# High-level Programming Models for Real-time

Jan Vitek

PURDUE UNIVERSITY

Fiji Systems LLC

# Real-time embedded systems

- **Large and complex** — from a few hundred lines of assembly to 20 mio lines of Ada for the Space Station Freedom
- **Concurrent control of separate components**— devices operate in parallel in the real-world; model this by concurrent entities
- **Facilities to interact with special purpose hardware** — need to be able to program devices in a reliable and abstract way
- **Extreme reliability and safe** — embedded systems control their environment; failure can result in loss of life, or economic loss
- **Guaranteed response times** — must predict with confidence the worst case; efficiency important but predictability is essential

# A new software crisis?

- Development time, code & certification are increasingly criteria
- For instance in the automotive industry:
  - 90% of innovation driven by electronics and software — *Volkswagen*
  - 80% of car electronics in the future will be software-based — *BMW*
  - 80% of our development time is spent on software — *JPL*
- Worst, software is often the source of missed project deadlines.

# A new software crisis?

- Typical productivity
  - *5 Line of Code / person / day*
  - From requirements to testing: *1 kloc / person / year*
- Typical avionics “box”
  - *100 kloc ⇒ 100 person years of effort*
  - Costs of modern aircraft is ~$500M

# A new software crisis?

- The important metrics are thus
  - Reusability
  - Software quality
  - Development time
- The challenges are
  - Sheer number and size of systems
  - Poor programmer productivity
- The solutions are
  - Better processes (software engineering)
  - Better tools (verification, static analysis, program generation)
  - Better languages and programming models

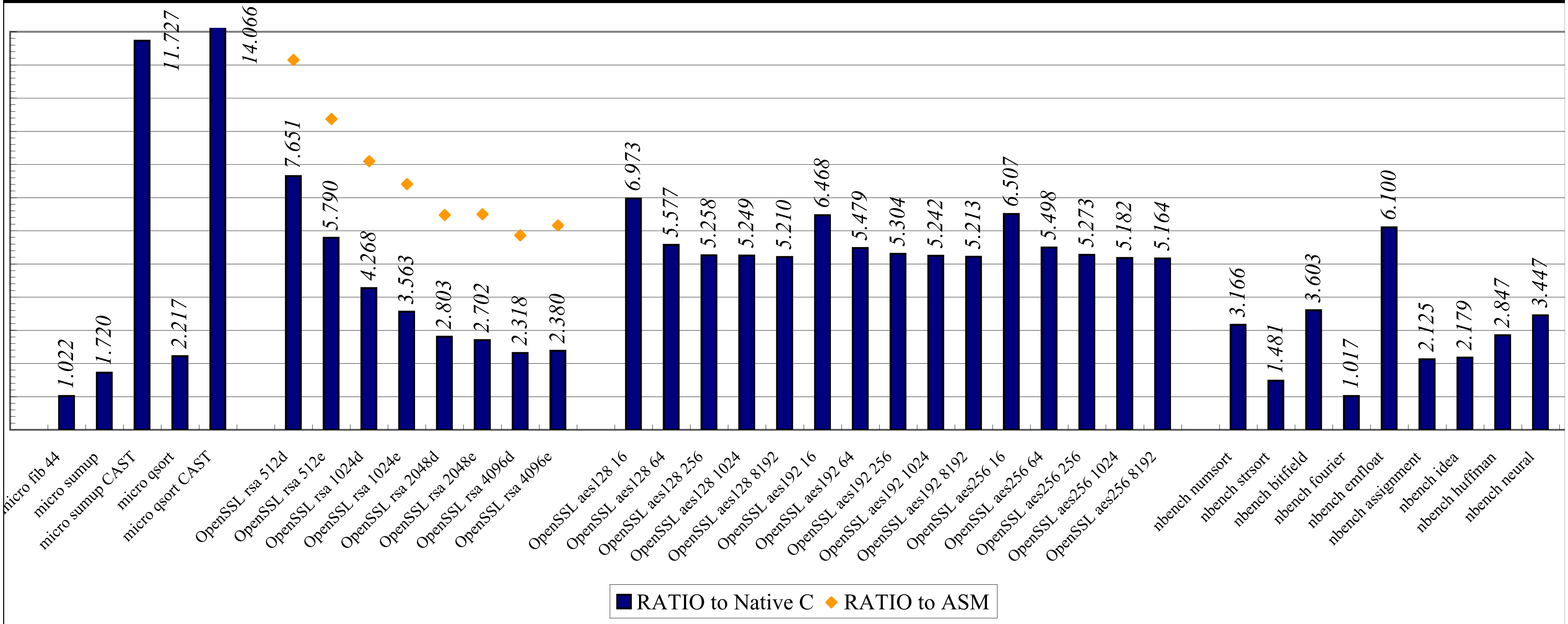
# What programming models?

- The **programming model** for most real-time systems is 'defined' as a function of the hardware, operating system, and libraries.
  - Consequently real-time systems are not portable across platforms

# What programming model?

- "Real-time systems require fine grained control over resources and thus the language of choice is C or assembly"
- ...entails the software engineering drawbacks of low-level code
- Consider the following list of defects that have to be eradicated (c.f. "Diagnosing Medical Device Software Defects" Medical DeviceLink, May 2009):
  - Buffer overflow and underflow (does not occur in a HLL)
  - Null object dereference (checked exception in a HLL)
  - Uninitialized variable (does not occur in a HLL)
  - Inappropriate cast (all casts are checked in a HLL)
  - Division by zero (checked exception in a HLL)
  - Memory leaks (garbage collection in a HLL)

# What programming model?




- Some of the guarantees can be retrofitted on legacy C programs.
- [Implementation of the Memory-safe Full ANSI-C Compiler, PLDI 2009]

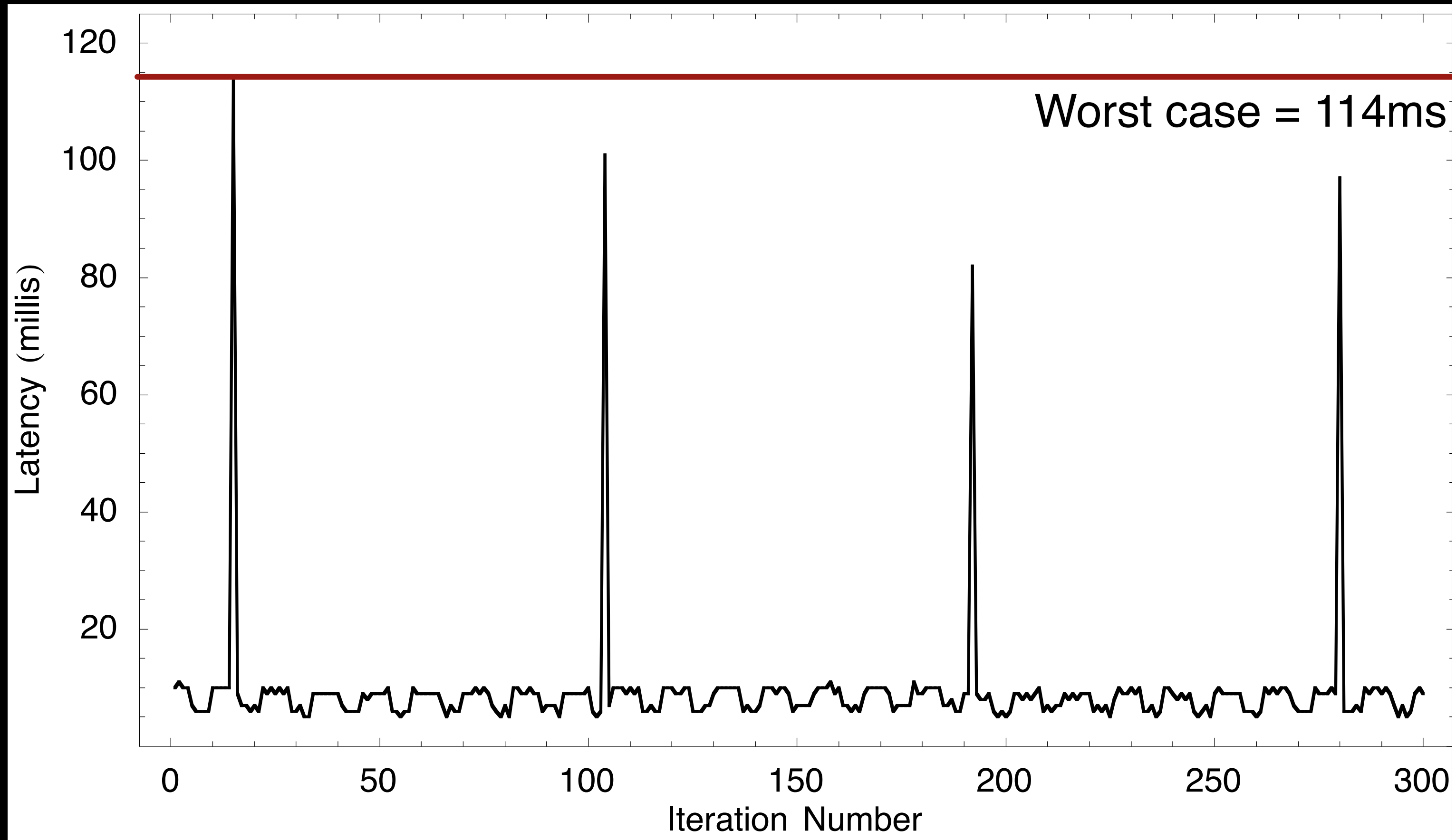

1

# Java?

- Object-oriented programming helps **software reuse**
- Mature **development environment** and **libraries**
- **Garbage collected** & **Memory-safe** high-level language
- **Portable**, little implementation-specific behavior
- **Concurrency** built-in, support for SMP, memory model
- **Popular** amongst educators and programmers

# Java?

- Predictable?




- **Java Collision Detector** running at 20Hz.
  - *Bartlett's Mostly Copying Collector. Ovm. Pentium IV 1600 MHz, 512 MB RAM, Linux 2.6.14, GCC 3.4.4*
- GC pauses cause the collision detector to miss up to three deadlines…*this is not a particularly hard should support KHz periods*

# The Real-time Specification for Java (RTSJ)

- Java-like programming model:
  - Shared-memory, lock-based synchronization, first class threads.
- Main real-time additions:
  - Physical memory access (memory mapped I/O, devices, …)
  - Real-time threads (heap and no-heap)
  - Synchronization, Resource sharing (priority inversion avoidance)
  - Memory Management (region allocation + real-time GC)
  - High resolution Time values and Clocks
  - Asynchronous Event Handling and Timers
  - Asynchronous Transfer of Control

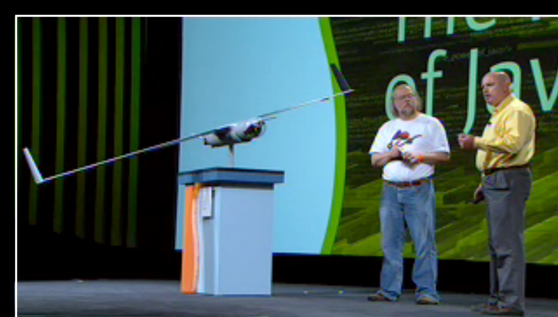
# Ovm

The Real-time Java experience

# Ovm


- Started on Real-time Java in 2001, in a DARPA funded project. At the time, no real RTSJ implementation.
- Developed the Ovm virtual machine framework, a clean-room, open source RT Java virtual machine.
- Fall 2005, first flight test with Java on a plane.

*Duke's Choice Award*

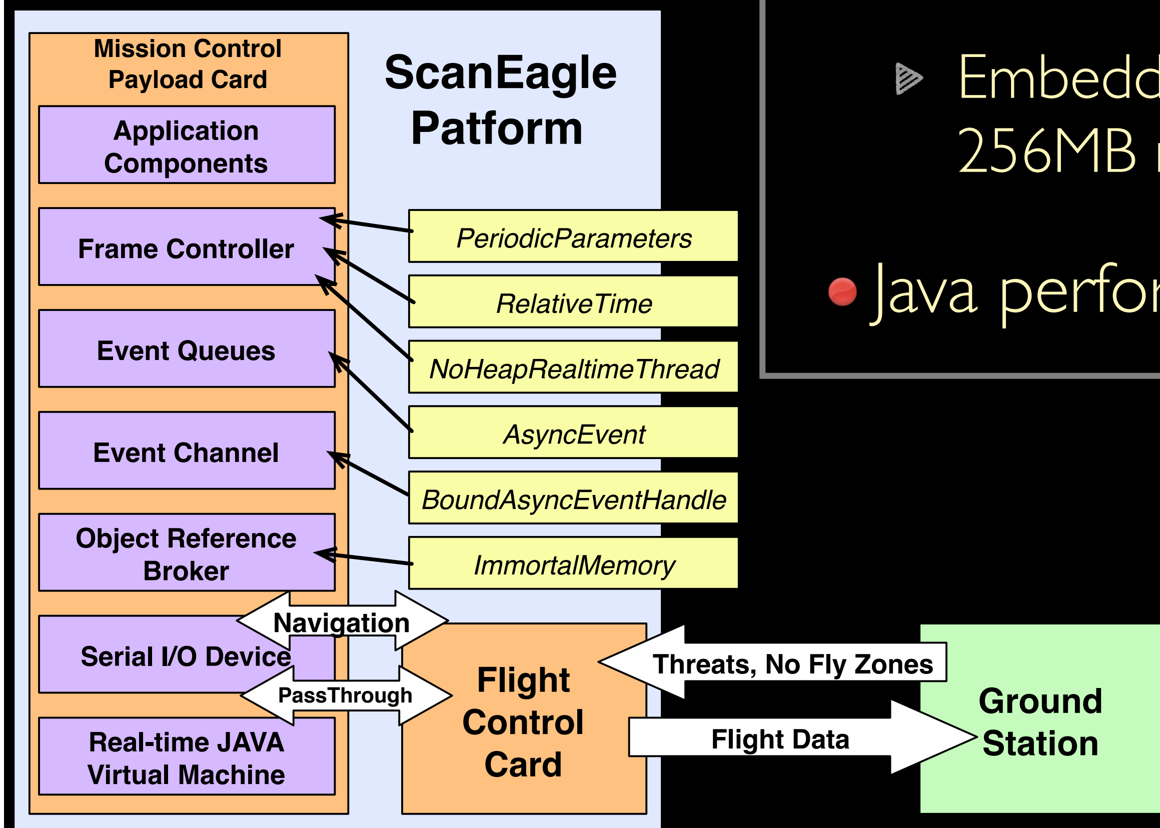


# Case Study: ScanEagle

# ScanEagle

# ScanEagle

- Flight Software:
  - 953 Java classes, 6616 methods. Multiple Priority Processing:
    - High (20Hz) - Communicate with Flight Controls
    - Medium (5 Hz) - Computation of navigation data
    - Low (1 Hz) - Performance Computation
  - Embedded Planet 300 Mhz PPC, 256MB memory, Embedded Linux
- Java performed better than C++

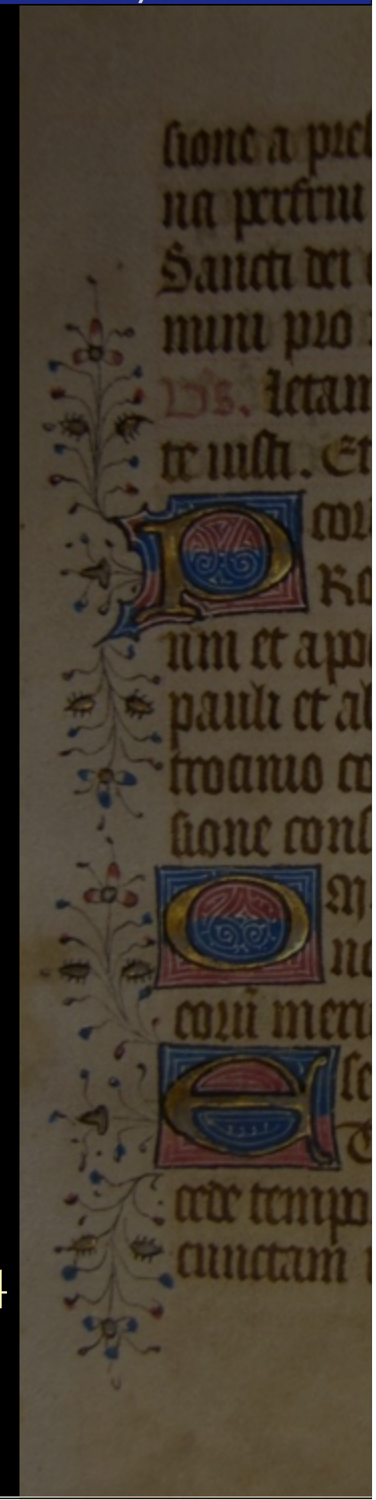



# References and acknowledgements


- **Team**
  - *J. Baker, T. Cunei, C. Flack, D. Holmes, C. Grothoff, K. Palacz, F. Pizlo, M. Prochazka and also J. Thomas, K. Grothoff, E. Pla, H. Yamauchi, P. McGachey, J. Manson, A. Madan, B. Titzer*
- **Funding:** *DARPA, NSF, Lockheed Martin, Boeing*

- **Availability:** open source, http://www.cs.purdue.edu
- **Paper trail**


- *A Real-time Java Virtual Machine for Avionics.* RTAS, 2006
- *Scoped Types and Aspects for Real-Time Systems.* ECOOP, 2006
- *A New Approach to Real-time Checkpointing.* VEE, 2006
- *Real-Time Java scoped memory: design patterns, semantics.* ISORC, 2004
- *Subtype tests in real time.* ECOOP, 2003
- *Engineering a customizable intermediate representation.* IVME, 2003

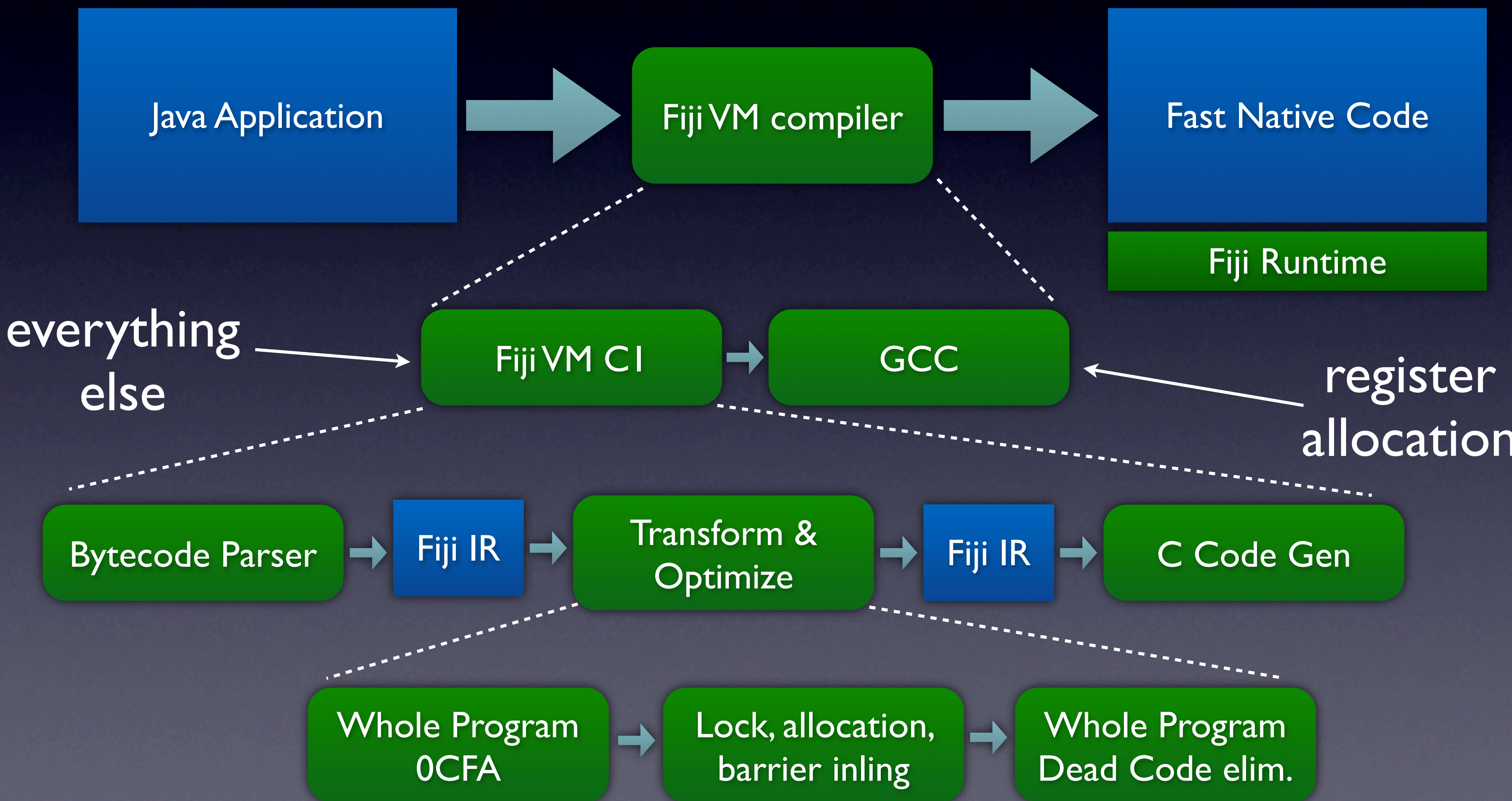




2

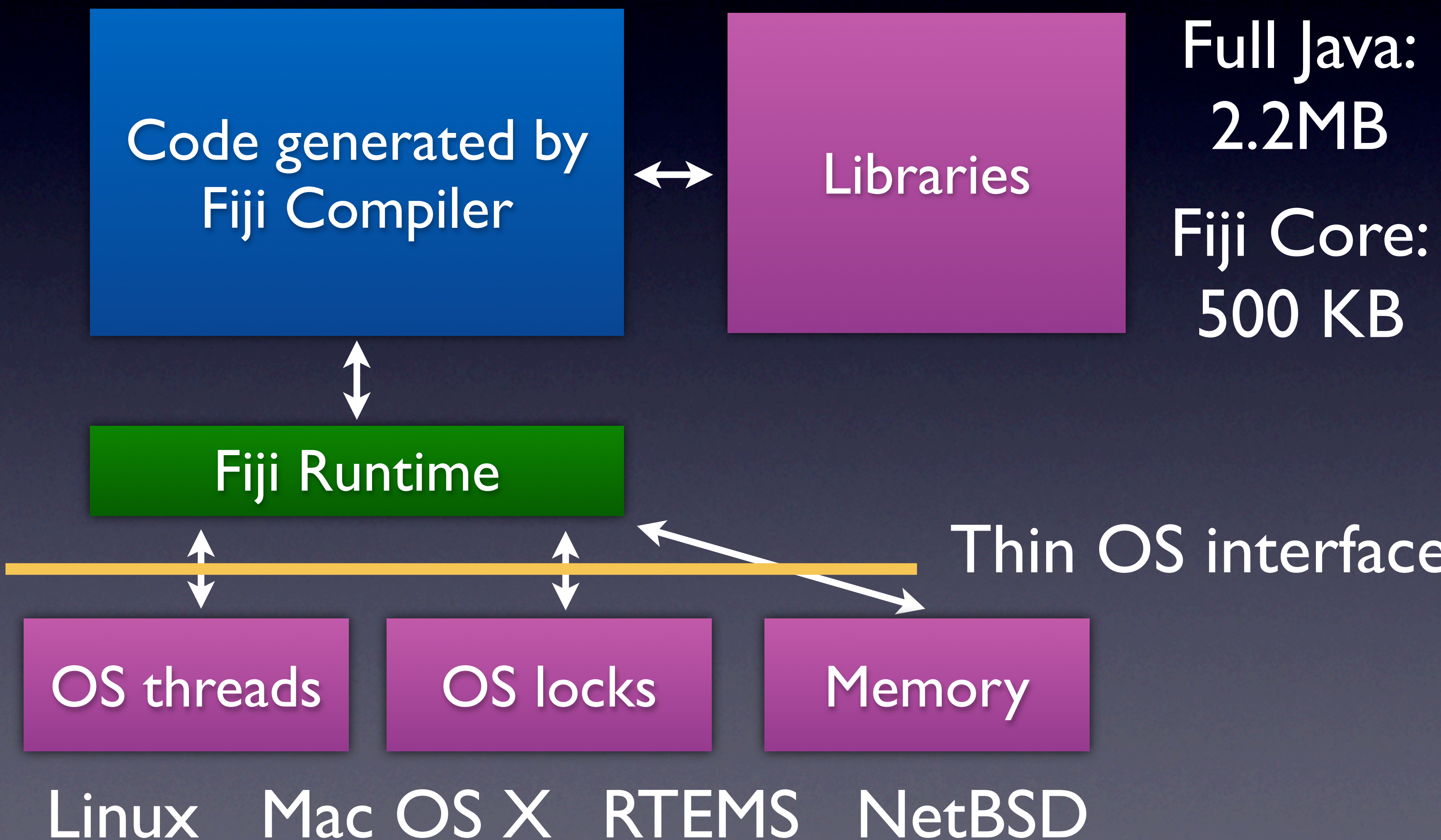
# Fiji VM technology

- Proprietary ahead-of-time compiler
  - Java bytecode to portable ANSI C
  - high-performance, predictable execution
  - Multi-core ready
- Proprietary real-time garbage collection
  - easy-to-use, fully preemptible, small overhead
  - zero pause times for RT tasks
- Current platforms
  - OS X, Linux, RTEMS
  - x86 and x64, SPARC, **LEON2/3, ERC32,** and PowerPC
  - **200KB** footprint

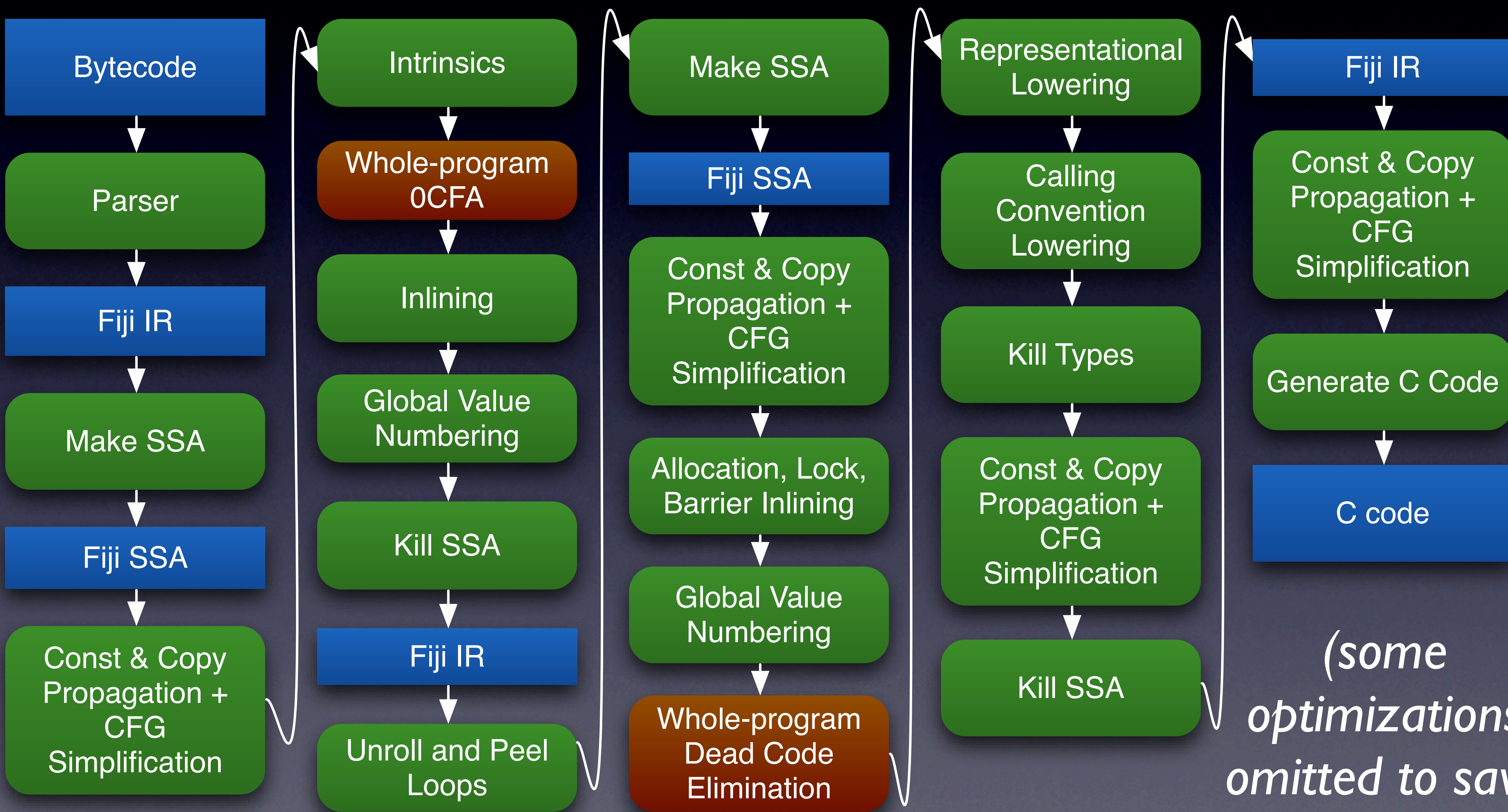
# The Fiji VM Overview



# The Runtime


Code generated by Fiji Compiler
Libraries
Full Java: 2.2MB
Fiji Core: 500 KB
Fiji Runtime
Thin OS interface
OS threads
OS locks
Memory
Linux
Mac OS X
RTEMS
NetBSD

# Better view of Fiji C1




*(some optimizations omitted to save space)*

# Performance/Predictability

| | |
|---|---|
| local assignments, simple arithmetic, casts, conditionals | *same performance as C/C++* |
| loops, method invocation, field/array access, static initialization | *slightly slower than C/C++* |
| allocation, locking, exceptions | *faster than C/C++* |
| condition variables, threading, I/O | *identical to C/C++* |

# CDx Benchmark

- Representative Real-time benchmark
  - Aircraft detection based on simulated radar frames
- CDc - written in idiomatic C
- CDj - written in idiomatic Java
- Uses many arrays and is computationally intensive

# CDx Benchmark

- The algorithm detects a collision whenever the distance between aircraft is smaller than a specified "proximity radius"
- Step 1: ⟵ eliminates planes at large distances
  - split aircraft into clusters
- Step 2:⟵ closer examinations of potential collisions
  - for each cluster determine actual collisions

# CDx Benchmark

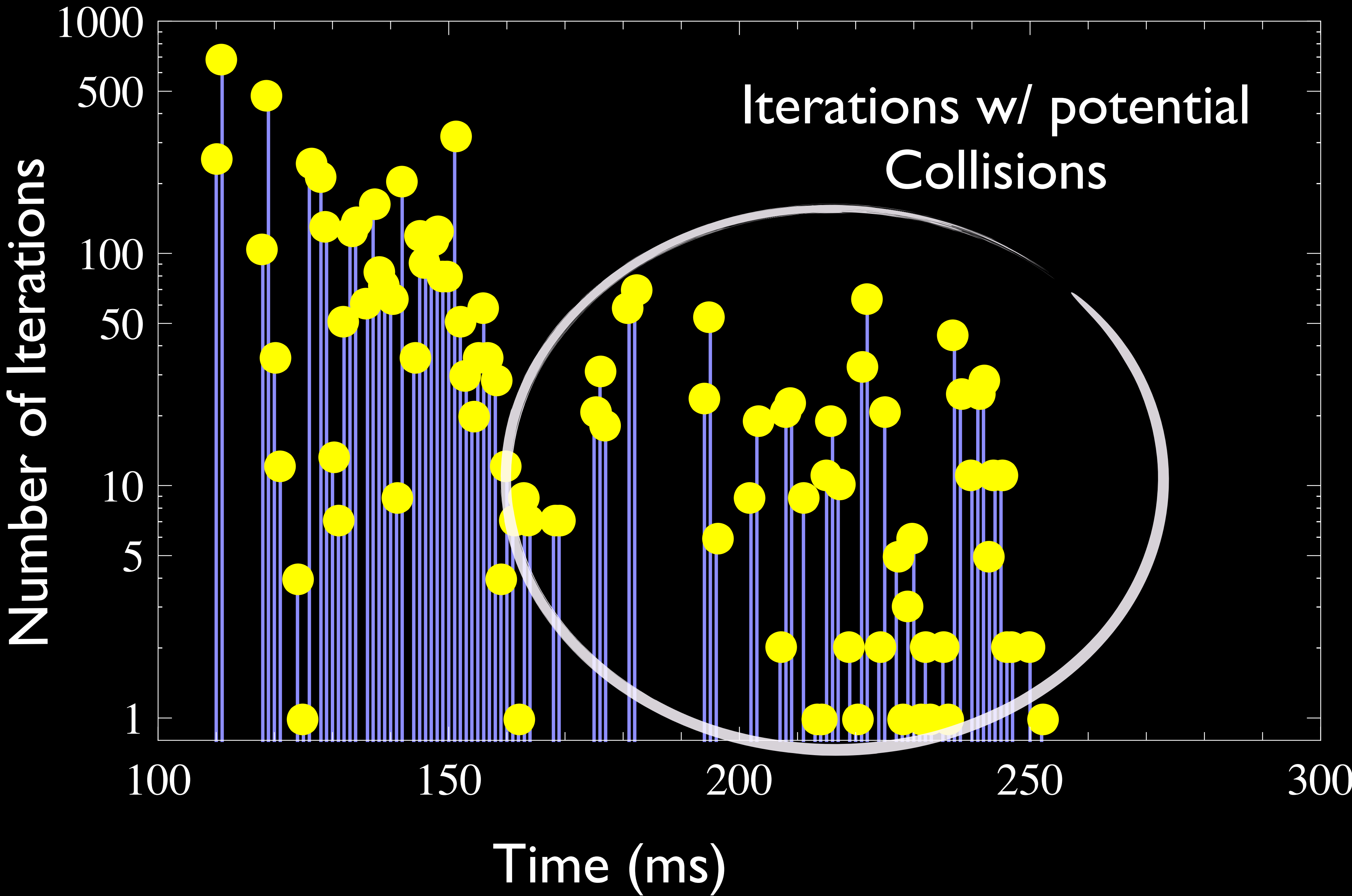
- What if we run CDx on a real-time setup?
  - RTEMS 4.9.1 (hard RTOS microkernel: no processes or virtual memory)
  - 40MHz LEON3 with 64MB RAM (radiation-hardened SPARC)
- This is the platform used by ESA and NASA

# CDx Configuration

- 6 airplanes in our airspace
- execute over 10,000 radar frames
  - runs take on average 45 minutes
  - slight modification to generate frames
- 300ms period for the collision detector task
  - between 145ms - 275ms
  - leaves less than 50% of the schedule for the GC

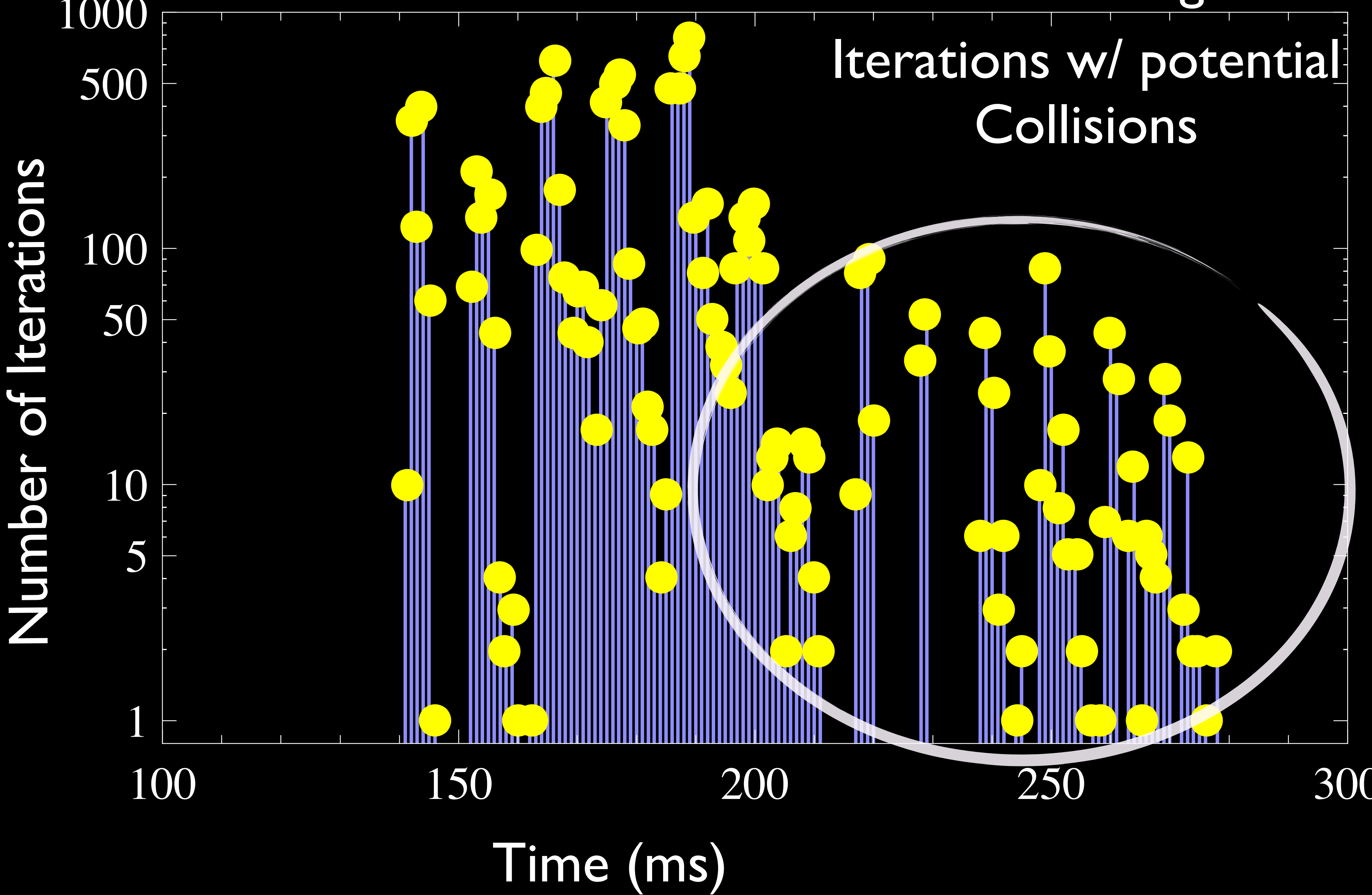
Worst case Fiji VM
Worst case C
10% slower
300
250
200
150
100
2000
2050
2100
2150
2200
Frame Number vs. Execution Time (ms)

