The Design of ZGC

A Scalable Low-Latency Garbage Collector for Java

Per Lidén (@perliden)
Consulting Member of Technical Staff
Java Platform Group, Oracle
June 12, 2019

developer.oracle.com

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.

The Design of ZGC

A Scalable Low-Latency Garbage Collector

Goals

TB Multi-terabyte heaps

10ms Max GC pause time

Easy to tune

15% Max application throughput reduction

ZGC at a Glance

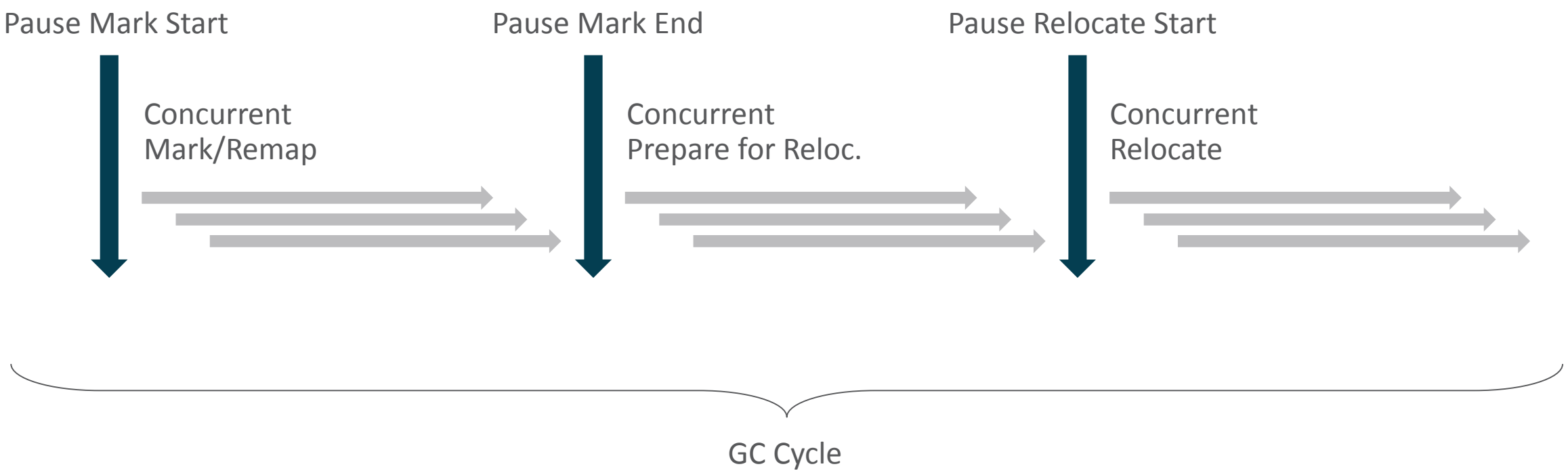
Concurrent	Region-based
Tracing	NUMA-aware
Compacting	Load barriers
Single generation	Colored pointers

ZGC pause times **do not** increase with the heap or live-set size

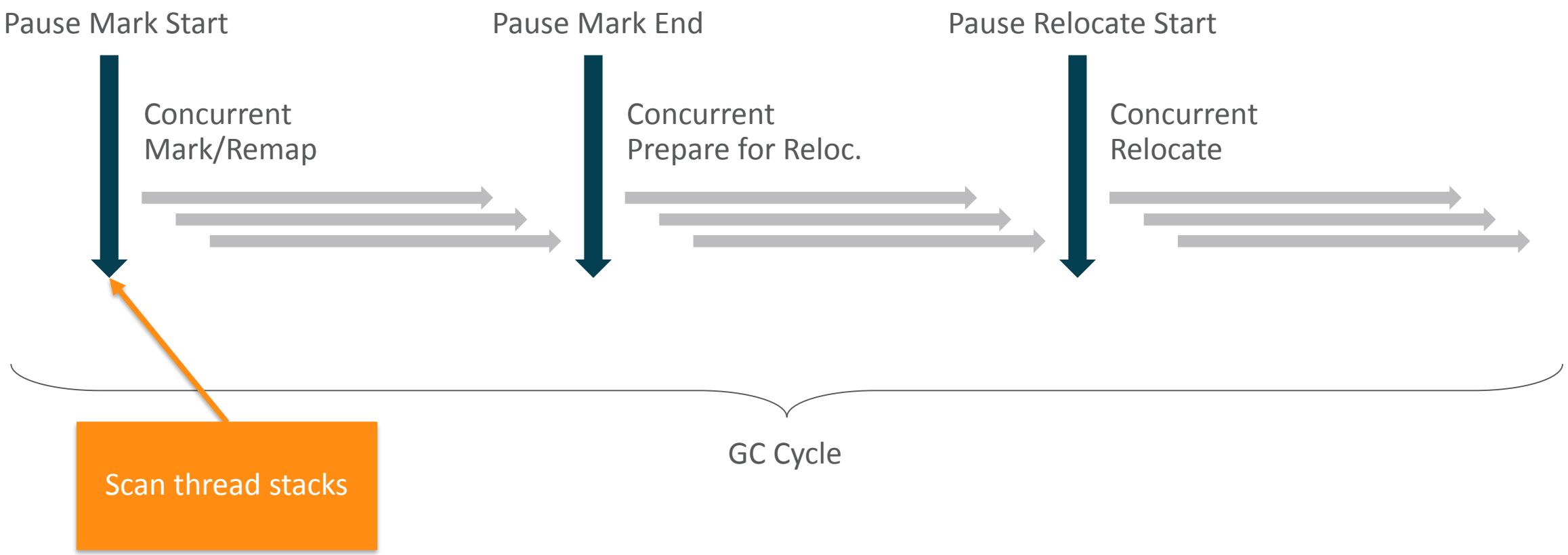
ZGC pause times **do** increase with the root-set size

(Number of Java Threads)

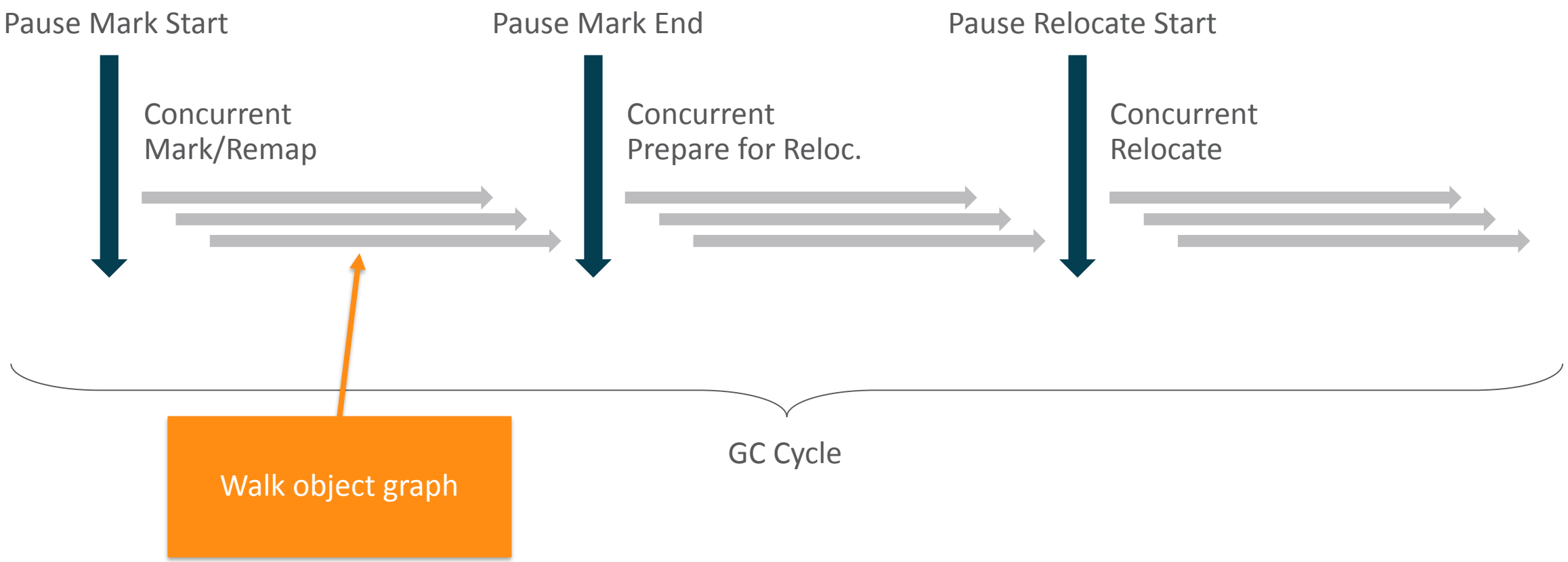
ZGC Phases



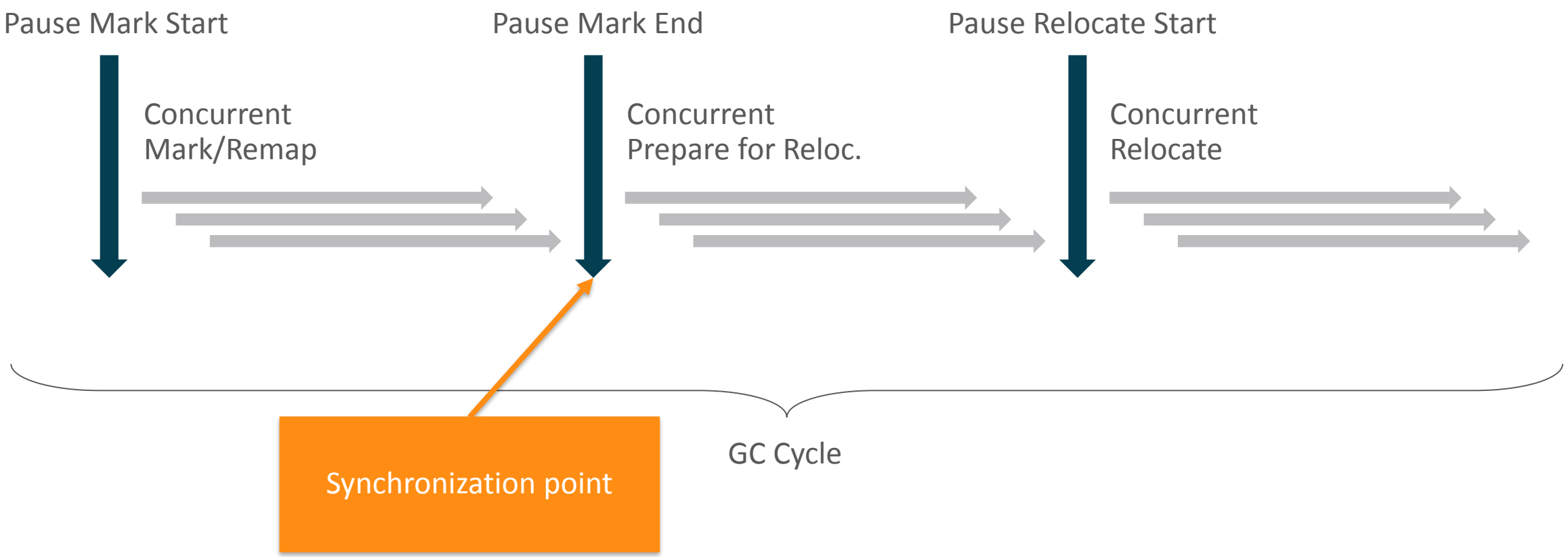
ZGC Phases



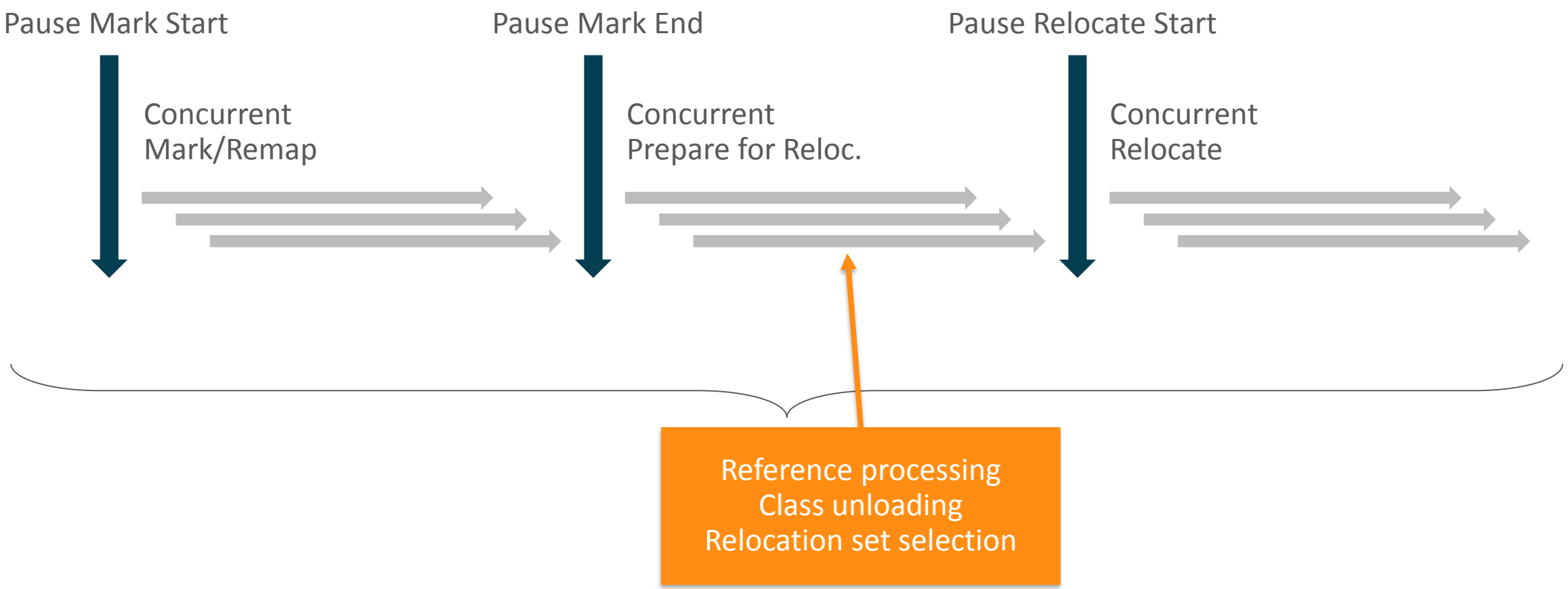
ZGC Phases



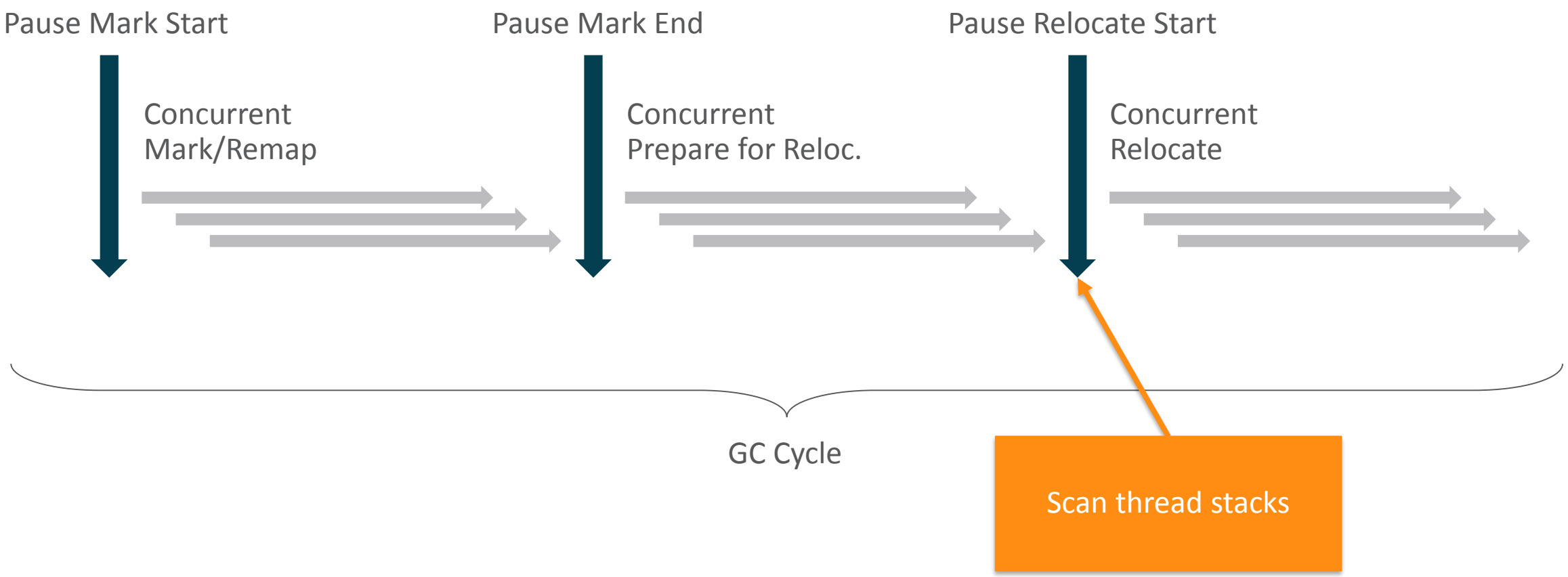
ZGC Phases



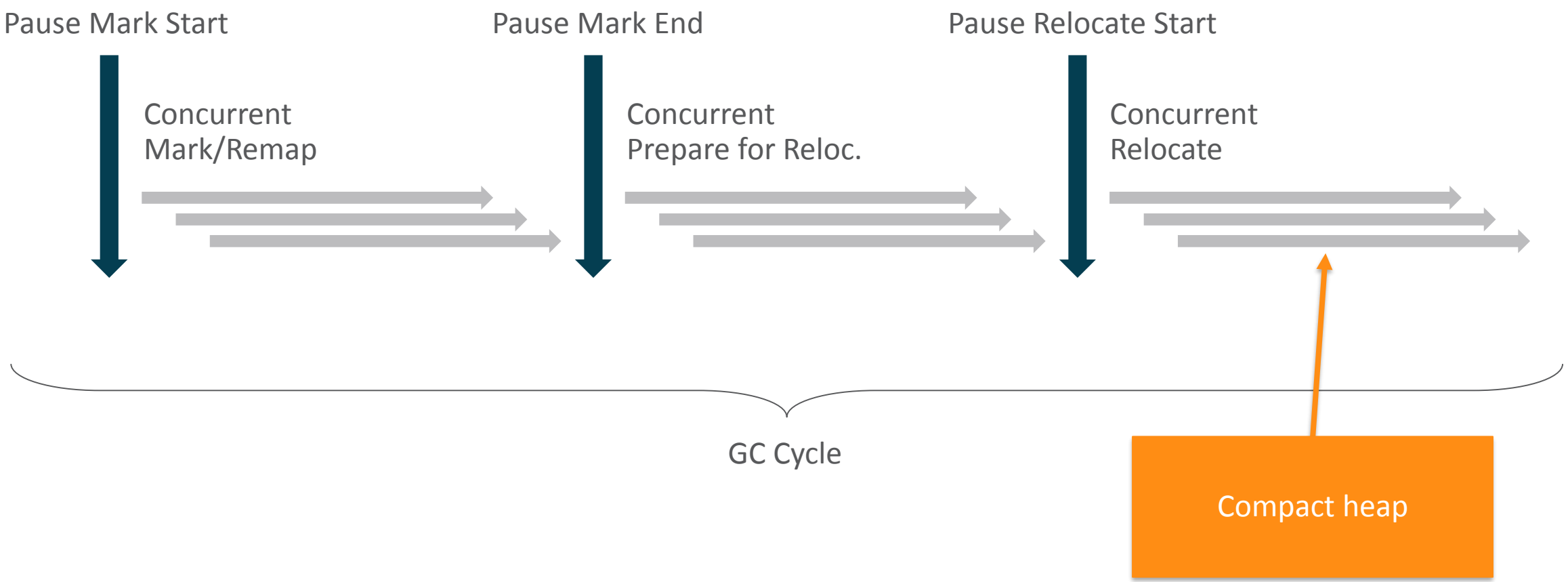
ZGC Phases



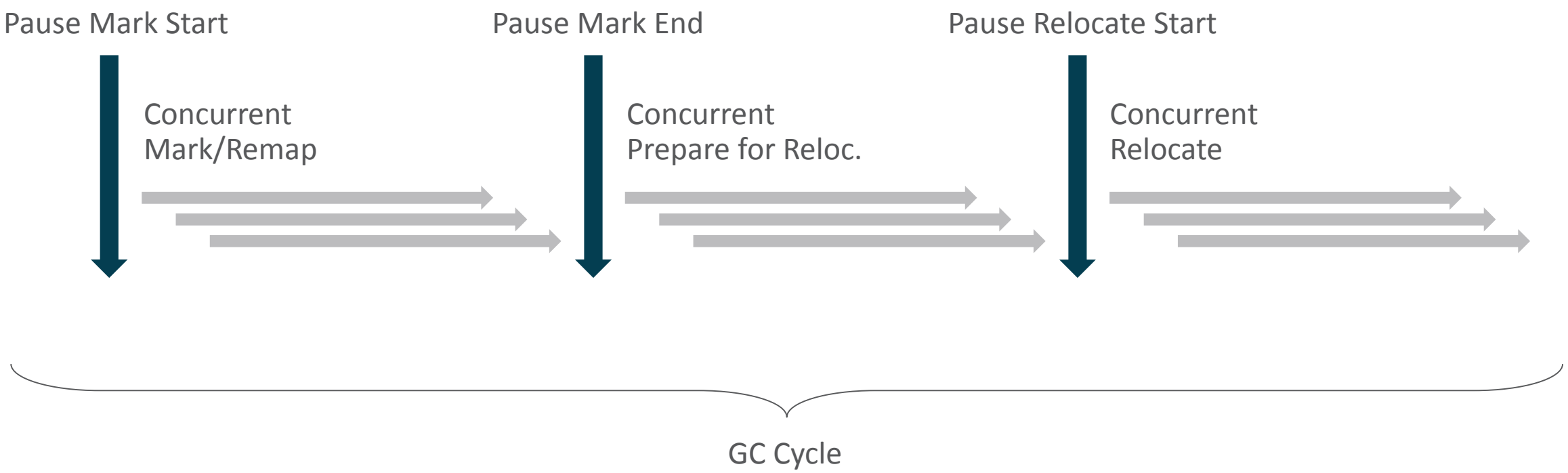
ZGC Phases



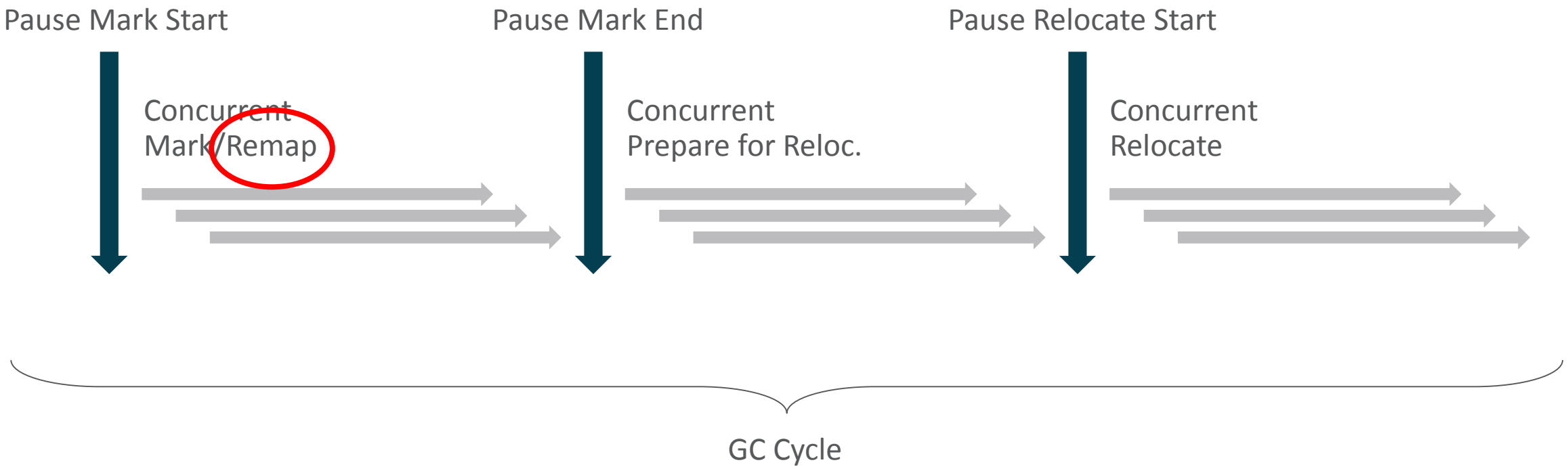
ZGC Phases



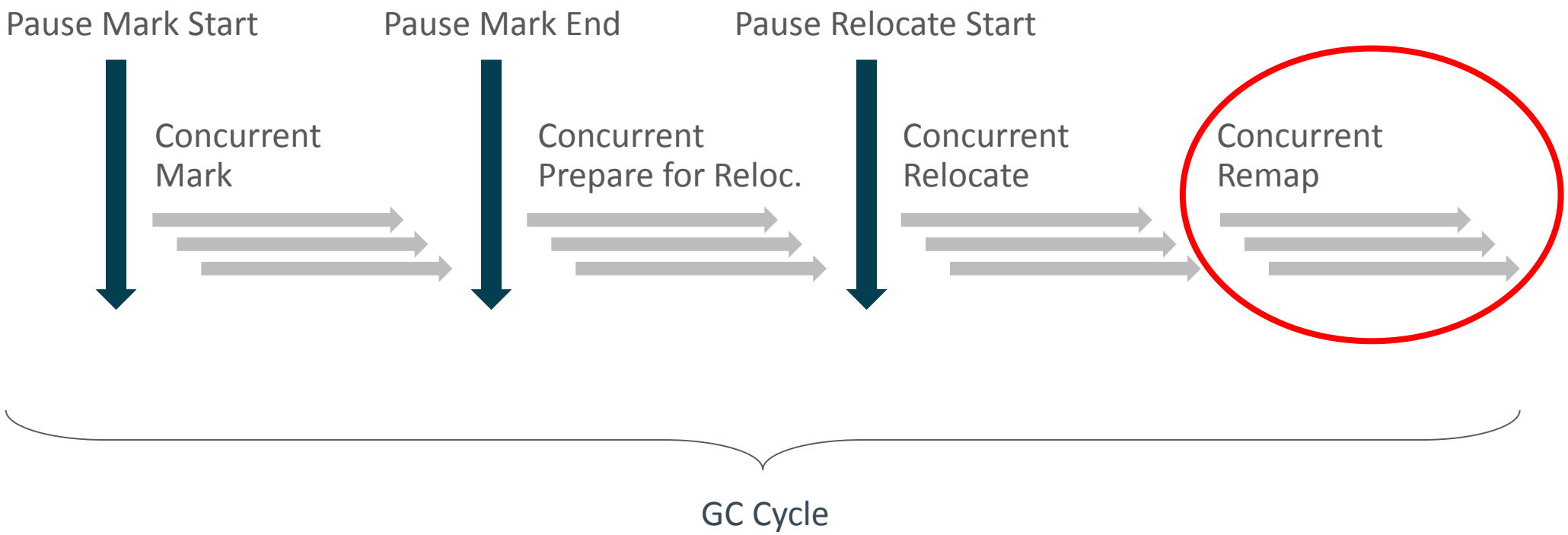
ZGC Phases



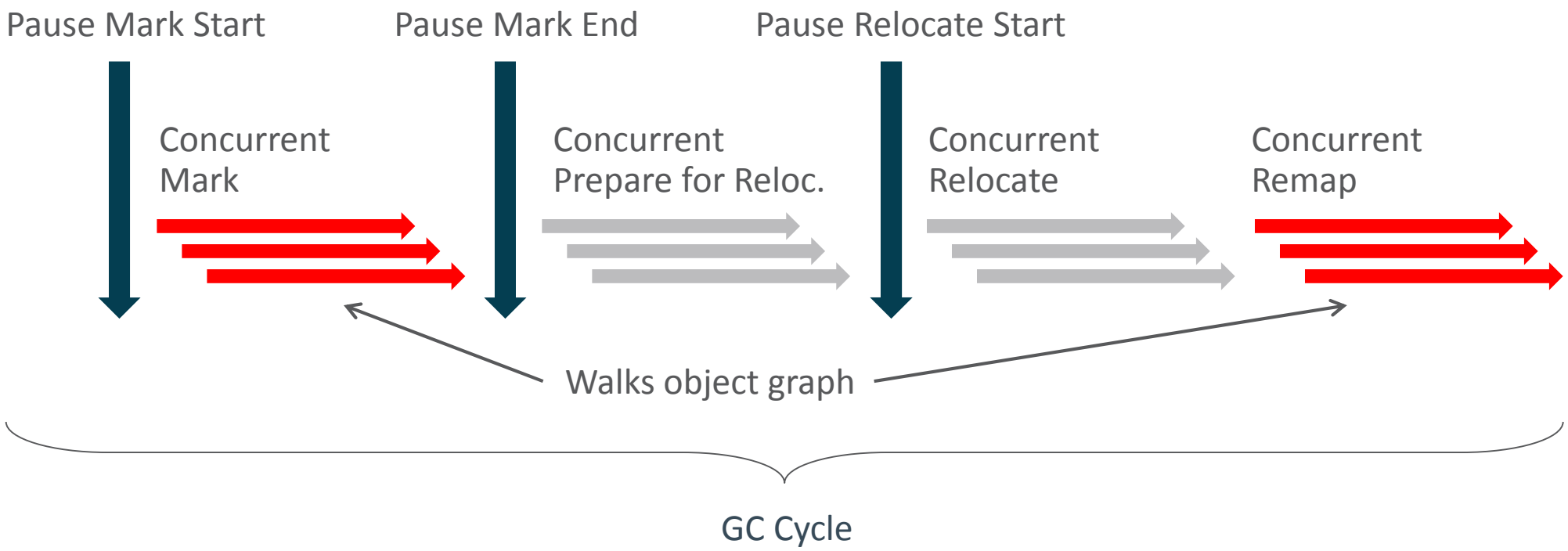
ZGC Phases



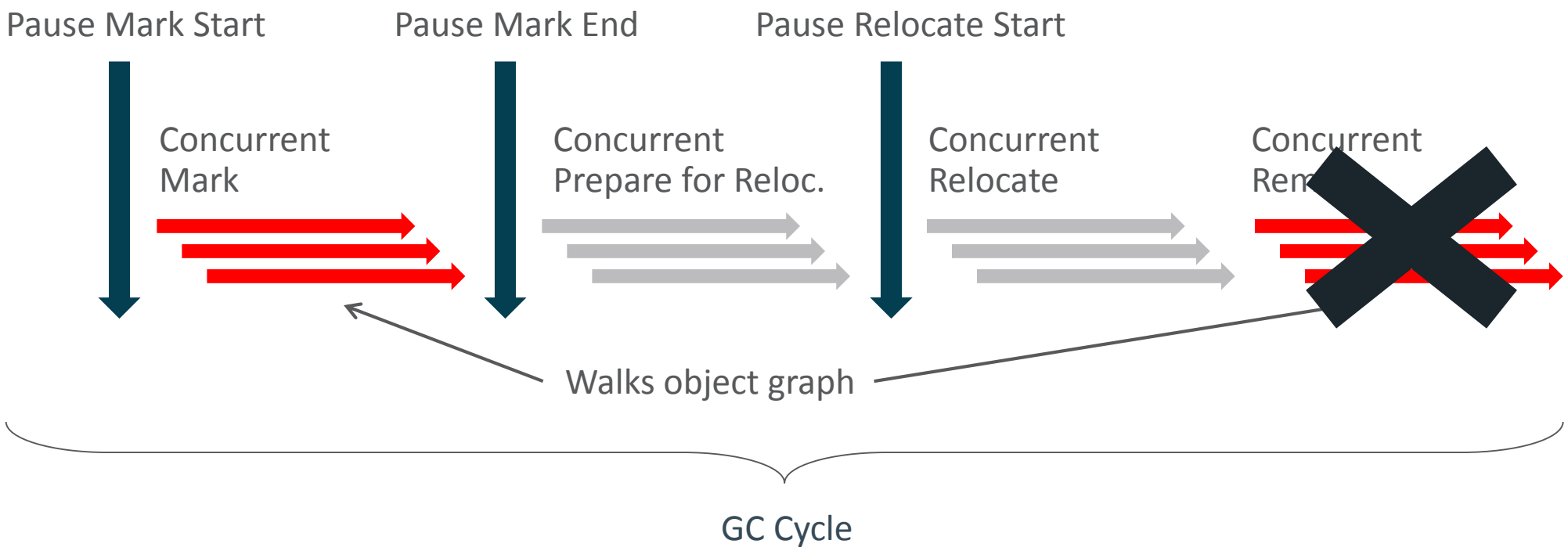
ZGC Phases



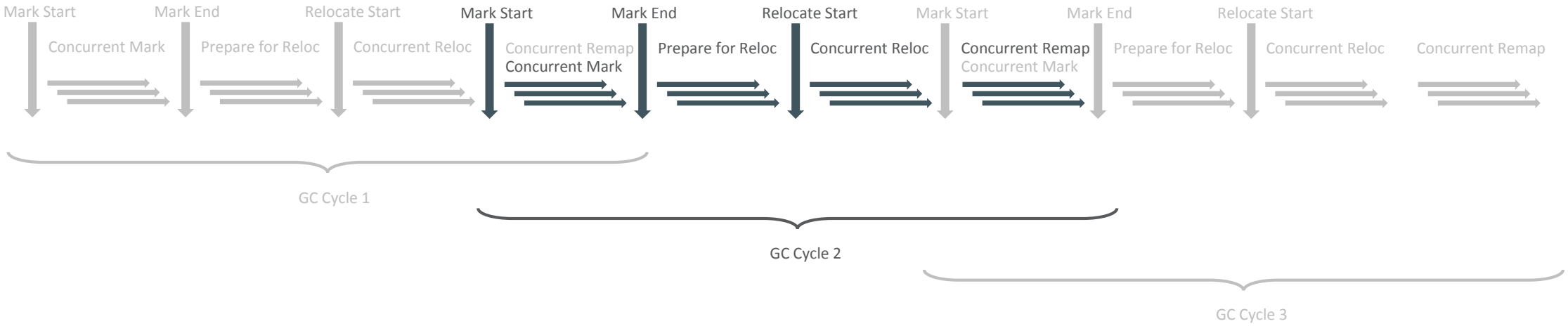
ZGC Phases



ZGC Phases

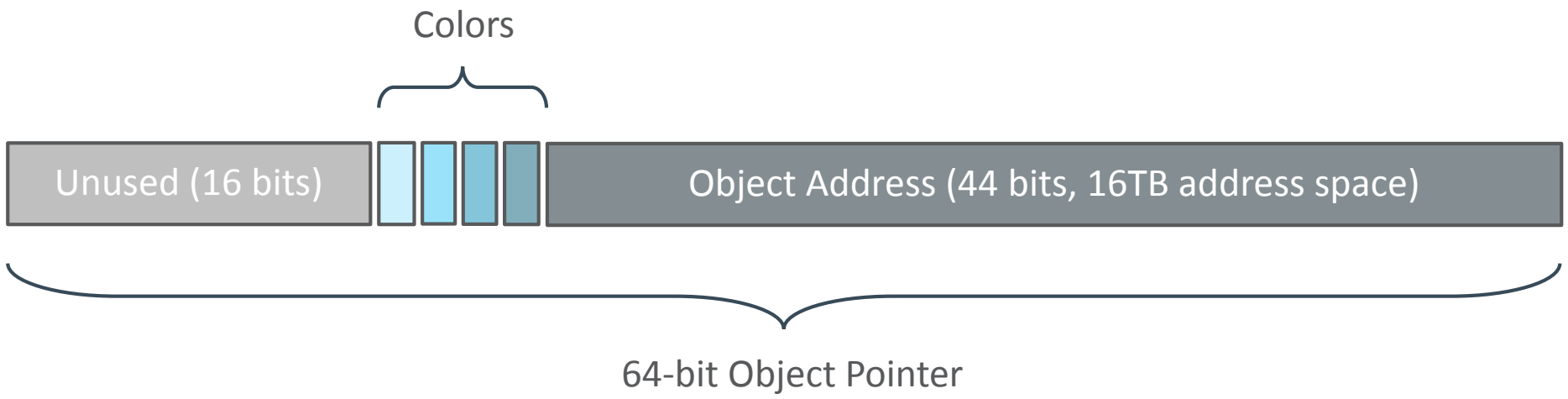


ZGC Phases

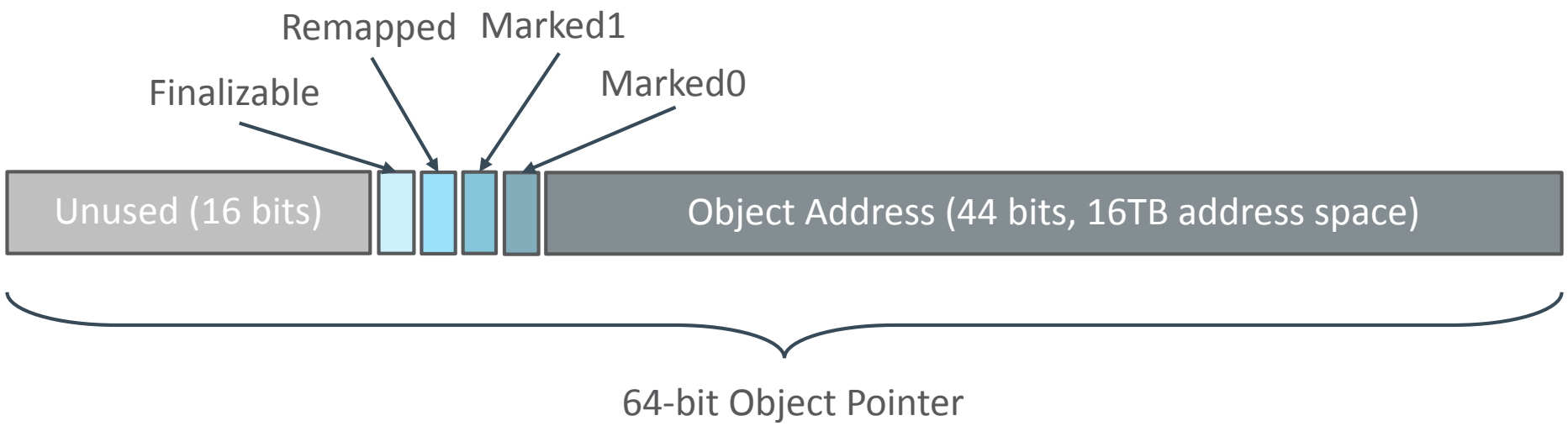


Colored Pointers

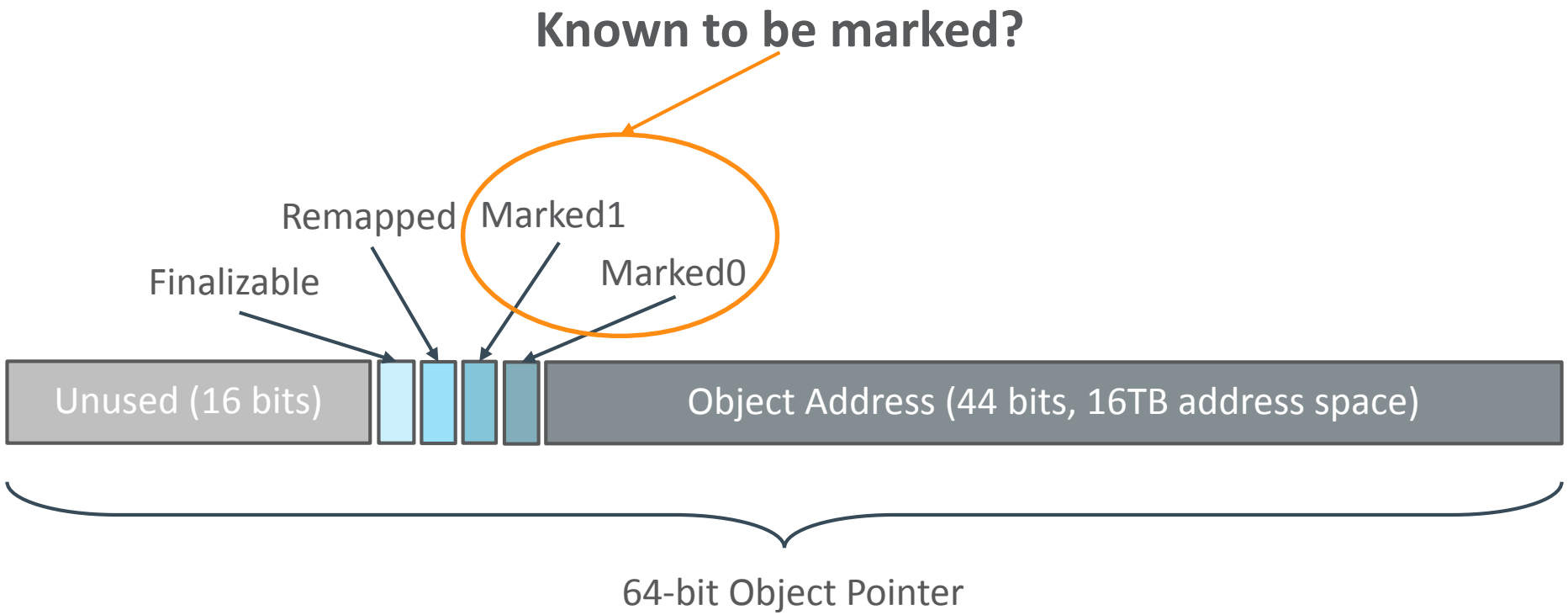
- Core design concept in ZGC
- **Metadata** stored in unused bits in 64-bit pointers
  - No support for 32-bit platforms
  - No support for CompressedOops



