LSU

School of

Electrical Engineering and Computer Science

05.11.14

Toward Scalable, Concurrent and Parallel Garbage Collection

Hari Krishnan

Love Purple
LIVE GOLD

Garbage Collection (GC)

- In heap memory, objects hold pointer/reference to other object in the heap. Objects and nodes are synonymous in this presentation.
- Stack variables and global variables hold reference to objects in heap. They are also called roots.
- A node is said to be reachable if there is any path from root to the node.
- Unreachable nodes are called garbage and memory allocated for those nodes can be reclaimed for future use.
- So garbage collection is the process of collecting garbage in the heap.

Why memory management?

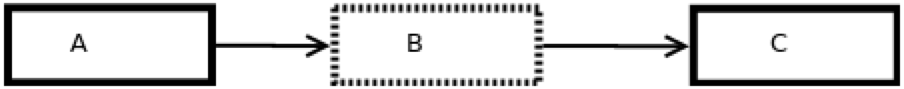
- Allocated but will be never used memory can used by the application so that the allocation for future object could be easy.
- Application could be possible to fit in available memory if the unused memory is reclaimed. This is true for most busy servers.

Garbage Collection

- Garbage collection can be done manually and with help of run-time systems.
- Manual Memory Management (MMM) is used by programmers who use languages with no managed run-time systems such as C/C++.
- Automatic garbage collection or simply garbage collector (GC) is the thread that runs in the run-time systems(managed run-time systems) to automatically detect the garbage and reclaim them.
- Garbage collectors are widely available in various run-time systems including Java Virtual Machine, LISP, and Scheme systems.

Why MMM is not good?

- MMM might appeal to someone who wants to control the world! But in practice, it is not possible.
- Dangling pointer is one of the serious issues of manual GC.

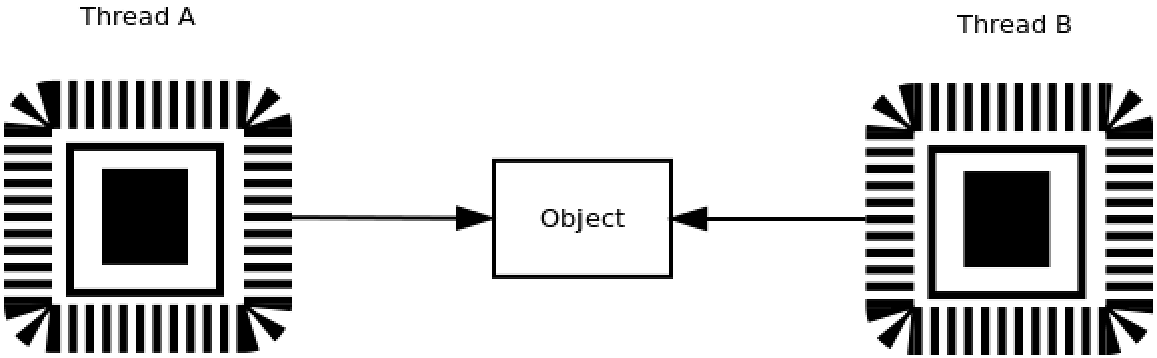


Dangling Pointer and memory leak

- Memory Leak is another big concern that arises from not de-allocating dead objects.

Are these issues can be solved in MMM?

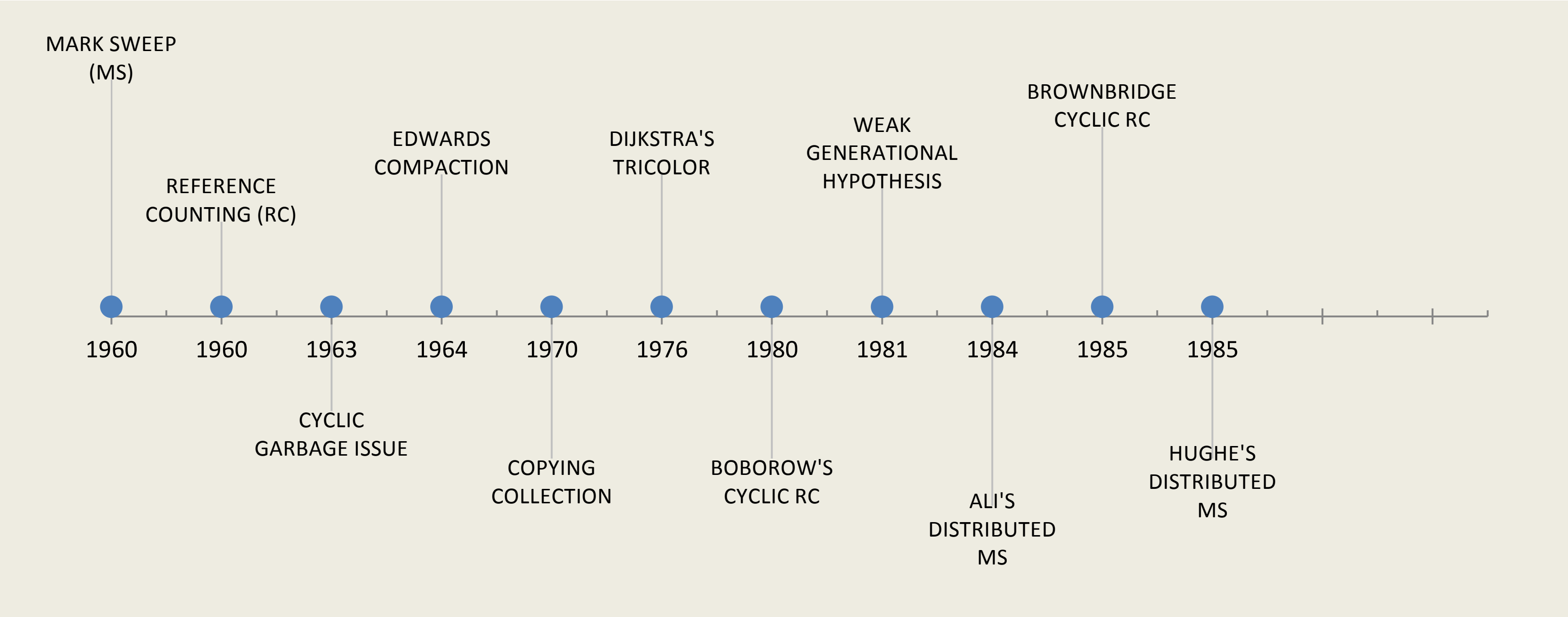
- NO!
- In concurrent programming, there is no easy way to solve these issues unless you borrow some ideas from GC algorithm.
- In concurrent programming, dangling pointers and memory leaks are very common among the GC less run-time systems.



Advantages of GC

- GC has the global knowledge to solve the dangling pointers and memory leaks issues.
- Software Engineering issues are solved.
- Software components when built does not need to worry about cleaning memory.
- Improves reusability, reduces number of lines of code to write.
- AHA! Boehm and Spertus announced that in the next C++ standard GC can be expected! [Boehm and Spertus,2009]
- Commercial Software Development research claims it reduces development cost. [Butters, 2007]

Literature Review



1960 MARK SWEEP (MS)

1960 REFERENCE COUNTING (RC)

1963 CYCLIC GARBAGE ISSUE

1964 EDWARDS COMPACTION

1970 COPYING COLLECTION

1976 DIJKSTRA'S TRICOLOR

1980 BOBOROW'S CYCLIC RC

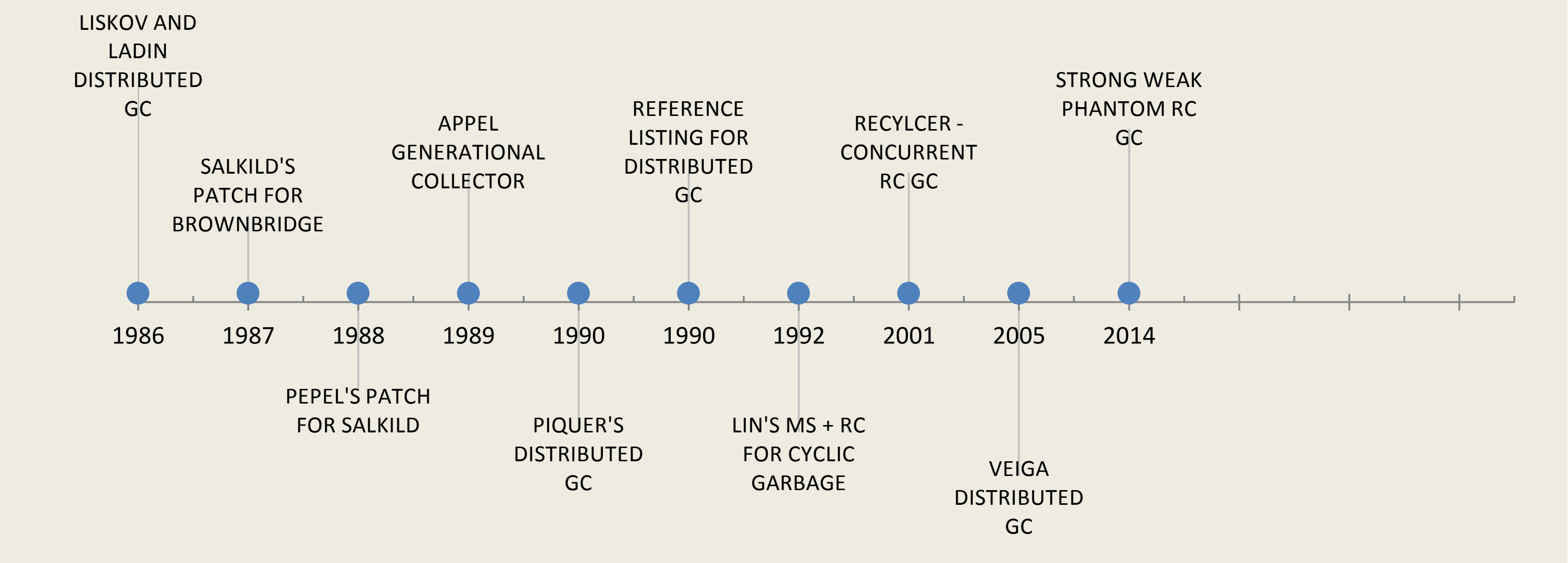
1981 WEAK GENERATIONAL HYPOTHESIS

1984 ALI'S DISTRIBUTED MS

1985 BROWNBRIDGE CYCLIC RC

1985 HUGHE'S DISTRIBUTED MS

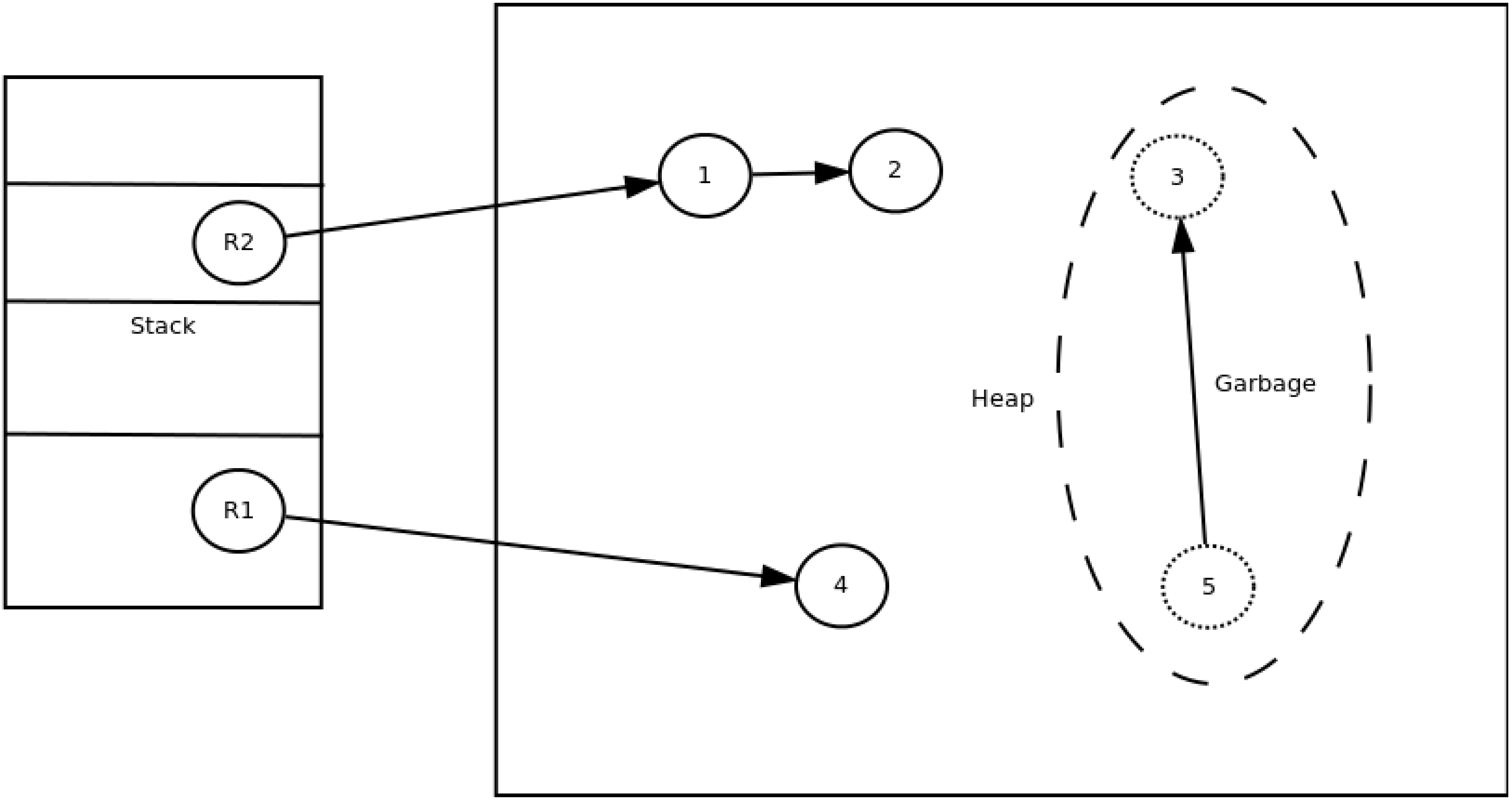
Literature Review


LISKOV AND LADIN DISTRIBUTED GC
1986
SALKILD'S PATCH FOR BROWNBRIDGE
1987
PEPEL'S PATCH FOR SALKILD
1988
APPEL GENERATIONAL COLLECTOR
1989
PIQUER'S DISTRIBUTED GC
1990
REFERENCE LISTING FOR DISTRIBUTED GC
1990
LIN'S MS + RC FOR CYCLIC GARBAGE
1992
RECYLCER - CONCURRENT RC GC
2001
VEIGA DISTRIBUTED GC
2005
STRONG WEAK PHANTOM RC GC
2014

Mark-Sweep

- When memory reached its threshold, the collector starts marking the nodes from the roots. Once the marking is completed, then it sweeps all the unmarked nodes.
- Mark and Sweep only needs one bit and can be easily made concurrent.
- Garbage cannot be collected promptly.
- The whole allocated heap is traversed for each collection as they first traverse all the live nodes and then they traverse all the dead nodes to delete.

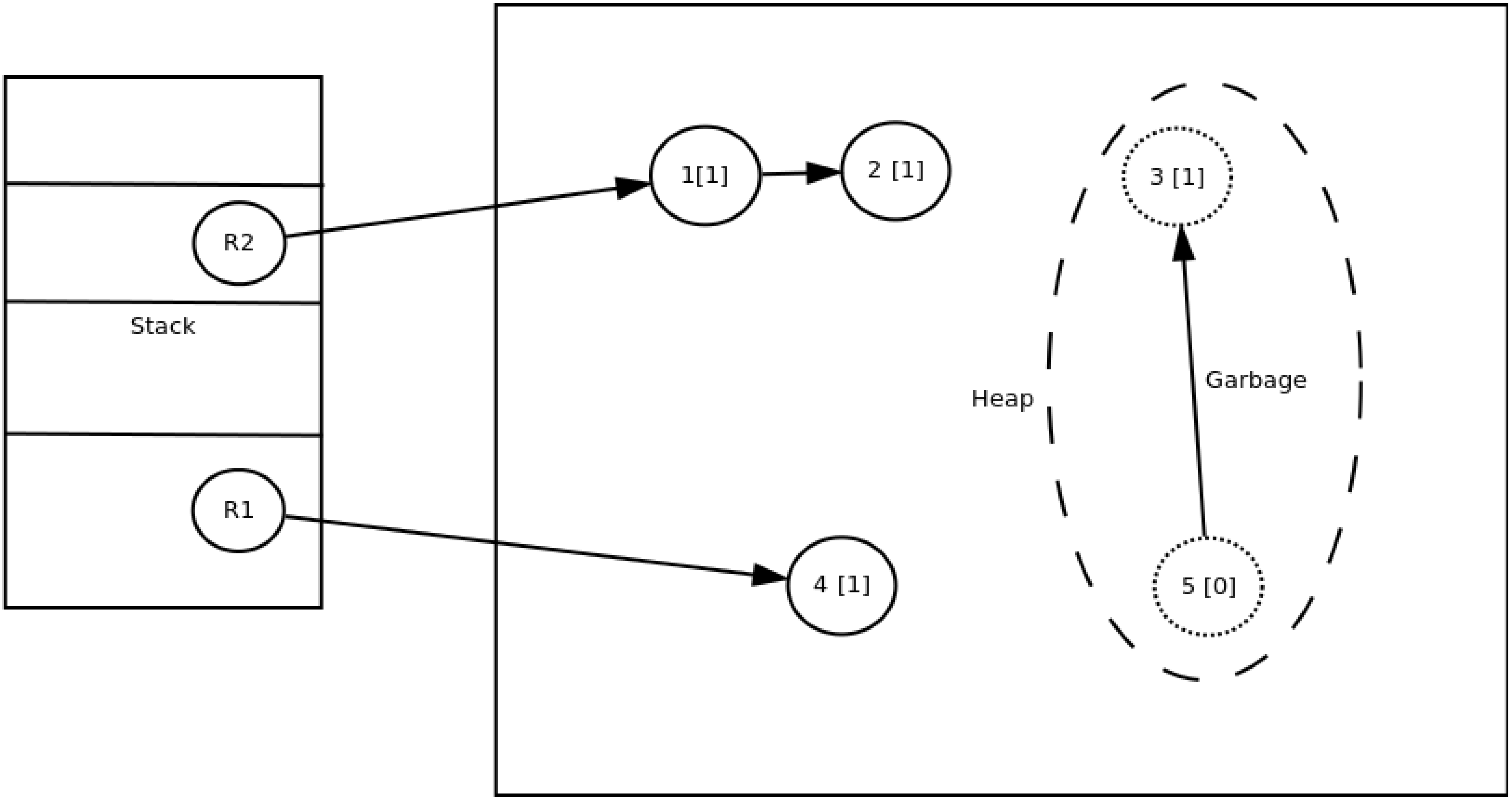
Mark-Sweep



Reference Counting

- Each object has RC that denotes how many incoming reference a node has.
- When an object has zero RC, it is garbage. So collector starts deleting the nodes.
- But the method cannot delete the cyclic garbage as all cyclic garbage nodes have positive RC.
- Apart from inability to detect cyclic garbage, reference count has to updated for object when new reference is made or deleted to it.
- Reference Counting method only traverses the garbage object.

Reference Counting



Compaction & Copying

- When garbage is detected and deleted, they create a hole in the memory. Some request cannot be me met because the memory is fragmented. So compaction was introduced to reduce the fragmentation issues. Mostly compaction is consistently used with Mark Sweep method.
- Copying collectors are predominantly used to avoid traversing all the dead objects. The idea is divide the available memory into two equal half. When half of the memory is full, every live object is shifted to the other half and compacted. Then the other half is cleaned.
- Copying collectors are very helpful when they are only very few objects left alive. Apart from this advantage, only half of the memory is used by the application.

Dijkstra‘s Tricolor Method

- This is a mark-sweep method that is perfectly suited to be concurrent.
- Concurrent collectors are the one which can run along with the application. When the application needs to be stopped for the collection, they are usually denoted as stop the world collectors.
- The method uses three colors.
- Black: A node is painted black If it is visited and immediate children of the nodes are visited. Collector need not visit them again.
- Grey: A node is painted grey if it is visited and immediate children are not visited or if the node experience the change (add/delete link) since the last visit.
- White : A node is white if it is garbage.

Ali‘s Distributed MS

- Each site uses MS to collect its own heap independently.
- At the end of a local garbage collection, the site informs all other sites which remote pointers it retains, and the sites then treat these as roots that must be marked during their own collections.
- These algorithms allows each site to do collection at their own time. But the cyclic garbage cannot be detected by this MS.

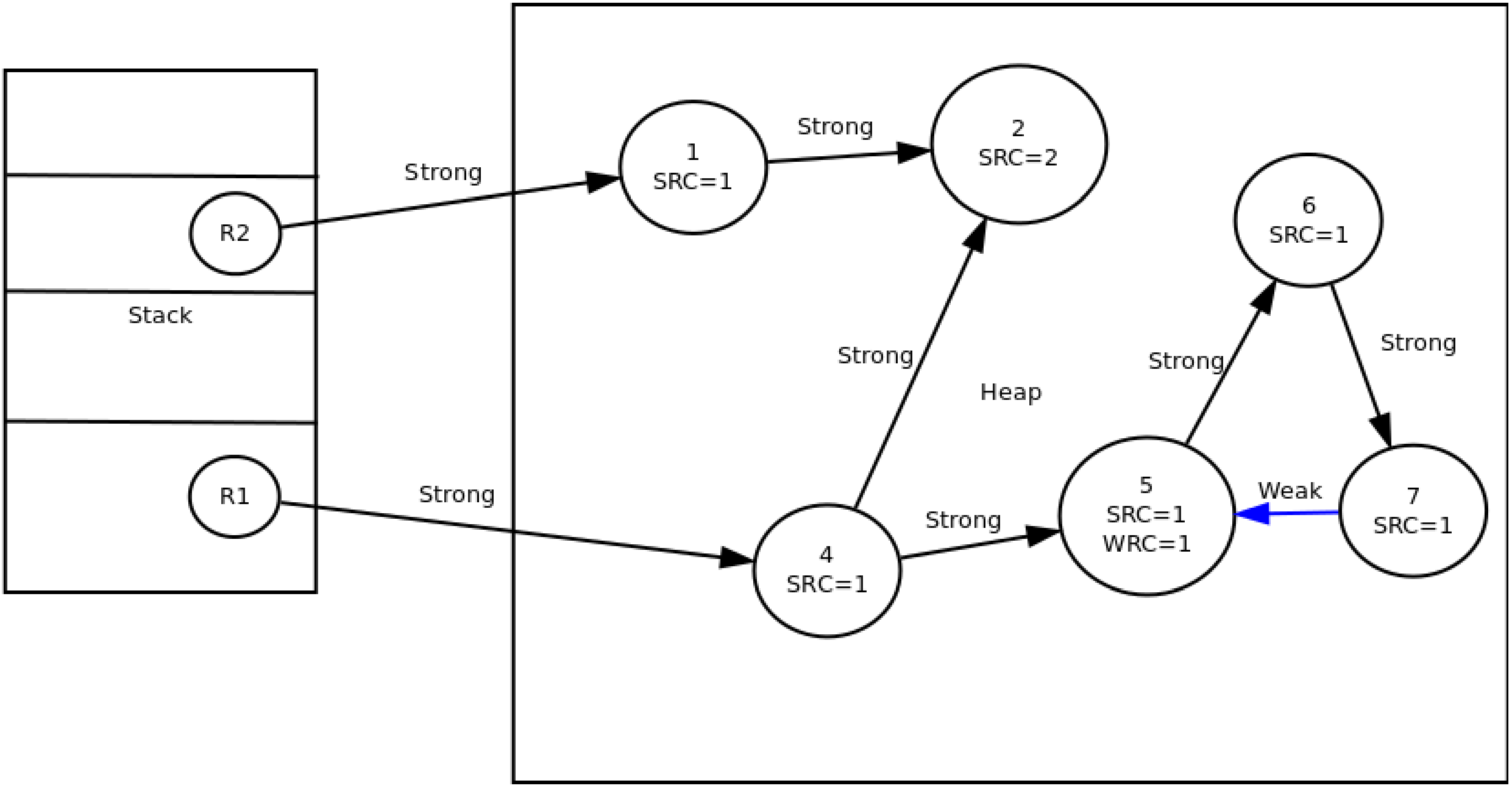
Hughes‘s Distributed MS

- Extends Ali’s method to reclaim cyclic structure.
- This method uses a centralized garbage collection that stops the application when the collection is initiated.

Brownbridge Method

- RC and tracing based technique to collect cyclic garbage.
- It uses two reference count instead of one.
- Strong Reference count (SRC) and Weak Reference Count (WRC).
- References are named as Strong and Weak.
- All live nodes have positive SRC.
- The invariant that this method strongly requires to follow is there are not cycles of strong references. So all cycle have at least one weak reference.
- When a reference is created to a node, if the node already has outgoing references, it is considered a weak reference.

Brownbridge Method



Brownbridge Method

Delete Method

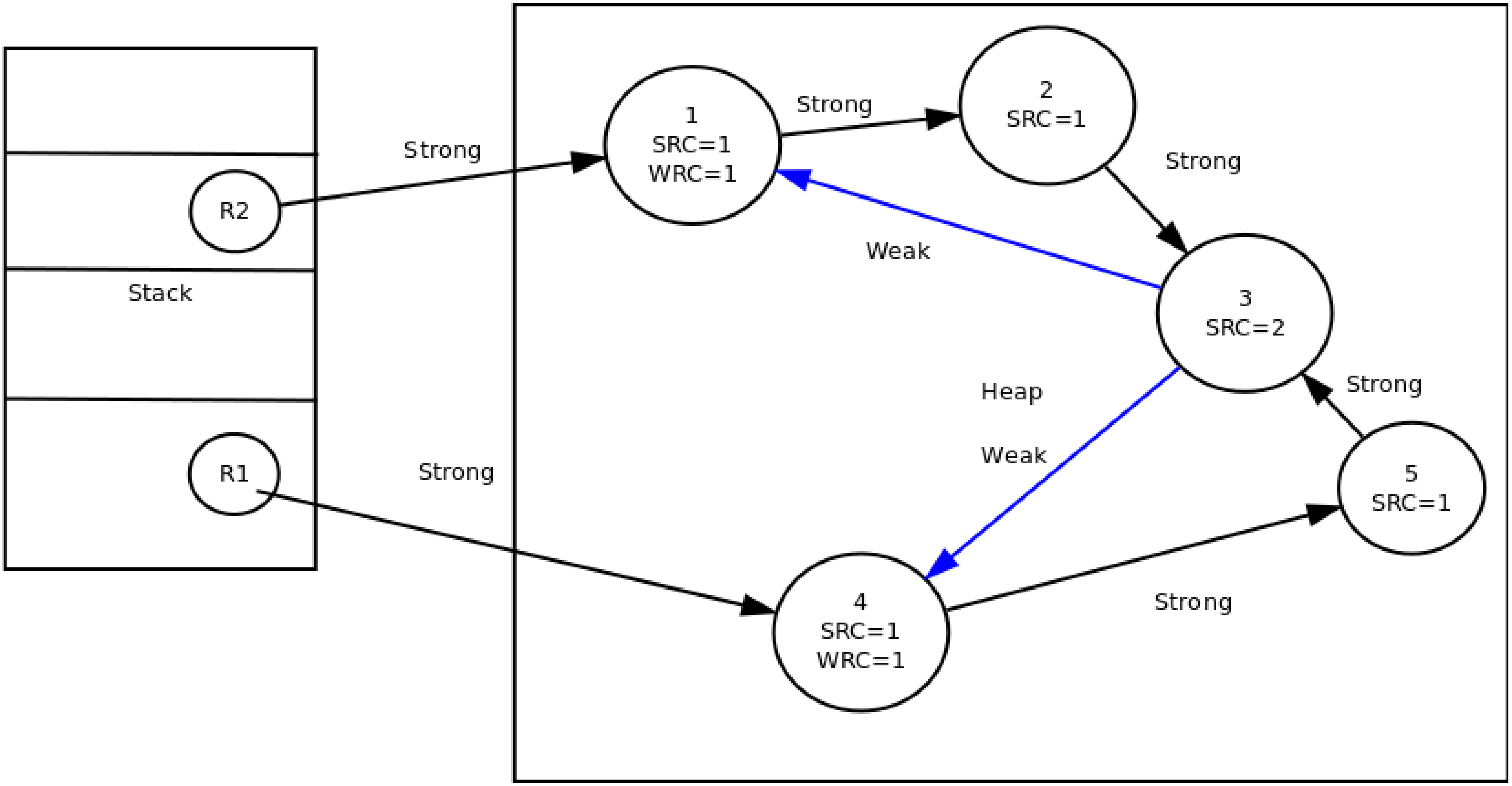
```
delete(SOURCE, DEST)
    if(SRC(DEST)==1 && SOURCE->DEST is strong && WRC(DEST)>0)
        delete the strong references and then convert all WRC to
        SRC(DEST)
        for T in children(D) suicide(DEST,DEST->T)
        if(SRC(DEST)==0) then
            for T in children(DEST) delete(DEST,T)
```

Brownbridge Method

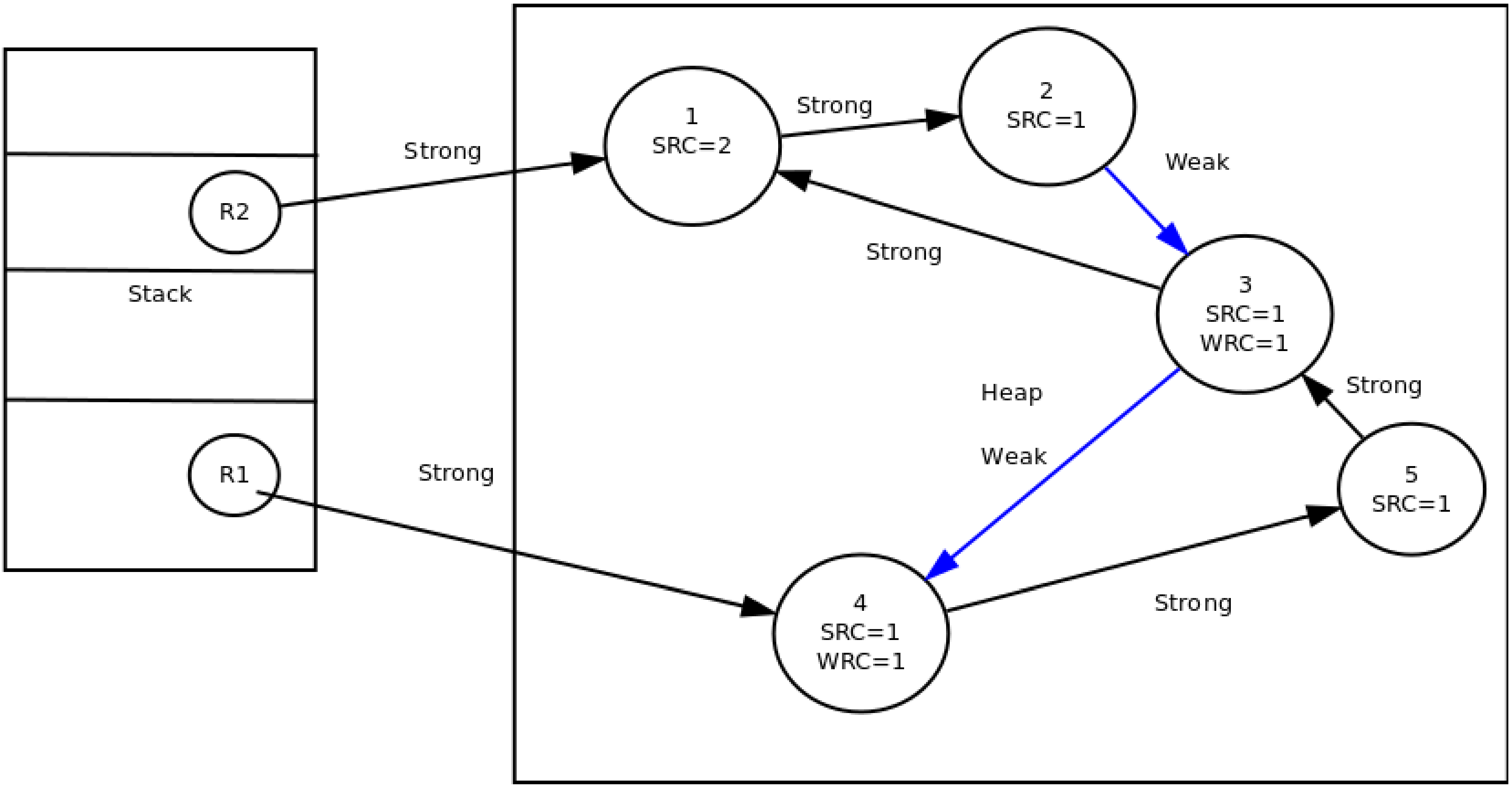
Suicide Method

```
Suicide (Start, SOURCE->DEST)
    if SOURCE->DEST is strong then
        if SOURCE == Start then make SOURCE->DEST to weak link
        else if SRC(SOURCE)>1 then make SOURCE->DEST to weak link
        else
            for T in children(DEST) suicide( Start, DEST->T )
```

Delete Case


Stack
R2
R1
Strong
Strong
Heap
1
SRC=1
WRC=1
Strong
2
SRC=1
Strong
Weak
3
SRC=2
Weak
Strong
4
SRC=1
WRC=1
Strong
5
SRC=1

Premature Delete Case



Salkild and Pepel‘s Work

- Salkild eliminated the premature deletion but introduced non-termination in certain cases.
- Pepel improved the Salkild work with trade-off of exponential cleanup cost.
- So practically all of the attempts to correct the Brownbridge failed.

Recycler

- David Bacon et al used the variant of Lin’s algorithm to design a concurrent garbage collection.
- Lin’s algorithm uses colors to distinguish the node as we mentioned in Dijkstra’s method.
- Apart from Dijkstra’s approach, this method uses reference counting along with tracing to identify cyclic garbage collection.
- Drawbacks of this method includes the collector does not guarantee that it will retrace the same sub-graph since the graph may be modified while the Recycler is detecting cyclic garbage. The collector is not considered as complete collector as it lacks some proofs.

Other Distributed GC work

- Liskov and Ladin method uses inter-site orchestration using table of inter-site incoming and outgoing references. The approach uses threshold value (heuristics) to detect the cycle. A variant of the method uses the object to be moved to one site to solve cyclic garbage. Moving objects may not be feasible in all distributed systems. The system has high overhead in both of the techniques.

Proposed Hypothesis

- Mark-Sweep traces all live objects and then detect garbage.
- Pure Reference counting always traces only garbage object although cannot distinguish cyclic garbage.
- Hybrid collector (MS + RC) traces potential garbage.
- Generational collection traces full heap once in a while and always traces live objects in the young generation.
- For distributed systems, hybrid collector would be the best solution considering the communication and orchestration overhead involved in the other approaches. So any hybrid collection that has good heuristic towards potential garbage perform better than other if it is optimized very well. Apart from that, the hybrid collector that uses local knowledge to detect does not incur orchestration overhead to identify cyclic garbage.

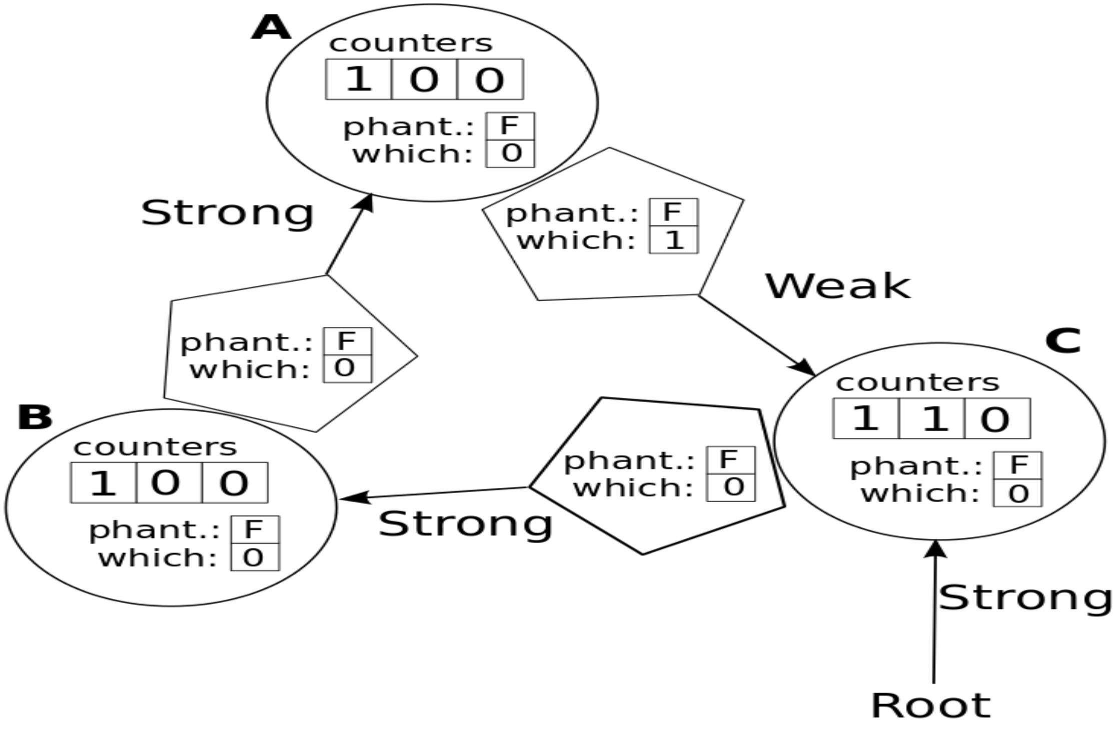
Preliminary Work

- Strong-Weak-Phantom(SWP) Reference Count GC.
- Based on Brownbridge cyclic RC GC.
- Every node has three reference counts: Strong Reference Count (SRC), Weak Reference Count (WRC), and Phantom Reference Count (PRC).
- Every edge/reference is classified as Strong, Weak, and Phantom. The use of Strong, Weak, and Phantom has to do nothing with Java's reference types.
- The idea is to create a path from root to each live node through strong reference and avoid creating cycle of strong reference by using weak reference at the potential cycle creation event.
- Every incoming strong reference increments SRC and respectively.
- Phantom is a transient state that is used only in the cycle detection algorithm.

SWP

- A positive SRC denotes the node is reachable from a root.
- A node with zero SRC and positive WRC denotes the potential cyclic garbage.
- A node with zero SRC and zero WRC denotes it is a garbage.
- A node with positive PRC denotes that node is processed by cycle detection algorithm.
- Apart from reference counts, each node also has which bit, phantomized flag, and array of which bits for outgoing references.
- The two bits for which bit and phantomized bit can be squeezed in the object headers and which for each reference is stored in the last bit of the reference address which are free due to 4 Byte addressing in many systems.

Object Header



Link Creation

- When object is created, the first incoming link will be Strong Reference and SRC is incremented.
- A link will be Weak when the link is created to when the node already has outgoing links.
- This ensures that no cycle of strong reference is created.

Delete Link

When a node lose its last strong reference:

```
if(WRC==0 && PRC==0)
    start deleting the node and delete all outgoing links
else if(WRC>0 && PRC==0)
    convert all the incoming weak references to strong and phantomized
    outgoing links
```

Phantomization

When an incoming link is phantomized, increment the phantom count and decrement appropriate RC.

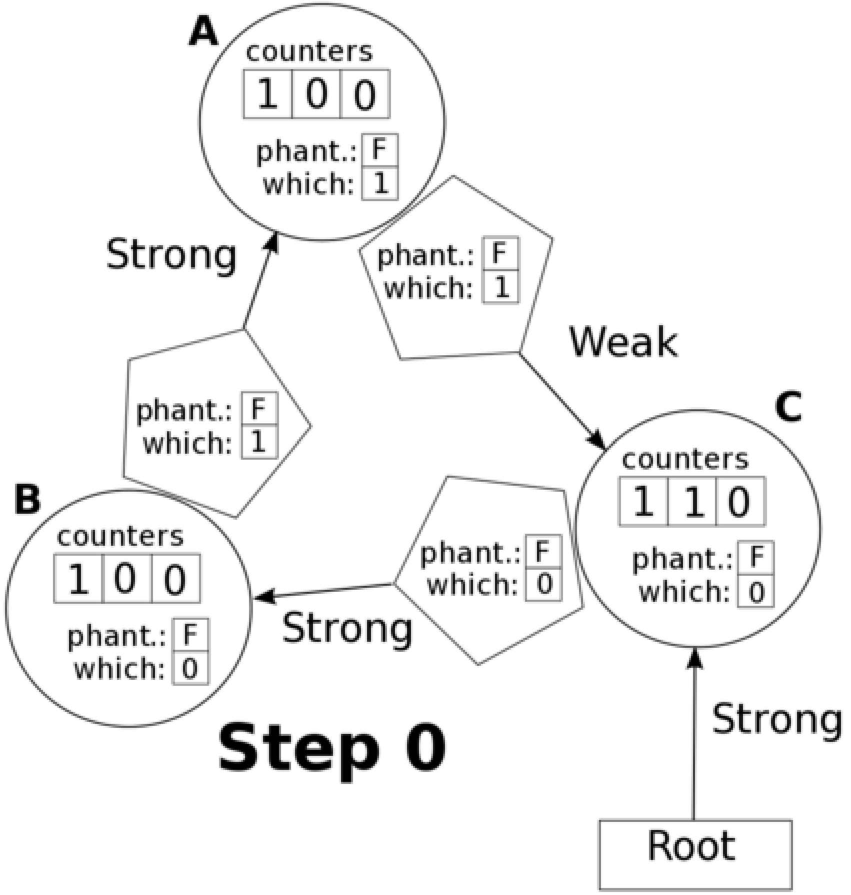
```
if there is a positive SRC
    stop phantomization
else if there is a positive WRC
    convert all WRC to SRC and phantomized outgoing links
```

Three phases

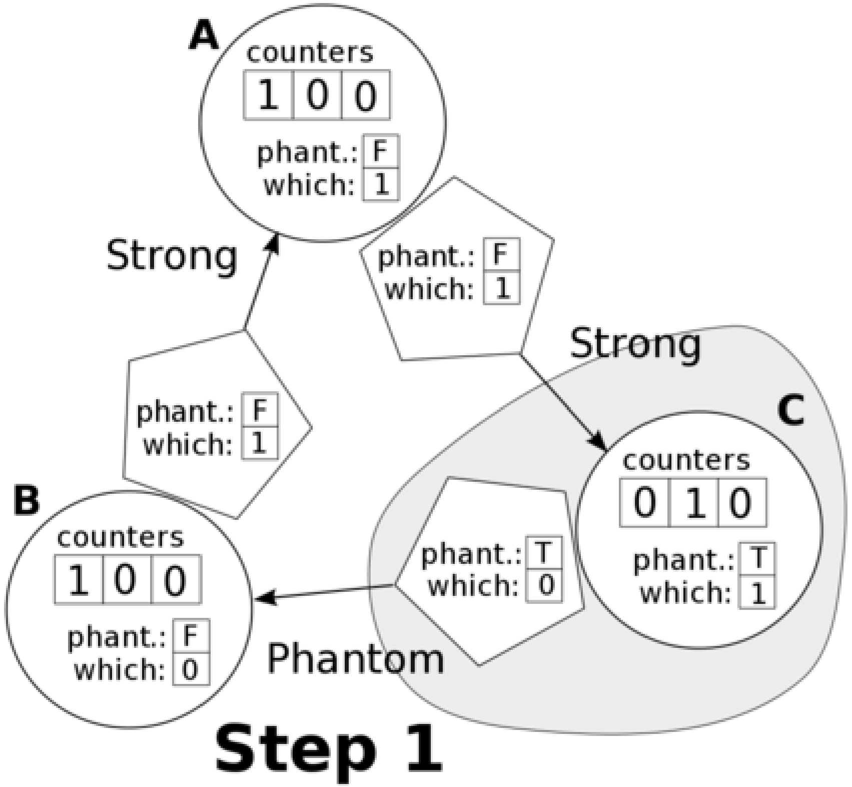
- Phantomize
- Recovery
- Cleanup

These three phases happen for each traversal initiated. Each collector thread maintains list of nodes it processed in each phase.

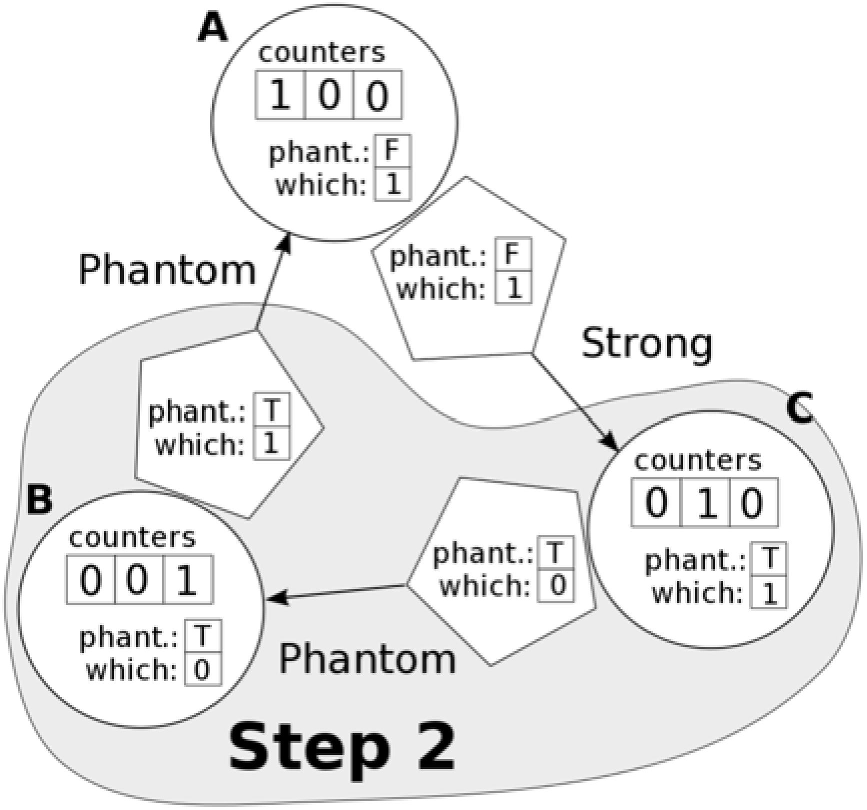
Simple Cycle Demo



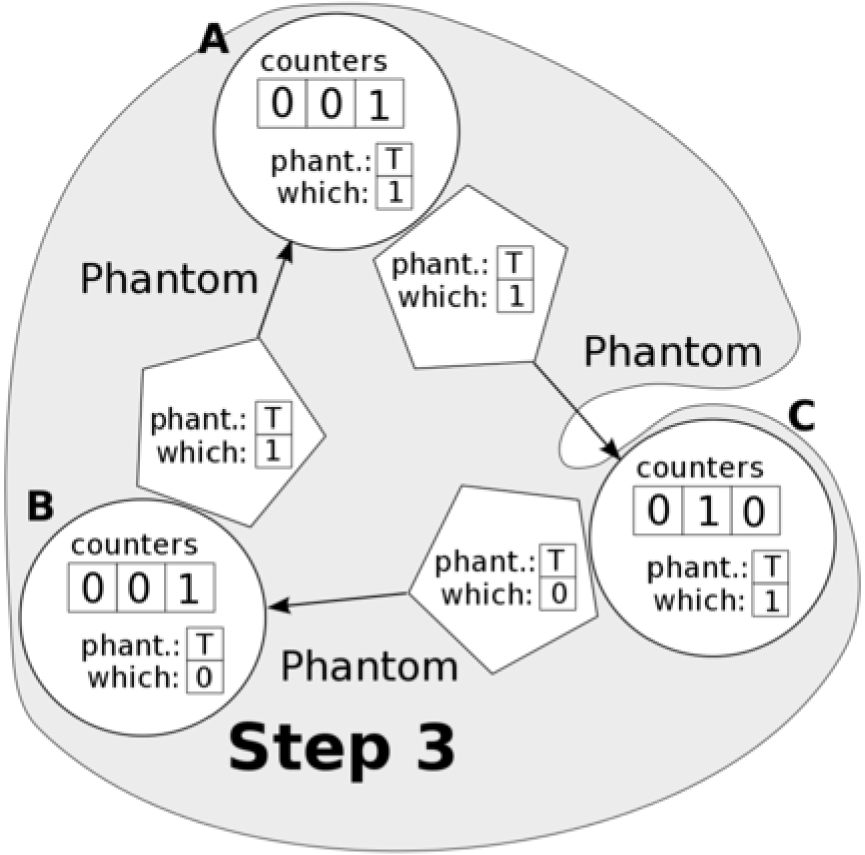
Simple Cycle Demo



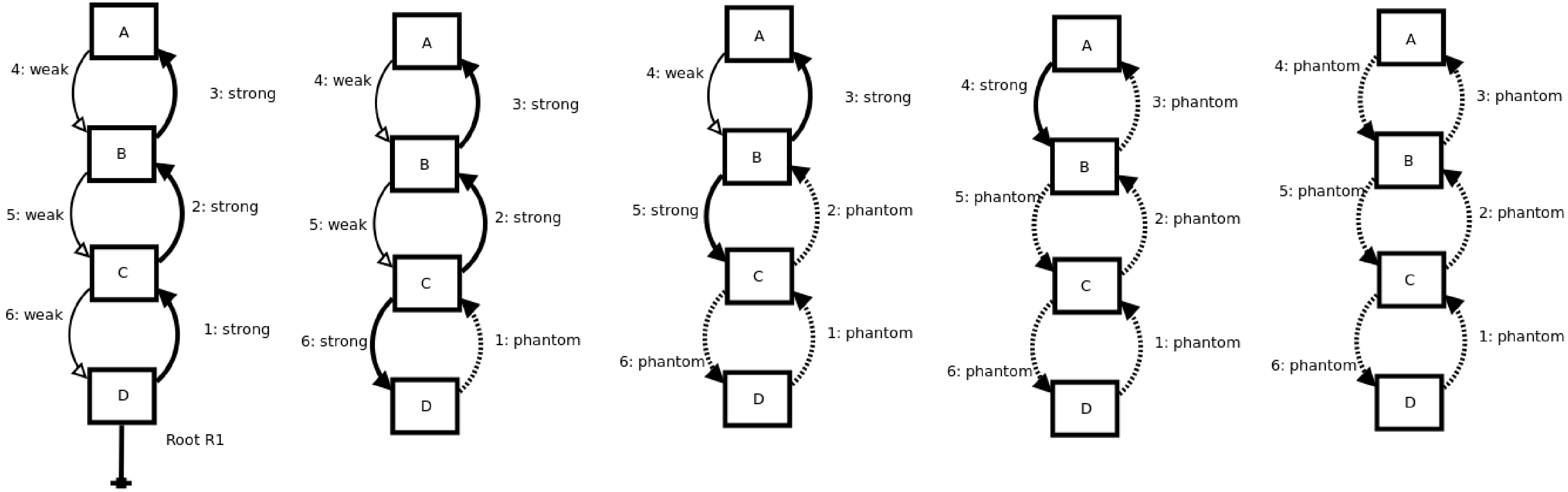
Simple Cycle Demo



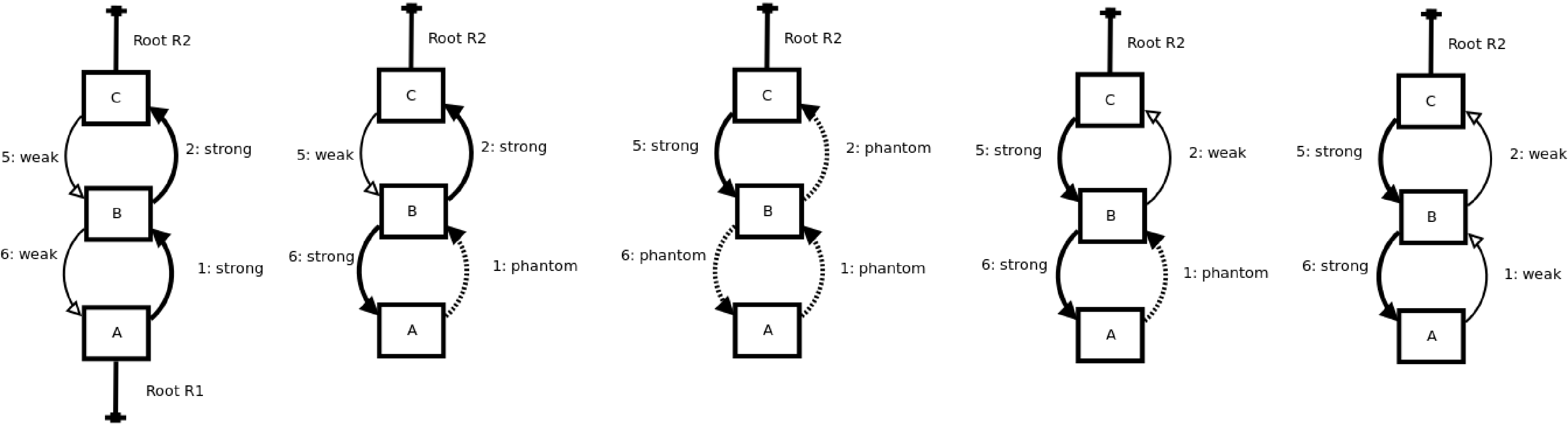
Simple Cycle Demo



Doubly Linkedlist Demo

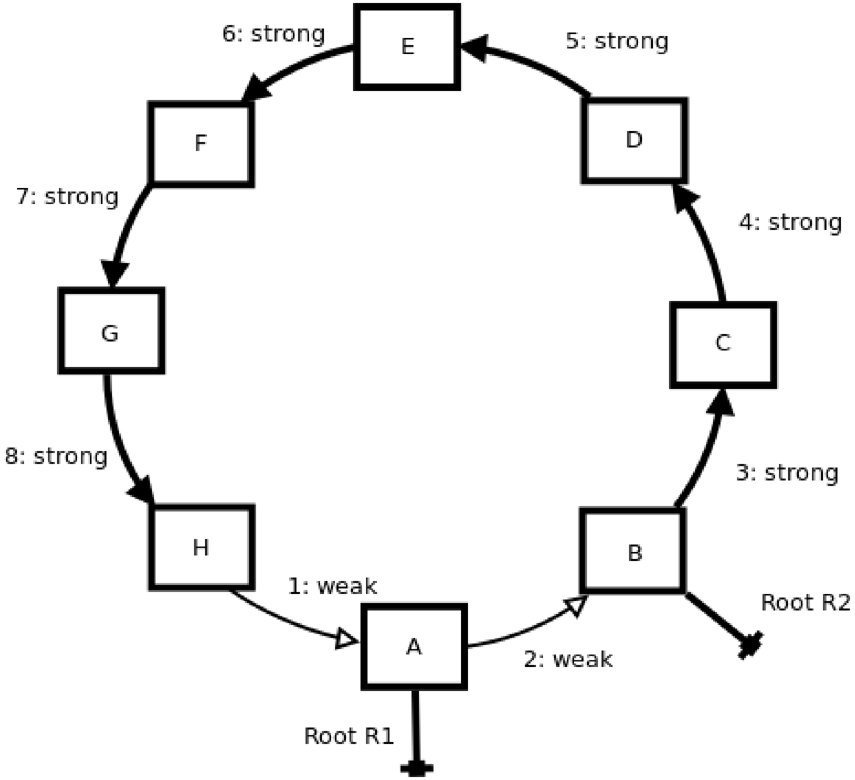


Doubly Linkedlist Demo



Traversal Optimization

-



Concurrent and parallel Collector

- We use locks to access RC. But atomic operations can be used.
- SWP collection does not stop the application.
- When a link is deleted and node needs phantomization traversal or deletion, it queues the request.
- Collector thread process the queue and starts the requested tasks.
- Each collector threads write their id in the node to denote the collector is processing the nodes. When other collector visits a node with id already written, it synchronizes with the collector directly.

Concurrent and parallel Collector

- When two or more collector synchronizes, it merges their list of processed nodes and one of the collector takes the processing of those nodes.
- Other collectors start to process the request in the queue.
- Parallel collector adds one more attribute to the object header.

Application of SWP

- SWP algorithm is directly applicable to shared memory systems.
- SWP algorithm is also applicable to the distributed system with centralized queue.
- Mobile actor based collector does not require centralized collector and makes it directly applicable to the distributed systems.

Issues in SWP

- Although it is applicable to the distributed systems, mobile actor based collector may not be efficient solution in practice as the collector has to move with huge size queue to different site.
- Too much overhead due to updating reference counts for every change in the node.
- So a distributed collector with complete local knowledge and no centralized queue is necessary for distributed systems. The concept of strong and weak reference counts greatly helps to solve the cyclic garbage and with all the pros of the method, this research focuses on using the concept of strong and weak reference for completely decentralized efficient distributed garbage collection with purely local knowledge of the nodes.

Future Work

- So a distributed collector with complete local knowledge and no centralized queue is necessary for distributed systems. The concept of strong and weak reference counts greatly helps to solve the cyclic garbage and with all the pros of the method, this research focuses on using the concept of strong and weak reference for completely decentralized efficient distributed garbage collection with purely local knowledge of the nodes.

Conclusion

- This proposal discusses about previously published method.
- The benefits of the method and its wide usage helped to better shape the future work and find the best possible solution in distributed garbage collection system.

HEADER HERE

Paragraph Copy Here Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum consequat, est at adipiscing tincidunt, diam lacus scelerisque turpis, id aliquam dolor arcu ut turpis. Nullam ac arcu leo, eget scelerisque massa. Nunc laoreet velit risus. Suspendisse faucibus eleifend iaculis. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce tempor condimentum mi, ac venenatis augue vulputate ut.

Header Here

- Bulleted Text Copy Here
- Lorem ipsum dolor sit amet, consectetur adipiscing.
- Vestibulum consequat, est at adipiscing.
- Nullam ac arcu leo, eget scelerisque massa.
- Nunc laoreet velit risus.
- Suspendisse gravida euismod lacinia.

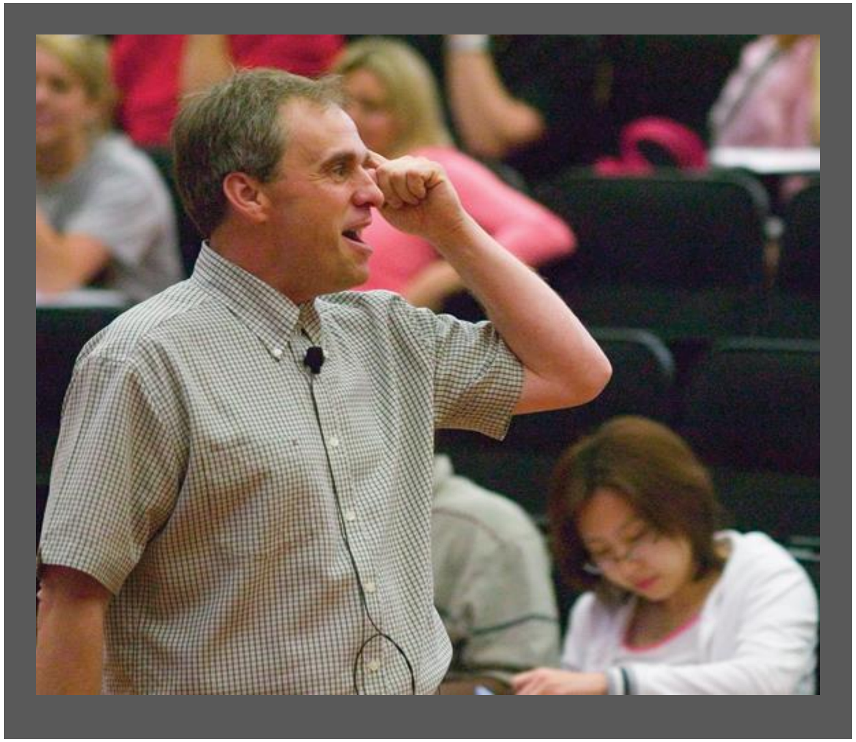


Photo Caption or description here

SubHeader Here ordescriptions etc.

- Bulleted Text Copy Here
- Lorem ipsum dolor sit amet, consectetur adipiscing.
- Vestibulum consequat, est at adipiscing.
- Nullam ac arcu leo, eget scelerisque massa.
- Nunc laoret velit risus.