<https://www.youtube.com/watch?v=Q_0_1mKTlnY>

Java 5 has ReadWriteReentrantLock that provides pessimistic lock.

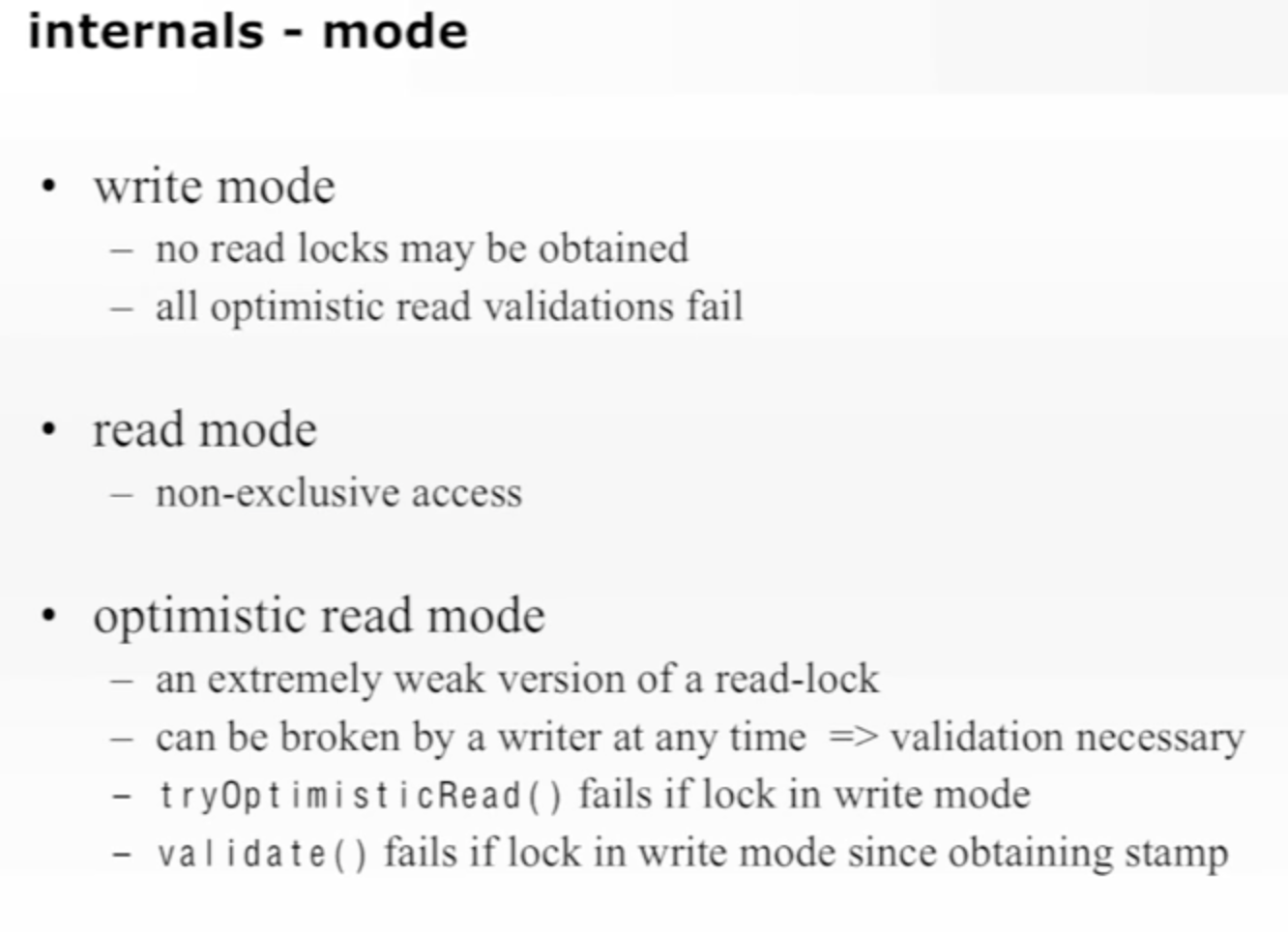
It means that once the write lock is applied, other threads asking for read/write locks have to wait till write-lock is released. Vice-a-versa is also true.

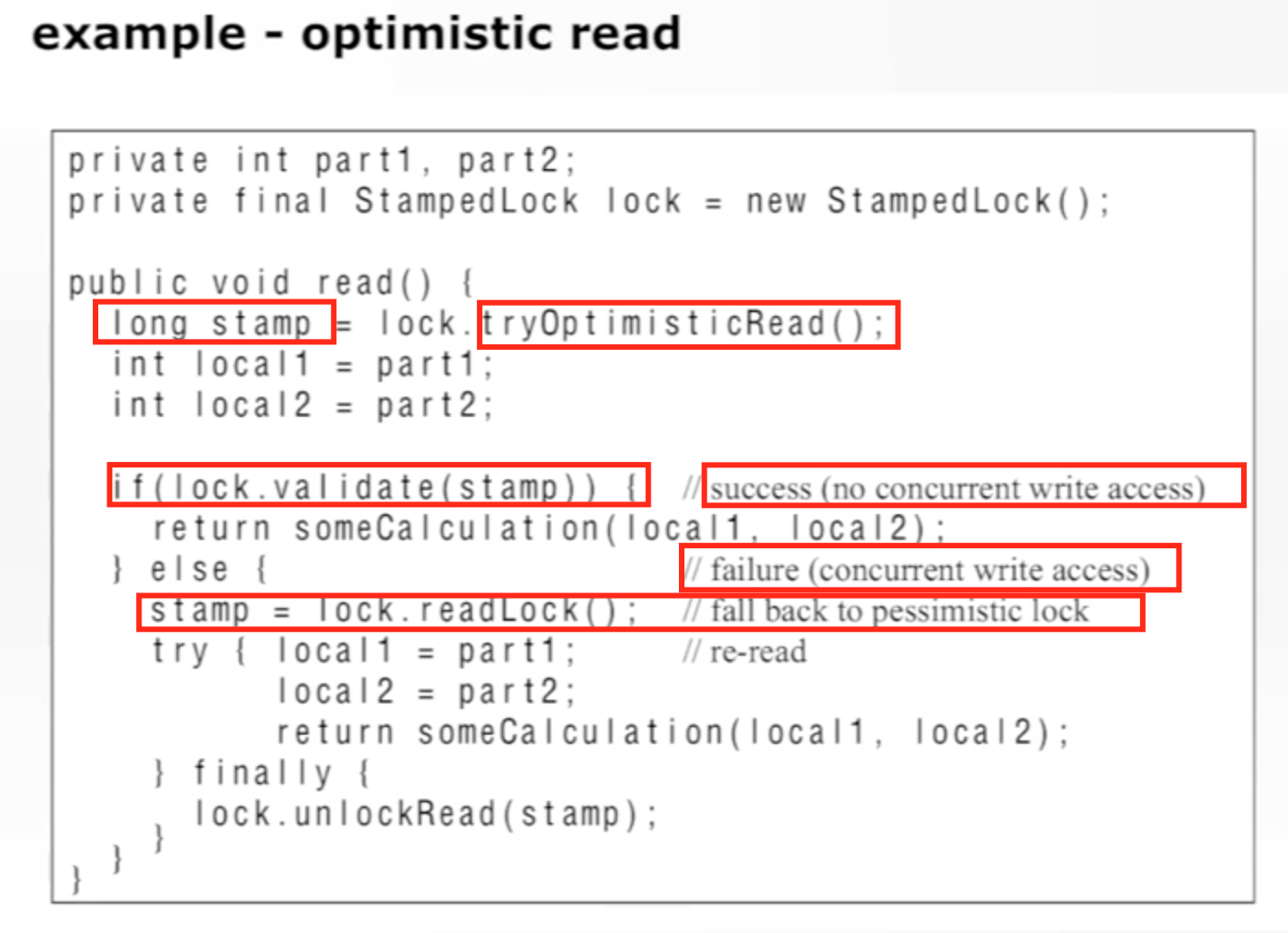
It allows multiple threads to acquire read locks at the same time, but that’s not true for the write-locks.

ReadWriteReentrantLock is very good, if you want to maintain synchronization in case where there are reads are more compared to writes.

Java 8 has come up with one more type of lock **StampedLock** lock that provides

* Version (Stamp)
* A mode

****

****

On **ReadWriteReentrantLock**, you can get Read lock or Write lock.

On **StampedLock**, you can get one additional type of lock **Optimistic Read Lock.**

Optimistic Read Lock doesn’t stop you from acquiring a read lock even though some other thread might have acquired a Write Lock.

It gives you a version(stamp) when you ask for an Optimistic Read Lock. Later on, it’s your responsibility to check whether that version is still valid (write-lock has not been acquired).

‘validate(stamp)’ method will tell you whether lock’s version(stamp) is changed. If it is not changed then you can proceed assuming write-lock won’t be acquired. But there is no guarantee. In between your process (here someCaculation(…)), write-lock might be acquired by same or other thread and data might have changed.

So, it is advisable to keep the calculation smaller after acquiring Optimistic Read Lock.

StampedLock has one more capability of **upgrading or downgrading the lock types**.

