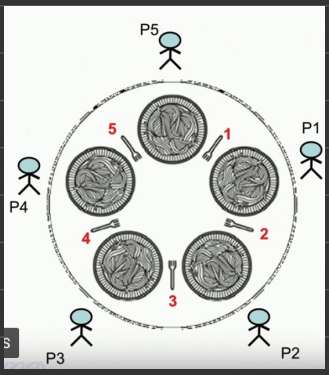
Dining Philosopher's Problem:



→ Philosophers either eat or think
→ To eat, one needs to hold both the forks (left & right one)
→ If one is not eating, they are thinking.

* Problem statement → Develop an algo where no one starves.

first try →

```
# define N 5

void philosophers (int i) {
    while (true) {
        think();
        take_fork (R:i);
        take_fork (L:i);
        eat();
        put_fork (Li);
        put_fork (Ri);
    }
}
```

what happens if only P_1 and P_3 are always given priority

P_4, P_3, P_2 will starve so scheme needs to be fair.

What happens if every decides to pick the right fork at same time? Possible starvation due to deadlock.

Deadlock → A situation where programs continue to run

indefinetely without making any progress. Each program is waiting for an event that another program can cause.

Second Try:

```
#define N 5

void philosopher (int i) {
    while (1) {
        think();
        take_fork(R);
        if( available (L)) {
            take_fork (L);
            eat();
            put_fork (R);
            put_fork (L);
        } else {
            put_fork(R);
            sleep (T);
        }
    }
}
```

Imagine,

Imagine all philosophers start at the same time Run simultaneously And think for same time

this could lead to them taking fork & putting it down continiously.

Scond try (A better solution);

Instead of sleeping for a fixed time we can sleep for random_time. Although this won't guarantee no starvation but will reduce the possibility.

Third attempt (Solution using Mutex)

Protect critical solution with a mutex

Prevents deadlock

But has performance issues -

- Only one philosopher can eat at a time.

```
#define N 5

void philosopher (int i) {
    while (1) {
        think (); // for some time
        lock (mutex);
        take_fork (R[i]);
        take_fork (L[i]);
        eat ();
        put_fork (R[i]);
        put_fork (L[i]);
        unlock (mutex);
    }
}
```

Solution with Semaphores :-

Uses N semaphores (s[1], s[2], ..., s[N]) all initialized to 0, and a mutex philosopher has 3 states : HUNGRY, EATING, THINKING.

A philosopher can only move to EATING state if neither neighbour is eating

```
void philosophers(int i) {
    while (1) {
        think();
        take_forks(i);
        eat();
        put_forks();
    }
}
```

```
void take_forks(int i){
    lock(mutex);
    state[i] = HUNGRY;
    test(i);
    unlock(mutex);
    down(s[i]);
}
```

```
void put_forks(int i) {
    lock(mutex);
    state[i] = THINKING;
    test(LEFT);
    test(RIGHT);
    unlock(mutex);
}
```

```
void test(int i){
    if(state[i] = HUNGRY &&
    state[LEFT] != EATING
    && state[RIGHT] !=
    EATING){
        state[i] = EATING;
        up(s[i]);
    }
}
```