CSE 330: Operating Systems

Class: 15 **Date:** 2/27

Spring 2018

Note Title

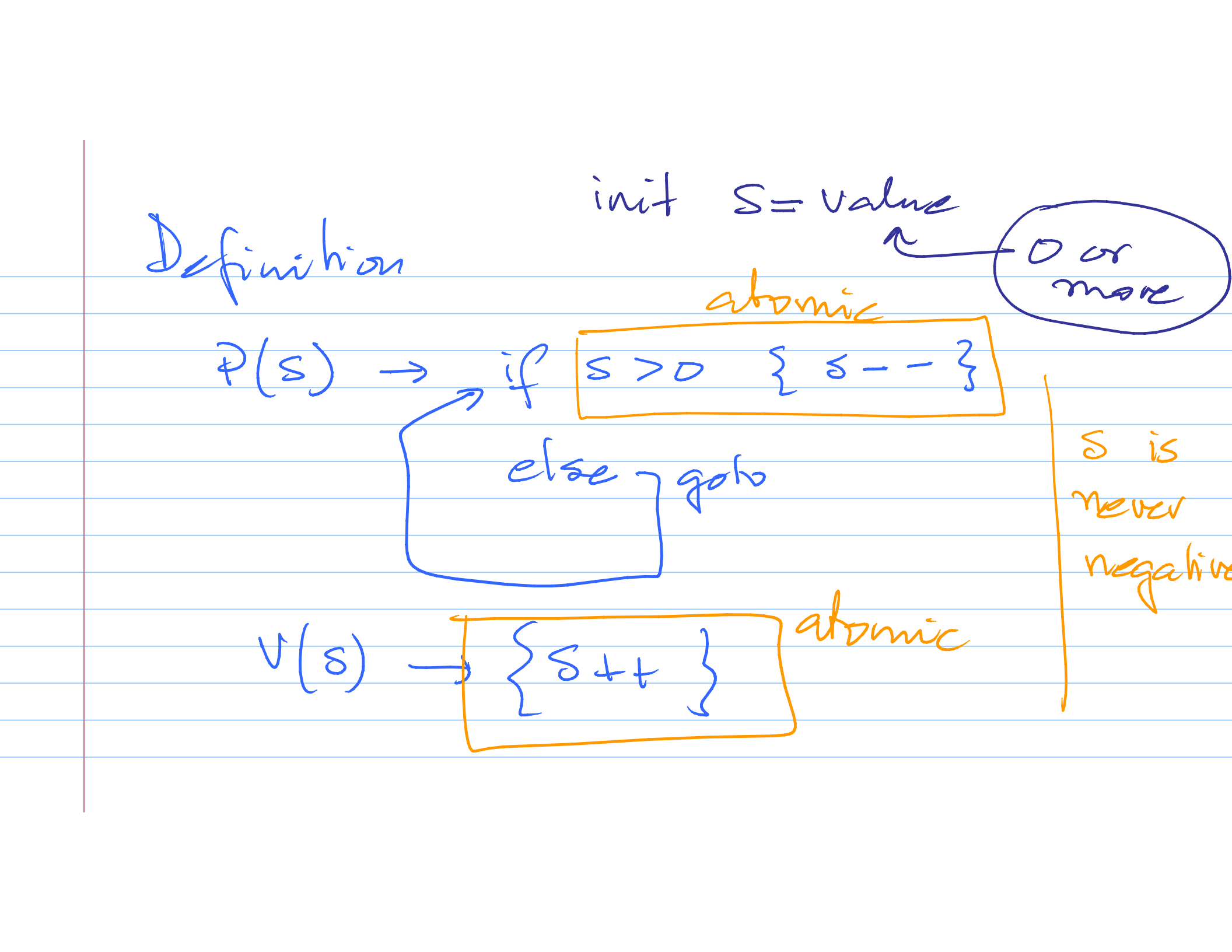
Semaphore

- critical section
- synchronization

→ prod/cons

→ R/W

→ Dining philosophers

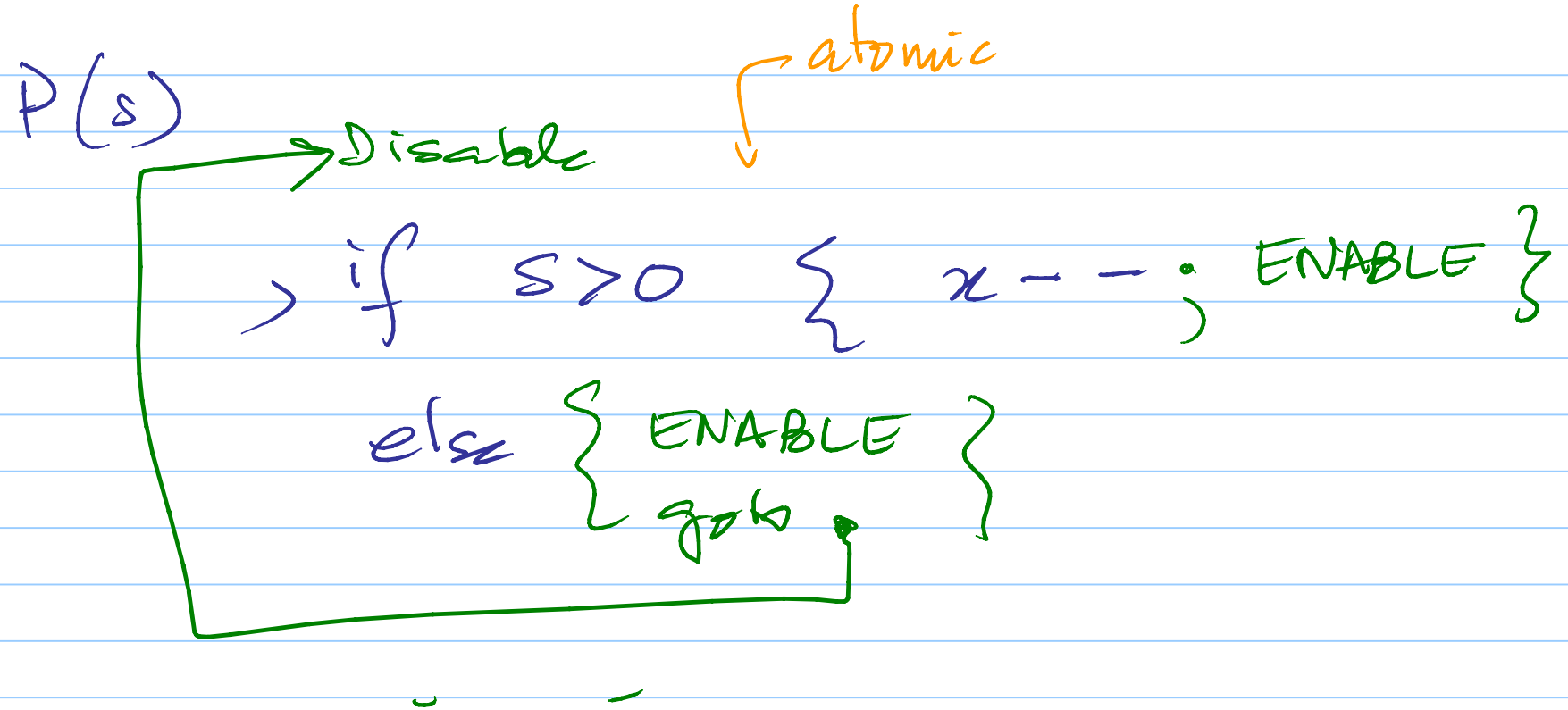

Definition
init S = value
0 or more
P(S) →
atomic
if S > 0 { S - - }
else goto
V(S) →
{ S + + }
atomic
S is never negative

Implementing Semaphores

1. Busy wait Semaphores
2. Blocking Semaphores

not good

UNIPROCESSOR



Multiprocessor

```
typedef semaphore
    struct { int count;
             int L }
```

