TRƯỜNG ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Operating system- IT3070E

Dining Philosopher Problem

Members:

- Chu Trung Anh 20225564
- Tran Nam Tuan Vuong 20225540
- Vu Minh Hieu 20225494

Supervisor:

- Dr. Do Quoc Huy

ONE LOVE. ONE FUTURE.

Team assignment

Member	Role
Chu Trung Anh	Resource hierarchy, Semaphore
Vu Minh Hieu	Limit number of dinners
Tran Nam Tuan Vuong	Chandy - Misra

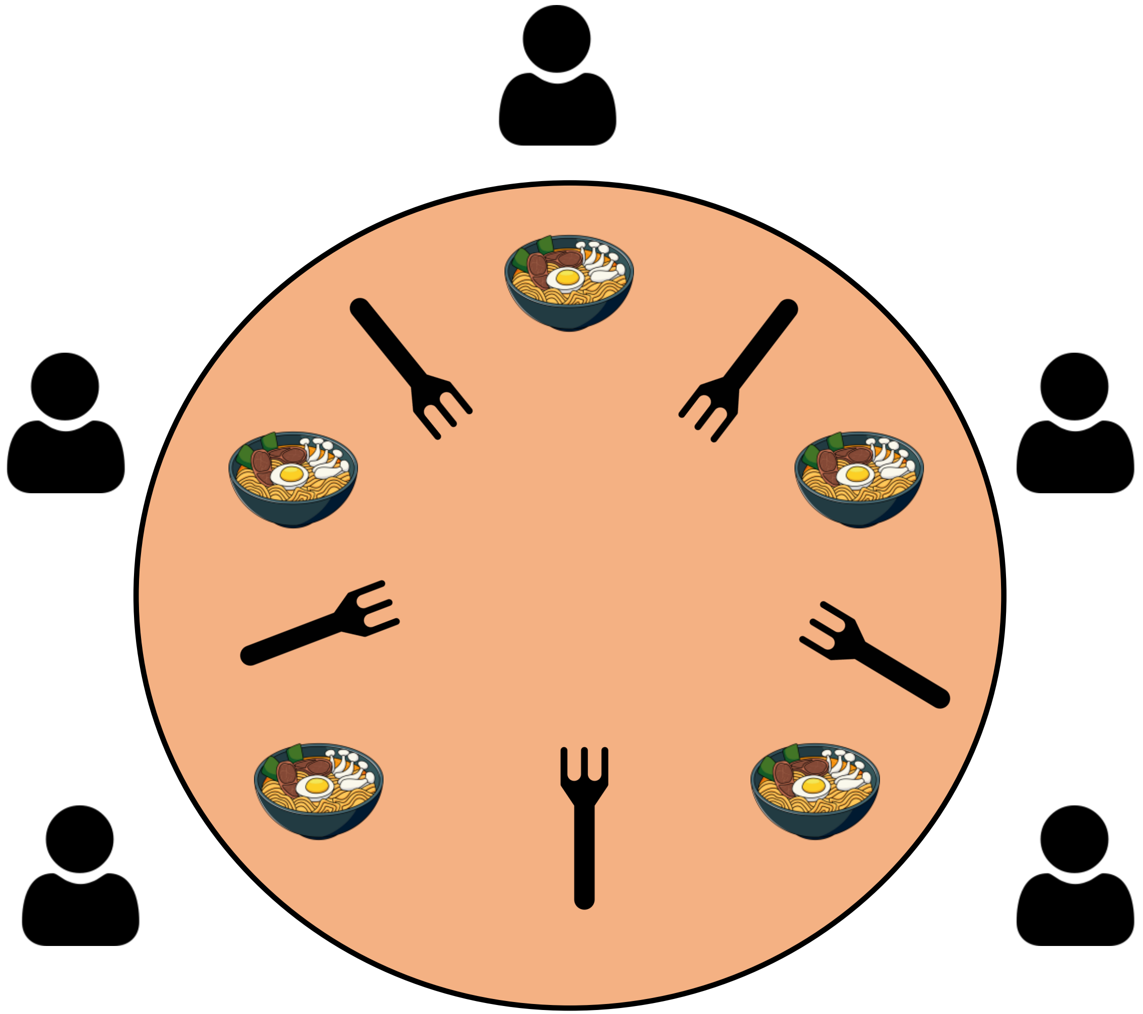
- **Slide and report are made by members with their corresponding task**

- **Report link:** *https://typst.app/project/rZjfADUA7rh7weCDnGpC6m*

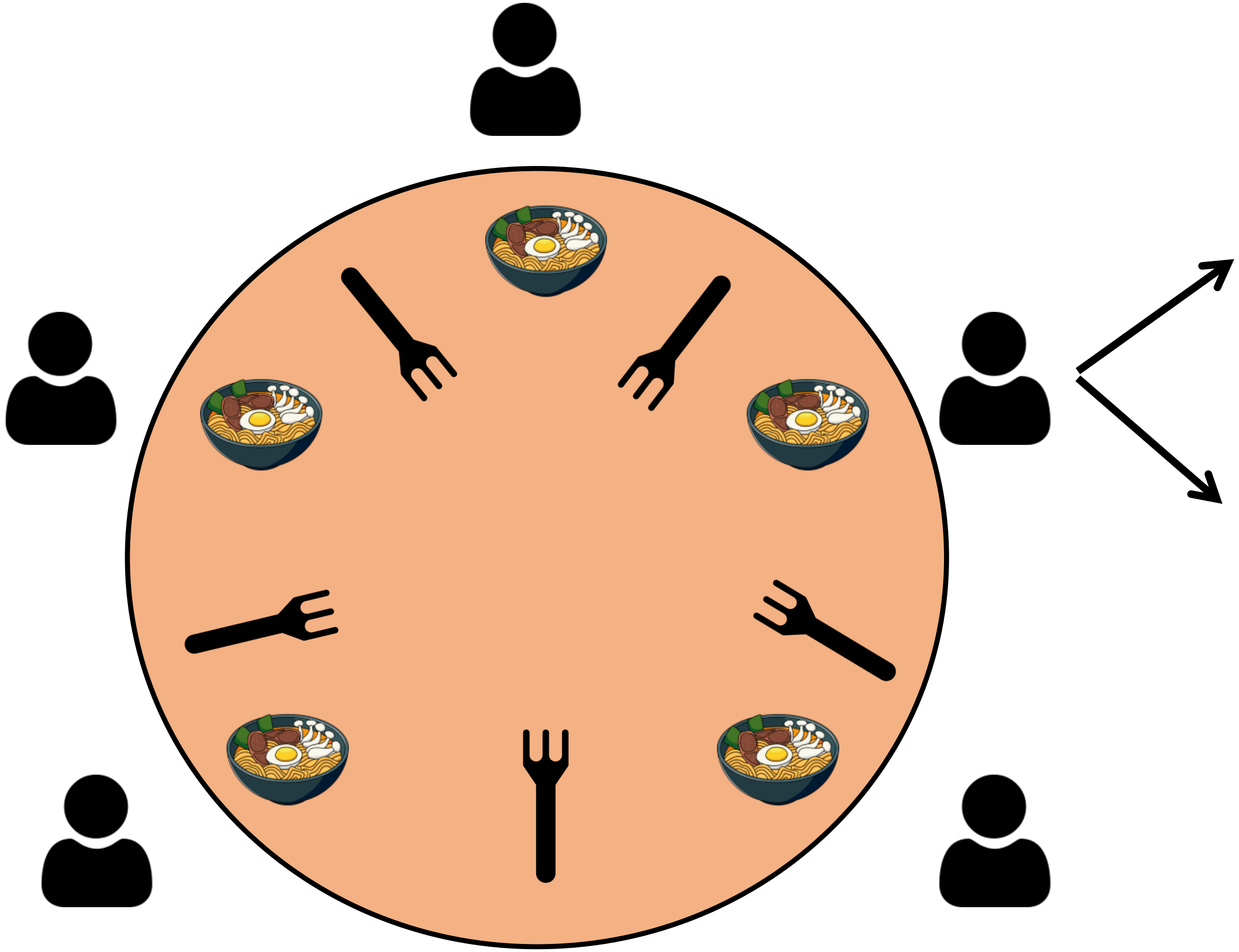
Table of content

1. **Problem**
2. **Challenge**
    - Deadlock
    - Livelock
    - Starvation
3. **Solution**
    - Resource Hierarchy
    - Semaphore
    - Limit number of dinners
    - Chandy/Misra

1. Problem



1. Problem



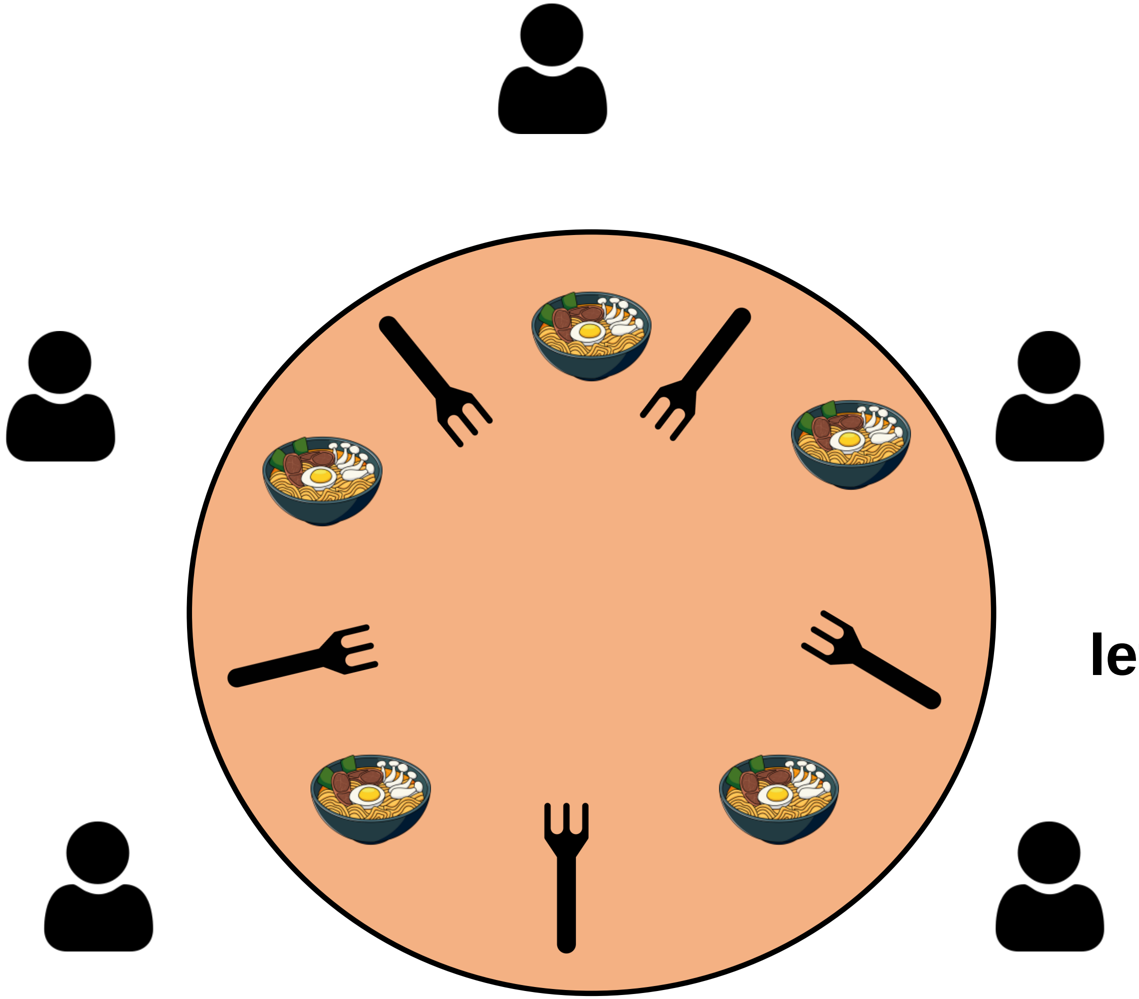
Think

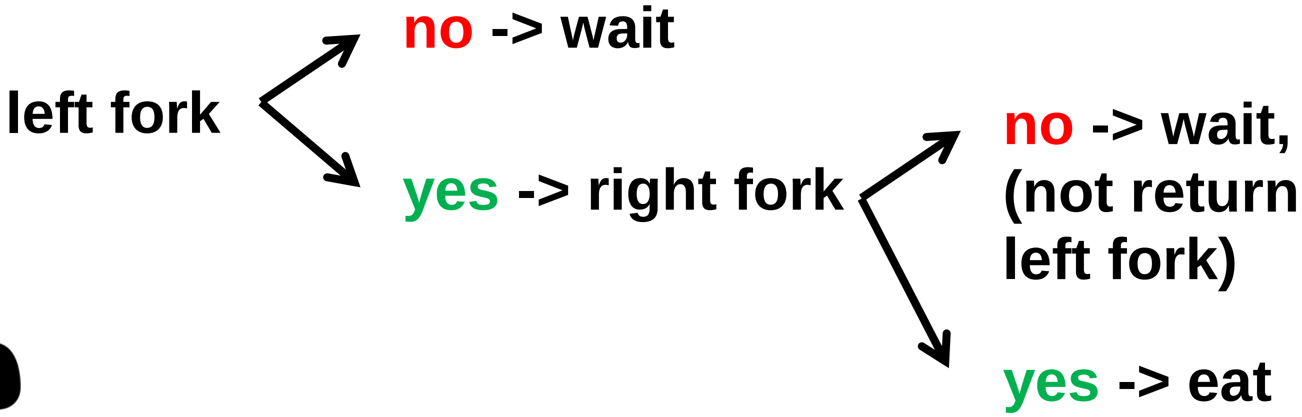
Eat:

– Two forks to eat, take the left fork then the right fork

– Finish eating, return the fork to original place

1. Problem





left fork

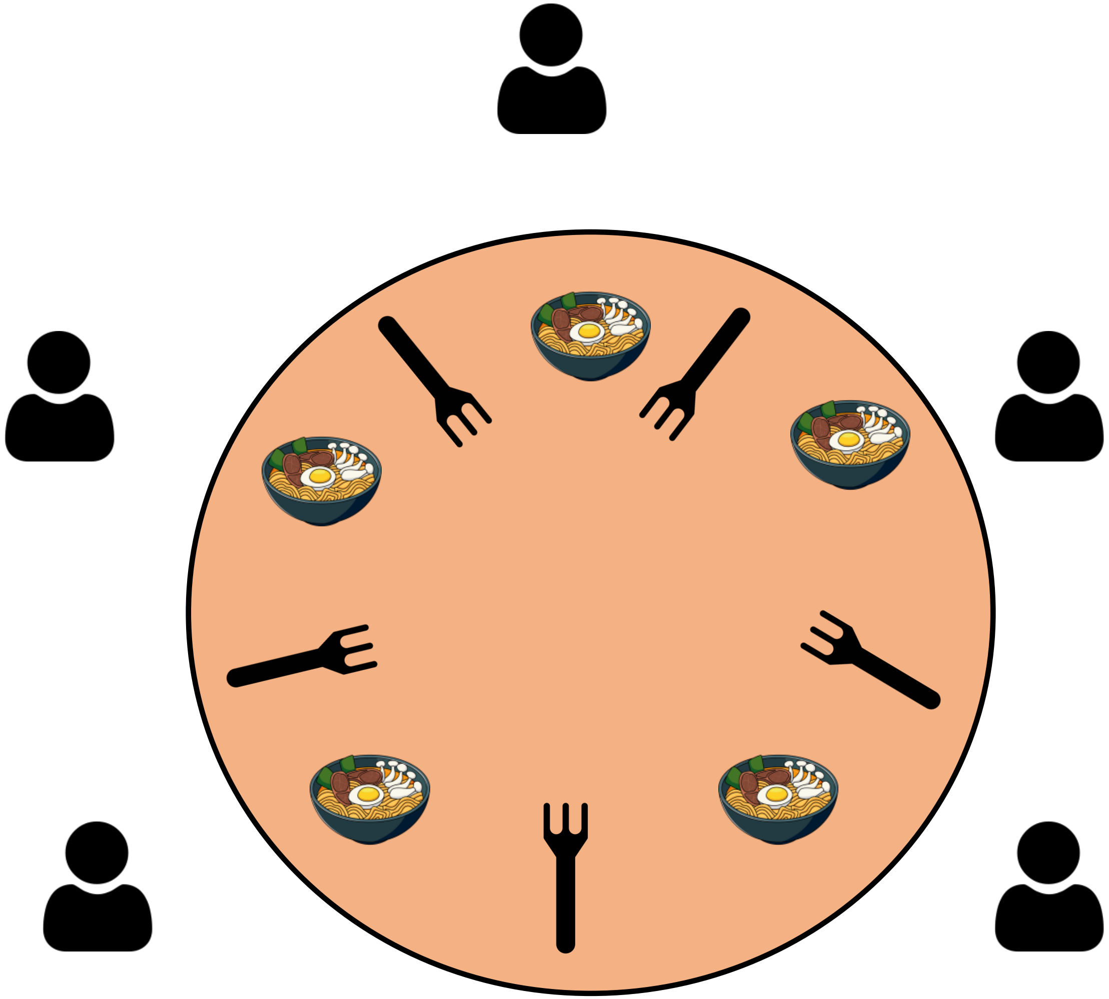
no -> wait

yes -> right fork

no -> wait, (not return left fork)

yes -> eat

1. Problem



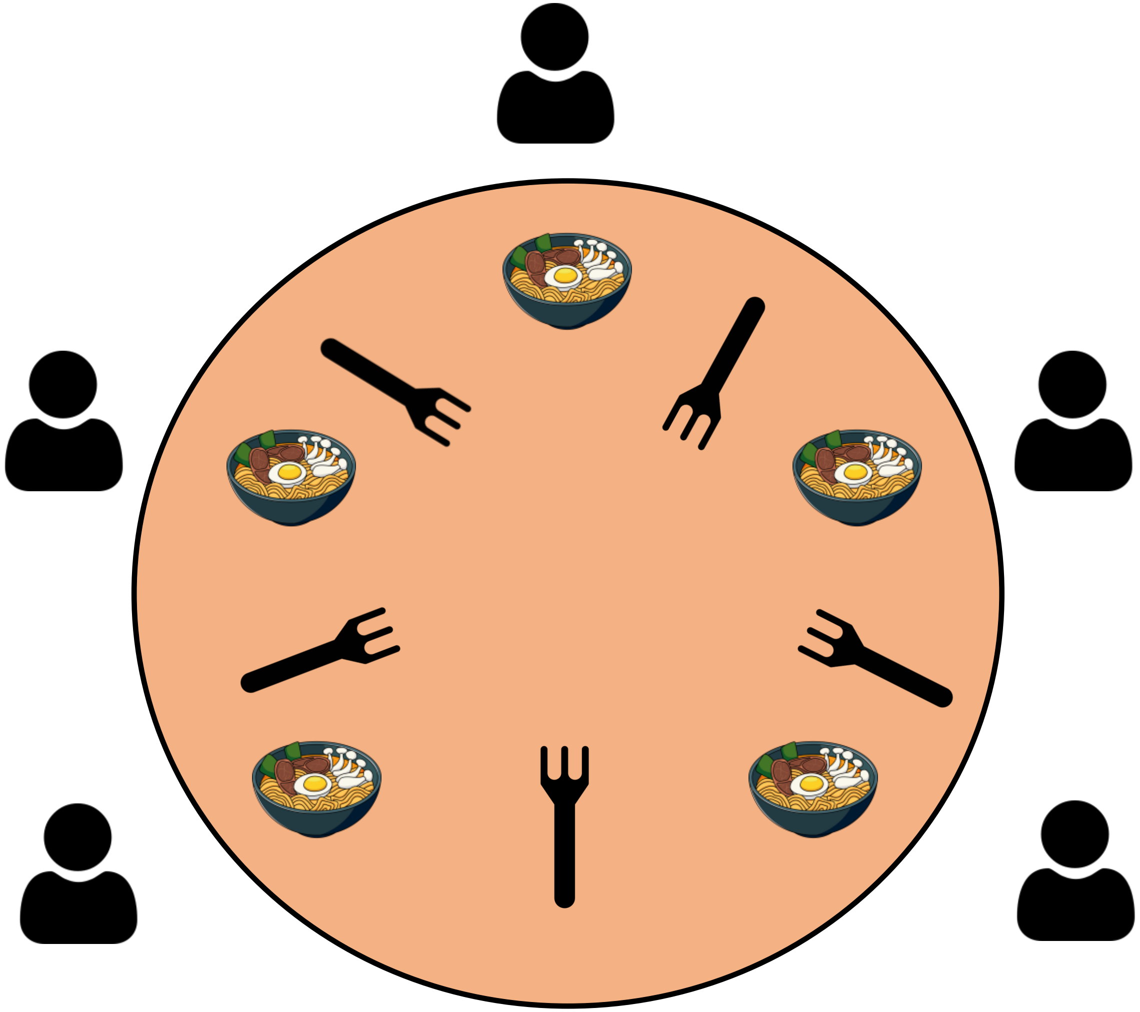
5 critical resources

- sharing limit: 1

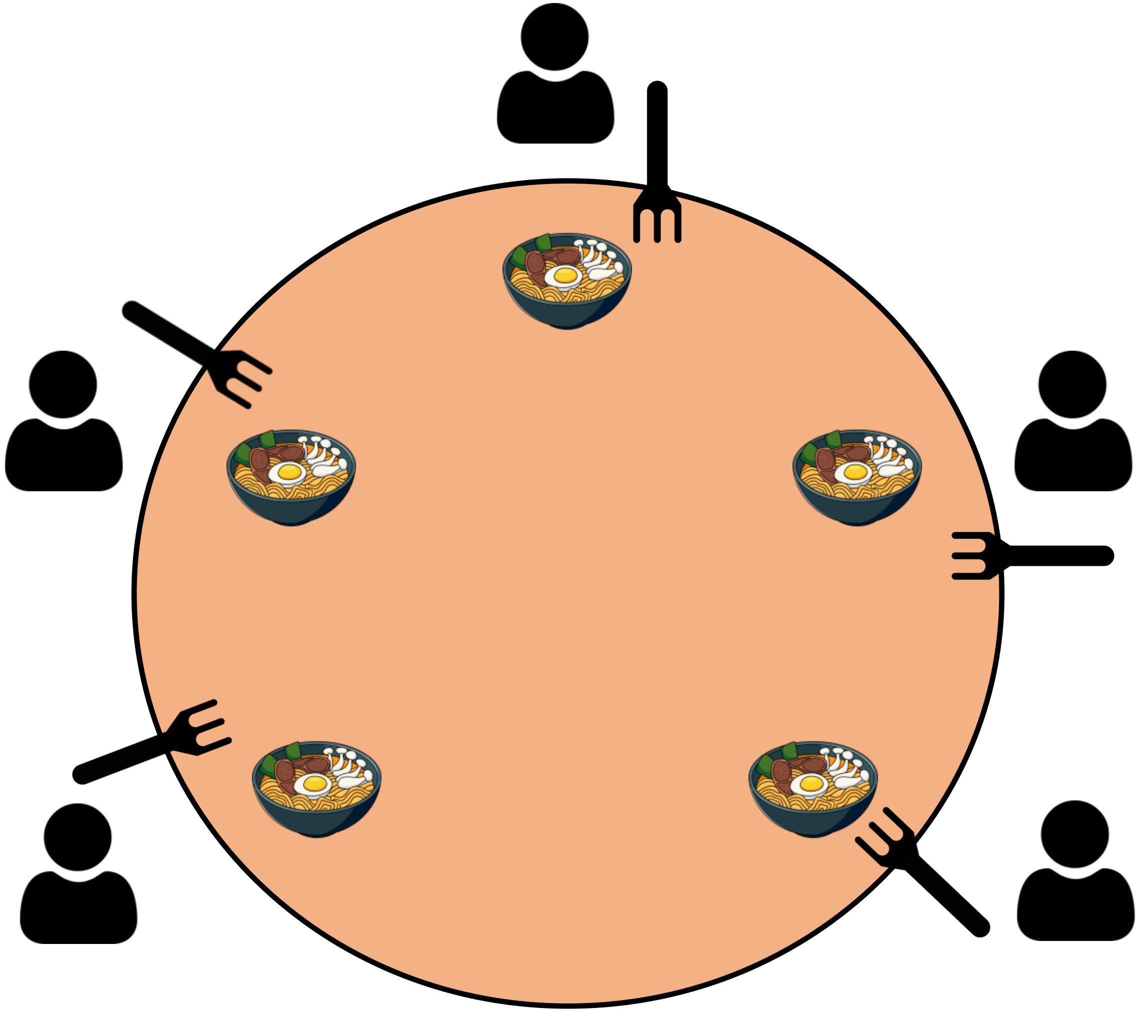
5 processes

- Critical section: using fork

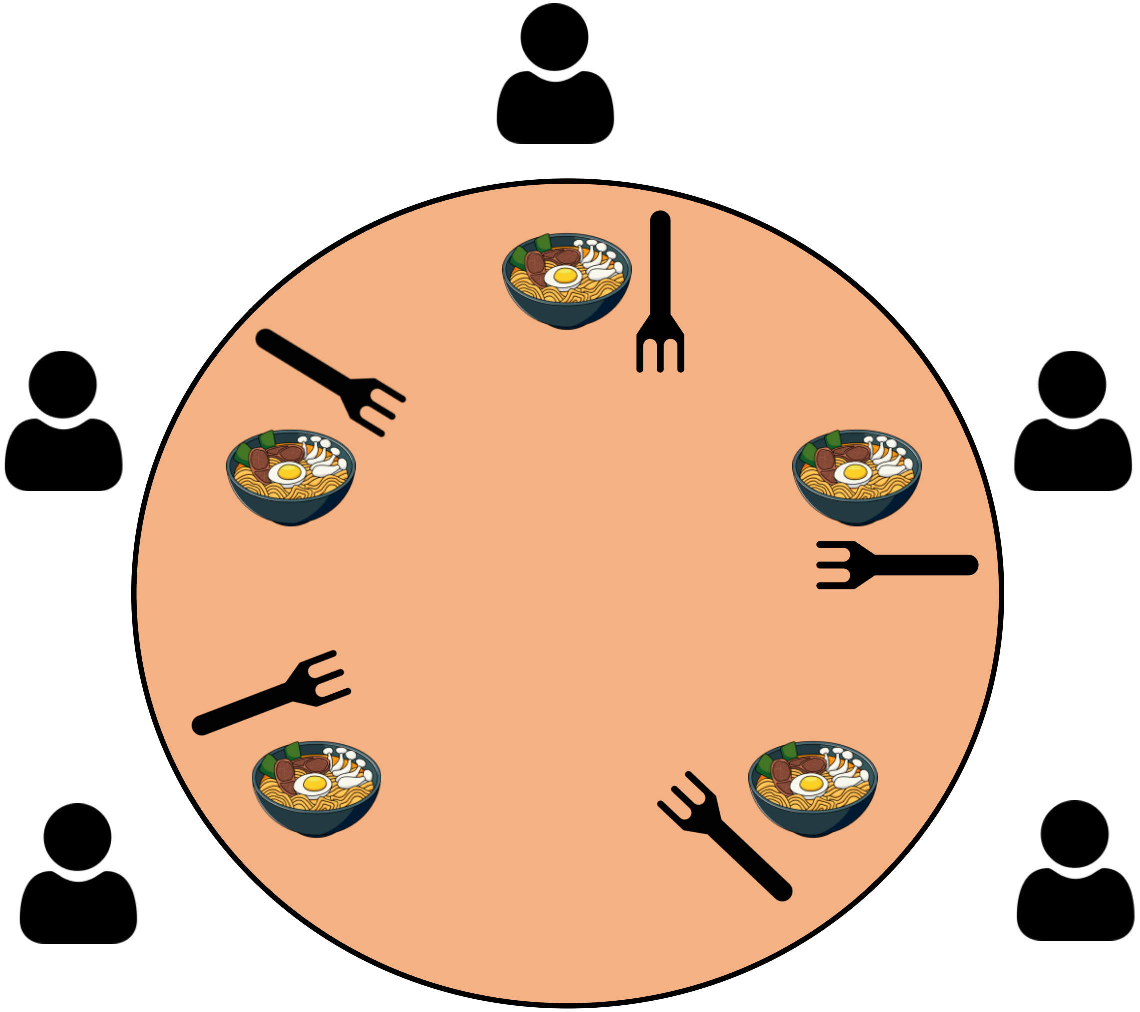
2. Challenge - deadlock



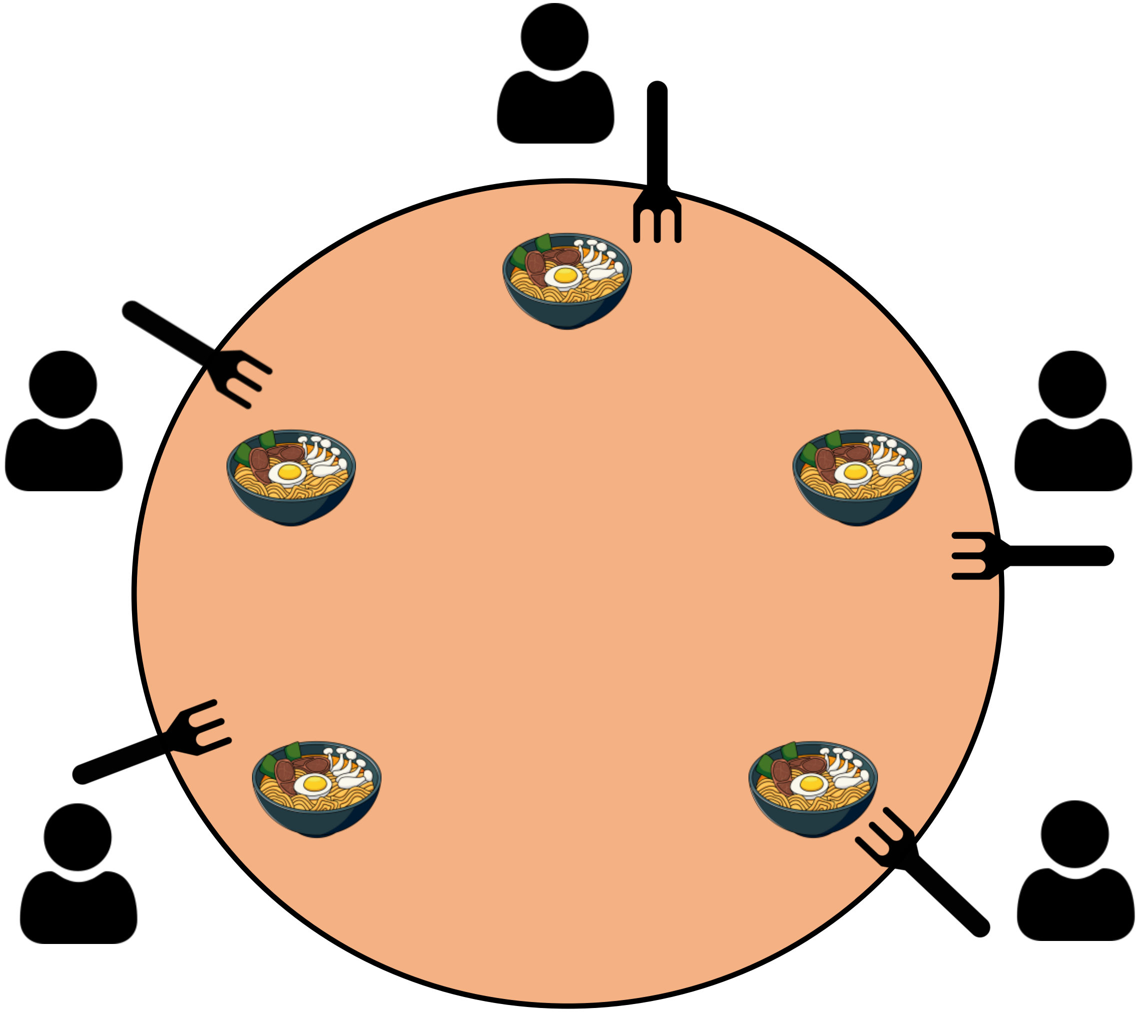
2. Challenge - deadlock



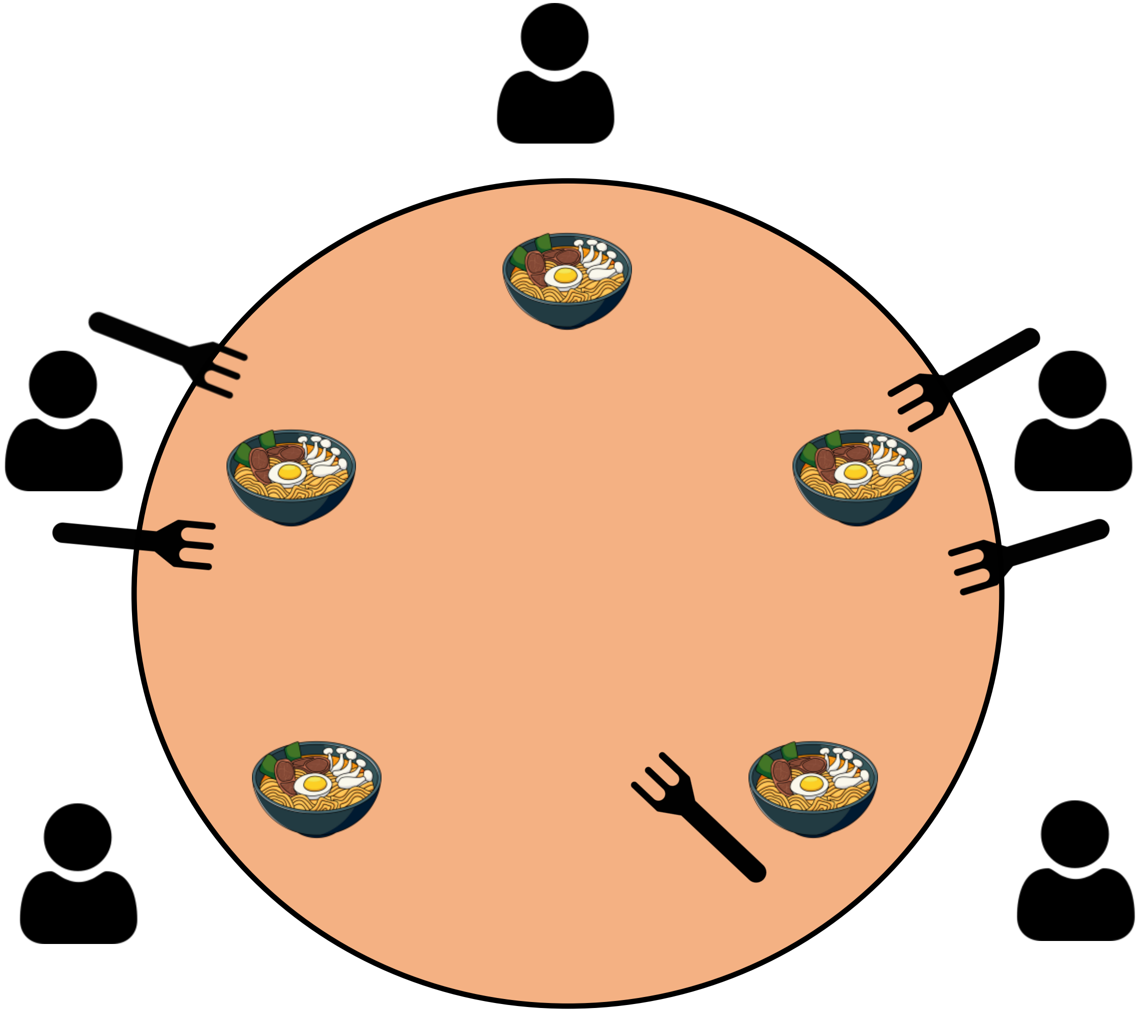
2. Challenge - livelock



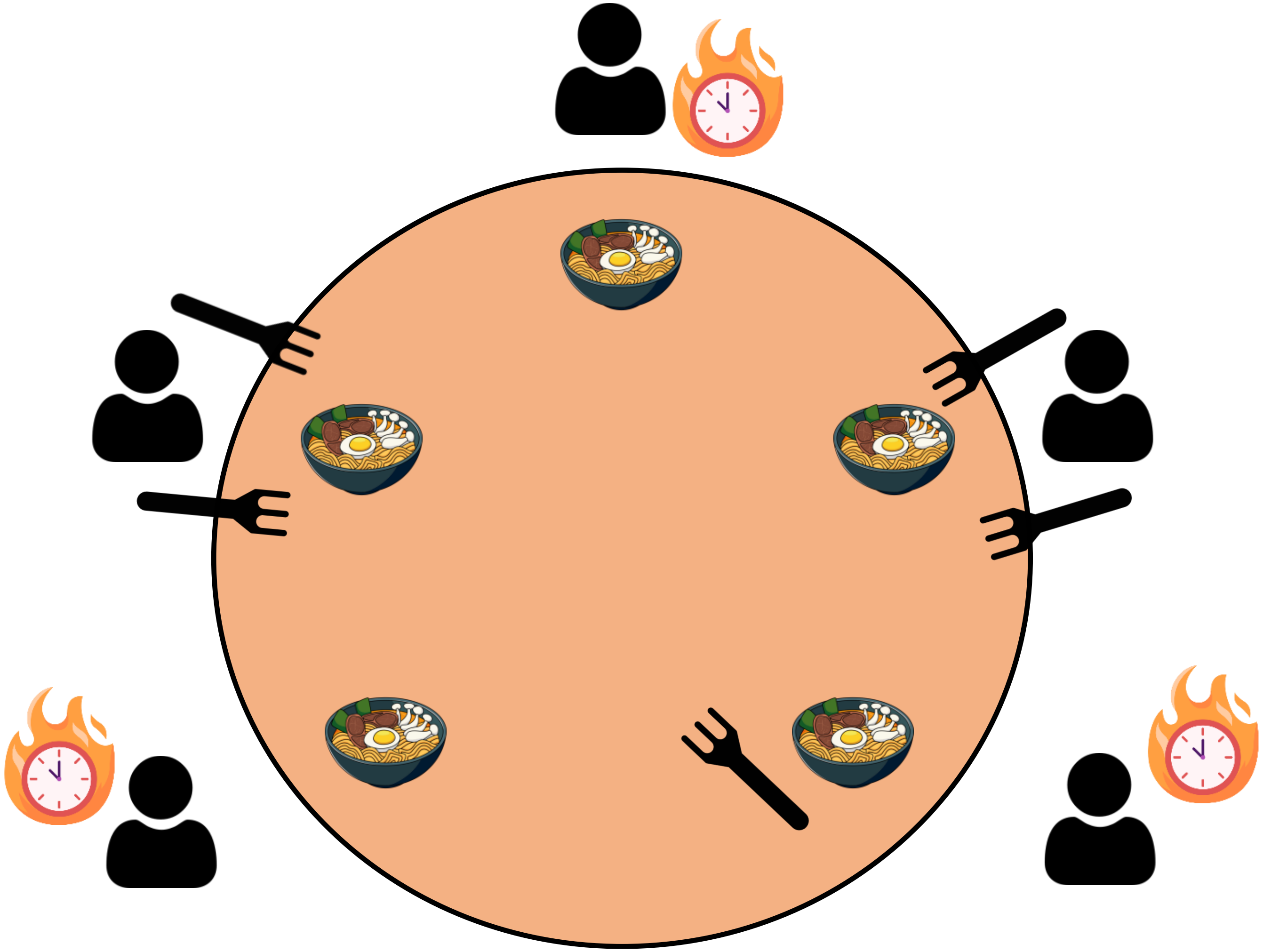
2. Challenge - livelock



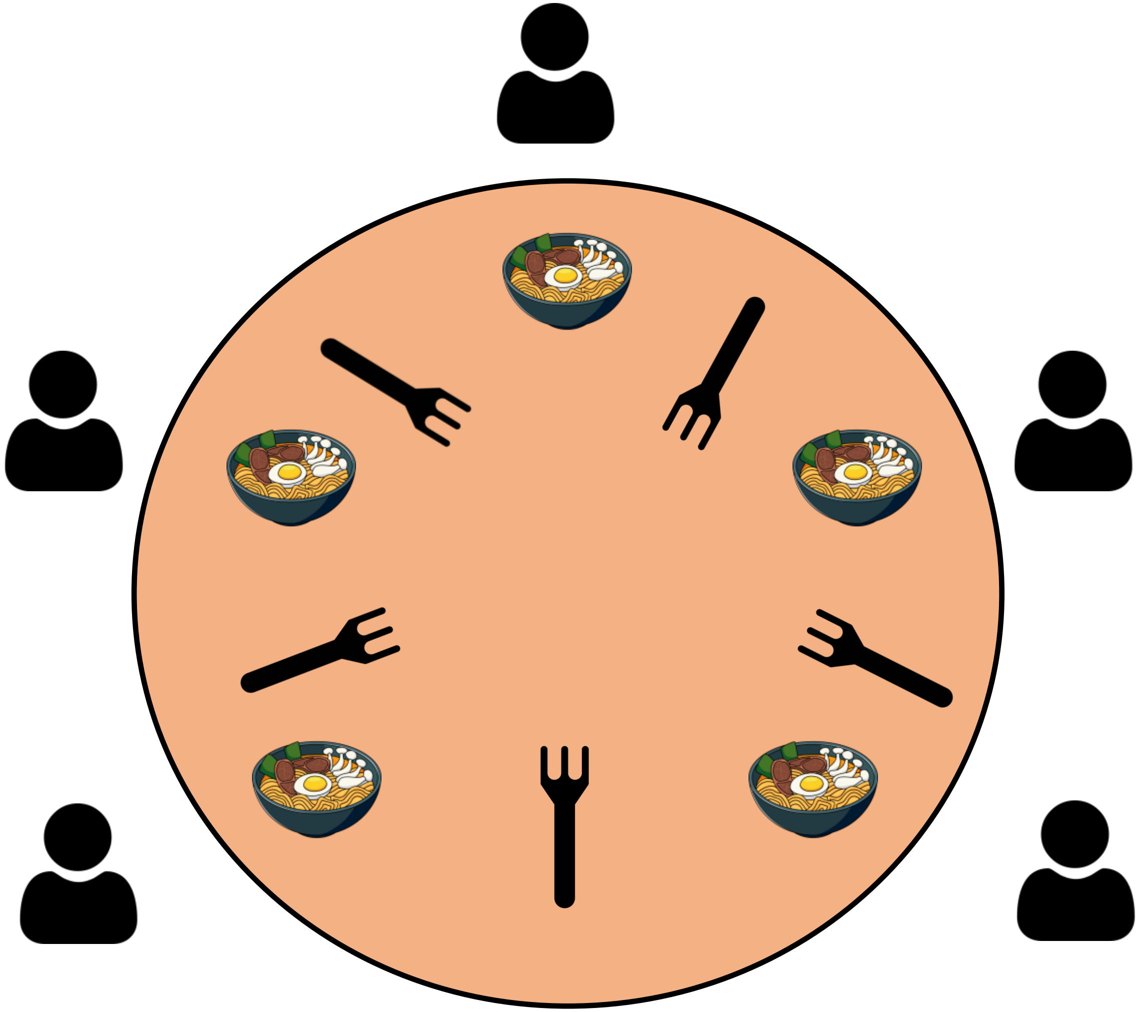
2. Challenge - starvation



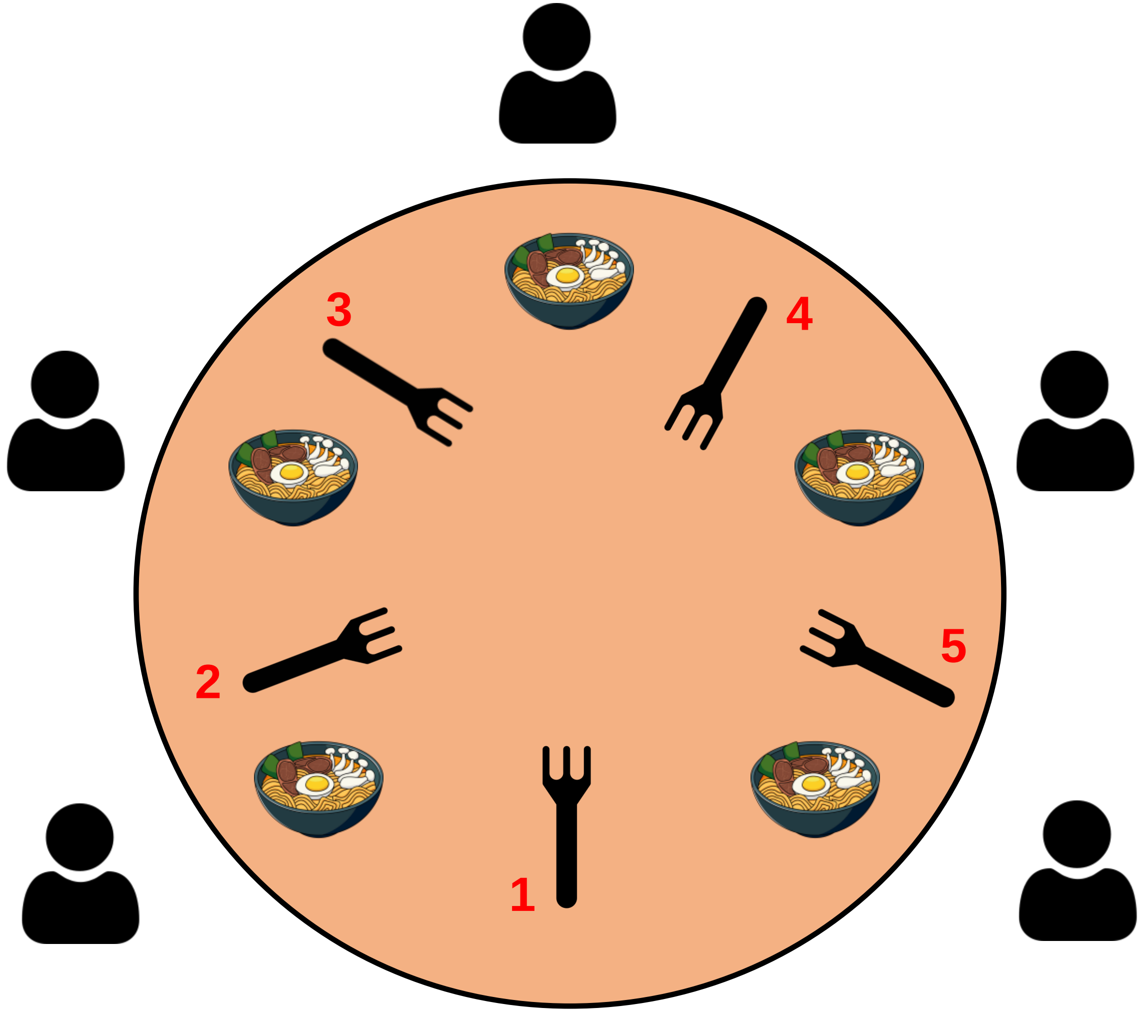
2. Challenge - starvation



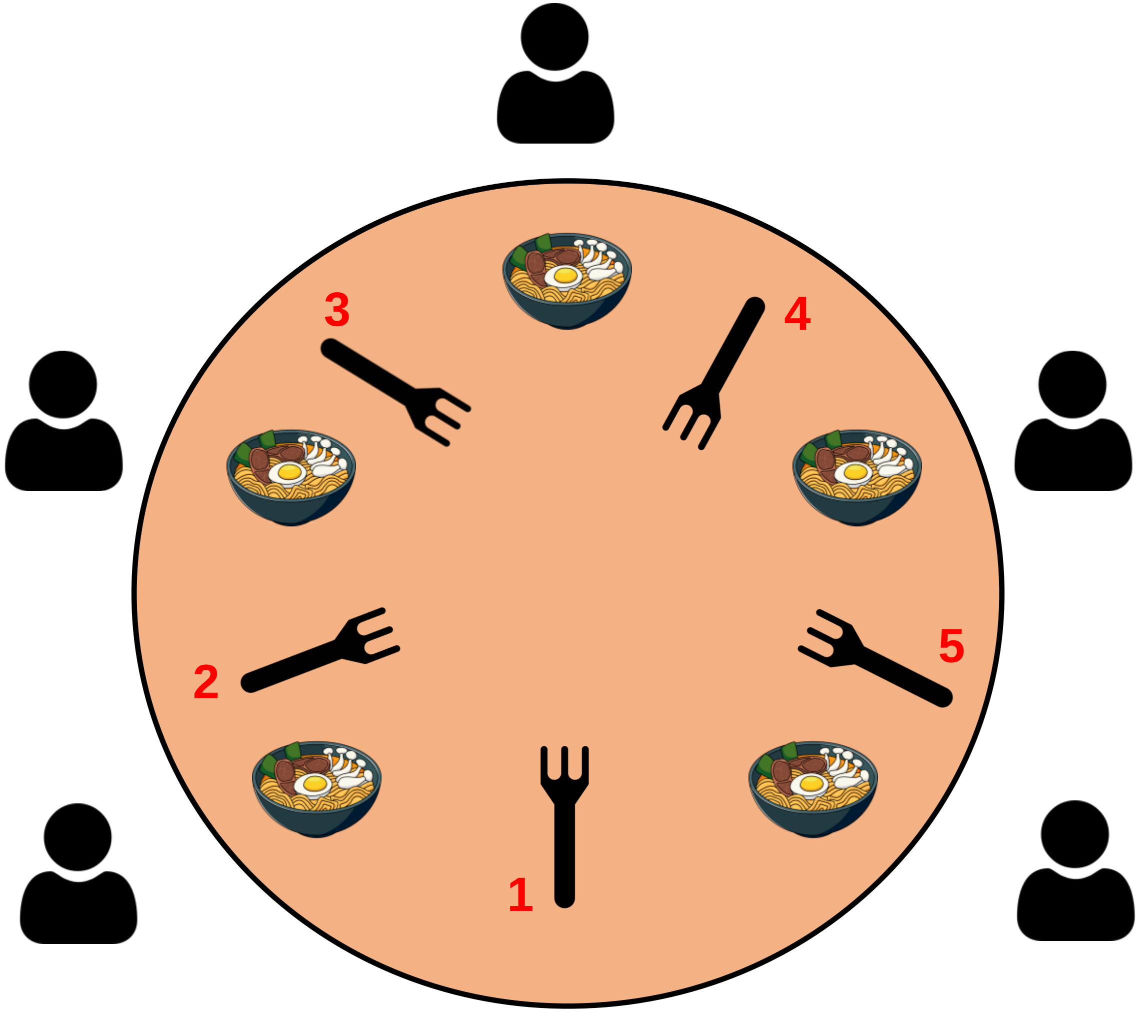
3. Solution - Resource hierarchy



3. Solution - Resource hierarchy



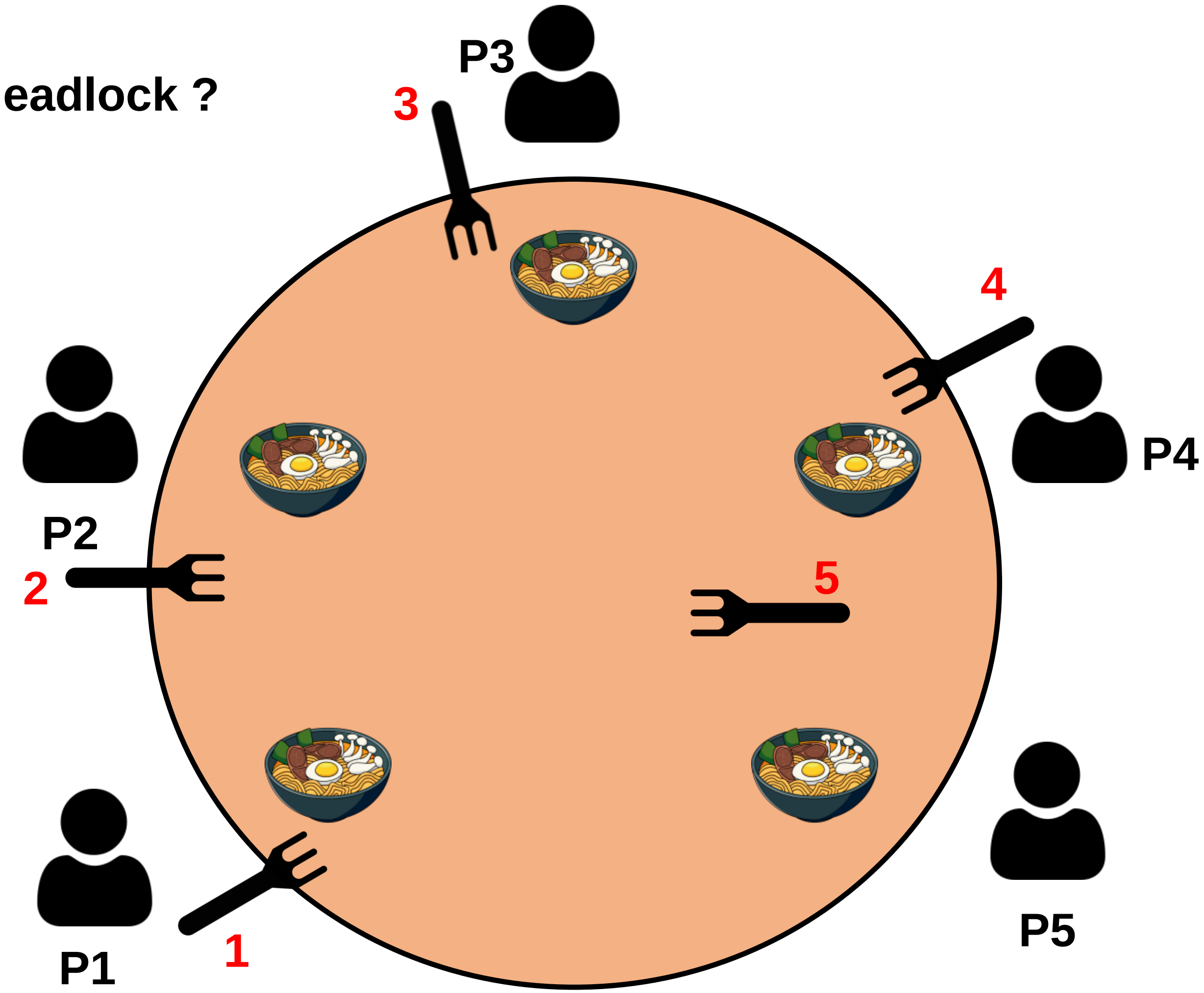
3. Solution - Resource hierarchy



Philosopher takes the lower-numbered fork first

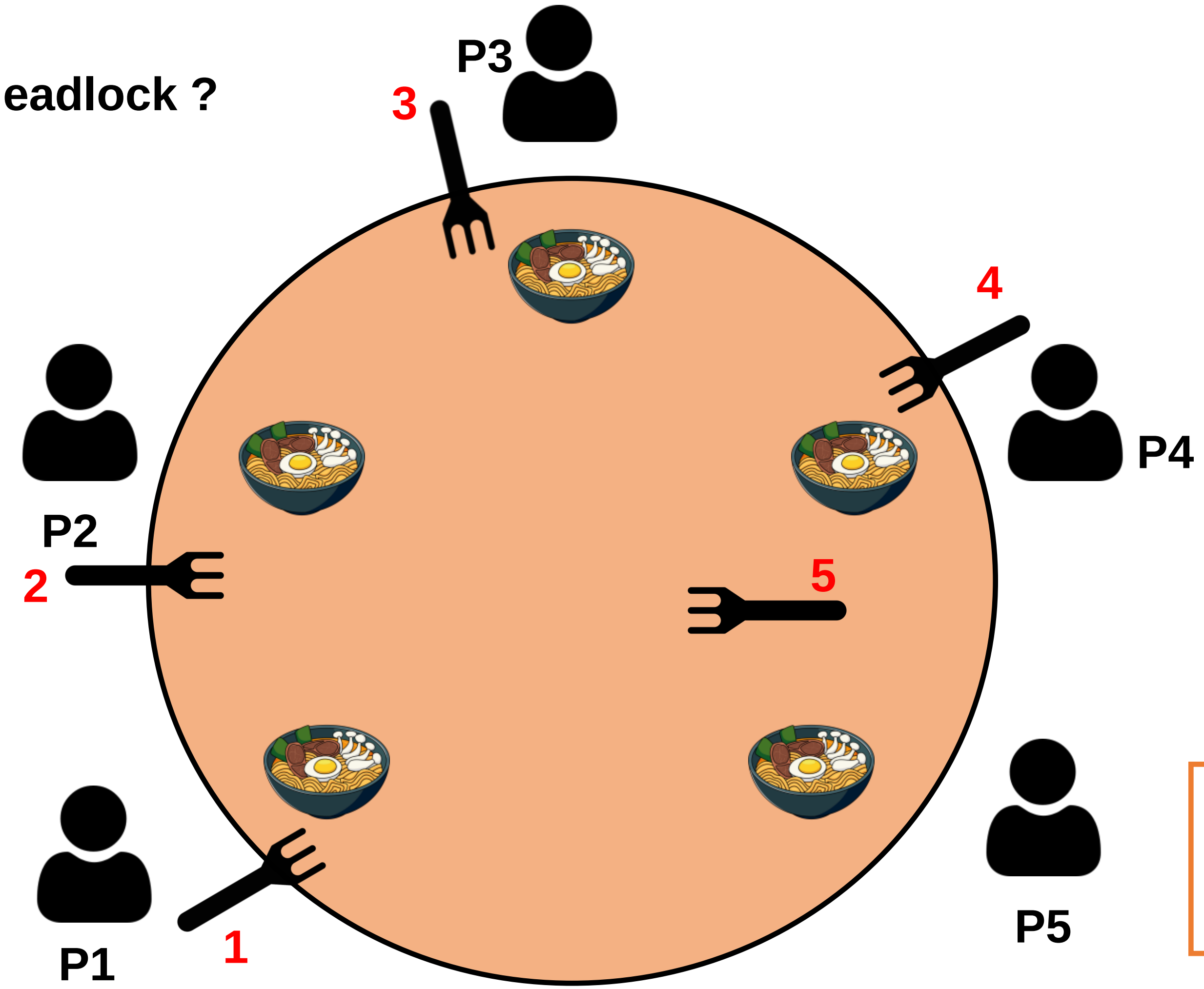
3. Solution - Resource hierarchy

- How this can prevent deadlock ?



3. Solution - Resource hierarchy

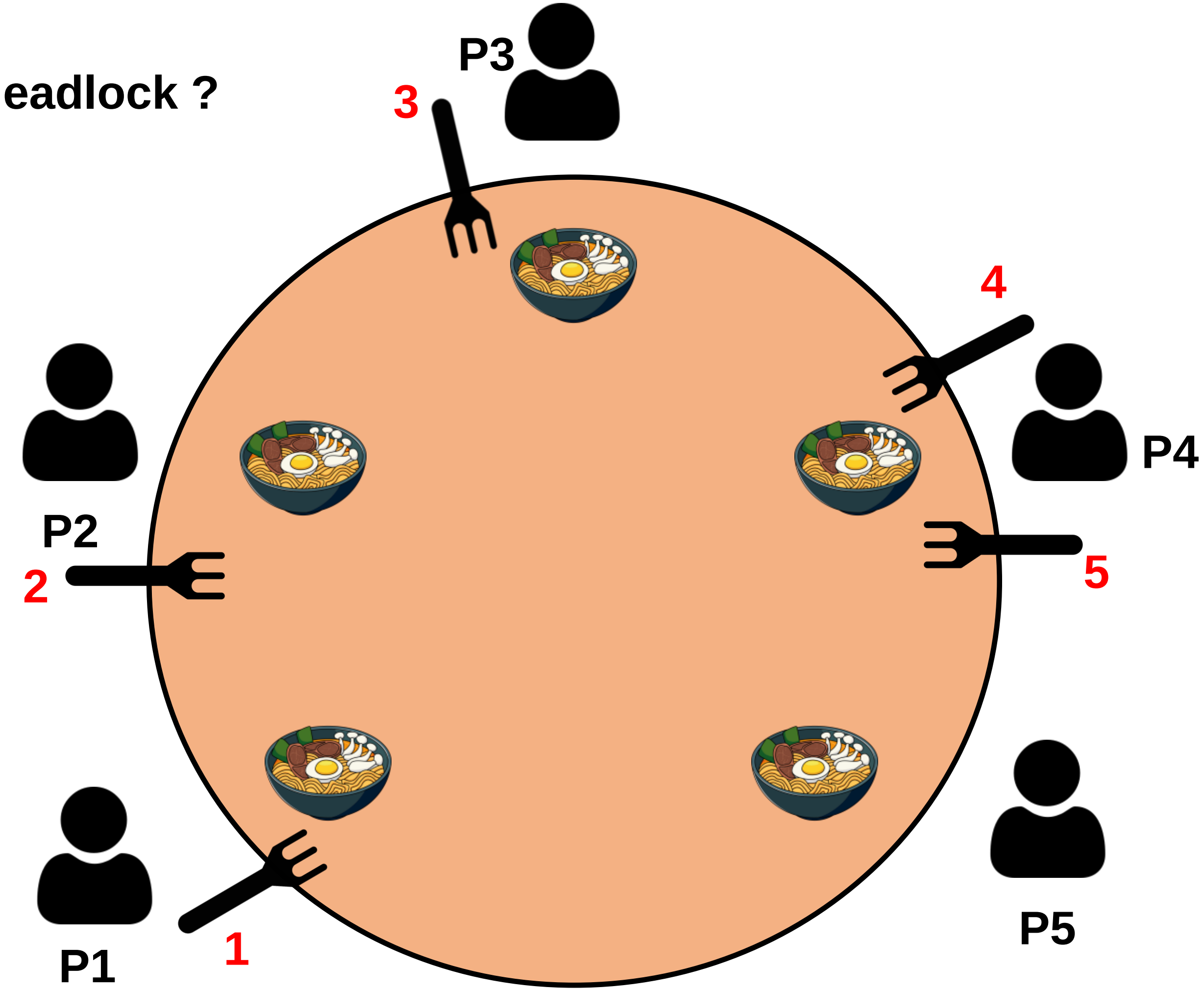
- How this can prevent deadlock ?



