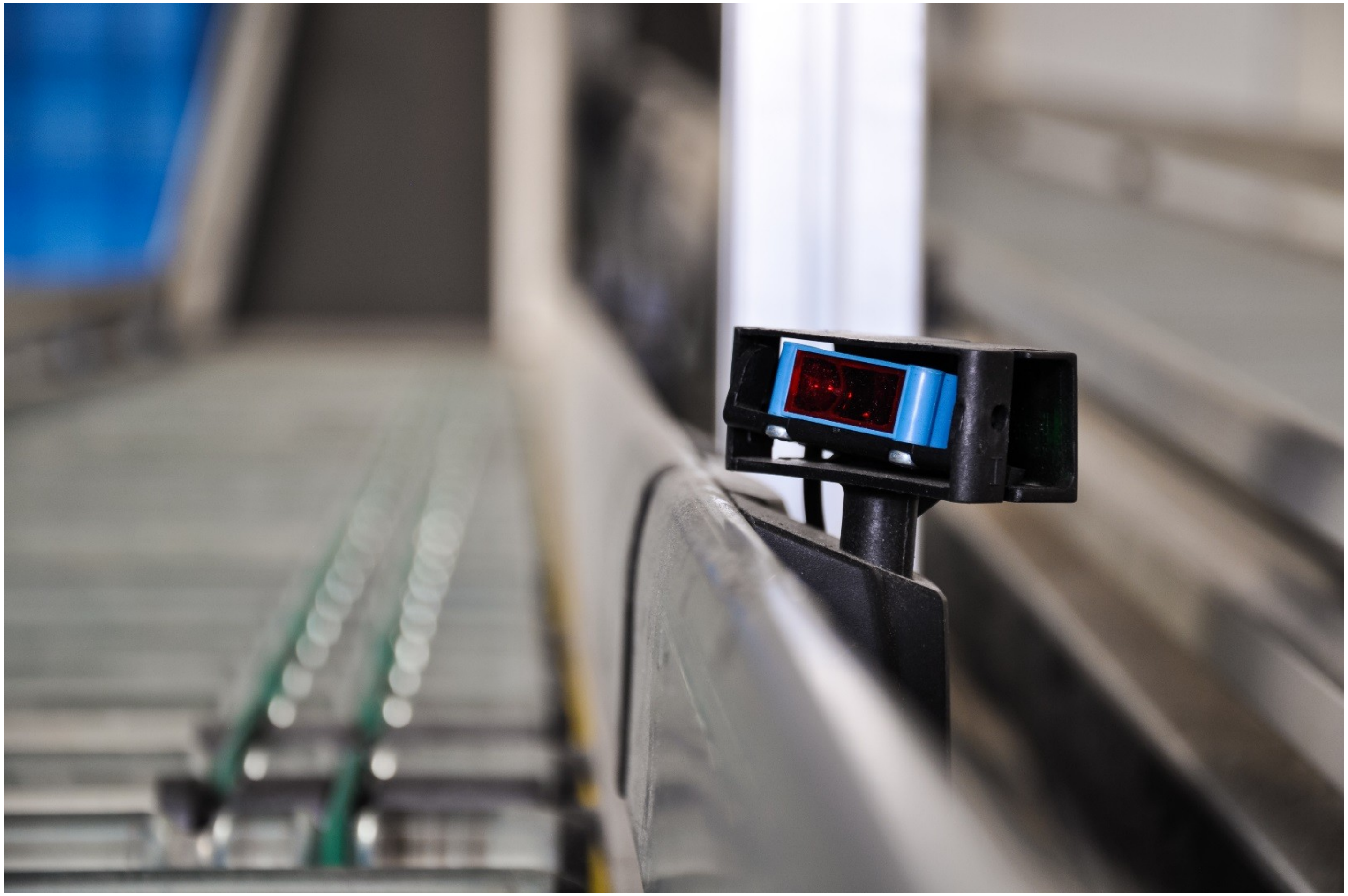
JTRES 2016 Lugano

The New Realtime Specification for Java and the Future of IoT

Dr. James J. Hunt
JSR 282 Spec Lead
CEO aicas GmbH

Internet of Things



Performance measurment via Embedded Sensors

Internet of Things

Pre-analysis: Information not Data Collection

Internet of Things

Remote Device Supervision and Control

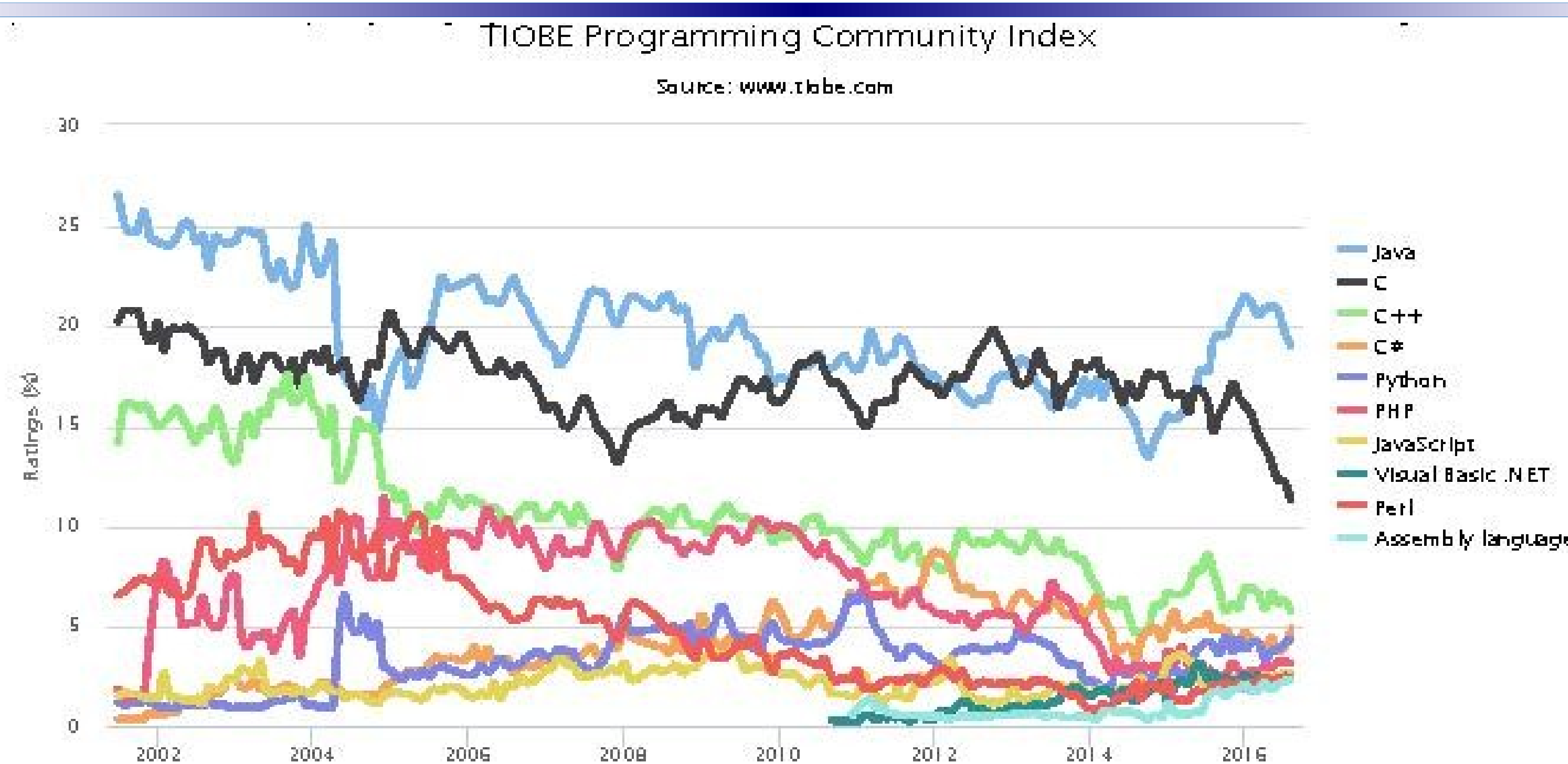
Internet of Things

Flexibility via Modular Functional Update

Trends for the Intenet of Things

Strong Security and Encryption Support

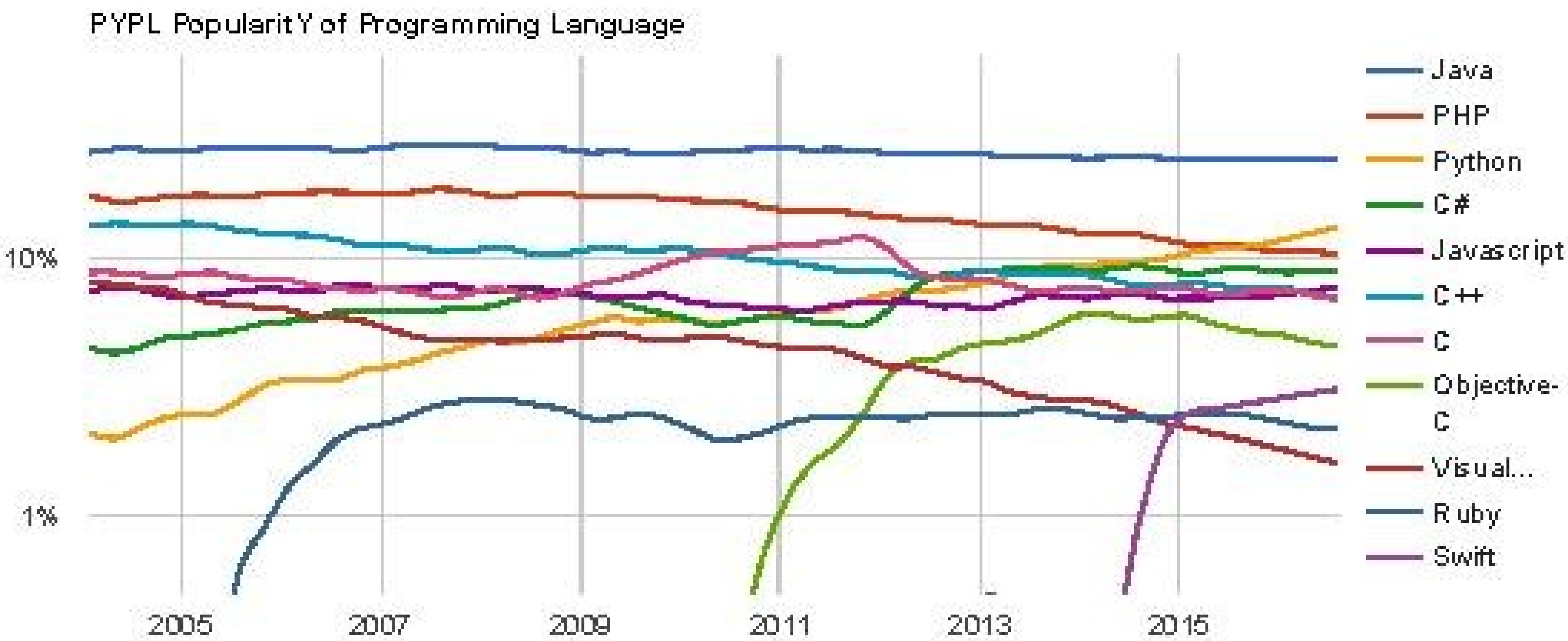
Java: Once and Future King



TOBIE Worldwide

- Java is the most popular language (19.1%) and
- C is second (11.3%)

Java: Long Term Favorite



PYPL Worldwide

- Java is the most popular language (24.1%)
- Python grew the most in the last 5 years (6.9% to 13.2%) and
- C lost the most (-4.7%)

Why not Use Java for IoT?

Many Pros

- Higher abstraction improve productivity
- Strong typing improve safety
- Exact garbage collection improves safety
- Support for dynamically loaded code (OSGi)
- Well defined security model
- Strong cryptography support
- Most popular language for new development

What is the Problem?

Poor Support for Realtime

- Scheduling not defined for JVM (assumes fair scheduling)
- Priorities are ill defined
- Synchronization is not aware of thread priorities
- Single tasking model: Threads
- Throughput optimized garbage collection

A Realtime Specification for Java is necessary

Why Realtime?

Realtime is not real fast!

- controlling physical objects requires predicable and timely response
