Garbage collection (GC) is Ubiquitous

- Born 52 years ago
- Two ideas underpin large literature:
  - Tracing [McCarthy60]
  - Reference Counting [Collins60]
- However
  - ✓ Tracing used in all high performance GCs
  - ❌ Reference counting (RC) only in non-performance critical settings

Reference Counting vs. Tracing

- Advantages
  - ✓ Immediate
  - ✓ Incremental (Reclaim as-you-go)
  - ✓ Object-local
  - ✓ Overhead distributed
  - ✓ Very simple (Trivial implementation for naive RC)
- Disadvantages
  - ❌ Maintain count (Time and space overheads)
  - ❌ Cycles (Can’t be collected)
  - ❌ Complex (High-performance implementation about as complex as tracing)

The Challenge

- ✓ One of the two fundamental GC algorithms
- ✓ Many advantages
  - ❌ Neglected by all performance-conscious VMs
  - ❌ High performance RC is 30% slower than mark-sweep (MS)

Can we get RC back in the ring?

Storing the Count

- Space
  - ✓ Dedicated word (32 bits) per object
  - ✘ Steal bits from each object’s header

- Findings
  - ✗ Max count < 8 in most cases
  - ✗ Very few overflows (The percentage of stuck objects is very low)

Design: Handling stuck objects

- Auxiliary data structure to store count of the overflowed objects
- Ignore them let backup tracing find and free
- Restore them let backup tracing restore count

Maintaining the Count

- Types
  - ✓ Deferred RC ignores changes to stacks and register
  - ✗ Coalescing RC ignores many changes to heap

- Findings
  - ✓ New objects are the source of most incs and decs
  - ✗ Survival ratio is low
  - ✗ New objects a fruitful focus for optimization

Handling of New Objects

- Existing
  - ✗ Implicitly dirty (marked as dirty and inc enqueued for next collection)
  - ✗ Implicitly clean (initial count of one and dec enqueued for next collection)

- New
  - ✓ Implicitly clean (lazily dirty at collection time, non-surviving never processed)
  - ✓ Implicitly dirty (lazily increment at collection time, available to free list if no inc)

Performance Improvement

- ✓ Our improved RC is 24% faster than the standard RC
- ✓ Standard RC was 30% slower than MS, but our improved RC performs the same
- ✓ Improved RC is only 2% slower than URC and 3% slower than Immix
- ✓ It performs same as Sticky MS but 10% slower than Sticky Immix (current project)

RC is back in the ring