Flexible Task Graphs:
A Framework for Experimental Real Time Programming in Java

Joshua Auerbach
IBM Research

with slides borrowed liberally from Jan Vitek (Purdue), representing joint work with Jan, Jesper Spring (EPFL), David Bacon (IBM), Rachid Guerraoui (EPFL) and also…

Other Contributers

Additional co-authors of precursors:

- Eventrons: Daniel Spoonhower (CMU), Perry Cheng, David Grove (IBM) (with Bacon and Auerbach)
- Reflexes: Filip Pizlo (Purdue) (with Vitek, Spring, and Guerraoui)
- Exotasks: Christoph Kirsch (U. Salzburg) (with Bacon and Auerbach)
- StreamFlex: Jean Privat (UQAM) (with Vitek. Spring, and Guerraoui)

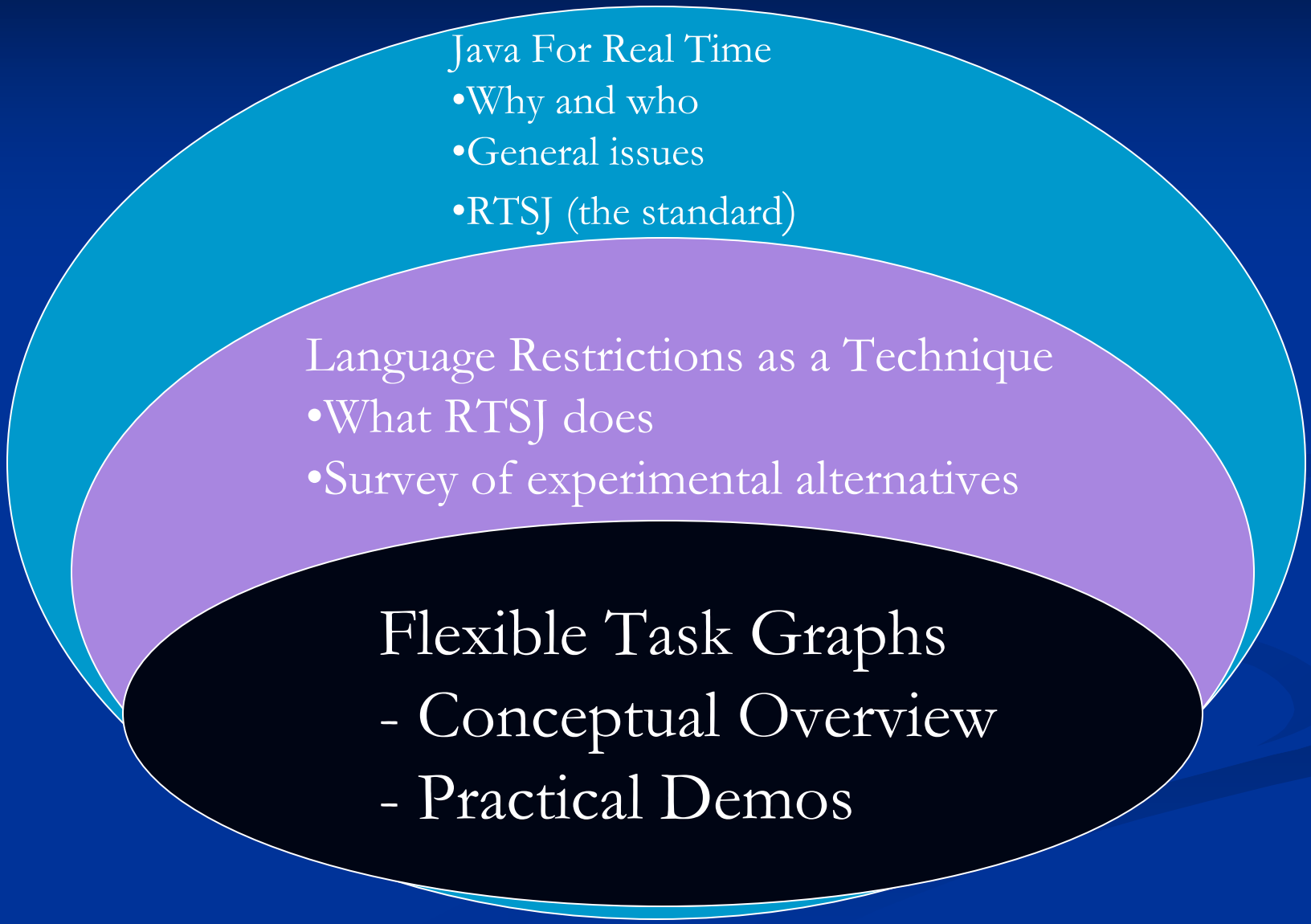
Scheduler developers:

- Daniel Iercan (U. Timisoara) (HTL, with Kirsch)
- Jia Zou and Edward Lee (U. Cal Berkeley) (PTIDES, with Bacon and Auerbach)

Applications:

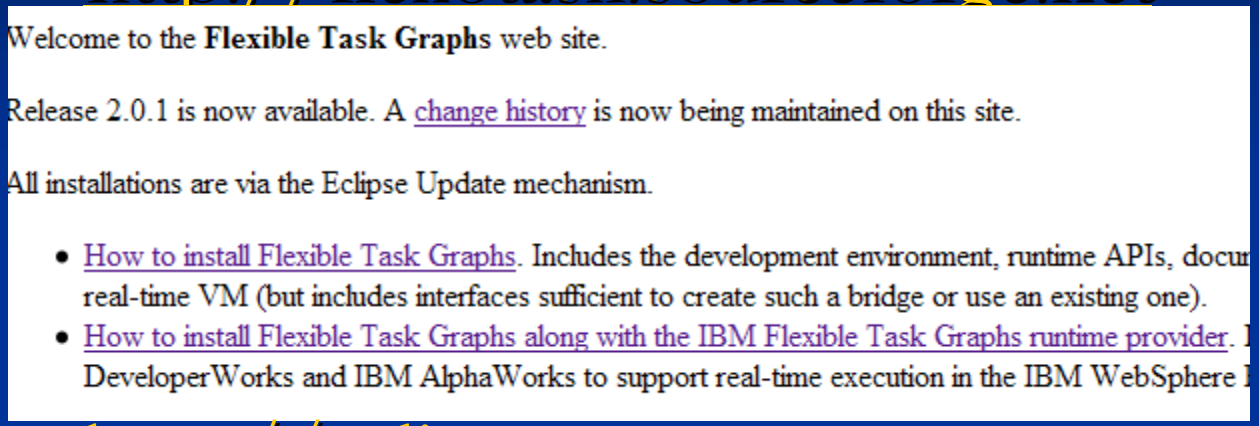
- Rainer Trummer and Harald Roeck (U. Salzburg) and V T Rajan (IBM) (the JAviator, with Auerbach and Bacon)

Broad Outline



Flexible Task Graphs Software

- http://flexotask.sourceforge.net


Welcome to the **Flexible Task Graphs** web site.

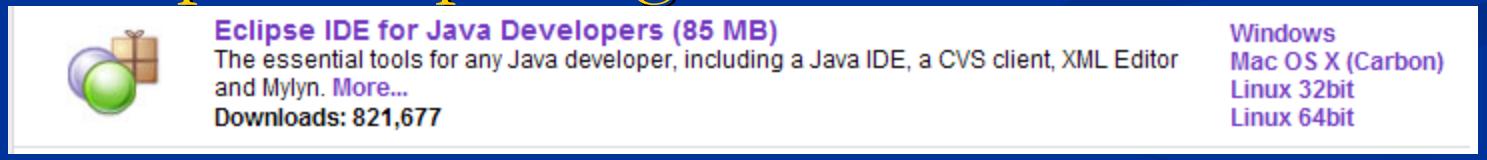
Release 2.0.1 is now available. A change history is now being maintained on this site.

All installations are via the Eclipse Update mechanism.

- How to install Flexible Task Graphs. Includes the development environment, runtime APIs, docu real-time VM (but includes interfaces sufficient to create such a bridge or use an existing one).
- How to install Flexible Task Graphs along with the IBM Flexible Task Graphs runtime provider. DeveloperWorks and IBM AlphaWorks to support real-time execution in the IBM WebSphere

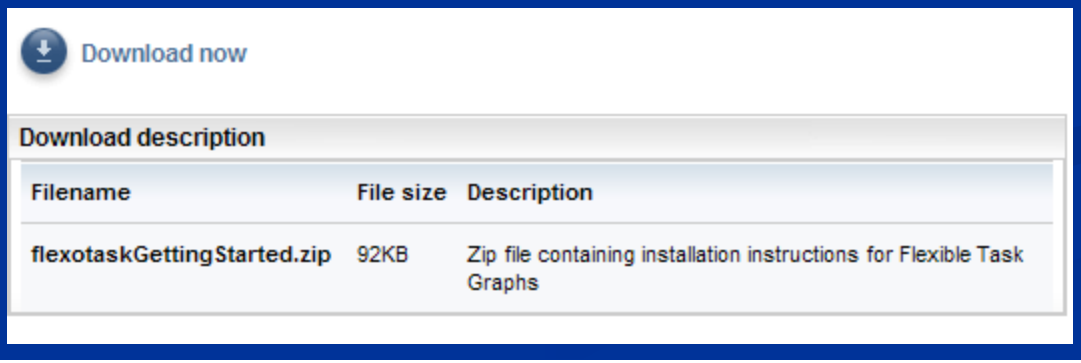
API classes, development tools, debug/test, checkers, builders, etc. Open source.

- http://eclipse.org


Eclipse IDE for Java Developers (85 MB)
The essential tools for any Java developer, including a Java IDE, a CVS client, XML Editor and Mylyn. More...
Downloads: 821,677

Windows
Mac OS X (Carbon)
Linux 32bit
Linux 64bit

- http://www.alphaWorks.ibm.com/tech/flexotasks


Download now

Download description

Filename	File size	Description
flexotaskGettingStarted.zip	92KB	Zip file containing installation instructions for Flexible Task Graphs

Runtime bridge to a specific Real-time JVM (IBM's WebSphere Real Time). Binary only.

Flexible Task Graphs Software

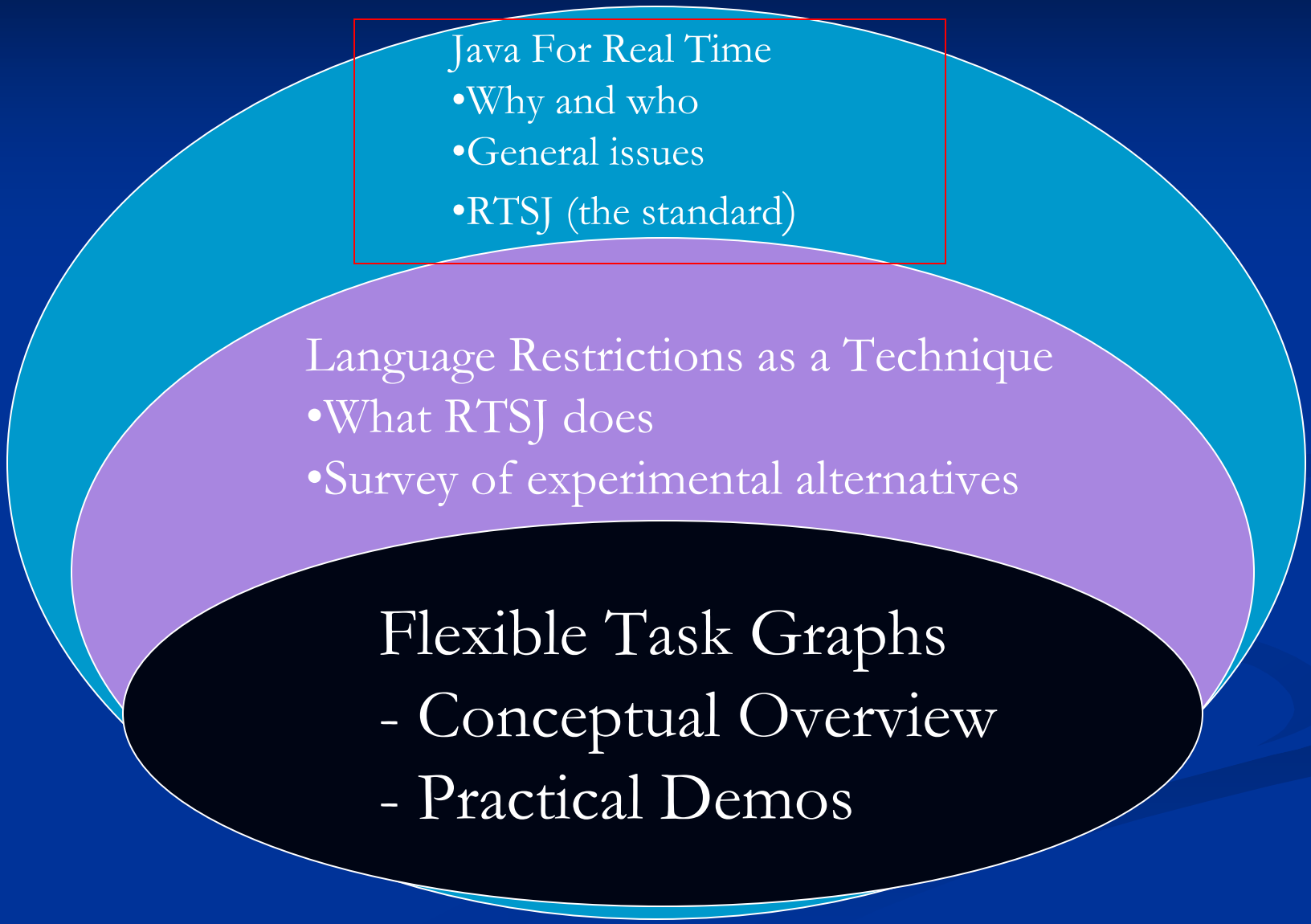
- http://www.ibm.com/developerworks/java/jdk/linux/download.html

