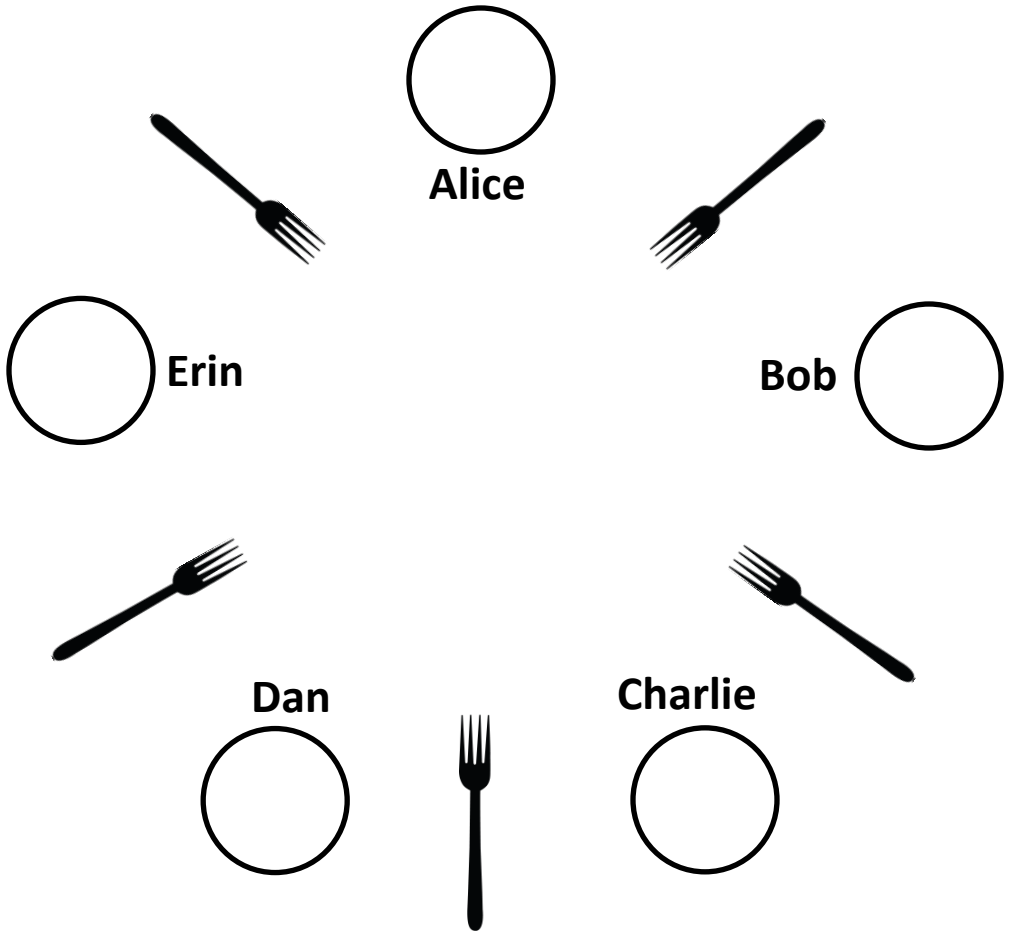
Classical Synchronization I

CS 241

Oct. 18, 2013

Dinning Philosophers Problem



Mutual Exclusion? ☐

Hold and Wait? ☐

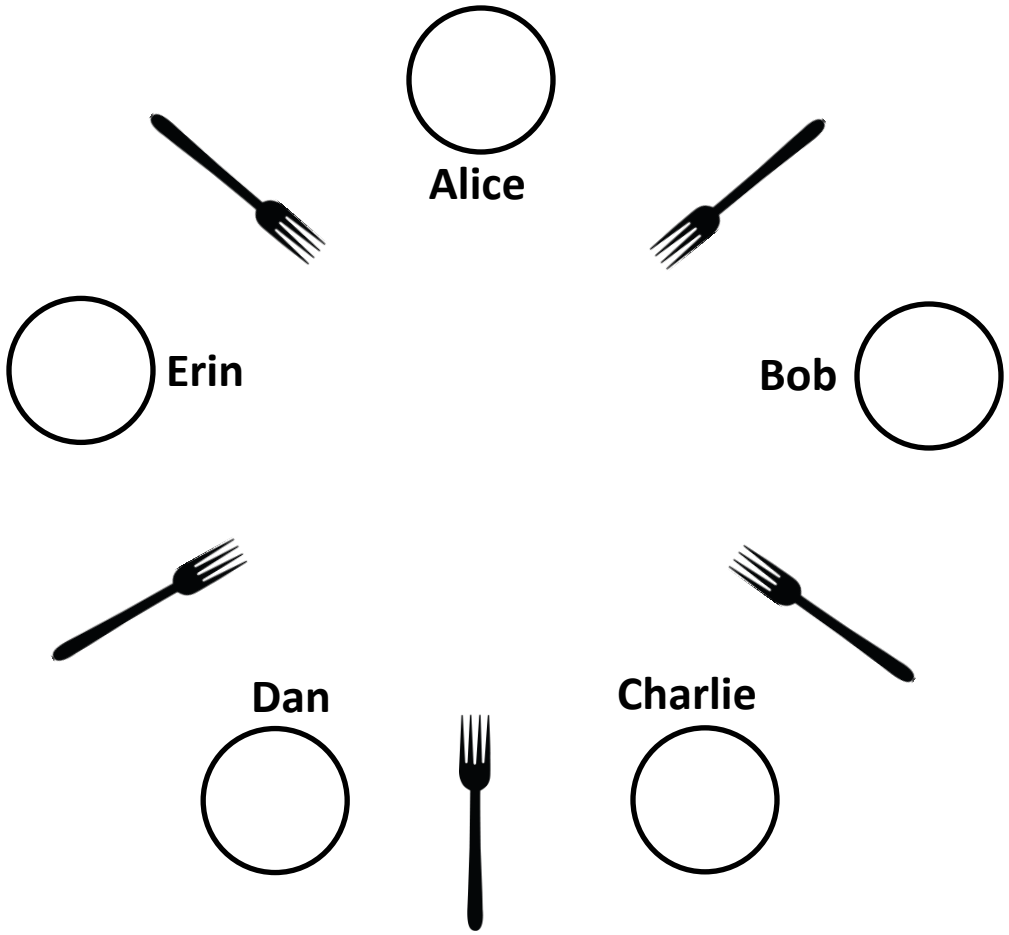
No Preemption? ☐

Circular Wait? ☐

Deadlock? ☐

Progress? ☐

Dinning Philosophers Problem



Mutual Exclusion? ☐

Hold and Wait? ☐

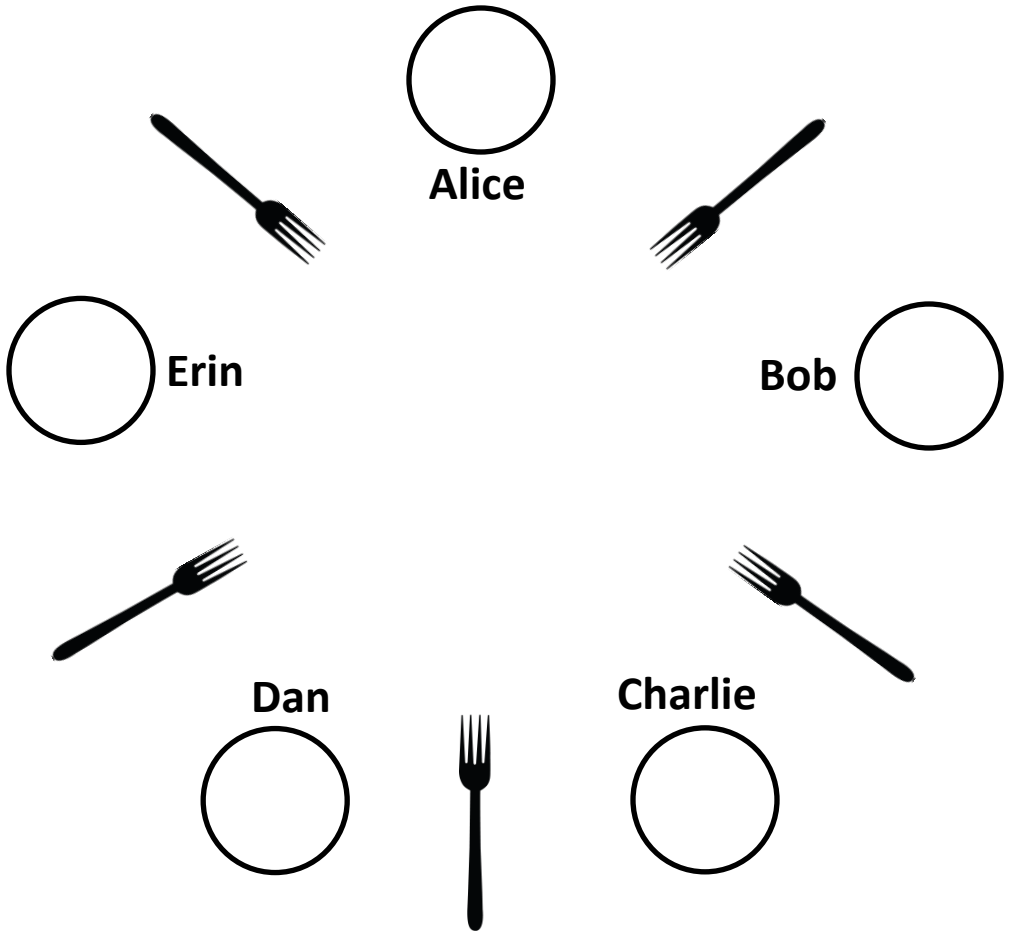
No Preemption? ☐

Circular Wait? ☐

Deadlock? ☐

Progress? ☐

Dinning Philosophers Problem



Mutual Exclusion? ☐

Hold and Wait? ☐

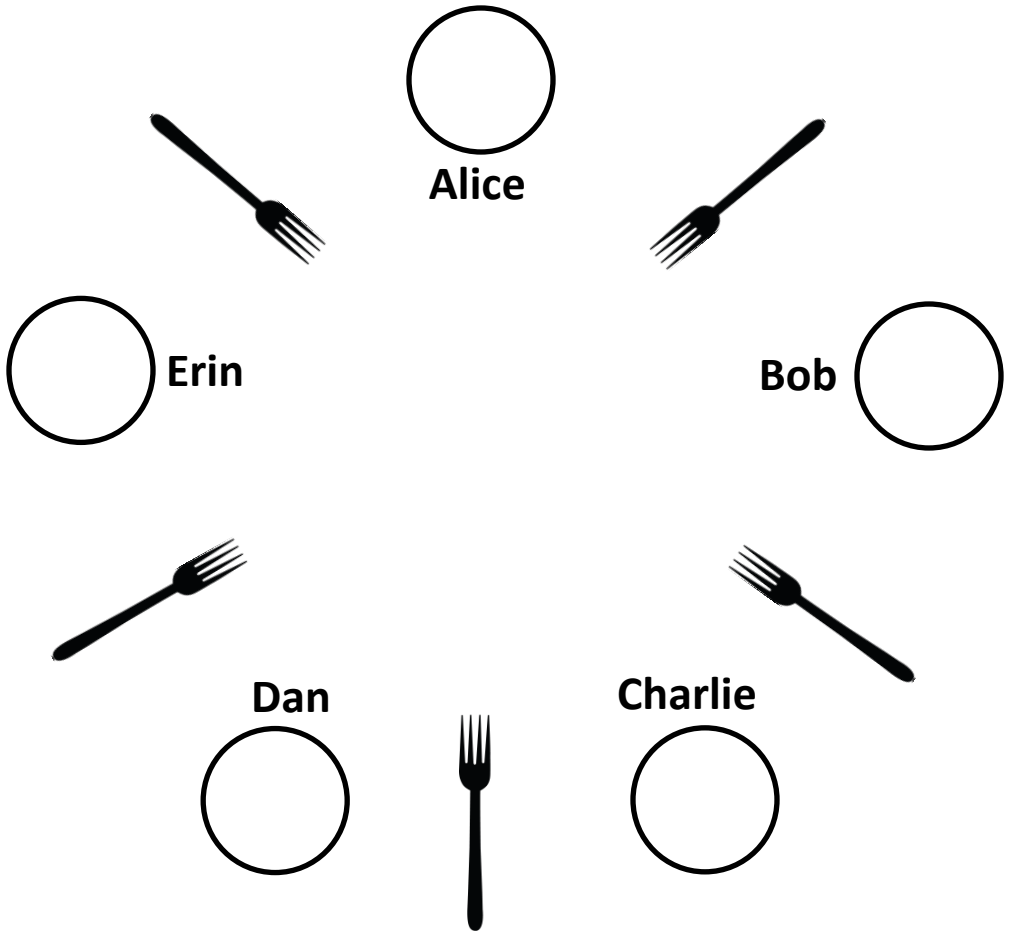
No Preemption? ☐

Circular Wait? ☐

Deadlock? ☐

Progress? ☐

Dinning Philosophers Problem



Mutual Exclusion? ☐

Hold and Wait? ☐

No Preemption? ☐

Circular Wait? ☐

Deadlock? ☐

Progress? ☐

Classical Synchronization

- Four classical problems
  -
  -
  -
  -

Producer-Consumer Problem

- The **producer-consumer problem** is an instance of a blocking bounded queue.
  - **Producers** add to the queue
  - **Consumers** consume from the queue

```
void produce(void *item) {
  sem_wait( &sem_empty_spots );
  pthread_mutex_lock(&mutex);
  queue_push(item);
  pthread_mutex_unlock(&mutex);
  sem_post( &sem_filled_spots );
}
```

```
void* consume() {
  sem_wait( &sem_filled_spots );
  pthread_mutex_lock(&mutex);
  void *item = queue_pop(item);
  pthread_mutex_unlock(&mutex);
  sem_post( &sem_empty_spots );
  return item;
}
```

Readers-Writers Problem

- Consider multiple processes accessing a single file, some reading some writing.
  - Multiple processes can **read** simultaneously so long as no process is **writing**.
  - No processes can **write** while a process is **reading**.
  - Only one process can **write** at a time.

- How do we develop a fair algorithm to implement these rules?

```
void read() {

  /* Read from the file */

}
```

```
void writer() {

  /* Write to the file */

}
```

```
void read() {

  /* Read from the file */

}
```

```
void writer() {

  /* Write to the file */

}
```

```
void read() {

  /* Read from the file */

}
```

```
void writer() {

  /* Write to the file */

}
```

Readers-Writers Solutions

- Solution #1:
- Solution #2:
- Solution #3: