J9: Under the Hood of the Next Open Source JVM

Dan Heidinga, J9 VM Interpreter Lead
Daniel_Heidinga@ca.ibm.com
@DanHeidinga
18 September 2016

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?

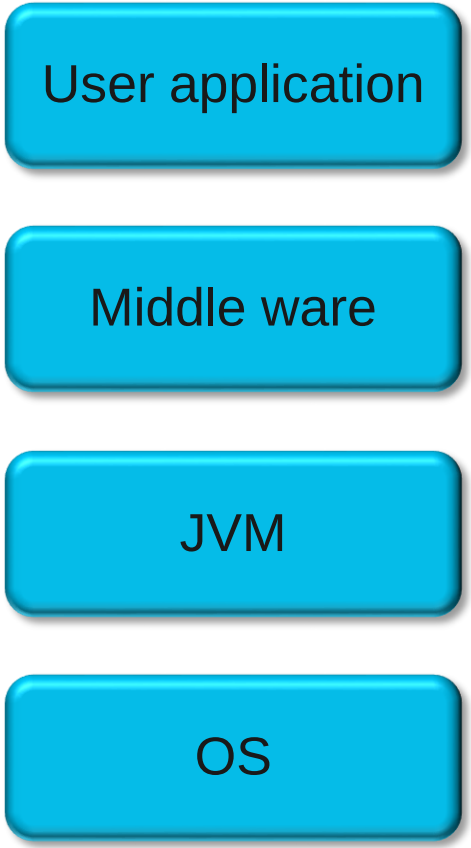
- I've been involved with virtual machine development at IBM since 2007 and am now the J9 Virtual Machine Team Lead. J9 is IBM's independent implementation of the JVM.
- I've represented IBM on both the JSR 292 ('invokedynamic') and JSR 335 ('lambda') expert groups and lead J9's implementation of both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

J9: IBM’s Java Virtual Machine

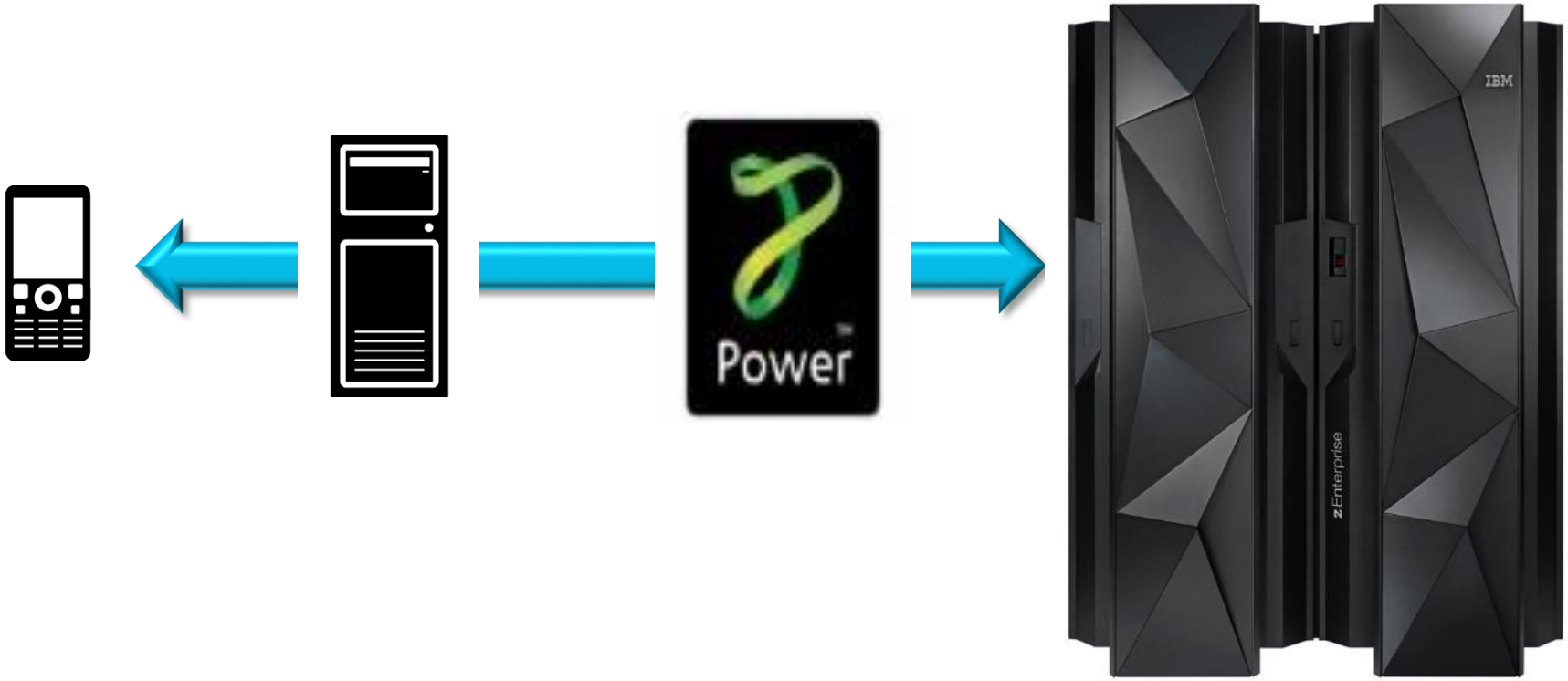
High performance

High reliability

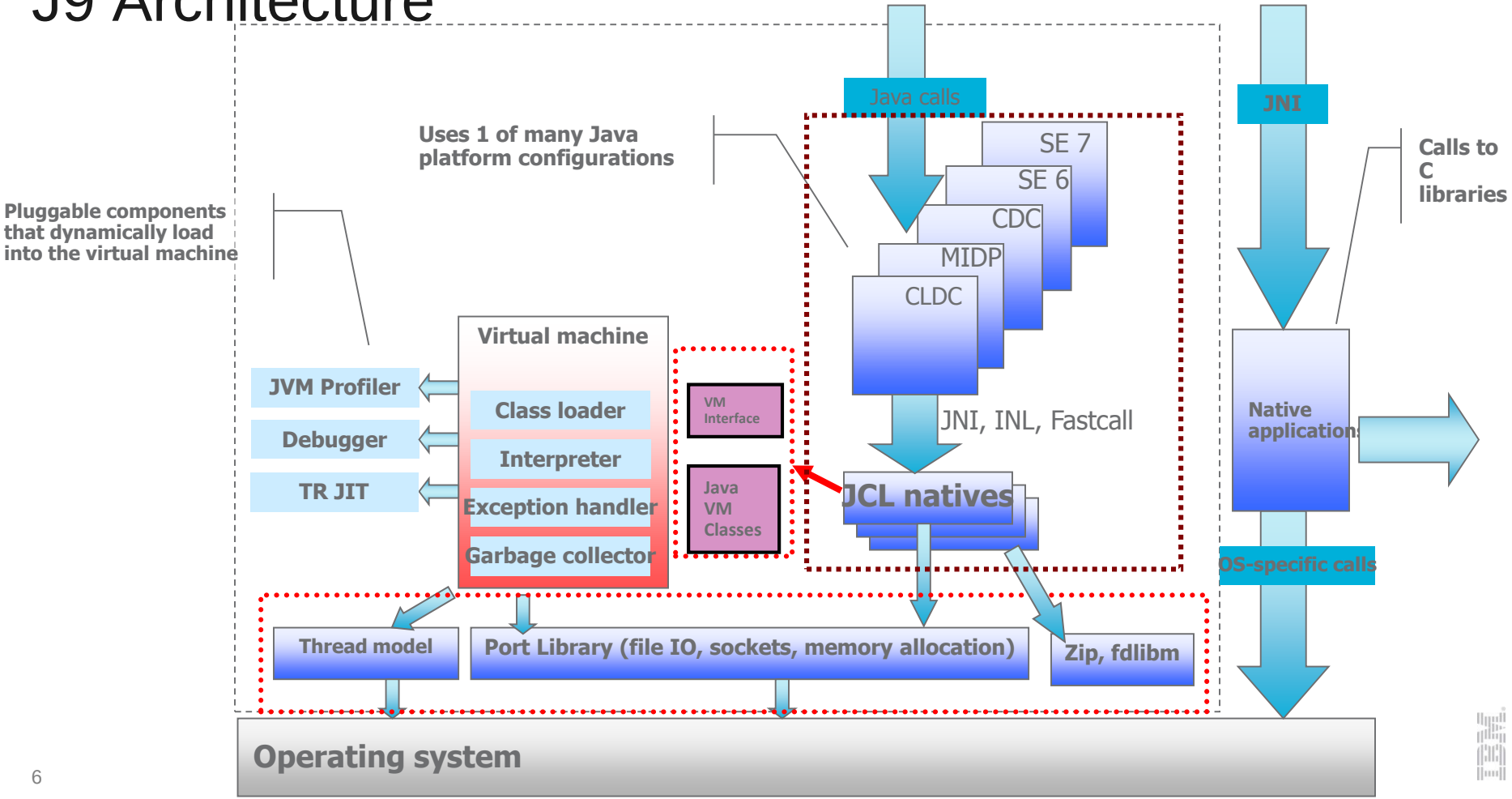
Serviceability



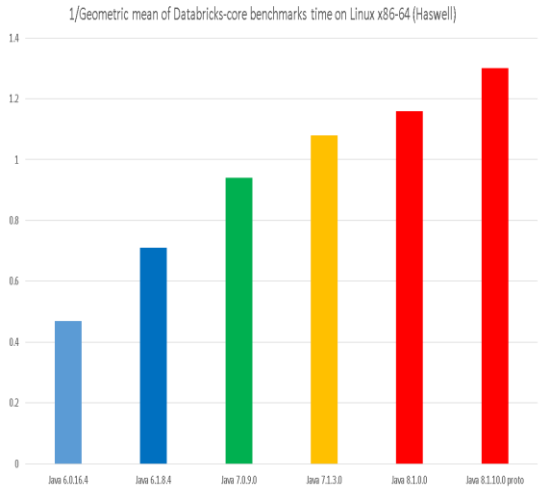
J9: IBM’s Java Virtual Machine scales



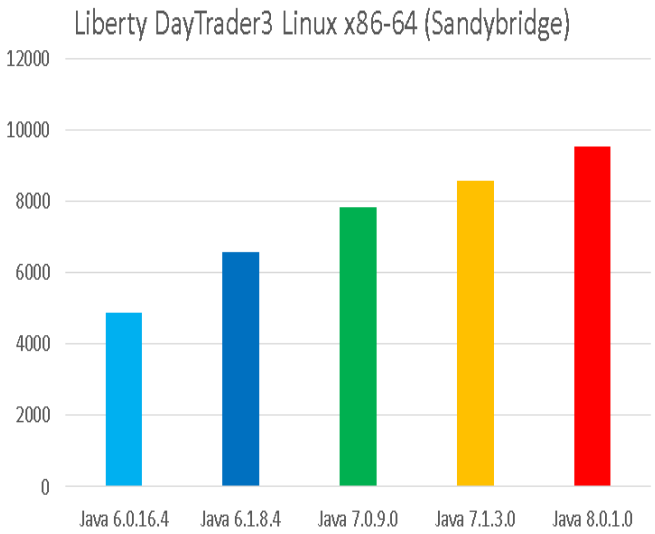
J9 Architecture



PERFORMANCE !



