I found my notes and thought I would type in a summary of a short talk I gave at ISMM on the topic of "Duplication". I'm sure I didn't say exactly these words, but I think this writeup is reasonably close.

--Guy

Duplication
Guy L. Steele Jr.
ISMM 2009 Wild and Crazy Ideas session

The notion of "value objects" or "immutable objects" (I dislike the term "immutable" because it is so often misheard as "mutable") often crops up in object-oriented and especially in functional programming. There are at least three neat tricks you can do with value objects in the literature.

(a) You can "intern" or "hash cons" them; that is, if you discover two value objects with the same fields, you can substitute one for the other, perhaps in such a systematic manner that eventually you can discard one of them.

(b) If you avoid the mistake of allowing what would otherwise be a value object nevertheless to enjoy the property of object identity (that is, pointer identity), then you can freely copy value objects; for example, if you represent a complex number as a value object, then a compiler can just pass the real and imaginary parts in registers or on the stack, rather than requiring a heap object.

(c) When you build a concurrent copying garbage collector, you need locks or some other synchronization mechanism to prevent multiple GC threads from copying the same object. This synchronization can impose substantial overhead. This is too bad, because measurements show that, on at least some likelihood of collision is low, but you have to pay the synchronization overhead for every object anyway, just in case. A recent observation is that, because it is okay to make copies of value objects, you could have GC threads simply not bother to synchronize when making copies of value objects, and accept the risk of duplication. For those same plausible workloads, the number of duplicated objects should be acceptably small.

This suggests to me that there may be other reasons why you might want to allow either accidental or intentional duplication of value objects. Think of this as the inverse of hash consing.

First of all, on a very-large-scale parallel system with, say, thousands or millions of threads, a popular object might become a memory hot spot. Making duplicates of popular value objects could relieve these hot spots. Now, modern cache systems, such a snoopy caches or other distributed hardware cache copies of value objects by allowing multiple readers to keep locally cached copies. But distributing these copies dynamically on demand creates intercache traffic, placing a load on the memory interconnect network. If software can anticipate these loads by detecting popular objects and creating copies within the main memory, it might reduce the dynamic load on the interconnect. (In fact, this is in some sense what a company like Akamai does for the Internet: trying to anticipate the need for copies of data that is unchanging or slowly changing.)

Now, I recognize that not all data structures used in large-scale computing will be value objects. Some computations may just naturally be easier or more efficient if they use large mutable arrays, Fortran-style. But even there, note that each such array typically has unchanging descriptor information associated with it, indicating the lengths of the axes, or lower each subscript. Even if the computation is organized so that different threads access different parts of the mutable array, the descriptor information is likely to be a very popular shared value object and probably will need to be duplicated to avoid its being a memory hot spot.

Finally, I note that some designs for generational garbage collectors use remembered sets, and popular objects can cause problems because the remembered sets get too large. Maybe this can be alleviated by intentionally creating copies of popular value objects so that no one object is too popular. Indeed, remembered sets might be a good mechanism in general for deciding which value objects should be duplicated.

And that is why this idea is relevant to ISMM: in the future, on large machines, it may be an important task of a memory manager not only to discard unneeded objects and perhaps to merge identical value objects, but also to intentionally.

This may be a fruitful area for research.