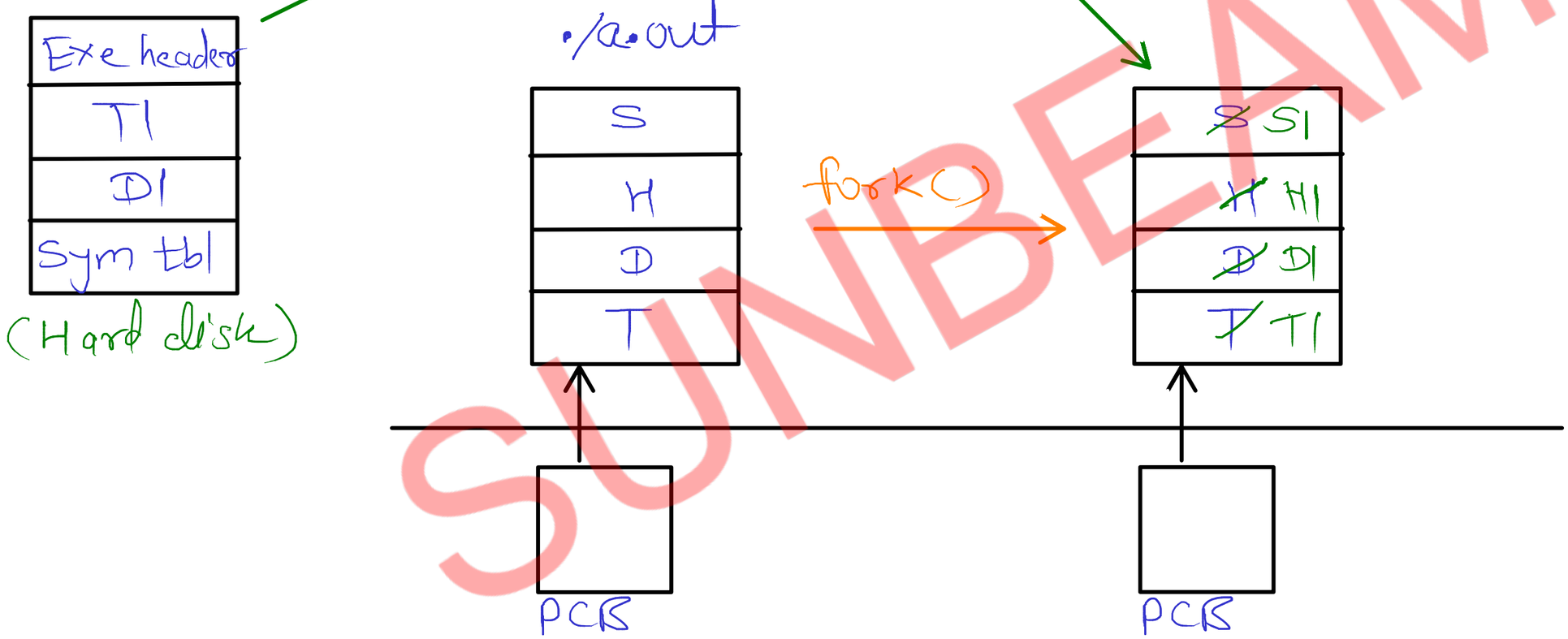