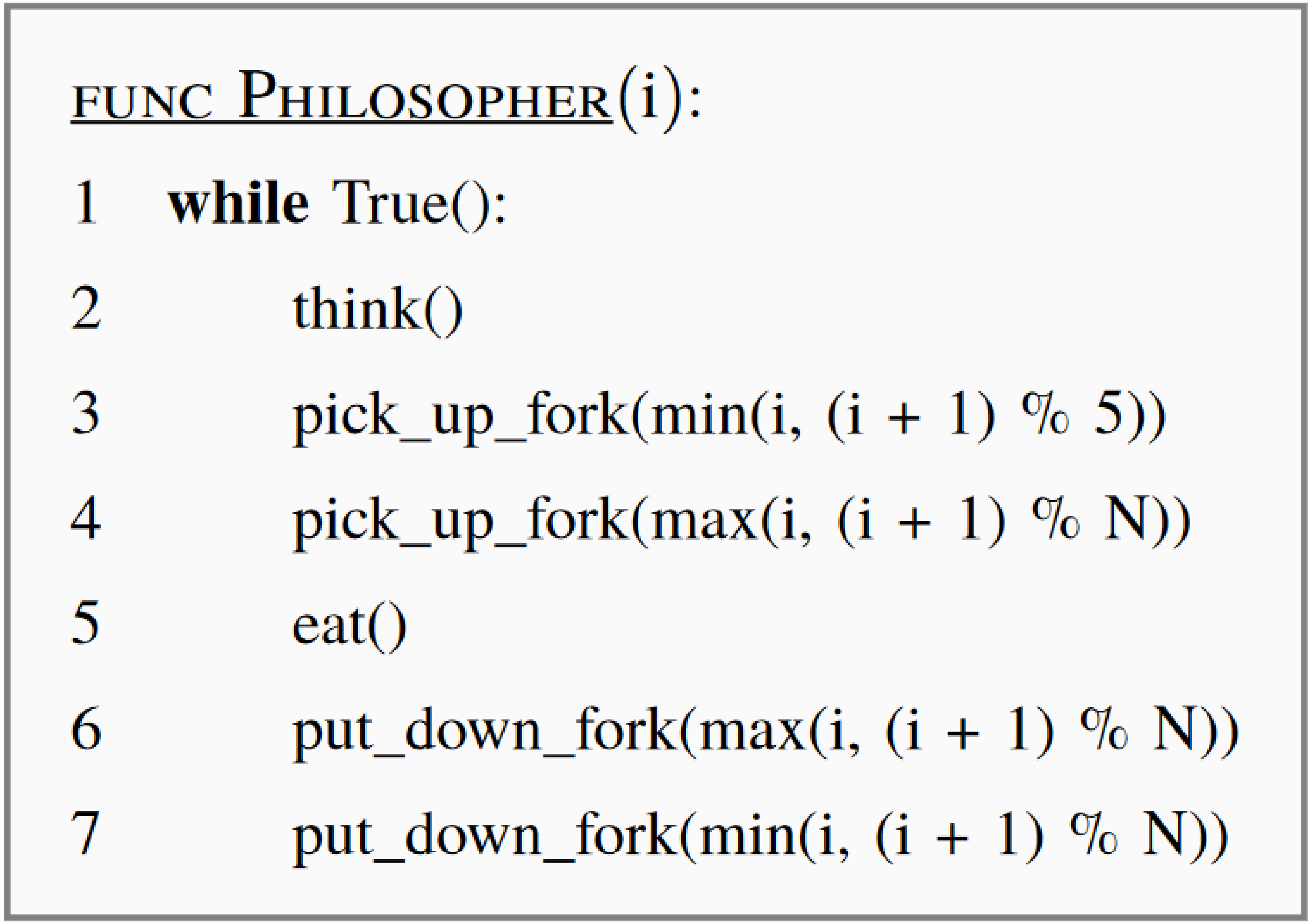
P5 can not take fork 5 -> Not from circular -> prevent deadlock

3. Solution - Resource hierarchy

- How this can prevent deadlock ?

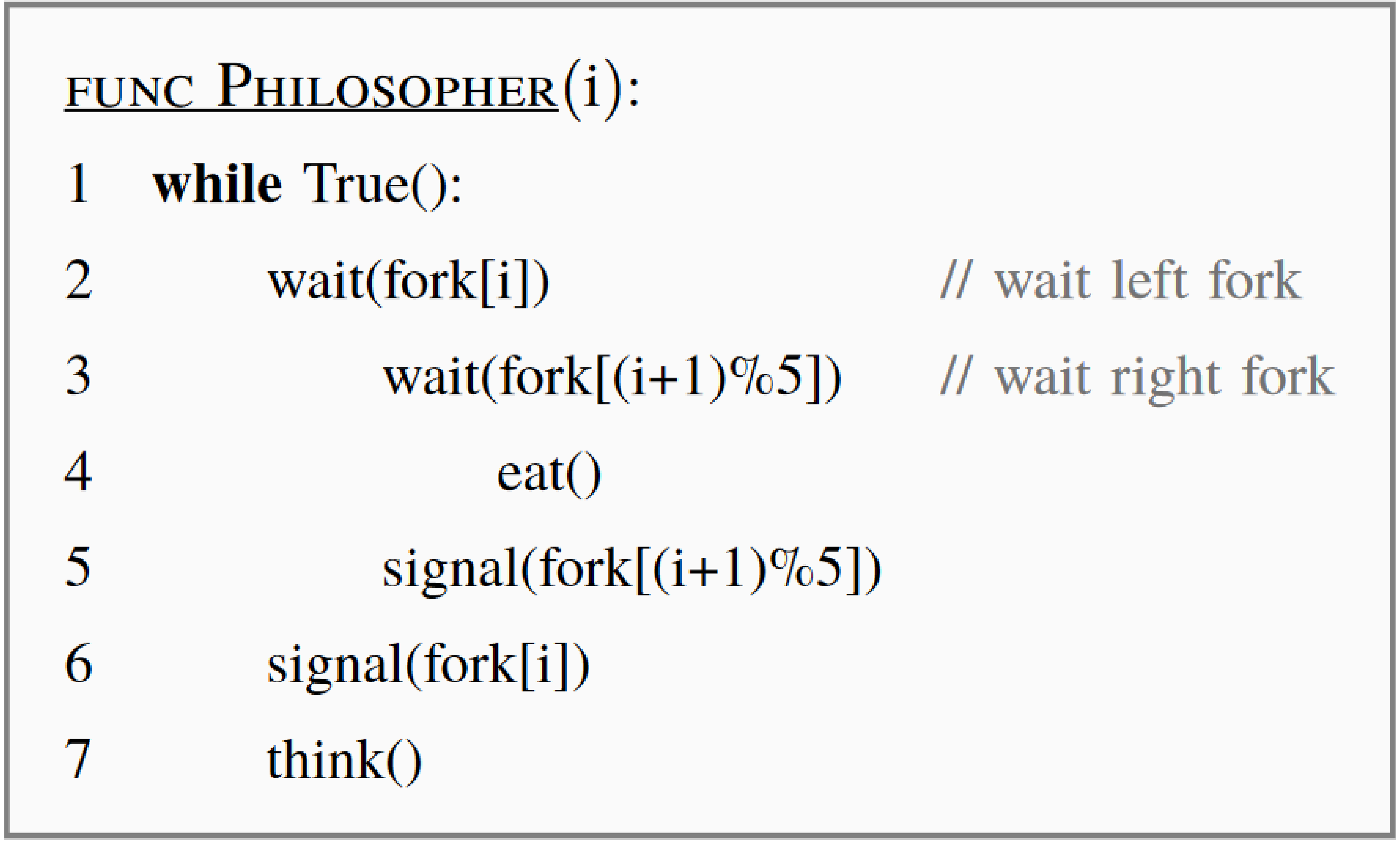


3. Solution - Resource hierarchy



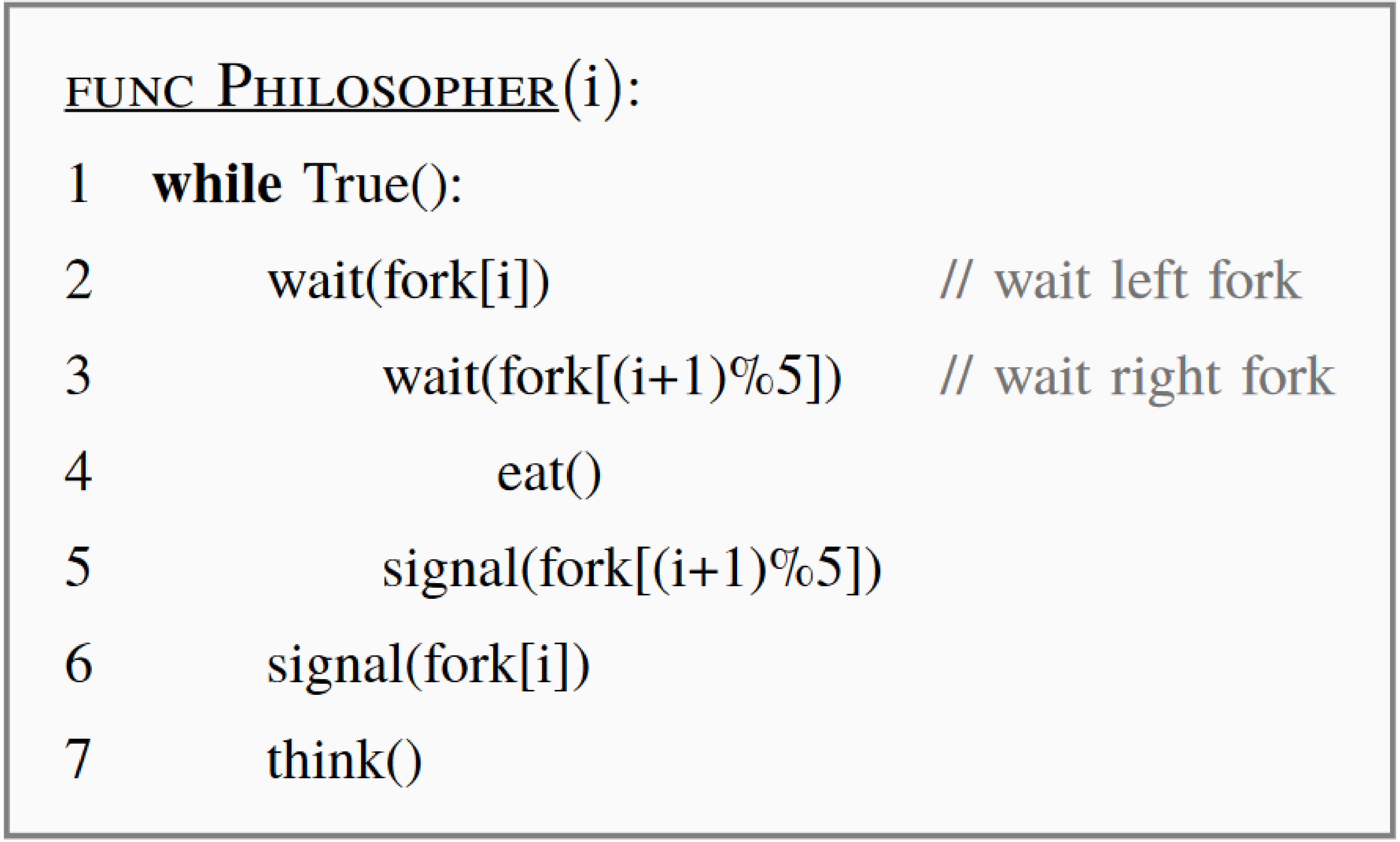
```
FUNC PHILOSOPHER(i):
    while True():
        think()
        pick_up_fork(min(i, (i + 1) % 5))
        pick_up_fork(max(i, (i + 1) % N))
        eat()
        put_down_fork(max(i, (i + 1) % N))
        put_down_fork(min(i, (i + 1) % N))
```

3. Solution - Semaphore



```
FUNC PHILOSOPHER(i):
while True():
        wait(fork[i])                // wait left fork
                wait(fork[(i+1)%5])  // wait right fork
                        eat()
                signal(fork[(i+1)%5])
        signal(fork[i])
        think()
```

