Introduction to Real Time Java

Deniz Oğuz

Initial:22.06.2008
Update:05.12.2009

Outline

- What is Real-time?
- What are the problems of Java SE?
- Real-Time Linux
- Features of RTSJ (Real Time Specification for Java), JSR 1, JSR 282
- IBM Websphere Real-Time
- SUN RTS

What is Real-time?

- Real-time does not mean fast
  - Throughput and latency are important but does not enough
- Real-time is about determinism
  - Deadlines must be met
- Real-time is not just for embedded systems
  - Financial applications
  - Telecommunication applications
  - Network centric systems etc...

Categories of Application Predictability

None Real-Time

Hard Real-Time

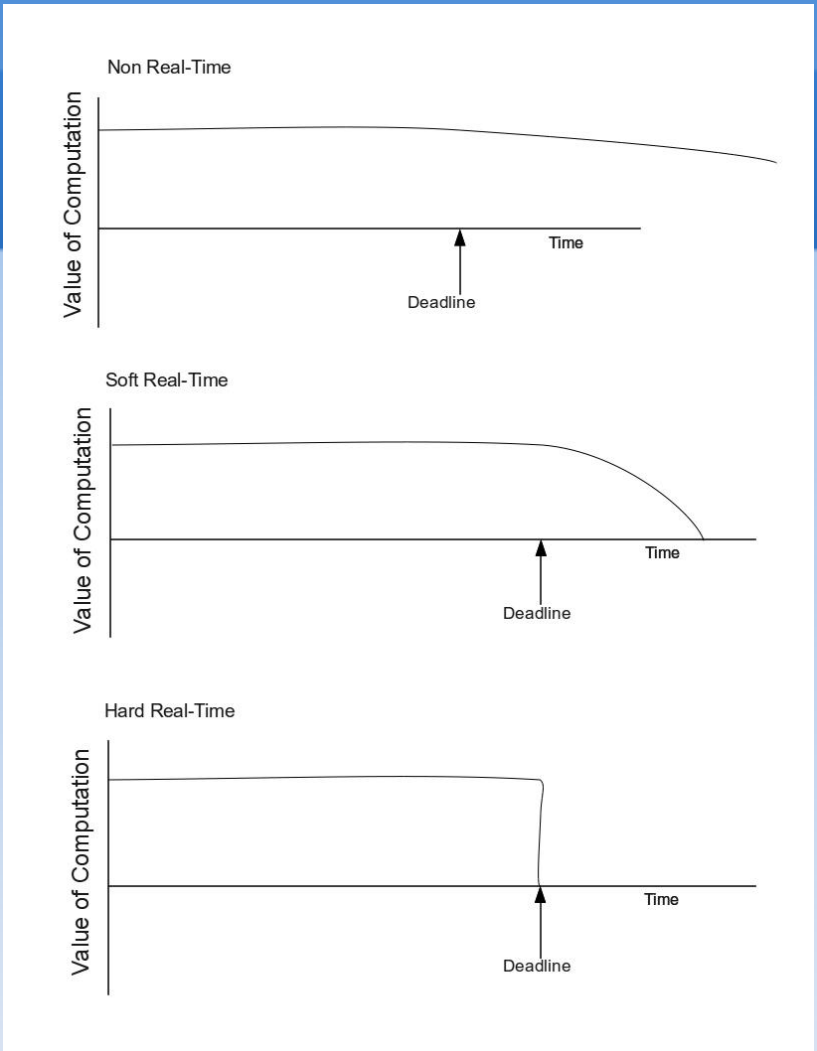
No time-based deadlines
Ex:Batch processing, web services

Deadlines may be missed occasionally
Ex:Router, automated trading systems

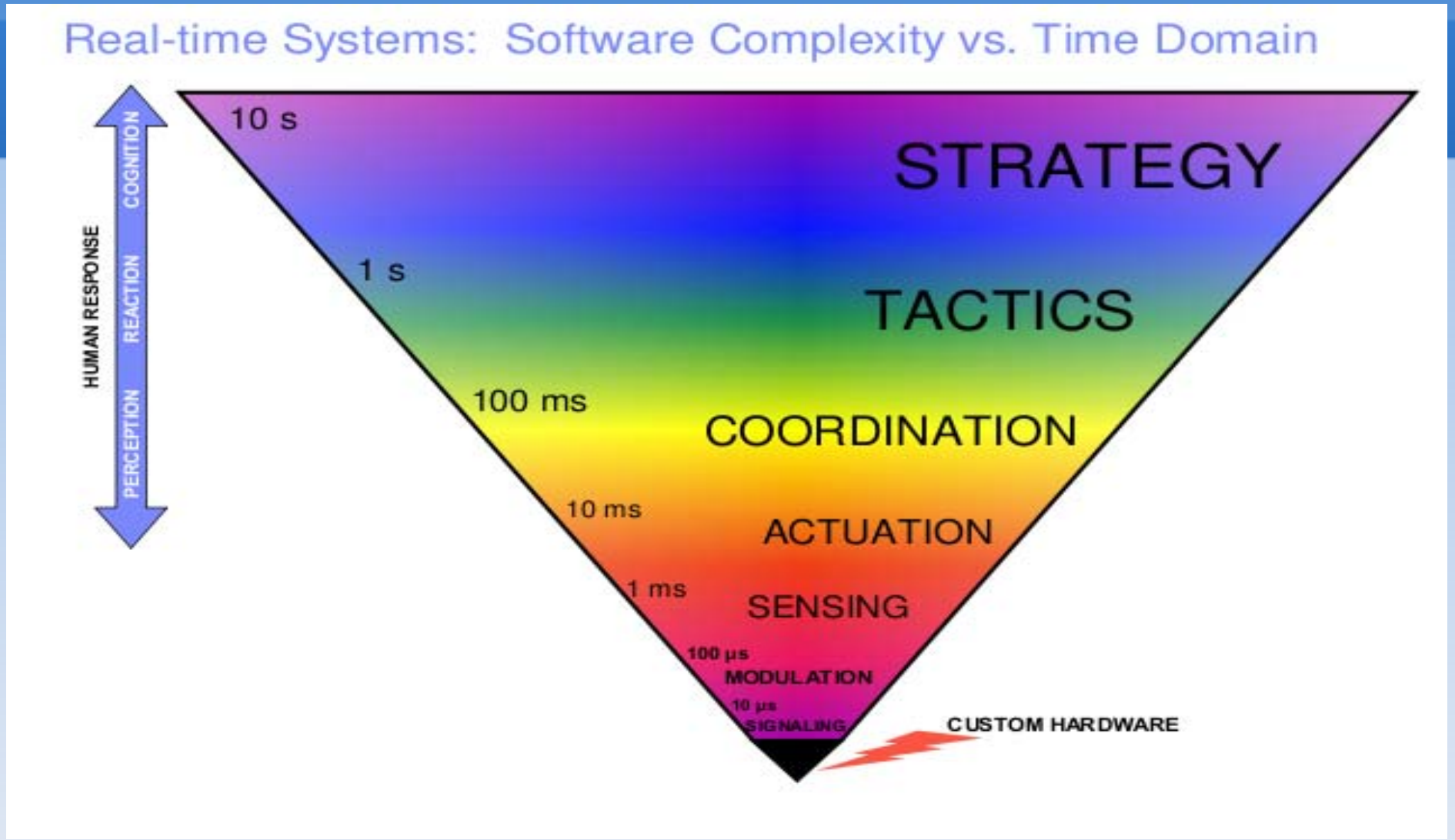
Deadlines can not be missed
Ex:fly-by-wire system, anti-lock brake system, motion control

Hard/Soft real-time

- Hard real-time
  - A late answer has no value
- Soft real-time
  - late answer has still a value but value of answer rapidly degrees



Software Complexity vs. Time



Taken From **IBM WebSphere Real Time: Providing predictable performance** by Michael S. Fulton, Java chief architect;Darren V. Hart, Real Time team lead; andGregory A. Porpora, IBM Software Group. December 2006

Latency and Jitter

- Latency is the time between external event and system’s response to that event
- Jitter is the variation in latency

Hardware Architecture & OS

- Modern processors and operating systems are optimized for throughput
  - It makes excellent sense for most systems to trade a rare factor of 100 times slowdown for a performance doubling everywhere else

Hardware & OS Cont.

- Worst-case scenario for an instruction
  - Instruction is not in the cache, processor must read it from memory
  - An address translation cache miss requires more memory access
  - Instruction might be in demand paged memory
  - Data may not be in cache
  - A large DMA may delay operations
  - SMI on X86 platforms
  - .....

Worst Case Execution Cont.

This example is taken from Real-Time Java Platform Programming by Peter C. Dibble

Event	Estimate Time (ns)
Execute Instruction	10
Instruction cache miss	50
Instruction ATC miss	500
Data cache miss	100
Dirty data cache write ATC miss	500
Data cache read ATC miss	500
Demand paging for instruction read (write dirty page)	20000000
Demand paging for dirty data cache write (read page)	10000000
Demand paging for data cache write (write dirty page)	20000000
Demand paing for data cache write (read page)	10000000
Demand paging for data cache read (write page)	20000000
Demand paging for data cache read (read page)	10000000
Interrupts	100000
One Big DMA	10000000
Total	**100101660**

How to prevent worst case

- Disable paging for time critical code
- Use processor affinity and pin RT Threads to cpus
- Use tunable DMA
- Use large pages to reduce load on TLB
- Disable interrupts or give RT Threads higher priority than some interrupts
- On x86 architecture pay attention to SMI (System Management Interrupts)
  - Do not disable it completely , you may burn down your processor.

Processor Pinning

- Less context switching jitter
- Decreased cache miss
- Example (Linux and Sun RTS only):
  - -XX:RTSJBindRTTToProcessors=0,1
  - -XX:RTSJBindNHRTToProcessors=0,1
  - Alternatively you can create a cpu set named xx and use /dev/cpuset/xx

Large Memory Pages

- Garbage collectors performs very bad if memory is swaped to disk
- For increased determinism use large memory pages and pin these pages to memory
- Example Linux and Sun's Java SE only:
  - echo shared_memory_in_bytes > /proc/sys/kernel/shmmax
  - echo number_of_large_pages > /proc/sys/vm/nr_hugepages
  - Start JVM using XX:+UseLargePages argument
  - Verify using cat /proc/meminfo | grep Huge

Refer following Sun article for large memory pages: Java Support for Large Memory Pages

RT-POSIX

- An extension to POSIX standard to address hard and soft real-time systems (POSIX 1003.1b)
  - Priority Inversion Control and Priority Inheritance
  - New schedulers
  - Asynchronous IO
  - Periodic, Aperiodic and Sporadic Threads
  - High Resolution Timers
  - RT File System
  - Some others ....

For further information refer to RT Posix standard

RT Linux

- Fully preemptible kernel
- Threaded interrupt handlers for reduced latency
- High-resolution timers
- Priority inheritance
- Robust mutexes and rt-mutexes
- To install use

  sudo apt-get install linux-rt

  in ubuntu. Select rt kernel at next system start-up

Why Java?

- Software (including embedded software) becomes more complex and gets unmanageable with old practices and tools
  - Ex:Financial systems, Network Centric systems
- Single language, tools for real-time and non-real time parts
- Java Platform provides a more productive environment
- A large set of 3rd party libraries are available

Has a very big community

  - Large number of developers
  - Support from big companies like IBM, SUN, Oracle
- A lot safer when compared to low level languages

Cons of Using C and Java Together

- Same functionality is codded twice
- High amount of integration problems
- Interfaces between C and Java is ugly and introduce overhead
- Communication via JNI can violate safe features of Java
- The JNI interface is inefficient
- Increased maintenance cost in the feature due to above problems

What are the problems of Java SE?

- Dynamic Class Loading and Linking
- JIT (Just in Time Compiler)
- Thread Handling
- Garbage Collector
- No Raw Memory Access
- No support for real-time operations, like deadline miss handling, periodic scheduling, processor pinning etc.

Dynamic Class Loading

- A Java-conformant JVM must delay loading a class until it's first referenced by a program
  - Early loading is not allowed so JVM can not do this for you at application startup
- This introduce unpredictable latency from a few microseconds to milliseconds.
  - Depends on from where classes are loaded
  - Static initialization
  - Number of classes loaded

JIT (Just In Time Compiler)

- Most modern JVMs initially interpret Java methods and, and later compile to native code.
- For a hard RT application, the inability to predict when the compilation will occur introduces too much nondeterminism to make it possible to plan the application's activities effectively.

Garbage Collection

- Pros
  - Pointer safety,
  - leak avoidance,
  - Fast memory allocation:faster than malloc and comparable to alloca
  - Possible de-fragmentation
- Cons
  - Unpredictable pauses:Depends on size of the heap, number of **live** objects on the heap and garbage collection algorithm, number of cpus and their speed

Main Garbage Collection Features

- Stop-the-world or Concurrent
- Moving objects
- Generational
- Parallel

Garbage Collection in HotSpot

- Serial Collector
  - -XX:+UseSerialGC
- Parallel-scavenging
  - -XX:+UseParallelGC
- Parallel compacting
  - -XX:+UseParallelOldGC
- Concurrent Mark and Sweep (CMS)
  - -XX:+UseConcMarkSweepGC

Garbage First Collector (G1)

- Planned for JDK 7
- Low pause and high throughput soft real-time collector
- It is parallel and concurrent
- Performs compaction
- Devides heap to regions and further devides them to 512 bytes cards
- It is not a hard real-time collector

Thread Management

- Although standard Java allows priority assignment to threads, does not require low priority threads to be scheduled before high priority ones
- Asynchronously interrupted exceptions
  - Similar to Thread.stop but this version is safe
- Priority Inversion may occur in standard Java/Operating systems
  - RTSJ requires Priority Inheritance

What is RTSJ?

- Designed to support both hard real-time and soft real-time applications
- Spec is submitted jointly by IBM and Sun
- Spec is first approved in 2002 (JSR 1)
- Minor updates started on 2005 (JSR 282)
- First requirements are developed by The National Institute of Standards and Technology (NIST) real-time Java requirements group

Expert Group of RTSJ

Specification Lead

Peter Dibble TimeSys Corporation

Expert Group

Ajile Systems	Apogee Software, Inc.	Belliardi, Rudy
Brosgol, Benjamin	Cyberonix	MITRE Corporation
Motorola	Nortel	NSI COM
QNX	Rockwell Collins	Sun Microsystems, Inc.
Thales Group	TimeSys Corporation	WindRiver Systems

RTSJ's Problem Domain

- If price of processor or memory is a small part of whole system RTSJ is a good choice
  - If processor and memory price is very significant when compared with the rest of the system RTSJ may not be a good choice because you will need slightly more powerful processor and larger memory
  - For very low footprint applications with very limited resources you may consider a non RTSJ compliant virtual machine like Aonix Perc Pico or you may even consider a hardware JVM.

Implementations

- IBM Webshere Real Time (RTSJ Compliant)
  - Works on real-time linux kernel
  - There is a soft real-time version
- SUN RTS (RTSJ Compliant)
  - Works on Solaris x86/Sparc or real-time linux kernel
- Aonix Perc (Not RTSJ Compliant but close)
  - Perc-Ultra works on nearly all platforms
  - They have a safety critical JVM (Perc-Raven)
- Apogee
  - Woks on nearly all platforms

Main Features of RT Java Implementations

Full Java SE compatibility and Java syntax

A way to write programs that do not need garbage collection (New API)

High resolution timer (New API)

Thread priorities and locking (New API)

Asynchronous Event Handers (New API)

Direct memory access (New API)

Asynchronous Transfer of Control (New API)

AOT (A Head of Time) compilation (Not in RTSJ specification)

Garbage collection (Not in RTSJ specification)

Real-Time Garbage Collectors

- Time based (Metronome of Websphere RT)
- Work based
- Henriksson’s GC (Garbage Collector of Sun’s Java RTS is based on this)

Sun’s RT Garbage Collector

- 3 modes of execution
- Non generational, concurrent and parallel collector
- In normal mode works with priority higher than non-rt threads but lower than any rt thread.
- When free memory goes below a certain threshold priority is increased to boosted priority but RT Threads (including the one with priority lower than GC) still continues to work concurently
- When free memory goes below a critical threshold.All threads whose priority is lower than GC boosted priority are suspended
- Most of its working can be tuned using command line switches like: NormalMinFreeBytes, RTGCNormalWorkers, RTGCBoostedPriority, BoostedMinFreeBytes etc.

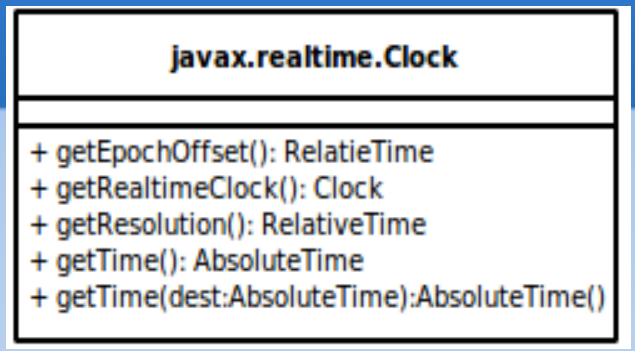
AOT, ITC and JIT

- Nearly all Java SE vendors use JIT (Just in time compiler) due to performance reasons
- Websphere RT uses AOT (Ahead of Time Compilation)
- SUN’s Java RTS uses ITC (Initialization Time Compilation)

Refer following IBM Article for further information: Real-time Java, Part 2: Comparing compilation techniques

