*WebSphere User Group*

*Edinburgh*

# ***Real-time Java***

## ***How to Avoid Unexpected Delays***

**Mike Fulton**

IBM Real-Time Technologies

## *Disclaimer*

All statements regarding IBM's future plans, direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Such statements do not represent a commitment of future availability, content, performance or function of any products or features.

# *Agenda*

- IBM’s perspective on real-time Java
- WebSphere Real Time
- IBM’s Full Real Time Offering
- More great technology from the lab
- The road ahead

# *IBM’s investment in Java*

Java is building block for hundreds of IBM applications

- Provides a consistent ‘operating system’
- Safe, efficient language for developing code
- Consistent, high quality tooling for all dev phases

Significant Performance work

- Heavy investment in JVM, GC, JIT, Class Lib
- Hardware ranges from MIPS,ARM,SH4 to x/p/zSeries
- JVM designed for easy target to new OS and HW

# *What does* real-time *mean?*

*real-time* : predictability of performance

- *hard* : violation of timing constraints are hard failures
- *soft* : timing constraints are simply performance goals

Constraints vary in magnitude (microseconds to seconds)

Consequences of missing a timing constraint:

- from service level agreement miss (stock trading)
- to life in jeopardy (airplanes)

*Real-fast is not real-time*, but *Real-slow is not real-good*

Need a balance between predictability and throughput

# *IBM's interest in real-time*

Classical real-time systems are getting more complex

- Military, telecom, industrial, automotive, gaming

Real-time systems becoming part of enterprise IT

- Sensor networks, Event processing

Commercial systems have unpredictable performance

- Service Level Agreement failures when overloaded

A need for a new way to build real-time systems

- Engineered for predictability and reliability
- Using the latest programming tools and techniques

## *Why Java?*

A business advantage over C, C++, Ada

- Productivity from tools, portability, error checking, security
- Many skilled programmers available
- Massive community of ISVs

Java has problems in real-time environments

- Lazy class loading and initialization, dynamic compilation
- Garbage collection, system-specific thread management

IBM has solved these problems

# *WebSphere Real Time*

WebSphere Real Time (WRT) V1 is Generally Available

WRT is a Highly Predictable Java runtime:

- Real-time garbage collection
- Static and dynamic compilation
- Full support for RTSJ (JSR #1)
- Java SE 5.0 compliant
- Rigorously tested on Red Hat MRG & Novell SLERT
  - using IBM xSeries hardware

# *Standards Compliance*

IBM is committed to Industry Standards

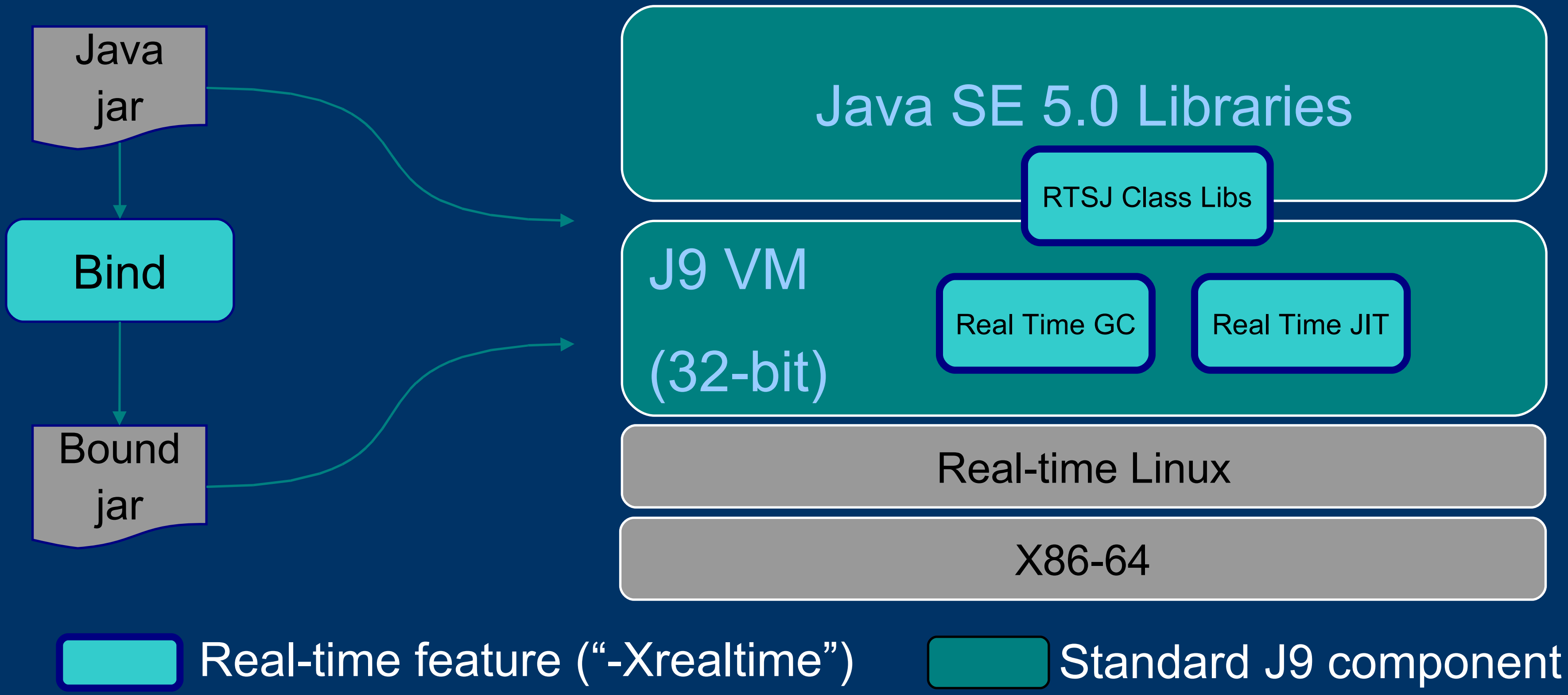
WebSphere Real Time JVM is fully Java SE 5.0 compliant

