Software Languages:

Are We Better Off in 2007 Than We Were in 1975?

Is Java the Answer?

Systems & Software Technology Conference
Tampa Bay, FL

June 2007

1975 …

- Japanese World War II Surrender
  - Teruo Nakamura – Morota - Indonesia
- Vietnam
- Watergate Sentencing
- Altair 8800 Microcomputer
- Rocky Horror Picture Show
- Angola & Mozambique Gain Independence
- Viking 1 Launched Towards Mars
- Attempted Assassinations of Gerald Ford
- Apache Helicopter Makes First Flight

1975 …

- DoD Decides to Overhaul Computer Languages Used by its Contractors
  - Over 450 Languages (inc. Dialects) in Use
  - Software Costs Were Rising
  - Impossible to Use Programmers and Software Across Projects
  - Major Source of Post-Deployment Problems, Interoperability, Operations, Maintenance, and Product Line Management Problems

1976 …

- DoD Directive 5000.29
  - Approved High-Order Languages
    - COBOL
    - Fortran
    - TACPOL
    - CMS-2
    - SPL/1
    - JOVIAL J3 & J73
- None of the Existing Languages Met the Requirements for Embedded Systems

1976 …

- DoD Designs their Own Language, Documenting the Requirements
  - Strawman - 1975
  - Woodenman - 1975
  - Tinman - 1976
  - Ironman - 1977
  - Steelman - 1978

1977 …

- Higher Order Language Working Group Evaluates All “Modern Languages” and Concludes that No Language Meets the Specifications of the Ironman Specification.
- Issues Requests for Proposals for a New Language to Meet the Specification of the HOLWG.

1978 …

- 4 Proposals Received
  - Red — Intermetrics
  - Green — CII Honeywell Bull
  - Blue — SofTech
  - Yellow — SRI International
- Red & Green Were Accepted

1979 …

- Jean Ichbiah's Green Proposal was Accepted and Given the Name Ada After Augusta Ada King, the Countess of Lovelace, Daughter of Lord Byron.

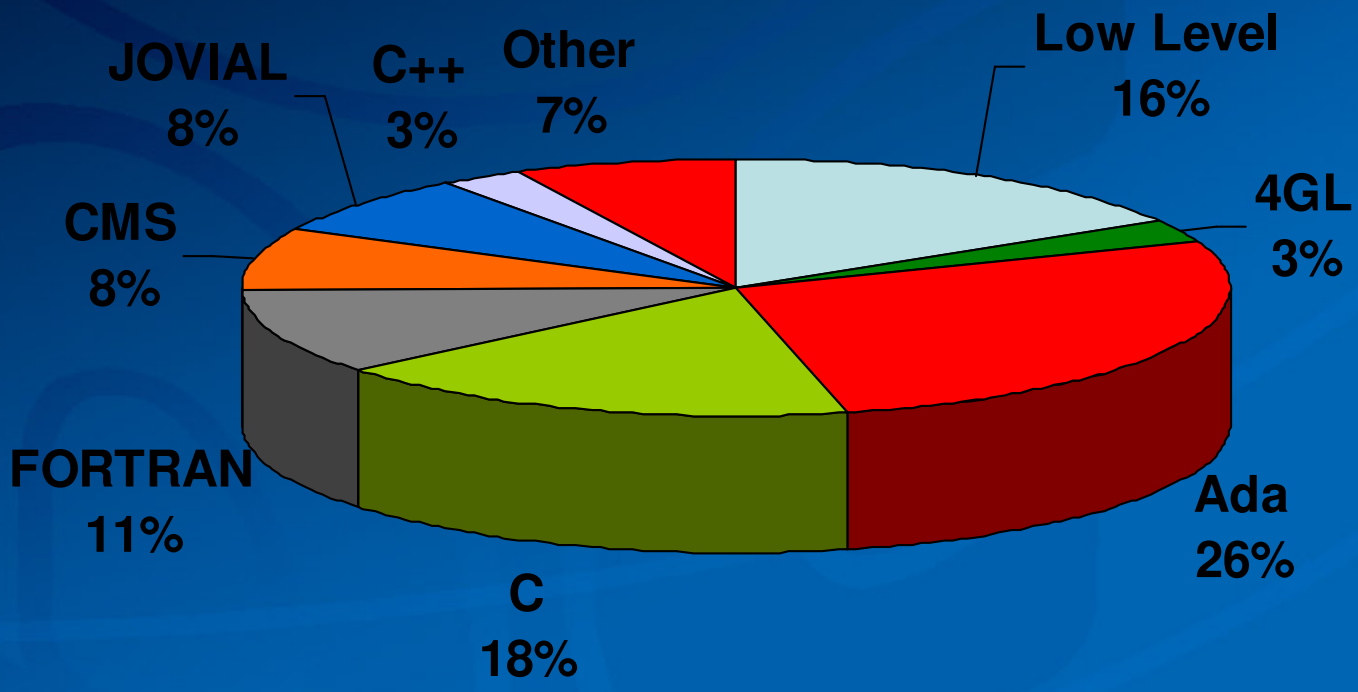
1983 …

- Ada Compilers Made Available
- Ada Language Mandated by DoD Directive 3405.1 for Most DoD Procurements and Internal Developments
- Minimal BASIC and Pascal were added to the List of High Order Approved Languages

But … While We Were Sleeping

- 1972 – C
- 1978 – C Programming Language was Published
- 1984 – C Compilers for Microprocessors
- 1986 – C++
- 1989 – ANSI C
- 1995 – Sun Announces Java
- 1996 – ANSI C++
- 1998 – ISO C++

1994 ...