Java SE Version 6

	Download	Plug-in support	Web Start support
64-bit AMD/Opteron/EM64T	SR4	No	Yes
32-bit xSeries (Intel compatible)	SR4	Yes	Yes
32-bit iSeries/pSeries	SR4	Yes	Yes
64-bit iSeries/pSeries	SR4	Yes	Yes
31-bit zSeries (S/390)	SR4	No	No
64-bit zSeries (S/390)	SR4	No	No
WebSphere Real Time V2.0 32-bit xSeries (Intel compatible)	SR1	No	No

IBM WebSphere Real Time itself … free download (no support, though, support costs $$ ☺) … requires a recent RT Linux kernel such as RHEL5RT

Broad Outline



What is a real-time system?

- Real-time system: a system which has to respond to external inputs within a finite and specified period
  - *correctness includes timeliness*
  - *being late is as bad as being wrong!*
- Usually, an *embedded* system (a component of some larger hardware structure)
- 99% of all processors are for the embedded systems market

Kinds of Real Time Systems

- "Hard" real time – a missed deadline is equated to program failure (airplane crashes)
- "Soft" real time – missed deadlines are bad but the occasional outlier may be tolerated (video glitch, dropped telephone call)
- Java is potentially usable for both
- Hard RT community may be slower to adopt

Relevance of Embedded Systems

- "[…] estimate that each individual […] may unknowingly use more than 100 embedded computers daily"
- "The world market for embedded software will grow from about $1.6 billion in 2004 to $3.5 billion by 2009, at an average annual growth rate (AAGR) of 16%."
- "Embedded hardware growth will be at the aggregate rate of 14.2% to reach $78.7 billion in 2009, while embedded board revenues will increase by an aggregate rate of 10%"

- http://www.bccresearch.com/comm/G229R.html, http://www.ecpe.vt.edu/news/ar03/embedded.html

What Programming Technologists can contribute to Real-Time

- Real time systems are getting more complex
  - Low level languages tedious
- Current RT programming models are function of hardware + OS
  - Not portable
- Only *some* of the code in a typical application is RT
- Managed languages like Java and C#: safe, platform independent, productive ….
  - But, can they achieve RT goals?

Why do Language and Model Matter?

- Development time, code quality and certification are increasingly criteria. For instance in the automotive industry:
  - 90% of future innovation in the auto industry will be driven by electronics and software — *Volkswagen*
  - 80% of car electronics in the future will be software-based — *BMW*
  - 80% of our development time is spent on software — *JPL*
- Worst, software is often the source of missed project deadlines.

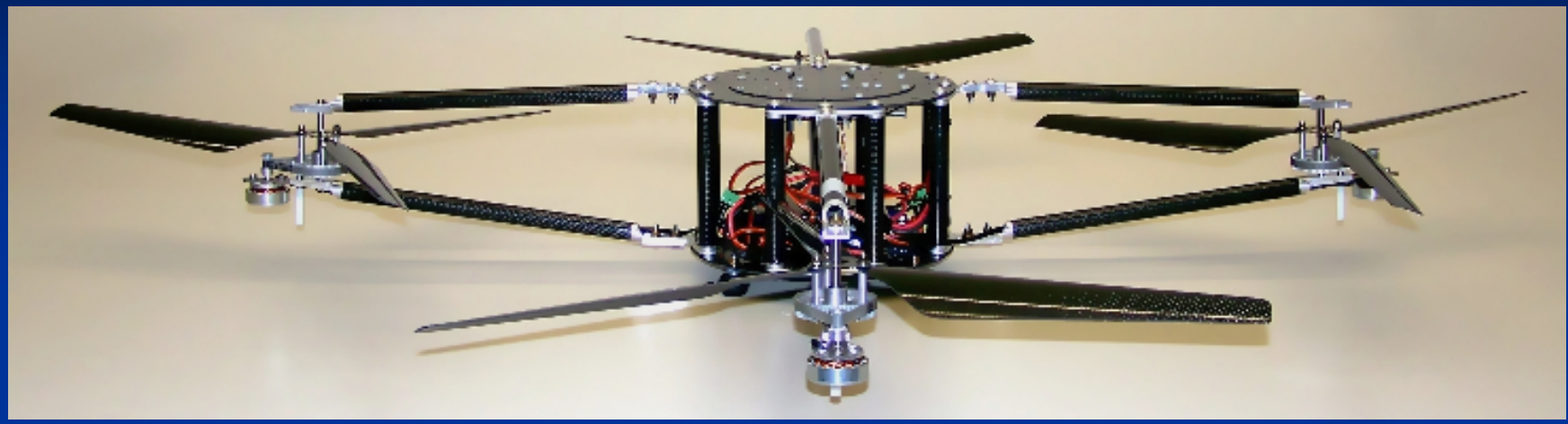
Real-time Java Challenges

- Garbage Collection
  - Requires specially designed GC
  - Or, an alternative memory model
- Distance from the "iron"
  - Scheduling (priorities etc)
  - Mapping physical memory
  - Handling interrupts
- JIT Compilation
  - AOT compilation
- Class loading
  - Closed world plus init-time loading

Real-time applications

- Shipboard computing
  - US navy Zumwalt-class Destroyer,
    Raytheon / IBM
    5 million lines of Java code
    Real-time GC key part of system.
- Avionics
  - Zedasoft's Java flight simulator
  - IBM Scaneagle UAV
  - The JAviator
- Financial information systems

JAviator



- Quad-rotor model helicopter built at U. Salzburg
  - Gyro, sonar, motor speed controllers
  - Arms don't tilt or move: interesting control problem
- Contains 600mhz ARM processor 128m memory, ROM disk,
- RT Linux kernel
- Runs modified version of IBM's WebSphere Real Time VM
- Flew it using Exotasks (precursor of Flexible Task Graphs) – LCTES 2007

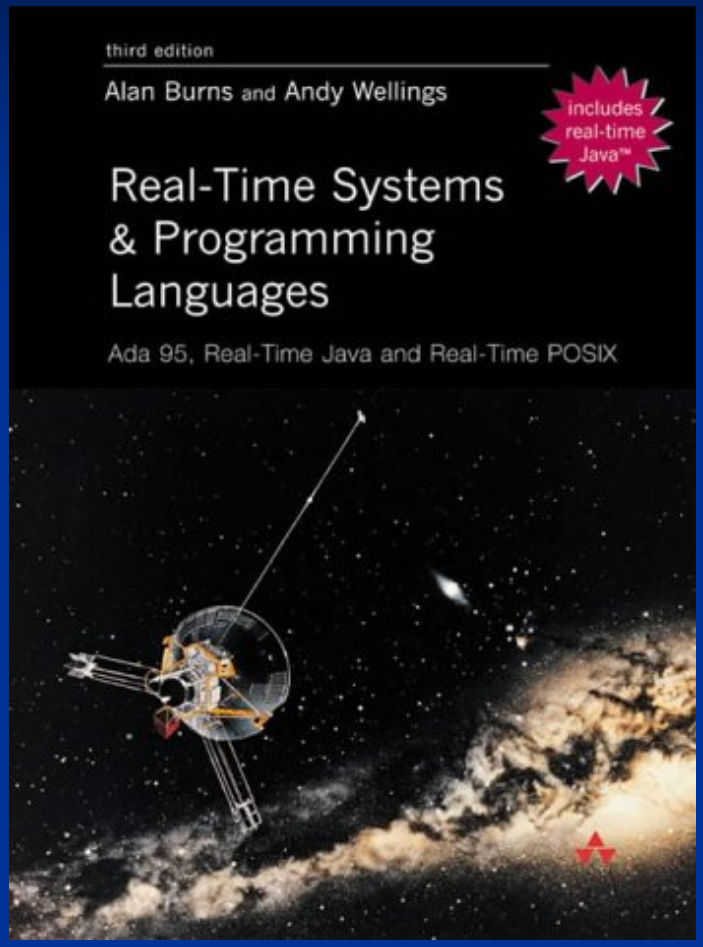
The Real Time Specification for Java

- (or RTSJ, or JSR-1)
- Implemented by (at least)
  - Commercial
    - IBM: WebSphere Real Time
    - Sun: Java RTS
    - AICAS: JamaicaVM
  - Experimental
    - OVM (Purdue)

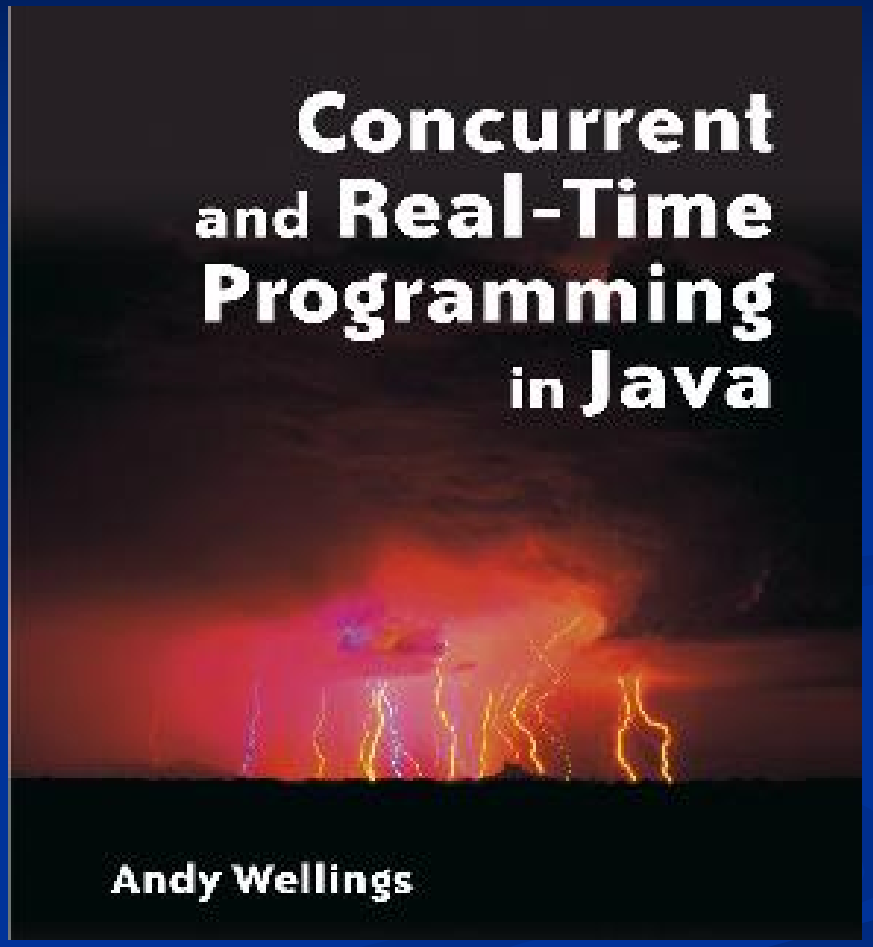
RTSJ Programming Model

- Java-like:
  - Shared-memory,
  - lock-based synchronization,
  - first class threads,
  - No new syntax!
  - No serious consideration of multi-core
- Main real-time additions:
  - Real-time threads + Real-time schedulers
  - Priority avoidance protocols: PIP or PCE
  - Region-based memory allocation (to avoid GC pauses)

