
ORACLE®


Java™
ORACLE®

G1 – ~~Not~~ Never Done!

Thomas Schatzl
thomas.schatzl@oracle.com
Oracle Java Hotspot Virtual Machine GC Team
February 03, 2018

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.

Program Agenda

1. Parallel Full GC
2. Faster Card Scanning
3. Rebuild Remembered Sets Concurrently
4. Abortable Mixed Collections
5. Automatic Thread Sizing
6. Participate!

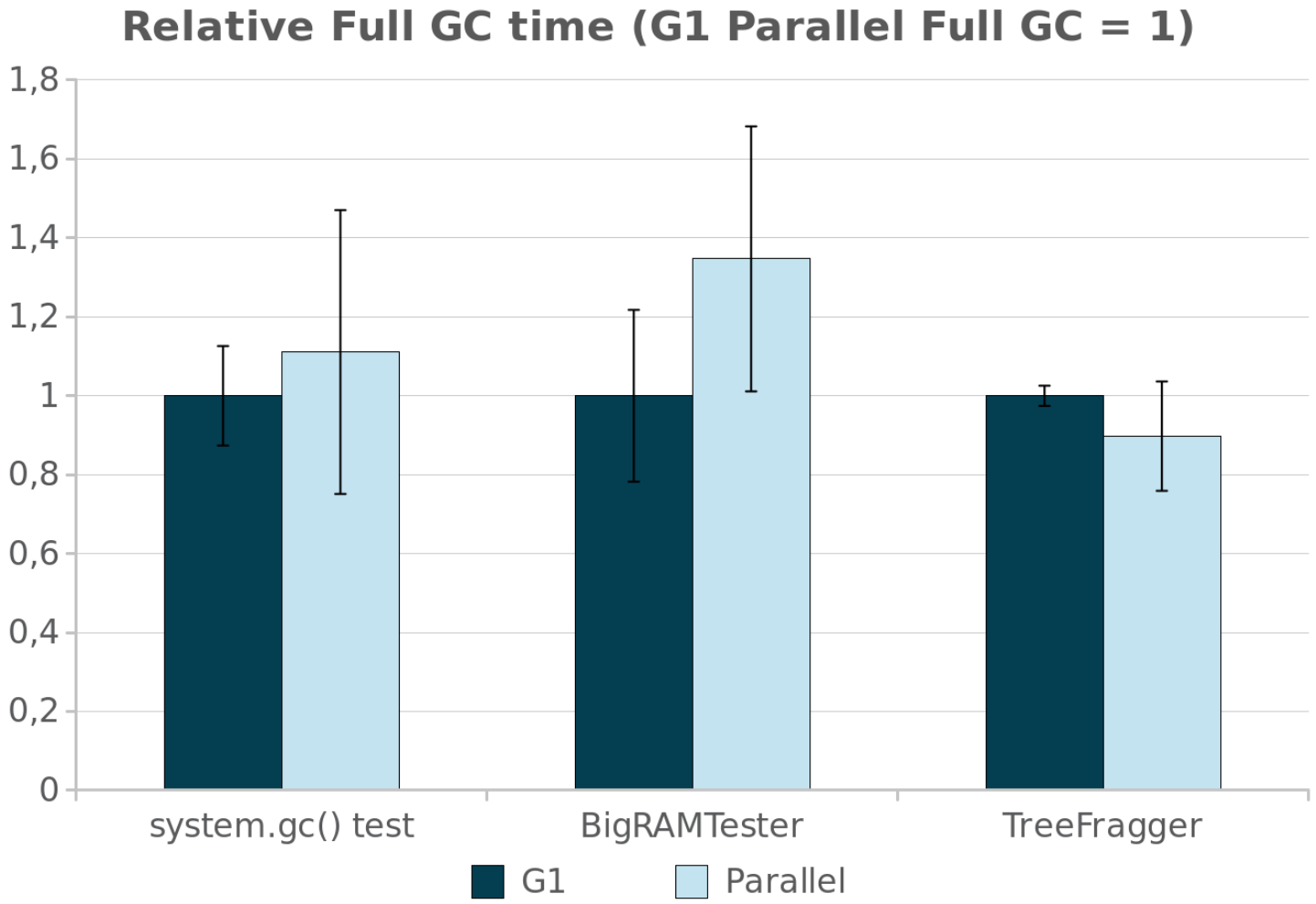
Program Agenda

1. Parallel Full GC
2. Faster Card Scanning
3. Rebuild Remembered Sets Concurrently
4. Abortable Mixed Collections
5. Automatic Thread Sizing
6. Participate!

G1 Parallel Full GC

- G1 Full GC very slow
  - High worst-case latencies and bad throughput
  - Goal: be on par with Parallel GC Full GC
- Solution
  - Parallelize Mark-Sweep-Compact

G1 Parallel Full GC



- system.gc() test
  - performs many System.gc(), very small live set, 5G heap
- BigRAMTester
  - LRU-cache-stress test application, many references, large (90%) live set, 10G heap (JDK-8152438)
- TreeFragger
  - Fragmentation-inducing benchmark from RedHat, medium live set, 20G heap

G1 Parallel Full GC

- Available since build jdk-10-ea+34

Program Agenda

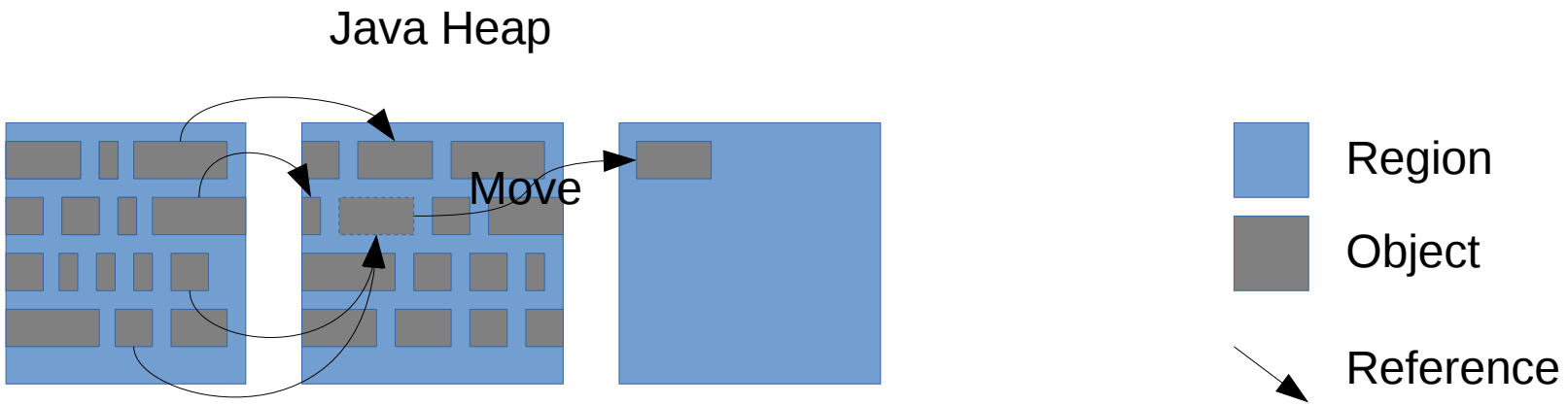
1. Parallel Full GC
2. Faster Card Scanning
3. Rebuild Remembered Sets Concurrently
4. Abortable Mixed Collections
5. Automatic Thread Sizing
6. Participate!

Faster Card Scanning

What is a card?

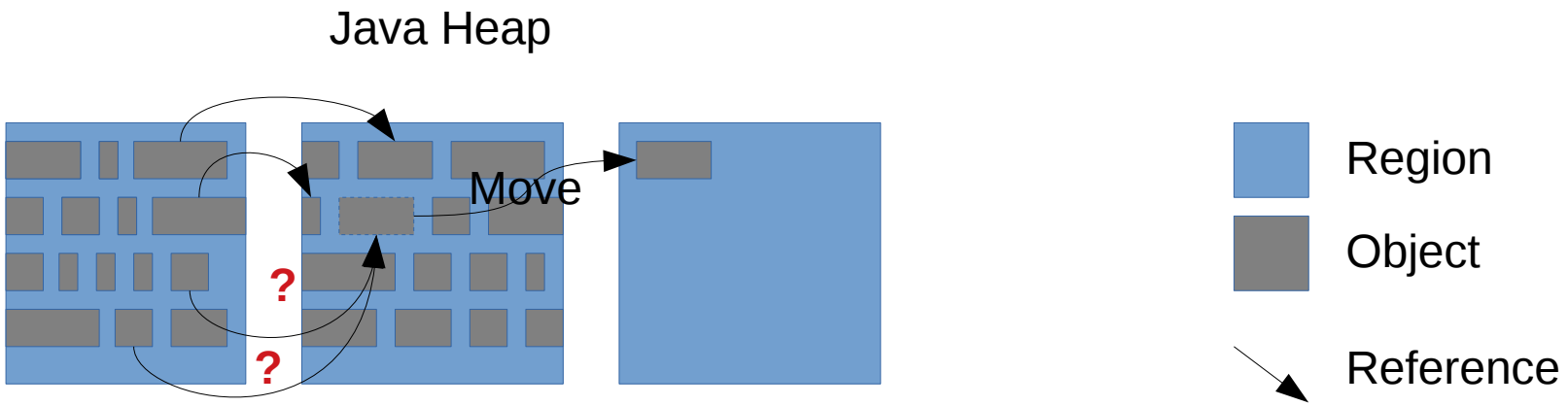
Faster Card Scanning

What is a card?



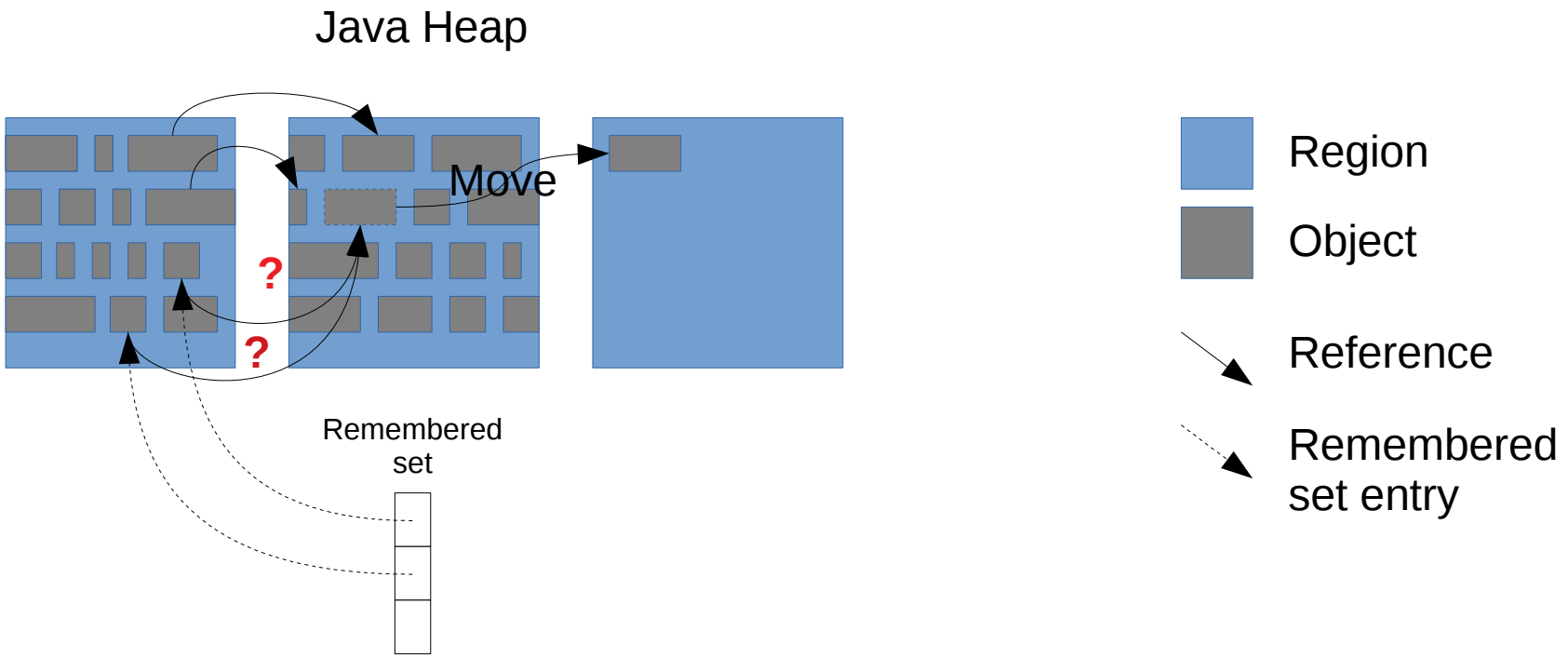
Faster Card Scanning

What is a card?



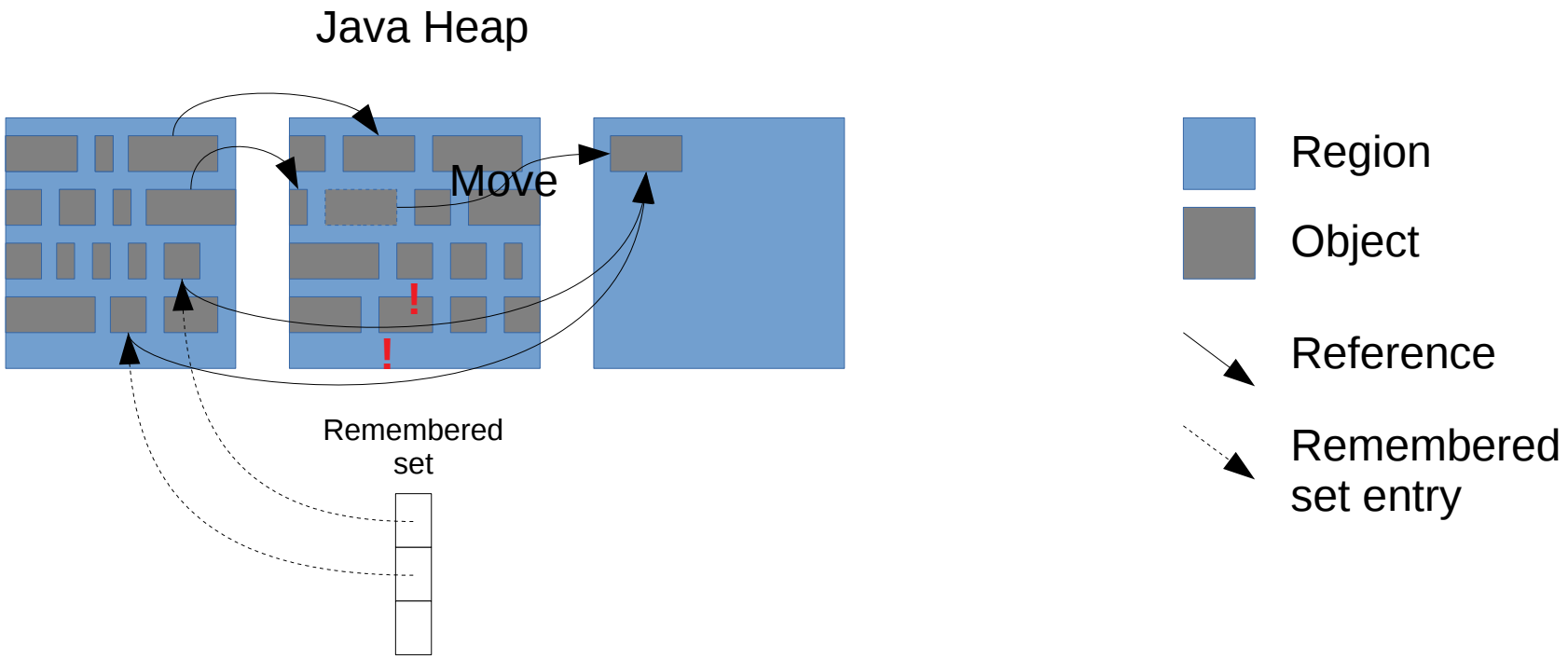
Faster Card Scanning

What is a card?



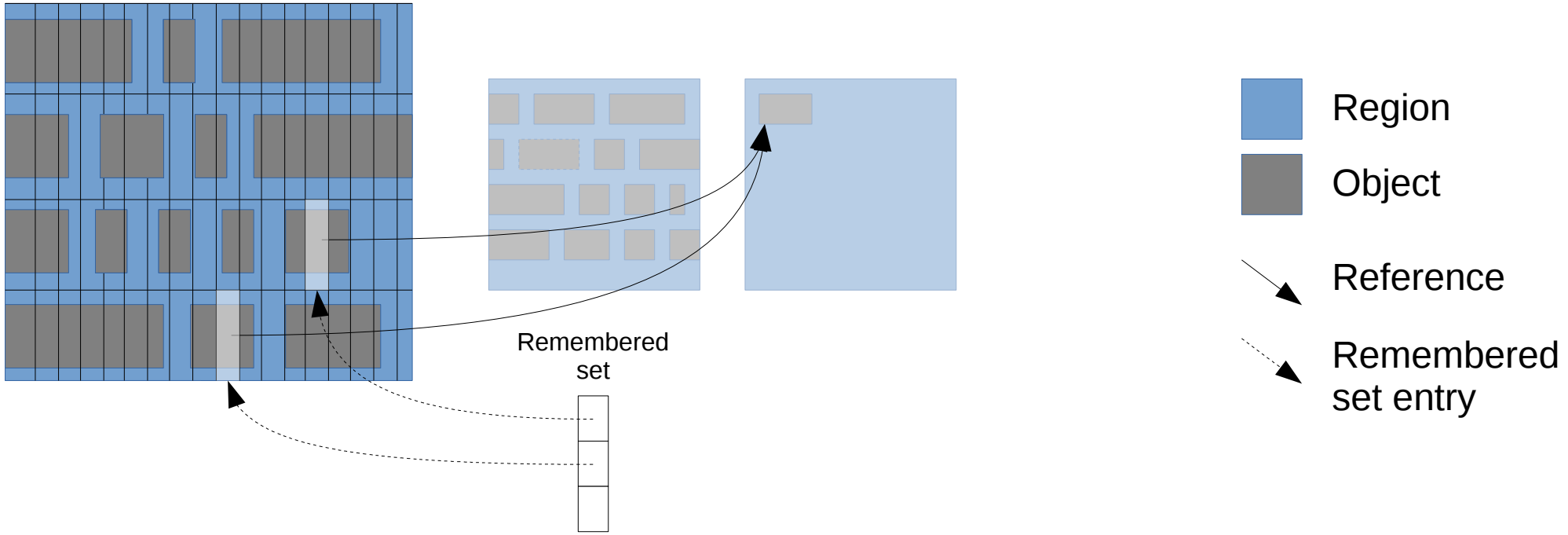
Faster Card Scanning

What is a card?



Faster Card Scanning

What is a card? A small subdivision of memory



Faster Card Scanning

Problem

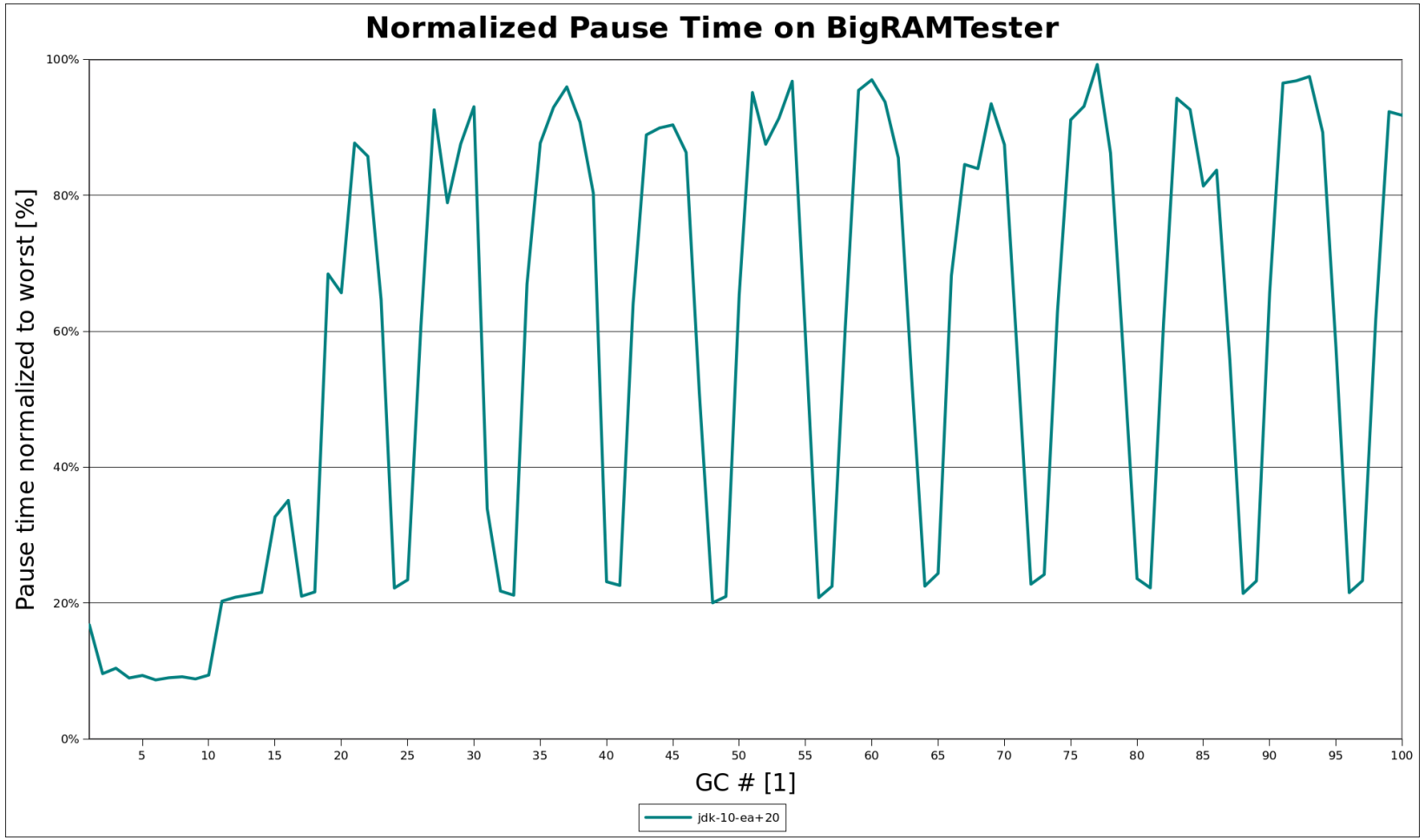
- GC needs to find references in cards in remembered sets to moved objects quickly