JOVIAL
8%
C++
3%
Other
7%
Low Level
16%
4GL
3%
CMS
8%
FORTRAN
11%
C
18%
Ada
26%

And … Universities Taught

1970s – Algol, Fortran, Pascal

1980s – Pascal, C

1990s – C, C++, Ada

2000s – Java, 4GLs

Created a Socio-Economic Climate Where Students Perceive that only New Languages Offer Job Security

Today …

- DoD Directives (1983)
  - COBOL
  - Fortran
  - TACPOL
  - CMS-2
  - SPL/1
  - JOVIAL J3 & J73
  - BASIC
  - Pascal
- In Practice
  - Fortran √
  - JOVIAL J73 √
  - C √
  - C++ √
  - Ada 83 √
  - Ada 95 √
  - Java √

Fortran

Legacy Code Only

JOVIAL J73

Legacy Code, Mainly in Aircraft on 1750A Processors

C

- Popular Language of Choice for Many New Programs, or When Ada / JOVIAL Programs are Migrated to Another Modern Language
- Accompanied by Coding Standard to Restrict Use of Unsafe C language Constructs
- C^{b} Constrained Subset of MISRA C

C++

- Safe Version of C++ is Also a Popular Language of Choice for Many New Programs, or When Ada / JOVIAL Programs are Migrated to a Modern Language.
- Each Company has Their Own Coding Standards to Create the "Safe Version"
  - No Exceptions
  - Strictly Defines How Inheritance is Used
  - JSF Adds 220 Rules to Make C++ Safe
  - Embedded C++ a Popular Dialect

Ada

- Ada is Still Regarded as the Closest Language to the Ideal Defense Language
- Most Legacy Code is Written in Ada 83
  - Still the Largest Language Code Base in Embedded Operational Military Applications
  - SPARK Ada for Safety Related Applications
- Developers are Starting to Move from Ada83 to Ada95 when Upgrades are Required
- Some Movement to Translate Ada Code to Other Languages

Ada – Why is it Not Popular?

- Perceived by Software Engineers as a Niche Language
  - Getting Locked in to Ada Restricts Your Resumé
- Perceived by Managers as an Expensive Language
  - Original Certification Requirements Made the Initial Release of a Compiler a Lengthy Process
- Biggest Problem is Lack of Ada Experience
  - Difficult to Find Ada Programmers

Ada – Why is it Not Popular?

- Perceived by Universities as "Not Cutting Edge"
  - Universities are All Teaching Java as the Primary Language
    - Industry wants Java
  - Ada is Mentioned in a General Course on Programming Languages, and One or Two Features Described to Contrast it With Other Languages.

Java

- Object-Oriented
- Architecture Neutral
- Robust
- Secure
- Dynamic
- Easy to Read and Write
- Small Language
  - Spec is Only 80 Pages (cc 200 pages for C)
- Automatic Memory Reclaiming
- Inherently Multi-threaded

What is Pushing Java?

- Language of Choice at Educational Institutes
- Most Widely Deployed
  - Cell Phones
  - PDAs
  - Web Applications
- Scaleable Through Several Editions
  - Standard Edition
  - Micro Edition
  - Enterprise Edition
- Industry

