

Meeting Your Most Critical Requirements



Applying the High-Level Benefits of Java to
Real-Time Development Challenges

Kelvin Nilsen, Ph.D., CTO

- What are real-time development challenges?
- What are the high-level benefits of Java?
- What does Java offer the real-time community?

Characterizing real-time software

- Very informally: “Real-time software has timing constraints that are distinct from the general desire to offer high throughput and/or user responsiveness.”
- Usually, real-time constraints are associated with:
 - Monitoring and control of the physical environment, or
 - Time-constrained communication and networking protocols

Real-Time Discipline

- System requirements include explicit mention of timing constraints
- As components are developed, the CPU time and memory resource needs of components are analyzed
- Contention for other shared resources (e.g. network bandwidth, locked data base records) is also studied
- When components are assembled, system integrators analyze system schedulability

Schedulability Analysis

- There are many different kinds of real-time schedulers and each requires different analyses of schedulability
- Example:
 - Fixed-priority scheduling uses rate monotonic analysis
 - Threads with tighter time constraints get higher priority – priority does not represent importance
 - If cumulative workload is less than 69%, all tasks will satisfy their timing constraints

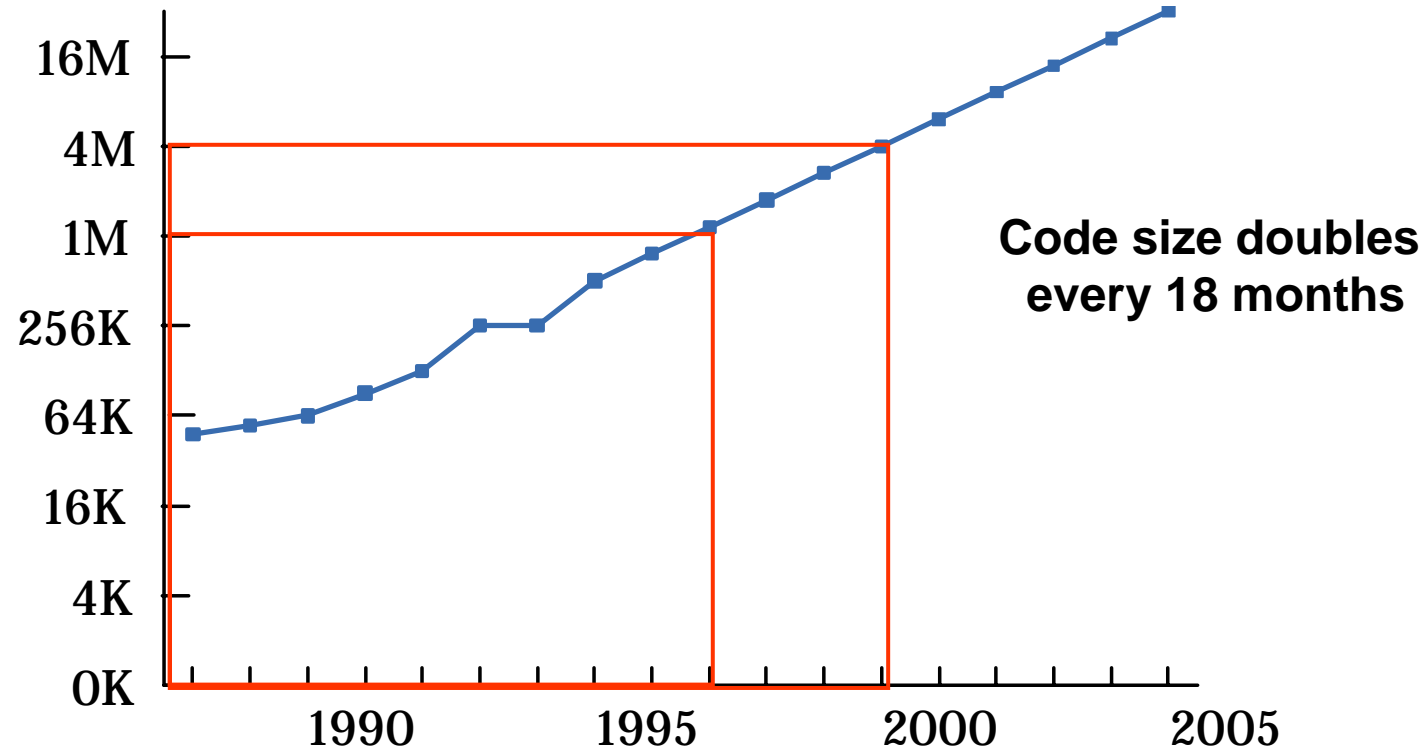
Two kinds of real-time

- Hard real-time:
 - Use analytical techniques to determine absolute bounds on worst-case resource needs. Use deterministic algorithms to enforce resource budgets.
- Soft real-time:
 - Use empirical techniques to determine estimates of resource needs. Use heuristics to enforce resource budgets.
- Hard real-time is hard. Soft real-time may be harder!
 - Soft real-time usually deals with greater complexity, larger program size, more dynamic behavior, and less precise characterization of timing constraints

Interplay of Embedded Constraints

- Reliability
- High Availability
- Environmental (radiation, vibration, G forces)
- Physical size
- Battery power consumption
- Memory
- Throughput
- Safety certification

ROM in Consumer Infotainment Device



Source: Philips Semiconductor data for high-end television receiver

Another Data Point

- GM CTO Anthony Scott (Oct. 2004):
 - “More than one-third of the cost of GM's automobiles now involves software and electronic components”
 - “Cars had approximately 1 million lines of software code in 1990, but this number will jump to 100 million by 2010”
 - Translation: code size doubles every 3 years



Breaking the Tradition

- Clearly, exponential growth of developer staffs is an unacceptable solution
 - Improve individual developer productivity
 - Reuse existing code with each new product release
 - License commodity software components from third parties
 - Reduce effort and costs of integrating independently developed software and hardware modules



Examples of Java Adoption

- Thousands of soft real-time systems have been successfully deployed with proven 5-9's and higher availability
- Hard real-time and safety-critical Java are just becoming available and have not yet been demonstrated in fielded systems, but several projects are actively investigating



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

Asset Tracking - Last Known Locations Map - Microsoft Internet Explorer

File Edit View Favorites Tools Help

XATANET™ Home | My Info | Contact | Help | Logout

System > XATA Corporation > Anns Fleet Welcome, sysadmin sysadmin

Pick View

- Administration
- Performance
- Compliance
- Dispatch
- Safety
- Maintenance
- Asset Management

Last Known Locations Map

Key	Vehicle ID	Location	Date/Time
1	AnnsDevEnv	13.4 Mi N of Clifton Forge, VA	6/27/2002 5:28:36 PM CDT
2	AnnsGPS	11.3 Mi N of Pierre, SD	8/2/2002 10:21:54 AM CDT
3	AnnsMobi	1.5 Mi N of Burnsville, MN	7/10/2002 10:56:00 AM CDT
4	annsmvpc	7.0 Mi N of Hunter, NY	8/13/2002 8:59:00 AM CDT
5	AnnsNew	3.4 Mi E of Marengo, IA	9/3/2002 8:08:17 AM CDT
6	AnnsSSRF	22.6 Mi NE of Manderson, WY	7/3/2002 11:21:08 AM CDT
7	focus	No location data	-
8	new4	No location data	-
9	newmbx	15.1 Mi SE of Lead, SD	7/29/2002 8:02:59 AM CDT
10	hoasset	No location data	-

Previous Page 1 of 2 Next

POWERED BY XATA

Copyright (©) 2002, XATA Corporation 1.800.745.9282 Home My Info Contact Help Logout

Done Local intranet

XATA



Calix C7 Broadband Loop Carrier

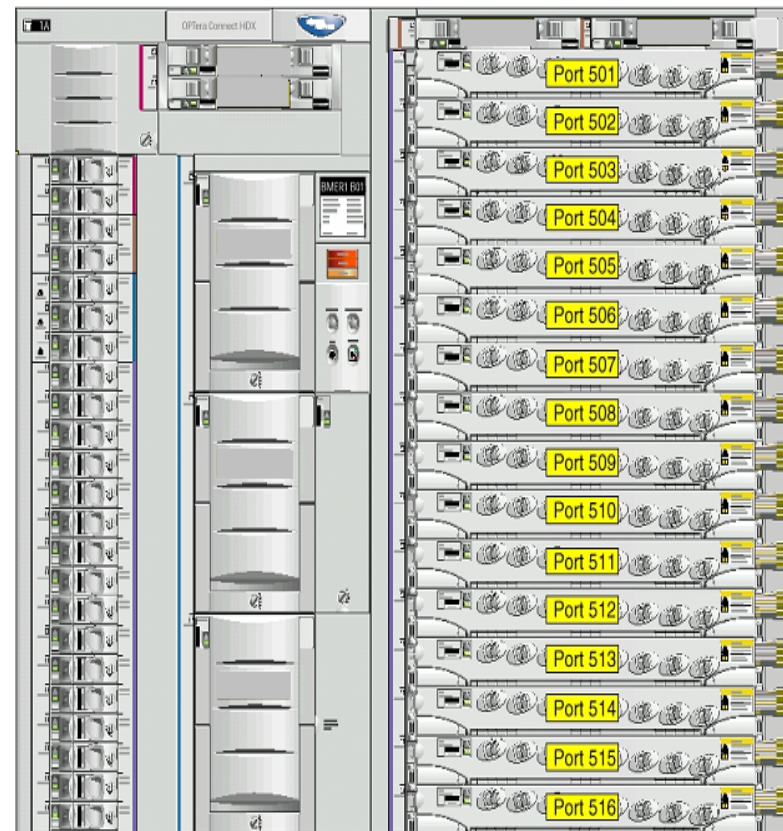


- Replaced C management plane with Java: better code reuse (5x), improved developer productivity (2x), fewer bugs, and more flexible architecture
- As of 1st quarter 2005, had shipped 6,500 units to 190 service providers, supporting 1.2 million communication “ports”
- Industry segment market leader,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

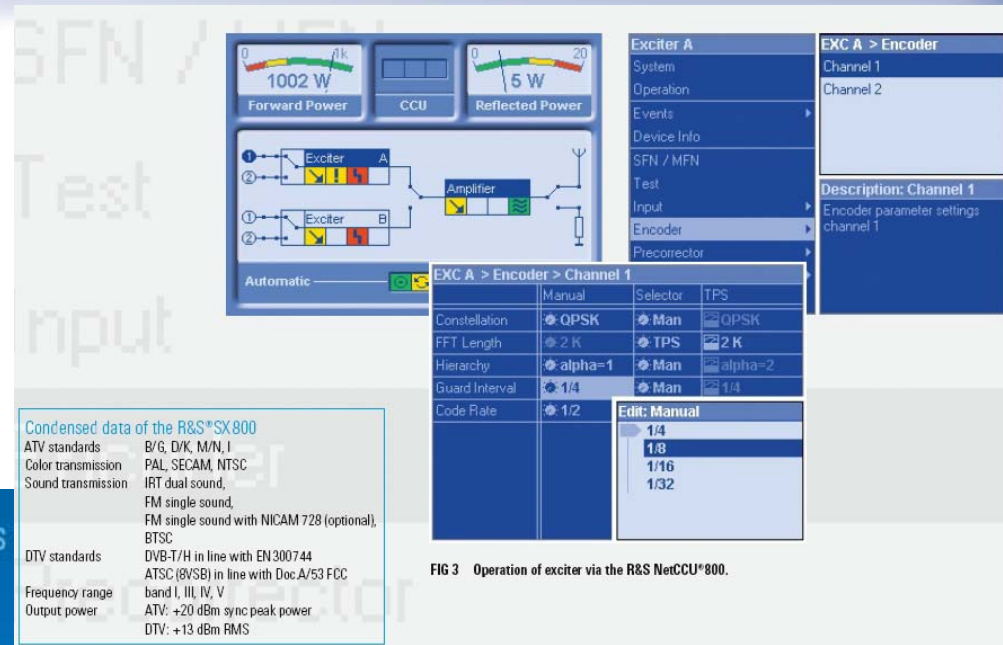


PERC®

NORTEL
NETWORKS

TV Broadcasting by Rohde & Schwarz

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





Boeing J-UCAS X-45C Unmanned Aircraft



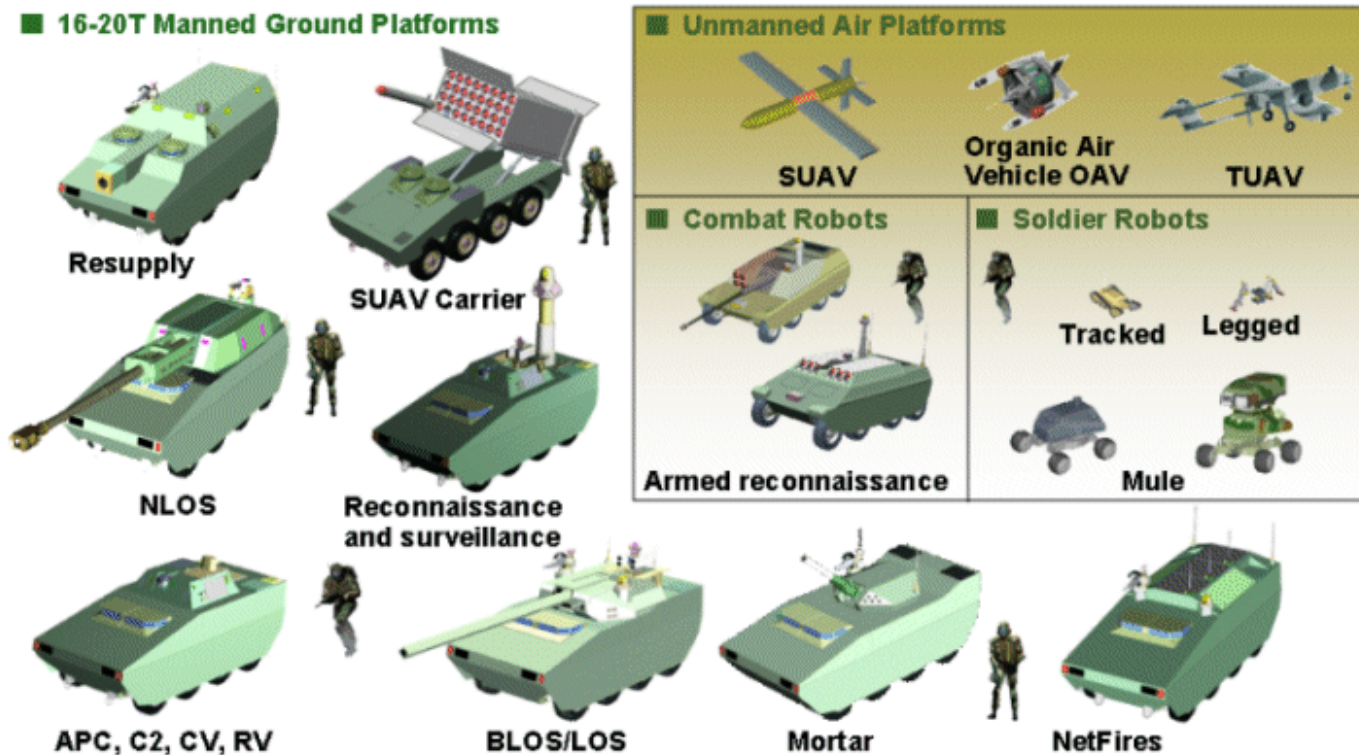
- Joint software development effort between Boeing and BAE
- Mission planning software implemented in Java
- Mission plan is continually updated to account for weather, fuel levels, weapons deployment status, enemy activities, and evolving objectives
- BAE characterization: “solve the traveling salesman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

French Military FELIN Project by Sagem



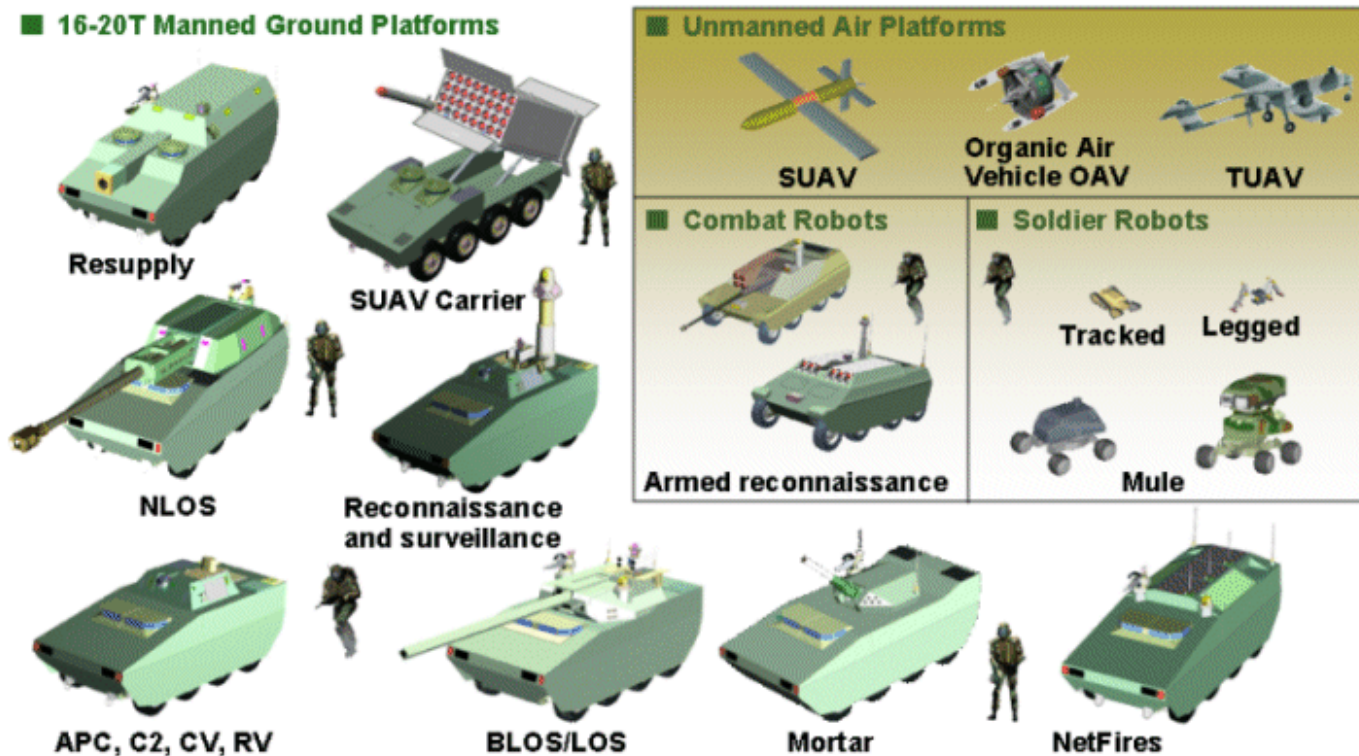
- Heads-up digital assistant supports:
 - Communication with other soldiers and with commanders
 - Map information, including known enemy positions and movements
 - Video streaming from other soldier perspectives
- Implementation uses real-time execution of Java

Future Combat Systems SOSCOE



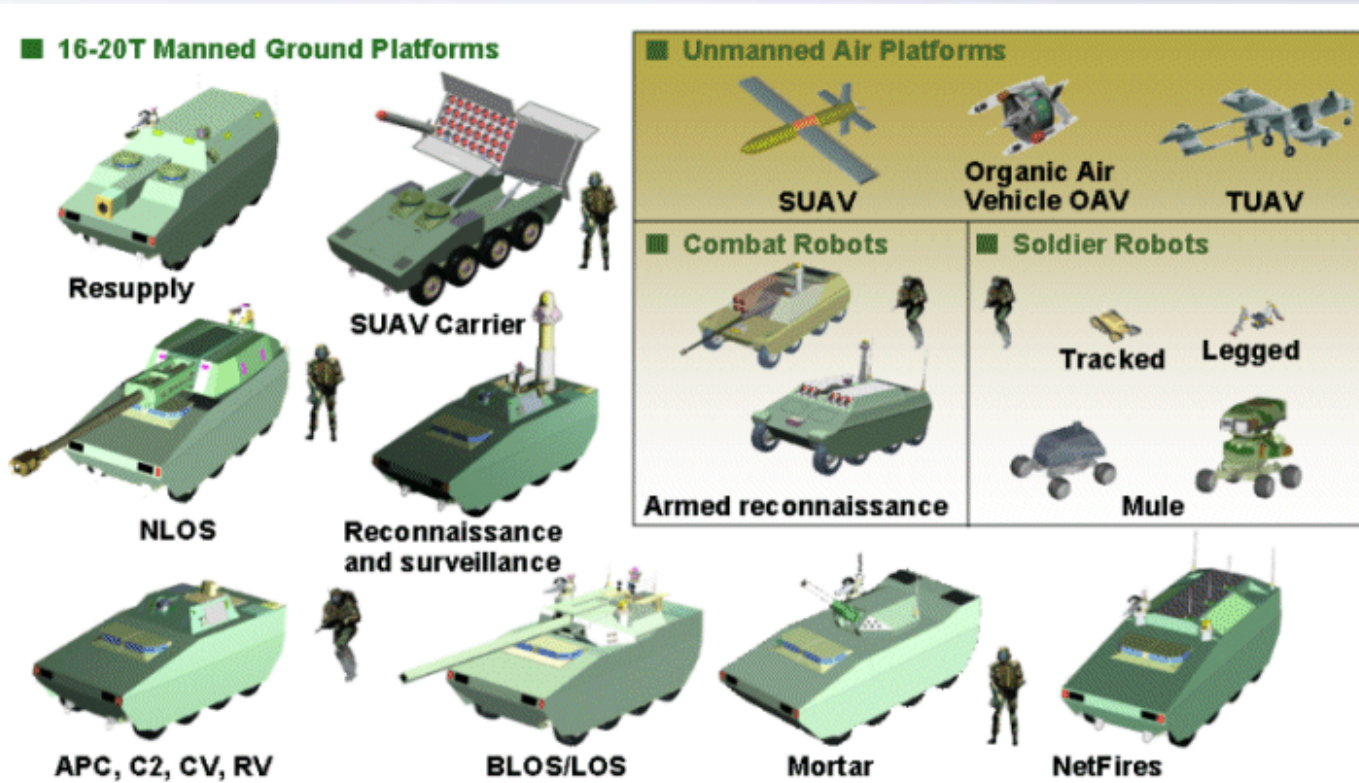
- System of Systems Common Operating Environment
 - Supports multiple configurations, and
 - Programming in Ada, C, C++, and Java

Future Combat Systems SOSCOE



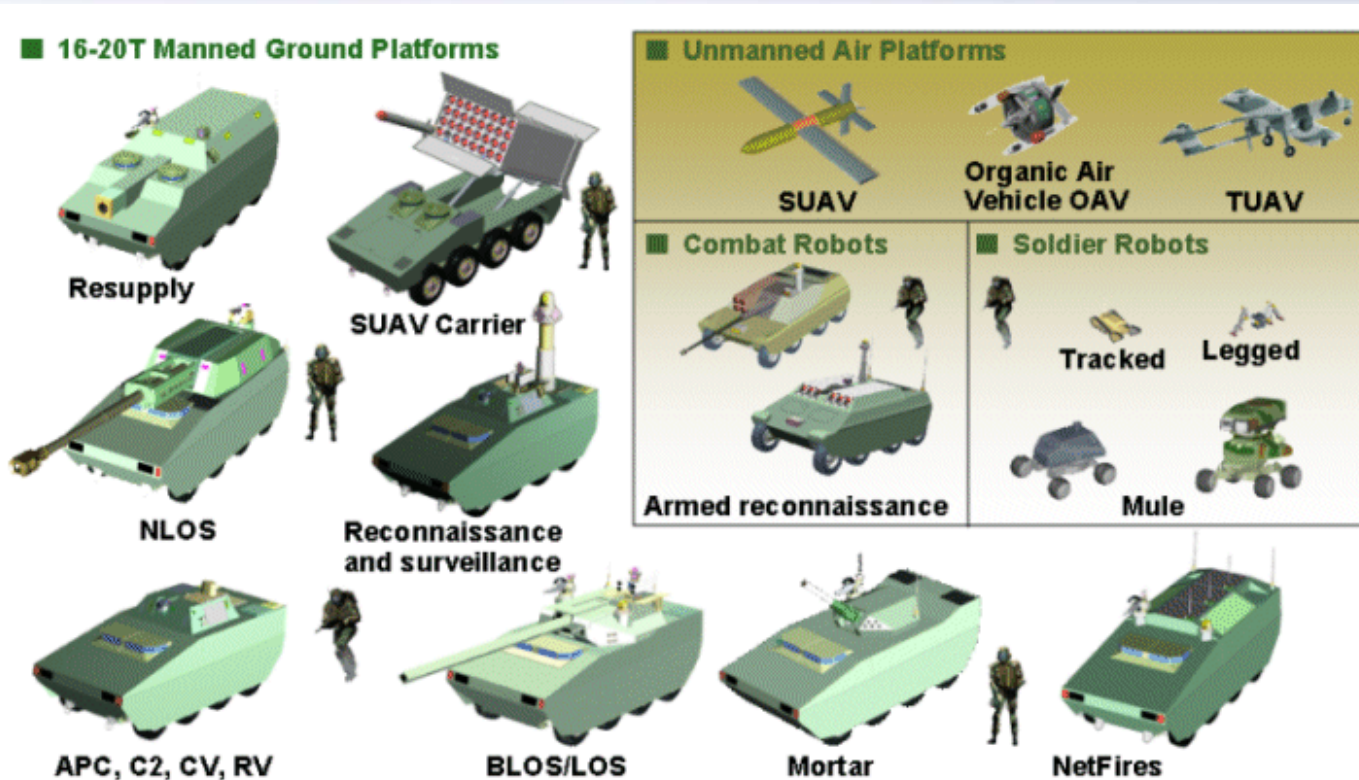
- System of Systems Common Operating Environment
 - All configurations support modular composition of components, portability, and integration of COTS components. Multiple configurations must coexist on same processor

Future Combat Systems SOSCOE



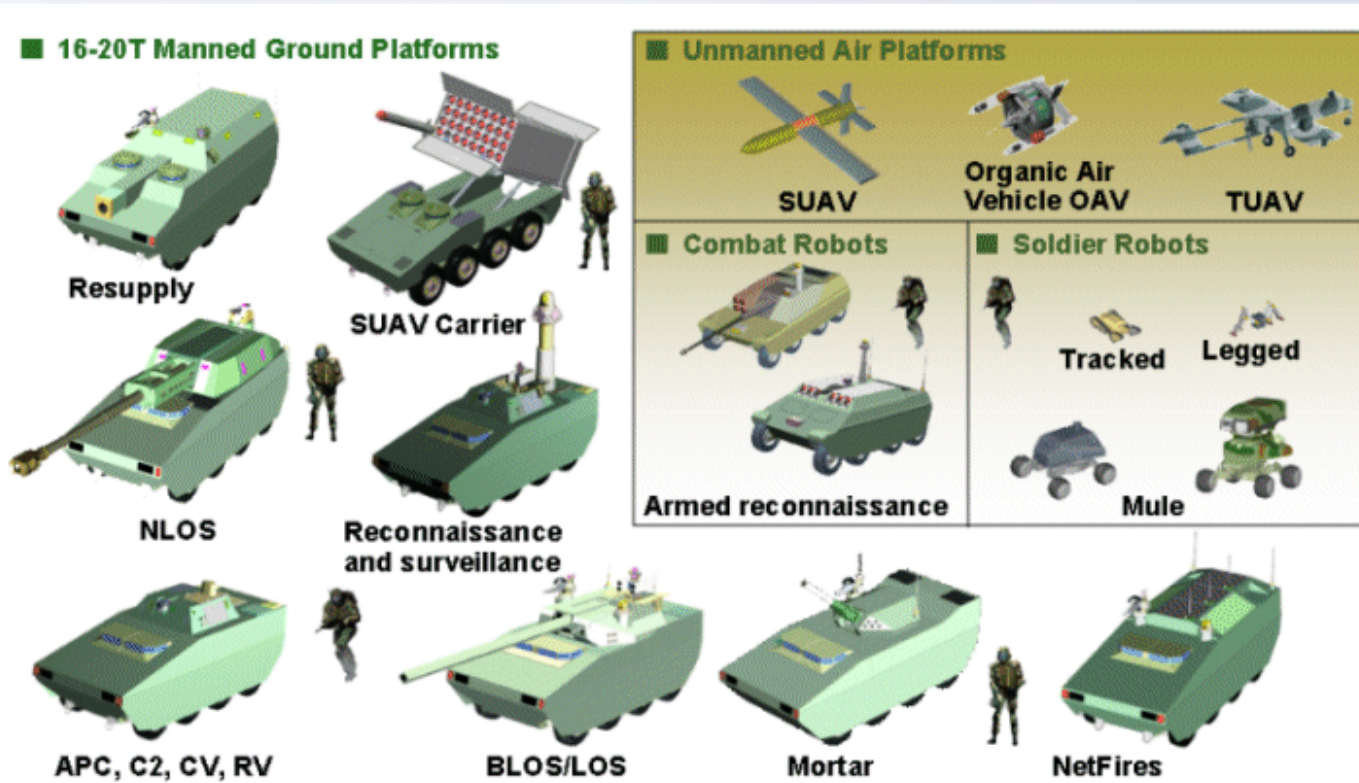
- System of Systems Common Operating Environment
 - Soft real-time configuration supports 20 Hz activities on Linux and/or Windows

Future Combat Systems SOSCOE



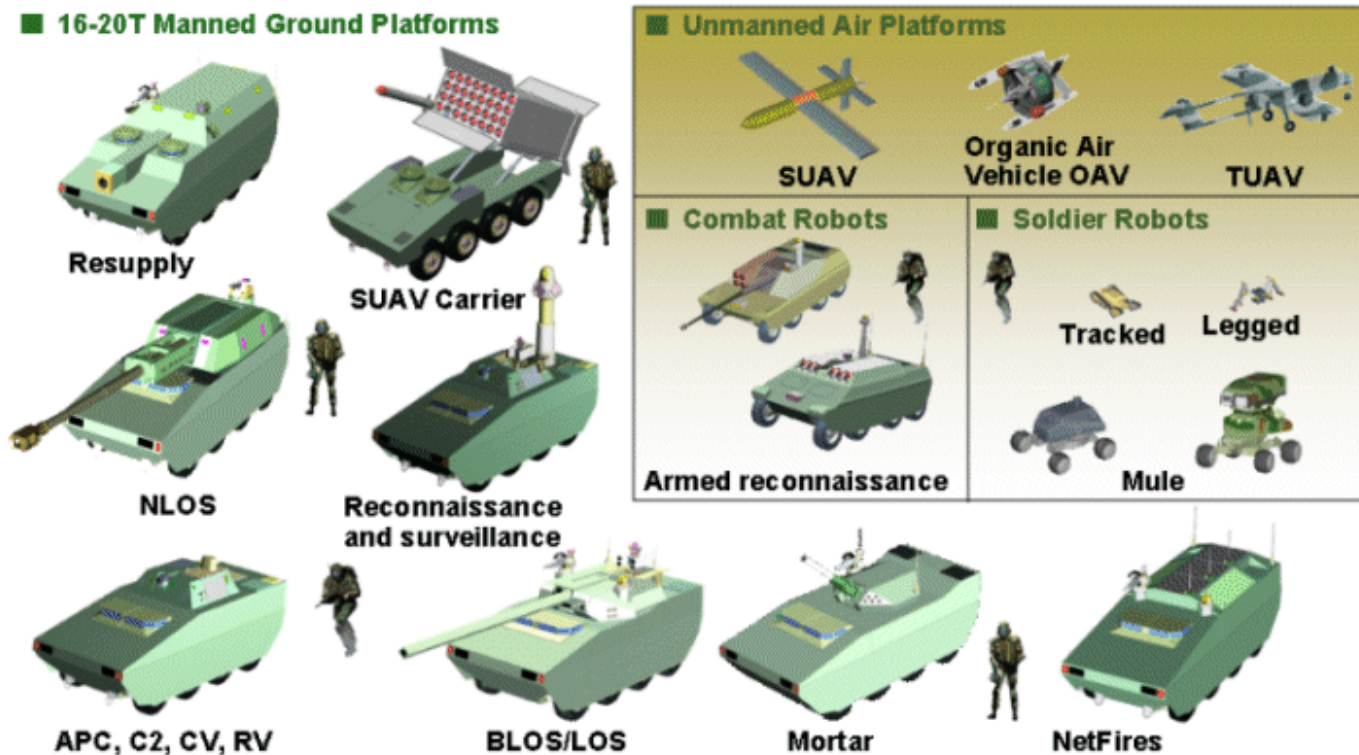
- System of Systems Common Operating Environment
 - Real-time configuration supports 200+ Hz activities on Integrity, LynxOS, VxWorks

Future Combat Systems SOSCOE



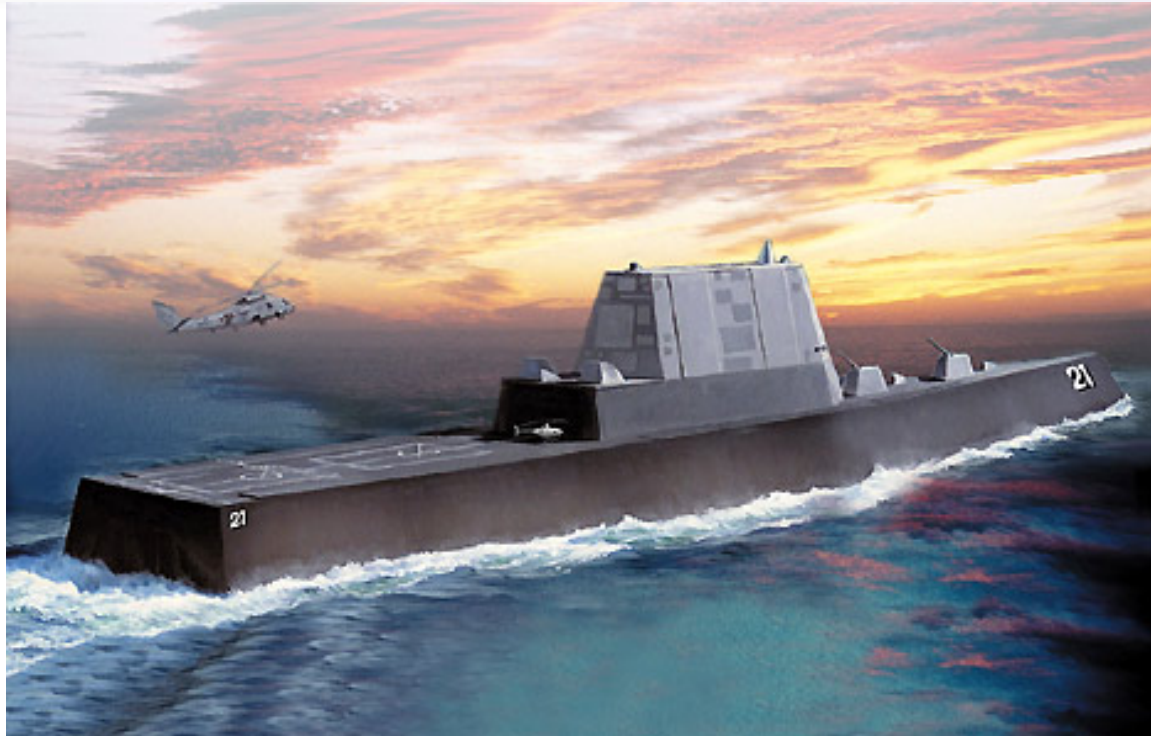
- System of Systems Common Operating Environment
 - Mobile configuration runs on COTS PDA devices

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

DDG-1000 (aka DDX)



- DDG-1000:
 - Anticipates extensive use of “real-time Java” for communications, command, and control
 - Is experimentally evaluating use of real-time Java for hard real-time and safety-critical purposes



Why use 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)



Findings in the Embedded Industry

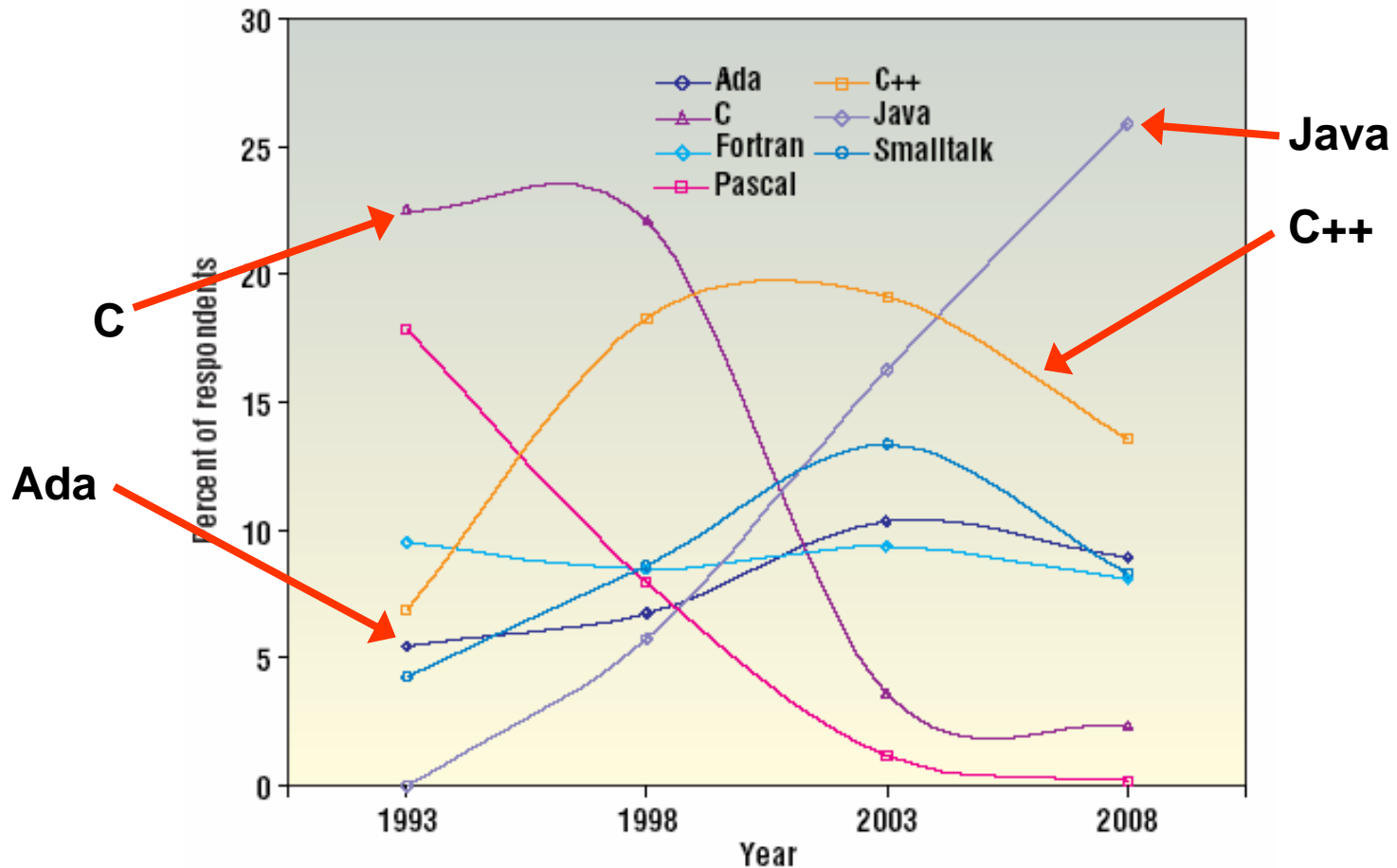
- Embedded real-time developers who move from C++ to Java are roughly twice as productive during development of new functionality.
- Maintenance and integration efforts are five to ten times less costly using Java than C++!
 - This is largely because traditional embedded real-time development is so miserably non-portable.



Representative experience

- Intel project: developed fault-tolerant distributed Java demonstration 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 virtual machine product increased developer productivity by two fold (including the time required to learn Java!), and 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!

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



Predictive Model Correlations

- Extrinsic Factors: Support from
 - Institutions (e.g. university curricula)
 - Industry (corporate endorsements, guidelines, adoption)
 - Government (research funding, procurement guidelines)
 - Organizations (e.g. JUG)
 - Grass roots (how many count this as “primary or favorite language?”)
 - Technology (vendor support, 3rd party involvement)



Predictive Model Correlations

- Note: statistical correlations do not necessarily reflect cause-and-effect relations
- Top Intrinsic Factors (with 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)



How To Do Soft Real-Time with J2SE APIs

- Real-Time Garbage Collection
- Fixed priority scheduling
- Priority inheritance synchronization
- VM management services
 - To calibrate and monitor resource consumption and adjust resource budgets



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

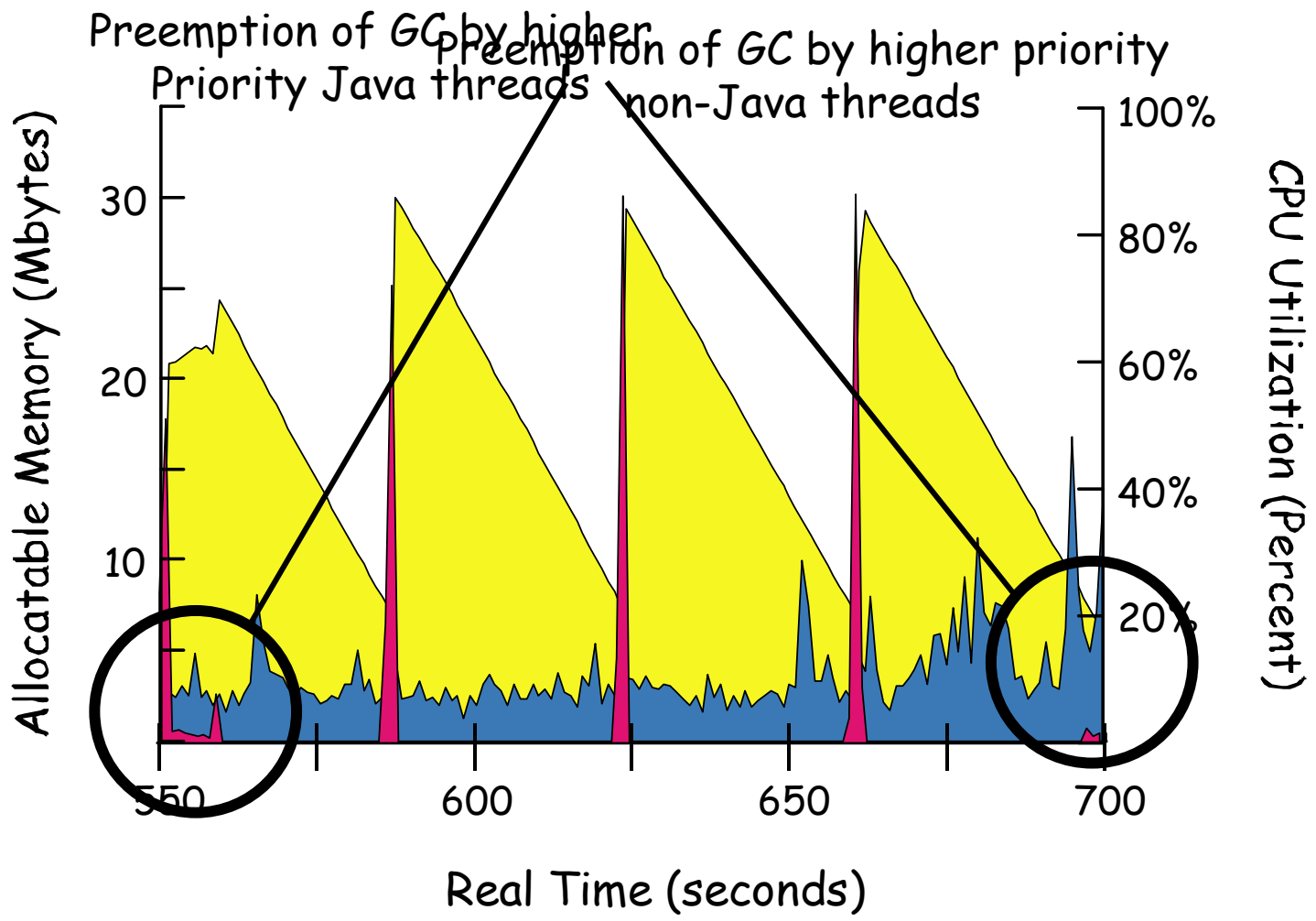


Real-Time GC

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

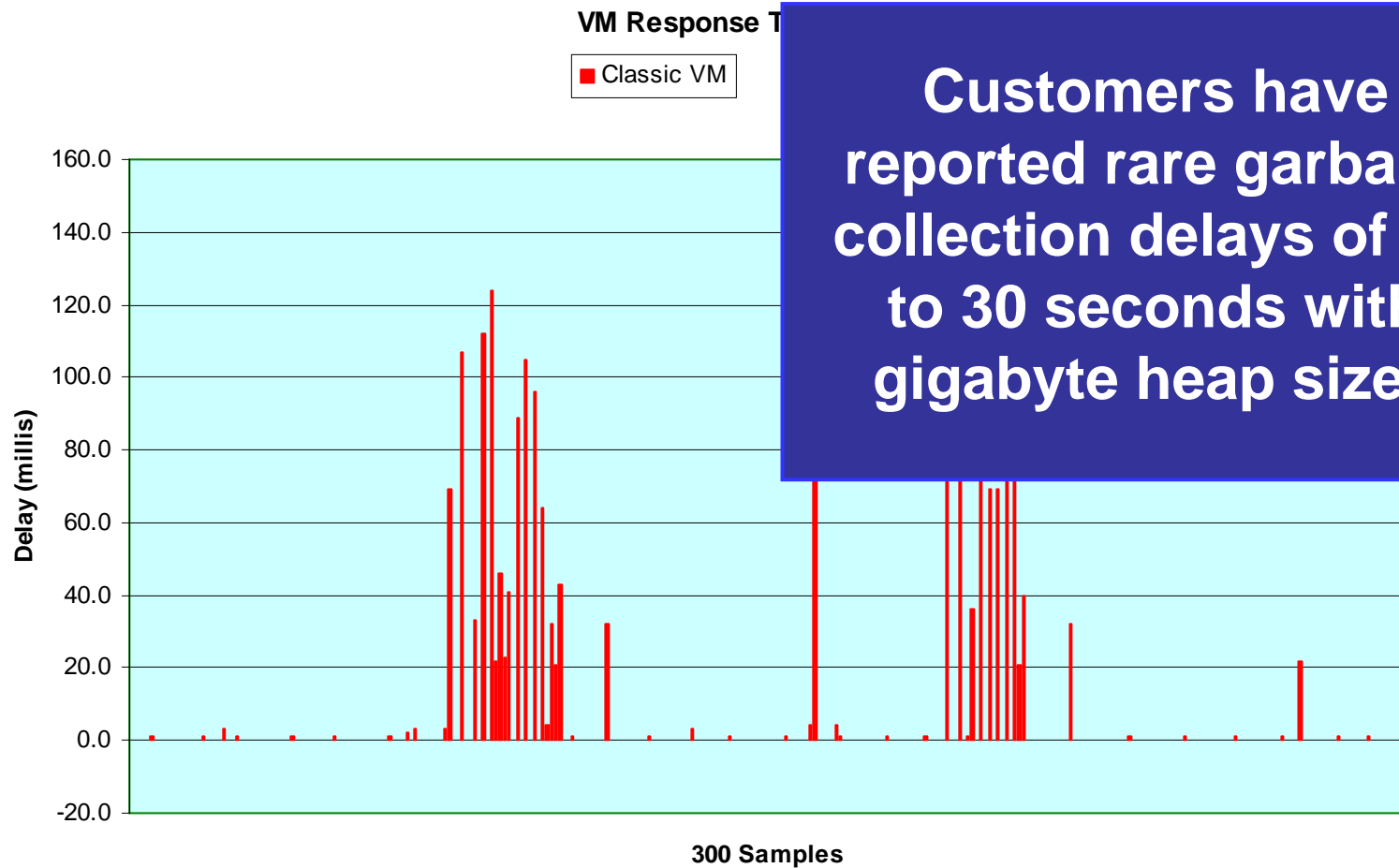


Trace of Garbage Collection Pacing





Classic VM running VM Response Test



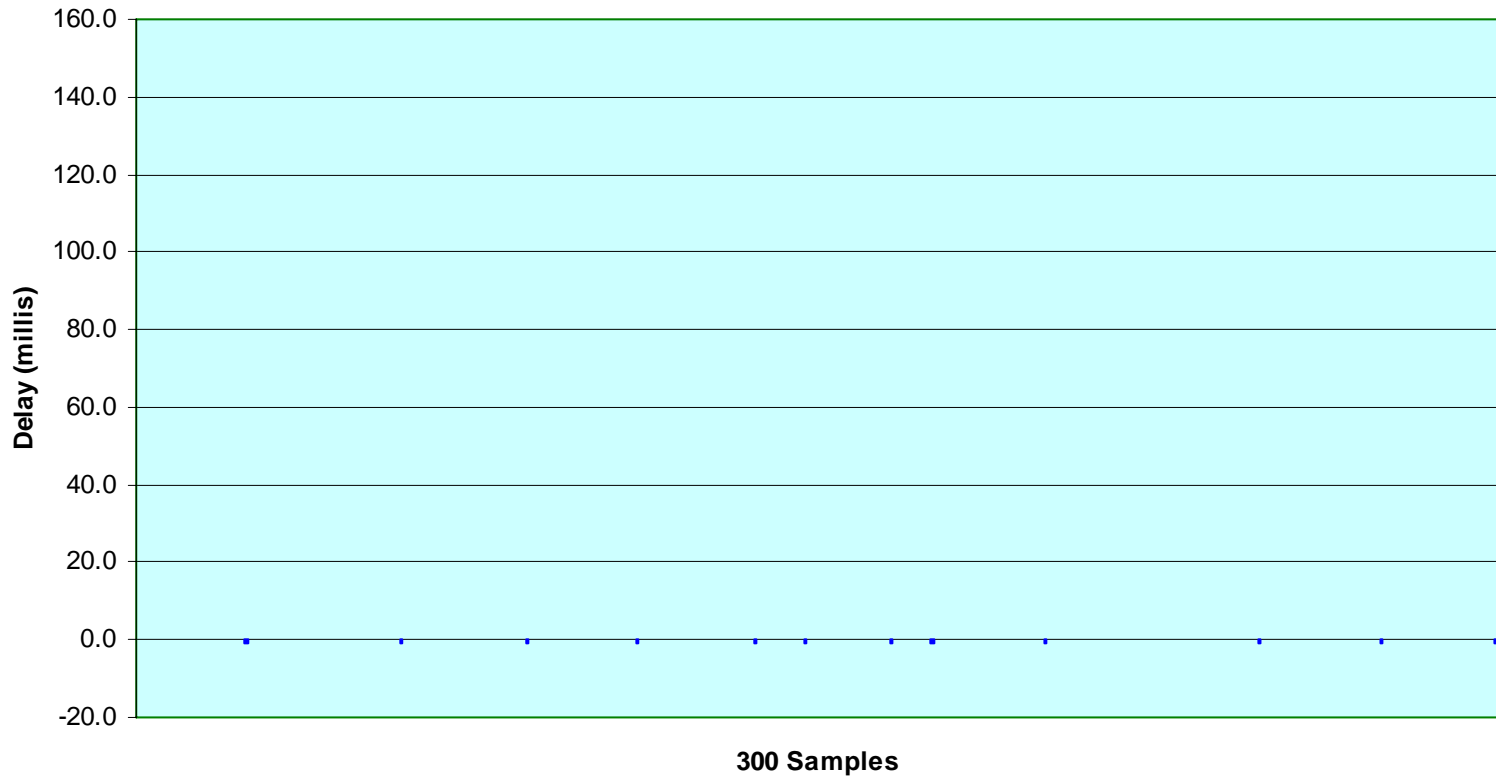
Customers have reported rare garbage collection delays of up to 30 seconds with gigabyte heap sizes



Real-Time VM running VM Response Test

VM Response Test

■ PERC





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)



Comparing C vs. Java Performance

	Optimized C	Java HotSpot	Ratio
Sieve (standalone)	1.20	3.07	2.56
Sieve (CaffeineMark)	0.90	1.48	1.64
Loop (CaffeineMark)	1.14	3.26	2.86
Logic (CaffeineMark)	0.65	1.48	2.28
Method (CaffeineMark)	0.71	1.08	1.52

- For large, complex, dynamic software systems, Java performance is much closer to C++ performance
- Since Java programmers complete required functionality with less total effort, they have more time available to tune and optimize
- In larger, more complex systems, C++ developers find it necessary to emulate many of the capabilities that are already built in to Java



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 μ s
- Typical periodic execution frequencies of 10-100 Hz



Relevance of Java to Hard Real-Time

- Relevant Questions:
 - Can Java address hard real-time constraints?
 - Can Java offer competitive (adequate?) throughput and memory footprint efficiency?
 - Will a Java variant that is designed to address determinism, throughput, and footprint constraints offer traditional Java developer productivity and reliability benefits?



Challenges of Embedded Development

