CS 450: Operating Systems Lecture 10: Dining Philosophers

Spring 2014, J. Sasaki
Dept of Computer Science
Illinois Institute of Technology

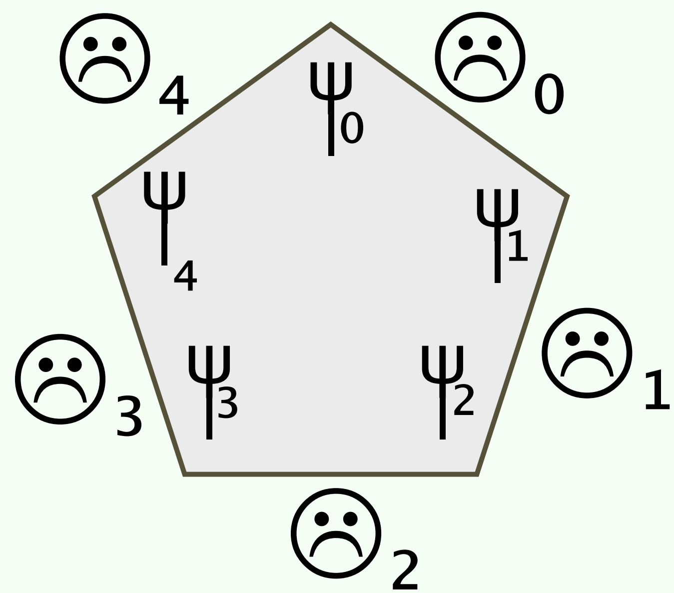
Dining Philosophers

Another Classical Problem

- *Producer-Consumer Problem*: Sharing a resource that can be used in different ways.
- ***Dining Philosopher Problem*** involves sharing multiple copies of the same resource.
  - Each user needs 2 of the 5 items.

Dining Philosphers

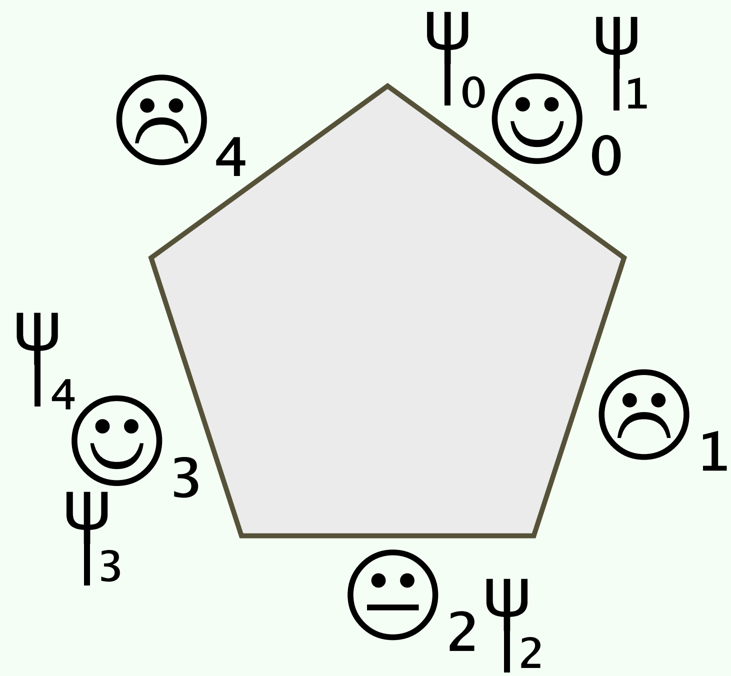
- Dining table, 5 philosophers, 5 forks, bowl of spaghetti in middle of table.
- To eat, each philosopher needs to grab the two forks on either side.
- A fork can be held only by 1 philosopher



4

Example: Dining Philosphers

- P_0 and P_3 each have 2 forks and can eat.
- P_1 and P_4 have no forks and can't eat.
- P_2 has a right fork but no left fork; it can't eat.



