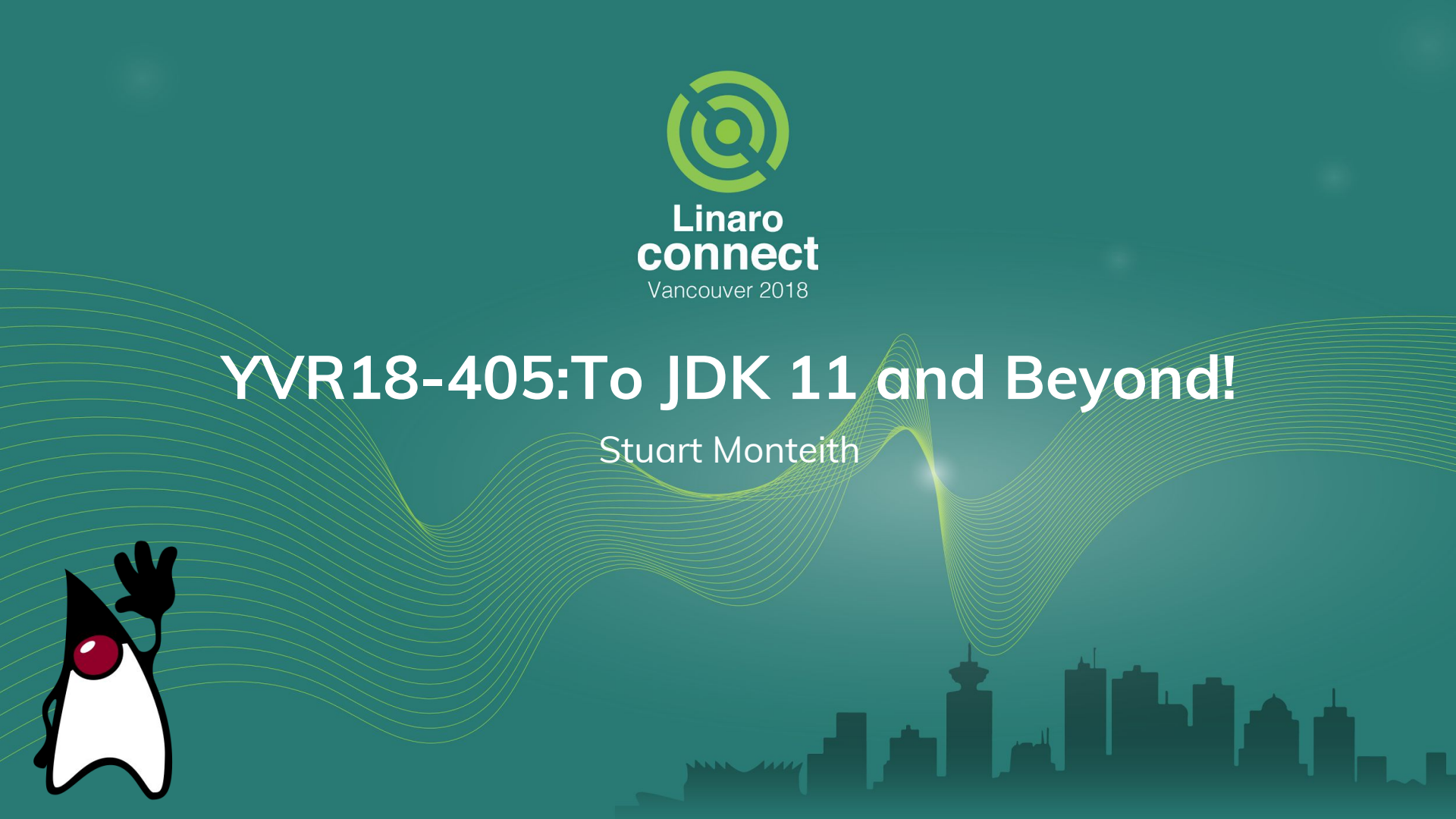

Linaro
connect
Vancouver 2018
YVR18-405:To JDK 11 and Beyond!
Stuart Monteith

Contents

- OpenJDK Releases.
- What has happened since JDK9, and JDK10.
- What to expect from JDK 11.
- JDK 12+
- ZGC.

- Please interject - I'm interested to know what you think is important.

OpenJDK Releases

- Since JDK9 there has been a 6 monthly cycle remains, and this is being adhered to.
  - JDK 9 was released on 21st September 2017, during SFO17.
  - JDK 10 was released in March 2018, during HKG18.
  - JDK 11 will be release on the 28th September 2018.
- OpenJDK 11 will be supported by Oracle for 6 months
  - Otherwise, commercial LTS support from Oracle, others.
  - jdk11u to be supported by the community from March next year. This is currently under discussion in the OpenJDK community.
- OpenJDK 8 being dropped from support from Oracle in January 2019
  - Red Hat might maintain it from then on (like with OpenJDK 7)
- JDK12 - 19th March 2019

AdoptOpenJDK

- Community effort for producing OpenJDK builds.
  - Outside of Oracle.
  - TCK
- See https://adoptopenjdk.net/
  - Builds of OpenJDK releases.
  - JDK 11 will appear once released.
- Sponsored by Azul, IBM, LJC, Microsoft Azure, Ocado, packet.
  - Linaro, Digital Ocean, Joyent, macincloud, MacStadium, Scaleway tier-2 sponsors.

What has happened since JDK9

- Selection of “interesting” JEPs
  - The ones of particular interest to Arm.

Story so far - September 2017

- JDK 9: The last big release
- 91 JEPs including:
  - JEP 237 - Linux AArch64 Port
  - JEP 254 - Compact Strings
  - JEP 280 - Indify String Concatenation
  - JEP 295 - AOT
  - Jigsaw: JEP 261 - Module System
    - JEP 200: The Module JDK
    - JEP 282: jlink
    - JEP 260: Encapsulate Most Internal APIs
  - JEP 193 - VarHandles
    - sun.misc.unsafe
  - JEP 297 - Unified arm32/arm64 Port

Story so far - March 2018

- JDK 10 - First release under new cadence.
  - 12 JEPs
- A few JEPs:
  - JEP 304 - Garbage Collector Interface
  - JEP 322 - Time based Release Versioning
    - 18.3, 18.9, etc?
  - JEP 317 - Experimental Java-based JIT Compiler
    - -XX:+UseJVMCICompiler
    - Works for AArch64
  - JEP 286 - Local Variable type Inference
    - “var”
- Replaced by JDK 11 soon...

Story so far - September 2018

- JDK 11 - 17 JEPs
- Significant JEPs:
  - JEP 318 - Epsilon: A No-Op Garbage Collector
  - JEP 333 - ZGC: A Scalable Low-Latency Garbage Collector (Experimental)
  - JEP 328 - Flight Recorder
  - JEP 315 - Improve AArch64 intrinsics
    - YVR18-502:Java on ARM: Theory, Applications, and Workloads
- Other changes
  - Java Mission Control was open sourced - http://hg.openjdk.java.net/jmc/jmc
  - 8185505: AArch64: Port AOT to AArch64

Story so far?

- JDK 12 in progress.
  - Two JEPs so far..
- Language changes - “Preview”
  - JEP 325 - Switch Expressions (Preview)
  - JEP 326 - Raw String Literals (Preview)
- Optional Language features.
  - “`--enable-preview`” on `javac` **and** `java` tools.
- There are also experimental features in Hotspot
  - `-XX:+UnlockExperimentalVMOptions -XX:+UseZGC`
- `And Java incubator APIs`
  - `jdk.incubator.httpclient` in JDK 10
  - `java.net.http` in JDK 11
  - jdk.incubator.vector in JDK ??
- Rampdown 13th December.

In development

- Vector API
  - See Ningsheng's talk on Tuesday:
    - YVR18-210:Bringing Armv8-a Scalable Vector Extension into OpenJDK
    - Will be an incubator API as part of project Panama.
- Z Garbage Collector
  - WIP on AArch64
- Spectre mitigation
  - JEP 342 - Limit Speculative Execution.
- Further intrinsics from Bellsoft
- ...

ZGC (1)

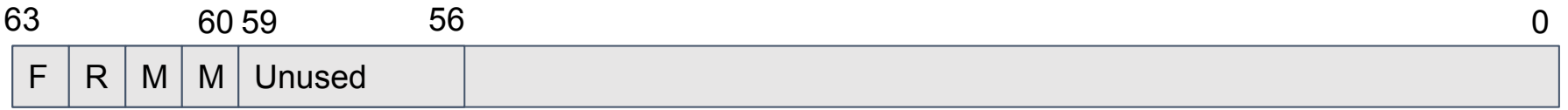
- Experimental feature introduced in JDK11 on x86_64
  - JEP 333: ZGC: A Scalable Low-Latency Garbage Collector (Experimental)
- Concurrent copying garbage collector
  - Low-latency through garbage collector operating at same time as mutator.
  - See Shenandoah, Android ART's concurrent copying collector.
- Goals are :
  - Multi-terabyte heaps.
  - 10ms max-pause time.
  - 15% maximum application throughput reduction.
- We should aim for the same...

ZGC (2)

- Marking, object copying, (Weak|Phantom...)Reference processing occurs concurrently with executing program.
  - Short pauses for roots handling.
- Region based.
  - Much like G1.
- Single generation.
  - For now.
- More will be done concurrently in future.
  - Class unloading.

ZGC - Memory

- Coloured object references.
  - 4 bits in address mean:
    - remapped, finalized, 2xMarked.
- x86_64 uses multimapping - memory mapped 4 TB in 3 places.
- Solaris SPARC port (not on mainline) uses VA Masking.
- In Aarch64 we use 4 bits from TBI (top byte ignore).



GC - Some remarks

- Marking and object remapping occurs concurrently.
  - Objects are moved as the program executes.
- Objects aren't marked - **references are** (hence colouring).
  - Efficiency of read barriers will be key.
  - Enables the concurrent update - references need to be updated atomically, cheaply.
- Use of TBI bits could reduce TLB pressure versus multi-mapping.

ZGC - current status

- Works with the interpreter.
- C2 stubbed-off for now.
  - C1 first...
- Currently debugging C1
  - Simple GC Benchmark runs without error.
  - 48-bit literal oops were one issue - stripped colour bits off
    - I don't anticipate using 52-bit VA.
    - Checking memory barriers
    - Single core execution still faulty.

Thank you!

Questions?

Tomorrow 10am:

YVR18-502:Java on ARM: Theory, Applications, and Workloads

Dmitry Chuyko, BellSoft