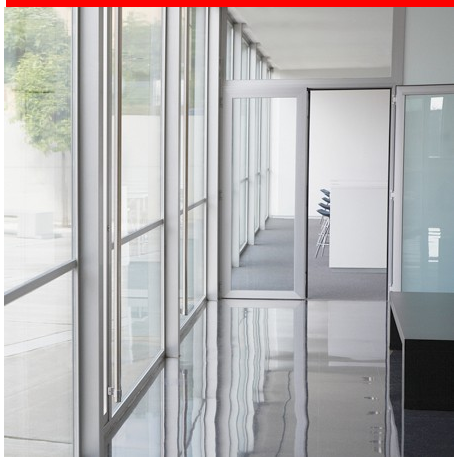

ORACLE®

ORACLE®

How to Tune and Write Low-Latency Applications on the Java Virtual Machine

Simon Ritter - Java Technology Evangelist

Program

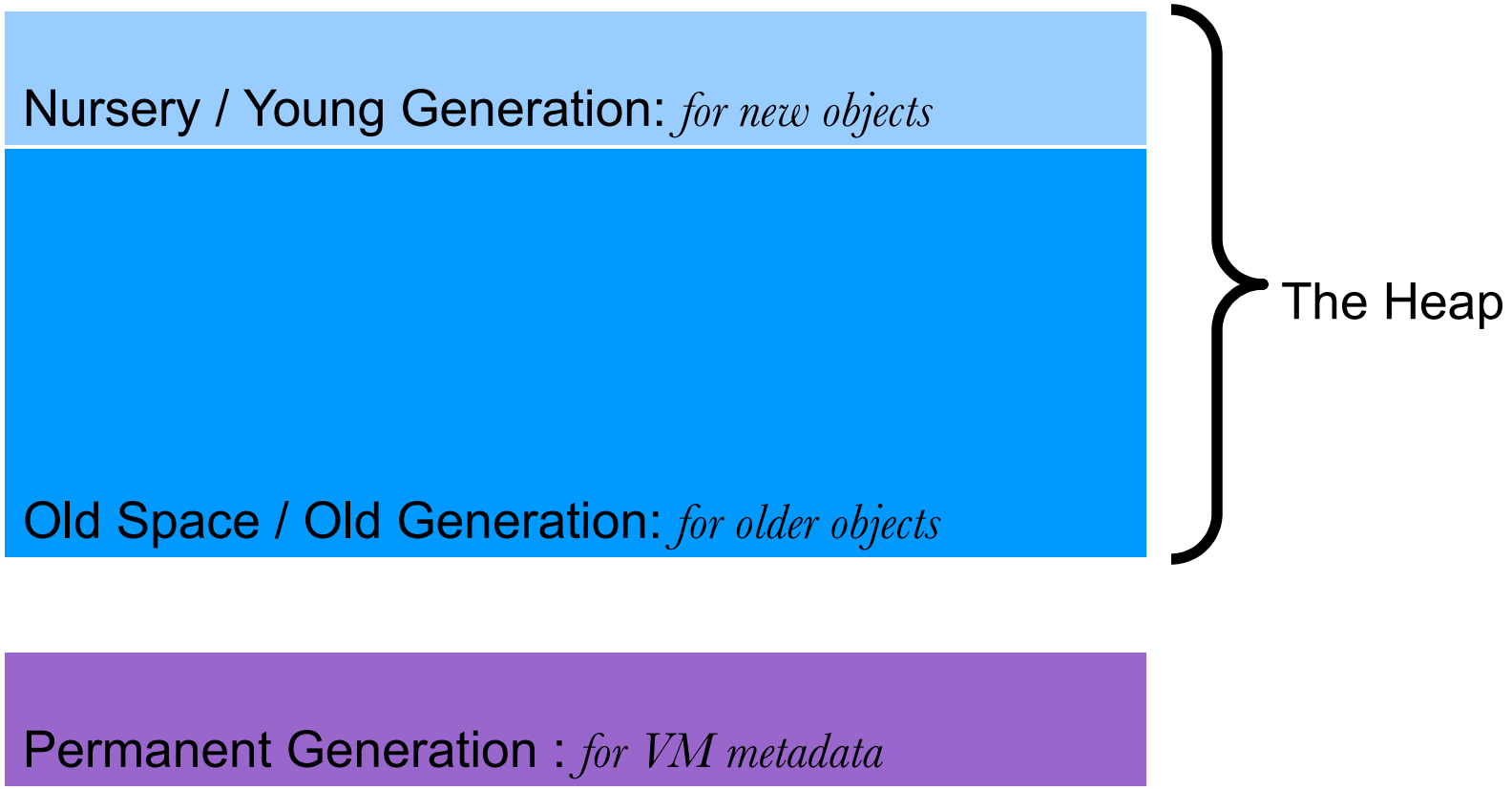


- Vocabulary
- What Is Low-Latency Java
- JVM Features
- What To Think About
- Application Analysis

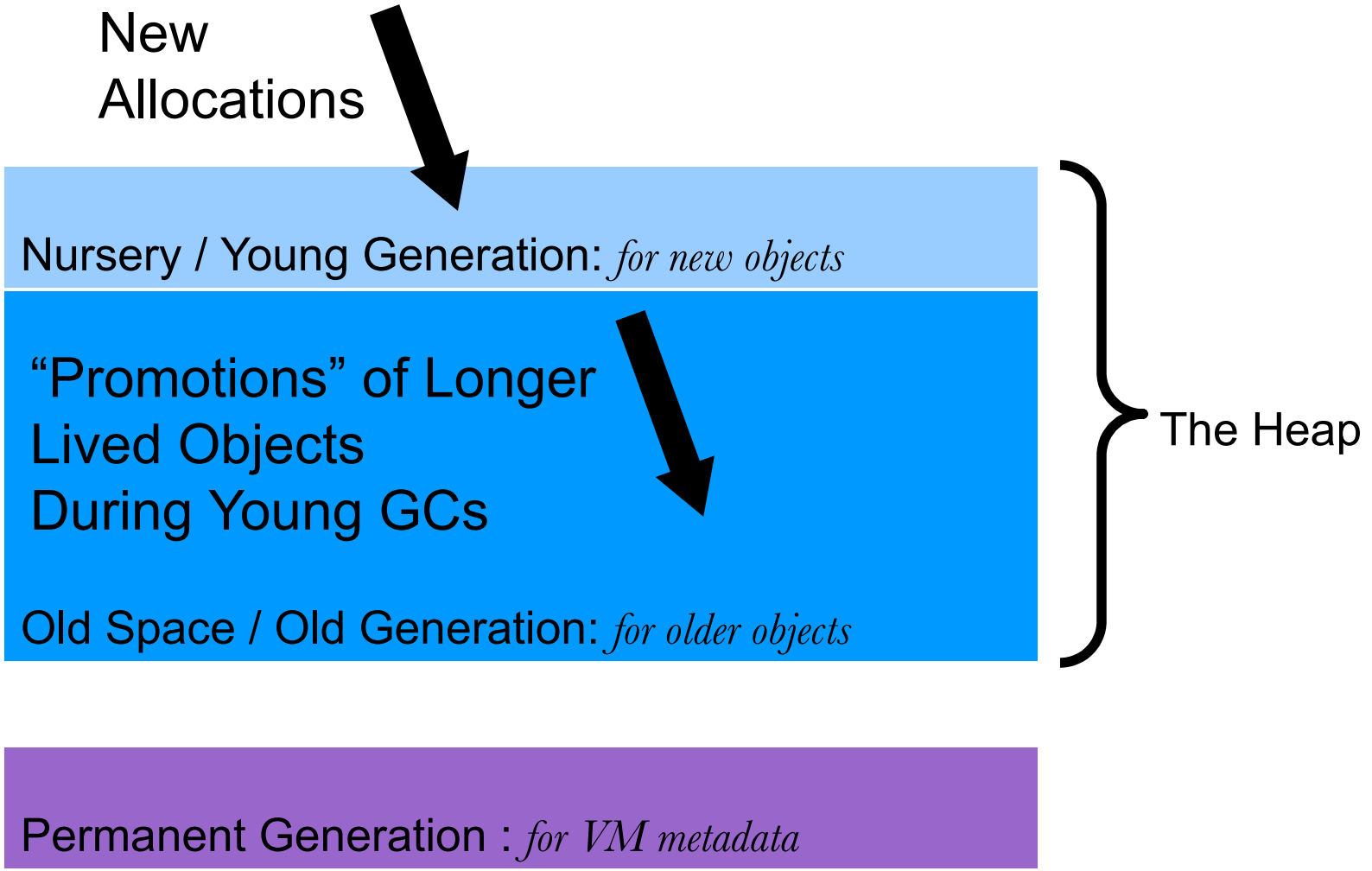
Vocabulary

- Nursery == Young Space == Eden
- Nursery Collection == Young Collection == Minor GC
- Promotion
  - Move objects to Old Space
-
- Old Space == Old Generation
- Old Collection == Full Collection
- Compaction
  - Move objects together to remove fragmentation

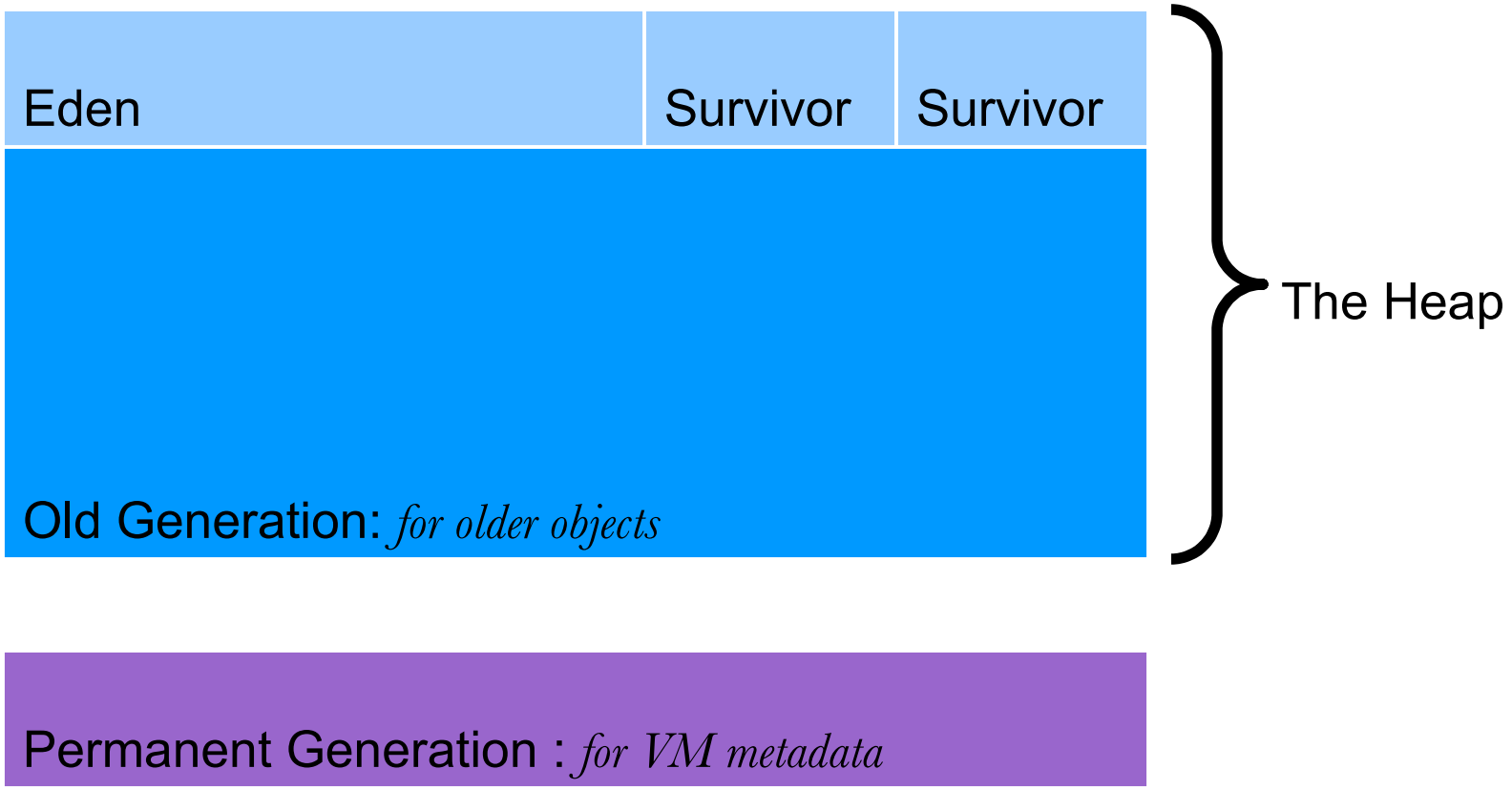
JVM Heap Layout



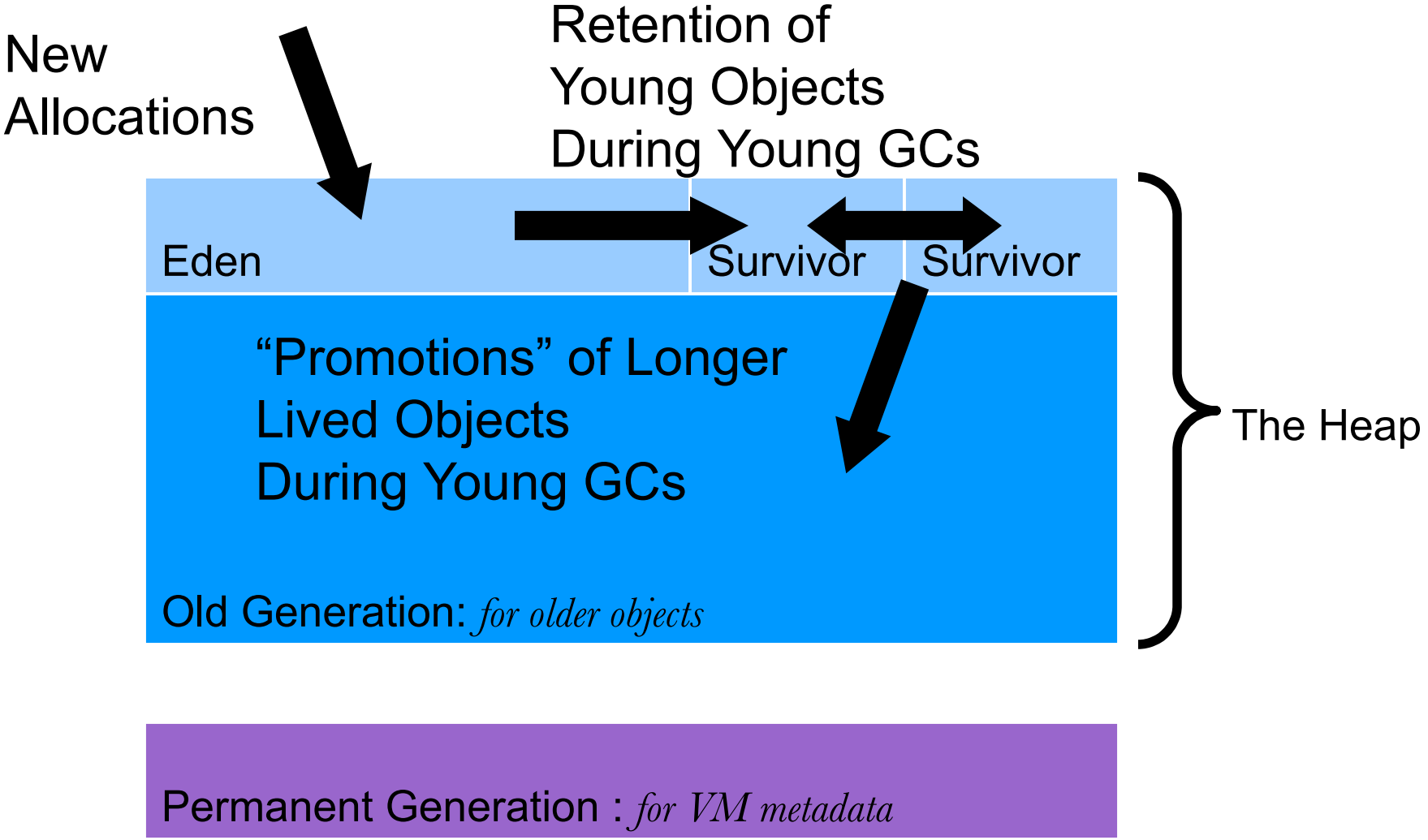
JVM Heap Layout



HotSpot JVM Heap – In More Detail…



HotSpot JVM Heap – In More Detail…



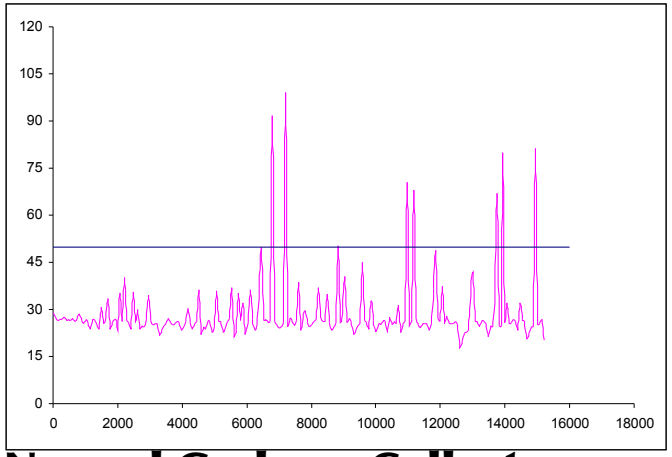
Low-Latency Java

What is Low-Latency Java

- Soft Real-Time
  - High throughput with low latency
  - No hard response time guarantees
  - No catastrophe happens at response time failures
    - But several nines response time guarantees
- Normal Java Code
  - No special APIs
  - Code tuning with focus on latency

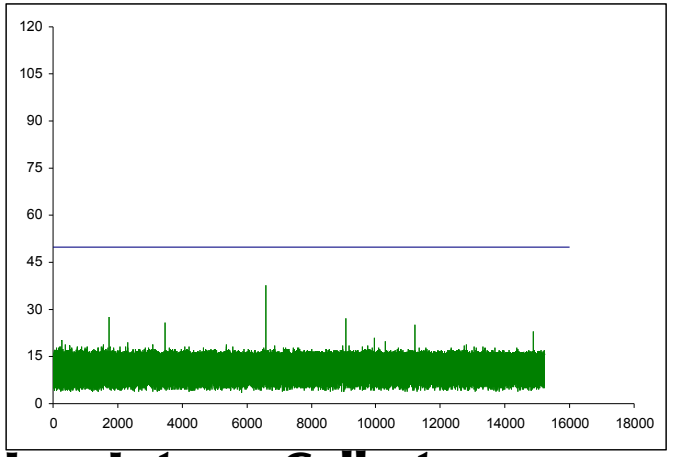
What is Low-Latency Java

- Run standard Java code with great response times



Normal Garbage Collector

GC spikes and occasional SLA breach occurs.



Low-Latency Collector

Deterministic GC pauses, allowing guarantees of SLAs.

What is Low-Latency Java

- Response Time Requirements
  - Typical latency requirement around 5 – 50 ms
  - Max latency = transaction time + max pause time
- Typical Applications
  - Financial
    - Automatic trading
    - Matching server
  - Telecom
    - SIP applications
  - Event processing
    - RFID scanning

JVM Features

What Do the JVMs Offer

- GC Implementations
  - Generational Concurrent Collectors
    - CMS – HotSpot
    - Gencon – JRockit
  - Regional Heap Collector
    - G1 – HotSpot (In development, EA available)
  - Optimized Concurrent Collector
    - Deterministic GC – JRockit Real-Time

Generational Concurrent Collectors

- Use Case
  - Normal heap sizes
  - Reasonable amount of live data
- Nursery Collector
  - Stop the world
  - Parallel
- Old Collection
  - Mostly concurrent Mark and Sweep
    - Some short parallel pauses
  - Partial compaction

Generational Concurrent Collectors

- Tuning
  - Nursery sizing
    - Large enough to reduce frequency
    - Small enough to get adequate pauses
  - Compaction tuning
    - How large part of the heap to compact each GC
      - Large enough to avoid fragmentation
      - Small enough to get adequate pauses

Generational Concurrent Collectors

- Tuning Cont.
  - Concurrent GC threshold
    - Start concurrent GC early enough
    - Complete GC before memory become scarce
    - Tuned automatically

G1 – Regional Heap Collector

- Use Case
  - Large heaps
  - Large amount of live data
- Regional Nursery Collector
  - Stop the world
  - Parallel
- Regional Old Collector
  - Evacuation of live data
  - Remembered sets
    - No mark phase

G1 – Partial Heap Collector

- Tuning
  - Pause target
    - Lower pause target requires a more GC aware code
    - -XX:MaxGCPauseMillis=X
  - Nursery sizing
    - Implicit by pause target tuning
  - Evacuation tuning
    - Implicit by pause target tuning

Deterministic GC – Optimized Concurrent Collector

- Use Case
  - Normal heap sizes
  - Live data around 1/2 of the heap
- Old Collection
  - Parallel mostly concurrent Mark and Sweep
  - Heavily optimized pauses
  - Abortable compaction

Deterministic GC – Optimized Concurrent Collector

- Tuning
  - Pause target
    - Lower pause target requires a more GC aware code
    - -XXpausetarget=Xms
  - Heap sizing
    - Large enough to hold live data and free heap
    - Small enough to get adequate pauses
  - Compaction tuning
    - Implicit by pause target tuning
    - Ensure enough pause time to keep heap unfragmented

What To Think About

What To Think About – Throughput

- Potential Performance Impact
  - Bookkeeping of objects
    - Tracking new objects during a concurrent collection
    - Updating remembered sets
- Don’t saturate the CPU
  - The concurrent GC threads shares the CPU

What To Think About – Allocation

- Allocation Rate
  - More allocation => More GCs => More pauses
  - Need enough time to finish concurrent phases
    - Complete GC before memory becomes scarce
  - Can still handle high allocation rates
    - Hundreds of MB per second
- Large Objects/Arrays
  - Requires free consecutive memory
  - Increases the requirement to do compaction

What To Think About – Allocation

- Avoid System.gc()
  - Let the memory system handle GCs
- Semi-Long Lived Objects
  - Increases YC times due to copying during promotion
  - Increases fragmentation in Old Space
    - Mix of short and long lived data
  - Avoid storing data from each transaction
  - Make sure tenuring thresholds is configured properly

What To Think About – Data Structures

- Understand Your Data Structures
  - Resizing Data Structures
    - Size your HashMap correctly
      - Rehashing when increasing size takes time
      - Hard to detect
    - StringBuilder/Buffer expands and copying
      - We do optimizations to try to avoid this
  - Optimal Use Case
    - ArrayList add/remove at beginning of list causes copying

What To Think About – Data Structures

- GC Friendly Data Structures
  - Mark phase iterates over live data
  - Some data structures are hard to mark in parallel