Books



RTSJ Version 0.9



RTSJ Version 1.0.1

Overview of RTSJ Coverage

- memory management (focus)
- time values and clocks (mention)
- schedulable objects and scheduling (overview)
- real-time threads (overview)
- asynchronous event handling and timers (overview)
- asynchronous transfer of control (mention)
- synchronization and resource sharing (mention)
- physical memory access (mention)

Quick Mention

- time values and clocks
  - HighResolutionTime nanosecond granularity
- asynchronous transfer of control (termination)
  - Alternative to deprecated stop() or imprecise interrupt() concepts
  - Thread enables/defers ATC (deferred by default)
  - Synchronized methods defer ATC as side-effect

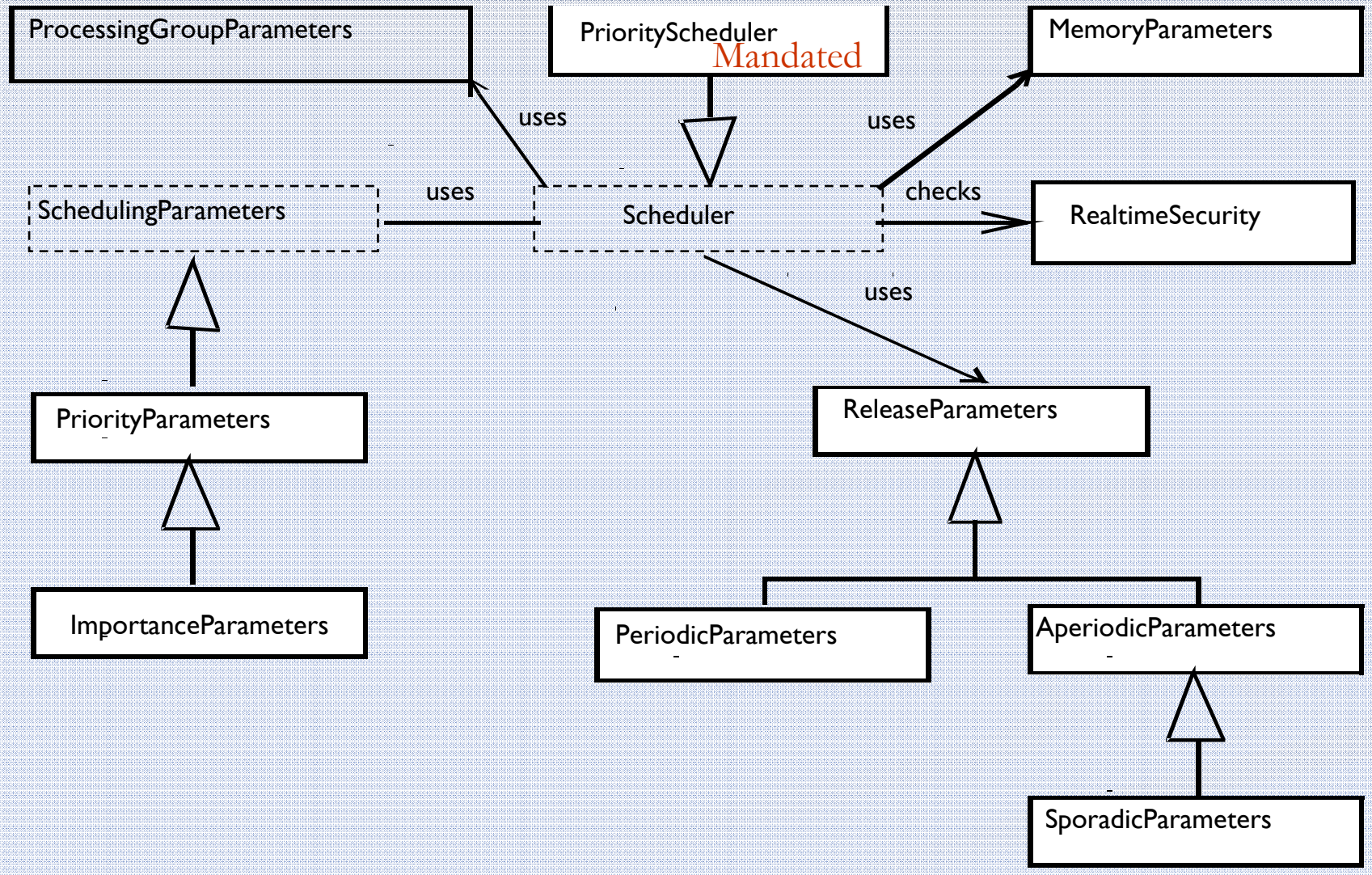
Quick Mention (2)

- synchronization and resource sharing
  - Selectable algorithms to avoid priority inversion (priority inheritance, priority ceiling emulation)
- physical memory access
  - Placement of objects in parts of memory with particular properties
  - Allow access to raw memory locations used to interface to the outside world (e.g memory-mapped I/O)
  - Access to raw memory in terms of reading and writing primitive data types (int, long, float etc.)

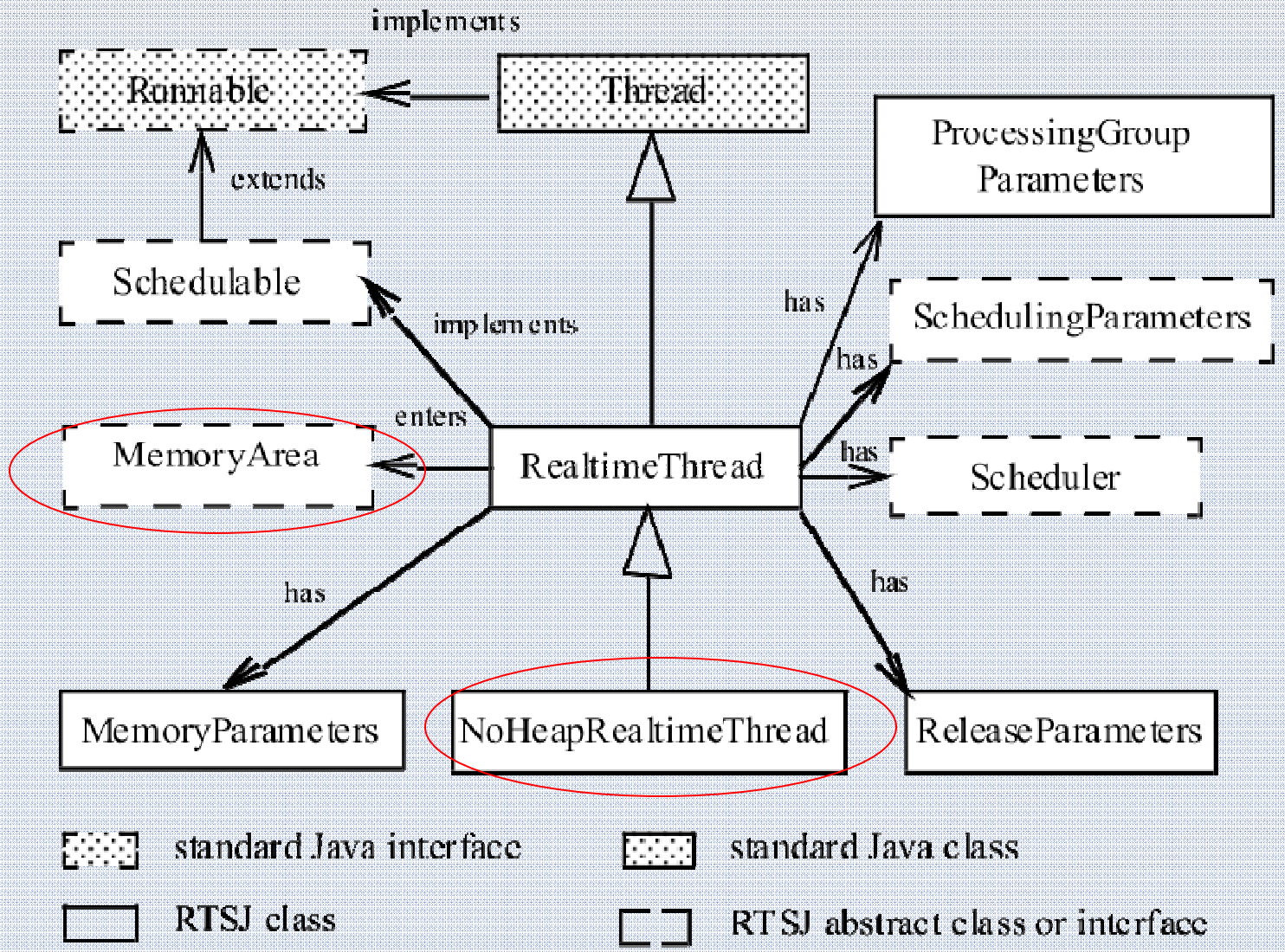
Threads, Events, and Scheduling in RTSJ

- Reuse Thread construct with more precise control over scheduling
  - Standard Java has only 10 priority levels and no real guarantees that priorities are honored
  - RTSJ mandates at least 28 levels with pre-emptive discipline
- Add asynchronous events as a distinct schedulable object on a par with Threads
- A schedulable object (**RealtimeThread** or **AsyncEventHandler**) implements the **Schedulable** interface

