Meeting Your Most Critical Requirements

Making Effective Use of Java in Real-Time Systems

Kelvin Nilsen, CTO

Presenter Background

- 1988: Completed Ph.D. research on high-level programming of real-time systems
  - Published real-time garbage collection paper in Software Practice & Experience (July 1988)
- 1988-1996: Research and instructional faculty at Iowa State University, focusing on programming language design and implementation for real-time software
- 1996-2003: Founded NewMonics to develop and market real-time virtual machine products, acquired by Aonix
  - Original architect and designer of PERC product
  - Contributor to NIST Requirements for Real-Time Java

Presenter Background

- 2003-current: CTO of Aonix, combining strengths of Ada, safety-certified run-time kernels, modeling tools, and real-time virtual machines
  - Key contributor to Open Group standardization of mission-critical and safety-critical Java specifications
  - Architect and designer of JRTK and JRaven products

Aonix

Productivity Benefits of Java

- Many experiments have concluded that Java developers are at least twice as productive as C or C++ developers. Why?
  - Stronger type checking reduces programmer errors
  - Object-oriented encapsulation reduces interference between independently developed components
  - Forced exception handling reduces programmer errors
  - Automatic garbage collection simplifies memory management and reduces programmer errors
  - Simpler language reduces programmer misunderstandings
  - Improved portability allows programmers to master the language, rather than mastering particular tool chains and RTOS services

Aonix

Productivity benefits of Java

- Many developers have found they are 5-10 times more productive integrating independently developed Java components than C or C++ components. Why?
  - Improved portability eliminates the need to "port" components to each new hardware or operating system
  - Object-oriented abstractions reduce name collisions and other interference between components
  - Automatic garbage collection simplifies integration by eliminating need to assign responsibility for reclaiming memory used by one component but allocated by another
  - Summary: components simply work "out of the box"

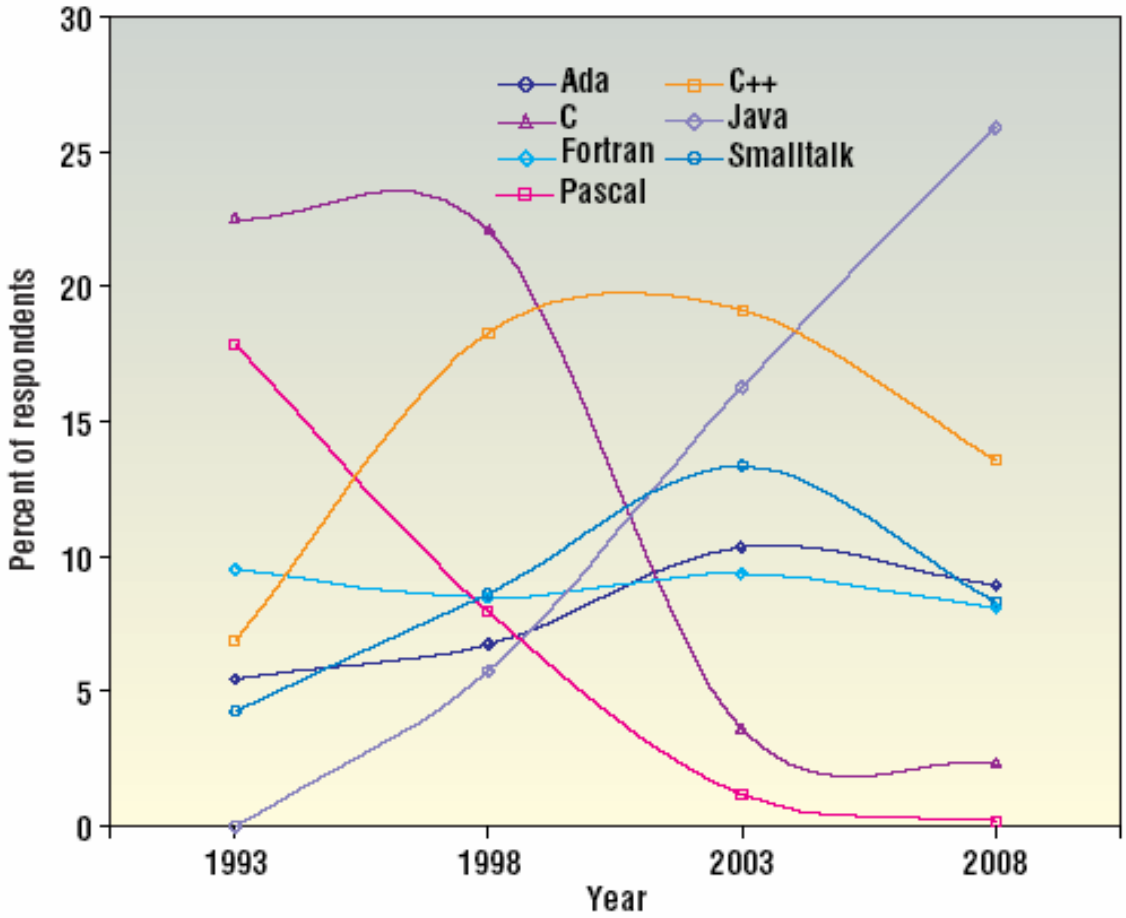
Representative experience

- Intel project: developed fault-tolerant distributed Java application in 3 days! Their assessment: “It would have taken three solid months to develop this demonstration without Java [i.e. VxWorks and C]”
- Calix customer experience report: Compared with C, the PERC real-time Java product increased developer productivity by two fold (including the time required to learn Java!), reduced code size to one fifth.
- Nortel experience: Java developers are more productive and their code more trouble-free than C++ developers.

- Trillium (now part of Continuous Computing) experience: Sells (sold) reusable network protocol stacks as C and C++ libraries. Each “product sale” was bundled with an engineer who went on-site for several weeks to assist with the integration!

Aonix

What is the Java Phenomenon?



Usage Trends of the 8 Most Popular Programming Languages

"An Empirical Study of Programming Language Trends", Chen, Dios, Mili, Wu, Wang
IEEE Software, May/June 2005

Aonix

Predictive Model Correlations

- Top Intrinsic Factors (statistical correlations)
  - Machine independence (portability) (0.8876)
  - Extensibility (scalability) (0.7625)
  - Generality (scalability) (0.6913)
  - Simplicity (-0.4703)
  - Implementability (-0.3390)
  - Reliability (scalability) (0.3199)

Aonix

Why Java for embedded development?

- Top seven stated reasons:
  - Reduced development costs (38%)
  - Availability of open-source objects and modules (30%)
  - Quicker development (29%)
  - Availability of qualified developers (21%)
  - Improved software reuse (19%)
  - Increased system functionality (18%)
  - Reduced maintenance costs (18%)

Source: Embedded Market Forecasters (2005) (108 total responses)

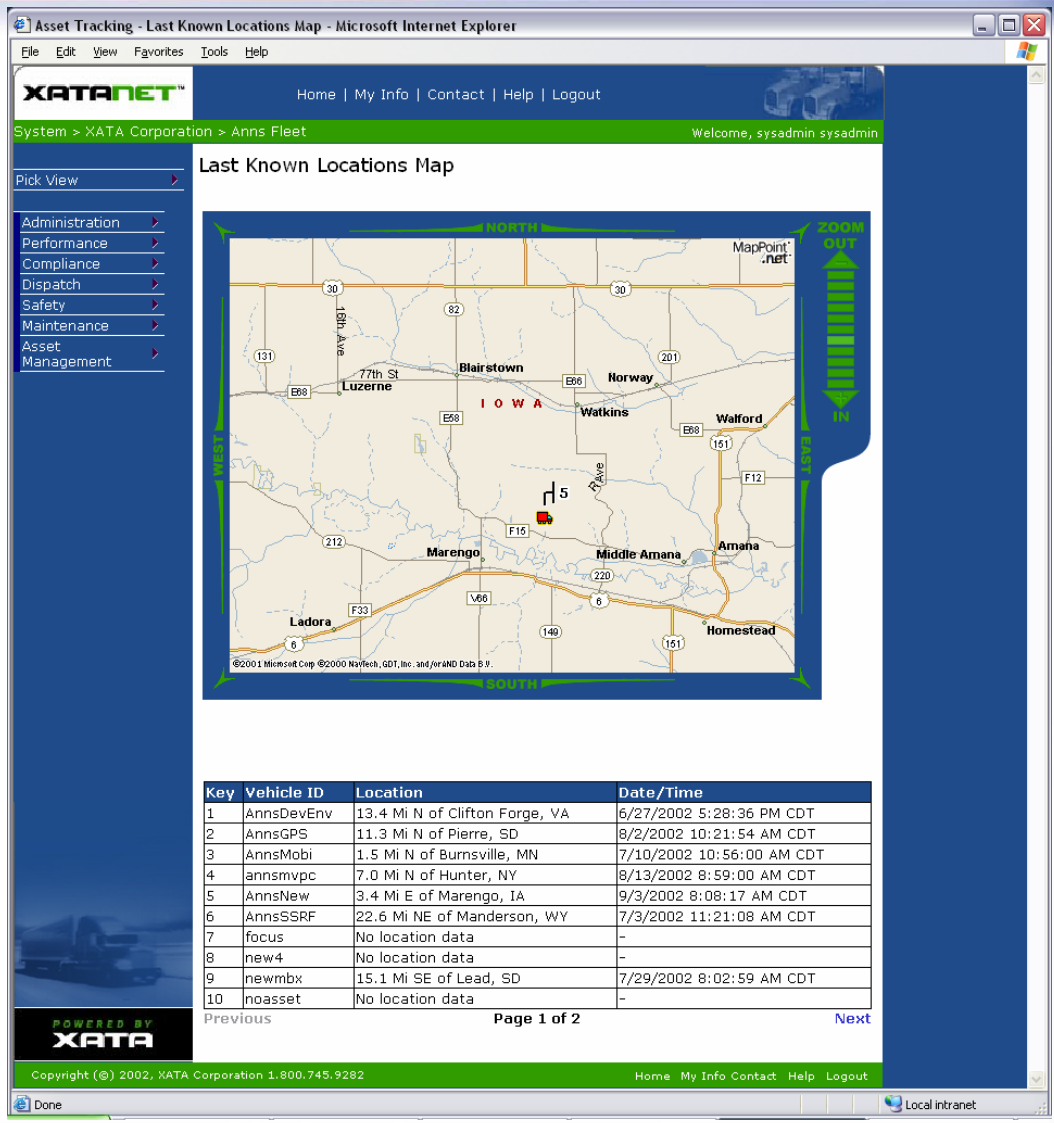
Aonix

A Soft Real-Time Profile

- Uses real-time garbage collection and J2SE APIs
  - Typical applications enforce time constraints of 1-100 ms
- The most mature approach to real-time execution of Java software, commercially available since 1997
- The easiest development, maintenance, and reuse of COTS and open-source Java components
- The only approach with proven commercial deployments:
  - Thousands of commercially deployed devices
  - Millions of hours of field-proven 5-9's reliability
- Currently supported by Aicas and Aonix. IBM and Sun claim future support

XataNet™ Fleet Telematics Application

- Real-time constrains sensor monitoring and wireless communication
- Regulatory reporting (government certification of algorithms)
- Engine and cargo diagnostic warnings
- Wireless communication with central office



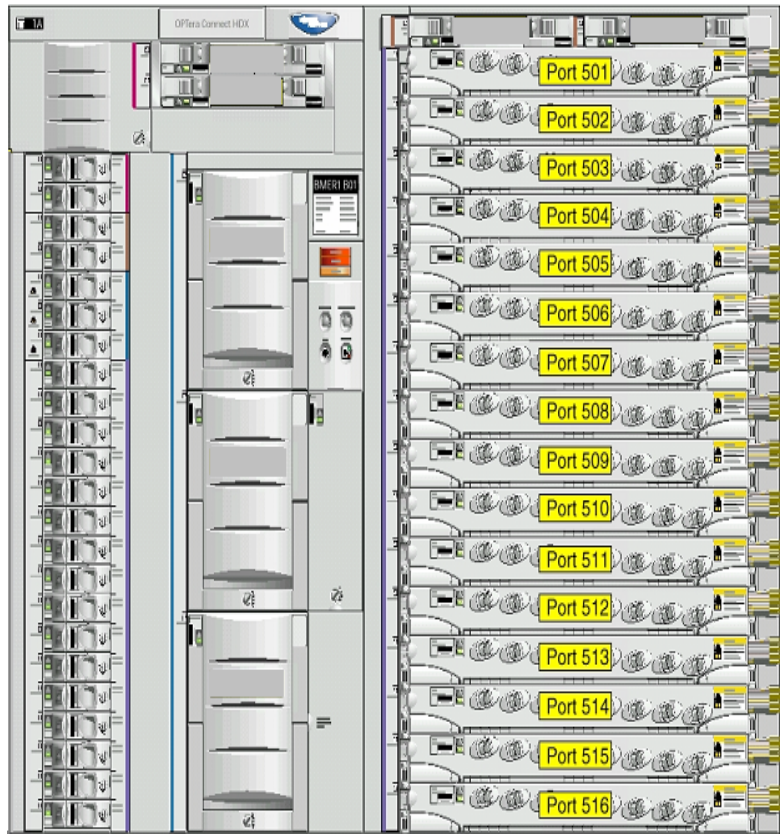
Aonix

Calix C7 Broadband Loop Carrier

- Replaced C management plane with Java: better code reuse (5x), improved developer productivity (2x), fewer bugs, and more flexible architecture
- As of 1st quarter 2005, had shipped 6,500 units to 190 service providers, supporting 1.2 million communication “ports”
- Industry segment market leader, 8 quarters running

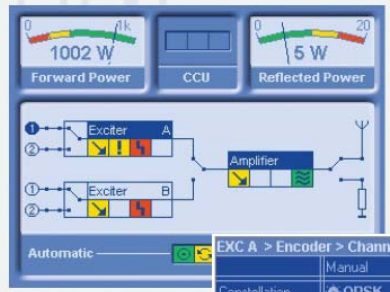
Nortel Fiber-Optic Switch Design

- Optera Connect HDX is Nortel's top-of-the-line long-haul fiber optic switch for connecting large metropolitan areas
- Java runs on every line card and on redundant shelf controllers (Power PC with VxWorks operating system)
  - Management plane consists of about 1 million lines of Java code
  - Control plane consists of approximately 4 million lines of legacy C code
- SONET fiber communication protocol has timing constraints of approximately 40 ms



TV Broadcasting by Rohde & Schwarz

- Java supports remote:
  - monitoring
  - configuration
  - provisioning.



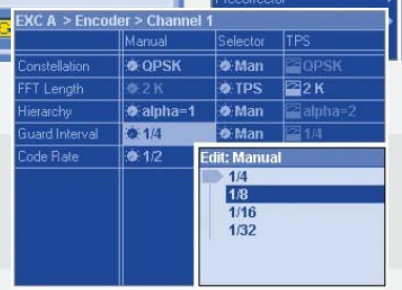


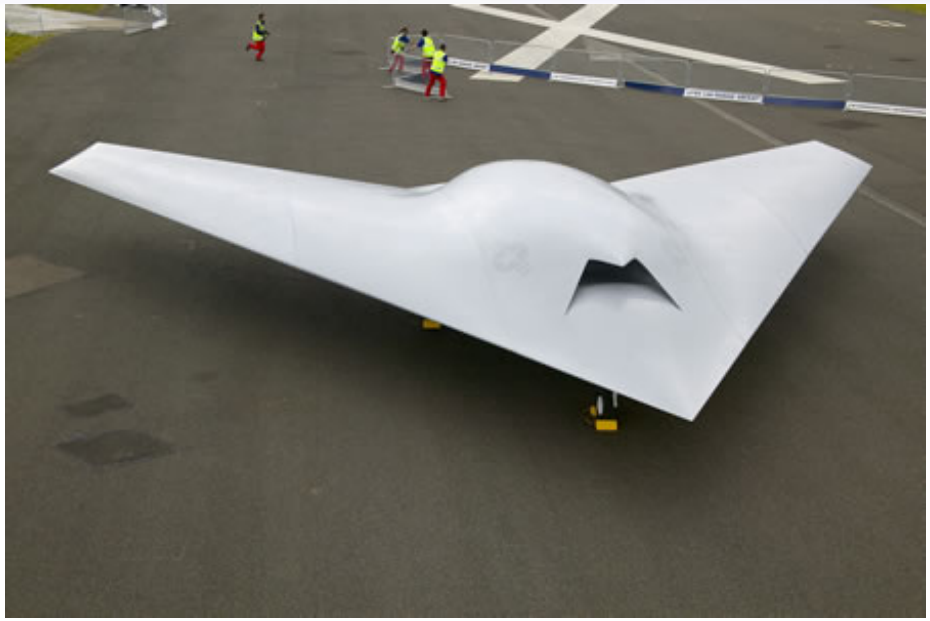
FIG 3 Operation of exciter via the R&S NetCCU®800.

Condensed data of the R&S®SX 800

ATV standards	B/G, D/K, M/N, I
Color transmission	PAL, SECAM, NTSC
Sound transmission	IRT dual sound, FM single sound, FM single sound with NICAM 728 (optional), BTSC
DTV standards	DVB-T/H in line with EN 300744 ATSC (8VSB) in line with Doc.A/53 FCC
Frequency range	band I, III, IV, V
Output power	ATV: +20 dBm sync peak power DTV: +13 dBm RMS

Aonix

Boeing J-UCAS X-45C Unmanned Aircraft



- Joint software development effort between Boeing and BAE
- Mission planning software implemented in Java
- Mission plan is continually updated to account for weather, fuel levels, weapons deployment status, enemy activities, and evolving objectives
- BAE characterization: “solve the traveling salesman problem in real time”

Aonix

French Military FELIN Project by Sagem

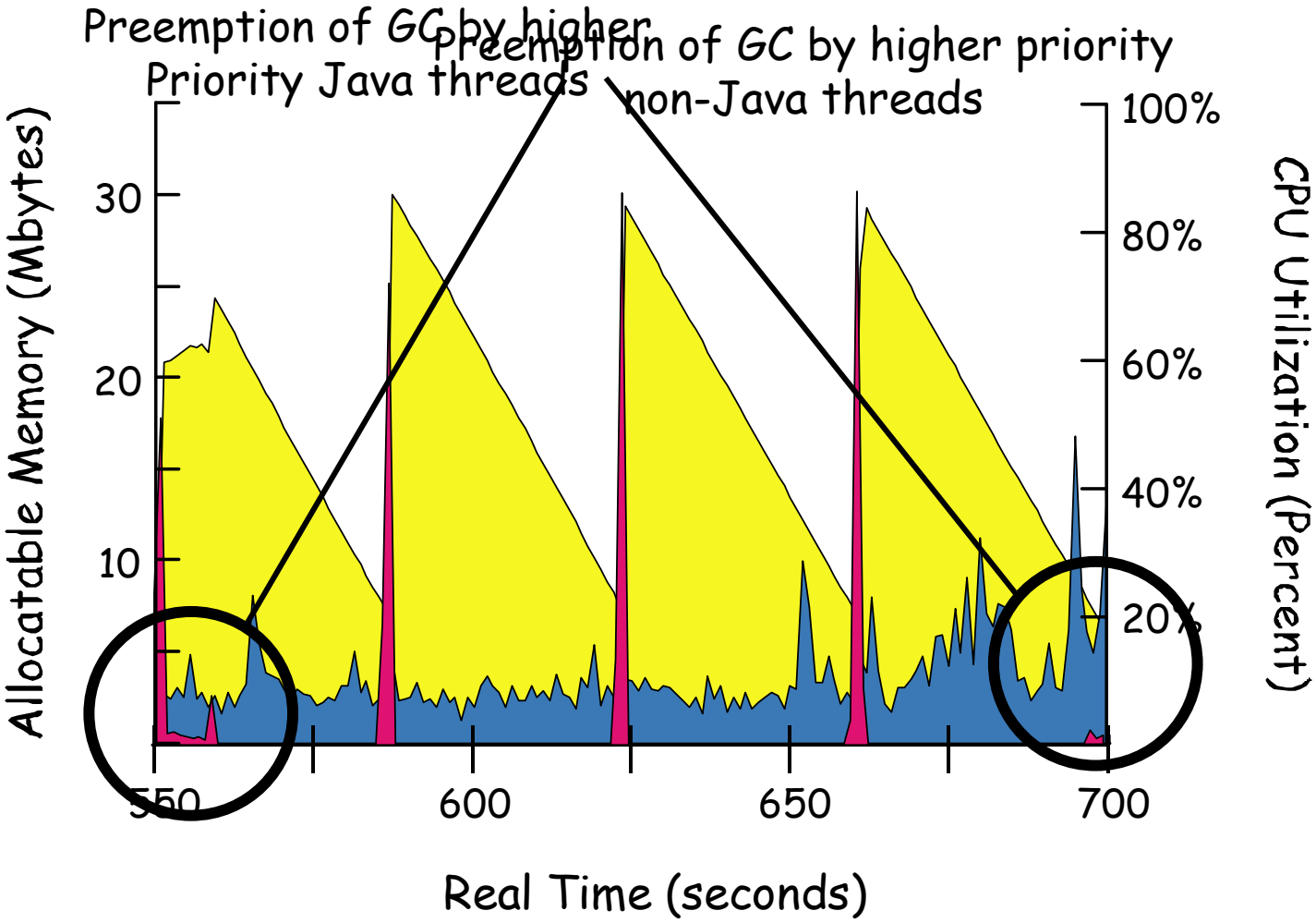
- Heads-up digital assistant supports:
  - Communication with other soldiers and with commanders
  - Map information, including known enemy positions and movements
  - Video streaming from other soldier perspectives
- Implementation uses real-time execution of Java

Aonix

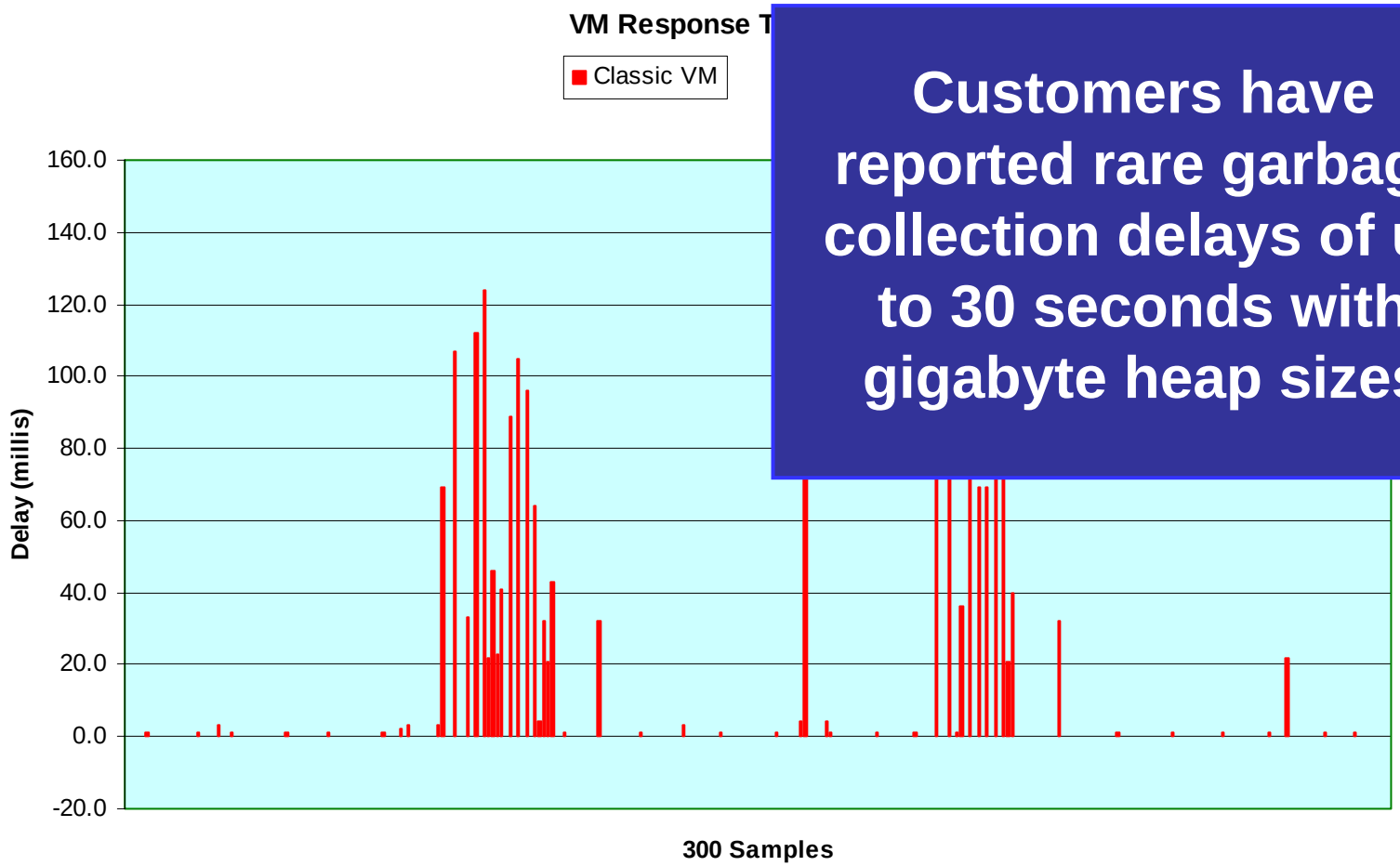
Real-Time GC

- Key attributes of real-time GC:
  - Preemptive
  - Incremental
  - Fully accurate
  - Defragmenting
  - Paced

Trace of Garbage Collection Pacing



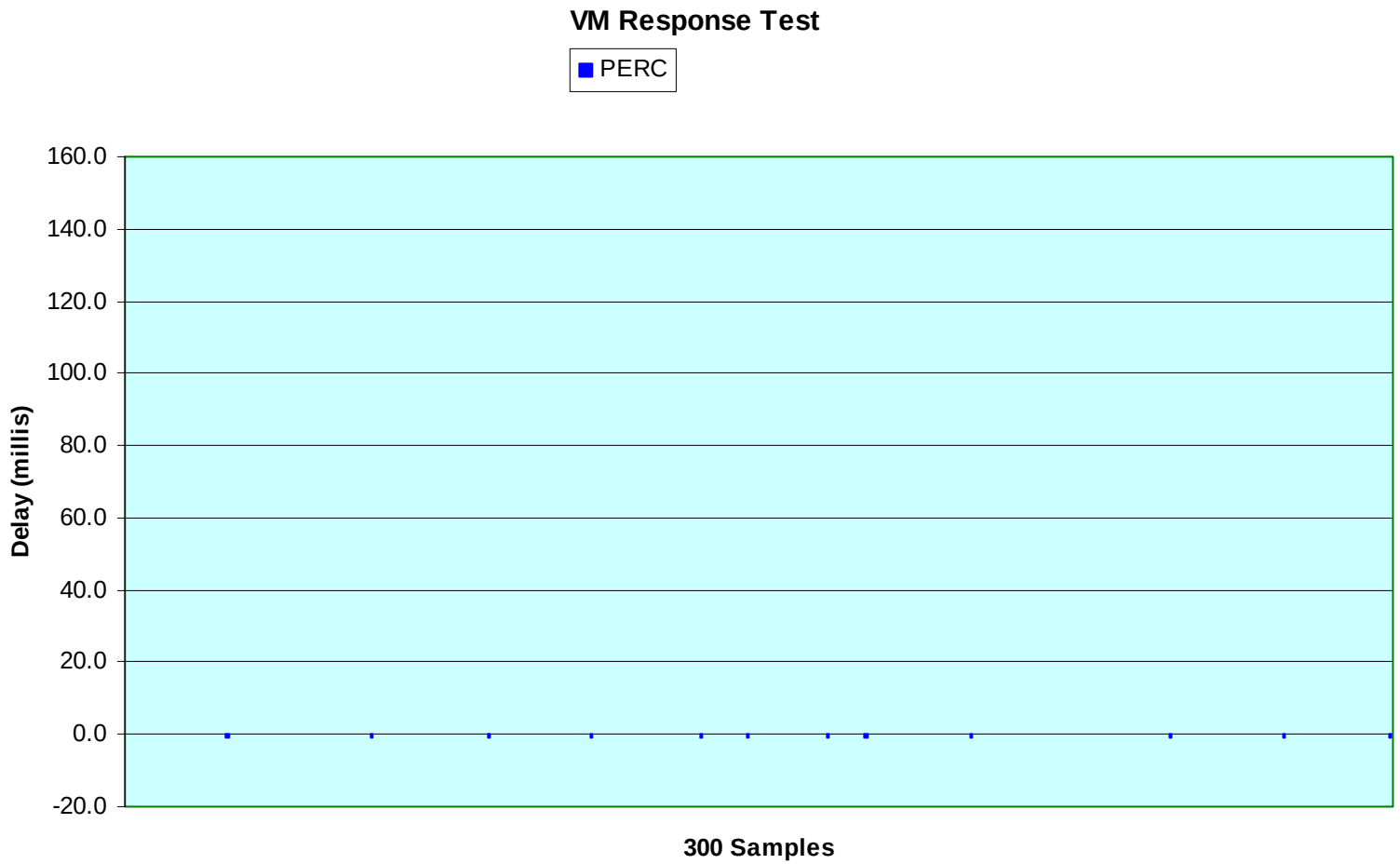
Classic VM running VM Response Test



Customers have reported rare garbage collection delays of up to 30 seconds with gigabyte heap sizes

Aonix

Real-Time VM running VM Response Test



Aonix

Scheduling and Synchronization

- Implement the same portable real-time scheduling and synchronization semantics for all platforms: Integrity, Linux, LynxOS, OSE, VxWorks, Windows, QNX, others
  - Fixed priority, preemptive scheduling under full programmer control – no automatic priority aging
  - Priority inheritance for all Java synchronization
  - All queues maintained in priority order

VM Management Services

- Status inquiries reveal:
  - CPU time consumed by each thread
  - Memory allocation rates
  - Total heap memory usage
  - Length of finalization queue
  - CPU time spent in garbage collection
  - Memory reclaimed by garbage collection
- Management API controls:
  - Priorities for garbage collection and finalization
  - Size of the heap (enlarge and shrink)

Aonix

Summary of Soft Real-Time Java

- Using standard-edition Java APIs with special implementation techniques is the only commercially proven approach:
  - Thousands of successfully deployed systems
  - Millions of hours of proven 5-9's and higher reliability
  - Ease of development, portability, maintainability, scalability
  - Full access to COTS Java components
  - Improved developer productivity (2 fold improvement during development, 5-10 fold improvement during maintenance and integration, in comparison to C++)
- Performance comparable to C++ for the large, complex, dynamic applications to which it is best suited
- GC preemption latency of approximately 100 μs
- Typical periodic execution frequencies of 10-100 Hz

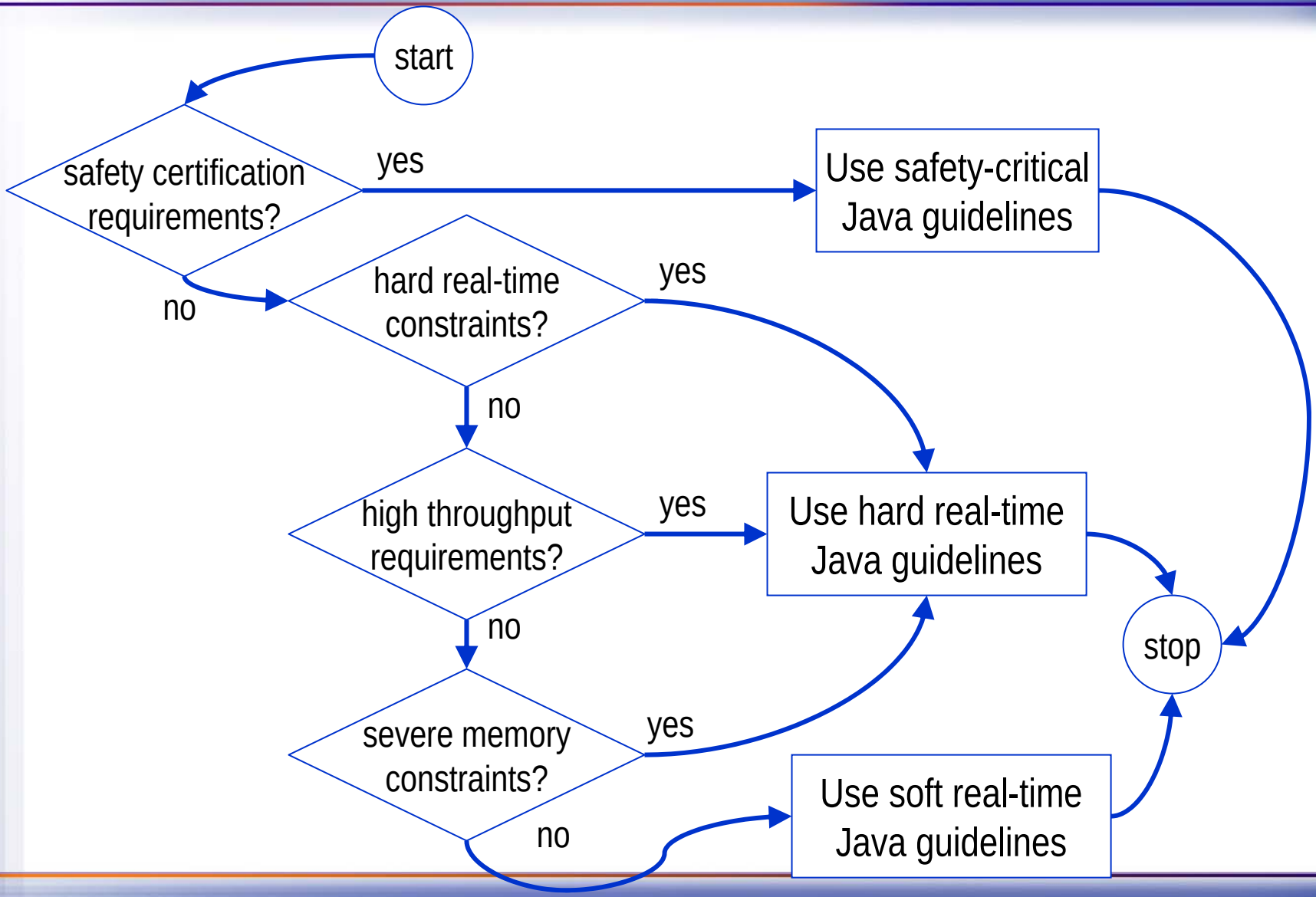
Why NOT use Java for embedded development?

- Top five stated reasons:
  - Run-time speed too slow (35%)
  - Lack of staff Java expertise (28%)
  - Memory requirements are large (26%)
  - Inability to satisfy hard real-time constraints (24%)
  - Inability to perform low-level operations, such as device I/O (18%)

Source: Embedded Market Forecasters (2005) (439 total responses)

Aonix

Horses for Courses



Aonix

Preliminary Hard Real Time Performance

JRTK performance vs.	C	Java HotSpot
Sieve (standalone)	1.20	3.07
Sieve (CaffeineMark)	0.90	1.48
Loop (CaffeineMark)	1.14	3.26
Logic (CaffeineMark)	0.65	1.48
Method (CaffeineMark)	0.71	1.08

Note: most other real-time Java technologies (including PERC and Mackinac) run slower than traditional Java