# CDc Summary


1000
500
100
50
10
5
1
100
150
200
250
300
Number of Iterations
Time (ms)
Iterations w/ potential
Collisions

# CDj Summary

30% slower on average



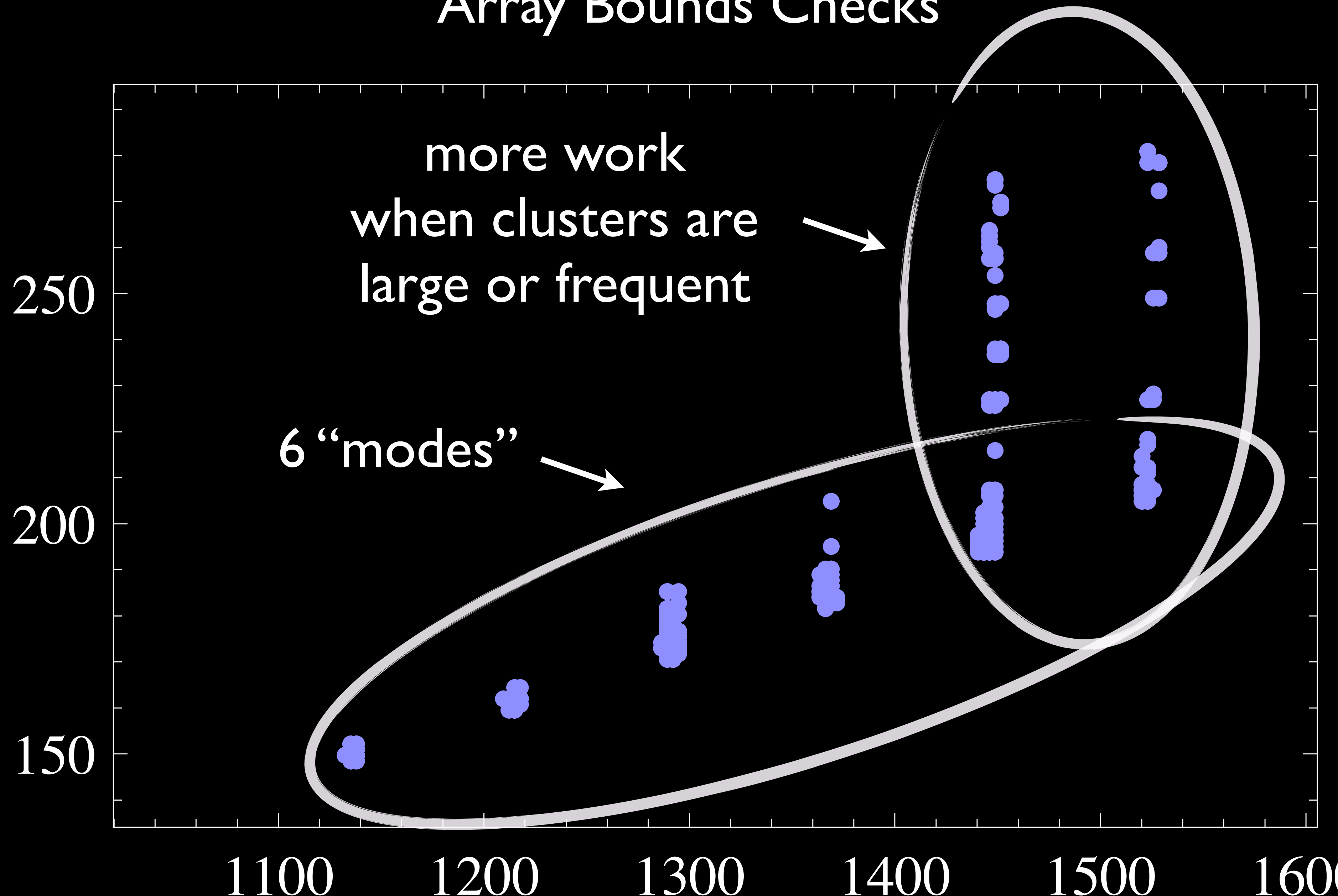
# Source of overheads

measured using RTBx data logger

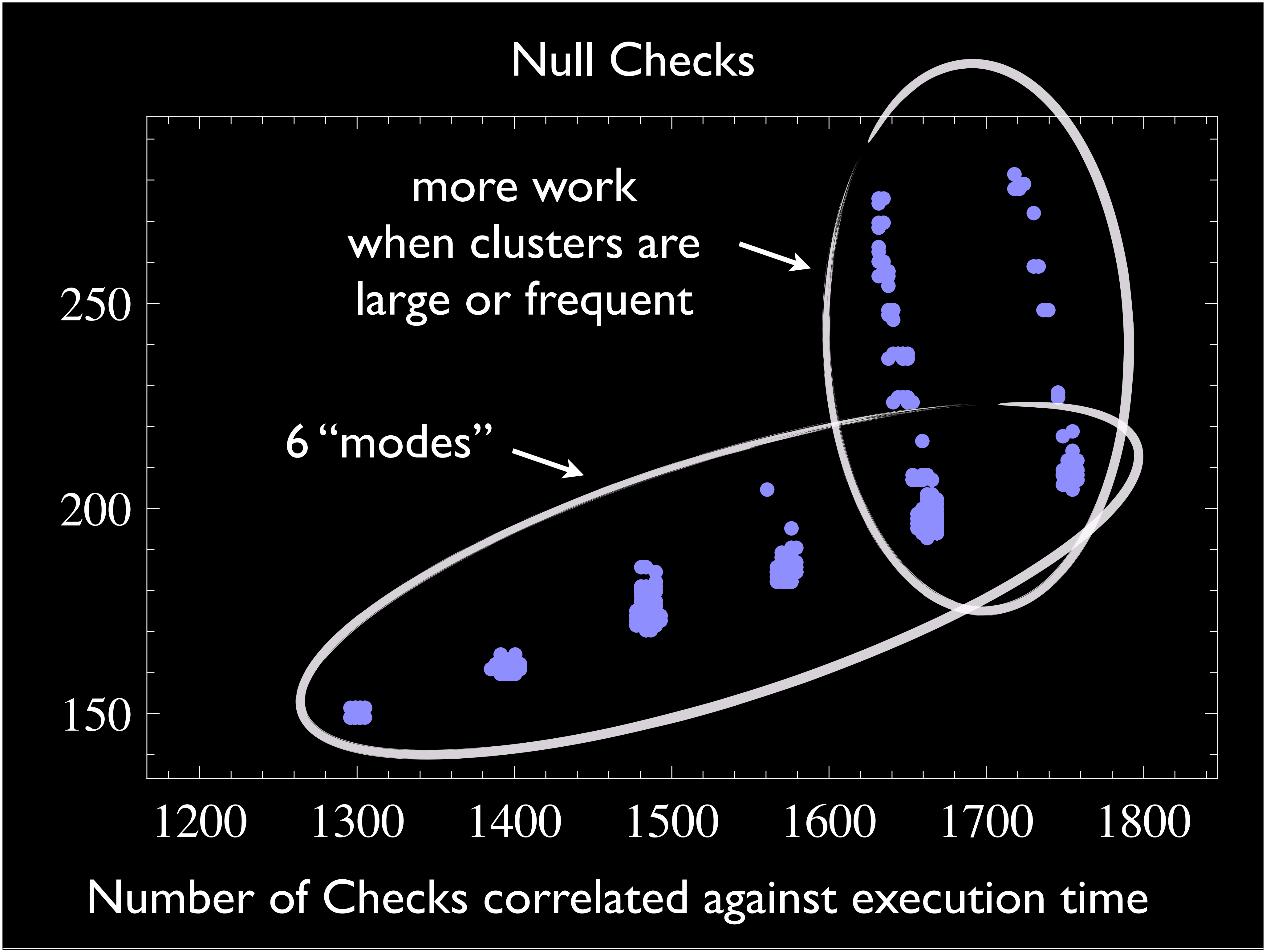
- Expect to see larger Java overheads when potential collisions are detected
- Array bounds checks
- Type checks
- Null checks

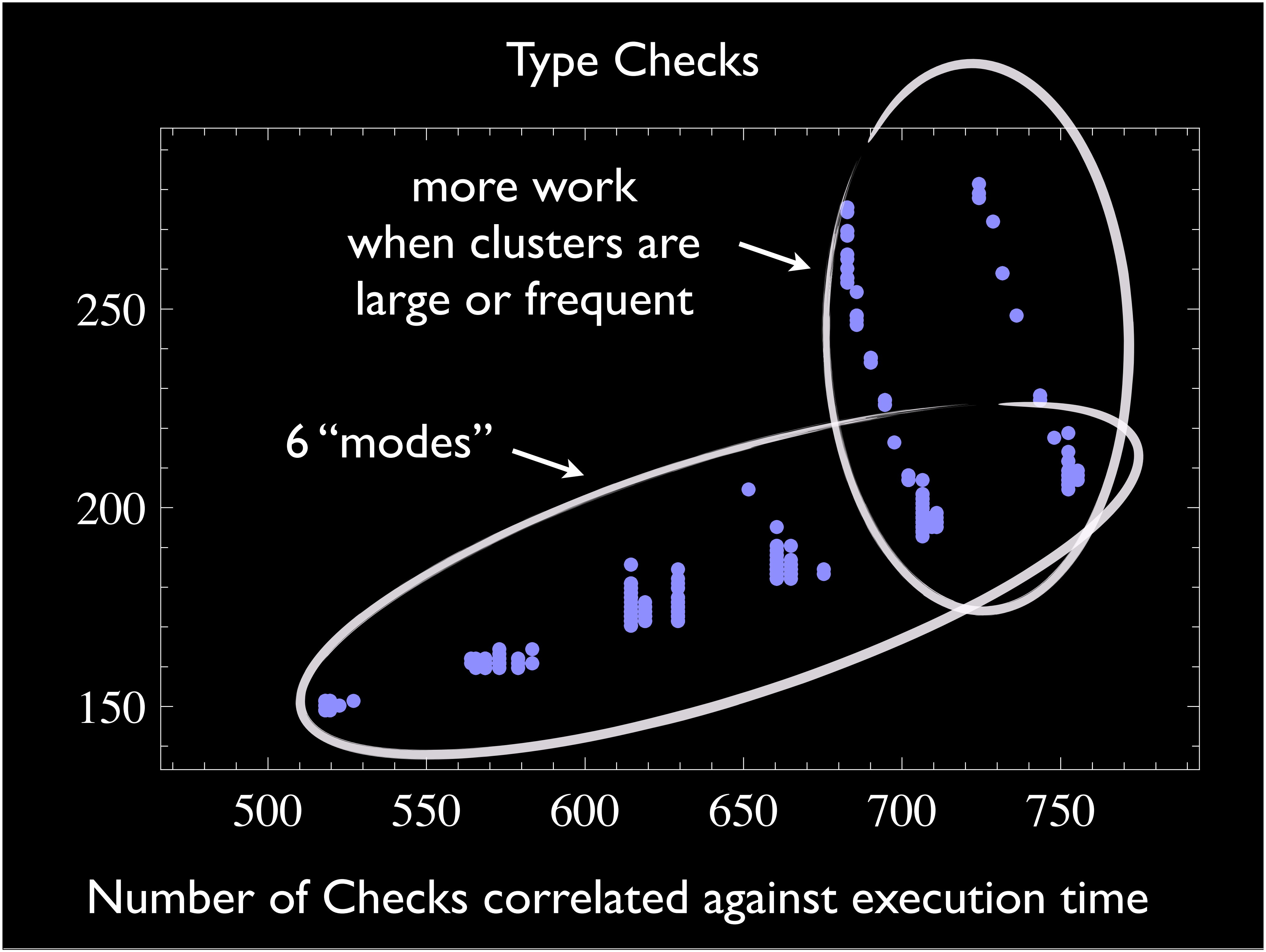

www.rapitasystems.com

# Array Bounds Checks




Number of Checks correlated against execution time


Null Checks
more work
when clusters are
large or frequent
6 “modes”
250
200
150
1200
1300
1400
1500
1600
1700
1800
Number of Checks correlated against execution time


Type Checks
more work
when clusters are
large or frequent
6 “modes”
250
200
150
500
550
600
650
700
750
Number of Checks correlated against execution time

# Correlation Java vs C when running on RTEMS/LEON3