Garbage Collection:

@EvaAndreasson, @Cloudera

AGENDA

- Garbage Collection (101)
- The Good
- The Bad
- The Ugly
- The Challenge!

GARBAGE COLLECTION

"When you have to shoot, shoot - don't talk!"

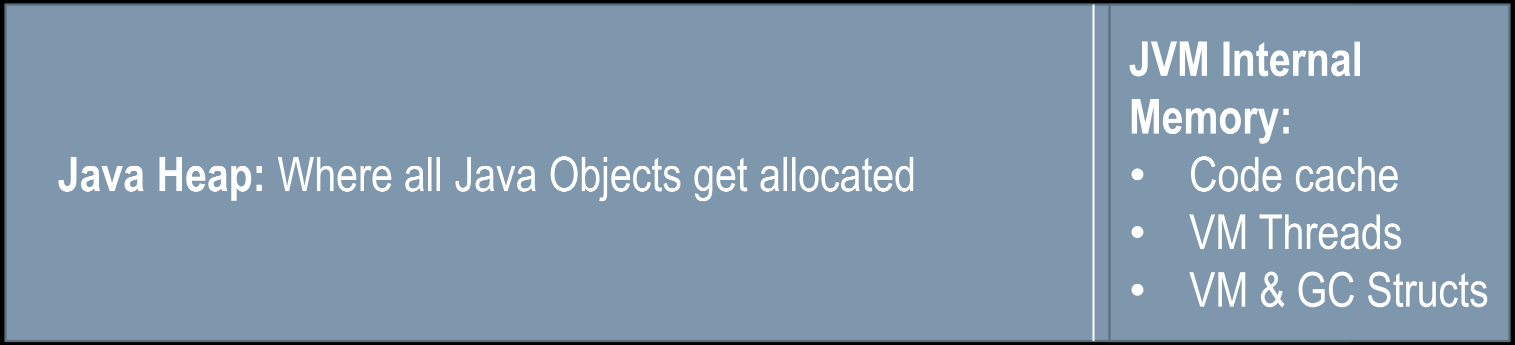
WHAT IS GARBAGE COLLECTION (GC)?

- What is a Java Virtual Machine (JVM)?
  - Runtime code compilation
  - Dynamic memory management

- What is dynamic memory management?
  - Does not require explicit memory allocation (when programming)
  - Frees up memory no longer referenced

JAVA HEAP - SPOTLIGHT

- The mythical (?) Java heap
  - -Xmx, -Xms
  - Top: RES / RSS --- total memory footprint
- Full = a thread fails to allocate a new object



