



## JTRES 2015 Paris

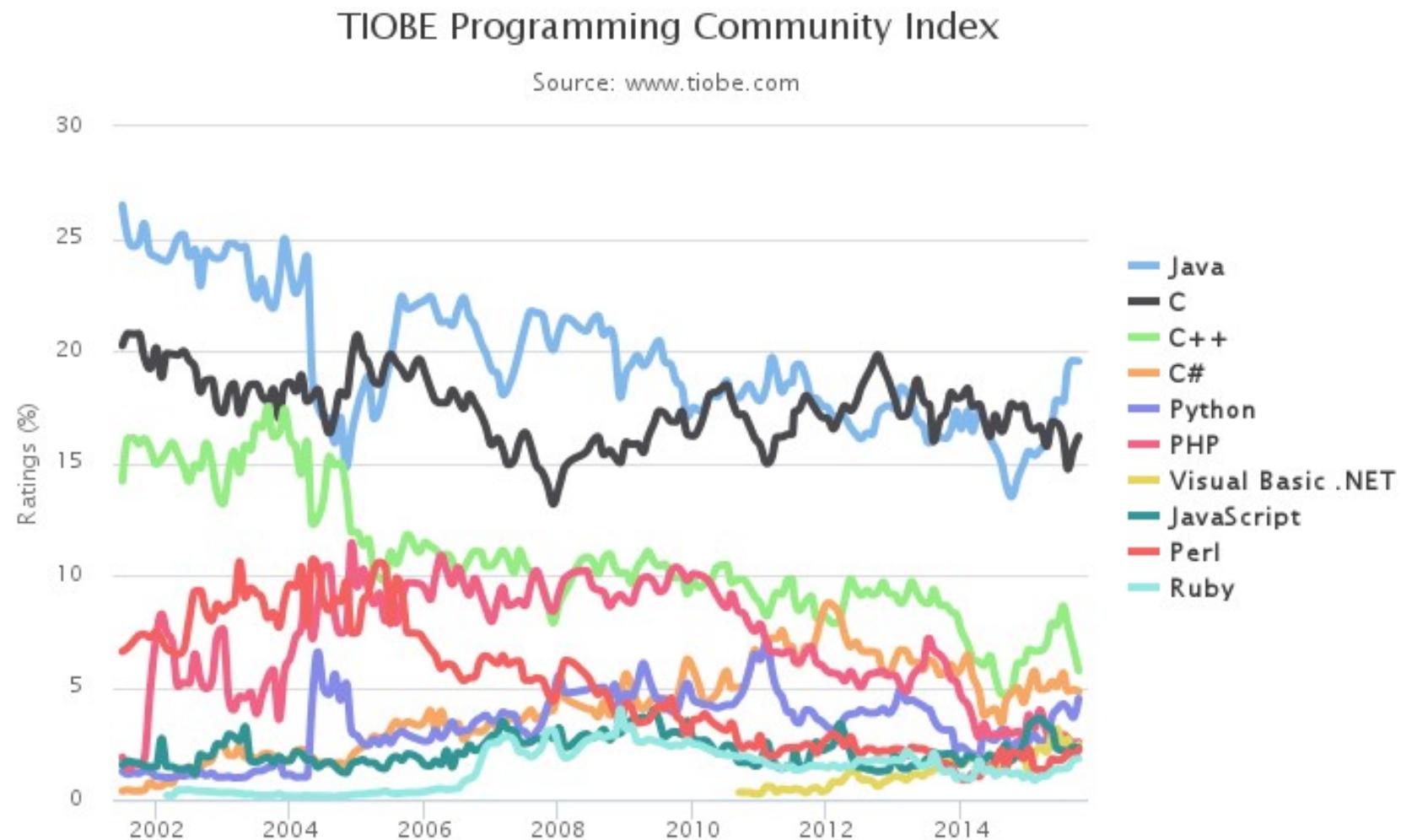
# The Next Generation of the Real-Time Specification for Java



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



# Java: Once and Future King





# Why Java for Embedded Systems?

## Pros

- Higher abstraction improve productivity
- Strong typing improve safety
- Exact garbage collection improves safety
- Most popular language for new development

## Cons

- Convention Garbage collection behavior
- No realtime support



# 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 features

Fully compatible with conventional Java implementations such as OpenJDK

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



# Why Update the RTSJ?

## Environment evolution

- better realtime garbage collection
- Java 1.4 → Java 1.8

## Marketing

- support different levels of realtime
- reduce need for JNI
- deemphasize 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

## Raw Memory

- Typed device access
- Factory based

## ActiveEvents

- Unify API for Timer, Happening, & Signal
- Happening object
- User defined Clocks

## ISR Support

## Stateful Events

- Object & long
- POSIX Realtime Signal

## Task Control

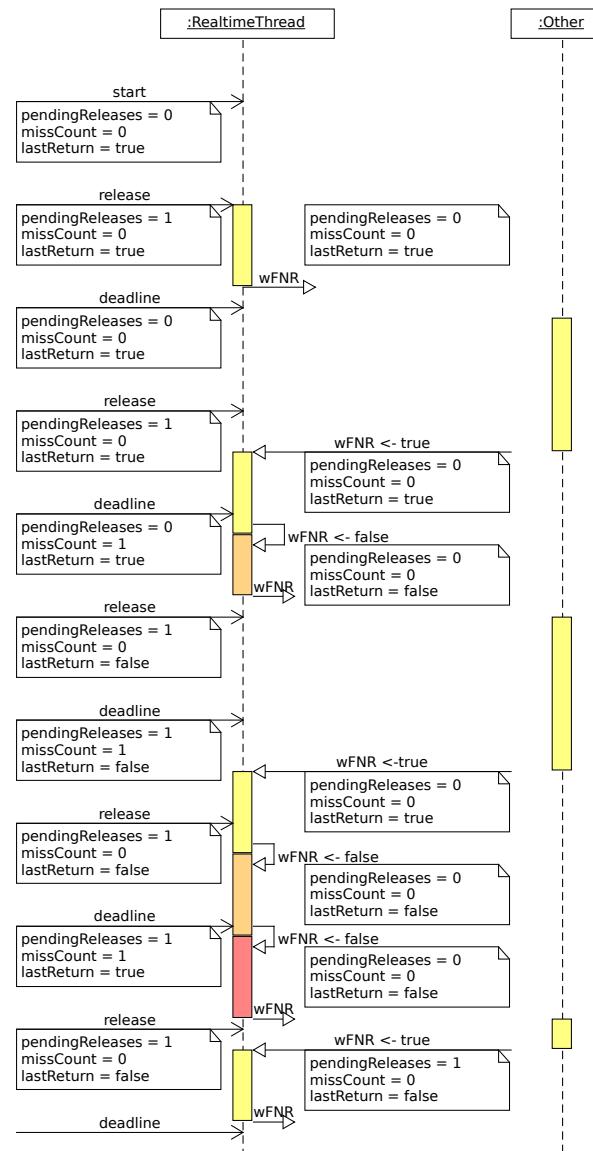
- CPU affinity
- Improved scheduling

## New Scope types

- PinnableMemory
- StackedMemory

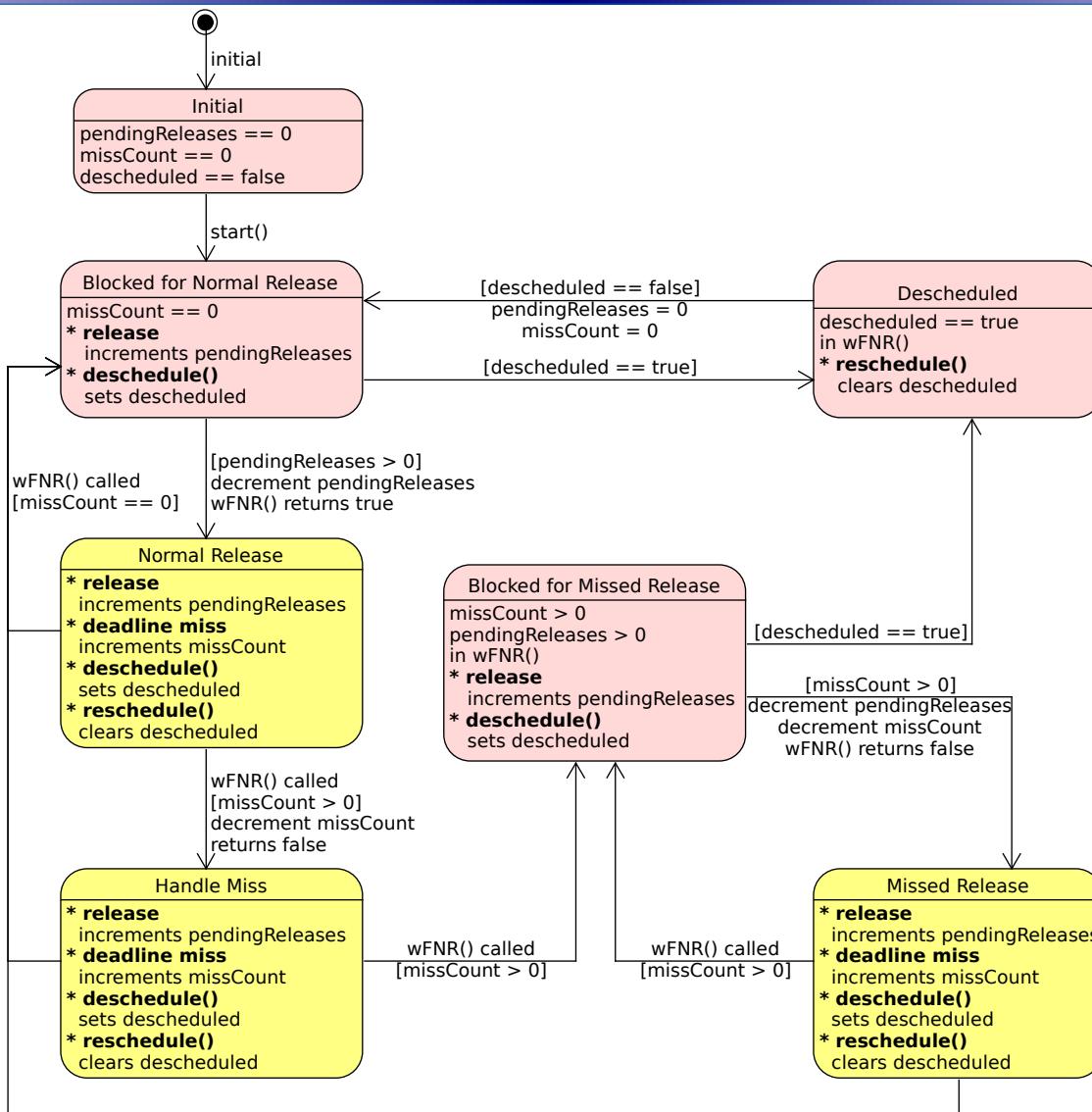


# Periodic RealtimeThread



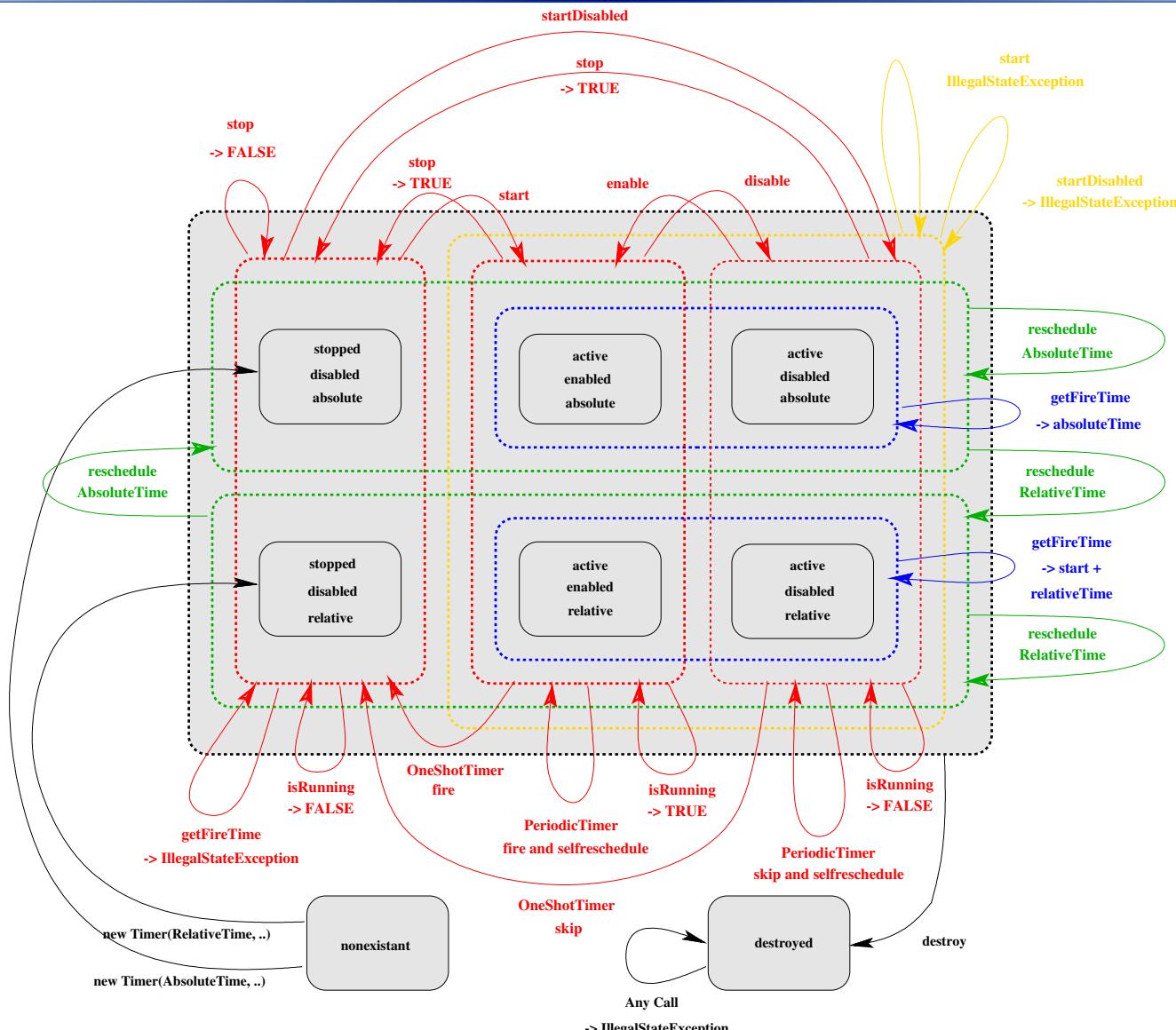


# Periodic RealtimeThread



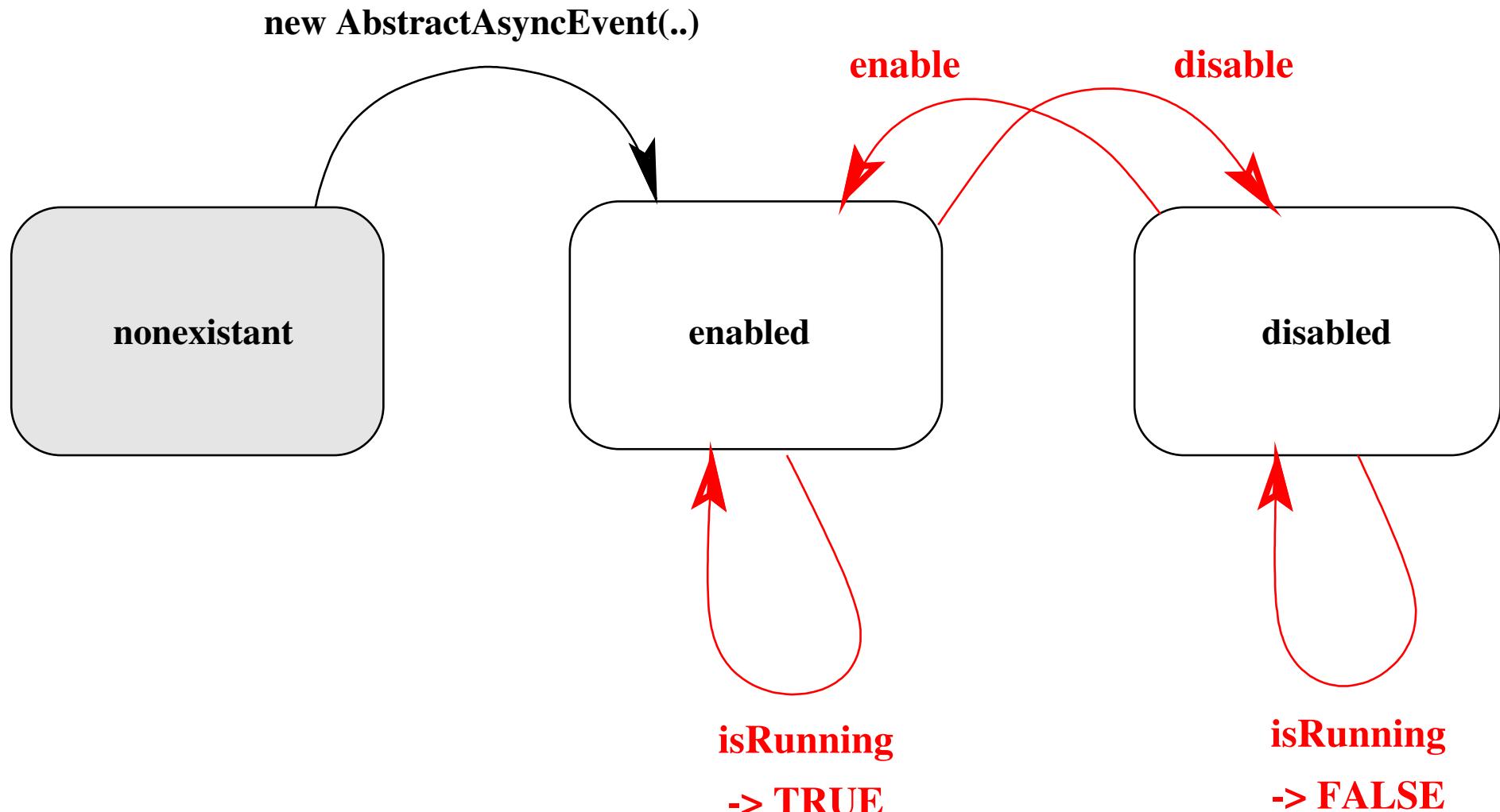


# Timer States



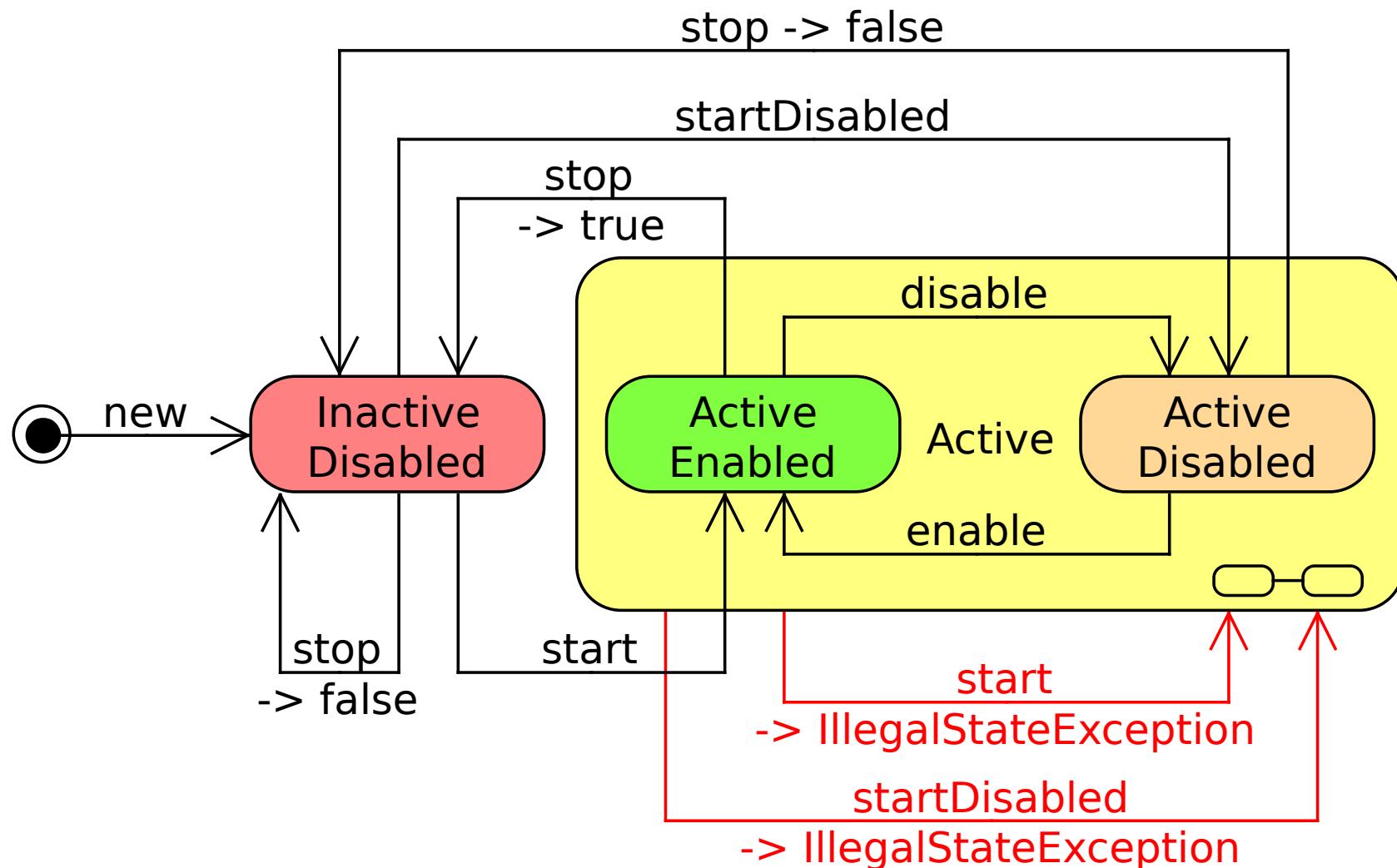


# AbstractAsyncEvent States



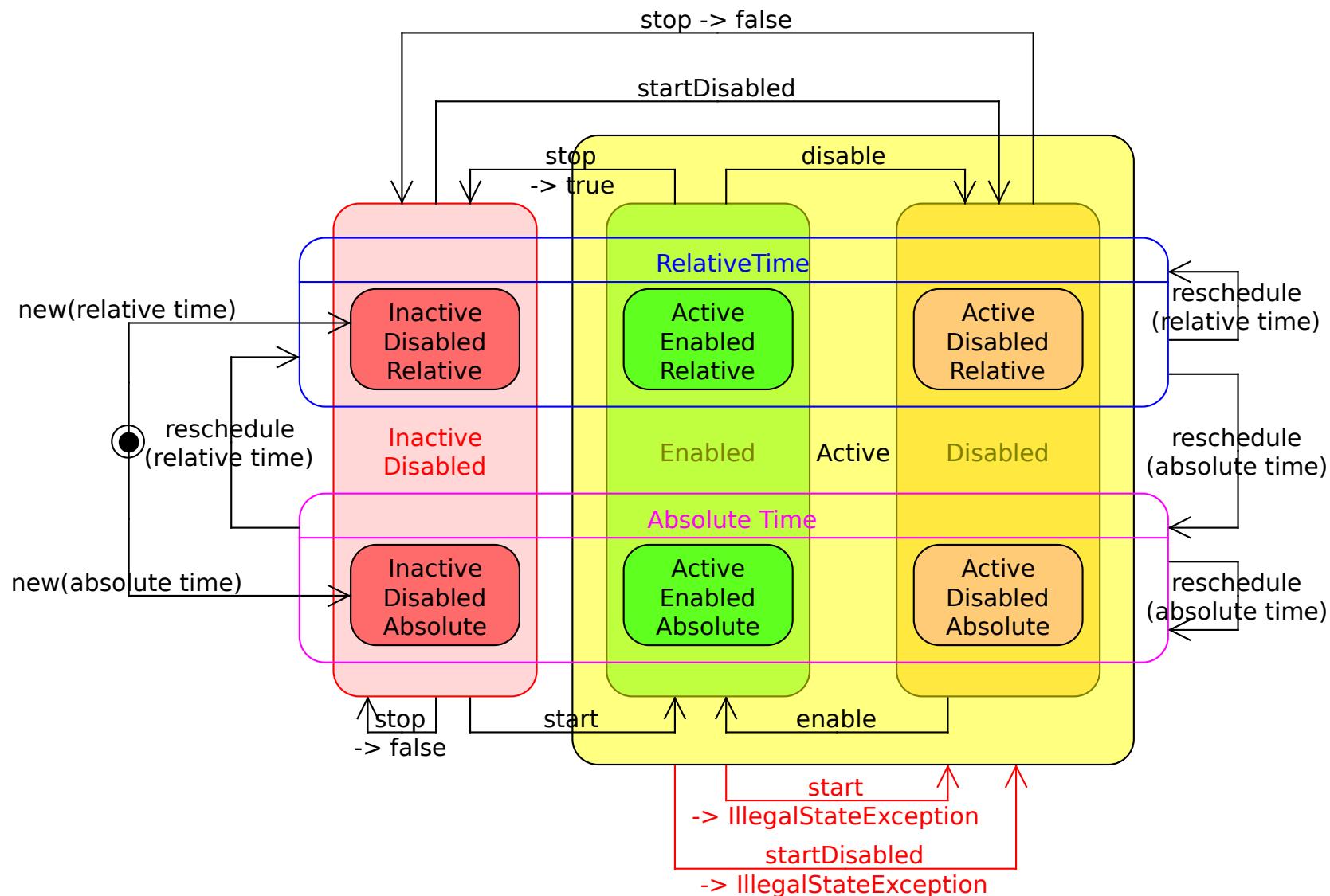


# ActiveEvent State Machine





# Time UML State Machine





# New Raw Memory Architecture

## FactoryBased

- RawMemory class for registration
- RawMemoryFactory for implementation

Interfaces for each access type:

RawInt, RawShort, RawByte, RawFloat, etc.

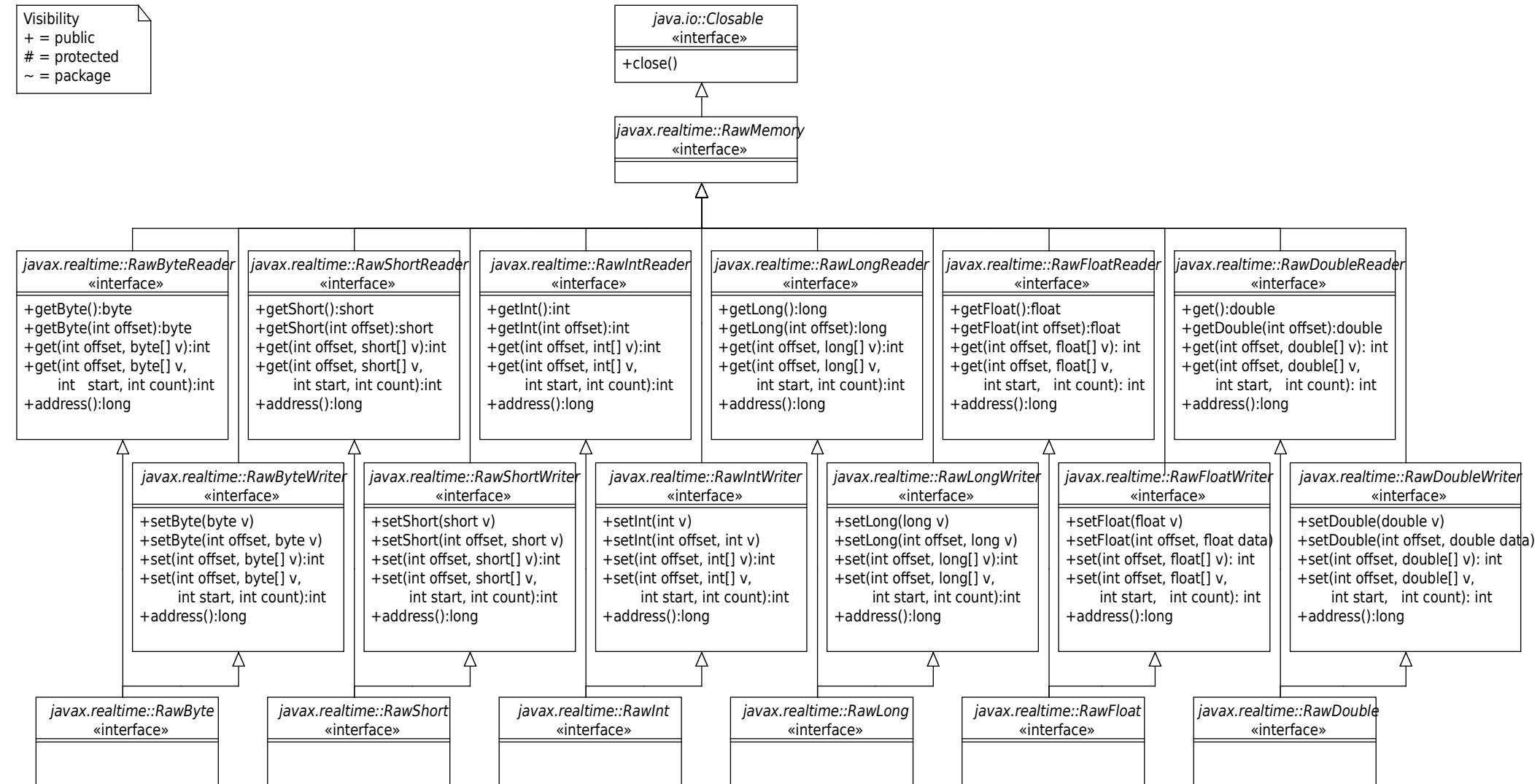
Concrete classes for

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



# RawMemory Interfaces

Visibility  
 + = public  
 # = protected  
 ~ = package





# 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

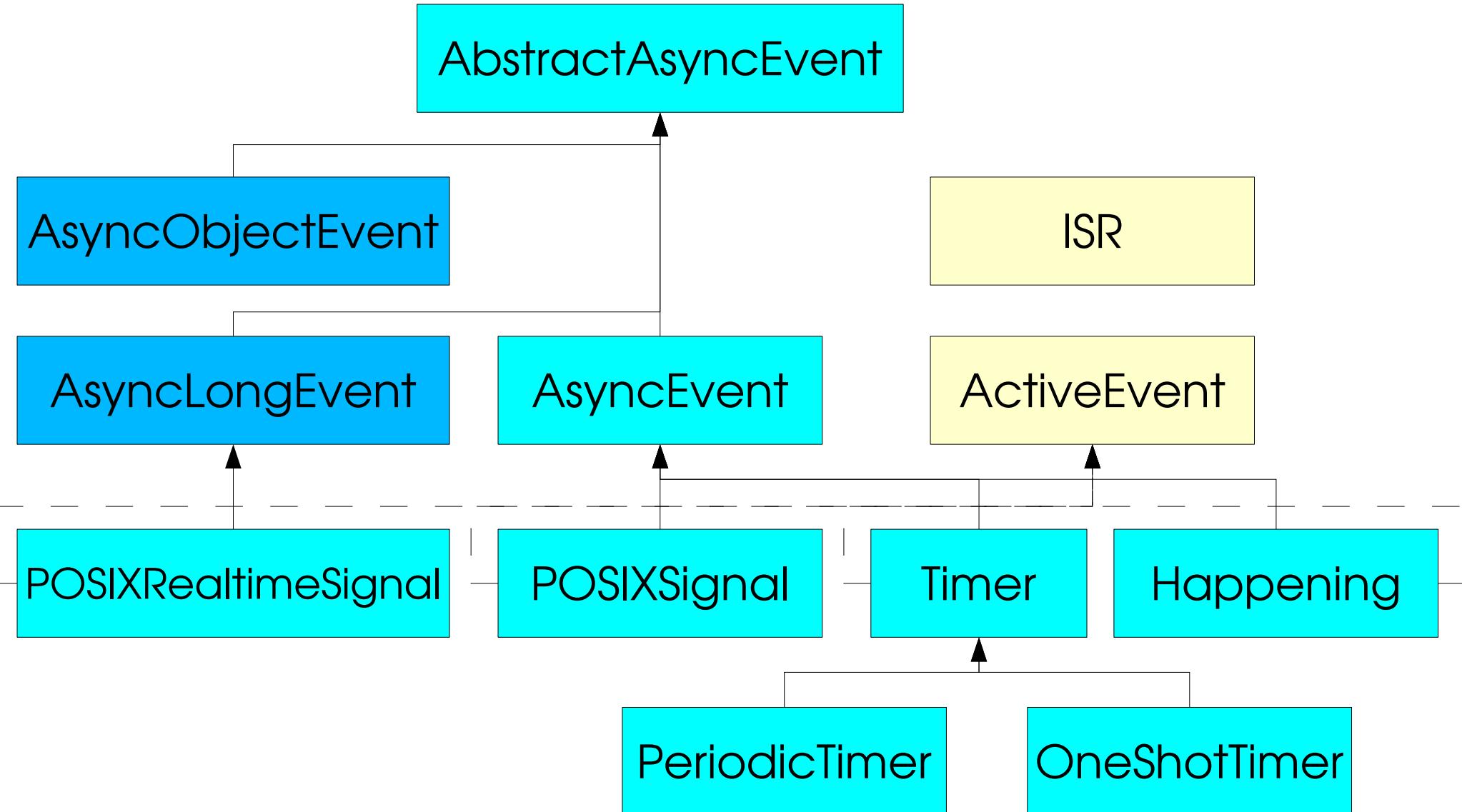
- Get byte buffer that is visible DMA controller
- Means to get address to pass to DMA controller
- Could be used to implement I/O Channels

