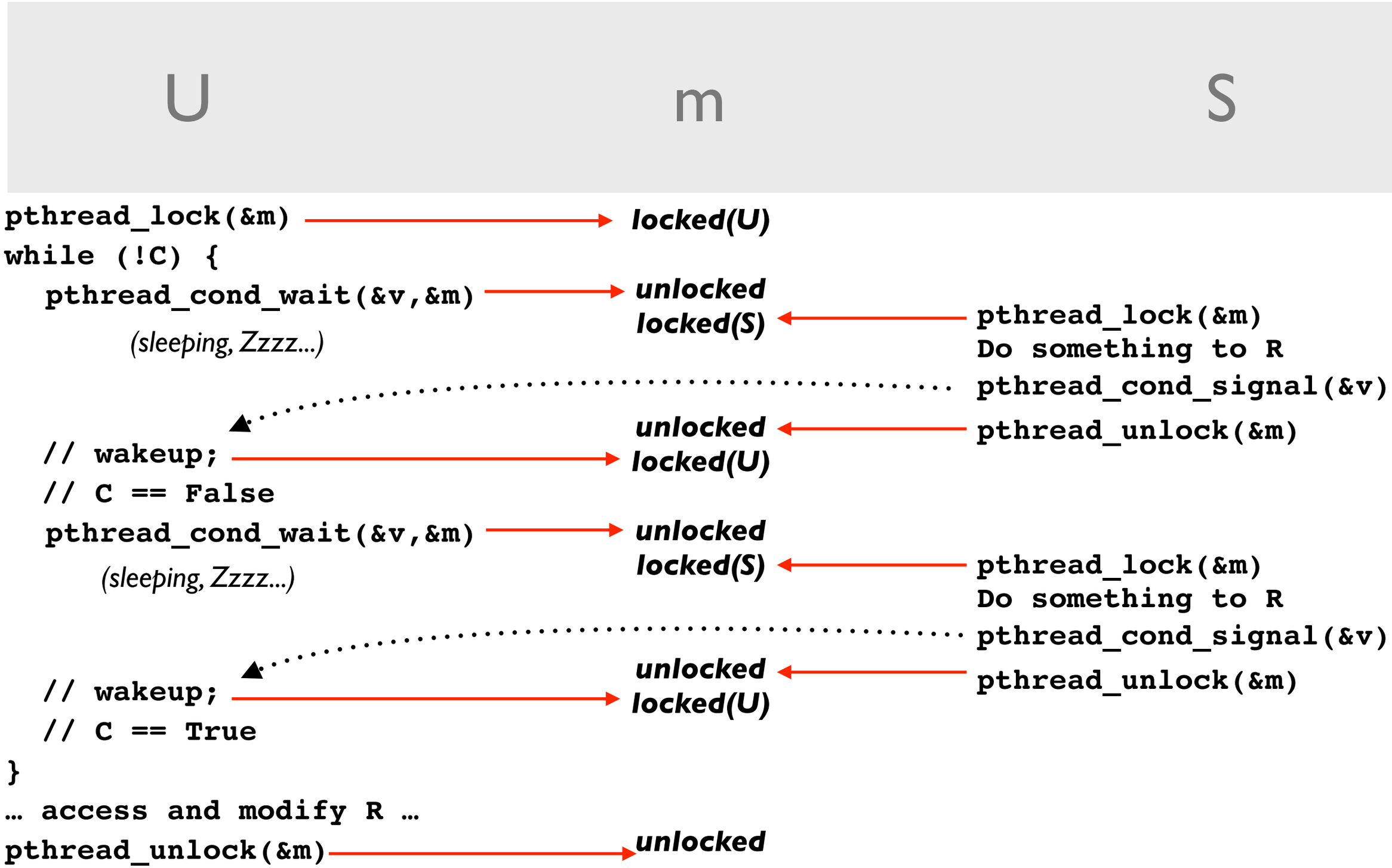
Condition Variable Timelines

- It can help to view timelines
- Next slide shows the following scenario:
  - U wants to update R when C is true
  - It grabs mutex M, checks C (false) and so does a Wait on V, linked to M
    - Wait unlocks M and sleeps...
  - S updates R (with proper M usage) and signals V
  - V locks M, wakes U, but C is still false, so U waits again
    - Wait unlocks M and sleeps...
  - S updates R again and signals V
  - V locks M, wakes U, and C is now true, so U proceeds to modify R and then unlocks M


U
m
S
pthread_lock(&m)
locked(U)
while (!C) {
pthread_cond_wait(&v,&m)
unlocked
locked(S)
pthread_lock(&m)
Do something to R
(sleeping, Zzzz...)
pthread_cond_signal(&v)
unlocked
pthread_unlock(&m)
// wakeup;
locked(U)
// C == False
pthread_cond_wait(&v,&m)
unlocked
locked(S)
pthread_lock(&m)
Do something to R
(sleeping, Zzzz...)
pthread_cond_signal(&v)
unlocked
pthread_unlock(&m)
// wakeup;
locked(U)
// C == True
}
… access and modify R …
pthread_unlock(&m)
unlocked

Using Multiple Mutexes

- A typical program may have many critical shared resources, each protected by its own mutex (e.g. `resource`*i* protected by mutex `m`*i*).
- A thread may want exclusive access to more than one such resource at a time.
- This should be easy:
  - ```
    pthread_mutex_lock(&m1);
    pthread_mutex_lock(&m2);
    … do stuff to resource1 and resource2 …
    pthread_mutex_unlock(&m1);
    pthread_mutex_unlock(&m2);
    ```
- Let's see….
    ```

2 Mutexes (Preamble)

We have two resources, one mutex each for protection.

```
int resource1, resource2 ; // Two global resources

pthread_mutex_t m1 = PTHREAD_MUTEX_INITIALIZER ; // Protects resource1
pthread_mutex_t m2 = PTHREAD_MUTEX_INITIALIZER ; // Protects resource2
```

2 Mutexes (GoingUp)

```
// Going Up: sets r1 = min(r1,r2), r2 = r1+r2
void *GoingUp(void *a) {

    int tmp;

    pthread_mutex_lock(&m1);
    pthread_mutex_lock(&m2);

    tmp = resource1 + resource2;
    if (resource2 < resource1) { resource1 = resource2 ;}
    resource2 = tmp;

    pthread_mutex_unlock(&m2);
    pthread_mutex_unlock(&m1);

    pthread_exit(NULL);
}
```

2 Mutexes (GoingDown)

```
// Going Down: sets r1 = max(r1,r2), r2 = max(r1,r2)-min(r1,r2)
void *GoingDown(void *a) {
    int tmp;

    pthread_mutex_lock(&m2);
    pthread_mutex_lock(&m1);

    if (resource2 < resource1) {
        resource2 = resource1 - resource2;
    } else {
        tmp = resource2 - resource1;
        resource1 = resource2;
        resource2 = tmp;
    }

    pthread_mutex_unlock(&m1);
    pthread_mutex_unlock(&m2);

    pthread_exit(NULL);
}
```

2 Mutexes (Main Program)

```
int main (int argc, const char * argv[]) {
    static pthread_t goingup,goingdown ;
    long rc;

    resource1 = 13; resource2 = 42;
    printf("r1,r2 = %d,%d\n", resource1, resource2);

    printf("Creating GoingUp:\n");
    rc = pthread_create(&goingup,NULL,GoingUp,(void *)0);
    if (rc) { ... }
    printf("Creating GoingDown:\n");
    rc = pthread_create(&goingdown,NULL,GoingDown,(void *)0);
    if (rc) { ... }

    printf("Waiting to join threads....\n");
    pthread_join( goingdown, NULL);
    pthread_join( goingup, NULL);

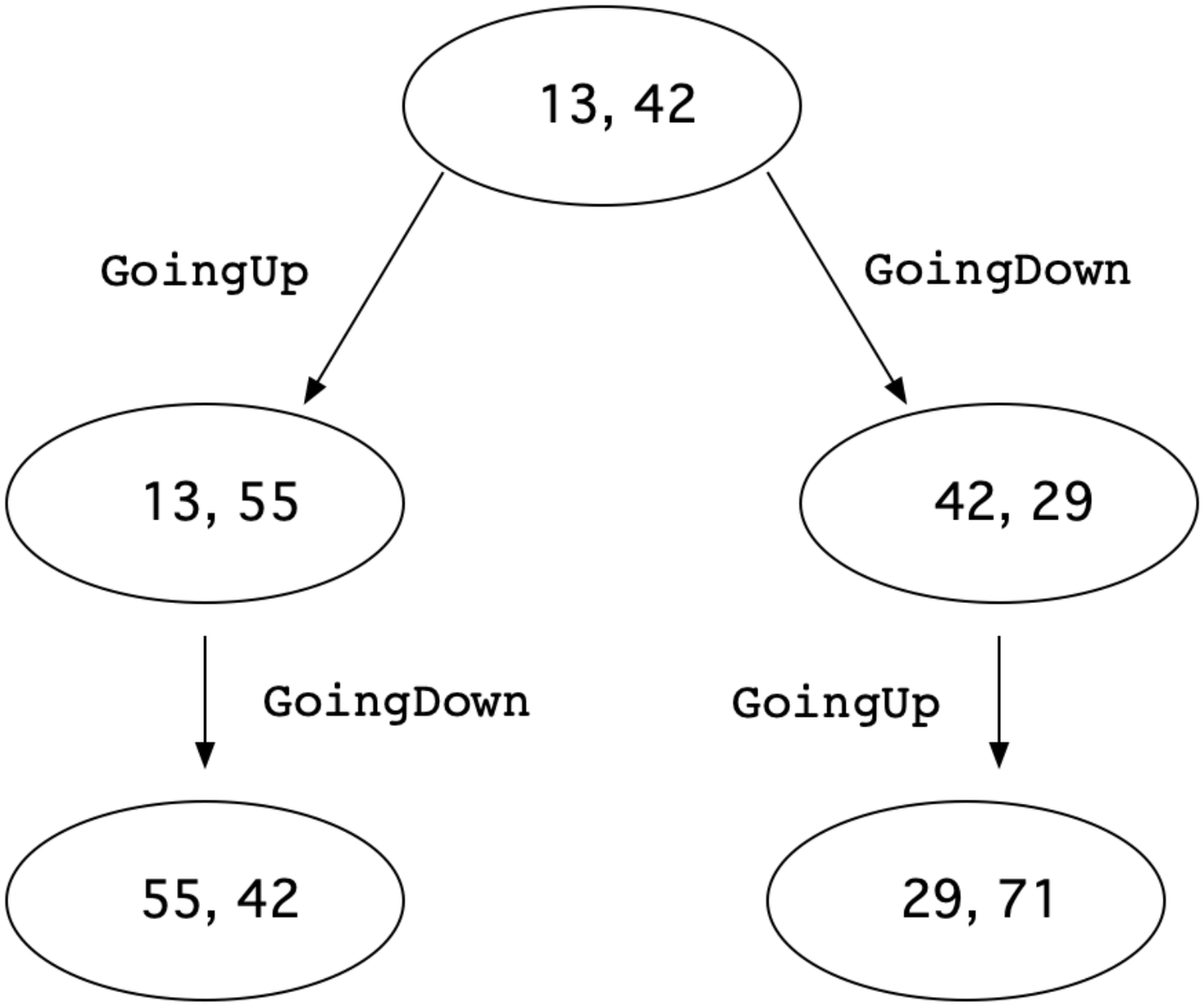
    printf("R1,R2 = %d,%d\n", resource1, resource2);
    printf("All Done!\n");

    return 0;
}
```

Expected Behaviour

- We have two threads, each of which (`GoingUp`,`GoingDown`) uses both resources.
  - r1,r2
- We expect to see one of two possibilities:
  - An execution of `GoingUp` followed by one of `GoingDown`.
    - 13,42 becomes 13,55 becomes 55,42
  - An execution of `GoingDown` followed by one of `GoingUp`.
    - 13,42 becomes 42, 29 becomes 29,71

Expected Behaviour (Diagram)



What actually happens!

- Many times we see the expected behaviour
- But occasionally,
  - it hangs……..

- Another way to cause DEADLOCK !
- The solution:
  - all threads should acquire multiple locks in the same order as each other.