Faster Card Scanning

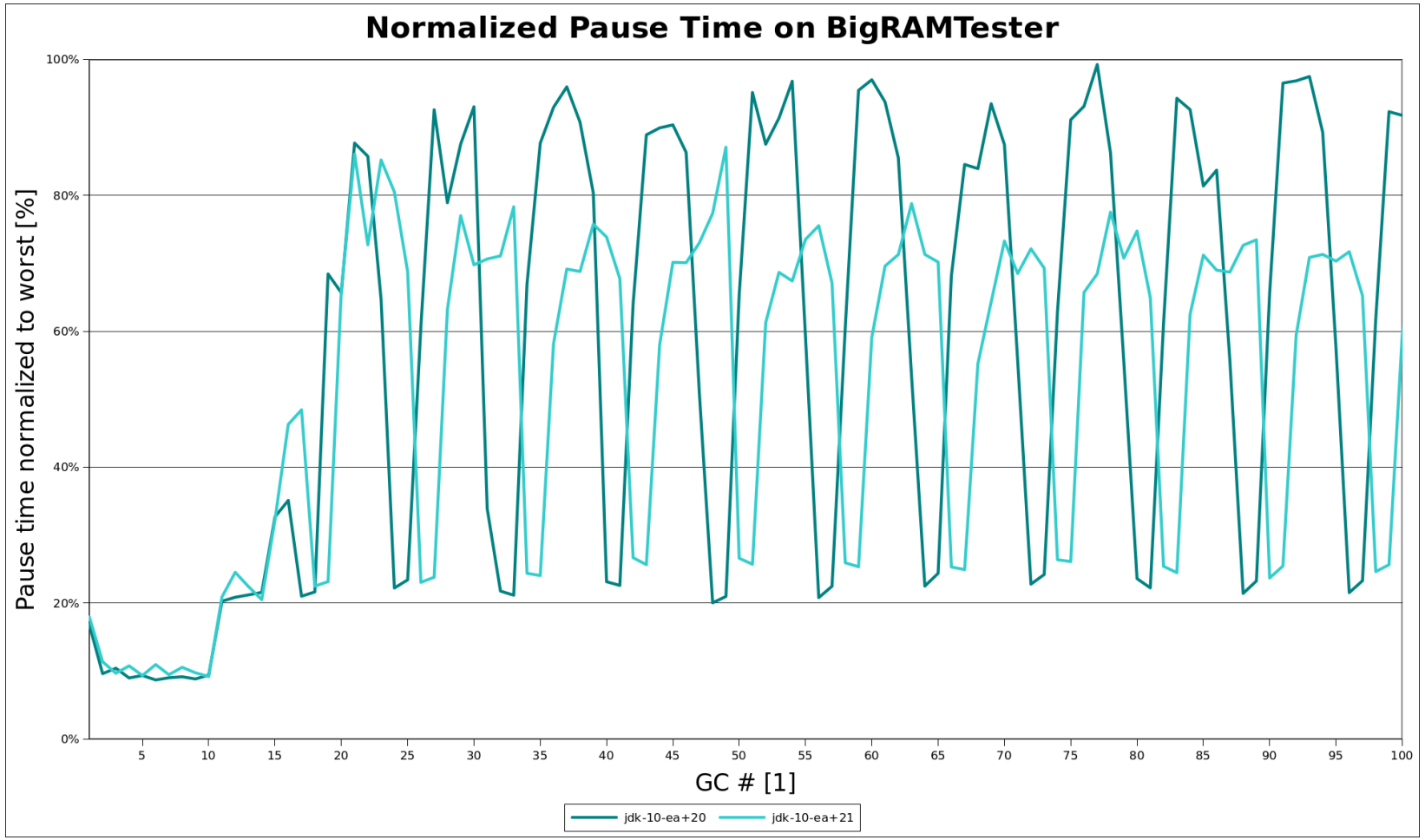
Solution

- Refactor and improve scanning and updating remembered sets
  - Remove overly generic code
  - Replace by specialized code for different situations
  - Subsume and remove obsolete checks

Faster Card Scanning



Faster Card Scanning



Faster Card Scanning

- Available since build jdk-10-ea+21

Program Agenda

1. Parallel Full GC
2. Faster Card Scanning
3. Rebuild Remembered Sets Concurrently
4. Abortable Mixed Collections
5. Automatic Thread Sizing
6. Participate

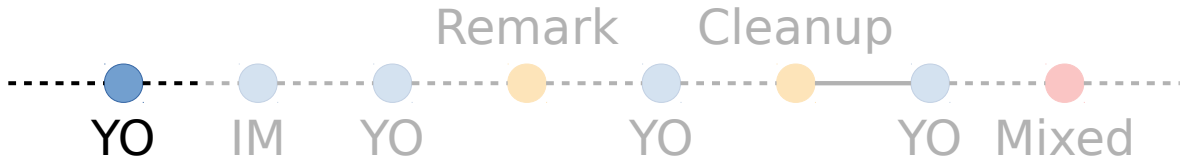
Rebuild Remembered Sets Concurrently

Problem

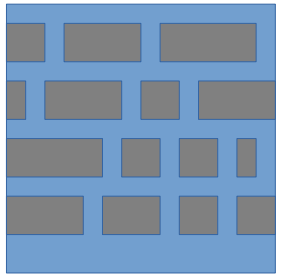
- Remembered sets may occupy a lot of memory
  - Known to take ~20% of total heap in some situations
    - E.g. 20GB with a 100GB heap
  - Upper Bounds are even higher
    - O(#regions^2)
- Old regions use most remembered set memory

Rebuild Remembered Sets Concurrently

Collection Cycle – Some Young-Only GCs



YO … Young-Only
IM … Initial-Mark



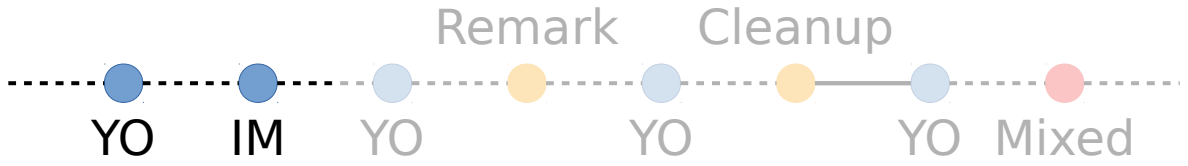
Remembered set

Region

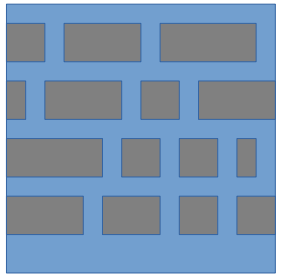
Object

Rebuild Remembered Sets Concurrently

Collection Cycle – Start marking with Initial Mark



YO … Young-Only
IM … Initial-Mark



Remembered set

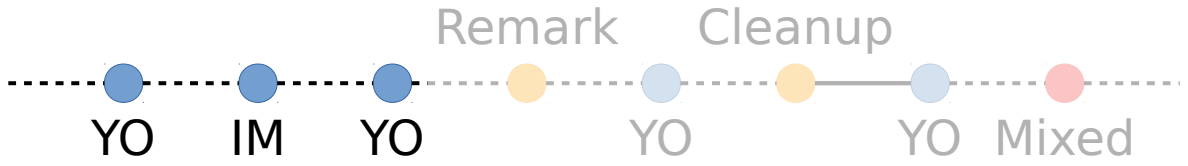
X
X

Region

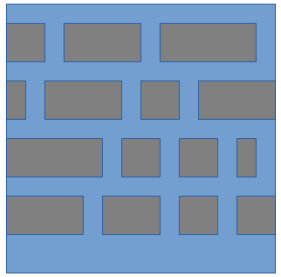
Object

Rebuild Remembered Sets Concurrently

Collection Cycle – Some more Young-Only while marking



YO … Young-Only
IM … Initial-Mark



Remembered set

X	X
X	
X	

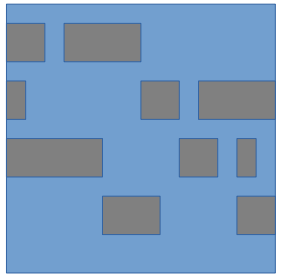
Region

Object

Rebuild Remembered Sets Concurrently

Collection Cycle – Marking finished

YO … Young-Only
IM … Initial-Mark



Remembered set

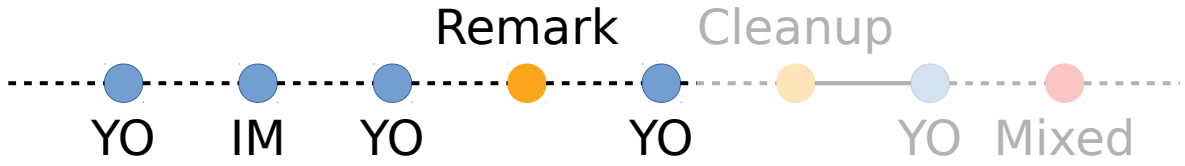
x	x	x
x	x	
x	x	

Region

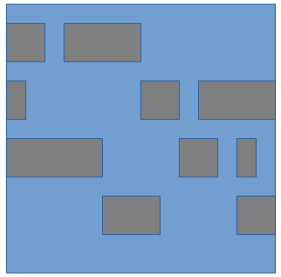
Object

Rebuild Remembered Sets Concurrently

Collection Cycle – Create “live data map”