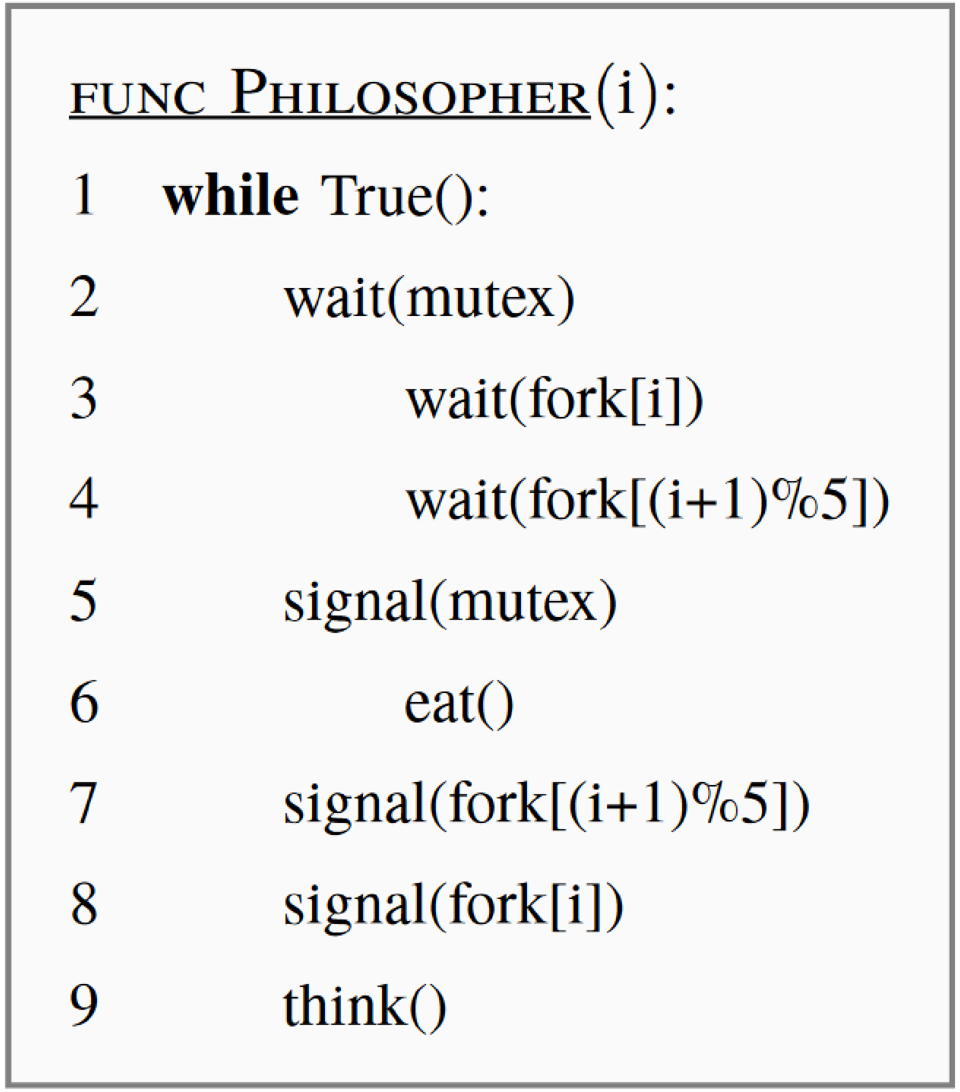
3. Solution - Semaphore



```
FUNC PHILOSOPHER(i):
while True():
    wait(fork[i])               // wait left fork
        wait(fork[(i+1)%5])     // wait right fork
            eat()
        signal(fork[(i+1)%5])
    signal(fork[i])
    think()
```

Still deadlock !!!

3. Solution - Semaphore



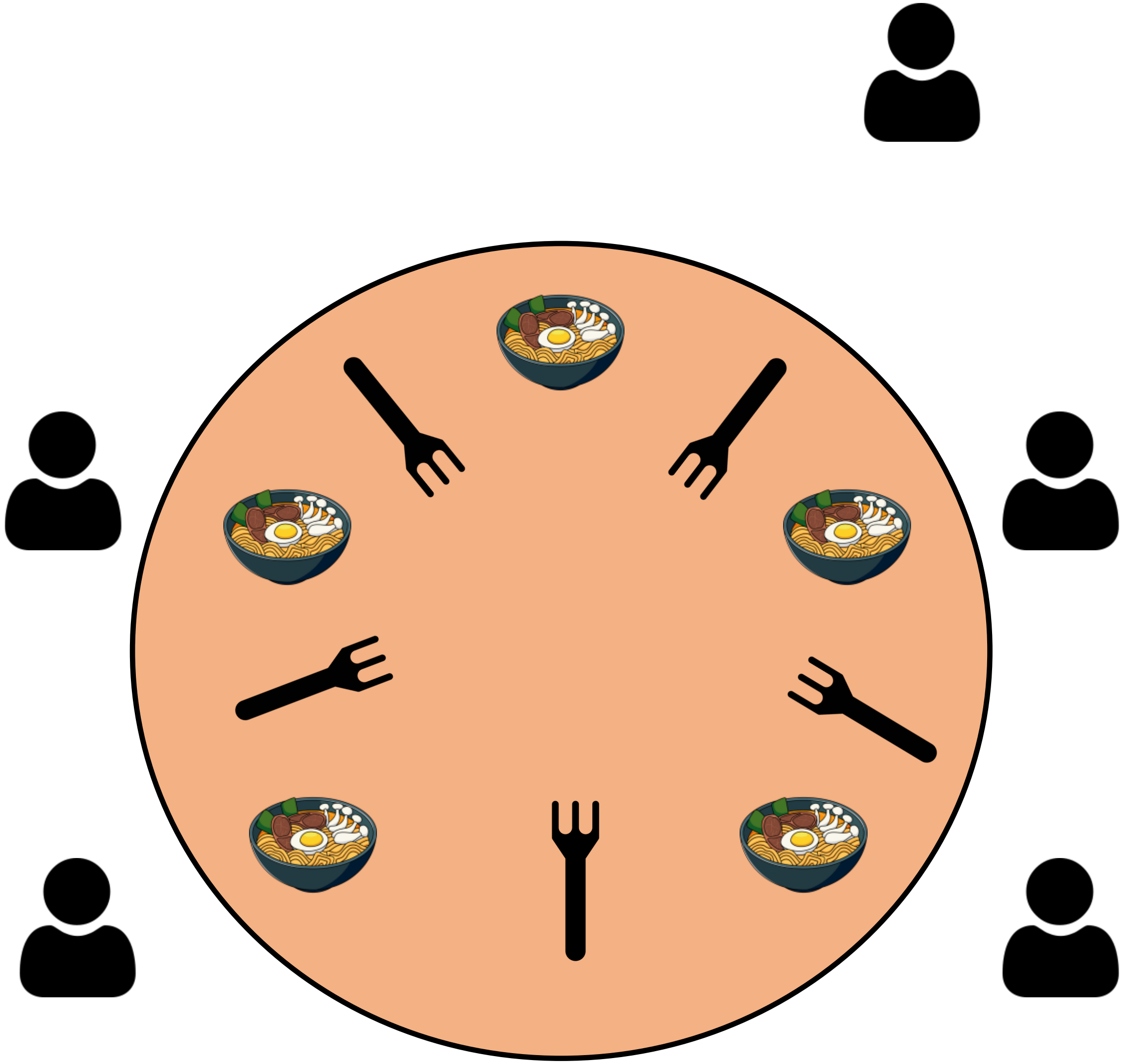
```
FUNC PHILOSOPHER(i):
while True():
    wait(mutex)
        wait(fork[i])
        wait(fork[(i+1)%5])
    signal(mutex)
        eat()
    signal(fork[(i+1)%5])
    signal(fork[i])
    think()
```

Use another semaphore mutex, which ensures that only one philosopher can attempt to pick up forks at a time

3. Solution – Limiting number of dinners

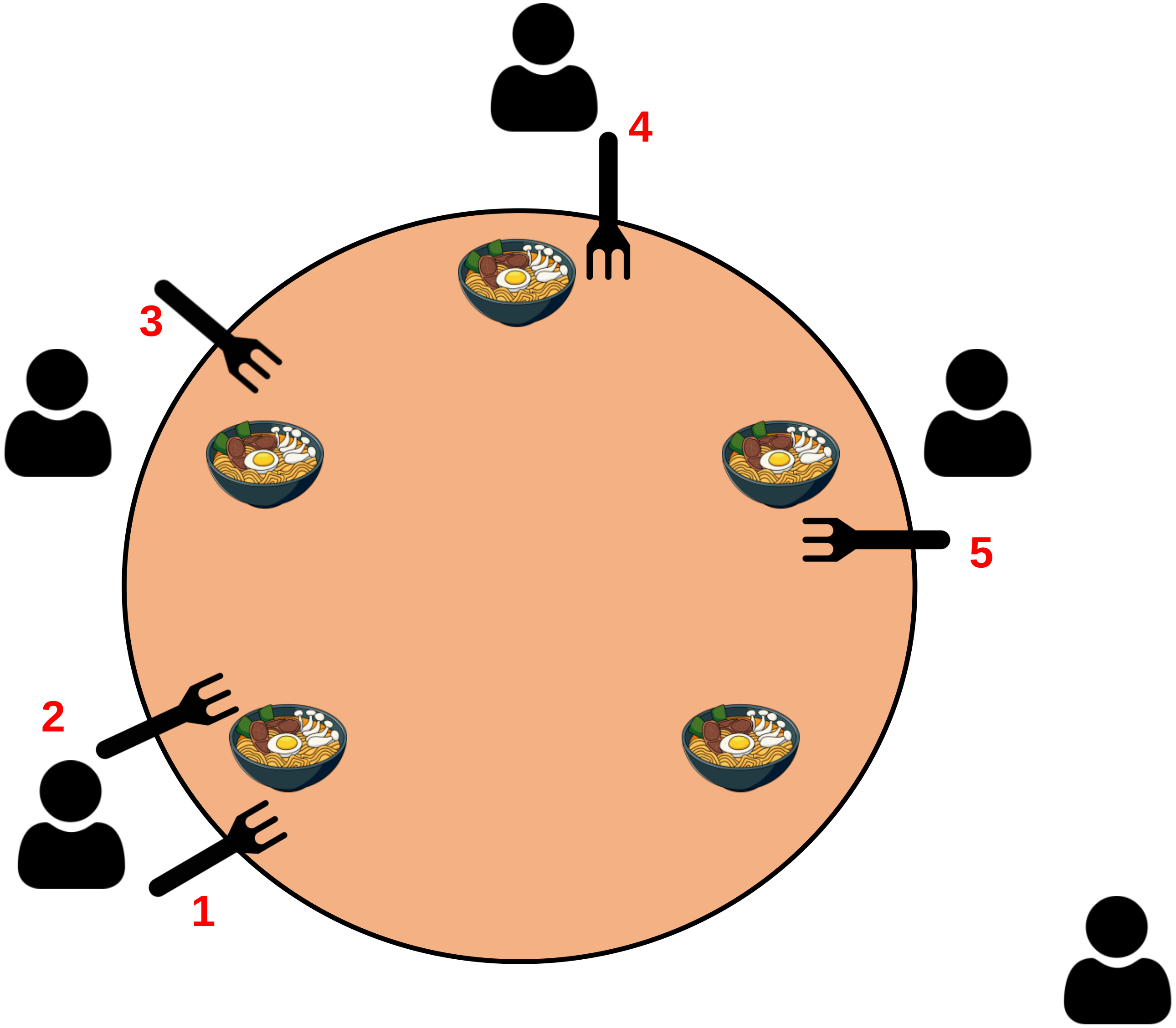
Limit the number of philosophers allowed to sit at the table at any given time to n-1.

Philosophers will take turn sitting and standing.



