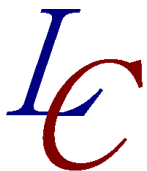


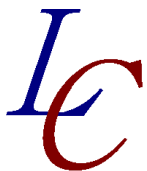
Application Design Using the Real-Time Specification for Java

Doug Locke, Ph.D.
Locke Consulting, LLC
www.douglocke.com

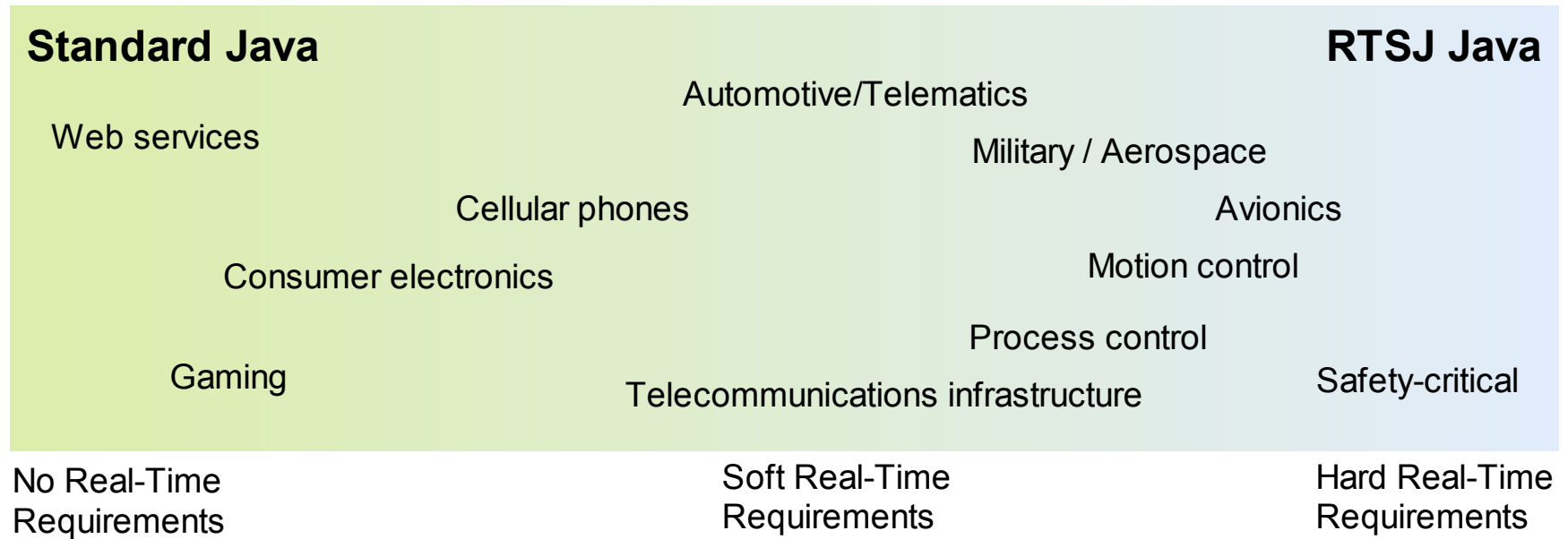


Outline

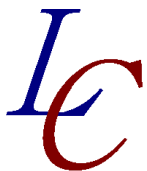
- **Why Use Java** in Real-Time and Embedded Systems?
- **Why Not Use Java** in Real-Time and Embedded Systems?
- **RTSJ** Highlights
- Conclusions



The RTSJ Broadens Java Technology's Reach

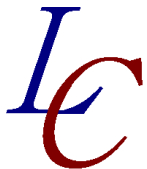


- Now many application domains can exploit the power of Java



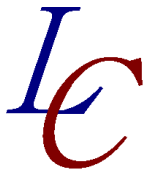
Why Java for Real-Time and Embedded Systems?

- Java has many advantages over C / C++:
 - Robust language – Designed to catch many errors at compile time
 - Object Oriented – Clean modularity, easy extensibility, design by perturbation, intrinsic information hiding
 - Safe object references – Can't generate General Protection Fault or Segmentation Fault
 - Concurrency - Threads as first-class constructs
 - Easy, safe synchronization – Uses “monitor” concept
- Result – High productivity, high degree of program correctness!



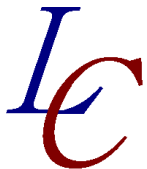
Why Not Java in Real-Time and Embedded Systems?

- Java Technology was not really designed for embedded or real-time systems:
 - Requires run-time garbage collection
 - Supports threads, but provides inadequate scheduling control
 - Synchronization delays unpredictable
 - No predictable event processing
 - No “safe” asynchronous transfer of control

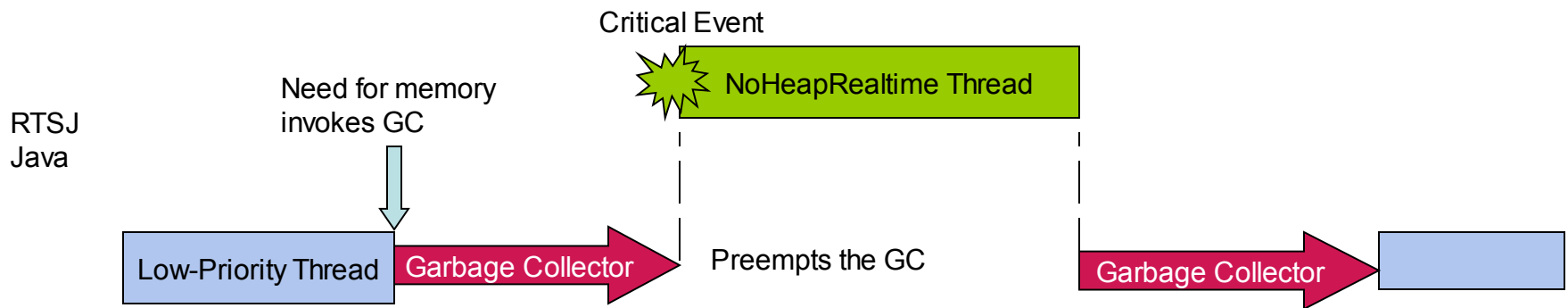
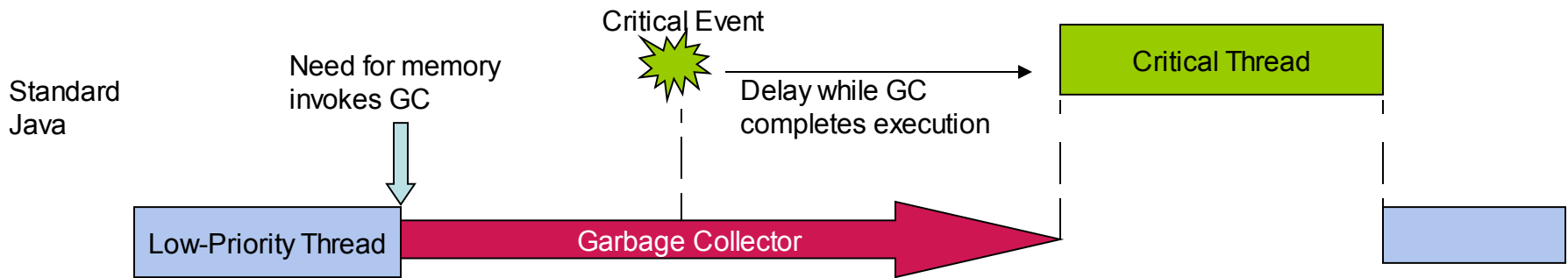


RTSJ Standardization

- The Real-Time Specification for Java (RTSJ) was specifically created to address these problems
- RTSJ was created under the Java Community Process (JCP)
 - Expert Group created by Dr. Greg Bollella (then with IBM) developed the RTSJ. A Technical Interpretations Committee (TIC) now maintains the RTSJ.
 - TimeSys Corp. (working with IBM and Sun) created its proof-of-concept Reference Implementation (RI) and its Technology Compatibility Kit (TCK).
 - The Specification, RI, and TCK, are available today at www.rtsj.org.
 - Version 1.0.2 is currently the official version.
 - JSR-282 (www.jcp.org) is currently enhancing the RTSJ

