# Multiple Reader Single Writer Solutions

## Multiple Reader, Single Writer

- What is meant by "Multiple Reader, Single Writer"?
  - A situation where many threads access shared data, but only a few threads modify it
- Give some examples of applications which use this pattern
  - Financial data feed for infrequently traded stocks
  - Audio/video buffers in multimedia players

## Multiple Reader, Single Writer

- What issues are there with "Multiple Reader, Single Writer"?
  - The shared data must be protected against data races
  - Threads which access the shared data must be synchronized
  - This could be done by making the threads lock an std::mutex
  - Each thread would have exclusive access to the data
  - Other threads would have to wait for access
  - It is safe to have multiple threads making interleaved reads (provided there are no modifying threads which could conflict and cause a data race)
  - Giving every thread an exclusive lock causes an unnecessary drop in performance

## std::mutex Example

- Write a program which has two task functions
  - A "writer" task which modifies shared data
  - A "reader" task which accesses shared data but does not modify it
- Use an std::mutex to synchronize these tasks
  - The reader task should sleep for 100ms before unlocking the mutex
  - This is to simulate activity
- The program creates twenty reader threads, then two writer threads, then another twenty reader threads
- How long do you expect it will take the program to run?
- Explain the results

## std::mutex Example

- The threads run and try to lock the mutex
- The first thread locks the mutex executes its critical section
- While it has the lock, no other threads can acquire a lock
- In effect, the threads are forced to execute their critical sections sequentially
- The time taken will be approximately 100ms * 40 = 4 seconds