# *Sun Java™ Real-Time System Revealed*

**Frederic Parain**
**David Holmes**

Java Real-Time System Engineering
Sun Microsystems, Inc.
java.sun.com

TS-1331

# Java Real-Time System (Java RTS) Revealed

For easy, yet efficient, real-time programming

Understand how Java RTS and the Solaris™ Operating System (Solaris OS) interact to make a powerful real-time platform.

# Agenda

- Introducing Java Real-Time System
- A Tour of Key Real-Time Features
  - Thread Scheduling
  - Asynchronous Event Handling
  - Periodic Execution
  - Monitoring

# Agenda

**Introducing Java Real-Time System**

A Tour of Key Real-Time Features

- Thread Scheduling
- Asynchronous Event Handling
- Periodic Execution
- Monitoring

# What Is Real-Time?

- What does it mean, actually?
  - It does not mean “super-fast”
  - It means “respond within a **predictable** time”
- It’s all about temporal correctness
  - The **time** at which a result is produced is as important as its logical correctness

# Why Real-Time for Java Code?

- Same answers as to “Why Java code?”
  - Easier and safer than C/C++
- Real-time software loads are evolving
  - Increase both in size and complexity
  - Traditional, low-level programming no longer provides the required level of abstraction
- Point to the need for a common, high-level, correct, advanced, real-time Java application development platform

# Where Could You Use Real-Time Java Technology?

- Military
  - It’s handy to know when there’s a missile inbound—even if you’re garbage collecting
- Telecommunication infrastructure
  - VoIP, PBX, IMS, new 3G services
- Banking
  - Meet customer QoS and regulatory requirements for pricing/trading
- Industry
  - Factory automation, process control

# The Real-Time Specification for Java (RTSJ)

- The RTSJ, JSR 001
  - The standard that defines how real-time behavior must occur within Java technology
  - Therefore, the only real-time Java technology!
- APIs and semantic enhancements which allow Java code developers **to correctly reason about and control the temporal behavior of applications**
  - Better, high-level, portable abstractions
  - 100% Java technology

# Java Real-Time System

- Sun’s implementation of the RTSJ
  - 100% compliant with Java technology and RTSJ
- Java RTS 2.0 highlights
  - Based on Java Platform, Standard Edition (Java SE platform) 5
  - Runs on Solaris OS, SPARC® technology, and x86/x64 platforms
  - Relies on Solaris platform built-in real-time capabilities
- Innovative Real-Time Garbage Collector
  - See session TS-2901

# Java RTS 2.0 Platform

- From embedded single-board computers
- To carrier-grade blade servers
- To enterprise servers

# Agenda

Introducing Java Real-Time System

**A Tour of Key Real-Time Features**

- Thread Scheduling
- Asynchronous Event Handling
- Periodic Execution
- Monitoring

# Java RTS Tour

- We will primarily focus on application execution
    - And review the features that support common real-time practices
    - Scheduling, asynchronous event handling, periodic execution, application monitoring
- The RTSJ and Java Real-Time System go way beyond that
    - RTSJ enhanced memory model
    - Java RTS real-time garbage collection (RTGC)

# Agenda

Introducing Java Real-Time System

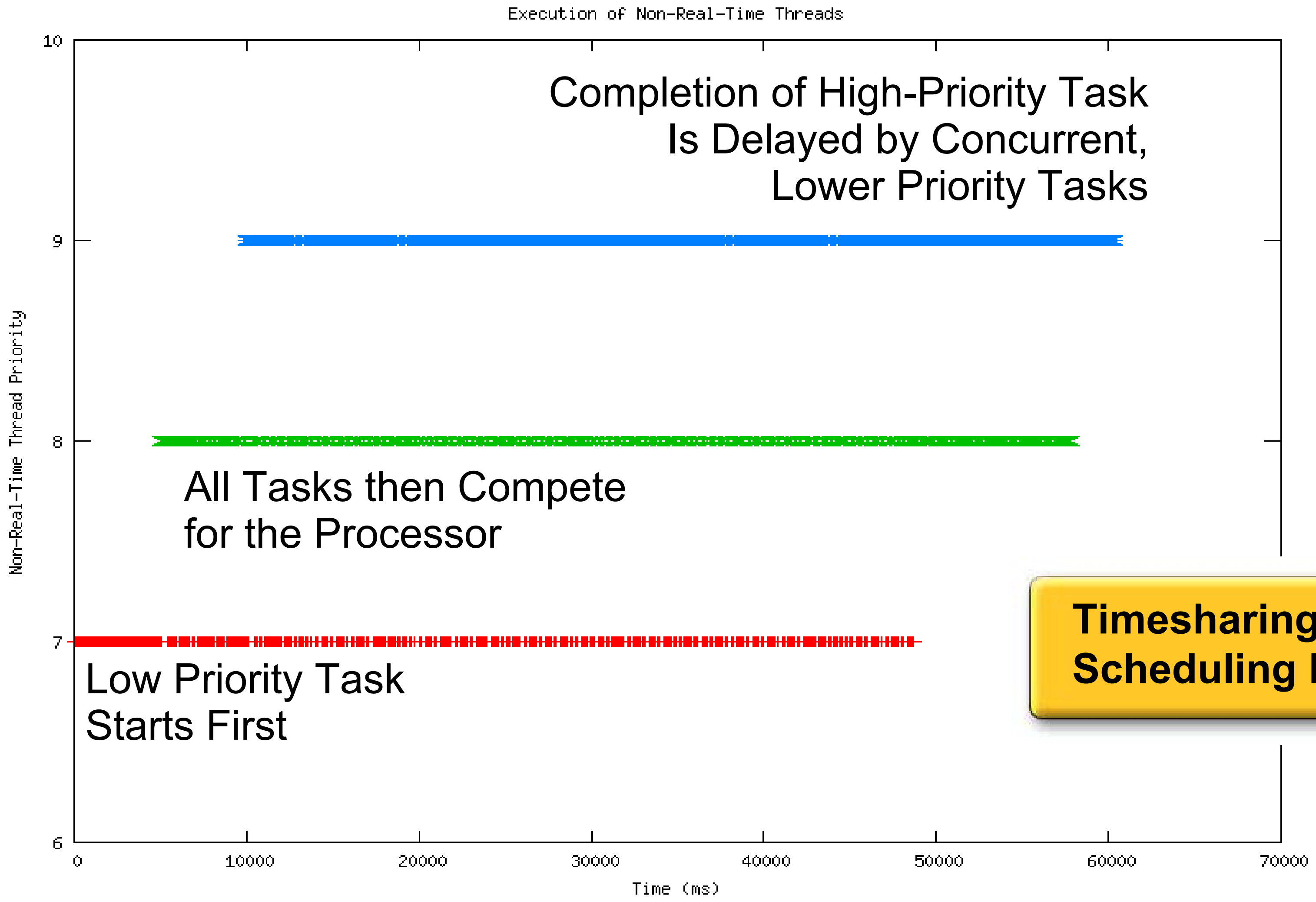
A Tour of Key Real-Time Features

- **Thread Scheduling**
- Asynchronous Event Handling
- Periodic Execution
- Monitoring

# Thread Scheduling

- Consider the following workload:
  - Task 1, priority low
  - Task 2, priority medium, starting 5 seconds later
  - Task 3, priority high, starting 5 seconds later
  - Each task requires 20 seconds of CPU time to complete
  - Only 1 processor
- Quiz
  - Which task completes first?
  - When does each task complete?

# Java SE Platform, Non-Real-Time Execution



Completion of High-Priority Task
Is Delayed by Concurrent,
Lower Priority Tasks

All Tasks then Compete
for the Processor

Low Priority Task
Starts First

**Timesharing Scheduling Policy**

# Observations

- Java platform thread priorities do not have precisely defined semantics
    - Just a “hint” given to the Java Virtual Machine (JVM™) and OS
- Timesharing scheduling policy does not strongly enforce priorities
    - Just aims at providing “a good response time to interactive processes and a good throughput to CPU-bound jobs”
- Can’t guess much about temporal behavior

The terms “Java Virtual Machine” and “JVM” mean a Virtual Machine for the Java™ platform.

# Programming in Java RTS, Step One

- Replace:

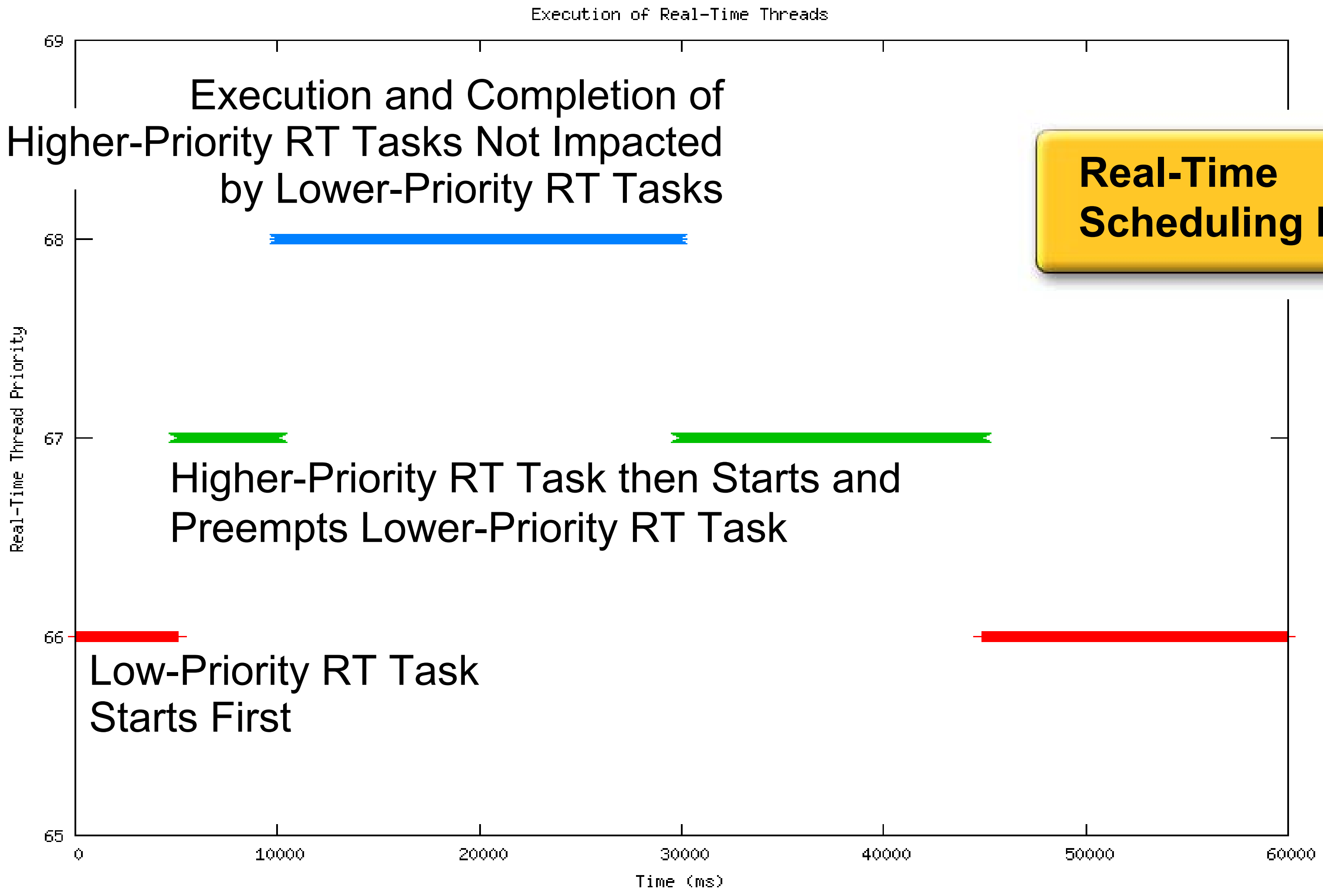
```
Thread T = new java.lang.Thread();
T.setPriority(prio);
```

- With:

```
RealtimeThread RTT =
   new javax.realtime.RealtimeThread();
RTT.setSchedulingParameters(prioParms);
```

- Then…

# Java RTS, Real-Time Execution



Execution and Completion of Higher-Priority RT Tasks Not Impacted by Lower-Priority RT Tasks

Higher-Priority RT Task then Starts and Preempts Lower-Priority RT Task

Low-Priority RT Task Starts First

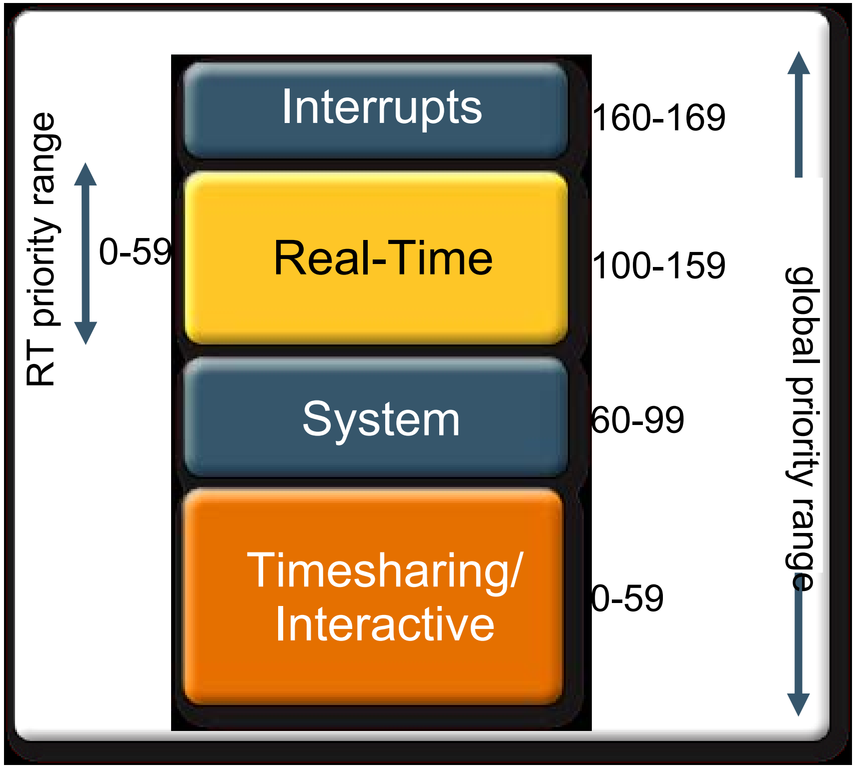
**Real-Time Scheduling Policy**

# Benefits

- Real-time Java platform priorities are strongly enforced
  - The RTSJ requires at least 28 real-time priority levels
- Scheduling policy explicitly specified by the RTSJ
  - Fixed priority, run-to-block, preemptive scheduler
  - Explicit rules on placement in the dispatch queue
- Enables the RT application designer to correctly reason about temporal behavior