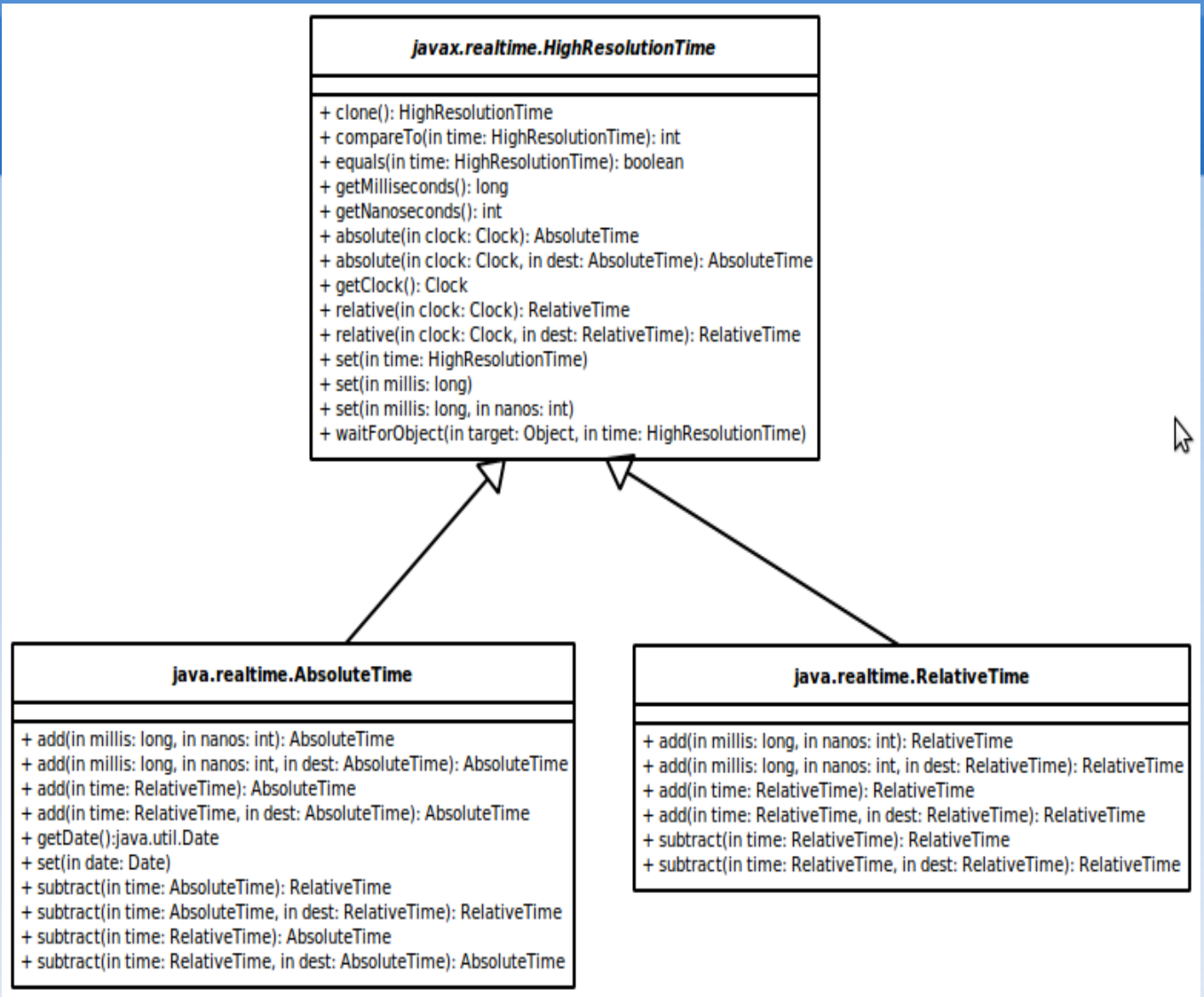
High-Resolution Timer

- A more powerful notion of time
  - AbsoluteTime
  - RelativeTime
- 84 bits value
  - 64 bits milliseconds and 32 bits nanoseconds
  - Range is 292 million years
- Every RTSJ implementation should have provide at least one high resolution clock accessible with *Clock.getRealTimeClock()*
- Resolution is system dependent
  - For example it is 200ns for SunFire v240 (Ref:Real-Time Java Programming by Greg Bollella)
  - Do not expect too much from a laptop



HighResolutionTime Base Class

- Base class for other 3 time classes.
- Define the interface and provides implementation of some methods
- This class and its subclass do not provide any synchronization



AbsoluteTime and RelativeTime

- AbsoluteTime represents a specific point in time
  - The string 03.12.2008 11:00:00.000 AM represents an absolute time
  - Relative to 00:00:00.000 01.01.1970 GMT
- RelativeTime represents a time interval
  - Usually used to specify a period for a schedulable object
  - Can be positive, negative or zero

Priority Scheduler

- Normal scheduling algorithms try to be fair for task scheduling.
  - Even lower priority tasks are scheduled some times
- There are different real-time and non real-time schedulers
  - Earliest-deadline first, least slack time, latest release time etc.
- Fixed priority preemptive scheduler is most used real-time scheduler

Thread Scheduling in RTSJ

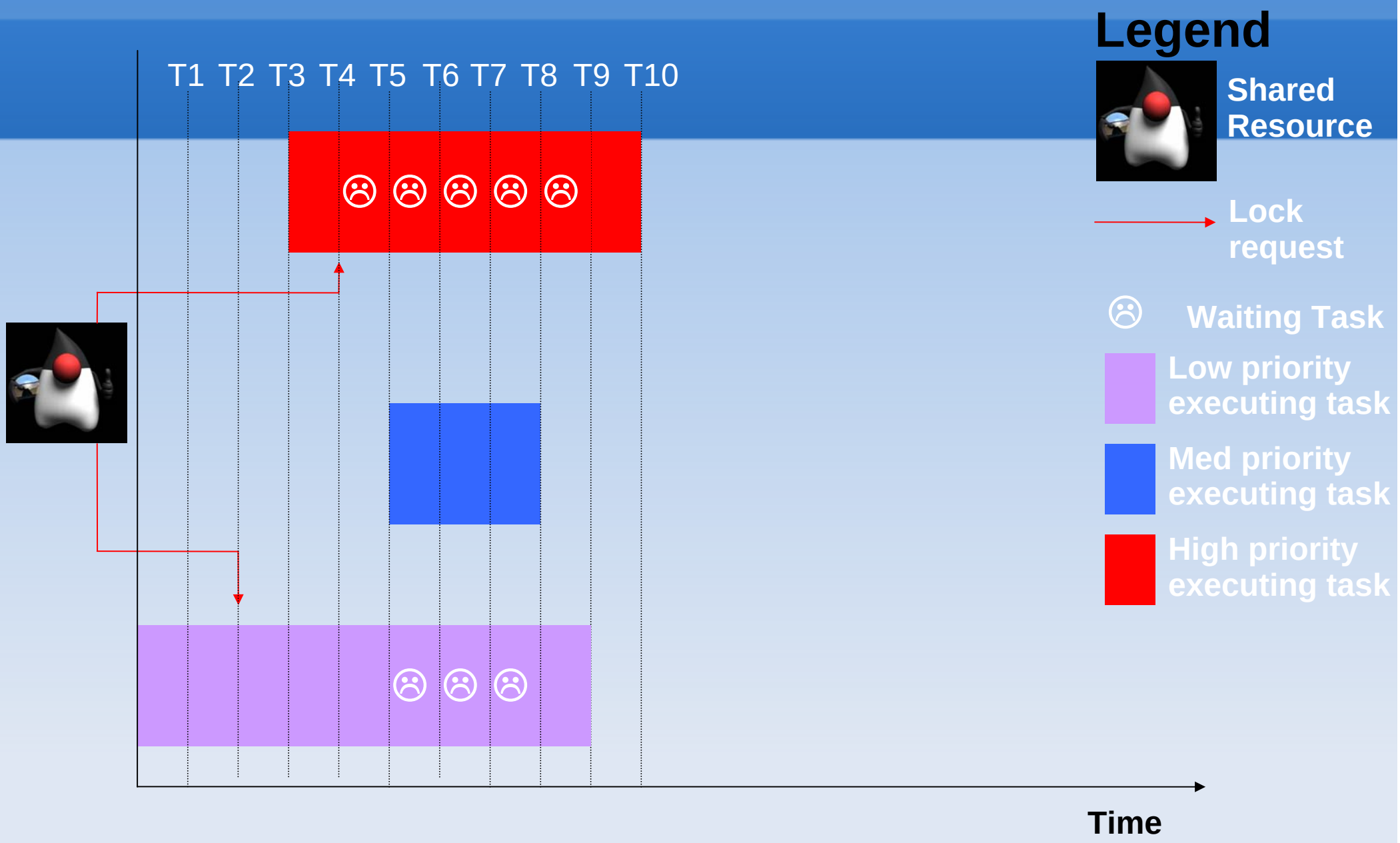
- Does not specify a scheduling implementation
- Allow different schedulers to be plugged
- Required base scheduler should be a priority scheduler with at least 28 unique priorities
- Most implementations provide more than 28 priorities

