Tips on correctness and debugging

- Getting to the point you can run your GC for the **first** time, all the way through, is relatively *easy*.

- The hard part is getting through the **second** GC: subtle bugs in the first GC will disturb the heap in a way that might only be detected by the second collection, and by then so much has happened it will be very hard, or impossible to connect cause with effect.

- What is needed is a way to have confidence that the first GC was correct.

Express and check invariants

- There are many invariants associated with a VM's internal components. Examples:
  - All object references in the heap and VM variables and structures point to the start of a valid object.
  - Every new-to-old space pointer has a corresponding entry in the remembered set.

Invariant checks

- It is useful to codify these invariants in functions within the VM, both to capture the invariants and provide an easy means to check them. Example, from Self:

```
inline bool oopClass::verify_oop(bool expectErrorObj) {
  fint t = tag();
  if (t == Mem_Tag) {
    return memOop(this)->verify_oop(expectErrorObj);
  } else if (t == Mark_Tag) {
    error1("oop 0x%x is a mark", this);
    return false;
  } else {
    return true;
  }
}
```

Invoking the verifiers

- Having coded up verification, it is useful to have them invokable before and after every major state change. Example:
  - Verify the heap before and after a GC
- This should be controlled by a configuration option (e.g., command-line flag, configuration file, or program-callable switch). It can also be useful to call such functions directly from within a debugger.

Forcing errors

- Another useful trick is to force some complex operation — such as GC — to happen at every possible opportunity, in an attempt to force a bug to occur.

- To do this, add a configuration option such as GCALot (see HotSpot and Self for examples) and then add code to force a GC at, say, each allocation when this option is enabled.