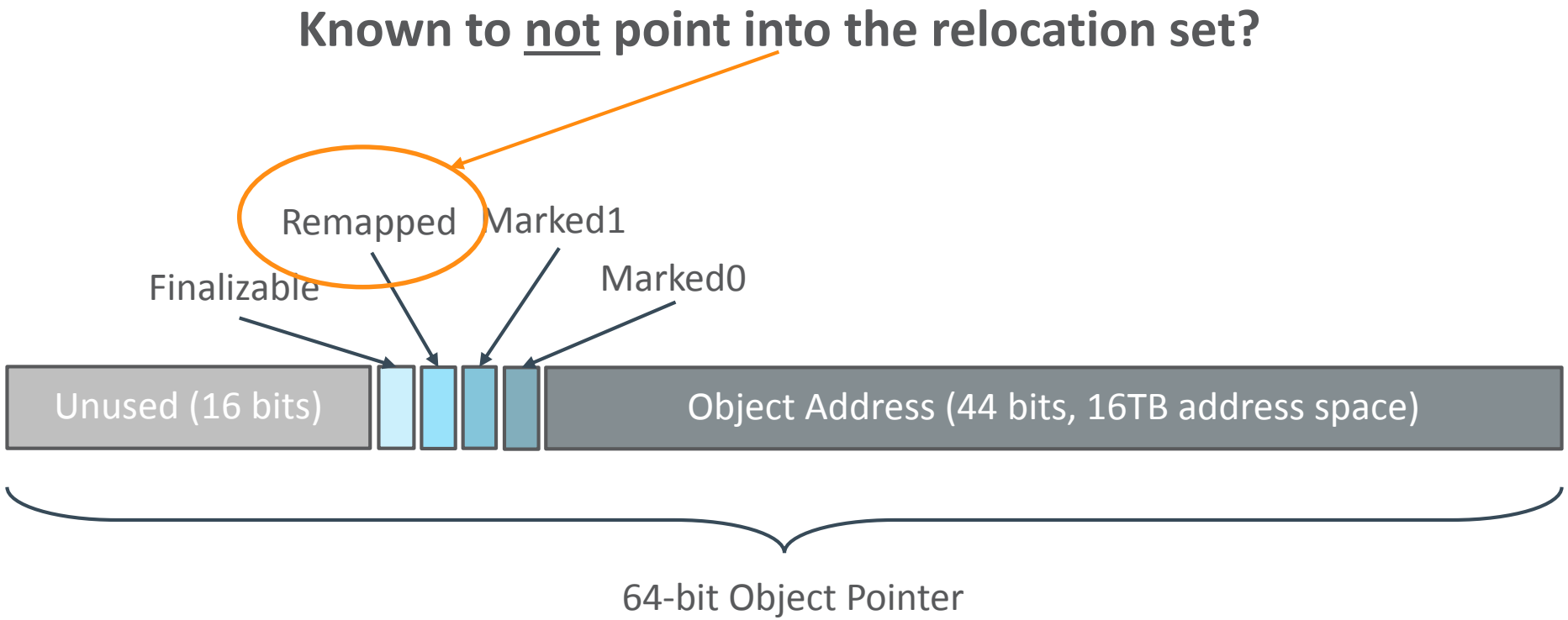
Colored Pointers



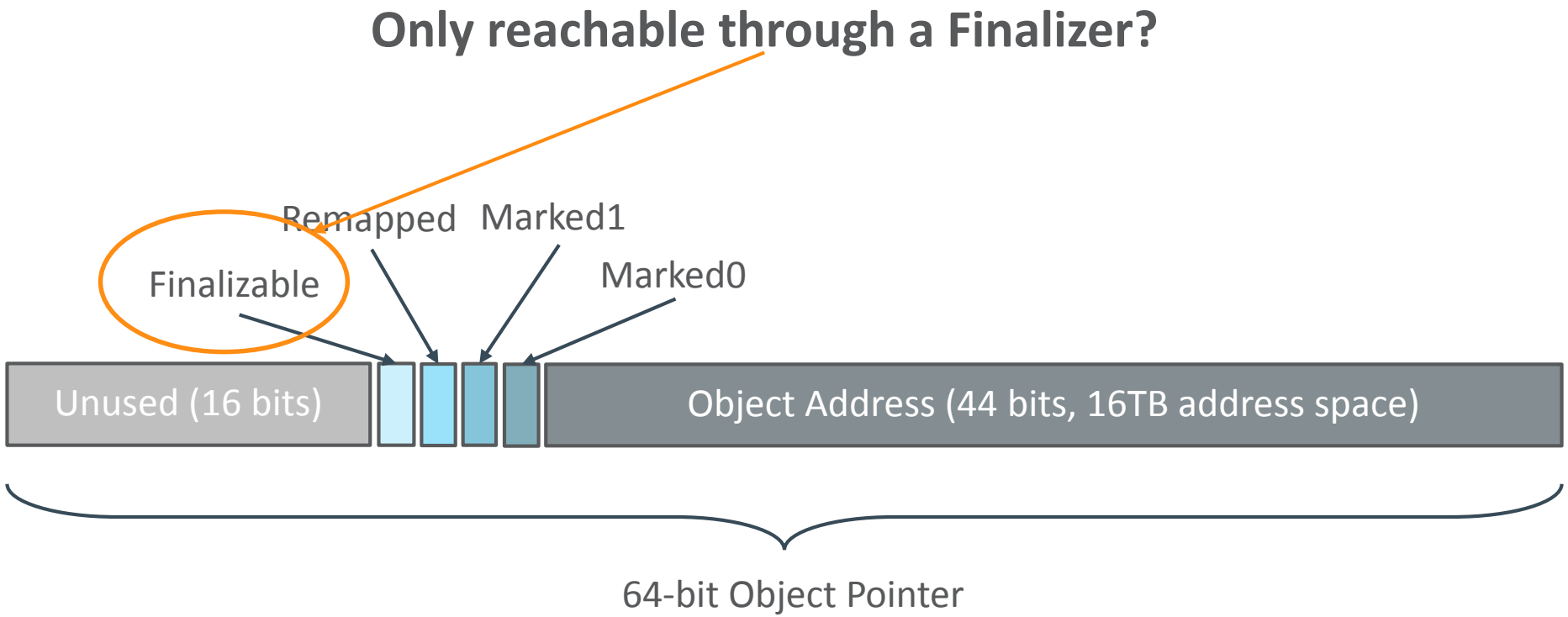
Colored Pointers



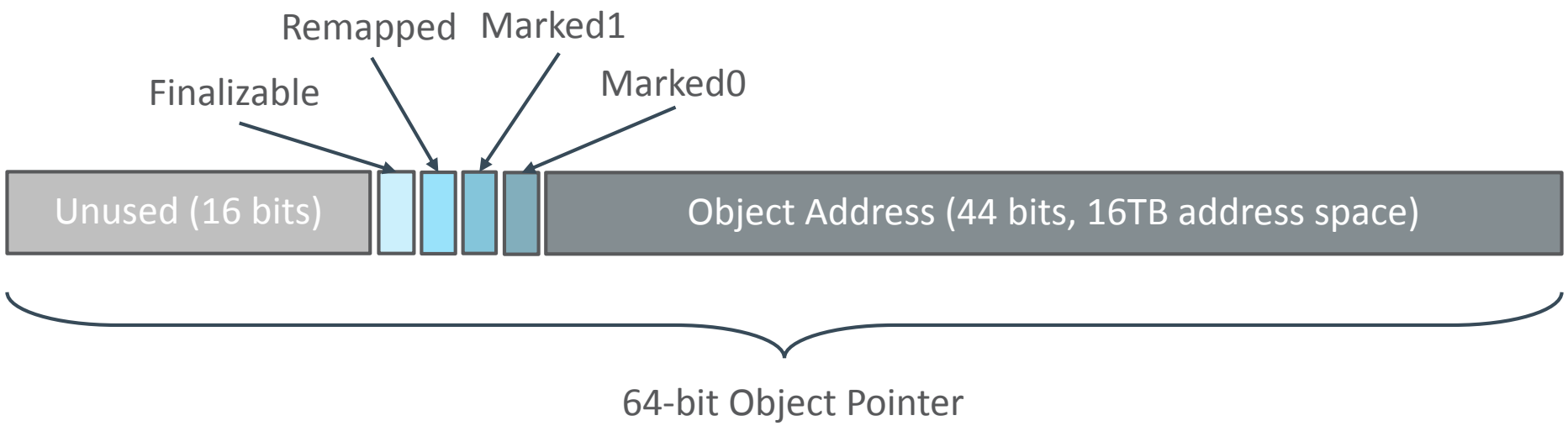
Colored Pointers



Colored Pointers



Colored Pointers



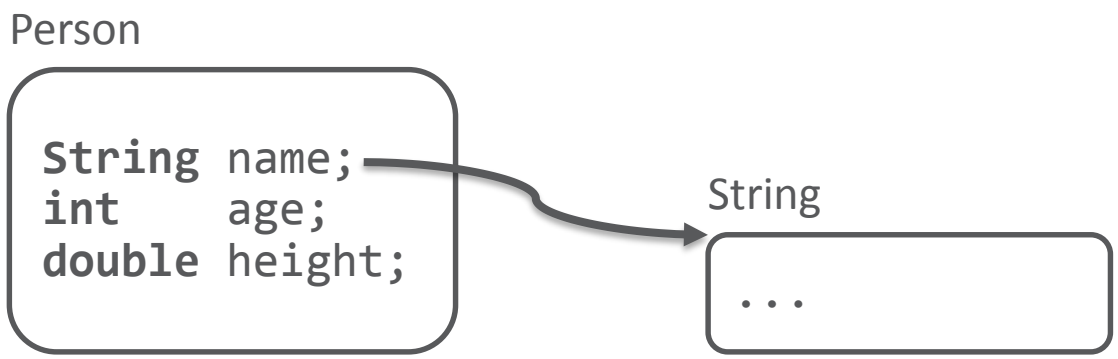
Load Barrier

- A small piece of code injected by the JIT in strategic places
  - When **loading an object reference from the heap**

- Checks if the loaded object reference has a **bad** color
  - If so, take **action** and **heal** it

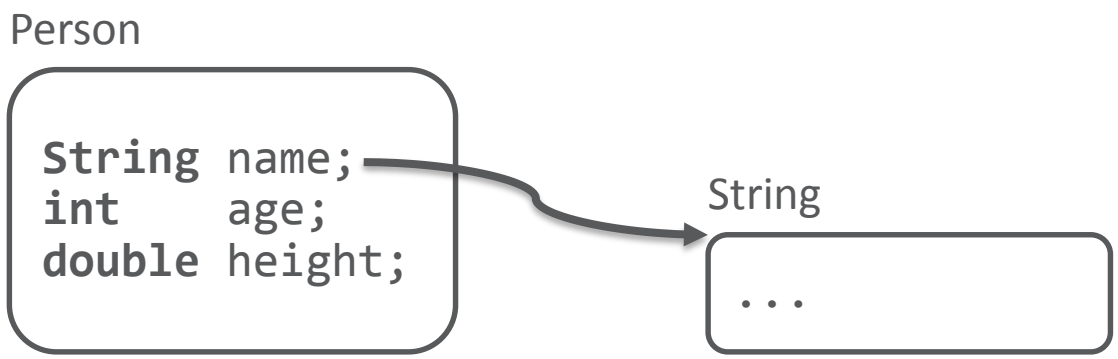
Load Barrier

```
String n = person.name;          // Loading an object reference from heap
```



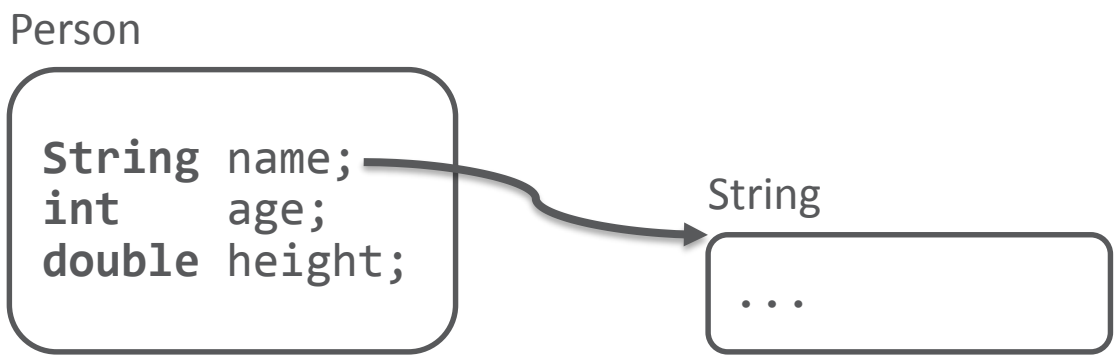
Load Barrier

```
String n = person.name;          // Loading an object reference from heap
<load barrier needed here>
```



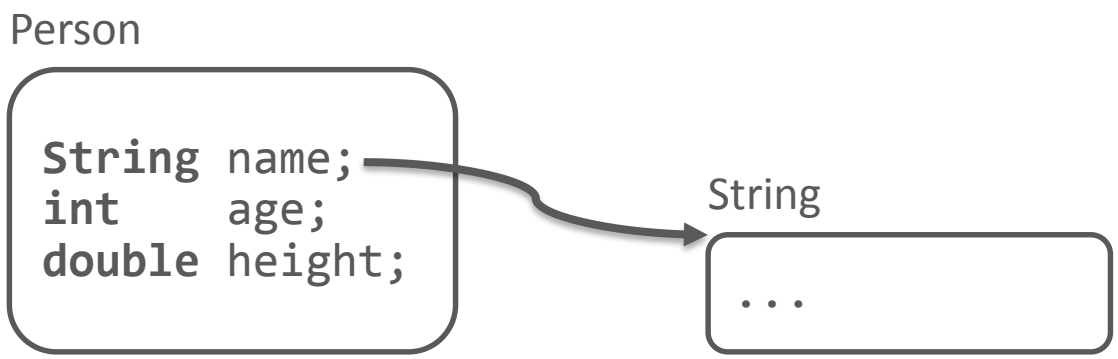
Load Barrier

```
String n = person.name;          // Loading an object reference from heap
<load barrier needed here>
String p = n;                    // No barrier, not a load from heap
n.isEmpty();                     // No barrier, not a load from heap
int age = person.age;            // No barrier, not an object reference
```



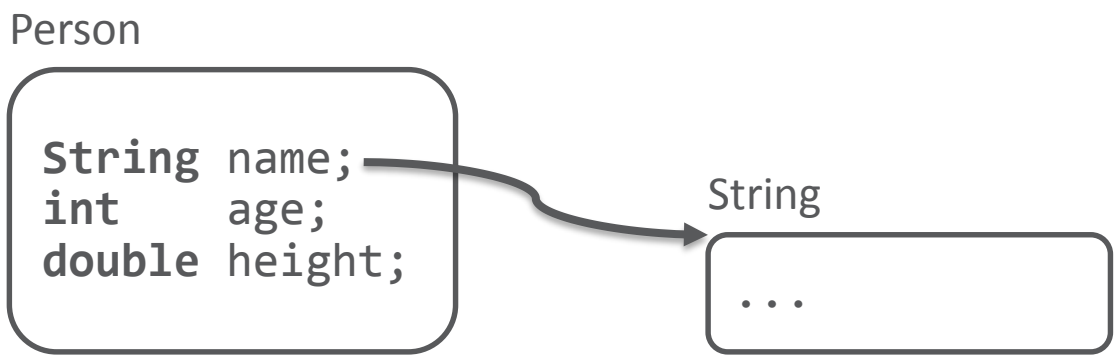
Load Barrier

```
String n = person.name;          // Loading an object reference from heap
<load barrier needed here>
```



Load Barrier

```
String n = person.name;          // Loading an object reference from heap
if (n & bad_bit_mask) {
  slow_path(register_for(n), address_of(person.name));
}
```



Load Barrier

```
mov     0x10(%rax), %rbx
test    %rbx, 0x20(%r15)
jnz     slow_path
```

```
// String n = person.name;
// Bad color?
// Yes -> Enter slow path and
// mark/relocate/remap, adjust
// 0x10(%rax) and %rbx
```

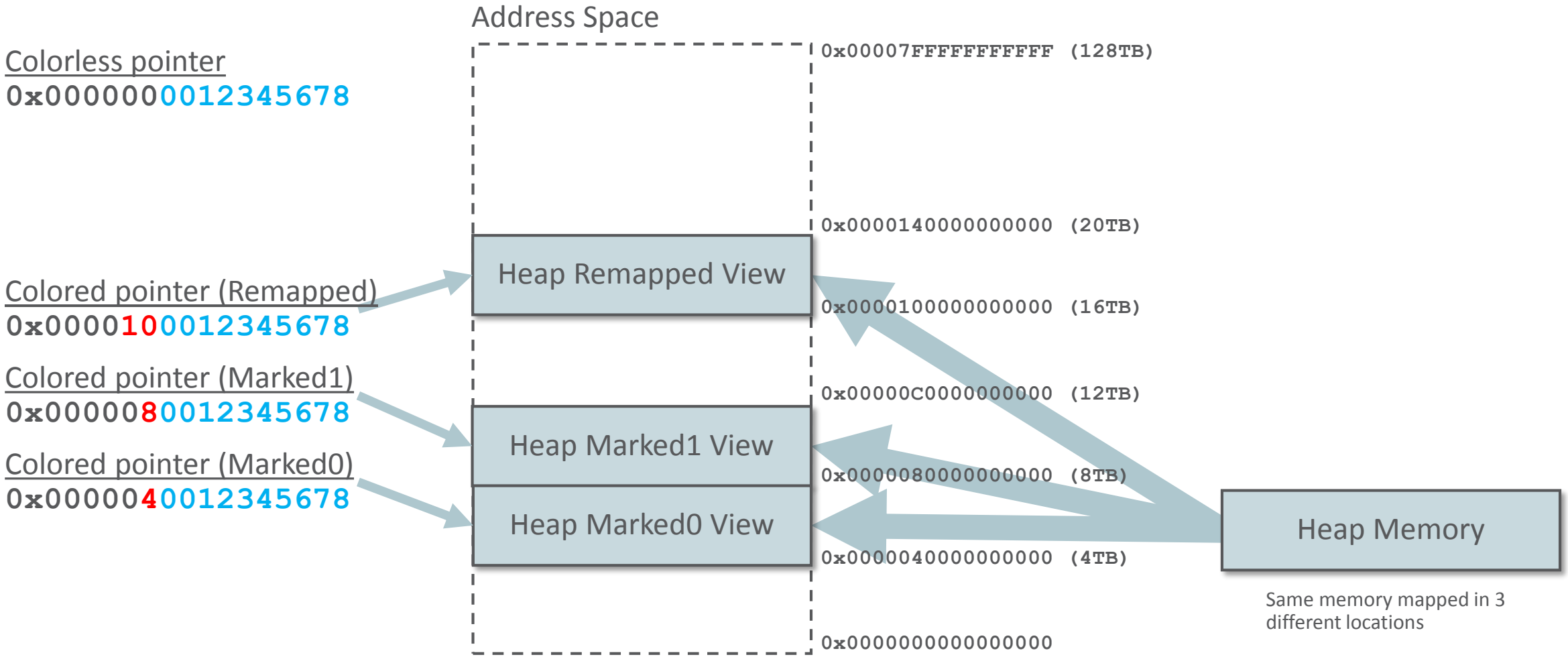
Load Barrier

```
mov     0x10(%rax), %rbx
test    %rbx, 0x20(%r15)
jnz     slow_path
```

```
// String n = person.name;
// Bad color?
// Yes -> Enter slow path and
// mark/relocate/remap, adjust
// 0x10(%rax) and %rbx
```

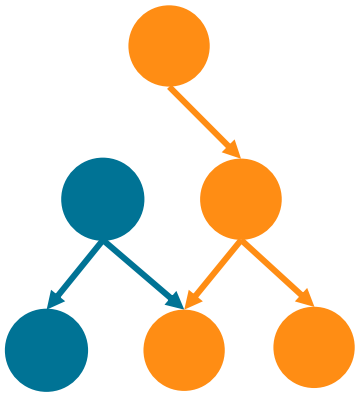
~4% execution overhead on **SPECjbb®2015**

Heap Multi-Mapping on **Linux/x86_64**

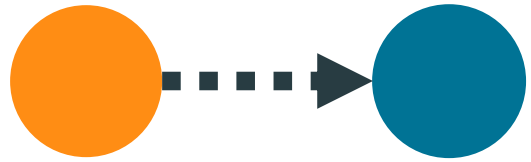


Mark

- Concurrent & Parallel
- Load barrier
  - Detects loads of non-marked object pointers
- Striped
  - Heap divided into logical stripes
  - Isolate each GC thread to work on its own stripe
  - Minimized shared state

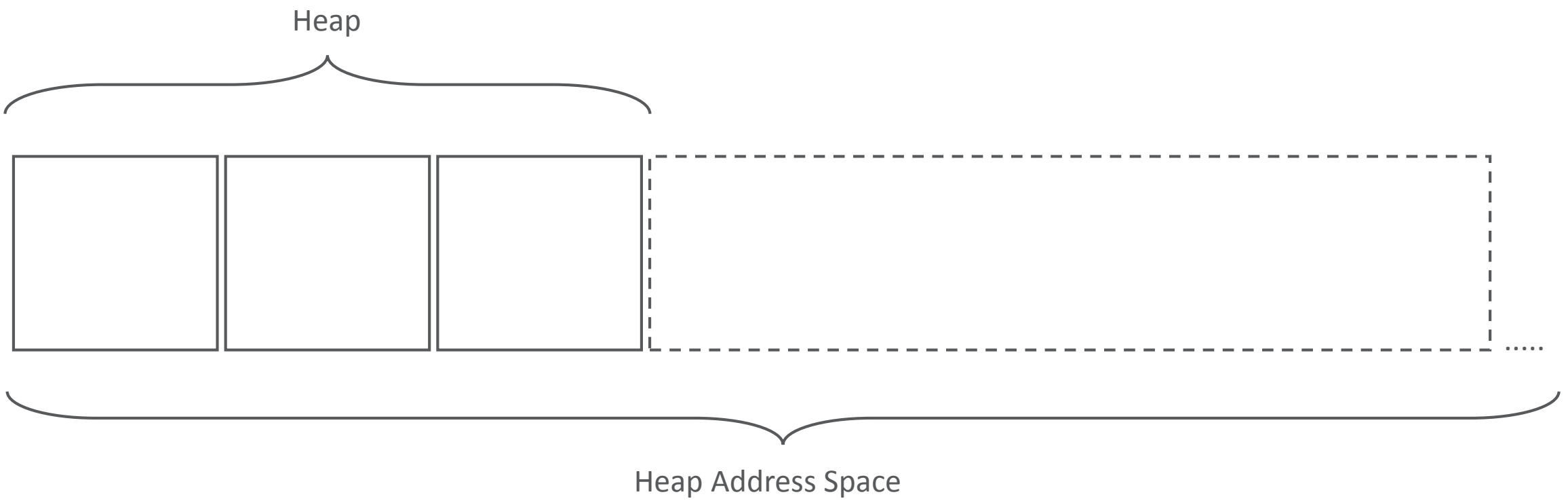


Relocation

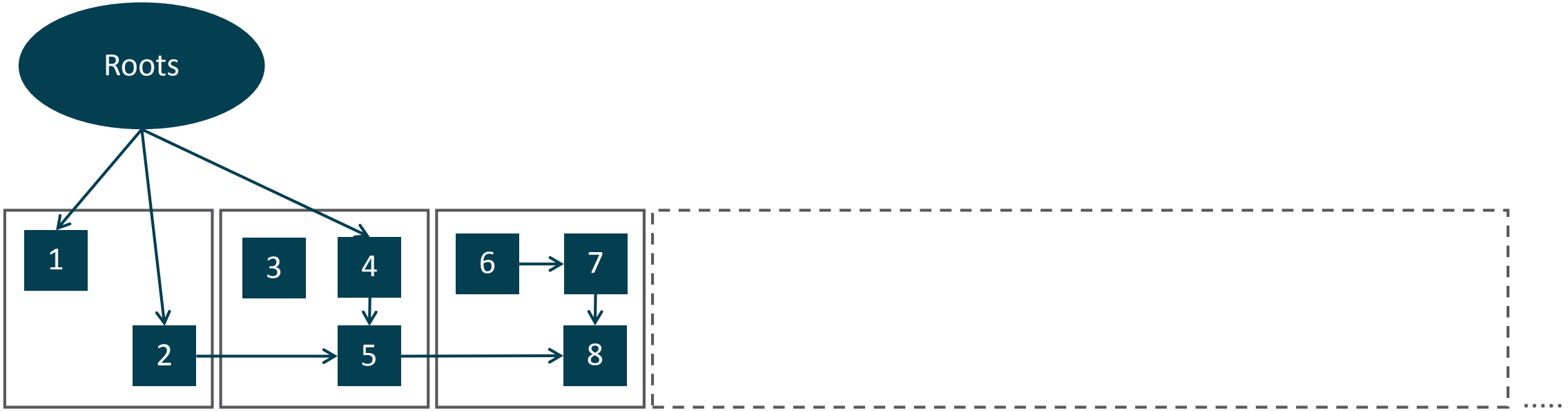


- Concurrent & Parallel
- Load barrier
  - Detects loads of object pointers pointing into the relocation set
  - Java threads help out with relocation if needed
- Off-heap forwarding tables
  - No forwarding information stored in old copies of objects
  - Important for immediate reuse of heap memory

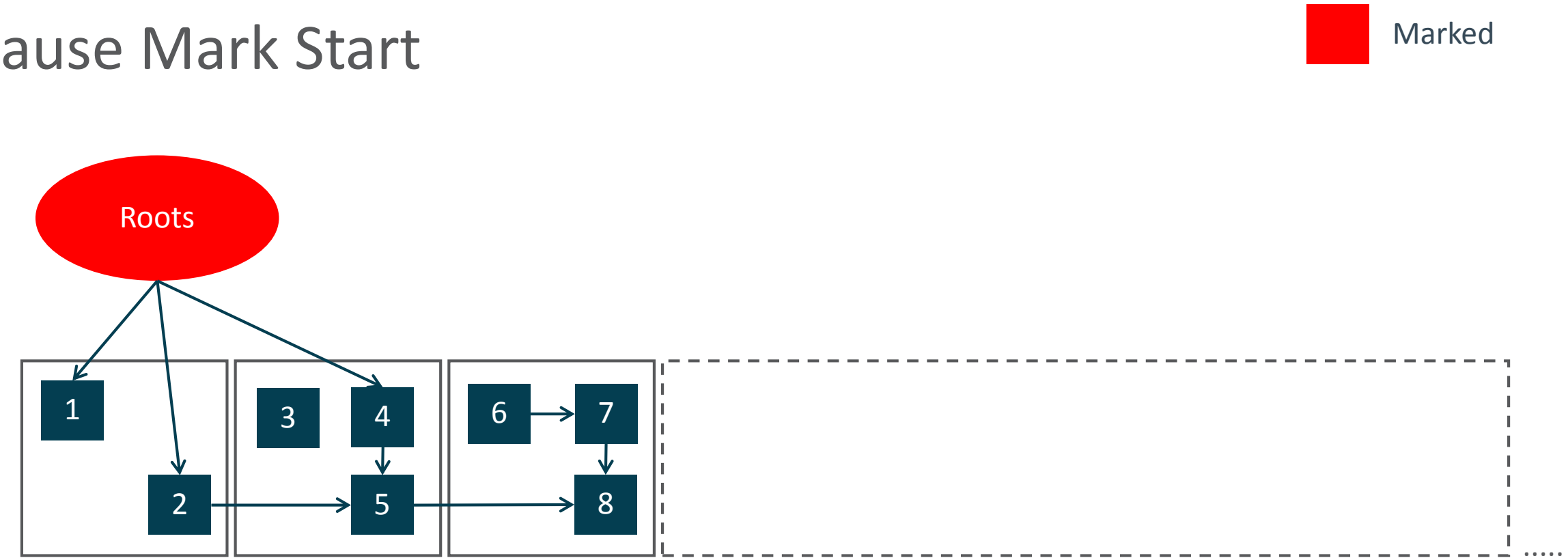
GC Cycle Example



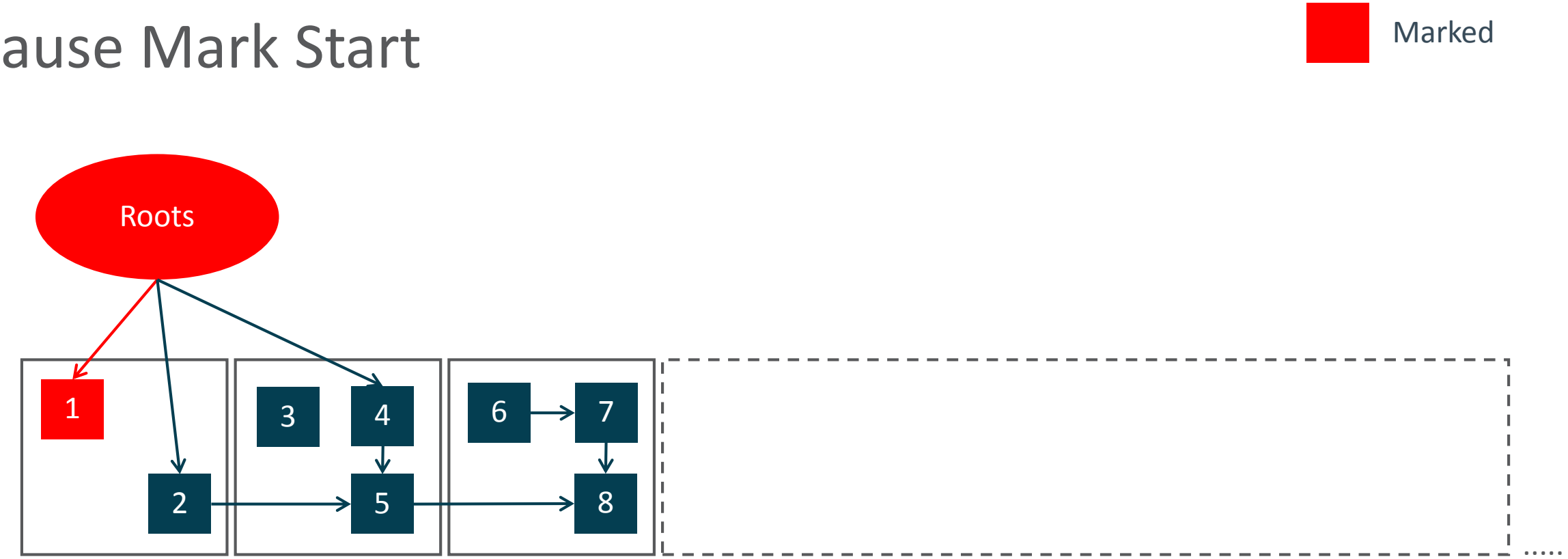
GC Cycle Example



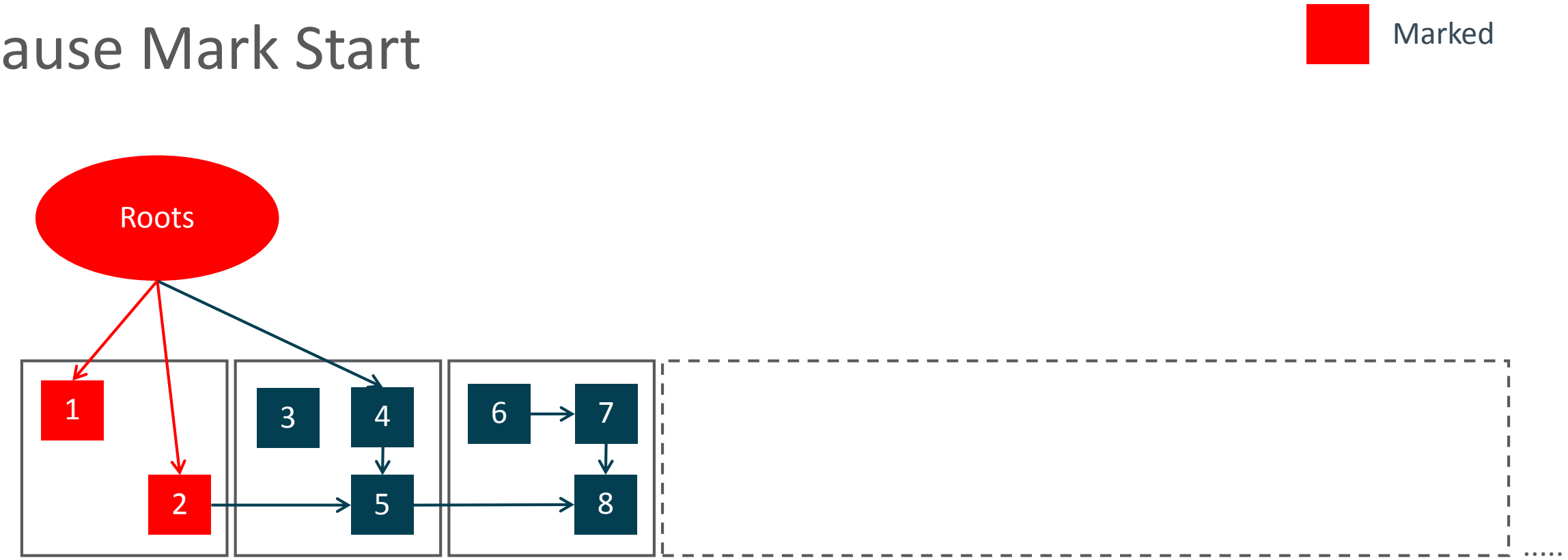
Pause Mark Start



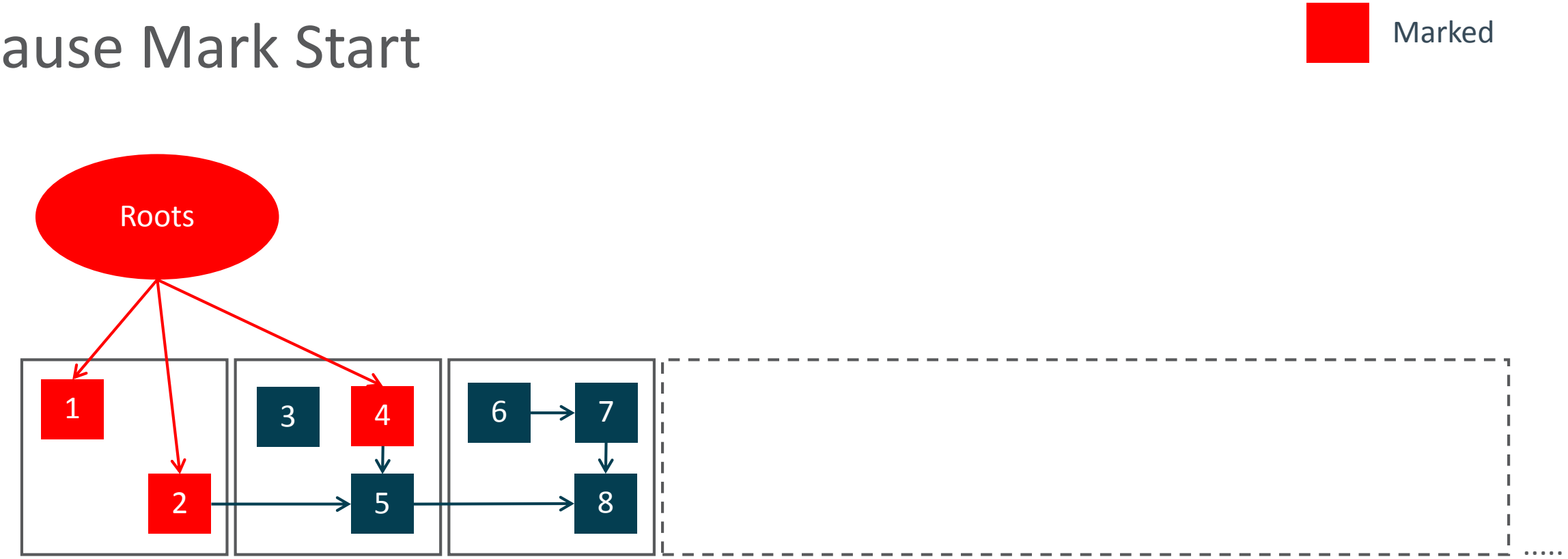
Pause Mark Start



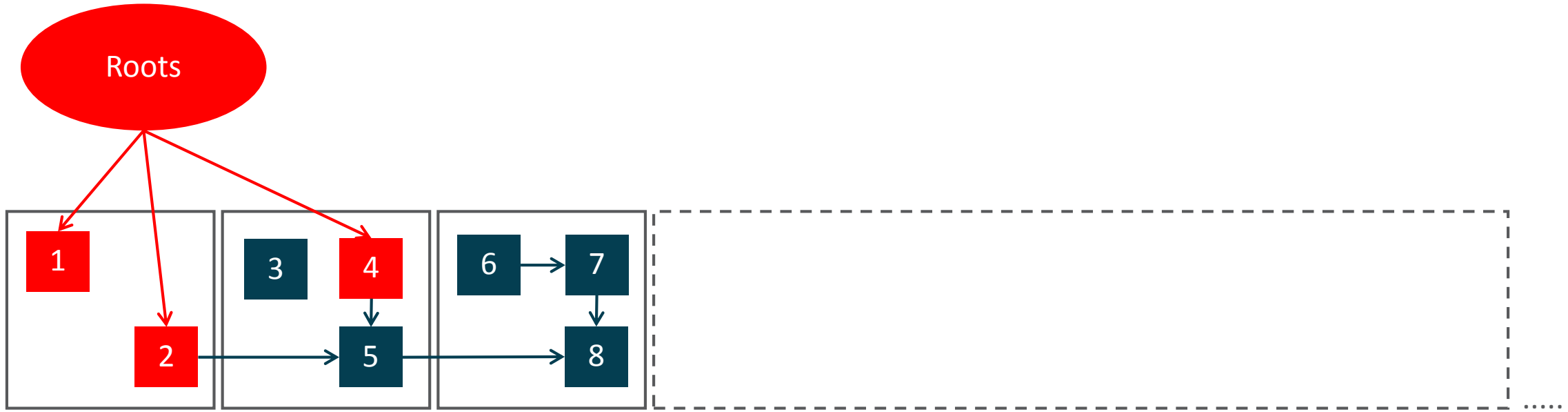
Pause Mark Start



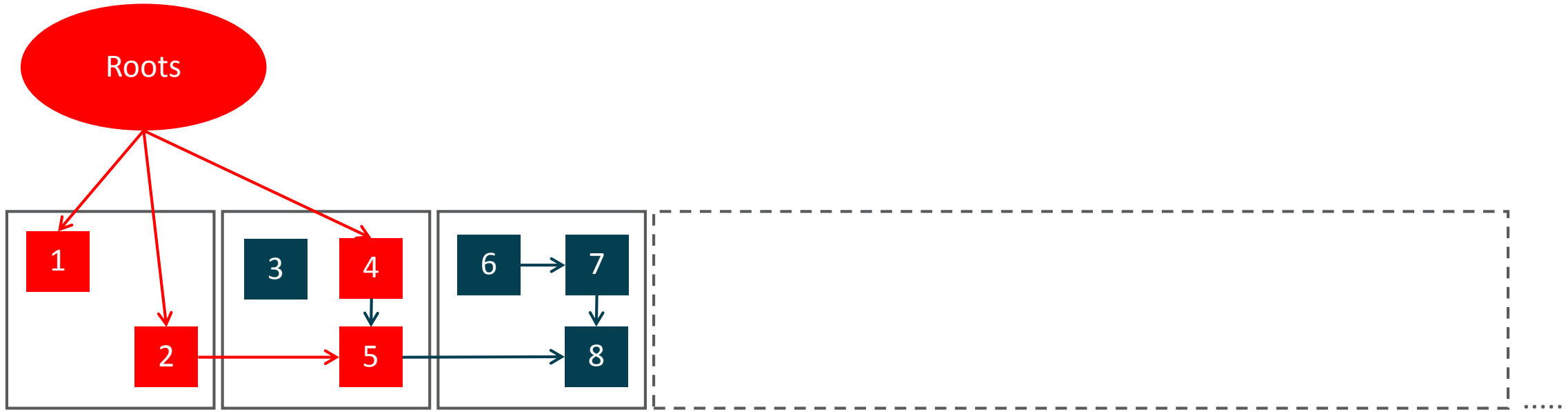
Pause Mark Start



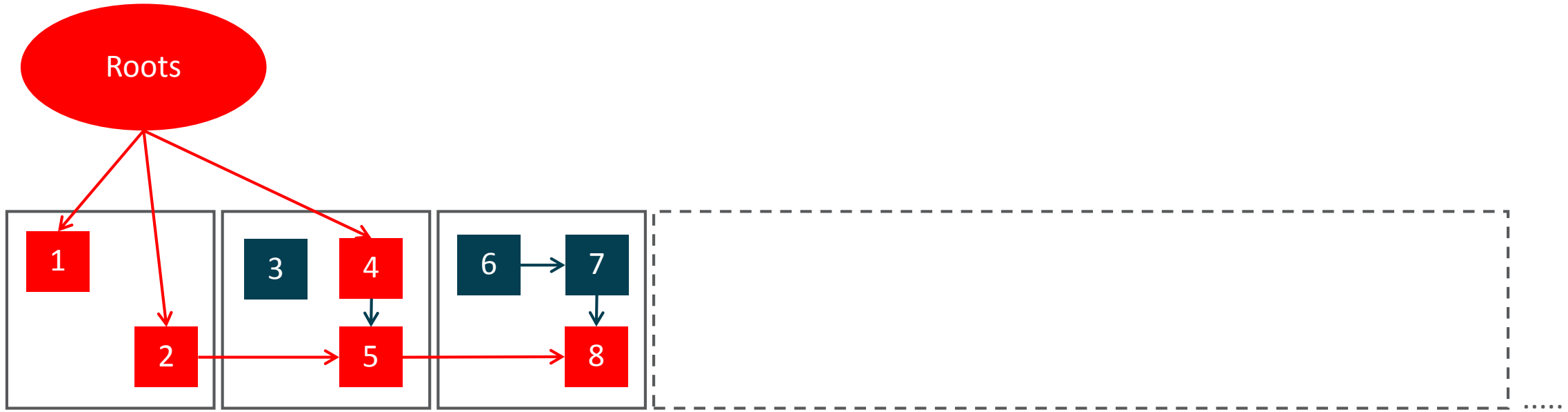
Concurrent Mark



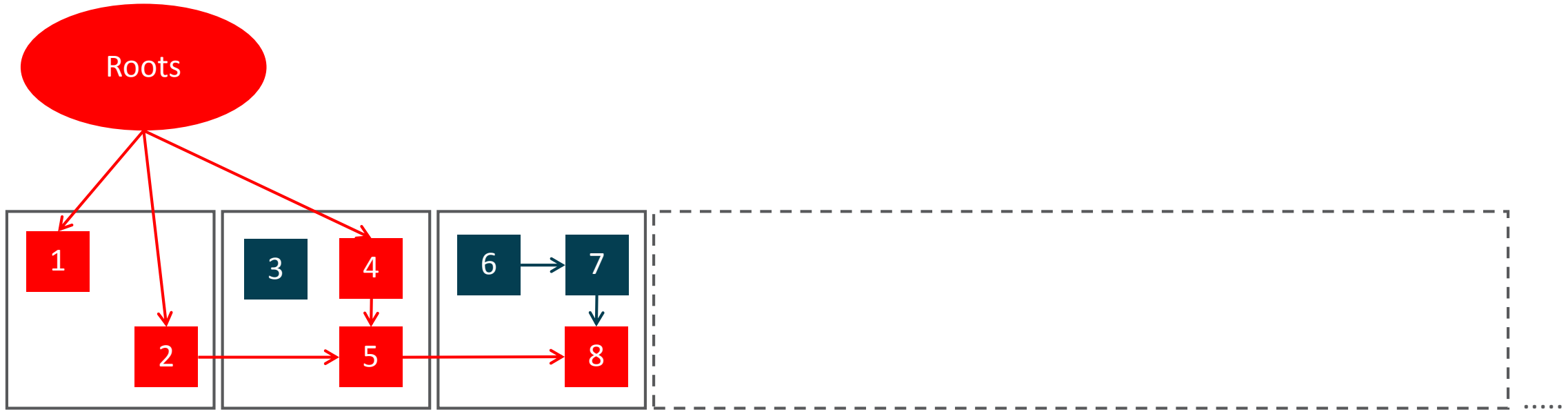
Concurrent Mark



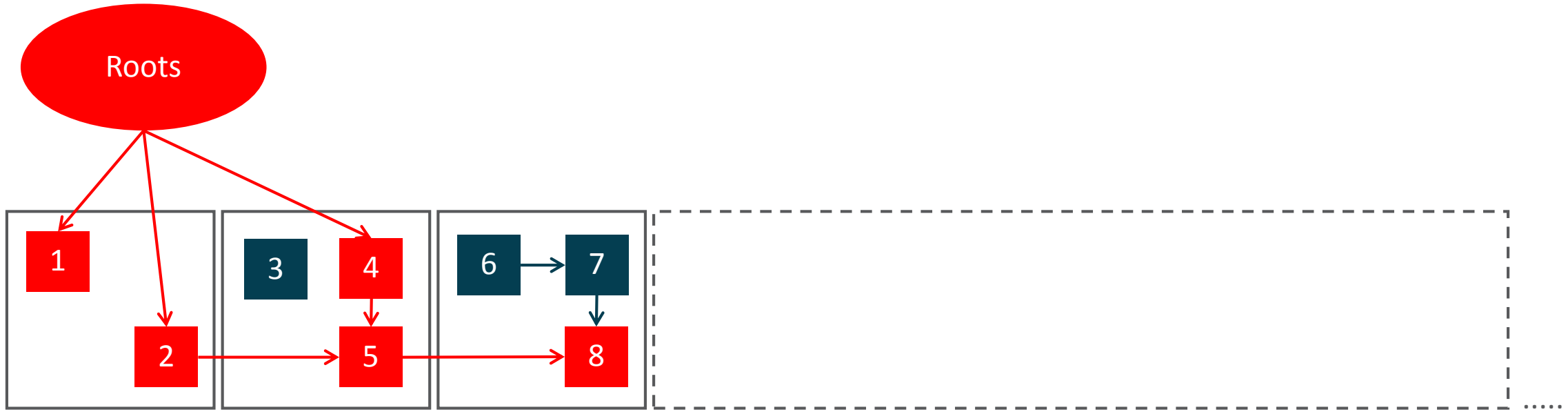