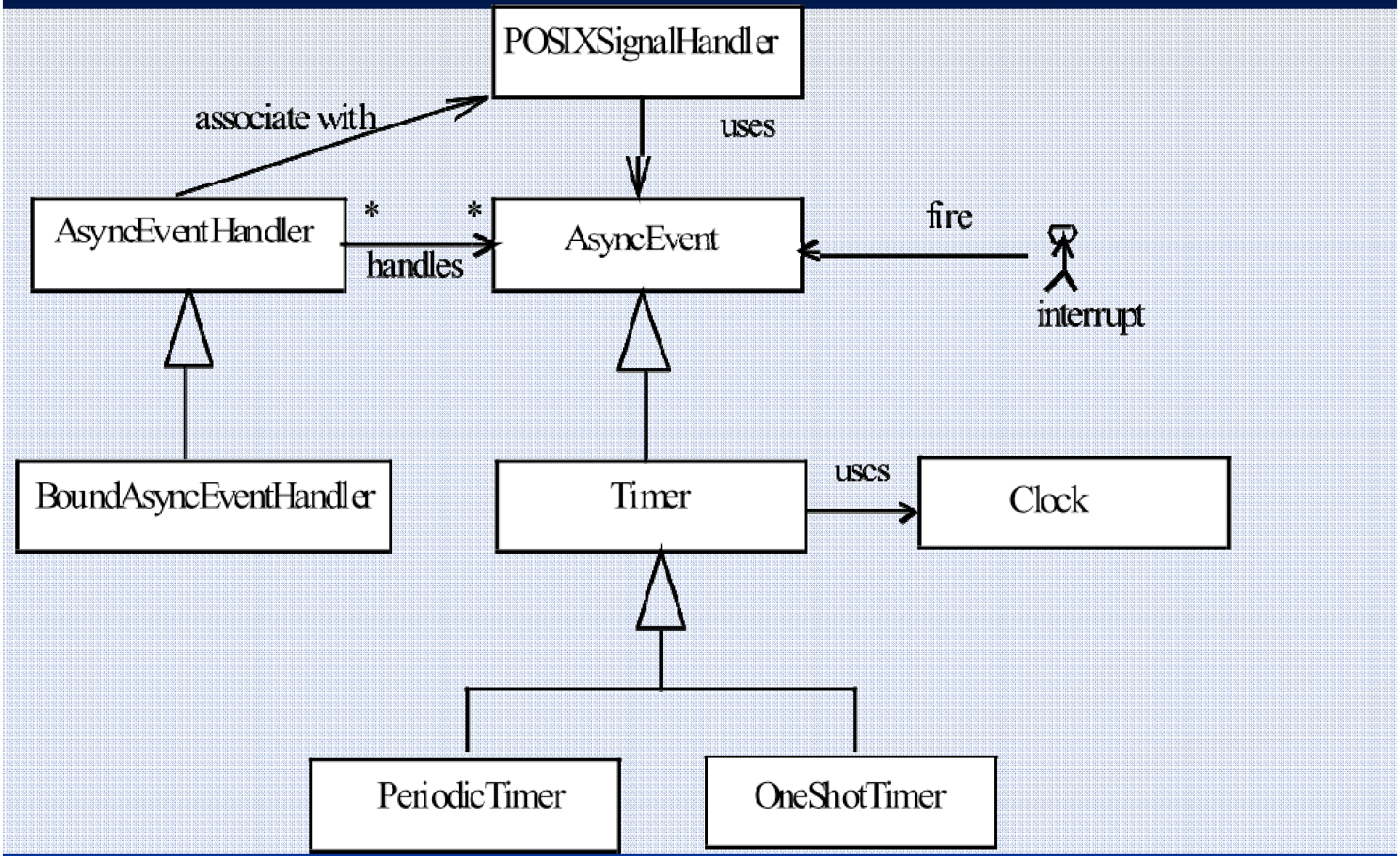
Scheduling-related Classes



Real-time Threads



Async Event related Classses



Memory Management Issues

Two main approaches

1. Preserve Java semantics exactly, make RT garbage collectors with sufficiently low latency.
2. Introduce alternative memory model
   - Still no 'free' operation
   - Allocations go to specialized memories based on context
   - Restrictions on subsets of code ensure safe behavior
- (Not seriously considered: malloc/free style memory management)

What RTSJ Does

- Originally developed under the assumption that RT GC was infeasible
  - So, alternative memory model
- In practice, there is usually also a RT GC and the standard acknowledges that.
- A given RT GC has a lower limit on latency, below which the alternatives become important
  - Note this can vary by vendor

Real Time Garbage Collection

- Basically, collects concurrently or in tiny increments *with attention to scheduling*
- Classical solutions (e.g. Baker [CACM 1978]) are "work based"
  - A snippet of GC work for each N bytes allocated
  - Good at keeping up
  - Inadequate for hard real time due to clustering of pauses
- Modern variants are
  - "time based" (e.g. Metronome, adopted by IBM and OVM),
  - "slack based" (e.g. Henriksson's collector, adopted by Sun)
  - or "creative hybrids" (e.g. Metronome-TS, experimental, IBM).

Metronome

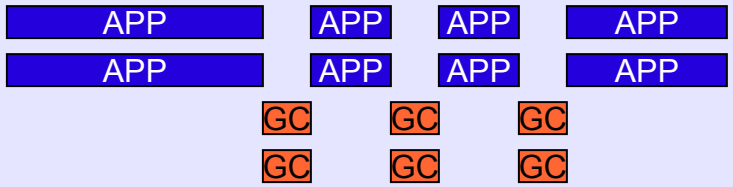
- Uses Minimum Mutator Utilization (MMU, Cheng and Blelloch) for scheduling
  - MMU(*w*) == minimum fraction of time available for application use over a (sliding) time window of size *w*
  - IBM WebSphere Real Time, MMU(10ms) = .70 by default *Oversampled* so that pauses of max 500*u*s are spread over the 10ms window
  - Avoids clustering of pausesCan still introduce jitter in RT threads as long as MMU is met
- Requires highly incremental collector with bounded termination
  - built on "snapshot at the beginning" model (Yuasa)

Henriksson's Collector

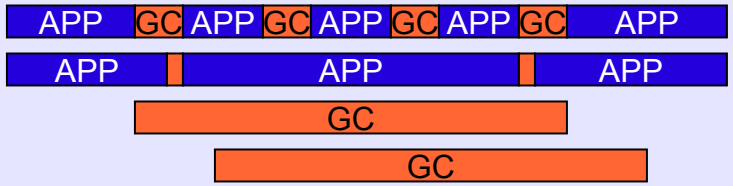
- Technique can be applied to any concurrent collector algorithm
- Scheduling is done by OS using priorities
- Threads that perform collection run at lower priority than any RT thread, higher than any ordinary thread
- Adapts well to MP
- Very good at avoiding jitter in RT threads
- But subject to catastrophic failure if there are any periods with insufficient slack

Metronome TS

- See EMSOFT 2008 for details
- Metronome pauses all threads, global MMU



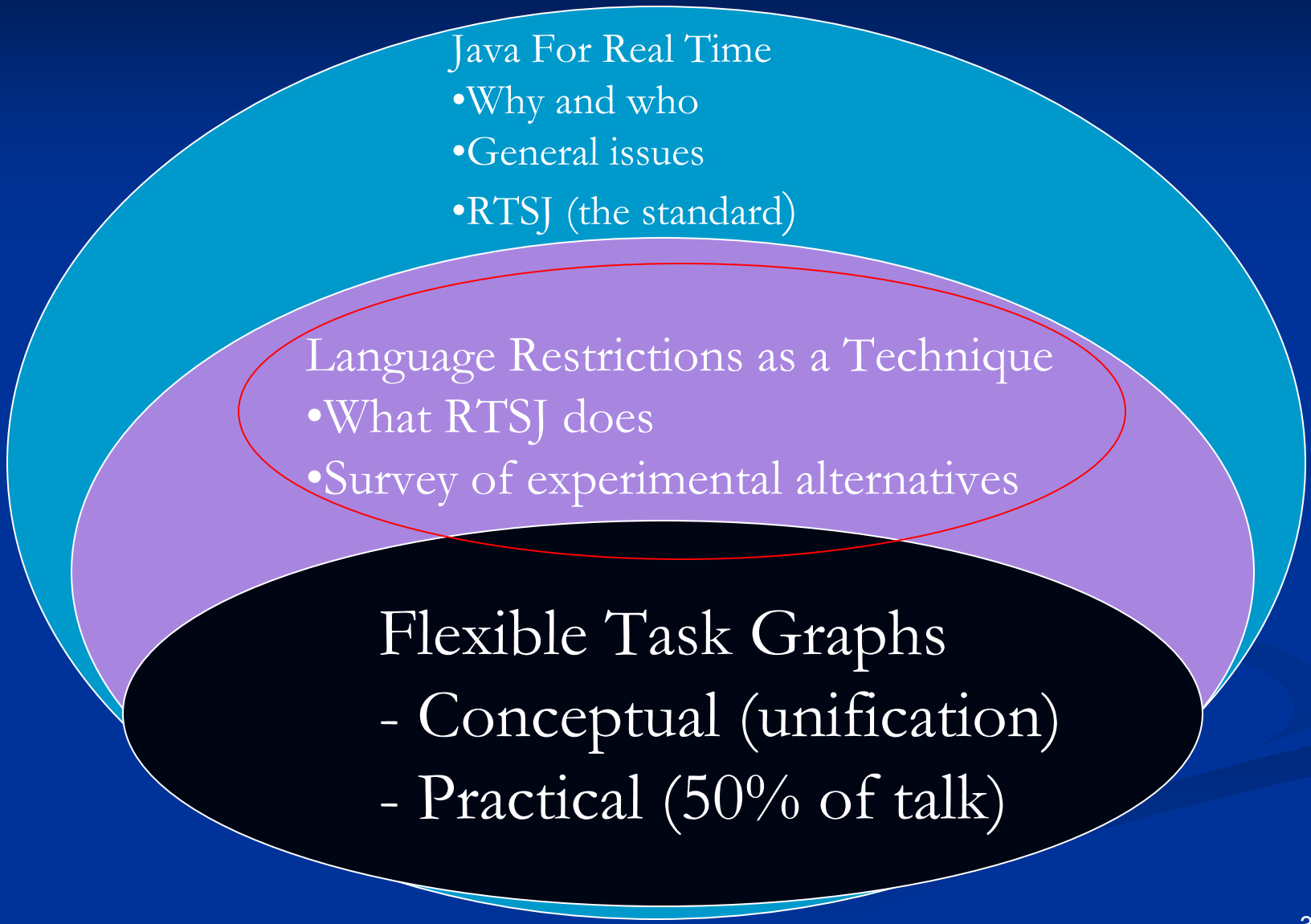
- Metronome-TS schedules GC per-thread
  - Requires GC to be both incremental and concurrent
  - Different threads can have different MMUs
  - “Critical” threads can run with minimal GC interference
  - Background GC threads exploit excess CPU capacity



Limits of RT GC

- Inherent
  - GC work and "real" work are different, context switching inevitable
  - Quantization of GC limited by the need to get something done in each quantum
  - Concurrency only really helps when there are enough cores
- Practical
  - Metronome-TS introduces jitter on the order of 250*u*s
  - Production VMs more like 1-2ms.

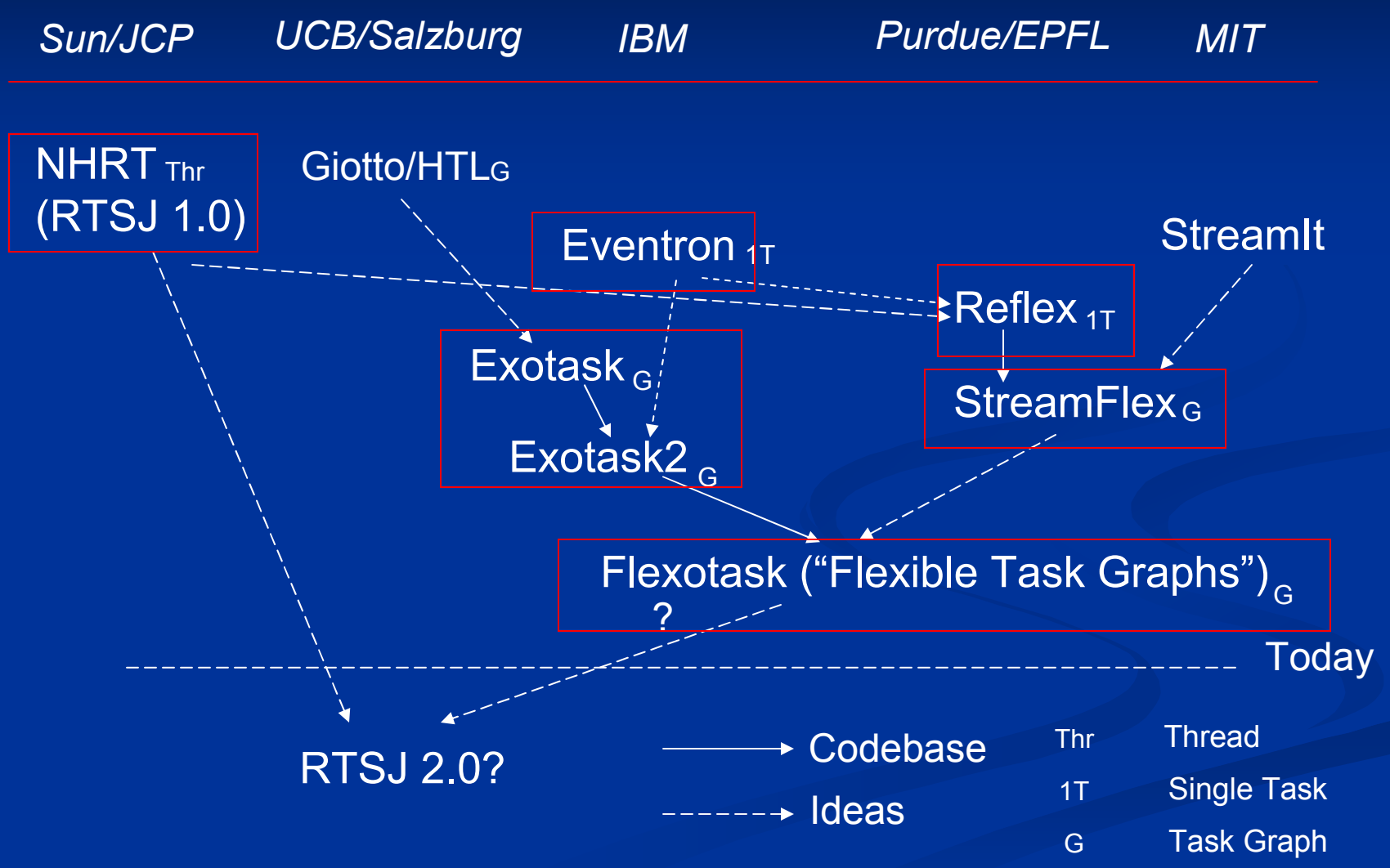
Broad Outline



Language Restrictions + Specialized Memory

- Establishes specialized memory (details vary)
  - These memory areas are not subject to normal GC
- RT-sensitive code (aka *special code*) allocates to specialized memory implicitly
  - Threads running special code not preempted by GC
- Rules (details vary) ensure that references between memory areas remain safe

