CS420: Operating Systems Classic Problems of Synchronization

James Moscola
Department of Engineering & Computer Science
York College of Pennsylvania

Classical Problems of Synchronization

- **Classical problems used to test newly-proposed synchronization schemes**
  - Bounded-Buffer Problem
  - Readers and Writers Problem
  - Dining-Philosophers Problem

Bounded-Buffer Problem (Producer/Consumer Problem)

- **In a generalized Bounded Buffer problem, N buffers, each can hold one data item**
- **Utilize three semaphores to control access to buffer between the producer and the consumer**
  - Semaphore mutex locks access to critical region of code where buffer is modified
    - Initialized to the value 1
  - Semaphore full keeps track of how many items are actually in the buffer
    - Initialized to the value 0
  - Semaphore empty keeps track of how many available slots there are in the buffer
    - Initialized to the value N

Bounded Buffer Problem (Cont.)

Producer Process

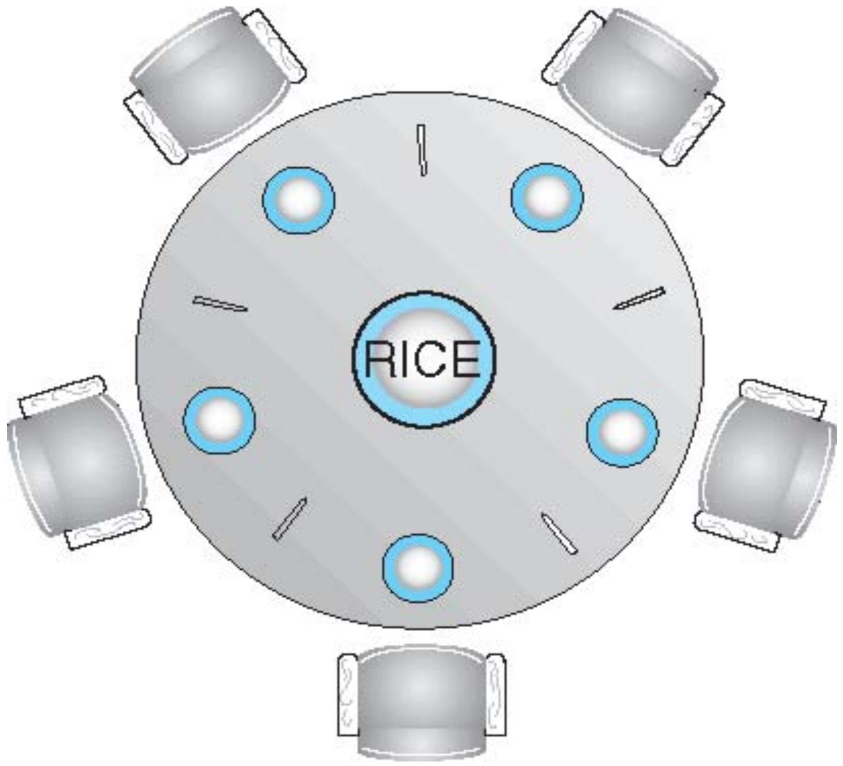
```
do {
    ....
  // produce an item
    ....
  wait (empty); // inited to N
  wait (mutex);
     ....
  // add item to the  buffer
     ....
  signal (mutex);
  signal (full);
} while (TRUE);
```

Consumer Process

```
do {
    ....
  wait (full); // full inited to 0
  wait (mutex);
    ....
  // remove item from buffer to nextc
    ....
  signal (mutex);
  signal (empty);
    ....
  // consume the item in nextc
    ....
} while (TRUE);
```

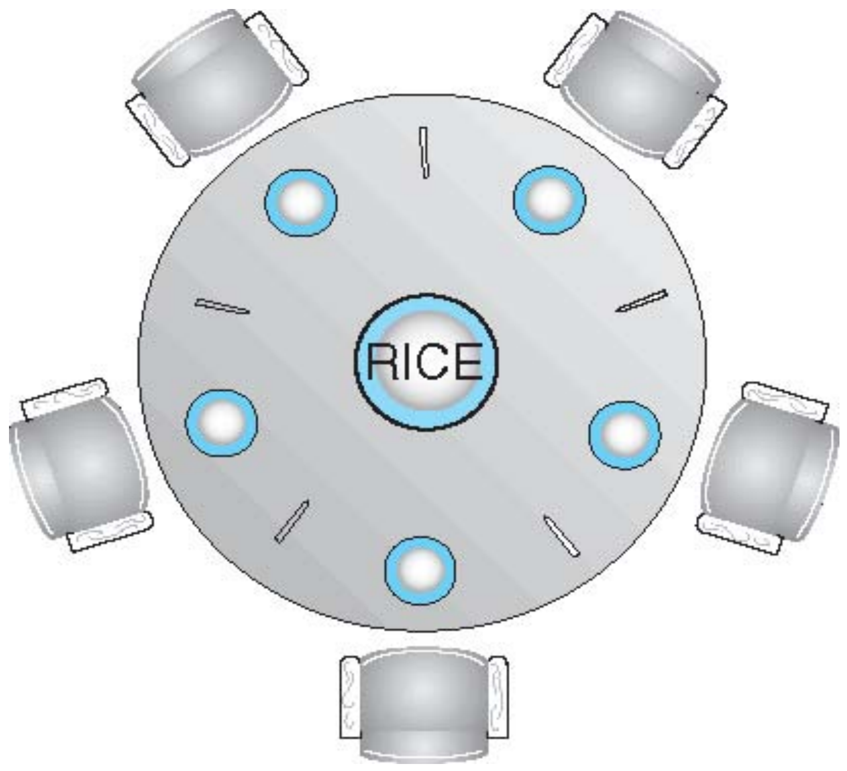
Dining-Philosophers Problem Statement

- **Five philosophers spend their lives thinking and eating while sitting at a round table around a bowl of rice**
- **A chopstick is placed between each philosopher**
- **Philosophers cannot interact with their neighbors**
- **Each philosopher will think and occasionally eat**
    - When ready to eat a philosopher will try to pick up 2 chopsticks (one at a time) so he can eat some rice
    - A philosopher needs 2 chopsticks to eat
    - When done eating a philosopher will put down each chopstick, one at a time
- **How can the philosophers sit and eat together without anyone starving?**
    - Think of each chopstick as a semaphore



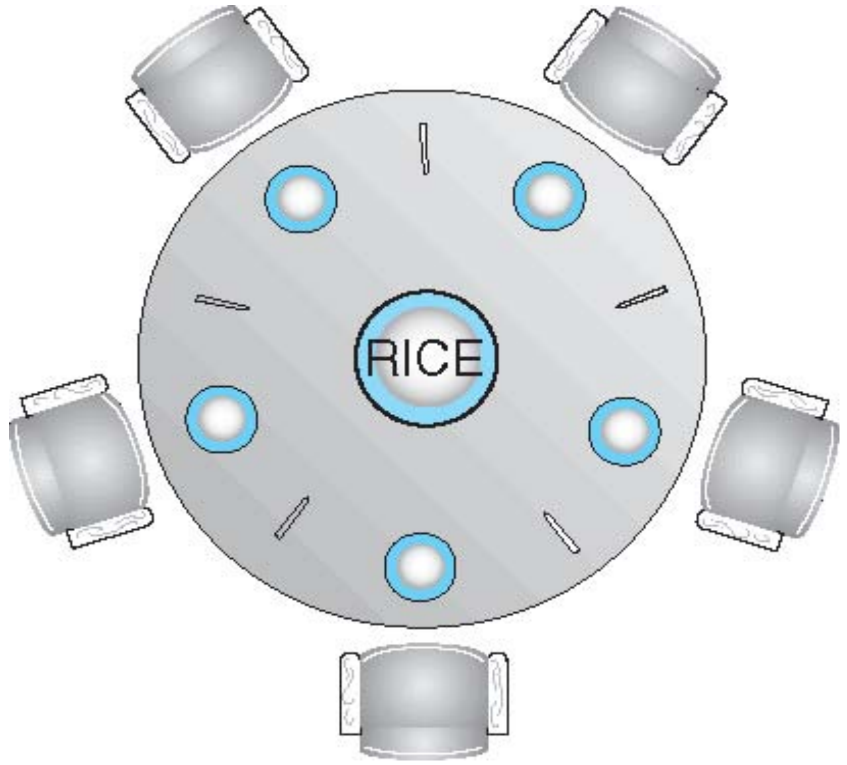
Dining-Philosophers Issues

- **Possible Solution??**
  - Instruct each philosopher to behave as follows:
    - Think until the left chopstick is available; when it is pick it up
    - Think until the right chopstick is available; when it is pick it up
    - Eat some rice
    - Put the left chopstick down
    - Put the right chopstick down
    - Go back to thinking
- **Why might this not work?**



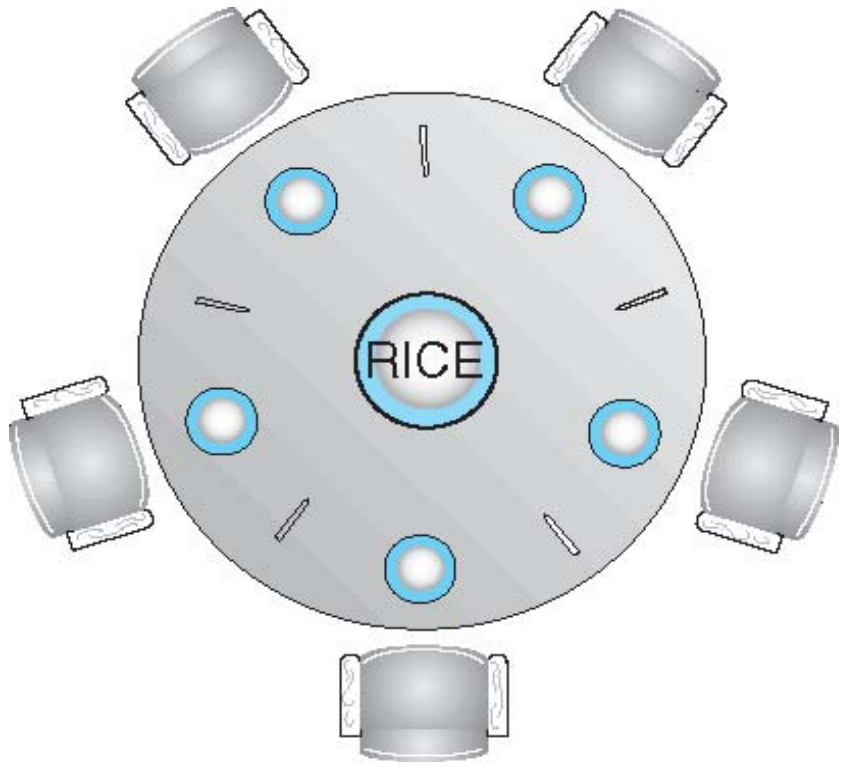
Dining-Philosophers Issues

```
do  {
  wait ( chopstick[i] );
   wait ( chopStick[ (i + 1) % 5] );
     ...
  // eat
    ...
  signal ( chopstick[i] );
   signal ( chopstick[ (i + 1) % 5] );
     ...
  // think
     ...
} while (TRUE);
```



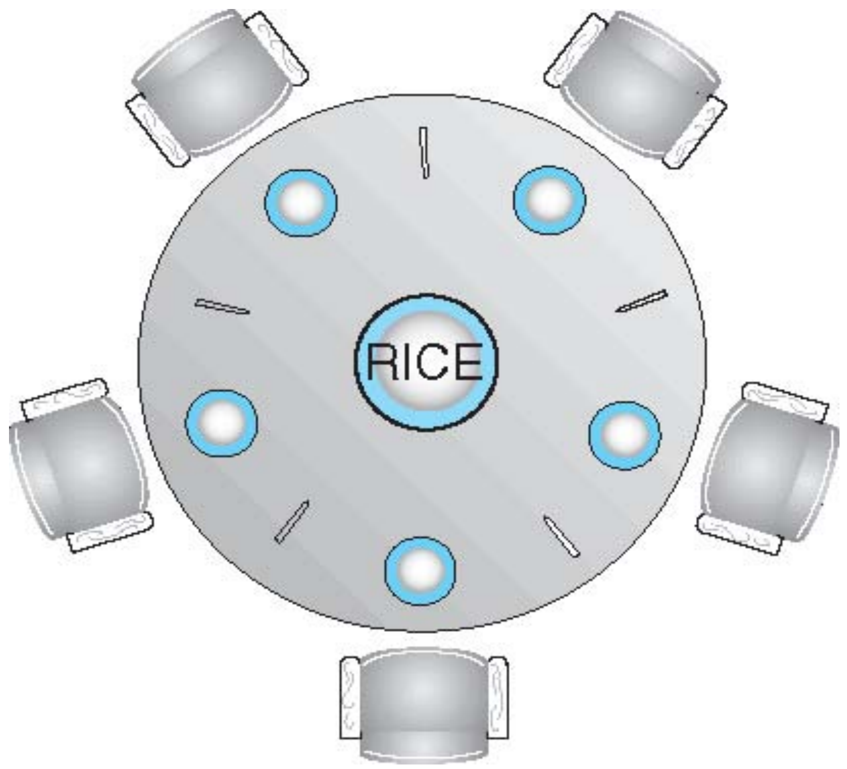
Dining-Philosophers Issues

- **Possible Solution??**
  - Instruct each philosopher to behave as follows:
    - Think until the left chopstick is available; when it is pick it up
    - Think until the right chopstick is available; when it is pick it up
    - Eat some rice
    - Put the left chopstick down
    - Put the right chopstick down
    - Go back to thinking
- **Why might this not work?**
  - If each philosopher picks up his left chopstick at the same time, then they all sit waiting for the right chopstick forever (i.e. deadlock)



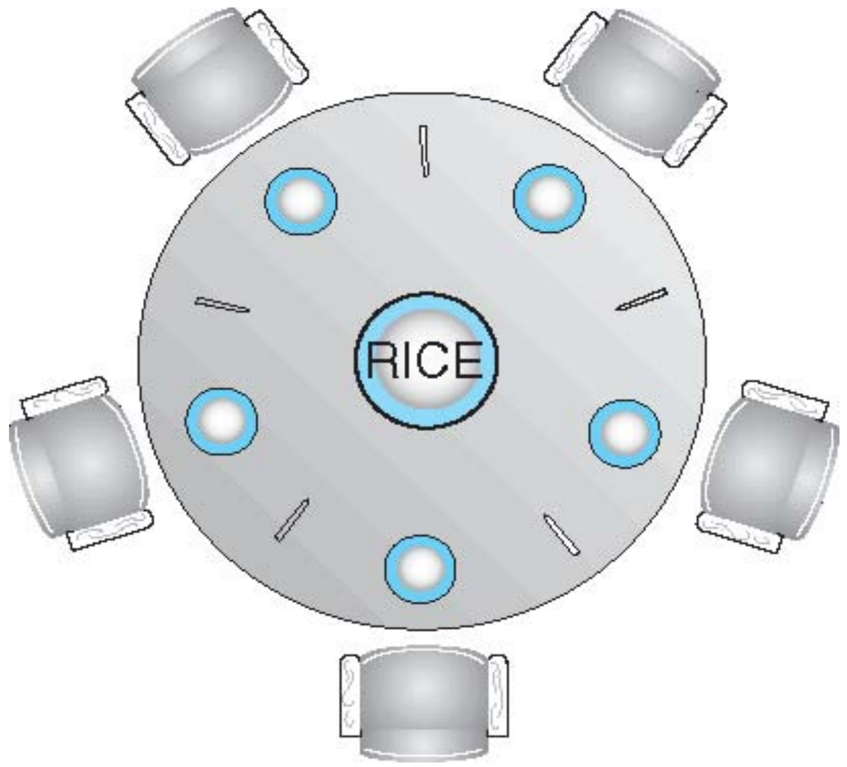
Dining-Philosophers Issues

- **How about telling the philosophers to put down their left chopstick if they've held it for some period of time?**
  - If the timing is just right, the philosophers may end up in **livelock**
    - The philosophers continue to alternate between thinking and (attempting) to eat, but if the philosophers all try to pick up their chopsticks at the same time, then they will all put them back down at the same time. This cycle will continue until the philosophers starve



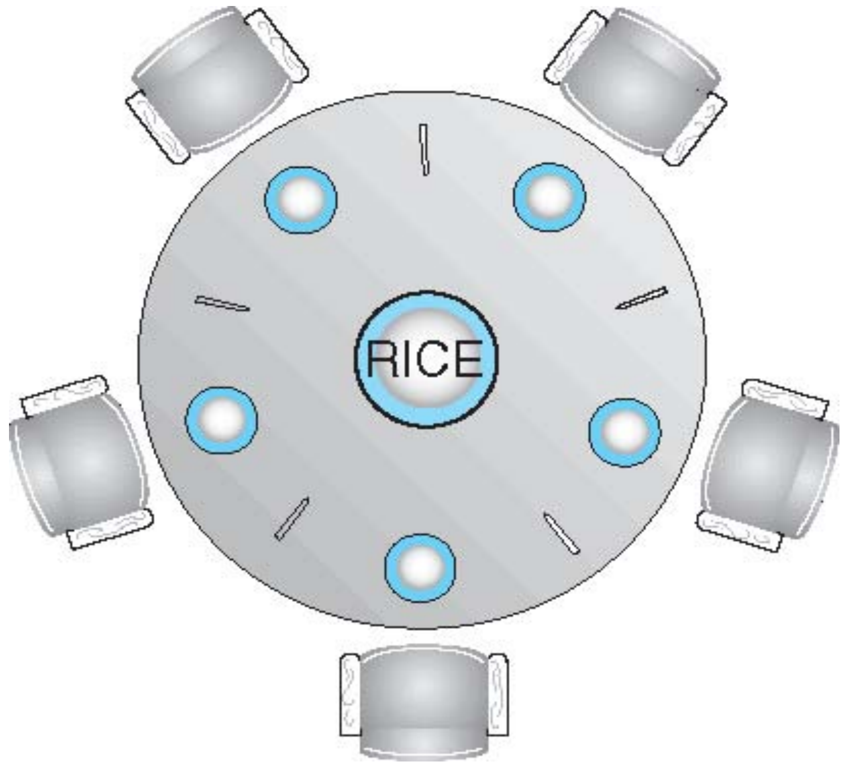
Dining-Philosophers Solutions

- **The Conductor Solution**
  - Add a waiter to the table
  - A philosopher must ask the waiters permission before picking up any chopsticks
  - The waiter knows how many chopsticks are in use and can therefore arbitrate and ensure that deadlock does not occur



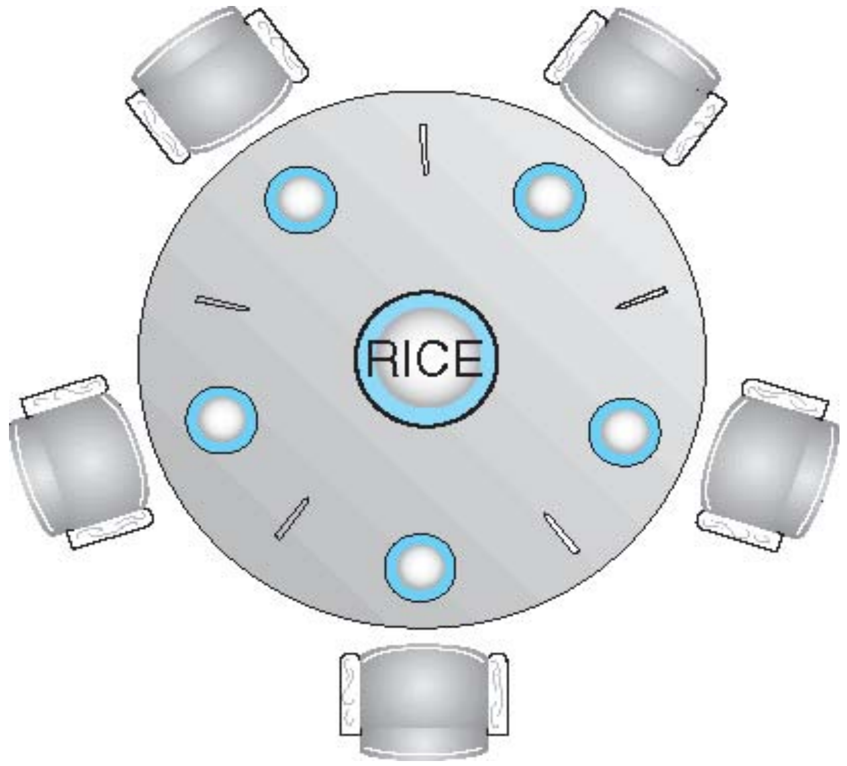
Dining-Philosophers Solutions

- **The Chandra-Misra Solution**
  - Does not require a waiter or any other central authority
  - Breaks the rules a little bit and allows philosophers to talk to their neighbors
  - Proposed Solution:
    - For every pair of philosophers contending for a chopstick create a chopstick and give it to one of them
    - When a philosopher wants to eat, he must obtain two chopsticks, so he requests one chopstick from each of his neighbors
      - The philosopher may already have one and therefore will only need to request one
    - When a philosopher with a chopstick gets a request from his neighbor he keeps the chopstick if it is clean, but cleans it and gives it up if it is dirty
    - After a philosopher has eaten, both his chopsticks are dirty. If another philosopher had requested his chopstick, he cleans the chopstick and passes it over.



Dining-Philosophers Solutions

- **The Chandra-Misra Solution (Cont.)**
  - This solution can be used for an arbitrarily large number of philosophers
  - Solves the problem of starvation with the clean/dirty chopsticks
    - Gives preference to the most ‘starved’ philosopher
    - A philosopher that has eaten must give up his chopsticks



Problems with Semaphores

- **Incorrect use of semaphore operations (bad programming ... maybe by someone else):**
  - `signal(mutex)` …. `wait(mutex)` -- wrong order
  - `wait(mutex)` … `wait(mutex)` -- decrement TWO resources
  - Omitting of `wait(mutex)` or `signal(mutex)` (or both) -- UGH
- **Deadlock and starvation**