Lecture 15 (part 2)

Advanced Garbage Collection

I. Break Up GC in Time (Incremental)
II. Break Up GC in Space (Partial)

Readings: Ch. 7.4 - 7.7.4

Incremental GC (in time)

- Garbage collection happens in multiple rounds
- Collect everything at the end after tracing all the reachable objects is done
- \( R \): reachable objects at the beginning of the first round of GC
- \( \text{New} \): Objects created since the beginning of the first round of GC
- \( \text{Lost} \): Objects that lose reachability since the beginning of the first round of GC

\[
\text{Ideal} = (R \cup \text{New}) - \text{Lost} \subseteq \text{Answer} \subseteq (R \cup \text{New})
\]
What We Need to Intercept When the Mutator Runs

- **Error:** A reachable object o classified as unreachable
  - the only pointer to an unreached object is in a scanned object
- **How it can happen:**
  - At the beginning of the mutator run
    - o must be in the unreached set (not scanned or unscanned) (C)
    - A pointer p in an unscanned or unreached object (B) points to o.
    - **Read Barrier:** remember loads of pointers from objects in B pointing at objects in C
  - At the end of the mutator run
    - p has been copied to a scanned object (A)
    - **Write Barrier:** remember stores of pointers into objects in A pointing at objects in C
    - p is no longer in the unscanned/unreached set (B)
    - **Overwrite Barrier:** remember values overwritten in B pointing to objects in C

---

**Generational GC**

- **ith generation**
  - Stable set: Partitions j ( j > i)
  - Target set: Partitions k (k <= i)
  - new root set
    - original root set + all pointers from the stable set
### Generational Garbage Collection

Partitions

<table>
<thead>
<tr>
<th>4</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1 is full</td>
<td>GC 1</td>
<td>1 is full</td>
<td>GC 1</td>
<td>1 is full</td>
<td>GC 1</td>
<td>2 is full</td>
</tr>
</tbody>
</table>

#### Summary

<table>
<thead>
<tr>
<th>Topic</th>
<th>Abstraction</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data flow optimizations</td>
<td>Graphs, Recurrent equations, Fixed-point</td>
<td>High-level programming without loss of efficiency</td>
</tr>
<tr>
<td>Parallelism and locality</td>
<td>Integer linear programming, Linear algebra</td>
<td>Hide parallelism and locality from programmers</td>
</tr>
<tr>
<td>optimizations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pointer alias analysis</td>
<td>Program Query Language Logic database (Datalog), Binary decision diagrams (BDDs)</td>
<td>Automate error-prone security inspection. Illustrates language abstraction</td>
</tr>
<tr>
<td>Natural language programming</td>
<td>Neural networks</td>
<td>Consumers can automate personal and professional tasks themselves, eliminating dependence on coders.</td>
</tr>
</tbody>
</table>