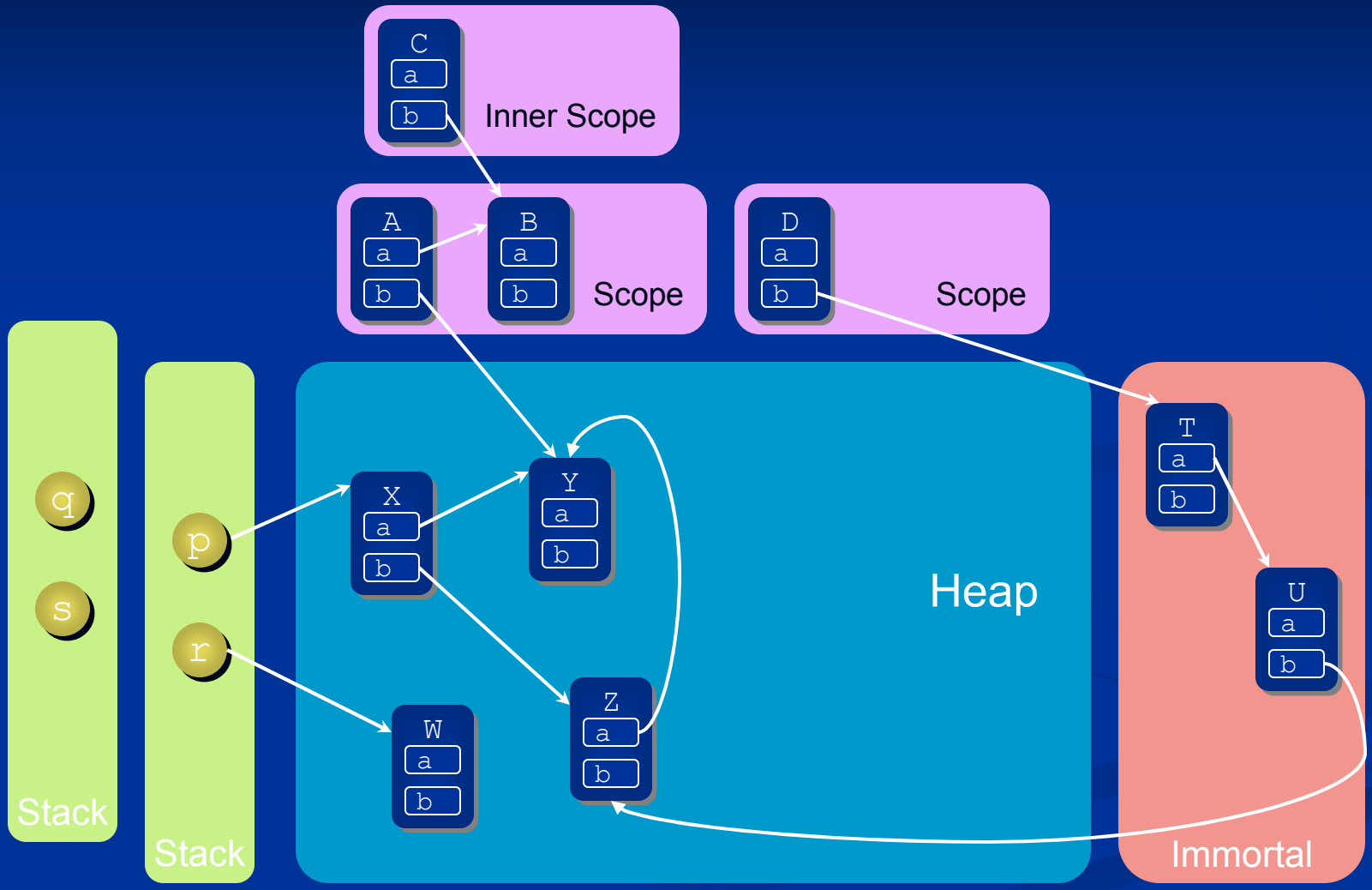
Alternative Models



Specialized Memory in RTSJ

- RTSJ provides two alternatives to using the heap: *immortal* memory and *scoped* memory
- Schedulable objects may enter and leave a specialized memory area
- While they are executing within that area, all memory allocations are performed to the specialized memory
- Immortal memory is *never subject to GC*
- When there are no schedulable objects active inside a *scoped* memory area, the entire area is reclaimed

RTSJ Memory Architecture



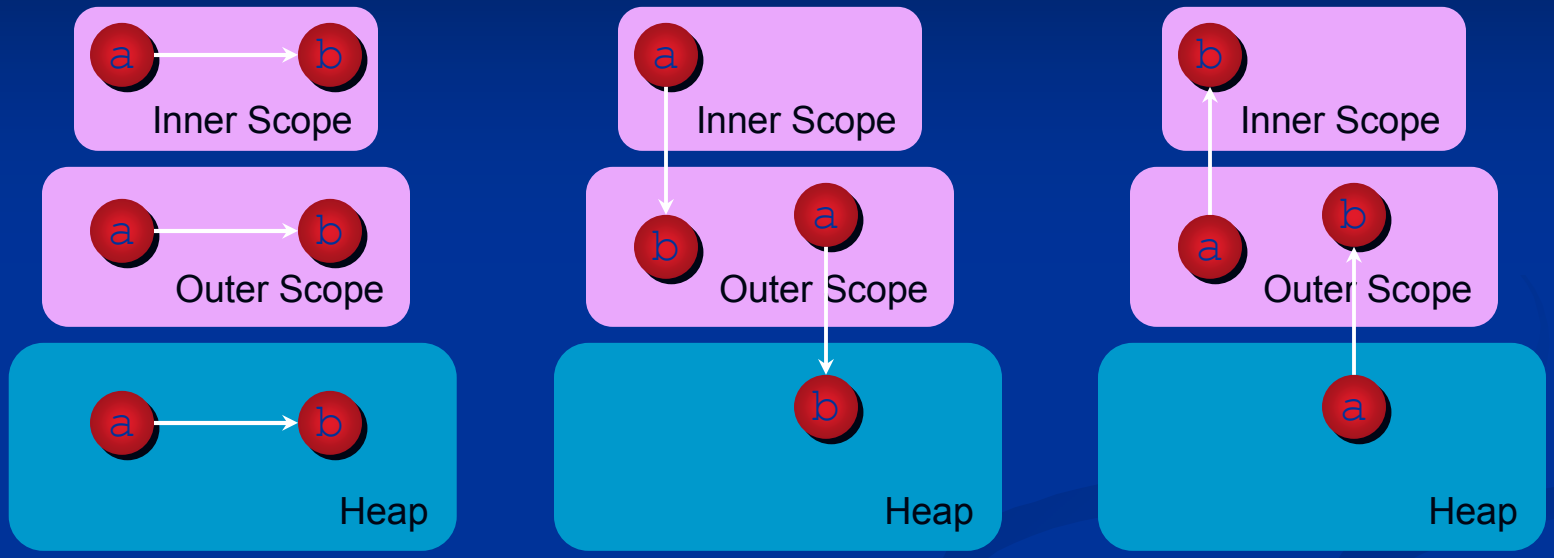
Memory Assignment Rules

From	To Heap Memory	To Immortal Memory	To Scoped Memory
Heap Memory	allowed	allowed	forbidden
Immortal Memory	allowed	allowed	forbidden
Scoped Memory	allowed	allowed	allowed is to same scope or outer scope forbidden if to an inner scope
Local Variable	allowed	allowed	generally allowed

Note

- If the program violates the assignment rules, the unchecked exception `IllegalAssignmentError` is thrown
- Since RTSJ wanted no changes to the Java language rules are enforced *on every assignment at run-time* by the JVM
- In practice, the overhead of the checks is 10 - 20% of execution time if implemented efficiently

```
f(a,b) { a.x = b; }
```



Ok

Ok

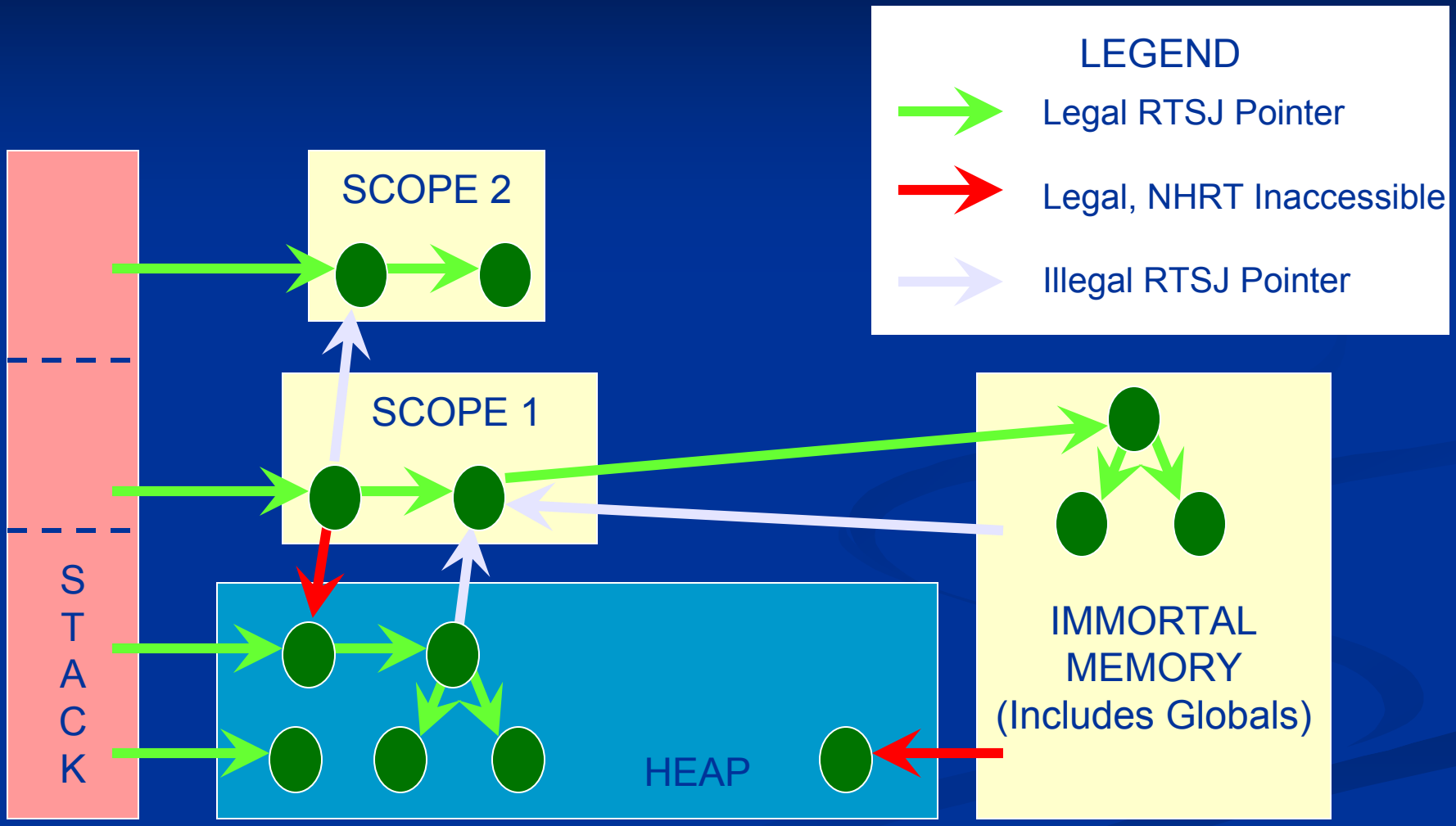
IllegalAssignmentError

Non-compositional behavior. No guarantee that a program that works for now will always work.