- C does not scale well to sizes typical of modern systems
- Both C and C++ lack standard semantics for multi-threading, device I/O, and real-time
 - Every RTOS, architecture, compiler, and compiler optimization flag manifests different semantics
- As university instructional programs move towards Java, it is increasingly difficult to recruit competent Ada, C, and C++ programmers



Challenges of Real-Time Development

- No standard platforms
 - Without platform standards, there is no market for real-time COTS and open-source components
 - It is very difficult to create reusable real-time software components, because developers can not predict how future platforms might differ from current platforms



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)



Our Approaches to Mission-Critical and Safety-Critical Java

- Start with Standard Java platforms and existing RTSJ specification
- Define specific profiles that both subset and supplement established standards in an open consensus-based standards forum
- Target three complementary market segments: complex dynamic soft real-time; static, high-performance hard real-time; safety critical
- Submit resulting specifications to the JCP (and ISO if possible) for official standardization



Open Group Standardization of Safety and Mission Critical Java



- Driven by defense and commercial aerospace industries including:
 - U.S. Air Force, U.S. Navy, NASA, Boeing, Lockheed Martin, Northrop Grumman, Raytheon, ...
- Includes participation of OOTiA committee members (e.g. John Chilenski, Boeing)
- Builds on lessons learned by Ada 95 community
- U.S. Navy subcontract to Aonix for exploratory research and prototyping

NORTHROP GRUMMAN
Information Technology

Raytheon

BOEING®

LOCKHEED MARTIN





What are we trying to accomplish?

- Provide a feasible upgrade path from safety-critical Ada to safety-critical Java
 - More economically viable, broader customer base, larger supplier community, greater commercial acceptance
- Provide a superior alternative to doing safety-critical code in C and C++
 - Safer, easier to certify safe, less expensive to develop, less expensive to maintain, greater longevity and generality of software



Applying Java benefits to hard real-time

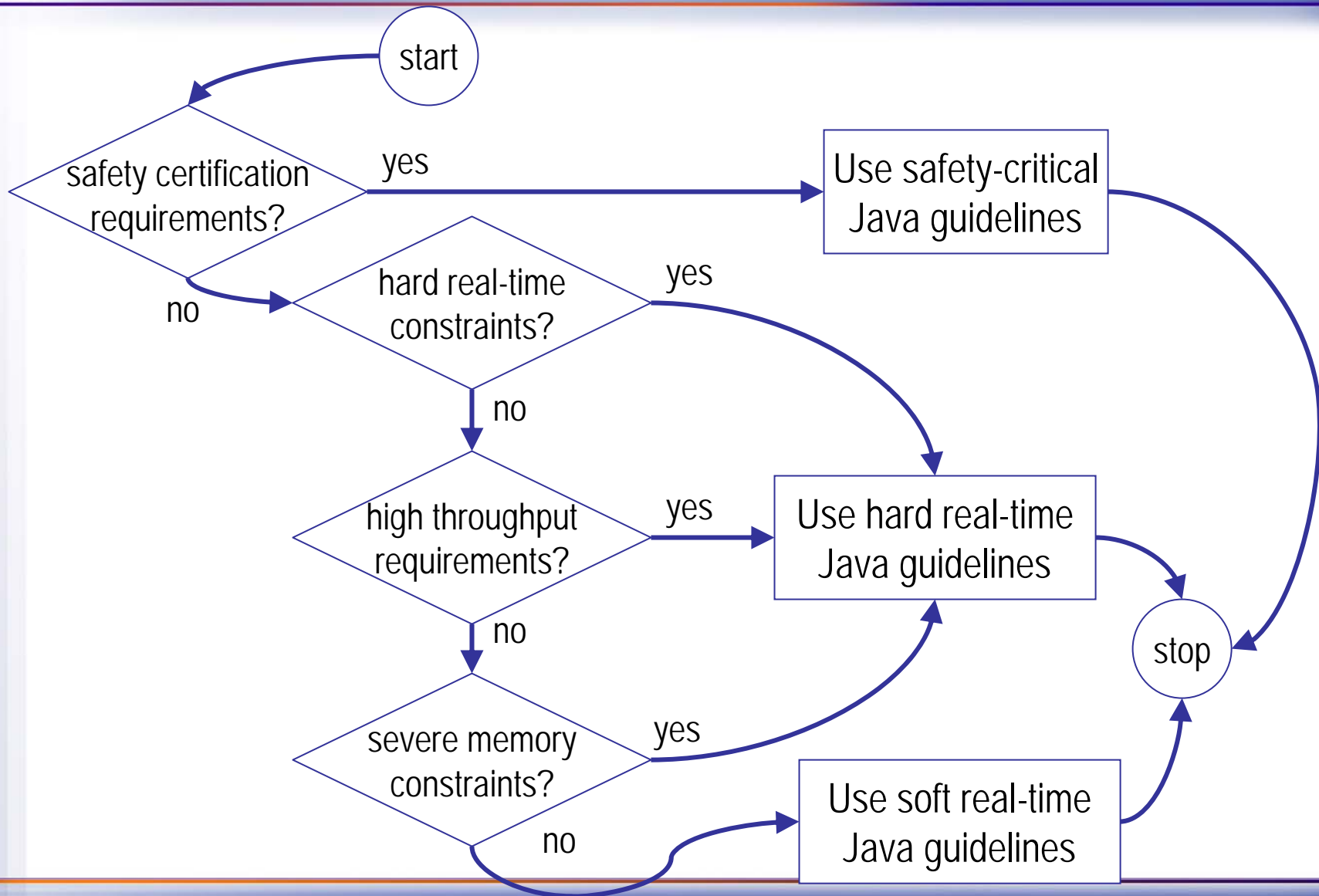
- Compared to traditional development of hard real-time software using C and/or C++, can a hard real-time mission-critical Java deliver 2-fold/10-fold productivity improvement?
 - Yes. But only if the hard real-time mission-critical Java standard provides:
 - Portable set of standard libraries
 - Strong type checking and composable object-oriented encapsulation
 - Portable byte-code representation of independently developed components
 - Safe logical partitioning between hard real-time and soft real-time components to make available shared access to all of the services of the soft real-time environment



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

Horses for Courses





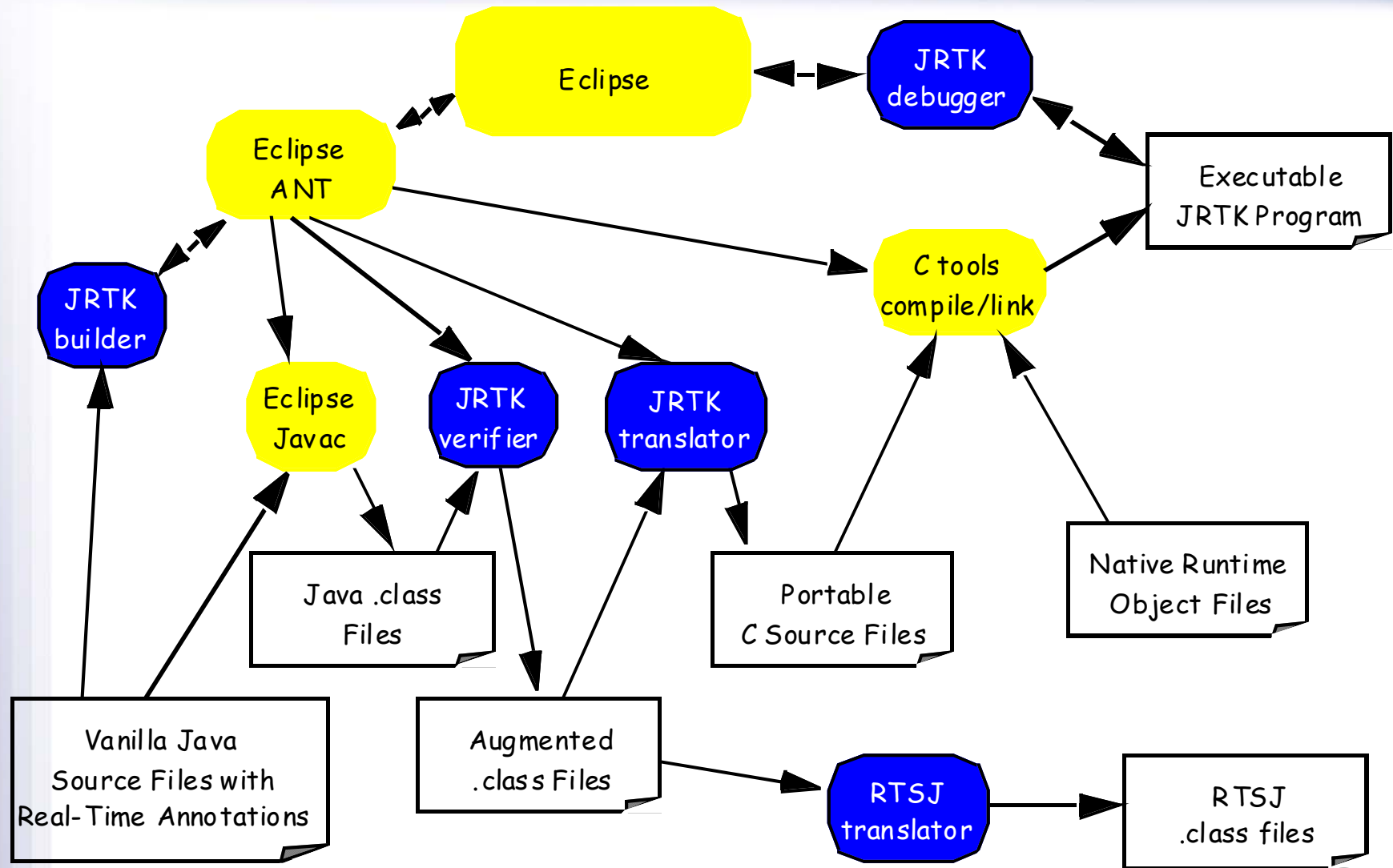
Preliminary Hard Real Time Performance

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

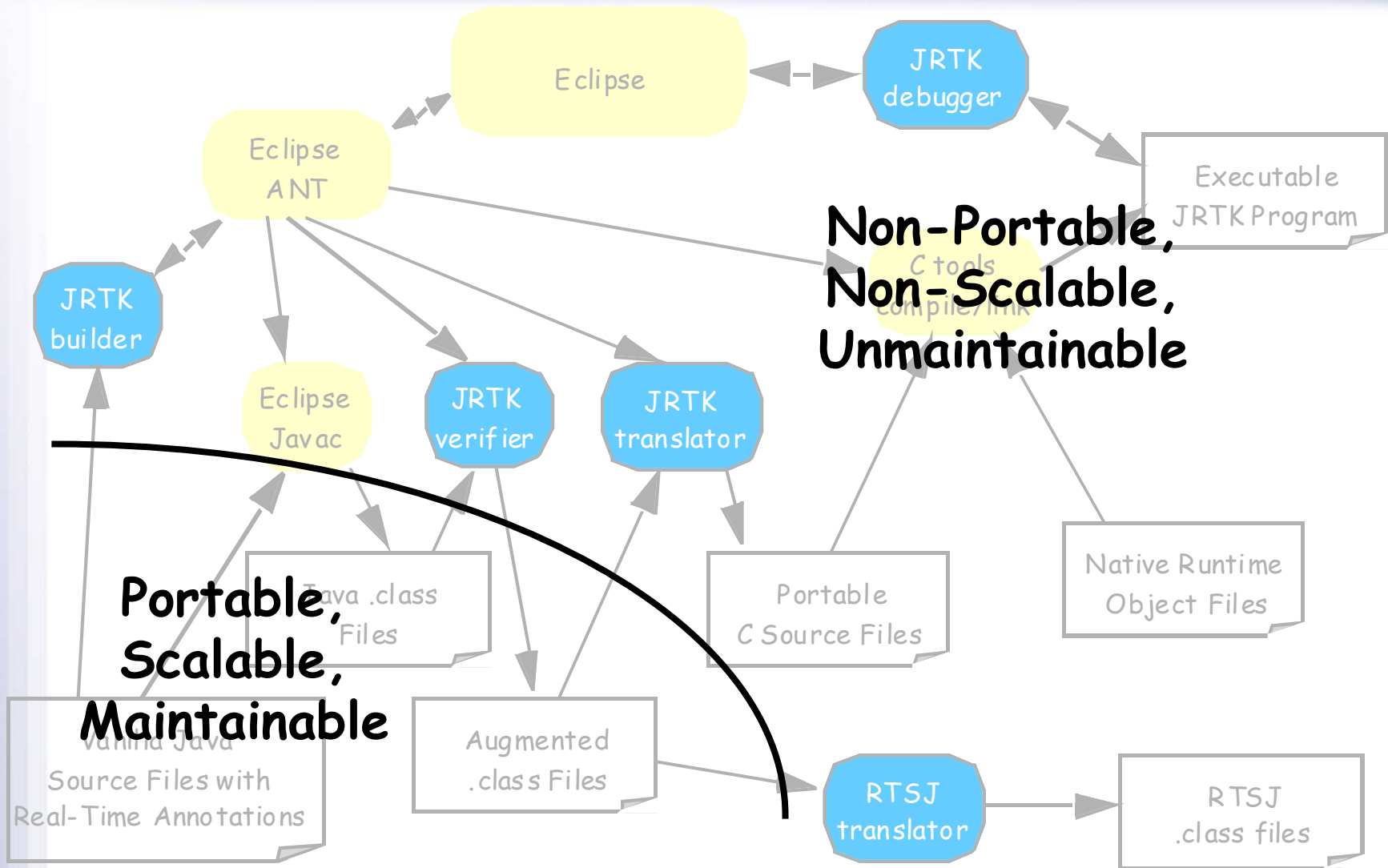


The Development Environment





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.

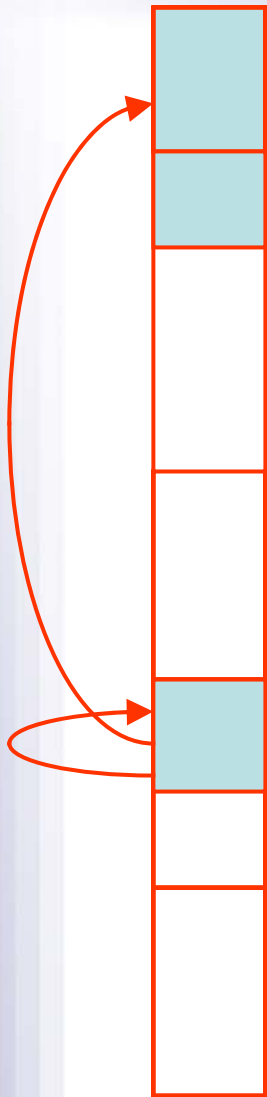
Memory for third spawned thread’s stack

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]    }
```



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] }
```



@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



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;
```



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] }
```

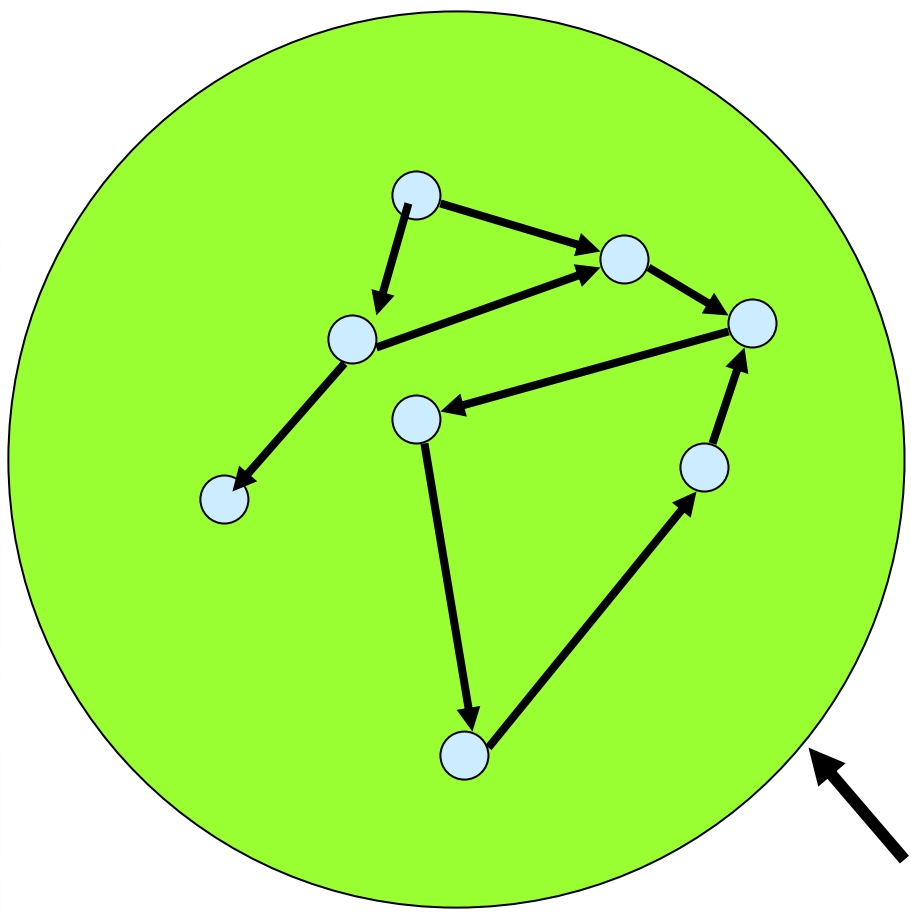


Annotated Program (3/3)

```
[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]}
```

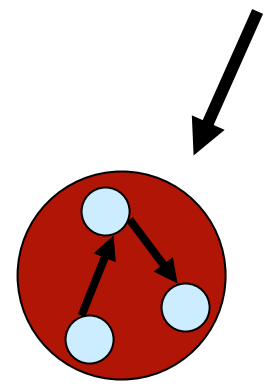


Cooperating HRT Components

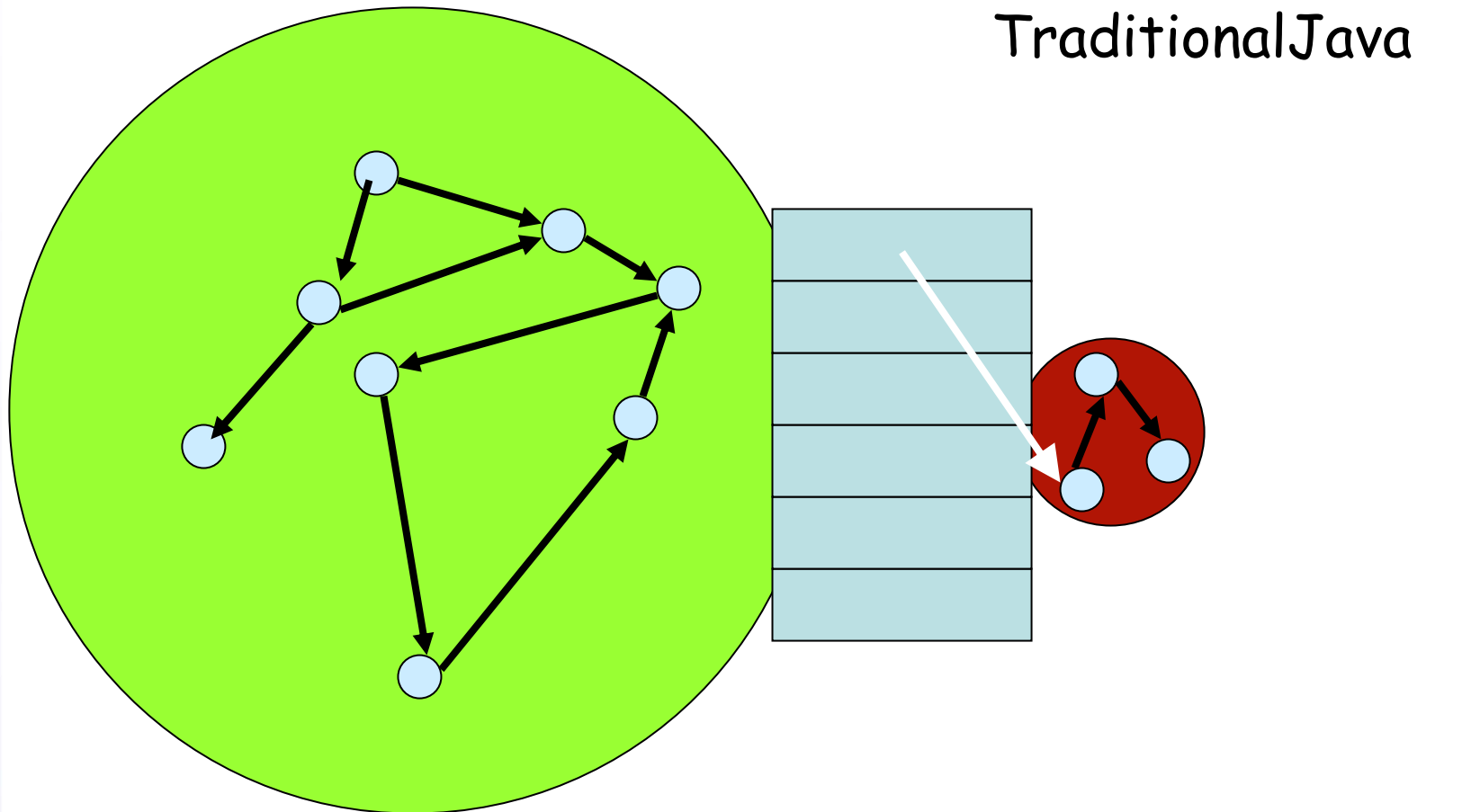


PERC Ultra Virtual Machine

PERC Pico Hard
Real-Time
Execution Engine

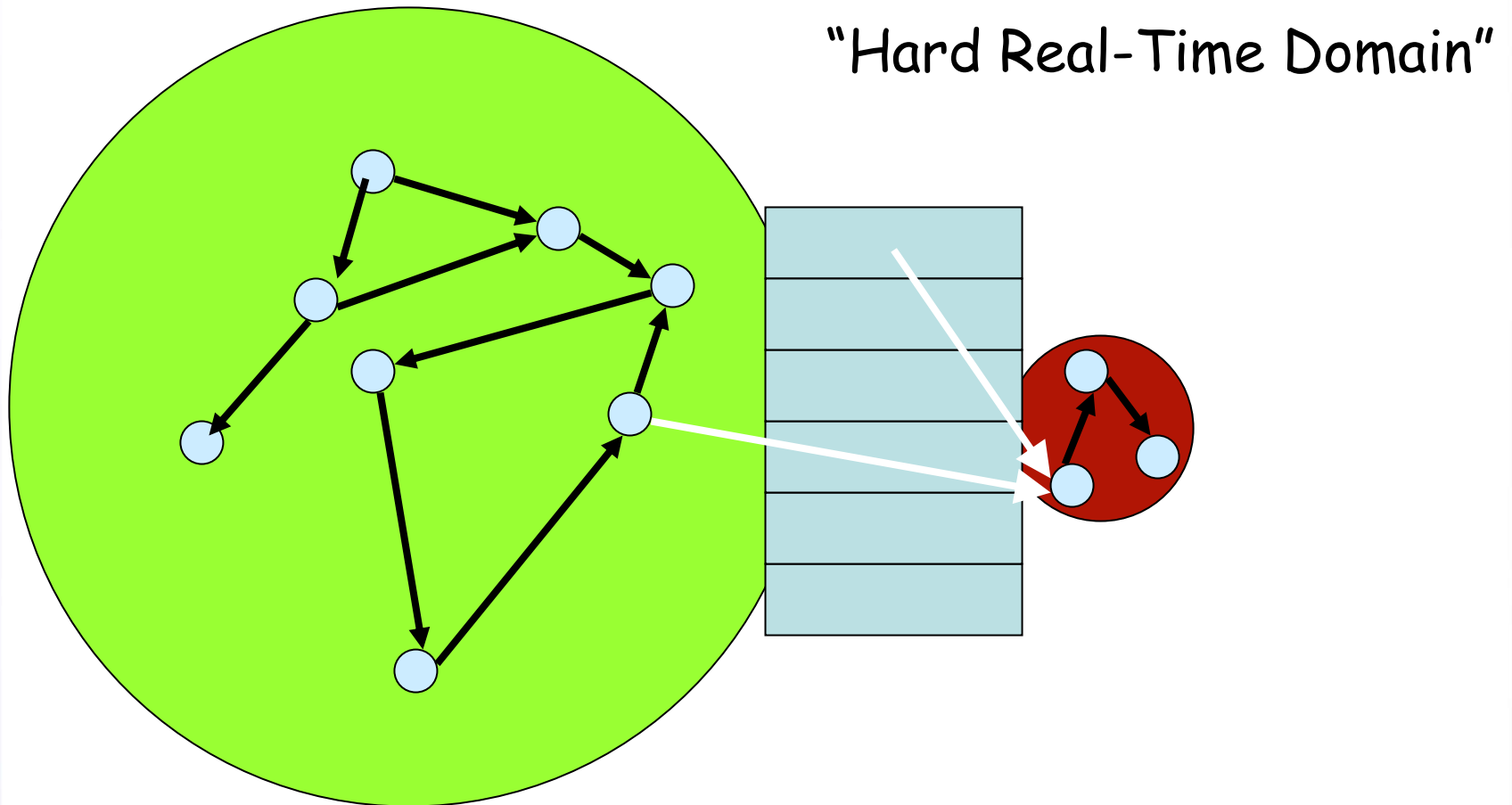


TraditionalJava Registry



`TraditionalJava.publish("deviceXYZ", x)`

Traditional Java 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