
Java™
ORACLE

Saga of Scalable Garbage Collectors

Vaibhav Choudhary
Java Platforms Team

bangalorejug.org
blogs.oracle.com/vaibhav

Java Your Next (Cloud)

Program Agenda

1. Basics of Garbage Collector
2. Prerequisites …
3. Working with Z GC
4. Some benchmarking reports

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.

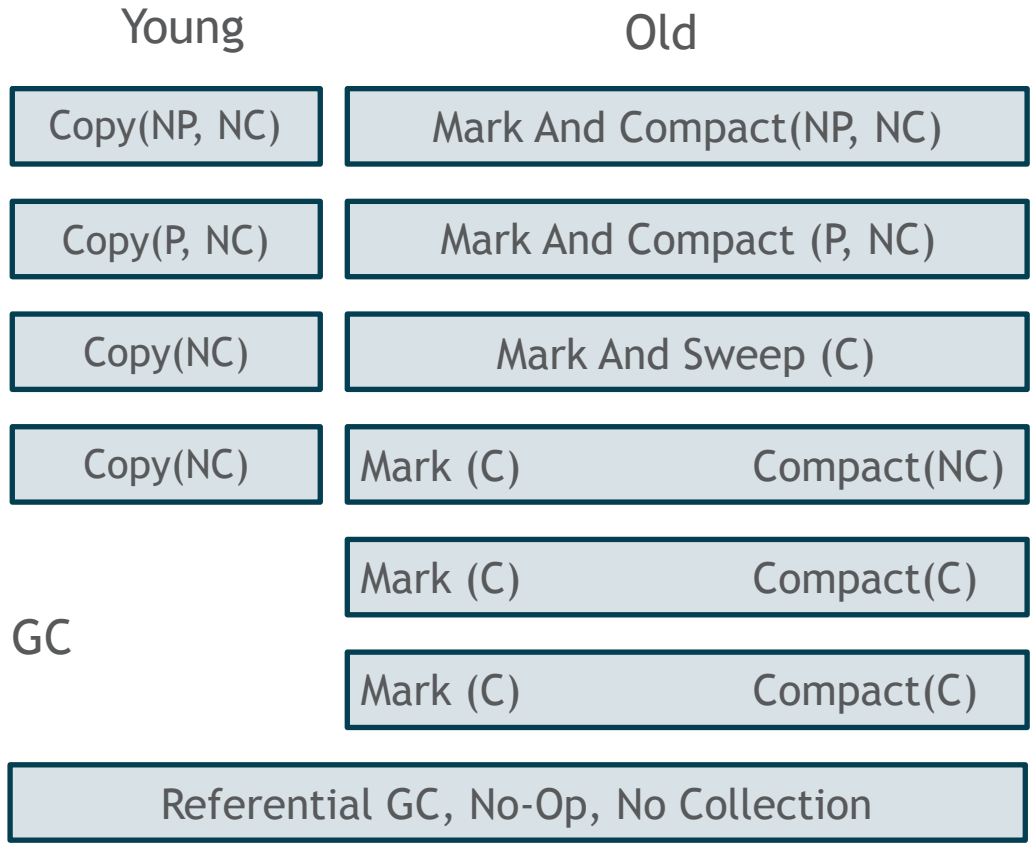
Basic of Garbage Collectors

All GC at a glance

- Serial GC
- Parallel GC
- CMS
- G1 GC
- Z GC
- Shenandoah GC
- Epsilon GC

Two generational GC

Single generational GC



Understanding Z GC

Z GC - In nutshell

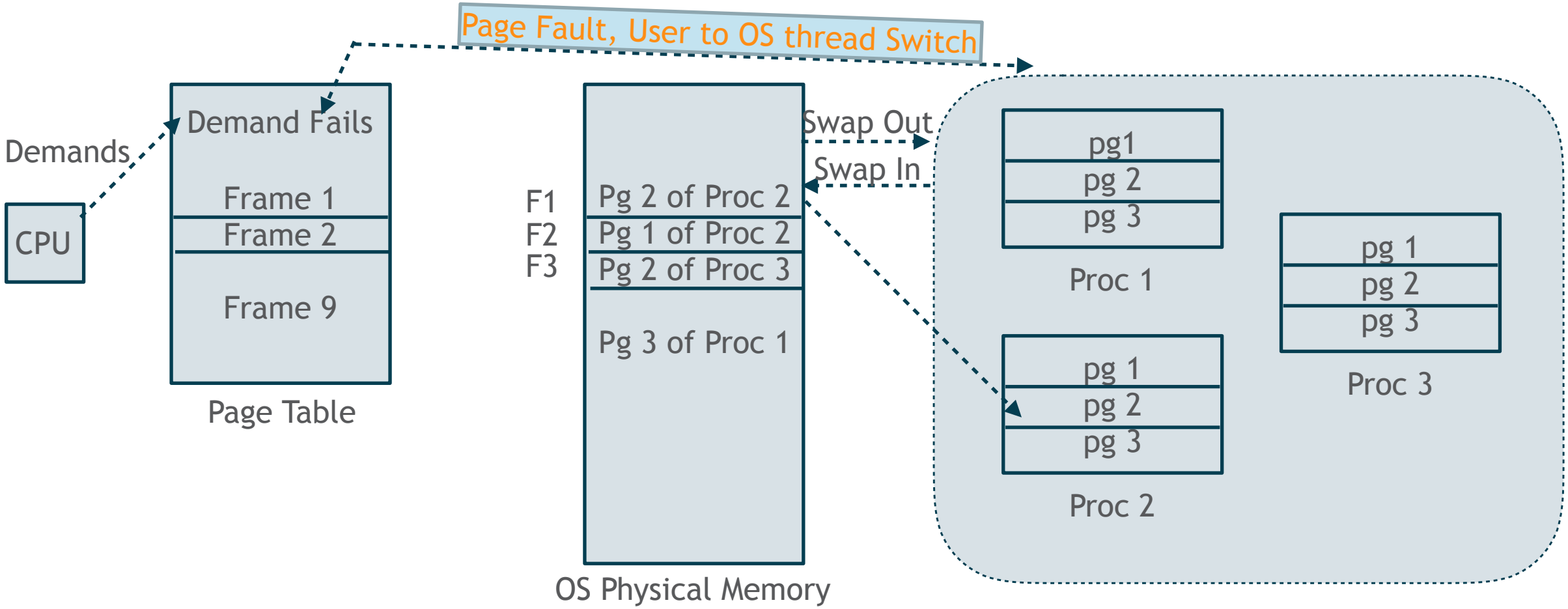
- The Z Garbage Collector, also known as ZGC, is a scalable low latency garbage collector designed to meet the following goals:
  - Pause times do not exceed 10ms
  - Pause times do not increase with the heap or live-set size
  - Handle heaps ranging from a few hundred megabytes to multi terabytes in size
- At a glance, ZGC is:
  - Concurrent
  - Region-based
  - Compacting
  - NUMA-aware
  - Using colored pointers
  - Using load barriers

Understanding of ZPages

- Z GC is a region-based Garbage Collector like G1.
- Z GC uses three different size of regions (always in multiple of 2MB for x64) unlike G1.
- The regions are :
  - ZPageTypeSmall (Size : 2 MB)
  - ZPageTypeMedium (Size : 32 MB)
  - ZPageTypeLarge (Size : n*2 MB)
- Details : http://hg.openjdk.java.net/zgc/zgc/file/59c07aef65ac/src/hotspot/share/gc/z/zPage.hpp

Understanding of Multi-mapping

- Basic understanding of virtual and physical memory



Understanding of Multi-mapping

- OS is responsible to map between virtual memory and physical memory.
- Generally done by MMU with the help of Page table and TLB.
- Z GC maps different virtual memory to same physical memory by multi-mapping
- mark0, mark1 and remap - by design only one can point to the same physical memory.

