Sustaining the zero assembler port in OpenJDK: An inside perspective of CPU specific issues

Severin Gehwolf
Senior Software Engineer, Red Hat
Roman Kennke,
Principal Software Engineer, Red Hat
February 1, 2015

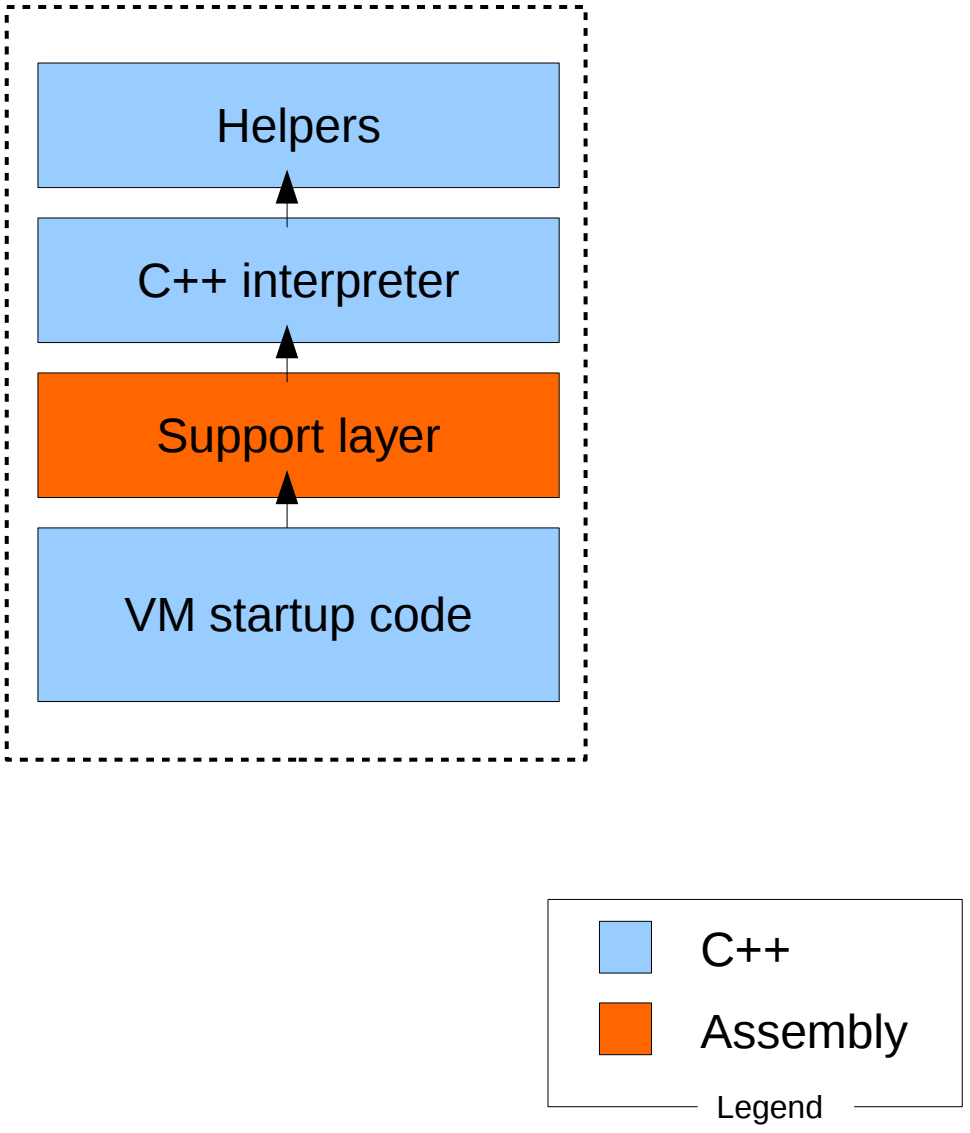
Slides available at: http://bit.ly/Zero-FOSDEM-2015

