10-1

Memory — Static + Dynamic

Static — an known position in program text to call free (e.g. when we're done with arg or fn)

Dynamic — not static, the free position is a property of evaluation.

"Stack Memory" an implementation *static memory

C's local vars

"Heap memory" is dynamic memory

CEK = static = { arg, fn } dyn = { clo, E[X ← V] }

Strategy for you in C:

- I make mistakes w/ off-by-1 (sometimes reference undefined memory)
- Input determines your allocation pattern
- Allocation is determined by a computation
  ↳ nervous about aliases (other pointers to the values)
- It's a global decision

Errors

Soundness - Freeing too early, may cause reuse, may cause inconsistent use => crash

Completeness - Free too late => use too much memory

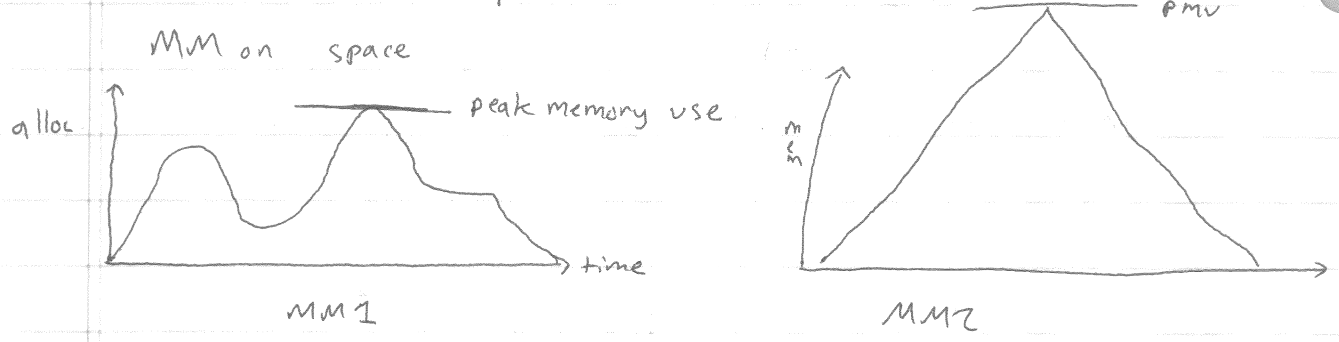
Soundness — never free too early

```
0. int * x = malloc( sz );
1.      x[2] = 28;
2. f(10);   ←——————— what if f never returns?
3. return x[2];
```

10-2

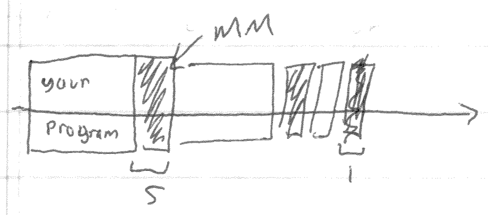
Memory Management:
- Necessary to be sound
- As close to complete and little impact on time as possible

MM on space



imagine MM_m where $pmu(MM_m) \leq pmu(MM_x) \quad \forall MM_x$

MM on time



minimize
total time = your + mm
minimize just mm
minimize % in mm
minimize the maximum mm slice
maximize the mini your slice

Common C MM
- Insert calls to free() when you think it's right
  ↳ if you're wrong, then ¬ sound
- Put in lots of copies to remove aliases
  ↳ spend more space (i.e. less complete)
- Doug Lea alloc - runs in lg n where n are the # of objects
  free - lg n
  space - at worst 2x space

"Garbage Collector"