Roos Instruments, Inc.

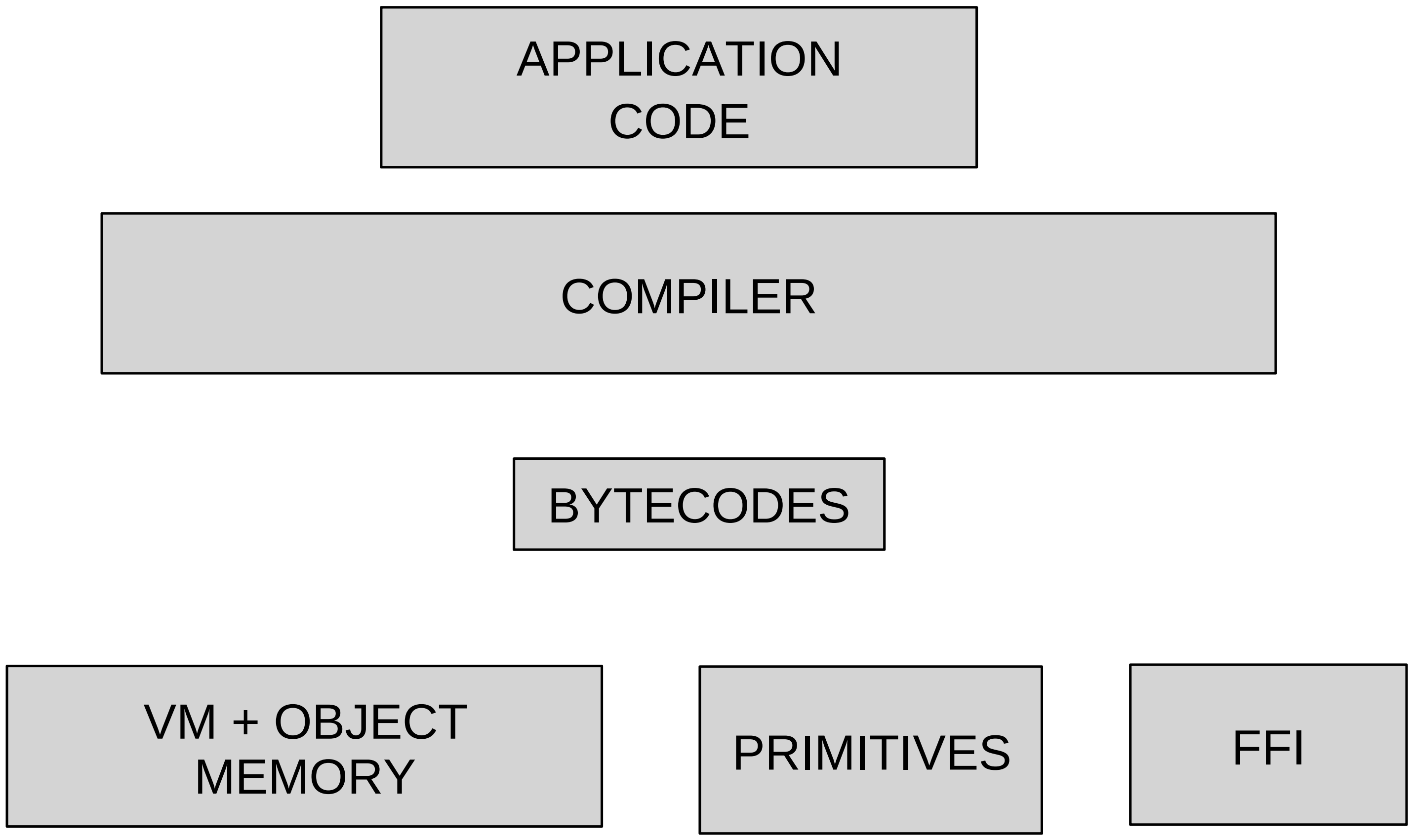
Building a Dynamic Language
on the JVM

code link on JVM Summit wiki

Smalltalk as an Example

- Message Based
- Everything is an Object
- Byte Coded VM
- Excellent FFI

