

OpenJDK: In the New Age of Concurrent Garbage Collectors

HotSpot's Regionalized GCs



@mon_beck

Monica Beckwith

JVM Performance

java-performance@Microsoft



Agenda

Part 1 – Groundwork & Commonalities

Laying the Groundwork

Stop-the-world (STW) vs concurrent collection

Heap layout – regions and generations

Basic Commonalities

Copying collector – from and to spaces

Regions – occupied and free

Collection set and priority

Feb 5th, 2020

Agenda

Part 2 – Introduction & Differences

Introduction to G1, Shenandoah and Z GCs

Algorithm

Basic Differences

GC phases

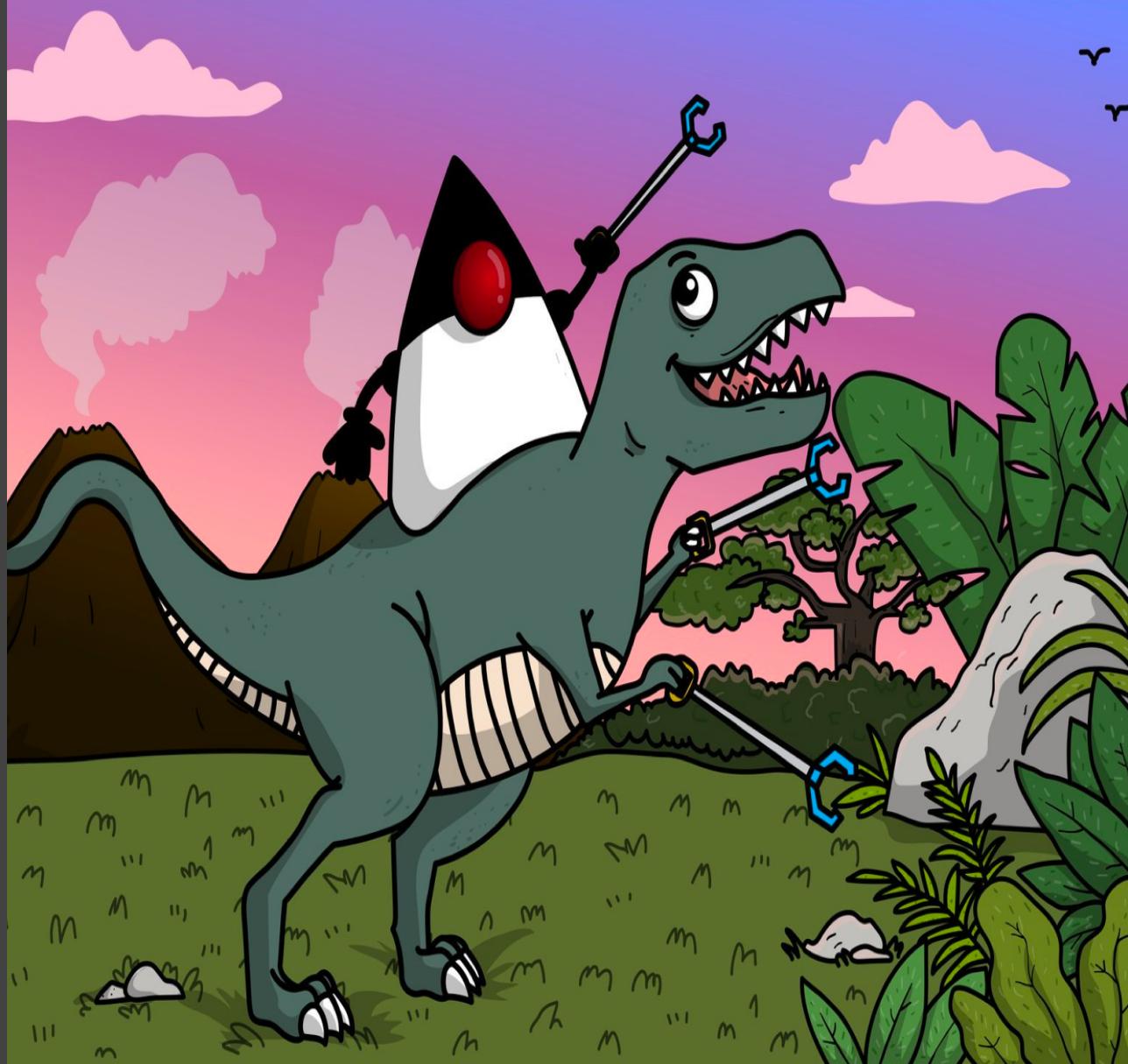
Marking

Barriers

Compaction

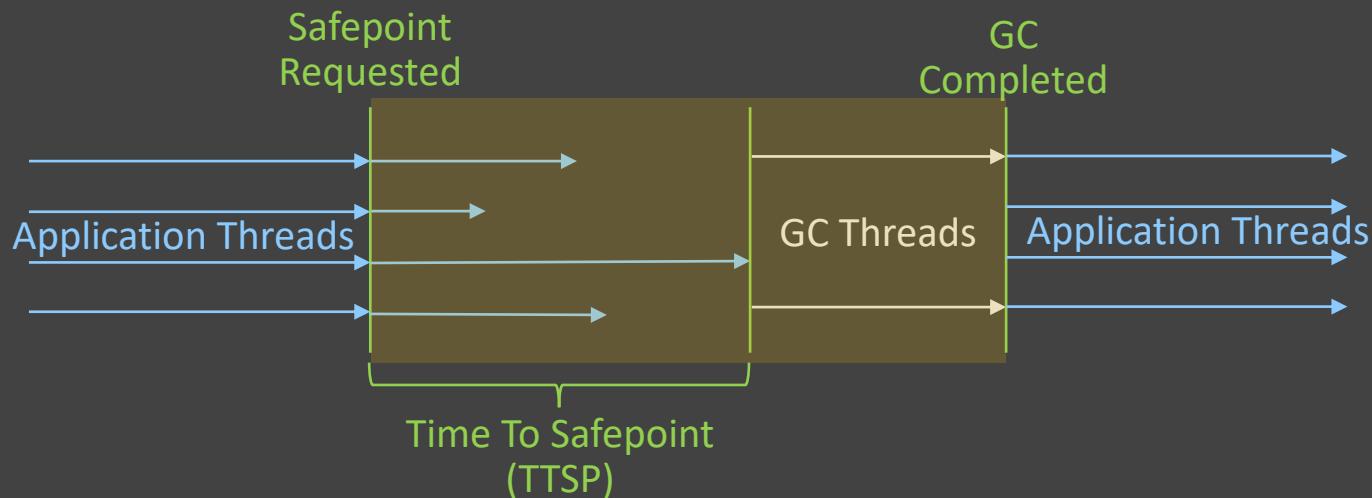
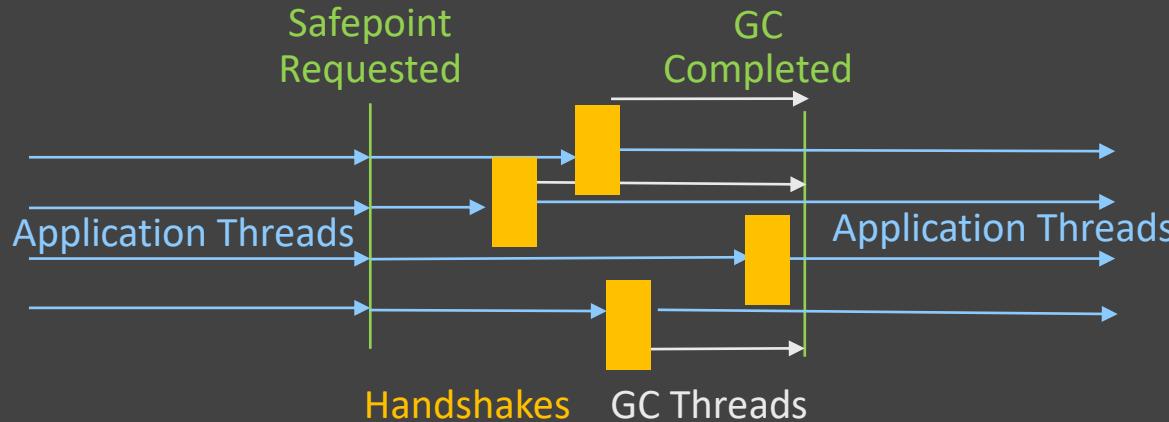
Feb 5th, 2020

