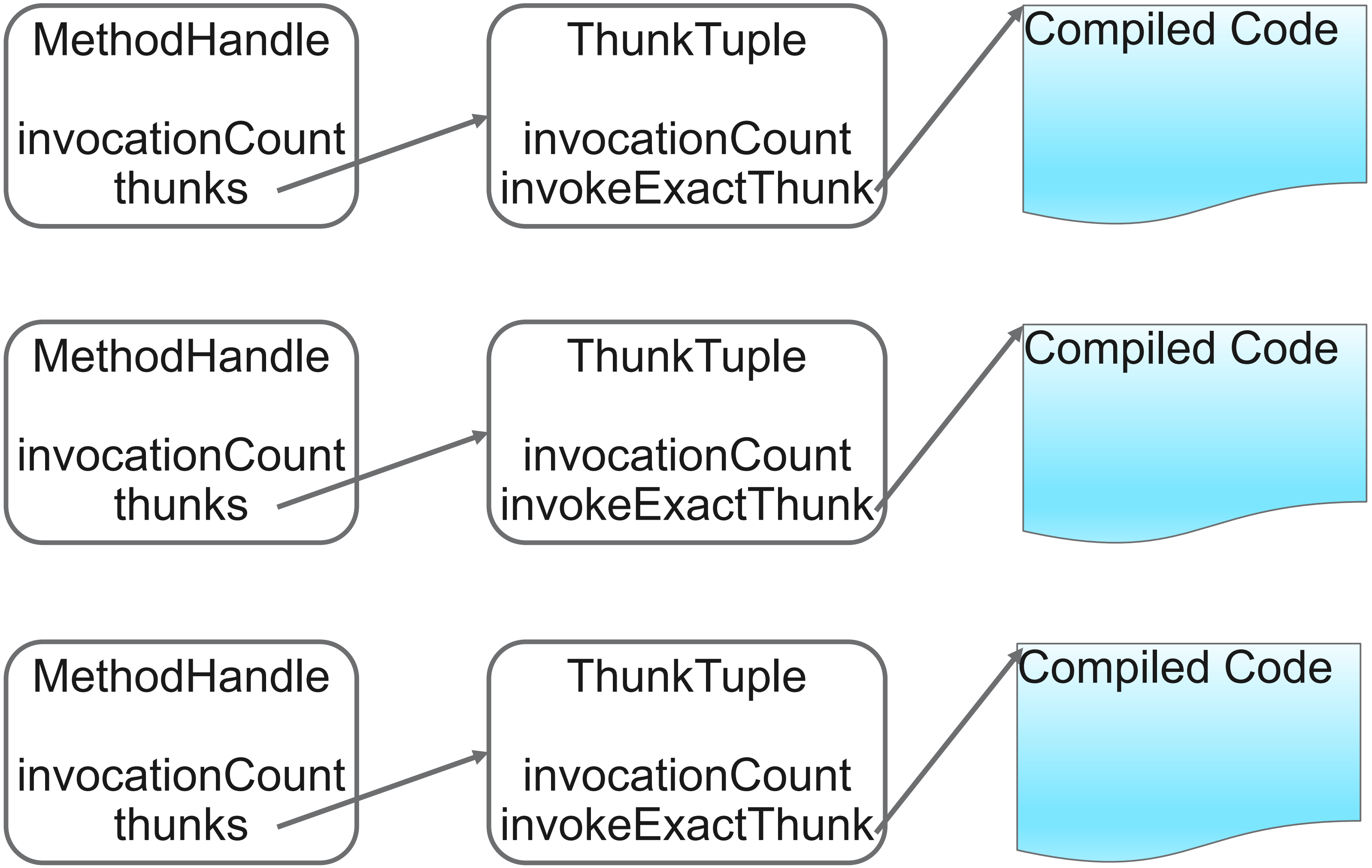
ThunkArchetypes: MethodHandle templates



```
@FrameIteratorSkip
private final int invokeExact_thunkArchetype_X(int argPlaceholder) {
    if (ILGenMacros.invokeExact_Z(guard, ILGenMacros.firstN(numGuardArgs(), argPlaceholder))) {
        return ILGenMacros.invokeExact_X(trueTarget, argPlaceholder);
    } else {
        return ILGenMacros.invokeExact_X(falseTarget, argPlaceholder);
    }
}

private static native int numGuardArgs();
```

- Signatures are written in terms of ‘int’ and edited at compile time
- Compile time macros are used to further specialize the code.
  – ‘numGuardArgs()’ determines how many arguments are passed to the guard handle
  – ILGenMacros.* are used to do signature editing, argument pushing and popping, etc
- This the MH equivalent of compiling a single ‘invokevirtual’ instruction
  – Specialized just enough to get out of the interpreter and into compiled code

But that’s a lot of duplicate code!



MethodHandle

invocationCount
thunks

ThunkTuple

invocationCount
invokeExactThunk

Compiled Code

MethodHandle

invocationCount
thunks

ThunkTuple

invocationCount
invokeExactThunk

Compiled Code

MethodHandle

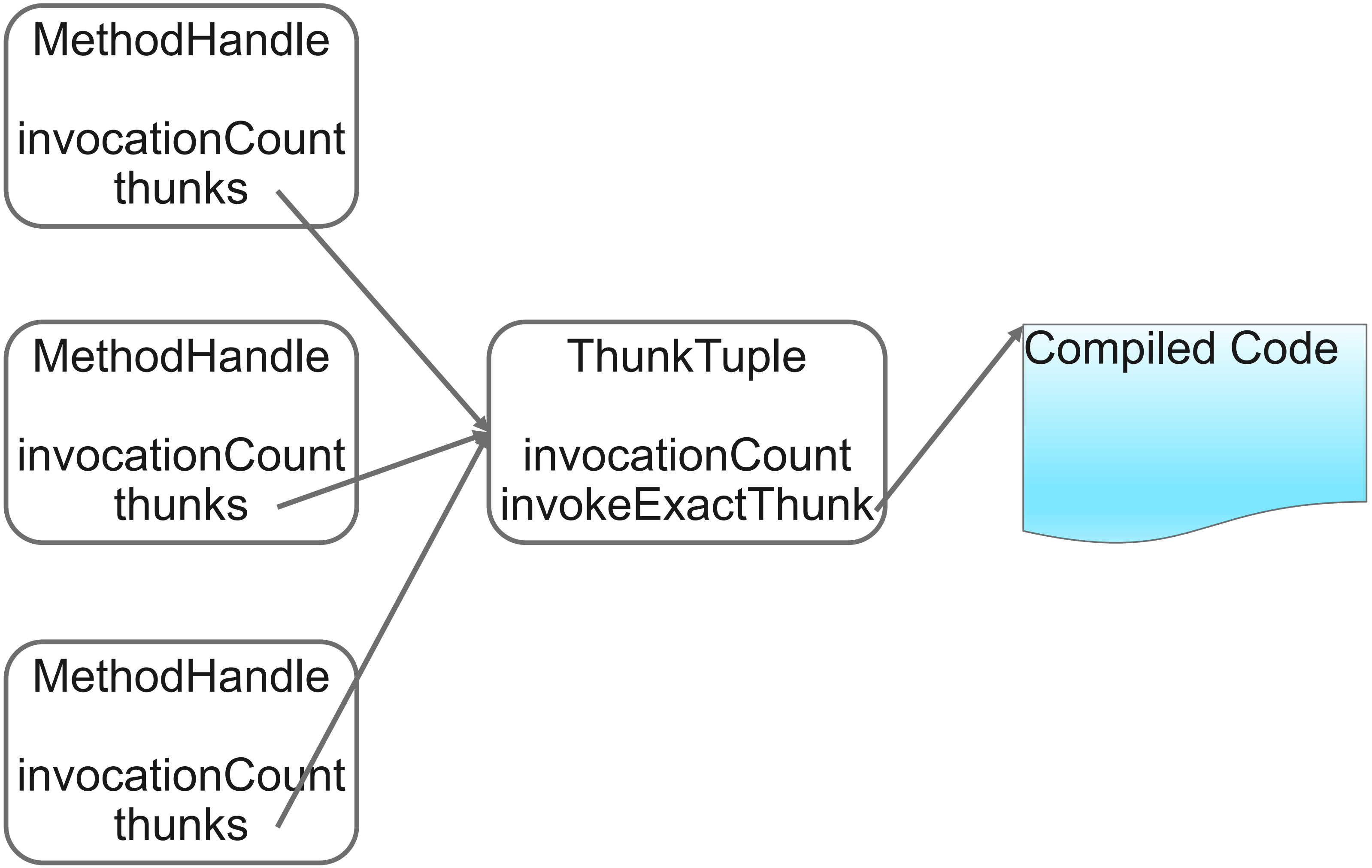
invocationCount
thunks

ThunkTuple

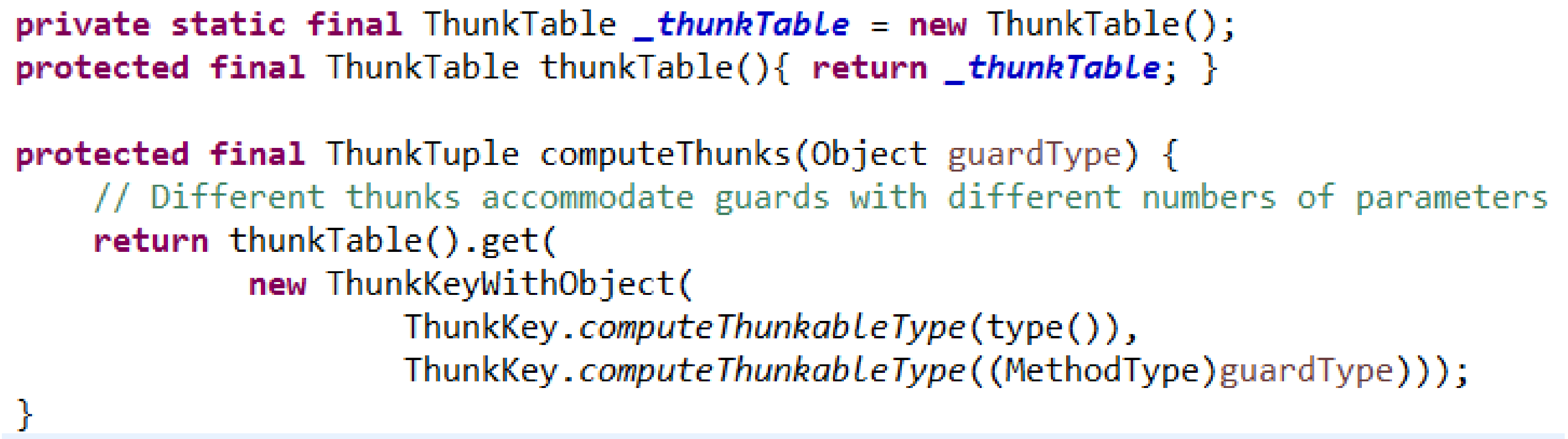
invocationCount
invokeExactThunk

Compiled Code

Avoiding duplicate compiles of equivalent MHs



ThunkTables allow sharing



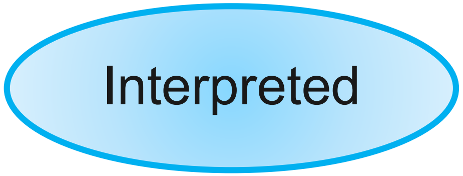
```
private static final ThunkTable _thunkTable = new ThunkTable();
protected final ThunkTable thunkTable(){ return _thunkTable; }

protected final ThunkTuple computeThunks(Object guardType) {
    // Different thunks accommodate guards with different numbers of parameters
    return thunkTable().get(
            new ThunkKeyWithObject(
                    ThunkKey.computeThunkableType(type()),
                    ThunkKey.computeThunkableType((MethodType)guardType)));
}
```

- Every MethodHandle subclass has a ThunkTable

- ThunkTables manage the mapping from MethodHandle to ThunkTuple

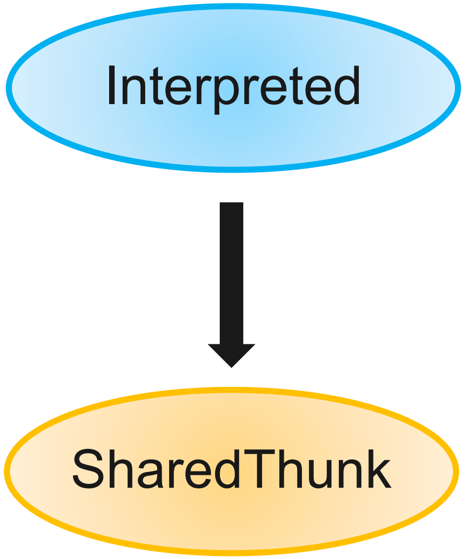
- Goal: Good compiled code with a high degree of sharing.
  - Stay out of the interpreter.
  - Don’t waste code cache

Compilation states

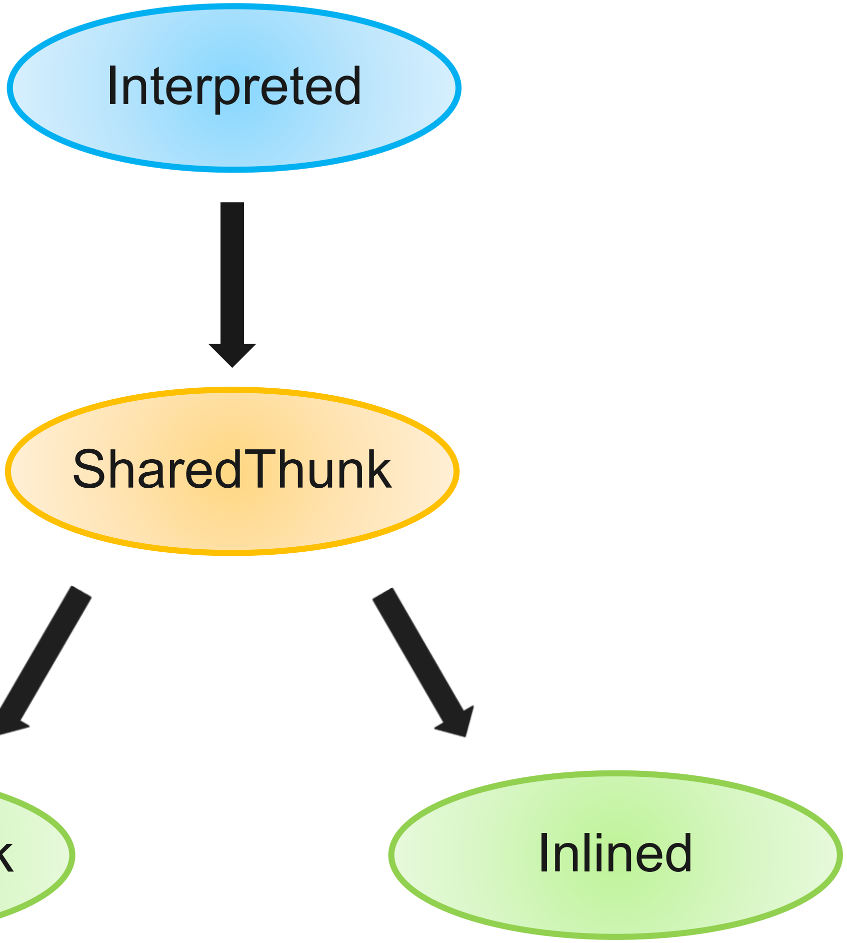


Interpreted

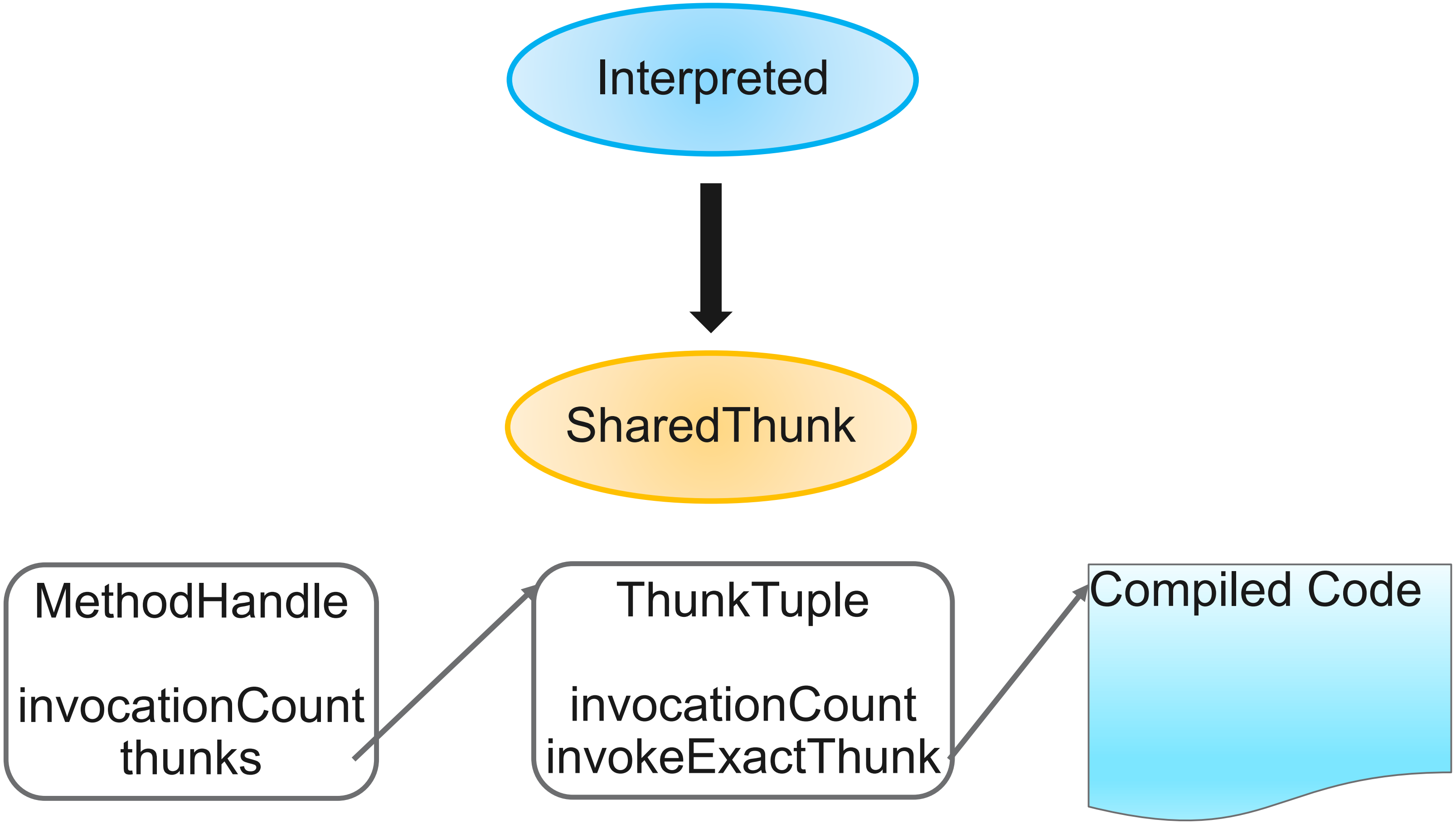
Compilation states



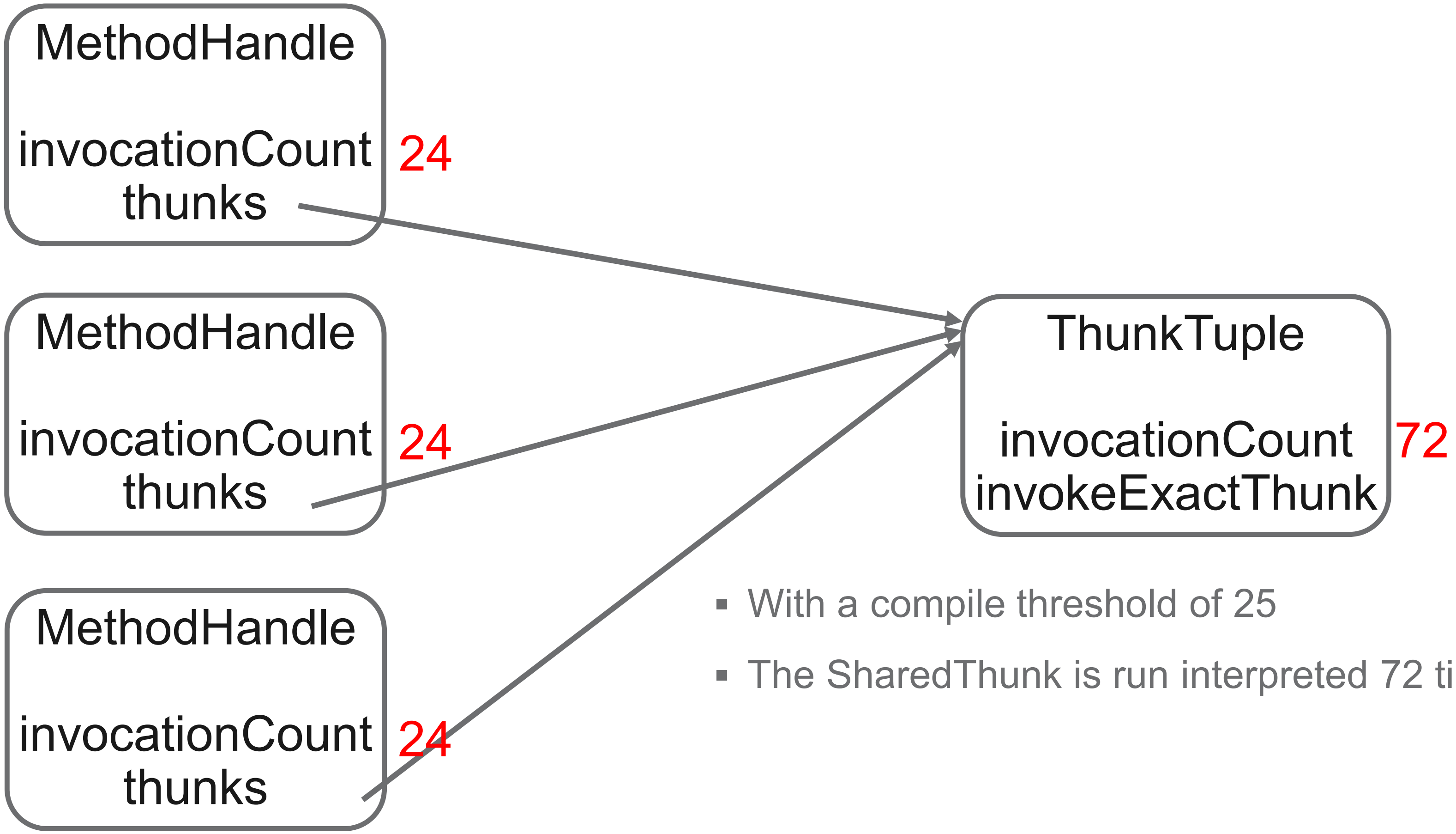
Compilation states



Initial JIT compilation

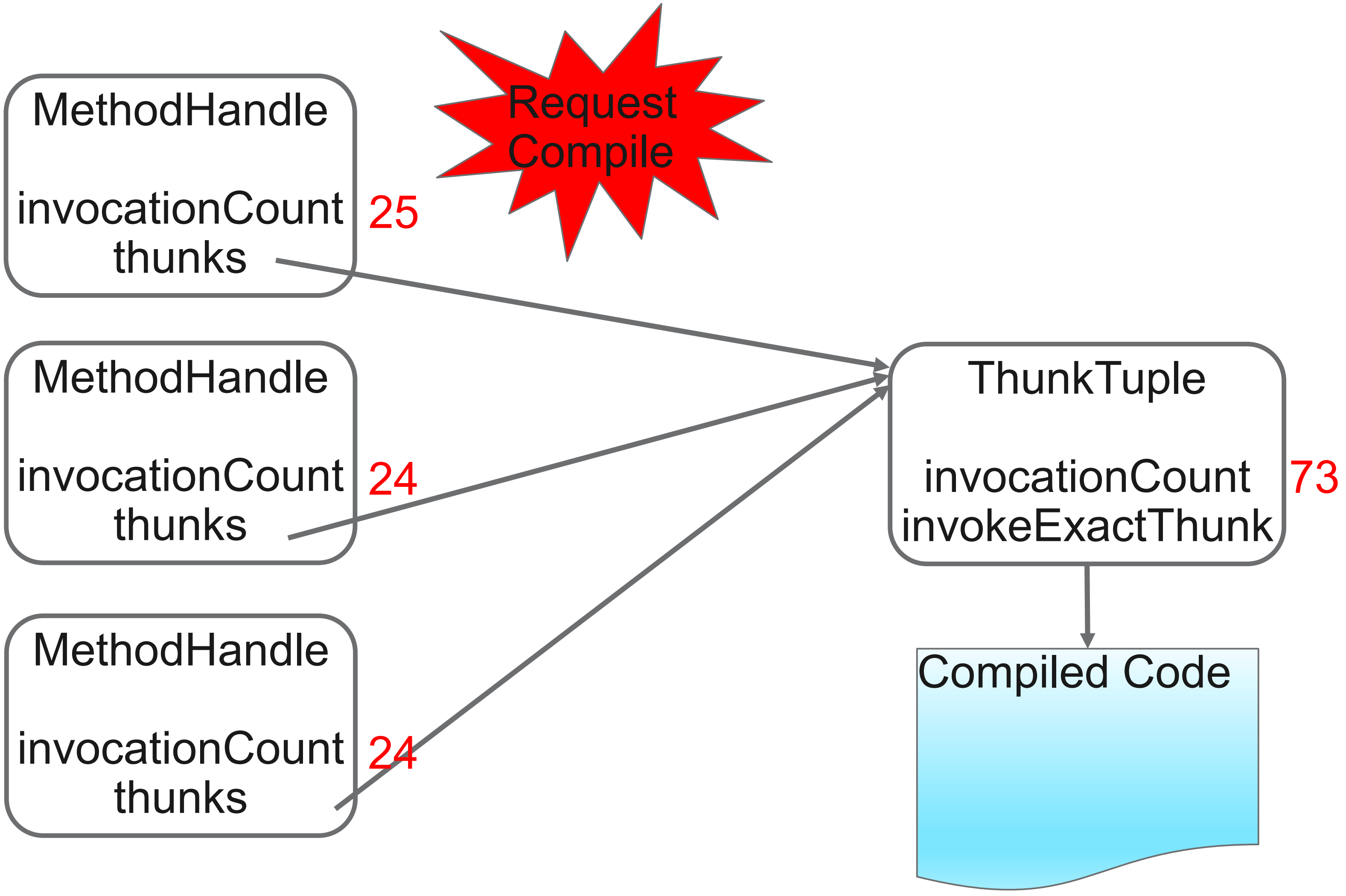


SharedThunk delays



- With a compile threshold of 25
- The SharedThunk is run interpreted 72 times

SharedThunk delays