Apache Spark 1.4



Daytrader3 Linux Intel

Java6
SR16 FP4

Java 6.1
SR8 FP4

Java 7
SR9

Java 7.1
SR3

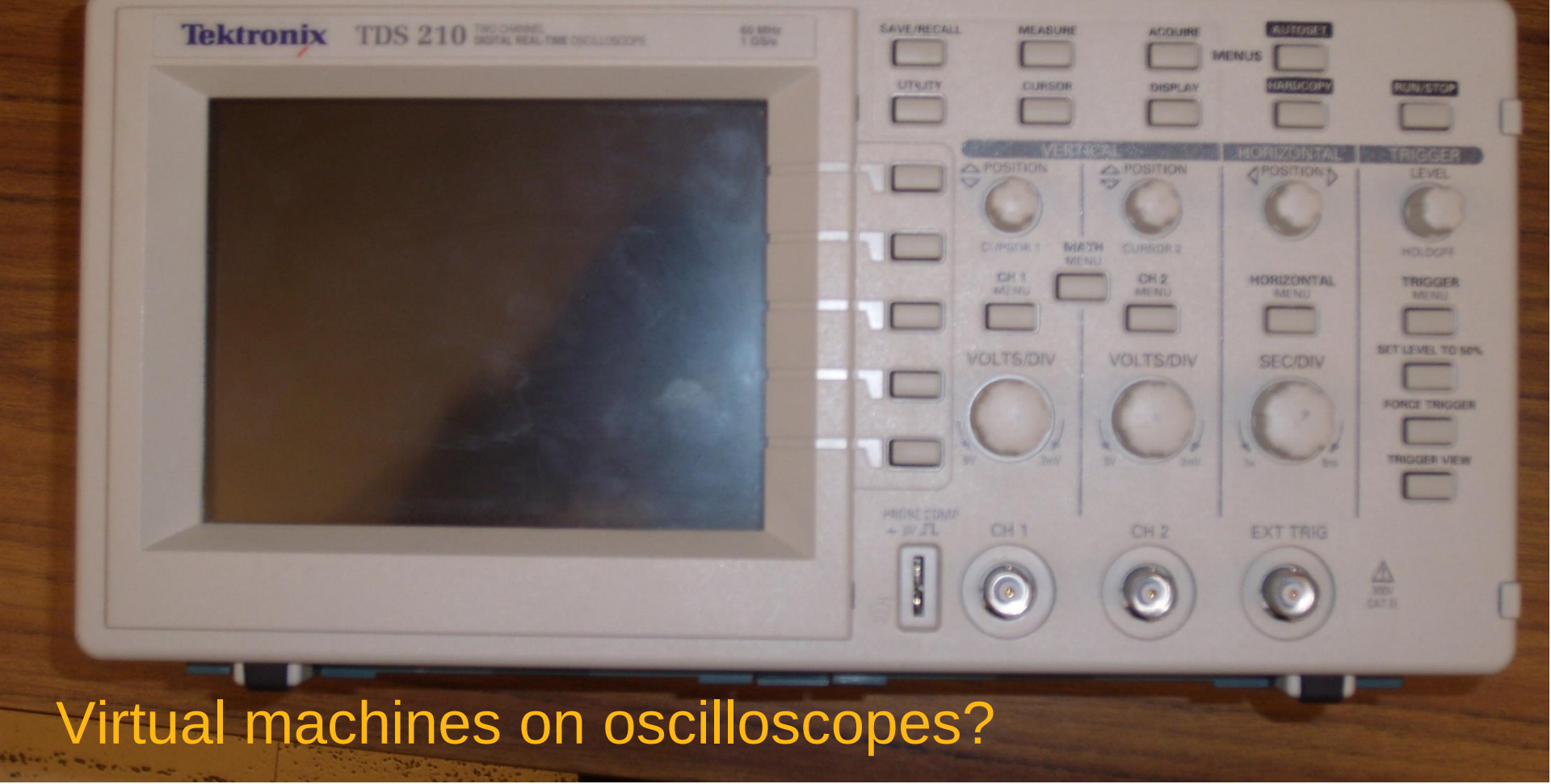
Java 8
SR1

File Edit Classes Applications Categories Methods Info Breakpoints

Object	Core	(AbtRun-ObsoleteAPI)	addToBeFinalized (public)
AbtAbstractCodePageConve	CLDT	(ACO-API)	finalize (public)
AbtAttributeMapping	CLIM	(CFS-API)	isOSObject (public)
AbtBase64Coder	CPM	CLDT-API	onFinalizeDo: (public)
AbtCharacterTranslator	AbtNlsKernelApp	(CLIM-API)	removeToBeFinalized (public)
AbtClassElementMapping...	DbgRuntimeFrameworr	(CT-Internal)	
AbtDependentsCollection	EsAsynchronousCallou	(EM-Internal)	
AbtDOMElementWrapper	K8Types	ES-API	
AbtDOMImplementation	SgmlSupport	ES-Internal	
public	Default: K8VMBuilderARI	instance	all

```
finalize
  "The receiver no longer has any referencers in the image
  and is being finalized. The default behavior is to do nothing."

  ^self
```

Envy/Developer: J9's Smalltalk Roots



Virtual machines on oscilloscopes?

“Lots” of ROM, very little RAM

https://upload.wikimedia.org/wikipedia/commons/7/7e/Tektronix_TDS210_Oscilloscope.jpg

Data dominates. If you've chosen the right data structures and organized things well, the algorithms will almost always be self-evident. Data structures, not algorithms, are central to programming.

– Rob Pike

You want me to interpret that?

4.1 The `ClassFile` Structure

A `class` file consists of a single `ClassFile` structure:

```
ClassFile {
    u4 magic;
    u2 minor_version;
    u2 major_version;
    u2 constant_pool_count;
    cp_info constant_pool[constant_pool_count-1];
    u2 access_flags;
    u2 this_class;
    u2 super_class;
    u2 interfaces_count;
    u2 interfaces[interfaces_count];
    u2 fields_count;
    field_info fields[fields_count];
    u2 methods_count;
    method_info methods[methods_count];
    u2 attributes_count;
    attribute_info attributes[attributes_count];
}
```

You want me to interpret that?

Constant offsets

Fast access

Easy to find constant_pool start

Relative offsets

Variable sized arrays

Variable sized entries

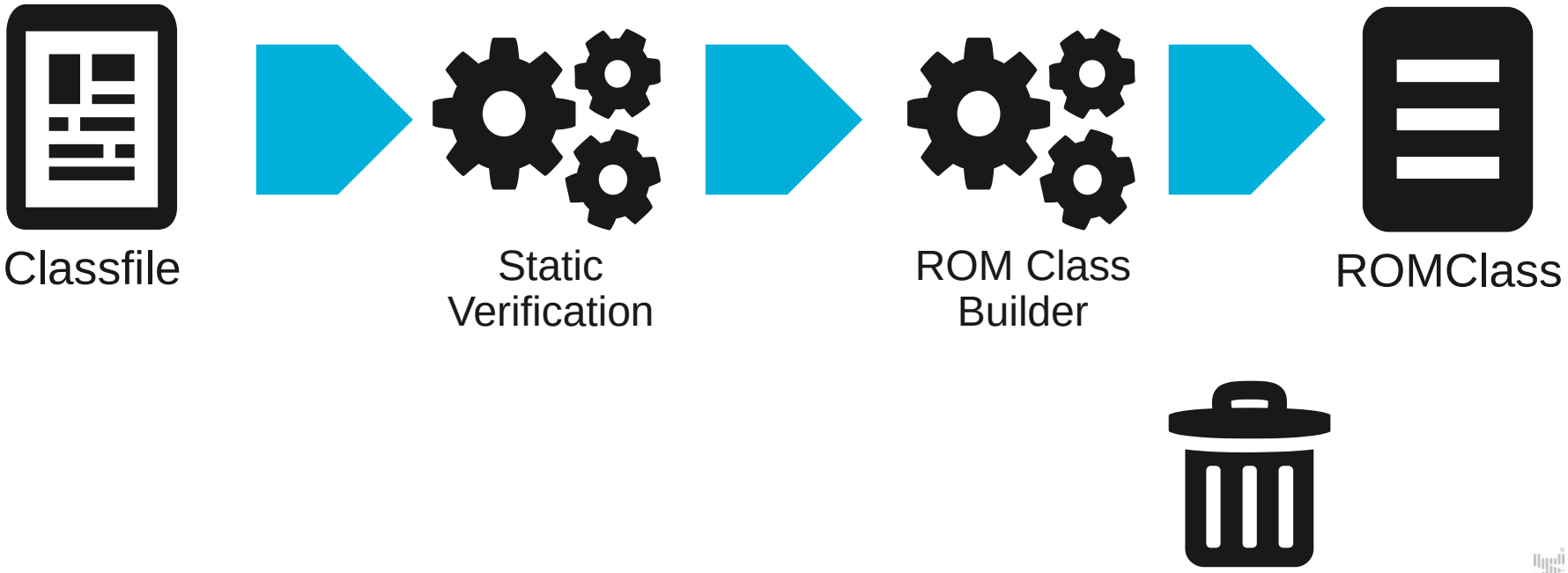
Slow access to specific index

4.1 The ClassFile Structure

A class file consists of a single ClassFile structure:

```
ClassFile {
    u4 magic;
    u2 minor_version;
    u2 major_version;
    u2 constant_pool_count;
    cp_info constant_pool[constant_pool_count-1];
    u2 access_flags;
    u2 this_class;
    u2 super_class;
    u2 interfaces_count;
    u2 interfaces[interfaces_count];
    u2 fields_count;
    field_info fields[fields_count];
    u2 methods_count;
    method_info methods[methods_count];
    u2 attributes_count;
    attribute_info attributes[attributes_count];
}
```

Conversion to ROM



Essential Classfile data

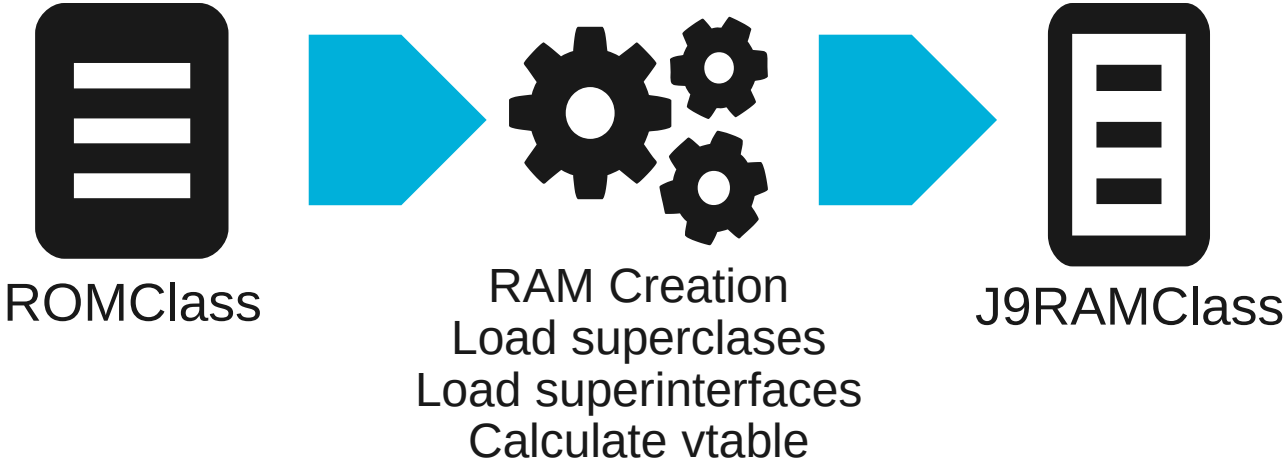
Position independent (SRP)

Less variable than a Classfile

Still variable, but structured

```
J9ROMClass at 0x2b768f835030 {
  Fields for J9ROMClass:
        0x0: U32 romSize = 0x000006B8 (1720)
        0x4: U32 singleScalarStaticCount = 0x00000000 (0)
        0x8: J9SRP(struct J9UTF8) className = !j9utf8 0x00002B768F83554E
        0xc: J9SRP(struct J9UTF8) superclassName = !j9utf8 0x0000000000000000
        0x10: U32 modifiers = 0x00000021 (33)
        0x14: U32 extraModifiers = 0x08A00000 (144703488)
        0x18: U32 interfaceCount = 0x00000000 (0)
        0x1c: J9SRP(J9SRP(struct J9UTF8)) interfaces = !j9utf8 0x00002B768F8355D6
        0x20: U32 romMethodCount = 0x0000000D (13)
        0x24: J9SRP(struct J9ROMMethod) romMethods = !j9rommethod 0x00002B768F8351A8
        0x28: U32 romFieldCount = 0x00000000 (0)
        0x2c: J9SRP(struct J9ROMFieldShape) romFields = !j9romfieldshape 0x00002B768F835150
        0x30: U32 objectStaticCount = 0x00000000 (0)
        0x34: U32 doubleScalarStaticCount = 0x00000000 (0)
        0x38: U32 ramConstantPoolCount = 0x00000011 (17)
        0x3c: U32 romConstantPoolCount = 0x00000011 (17)
        0x40: J9WSRP(U8) intermediateClassData = 0xB8 (184)
        0x48: U32 intermediateClassDataLength = 0x000006B8 (1720)
        0x4c: U32 instanceShape = 0x0000000E (14)
        0x50: J9SRP(U32) cpShapeDescription = 0x99991AA0 (2576947872)
        0x54: J9SRP(struct J9UTF8) outerClassName = !j9utf8 0x0000000000000000
        0x58: U32 memberAccessFlags = 0x00000000 (0)
        0x5c: U32 innerClassCount = 0x00000000 (0)
        0x60: J9SRP(J9SRP(struct J9UTF8)) innerClasses = !j9utf8 0x00002B768F8355D6
        0x64: U16 majorVersion = 0x0034 (52)
        0x66: U16 minorVersion = 0x0000 (0)
        0x68: U32 optionalFlags = 0x00000001 (1)
        0x6c: J9SRP(U32) optionalInfo = 0x0000013A (314)
        0x70: U32 maxBranchCount = 0x00000002 (2)
        0x74: U32 methodTypeCount = 0x00000000 (0)
        0x78: U32 bsmCount = 0x00000000 (0)
        0x7c: U32 callSiteCount = 0x00000000 (0)
        0x80: J9SRP callSiteData = !j9x 0x0000000000000000
        0x84: U32 classFileSize = 0x00000732 (1842)
        0x88: U32 classFileCPCount = 0x00000055 (85)
```

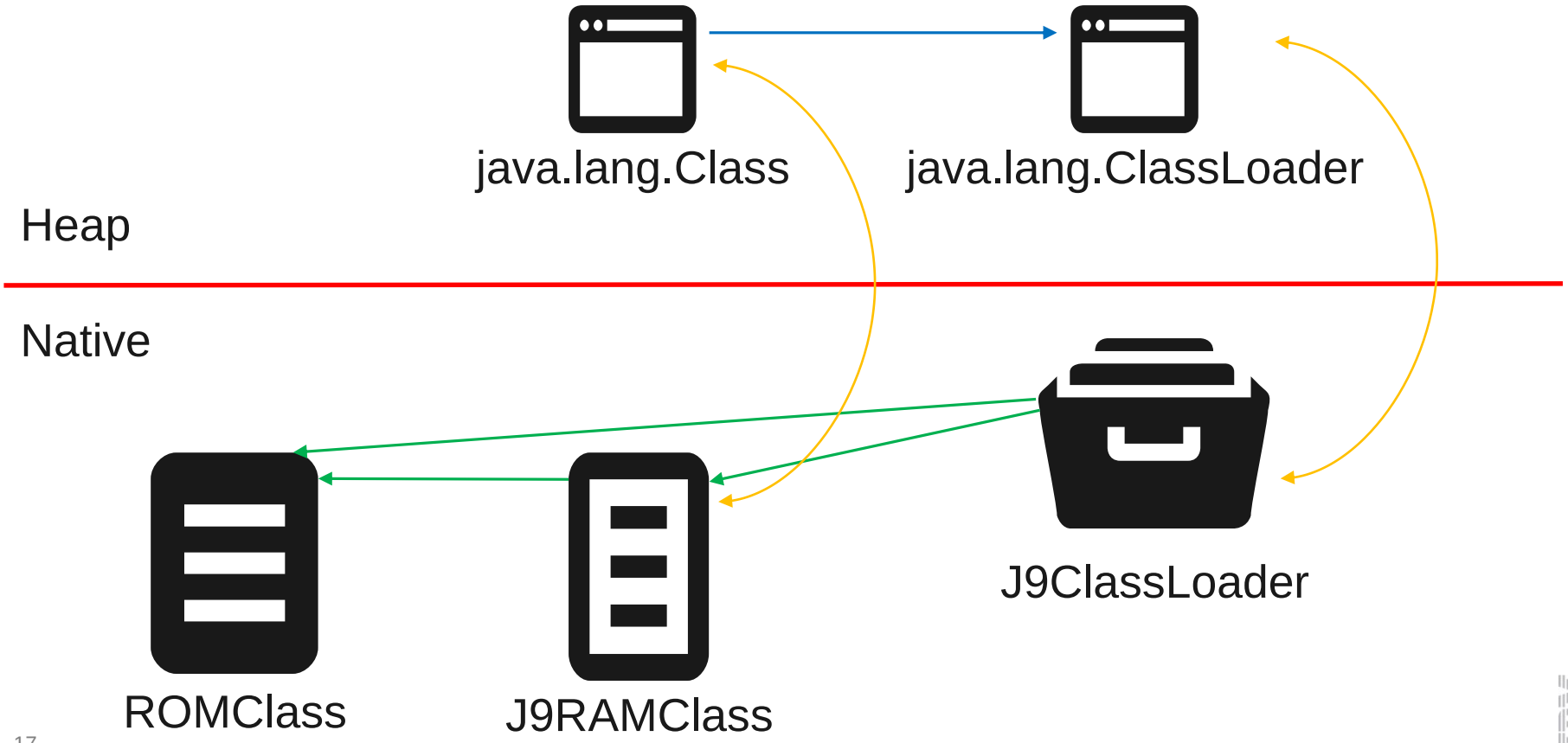
From ROM to RAM



- Live pointers, not SRPs
- Pointer to the ROMClass
- Live data
- Describes the shape of **this** class in **this** VM.
- Requires its entire hierarchy and interfaces to be loaded.

```
j9build@lnxem64tvm5:/team/danh/javaone/j9
J9Class at 0x636400 {
  Fields for J9Class:
        0x0: UDATA eyecatcher = 0x0000000099669966 (2573637990)
        0x8: struct J9ROMClass * romClass = !j9romclass 0x00002B768F835030
        0x10: struct J9Class ** superclasses = !j9x 0x0000000000636848
        0x18: UDATA classDepthAndFlags = 0x00000000001E0000 (1966080)
        0x20: U32 classDepthWithFlags = 0x00000000 (0)
        0x24: U32 classFlags = 0x00000000 (0)
        0x28: struct J9ClassLoader * classLoader = !j9classloader 0x00000000167C80B8
        0x30: j9object_t classObject = !j9object 0x00000000E0000200 // java/lang/Class
        0x38: volatile UDATA initializeStatus = 0x0000000000000001 (1)
        0x40: struct J9Method * ramMethods = !j9method 0x0000000000636590 // java/lang/Object.<init>()V
        0x48: UDATA * ramStatics = !j9x 0x0000000000000000
        0x50: struct J9Class * arrayClass = !j9class 0x000000000064D700 // [Ljava/lang/Object;
        0x58: struct J9Class * packedArrayClass = !j9class 0x0000000000000000
        0x60: UDATA totalInstanceSize = 0x0000000000000004 (4)
        0x68: UDATA packedDataSize = 0x0000000000000000 (0)
        0x70: struct J9ITable * lastITable = !j9itable 0x00002B768A3E4370
        0x78: UDATA * instanceDescription = !j9x 0x0000000000000001
        0x80: UDATA * instanceLeafDescription = !j9x 0x0000000000000001
        0x88: UDATA instanceHotFieldDescription = 0x0000000000000000 (0)
        0x90: struct J9Method * initializerCache = !j9method 0x0000000000000000
        0x98: UDATA romableAotITable = 0x00002B768CAAF64C (47788166149708)
        0xa0: UDATA packageID = 0x00002B768F835031 (47788213882929)
        0xa8: IDATA classPathIndex = 0x0000000000000000 (0)
        0xb0: struct J9Class * subclassTraversalLink = !j9class 0x0000000000751D00 // [Ljava/lang/Thread$State;
        0xb8: struct J9Class * subclassTraversalReverseLink = !j9class 0x0000000000637400 // java/lang/J9VMInternals
        0xc0: void ** iTable = !j9x 0x0000000000000000
        0xc8: struct J9PackedFieldsTable * packedFieldsTable = !j9packedfieldstable 0x0000000000000000
        0xd0: UDATA castClassCache = 0x0000000000000000 (0)
        0xd8: void ** jniIDs = !j9x 0x0000000016A72C70
        0xe0: UDATA lockOffset = 0x0000000000000004 (4)
        0xe8: UDATA newInstanceCount = 0x0000000000000BB8 (3000)
        0xf0: IDATA backfillOffset = 0x0000000000000008 (8)
        0xf8: struct J9Class * replacedClass = !j9class 0x0000000000000000
        0x100: UDATA finalizeLinkOffset = 0x0000000000000000 (0)
        0x108: struct J9Class * nextClassInSegment = !j9class 0x0000000000000000
        0x110: UDATA * ramConstantPool = !j9x 0x0000000000636730
        0x118: j9object_t * callSites = !j9x 0x0000000000000000
        0x120: j9object_t * methodTypes = !j9x 0x0000000000000000
        0x128: struct J9Method ** staticSplitMethodTable = !j9x 0x0000000000000000
        0x130: struct J9Method ** specialSplitMethodTable = !j9x 0x0000000000000000
        0x138: struct J9JITExceptionTable * jitMetaDataList = !j9jitexceptiontable 0x0000000000000000
        0x140: struct J9Class * gcLink = !j9class 0x0000000000000000
}
Class name: java/lang/Object
```

Classloading: tying it all together



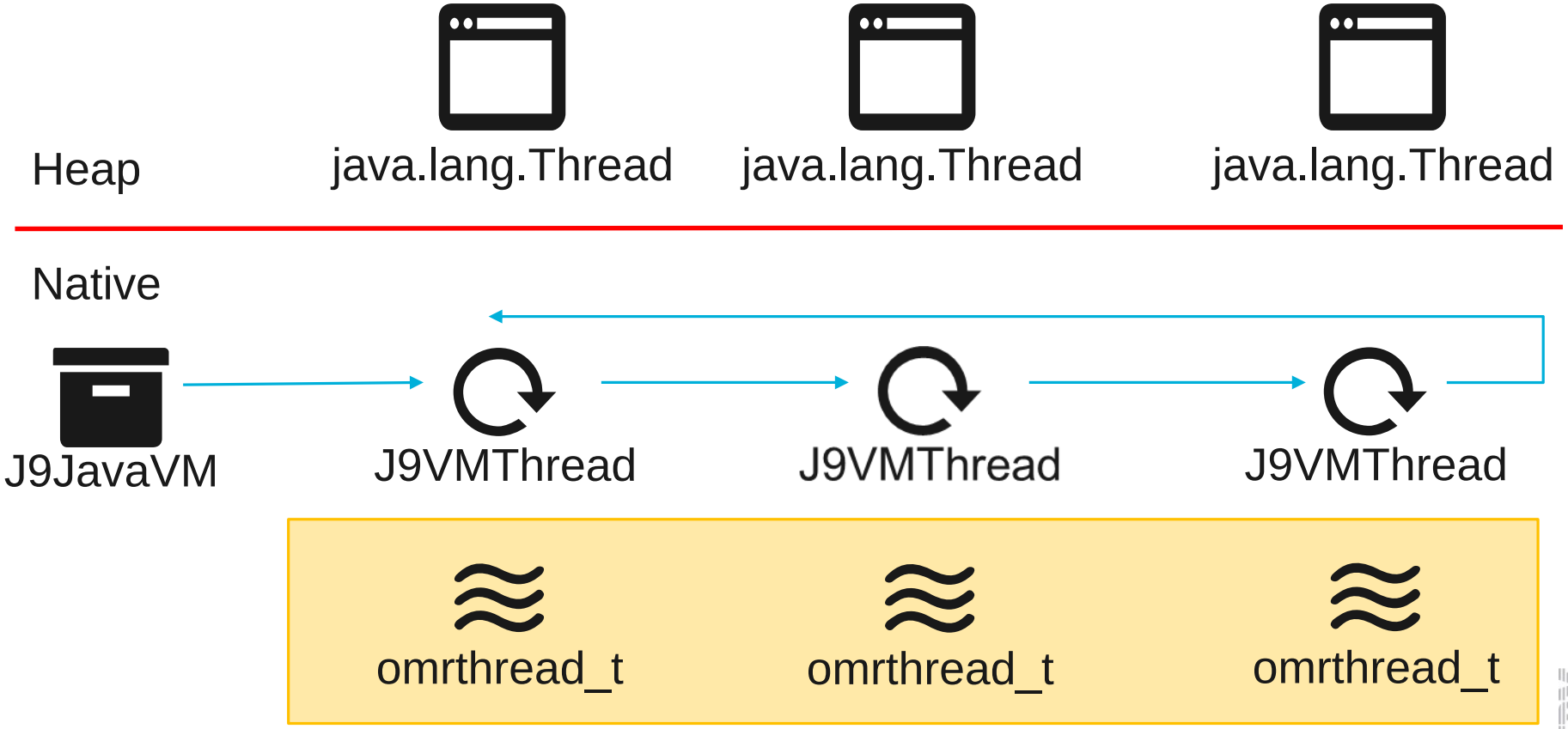
Bytecode loaded? Check

J9RAMClass

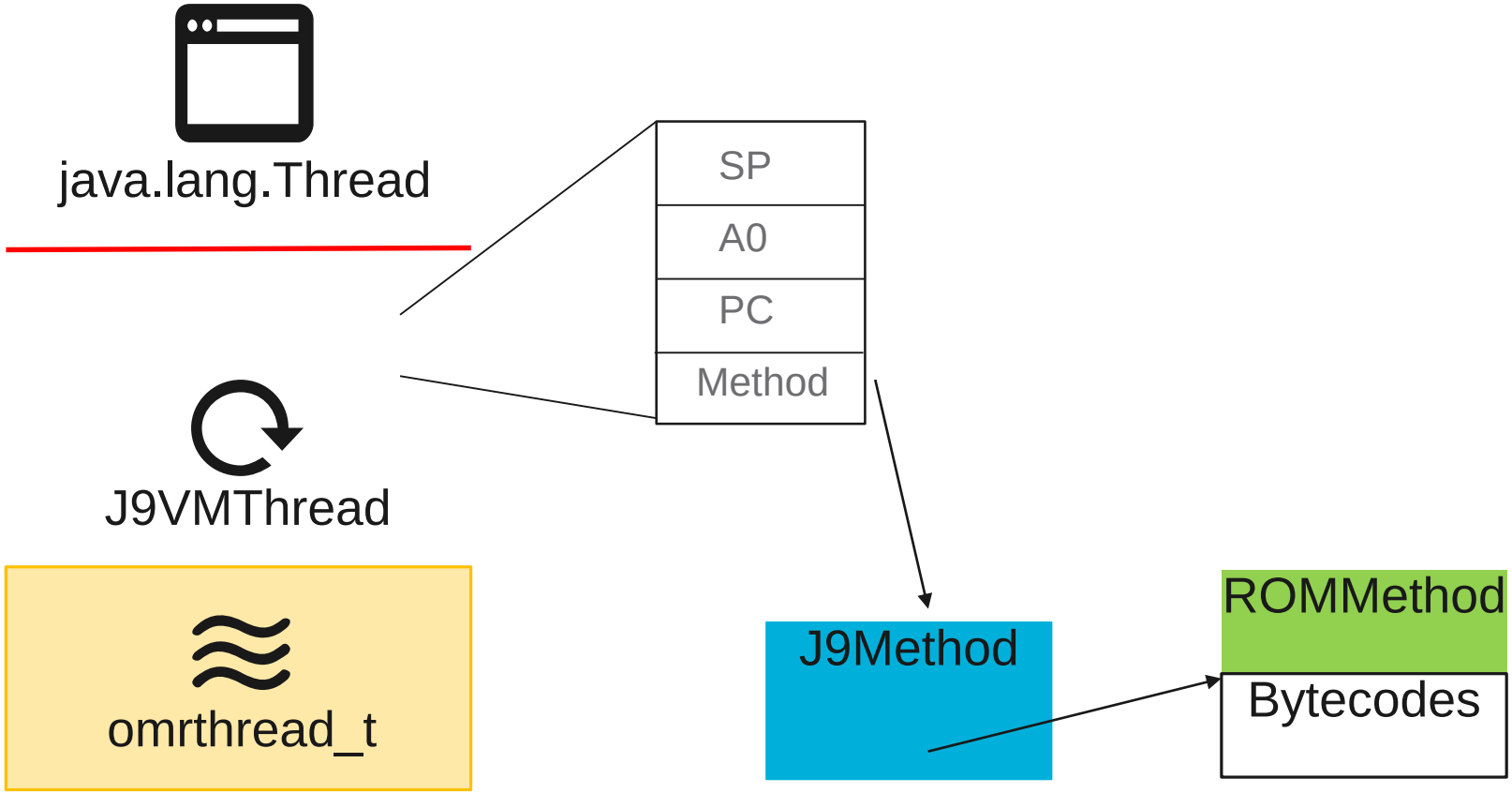
ROMClass

```
stack=2, locals=1, args_size=1
       0: getstatic     #2
       3: ldc           #3
       5: invokevirtual #4
       8: return
```

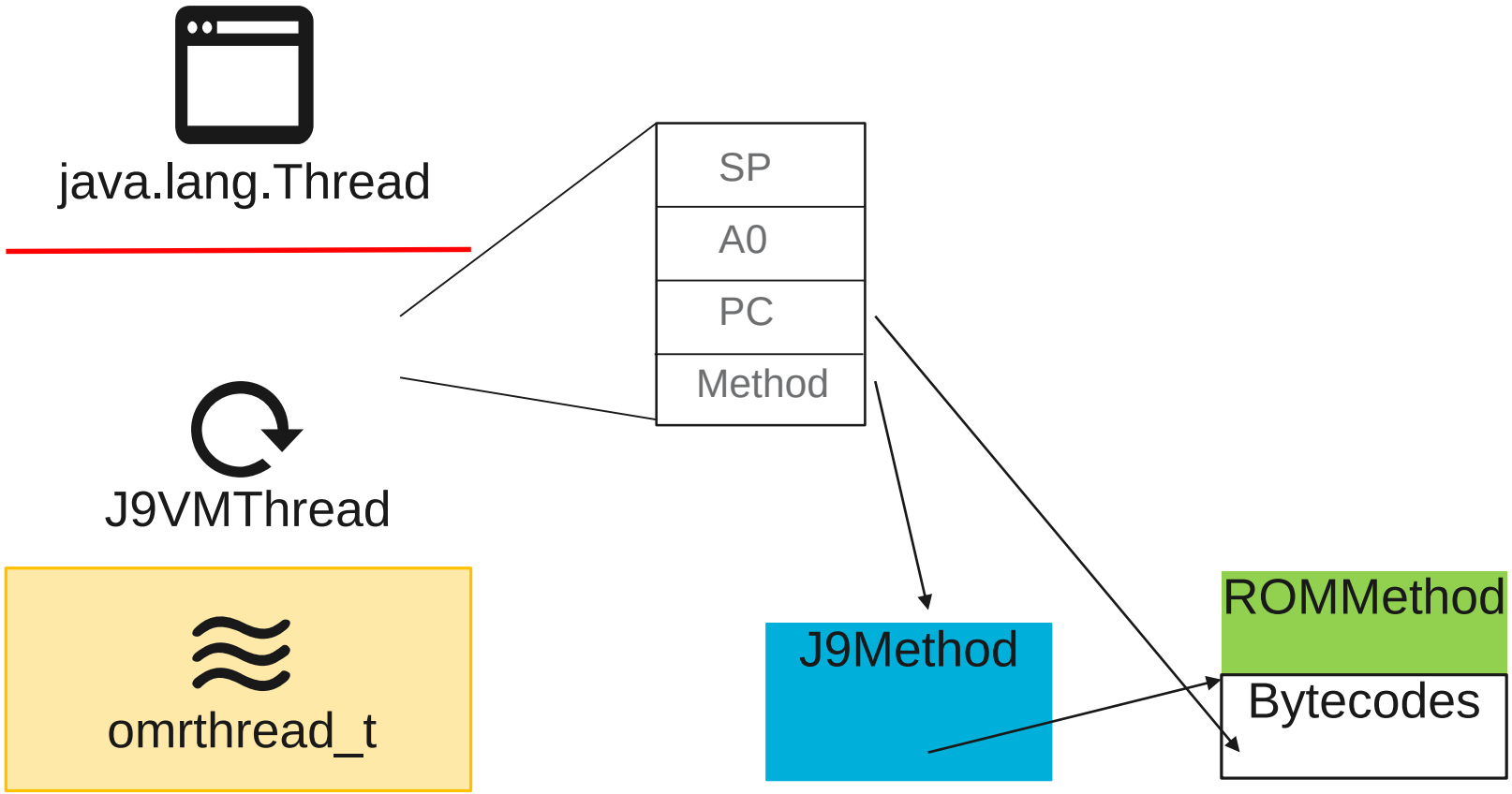
Threading


Heap
java.lang.Thread
java.lang.Thread
java.lang.Thread
Native
J9JavaVM
J9VMThread
J9VMThread
J9VMThread
omrthread_t
omrthread_t
omrthread_t

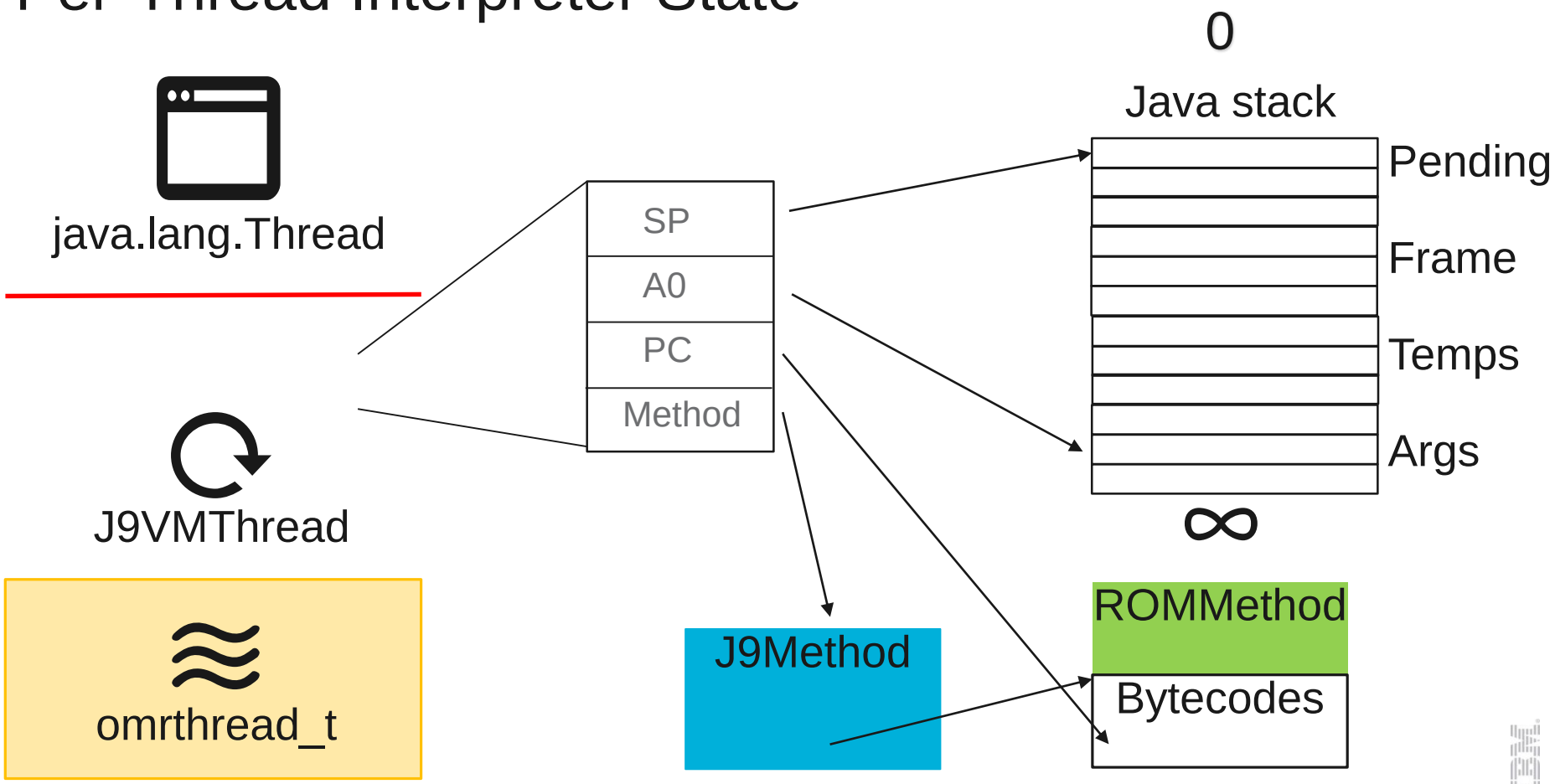
Per-Thread Interpreter State



Per-Thread Interpreter State



Per-Thread Interpreter State



Interpreter

- Written in C++
- Switch statement / computed goto
- Executes:
  - bytecodes
  - INLs
  - builds stack frames
- Transition to the JIT

Resource - VM_Common/vm/BytecodeInterpreter.hpp - Eclipse - C:\workspaces\git\jvm.29

File Edit Navigate Search Project Run Window Help

Quick Access | C/C++ | Git | Work Items | Team Synchronizing | Resource | Java

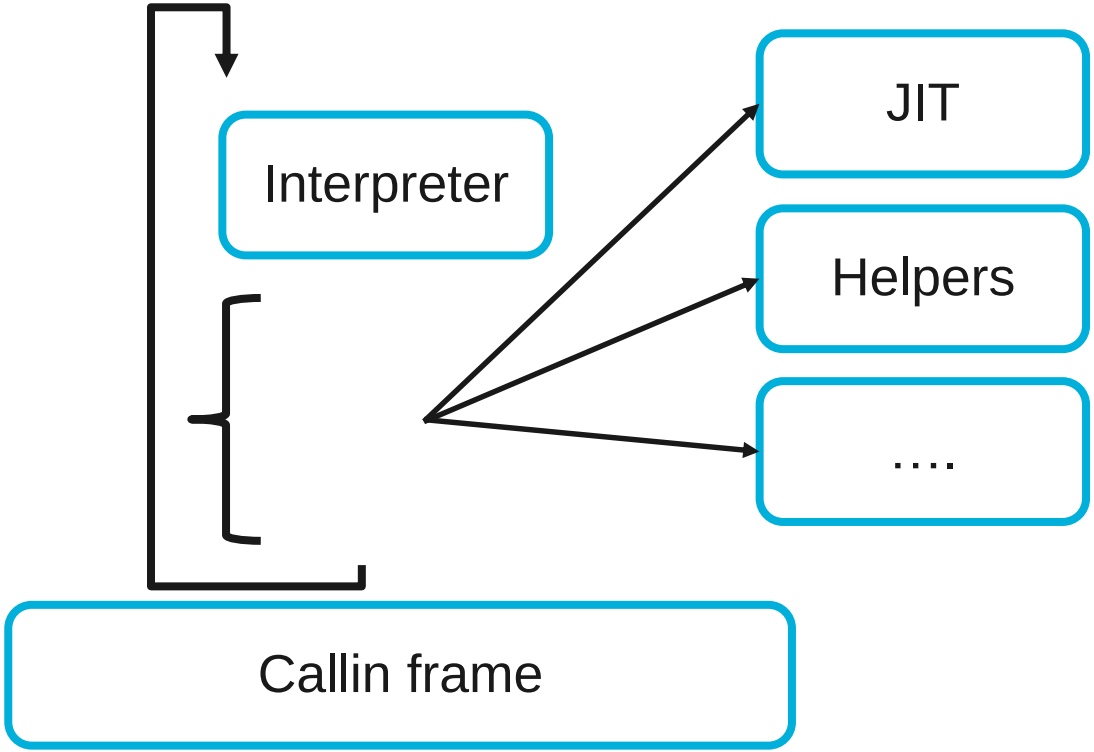
BytecodeInterpreter.hpp

```
/* ... => ..., value */
VMINLINE VM_BytecodeAction
aload(REGISTER_ARGS_LIST)
{
    U_8 index = _pc[1];
    _pc += 2;
    _sp -= 1;
    *_sp = *(_arg0EA - index);
    return EXECUTE_BYTECODE;
}

/* ... => ..., value */
VMINLINE VM_BytecodeAction
aload(REGISTER_ARGS_LIST, UDATA index) {
    _pc += 1;
    _sp -= 1;
    *_sp = *(_arg0EA - index);
    return EXECUTE_BYTECODE;
}

/* ..., value => ... */
VMINLINE VM_BytecodeAction
astore(REGISTER_ARGS_LIST)
{
```

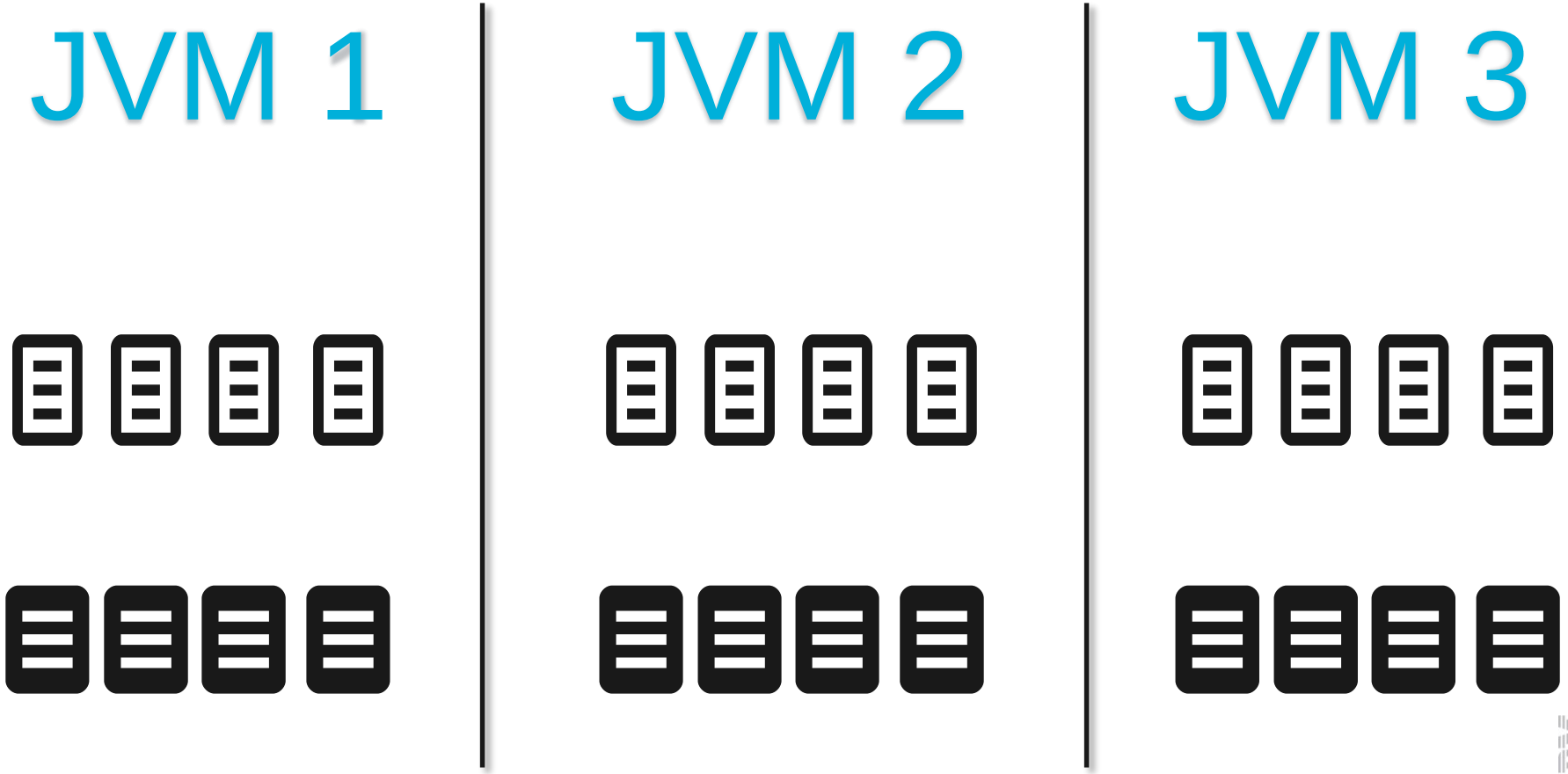
Not connected | win_x86

Writable | Insert | <No Current Work> | 0 Hits

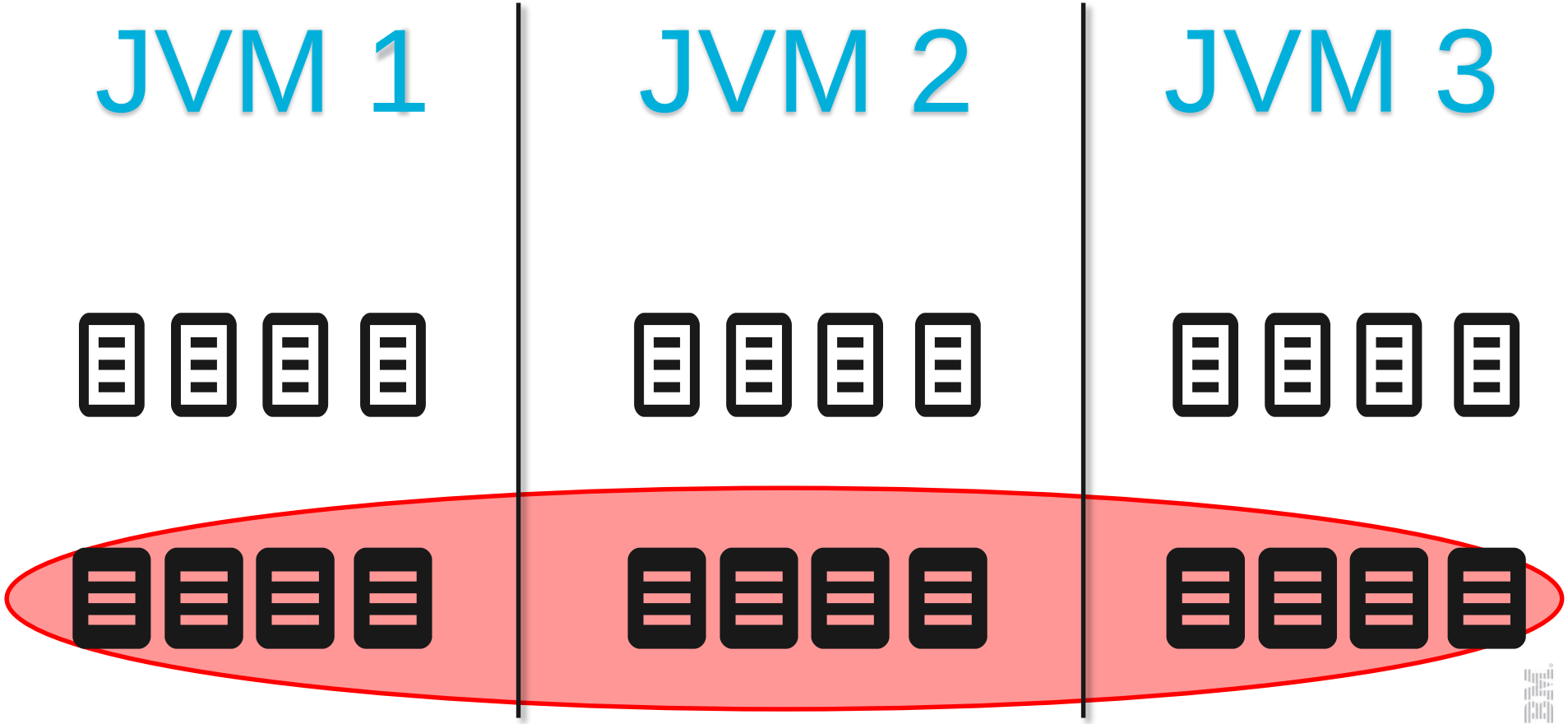
Primordial Call-in frame



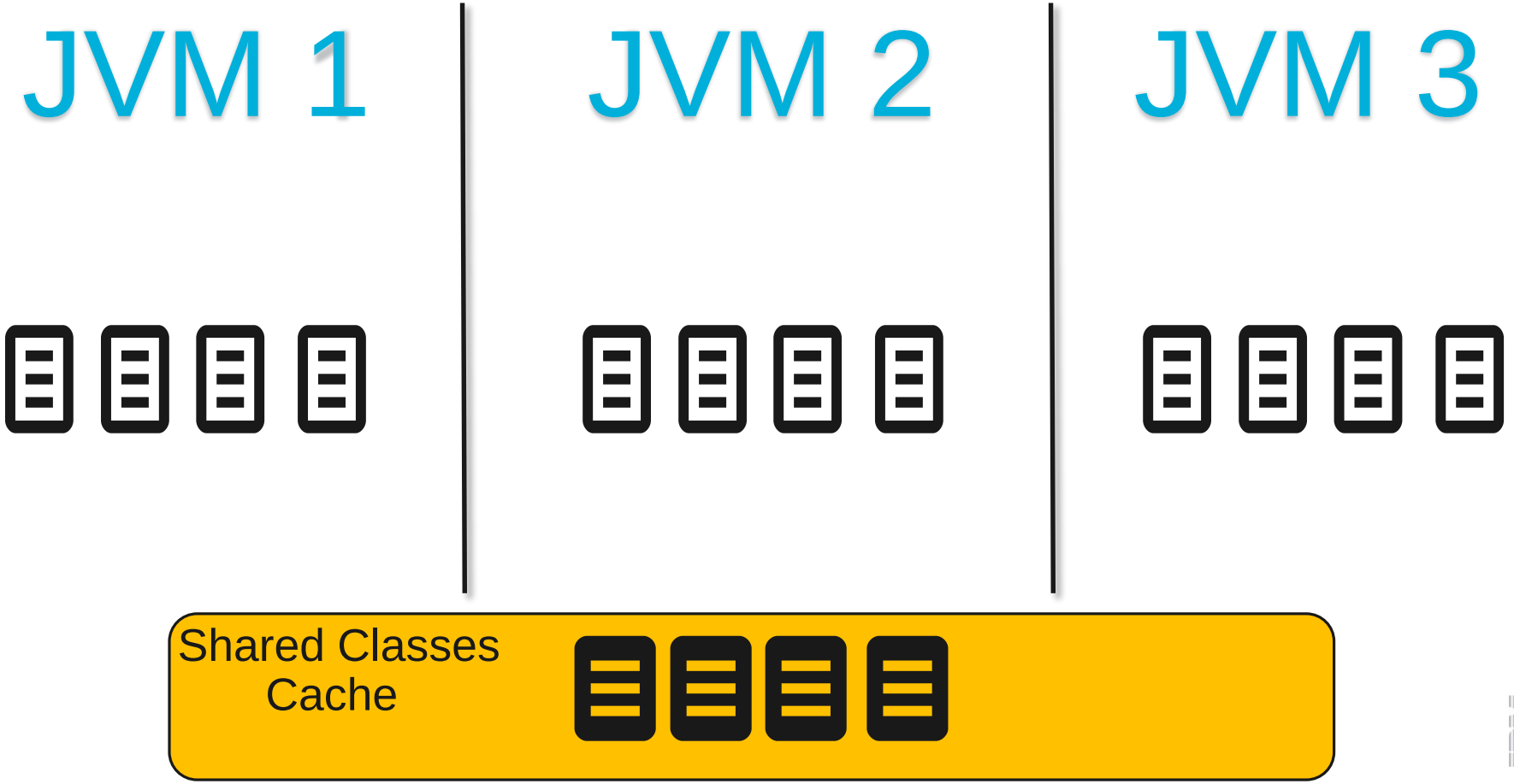
SharedClasses: ROM pays off



SharedClasses: ROM pays off



SharedClasses: ROM pays off

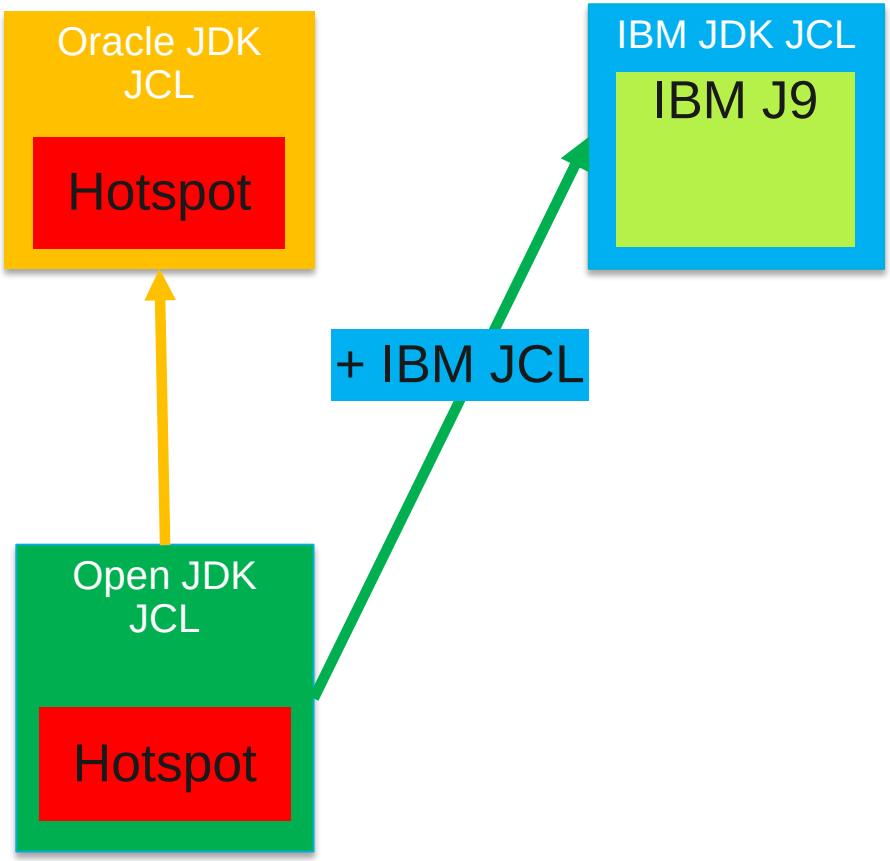


SharedClasses: “dynamic” AOT

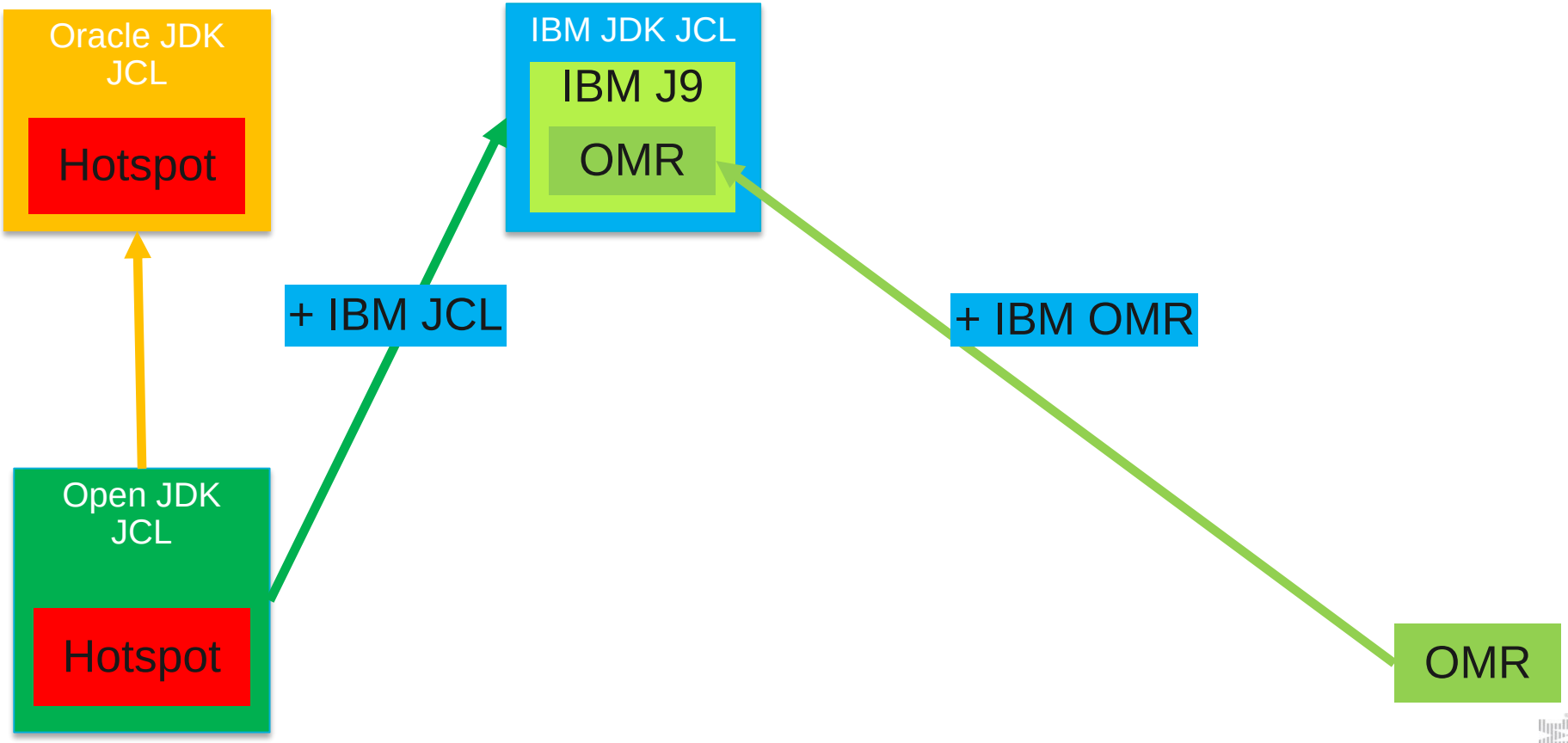
Diagnostic tools

- Diagnostic information (-Xdump)
  - Java dump (javacore)
  - Heap dump
  - System dump (core)
  - JIT dump
- Tracepoints (-Xtrace)
- bin/jdmpview –core <corefile>

J9: Meet IBM's OpenJDK distro



J9: Meet IBM's OpenJDK **& OMR** distro



Eclipse OMR Mission

Build an open reusable language runtime foundation for cloud platforms

- To accelerate advancement and innovation
- In full cooperation with existing language communities
- Engaging a diverse community of people interested in language runtimes
  - Professional developers
  - Researchers
  - Students
  - Hobbyists

Eclipse OMR
Created March 2016

http://www.eclipse.org/omr
https://github.com/eclipse/omr
https://developer.ibm.com/open/omr/

Dual License:
Eclipse Public License V1.0
Apache 2.0

Users and contributors very welcome
https://github.com/eclipse/omr/blob/master/CONTRIBUTING.md