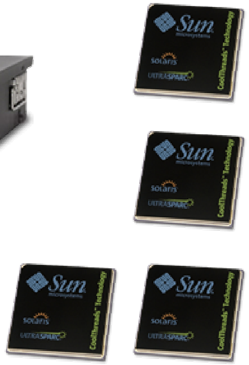
# Java RTS on Solaris OS

- Uses Solaris platform Real-Time (RT) scheduling class
  - 60 priority levels
  - Highest range of thread priorities in the system
  - Highly scalable on multi-processor platforms
- JVM implementation is locked into memory
  - No page swap in/swap out



# Solaris Platform Resource Partitioning

- Processors can be devoted to particular activities
  - Via processor sets, pools, or containers
  - Enable processors to be assigned to critical threads
  - Prevent unrelated activities to thrash processor caches
- Processors can be sheltered from h/w interrupts



1 x core for hard RT threads; set to *no-intr*

1 x core for soft RT threads

2 x cores for non-RT threads

# Agenda

Introducing Java Real-Time System

**A Tour of Key Real-Time Features**

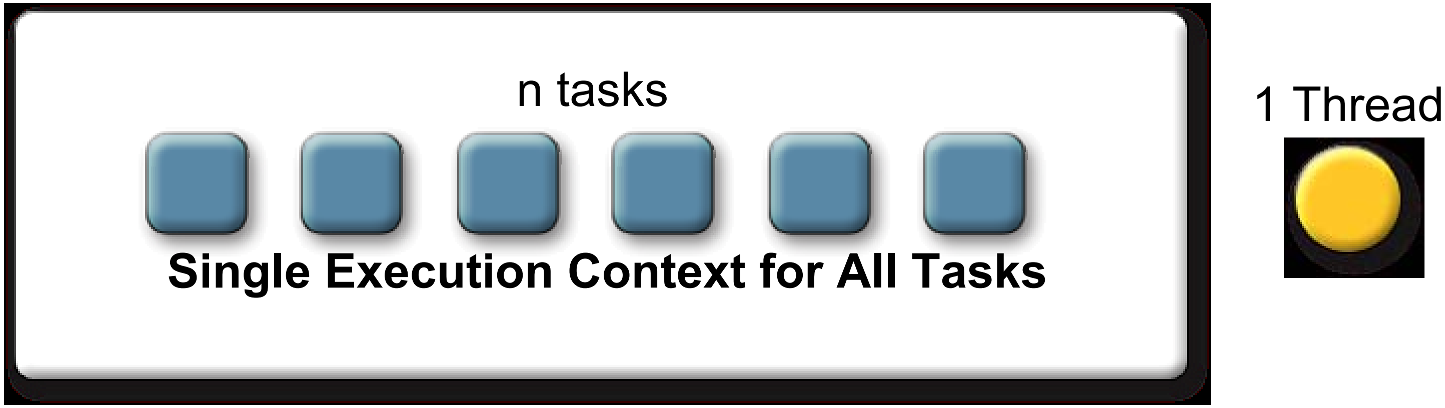
- Thread Scheduling
- **Asynchronous Event Handling**
- Periodic Execution
- Monitoring

# Handling of Asynchrony in the RTSJ

- Real-time systems interact with the outside, physical world
- Most physical systems have an asynchronous behavior
  - Time-triggered
  - Event-triggered
- RTSJ guiding principles for asynchronous events
  - Many handlers can be associated with the same event
  - Execution of the logic is scheduled, and dispatched by an explicit scheduler

