Programming Models for Concurrency and Real-time

Jan Vitek

Purdue University

Fiji Systems LLC

IBM

Outline

0: Real-time and embedded systems

1: Real-time Java with Ovm

2: Memory management with Minuteman

3: Low latency programming with Flexotasks

4: Java in aerospace with the Fiji VM

5: Conclusion

What is a real-time system?

- A real-time system is any information processing system which has to respond to externally generated input stimuli within a finite and specified period
  - *correctness depends not only on logical result but also time it is delivered*
  - *failure to respond as bad as a wrong response*

What is an embedded system?

- Computer that is part of some *other* piece of equipment
  - Usually dedicated software
  - Often no "real" keyboard or general purpose display
- … we use 100+ embedded computers daily
- … embedded hardware growth rate of 14% to reach $78 billion

http://www.bccresearch.com/comm/G229R.html, http://www.ecpe.vt.edu/news/ar03/embedded.html

Characteristics of 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
- **Good news**
  - programming languages, such as Java and C#, are wrestling control from the lower layers of the stack and impose well-defined semantics (on threads, scheduling, synchronization, memory model)

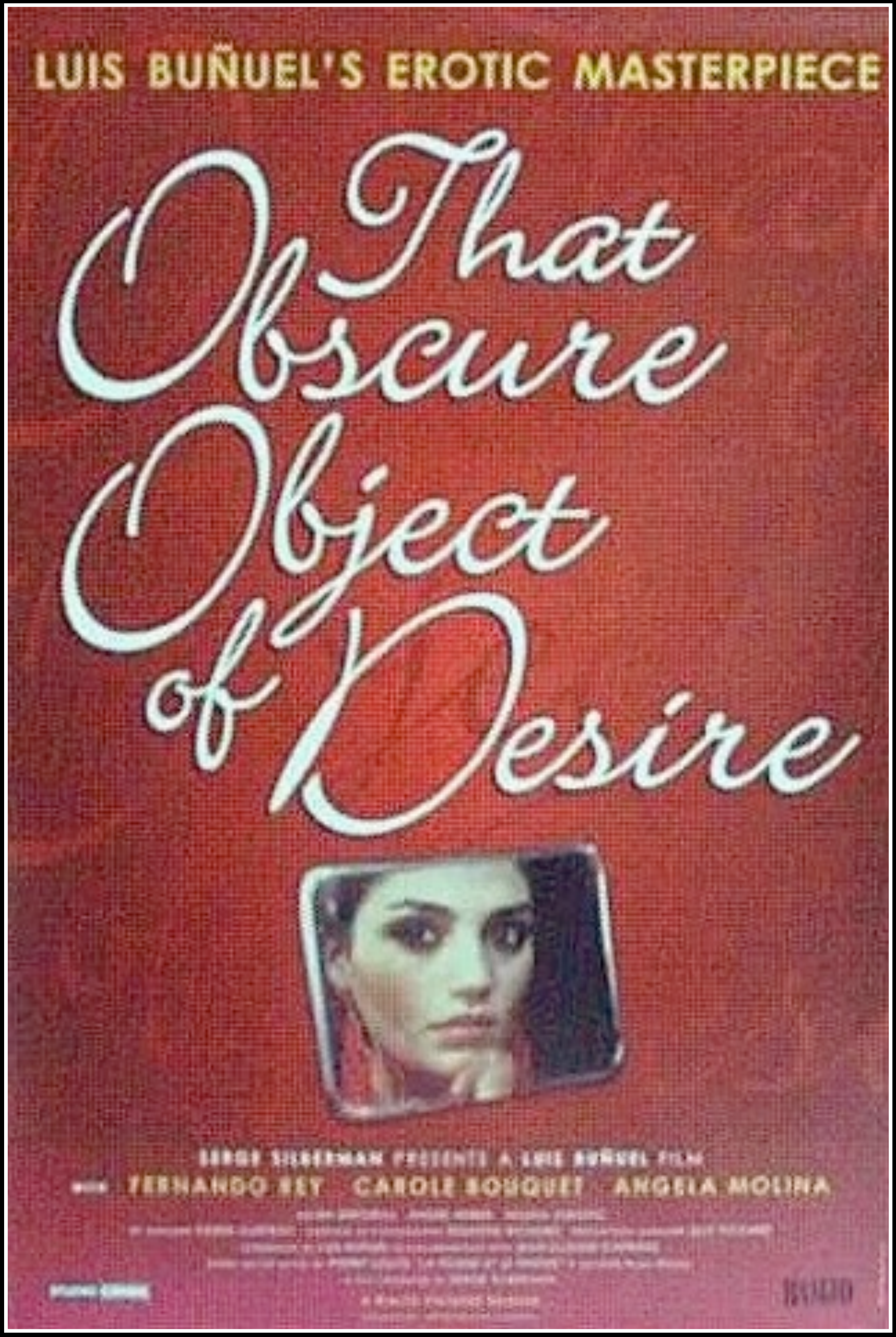
What programming model?

- "Real-time systems require fine grained control over resources and thus the language of choice is C, 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 models?

- There are many dimensions:
  - Imperative vs. Functional
  - Shared memory vs. Message passing
  - Explicit lock-based synchronization vs. Higher-level abstractions (data-centric synchronization, transactional memory)
  - Time-triggered vs. synchronous / logic execution time
- And multiple languages, systems:
  - C, C++, Ada, SystemC, Assembler, Erlang, Esterel, Lustre, Giotto …

Are object oriented technologies the silver bullet for the real-time software crisis?




1

Ovm

The Real-time Java experience

Java?

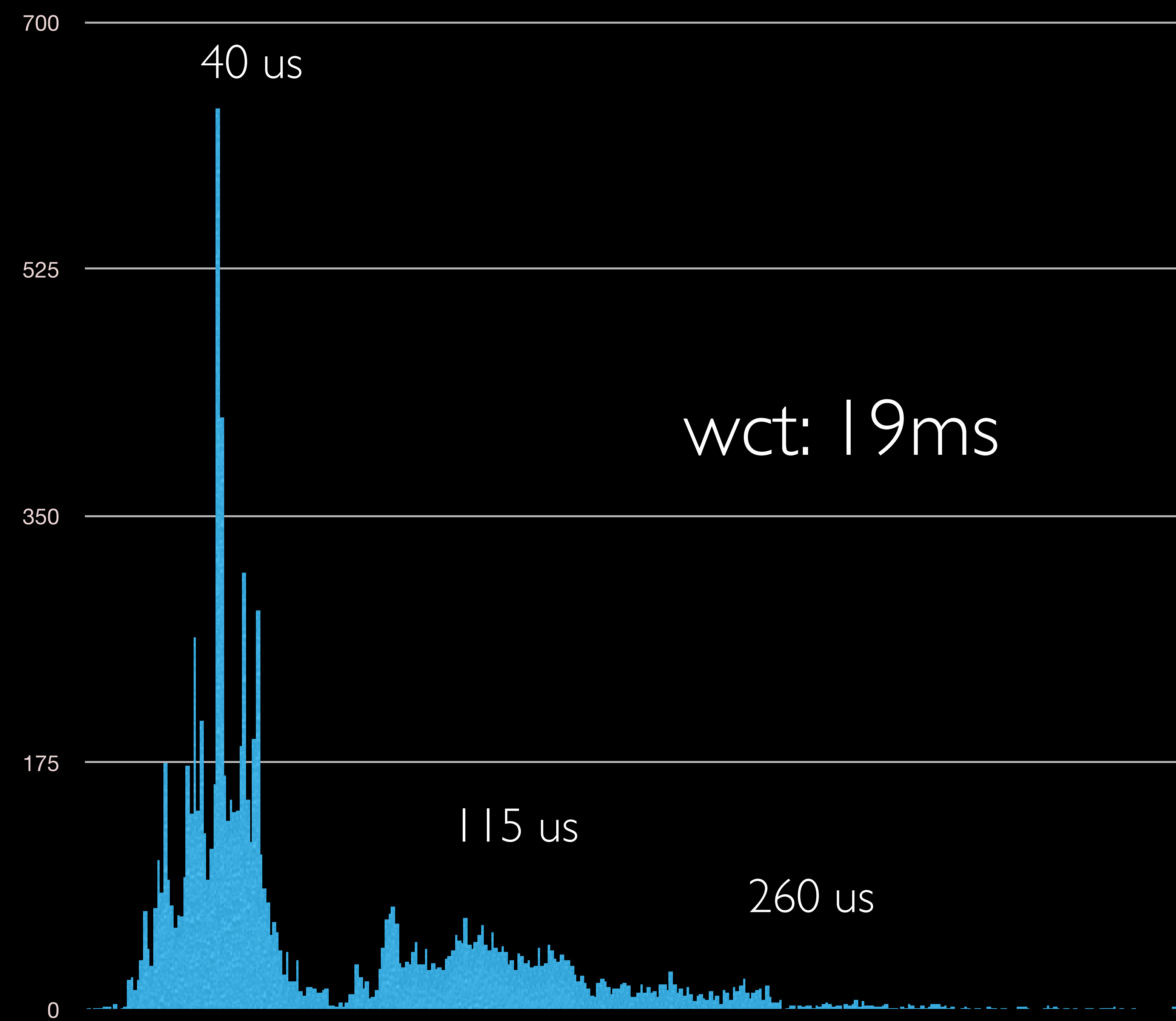
- Object-oriented programming helps **software reuse**
- Mature **development environment** and **libraries**
- **Garbage collected** & **Memory-safe** high-level language
- **Portable**, little implementation-specific behavior
- **Concurrency** built-in, support for SMP, memory model
- **Popular** amongst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

Java?

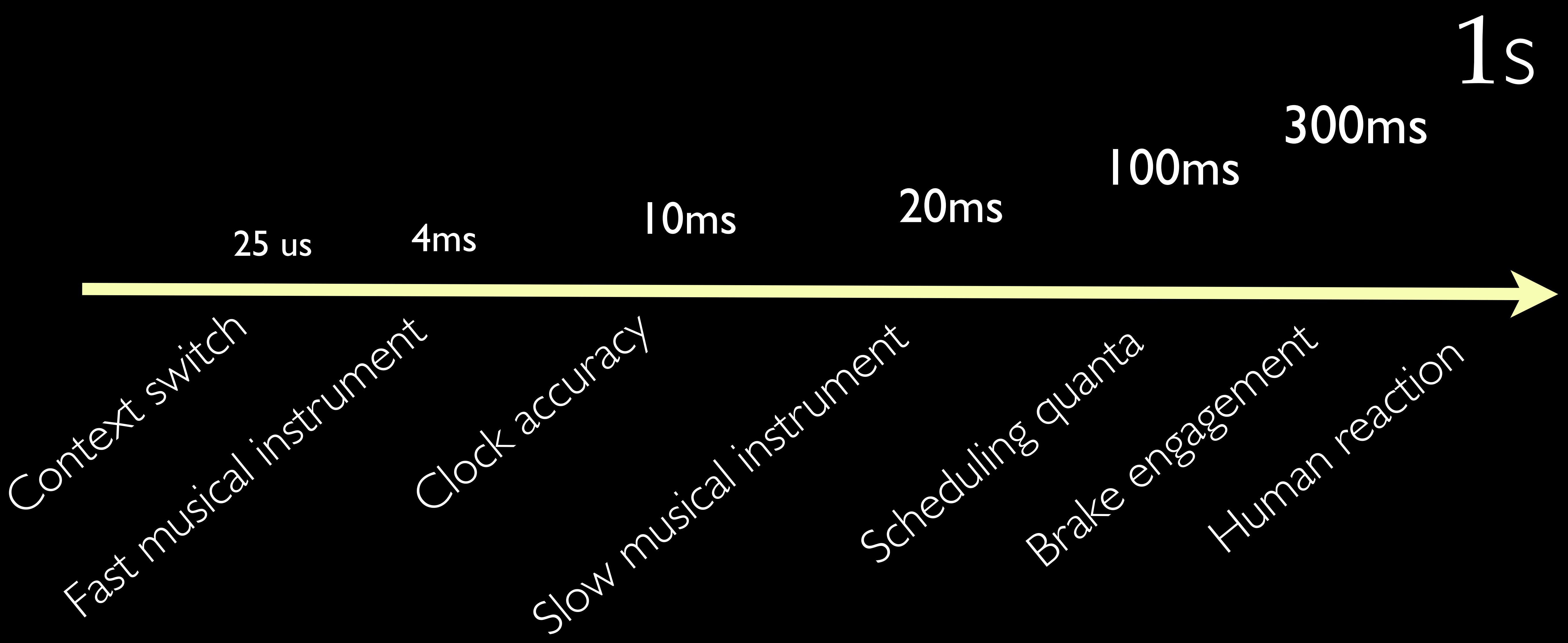
- Predictable?
- *Not really.*

Call sleep(10ms) and get up 20 milis.sec. variability.

Hard real-time often requires microsecond accuracy.

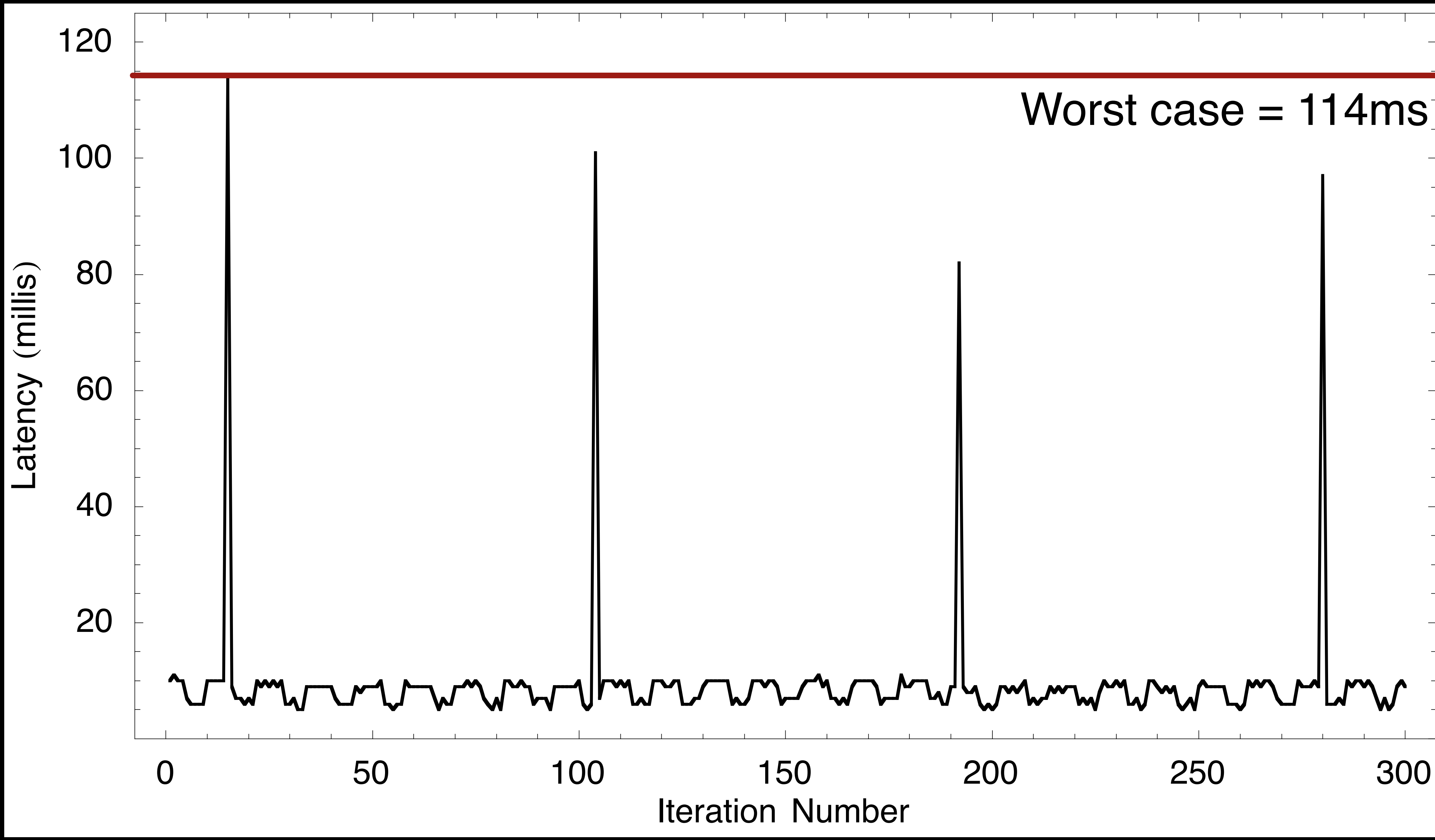


Time scale


1s
300ms
100ms
20ms
10ms
4ms
25 us
Context switch
Fast musical instrument
Clock accuracy
Slow musical instrument
Scheduling quanta
Brake engagement
Human reaction

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

- 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.

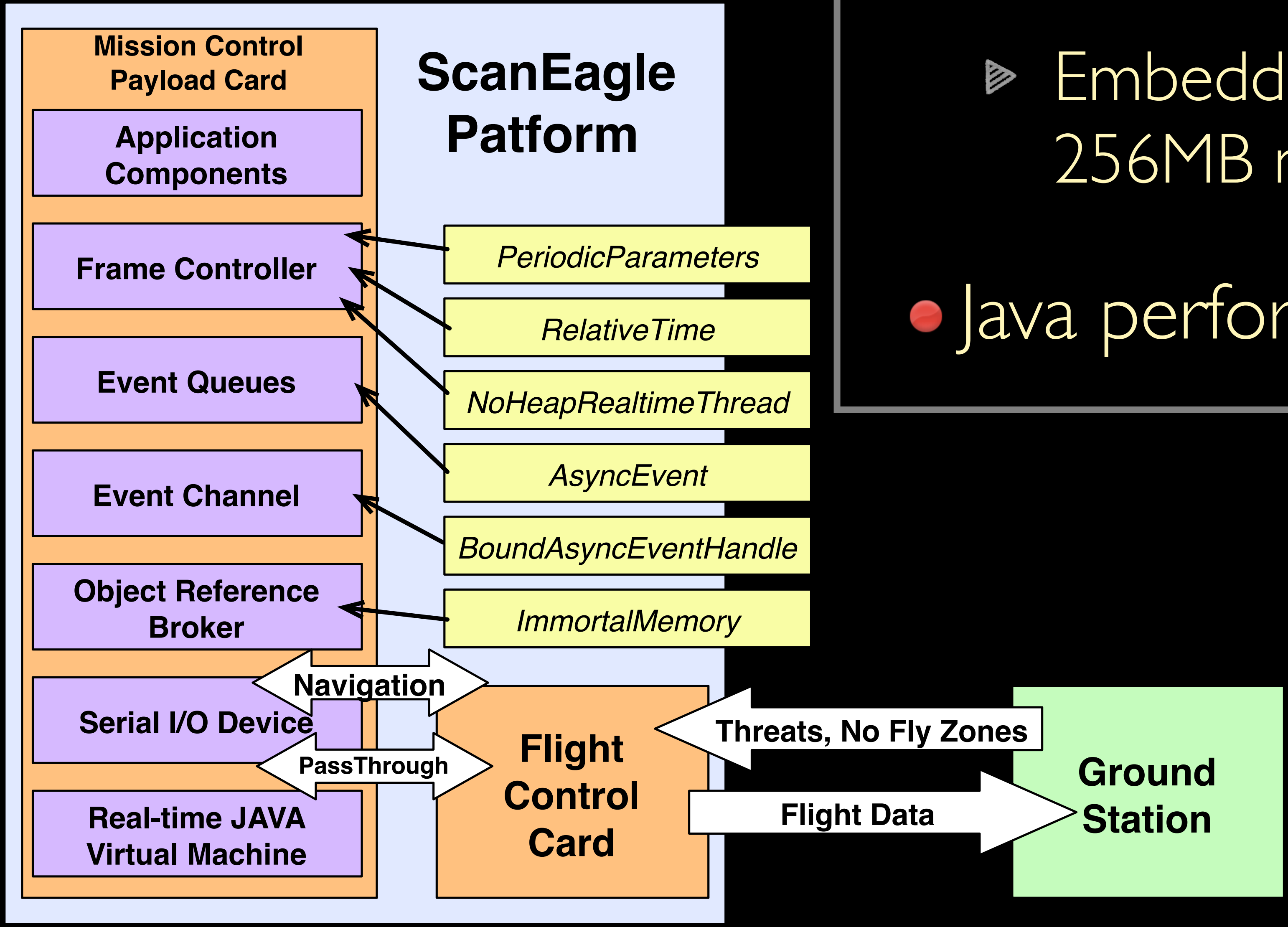
Duke's Choice Award

Case Study: ScanEagle

ScanEagle

ScanEagle

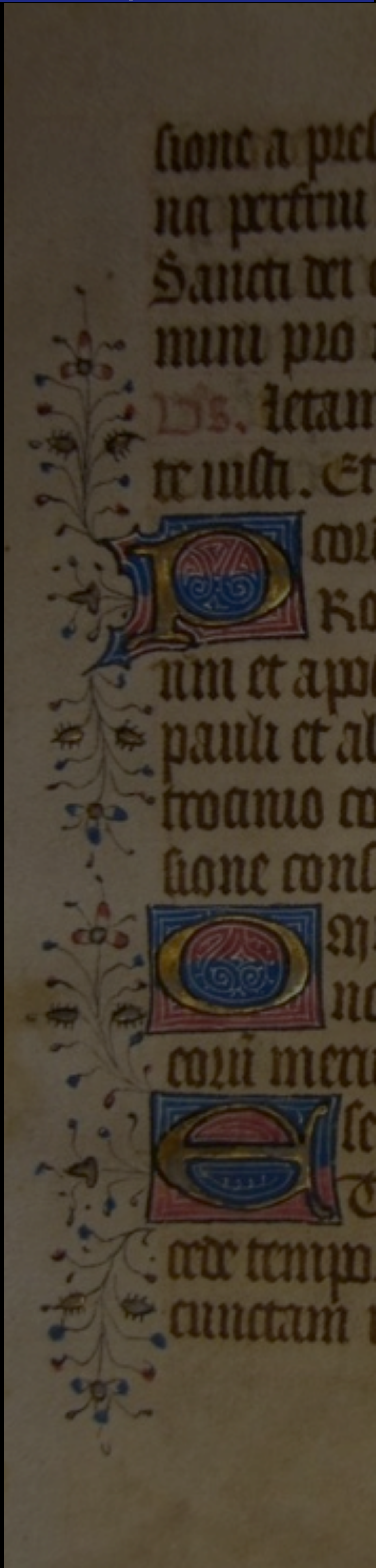
- Flight Software:
  - 953 Java classes, 6616 methods. Multiple Priority Processing:
    - High (20Hz) - Communicate with Flight Controls
    - Medium (5 Hz) - Computation of navigation data
    - Low (1 Hz) - Performance Computation
  - Embedded Planet 300 Mhz PPC, 256MB memory, Embedded Linux
- Java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

Minuteman

Real-time Garbage Collection

Memory management and programming models

- The choice of memory management affects productivity
- Object-oriented languages naturally hide allocation behind abstraction barriers
  - Taking care of de-allocation manually is more difficult in OO style
- Concurrent algorithms usually emphasize allocation
  - because freshly allocated data is guaranteed to be thread local
  - "transactional" algorithms generate a lot of temporary objects
- … but garbage collection is a global, costly, operation that introduces unpredictability

Alternative 1: No Allocation

- If there is no allocation, GC does not run.
  - This approach is used in JavaCard

Alt 2: Allocation in Scoped Memory

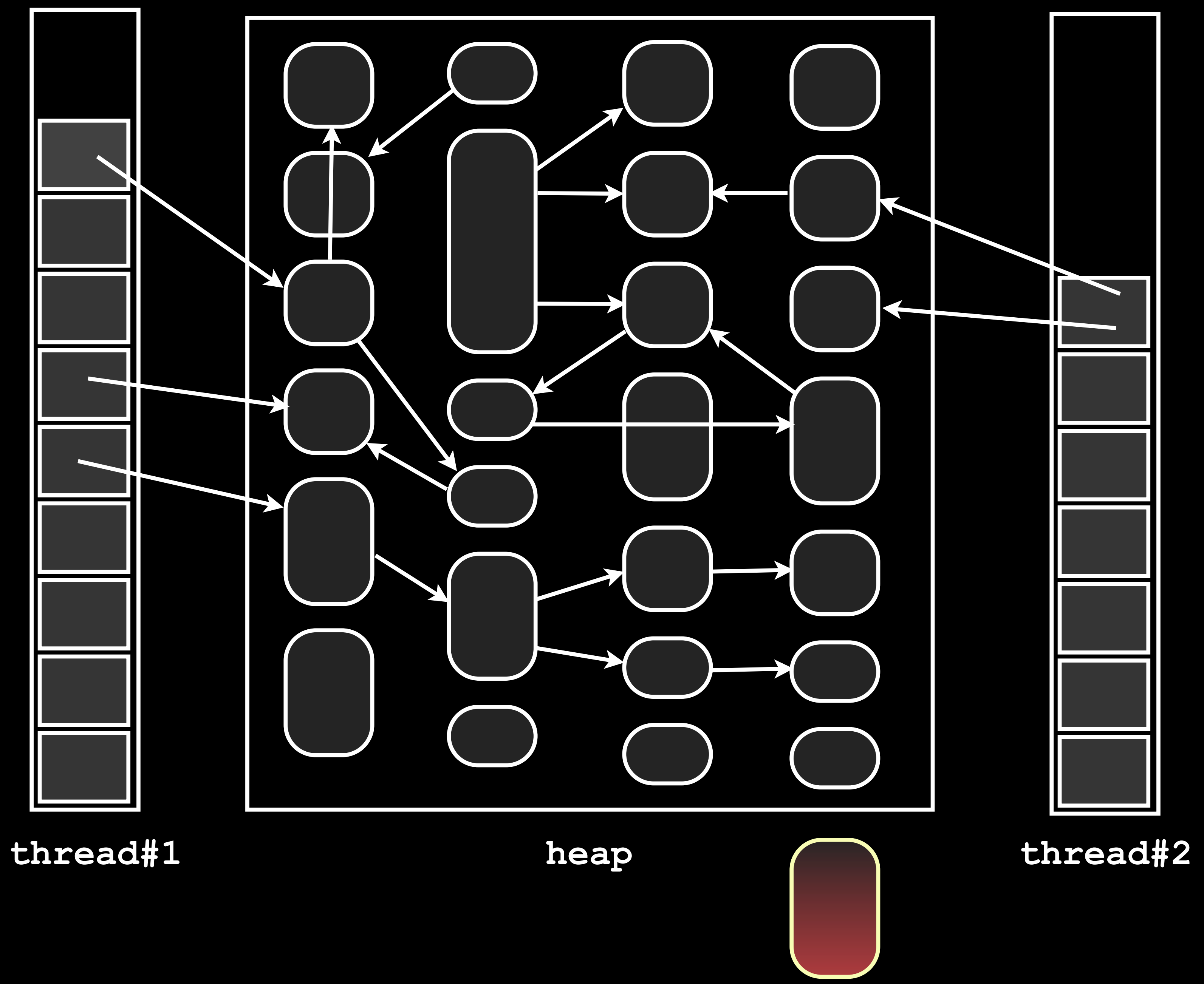
- RTSJ provides scratch pad memory regions which can be used for temporary allocation
  - Used in deployed systems, but tricky as they can cause exceptions

```
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*

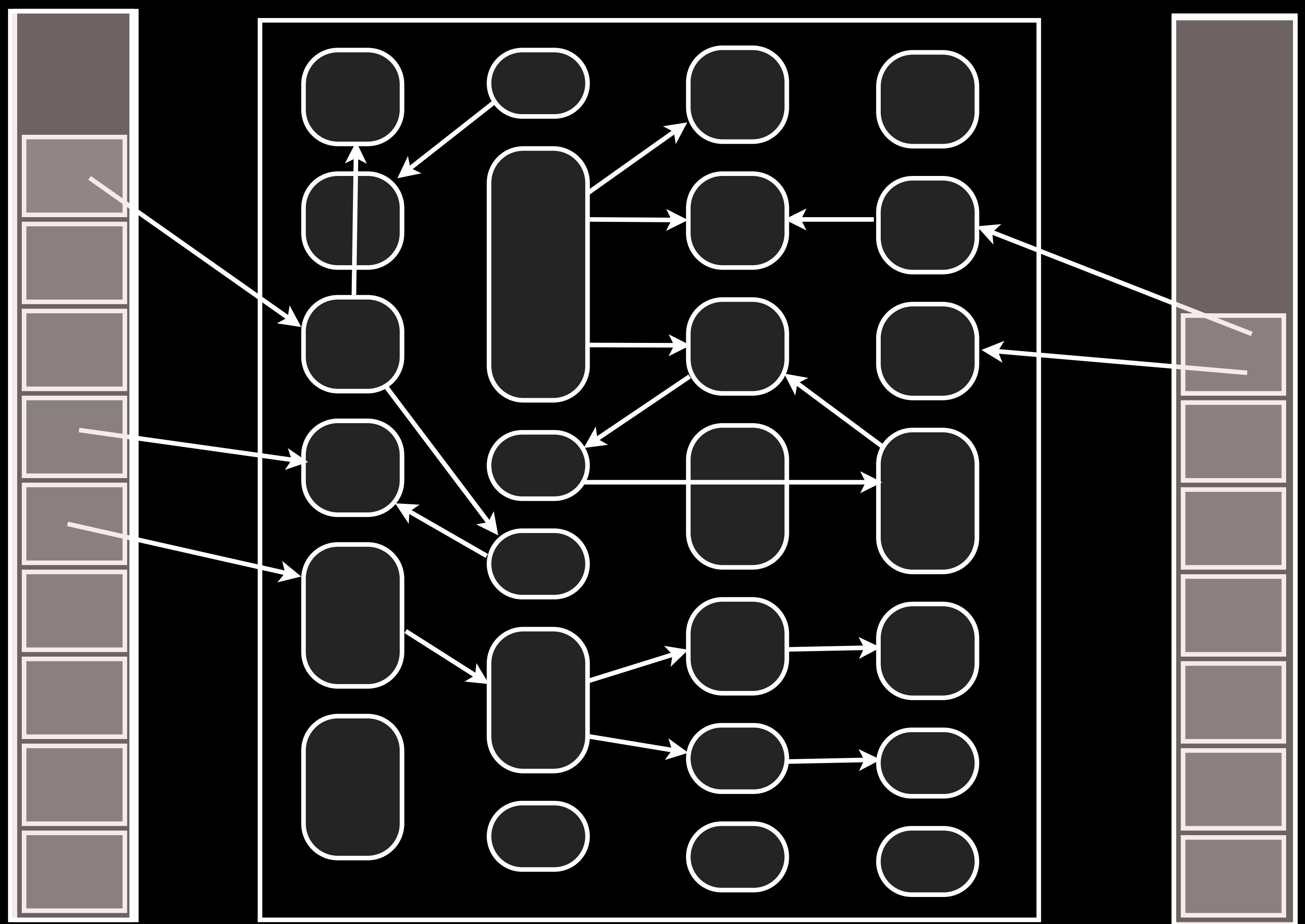
Garbage Collection



Phases

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

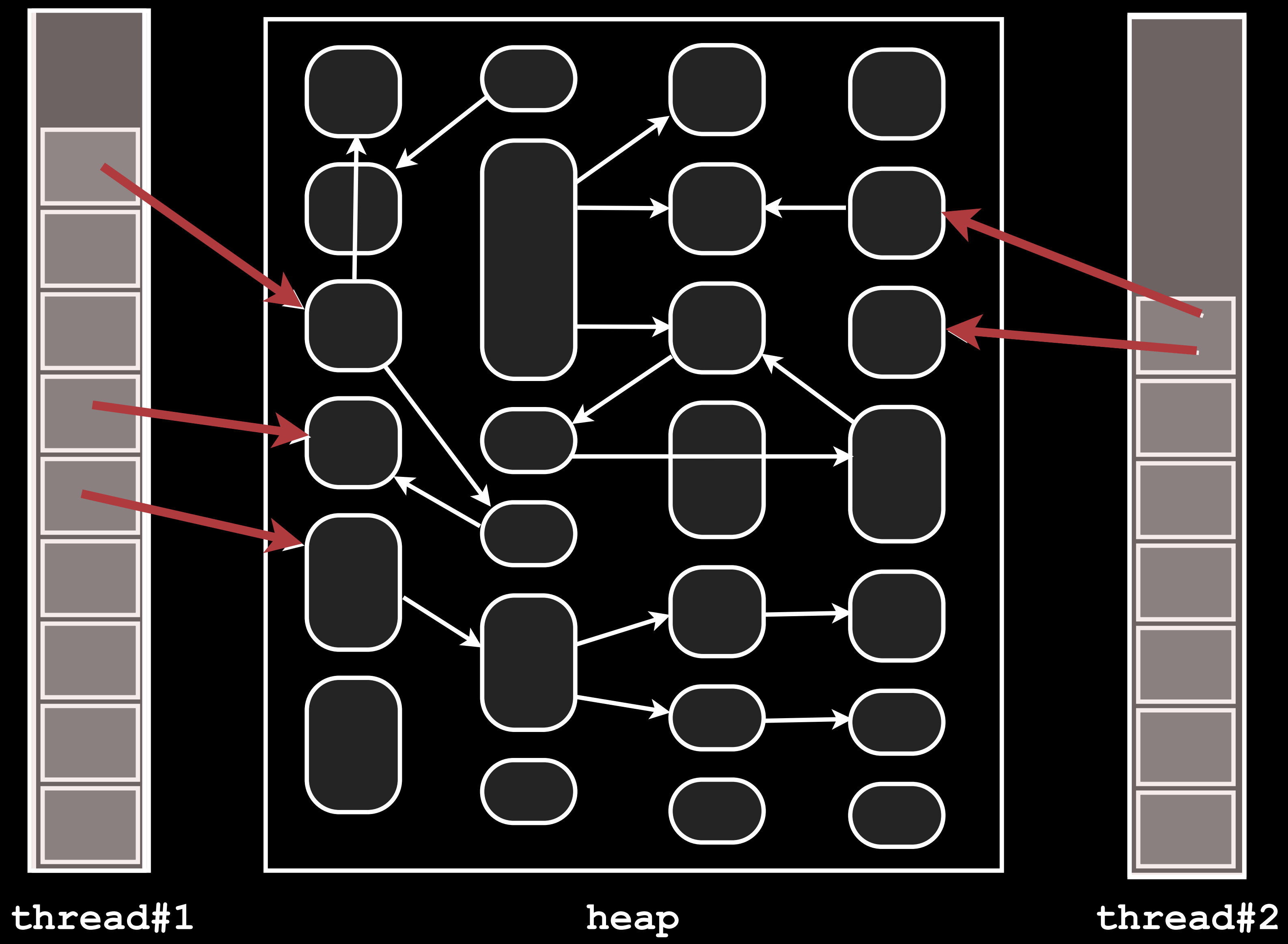
Garbage Collection



Phases

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

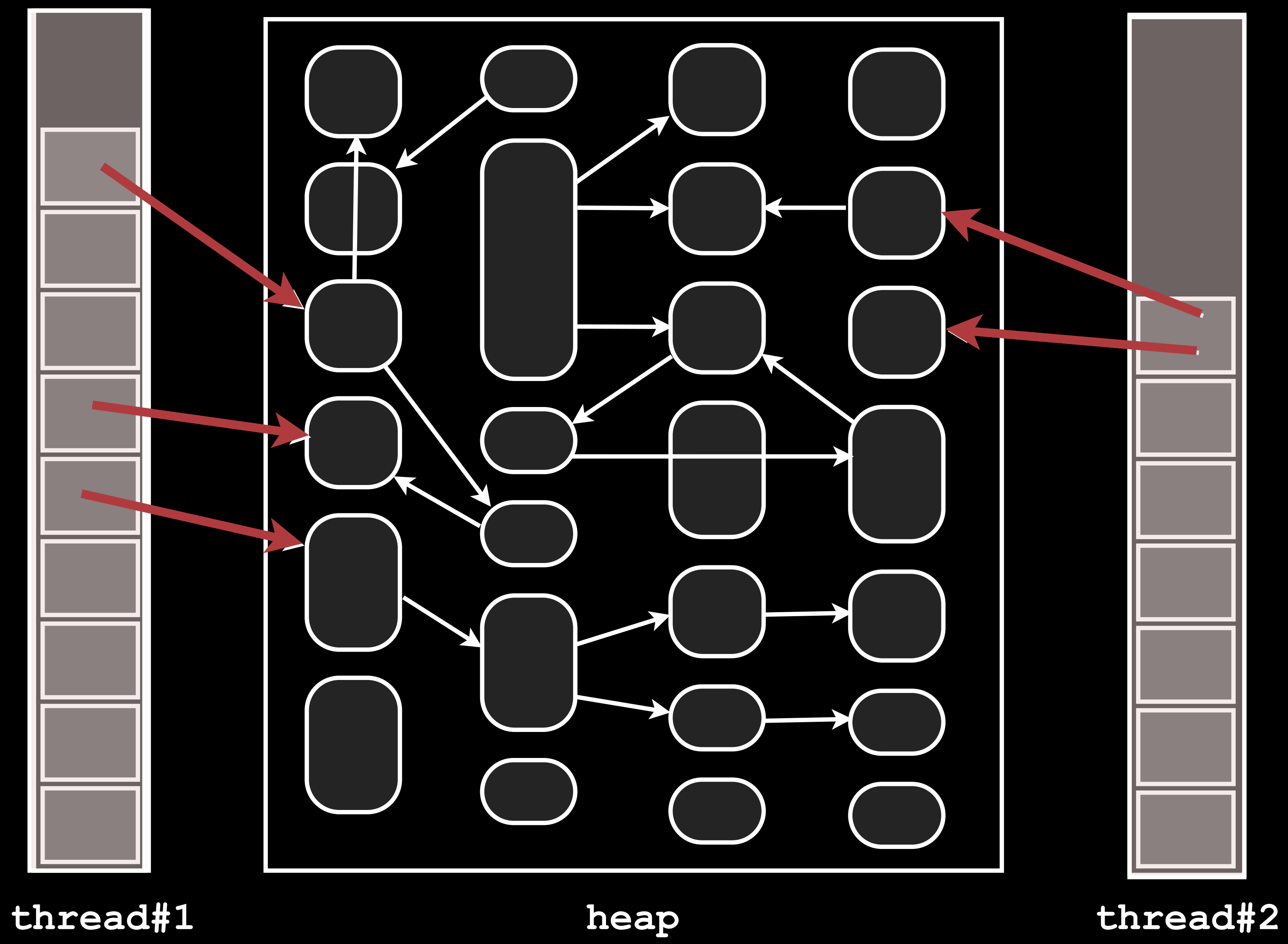
Garbage Collection



Phases

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

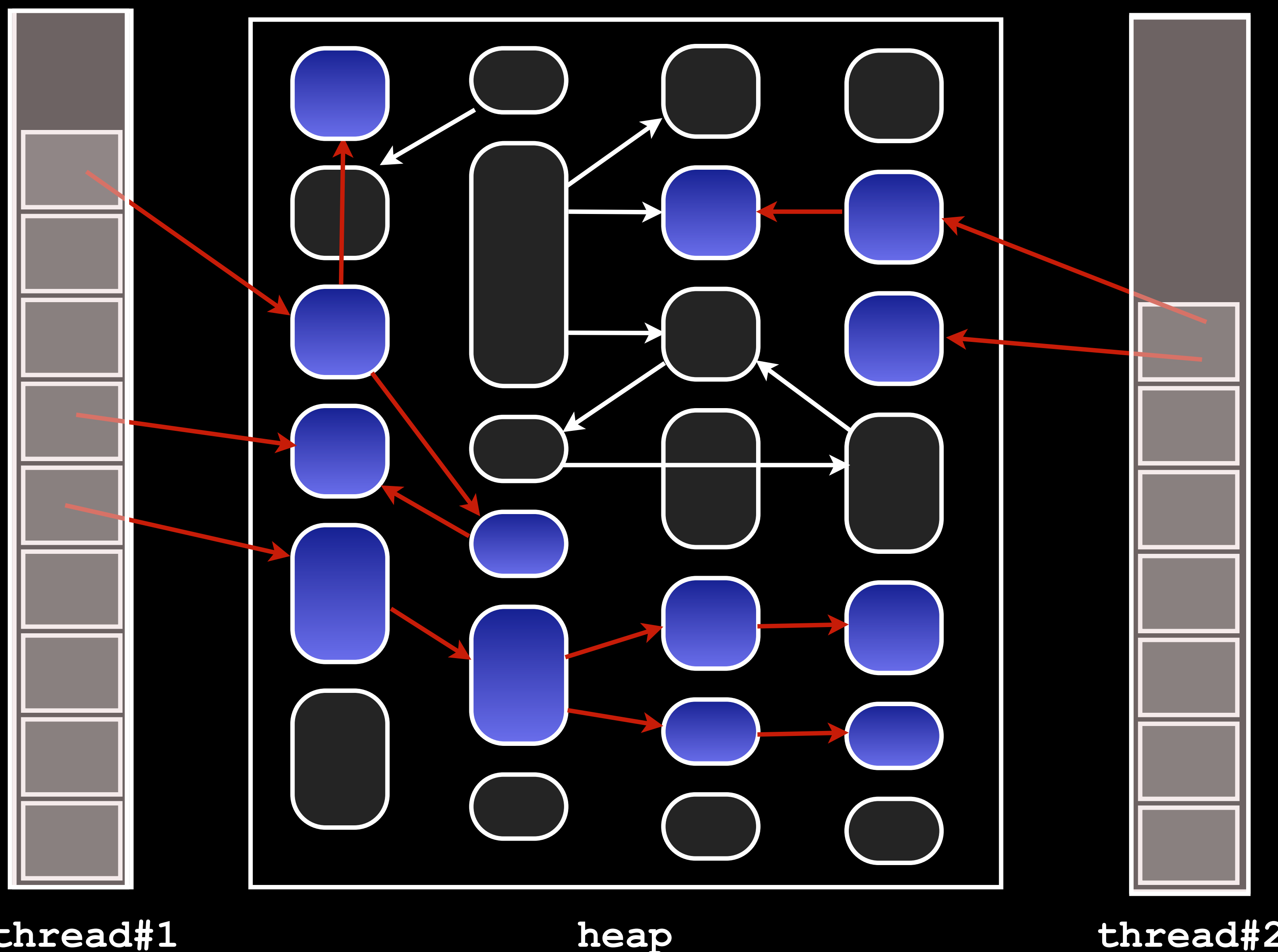
Garbage Collection



Phases

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

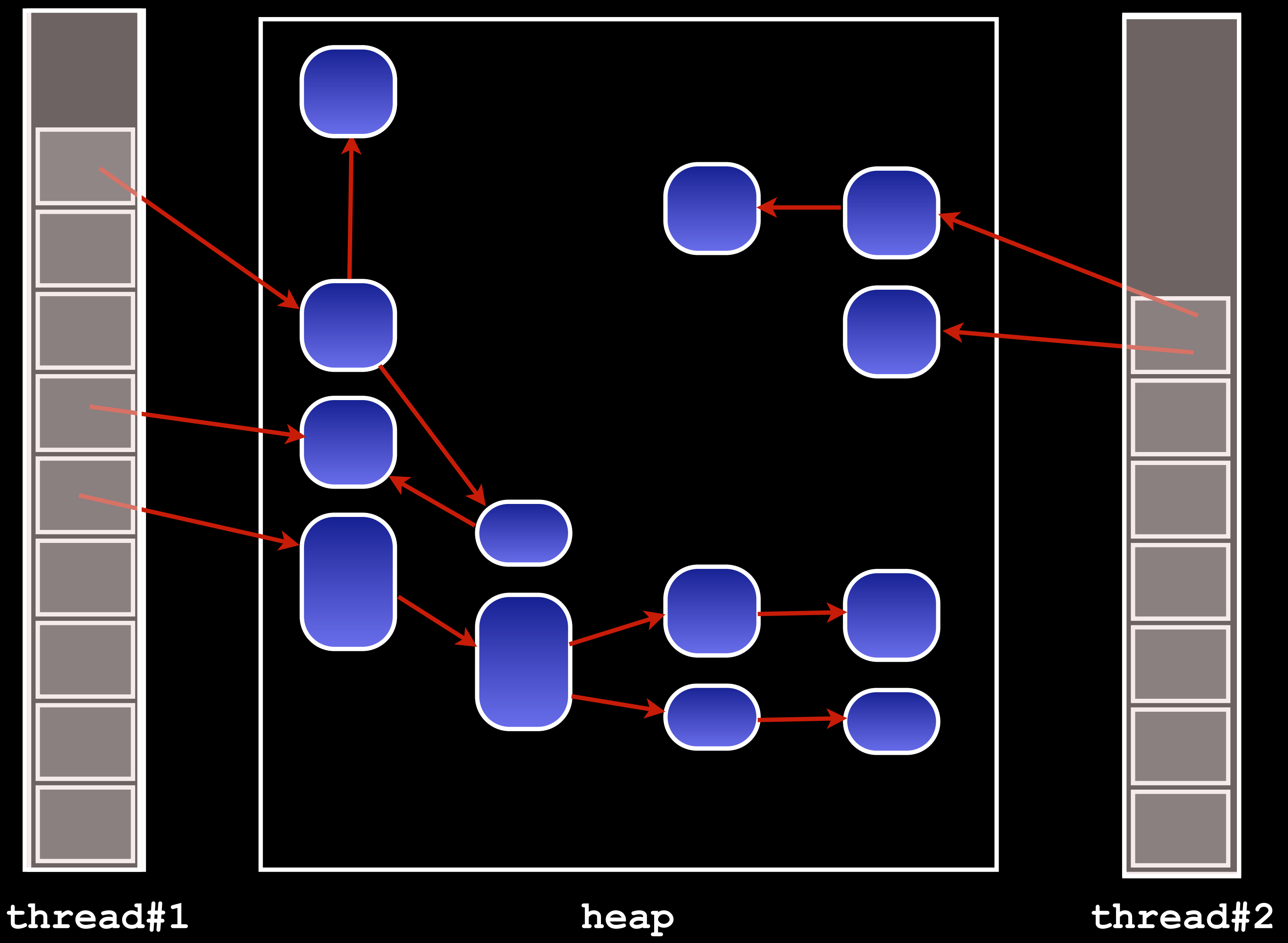
Garbage Collection



Phases

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

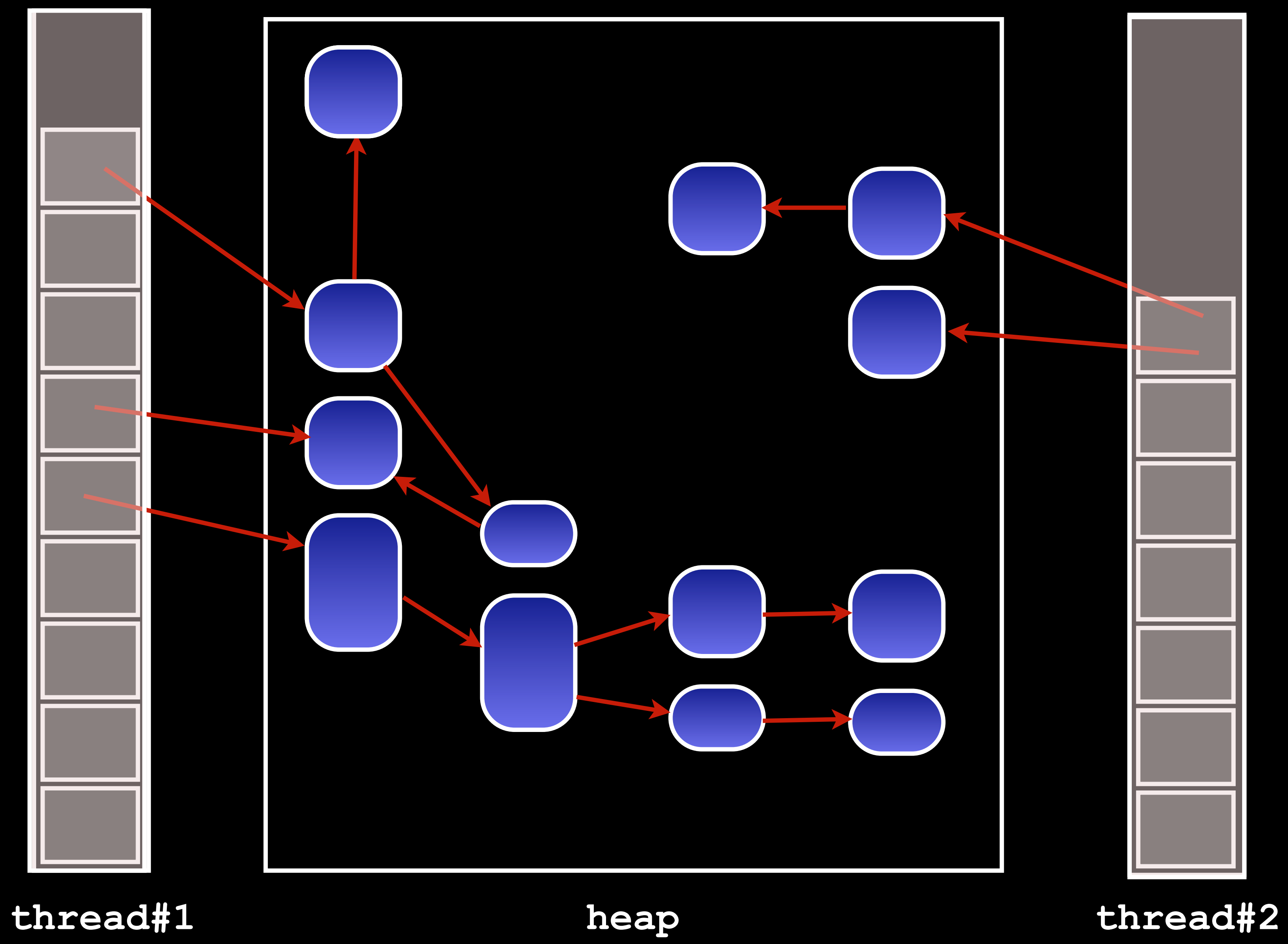
Garbage Collection



Phases

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

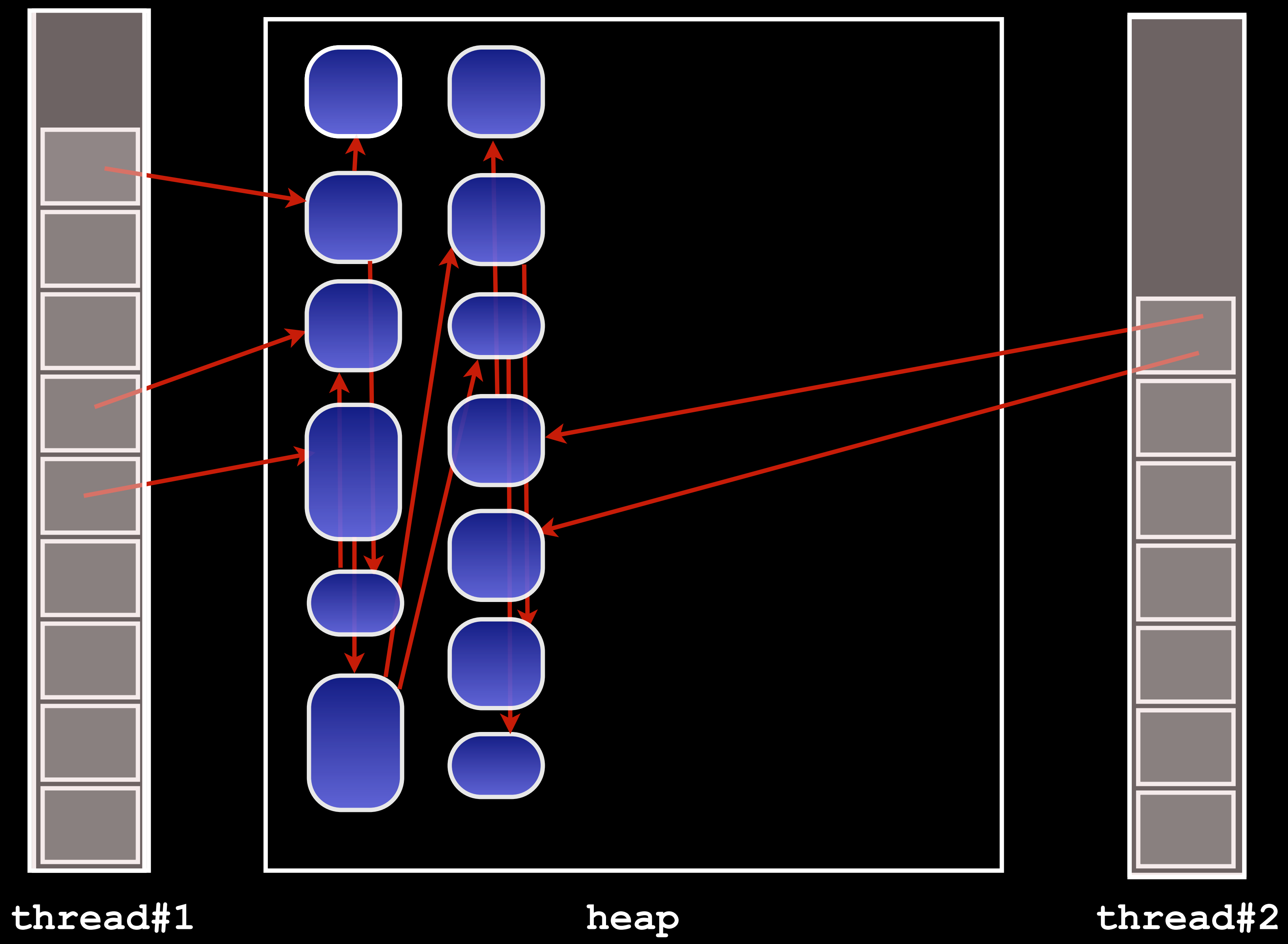
Garbage Collection



Phases

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

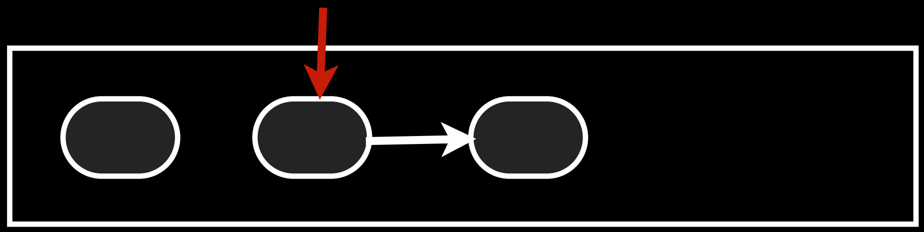
Garbage Collection

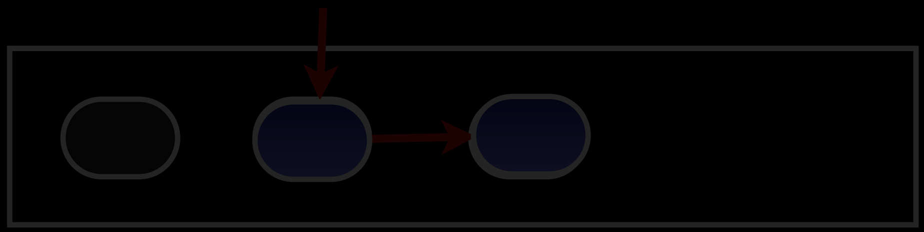


Phases

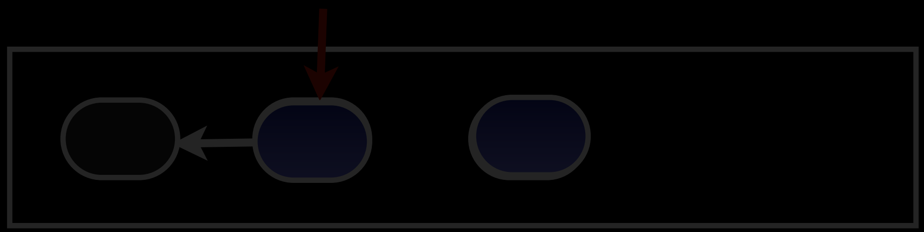
- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction

Incrementalizing marking

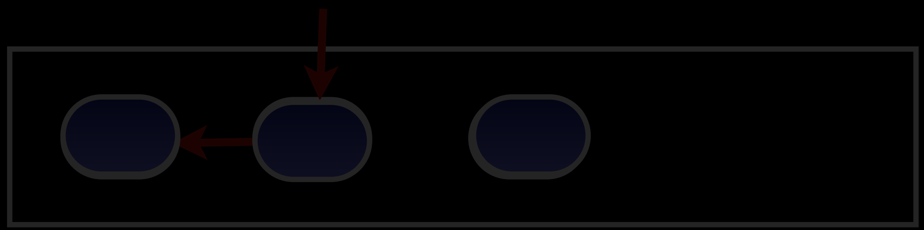




Collector marks object



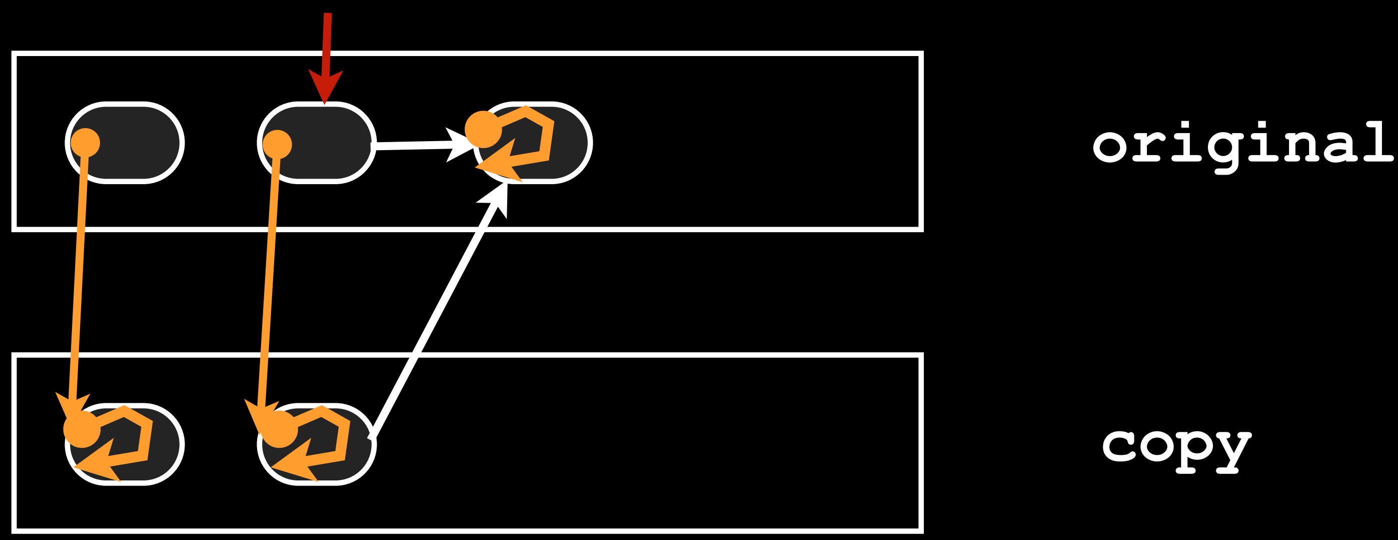
Application updates reference field



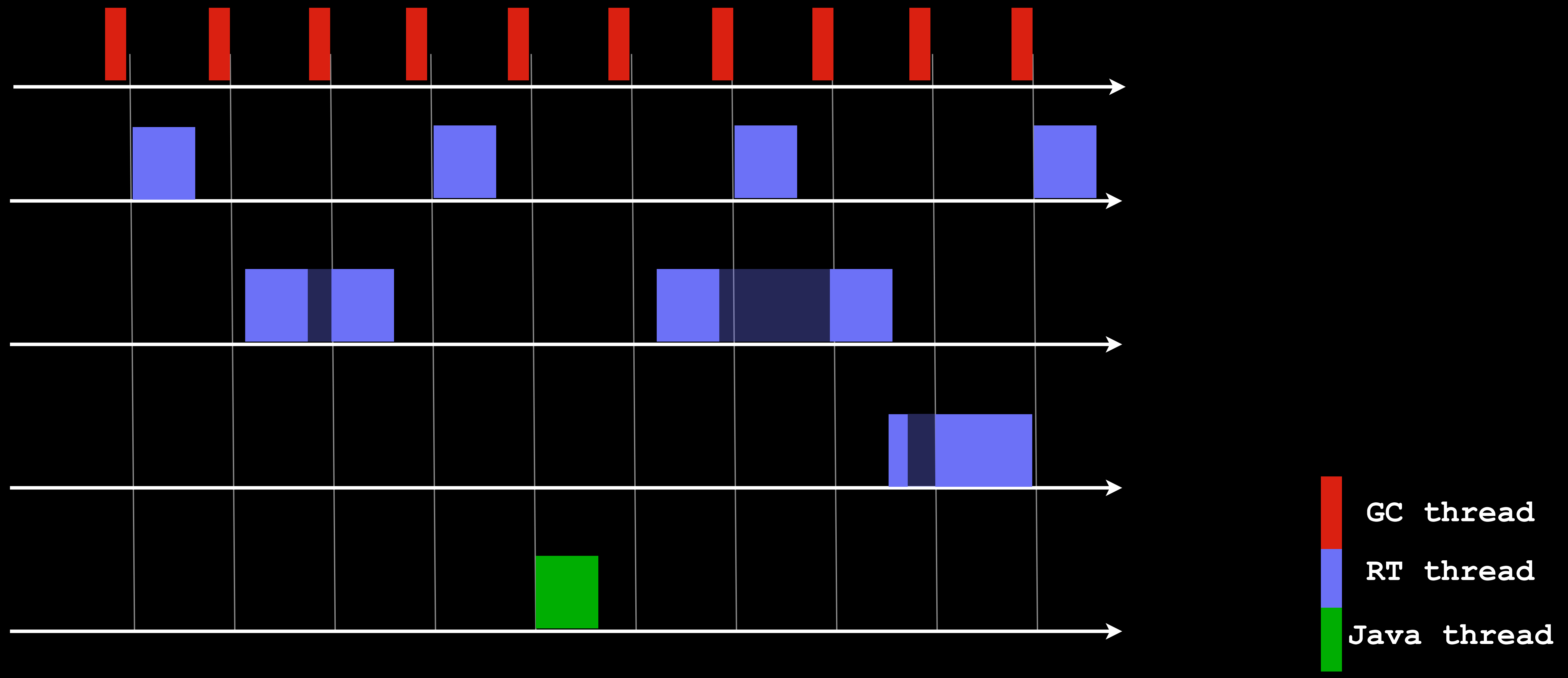
Compiler inserted write barrier marks object

Incrementalizing compaction

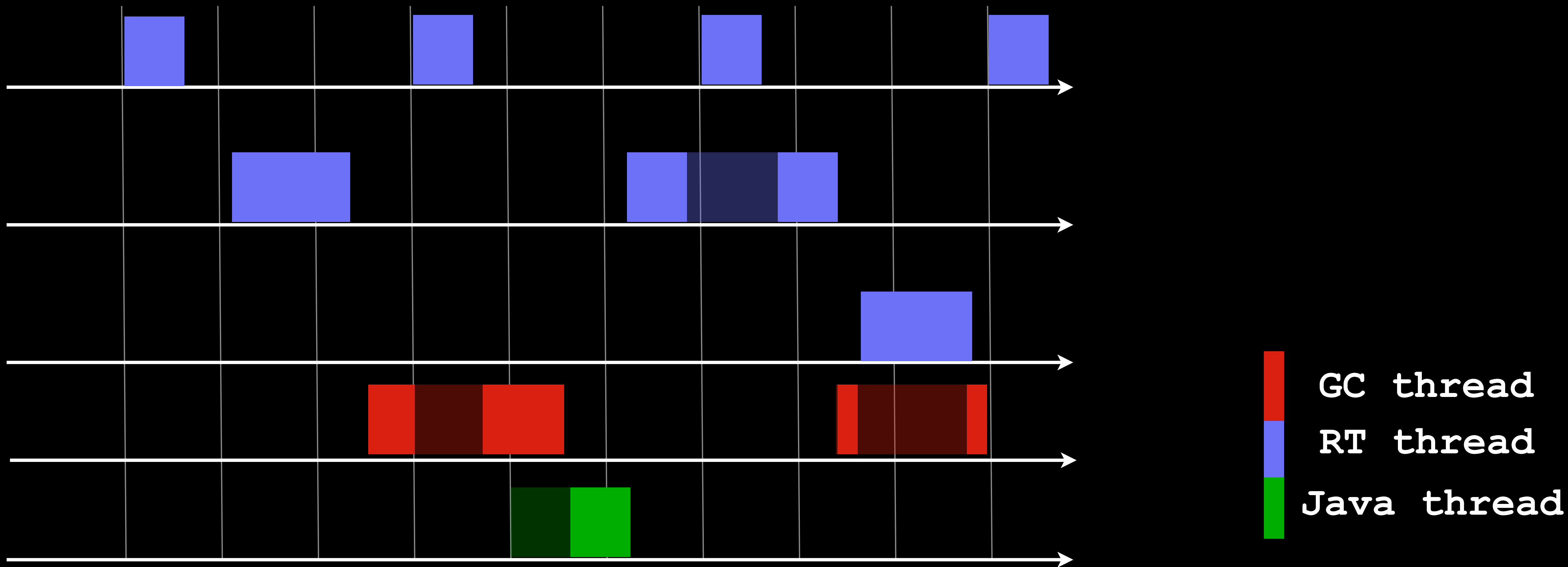
- Forwarding pointers refer to the current version of objects
- Every access must start with a derefence

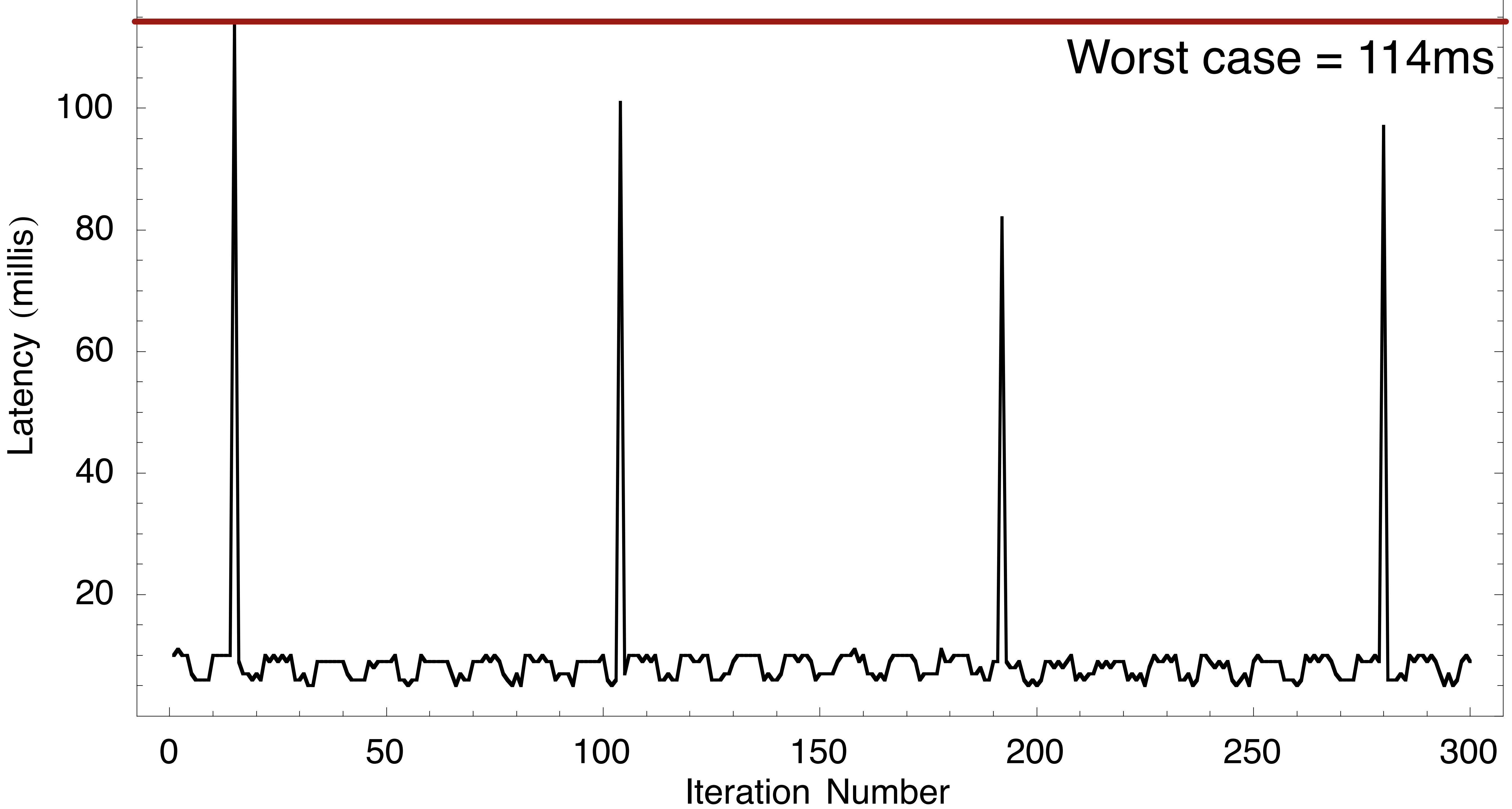


Time-based GC Scheduling

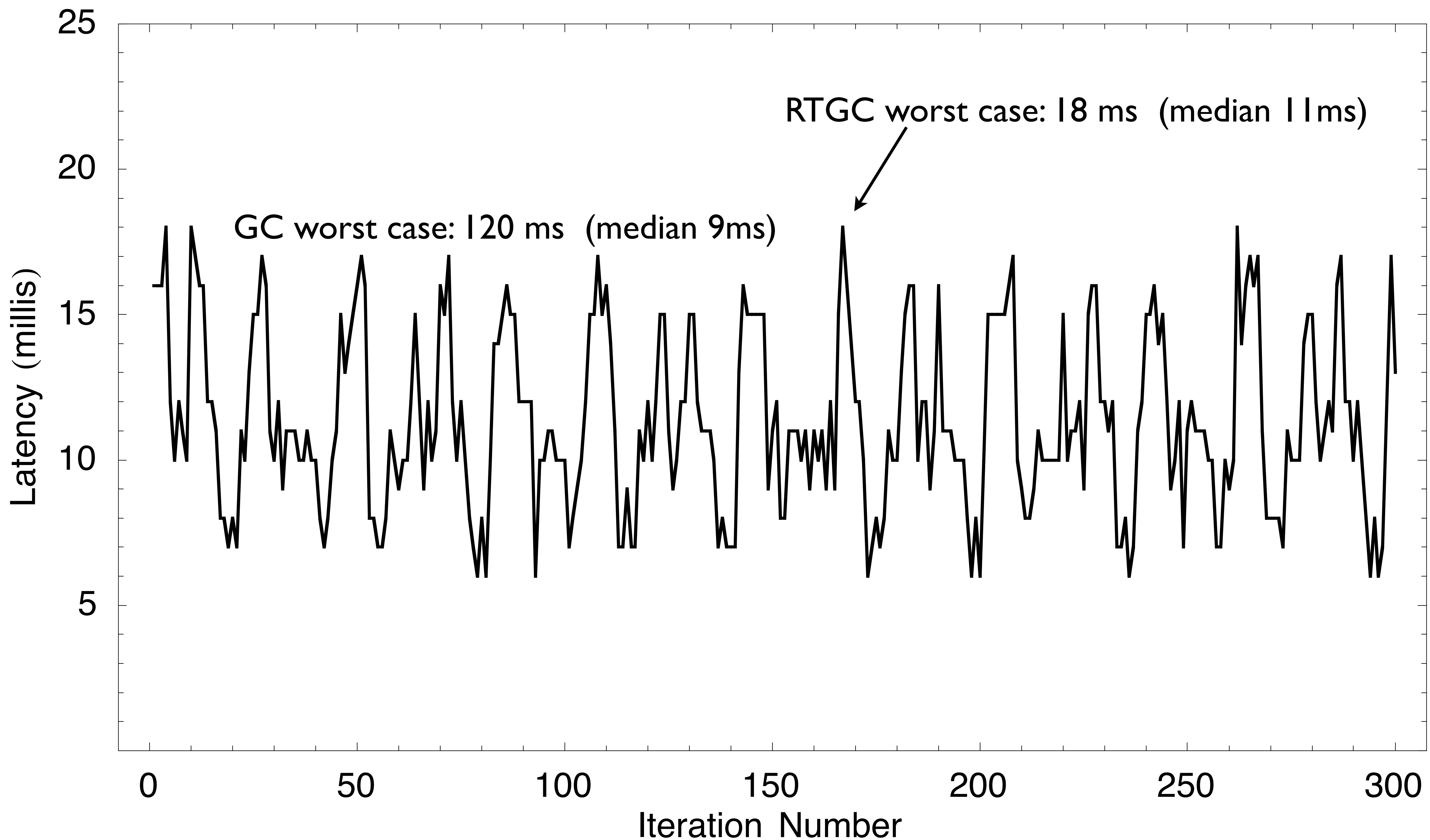


Slack-based GC Scheduling

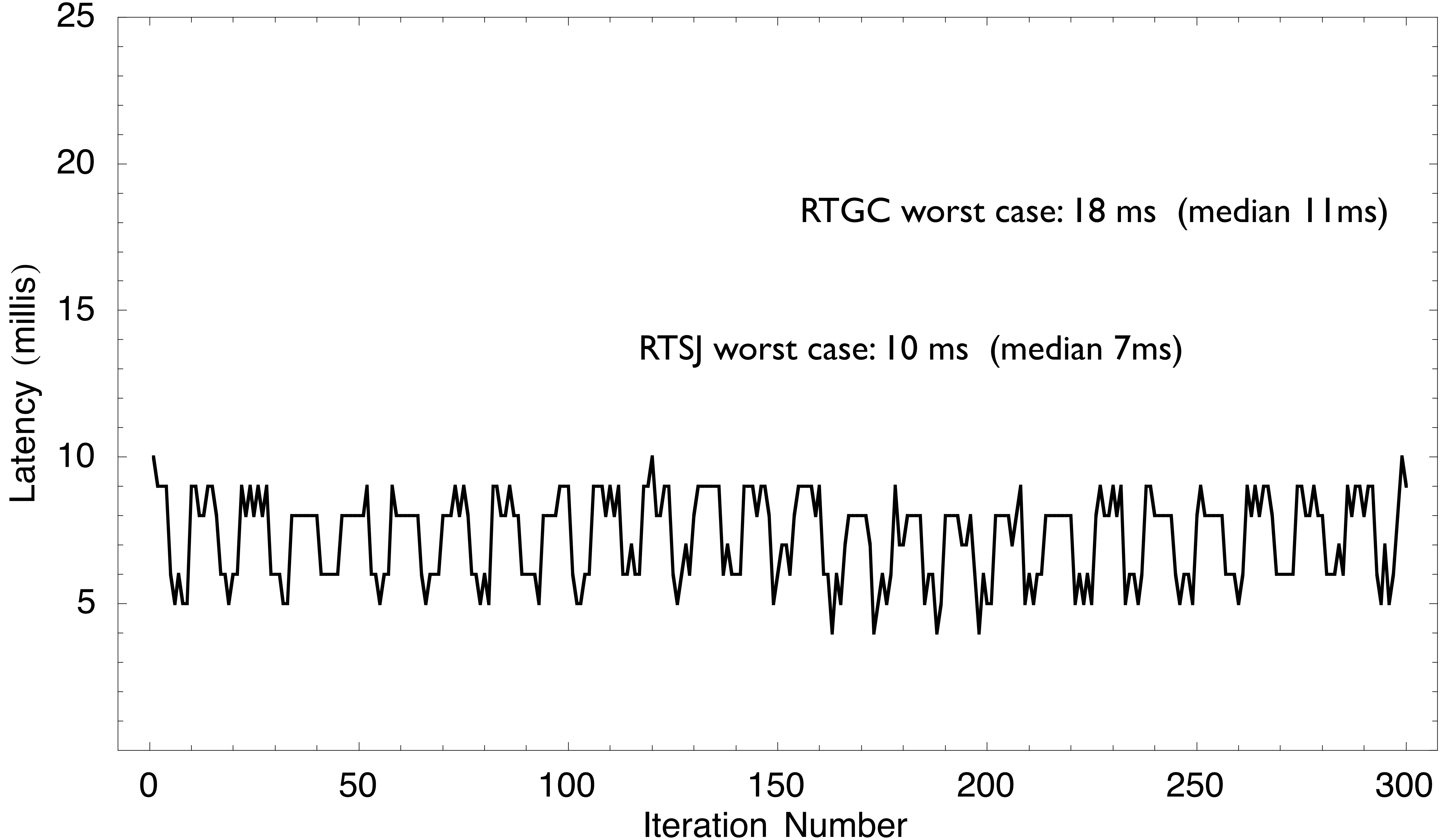




- GC pauses cause the collision detector to miss deadlines…
  and this is not a particularly hard problem should support KHz periods



CD with periodic RTGC

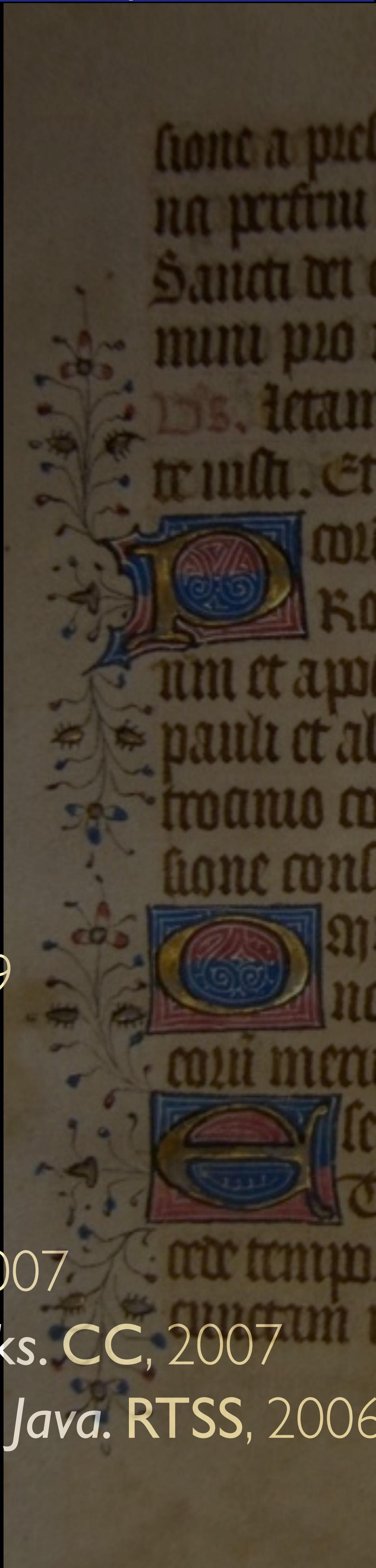


Slack-based GC

References and acknowledgements

- **Team**
  - *J. Baker, T. Cunei, T. Kalibera, T. Hosking, F. Pizlo, M. Prochazka*
- **Funding:** *NSF*
- **Availability:** open source
- **Paper trail**

- *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




3

Flexotask

Flexible Task Graphs

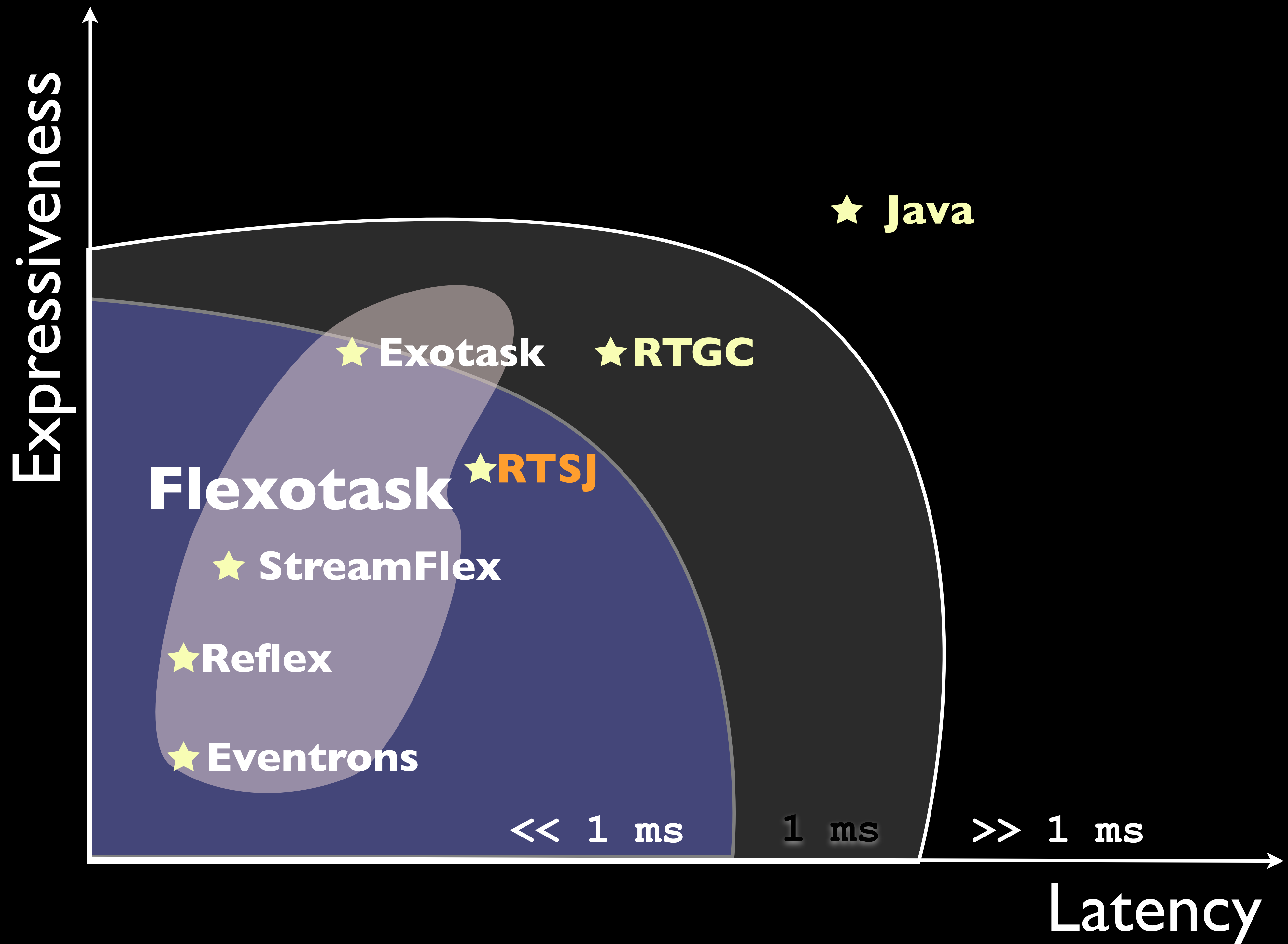
Goals

- Design a new real-time programming model that allows embedding hard real-time computations in timing-oblivious Java applications
- Principle of Least Surprise
  - Semantics of non-real-time code unchanged
  - Semantics of real-time code unsurprising
- Limited set of new abstractions that compose flexibly
- No cheating
  - Run efficiently in a production environment

Unification of previous work

- **Eventrons** [PLDI'06] (IBM)
- **Reflexes** [VEE'07] (Purdue/EPFL)
  - *Inspired by RTSJ and Eventrons*
- **Exotasks** [LCTES'07] (IBM)
  - *Inspired by Giotto, and E-machine*
- **StreamFlex** [OOPSLA'07] (Purdue/EPFL)
  - *Inspired by Reflexes, StreamIt and dataflow languages*

Design space

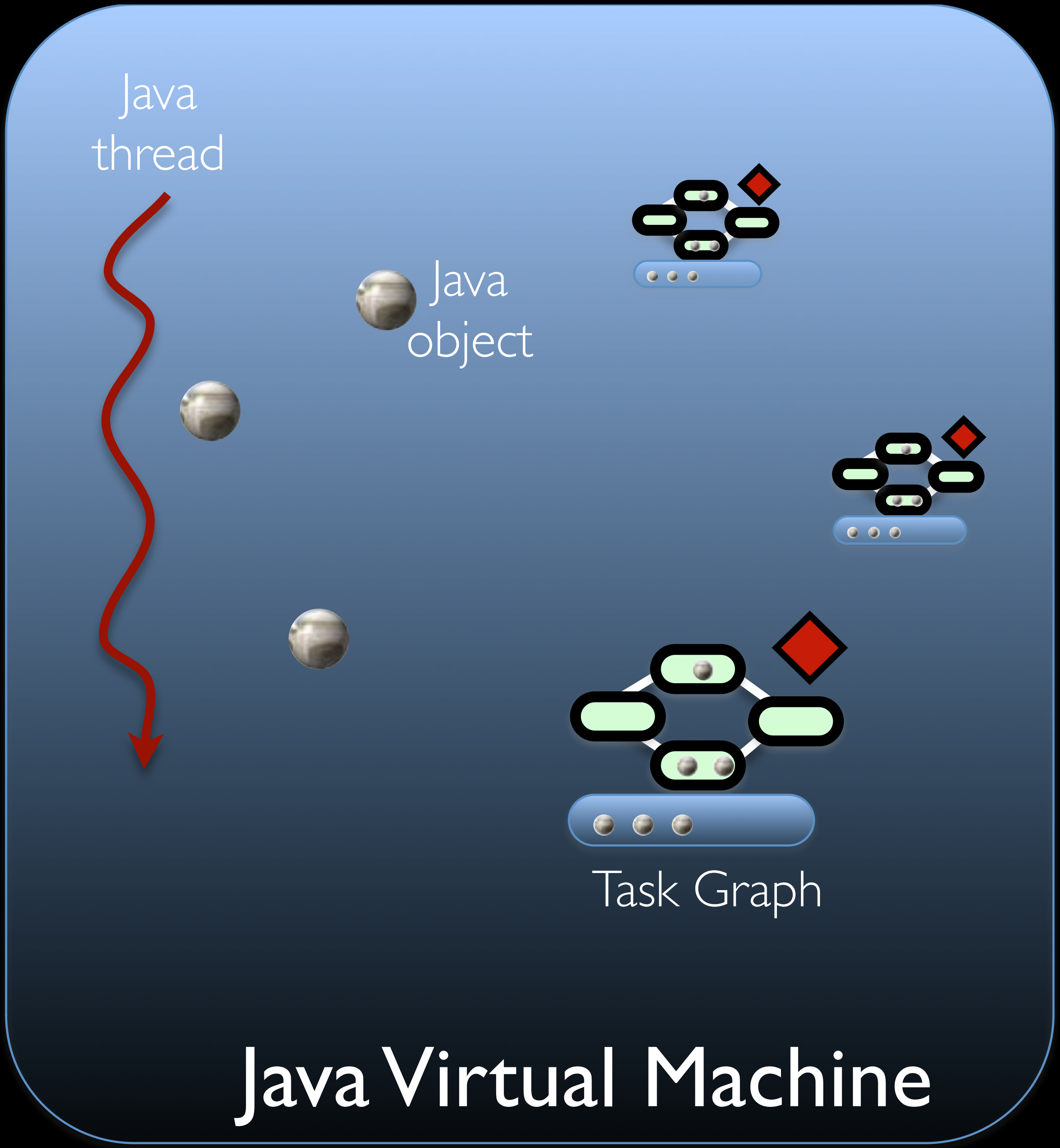


Programming model

- Basic model:
  - No shared state, (but local sate), no low-level data races
  - Components communicate via atomic channels
  - Memory management is either GCed or Region-allocated
  - Time triggered scheduler
    - *Inspiration: Actors, Erlang, ...*
- Extensions:
  - Rate driven schedulers
    - *Inspiration: StreamIt, Giotto, ...*
  - Weak isolation for throughput
  - Transactional memory for external interaction

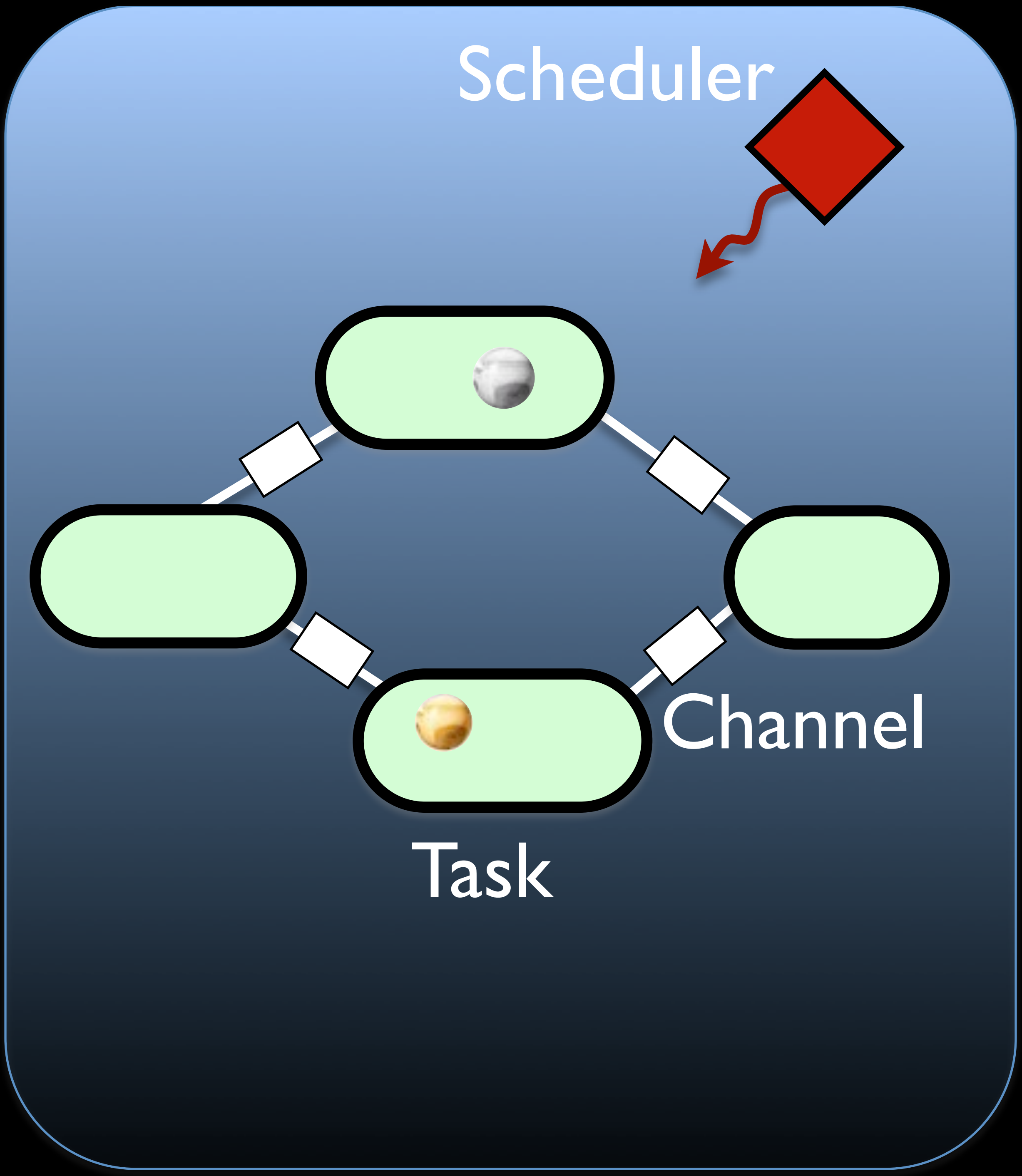
Flexible Task Graphs

- A *FlexoTask Graph* is a set of *concurrently* executing, *isolated*, tasks communicating through non-blocking *channels*
- Semantics of legacy code is unaffected
- Real-time code has restricted semantics, enforced by compile and start-up time static checks



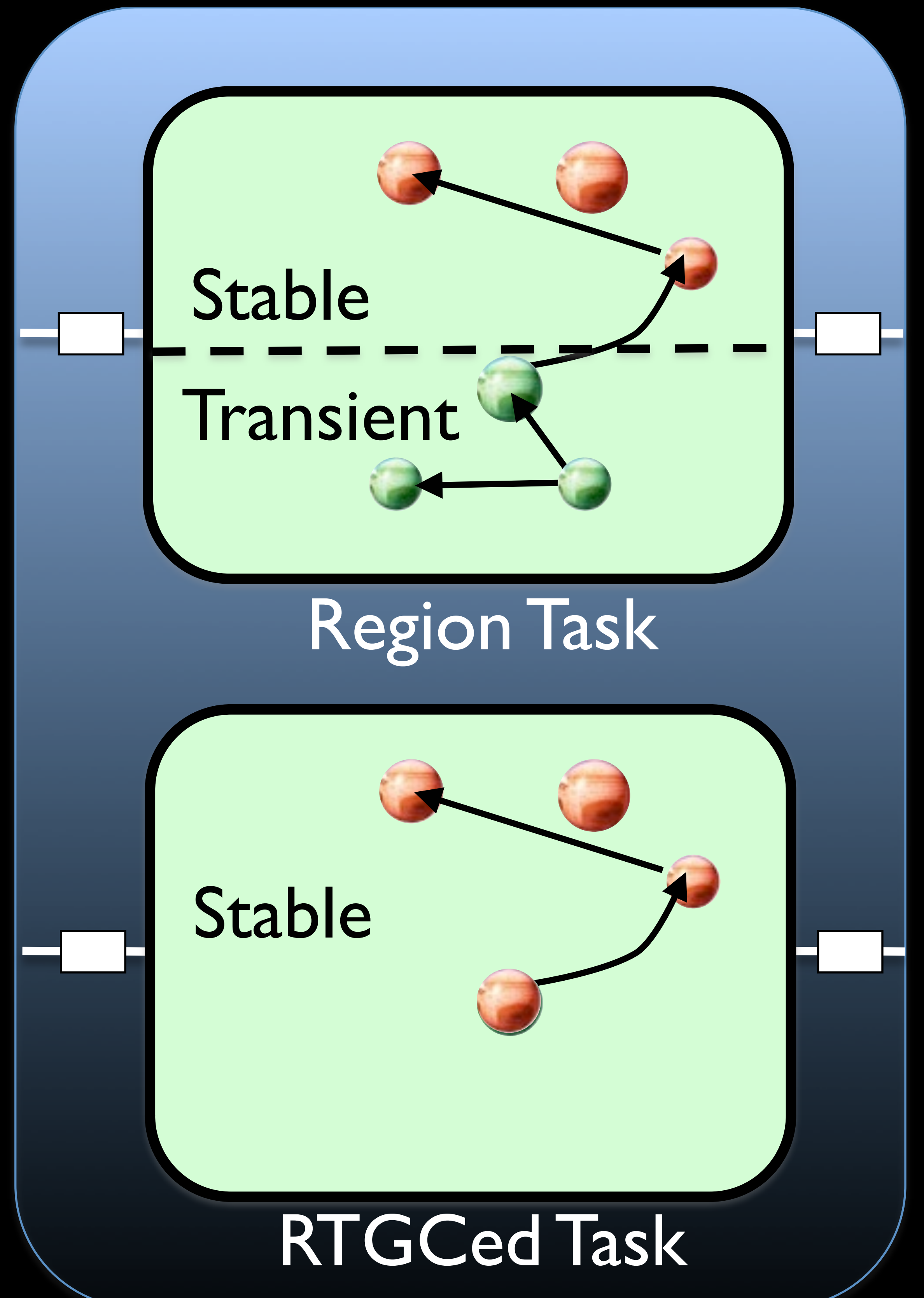
Task Graph

- A *FlexoTask Graph* is a set of *concurrently* executing, *isolated*, tasks communicating through *channels*
- Schedulers control the execution of tasks with user-defined policies (eg. logical execution time, data driven)
  - atomically update task's in ports
  - invoke task's **`execute`**()
  - update the task's output ports



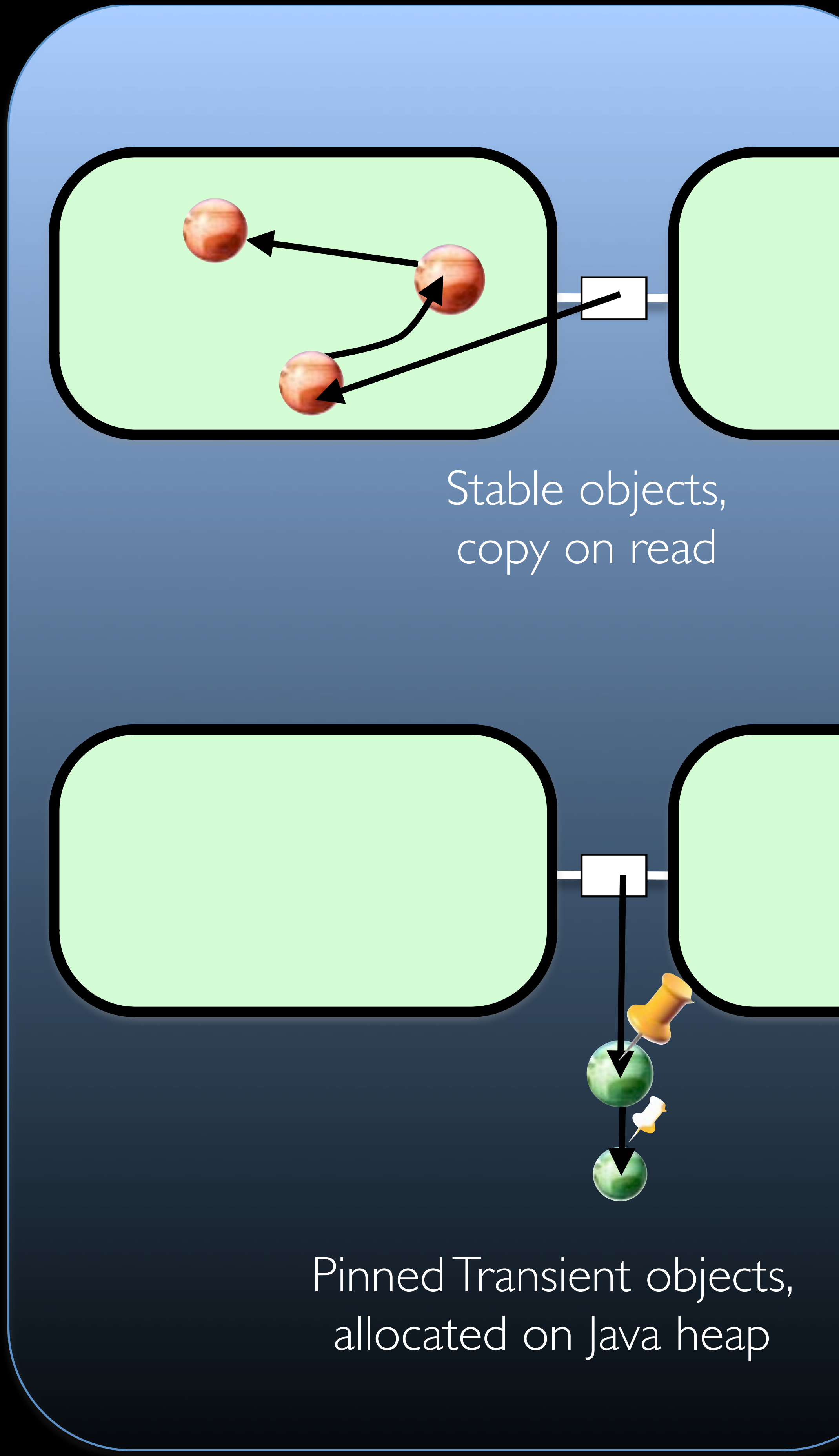
Memory management

- Either garbage collected with a real-time GC, or a region allocator for sub-millisecond response times.
- Region tasks are split between
  - Stable objects
  - Transient (per invocation) objects
- Region-allocated tasks preempt task RTGC and Java GC



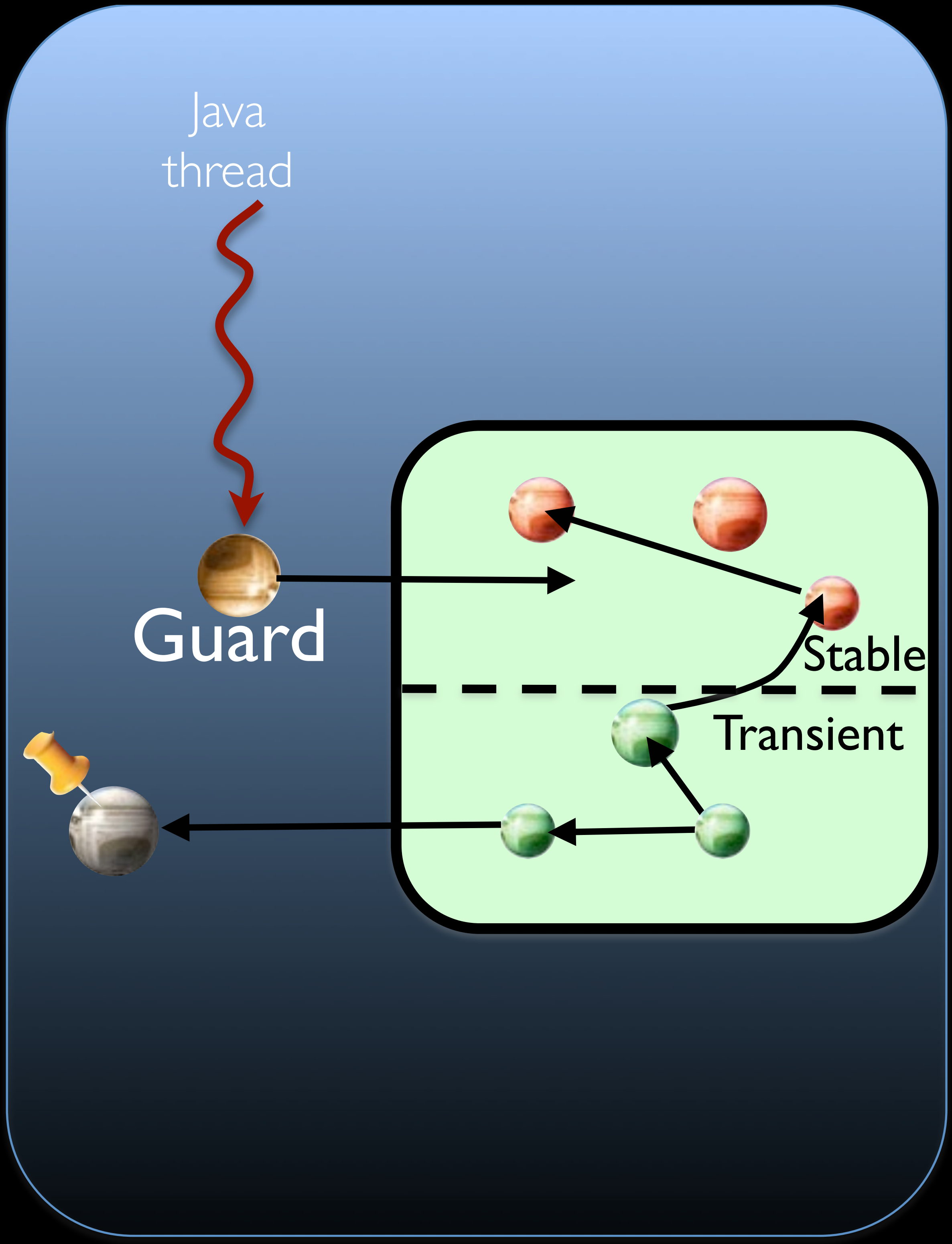
Channels

- Stable channels
  - Can refer to any stable object (complex structures)
  - Deep copy on read (atomic)
- Transient channels
  - Can refer to Capsules (transient objects, arrays)
  - Zero-copy (linear reference)



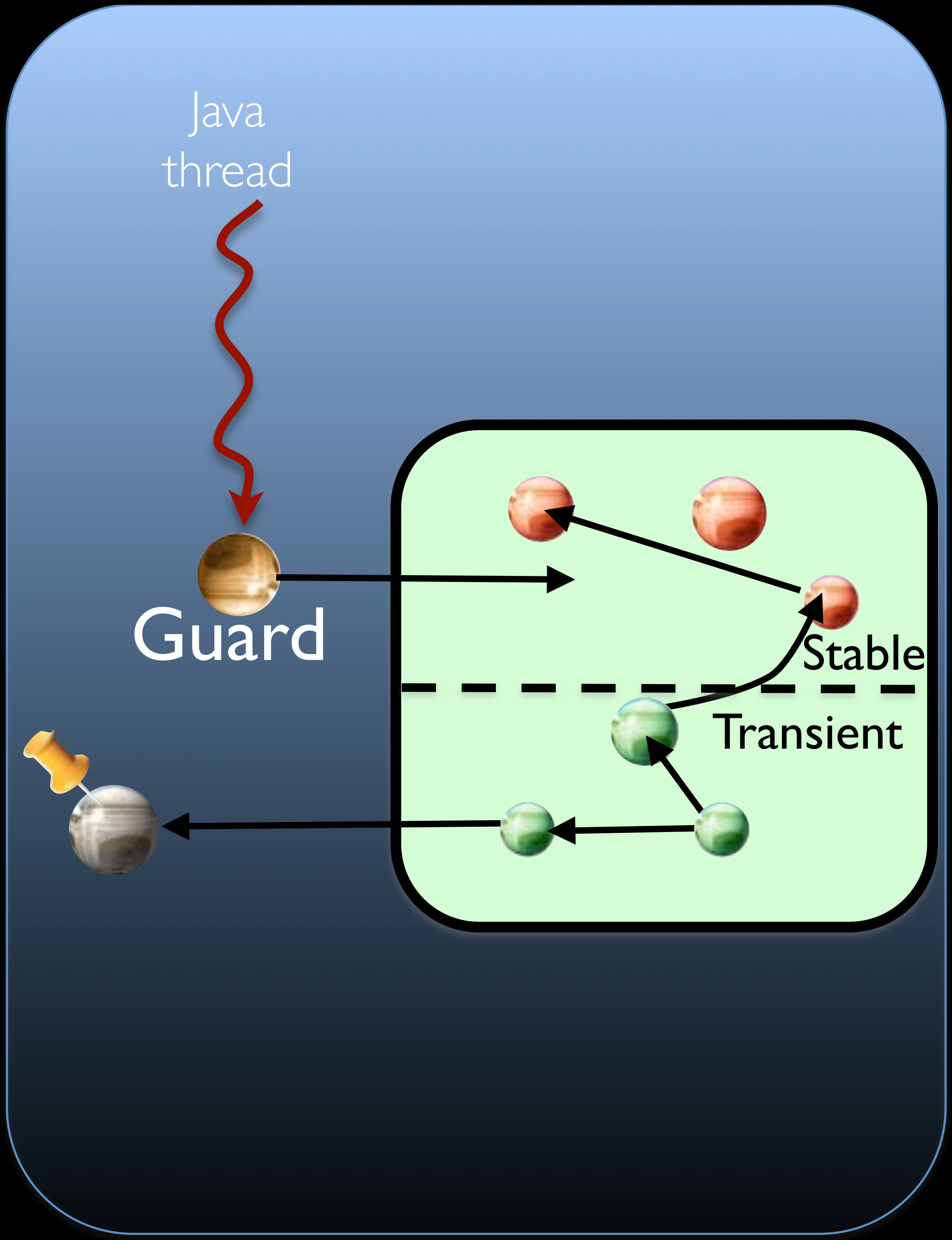
Communication with Java

- Every task has an automatically generated proxy-object
- User-defined atomic methods can be called from Java with transactional semantics
- Arguments are reference-immutable pinned objects



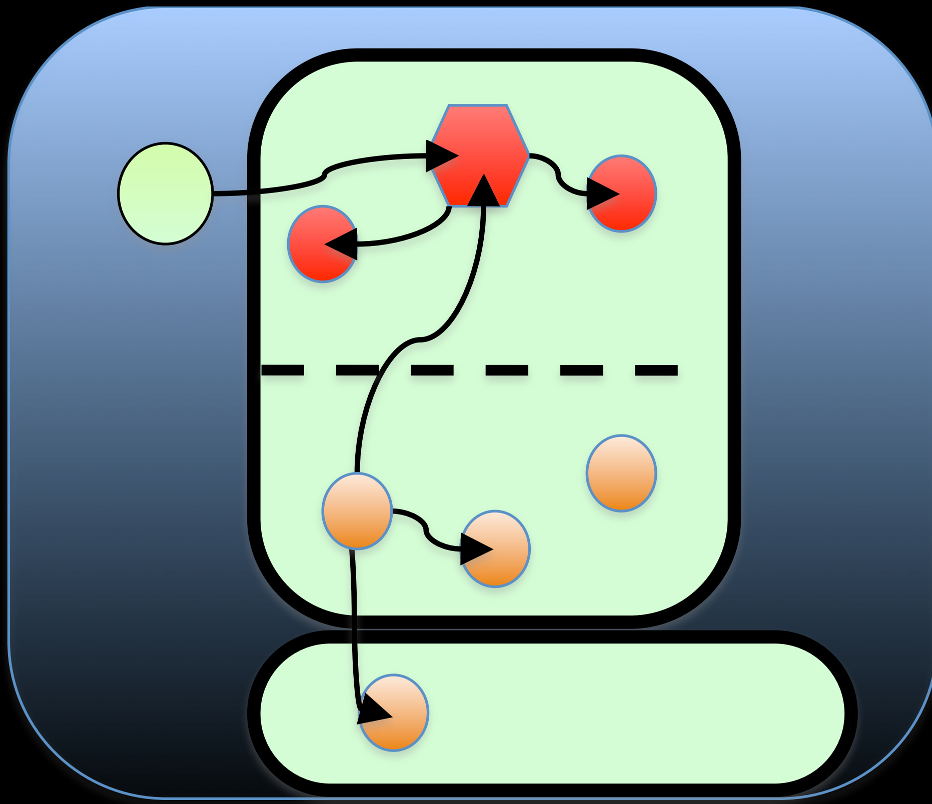
Communication with Java

- Atomic Methods:
  - acquire a lock on guard & pin all reference-immutable arguments
  - start transaction;
  - execute method
  - commit transaction
  - reclaim transient memory
  - unpin all arguments & release lock on guard
- If during execution of the method the Task is scheduled, the transaction is immediately aborted.

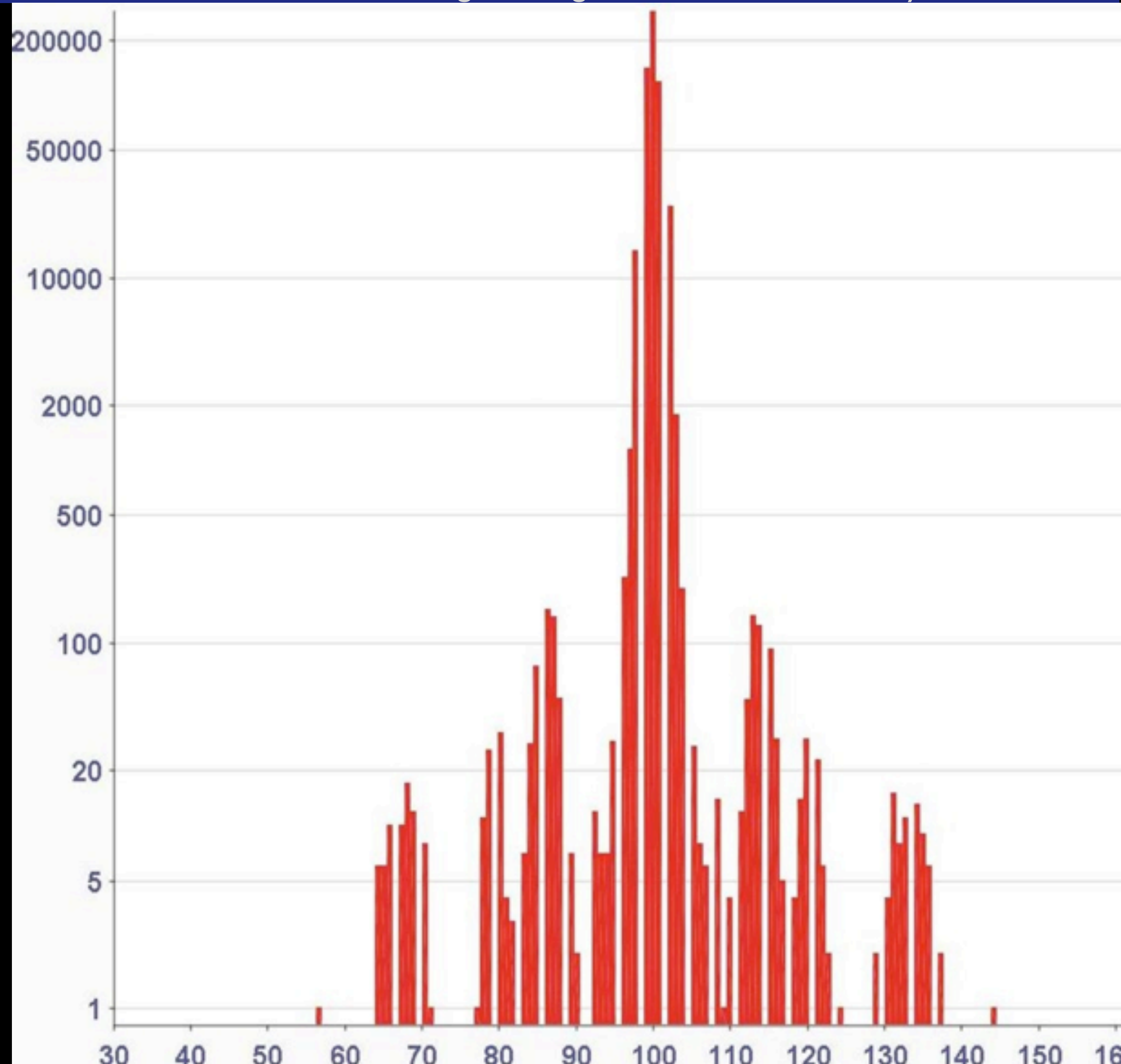


Static safety

- Safety checks prevent references to transient objects after they have been deallocated and to capsules once they have been sent.
- A simple form of ownership types is used where **`Stable`** is a marker interface for data allocated in the stable heap and **`Capsule`** for messages. Some polymorphism needed for arrays.
- Checking is done statically, no dynamic tests are need.



Predictability

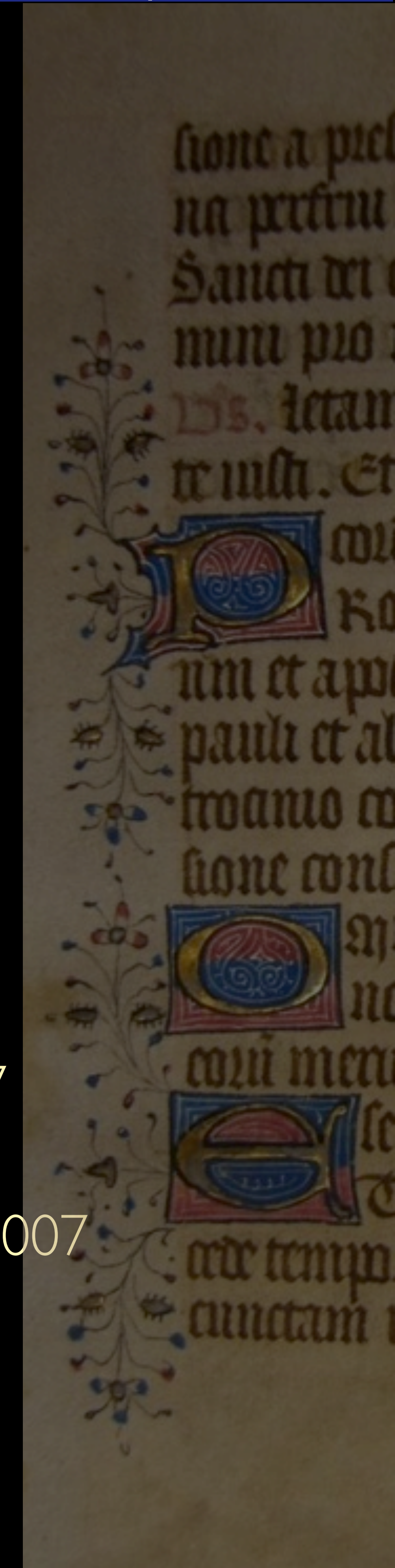


- 600K periodic invocations
- Inter-arrival time bw 57 and 144us
- 516 aborts of the atomic method

References and acknowledgements

- **Team**
  - *J. Spring, J. Auerbach, D. Bacon, F. Pizlo, R. Guerraoui, J. Manson*
- **Funding:** *NSF & IBM*
- **Availability:** *released open source by IBM on sourceforge*
- **Paper trail**

- *A Unified Restricted Thread Programming Model for Java.* **LCTES**, 2008
- *StreamFlex: High-throughput Stream Programming in Java.* **OOPSLA**, 2007
- *Reflexes: Abstractions for Highly Responsive Systems.* **VEE**, 2007
- *Scoped Types and Aspects for Real-time Java Memory management.* **RTS**, 2007
- *Scoped Types and Aspects for Real-Time Systems.* **ECOOP**, 2006
- *Preemptible Atomic Regions for Real-time Java.* **RTSS**, 2005
- *Transactional lock-free data structure for Real Time Java.* **CSJP**, 2004




4

Fiji

Safety Critical Java

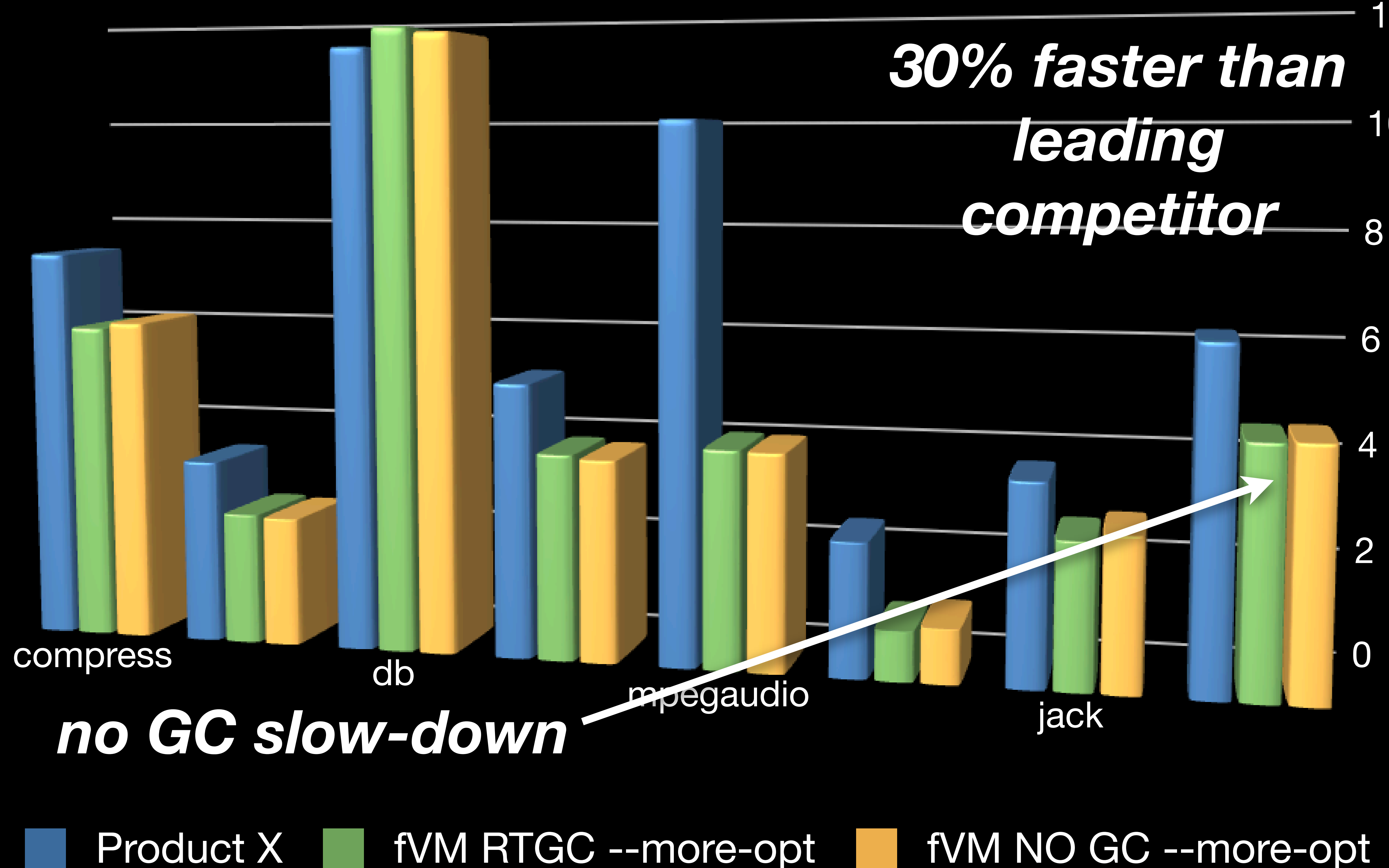
SC Java Goal

- 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
  - Based on HIJA – High-Integrity Java Application (EU project)
  - **Final draft due this year (already 300+ page book)**

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

Execution time vs. Competitor RTJVM



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 Integer[1000] array, fill with Integer instances
  - Allocate 1000 more 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 +
                   "\nNumber of timer interrupts when GC running: "+iWGC +
                   "\resMax interrupt exec time with GC: "+ mDWGC);
     System.out.println(res);
     Thread.sleep(1000);
} } }
```

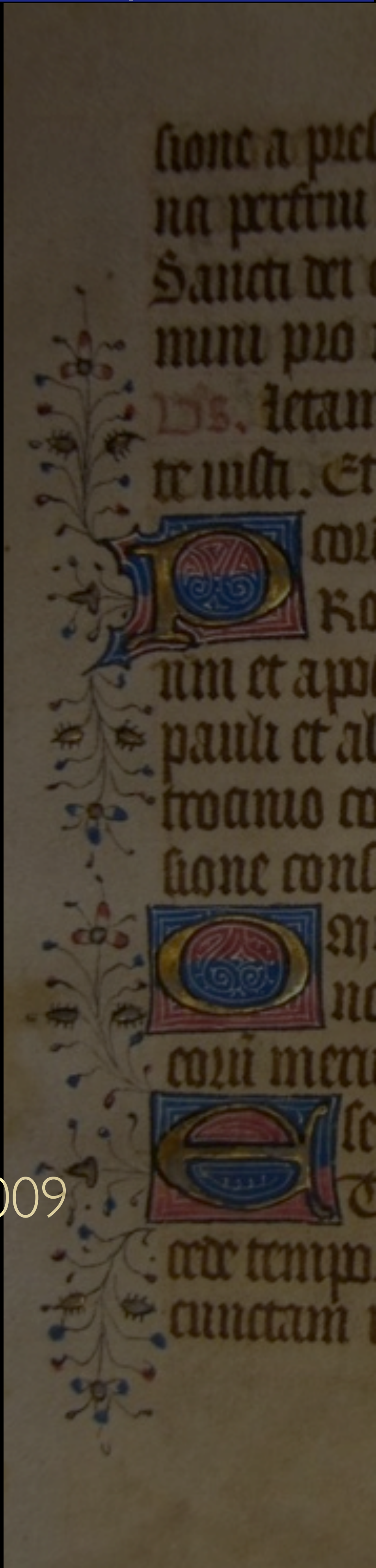
```
class Demo {

  static void main(String[] v) {
   final Timer t = new Timer();
   t.fireAfter(10, new Runnable(){
       public void run() {
         long before = HardRT.getCPUTimestamp();
         if (GC.inProgress()) iterationsWGC++;
         arr = new Integer[1000];
         for (int i=0;i<arr.length;++i)
             arr[i] = new Integer(i);
         t.fireAfter(10, this);
         ...
   }});
   ...
  }
```

```
t = new Timer();
t.fireAfter(10,
  new Runnable(){ void run(){
    long before=getCPUTimestamp();
    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);
    ...
}});
```

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**

- *Real-time Java in Space: Potential Benefits and Open Challenges.* DASIA, 2009
- *A Technology Compability Toolkit for Safety Critical Java.* 2009




5

Conclusion

- 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`