geecon

G1 Garbage Collector Details and Tuning

Simone Bordet
sbordet@intalio.com

Who Am I

- **Simone Bordet**
  - sbordet@intalio.com - @simonebordet
- **Lead Architect at Intalio/Webtide**
  - Jetty's HTTP/2, SPDY and HTTP client maintainer
- **Open Source Contributor**
  - Jetty, CometD, MX4J, Foxtrot, LiveTribe, JBoss, Larex
- **CometD project leader**
  - Web messaging framework
- **JVM tuning expert**

G1
Overview

Simone Bordet
sbordet@intalio.com

G1 Overview

- **G1 is the HotSpot low-pause collector**
  - First papers date back to 2004
  - Available and supported since JDK 7u4 (April 2012)
- **Long term replacement for CMS**
- **Scheduled to be the default GC for JDK 9**
  - JEP 248: http://openjdk.java.net/jeps/248
- **Low pauses valued more than max throughput**
  - For majority of Java Apps
  - For the others, ParallelGC will still be available

Simone Bordet
sbordet@intalio.com

G1 Overview

- G1 is designed to be really easy to tune
- `java -Xmx32G -XX:MaxGCPauseMillis=100 ...`
- Tuning based on max Stop-The-World pause
  - `-XX:MaxGCPauseMillis=<>`
  - By default 250 ms

Simone Bordet
sbordet@intalio.com

G1 Overview

- **G1 is a generational collector**
- **G1 implements 2 GC algorithms**
- **Young Generation GC**
  - Stop-The-World, Parallel, Copying
- **Old Generation GC**
  - Mostly-concurrent marking
  - Incremental compaction
    - Piggybacked on Young Generation GC

Simone Bordet
sbordet@intalio.com

G1 Logging

- **G1 has a very detailed logging**
  - Keep it ALWAYS enabled !

- **-XX:+PrintGCDateStamps**
  - Prints date and uptime
- **-XX:+PrintGCDetails**
  - Prints G1 Phases
- **-XX:+PrintAdaptiveSizePolicy**
  - Prints ergonomic decisions
- **-XX:+PrintTenuringDistribution**
  - Print aging information of survivor regions

Simone Bordet
sbordet@intalio.com

G1
Memory Layout

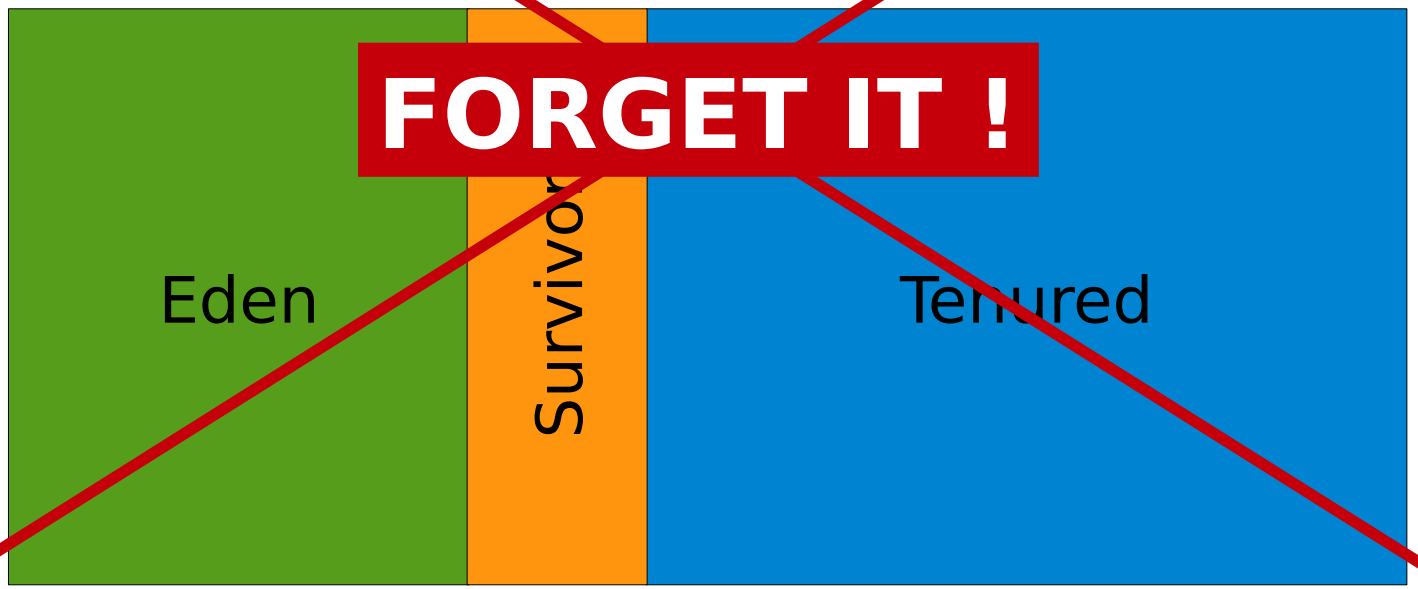
Simone Bordet
sbordet@intalio.com

G1 Memory Layout

- **Familiar with this ?**
  - Eden
  - Survivor
  - Tenured

Young Generation

Old Generation



Simone Bordet
sbordet@intalio.com

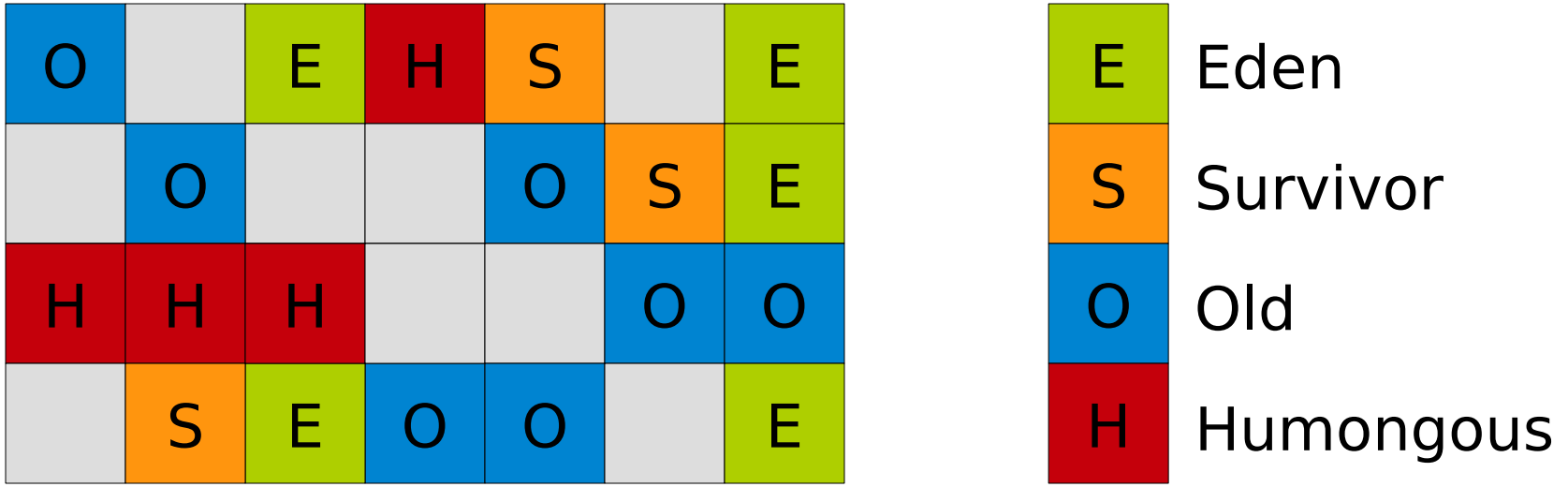
G1 Memory Layout

- **G1 divides the heap into small “regions”**
- **Targets 2048 regions**
  - Tuned via `-XX:G1HeapRegionSize=<>`
- **Eden, Survivor, Old regions**
- **“Humongous” regions**
  - When a single object occupies > 50% of the region
  - Typically `byte[]` or `char[]`

Simone Bordet
sbordet@intalio.com

G1 Memory Layout

G1 Memory Layout



Simone Bordet
sbordet@intalio.com

G1 Young GC

- JVM starts, G1 prepares Eden regions
- Application runs and allocates into Eden regions
- Eden regions fill up
- When all Eden regions are full → Young GC

Simone Bordet
sbordet@intalio.com

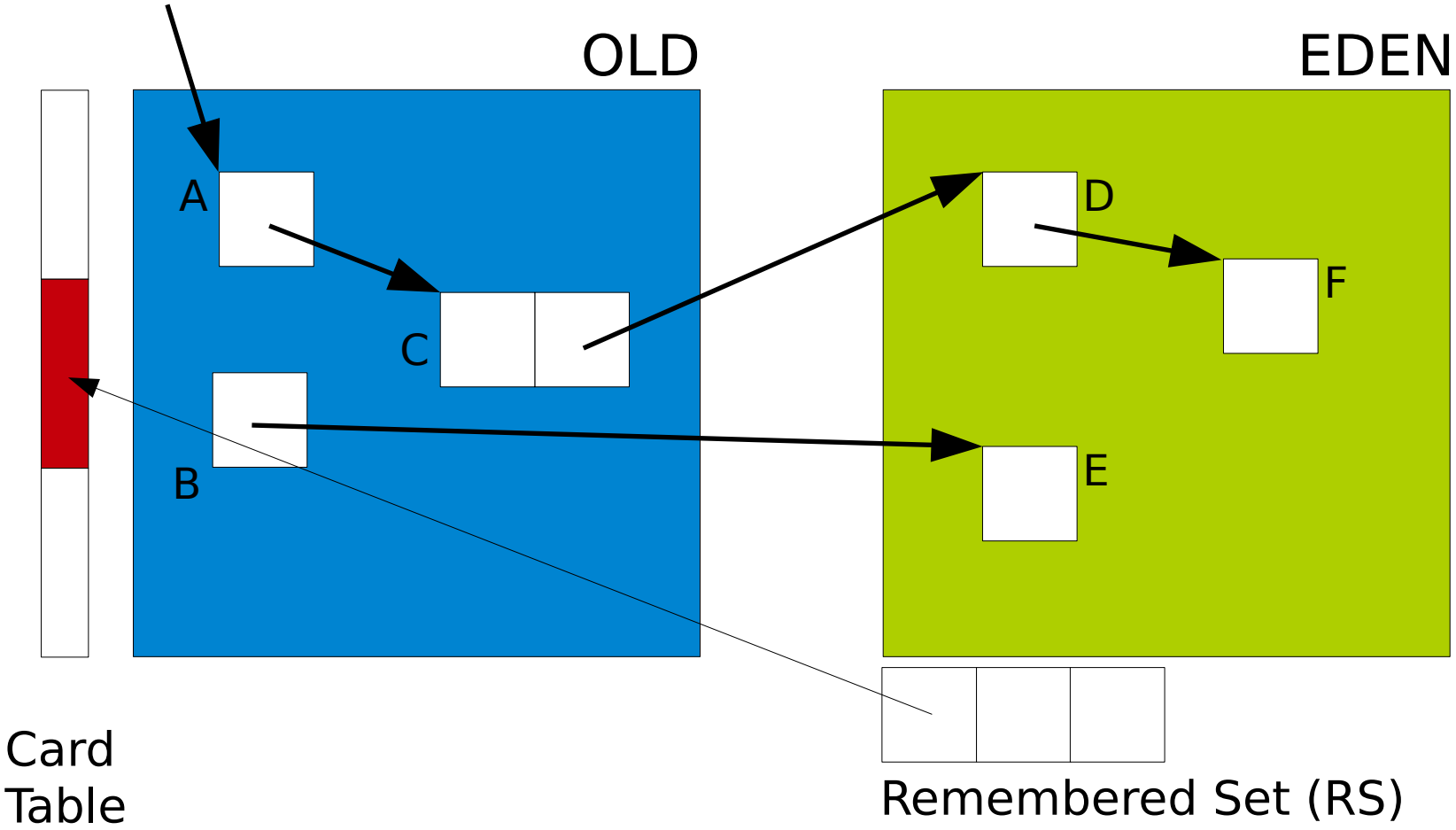
G1 Young GC

- **The application does not only allocates**
- **Application modifies pointers of existing objects**
- **An “old” object may point to an “eden” object**
  - E.g. an “old” Map has just been put() a new entry
- **G1 must track these inter-generation pointers**
  - (Old | Humongous) → (Eden | Survivor) pointers

Simone Bordet
sbordet@intalio.com

G1 Young GC

Inter-generation references



Simone Bordet
sbordet@intalio.com

G1 Remembered Set

- **A write barrier tracks pointer updates**
- `object.field = <reference>`
- **Triggers every time a pointer is written**
  - Records the write information in the card
  - Cards are stored in a queue (dirty card queue)
  - The queue is divided in 4 zones: white, green, yellow, red

Simone Bordet
sbordet@intalio.com

G1 Remembered Set

- **White zone**
  - Nothing happens, buffers are left unprocessed
- **Green zone** (`-XX:G1ConcRefinementGreenZone=<>`)
  - Refinements threads are activated
  - Buffers are processed and the queue drained
- **Yellow zone** (`-XX:G1ConcRefinementYellowZone=<>`)
  - All available refinement threads are active
- **Red zone** (`-XX:G1ConcRefinementRedZone=<>`)
  - Application threads process the buffers

Simone Bordet
sbordet@intalio.com

G1
Young GC
Phases

Simone Bordet
sbordet@intalio.com

G1 Young GC Phases

- **G1 Stops The World**
- **G1 builds a “collection set”**
  - The regions that will be subject to collection
- **In a Young GC, the collection set contains:**
  - Eden regions
  - Survivor regions

Simone Bordet
sbordet@intalio.com

G1 Young GC Phases

- **First phase: “Root Scanning”**
  - Static and local objects are scanned
- **Second phase: “Update RS”**
  - Drains the dirty card queue to update the RS
- **Third phase: “Process RS”**
  - Detect the Eden objects pointed by Old objects

Simone Bordet
sbordet@intalio.com

G1 Young GC Phases

- **Fourth phase: “Object Copy”**
  - The object graph is traversed
  - Live objects copied to Survivor/Old regions
- **Fifth phase: “Reference Processing”**
  - Soft, Weak, Phantom, Final, JNI Weak references
  - Always enable **`-XX:+ParallelRefProcEnabled`**
  - More details with **`-XX:+PrintReferenceGC`**

Simone Bordet
sbordet@intalio.com

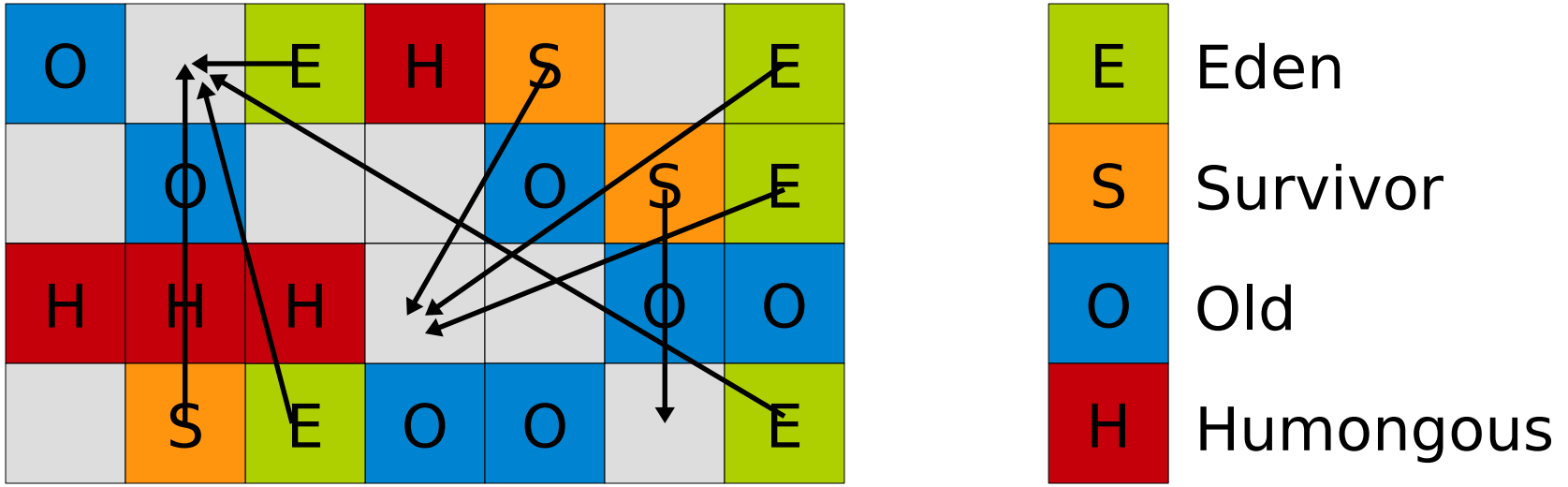
G1 Young GC Phases

- **G1 tracks phase times to autotune**
- **Phase timing used to change the # of regions**
  - Eden region count
  - Survivor region count
- **Updating the # of regions**
  - Respect of max pause target
- **Typically, the shorter the pause target, the smaller the # of Eden regions**