Understanding of Multi-mapping

```
//  +-----------------+-+----+-----------------------------------------------+
//  |00000000 00000000 0|0|1111|11 11111111 11111111 11111111 11111111 11111111|
//  +-----------------+-+----+-----------------------------------------------+
//  |                   | |    |
//  |                   | |    * 41-0 Object Offset (42-bits, 4TB address space)
//  |                   | |
//  |                   | * 45-42 Metadata Bits (4-bits)  0001 = Marked0      (Address view 4-8TB)
//  |                   |                                 0010 = Marked1      (Address view 8-12TB)
//  |                   |                                 0100 = Remapped     (Address view 16-20TB)
//  |                   |                                 1000 = Finalizable  (Address view N/A)
//  |                   |
//  |                   * 46-46 Unused (1-bit, always zero)
//  |
//  * 63-47 Fixed (17-bits, always zero)
```

* http://hg.openjdk.java.net/zgc/zgc/file/59c07aef65ac/src/hotspot/os_cpu/linux_x86/zGlobals_linux_x86.hpp#l61

Understanding of Colored Pointers

- Z GC works with 64-bit only.
- No CompressedOOPS
- 42 bit point to 4 TB space.
- 4 bit for mapping - finalizable, remap, mark0, mark1

Understanding of Load Barriers

- Piece of code insert by JIT when application thread loads a reference.
- C1 Barrier set - http://hg.openjdk.java.net/zgc/zgc/file/3cf81d157e9f/src/hotspot/share/gc/z/c1
- C2 Barrier set - http://hg.openjdk.java.net/zgc/zgc/file/3cf81d157e9f/src/hotspot/share/gc/z/c2
- GC need to do additional action whenever a reference gets loaded.
  - Object o = obj.field; // triggers load barrier
  - Object o1 = o; // not trigger load barrier
- ZGC need a read barrier not write barrier unlike generational GC.
- Required for concurrent compaction.

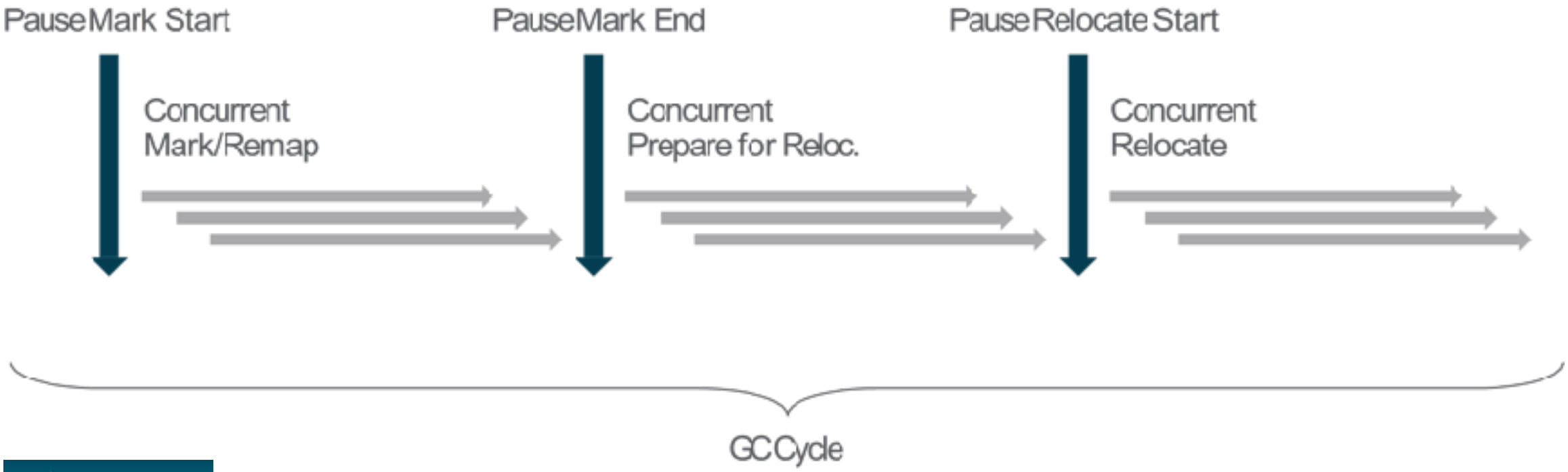
Phases of Z GC

- http://hg.openjdk.java.net/zgc/zgc/file/59c07aef65ac/src/hotspot/share/gc/z/zDriver.cpp#l304
- Phases :
  - Phase 1: Pause Mark Start (STW - very small)
  - Phase 2: Concurrent Mark
  - Phase 3: Pause Mark End (STW - very small)
  - Phase 4: Concurrent Reference Processing
  - Phase 5: Concurrent Reset Relocation Set
  - Phase 6: Concurrent Destroy Detached Pages
  - Phase 7: Concurrent Select Relocation Set
  - Phase 8: Prepare Relocation Set
  - Phase 9: Pause Relocate Start (STW - very small)
  - Phase 10: Concurrent Relocate