# Asynchronous Execution in Java SE Platform

## java.util.Timer



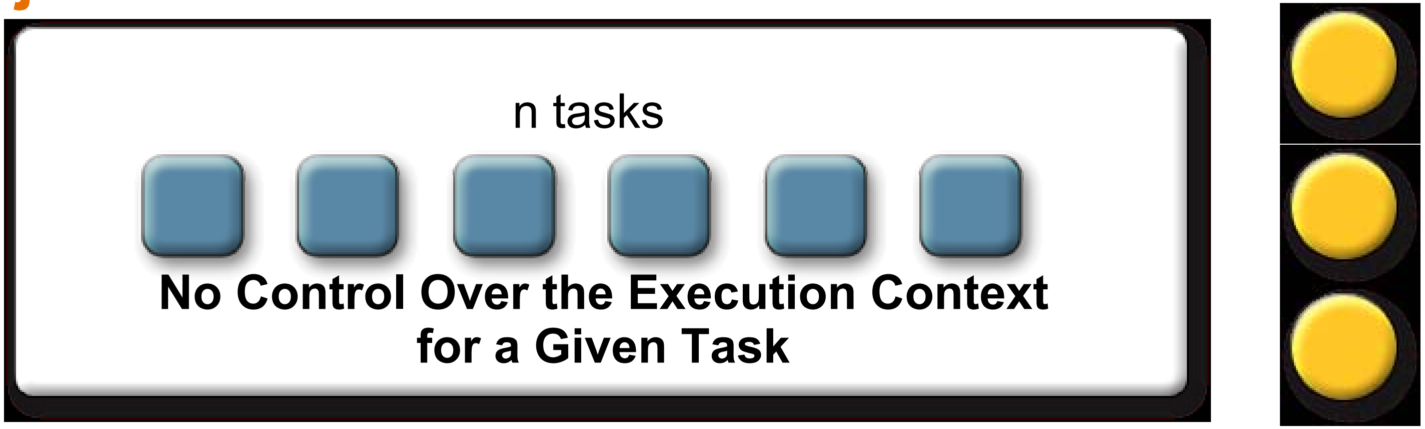
n tasks

Single Execution Context for All Tasks

1 Thread

## j.u.c.ThreadPoolExecutor

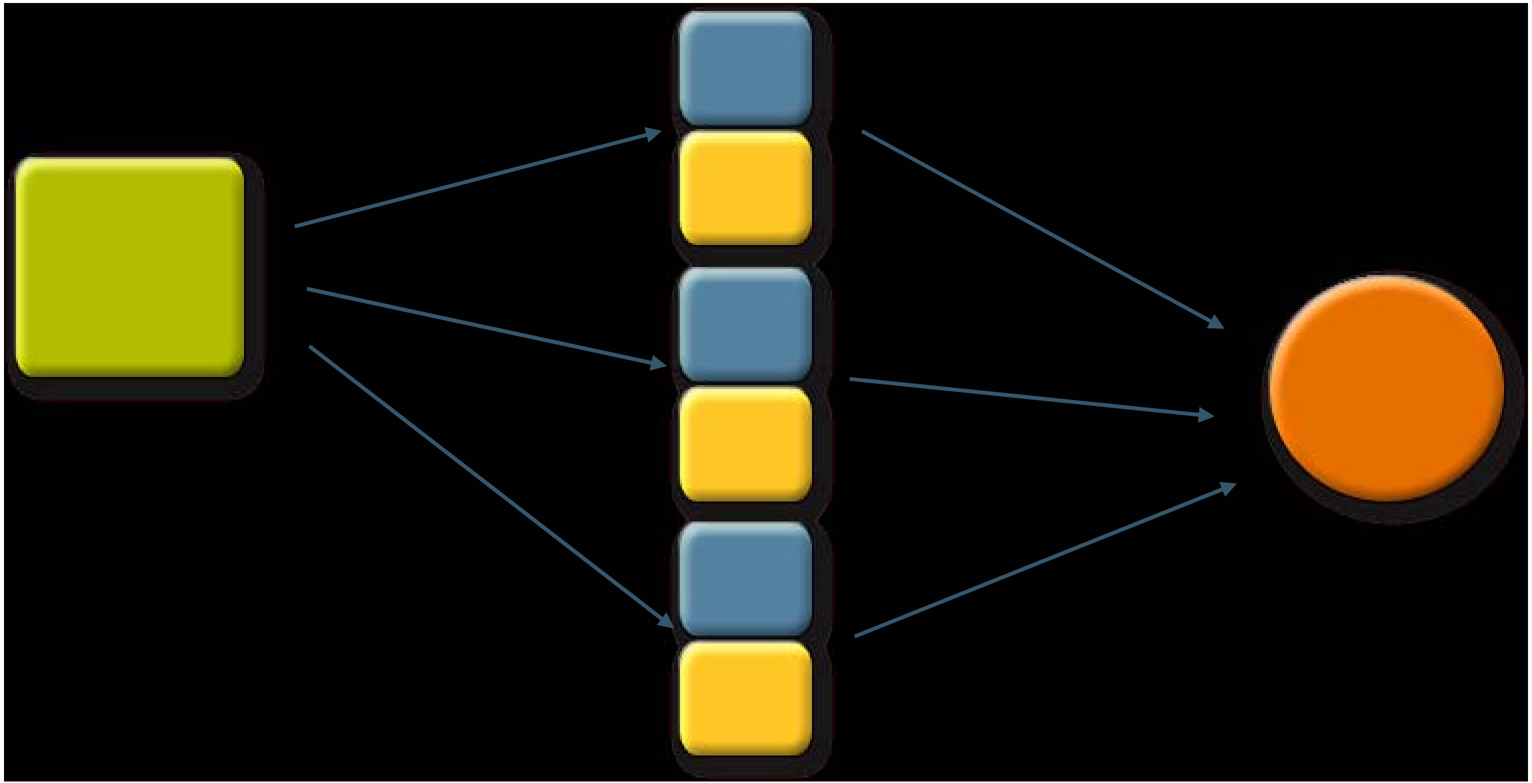
m Threads



n tasks

No Control Over the Execution Context for a Given Task

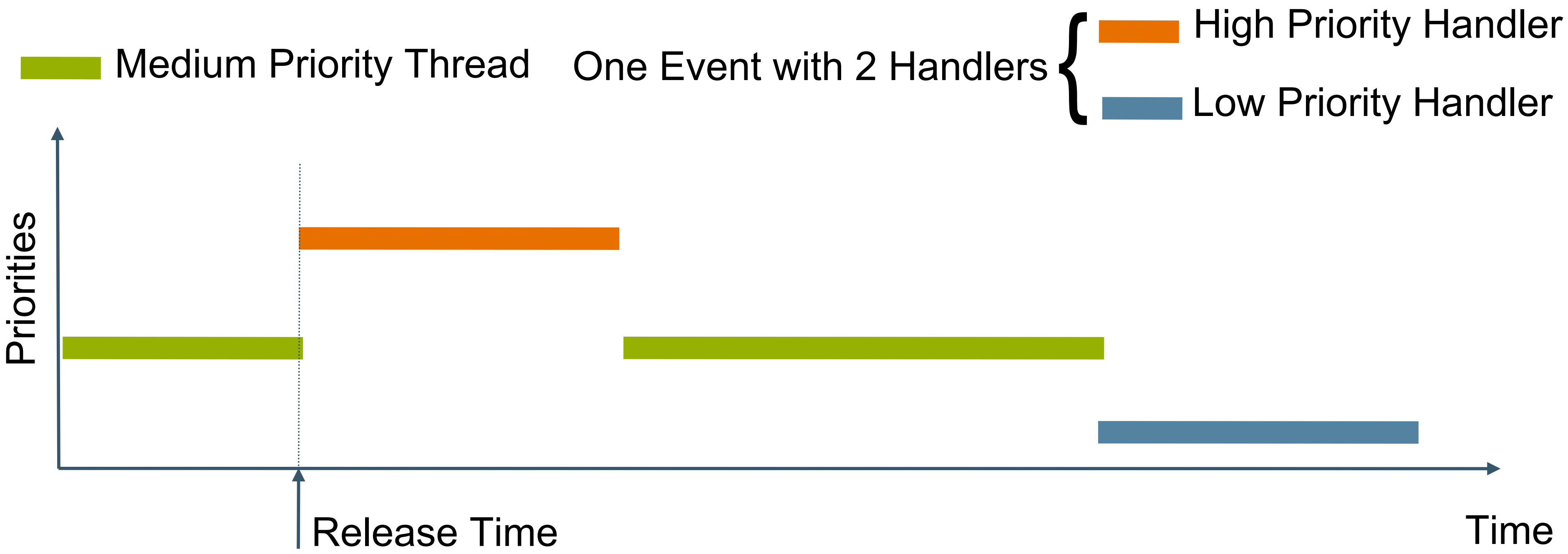
# Asynchronous Events Architecture



Event Handler: logic +
Real-Time Parameters

# Release vs. Execution

- When the event occurs, all associated handlers are released and are then executed according to their scheduling parameters



# The `AsyncEventHandler` Class

- Logic
  - **`handleAsyncEvent()`**
  - Serialized executions
- Scheduling parameters
  - Handler’s execution subject to real-time scheduling
- Optional release parameters
  - Deadline
  - Release control via Arrival Queue and Minimum Interarrival Time