- Fully conformant JVM that runs on Real Time Linux
- The **-Xrealtime** option gives additional Real Time function
  - Conformant to JSR #1: Real Time Specification for Java

Java applications will run under WebSphere Real Time

- ... but will have more predictable performance
- ... and can be extended, where required, to use RTSJ

# *WRT architecture*



Java jar

Bind

Bound jar

Java SE 5.0 Libraries

RTSJ Class Libs

J9 VM

(32-bit)

Real Time GC

Real Time JIT

Real-time Linux

X86-64

Real-time feature (“-Xrealtime”)

Standard J9 component

# Metronome Garbage Collection

- Unique technology from IBM R&D
  - Garbage collection is scheduled as just another periodic real-time task
  - Provides bounded pause times as small as 1ms and a minimum utilization level for application tasks
  - Exploits RTOS hi-res timers and scheduling
- Enables the use of off-the-shelf Java code
  - No need for specialized allocation schemes outside the Java heap
  - Greatly simplifies real-time application development
  - Enables complex real-time applications through easier composition

**Figure 1. Traditional GC pauses**

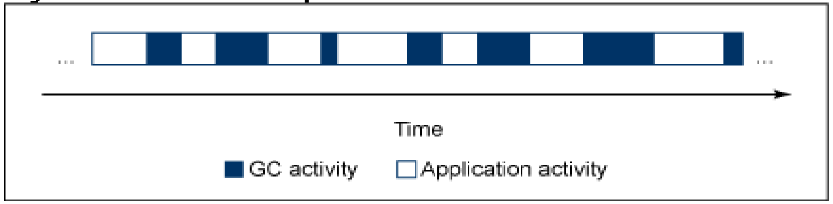


**Figure 2. Short pause times but little application time**

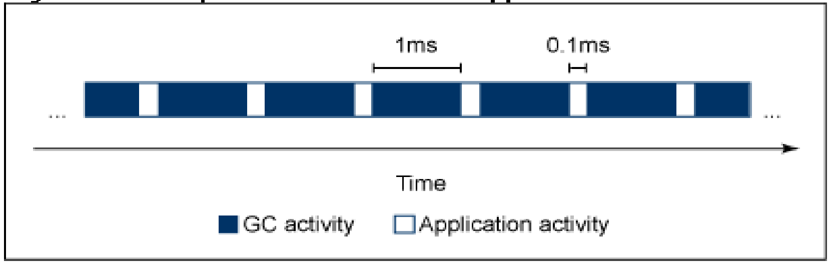
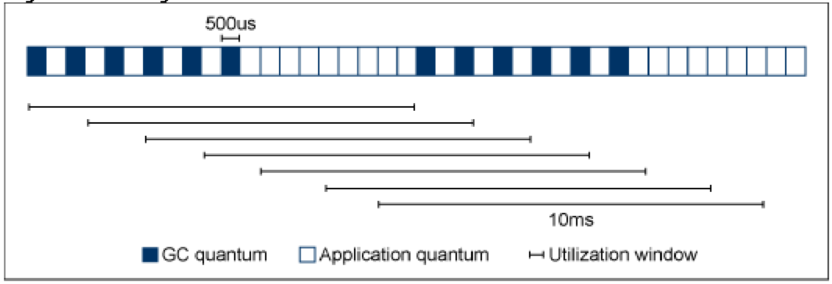


**Figure 3. Sliding window utilization**



# Compilation Strategies for Real Time

Compilation in J9 is dynamic by default

- High throughput, but JIT may not run early enough in non-real-time JVM to guarantee consistent performance

Multiple compilation choices with WRT:

- Ahead-of-time (AOT) (much better than interpreted performance)
- User-controlled JIT (faster than AOT, controlled via API)
- JIT-at-low-priority (best performance, runs on low priority thread)
- Tooling-controlled compilation as part of application start-up

# Real Time Specification for Java (RTSJ)

Augments Java language to support building real-time systems

Thread scheduling

- “RealtimeThread” allows specification of scheduling parameters
- Used in conjunction with Metronome, low latency achieved with no change in programming model
- Fixed priority scheduling and additional priority settings
- Many event management services provided

Memory Management

- Partitioned, non-garbage collected memory spaces
- No Heap Realtime Threads (NHRTs) can run independent of GC
- Very low latency achieved using standard RTSJ scoped memory techniques with NHRTs

