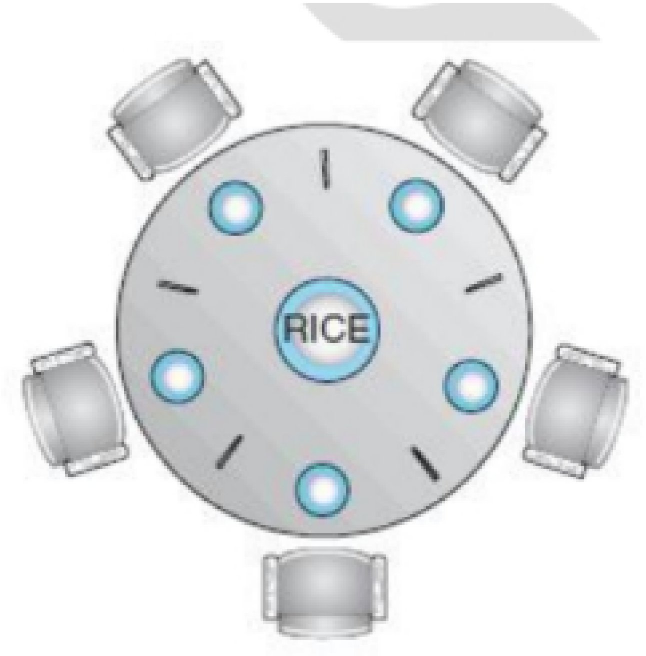
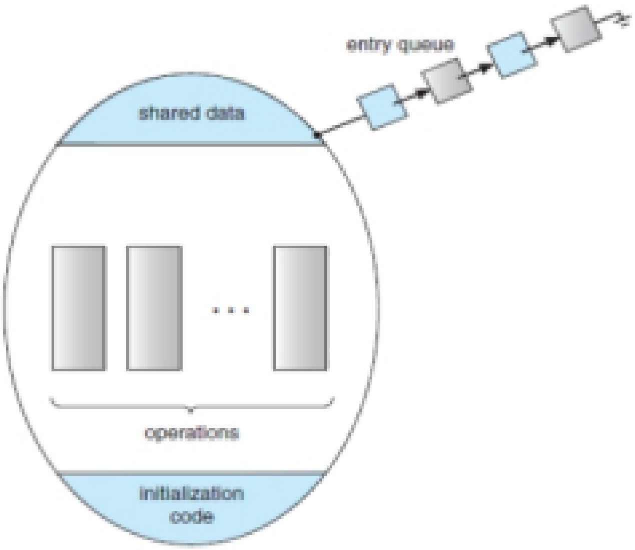


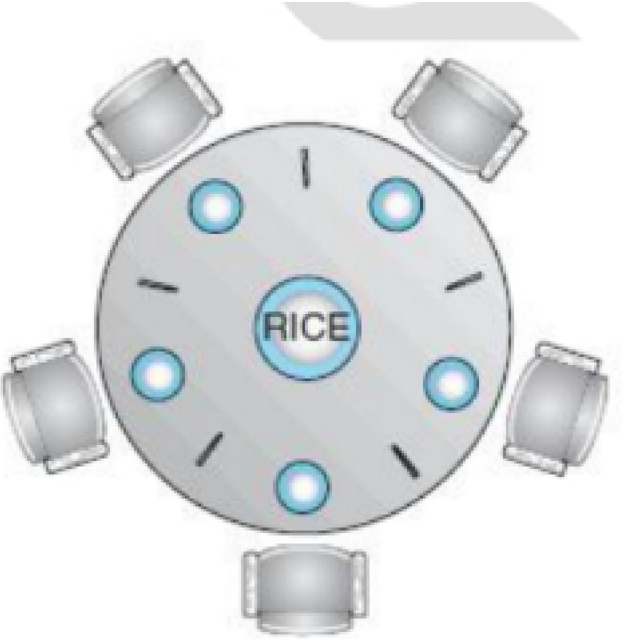
- It is a simple representation of the need to allocate several resources among several processes in a deadlock-free andstarvation-free manner.
- Consider five philosophers who spend their lives thinkingandeating.
- The philosophers share a circular table surrounded byfivechairs,each belonging to one philosopher.
- In the center of the table is a bowl of rice, and the table is laidwith five single chopsticks.
- A philosopher gets hungry and tries to pick up the twochopsticks that are closest to her (the chopsticks that arebetween her and her left and right neighbors).
- A philosopher may pick up only one chopstick at a time. • When a hungry philosopher has both her chopsticks at thesame time, she eats without releasing the chopsticks. • When she is finished eating, she puts down both chopsticksandstarts thinking again.
- Problem: Develop an algorithm where no philosopher starvesi.e., every philosopher should get a chance to eat
- Initial Solution: One simple solution is to represent eachchopstick with a semaphore.
- A philosopher tries to grab a chopstick by executing await() operation on that semaphore. She releases her chopsticksbyexecuting the signal() operation on the appropriate semaphores.