Dining Philosophers

- Model: 1 threads/philosopher, 1 mutex semaphore per fork.
- Fork `left(i)` is philosopher i's left fork
- Fork `right(i)` is philosopher i's right fork

```
Semaphore forks[5];
define right(i) = i;
define left(i) = (i+1) % 5
```


6

Dining Philosophers

- Philosophers alternate between eating and not eating

```
philosopher P_i : do {
  …
  get_forks(i)
  … eat …
  release_forks(i);
  …
} while (…);
```

1: Naive Solution

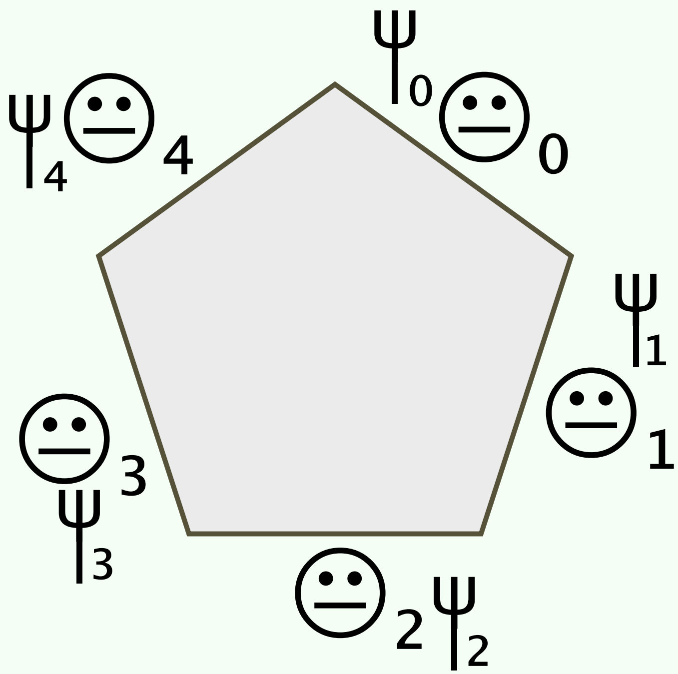
- Solution 1:

```
get_forks(i):
  forks[right(i)].wait();
  forks[left(i)].wait();
release_forks():
  forks[right(i)].signal();
  forks[left(i)].signal();
```

- But what happens if all P's grab their right fork before any grabs their left one?

Deadlock

- Everyone holds a right fork & waits for left fork



1a: Drop Right Fork if Left Fork Unavailable?

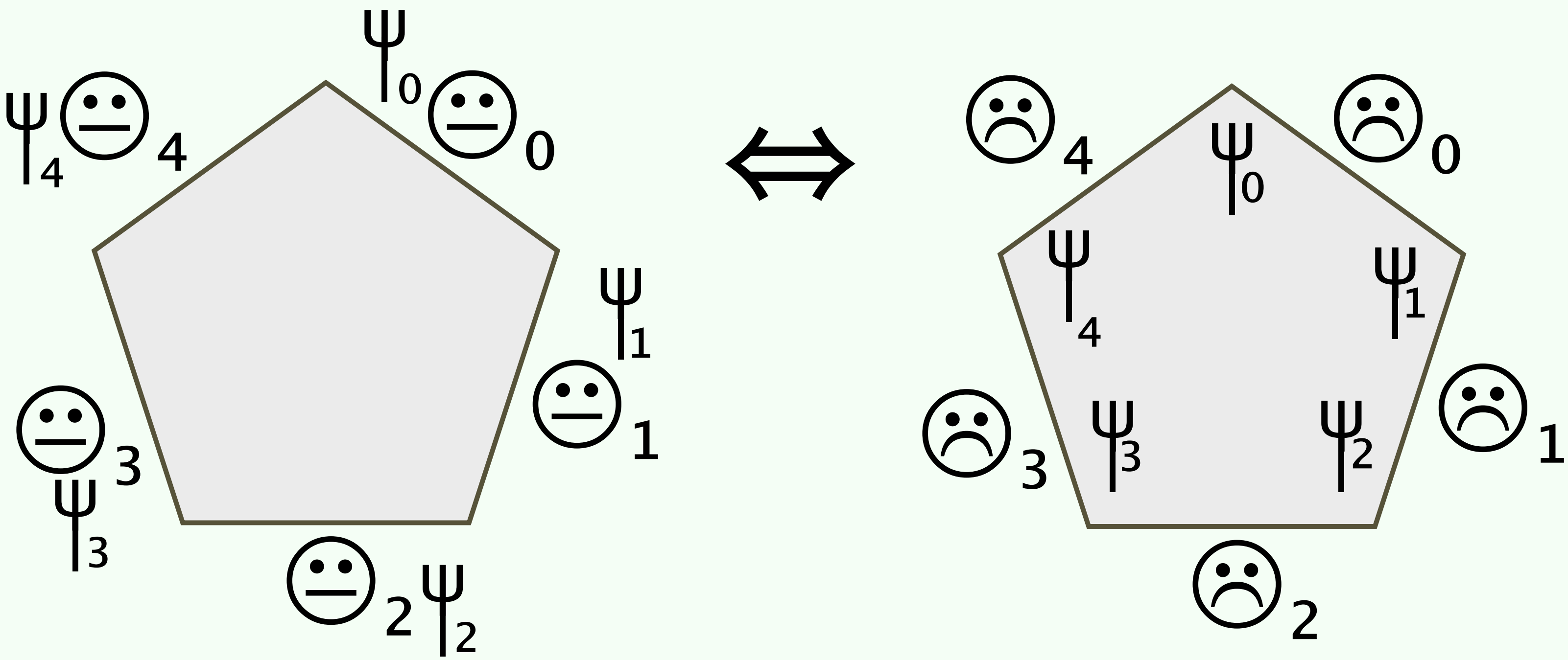
- Can create a version of `wait()` that doesn't wait but returns boolean true/false saying whether or not we succeeded in picking up a fork.

```
while (!success) {
   forks[right(i)].wait();
   if (!forks[left(i)].try())
      forks[right(i)].signal();
   else success = true;
}
```

  - Possible to get “live lock”

Livelock

- Alternate two states; unlikely due to timings



2: Global Lock?

- Define a mutex for eating?

```
Semaphore can_eat_mutex = 1;
get_forks(i):
  can_eat_mutex.wait();
   forks[right(i)].wait();
   forks[left(i)].wait();
  can_eat_mutex.signal();
```

- Any starvation possible?
- How much concurrency?

3: Multiplex Two Eaters

- Let 2 diners eat simultaneously?

```
Semaphore can_eat = 2;

get_forks(i):
  can_eat.wait();
   forks[right(i)].wait();
   forks[left(i)].wait();
  can_eat.signal();
```

- Now, how about starvation and concurrency?

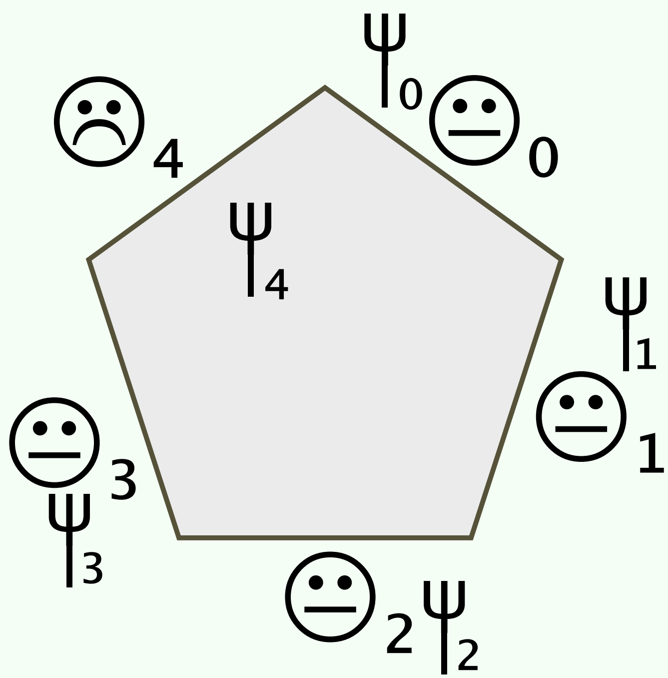
4: Slightly Asymmetric Diners

- Let P_0, …, P_3 try to grab their forks right then left, but P_4 tries to grab forks left then right. Can deadlock still occur?
- Say P_0, …, P_3 each grabs their right fork; then P_4 tries to grab its left fork
- Who eats? Who waits?

14

Slightly Asymmetric

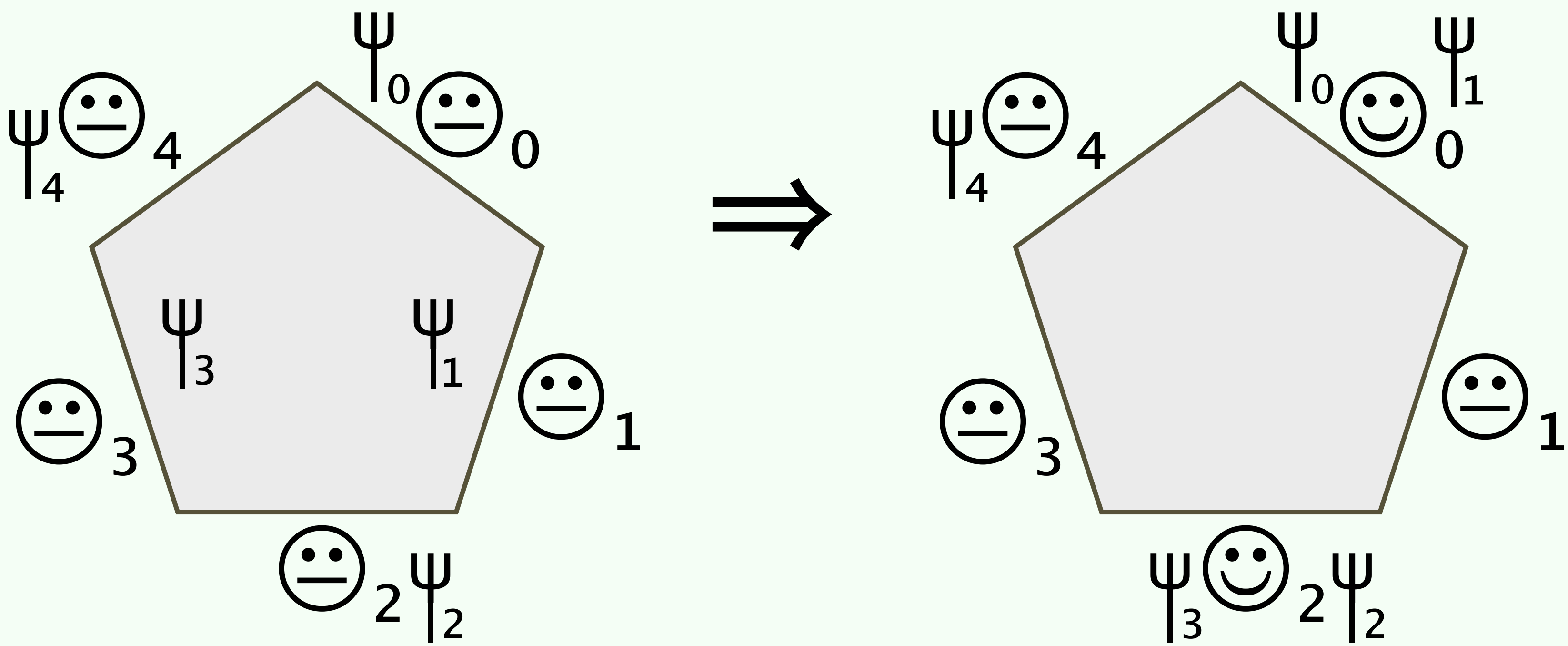
- What if P_3 is much faster than the others?



5: Alternate Lefty-Righty

- Even-numbered philosophers get right fork then left fork
- Odd-numbered philosophers get left fork then right fork.
  - Say P_0, P_2, P_4 get left forks 0, 2, 4
  - P_1, P_3 block trying for 2, 4
  - So 1 & 3 are available for P_0, P_2.

Alternate Lefty-Righty



6: Limit Attempts to Eat

- No deadlock if only four P's attempt to eat.
- Introduce 4 napkins; to eat, you must first get a napkin and then get your forks.

```
Semaphore napkins = 4;
…
napkins.wait();
  forks[right(i)].wait();
  forks[left(i)].wait();
napkins.signal();
```

- Starvation? Concurrency?

Need a Napkin

- P_0 and P_2 have napkins and got forks.
- P_1 and P_4 have napkins but are still missing forks.
- P_3 has no napkin, so it can't even try to get a fork
- No deadlock, but what about starvation and concurrency?