Z GC: A Scalable Low Latency Garbage Collector

Goals:

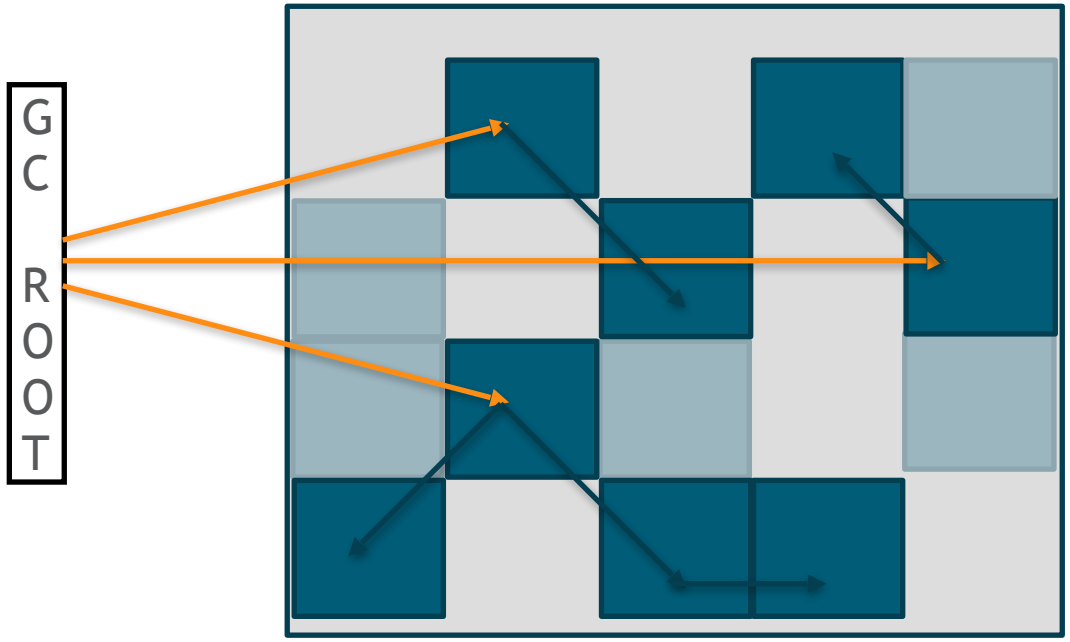
- Scales up to Multi-terabyte heaps
- 10ms as max GC pause time
- GC pause times do not increase with heap or live-set size -XX:+UseZGC



GC Root Marking and concurrent marking

* Class loaded by boot_class_loader. Object holding
    * via static field.
* Live threads.
* Stack Locals
* JNI Local and Globals
* And few more like monitors, JVM specials.

— Object not reachable from Object Graph are free be collected.

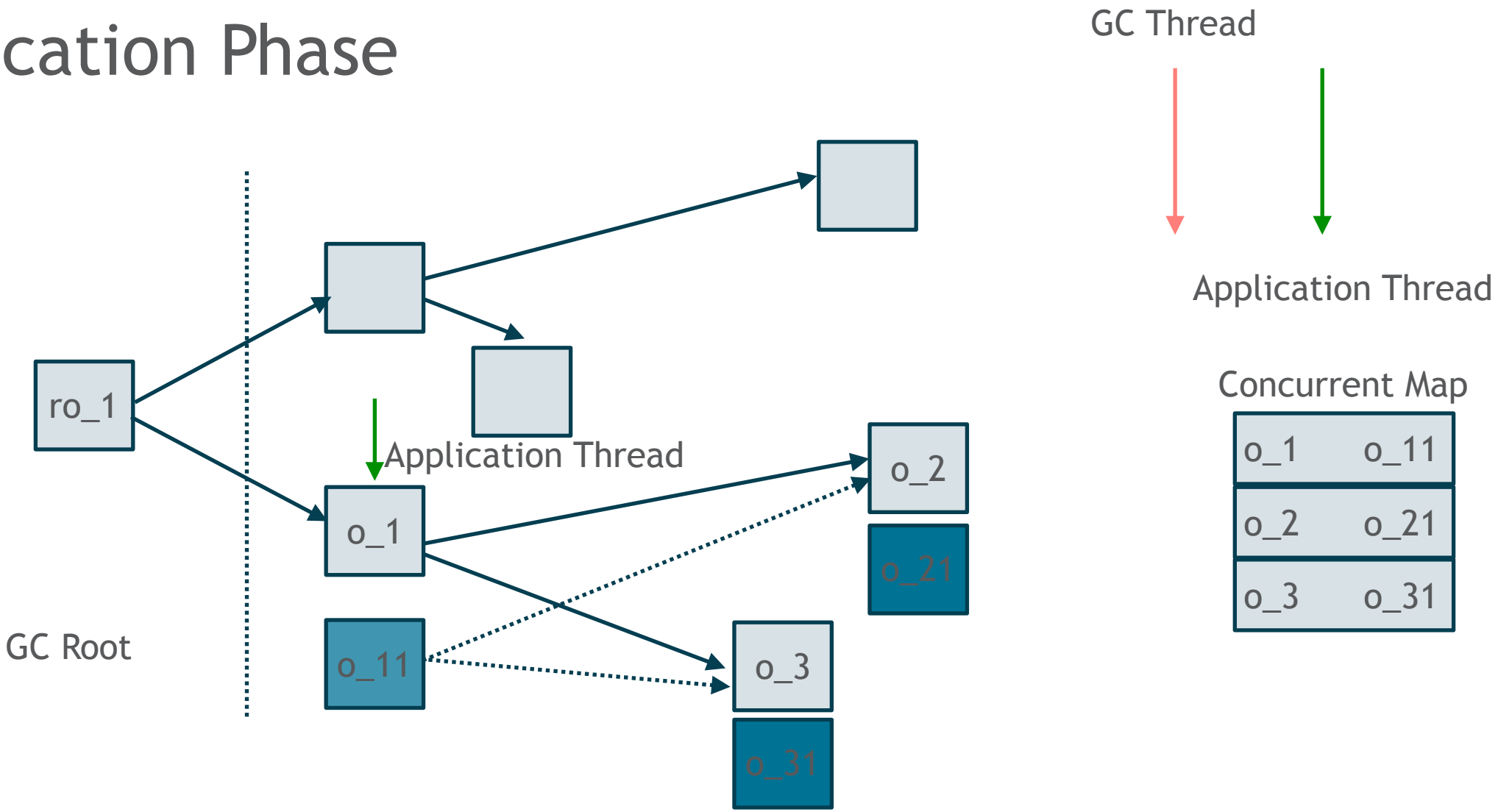


- Once GC root will be analyzed, GC will begin the next phase concurrently and mark all the objects.
- Load barrier will be tested for masking.

Relocation Phase

- Z GC divides the heap into ZPages.
- At the start, concurrently select the set of pages whose live object need to be relocated.
- A small STW to remap the root to the new reference.
- These STW don’t scale with overall heap size, because roots are little.
- Next phase is concurrent relocation and in this phase it walks to relocation set and relocate all the object in the pages it contains.
- Application thread can also relocate object [Use Load Barrier]

Relocation Phase



* http://hg.openjdk.java.net/zgc/zgc/file/59c07aef65ac/src/hotspot/share/gc/z/zRelocate.cpp#l121

Links and QA

- Z GC Main Page: https://wiki.openjdk.java.net/display/zgc/Main
- First look at Z GC : https://dinfuehr.github.io/blog/a-first-look-into-zgc/
- New Z GC is exciting : https://www.opsian.com/blog/javas-new-zgc-is-very-exciting/
- Some videos :
  - Concurrent Z GC : https://www.youtube.com/watch?v=WU_mqNBEacw&t=1776s
- Benchmarking reports : https://ionutbalosin.com/2019/12/jvm-garbage-collectors-benchmarks-report-19-12/