Ulterior Reference Counting: Fast Garbage Collection without a Long Wait

Stephen M Blackburn and Kathryn S Mckinley

Present by Qi Chen
March 6, 2012

Slides adapted from presentation by Dimitris Prountzos

Outline

- Problem Statement
- Background
  - Reference Counting
- Ulterior Reference Counting (URC)
- URC Implementation
- Evaluation
- Conclusion

Problem Statement

- Throughput/Responsiveness trade-off
  - High throughput: mark-sweep (MS)
  - Short pause time: reference counting (RC)

benchmark	Total Time (sec)		Max Pause Time (ms)	
	BG-MS	RC	BG-MS	RC
_228_jack	7.2	12.7	185	72
_209_db	19.2	21.3	238	43

Problem Statement

- Throughput/Responsiveness trade-off
  - High throughput: mark-sweep (MS)
  - Short pause time: reference counting (RC)
- Any collector can achieve the two goals?

Problem Statement

- Throughput/Responsiveness trade-off
  - High throughput: mark-sweep (MS)
  - Short pause time: reference counting (RC)
- Any collector can achieve the two goals?
  - **Ulterior Reference Counting (URC)**

Ulterior RC Approach

- Match mechanisms to object demographics
  - Copying nursery space
    - Highly mutated, high mortality young objects
    - Ignore nursery pointer mutations
    - GC time proportional to survivors
  - RC mature space
    - Low mutation, low mortality old objects
    - GC time proportional to dead objects and pointer mutations
- Generalize deferred RC to heap objects
  - Defer fields of highly mutated objects and enumerate them quickly
  - Reference count only infrequently mutated fields

Background

- Reference Counting
  - Advantage
    - Incremental: the work of garbage detection is spread out over every mutation
  - Disadvantage
    - Unable to reclaim cycles
      - Solution: additional algorithm
    - Tracking every pointer mutation is expensive
      - Solution: Deferal, Buffering, Coalescing

Background

- RC Formal Definitions
  - Mutation event: RCM(p)
    - RC(Pbefore)--, RC(Pafter)++
    - May be buffered or performed immediately
  - Retain event: RCR(p)
    - Zero count table (ZCT)
    - Generate a temporary increment for p
  - Deferral
    - No mutation event generates RCM(p)
    - Need a RCR(p) to preserve objects

Background

- RC Optimization Mechanism: to reduce computation overhead
  - **Buffering**
    - apply RC(p)--, RC(p)++ later
  - **Coalescing**
    - apply RCM (p) only for the initial and final values of p (coalesce intermediate values)

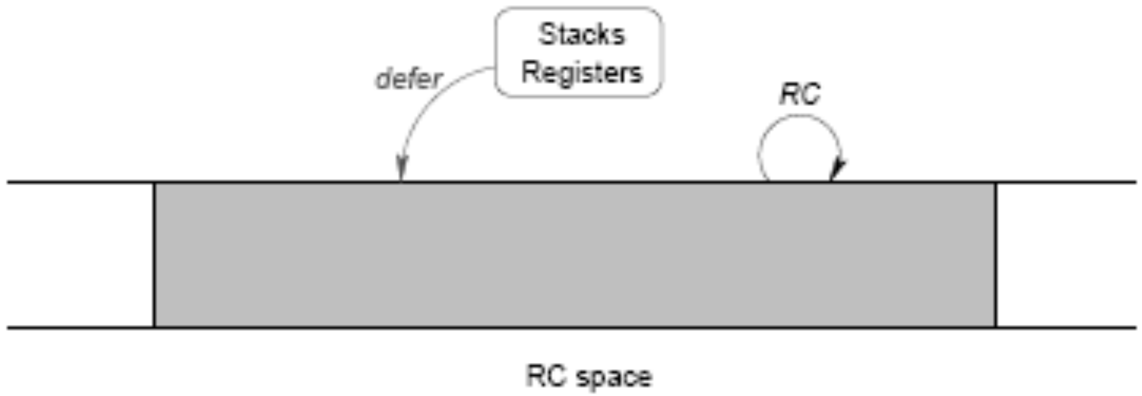
      {RCM(p), RCM(p^1), ... RCM(p^n)} → RC($p_{initial}$)--, RC(p_{final})++
  - **Deferral**
    - Defer RC events.

Ulterior Reference Counting

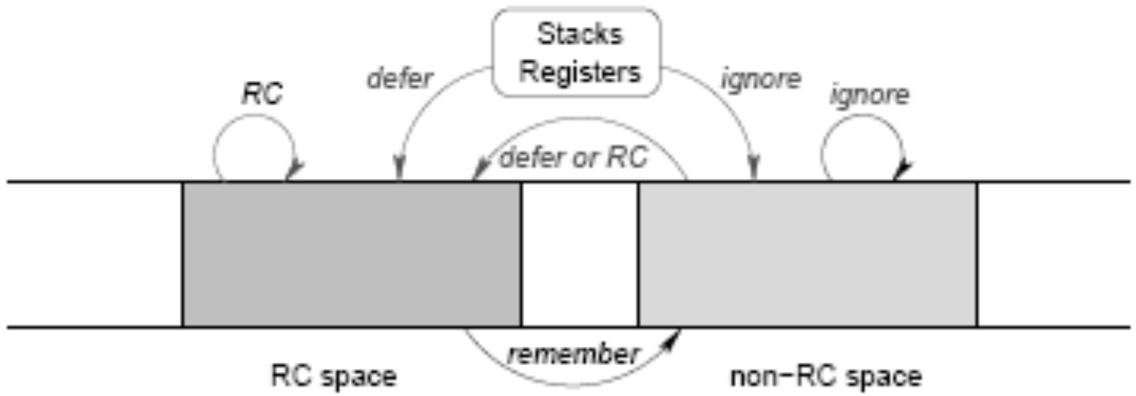
- Idea: Extends deferral to select heap pointers
  - e.g. pointers from nursery space to mature space
- Deferral is not a fixed property of a pointer
  - e.g. an object can be moved between nursery and mature spaces.
- Integrate Event: RCI(p)
  - Change p from deferred to not-deferred.

Ulterior Reference Counting

- Generalizing Deferral



(a) Classic deferred reference counting



(b) A simple instance of URC

Ulterior Reference Counting

- A Generational RC Hybrid Collector (BG-RC)
  - Combine a bounded copying nursery with RC.
  - For young objects
    - Bump-pointer allocation
    - Copying collection
  - For old objects
    - Free-list allocation
    - Reference counting collection

Ulterior Reference Counting

- Nursery phase
  - Scan roots
  - Process the modified object buffer
  - Reclaim nursery
- RC phase
  - Process decrement buffer, recursively decrement
  - Reclaim old objects
  - Cycle detection if needed

Ulterior Reference Counting

- Write Barrier
  - Remember pointers into the nursery from the non-nursery spaces. (RC, immortal and boot image spaces)
  - Generate RCM(p) for mutations to pointer fields within the non-nursery spaces.
  - An object remembering coalescing barrier.

Ulterior Reference Counting

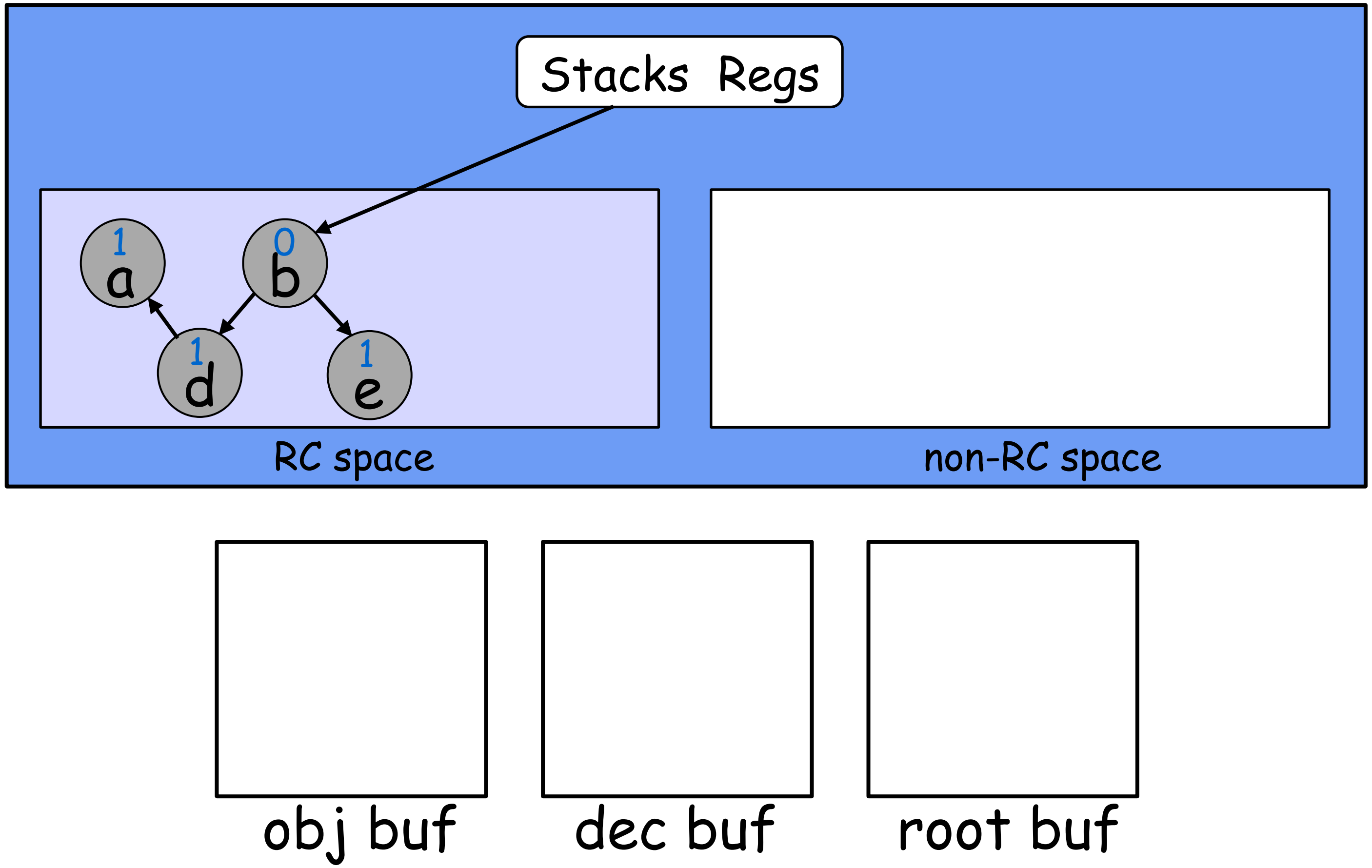
- Write Barrier

```
private void writeBarrier(VM_Address srcObj,
                          VM_Address srcSlot,
                          VM_Address tgtObj)
  throws VM_PragmaInline {
  if (getLogState(srcObj) != LOGGED)
    writeBarrierSlow(srcObj);
  VM_Magic.setMemoryAddress(srcSlot, tgtObj);
}


private void writeBarrierSlow(VM_Address srcObj)
  throws VM_PragmaNoInline {
   if (attemptToLog(srcObj)) {
    modifiedBuffer.push(srcObj);
    enumeratePointersToDecBuffer(srcObj);   // trade-off for sparsely
    setLogState(srcObj, LOGGED);            // modified objects
  }
}
```

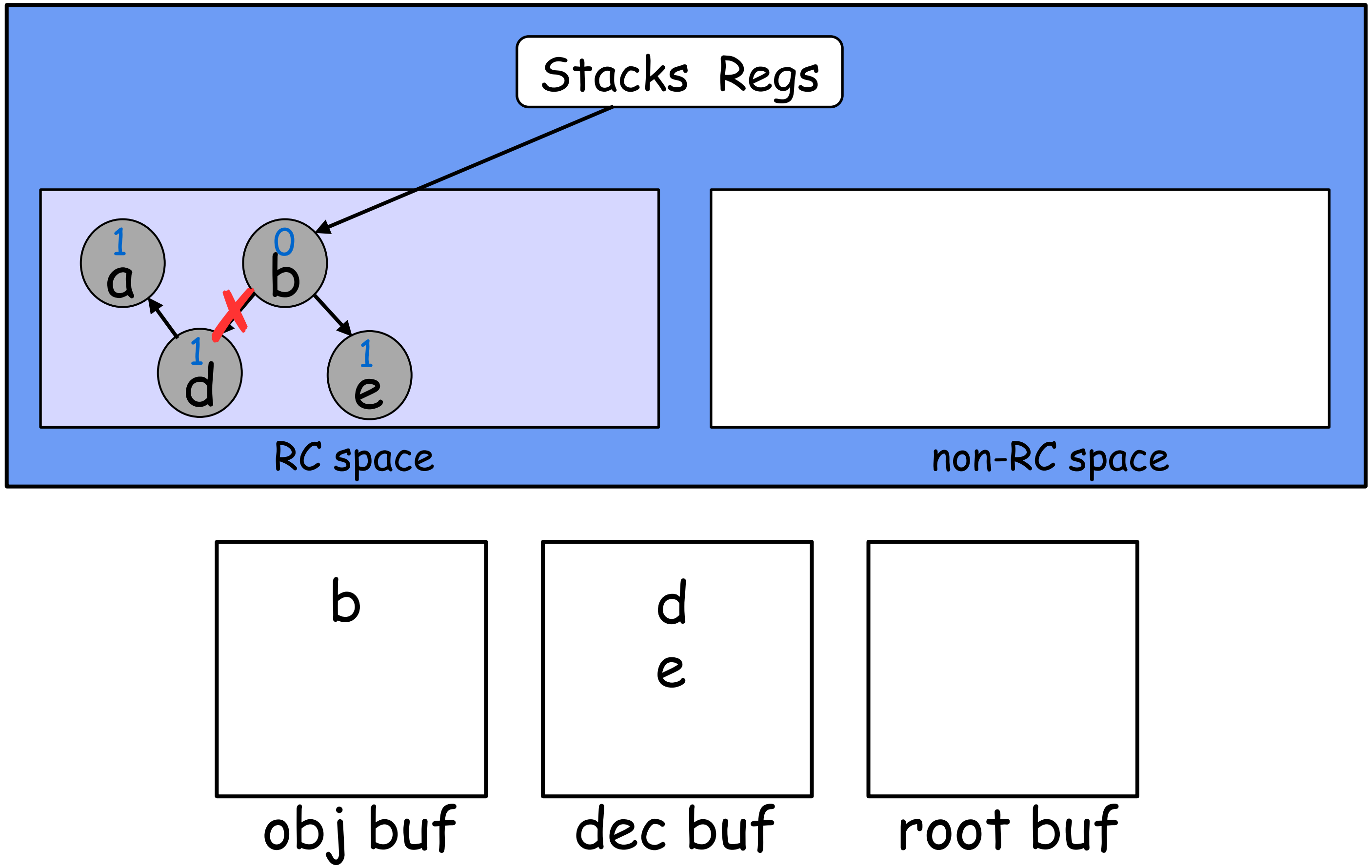
Ulterior Reference Counting

- Mutation Phase



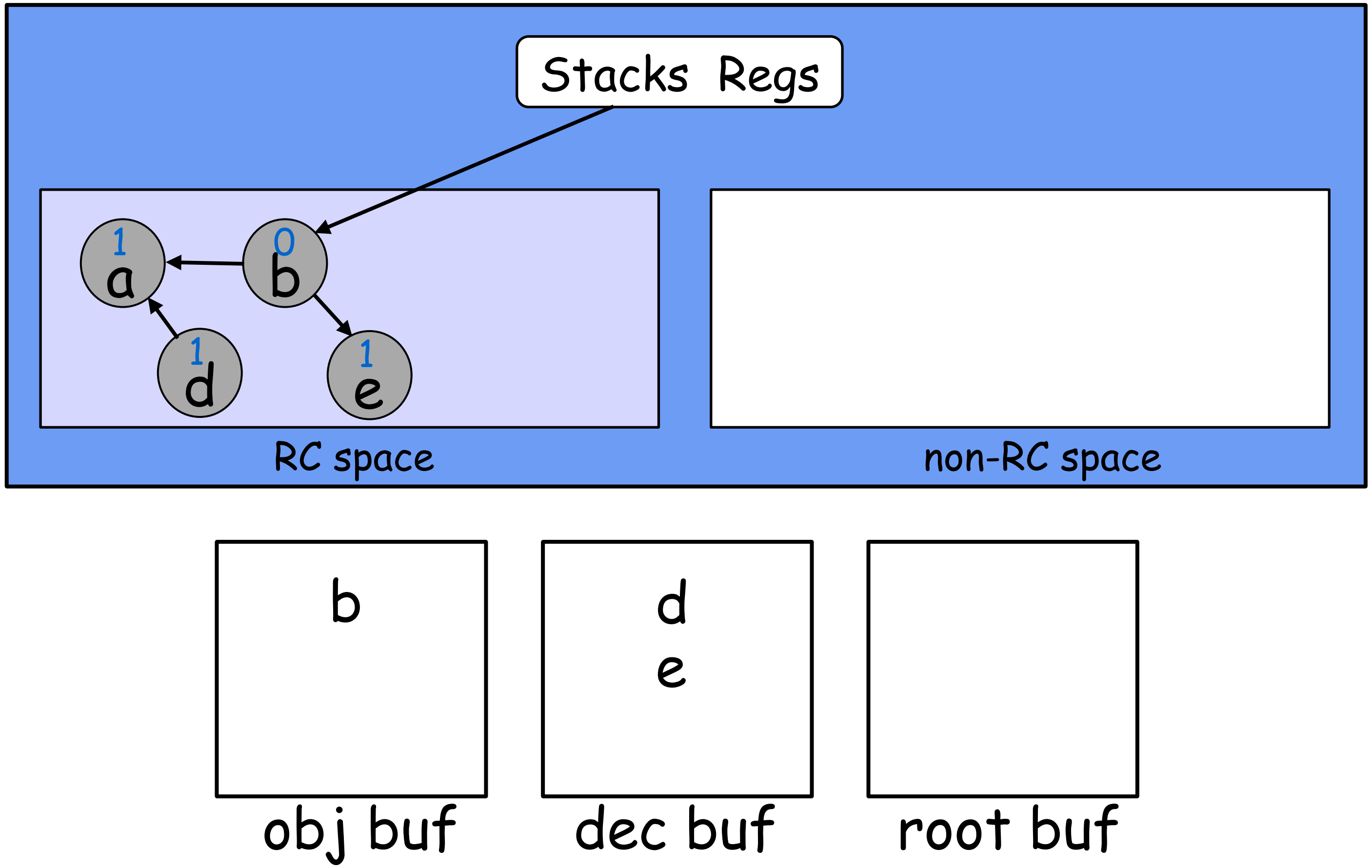
Ulterior Reference Counting

- Mutation Phase



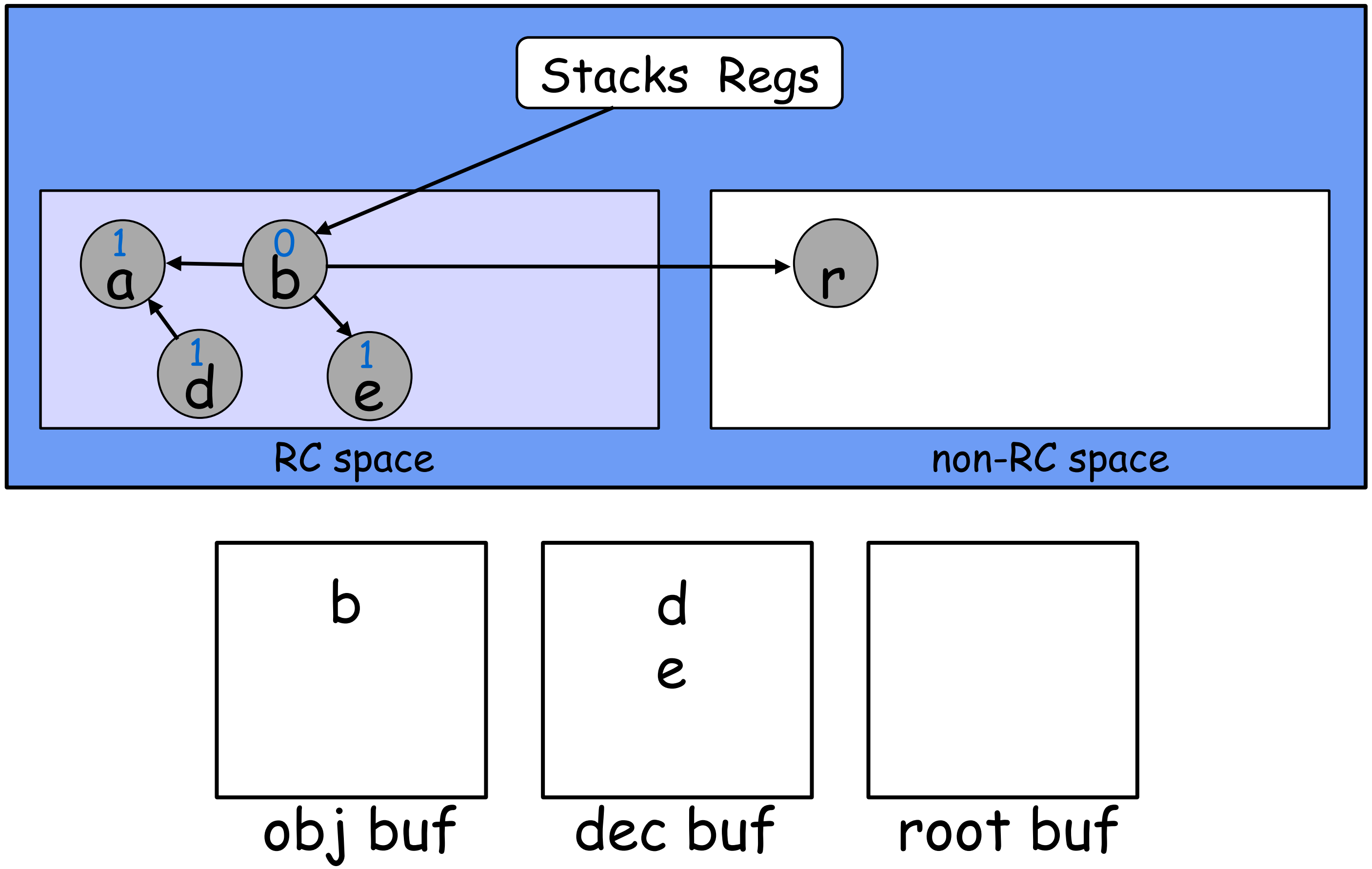
Ulterior Reference Counting

- Mutation Phase



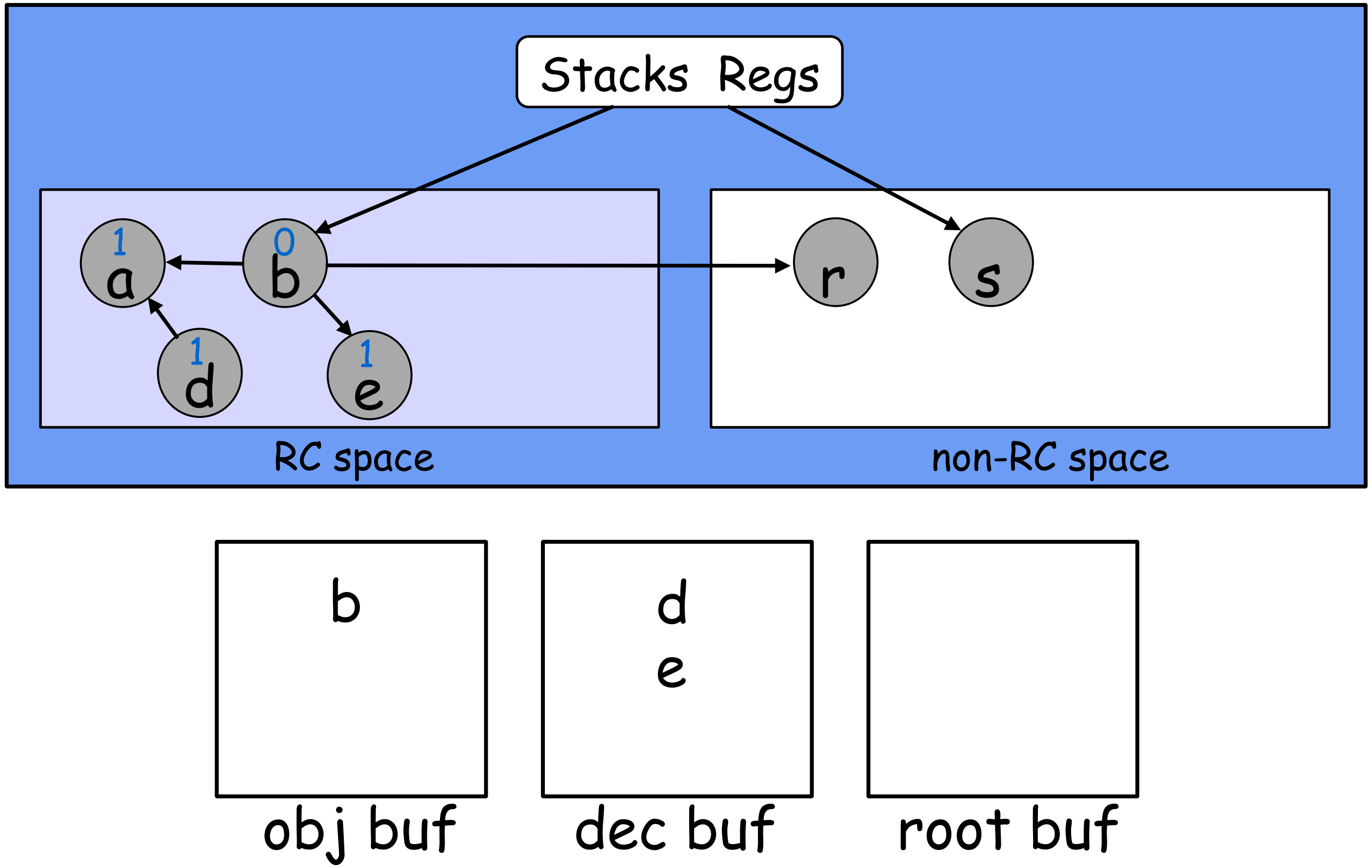
Ulterior Reference Counting

- Mutation Phase



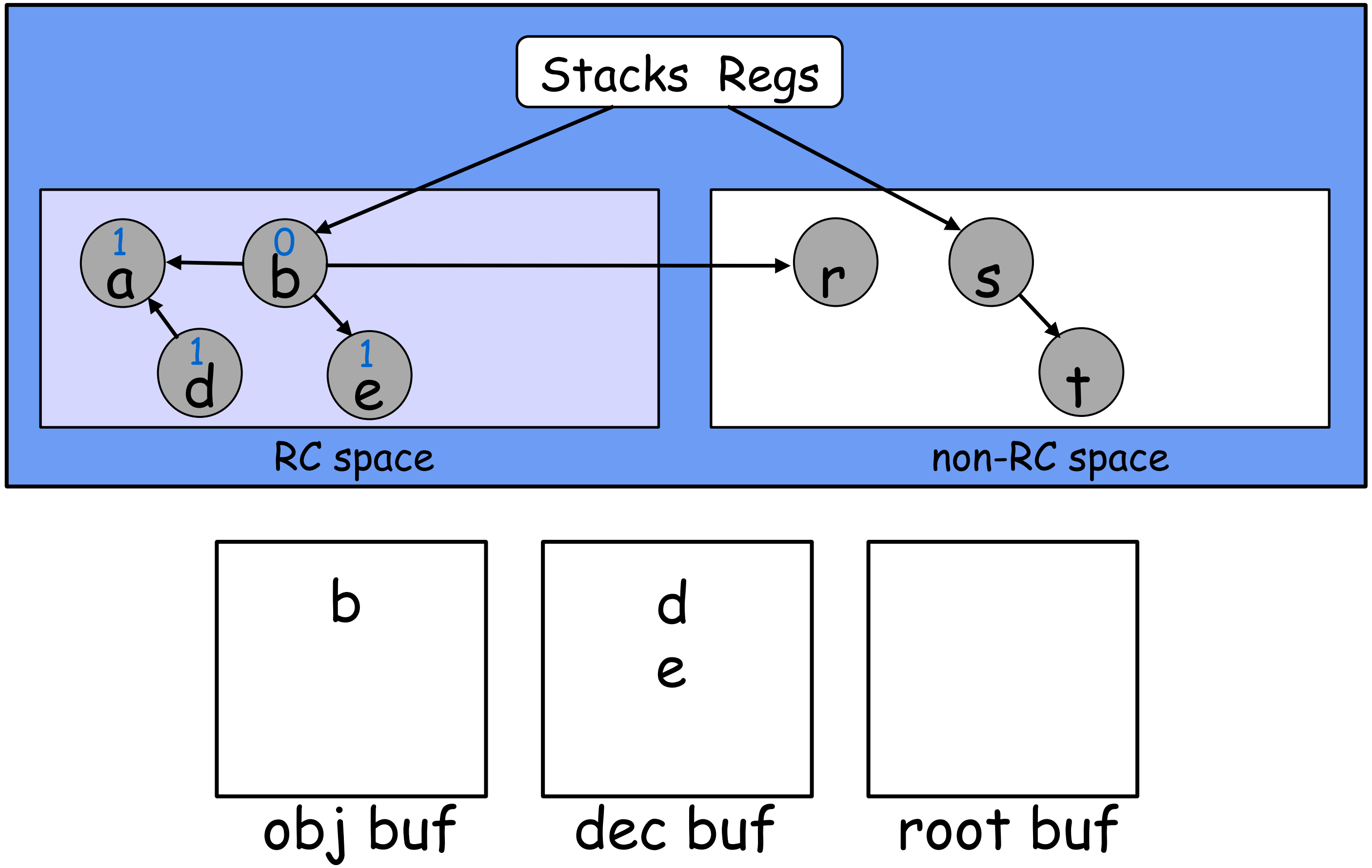
Ulterior Reference Counting

- Mutation Phase



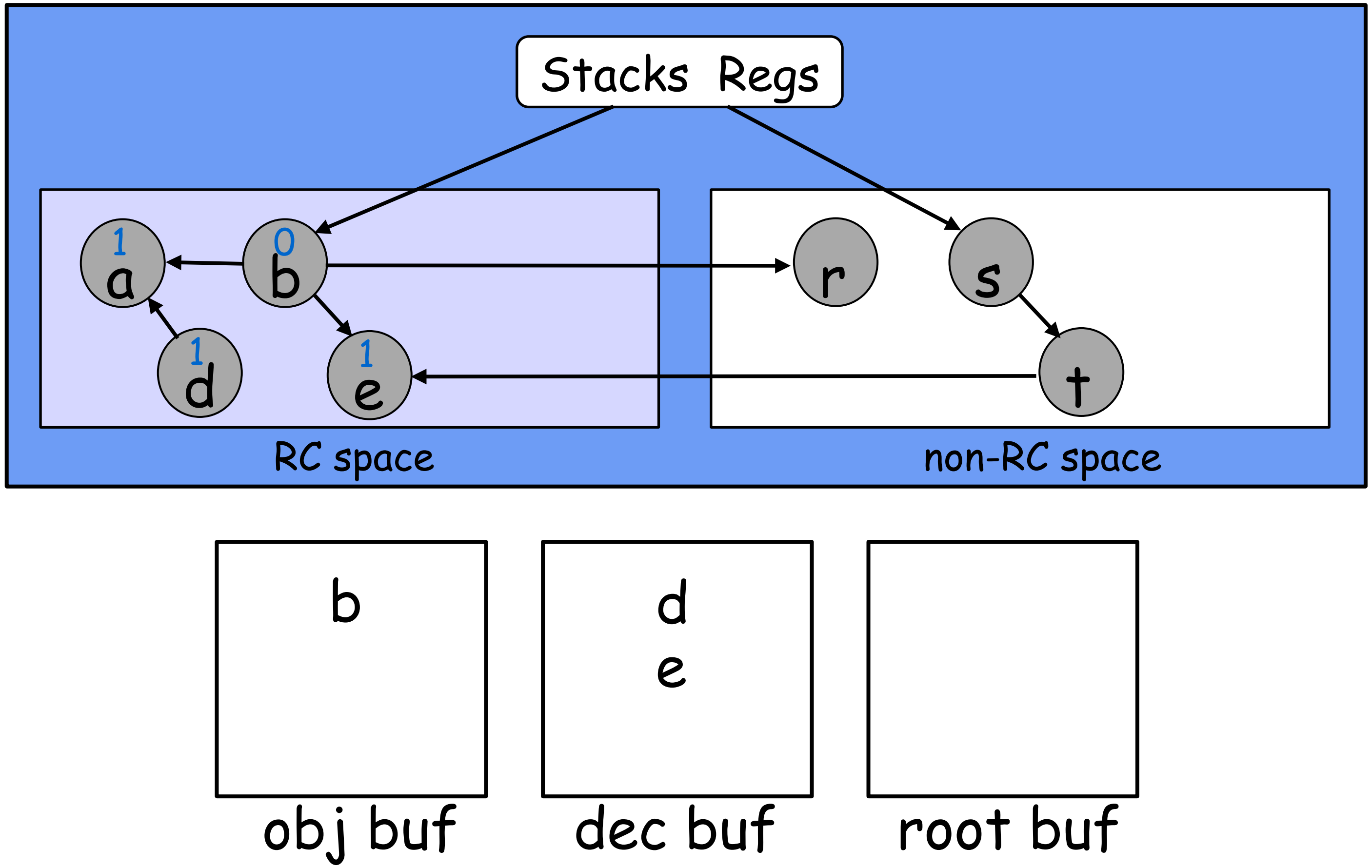
Ulterior Reference Counting

- Mutation Phase



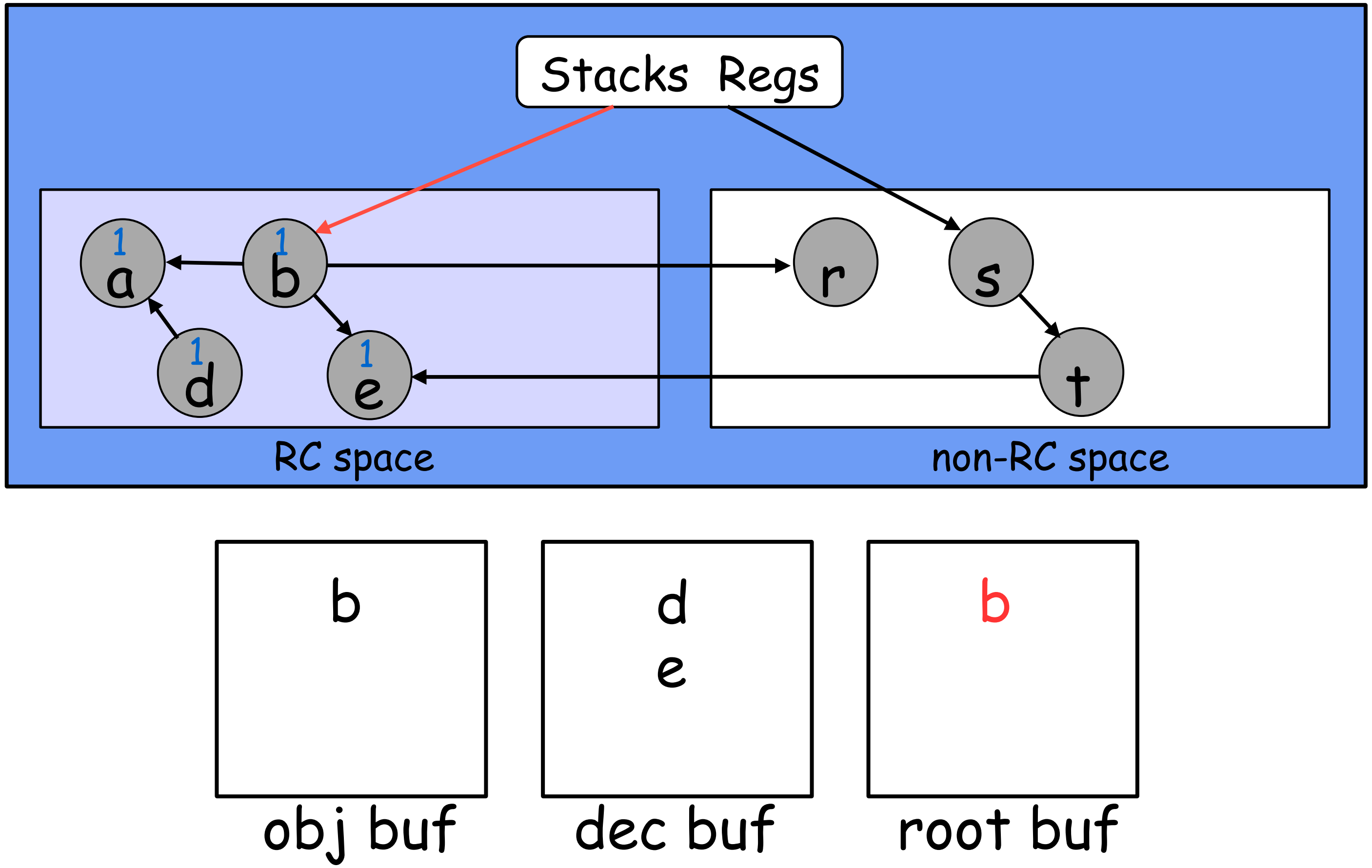
Ulterior Reference Counting

- Mutation Phase



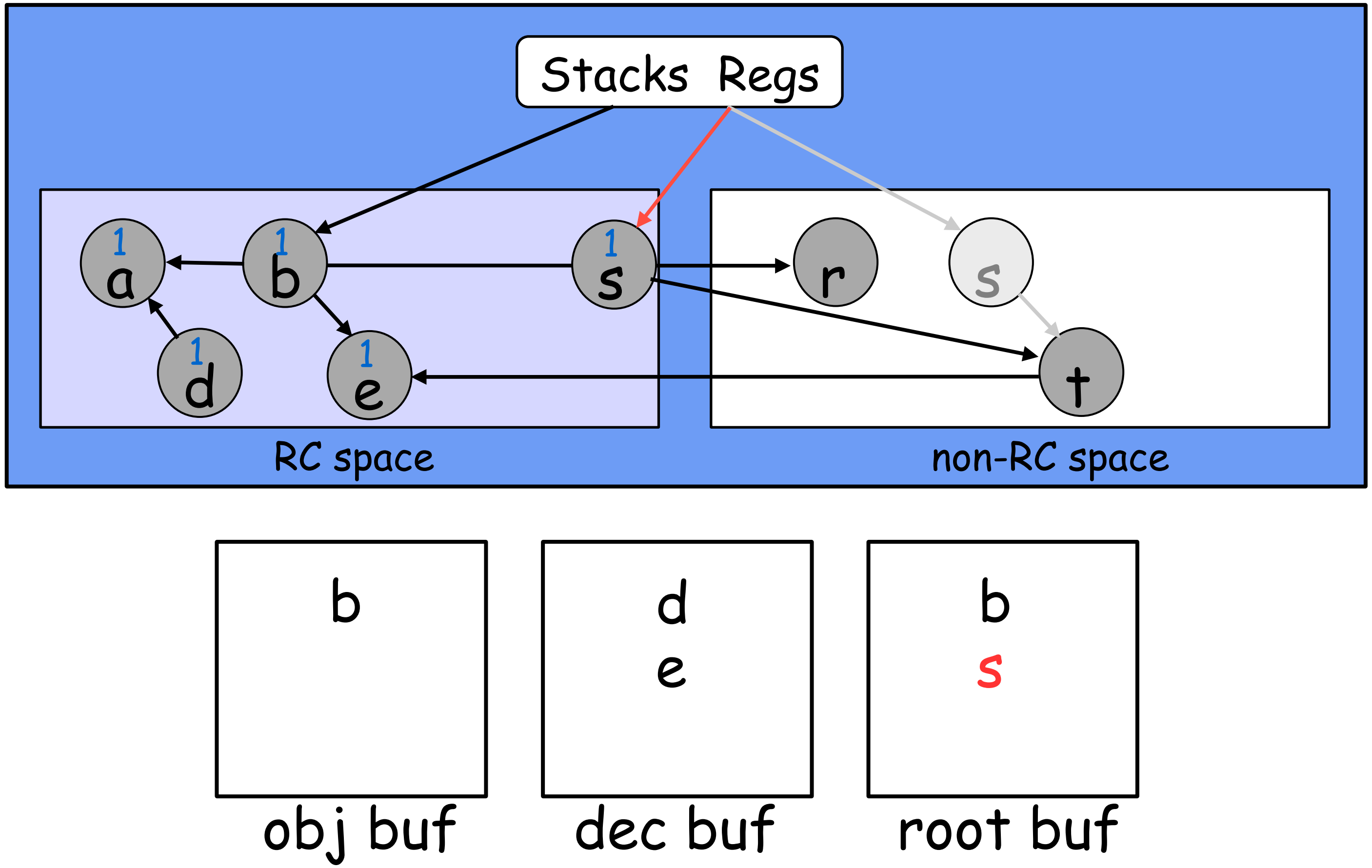
Ulterior Reference Counting

- Nursery Collection: Scan Roots



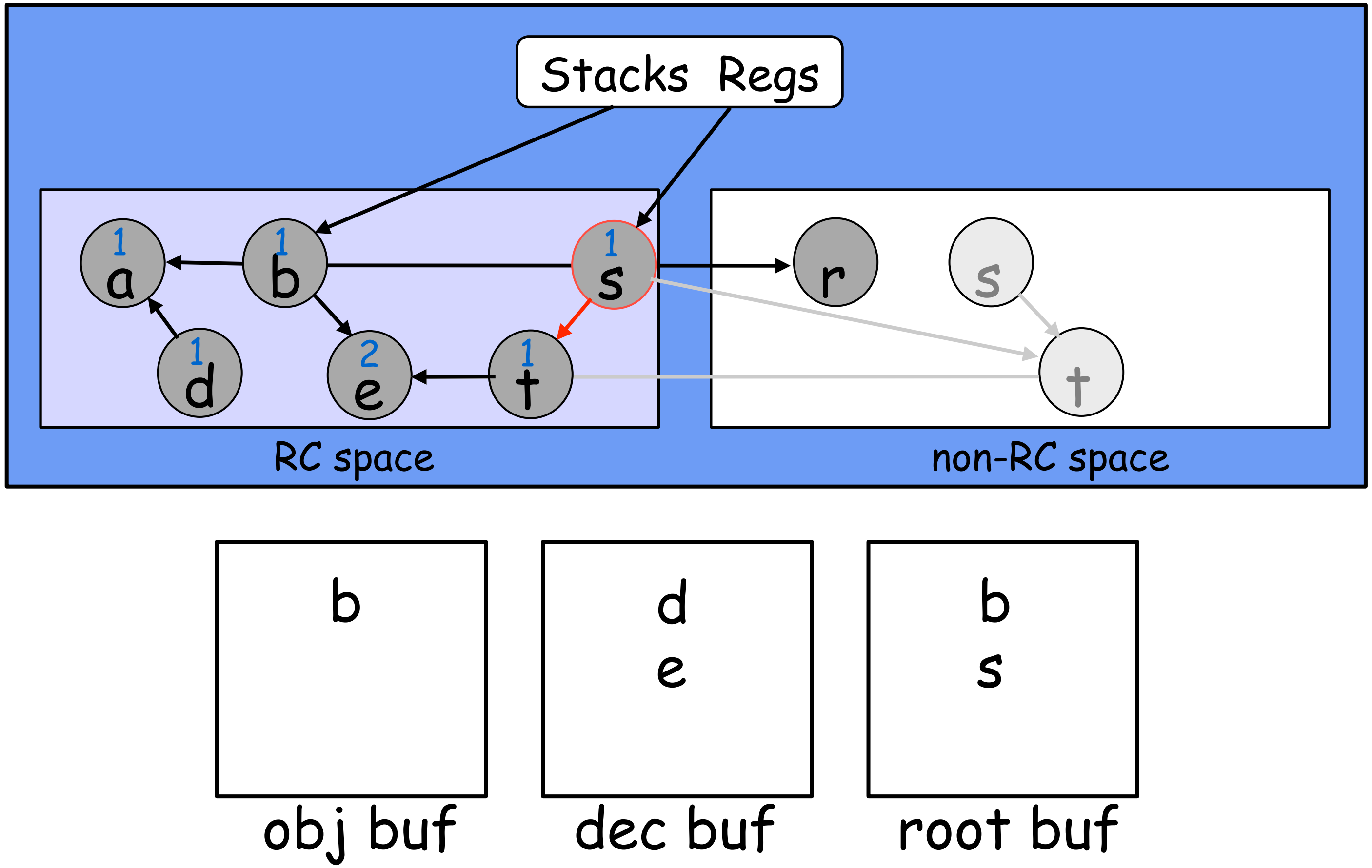
Ulterior Reference Counting

- Nursery Collection: Scan Roots



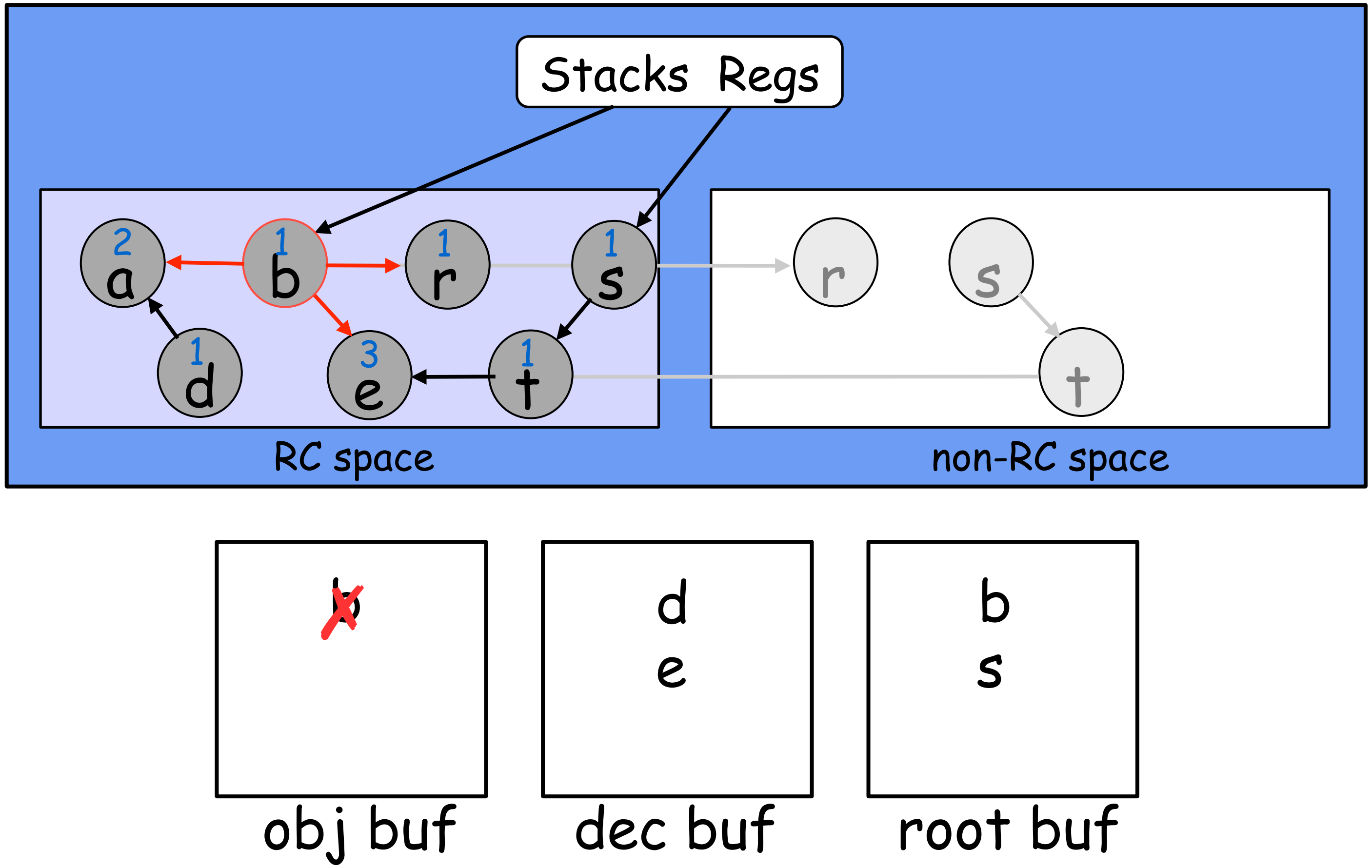
Ulterior Reference Counting

- Nursery Collection: Scan Roots



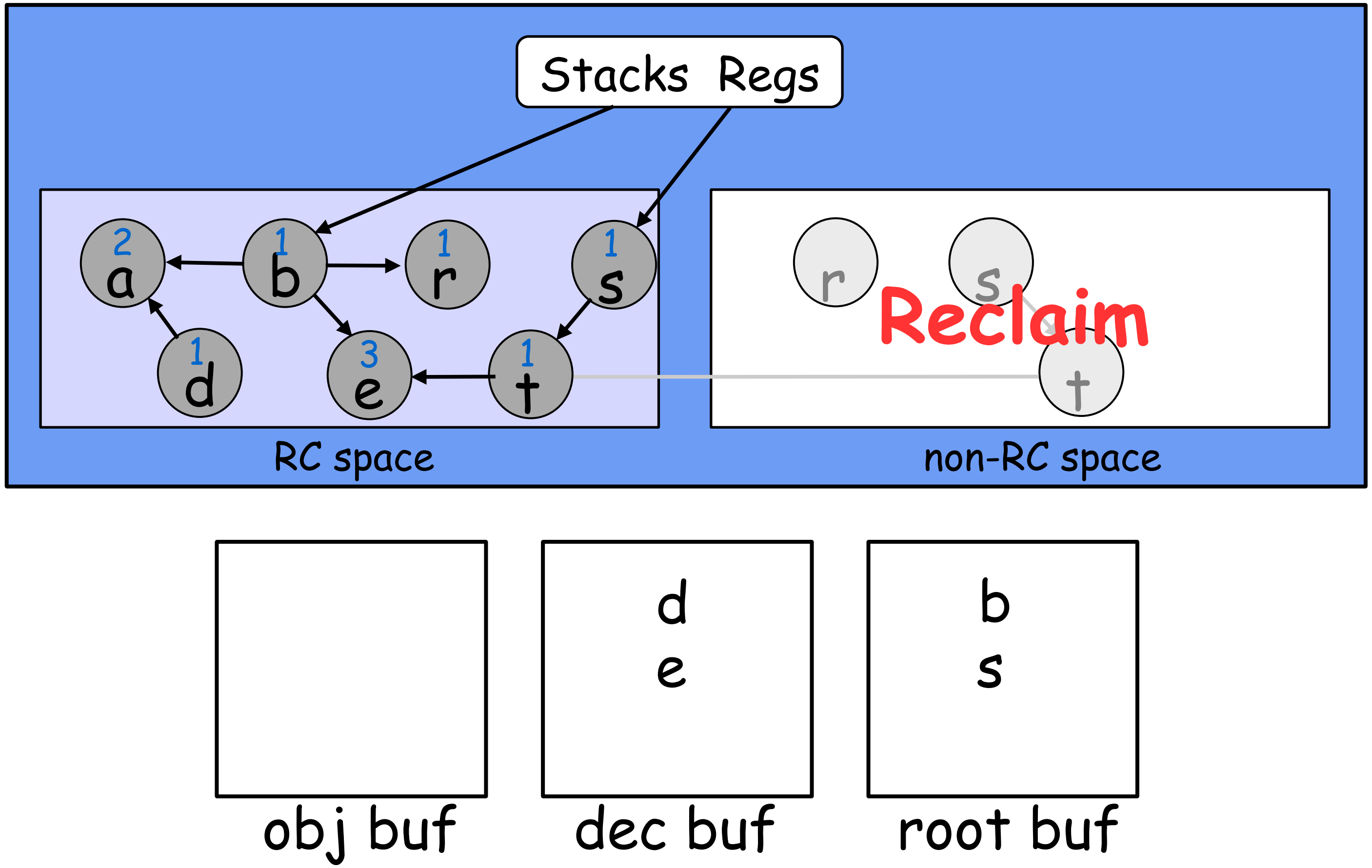
Ulterior Reference Counting

- Nursery Collection: Process Object Buffer



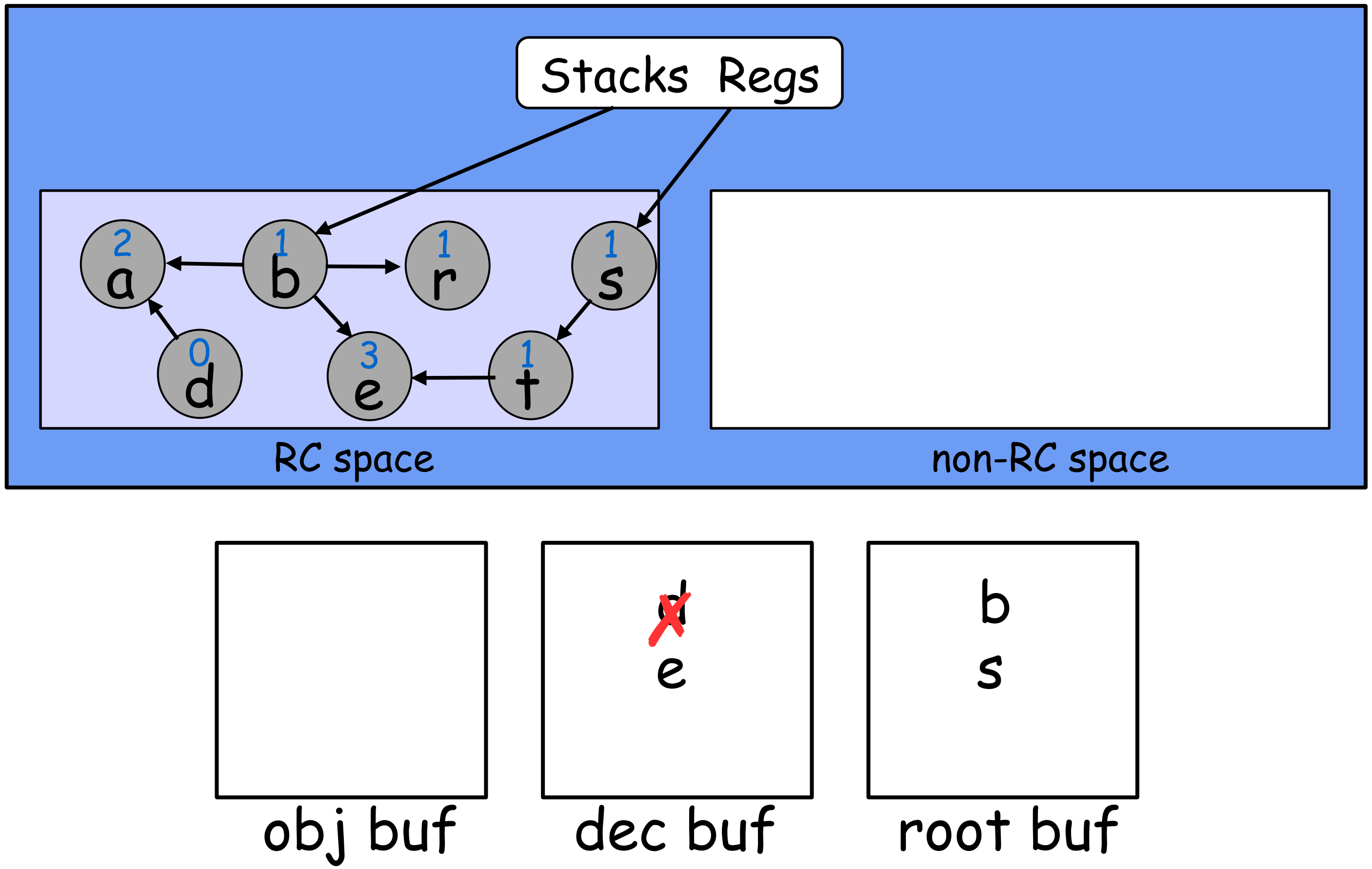
Ulterior Reference Counting

- Nursery Collection: Reclaim Nursery



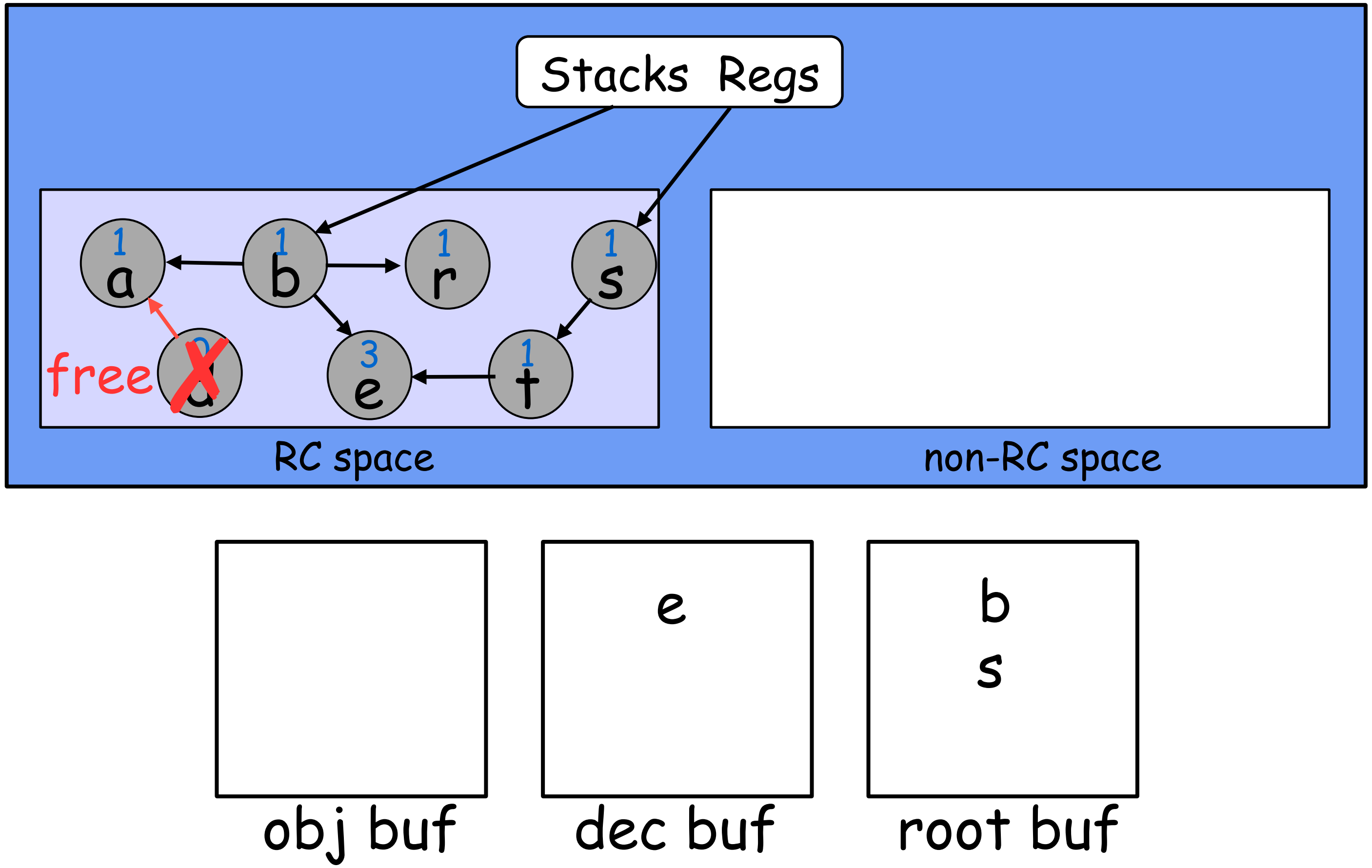
Ulterior Reference Counting

- RC Collection: Process Decrement Buffer



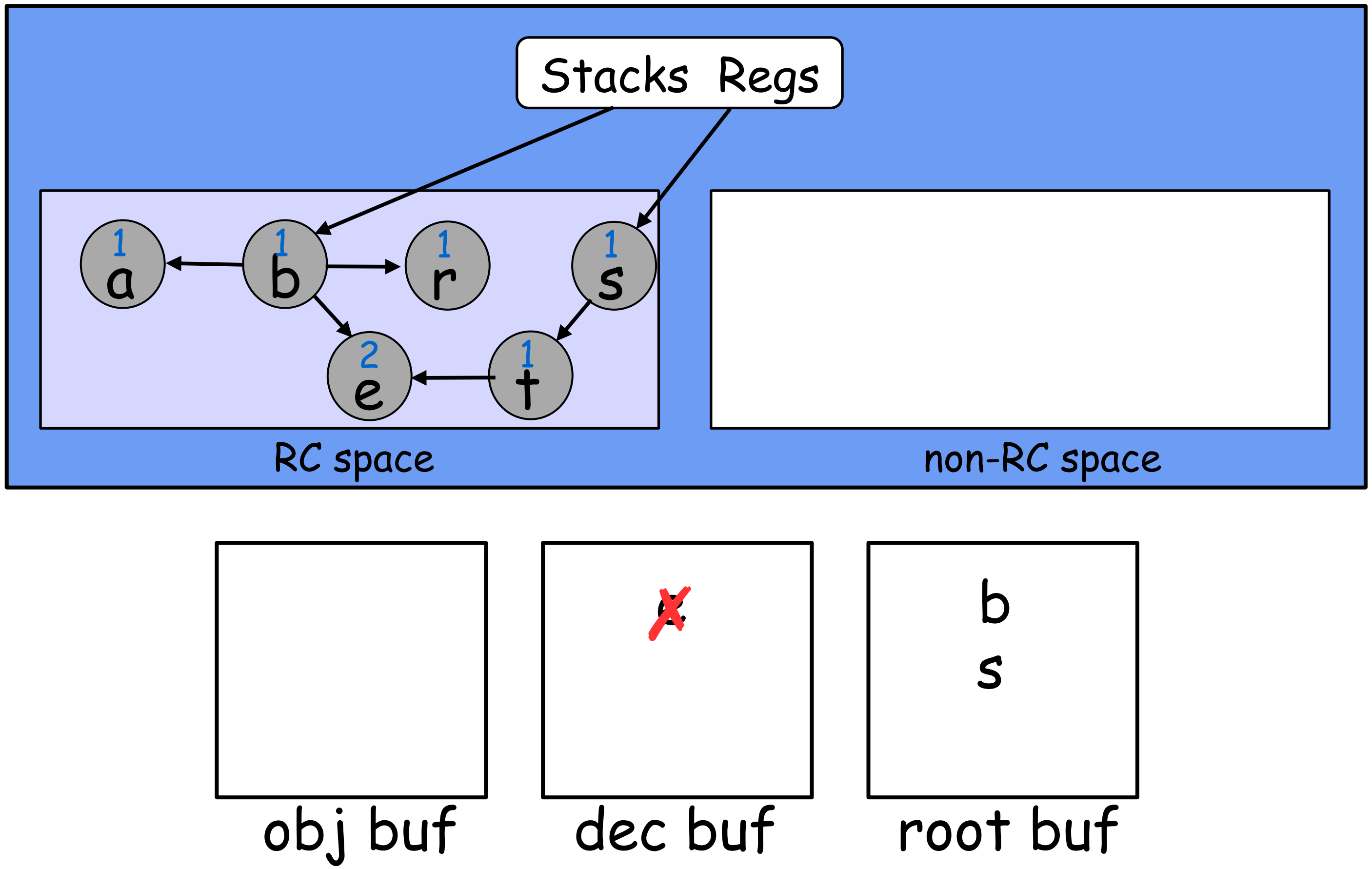
Ulterior Reference Counting

- RC Collection: Recursive Decrement



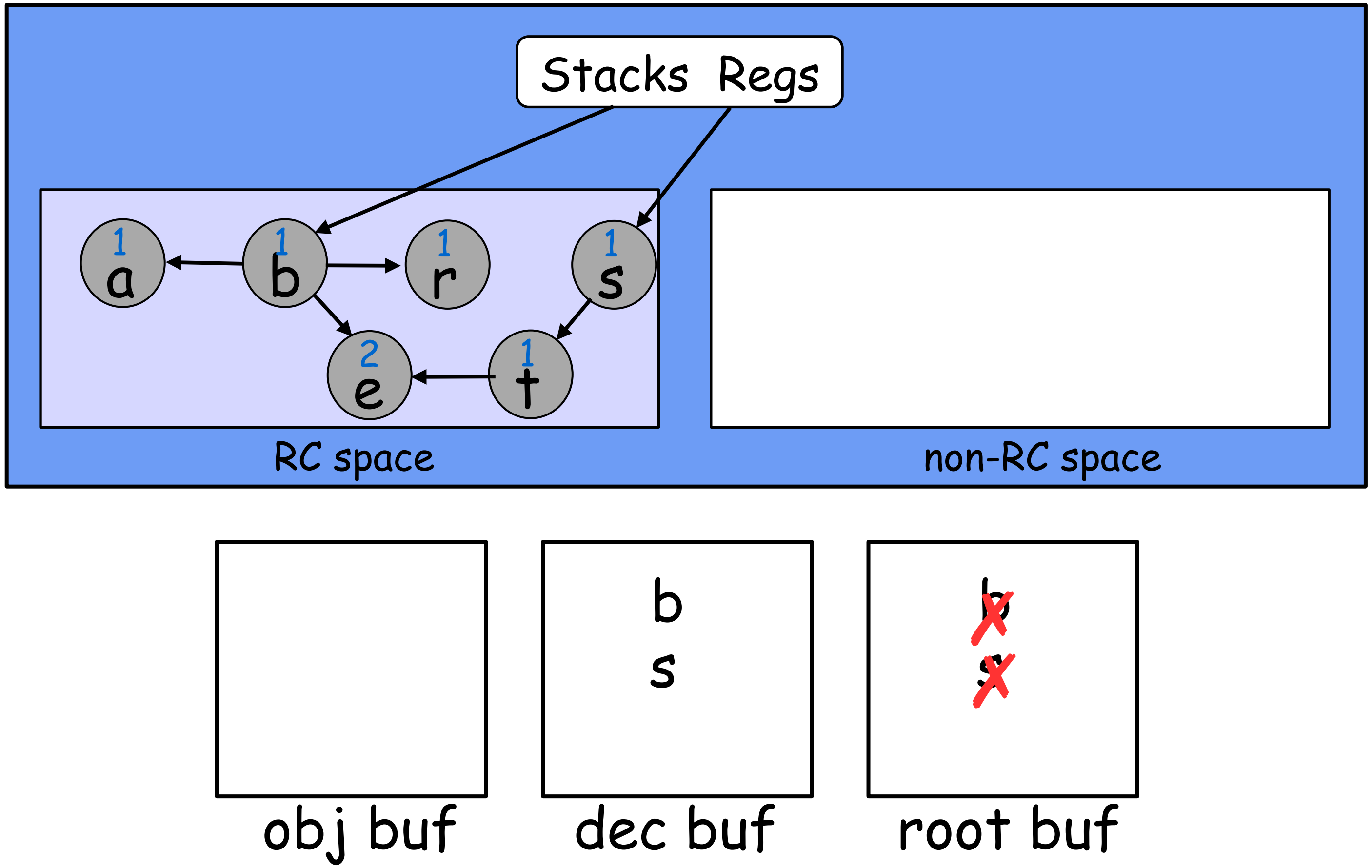
Ulterior Reference Counting

- RC Collection: Process Decrement Buffer



Ulterior Reference Counting

- Collection Complete.



Ulterior Reference Counting

- Controlling Pause Times: nursery collection & reference counting times
  - Modest bounded nursery size
  - Limit the growth of meta data
    - Decrement and modified object buffers
    - Trigger a collection if too big
  - RC time cap
    - Limit time recursively decrementing RC obj & in cycle detection
