Java

JavaOne

Building Real-Time Systems for the Real-World

Session TS-6989

Mike Fulton
IBM Canada Ltd.

Agenda

- What Are Real-Time Systems?
- What Business Sectors Need Real-Time?
- How are Java™ Systems Adapting to Real-Time?
- What Real-Time Tools Are Available?
- Does Any Middleware Run on Real-Time JVMs?

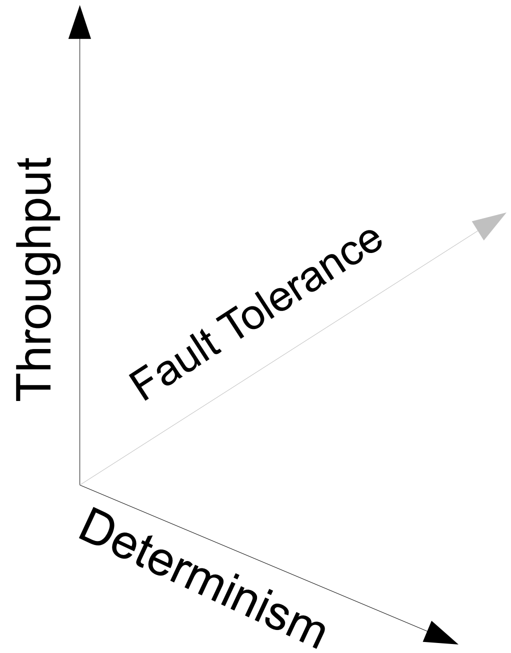
What Are Real-Time Systems?

> Broad Category Describing a Range of Systems

- Any System With Real-World Time Constraints

Examples:

- Cruise Control speed change *never* fails
- Assembly Line Advance every 20 minutes
- All Trades Must Complete in <25ms
- 99.99% of all radar scan events captured
- 95% of call packets processed in <20ms
  - 99.9% of packets processed in <40ms
    - 100% of packets processed in <50ms



What Business Sectors Need Real-Time?

- A better question would be who doesn't need it
- Improved predictability would help most systems
  - Telco: Could you repeat that? The line is crackling.
  - Financial: 'most' trades complete quickly.
  - Desktop Systems: Ever had a tool 'freeze up'?
  - Web Servers: Click 'Reload' – it's taking too long.
  - Safety Critical: Want to stop when you hit the brake?

Writing Real-Time Java Applications

- Using Standard Java Virtual Machines (JVMs)
  - May not satisfy Service Level Agreements (SLAs)
    - Garbage Collection causes application delays
    - Java Threads may not use Real-Time Scheduling
    - Compilation can cause unexpected CPU spikes
    - Class Loading causes loading from disk
    - Underlying OS may not provide consistent services
    - Underlying Hardware may have *random* interrupts

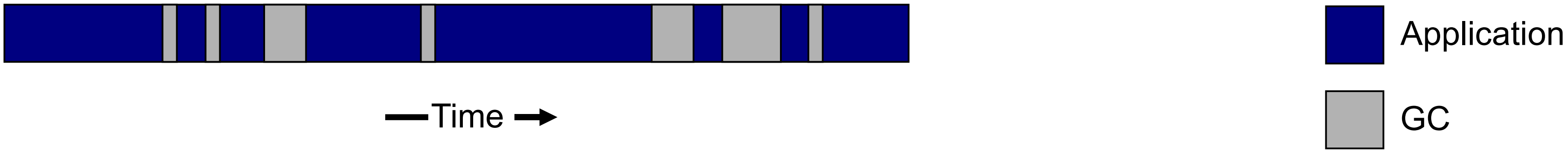
Garbage Collection (GC)
Different Policies for Different SLAs

- Several Popular GC Policies Available Today
  - High Throughput Stop-The-World Collection
  - Generational, Concurrent Collection
  - Incremental Collection
  - Work-Based Collection
  - Event-Based Collection

Garbage Collection Policies

High Throughput Stop-The-World

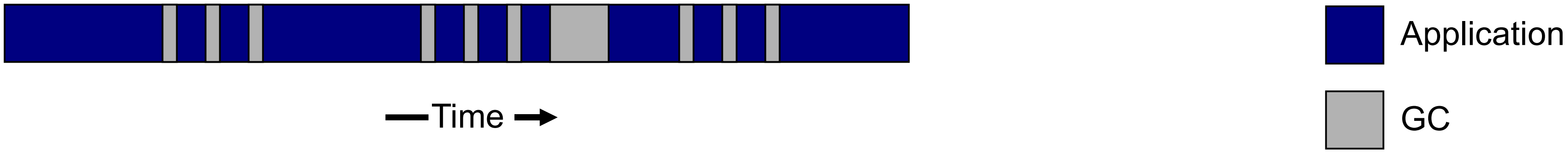
- Run Application at full speed until memory low
  - Stop all application threads
  - Clean up objects that are no longer referenced
  - Transfer control back to application
  - Garbage collection delays are variable



Garbage Collection Policies

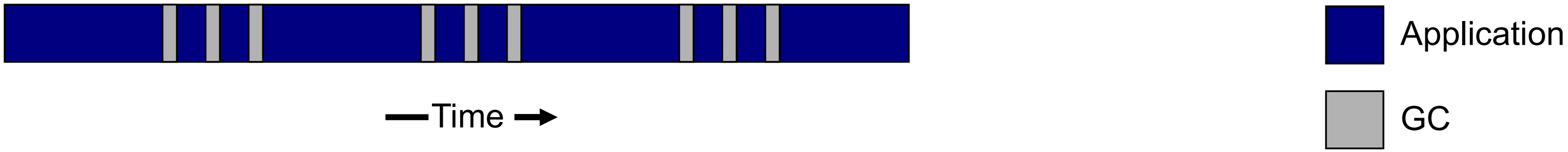
Generational, Concurrent

- Run Application, Garbage Collector Mark in Parallel
  - Perform Very Small, Fast, Nursery collect often
  - Perform Large Tenured Space collect infrequently
  - Less variable than Stop-the-World, but not consistent



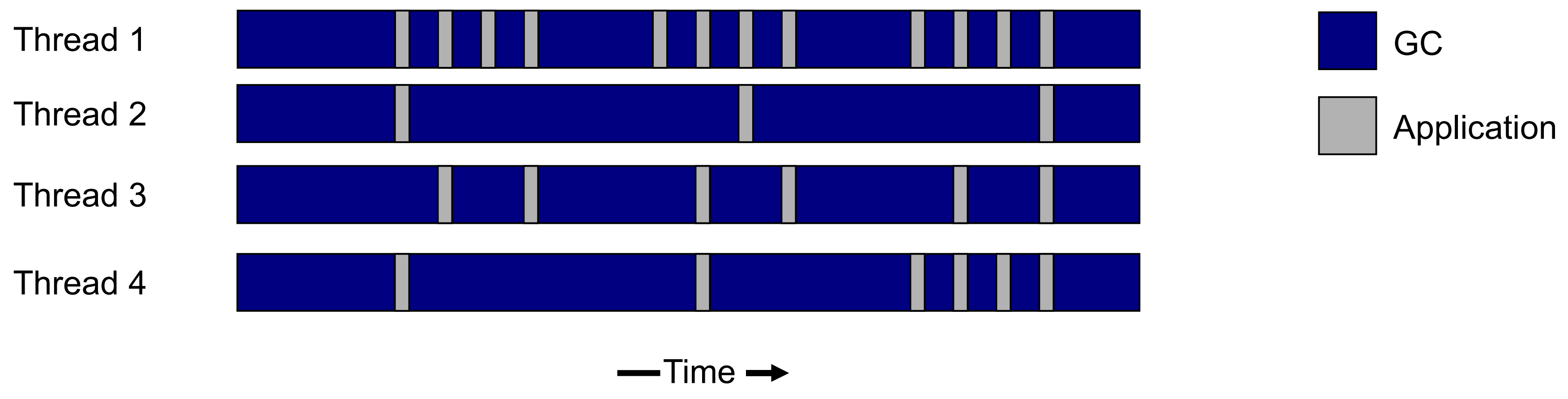
Garbage Collection Policies
Incremental Collection

- Run Application for Short Periods of Time
- Perform Very Small, Partial Collects Very Often
- Garbage Collection Keeps Up with Creation
- Collection Pauses are Consistent



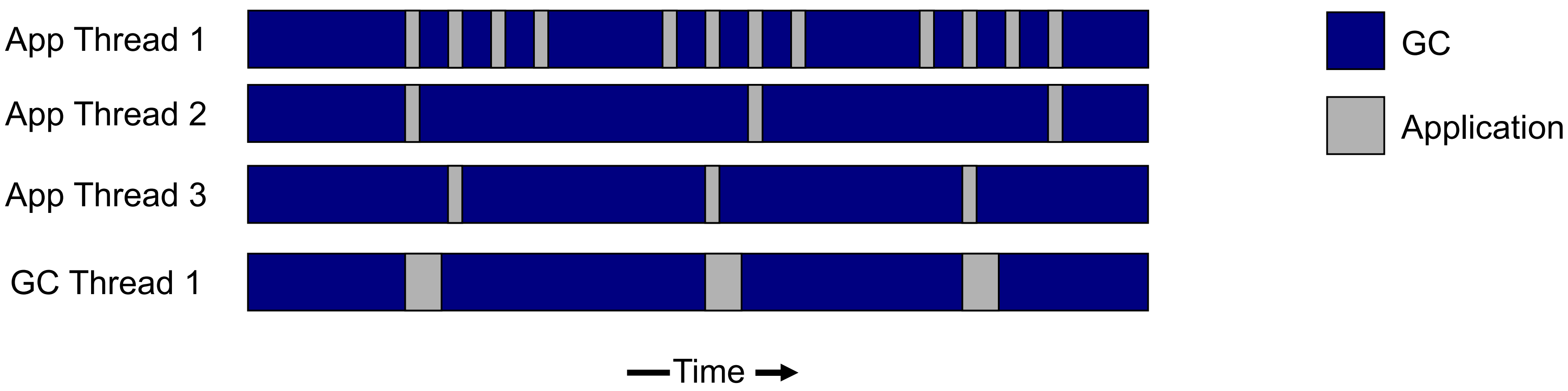
Garbage Collection Policies
Work-Based Collection

> Free Space Tracking on a per Thread Basis
> Trigger Thread Collect at Allocation Point
> Typically Thread-Based Incremental Collection



Garbage Collection Policies
Event-Based Collection

- Application is Designed as an Event-Based System
- Garbage Collection is Scheduled as Another Event
- GC Algorithm could be STW, Incremental, ...



Real World Garbage Collection

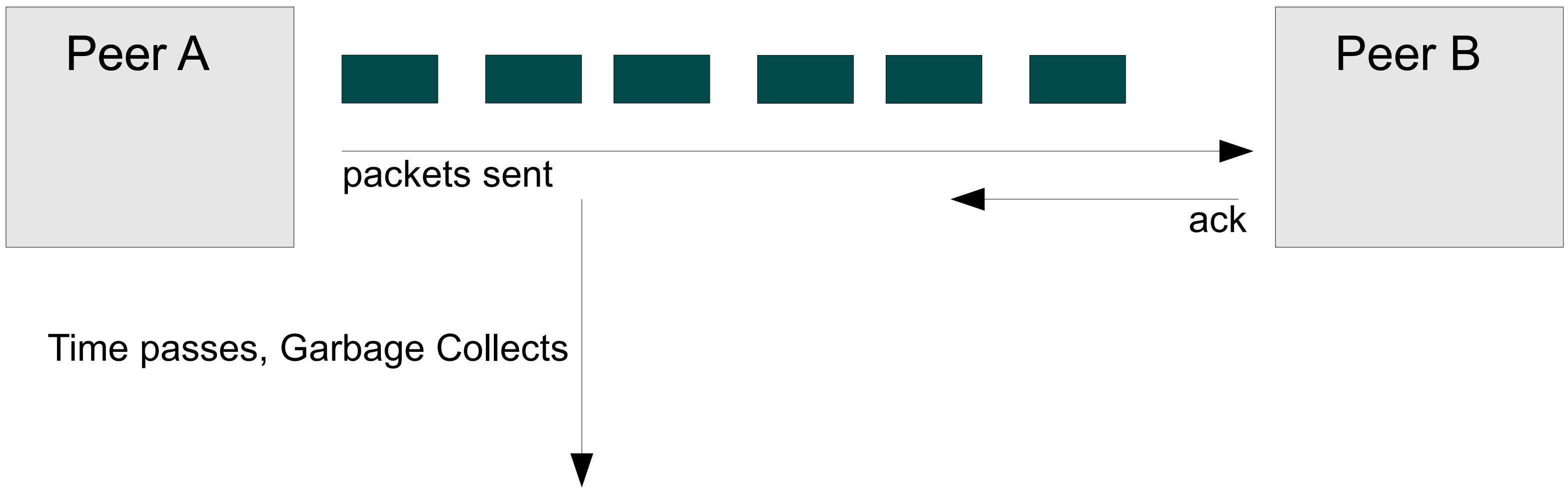
Comparing Real-Time Incremental to Generational GC

- The following slides show the effects of GC
  - Session Initiation Protocol (SIP) Server
  - Processing Incoming Phone Calls
  - Compares IBM Generational GC to Incremental GC

SIP (Session Initiation Protocol) Server

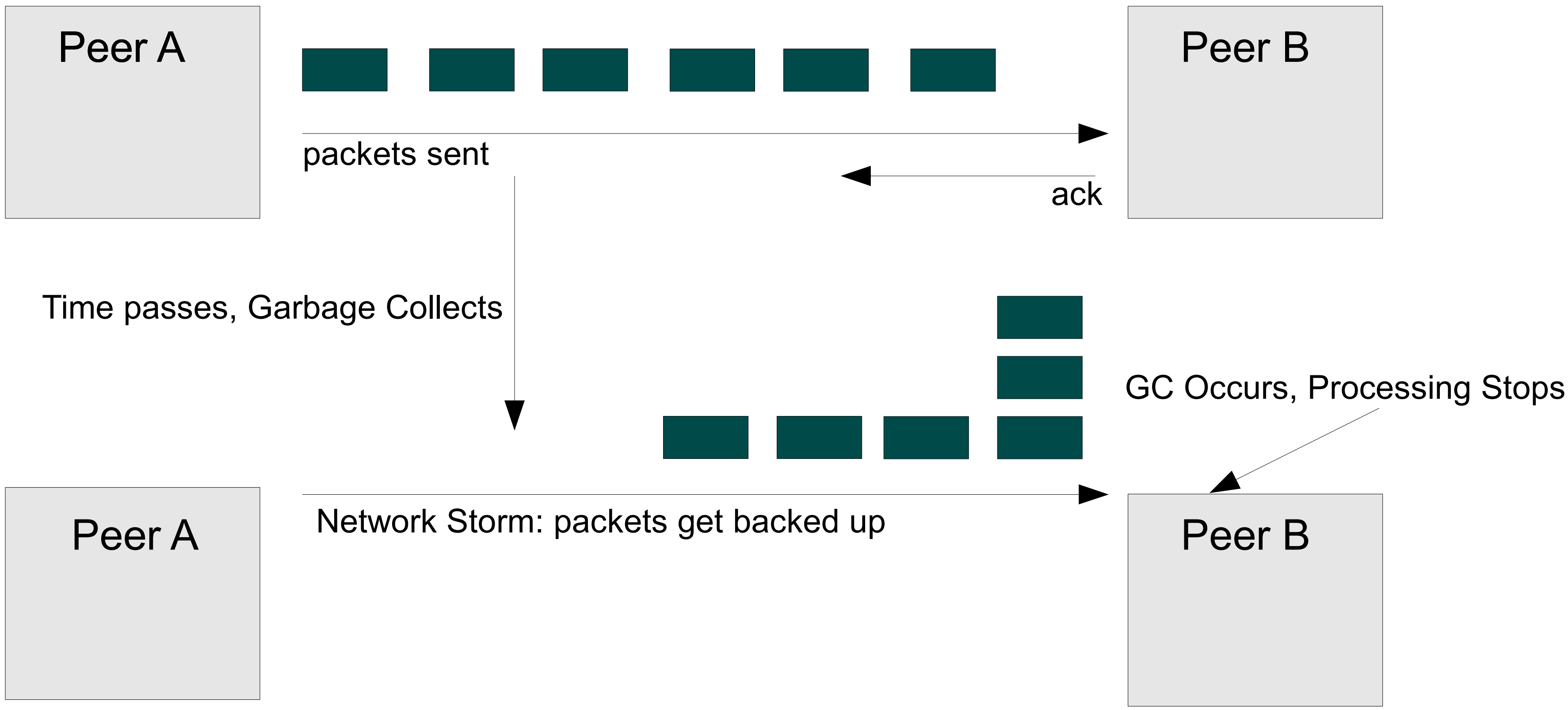
Real System running with Generational GC

SIP (Session Initiation Protocol) Server
Real System running with Generational GC



SIP (Session Initiation Protocol) Server

Real System running with Generational GC



SIP (Session Initiation Protocol) Server

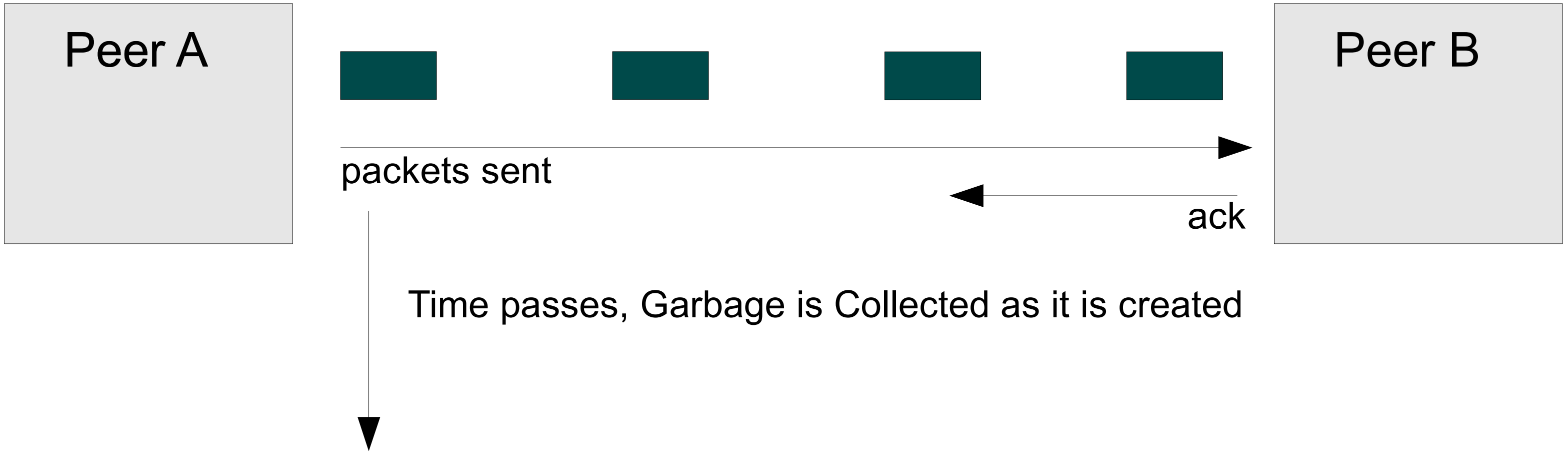
Real System running with Generational GC

SIP (Session Initiation Protocol) Server

Real System with Real-Time Incremental GC

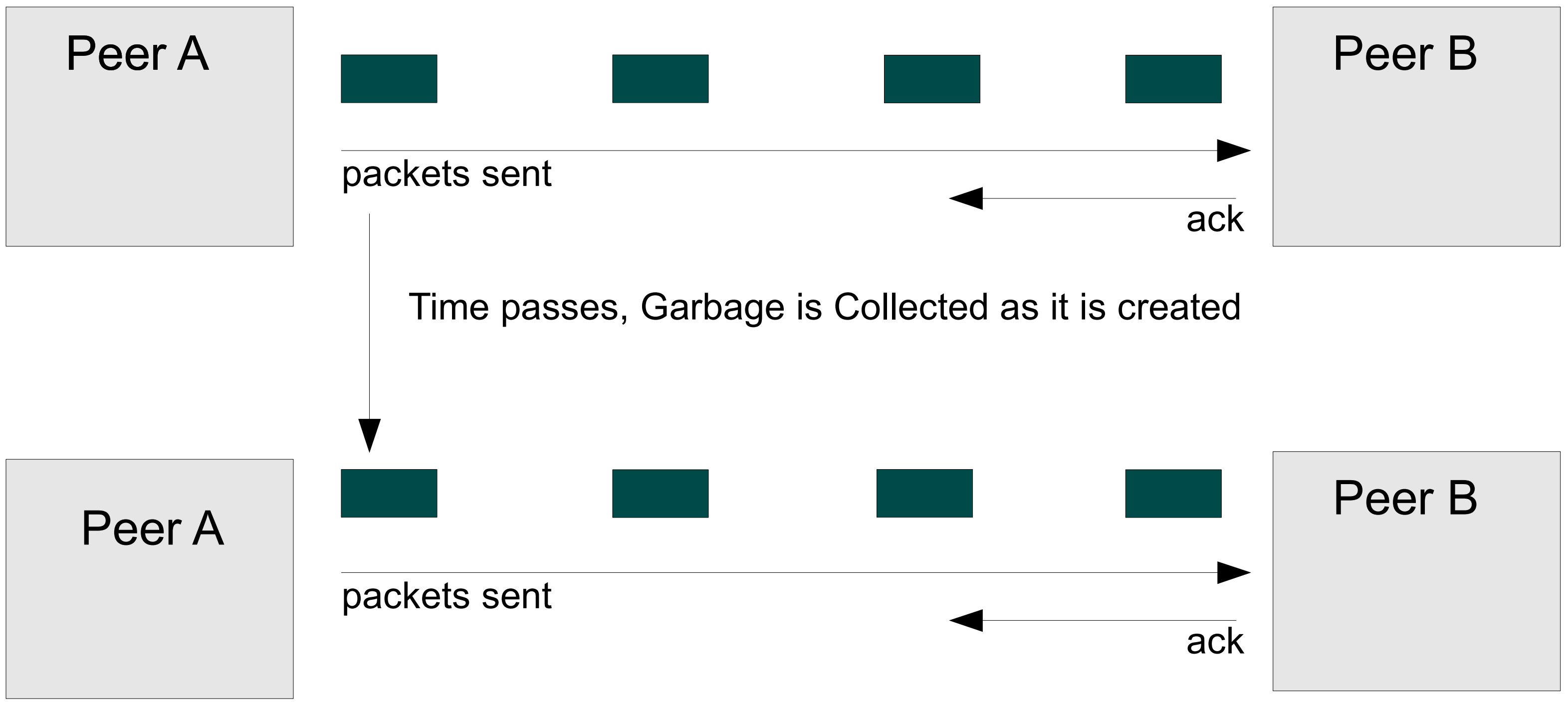
SIP (Session Initiation Protocol) Server

Real System with Real-Time Incremental GC

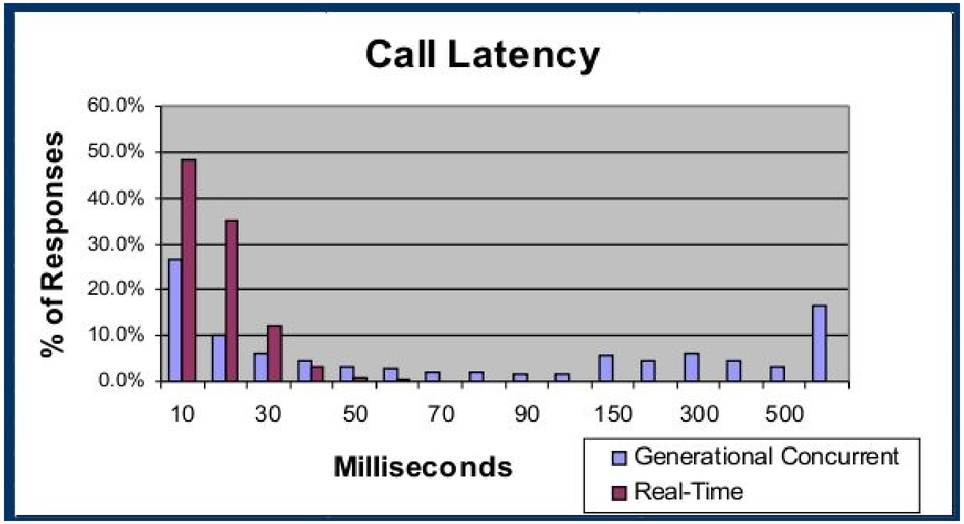


SIP (Session Initiation Protocol) Server

Real System with Real-Time Incremental GC



Real SIP Server Performance Results
Generational GC Compared to Incremental GC



Throughput:

Real-Time throughput less than Generational

Maximum Latencies

Real-Time less than 100ms

Generational less than 1000ms (1s)

Latencies greater than 50 ms:

Real-Time 0.3%,

Generational 50%

Real-Time (Incremental) GC has slightly less throughput than Generational

•But 98% reduction in standard deviation of GC pause times

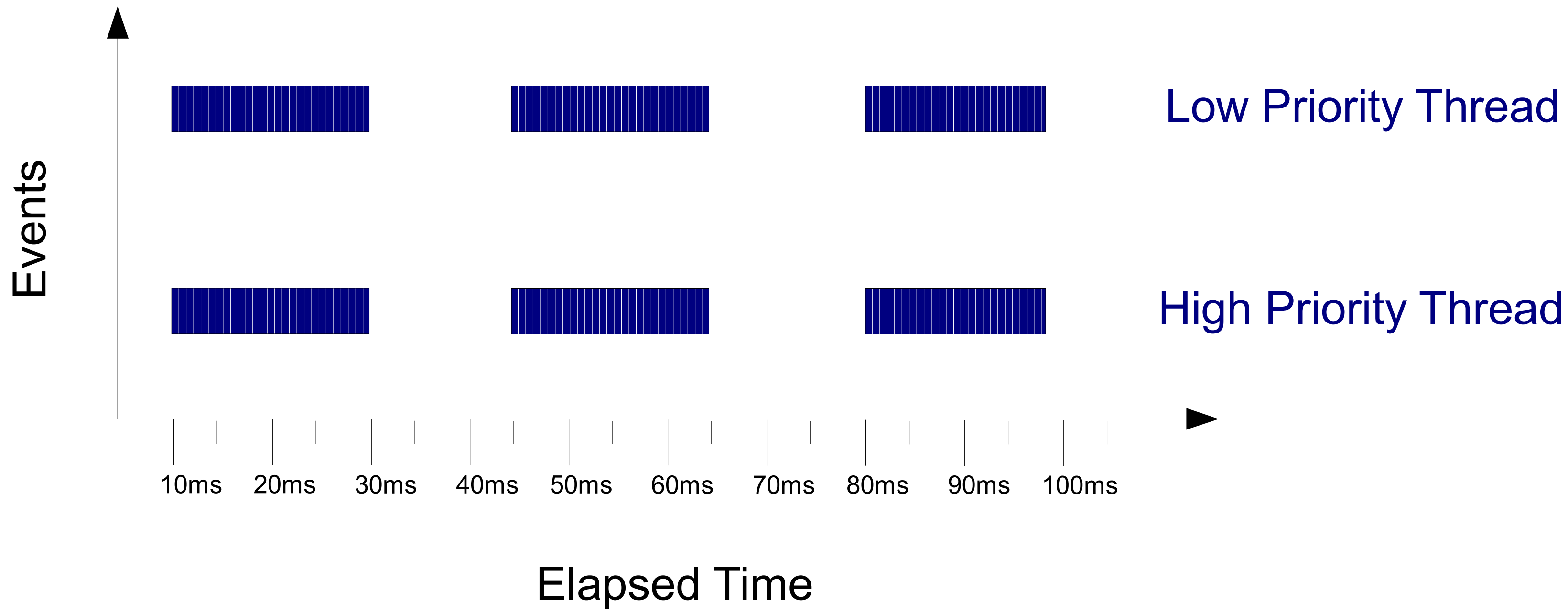
Reduced pause times results in reduced latencies

Real-Time Thread Scheduling

- Java does not mandate a scheduling policy
  - Low priority and High priority work runs together
  - Many JVMs use SCHED_OTHER *ix policy
- Real-Time JVMs Expose Scheduling Policies
  - In particular:
    - RTSJ JVMs provide SCHED_FIFO RealTimeThread
    - Could alternately run Java Threads SCHED_FIFO
- Thread Priority Scheduling is Critical

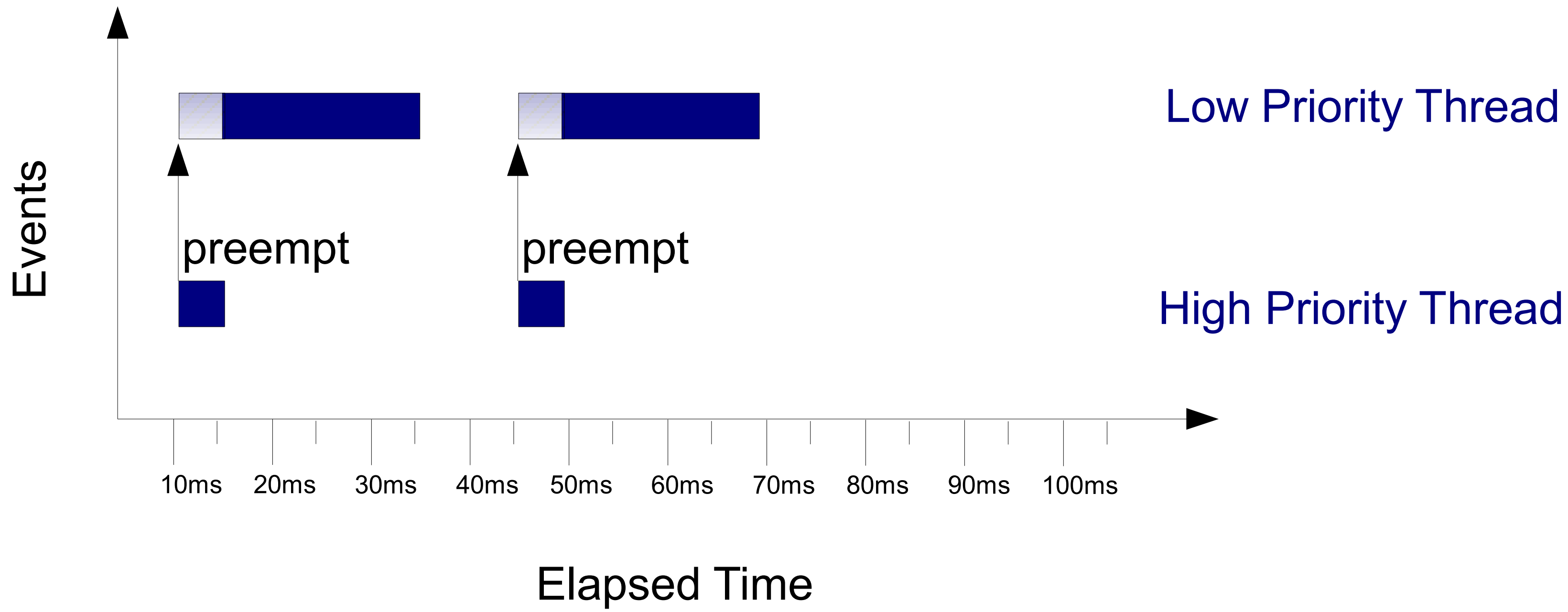
Java Application on Single CPU
Running Application Threads as SCHED_OTHER

Low and High Priority Threads Share CPU to complete work



Java Application on Single CPU
Running Application Threads as SCHED_FIFO

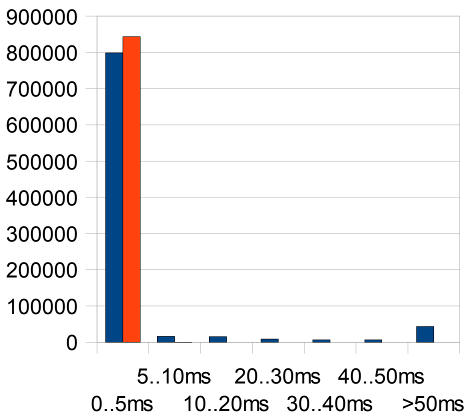
High Priority Thread Takes Control and Preempts Low Priority Thread
•High Priority Thread completes quicker
•Low Priority Thread takes longer to complete because it was preempted



