**Runtime interface and language specific concerns**

Presented by Oleg Dobkin  
2/12/2014

Summary of chapters 11 and 12:

Chapter 11 talks about tasks that runtime/language developer needs to face if he wishes to implement a GC. Among the discussed topics are various methods of finding pointers (conservative and accurate), handling external (unmanaged) code, GC-safe points and write barriers.

The chapter goes into a lot of low-level details, that are required for efficient and correct implementation of garbage collection.

Chapter 12 talks about language interface, in particular about finalization – method that allows the programmer to release non-managed resources using the help of GC, and weak references, as opposed to the regular strong references used until now, that are helpful in some cases.

Contribution (things that weren't present in the book):

Slide 17: an example of stack structure, that shows stack frame of a function

Slide 20: an example of Thumb encoding of B opcode, that shows a highly convoluted embedding of a constant value inside of an opcode

Slide 78: smart pointers in C++ (Boost/C++11)

Slide 79-80: discussion and conclusions

Discussion: several issues were brought up during the talk:

1. Definition of runtime. The definition was missing from the presentation on the assumption that everyone knew at least at some level what runtime is.

2. Bit stealing slide - why SPARC implementation has special encoding for non-integer primitive values: processor support for frequent integer-based opcodes.

3. Finalizers vs destructors and their connection.

4. Discussion slide - another use of garbage collector: leak detection.