



Real-time Java

Simon Ritter

Technology Evangelist

Sun Microsystems

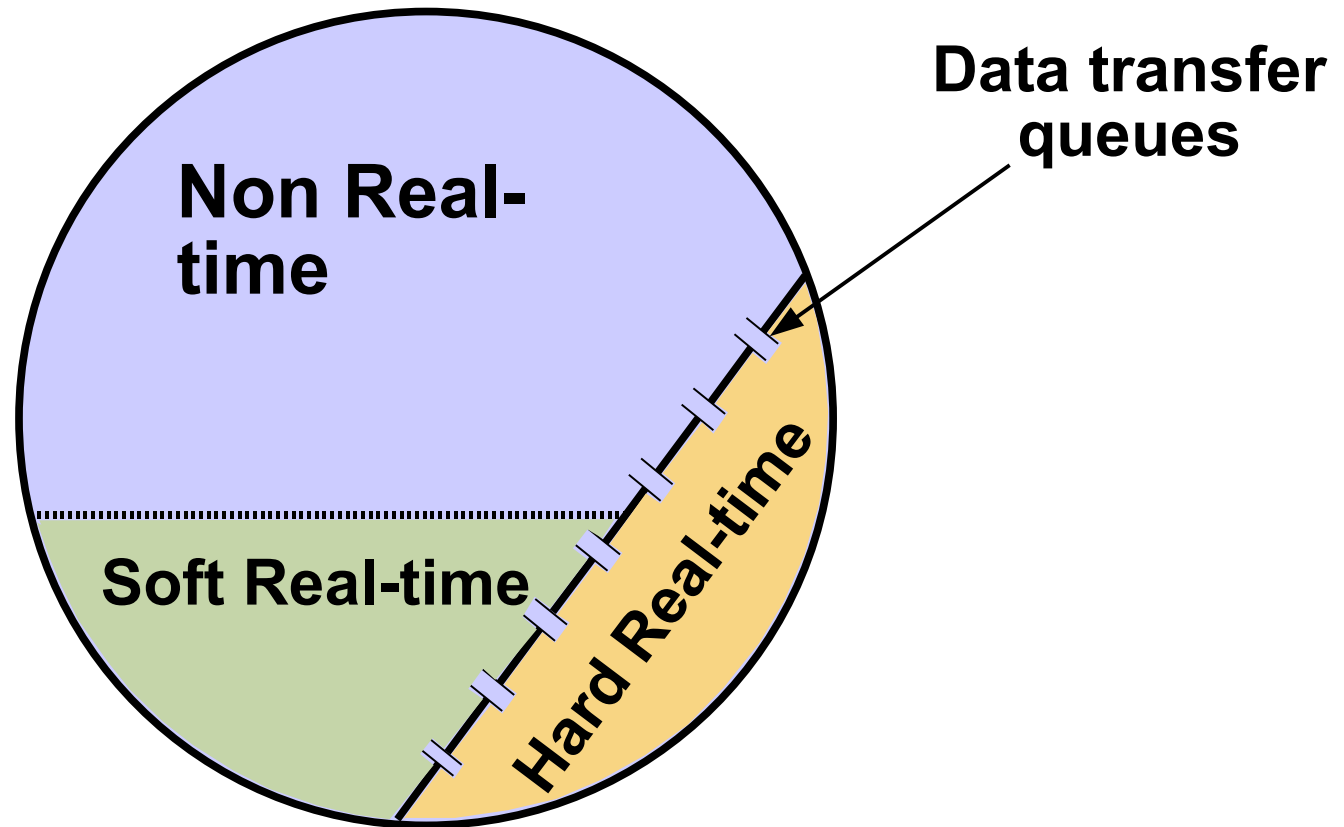
Why Real-Time Java

- Many systems need guaranteed response time
 - > Doesn't have to be fast
 - > Non-deterministic behaviour can be fatal
- Existing mechanical solutions too complex
 - > Analog systems, hydraulic, pneumatic control
- Computers are taking over control
 - > Fly by wire
 - > Software systems needed for flexible control
- Ease of development essential

Real-Time Specification for Java (RTSJ) JSR-001 (Project Macinac)

- Started in 1998. Experts from many communities
 - > Embedded systems design, Ada design, Java-based design, embedded processor design, real-time systems design, academia, RTOS design
- 100% pure Java
- Provides API set, semantic Java VM enhancements, and Java VM-to-OS layer modifications
- Allow developers of Java-based application to correctly reason about and control the temporal behavior of Java-based applications