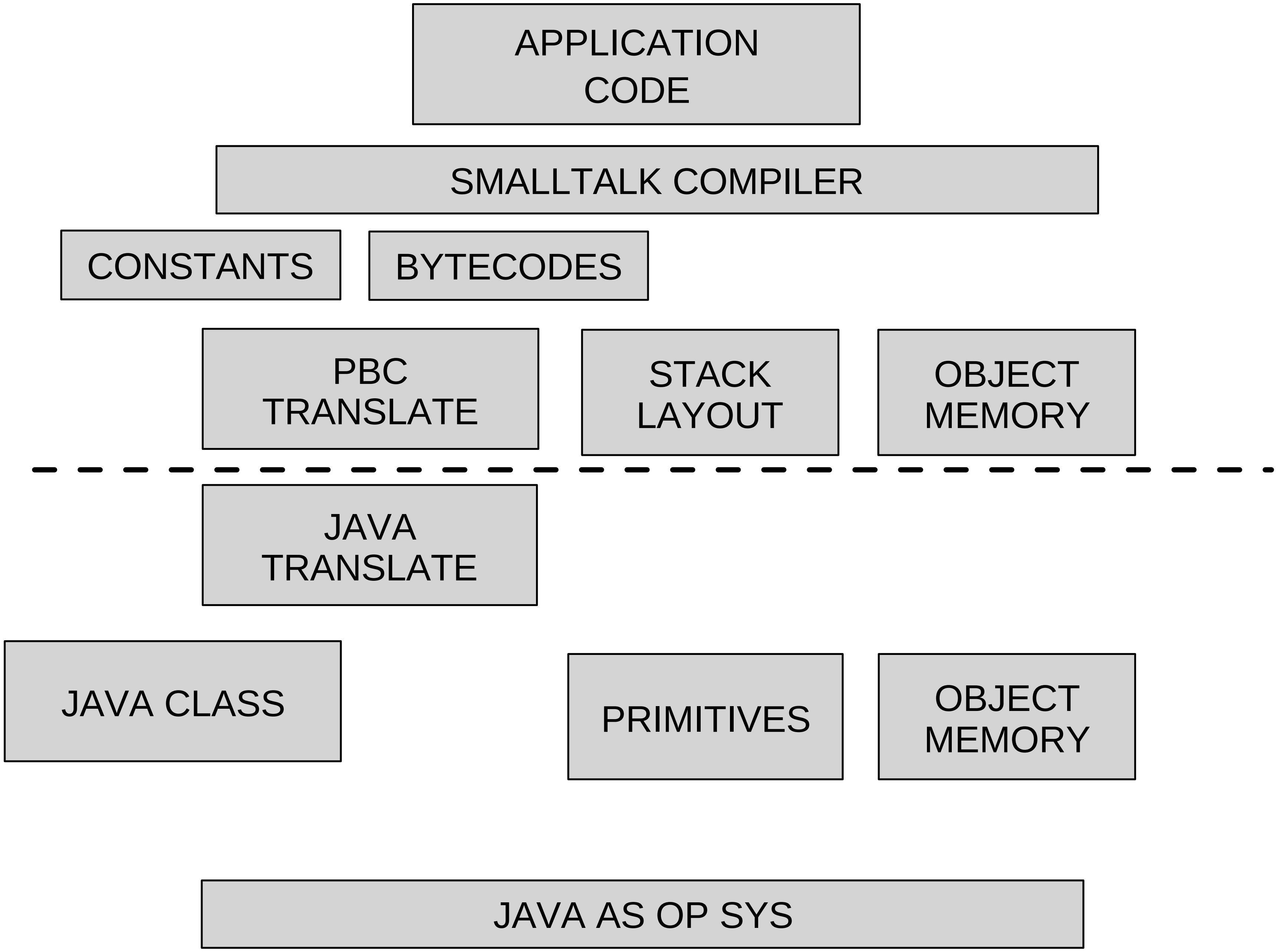
Existing Architecture



Steps

- Analyze Existing byte code usage
- Define a translation interface
  - Object memory
  - Stack
  - byte codes
  - primitives
  - constants
- Port as is first(don't try to improve it)

