ELEC 377 – Operating System

Week 3 – Class 2

Last Class

- n process solution - Bakery Algorithm
- Hardware Support
- Semaphores
- Classic Problems

Semaphores

- Semaphores generalize easily
- Limited Number of processes (> 1)

  ◊ allocate tape drives for example

```
Semaphore TapeDrives = 5;                  // 5 tape drives


    …
    wait(TapeDrives)
    … use a tape drive
    signal(TapeDrives)
    ...
```

Semaphores Other Uses

- Can Make one line wait for another.

◊ S=0

◊ what happens?

```
P0      P1
…       …
wait(S) signal(S)
…       …
```

Today

- Semaphores
- Classic Problems <<<<<<<<

Bounded Buffer

- Shared Data:

  semaphore full, empty, mutex;

  initially:

  full = 0; empty = n; mutex = 1;

  n is size of buffer;

Bounded Buffer - Producer

```
wait(empty);
wait(mutex);
  ... add to buffer ...
  signal(mutex);
signal(full);
```

- Note that mutex is symmetric, empty and full semaphores are not

Bounded Buffer - Consumer

```
wait(full);
wait(mutex);
   ... remove from buffer ...
   signal(mutex);
signal(empty);
```

- the empty semaphore contains the number of empty spaces left in the buffer
- the full semaphore contains the number of items in the buffer

Bounded Buffer

- Synchronized Counting of Items (producer/consumer)

```
Sempaphore empty = <num of queue entries>;
Sempaphore full = 0;
Sempaphore Mutex = 1;

Producer                 Consumer
… …
wait(empty)              wait(full)
wait(mutex)              wait(mutex)
… …
signal(mutex)            signal(mutex)
signal(full)             signal(empty)
… …
```

Bounded Buffer

- Synchronized Counting of Items (producer/consumer)

```
Sempaphore empty = <num of queue entries>;
Sempaphore full = 0;
Sempaphore Mutex = 1;

Producer                 Consumer
… …
wait(empty)              wait(full)
wait(mutex)              wait(mutex)
… …
signal(mutex)            signal(mutex)
signal(full)             signal(empty)
… …
```

Bounded Buffer

- Synchronized Counting of Items (producer/consumer)

```
Sempaphore empty = <num of queue entries>;
Sempaphore full = 0;
Sempaphore Mutex = 1;
```

Producer

```
… …
wait(empty)
wait(mutex)
… …
signal(mutex)
signal(full)
… …
```

Consumer

```

wait(full)
wait(mutex)

signal(mutex)
signal(empty)
```

Bounded Buffer

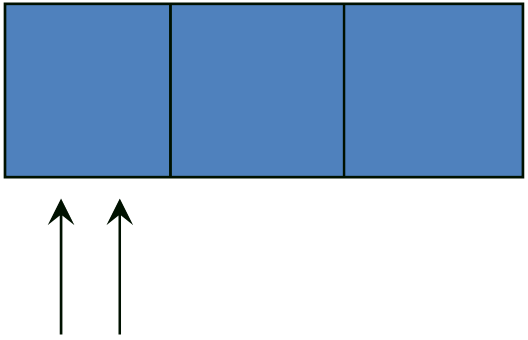
- Synchronized Counting of Items (producer/consumer)

Sempaphore empty = <num of queue entries>;
Sempaphore full = 0;
Sempaphore Mutex = 1;

```
Producer                 Consumer
… …
wait(empty)              wait(full)
wait(mutex)              wait(mutex)
… …
signal(mutex)            signal(mutex)
signal(full)             signal(empty)
… …
```

Bounded Buffer example

1 2 3 4 5
6

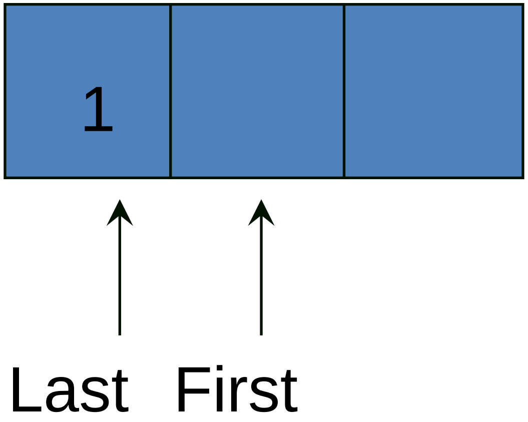


First Last

Empty = 3 Full = 0

Bounded Buffer example

2 3 4 5
6



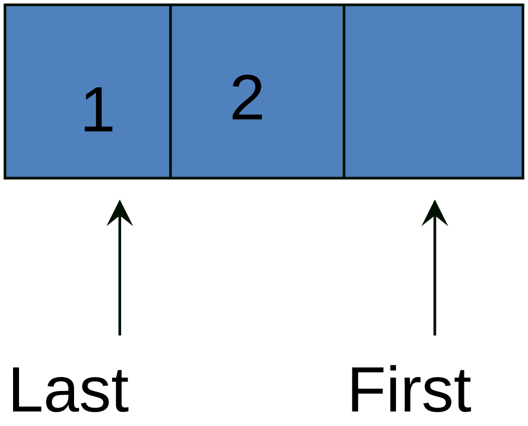
1

Last First

Empty = 2 Full = 1

Bounded Buffer example

3 4 5
6



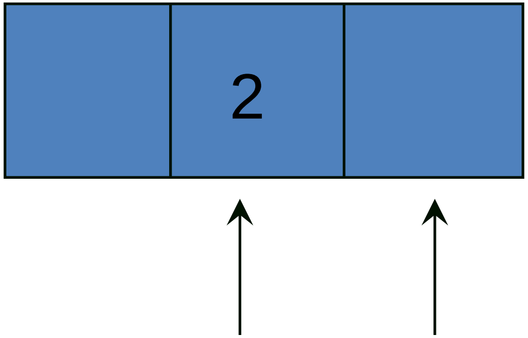
1 2

Last First

Empty = 1 Full = 2

Bounded Buffer example

3 4 5
6



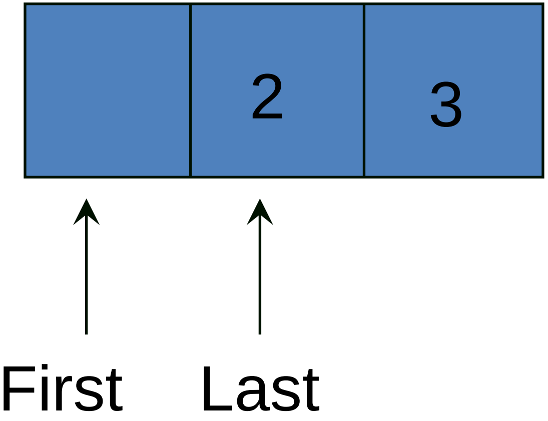
Last First

Empty = 2 Full = 1

1

Bounded Buffer example

4 5
6



2 3

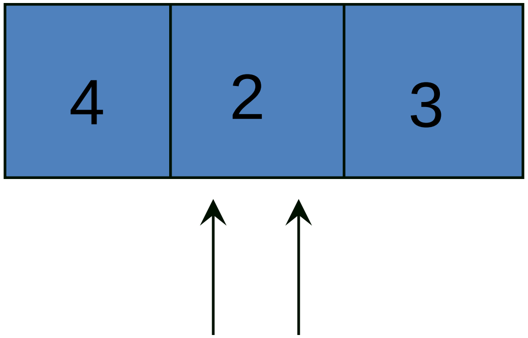
First Last

Empty = 1 Full = 2

1

Bounded Buffer example

5 6



4 2 3

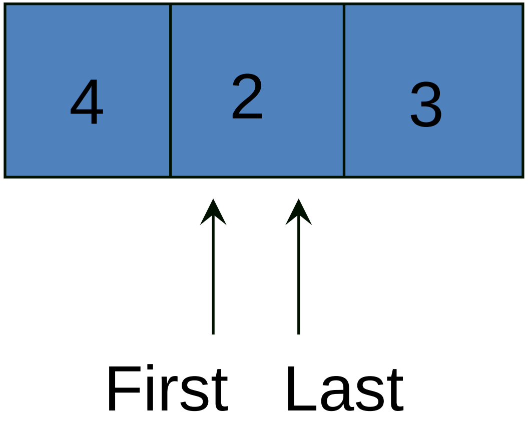
First Last

Empty = 0 Full = 3

1

Bounded Buffer example

6



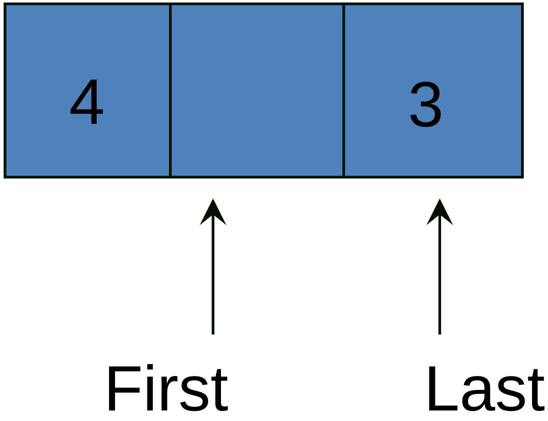
1

Empty = 0 Full = 3

5

Bounded Buffer example

6



4

3

First

Last

1 2

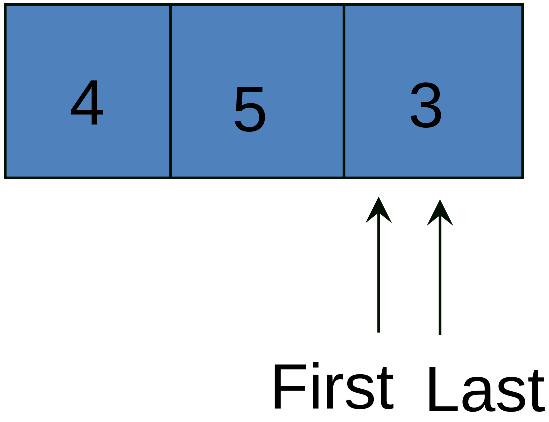
Empty = 1

Full = 2

5

Bounded Buffer example

6



4 5 3

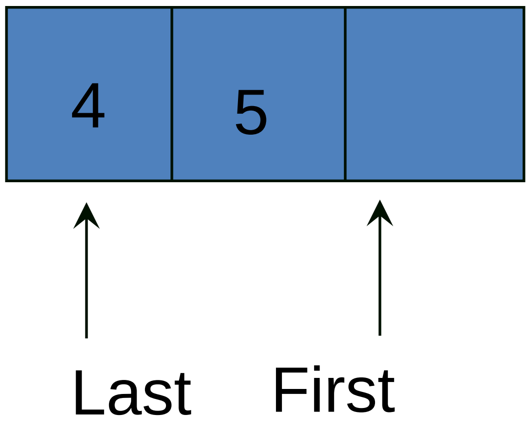
