Priority Inversion

- spin locks can interfere with schedulers which exclusively preference higher-priority jobs over lower-priority ones
  - e.g. MLFQ without boosting
- scenario
  - higher-priority thread T2 blocks waiting for I/O, so lower-priority T1 runs
  - T1 acquires a spin lock
  - T2's I/O completes, so the scheduler switches back to it
  - T2 tries to acquire the lock and spins
  - ...and things are stuck
    - T2 is actively running, so it is always scheduled over T1
    - T1 never gets a chance to run and release the lock, so T2 keeps spinning

Priority Inversion

- priority inversion can also occur with other kinds of locks
  - scenario
    - lowest-priority thread T1 acquires lock
    - highest-priority thread T3 blocks trying to acquire the lock
    - middle-priority thread T2 is the highest-priority runnable thread and is scheduled instead of T1
- can be addressed through *priority inheritance*
  - allow a higher-priority thread waiting for a lower-priority thread to temporarily boost the lower-priority thread's priority

Two-Phase Locks

- phase 1: spin in the hopes of obtaining the lock quickly
- phase 2: if lock is not acquired, the thread sleeps
- avoids overhead of queue management in the (hopefully common) situation of locks not being held for very long