  - Realtime tasks must always complete within their deadline
  - Often minimum response time are also necessary
- Example: steering
  - understeering risks collision and instability
  - oversteering risks efficiency lose and instability
- Nonexamples: video and audio playback

What is the RTSJ?

Support for realtime programming in Java

- importance vs fair scheduling
- determinism vs responsiveness
- timeliness vs throughput
- priority inversion avoidance vs antistarvation

Support for embedded programming in Java

- device access
- interact with environment

A standard refinement of JVM semantics

Constraints

No changes to the language

- same bytecode
- no new language keywords

Fully compatible with convention Java implementations such as OpenJDK

- Java programs must run correctly on RTSJ implementations that supports the required profile
- Maximize code reuse under time complexity constraints

Why Update the RTSJ?

Language and technology evolution

- better realtime garbage collection
- Java 1.4 → Java 1.8

Marketing

- support different levels of realtime
- reduce need for JNI
- de-emphisize memory areas
- differential to Android
- better documentation

Core Features

Ada realtime semantics in Java

- realtime threads
- priority preemptive scheduling
- priority inversion avoidance

Event handling

- periodic tasks (Timers)
- aperiodic tasks

Direct device access in Java

Major RTSJ 2.0 Improvements

Core

- CPU affinity
- Task control groups
- Unify Events: Timer, Happening, & Signal
- Stateful events
- User defined clocks

