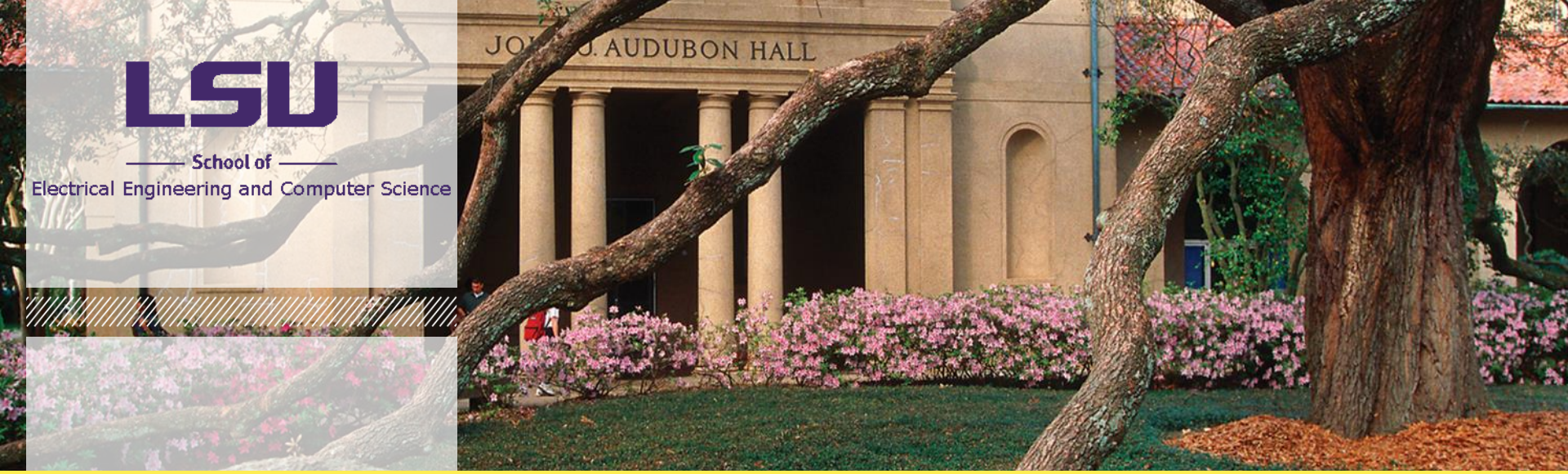


06.10.16

Garbage Collection for General Graphs

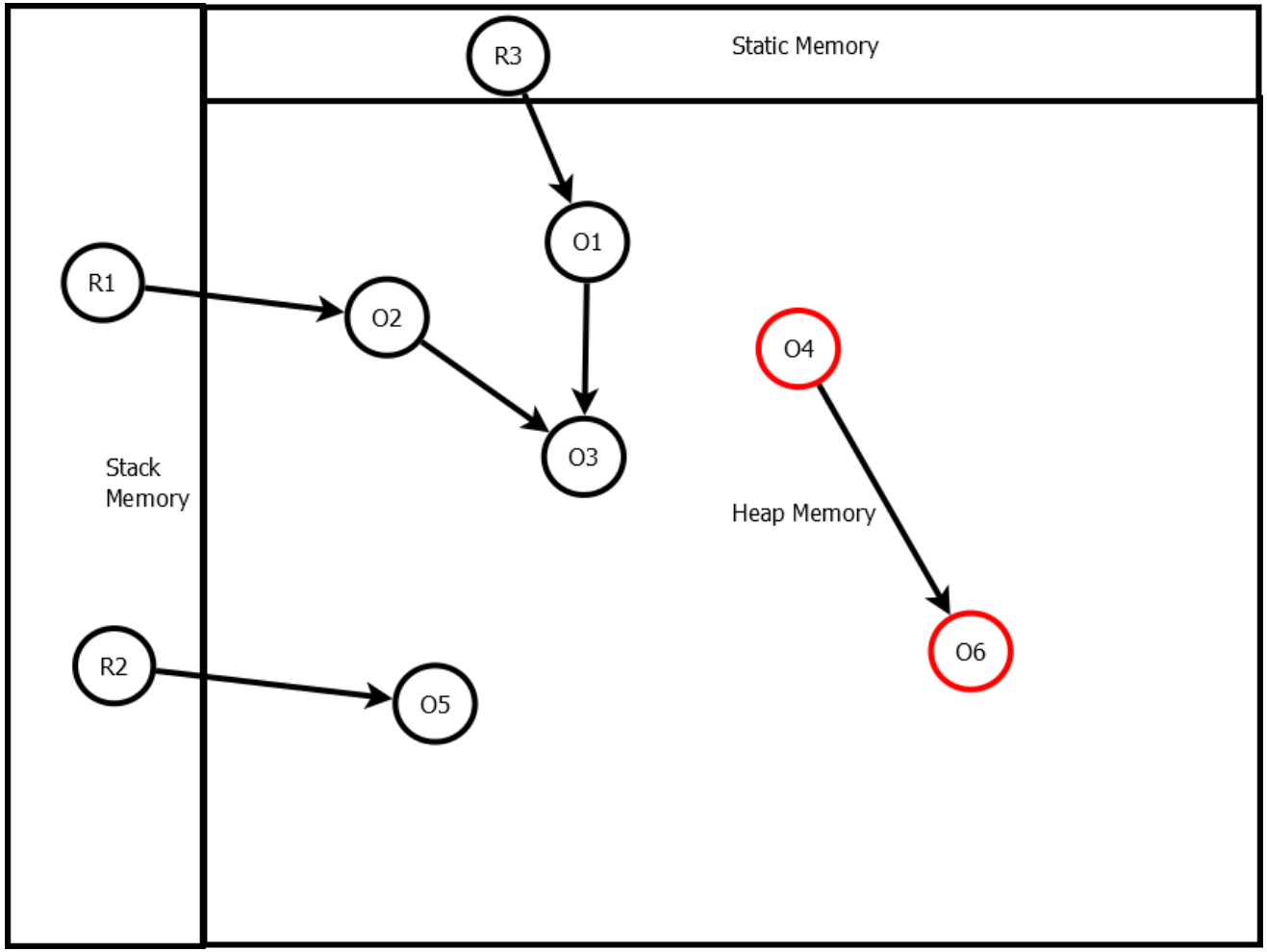
Hari Krishnan

LOVE PURPLE
LIVE GOLD

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)

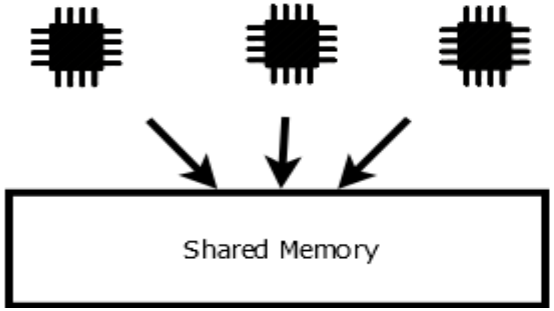


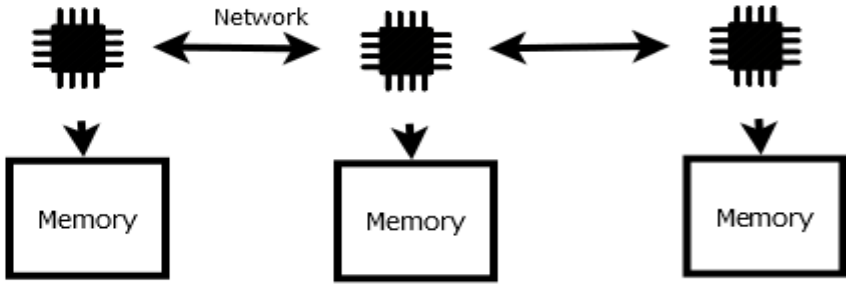
How to collect garbage?

- 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.

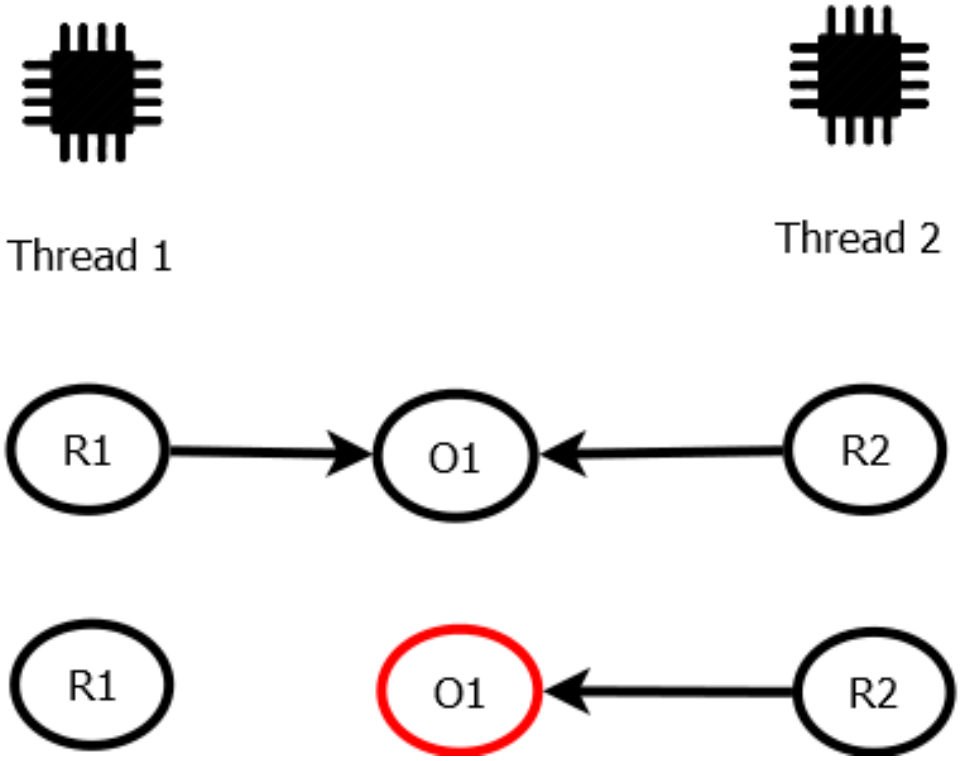
Dangling Pointers

- 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.





Dangling Pointers



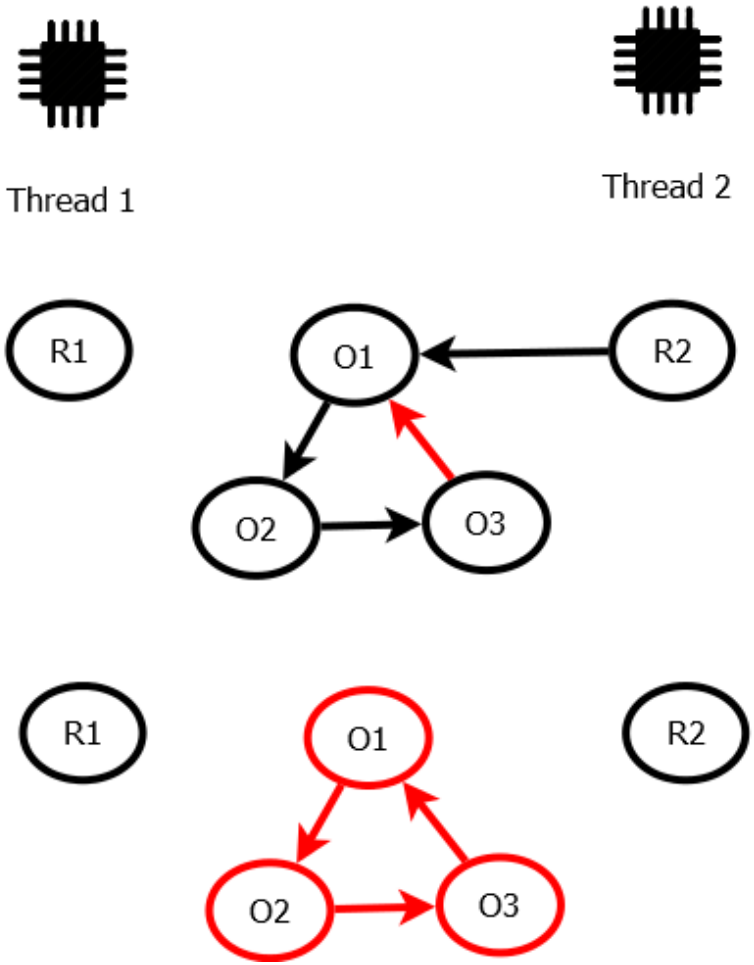
Double-free Bugs (DFB)

- 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).