MethodHandle

invocationCount 25
thunks

Request Compile

MethodHandle

invocationCount 24
thunks

MethodHandle

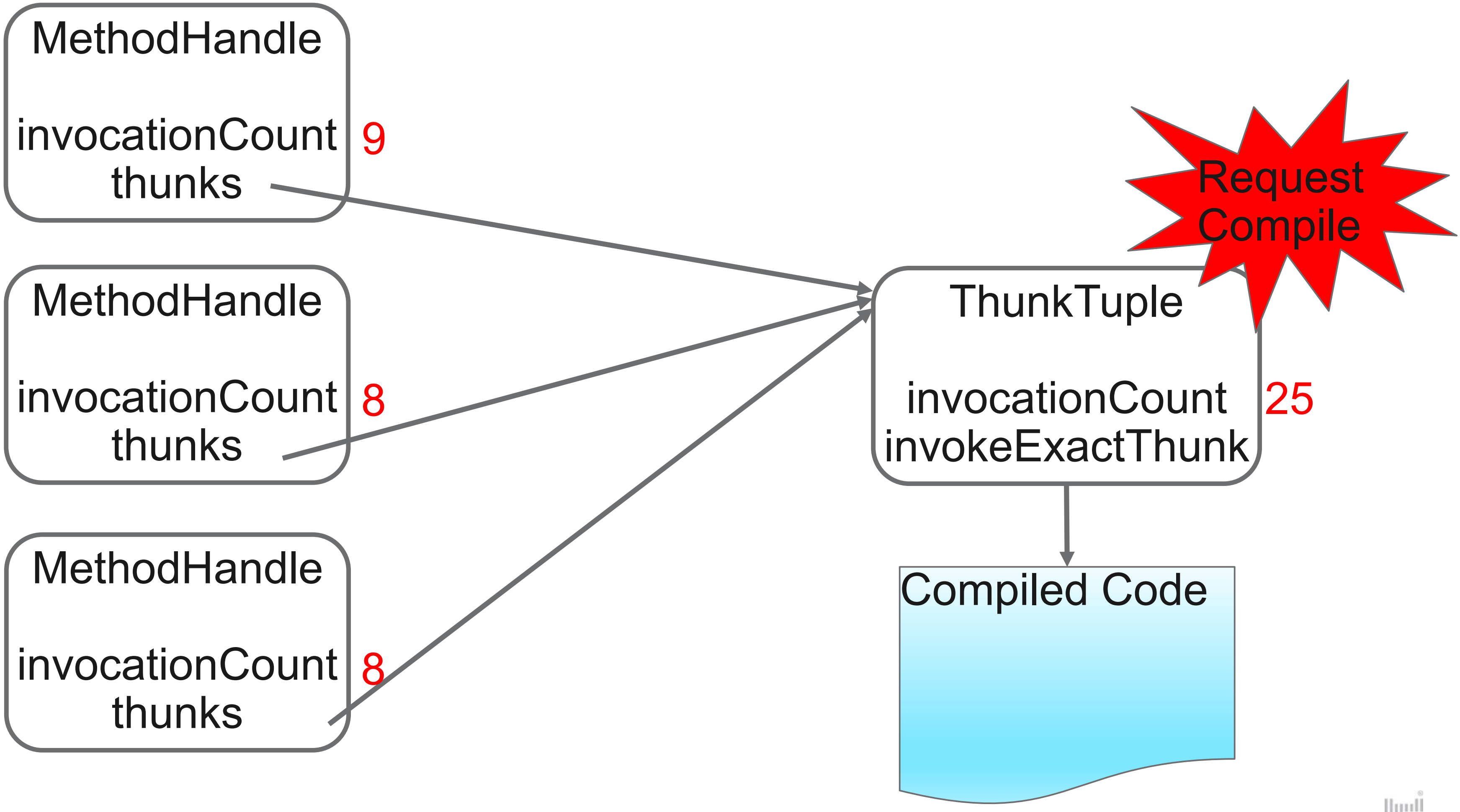
invocationCount 24
thunks

ThunkTuple

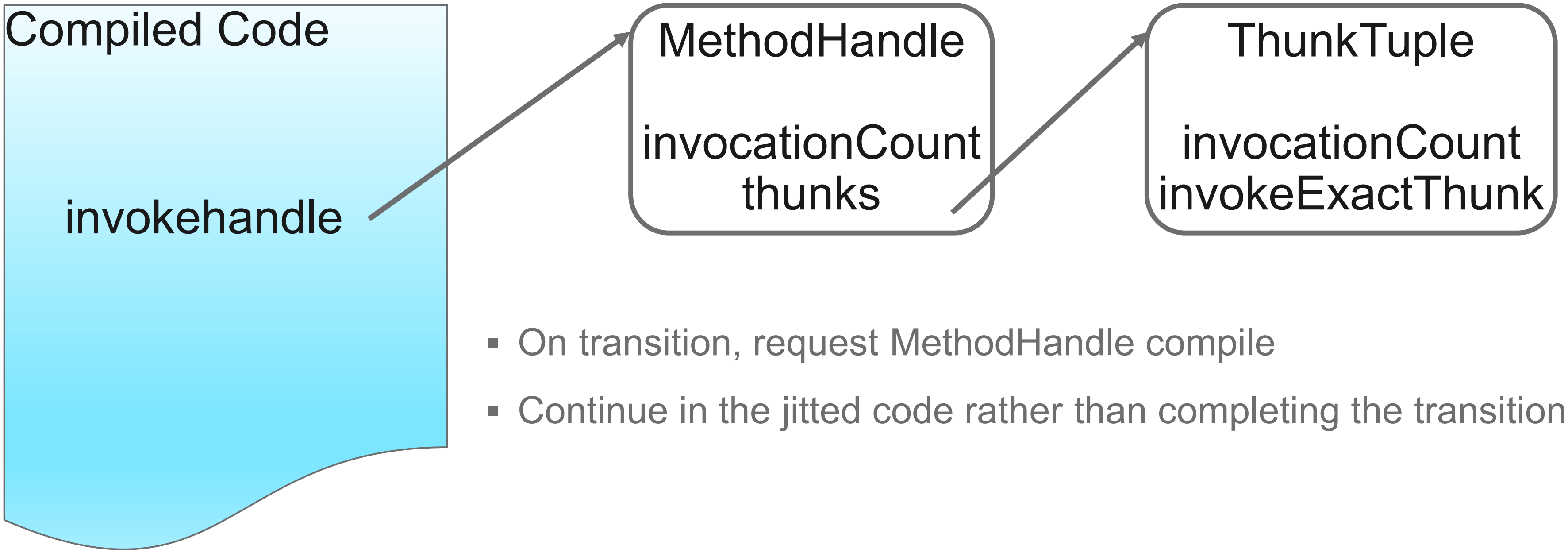
invocationCount 73
invokeExactThunk

Compiled Code

SharedThunk delays resolved

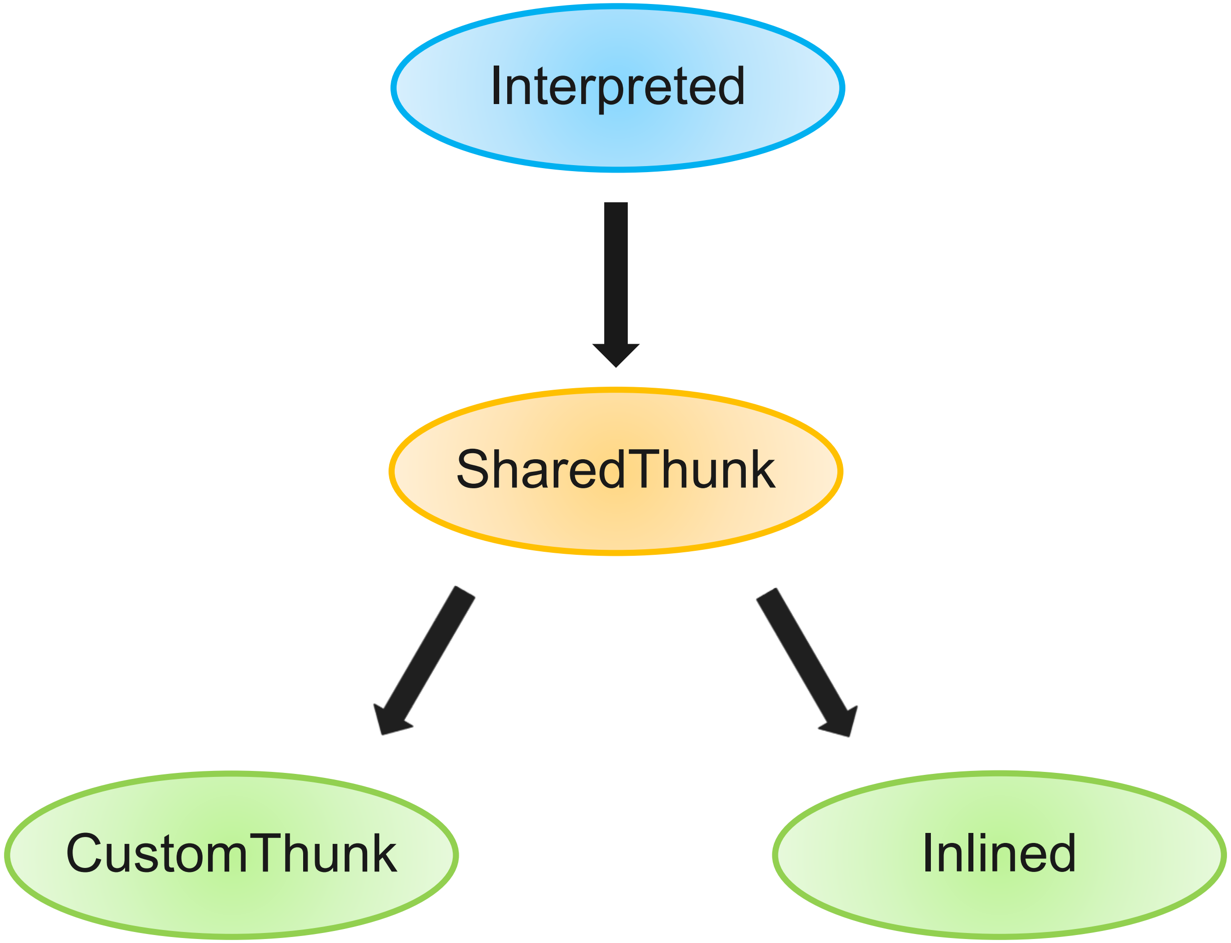


Addressing the cost of J->I transitions



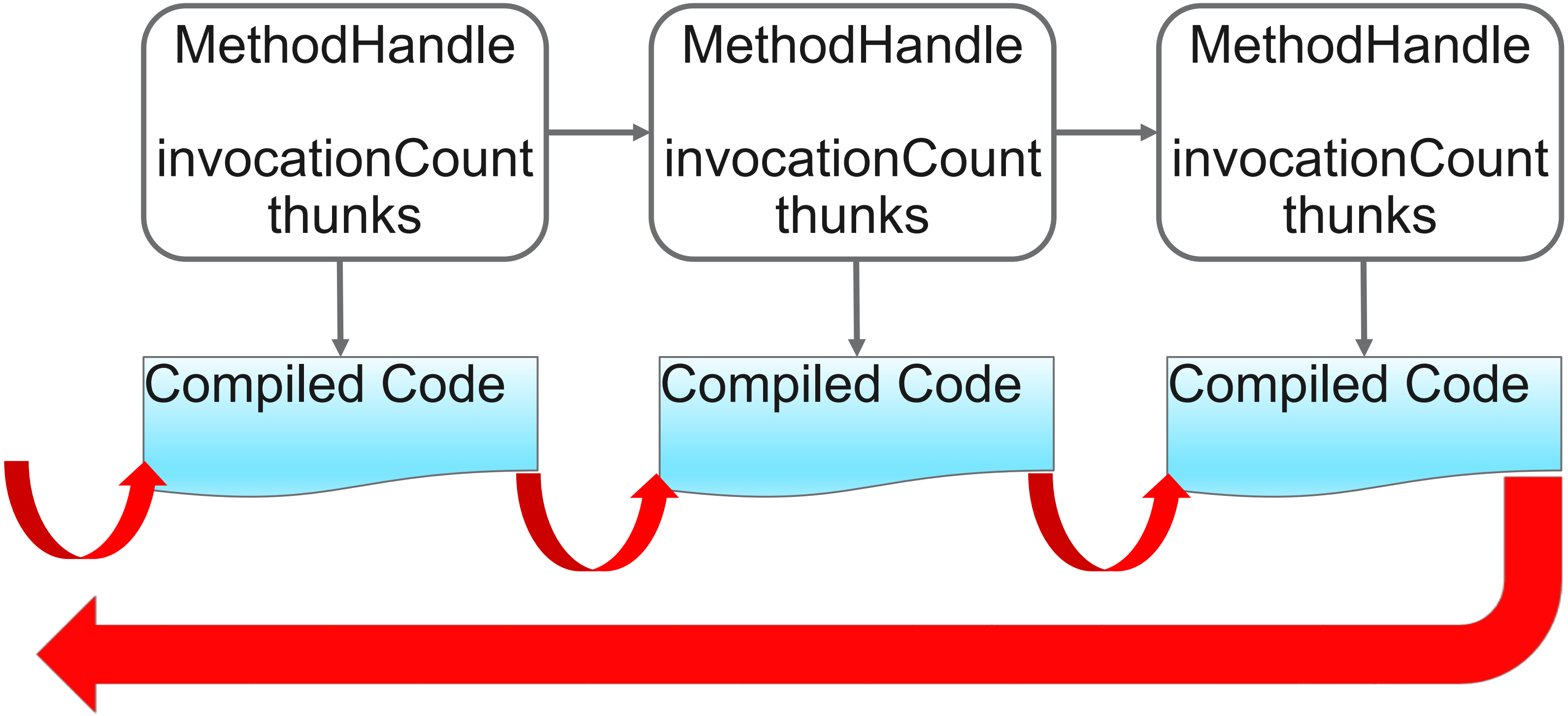
- On transition, request MethodHandle compile
- Continue in the jitted code rather than completing the transition

Compilation states

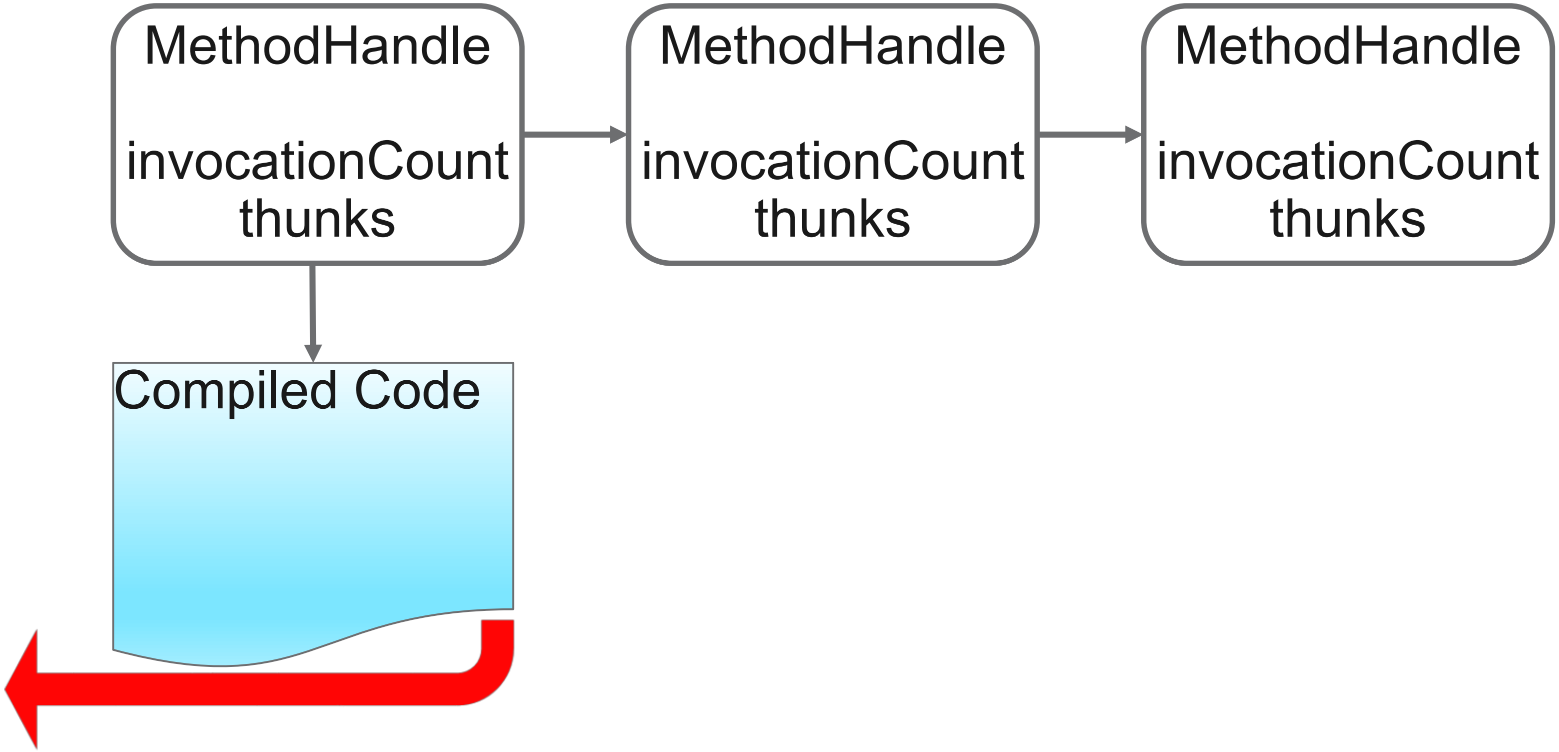


http://mccom.com/blog/wp-content/uploads/2014/07/Customize-Key-

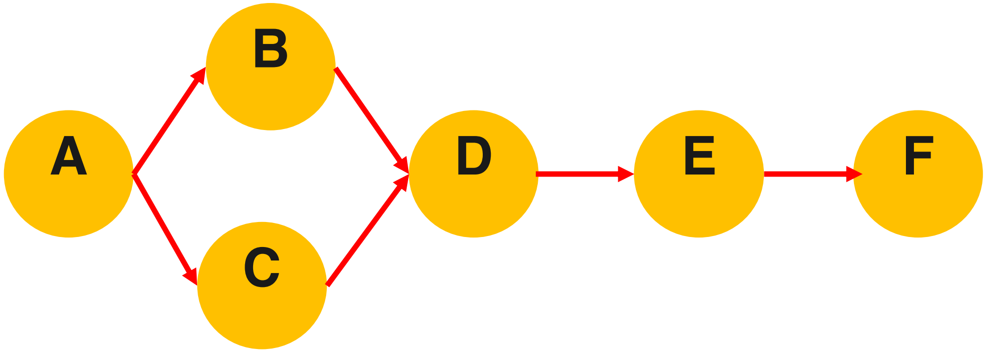
Why CustomThunks?

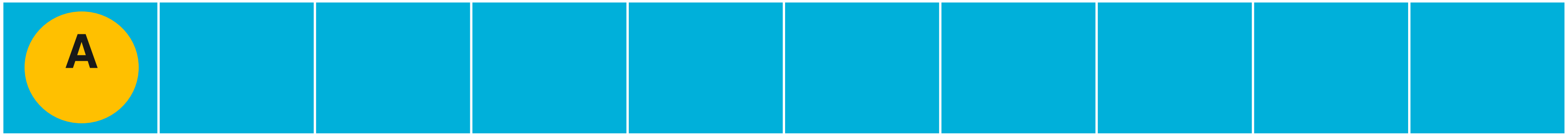


Why CustomThunks?

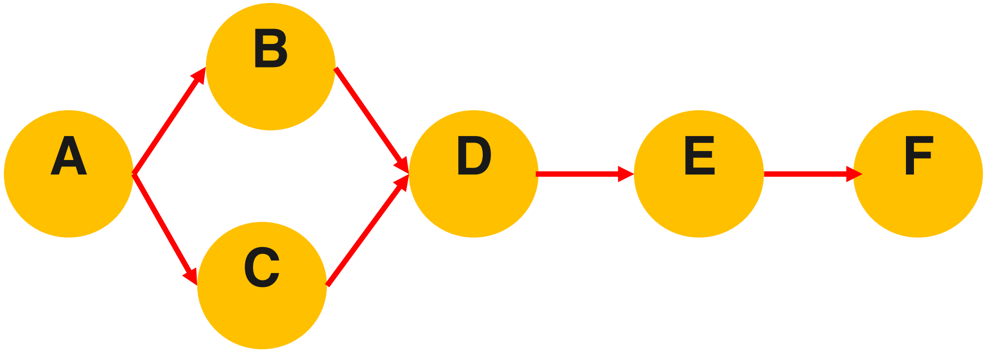


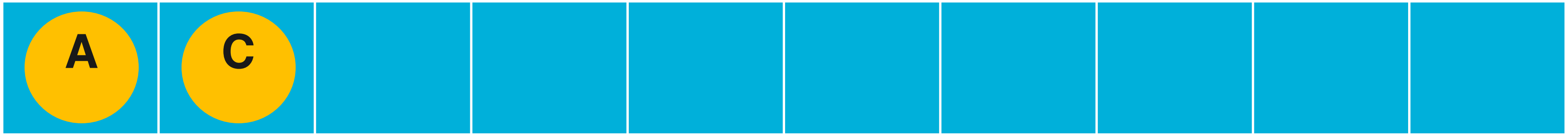
Invocation counts are not enough



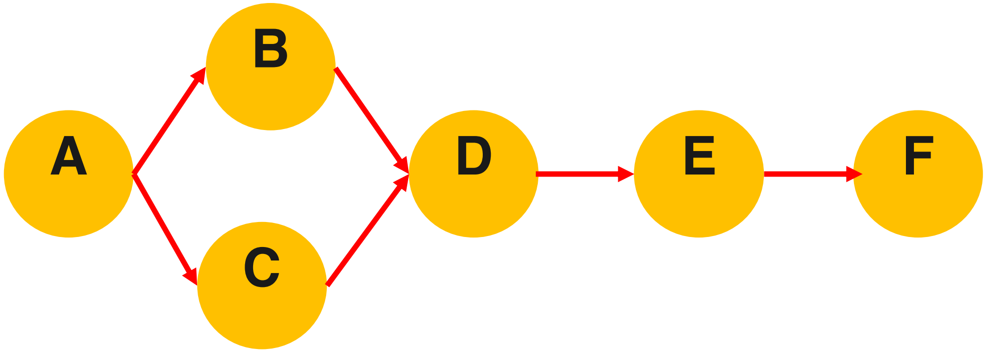


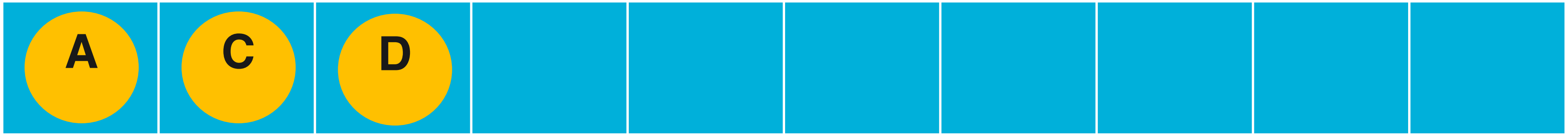
Invocation counts are not enough



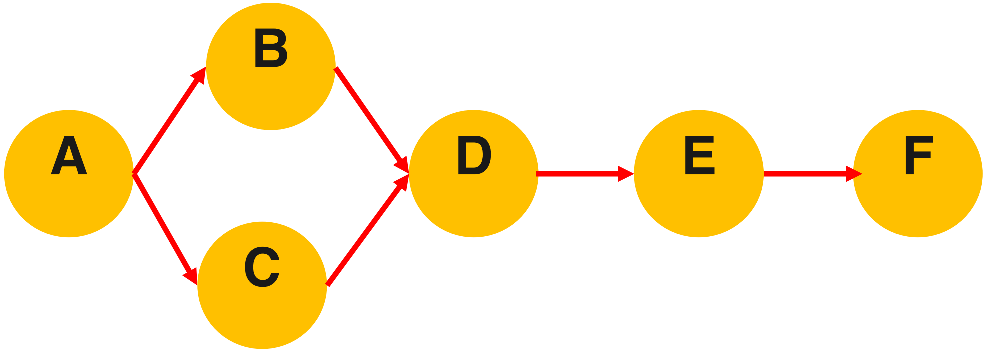


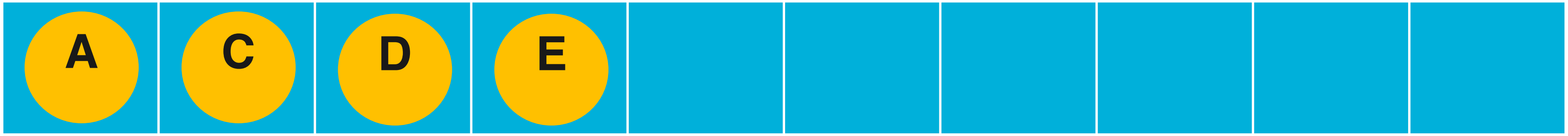
Invocation counts are not enough



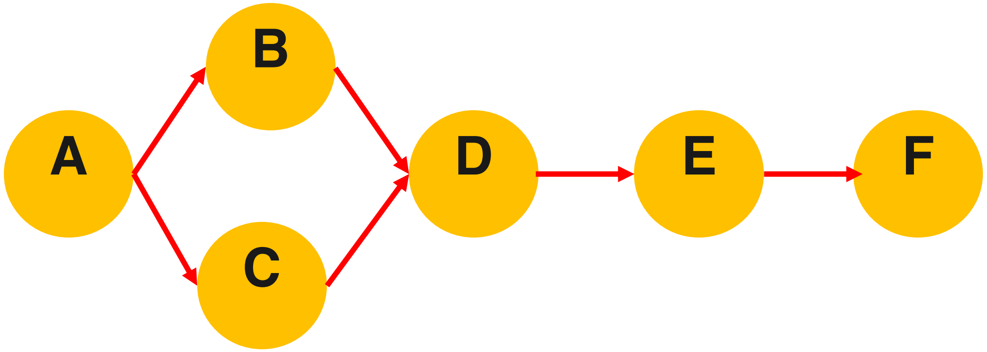


Invocation counts are not enough



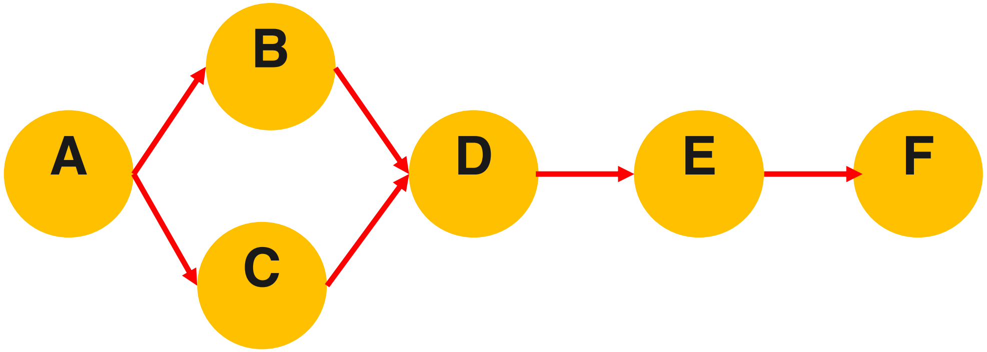


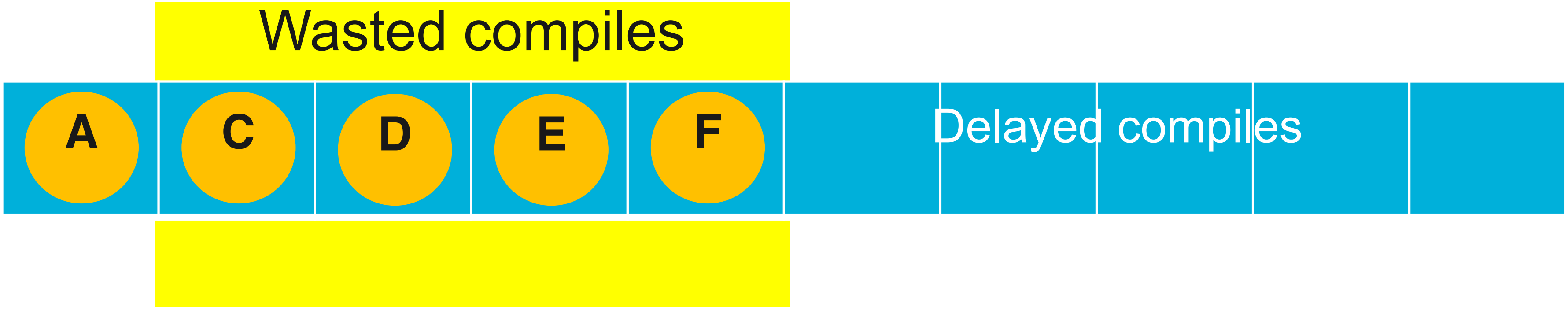
Invocation counts are not enough



A	C	D	E	F					

Invocation counts are not enough



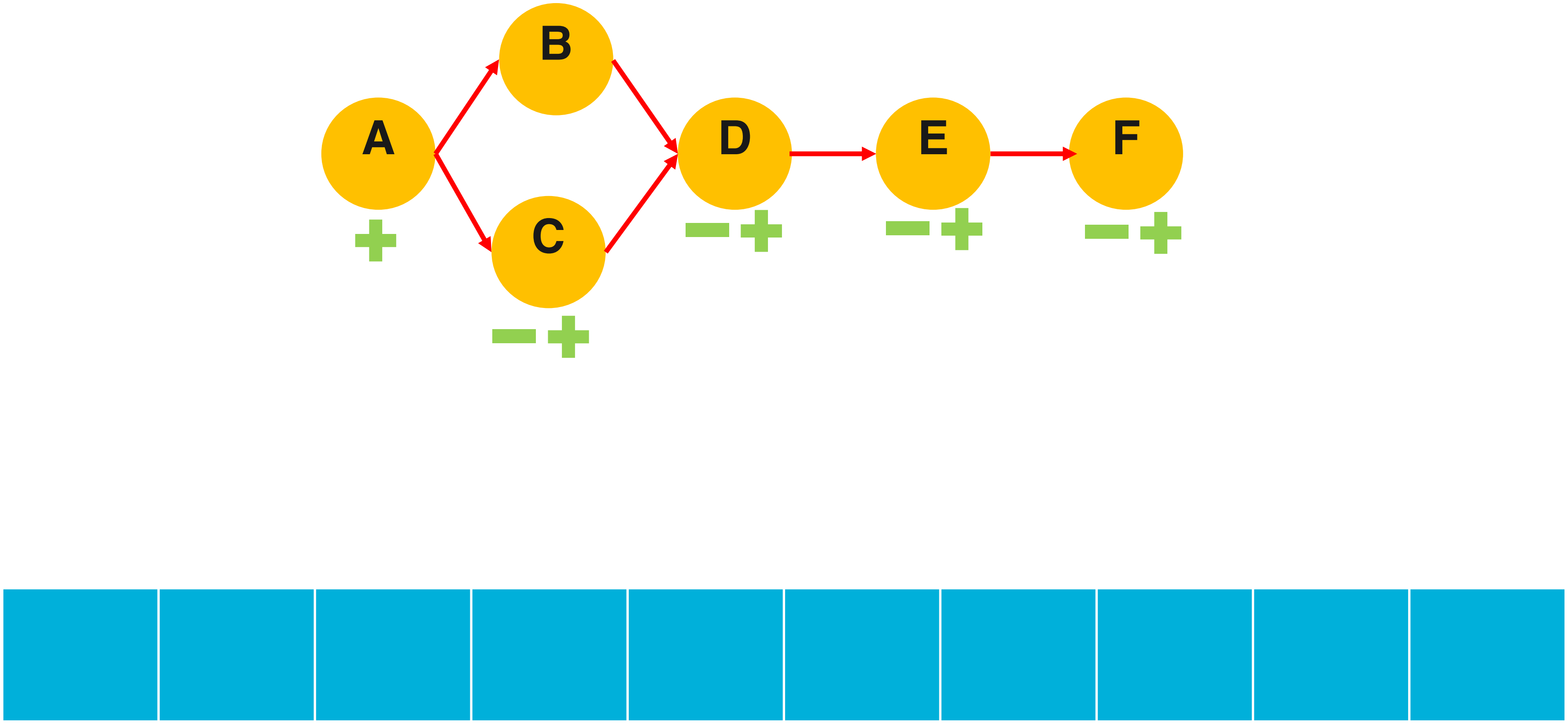


Wasted compiles

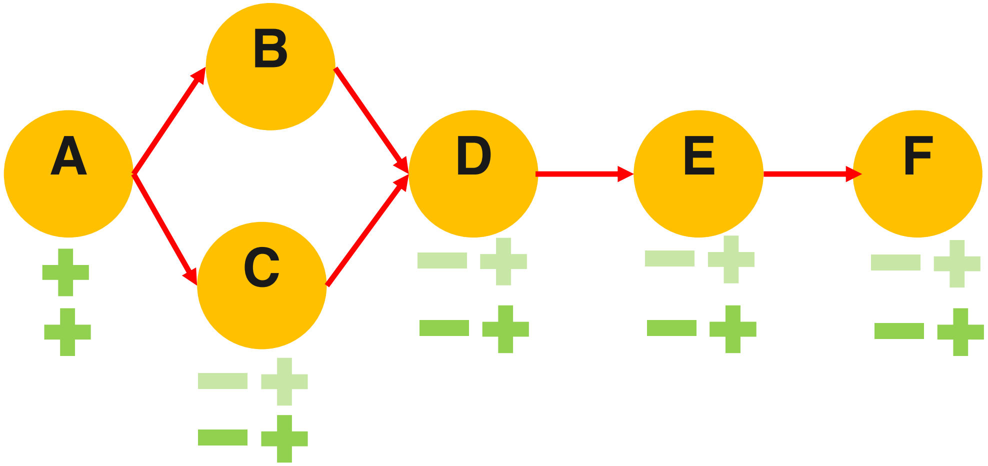
Delayed compiles

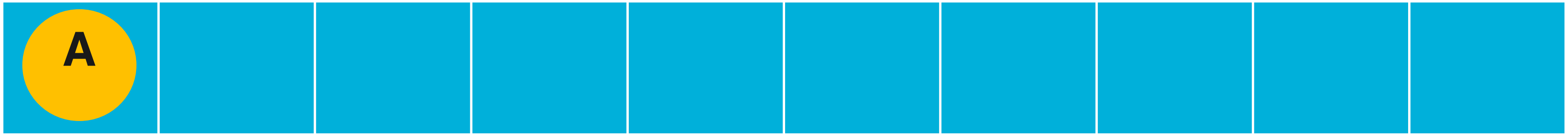
"**Brisbane Lightening**" by Steve Arnold - http://www.flickr.com/photos/stevoarnold/3161660942.
Licensed under **Attribution 2.0 Generic** CC BY 2.0

Avoiding compile storms

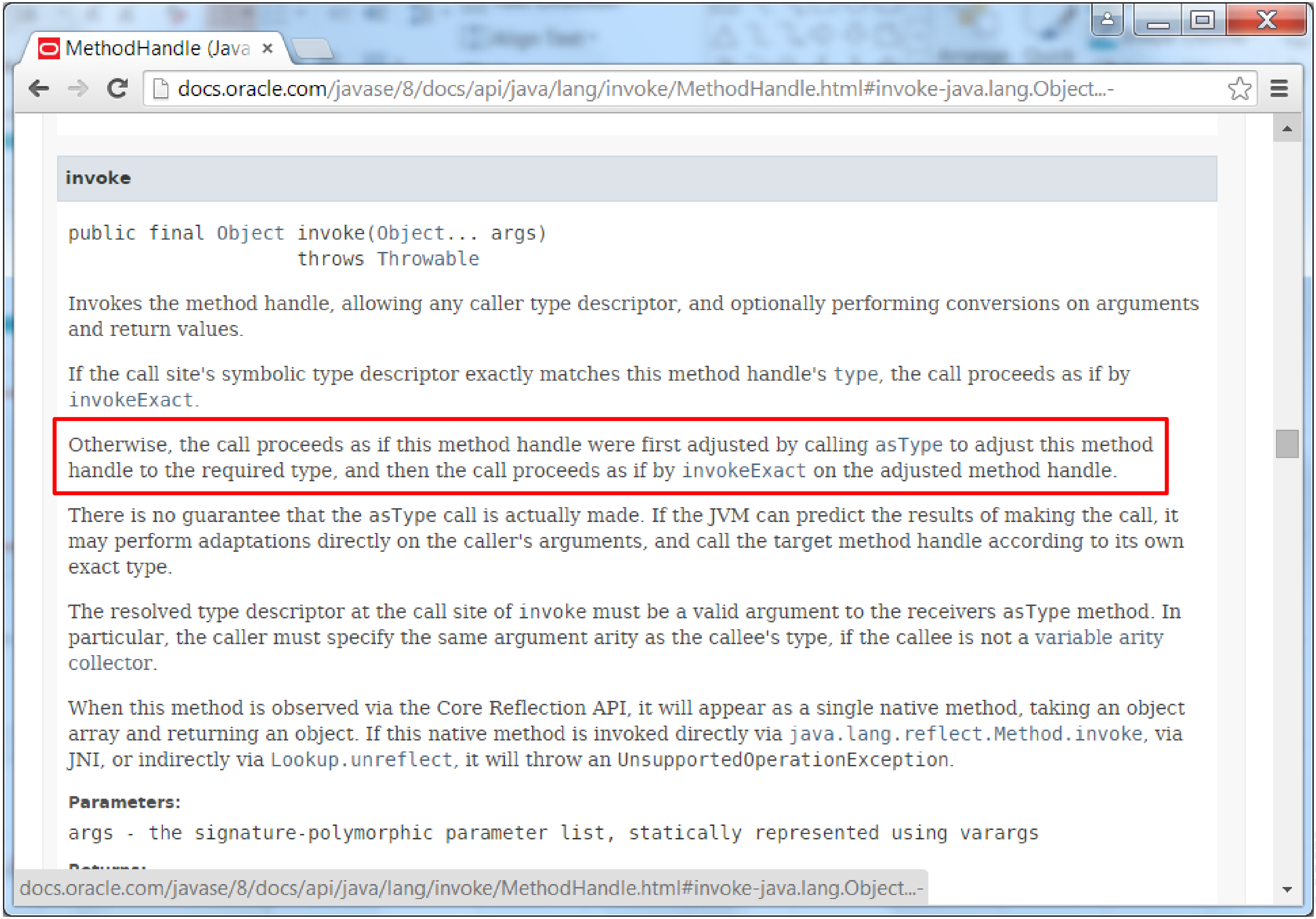


Avoiding compile storms





Avoid MethodHandles.invoke()



MethodHandle (Java ×
docs.oracle.com/javase/8/docs/api/java/lang/invoke/MethodHandle.html#invoke-java.lang.Object...-

invoke

```
public final Object invoke(Object... args)
                    throws Throwable
```

Invokes the method handle, allowing any caller type descriptor, and optionally performing conversions on arguments and return values.

If the call site's symbolic type descriptor exactly matches this method handle's type, the call proceeds as if by invokeExact.

Otherwise, the call proceeds as if this method handle were first adjusted by calling asType to adjust this method handle to the required type, and then the call proceeds as if by invokeExact on the adjusted method handle.

There is no guarantee that the asType call is actually made. If the JVM can predict the results of making the call, it may perform adaptations directly on the caller's arguments, and call the target method handle according to its own exact type.

The resolved type descriptor at the call site of invoke must be a valid argument to the receivers asType method. In particular, the caller must specify the same argument arity as the callee's type, if the callee is not a variable arity collector.

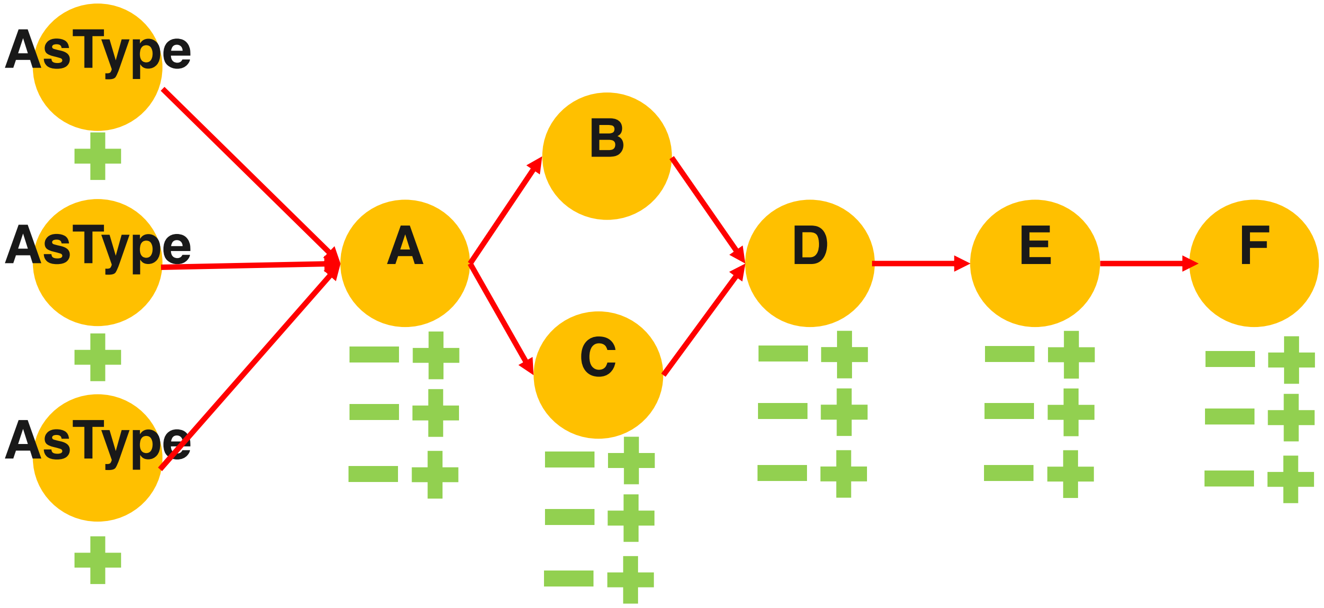
When this method is observed via the Core Reflection API, it will appear as a single native method, taking an object array and returning an object. If this native method is invoked directly via java.lang.reflect.Method.invoke, via JNI, or indirectly via Lookup.unreflect, it will throw an UnsupportedOperationException.

Parameters:

args - the signature-polymorphic parameter list, statically represented using varargs

docs.oracle.com/javase/8/docs/api/java/lang/invoke/MethodHandle.html#invoke-java.lang.Object...-

Avoid MethodHandles.invoke()



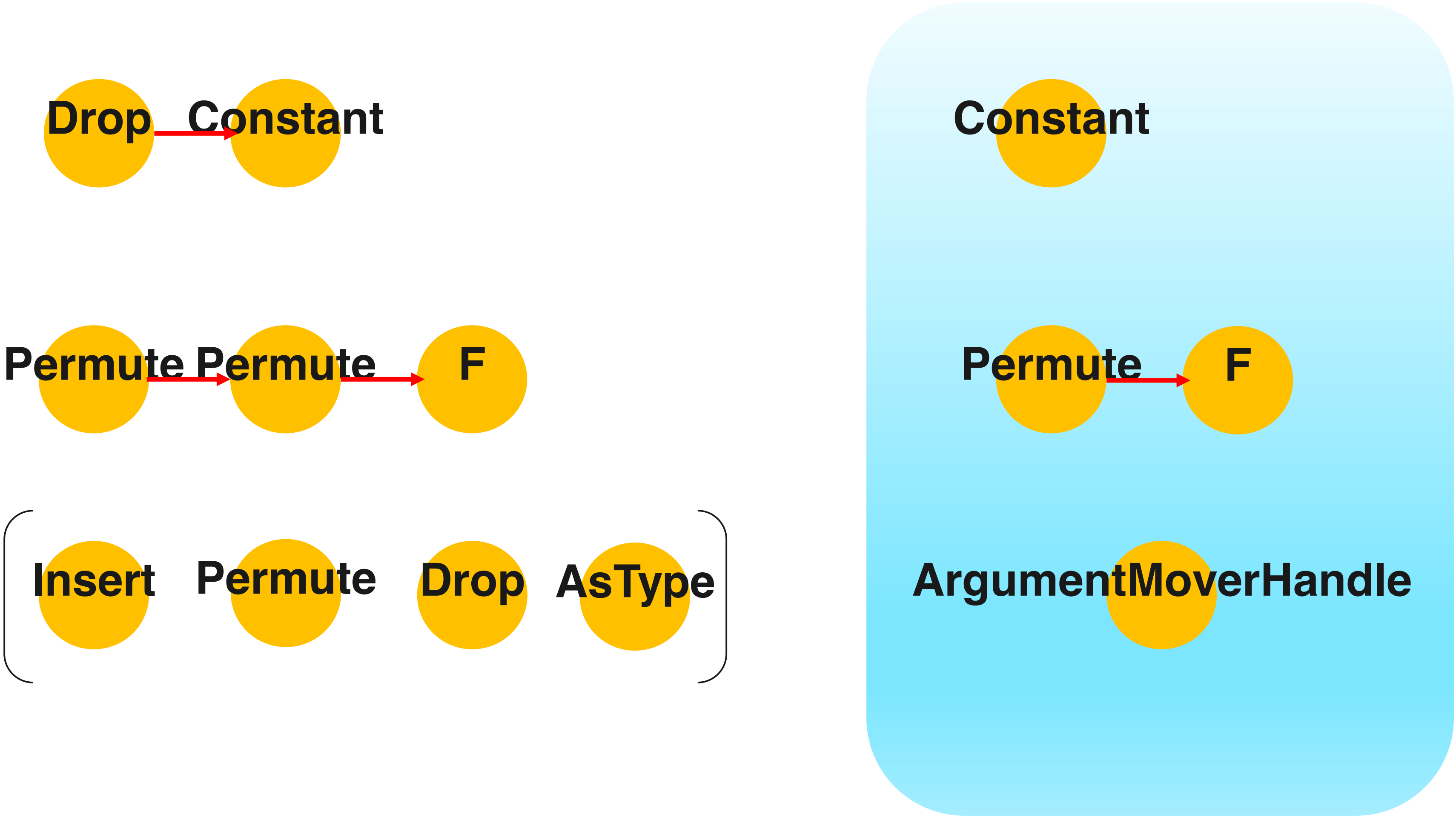
- Counting occurs on the AsType handle, not the head of the chain
- AsType from multiple signatures defeats one-element cache solution

Static optimizations

http://www.mathpirate.net/log/wp-content/uploads/2009/09/Static.jpg

Super bytecodes!

Static optimizations



Drop Constant

Constant

Permute Permute F

Permute F

Insert Permute Drop AsType

ArgumentMoverHandle

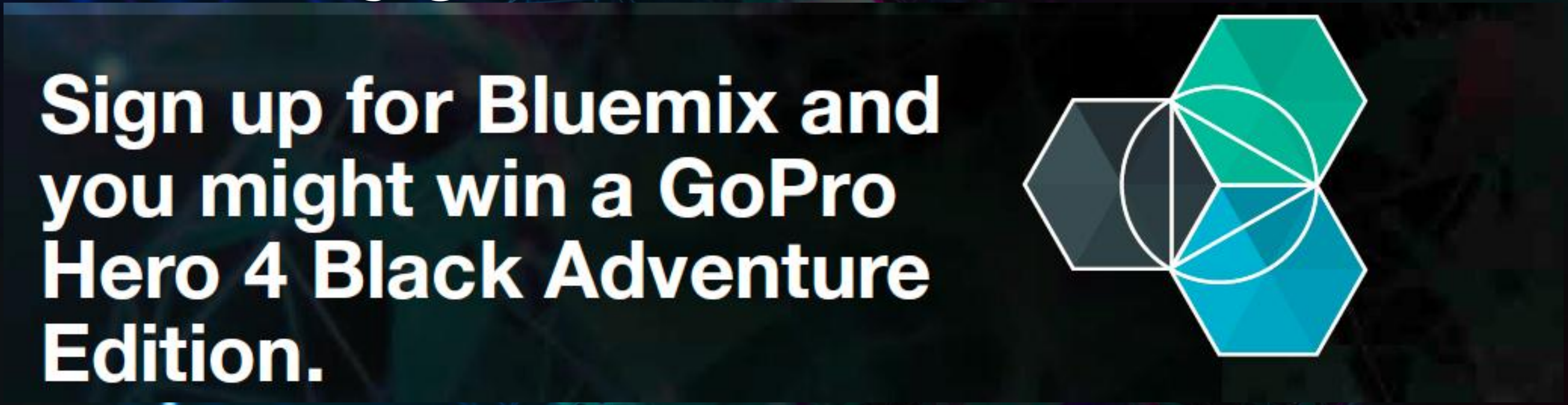
Future directions

- AOT SharedThunks
- Additional “super handles” like drop+constant
- AsType optimizations
- Faster / smaller MethodHandle compiles
- UNB PhD candidate looking at data mining MH chains from existing applications

Meet IBM at Jfokus!

IBM Bluemix™

IBM



Sign up for Bluemix and you might win a GoPro Hero 4 Black Adventure Edition.

Sign up here: ibm.biz/bluemix-jfokus

Winner will be found at IBM's booth
4th of February at 15.00

Follow us on Twitter
@IBMBluemix

Make Your Day CHALLENGE is coming soon!

Pre-signup at vaadin.com/challenge

**Event by
Vaadin &
IBM Bluemix**

IBM Global Entrepreneur Program

IBM Global Entrepreneur offer Startups resources including free software and technical experts, exposure to 600+ expert mentors, plus access to a global network of clients.

Also eligible startups can apply for getting between 1 K USD to 10 K USD a month credits for 12 months on their Softlayer and/or Bluemix account

IBM

IBM Global Entrepreneur Program
Sign up here: ibm.com/isv/startup

IBM Global Entrepreneur Program for Cloud Startups – apply for credits
Sign up here: ibm.biz/CloudStartup

IBM Analytics Starter Program
Sign-up here:
ibm.biz/analyticsstarter

Go-to-market support

Business mentoring

Technical expertise