- Cycle detection
    - Use Bacon/Rajan trial deletion algorithm
    - Add a trigger to invoke cycle detection

Evaluation

- Jikes RVM and JMTK
- 4 Collectors
  - MS, RC, BG-MS, BG-RC
- Benchmarks
  - SPEC JVM & pseudojbb
- Collection triggers
  - Each 4MB of allocation for BG-RC (1MB for RC)
  - Time cap of 60ms
  - Cycle detection at 512KB

Throughput/Pause time

benchmark	heap used MB	BG-MS time sec	MS		BG-MS		BG-RC		RC	
			norm time	max pause	norm time	max pause	norm time	max pause	norm time	max pause
_202_jess	24	6.2	1.91	182	1.00	181	0.99	44	*2.36*	*131*
_213_javac	68	13.4	1.01	268	1.00	285	1.00	68	*1.78*	*580*
_228_jack	21	7.7	1.52	184	1.00	185	0.94	44	1.66	72
_205_raytrace	27	7.5	1.31	203	1.00	184	1.03	49	*1.71*	*133*
_227_mtrt	32	8.3	1.29	241	1.00	180	1.04	49	*1.75*	*130*
_201_compress	25	11.6	0.98	160	1.00	175	0.88	68	0.93	72
pseudojbb	74	20.0	1.00	264	1.00	281	1.00	53	*1.33*	*297*
_209_db	30	19.2	1.01	238	1.00	244	1.01	59	1.11	43
_222_mpegaudio	18	10.3	1.05	185	1.00	178	0.96	43	*1.14*	*121*
mean	**35**	**11.3**	**1.23**	**214**	**1.00**	**210**	**0.98**	**53**	**1.53**	**175**
geometric mean	31	10.4	1.20	211	1.00	206	0.98	52	1.47	130

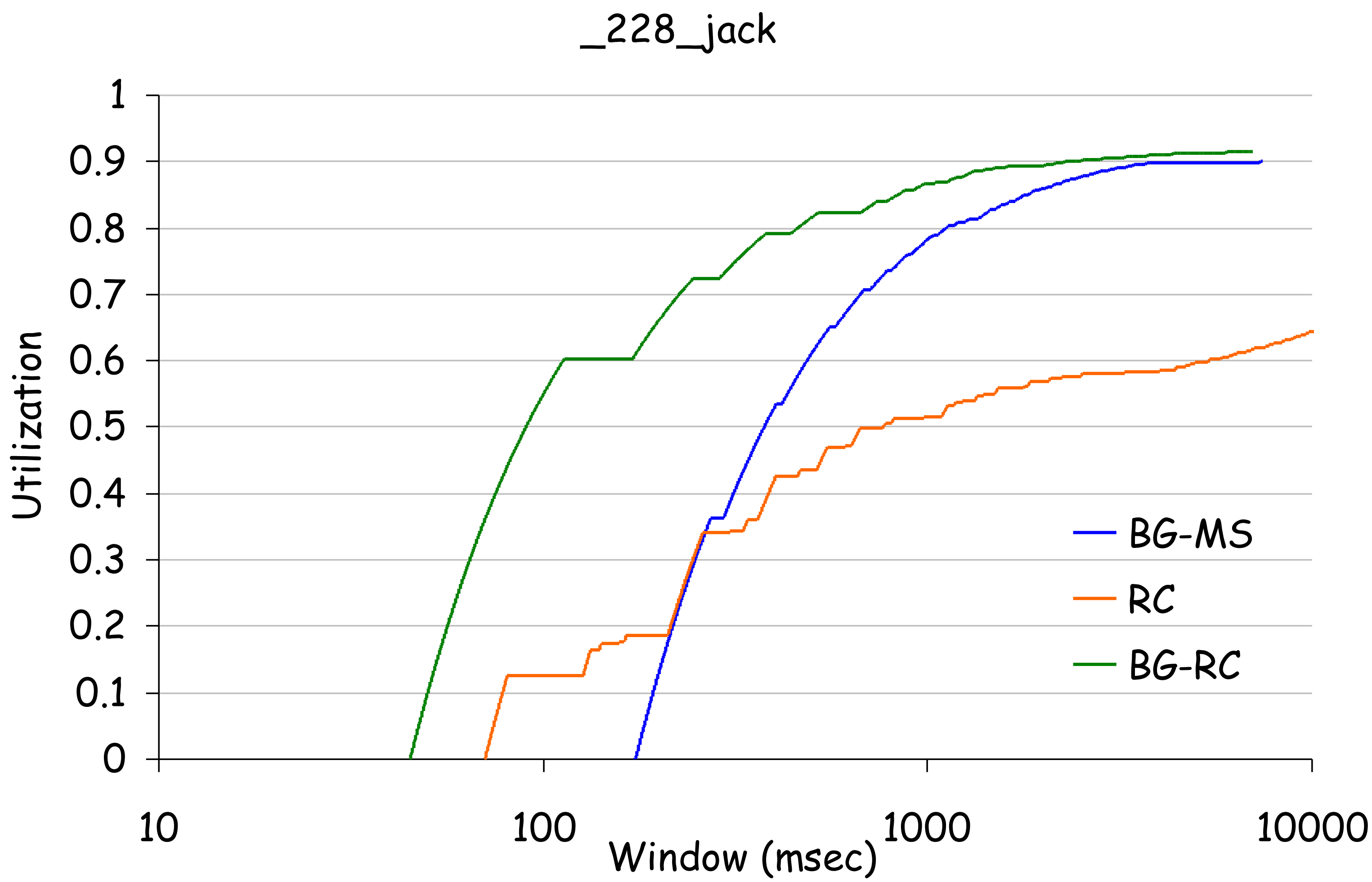
Table 3: Throughput and Responsiveness of MS, BG-MS, BG-RC, and RC at a Moderate Heap Size

Throughput/Pause time

benchmark	heap used MB	BG-MS time sec	MS norm time	MS max pause	BG-MS norm time	BG-MS max pause	BG-RC norm time	BG-RC max pause	RC norm time	RC max pause
_202_jess	24	6.2	1.91	182	1.00	181	0.99	44	*2.36*	*131*
_213_javac	68	13.4	1.01	268	1.00	285	1.00	68	*1.78*	*580*
_228_jack	21	7.7	1.52	184	1.00	185	0.94	44	1.66	72
_205_raytrace	27	7.5	1.31	203	1.00	184	1.03	49	*1.71*	*133*
_227_mtrt	32	8.3	1.29	241	1.00	180	1.04	49	*1.75*	*130*
_201_compress	25	11.6	0.98	160	1.00	175	0.88	68	0.93	72
pseudojbb	74	20.0	1.00	264	1.00	281	1.00	53	*1.33*	*297*
_209_db	30	19.2	1.01	238	1.00	244	1.01	59	1.11	43
_222_mpegaudio	18	10.3	1.05	185	1.00	178	0.96	43	*1.14*	*121*
mean	**35**	**11.3**	**1.23**	**214**	**1.00**	**210**	**0.98**	**53**	**1.53**	**175**
geometric mean	31	10.4	1.20	211	1.00	206	0.98	52	1.47	130

Table 3: Throughput and Responsiveness of MS, BG-MS, BG-RC, and RC at a Moderate Heap Size

Throughput & Responsiveness



Conclusion

- Match allocation and collection policies to the behaviors of older and younger object demographics
- Extend deferral to select heap objects
- Achieve good throughput performance and good responsiveness