Java Heap: Where all Java Objects get allocated

JVM Internal Memory:
- Code cache
- VM Threads
- VM & GC Structs

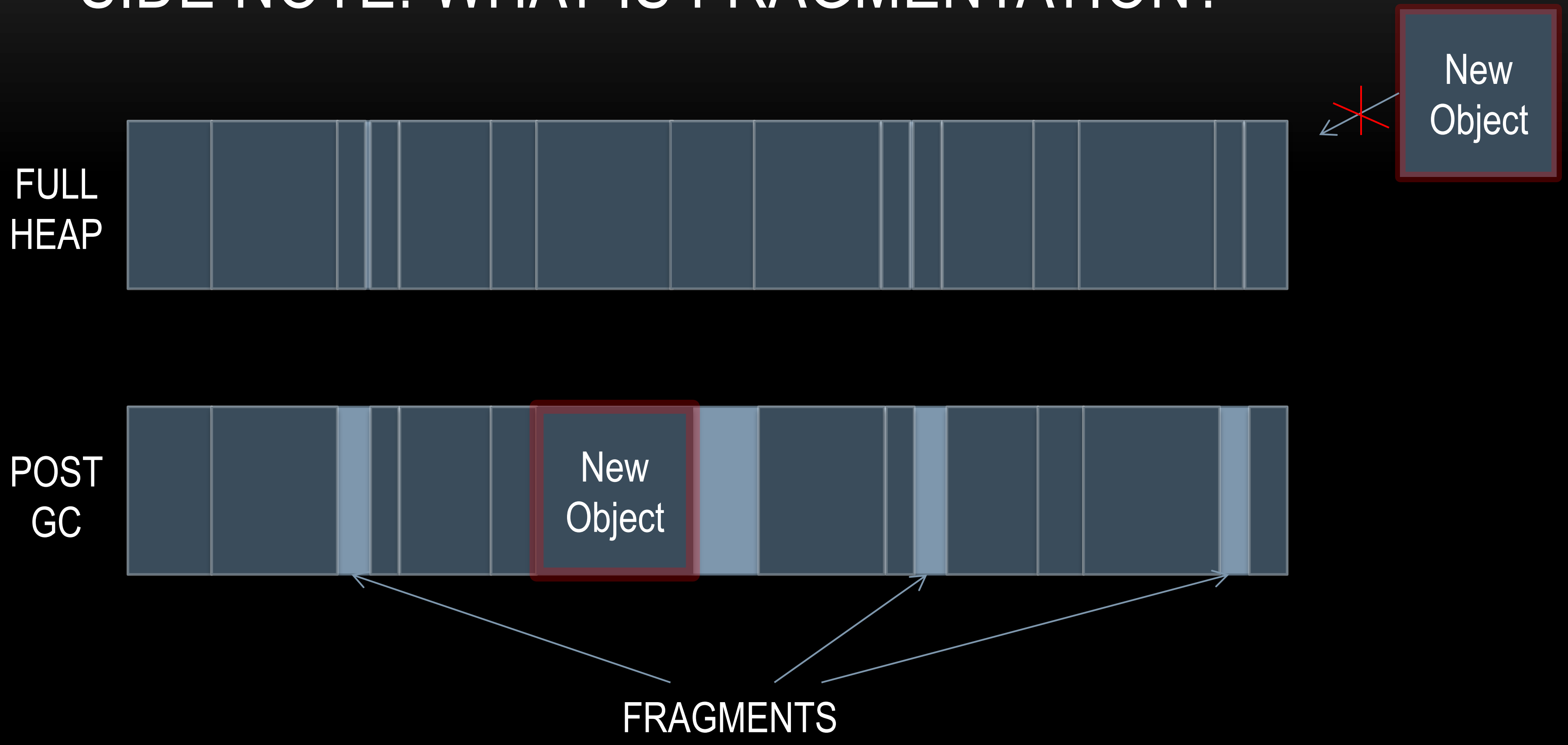
REFERENCE COUNTING VS. TRACING

- Reference Counting
  - Plus a counter for each reference to an object
  - Subtract for each removed reference to an object
  - When 0, reclaim the heap space occupied by that object
- Simple to reclaim
- Hard to maintain counters
- Difficult (costly) to handle cyclic structures

REFERENCE COUNTING VS. TRACING

- Tracing
  - Identify roots (thread stacks, …, etc)
  - Trace all references from those objects, recursively
  - Anything not found is garbage
- Simple to maintain, handles cyclic structures
- Needs to trace all live objects before reclaiming memory

SIDE NOTE: WHAT IS FRAGMENTATION?



COPYING VS. MARK’N’SWEEP

- Copying
  - Split the Java Heap into sections
  - Allocate only in the one section, until full
  - Stop the world
  - Trace all reachable objects in the section and move (copy) them to another section
  - Reclaim the original section as “free”
- Prevents fragmentation
- Wasteful in space
- Stops the world

COPYING VS. MARK'N'SWEEP

- Mark'n'sweep
  - Allocate objects in the entire heap space, until full
  - Trace and *mark* live objects (no moving)
  - When all live objects are marked, *sweep* all non-marked areas (build free lists)
  - Allocation can now happen at the address spaces of the free lists
- Allows entire heap for allocation
- Suffers from fragmentation
  - Over time free list chunks too small to fit new objects

PARALLEL VS. CONCURRENT

- Parallel
  - Stop the world
  - Allow all available threads to do GC work
  - Allow allocation once the entire GC cycle is complete
- Concurrent
  - Allow some of the available threads to do as much GC work in the background as possible, without impacting running applications too much
    - Iterative marking, track areas where running applications have made modifications and re-mark
  - Needs to start in time…

GENERATIONAL

- Generational (-Xns)
  - Most objects die young
  - Define a space (could be distributed) on the heap for allocation
  - The rest of the heap is considered “old space” or “old generation”
  - As objects “survive” garbage collection in the young space (a.k.a. nursery), promote them to the old space
- Reduces the speed of fragmentation of the heap
- Can use different algorithm than old space: copying, parallel and copying…

COMPACTING

- Compaction
  - The operation of moving objects on the heap together
  - Opens up larger consecutive spaces of free memory
  - Mitigates fragmentation
- Compaction area size
- Incremental
- Intelligent

- Parallel mark'n'sweep and copying implementations usually do this during their normal stop the world phase
- Concurrent mark'n'sweep needs to handle this somehow, eventually…

GOOD

GARBAGE IS GOOD!

- Wait…what now?!?

THE GOOD

- Garbage means you are using Java the way it was intended – truly object oriented!!
- Without GC, the world would have looked differently
  - Java helped software (and hardware) innovation
  - Programming became “mainstream” (no offence…)
  - Coding could be done faster
  - More jobs were created
  - More products and businesses popped up
- If tuned “right”…

BAD

THE DESERT OF TUNING

- Endless tuning and re-tuning
  - Rant-warning!

- Ok for some application profiles
  - Time window applications
  - Client applications
  - Specially architected applications (new-objects only)
  - Applications not sensitive to latency

A DESERT SURVIVAL KIT

- Chose the right GC algorithm for your application
- Understand your application allocation rate (in production) and allocate enough heap
- Measure the right thing!!
    - Test != Production
    - Not average or std dev – latency is not a standard distribution!
    - Check out: Gil Tene’s jHiccup tool (and his talk) – a great new approach!

UGLY

RECOGNIZE THIS?

- Initially everything is fine, GCs are happening without much impact
- Over time application seems to freeze up on occasion, or starts responding slower and slower
- Soon, the entire application hangs, affecting other servers to start firing up
- Eventually the JVM gives up and “crashes”
- GC logs show back-to-back GCs and in the end some sort of Out Of Memory, Allocation, or Promotion Error

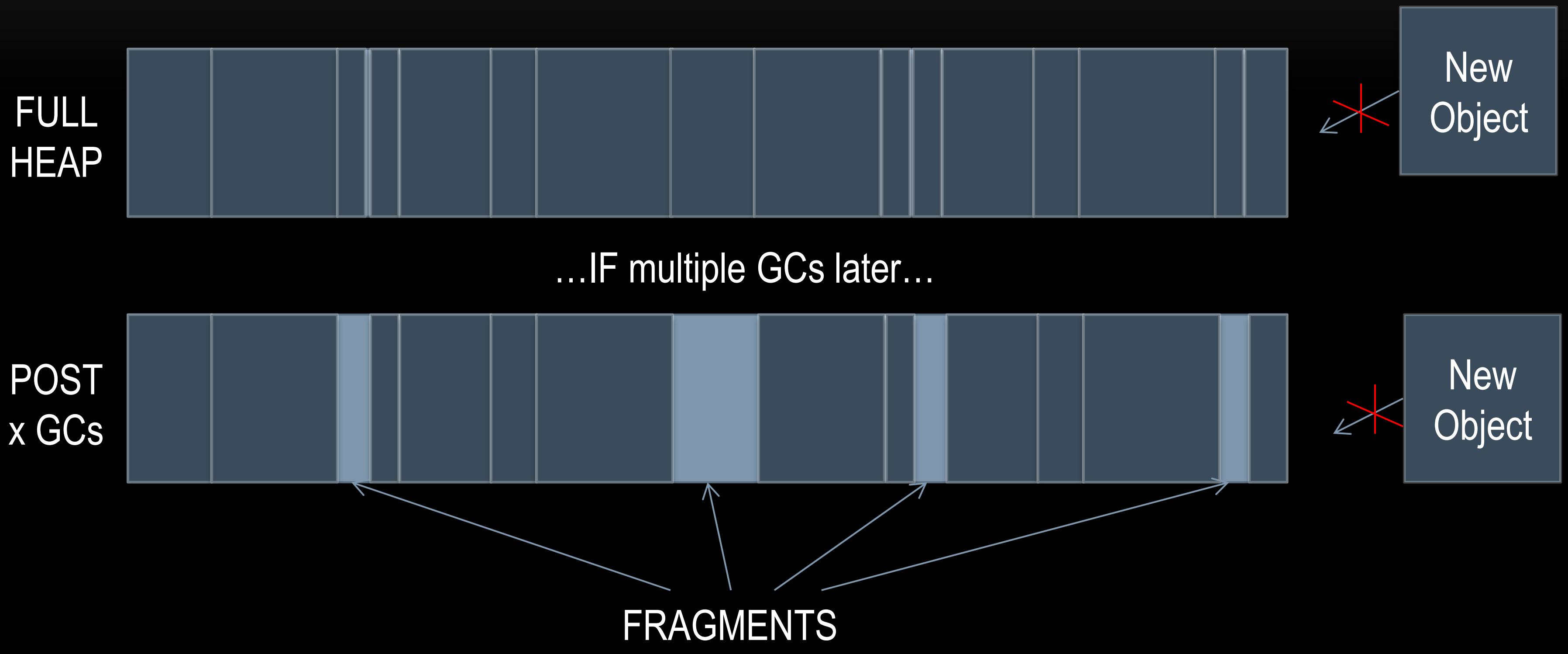
WHAT REALLY HAPPENS

- When allocation fails, GC is triggered
- GC is doing its job, but no memory opens up (everything is live)
- Back-to-back GCs, still no new memory => OOME..
- OOME indicates not enough heap for your allocation rate

WHY NOT CONFIGURE A LARGER HEAP?

“GC PAUSES”

REMEMBER FRAGMENTATION?



REMEMBER COMPACTION?

- Most GC implementations do not handle compaction well
- Moving objects is costly – stop the world is easy
- Generational added
- Tuning options and heuristics added

- Only one JVM that I know of that does compaction concurrently today (Zing)

PREPARE FOR THE REAL VILLAIN…

"There are two kinds of people in the world my friend, those with a rope around their neck and the people that have the job of doing the cutting!"

STOP THE WORLD OPERATIONS!!!

STOP THE WORLD OPERATIONS

- Prevents efficient memory utilization
- Creates complex JVM deployments
- Sends you out in the tuning desert…

SUMMARY

- Garbage is GOOD
- The need to tune is BAD
- Stop the world operations are UGLY

I CHALLENGE THEE

"You see in this world, there's two kinds of people my friend…
…those with loaded guns and those who dig…you dig!"

JOIN THE FUTURE OF JAVA!

- Open JDK is a great opportunity for innovation - join the community!
- Have all GC algorithms been invented yet?
- How do we enable a better world of self-tuning, adaptive JVMs?
  - Relieve the admin of the pain of the tuning!
- How about fixing the core problem?
  - Implement concurrent compaction
- Be creative around allocation / dynamic allocation rates!

Please evaluate my talk via the mobile app!

QCon
INTERNATIONAL
OFTWARE DEVELOPMENT
CONFERENCE

Q&A

@EvaAndreasson