POSIX

- Realtime Signals

Device Access

- Typed device access
- Factory based
- DMA & ISR support

Alternate Memory

- PinnableMemory
- StackedMemory
- Physical memory factory

Schedulables

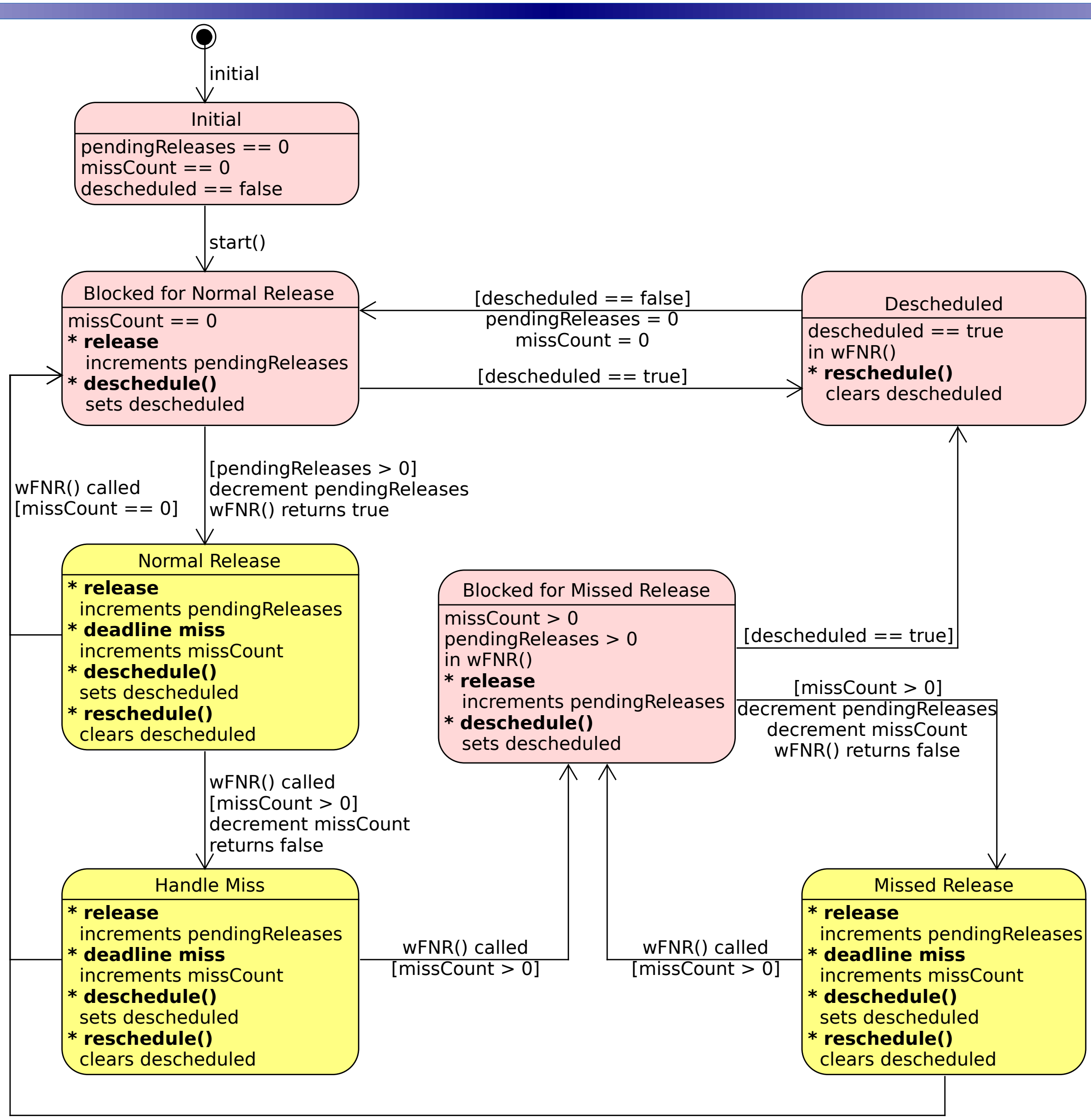
RealtimeThread

- Modeled on java thread (subclass)
- Realtime scheduled
- explicit looping using waitForNextRelease.
- triggering
  - by call to release()
  - implicit for Clocks

AsyncEventHandler

- Event model
- May have payload
- Realtime scheduled
- implicit looping; code in handleAsyncEvent
- triggering
  - implicit by events
  - Timer is an event

Periodic RealtimeThread



Scheduling

Realtime Schedulers

- FirstInFirstOutScheduler
- RoundRobinScheduler
- Both types of PriorityScheduler

Java Thread Control

- enable java threads to use a realtime scheduler

Synchronization (Priority Inversion Avoidance)

- Priority inheritance
- Priority ceiling emulation

Affinity

Multicore control

- Enable pinning Threads and AsyncEventHandlers to a subset of processors.
- support collective pinning w/ ProcessingGroupParameters
- Find out what processors and processor subsets are available for pinning
- Pinning to single processors is always supported.
- Orthogonal to all other RTSJ classes

Task Groups

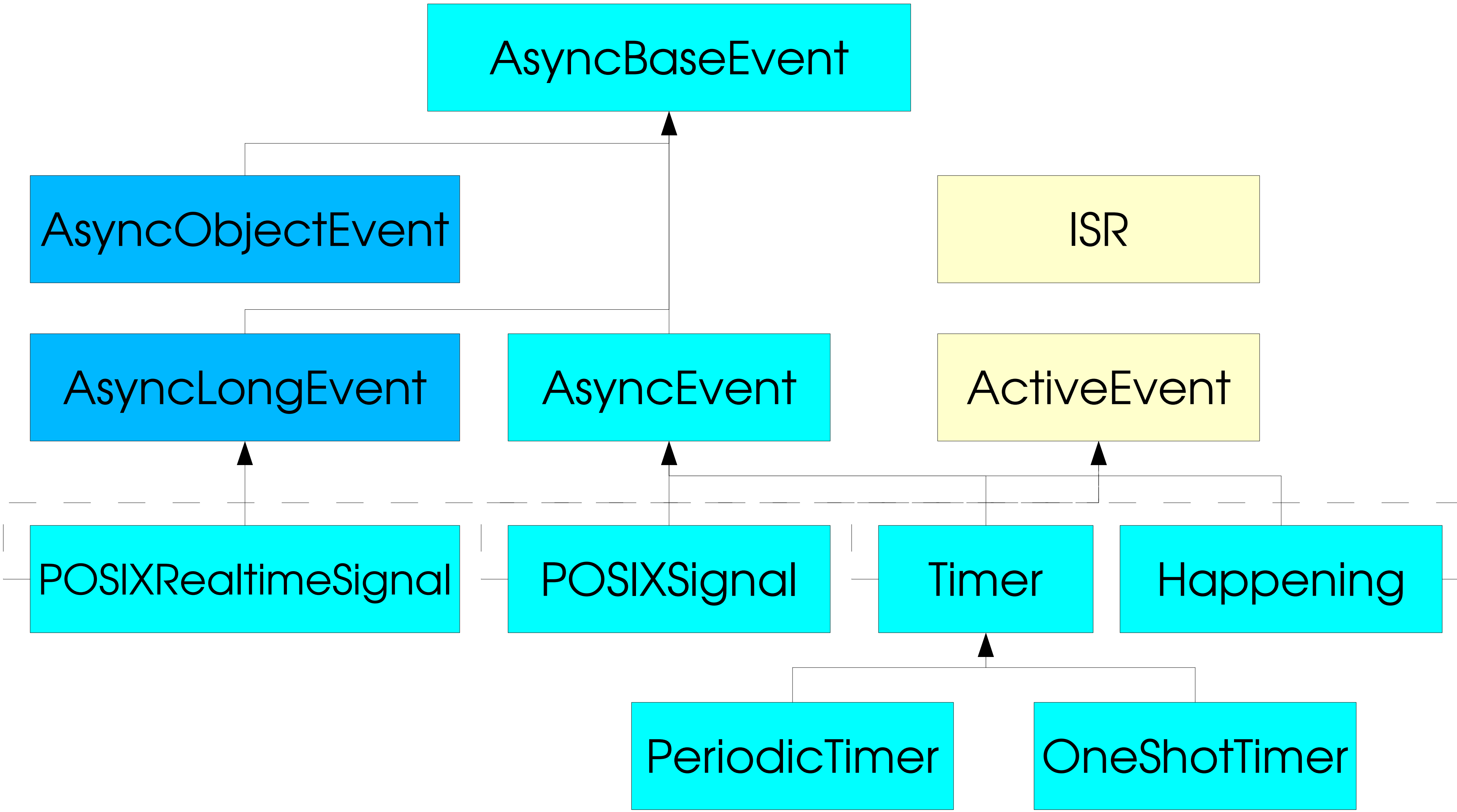
Extended through Subclasses of ThreadGroup

- SchedulingGroup — realtime scheduling
- ProcessingGroup — deadlines, overrun, underrun
- MemoryGroup — memory limits

