Memory management issues:

1. Memory leak: Memory allocated but no references to it. You can’t deallocate it.
2. Dangling pointers: An active reference to memory that has been deallocated (active is something that exists in scope for your code)
3. Stack smashing: Overwrite a stack frame
4. Illegal access: Program tries to access memory not reserved for that program

Dangling pointer: We have a pointer to deallocated memory

Solutions:

1. Tombstones: Every pointer access goes through a tombstone. There is a middleman pointer between pointer and memory. This middleman is called the tombstone. When you deallocate the memory, the tombstone value changes to indicate deallocation.  
    Pointer 🡪 Memory Pointer 🡪 Tombstone 🡪 Memory   
   Downside is that tombstones live forever. Lots of memory allocations use up your memory. 2 steps to access memory.
2. Lock and key:
   1. Every address of allocated memory has a “lock” value. Every pointer has a “key” value.
   2. When accessing memory, the lock and key values must match
   3. On deallocation the lock value is changed to indicated “invalid”
   4. On reallocation a new lock value is used

Key | Pointer 🡪 Lock | Memory

Downsides: Requires a little more time for memory access. # of allocation is limited or lock values have to cycle. When lock gets big, uses extra memory. Causes memory fragmentation.

Tombstones can work with stack allocated objects

Lock and key can’t. It is heap only.