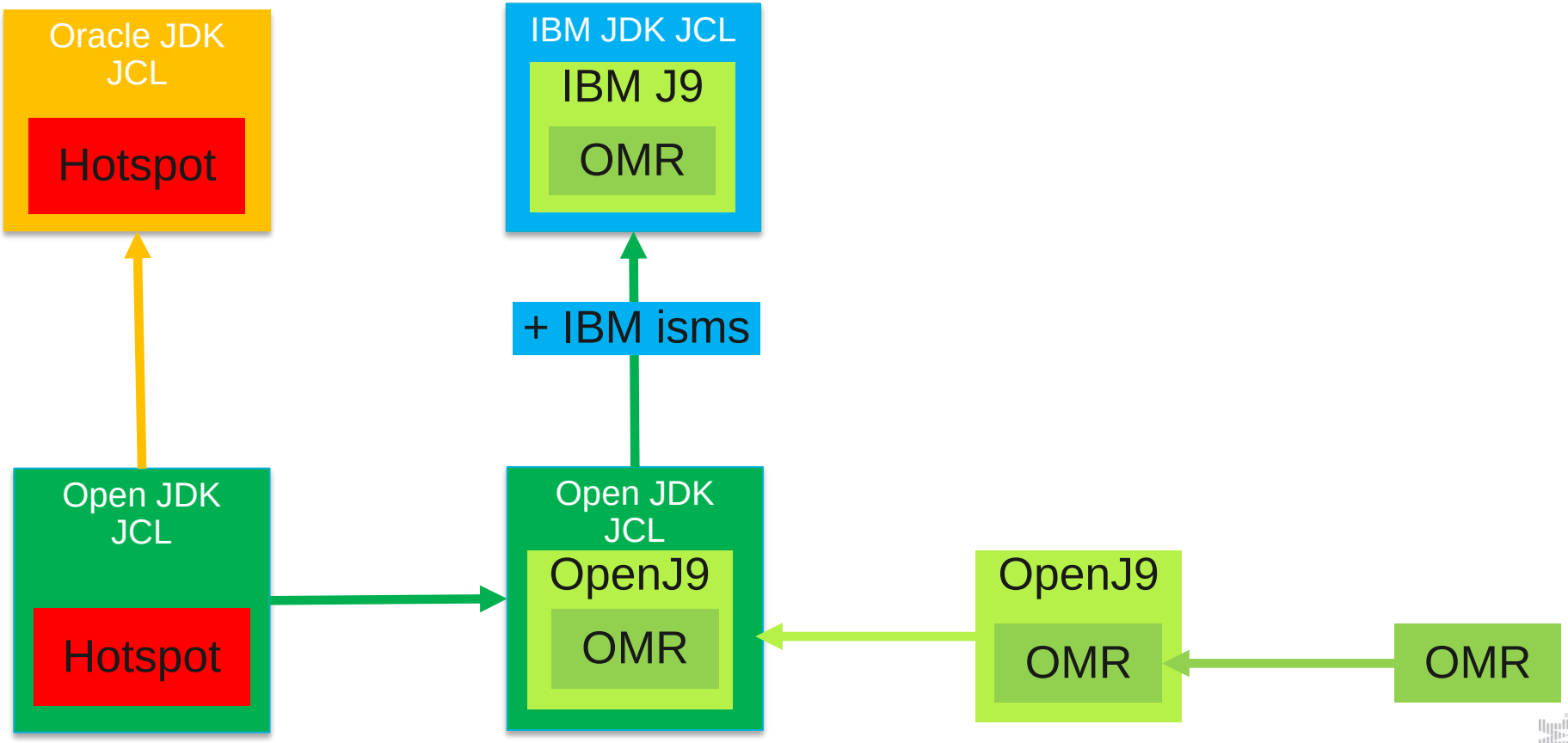
OMR components

port	platform abstraction (porting) library
thread	cross platform pthread-like threading library
vm	APIs to manage per-interpreter and per-thread contexts
gc	garbage collection framework for managed heaps
compiler	extensible compiler framework
jitbuilder	WIP project to simplify bring up for a new JIT compiler
omrtrace	library for publishing trace events for monitoring/diagnostics
omrsigcompat	signal handling compatibility library
example	demonstration code to show how a language runtime might consume OMR components, also used for testing
fvtest	language independent test framework built on the example glue so that components can be tested outside of a language runtime, uses Google Test 1.7 framework

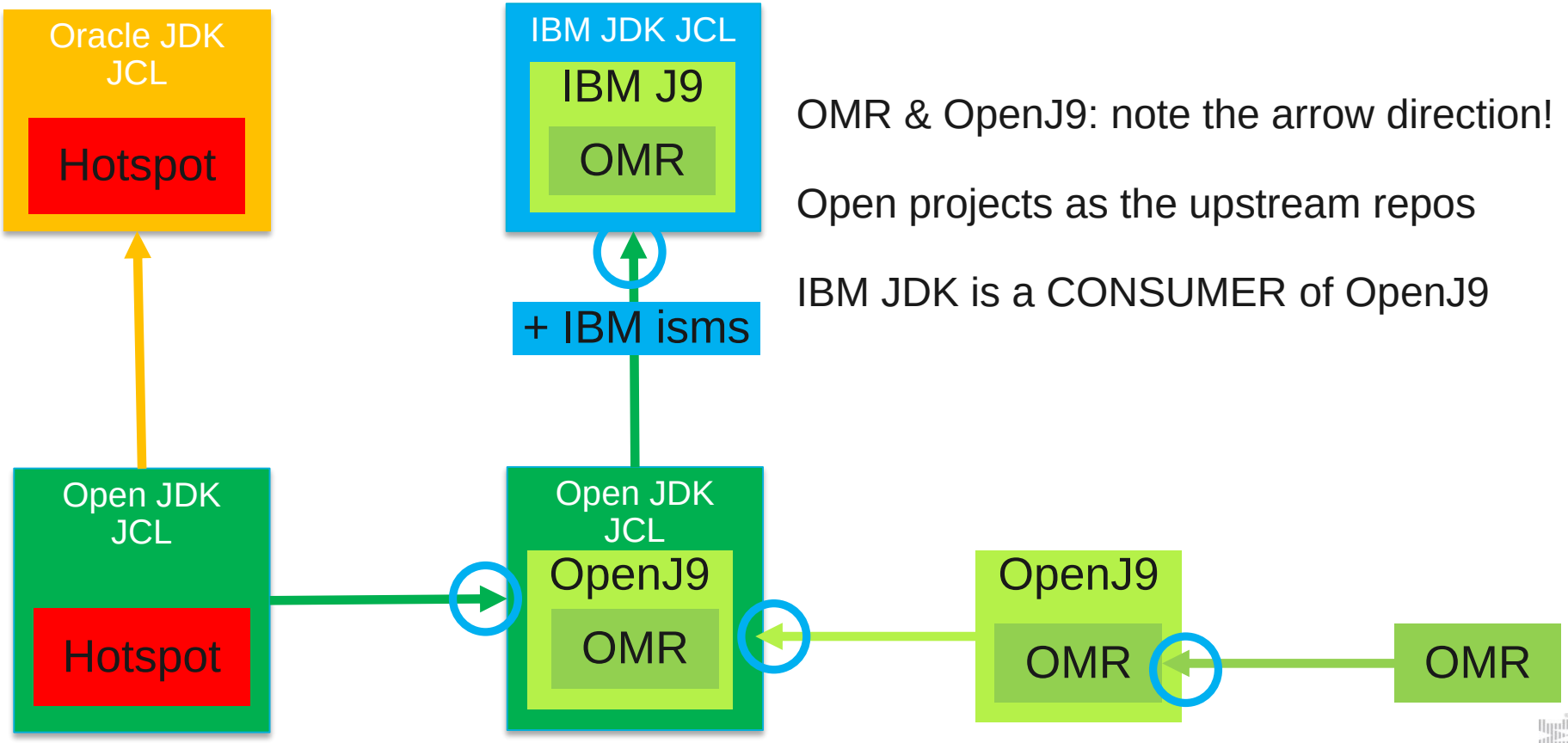
+ a few others

~800KLOC at this point, more components coming!

OpenJ9: Meet an OpenJDK & OMR distro



OpenJ9: Meet an OpenJDK & OMR distro



OMR & OpenJ9: note the arrow direction!

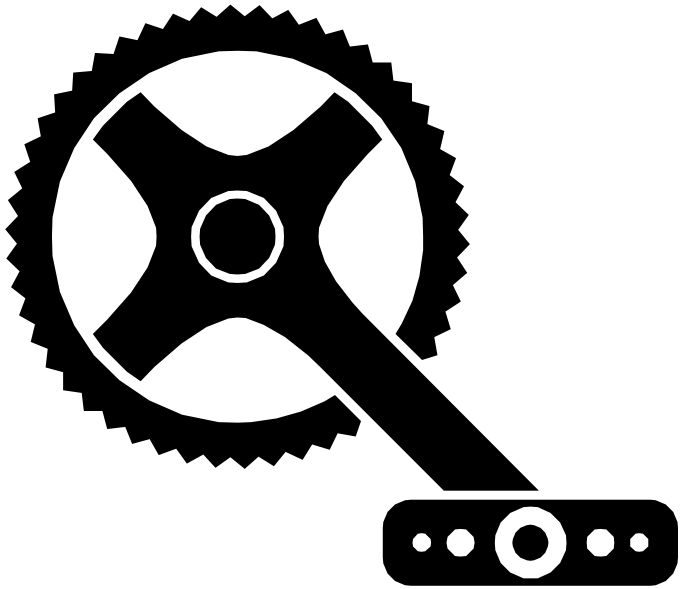
Open projects as the upstream repos

IBM JDK is a CONSUMER of OpenJ9

“We believe open ecosystems and partnerships are key to our future innovation.”

-- Ginni Rometty
(http://www.ibm.com/annualreport/2014/chairmans-letter.html)

Really? Why are you doing this?



- Collaboration
- Competition
- Polyglot
- Platforms

When

Soon

When

Goal: release concurrently with Java 9

Legal Notice

Open source

- IBM is open sourcing its J9 JVM technology
  - Includes Testarossa Just in Time (JIT) compiler

- Eclipse OMR project is leading edge: github.com/eclipse/omr
  - Project created March 7, 2016 : ~300KLOC
  - NEW! Compiler contributed September 16, 2016 : ~500KLOC

- Open J9 project is also coming
  - We’re working on it at same time as Java 9 development