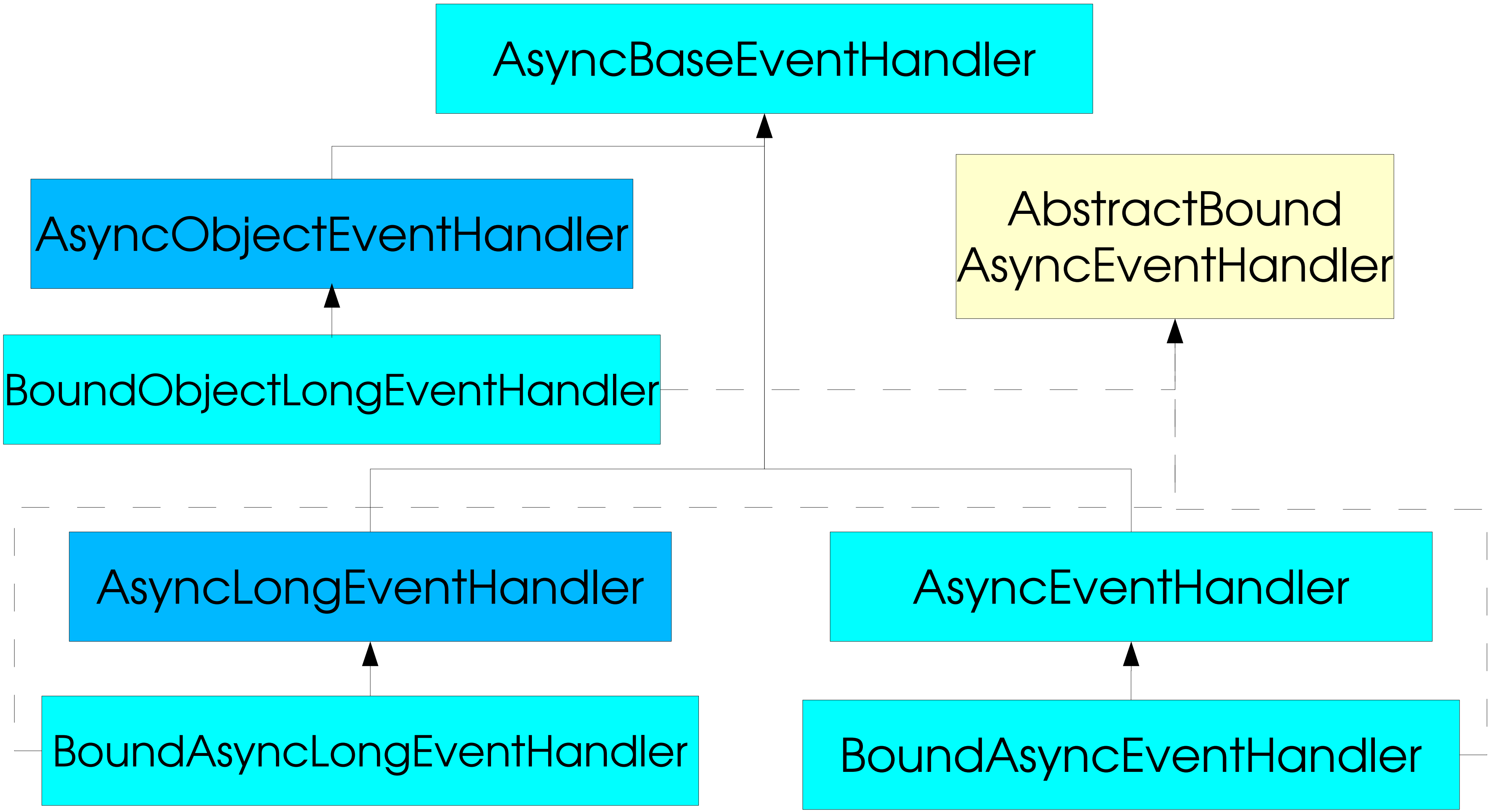
New Rules

- Schedulables must be in a SchedulingGroup (Primordial and initial TG must be SG too!)
- Threads in a base ThreadGroup may not have realtime characteristics