NoHeapRealtimeThread

- All code in the application is able to use Immortal and Scoped memory.
- Since such code is also potentially using heap memory, it can be paused by the GC.
- A no-heap real-time thread (NHRT) is a real-time thread which only ever accesses non-heap memory areas.
- Hence, it can safely be executed even when GC is occurring

Adding the NHRT



Attempts to do Better

Common Elements

- Restrict a reachable extent of code, not a thread.
- Use static analysis techniques
  - At development (Java->bytecode) time
  - or at program initialization (or both)
- Avoid dynamic checks
- A program that [compiles | initializes] won't experience an error

http://flexotask.sourceforge.net

The **Flexible Task Graphs** system is described in the following paper, which provides an introduction to our work.

- Flexible task graphs: a unified restricted thread programming model for java by Joshua Auerbach, David F. Bacon, Rachid Guerraoui, Jesper Honig Spring, and Jan Vitek, published in LCTES 2008.

The Flexible Task Graphs open source project, far from reflecting "completed" work, is an active project with much left to accomplish. We expect to accept contributions from new authors. Our goal is to obtain consensus on the best possible restricted thread programming model to incorporate into the next generation of standards for Real Time Java. Contributions can take the form of new plugins (timing grammars, schedulers, distributers, or tracing packages), new front-end tools, or improvements to the framework itself.

- Please contact Joshua Auerbach if you are interested in contributing to this project.
- You may also visit our our sourceforge project page for more information and subscribe to one or more of our mailing lists. The mailing lists are structured so that all subscribers (but only subscribers) may send to them.

The Flexible Task Graphs open source codebase was developed at IBM (or under research agreements with IBM) starting from the older Exotasks code base. It is being made available by IBM under the Eclipse Public License. Code was written by Joshua Auerbach and Jesper Honig Spring, with design guidance from David Bacon, Jan Vitek, and Rachid Guerraoui, and reflecting earlier design guidance on the Exotask system from Christoph Kirsch. Flexible Task Graphs are a unification of four previous programming models, so the authors of those models can all be counted as having influenced the present model. Here are pointers to the papers describing the original models that influenced this work.

- Eventrons: a Safe Programming Construct for High-frequency Hard Real-time Applications by D. Spoonhower, J. Auerbach, D. F. Bacon, P. Cheng, and D. Grove, published in PLDI 2006.
- Java Takes Flight: Time-portable Real-time Programming with Exotasks by J.Auerbach, D. F. Bacon, D. T. Iercan, C. M. Kirsch, V. T. Rajan, H. R. Roeck, and R. Trummer, published in LCTES 2007
- Low-Latency Time-Portable Real-Time Programming with Exotasks by J.Auerbach, D. F. Bacon, D. T. Iercan, C. M. Kirsch, V. T. Rajan, H. R. Roeck, and R. Trummer, published in TECS. A mo detailed treatment of Exotasks that also described the integration with Eventrons, a precursor to this work.
- Reflexes: abstractions for highly responsive systems by J. Spring, F. Pizlo, R. Guerraoui, and J. Vitek, published in VEE 2007.
- Streamflex: high-throughput stream programming in java by J. Spring, J. Privat, R. Guerraoui, J. Vitek, published in OOPSLA 2007

Pointers to all the papers are on our web site.

Techniques Differ

- Kind of specialized memories
  - None, heap-like, immortal-like, scope-like, combinations…
- When restrictions enforced
  - Compile time (earlier feedback)
  - Program initialization (more information)
- How special code communicates with non-RT code
  - Important since most "RT" applications are mixtures
- Other details

Single Task Models

- RTSJ NoHeapRealtimeThread
- Eventron (IBM), aka Expedited Real-time Threads
- Reflex (Purdue / EPFL)
- Exotask (IBM)
- Exotask 2 (IBM), also called Expedited Real-time Task Graphs
- StreamFlex (Purdue / EPFL)
- Flexible Task Graphs (IBM, Purdue, EPFL)

Eventron

- IBM (PLDI 2006)
- Kind of specialized memories
  - None! Relies entirely on pinned objects in public heap
  - Safe because the pinned objects are linked by **final** references so form a permanently closed set.
- When restrictions enforced
  - Program initialization. Analysis also yields the set of objects that must be pinned.

Eventron

- How special code communicates with non-RT code
  - Pinned objects non-reference fields (primitive type, including elements of primitive arrays) are still mutable
  - Essentially unsynchronized, ameliorated by **notifyIfWaiting** facility
- Other details
  - Because no specialized memory, Eventron code cannot allocate at all.
  - Synchronization also forbidden

Reflex

- Purdue / EPFL (VEE 2007)
- Kind of specialized memories
  - Each reflex has two: immortal-like *stable-area*, and scope-like *transient area*
- When restrictions enforced
  - Compile time. Dangling stable->transient pointers eliminated by pure type-based analysis: divide into **Stable** and transient classes.

Reflex

- How special code communicates with non-RT code
  - Pre-emptive atomic regions in methods callable by both special and non-special code: special code wins and aborts non-special
- Other details
  - Treatment of **static**-reachable objects reuses Eventron insights ("reference immutable")
  - Compile-time analysis doesn't yield "objects to be pinned" … relies on VM to run class initializers in immortal

Task Graph Models

- Exotask
  - Influenced by Giotto and HTL (UCB / Salzburg)
  - Original idea was to achieve perfect time portability, determinism
  - Originally incompatible with Eventrons
- StreamFlex
  - Built on Reflex memory model
  - Influenced by StreamIt (MIT)

Exotask 1

- IBM (LCTES 2007)
- Kind of specialized memories
  - Private heap, with *scheduled* garbage collection
- When restrictions enforced
  - Program initialization
    - but data-sensitive only for **static**-reachable fields
  - Static-accessible data pinned as in Eventrons; must be *immutable*, not just reference-immutable
  - Synchronization on these pinned immutable objects is no-op'd.

Exotask 1

- How special code communicates with non-RT code
  - Doesn't! Exploring high levels of determinism, time portability, which requires total isolation
- Communication between nodes of the graph is by *deep clone*
  - Expensive but guarantees isolation

Key Exotask Ideas in Flexotask

- Pluggable schedulers. The scheduler decides
  - when each task runs
  - when each task is garbage-collected
  - when every connection copies its data
- Instantiation from a template (XML document)
  - The template is a declaration of the graph's structure
  - The actual task graph is instantiated by the system
- Pluggable timing grammars
  - Provide the right timing semantics for a given scheduler
  - Template syntax includes timing annotations conforming to the declared grammar

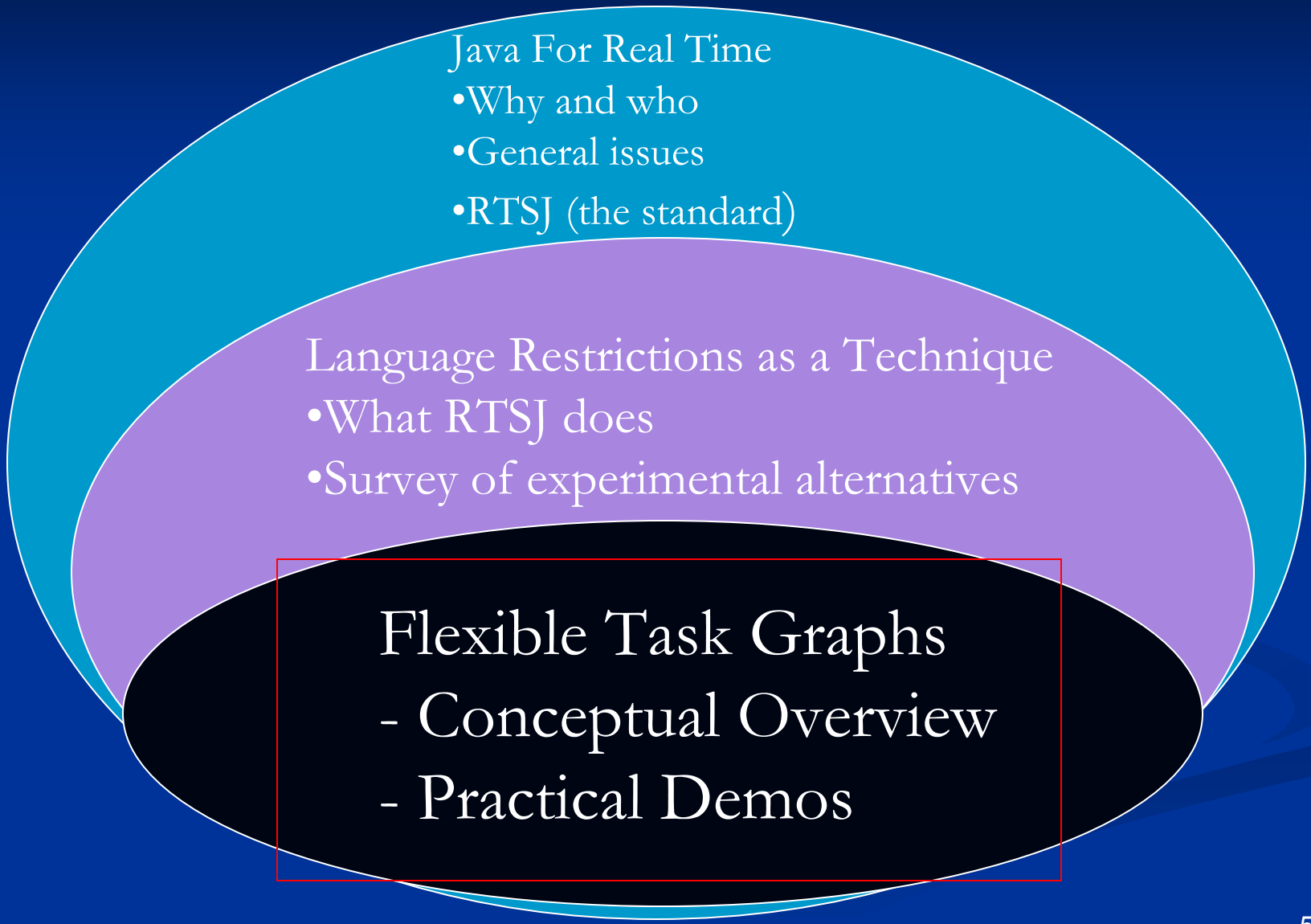
Exotask 2 (aka XRTG)

- IBM (TECS 2009)
- Unified Exotask and Eventron ideas
- "Weak isolation" option relaxes normal Exotask restriction on static to "reference immutable," as in Eventron and Reflex models.
- Also permits a public-heap pinned "parameter" to be passed to each weakly isolated Exotask.

StreamFlex

- Purdue / EPFL (OOPSLA 2007)
- Multi-node extension of Reflex model
- Same memory model for individual Reflex node
- In contrast to Exotask
  - Communication channel is a bounded buffer
  - Pass-by-reference semantics
    - Introduce a third category of classes: **Stable**, transient and **Capsule**
    - **Single-reference invariant supports efficient freeing**

Broad Outline



Flexotask Goals

- Unify previous work (statically checkable models).
- Provide a basis for serious standardization discussions.
- Provide a useful framework for further development
  - Development environment based on Eclipse
  - Leverage Eclipse's plugin concept
  - Well-defined boundary between programming model support and the RT VM.