Agenda

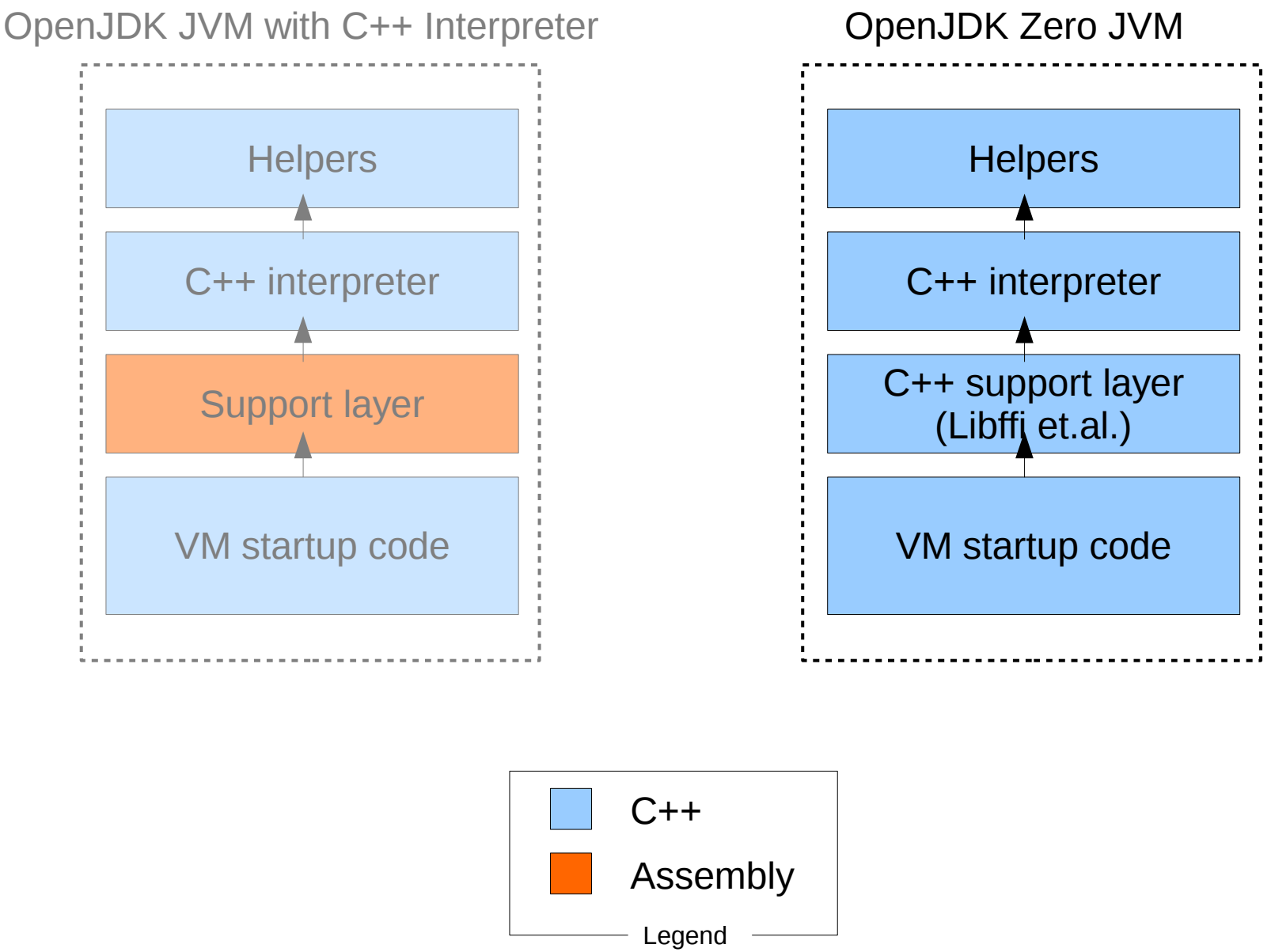
- What is Zero? How does it work? Why is it useful?
- Examples: Selection of issues found recently
- Lessons learned
- Shark status
- Questions?

What is Zero? How does it work?

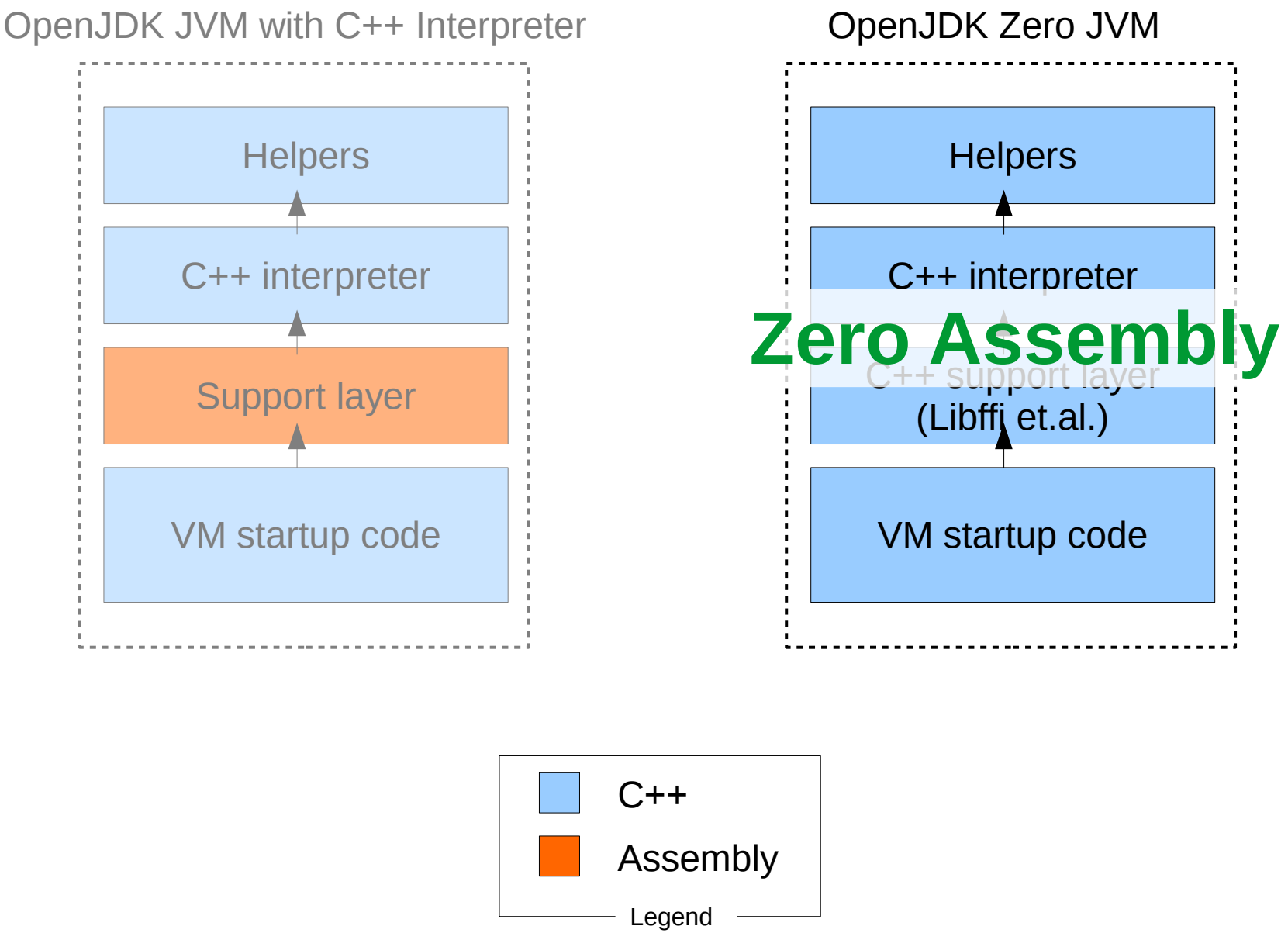
OpenJDK JVM with C++ Interpreter



What is Zero? How does it work?



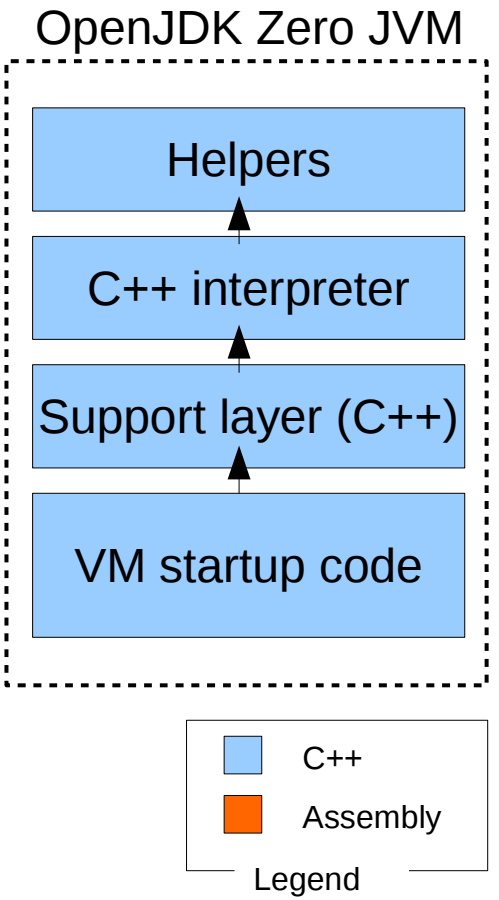
What is Zero? How does it work?



What is Zero? How does it work?

- Zero assembler OpenJDK port (configure option **`--with-jvm-variants=zero`**)
- Uses the C++ template interpreter
- Uses libffi for JNI calls
- Interpreter only JVM (no JIT)

```
./java -version 2>&1 | grep Zero
OpenJDK 64-Bit Zero VM (…, interpreted mode)
```



Why is it useful?

- Zero assembler OpenJDK port (configure option `--with-jvm-variants=zero`)
- Uses the C++ template interpreter
- Uses libffi for JNI calls
- Interpreter only (no JIT)
- Builds on any Linux system with GCC
  - Useful for Java builders (non-mainstream arches)
  - Useful for bootstrapping JIT ports using free software
- Many Linux distributions build Zero JVMs
  - Fedora (s390[1],ARM[2]), Debian (mips[3], PPC[4]), etc.

Example 1)
JDK-8064815: Stack overflow when running Maven

- Surfaced on PPC/PPC64, worked on x86_64
- Zero's min_stack_allowed set to 64K
- Page size on PPC Linux is 64K (vs 4K x86_64) =>
  Only minimal pages got allocated for java thread: 3 pages: 1 red, 1 yellow, 1 shadow.
- Zero calculated remaining stack size wrong
- Running maven with no-arguments was enough to trigger problem.
- Original patch by Andrew Haley

Example 2)
GCC bug, PR63341: Vectorization miscompilation with -mcpu=power7

- Discovered when self-builds of Zero JVMs would randomly fail.
- Turned out, that hotspot has GCC optimization level -O2 on (-O3, in fact).
- Due to wrongly generated code, Zero JVM would segfault for self-rebuilds.
- Discovered by Andrew Haley

Example 3)
JDK-8067330: ZERO_ARCHDEF incorrectly defined for PPC/PPC64 architectures

- Zero defined ZERO_ARCHDEF=PPC
- Results in -DPPC being passed to hotspot build
- src/share/vm/utilities/macros.hpp only expects PPC32 and PPC64 and silently undefines PPC
- Possibly similar issues on other architectures
- Discovered by Andrew Haley

Example 4)

JDK-8046938: Using mixed types in MIN2/MAX2 functions causes build problems on s390 (32-bit)

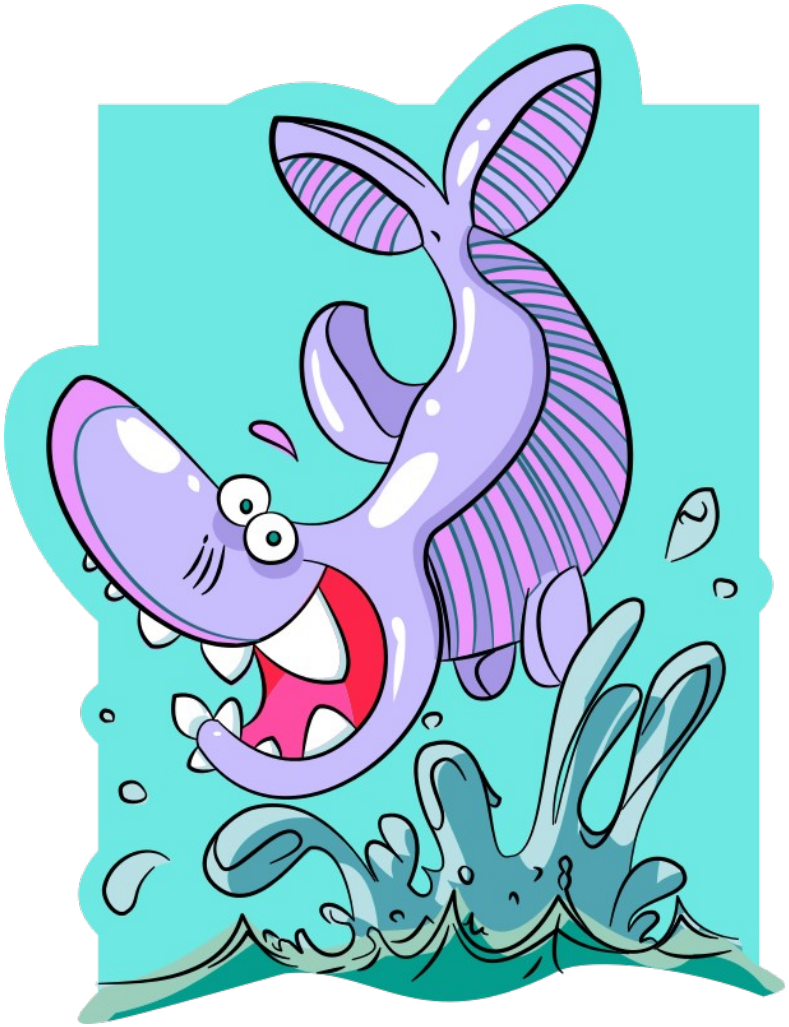
- MIN2/MAX2 implemented using C++ templates
- Hotspot code calls MIN2 and MAX2 with mixed types
  e.g. type size_t for one argument and uintx
- size_t defined as unsigned long on s390
- uintx defined as unsigned int (by hotspot)
- Discovered by Dan Horák

Lessons Learned

- Zero is still useful (> 5 years after it was written)
- Zero can help find architecture specific issues in OpenJDK's codebase.
- Zero variant JVM not tested by Oracle[5]

  => Upstream changes may break Zero/Shark
- External CI jobs for building Zero JVMs needed[6]
  - Possibly run them on multiple architectures
  - Good tests: Self-builds, SPECjvm2008, etc.
- **We need your help!**
  - Propose your local Zero build/runtime fixes upstream (if any)
  - Host CI jobs for building Zero on your architecture

Shark Status Update



(source: openclipart.org)

Shark

- The JIT for Zero
- Uses LLVM JIT for compiling to target
- 100% C++, no target specific code involved
- LLVM versions 3.2 to 3.4 known to work

Shark

- Uses Zero interpreter for initial execution and profiling
- Parses bytecodes and generates LLVM IR
- Passes IR to LLVM JIT to compile machine code

Shark

- No JSR292/invokedynamic support yet
  - Falls back to Zero interpretation -> **very slow!**
- Clunky GC support
  - We don't know which registers/stack slots IR values get allocated to
  - Write live values to shadow/zero stack before potential safepoints, read back after -> **very slow!**
- JIT -> MCJIT
  - MCJIT compiles modules not single methods

Shark

- Many unimplemented intrinsics, some fairly imporant: e.g. Object.<init>
- Link LLVM dynamically
- LLVM only supports JIT on x86 and ppc and maybe arm -> Hotspot has JITs for those
- How to deal with LLVM versions?
- LLVM 3.5? C++11?

Questions?

References

[1] http://s390.koji.fedoraproject.org/koji/buildinfo?buildID=291962

[2] http://koji.fedoraproject.org/koji/buildinfo?buildID=605792

[3] https://packages.debian.org/sid/mips/openjdk-8-jre

[4] https://packages.debian.org/sid/powerpc/openjdk-8-jre

[5] Upstream Zero build breakages in JDK-9: JDK-8055231, JDK-8041992, JDK-8067231

[6] Public OpenJDK Zero variant CI job (x86_64):
http://builder.classpath.org/jenkins/job/OpenJDK9_hscomp_Zero/

[7] Zero architecture overview:
https://today.java.net/pub/a/today/2009/05/21/zero-and-shark-openjdk-port.html

[8] OpenJDK hotspot-dev mailing list (Zero upstream development):
http://mail.openjdk.java.net/mailman/listinfo/hotspot-dev