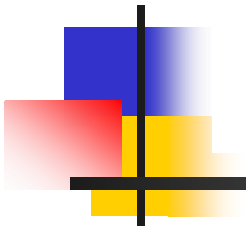
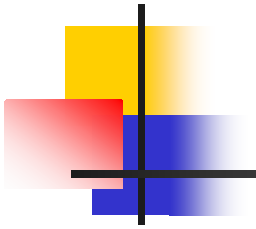


An Efficient On-the-Fly Cycle Collection

Harel Paz, Erez Petrank - Technion, Israel

David F. Bacon, V. T. Rajan - IBM T.J. Watson Research Center

Elliot K. Kolodner - IBM Haifa Research Lab

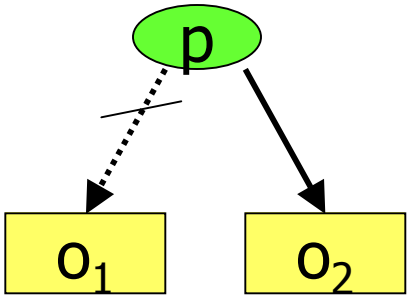


Garbage Collection

- Manual de-allocation may cause notorious bugs (memory leaks, dangling pointers).
- Garbage collection (GC): automatic recycling of dynamically allocated memory.
  - Garbage: objects that are not live, but are not free either.

Reference Counting

- Each object has an rc field.
  - New objects get o.rc:=1.
- When *p* that points to o_1 is modified to point to o_2 we do: o_1.rc--, o_2.rc++.
- if o_1.rc==0:
  - Decrement rc for all sons of o_1.
  - Recursively delete objects whose rc is decremented to 0.
  - Delete o_1.



Three Main Drawbacks of RC

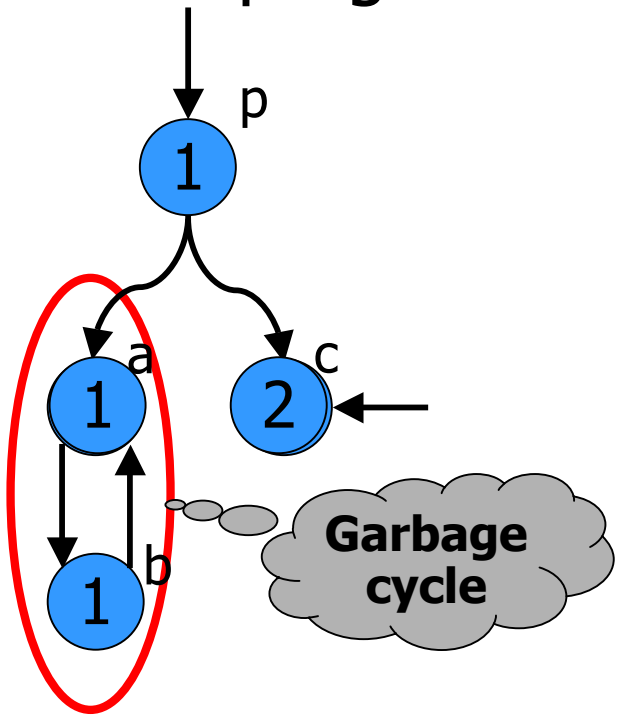
- High overhead
- Costly parallelism
- Inability to reclaim cycles

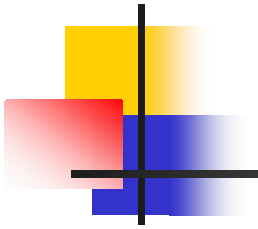
Drastic improvement by Levanoni-Petrank 2001

This work

Cyclic Structures Reclamation Problem

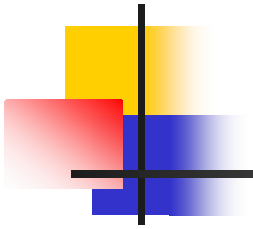
- A garbage cycle denotes a strongly connected component in the objects graph which is unreachable from the program roots.





Collecting Garbage Cycles in Reference Counting Systems

- Reference counting collectors employ one of 2 avenues to collect garbage cycles:
  - A backup tracing collector.
  - A cycle collector.
- This work proposes a new concurrent cycle collection.
- Contributions:
  - More efficient than previous concurrent cycle collector.
  - Solves termination problem.
  - First throughput comparison between cycle collection and a backup tracing collector.



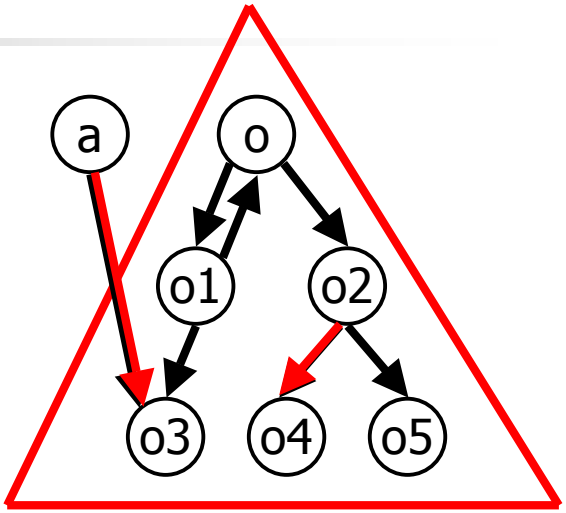
Cycle Collection Basic Idea - 1

- Observation 1: Garbage cycles can only be created when a rc is decremented to a non-zero value.

⇒ Objects whose rc is decremented to a non-zero value become candidates.

Cycle Collection Basic Idea - 2

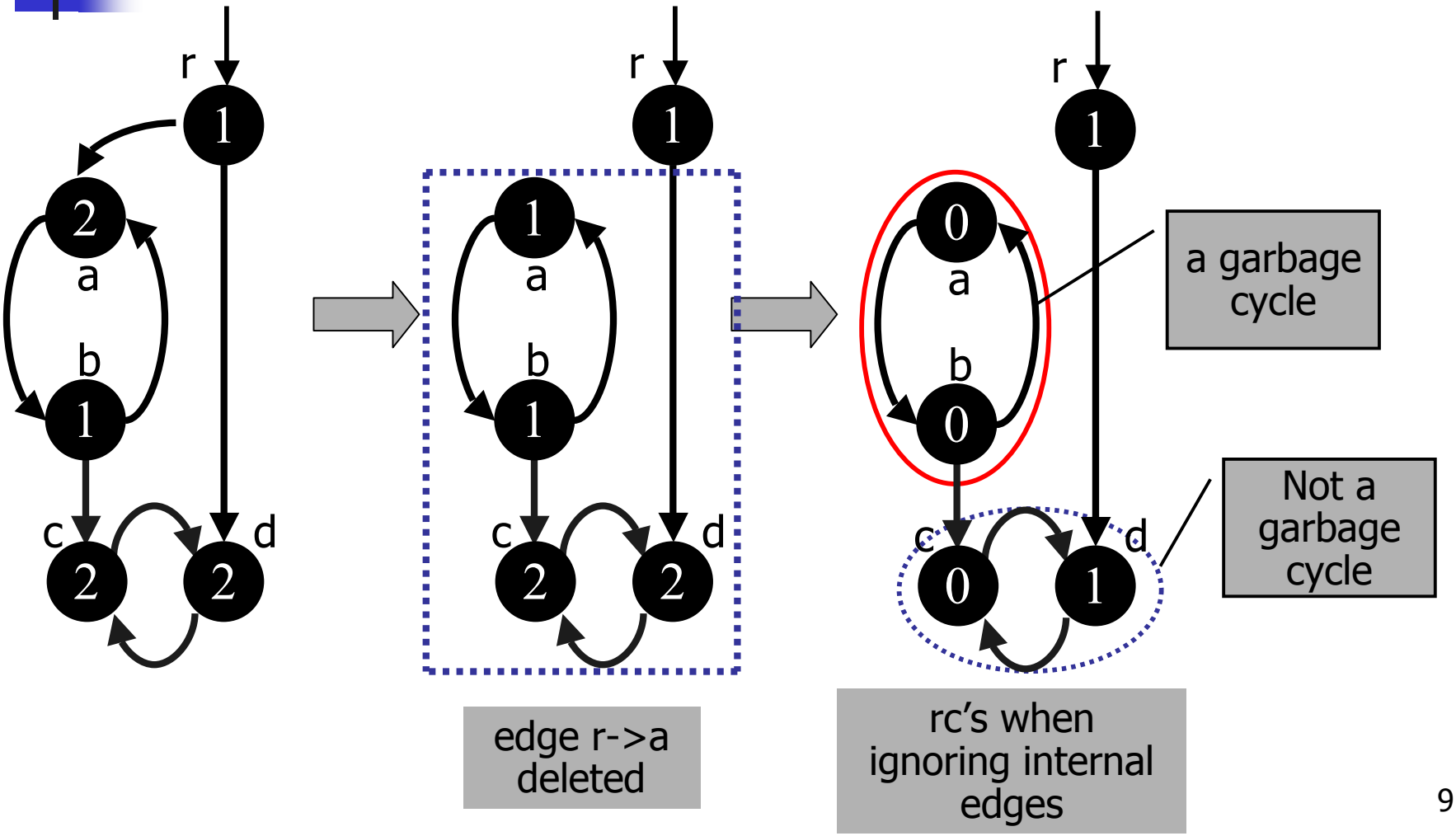
- Terms:
  - Sub-graph of *O*: graph of objects reachable from *O*.
  - External pointer (to a sub-graph): a pointer from a non sub-graph object to a sub-graph object.
  - Internal pointer (of a sub-graph): a pointer between 2 sub-graph objects.



- Observation 2: In a garbage cycle all the reference counts are due to internal pointer of the cycle.

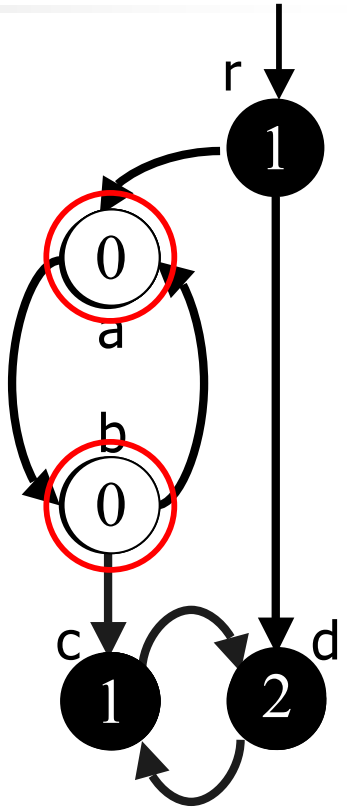
⇒ For each candidate's sub-graph, check if external pointers point to this sub-graph.

Goal: Compute Counts for External Pointers Only



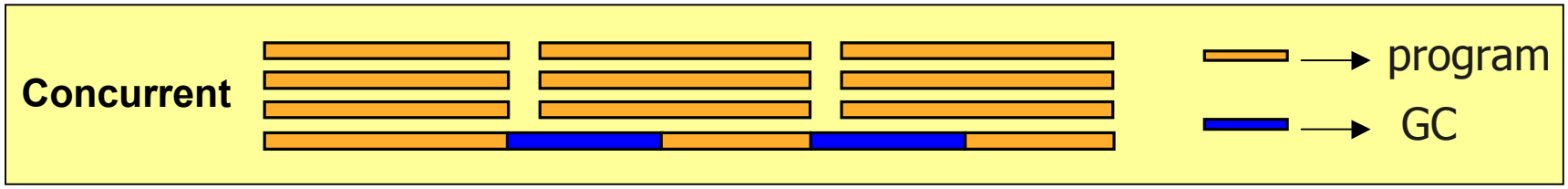
Implementing the Cycle Collector Idea

- Object is colored black/ gray/ white.
- ⇒ Whenever a rc of an object 'a' is decremented to a non-zero value, perform 3 **local** traversals over the graph of objects of 'a'.
  - Mark: Updates rc's to reflect only pointers that are external to the graph of 'a', marking nodes in gray.
  - Scan: Restores rc's of the externally reachable objects, coloring them in black. Rest of the nodes are marked as garbage (white).
  - Collect: collects garbage objects (the white ones).



Concurrent Cycle Collection

- A concurrent collector: a collector that runs concurrently with the program threads.

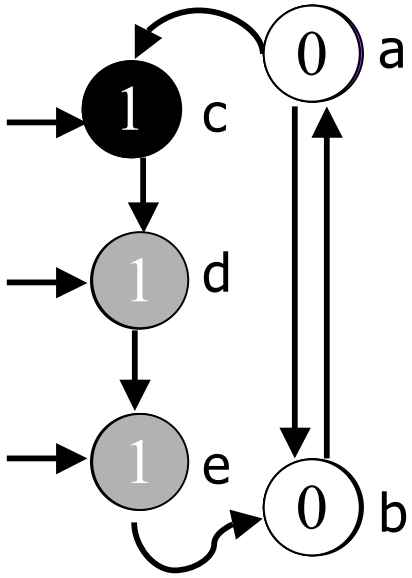


- Concurrent cycle collection is more complex: objects graph may be modified while the collector scans it.
  - **Cannot rely on repeated traversals of a graph to read the same set of nodes and edges.**
  - ⇒ Using the algorithm above may produce incorrect results.

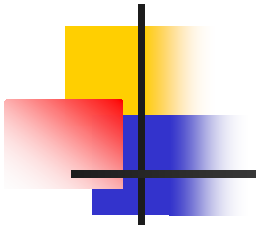
Safety Problem - Example

- A mutator deletes the edge c->d, between the *MarkGray* and *Scan* procedures.

⇒ The *Scan* phase incorrectly infers live objects (a & b) to be garbage.



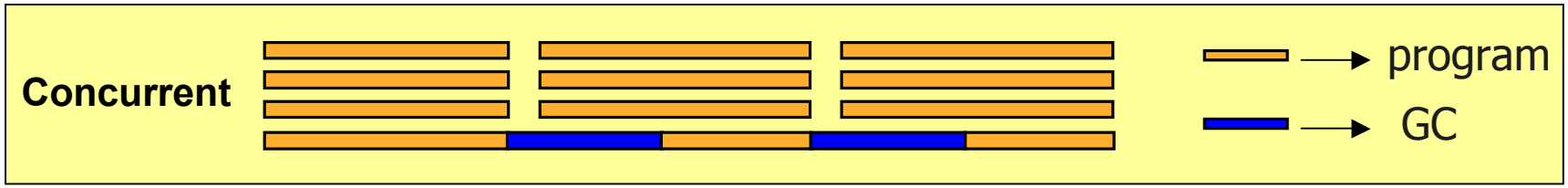
Confronting Drawbacks in Previous Work



- Previous concurrent cycle collector by Bacon & Rajan added overhead to achieve safety in light of inconsistent view of the heap.
  - Overhead reduces efficiency.
  - Completeness could not be achieved.
- Our solution: use a fixed-view of the heap!
  - Multiple heap traces consider the same graph each time.

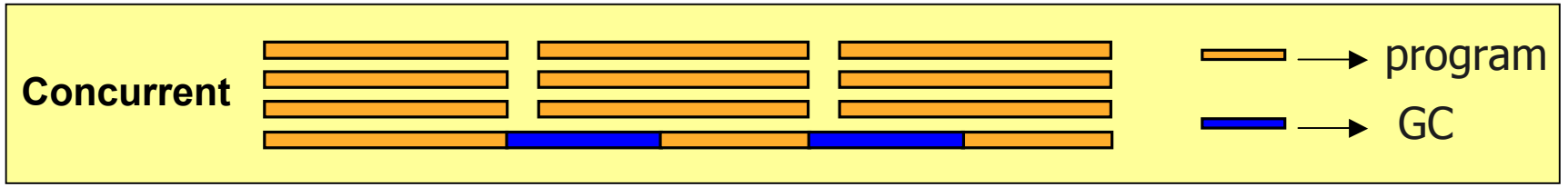
Getting a Snapshot of the Heap

- A snapshot of the heap could be taken by a concurrent collector.
- Levanoni-Petrank's snapshot:
  - Copy-on-first-write mechanism: for each pointer modified for the first time after a collection:
    - Save its snapshot value in a buffer.
    - Mark the pointer "dirty" (no need to be logged again).
- The cycle collector traverses each object according to the pointers' values as existed in the snapshot time.



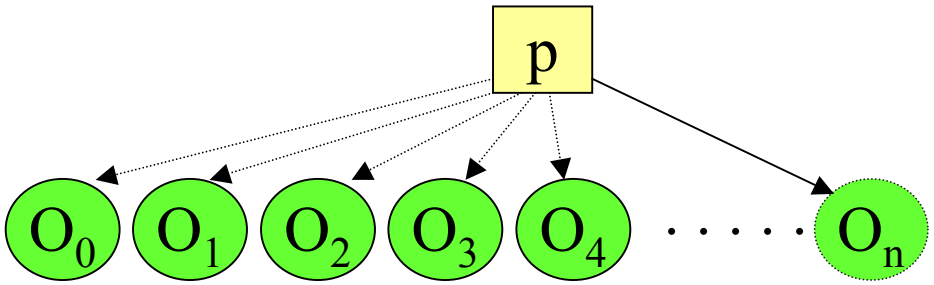
Cycle Collection on Heap's Snapshot

- The standard (non-concurrent) cycle collection correctly identifies garbage cycles on a snapshot.
  - It is not disturbed by mutator activity.
- All garbage cycles are collected.
  - A garbage cycle created, must exist in next snapshot.
- Only garbage cycles are collected.
  - A non reachable cycle in the snapshot is indeed a garbage cycle.



Levanoni-Petrank's RC

- Consider a pointer *p* that takes the following values between GC's: $O_0, O_1, O_2, \ldots, O_n$.



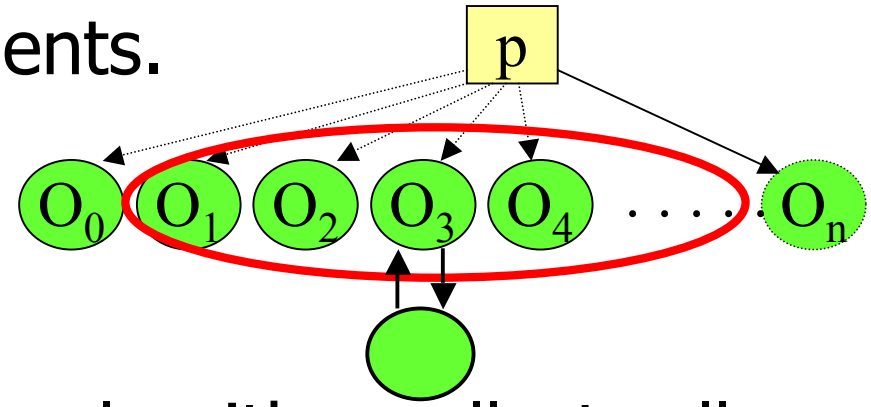
- All RC algorithms perform 2n operations:

  O_0.rc--; O_1.rc++; O_1.rc--; O_2.rc++; O_2.rc--; … ; O_n.rc++;

- But only 2 operations are needed: O_0.rc--, O_n.rc++

Less Cycle Candidates

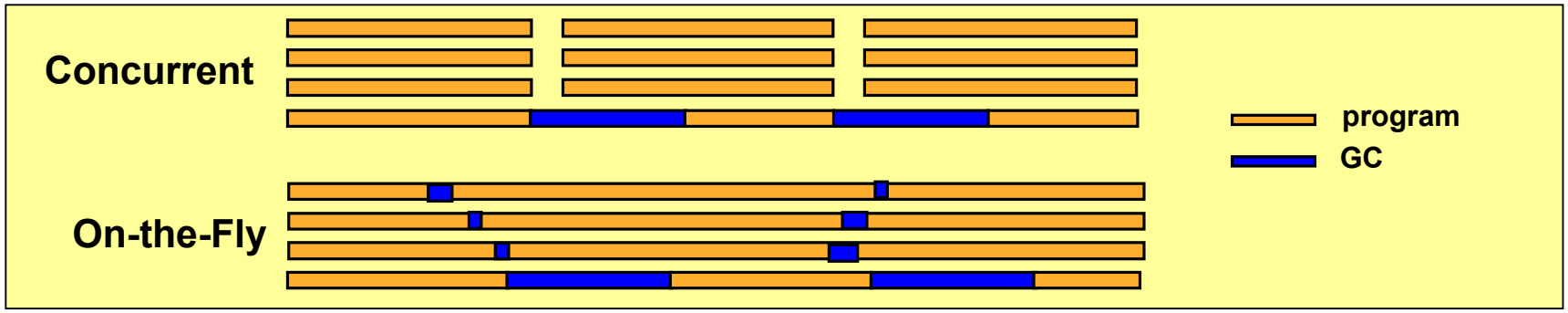
- Previous algorithms: object whose rc is decremented to a non-zero value is considered as a candidate.
- The Levanoni-Petrank's write-barrier does not log most of the decrements.
  - Does it "miss" cycles?



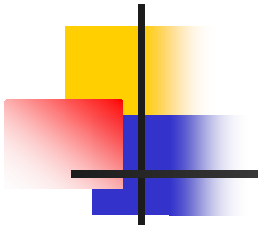
- The new cycle collection algorithm collects all cycles, although performing less work.

More in the Paper

- Reducing pauses further by stopping each thread separately (instead of all together).
  - Care with new races…



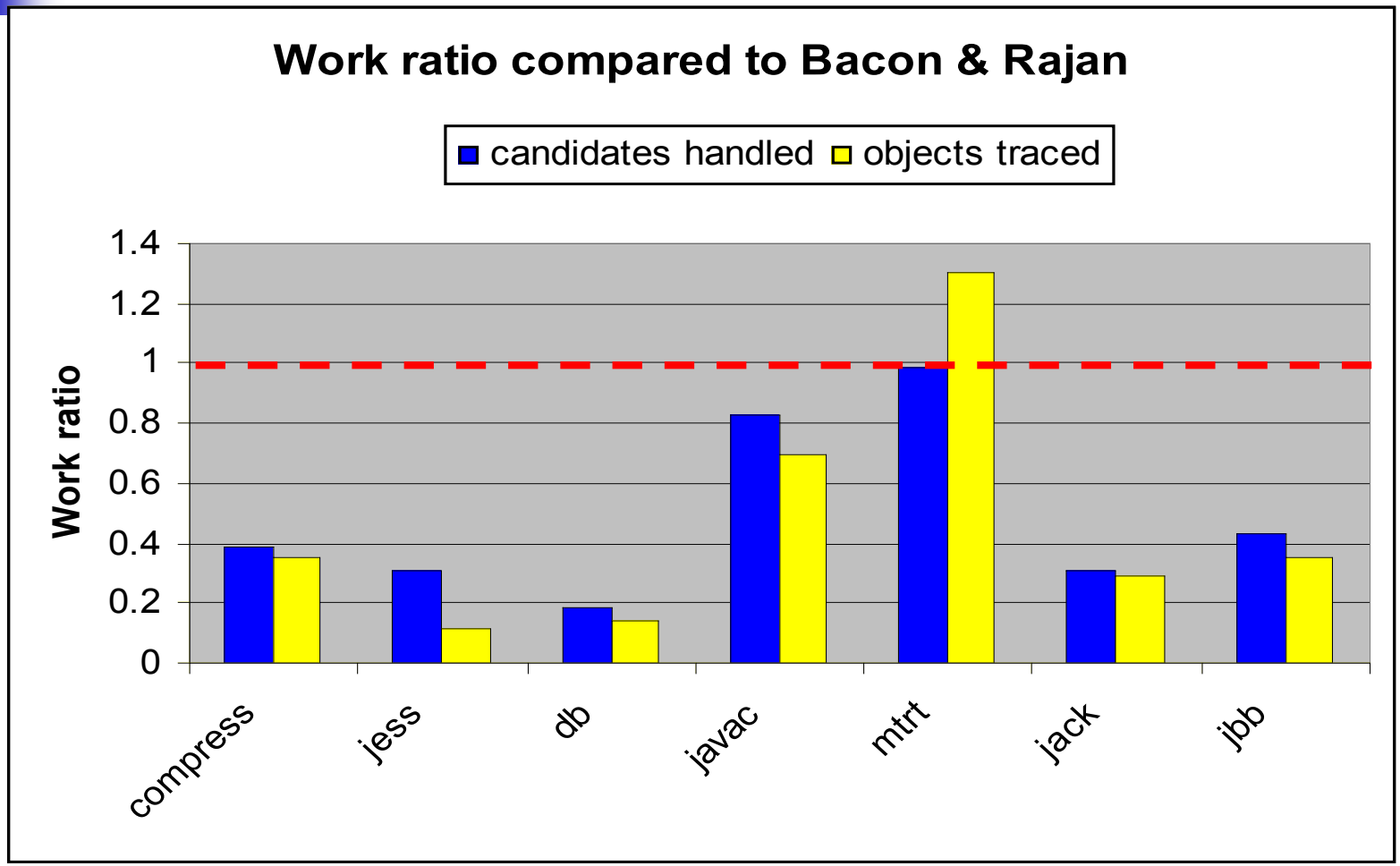
- More techniques to reduce the number of traced objects.



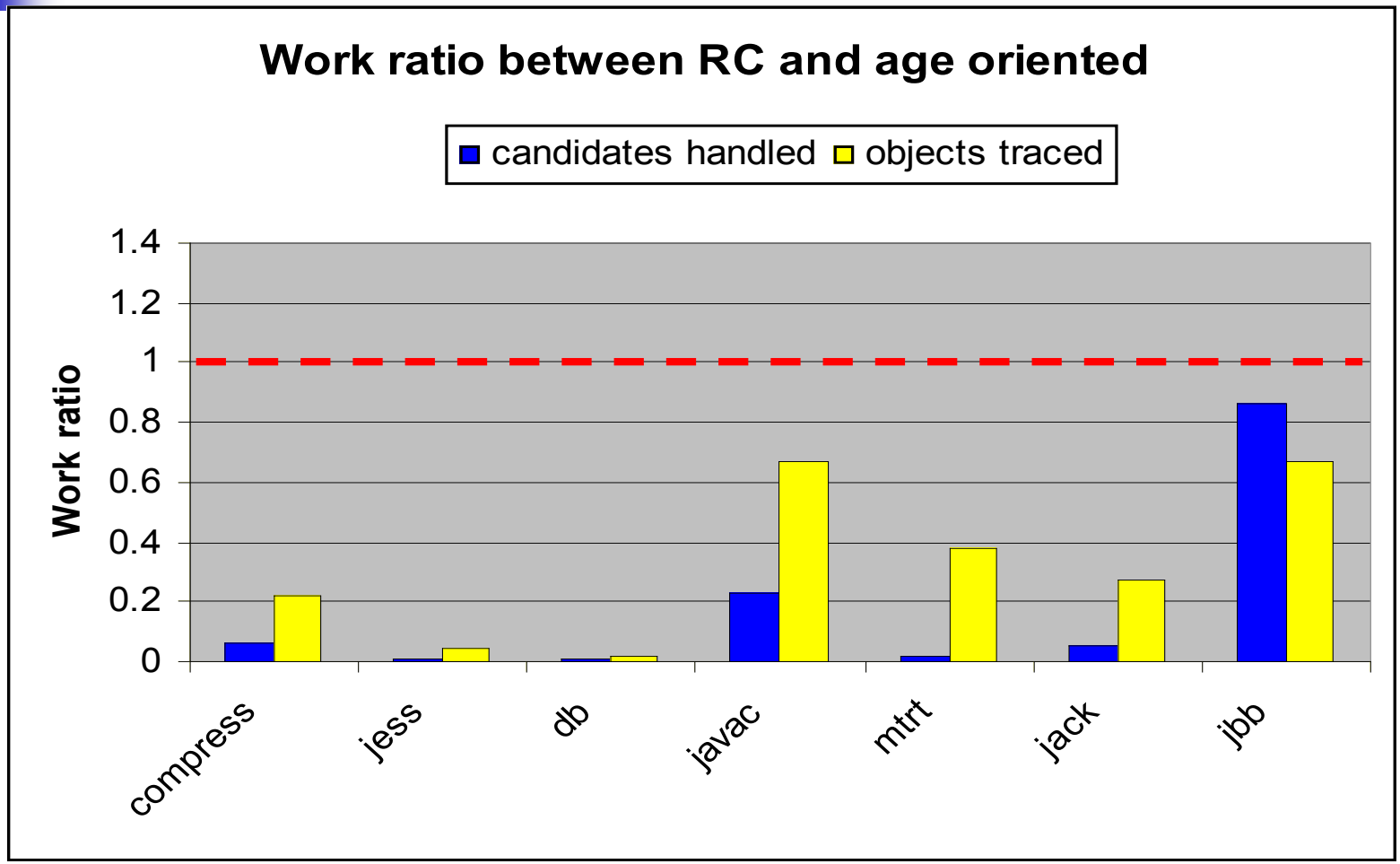
Measurements

- Implemented in Jikes with two collectors:
  - The sliding-views reference-counting collector.
  - The age-oriented collector:
    - Uses mark and sweep for the young generation and reference counting for the old generation.
- Measurements:
  - Throughput comparison between cycle collection and a backup tracing collector.
  - Characteristic comparison to the previous on-the-fly cycle collector.

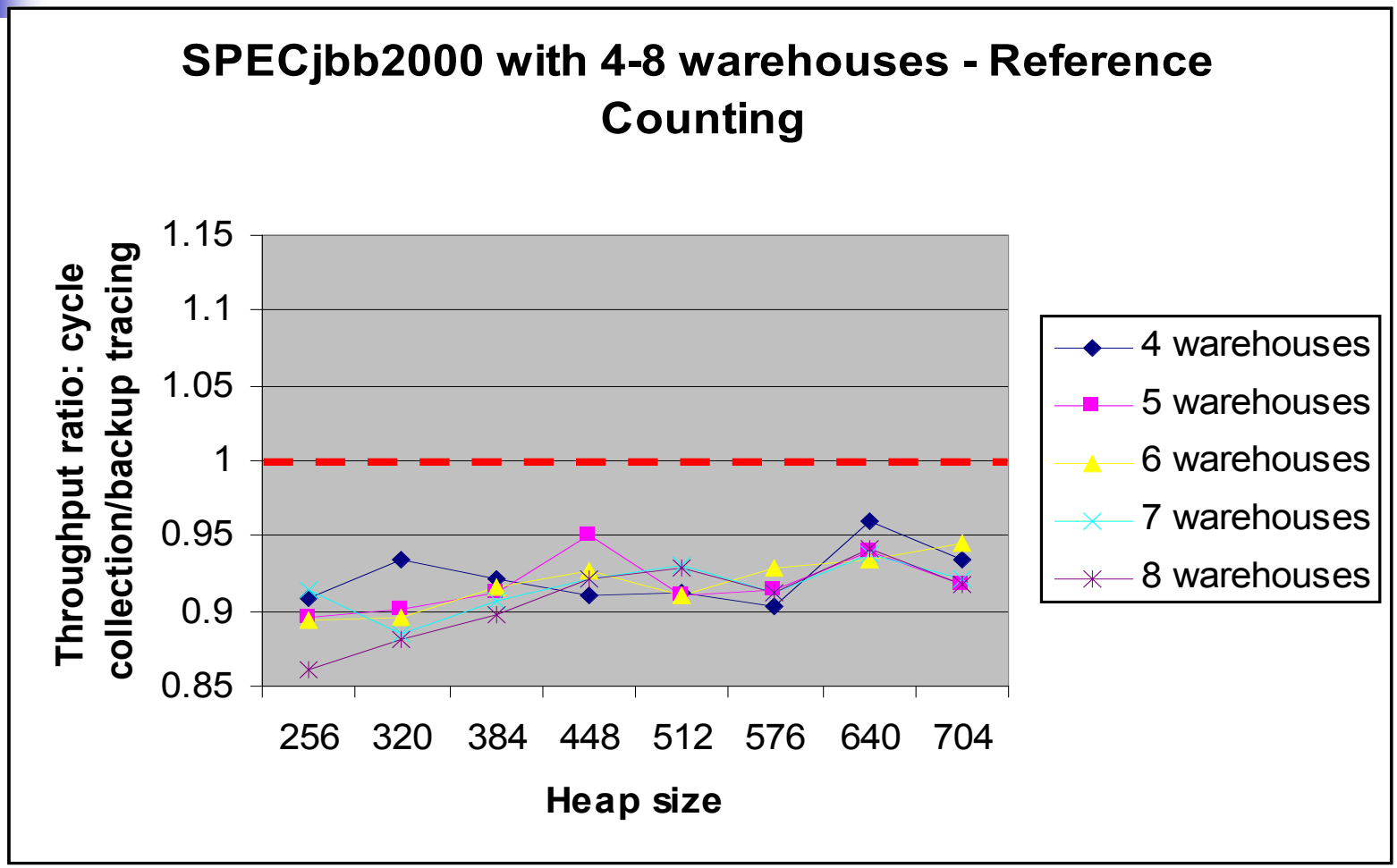
Work Reduction



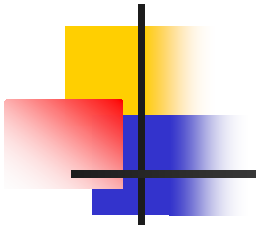
Work Reduction with the Age-Oriented Collector



Throughput Comparison of Cycle-Collection with Backup Tracing

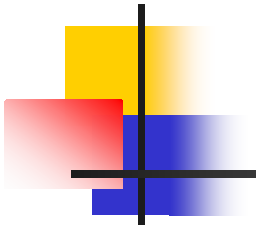


Throughput Comparison of Cycle-Collection with Backup Tracing



Related Work

- Cycle collection:
  - Cyclic reference counting with local mark-scan. Martinez, Wachenchauzer and Lins [1990].
  - Cyclic reference counting with lazy mark-scan. Lins [1992].
  - Concurrent cycle collection in reference counted systems. Bacon and Rajan [2001].
- Other:
  - An on-the-fly reference counting garbage collector for Java. Levanoni and Petrank [2001].
  - Age-Oriented Concurrent Garbage Collection. Paz, Petrank, and Blackburn [2005].



Conclusions

- Cycle collection may be efficiently executed on-the-fly by using Levanoni-Petrank's RC with the efficient standard cycle collector.
- Today's benchmarks:
  - Reference counting for full heap: prefer a backup tracing.
  - Reference counting for old generation: slight preference to cycle collection.
- Eyes for the future: with large heaps cycle collection may outperform backup tracing.