YO … Young-Only
IM … Initial-Mark



Remembered set

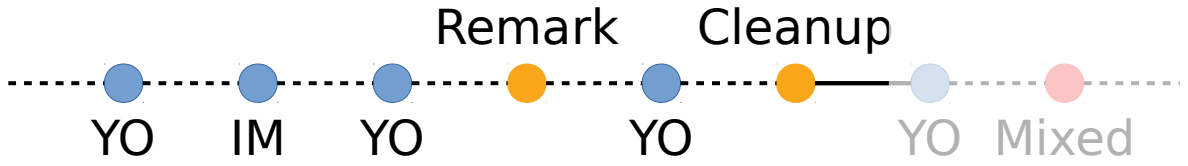
x	x	x
x	x	x
x	x	

Region

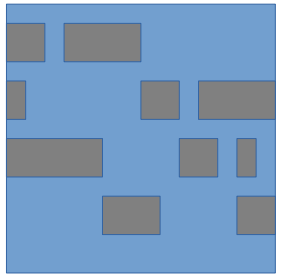
Object

Rebuild Remembered Sets Concurrently

Collection Cycle – Clean out obsolete remembered set entries



YO … Young-Only
IM … Initial-Mark



Remembered set

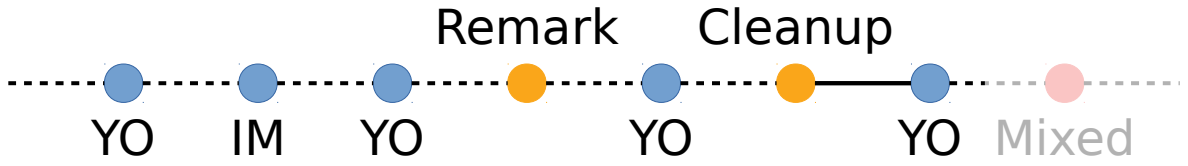
X		X
X	X	
	X	

Region

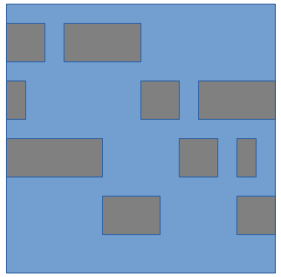
Object

Rebuild Remembered Sets Concurrently

Collection Cycle – Wait for old gen reclamation start



YO … Young-Only
IM … Initial-Mark

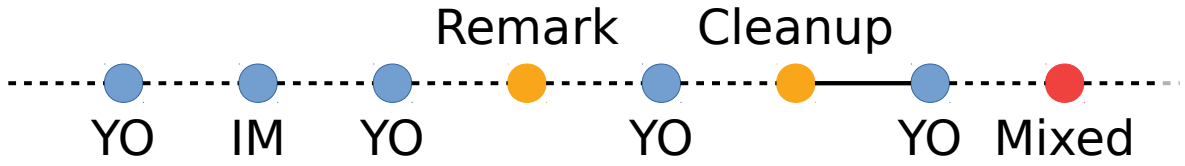


Remembered set

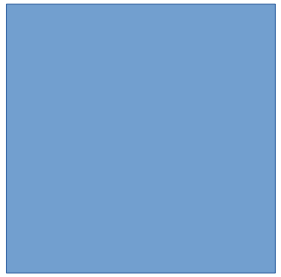
X	X	X
X	X	
	X	X

Rebuild Remembered Sets Concurrently

Collection Cycle – Region gets evacuated



YO … Young-Only
IM … Initial-Mark

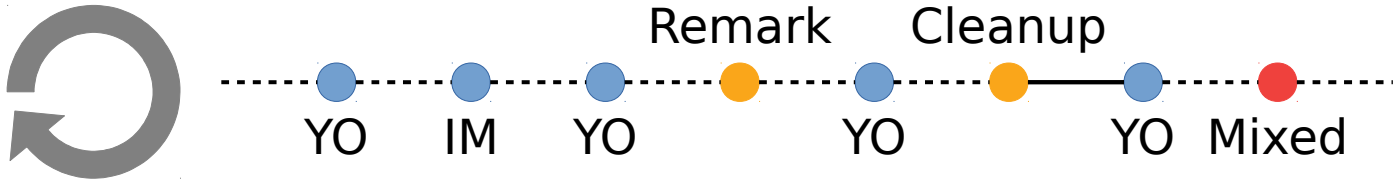


Region

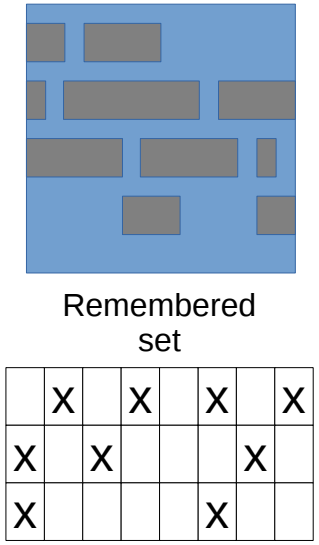
Object

Rebuild Remembered Sets Concurrently

Collection Cycle – Region does not get reclaimed



YO … Young-Only
IM … Initial-Mark



Rebuild Remembered Sets Concurrently

Key observations

- G1 maintains remembered sets all the time for all regions
  - Not required
    - Young regions: always
    - Old regions: only needed during Mixed GC
- Removing obsolete remembered set entries is costly
  - Create live data map
  - Remove remembered set entries during Cleanup

Rebuild Remembered Sets Concurrently

Solution

- Only keep required remembered sets when needed
  - For collection set regions only (<< all regions!)
  - Minimizes fragmentation
- Construct remembered sets concurrently between Remark and Cleanup
  - Instead of live data map calculation
  - No removal of obsolete remembered set entries during Cleanup pause

Rebuild Remembered Sets Concurrently

Side effects

- Lengthens time from Remark to Cleanup
  - Up to 30% longer marking cycles
  - Dynamic IHOP automatically adapts
- Improves Throughput and Pause Times
  - Less work outside of rebuild phase, creates dense remembered sets

Rebuild Remembered Sets Concurrently

Side effects

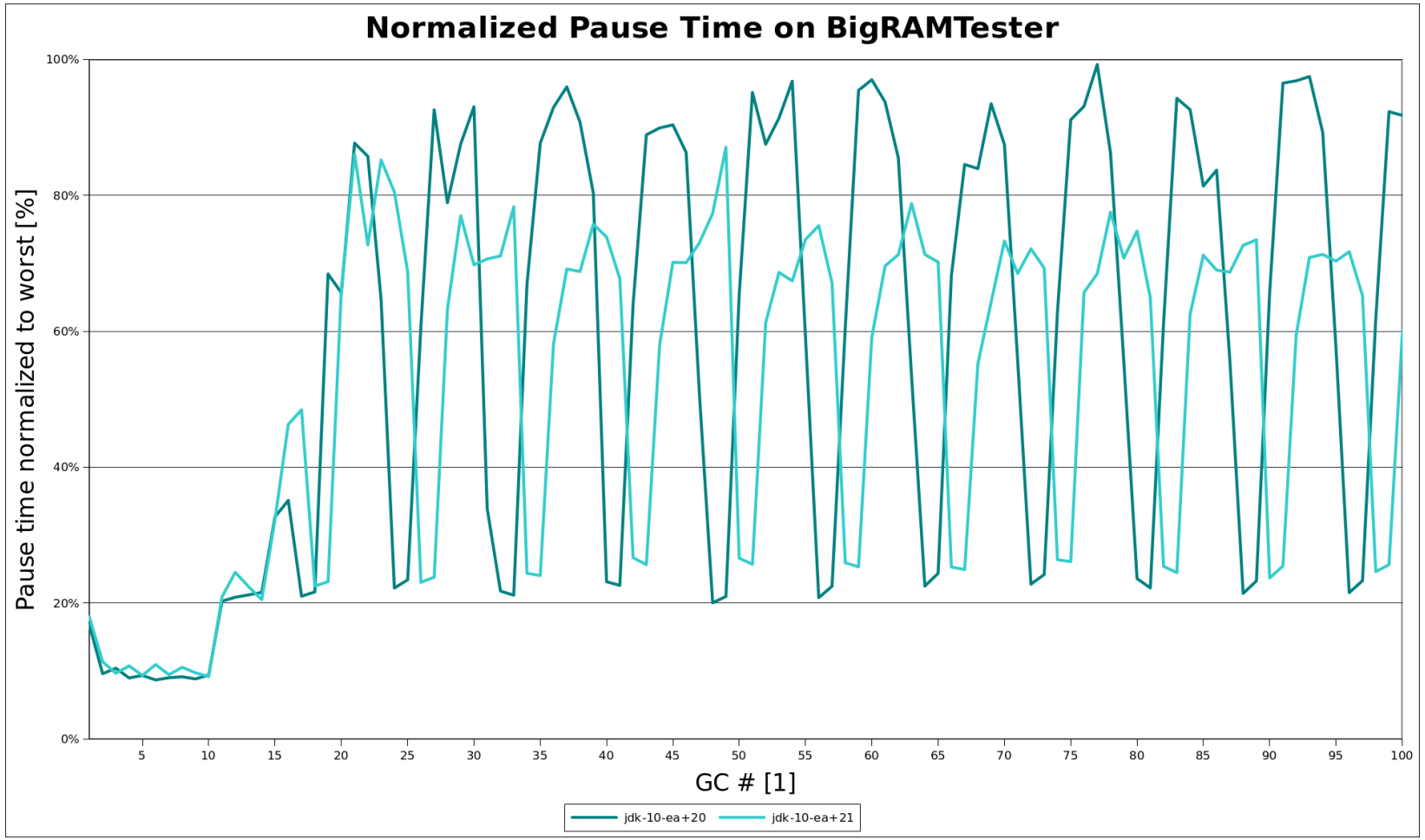
- Lengthens time from Remark to Cleanup
  - Up to 30% longer marking cycles
  - Dynamic IHOP automatically adapts
- Improves Throughput and Pause Times
  - Less work outside of rebuild phase, creates dense remembered sets
- Allows bounded remembered set memory usage
  - Just stop collecting remembered sets for some regions