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)

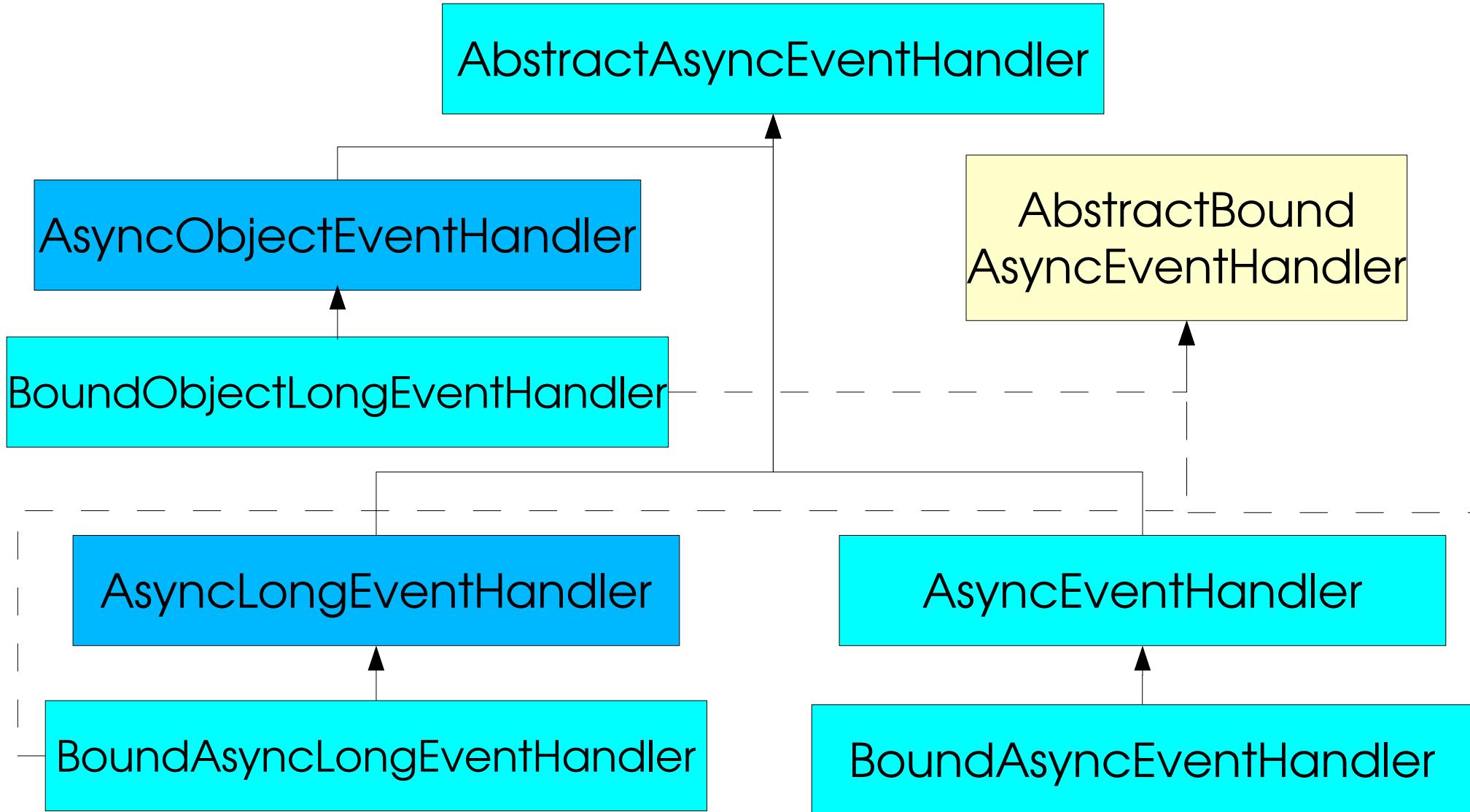


# Event Architecture





# 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



# Happening: Kind of AsyncEvent

## AsyncEvent

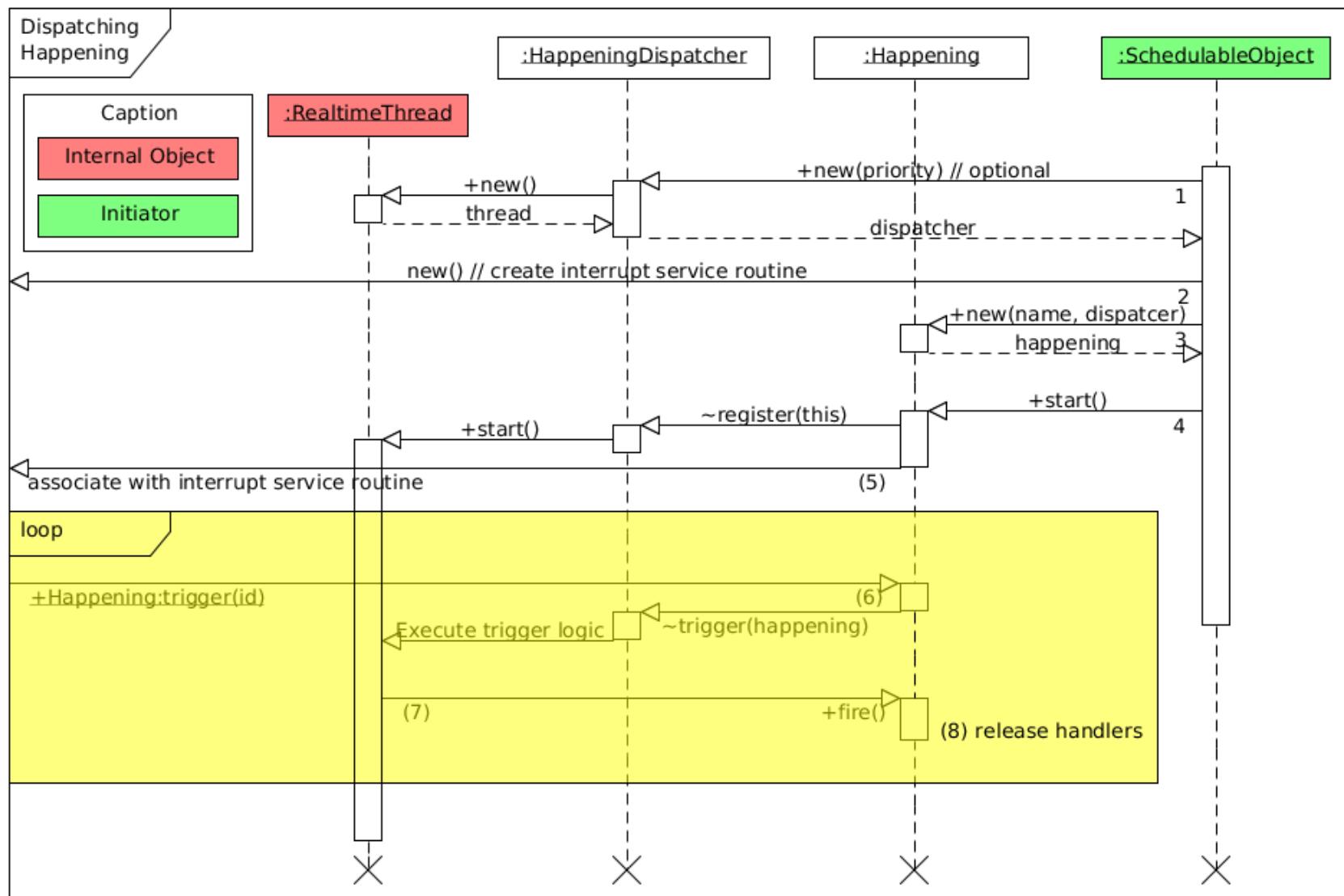
- Passive (fire mechanism)
