Synchronization (cont.)

High Level Synchronization

- **Barriers** (All work being performed by any thread is guaranteed to be completed at barrier exit). Usage:

```
#pragma omp barrier
```

- For OpenMP Tasks, the barrier is the **taskwait** clause. Usage:

```
#pragma omp taskwait
```

- **Critical Section** provides for mutual exclusion: only one thread can enter critical section at a time

```
int myValue = 0;
#pragma omp parallel num_threads(500) shared(myValue)
{
    #pragma omp critical
    myValue++;
}
printf("Value: %d\n", myValue);
```

- **atomic** also provides mutual exclusion by making sure read/write to memory location is protected and singular. (Different from critical clause, which can be any sequence/type of code)

```
#pragma omp atomic
```

Synchronization (cont.)

Low Level Synchronization

- OpenMP **Simple Locks**: A lock is available if it is **unset**

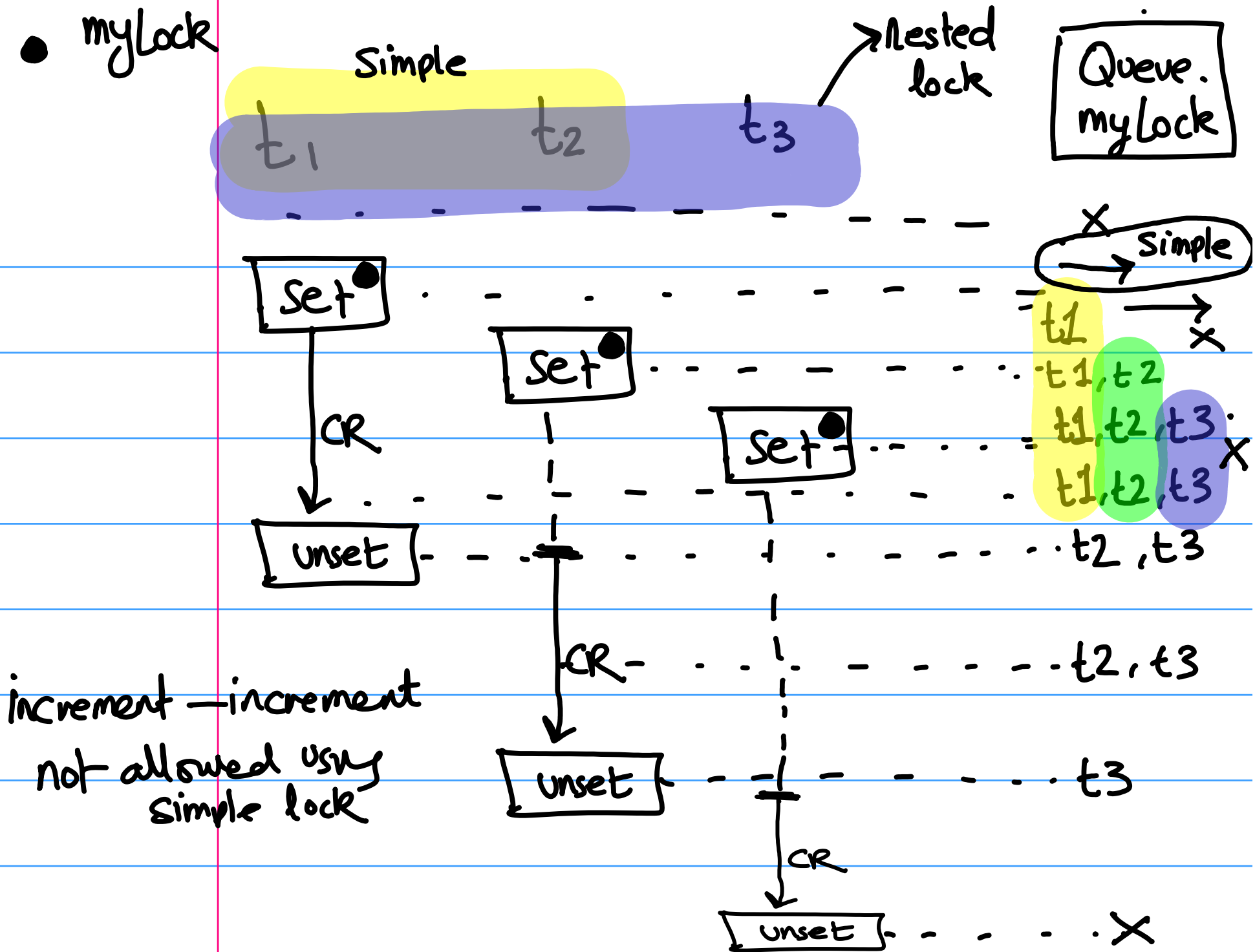
```
omp_lock_t myLock;              // the lock object
omp_init_lock(&myLock);         // initialize to unset
#pragma omp parallel
{
    omp_set_lock(&myLock);      // wait (block) if already locked
    // do some work
    omp_unset_lock(&myLock);    // release lock
}
omp_destroy_lock(&myLock);      // destroy/deallocate lock
```

  - Can also test if a lock is set or not (without blocking) by calling:

```
omp_test_lock(&myLock);    // returns true if set, false otherwise
```

- OpenMP **Nested Locks**:
  - A set is an increment, an unset is a decrement.
  - Can be locked a number of times. Doesn't unlock until you have unset it as many times it was locked.

```
omp_nest_lock_t    myLock;
omp_init_nest_lock(&myLock);
omp_set_nest_lock(&myLock);
omp_unset_nest_lock(&myLock);
omp_destroy_nest_lock(&myLock);
```


myLock
Simple
t1
t2
t3
Nested lock
Queue. myLock
Set
Set
Set
CR
Unset
Simple
t1
t1, t2
t1, t2, t3
t1, t2, t3
t2, t3
CR
t2, t3
increment — increment
not allowed using simple lock
Unset
t3
CR
Unset

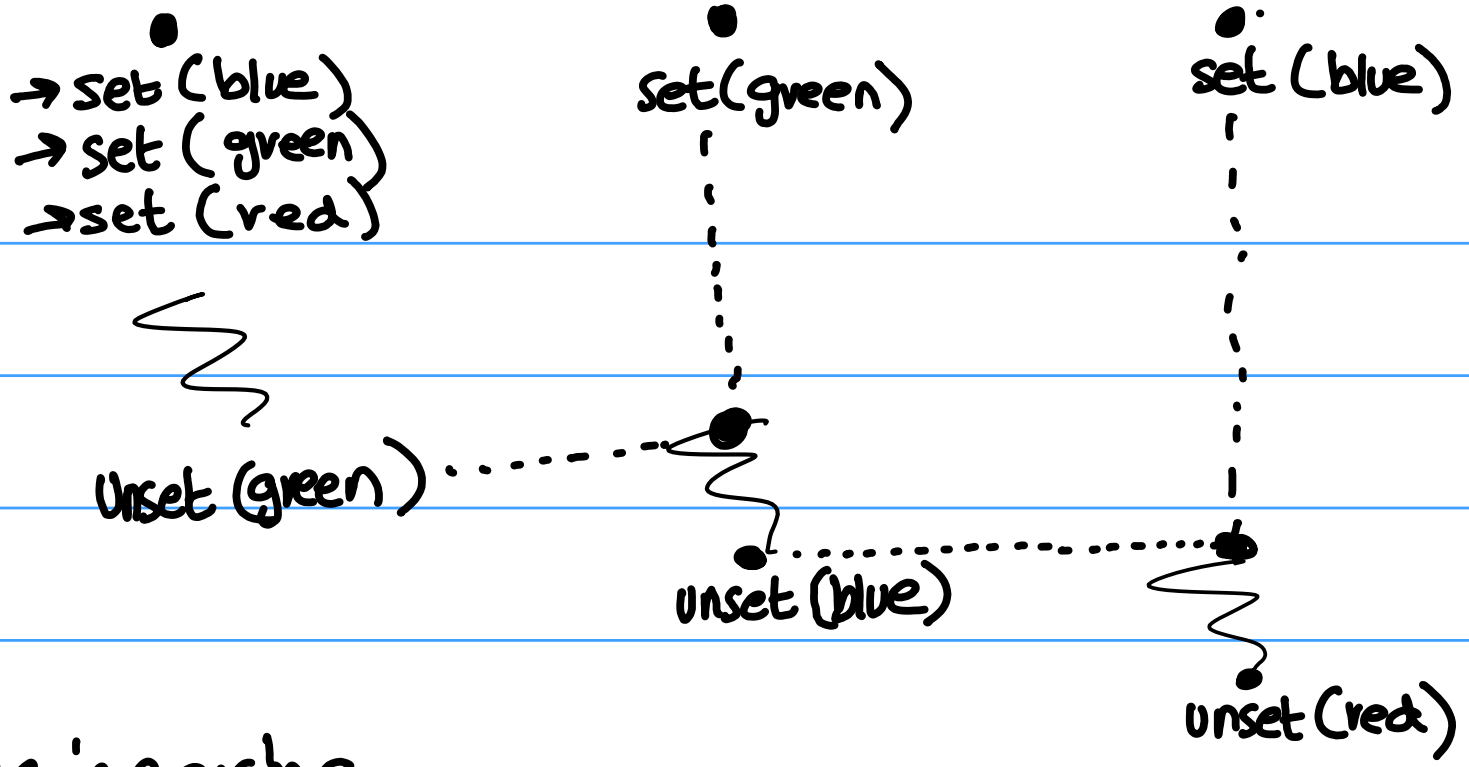
NO CR

1 lock.

3 locks.

Scenario 2 : Synchronization

Red → Green → Blue



set / increment

unset / decrement

lock engineering.