CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Dining-Philosophers Problem

Look at another well-studied synchronization problem

Previously saw

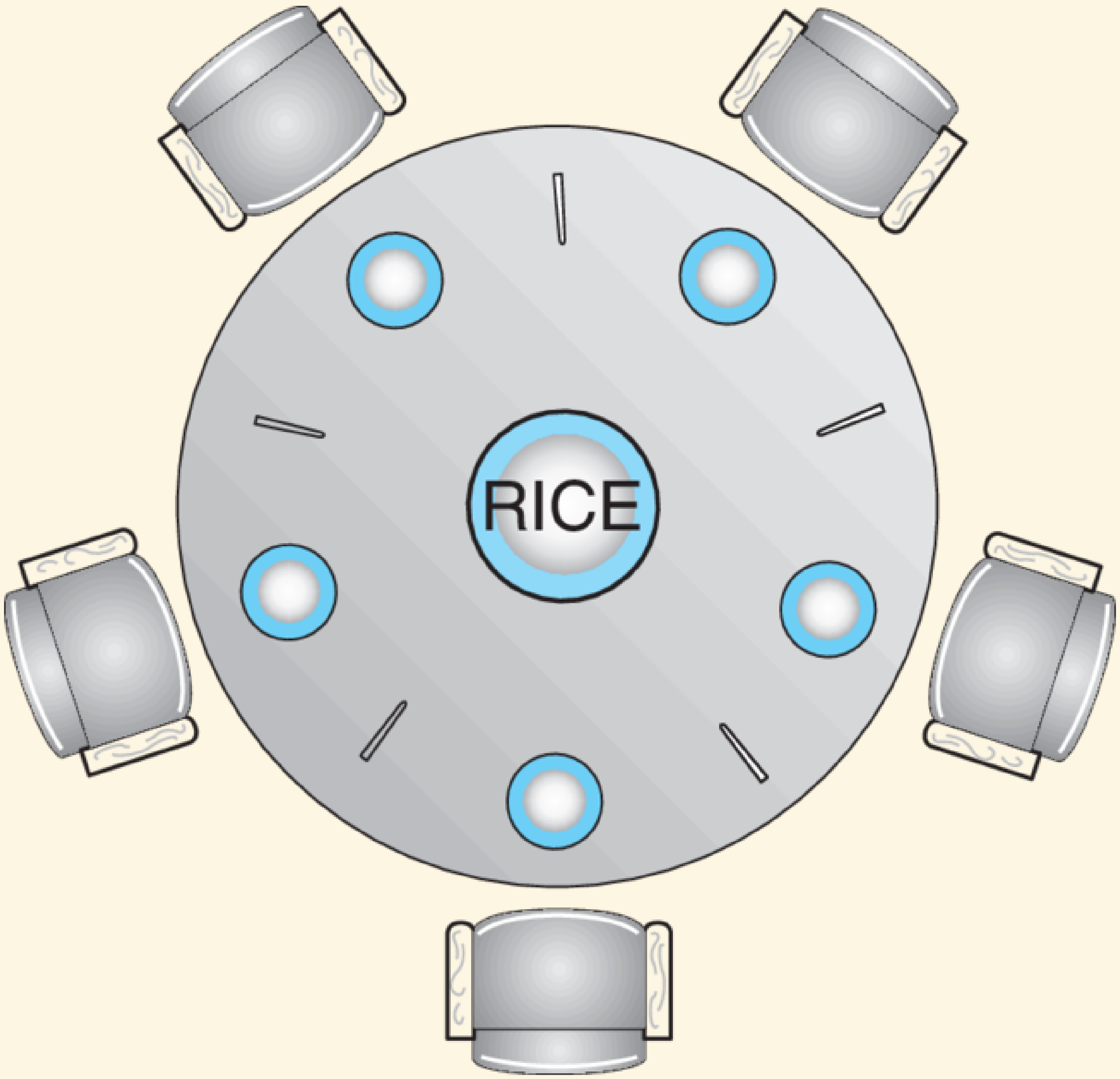
- Producer-Consumer Problem
- Readers-Writers Problem

Dining-Philosophers Problem

Imagine five philosophers seated at a circular table

All these philosophers do is think and eat rice, but not at the same time

Unfortunately, there is a chopstick shortage -- only one chopstick is available per philosopher


RICE

Rules

When philosopher is thinking, they do not interact with other philosophers

When philosopher is hungry, they will try to pick up chopsticks to their left and right and eat

Only one chopstick can be picked up at a time

Philosophers are too polite to steal a chopstick that is already in use

Rules

If a philosopher succeeds in aquiring both chopsticks at once, they will eat without putting them down

Once they are full, they put down both chopsticks and go back to thinking

Sounds silly, but is a very famous problem

Represents a large class of problems where there is a need to allocate *several resources* among several processes

Other examples we have looked at had one shared resource

Good solutions must avoid deadlock and starvation

Representation

Each chopstick is a semaphore -- shared data is

```
semaphore chopstick[5];
```

Grabbing a chopstick is executing a `wait` on that particular semaphore, and setting it down is executing a `signal` on that particular semaphore

All semaphores initialized to 1 (chopsticks are on the table)

Bad solution

Philosopher i

```
do {
    wait(chopstick[i]);
    wait(chopstick[(i+1) % 5]);
    ...
    /* eat for awhile */
    ...
    signal(chopstick[i]);
    signal(chopstick[(i+1) % 5]);
    ...
    /* think for awhile */
    ...
} while (true);
```

If all philosophers get hungry at once, can result in deadlock

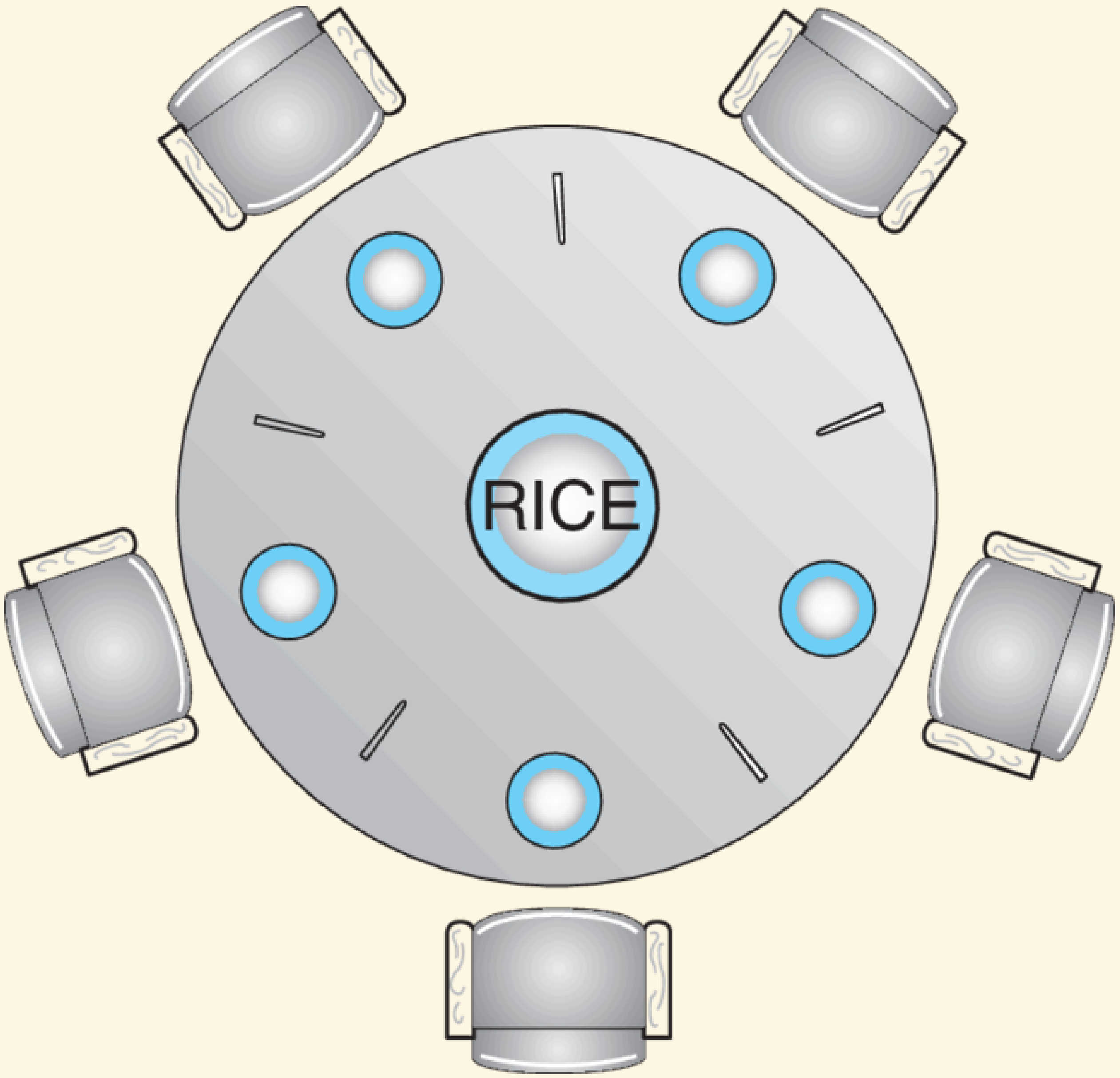
Would be as if each one grabbed the chopstick to their left, then waited forever

Can get around deadlock by using asymmetric solution

Odd-numbered philosopher always picks up left chopstick first, and even-numbered philosopher always picks up right chopstick first

If a process is waiting and already holding a chopstick, we know that the chopstick it is waiting on will eventually be released

Neighboring process would only have grabbed that chopstick if it already had the other


RICE

This solution eliminates deadlock

Also eliminates starvation -- a philosopher who wants to eat will eventually be able to do so (though the wait is not bounded)

Important to keep in mind that eliminating deadlock does not necessarily eliminate starvation. You need to consider both

Starvation solution

For example, if you put one of the philosophers in a straightjacket so that they cannot grab a chopstick, you have solved the issue of deadlock without using right-left solution

With five chopsticks for four philosophers, at least one of them will always be able to eat

However, one of the philosophers will starve