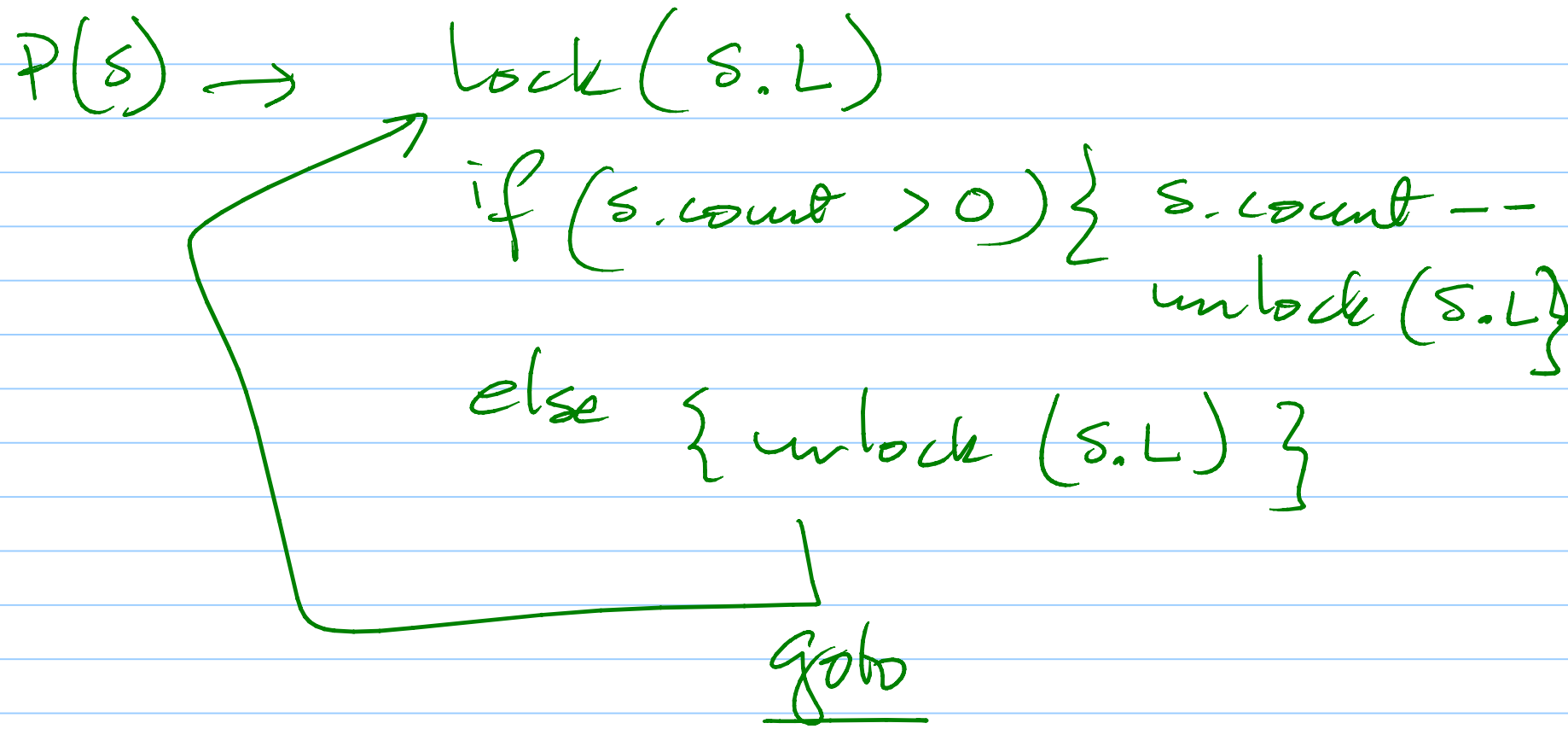
```
init Sem(S, v)
    { S.count = 1
      S.L = 0 }
```

??

int

lock(i) { while(test&set(i)); }

unlock(i) { i = 0 }


P(s) →
lock(s.L)
if (s.count > 0) { s.count --
unlock(s.L)
else { unlock(s.L) }
goto