JVM IMPLEMENTATION



Main RTALK Java Class Files

ri.core.rtalk

- RtObject (object structure)
- RtCallSite (method sends)
- RtPrimitives (java interface)
- RtFixedObjects (jvm and rtalk shared)
- PbcToJvmTranslate (jvm class generator)
- ClassLoaderForRtalk (small loader)

Misc RTALK Java Class Files

ri.core.rtalk

PbcByteCodes	(constants)
PbcHexStream	(disassem)
RtCallSites	(list of sites)
RtDebugger	(debug support)
RtDebugTerminateThread	
RtNonLocalReturn	

Architecture Mismatches

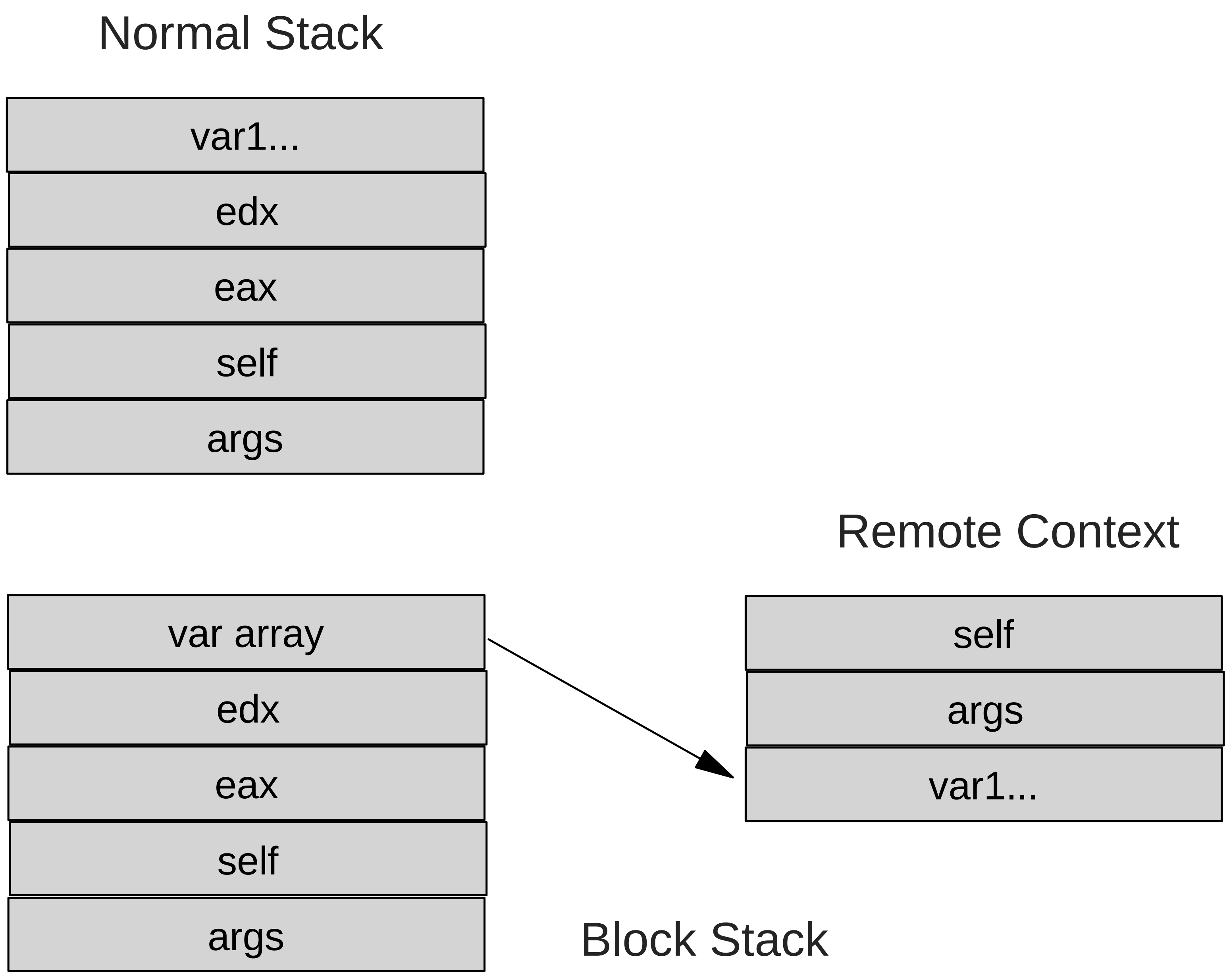
- Stack + 2 registers (eax edx)
- Stack space == variable space
- Object Memory (ints stored in pointers)
- Constant Type differences