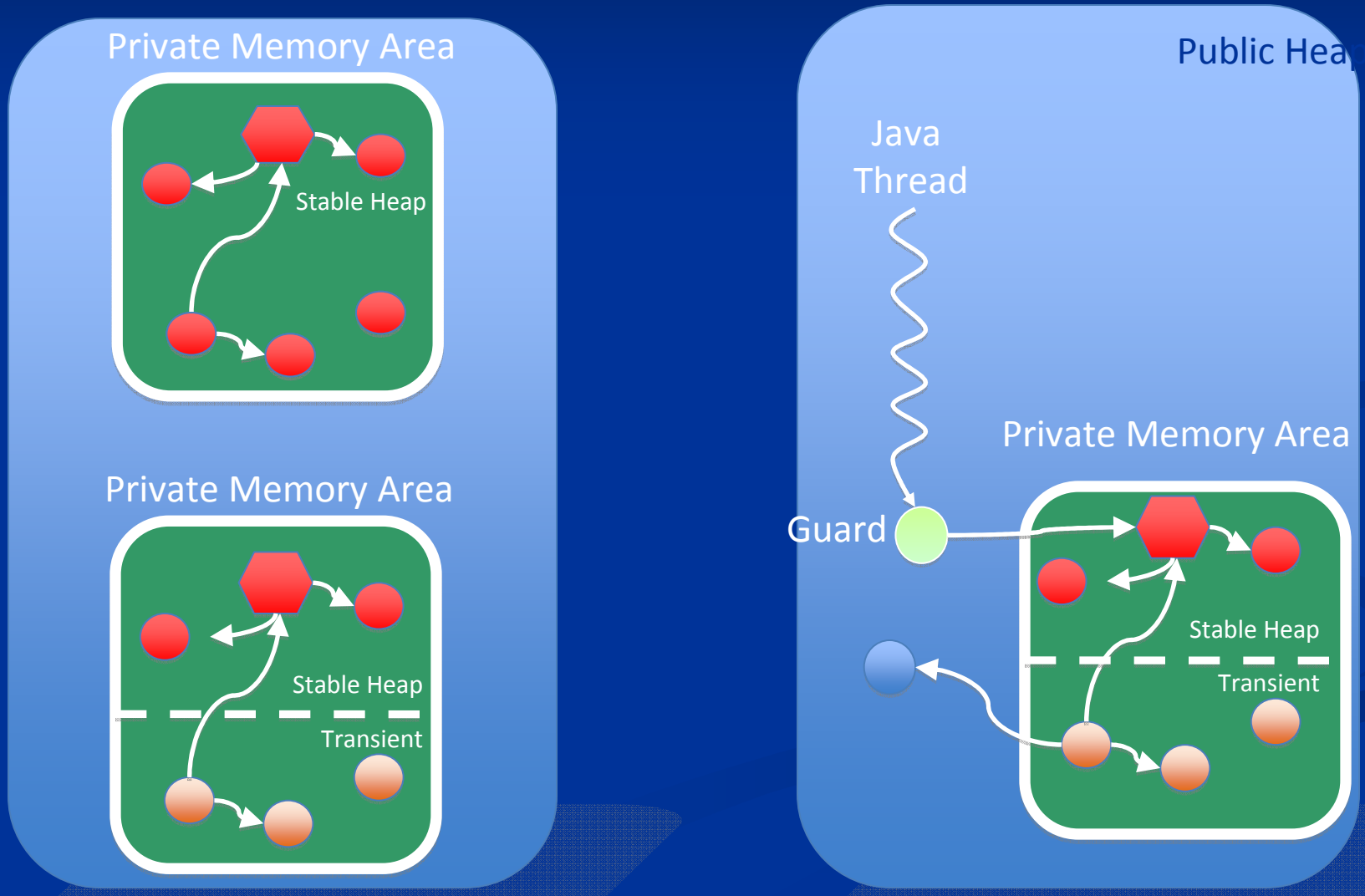
Flexotask Unification

- Kind of specialized memories
  - Each Flexotask has a private heap (like Exotask)
  - …optional transient area (like Reflex)
  - …supplemented by pinned objects (like Eventron)
- When restrictions enforced
  - *Both at* compile time (earlier feedback)
  - *And at* program initialization (more information)
  - Correctness defined by the later check, so earlier check may be just a warning
  - Stable/transient distinction as in Reflexes
    - But, an "all stable" default yields Exotask-like behavior
  - Optional restrictions on allocation and synchronization

Flexotask Unification

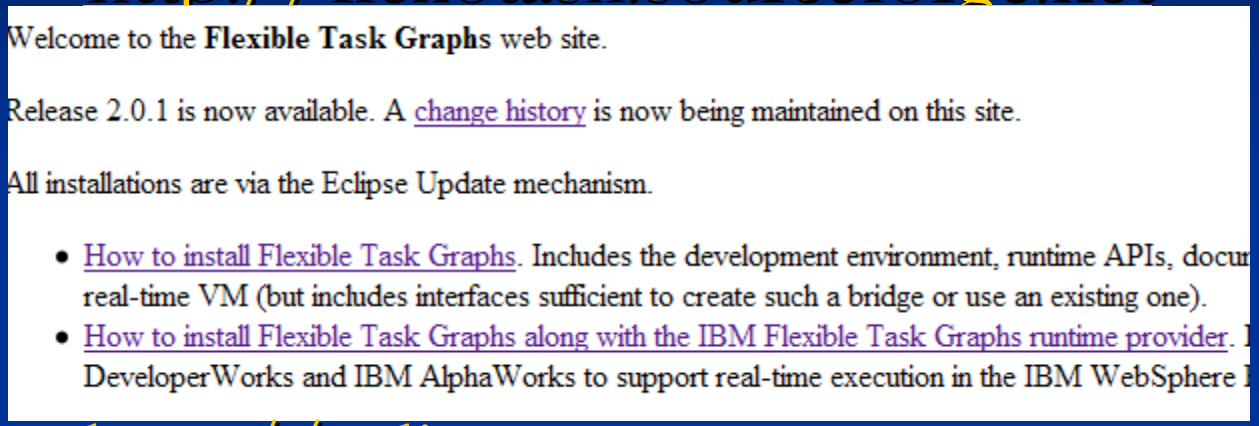
- How special code communicates with non-RT code
  - Eventron-style (unsynchronized access through shared non-reference fields).
  - Reflex-style (pre-emptive atomic regions)
    - Accomplished through rewriting at development time.
- Has pluggable scheduling, timing grammars, template approach as in Exotasks.
- Task graphs as in Exotask and StreamFlex
  - Deep copy option (Exotask-like)
  - By-reference option for shared pinned objects (StreamFlex-like)
  - Connection buffering accomplished by making the ports at either end buffered.

Flexotask Memory Picture



Flexible Task Graphs Software

- http://flexotask.sourceforge.net


Welcome to the **Flexible Task Graphs** web site.

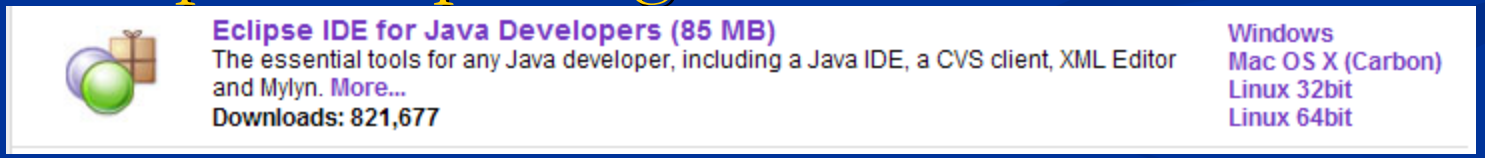
Release 2.0.1 is now available. A change history is now being maintained on this site.

All installations are via the Eclipse Update mechanism.

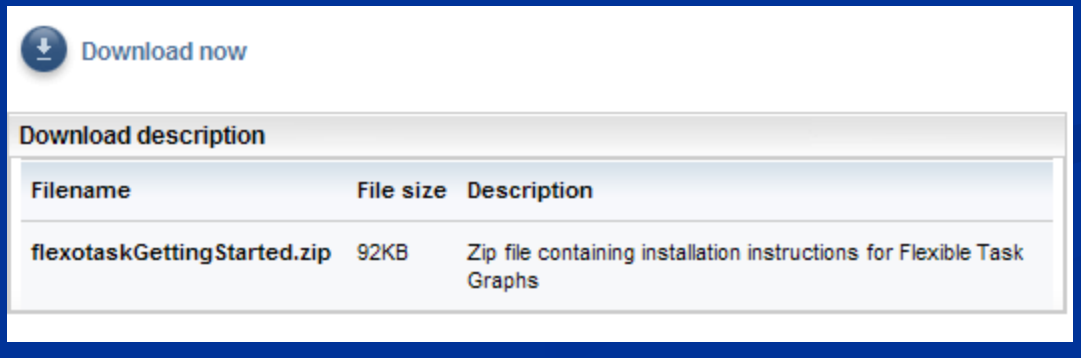
- How to install Flexible Task Graphs. Includes the development environment, runtime APIs, docu real-time VM (but includes interfaces sufficient to create such a bridge or use an existing one).
- How to install Flexible Task Graphs along with the IBM Flexible Task Graphs runtime provider. DeveloperWorks and IBM AlphaWorks to support real-time execution in the IBM WebSphere

API classes, development tools, debug/test, checkers, builders, etc. Open source.

- http://eclipse.org



- http://www.alphaWorks.ibm.com/tech/flexotasks



Runtime bridge to a specific Real-time JVM (IBM's WebSphere Real Time). Binary only.

Flexible Task Graphs Software

- http://www.ibm.com/developerworks/java/jdk/linux/download.html

Java SE Version 6

	Download	Plug-in support	Web Start support
64-bit AMD/Opteron/EM64T	SR4	No	Yes
32-bit xSeries (Intel compatible)	SR4	Yes	Yes
32-bit iSeries/pSeries	SR4	Yes	Yes
64-bit iSeries/pSeries	SR4	Yes	Yes
31-bit zSeries (S/390)	SR4	No	No
64-bit zSeries (S/390)	SR4	No	No
WebSphere Real Time V2.0 32-bit xSeries (Intel compatible)	SR1	No	No

IBM WebSphere Real Time itself … free download (no support, though, support costs $$ ☺) … requires a recent RT Linux kernel such as RHEL5RT

Flexotask Runtime Structure

API Classes (also support for provided schedulers and timing grammars)

Packages	
com.ibm.realtime.flexotask	Contains the types needed to write the Flexotasks that make up a Flexible Task Graph
com.ibm.realtime.flexotask.scheduling	Contains interfaces to be implemented by schedulers and interfaces provided to schedulers.
com.ibm.realtime.flexotask.template	Defines the Flexible Task Graph *template* classes and their API
com.ibm.realtime.flexotask.timing	Contains interfaces to be implemented when adding a new *timing grammar* to the Flexotask System
com.ibm.realtime.flexotask.tools	Provides tools for converting between the XML and programmatic representation of Flexotask templates
com.ibm.realtime.flexotask.tracing	Provides interfaces to be implemented when tracing (instrumenting) Flexible Task Graphs or schedulers

Common Implementation Layer

FlexotaskVMBridge: provides standard interface to VM

Flexotask-enabled JVM. Does *not* have to implement RTSJ, but it is useful in practice to leverage RTSJ building blocks.

