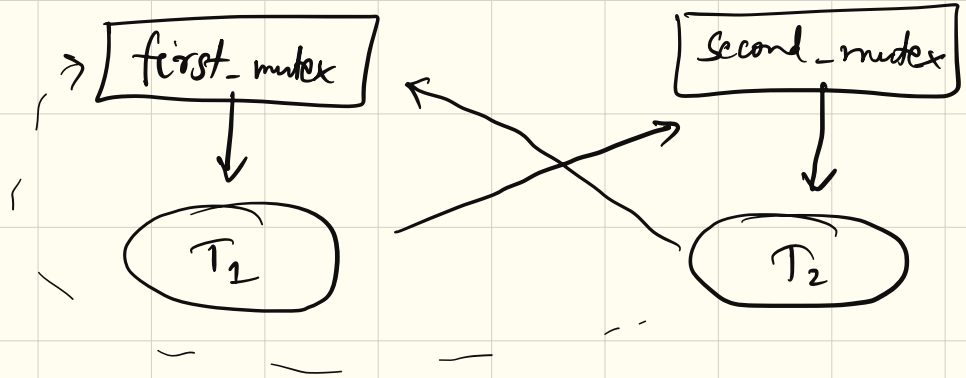
18 Mar 2025 - Operating Systems - II

Deadlock

mutex lock ≡

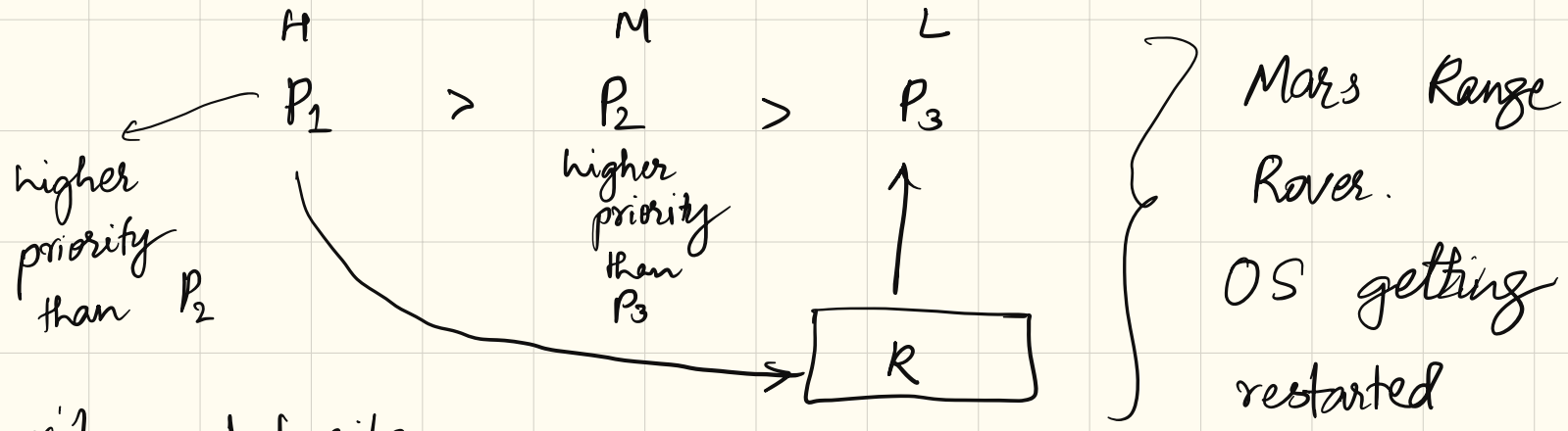
P_0		P_1
wait (S)	obtain ✓	wait (Q)
wait (Q)	wait	wait (S)
...		...
signal (S)		signal (Q)
signal (Q)		signal (S)



Resource Allocation Graph

Deadlock and starvation are different

starvation → some thread is making progress (Dining Philosophers)



Priority Inheritence Problem

P_2 → not waiting for resource preempts P_3

Mars rover : H was waiting on L → low priority

bc_dist ↓ high priority

L ↓ pre-empted by multiple medium priority tasks

Lock free algorithms

Alternatives to locks

- → Open MP
- → Transactional Memory
- → Thread Pools

```
void update() {
    acquire()
    modify shared data
    release()
}
```

```
void update()
{
    atomic {
        ---
    }
}
```

atomic {S}

- Memory transaction defn

```
void update (int value) {
    # pragma omp critical ──→ CS
    {
        count += value
    }
}
```

create threads and handle sync problems

→ threads are created automatically

→ very popular among non-CS people
  → easy to use
  → may not be the most efficient

Functional Programming Language

WhatsApp

"Open Ganesh"

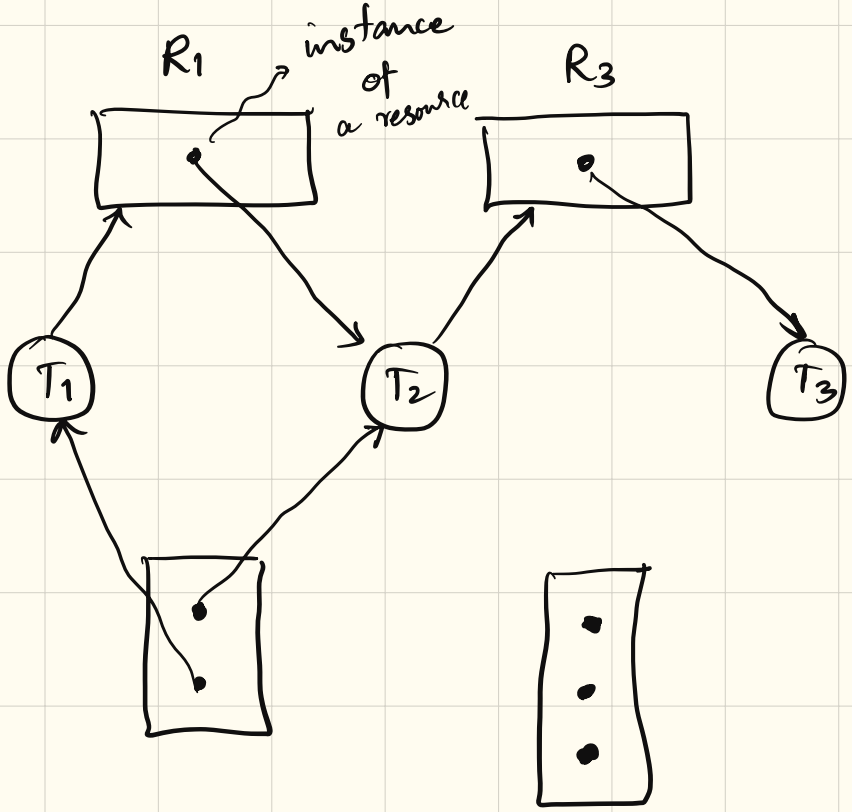
— Kaushal

Deadlock Characterization

- Mutual exclusion
- Hold and wait: Bank transfer.
- No preemption
- Circular wait } → can be avoided

Circular waiting

→ in some cases may lead to deadlock (not always)



→ If there is only one instance of each resource, circular waiting ⇒ deadlock

Ex
Printer → multiple resources (?)

Other

* Deadlock prevention
* Deadlock avoidance

* Allow the system to enter a deadlock state and then recover
* Ignore deadlock → easiest
  → resource locked forever
  restart

Circular waiting solution : acquire in increasing order of subscript ($T_n \rightarrow R_0$ first then R_n)

will lead to deadlock: $T_2 \rightarrow R_1 \rightarrow T_2 \rightarrow R_2 \cdots T_{n-1} \rightarrow R_n \rightarrow T_n \rightarrow R_0 \rightarrow T_2$

Deadlock avoidance

→ unsafe : possibility of deadlock