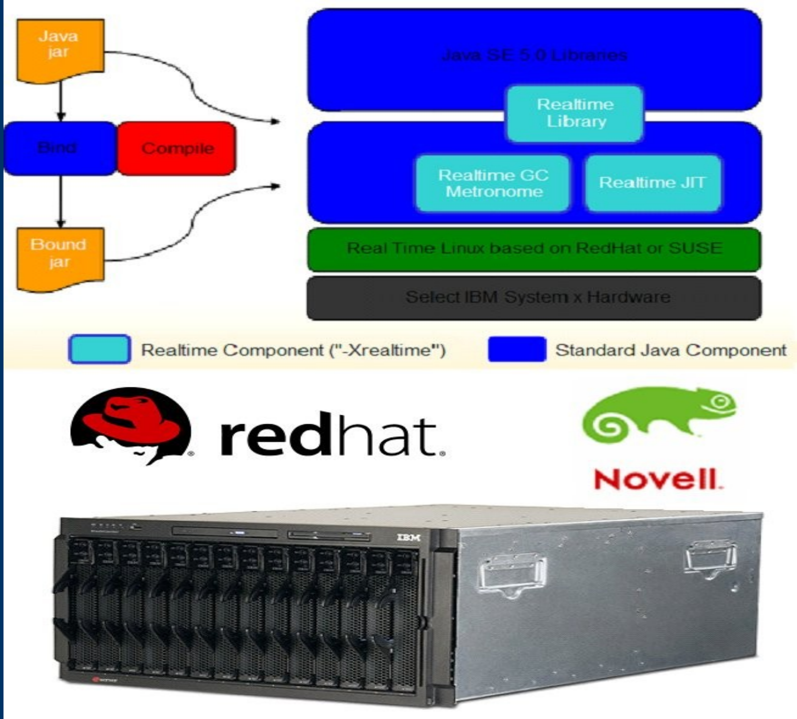
# *IBM’s Current Real-time Offering*

## The Power of Java and Linux Combined to Deliver Real-time Capabilities

Select IBM Hardware
- LS21 and HS21XM xSeries blades
- Enhancements for real-time workloads

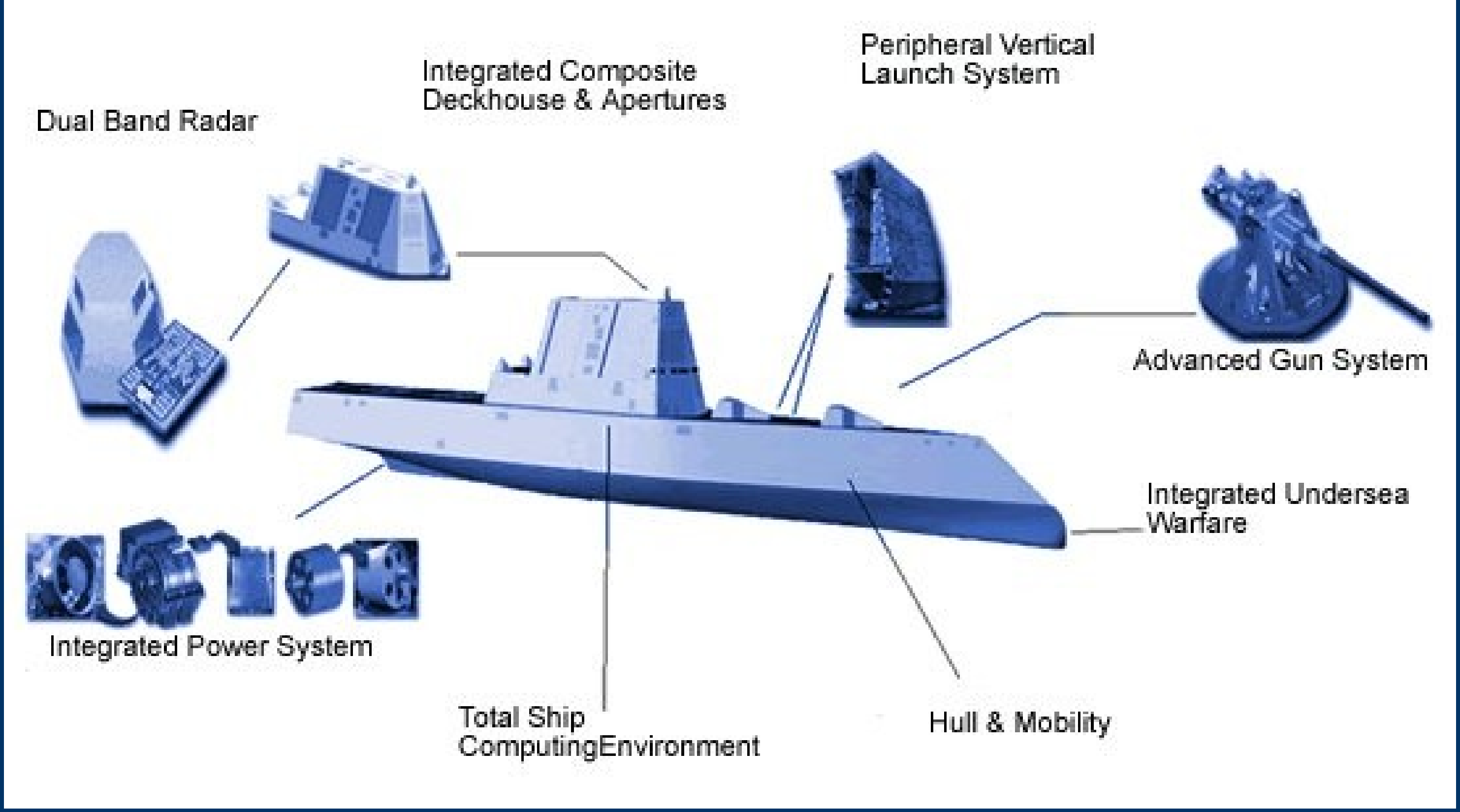
Real-Time Linux (RedHat MRG, Novell SLERT)
- High resolution time and timers
- Fully pre-emptible kernel
- Threaded interrupt handlers
- Priority inheritance & fast user-space mutexes
- Symmetric Multiprocessing (SMP) RT scheduling

WebSphere® Real-Time (WRT)
- Java 2 Standard Edition & IBM J9 technology
- Real-Time Specification for Java (RTSJ: JSR 1)
- Metronome Garbage Collector (GC)
- Real-Time Compilation Strategies (AOT, JIT)



# *WRT V1 In The Real World*

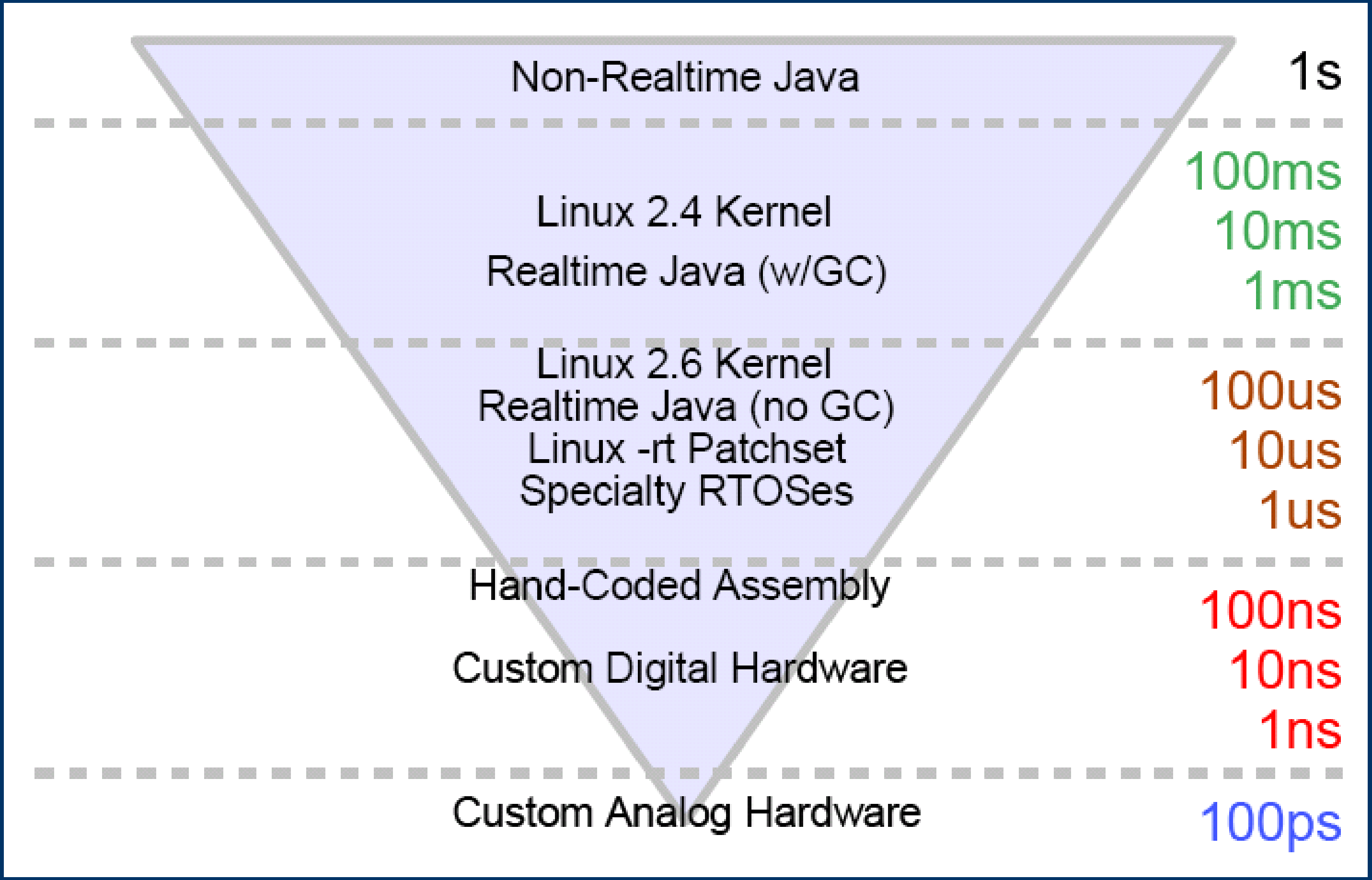
DDG 1000 Next Generation Navy Destroyers Developed with WRT on RT Linux


Dual Band Radar
Integrated Composite Deckhouse & Apertures
Peripheral Vertical Launch System
Advanced Gun System
Integrated Undersea Warfare
Integrated Power System
Total Ship ComputingEnvironment
Hull & Mobility

http://findarticles.com/p/articles/mi_pwwi/is_200702/ai_n17168257

http://www.raytheon.com/capabilities/products/zumwalt/index.html

# Real-time Capability Triangle



**Updated from: SMP and Embedded Real-time** (article in the Linux Journal)

by Paul McKenney (Distinguished Engineer, Linux Technology Center) http://www.linuxjournal.com/article/9361

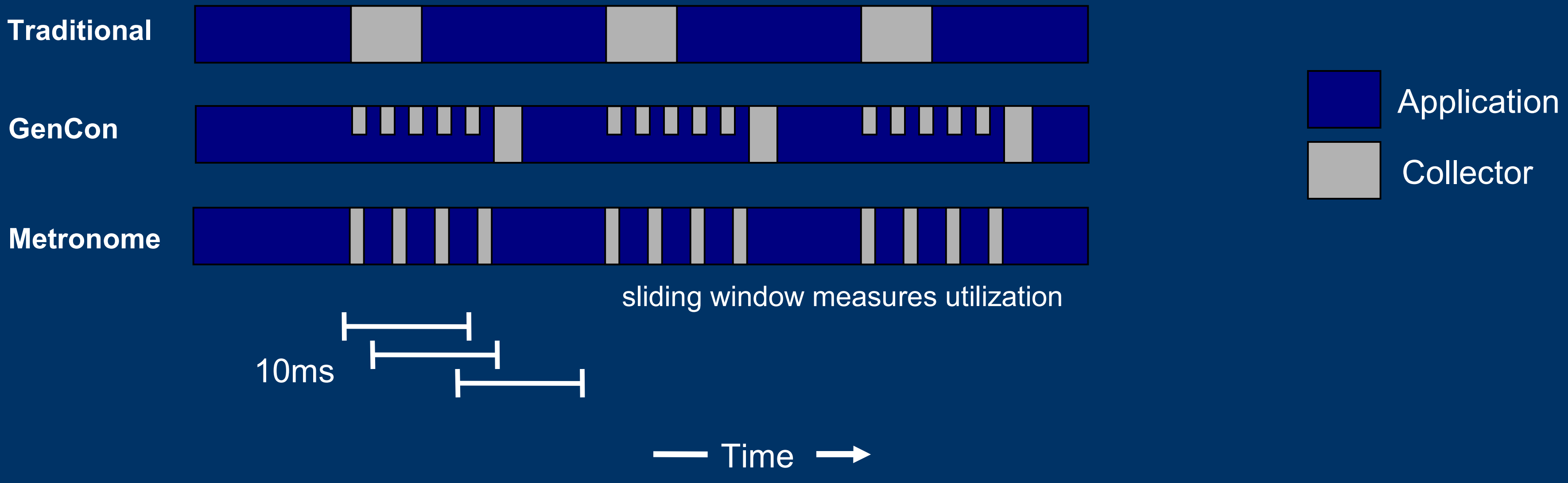
# WebSphere Real-Time V2

Development underway for WRT V2

- Continued support for the latest RTSJ (1.0.2)
- Continued support for the latest JSE (Java 6)
- Throughput/scalability improvements
  - Specifically in compilation and garbage collection
  - Exploitation of the largest xSeries blades
- Support for the latest xSeries blades, Red Hat and Novell RT distros
- Mixed AOT/JIT/Interpreter with shared classes
- *Soft* Real-Time Offering being added for Standard x86 Linux Distros
  - Available stand-alone and as part of WebSphere Virtual Enterprise
  - Provides Deterministic JVM without RTSJ for JSE 6 applications

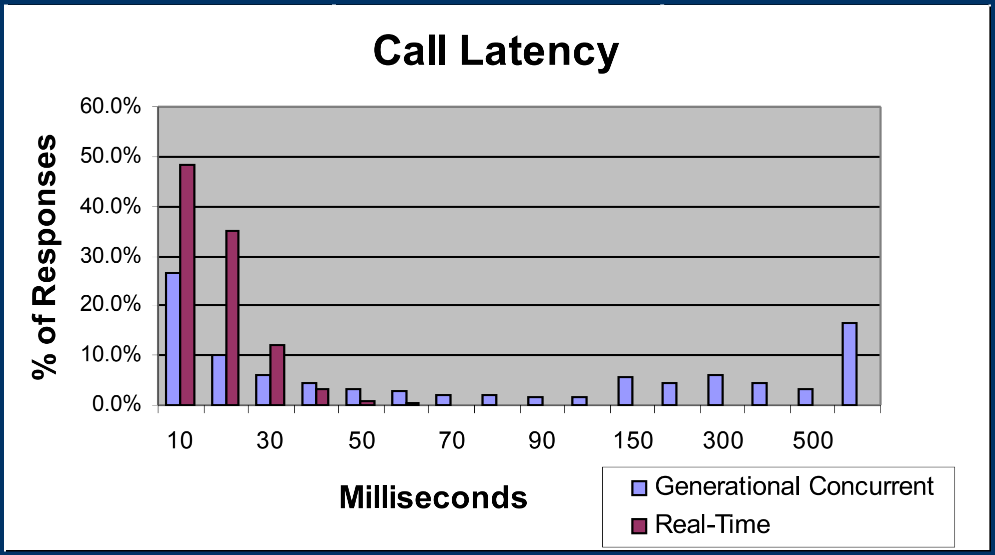
# *Comparison of Different Garbage Collection Policies*

- Traditional garbage collection requires a single Stop-the-World event
  - Stop-the-World: all Java threads stop to permit collection
- Generational Concurrent (GenCon) garbage collection
  - primarily shorter collections concurrent with application thread on multi-processor systems
  - very infrequent stop-the-world global collections, typically shorter than traditional garbage collection
- Metronome garbage collection guarantees maximum pause times with a minimum utilization
  - Utilization is processor time dedicated to the application
  - Shortest pause times, but may have greater performance impact



