

Empirical Analysis of Real-Time Java Performance and Predictability

Angelo Corsaro and Douglas C. Schmidt

{corsaro, schmidt}@ece.uci.edu

Electrical and Computer Engineering Dept.

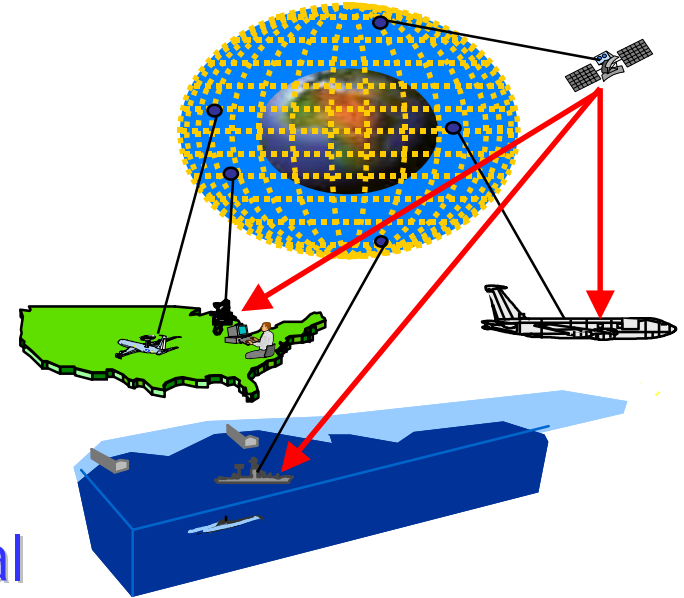
University of California, Irvine

(Work Supported by Siemens MED, SAIC, ATD)



Assessing Real-Time Java Effectiveness

- Two dimension should be considered when assessing the effectiveness of the RTSJ as a technology for embedded real-time Systems
 - Quality of the RTSJ API
 - Quality of RTSJ Implementation
- Here, we focus on the implementation quality aspect, and provide an evaluation for most of the RTSJ feature critical in Embedded Systems.

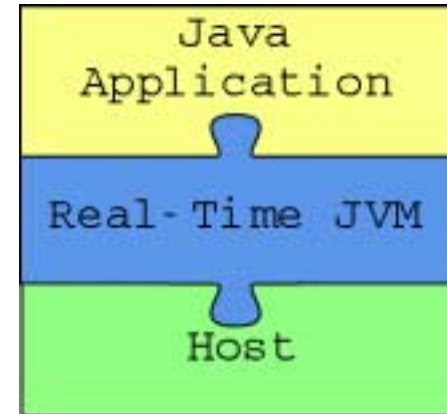


RTJPerf

- RTJPerf is a benchmarking suite for RTSJ compliant platforms that focuses on *Time-efficiency* performance indexes.
- RTJPerf provides a series of tests based on *synthetic workload*
- RTJPerf currently covers the following RTSJ areas:
 - Memory
 - Threading
 - Asynchrony
 - Timers

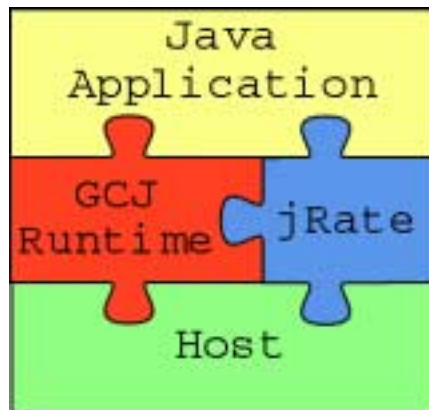
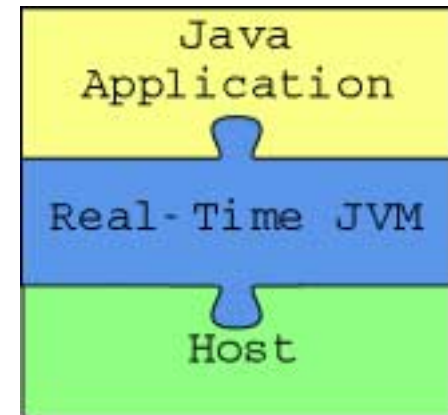
Tested Platforms

- RTSJ Reference Implementation, developed by TimeSys



Tested Platforms

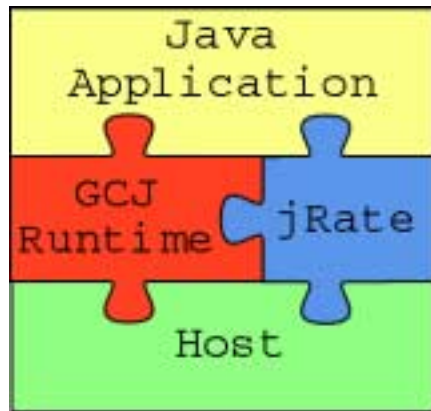
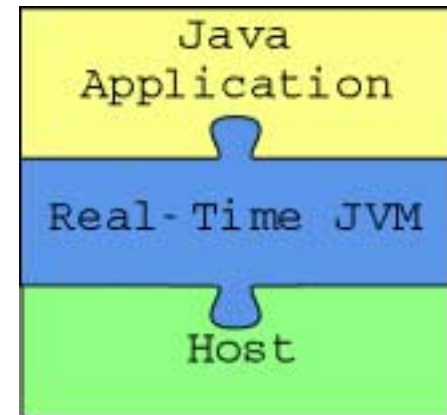
- RTSJ Reference Implementation, developed by TimeSys



- jRate, an Open Source RTSJ-based extension of the GCJ runtime system

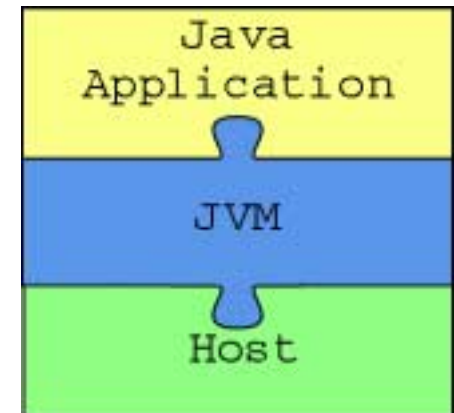
Tested Platforms

- RTSJ Reference Implementation, developed by TimeSys



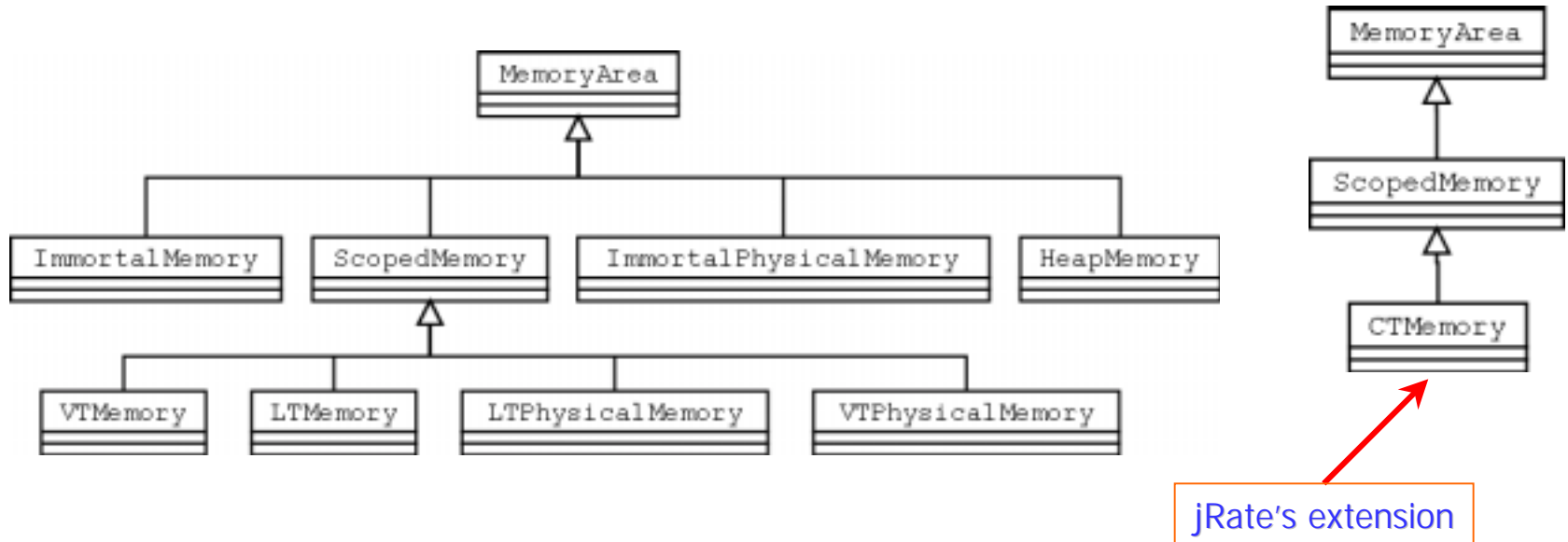
- jRate, an Open Source RTSJ-based extension of the GCJ runtime system

- The C Virtual Machine (CVM), a J2ME platform
- JDK 1.4, the latest Sun's Java Development Kit JVM



RTSJ Memory

- The RTSJ extends the Java Memory model by introducing the concept of memory areas
 - Memory areas provide different characteristic in term of the allocation time and of the lifetime of the object allocated within it
 - Memory Scopes are particular kind of memory areas that provide guarantees on the allocation time characteristics
- RTJPerf provide a test that allows to measure the time efficiency of scoped memory implementation



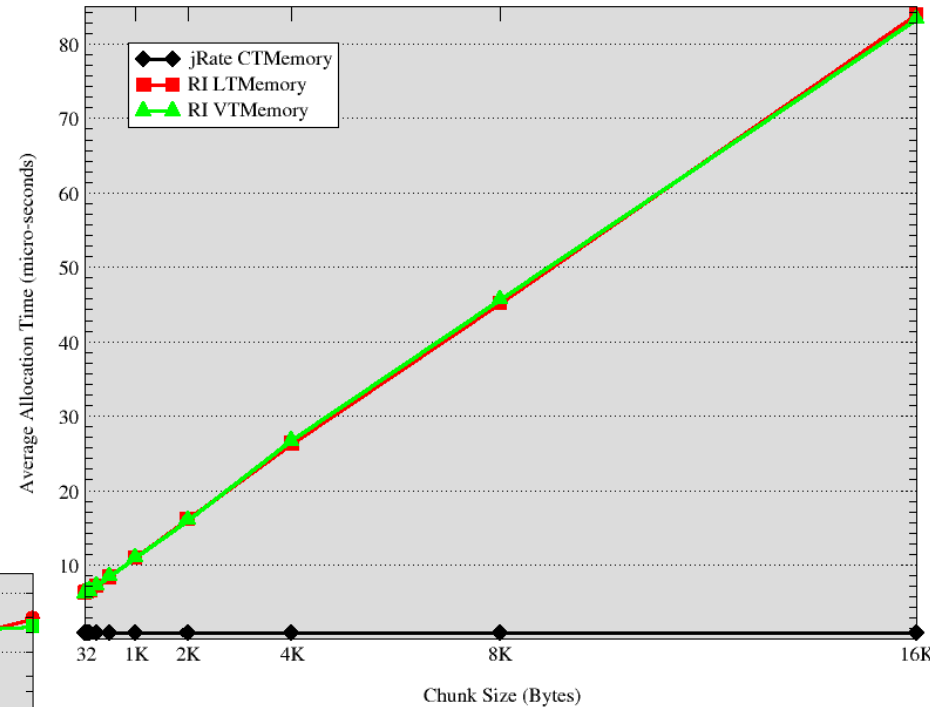
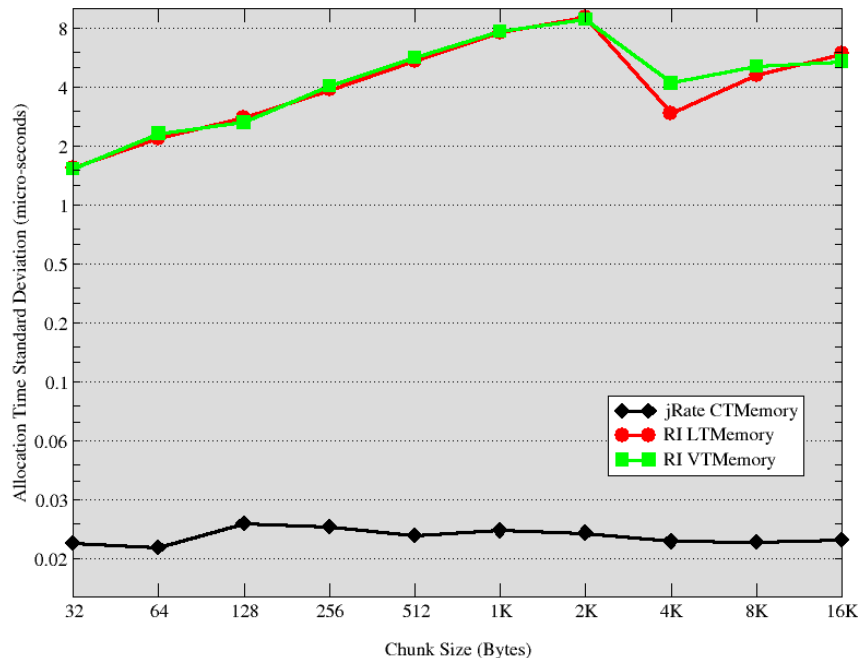
Allocation Time Test

- Objective:
 - Determine the allocation time for different kind of ScopedMemory provided by the RTSJ
- Technique:
 - Allocate vector of characters for different sizes ranging from 32 to 16K bytes.
- Tested Platform:
 - RTSJ RI
 - jRate
- Test Parameters:
 - 1,000 Sample for each *chunk size* were collected

Average & Dispersion Measures

Memory: Allocation Time

- RI's LT/VTMemory allocation time grows linearly with the size of the chunk being allocated
- RI's LTMemory & VTMemory are statistically equivalent
- jRate's CTMemory provides constant allocation time

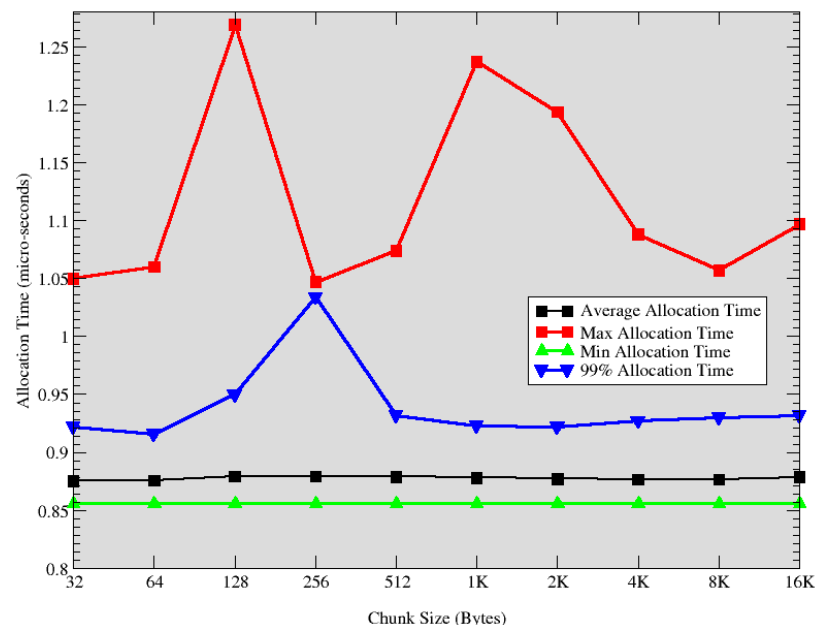
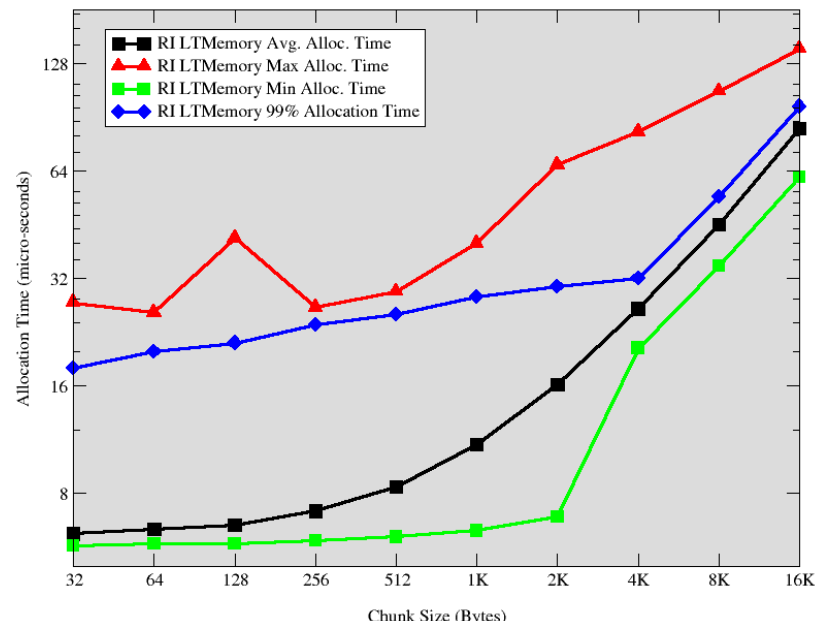
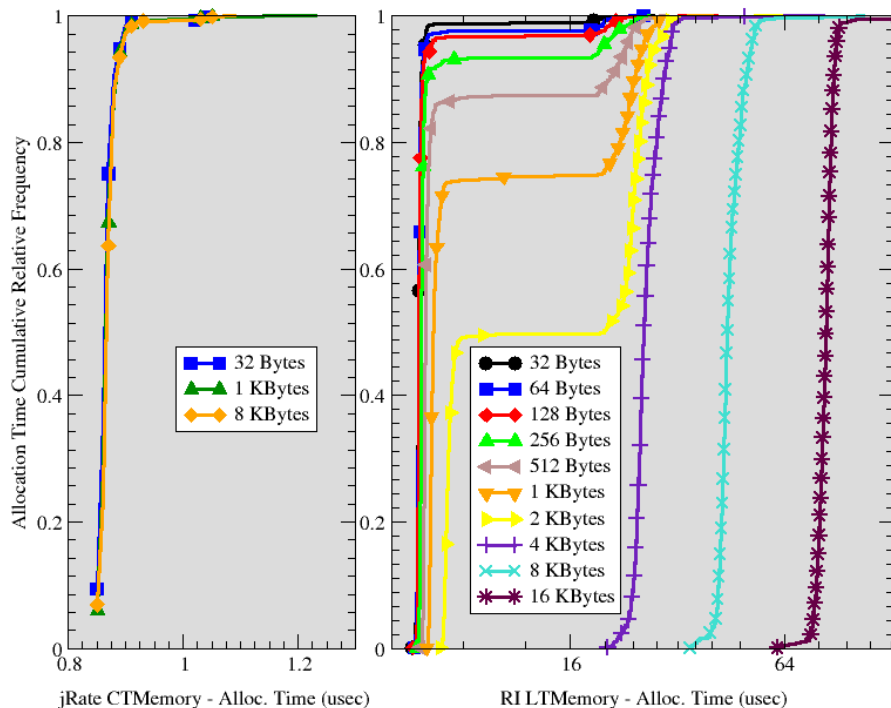


- RI's LT/VTMemory behaves less predictably as the chunk size increases *i.e.* allocation time gets more dispersed
- jRate's CTMemory predictability is not influenced by the chunk size

Worst-Case Measures

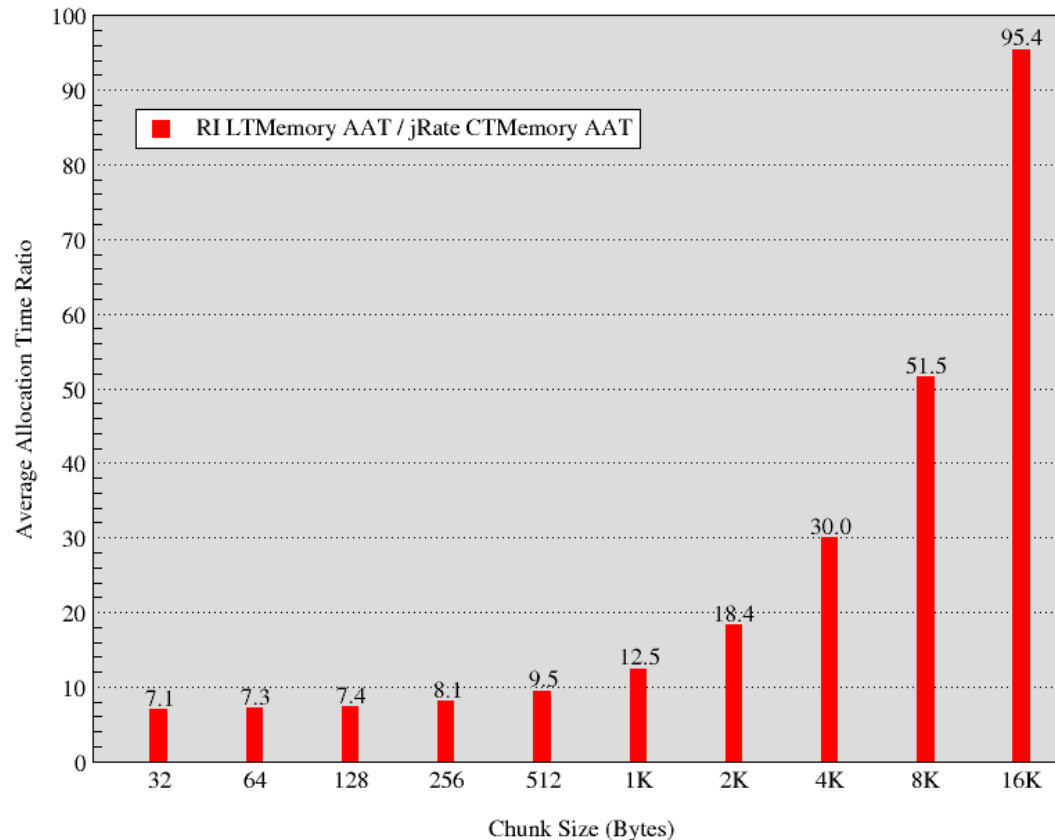
- RI's LT/VTMemory worst case, 99% and average case behaviour converge as the chunk size increases
- jRate's CTMemory worst case performances are quite close to the average and the 99% case

Memory: Allocation Time



CTMemory vs. LTMemory

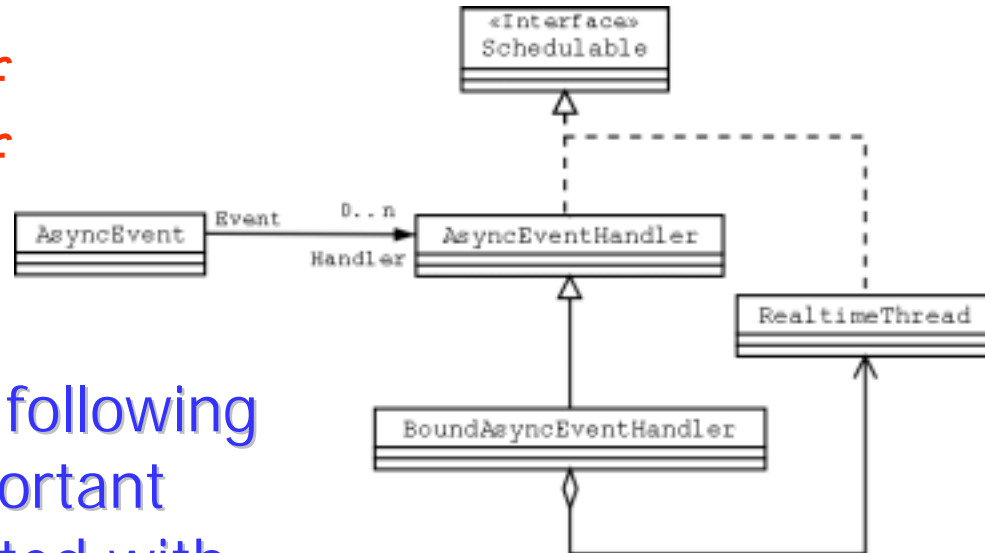
- As the size of the chunk allocated increases the speedup of jRate's CTMemory over the RI LTMemory can be as much as 95.
- The CTMemory speedup over the LTMemory grows linearly with the size of the chunk being allocated



RTSJ Asynchrony

- The RTSJ provides a way of binding the occurrence of internal and/or external event to handler, by using:

- **AsyncEventHandler**
- **BoundEventHandler**



- RTJPerf provides the following tests to measure important characteristic associated with Event Handlers:

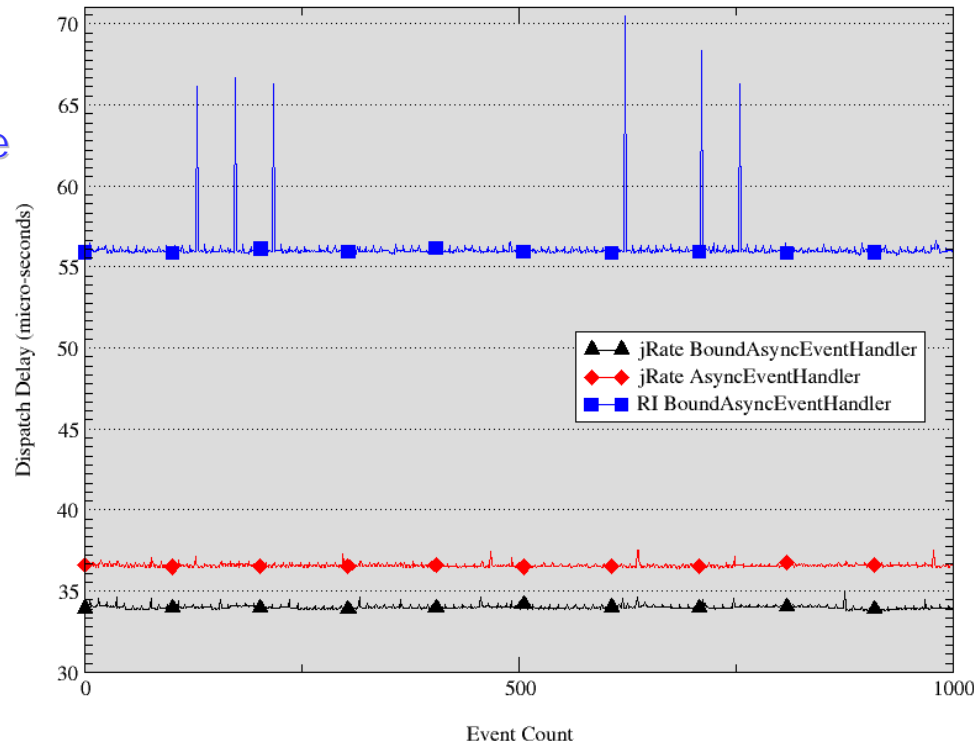
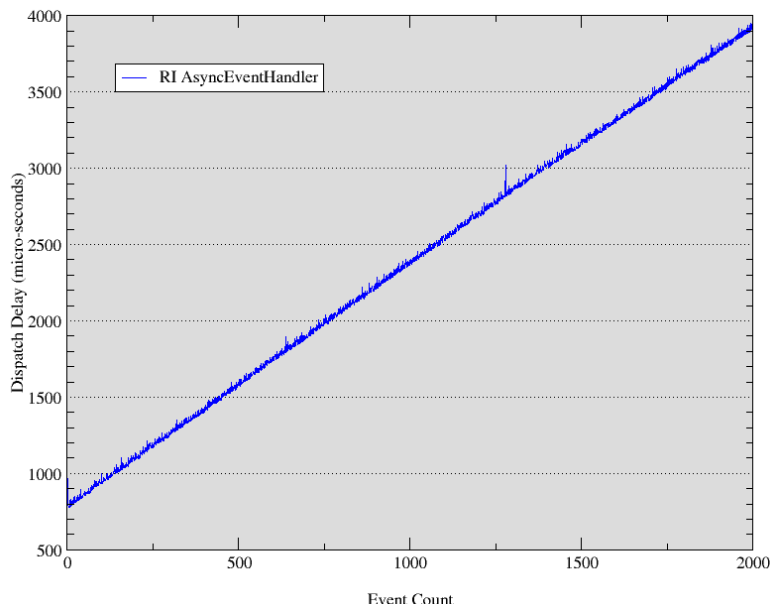
- Dispatch Delay Latency
- Dispatch Priority Inversion

Dispatch Delay Latency

- Objective:
 - Measure the time elapsed from when an event is fired up to when its associated handler is invoked
- Technique:
 - Associate an handler with and AsynchEvent, fire repeatedly in lock step mode the event. Measure the time elapsed between the firing of the event and the handler activation
- Tested Platforms:
 - RTSJ RI, jRate
- Test Parameters:
 - 2,000 samples of the dispatch delay time were collected

Sample Trace

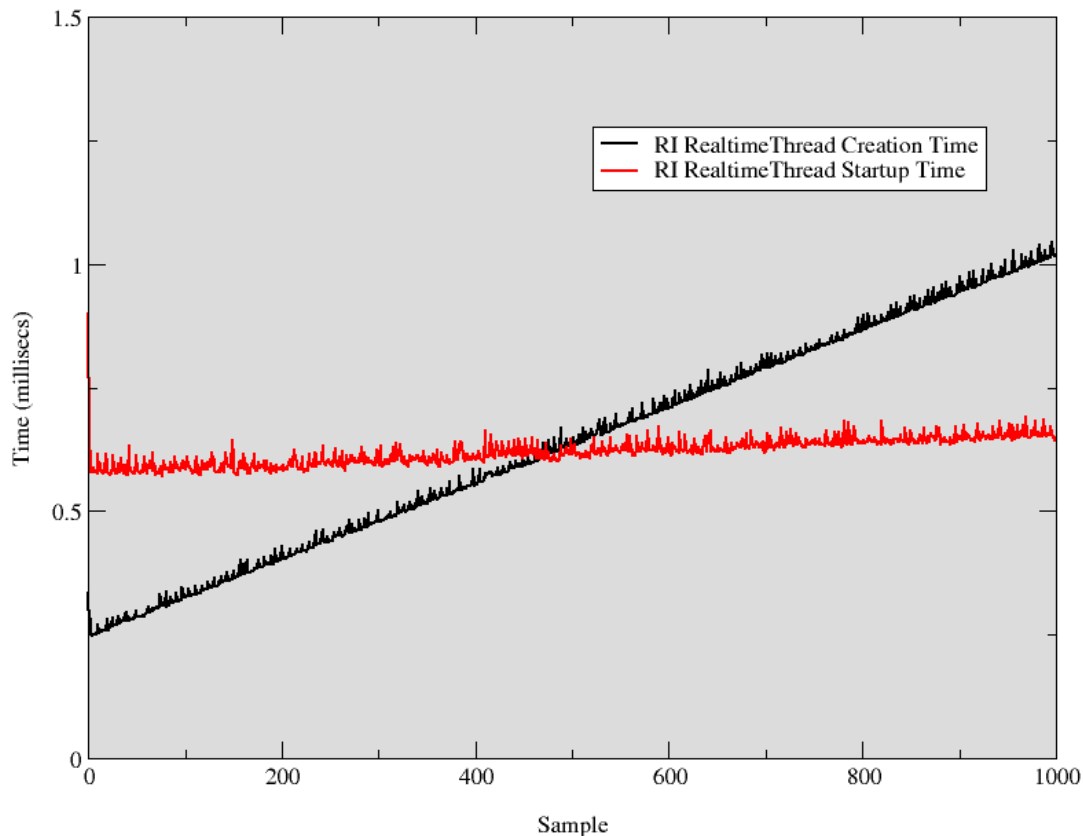
- jRate's `AsyncEventHandler` and `BoundAsyncEventHandler` have similar performances
- The sample trace shows a very predictable dispatch delay for jRate
- RI's `BoundAsyncEventHandler` provide a quite good dispatch delay latency, but is less predictable



- RI's `AsyncEventHandler` expose a strange behaviour (caused perhaps by a resource leaks associated with thread management)
- The Dispatch Latency grows linearly with the number of event fired

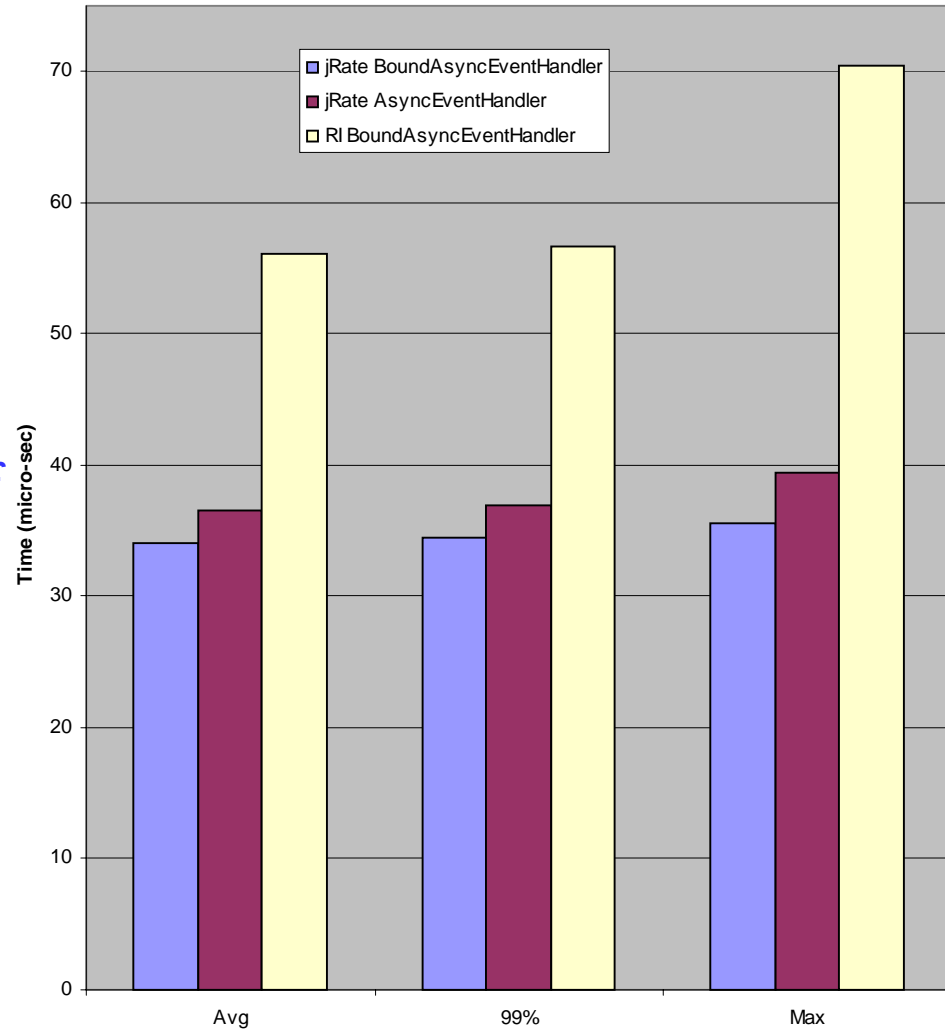
The source of RI's AEH problem

- The problem with the RI's `AsynchEventHandler` was traced back to a problem associated with `RealtimeThread` creation
- If a `RealtimeThread` is created by another `RealtimeThread`, the creation latency grows linearly



Average and Worst Case Results

- jRate's handlers have worst cases performances very close to the average and 99% case
- jRate's handler behavior quite predictable, as the low std. dev. indicates
- RI's **BoundAsyncEventHandler** has a 99% behaviour that is very close to the average case while the worst case behaviour is a little bit off.
- RI's **AsyncEventHandler** exposes an odd behaviour (as shown previously), so its data is not very representative



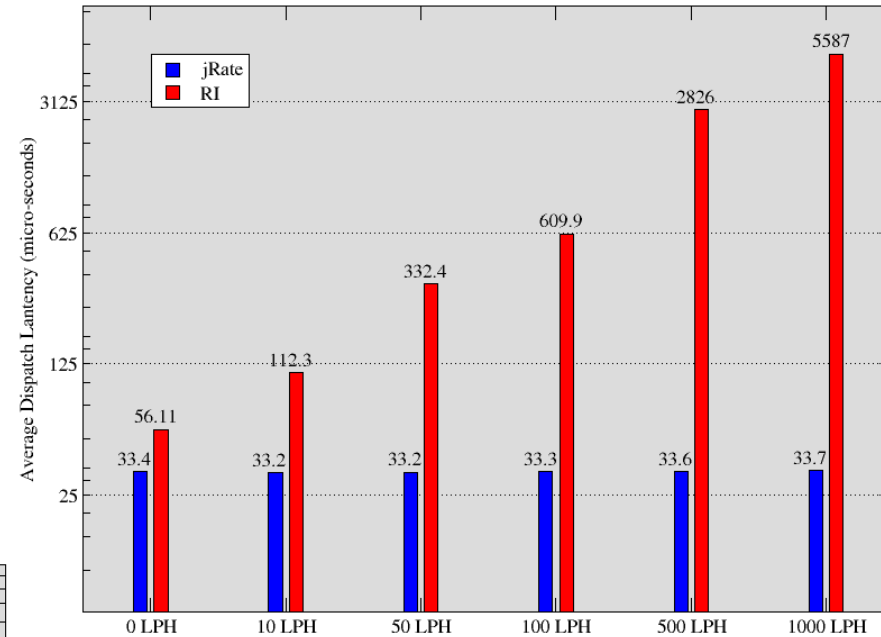
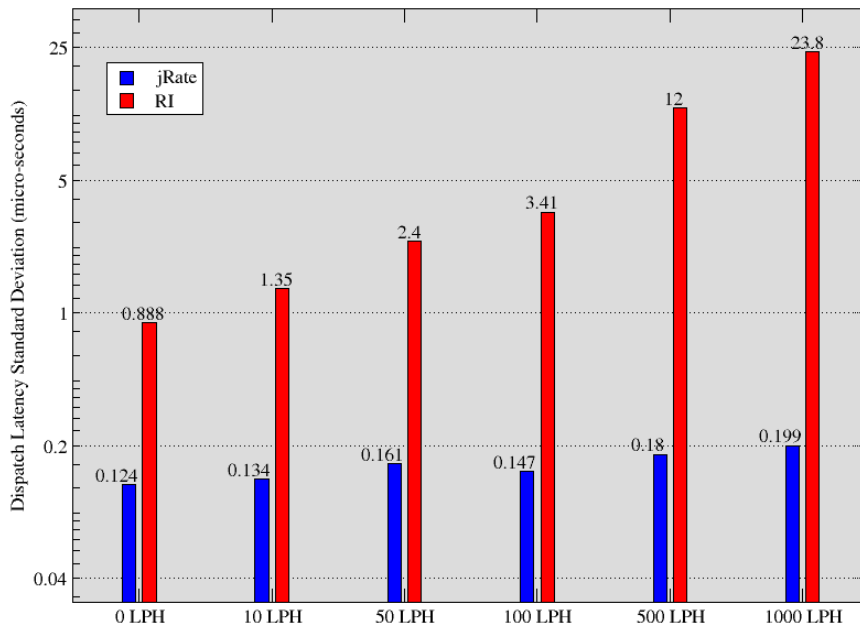
| | Avg | 99% | Max | STD |
|------------------------------|--------|--------|---------|--------|
| jRate BoundAsyncEventHandler | 34.004 | 34.472 | 35.555 | 0.148 |
| jRate AsyncEventHandler | 36.574 | 36.945 | 39.4 | 0.113 |
| RI BoundAsyncEventHandler | 56.1 | 56.692 | 70.462 | 0.848 |
| RI AsyncEventHandler | 2373 | 3892.5 | 39950.8 | 909.92 |

Dispatch Priority Inversion

- Objective:
 - Measure the presence of priority inversion present in the event handler dispatching scheme used by the RTSJ implementation
- Technique:
 - Register with an **AsyncEvent N** event handler in increasing order of execution eligibility. Measure the dispatch delay experienced by the most eligible handler for different values of N
- Tested Platforms:
 - RTSJ RI, jRate
- Test Parameters:
 - 2,000 samples of the dispatch delay experienced by the most eligible handler time were collected

Average & Dispersion Measures

- The dispatch delay experienced by the most eligible handler increases with the number of low priority handler in the RI, while it is constant for jRate
- The RI event dispatch mechanism presents, potentially, an unbounded priority inversion



- On the RI, the dispatch delay experienced by the most eligible event handler is more and more dispersed *i.e.* less predictable as the LPH number increases
- On jRate the dispatch delay standard deviation is mostly independent on the LPH number

Relevant Measures

- jRate's event dispatching mechanism provides very predictable behaviour regardless of the number of LPs

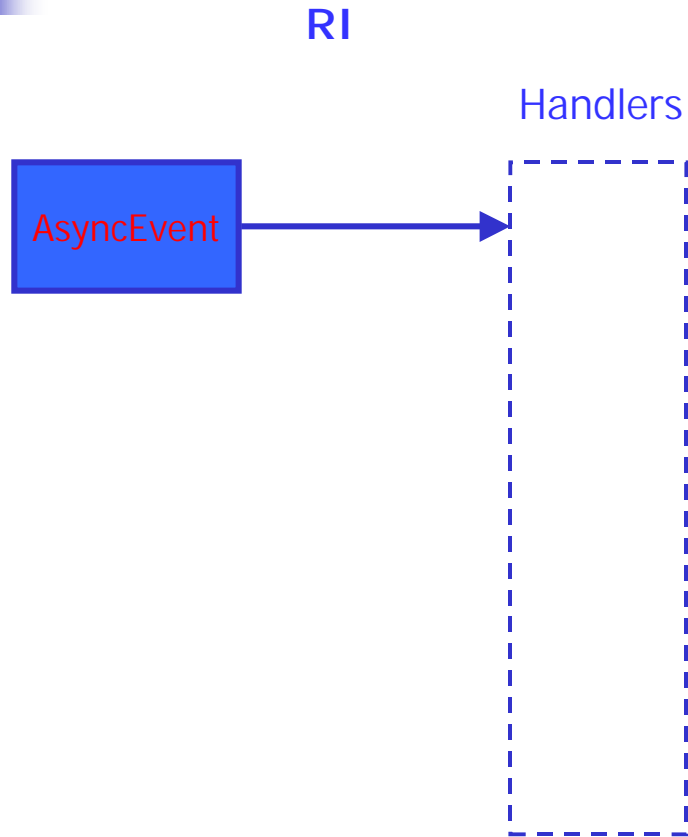
| | Avg. | Std. Dev. | Max | 99% |
|----------------|----------------|------------------|----------------|----------------|
| 0 LP | 33.375 μ s | 0.124 μ s | 34.877 μ s | 34.116 μ s |
| 10 LP | 33.154 μ s | 0.134 μ s | 34.903 μ s | 33.797 μ s |
| 50 LP | 33.205 μ s | 0.161 μ s | 36.063 μ s | 33.825 μ s |
| 100 LP | 33.264 μ s | 0.147 μ s | 35.959 μ s | 33.851 μ s |
| 500 LP | 33.632 μ s | 0.180 μ s | 37.149 μ s | 34.283 μ s |
| 1000 LP | 33.739 μ s | 0.199 μ s | 37.565 μ s | 34.458 μ s |

- RI's event dispatching mechanism suffer of a form of unbounded priority inversion

| | Avg. | Std. Dev. | Max | 99% |
|----------------|----------------|------------------|----------------|----------------|
| 0 LP | 56.106 μ s | 0.887 μ s | 70.462 μ s | 56.706 μ s |
| 10 LP | 112.33 μ s | 1.346 μ s | 133.90 μ s | 122.18 μ s |
| 50 LP | 332.41 μ s | 2.396 μ s | 353.17 μ s | 344.86 μ s |
| 100 LP | 609.92 μ s | 3.410 μ s | 631.51 μ s | 624.96 μ s |
| 500 LP | 2826.4 μ s | 12.005 μ s | 2884.0 μ s | 2862.1 μ s |
| 1000 LP | 5587.0 μ s | 23.768 μ s | 5672.7 μ s | 5650.3 μ s |

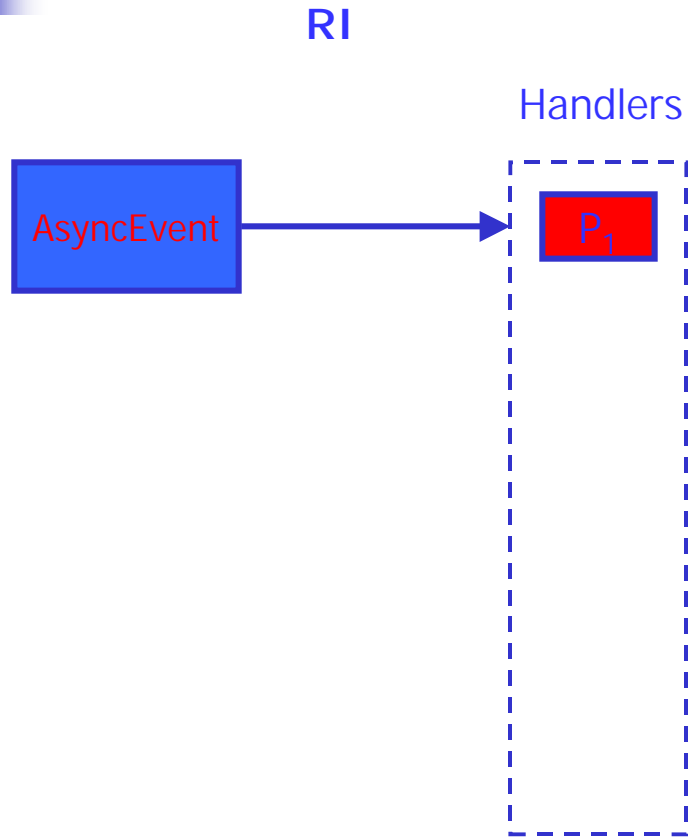
The source of RI's AEH problem

Dispatch Priority Inversion

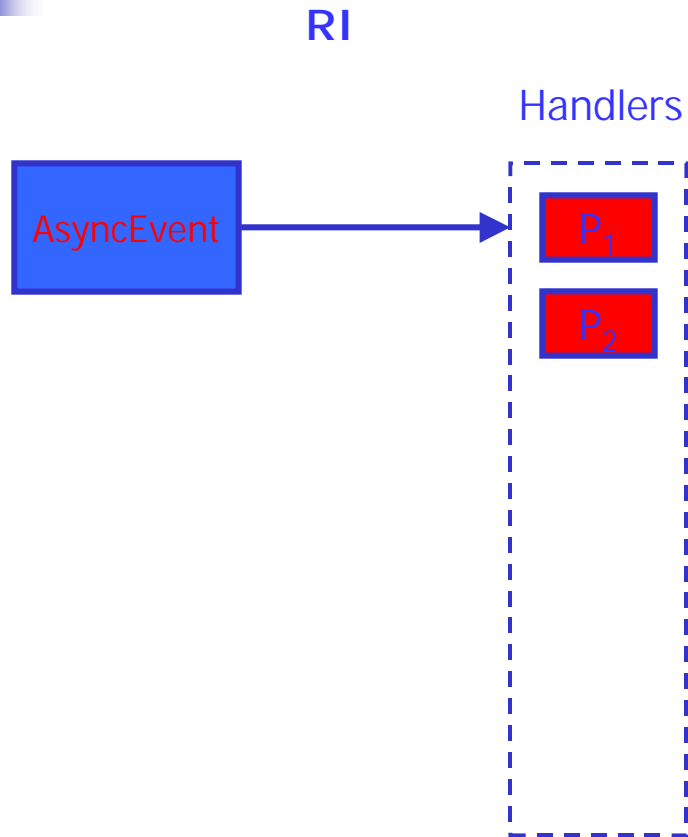


The source of RI's AEH problem

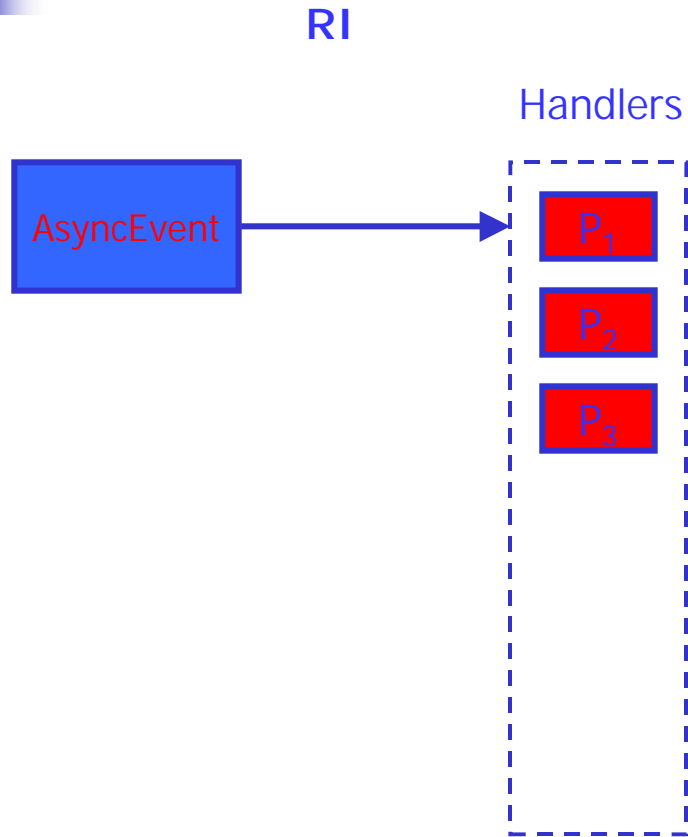
Dispatch Priority Inversion



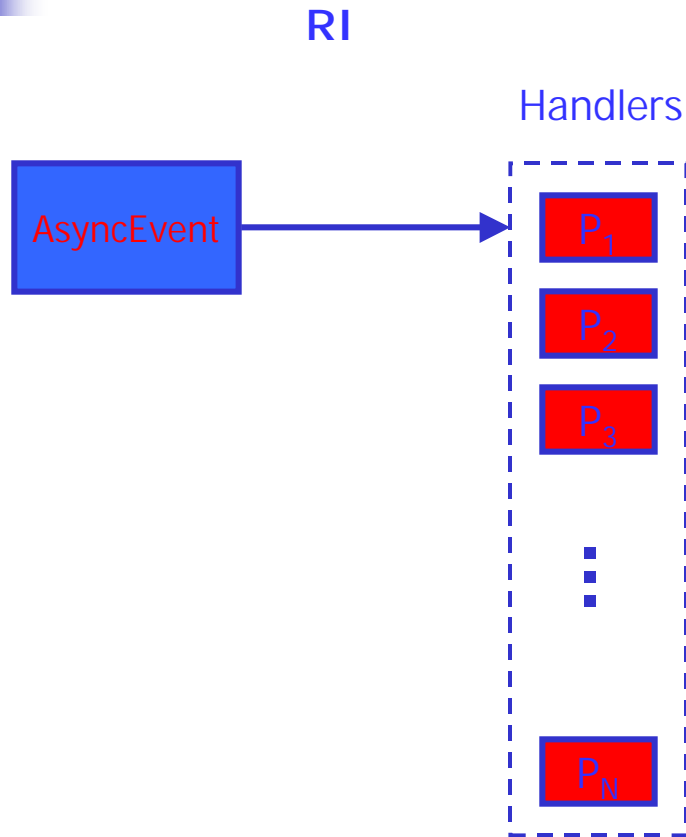
The source of RI's AEH problem