Simone Bordet
sbordet@intalio.com

G1 Young GC Phases

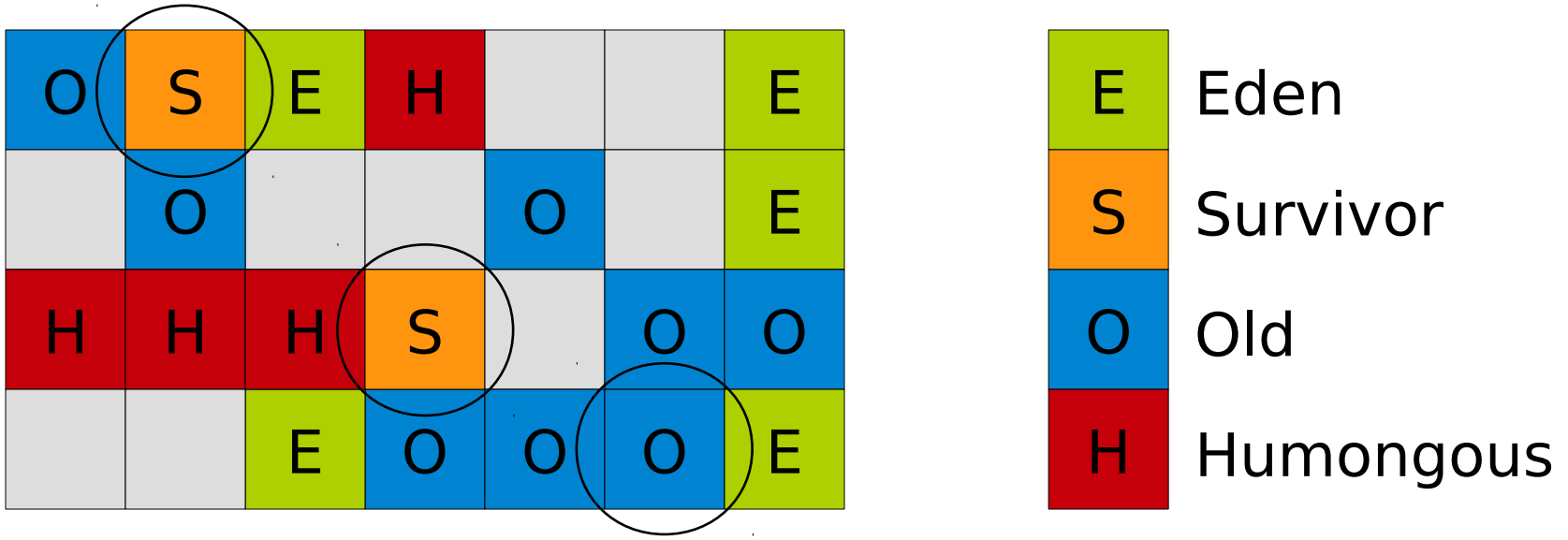
- **G1 Young GC**
  - E and S regions evacuated to new S and O regions



Simone Bordet
sbordet@intalio.com

G1 Young GC Phases

G1 Young GC



Simone Bordet
sbordet@intalio.com

G1
Old GC

Simone Bordet
sbordet@intalio.com

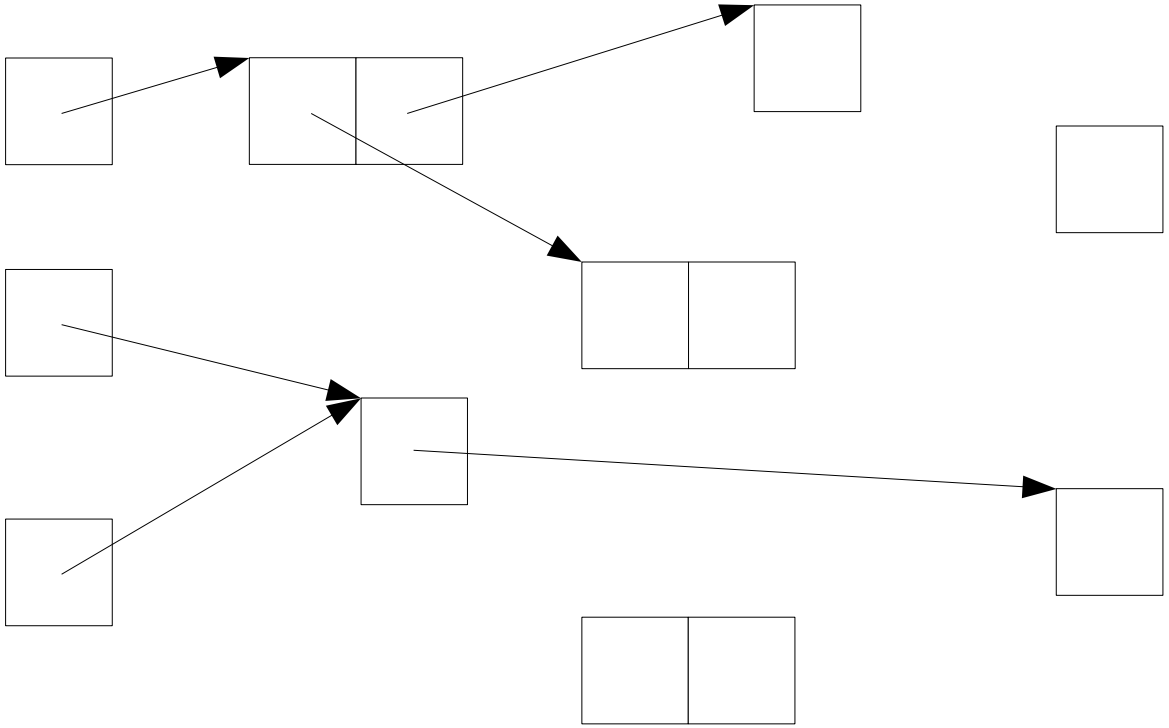
G1 Old GC

- **G1 schedules an Old GC based on heap usage**
  - By default when the entire heap is 45% full
    - Checked after a Young GC or a humongous allocation
  - Tunable via `-XX:InitiatingHeapOccupancyPercent=<>`
- **The Old GC consists of old region marking**
  - Finds all the live objects in the old regions
  - Old region marking is concurrent

Simone Bordet
sbordet@intalio.com

G1 Old GC

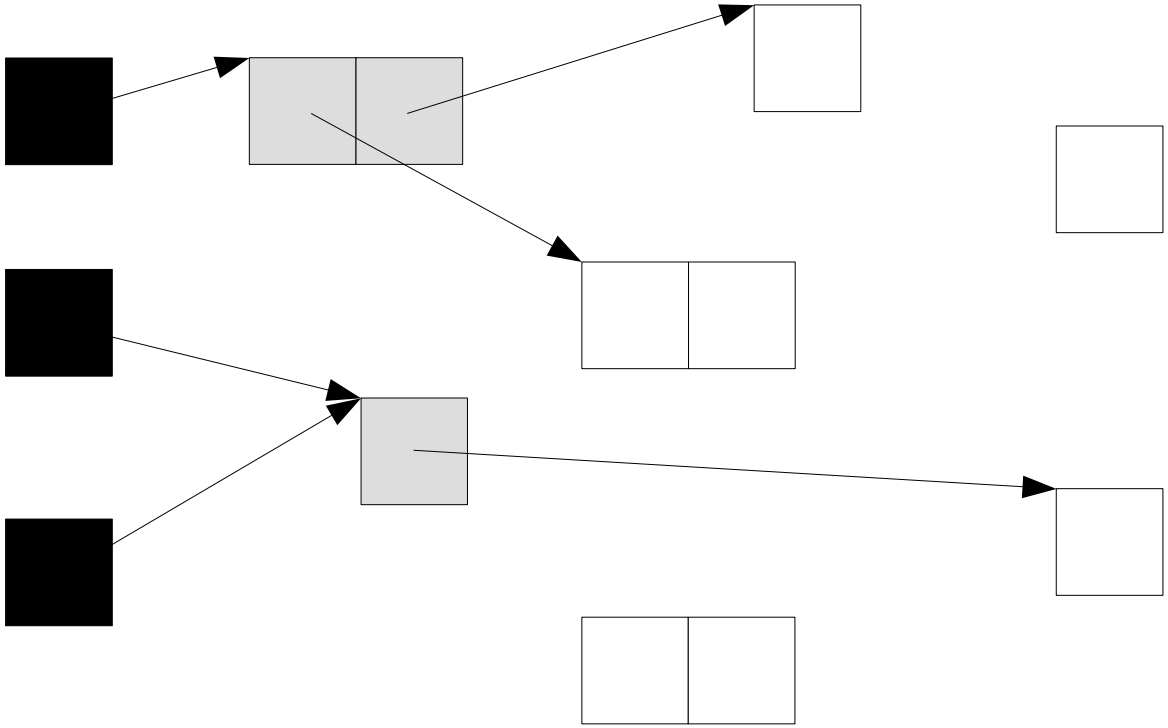
- **Concurrent marking**
  - Tri-color marking



Simone Bordet
sbordet@intalio.com

G1 Old GC

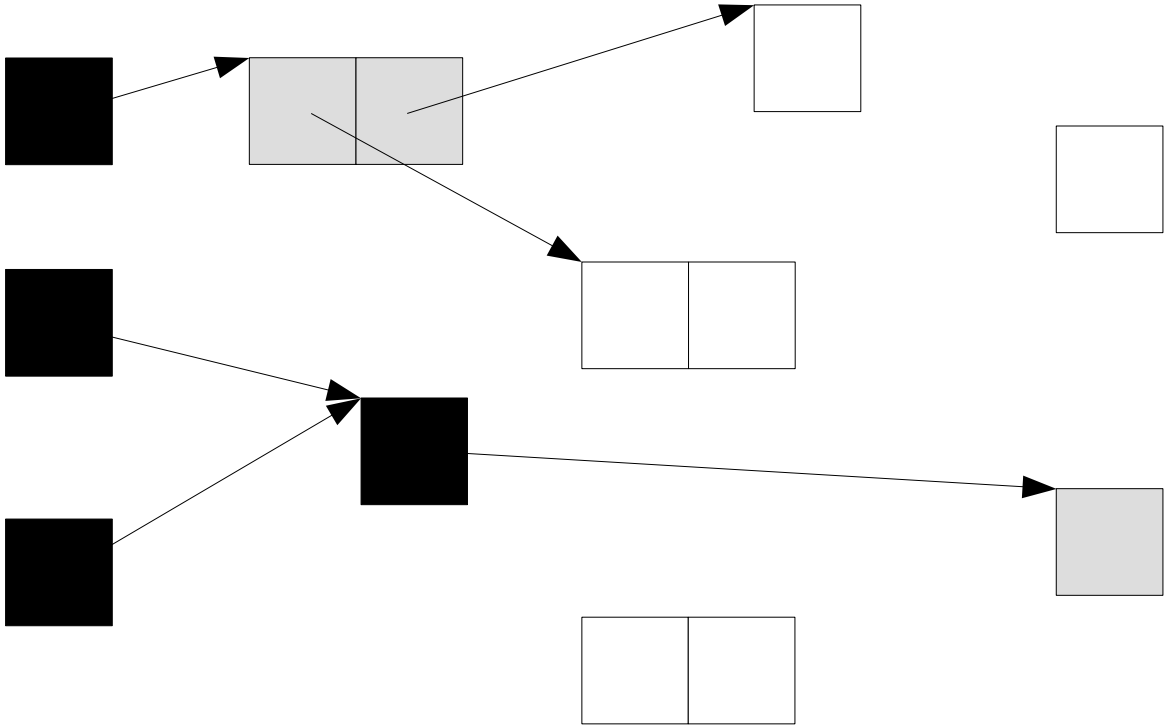
- **Concurrent marking**
  - Roots marked black – children marked gray



Simone Bordet
sbordet@intalio.com

G1 Old GC

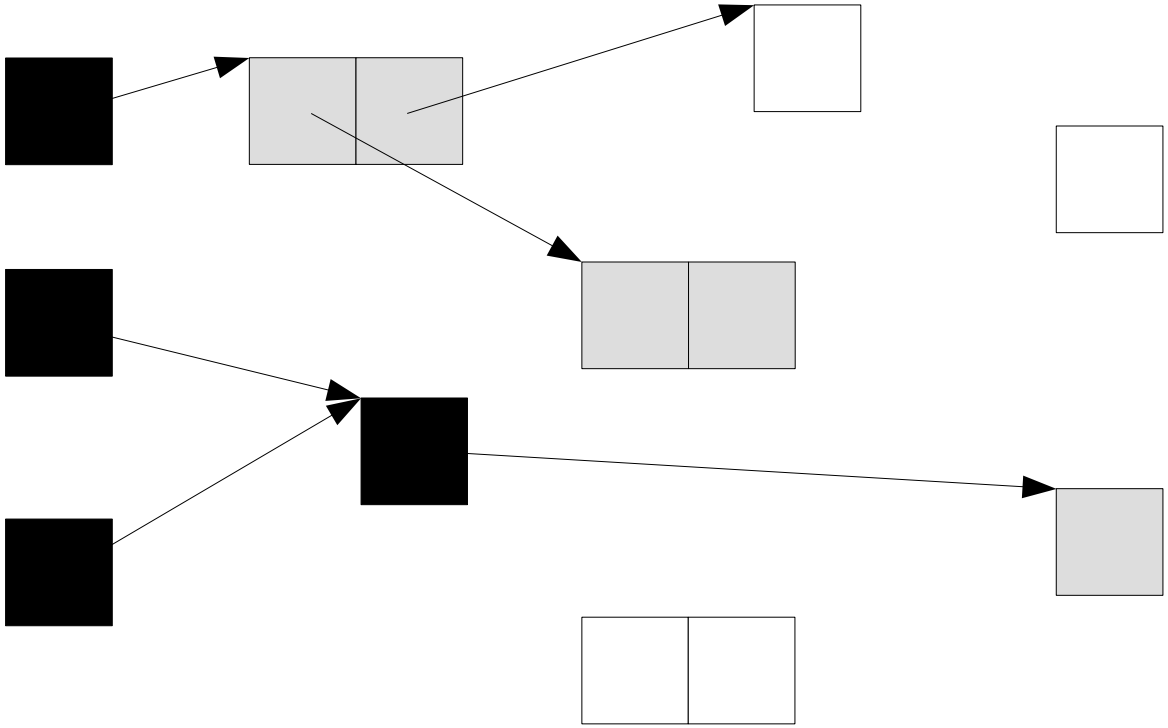
- **Concurrent marking**
  - Gray wavefront advances



Simone Bordet
sbordet@intalio.com

G1 Old GC

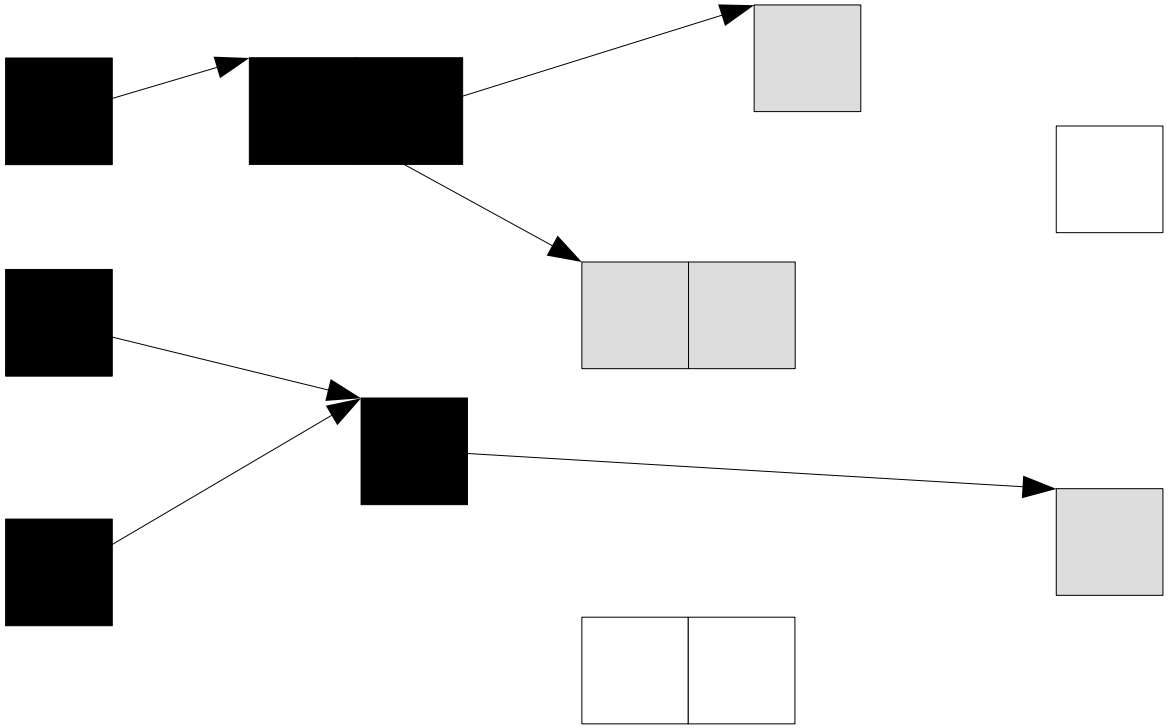
- **Concurrent marking**
  - Gray wavefront advances



Simone Bordet
sbordet@intalio.com

G1 Old GC

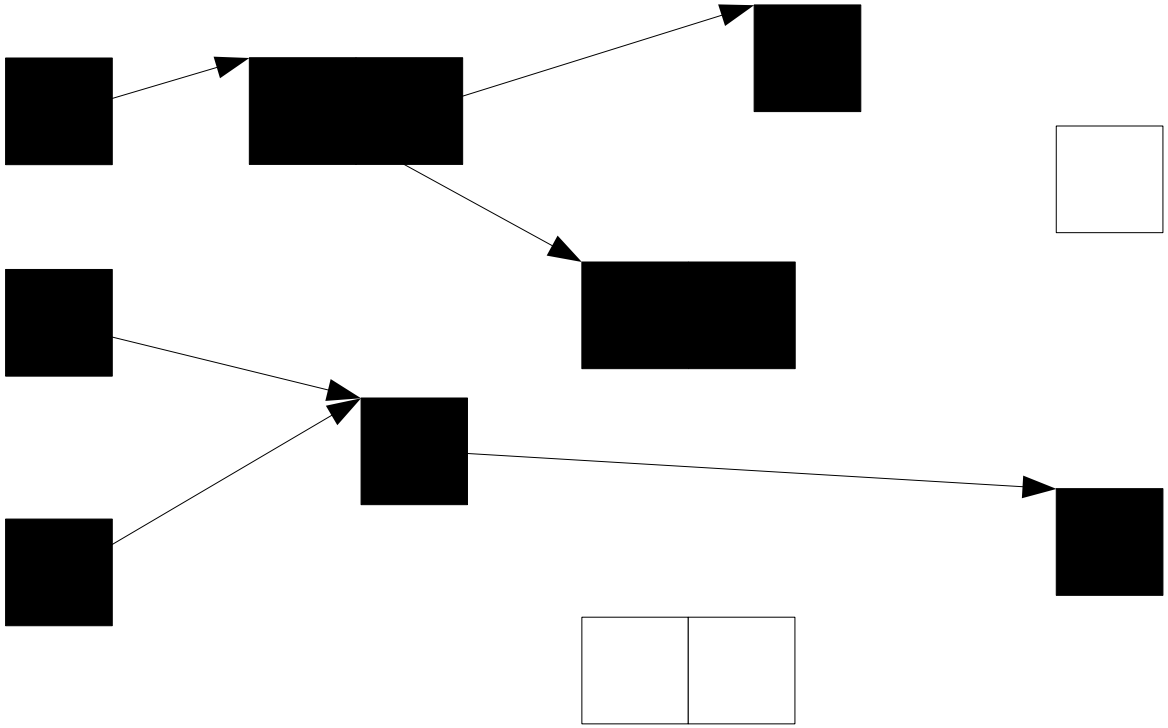
- **Concurrent marking**
  - Gray wavefront advances



Simone Bordet
sbordet@intalio.com

G1 Old GC

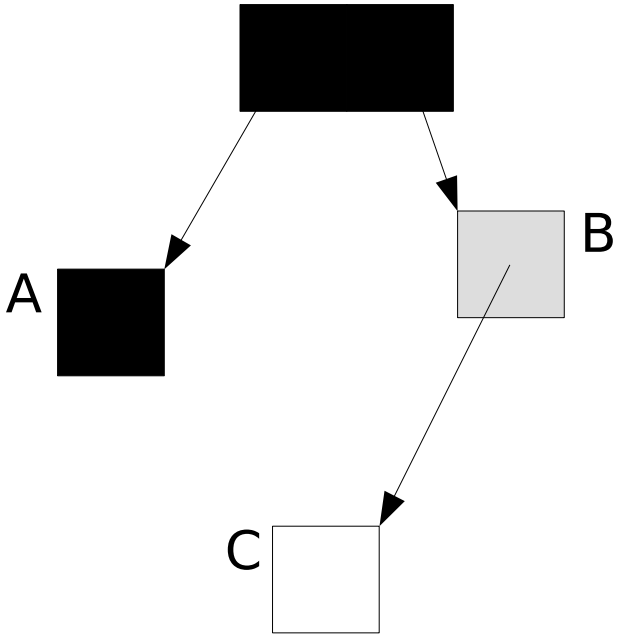
- **Concurrent marking**
  - Marking complete – white objects are garbage



Simone Bordet
sbordet@intalio.com

G1 Old GC

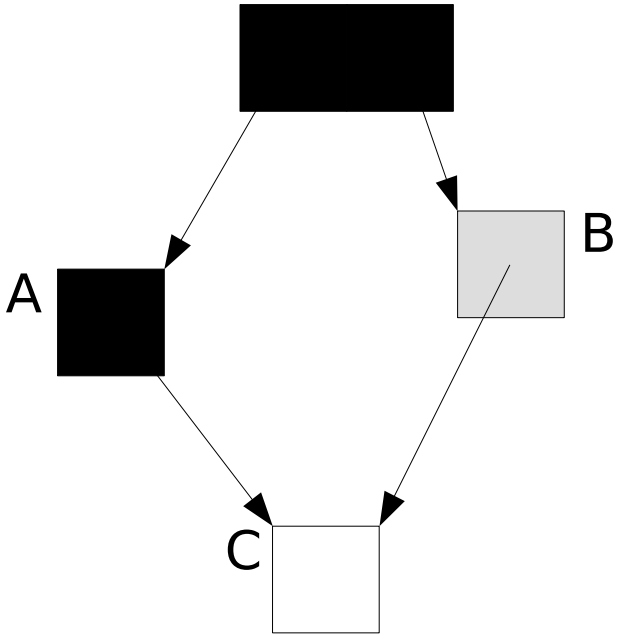
- **Concurrent marking: Lost Object Problem**
  - Marking in progress



Simone Bordet
sbordet@intalio.com

G1 Old GC

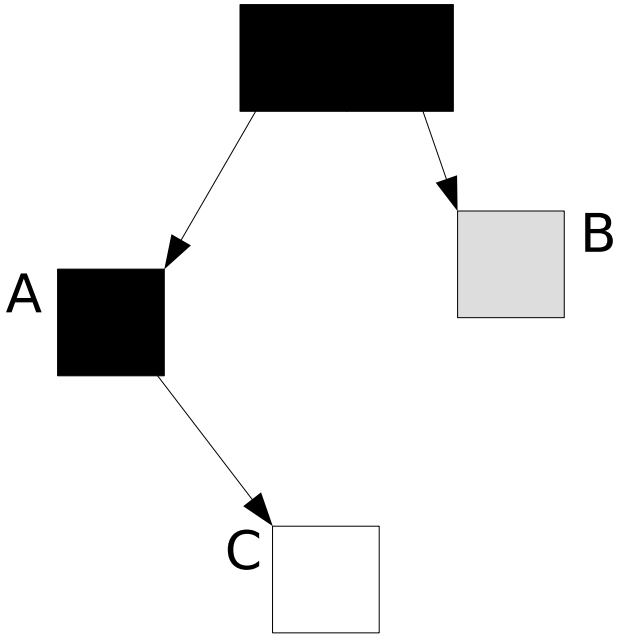
- **Concurrent marking: Lost Object Problem**
  - Application write: `A.c = C;`



Simone Bordet
sbordet@intalio.com

G1 Old GC

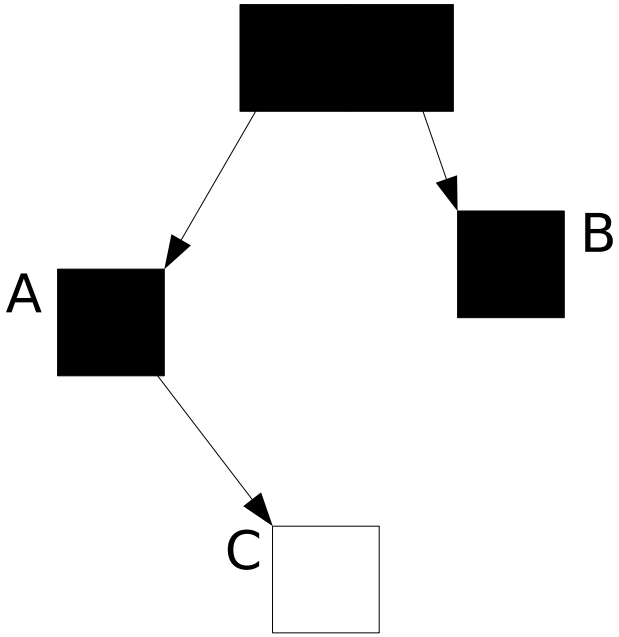
- **Concurrent marking: Lost Object Problem**
  - Application write: `B.c = null;`



Simone Bordet
sbordet@intalio.com

G1 Old GC

- **Concurrent marking: Lost Object Problem**
  - Marking completes



Simone Bordet
sbordet@intalio.com

G1 Old GC

- **G1 uses a write barrier to detect: `B.c = null;`**
  - More precisely that a pointer to C has been deleted
- **G1 now knows about object C**
  - Speculates that object C will remain alive
- **Snapshot-At-The-Beginning (SATB)**
  - Preserves the object graph that was live at marking start
  - C is queued and processed during remark
  - May retain floating garbage, collected the next cycle

Simone Bordet
sbordet@intalio.com

G1
Old GC
Phases

Simone Bordet
sbordet@intalio.com

G1 Old GC Phases

- **G1 Stops The World**
- **Performs a Young GC**
  - Piggybacks Old region roots detection (initial-mark)
- **G1 resumes application threads**
- **Concurrent Old region marking proceeds**
  - Keeps track of references (soft, weak, etc.)
  - Computes per-region liveness information

Simone Bordet
sbordet@intalio.com

G1 Old GC Phases

- **G1 Stops The World**
- **Remark phase**
  - SATB queue processing
  - Reference processing
- **Cleanup phase**
  - Empty old regions are immediately recycled
- **Application threads are resumed**

Simone Bordet
sbordet@intalio.com

G1 Old GC Phases

- [GC pause (G1 Evacuation Pause) (young) (initial-mark)
- [GC concurrent-root-region-scan-start]
- [GC concurrent-root-region-scan-end, 0.0566056 secs]
- [GC concurrent-mark-start]
- [GC concurrent-mark-end, 1.6776461 secs]
- [GC remark, 0.0488021 secs]
- [GC cleanup 16G->14G(32G), 0.0557430 secs]