First Last

Empty = 0 Full = 3

1 2

Bounded Buffer example

6



4

5

Last

First

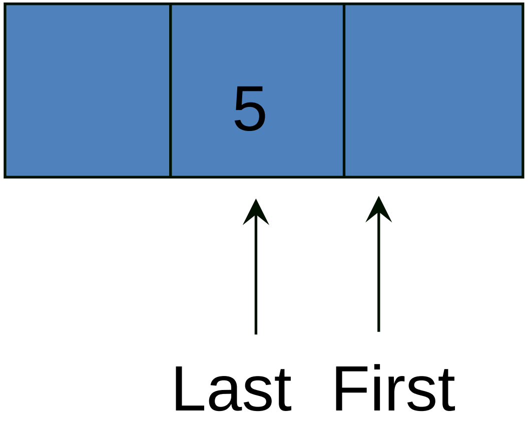
Empty = 1

Full = 2

1 2 3

Bounded Buffer example

6

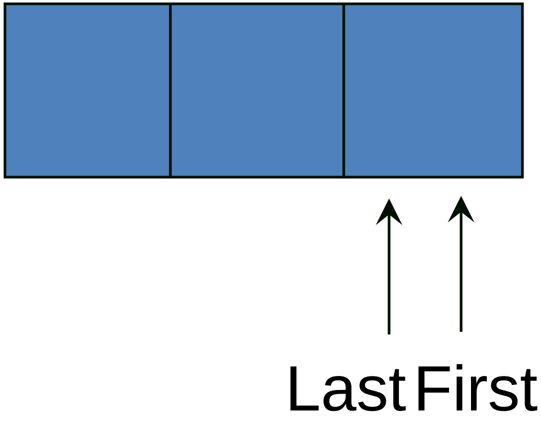


Empty = 2 Full = 1

1 2 3
4

Bounded Buffer example

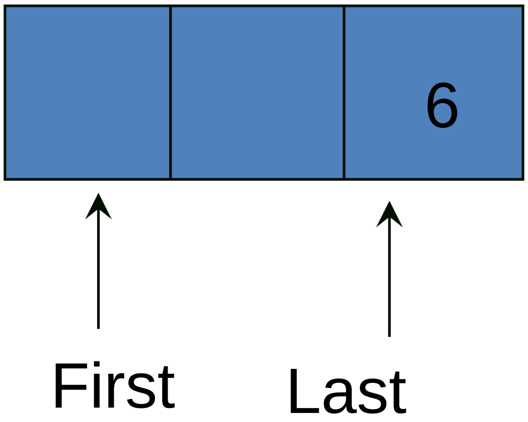
6



Empty = 3 Full = 0

1 2 3
4 5

Bounded Buffer example



6

First

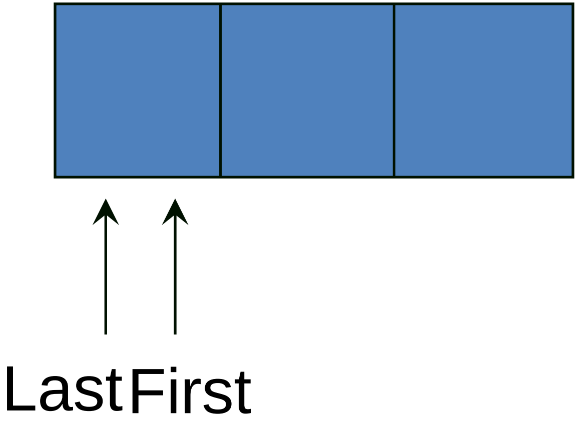
Last

Empty = 2

Full = 1

1 2 3
4 5

Bounded Buffer example



Empty = 3 Full = 0

1 2 3
4 5 6

Reader Writer Problem

- Only one process is allowed to write to a resource at a time
- More than one process is allowed to read from the resource at a time
- If a process is writing, no process can read
- If a process is reading, a writer must wait

Reader Writer

- Shared Data:
  semaphore mutex,wrt;
   int readcount

  initially:
  mutex = 1; wrt = 1; readcount = 0

Reader Writer - Writer

```
wait(wrt);

   ... read modify write ...

