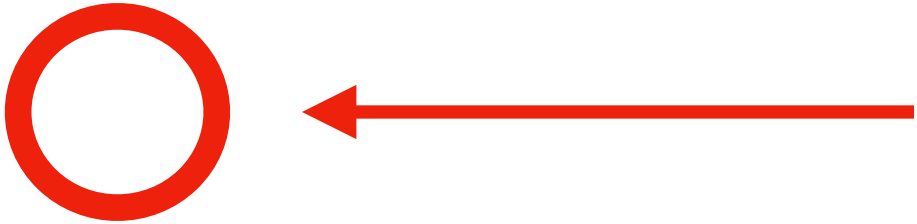
```
class spin_lock
{
public:
    void lock() noexcept          { while (flag.test_and_set()); }
    void unlock() noexcept        { flag.clear(); }
    bool try_lock() noexcept      { return ! flag.test_and_set(); }

private:
    std::atomic_flag flag = ATOMIC_FLAG_INIT;
};
```



Simply spins

Basic spin_lock

59

Basic spin_lock

```
class spin_lock
{
public:
    void lock() noexcept          { while (flag.test_and_set()); }
    void unlock() noexcept        { flag.clear(); }
    bool try_lock() noexcept      { return ! flag.test_and_set(); }

private:
    std::atomic_flag flag = ATOMIC_FLAG_INIT;
};
```

Simply spins

try_lock Summary

- Scenario:
  - Data is big: `std::atomic<>::is_always_lock_free == false`
  - Failure to acquire the resource is ok
- Trade-off:
  - Non-real-time thread waits on real-time thread for access to the resource
  - Real-time thread will have to fail gracefully
- Examples:
  - Passing large data to the real-time thread for exclusive use
  - Audio samples, wavetables, filter coefficients etc.