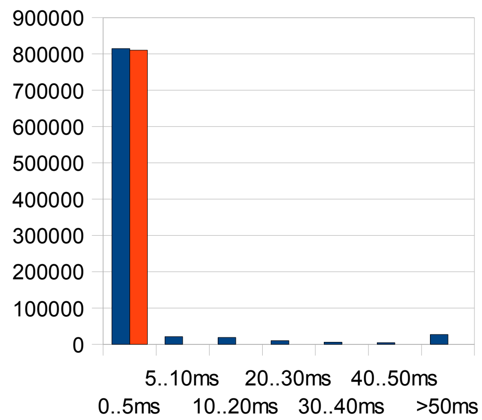
Real World Java Messaging Application
Comparing SCHED_FIFO and SCHED_OTHER

- Application for publishing 3K, 4K, 5K messages
  - Identical binary, RHEL 5.1, IBM Real-Time JVM
  - Java threads run SCHED_OTHER, SCHED_FIFO

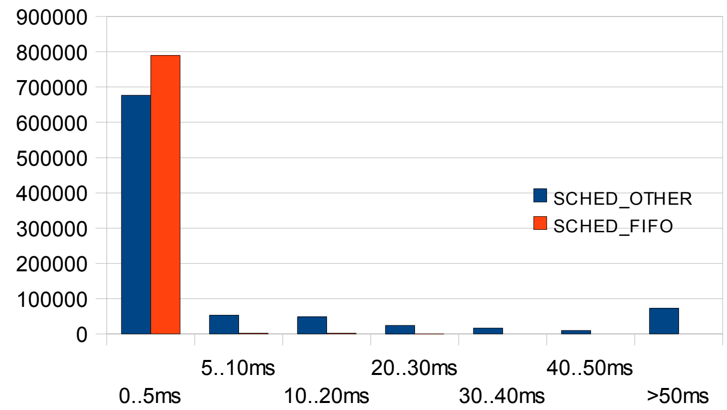
3K Message Size



4K Message Size



5K Message Size

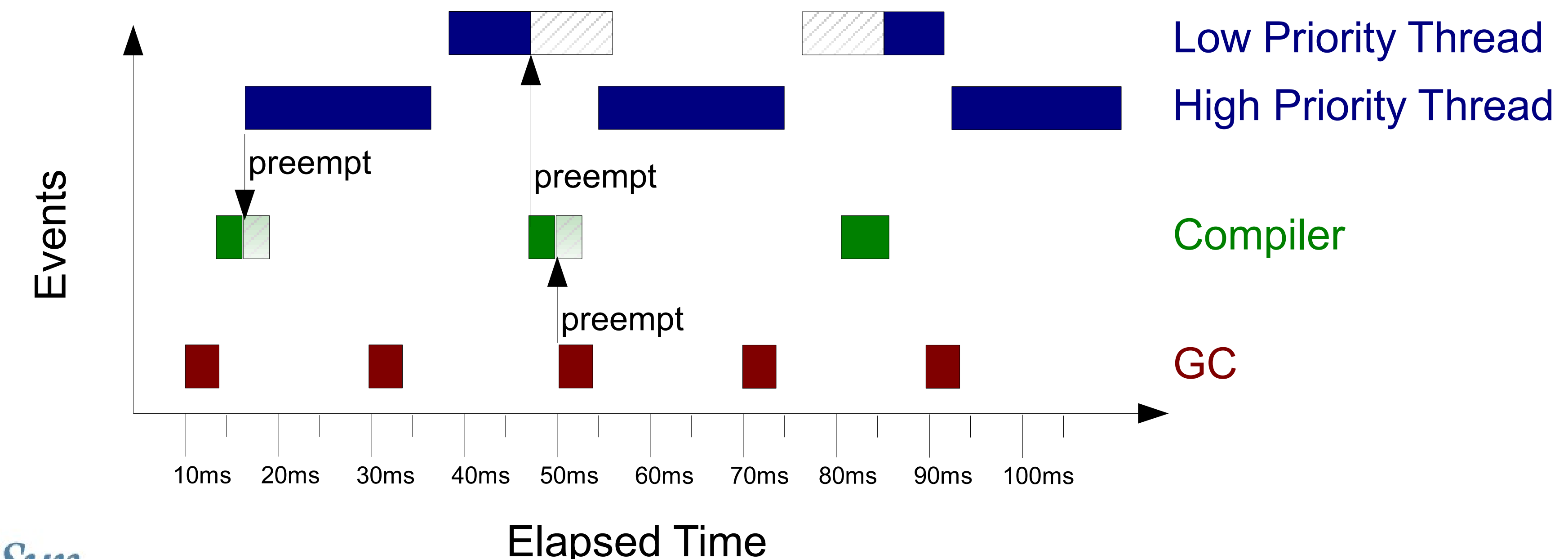


Compilation Approaches

- A: Interpreter Only, Ahead-of-Time Compilation
  - Conservative, easy to analyze, lower throughput
- B: Dynamic Compilation at Start Up
  - Higher throughput, Deterministic, Slow Start-up
- Real-Time Dynamic Compilation
  - Highest throughput – good supplement to A or B
  - Should Provide:
    - Compilation on Separate Thread
    - Incremental Compilation that can be suspended
    - Compiler capable of being preempted by GC or App

Real-Time Compilation
Blended Compilation Strategy

- Ahead of Time Compilation for fast start
- Code Compilation/Class-Loading at Start-up
- Incremental, Preemptable, Dynamic Compilation



Are All Operating Systems the Same?

Why an RTOS Can Be Critical

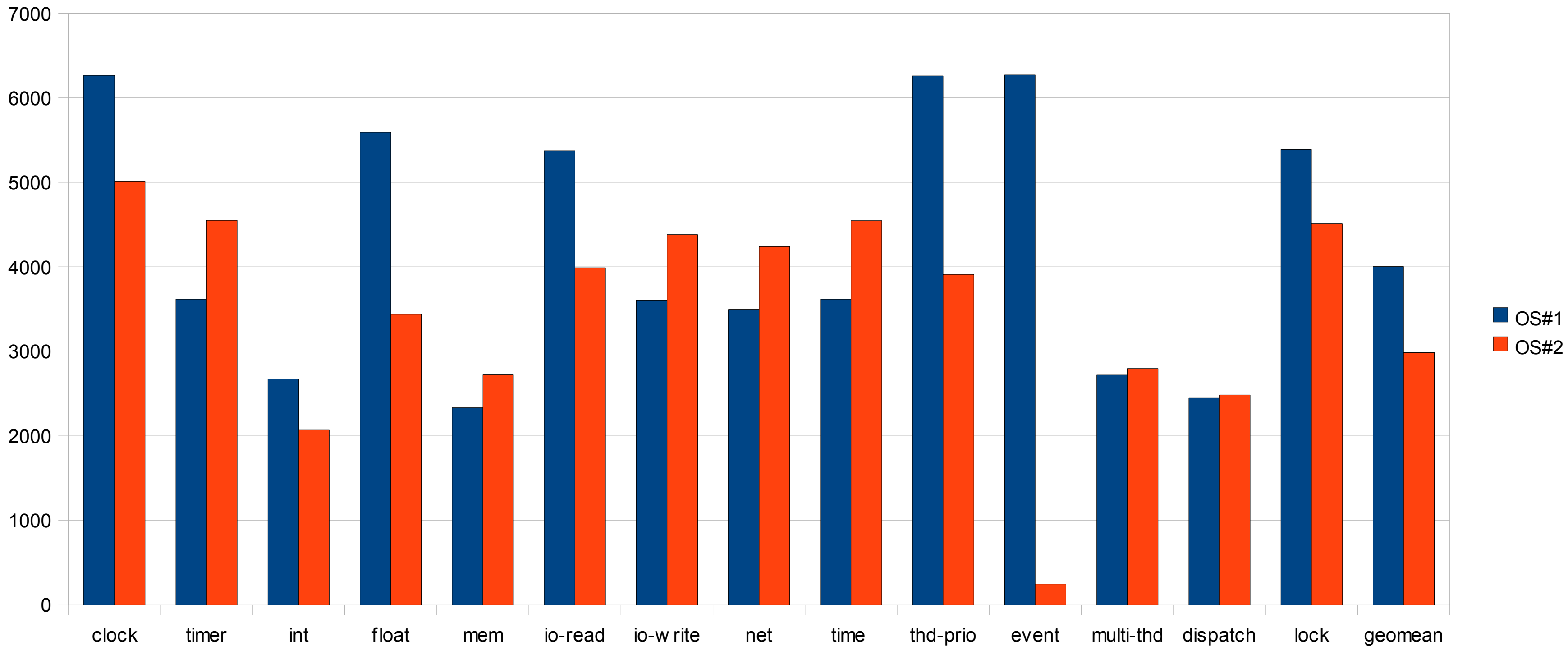
- Consistency of System Services Matters
  - Time-of-Day Clock, Sleep Very Important
    - Dispatch accuracy of system/application events?
    - In Java, what is the accuracy of System.nanosleep()?
    - Ranges from sub-microsecond to tens of milliseconds
  - Accurate systems not completely free
    - Caching algorithms disabled for consistent operation
    - Otherwise 1st invocation much slower than 2nd
  - Real-Time Industry Benchmarks being developed
    - Measure the Determinism of JVMs and OSs

Real-Time Micro Benchmark C Results

for two popular operating systems

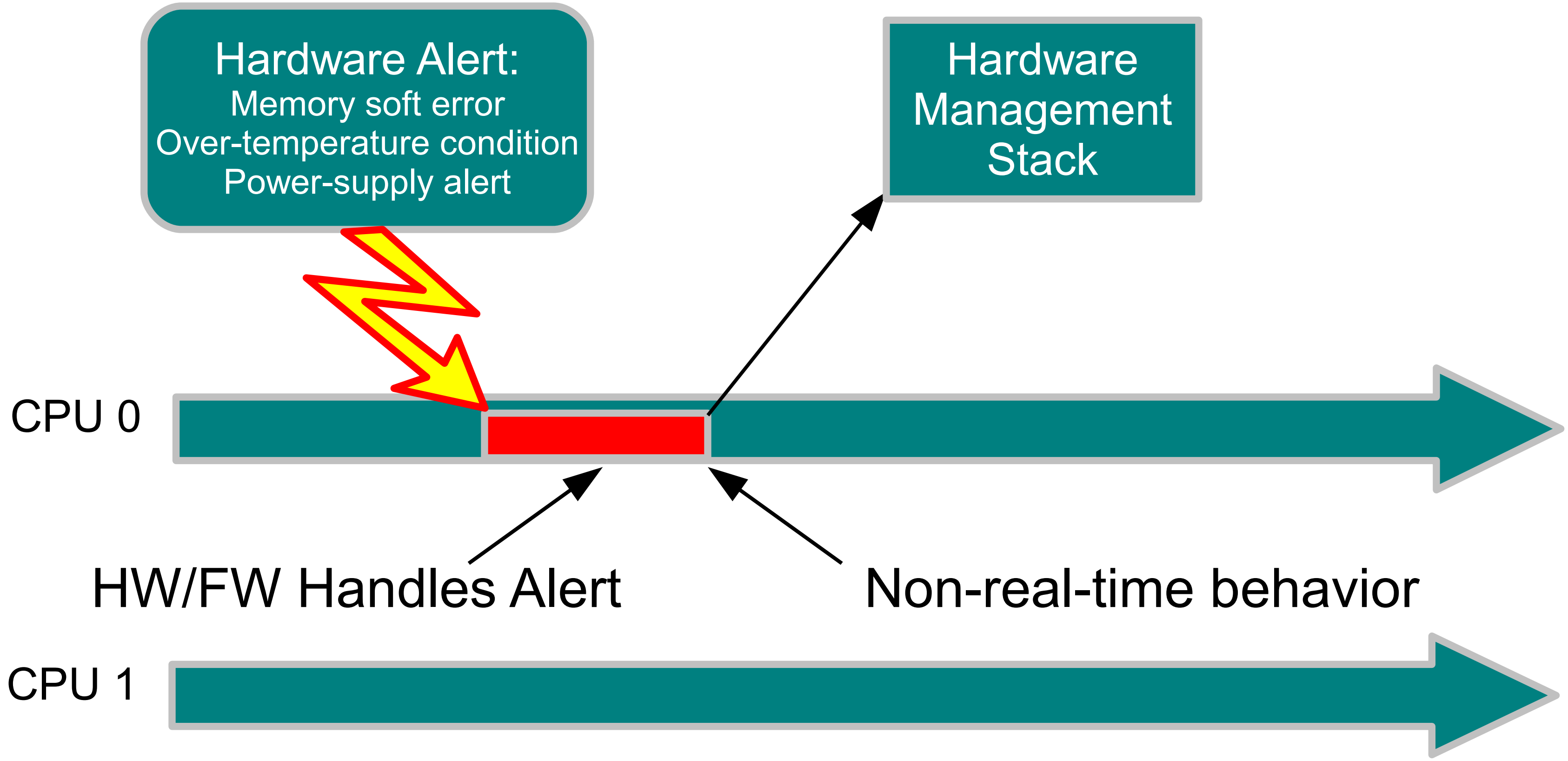
> Real-Time C Benchmark

- Suite of Micro-Benchmarks measuring determinism



Is All Hardware the Same?

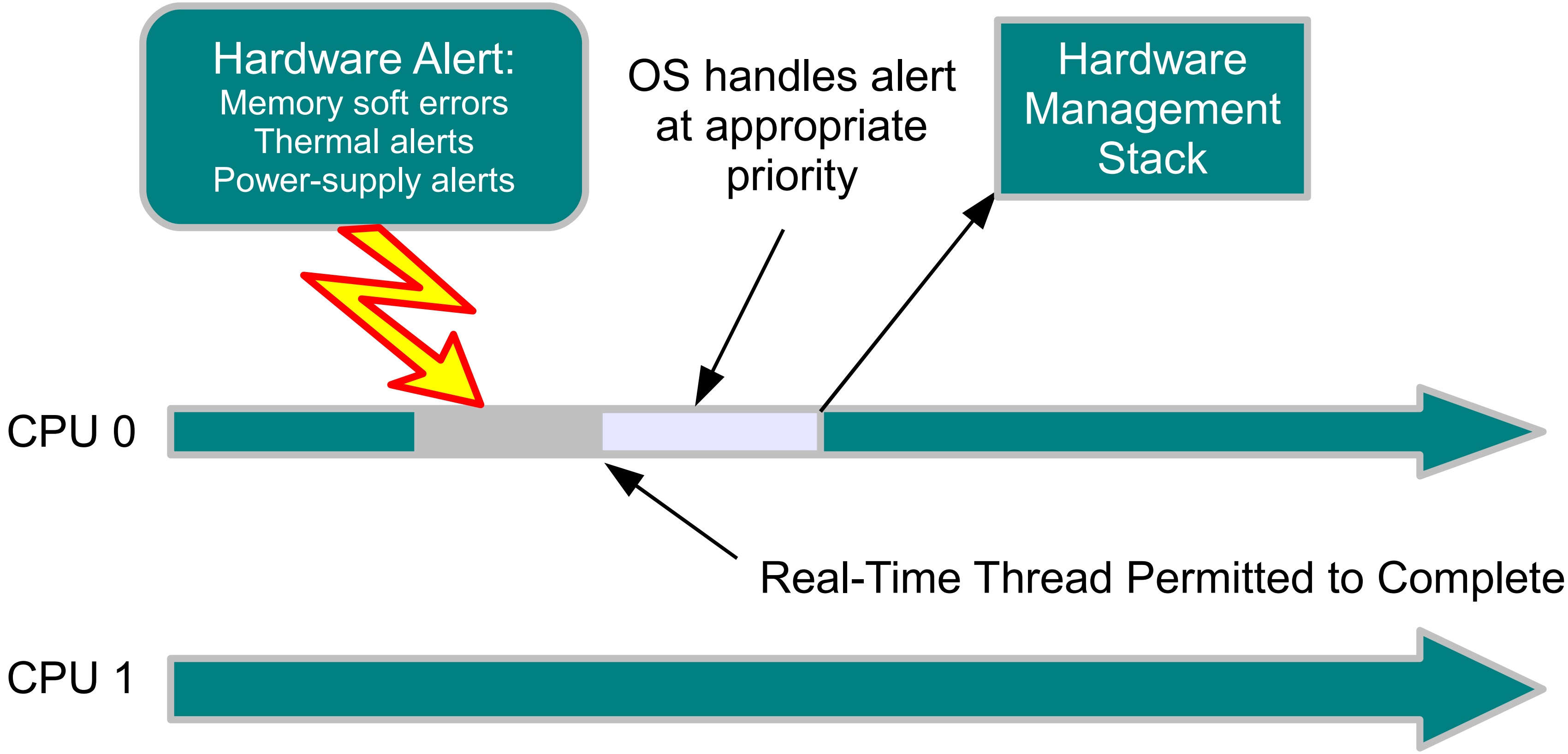
What Can Go Wrong with Hardware Interrupts



There is nothing that the OS or higher-level software can do to make up for this HW/FW non-realtime behavior.