Concurrent Mark



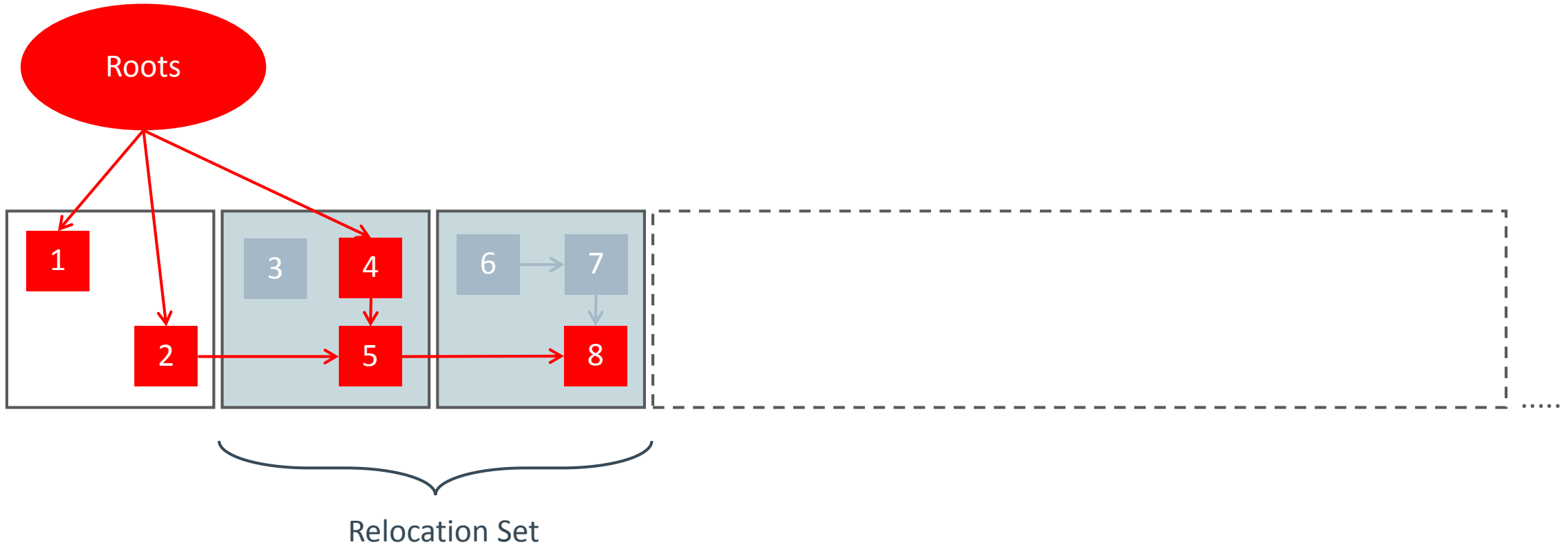
Concurrent Mark



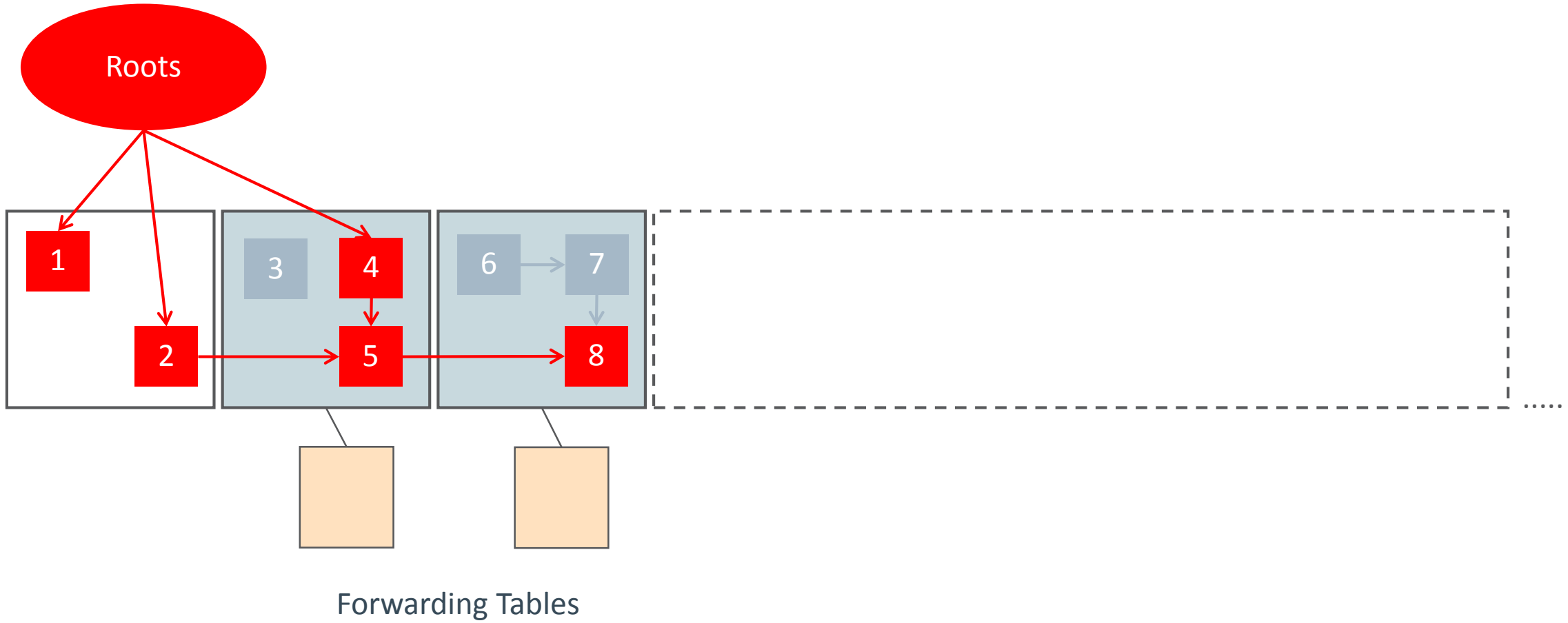
Pause Mark End



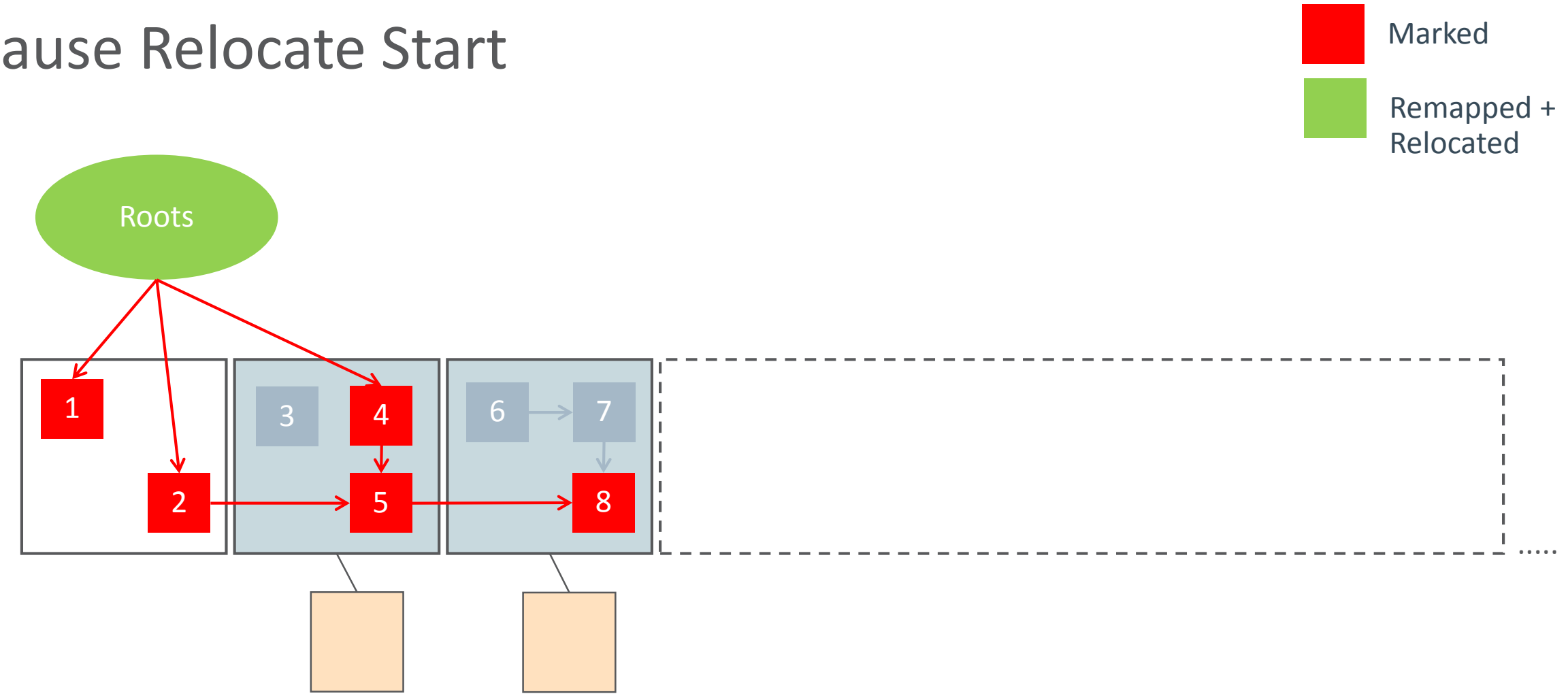
Concurrent Prepare for Relocate



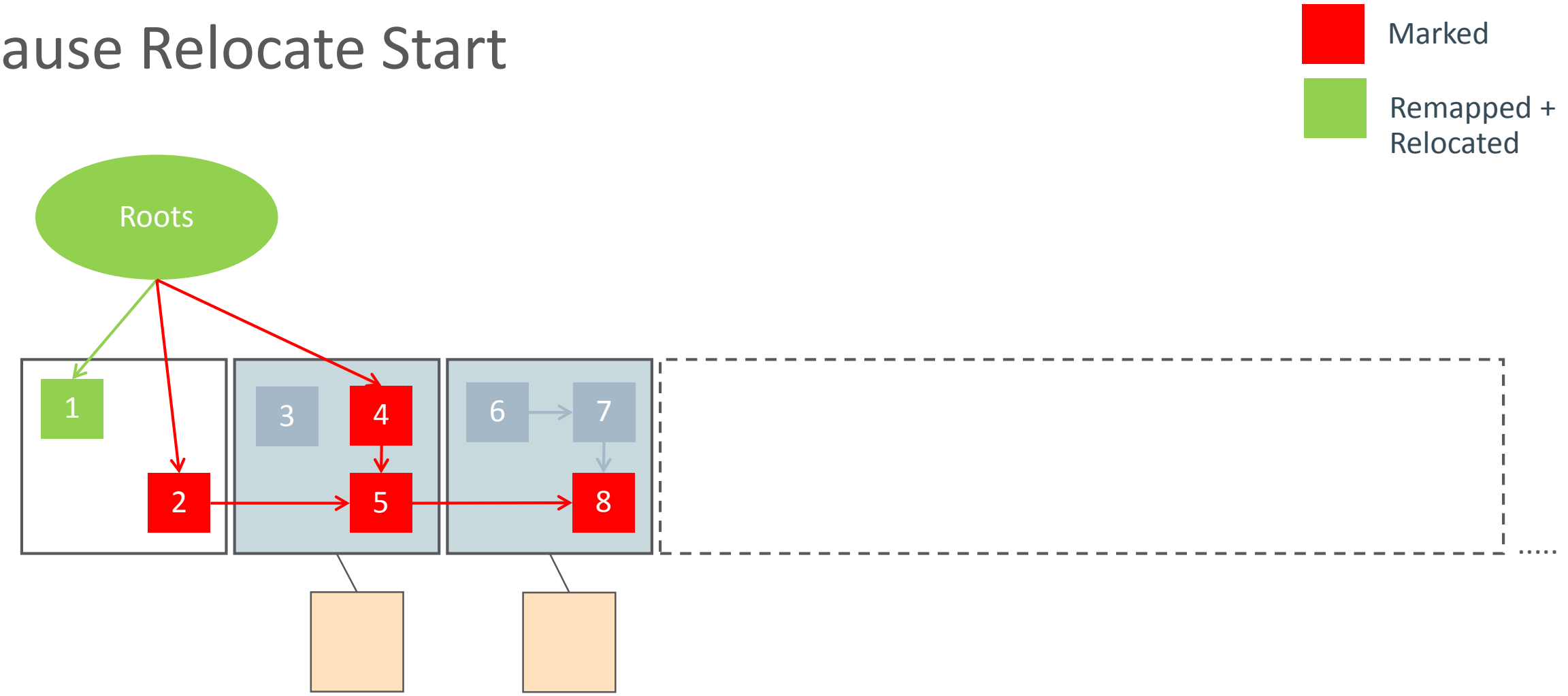
Concurrent Prepare for Relocate



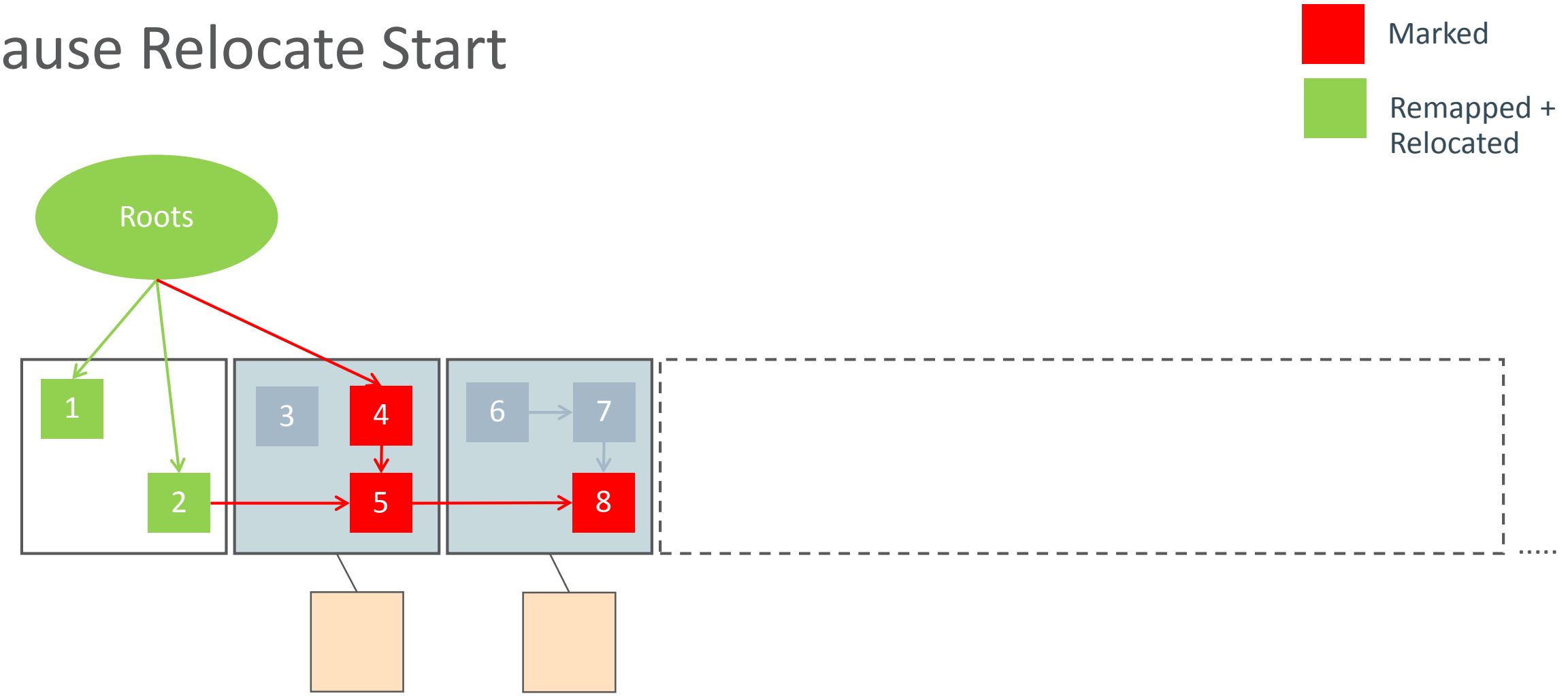
Pause Relocate Start



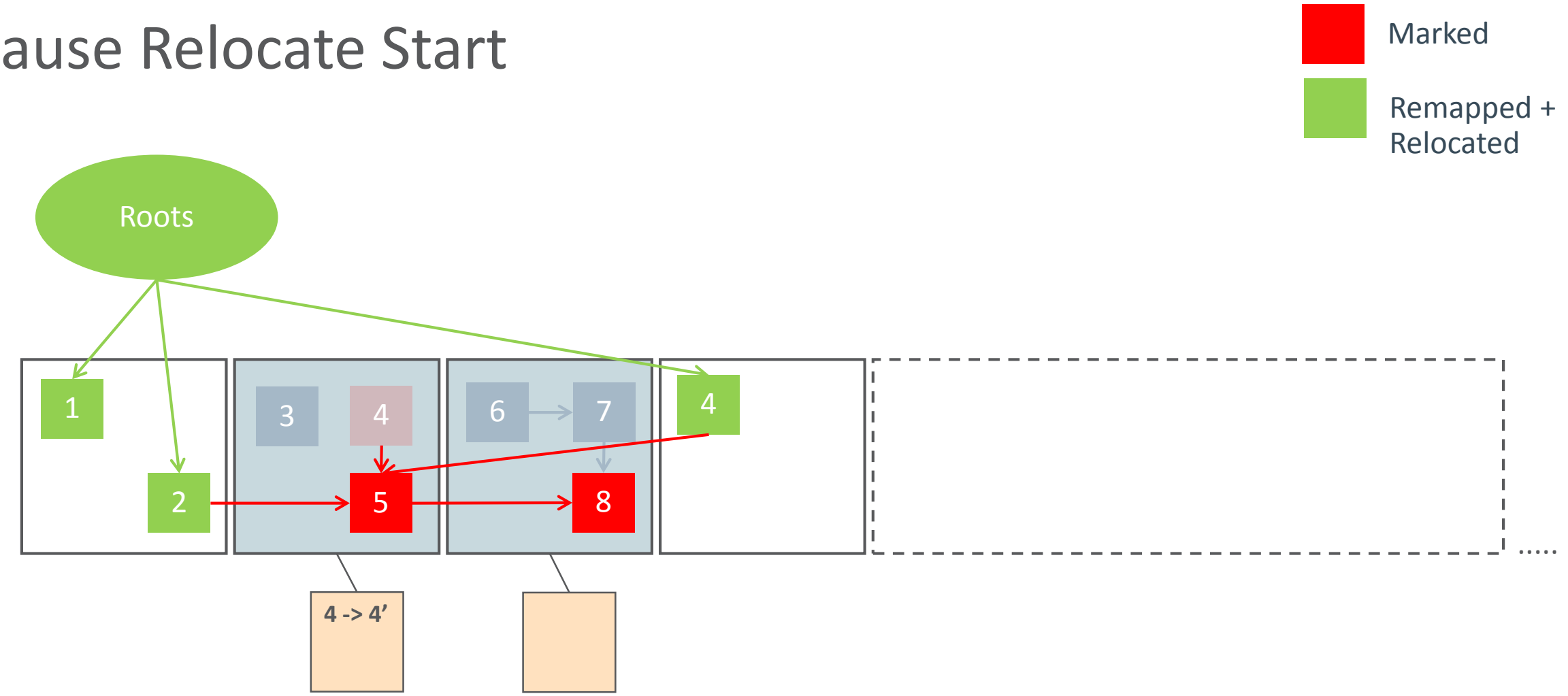
Pause Relocate Start



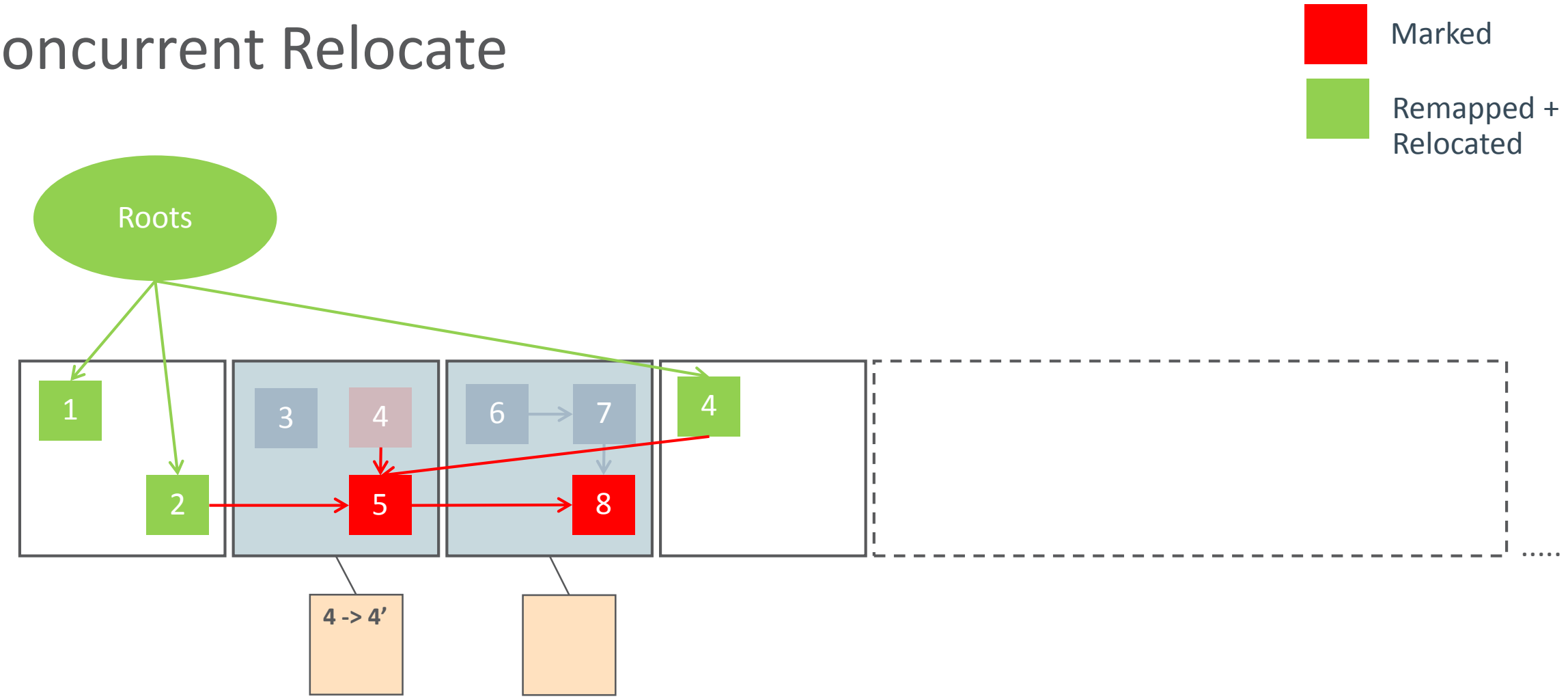
Pause Relocate Start