Object Structure

RtObject

Shape + flags	
SIZE	
METHOD ARRAY	[[methods][methods][]...]
POINTER	byte[], double[], RtObject[], Object
PRIMITIVE	long, double

Stack Var Structure



Methods from ST to Java

- Start from existing bytecodes
- Translate to PBC (minimal bytecode set)
  - constants serialization
  - byte code conversion
  - fixup dead code, order
- Translate from PBC to Java Class
  - Use ASM asm.ow2.org/

ByteCode Differences

- 25 PBC but only 4 real differences
- Method Invocation
- Primitives
- Blocks and returns
- Constants

Method Sends

PBC Description

[10][n][size][selector]   tos perform selector with n args

PBC Translation

PbcToJvmTranslate line 297

Bootstrap Method

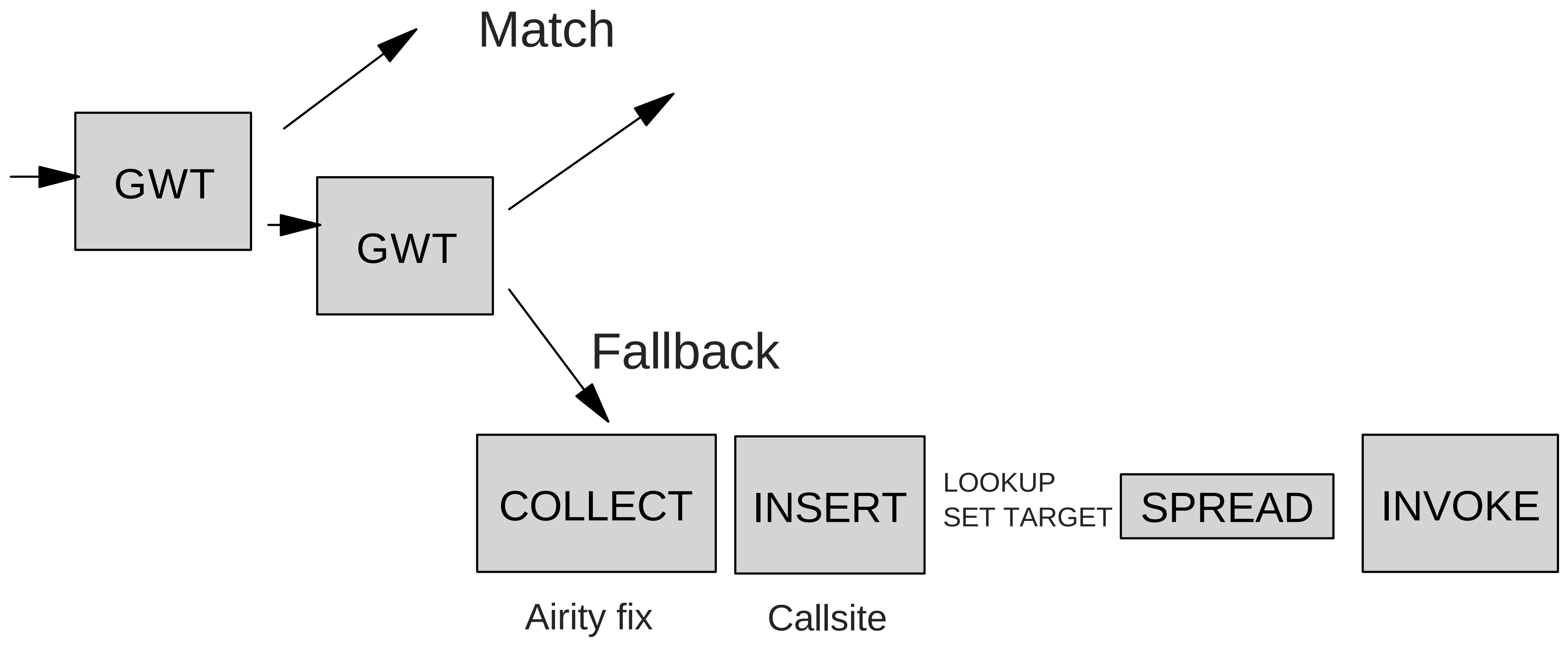
RtCallSite   line 355

Fallback method

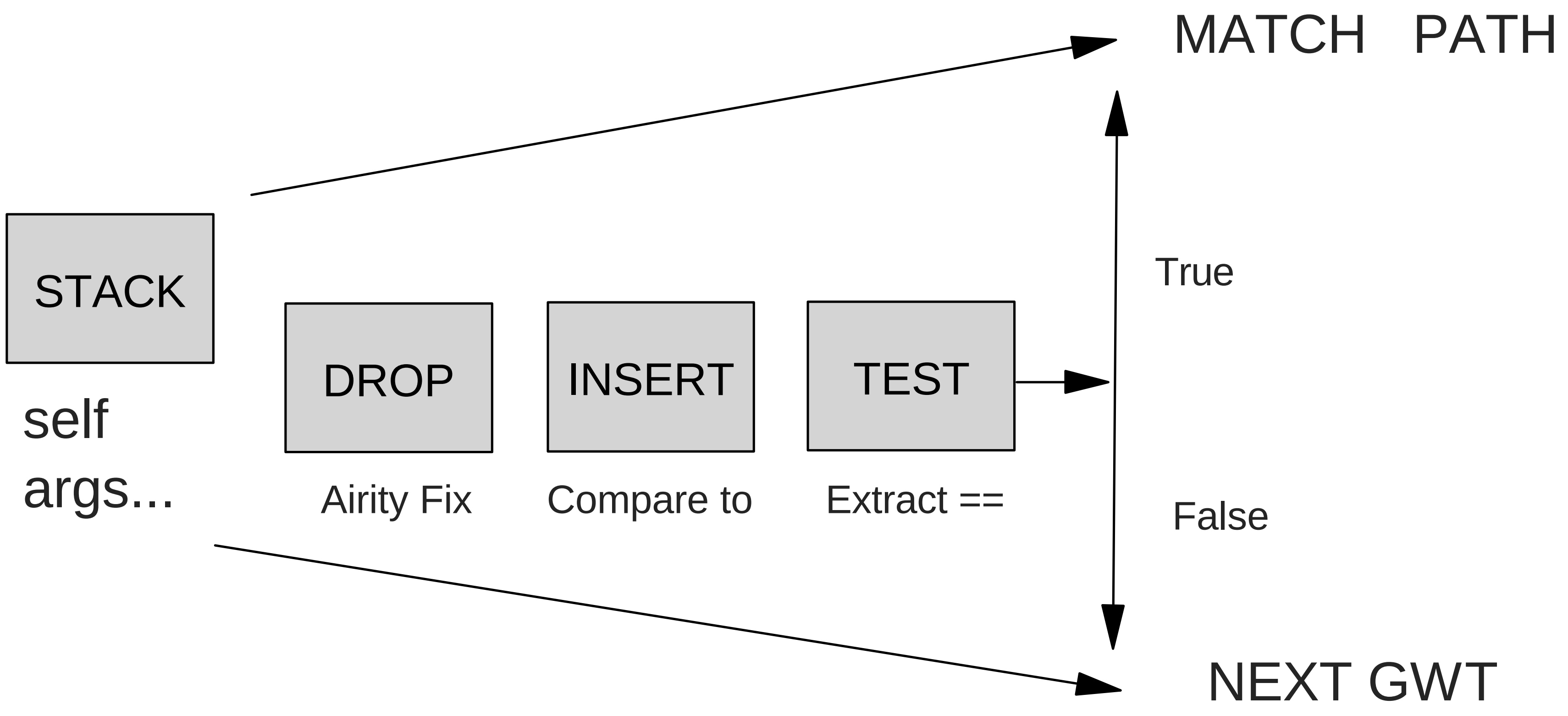
RtCallSite   line 411

GWT as inline cache

RtCallSite



GWT


MATCH PATH
STACK
self
args...
DROP
Airity Fix
INSERT
Compare to
TEST
Extract ==
True
False
NEXT GWT

Primitives

- Along with bytecodes do all the work
- Written in Java with RtObject args
- Supports fallback to Smalltalk code
- Low level (math) and high level (string)
- Largest Java Code effort (2000 lines)

Primitive Code Example

```
exp
        "Answer the exponential of the receiver "
    <jprim: ri/core/rtalk/RtPrimitives primFloatExp>
    ^self primitiveFailed



in ri.core.rtalk.RtPrimitives

  static public RtObject primFLoatExp(RtObject rcvr) {
    // return exponential of the receiver
    double c=rcvr.getDoubleValue();
    return new RtObject(Math.exp(c));
  }

PbcTpJvmTranslate
     invoke static  line 384
```