More on this with Scheduling Parameters Slide

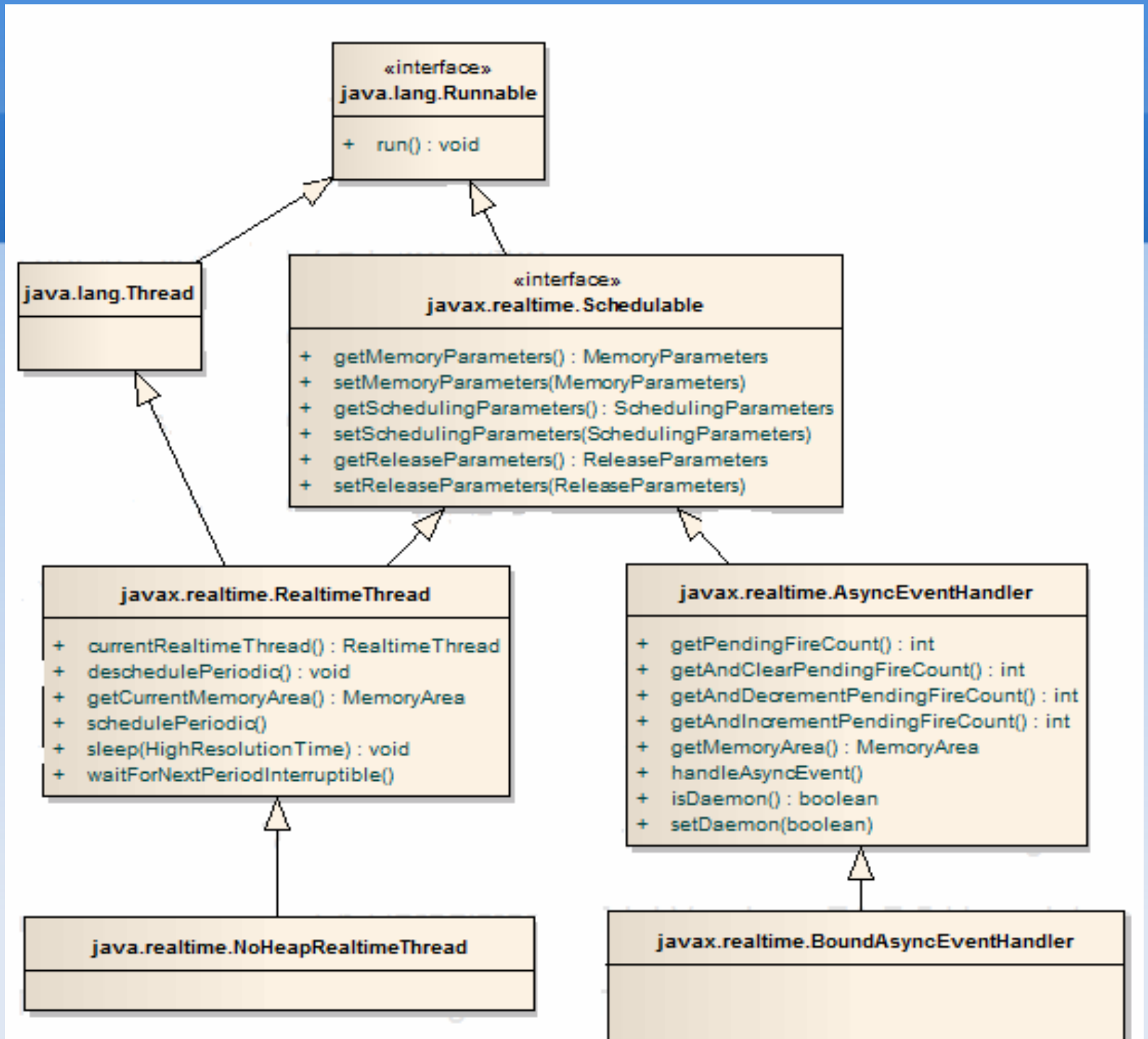
What is Priority Inversion?

- Priority Inversion is the situation where a higher priority task waits for a low priority task.
- There are 2 generally accepted solution
  - Priority Inheritance (required by RTSJ)
  - Priority Ceiling Emulation (optional for RTSJ)

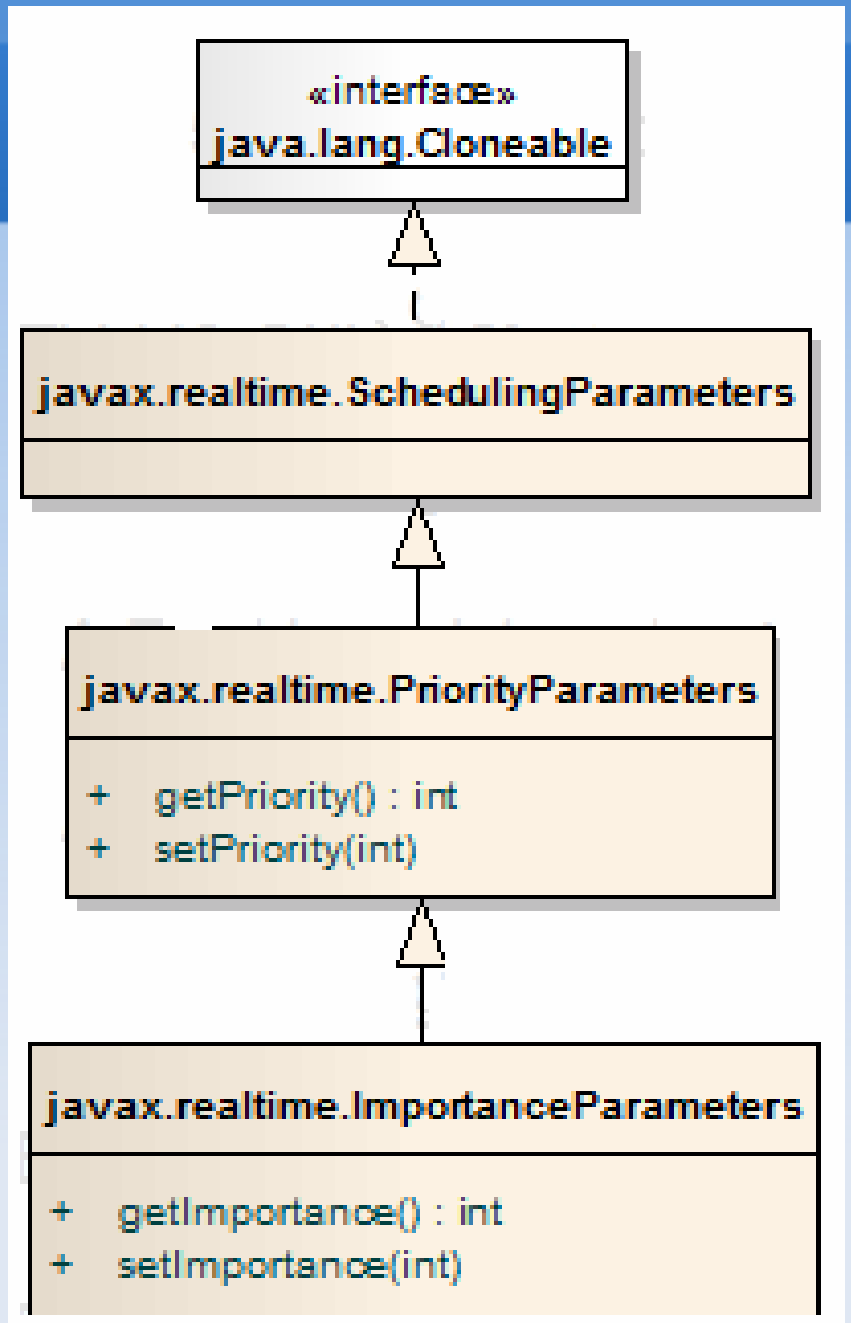
What is Priority Inversion? cont.



Schedulable Interface



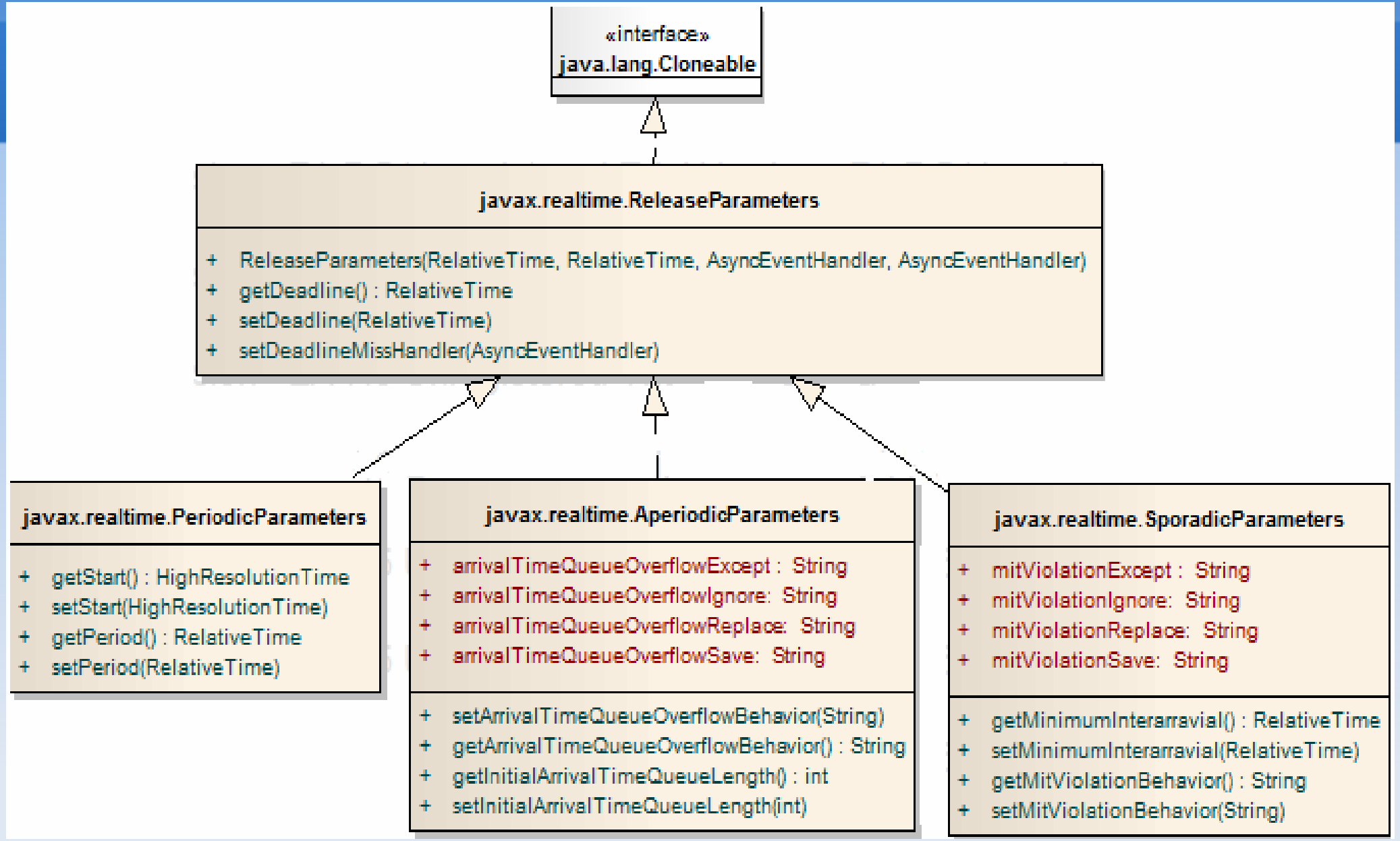
Scheduling Parameters



Different Task Types

- Periodic
- Aperiodic Tasks
- Sporadic

Release Parameters



Sample Periodic Task

```
public class HelloWorld extends RealtimeThread {
    public static long[] times = (long[]) ImmortalMemory.instance().newArray(long.class, 100);

    public HelloWorld(PeriodicParameters pp) {
        super(null, pp);
    }

    public void run() {
        for (int i = 0; i < 100; i++) {
            times[i] = System.currentTimeMillis();
            waitForNextPeriod(); //wait for next period
        }
    }

    public static void main(String[] argv) {
        //schedule real time thread at every 100 milisecond
        PeriodicParameters pp = new PeriodicParameters(new RelativeTime(100, 0));
        HelloWorld rtt = new HelloWorld(pp);
        rtt.start();

        //wait real time thread to terminate
        try {
            rtt.join();
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }

        //print results
        for (int i = 0; i < 100; i++) {
            System.out.println(times[i]);
        }
    }
}
```

Periodic Execution vs Thread.sleep

```
While (true) {
   performPeriodicTask()
   Thread.sleep(period)
}
```

WRONG

Real-time systems do not use a loop with a sleep in it to drive periodic executions

Deadline Miss!

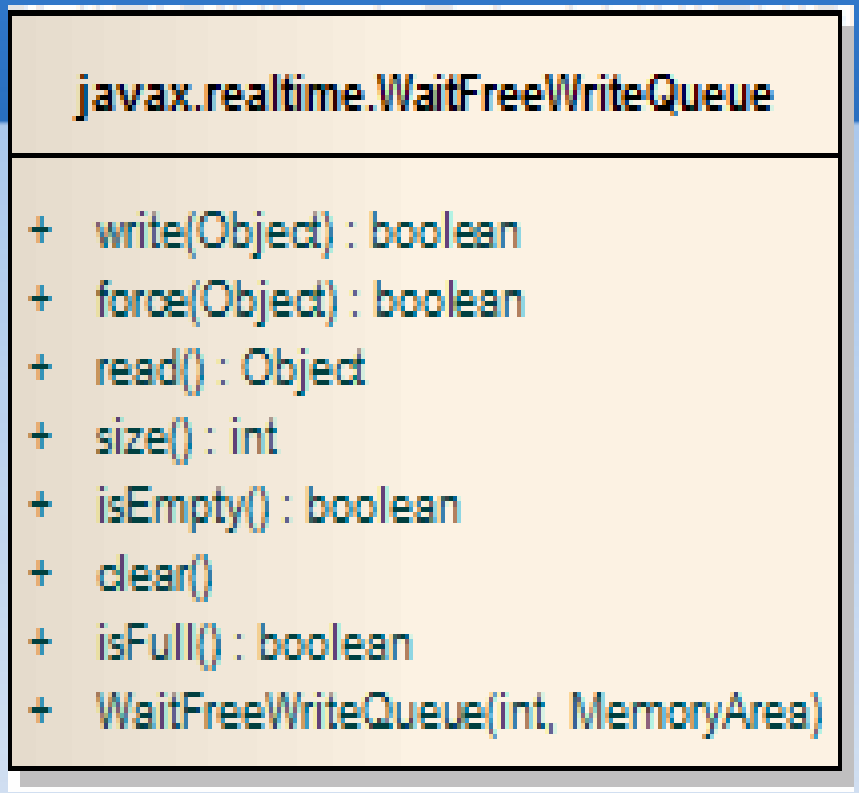
- For some applications a deadline should not be missed for any reason
- For most applications dealine miss is bad, but it is not a total failure and a recovery is possible
- In RTSJ there are 2 ways to detect a deadline miss
  - waitForNextPeriod() returns false **immediately**
  - Use a deadline miss handler which is an instance of AEH

NoHeapRealTime Thread

- Can not access heap
  - Can preempt GC immediately
  - If this rule is violated, MemoryAccessError is thrown
- Can use ScopedMemory or ImmortalMemory
- Should be created and started in a scoped or immortal memory
  - They can not be created from normal Threads

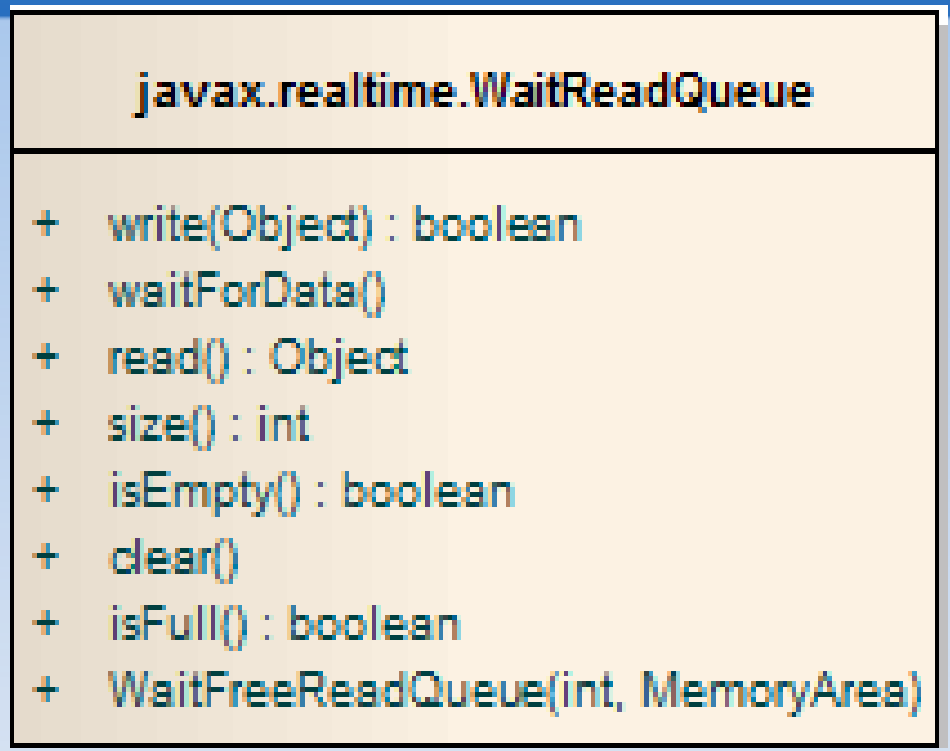
WaitFreeWriteQueue

- Intendent for exchanging data between real-time and non real-time part
- Real-time producer does not block when queueing data
- Multiple non-real time consumers may dequeue data



WaitFreeReadQueue

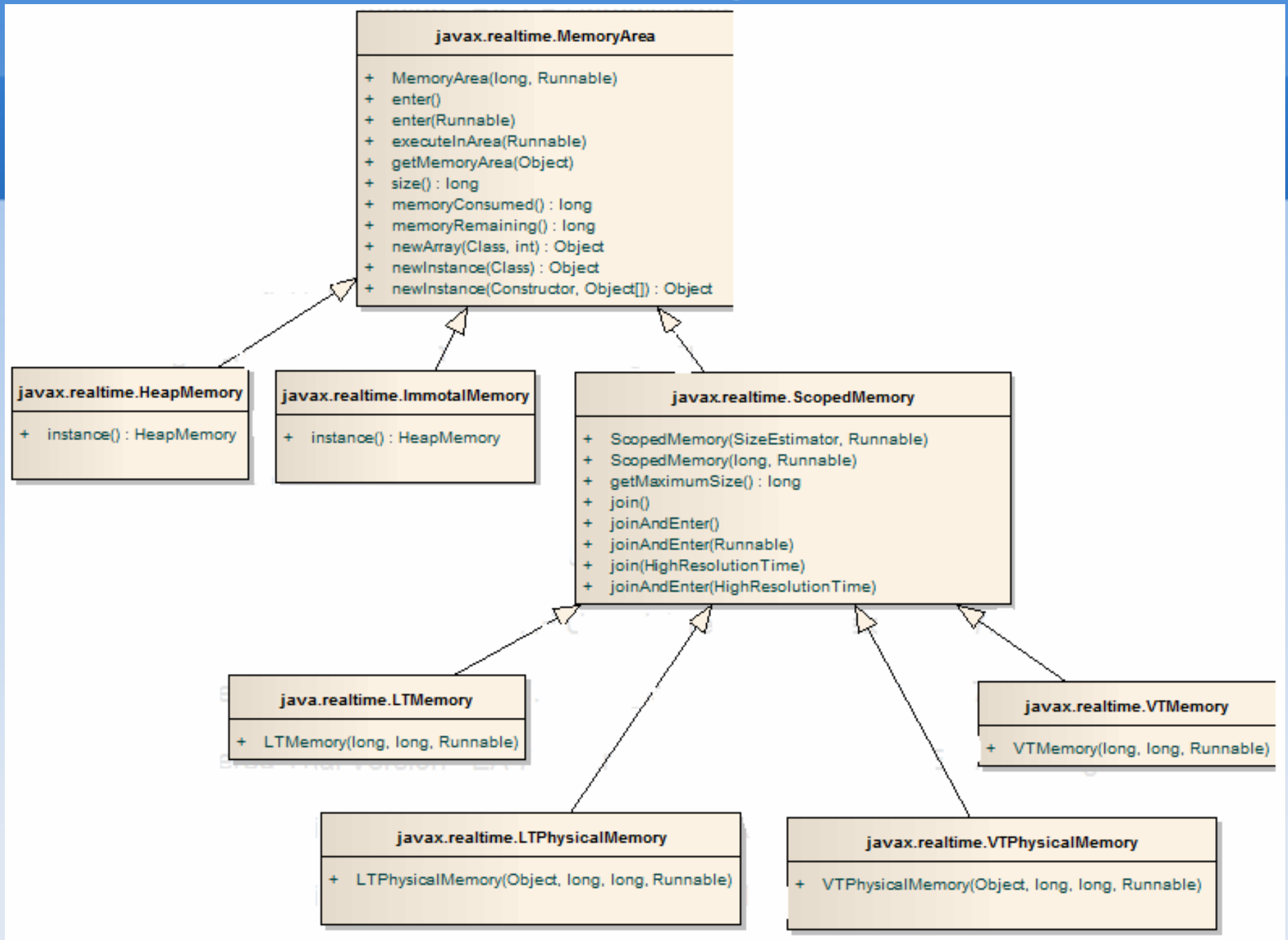
- Intendent for exchanging data between real-time and non real-time part
- Real-time consumer does not block when reading data
- Multiple non-real time producers may queue data



Memory Regions

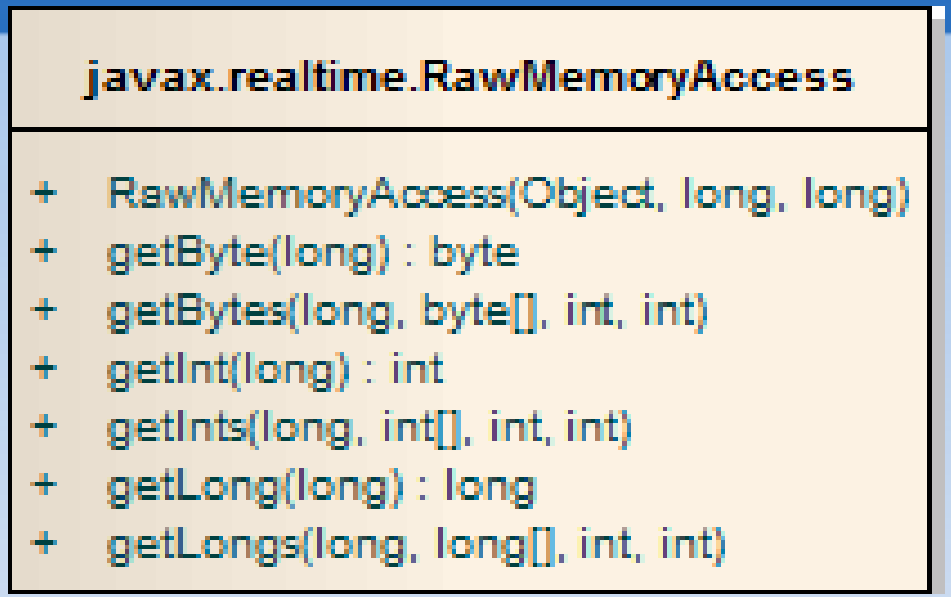
- Heap Memory
  - Same as in Java SE
  - Can be accessed with javax.realtime.HeapMemory
- Scoped Memory
  - Created and sized at development time
  - Can be accessed with javax.realtime.ScopedMemory
  - Can not be garbage collected; reference count to ScopedMemory object is used. Finalize method of objects are called
  - Can be stacked
- Immortal Memory
  - Can be accessed with javax.realtime.ImmortalMemory
  - Only one instance exist and size is determined at development time.
  - All static data and allocations performed from static initializers are allocated in Immortal memory. Interned Strings are also allocated in immortal memory
- Physical Memory
  - There are LTPhysicalMemory, VTPhysicalMemory, and ImmortalPhysicalMemory

Memory Regions



Raw Memory Access

- Models a range of physical memory as a fixed sequence of bytes
- Allows device drivers to be writen in java
- All raw memory access is treated as volatile, and *serialized*



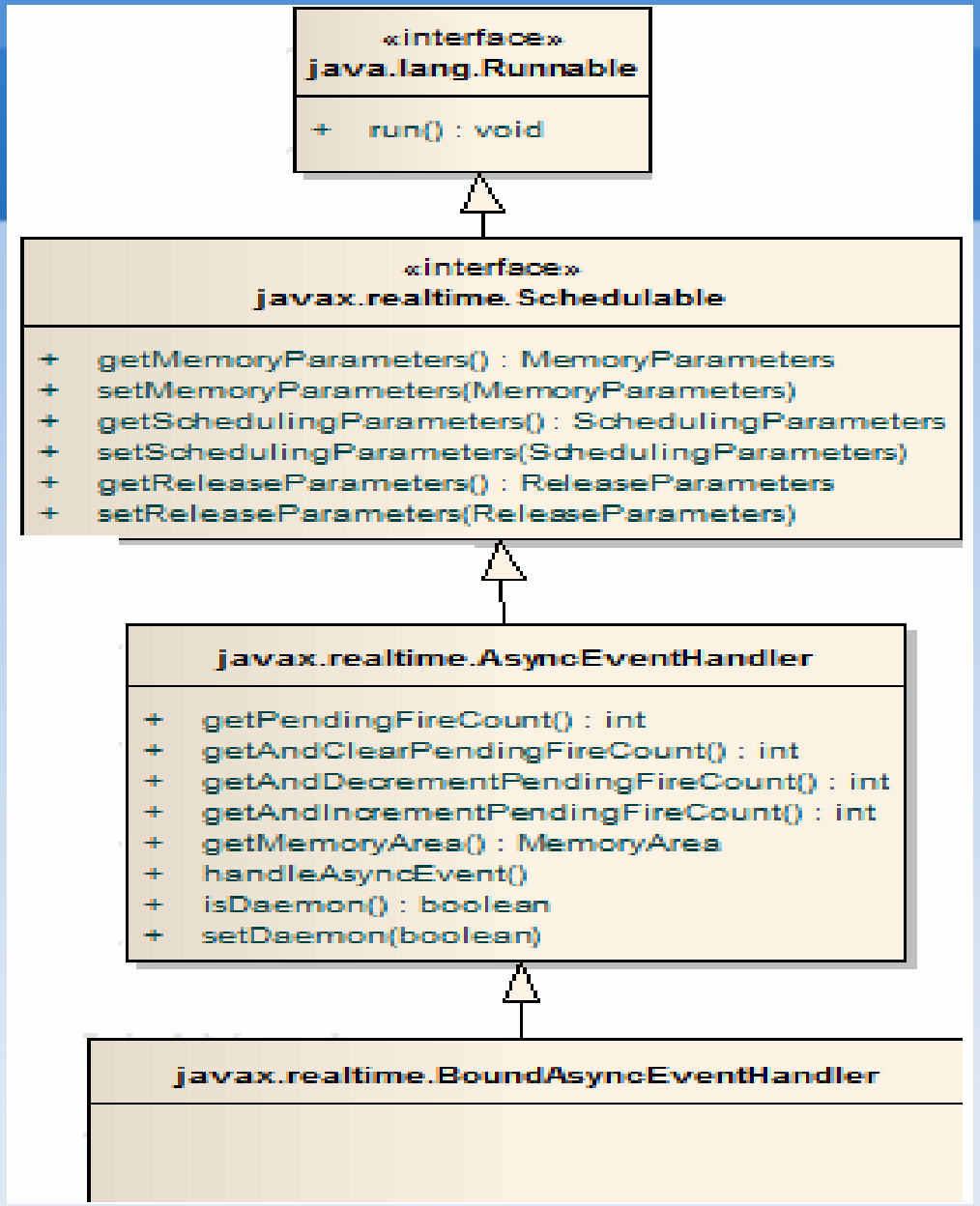
Raw Memory Access Cont

```
private final long DEVICE_BASE_ADDRESS = xxxxxx;
private final long CTRLREG = 0;
private final long STATREG = 4;
……..

public void init() {
    RawMemoryAccess device = new RawMemoryAccess(type, DEVICE_BASE_ADDRESS);
    device.setInt(CTRL_REG, MY_COMMAND); //send command to device
    while (device.getInt(STATREG) != 0); //wait for device to response
}
```

