Programming Models for Real-time Computing

or

Is Java ready for Real-time?

Jan Vitek

Real-time systems

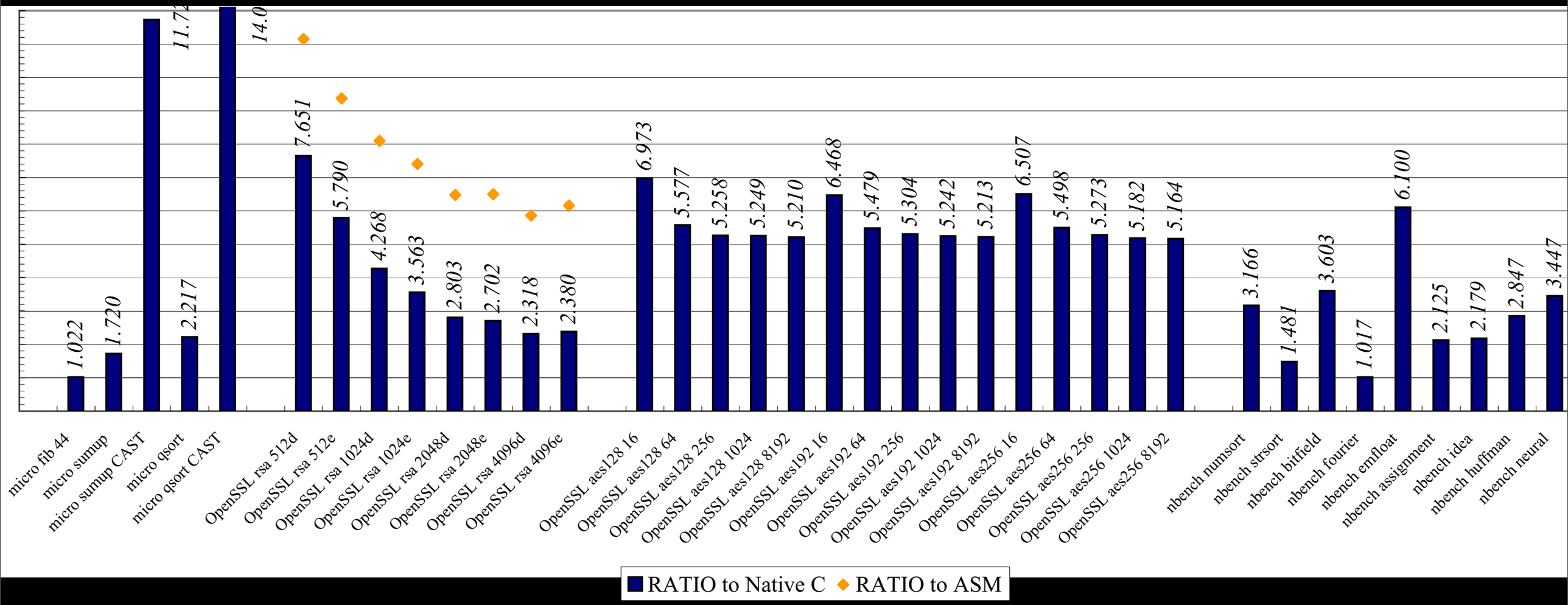
- **Guaranteed response times** — must predict with confidence the worst case; efficiency important but predictability is essential
- **Large and complex** — from a few hundred lines of assembly to 20 mio lines of Ada for the Space Station Freedom
- **Concurrent control of separate components**— devices 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

What programming model?

- "fine grained control over resources means C or assembly"
- 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)
Memory leaks	(garbage collection in a HLL)

What programming model?



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

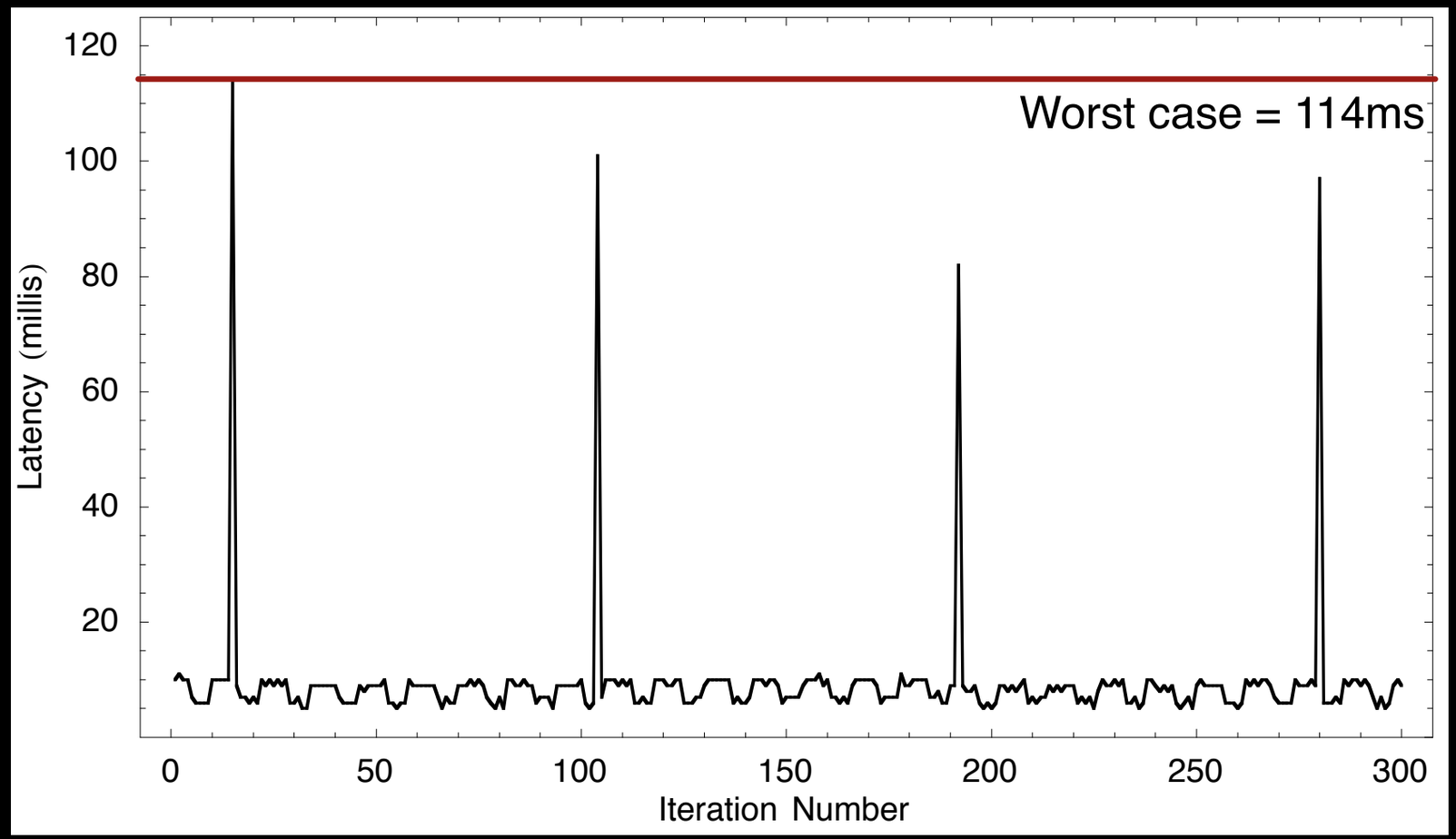
A new software crisis?

- Development time, code & certification are increasingly criteria
- For instance:
  - 90% of innovation driven by electronics and software — *Volkswagen*
  - 80% of our development time is spent on software— *JPL*
- **Productivity**
  - From requirements to testing: *1 kloc / person / year*

1

Java

- Predictable?



- **Java Collision Detector** running at 20Hz. *Bartlett's Mostly Copying Collector. Ovm. Pentium IV 1600 MHz, 512 MB, Linux 2.6. GCC 3.4*

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**
  - **...**

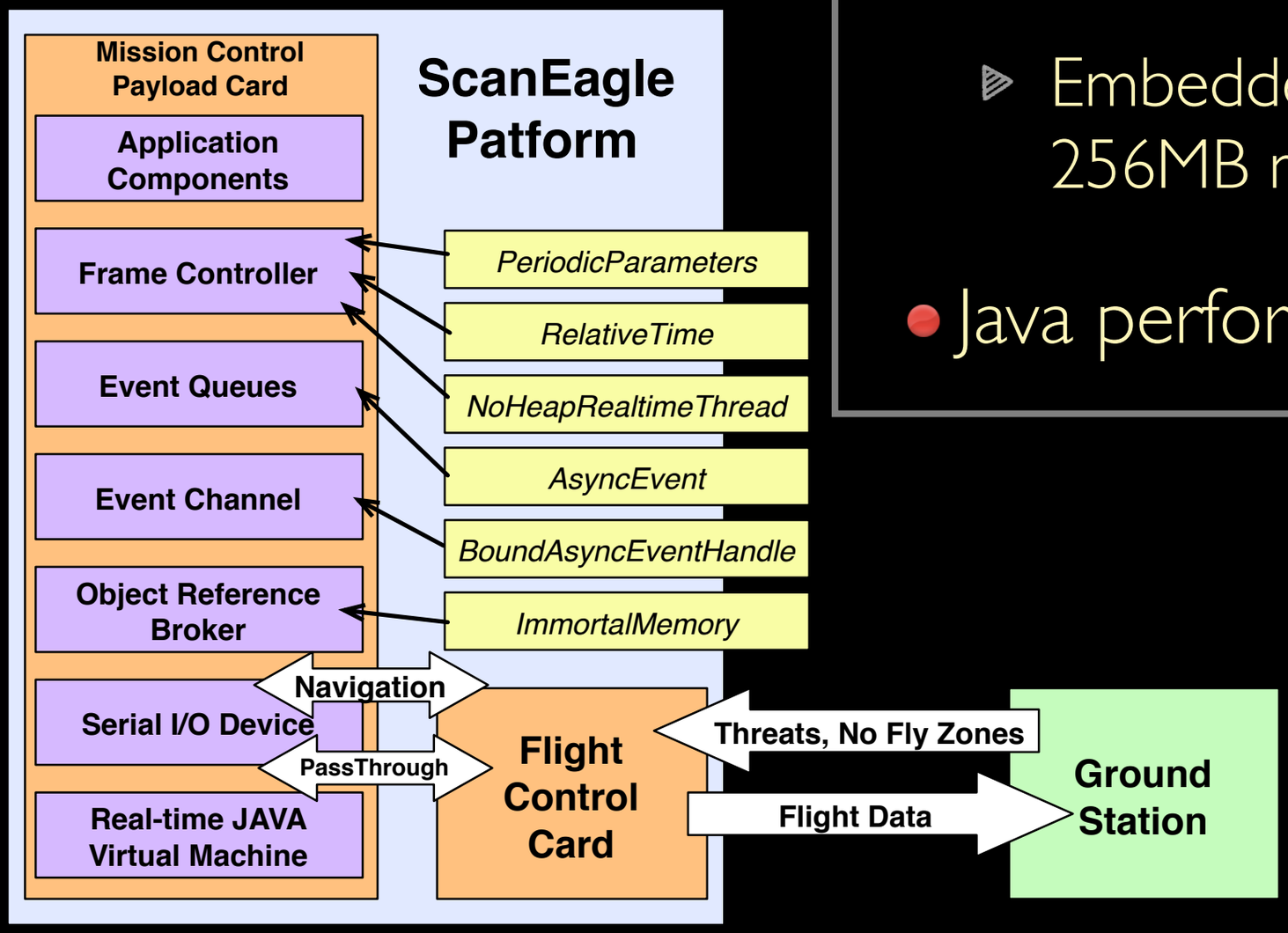
Ovm

- Started in 2001, in a DARPA funded project.
- Developed the Ovm virtual machine framework, a clean-room, open source RT Java virtual machine.
- Fall 2005, first flight test

ScanEagle

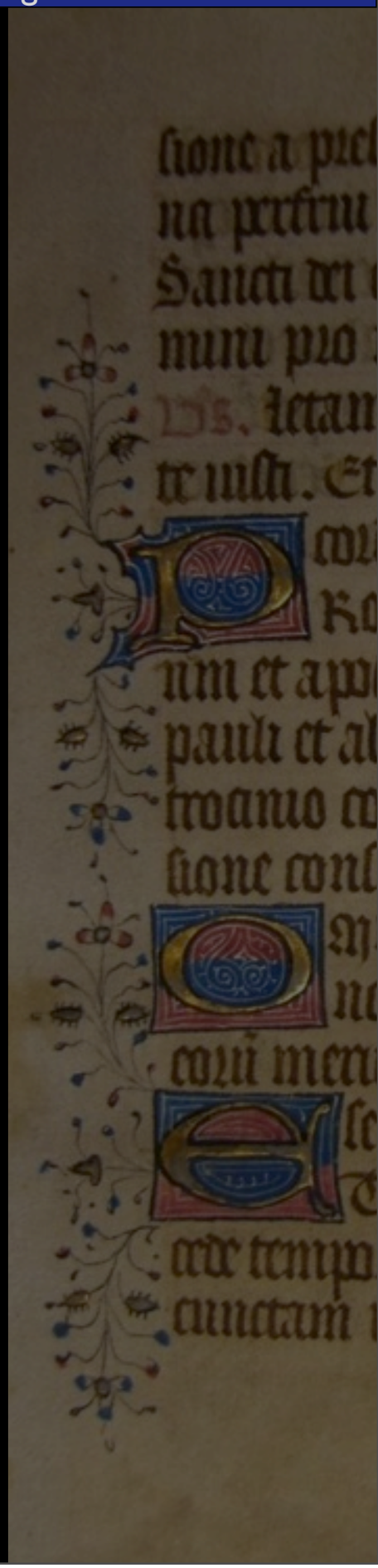
ScanEagle

- 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

- 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
- 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

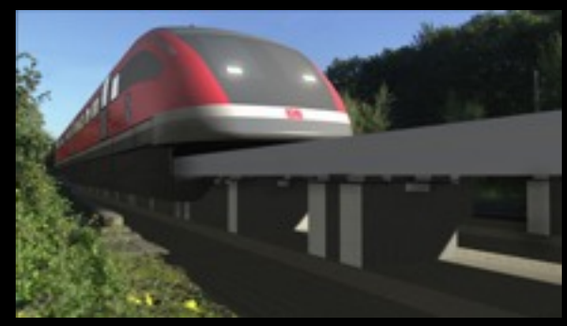


2

Safety Critical Java

Problems to be solved

- In a real time system it is mission-critical that results be available in a given time frame. If the deadline is missed, for:
  - *soft real time*: Loss of quality
    - video streaming, circuit switching
  - *hard real time*: Engine failure or damage
    - robot control
  - *safety critical*: Human beings may die
    - aviation, military, medicine, power plants



SC Java Goal

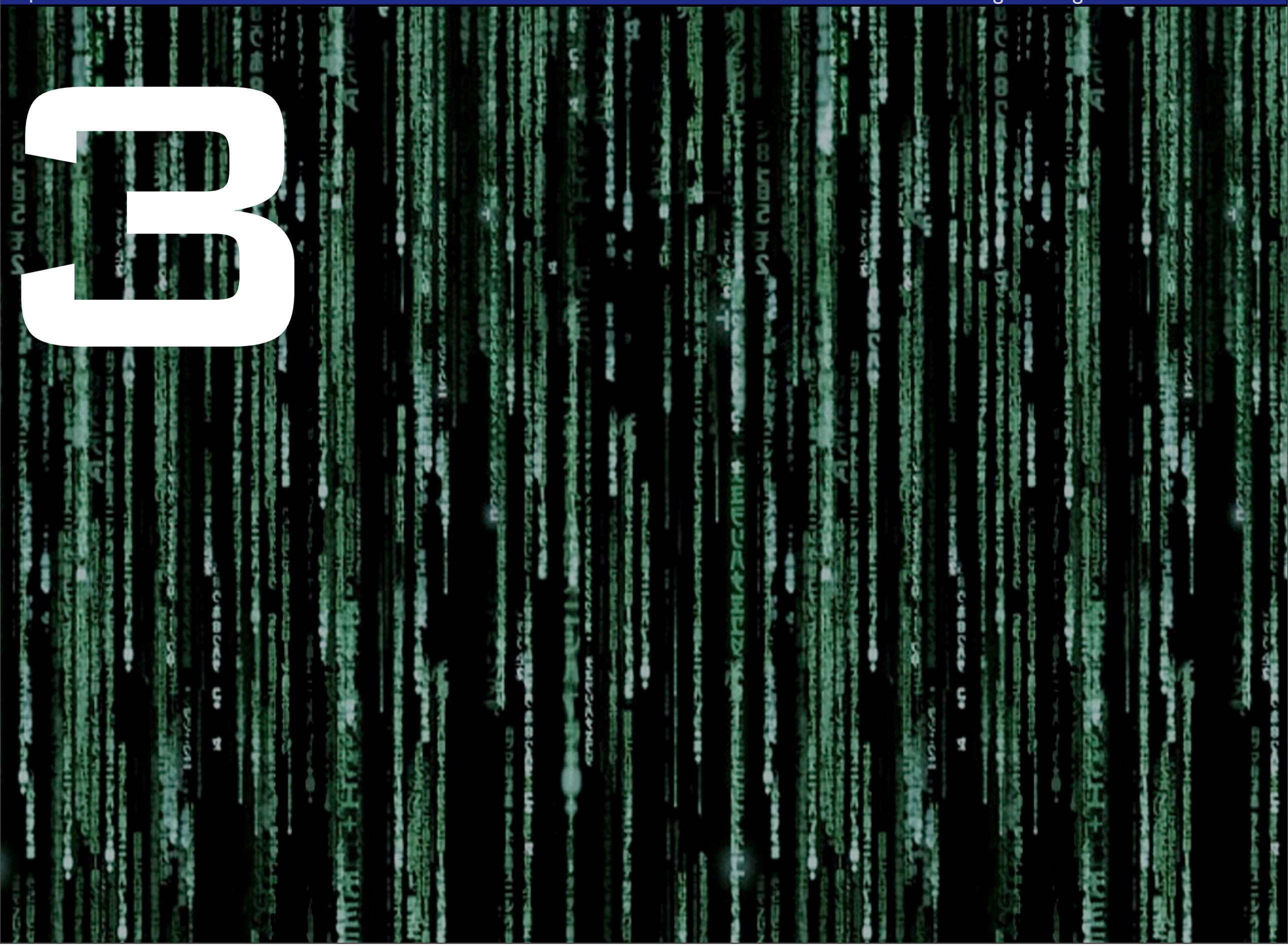
- A specification for Safety Critical Java capable of being certified under **DO-178B Level A**
  - Implies small, reduced complexity infrastructure (i.e. JVM)
  - Emphasis on defining a minimal set of capabilities required by implementations

SCJ Compliance level

- Level 0
  - Single threaded, single mission, single core
- Level 1
  - *Multi threaded, multi mission, multi core*
- Level 2
  - Multi threaded, multi *nested* mission, multi core

SC Java

- excluded:
    - Heap allocation
    - Class loader
    - Finalizers
    - Priority Inheritance (not required)
- supported:
    - Raw Memory (atomic) (no endianess change, no floating point)
    - Priority Ceiling for priority inversion management
    - Support for SMPs to follow RTSJ lead


3

Fiji VM

- 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

RTEMS demo

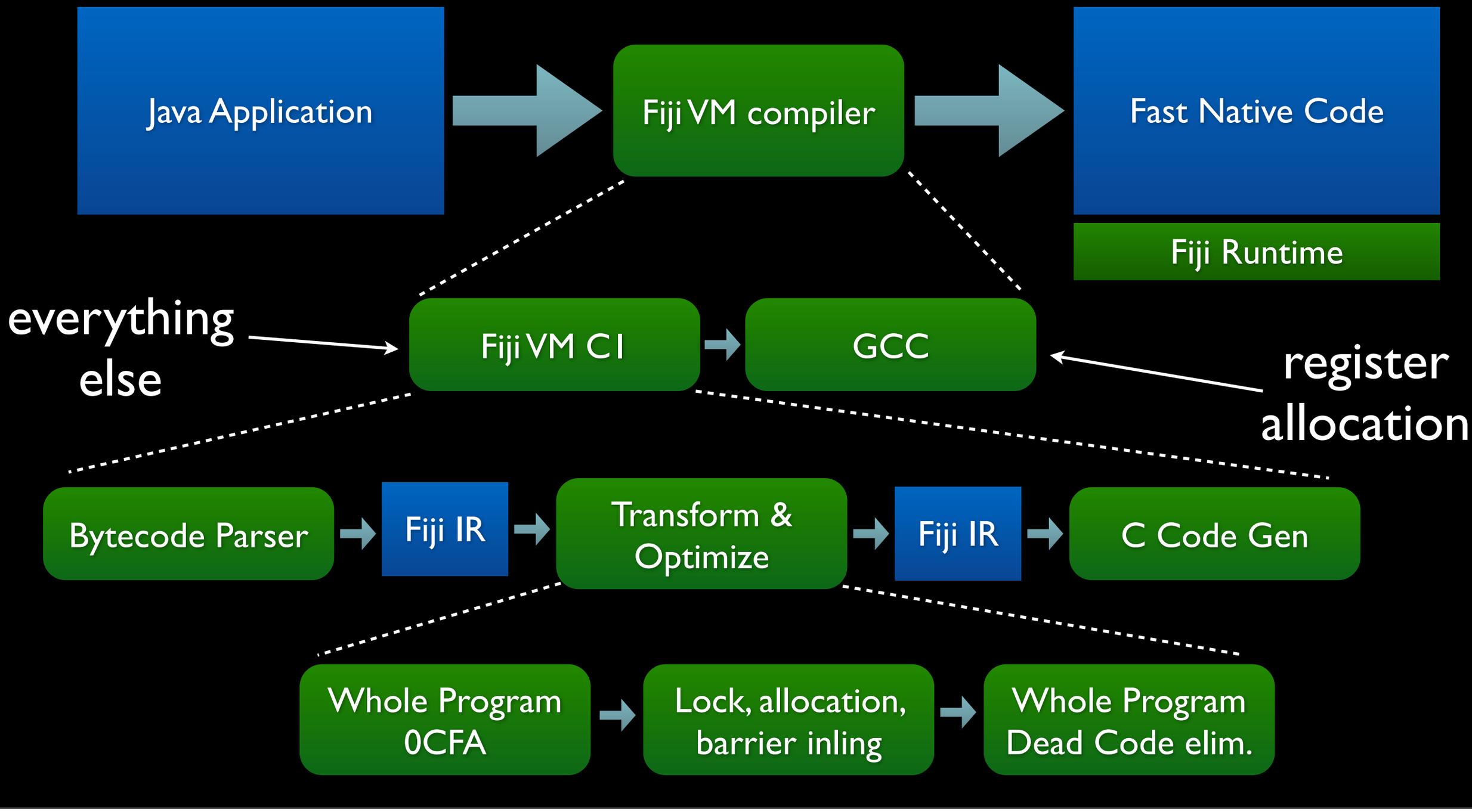
- fVM runs on RTEMS 4.9.2
  - Java threads run side-by-side with RTEMS C, C++, Ada threads
- Repeat every 10 ms
  - Allocate 1000 Integer instances
  - Allocate Integer[1000] array, fill with Integer instances
- Run code as an RTEMS interrupt handler
  - *fVM's Java runtime is robust enough to allow pure Java code to run in an interrupt context while using all of Java's features*

```
class Demo {
 static Integer[] arr; static int iter, iWGC;
 static long mDWoGC, mDWGC;
 public static void main(String[] v) {
   final Timer t=new Timer();
   t.fireAfter(10,new Runnable(){
    public void run() {
     long before=HardRT.readCPUTimestamp();
     iter++; if (GC.inProgress()) iWGC++;
     if (arr==null) {
        arr=new Integer[100000];
       for (int i=0;i<arr.length;++i) arr[i] = new Integer(i);
     } else
      for (int i=0;i<arr.length;++i)
          if (!arr[i].equals(new Integer(i))) throw new Error("failed "+i);
     t.fireAfter(10,this);
     long diff = before-HardRT.readCPUTimestamp();
     if (GC.inProgress()){
        if (diff>mDWGC) mDWGC = diff;
     } else if (diff>mDWoGC) mDWoGC = diff;
   }});
   for (;;) {
     String res = "Number of timer interrupts: "+iter +
                  "\n          when GC running: "+iWGC +
                  "\n  Max  exec time with GC: "+ mDWGC);
     System.out.println(res); Thread.sleep(1000);
} } }
```

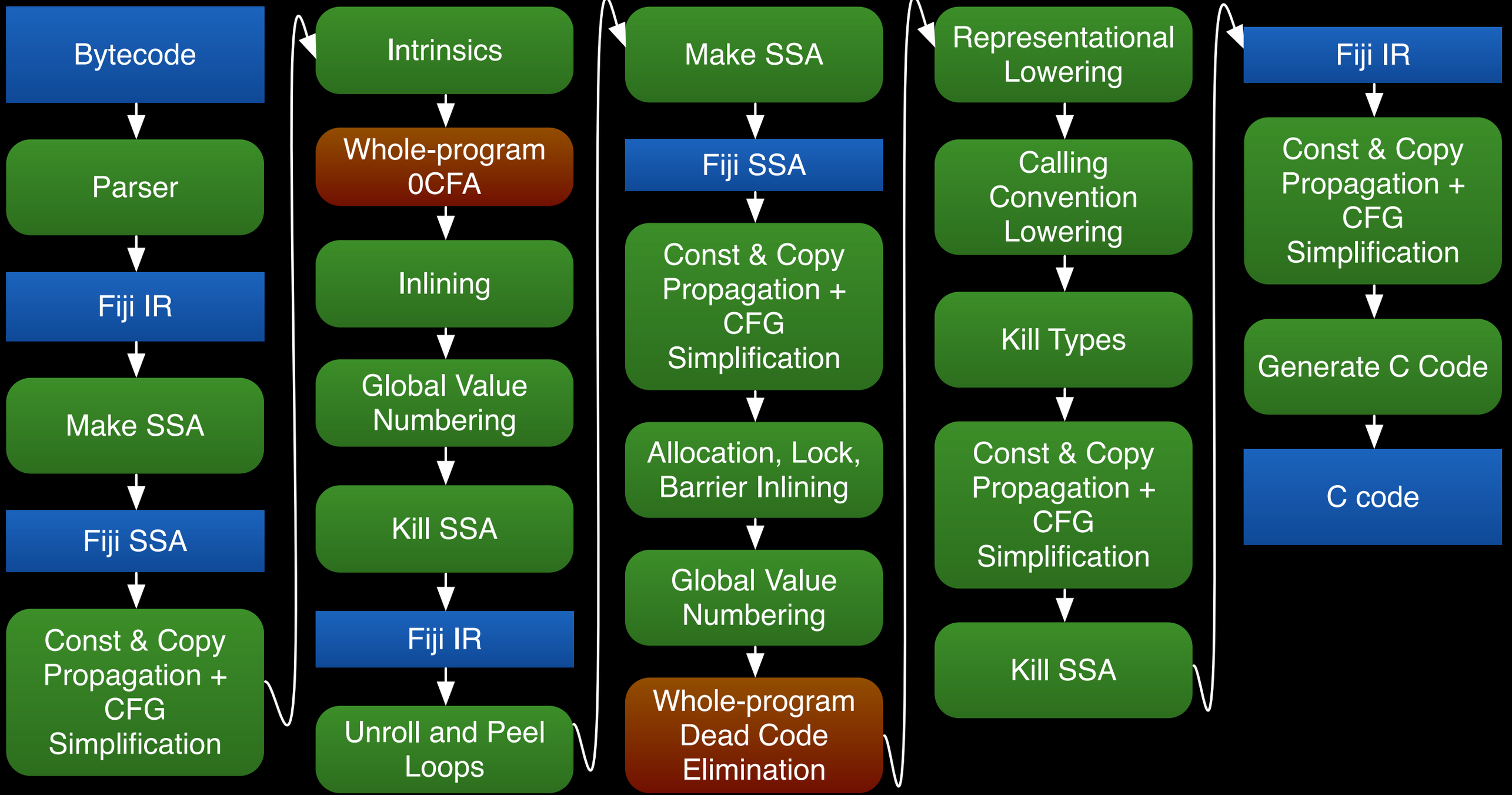
```
final Timer t = new Timer();
t.fireAfter(10, new Runnable(){
    public void run() {
     before=HardRT.getCPUTimestmp();
     if (GC.inProgress()) iWGC++;
     arr = new Integer[1000];
     for (int i=0;i<arr.length;++i)
          arr[i] = new Integer(i);
     t.fireAfter(10, this);
     diff=before-HardRT.readCPUTimestmp();
      if (GC.inProgress())
          mDWGC=(diff>mDWGC)?diff:mDWGC;
     else mDWoGC=(diff>mDWoGC)?diff:mDWoGC;
}});
```

```
t = new Timer();
t.fireAfter(10,
 new Runnable(){ void run(){
    long before=getCPUTimestmp();
    if (GC.inProgress()) iWGC++;
    arr = new Integer[1000];
    for(int i=0;i<arr.length;++i)
       arr[i] = new Integer(i);
    t.fireAfter(10, this);
    ...
}});
```

VM Overview

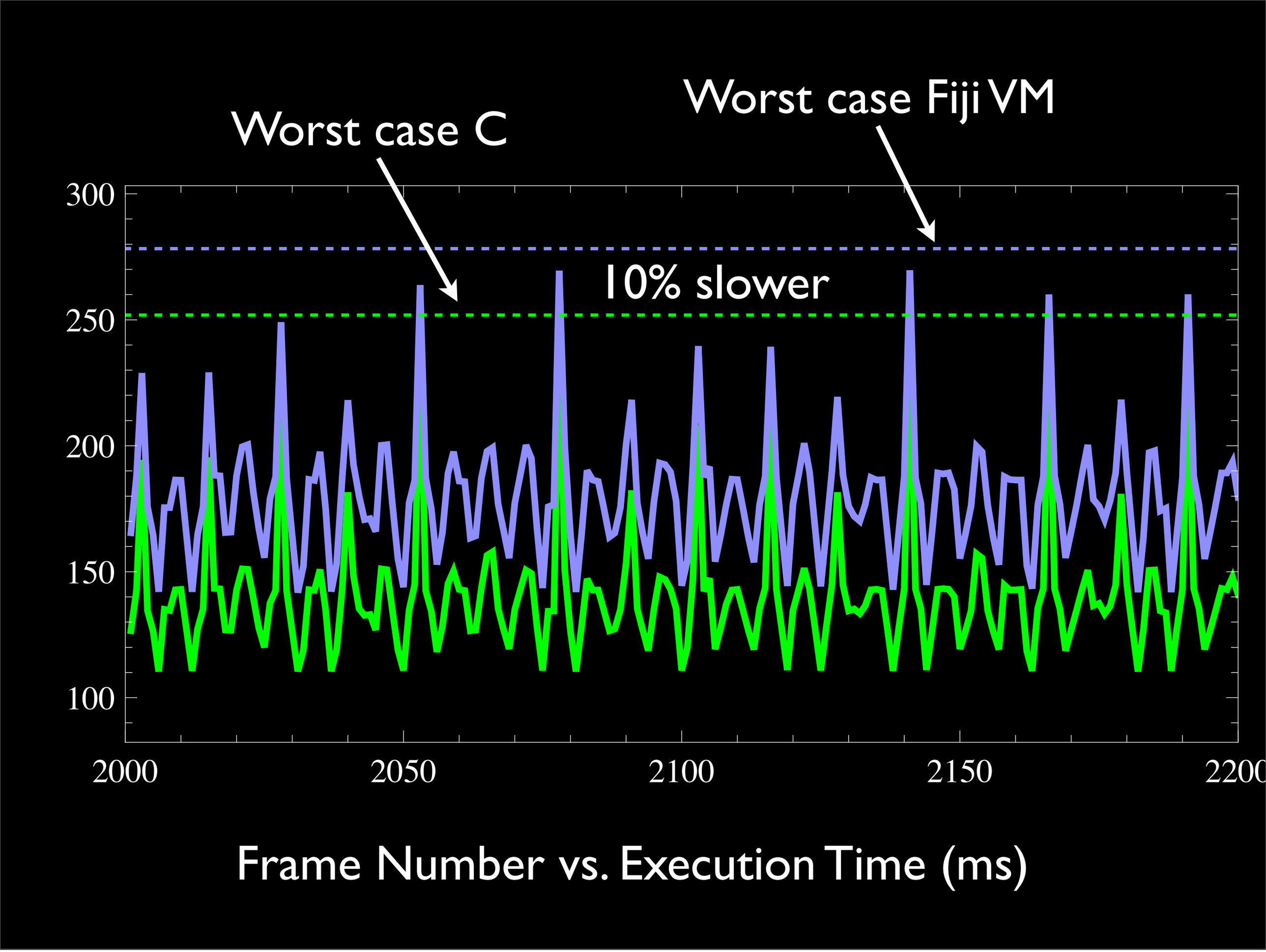

Java Application
Fiji VM compiler
Fast Native Code
Fiji Runtime
everything else
Fiji VM C1
GCC
register allocation
Bytecode Parser
Fiji IR
Transform & Optimize
Fiji IR
C Code Gen
Whole Program 0CFA
Lock, allocation, barrier inling
Whole Program Dead Code elim.

Better view


Bytecode
Parser
Fiji IR
Make SSA
Fiji SSA
Const & Copy Propagation + CFG Simplification
Intrinsics
Whole-program 0CFA
Inlining
Global Value Numbering
Kill SSA
Fiji IR
Unroll and Peel Loops
Make SSA
Fiji SSA
Const & Copy Propagation + CFG Simplification
Allocation, Lock, Barrier Inlining
Global Value Numbering
Whole-program Dead Code Elimination
Representational Lowering
Calling Convention Lowering
Kill Types
Const & Copy Propagation + CFG Simplification
Kill SSA
Fiji IR
Const & Copy Propagation + CFG Simplification
Generate C Code
C code

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
- Representative Real-time platform
  - RTEMS 4.9.1 (hard RTOS: no processes or virtual memory)
  - 40MHz LEON3 with 64MB RAM (radiation-hardened SPARC)
- A platform used by ESA and NASA


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

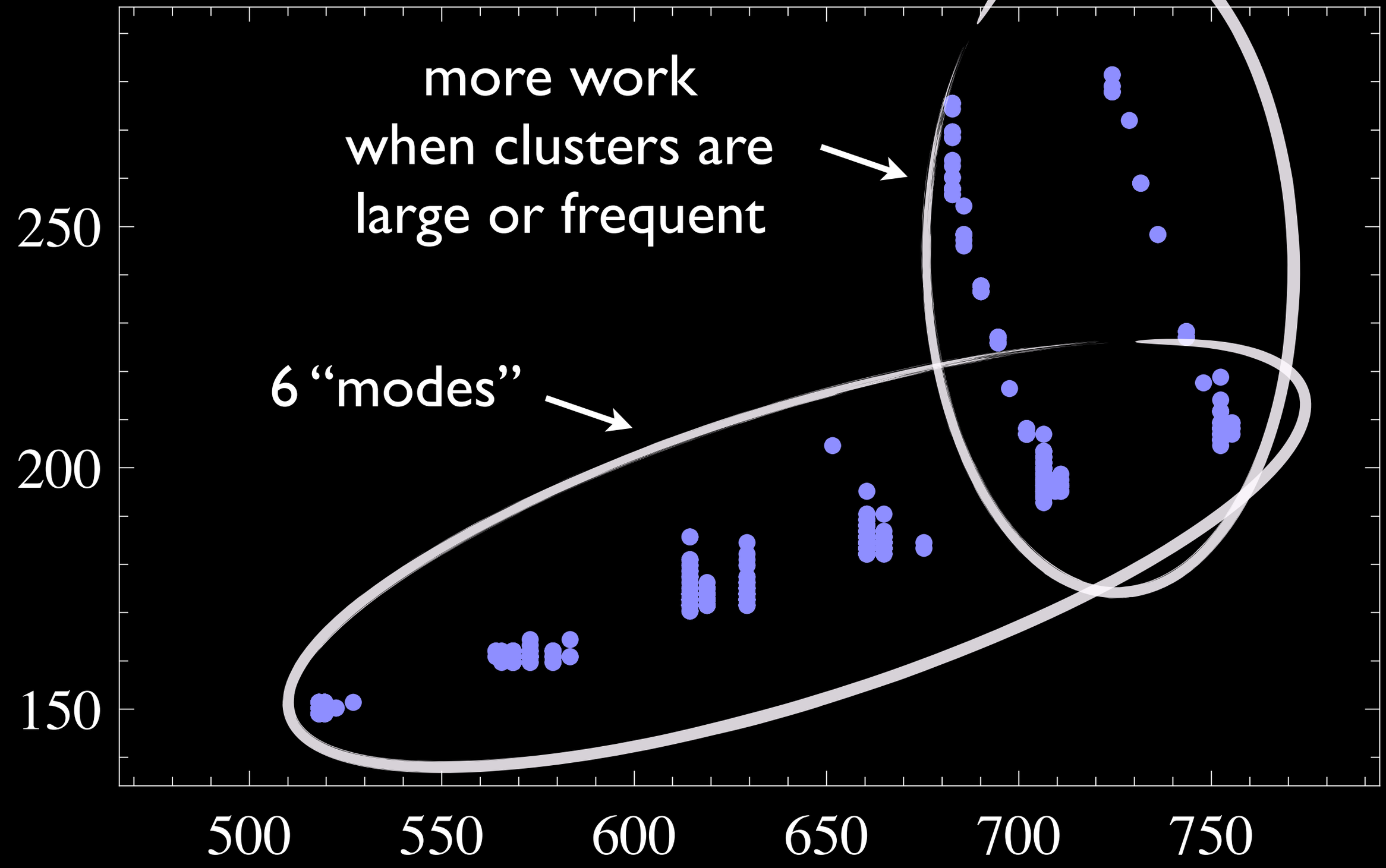
Source of overheads

measured using RTBx data logger

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

www.rapitasystems.com

Type Checks



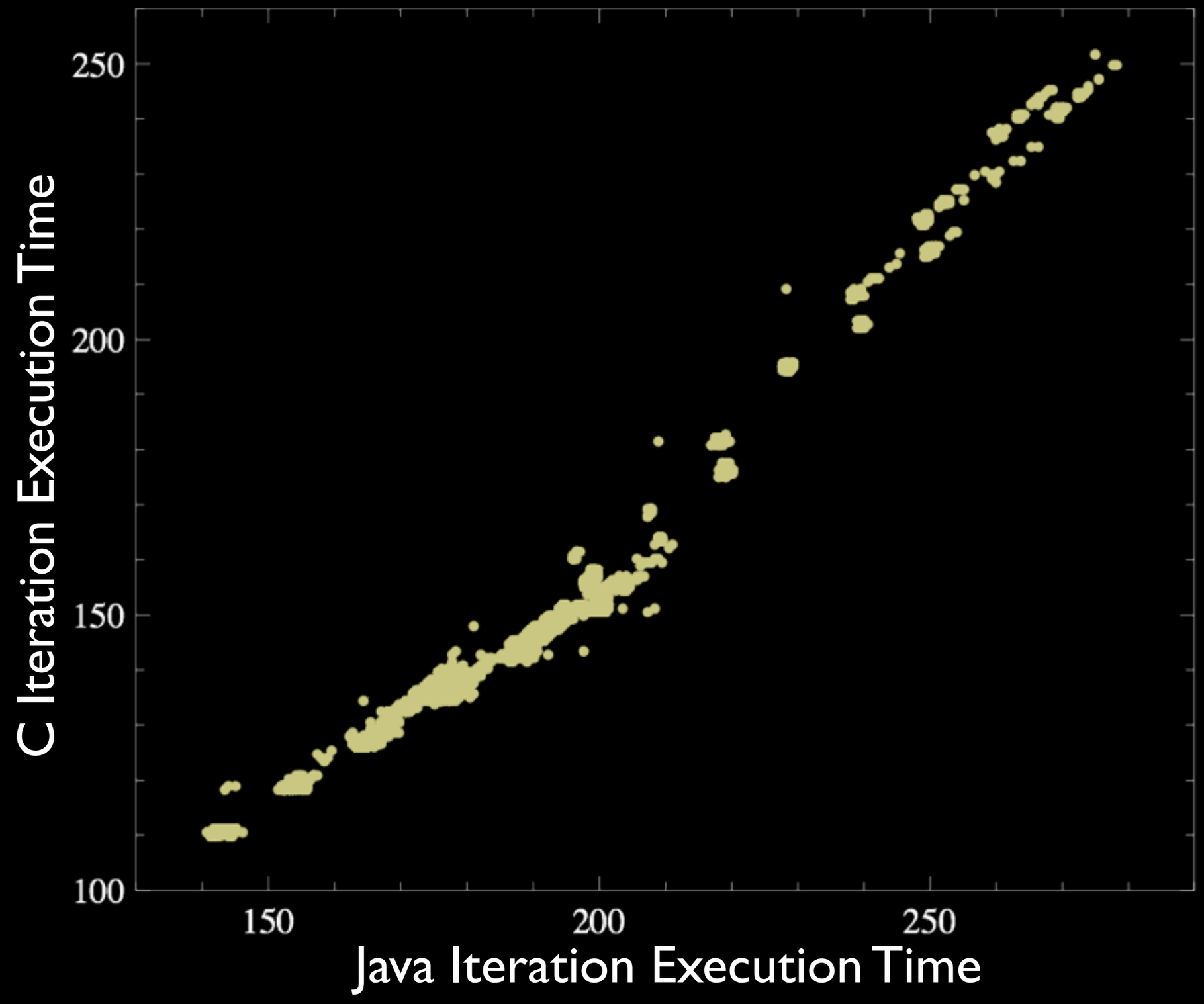
Number of Checks correlated against execution time

Correlation Java vs C on RTEMS/LEON3

15 GC collection cycles!

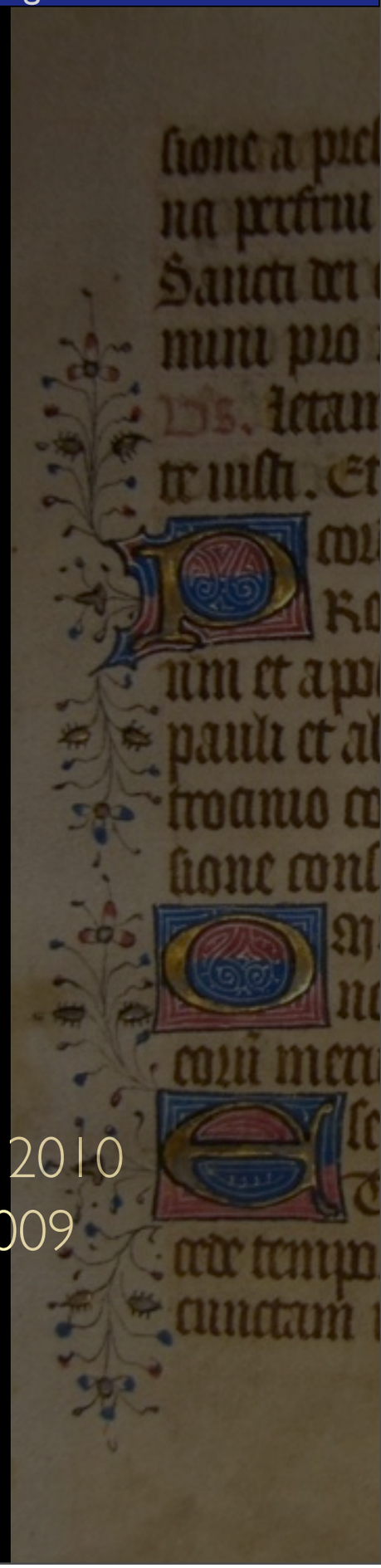
10,000 samples

no outliers



References and acknowledgements

- Team
  - *F. Pizlo, L. Ziarek, T. Kalibera, D. Tang, L. Zhao*
- **Funding:** *NSF, Fiji Systems LLC*
- **Availability:** *to be GPLed for research*
- Paper trail
  - *High-level Programming of Embedded Hard Real-Time Devices.* EUROSYS, 2010
  - *Real-time Java in Space: Potential Benefits and Open Challenges.* DASIA, 2009
  - *A Technology Compatibility Toolkit for Safety Critical Java.* JTRES, 2009



4


60
55
5
50
10
40
20
30
50
10
45
15
40
20
SHOCKRESISTANT
7 JEWELS
35
30
25
SWISS
MADE

Memory management and programming models

- The choice of memory management affects productivity
- Object-oriented languages hide allocation behind abstractions
  - Manual de-allocation more difficult in OO style
- Concurrent algorithms emphasize allocation
  - because freshly allocated data is guaranteed to be thread local
- … but garbage collection is a global, costly, operation that introduces unpredictability

Alternative 1: No Allocation

- No allocation means the GC does not run

Alt 2: Allocation in Scoped Memory

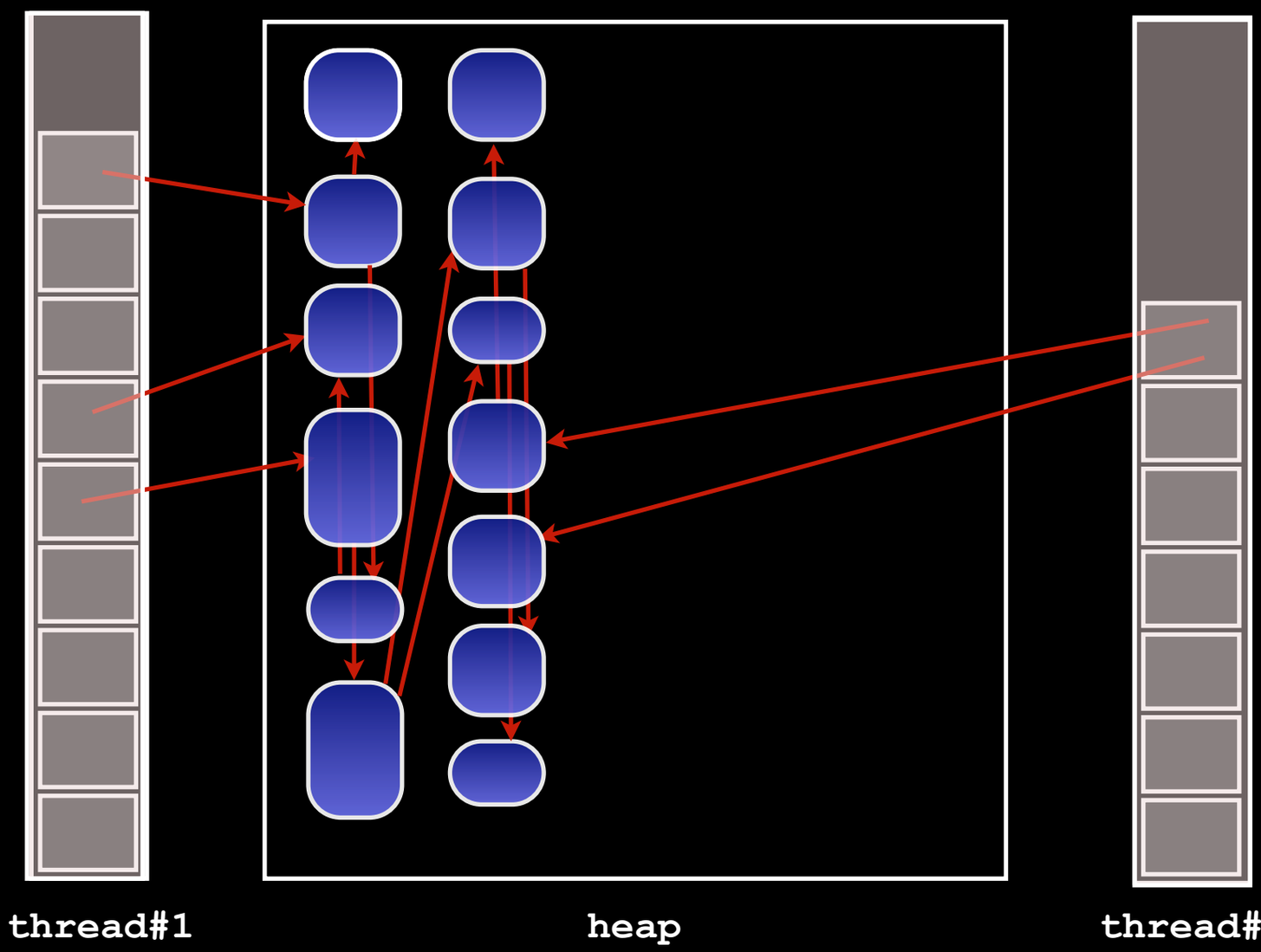
- RTSJ provides scratch pad memory regions for temporary allocation
  - Used but tricky as they can cause exceptions

```
s = new SizeEstimator();
s.reserve(Decrypt.class, 2);
…
shared = new LTMemory(s.getEstimate());
shared.enter(new Run(){ public void run(){
    ...d1 = new Decrypt() ...
}});
```

Alt 3: Real-time Garbage Collection

- There are three main families of RTGC implementations
- **Work-based**
  - *Aicas JamaicaVM*
- **Time-triggered, periodic**
  - *IBM Websphere*
- **Time-triggered, slack**
  - *SUN Java Real Time System*

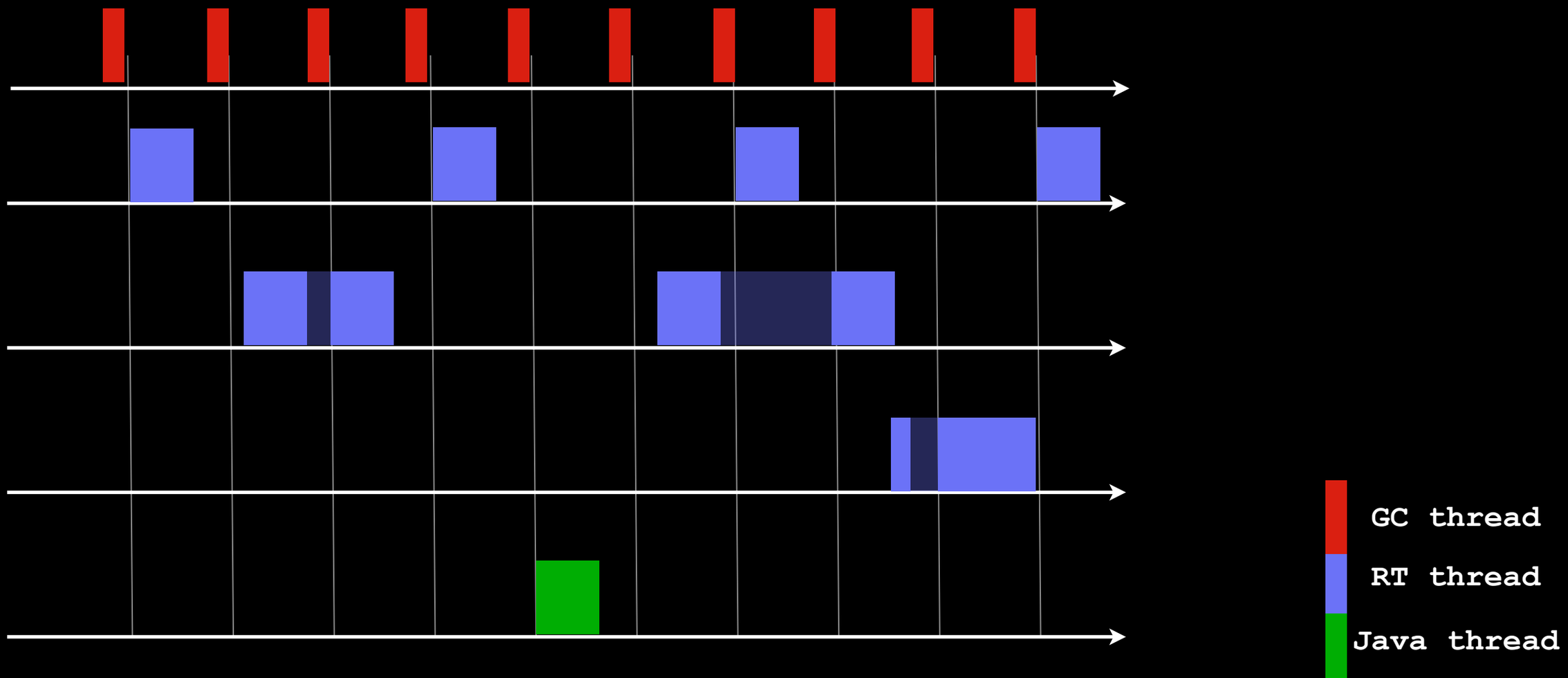
Garbage Collection



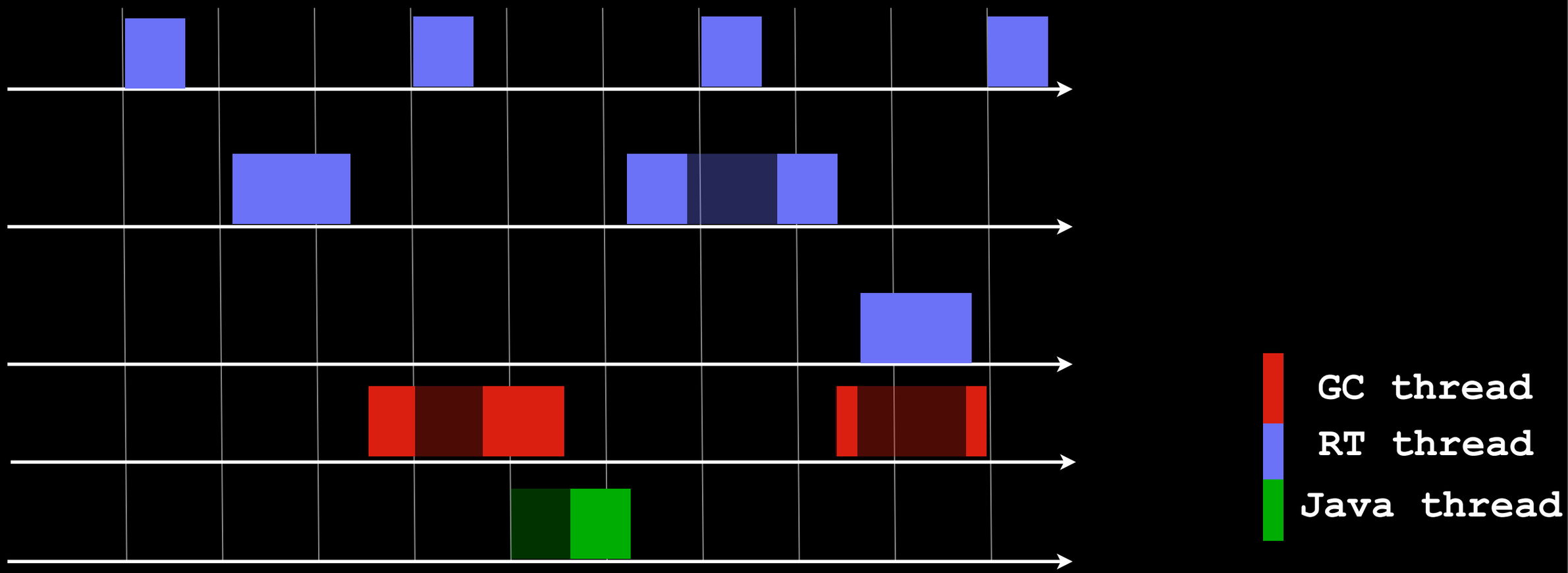
Phases

- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction

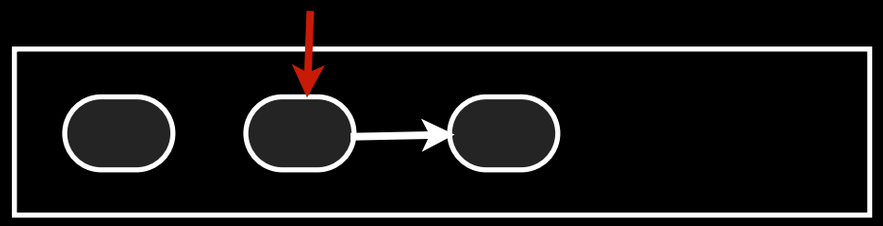
Time-based GC Scheduling

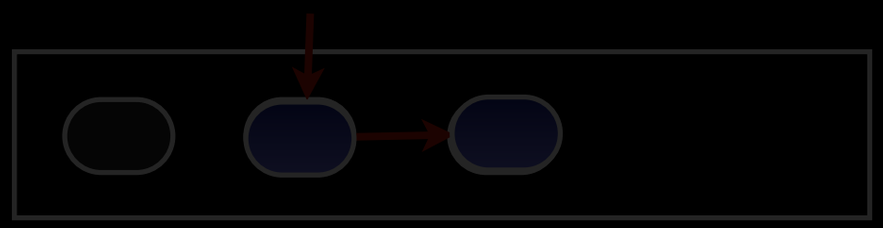


Slack-based GC Scheduling



Incrementalizing marking





Collector marks object

Application updates reference field

Compiler inserted write barrier marks object

Sch

ism

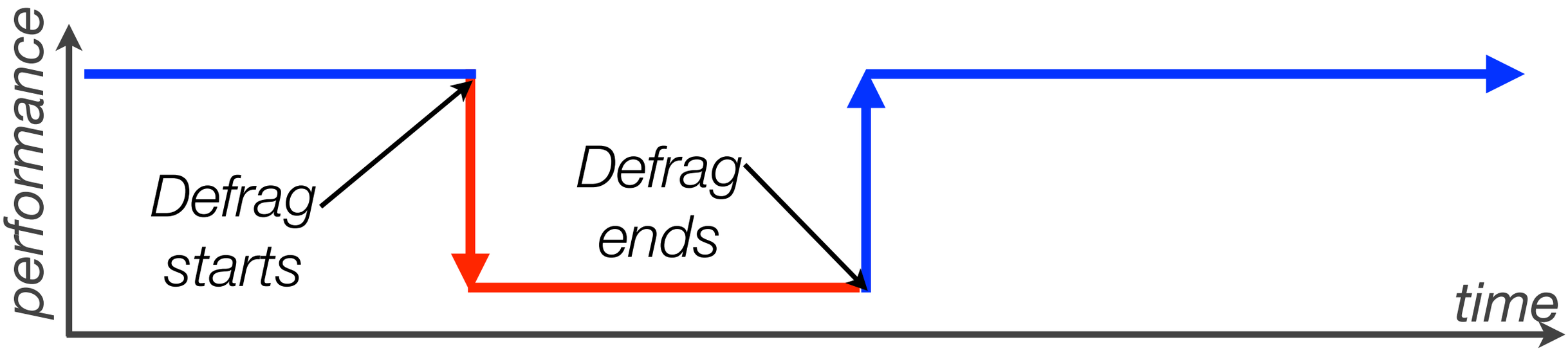
Real Time Garbage Collection

- **HotSpot** - fast & space bounded
  - *but*: **blocking**
- **Java RTS**: space bounds, concurrent, wait-free
  - *but*: **60% slow-down**, **logarithmic** heap access
- **IBM SRT**: 30% slow-down, concurrent, wait-free
  - *but*: **susceptible to fragmentation**

Minimizing Fragmentation

On-demand Defragmentation

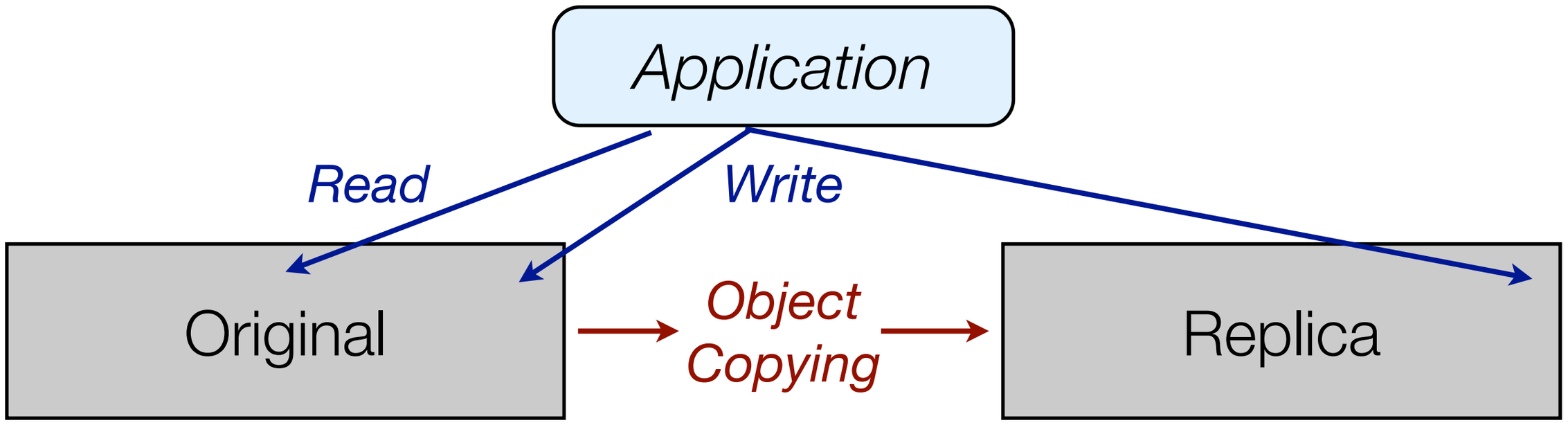
- Stop-the-world/incremental: simple, but causes pauses
- Concurrent: still has draw-backs
  - slow down during defrag more than 5x [Pizlo07,Pizlo08]



Replication-based GC

- [NettlesOToole93,ChengBlelloch01]
- Allows concurrent defragmentation
- Two spaces: one space for reads; writes “replicated” to both
- Problem: Writes not atomic!

Works best for immutable objects.

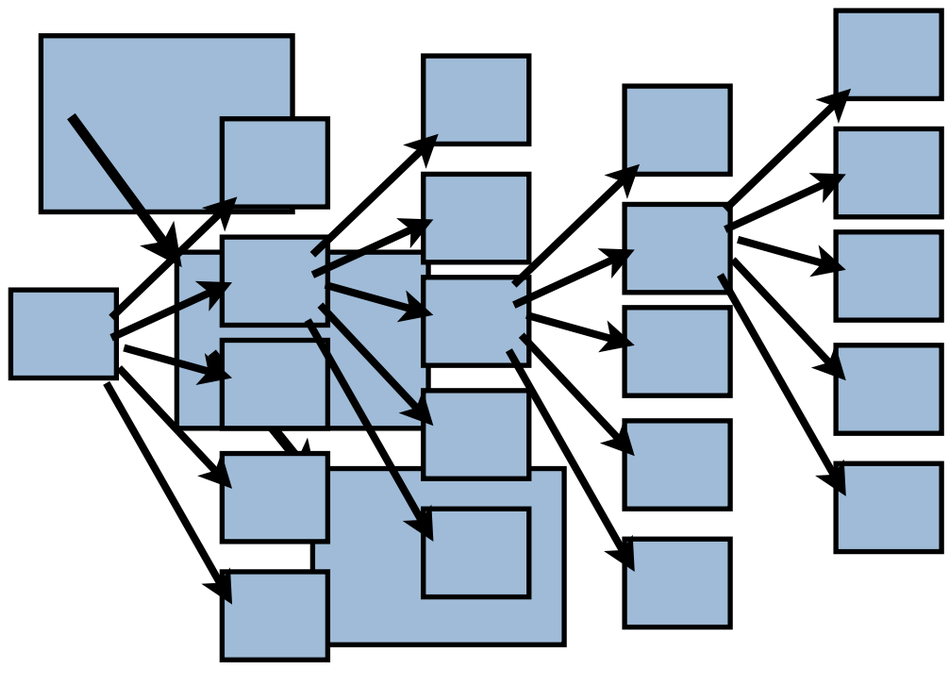


Allocate in fragments [Siebert ’99]

- All objects split into small fragments
- Fragment size is typically fixed at 32 bytes
- Fragments are linked, application follows links on reads

Plain Object

Array



Access cost is logarithmic.

Access cost is known statically, does not vary.

Most objects require only two fragments.

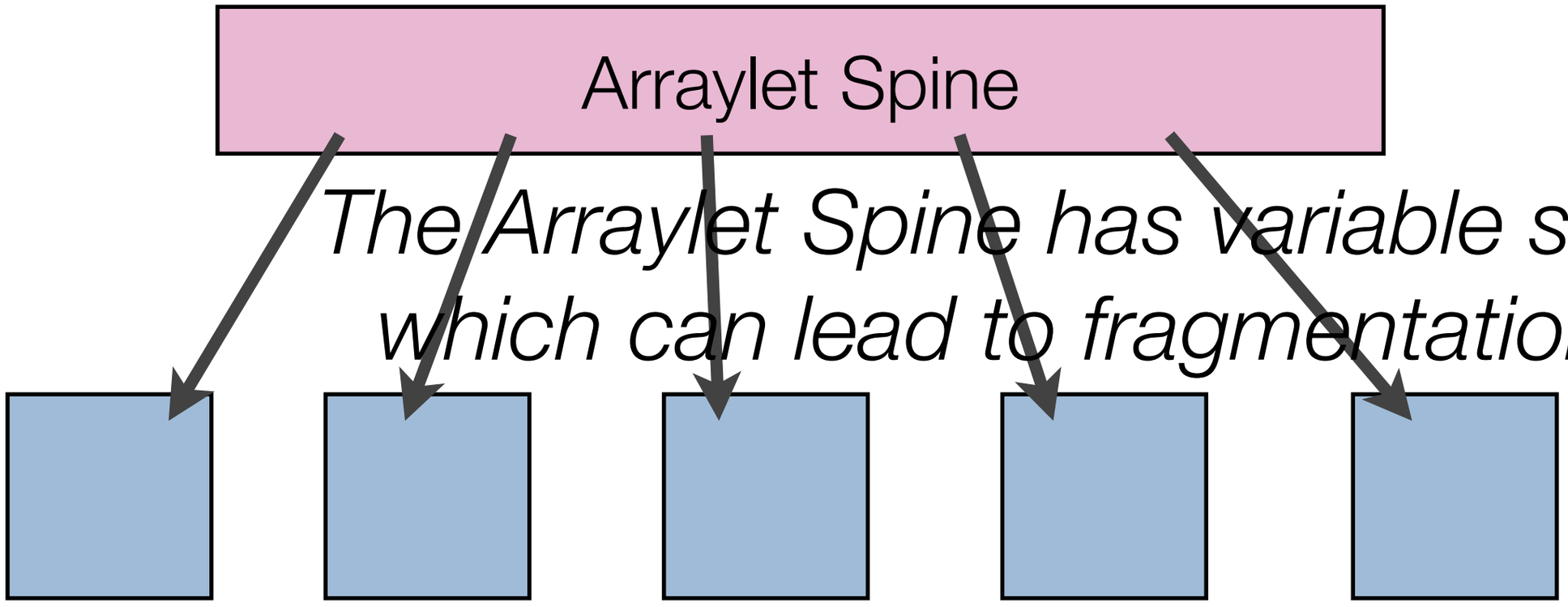
Synopsis

- Replication-copying Collection:
  - *good for objects that are not mutated*
- Fragmented Allocation:
  - *good for small objects*

Combine the two?

A new way of exploiting Arraylets

But the spine is immutable ...
... and <u>replication</u> is ideal for immutable objects



The Arraylet Spine has variable size, which can lead to fragmentation!

Fixed size fragments - no external fragmentation

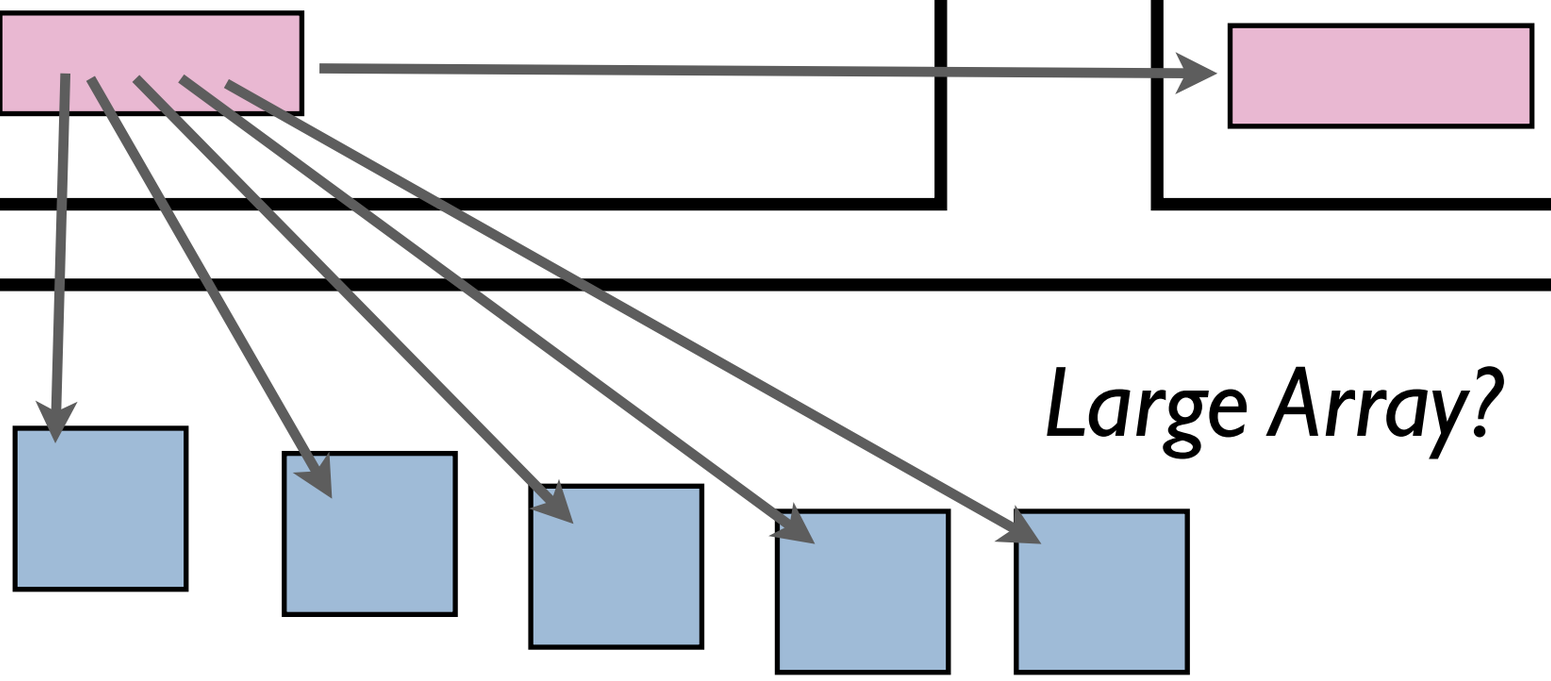
Schism = arraylets + replication + fragments

- Combination:
  - Concurrent **mark-sweep GC** for fixed-size **fragments**
  - **Replication copying** for variable-size **arraylet** spines
- *No external fragmentation*
- *Heap access is O(1), wait-free, and coherent.*

Concurrent Replication Heap for Spines

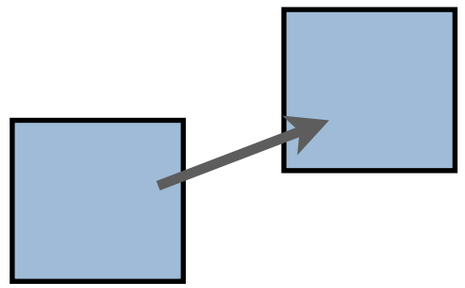
To-space for Spines

From-space for Spines



Large Array?

Small Object

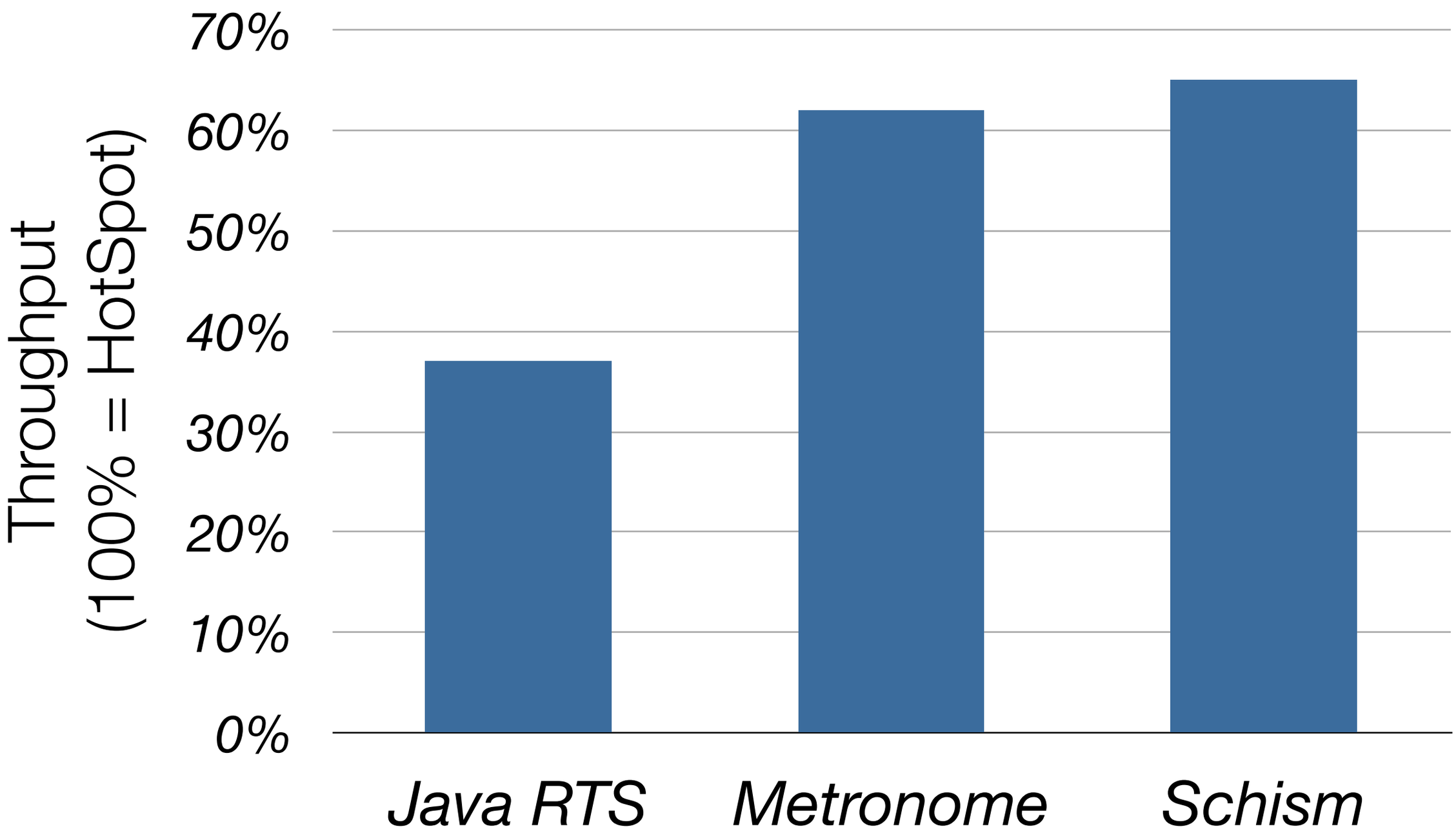


Concurrent Mark-Sweep Heap for Fragments

Results

- Goal: as fast as Metronome
- Goal: fragmentation tolerant like Java RTS
- Goal: deterministic

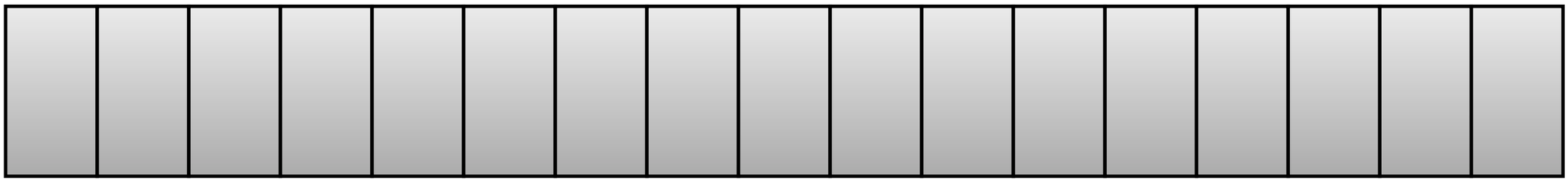
SPECjvm98 throughput summary



Results

- Goal: as fast as Metronome ✓
- **Goal: fragmentation tolerant like Java RTS**
- Goal: deterministic

Fragger Results



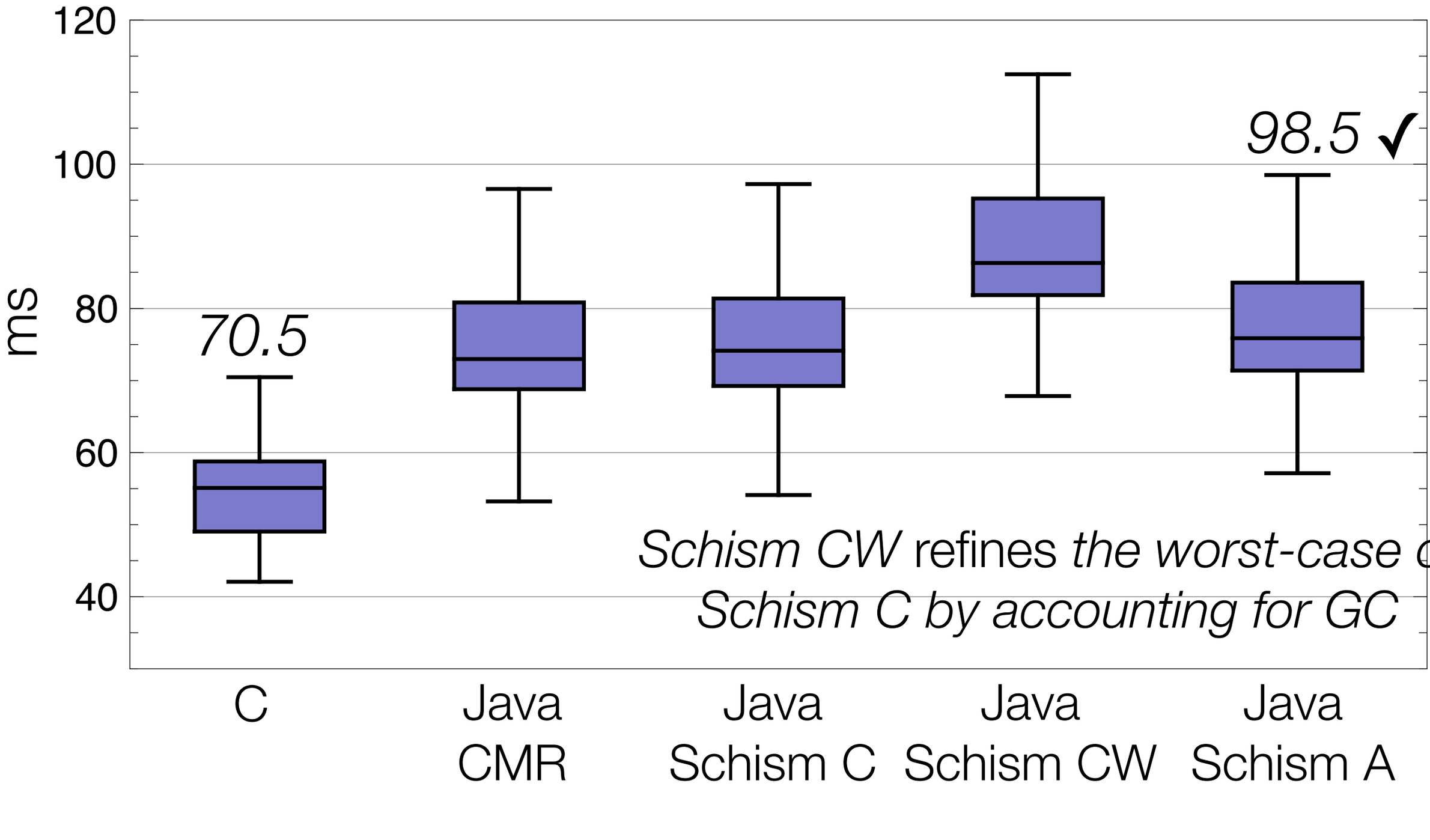
- Amount of free memory successfully allocated under fragmentation:
  - *HotSpot*: ~**100%**
  - *Java RTS*: ~**80%**
  - *Metronome*: ~**1%**
  - *Schism*: ~**100%**

Results

- Goal: as fast as Metronome ✓
- Goal: fragmentation tolerant like Java RTS ✓
- **Goal: deterministic**

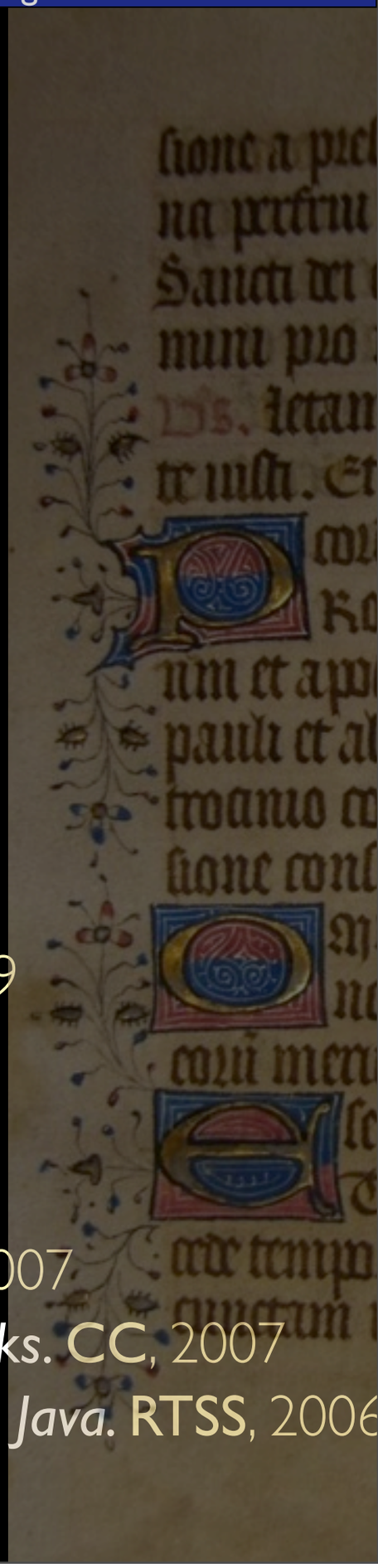
Java versus C on CDx

Schism A is completely determinis
no further refinement necessary



References and acknowledgements

- Team
  - *F. Pizlo, E. Blanton, L. Ziarek*
    T. Kalibera, T. Hosking, P. Maj, T Cunei, M Prochazka, J Baker
- **Funding:** *NSF*



- Paper trail
  - *Schism: Fragmentation-Tolerant Real-Time Garbage Collection.* **PLDI**, 2010
  - *Accurate Garbage Collection in Uncooperative Environments.* **CC:P&E**, 2009
  - *Memory Management for Real-time Java: State of the Art.* **ISORC**, 2008
  - *Garbage Collection for Safety Critical Java.* **JTRES**, 2007
  - *Hierarchical Real-time Garbage Collection.* **LCTES**, 2007
  - *Scoped Types and Aspects for Real-time Java Memory management.* **RTS**, 2007
  - *Accurate Garbage Collection in Uncooperative Environments with Lazy Stacks.* **CC**, 2007
  - *An Empirical Evaluation of Memory Management Alternatives for Real-time Java.* **RTSS**, 2006
  - *Real-Time Java scoped memory: design patterns, semantics.* **ISORC**, 2004

References

- Realtime Specification for Java:
  - `http://www.rtsj.org`
- Safety Critical Java:
  - `JSR-302 http://jcp.org`
- Fiji VM:
  - `http://www.fiji-systems.com`
- Ovm:
  - `http://www.cs.purdue.edu/homes/jv`

Challenge problems

- Verifying the infrastructure
  - Memory models (e.g. Sewell ea)
  - RTOS (e.g. seL4)
  - Compiler (e.g. Compcert)
  - Garbage collector
- Verifying the programs
  - WCET analysis (e.g. aIT)
  - Software errors (e.g. Astree)
  - Specification languages (e.g. JML, Spec#)
- Better languages