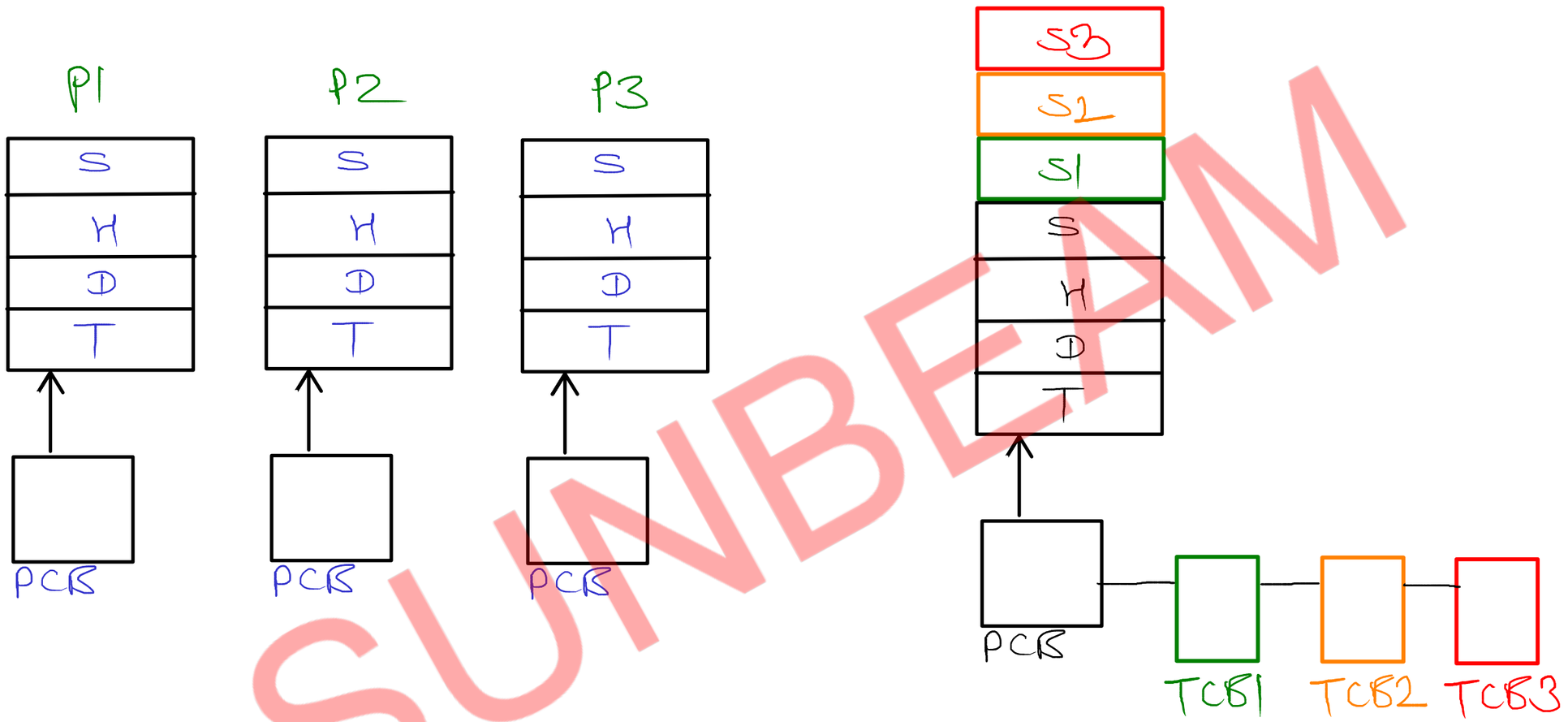
ls -l ⇒

argc = 2
argv[0] = ls
argv[1] = -l
argv[2] = NULL

execl("/usr/bin/ls", "ls", "-l", NULL);



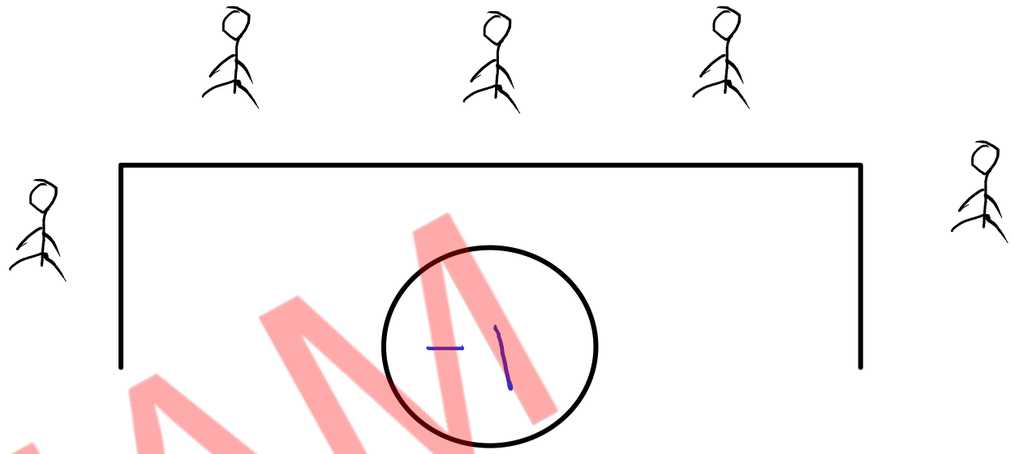
Process Vs Thread



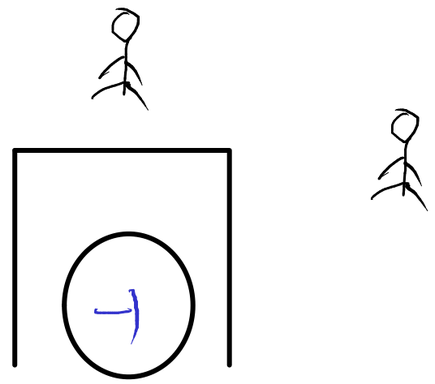
Semaphore

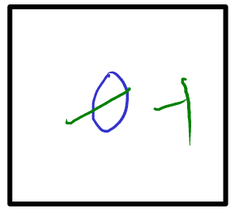
- internally a counter
- Semaphore Operations:
    1. Dec/wait/P Operation:
        - deccrement counter
        - if counter < 0, then block current process/thread
    2. Inc/post/V Operation
        - increment counter
        - if someone is blocked on that semaphore counter wakeup one

Counting Semaphore



Binary Semaphore





sem_init();

```
thread1_fun()
{
    for(i=1; i<=10; i++)
    {
        //dec-sem_wait()
        count--;
        //inc-sem_post()
    }
}
```

```
thread2_fun()
{
    for(i=1; i<=10; i++)
    {
        //dec-sem_wait()
        count++;
        //inc-sem_post()
    }
}
```

Semaphore counter

sem_destroy()

Mutex

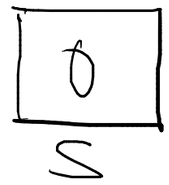
- Mutal Exclusion (one at a time)

- working of mutex is simillar to binary semaphore
- Operations : lock() and unlock()
- the process which will lock the mutex,
  will become owner of that mutex
- Only owner can unlock the mutex

pthread_mutex_create()
pthread_mutex_lock()
pthread_mutex_unlock()
pthread_mutex_destroy()

Thread 1

pf("sunbeam")
sem_post(&S)



Thread 2

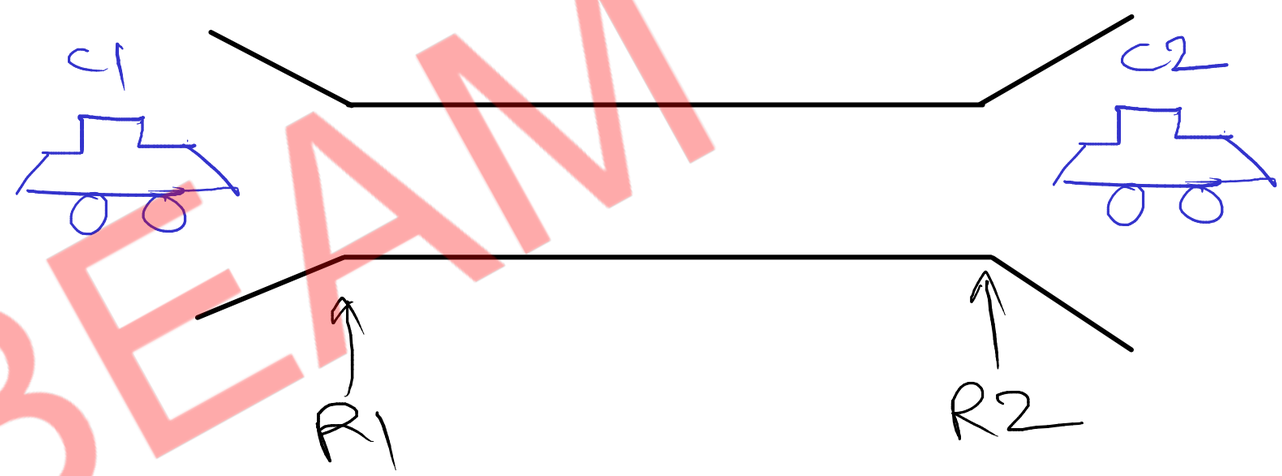
sem_wait(&S)

pf("Karad")

Deadlock

- infinite waiting of process for some resource

- deadlock will occur in system only when below four conditions hold true at a time
    1. Mutual Exclusion
    2. No Preemption
    3. Hold and Wait
    4. Circular Wait



Deadlock Prevention:

- while writing OS code we ensure that 1 out 4 condition will always be false

Deadlock Avoidance:

- Banker's Algorithm
- Resource Allocation Graph
- Safe state Algorithm

Deadlock Recovery:

- resource preemption
- forceful termination of process