**Concurrent copying and compaction GC**

Presented by Hanan Rofe Haim
6/1/2015

**Chapter summary:**

The first part of the chapter discusses and presents different solutions for handling concurrent copying.

Every algorithm in the chapter has the main goal of garbage collecting using copying while protecting the mutator and the collector from each other.

Each algorithm is based upon one of two invariants: the To-space and the From-space invariants.

The first algorithms discussed are based on the To-space invariant which is easier to maintain while correction and termination are also guaranteed.

Most algorithms in the chapter has a performance issue due to the fact they all have a stop-the-world phase.
The last part of the chapter discusses two lock-free algorithms which never stop all the mutators.

The second part of the chapter revisits compaction algorithms from earlier chapters and describes how these algorithms implement concurrent compaction.

 The last algorithm in the chapter is a Pauseless compaction algorithm which is composed of fully concurrent and parallel phases.

**Original contribution:**

Firstly, before presenting the copying algorithms from the book I've created two animated examples which help better understand the problem we're facing and what it'd take to solve it.

Those examples are then revisited after the class have seen concurrent copying algorithms, to make sure the problems addressed at the start of the presentation are in fact solved.

Secondly, I've created an animated running example of the Compressor algorithm, which I found much easier to understand than separate images as in the book.

**Discussion:**

We discussed the main issue copying algorithms have: they copy.

Several solutions were brought up until finally someone recalled baker's treadmill algorithm (presented by Hadar Getzel) which remove the need to copy.

We also discussed how frequent use of virtual memory techniques can affect performance and came to the conclusion that those operations should be batched together to minimize the overhead.