CLR/JVM implementation differences

Jean-Philippe BEMPEL

WebScale
@jpbempel

JVM/CLR implementation differences

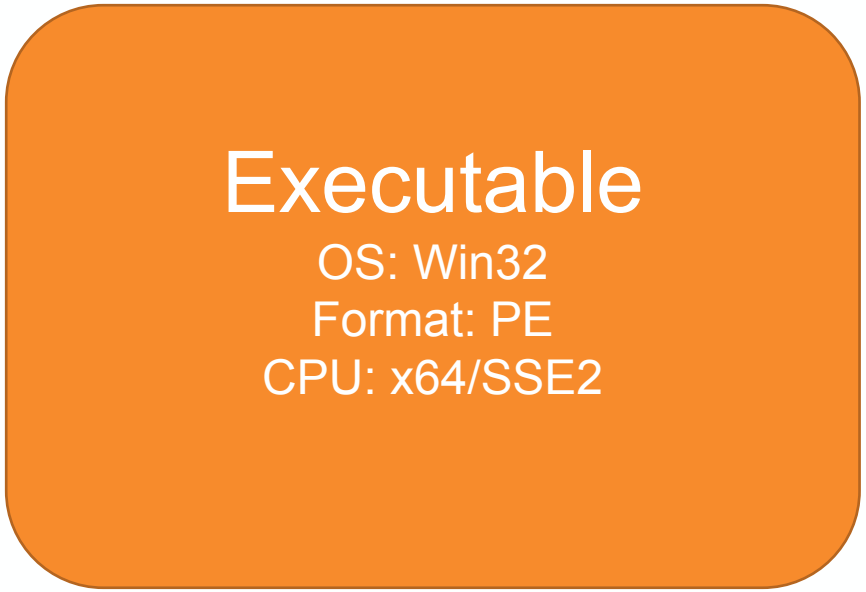
- Common Design Goals
- JIT
- GC
- Tooling

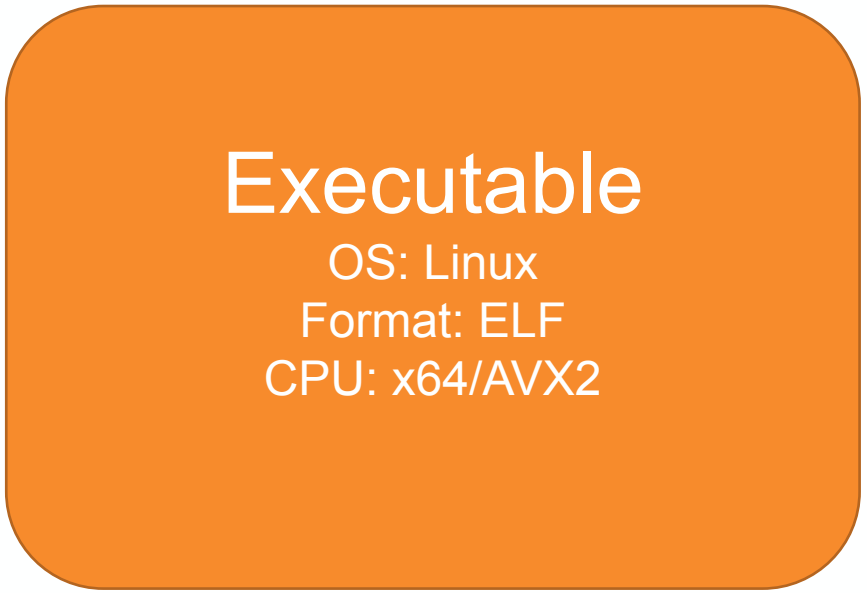
criteo

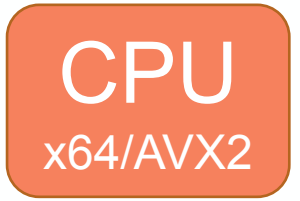
Common Design Goals

criteo

Native

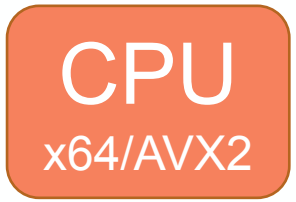






criteo

Runtime



criteo

JIT

criteo

JIT Pros & Cons

- Dynamic compilation opposed to AOT (Ahead Of Time)

+ In theory can produce code faster than static compiler (more information at runtime than statically, usage),

+ Global code optimization (instead of linking, inlining)

+ Cpu instruction detection

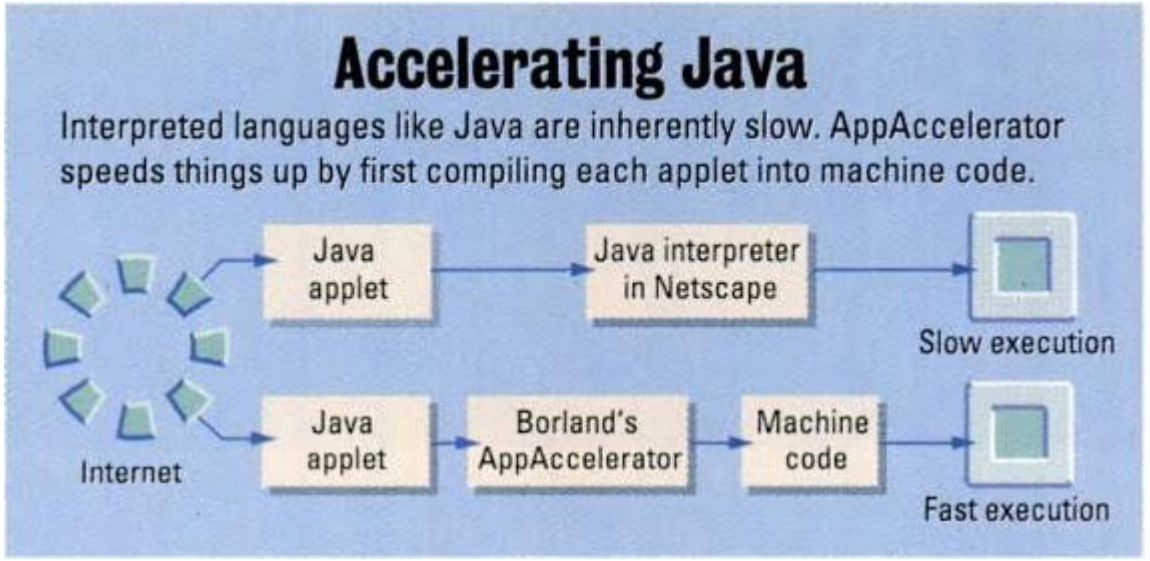
- Time constraints, startup/warmup time

- No shareable code across process

criteo

JVM JIT History

- No JIT at first, interpreter only
- Some companies worked on JIT (Borland, Symantec, IBM, BEA)
- First JIT in Sun JVM in 1999 with JDK 1.3 HotSpot



HotSpot JIT Characteristics

- Start compiling after a threshold
  Allow to compile only Hot methods => Hot spots !
- Profile guided optimizations
- Optimistic/Aggressive optimizations - Deoptimizations
- 2 flavors: Client (C1) – Server (C2)
- Since JDK 8: Tiered Compilation

C1

- Client Compiler
- Reduce warmup time, quick compilation
- Threshold set to 1500 calls before compilation
- Less optimizations, less aggressive or advanced

C2

- Server Compiler
- Peak performance
- Threshold set to 10 000 calls before compilation
- More optimizations, more aggressive and advanced

criteo

JVM Specific optimizations

- Start with interpreter and statistics gathering (PGO)

- Use stats to apply aggressive/optimistic optimizations:
  - Devirtualization (very important for Java)
  - Inlining (Mother-Of-All-Optimizations)
  - Exception handling elimination
  - Uncommon branch elimination
  - Null Check elimination

- OnStack Replacement

JVM JIT Pros & Cons

+ Optimistic/PGO optimizations

+ Interpreter allow any fallback in case of issue and does not prevent execution even on startup. Allow also longer compilation phases => more optimized

+ instrumentation (Changing bytecode on the fly)

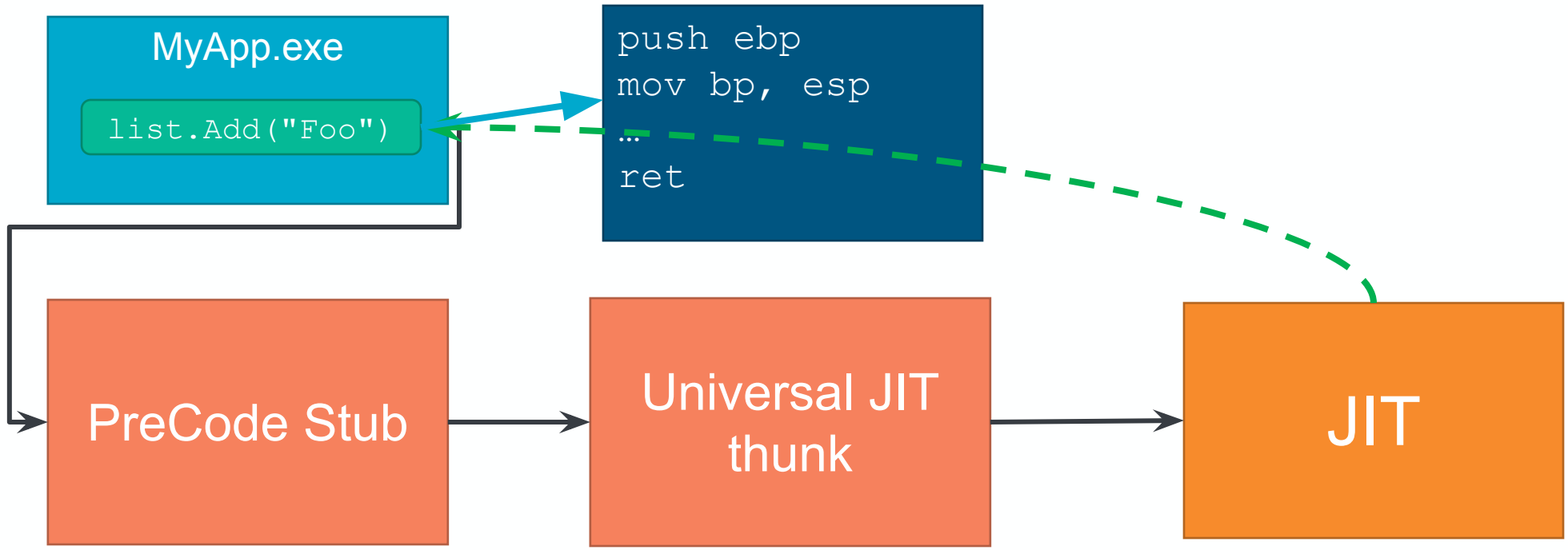
- Warmup can be long (10,000 calls before kicking JIT) (Mitigation: configurable, Tiered Compilation)

- Deoptimizations are STW

CLR JIT History

- JIT Released with 1.0 in 2002 but only 32bits
- JIT 64 bits for .NET 2.0, more optimized for server, slower to compile
- RyuJit x64 in .NET 4.6 (2015), Faster to compile with same level of optimizations

CLR JIT Machinery



criteo

CLR JIT Characteristics

- No interpreter, Execution is always with native code
- Fast compile time
- Compilation happening on the calling threads

CLR JIT Pros & Cons

+ After first call, method is always fast (warmup short) and stable
+ Fast compile time
+ Value type support

- Optimizations are not advanced and conservatives
- Interface call dispatch has relatively high overhead and not inlinable

GC

criteo

GC Pros & cons

+ Simplify memory management (including some Lock-free algorithms)
+ Fast allocation
+ Reduce Fragmentation
+ Can be cache friendly

- unpredictable Stop-The-World Pauses
- hard to tune

JVM GC Algorithms

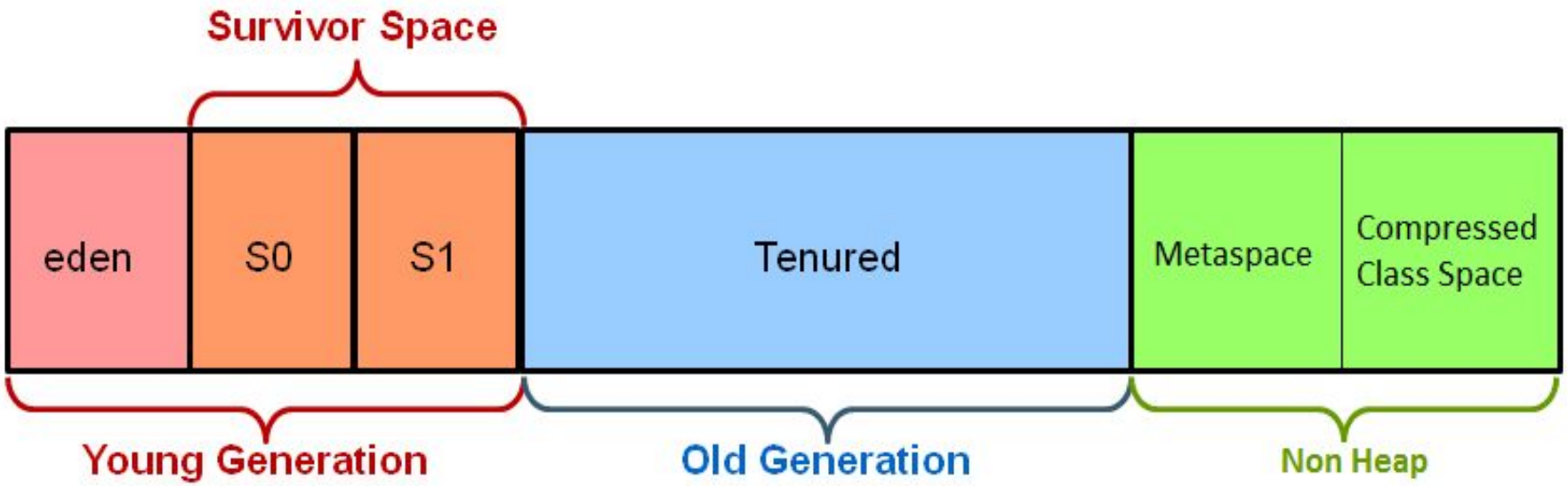
- Serial
- Parallel
- Concurrent Mark & Sweep
- G1

