

---

**“A unified theory of garbage collection,” Bacon, Cheng, and  
Rajan, OOPSLA '04**

Presented by Donald Nguyen

May 4, 2009

# The Science of Garbage Collection

---

- Algorithms:

- ◆ Baker's Real-time, Generational, Older-first, Beltway, Semi-space, Mark-sweep, Mark-sweep-compact, Mark-copy, Immix, Deferred Reference Counting, Ulterior Reference Counting, Cycle detection, Concurrent mark-sweep, Metronome, Boehm collector, Hoard

- Themes:

- ◆ Tracing, Reference Counting, Generational hypothesis, increments, belts, concurrent collectors, real-time collectors, compaction, cycle detection, correctness, memory leaks, static analysis for garbage collection, locality

- And...

# Axis: Reference Counting and Tracing

- Naive reference counting and tracing are qualitatively different

	<b>Tracing</b>	<b>Reference Counting</b>
Collection style	Batch	Incremental
Cost per mutation	None	High
Throughput	High	Low
Pause times	Long	Short
Real time?	No	Yes
Collects cycles?	Yes	No

- But, we want the best of both worlds!!!!1111!
  - ◆ Why?
  - ◆ How?

# Synthesis 1: Duality of RC and Tracing

- Consider instead generalizations of the algorithms:
  - ◆ Reference Counting+: decrements are batched instead of applied immediately
  - ◆ Tracing+: reconstruct reference counts instead of marked bits
- RC increments count whenever a pointer is installed; begins collection with a list of subtractions to apply
- Tracing starts with count of zero and begins collection with a list of additions to apply (the rootset)
- RC and Tracing are duals:
  - ◆ RC over dead data; Tracing over live data
  - ◆ For RC, initial reference count (at collection time) is an over-approximation; for Tracing, an under-approximation
  - ◆ Over-approximation implies incremental RC

# RC versus Tracing

```
scan-by-tracing( $W$ ):
```

```
  while  $W \neq \emptyset$ :
```

```
     $w \leftarrow W.remove()$ 
```

```
     $\rho(w) \leftarrow \rho(w) + 1$ 
```

```
    if  $\rho(w) = 1$ :
```

```
       $W \leftarrow W \cup [v : (w, v) \in E]$ 
```

```
sweep-for-tracing():
```

```
  for each  $v \in V$ :
```

```
    if  $\rho(v) = 0$ :
```

```
       $V\_F \leftarrow V\_F \cup v$ 
```

```
       $\rho(v) \leftarrow 0$ 
```

```
new( $x$ ):
```

```
   $\rho(x) \leftarrow 0$ 
```

```
scan-by-counting( $W$ ):
```

```
  while  $W \neq \emptyset$ :
```

```
     $w \leftarrow W.remove()$ 
```

```
     $\rho(w) \leftarrow \rho(w) - 1$ 
```

```
    if  $\rho(w) = 0$ 
```

```
       $W \leftarrow W \cup [v : (w, v) \in E]$ 
```

```
sweep-for-counting():
```

```
  for each  $v \in V$ :
```

```
    if  $\rho(v) = 0$ 
```

```
       $V\_F \leftarrow V\_F \cup v$ 
```

```
new( $x$ ):
```

```
   $\rho(x) \leftarrow 0$ 
```

## RC versus Tracing (continued)

$\text{dec}(x) :$

$$W \leftarrow W \cup [x]$$

$\text{inc}(x) :$

$$\rho(x) \leftarrow \rho(x) + 1$$

$\text{assign}(a, p) :$

$$l \leftarrow \langle a \rangle$$

$$\langle a \rangle \leftarrow p$$

$\text{dec}(l)$

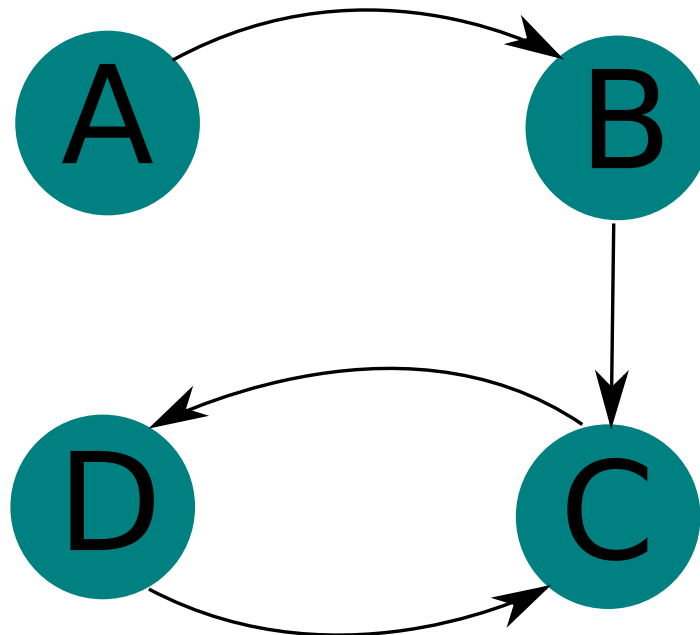
$\text{inc}(p)$

# Big Picture 1: Fixpoints

- RC and Tracing compute the greatest and least fixpoint, respectively, of the following:

$$\rho(x) = |[x : x \in R]| + |[ (w, x) : (w, x) \in E \wedge \rho(w) > 0 ]| \quad (1)$$

$$V_F = [v \in V : \rho(v) = 0] \quad (2)$$



## Synthesis 2: Hybridization

- Many algorithms are a hybrid of RC and Tracing applied over (potentially logical) segments of the heap
  - ◆ Remembered sets, write barriers, and “train”-style collectors are types of reference counting
    - Unidirectional write barriers are bidirectional reference counts where the reverse edge represents an aggregate “may reference” relationship
  - ◆ Cycle detection and backup tracing are a form of segmentation
    - If you can partition objects into a segment such that the incoming count to the segment is zero, you can collect it independently
- RC gives incremental collection with potentially high overhead and the inability to collect cycles
- Tracing gives high-throughput at the cost of incremental collection

# Big Picture 2: Hybrids

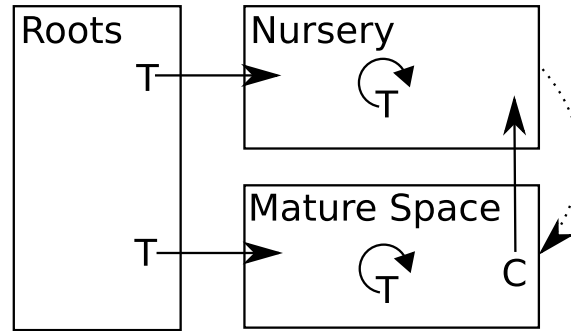


Figure 1: Generational Collector

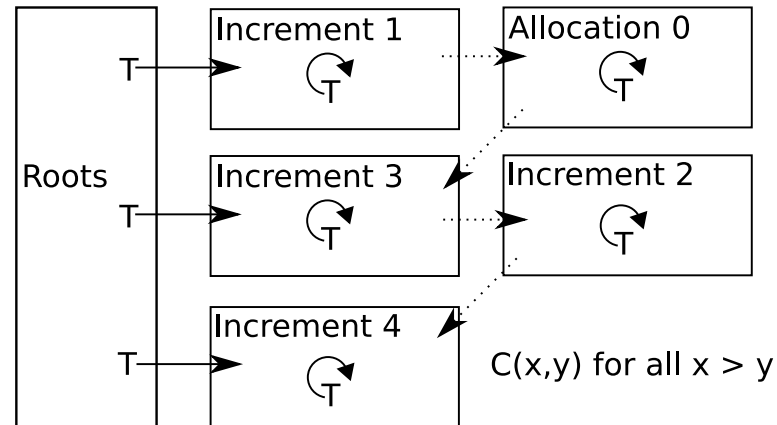


Figure 2: Beltway Collector

# Synthesis 3: Three Big Design Questions

1. How to segment memory?
  - Pros and cons of different strategies?
2. How to traverse each partition?
  - Ditto.
3. When to apply different time-space trade-offs? E.g., semi-space versus sliding compaction, pointer reversal versus stack traversal.
  - What other time-space mechanisms are there?

# Questions

---

- What ideas do not fit in this abstraction?
- What other design points would be interesting to consider?
- The paper goes into examples of collection algorithms outside the context of memory management (i.e., log-structured file systems, partial tracing). What other examples are there and how do those contexts differ from that of memory management?