Write Barriers in Garbage First Garbage Collector

-By Monica Beckwith
Code Karam LLC
@mon_beck; monica@codekaram.com

About me

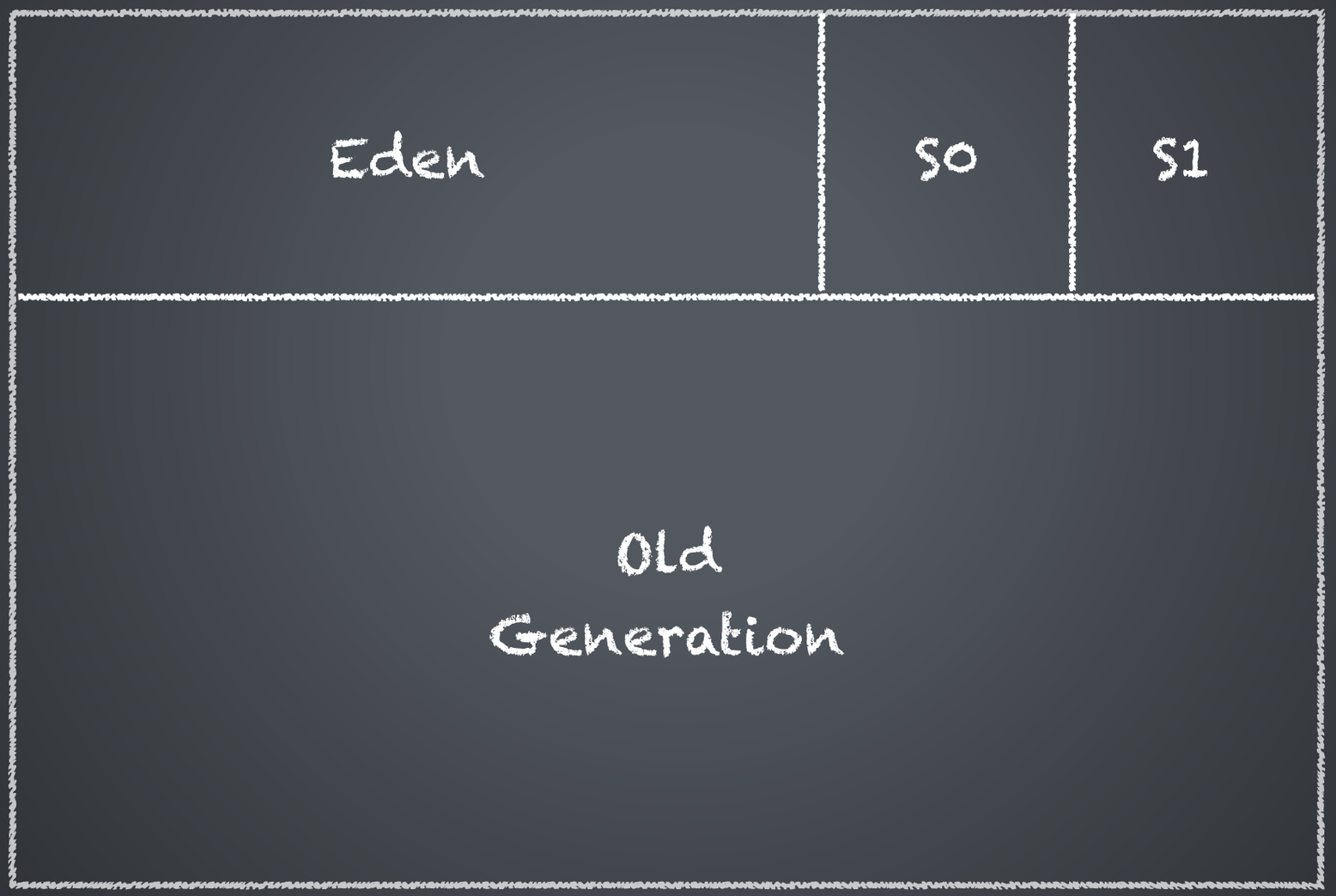
- Java performance engineer
- I am working as a consultant and an instructor
- I have worked with Oracle, Sun, AMD …
  - I used to work in the capacity of G1 GC performance lead @Oracle.

Agenda

- Heap regions and additional data structures
  - RSets
  - Barriers
    - Concurrent refinement
- G1 GC Stages
- Concurrent marking in G1 GC
  - Pre-write barrier
  - SATB algorithm

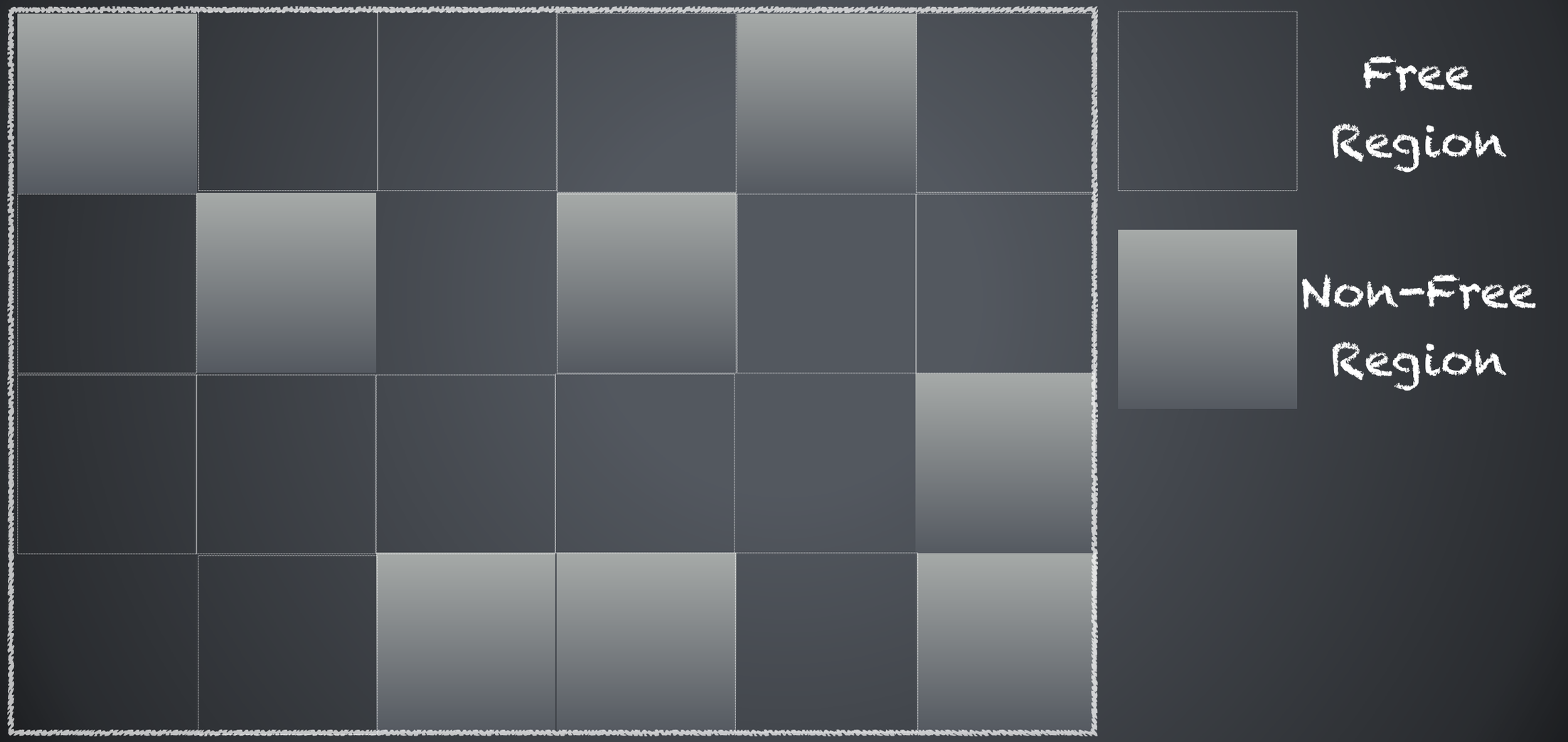
Regionalized Heap

Traditional Java Heap



Contiguous Java Heap

Garbage First GC - Heap Regions



G1 GC Heap Regions

- Young Regions - Regions that house objects in the Eden and Survivor Spaces

- Old Regions - Regions that house objects in the Old generation.