# *SIP (Session Initiation Protocol) Server Latency*

## *Real-Time GC Compared To Generational Concurrent GC*



Throughput:

Metronome ~ 150 calls per second

GenCon ~300 calls per second

Maximum Latencies

Metronome less than 100ms

GenCon less than 1s

Latencies greater than 50 ms:

Metronome 0.3%,

GenCon 50%

- Metronome trades off 50% performance capacity for a 98% reduction in average GC pause times, worst-case pause times and pause time variability
- Reduced pause times results in reduced latencies
- WRT V2 Throughput performance is significantly improved over these numbers
  - B2BUA benchmark on IBM HS21 blade server using the Websphere Real-Time for Linux early drivers compared to the IBM Java SE generational concurrent GC, both using ObjectGrid asynchronous replication. Latency values +- 0.03% error.

# *More great technology from the lab*

Now we turn to our futures work...

- Tuning fork:  Eclipse-based visualization of real-time systems
- Real-Time Class Analysis:  Class pre-loading and pre-compilation analysis
- Expedited Real-Time Threads:  A programming model for low latency tasks
- Application test beds:  We're pushing the envelope with Real-Time Java

## *Tuning Fork*

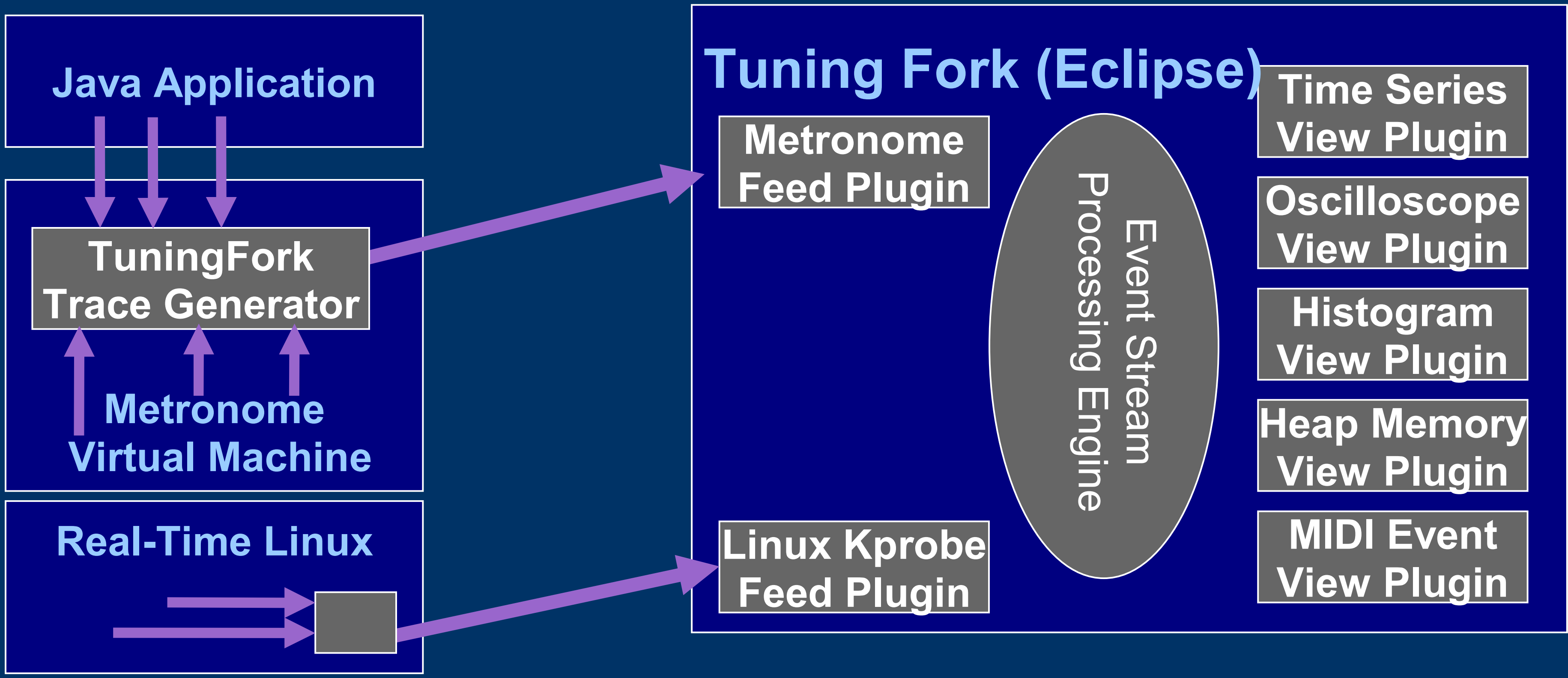
How do you debug a timing failure in a 25 MLOC application?