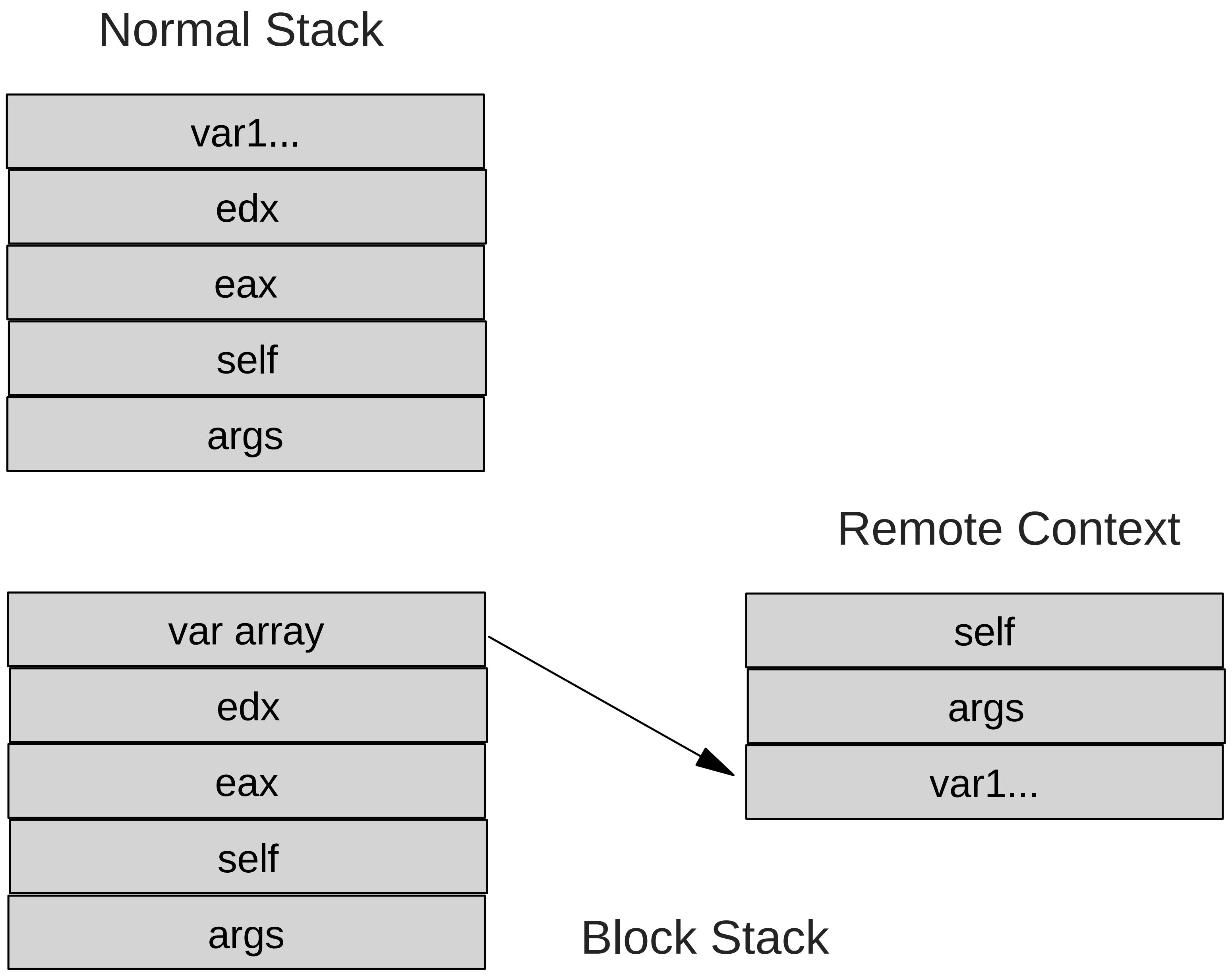
Blocks

- Code plus context
- Code is just another method (block$n)
- Replaced stack vars with shared array
- Non local return
  - returns to caller of creator
  - use var array to locate return frame
  - throw exception with var array + return

Block Code Example

```
includes: anObject
        "Answer true if the receiver contains an
         element equal to anObject, else answer false."
    self do: [ :element |
        anObject = element
            ifTrue: [^true]].
    ^false
```

Stack Var Structure



Blocks

PBC Description

[18][n][m] creates an n (n = 0 to 2) argument block with code

PBC Translation

PbcToJvmTranslate line 445

Bootstrap Method

RtCallSite   line 324

RtObject support (create the object)

RtObject   line 441

RtPrimitive support to invoke the block

RtPrimitives   line 1753

Constants/Literals

- In Smalltalk can be any object
- In Java are limited to primitives
- In reality are also limited in ST
  - primitives and arrays of primitives
  - Globals and Class Vars (use prim)
- Use Constant Methodhandle to create
  - name is serialized constant

Constant creation

PBC Description

[40][size][b][...] Convert next size hex bytes to an instance of type b and push onto stack

PBC Translation

PbcToJvmTranslate line 699

Bootstrap Method (ConstantCallSite)

RtCallSite   line 302

Support Code

RtObject   line 501

JVMTI

ri.core.rtalk.RtDebugger

- Stack var inspection
- Hop step jump
- instances inspection
- JVMTI with JNI wrapper
  - C dll - javaDebug.cp
  - attach as a debug agent