SYSTEMS ENGINEERING · WEEK 9 · STATION III

Runtime Systems

Malloc, garbage collection. Liveness is undecidable, so collectors compute reachability instead.

MALLOC

Selfie's `malloc` is a bump pointer. Freeing memory makes everything harder.

```
// emit_malloc, in spirit
uint64_t* malloc(uint64_t size) {
uint64_t* block = _bump; // read
_bump = _bump + size; // modify, write
if (_bump > brk) brk(_bump); // ask the kernel for pages
return block;
}
```

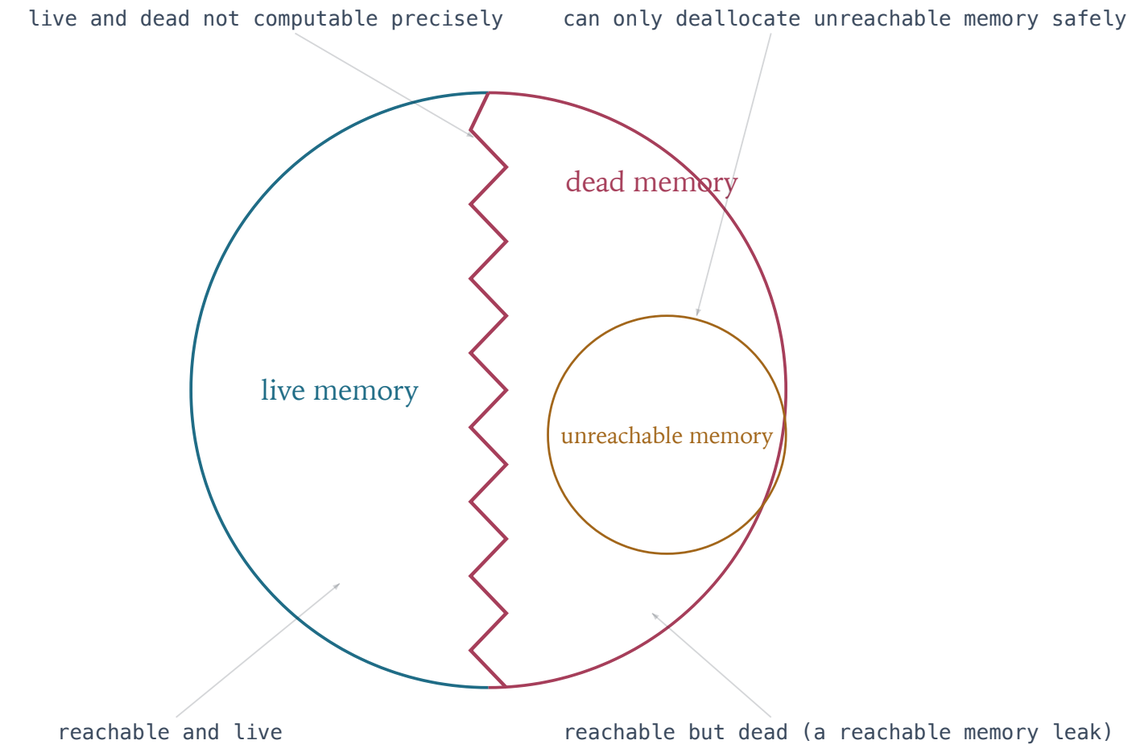
A global pointer that moves up. Allocation is constant time, memory is never reused, and the heap grows until the kernel refuses. Selfie's own compiler runs on this and never frees a byte; for a compiler that is fine.

Add `free` and you need a free list, a fit policy, coalescing against fragmentation, and metadata per block: the book's tour from first-fit to slab allocators. Every design is a different notation for the same meaning, a function from requests to disjoint blocks, with a different cost profile.

And the read-modify-write of `_bump` is last week's race. Two threads can receive the same block. That is the assignment.

THE THEOREM

Will this block be used again? Undecidable. Can it be reached? Decidable.



live memory may be used again; dead memory will not be; unreachable memory cannot be

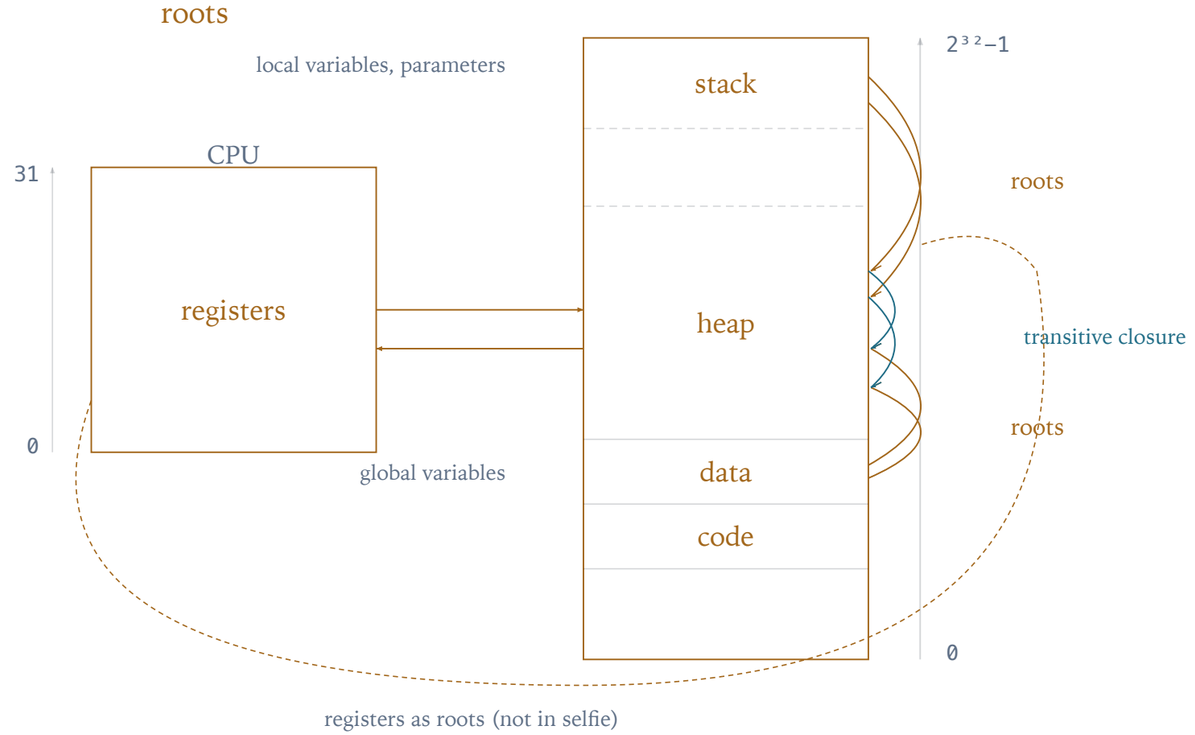
A block is live if the program will use it again and dead if it will not. Whether a program will ever use a given address is a property of its future behaviour: Rice, exactly. No collector decides liveness.

A block is reachable if some chain of pointers leads to it from a variable. Reachability is a property of the current state, computed by a graph traversal in time linear in the heap. Unreachable implies dead; the converse fails, and the difference is the memory a collector keeps although it is dead.

So every garbage collector computes the decidable over-approximation of the undecidable property. Safe, never precise. The design principle of the class again: bound what you cannot decide.

ROOTS AND TRACING

Start from the variables. Follow every word that looks like a pointer.



roots in the data and stack segments; the heap reached from them

The roots are the global variables in the data segment and the locals and parameters on every stack. Mark everything reachable from them, transitively; sweep everything unmarked. Mark-and-sweep, and selfie's collector does this on the emulator's view of the guest's memory, or as a library inside the program, or both at once.

Which words are pointers? C* does not say; a word is a number. A conservative collector treats every word that could be a heap address as if it were one: an over-approximation of an over-approximation, still safe, still a bound. Boehm's collector made that fast; selfie's makes it readable.

Self-collecting selfie: the emulator's collector collects the memory of a guest that is running its own collector on the memory of its guest. Try –gc at two levels and read the profile.

COST

Time, space, and the pause.

STOP-THE-WORLD

Mark-and-sweep stops the program for a time linear in the heap. Fine for a compiler; not for a game, a pacemaker, or a kernel. Every collector since is a way to spread that time out.

GENERATIONAL, INCREMENTAL, CONCURRENT

Most objects die young: collect the young ones often. Do a little marking per allocation. Mark while the program runs, with a barrier that tells the collector what changed. Each is a bound traded for a cost.

REFERENCE COUNTING

Count pointers instead of tracing; free at zero. No pause, but cycles are never freed: an under-approximation of unreachability. Safe in the other direction: leaks, never dangling pointers.

Managed languages are languages whose runtime does this for you, and the price is paid on every allocation. The Cost chapter's trade, once more.

ASSIGNMENT

threadsafe-malloc: add lr.d and sc.d, then use them.

1. Add `lr.d` and `sc.d` to RISC-U: encoding and decoding in selfie, a case each in mipster's `execute`, and the reservation, cleared by any store to the address, in the machine context. Make hypster honour it too.
2. Provide them to C* as procedures `lr()` and `sc()`, emitted like the other library procedures.
3. Rewrite `emit_malloc` so that the read-modify-write of `_bump` is a retry loop on `lr` and `sc`. Test with last week's threads: many threads, many allocations, no two blocks overlapping.
4. `./grader/self.py threadsafe-malloc`.

NEXT WEEK

The last systems assignment, `treiber-stack`, builds a lock-free stack on the same two instructions. And `rotor-bounds` turns a model checker on your sequential systems code, because rotor does not yet model threads. Both are introduced in week 10.