Async Events

- Many of the processing in RT systems are triggered by internal or external events
- You don’t need to manage threads
- Hundreds of AEHs may share a pool of threads
- BoundAsyncEventHandler has always bounded to a dedicated thread



Handling Posix Events

- Use POSIXSignalHandler
  - Ex:

```
.....

class SigintHander extends AsynchEventHandler {
    public SigintHandler() {
        //set it to highest priority
        setSchedulingParameters(new PriorityParameters(RTSJ_MAX_PRI);
    }
    public void handleAsynchEvent() {
        //handle user specified signal
    }
}

......
//add handler to posix predefined posix signal
POSIXSignalHandler.addHandler(PosixSignalHandler.SIGUSR1, sigintHandler)


Use kill -s SIGUSR1 PID to test
```

Time Triggered Events

- **OneShotTimer :** execute handleAsyncEvent method once at the specified time
- **PeriodicTimer :** execute handleAsyncEvent method repeatedly at specified interval. A periodicTimer and a AEH combination is roughly equivalent to Periodic Threads.
- **Enable/Disable Timer :** A disabled timer is still kicking. But it does not generate events. When enabled again, it continues like never disabled.

Javolution Library (http://javolution.org)

- High performance and time deterministic (util/lang/text/io/xml)
- Struct and Union base classes for direct interfacing with native applications
- NHRT Safe
- Pure Java and less than 300Kbytes
- BSD License

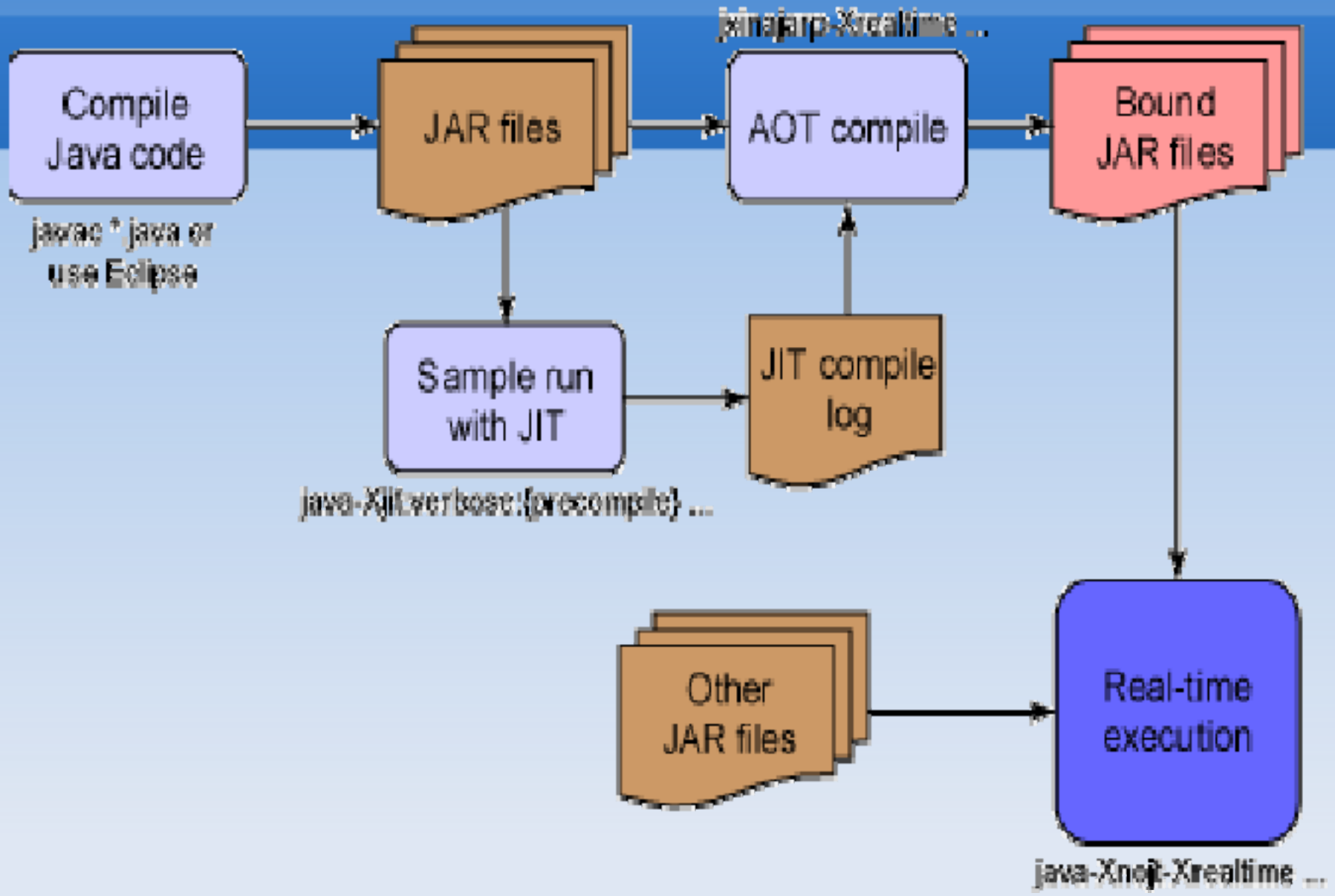
IBM WebSphere Real Time

- Runs on linux with real time patches applied to linux kernel on x86 platforms
- Has AOT and JIT Compilers
- Shared classes support
- Contains Metronome: a real-time garbage collector
- Full RTSJ 1.0.2 and Java SE 6.0 support
- A well defined list of NHRT safe classes

Metronome Garbage Collector

- Uses a time based method for scheduling
- Applications threads are given a minimum percentage of time (utilization)
  - User supplied at startup
- Uses a new two-level object model for arrays called arraylets

