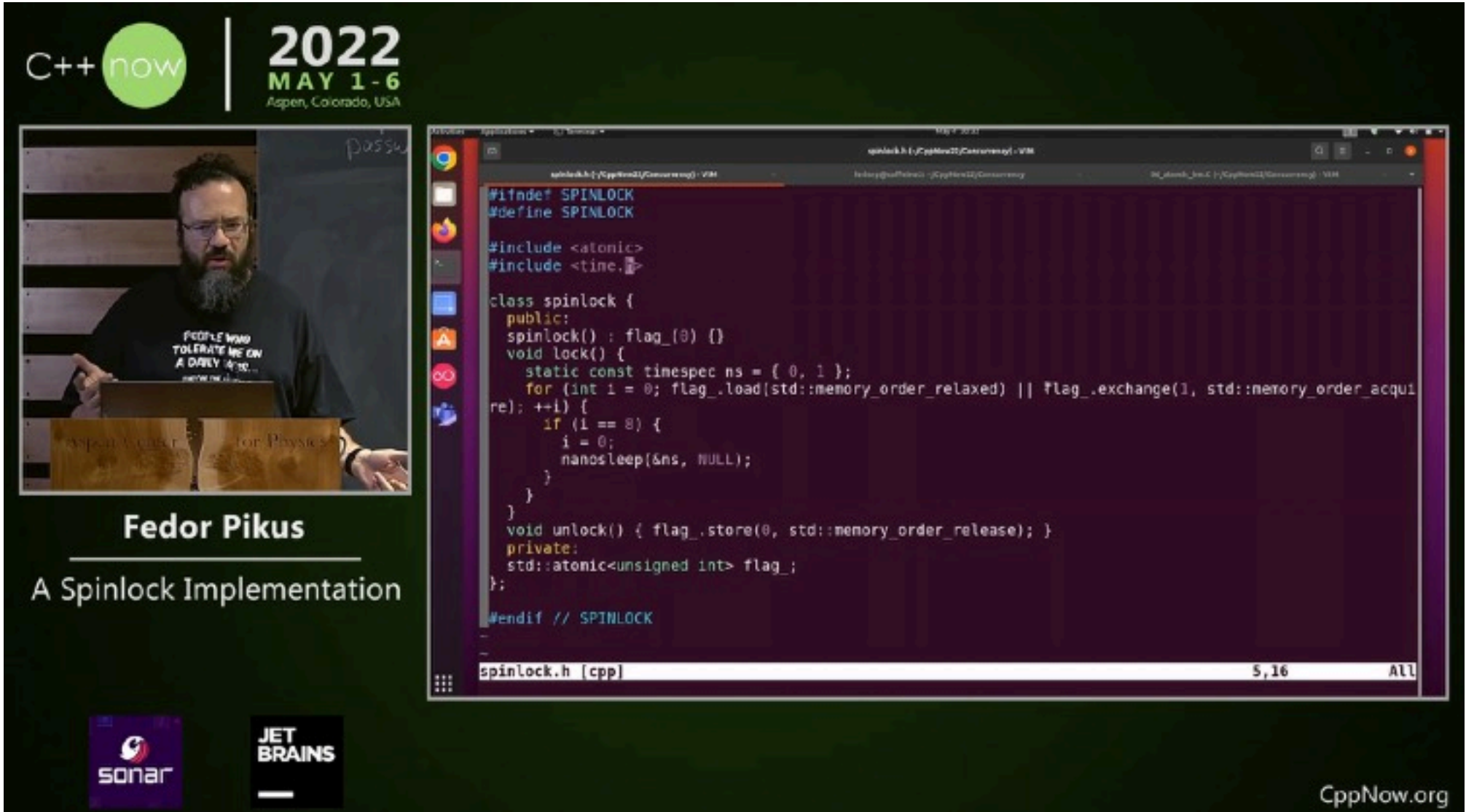
```
class realtime_mutex
{
    std::atomic_flag flag = ATOMIC_FLAG_INIT;

public:
     // Non-blocking try_lock for real-time contexts
     bool try_lock() noexcept
     {
          // First test: cheap read-only check
          if (flag.test (std::memory_order_relaxed))
               return false;  // Already locked

          // Then test-and-set if the first test passed
          return ! flag.test_and_set (std::memory_order_acquire);
     }

     // Blocking lock with test, test-and-set optimization
     void lock() noexcept
     {
          while (true)
          {
               // Optimistically assume the lock is free on the first try
               if (! flag.test_and_set (std::memory_order_acquire))
                    break;  // Successfully acquired the lock

               // Wait for lock to be released without generating cache misses
               while (flag.test (std::memory_order_relaxed))
                    CPU_PAUSE();  // CPU-specific pause instruction
          }
     }

     void unlock() noexcept
     {
          flag.clear (std::memory_order_release);
     }
};
```

https://rigtorp.se/spinlock/

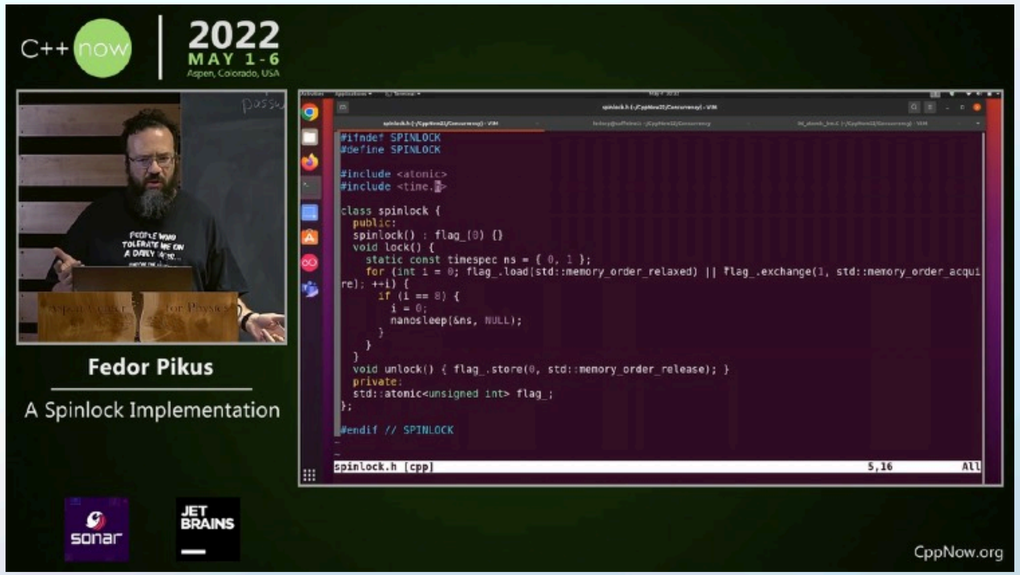
Lightning Talk: A Spinlock Implementation
- Fedor Pikus
- CppNow 2022



```cpp
class realtime_mutex
{
   std::atomic_flag flag = ATOMIC_FLAG_INIT;

public:
    // Non-blocking try_lock for real-time contexts
    bool try_lock() noexcept
    {
        // First test: cheap read-only check
        if (flag.test (std::memory_order_relaxed))
            return false;  // Already locked

        // Then test-and-set if the first test passed
        return ! flag.test_and_set (std::memory_order_acquire);
    }

    // Blocking lock with test, test-and-set optimization
    void lock() noexcept
    {
        while (true)
        {
            // Optimistically assume the lock is free on the first try
            if (! flag.test_and_set (std::memory_order_acquire))
                break;  // Successfully acquired the lock

            // Wait for lock to be released without generating cache misses
            while (flag.test (std::memory_order_relaxed))
                CPU_PAUSE();  // CPU-specific pause instruction
        }
    }

    void unlock() noexcept
    {
        flag.clear (std::memory_order_release);
    }
};
```

https://rigtorp.se/spinlock/

Lightning Talk: A Spinlock Implementation
- Fedor Pikus
- CppNow 2022



182,987,000

x4.5

41,400,000

20,178,300

x0.5