LSU
School of
Electrical Engineering and Computer Science

06.10.16

Garbage Collection for General Graphs

Hari Krishnan

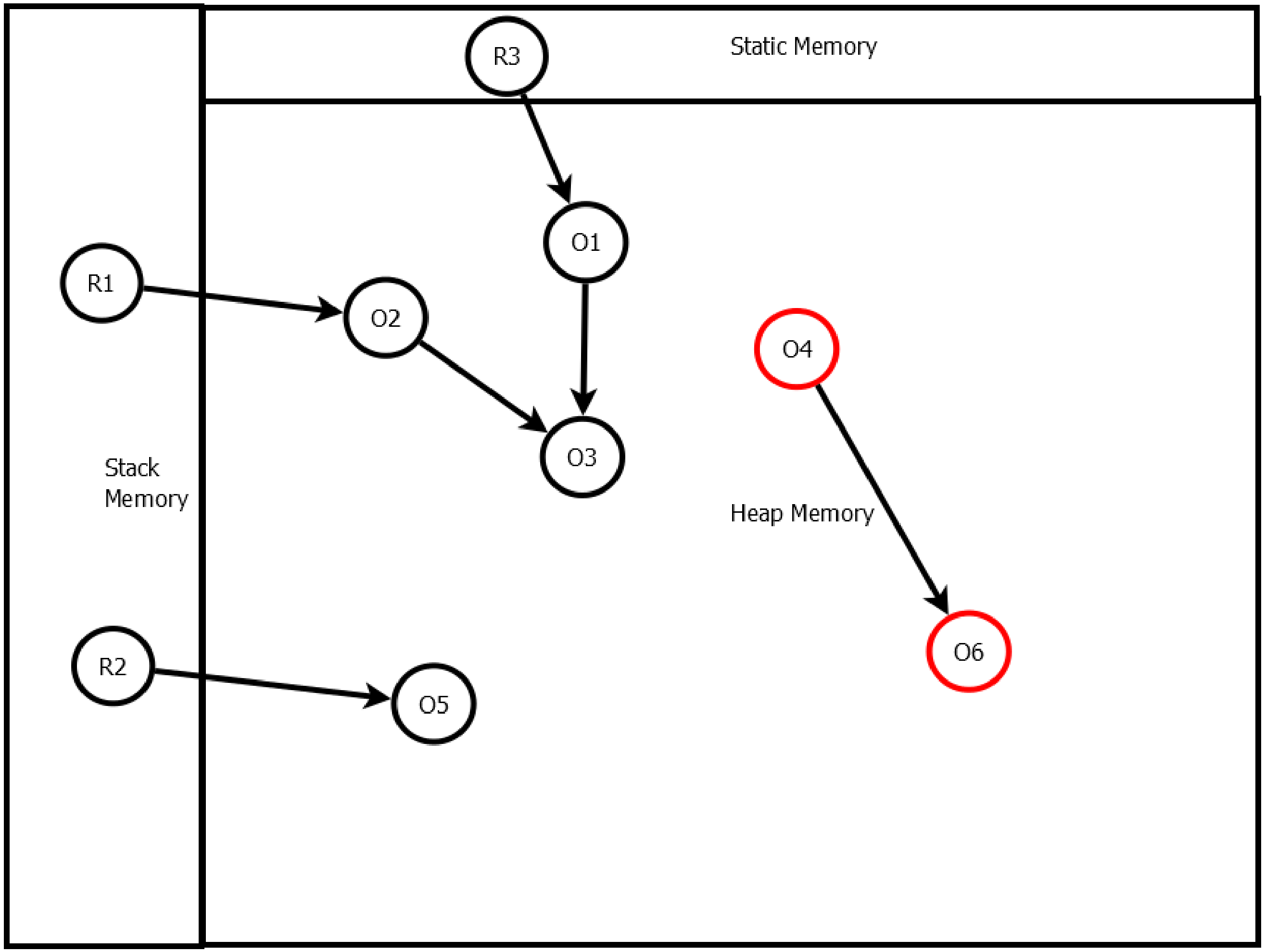
LOVE PURPLE
LIVE GOLD

LSU School of Electrical Engineering and Computer Science

Garbage Collection (GC)

- In heap memory, objects hold pointers/references/links to other objects in the heap.
- Stack variables and static variables hold references to objects in the heap. They are called roots (R).
- An object is said to be reachable / live if there is any path from a root to the object.
- Unreachable objects are called garbage and memory allocated for those objects can be reclaimed for future use.
- Garbage collection is the process of collecting unreachable objects in the heap.

Garbage Collection (GC)

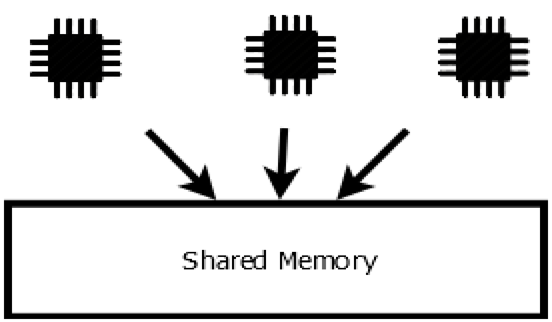


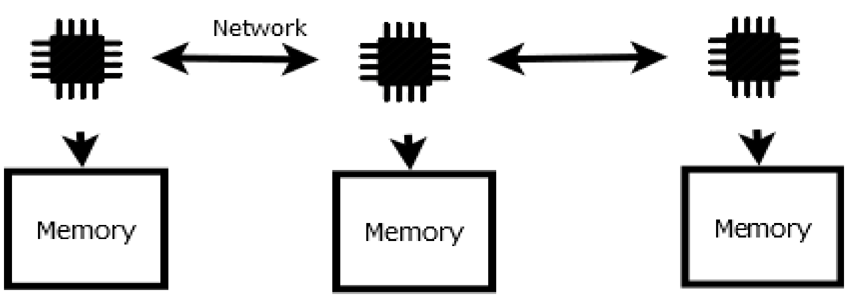
How to collect garbage?

- Manual Memory Management (MMM) and Automatic Memory Management (GC).
- MMM is used by programmers who use languages with no managed run-time systems such as C/C++.
- GC is the thread that runs in the runtime systems(managed runtime systems) to automatically detect the garbage and reclaim them.
- GCs are widely available in various runtime systems including Java Virtual Machine, LISP, and Scheme systems.

Dangling Pointers

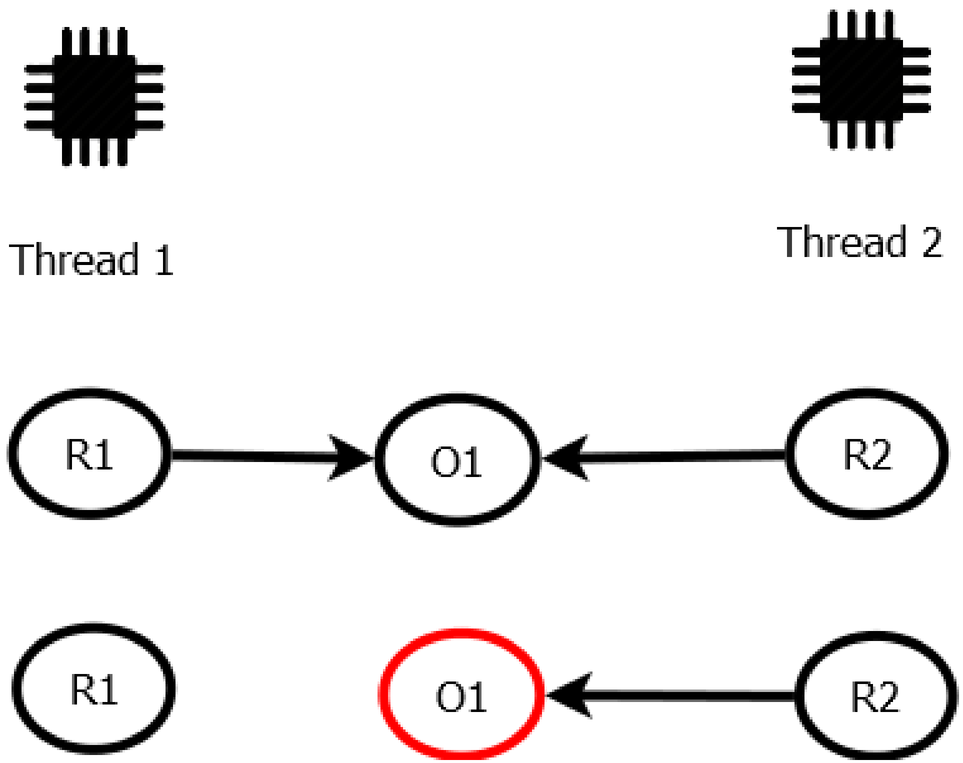
- Dangling pointer is a pointer that points to a deleted object.
- Some other object is allocated in the same address in the interim time.
- In MMM, simple reference counting based smart pointers can sometimes solve this problem in a concurrent programming environment.
- We focus on concurrent programs and their environments in this presentation.





School of
Electrical Engineering and
Computer Science

Dangling Pointers



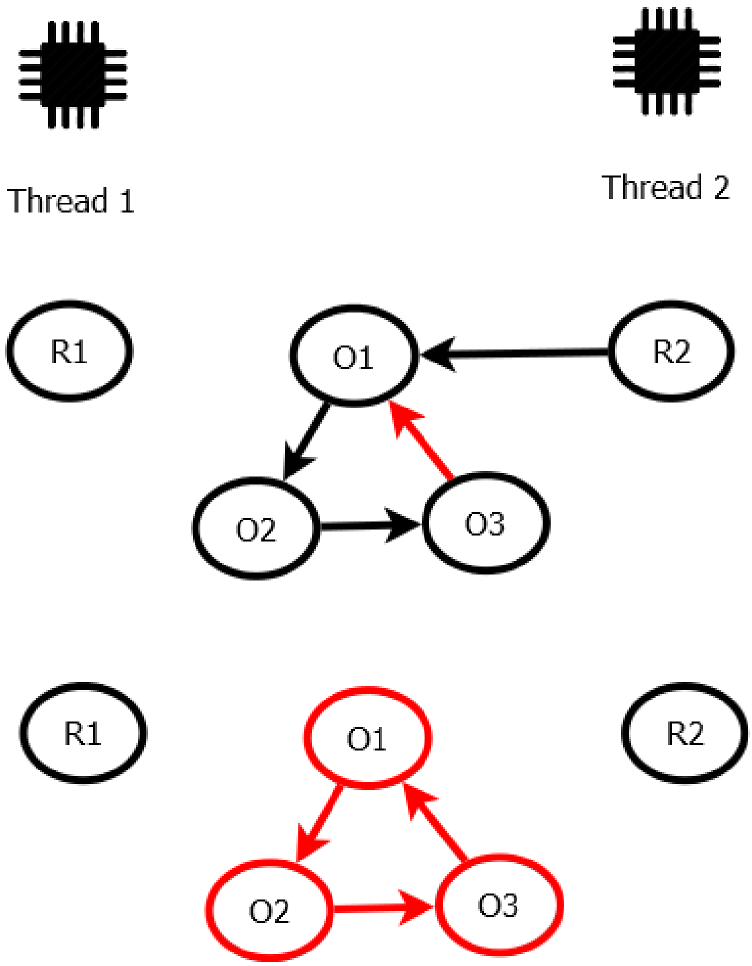
Double-free Bugs (DFB)

- DFB is pointer that points to a deleted object calls delete again.
- Some other object is allocated in the same address in the interim time and creates dangling pointers, or crashes.
- In MMM, simple reference counting based smart pointers can solve this problem in concurrent programming environment in an acyclic graph.
- Otherwise, timestamps are used in lock-free algorithms to avoid this problem (similar to ABA).

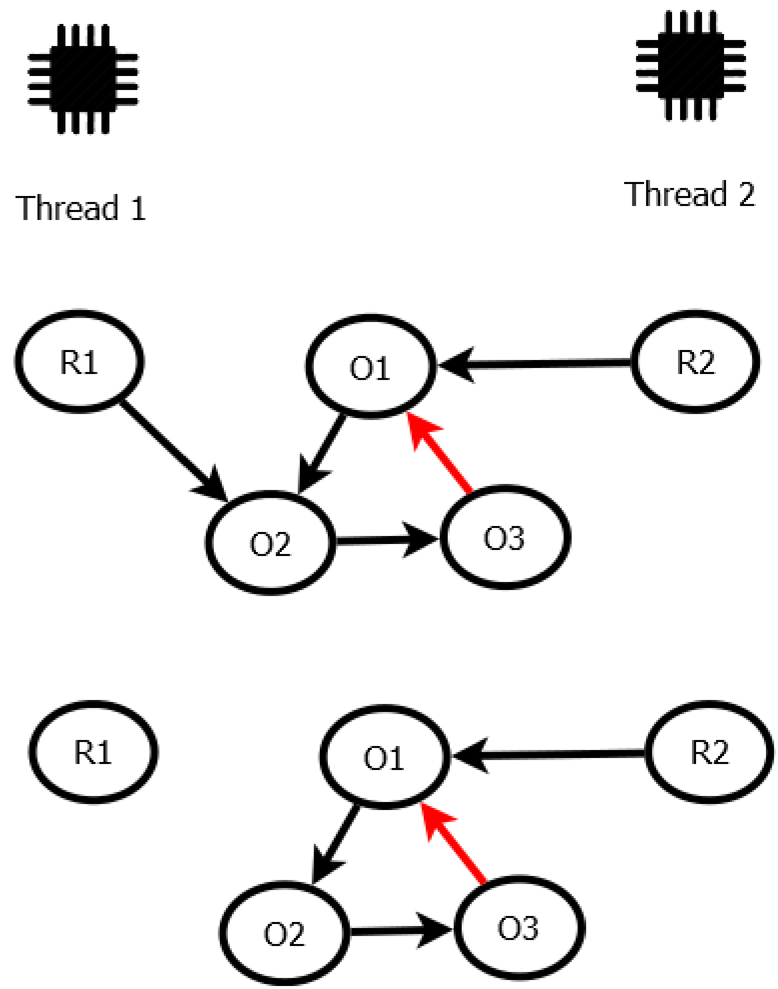
Memory Leak

- Objects in heap memory are not referenced by any roots but not deleted.
- Low memory utilization.
- Cycles in the heap cannot be reclaimed by smart pointers.

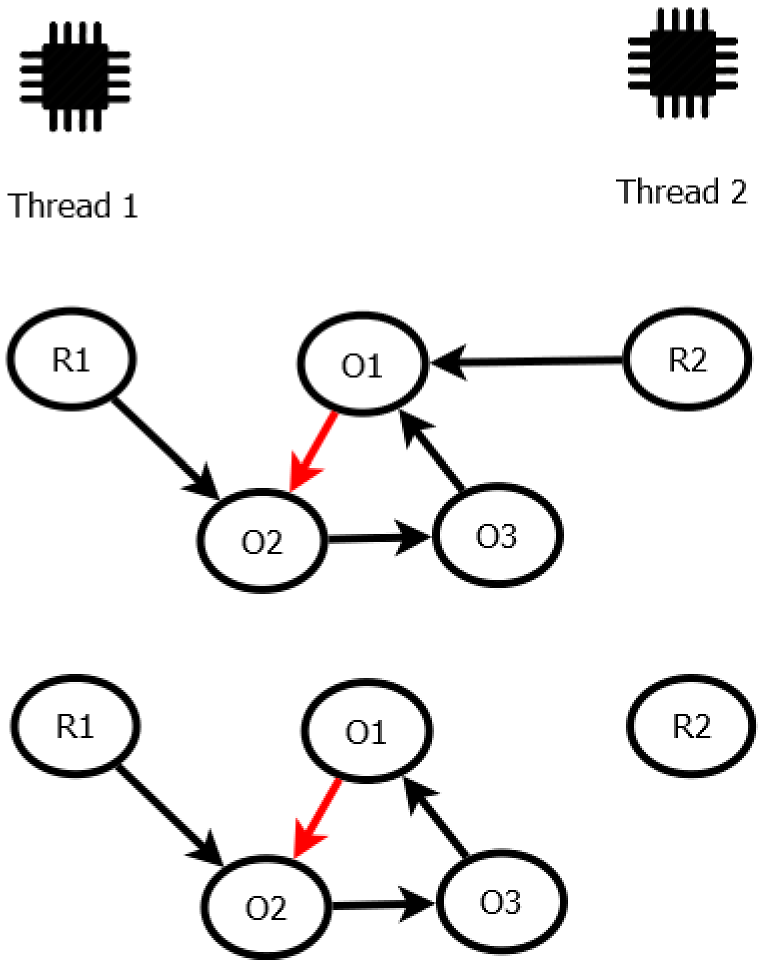
Memory Leak



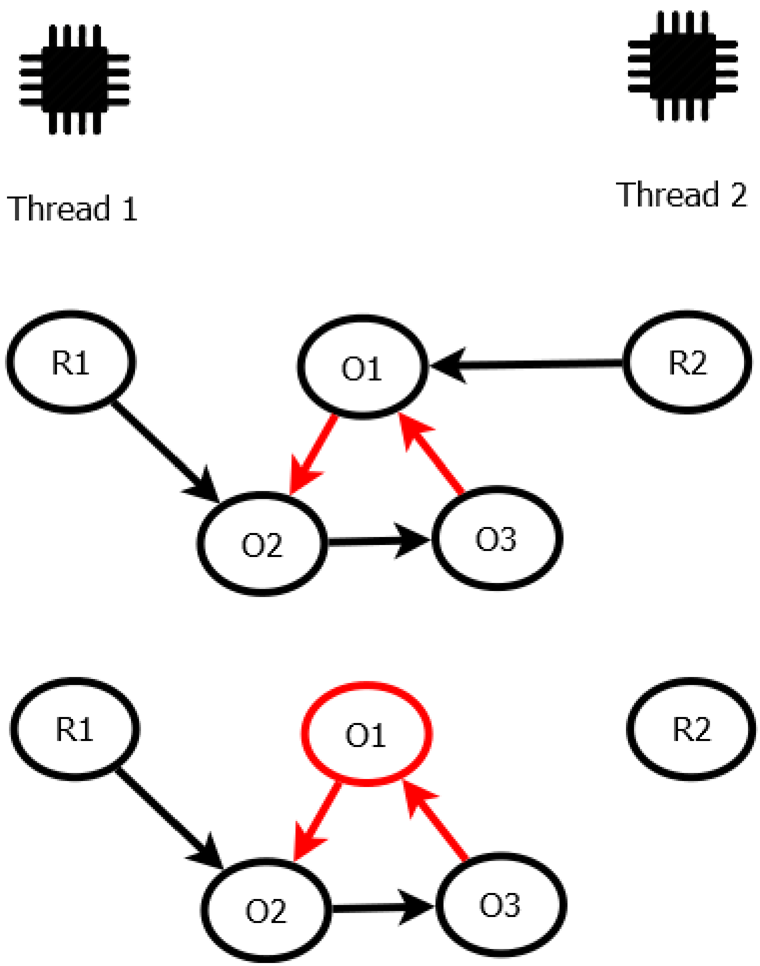
Memory Leak



Memory Leak



Dangling Pointer



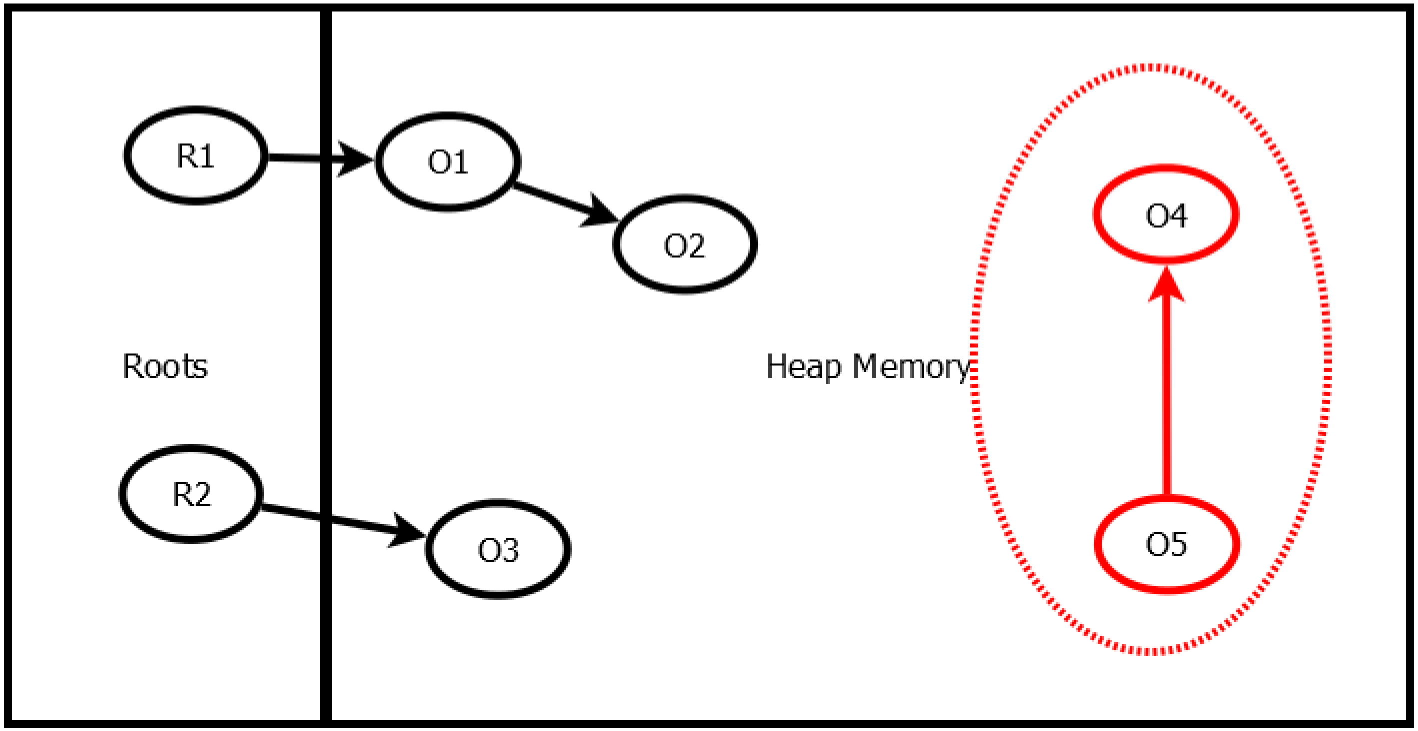
Advantages of GC

- GC has the global knowledge solves the problem of dangling pointers, double-free bugs and memory leaks.
- Reduces number of lines of code to write.
- AHA! Boehm and Spertus announced that in the next C++ standard, GC can be expected!
- Commercial Software Development research claims GC reduces development cost. [Butters, 2007]
- Useful for distributed object stores, parallel and distributed programming languages, distributed databases, and WWW.

Mark-Sweep

- When the amount of memory consumed reached its threshold, the collector starts marking the nodes reachable from the roots.
- 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.

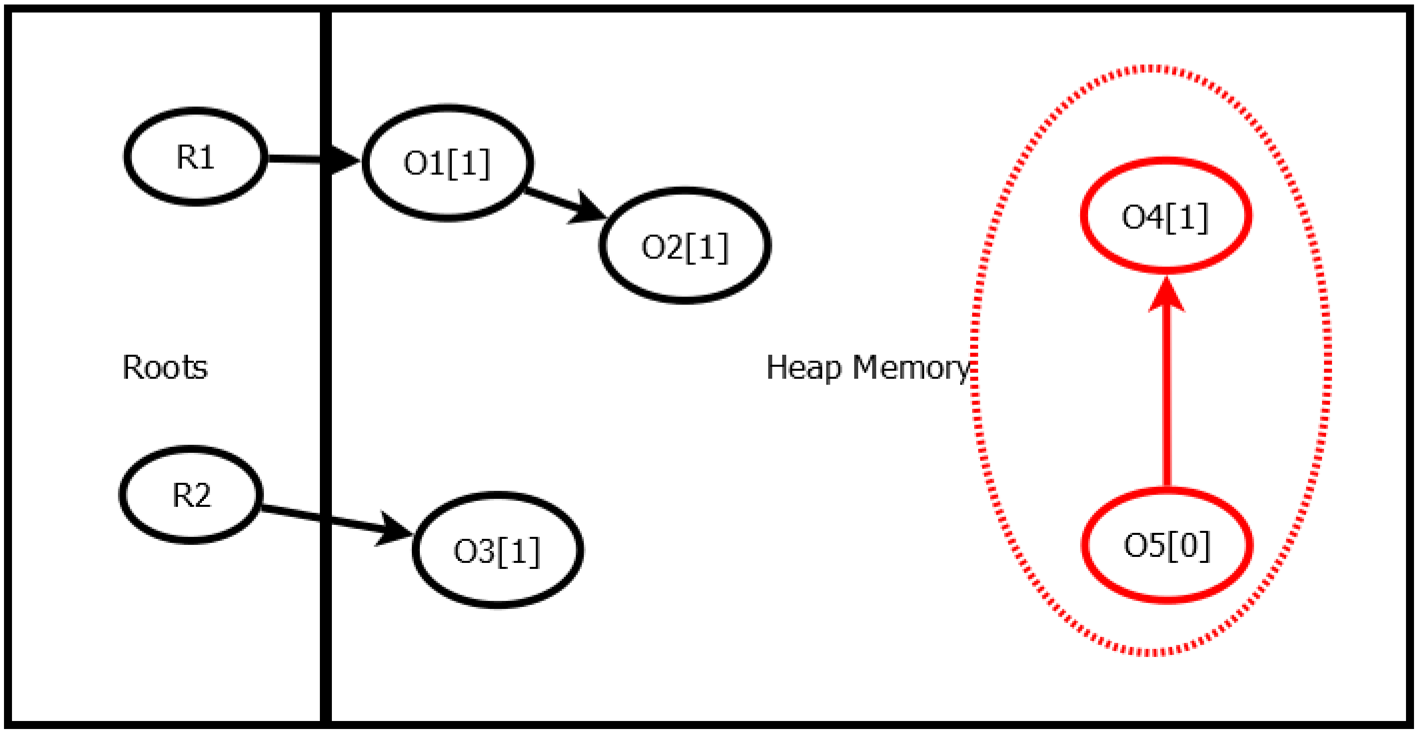
Mark-Sweep



Reference Counting (RC)

- Each object has a RC that denotes how many incoming references it has.
- When an object has zero RC, it is garbage.
- The method cannot delete cyclic garbage as all cyclic garbage nodes have positive RC.
- Apart from the inability to detect cyclic garbage, reference counts have to be updated for object when new reference is made or deleted.

Reference Counting



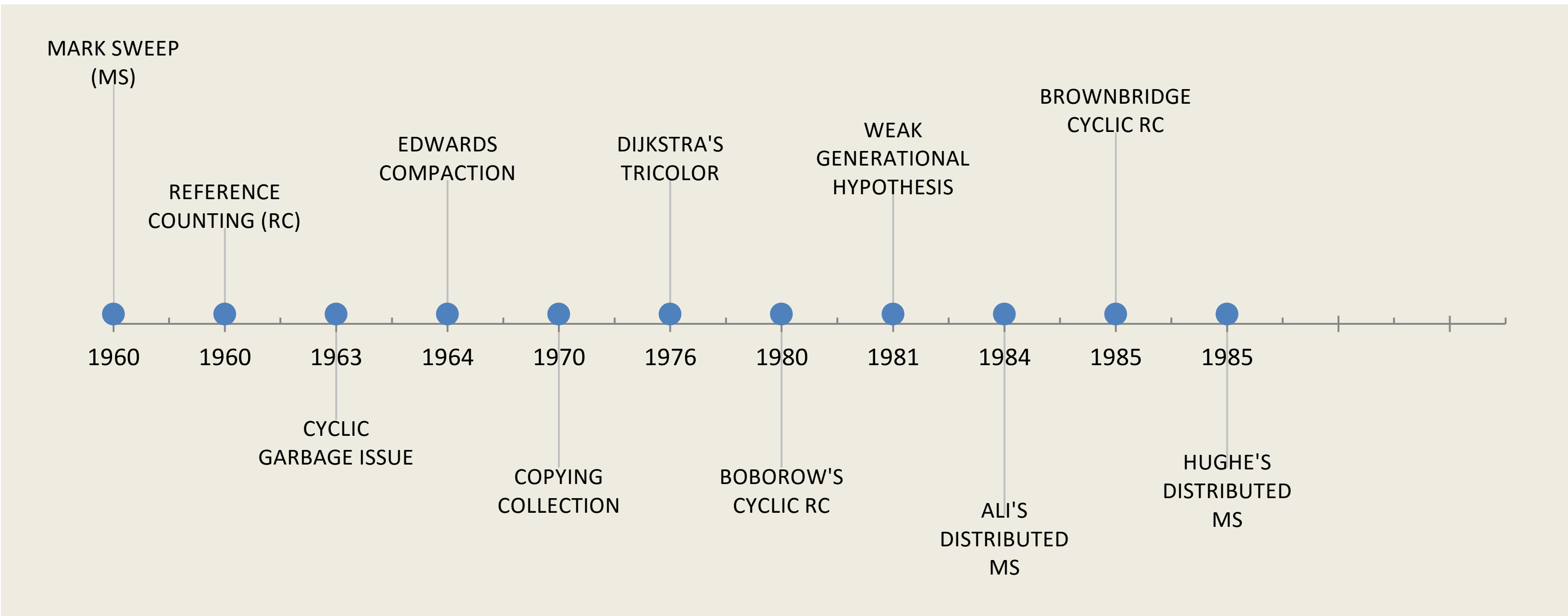
Proposed Hypothesis

- Concurrent GC (Low Pause Time)
- Multi-collector GC (Parallel and High Throughput)
- No global Synchronization (High Throughput)
- Locality-based GC (High Throughput)
- Scalable
- Prompt
- Safe
- Complete

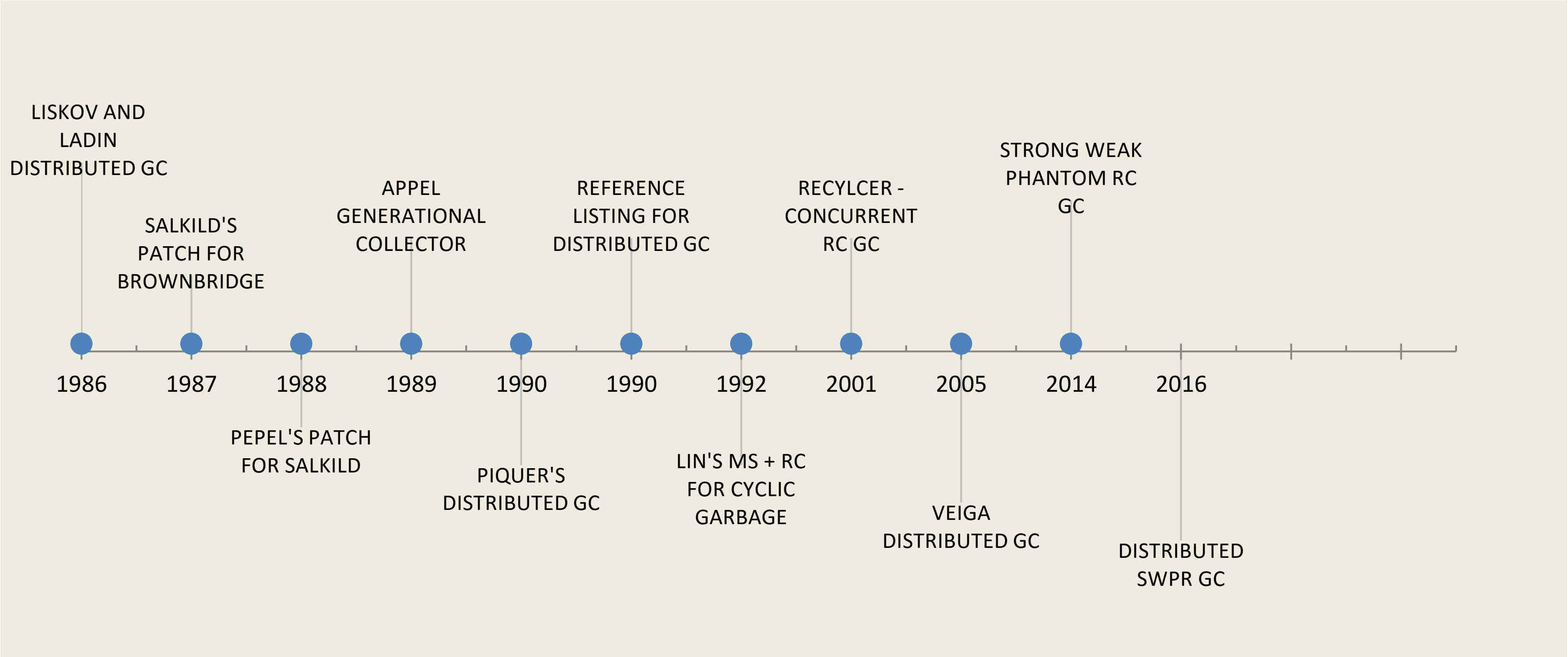
Literature Review

Name	C	MCA	GS	Locality	Scalable	Safe	Complete	Prompt	S/D
Mark Sweep	Yes	Yes	Yes	No	No	Yes	Yes	No	S & D
Reference Counting	Yes	No	No	Yes	No	Yes	No	Yes	S
Cyclic Reference Counting	Yes	No	No	Yes	No	Yes	Yes	Yes	S
Generational	Yes	No	Yes	No	No	Yes	Yes	No	S
Liskov	No	No	Yes	No	No	Yes	Yes	No	D

Literature Review



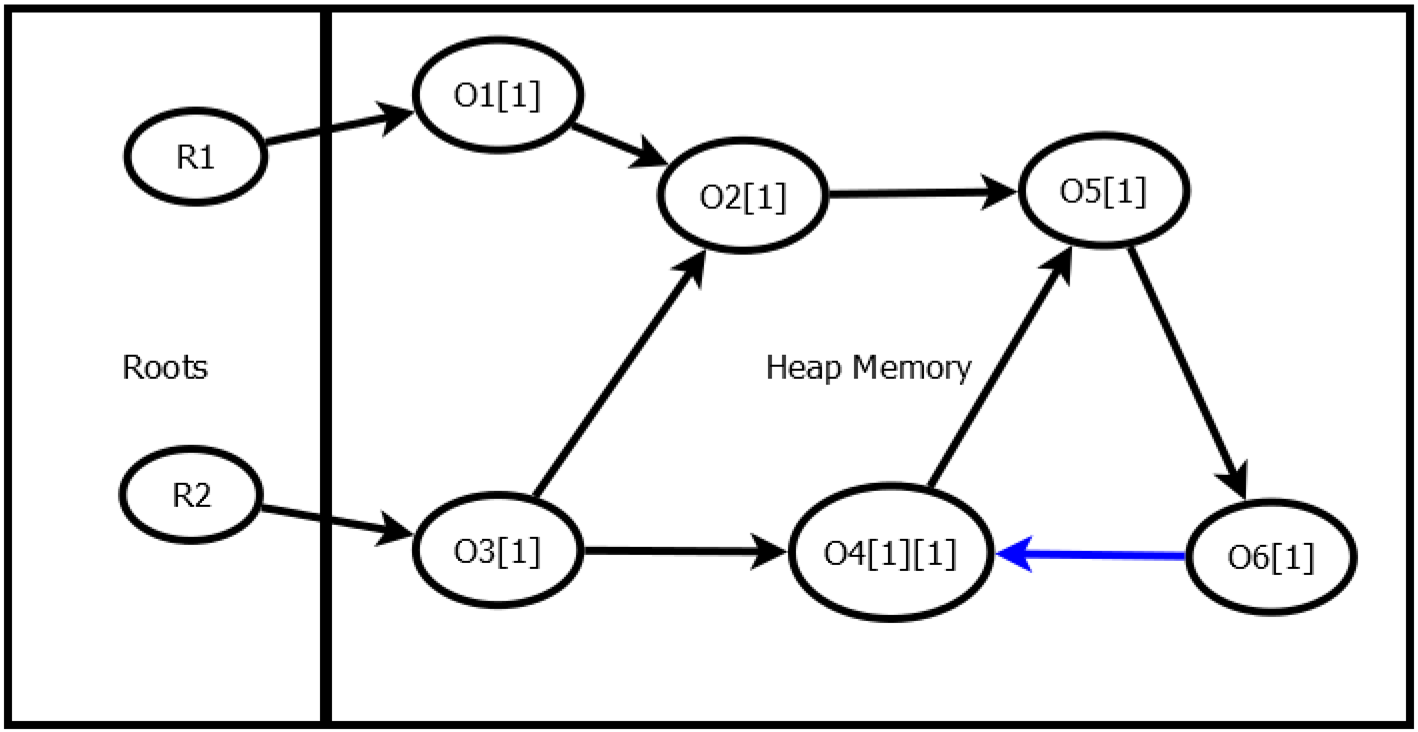
Literature Review



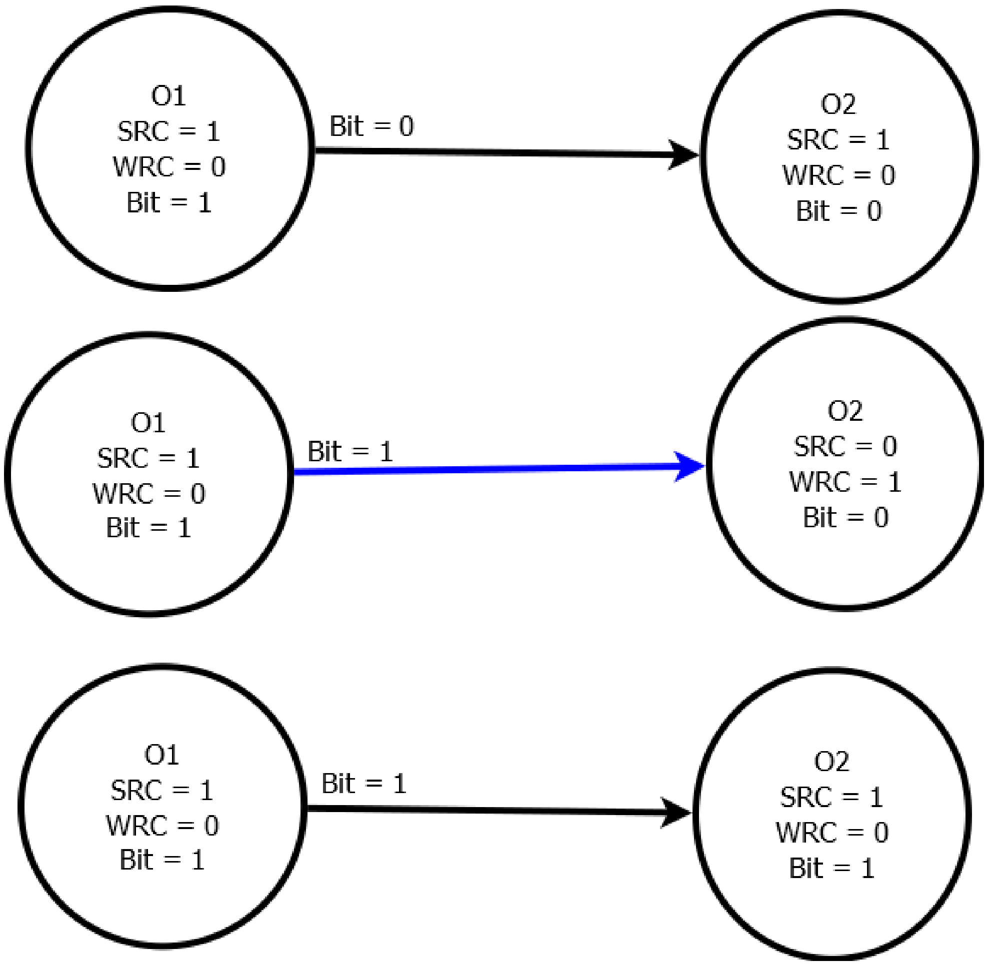
Brownbridge Method

- RC and tracing technique to collect cyclic garbage.
- It uses two reference counts: Strong Reference Count (SRC) and Weak Reference Count (WRC).
- All live nodes have positive SRC.
- No strong cycles are allowed.
- When a reference is created to a node, if the target node already has outgoing references, it is considered a weak reference.

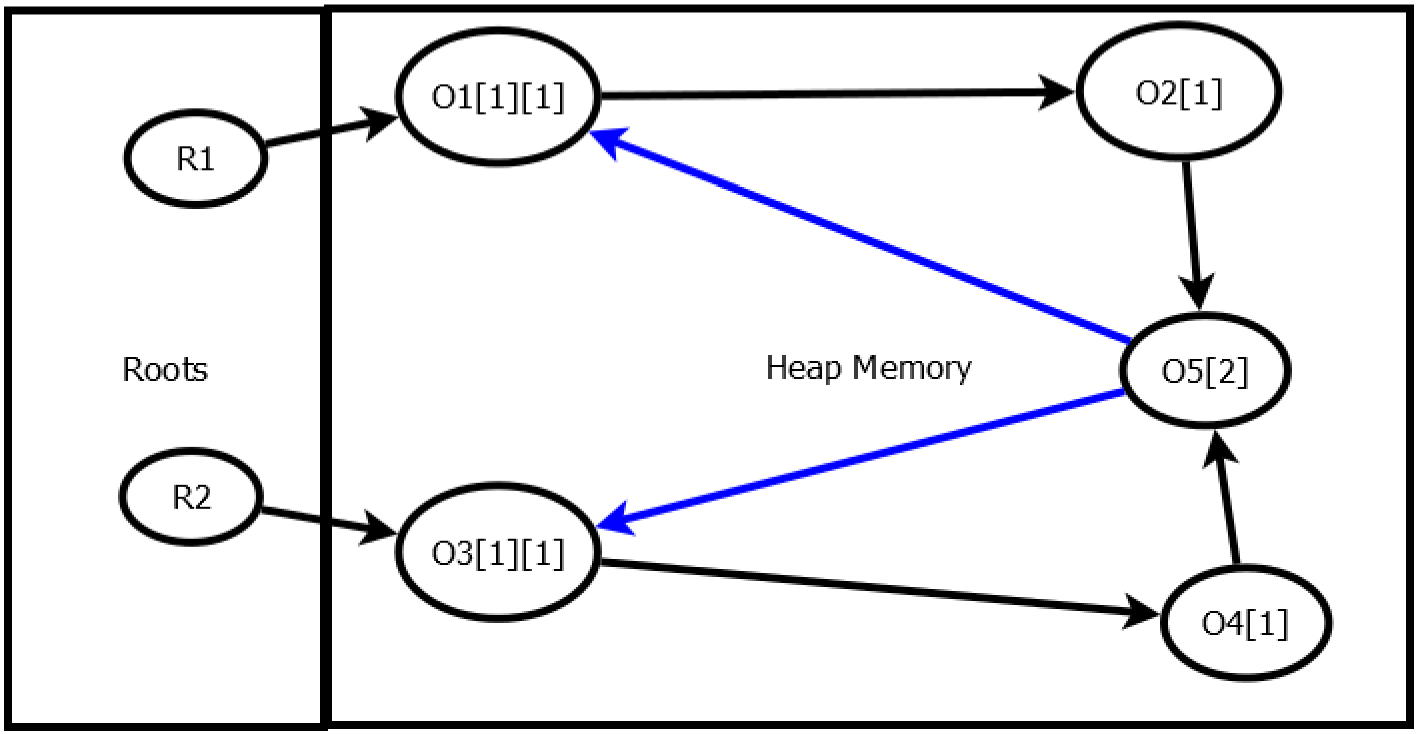
Brownbridge Method



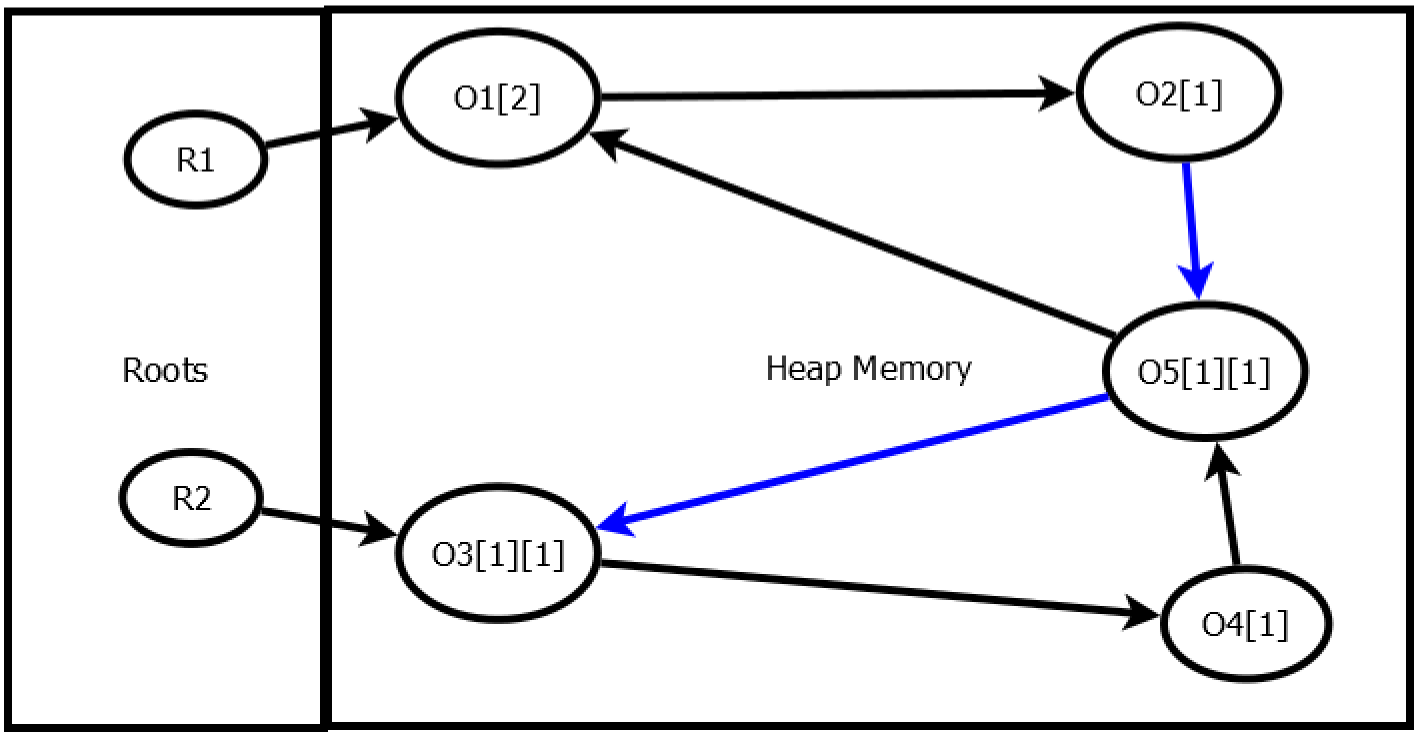
Brownbridge Toggle



Recover Case



Premature Delete Case



Salkild's and Pepel's Work

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

Recycler

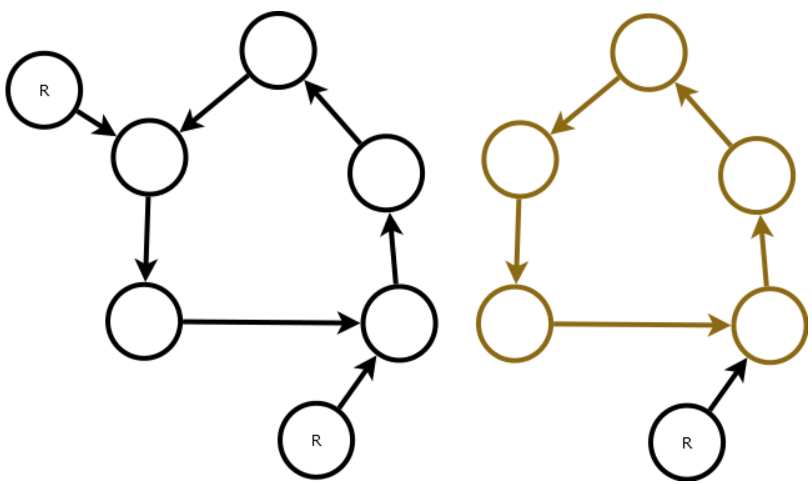
- David Bacon et al used the color algorithm to design a hybrid concurrent garbage collector.
- This method uses reference counting along with tracing to identify cyclic garbage and runs concurrently.
- It traces the entire connected graph and cannot be made to run parallel.
- Antony Hosking et al claims that Bacon’s method is incomplete and proof is insufficient.

SWP

- Strong-Weak-Phantom(SWP) Reference Count GC.
- The idea is to create a path from a root to each live node through strong references and avoid creating cycles of strong reference by using weak references.
- **Strong Cycle Invariant :** No strong cycles will exist in the reference graph at any point in time.
- **Weak Heuristic** : All weak edges are not cycle closing edges.
- **Edge Label Heuristic**: An edge will be weak when the target node has outgoing edges at the time of creation.
- Phantom is a transient state that is used only in the cycle detection algorithm.

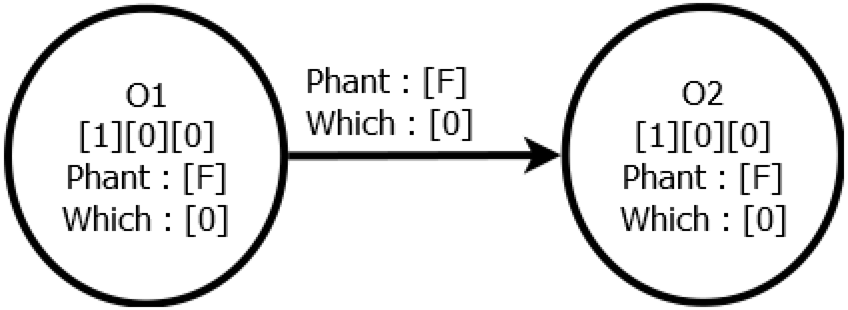
SWP

Counters	State
SRC > 0	Live
SRC = 0 & WRC > 0	Potential Cyclic Garbage
PRC > 0	GC processing node
SRC = 0 & WRC = 0 & PRC = 0	Garbage



SWP & Header

- Apart from reference counts, each node also has a which bit, phantomized flag, and an array of which bits for outgoing references.
- Process lists and request queues are used.



Delete Edge

When a node loses its last strong reference:

    if(WRC==0 && PRC==0)

        start deleting the node and delete all outgoing edges

    else if(WRC>0)

        convert all the incoming weak references to strong and phantomize outgoing edges

Three phases

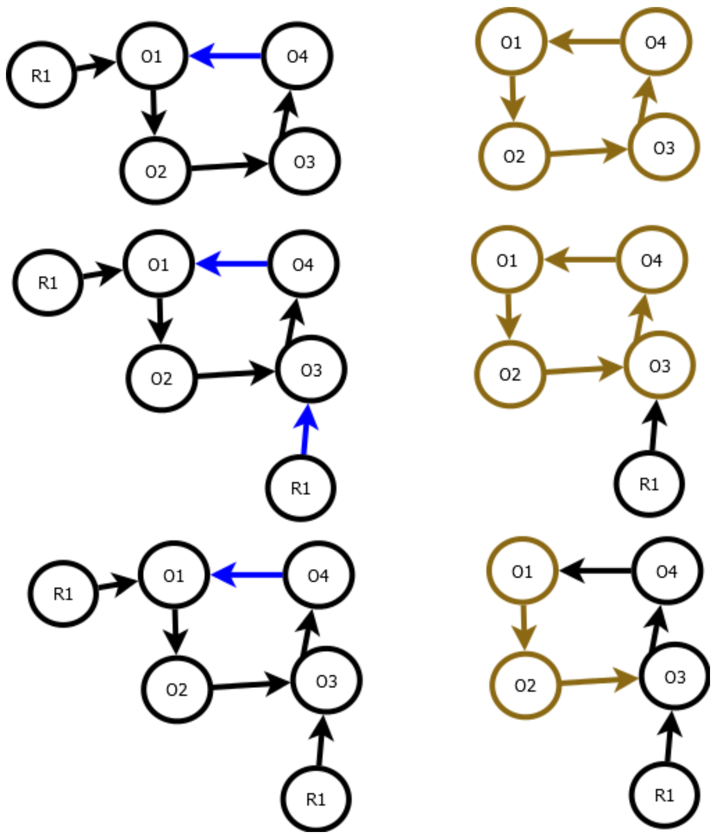
- Phantomize
- Recovery
- Cleanup

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

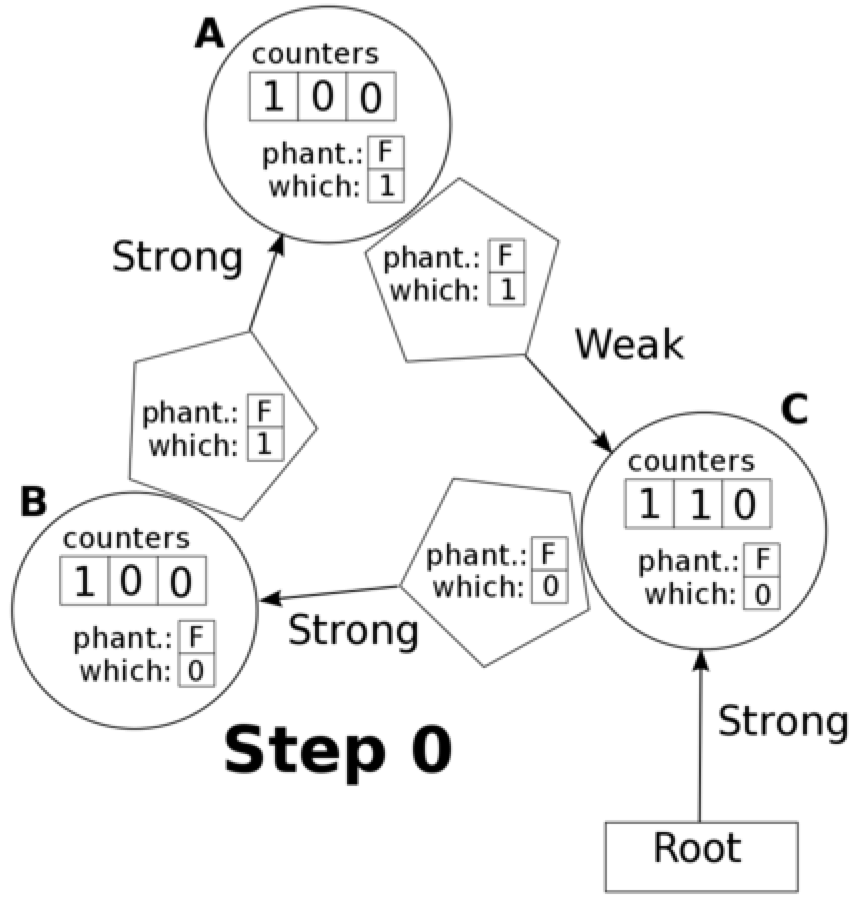
Phantomization

- Decrement all internal references.
- Phantomization stops when it reaches a phantomized node, or a live node.
- Nodes toggle if any weak reference remain after phantomization.
- Phatomization propagates when any of the stop criteria is not met.

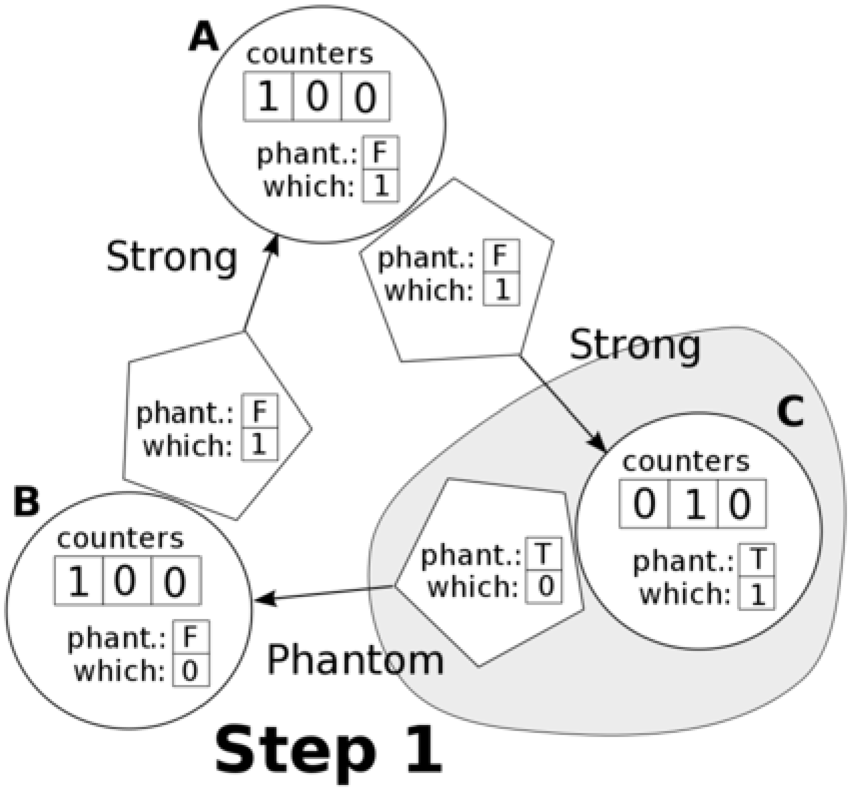
Phantomization



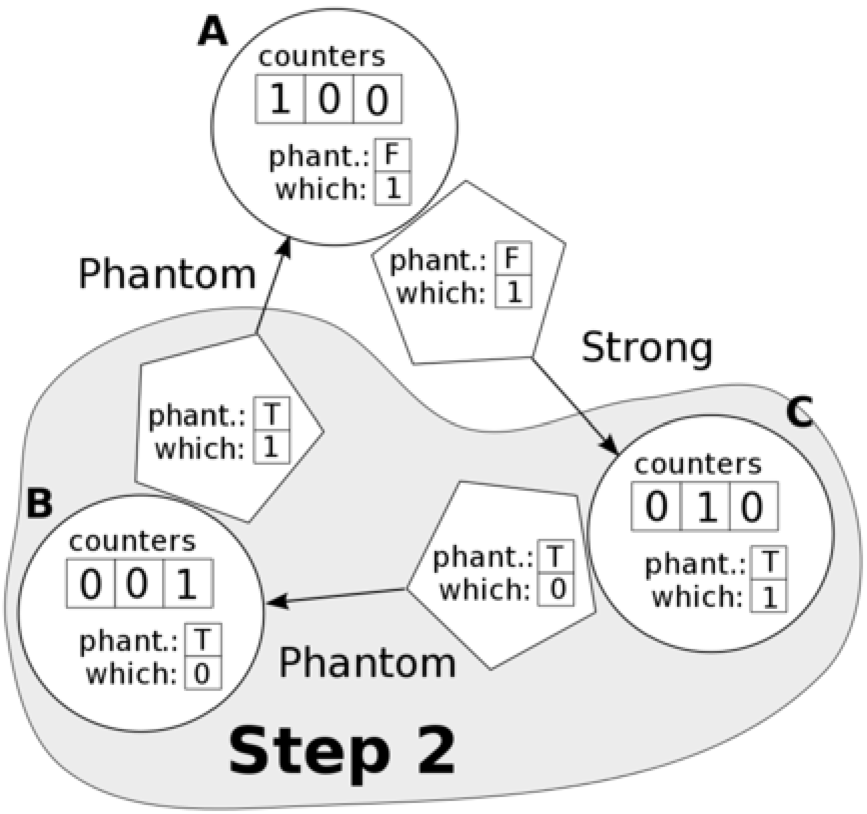
Simple Cycle Demo



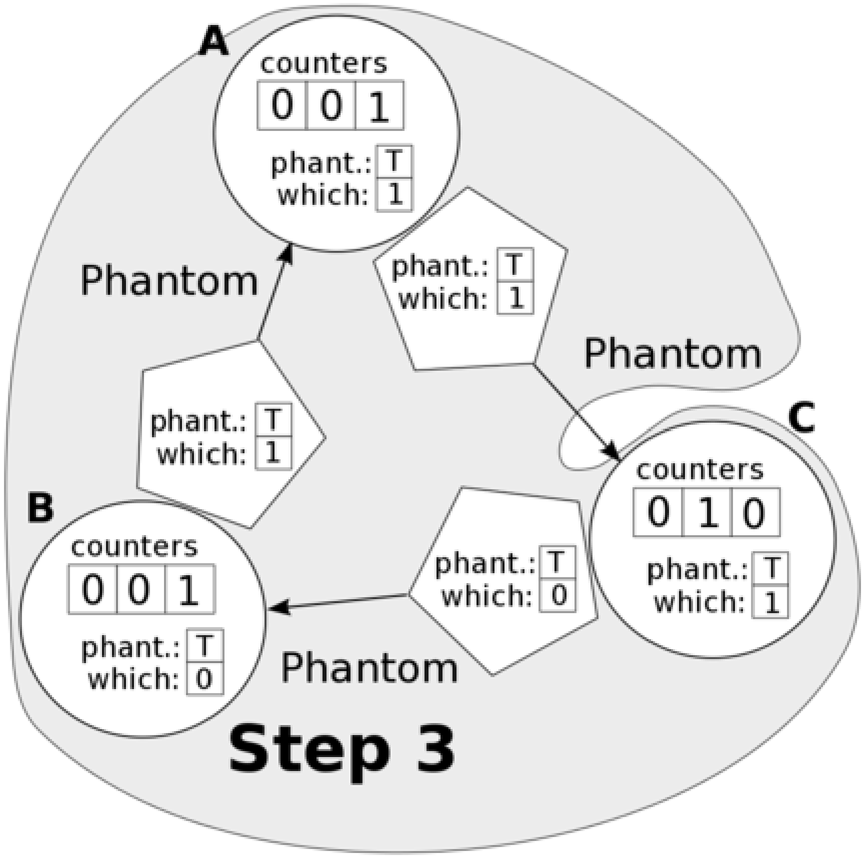
Simple Cycle Demo



Simple Cycle Demo



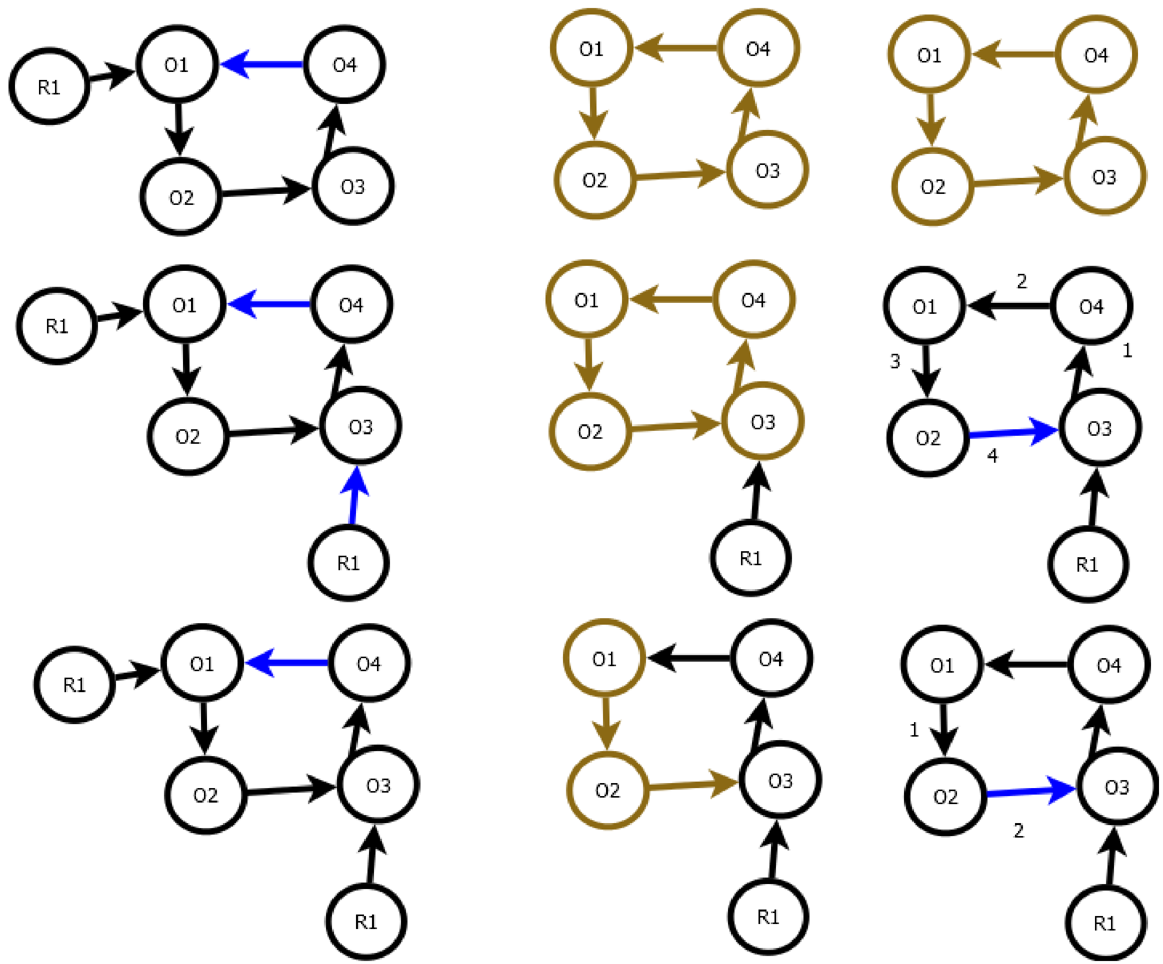
Simple Cycle Demo



Recovery

- If any external reference found, correct the graph to strong / weak graph
- Delete nodes from process list as they are recovered.
- Process starts by scanning all the process list contents and verifying if any of them has any strong references after phantomization.

Recovery


R1
O1
O4
O2
O3
R1
O1
O4
O2
O3
R1
O1
O4
O2
O3
R1
O1
O4
O2
O3
R1
O1
O4
O2
O3
R1
O1
2
O4
3
1
O2
4
O3
R1
R1
O1
O4
O2
O3
R1
O1
O4
O2
O3
R1
O1
O4
1
O2
2
O3
R1

Concurrent and parallel Collector

- Atomic operations can be used to access RCs.
- SWP collection does not stop the application.
- Requests are queued.
- Collector thread processes the queue and starts the requested tasks.
- Collectors write their own id in collector id in parallel collector mode.
- When the collector visits a node with a different id, then they synchronize.

Concurrent and parallel Collector

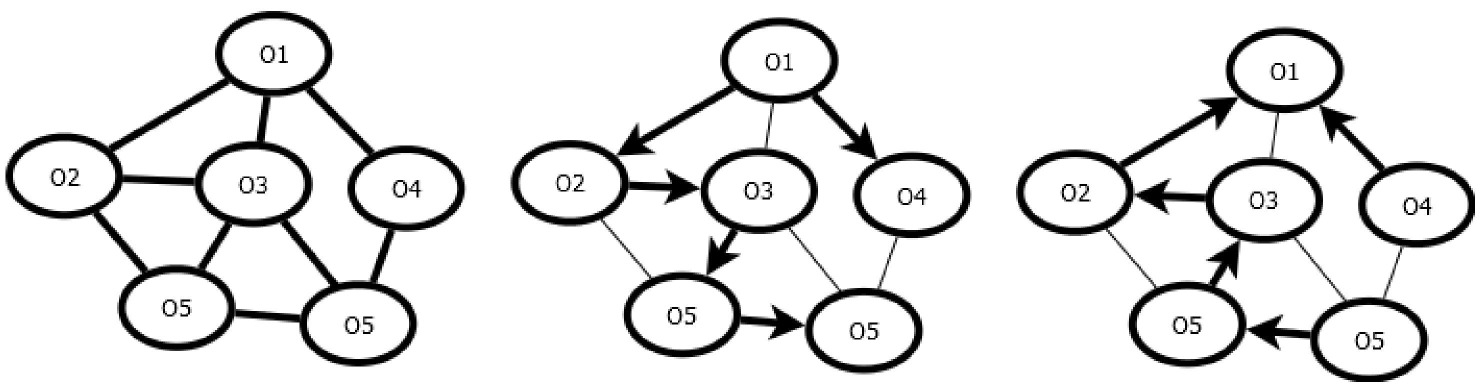
- **No Partition Principle** : When two or more collectors synchronize, they merges their lists of processed nodes and one of the collectors takes over the processing of those nodes.
- Parallel collector adds one more attribute to the object header.

How SWP can be used?

- The SWP algorithm is directly applicable to shared memory systems.
- The SWP algorithm is also applicable to distributed system with centralized queues.
- Unlike other distributed GC systems which require the application to be halted, this one does not require it.
- Mobile actor based collectors do not require centralized queues and are directly applicable to the distributed systems with message size proportional to graph size.

SWPR GC

- SWPR - > (Strong, Weak, Phantom, Recovery).
- Strong Cycle Invariant.
- Weak Heuristic.
- Distributed Termination Detection is used. (Parent attribute and Wait Count).
- SWPR is concurrent, multi-collector, locality-based, scalable, prompt, safe, complete, and functions without global synchronization.

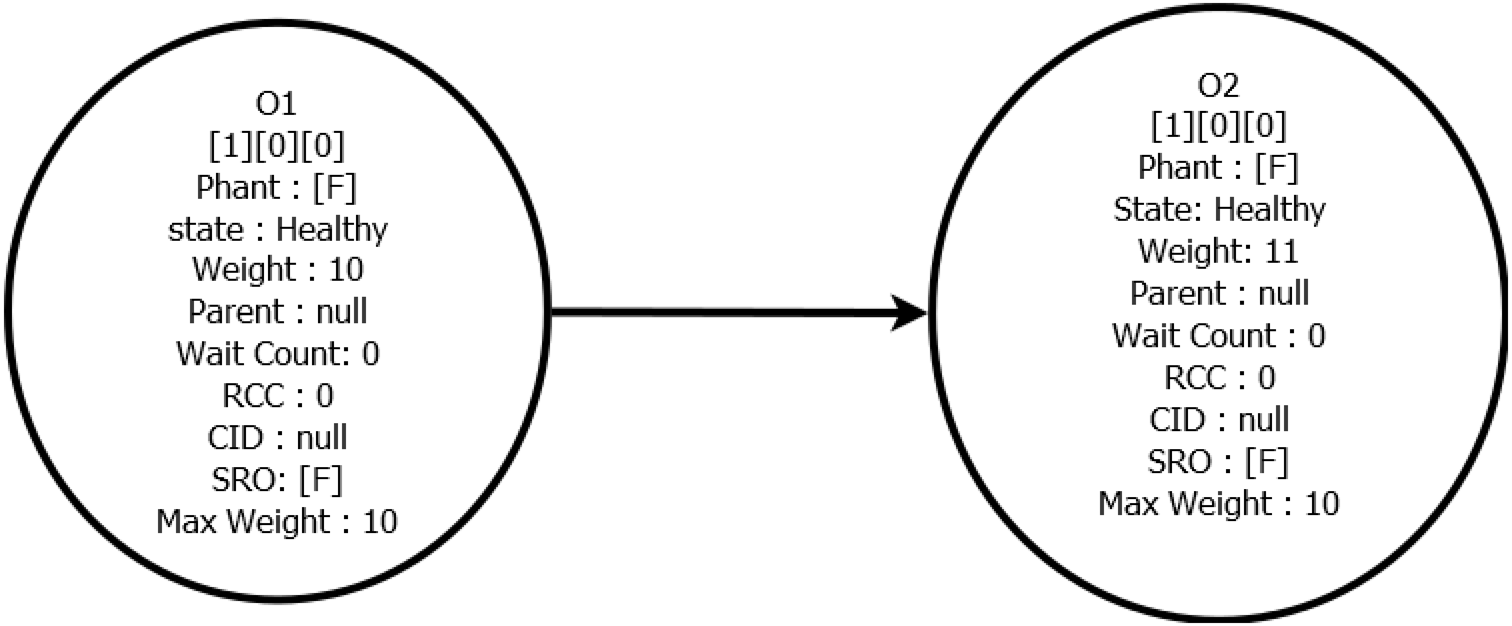


Model

- Congest: O(log n) message size.
- No reference listings allowed.
- Garbage stable property.
- Nodes will only know out-neighbors.
- Nodes can send messages only to neighbors.
- Asynchronous network model.

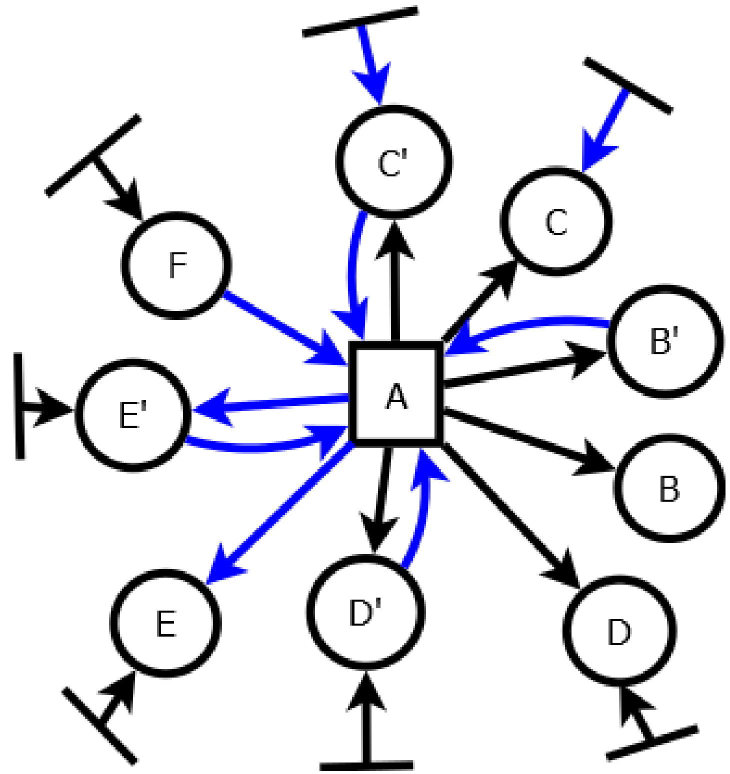
Weight System

- Weights of source and target nodes determine the type of the edge.
- Strong -> (Weight of Source < Weight of Target).
- Weight of Root = 0.

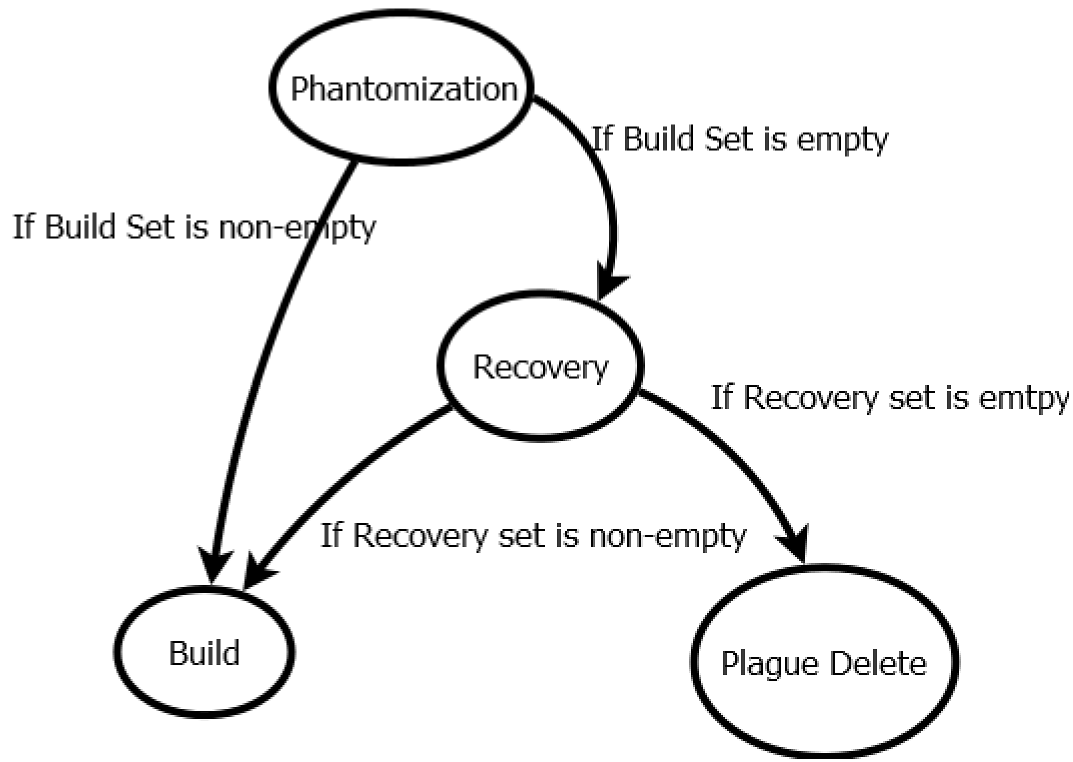


Node Classification

- A - > Initiator
- B, B' -> Purely Dependent Set
- C, C' -> Partially Dependent Set
- D, D', E, E', F -> Independent Set
- C', D', E', F -> Supporting Set
- F, D', E' -> Build Set
- C' -> Recovery Set



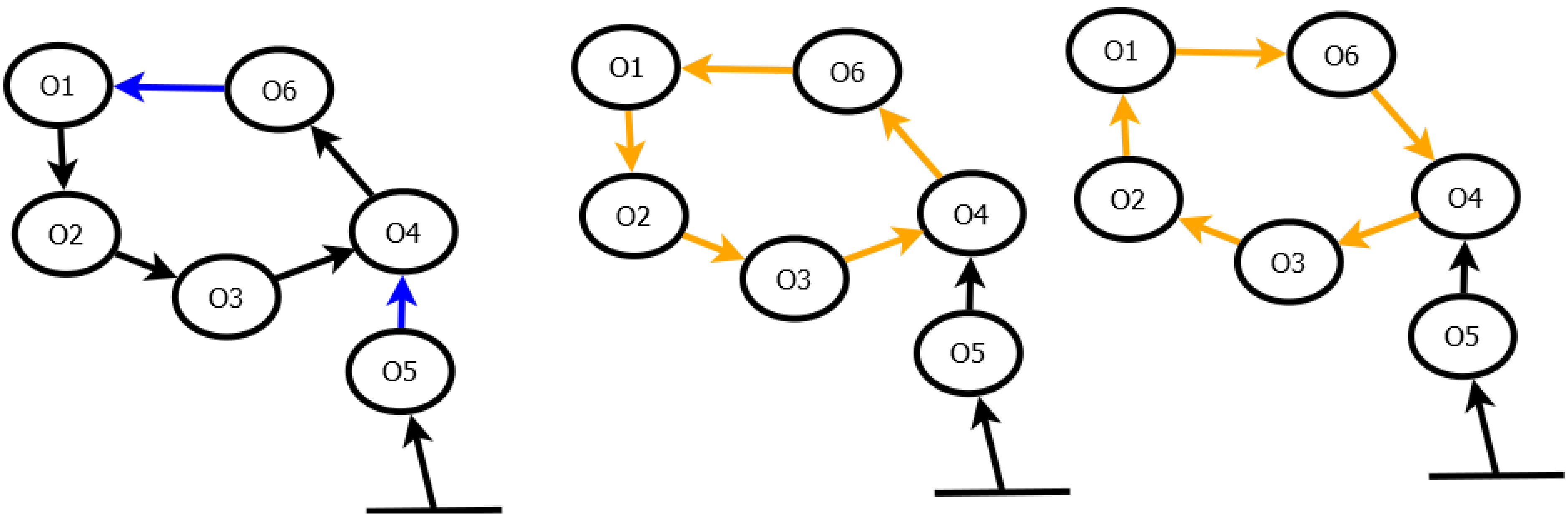
State Diagram



Phantomization

- Converts all the internal edges in the non-independent nodes of the subgraph to phantom.
- The process makes sure the internal edges are not counted for the decision process.
- External weak edges are converted into strong during this process.
- The process uses a forward phase and a backward phase to finish operations.

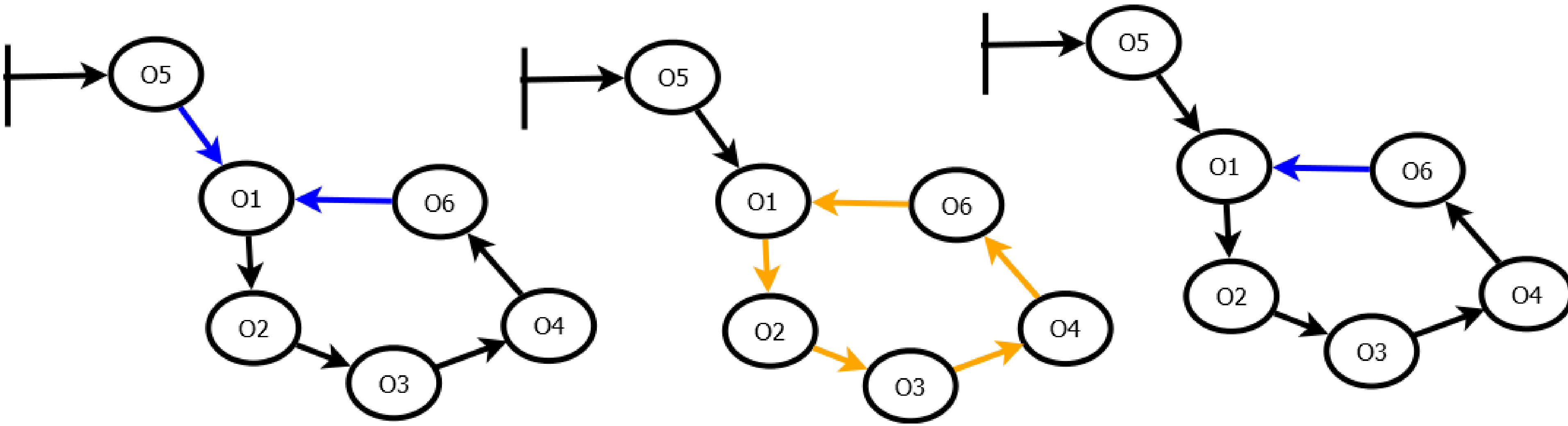
Phantomization



Correction

- Recovery, Build, and Plague Delete are correction phases.
- After phantomization, if the initiator has nodes in the build set, it converts all the phantom edges in the subgraph into strong / weak based on weak heuristic.
- Test to find build set: true If the initiator has any strong edges after phatonmization.
- Build by initiator transforms the graph back to regularity.

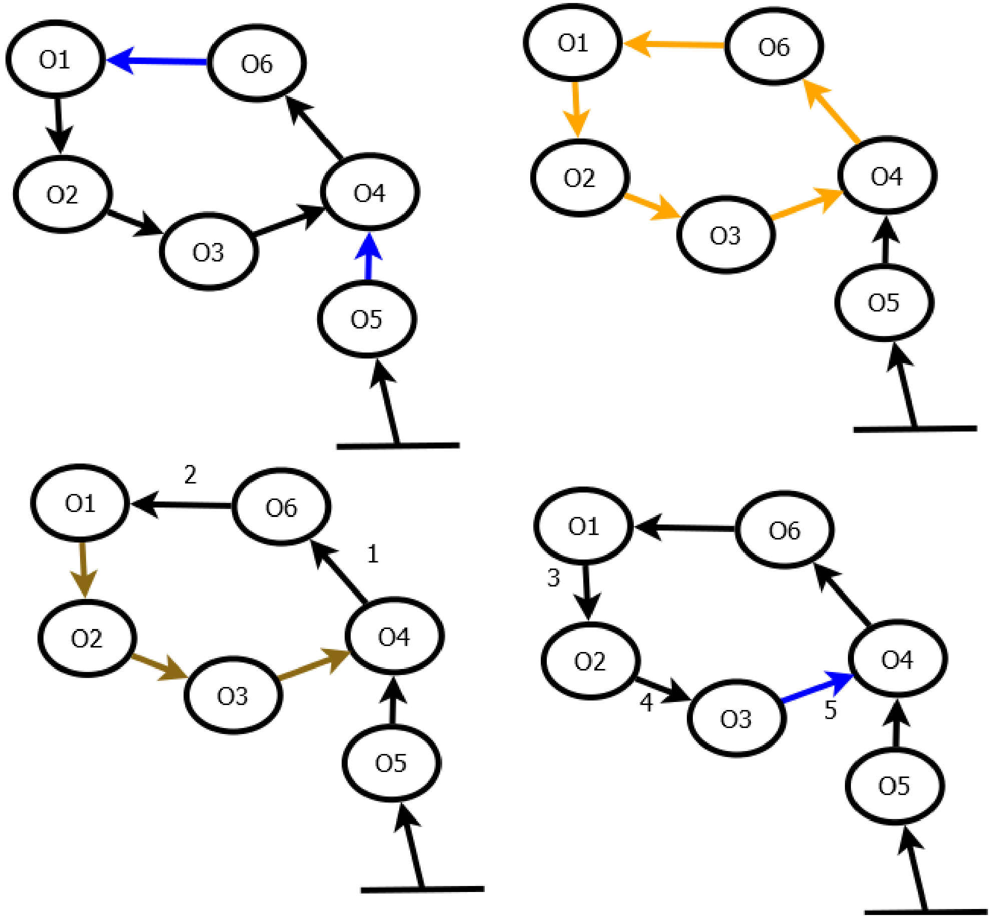
Build Phase



Recovery

- Recovery is a complicated phase.
- If initiator does not have any build set, recovery messages are sent.
- A recovery message will only affect the subgraph that is not yet processed.
- On the reverse phase, a recovery node can start building too.

Recovery


O1
O6
O2
O4
O3
O5
O1
O6
O2
O4
O3
O5
2
O1
O6
1
O2
O4
O3
O5
O1
O6
3
O2
O4
4
O3
5
O5

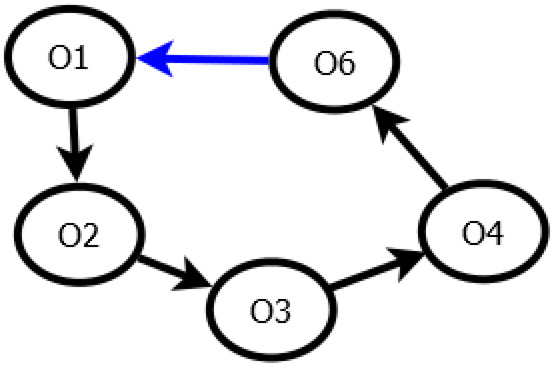
Plague Delete

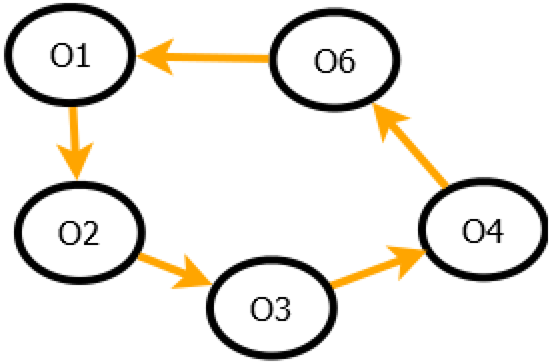
- A node will be deleted only if all the counts are zero.
- A node on receiving plague delete will send plague delete only if there is no strong incoming edge.
- Plague Delete is invoked by the initiator if there is no build set and no recovery set.

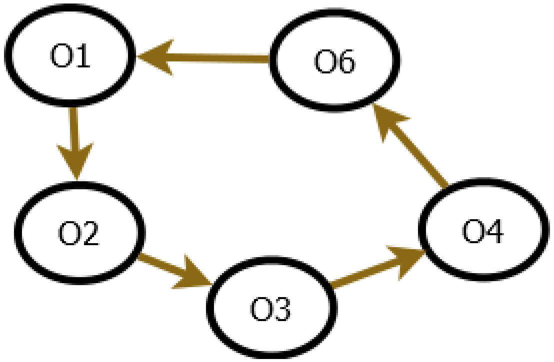
Plague Delete

- A node will be deleted only if all the counts are zero.
- A node on receiving plague delete will send plague delete only if there is no strong incoming edge.
- Plague Delete is invoked by initiator if there is no build set and no recovery set.

Plague Delete







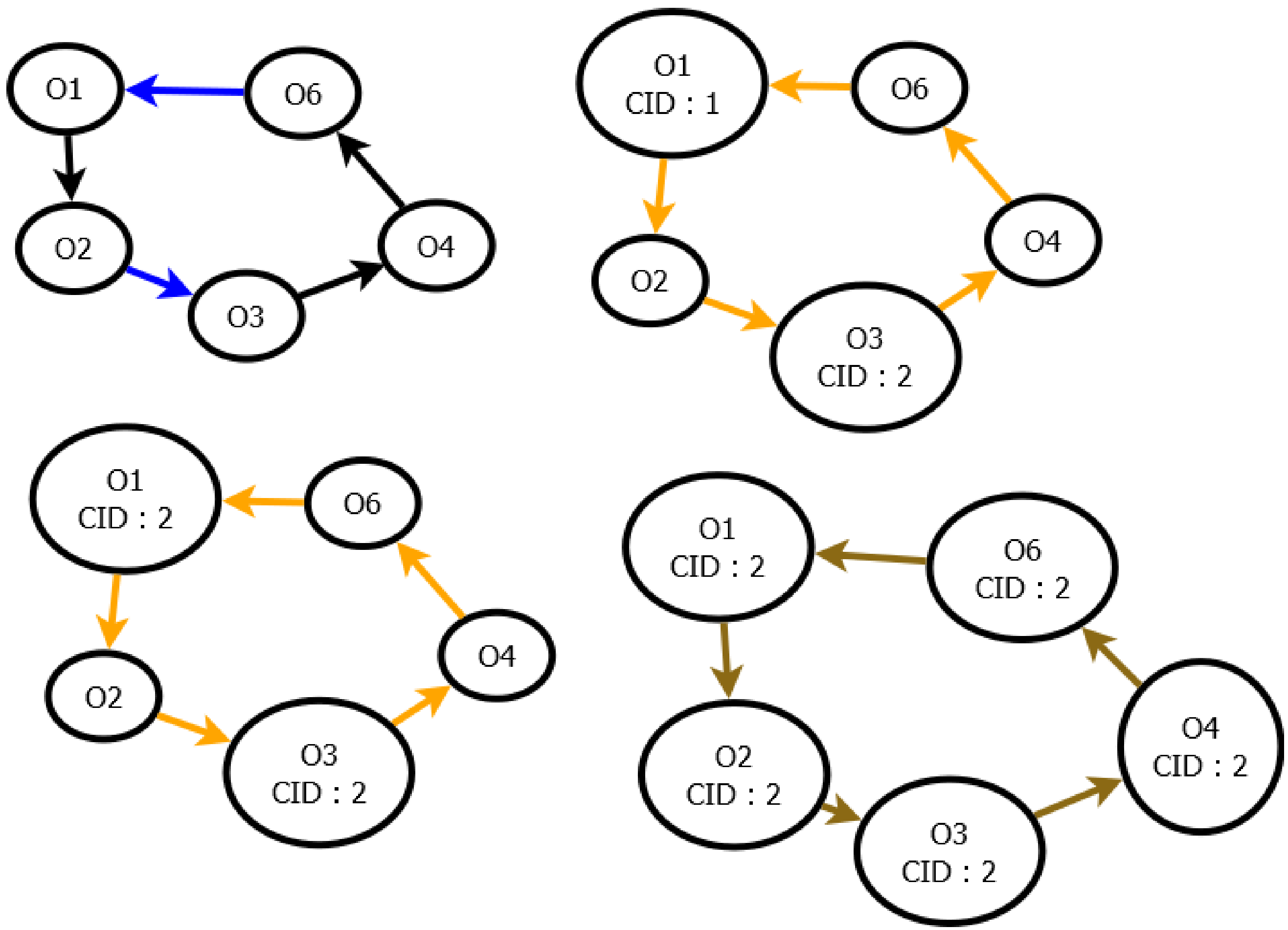
Isolation property

- The affected subgraph is not mutated.
- Mutation to the affected subgraph occurs, but do no affect the decisions of the initiator to delete or build.
- Do not affect the decision of recovery set to build by the correction phases.

Symmetry Breaking

- When multiple collector proces the same subgraph, there is possibility of cycle dependency.
- A leader needs to be elected among conflicting collector operations.
- Each node contains a collector id in the correction phase.
- During the correction phases, nodes are marked with the collector id they belongs to.
- All nodes prefer to be in a collection with higher id.
- All collection ids are unique and so there is a total ordering among collections.

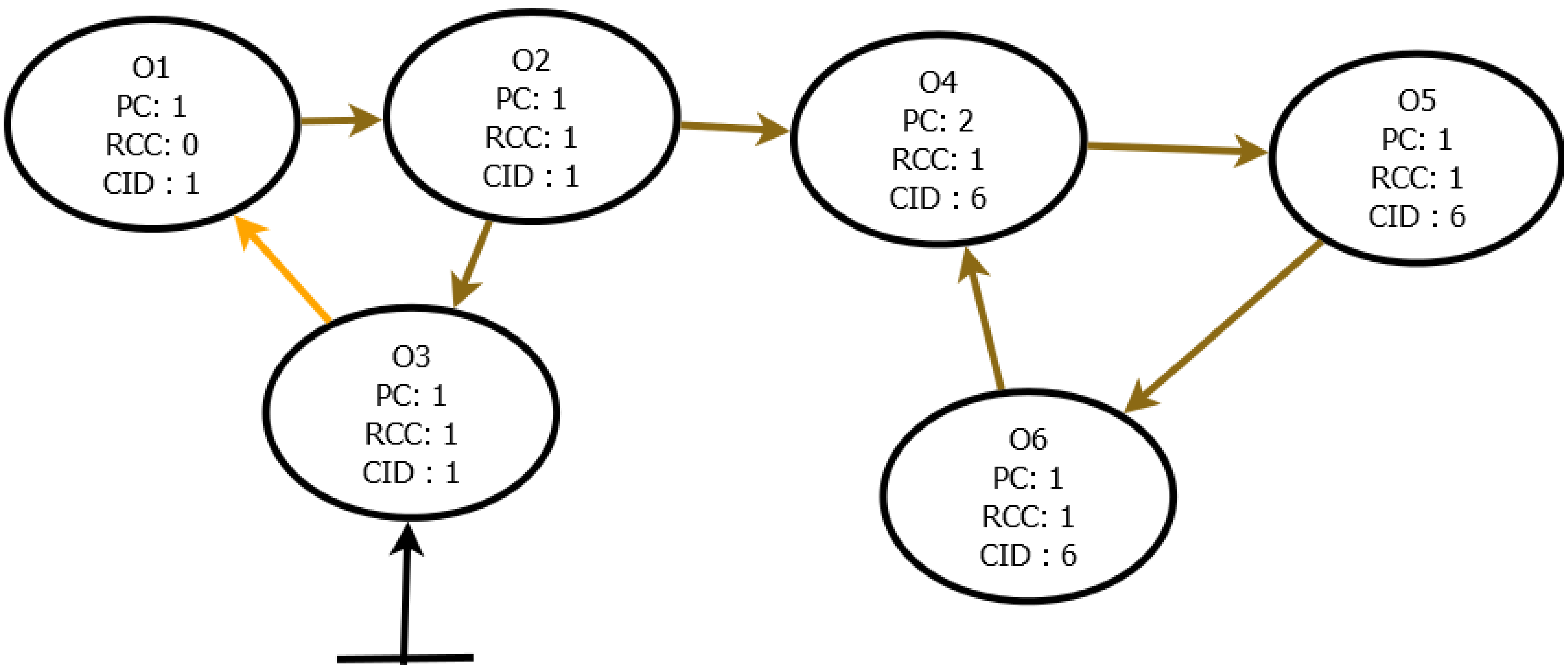
Acyclic Principle (Isolation)



Recovery Count

- For the recovery phase to start reverse phase, it must have recovery count equal to phantom count.
- Every recovery message received increments recovery count.
- This creates ordering among subgraphs based on topology, regardless of uncertainty in the collection ids.
- Low collector id cannot increment higher collection recovery count.

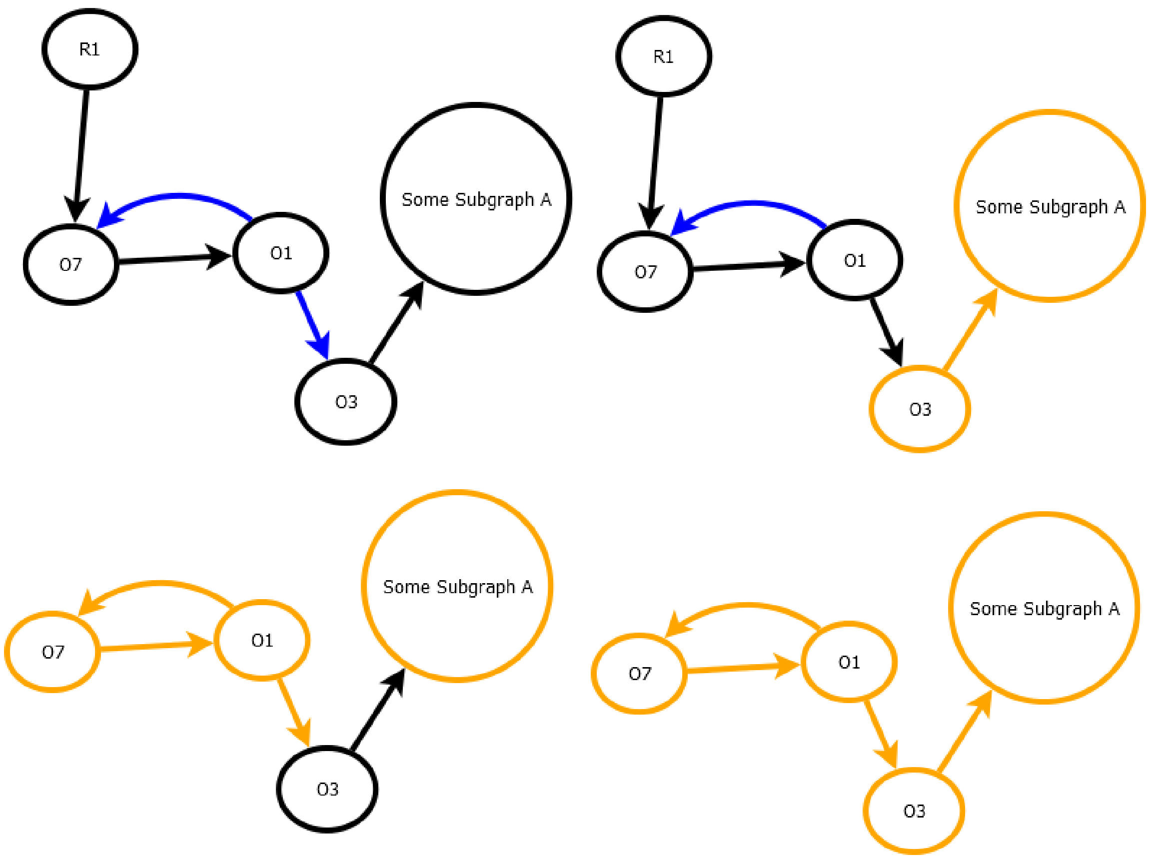
Topology Ordering (Isolation)



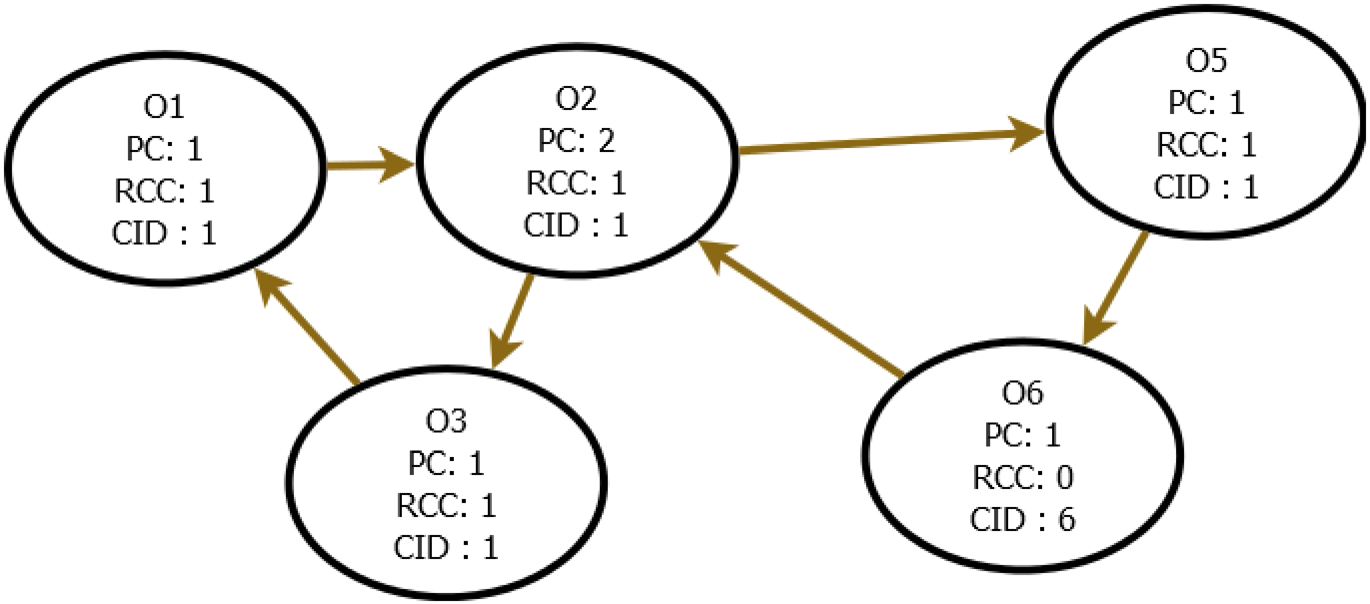
Transactional Approach

- When two collectors’ operations meet at a point, the lower collection id will not proceed because of symmetry breaking rules.
- To complete the computation of the lower collection process during recovery, recovery phases alone are restarted. Other phases work seamlessly with multiple collectors colliding.
- When multiple collectors of different phases meet, we wait until the reverse phase finishes and then redo if they need to upgrade.

Redo transaction (Isolation)



Recovery Start Over



Advantages of SWPR

- Concurrent
- Multi-collector algorithm
- Locality-based algorithm
- No global synchronization
- Scalable
- Prompt
- Safe
- Complete
- O(log n) message size.

Future Work

- Destructor ordering is necessary.
- Fault-tolerance is necessary to make it more reliable.