- Azul’s C4
- Shenandoah
- Z

JVM GC Characteristics

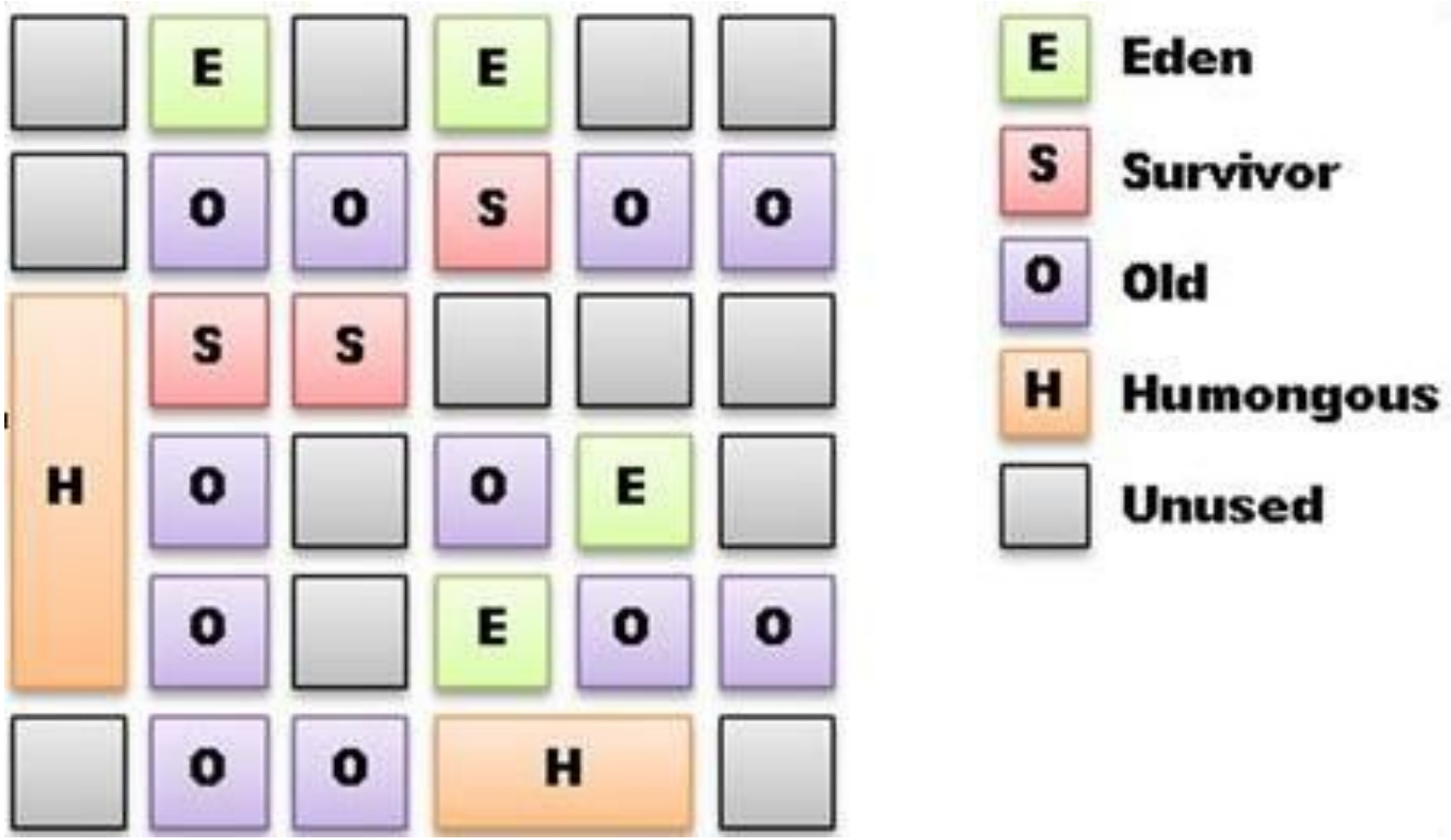
- Maximum heap need to be fixed
- Minor GC triggered by young gen being full
- Lot of options for tuning (too much?)

JVM GC Layout



criteo

JVM GC Layout



criteo

JVM GC Pros & Cons

+ Large choice of algorithms adapted to workload
+ Few or no fragmentation (often compacting)

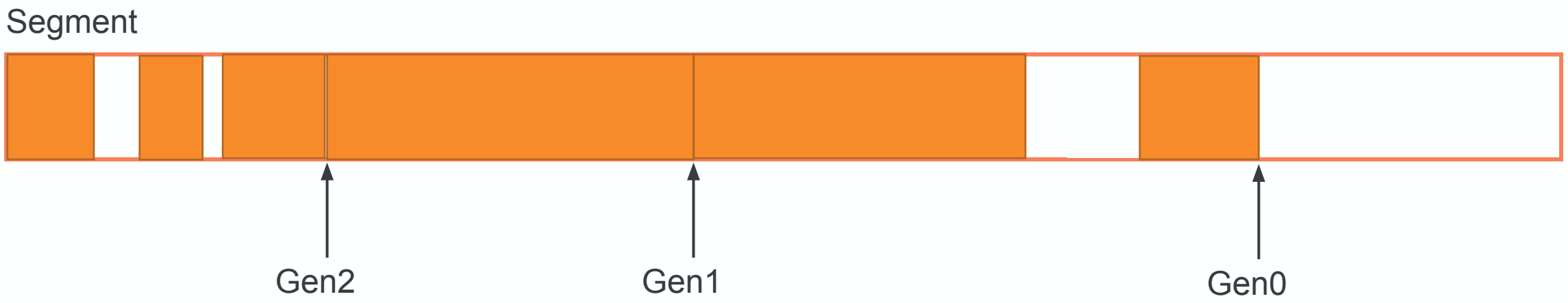
+/- Fine Control/Tuning

- Pause time can be large

CLR GC Algorithms

- Workstation
  - STW
  - Background
- Server
  - STW
  - Background

CLR GC Layout



criteo

CLR GC Characteristics

- Total heap sized depends on physical memory available
- GC triggered based on an estimated allocation budget per generation
- Leverage the non-fixed size of generation to adjust them
- On each GC a plan phase determines if a compacting phase is required
- Large Object Heap (>85,000B)
- No option, auto-tuning

CLR GC Pros & Cons

+ Pause time relatively short
+ Suits interactive apps mostly, with small/medium heap

+/- Almost No Control/Tuning

- Fragmentation/FreeLists
- Pinned Objects

Tooling

criteo

JVM troubleshooting

- Command line tools:
  - Thread Dump: jstack, jcmd
  - Heap Dump & Class Histogram: jmap, jcmd
  - GC: jstat, jcmd, GC logs
  - Config: jinfo, jcmd

- Monitoring/Profiling/Analysis tools:
  - Jconsole
  - VisualVM
  - Mission Control
  - MXBeans

CLR troubleshooting

- Command line tools:
  - Thread Dump: procdump
  - Heap Dump, Class Histogram: procdump
  - GC: PerfCounters/ETW events

- Monitoring/Profiling/Analysis tools:
  - WinDBG + SOS/SOSEX
  - Visual Studio
  - ETW events
  - PerfView

Conclusion

criteo

CLR/JVM

- Common Design goals
  - Runtime
  - Hardware abstraction
  - Memory safety

- JIT differences
  - JVM: Hybrid interpreter/compiled
  - CLR: First execution compilation

- GC differences
  - JVM: Compacting, highly tunable, many choices
  - CLR: auto-tunable, Free Lists, Pinned object

- Tooling
  - JVM: many tools, OS independent
  - CLR: raw tools, OS dependent

References

References JVM JIT

- Just-In-Time compilation Wikipedia
- Just-In-Time Compilers from Java 1.1 Unleashed
- PC Mag April 23 ’96
- Combining Analyses, Combining Optimizations - Cliff Click ‘s PhD thesis
- Combining Analyses, Combining Optimizations – Summary
- Design of the Java HotSpot Client Compiler for Java 6
- The Java HotSpot Server Compiler
- Null Check elimination
- Black Magic of (Java) Method Dispatch
- Virtual Call 911
- JIT Watch

References JVM GC

- Java Garbage Collection Distilled
- The Java GC mini book
- Oracle's white paper on JVM memory management & GC
- CMS phases
- G1 One Garbage Collector to rule them all
- Tips for Tuning The G1 GC
- G1 Garbage Collector Details and Tuning by Simone Bordet
- C4 Algorithm paper
- Shenandoah: The Garbage Collector That Could by Aleksey Shipilev
- ZGC - Low Latency GC for OpenJDK

References JVM tooling

- jmap
- jstack
- jcmd
- jinfo
- jstat
- Understanding GC logs
- Java Mission Control
- Using the Platform MBean Server and Platform MXBeans

References CLR JIT

- JIT Optimizations
- TieredCompilation in .NET Core
- TieredCompilation – design doc
- RyuJIT: The next-generation JIT compiler for .NET
- RyuJIT Overview
- Managed Profile-Guided Optimization Using Background JIT
- .NET Just in Time Compilation and Warming up Your System
- Performance decrease when using interfaces
- Virtual Stub Dispatch
- Simple devirtualization
- Collect Optimization Candidates for Using Intel Hardware Intrinsics in mscorlib
- sharplab.io

References CLR GC

- Maoni's blog
- Garbage Collection Design
- Learning How Garbage Collectors Work
- GC Pauses and Safe Points
- Back to basics: Generational Garbage Collector
- Garbage Collector Basics and Performance Hints
- Fun with .NET GC, Paging, Generations and Write Barriers

References CLR tooling

- Analysing .NET Memory Dumps with CLR MD
- Analysing Pause times in the .NET GC
- CLR MD Going beyond SOS
- Intro to WinDBG for .NET developers
- SOS.dll
- SOSEX
- CLR ETW Events

Thank You!

@jpbempel

criteo