Event Architecture



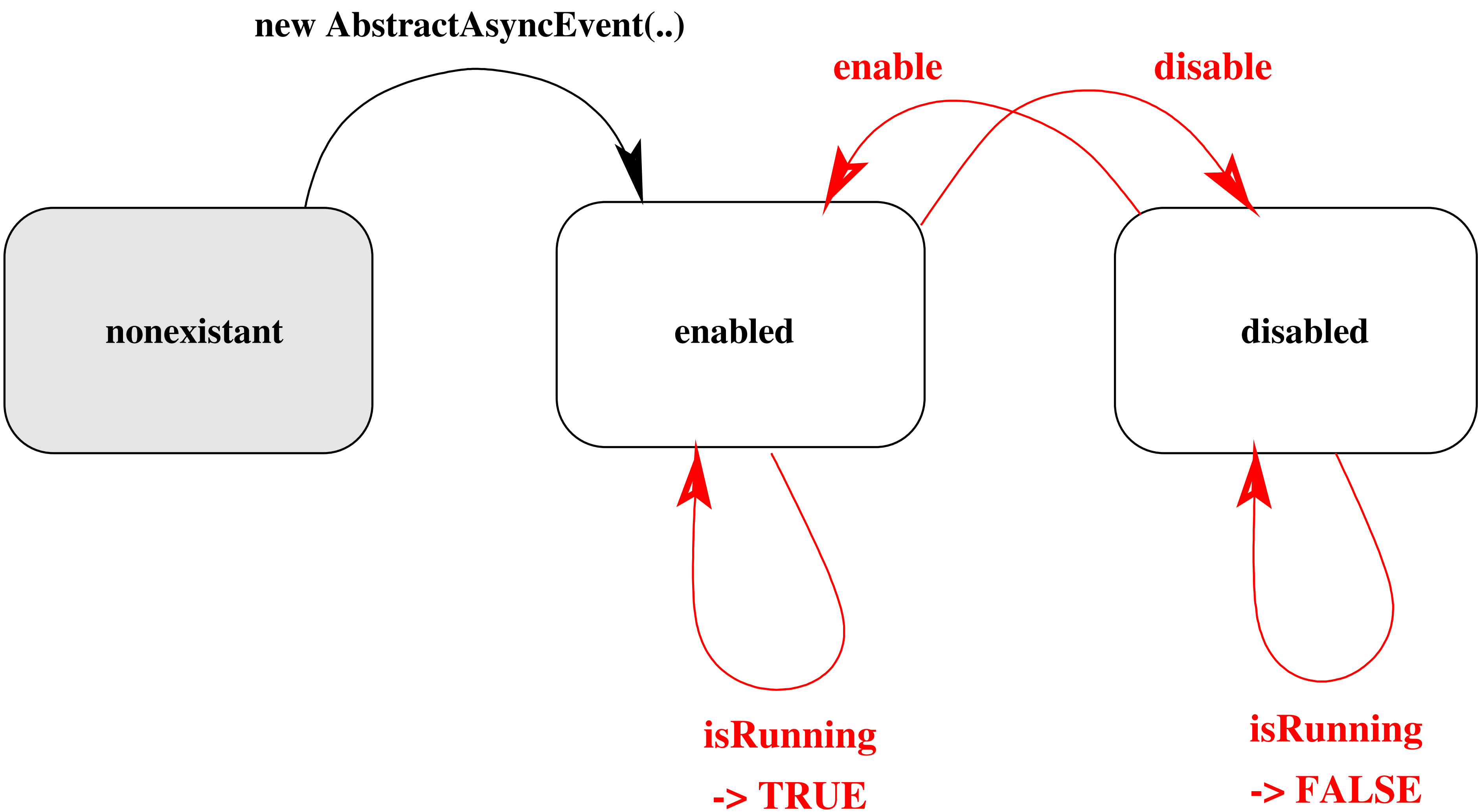
Event Handler Architecture



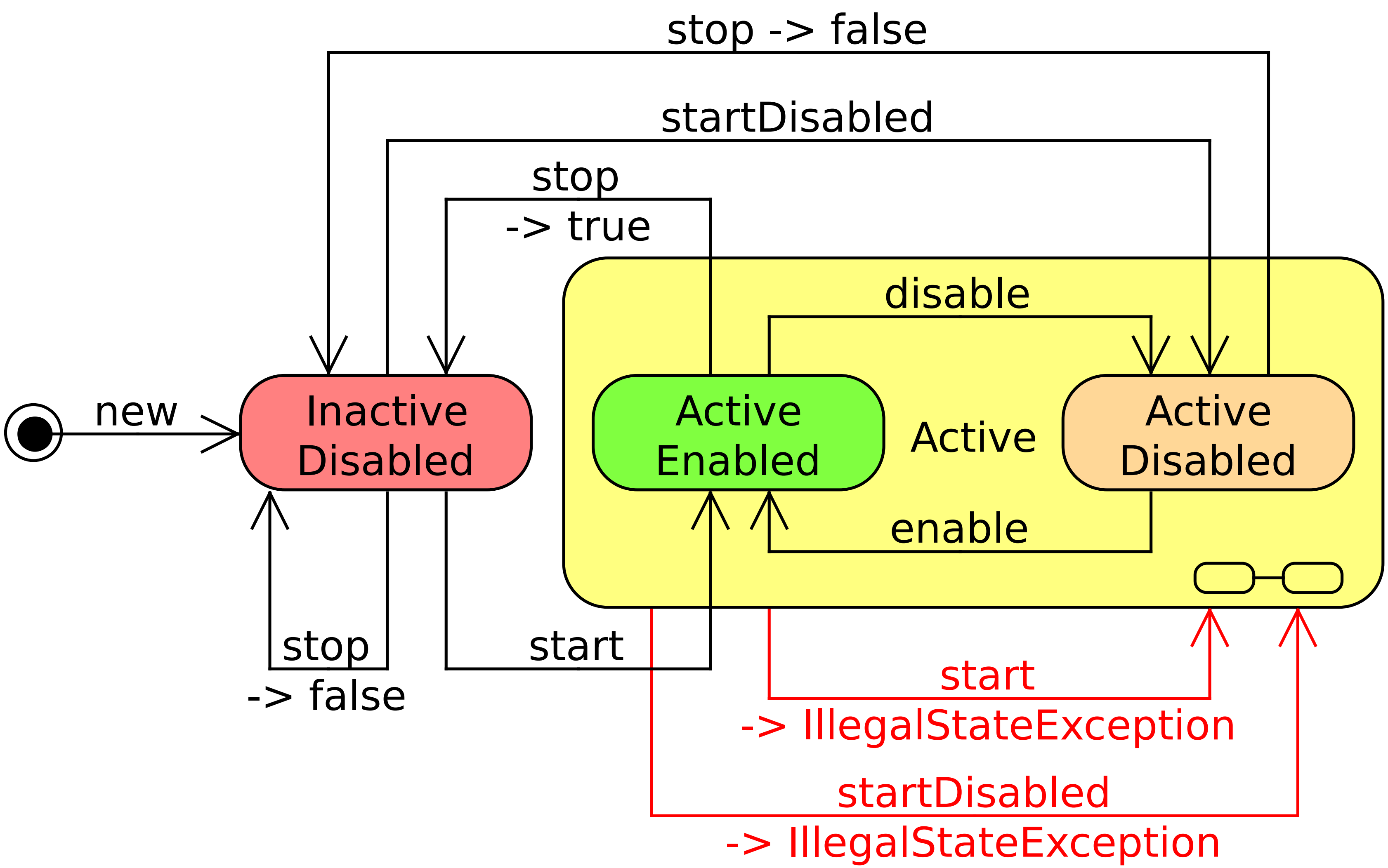
Mix and Match

Types	AsyncEvent	AsyncLongEvent	AsyncObjectEvent
AsyncEventHandler	Nothing	Nothing	Nothing
AsyncLongEventHandler	Event ID	Payload	Event ID
AsyncObjectEventHandler	Event Object	Event Object	Payload

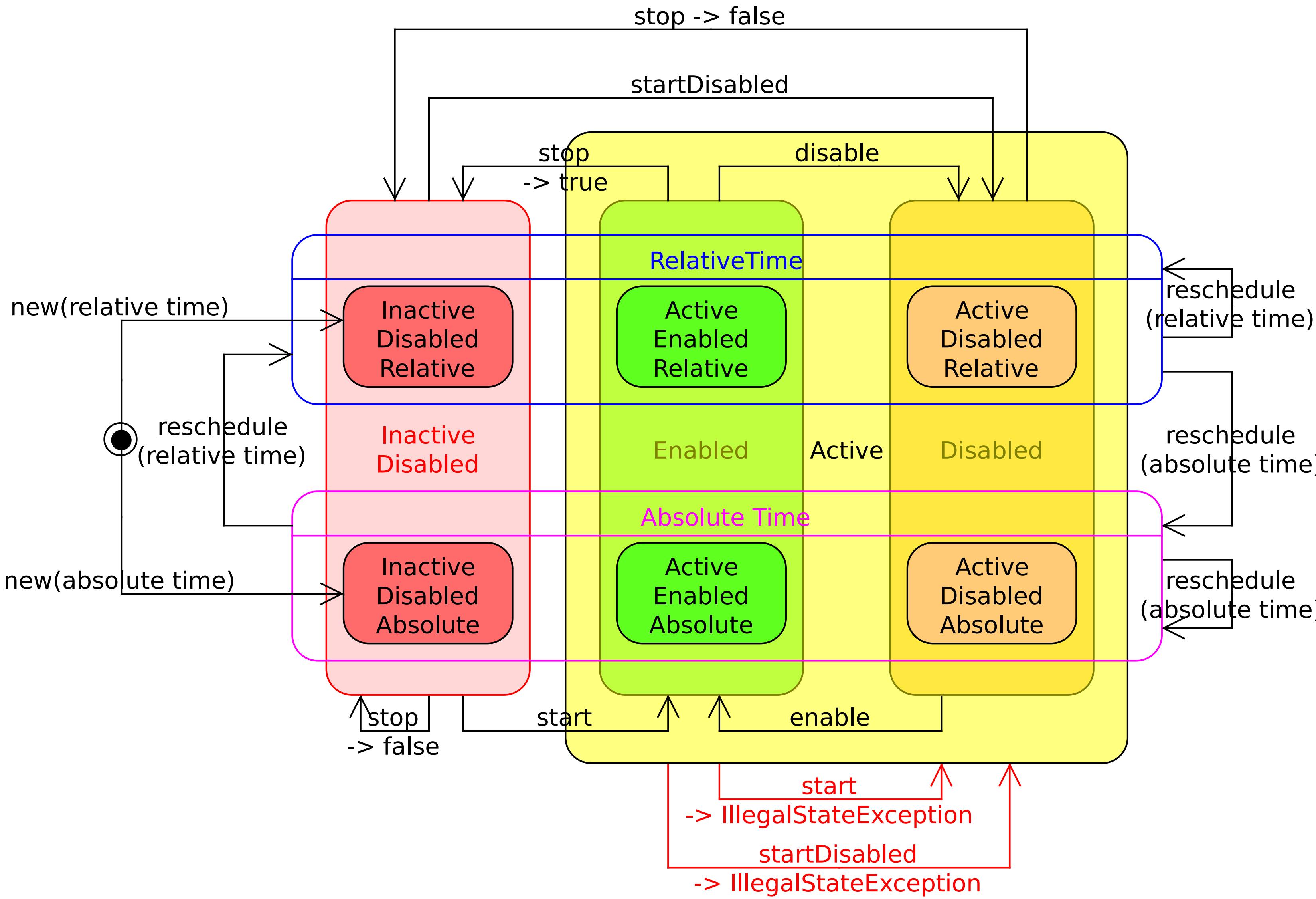
AsyncBaseEvent States



ActiveEvent State Machine



Timer UML State Machine



Happening: Kind of AsyncEvent

AsyncEvent

- Passive (fire mechanism)
- Runs all associated event handlers
- User definable

Happenings

- supports active behavior too (trigger mechanism)
- Can have (needs) dispatcher to manage activity
- Can be triggered from outside the VM