- Runs all associated event handlers
- User definable

## Happenings

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



# Happening Sequence





# User Defined Clocks

Like an ISR

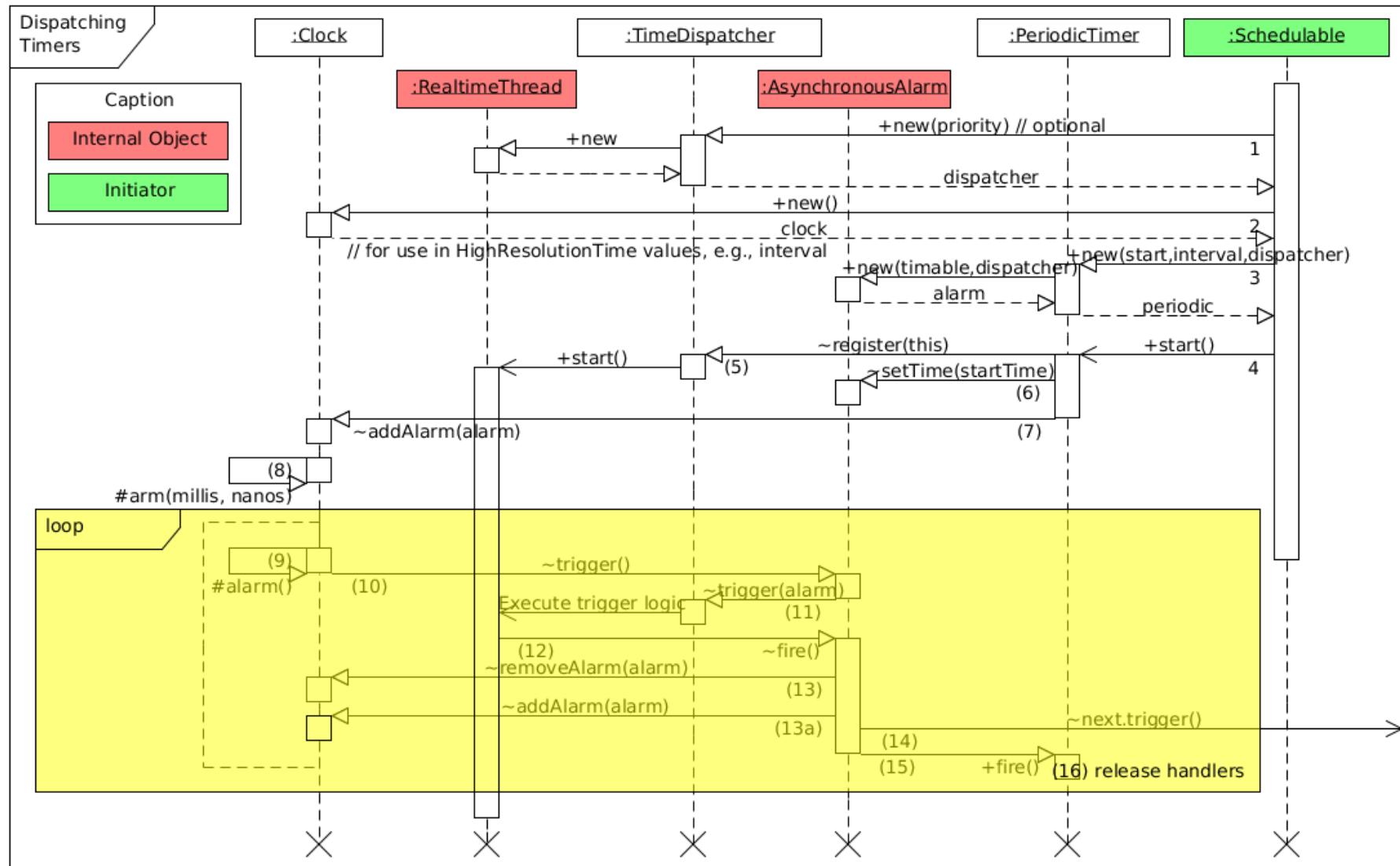
- no active thread
- just triggers associated Timers

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

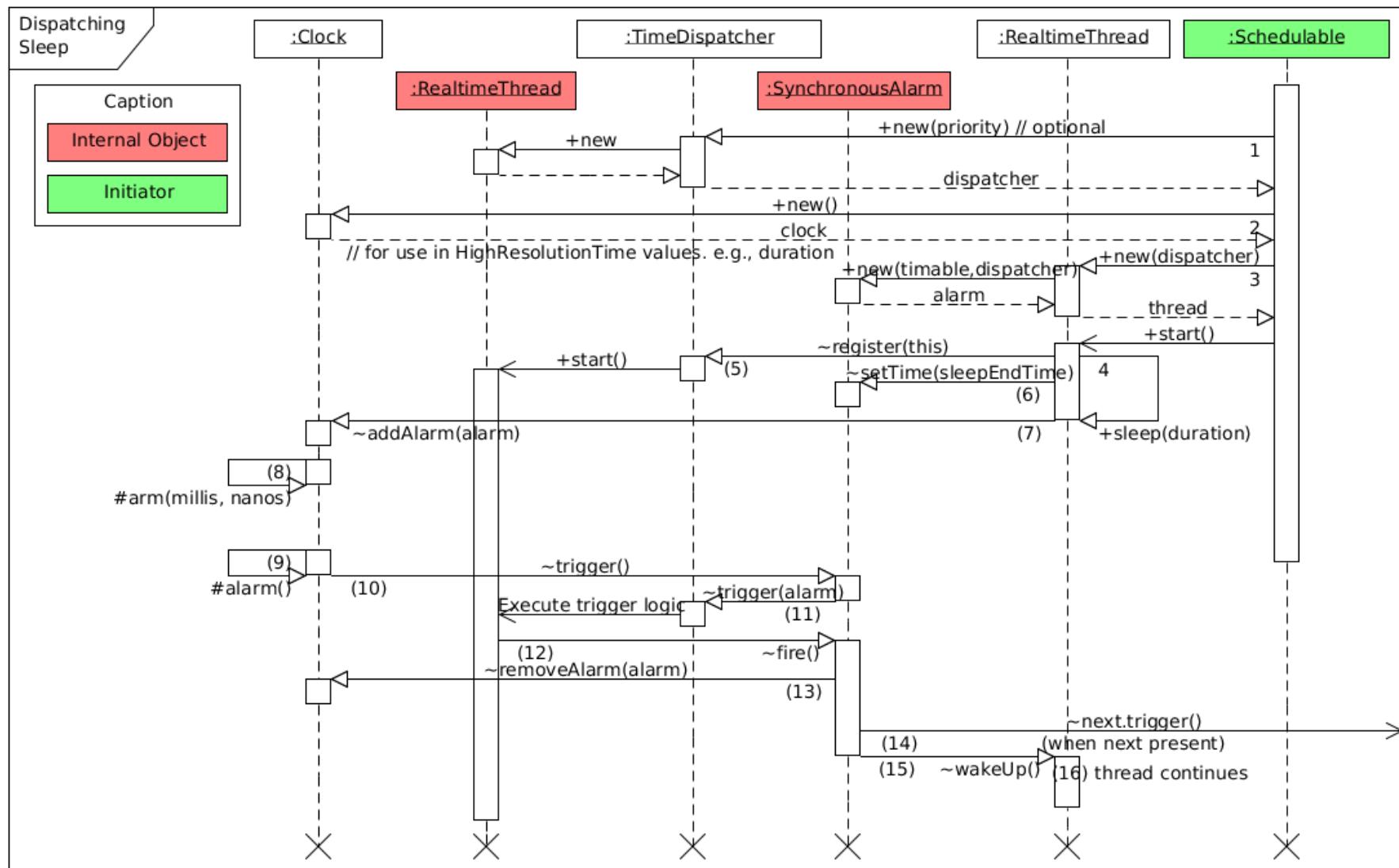


# Clock Sequence Diagram





# Sleep with Application Clock





# 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



# Scheduling

## New Scheduler

- FirstInFirstOutScheduler
- RoundRobinScheduler
- Subclasses of original PriorityScheduler

## Java Thread Control

- give java thread realtime priority
- give RealtimeThread java priority
- clarified what that means



# ThreadGroups

Extended through Subclasses

- SchedulingGroup — provides realtime group management
- ProcessingGroup — replacement for ProcessingGroupParameters

New Rules

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



# 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





# Small Change

## RealtimeThread

- add waitForNextRelease() and release()
- for aperiodic processing

## AsyncEventHandler

- memory area entered on each release
- old behavior obtained by using PinnableMemory

## ReleaseParameters

- add blocking factor to support feasibility analysis



# Small Change

## PreallocatedThrowable

- ThrowBoundaryException
- OutOfMemoryException

## Phasing control for periodic schedulables

- when start() called after absolute start time
- now, phased, all past, & exception

## PhysicalMemory

- Changed to factory w/o special types
- Separate from RawMemory



# Small Change

## RelativeTime

- add compareTo0()
- add scale(int factor)

## Timed

- add reset() method
- better support for watchdog timers

## PriorityCeilingEmulation

- now required in base module



# Small Change

## Consistency

- provide for firing an AsyncEvent (or release an AsyncEventHandler) on all resource limit errors.
- new methods as necessary to provide both a returned object and one that takes a destination, e.g., to reuse HighResolutionTime objects
- update semantics for cost enforcement to permit a schedulable's CPU use to be bound by its cost parameter in each period
- soft cost enforcement



# 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

- Schedulables
- Events
- Priority Inheritance
- Clock
- MemoryArea
  - ◆ HeapMemory
  - ◆ ImmortalMemory
- ...

## Device

- Happenings
- RawMemory
- ISR (Option)

## Memory Area

- physical
- scoped



# Conclusion

A common API for realtime Java

- provides for realtime scheduling and control
- support interaction with environment
- reduce need for external code

Base for a dynamic cyberphysical platform

- robust basis
- resource control
- recognizes realtime requirement



# aicas GmbH



Jamaica VM



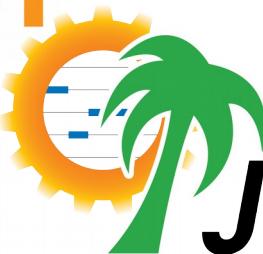
Jamaica-IoT



Jamaica 3DFX



Jamaica CAR



Jamaica Trace





# Contact Information

Dr. James J. Hunt

aicas GmbH  
Haid-und-Neu-Str. 18  
D-76139 Karlsruhe  
Germany  
[jjh@aicas.com](mailto: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](mailto:jsr282-feedback@aicas.com)

Discussion:

<http://www.linkedin.com/groups/RTSJ-8147216>

Twitter:

@realtimejava #RTSJ