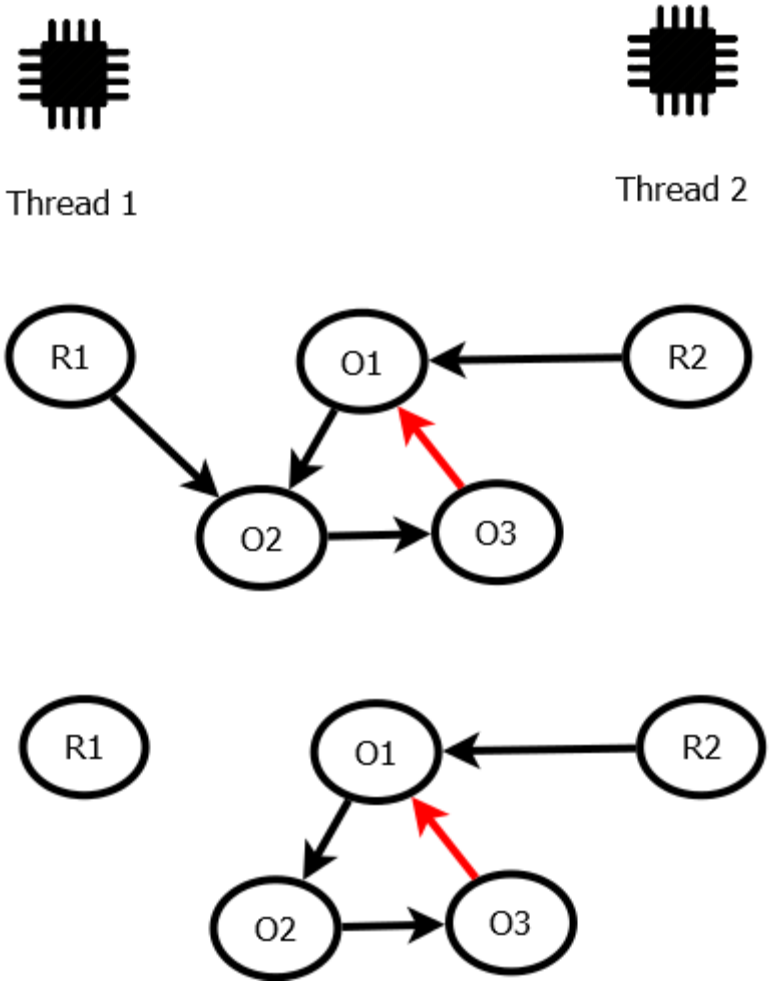
Memory Leak

- 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.

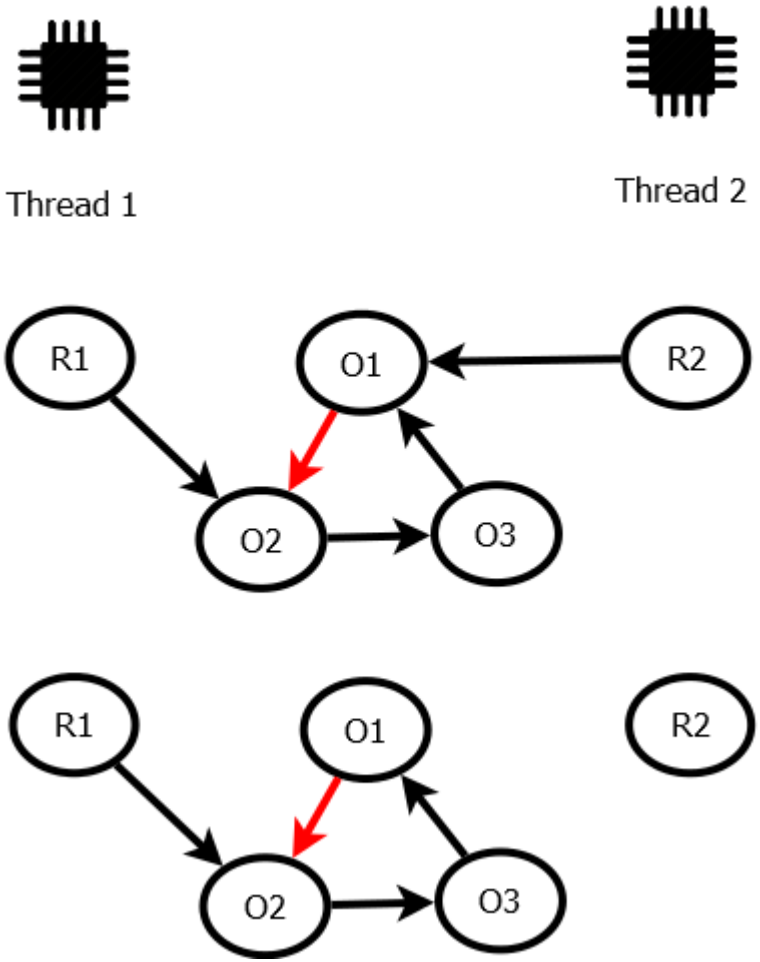
Memory Leak



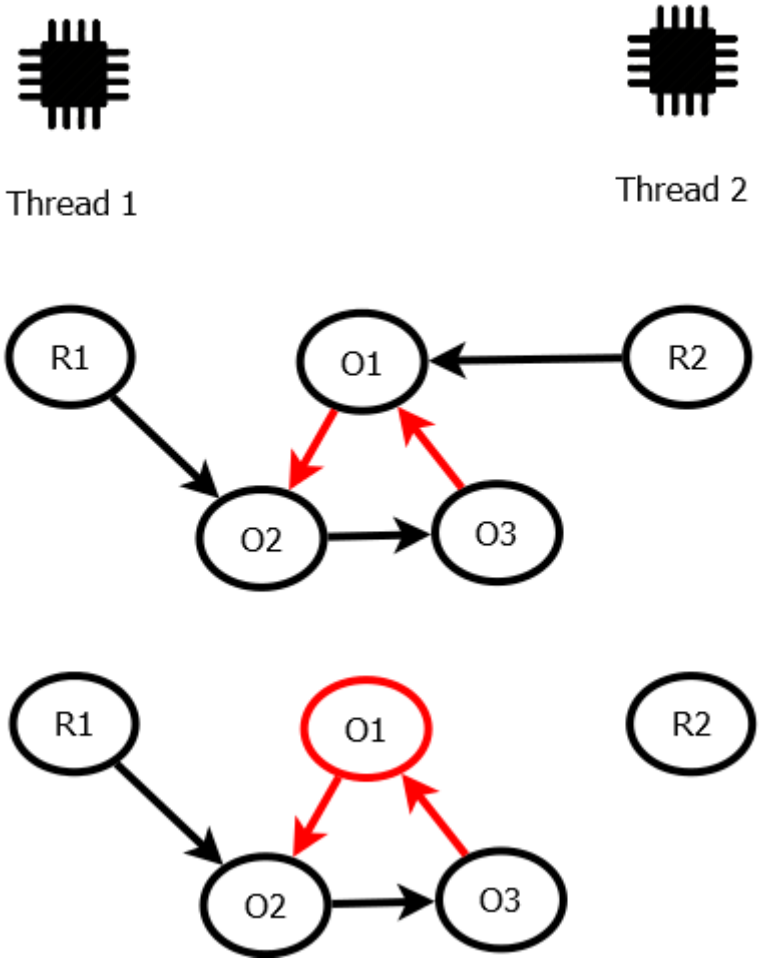
Memory Leak



Memory Leak



Dangling Pointer



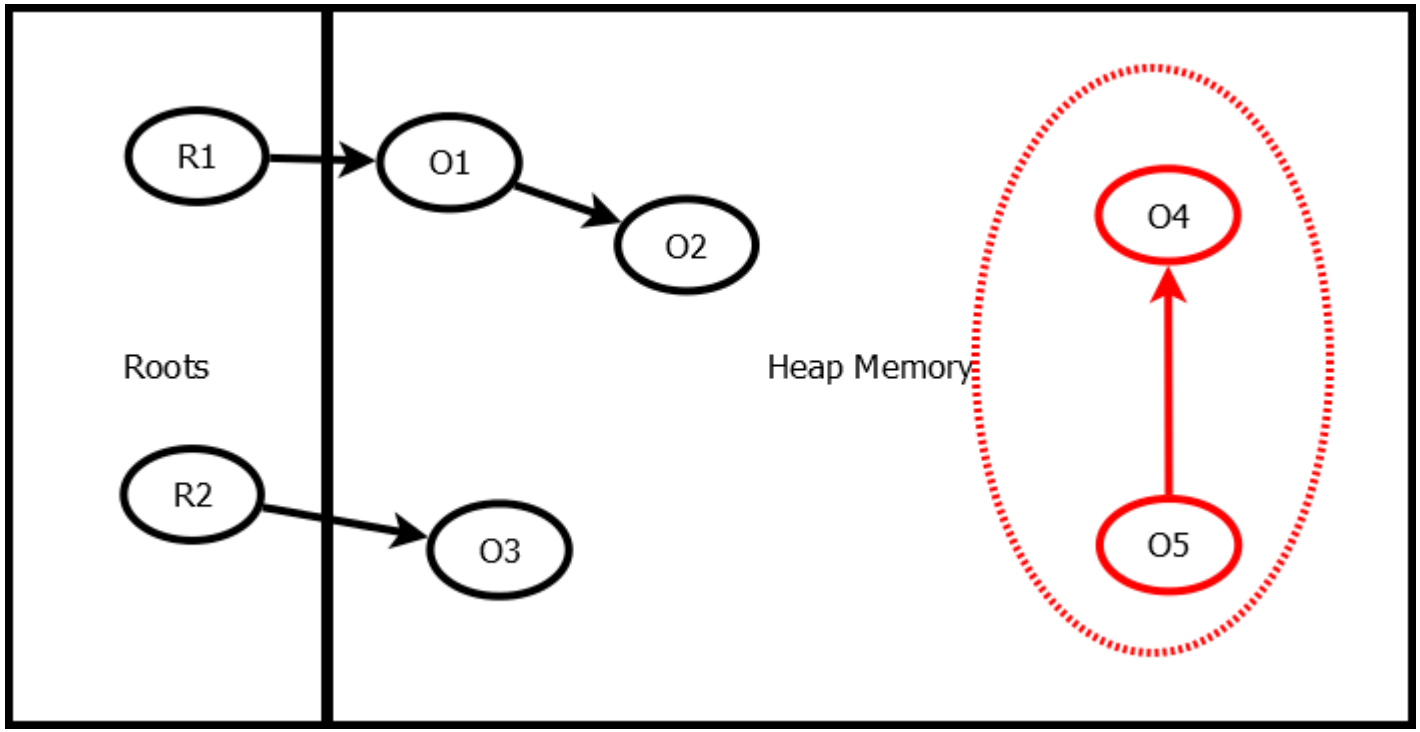
Advantages of GC

- 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]
- Distributed object stores, parallel and distributed programming languages, distributed databases, and WWW.

Mark-Sweep

- When memory 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.

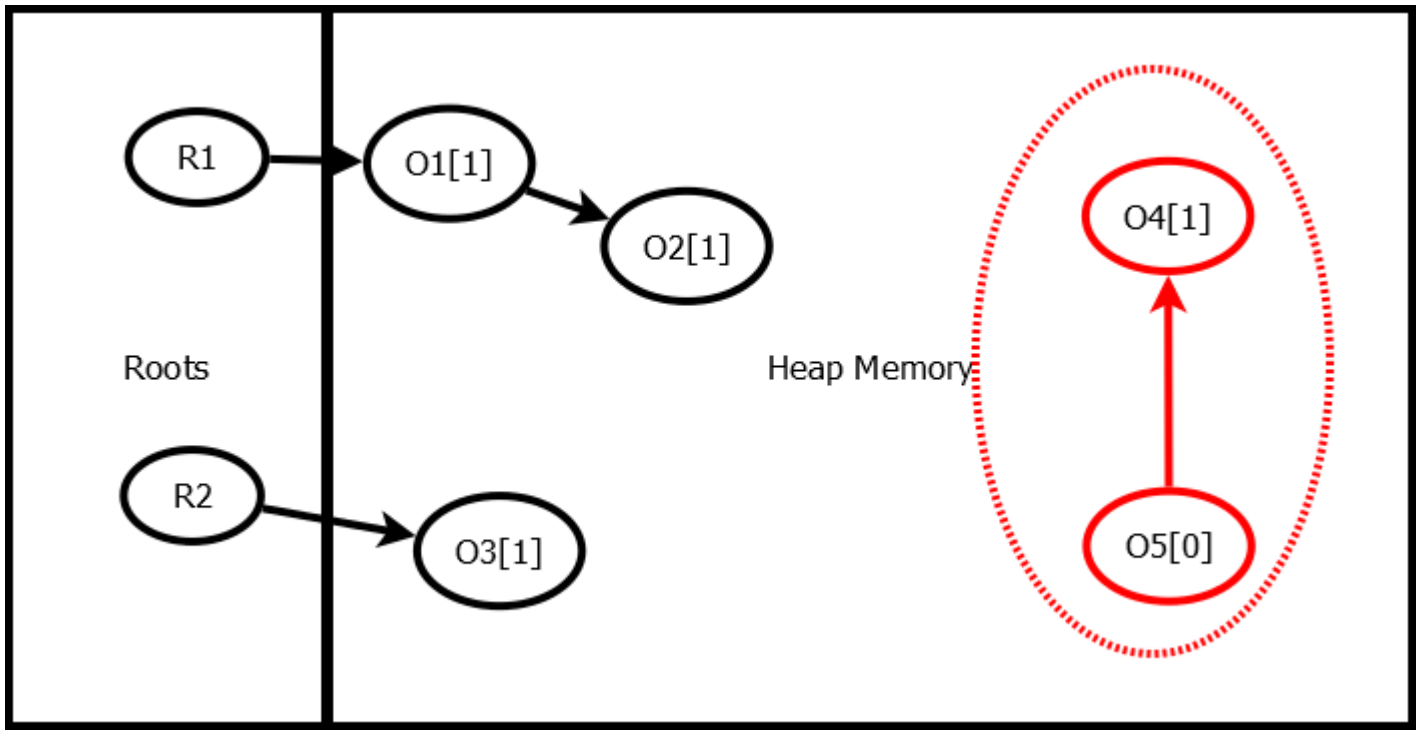
Mark-Sweep



Reference Counting (RC)

- Each object has RC that denotes how many incoming references a node has.
- When an object has zero RC, it is garbage.
- 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 garbage objects.

Reference Counting



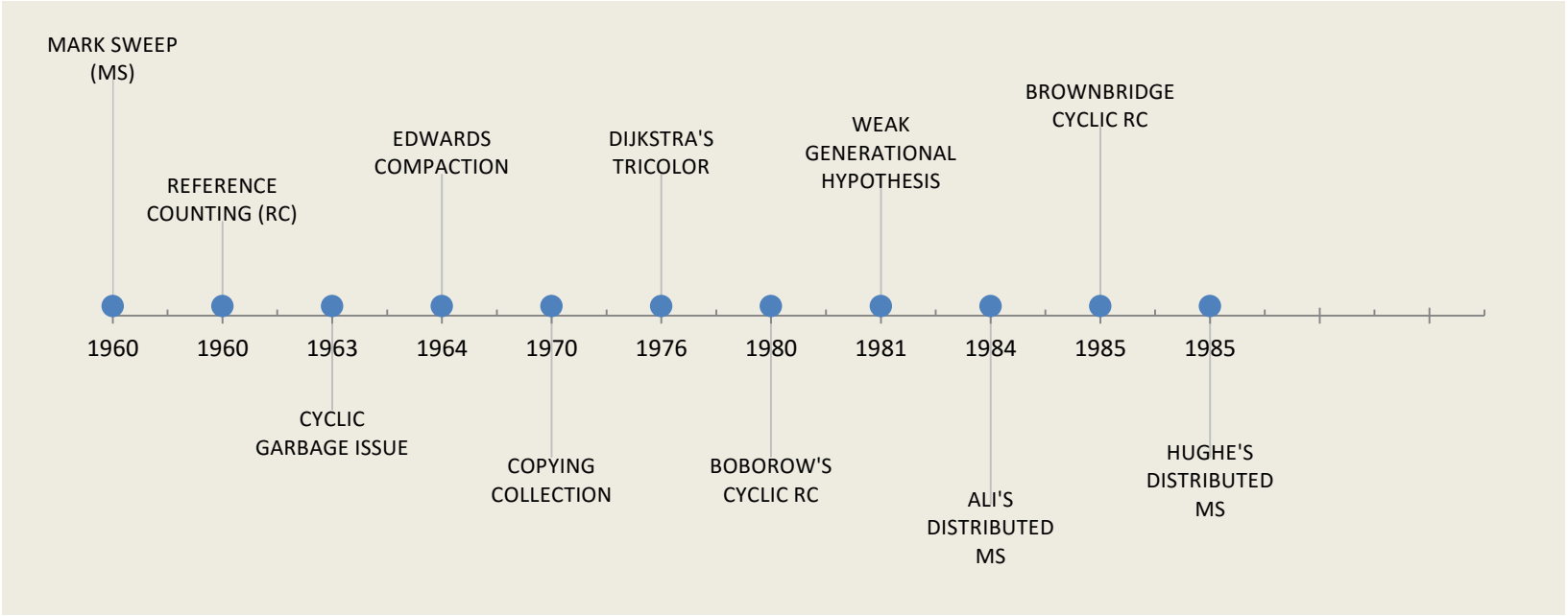
Proposed Hypothesis

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

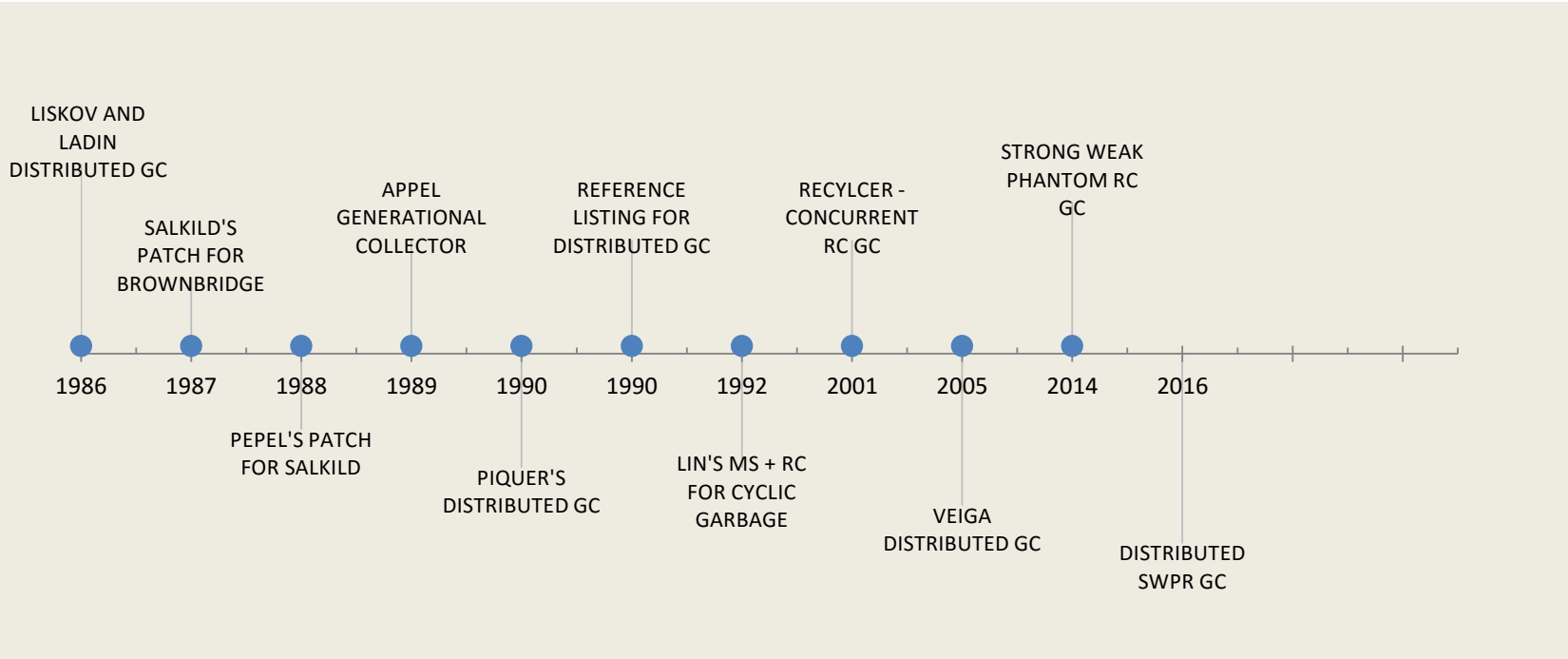
Literature Review

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



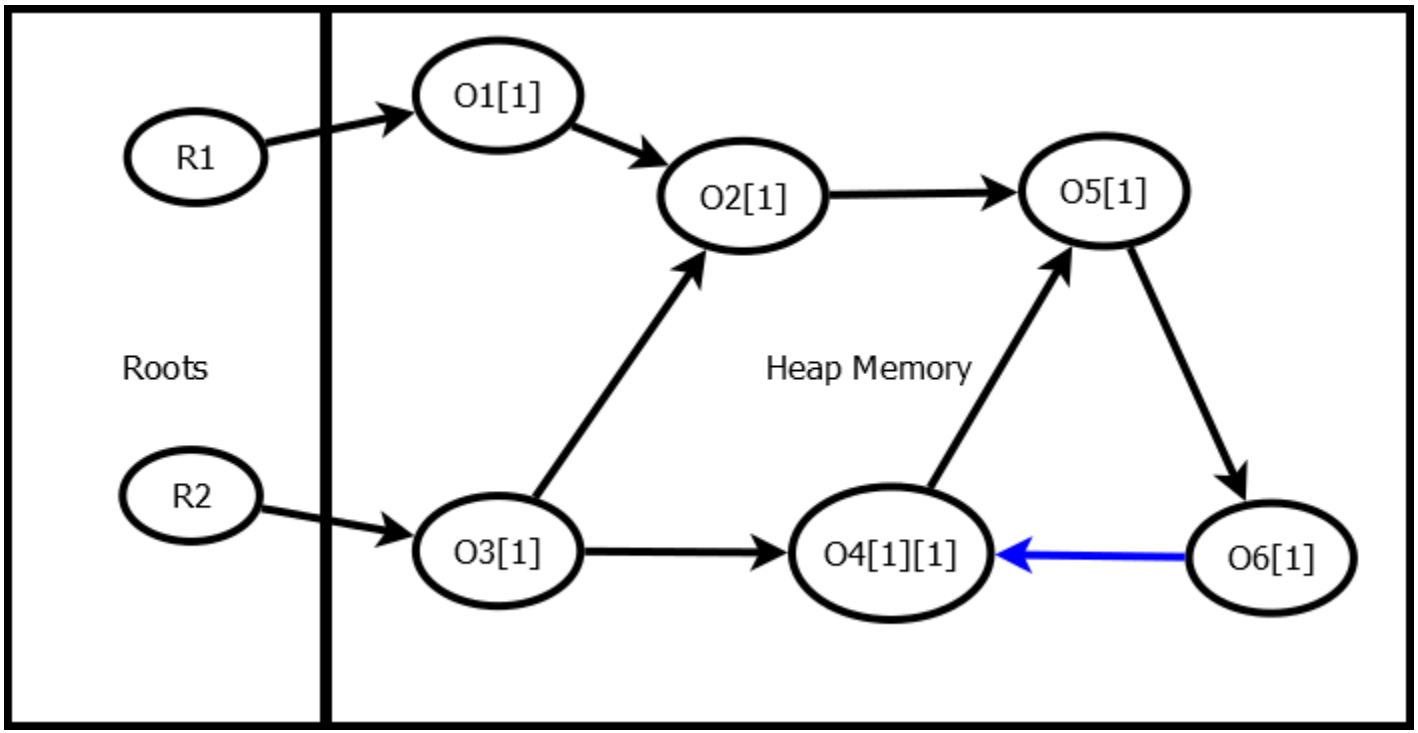
Literature Review



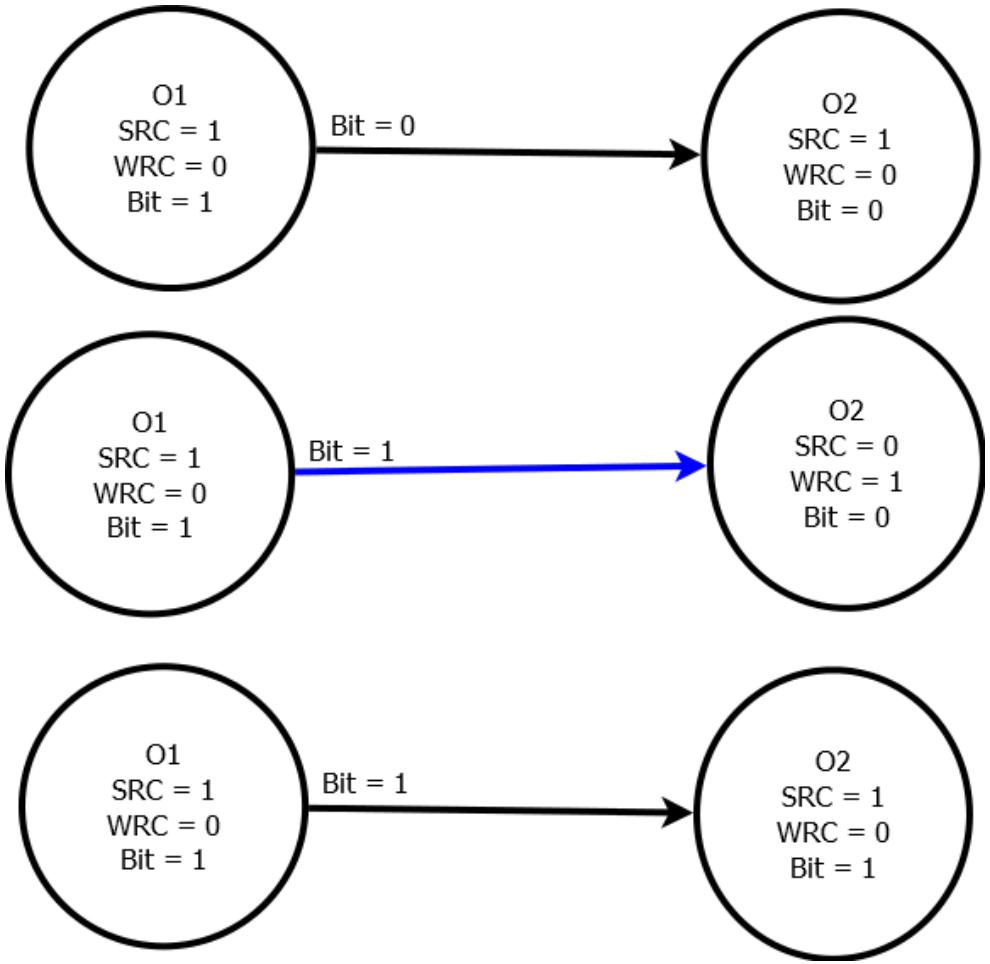
Brownbridge Method

- RC and tracing based technique to collect cyclic garbage.
- It uses two reference counts 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.
- Strong Cycle Invariant.
- When a reference is created to a node, if the target node already has outgoing references, it is considered a weak reference.

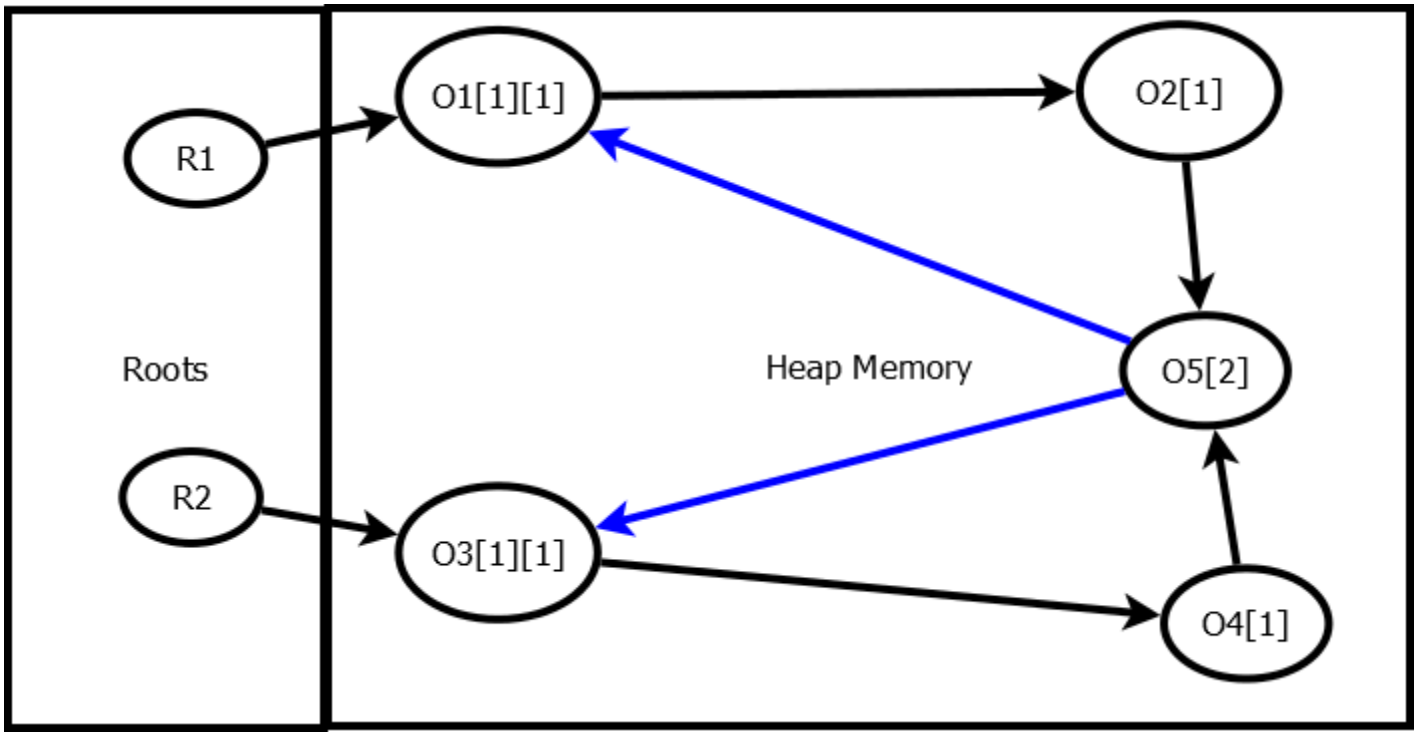
Brownbridge Method



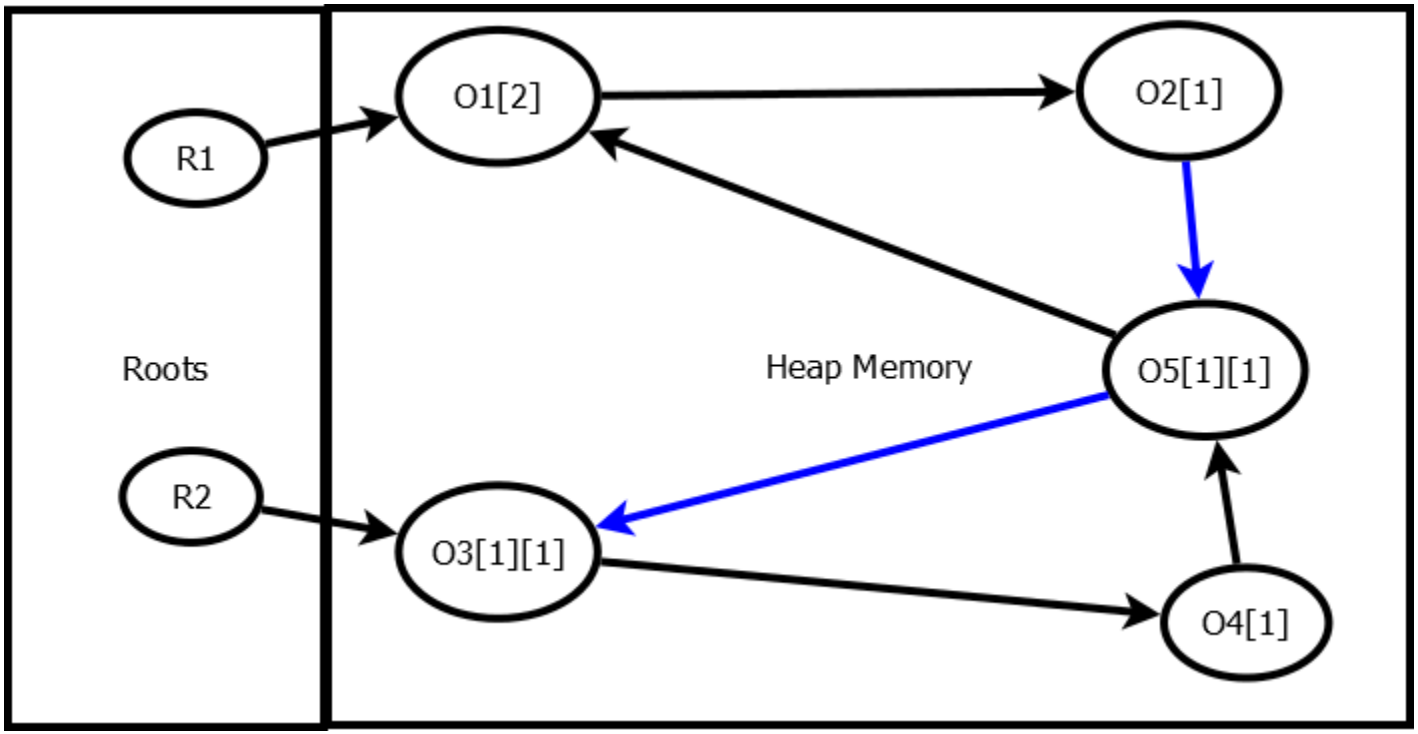
Brownbridge Toggle

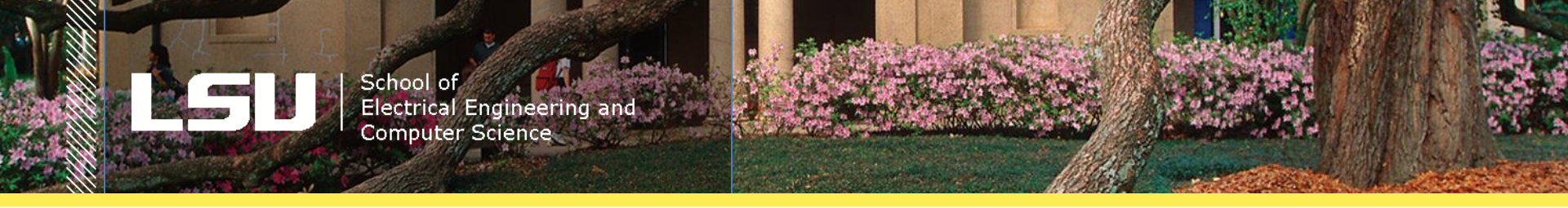


Delete Case



Premature Delete Case





Salkild's and Pepel's Work

- Salkild eliminated the premature deletion but introduced non-termination in certain cases.
- Pepel improved the Salkild’s 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 color algorithm to design a concurrent garbage collection.
- This method uses reference counting along with tracing to identify cyclic garbage collection and runs concurrently.
- It traces entire connected graph and cannot be made to run parallel.
- Antony Hosking et al claims that the 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 reference and avoid creating cycle of strong reference by using weak reference at the potential cycle creation event.
- **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.

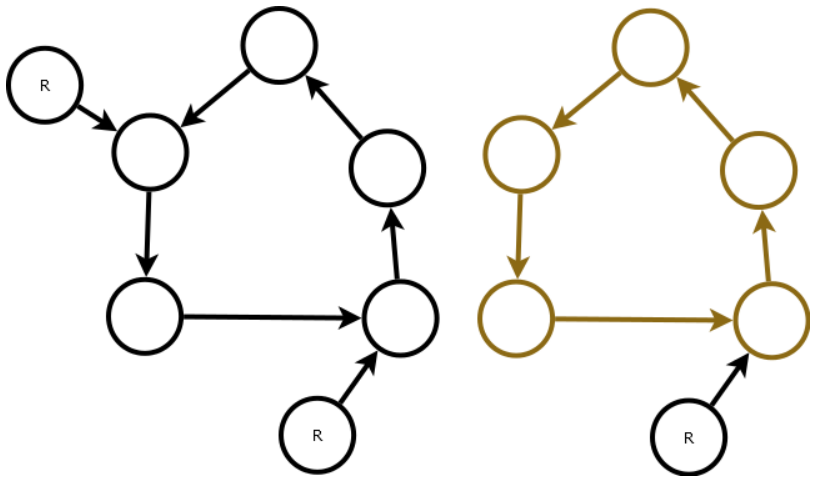
SWP

- Graph of strong edges form a connected, directed acyclic graph.
- Strong edge represents liveness.

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

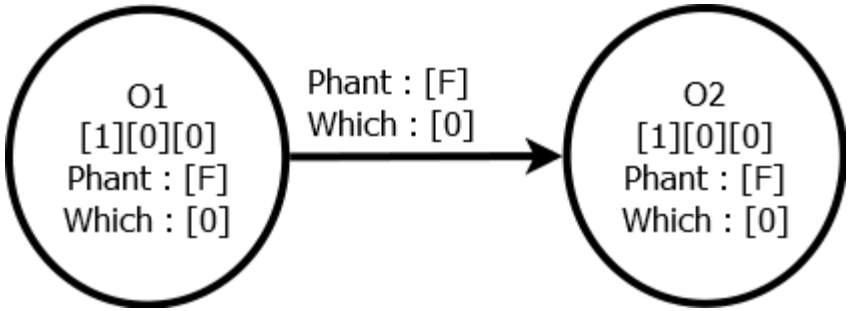
SWP

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 which bit, phantomized flag, and array of which bits for outgoing references.
- Process list and request queues are used.



Delete Edge

When a node lose 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 phantomized outgoing edges

Three phases

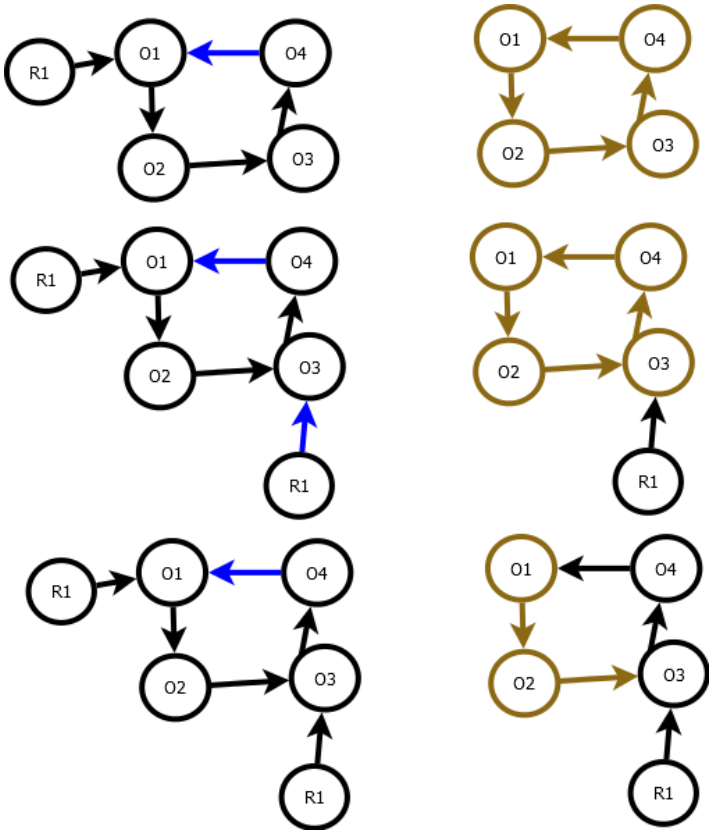
- Phantomize
- Recovery
- Cleanup

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

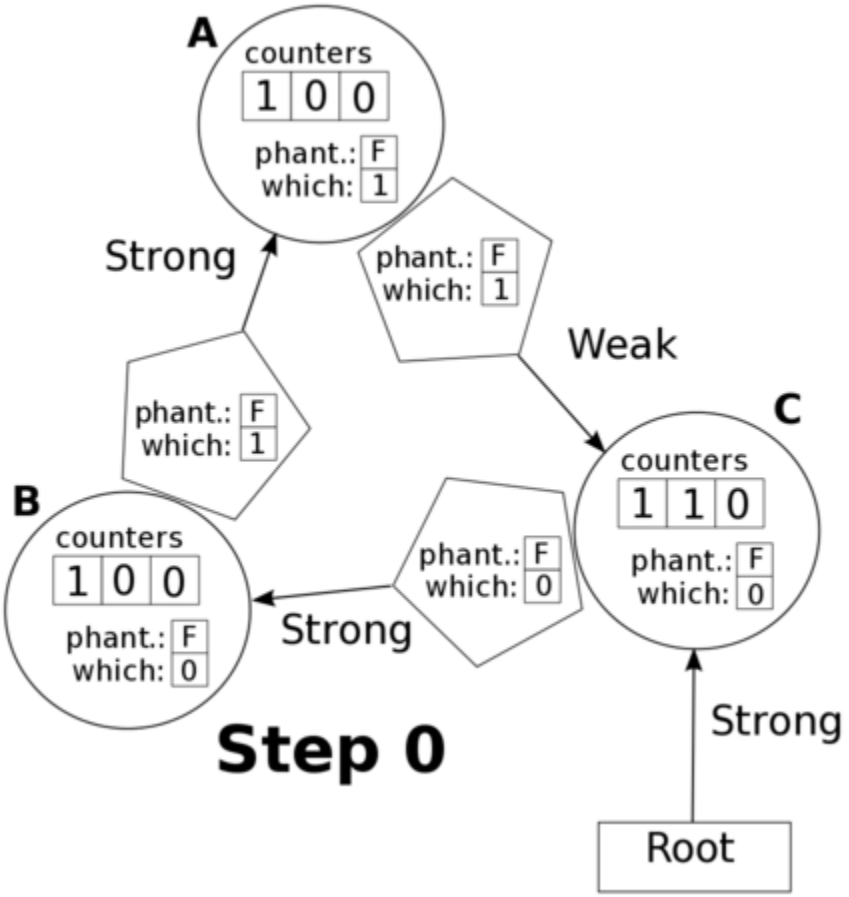
Phantomization

- Decrement all internal references.
- Phantomization stop when it reaches a phantomized node, live node.
- It performs toggling if any weak reference appear and has to propagate the phantomization.

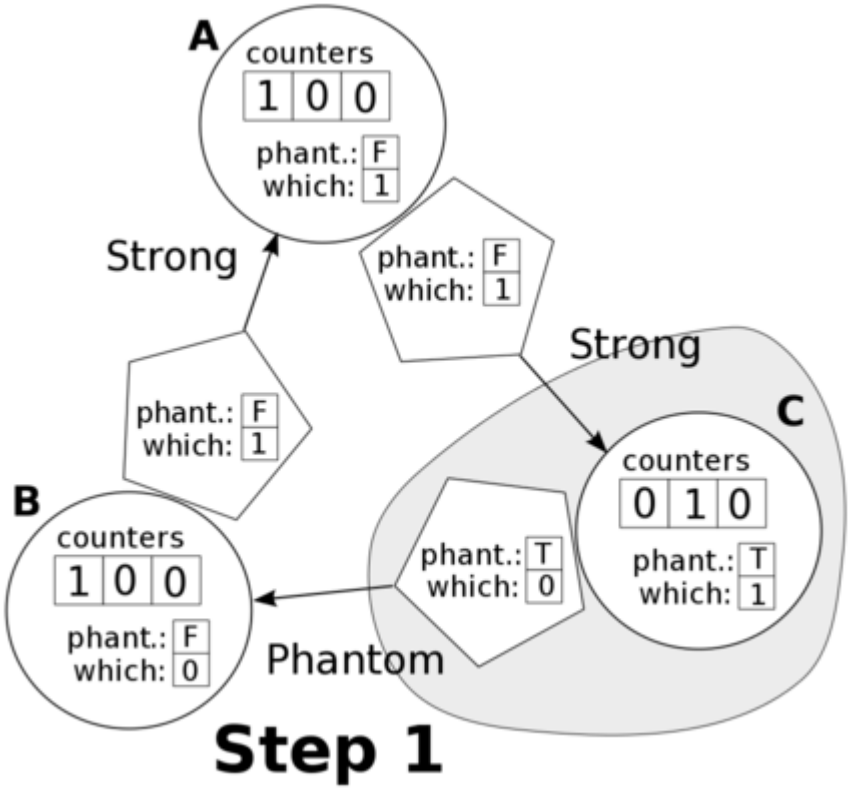
Phantomization



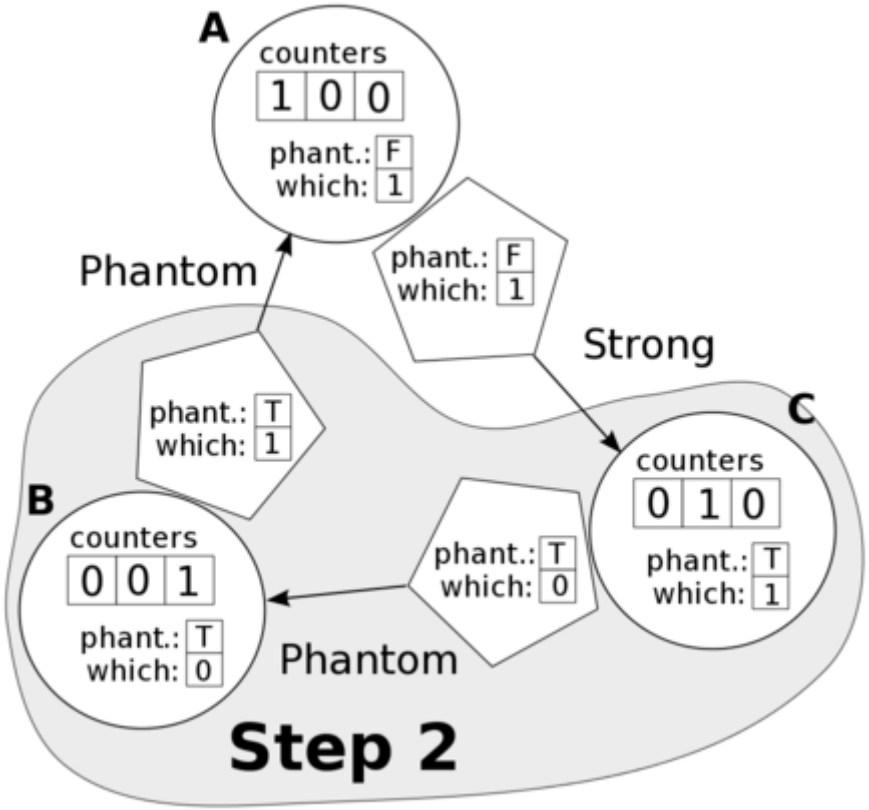
Simple Cycle Demo



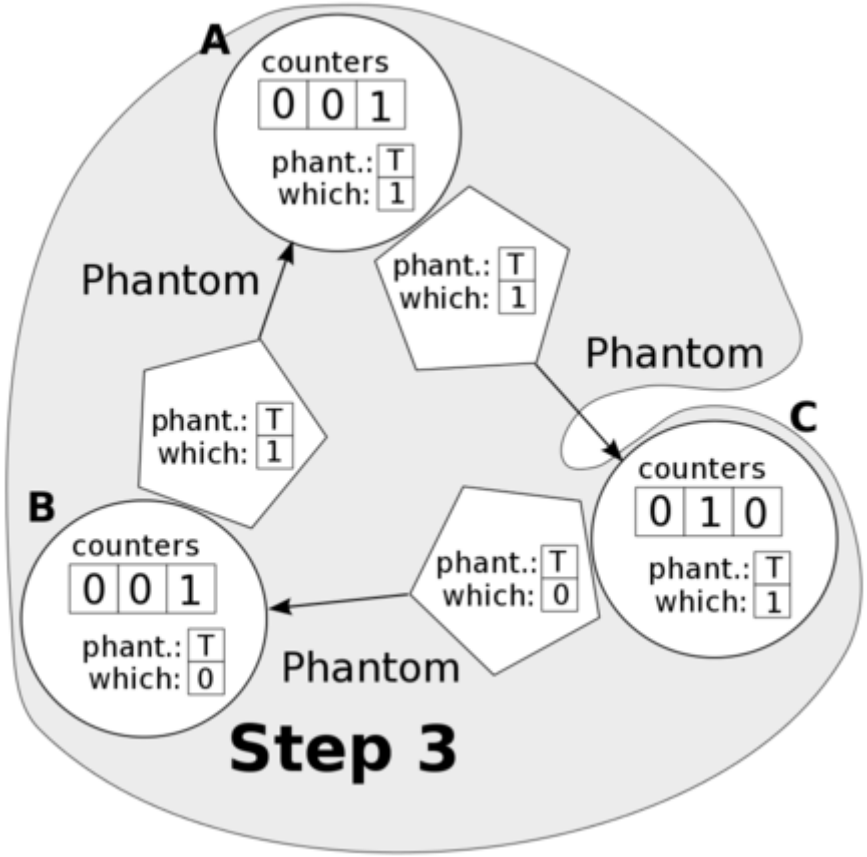
Simple Cycle Demo



Simple Cycle Demo



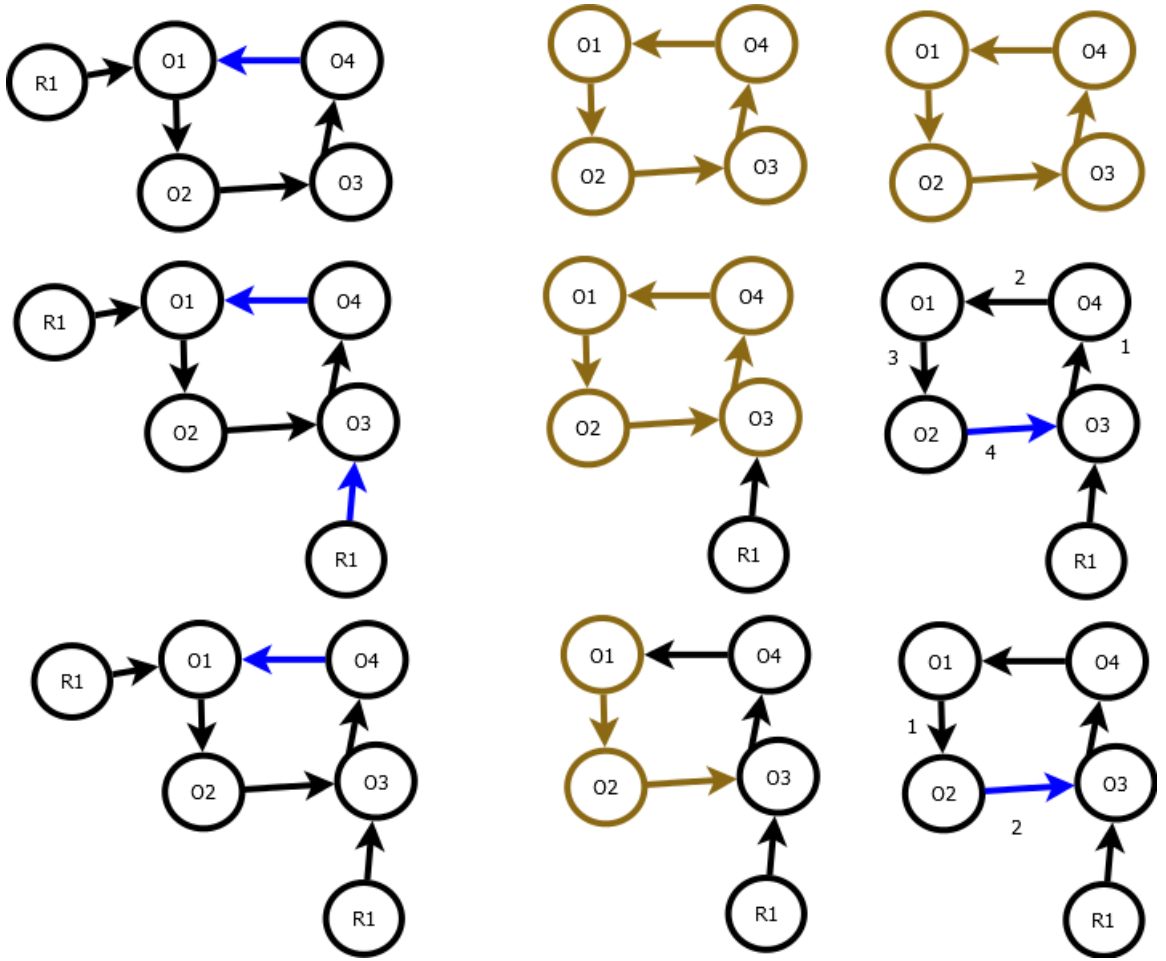
Simple Cycle Demo



Recovery

- If any external reference found, correct the graph to strong / weak graph
- Delete nodes from process list as they are recovered.

Recovery



Concurrent and parallel Collector

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

Concurrent and parallel Collector

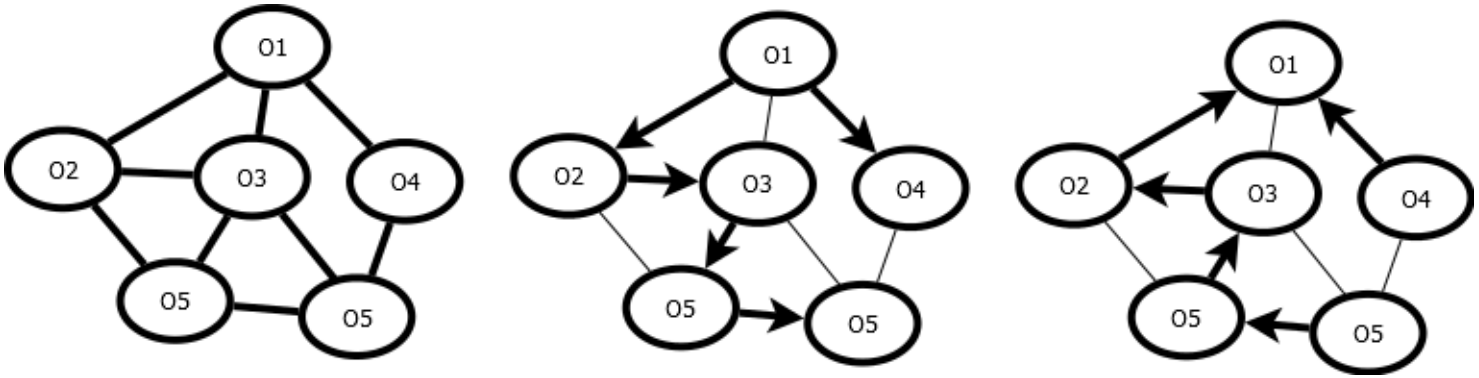
- **No Partition Principle** : When two or more collector synchronizes, it merges their list of processed nodes and one of the collector takes the processing of those nodes.
- Other collectors start to process the request in the queue.
- Parallel collector adds one more attribute to the object header.

How SWP can be used?

- SWP algorithm is directly applicable to shared memory systems.
- SWP algorithm is also applicable to the distributed system with centralized queue.
- Unlike other distributed systems which require application to be halted, this does not require it.
- This collector can function concurrently as it is now, but requires centralized queues.
- Mobile actor based collector does not require centralized queue and makes it directly applicable to the distributed systems with longer 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.

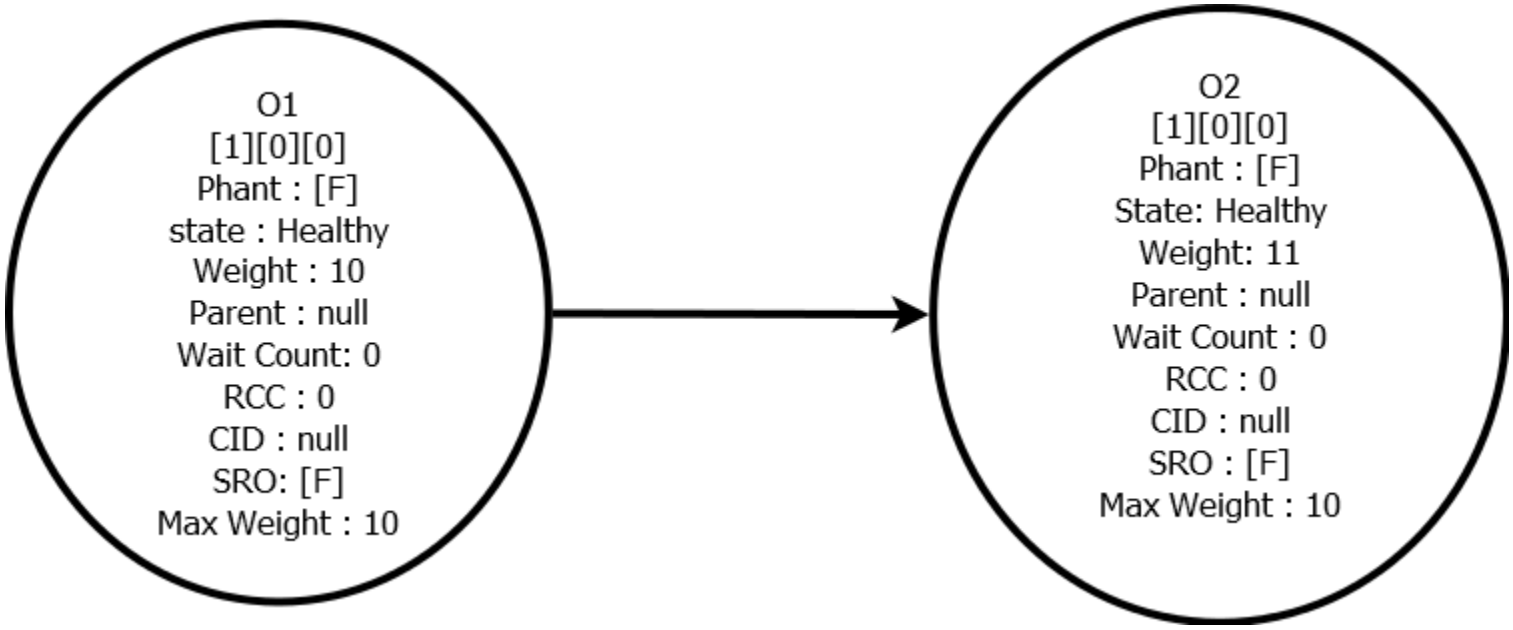


Model

- 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.

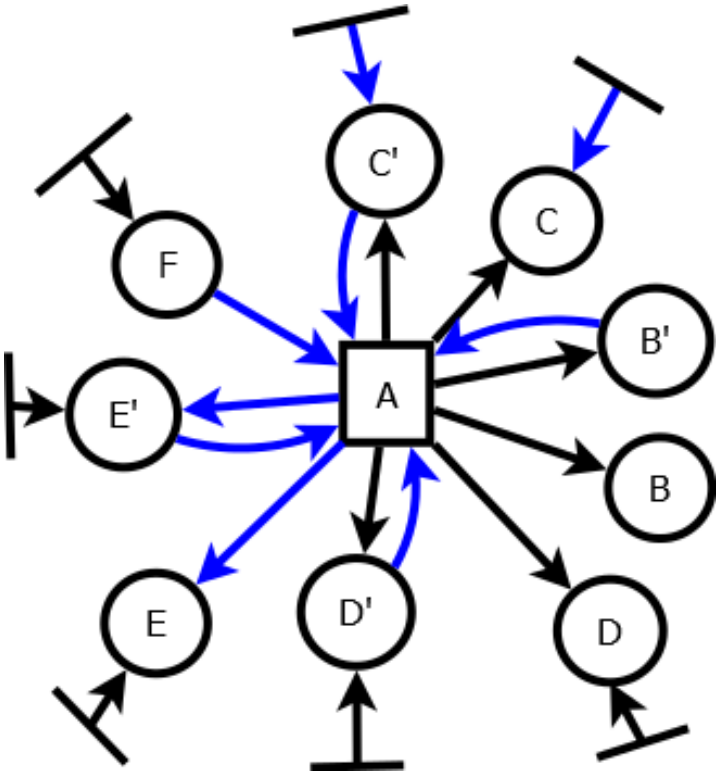
Weight System

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

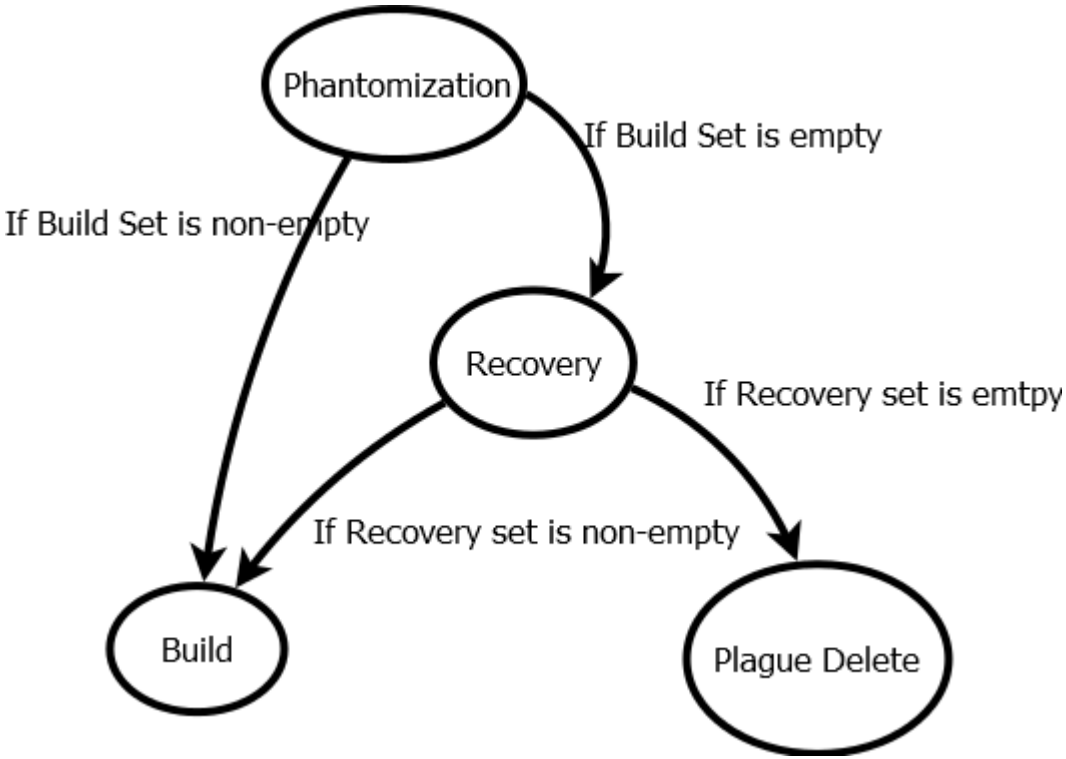


Node Classification

- 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



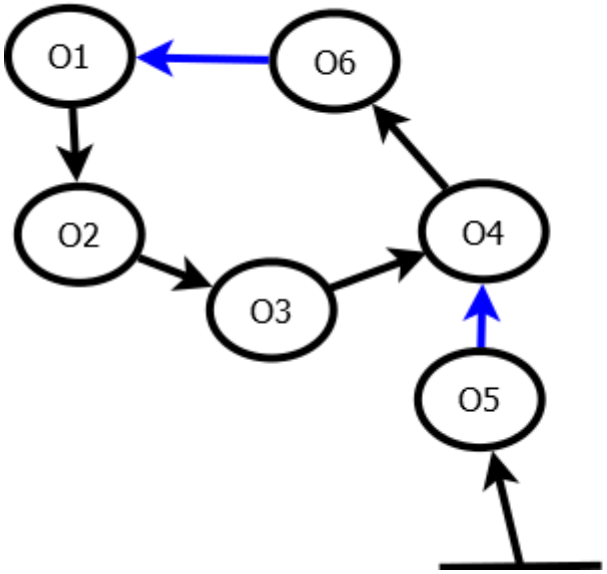
State Diagram

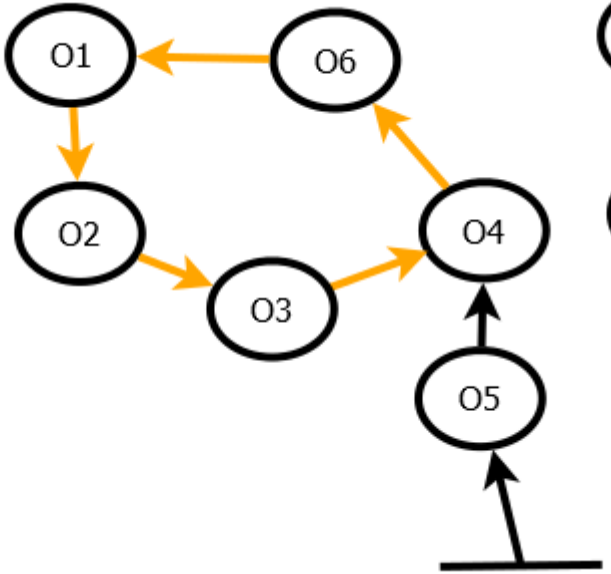


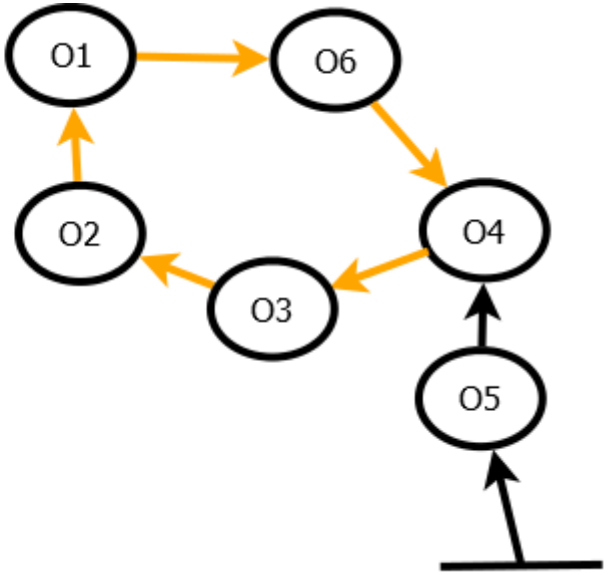
Phantomization

- Converts all the internal edges that are reachable to non-independent nodes to phantom.
- The process is to make sure all the internal edges are not counted for the decision process.
- External weak edges are converted into strong during this process.
- The process uses forward phase and backward phase to finish operations.

Phantomization



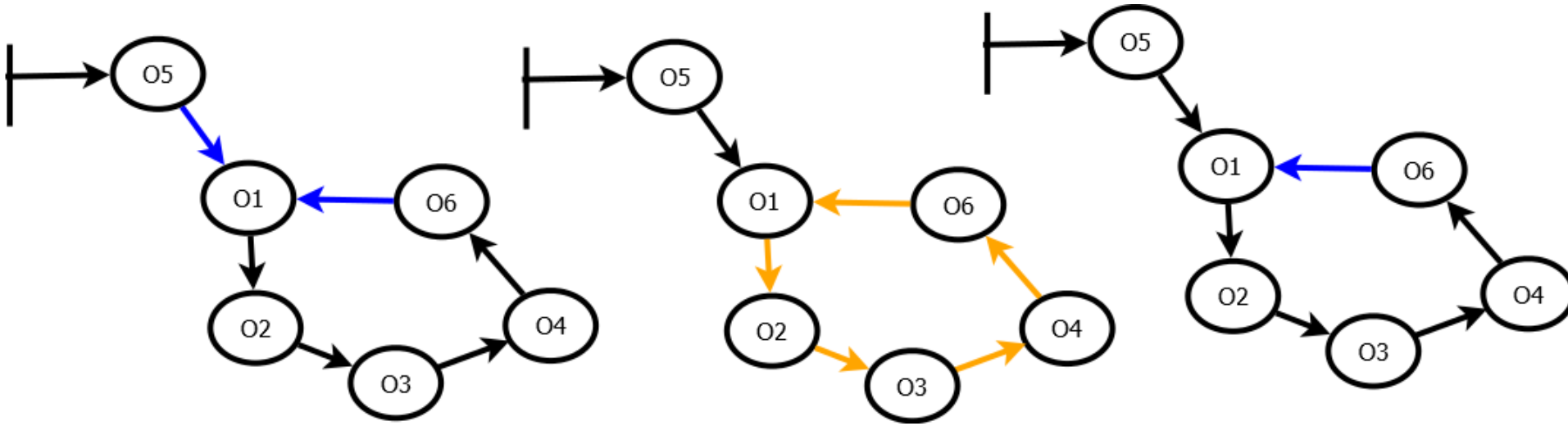




Correction

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

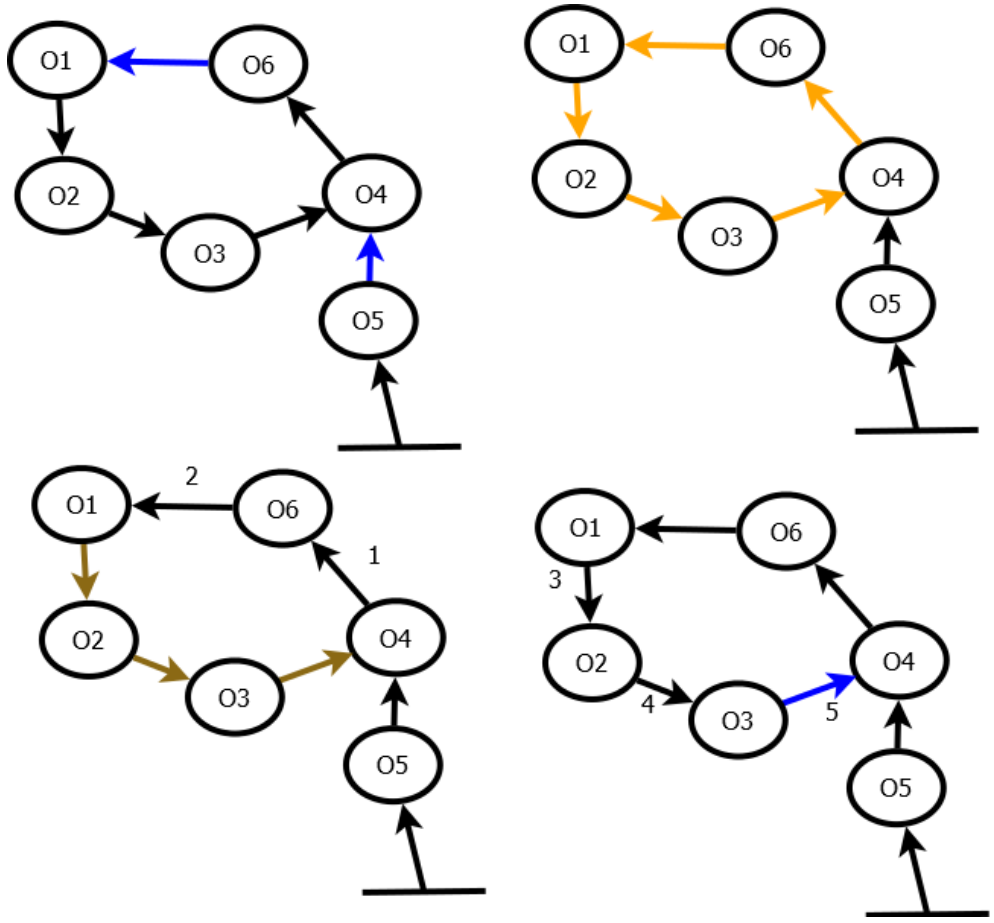
Build Phase



Recovery

- Recovery is a complicated phase.
- If initiator does not have any build set, recovery messages are sent.
- A recovery node message will start building the subgraph that is not yet processed.
- On reverse phase, a recovery node can start building too.
- Recovery node also reduces the size of the affected subgraph during processing.

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

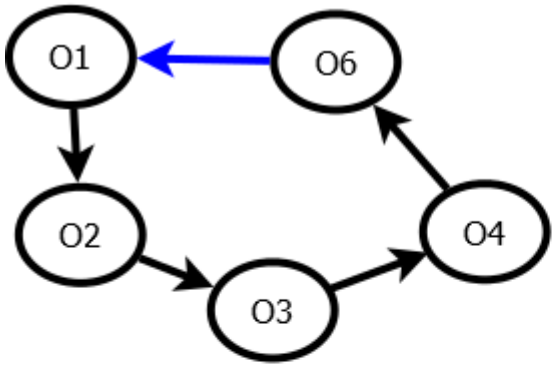
Plague Delete

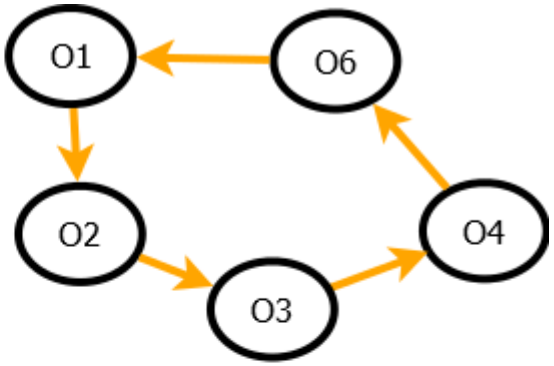
- A node will be deleted only if all the counts are zero.
- A node on receiving plague delete will send plague delete only if there is no strong incoming edge.
- Plague Delete is invoked by initiator if there is no build set and no recovery set.

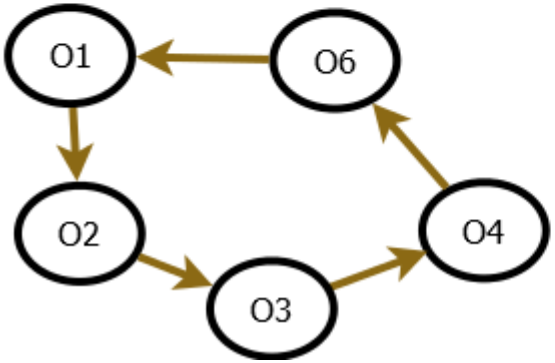
Plague Delete

- 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







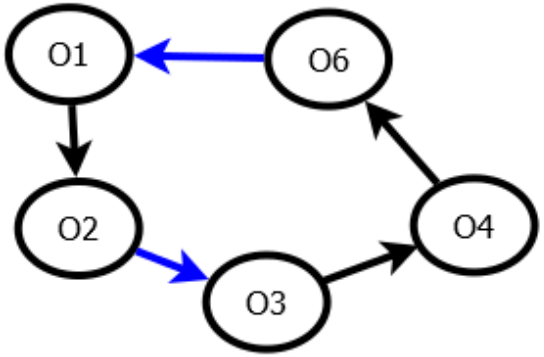
Isolation property

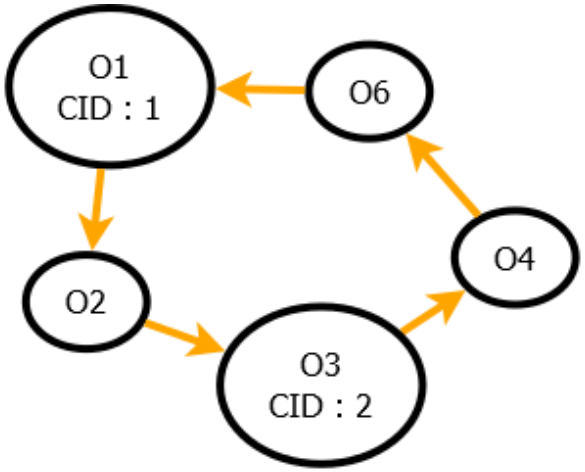
- The affected subgraph is not mutated.
- Mutation to the affected subgraph occurs, but do no affect the decisions of the initiator to delete or build.
- Do not affect the decision of recovery set to build by the correction phases.

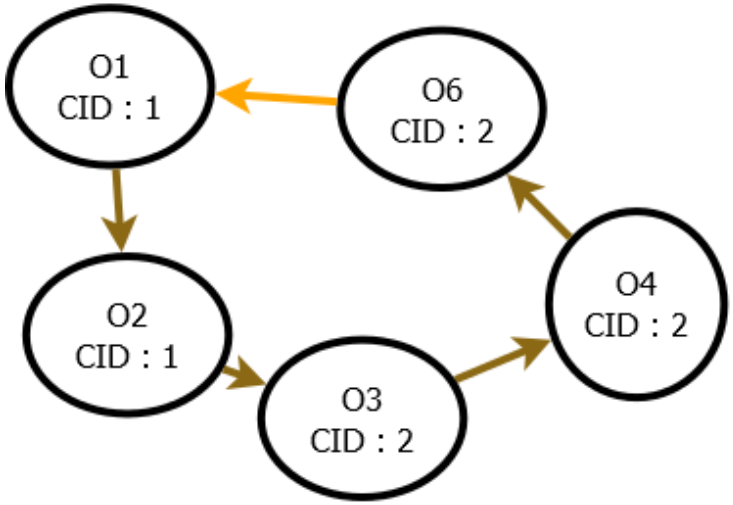
Symmetry Breaking

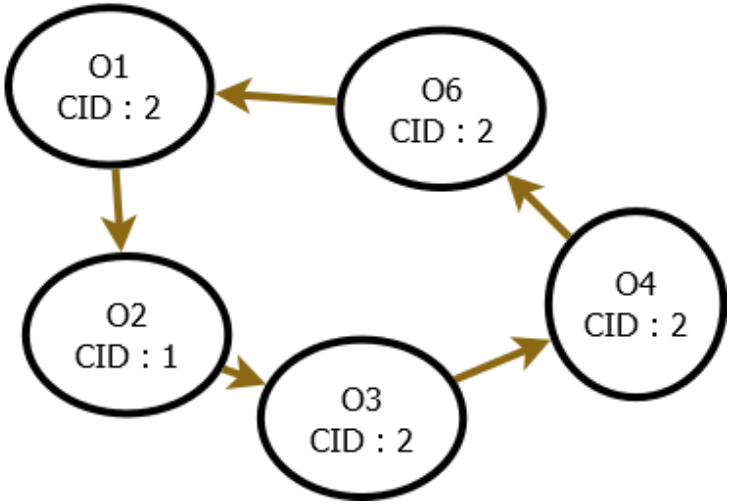
- When multiple collector processes the same subgraph, there is possibility of cycle dependency.
- Leader needs to be elected among conflicting collector operations.
- Each node contains a collector id in the correction phase.
- Phantomization phase is similar to marking where the nodes does not need to know which collection process it belongs to.
- During correction phases, a node get the collector id it belongs to.
- All nodes prefer to be in higher collection id.
- All collection ids are unique and so there is a total ordering among collections.

Acyclic Principle (Isolation)





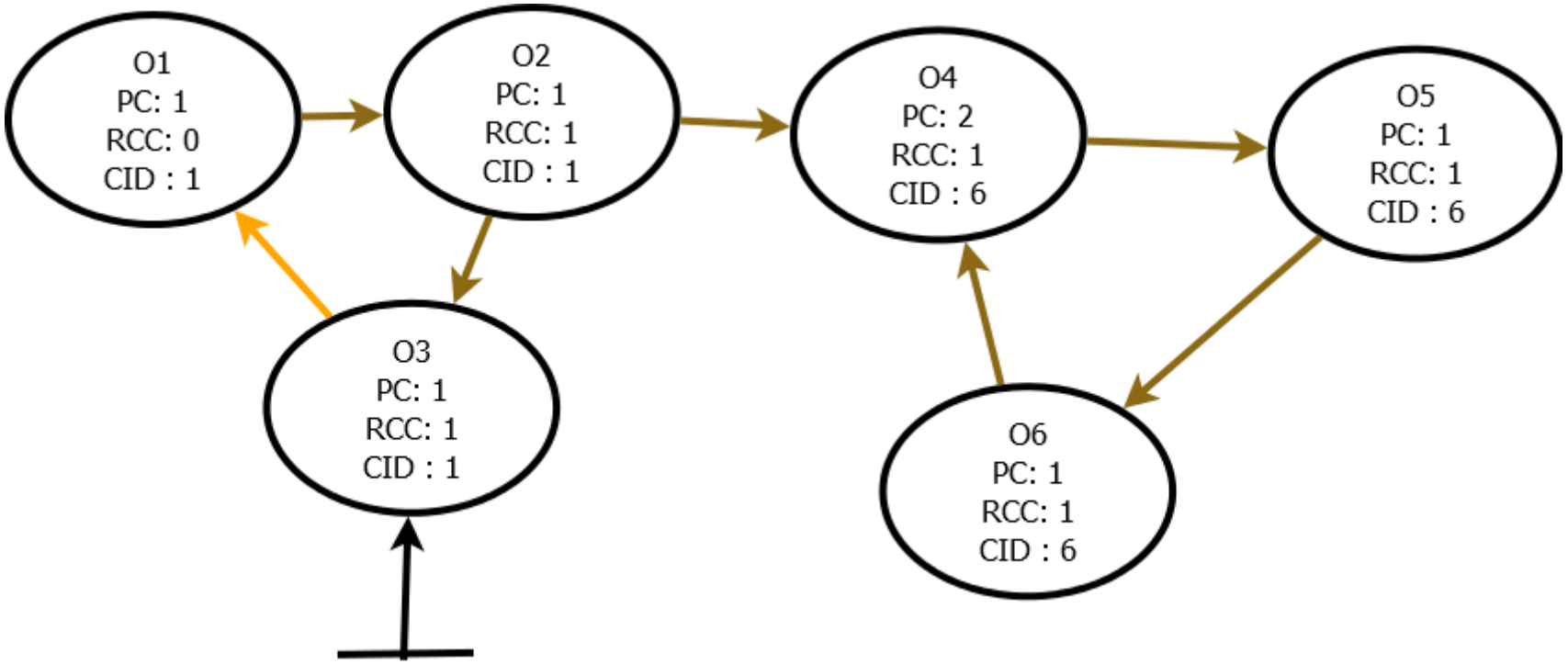




Recovery Count

- For recovery 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.

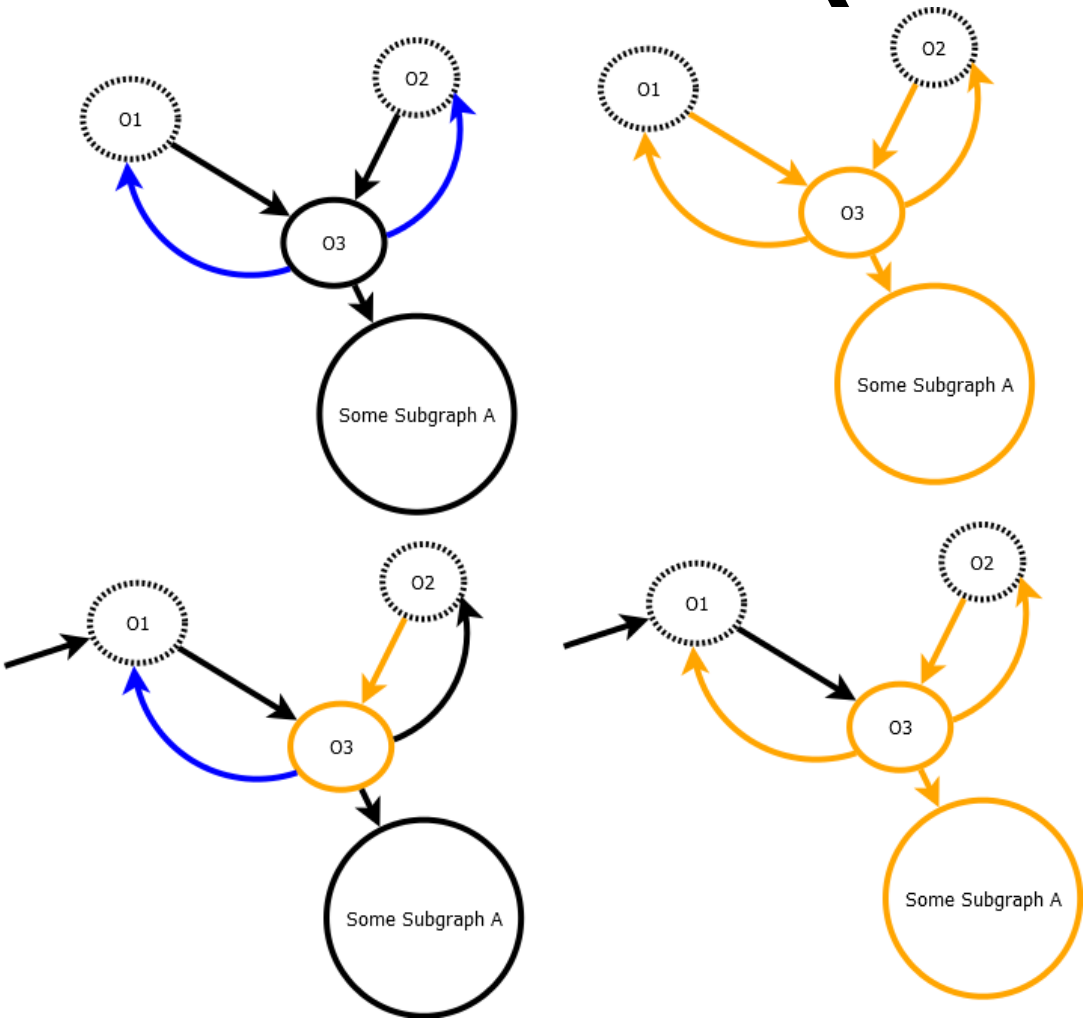
Topology Ordering (Isolation)



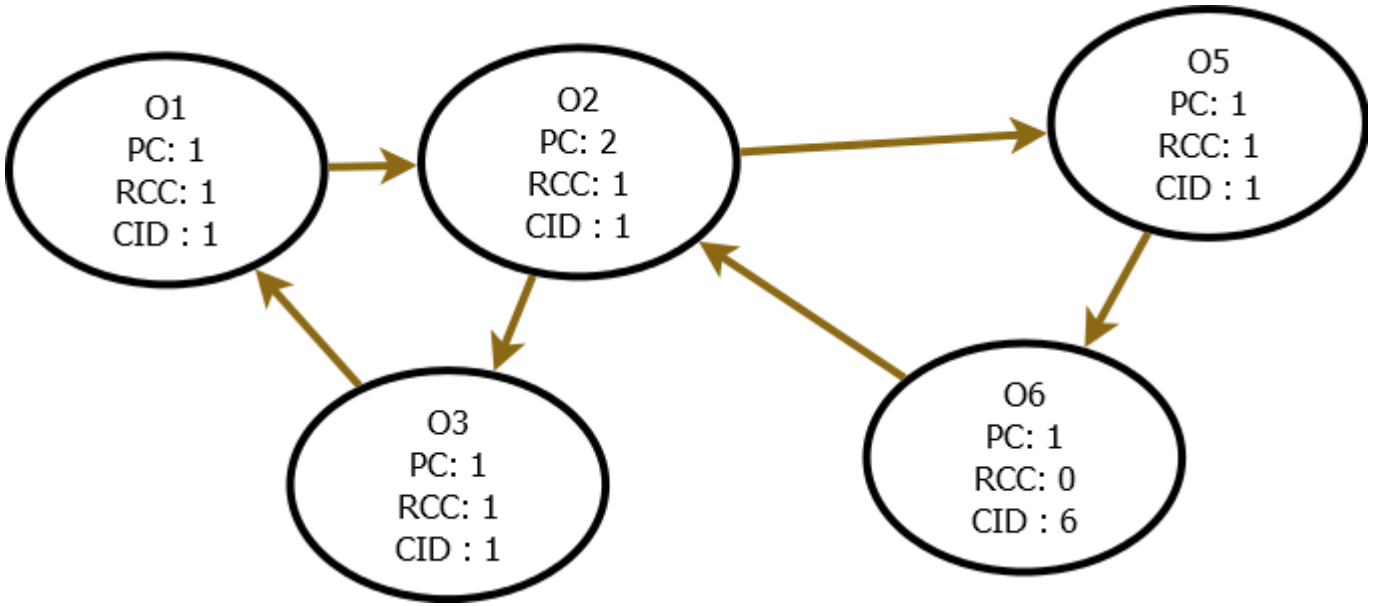
Transactional Approach

- When two collectors' operation meet at a point, the lower collection id will not proceed through 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 collector colliding.
- When multiple collectors of different phase meet, we wait until the reverse phase gets over 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.