User Defined Clocks

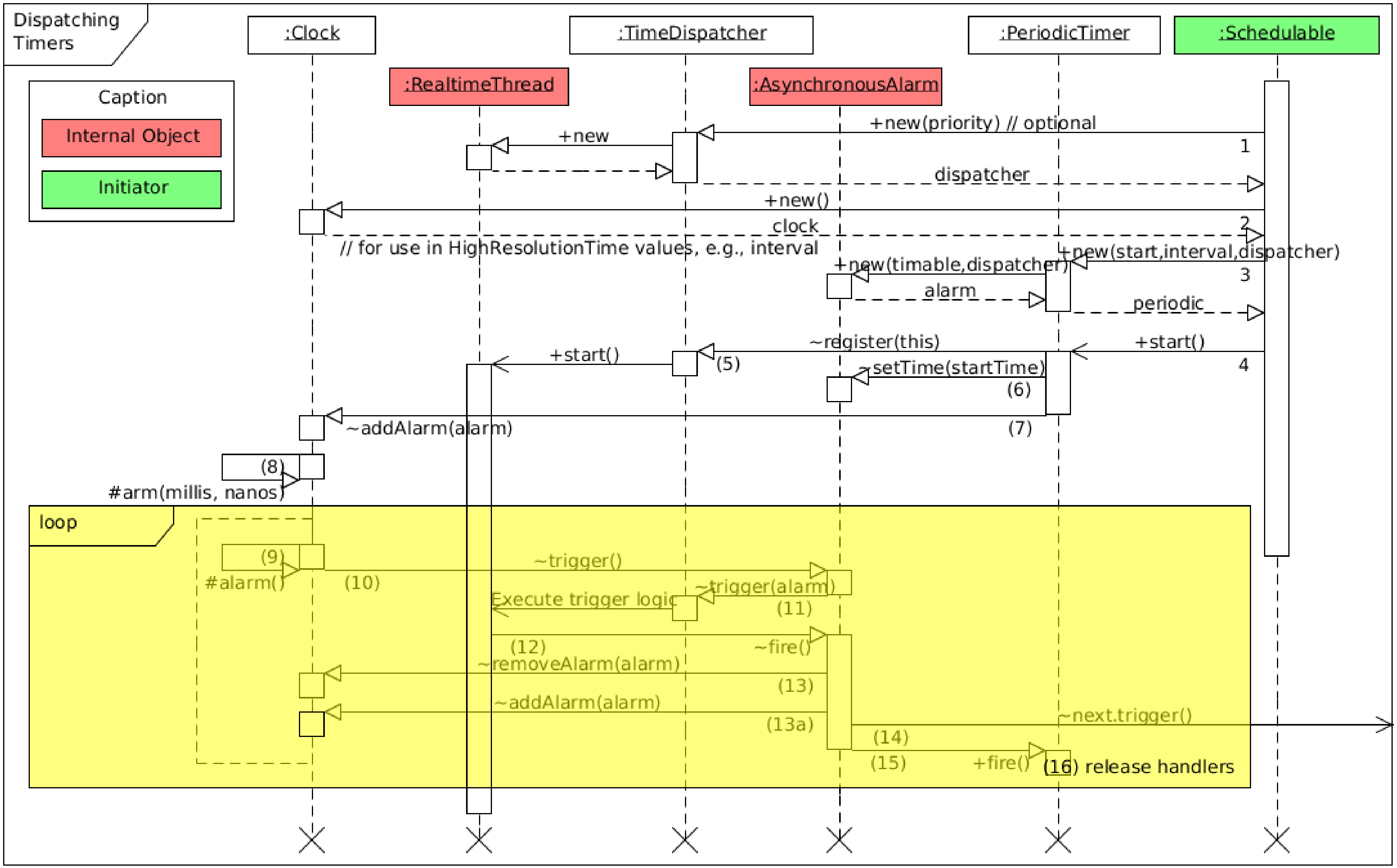
Similar to an ISR

- no active thread
- just triggers associated Timers

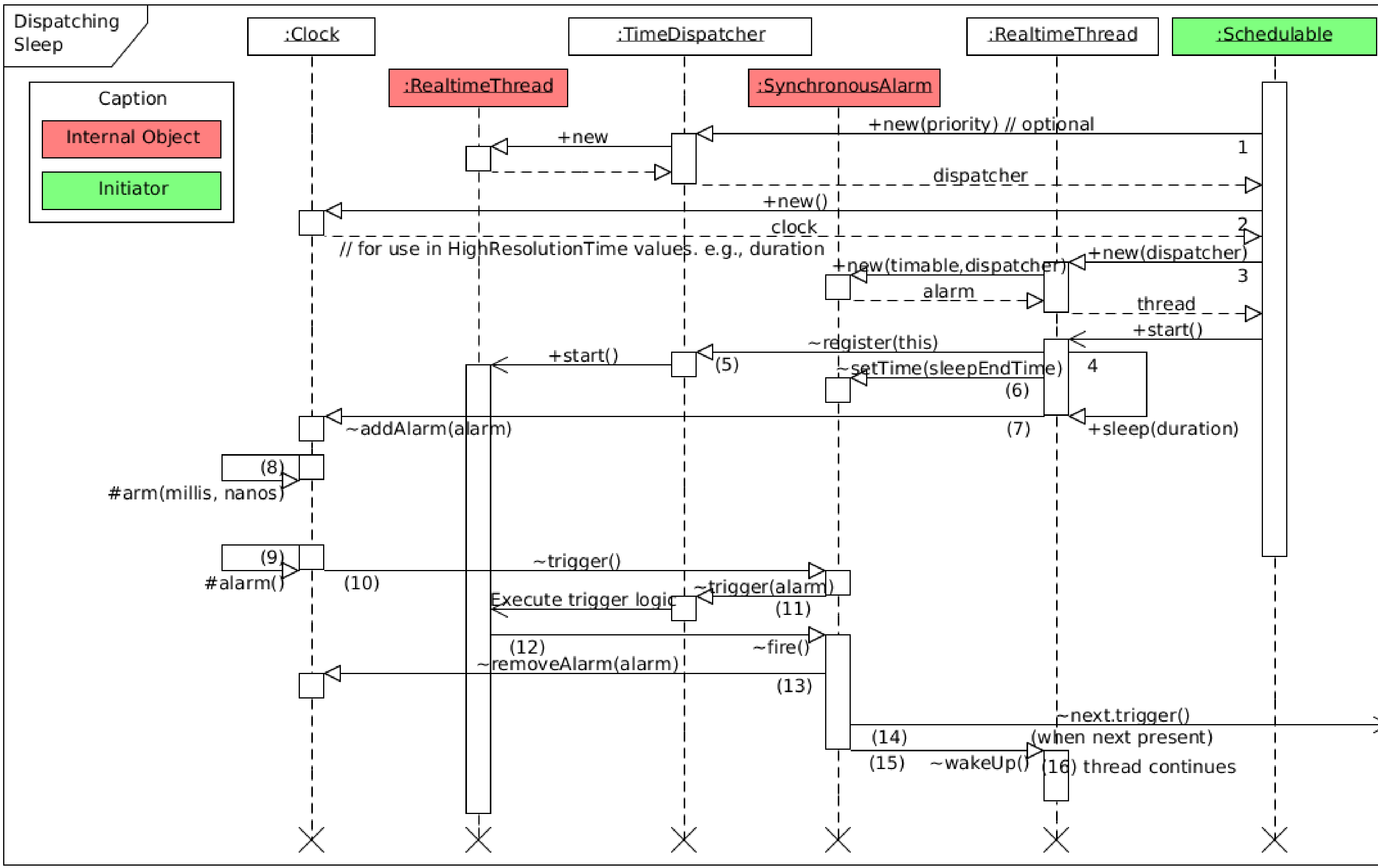
Manages Trigger Queue

- next set of Timers (in priority order) to trigger
- time ordered set of Timer sets
- constant trigger time for top next Timer
- bound adding and deleting

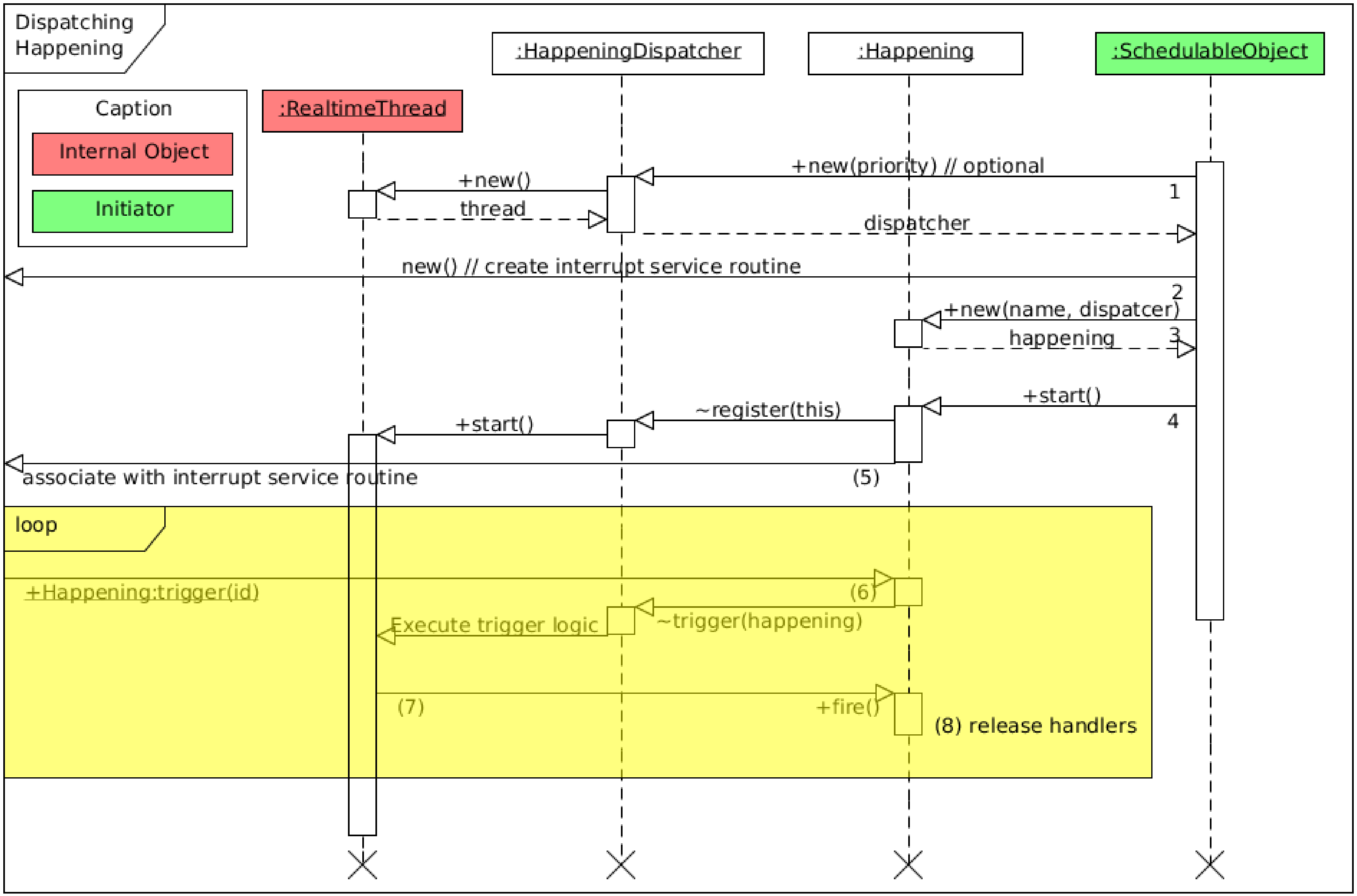
Clock Sequence Diagram



Sleep with Application Clock



Happening Sequence



New Raw Memory Architecture

FactoryBased

- RawMemory class for registration
- RawMemoryFactory for implementation

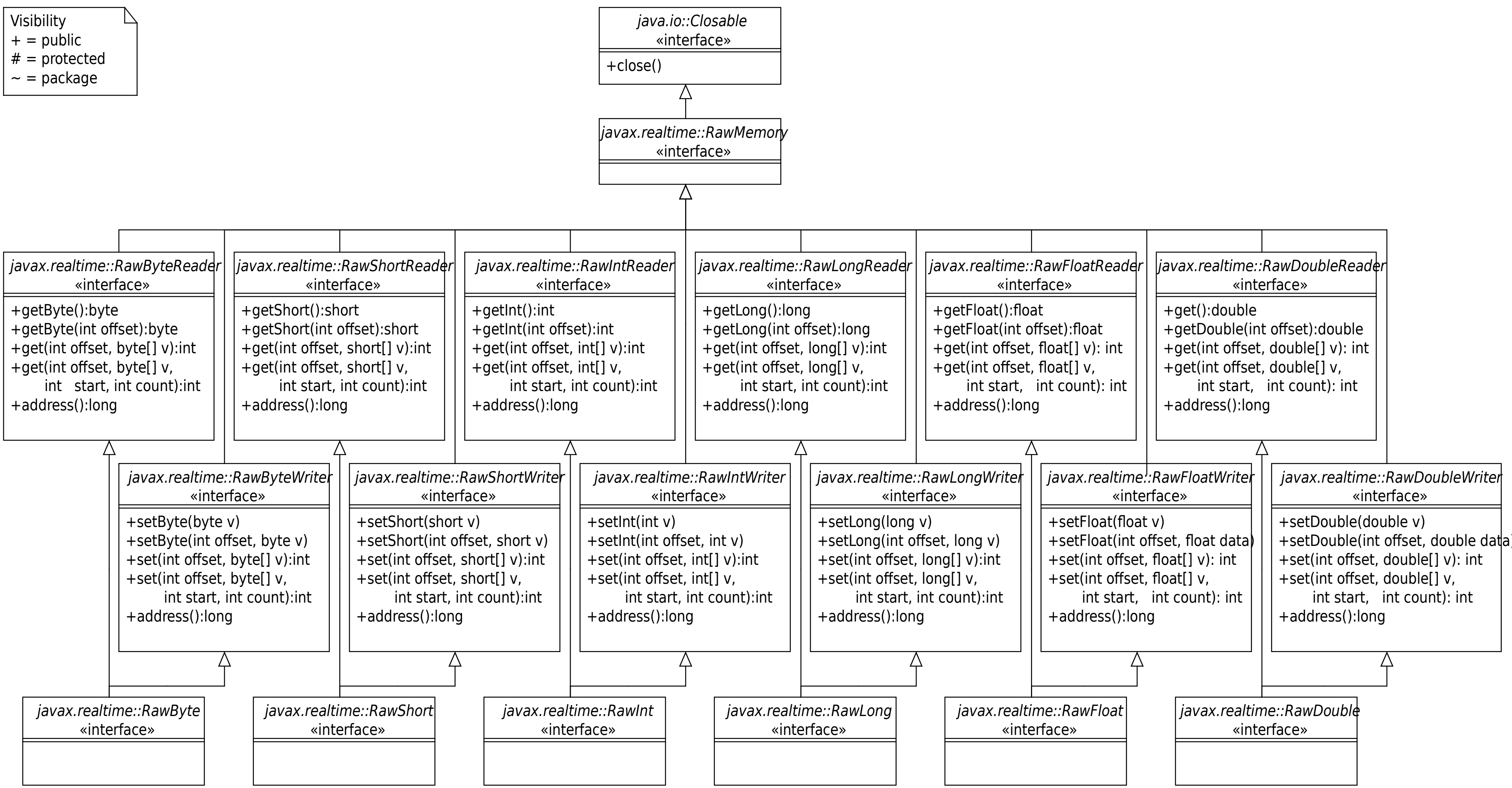
Interfaces for each access type:

RawInt, RawShort, RawByte, RawFloat, etc.

Concrete classes for

- Memory mapped devices,
- I/O mapped devices, and
- Generic mapped devices.

RawMemory Interfaces



Example

```
Public class IOBusController implements RawShort
{
  private MemoryRawByte command;
  private MemoryRawByte flag;
  private MemoryRawShort address;
  private MemoryRawInt data;

  public int get(short address)
  {
    address.put(address);
    command.put(READ);
    while (flag.get() != DONE);
    return data.get();
  }
...
```

DMA Support

Special factory for direct byte buffer

- Get byte buffer that is visible DMA controller
- Means to get address to pass to DMA controller
- Could be use to implement I/O Channels

Additional barrier types

- provide write visibility across JNI boundary
- for supporting DMA with direct byte buffers
- Coordinating with Doug Lea
  (JEP 188: Java Memory Model Update)

Entering MemoryAreas

Provide for passing arguments

- use lambda and closure
- can be optimized to prevent allocation

Provide a return value

- Use Supplier API
- adds many methods
  - five types: Object<T>, int, long, double, and boolean
  - enter, executeInArea, joinAndEnter, and joinAndEnter with timeout
  - 10–20 new methods!

