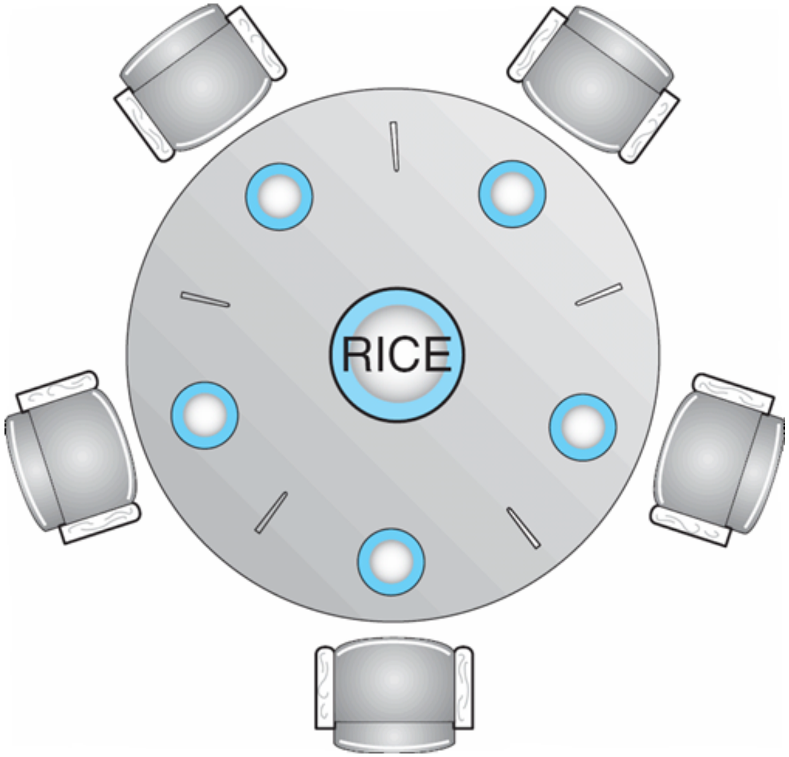
Dining-Philosophers Problem



- Shared data
  - Bowl of rice (data set)
  - Semaphore *chopstick [5]* initialized to 1

Dining-Philosophers Problem (Cont.)

- The structure of Philosopher *i*:

```
while (1)  {
        P ( chopstick[i] );
        P ( chopStick[ (i + 1) % 5] );

               //  eat

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

               //  think

}
```

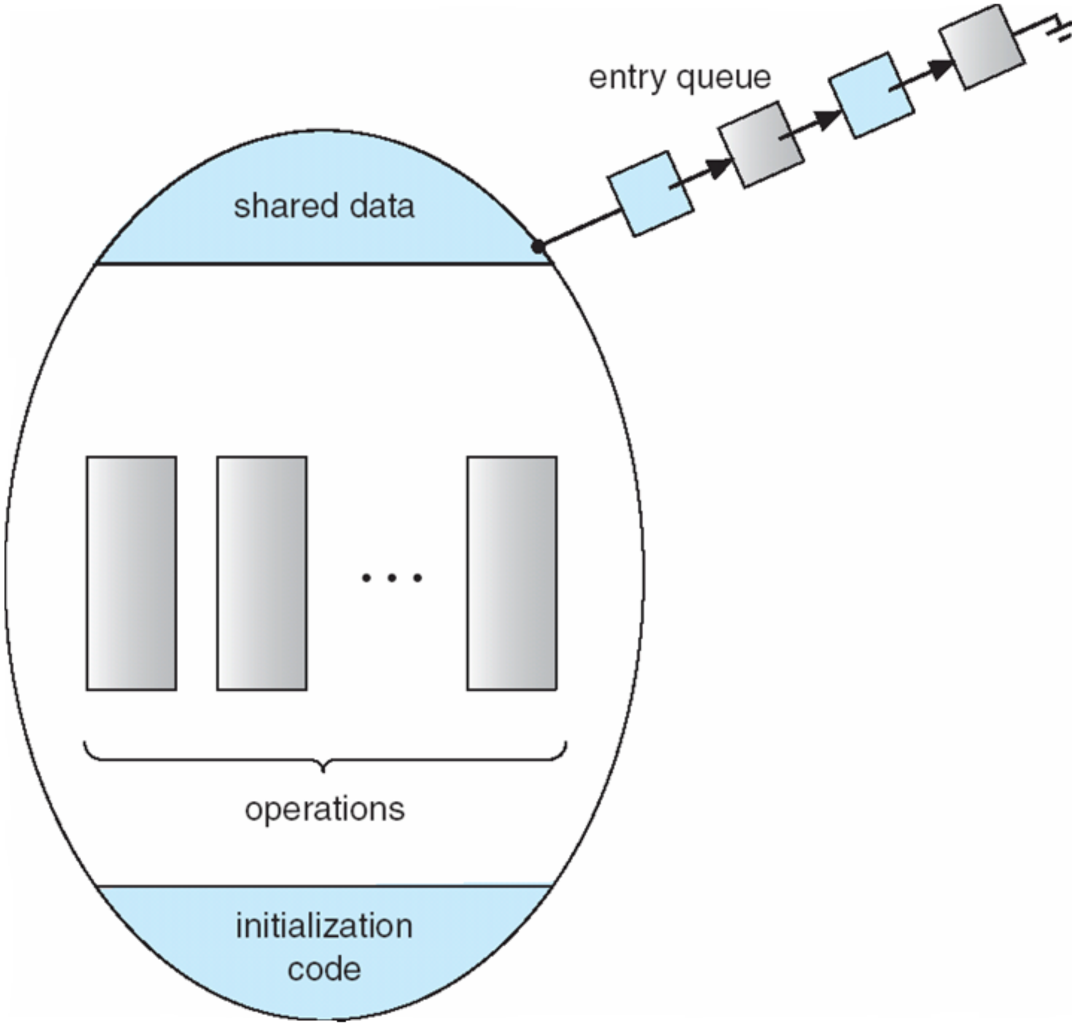
Monitors

- Higher-level, considered more intuitive than semaphores
- Usually requires a language construct (i.e. C doesn’t have it)
- Consists of shared data, subroutines, initialization code
- Only one process can be in any monitor subroutine at any time, all others are forced to wait

Monitors

- To use a monitor to ensure mutual exclusion for a critical section:
  - put the CS in a subroutine that can be called by all processes that want access to the CS
  - put that subroutine in a monitor (e.g. put it in the "procedures" section of the monitor)
- easy to extend to several related critical sections
  - e.g. list manipulation routines

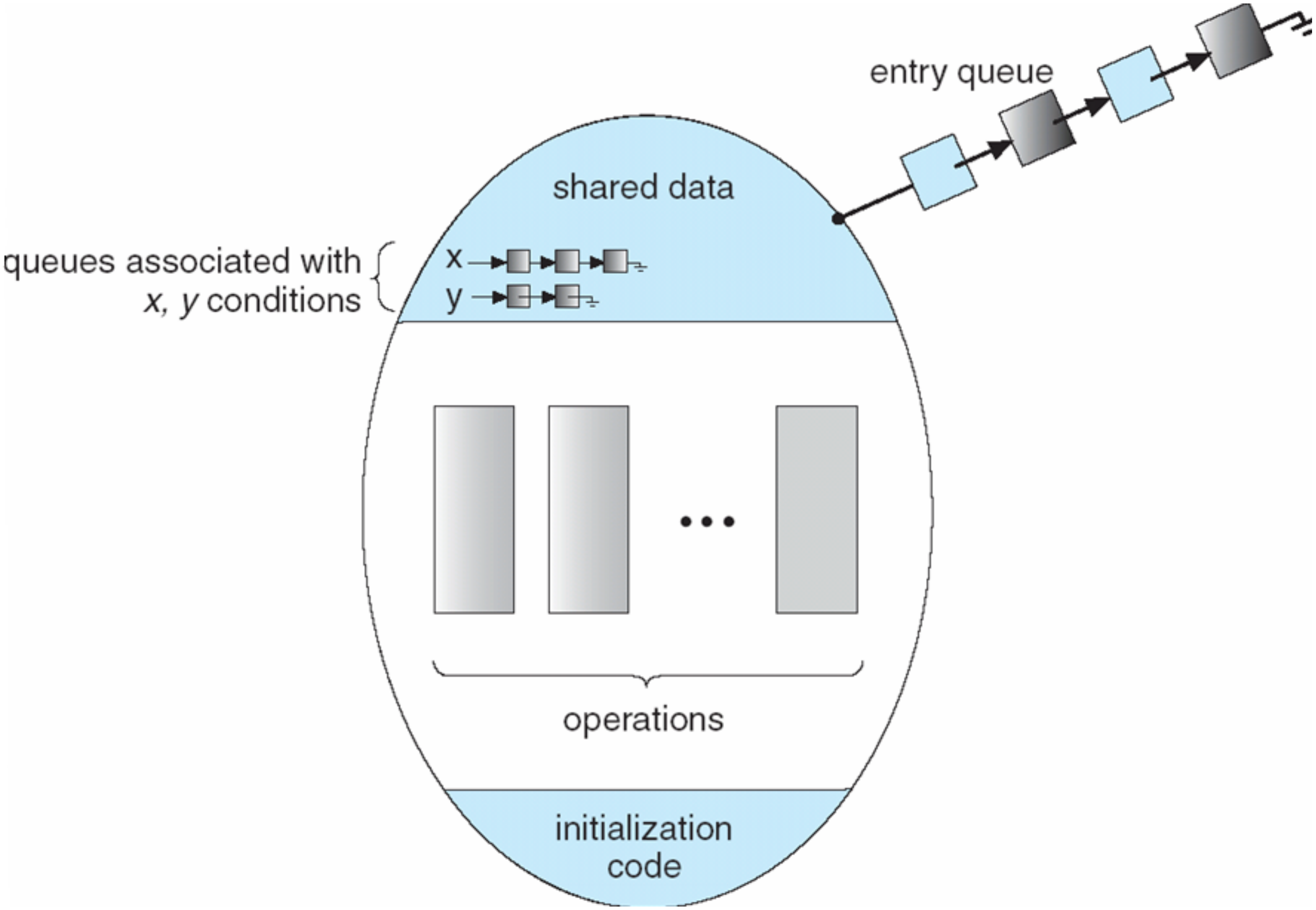
Schematic View of a Monitor



Monitors

- Wait(C)
  - suspends the current process until another process calls Signal(C), C is a *condition variable*
    - a suspended process is considered to be “out of the monitor” so other processes can access monitor routines
- Signal(C)
  - if there is a process suspended on C, resume it and wait until it has left the monitor before proceeding
  - otherwise, continue execution
- Notify(C)
  - like Signal, but resume the suspended process after the notifying process leaves the monitor

Monitor with Condition Variables



Binary Semaphore via a Monitor

```
MONITOR SEM
    -shared data:
        int busy;
        condition nonbusy;
    -Procedures: P() {
                        if (busy)
                            nonbusy.wait();
                        busy++;
                 }
                 V() {
                        busy = 0;
                        nonbusy.signal();
                 }
    -Init Code:  begin() {
                        busy = 0;
                 }
END SEM
```

Producer/Consumer via Monitor

```
MONITOR LISTMON
    - shared data:
        LIST itemList;
        int maxListSize = N;
        condition bufavail, itemavail;
    -Procedures:
        enqueueItem(item) {
            if(ListCount(itemList) == maxListSize)
                bufAvail.wait();
            ListPrepend(itemList, item);
            itemAvail.signal();
        }
```

Producer/Consumer via Monitor

```
        getItem() {
            if(ListCount(itemList) == 0) /* list is empty */
                itemAvail.wait();
            item = ListTrim(itemList);
            bufAvail.signal();
            return(item);
        }
    -Init Code:
        itemList = ListCreate();
END LISTMON
```