Java Improves on C/C++

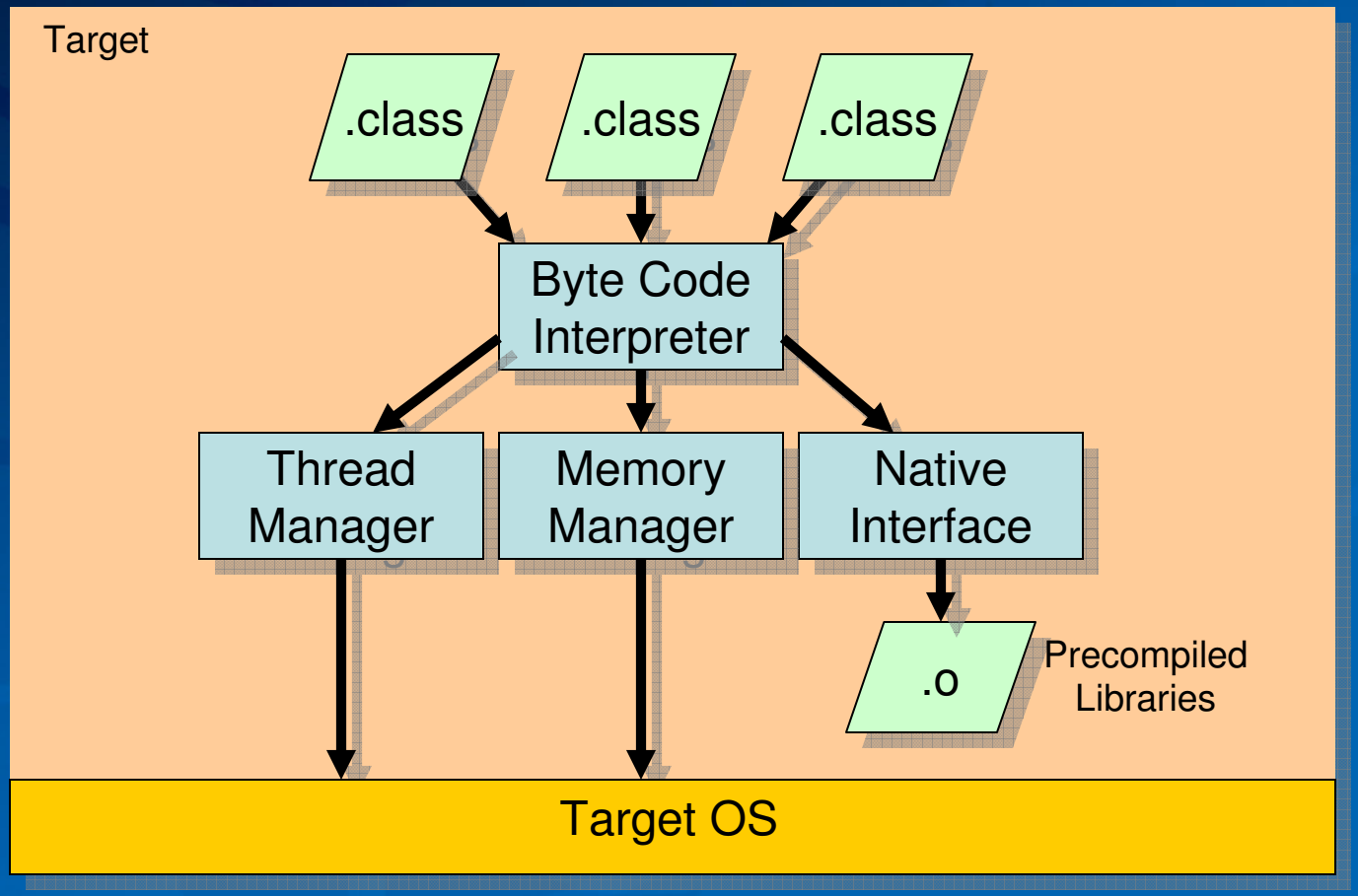
- Removes Dangerous C Operations
  - No Operator Overloading
  - No Redundant Features
- Everything is a Class
  - References Have to Point to an Object
  - One Main Object References All Other Objects
    - Directly or Indirectly
- No Header Files
- Automatic Garbage Collection
  - No Dangling References
  - No Memory Leaks

But – Java is Really More …

… a Specification of a Computer Language with the Underlying Implementation to Allow Easy Deployment

- Write Once, Run Anywhere

Java Run-Time Environment

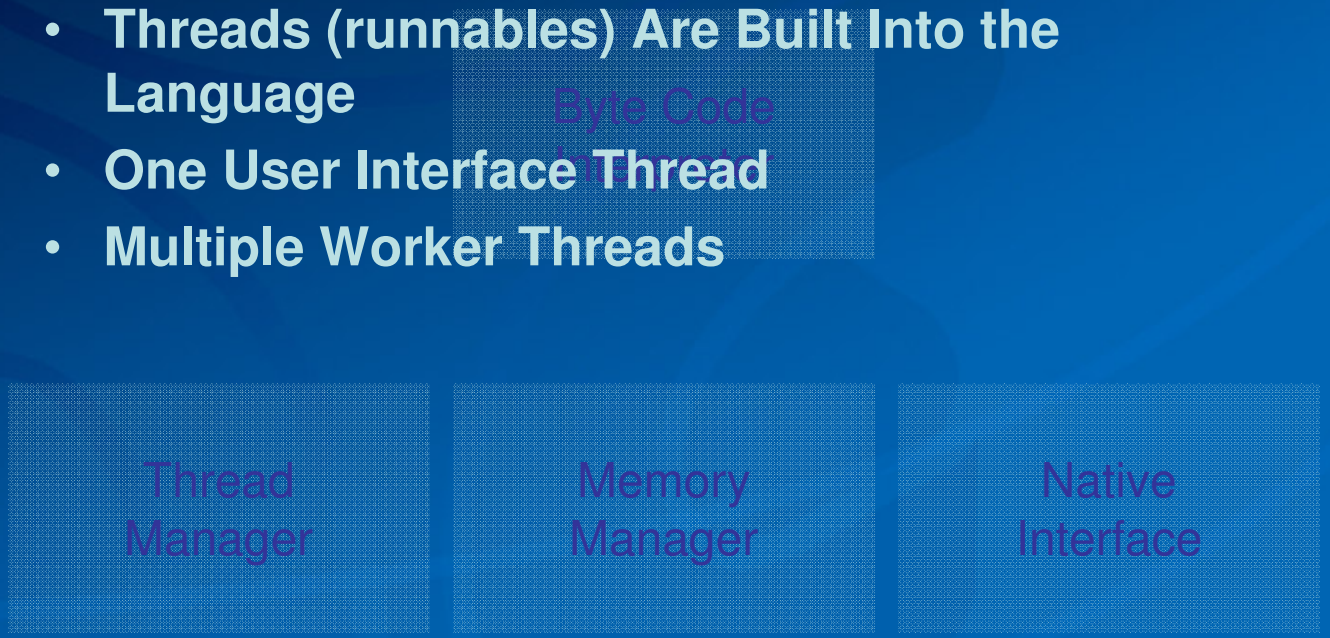


Java Virtual Machine

- Byte Code Interpreter
  - Byte-by-Byte Interpretation
  - Just-in-Time (JIT) Compilation
    - Most Widely Used
    - Very Advanced Algorithms
    - As Fast as Running Native C++ Code
  - Ahead-of-Time (AOT) Compilation
    - Complete AOT Compilation Prohibits Dynamic Class Loading

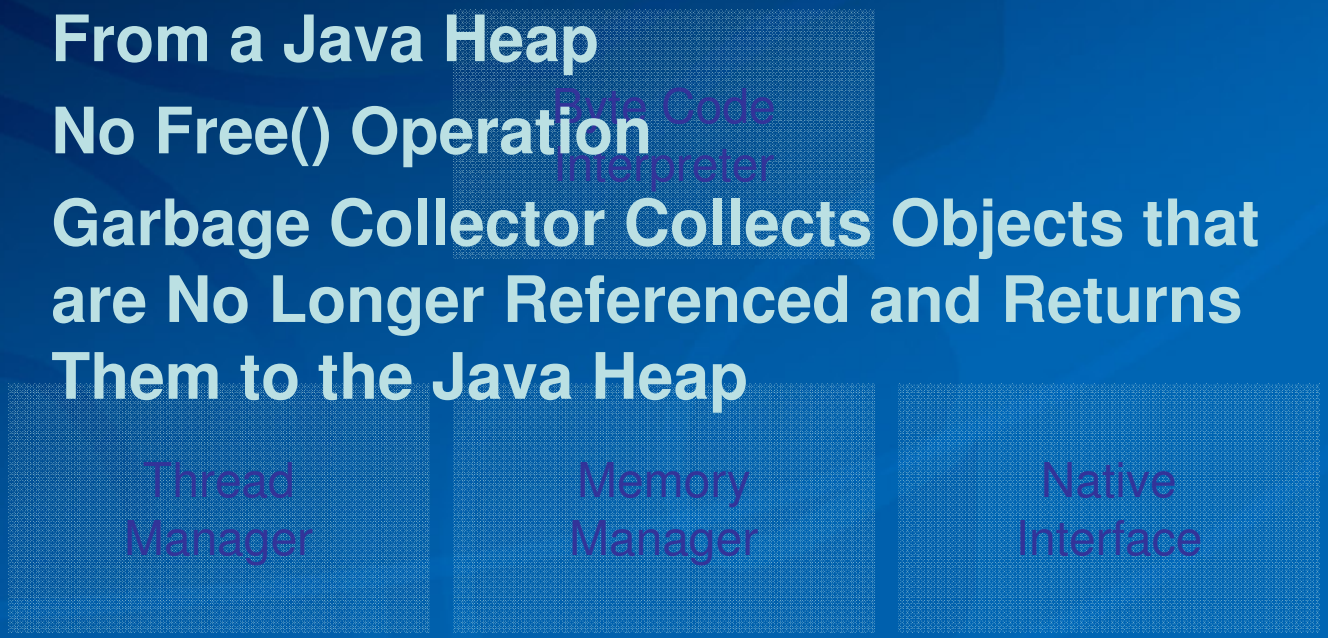
Java Thread Manager

- Java is Inherently Multi-threaded
  - Threads (runnables) Are Built Into the Language
  - One User Interface Thread
  - Multiple Worker Threads



Java Memory Manager

- Classes Store Data in Memory Allocated From a Java Heap
- No Free() Operation
- Garbage Collector Collects Objects that are No Longer Referenced and Returns Them to the Java Heap



Java Native Interface

- Interface Between Java and Compiled Code
  - Must Ensure That Objects Passed to Native Code Are Not Collected by the Garbage Collector
  - Allows Interfacing to C and Ada
- Good Transition Path From Legacy Code
  - Mixed Language Development

Why Not Java?

- No REAL-TIME Capability
- Not Deterministic
- Poor Scheduling
- Cannot Access Hardware Directly
  - Physical Memory
- No Asynchronous Event Handlers
  - Interrupt Handlers
  - Signal Handlers

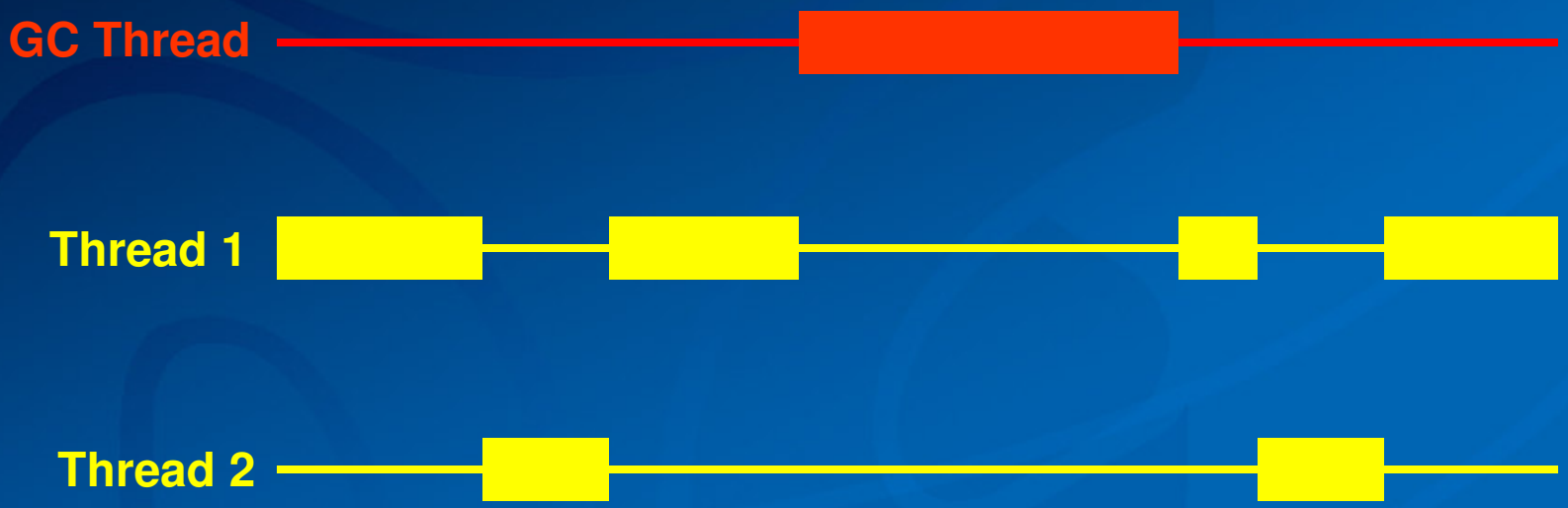
Real-Time Requirements

- Real-Time Behavior Requires Timing Deadlines to Be Met – Always
- Real-Time Behavior Does Not Need to Be Fast
  - Speed Can Be Sacrificed to Achieve a Bounded Predictable Response
- Main Barrier to Predictable Real-Time Performance in Java is the Garbage Collector

Garbage Collector

- Task That Runs to Free Objects No Longer Used
  - All Code is Stored in Objects
  - All Objects Use Memory From the Heap
- When the Garbage Collector Runs:
  - All the Threads Stop
  - The Garbage Collector Re-arranges the Heap, Freeing up Objects That Are No Longer Referenced
  - The Threads Continue

Garbage Collector



Garbage Collection

- Method 1 – Two Heaps
  - Starting From the Main Program, All the Active Objects Are Copied From Heap 1 to Heap 2
  - Heap 2 is Made the Active Heap
    - No De-fragmenting is Required
  - “Dead” Objects Are Not Copied
  - Heap 1 is Re-initialized

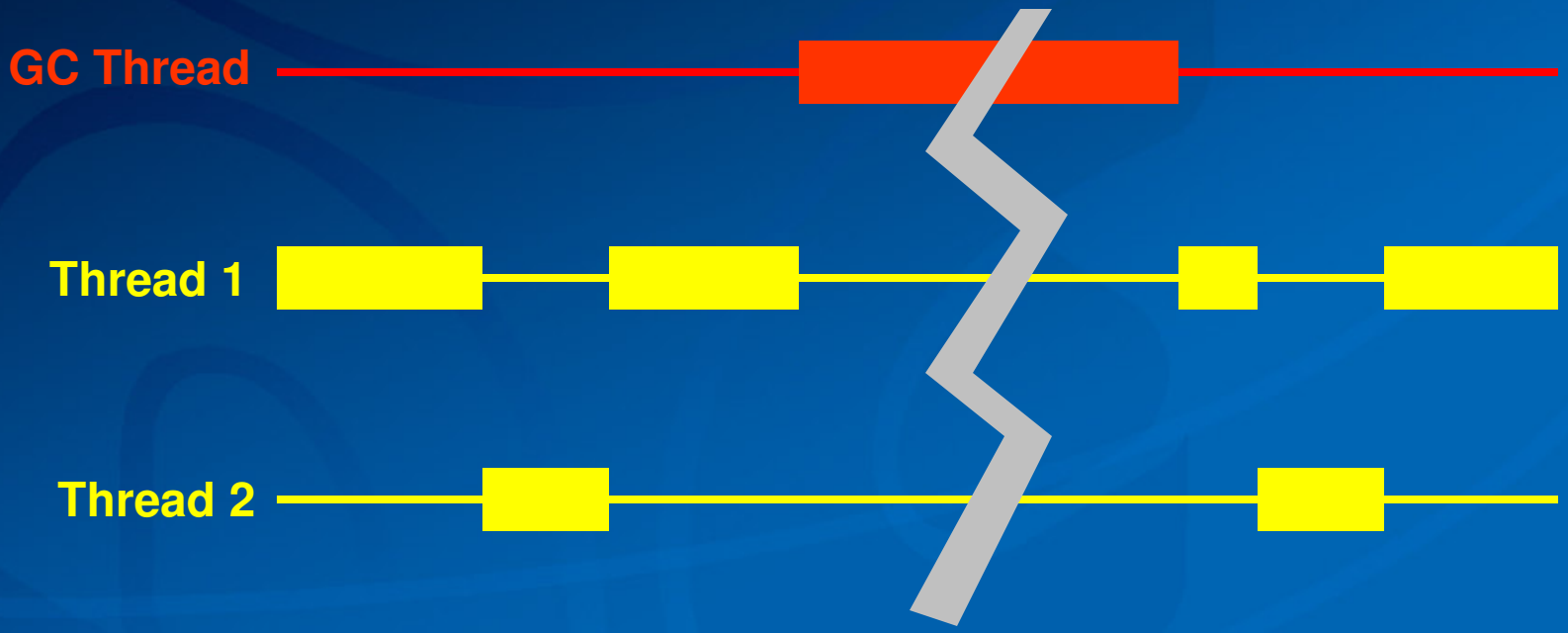
Garbage Collection

- Method 2 – Mark and Sweep
  - Starting From the Main Program, All the Active Objects Are Marked as Being "Alive"
  - The Heap is Swept, and All the Objects Not Marked as "Alive" Are Freed, and the Marks are Removed in Preparation For the Next Sweep
  - De-fragment the Heap

Garbage Collection

- Problem
  - Impossible to Predict When the Garbage Collector Will Run
  - Impossible to Predict How Long Your Program Will Be Suspended While the Garbage Collector Runs
  - No Real-Time Capability

Garbage Collector



Real-Time Java (RTSJ)

- A Set of Extensions to the Java Language That Enables the Construction of Systems That Require Real-Time Behavior
- First Approved in 2001, Revised 2006
- Add 2 New Data Storage Areas
  - Immortal Memory
  - Scoped Memory
- Added Real-Time Threads
- Added Memory Access
- Added Asynchronous Transfer of Control

Immortal Memory

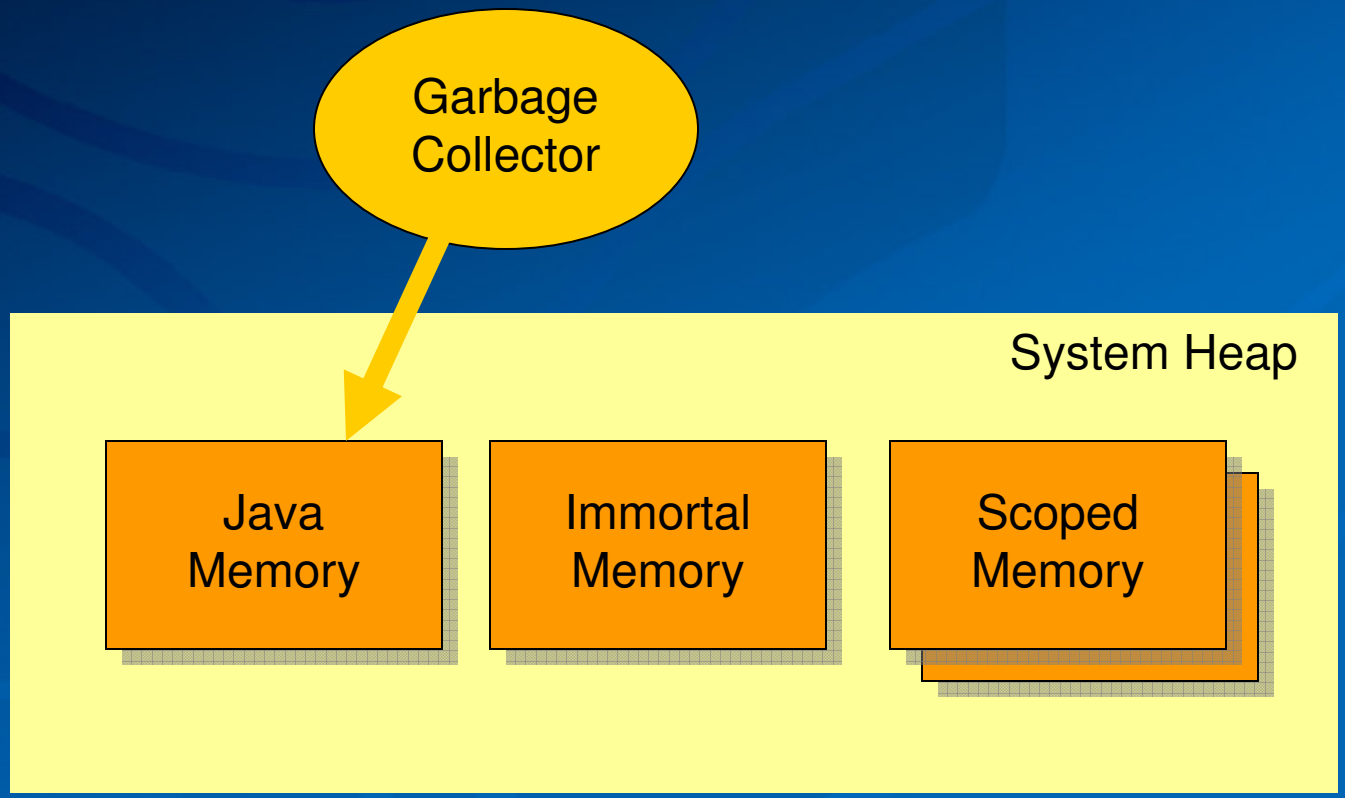
- Objects in Immortal Memory Are Never Subject to Garbage Collection
  - Programmer Beware
  - Can Introduce Leaks if Not Used Properly, But Generally This Can Be Limited to a Small Subset of the Program
- Great for Some Embedded Applications
  - Most of Your Data Structures Are Static and Have a Life That Ends When You Switch Off

Scoped Memory

Scoped Memory is an Object With its Own Memory Area, Capable of Executing Code

- The Memory is Not Subject to Garbage Collection
  - Memory is Allocated When the Program Enters a Scoped Memory Object
  - Within the Object, All Memory Requests Are Made From the Scoped Memory
  - All the Memory is Freed When the Scoped Memory Object is Exited
- Programmer Beware

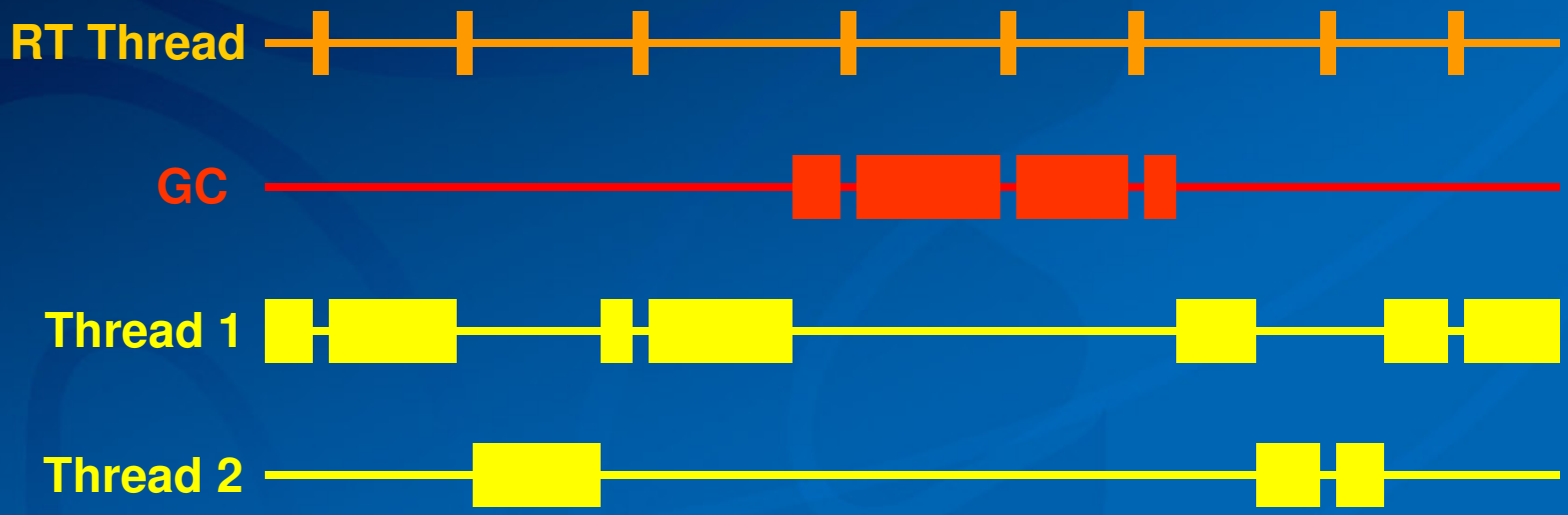
RTSJ Memory


Garbage
Collector
System Heap
Java
Memory
Immortal
Memory
Scoped
Memory

RTSJ - Real-Time Threads

- RTSJ Adds the Concept of a Real-Time Thread
  - Higher Priority Than the Garbage Collector
    - 32 Different Priorities
  - May Interrupt the Garbage Collector
  - May Not Use Memory That is Controlled by the Garbage Collector
    - Only Uses Immortal Memory or Scoped Memory

Real-Time Thread



Real-Time Threads

- Applications Must Be Split into Two Parts
  - Real-Time Part
    - Must Not Use Normal Java Memory
  - Non Real-Time Part
    - Must Not Block Real-Time Threads (Priority Inversion)
  - Special Java Classes are Required to Send Messages Between Parts
    - WaitFreeQueues

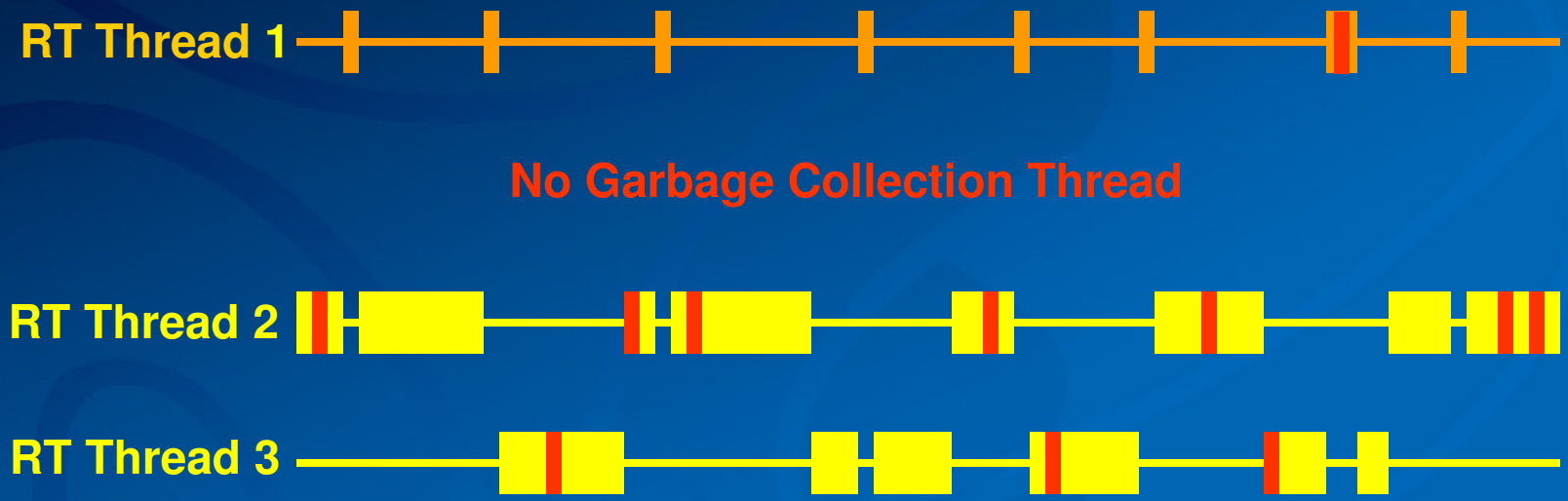
Alternative to RTSJ Method

- Real-Time Garbage Collection
  - Not Part of the Java Standard
  - Not in Conflict With the Standards
  - Does Not Add to the Standard
    - Uses the Same API as the Standard
    - Uses the Same Byte Codes as the Standard
- Real-Time Performance Without Using Real-Time Threads

Real-Time Garbage Collection

- Does Not Require Scoped Memory or Immortal Memory
- Distributed Concurrent Garbage Collection
  - Proposed by Dijkstra et. al. in 1978
  - Mark and Sweep Algorithm
  - Each Increment Only Looks at a Finite Size of Memory
  - Each Pass is Bounded in Time, and Will Only Occur at Known Points (new)

All Threads Are Real-Time



Real-Time Garbage Collection

- All Threads May Now Have Real-Time Performance
  - Real-Time Does Not Mean Fast - It Means Predictable Timing
- No Special Memory Requirements
- May Still Use Scoped or Immortal Memory to Reduce Amount of Memory to Be Considered for Garbage Collection

Safety Critical Aspects

- Main Requirement of Any Safety Critical Application is Formal Testability to a Standard
  - DO-178B/ED-12B (Avionics)
- Formal Methods
  - Schedulability Analysis
  - Modified Condition/Decision Coverage (MC/DC) Analysis

Proved by Testability

- Proof of Deterministic Behavior
- No Dead Code – All Branches of the Code Must Be Reachable with Test Cases
- Each Pass of the Same Test Must Give the Same Results
  - Avoid Garbage Collection at Random Places During the Test

Safety Critical Java Technology

- JSR-302 Experts Group Aims to Have Safety Critical Java Meet the Requirements of the DO-178B Standard, Level A
- Members (Companies Only)

aicas	Anteon
Aonix	Apogee
AXE Inc.	Boeing
DDC-I Inc.	IBM
Mitre	Raytheon
Rockwell Collins	Siemens
Sun	The Open Group

Safety Critical Java Technology

- Targeted at Embedded Java Platform (J2ME)
- Goal is to Subset RTSJ – Not a New Java Platform
- Code Must Run on Other Java Platforms Different Requirements for Different Levels of Criticality

- http://jcp.org/en/jsr/detail?id=302

Testability (DO178B-A)

- Highest Criticality Level
  - Catastrophic Failure
- Entire Application Including Operating System Must Be Certified to Level A
- JSR-302 Experts Group Expects Only Pre-compiled Code Will Be Allowed

Testability (DO178B-C)

- Non-Hazardous (Mission Critical)
- Entire Application Including Operating System Must Be Certified to Level C or Above
- JSR-302 Experts Group Expects Interpreted Code Will Be Acceptable
  - Combinations of Pre-compiled Code and Interpretation
- Separation of Criticality Levels Will Be Up to the Operating System

SCJ EG Timeline

- Original Dates
  - First Meeting: July 31, 2006
  - Early Draft Review: October 1, 2006
  - Public Review: December 4, 2006
  - EC Approval: February 5, 2007
- Current Dates
  - Start Drafting the Specification in Paris April 24 2007

Byte Code Interpreter

- Too Large to Be Certified to Level A
- Today's Interpreter is Complex, with Just-In-Time Compiling and Predictive Algorithms to Speed Up Execution of the Code
- Removing the Byte Code Interpreter Requires an Ahead-Of-Time Compiler

Java Libraries

- Too Large to Be Certified to Level A
- Safety Critical Versions of a Subset of the Libraries Will Be Produced
  - Programmers Will Be Restricted to Use These Versions of the Libraries

Other Restrictions

- No Dynamic Loading of Classes
  - Permits AOT Compilation
- Class Initialization Done “Up-Front”
  - Statically Ordered
- No JIT Compiling
  - JIT Compiling Produces an Initial Compilation Delay – Unacceptable for Deterministic Execution

Conclusion

- Java is the Language of the 2000's
- Real-Time Java is a Reality Today
- Safety Critical Java is Just a Matter of Time
  - Millions of Lines of Java Code Will Be Used Where Lives Depend on Them
  - < 5 Years
- Is Java the Answer ?
  - Time Will Tell

Thank You

Contact:

Sales: 602-275-7172
sales@ddci.com

Alex Polmans: 602-386-4362
apolmans@ddci.com

www.ddci.com

Acronyms

- ANSI – American National Standards Institute
- AOT – Ahead-of-Time Compilation
- API – Application Programming Interface
- DoD – Department of Defense
- HOLWG - Higher Order Language Working Group
- JIT – Just-in-Time Compilation
- JSF – Joint Strike Fighter
- MISRA – Motor Industry Software Reliability Association
- RTSJ - Real-Time Specification for Java
- SPARK -
- 4GL – 4th Generation Language