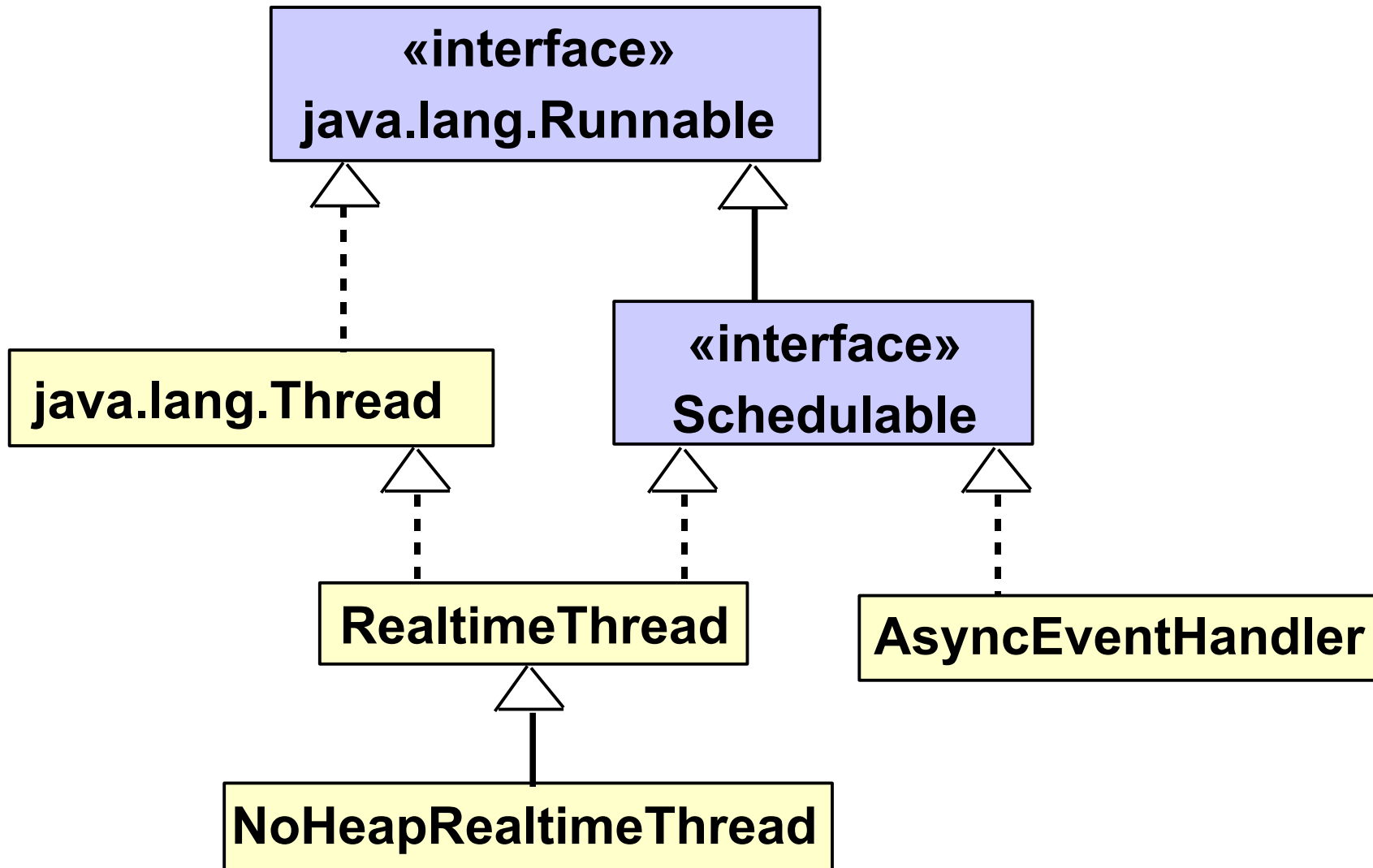
Real-Time and Non-Real-Time



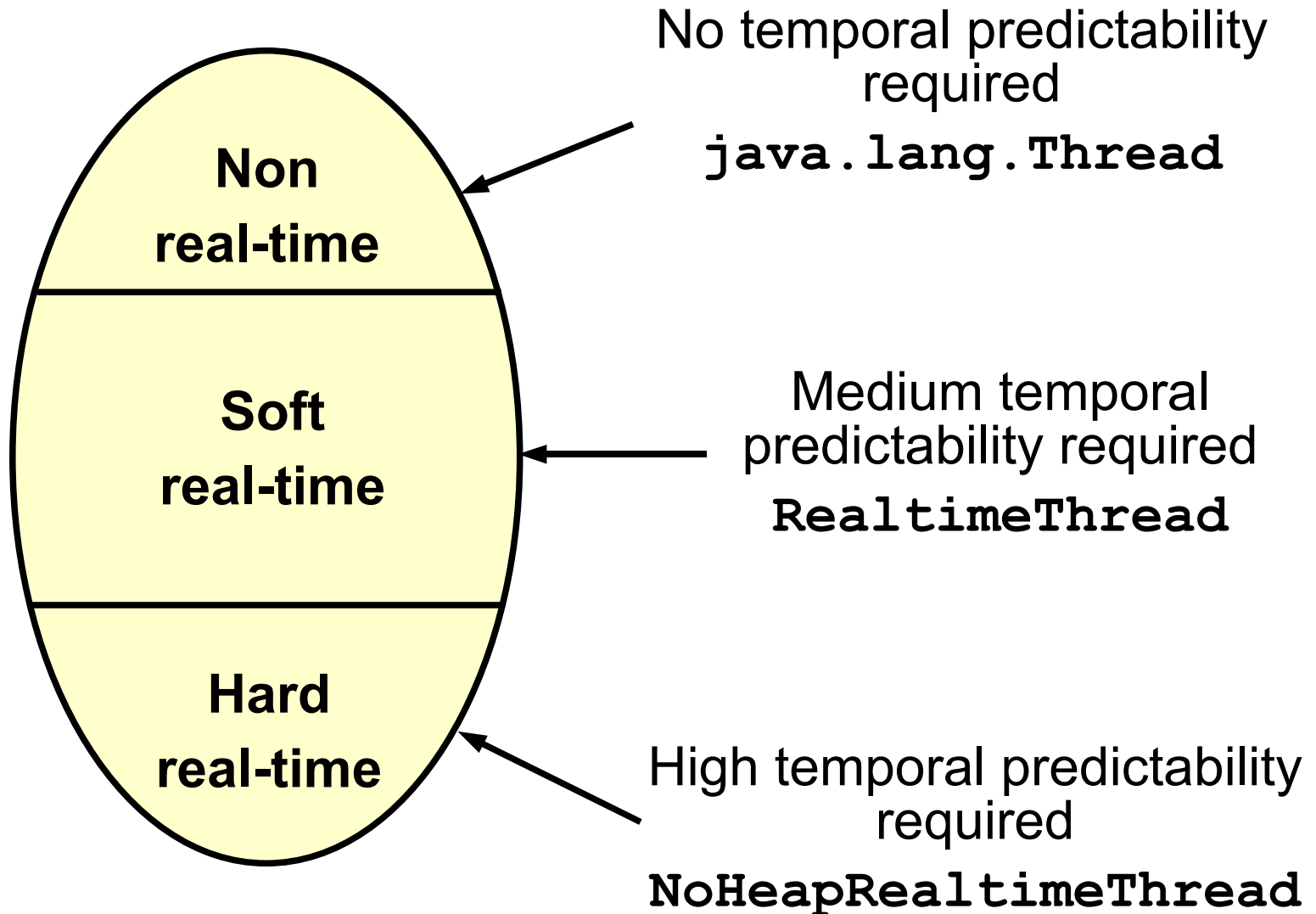
Real-time Java Features

- Threads and scheduling
- Synchronization and priority inversion avoidance
- Memory management (Garbage collection)
- Asynchrony
- Time/Timers

