# CSCI 3753: Operating Systems Fall 2024

Dylan Sain

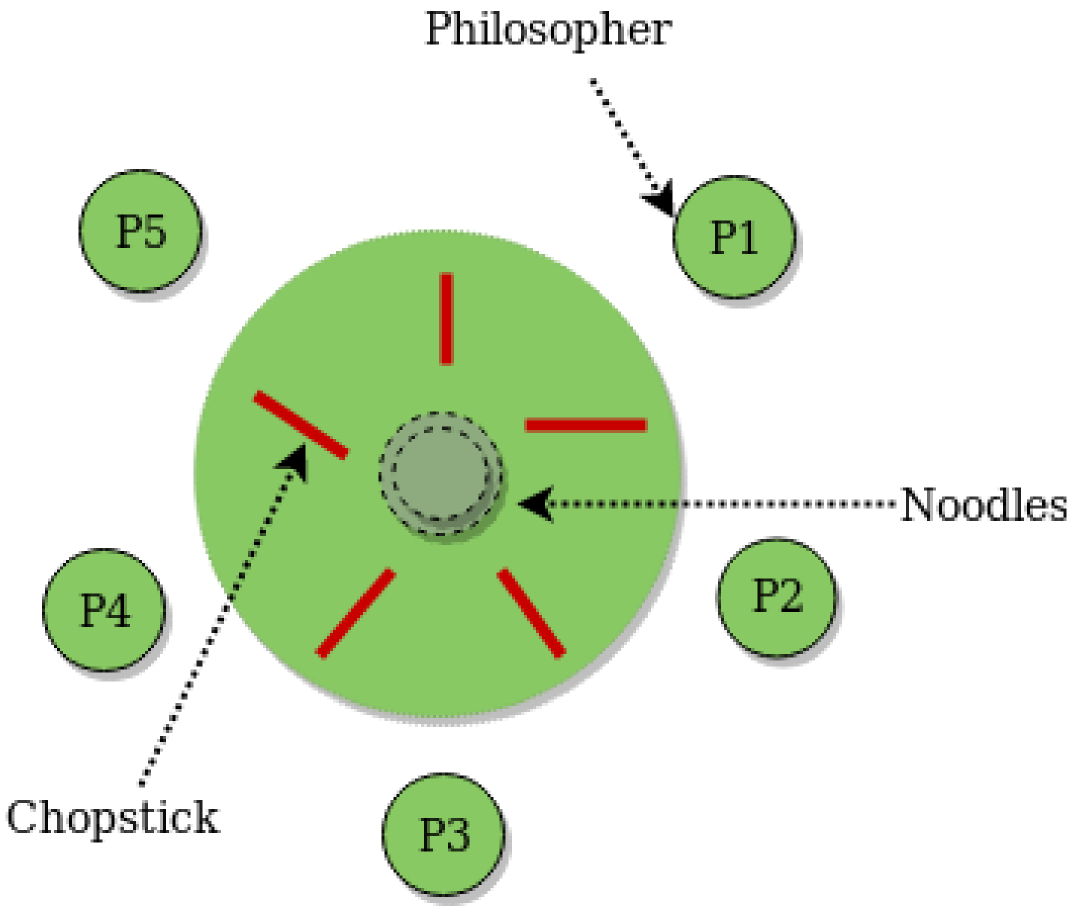
Department of Computer Science

University of Colorado Boulder

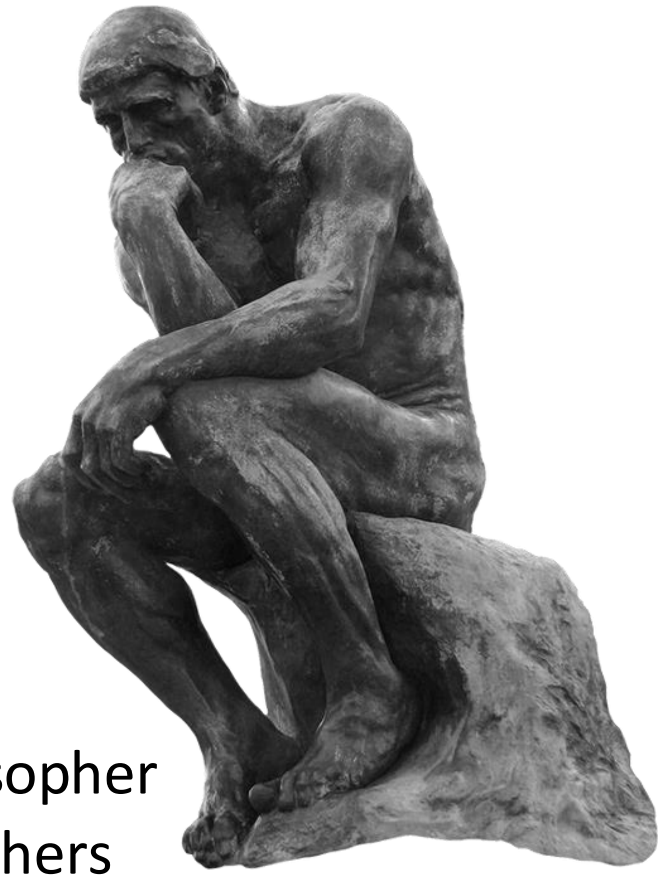
## PA4 and Midterm questions?

## Week 7: Hungry Hungry Philosophers

## The Problem:



## The Rules:



- Philosopher can either:
  - Eat
  - Think
  - Pick up chopstick
  - Set down chopstick
- A philosopher cannot:
  - Take a chopstick from another philosopher
  - Know the state of the other philosophers
  - Do two actions at once (ie pick up both chopsticks)

## Solution 1: Basic Solution

Think
Pick up right chopstick
Pick up left chopstick
Eat food
Put down left chopstick
Put down right chopstick
Repeat

Problem?

## Solution 2: Semaphores

```
Semaphore  array [0..4] fork <- [1,1,1,1,1]
Loop forever:
    think
    wait(fork[i])
    wait(fork[i + 1])
    eat
    signal(fork[i])
    signal(fork[i + 1])
```

## Last week's example!

```
a_function(){
    lock(mutex a)
        //1st critical code
        lock(mutex b)
            //2nd critical code

        unlock(mutex b)
    unlock(mutex a)
}
```

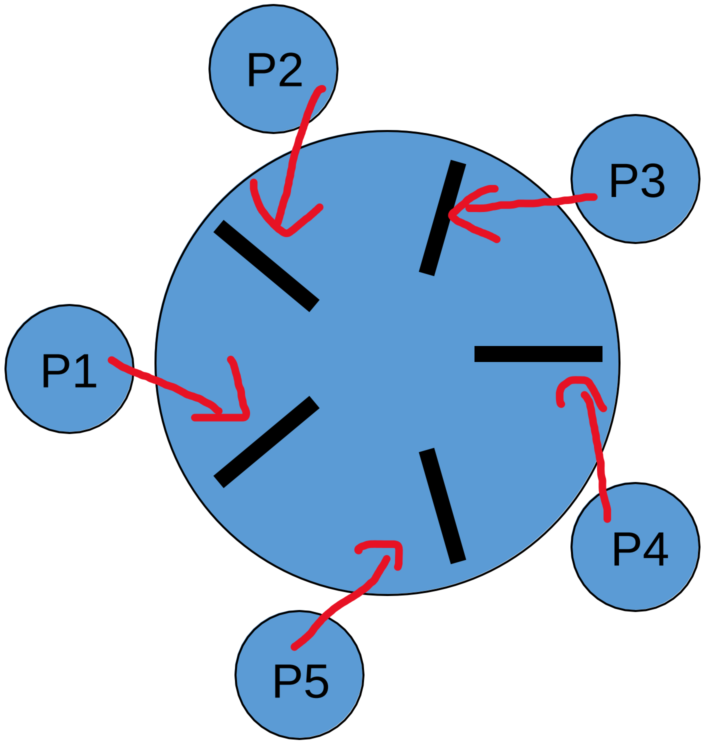
```
b_function(){
    lock(mutex b)
        //1st critical code
        lock(mutex a)
            //2nd critical code

        unlock(mutex a)
    unlock(mutex b)
}
```

## First problem

What if all philosophers arrive and start picking up their chopsticks at the same time?

No body gets to EAT!

