Java in Safety Critical Systems

Towards Java Certification

aicas GmbH

Andy Walter, COO

2nd July 2009

Critical Software Challenges

- Complexity of applications increasing
- Ada developers diminishing
- C/C++ error prone and dangerous
- Java:
  - modern development tools
  - safe programming language
  - comprehensive set of standard libraries
- Is Java certifiable for safety critical applications?

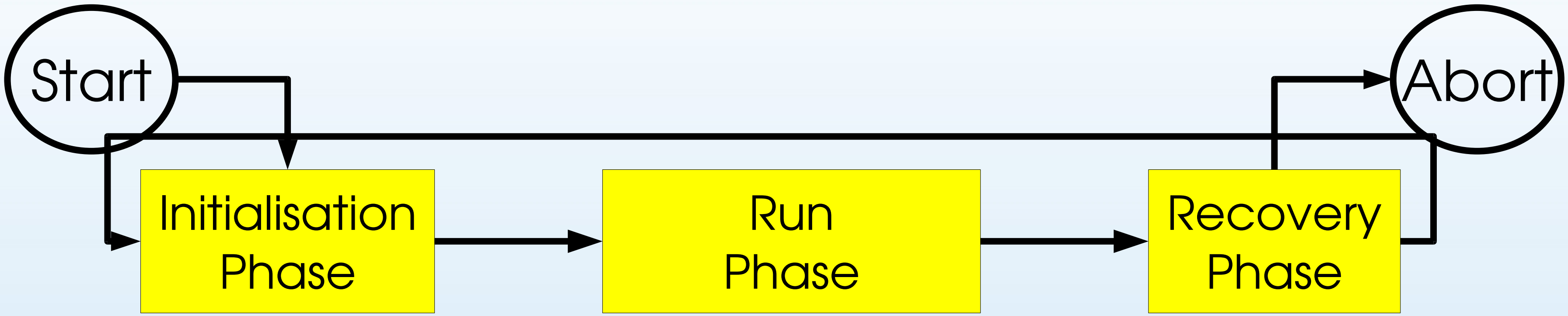
Realtime Specification for Java

- Java Community Standard (JSR 1, JSR 282)
- Most common for realtime Java applications
- New Thread model: NoHeapRealtimeThread
  - Never interrupted by Garbage Collector
  - Threads may not access Heap Objects
- Does not address certification of Safety Critical applications

Safety Critical Java Proposal

- Upcoming Java Community Standard (JCP 302)
- Aims for DO-178B, Level A
- Based on very limited RTSJ subset
- ScopedMemory instead of Garbage Collection
- Extended typing to support static analysis

Safety Critical Java Proposal

- Only RealtimeThreads are allowed
- No heap objects/ no GC
- Object allocation in initialisation phase only
- 

- Thread priorities may not be changed
- Always priority ceiling emulation
- OutOfMemoryError may not occur

Class Libraries for SCJava

- Base: CLDC (IMP - Information Module Profile) with
  - Floating point support
  - Error Handling
  - JNI
  - RTSJ-Subset (javax.realtime)
- ==> circa: lang, reflect, io, net, util, and realtime
- Exact library set has not been fixed
- Super sets for DO-178B, Level B and C will be specified.

New Standard DO-178C

- No OOT in DO-178B
- OOT provides
  - Reusability
  - Extendibility
  - Code efficiency
  - Modern development paradigm
  - New vulnerabilities
- DO-178C introduces guidance and guidelines on certification of OOT applications.