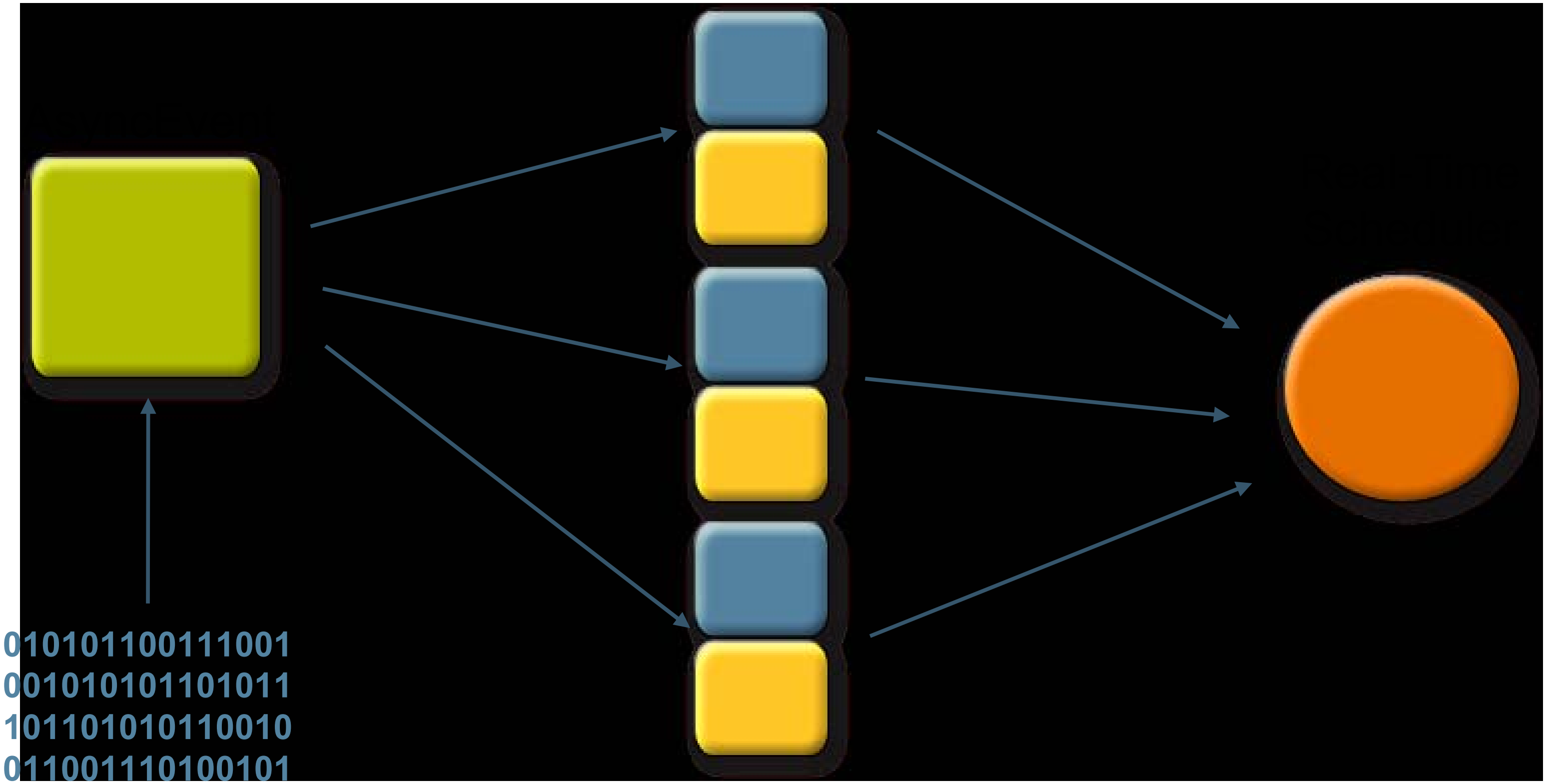
# AsyncEventHandler Instantiation

```
import javax.realtime.*;

AsyncEventHandler handler = new AsyncEventHandler(){
   public void handleAsyncEvent() {
       do_something(); }
};

SchedulingParameters sp = new PriorityParameters(
   PriorityScheduler.instance().getMaxPriority());

ReleaseParameters rp = new AperiodicParameters(
    null, deadline, null, deadline_miss_handler);

handler.setSchedulingParameters(sp);
handler.setReleaseParameters(rp);
```

# The AsyncEventHandler’s Family

- **AsyncEventHandler**
  - Dynamically bound to OS thread at execution time
  - Optimized resource usage
- **BoundAsyncEventHandler**
  - Sub-class of **AsyncEventHandler**
  - Permanent binding to OS thread
  - Lower latencies

# Application-Triggered Event



010101100111001
001010101101011
101101010110010
011001110100101
101011001000101

Application

AsyncEventHandlers

# The AsyncEvent Class

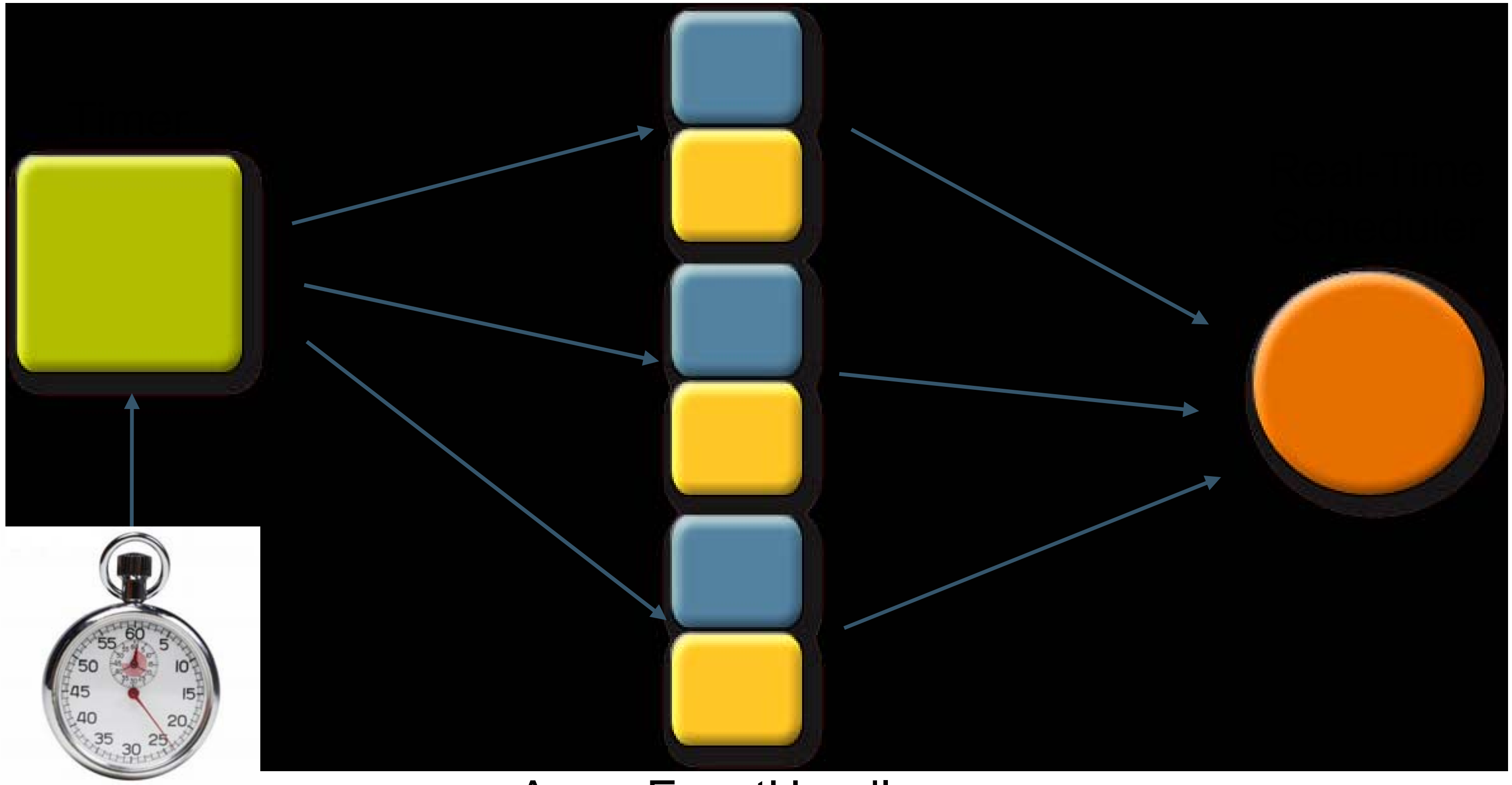
- Creating an event

```
AsyncEvent event = new AsyncEvent();

AsyncEventHandler handler = new AsyncEventHandler() {
    public void handleAsyncEvent() {
        do_something();
    }
};

event.addHandler(handler);
```