- Humongous Regions - Regions that house Humongous Objects.

Additional Data Structures

G1 GC Collection Set & Remembered Sets

- Additional data structure to help with maintenance and collection

- Add a slight footprint overhead (~5%)

Remembered Sets (RSets)

Remembered Sets

- Maintains and tracks incoming references into its region
- old-to-young references
- old-to-old references

Remembered Sets

- Remembered sets have varying granularity based on the “popularity” of objects or regions.

Remembered Sets

- Different granularities:
  - sparse per-region-table (PRT)
  - fine-grained PRT
  - coarse-grained bitmap

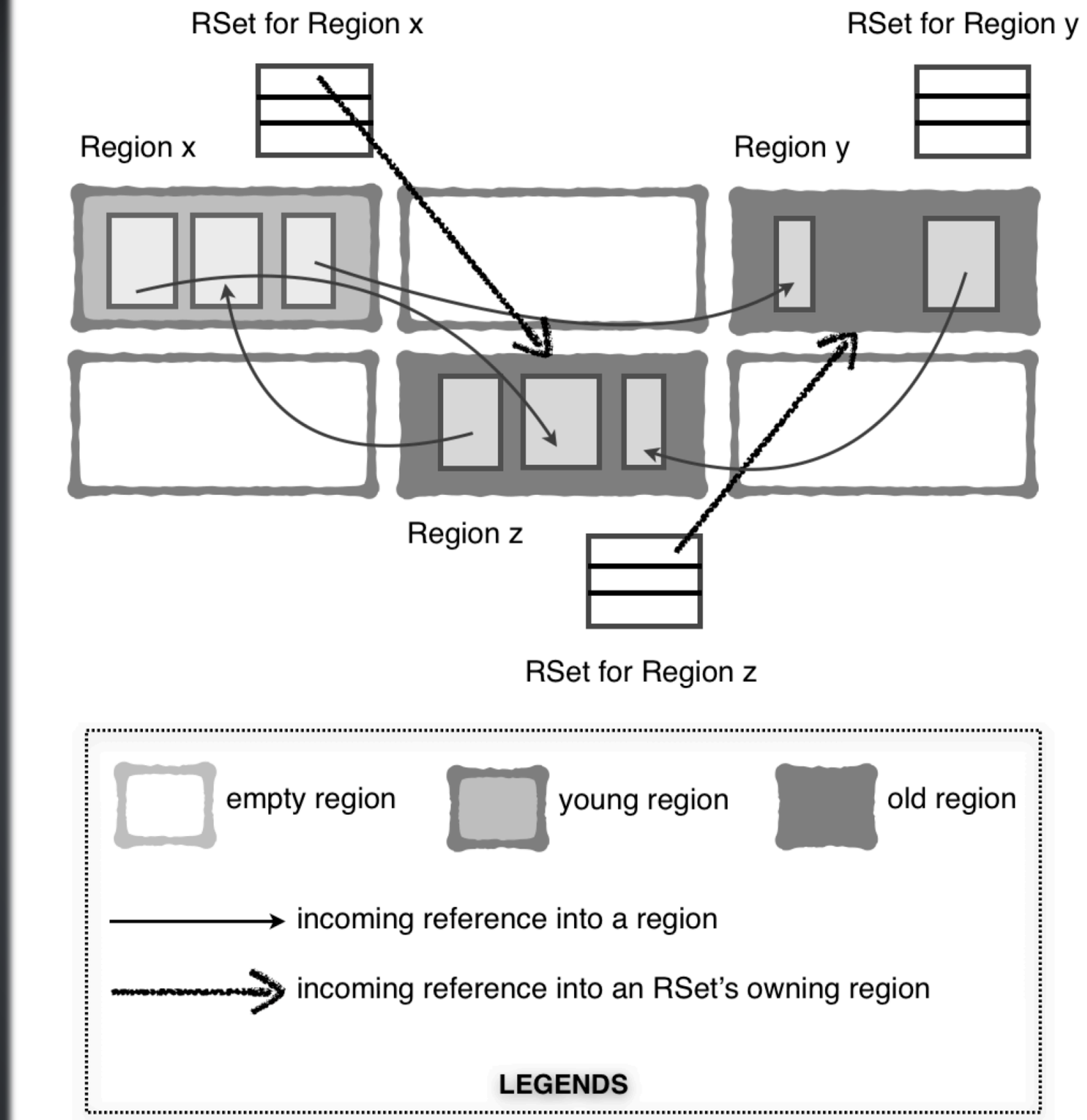


Figure 2.3 Remembered sets with incoming object references

RSet Maintenance: Barriers + Refinement Threads

Write Barrier

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.

Post-Write Barrier

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

Concurrent Refinement Threads

Concurrent Refinement Threads

- The refinement threads will scan cards in the filled update log buffers to update the RSets for their corresponding regions.
- The refinement threads are always active
- G1 GC deploys them in a tiered manner to keep up with the filled buffers

Concurrent Refinement Threads

- Mutator threads can be enlisted to help with processing of filled buffers.
- Avoid this scenario, since the Java application will be halted until the filled buffers are processed!

G1 GC Stages

A Young Collection

G1 GC Stages - Young Collection

- When young regions are full and no further allocations can happen
- Need to start a stop-the-world collection
  - Age objects in survivor regions
  - Promote aged objects into the old regions

Marking Threshold

Initiating Heap Occupancy Percent

- Threshold to start a marking cycle to identify candidate old regions for collection during a mixed/incremental collection
- When old generation occupancy crosses this adaptive threshold, a marking cycle can start

Marking

Stages of Marking

- Initial-mark
- Root region scan
- Concurrent mark
- Remark/ Final mark
- Cleanup

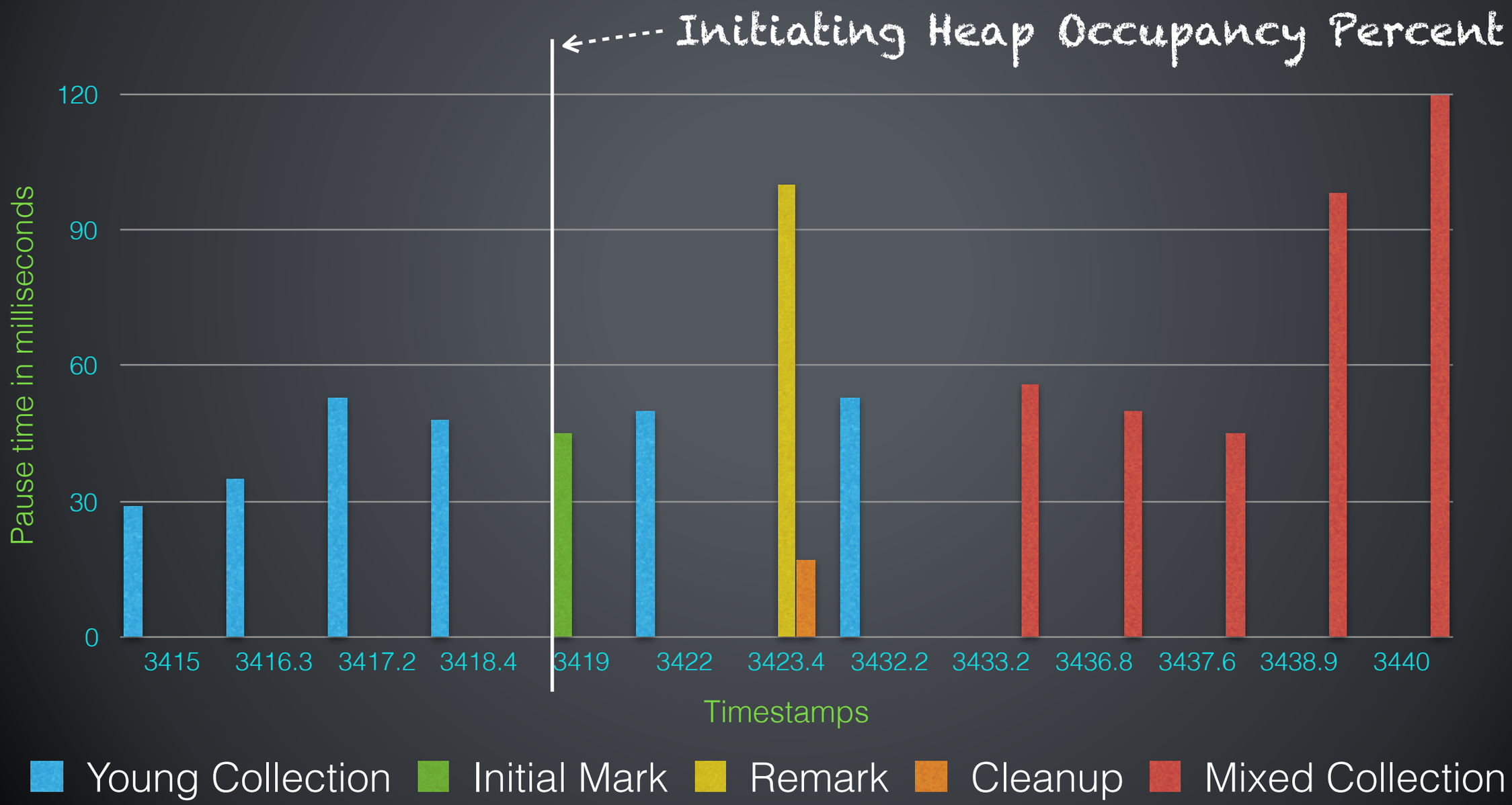
A Mixed Collection

G1 GC Stages - Mixed Collection

- When candidate old regions are available and the potential of recovered space is over the (internal) reclaimable threshold
  - This is a stop-the-world collection
  - All regions in the young generation are included in this collection
  - Candidate old regions are added based on the reclaimable space and other min/max thresholds
  - Can have multiple mixed collection pauses based on the total old regions identified and a (internal) count target.

G1 GC Pause Histogram

The Garbage First Collector - Pause Histogram



Concurrent Marking in G1 GC

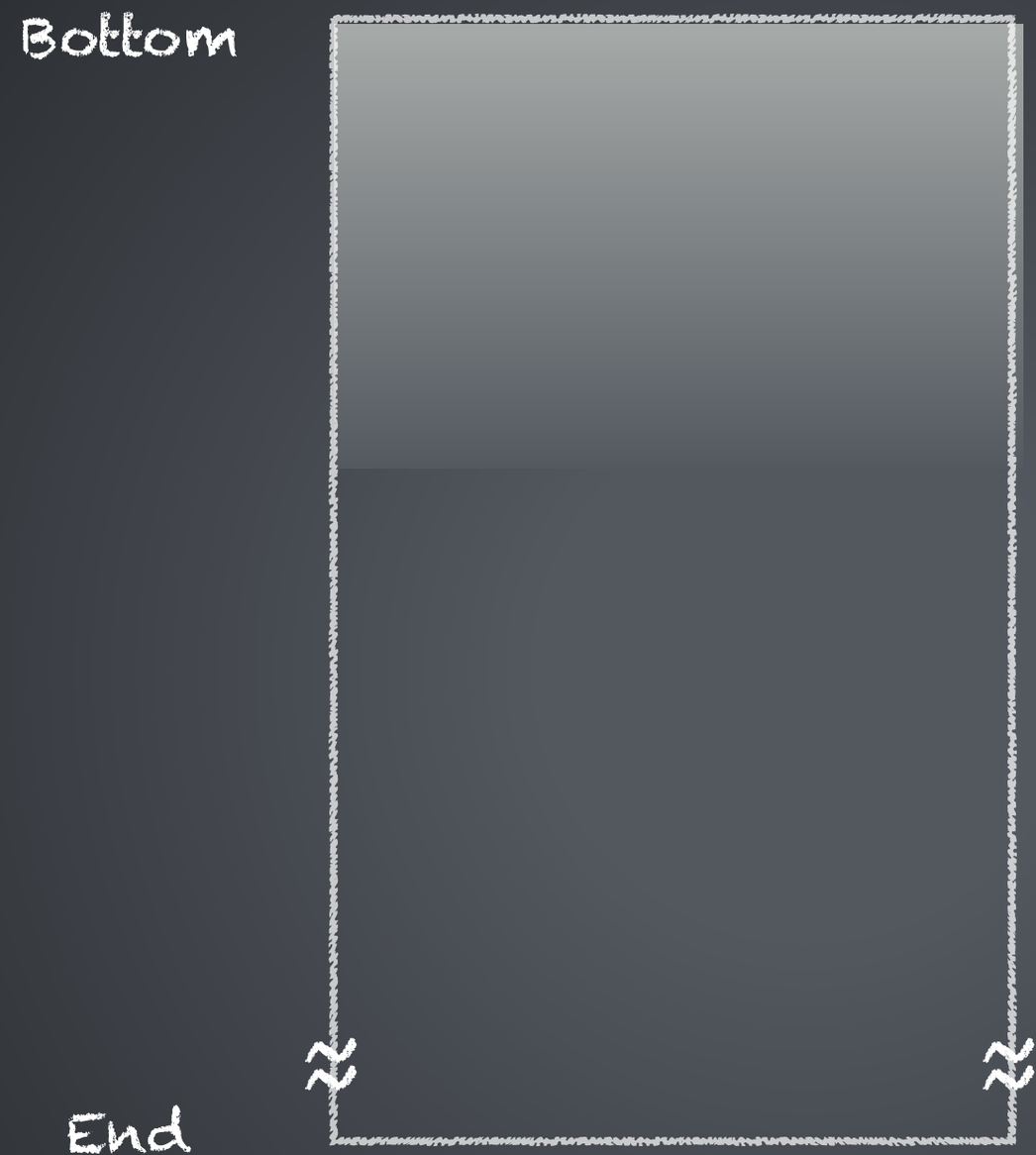
Recap: Stages of Marking

- Initial-mark
- Root region scan
- Concurrent mark
- Remark/ Final mark
- Cleanup

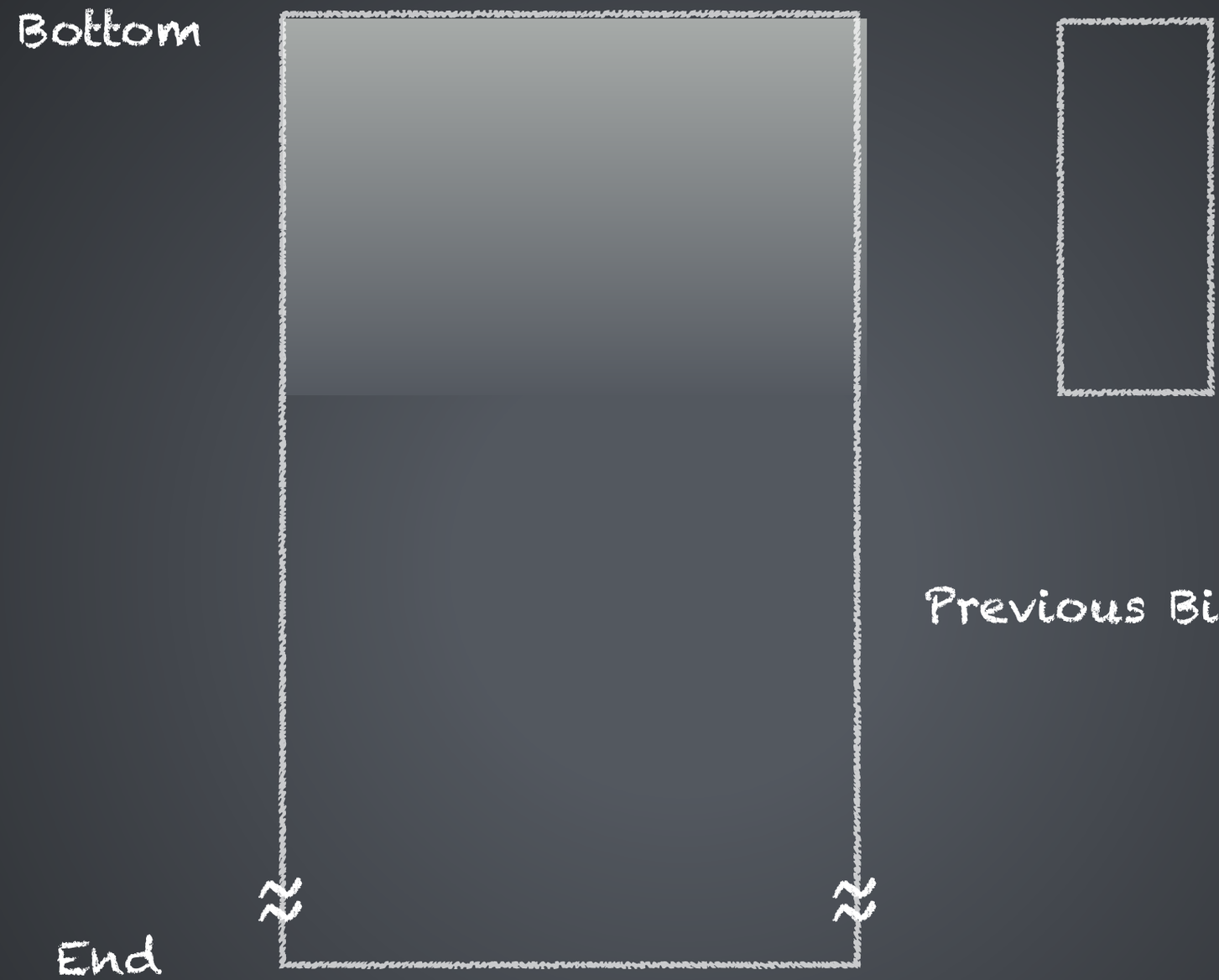
Marking Highlights

- Employs ‘Snapshot At The Beginning (SATB) algorithm
  - Incremental and concurrent marking algorithm.
- A pre-write barrier is needed to gather the snapshot

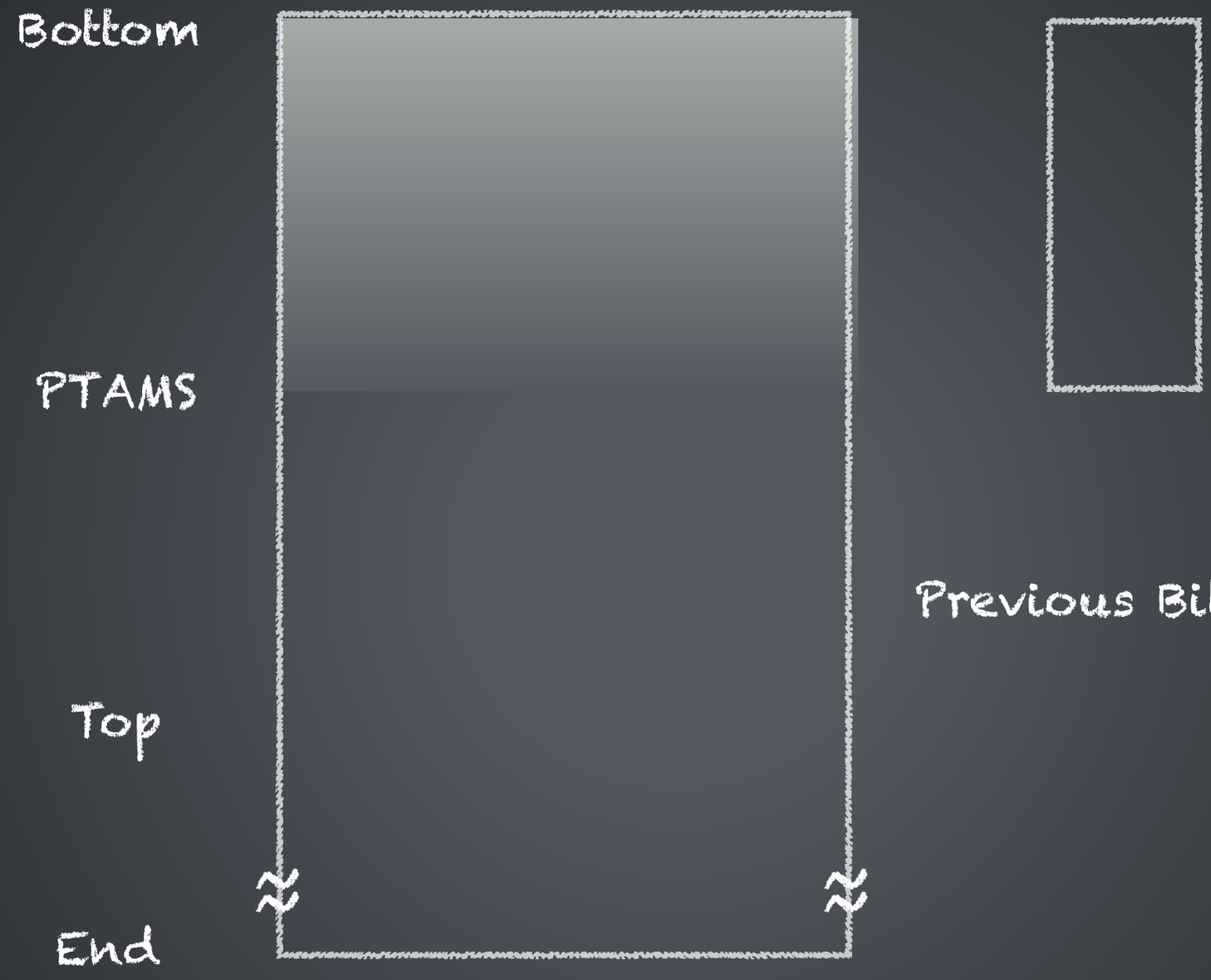
SATB Algorithm For Concurrent & Incremental Marking



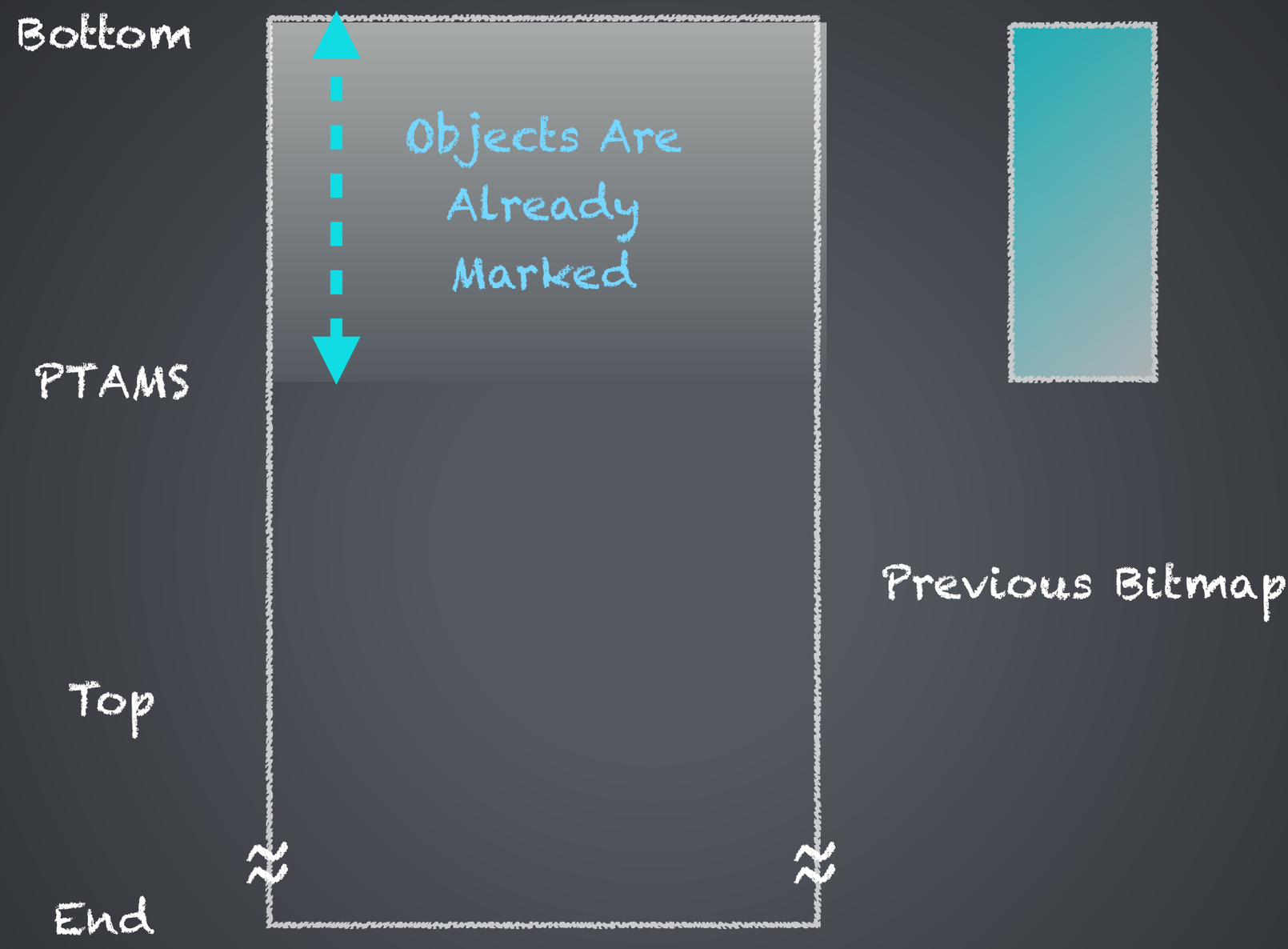
A Java Heap



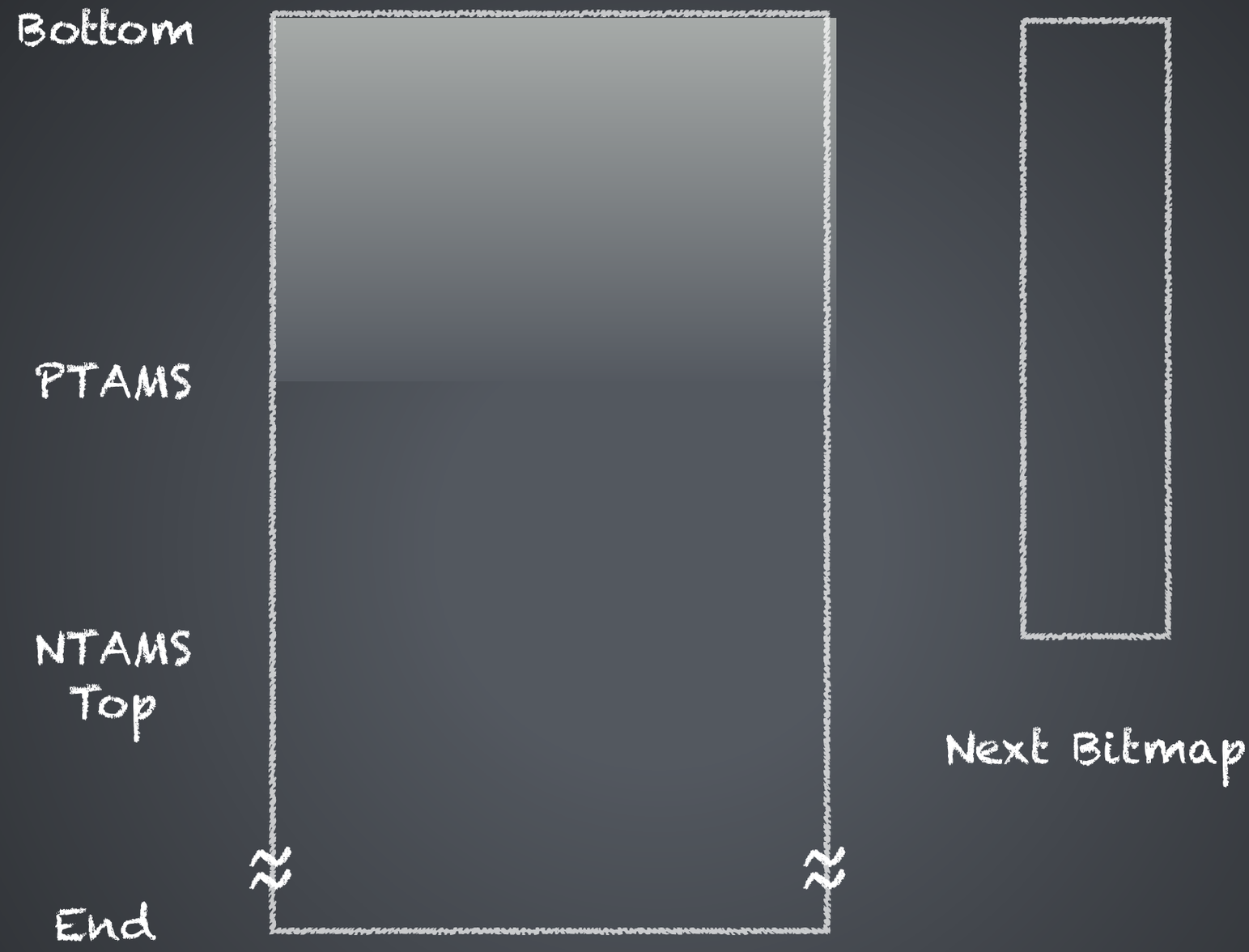
A Java Heap With Previous Bitmap



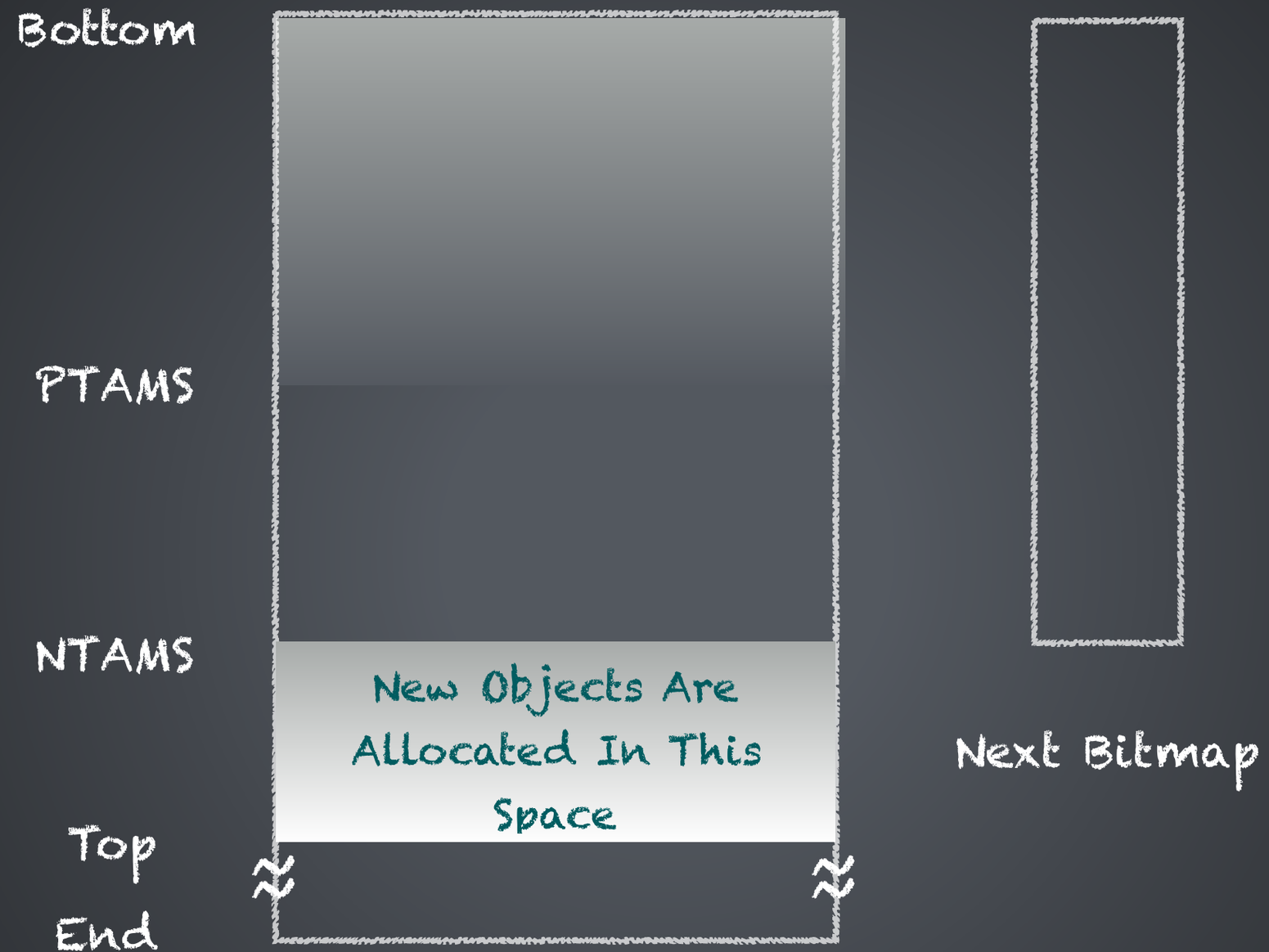
A Java Heap (Showing PTAMS) With Previous Bitmap



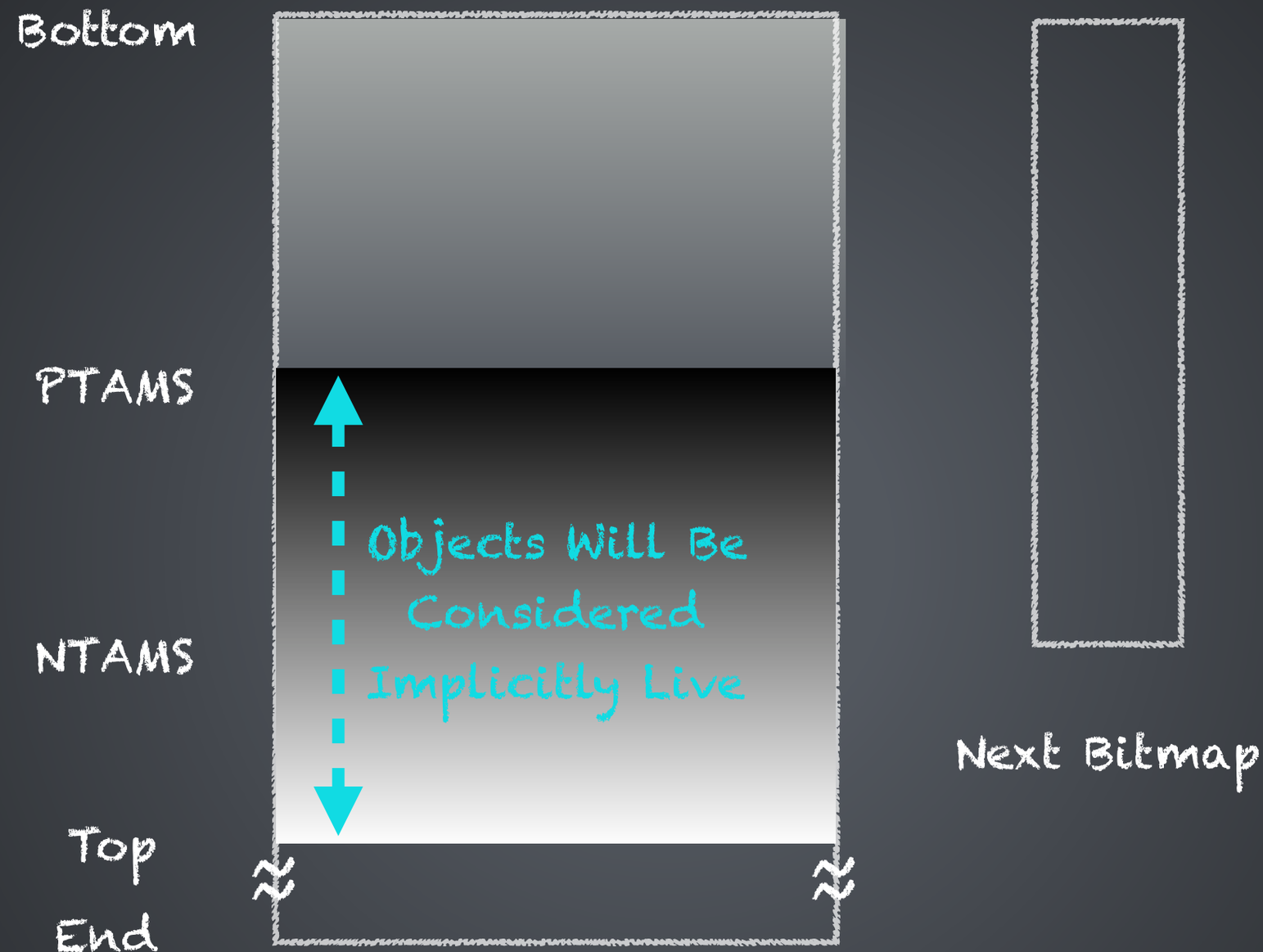
A Java Heap (Showing PTAMS) With Previous Bitmap



A Java Heap (Showing PTAMS + NTAMS) With Previous Bitmap



Same Java Heap During Concurrent Marking



Same Java Heap During Concurrent Marking

Pre-Write Barrier

Pre-Write Barrier

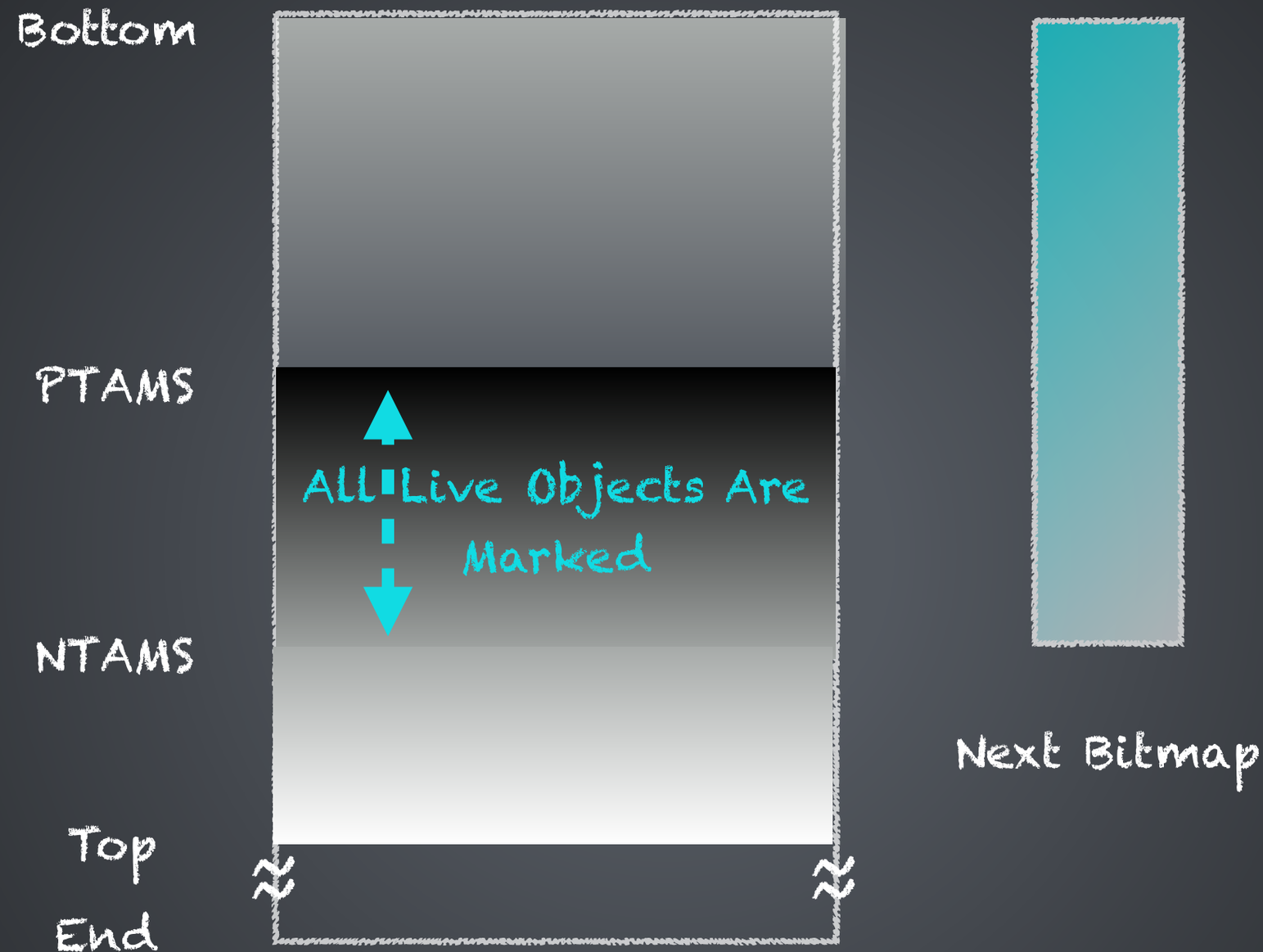
- To record the previous value of the reference fields of objects that were reachable at the start of marking + were a part of the snapshot.