The source of RI's AEH problem

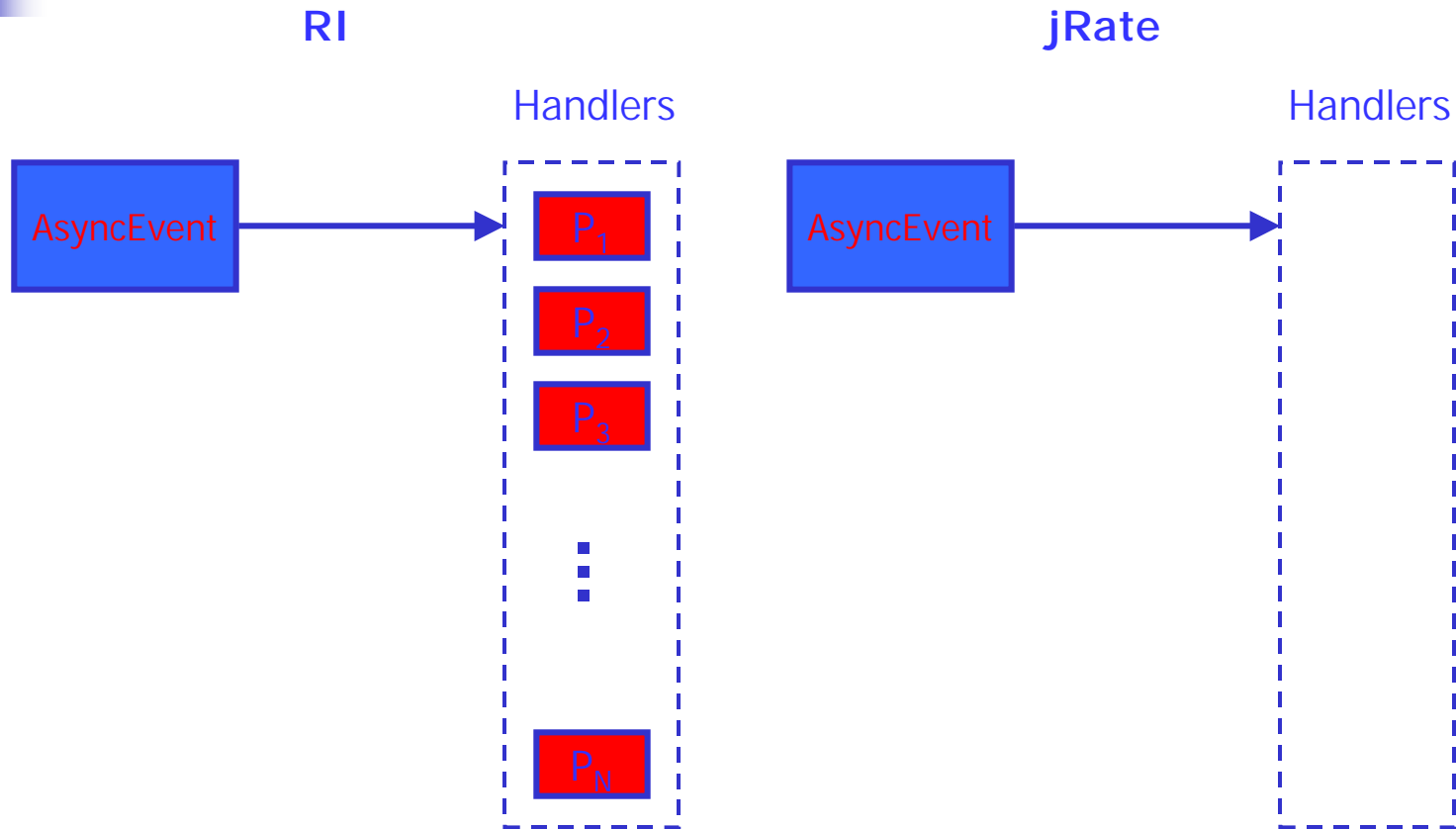


The source of RI's AEH problem



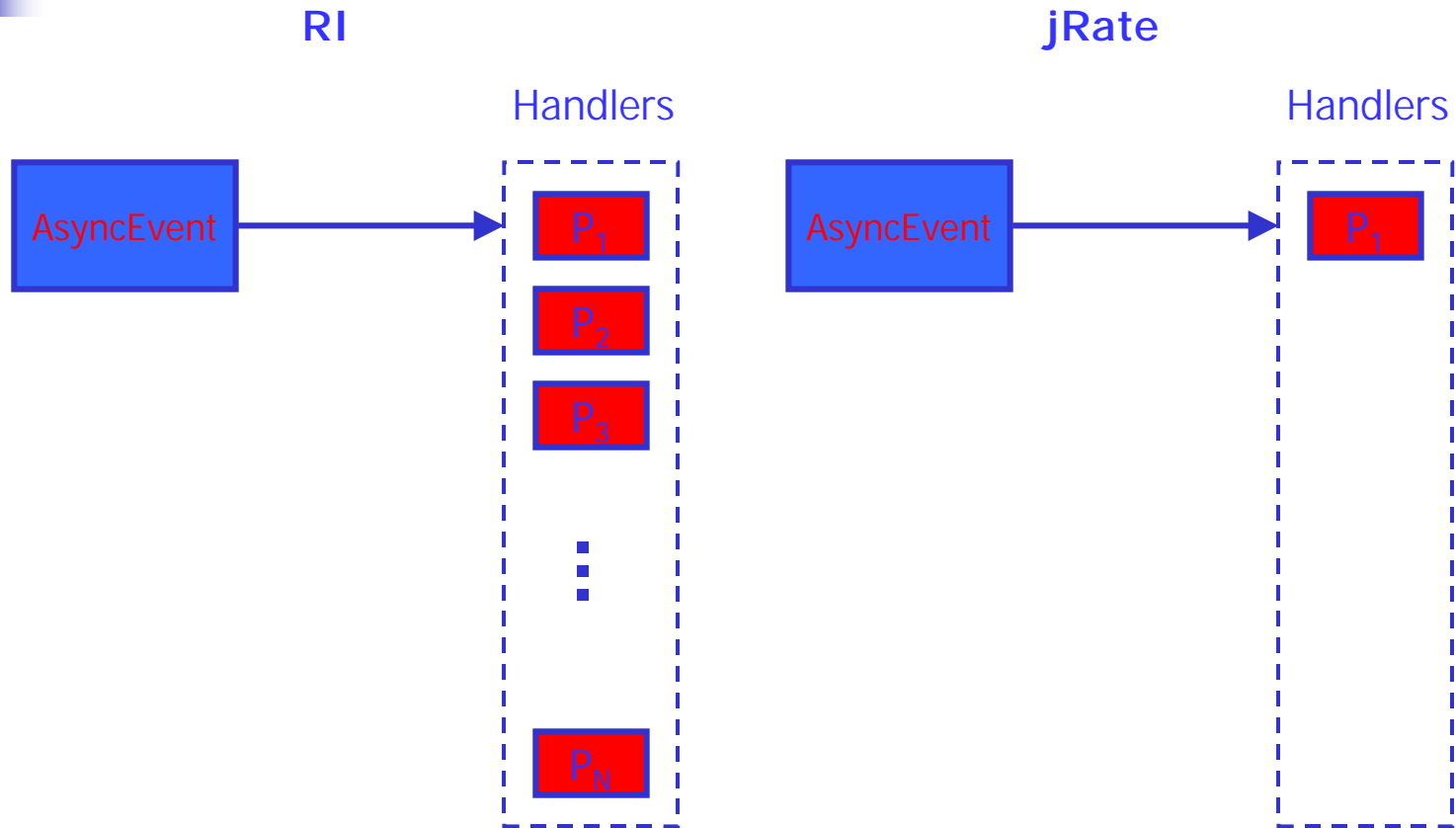
$$P_1 \leq P_2 \leq P_3 \leq \dots \leq P_{N-1} < P_N$$

The source of RI's AEH problem



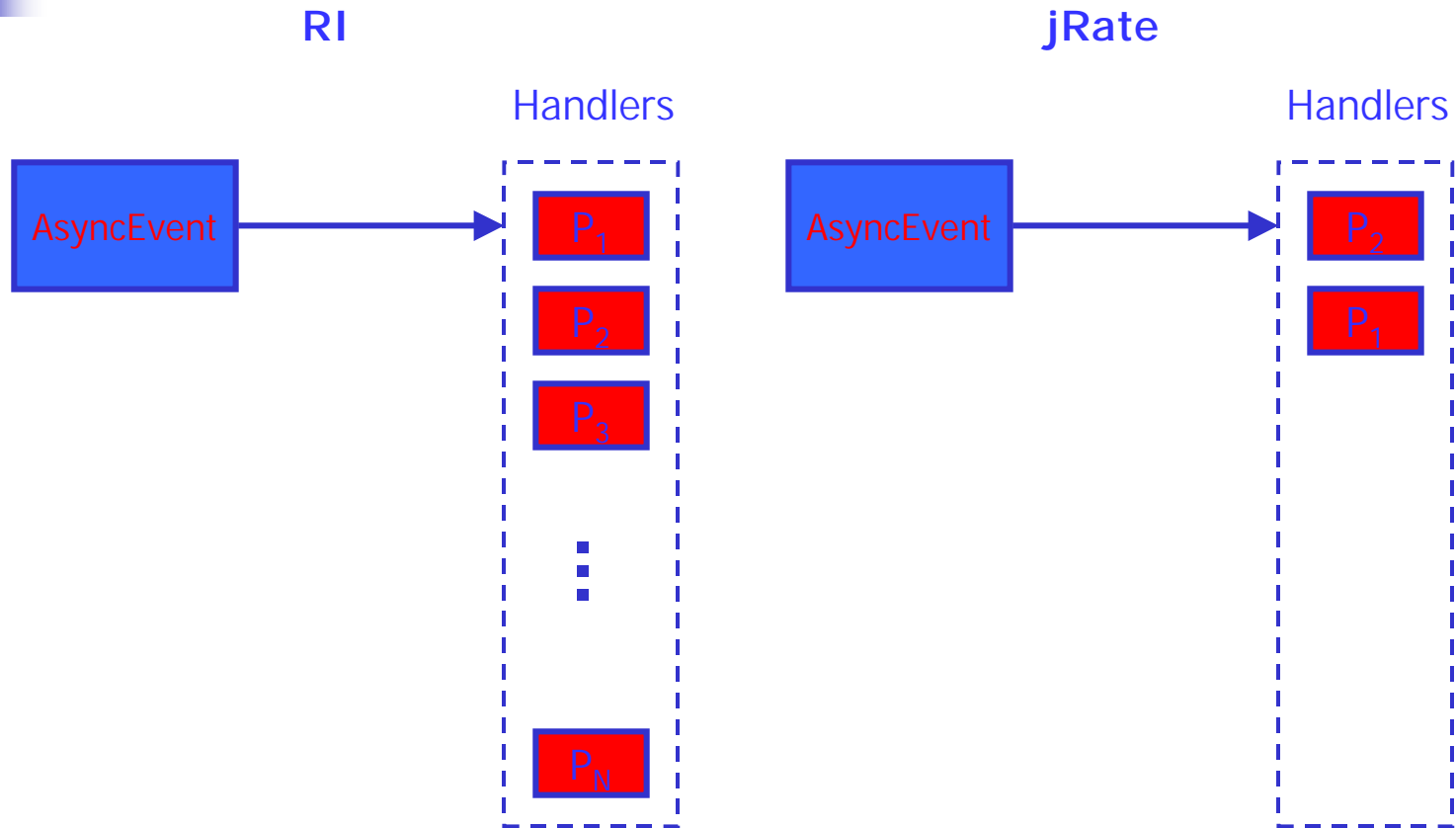
$$P_1 \leq P_2 \leq P_3 \leq \dots \leq P_{N-1} < P_N$$

