Introduction to OpenJ9

Department name: Compiler Lab (Hong Kong Research Center)
Author’s name: Cheng Jin

Date: 2025/03/27

Security Level:

OpenJ9 or IBM J9?

1) Never/Seldom heard of it
2) Hmm, might be Java 9?

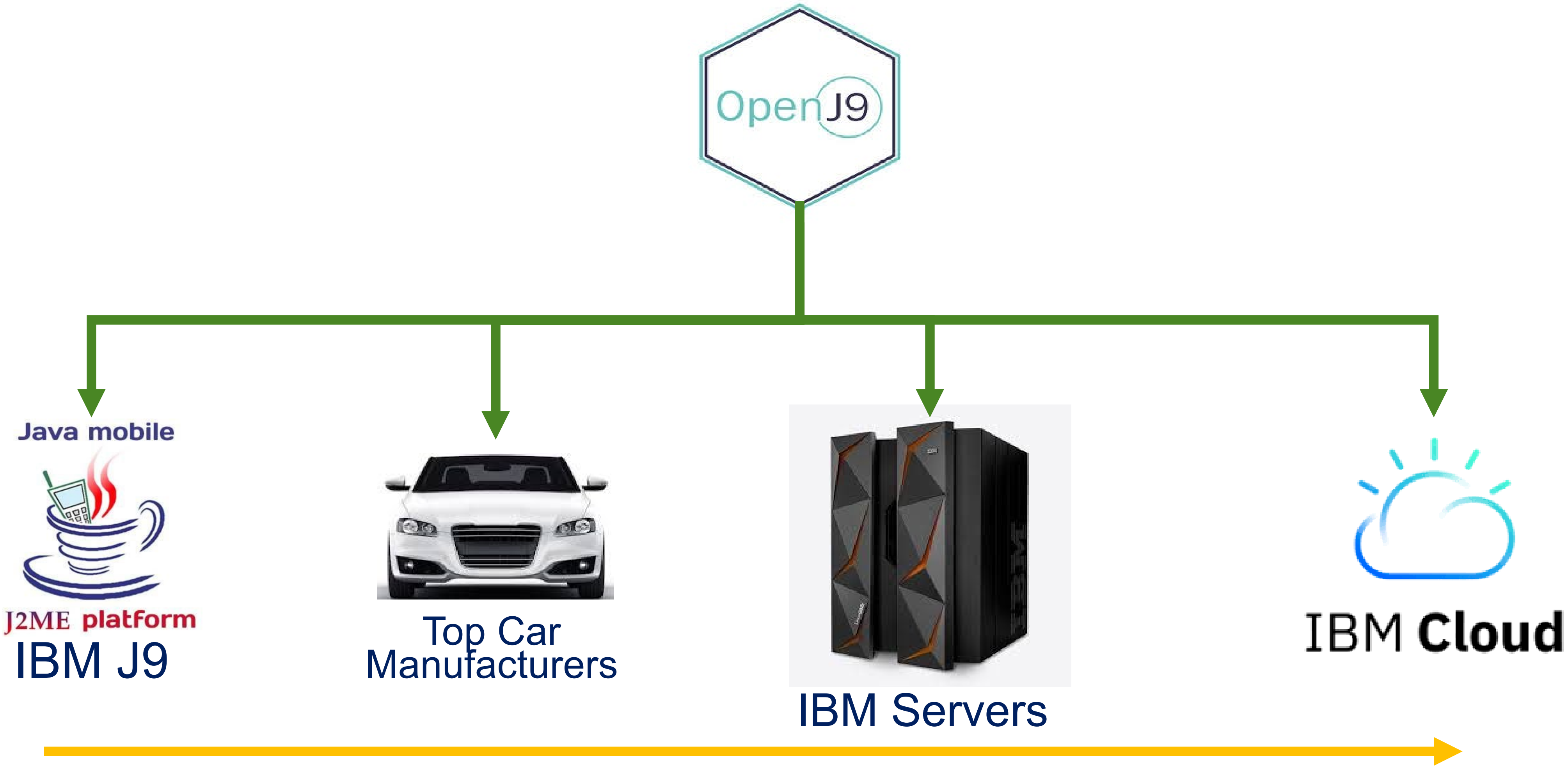
1) It’s been around with us for many years. e.g. credit card transactions

IBM Technology for Java Virtual Machine

2) J9 =! Java 9 (originated from K8 based on the naming convention of the Smalltalk source code)

How did the J9 in OpenJ9 get its name? (posted by Ronald Servant / Former IBM J9 R & D Manager)

Where is OpenJ9 ?

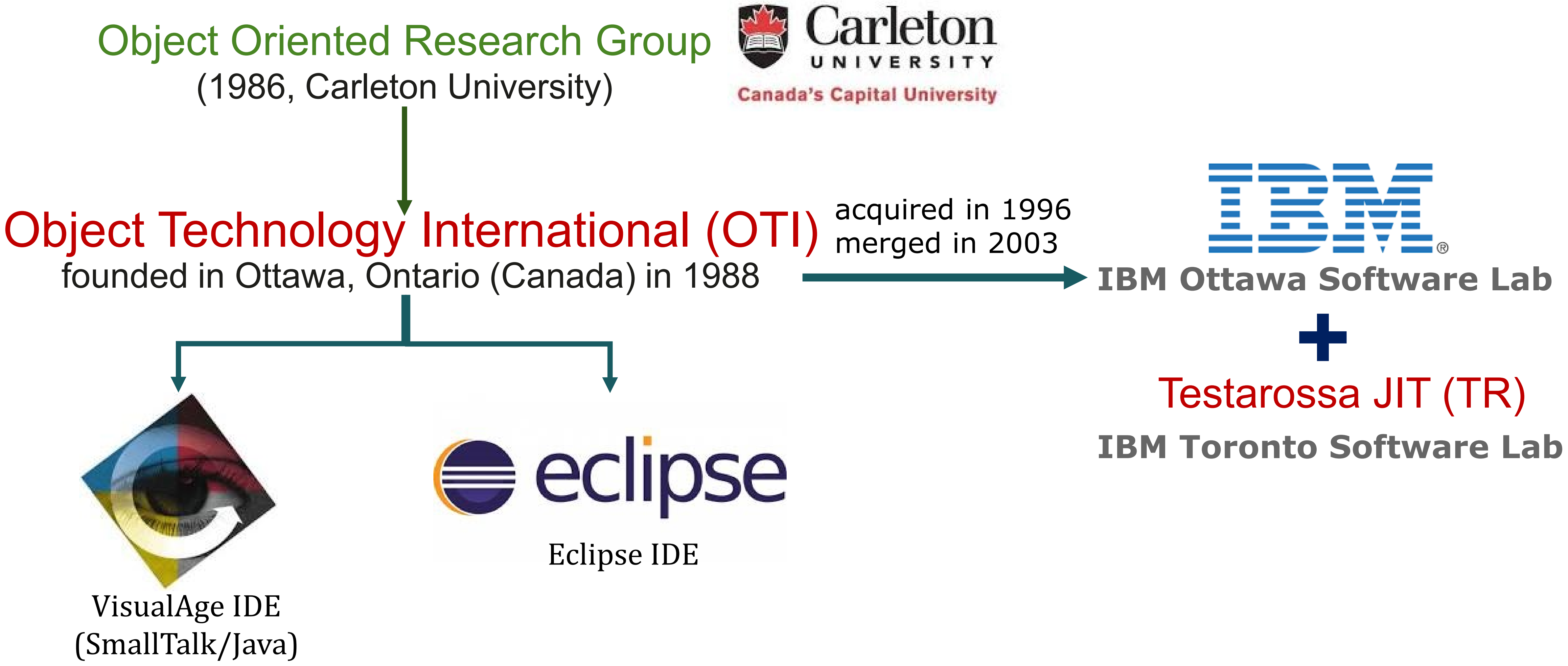


IBM J9

Top Car Manufacturers

IBM Servers

A Brief History of Open J9



Object Oriented Research Group
(1986, Carleton University)

Carleton
UNIVERSITY
Canada's Capital University

Object Technology International (OTI)
founded in Ottawa, Ontario (Canada) in 1988

acquired in 1996
merged in 2003

IBM Ottawa Software Lab

+

Testarossa JIT (TR)

IBM Toronto Software Lab

VisualAge IDE
(SmallTalk/Java)

Eclipse IDE

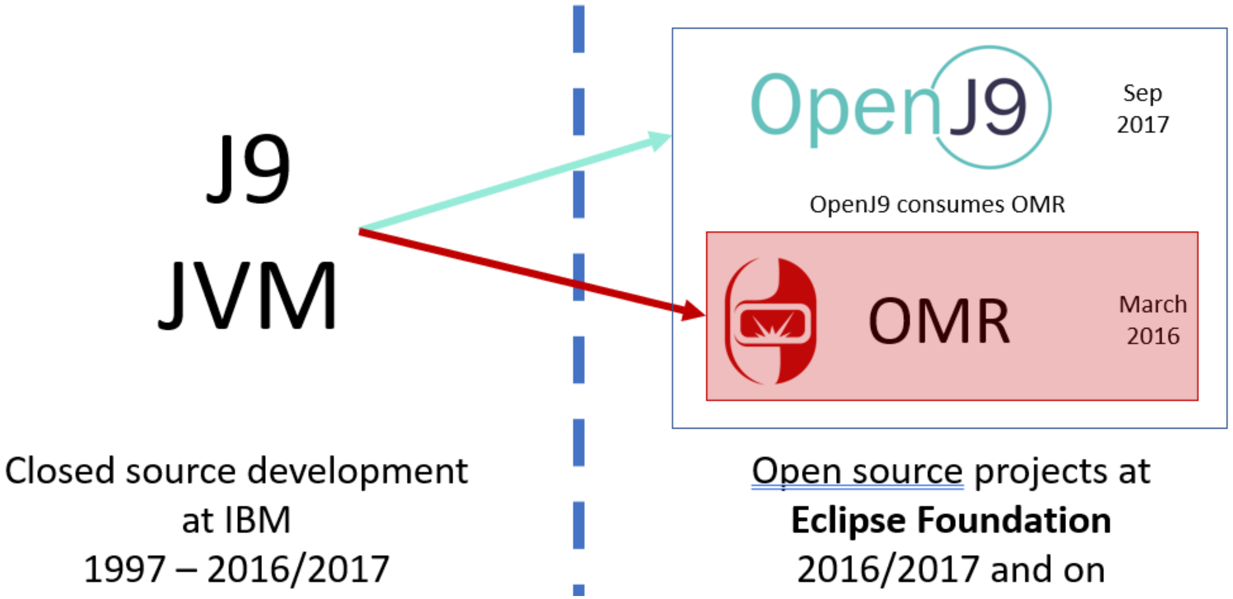
https://en.wikipedia.org/wiki/Object_Technology_International

A Brief History of OpenJ9 (Cont.)

- J9 VM developed independently by IBM (originally in the embedded world) as Enterprise middleware over 20 years (small footprint, fast startup & high performance)
- Donated to the Eclipse Foundation in 2017 as Eclipse OpenJ9 (more innovations in collaboration with the open source community)
- Transition from ENVY/Smalltalk to C/C++ in recent years to lower barrier for developers
- Dual License: Eclipse Public License v2.0 & Apache v2.0

https://en.wikipedia.org/wiki/OpenJ9

Eclipse OpenJ9 comes from IBM J9 JVM



J9 JVM

OpenJ9 Sep 2017

OpenJ9 consumes OMR

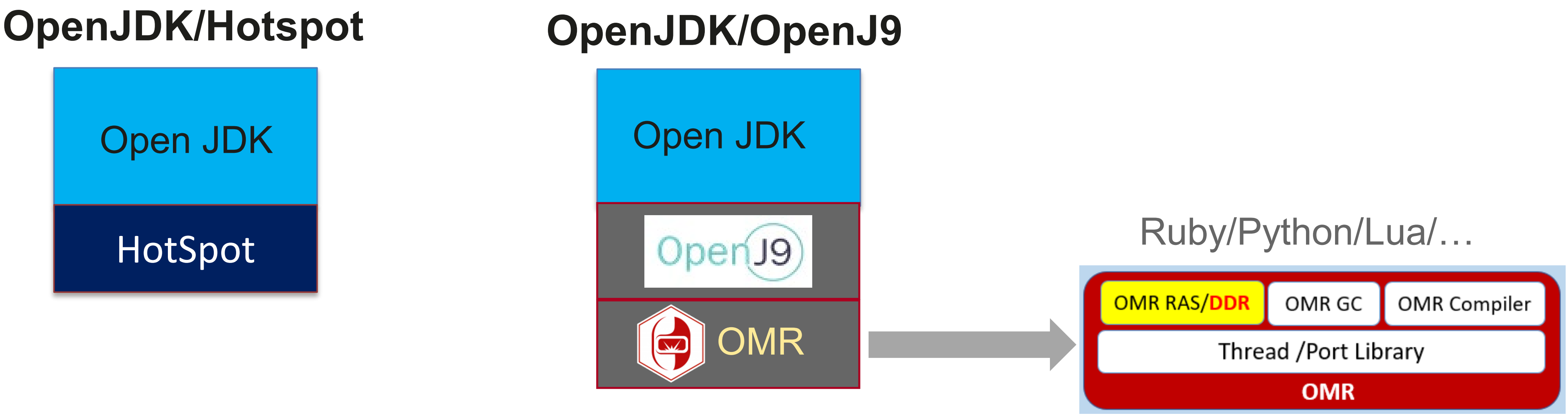
OMR March 2016

Closed source development at IBM 1997 – 2016/2017

Open source projects at **Eclipse Foundation** 2016/2017 and on

Java 7 (IBM J9/Smalltalk) → Java 8 (OpenJ9/C&C++)

What is OpenJ9?



OpenJDK/Hotspot

Open JDK

HotSpot

OpenJDK/OpenJ9

Open JDK

OpenJ9

OMR

Ruby/Python/Lua/…

OMR RAS/DDR | OMR GC | OMR Compiler

Thread /Port Library

OMR

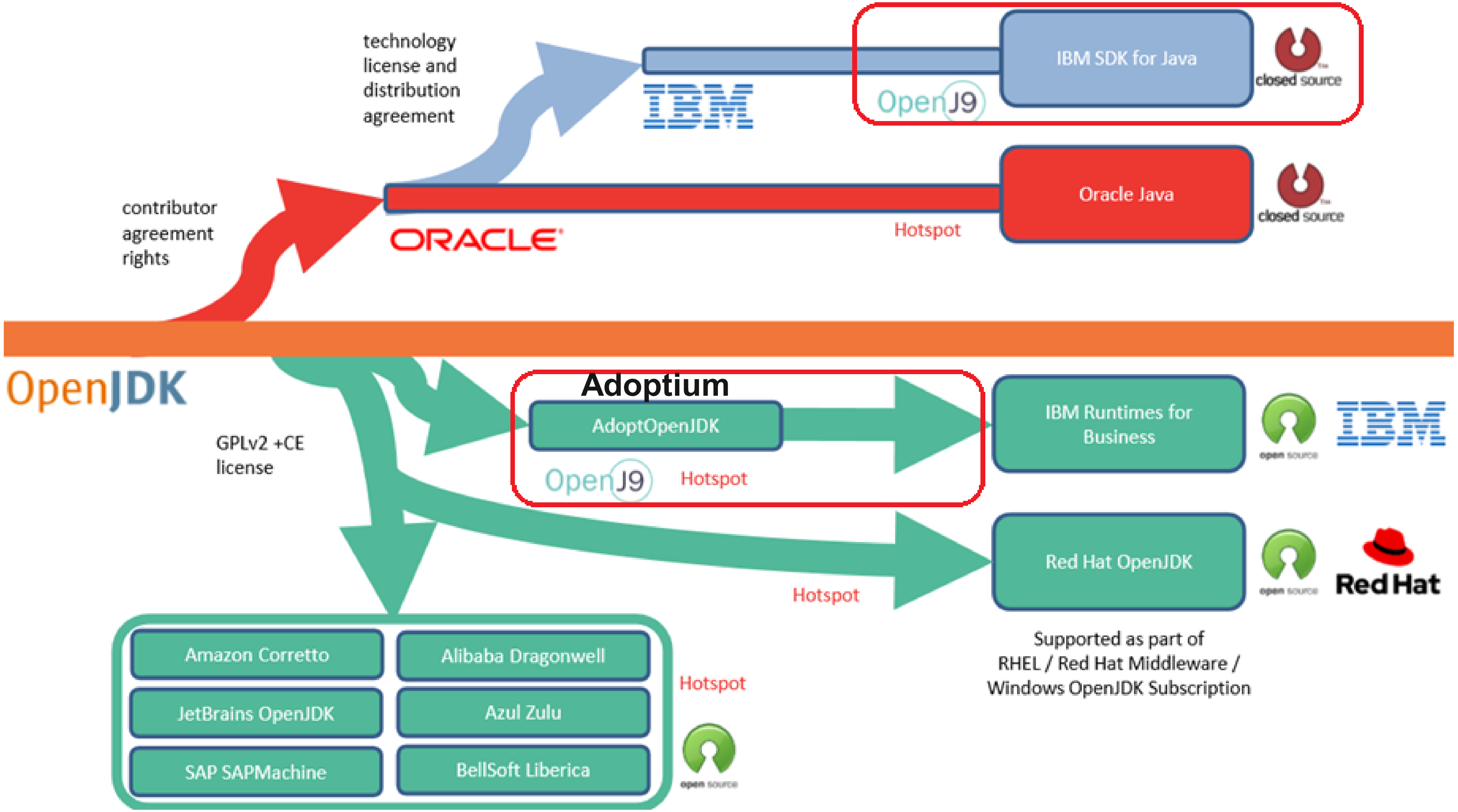
Platforms: X86_64/Aarch64/PPC64/Z

Key components of OMR

- GC framework for managed heaps
- Components for building compiler technology (JIT)
- Cross-platform threading library & platform porting library
- APIs to manage per-interpreter and per-thread contexts

- OpenJDK (https://github.com/ibmruntimes/openj9-openjdk-jdk11): Building framework/Java Class libraries)
- OpenJ9 (https://github.com/eclipse/openj9): Java Virtual Machine Core (equivalent of Hotspot)
- OMR (https://github.com/eclipse/omr): Split from IBM J9 & refactored for polyglot runtimes

Packaging of JDKs

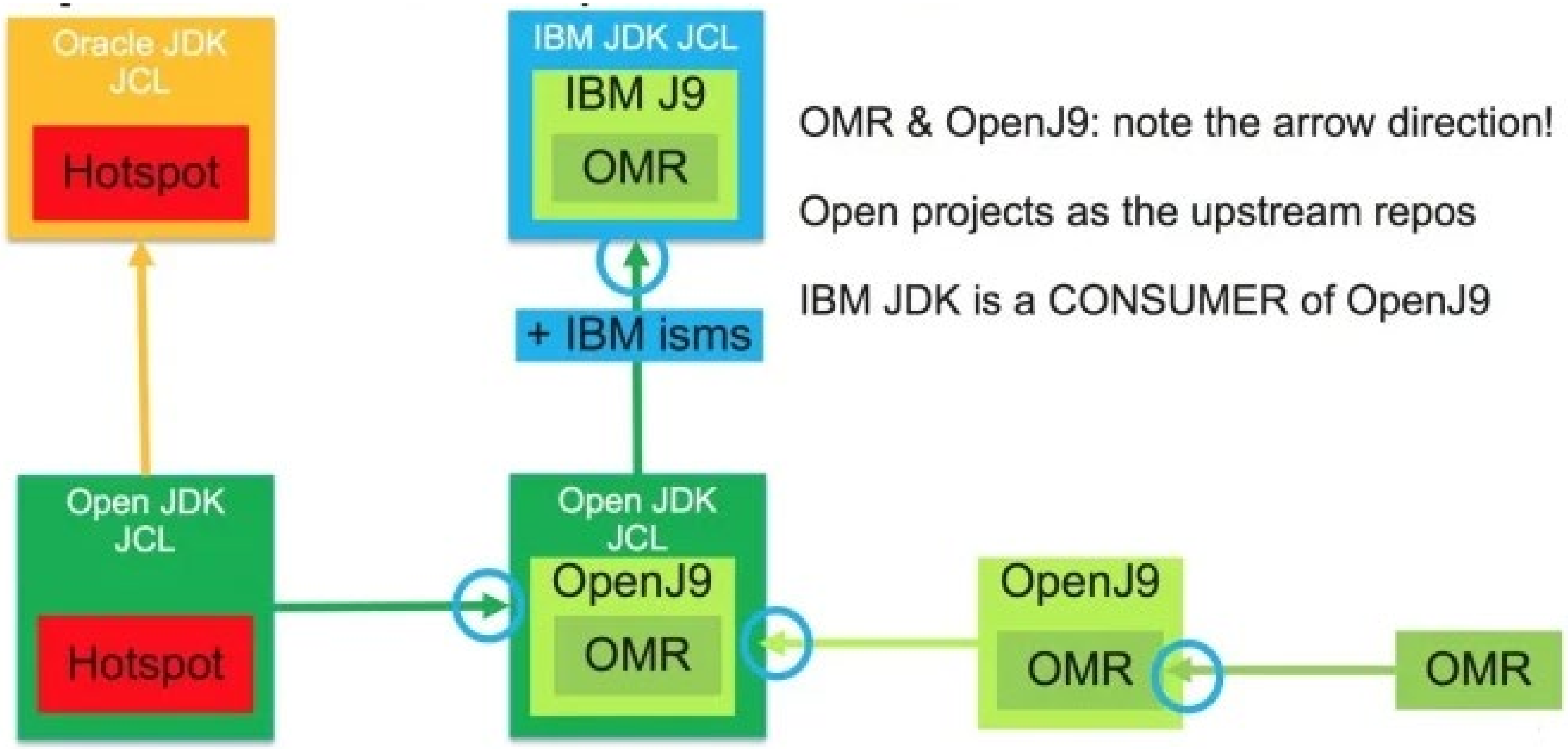


Closed-Source JDK

Open Source JDK

http://ibmhybridcloud.lookbookhq.com/c/ps07-java-strategy-a?x=5so0jp: Java Strategy And Roadmap

Packaging of JDKs (Cont.)



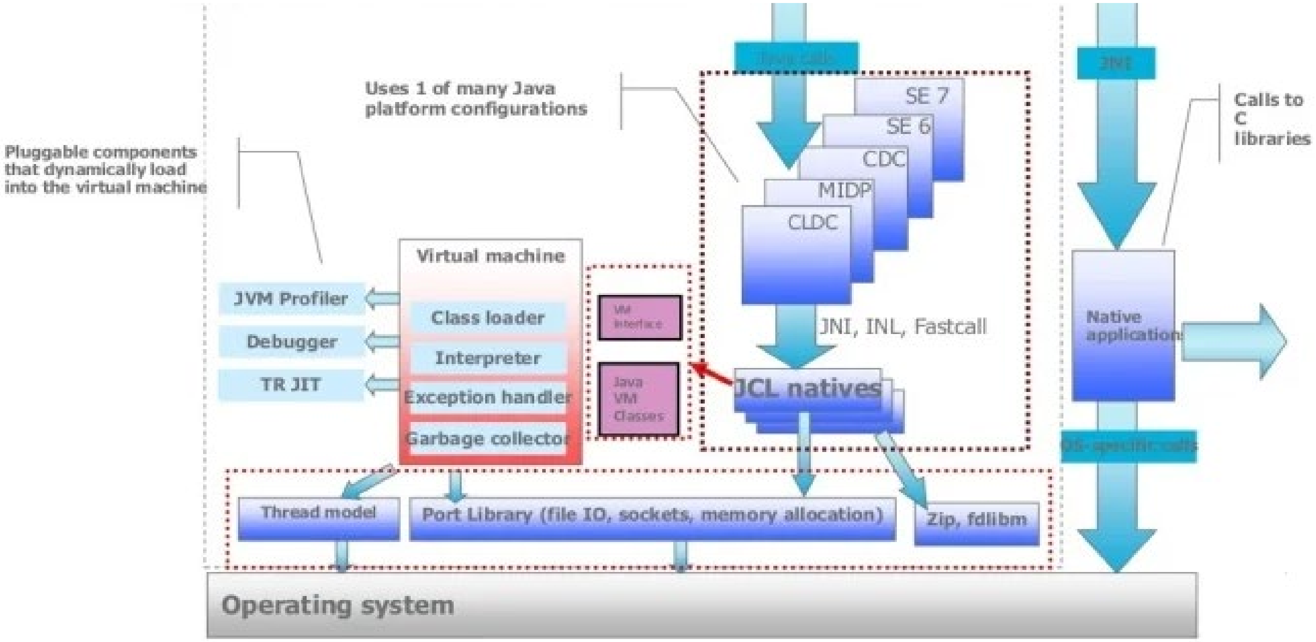
OMR & OpenJ9: note the arrow direction!

Open projects as the upstream repos

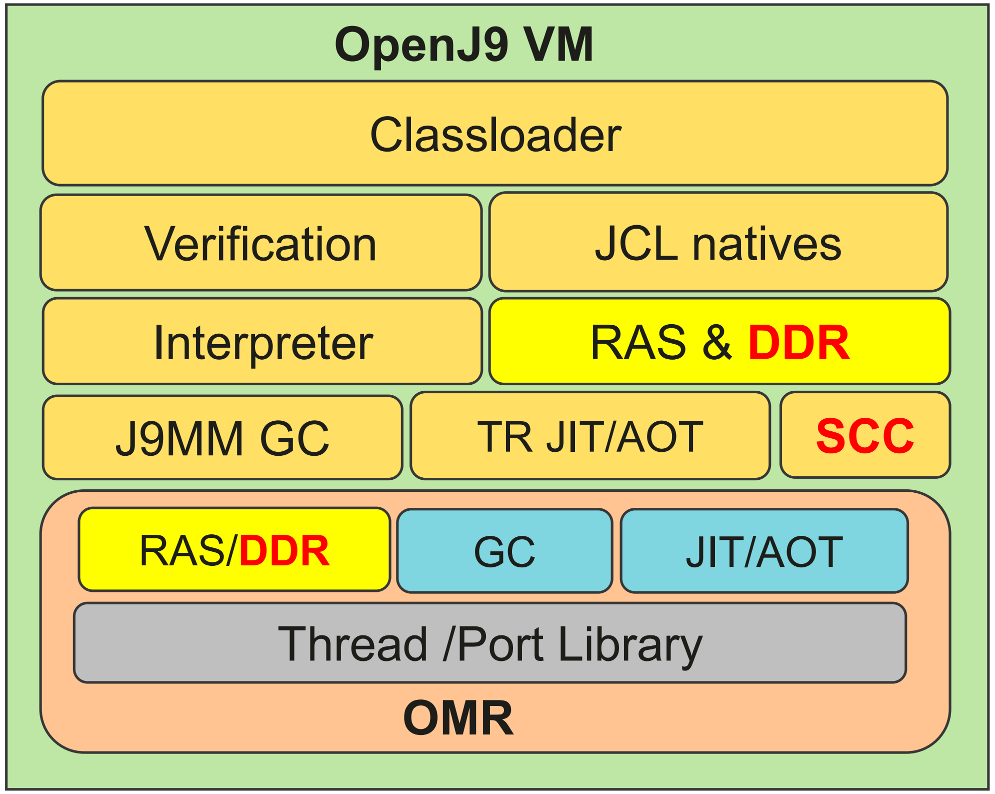
IBM JDK is a CONSUMER of OpenJ9

https://www.infoq.cn/article/2017/09/ibm-jvm-openj9-eclipse

OpenJ9 Architecture

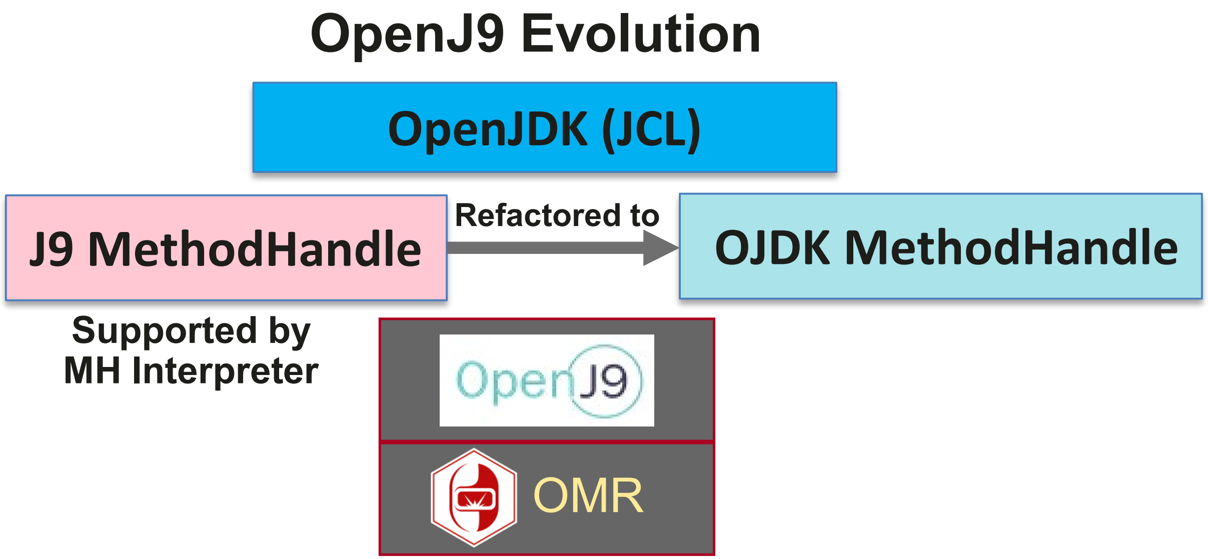


https://www.infoq.cn/article/2017/09/ibm-jvm-openj9-eclipse



- **Direct Dump Reader (DDR):** diagnose issues of OpenJ9 in Java stacktraces at the bytecode level
  https://www.slideshare.net/Dev_Events/secrets-of-building-a-debuggable-runtime-learn-how-language-implementors-solve-your-runtime-issues
- **Shared Classes Cache (SCC / equivalent of CDS in Hotspot):** store ROM Classes & AOT code for better performance (startup time, footprint, etc)
  https://developer.ibm.com/technologies/java/tutorials/j-class-sharing-openj9/

J9 Architecture (Cont.)



OpenJ9 Evolution

OpenJDK (JCL)

J9 MethodHandle

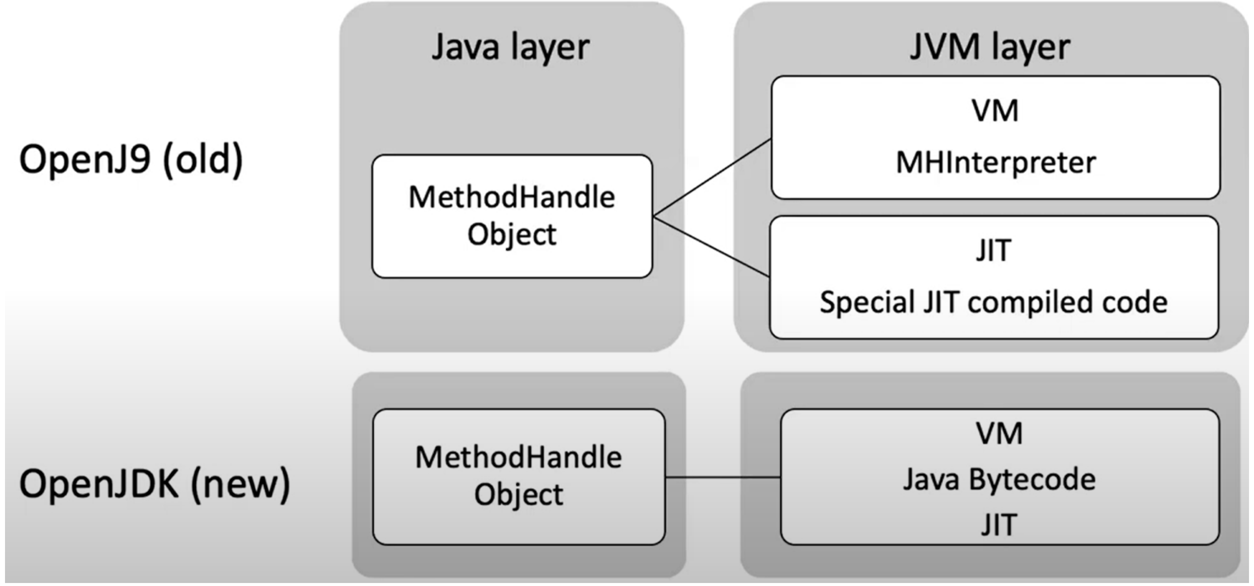
Refactored to

OJDK MethodHandle

Supported by
MH Interpreter

OpenJ9

OMR



https://www.youtube.com/watch?v=kEzBsFoV9PQ&t=88s:
Supporting OpenJDK MethodHandles in OpenJ9

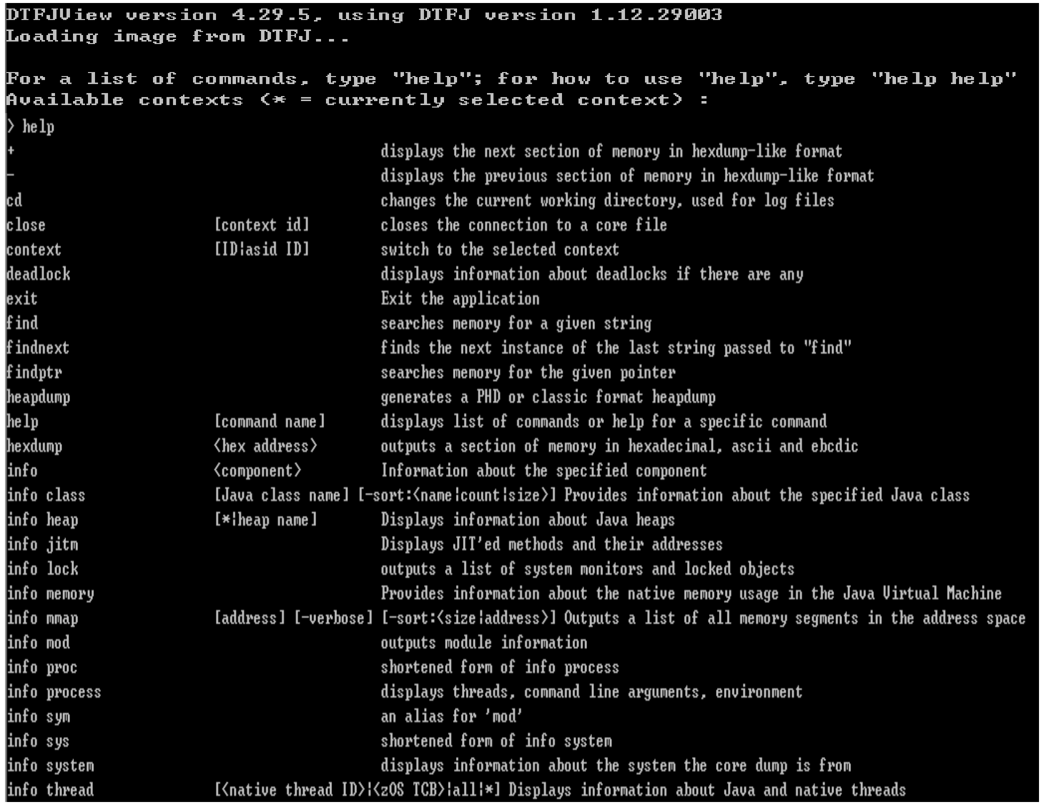
Why Adopt OpenJDK MH in OpenJ9?

- OpenJDK MH (JSR292) serves as a foundation layer for method invocation (Reflection, JNI in Project Panama, etc)
- share a common representation between VM and JIT

Java Diagnostic Infrastructure

- Diagnostic Tool Framework for Java (DTFJ)
  - Java APIs that is used to support the building of Java diagnostic tools which works with data from a system/Java dump
- DDR (Direct Dump Reader)
  - a Java implementation of the DTFJ APIs which works by walking the J9 structures inside a dump to extract the VM and application state
- Jdmpview
  - a command-line tool that allows you to examine the contents of system dumps produced from OpenJ9 (both Java/native information from the time the dump was produced)
  - DDR commands start with **“!”**

Jdmpview command list



```
DTFJView version 4.29.5, using DTFJ version 1.12.29003
Loading image from DTFJ...

For a list of commands, type "help"; for how to use "help", type "help help"
Available contexts (* = currently selected context) :
> help
+                                              displays the next section of memory in hexdump-like format
-                                              displays the previous section of memory in hexdump-like format
cd                                             changes the current working directory, used for log files
close                     [context id]         closes the connection to a core file
context                   [ID|asid ID]         switch to the selected context
deadlock                                       displays information about deadlocks if there are any
exit                                           Exit the application
find                                           searches memory for a given string
findnext                                       finds the next instance of the last string passed to "find"
findptr                                        searches memory for the given pointer
heapdump                                       generates a PHD or classic format heapdump
help                      [command name]       displays list of commands or help for a specific command
hexdump                   <hex address>        outputs a section of memory in hexadecimal, ascii and ebcdic
info                      <component>          Information about the specified component
info class                [Java class name] [-sort:<name|count|size>] Provides information about the specified Java class
info heap                 [*|heap name]        Displays information about Java heaps
info jitm                                      Displays JIT'ed methods and their addresses
info lock                                      outputs a list of system monitors and locked objects
info memory                                    Provides information about the native memory usage in the Java Virtual Machine
info mmap                 [address] [-verbose] [-sort:<size|address>] Outputs a list of all memory segments in the address space
info mod                                       outputs module information
info proc                                      shortened form of info process
info process                                   displays threads, command line arguments, environment
info sym                                       an alias for 'mod'
info sys                                       shortened form of info system
info system                                    displays information about the system the core dump is from
info thread               [<native thread ID>|<zOS TCB>|all|*] Displays information about Java and native threads
```

https://www.ibm.com/docs/en/sdk-java-technology/8?topic=interfaces-dtfj

https://eclipse.dev/openj9/docs/tool_jdmpview/

DDR/Jdmpview (1)

```
Jdk/bin/jdmpview -core  core.20201123.073314.15487.0001_140323-073326.dmp

> !threads
      !stack 0x020be500    !j9vmthread 0x020be500  !j9thread 0x7fb0740074e0     tid 0x25ac (9644) // (main)
      !stack 0x020c7f00    !j9vmthread 0x020c7f00  !j9thread 0x7fb0740b9690     tid 0x25ae (9646) // (JIT Compilation Thread-0)
      !stack 0x020cd000    !j9vmthread 0x020cd000  !j9thread 0x7fb0740b9c08     tid 0x25af (9647) // (JIT Compilation Thread-1 Suspended)
      !stack 0x020d2000    !j9vmthread 0x020d2000  !j9thread 0x7fb0740baba0     tid 0x25b0 (9648) // (JIT Compilation Thread-2 Suspended)
      !stack 0x020d7100    !j9vmthread 0x020d7100  !j9thread 0x7fb0740bb118     tid 0x25b1 (9649) // (JIT Compilation Thread-3 Suspended)
...
      !stack 0x020e1200    !j9vmthread 0x020e1200  !j9thread 0x7fb0740bc638     tid 0x25b3 (9651) // (JIT-SamplerThread)
      !stack 0x020e6300    !j9vmthread 0x020e6300  !j9thread 0x7fb0740f5220     tid 0x25b4 (9652) // (IProfiler)
      !stack 0x021aac00    !j9vmthread 0x021aac00  !j9thread 0x7fb0740f5798     tid 0x25b5 (9653) // (Signal Dispatcher)
      !stack 0x021afc00    !j9vmthread 0x021afc00  !j9thread 0x7fb074386a48     tid 0x25b7 (9655) // (GC Slave)
      !stack 0x021b9d00    !j9vmthread 0x021b9d00  !j9thread 0x7fb0743aa278     tid 0x25b9 (9657) // (Attach API wait loop)
      !stack 0x021bcd00    !j9vmthread 0x021bcd00  !j9thread 0x7fb0743a9d00     tid 0x25ba (9658) // (Reflecting Thread) <--------
. . .
```

DDR/Jdmpview (2)

```
> !stackslots  0x021bcd00
<21bcd00> *** BEGIN STACK WALK, flags = 00400001 walkThread = 0x00000000021BCD00 ***
<21bcd00>       ITERATE_O_SLOTS
<21bcd00>       RECORD_BYTECODE_PC_OFFSET
<21bcd00> Initial values: walkSP = 0x00000000021D9A80, PC = 0x00007FB0742DD911, literals = 0x0000000002145368, A0 =
0x00000000021D9AD0, j2iFrame = 0x00000000021D9B58, ELS = 0x00007FB061AF6B28, decomp = 0x00007FB0743ECE60
<21bcd00> Bytecode frame: bp = 0x00000000021D9AB0, sp = 0x00000000021D9A80, pc = 0x00007FB0742DD911, cp =
0x00000000021462B0, arg0EA = 0x00000000021D9AD0, flags = 0x0000000000000000
<21bcd00>       Method: com/ibm/jit/JITHelpers.getIntFromObject(Ljava/lang/Object;J)I  !j9method 0x0000000002145368 <-----
<21bcd00>       Bytecode index = 5 <-------
<21bcd00>       Using debug local mapper
<21bcd00>       Locals starting at 0x00000000021D9AD0 for 0x0000000000000004 slots
<21bcd00>               O-Slot: a0[0x00000000021D9AD0] = 0x00000000E000B240 <--------------------
<21bcd00>               O-Slot: a1[0x00000000021D9AC8] = 0x0000000000000000
<21bcd00>               I-Slot: a2[0x00000000021D9AC0] = 0x0000000000000000
<21bcd00>               I-Slot: a3[0x00000000021D9AB8] = 0x0000000000000038
…
```

DDR/Jdmpview (3)

```
> j9object  0x00000000E000B240
!J9Object 0x00000000E000B240 {
      struct J9Class* clazz = !j9class 0x2145F00   // com/ibm/jit/JITHelpers <-----
      Object flags = 0x00000000;
      I lockword = 0x00000000 (offset=0) (java/lang/Object) <hidden>


>!j9class 0x0000000002145F00
J9Class at 0x2145f00 {
  Fields for J9Class:
        0x0: UDATA eyecatcher = 0x0000000099669966 (2573637990)
        0x8: struct J9ROMClass * romClass = !j9romclass 0x00007FB0742DCFB8
        0x10: struct J9Class ** superclasses = !j9x 0x0000000002146870
        0x18: UDATA classDepthAndFlags = 0x00000000000E0001 (917505)
        0x20: U32 classDepthWithFlags = 0x00000000 (0)
        0x24: U32 classFlags = 0x00000000 (0)
        0x28: struct J9ClassLoader * classLoader = !j9classloader 0x00007FB074055AD8
        0x30: j9object_t classObject = !j9object 0x00000000E000E858 // java/lang/Class
        0x38: volatile UDATA initializeStatus = 0x0000000000000001 (1)
        0x40: struct J9Method * ramMethods = !j9method 0x0000000002145268 //
com/ibm/jit/JITHelpers.<init>()V
```

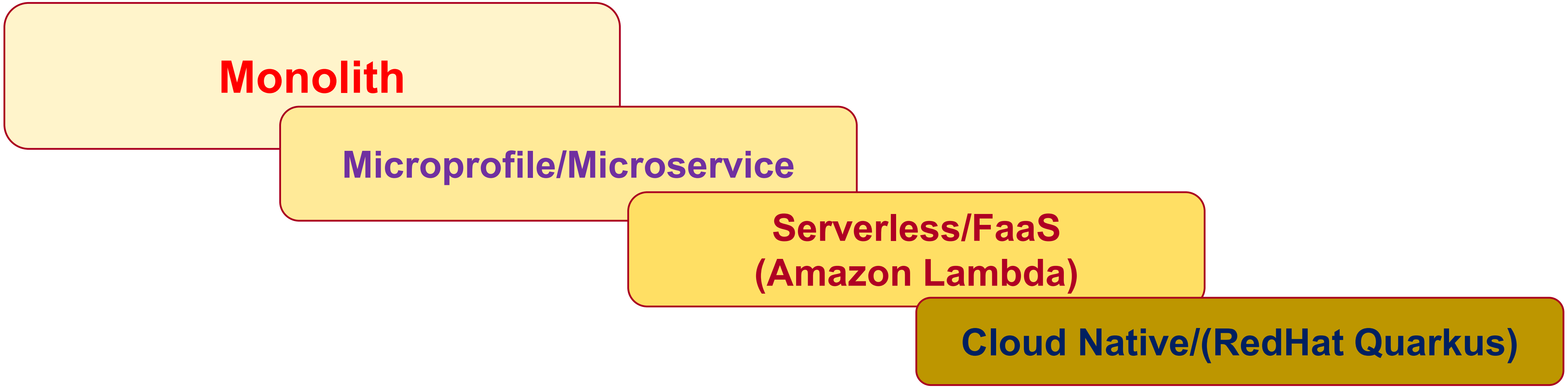
DDR/Jdmpview (4)

```
> !j9method 0x0000000002145368
J9Method at 0x2145368 {
  Fields for J9Method:
        0x0: U8 * bytecodes = !j9x 0x00007FB0742DD90C
        0x8: struct J9ConstantPool * constantPool = !j9constantpool 0x00000000021462B0
        0x10: void * methodRunAddress = !j9x 0x0000000000000005
        0x18: void * extra = !j9x 0x0000000000000001
}
Signature: com/ibm/jit/JITHelpers.getIntFromObject(Ljava/lang/Object;J)I !bytecodes 0x0000000002145368 <------

> !bytecodes 0x0000000002145368
  Name: getIntFromObject
  Signature: (Ljava/lang/Object;J)I
  Access Flags (50001): public
  Max Stack: 4
  Argument Count: 4
  Temp Count: 0

    0 getstatic 9 com/ibm/jit/JITHelpers.unsafe Lsun/misc/Unsafe;
    3 aload1
    4 lload2
    5 invokevirtual 24 sun/misc/Unsafe.getInt(Ljava/lang/Object;J)I <-------- Bytecode index = 5 indicated in stackslots
    8 return1
```

The Challenges of Java Ecosystem

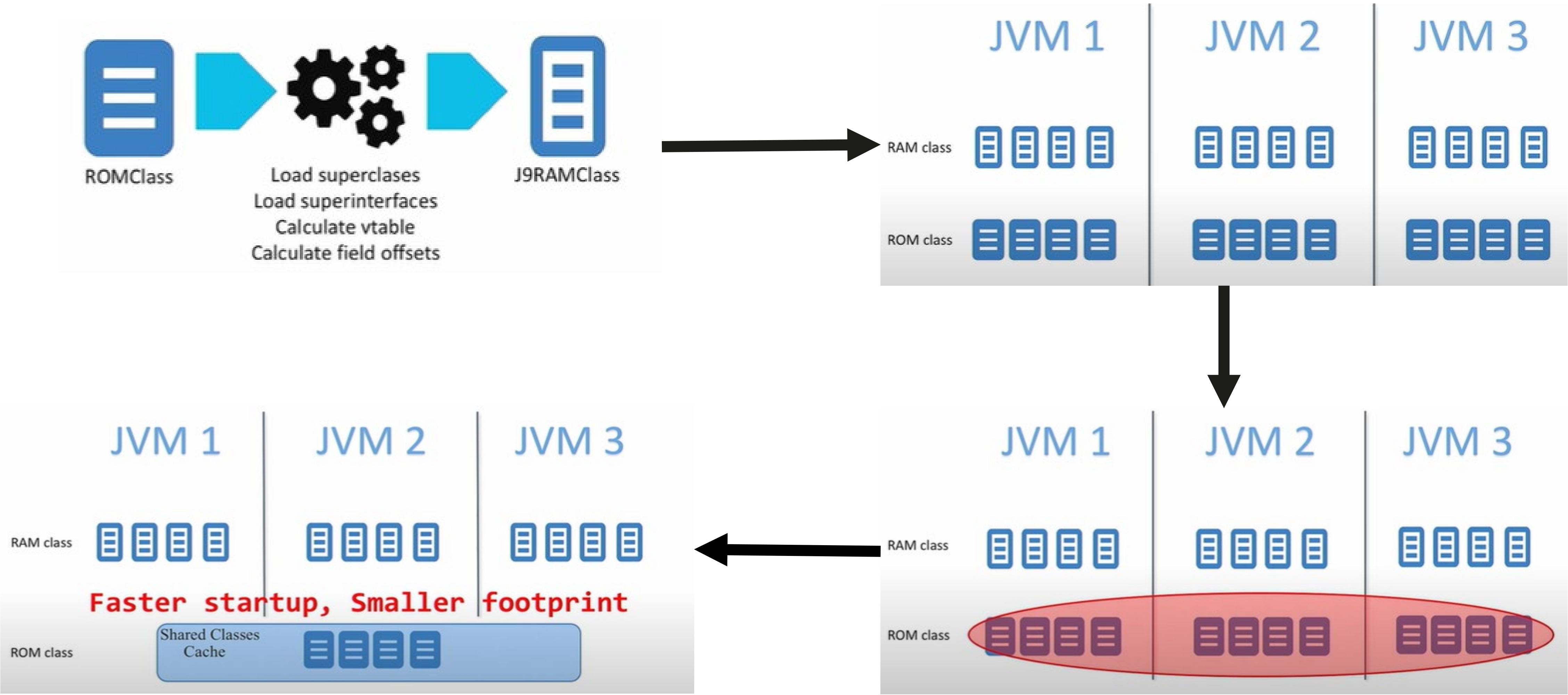


Serverless and container-based applications deploy the fastest
Servers
Deploy in months
Virtual Machines
Deploy in minutes
Containers
Deploy in seconds
Serverless
Deploys in milliseconds

Traditional/Legacy Applications → Cloud/AI

https://www.cloudflare.com/learning/serverless/serverless-vs-containers/

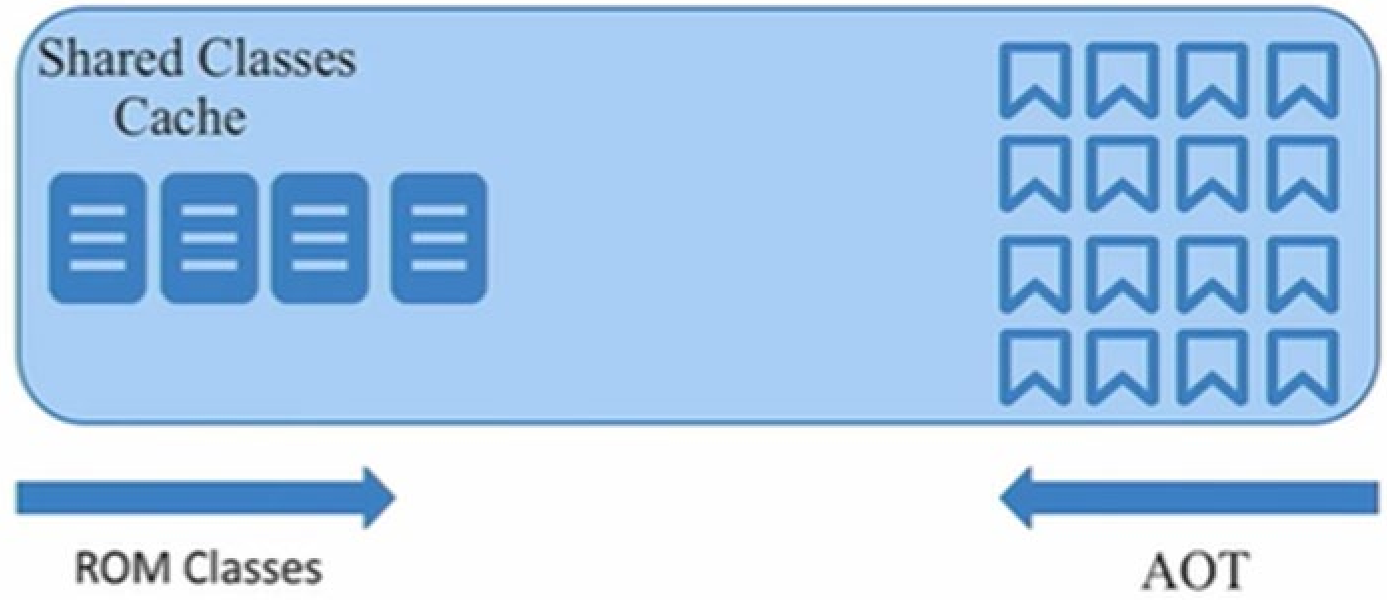
Shared Classes Cache (SCC)



https://www.youtube.com/watch?v=BUAESSI2sy8: The Eclipse OpenJ9 JVM a deep dive!

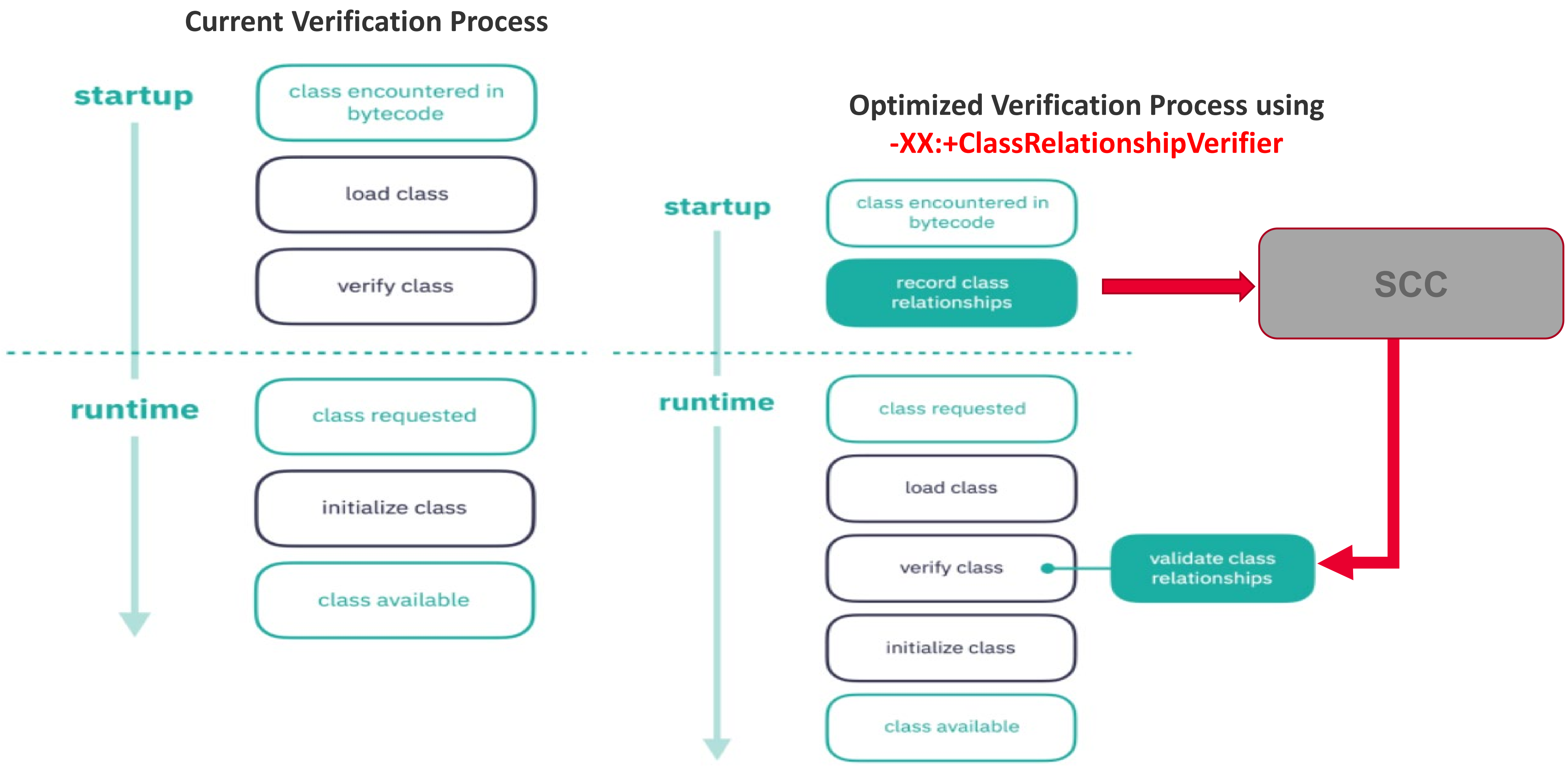
Shared Classes Cache (Cont.)

- SCC (since 2005) serves as a data container for everything related to performance & debugging
- It holds both class metadata, compiler specific data (AOT), and others.
- Data placement for VM & Compiler starts from both sides



https://www.youtube.com/watch?v=BUAESSl2sy8: The Eclipse OpenJ9 JVM a deep dive!

Lazy Verification via SCC

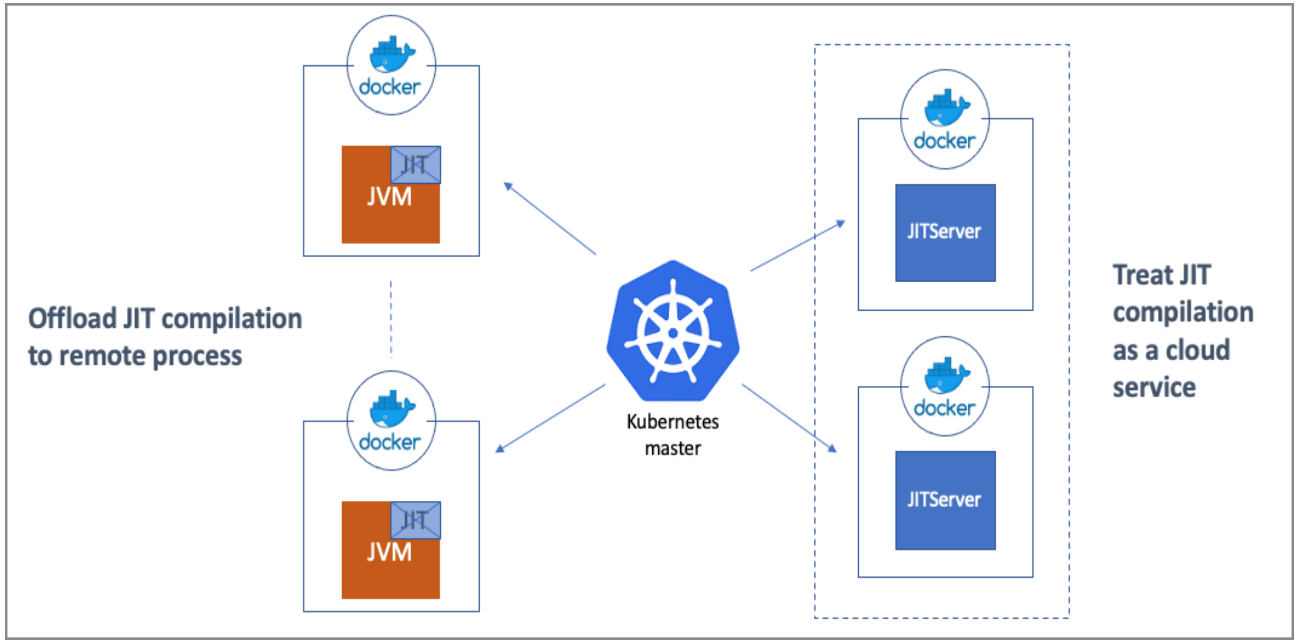


Current Verification Process

Optimized Verification Process using
-XX:+ClassRelationshipVerifier

https://blog.openj9.org/2019/10/29/relationship-verification-lets-get-lazy/: Relationship verification

JIT-as-a-Service (JITaaS) with SCC


docker
JIT
JVM
Offload JIT compilation to remote process
docker
JIT
JVM
Kubernetes master
docker
JITServer
docker
JITServer
Treat JIT compilation as a cloud service

https://developer.ibm.com/articles/jitserver-optimize-your-java-cloud-native-applications/

Cons of JIT compiler

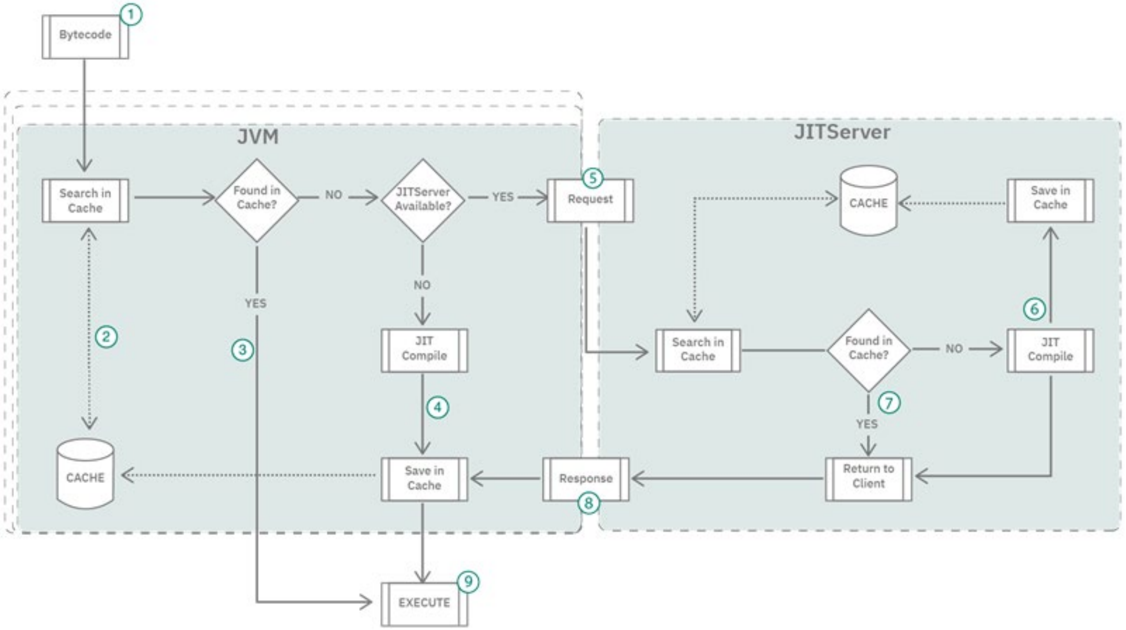
- consumes more CPU cycles and memory
- slows application startup
- can create memory spikes and out-of-memory (OOM) crashes
- can create CPU consumption spikes degrading the QoS

Pros of JITServer

- decouples the JIT compiler from the VM
- lets the JIT compiler run remotely in its own process
- mitigates CPU and memory consumption triggered by JIT
- not affected by the stability of the compilation

Note:

JVM retains the ability to compile locally caused by unavailability of the JITServer due to a crash/network issues



Q & A

Thank you.

Bring digital to every person, home and organization for a fully connected, intelligent world.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.

HUAWEI