Rebuild Remembered Sets Concurrently

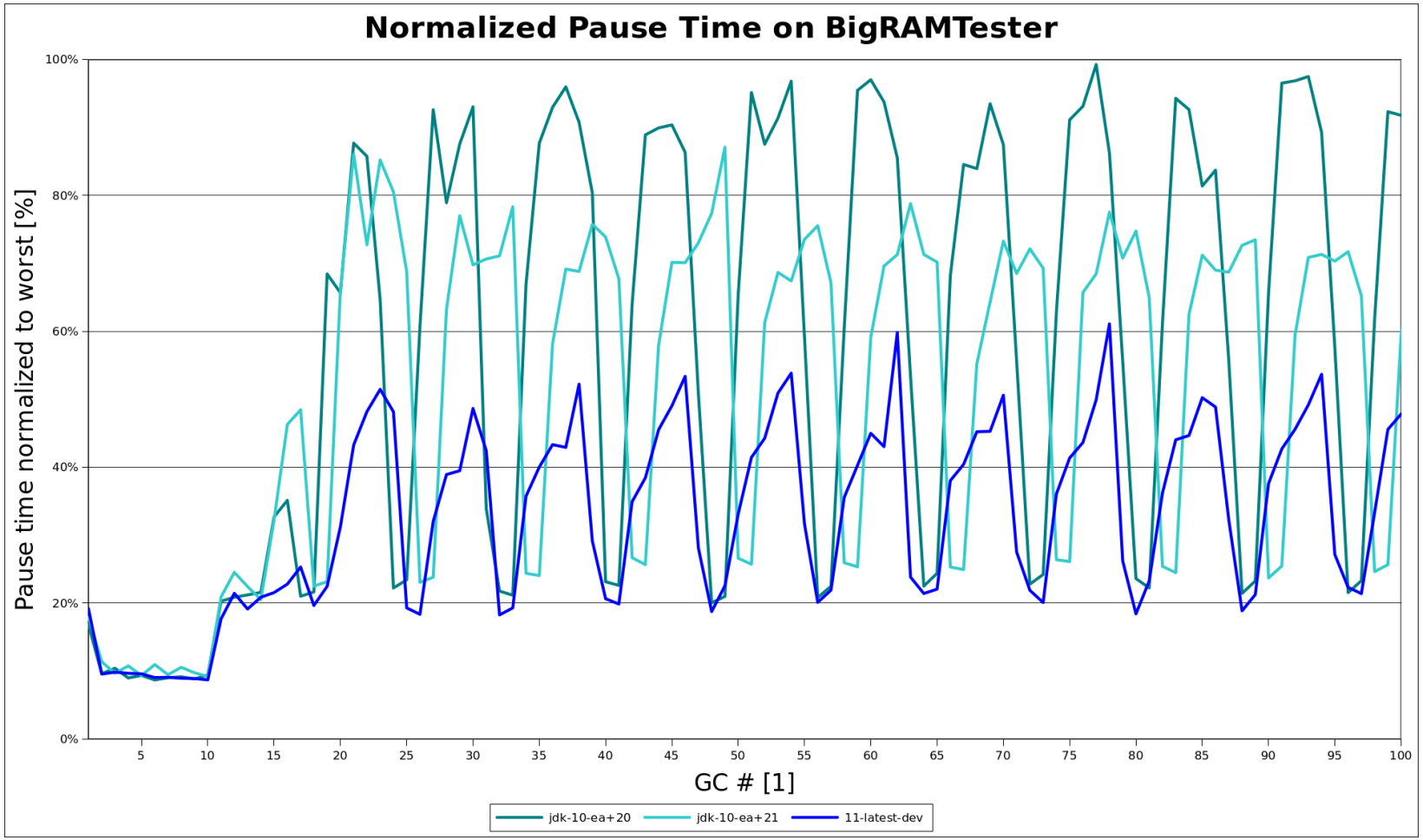
Remembered set memory usage on BigRAMTester

- Baseline:
  - ~10% of maximum heap size
- Current:
  - ~0.5% outside of rebuilding and mixed gc phase
  - ~7.5% after rebuilding 60% of the heap

Rebuild Remembered Sets Concurrently



Rebuild Remembered Sets Concurrently



Rebuild Remembered Sets Concurrently

- More information: JDK-8180415
- Work in progress

Program Agenda

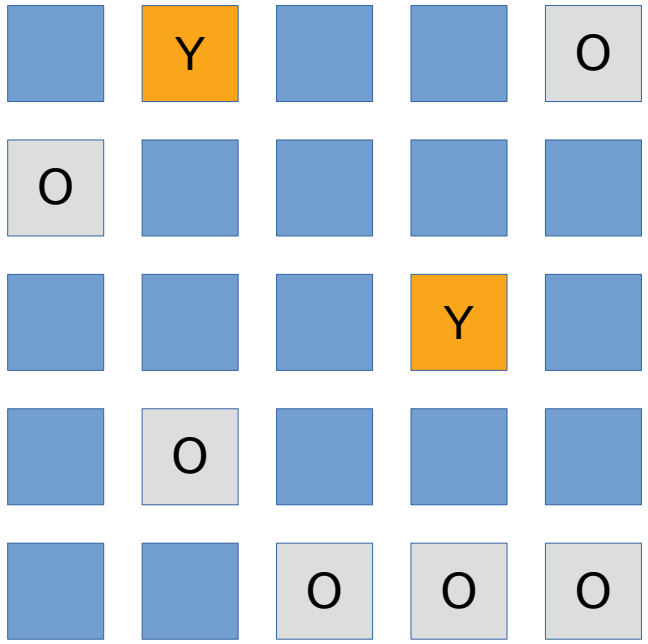
1. Parallel Full GC
2. Faster Card Scanning
3. Rebuild Remembered Concurrently
4. Abortable Mixed Collections
5. Automatic Thread Sizing
6. Participate!

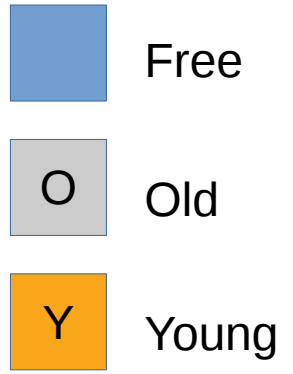
Abortable Mixed Collections

Problem

- G1 strives to keep pause time goal
  - Determines “Collection set” using predictions at the start of GC
- Particularly during Mixed collections predictions are hard
  - G1 mispredicts often
  - Significant effort to tune Mixed collection pauses

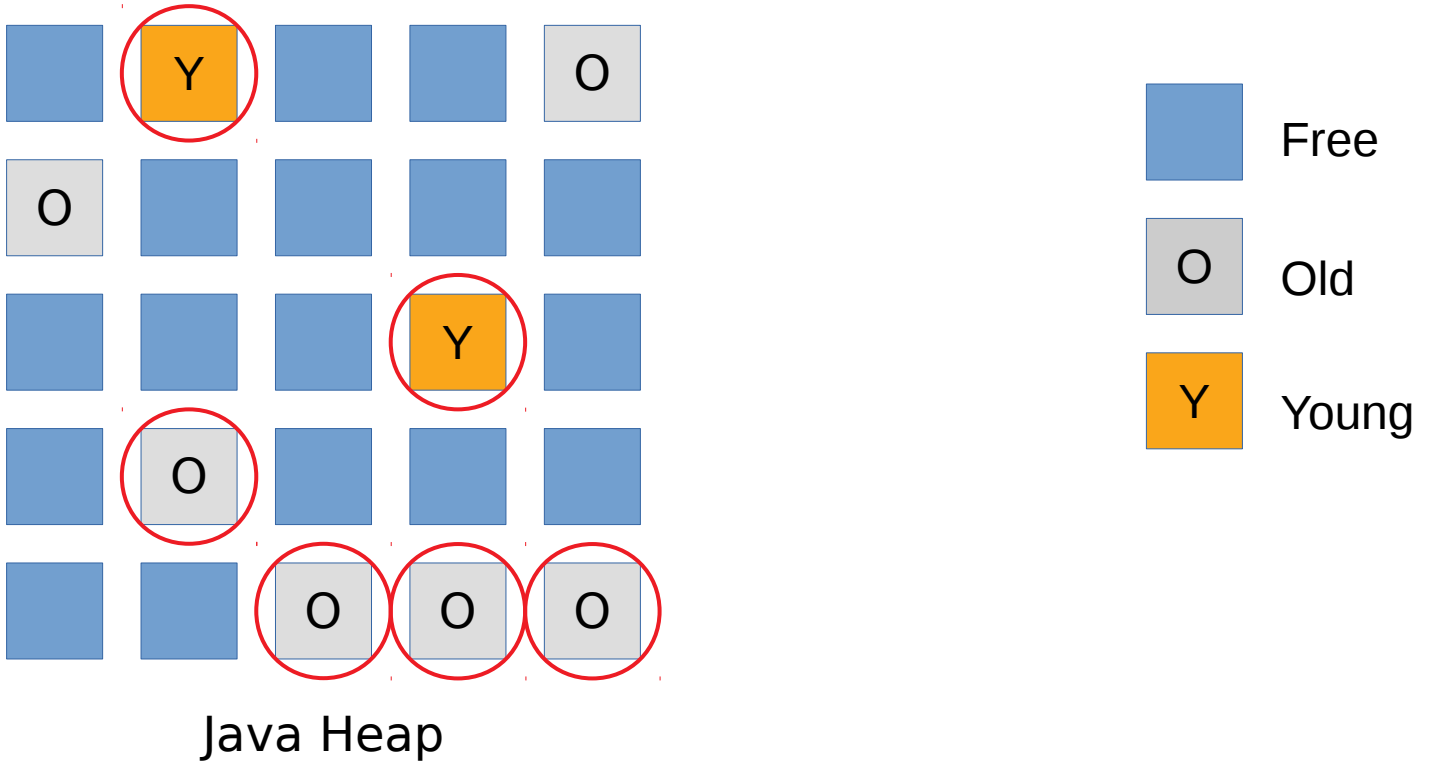
Abortable Mixed Collections





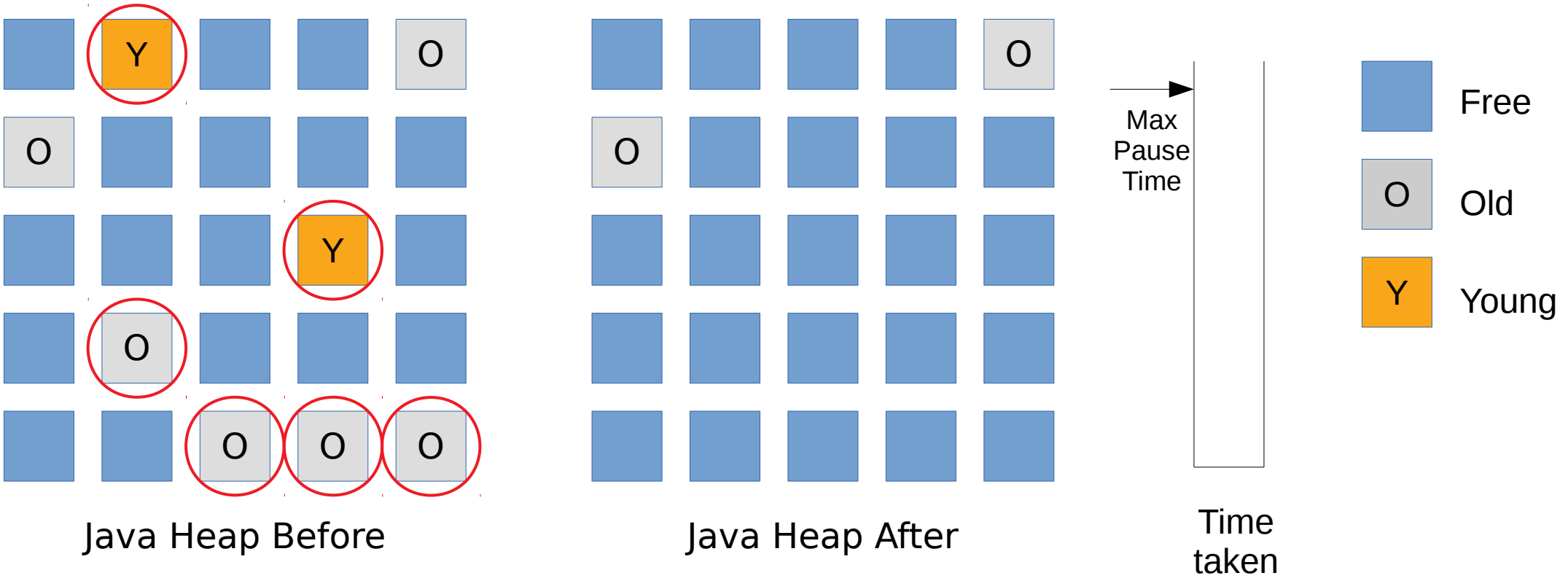
Abortable Mixed Collections

Collection set



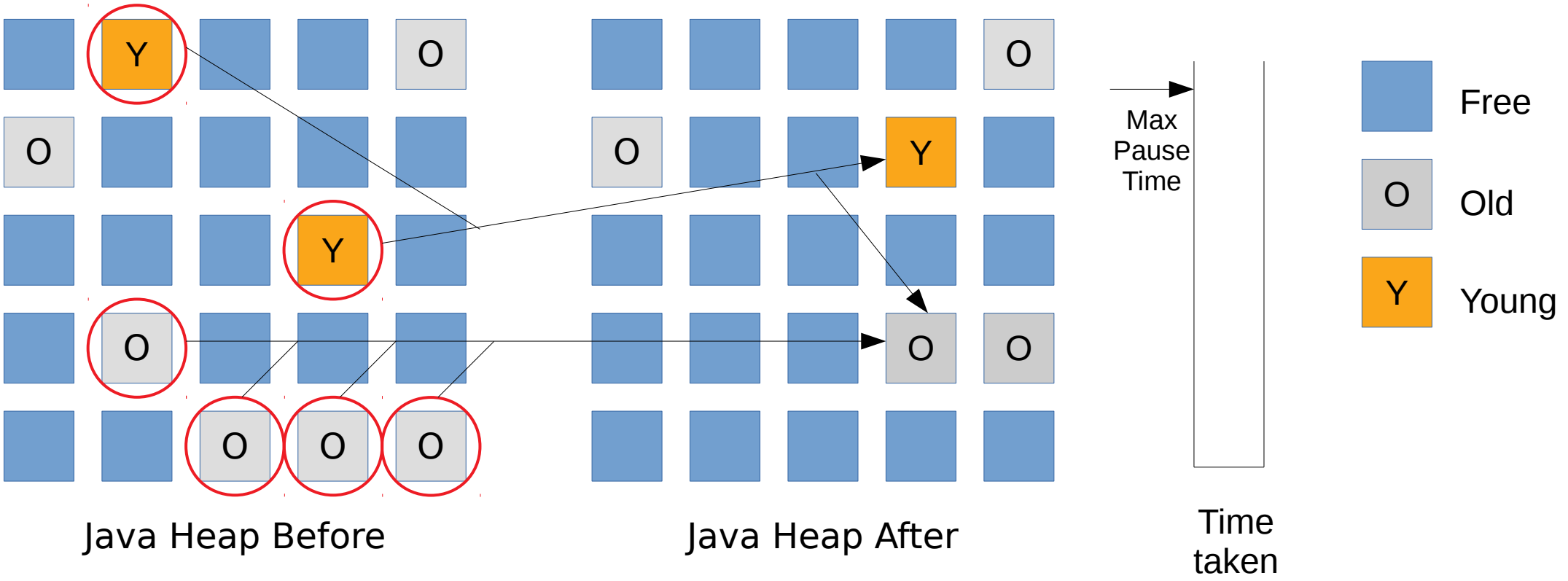
Abortable Mixed Collections

Current evacuation policy



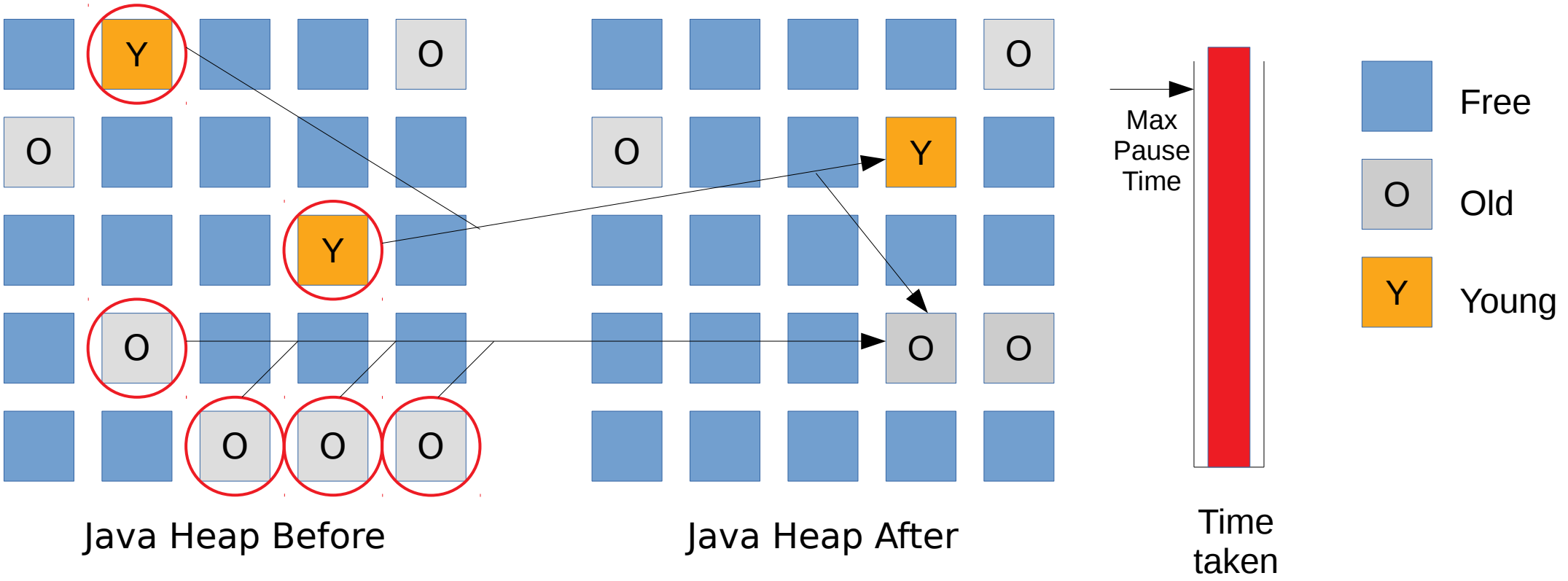
Abortable Mixed Collections

Current evacuation policy



Abortable Mixed Collections

Current evacuation policy exceeds pause time



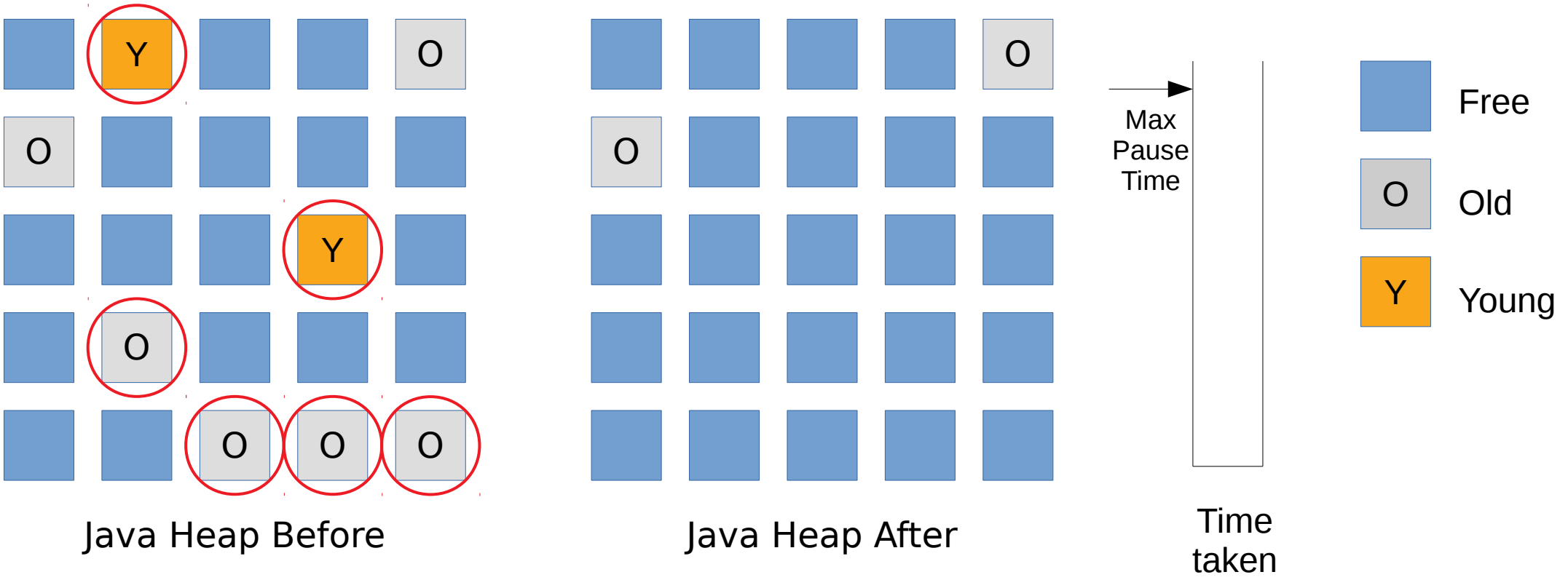
Abortable Mixed Collections

Solution

- Incrementally evacuate collection set
  - "Abort" evacuation if next increment would take too long

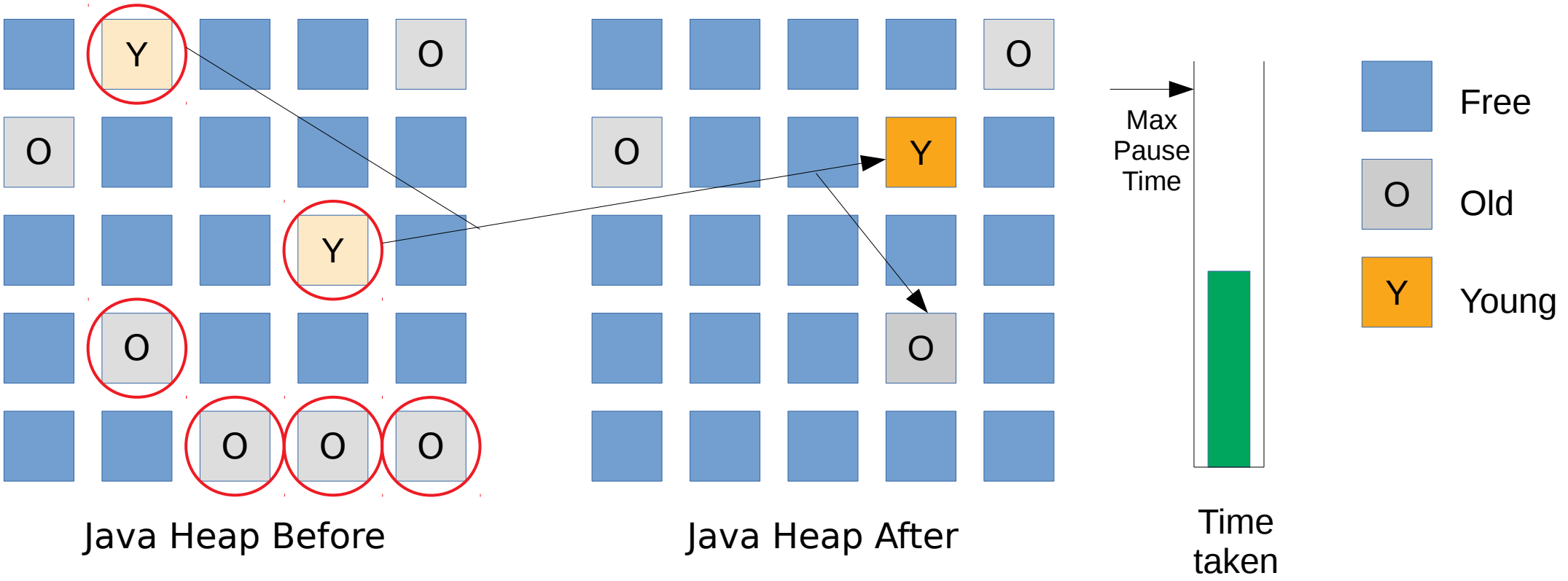
Abortable Mixed Collections

Try again, incrementally



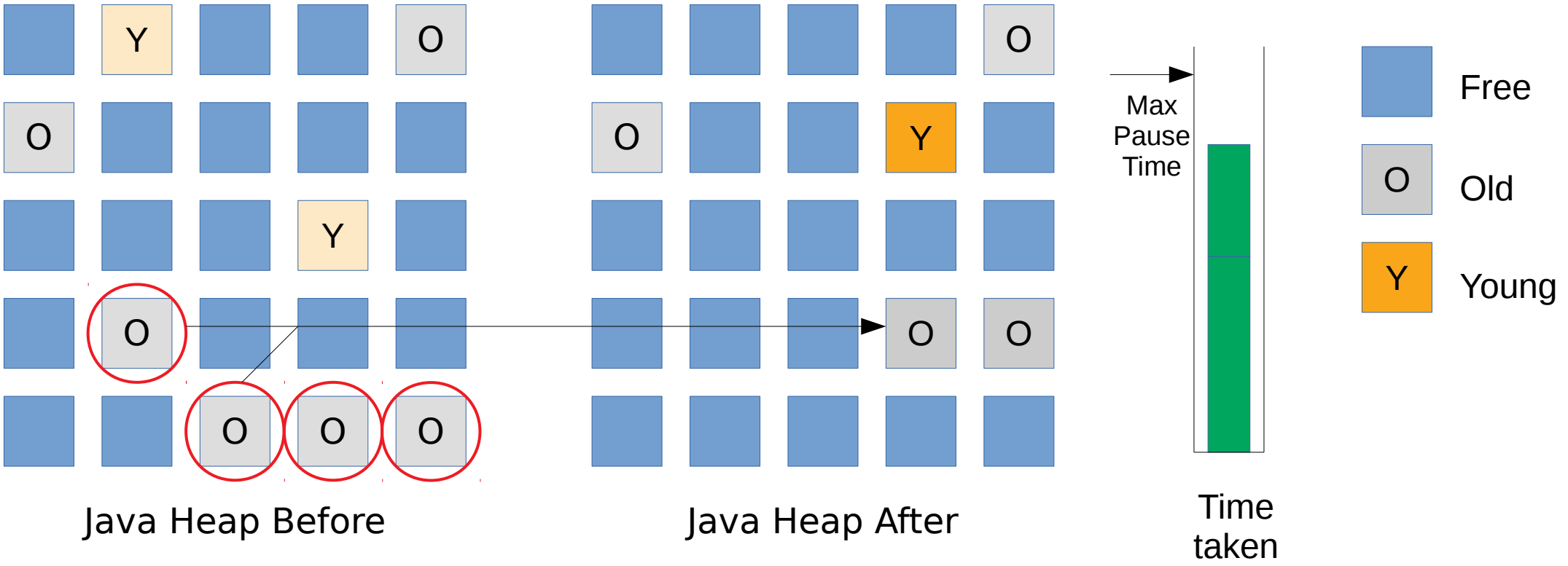
Abortable Mixed Collections

First Young regions as a whole



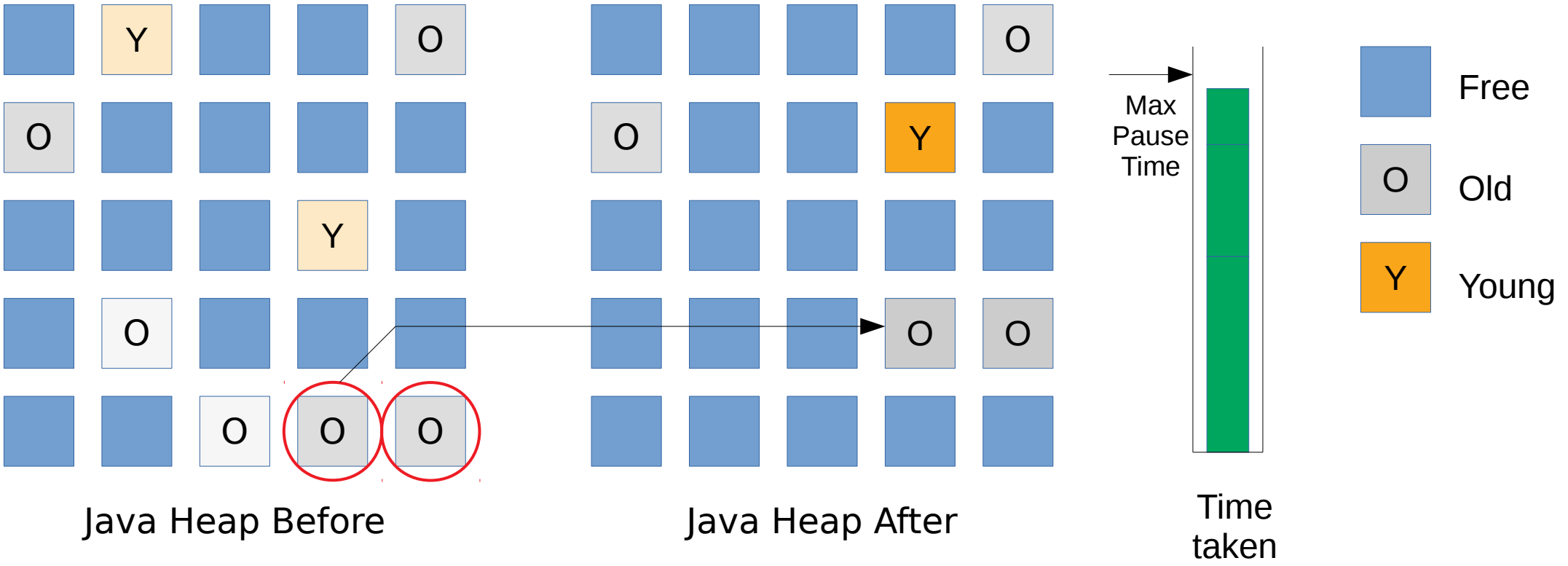
Abortable Mixed Collections

“Large” set of Old regions



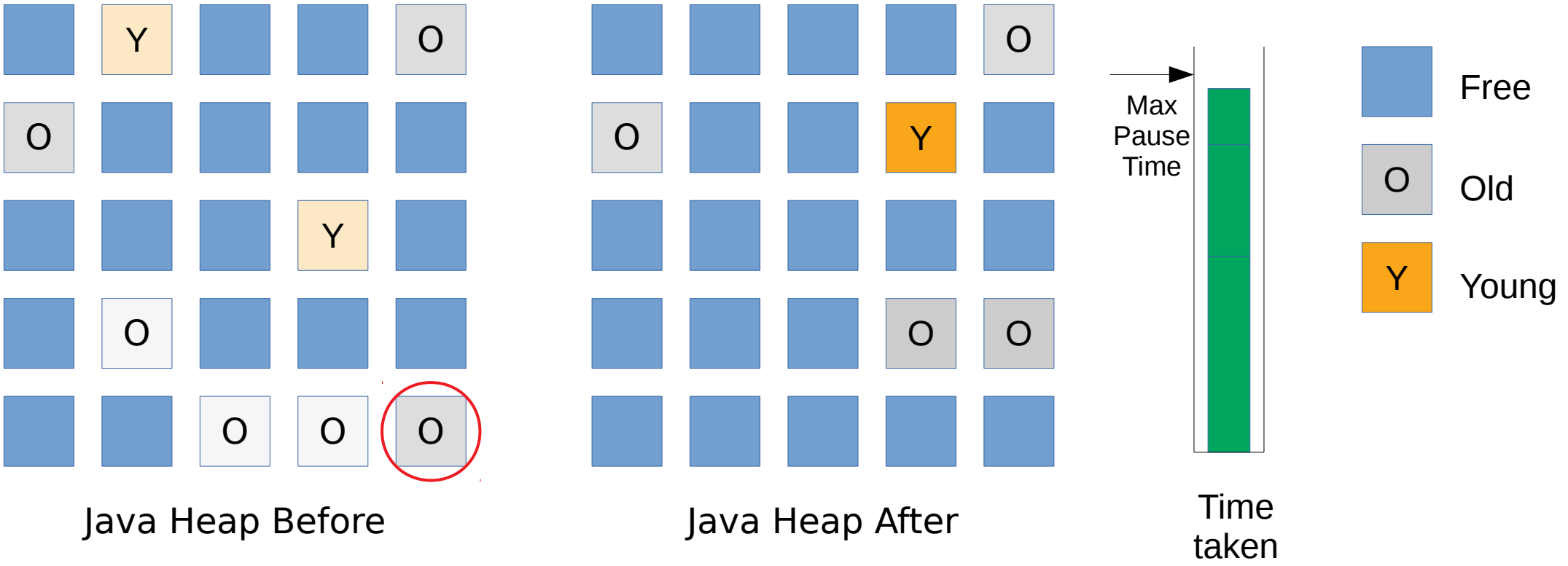
Abortable Mixed Collections

“Small” set of Old regions



Abortable Mixed Collections

“Abort”



Abortable Mixed Collections

- Enter “abortable mode” only if needed
  - To decrease overhead
- More information
  - JEP draft: Abortable mixed collections for G1 JDK-8190269
- Work in progress

Program Agenda

1. Parallel Full GC
2. Faster Card Scanning
3. Rebuild Remembered Concurrently
4. Abortable Mixed Collections
5. Automatic Thread Sizing
6. Participate!

Automatic Thread Sizing

Problem

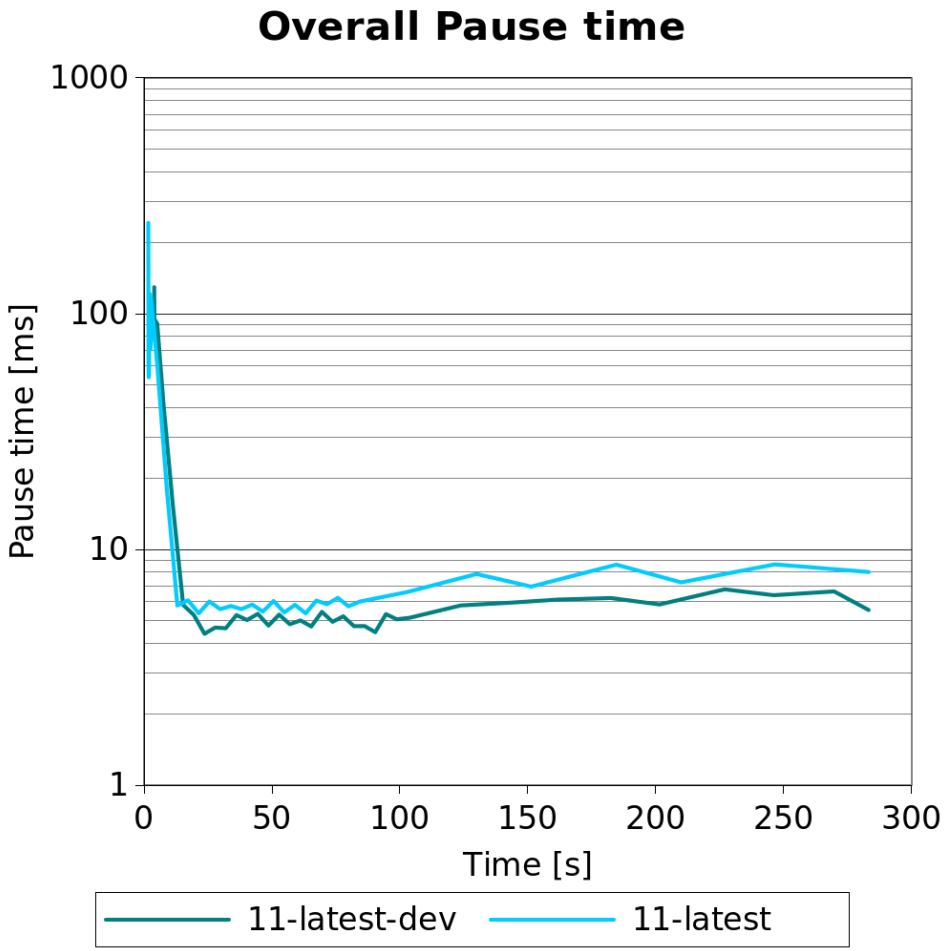
- Manually setting the right number of threads impossible
  - Or even not even desired
  - Lots of work as hardware, application, even application phase specific
  - Can only set number of threads statically for everything
    - e.g. -XX:ParallelGCThreads, -XX:ConcGCThreads, -XX:ParallelRefProcEnabled
- Benefits of using the right number of threads
  - Saves resources, faster startup
  - (Small pause time improvements)

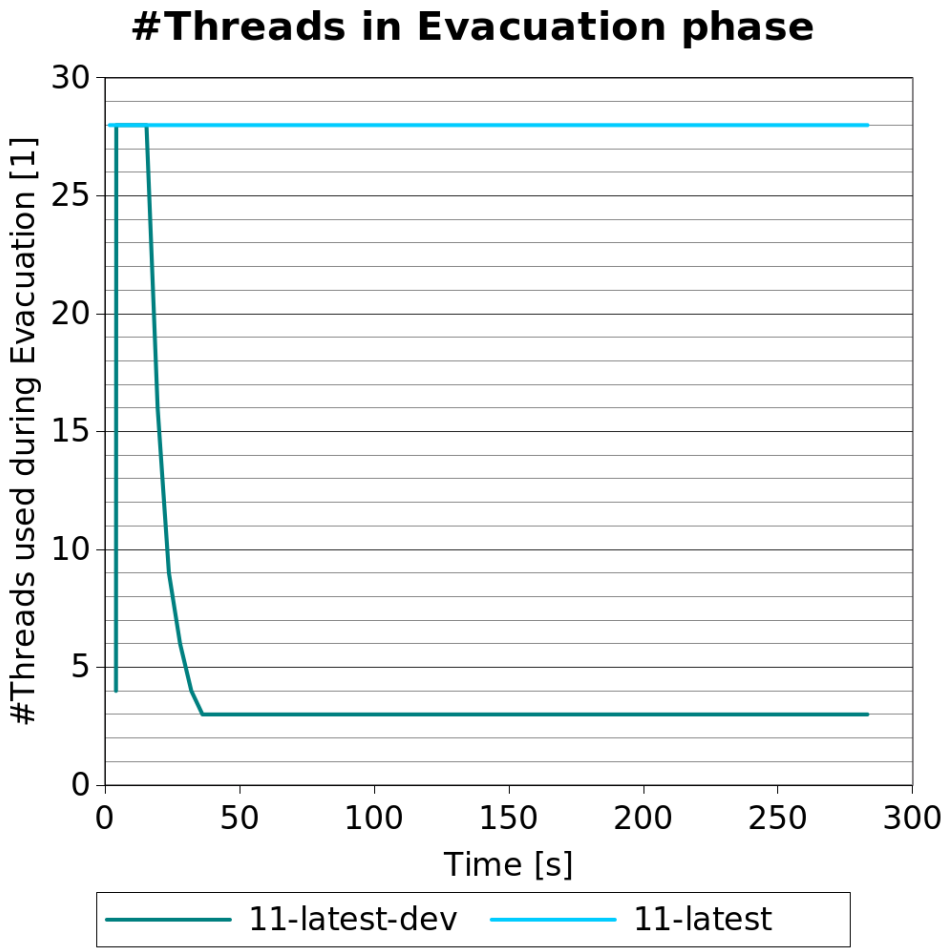
Automatic Thread Sizing

Solution

- Let G1 automatically decide the optimal number of threads
  - G1 already tracks lots of statistics about a GC
    - Actual bytes copied
    - References processed
    - Cards scanned
    - ...
  - Actually G1 already does that for a lot of phases since JDK9....

Automatic Thread Sizing - Example





