Deadlock and Monitors

Dining Philosophers: Solution

```
do forever:
    think
    get hungry
    down(chopstick[i])
    down(chopstick[(i+1) % N])

    eat

    up(chopstick[i])
    up(chopstick[(i+1) % N])
```

Does this solution work?

A. Yes
B. No

Bringing It All Together

- Processes
  - Abstraction for protection
  - Define address space
- Threads
  - Share (and communicate) through global and static data, share the heap, each has its own stack and full use of the registers
  - Race conditions may be a problem!
- CPU Schedulers
  - May pre-empt a process or thread at any time
- Ensuring correctness (OR eliminating race conditions and deadlock)
  - Safety and liveness
  - Atomic instructions
  - Synchronization: mutual exclusion, counted resources…
  - Locks, semaphores, monitors
  - Common patterns: Bounded Buffer, Dining Philosophers

Vocabulary

stolen resource
mutual exclusion
lock
deadlock
while loop
signal()
monitor

wait()
critical sections
no pre-emption
Hoare semantics
circular wait
resource variables
starvation

hold and wait
broadcast()
queue of waiting threads
general synchronization
condition variables
Mesa/Hansen semantics

Pemberley!