Groundwork : Stop-the world vs concurrent collections

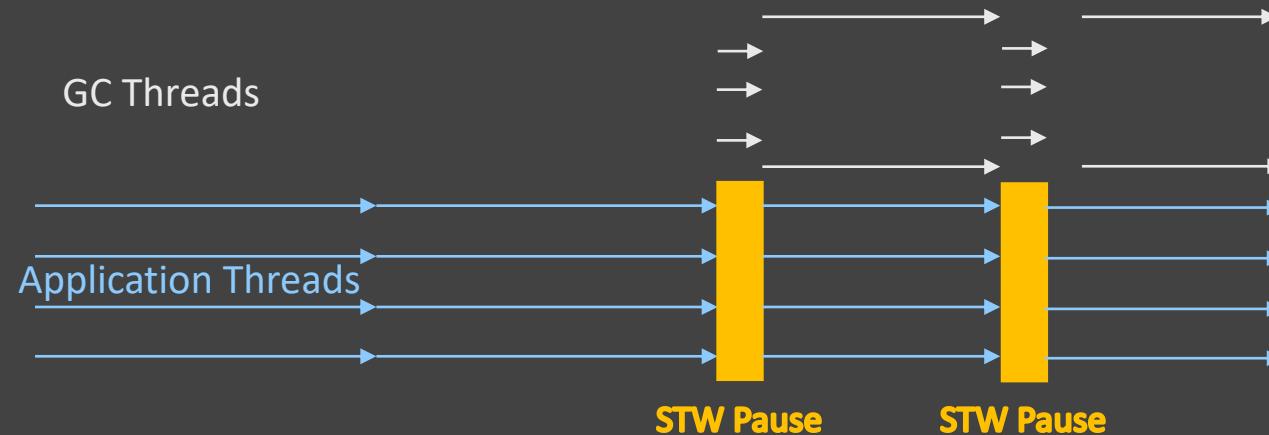


Stop-the-world aka STW GC

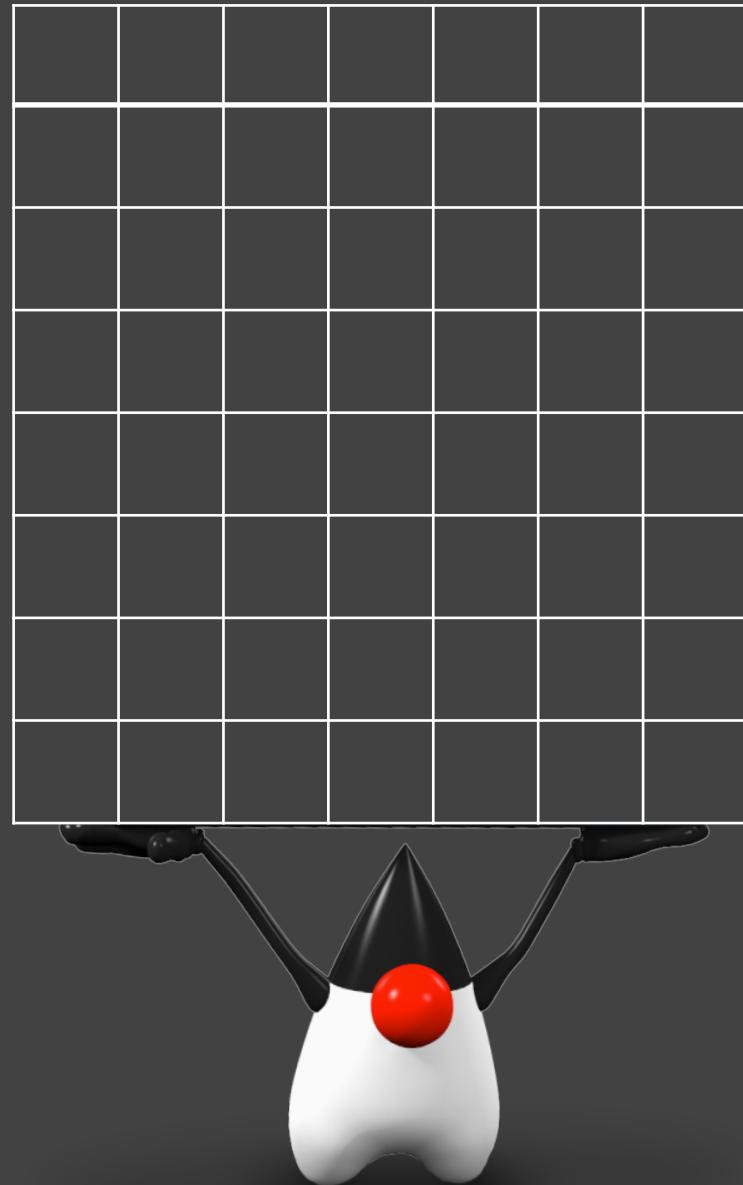
Thread local handshakes vs Global



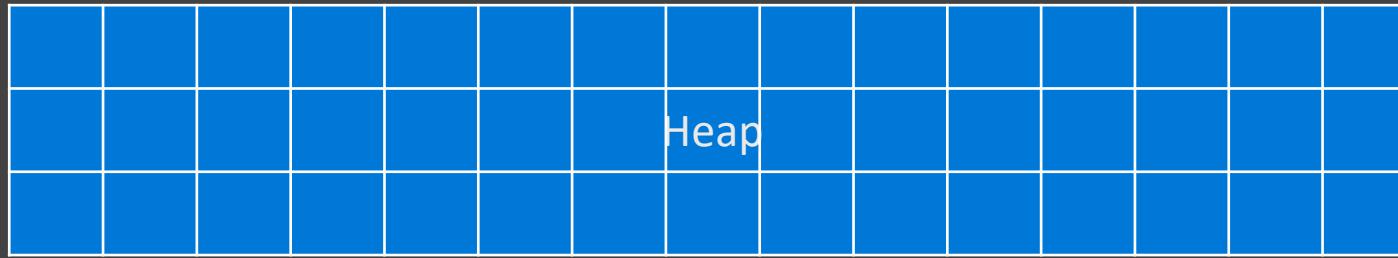
Concurrent GC



Groundwork : Heap layout - regions and generations



Heap Layout

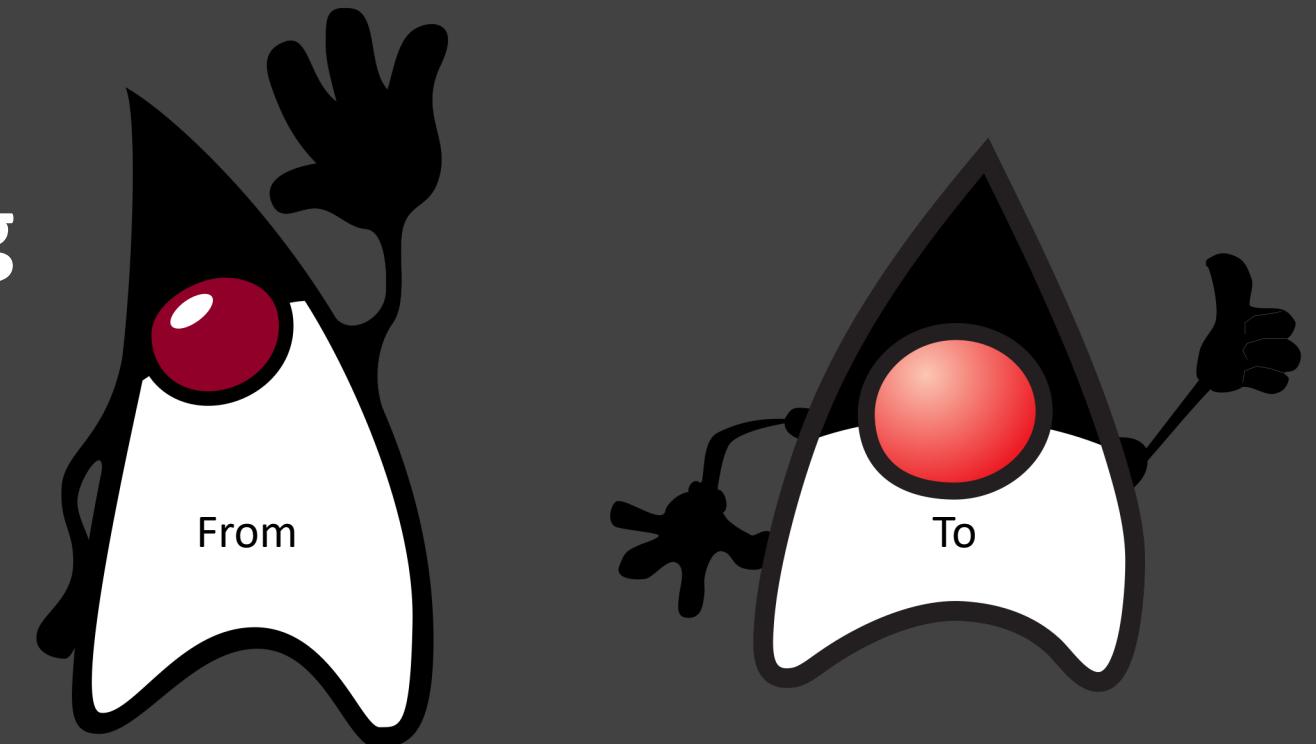


Z GC

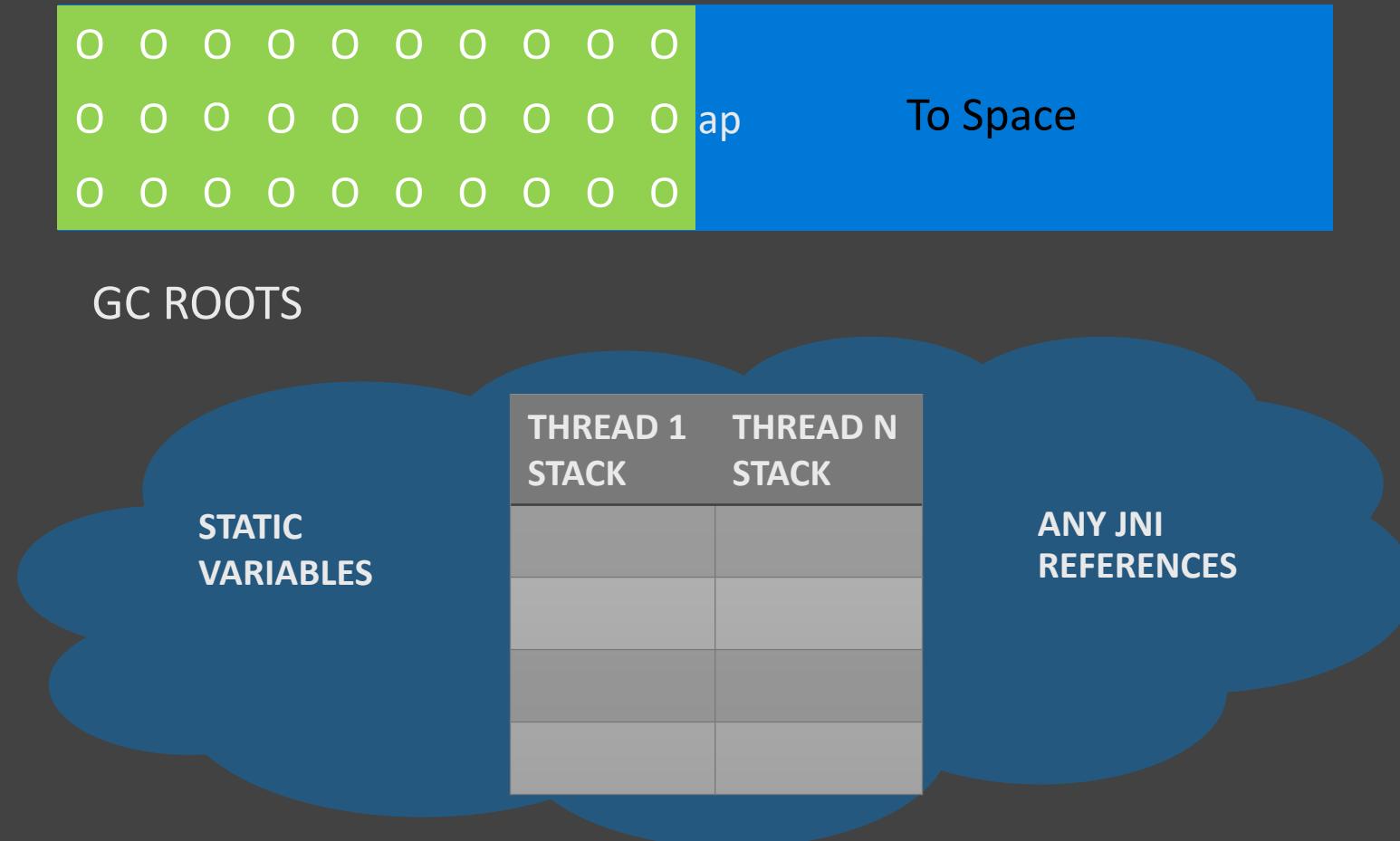


G1 GC

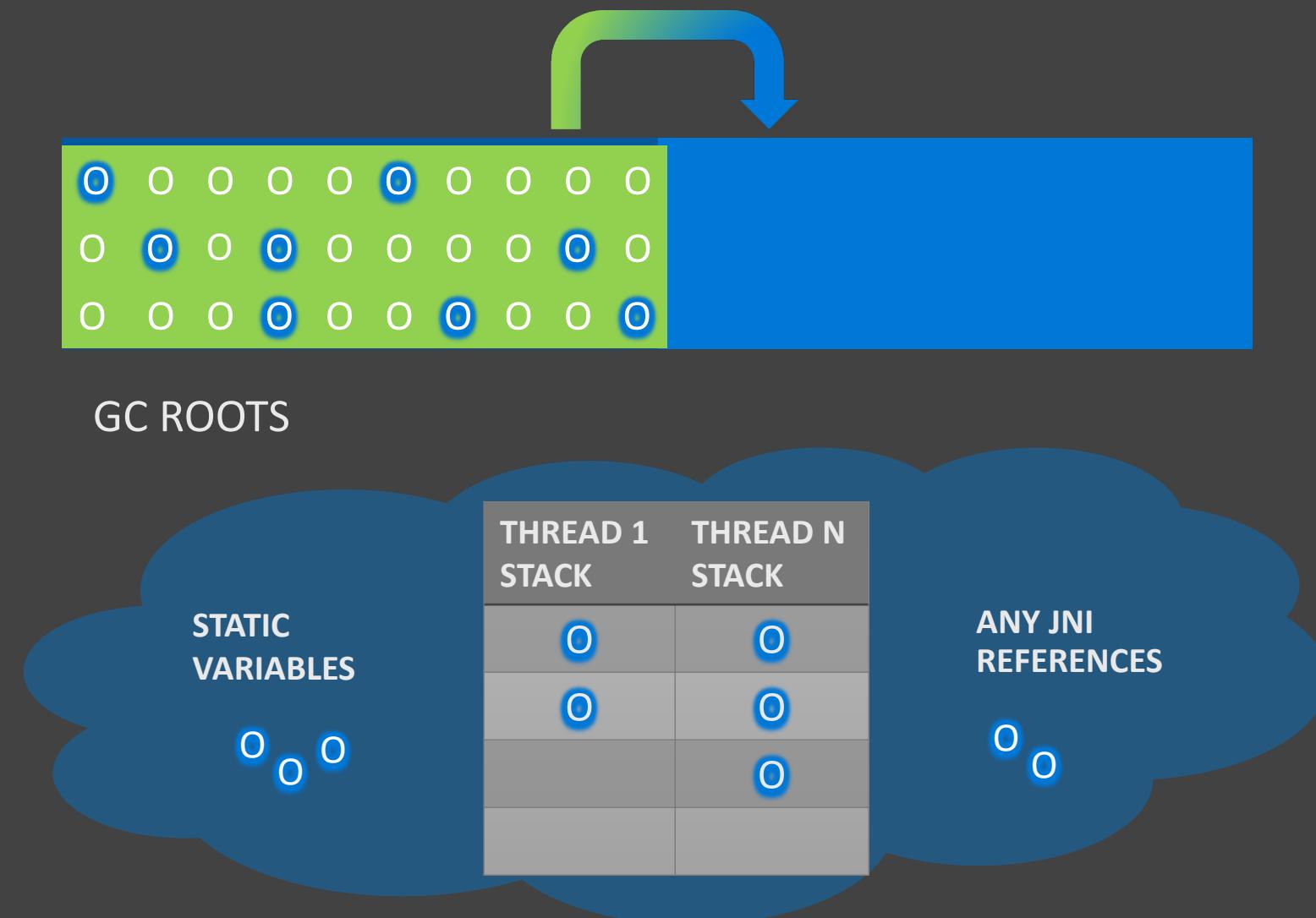
Commonalities : Copying collector – from and to spaces



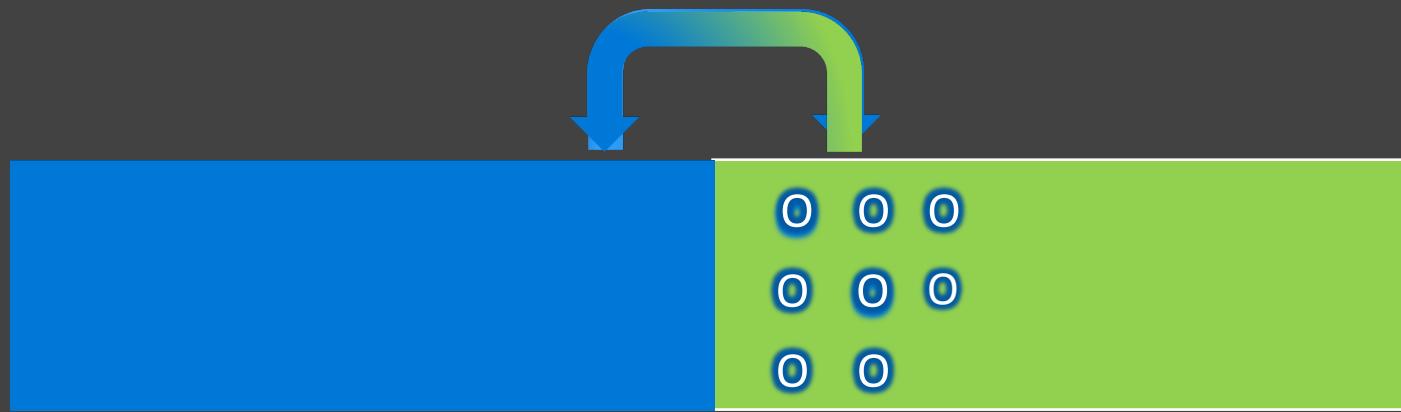
Copying Collector aka Compacting Collector aka Evacuation



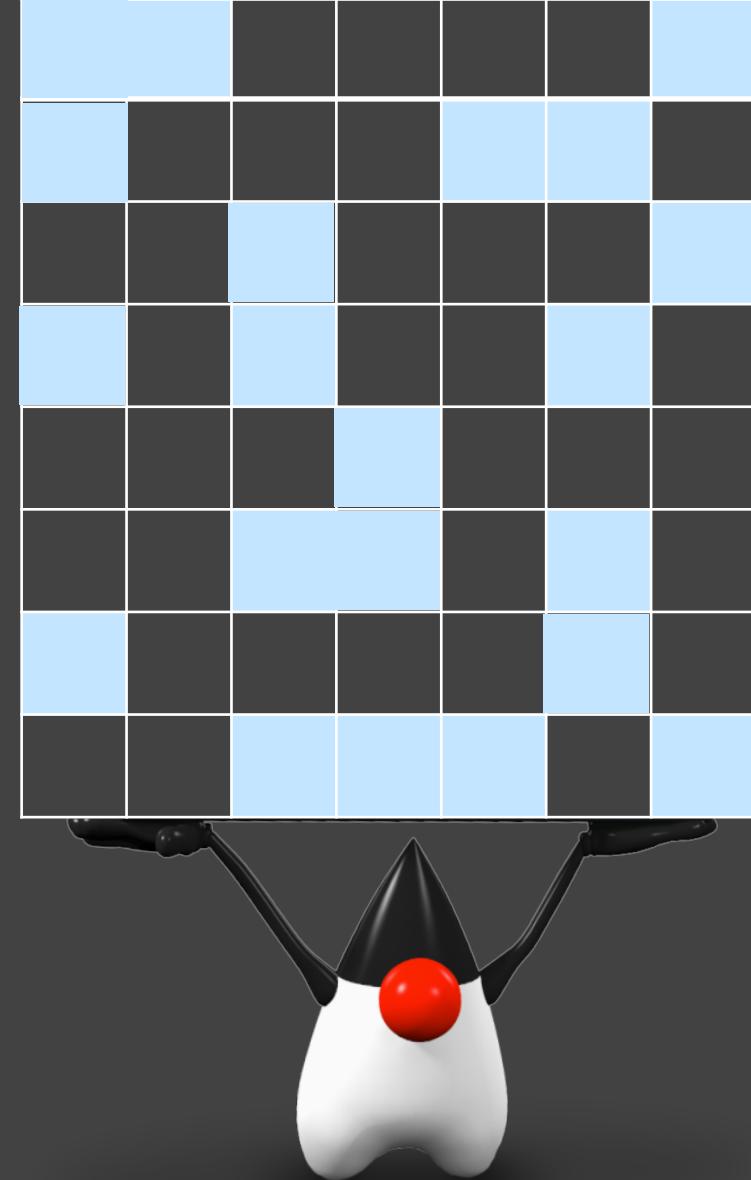
Copying Collector aka Compacting Collector aka Evacuation



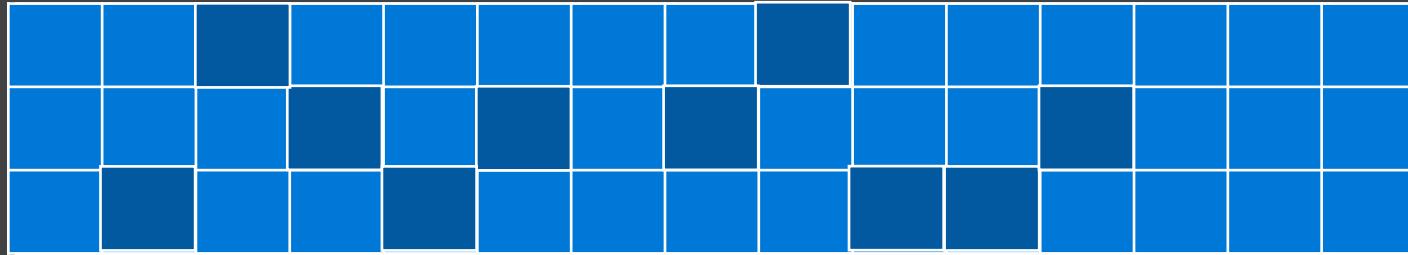
Copying Collector aka Compacting Collector aka Evacuation



**Commonalities : Regions
– occupied and free**

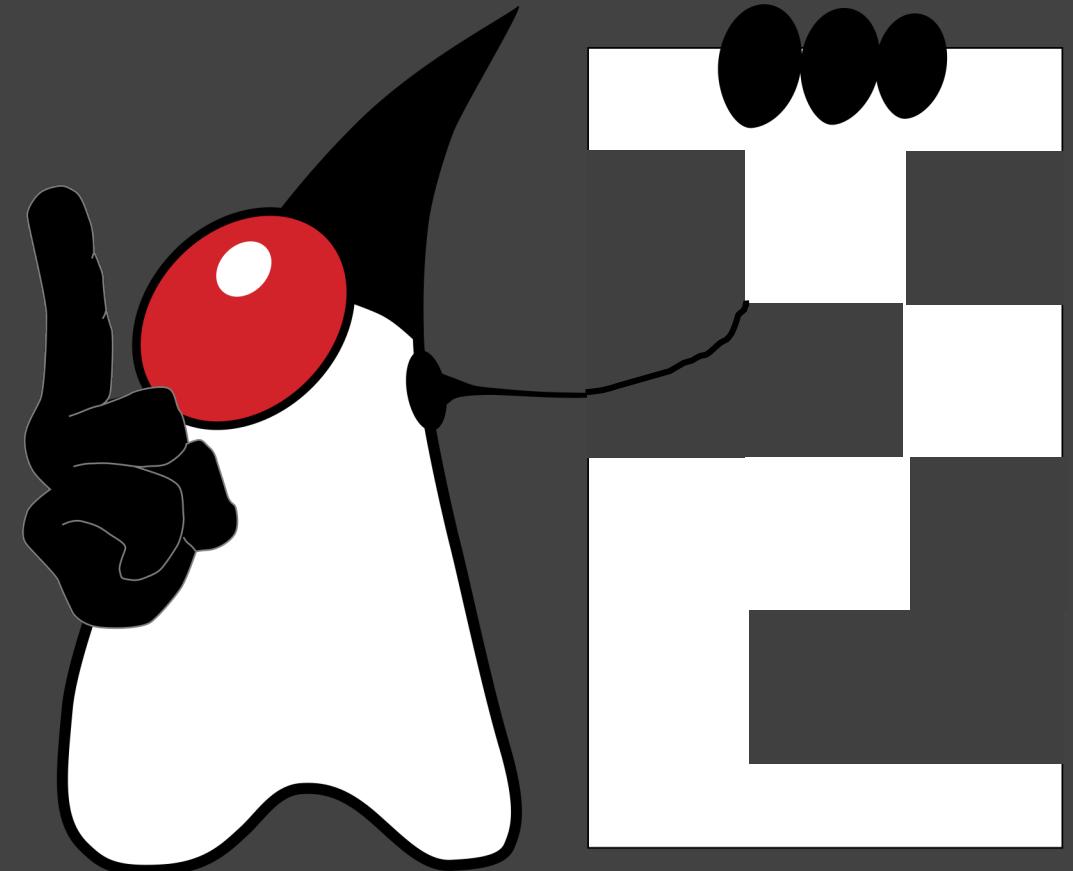


Occupied and Free Regions

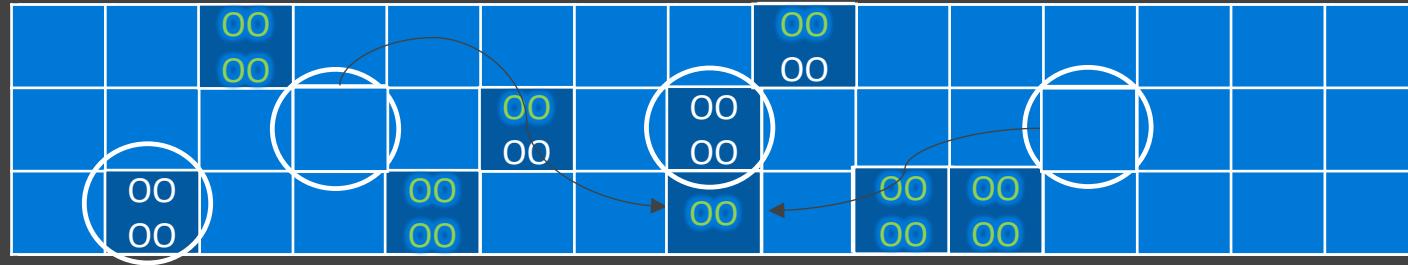


- List of free regions
- In case of generational heap (like G1), the occupied regions could be young, old or humongous

Commonalities : Collection set and priority

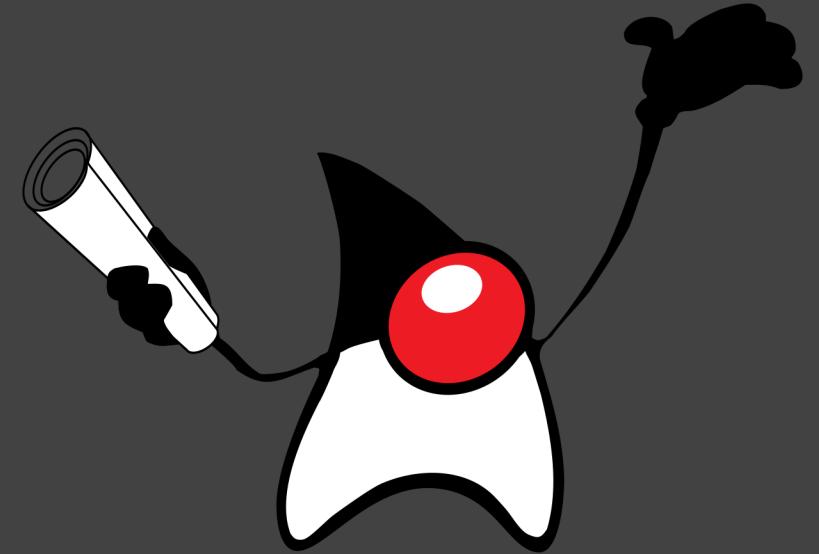


Collection Priority and Collection Set



- Priority is to reclaim regions with most garbage
- The candidate regions for collection/reclamation/relocation are said to be in a collection set
 - There are thresholds based on how expensive a region can get and maximum regions to collect
- Incremental collection aka incremental compaction or partial compaction
 - Usually needs a threshold that triggers the compaction
 - Stops after the desired reclamation threshold or free-ness threshold is reached
 - Doesn't need to be stop-the-world

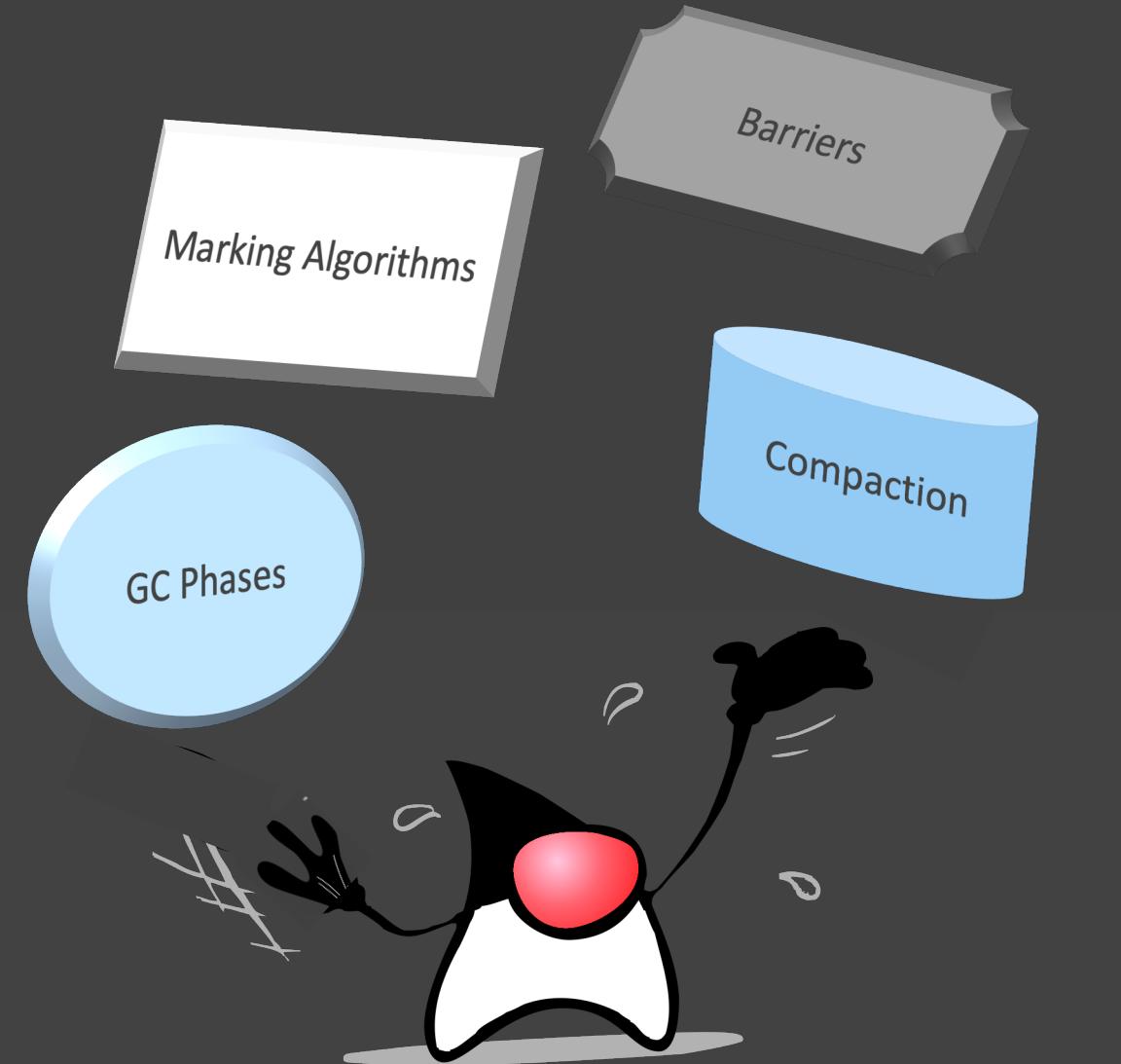
Introduction : G1, Shenandoah & Z - Algorithms



Algorithm and Other Considerations

Garbage Collectors
Regionalized?
Generational?
Compaction?
Target Pause Times?
Concurrent Marking Algorithm?

Differences – G1



GC Phases of Marking and Compaction

G1 GC	Gist
Initial Mark	Mark objects directly reachable by the roots
Concurrent Root Region Scanning	Since initial mark is piggy-backed on a young collection, the survivor regions need to be scanned
Concurrent Marking	Snapshot-at-the-beginning (SATB) algorithm
Final Marking	Drain SATB buffers; traverse unvisited live objects
Cleanup	Identify and free completely free regions, sort regions based on liveness and expense
STW Compaction	Move objects in collection set to “to” regions; free regions in collection set

- C. Hunt, M. Beckwith, P. Parhar, B. Rutisson. Java Performance Companion.

Concurrent Marking

Snapshot-at-the-beginning (SATB) Algorithm

Logical snapshot of the heap

SATB marking guarantees that all garbage objects that are present at the start of the concurrent marking phase will be identified by the snapshot

But, application mutates its object graph

- Any new objects are considered live

- For any reference update, the mutator needs to log the previous value in a log queue

- This is enabled by a pre-write barrier

•C. Hunt, M. Beckwith, P. Parhar, B. Ruttisson. Java Performance Companion.

•<https://www.jfokus.se/jfokus17/preso/Write-Barriers-in-Garbage-First-Garbage-Collector.pdf>

Barriers

SATB Pre-Write Barrier

The pseudo-code of the pre-write barrier for an assignment of the form $x.f := y$ is:

```
if (marking_is_active) {  
    pre_val := x.f;  
    if (pre_val != NULL) {  
        satb_enqueue(pre_val);  
    }  
}
```

- C. Hunt, M. Beckwith, P. Parhar, B. Rutisson. Java Performance Companion.

Barriers

Post Write Barrier

Consider the following assignment:

```
object.field = some_other_object
```

G1 GC will issue a write barrier after the reference is updated, hence the name.

G1 GC filters the need for a barrier by way of a simple check as explained below:

```
(&object.field XOR &some_other_object) >> RegionSize
```

If the check evaluates to zero, a barrier is not needed.

If the check != zero, G1 GC enqueues the card in the update log buffer

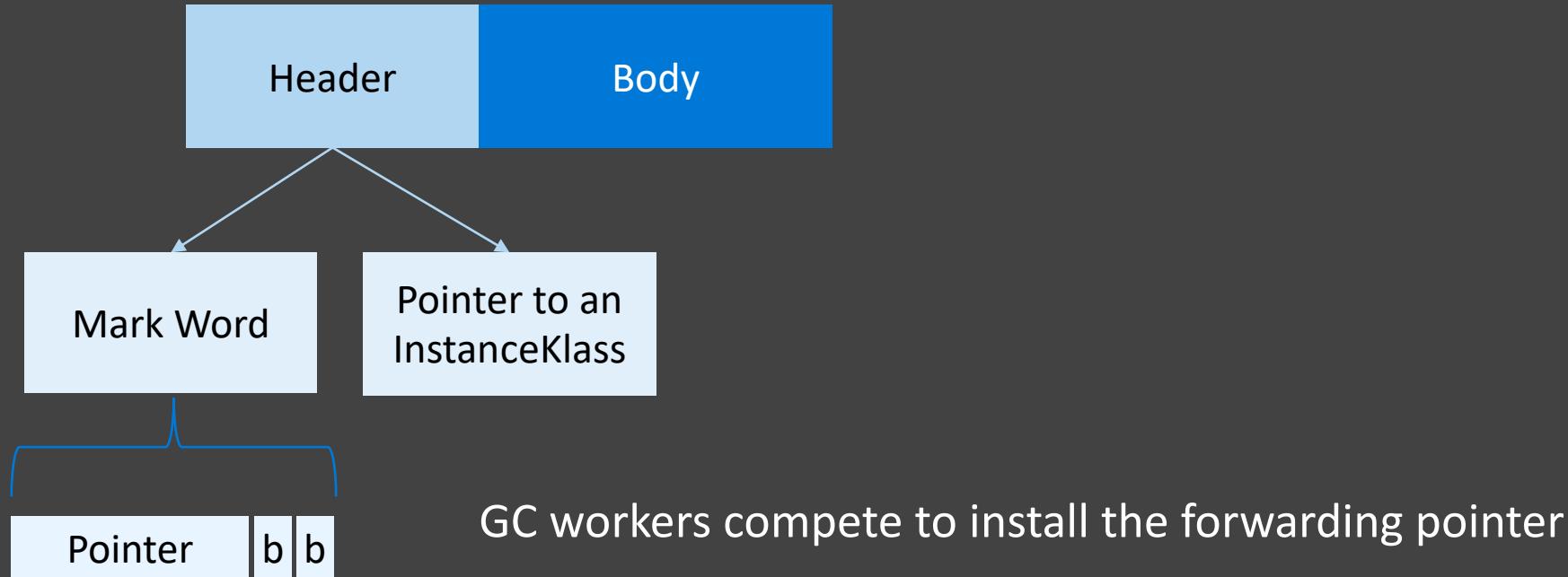
<https://www.jfokus.se/jfokus17/preso/Write-Barriers-in-Garbage-First-Garbage-Collector.pdf>

•C. Hunt, M. Beckwith, P. Parhar, B. Rutisson. Java Performance Companion.

STW Compaction

Forwarding Pointer in Header

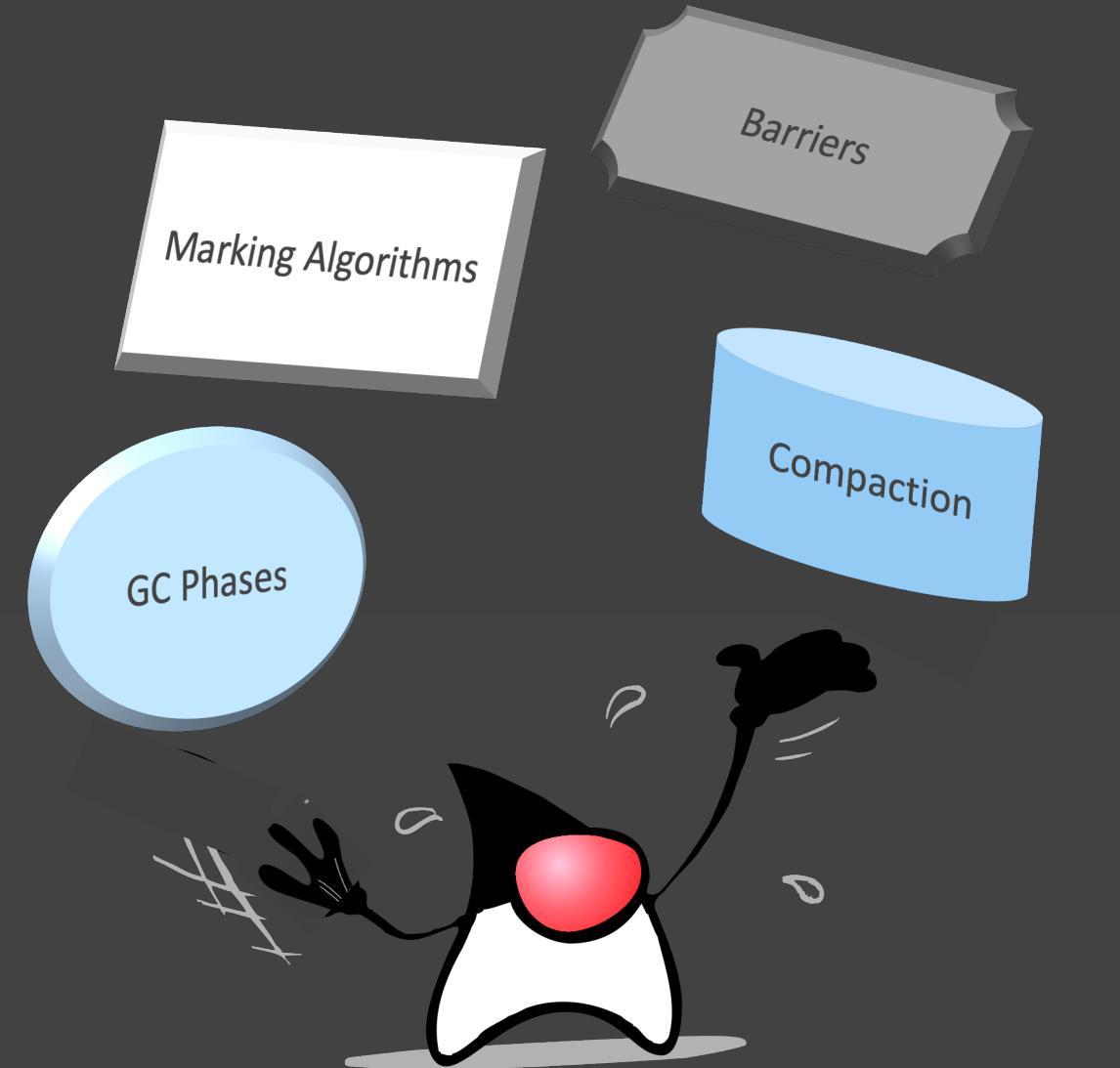
A Java Object



From source:

- An InstanceKlass is the VM level representation of a Java class. It contains all information needed for at class at execution runtime.
- When marked the bits will be 11

Differences – Z



GC Phases of Marking and Compaction

Z GC	Gist
Initial Mark	Mark objects directly reachable by the roots
Concurrent Marking	Striping - GC threads walk the object graph and mark
Final Marking	Traverse unvisited live objects; weak root cleaning
<i>Concurrent Prepare for Compaction</i>	Identify collection set; reference processing
<i>Start Compaction</i>	<i>Handles roots into the collection set</i>
<i>Concurrent Compaction</i>	Move objects in collection set to “to” regions
Concurrent Remap (done with Concurrent Marking of next cycle since walks the object graph)	Fixup of all the pointers to now-moved objects

<http://cr.openjdk.java.net/~pliden/slides/ZGC-Jfokus-2018.pdf>

Concurrent Marking

Colored Pointers

Heap divided into logical stripes

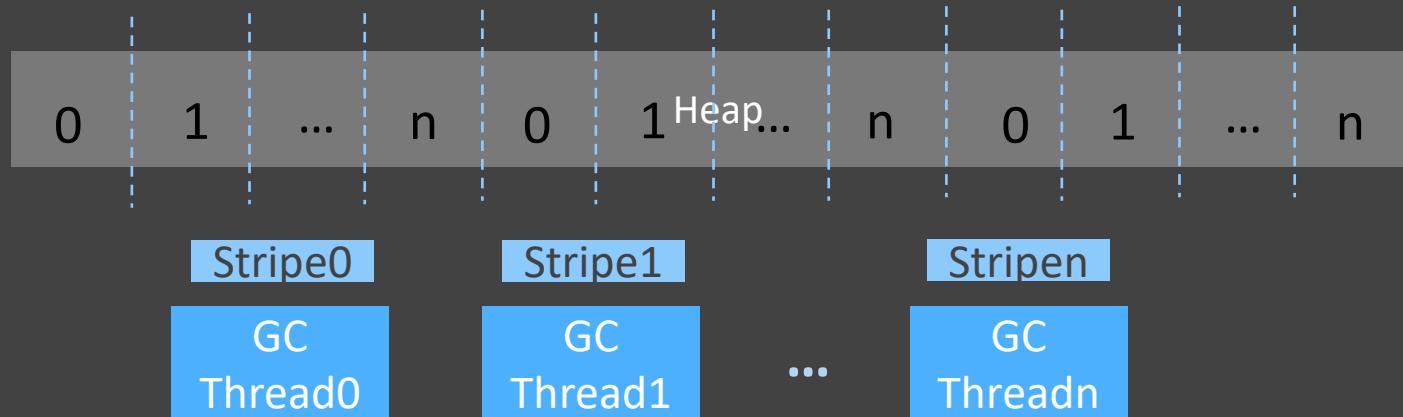
GC threads work on their own stripe

Minimizes shared state

Load barrier to detect loads of non-marked object pointers

Concurrent reference processing

Thread local handshakes



Barriers

Loaded Reference Barrier

Update a “bad” reference to a “good” reference

Can be self-healing/repairing barrier when updates the source memory location

Imposes a set of invariants –

“All visible loaded reference values will be safely “marked through” by the collector, if they haven’t been already.

All visible loaded reference values point to the current location of the safely accessible contents of the target objects they refer to.”

Tene, G.; Iyengar, B. & Wolf, M. (2011), C4: The Continuously Concurrent Compacting Collector, *in 'Proceedings of the international symposium on Memory management'* , ACM, New York, NY, USA , pp. 79--88 .

Example

```
Object o = obj.fieldA;           // Loading an object reference from heap  
  
load_barrier(register_for(o), address_of(obj.fieldA));  
  
if (o & bad_bit_mask) {  
    slow_path(register_for(o),  
    address_of(obj.fieldA)); }
```

Example

```
mov 0x20(%rax), %rbx // Object o = obj.fieldA;  
test %rbx, (0x16)%r15 // Bad color?  
jnz slow_path          // Yes -> Enter slow path and mark/relocate/remap,  
                         // adjust 0x20(%rax) and %rbx
```

<http://cr.openjdk.java.net/~pliden/slides/ZGC-Jfokus-2018.pdf>

Core Concept

Colored Pointers

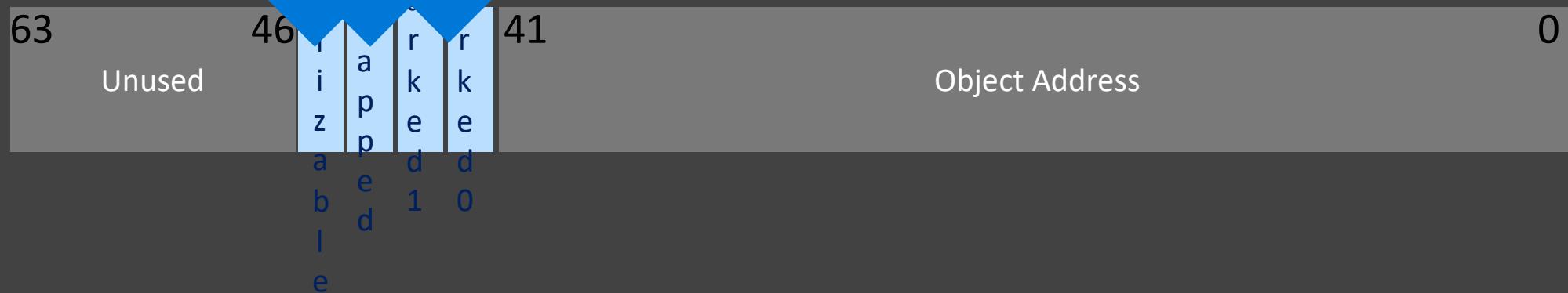
Metadata stores in the unused bits of the 64 bit pointers

Virtual address mapping/tagging

Multi-mapping on x86-64

Hardware support on SPARC, aarch64

Object is reachable
only through a
Finalizer?
?



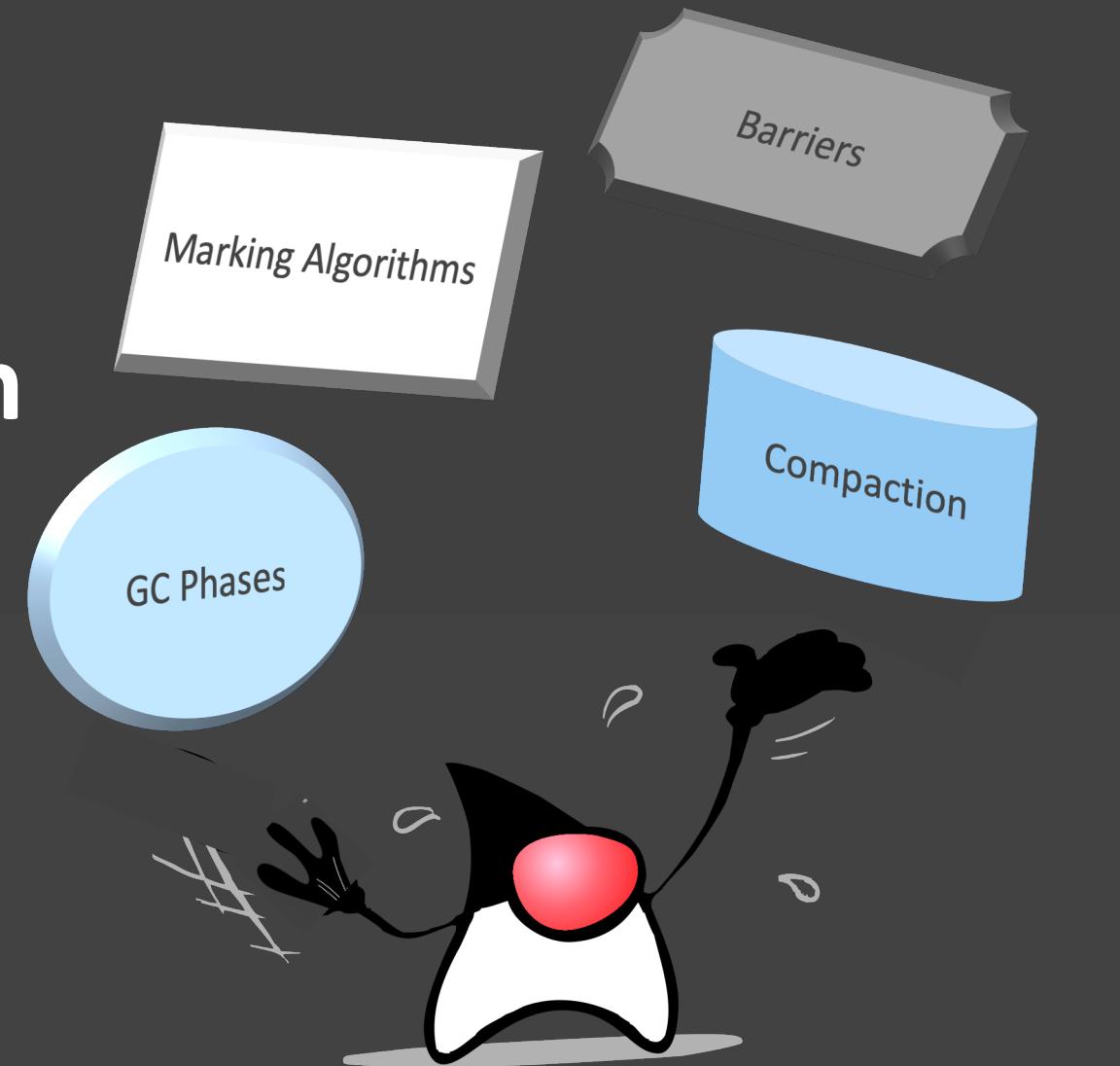
Concurrent Compaction

Load barrier to detect object pointers into the collection set

Can be self-healing

Off-heap forwarding tables enable to immediately release and reuse virtual and physical memory

Differences – Shenandoah



GC Phases of Marking and Compaction

Shenandoah GC	Gist
Initial Mark	Mark objects directly reachable by the roots
Concurrent Marking	Snapshot-at-the-beginning (SATB) algorithm
Final Marking	Drain SATB buffers; traverse unvisited live objects; identify collection set
<i>Concurrent Cleanup</i>	Free completely free regions
<i>Concurrent Compaction</i>	Move objects in collection set to “to” regions
Initial Update Reference	Initialize the update reference phase
Concurrent Update Reference	Scans the heap linearly; update any references to objects that have moved
Final Update Reference	Update roots to point to to-region copies
Concurrent Cleanup	Free regions in collection set

<https://wiki.openjdk.java.net/display/shenandoah/Main>

Concurrent Marking

Snapshot-at-the-beginning (SATB) Algorithm

- C. Hunt, M. Beckwith, P. Parhar, B. Ruttisson. Java Performance Companion.
- <https://www.jfokus.se/jfokus17/preso/Write-Barriers-in-Garbage-First-Garbage-Collector.pdf>

Barriers

SATB Pre-Write Barrier - Recap

Needed for all updates

Check if “marking-is-active”

SATB_enqueue the pre_val

- C. Hunt, M. Beckwith, P. Parhar, B. Rutisson. Java Performance Companion.

Barriers

Read Barrier – For Concurrent Compaction

Here's an assembly code snippet for reading a field:

```
mov 0x10(%rsi),%rsi ; *getfield value
```

Here's what the snippet looks like with Shenandoah:

```
mov -0x8(%rsi),%rsi ; read of forwarding pointer at address object - 0x8
mov 0x10(%rsi),%rsi ; *getfield value
```

*Flood, Christine & Kennke, Roman & Dinn, Andrew & Haley, Andrew & Westrelin, Roland. (2016). Shenandoah: An open-source concurrent compacting garbage collector for OpenJDK. 1-9. 10.1145/2972206.2972210.

Barriers

Copying Write Barrier – For Concurrent Compaction

Needed for all updates to ensure to-space invariant

Check if “evacuation_in_progress”

Check if “in_collection_set” and “not_yet_copied”

CAS (fwd_ptr(obj), obj, copy)

*Flood, Christine & Kennke, Roman & Dinn, Andrew & Haley, Andrew & Westrelin, Roland. (2016). Shenandoah: An open-source concurrent compacting garbage collector for OpenJDK. 1-9. 10.1145/2972206.2972210.

Barriers

Read Barrier – For Concurrent Compaction

Here's an assembly code snippet for reading a field:

```
mov 0x10(%rsi),%rsi ; *getfield value
```

Here's what the snippet looks like with Shenandoah:

```
mov -0x8(%rsi),%rsi ; read of forwarding pointer at address object - 0x8  
mov 0x10(%rsi),%rsi ; *getfield value
```

*Flood, Christine & Kennke, Roman & Dinn, Andrew & Haley, Andrew & Westrelin, Roland. (2016). Shenandoah: An open-source concurrent compacting garbage collector for OpenJDK. 1-9. 10.1145/2972206.2972210.

Barriers

Copying Write Barrier – For Concurrent Compaction

Needed for all updates to ensure to-space invariant

Check if “evacuation_in_progress”

Check if “in_collection_set” and “not_yet_copied”

CAS (fwd_ptr(obj), obj, copy)

Superseded

*Flood, Christine & Kennke, Roman & Dinn, Andrew & Haley, Andrew & Westrelin, Roland. (2016). Shenandoah: An open-source concurrent compacting garbage collector for OpenJDK. 1-9. 10.1145/2972206.2972210.

Barriers

Loaded Reference Barrier - Recap

Ensure strong ‘to-space invariant’

Utilize barriers at reference load

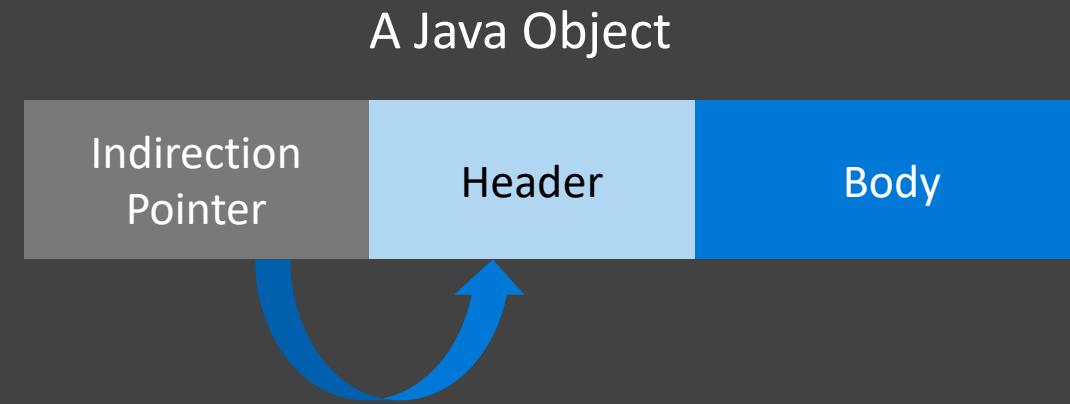
Check if fast-path-possible; else do-slow-path

<https://developers.redhat.com/blog/2019/06/27/shenandoah-gc-in-jdk-13-part-1-load-reference-barriers/>

Tene, G.; Iyengar, B. & Wolf, M. (2011), C4: The Continuously Concurrent Compacting Collector, *in* 'Proceedings of the international symposium on Memory management' , ACM, New York, NY, USA , pp. 79--88 .

Concurrent Compaction

Brooks Style Indirection Pointer



Forwarding pointer is placed before the object
Additional work of dereferencing per object

Concurrent Compaction

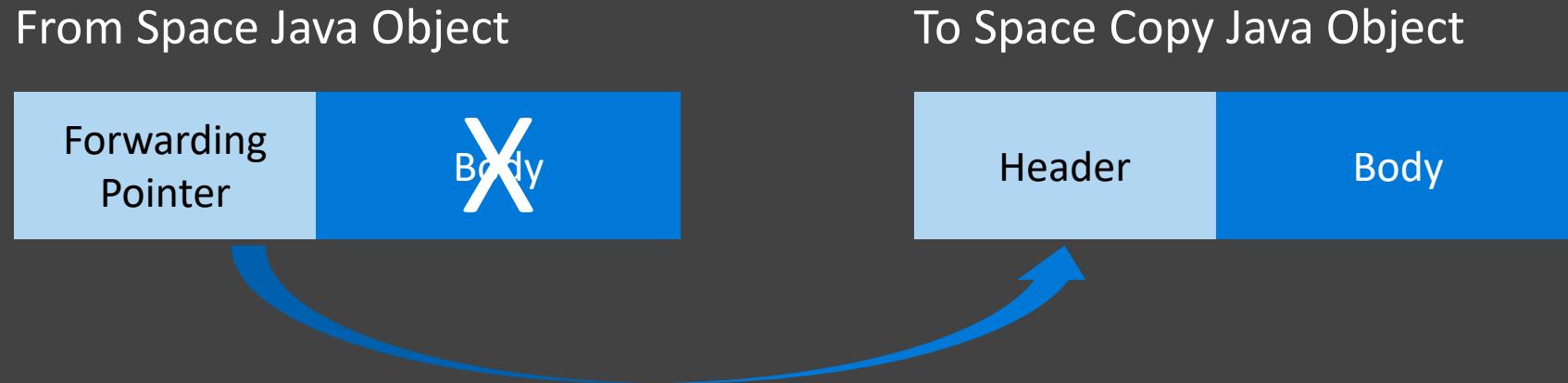
Brooks Style Indirection Pointer



Forwarding pointer is placed before the object
Additional work of dereferencing per object

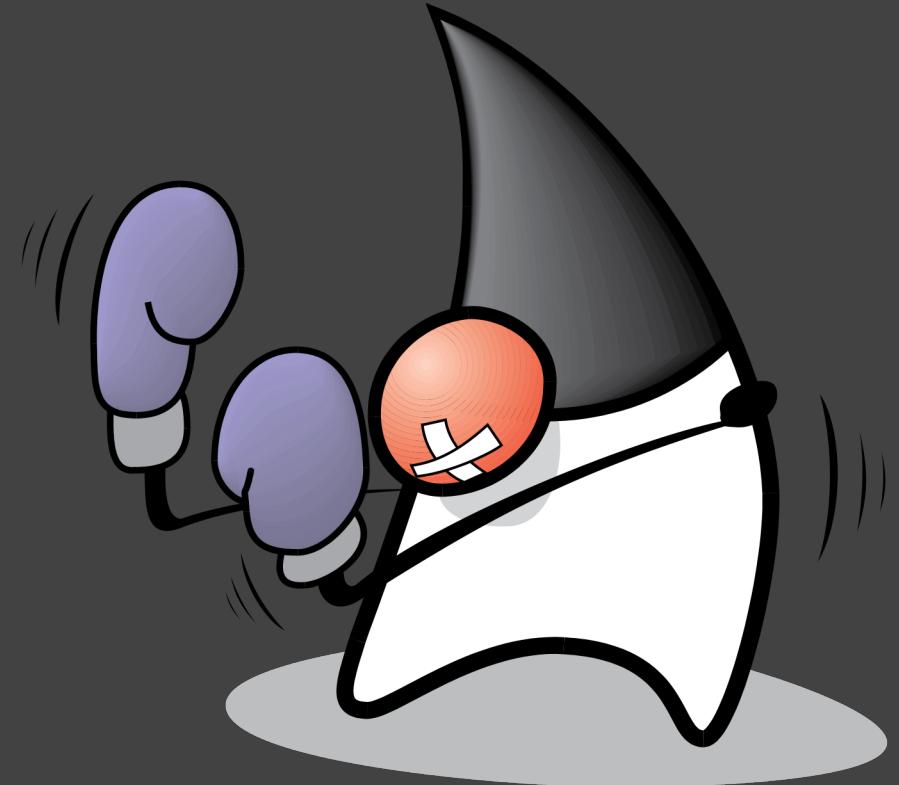
Concurrent Compaction

Forwarding Pointer in Header

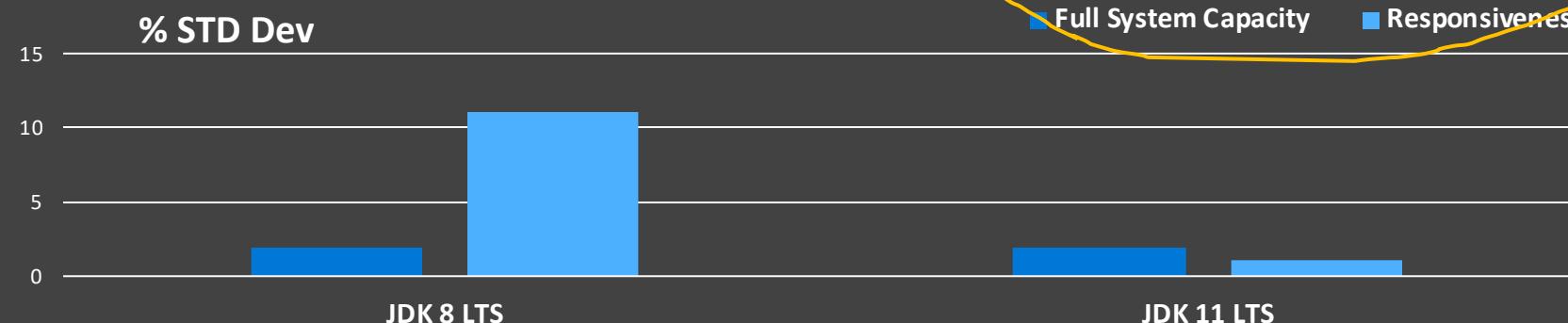
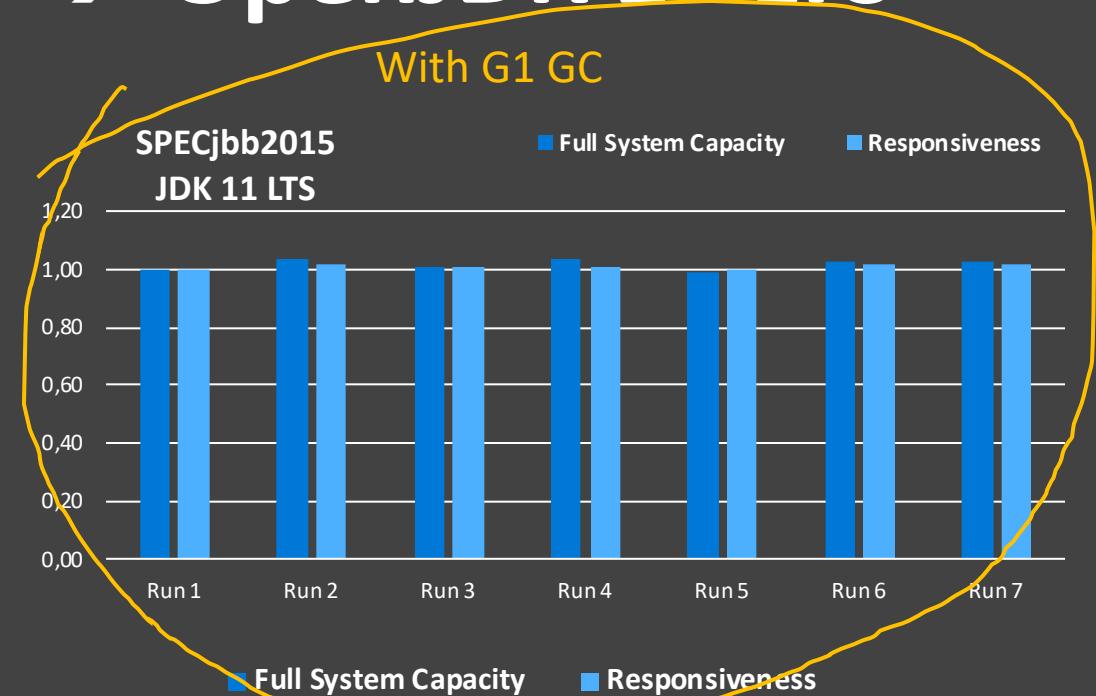
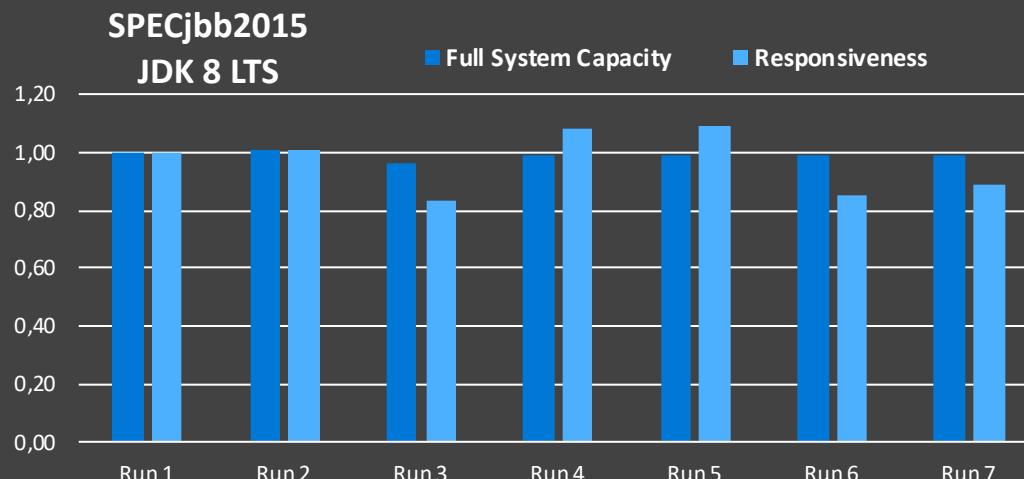


<https://developers.redhat.com/blog/2019/06/28/shenandoah-gc-in-jdk-13-part-2-eliminating-the-forward-pointer-word/>

Performance!