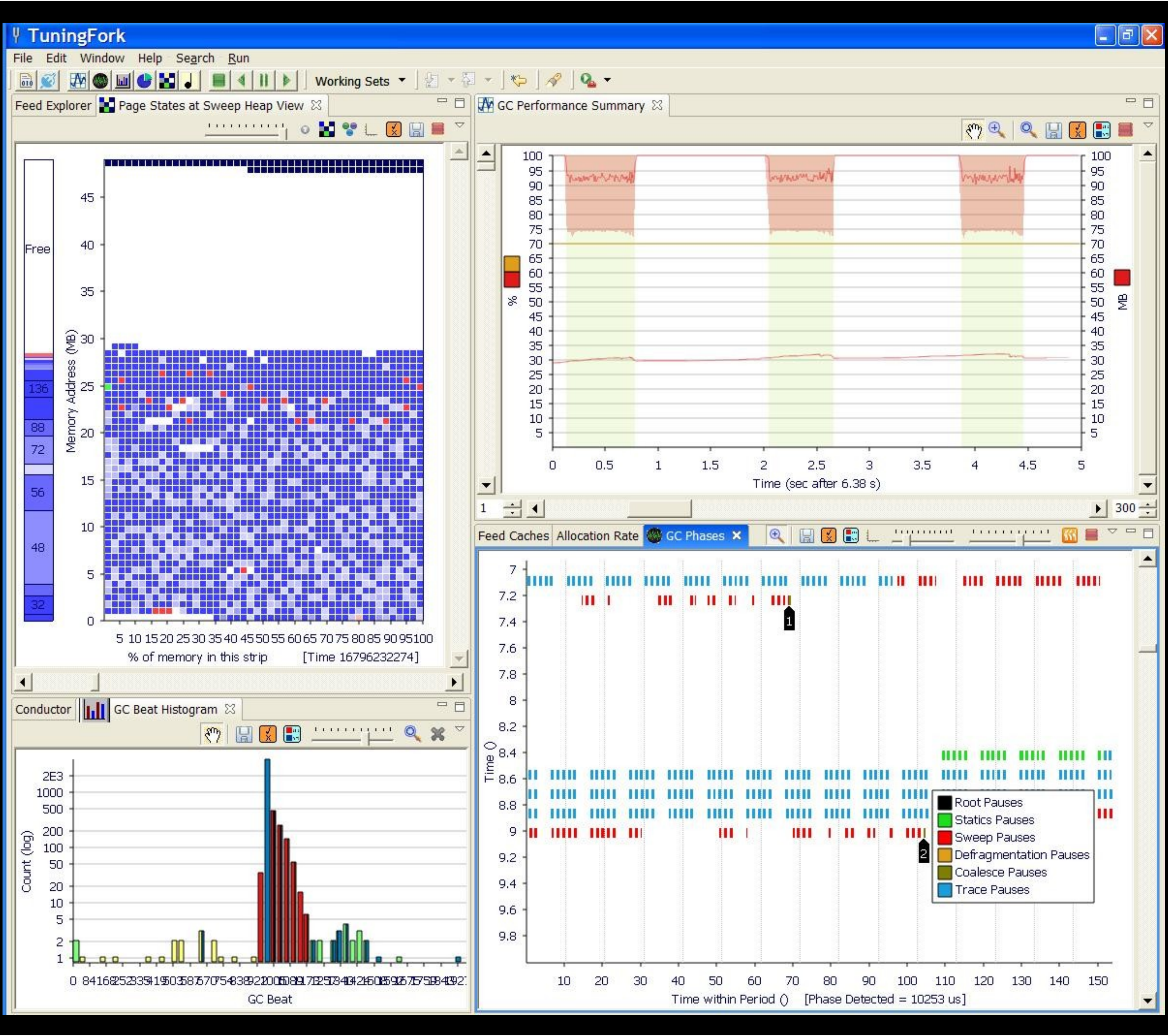
- How and when did the failure happen?
- Which component was responsible for the delay?

The Tuning Fork Project is investigating the production, consumption and visualization of high volume trace data

- JVM and Linux kernel instrumentation
- API for application level events
- Visualization and navigation of correlated event streams
- Available through alphaworks today

# *Tuning Fork Architecture*




TuningFork
File Edit Window Help Search Run
Working Sets
Feed Explorer
Page States at Sweep Heap View
Free
136
88
72
56
48
32
Memory Address (MB)
45
40
35
30
25
20
15
10
5
0
5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
% of memory in this strip
[Time 16796232274]
GC Performance Summary
%
MB
100 95 90 85 80 75 70 65 60 55 50 45 40 35 30 25 20 15 10 5
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
Time (sec after 6.38 s)
1
300
Conductor
GC Beat Histogram
Count (log)
2E3
1000
500
200
100
50
20
10
5
2
1
GC Beat
Feed Caches
Allocation Rate
GC Phases
Time ()
7 7.2 7.4 7.6 7.8 8 8.2 8.4 8.6 8.8 9 9.2 9.4 9.6 9.8
1
2
■ Root Pauses
🟩 Statics Pauses
🟥 Sweep Pauses
🟧 Defragmentation Pauses
■ Coalesce Pauses
🟦 Trace Pauses
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150
Time within Period ()
[Phase Detected = 10253 us]

## *Real-Time Class Analysis*

Critical regions must conform to rigid real-time constraints

- Code must be pre-loaded and pre-initialized before use
- Performance critical code must be pre-compiled before use

Currently, these regions are verified by inspection and testing

- Time consuming, Error-prone, Hard to maintain

We developed automatic program analysis for this process

- All-paths static analysis of one or more code regions
  - Generates code to load, initialize and compile classes
  - Generates code to pre-compile these classes
- Available through alphaWorks today

## *Very Low Latency Events*

There are limits to the latency for garbage collection

- Some work needs to get done, in every pause
- The limit will be >100μs on current hardware
- Some threads need to run at higher priority than GC

The RTSJ solution (NHRT) has serious problems

- Both reads and writes can fail (“safe SEGFAULT”)
- Can be costly in performance (checking overhead)
- Presents problems for modularity (scoping rules)
- Architected storage leak (immortal memory)

## *Expedited Real-Time Threads*

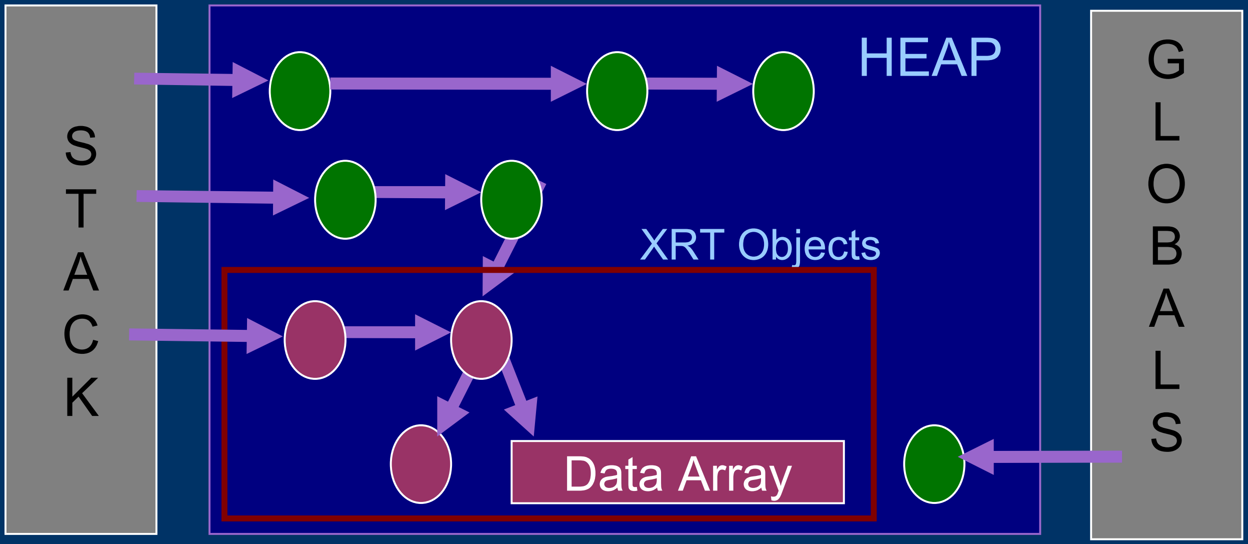
Some observations:

- The higher the frequency, the simpler the task
- Often do buffer processing – just move data

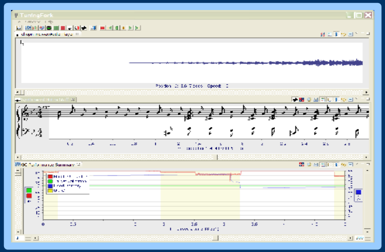
Expedited Real-Time Threads (XRT):

- Data structures must be allocated in advance
  - Allocation done in the heap beside other data structures
  - Usually includes some buffers
  - XRT region definition verifies and locks down XRT objects

# *Expedited Real-Time Threads*



# Application test beds



100% Java MIDI Synthesizer

1 ms (1KHz) timing for MIDI control (GC)

44.1KHz for waveform synthesis (Eventrons)

*Joint work with Bohm Software*

http://domino.research.ibm.com/comm/research_projects.nsf/pages/metronome.harmonicon.html

Air Java:  Collaborating UAV Swarms

Real-time but highly dynamic

Software engineering complexity limiting innovation

Focus on productivity + reliability and recovery

*Joint work with UC Berkeley*

Autonomous Quad Rotor Helicopter

100% Java, 3ms control loop

Goal is to validate Java in a critical physical control system

*Joint work with University of Salzburg*

# ***The road ahead***

Standards

- JSR 282 (RTSJ 1.1) and JSR 302 (Safety-critical Java)
- RTSJ profiles with alternate memory managers

Technology

- Most vendors have real-time GC
  - differentiation will be based on quality, performance
- Static Compilation and deterministic dynamic compilation
- Tooling for real-time model, assemble, deploy, analyze
- Real-time Java on a broader range of hardware and OS's

# Summing up: What makes WRT tick (and tock)?

J9 JVM technology

- IBM-authored virtual machine used in all IBM products and platforms
- Leadership performance, scalability and reliability

Optimizing compilation

- Static (aka ahead-of-time - AOT) compilation for predictable performance
- Dynamic (aka just-in-time - JIT) compilation for best performance (running on a low priority thread)

RTSJ

- Fully compliant to latest level
- Includes fixed priority scheduling, priority inheritance, asynchronous event handling, scoped and immortal memory management

Metronome

- Real-time garbage collection with 1ms worst case pause time

Linux

- RHEL MRG, SLERT
- Updated (open source) kernel and libraries engineered for real-time

# *Real-time Java Articles on developerWorks*

http://www-128.ibm.com/developerworks/views/java/libraryview.jsp?search_by=Real+time+Java+Part

- Real-time Java, Part 1: Using the Java language for real-time systems
- Real-time Java, Part 2: Comparing compilation techniques
- Real-time Java, Part 3: Threading and synchronization
- Real-time Java, Part 4: Real-time garbage collection
- Real-time Java, Part 5: Writing and deploying real-time Java applications
- Real-time Java, Part 6: Simplifying real-time Java development