Is All Hardware the Same?

Priority-Based Hardware Interrupts



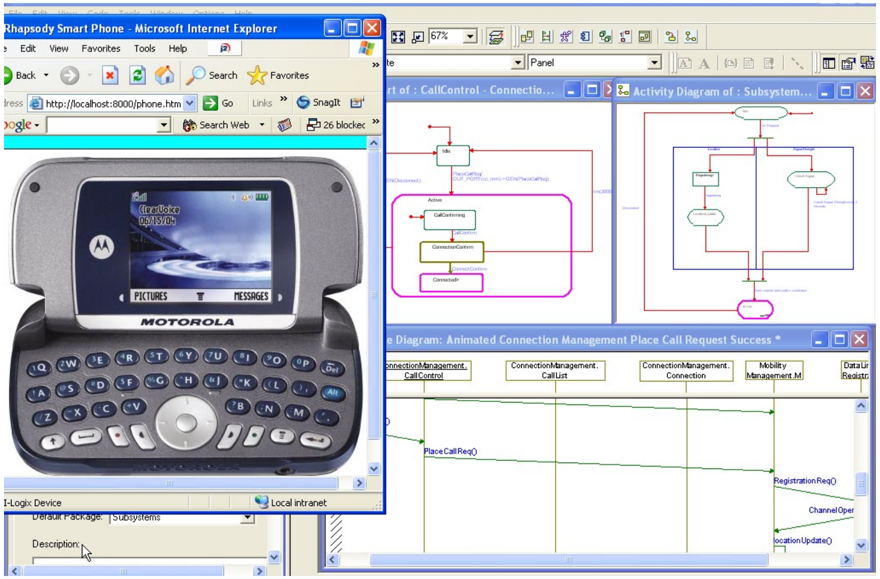
The OS and higher-level software now see Real-Time behavior.

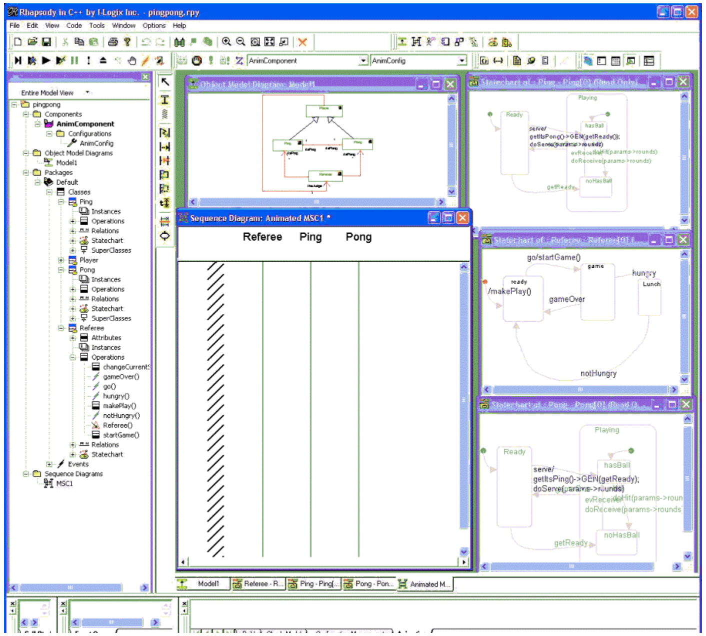
Does Real-Time Need Specific Tools?

- Yes!
  - Real-Time Modeling Tools
    - Tailored for creating event-driven applications
  - OS/JVM Tracing Tools
    - Find performance outliers, not throughput issues
    - Traditional Performance Analysis based on averages
      - Statistical approaches like sampling work very well
    - Worst-case Execution Time Analysis focus on outliers
      - Sampling is of little value

Summary

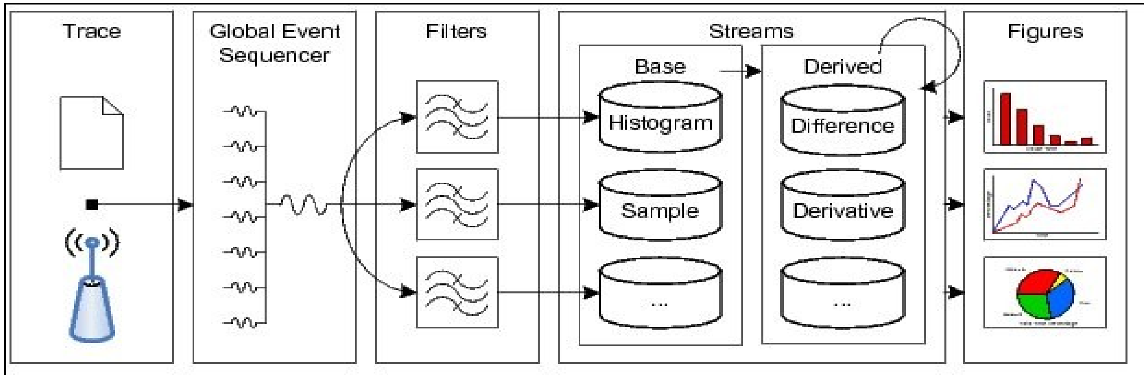
- Model Event-Based Systems
  - Simulate/Trace events using models
  - Define real-time event dispatch / scheduling





Real-Time OS/JVM Tracing

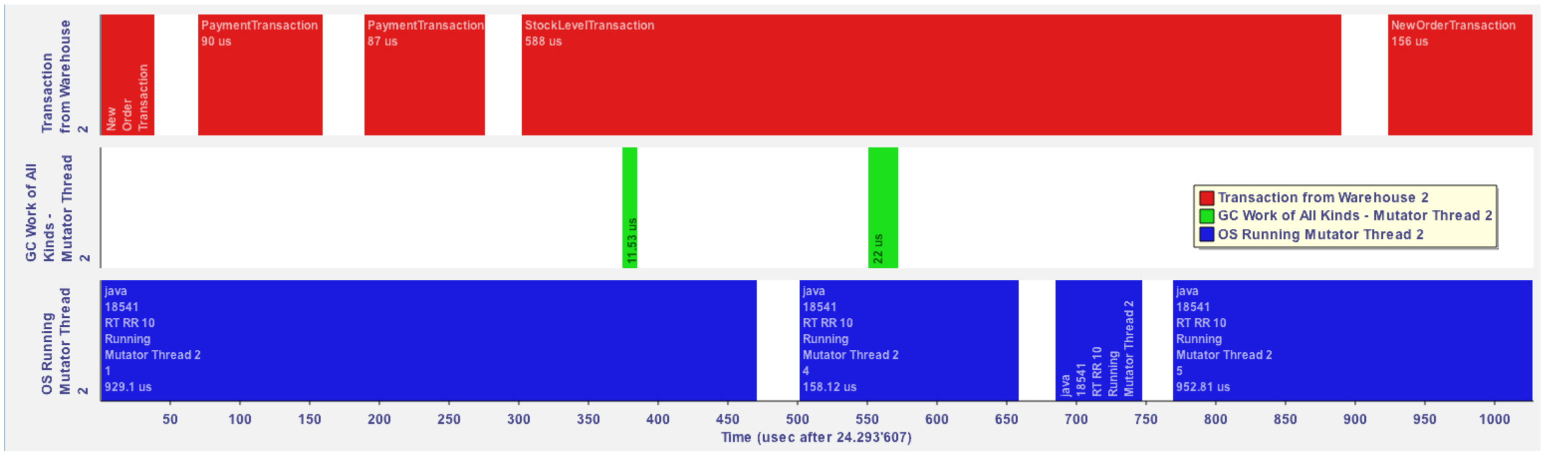
> Very Low Overhead Trace Daemon
  - Capture data at Application, JVM, OS Level
  - Transmit data on low priority socket to other OS
  - On other OS, process event stream



Real-Time Outlier Detection

Outlier Analysis : Diagnosing an application outlier

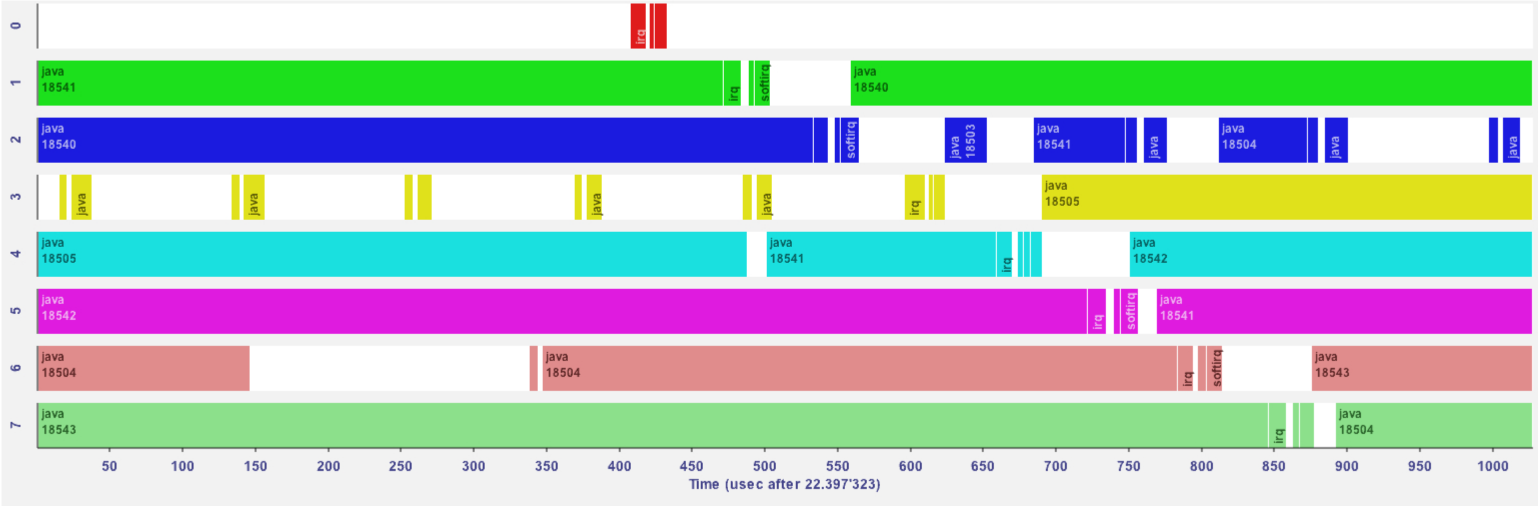
- 8 CPU System with Single 588µs Outlier (red)
  - Drilling down to event trace at point of outlier
  - Initial thought is GC (green) causing interference



Real-Time Outlier Detection

Outlier Analysis : Diagnosing an OS Outlier

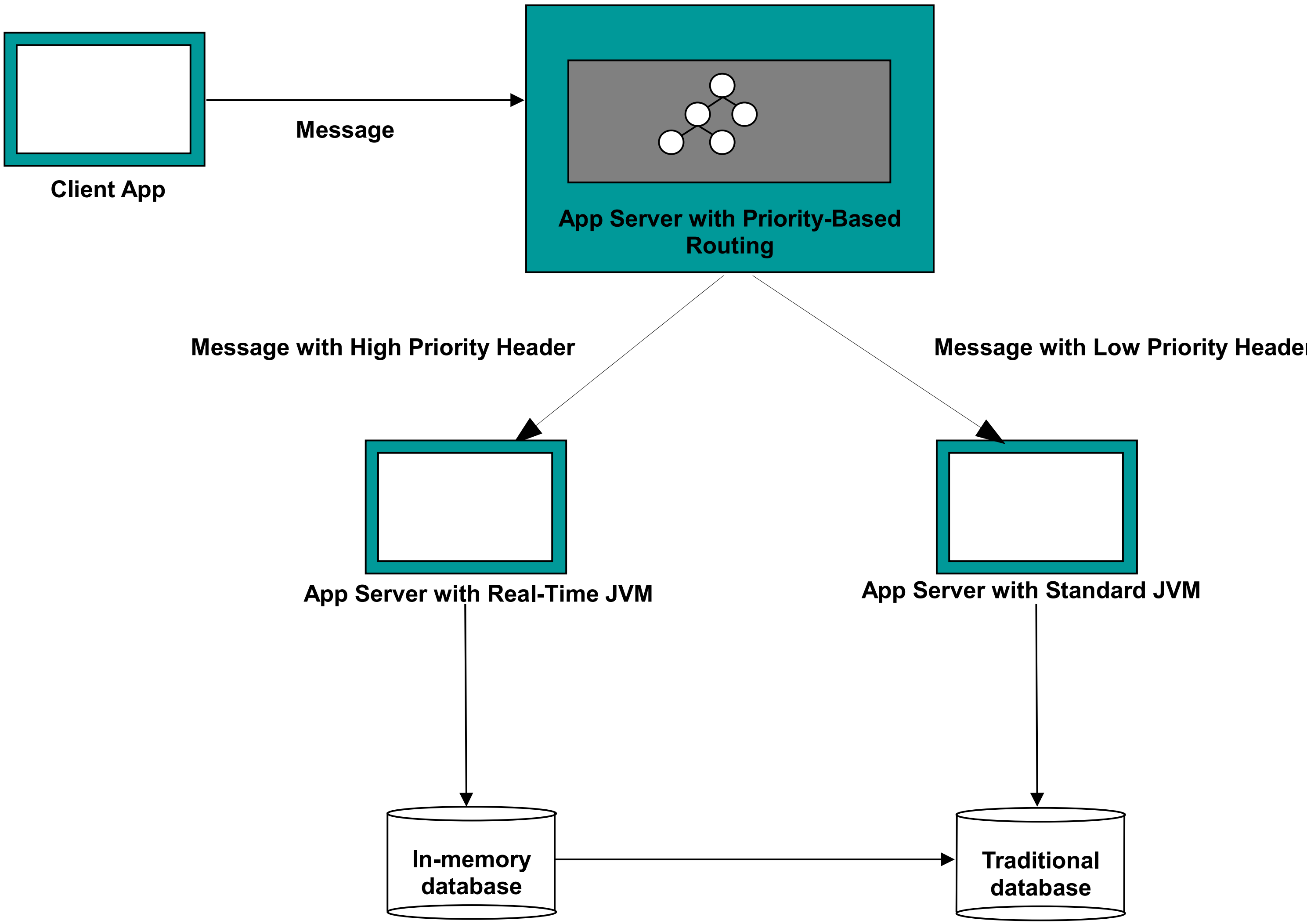
- 8 CPU System with Single 588µs Outlier
  - Drilling down to OS event trace at point of outlier
  - Rolling IRQ across CPUs causes Java process bump
  - Process pre-empted across 4 CPUs in turn (1->4->2->5)



Real-Time Middleware

- Variety of Real-Time Middleware Available
  - Some runs 'as-is' on RT JVMs
    - Better Determinism 'for free'
  - Some middleware exploits RT JVM Capabilities
    - Priority-based routing in Application Servers
  - Next Generation Extreme Transaction Processing
    - Working with huge data sets – hundreds of gigabytes
    - Performing Complex Event Processing in Real-Time
    - Will be running on Real-Time Systems Developed today

Real-Time Application Server
Priority Based Routing



Summary

What I hope you gleaned from my ramblings

- Most applications can benefit from Real-Time Java
- JVMs require core enhancements for real-time
    - The OS, hardware, and middleware are also key
- Real-Time has distinct tooling demands
- The benefits of real-time are real, not theoretical

JavaOne[SM]

Thank You

Mike Fulton
fultonm@ca.ibm.com

IBM Canada Ltd.