signal(wrt);
```

Reader Writer - Reader

```
wait(mutex);
readcount ++;
if (readcount ==1) wait(wrt);
  signal(mutex);

  ... read ...

  wait(mutex);
readcount --;
if (readcount == 0) signal(wrt);
signal(mutex);
```

Reader/Writer

- General Synchronization between processes

writer

```
   …
wait(wrt)
…write data…
signal(wrt)
…
```

reader

```
…
wait(mutex)
readcount++
if (readcount == 1)
    wait(wrt)
signal(mutex)
…read data…
wait(mutex)
readcount --;
if (readcount == 0)
signal(wrt)
signal(mutex)
…
```

Reader/Writer

- General Synchronization between processes

writer

```
   …
wait(wrt)
…write data…
signal(wrt)
…
```

reader

```
…
wait(mutex)
readcount++
if (readcount == 1)
   wait(wrt)
signal(mutex)
…read data…
wait(mutex)
readcount --;
if (readcount == 0)
signal(wrt)
signal(mutex)
…
```

Reader Writer

- General Synchronization between processes

writer

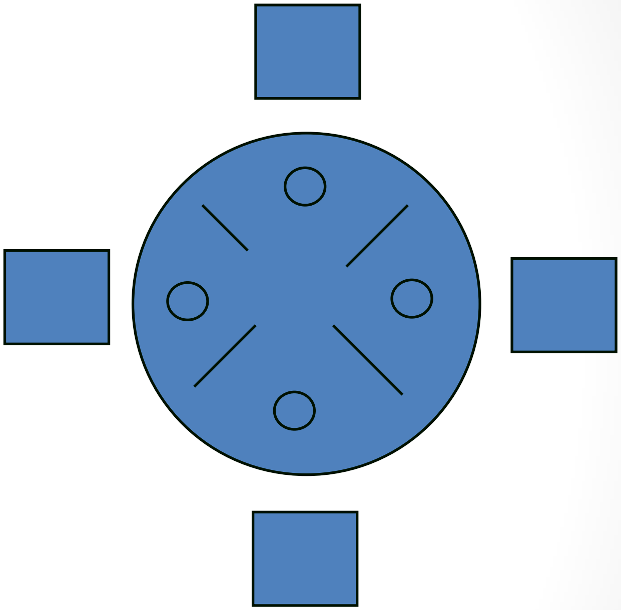
```
  …
wait(wrt)
…write data…
signal(wrt)
…
```

reader

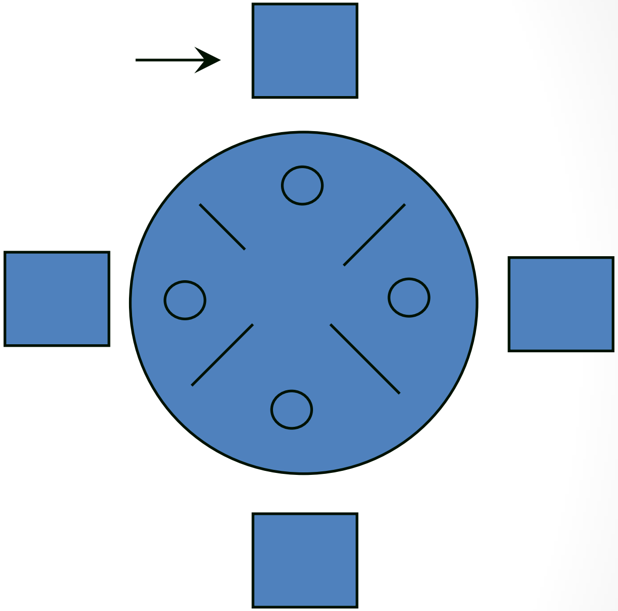
```
…
wait(mutex)
readcount++
if (readcount == 1)
    wait(wrt)
signal(mutex)
…read data…
wait(mutex)
readcount --;
if (readcount == 0)
signal(wrt)
signal(mutex)
…
```

Dining Philosophers

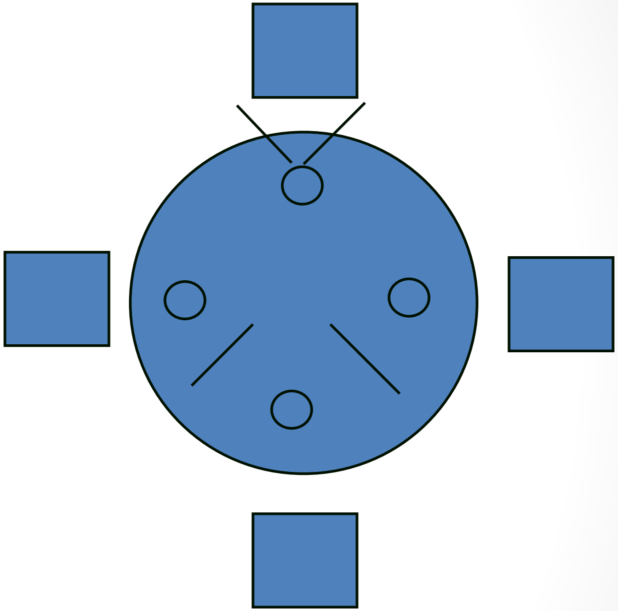
- N philosophers eating rice
- N chopsticks (1 between each philosopher)
- Each philosopher needs two chopsticks to eat
- Philosophers alternate between eating and thinking....



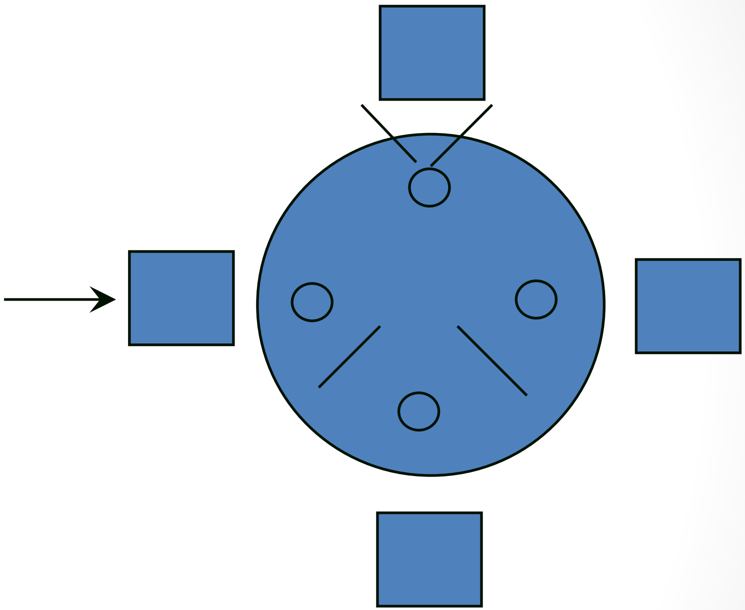
Dining Philosophers



Dining Philosophers

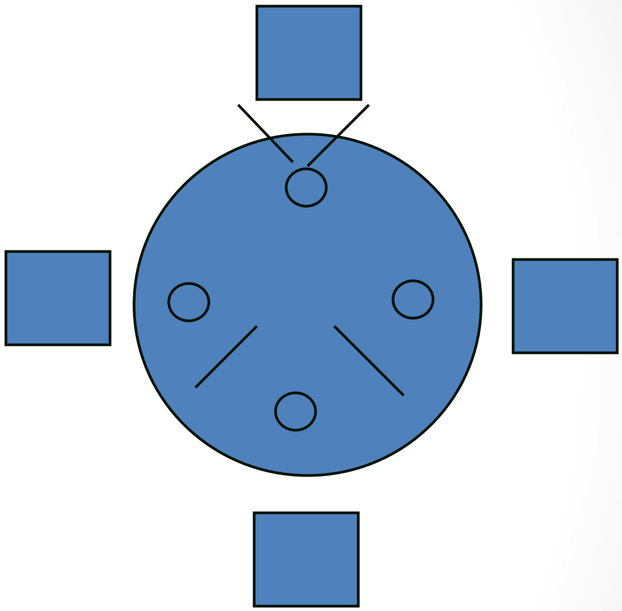


Dining Philosophers



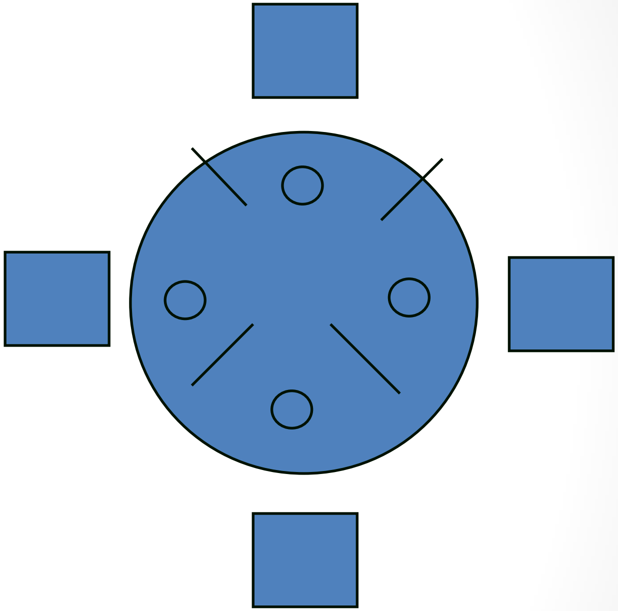
Dining Philosophers

wait

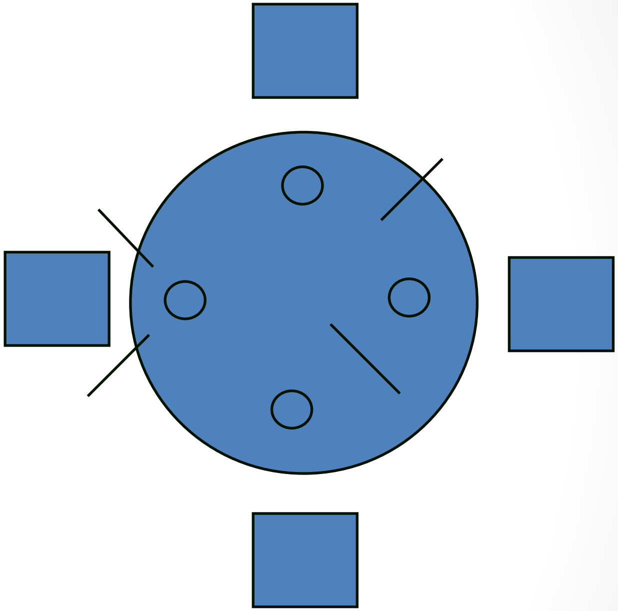


Dining Philosophers

wait



Dining Philosophers



Dining Philosophers

Sempaphore chopstick[n]

Dining Philosophers

```
Sempaphore chopstick[n]; // each init to 1

philosopher(int i){
  do {
    wait(chopstick[i]);
    wait(chopstick[(i+1)%n];
      // eat
    signal(chopstick[i]);
    signal(chopstick[(i+1)%n];
                      // think
  } while(1)
}
```