Websphere AOT

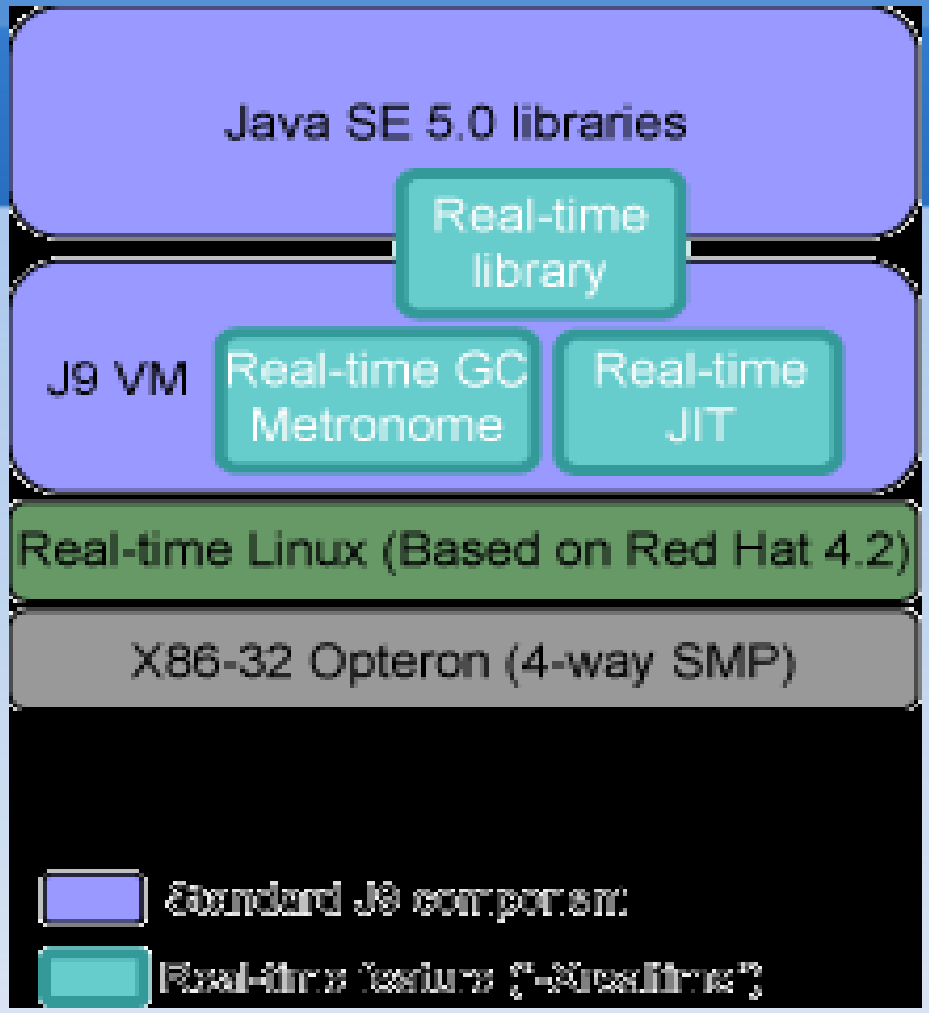


SUN RTS

- Achieves maximum latencies of 15 microseconds, with around 5 microseconds of jitter.
- Runs on real-time linux and Solaris 10
- Has a real-time garbage collector

IBM WebSphere Real Time

- Runs on linux with real time patches applied to linux kernel
- Has AOT and JIT Compilers
- Contains Metronome real-time garbage collector
- RTSJ and Java SE 5.0 Compliant



Real World Projects:J-UCAS X-45C

Figure 1

The mission planning software for the J-UCAS X-45C unmanned aircraft was written in the Java language and deployed on a real-time virtual machine. The code was developed as a collaboration between engineers at Boeing and British Aerospace Engineering (BAE).

Real World Projects:FELIN

Figure 2

The FELIN project is a helmet-mounted personal digital assistant to help infantry communicate with each other and with commanders, and to provide navigation and situational awareness aids. This application software, developed in France by Sagem, was written in Java and deployed on a real-time virtual machine.

Real World Projects:DDG-1000

Figure 1

For the DD(X) destroyer program—now called DDG-1000—software developers have pushed the envelope of Java real-time garbage collection technology. Release 4 of the DD(X) software environment will switch completely to a real-time Java VM.

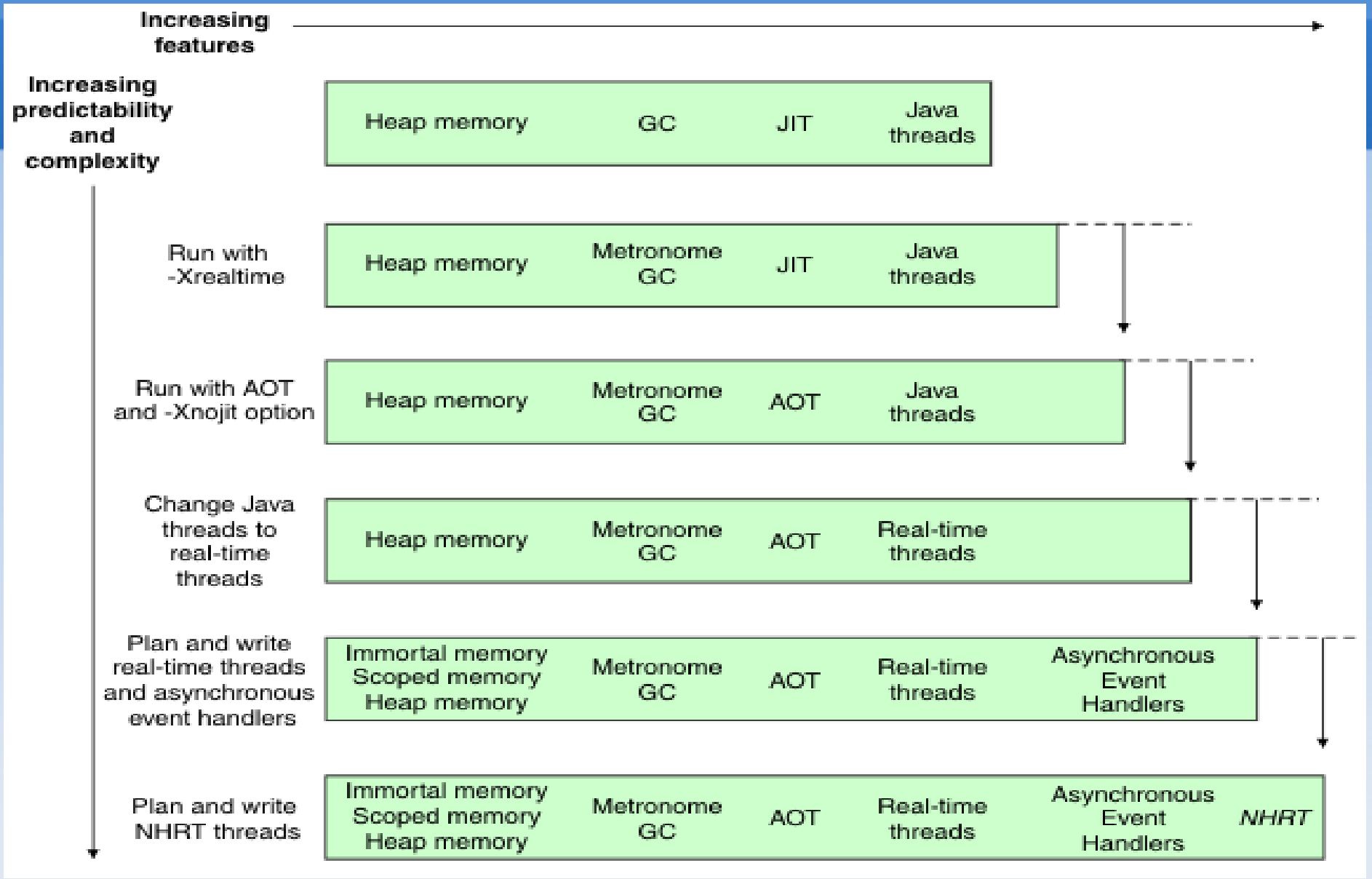
Real World Projects:ScanEagle

This milestone marked the first flight using the RTSJ on an UAV and received the Java 2005 Duke's Choice Award for innovation in Java technology.

Is RTSJ Required for You?

- If your application can tolerate some degree of indeterminism use standard JVM and tune it to milliseconds level
- Only if you fail the first approach use RTSJ.
  - Try to meet your timing constrains with real-time garbage collector (if available) without using advance/complex features like scoped memory
  - If first approach fails make use of NonHeapRealTimeThread, ImmortalMemory, ScopedMemory

A comparison of the features of RTSJ with the increased predictability



From IBM WebSphere Real Time Manual

Safety Critical Java (JSR 302)

- A subset of RTSJ that can be certified DO-178B and ED-128B.
- Most probably garbage collector will not be available (not needed)
- Will be targeted to Java ME platform, because Java SE and Java EE are too complex to be certified.

Expert Group of JSR 302

- **Specification Lead:**C. Douglass Locke (POSIX 1003.4 Real−Time Extensions Working Group)
- Expert Group

Aicas GmbH	Aonix North America, Inc	Apogee Software, Inc.
AXE, Inc	Boeing	DDC-I, Inc
IBM	Siemens AG	Rockwell Collins, Inc
	Sun Microsystems	The Open Group

Resources

- Real-Time Java Platform Programming by Peter C. Dibble
- Real-Time Java Programming with Java RTS by Greg Bollea
- RTSJ Main Site (www.rtsj.org)
- IBM's Developerworks Articles (http://www.ibm.com/developerworks/)
- SUN RTS (http://java.sun.com/javase/technologies/realtime/reference.jsp)
- Deniz Oğuz's blog (www.denizoguz.com)

Resources Cont. (JavaOne Presentations in 2008)

- TS-4797 Fully Time-Deterministic Java Technology
  - Explains Javalution Library
- TS-5767 Real-Time Specification for Java (JSR 1)
  - Explains status of RTSJ. Join presentation from SUN, IBM, Locke Consulting (JSR 302 spec lead)
- TS-5925 A City-Driving Robotic Car Named Tommy Jr.
  - An autonomous ground vehicle fully powered by Java.
- TS-5609 Real Time: Understanding the Trade-offs Between Determinism and Throughput