New ScopedMemory Areas

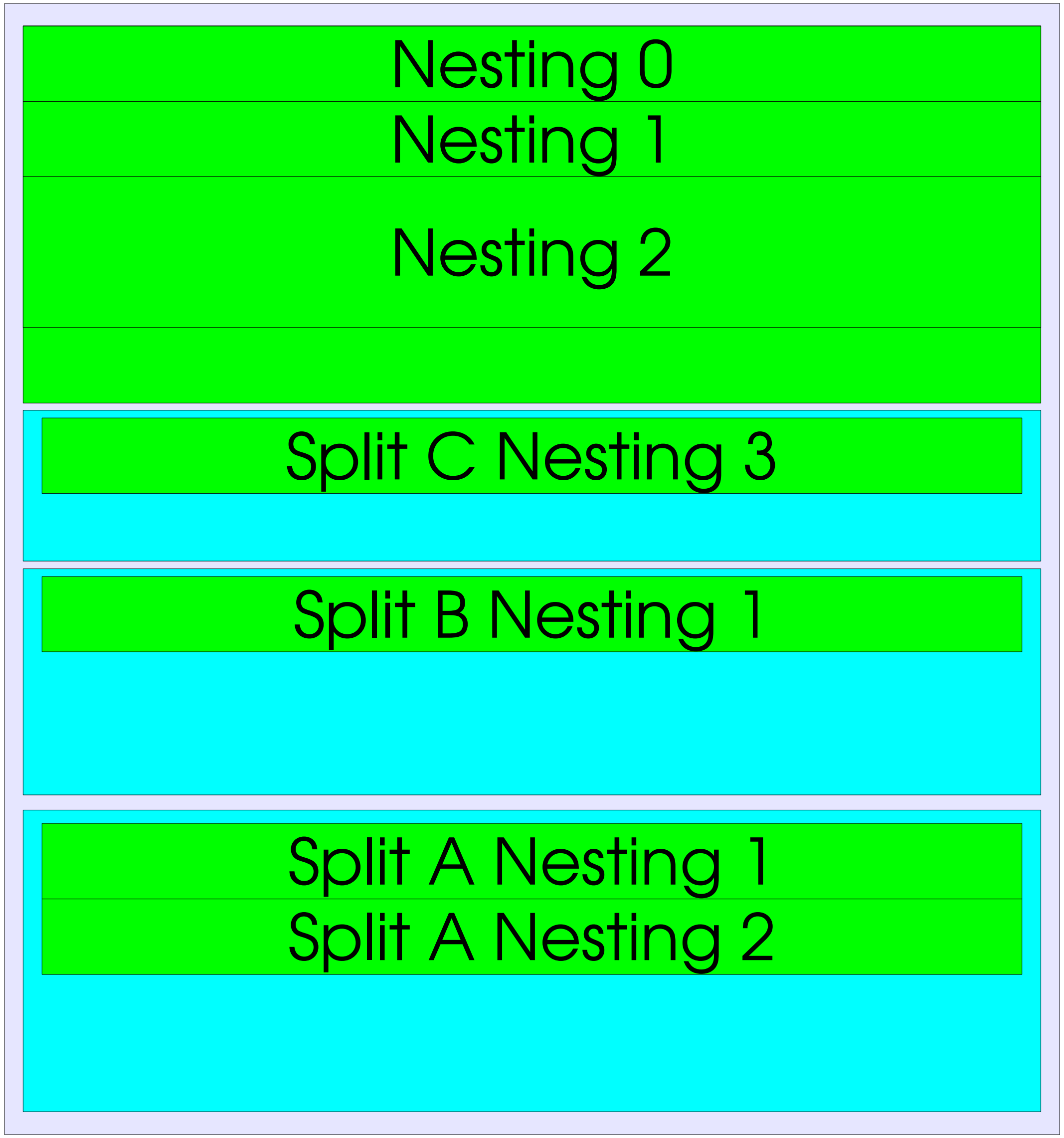
PinnedMemory

- subclass of ScopedMemory
- similar to LT Memory except supports pinning

StackedMemory

- subclass of ScopedMemory (supports SCJ Model)
- similar to LT Memory but reserves backing store
- backing store from parent when StackedMemory
- enterable only from MemoryArea where created
- resize to max reserved (when no sub area)

StackedMemory Example



New Physical Memory Model

Factory based

- completely separate from raw memory
- virtual memory address agnostic

Type compatible with other memory areas

- Immortal
- Pinned
- Stacked

Exception Handling

Preallocated Throwables

- Uses thread local storage for thowable data: StaticThrowableStorage
- Different throw pattern
  throw RegistrationException.get().initMessage(..);
- All RTSJ exceptions, e.g., ThrowBoundryException
- StaticOutOfMemoryException
- Base types for simple user extension: StaticError StaticCheckedException, StaticRuntimeException

Modularization Goals

Provide useful subsets of the RTSJ

- with and without a realtime GC
- with and without device support

Encourage more implementations

- Hard realtime, e.g., for control systems
- Soft realtime, e.g. for system monitoring
- No realtime, e.g., for development

Modules

Base Module

- Schedulables
- Events & Handlers
- Priority Inheritance
- Clock
- MemoryArea
  - HeapMemory
  - ImmortalMemory
- ...

Device

- Happenings
- RawMemory
- ISR (Option)

Alternate Memory

- physical
- scoped

POSIX

- POSIX signals

RTSJ Status

Specification mostly finished

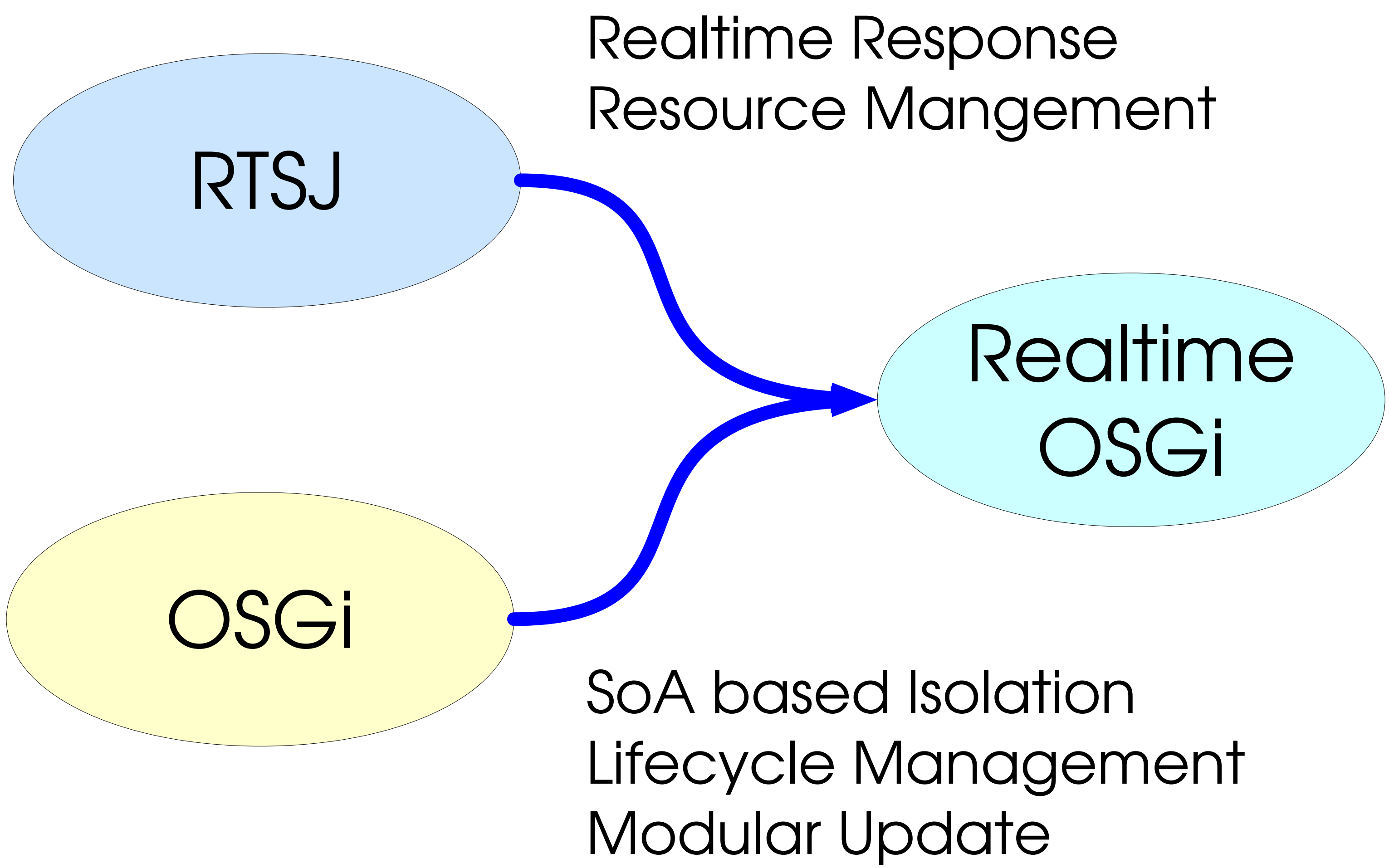
Last Open Issue: Security Management

- Methods needing security checks defined
- Some security classes defined
- How can allocation for security checks be minimized?

Future Work

- Finish reference implementation
- Update TCK

Supporting to IoT



Conclusion

RTSJ is a common API for realtime Java

- Provides for realtime sceduling and control
- Support interaction with environment
- Reduce need for external code

Strong Basis for Dynamic IoT Platforms

- Robust basis: work w/ OSGi
- Resource control
- Accommodates realtime requirement
- Provides direct device access (no JNI)

Contact Information

Dr. James J. Hunt

aicas GmbH
Haid-und-Neu-Str. 18
D-76139 Karlsruhe
Germany
jjh@aicas.com

JSR 282

Current specification:
https://www.aicas.com/cms/en/rtsj

Java Project Page:
http://java.net/projects/rtsj-2

Mailing List:
jsr282-feedback@aicas.com

Discussion:
http://www.linkedin.com/groups/RTSJ-8147216

Twitter:
@realtimejava #RTSJ

aicas GmbH is Hiring!

JamaicaVM

Jamaica JAR Accelerator

Jamaica-IoT

Jamaica3DFX

JamaicaTrace

JamaicaCAR

VeriFlux