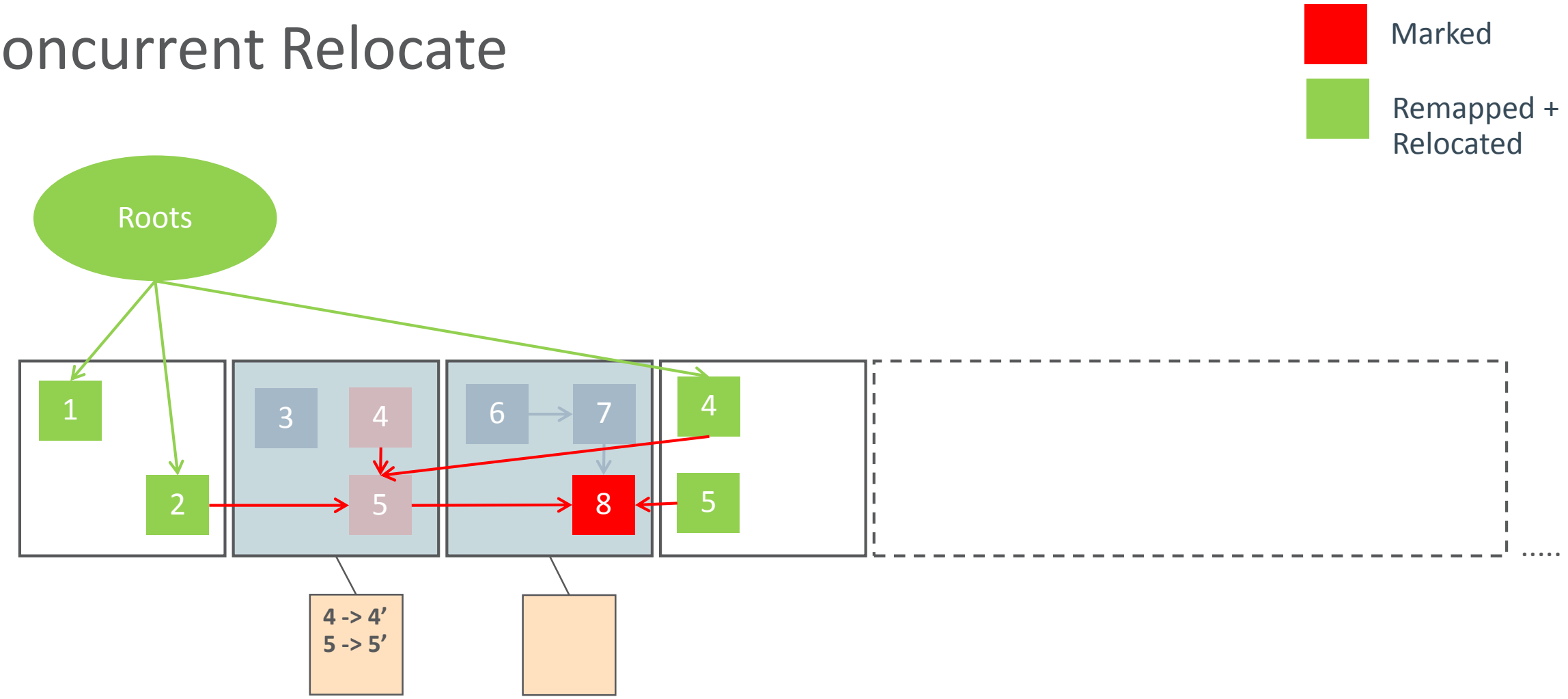
Pause Relocate Start



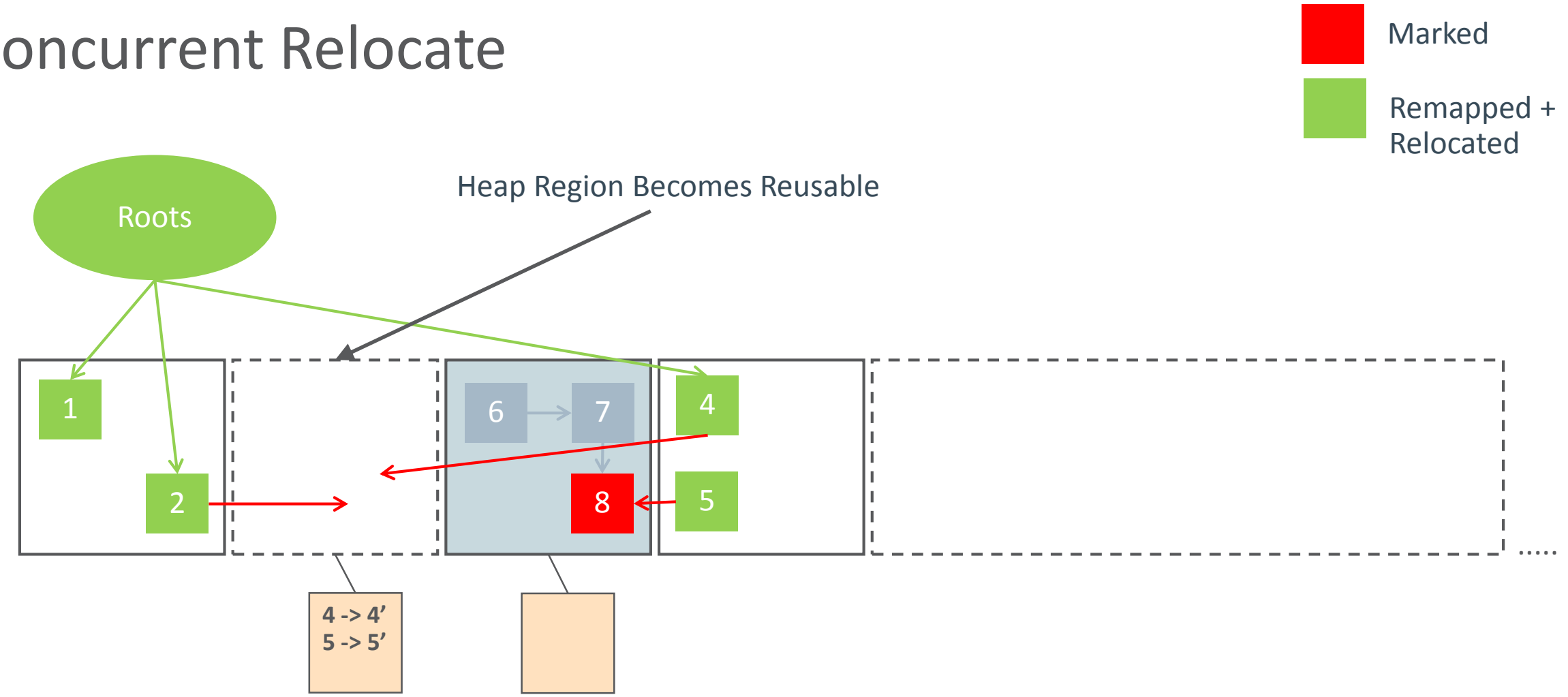
Concurrent Relocate



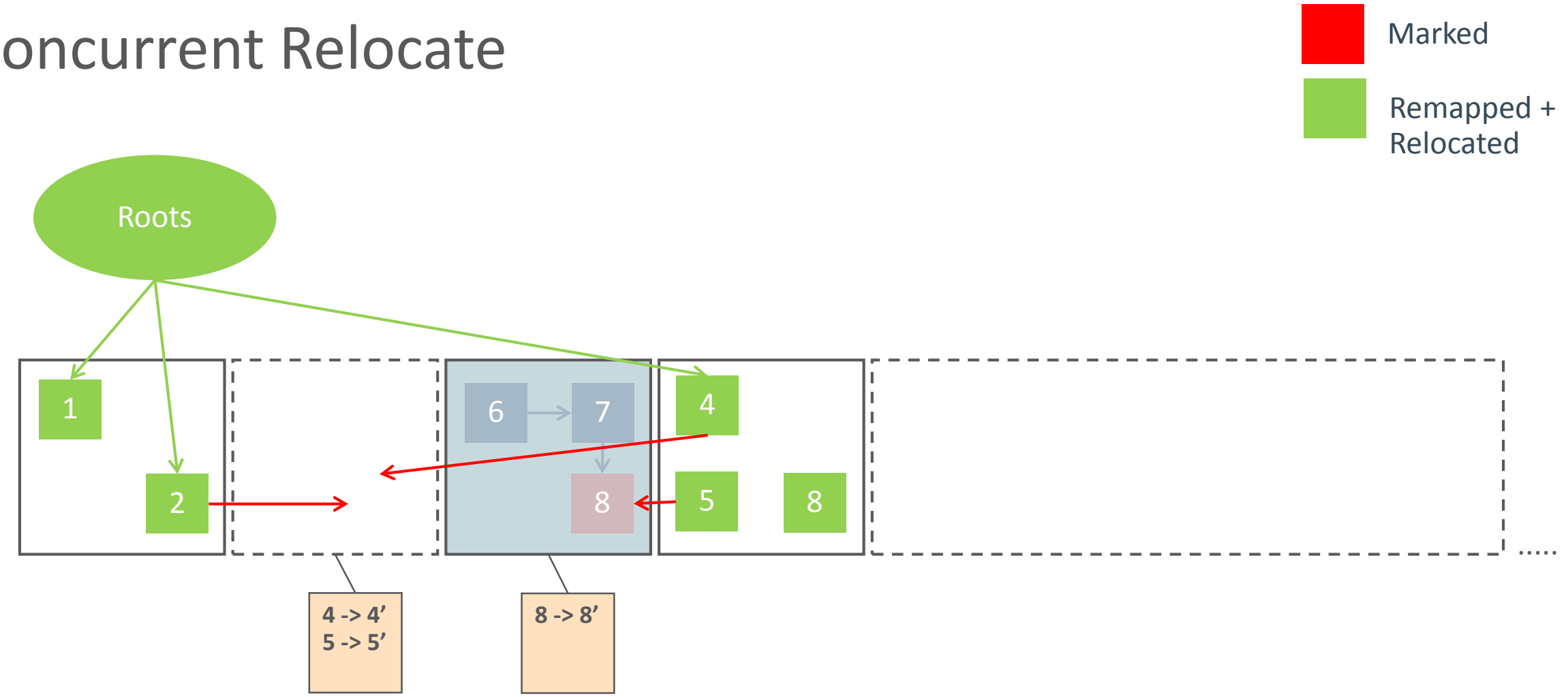
Concurrent Relocate



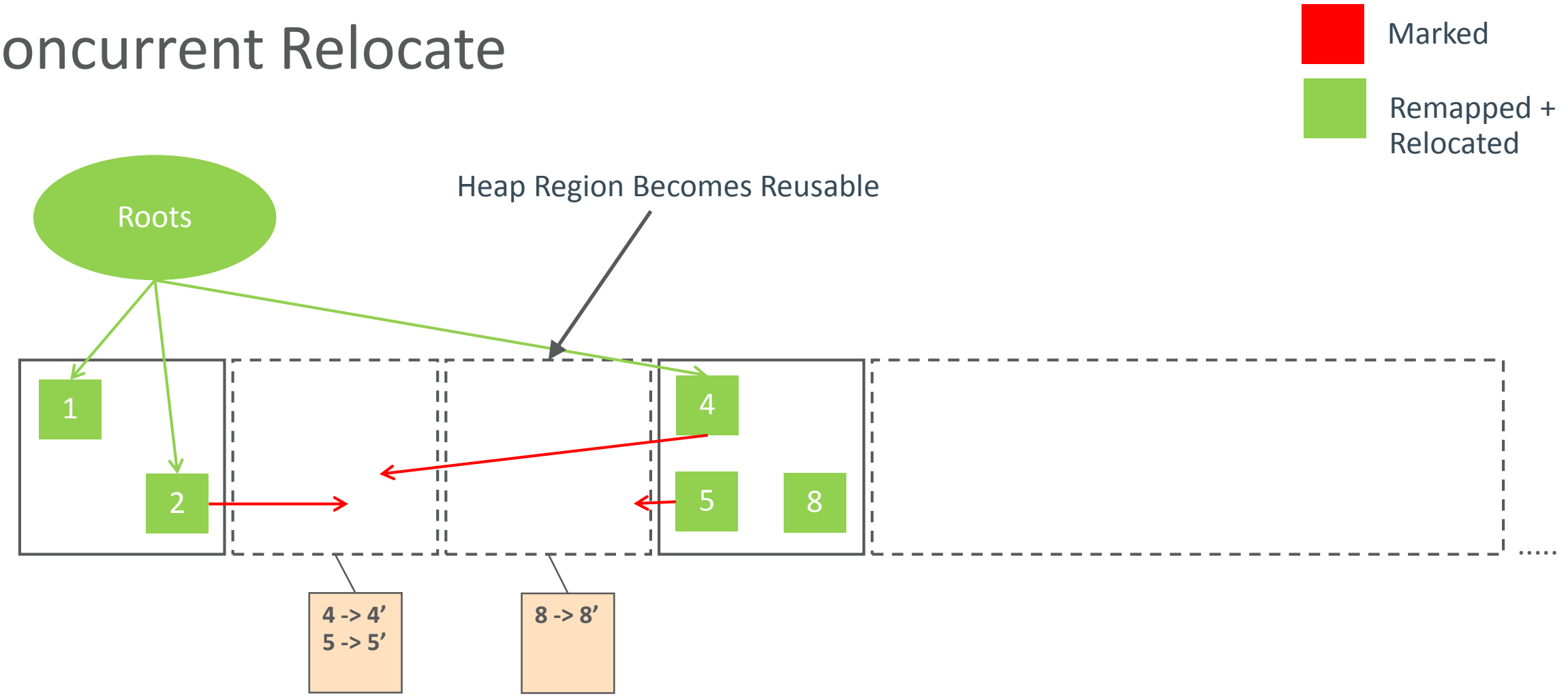
Concurrent Relocate



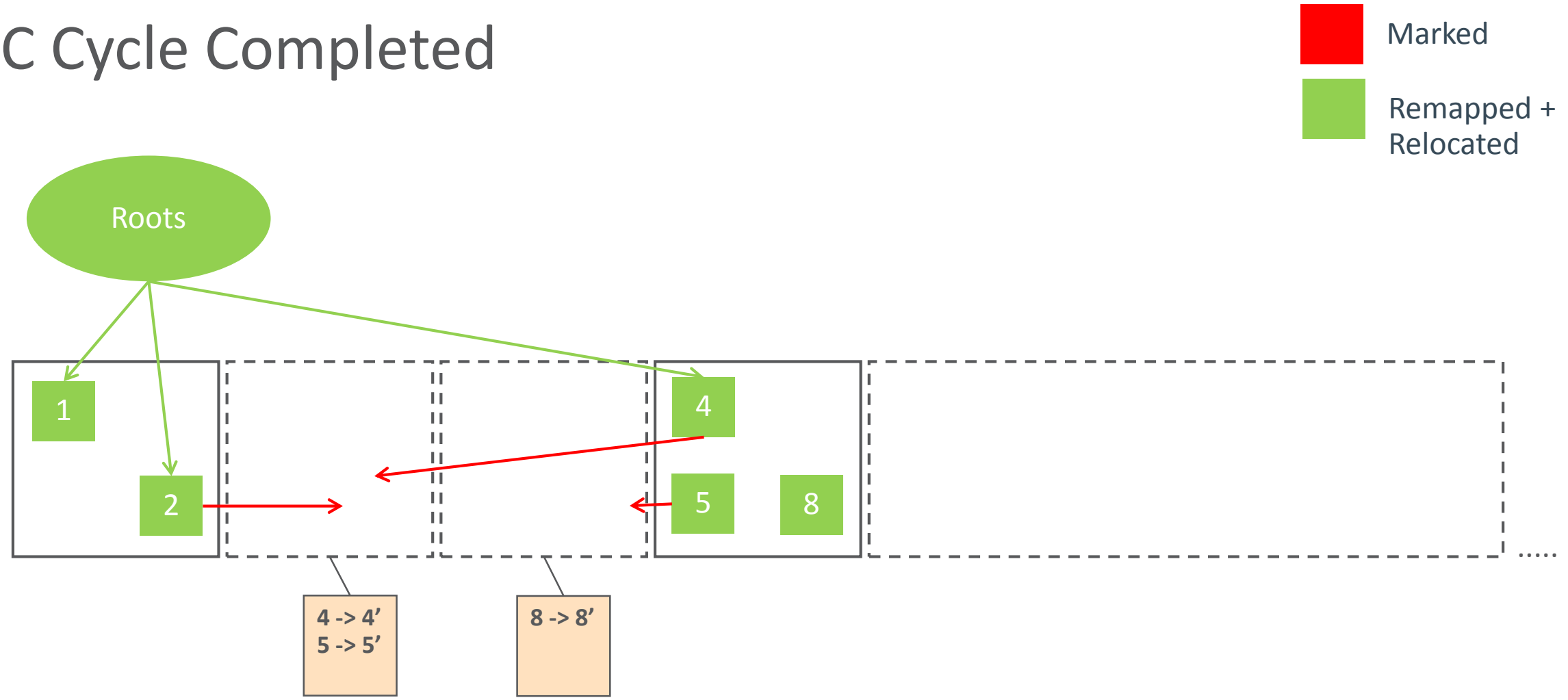
Concurrent Relocate



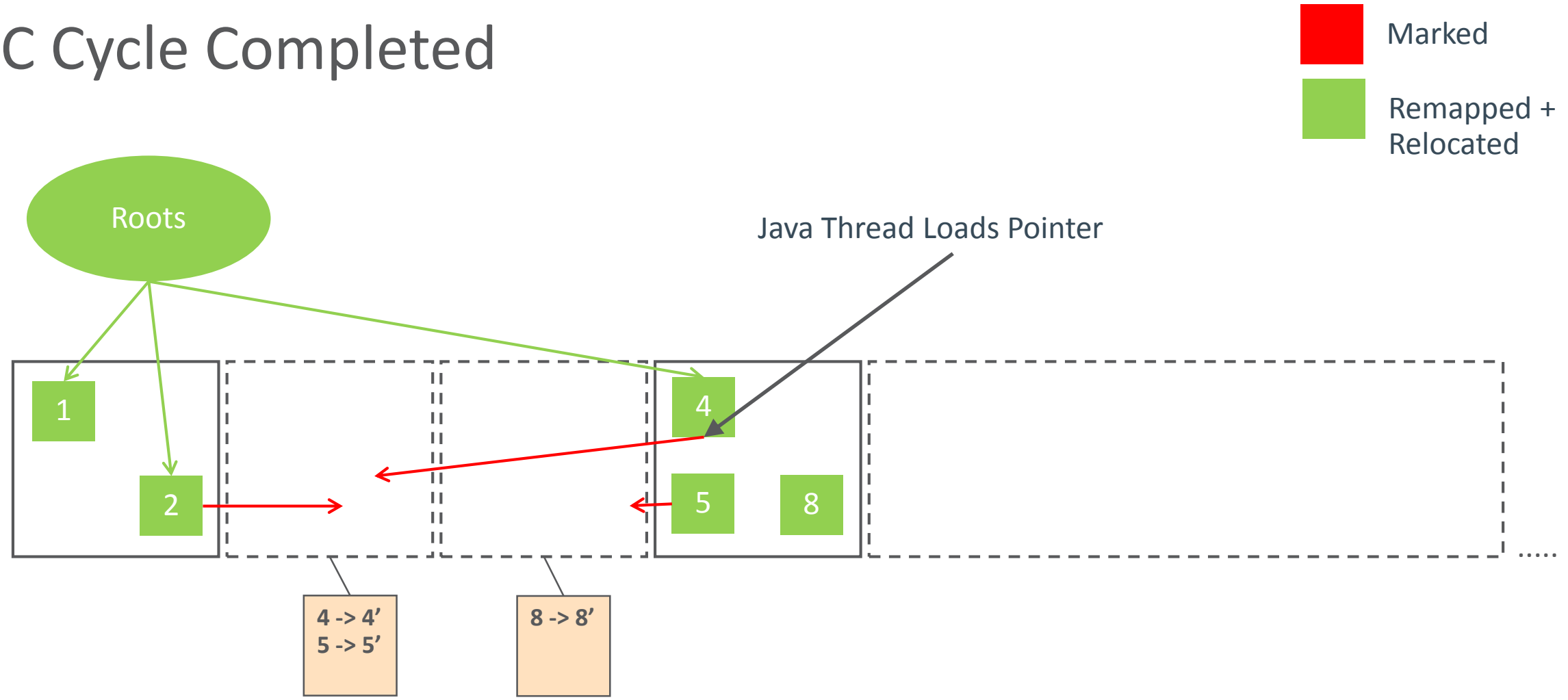
Concurrent Relocate



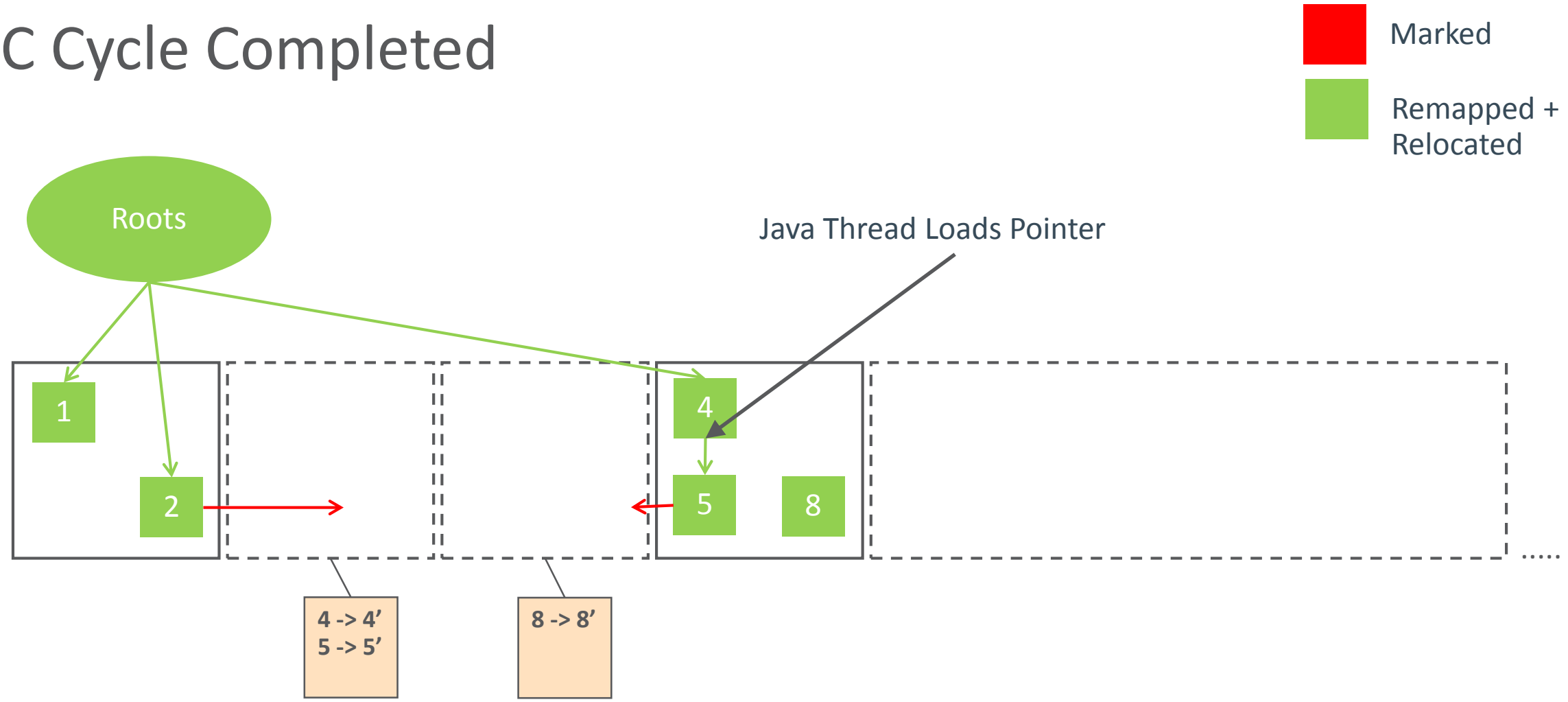
GC Cycle Completed



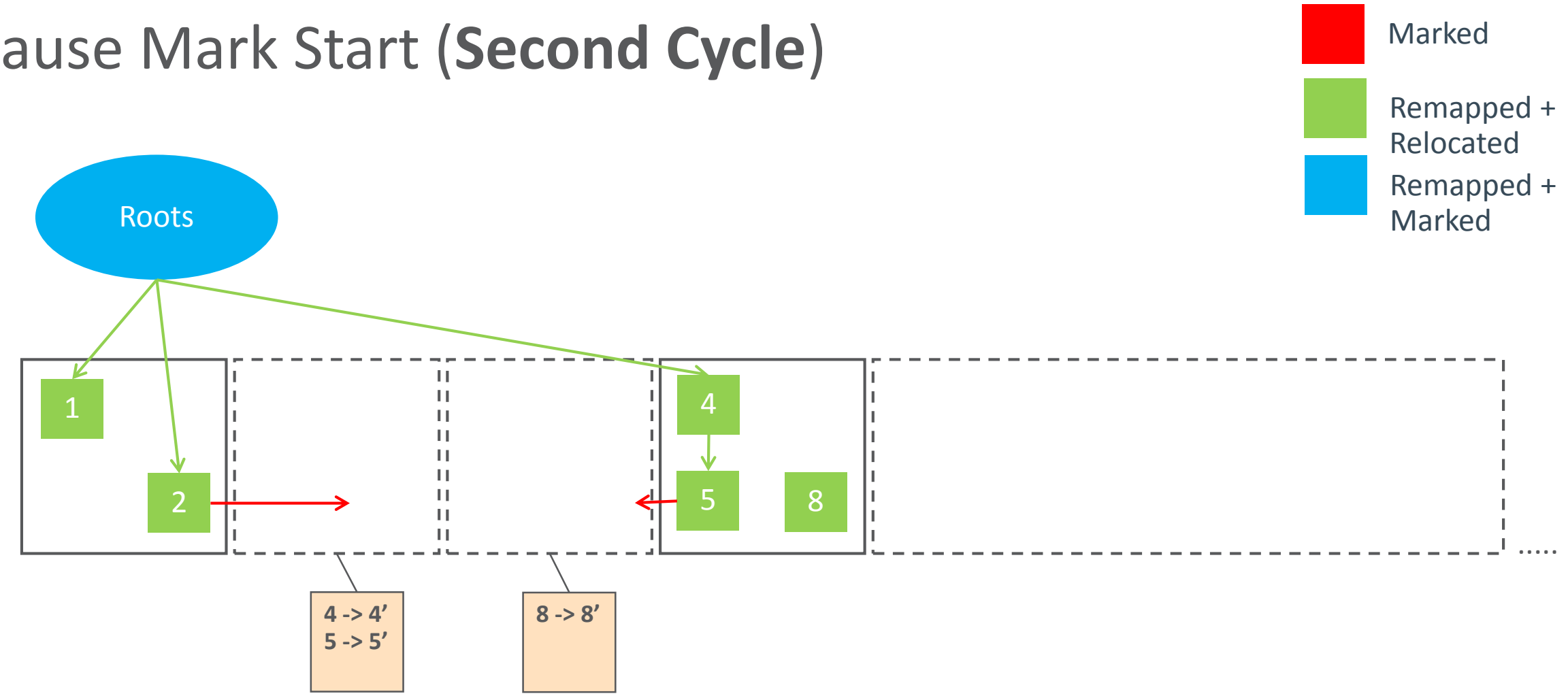
GC Cycle Completed



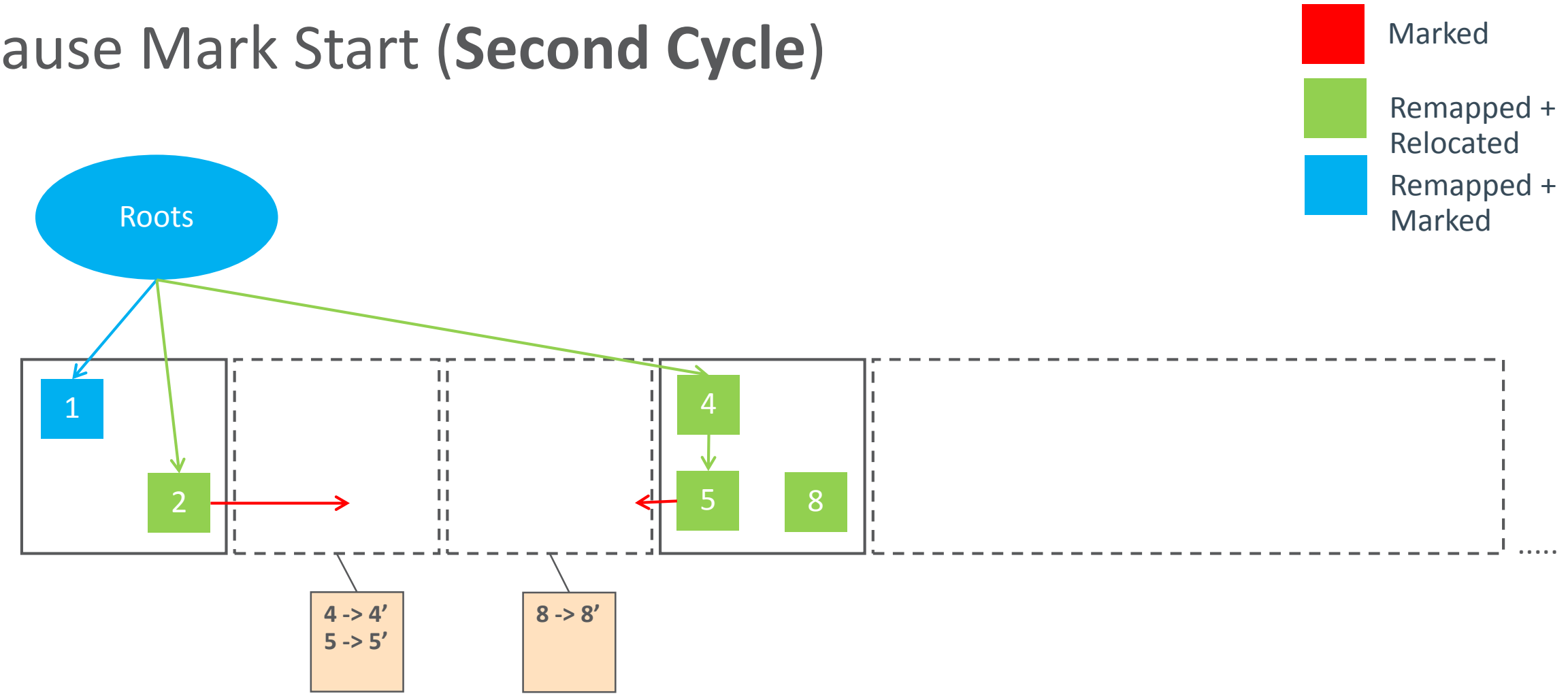
GC Cycle Completed



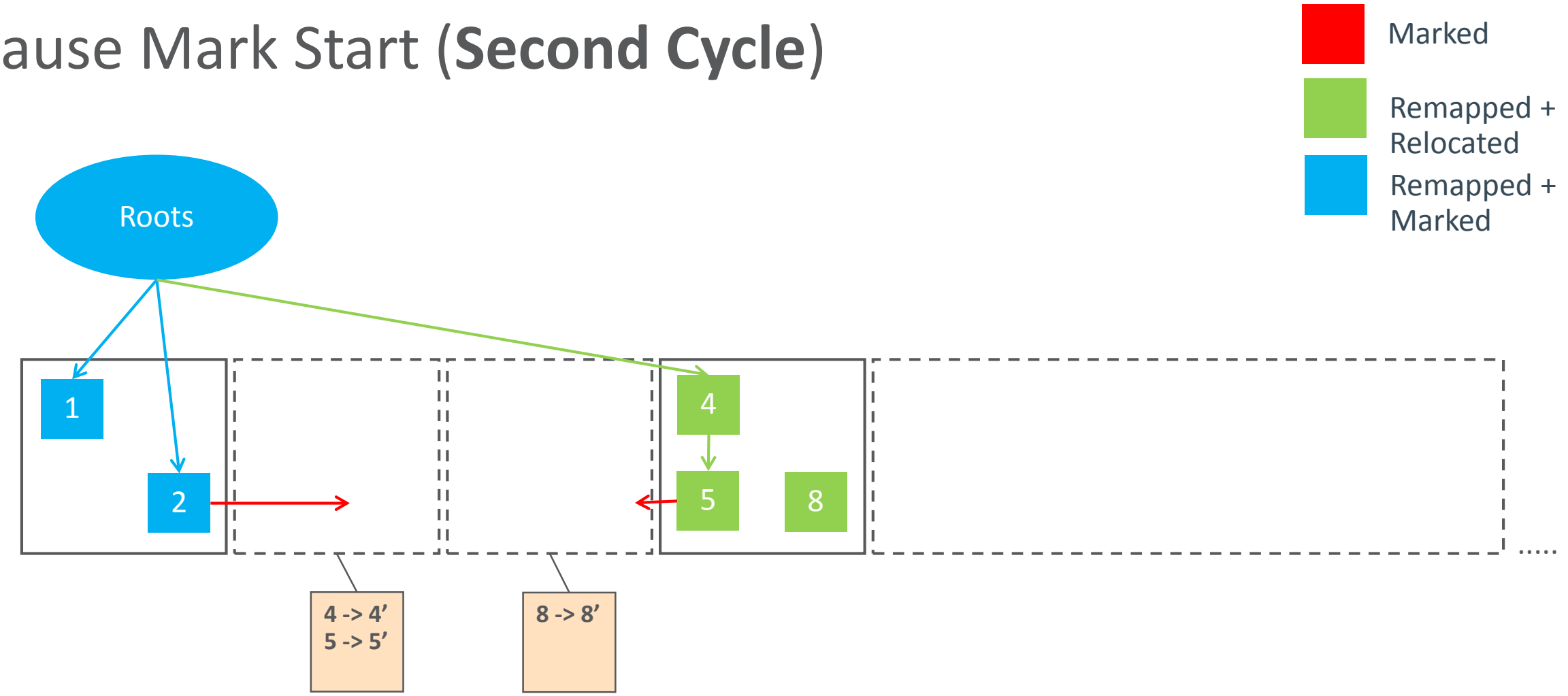
Pause Mark Start (**Second Cycle**)



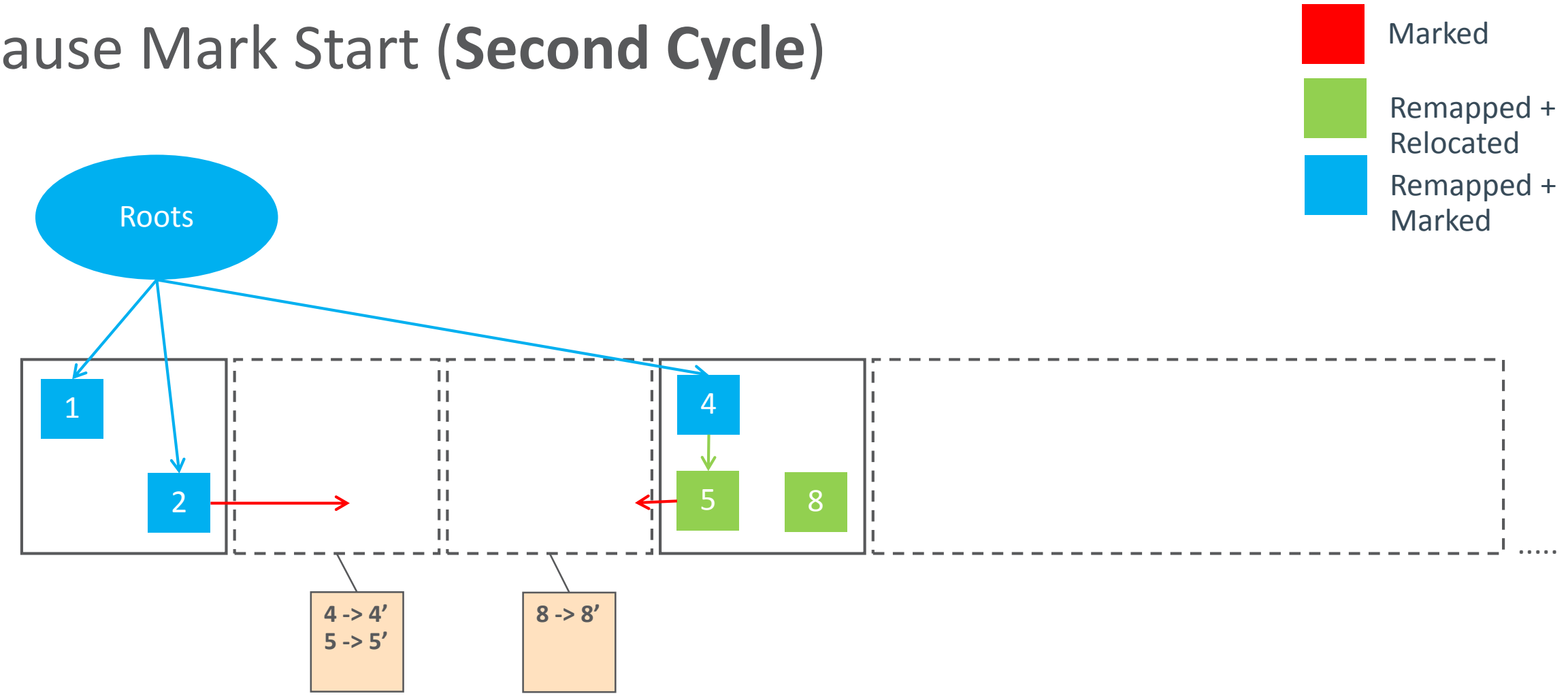
Pause Mark Start (**Second Cycle**)



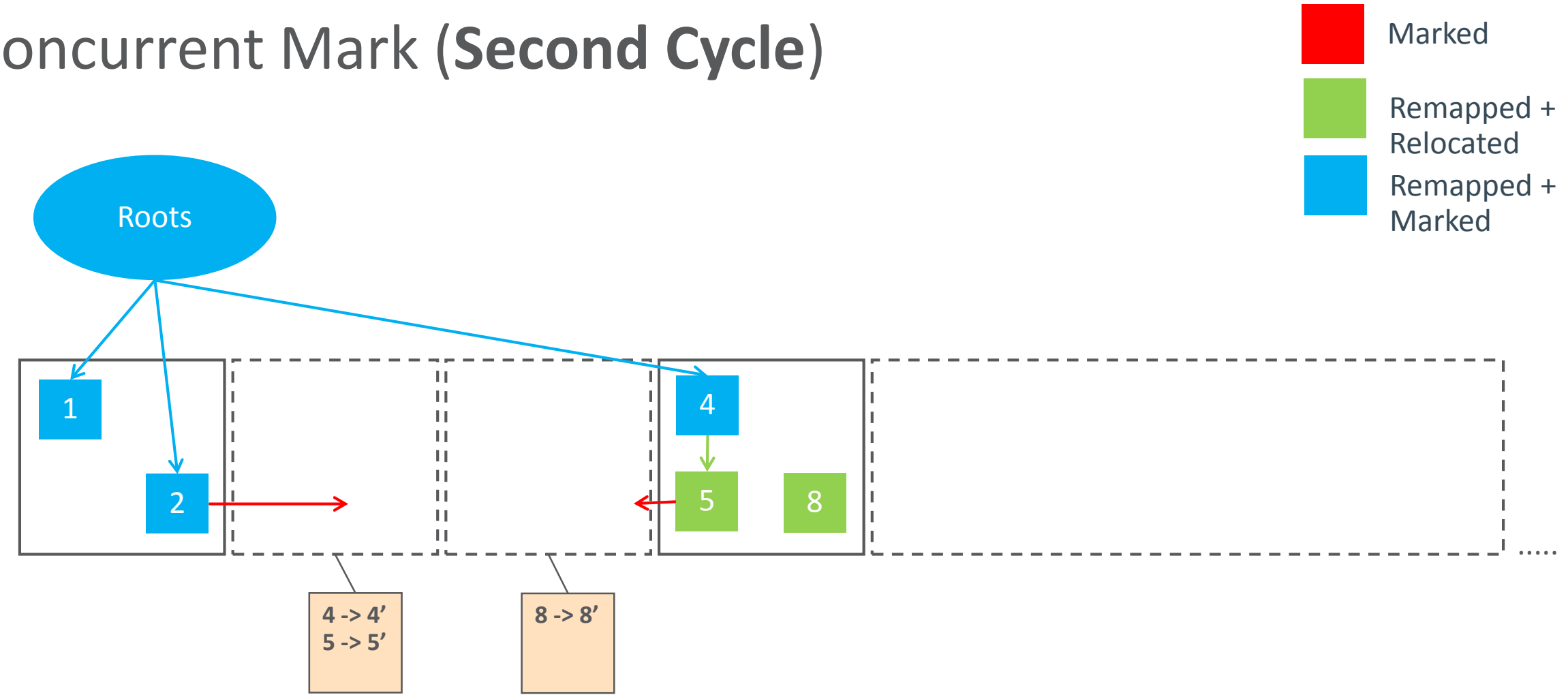
Pause Mark Start (**Second Cycle**)



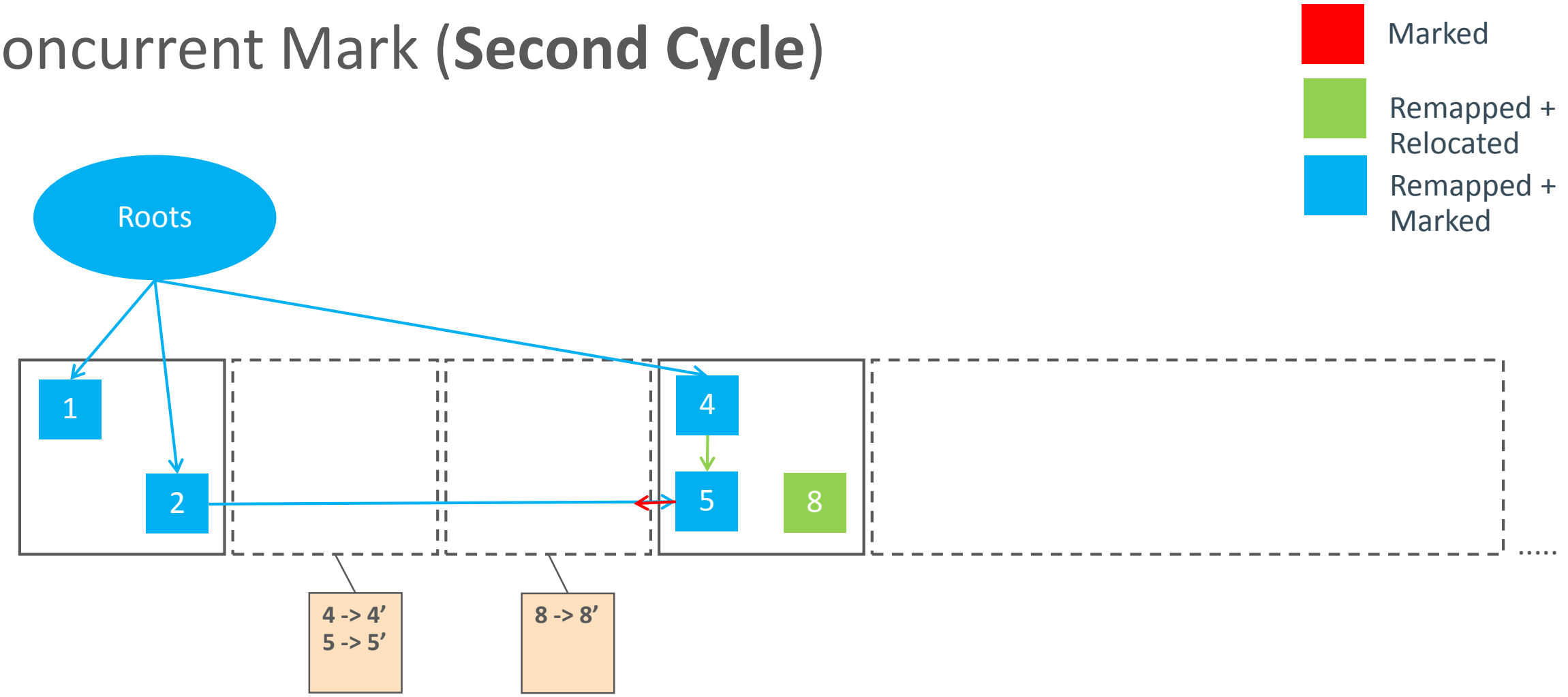
Pause Mark Start (**Second Cycle**)



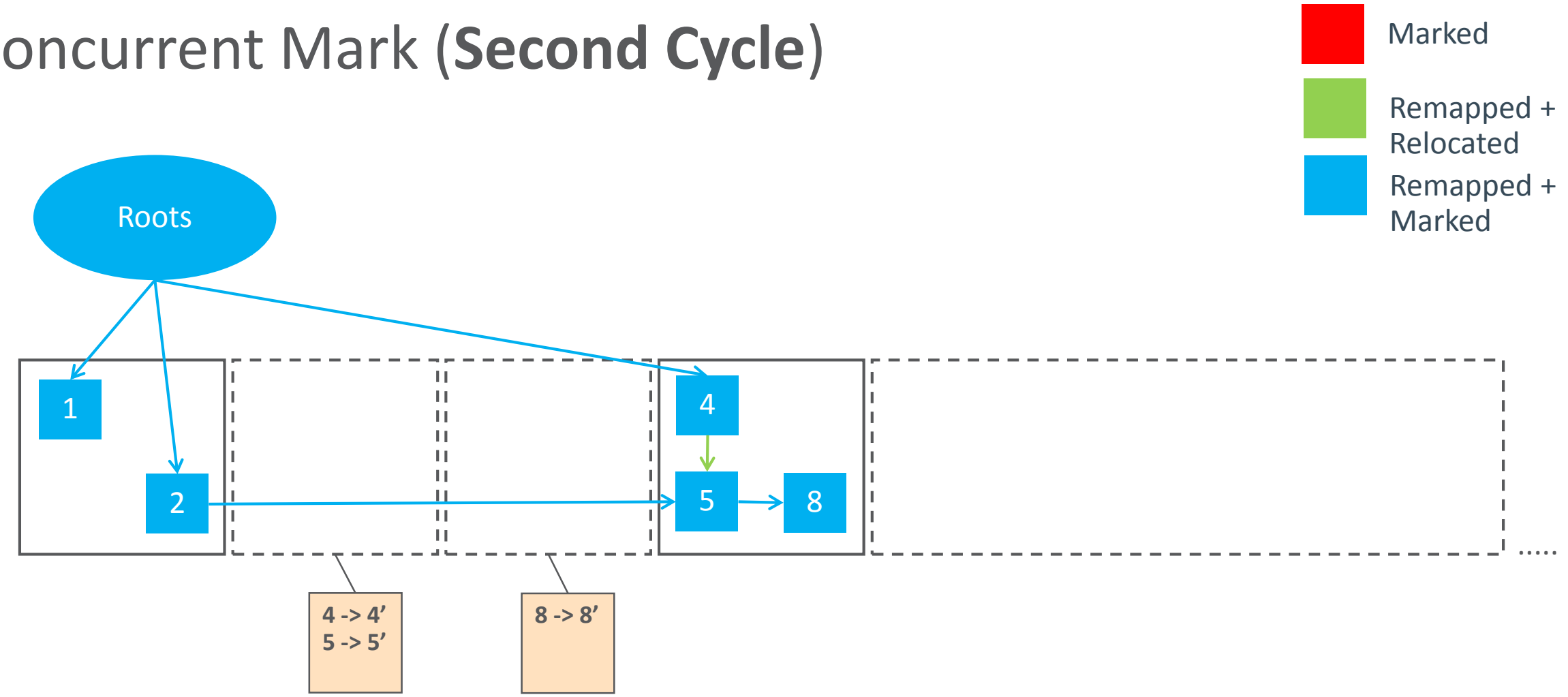
Concurrent Mark (**Second Cycle**)



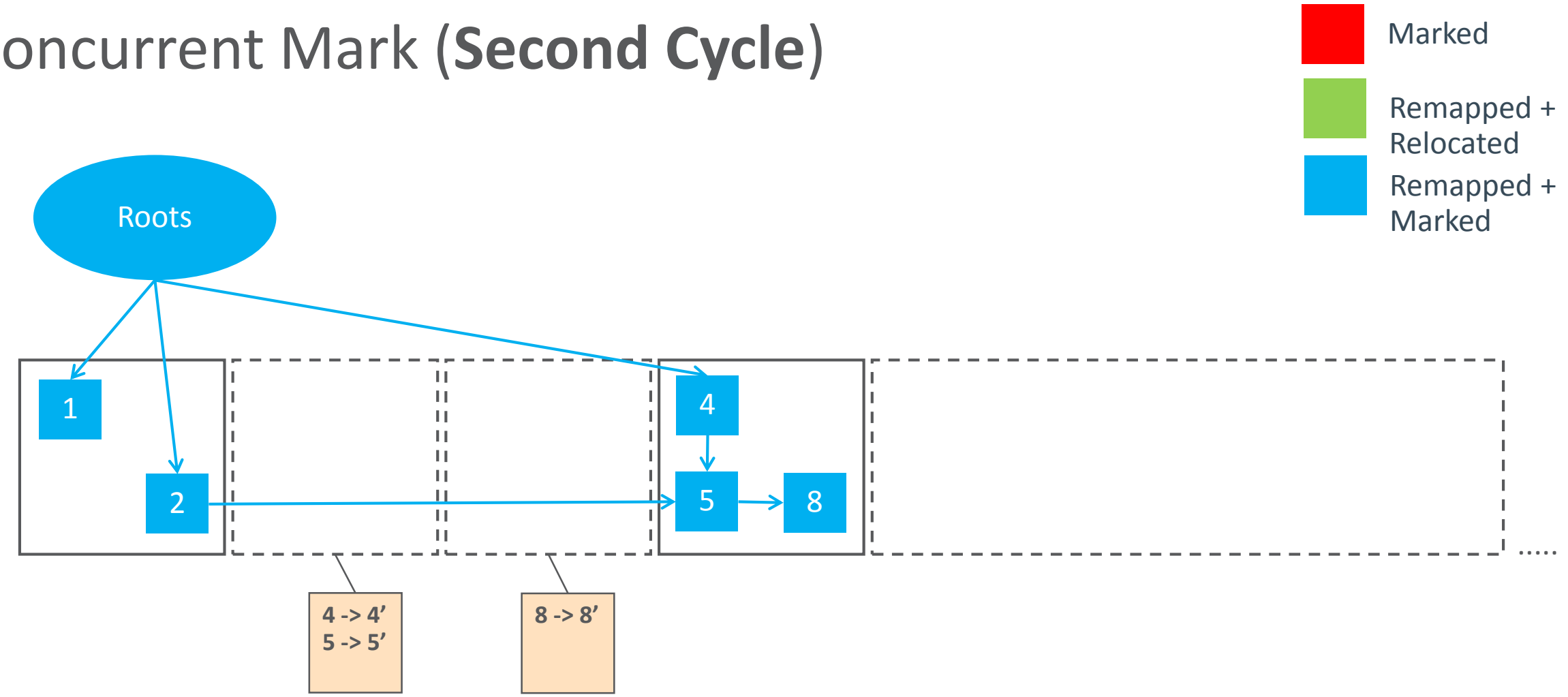
Concurrent Mark (**Second Cycle**)



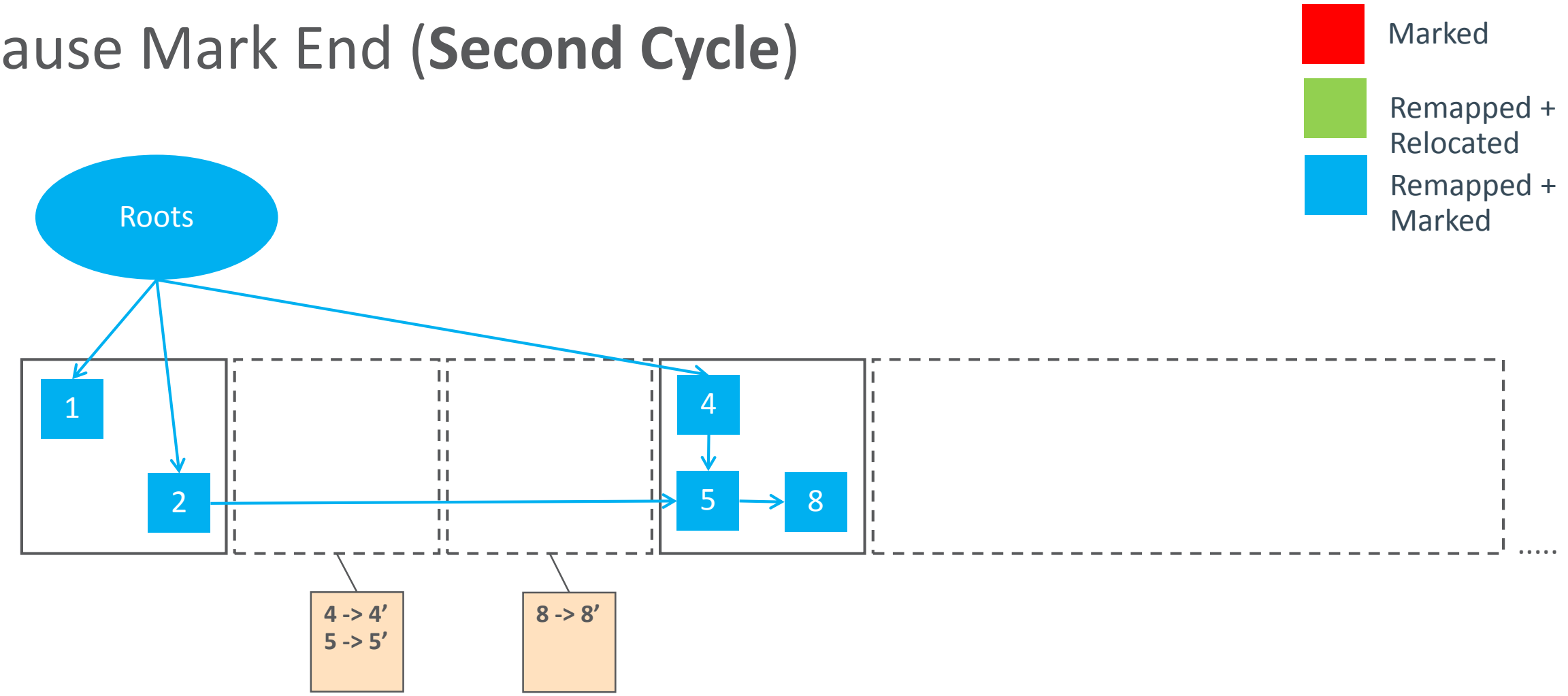
Concurrent Mark (**Second Cycle**)



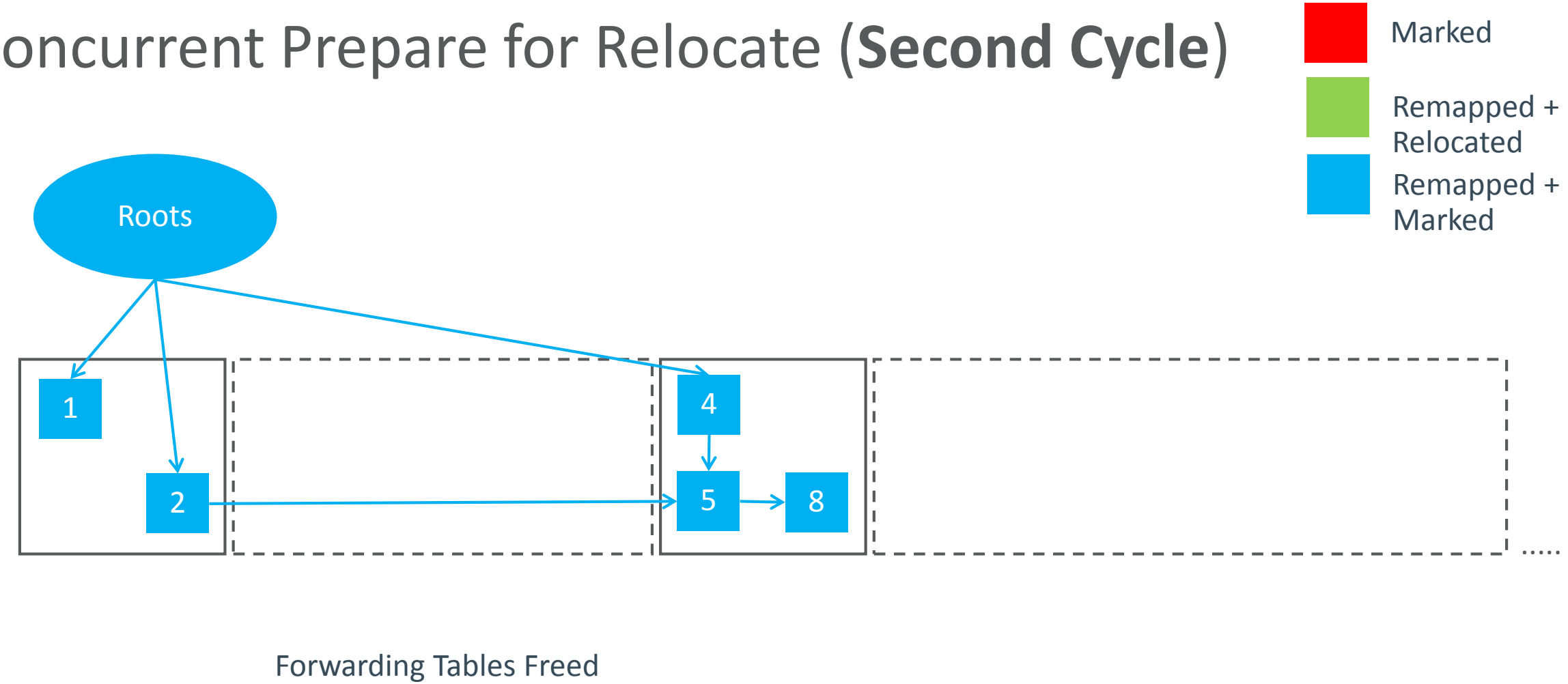
Concurrent Mark (**Second Cycle**)



Pause Mark End (**Second Cycle**)

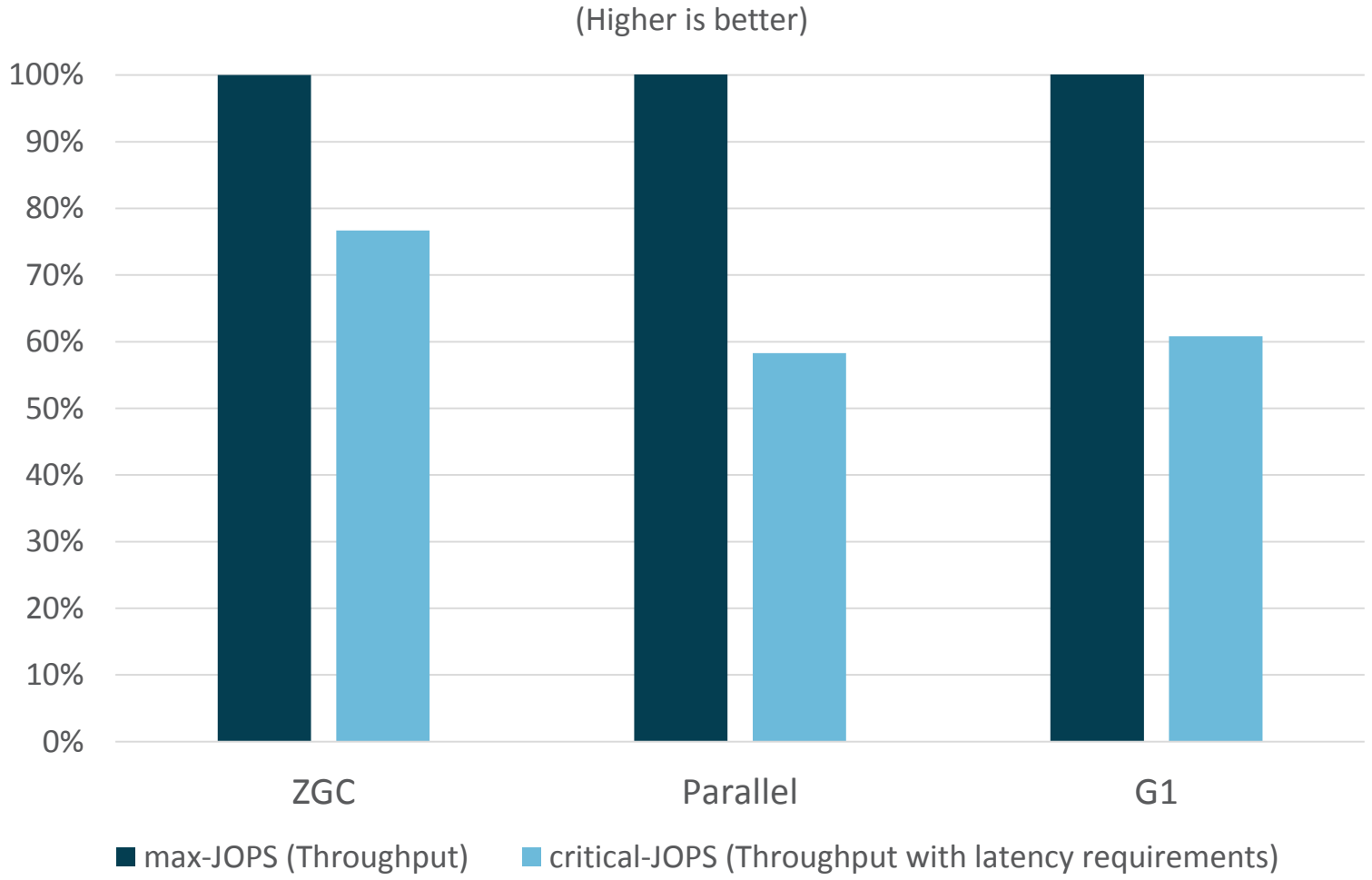


Concurrent Prepare for Relocate (**Second Cycle**)



Performance

SPECjbb®2015 – Score



Mode: Composite

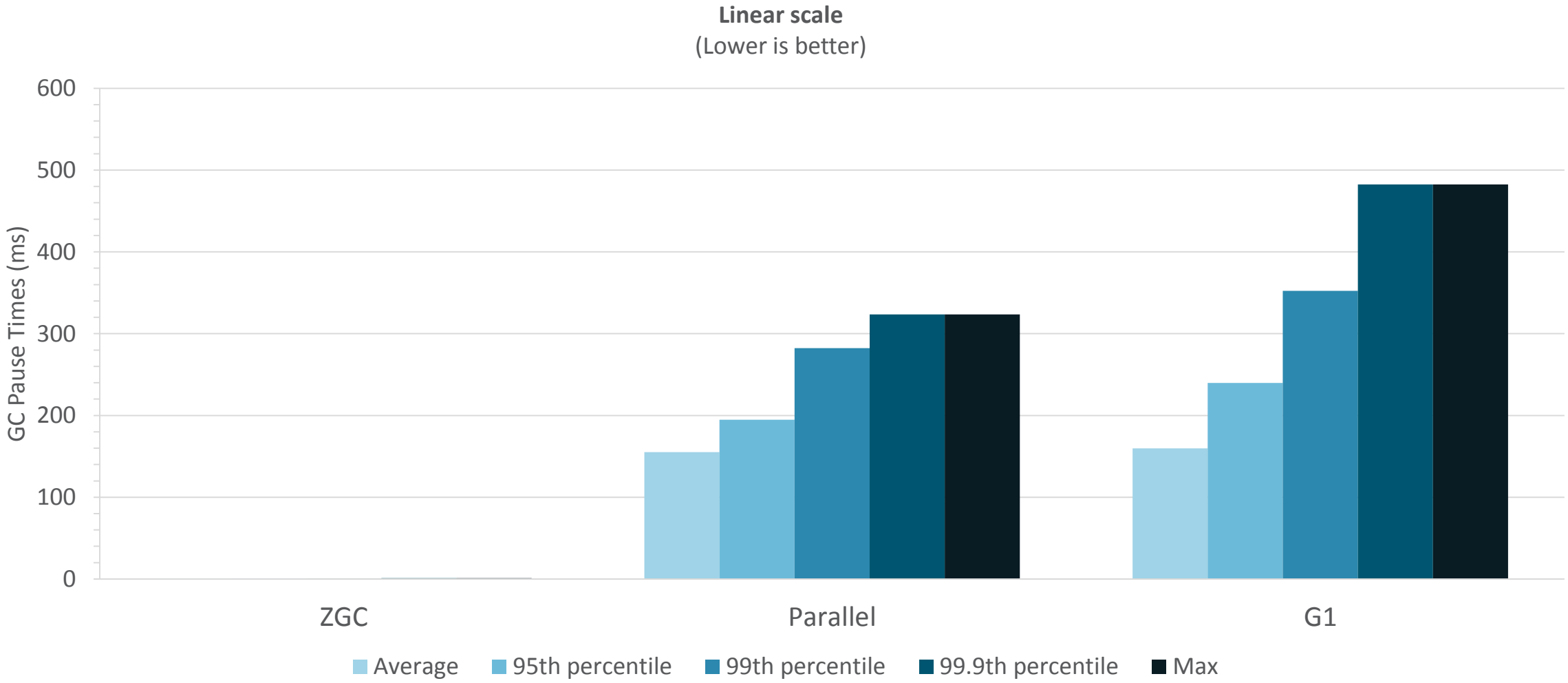
Heap Size: 128G

OS: Oracle Linux 7.5

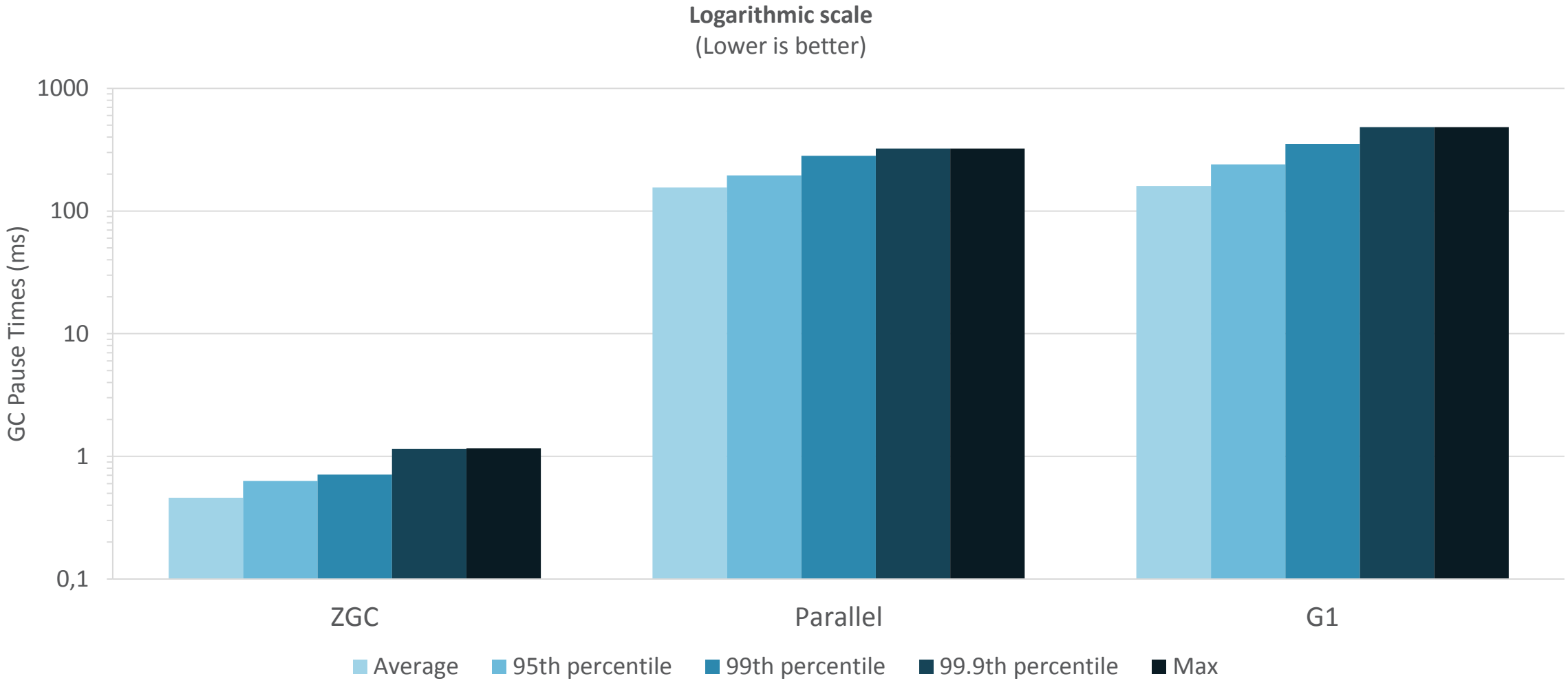
HW: Intel Xeon E5-2690 2.9GHz
2 sockets, 16 cores (32 hw-threads)

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.

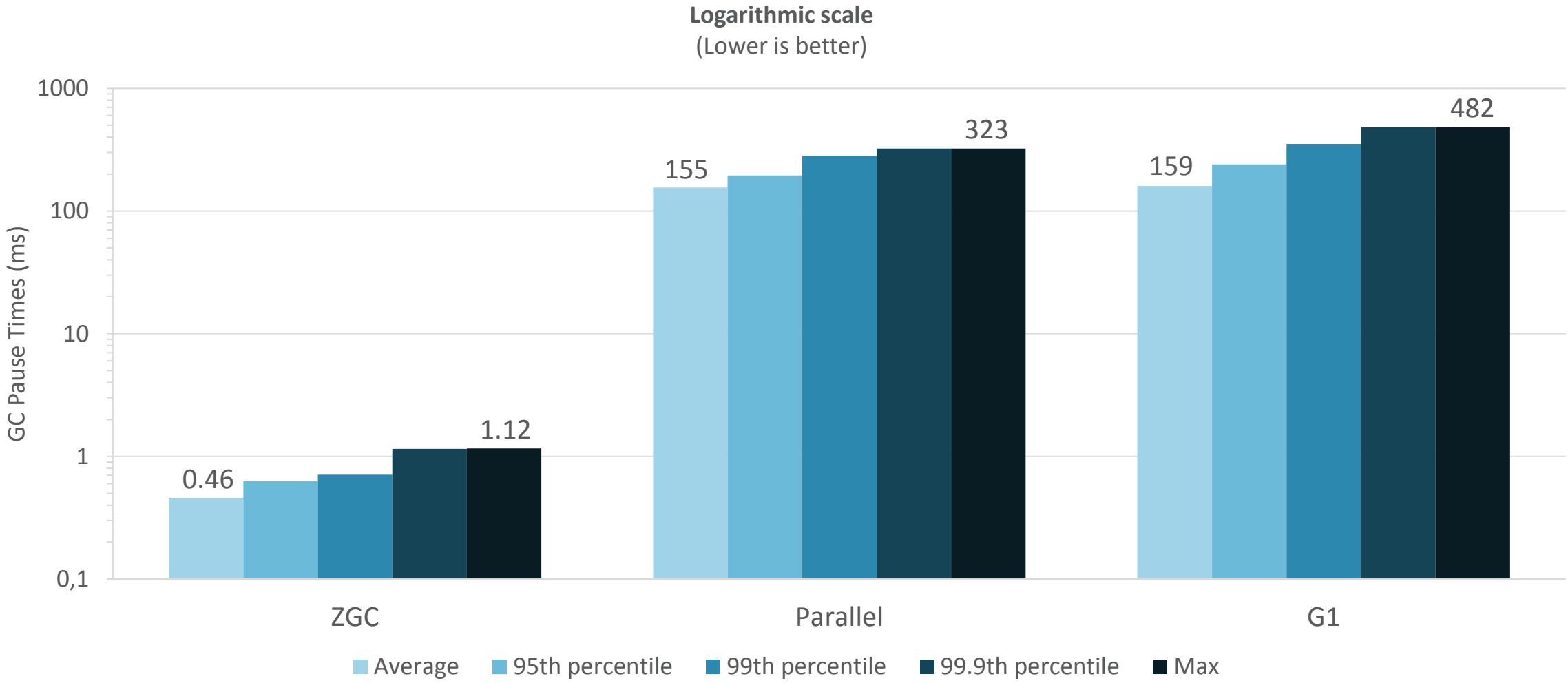
SPECjbb®2015 – GC Pause Times



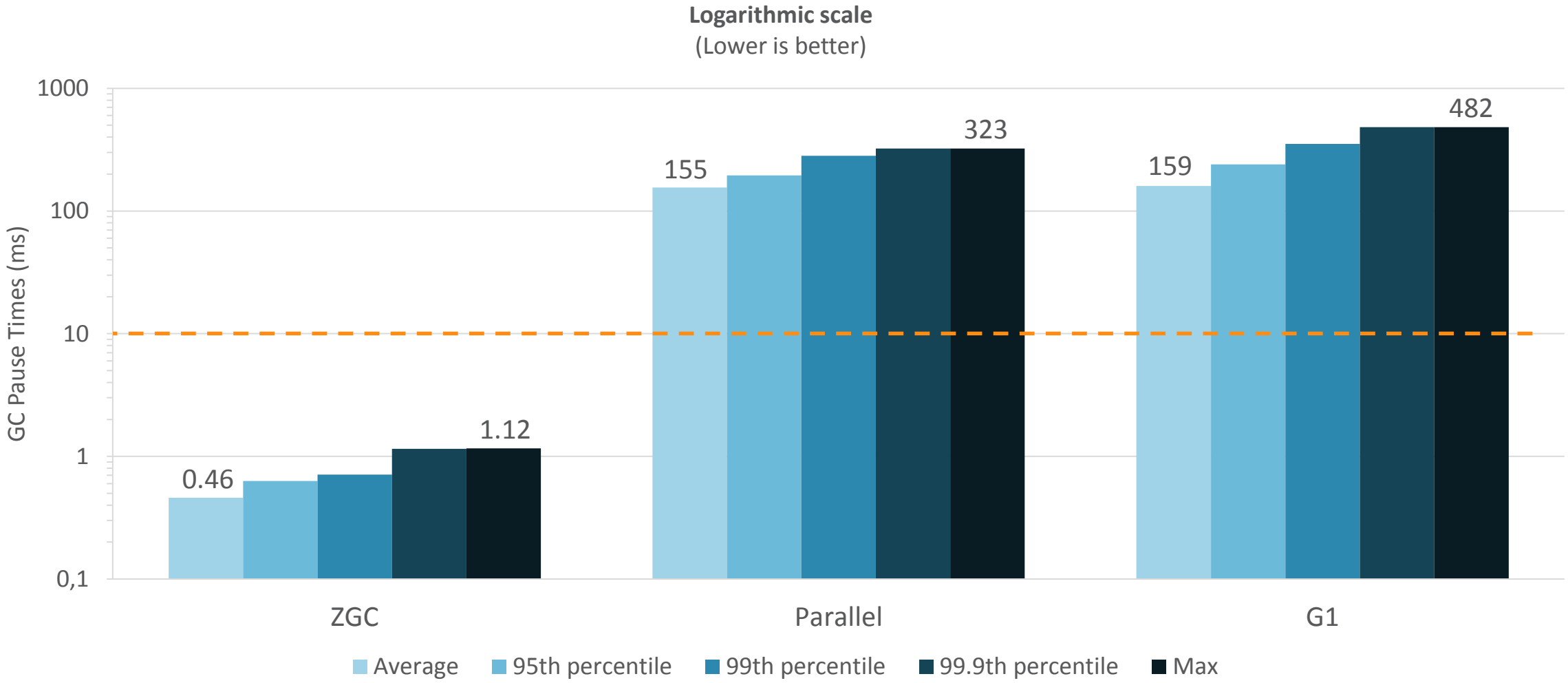
SPECjbb®2015 – GC Pause Times



SPECjbb®2015 – GC Pause Times



SPECjbb®2015 – GC Pause Times



Going Forward

- Generational
- Sub-millisecond *max* pause times
- Support additional platforms
- Graal JIT support

ZGC Project

OpenJDK

Mailing lists

hotspot-gc-dev@openjdk.java.net
zgc-dev@openjdk.java.net

Wiki

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

Source

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

Java™ ORACLE

Thanks!

Questions?


Java™
ORACLE®