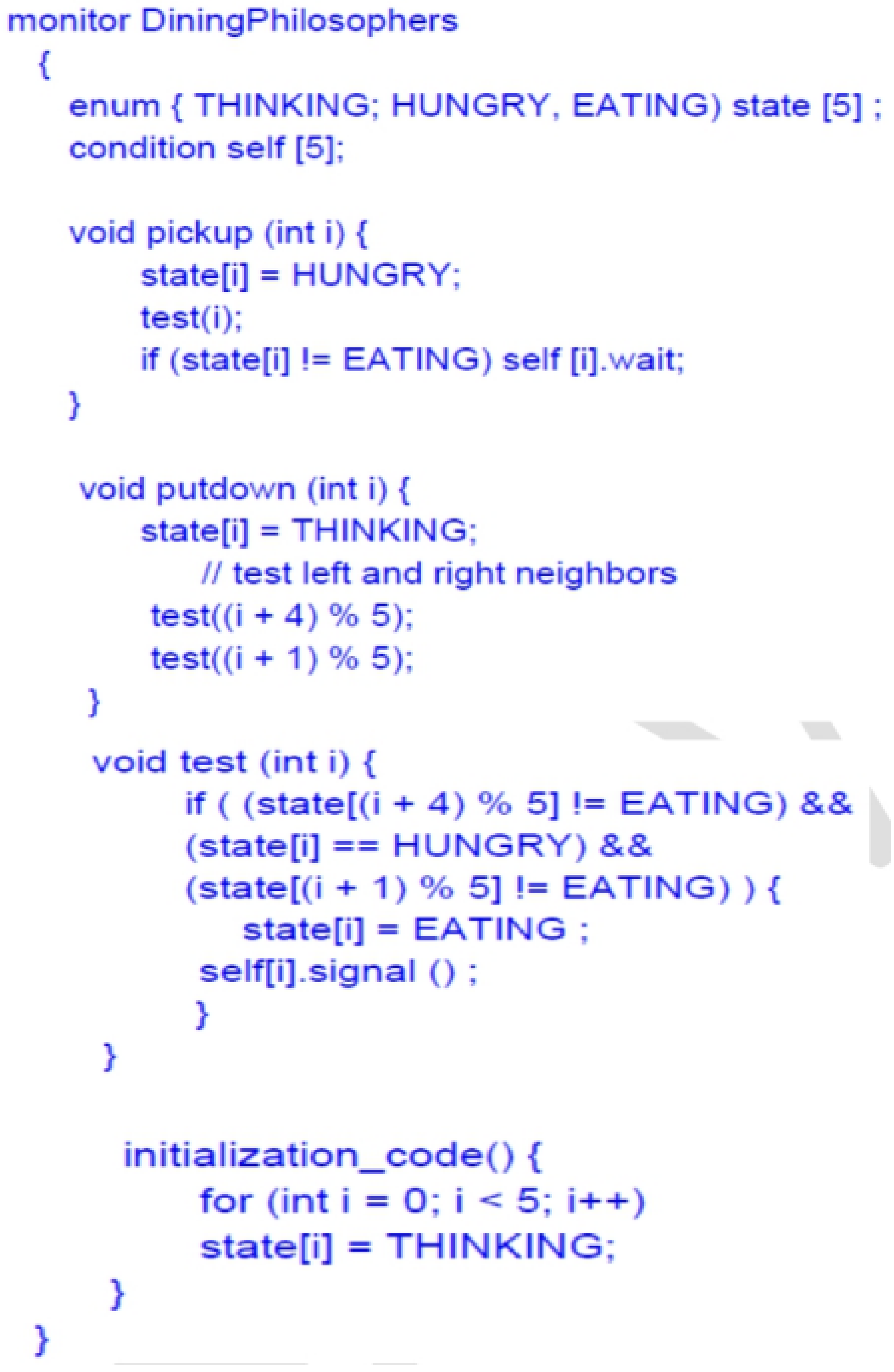
Several possible remedies to the deadlock problem are: • Allow at most four philosophers to be sitting simultaneously at the table.

- Allow a philosopher to pick up her chopsticks only if both chopsticksareavailable.

- Use an asymmetric solution—that is, an odd-numbered philosopher picksupfirst her left chopstick and then her right chopstick, whereas anevennumbered philosopher picks up her right chopstick and then her left chopstick.







```
monitor DiningPhilosophers
  {
    enum { THINKING; HUNGRY, EATING) state [5] ;
    condition self [5];

    void pickup (int i) {
         state[i] = HUNGRY;
         test(i);
         if (state[i] != EATING) self [i].wait;
    }

     void putdown (int i) {
         state[i] = THINKING;
              // test left and right neighbors
          test((i + 4) % 5);
          test((i + 1) % 5);
      }

      void test (int i) {
             if ( (state[(i + 4) % 5] != EATING) &&
             (state[i] == HUNGRY) &&
             (state[(i + 1) % 5] != EATING) ) {
                 state[i] = EATING ;
              self[i].signal () ;
              }
       }


        initialization_code() {
              for (int i = 0; i < 5; i++)
              state[i] = THINKING;
        }
  }
```