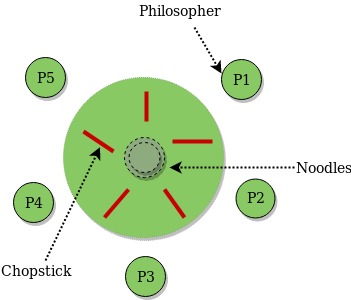
**The Dining Philosopher Problem –** The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both. 



**Constraints and Condition for the** **problem :**

1. Every Philosopher needs two forks in order to eat.
2. Every Philosopher may pick up the forks on the left or right but only one fork at once.
3. Philosophers only eat when they had two forks. We have to design such a protocol i.e. pre and post protocol which ensures that a philosopher only eats if he or she had two forks.
4. Each fork is either clean or dirty.

**Solution :**  
Correctness properties it needs to satisfy are :

1. **Mutual Exclusion Principle –**  
   No two Philosophers can have the two forks simultaneously.
2. **Free from Deadlock –**  
   Each philosopher can get the chance to eat in a certain finite time.
3. **Free from Starvation –**When few Philosophers are waiting then one gets a chance to eat in a while.
4. No strict Alternation.
5. Proper utilization of time.

**Semaphore Solution to Dining Philosopher –**  
Each philosopher is represented by the following pseudocode: 

|  |
| --- |
| process P[i]  while true do  { THINK;  PICKUP(CHOPSTICK[i], CHOPSTICK[i+1 mod 5]);  EAT;  PUTDOWN(CHOPSTICK[i], CHOPSTICK[i+1 mod 5])  } |

There are three states of the philosopher: **THINKING, HUNGRY, and EATING**. Here there are two semaphores: Mutex and a semaphore array for the philosophers. Mutex is used such that no two philosophers may access the pickup or putdown at the same time. The array is used to control the behavior of each philosopher. But, semaphores can result in deadlock due to programming errors.