- Prevents those objects from being overwritten by the mutator thread

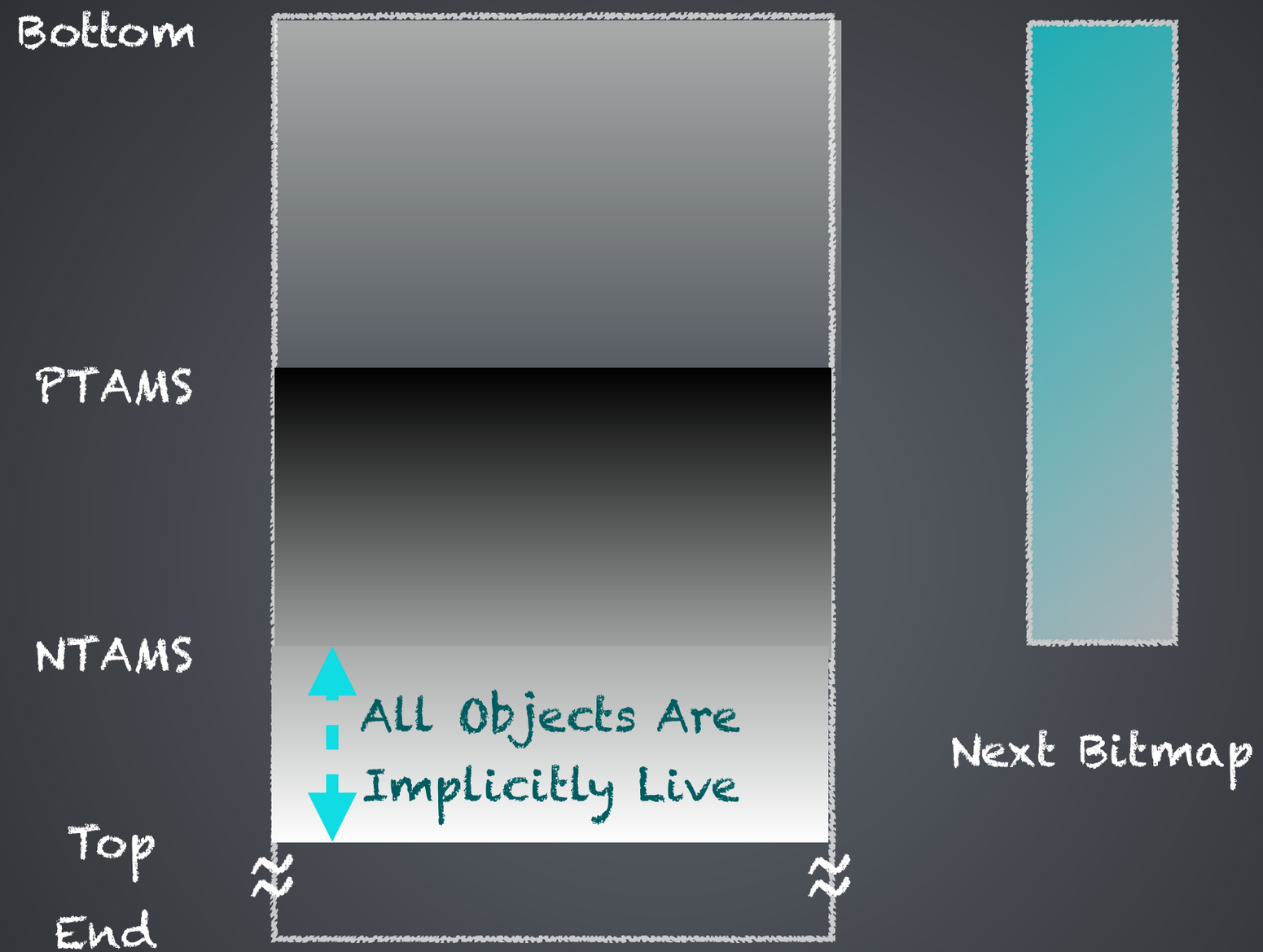
- Mutator thread logs the previous value of the pointer in an SATB buffer

Pre-Write Barrier - Pseudo Code

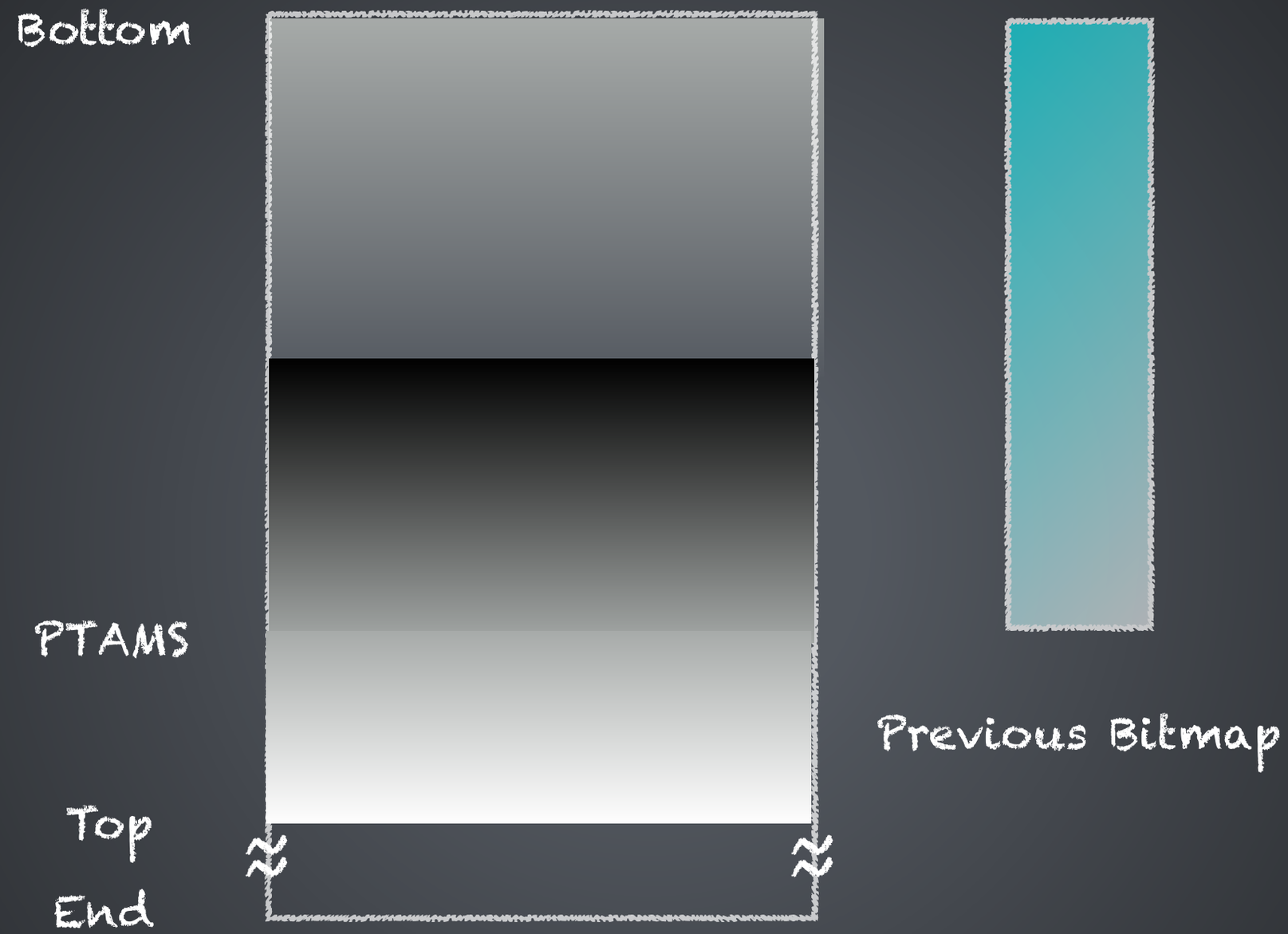
```
if (marking_is_active) {

 pre_val:= x.f;

 if (pre_val:= NULL) {

  satb_enqueue(pre_val);

 }

}
```



Same Java Heap At The End Of Remark Pause



Same Java Heap At The End Of Remark Pause



Same Java Heap During Cleanup

References and Additional Reading

- D. L. Detlefs, C. H. Flood, S. Heller, and T. Printezis. Garbage-First Garbage Collection.

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