Automatic Thread Sizing

- More information: JEP 308: G1 ergonomics JDK-8172792
- Work in progress

Program Agenda

1. Parallel Full GC
2. Faster Card Scanning
3. Rebuild Remembered Sets On The Fly
4. Abortable Mixed GC
5. Automatic Thread Sizing
6. Participate!

Participate!

- Hang out on hotspot-gc-use@openjdk.java.net
  - Provide answers to community
- Fix small bugs
  - Bugs labeled “starter”/”cleanup” on Hotspot GC component at https://bugs.openjdk.java.net
  - Discuss at hotspot-gc-dev@openjdk.java.net
- Interesting larger projects

Participate! - Larger projects

- NMethod barriers
- Throughput barriers
- NUMA support

Participate! – NMethod barriers

- NMethod barriers
  - Small piece of code that is run before NMethod is entered
    - Could be used to disable pre-barrier when not in use
    - Most of the time!

```
  ...                                  ...
  cmpb ofs(%tls), 0                    nop
  jz NoPreBarrier                      nop
  call slow_path                       nop
NoPreBarrier:                          ...
  ...
```

Participate! – Throughput barriers

- Throughput barriers
  - G1 has throughput deficiencies
  - Mostly write barrier related
- Use Parallel GC barrier instead of large G1 barrier
  - Increases throughput
  - May or may not have some impact on latency/pause time
- More information: FOSDEM 2017 talk
  Three ideas for the G1 GC (and how to get involved)

Participate! - NUMA support

- NUMA support
  - Improve throughput on large multi-socket machines
  - Exploit memory locality
  - JEP 137 open for 6 years now (JDK-8046147)

Questions?

(See you on hotspot-gc-use/dev@openjdk.java.net)

(thomas.schatzl@oracle.com)