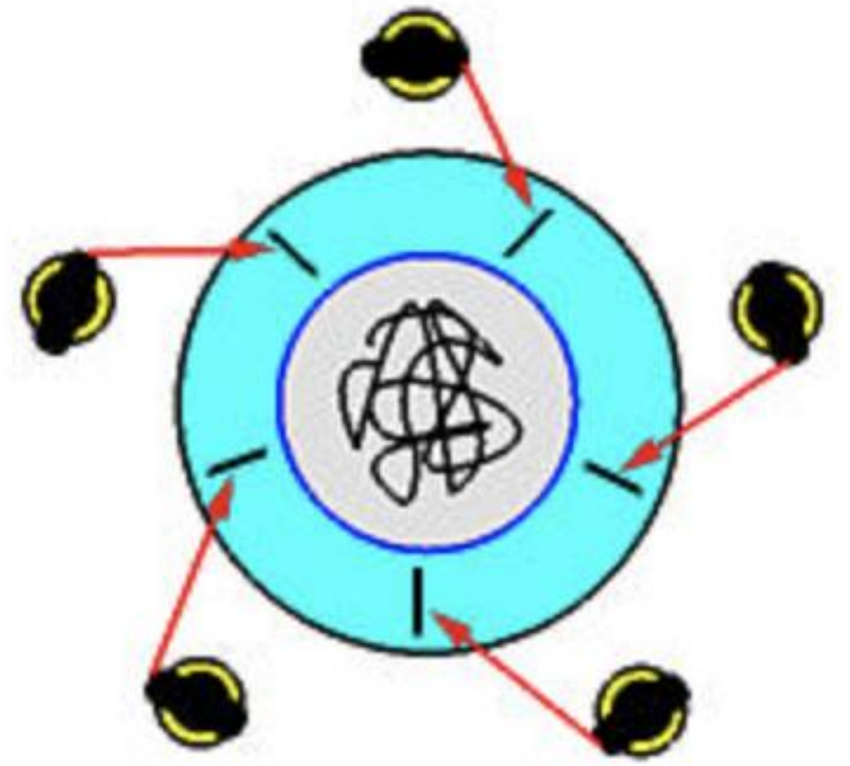


# 4 Conditions for Deadlock

1. **Mutual Exclusion**: at least one chopstick must be held by a philosopher.
2. **Hold and Wait**: at least one philosopher must hold a chopstick and be waiting for another chopstick
3. **No Preemption**: a philosopher cannot take the chopstick from another philosopher
4. **Circular Wait**: all philosophers must be waiting in a circular pattern

## Deadlock

When all resources are allocated but no eating is being done



## Solution 3: Asymmetrical Solution

Semaphore array [0..4] fork <- [1,1,1,1,1]

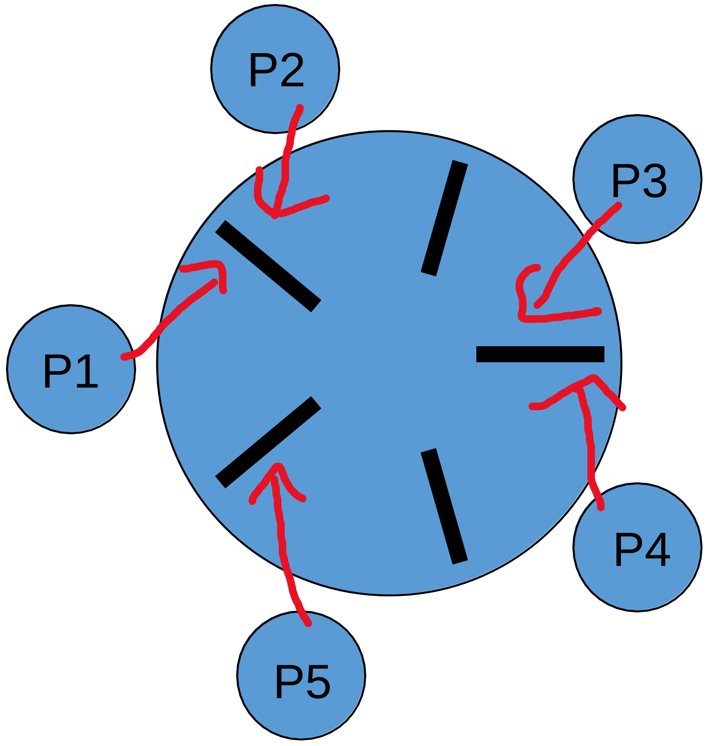
```
Loop forever (even):
    think
    wait(fork[i])
    wait(fork[i + 1])
    eat
    signal(fork[i])
    signal(fork[i + 1])
```

```
Loop forever (odd):
    think
    wait(fork[i + 1])
    wait(fork[i])
    eat
    signal(fork[i + 1])
    signal(fork[i])
```

## Second problem

- P1 and P2 fight for a chopstick
- P3 and P4 fight for a chopstick
- P5 always gets their chopstick

- This approach favors certain processes over others and can lead to in optimal allocation of resources

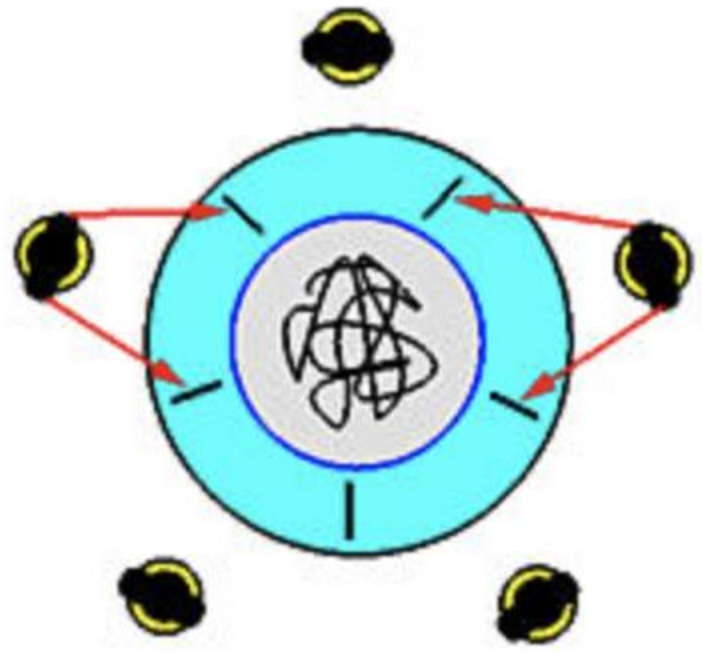


## Starvation

When two philosophers are very fast thinkers and eaters and are opposite of each other

When one philosopher is prioritized over the others

When one philosopher is made to wait much more than the others



## Solution 5: State tracking

Solution:

- When a philosopher wants to eat they check both chopsticks
- If free, eat, if not wait on condition variable
- When a philosopher finishes eating checks if neighbors are hungry and releases condition variable
- Problems?

# Third problem

```
UNIX> dphil_5 1
  0 Total blocktime:     0 :     0     0     0     0
  0 Philosopher 0 thinking for 1 seconds
  0 Philosopher 1 thinking for 1 seconds
  0 Philosopher 2 thinking for 1 seconds
  0 Philosopher 3 thinking for 1 seconds
  0 Philosopher 4 thinking for 1 seconds
  1 Philosopher 1 no longer thinking -- calling pickup()
  1 Philosopher 1 eating for 1 seconds
  1 Philosopher 2 no longer thinking -- calling pickup()
  1 Philosopher 3 no longer thinking -- calling pickup()
  1 Philosopher 3 eating for 1 seconds
  1 Philosopher 0 no longer thinking -- calling pickup()
  1 Philosopher 4 no longer thinking -- calling pickup()
  2 Philosopher 1 no longer eating -- calling putdown()
  2 Philosopher 1 thinking for 1 seconds
  2 Philosopher 0 eating for 1 seconds
  2 Philosopher 3 no longer eating -- calling putdown()
  2 Philosopher 3 thinking for 1 seconds
  2 Philosopher 2 eating for 1 seconds
  3 Philosopher 2 no longer eating -- calling putdown()
  3 Philosopher 2 thinking for 1 seconds
  3 Philosopher 1 no longer thinking -- calling pickup()
  3 Philosopher 0 no longer eating -- calling putdown()
  3 Philosopher 0 thinking for 1 seconds
  3 Philosopher 3 no longer thinking -- calling pickup()
  3 Philosopher 3 eating for 1 seconds
  3 Philosopher 1 eating for 1 seconds
  4 Philosopher 0 no longer thinking -- calling pickup()
  4 Philosopher 3 no longer eating -- calling putdown()
  4 Philosopher 3 thinking for 1 seconds
  4 Philosopher 1 no longer eating -- calling putdown()
  4 Philosopher 1 thinking for 1 seconds
  4 Philosopher 2 no longer thinking -- calling pickup()
```

```
3 Philosopher 2 thinking for 1 seconds
3 Philosopher 1 no longer thinking -- calling pickup()
3 Philosopher 0 no longer eating -- calling putdown()
3 Philosopher 0 thinking for 1 seconds
3 Philosopher 3 no longer thinking -- calling pickup()
3 Philosopher 3 eating for 1 seconds
3 Philosopher 1 eating for 1 seconds
4 Philosopher 0 no longer thinking -- calling pickup()
4 Philosopher 3 no longer eating -- calling putdown()
4 Philosopher 3 thinking for 1 seconds
4 Philosopher 1 no longer eating -- calling putdown()
4 Philosopher 1 thinking for 1 seconds
4 Philosopher 2 no longer thinking -- calling pickup()
4 Philosopher 2 eating for 1 seconds
4 Philosopher 0 eating for 1 seconds
5 Philosopher 1 no longer thinking -- calling pickup()
5 Philosopher 2 no longer eating -- calling putdown()
5 Philosopher 2 thinking for 1 seconds
5 Philosopher 0 no longer eating -- calling putdown()
5 Philosopher 0 thinking for 1 seconds
5 Philosopher 3 no longer thinking -- calling pickup()
5 Philosopher 3 eating for 1 seconds
5 Philosopher 1 eating for 1 seconds
6 Philosopher 1 no longer eating -- calling putdown()
6 Philosopher 1 thinking for 1 seconds
6 Philosopher 0 no longer thinking -- calling pickup()
6 Philosopher 0 eating for 1 seconds
6 Philosopher 3 no longer eating -- calling putdown()
6 Philosopher 3 thinking for 1 seconds
6 Philosopher 2 no longer thinking -- calling pickup()
6 Philosopher 2 eating for 1 seconds
7 Philosopher 0 no longer eating -- calling putdown()
7 Philosopher 0 thinking for 1 seconds
7 Philosopher 3 no longer thinking -- calling pickup()
7 Philosopher 2 no longer eating -- calling putdown()
7 Philosopher 2 thinking for 1 seconds
7 Philosopher 1 no longer thinking -- calling pickup()
7 Philosopher 1 eating for 1 seconds
7 Philosopher 3 eating for 1 seconds
8 Philosopher 2 no longer thinking -- calling pickup()
8 Philosopher 1 no longer eating -- calling putdown()
```

## Starvation continued

Philosopher 4 never gets to eat?
Why?

1 and 3 eat then 0 and 2 eat so there is never a time when 0 and 3 are not eating

Starvation can either be prevented by

- A guarantee from the thread system that threads will be unblocked in the same order they are blocked
    - Does not solve one philosopher eating more than others!
- Do it yourself

## Solution 6: Preventing Starvation...queue?

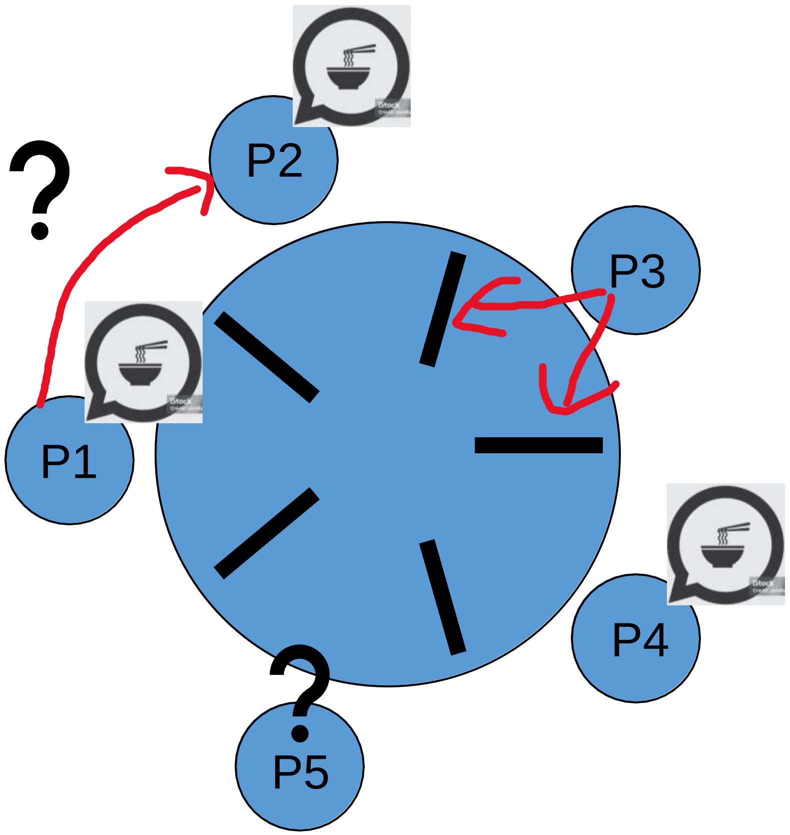
Solution:

- Queue-like solution
- Each philosopher takes a number when hungry
- If a philosopher is hungry and so is his neighbors, then he only eats if he has the lowest number.

# Final solution: bringing it all together

Solution:

- Combining solution 5 and solution 6
- Check if neighbors are hungry and also check if they have been waiting
- Only pickup if neighbors have not been waiting for a long time.



## Credits

Really good examples with code and outputs! Check it out!

https://web.eecs.utk.edu/~mbeck/classes/cs560/560/notes/Dphil/lecture.html