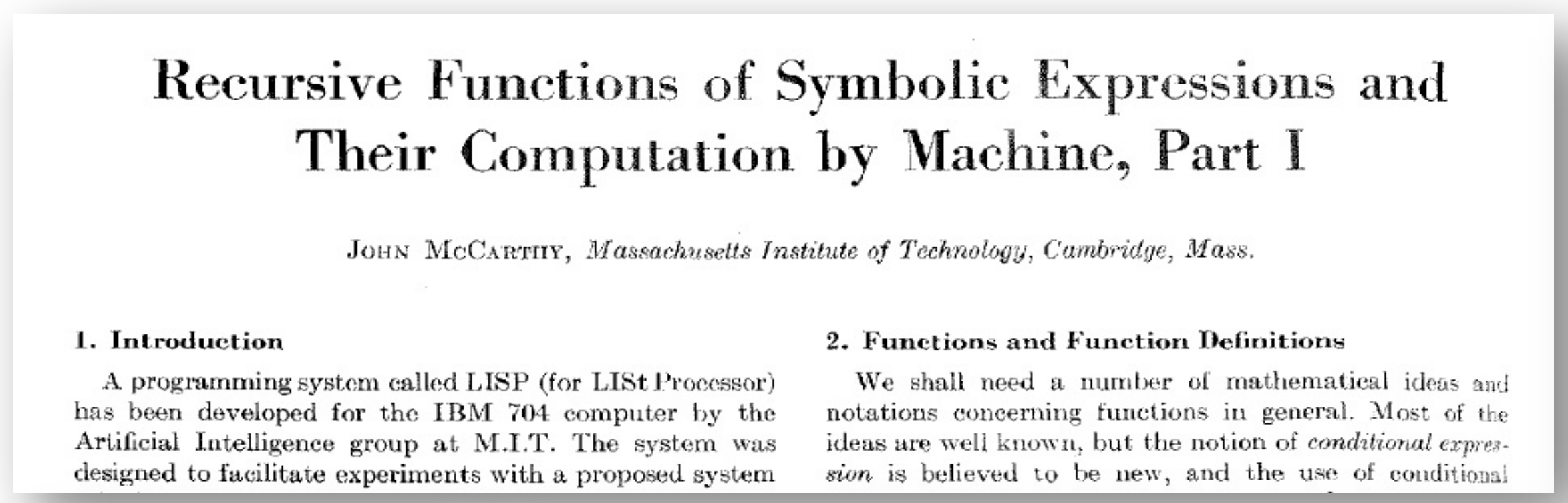
Taking Off the Gloves with Reference Counting Immix

Rifat Shahriyar†, Steve Blackburn†, Xi Yang† and Kathryn McKinley$^{\delta}$

Garbage collection (GC) is Ubiquitous

- Born 53 years ago



Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

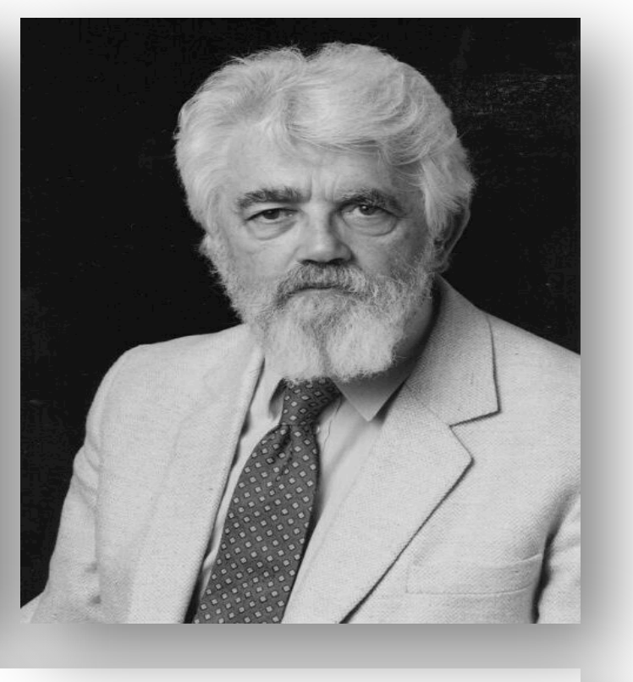
John McCarthy, *Massachusetts Institute of Technology, Cambridge, Mass.*

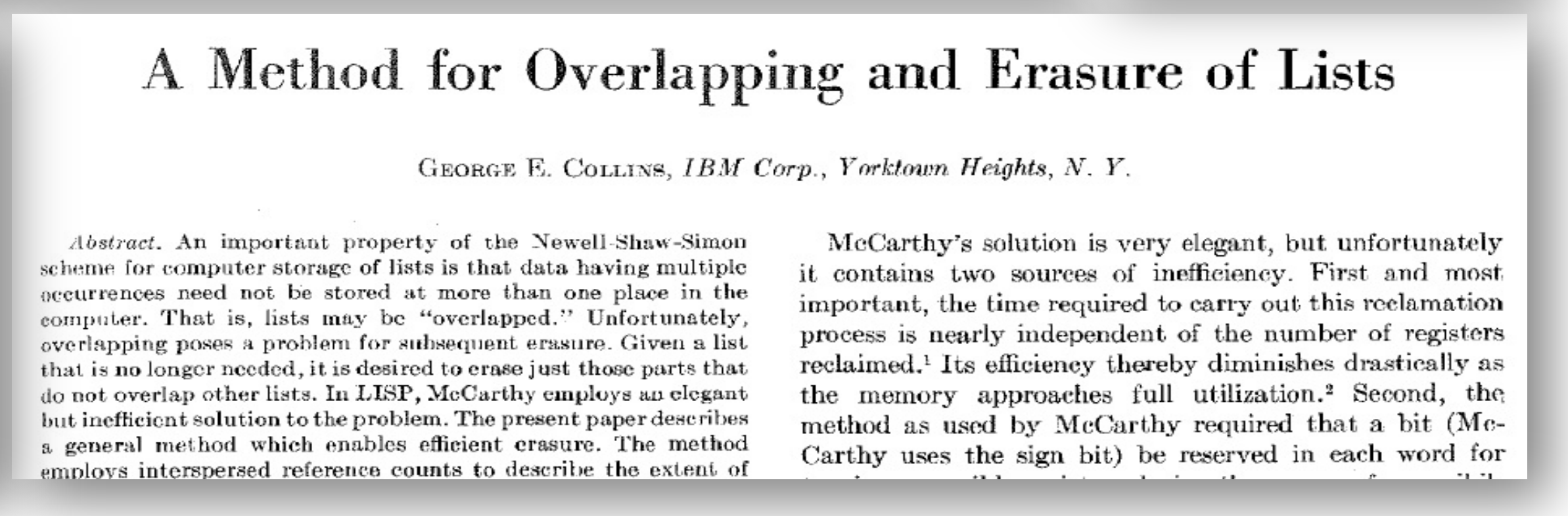
1. Introduction

A programming system called LISP (for LISt Processor) has been developed for the IBM 704 computer by the Artificial Intelligence group at M.I.T. The system was designed to facilitate experiments with a proposed system

2. Functions and Function Definitions

We shall need a number of mathematical ideas and notations concerning functions in general. Most of the ideas are well known, but the notion of *conditional expression* is believed to be new, and the use of conditional





A Method for Overlapping and Erasure of Lists

George E. Collins, *IBM Corp., Yorktown Heights, N. Y.*

Abstract. An important property of the Newell-Shaw-Simon scheme for computer storage of lists is that data having multiple occurrences need not be stored at more than one place in the computer. That is, lists may be "overlapped." Unfortunately, overlapping poses a problem for subsequent erasure. Given a list that is no longer needed, it is desired to erase just those parts that do not overlap other lists. In LISP, McCarthy employs an elegant but inefficient solution to the problem. The present paper describes a general method which enables efficient erasure. The method employs interspersed reference counts to describe the extent of

McCarthy's solution is very elegant, but unfortunately it contains two sources of inefficiency. First and most important, the time required to carry out this reclamation process is nearly independent of the number of registers reclaimed.[1] Its efficiency thereby diminishes drastically as the memory approaches full utilization.[2] Second, the method as used by McCarthy required that a bit (McCarthy uses the sign bit) be reserved in each word for

- Two ideas underpin large literature:
  - Tracing [McCarthy60]
  - Reference Counting [Collins60]
- However
  - ✔ Tracing used in all high performance GCs
  - ✔ Reference counting (RC) has interesting advantages
  - ✘ Reference counting only in non-performance critical settings

Status of Reference Counting

- **High performance reference counting**
  - ✔ Significantly faster than naïve RC
  - ✘ 30% slower than MS (well tuned simple tracing)
  - ✘ 40% slower than GenImmix (production collector in Jikes RVM)
- **Reference counting was improved [ISMM12]**
  - ✔ Deferred and coalesced limited bit RC with new object optimization
  - ✔ Performs same as MS
  - ✘ But 10% slower than GenImmix

Allocator

- **Contiguous allocator**
  - ✔ Better cache locality
  - ✔ Fewer instructions per allocation
- **Free list allocator**
  - ✔ Suitable for RC
  - ✘ Poor cache locality
  - ✘ Higher instructions per allocation
  - ✘ Suffers from both internal and external fragmentation

† Australian National University $^{\delta}$ Microsoft Research

Motivating Analysis

GC	Allocator	Mutator time	Instruction retired	Cache miss	Mutator locality
Immix	Contiguous	1.00	1.00	1.00	✔
MS	Free list	1.09	1.07	1.27	✘
RC	Free list	1.12	1.12	1.31	✘
SS	Contiguous	1.01	1.00	0.97	✔

Contributions

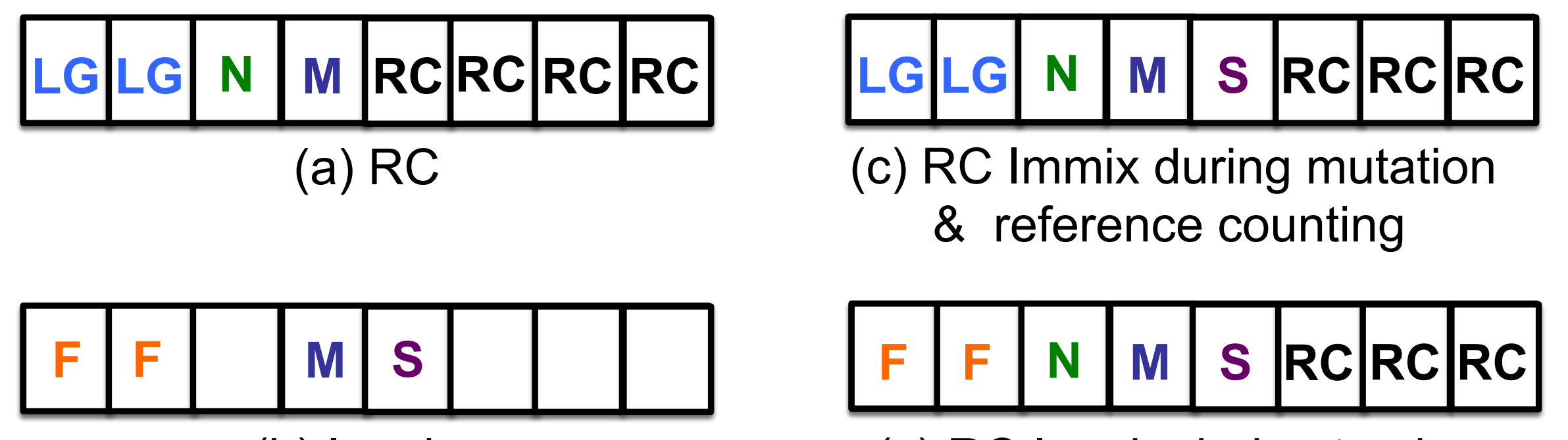
- ✔ Identify heap organization as performance bottleneck for RC
- ✔ Merge RC with Immix - RCImmix
- ✔ Eliminate fragmentation by integrating copying with RC
- ✔ RCImmix achieved great performance, 3% faster than fastest production

Challenges of RCImmix

- Adapt Immix line/block reclamation strategy to RC context
- Share limited header bits to satisfy both RC and Immix
- Defragment in RC context to eliminate fragmentation

How RCImmix works

- **Immix heap organization**
  - Contiguous allocation into regions (lines and blocks)
  - Mark objects and their region, unmarked regions can be freed
- **Reference counting collection**
  - Reference count for each object, live object count for each line
  - Collect lines with no live objects
- **Cycle collection**
  - Mark objects and their lines, sweep to collect unmarked lines
  - Restore stuck object counts and correct incorrect line counts
  - Sweep dead lines instead of sweep dead objects
- **Defragmentation**
  - Proactively copies surviving new objects with bounded copy reserve
  - Copy reserve using line survival rate without any overhead
  - Reactively with cycle collection based on some statistics and threshold
  - Both copies opportunistically and stops when available space exhausted
- **Header Bits**



(a) RC

(c) RC Immix during mutation & reference counting

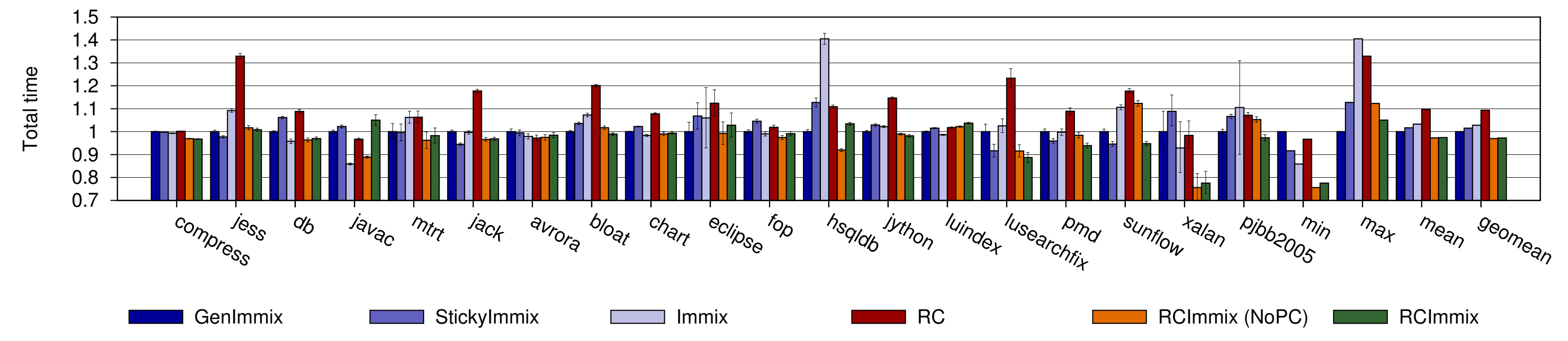
(b) Immix

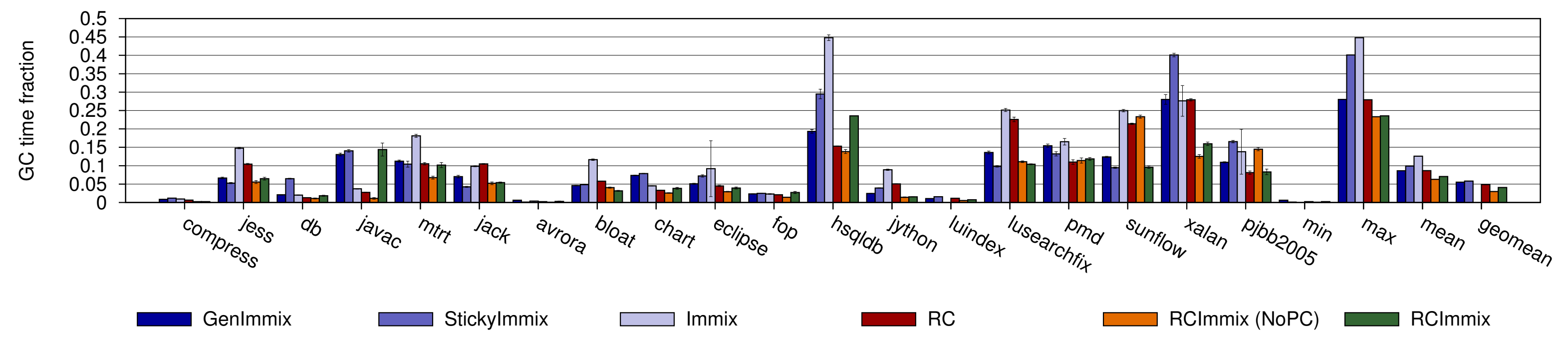
(c) RC Immix during tracing

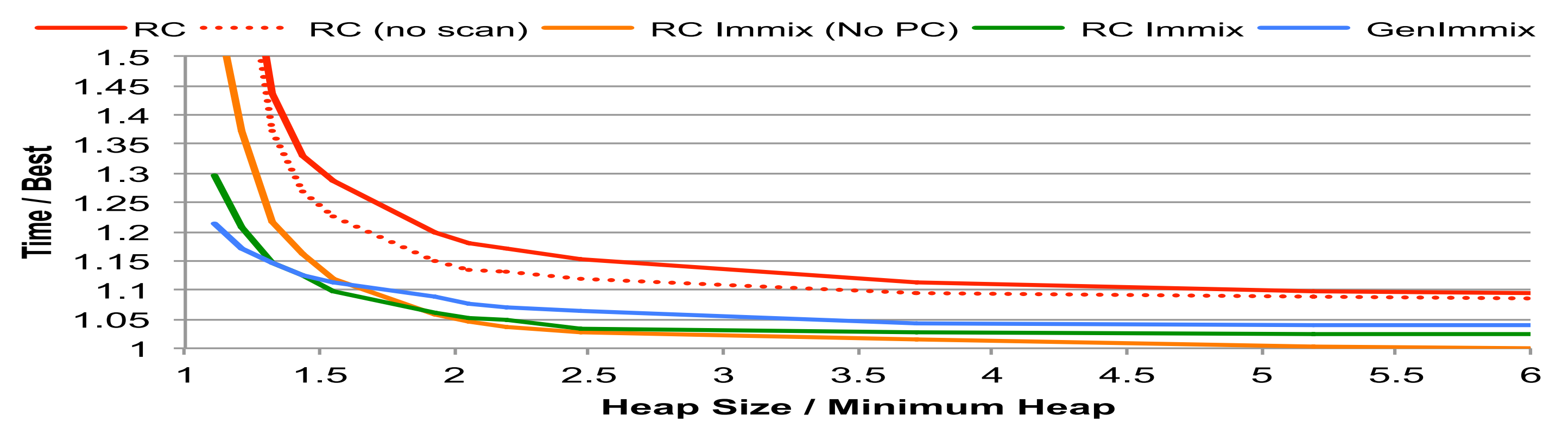
LG: Logged, M: Marked, N: New, RC: Count, F: Forwarded, S: Spans line

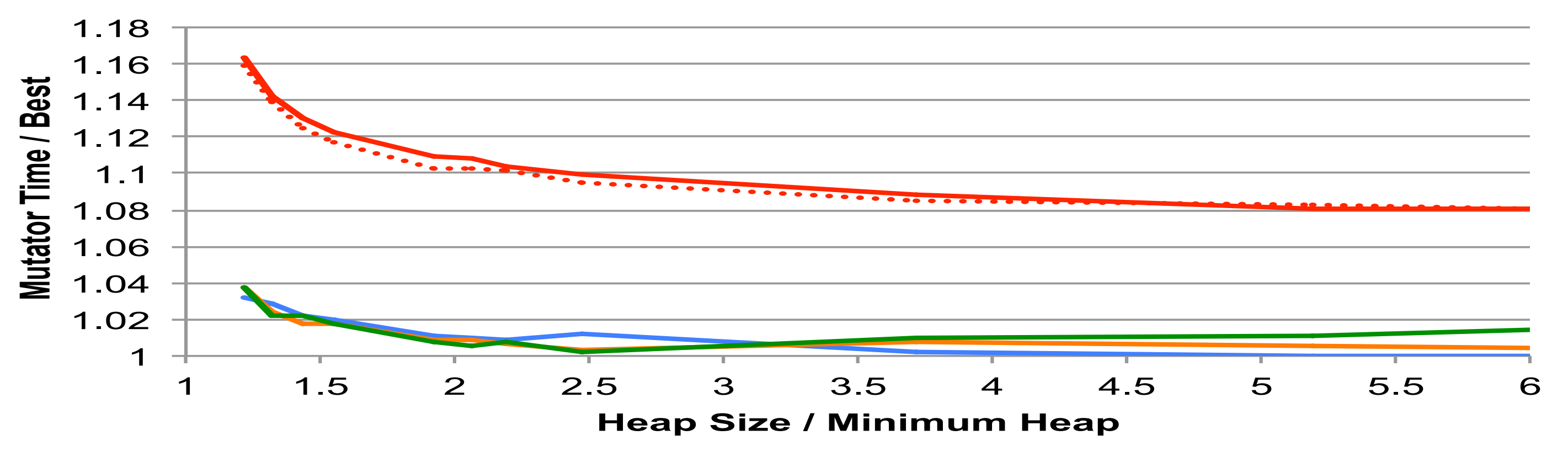
Performance Improvement

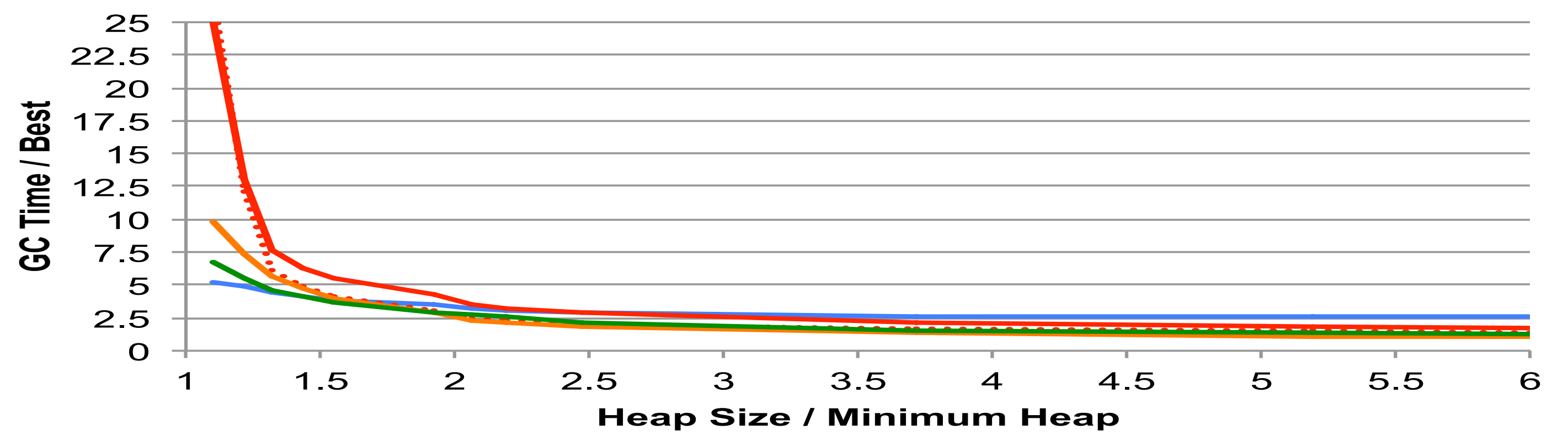
- ✔ RCImmix is 12% faster than RC at moderate (2x) heap size
- ✔ RCImmix outperforms the fastest production (GenImmix) by 3% at 2x
- ✔ RCImmix matches GenImmix at 1.3x and outperforms from 1.4x











Future Opportunities

- Root Coalescing – unnecessary increment and decrement for unchanged roots
- Conservative Stack Scanning – enable to use RCImmix instead of naïve RC

Summary

- ✔ RCImmix, a new GC by combining RC and Immix, outperforms fastest production
- ✔ Transforms RC into a serious alternative to meet high performance objectives for GC languages