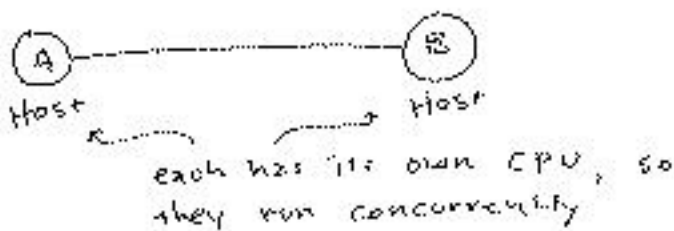


## Concurrency in Python

Network programming implies concurrent execution



Often we use concurrency even in a single host:

- ▷ So a server using TCP has multiple sockets
- ▷ Perhaps it is a dual CPU host
- ▷ Operating system usually simulates multiprocessor architecture

### Methods of concurrency in Python

1. Processes (different windows, etc)  
sharing variables is quite difficult
2. Threads  
sharing variables is "easy"
  - a. thread package (like Unix pthread)
  - b. threading package (like Java)

But how easy is it to share variables between threads?

Thread 1

$$r = x$$

$$r = r + 1$$

$$x = r$$

Thread 2

$$s = x$$

$$s = s + 1$$

$$x = s$$

But what we really want to see is

$$r = x$$

$$r = r + 1$$

$$x = r$$

$$s = x$$

$$s = s + 1$$

$$x = s$$

I.E.,  $x$  is a shared resource, and we need to have the threads take turns.

## "Taking Turns"

- locks
  - mutual exclusion
  - semaphores
  - ~~synchronized~~ not in Python!
- } can build objects,  
like shared queues, etc.

threading module

creation: `thread.Thread ( name = m )`  
↑  
name your new thread

ex: `t = thread.Thread ( name = 'thread one' )`

methods:

`t.start ( )`

`t.run ( )` ← you must  
write this one

`t.join ( )`

nb. `t.stop ( )` !

locks - mutually exclusive access to something (variable, printer, etc)

L = threading.Lock()

L.acquire()  
L.release()

} only one thread at a time

semaphores - generalize locks

S = threading.Semaphore(n=2)

← default is 1

S.acquire()  
S.release()

} only two at a time

queues (like Unix pipes)

Q = threading.Queue()

Q.get() → get an item or wait for one

Q.put() → insert an item