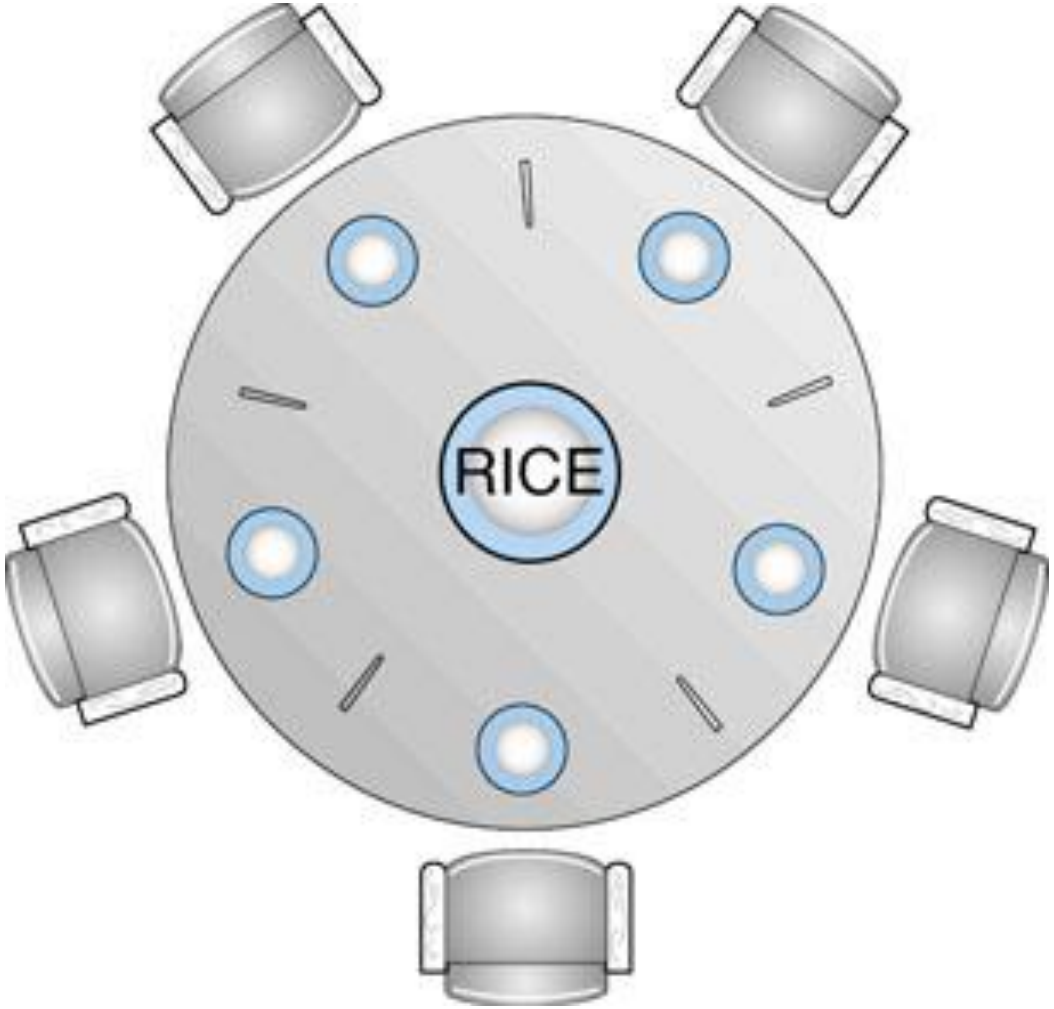
Sistemas Operacionais I

Sincronização

Prof. Leandro Marzulo

Filósofos Comensais (Dining Philosophers) – Dijkstra 65



Solução com semáforos

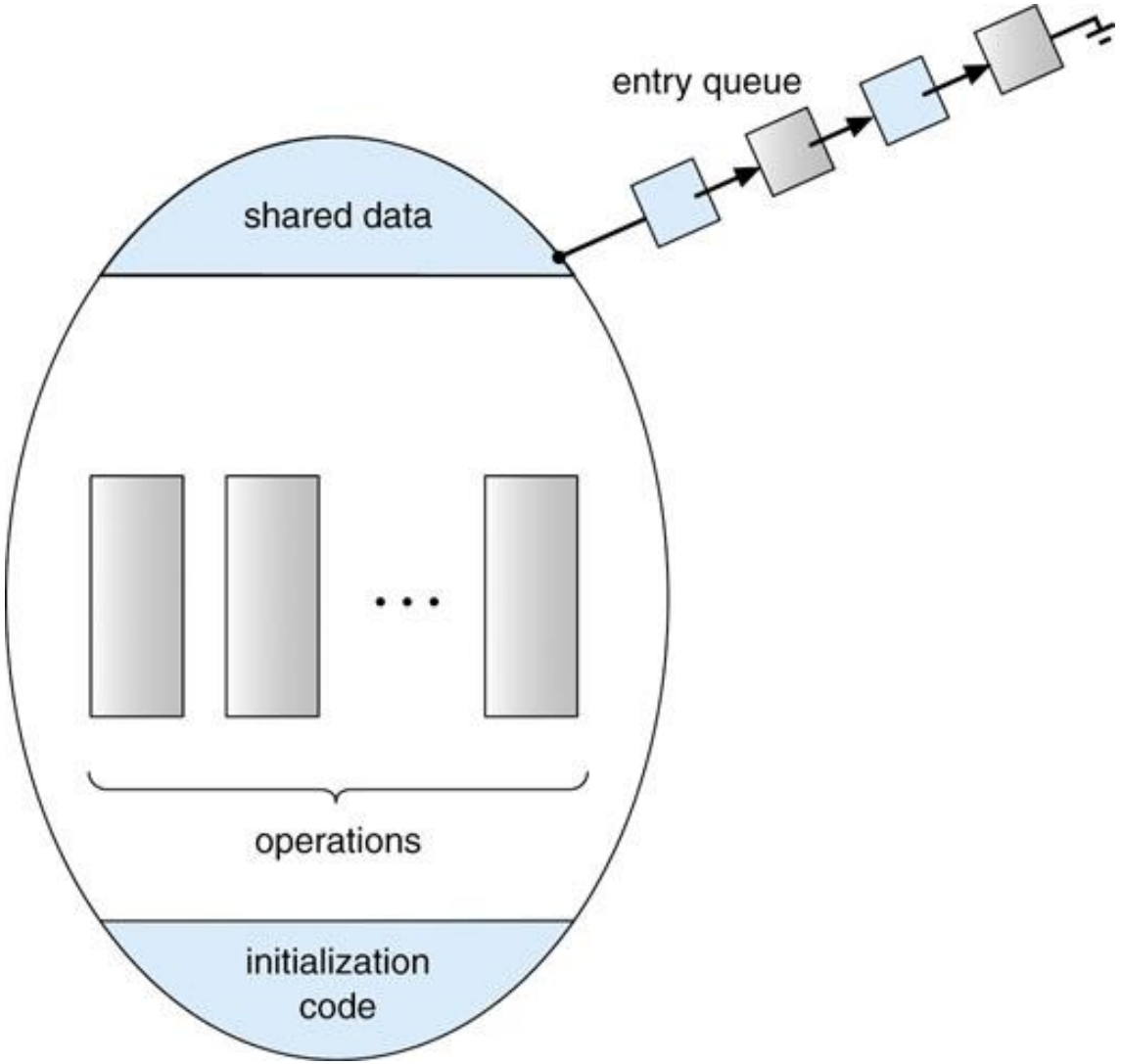
```
While (true) {
   sem_wait(chopstick[i]);
   sem_wait(chopstick[(i+1)%5]);

   //come

   sem_post(chopstick[i]);
   sem_post(chopstick[(i+1)%5]);

   //pensa
}
```

Monitor

- Abstração de mais alto nível (evitar erros no uso de semáforos..), provendo um conjunto mais rico funcionalidades para a coordenação de processos;

- Somente um único processo pode ficar ativo dentro de um monitor: exclusão mútua implícita no acesso à uma região crítica;

Monitor

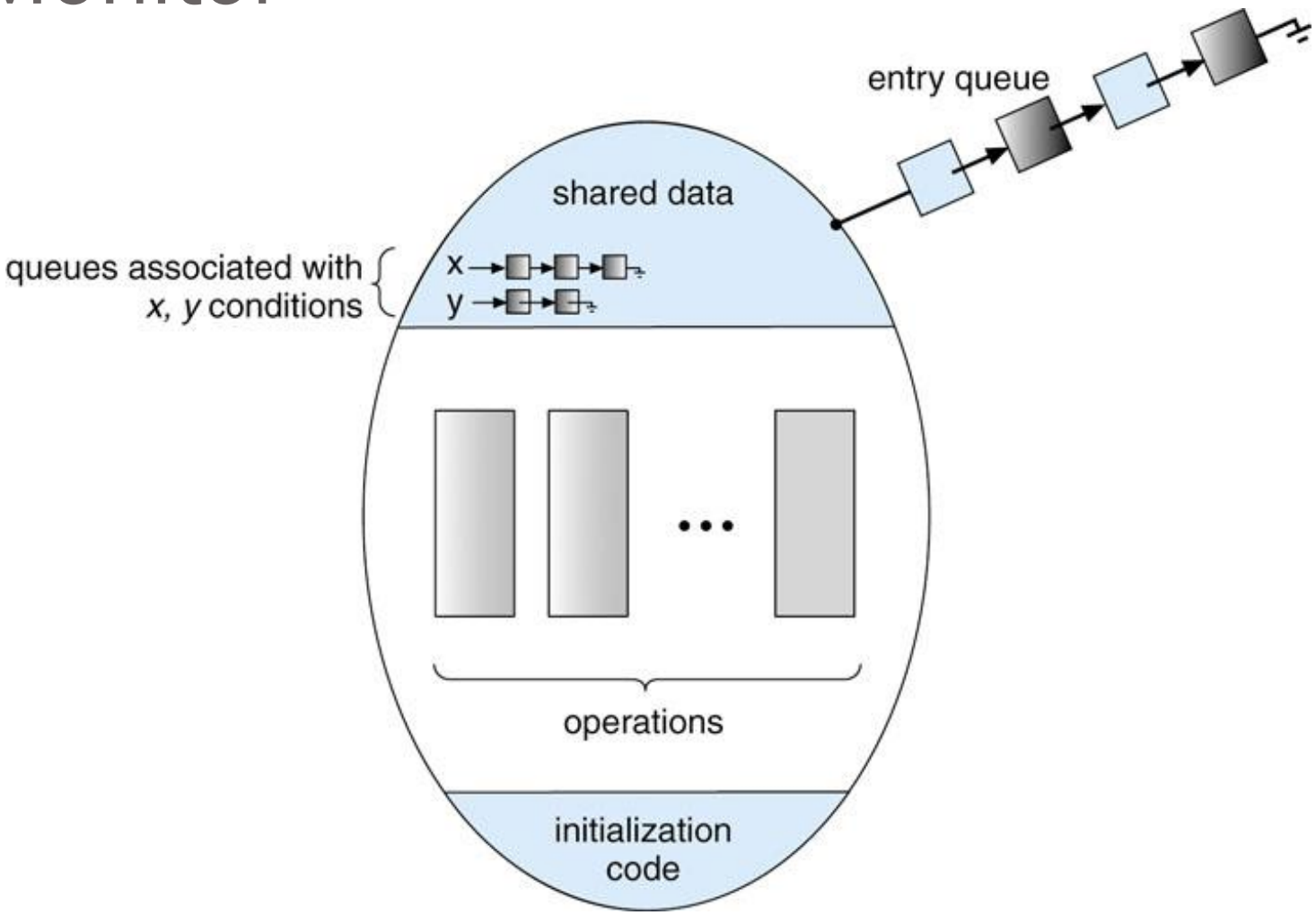


Monitor

- Contém variáveis de condição (ex: x) sobre operações: x.wait() e x.signal().

- Se um processo **espera por uma condição dentro do monitor** para acessar uma região crítica, é automaticamente bloqueado, liberando o lock do monitor para outro processo.

Monitor


entry queue
shared data
queues associated with
x, y conditions
x
y
operations
initialization
code

Monitor

```
monitor DP
{
   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);
   }
```

Monitor

```
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;
}
}
```

Monitor

```
While(true){
    DP.pickup(i);

    //come;

    DP.putdown(i);

    //pensa
}
```