critical region: region of code we only want 1 thread accessing at a time

mutexes

mutual exclusion

"synchronization primitive"

like a lock on an open door

```
mutex m;
m.lock();
        ] critical region
m.unlock(); // any other thread waiting on m.lock() can proceed at this pt., "taking the lock"
```

↑ does not depend on which thread arrived at lock first! a set, not a queue of waiting threads

typically named `mutex varBeingProtectedLock;`

pass in mutex by reference!

ref(mLock) and mutex& mLock

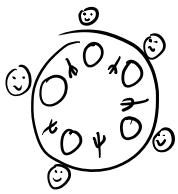
↑

in thread()

constructor

deadlock: all threads are "stuck" (maybe at an m.lock() but maybe for other reasons)

Dining Philosophers Problem



philosophers as array of threads

forks as array of mutexes

separate locks for all of your shared variables

*one way to check for race conditions in code: if you put in sleep calls & problems start to arise — they aren't creating problems, they're exposing them *

avoiding deadlock...

what about a stack of permits? only 2 philosophers can be eating at any given time

keep 4 permits — 1 fewer than would cause deadlock (e.g. w/ all 5 philosophers holding left forks)

mutexes: to prevent a thread from getting a stale old value of a variable

not necessary if all threads are only reading the variable!

necessary if any writing is happening

goal: make critical region as small as possible while solution still works