What To Think About – Reference Objects

- Use Reference Objects Sensibly
  - Soft, weak and phantom references require special processing during GC
  - Finalizers should be avoided
    - Will keep objects alive for one extra GC cycle
    - Detrimental to pause time goals

What To Think About – Profiling

- Externally Measure End-to-End Response Time
  - Internal measurements affected in the same way as the application
- GC Analysis
  - Understand your object allocation and survival patterns
  - GC pause times
    - The name of the pause is often a good hint
- Lock Profiling
  - #1 issue for scaling problems and unexplained outliers

What To Think About – Finally

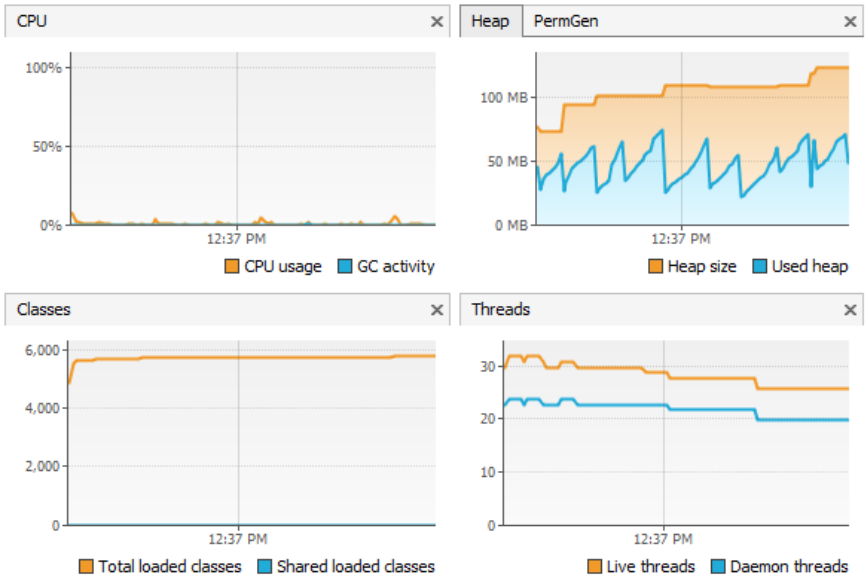
- **Don’t Overdo It**
  - The JVM can handle a lot
  - Analyze to detect your current bottleneck
    - Optimize and tune

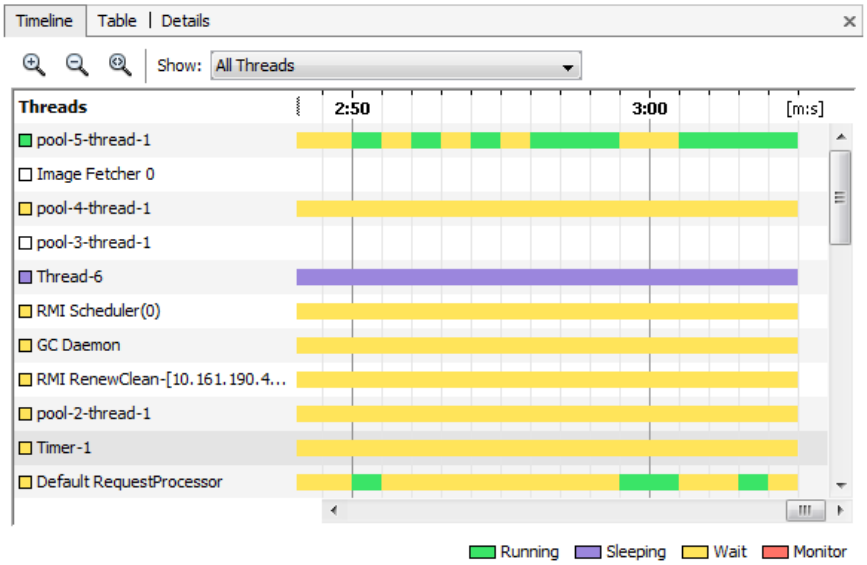
Application Analysis

Know What to Optimize

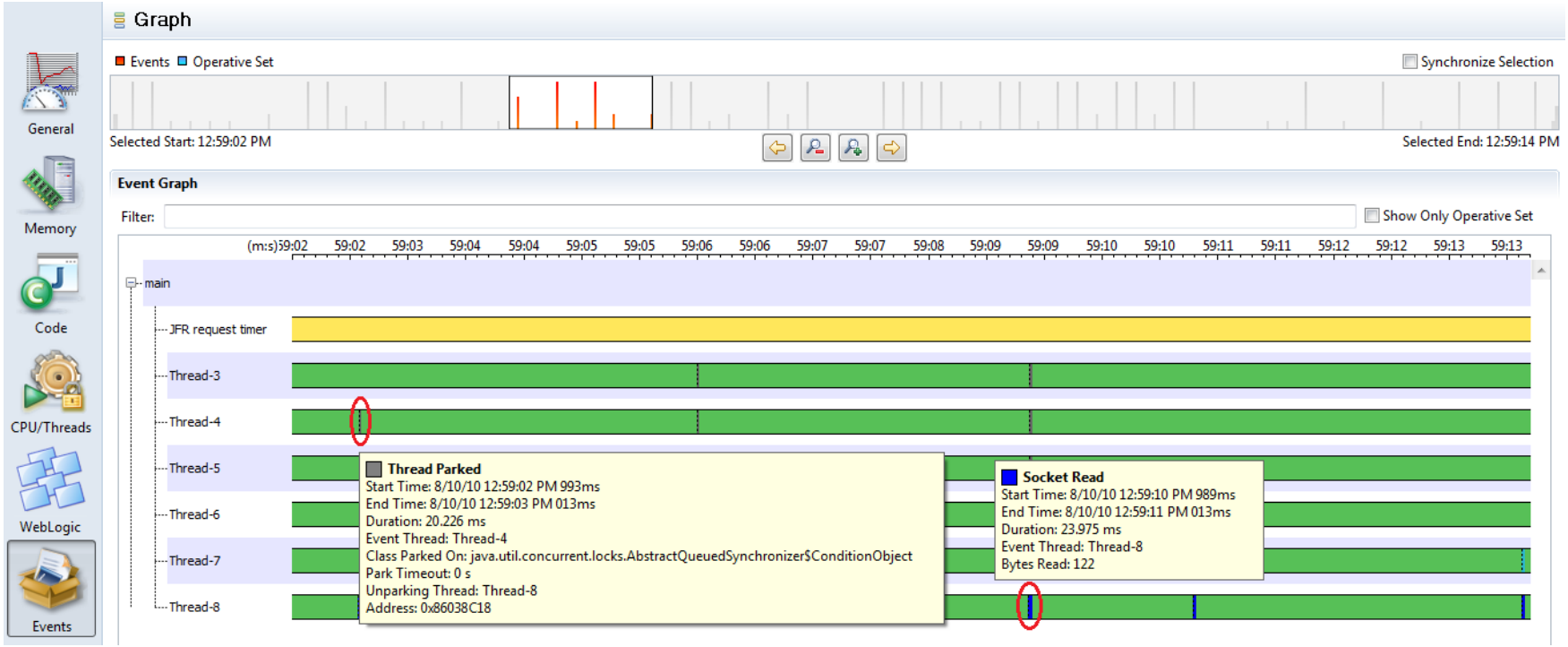
- Analyze Your Application
  - Garbage collections
    - Pauses
    - Heap usage
  - Hot Methods
  - Lock contention
  - IO events

HotSpot VisualVM





JRockit Mission Control



Other Tools

- Java Tools
  - IBM Health Center
  - JProfiler
- Low Level Tools – Hardware Profiling
  - VTune
  - CodeAnalyst
  - Oracle Solaris Studio
  - oProfile

JavaOne and Oracle Develop

Russia 2011

April 9-10, 2011

SOFTWARE. HARDWARE. COMPLETE.