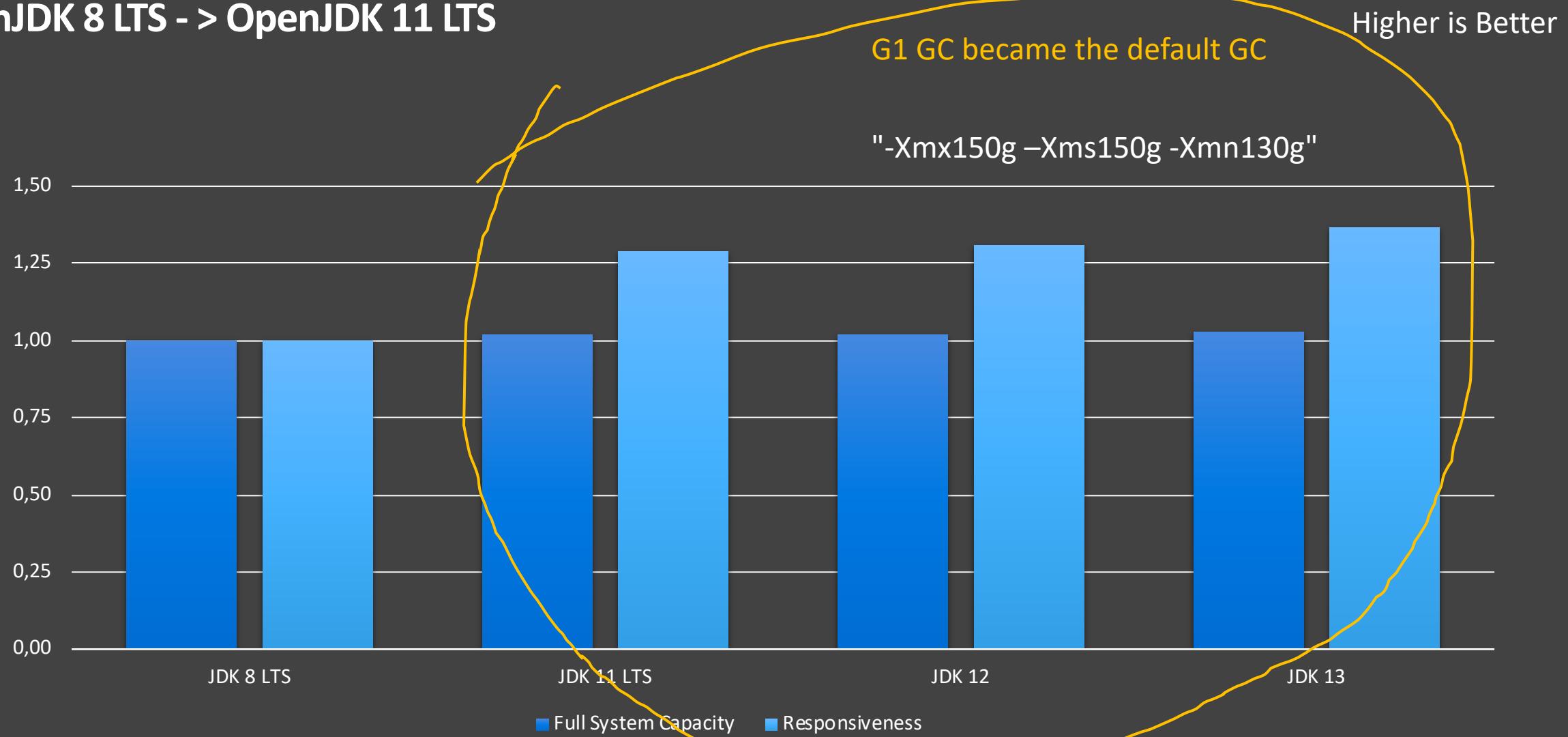
Variability: OpenJDK 8 LTS → OpenJDK 11 LTS



JDK 11 LTS significantly less variability than JDK 8 LTS for responsiveness

Out-of-box* GC Performance

OpenJDK 8 LTS -> OpenJDK 11 LTS

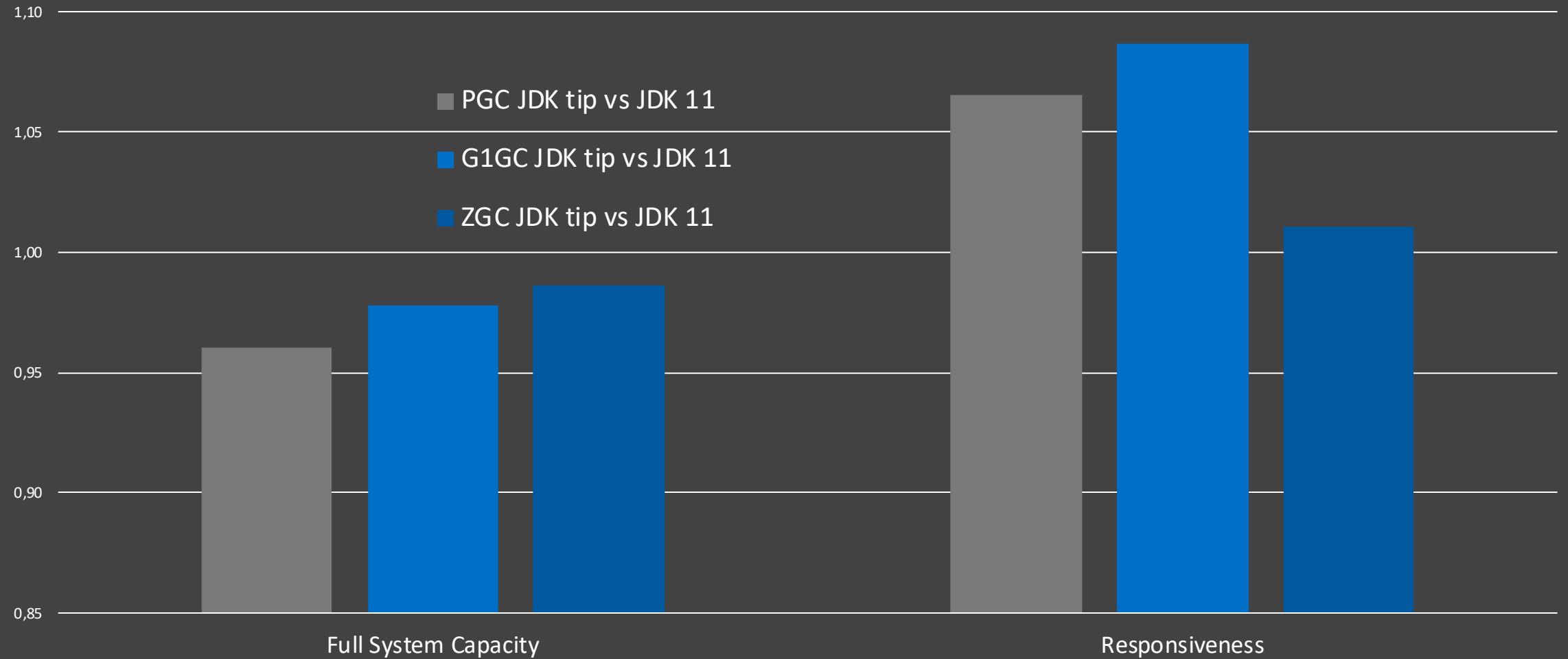


Out-of-box* OpenJDK GC Performance

Innovation happens at tip

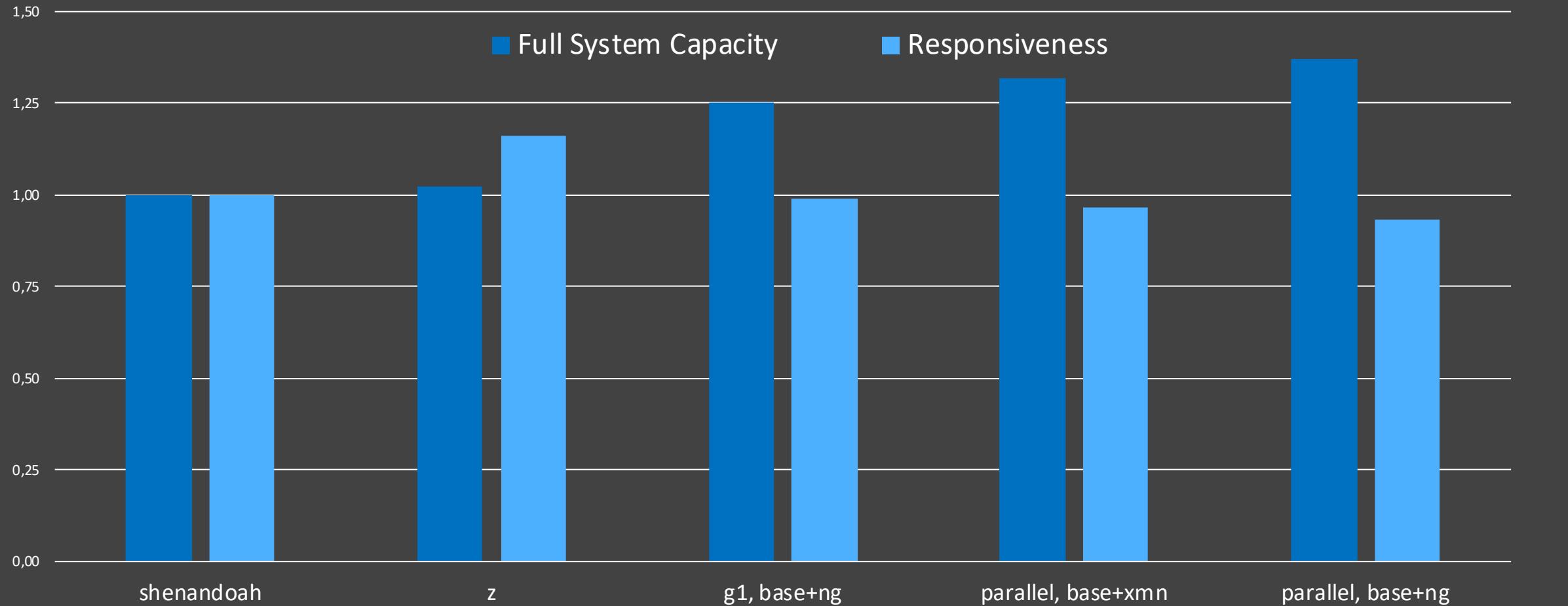
Higher is Better

*With Xmx=Xms



GCs Head-to-Head Performance

Higher is Better



Further Reading

<https://www.youtube.com/watch?v=VCeHkcwfF9Q>

https://www.usenix.org/legacy/events/vee05/full_papers/p46-click.pdf

<http://mail.openjdk.java.net/pipermail/zgc-dev/2017-December/000047.html>

<http://hg.openjdk.java.net/zgc/zgc/file/ffab403eaf14/src/hotspot/share/gc/z/zBarrier.cpp>

<https://wiki.openjdk.java.net/display/zgc/Main>



Microsoft