RTSJ Threads



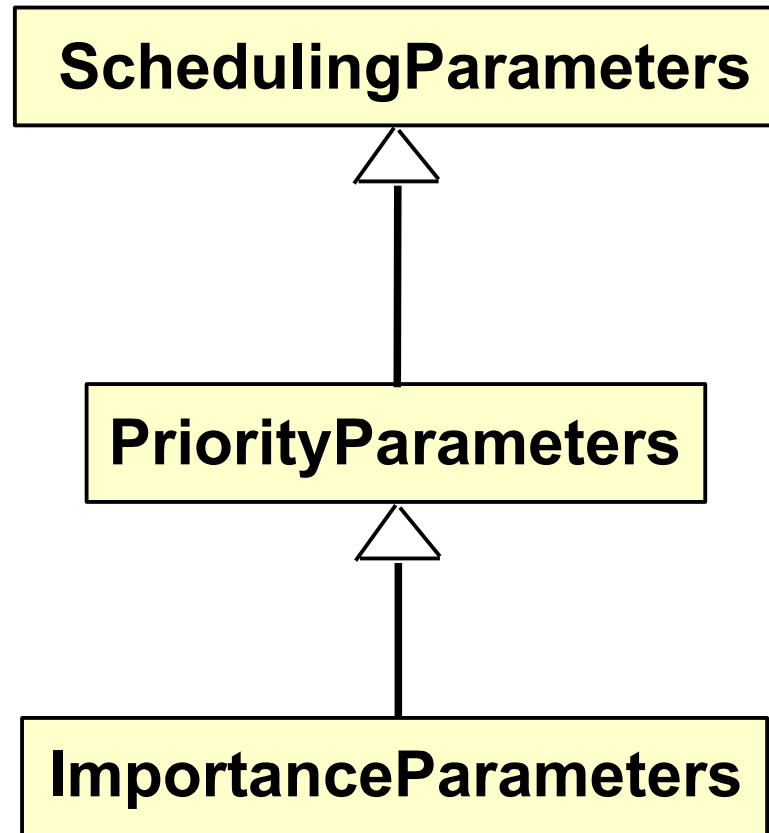
Threads



RTSJ Scheduling

- Fixed-priority preemptive scheduler
 - > At least 28 distinct real-time priorities
 - > 32-bit integer for priority, so many, many more possible
 - > 10 non-real-time priorities for backwards compatibility
- Tasks at higher priority always run in preference to tasks with lower priority
- Tasks with higher priority will preempt lower-priority tasks
- Implements priority-inversion avoidance