- Firing an event

```
event.fire();
```

# javax.realtime.Timer Architecture



High-Resolution Clock

AsyncEventHandlers

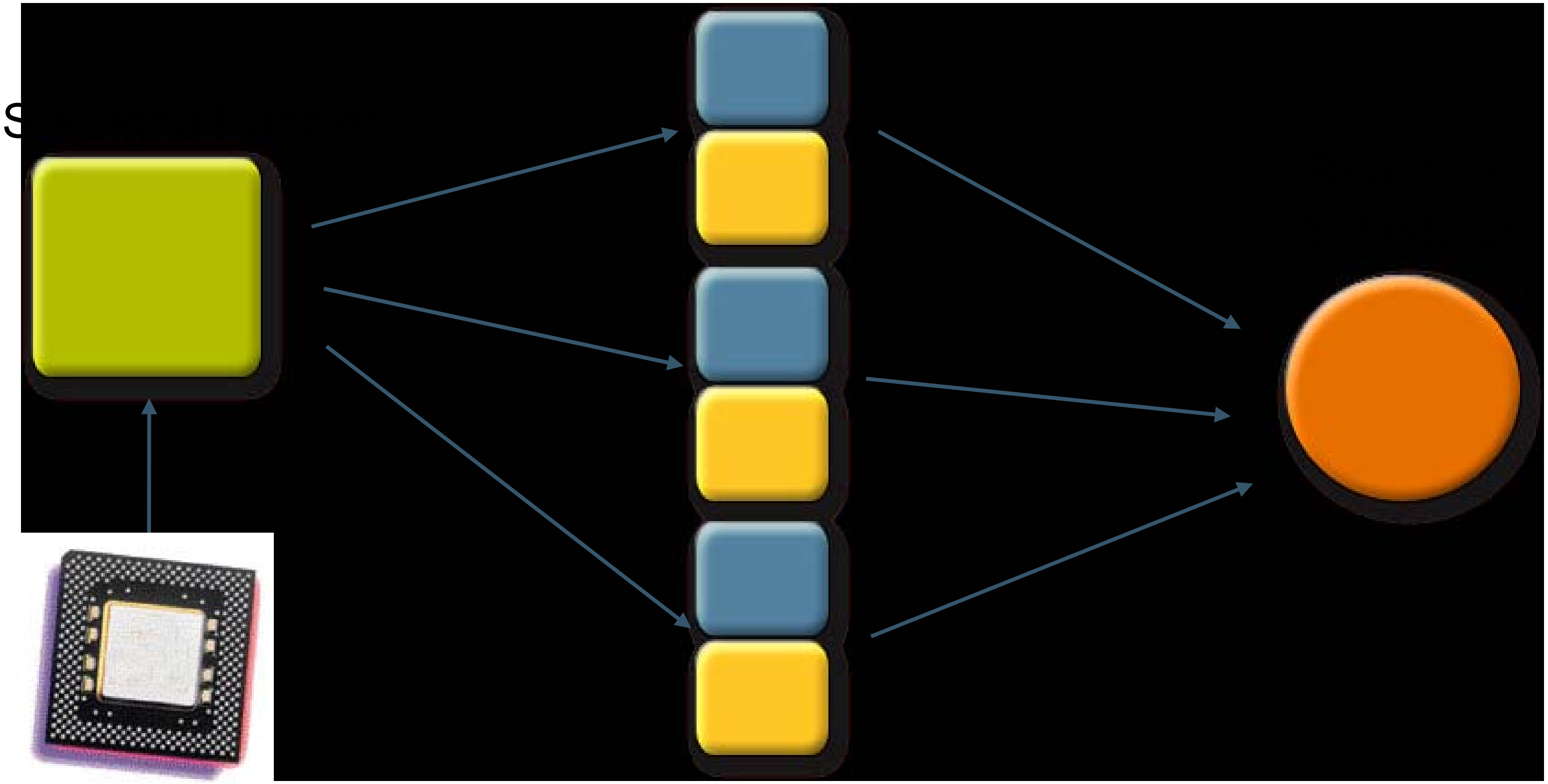
# High-Resolution Timers

- Timers are clock-dependent
  - `Timer(HighResolutionTime time, Clock clock, AsyncEventHandler handler);`
- **`HighResolutionTime` class**
  - **Representation of absolute and relative times up to nanosecond accuracy and precision**
- The default real-time clock
  - `javax.realtime.Clock.getRealtimeClock()`
- Java RTS relies on Solaris platform’s high-resolution clock

# Timer Creation

```
// Timer will start in 20 milliseconds from now
AbsoluteTime now = Clock.getRealtimeClock().getTime();
AbsoluteTime start = now.add(new RelativeTime(20,0));

// A periodic timer with a 8.5 millisecond period
RelativeTime period = new RelativeTime(8,500000);


PeriodicTimer timer =
    new PeriodicTimer(start, period, handler);

timer.addHandler(another_handler);

timer.start();
```

# `POSIXSignalHandler` Architecture



# POSIX Signals

- Supported by most modern OSes
- Simple and efficient
- Still, hard to use when shared across multiple libraries and in multithreaded programs
  - Signal masks
  - Signal handler chaining
- `javax.realtime.POSIXSignalHandler`
  - **Can be associated with many handlers**
  - **No signal mask to configure**

# Installing a Signal Handler

```
AsyncEventHandler handler = new AsyncEventHandler() {
   public void handleAsyncEvent() {
      do_cleanup();
   }
};

AsyncEventHandler handler2 = new AsyncEventHandler() {
   public void handleAsyncEvent() {
      do_log_event();
   }
};

POSIXSignalHandler.addHandler(SIGQUIT, handler);

POSIXSignalHandler.addHandler(SIGQUIT, handler2);
```

# Asynchronous Events in Java RTS

- Generic architecture
  - Applied to many event sources
    - High-resolution timers, POSIX signals, application events
- Handling of an event decoupled from occurrence
  - Easy to associate multiple handlers to the same event
- All tasks have a specific real-time context
  - AsyncEventHandler
    - Scheduling parameter
    - Release parameters

# Agenda

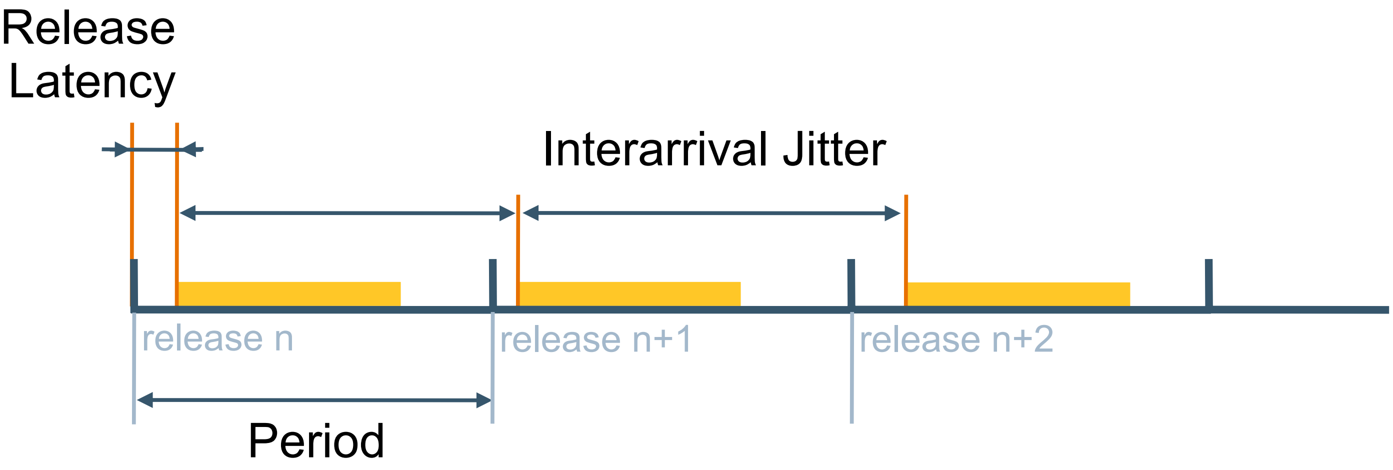
Introducing Java Real-Time System

A Tour of Key Real-Time Features

- Thread Scheduling
- Asynchronous Event Handling
- **Periodic Execution**
- Monitoring

# Periodic Execution

- Fundamental paradigm for most control systems
  - Closed-loop, PID controllers



# Creating Periodic Real-Time Threads

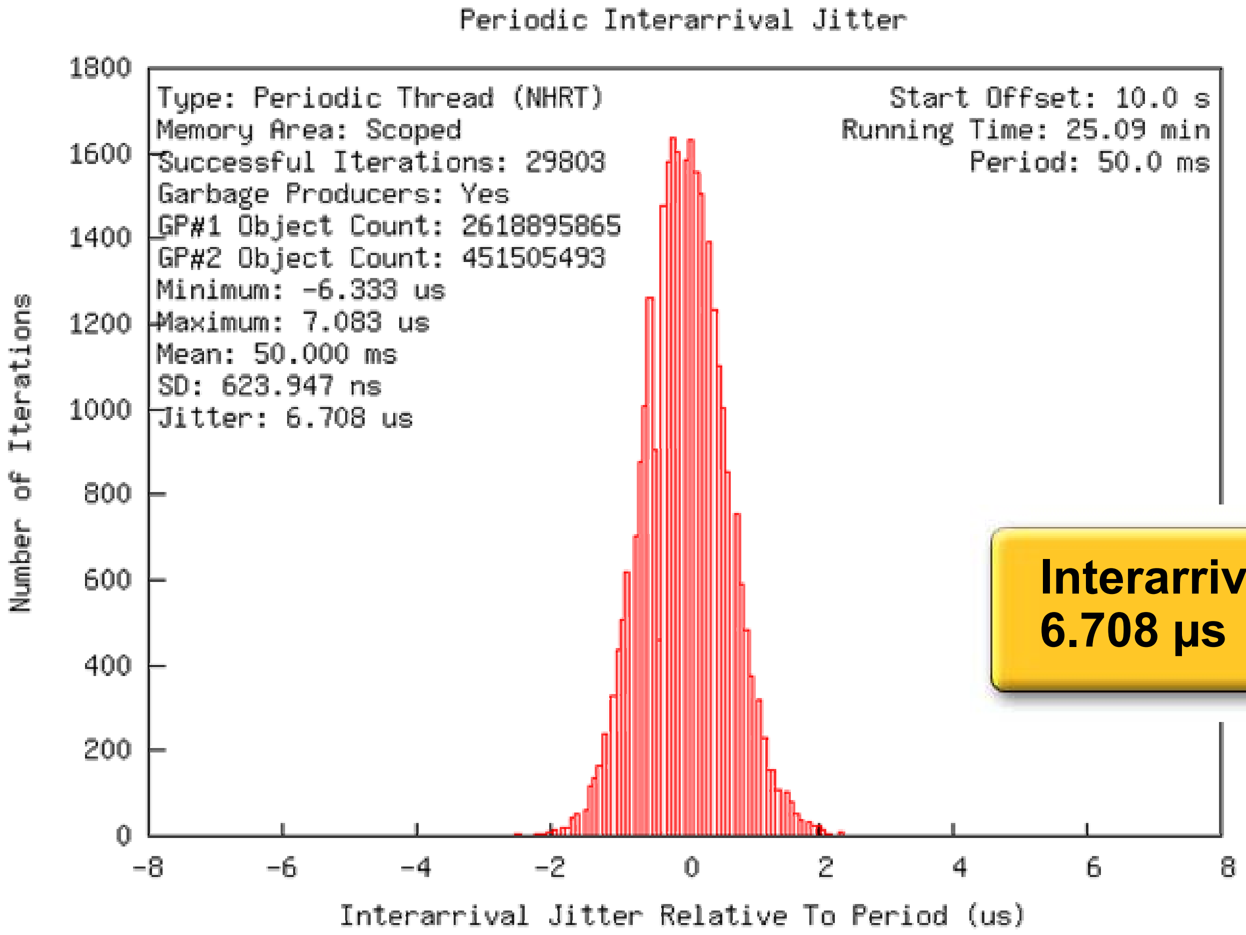
- **Controlled by periodic release parameters**

```
relParms = new PeriodicParameters(start, period);
setReleaseParameters(relParms);
```

- **Periodic behavior achieved by:**
  - Executing in a loop
  - Invoking waitForNextPeriod()

```
while (true) {
    do_control();
    waitForNextPeriod();
}
```

# Java RTS, Interarrival Jitter



Type: Periodic Thread (NHRT)
Memory Area: Scoped
Successful Iterations: 29803
Garbage Producers: Yes
GP#1 Object Count: 2618895865
GP#2 Object Count: 451505493
Minimum: -6.333 us
Maximum: 7.083 us
Mean: 50.000 ms
SD: 623.947 ns
Jitter: 6.708 us

Start Offset: 10.0 s
Running Time: 25.09 min
Period: 50.0 ms

**Interarrival Jitter: 6.708 µs**

Java RTS 1.0 on a Sun Fire™ V210 Server, 2 x UltraSPARC® IIIi @ 1GHz

# Java RTS Support for Periodic Execution

- Supported via a dedicated device driver
  - Leverages Solaris platform kernel’s internal “cyclic” subsystem
  - Features low-latency, high-precision timed operations
  - Time source is consistent with JVM software’s real-time clock
- Device driver’s built-in RTSJ semantics
  - Periodic activities
  - Deadline monitoring

# Agenda

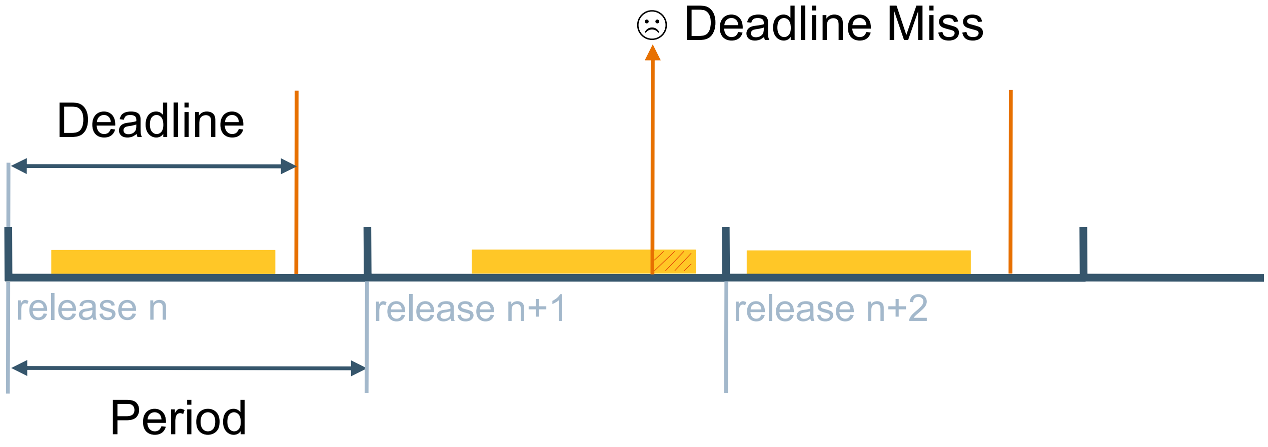
Introducing Java Real-Time System

A Tour of Key Real-Time Features

- Thread Scheduling
- Asynchronous Event Handling
- Periodic Execution
- **Monitoring**

# Execution Monitoring

- The RTSJ offers facilities to monitor, and react to, abnormal temporal conditions



- **Deadline miss condition** entered when the absolute deadline is reached

# Deadline Miss Handling

- **Opportunity offered to the application to recover**
- Handling is either:
    - Deferred to a deadline miss handler

```
ReleaseParameters.setDeadlineMissHandler(
        AsyncEventHandler handler);
```

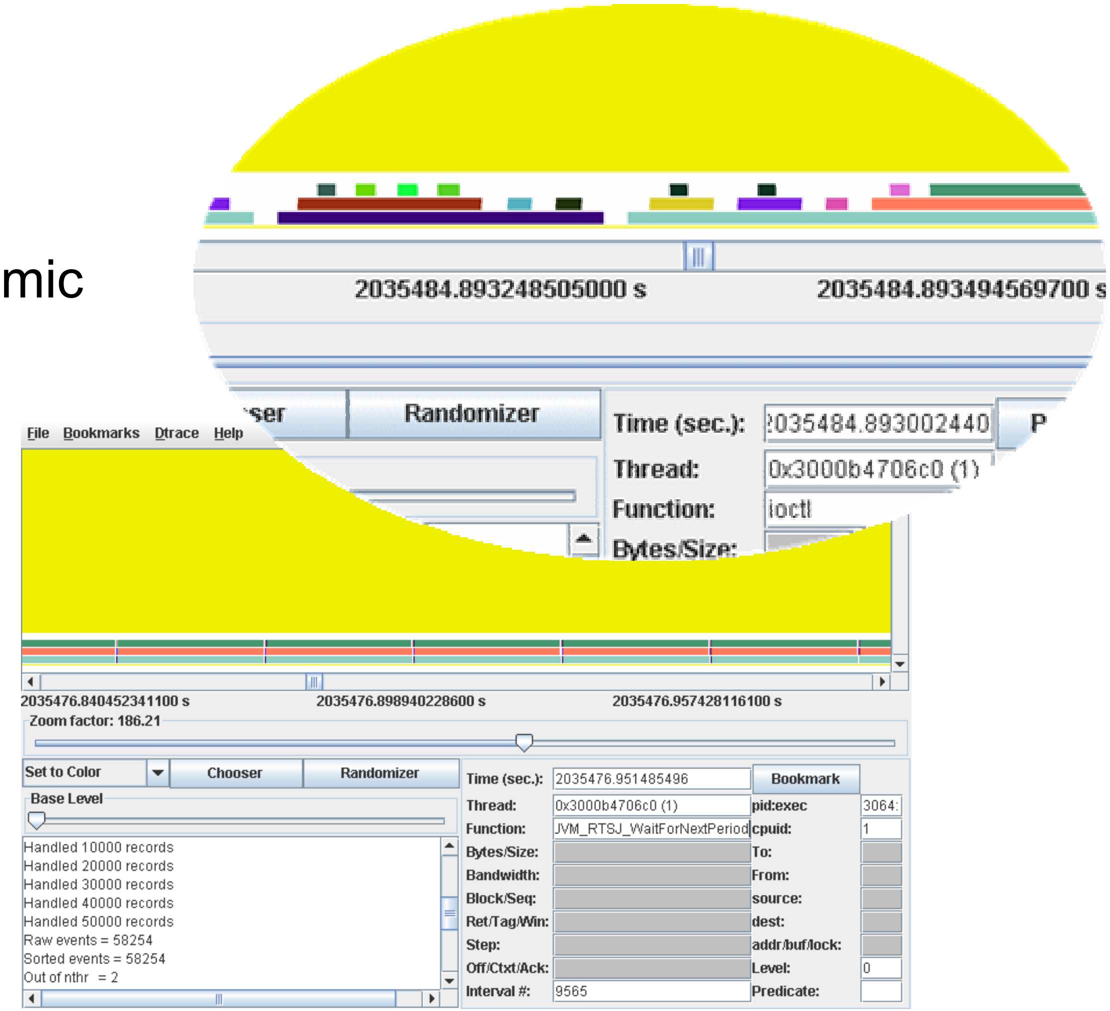
    - Or, if no handler, performed by the thread itself

```
if (waitForNextPeriod() == false) {
        handle_deadline_miss();
}
```

# Java RTS Monitoring

Call Stack Analysis

- Built-in Support for DTrace
  - Solaris platform dynamic tracing facility
  - Java RTS-specific probes
- DTrace features
  - Kernel probes
  - JVM software probes
  - Scheduling events
  - Call stack
  - And much more…



2035484.893248505000 s

2035484.893494569700 s

Time (sec.): 2035484.893002440

Thread: 0x3000b4706c0 (1)

Function: ioctl

Bytes/Size:

2035476.840452341100 s

2035476.898940228600 s

2035476.957428116100 s

Zoom factor: 186.21

```
Handled 10000 records
Handled 20000 records
Handled 30000 records
Handled 40000 records
Handled 50000 records
Raw events = 58254
Sorted events = 58254
Out of nthr   = 2
```

Time (sec.): 2035476.951485496

Thread: 0x3000b4706c0 (1)

pid:exec 3064:

Function: JVM_RTSJ_WaitForNextPeriod

cpuid: 1

Bytes/Size:

To:

Bandwidth:

From:

Block/Seq:

source:

Ret/Tag/Win:

dest:

Step:

addr/buf/lock:

Off/Ctxt/Ack:

Level: 0

Interval #: 9565

Predicate:

# Summary

- Real-time is about **time** and **control**, not speed
- The **Real-Time Specification for Java** defines how real-time occurs within Java technology
- **Java Real-Time System**, based on Java SE platform 5, is Sun's implementation of the RTSJ
- The **Solaris Operating System**'s built-in real-time capabilities make it the platform of choice for Java Real-Time System

# For More Information

- TS-1205, The Sun Java Real-Time System Meets Wall Street
- TS-2901, A Real-Time Garbage Collector for a Real-Time Java Virtual Machine
- LAB-7250, The Real-Time Java Programming Challenge
- java.sun.com/javase/technologies/realtime.jsp
- www.rtsj.org

# Q&A

# *Sun Java™ Real-Time System Revealed*

**Frederic Parain**
**David Holmes**

Java Real-Time System Engineering
Sun Microsystems, Inc.
java.sun.com

TS-1331