Simone Bordet
sbordet@intalio.com

G1 Old GC Phases

- **Cleanup phase → recycles empty old regions**
- **What about non-empty old regions ?**
  - How is fragmentation resolved ?
- **Non-empty Old regions processing**
  - Happens during the next Young GC cycle
  - No rush to clean the garbage in Old regions

Simone Bordet
sbordet@intalio.com

G1 Mixed GC

- **“Mixed” GC – piggybacked on Young GCs**
  - By default G1 performs 8 mixed GC
  - `-XX:G1MixedGCCountTarget=<>`
- **The collection set includes**
  - Part (1/8) of the remaining Old regions to collect
  - Eden regions
  - Survivor regions
- **Algorithm is identical to Young GC**
  - Stop-The-World, Parallel, Copying

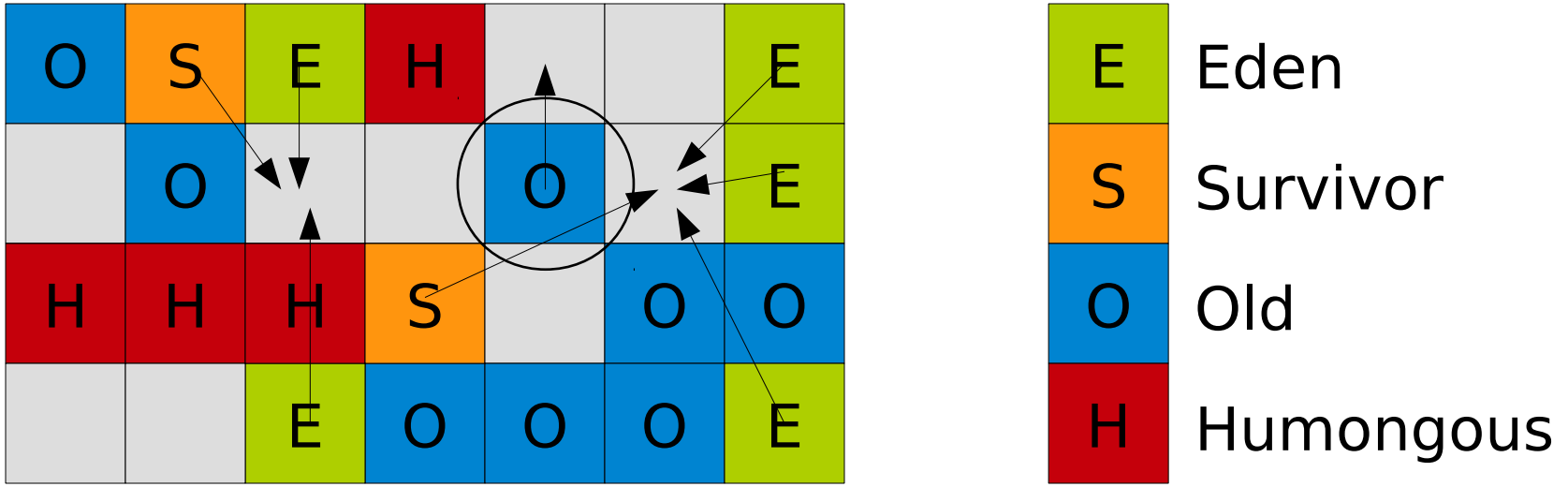
Simone Bordet
sbordet@intalio.com

- **Old regions with most garbage are chosen first**
  - `-XX:G1MixedGCLiveThresholdPercent=<>`
  - Defaults to 85%
- **G1 wastes some heap space (waste threshold)**
  - `-XX:G1HeapWastePercent=<>`
  - Defaults to 5%
- **Mixed GCs are stopped**
  - When old region garbage <= waste threshold
  - Therefore, mixed GC count may be less than 8

Simone Bordet
sbordet@intalio.com

G1 Mixed GC

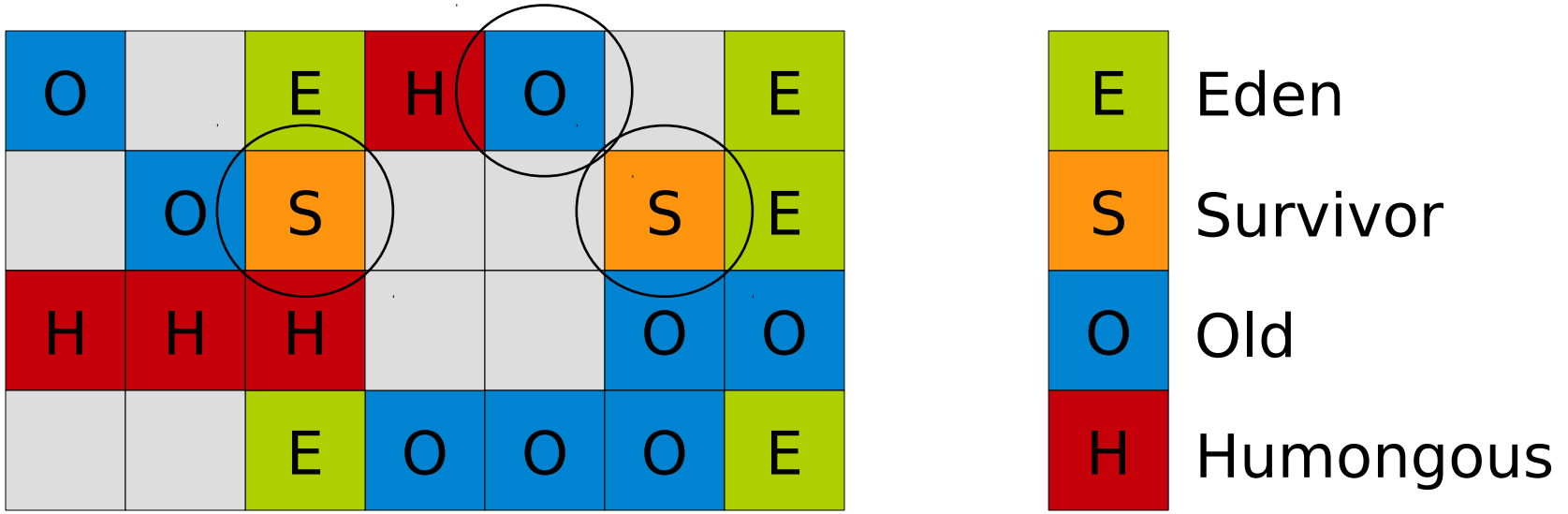
G1 Mixed GC



Simone Bordet
sbordet@intalio.com

G1 Mixed GC

G1 Mixed GC



Simone Bordet
sbordet@intalio.com

G1
General Advices

Simone Bordet
sbordet@intalio.com

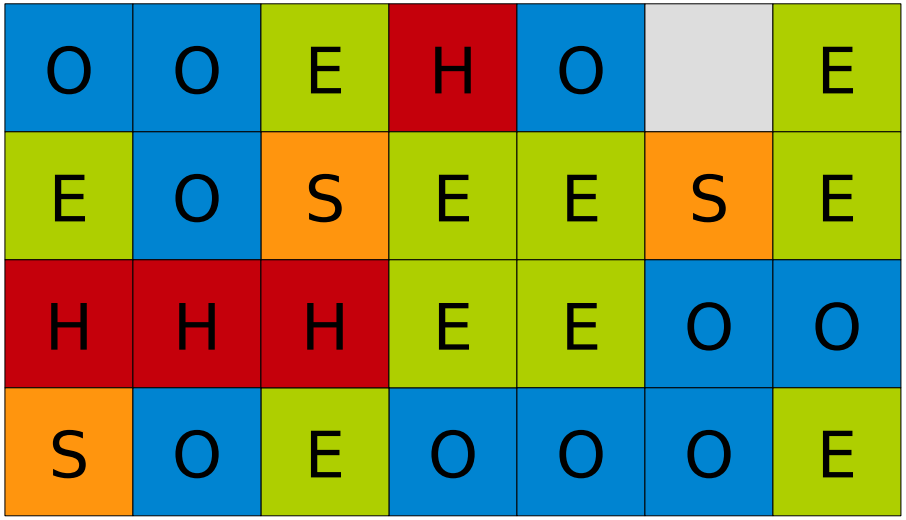
G1 General Advices

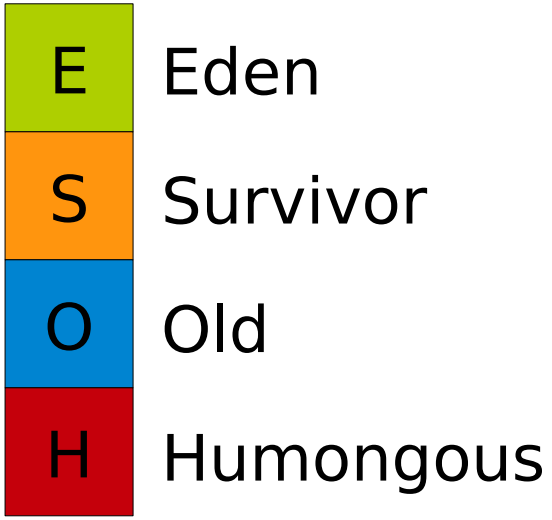
- **Avoid at all costs Full GCs**
  - The Full GC is single threaded and REALLY slow
  - Also because G1 likes BIG heaps !
- **Grep the GC logs for “Full GC”**
  - Use `-XX:+PrintAdaptiveSizePolicy` to know what caused it

Simone Bordet
sbordet@intalio.com

G1 Mixed GC

- **Avoid “to-space exhausted”**
  - Not enough space to move objects to
  - Increase max heap size
  - G1 works better with more room to maneuver





Simone Bordet
sbordet@intalio.com

G1 General Advices

- **Avoid too many “humongous” allocations**
  - `-XX:+PrintAdaptiveSizePolicy` prints the GC reason
  - Increase max heap size
  - Increase region size: `-XX:G1HeapRegionSize=<>`

- **Example**
  - Max heap size 32 GiB → region size = 16 MiB
  - Humongous limit → 8 MiB
  - Allocations of 12 MiB arrays
  - Set region size to 32 MiB
  - Humongous limit is now 16 MiB
    - 12 MiB arrays are not humongous anymore

Simone Bordet
sbordet@intalio.com

G1 General Advices

- **Avoid lengthy reference processing**
  - Always enable `-XX:+ParallelRefProcEnabled`
  - More details with `-XX:+PrintReferenceGC`
- **Find the cause for WeakReferences**
  - ThreadLocals
  - RMI
  - Third party libraries

Simone Bordet
sbordet@intalio.com

Real World Example

Simone Bordet
sbordet@intalio.com

Real World Example

- Online Chess Game Application
- 20k requests/s – Jetty server
- 1 server, 64 GiB RAM, 2x24 cores
- Allocation rate: 0.5-1.2 GiB/s
- CMS to G1 Migration

Simone Bordet
sbordet@intalio.com

Real World Example

- **Issue #1: MetaSpace**

- `[Full GC (Metadata GC Threshold) [Times: user=19.58 sys=0.00, real=13.72 secs]`

13.72 secs Full GC !

- **Easy fix:** `-XX:MetaspaceSize=<>`

- **Cannot guess how much MetaSpace you need ?**
  - Start big (e.g. 250 MiB) and monitor it

Simone Bordet
sbordet@intalio.com

Real World Example

- **Issue #2: Max target pause**
- **We set `-XX:MaxGCPauseMillis=250`**
- **Percentiles after GC log processing (24h run):**
  - 50.00% → 250 ms
  - 90.00% → 300 ms
  - 95.00% → 360 ms
  - 99.00% → 500 ms
  - 99.90% → 623 ms
  - 99.99% → 748 ms
  - 100.0% → 760 ms

Simone Bordet
sbordet@intalio.com

- **Issue #3: Mixed GCs**

- **G1 tries to respect max pause target**
- **Must account for old regions, not only young**
  - Currently G1 shrinks the young generation

- `[GC pause (G1 Evacuation Pause) (young)  [Eden: 12.4G(12.4G)->0.0B(608.0M) ...`
- `[GC pause (G1 Evacuation Pause) (mixed)`

Eden: 12.4 GiB → 0.6 GiB

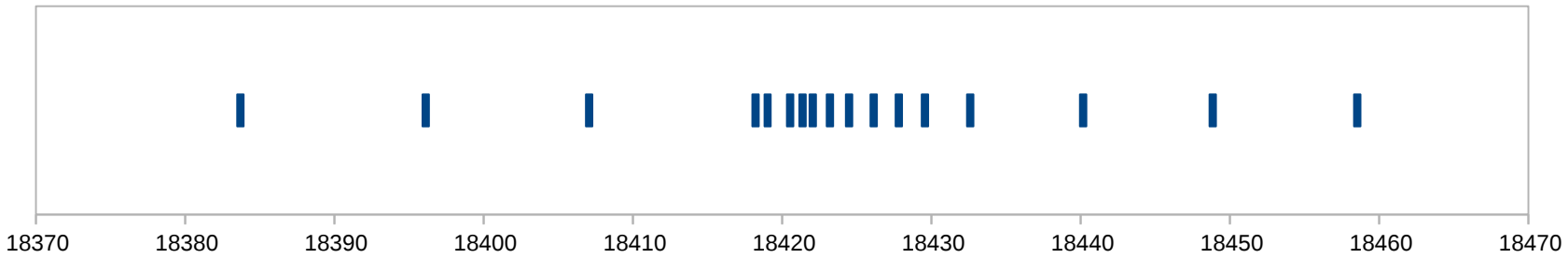
Simone Bordet
sbordet@intalio.com

- **Issue #3: Mixed GC**
- **Young Generation shrunk by a factor 20x**
  - But the allocation rate does not change !
- **Young GCs become more frequent**
  - Application throughput suffers
- **Minimum Mutator Utilization (MMU) drops**

Simone Bordet
sbordet@intalio.com

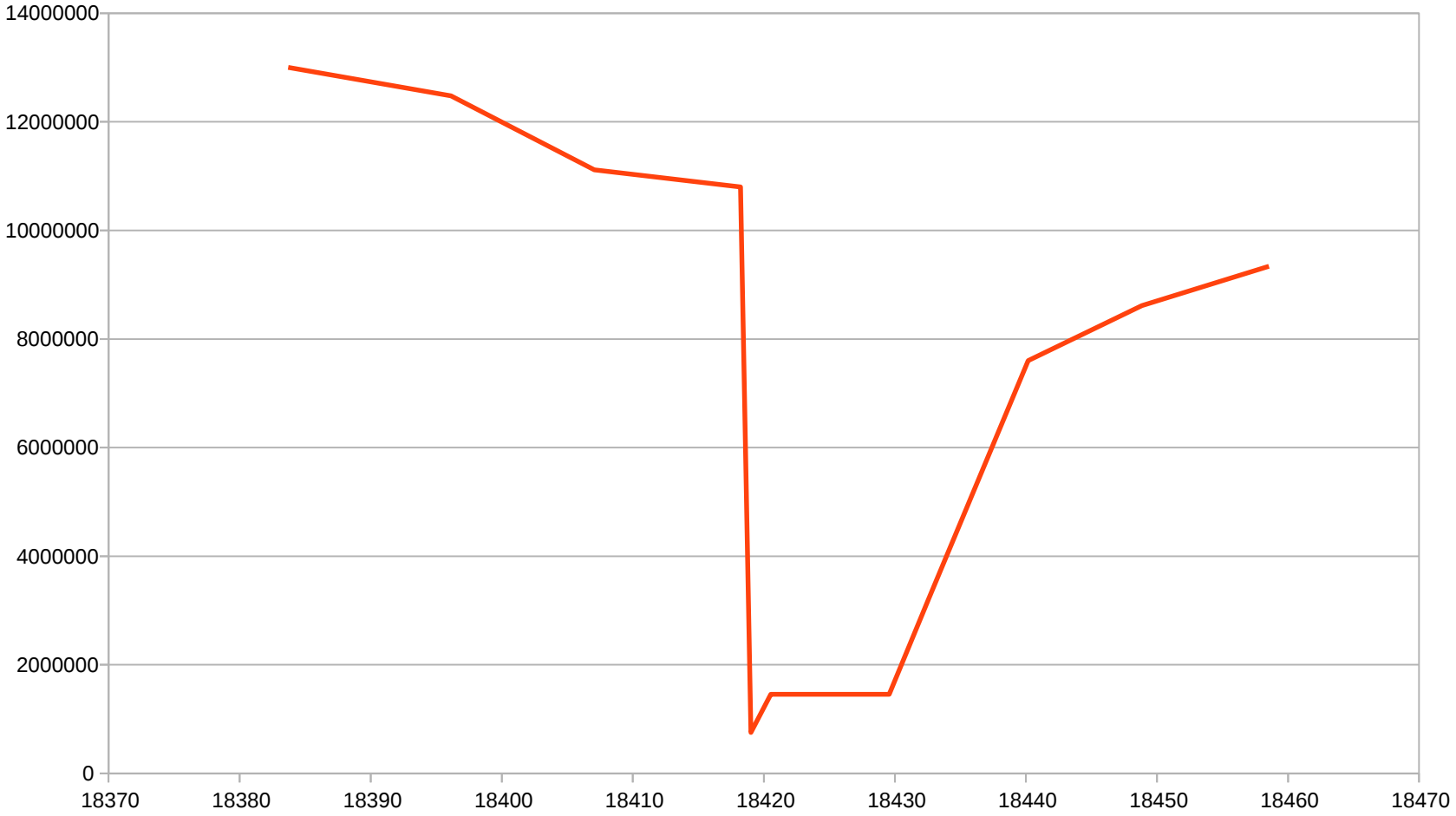
Young/Mixed GCs: Events

- X axis: event time



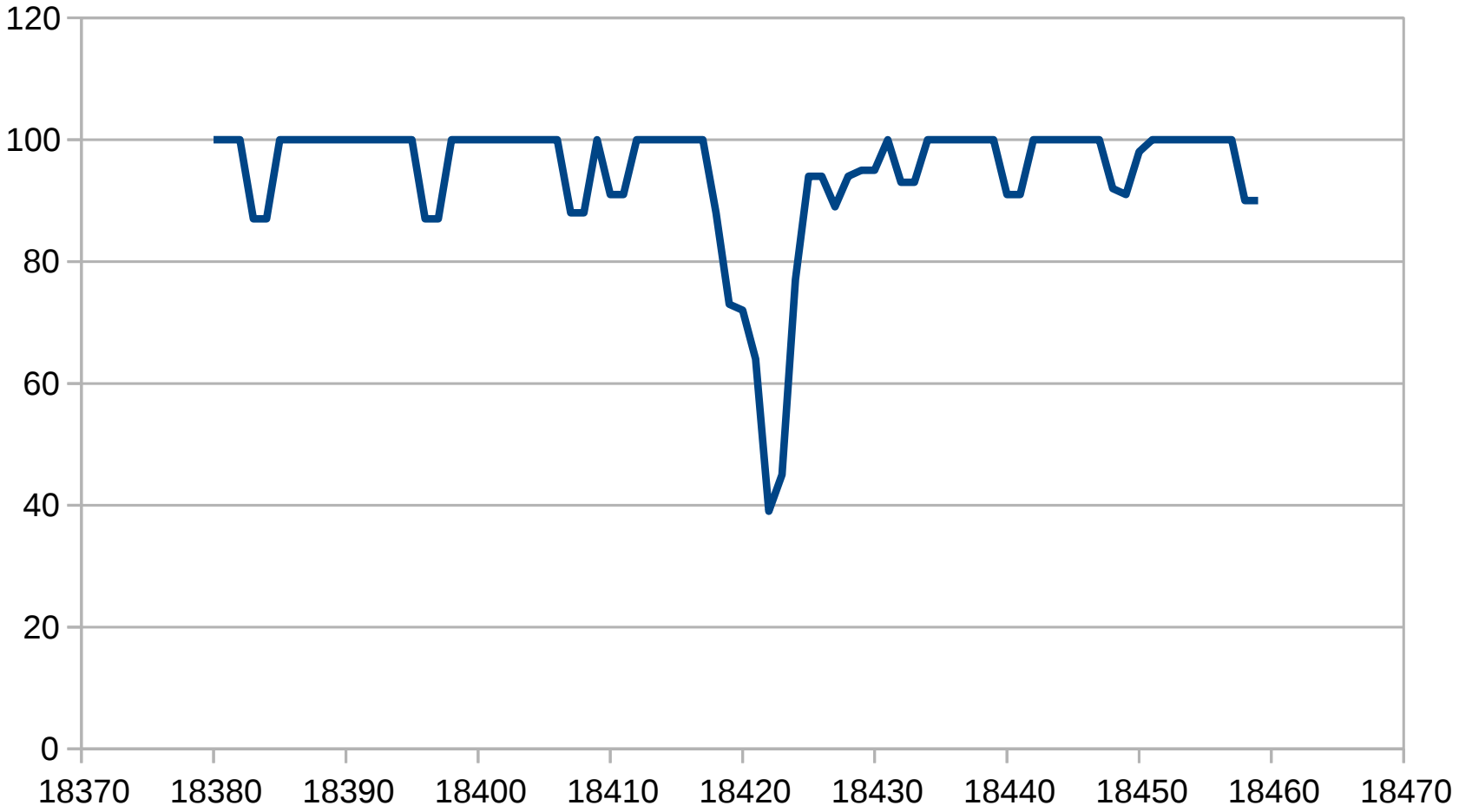
Simone Bordet
sbordet@intalio.com

Young/Mixed GCs: Eden Size



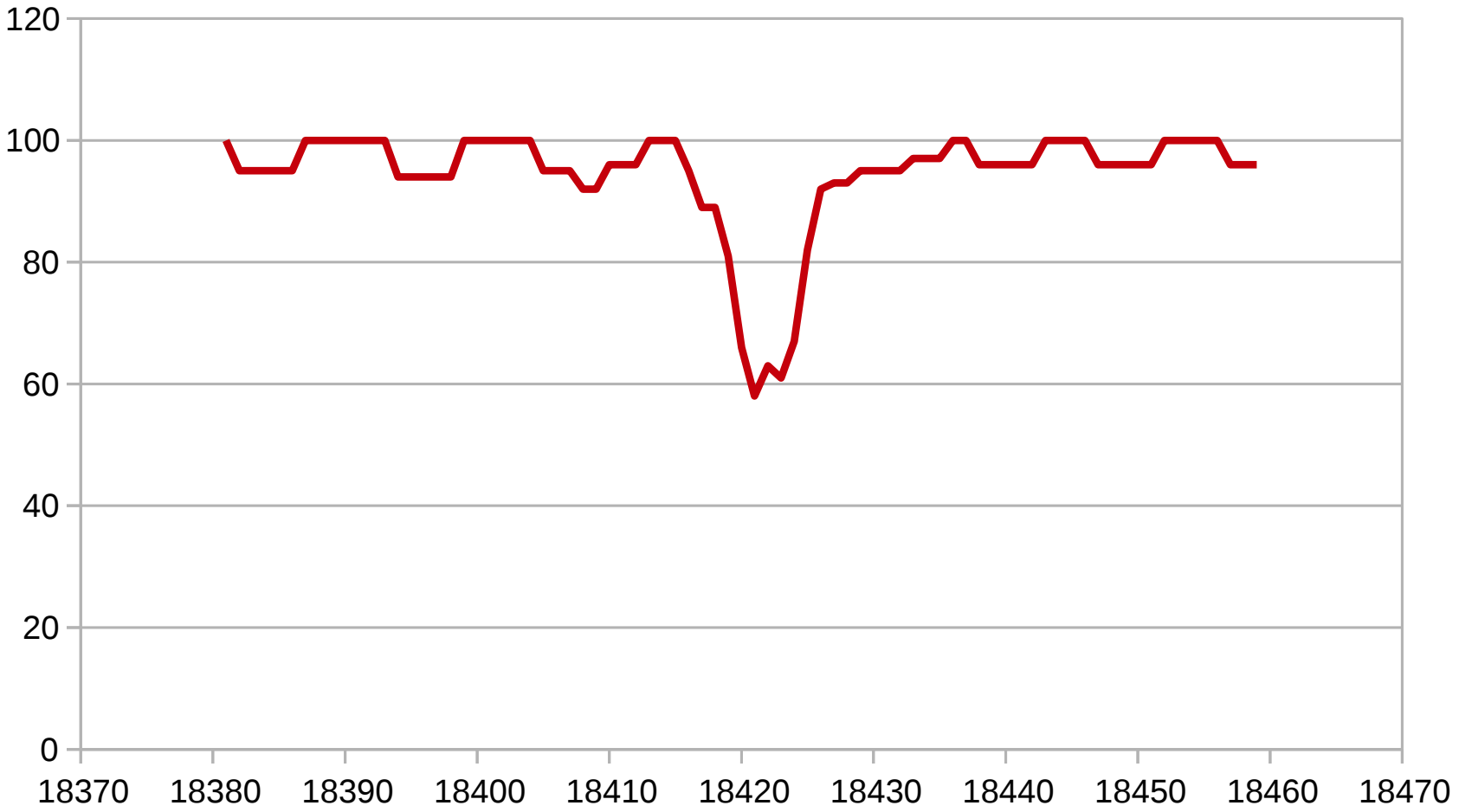
Simone Bordet
sbordet@intalio.com

Young/Mixed GCs: MMU (2 s window)



Simone Bordet
sbordet@intalio.com

Young/Mixed GCs: MMU (5 s window)



Simone Bordet
sbordet@intalio.com

Conclusions

Simone Bordet
sbordet@intalio.com

Conclusions

- **G1 is the future**
- **Good chances that “it just works”**
- **Easier to tune than CMS**
  - Yet, you must know exactly how it works to tune it better
- **Not yet that respectful of GC pause target**
  - At least in our case, YMMV

Simone Bordet
sbordet@intalio.com

Conclusions

- **Still based on Stop-The-World pauses**
  - For extremely low latencies you need other solutions
- **Always use the most recent JDK**
  - You'll benefit from continuous improvements to G1

Simone Bordet
sbordet@intalio.com

References

Simone Bordet
sbordet@intalio.com

References

- **Search SlideShare for “G1 GC”**
  - Beckwith, Hunt, and others
- **Numerous articles on Oracle's site**
  - Yu Zhang
  - Poonam Bajaj
  - Monica Beckwith
- **OpenJDK HotSpot GC Mailing List**
  - hotspot-gc-use@openjdk.java.net

Simone Bordet
sbordet@intalio.com

Questions
&
Answers

Simone Bordet
sbordet@intalio.com