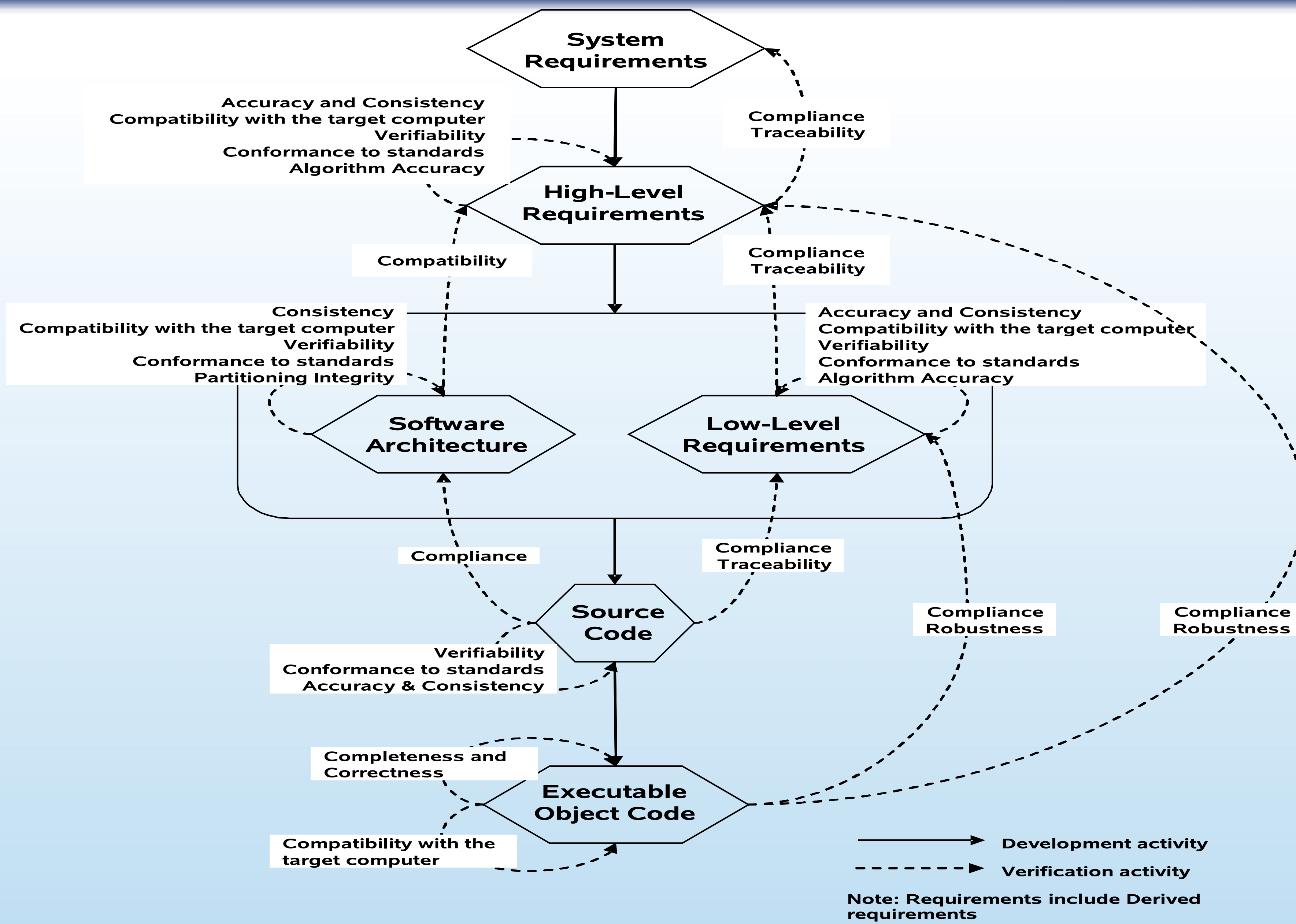
SC-205 / WG-71 Plenary

- Lead by RTCA and EUROCAE
- Update software standards for aviation
  - DO-178B/ED-12B: flight software regulations
  - DO-248B/ED-94B: flight software addendum
  - DO-278/ED-109: ground support software
- Open to all interested parties
- Organized in seven subgroups

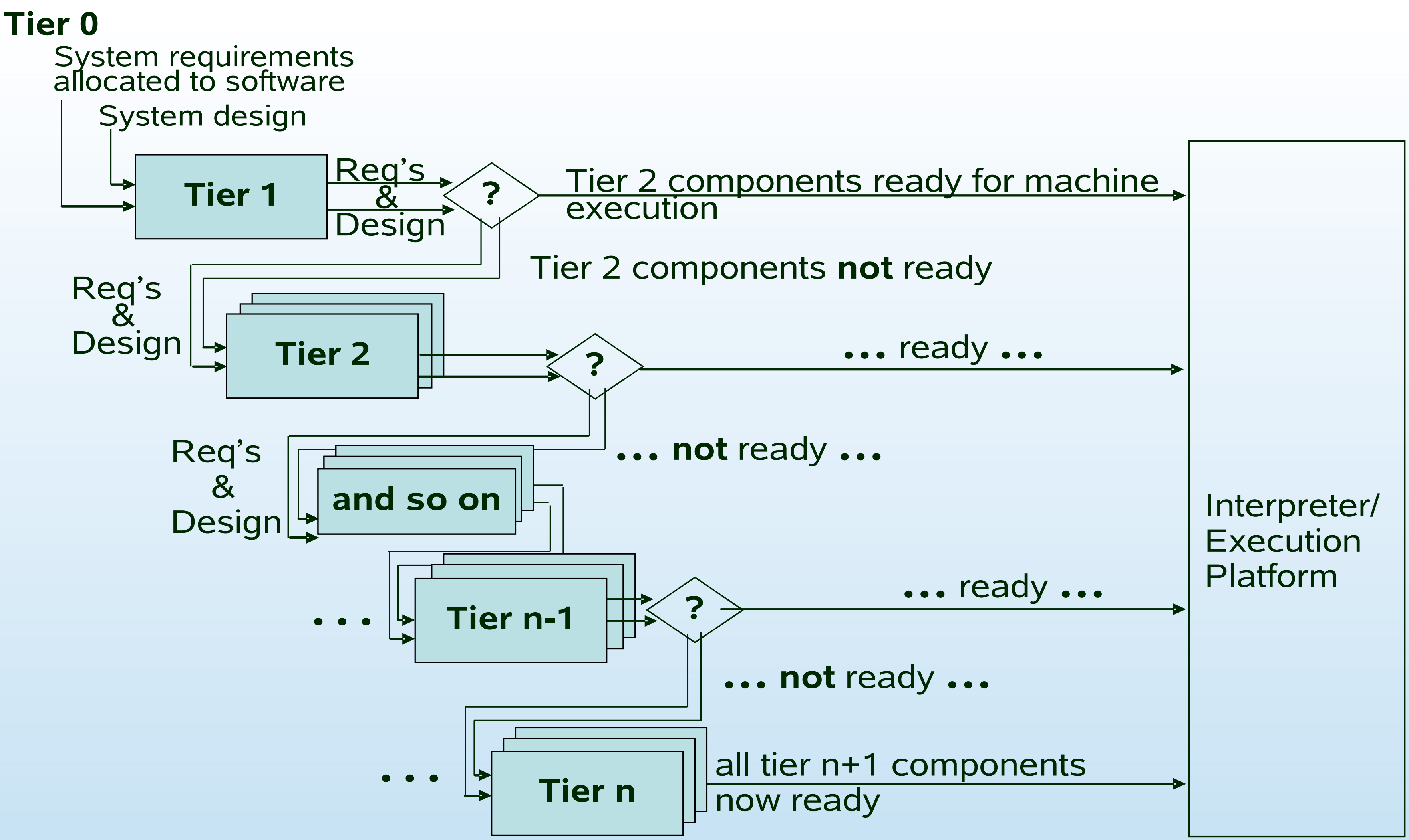
SC-205 / WG-71 Subgroups

- SG-1: Document Integration
- SG-2: Issues and Rationale
- SG-3: Tool Qualification
- SG-4: Model Bases Design and Verification
- SG-5: Object-Oriented Technology
- SG-6: Formal Methods
- SG-7: Safety and CNS Related Considerations (communication, navigation, surveillance)

DO-178B Verification



DO-178C is Multitiered



OOT Key Features

- Prove required that OOT doesn't introduce vulnerabilities in certified application
  - Inheritance and redefinition
  - Polymorphism
  - Type conversion
  - Overloading
  - Exception management
  - Dynamic memory management
  - Virtualisation

Inheritance and Redefinition

- Multiple inheritance
  - Interface level
  - Implementation level
- Vulnerabilities
  - Indeterministic dispatch time
  - Semantic dissonance
  - Implementation dissonance
- Objectives
  - Fulfil specifications of all parent classes
  - Include full class model in design

Method Dispatch

- Static vs. dynamic dispatch
  - Static: called method depends on *declared* type
  - Dynamic: called method depends on *real* type
- Vulnerabilities:
  - Mixing static and dynamic dispatch in one application might cause confusion
- Guidance:
  - Style guide should specify which dispatch is to be used.
  - In Java, method dispatch is always dynamic

Subclassing / Subtyping

- Method and class specification:
  - Preconditions: acceptable input values
  - Postconditions: return values, including exceptions and errors, and side effects
  - Invariants
- Subclass, subtype equivalence
  - Liskov's substitution principle:
    Preconditions may not be strengthened, postconditions and invariants may not be weakened.

Ad Hoc Polymorphism (Overloading)

- Improves readability and maintenance
- Vulnerabilities:
  - Ambiguity due to implicit type conversion
- Guidance:
  - Use explicit type conversion instead

Parametric Polymorphism

- Enables reuse without subtyping
- Vulnerabilities:
  - Substitution mismatch
  - Unverified code
- Guidance:
  - Each instantiation of parametric type needs to be verified
- Objectives
  - Ensure type consistency
  - Ensure all code is covered

Type Conversions

- Vulnerabilities
  - Data loss
  - Data corruption or exception
- Objectives
  - Ensure that type conversions are safe
- Data Flow Analysis can statically prove correct typing
- In Java, wrong type casts at least throw an Exception

Exceptions

- Separating exceptional behaviour from normal behaviour
- Vulnerability
  - uncaught or improperly handled exception
- Objective
  - ensure all exceptions are properly handled
  - test coverage includes exceptional control paths
- Data Flow Analysis can statically prove that all exceptions are handled

Dynamic Memory Vulnerabilities

1. Ambiguous references
2. Fragmentation starvation
3. Deallocation starvation
4. Heap memory exhaustion
5. Premature deallocation
6. Lost update or stale reference
7. Indeterministic allocation or deallocation

Dynamic Memory Safety Objectives

1. Unique allocation
2. Fragmentation avoidance
3. Timely deallocation
4. Sufficient Memory
5. Reference consistency
6. Atomic move
7. Determinism

Memory Management

Technique	Ambiguous References	Fragmentation Starvation	Deallocation Starvation	Heap Memory Exhaustion	Premature Deallocation	Lost Update / Stale Reference	Indeterministic Allocation / Deallocation
Manual Heap Allocat	✗	?	✗	✗	✗	N/A	✔
Object Pooling	✗	✗	✗	✗	✗	N/A	✔
Stack Allocation	✗	✔	✔	✗	✗	N/A	✔
Scope Allocation	✔	✔	✔	✗	✗	N/A	✔
Automated Heap Allocation	✔	✔	✔	✗	✔	✔	✔

✔ = prevented automatically, ✗ = by the application
N/A = not applicable, ? = difficult to ensure

Virtualisation Techniques

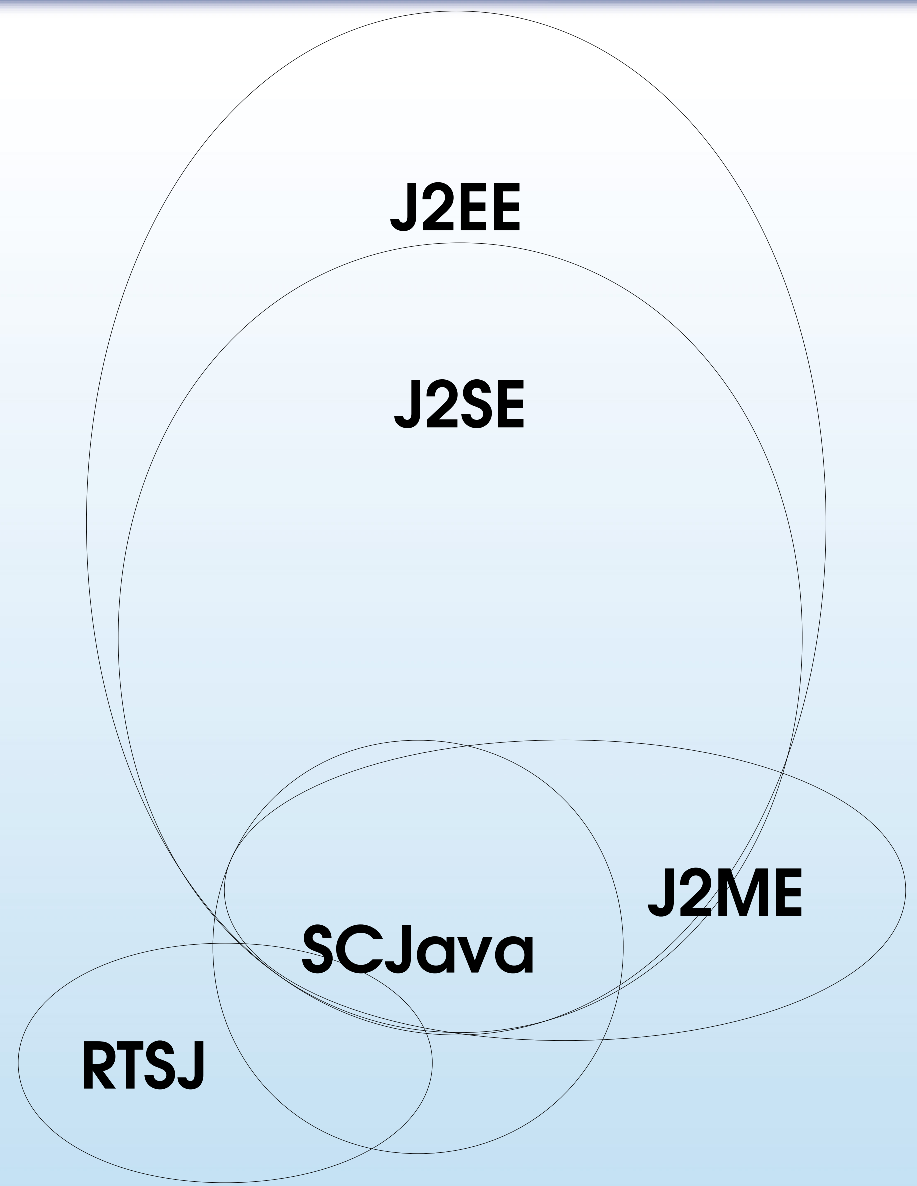
- Vulnerability:
  interpreted code treated as data and not validated
- Objective:
  Certify system in layers
  - Certify interpreter where its input is treated as data
  - Certify interpreted program as code where interpreter is treated as execution platform
- Applies to any data that is interpreted

Java Advantage over C/C++

- Clean syntax and semantics w/o preprocessor
  - ➔ wide ranging and better tool support
- Multiple inheritance on interface level
- No explicit pointer manipulation
- Pointer safe deallocation
- Single dispatch style
- Strong, extendible type system
- With RTSJ, well defined tasking model

Java Variants

- J2EE—J2SE & enterprise extensions
- J2SE—Standard Java
- J2ME—Subset of J2SE & additional classes
- RTSJ—Add on to J2EE, J2SE, or J2ME for realtime
- SCJava—Subset of RTSJ, subset of J2SE, & additional classes



Certifying a Garbage Collector

- Not possible for all collectors
  - Must be deterministic; no unbound steps
  - Must assume maximum memory use
  - Must consider allocation rate
- Example: Jamaica Collector
  - No root scan and compaction (unbound)
  - Mark and sweep steps on fixed size blocks
  - Automatically tracks allocation rate
  - GC work performed at allocation time
    - Other threads not influenced

Summary

- Certification of OOT introduces new questions
- Automated Memory Management safer than manual for complex tasks
- Java brings safety and reliability to complex applications
- Clear Guidelines for OOT in the DO-178C standard will ease Certification of Java applications
- DO-178C will allow for certifiable Garbage Collectors