ZGC Concurrent Class Unloading

Another safepoint operation bites the dust

developer.oracle.com

Erik Österlund
Garbage Collection Engineer
Java Platform Group, Oracle
February 04, 2019

Live for the Code

ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Agenda

1. What is ZGC? What is class unloading?
2. Overview of phases
3. Concurrent code and metadata unloading
4. Evaluation
5. Future plans

What is ZGC?

New Concurrent GC in **JDK 11**

(Experimental feature, Linux/x86_64 only)

ZGC Goals

TB Multi-terabyte heaps

10ms Max GC pause time

Easy to tune

15% Max application throughput reduction

*) Old Gen strong references Only

What is a concurrent GC

	Serial	Parallel	CMS	G1	ZGC
Marking	-	-	*	*	
Relocation/Compaction	-	-	-	-	
Reference Processing	-	-	-	-	
Relocation Set Selection	-	-	-	-	
StringTable Cleaning	-	-	-	-	
JNI WeakRef Cleaning	-	-	-	-	
JNI GlobalRefs Scanning	-	-	-	-	
Class Unloading	**-**	**-**	**-**	**-**	
Thread Stack Scanning	-	-	-	-	-

Concurrent Class Unloading Released in **JDK 12** for **ZGC**

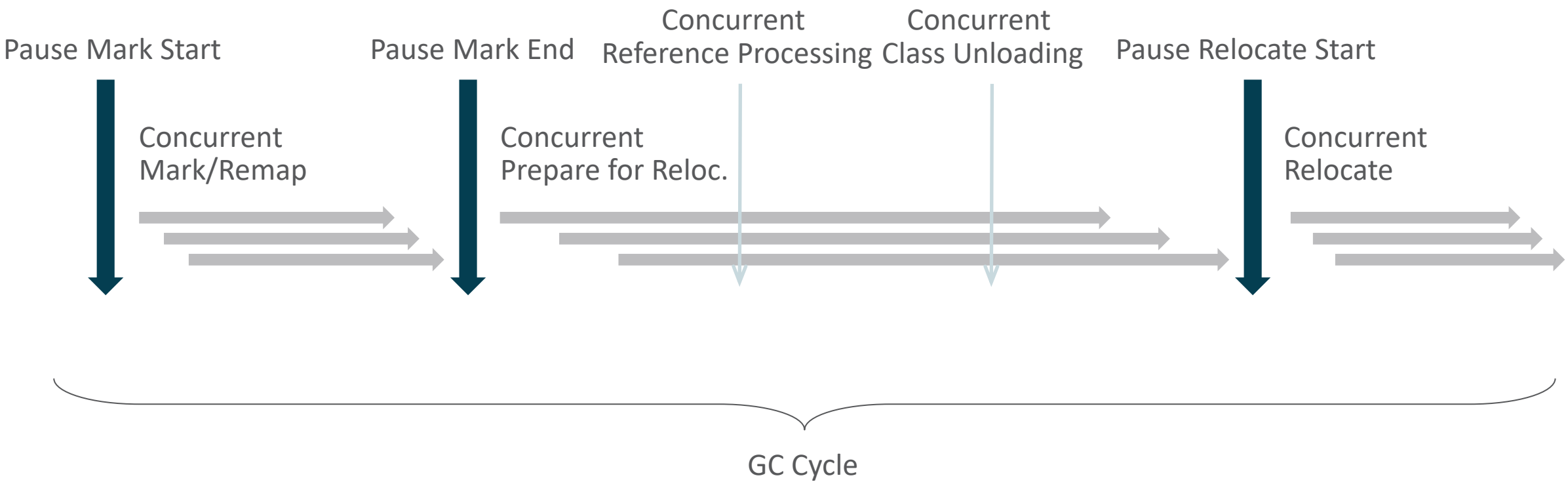
Traditional Class Unloading

- Step 1: Marking (**concurrent**)
  - Mark metadata (classes, CLDs) when marking objects
- Step 2: Reference processing (**STW**)
  - Need to know what is reachable from finalizers before class unloading
- Step 3: Unloading (**STW**)
  - Unload code cache
  - Unload metadata

ZGC Concurrnet Class Unloading

- Step 1: Marking (**concurrent**)
  - Mark metadata (classes, CLDs) when marking objects
  - Mark both strong and final reachable graphs
- Step 2: Reference processing (**concurrent**)
  - Already know what is reachable from finalizers before class unloading
- Step 3: Unloading (**concurrent**)
  - Unload code cache
  - Unload metadata

ZGC Phases

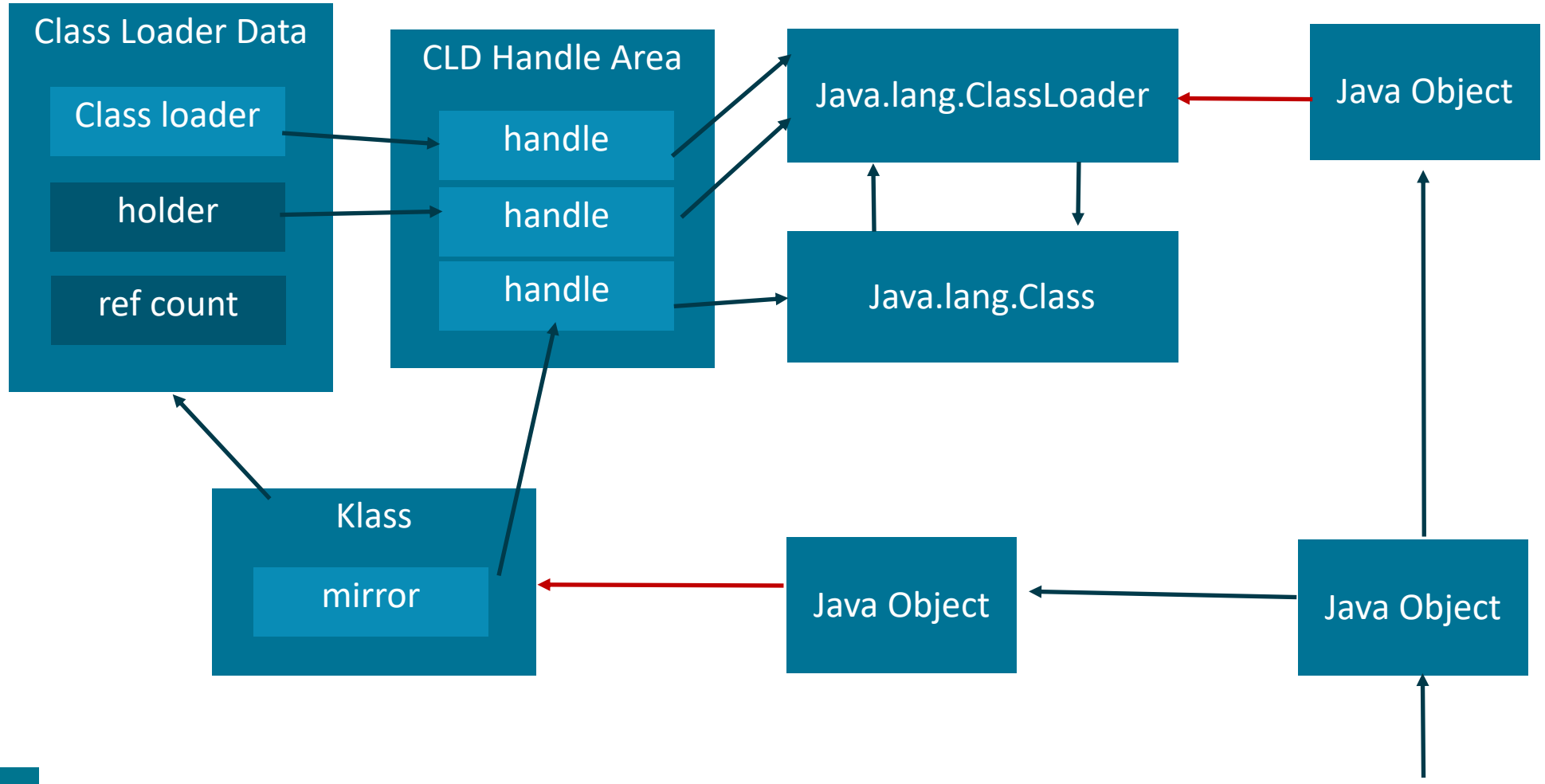


Step 1: Marking

Marking overview

- Color heap object pointers with appropriate marked color
  - Special bit pattern for edges reachable from finalizers only
  - Mutator load barriers upgrade them to strongly reachable when loaded
- Mark metadata objects similarly
  - Mark metadata reachable from objects, with strong/final reachability

Metadata graph



Step 2: Reference Processing

Reference processing overview

- WeakReferences cleared if referent not strongly reachable
- PhantomReferences cleared if referent not reachable
- "Weak" VM datastructures have "phantom" strength
  - Classes die if not reachable (including from finalizers)
- Each access on weak/phantom is annotated using my Access API
- A class is dead if a phantom load of its holder returns NULL

Step 3: Unloading

Stale Datastructures after Reference Processing

Subklass/sibling/implementor lists

Method data objects

Instance class dependency context

jli.CallSite dependency context

Class loader data graph

Protection domain cache table

Module table

String table

Package table

Symbol table

Resolved method table

Loader constraint table

Resolution error table

Metaspace

Code cache (JIT compiled code)

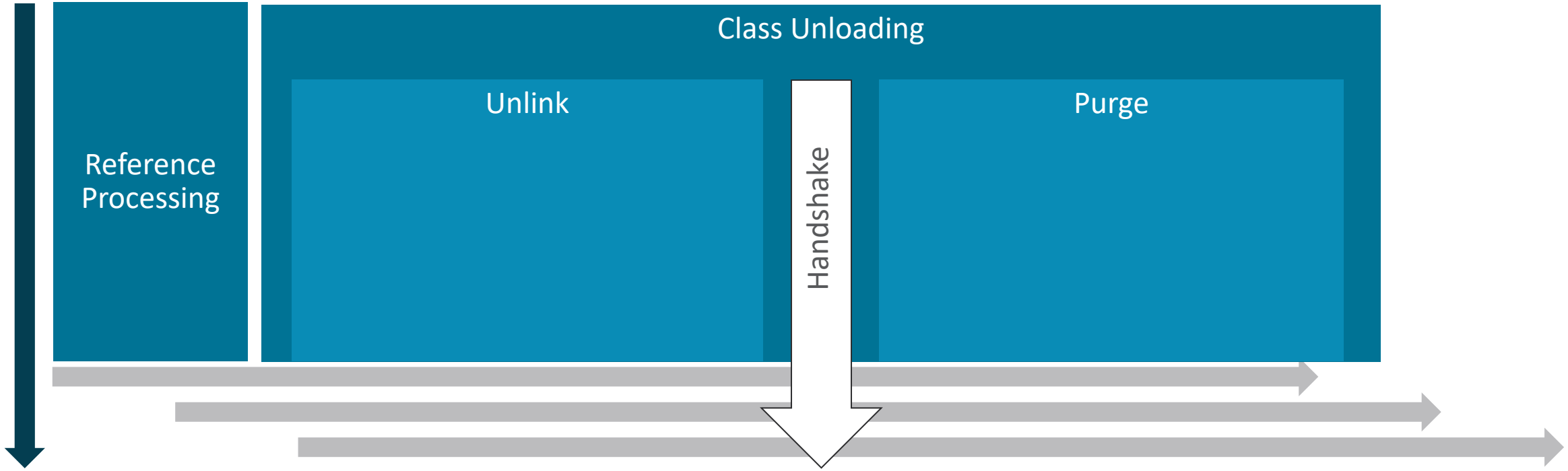
Inline caches

Basically everything is a **huge mess...**

...and we just continue **running anyway**

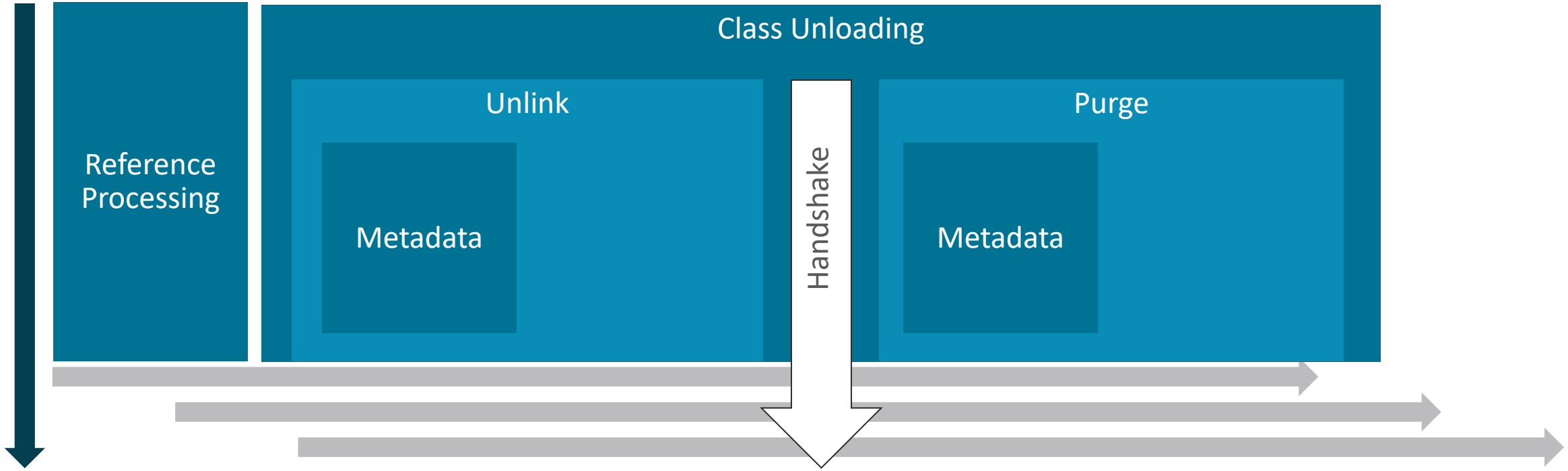
ZGC Unloading Overview

Pause Mark End



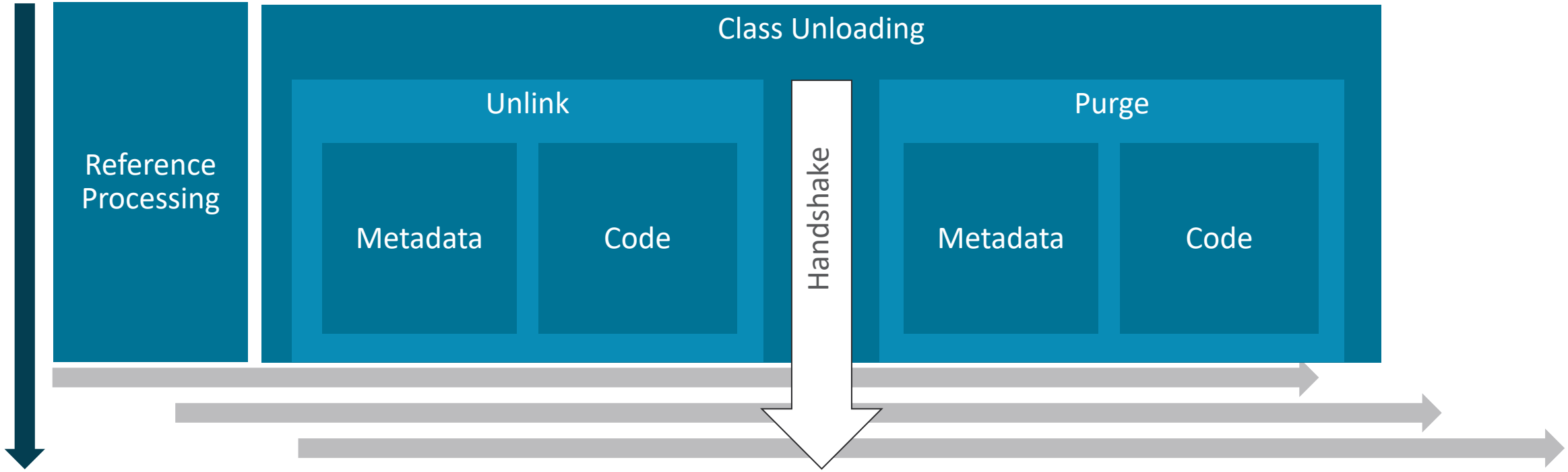
ZGC Unloading Overview

Pause Mark End



ZGC Unloading Overview

Pause Mark End



Concurrent Code Unloading

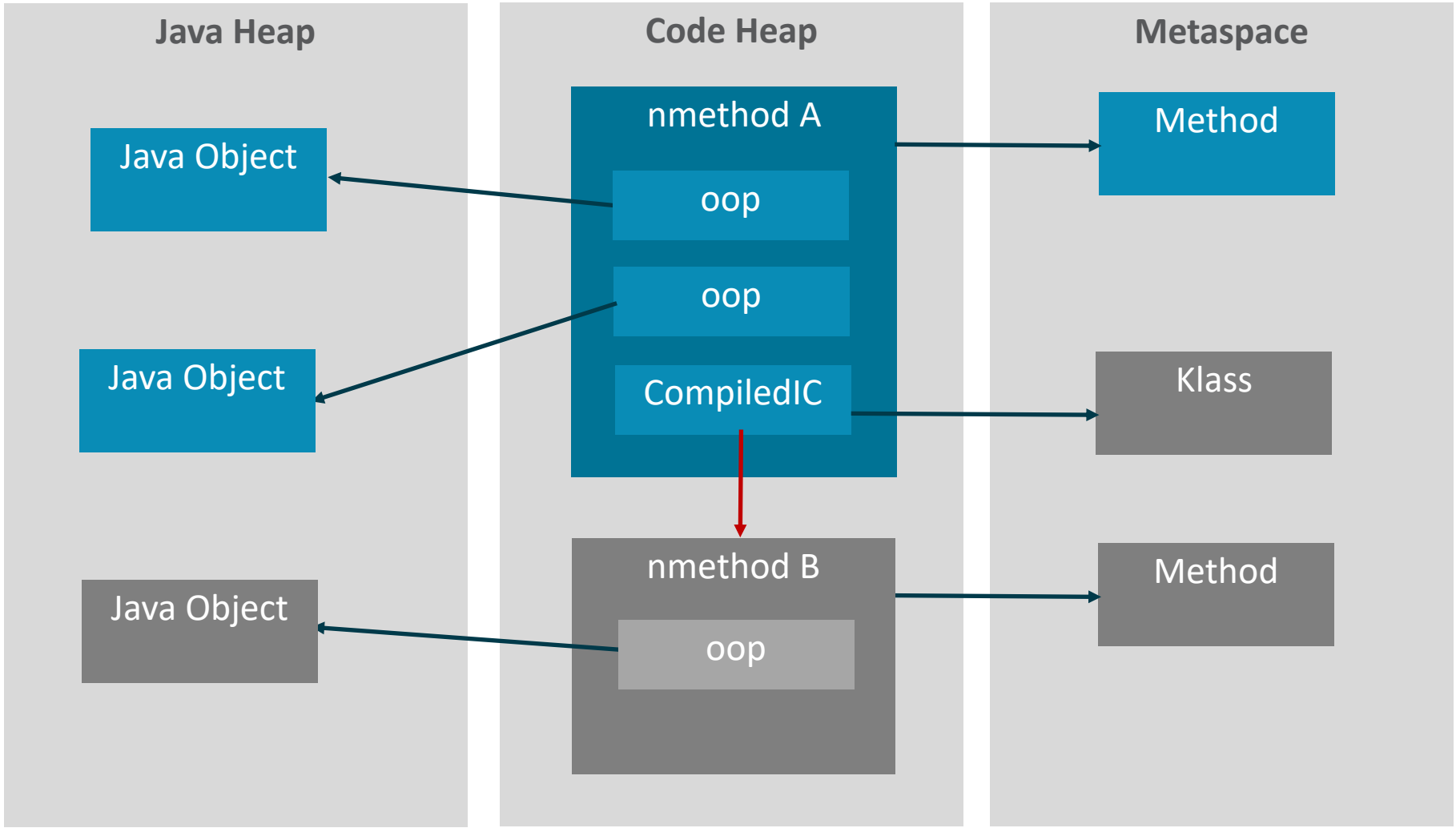
Code cache

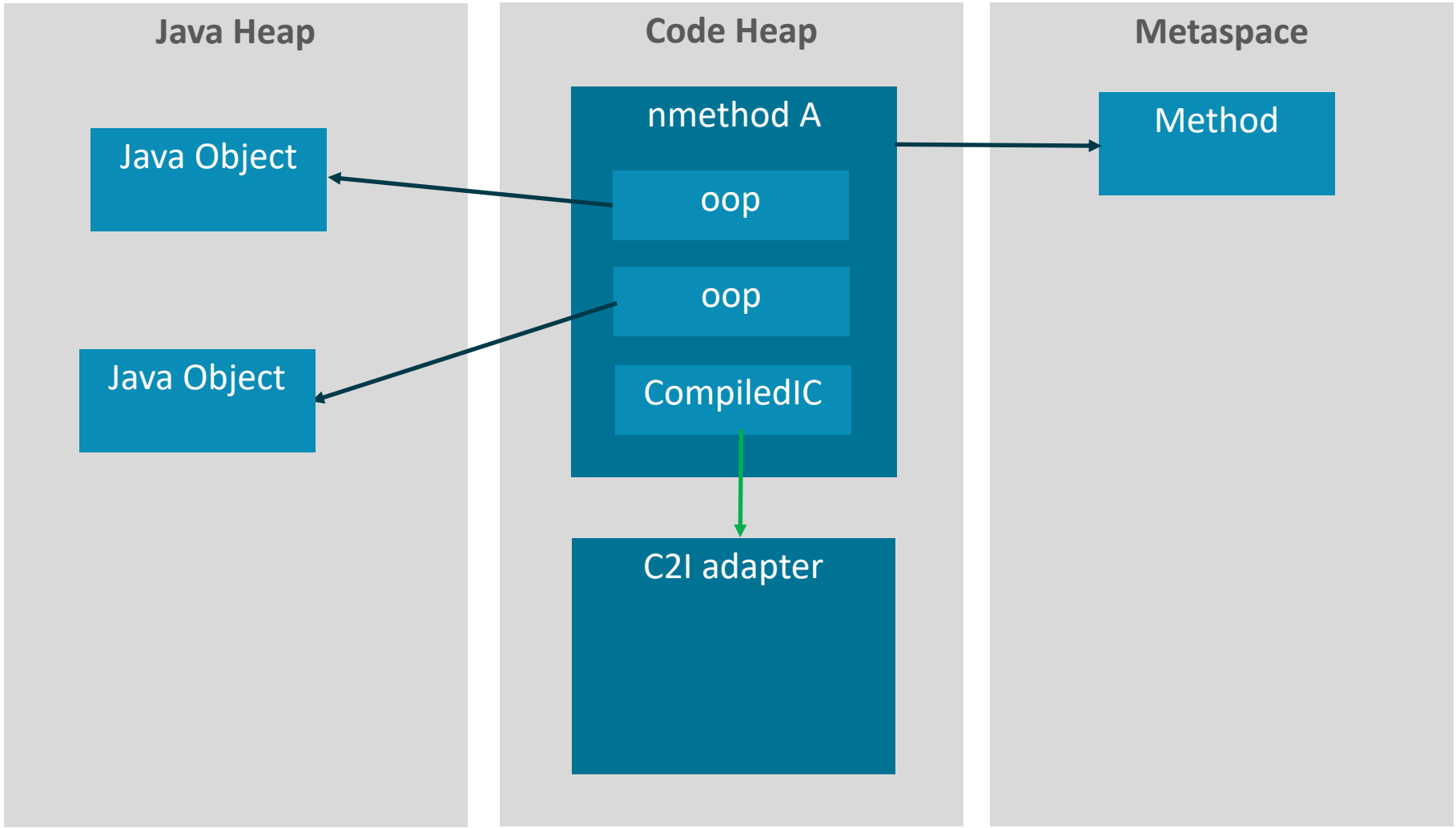
- Colored pointers into Java heap
  - Misaligned immediate values
  - Which color should native compiled object references have?
  - Need a way to paint native compiled object references