Scheduling Parameters



Release Parameters

Execution cost
 Deadline
 Cost overrun handler
 deadline overrun handler

ReleaseParameters

Start time
 period

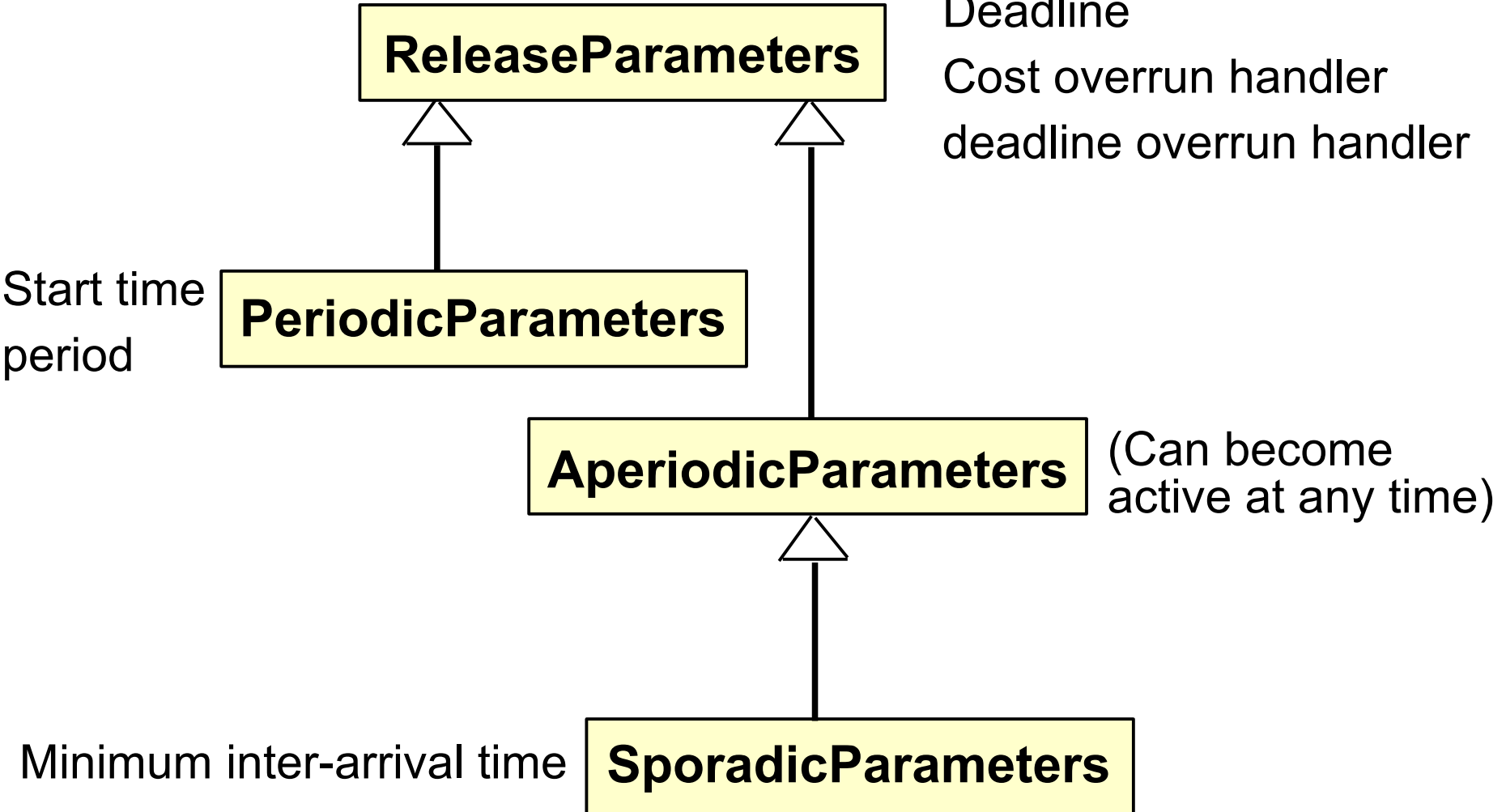
PeriodicParameters

AperiodicParameters

(Can become active at any time)

Minimum inter-arrival time

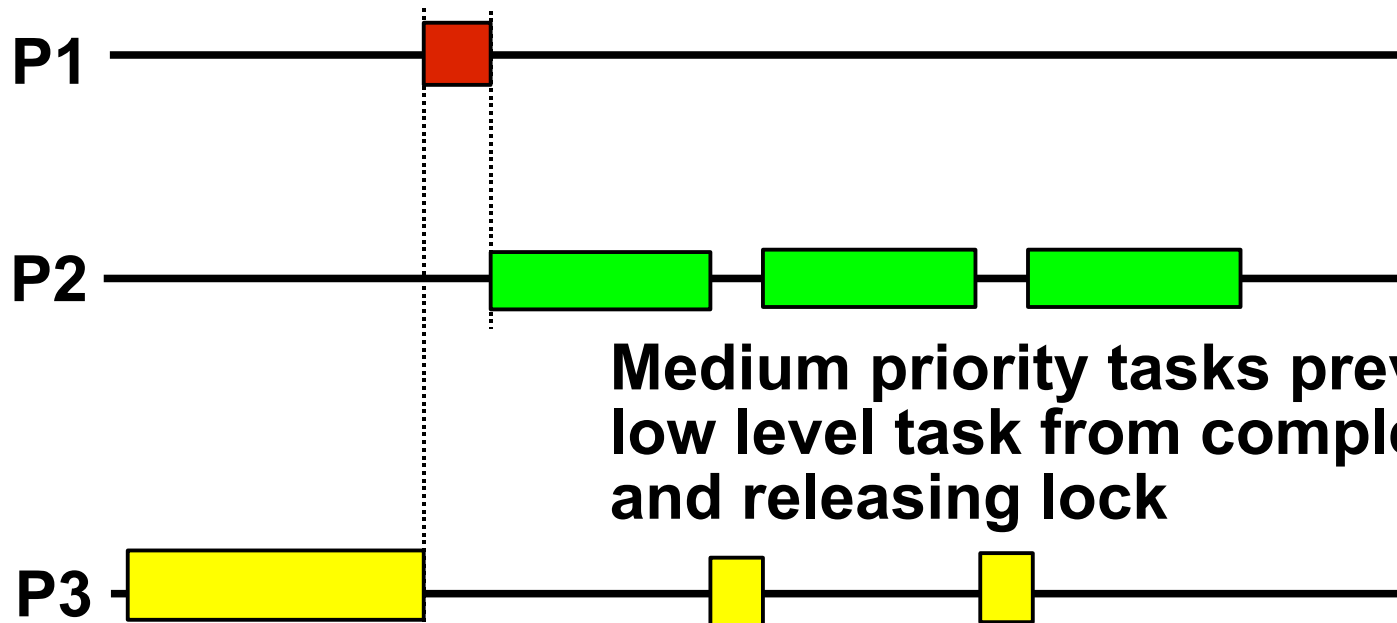
SporadicParameters



Priority Inversion Problem

Tries to acquire same lock - thread blocked

High priority task is blocked indefinitely by low priority task



Medium priority tasks prevent low level task from completing and releasing lock

Lock Aquired

Priority

Priority Inversion

A “Real World” Example: Mars Pathfinder

- Spacecraft had two high priority tasks
 - > Data distribution, task scheduling
- Low priority task (gathering data) aquired lock via call to IPC mechanism
- Distribution task got blocked trying to aquire same lock
 - > Other medium priority tasks prevented data gathering task from completing and releasing lock
- Scheduler detects distribution task has not completed in required time and takes action
 - > Reboots spacecraft!

Priority Inversion Avoidance

- Priority Inheritance
 - > Task holding lock gets priority elevated to that of blocked thread until lock is released
 - > No application code changes required
 - > Must be supported by all RTJS implementations
- Priority Ceiling Emulation
 - > Task sets its own priority to be higher than any other thread that might try to acquire the lock
 - > Priority returned after lock released
 - > Requires application code changes (correct usage)
 - > Optional part of RTSJ spec

Programming Soft Real-Time in Java RTS

Step 1:

Replace:

```
Thread T = new Thread(myRunnable);
```

with

```
RealtimeThread RT =  
    new RealtimeThread(..., myRunnable);
```

Step 2:

There is no step 2!

Data Transfer Queues

- Communication/synchronisation between RealTimeThread and java.lang.Thread
- WaitFreeDequeue
- WaitFreeReadQueue
- WaitFreeWriteQueue

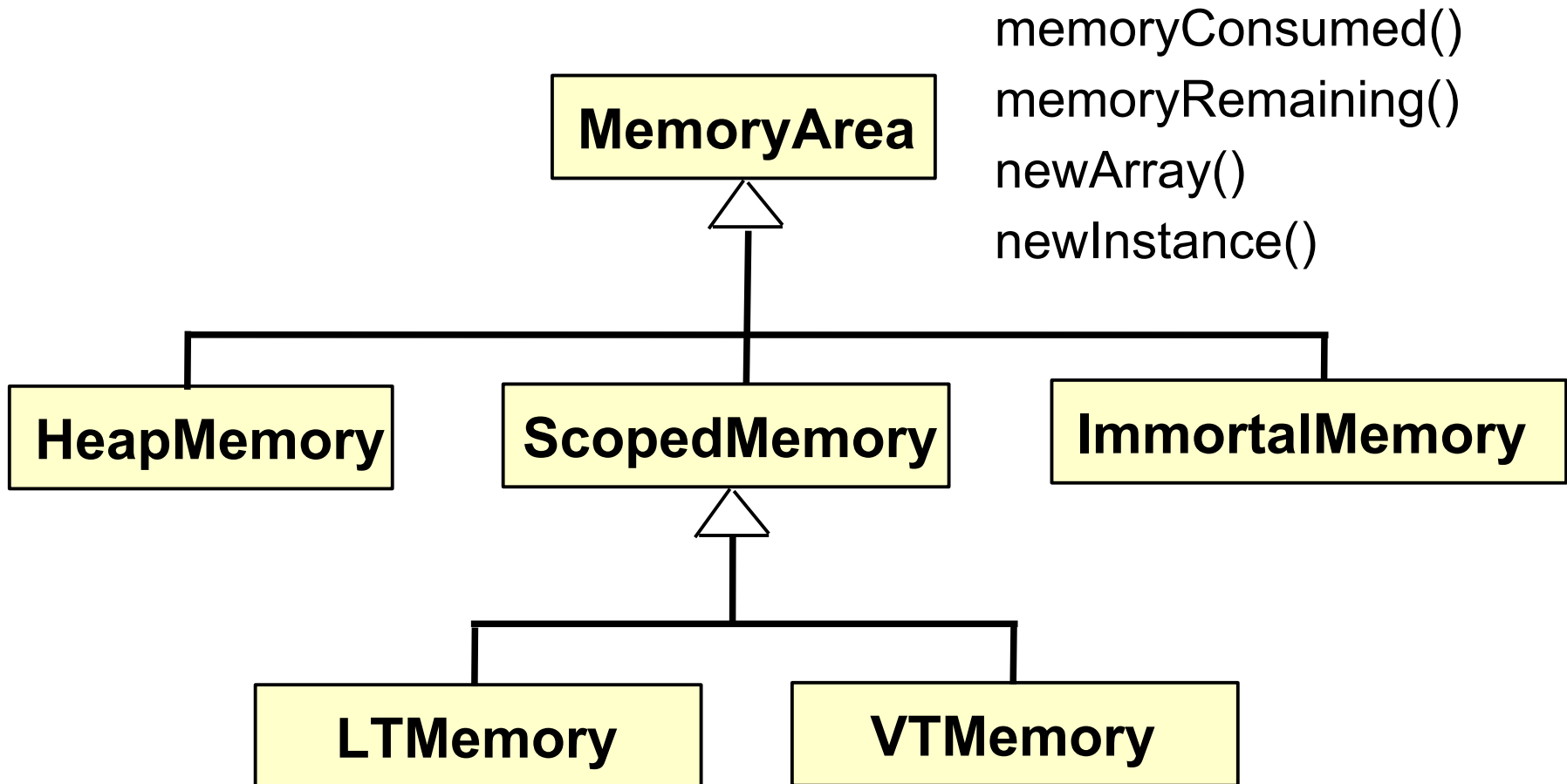
Memory Management

- C/C++ memory management is completely under program control
 - > malloc(), free()
 - > Obvious disadvantages for memory leaks, invalid pointers
- Java uses automatic memory management
 - > Eliminates problems of free()
 - > Introduces non-deterministic behaviour to application as GC cannot be controlled directly

RTSJ Memory Management

- Regions of memory available outside of Java heap
- Not subject to traditional GC (non-deterministic)
- Can be used to communicate results
 - > Different RT threads
 - > RT and non-RT threads
- Access to physical memory

RTSJ Memory Management



Scoped Memory

- Lifetime of an object is determined by its scope
 - > Exists until the run method exits for the thread
 - > Collection is simply invalidating the scoped memory
- Uses application-defined heap
- Allocation time can be linear or variable
 - > LTMemory
 - > VTMemory

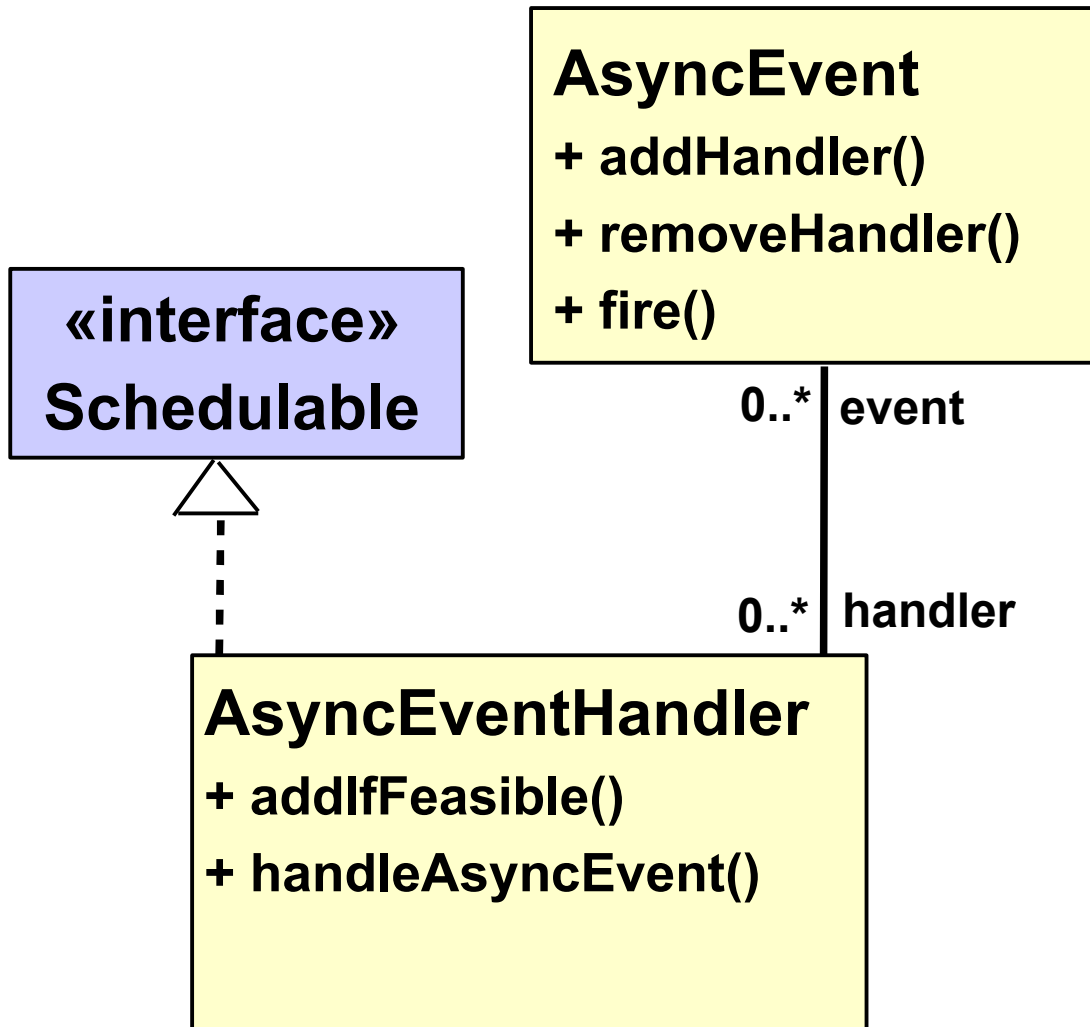
Immortal Memory

- Shared amongst all threads
- Objects allocated here are never garbage collected
 - > Live until end of application
- Five mechanisms for allocating immortal memory
 - > implicit
 - > newInstance()
 - > newArray()
 - > enter()
 - > executeInArea()

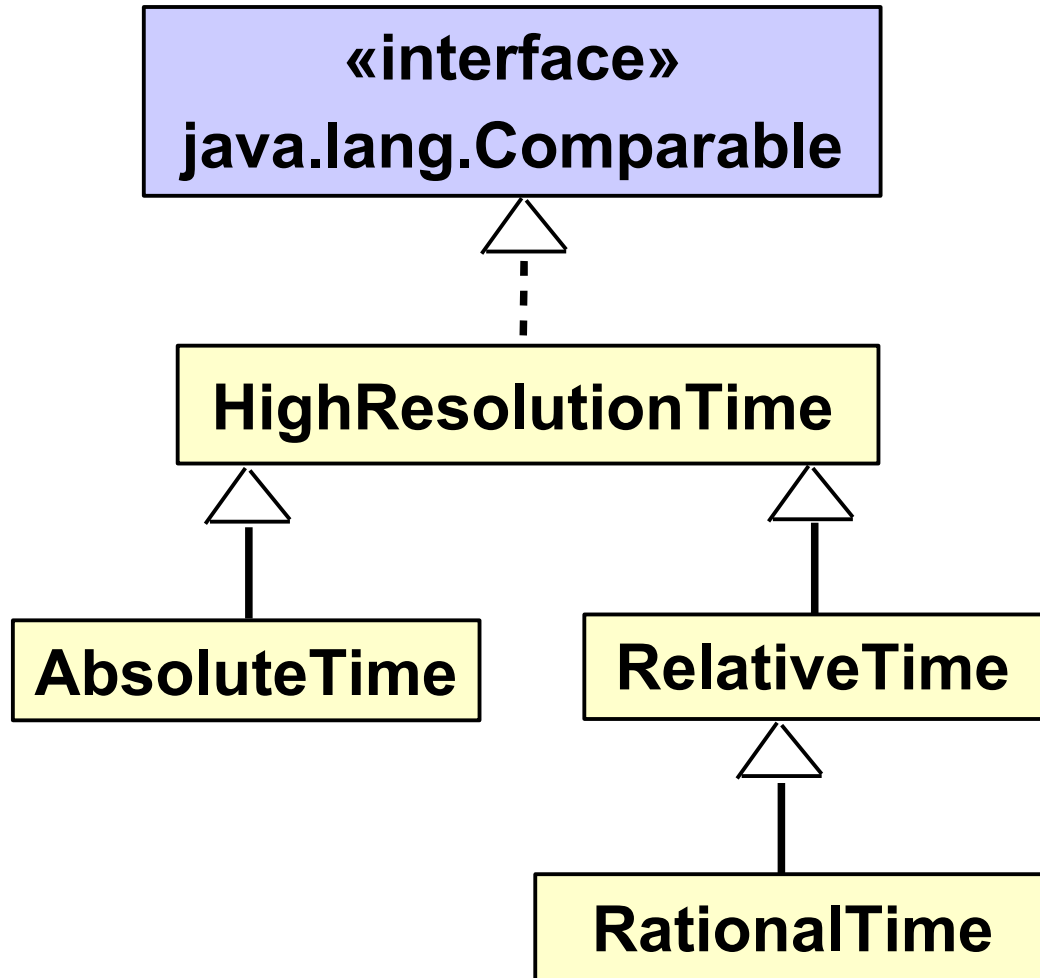
Physical Memory

- PhysicalMemoryManager
- ImmortalPhysicalMemory
- Allocation
 - > VTPhysicalMemory, LTPhysicalMemory
- RawMemoryAccess
 - > RawMemoryFloatAccess

Asynchrony

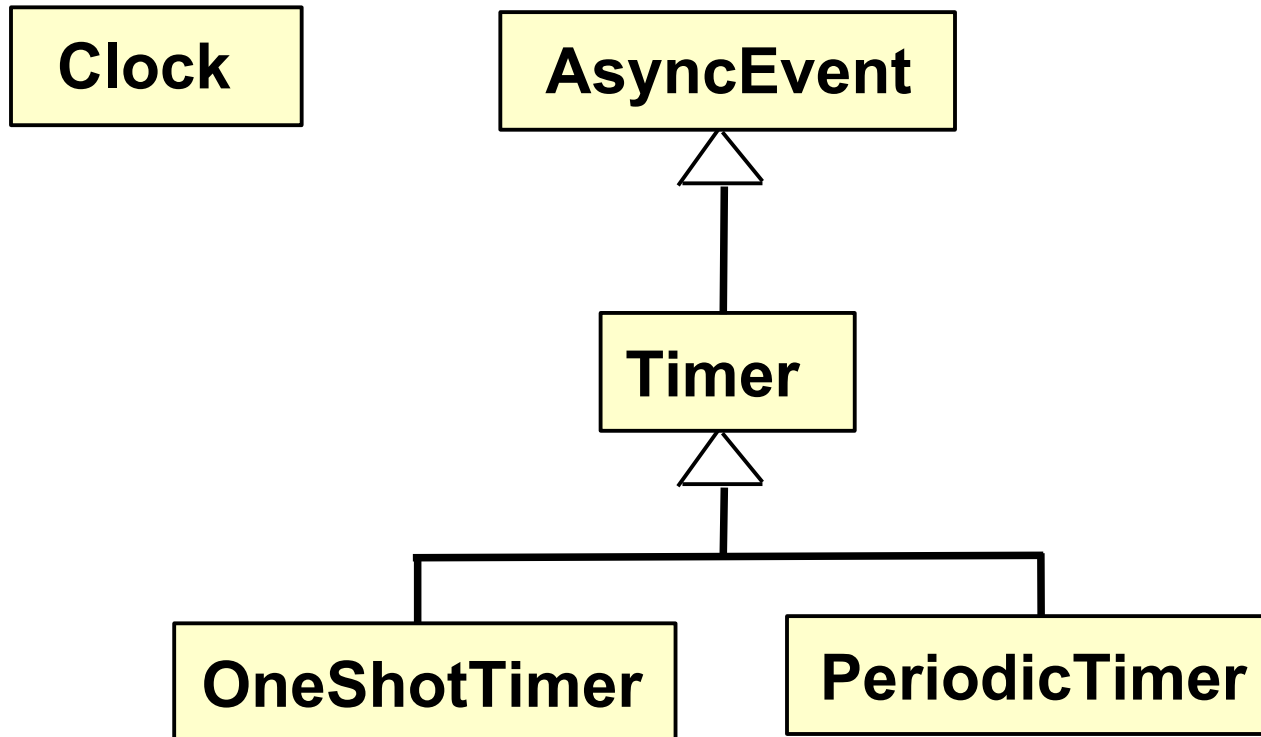


Time

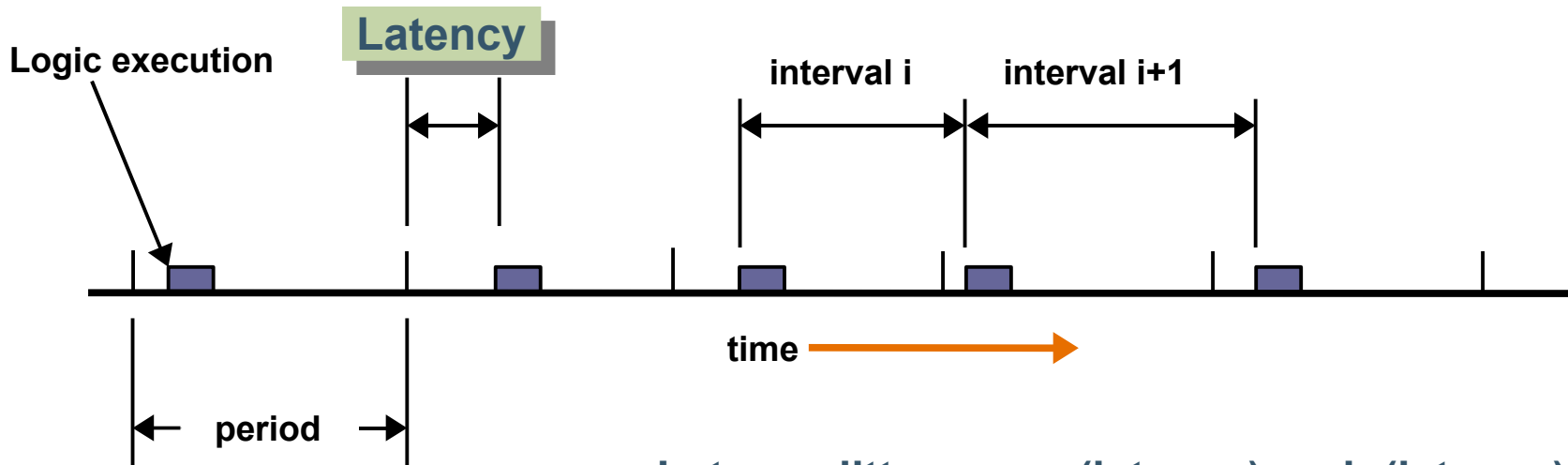


All classes require nanosecond precision

Timers



Latency and Jitter



$$\text{Latency jitter} = \max(\text{latency}) - \min(\text{latency})$$

$$\text{Interval jitter} = (\max(\text{interval}) - \min(\text{interval})) / 2$$

Latency is the measure of how long it takes the system to respond to an event.

Jitter is the variability of the latency value.

For both: lower is better.

RTSJ Latency and Jitter Numbers

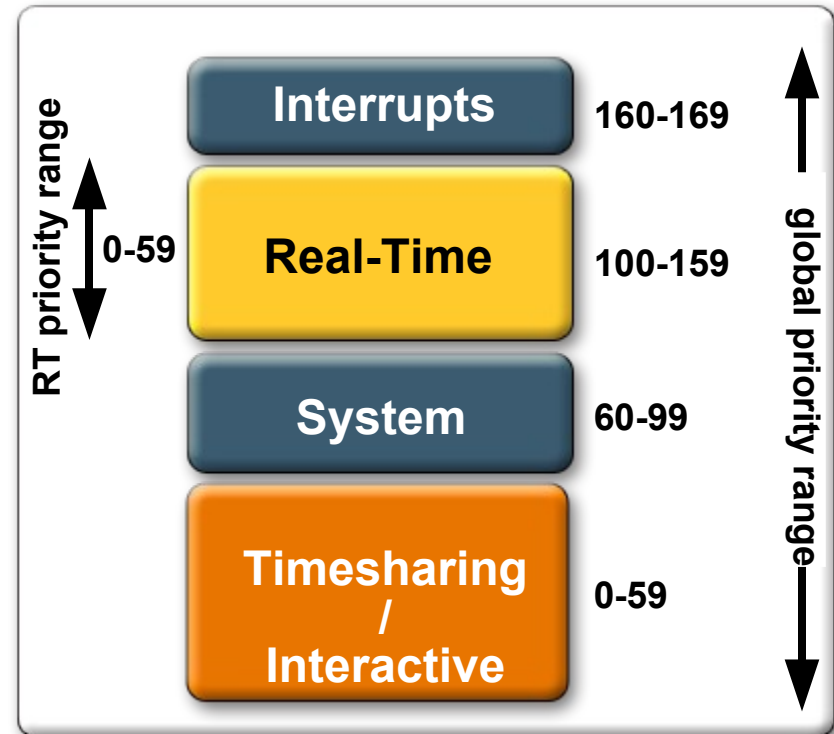
- Java RTS system configurable to:
 - > Maximum Jitter: < 10 microseconds
 - > Maximum Latency: < 20 microseconds
- As good as the best commercial RTOSs
 - > And often better, particularly on faster processors with more memory

RTJS 2.0

- Java SE 5 compatible
- Reference Implementation runs on Solaris
 - > Solaris supports true real-time scheduler
- Also supports RT POSIX compliant Linux
- Real Time garbage collector
- NetBeans plugin module

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



Uses of RTSJ



**Inverted pendulum
problem**



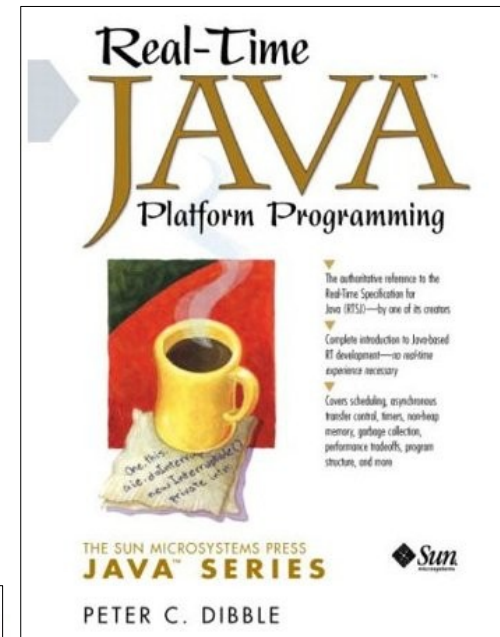
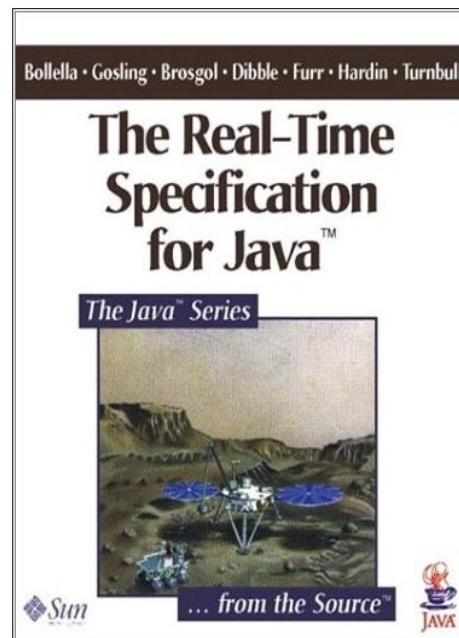
Boeing Scan-Eagle UAV

Summary

- RTSJ provides a genuine hard real-time system for Java
- RTJS 2.0 brings this to the mainstream with Solaris
- Expect to see a RT app server in the near future

Resources

- www.jcp.org
- www.rtfj.org
- rtsj.dev.java.net



Real-time Java

Simon Ritter

simon.ritter@sun.com