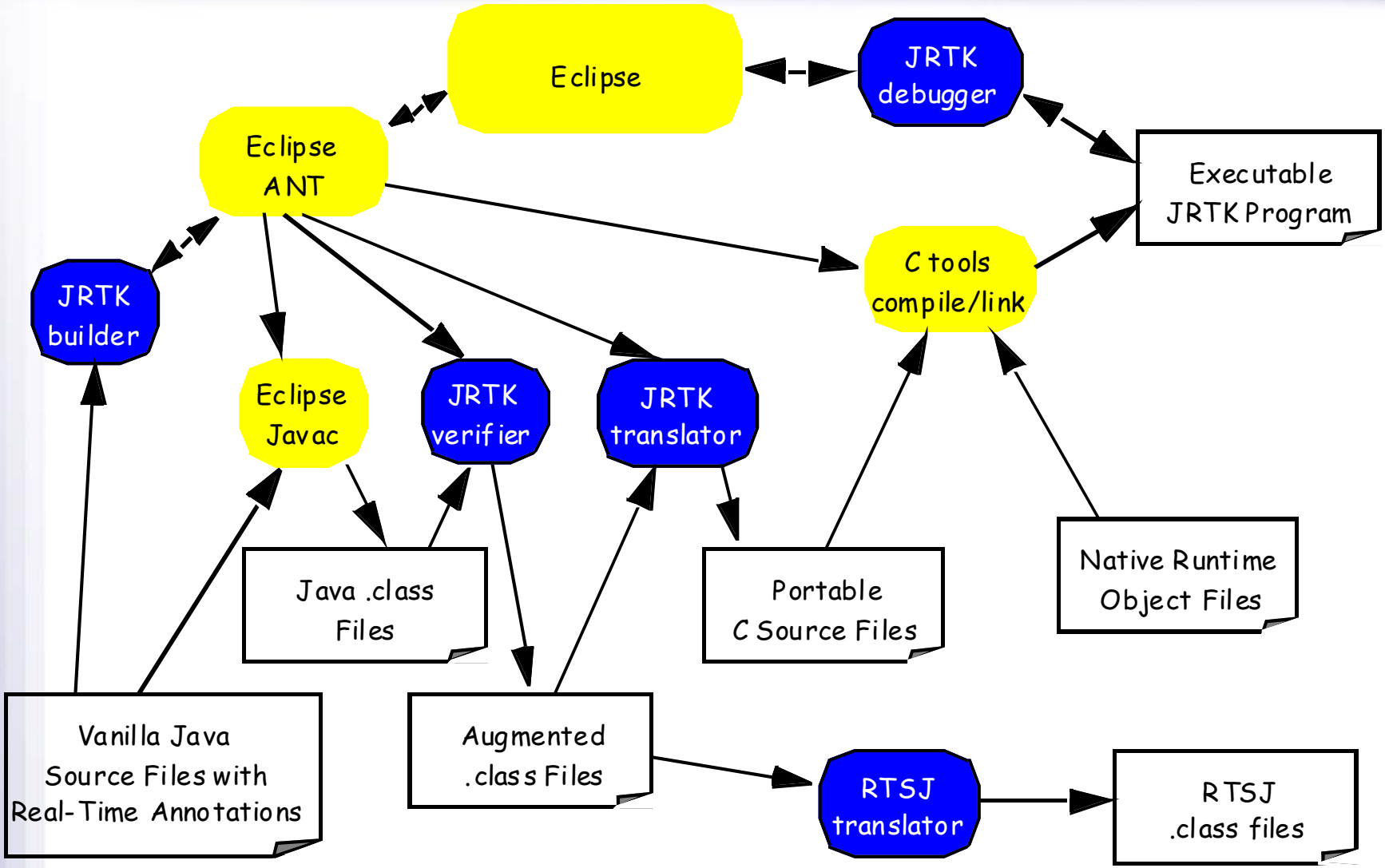
Aonix

A Hard Real-Time Profile

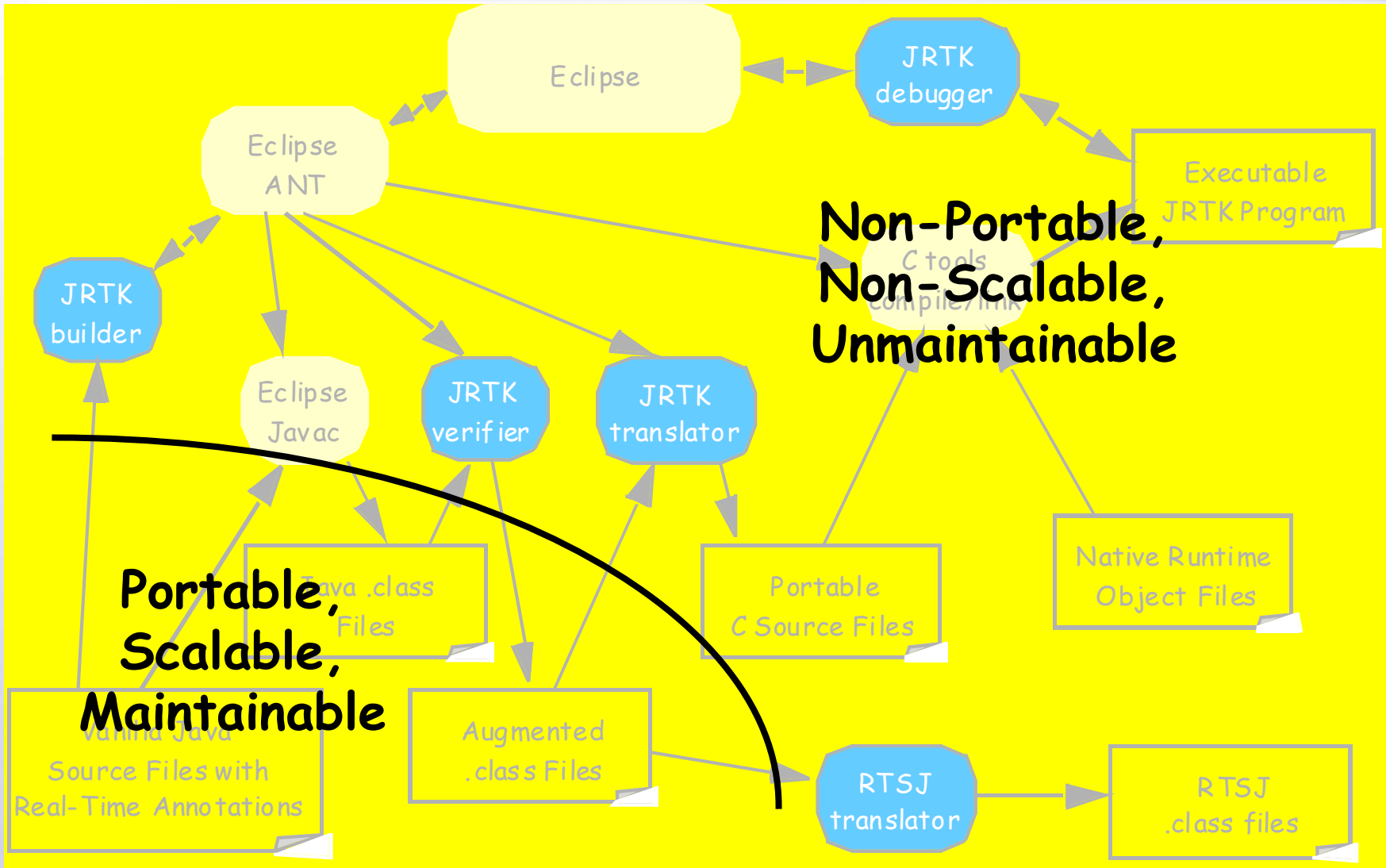
- No garbage collection
- Establish standard subset of RTSJ and J2SE
  - Improved portability
  - Improved efficiency (avoids costly features that are not relevant to hard real-time niche)
- Establish standard library extensions
  - Device I/O and interrupt handling
  - Passive and active real-time clocks
- Java 5.0 standardized meta-data annotations clarify programmer intentions and enable modular composition of components
- Compile-time enforcement of scoped-memory abstractions
  - Fewer programmer and integration errors, improved reliability
  - Higher performance and smaller memory footprint
  - Straightforward and reliable integration of components

Aonix

The Development Environment



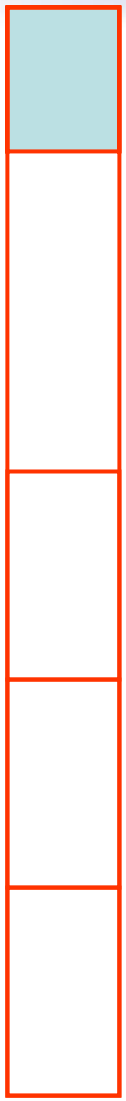
The Development Environment



The Thread Stack Model

Initially, the run-time stack (grows downward) for the main thread represents all non-immortal memory.

The Thread Stack Model



The main thread may spawn additional threads, setting aside part of its own stack to represent the stack memory for the spawned threads.

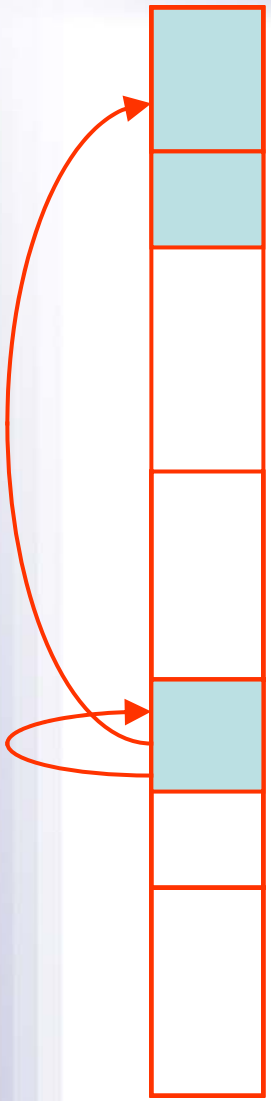
Any of the spawned threads may in turn carve up its stack in order to spawn “grandchildren” threads.

Memory for third spawned thread’s stack

Memory for second spawned thread’s stack

Memory for first spawned thread’s stack

The Thread Stack Model



But objects residing in the parent thread's stack below the point from which the child thread was spawned are not visible to the child thread. And outer-nested objects are not allowed to see objects residing in more inner-nested scopes.

And a child thread may see scoped objects that reside in the parent thread's stack above the point at which the child thread was spawned.

Individual threads populate their run-time stacks as appropriate.

Each thread's scoped objects can see scoped objects allocated In more outer-nested scopes of the same thread.

The Thread Stack Model

The parent thread is required to join with its spawned threads before returning from the context from which it spawned those threads..

After the child threads have joined with the parent thread, their memory is fully reclaimed (and defragmented).

Sample Stack Allocation

```
[1]  package samples;
[2]
[3]  import javax.realtime.util.sc.*;
[4]
[5]  public class Complex {
[6]       public float real, imaginary;
[7]
[8]       public @ScopedThis Complex(float r, float i) {
[9]            real = r;
[10]           imaginary = i;
[11]      }
[12]
[13]      public @CallerAllocatedResult @ScopedPure Complex add(Complex arg)
[14]      {
[15]           float r, i;
[16]           r = this.real + arg.real;
[17]           i = this.imaginary + arg.imaginary;
[18]           return new Complex(r, i);
[19]      }
```

Aonix

Sample Stack Allocation

```
[21]     public @CallerAllocatedResult @ScopedPure
                        Complex multiply(Complex arg)
[22]     {
[23]         float r, i;
[24]
[25]         r = this.real * arg.real - this.imaginary * arg.imaginary;
[26]         i = this.real * arg.imaginary + this.imaginary * arg.real;
[27]         return new Complex(r, i);
[28]     }
[29] }
```

Aonix

@StaticAnalyzable

- Programmers may annotate any method or interface to declare that important attributes of its behavior must be verifiable
  - Execution time
  - Stack growth
  - Immortal memory allocation
- Special byte-code verifier assures that programmer follows conventions required to enable static verification
- This annotation is inherited to any overriding methods in subclasses

Aonix

Annotated Program (1/3)

```
[1]  import javax.realtime.util.sc.StaticAnalyzable;
[2]  import javax.realtime.util.sc.Stackable;
[3]  import javax.realtime.util.sc.StaticLimit;
[4]
[5]  public class BubbleSort {
[6]       public enum AnalysisModes
[7]       { UNBOUNDED,              // can't analyze the most general case
[8]         SMALL,                  // array smaller than 16 elements
[9]         BIG,                    // array up to 64 elements
[10]        SMALL_SORTED,           // small array, one element out of order
[11]        BIG_SORTED              // big array, one element out of order
[12]      }
[13]      @StaticAnalyzable(
[14]           enforce_analysis = { false, true, true, true, true },
[15]           modes = AnalysisModes.class)
[15]      public void sort(@Scoped int a[]) {
[16]           int i, j, k, t;
[17]           int len = a.length;
[18]           boolean sorted = false;
```

Aonix

Annotated Program (2/3)

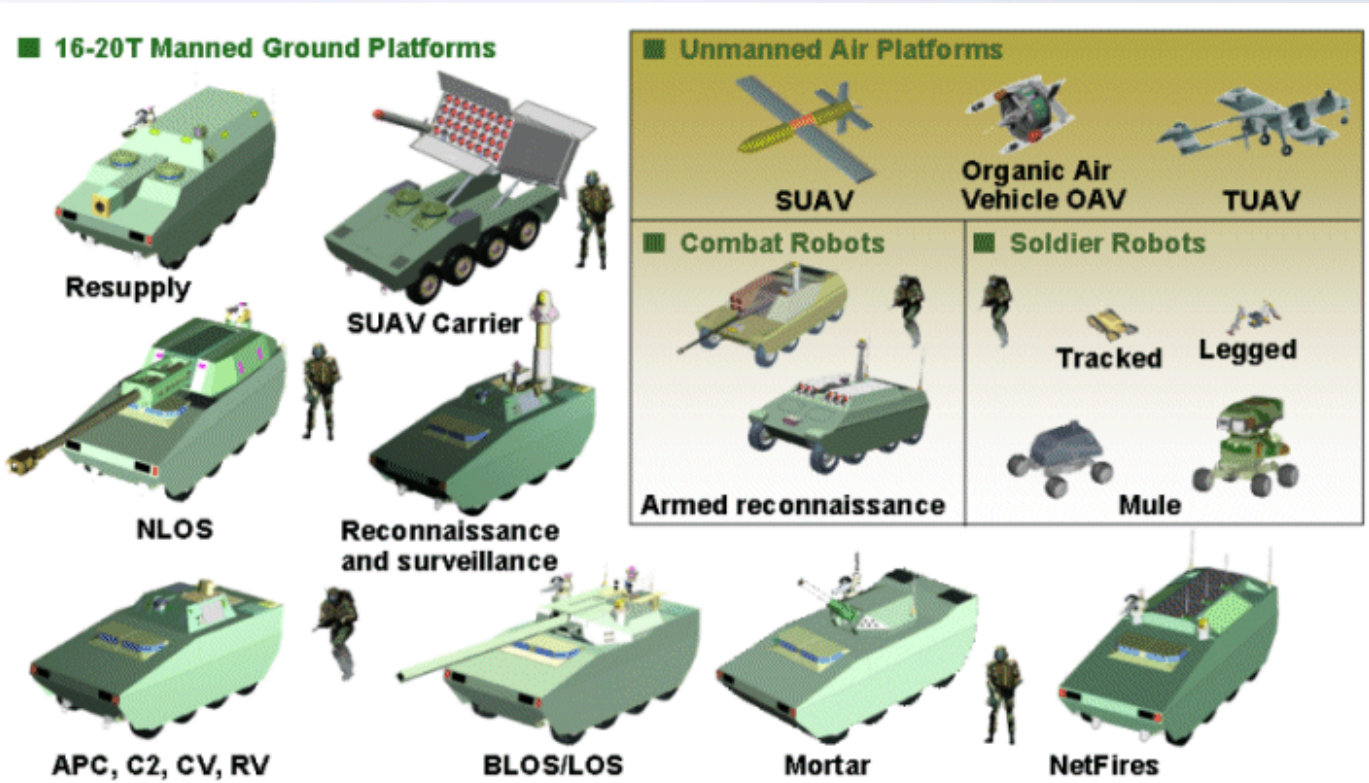
```
[19] for (i = 0, k = len; !sorted && (i < len); i++) {
[20]     assert StaticLimit.IterationBound(AnalysisModes.SMALL, 16);
[21]     assert StaticLimit.IterationBound(AnalysisModes.BIG, 64);
[22]     assert StaticLimit.IterationBound(
                            AnalysisModes.SMALL_SORTED, 2);
[23]     assert StaticLimit.IterationBound(
                            AnalysisModes.BIG_SORTED, 2);
[24]     k--;
[25]     sorted = true;// assume array is sorted
[26]     for (j = 0; j < k; j++) {
             // Missing assertions
[31]         if (a[i] < a[j]) {
                 // Missing assertions
[36]             t = a[i]; a[i] = a[j]; a[j] = t; sorted = false;
[37]         }
[38]     }
[39] }
```

Annotated Program (3/3)

```
[26]for (j = 0; j < k; j++) {
[27]     assert StaticLimit.NestedIterationBound(SMALL, 1, 120);
[28]     assert StaticLimit.NestedIterationBound(BIG, 1, 2016);
[29]     assert StaticLimit.NestedIterationBound(SMALL_SORTED,1, 29);
[30]     assert StaticLimit.NestedIterationBound(BIG_SORTED, 1, 125);
[31]     if (a[i] < a[j]) {
[32]          assert StaticLimit.NestedIterationBound(SMALL_SORTED,
[33]                                                  1, 15);
[34]          assert StaticLimit.NestedIterationBound(BIG_SORTED,
[35]                                                  1, 63);
[36]          t = a[i]; a[i] = a[j]; a[j] = t; sorted = false;
[37]     }
[38] }
```

Aonix

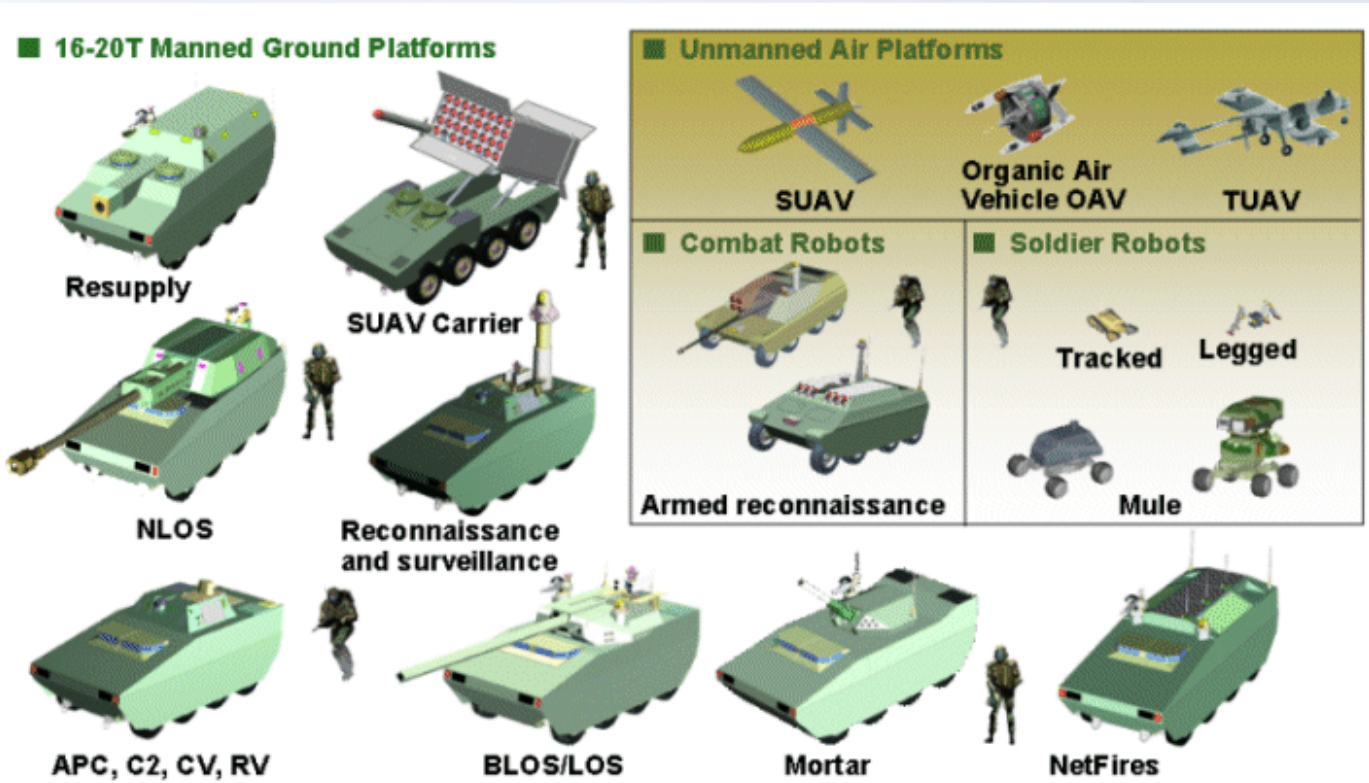
Future Combat Systems SOSCOE



- System of Systems Common Operating Environment
  - Supports multiple configurations, and
  - Programming in Ada, C, C++, and Java

Aonix

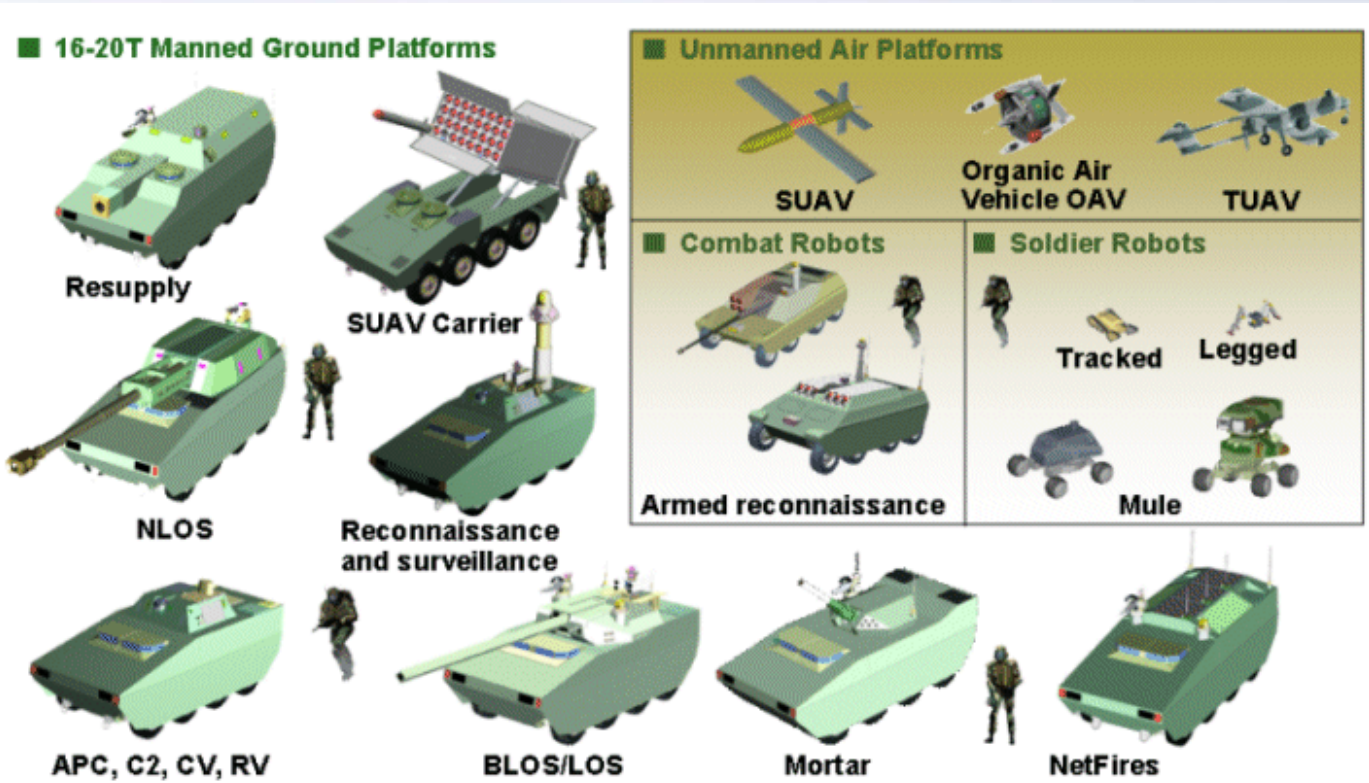
Future Combat Systems SOSCOE



- System of Systems Common Operating Environment
  - All configurations support modular composition of components, portability, and integration of COTS components. Multiple configurations must coexist on same processor

Aonix

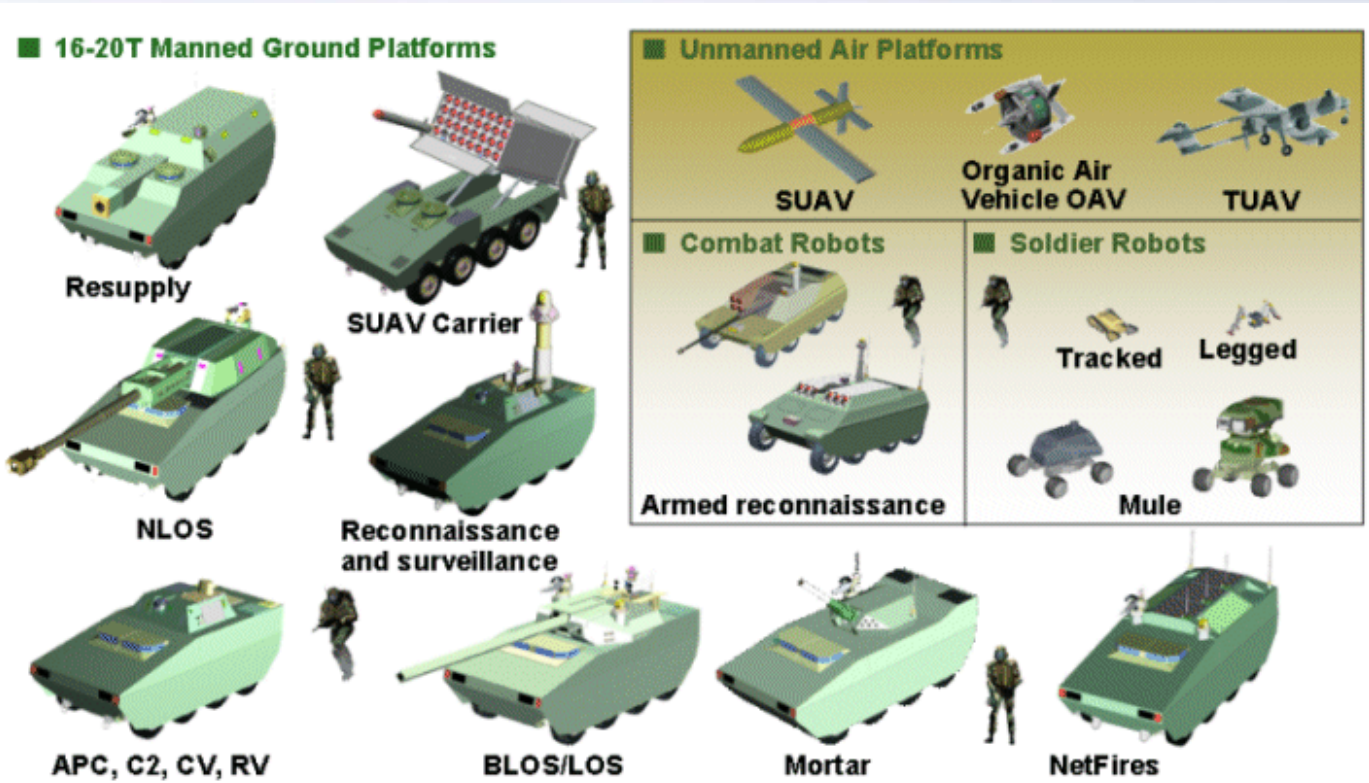
Future Combat Systems SOSCOE



- System of Systems Common Operating Environment
  - Soft real-time configuration supports 20 Hz activities on Linux and/or Windows

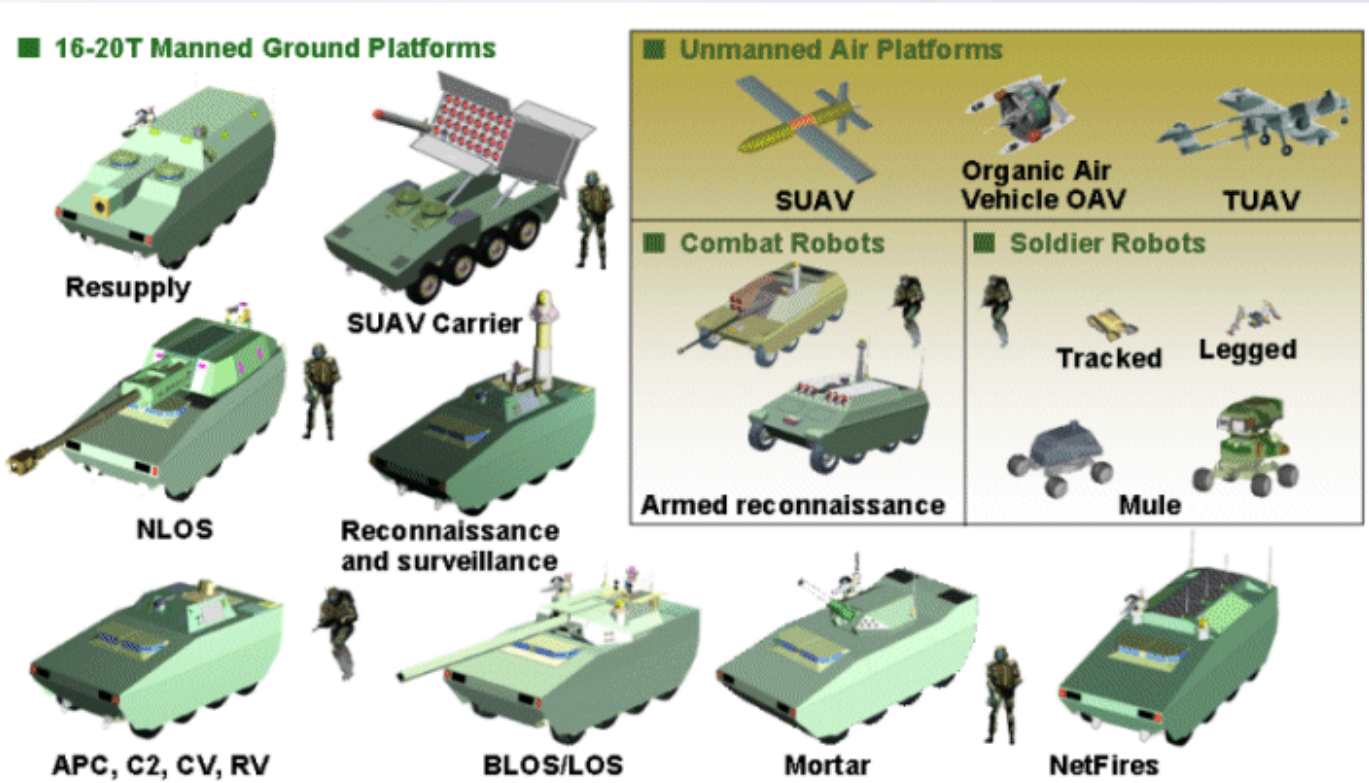
Aonix

Future Combat Systems SOSCOE



- System of Systems Common Operating Environment
  - Real-time configuration supports 200+ Hz activities on Integrity, LynxOS, VxWorks

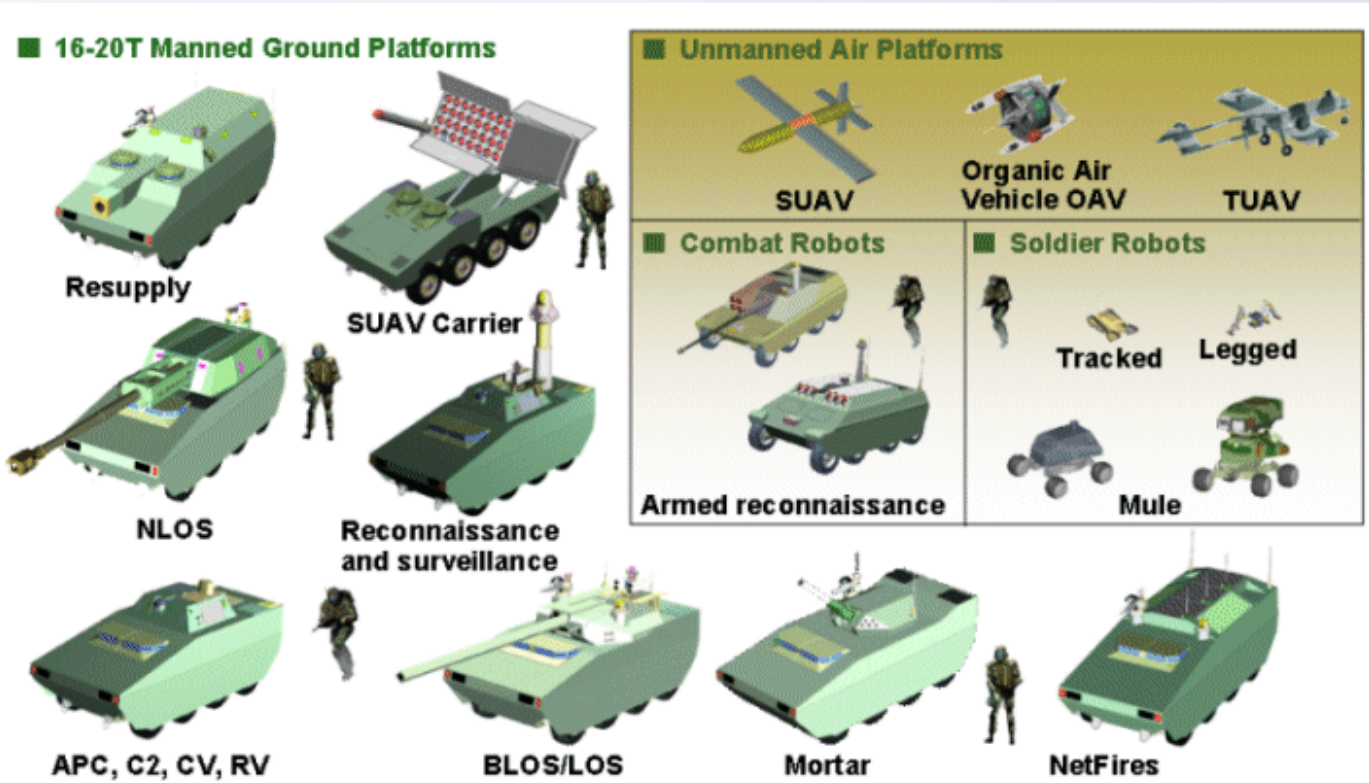
Future Combat Systems SOSCOE



- System of Systems Common Operating Environment
  - Mobile configuration runs on COTS PDA devices

Aonix

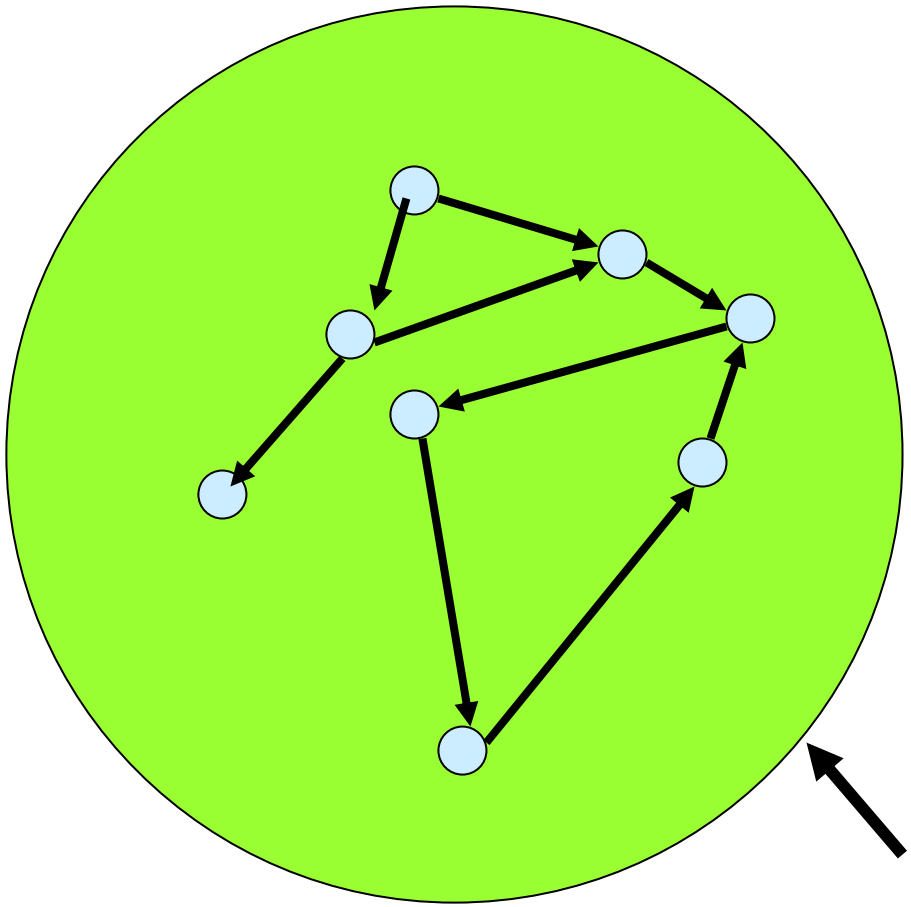
Future Combat Systems SOSCOE



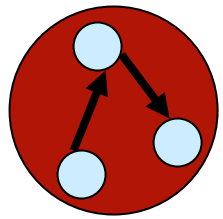
- System of Systems Common Operating Environment
  - Micro configuration runs in severely constrained environments, includes power management capabilities, and runs with very limited or no RTOS support

Aonix

Cooperating HRT Components

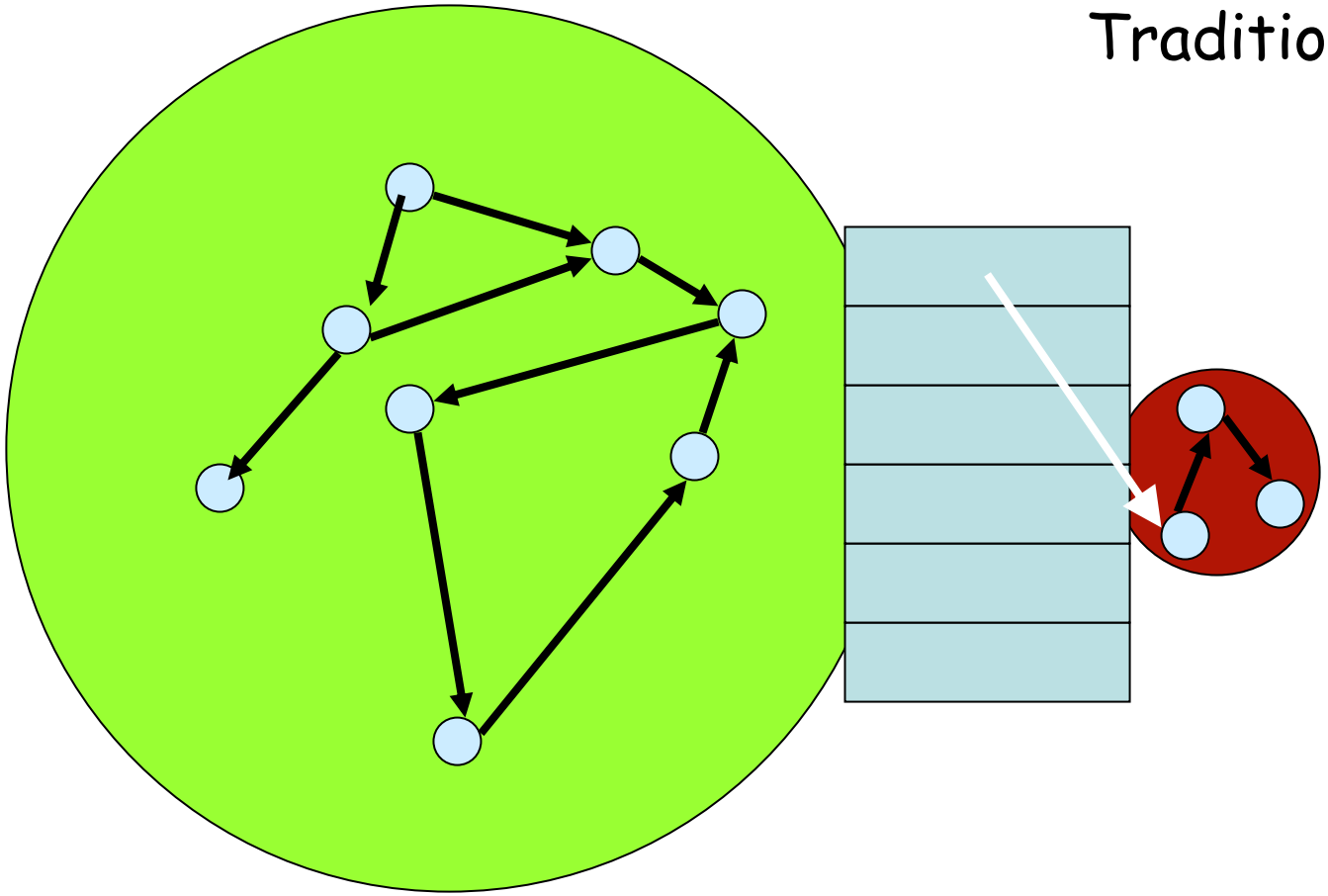


PERC Pico Hard Real-Time Execution Engine



PERC Ultra Virtual Machine

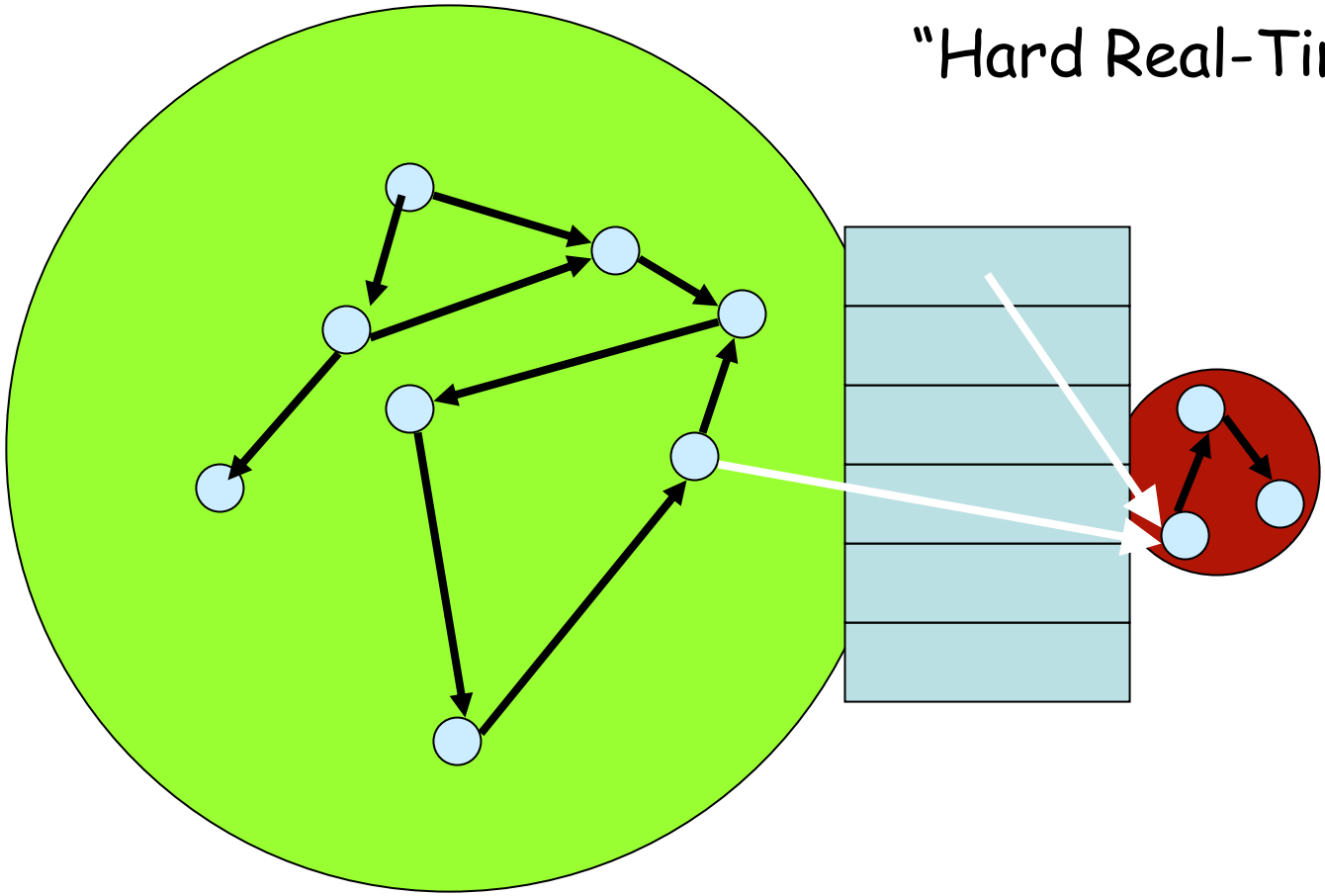
TraditionalJava Registry



TraditionalJava.publish("deviceXYZ", x)

Aonix

TraditionalJava Registry



HardRealTimeDomain.lookup("deviceXYZ")

Resource Material

- *Guidelines for Scalable Java Development of Real-Time Systems*, Kelvin Nilsen, Ph.D., CTO, Aonix
  - Based on standardization discussions within the Open Group Real-Time and Embedded Forum
  - A foundation for the European Space Agency's guidelines for Java development of real-time software
- See http://research.aonix.com/jsc/index.html and http://research.aonix.com/jsc/rtjava.guidelines.3-28-06.pdf
- For RTSJ background, see https://rtsj.dev.java.net/

Summary

- Real-Time Java offers significant productivity benefits during development and maintenance
  - More projects complete on schedule, within budget
  - More features can be added under same budget
  - More time available for quality assurance and performance tuning
  - Resulting systems are more reliable and more flexible
- Exploiting the benefits of real-time Java requires careful selection of the most appropriate tools for each specific job