Real-Time Java™ Technology: Why It Matters to You and What You Should Do About It

Greg Bollella, Distinguished Engineer

Dave Hofert, Group Marketing Manager

Sun Microsystems
http://java.sun.com/j2se/realtime/

TS-1904

Goal

Learn a bit about the Real-Time Specification for Java™ (JSR-01), how easy it is to convert a 'normal' Java-based program to a real-time Java-based program, and enjoy descriptions of three actual, industrial case studies of the use of real-time Java technology

Agenda

Why Real-Time Java Technology at All?

Brief Introduction to the Real-Time Specification for Java Technology

Real-Time Garbage Collection

Converting Java technology to Real-Time Java Technology

Case Studies

- 1: An autonomous ground vehicle
- 2: A real-time CORBA ORB
- 3: An autonomous aircraft

Summary

Why Real-Time Java?

- Predictability
  - “Real-Time” in this context does not mean “super-fast” —rather, it means “respond within a predictable time”
- Better design/architecture choices
  - Instead of a flat topology for requests, discriminate between more and less important events
  - Handle most important events first, allow others to complete when possible—all on one system
- Dealing with the Real-World
  - Stuff happens—RTJ helps you deal with it

Where Is Real-Time Java Technology Used?

Where You Might—and Might Not—Expect it

- Military
  - It's handy to know when there's a missile inbound—even if you are garbage collecting
- Telecommunications infrastructure
  - Excellent call handling, but occasionally a line is broken and must be addressed within a narrow time slice
- Banking
  - In some situations the value of a position can change by $1M/second…
- Industrial automation, automotive, etc.

Source: Customer and Analyst conversations

Real-Time Specification for Java Technology Overview

- The Real-Time Specification for Java, JSR-001
  - The only real-time Java technology
  - Not a silver bullet—but a sharper tool
  - Higher level RT abstractions, portable
- 100% Java technology
- Started in 1998 and developed by a team of experts from these communities
  - RT-sched, embedded systems design, Ada design, Java-based design, embedded processor design, real-time systems design, academia, RTOS design, etc.

Source: Customer and Analyst conversations

RT Application Development

Requires an API set and semantics which allow developers to **correctly reason about and control the temporal behavior of applications**

Real-Time Specification for Java technology and Sun’s implementation provide:

- An API set, semantic Java VM enhancements, and Java VM-to-OS layer modifications which allow developers of Java-based application to correctly reason about and control the temporal behavior of Java-based applications

Core Requirement Definitions

How Does the Application Need to Respond to Events?

- Real-time requirement
  - An application requirement which includes temporal correctness conditions (TCC)
- Hard real-time requirement
  - Requirements state that the TCC always be met
- Soft real-time requirement
  - Requirements state that the TCC can be missed in well-defined ways
  - Not just “well, whatever”
- Non real-time requirement—No TCC

Source: Customer and Analyst conversations

Real-Time Specification for Java Technology Features

- New system model
- Scheduling Abstractions
- Feasibility as a first-class abstraction
- Separation of hard real-time and everything else
- Priority inversion control
- New memory management abstractions
- New asynchronous event handling
- New, correct, asynchronous transfer of control
- Physical memory access methods

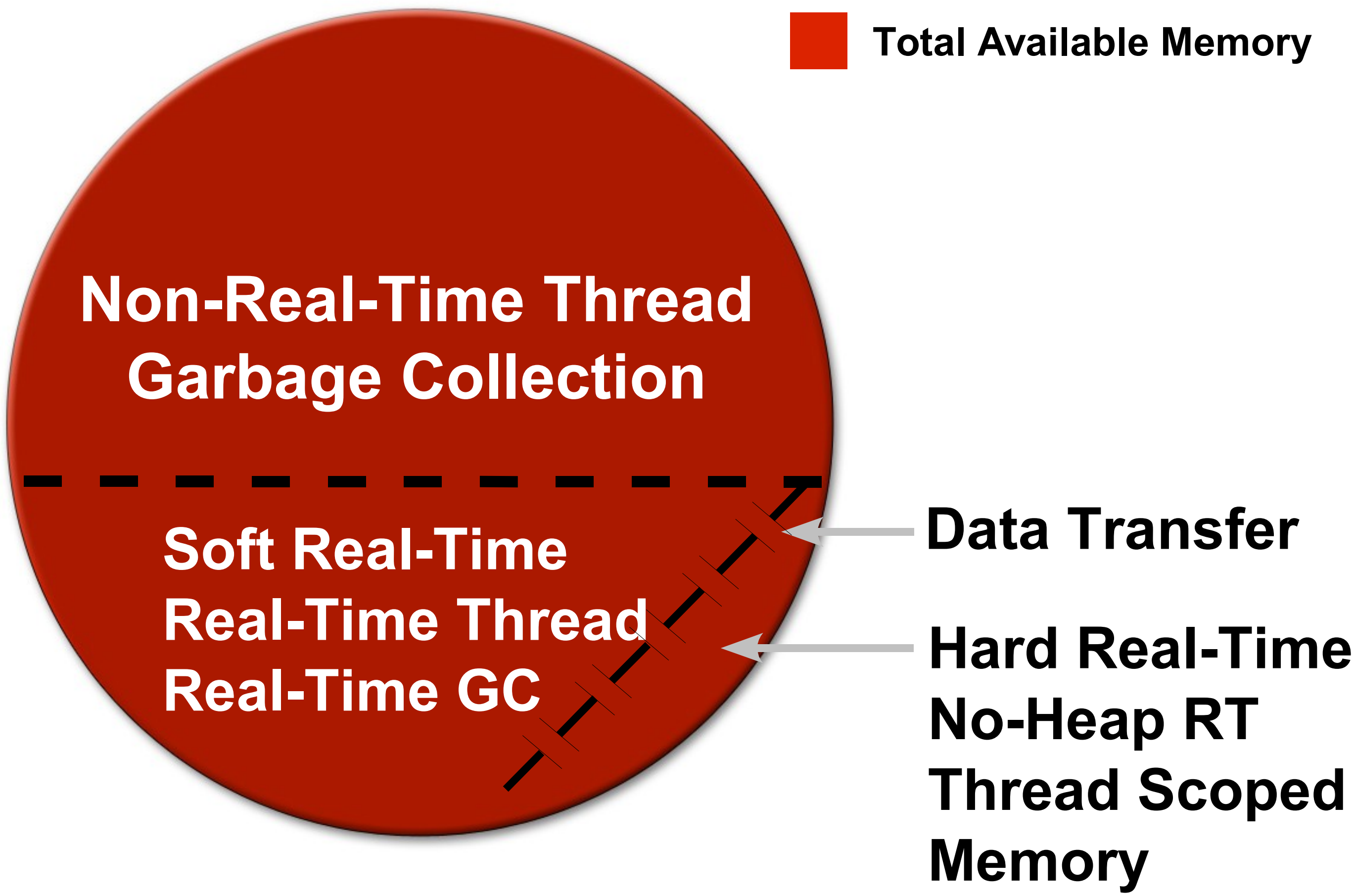
Source: Customer and Analyst conversations

Key Schedulable Classes

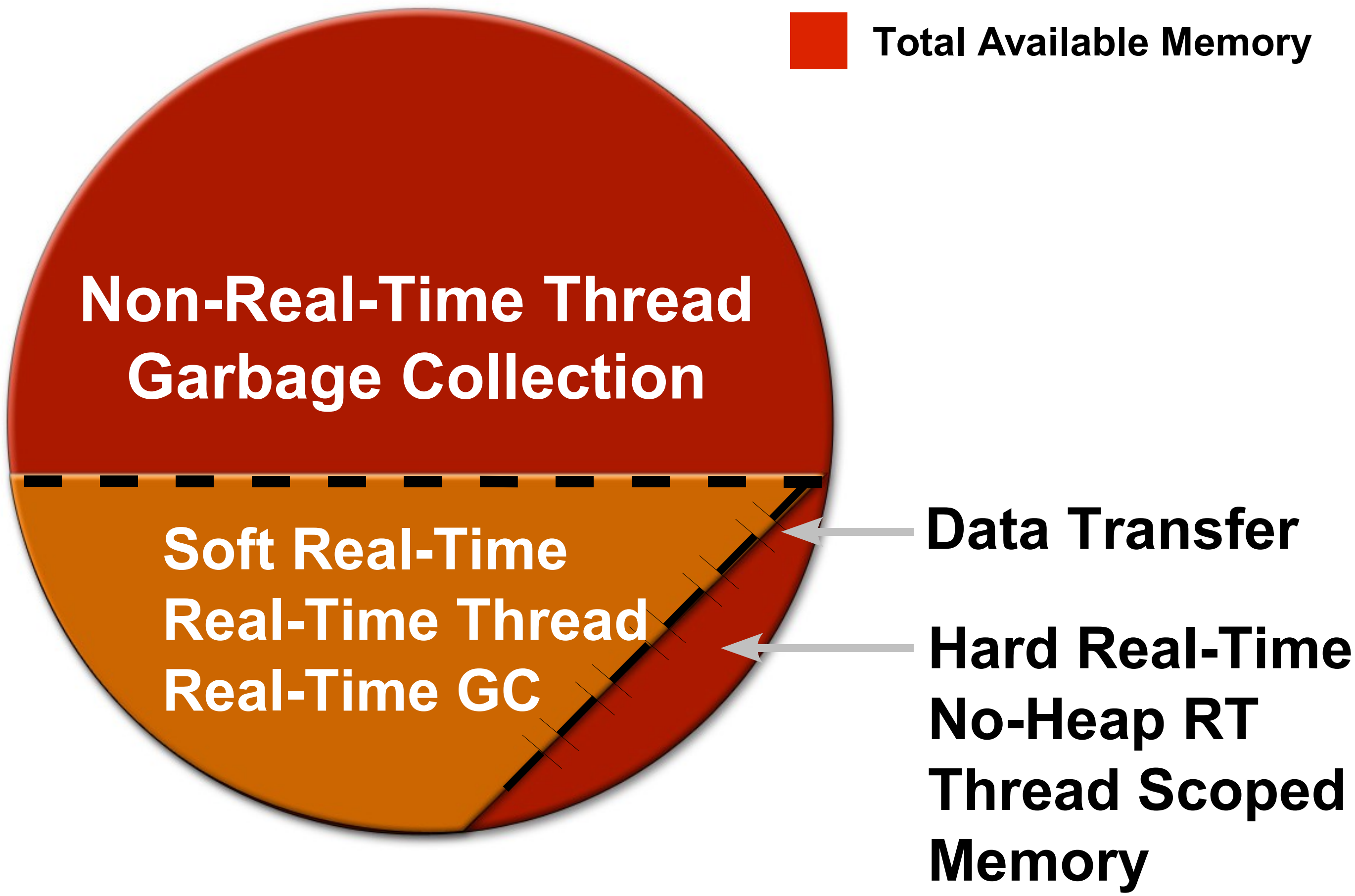
```
javax.realtime.RealtimeThread
// for 'soft' real-time
// can use the Heap, Immortal, and ScopedMemory
// program just like a regular Java thread
// all libraries available
// depends on the use of the real-time GC

javax.realtime.NoHeapRealtimeThread
// for 'hard' real-time
// can use ONLY Immortal, and ScopedMemory
// need care when using library methods
// necessary for only very small, well-defined logic
// application managed real-time garbage collection
```

Real-Time Specification for Java Technology System Model



Real-Time Specification for Java Technology System Model



Real-Time Specification for Java Technology
Solutions for Real-Time Java Technology

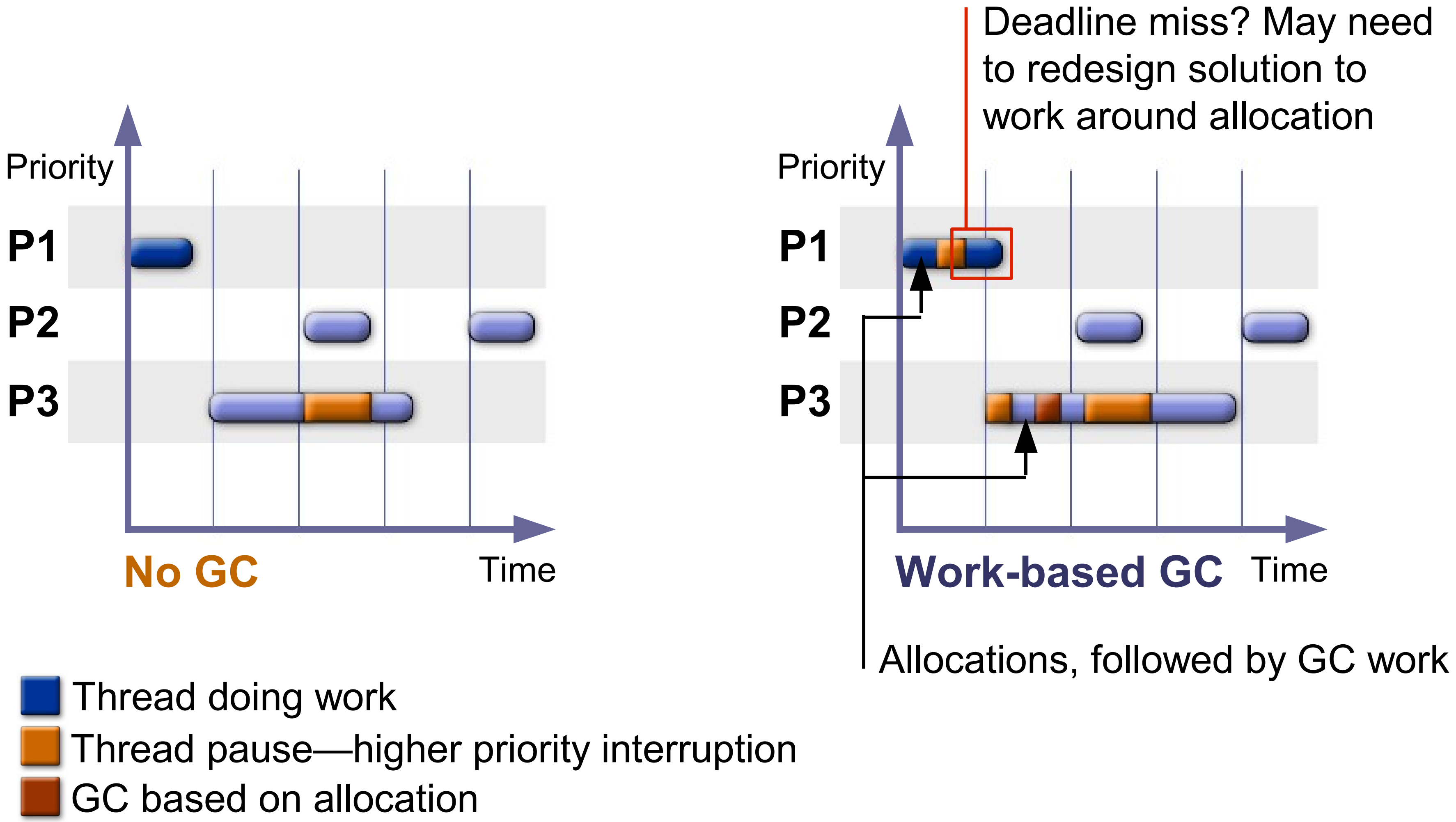
- Two main approaches
  - Use scoped memory and NHRT/RT threads
  - Use real-time garbage collector
- Real-Time GC
  - Essentially a “garbage collector with knobs”
  - Core concept: memory allocation and collection rates are assessed, and the minimum set of GC occurs every period
  - Assuming enough GC, large interrupts are avoided at the cost of small, regular, and predictable collections

RTGC Designs: Work-Based GC

Each Thread Pays “Cost” of GC at Allocation Time

- Benefits
  - Predictable “payment”, but “cost” may vary depending on memory allocations of non-RT threads
  - Only mutators/threads that allocate memory pay cost
  - Easier to implement VM
- Costs
  - Must budget enough room for possible GC or miss deadlines
- Example: Aicas—Jamaica VM

Work-Based GC

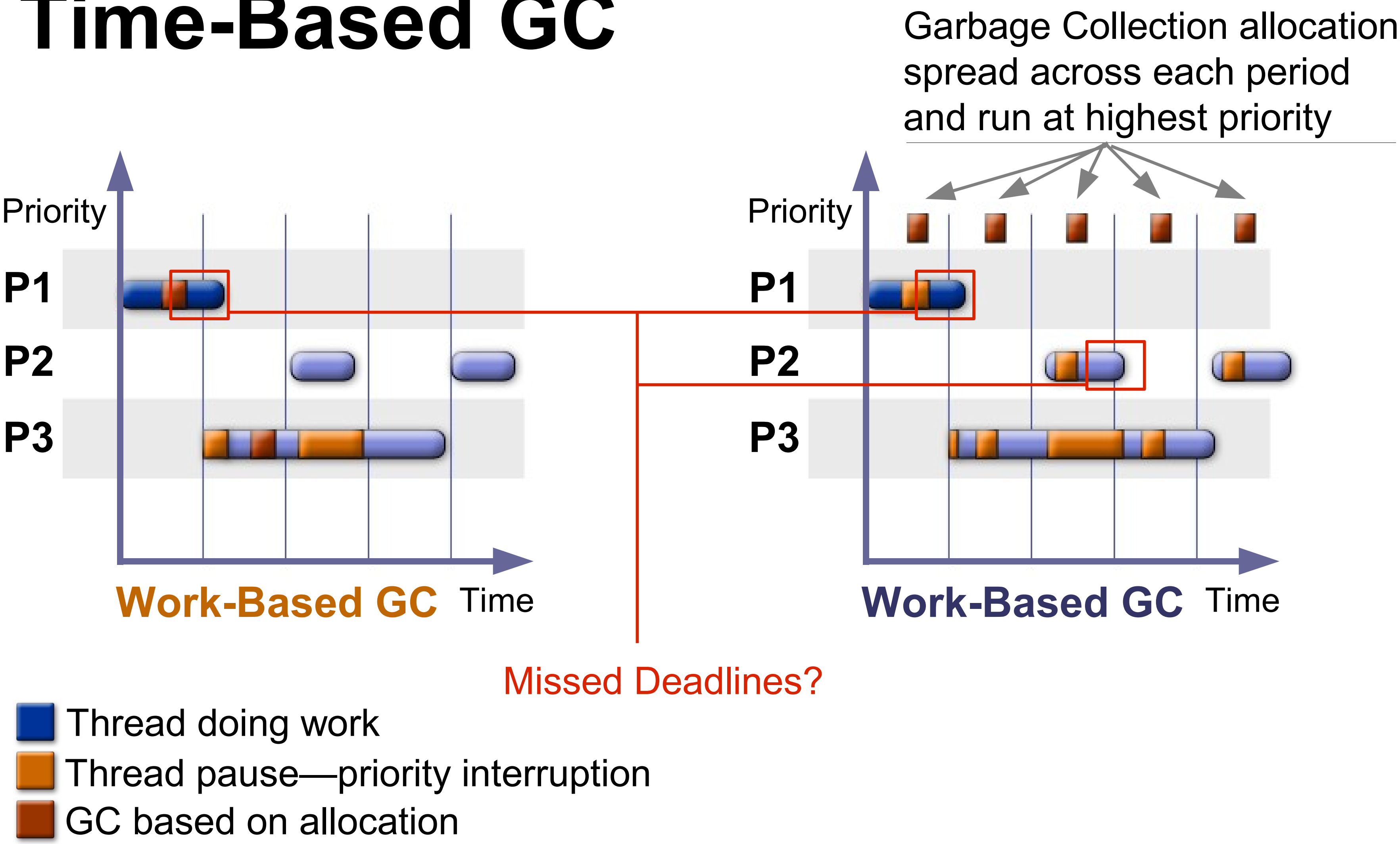


RTGC Designs: Time-Based GC

Allocate Cost of GC Across All Periods; GC Is Top Priority

- Benefits
  - Deterministic GC for each period
  - Spreads cost of GC across all periods—thus avoiding any one large GC interrupt
- Costs
  - Cost of GC in all periods
  - All threads are impacted (because GC still runs at highest priority)
- Example: IBM—Metronome

Time-Based GC

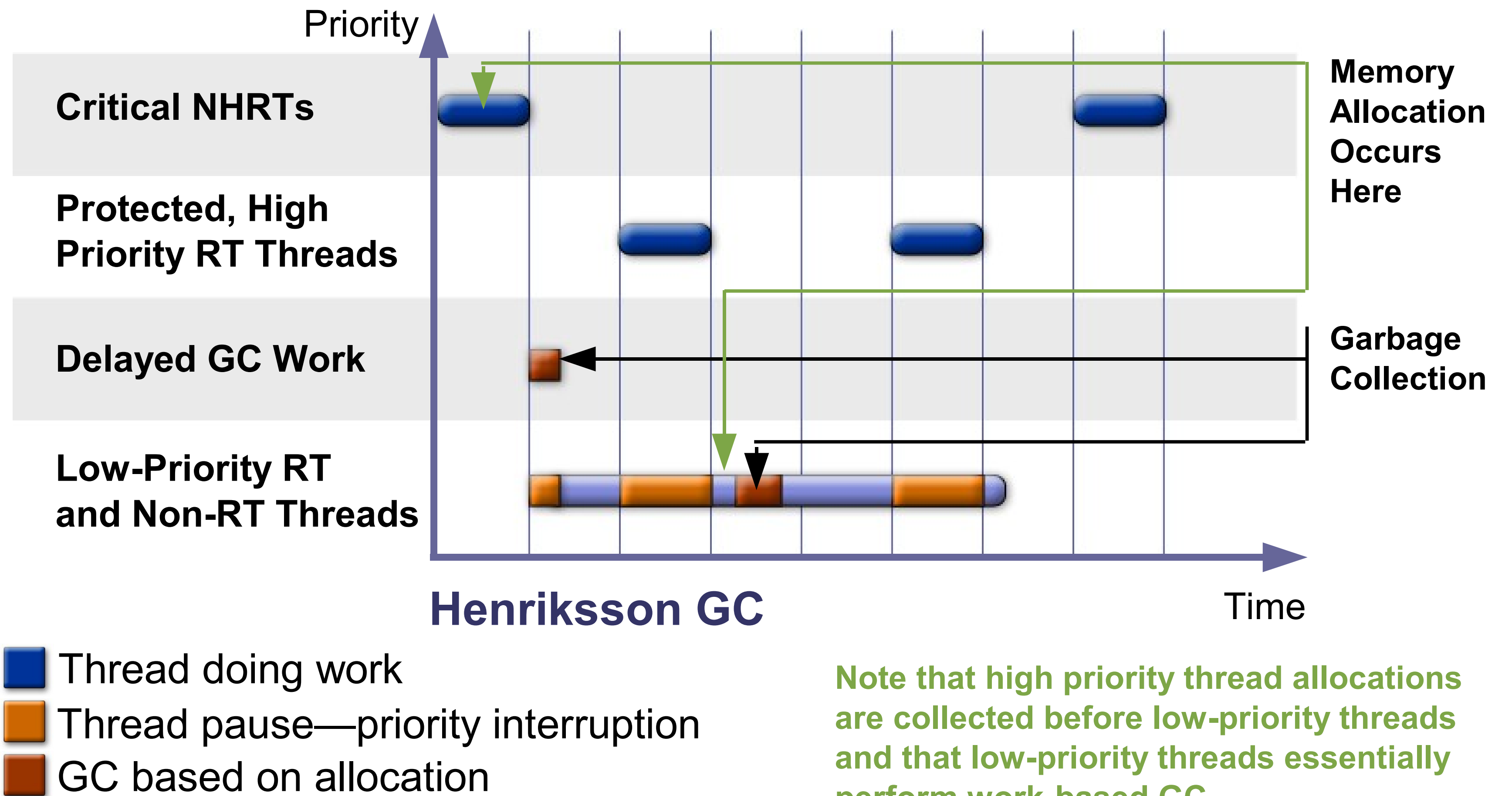


RTGC Designs: Henriksson’s GC

Don’t Interrupt Highest Priority Threads; GC Cost Paid Elsewhere

- Benefits
  - Higher priority threads unaffected by GC
  - Thus they are more deterministic and can have lower latencies
- Costs
  - No silver bullet: GC cost remains same and must fit into overall schedule (e.g., Lower priority threads have to run some time)
  - Lower-priority and non-RT threads carry GC load