- Inline caches (CompiledIC) pointing at now dead native methods (nmethods), because of dead object references
  - Running any such code yields crashes
  - Need a way of preventing calls to dead native methods

NMethod Entry Barriers

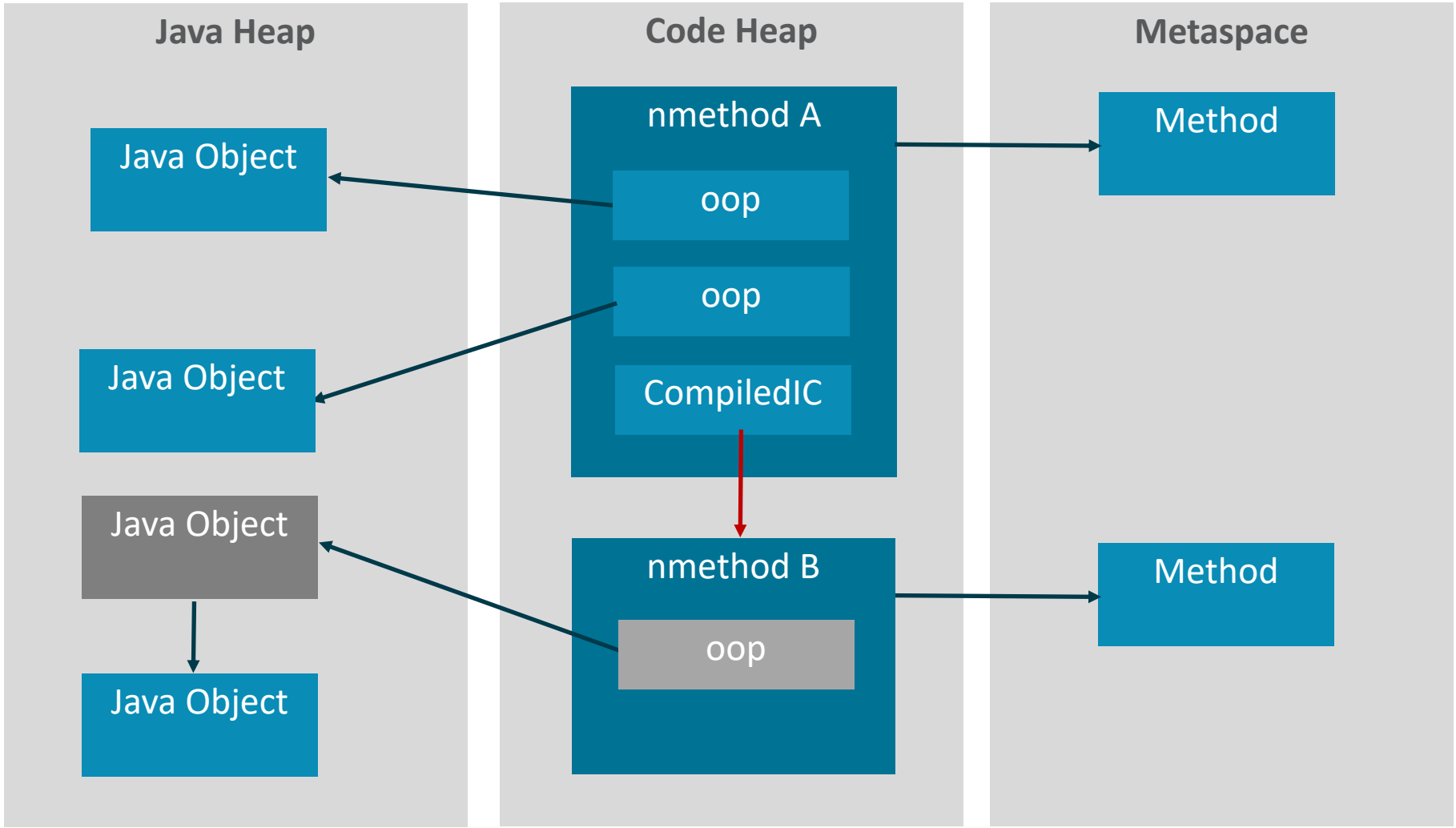
- Arm all nmethods not on stack in GC pause
  - Change global epoch value, caught with cmp; je; at verified entry
- Trap calls to armed nmethods
  - NMethods are "good" or "bad" based on object pointer liveness
- When entering good nmethods
  - Fix up object pointers (oops)
  - Disarm barrier by patching cmp immediate value
- When entering bad nmethods
  - Re-resolve the call

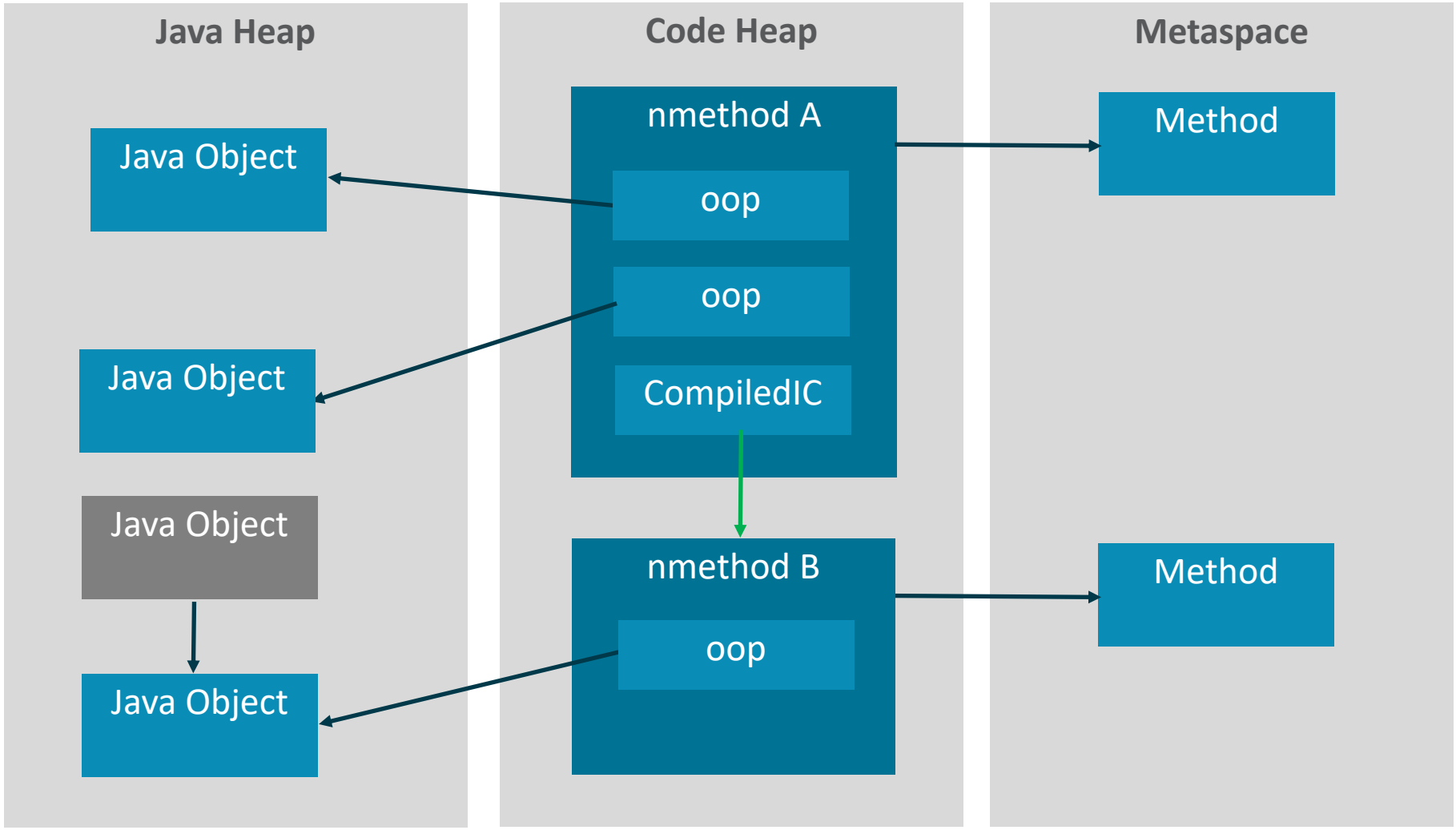
Example: Calling inline cache to dead nmethod





Example: Calling inline cache to stale but live nmethod





Concurrent Code Unloading

- Unlink
  - Clean stale inline caches (patch machine code that Java threads run)
  - Fixup object references (patching more machine code)
  - Disarm entry barriers (yet some more machine code patching)
  - Unlink nmethods from dependency contexts (lock-free unlinking)
  - Unlink exception caches (more lock-free unlinking)
- Global rendezvous handshake
- Purge
  - Purge unloading nmethods with make_unloaded()
  - Sweeper subsequently frees up memory in code cache

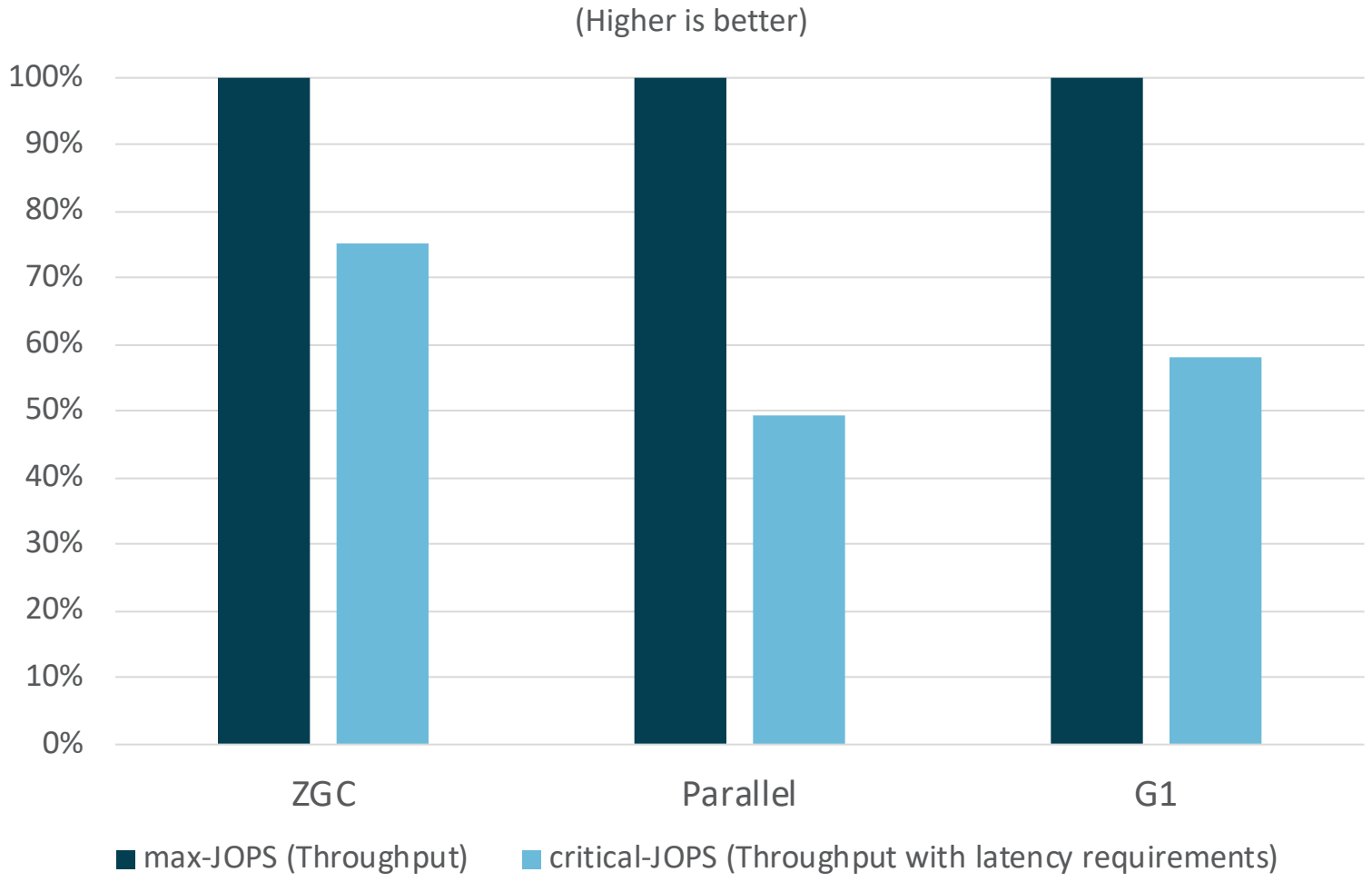
Concurrent Metadata Unloading

Structure

- Unlink
  - Expose logically already unlinked view of data to mutators
  - Subclass/sibling/implementor lists (lock-free)
  - Method data object (lock-free and per-MDO lock)
  - Protection domain cache (lock)
  - Class loader data graph (lock)
  - StringTable and SymbolTable (crazy concurrent)
- Rendezvous handshake
- Purge
  - Delete Klass, Method, CLD, handles, table entries, etc.

Evaluation

SPECjbb®2015 – Score



Mode: Composite

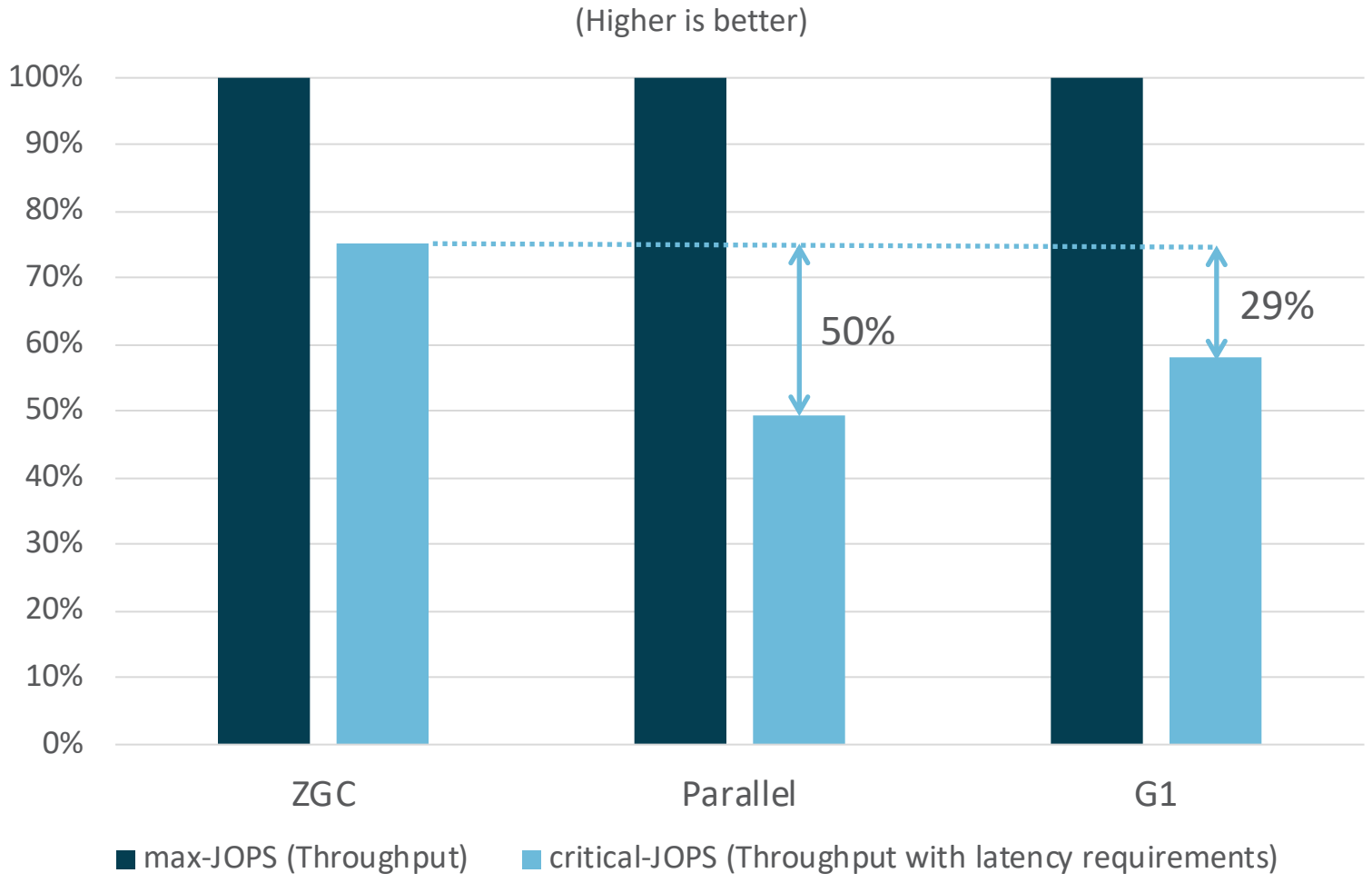
Heap Size: 128G

OS: Oracle Linux 7.4

HW: Intel Xeon E5-2690 2.9GHz
2 sockets, 16 cores (32 hw-threads)

SPECjbb®2015 is a registered trademark of the Standard Performance Evaluation Corporation (spec.org). The actual results are not represented as compliant because the SUT may not meet SPEC's requirements for general availability.

SPECjbb®2015 – Score



Mode: Composite

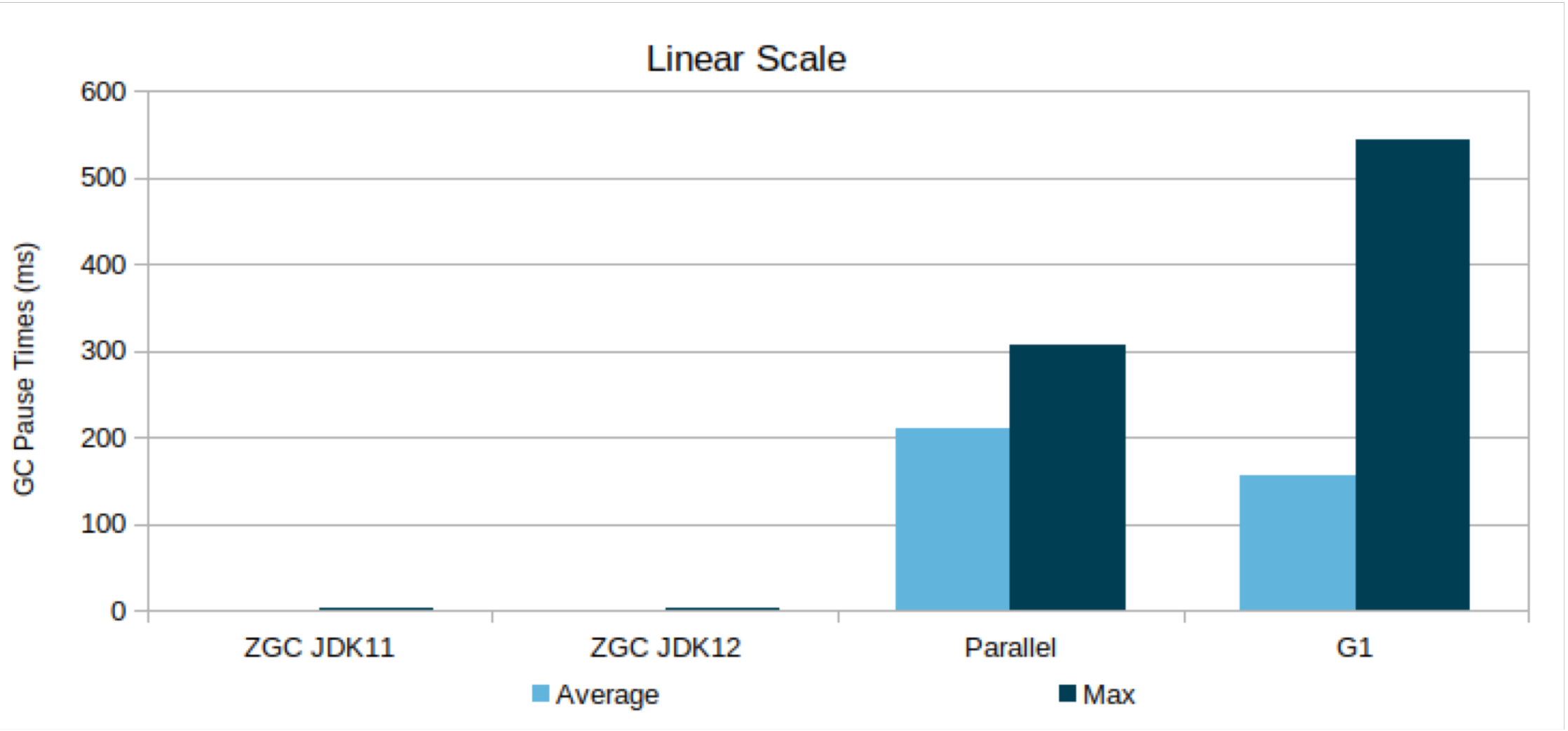
Heap Size: 128G

OS: Oracle Linux 7.4

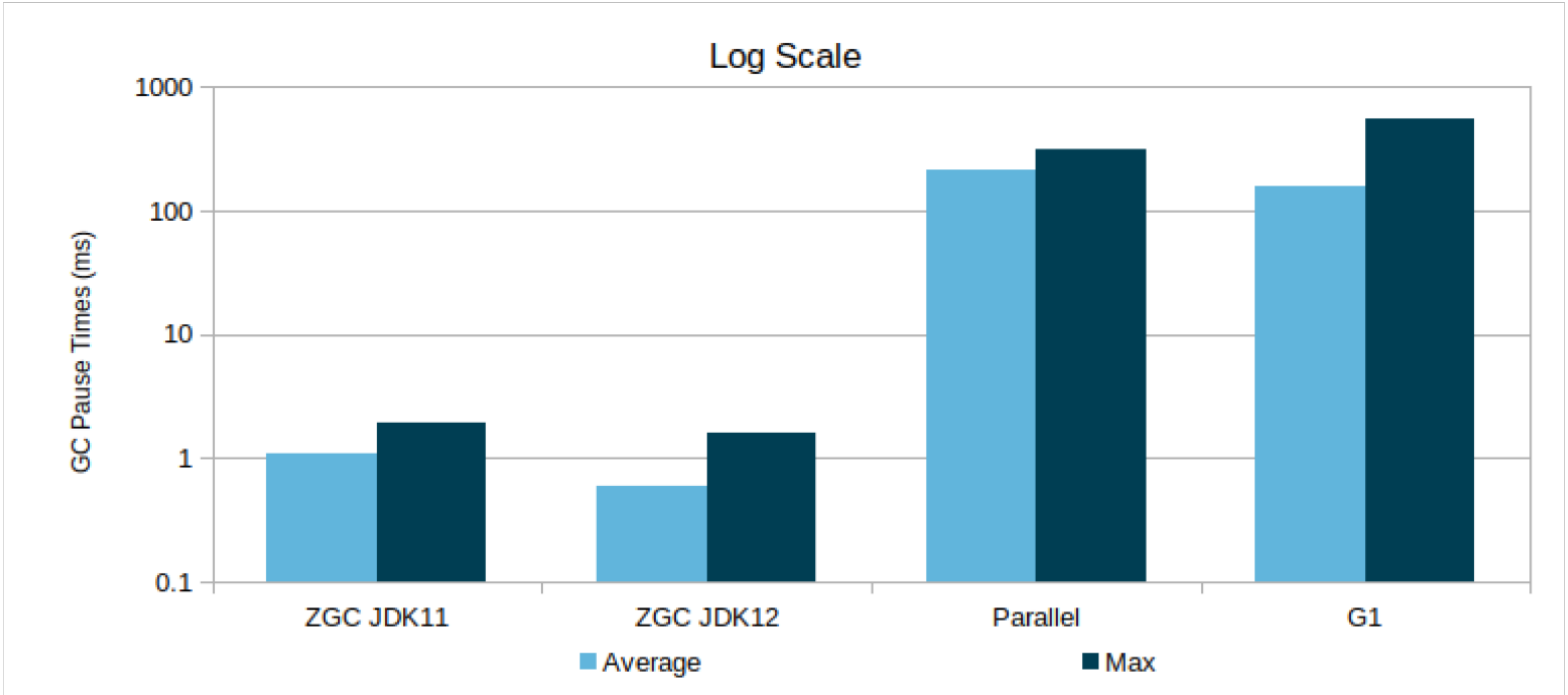
HW: Intel Xeon E5-2690 2.9GHz
2 sockets, 16 cores (32 hw-threads)

SPECjbb®2015 is a registered trademark of the Standard Performance Evaluation Corporation (spec.org). The actual results are not represented as compliant because the SUT may not meet SPEC's requirements for general availability.

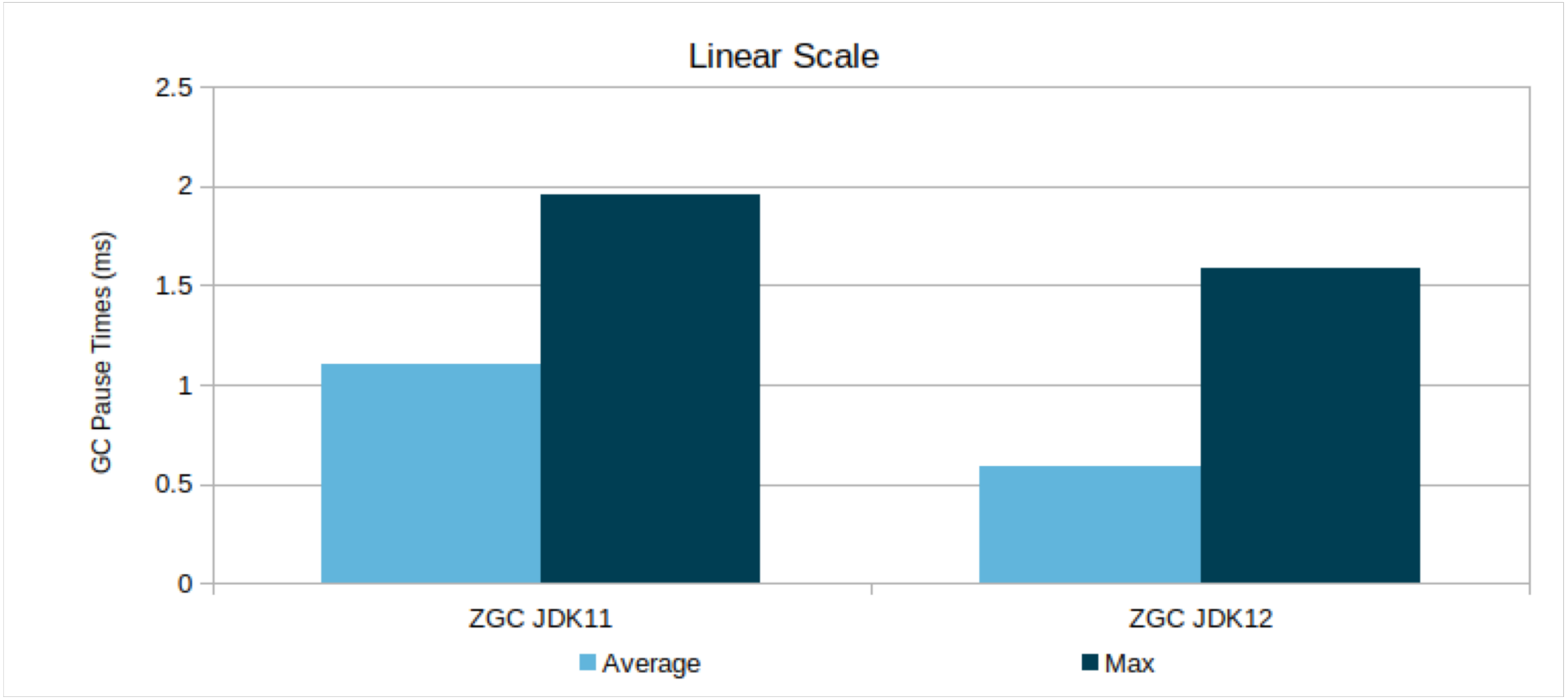
SPECjbb®2015 – Pause Times



SPECjbb®2015 – Pause Times



SPECjbb®2015 – Pause Times



Future Plans

Future Plans

- **Short-term**
  - Turn ZGC into a product feature

Future Plans

- **Short-term**
  - Turn ZGC into a product feature

Remove experimental status

Future Plans

- **Short-term**
  - Turn ZGC into a product feature

Future Plans

- **Short-term**
  - Turn ZGC into a product feature

- **Long-term**
  - Generational
  - Sub-millisecond max pause times
  - Additional platform support
  - Graal JIT support

Future Plans

- **Short-term**
  - Turn ZGC into a product feature
- **Long-term**
  - Generational
  - Sub-millisecond max pause times
  - Additional platform support
  - Graal JIT support

Generational
- Withstand higher allocation rates
- Lower heap overhead
- Lower CPU usage

Future Plans

- **Short-term**
  - Turn ZGC into a product feature
- **Long-term**
  - Generational
  - Sub-millisecond max pause times
  - Additional platform support
  - Graal JIT support

Sub-millisecond max pause times
- Within reach
- Reduce root set size
- Time-to-Safepoint, etc

Future Plans

- **Short-term**
  - Turn ZGC into a product feature
- **Long-term**
  - Generational
  - Sub-millisecond max pause times
  - Additional platform support
  - Graal JIT support

Additional platform support
- macOS?
- Windows?
- Sparc?
- Aarch64?

Future Plans

- **Short-term**
  - Turn ZGC into a product feature

- **Long-term**
  - Generational
  - Sub-millisecond max pause times
  - Additional platform support
  - Graal JIT support

OpenJDK

Get Involved!

Java™ ORACLE

ZGC Project

Follow, Participate, Give Feedback

OpenJDK

zgc-dev@openjdk.java.net

http://wiki.openjdk.java.net/display/zgc/Main

Java™ ORACLE

ZGC Project

Source Code

OpenJDK

Latest Stable

http://hg.openjdk.java.net/jdk/jdk

Bleeding Edge

http://hg.openjdk.java.net/zgc/zgc

Java™ ORACLE

Thanks!

Questions?


Java™
ORACLE®