Local/Distributed Garbage Collection in Renaissance

Memory Management (MM)

- Explicit
  - Use free() or delete()
- Implicit
  - Use Garbage Collection

Explicit Memory Management

Advantages:

* Space Efficient
* Incremental

Disadvantages:

* Prone to errors
* Memory Leaks

Explicit Memory Management

Disadvantages (cont.):

* Need to keep track of object liveness
* Too much copying
* Very time consuming
* Exceptions are difficult to implement

Implicit Memory Management

Advantages:

- **No need to worry about when to free objects**
- **Complex data structures can be passed as parameters**
- **Simplifies programming (Lisp, Prolog, and ML have built-in GC)**
- **Better parallelism between program and memory manager**

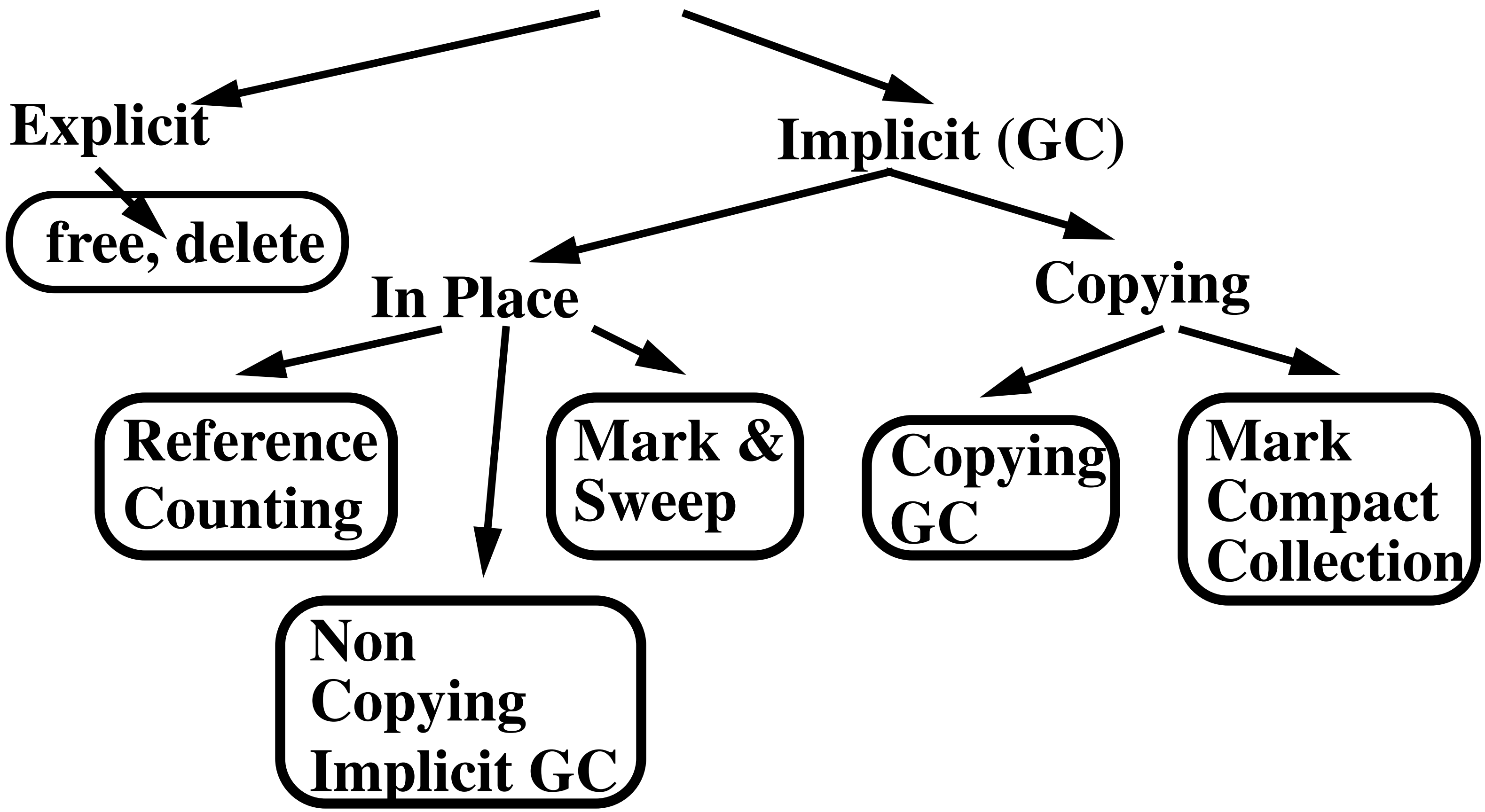
Implicit Memory Management

Disadvantages:

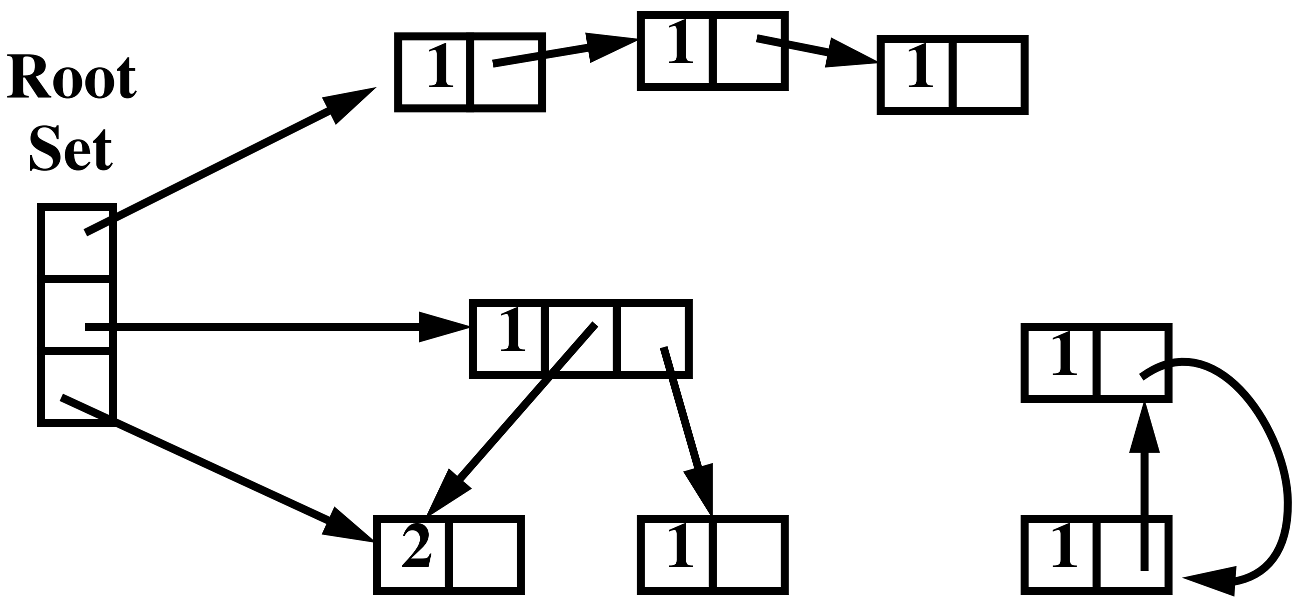
* Large Pause Times
  Use Incremental Techniques

* Overhead
  Use Generational Techniques

Memory Management Classification [Wilson 92]

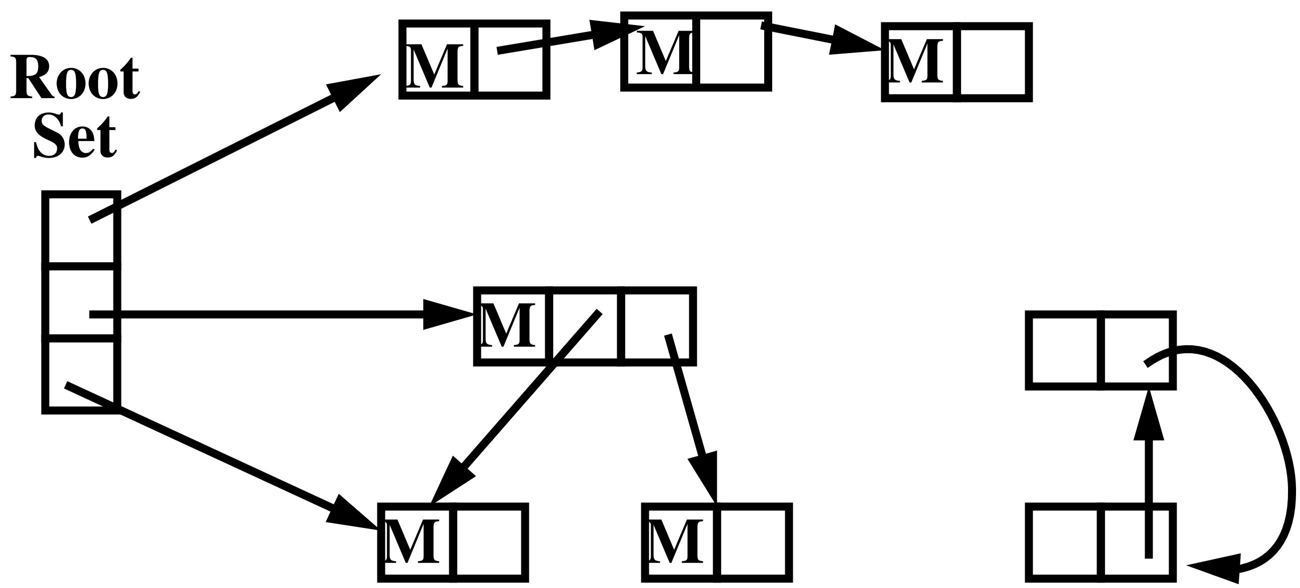


Reference Counting [Collins 60]



- Increment/decrement references in each pointer assignment

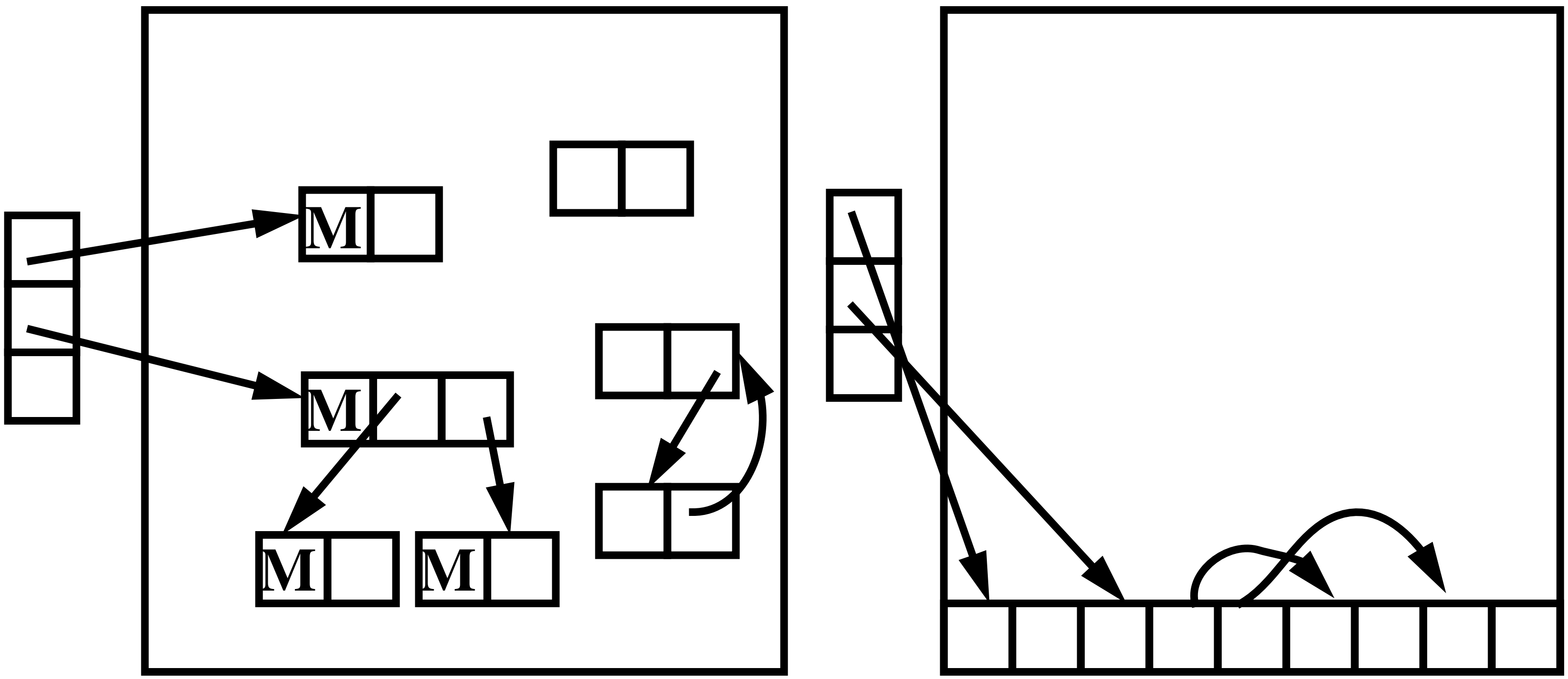
Mark & Sweep GC [McCarthy 60]



-Mark all reachable objects

-Return unmarked objects to free list

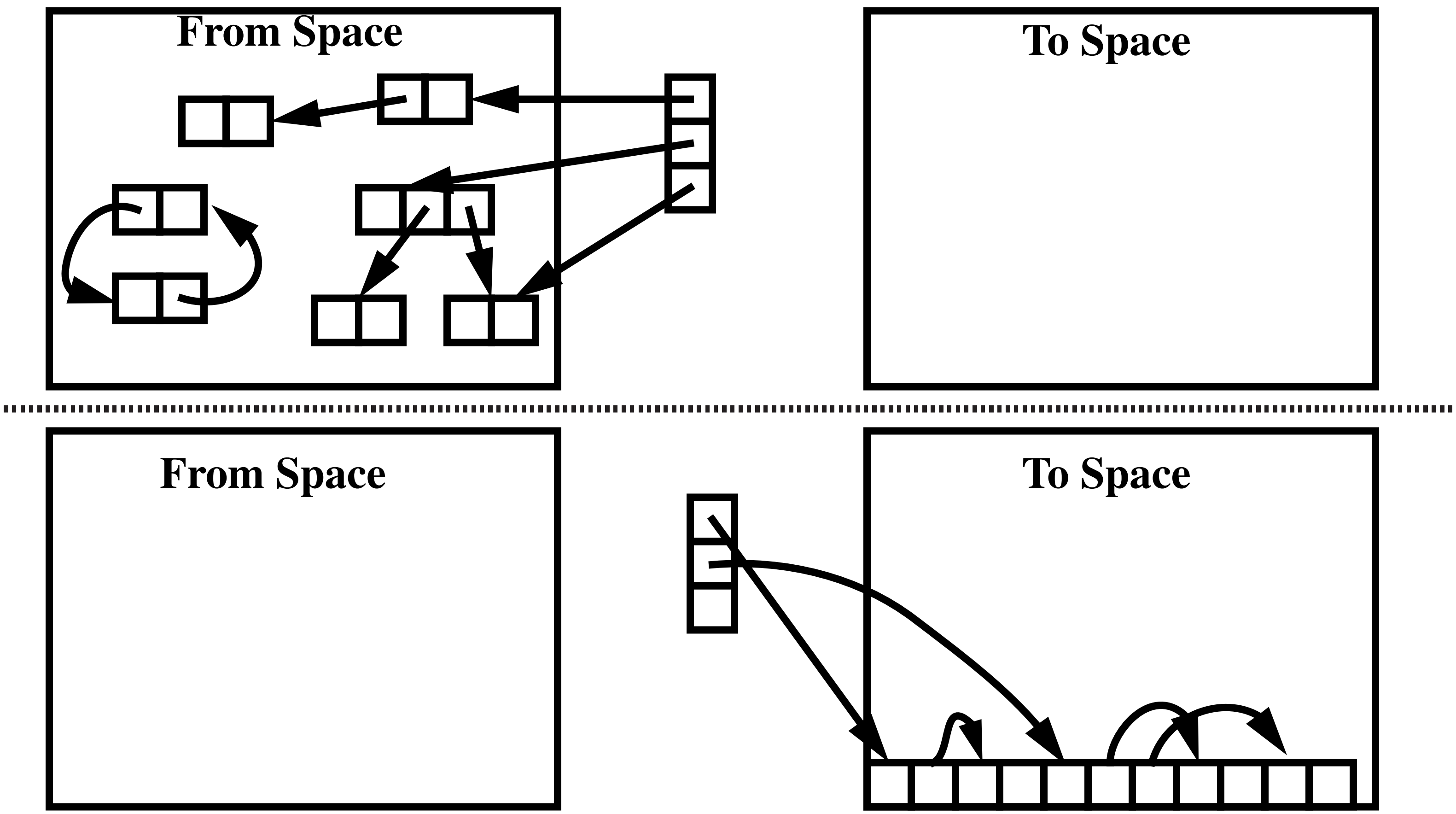
Mark - Compact Collection



-Mark all reachable objects

-Slide up marked objects to fill up empty spaces

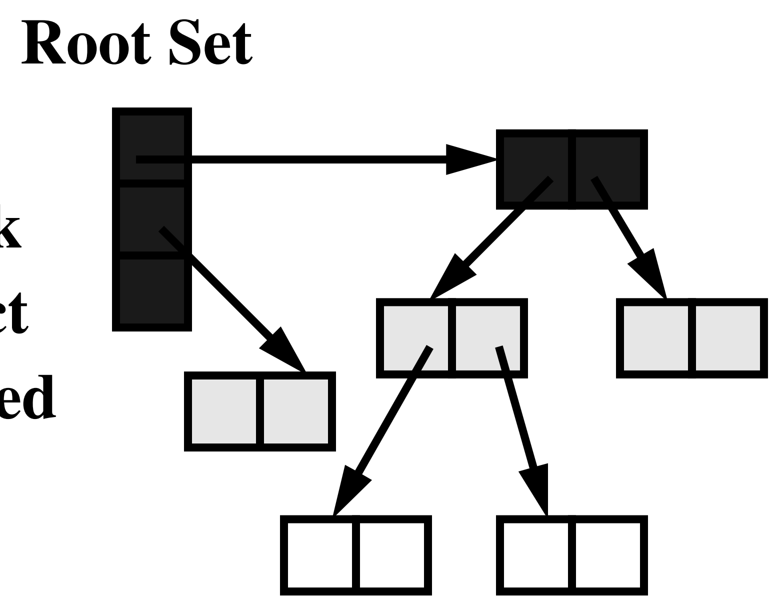
Copying Garbage Collection [Cheney 70]



- Copy live objects from "from space" to "to space"

GC Mark & Sweep, Basic Algorithm

Stop the World
Push roots to stack
While stack not empty
    pop object from stack
    for every ptr in object
        if *ptr is not marked
            mark *ptr
            push *ptr
Restart the world
Return unmarked objects to free-list.



Incremental GC

Mutator: Running Program

Collector: Collects Garbage

Problem:

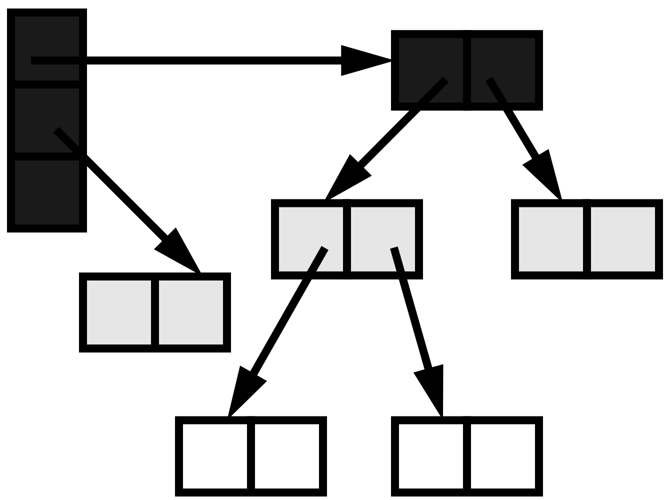
> **How to make Mutator and Collector run in parallel?**

Tricolor Marking [Dijkstra et al 78]

Black: Marked objects already scanned

Grey: Marked objects not yet scanned

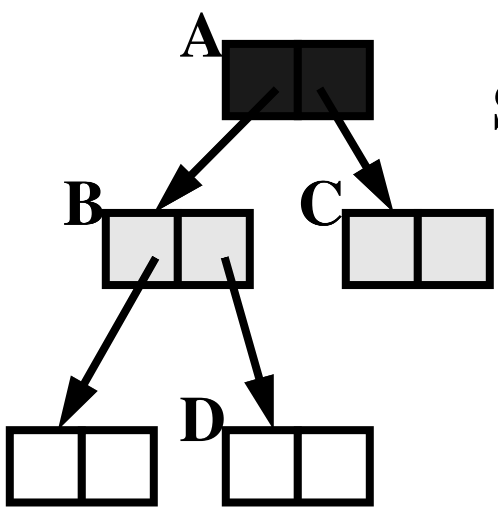
White: Unmarked objects



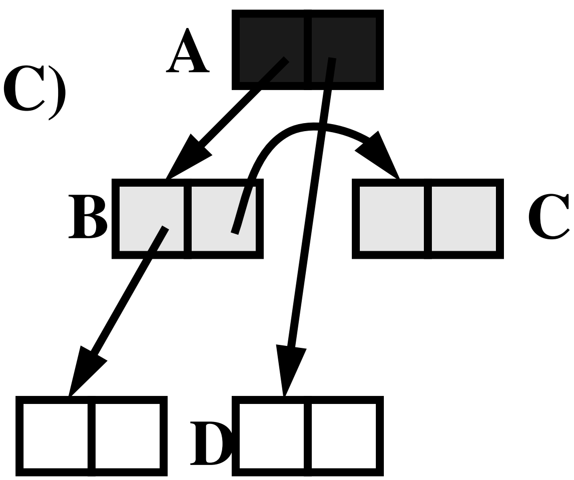
Tricolor Marking (cont.)

Invariant:

No black object should point to a white object directly.



SWAP(B->D,A->C)



It should not happen!!!

How to Preserve Invariant?

Read Barrier:

Whenever mutator tries to <u>read</u> a pointer to a white object:

-Make the white object grey [Baker 78]

-Make the white object black [Juul and Jul 92].

How to preserve the invariant? (cont.)

Write Barrier:

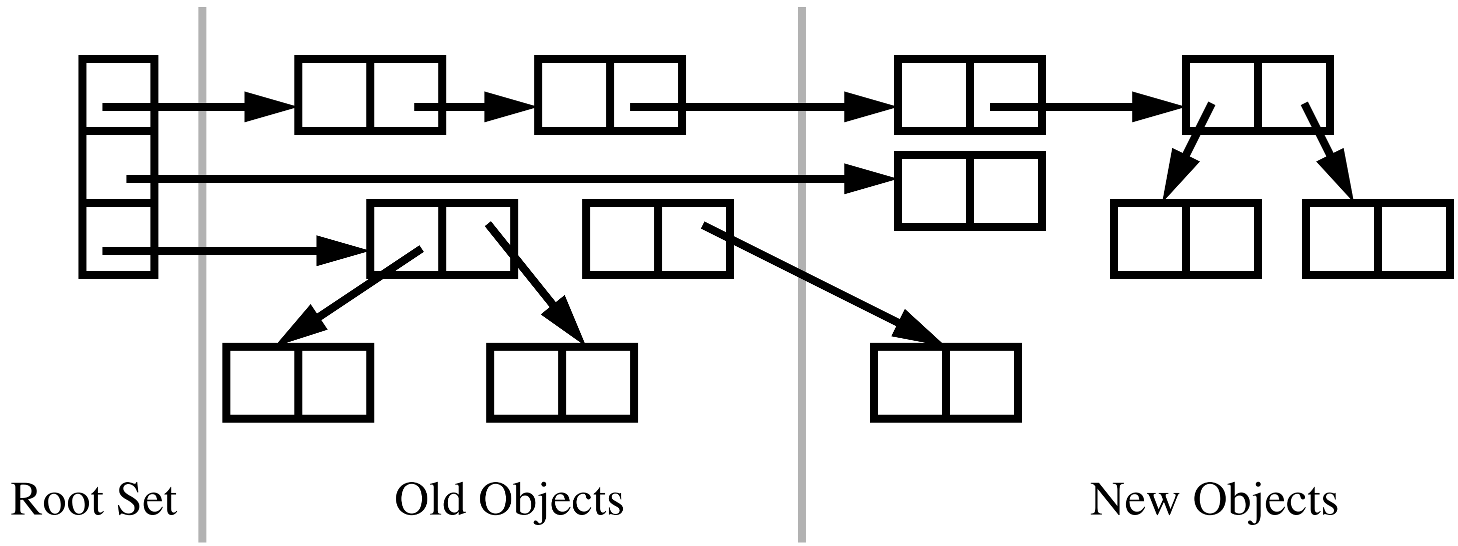
Whenever mutator tries to write a pointer to a white object on a black object:

- Make the black object grey [Steele 75]

- Make the white object grey [Dijkstra et al 78]

Generational GC

- **Focus collection in newly allocated objects**
- **Use a write barrier to keep track of intergenerational pointers**



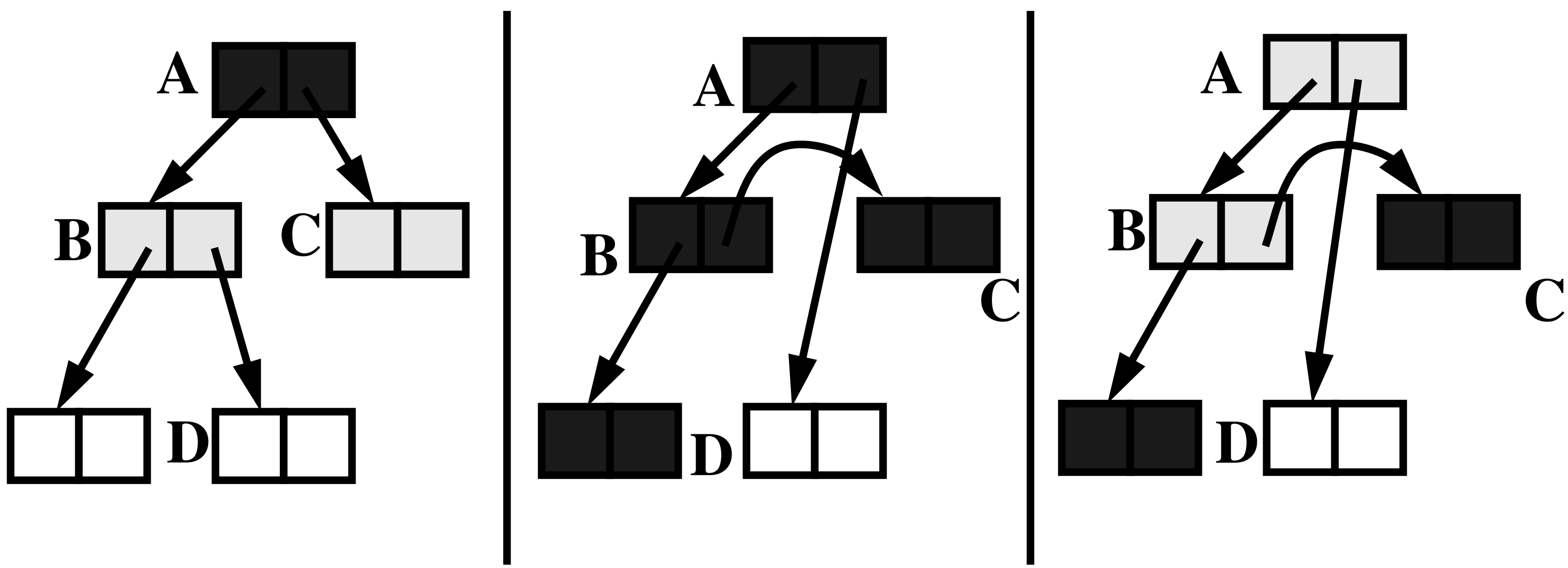
Local Garbage Collection in Renaissance

- Incremental
- Conservative
- Based on [Boehms 91]
- Use Virtual Memory operations
- Algorithm:
  1. Mark reachable objects in parallel
  2. Stop the world
  3. Trace again from marked objects in dirty pages
  4. Start the world
  5. Sweep unmarked objects

Local Garbage Collection in Renaissance (cont.)

1.Mark in Parallel **2. Stop the World** **3. Trace again marked objects in dirty pages**

1.5. SWAP(B->D,A->C)



Distributed Garbage Collection

Why is Distributed GC different than local GC?

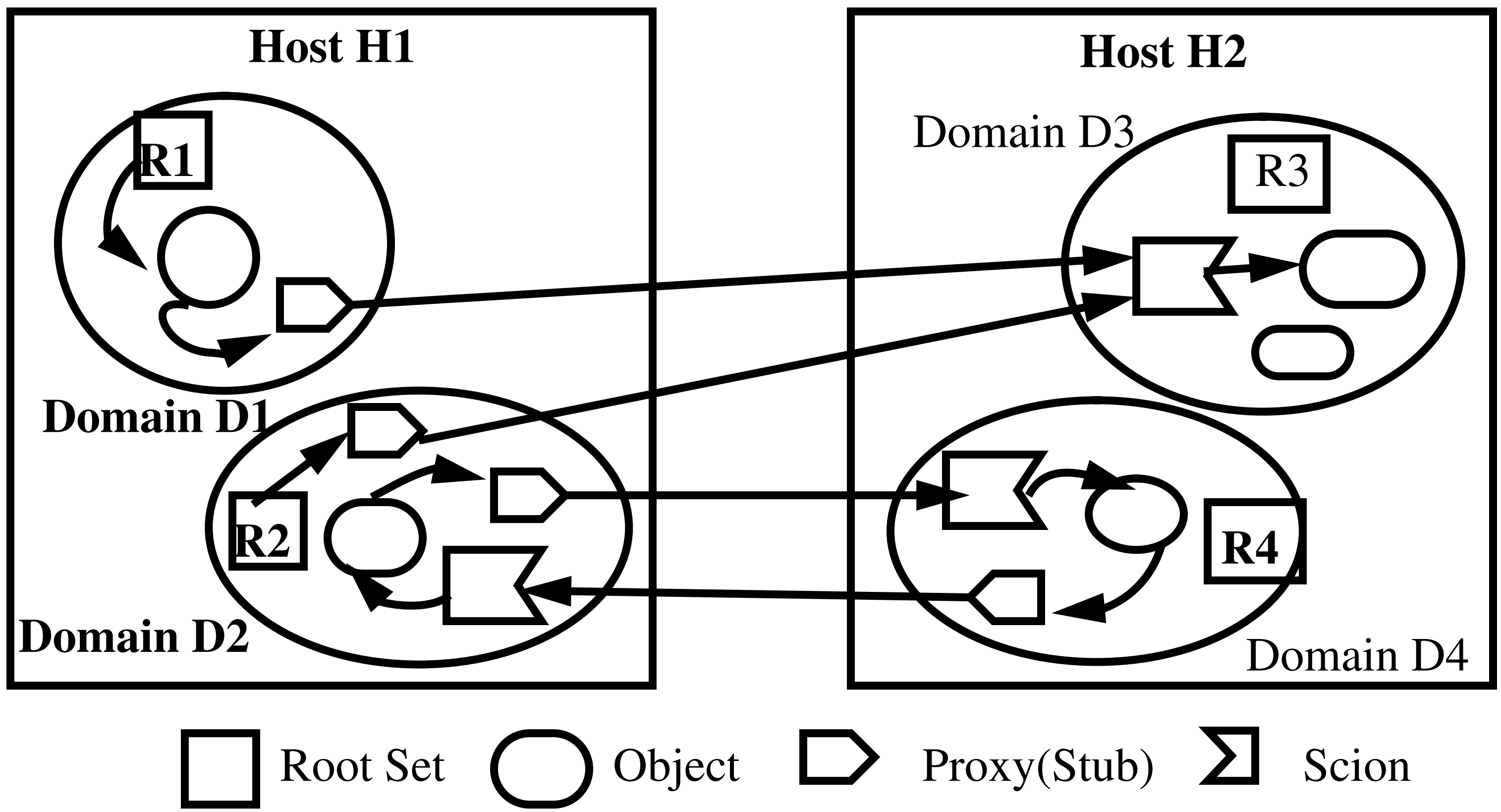
- Very expensive to stop the world
- Very expensive to synchronize
- Message failures

Distributed Garbage Collection (cont.)

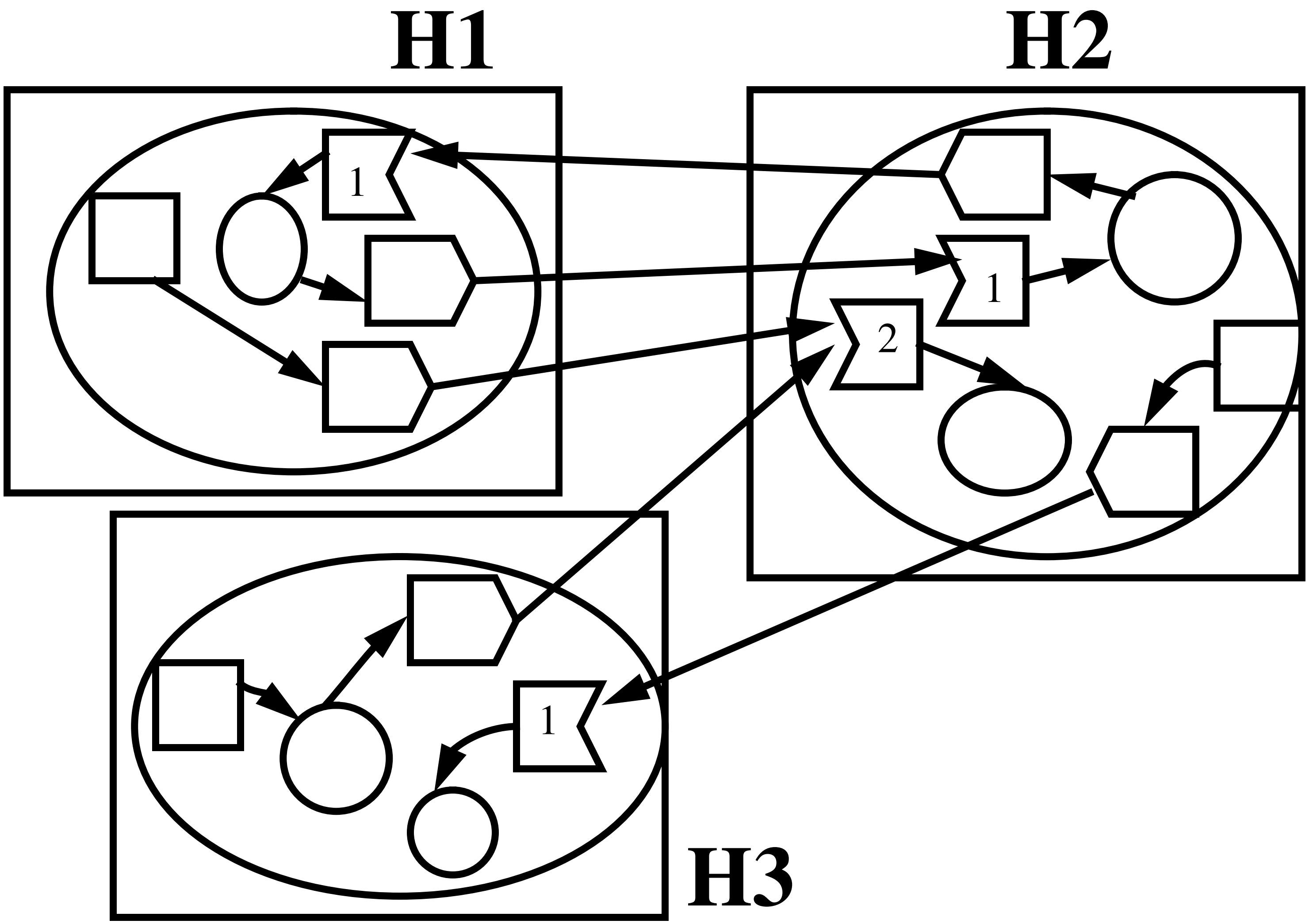
Algorithms:

- Reference Counting
- Reference Listing
- Tracing

Objects, Proxies and Scions



Reference Counting



Reference Counting (cont.)

Advantages:

- **Simple**

Disadvantages:

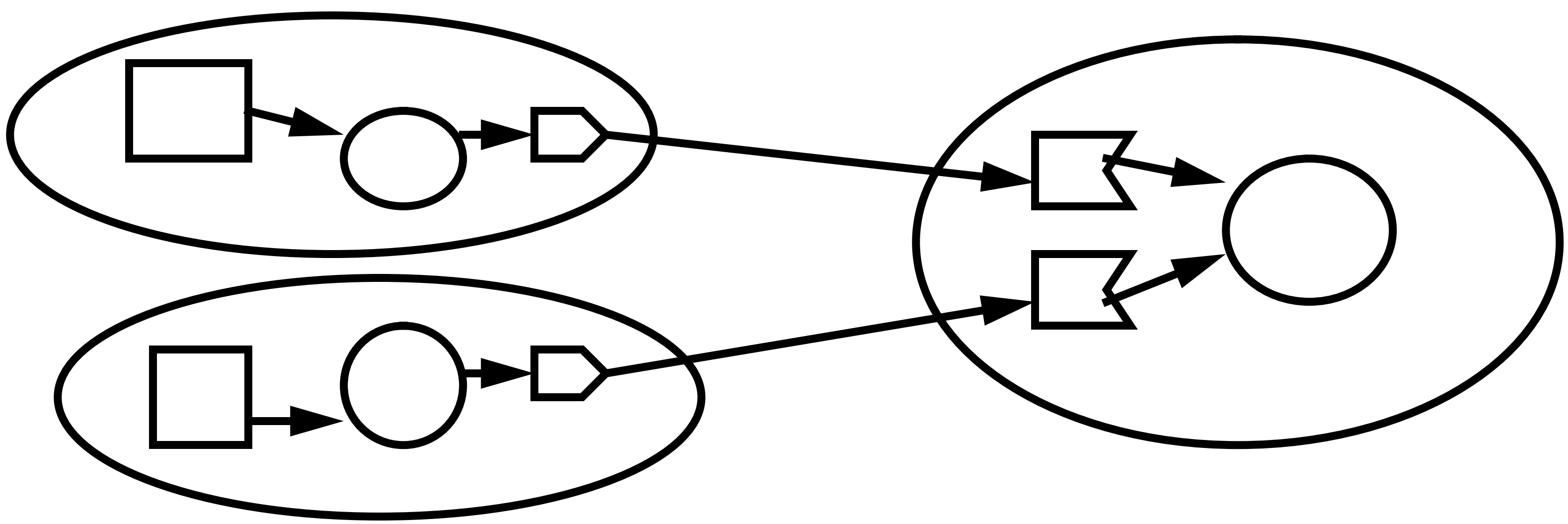
- **Operations are not idempotent**
- **No resilience to message failures**
- **Unable to collect garbage cycles**

-

Reference Listing

[Birrel et al., 1993a][Shapiro et al. 1990]

Use a proxy-scion pair for every domain instead of only a counter.



Reference Listing (cont.)

Advantages:

- Operations are idempotent
- Resilient to message failures
- Scalable

Disadvantages:

- Unable to collect garbage cycles

Tracing Distributed Garbage Collection

Mark & Sweep

[Juul & Jul, 1992]

- Expensive to synchronize domains

- Not scalable

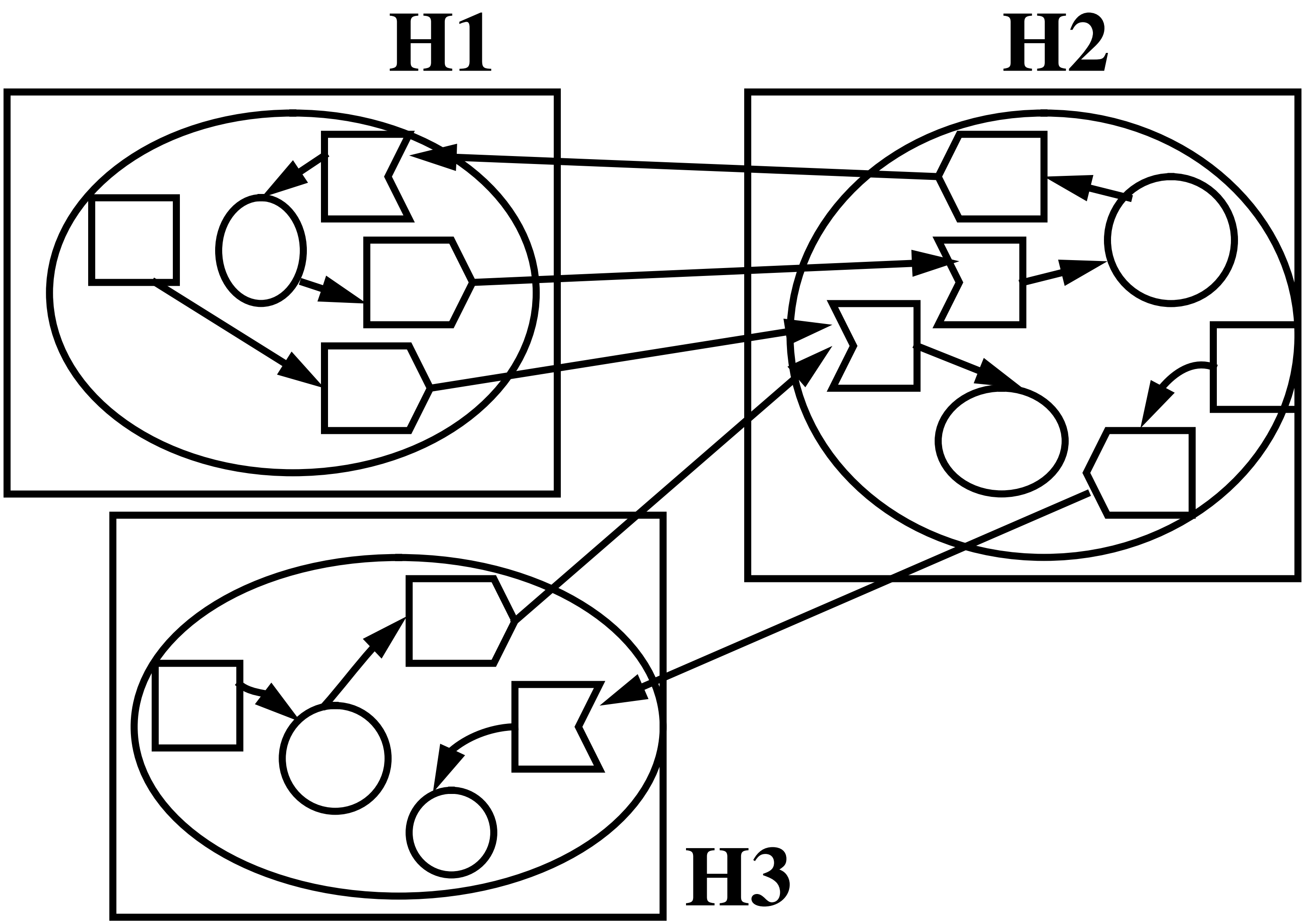
[Lang et al., 1993] divides domains in groups before doing M&S. It is still not scalable

Tracing Distributed Garbage Collection (cont.)

Hughes Algorithm [Hughes, 1985]

- Propagates timestamp of garbage collection during marking.
- Timestamps of garbage objects are not increased.
- Objects with a timestamp below some global threshold are considered garbage.
- Requires a termination algorithm and global time. Not scalable.
- [Ladin & Liskov, 1992] Use a centralized server.

Hughes Algorithm [Hughes, 1985] (cont.)



Distributed GC in Renaissance

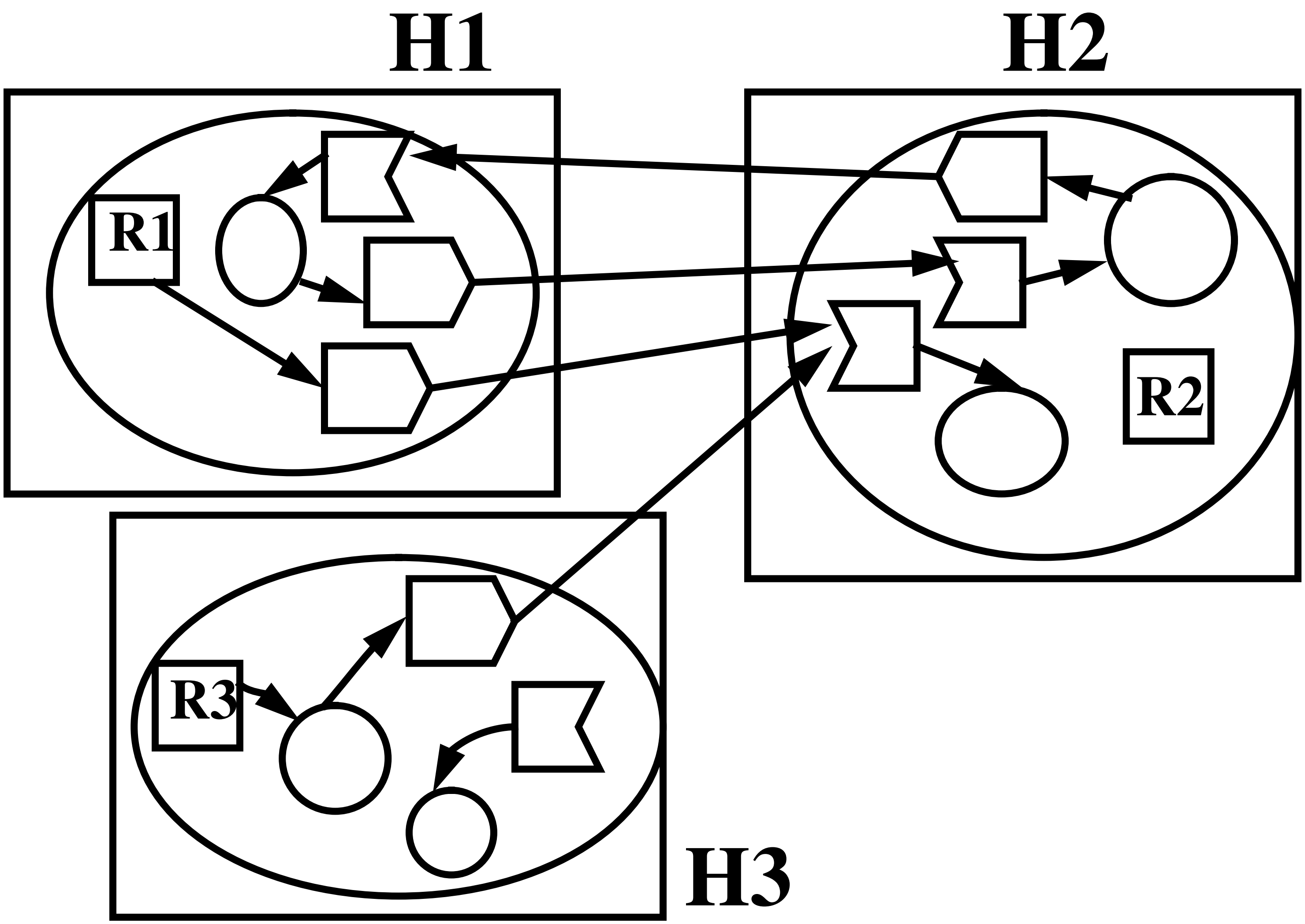
Two methods:

1. **Ultra Light-Weight Garbage Collection.**
2. **Safe Cyclic Garbage Collection.**

Ultra Light-Weight Garbage Collection

- Stubs send keep alive messages to maintain objects alive. Objects are collected after some time-out.
- Advantages:
  - Passing pointer references is extremely cheap.
  - No need of reference listing.
  - Very easy to implement.
  - No need of synchronization.
- Disadvantages:
  - Acyclic
  - Takes too long to collect garbage
  - Temporal network failures can cause erroneous collections

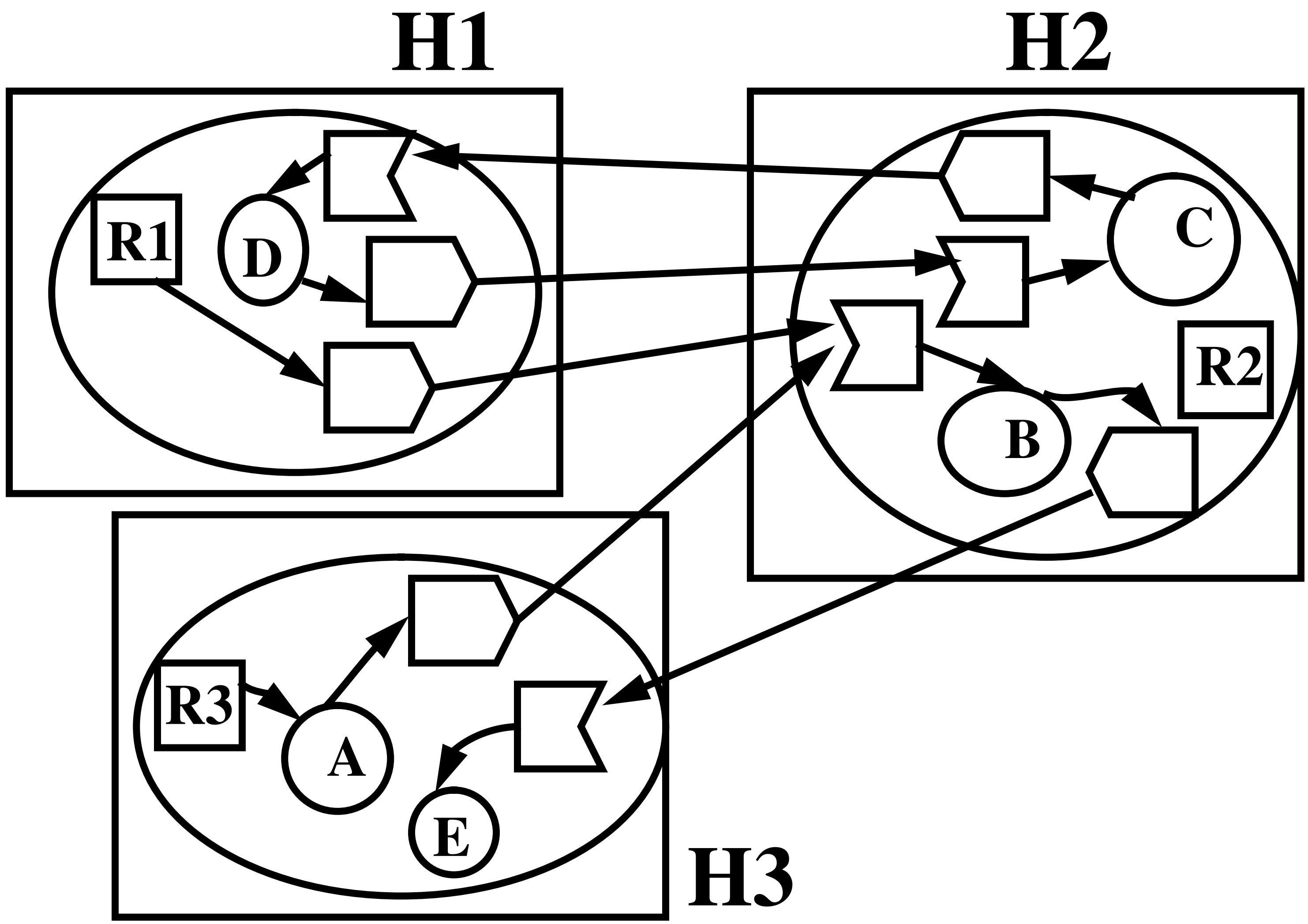
Ultra Light-Weight Garbage Collection



Safe Cyclic Garbage Collection.

- Hybrid: Uses reference listing and tracing.
- It propagates reachability information to every remote object in each mark phase.
- All the elements in the reachability information are garbage when:
  * It is complete
  * It is valid
  * There are not roots included

Safe Cyclic Garbage Collection.



Taxonomy of Distributed GC Techniques

Distributed GC Techniques	Cyclic	Floating Garbage	Large Scale	Imple-mented	Space Failure	Message Failure		
						duplic.	lost	late
Reference Counting								
[Bevan, 1987]								●
[Dickman, 1991]	○	●				●		●
[Picker, 1991]		●	●	●				●
[Goldberg, 1989]								●
Reference Listing								
[Shapiro et al., 1990]		●	●	●	●	●	●	●
[Birrel et al., 1993]		●	●	●	○	●	●	●
Tracing								
[Ali, 1984]	●							
[Hughes, 1985]	●	●						
[Ladin & Liskov, 1992]	●				●			
[Lang et al., 1992]	○	●			●			
[Juul and Jul, 1992]	●			●				