Henriksson GC



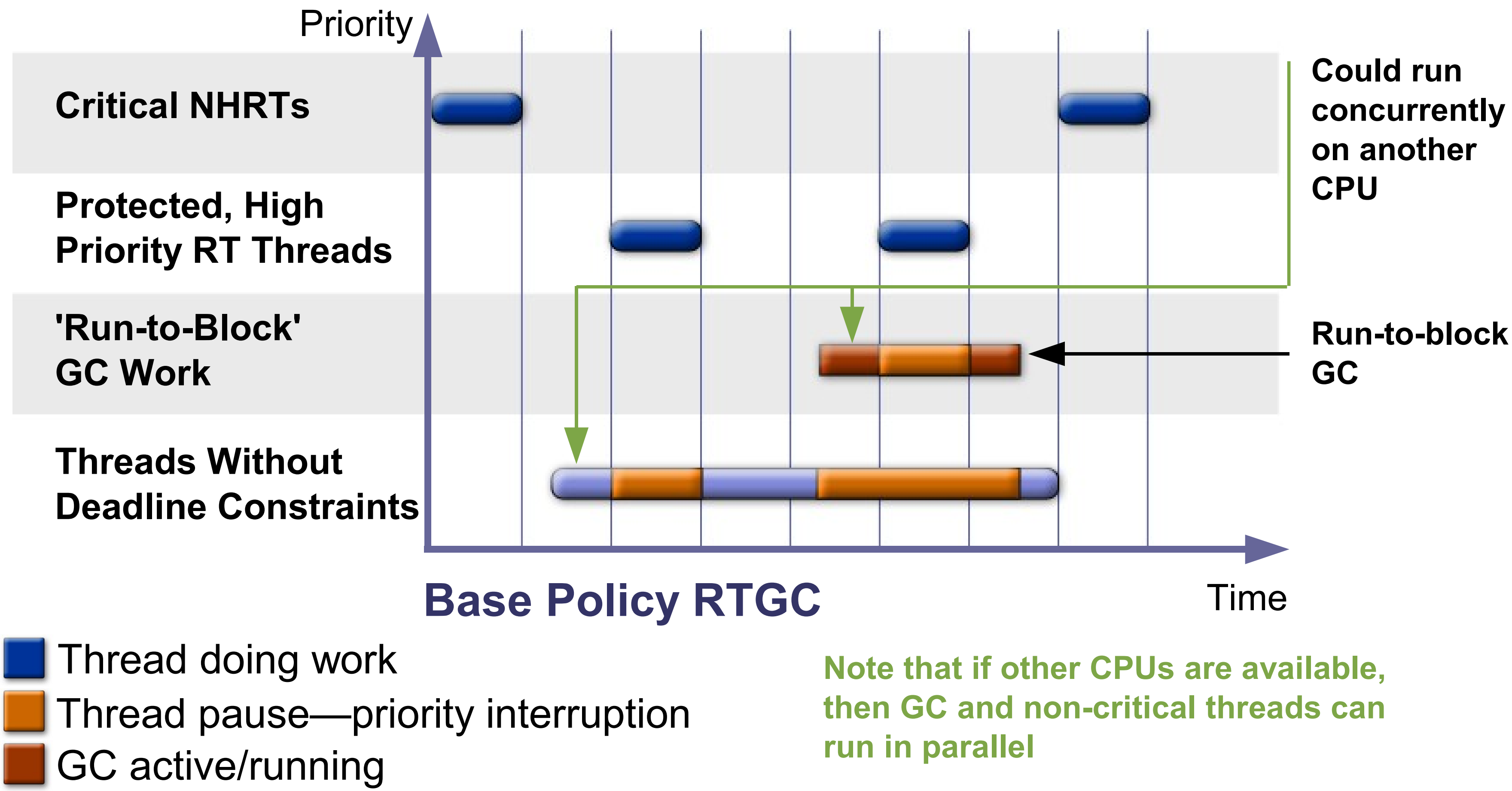
Note that high priority thread allocations are collected before low-priority threads and that low-priority threads essentially perform work-based GC

RTGC Designs: Sun's RTGC

Apply Concept of Policy to Henriksson's Approach

- Goal: Smallest latencies for high priority GC'd threads
- Defer GC work thanks to Henriksson's approach
  - Minimize mutator overhead (read/write barriers)
- Advantages
  - Scalable (no issues with multi-processor support)
  - Flexible (Henriksson's approach works with different policies for low priority RT threads)
    - GC overhead can be paid by these threads if total memory consumption goes up
    - More efficient policies possible (could pause certain threads)
    - Simpler policies like running the GC on a dedicated CPU

Base Policy, Sun RTGC



Note that if other CPUs are available, then GC and non-critical threads can run in parallel

Real-Time GC Summary

Comparison Point	Work-Based	Time-Based	Sun RT 2.0
NHRT support	? (could be done)	? (1)	Yes
Non-allocating High Priority Thread overhead	None	Pre-empted (2)	None
Allocating High Priority Thread overhead	Overhead (2)	Pre-empted (2)	None
All other threads	Same as High Priority	Same as High Priority	Flexible policy
Multi-processor support	OK ? (Should be fully concurrent)	??? (1)	OK (Fully concurrent)

(1) : Likely requires all threads to be suspended during each small GC work... on all CPUs
(2) : overhead/preemption time depends on the allocation behavior of non RT threads

Myth Busting

Don't Believe Everything You Hear!

- **Myth**: Programming in RTSJ is hard and/or weird
- **Truth**: Getting started in RTSJ is easy!

- **Myth**: Most libraries don't work in a NoHeap context
- **Truth**: Wrong!! Most libraries work fine in a NoHeap context

---

- **Myth**: LowLatency requires ScopedMemory
- **Truth**: Sun's Real-Time GC can get down to about 300 μseconds latency

Converting Java Code to Real-Time Java Code

- Essentially just a syntax change to start
  - With this you'll get lots of predictability
  - Real-time garbage collection
  - 28+ priorities which actually work (60 in Java technology RTS)
  - Priority inheritance protocol
  - Lots of internal Java VM changes to enhance predictability
  - Initialization-time compilation
- Can also use scoped memory for more precise control of memory usage

Step One

Replace:

```
Thread T = new java.lang.Thread();
```

with

```
RealtimeThread RT = new javax.realtime.RealtimeThread();
```

and you get:

- Industrial-strength, real-time garbage collection
- 28+ priorities that actually work, and work precisely
- Priority inheritance protocol
- And everything else, AEH, physical memory, etc.

Step Two

- Ha—there is no step two…

the POWER of JAVA™

Case Study: A Robotic Vehicle Named Tommy

Paul J. Perrone

CEO
Perrone Robotics, Inc.
www.perronerobotics.com

ORACLE

Robotics Applications

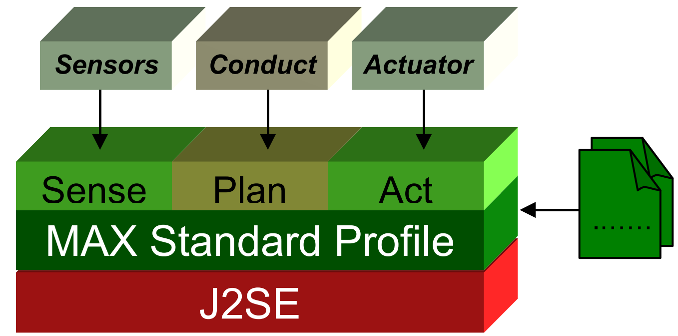
- Sense-Plan-Act
  - Acquire data from sensors
  - Formulate some plan of action
  - Actuate motors for movements
- Timeliness is good for robots
  - Feedback control of motors
  - PWM of a motor
  - Counting time between events
- Tardiness is bad for robots
  - Sloppy control
  - Inefficient control
  - Loss of control

Tommy

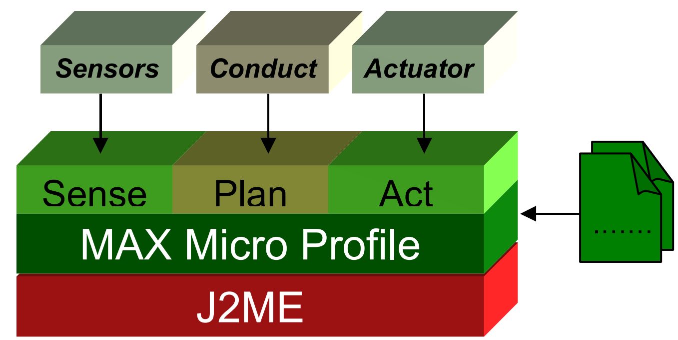

PERRONE
Perrone Robotics
Assured Technologies
TOMMY
Team Jefferson
Java
Sun Microsystems
06
FairIsaac
csi wireless
OmniSTAR
BODINE
ELECTRIC
COMPANY
Sandtec
Charlottesville
AUTOMOTIVE
REFINISH
SUPPLY

Tommy's Software Architecture

- MAX Standard Profile
  - J2SE™ technology-based
  - All main processing/decisions
  - GPS/INS/Laser/Radar sensing
  - Navigation and obstacle avoidance
  - Actuate commands to MAX micro

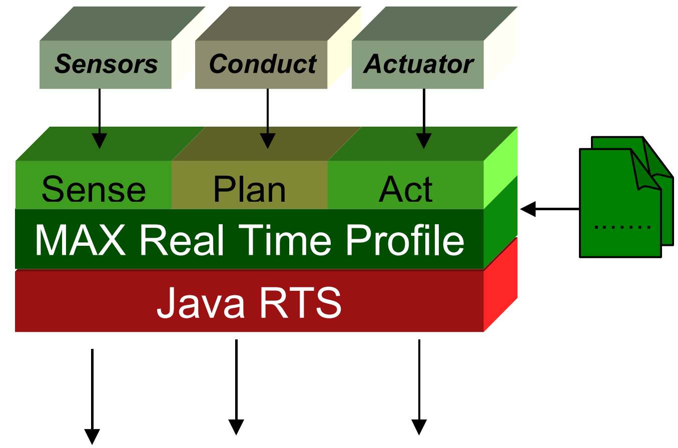


- MAX Micro Profile
  - J2ME technology-based
  - Low-level feedback controls
  - Commands received
  - Feedback and vehicle state sensing
  - Actuate steering/throttle/brake/shift



Java Technology RTS to the Rescue

- Pros of current approach
  - Can do lots in standard J2SE/J2ME technology
  - Very fast micro latencies (1 mSec)
  - Faster latencies with hardware controls (<< 1 mSec)
- Cons of current approach
  - Excise real-time behavior to J2ME technology
  - Much care to not generate garbage
  - Need more rigor for industrial-grade
- Java technology RTS Advantage
  - Alleviates pains of current approach
  - Provides faster industrial grade path for the most time critical operations



Case Study: PrismTech RTOrb: Real-Time Java Technology-Based ORB

David Atkinson

Product Marketing
PrismTech
www.prismtech.com

Why Real-Time CORBA?

- CORBA is well-established as a technology for integrating diverse systems
  - Used extensively for mission and business critical applications in areas such as defense, telecommunications, and manufacturing
- OMG's Real-Time CORBA Specification extends the benefits of CORBA to the Real-Time domain
  - RT CORBA addresses end-to-end predictability across CORBA systems and provides a solution in terms of priority control, synchronization, and resource control

RT CORBA in the Field

- Large scale defense integration
  - C4i—wide range of Command Control Computers Communication and Intelligence Systems
- Telecommunications and networking
  - Business management applications
  - Operations support systems/call control
  - Intelligent networking
  - STN/Internet convergence
- Aerospace
  - Air traffic management
- Manufacturing—Controllers/Robotics

Key Benefits of RTOrb on Java Technology RTS

- RT system developers can use Java technology, CORBA
  - Write once run anywhere portability, ease of use and security, enterprise scalability, full CORBA functionality
- Excellent performance (latency and throughput)
  - Low jitter (< 1ms), performance better than other Java ORBs
- Use as RT ORB, general Enterprise ORB, or both
  - Single ORB solution for systems with a mix of uses (both RT and non-RT)
  - Single ORB solution minimizes ORB interoperability issues and requires less training/support for developers

