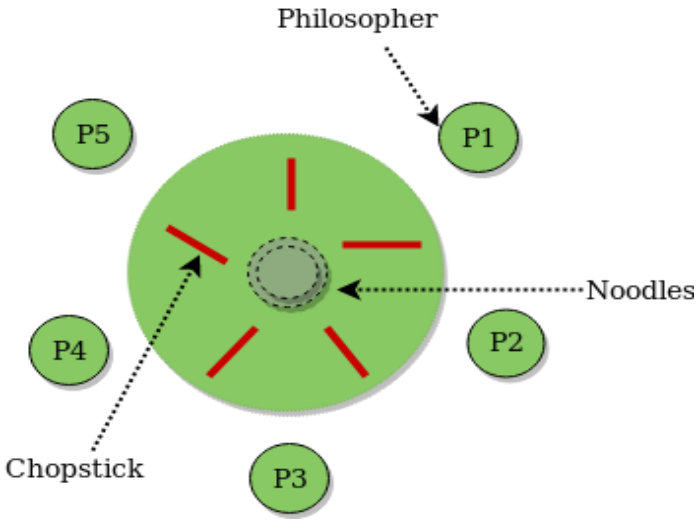
Philosopher Algorithm

What is Dining Philosopher concept?

The dining philosopher problem is a classic example in computer science often used to illustrate synchronization issues and solutions in concurrent algorithm design. It illustrates the challenges of avoiding a system state where progress is not possible, a deadlock. The problem was created in 1965 by J.K. Dijkstra.

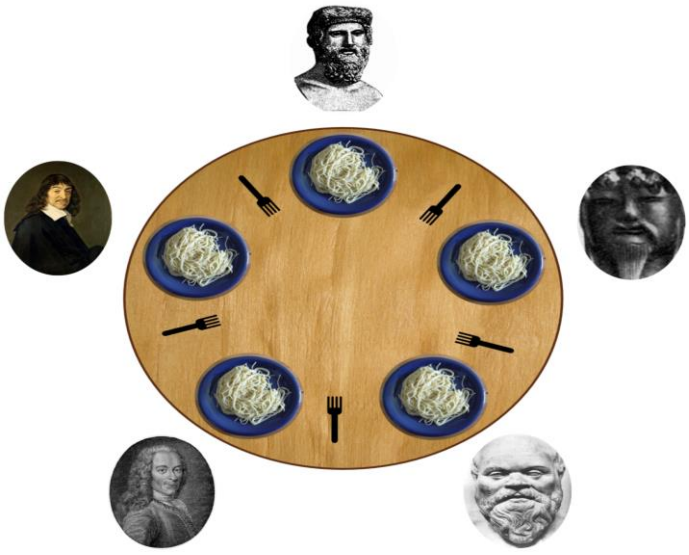
How do you solve the dining philosopher problem?

Dijkstra's solution using semaphores: Assign a semaphore to each chopstick, representing its availability. Philosophers must acquire both chopsticks by locking their corresponding semaphores. If a philosopher cannot acquire both chopsticks, they release the acquired chopstick and wait before trying again.



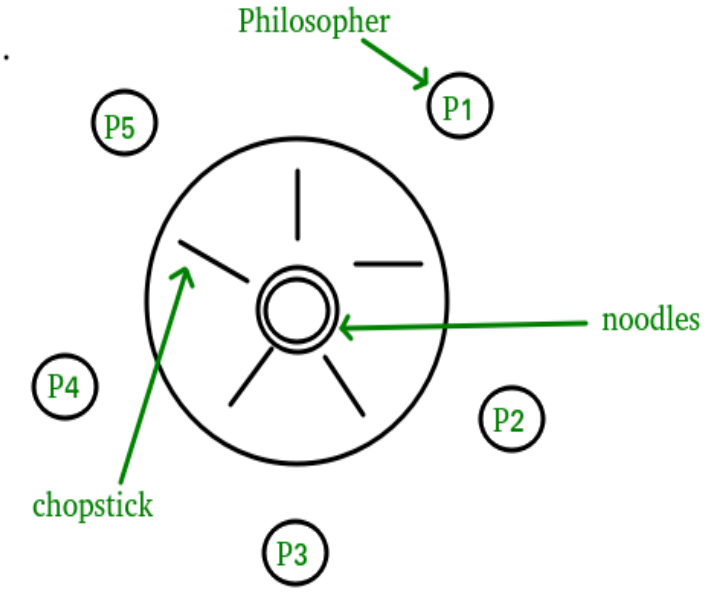
What is the purpose of dining philosophers?

In computer science, the dining philosopher problem is an example problem often used in concurrent algorithm design to illustrate synchronization issues and techniques for resolving them.



Dining Philosopher Problem Using Semaphores

The Dining Philosopher Problem states that K philosophers are 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.



Implementation of the Dining Philosopher Problem solution using semaphores in Python

SOURCE CODE:

```
import threading
import time
import random
num_philosophers = 5
num_forks = num_philosophers
forks = [threading.Semaphore(1) for i in range(num_forks)]
mutex = threading.Semaphore(1)
def philosopher(index):
    while True:
        print(f"Philosopher {index} is thinking...")
        time.sleep(random.randint(1, 5))
```

```
            mutex.acquire()
            left_fork_index = index
            right_fork_index = (index + 1) % num_forks
            forks[left_fork_index].acquire()
            forks[right_fork_index].acquire()
            mutex.release()
            print(f"Philosopher {index} is eating...")
            time.sleep(random.randint(1, 5))
            forks[left_fork_index].release()
            forks[right_fork_index].release()
philosopher_threads = []
for i in range(num_philosophers):
    philosopher_threads.append(threading.Thread(target=philosopher, args=(i,)))
for thread in philosopher_threads:
    thread.start()
for thread in philosopher_threads:
    thread.join()
```

OUTPUT: