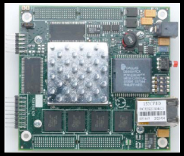
program the

99%

Objectives of this talk



After almost a **decade** working on real-time Java

- ***Self-contained*** overview of Real-time Garbage Collection

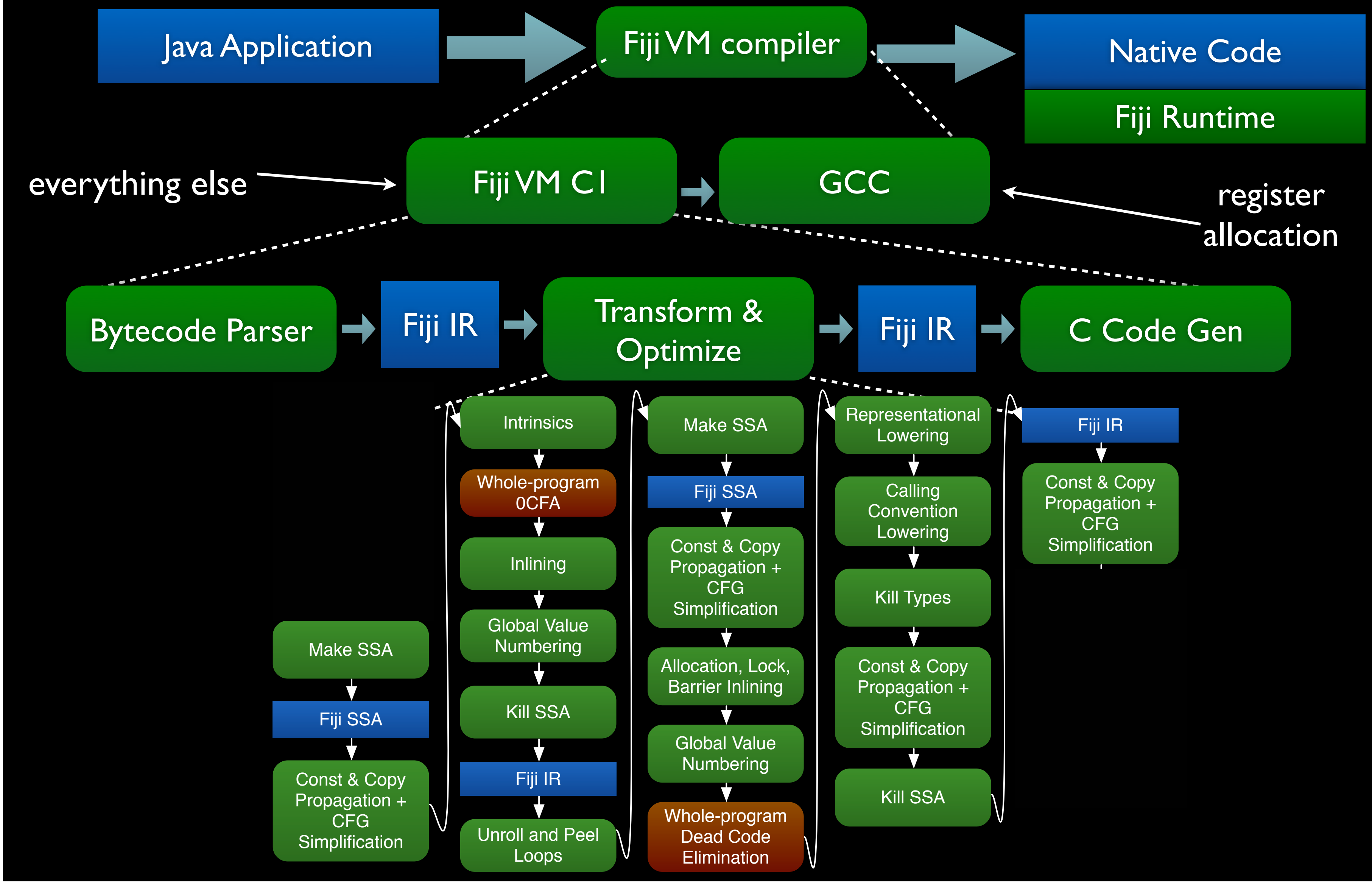
- Highlight results from **Filip Pizlo**'s PhD thesis
  [**PLDI**'10, **EUROSYS**'10, **RTSS**'09, **ECOOP**'09, **ISMM**'08, **PLDI**'08, **ISMM0**'7, **LCTES**'07, **CC**'07, **RTAS**'06]

Expectations

- A managed language should be <2x slower than C
- Real-time support should cost <2x
- Worst case performance matters

Reality

After 10 years of work… FijiVM



Java Application

Fiji VM compiler

Native Code

Fiji Runtime

everything else

Fiji VM C1

GCC

register allocation

Bytecode Parser

Fiji IR

Transform & Optimize

Fiji IR

C Code Gen

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

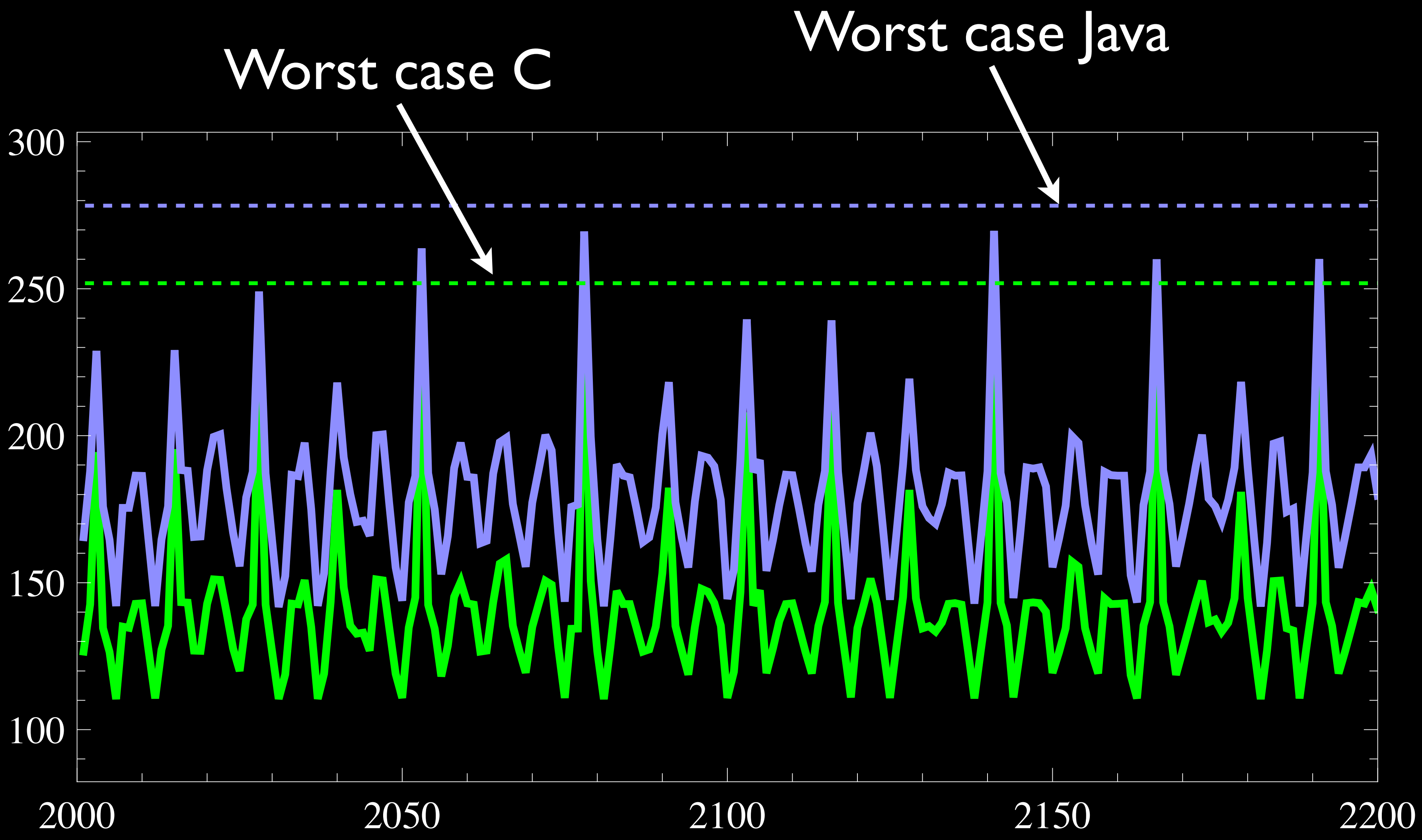
Kill SSA

Fiji IR

Const & Copy Propagation + CFG Simplification

Reality

- Real-time benchmark
  - Aircraft collision avoidance w. simulated radar frames
  - CDc - idiomatic C
  - CDj - idiomatic Java
- Real-time platform
  - RTEMS 4.9.1 (hard RTOS)
  - 40MHz LEON3, 64MB RAM (radiation-hardened SPARC)

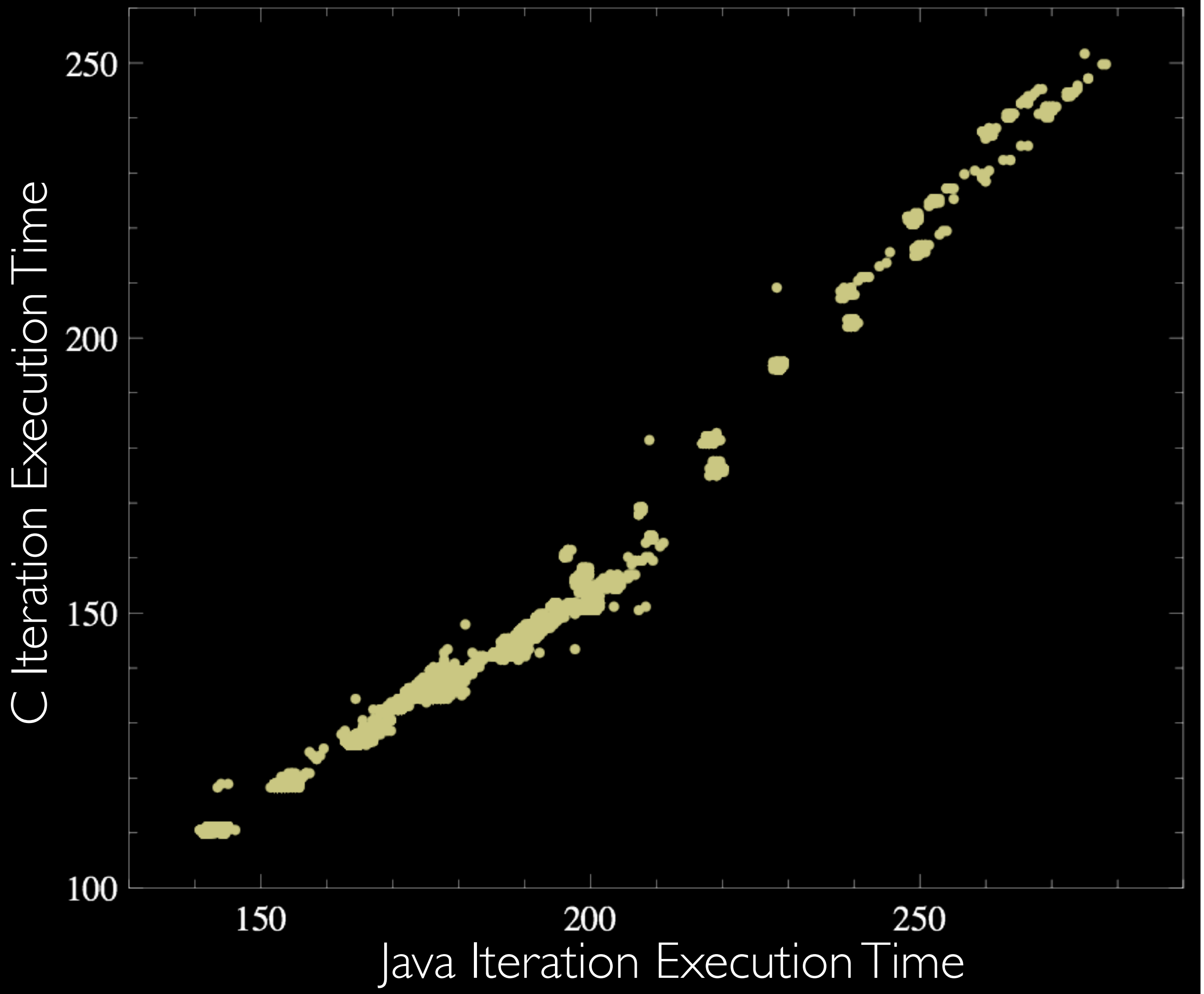


Worst case C

Worst case Java

Frame Number vs. Execution Time (ms)

Correlation C/Java



100K samples
15 GC cycles

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

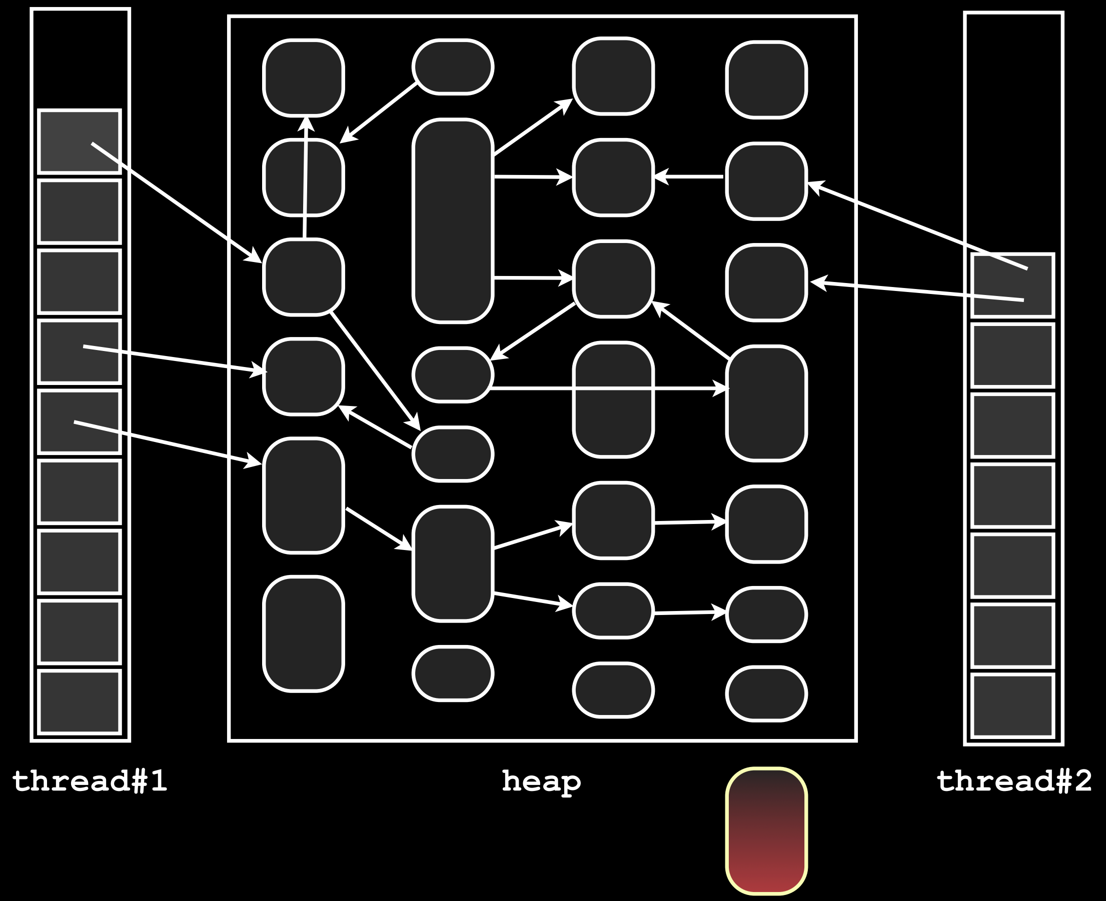
```
s = new SizeEstimator();
s.reserve(Decrypt.class, 2);
…
shared = new LTMemory(s.getEstimate());
shared.enter(new Run(){ public void run(){
    ...d1 = new Decrypt() ...
}});
```

1

GC is easy*

* good performance is hard

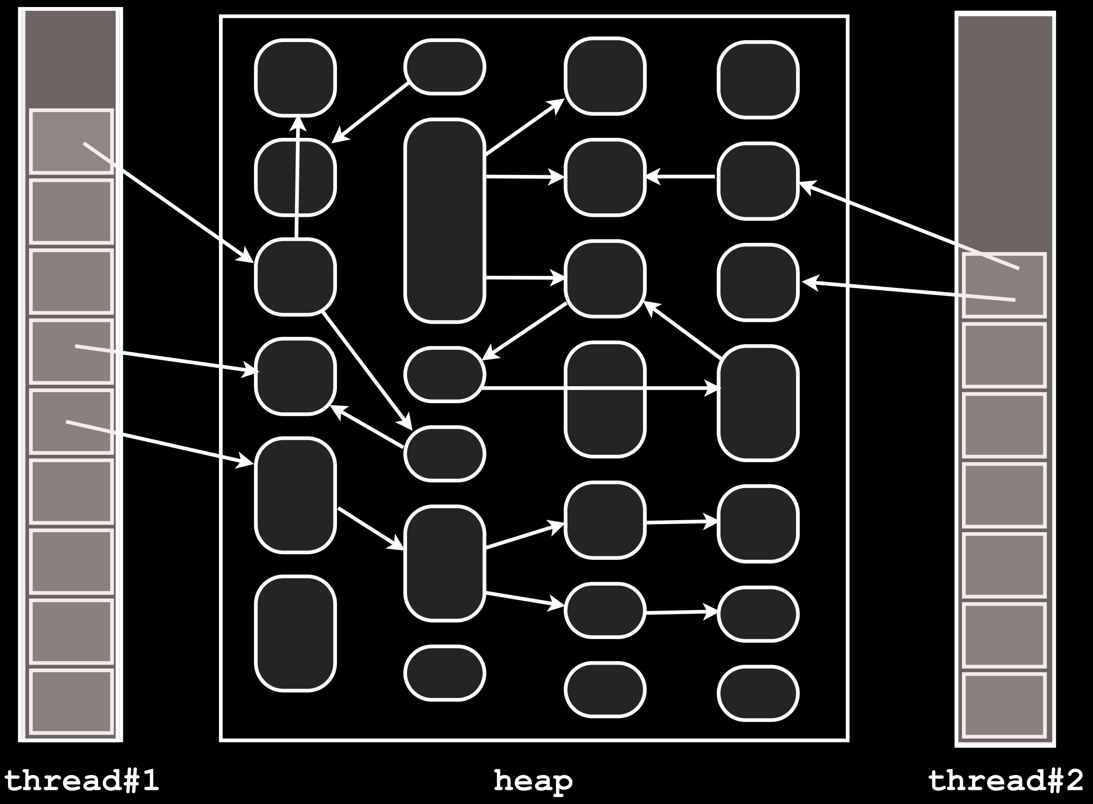
Garbage Collection: Mark & Sweep



Phases

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

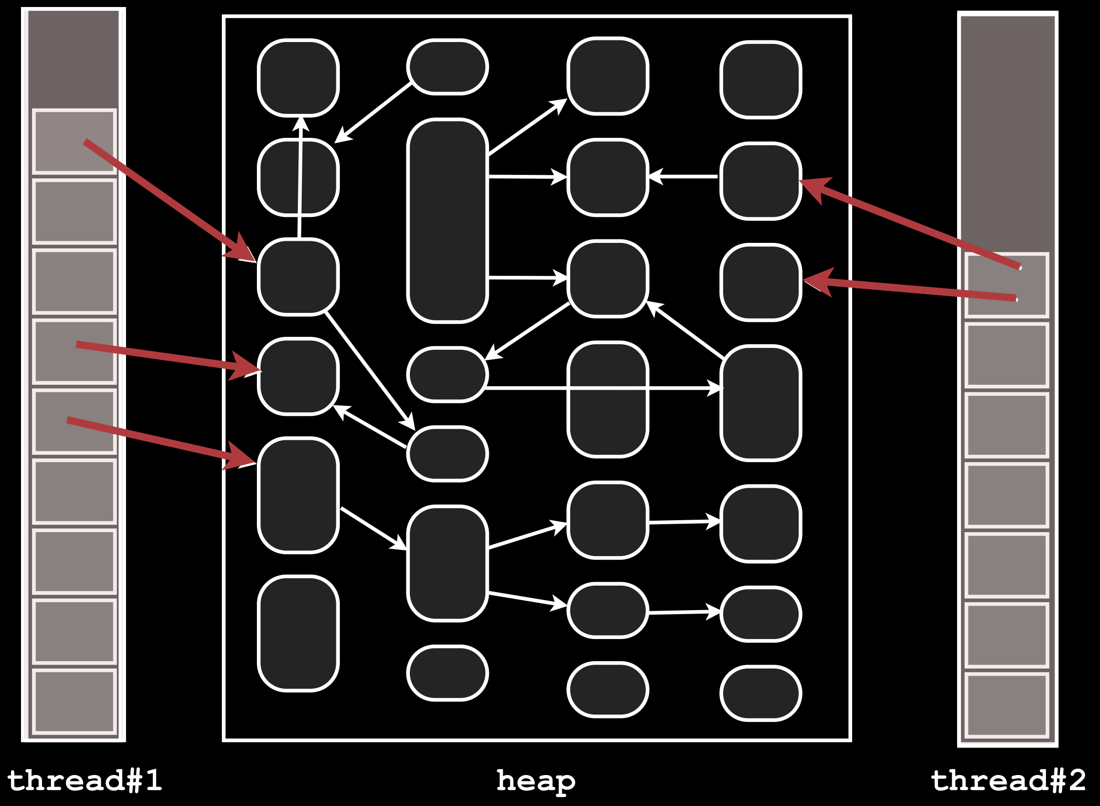
Garbage Collection



Phases

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

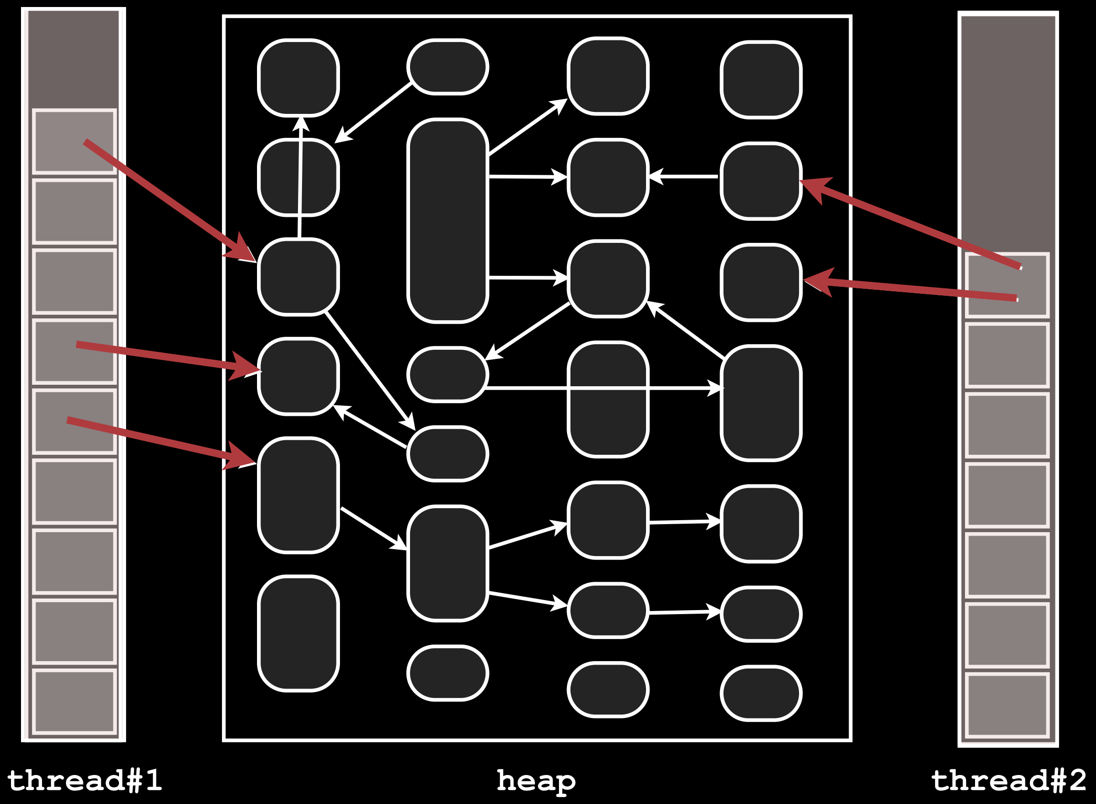
Garbage Collection



Phases

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

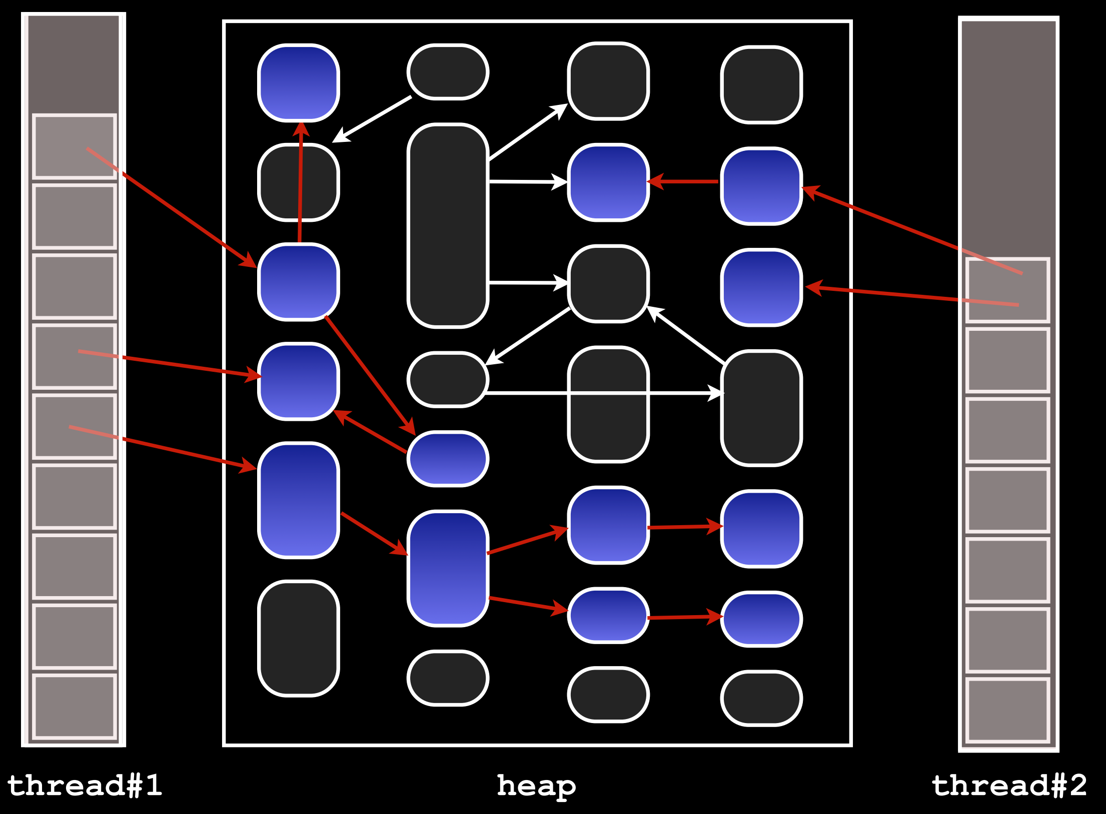
Garbage Collection



Phases

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

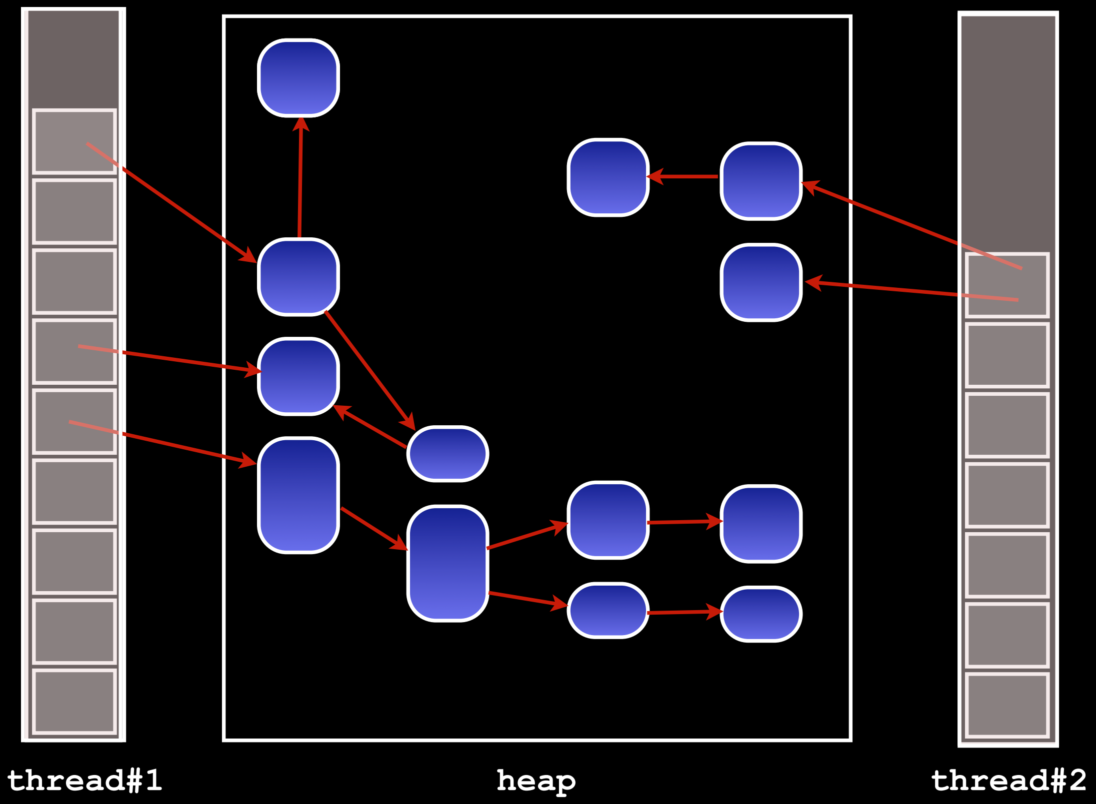
Garbage Collection



Phases

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

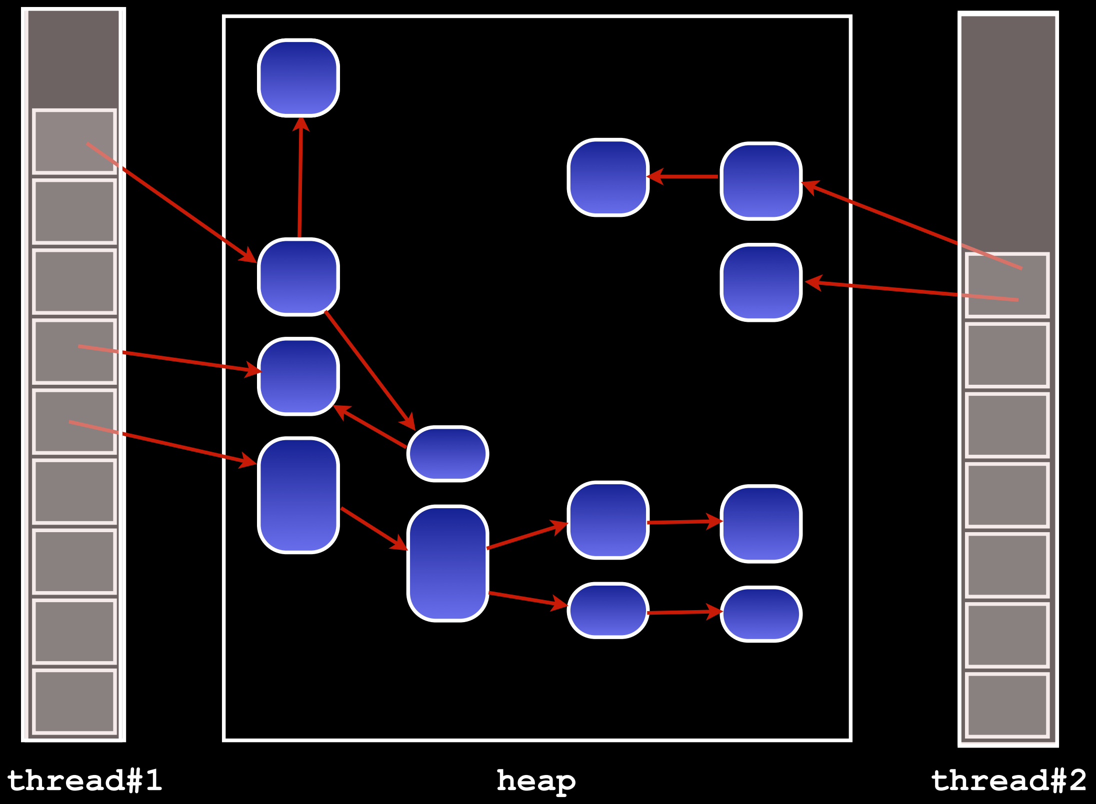
Garbage Collection



Phases

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

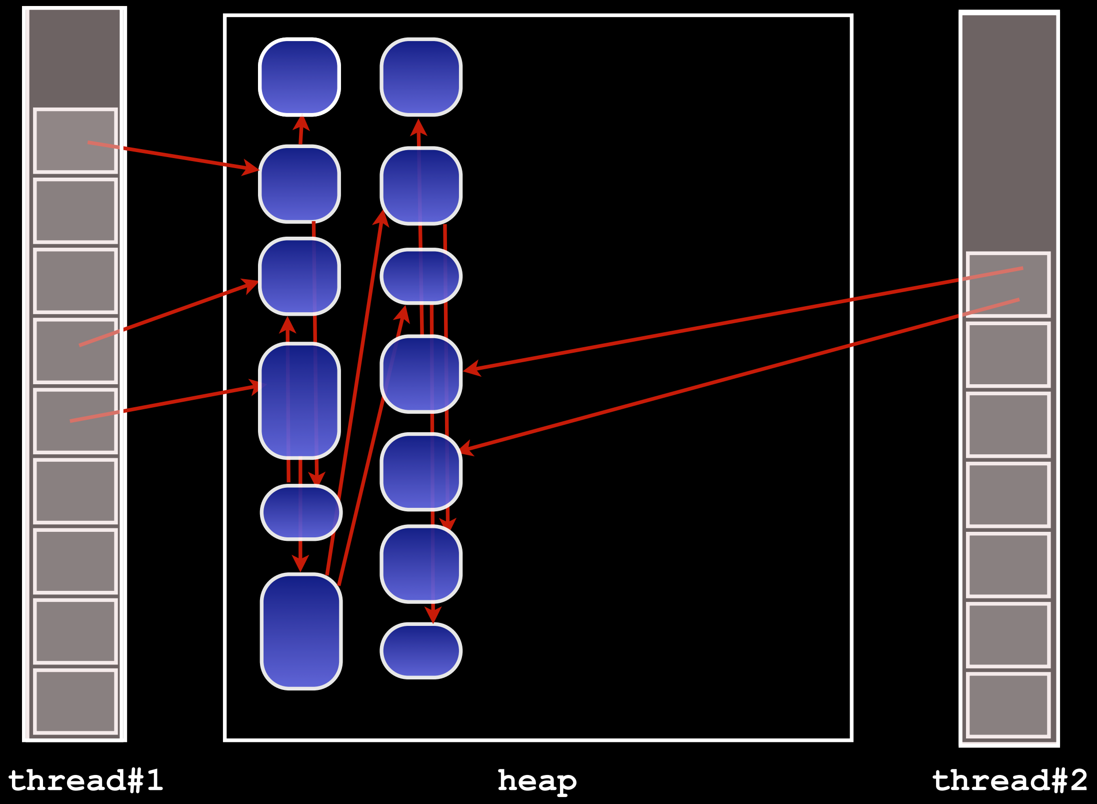
Garbage Collection



Phases

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

Garbage Collection



Phases

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

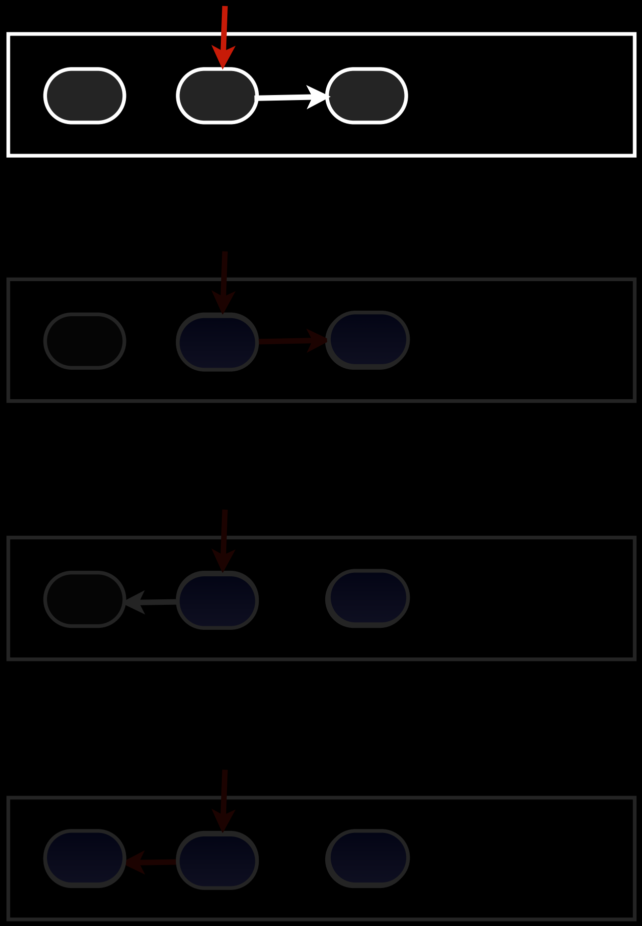
2


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

RTGC is easy*

* good performance is *harder*

Incrementalizing marking

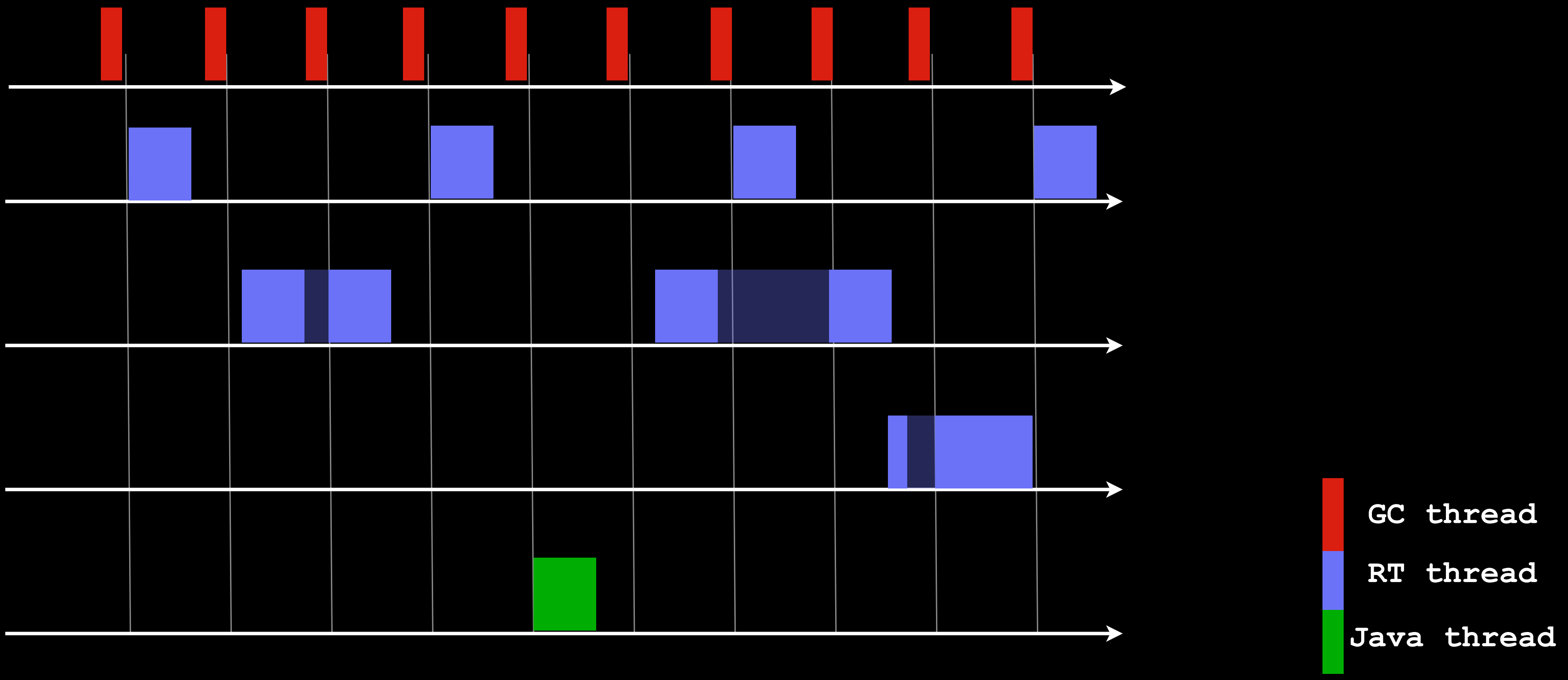


Collector marks object

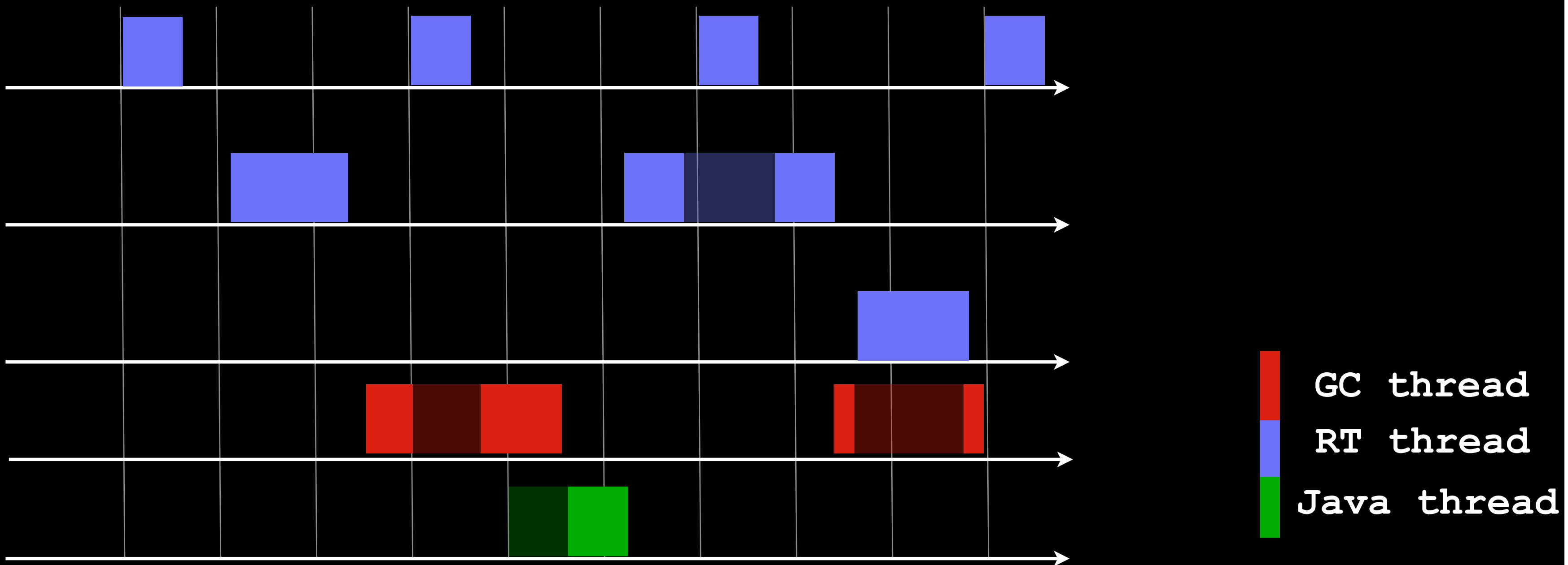
Application updates reference field

Compiler inserted write barrier marks object

Time-based GC Scheduling



Slack-based GC Scheduling




3

Compaction is easy*

* that’s a lie

State of the art

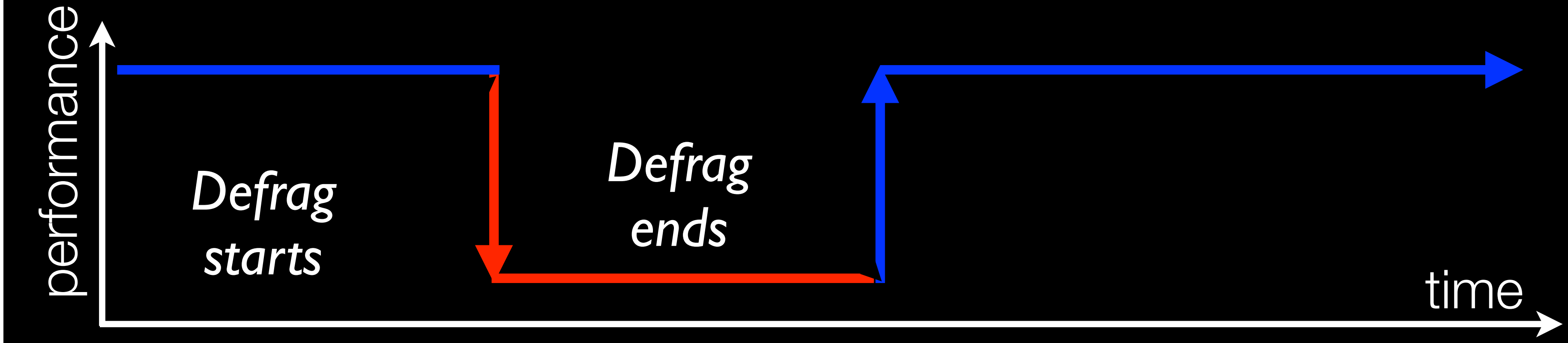
- Oracle HotSpot
  - fast & space bounded
  - *but blocking*
- Oracle Java RTS
  - space bounds, concurrent, wait-free
  - *but 60% slow-down*
- IBM Websphere SRT
  - 30% slow-down, concurrent, wait-free
  - *but susceptible to fragmentation*

Minimizing fragmentation

Previous Work

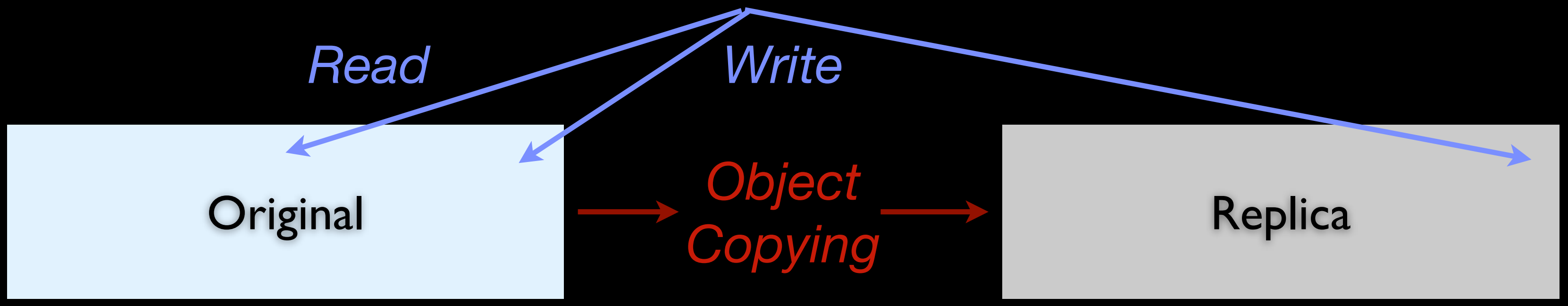
On-demand Defragmentation

- Concurrent defragmentation has draw-backs
    - slow down during defrag more than 5x [Pizlo07,Pizlo08]



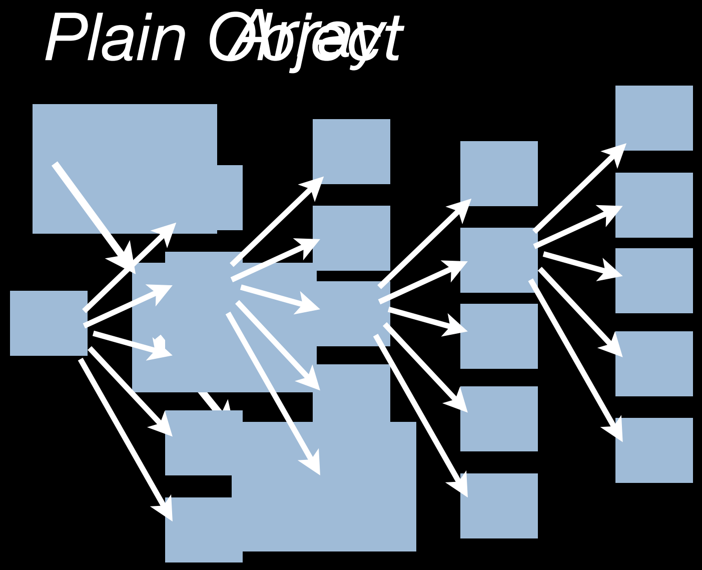
Replication-based GC

- Allows concurrent defragmentation [NettlesOToole93, ChengBlelloch01]
- Two spaces: one space for reads; writes "replicated" to both
- … but writes not atomic



Fragmented allocation

- All objects split into small fragments [Siebert'99]
- Fragment size is fixed at 32 bytes
- Fragments are linked, application follows links on reads



Access cost is known statically, does not vary.

Access cost is logarithmic.

Most objects require only two fragments.

Schism

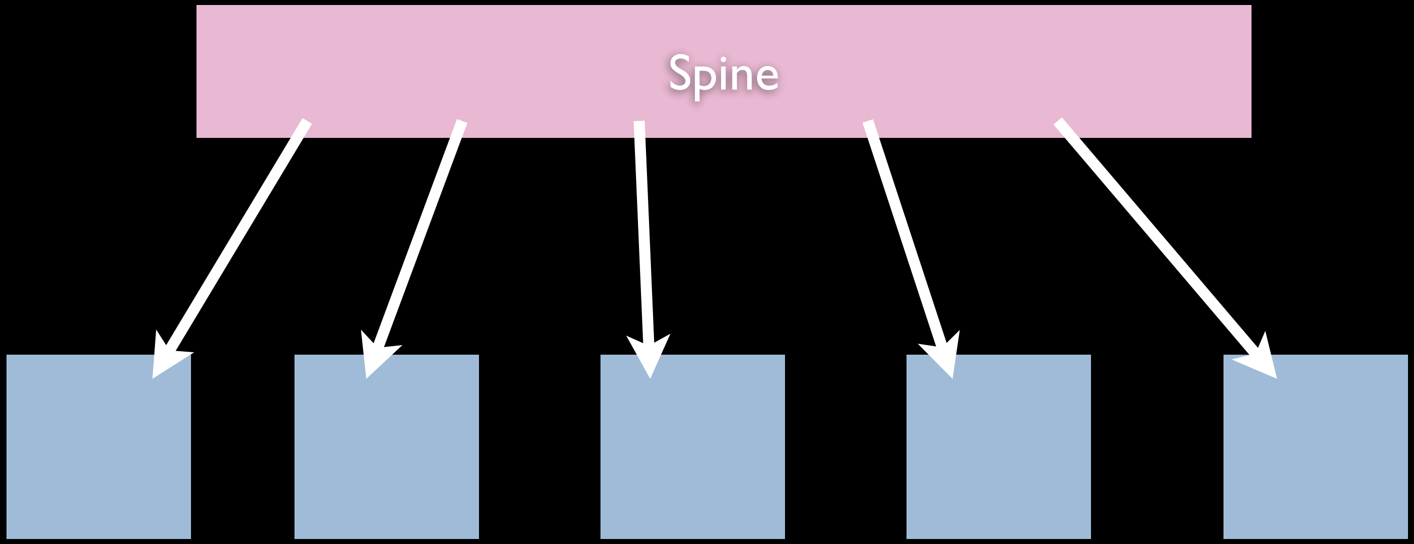
[PLDI'10]

Schism = CM&S + Replication + Fragments

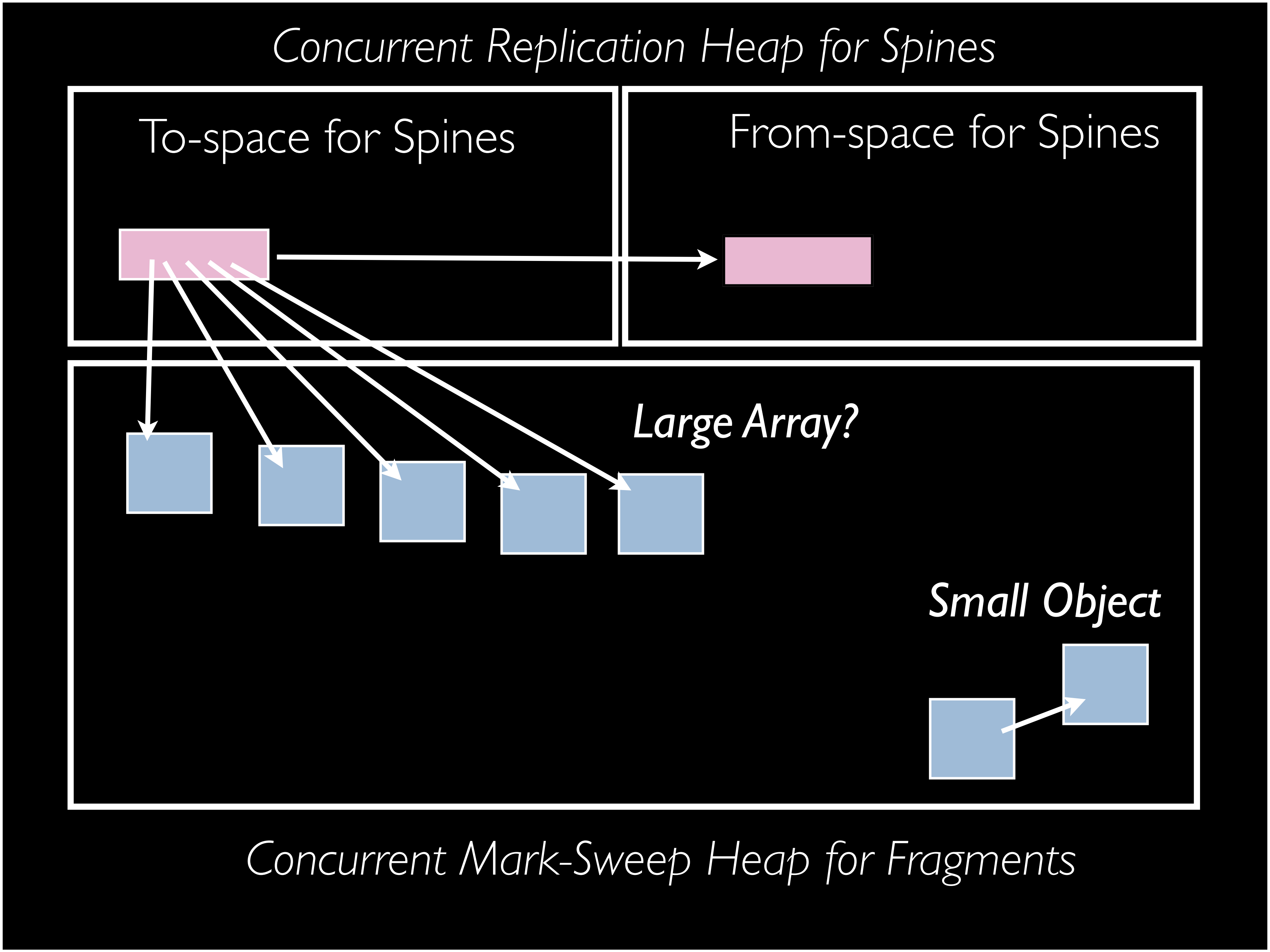
- Insight:
  - replicated collectors are good immutable data
  - fragmented allocation works well for fixed-size data
- Combination:
  - Concurrent mark-sweep for fixed-size fragments
  - Replication for array spines
- No external fragmentation, O(1) heap access, wait-free & coherent

Arrays

Index in a variable sized spine… which is immutable



Data in fixed size fragments



Concurrent Replication Heap for Spines

To-space for Spines

From-space for Spines

Large Array?

Small Object

Concurrent Mark-Sweep Heap for Fragments

Proof ?

Tunable throughput/predictability trade-off

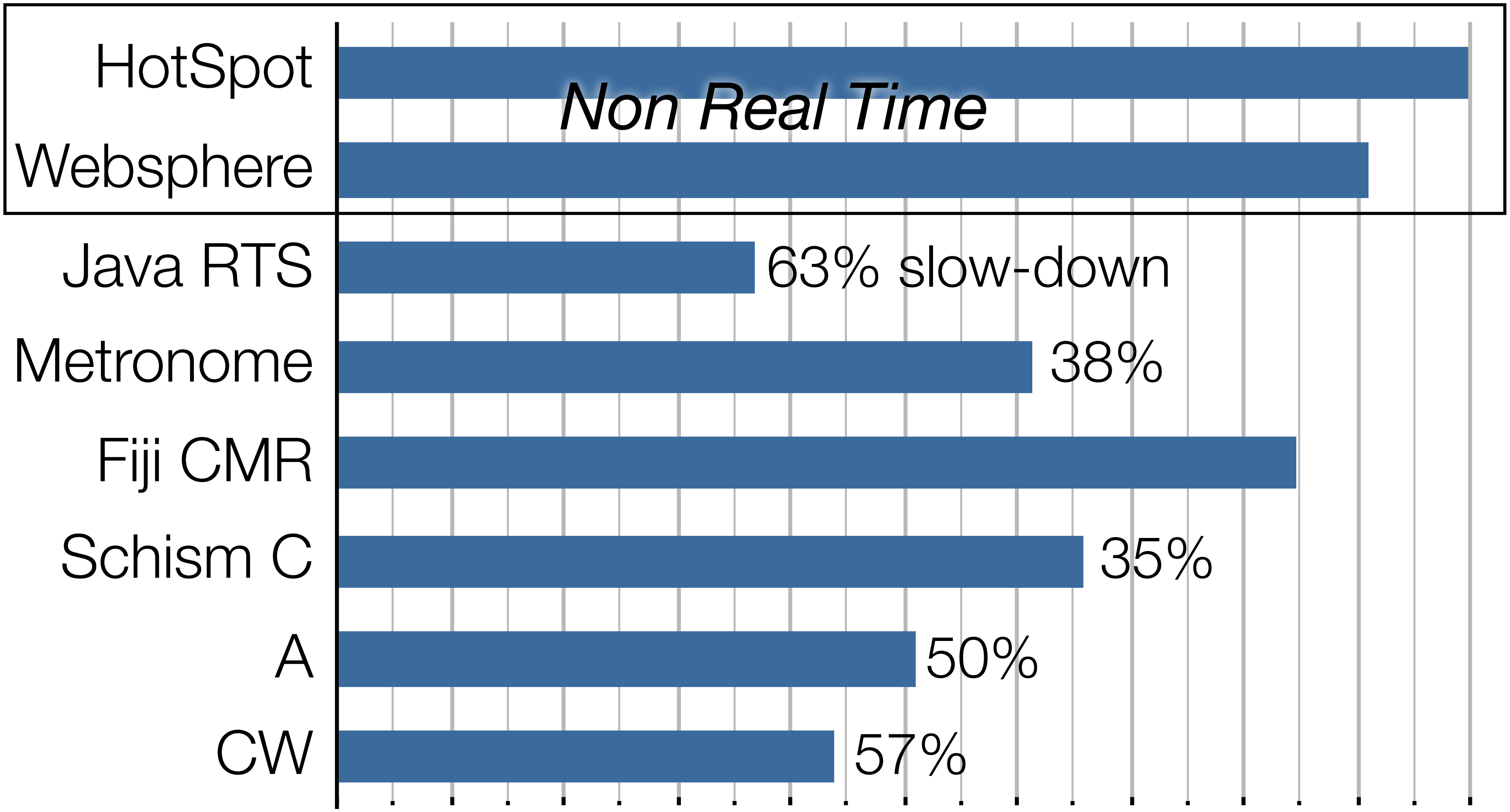
- **A** deterministic
  - allocate fragmented
- **C** throughput
  - allocate contiguously if possible
- **CW** worst-case for level C
  - poison all fast-paths (array accesses, write barriers, allocations)

Summary of Results

- **Goal: fast**
- Goal: fragmentation tolerant
- Goal: deterministic

SPECjvm98

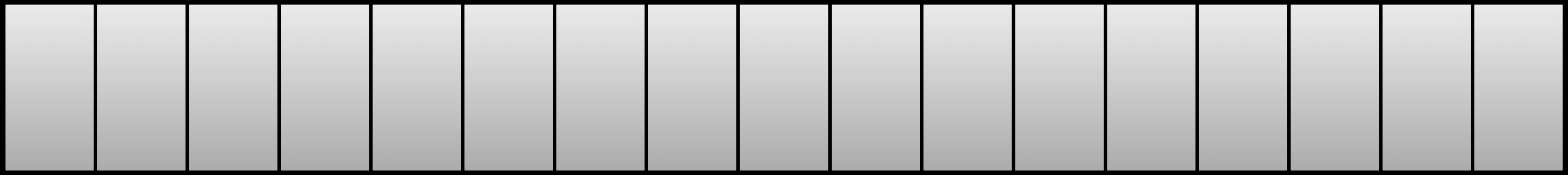
(50MB heap)



Summary of Results

- *Goal: fast* ✓
- **Goal: fragmentation tolerant**
- Goal: deterministic

Torture tests



% free memory allocated under fragmentation

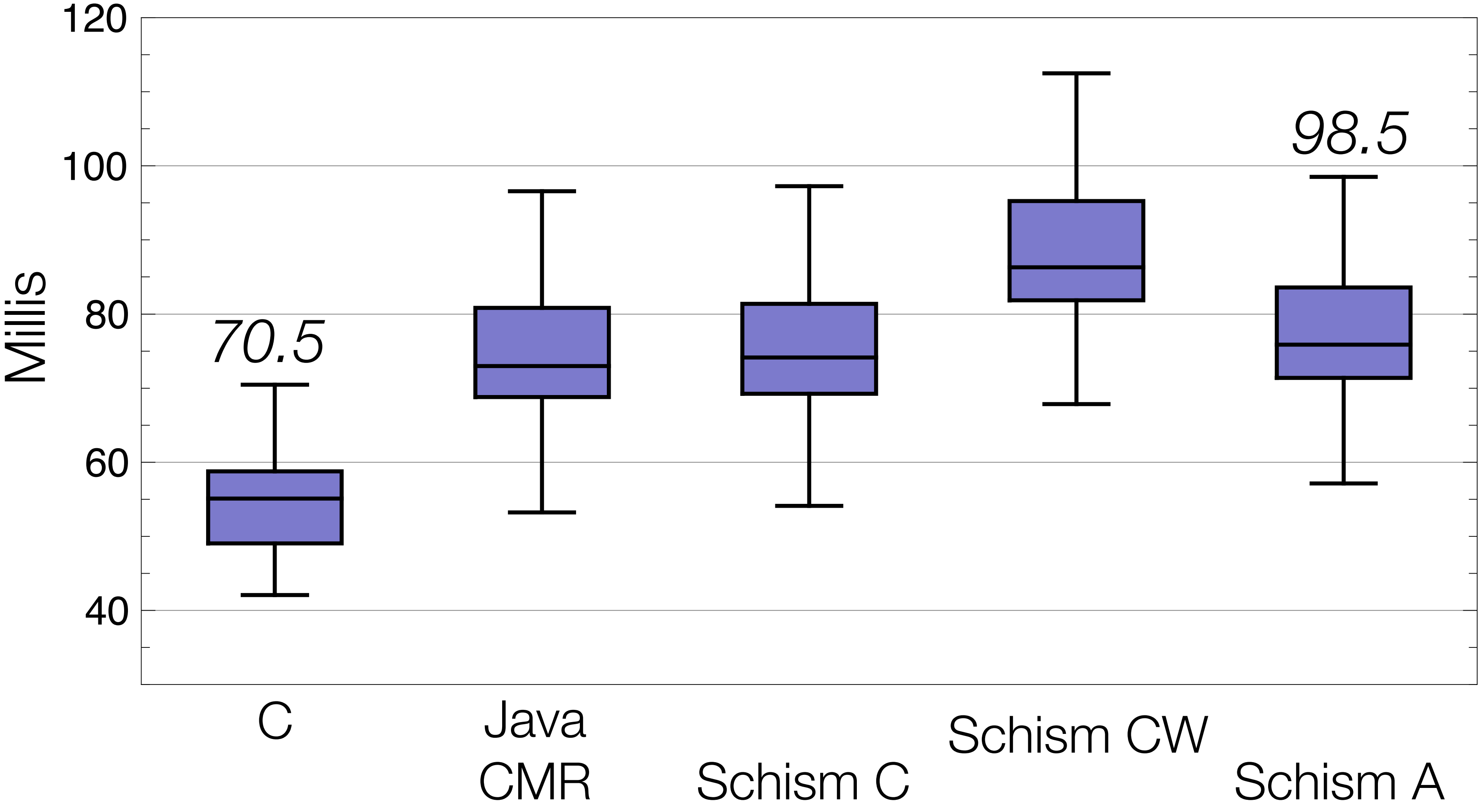
- HotSpot: 100%
- Java RTS: ~80%
- Metronome: ~1%
- Schism: 100%

Summary of Results

- *Goal: fast* ✓
- *Goal: fragmentation tolerant* ✓
- **Goal: deterministic**

Java vs C on **CDx**

< 40% slower than C
as deterministic



References and acknowledgements

- Team
  - *F Pizlo, E Blanton, L Ziarek,
    T Kalibera, T Hosking, P Maj,
    T Cunei, M Prochazka, J Baker*
- Paper trail

- *Schism: Fragmentation-Tolerant Real-Time Garbage Collection.* PLDI10
- *High-level Programming of Embedded Hard Real-Time Devices.* EUROSYS10
- *Accurate Garbage Collection in Uncooperative Environments.* CCP&E09
- *A Study of Concurrent Real-time Garbage Collectors.* PLDI08
- *Memory Management for Real-time Java: State of the Art.* ISORC08
- *Hierarchical Real-time Garbage Collection.* LCTES07