Utility natives for debug, I/O, event handling

RT-enabled Operating System (we consider only RT Linux at this point)

Preliminary (non-RT) Testing

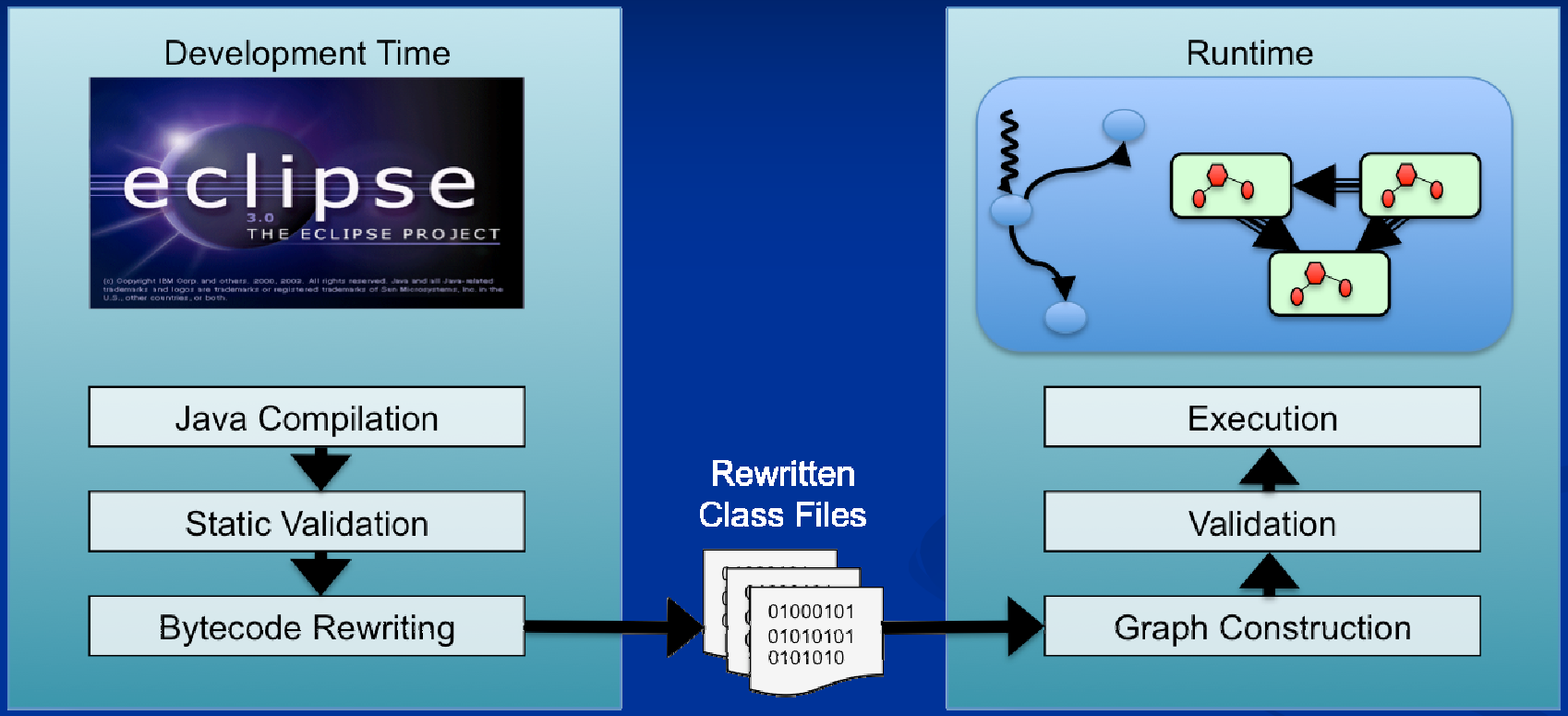
API Classes (also support for provided schedulers and timing grammars)

Packages	
com.ibm.realtime.flexotask	Contains the types needed to write the Flexotasks that make up a Flexible Task Graph
com.ibm.realtime.flexotask.scheduling	Contains interfaces to be implemented by schedulers and interfaces provided to schedulers.
com.ibm.realtime.flexotask.template	Defines the Flexible Task Graph *template* classes and their API
com.ibm.realtime.flexotask.timing	Contains interfaces to be implemented when adding a new *timing grammar* to the Flexotask System
com.ibm.realtime.flexotask.tools	Provides tools for converting between the XML and programmatic representation of Flexotask templates
com.ibm.realtime.flexotask.tracing	Provides interfaces to be implemented when tracing (instrumenting) Flexible Task Graphs or schedulers

Common Implementation Layer

Any VM, on any platform, running under Eclipse or not.
Supports full API without RTguarantees.

Usage Overview



1) Dev.time code validation (on bytecodes, not source)

2) Code rewriting enabling transactional methods (if needed)

3) Graph construction

4) Startup-time validation

5) Execution

Demo

On Linux

- Unzip runtime
- Set FTG_HOME to point to it
- Set WRT_HOME to point to WebSphere Real Time
- Place test.jar in a convenient place
- Run the application:
  - $FTG_HOME/bin/ejava –cp test.jar test.Main

Console Output

```
/gsa/yktgsa/home/j/s/jsa:  $FTG_HOME/bin/ejava -cp test.jar test.Main
200 199 198 197 196 195 194 193 192 191 190 189
New time: 0
188 187 186 185 184 183
New time: 500
182 181 180 179
New time: 1000
178 177 176 175 174 173
New time: 1500
172 171 170 169 168 167 166 165
New time: 2000
164 163 162 161 160 159
New time: 2500
158 157 156 155 154
New time: 3000
153 152 151 150
New time: 3500
149 148 147 146
New time: 4000
145 144 143 142 141 140
New time: 4500
139 138 137
New time: 5000
136 135 134 133 132
New time: 5500
131 130 129 128
New time: 6000
127 126 125 124 123 122 121
New time: 6500
120 119 118 117 116
New time: 7000
115 114 113 112
New time: 7500
111 110 109 108 107 106
New time: 8000
105 104 103
New time: 8500
102 101 100 99 98
New time: 9000
97 96 95 94 93
New time: 9500
92 91 90 89 88 87 86
```

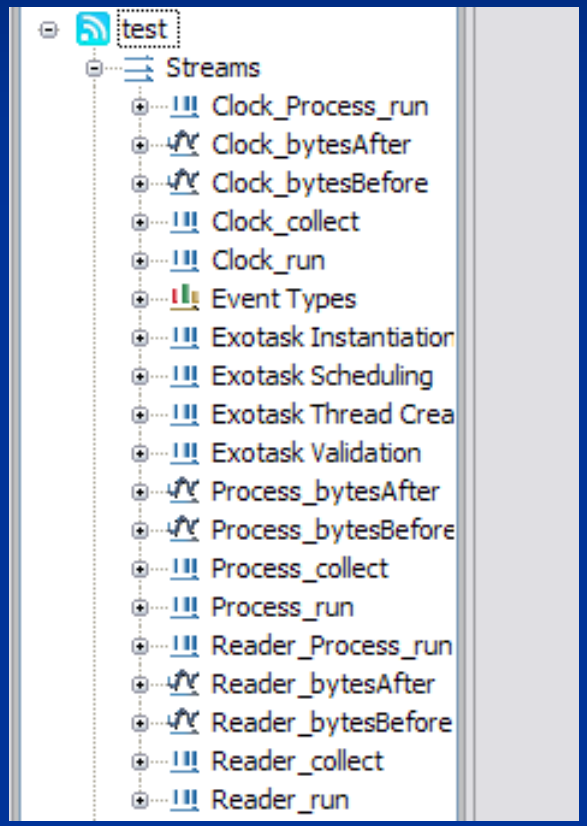
```
New time: 10000
85 84 83
New time: 10500
82 81 80 79 78 77
New time: 11000
76 75 74 73
New time: 11501
72 71 70 69
New time: 12001
68 67 66 65 64 63 62
New time: 12501
61 60 59 58 57 56
New time: 13001
55 54 53 52 51 50 49 48
New time: 13501
47 46 45 44
New time: 14001
43 42 41 40 39
New time: 14501
38 37 36 35 34
New time: 15001
33 32 31 30 29 28
New time: 15501
27 26 25 24 23
New time: 16001
22 21 20 19 18
New time: 16501
17 16 15 14 13
New time: 17001
12 11 10 9
New time: 17501
8 7 6 5
New time: 18001
4 3 2 1
New time: 18501
0
-1
/gsa/yktgsa/home/j/s/jsa:
```

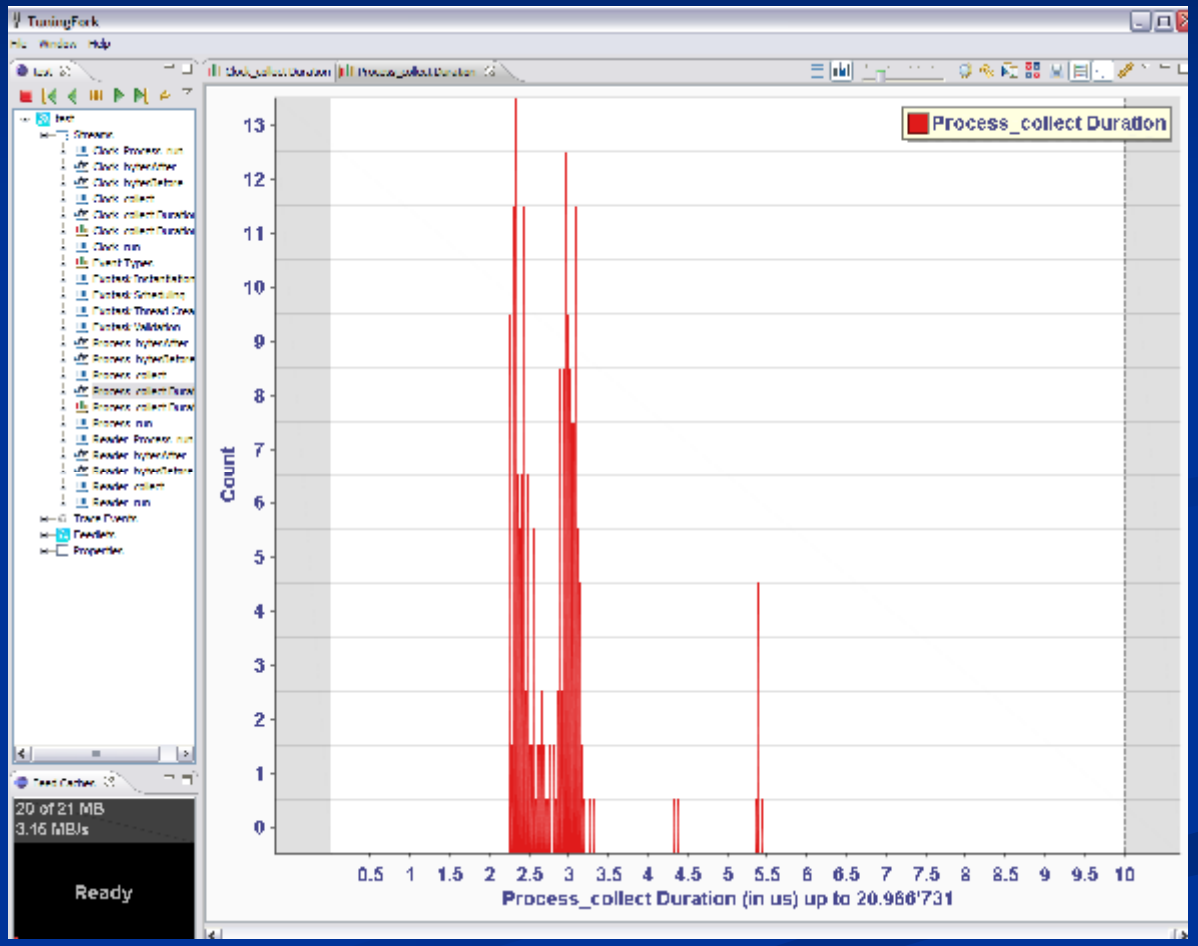
Add Tracing

```
/* Load development-time template into runtime template (multiple ways
FlexotaskTemplate template = FlexotaskXMLParser.parseStream(Main.class
/* Add TuningFork tracing */
TracingSupport.setTracer(new TFTracerFactory("test.trace"));
/* Validate the graph */
FlexotaskGraph graph = template.validate("ReactiveScheduler", inits);
```

- General framework for tracing
  - Implement **FlexotaskTracerFactory** – it decides which tasks and connections shall be traced and what shall be done at each trace point.
- TuningFork tracing plugin for Flexotasks (and TuningFork itself) available from SourceForge
- TuningFork can trace to a file or a socket.
- Custom tracers can be written to store information in memory, dump later.

TuningFork


test
Streams
Clock_Process_run
Clock_bytesAfter
Clock_bytesBefore
Clock_collect
Clock_run
Event Types
Exotask Instantiation
Exotask Scheduling
Exotask Thread Crea
Exotask Validation
Process_bytesAfter
Process_bytesBefore
Process_collect
Process_run
Reader_Process_run
Reader_bytesAfter
Reader_bytesBefore
Reader_collect
Reader_run


TuningFork
20 of 21 MB
3.16 MB/s
Ready
Process_collect Duration
Count
0 1 2 3 4 5 6 7 8 9 10 11 12 13
0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5 7 7.5 8 8.5 9 9.5 10
Process_collect Duration (in us) up to 20.966'731

Developing a Scheduler

Conclusion

- http://flexotask.sourceforge.net
- Use it for experimentation with
  - RT applications in Java
  - Scheduling algorithms
- Use it as a back end for RT modeling
- Contribute to the open source project!
- RT Java is real