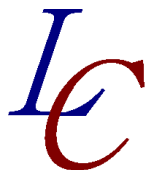


Meeting Timing Requirements Java



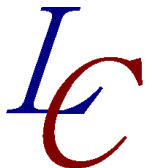
Real-Time Specification for Java - 1

- Memory allocation regions:
 - Heap memory – Objects remain until no references exist, then reclaimed by garbage collector
 - Can contain references to Immortal Memory
 - Garbage collector may cause delays, depending on garbage collector performance
 - Immortal memory – Objects remain until the application terminates
 - Never Garbage Collected
 - Can contain references to Heap Memory
 - Scoped memory – Objects remain until last thread leaves it, then it is totally reclaimed
 - Never Garbage Collected
 - Can contain references to Immortal, Heap Memory, and other Scoped Memory if higher in scope stack



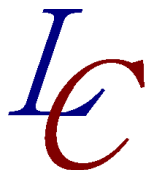
Real-Time Specification for Java - 2

- Threads
 - Normal Java Threads (*java.lang.Thread*)
 - No restrictions on its access, but cannot reach scoped memory
 - Cannot get predictable performance.
 - *RealtimeThread*
 - Can access all memory regions
 - May be delayed by Garbage Collector
 - Depends on the Garbage Collector performance
 - *NoHeapRealtimeThread*
 - Can access only scoped and immortal memory
 - Can always preempt Garbage Collector
 - Never delayed by Garbage Collector
 - Flexible scheduler interface for custom scheduling policies
 - Priority scheduler is minimally required
 - Dramatic effects on RealtimeThreads, NHRT's, AsyncEventHandlers
 - More advanced schedulers (e.g., EDF, utility functions) can be implementation-defined
 - Priorities: at least 28 real-time priority levels in addition to the 10 standard priority-like levels, with real-time priorities taking precedence over standard priorities
 - Most implementations offer more than 28 priorities (e.g., Sun's Java RTS provides at least 60 priorities)



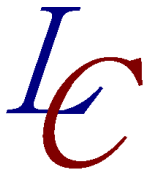
Real-Time Specification for Java - 3

- Synchronization
 - Priority inheritance to control priority inversion
 - Priority ceiling emulation to control priority inversion (optional)
- Asynchronous Events
 - Asynchronous Events connect to Asynchronous Event Handlers
 - Events can be triggered arbitrarily by *fire()*, by timers, by signals, or by happenings
- Asynchronous Transfer of Control
 - Asynchronously Interrupt Exceptions raised by any thread in a target thread
 - Safe - may be disabled (default) or enabled in the target thread



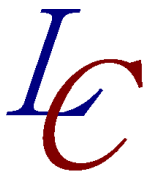
Real-Time Specification for Java - 4

- High Resolution Timers
 - Specified in nanoseconds
 - Actual resolution is implementation-defined
- Raw Memory Access
 - Provides direct access to memory areas accessible through the Operating System. The details are implementation-defined, but might include:
 - DMA areas
 - Memory mapped areas
 - Flash memory areas



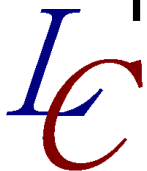
Is Real-Time Java Application Design Harder?

- Yes
 - Real-time application design harder than non-real-time application design.
 - Regardless of language.
 - Resource management is critical to performance management and prediction – it can no longer be ignored until integration.
- Real-time Java transition will require careful planning.
 - Java developers starting real-time design.
 - Must now take thread, memory, and synchronization issues seriously.
 - Real-time developers starting to use Java.
 - Must understand Java resource management principles (e.g., inheritance, polymorphism, binding time, garbage collection).



Conclusions

- Java Technology is a powerful enabler for robust embedded and real-time applications.
- The RTSJ fills the principal gaps in Java Technology for embedded and real-time applications.
- Real-Time application design is harder than non-real-time application design.
 - Regardless of whether Java is used.
 - Real-time application designers must reason about resource usage to meet time constraints
- Real-Time and Embedded system transition to RTSJ Technology should be carefully planned.



Dr. Doug Locke
Locke Consulting, LLC
www.douglocke.com
doug@douglocke.com