The source of RI's AEH problem



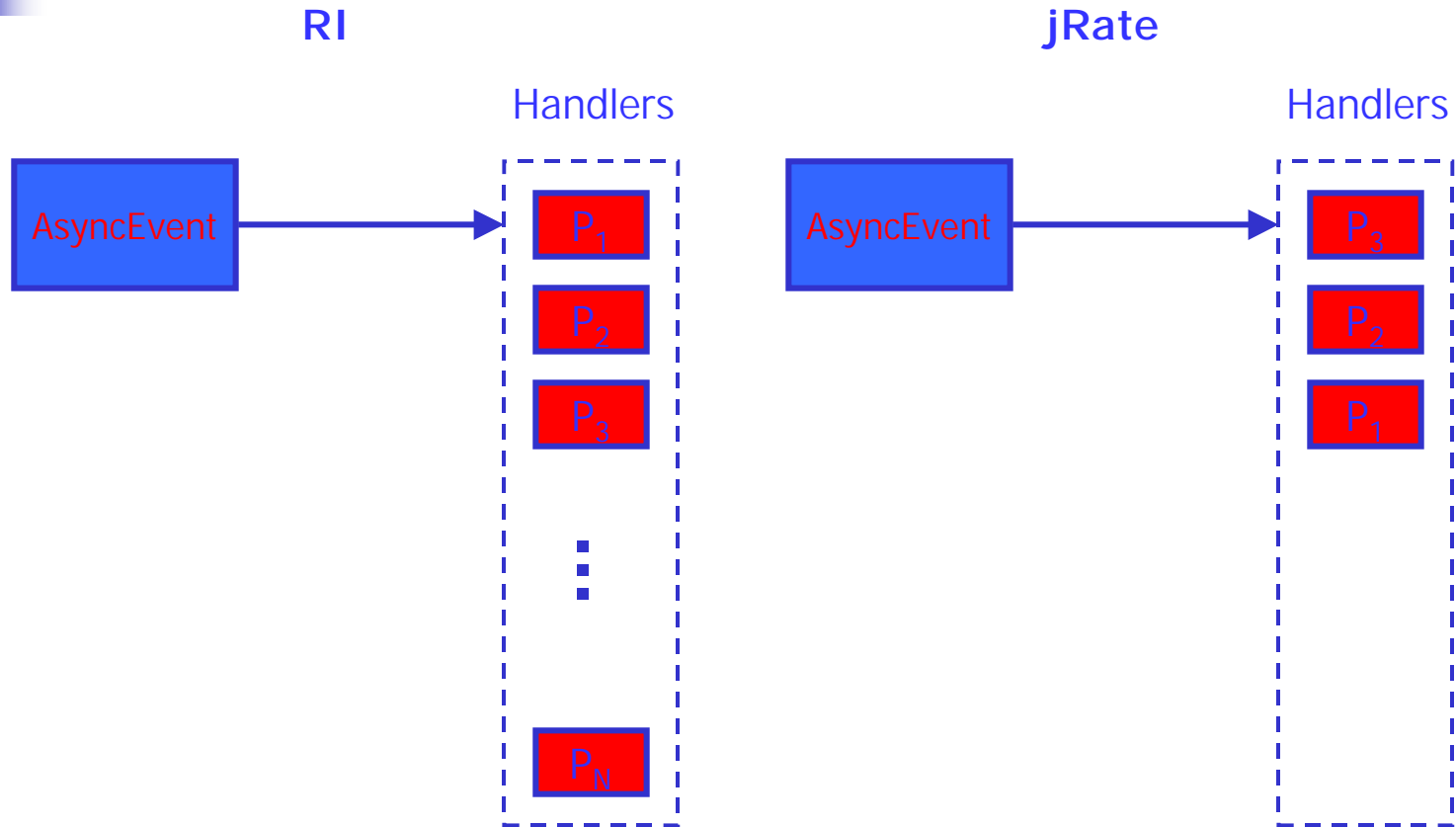
$$P_1 \leq P_2 \leq P_3 \leq \dots \leq P_{N-1} < P_N$$

The source of RI's AEH problem



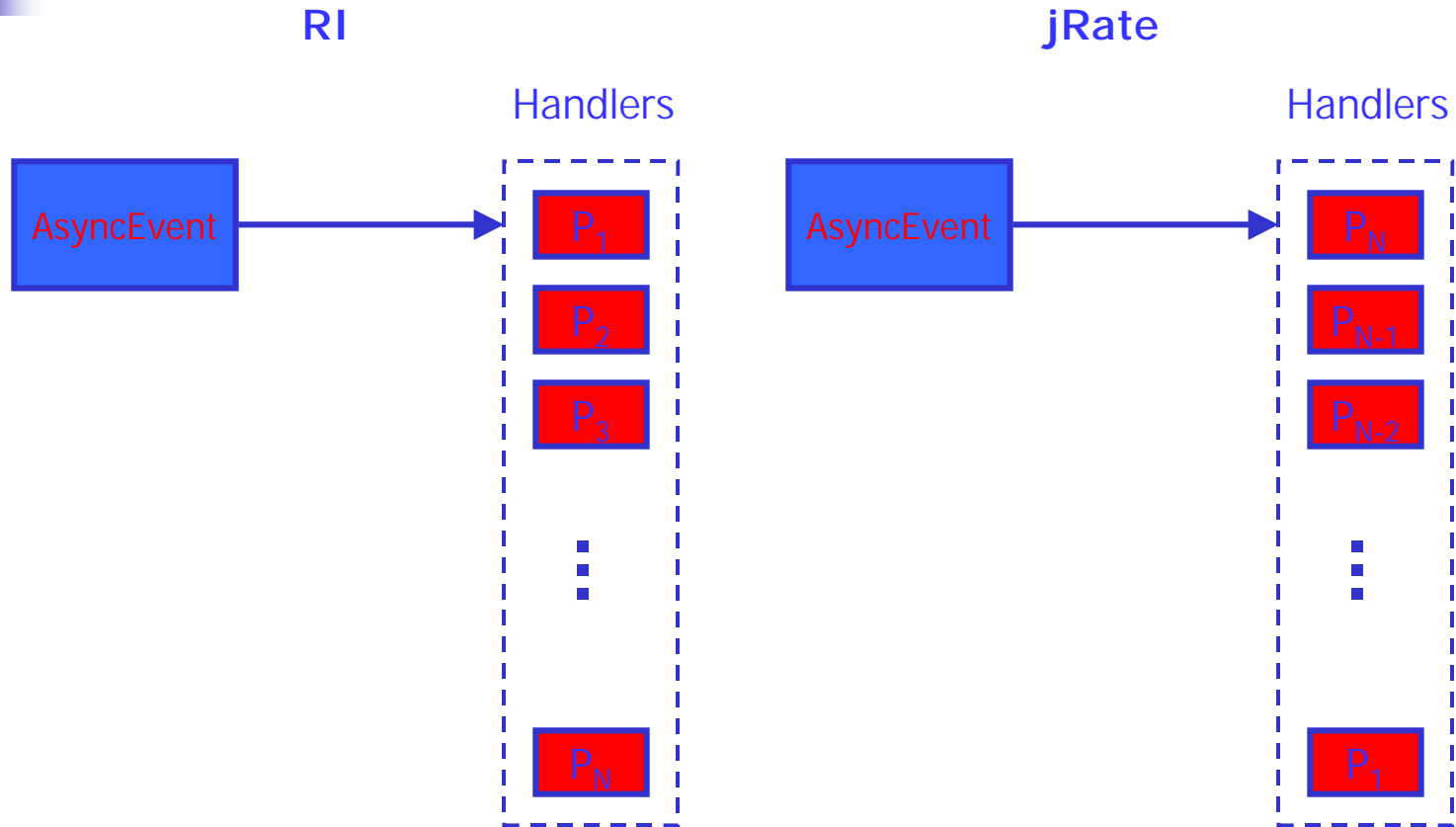
$$P_1 \leq P_2 \leq P_3 \leq \dots \leq P_{N-1} < P_N$$

The source of RI's AEH problem



$$P_1 \leq P_2 \leq P_3 \leq \dots \leq P_{N-1} < P_N$$

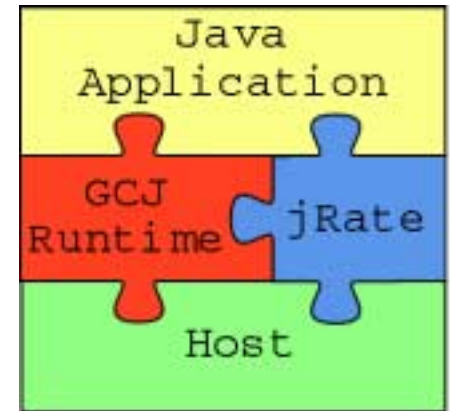
The source of RI's AEH problem



"The jRate Plan"

- jRate's is an extension of the GCJ runtime system that provides a subset of the RTSJ features such as:

- Realtime Threads
- Scoped Memory
- Asynchrony



- jRate's goals:
 - Apply AOP technique to produce an untangled, and "pick what you need" RTSJ implementation
 - AspectJ & AspectC++ will be used to do AOP in the Java and C++ portion of **jRate** respectively
 - Provide higher level programming model to developers
 - Provide advanced scheduling services

RTJPerf

- RTJPerf will be extended to support:
 - Test that cover remaining areas of the RTSJ
 - Memory Efficiency Tests
 - Tests under Heavy Load Condition
- RTJPerf will become part of a bigger benchmarking effort that is being carried by Boeing and AFRL

Concluding Remarks

- jRate provide quite good performances, and in many cases its performances index were quite close to C++
- The Reference Implementation, has some weak points, that have to do in some cases with *naïve* implementation techniques
- Some of the results provided by our experiments point out that Real-Time Java implementation that can be used for real problem are starting to come along... But more work is still needed

References

- A paper that provides a complete description of the RTJPerf tests, and the results measured for jRate, RI, CVM and JDK 1.4 is available at:
 - <http://tao.doc.wustl.edu/~corsaro/papers.html>
- jRate and RTJPerf are Open Source, for scheduled releases and download check:
 - <http://tao.doc.wustl.edu/~corsaro/jRate>
- The RTSJ Spec is available at:
 - <http://www.rti.org>