3. Solution – Limiting number of dinners

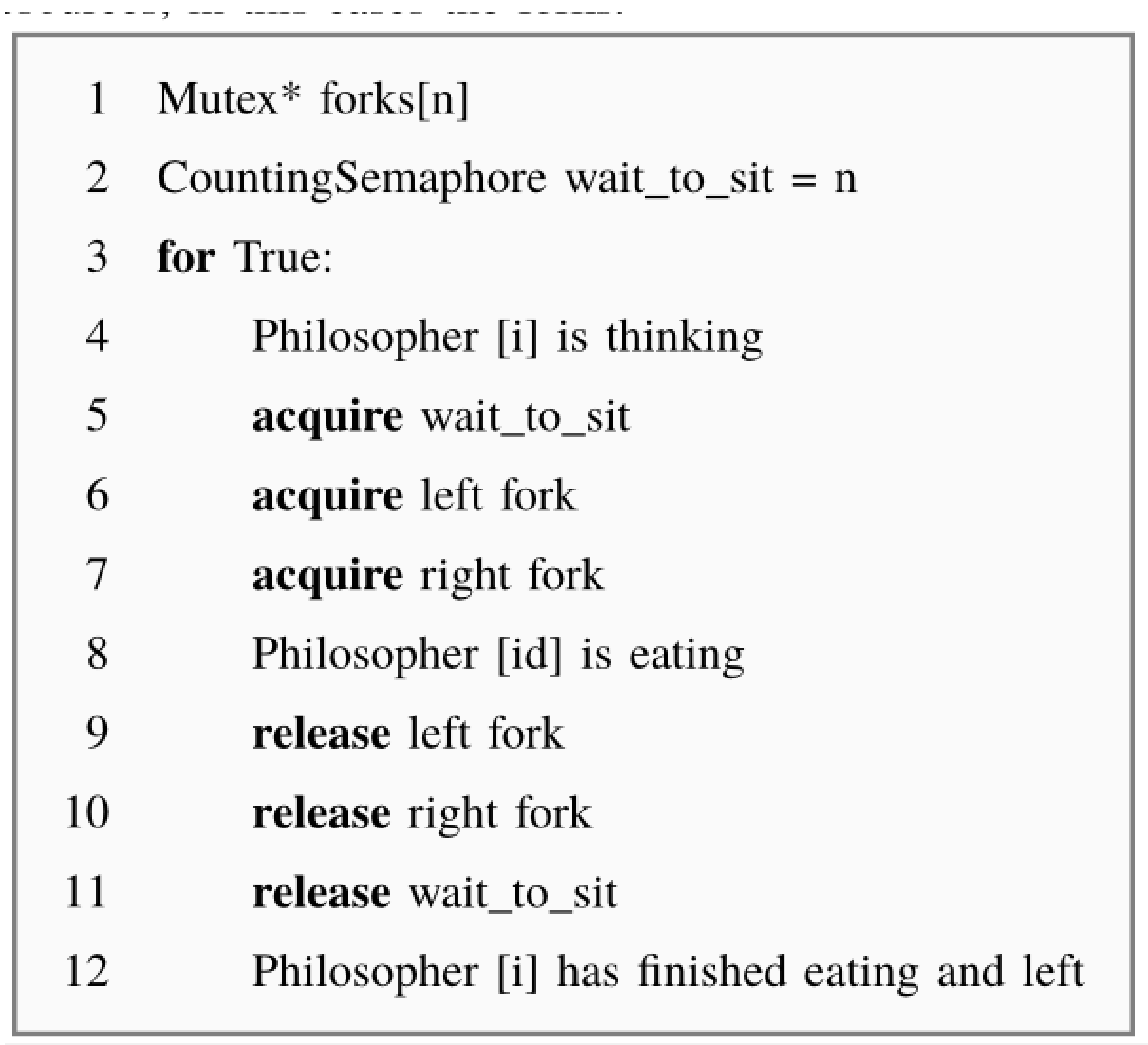
This make sures at least one philosophers will have two forks, effectively preventing deadlock.



3. Solution – Limiting number of dinners

Use a Counting Semaphore to limit the number of concurrent dinners.

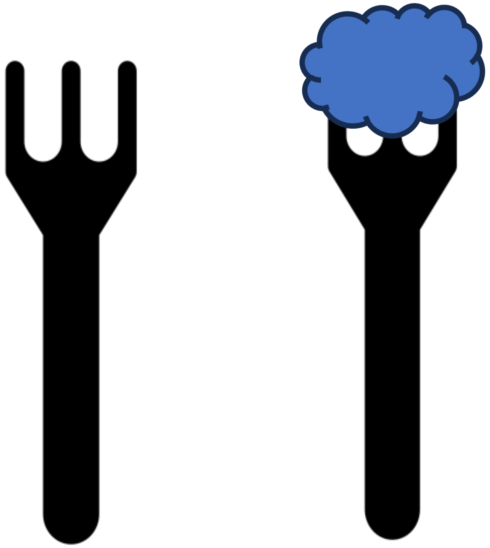
Mutex for each fork.

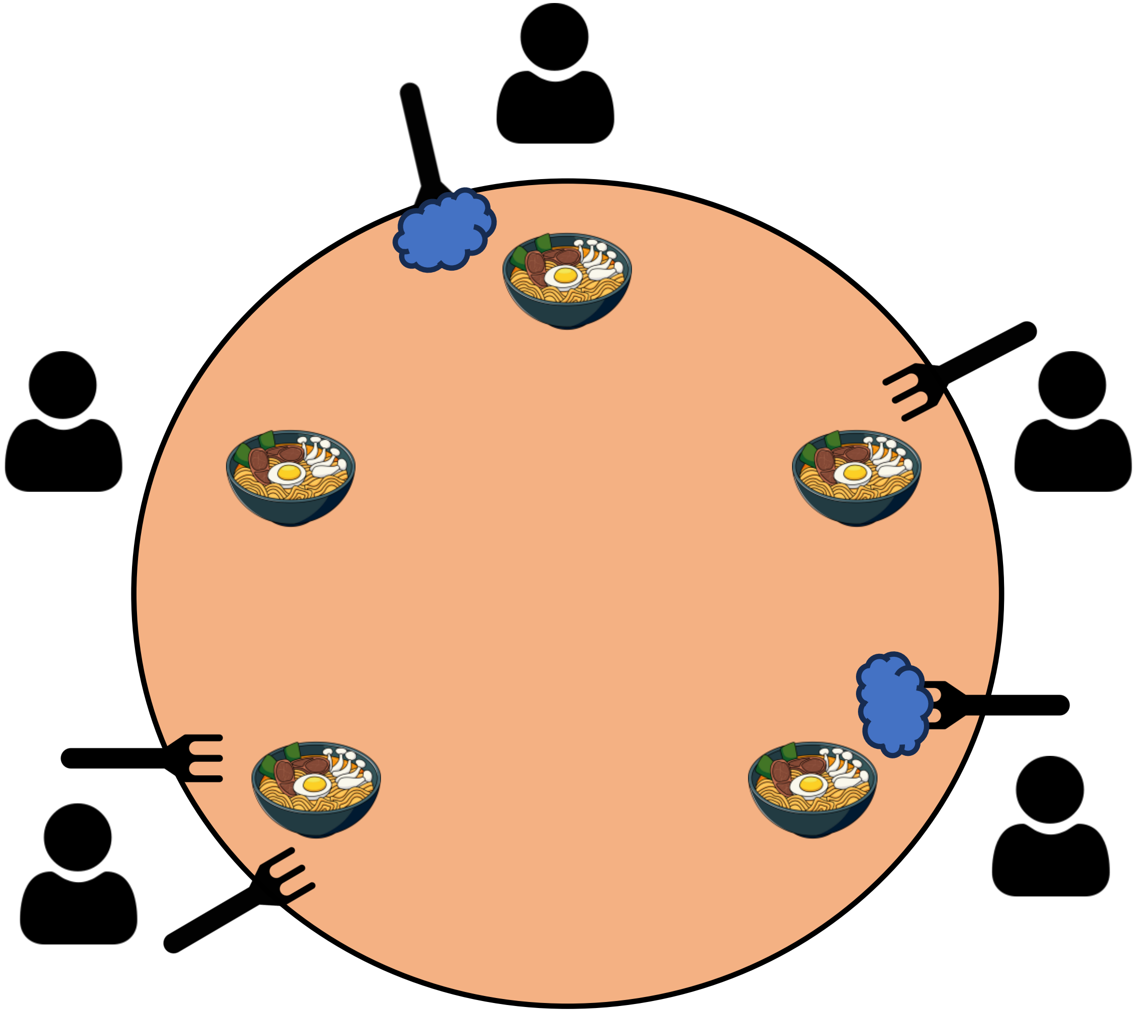


```
Mutex* forks[n]
CountingSemaphore wait_to_sit = n
for True:
        Philosopher [i] is thinking
        acquire wait_to_sit
        acquire left fork
        acquire right fork
        Philosopher [id] is eating
        release left fork
        release right fork
        release wait_to_sit
        Philosopher [i] has finished eating and left
```

4. Solution – Chandy/Misra Approach

Chandy-Misra's algorithm can be explained using the concepts of "clean" and "dirty" forks.



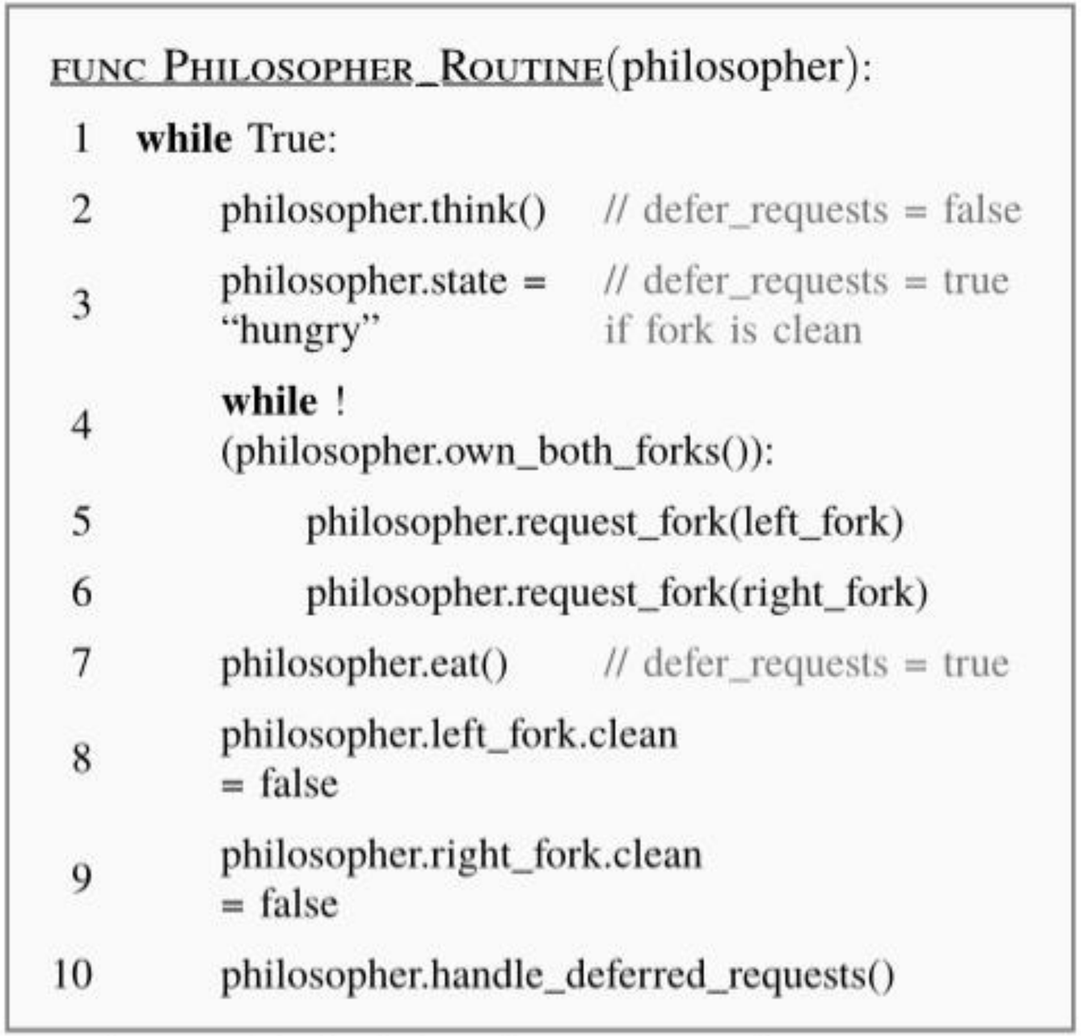


4. Solution – Chandy/Misra Approach

Initially, the fork is assigned to the philosopher with the lower ID (lower neighbor) and marked as "dirty."

Then, each philosopher acts in a routine with 4 states:

- Thinking
- Waiting (hungry)
- Eating
- Cleanup



```
FUNC PHILOSOPHER_ROUTINE(philosopher):
while True:
    philosopher.think()                    // defer_requests = false
    philosopher.state = “hungry”           // defer_requests = true if fork is clean
    while !(philosopher.own_both_forks()):
        philosopher.request_fork(left_fork)
        philosopher.request_fork(right_fork)
    philosopher.eat()                      // defer_requests = true
    philosopher.left_fork.clean = false
    philosopher.right_fork.clean = false
    philosopher.handle_deferred_requests()
```

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !