



Java 2024

Hablemos de Java y Java 23

JoeDayz

José Amadeo Díaz Díaz



Fundador de JoeDayz.pe



Java Champion



Miembro de @PeruJUG



[@jamdiazdiaz](https://twitter.com/jamdiazdiaz)



<https://github.com/joedayz>



ORACLE®

Agenda

1. Historia
2. Calendario de Lanzamientos
3. Versiones, parches y actualizaciones
4. Beneficios
5. Lo nuevo de Java 23



Nuestro mundo Movido por Java

Confianza

Modelo de desarrollo abierto y transparente.
Comunidad global de desarrolladores de Java que se ayudan mutuamente a tener éxito.

Innovación

Innovación continua para satisfacer las necesidades actuales mientras se construye para el futuro.

Previsibilidad

Enfoque implacable en la productividad del desarrollador, la seguridad y la estabilidad de la plataforma, la disponibilidad oportuna de nuevos lanzamientos.

29

Años desde Su lanzamiento

10M

Desarrolladores Java Full-time²

75%

Desarrollo Cloud Native con Java²

#1

Lenguaje para las tendencias tech¹

#1

Lenguaje para desarrollo cloud native¹

#1

Lenguaje para iniciativas de seguridad¹

60B

JVM activas de forma global en el 2021¹

38B

De JVMs basadas en cloud en el 2021¹

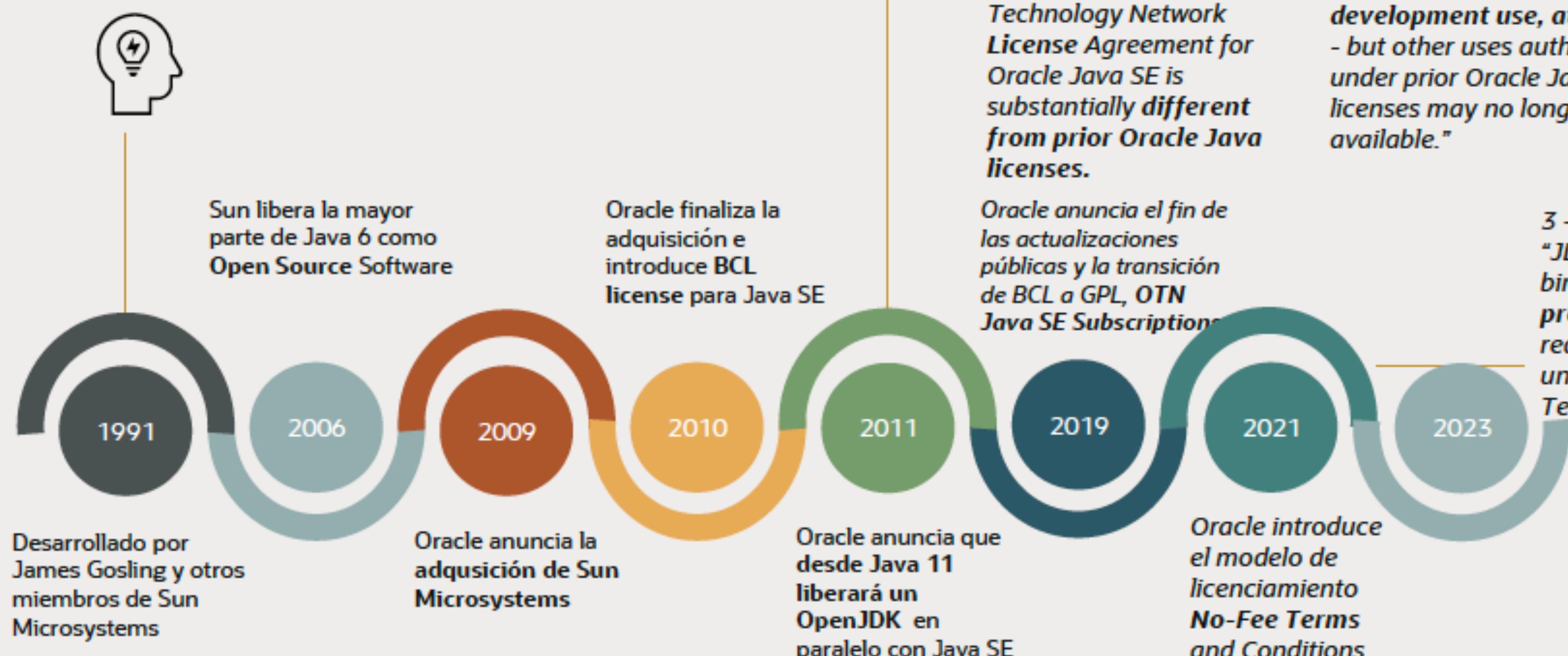
1 VDC 2022 study

2 IDC 2021 study



Java en el tiempo

Línea de Tiempo



1 - PM's Blog:
"Starting with Java 11,
Oracle will provide
OpenJDK releases under
the open source GNU
General Public License v2.
Functionally compatible
and interchangeable
with Java SE"

2 - "The Oracle
Technology Network
License Agreement for
Oracle Java SE is
substantially different
from prior Oracle Java
licenses."

"Personal use and
development use, at no cost -
but other uses authorized
under prior Oracle Java
licenses may no longer be
available."

3 - PM's Blog:
"JDK 18 and JDK 17
binaries are free to use in
production and free to
redistribute, at no cost,
under the Oracle No-Fee
Terms and Conditions."

Source:

1 - <https://blogs.oracle.com/java/post/oracle-jdk-releases-for-java-11-and-later>

2 - <https://www.java.com/en/download>

3 - <https://www.oracle.com/java/technologies/downloads/>





Versiones, Parches y Actualizaciones

Lanzamiento de Java	Fecha de lanzamiento	Final de actualizaciones públicas (EoPU)	Ultima version Publica	Soporte extendido hasta	Version mas reciente	Vulnerabilidades de seguridad solucionadas desde EoPU 	Tickets de soporte solucionados
Java 6	Dic 2006	Abr 2013	6.0_45	Sep 2018	6.0_221	330	-
Java 7	Jul 2011	Abr 2015	7.0_80	Jul 2022	7.0_431	275	1726
Java 8	Mar 2014	Enero 2019	8.0_201/202	Dic 2030	8.0_421	148	3338
Java 11*	Sep 2018	N/A	N/A	Ene 2032	11.0.24	137	4826
Java 17 (LTS)	Sep 2021	Sep 2024	TBC	Sep 2029	17.0.12	66	2719
Java 21 (LTS)	Sep 2023	Sep 2026	21.0.1	Sep 2031	21.0.4	11	566
Java 22	Mar 2024	Sep 2024	22.0.1	No Aplica	22.0.2		

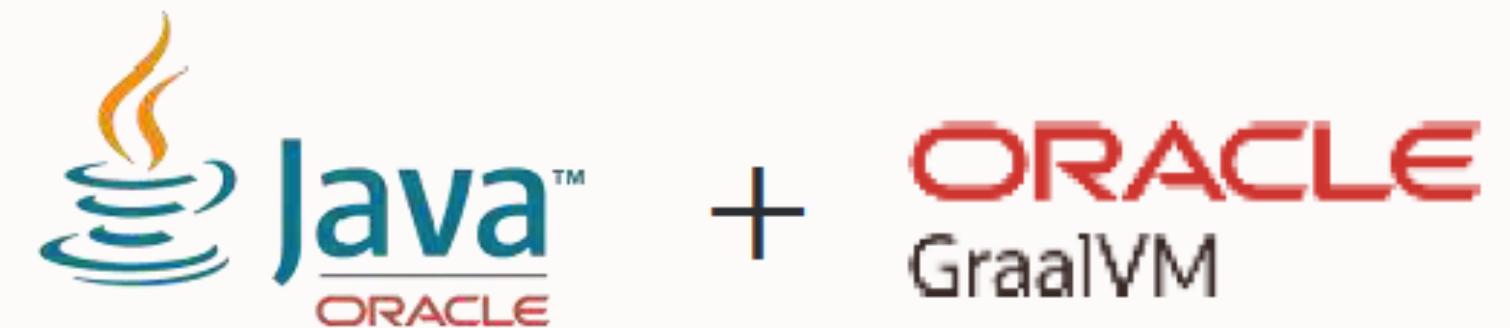
Ultima actualización: Julio de 2024

Información adicional:

<https://www.oracle.com/java/technologies/java-se-support-roadmap.html>

<https://www.java.com/releases/matrix/>

<https://www.oracle.com/java/technologies/javase/22all-relnotes.html>



High perf compiler



Native image enables
microservices



Multi-language on
one JVM



24x7 support

<https://www.oracle.com/graalvm>

<https://bit.ly/WebinarsPeruJUG>

Beneficios de Graal VM



1. Mejora el rendimiento

- Promedio 30% más rápido en puntos de referencia del mundo real.
- Reducir la latencia para la recolección de basura.
- Optimizaciones patentadas que maximizan el rendimiento.

2. Impulsa la innovación

- Implementar microservicios y modernizar aplicaciones.
- Las aplicaciones compiladas con anticipación se inician en un 94% más rápido que en JVM.
- Ejecute varios idiomas en una maquina virtual Java (JVM).

3. Reduce costos

- Ejecute más cargas de trabajo con los mismos servidores.
- Reduzca el consumo de memoria entre 5-7 veces menos.
- Reduzca el costo de TI entre 12% - 35% menos de servidores.

Modernización poliglota de alto rendimiento



- Start lenguaje C (polyglot.c)
- Start Lenguaje R (polyglot.R)
- Start Lenguaje Ruby (polyglot.rb)
- Start Lenguaje Python (polyglot.py)

Start from JavaScript / Node.js

Create the file `polyglot.js`:

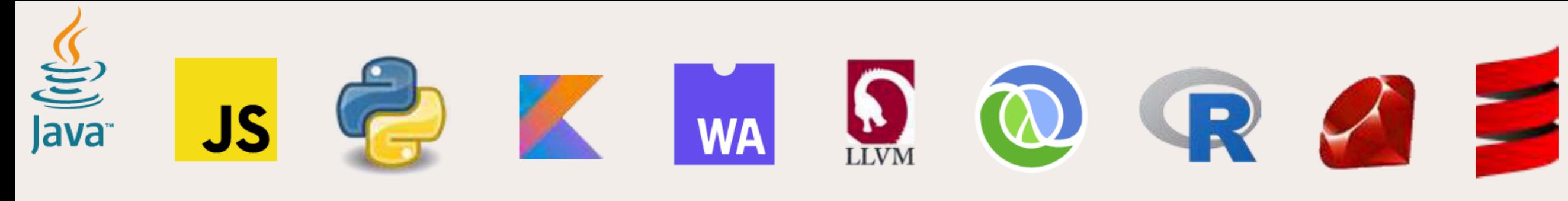
Ruby Python Java LLVM

Target Language

```
var array = Polyglot.eval("R", "c(1,4)")
console.log(array[2]);
```

Run:

```
js --polyglot --jvm polyglot.js
42
node --polyglot --jvm polyglot.js
42
```



Modernización poliglota de alto rendimiento



- Rapido
 - 50% mas rapido según benchmarks
 - Ejecutables nativos con inicio casi instantáneo
- Inteligente
 - 60+ optimización de compilación
 - Compilación nativa de ejecutables
 - Múltiples lenguajes soportados
- Ahorrador
 - Menor consumo de memoria
 - Mayor rendimiento y menos uso de CPU
 - Reducción de carga de Garbage Collection
- Barato
 - Ejecute más cargas de trabajo con los mismos servidores
 - Reduzca el consumo de memoria hasta 5-7 veces menos
 - Reduzca el costo de TI con hasta un 12-35% menos de servidores.



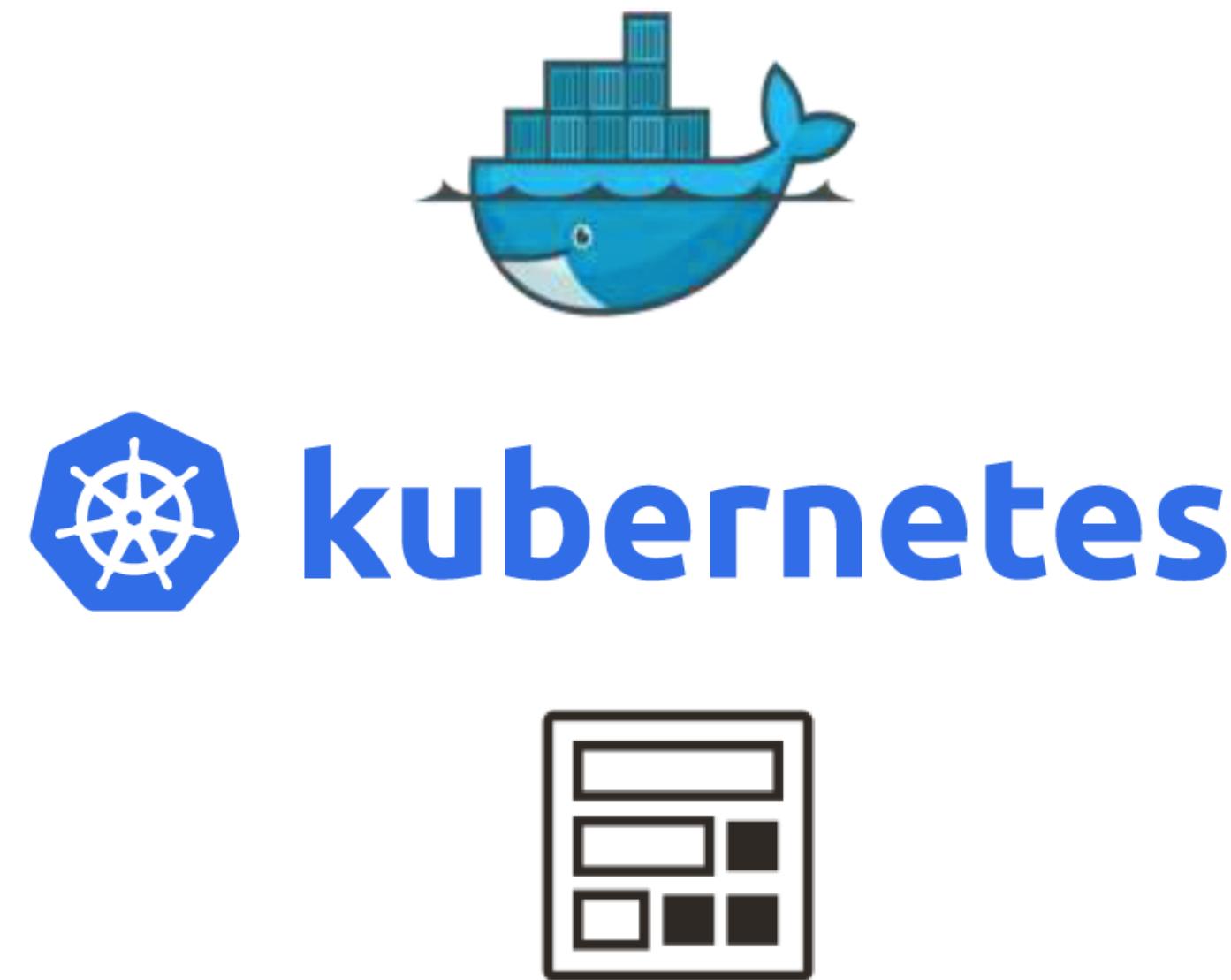
Compilación anticipada de imágenes nativas Empresariales de GraalVM



Compatible con frameworks y
plataformas de microservicios



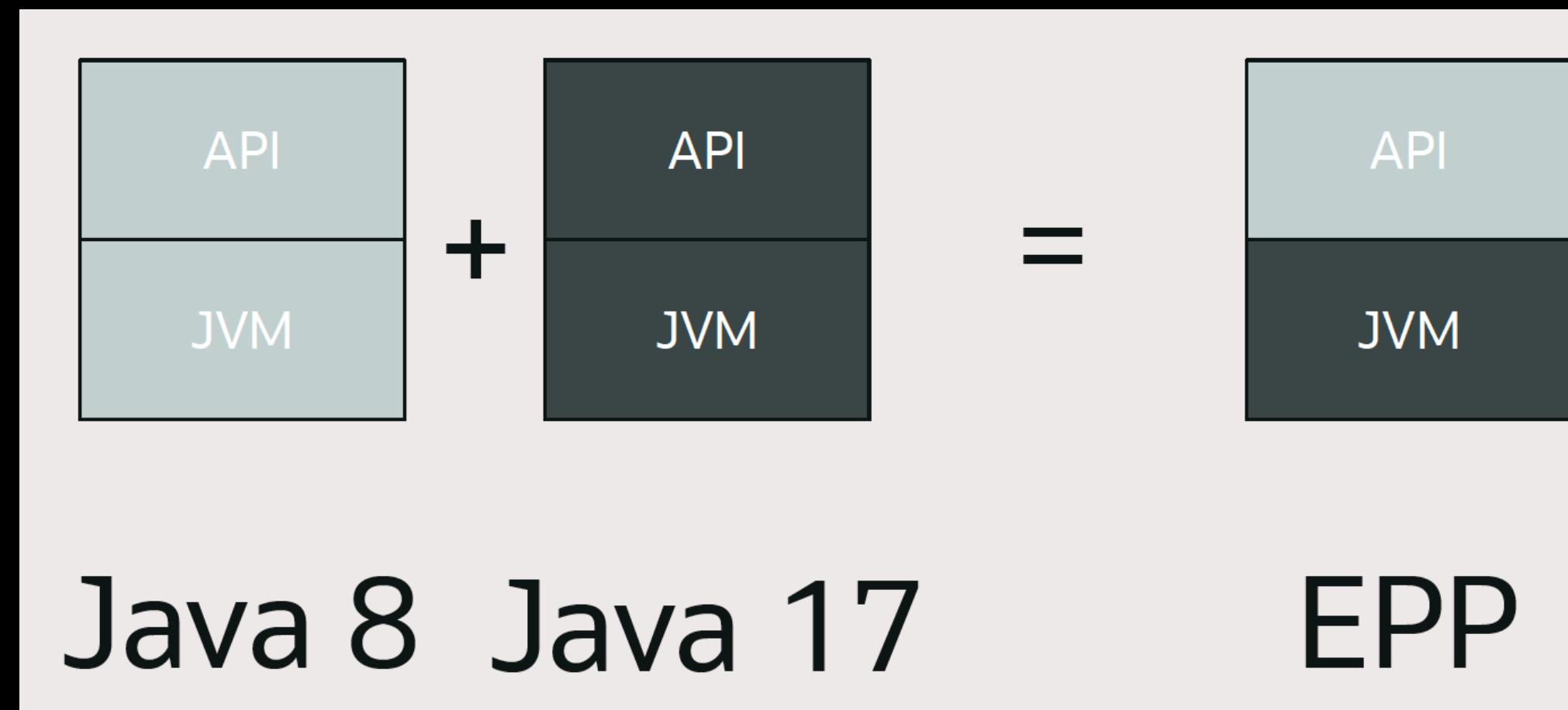
Ideal para contenedores y
serverless



Enterprise Performance Pack



- Al usar la VM de Java 17 VM, EPP entrega las mejoras en el desempeño de Java 17 a las cargas de trabajo de Java 8, utilizando menos memoria.



- Incorpore las mejoras de Java 17 a sus aplicaciones de Java 8 sin realizar ningún cambio en el código.
- Obtenga el mismo roadmap de soporte de Java 8
- Enfocado a aplicaciones desarrolladas en Java 8 que no están siendo actualizadas ni tienen desarrollos activos.
- Disponible únicamente para aplicaciones headless en Linux de 64 bits (x86 & ARM)

<https://bit.ly/OracleEPP>

Oracle hace avanzar a Java



- Java ofrece innovación continua y programada, con funciones modernas de lenguaje de programación, mejoras en la experiencia del desarrollador, servicios administrados en la nube y estabilidad, siendo especialmente relevante para las aplicaciones y cargas de trabajo modernas.

Oracle supports the Java community and ecosystem around the world

OpenJDK



ORACLE
University

ORACLE
Academy

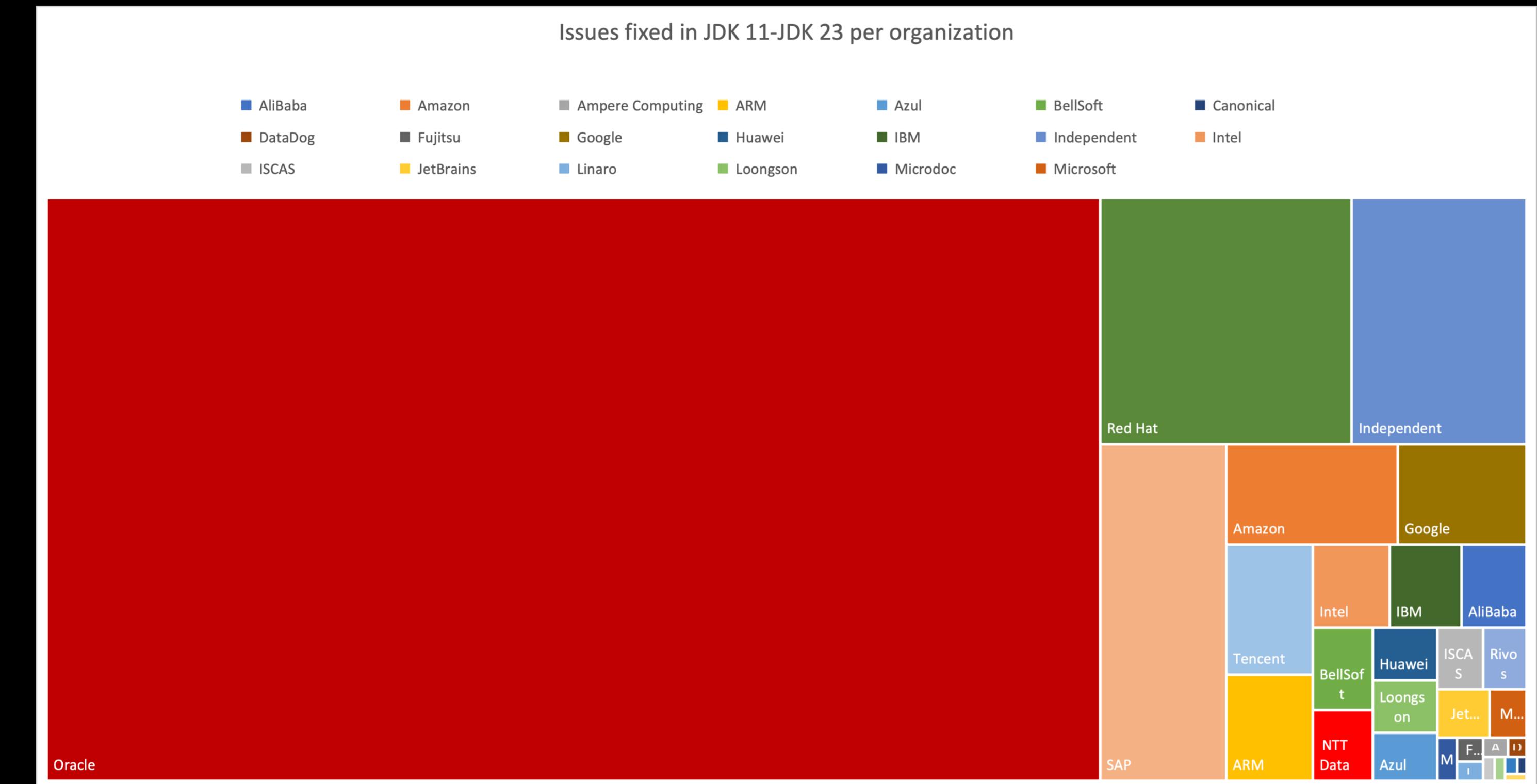
Java
magazine



Oracle es el principal contribuyente



- Autor principal y colaborador de la tecnología Java
- Patrocinador líder y administrador del ecosistema de Java
- Impulsor líder de la innovación de plataformas.



Beneficios plus



- JDK Mission Control (JMC)
Es un conjunto avanzado de herramientas para administrar, monitorear, crear perfiles



<https://bit.ly/OracleJMC>



Lo nuevo en Java 23



The Hotspot Garbage Collectors

GC	OPTIMIZED FOR
Serial	Memory footprint
Parallel	Throughput
G1	Throughput/Latency
ZGC	Low Latency and Scalability



Try them out!

Start with the default GC, G1

- `-XX:+UseG1GC` (don't need to specify since it is default)

If the throughput is insufficient , try enabling Parallel

- `-XX:+UseParallelGC`

If pause times are too long, try ZGC

- `-XX:+UseZGC`



Performance tuning

If most of the characteristics are pretty close, try tuning

GC tuning guide

- <https://docs.oracle.com/en/java/javase/20/gctuning/introduction-garbage-collection-tuning.html>

GC tuning



Help Center HotSpot Virtual Machine Garbage Collection Tuning Guide Search

Java / Java SE / 19

HotSpot Virtual Machine Garbage Collection Tuning Guide

Table of Contents

- Title and Copyright Information
- Preface
- 1 Introduction to Garbage Collection Tuning**
 - What Is a Garbage Collector?
 - Why Does the Choice of Garbage Collector Matter?
 - Supported Operating Systems in Documentation
- 2 Ergonomics
- 3 Garbage Collector Implementation
- 4 Factors Affecting Garbage Collection Performance
- 5 Available Collectors
- 6 The Parallel Collector
- 7 Garbage-First (G1) Garbage Collector**
 - Introduction to Garbage-First (G1) Garbage Collector
 - Enabling G1
- Basic Concepts

1 Introduction to Garbage Collection Tuning

A wide variety of applications, from small applets on desktops to web services on large servers, use the Java Platform, Standard Edition (Java SE). In support of this diverse range of deployments, the Java HotSpot VM provides multiple garbage collectors, each designed to satisfy different requirements. Java SE selects the most appropriate garbage collector based on the class of the computer on which the application is run. However, this selection may not be optimal for every application. Users, developers, and administrators with strict performance goals or other requirements may need to explicitly select the garbage collector and tune certain parameters to achieve the desired level of performance. This document provides information to help with these tasks.

First, general features of a garbage collector and basic tuning options are described in the context of the serial, stop-the-world collector. Then specific features of the other collectors are presented along with factors to consider when selecting a collector.

Topics

- What Is a Garbage Collector?
- Why Does the Choice of Garbage Collector Matter?
- Supported Operating Systems in Documentation

What Is a Garbage Collector?

The garbage collector (GC) automatically manages the application's dynamic memory allocation requests.

A garbage collector performs automatic dynamic memory management through the following operations:

- Allocates from and gives back memory to the operating system.
- Hands out that memory to the application as it requests it.
- Determines which parts of that memory is still in use by the application.

ZGC

A Scalable Low-Latency Garbage Collector





Project ZGC

- Jan 2014 Prototyping starts
- Sep 2018 First experimental release (JDK 11)
- Mar 2019 Concurrent class unloading (JDK 12)
- Sep 2019 AArch64 support (JDK 13)
- Mar 2020 Windows & macOS support (JDK 14)
- Sep 2020 First production ready release (JDK 15)
- Mar 2021 Constant pause times (JDK 16)
- Sep 2021 Dynamic tuning of GC threads (JDK 17)

Goals



Scalable

Up to **TB** heaps

Auto-tuning

Simpler configuration

Low Latency

< 1 ms pauses

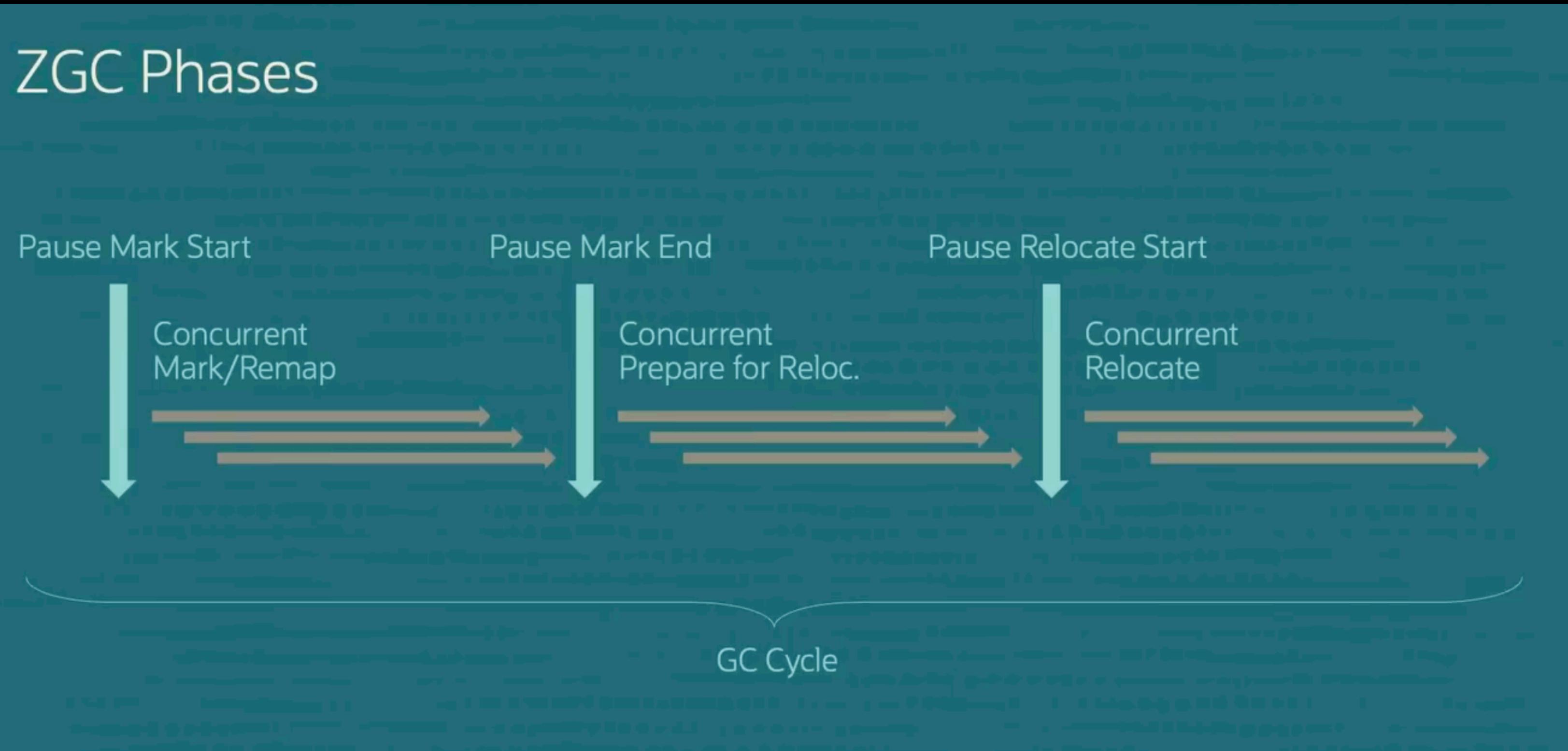
15%

Max application
throughput reduction



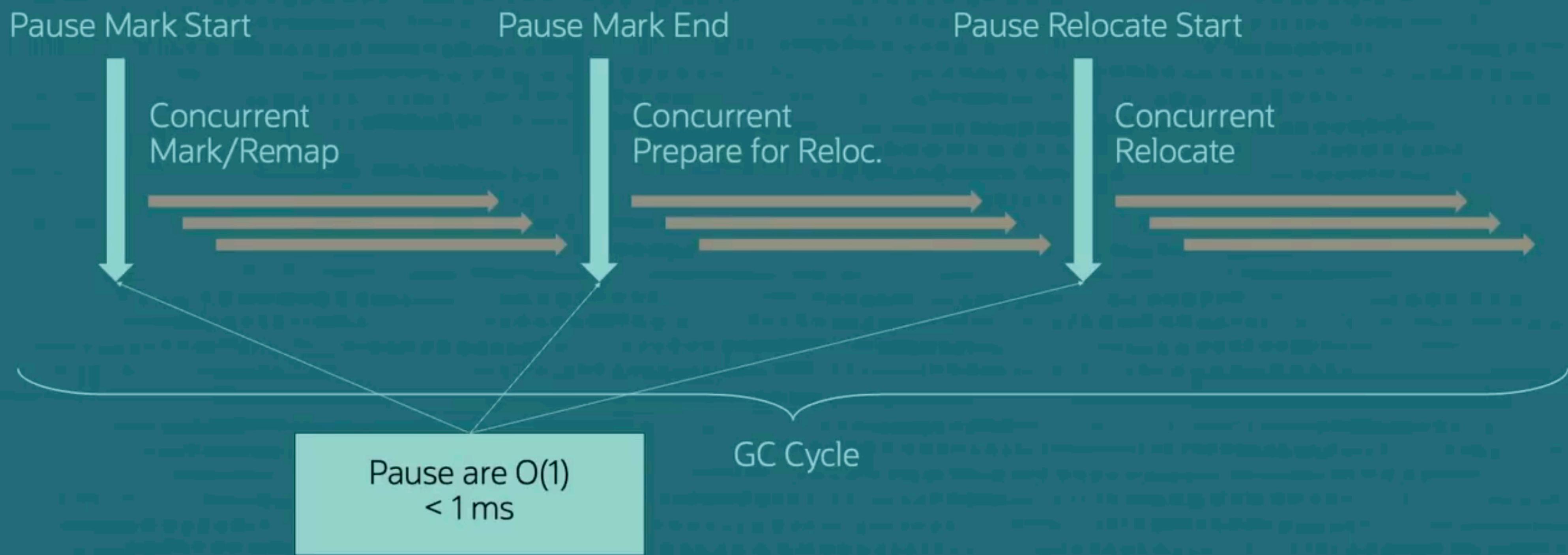
ZGC at a Glance

- Concurrent
 - All heavy lifting is done while Java threads continue to execute
- Constant pause times
 - Does not scale with the heap size, live-set size, etc
- Parallel
 - Uses multiple threads to collect garbage
- Compacting
 - Moves objects around to fight fragmentation
- Region-based
 - Focuses on regions with lots of garbage
- NUMA-aware
 - Allocates objects in memory local to the CPU
- Auto tuning
 - No complex configuration knobs
- Load barriers & Colored pointers
 - The two main techniques used to achieve concurrent garbage collection

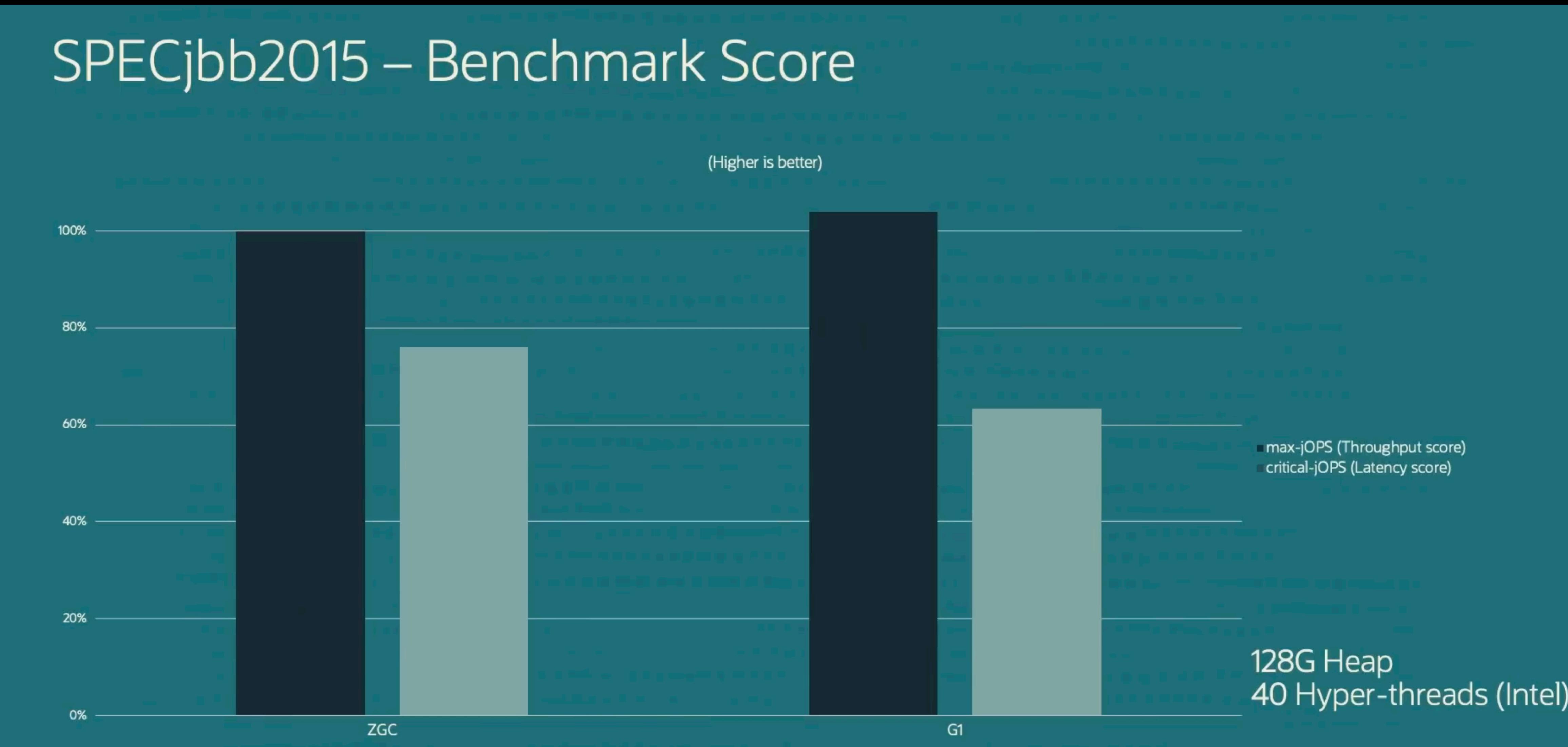




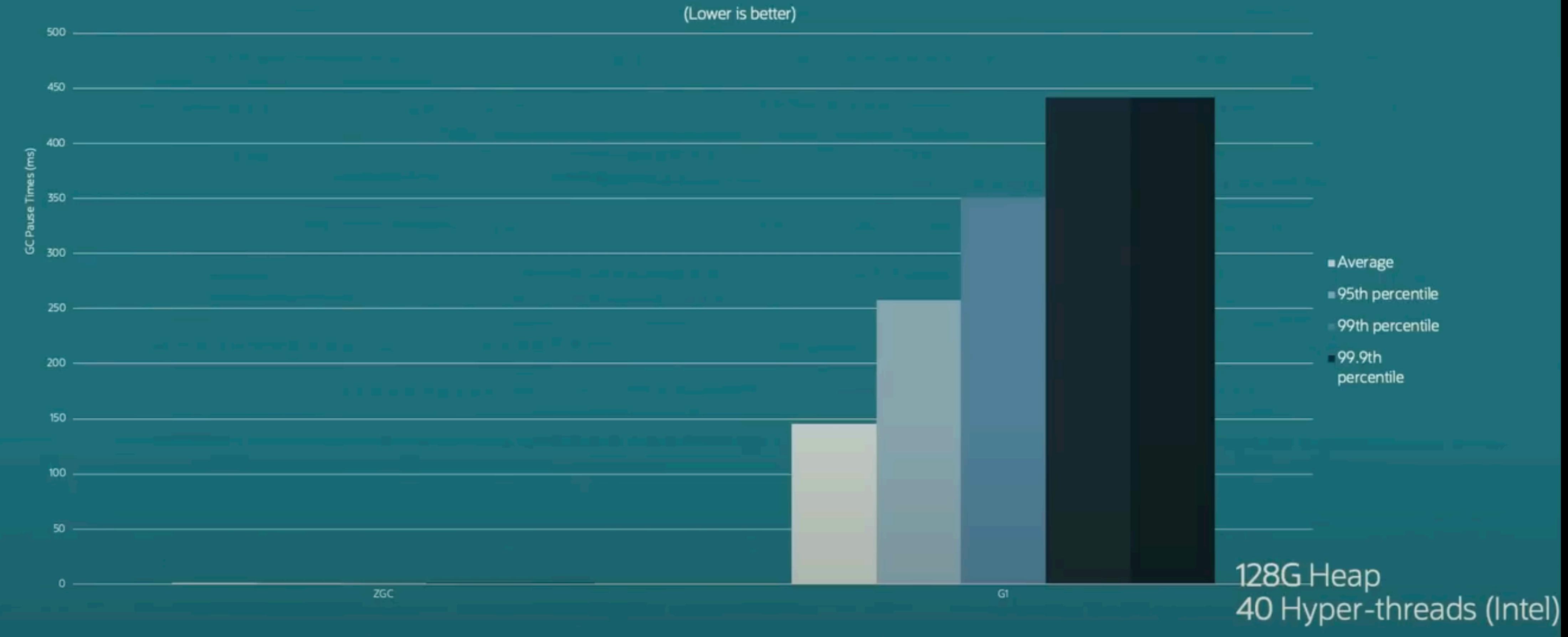
ZGC Phases



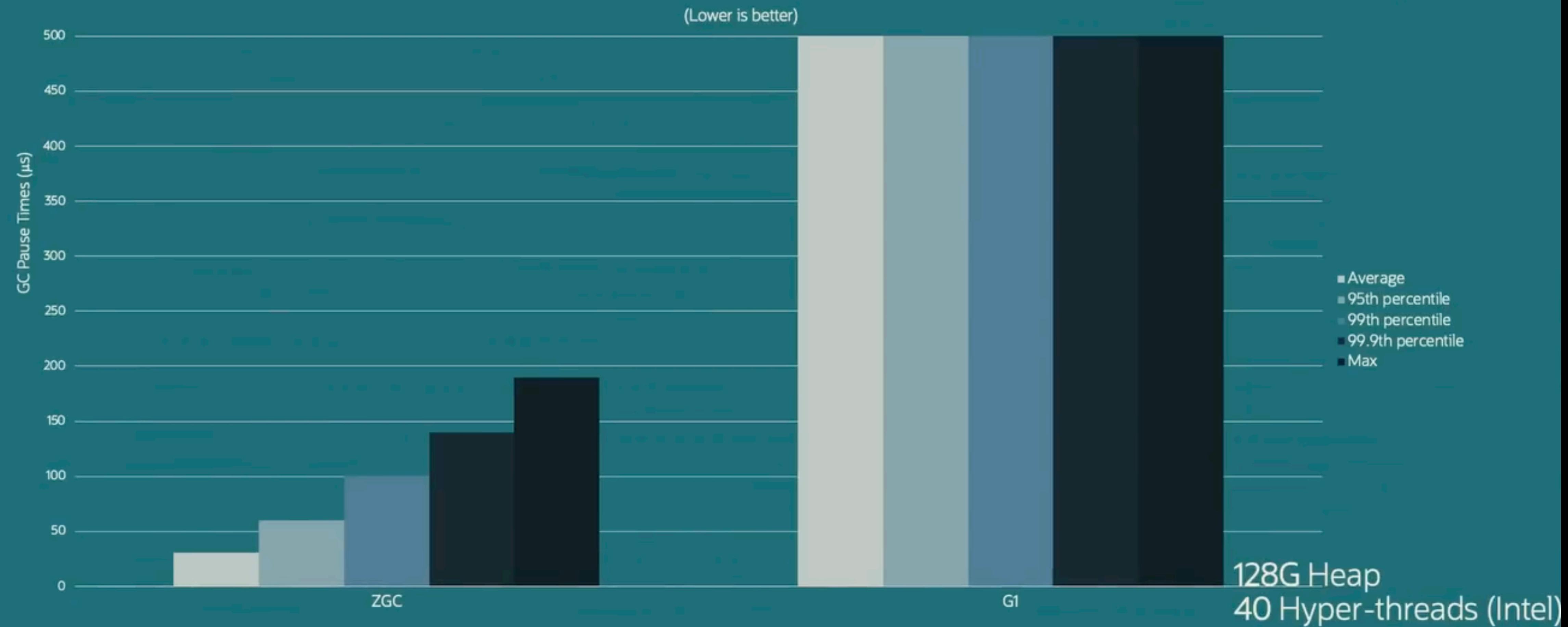
SPECjbb2015 – Benchmark Score



SPECjbb2015 – GC Pause Times



SPECjbb2015 – GC Pause Times



Performance tuning ZGC

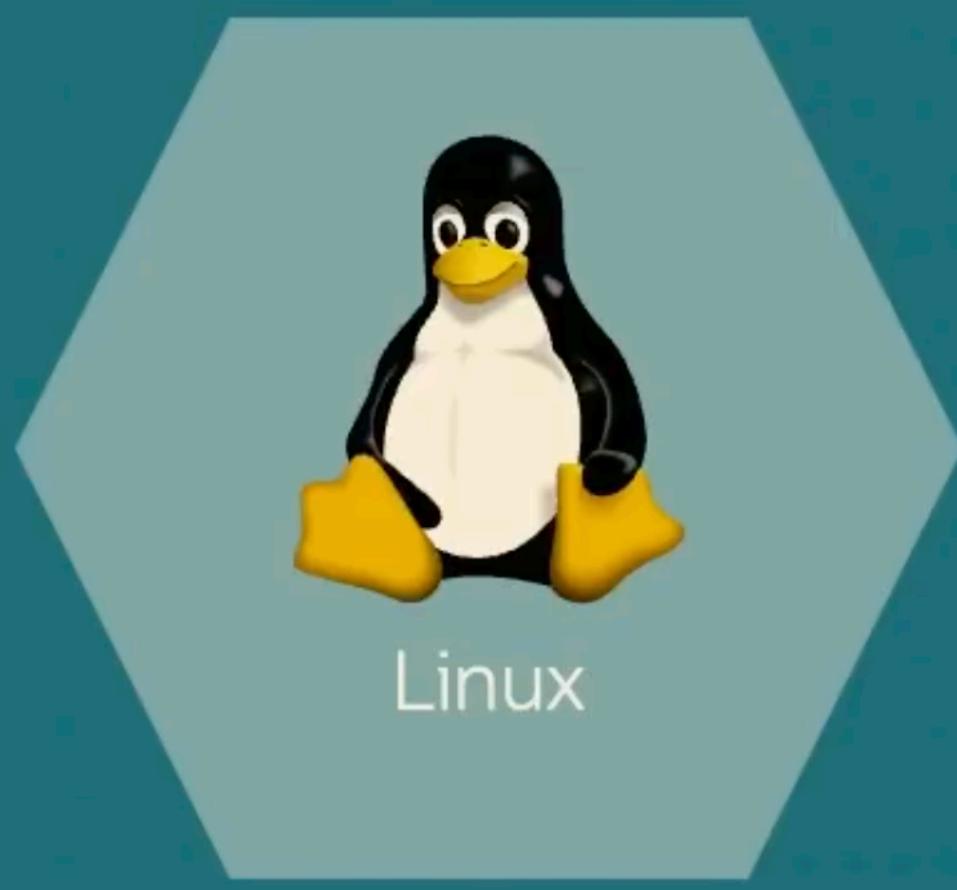
Automatic Tuning

- Dynamic number of threads
 - (No `-XX:ConcGCThreads` needed)

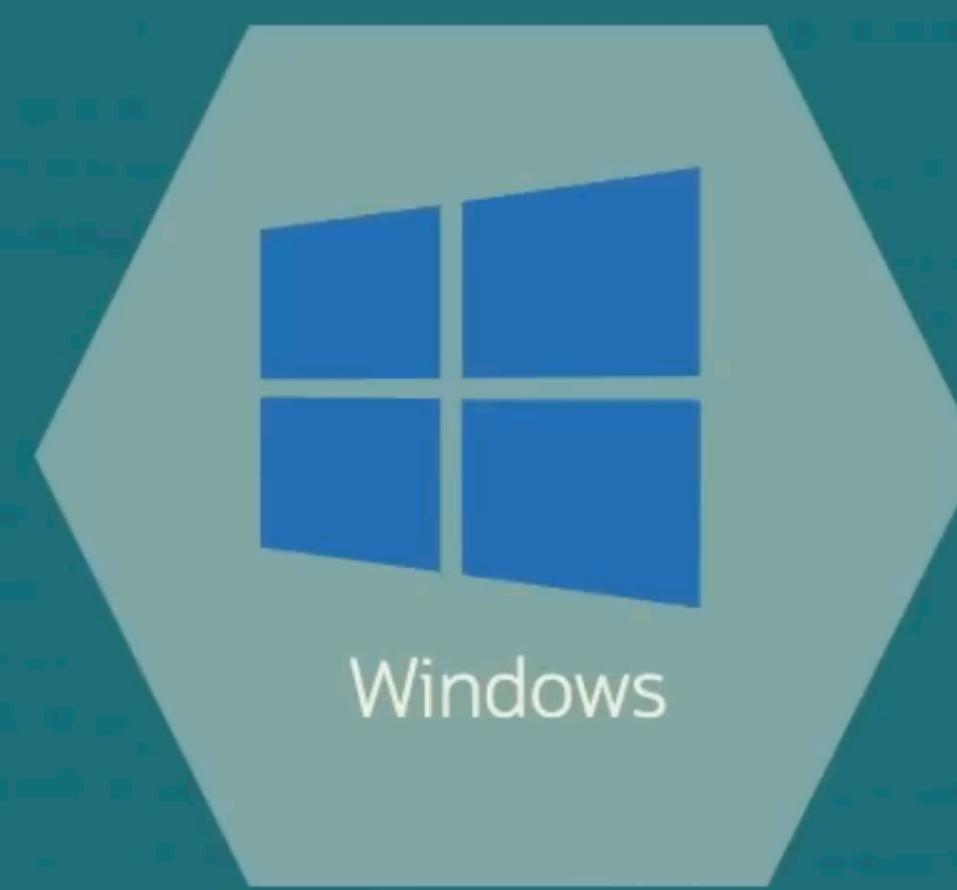
Max heap size

- `-Xmx`

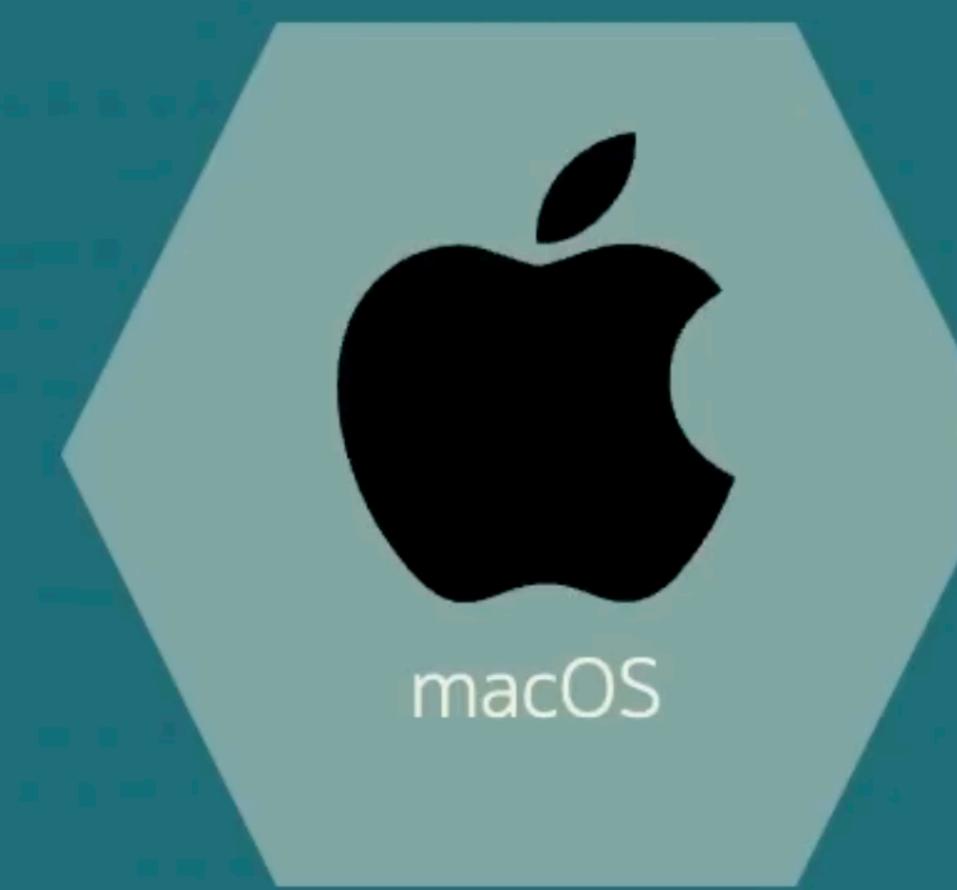
Available on All Commonly Used Platforms



Linux



Windows



macOS

Generational support in ZGC



Why generations?

Workloads where the allocation rate exceeds the ability of the GC to keep up



Allocation Stalls

Garbage collection is a constant race between

- The java threads allocating objects
- The GC reclaiming dead objects

Allocation stalls happen when

- Heap filled up before the GC can reclaim memory of dead objects
- Java threads stand still waiting for memory



The Weak Generational Hypothesis

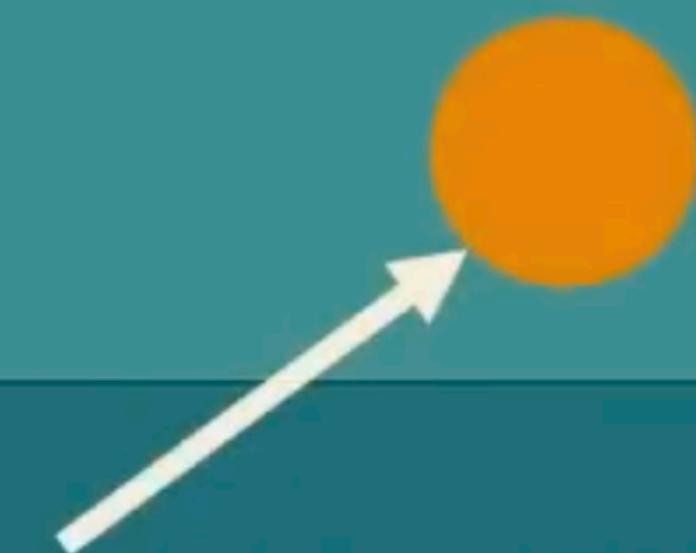
A very common pattern in Java applications is that most objects are short lived



Heap

Young Generation

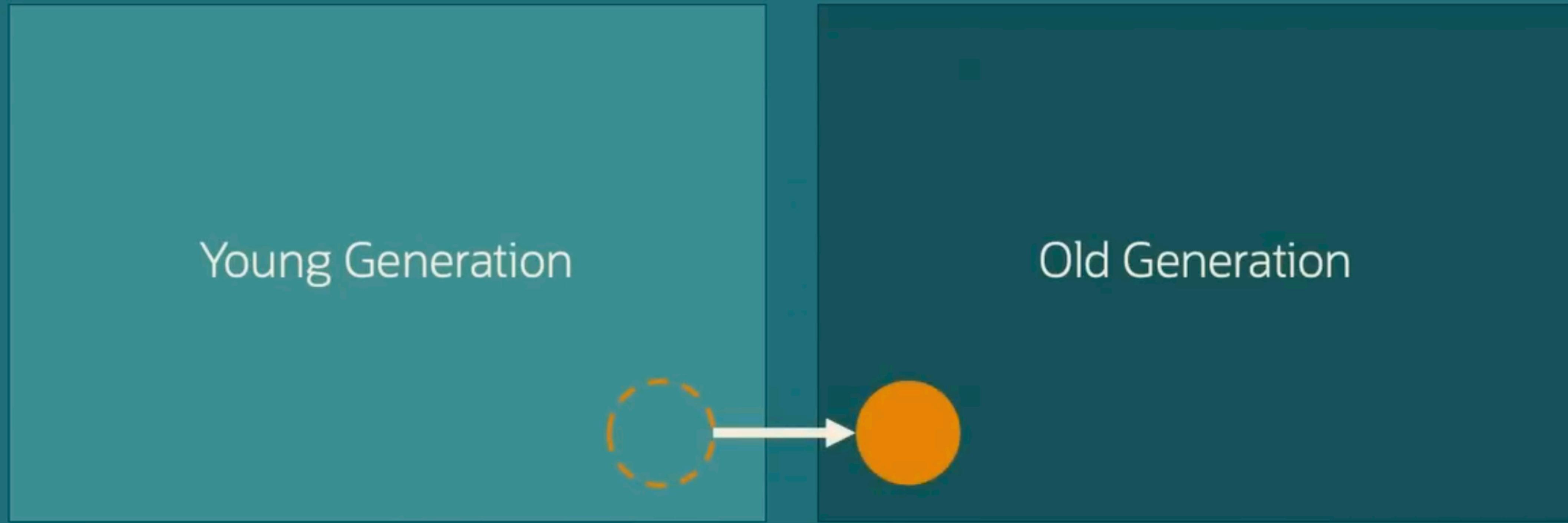
Old Generation



New object allocated in the young generation



Heap



Young Generation

Old Generation

Object promoted to the old generation

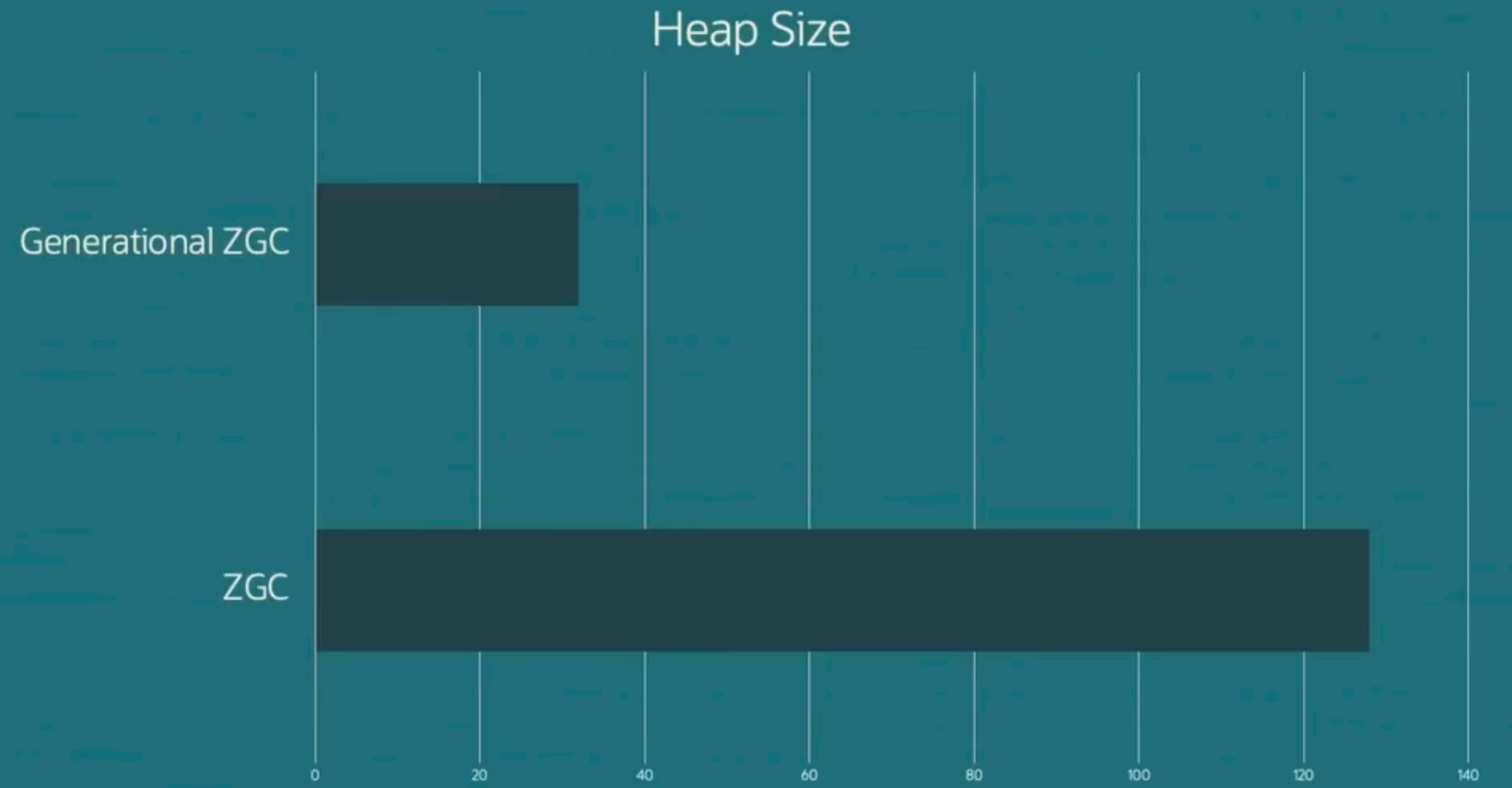


Reduced Effort to Collect Garbage

- Withstand higher allocation rates*
- Lower heap headroom*
- Lower CPU usage*

Cassandra 4 Memory Usage

With same injection rate



Cassandra 4 Throughput

With 128G heap



Throughput

Generational ZGC

ZGC

0 50 100 150 200 250





- 4x throughput
- 25% memory
- <1 ms GC Pauses



Performance tuning generational ZGC

Performance tuning generation

Efficiency tuning generation

Digitized by srujanika@gmail.com

CC BY-NC-ND 4.0 International license available at <https://creativecommons.org/licenses/by-nd/4.0/>

• [View Details](#) • [View Details](#) • [View Details](#) • [View Details](#) • [View Details](#)



Performance tuning generational ZGC

Automatic Tuning

- Dynamic Generation Sizing
 - (No `-Xmn` needed)
- Dynamic Tenuring Threshold
 - (No `-XX:TenuringThreshold` needed)
- Initiating Heap Occupancy
 - (No `-XX:InitiatingHeapOccupancyPercent` needed)
- Dynamic number of threads
 - (No `-XX:ConcGCThreads` needed)

Max heap size

- `-Xmx`

ZGC Wiki



<https://wiki.openjdk.org/display/zgc>

wiki.openjdk.org/display/zgc

The Z Garbage Collector, also known as ZGC, is a scalable low latency garbage collector designed to meet the following goals:

- Sub-millisecond max pause times
- Pause times do not increase with the heap, live-set or root-set size
- Handle heaps ranging from a 8MB to 16TB in size

ZGC was initially introduced as an experimental feature in JDK 11, and was declared Production Ready in JDK 15.

At a glance, ZGC is:

- Concurrent
- Region-based
- Compacting
- NUMA-aware
- Using colored pointers
- Using load barriers

At its core, ZGC is a concurrent garbage collector, meaning all heavy lifting work is done while Java threads continue to execute. This greatly limits the impact garbage collection will have on your application's response time.

This OpenJDK project is sponsored by the HotSpot Group.

Contents

- Supported Platforms
- Quick Start
- Configuration & Tuning
 - Overview
 - Enabling ZGC
 - Setting Heap Size
 - Setting Concurrent GC Threads
 - Returning Unused Memory to the Operating System
 - Enabling Large Pages On Linux
 - Enabling Transparent Huge Pages On Linux
 - Enabling NUMA Support
 - Enabling GC Logging
- Change Log
 - JDK 18
 - JDK 17
 - JDK 16
 - JDK 15
 - JDK 14
 - JDK 13
 - JDK 12
 - JDK 11
- FAQ
 - What does the "Z" in ZGC stand for?
 - Is it pronounced "zed gee see" or "zee gee see"?

Download

Latest Stable: JDK 19
Latest Development: JDK 20 Early Access

Source Code

Stable: github.com/openjdk/jdk
Development: github.com/openjdk/zgc/tree/zgc_generational

Blog Posts

ZGC | What's new in JDK 18
ZGC | What's new in JDK 17
ZGC | What's new in JDK 16
ZGC | What's new in JDK 15
ZGC | Using -XX:SoftMaxHeapSize
ZGC | What's new in JDK 14
Compact Forwarding Information
How do hot and cold objects behave?

Talks/Presentations/Podcasts

Oracle Developer Live 2022 - Slides | Video (28 min)
Jokerconf 2021 - Slides
Inside Java Podcast - ZGC - Sound (30 min)
Oracle Developer Live 2020 - Slides | Video (40 min)
Oracle Code One 2019 - Slides
PL-Meetup 2019 - Slides
JOkus 2019 - Slides | Video (21 min)
Devoxx 2018 - Slides | Video (40 min)
Oracle Code One 2018 - Slides | Video (45 min)
JOkus 2018 - Slides | Video (45 min)
FOSDEM 2018 - Slides

Mailing List

Subscribe | Archive

Project

Members
JIRA Dashboard
JEPs 333, 351, 364, 365, 376, 377

JEP 439: Generational ZGC



<https://openjdk.org/jeps/439>

- Motivations and goals behind generational ZGC
- Insight into the technical design features enabling generational ZGC
- Now in Candidate state

OpenJDK **JEP 439: Generational ZGC**

Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GAVBA Builds
Mailing lists
Wiki-IRC
Bylaws - Census
Legal
JEP Process
Source code
Mercurial
GitHub
Tools
Git
JITprof harness
Groups
(overview)
Adoption
Build
Client Libraries
Compatibility & Specification
Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support
Internationalization
jMX
Members
Networking
Partners
Quality
Security
Serviceability
Vulnerability
Web
Projects
(overview, archive)
Amber
Audio Engine

Owner Stefan Karlsson
Type Feature
Scope Implementation
Status Candidate
Component hotspot/gc
Discussion hotspot dash gc dash dev at openjdk dot org
Effort XL
Duration XL
Relates to JEP 377: ZGC: A Scalable Low-Latency Garbage Collector
(Production)
Reviewed by Erik Helin, Erik Österlund, Vladimir Kozlov
Created 2021/08/25 12:01
Updated 2023/03/14 19:36
Issue 8272979

Summary
Improve application performance by extending the Z Garbage Collector (ZGC) to maintain separate generations for young and old objects. This will allow ZGC to collect young objects — which tend to die young — more frequently.

Goals
Applications running with Generational ZGC should enjoy

- Lower risks of allocations stalls,
- Lower required heap memory overhead, and
- Lower garbage collection CPU overhead.

These benefits should come without significant throughput reduction compared to non-generational ZGC. The essential properties of non-generational ZGC should be preserved:

Generational ZGC Early Access Build



<https://jdk.java.net/genzgc/>

Please send feedback via e-mail to zgc-dev@openjdk.org

jdk.java.net

GA Releases
JDK 19
JMC 8

Early-Access Releases
JDK 21
JDK 20
Generational ZGC
JavaFX 20
JavaFX 21
Jextract
Loom
Metropolis
Panama
Valhalla

Reference Implementations
Java SE 20
Java SE 19
Java SE 18
Java SE 17
Java SE 16
Java SE 15
Java SE 14
Java SE 13
Java SE 12
Java SE 11
Java SE 10
Java SE 9
Java SE 8
Java SE 7

Project ZGC: Generational ZGC Early-Access Builds

These are early access binaries of Generational ZGC JDK-8272979

Warning: This build is based on an incomplete version of JDK 21.

Build 21-genzgc+5-33 (2023/3/9)

These early-access builds are provided under the GNU General Public License, version 2, with the Classpath Exception.

Linux/AArch64	tar.gz (sha256)	197637312 bytes
Linux/x64	tar.gz (sha256)	199295478
macOS/AArch64	tar.gz (sha256)	193580674
macOS/x64	tar.gz (sha256)	195948493
Windows/x64	zip (sha256)	197656230

Notes

- If you have difficulty downloading any of these files please contact download-help@openjdk.org.

Feedback

Please send feedback via e-mail to zgc-dev@openjdk.org. To send e-mail to this address you must first subscribe to the mailing list.

Inside Java



<https://inside.java>

- Aggregated blog articles and presentations

The screenshot shows the Inside Java website with a dark teal background featuring binary code patterns. The header includes the site's name, a coffee cup icon, and a subtitle: "News and views from members of the Java team at Oracle". Below the header are links for "Shows: Podcast | Newscast | JEP Café | Sip Of Java" and navigation links "dev.java | About | Jobs". A search bar is present above the post list. The main content area displays five blog posts:

- SuperWord (Auto-Vectorization) - An Introduction**
Emanuel Peter on March 20, 2023 HotSpot
- G1/Parallel/Serial GC improvements in JDK 20**
Thomas Schatzl on March 17, 2023 JDK 20 GC
- Write performant Java code with the Vector API - JEP Café 18** ▶
José Paumard on March 14, 2023 Panama Performance JDK 20
- Running JDK Tools within a JShell Session**
Christian Stein on March 13, 2023
- Data-Oriented Programming in Java** ▶
Gavin Bierman on March 9, 2023 Amber



PREGUNTAS?



MUCHAS GRACIAS POR SU
ASISTENCIA

+51 939 965 148