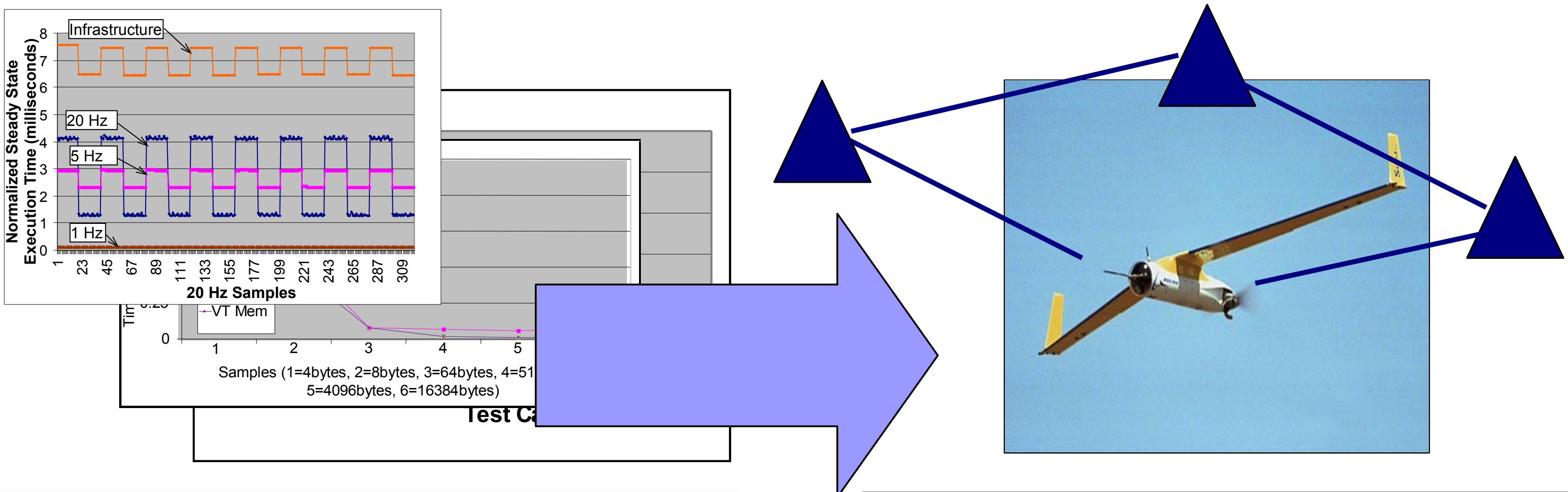
the POWER of JAVA™

Case Study: Mission Control for an Unmanned Autonomous Aircraft

Edward Pla

Real-Time Java Researcher
Boeing Phantom Works
www.boeing.com

RTSJ Demonstration



Benchmarking, lab demonstrations, and mission qualification testing performed to validate Real-Time Java technologies

Demonstrated first flight of RTSJ using ScanEagle aircraft performing real-time autonomous auto-routing

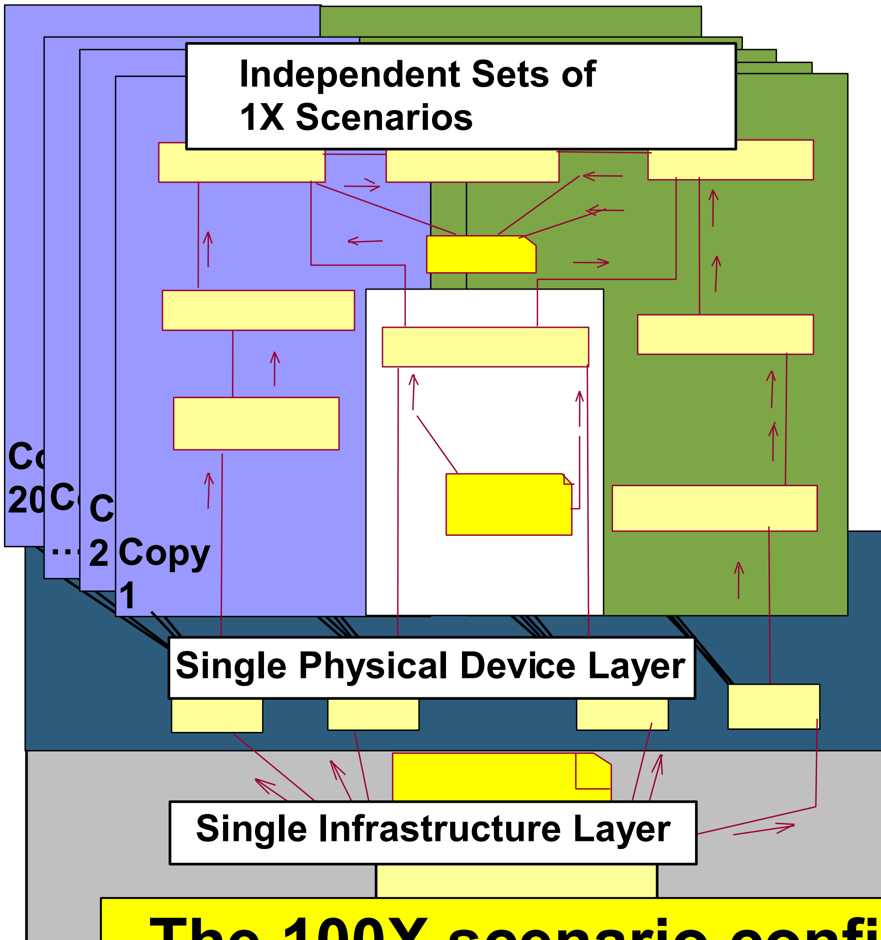
Learn details on the RTSJ experimentation configuration that led to first flight of the RTSJ on a ScanEagle UAV

RTSJ Low-Level Benchmarks Summary

- Thread throughput (< 1% jitter)
  - (NHRT, RT, Java Thread) with/without contending threads
- Determinism (< 0.1 % jitter)
  - Periodic start of frame
  - Periodic event determinism
- Latency
  - Context switch latency (5 us)
  - Priority inheritance latency (5 us)
  - Synchronization latency (30 us)
  - Event latency (2 us)
- Memory management
  - Allocation size (1 byte to 16k bytes) vs. memory type (heap, immortal, linear memory, variable memory) (2 us/byte)
  - Throughput (floating point, logarithmic, No Op) vs. memory type (< 5% jitter)
  - Memory area entry/exit criteria (20 us)

Acceptable performance in key areas

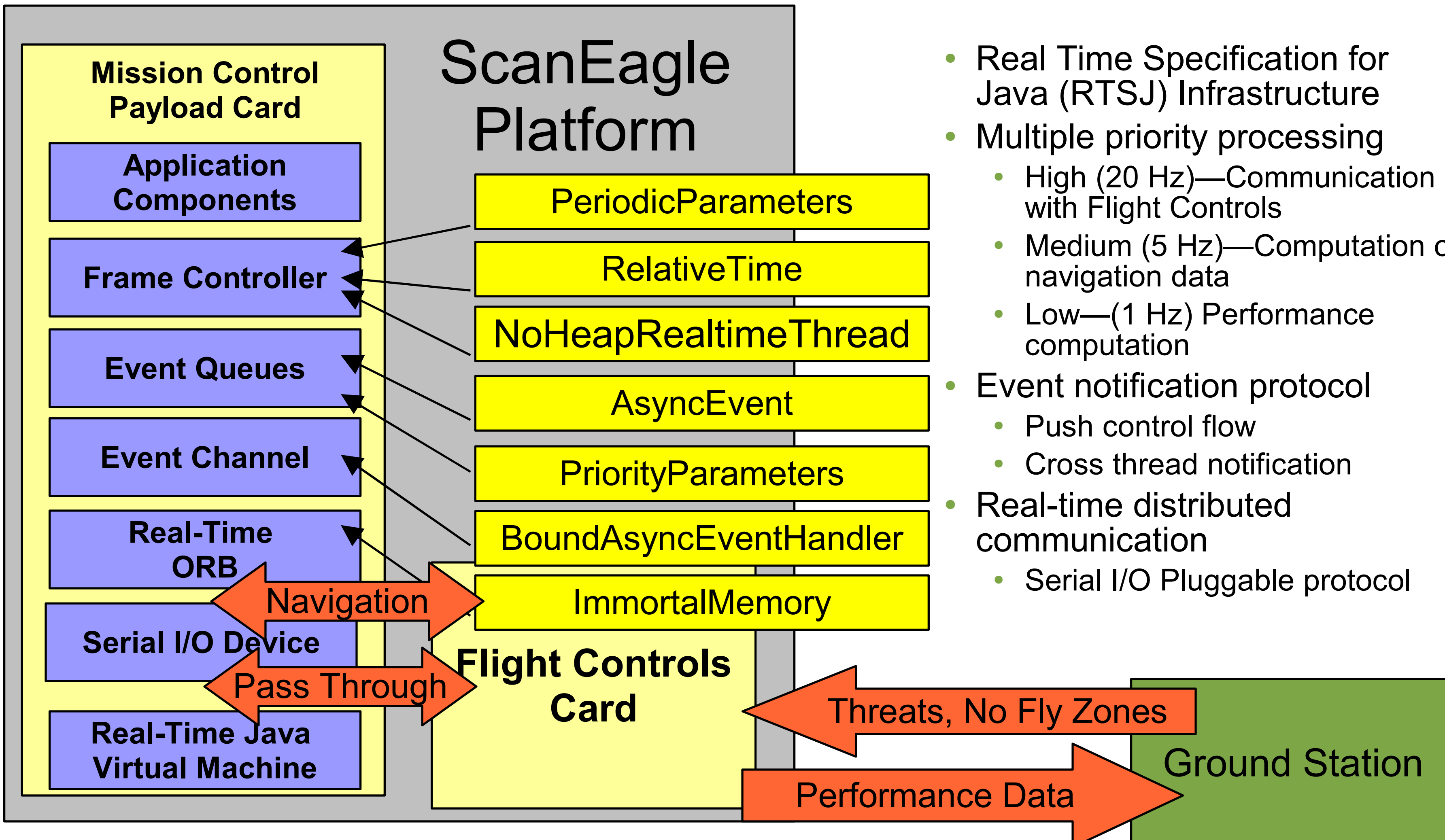
Application-Level Benchmarks



Scenario	Component Types	Component Instances
1X	**7**	**12**
20X	**14**	**164**
50X	**35**	**404**
100X	***70***	***804***
200X	**140**	**1604**

The 100X scenario configured to size of Boeing domain-specific single processor platform

RTSJ Flight Experimentation Architecture



- Real Time Specification for Java (RTSJ) Infrastructure
- Multiple priority processing
  - High (20 Hz)—Communication with Flight Controls
  - Medium (5 Hz)—Computation of navigation data
  - Low—(1 Hz) Performance computation
- Event notification protocol
  - Push control flow
  - Cross thread notification
- Real-time distributed communication
  - Serial I/O Pluggable protocol

Summary

- JSR-01, the Real-Time Specification for Java technology gives developers the ability to correctly reason about and control the temporal behavior of logic
- Getting started using RTS Java technology is really, really, simple
- Implementations of the RTS Java technology are available now
- The RTS Java technology is rich and offers a wide range of APIs and semantics to help developers write code which behaves “well” with respect to time
- The RTS Java technology is the correct way to do real-time in Java technology

For More Information

Web Resources:

- Sun's Real-Time Java technology site: http://java.sun.com/j2se/realtime/
- RTSJ Specification: http://www.jcp.org/en/jsr/detail?id=1

Books:

- Real-Time Java Platform Programming, Dibble: http://www.sun.com/books/catalog/dibble.xml
- Concurrent and Real-Time Programming in Java, Wellings: http://www.amazon.com

Q&A

Greg Bollella

Dave Hofert

Paul Perrone

David Atkinson

Edward Pla

Real-Time Java™ Technology: Why It Matters to You and What You Should Do About It

Greg Bollella, Distinguished Engineer
Dave Hofert, Group Marketing Manager

Sun Microsystems
http://java.sun.com/j2se/realtime/

TS-1904