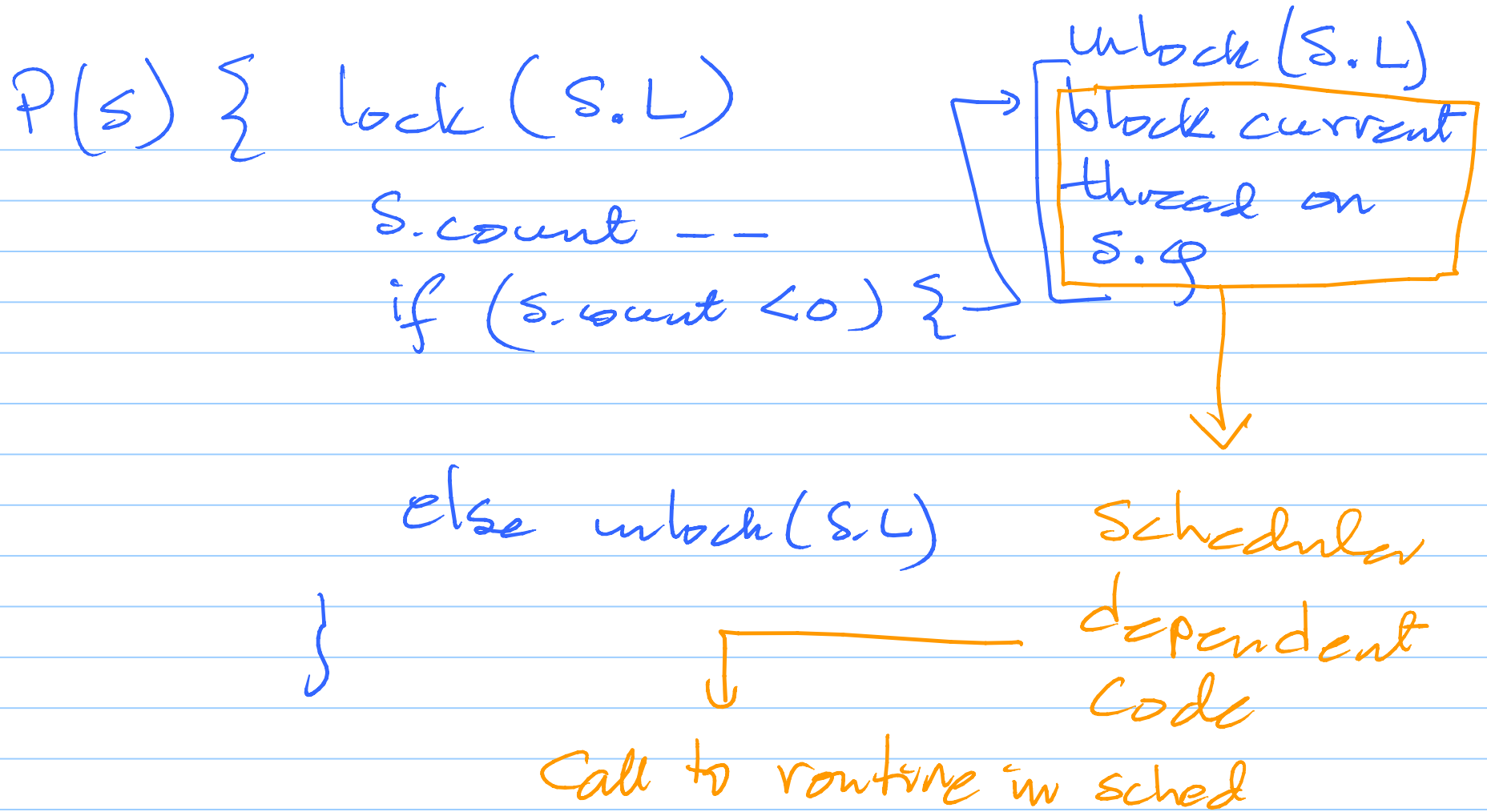
typedef Semaphore

{ int count

int L

PCB queue

*s_header q; }


P(s) { lock(S.L)
S.count --
if (S.count < 0) {
unlock(S.L)
block current thread on S.Q
else unlock(S.L)
}
Scheduler dependent code
Call to routine in sched

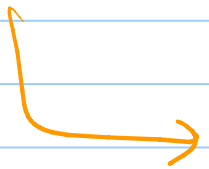
```
V(S) { lock(S.L)
       S.count ++
       if (S.count <= 0)
           { delete 1 PCB from
             S.Q and add to
             Ready Q }
       unlock(S.L) }
```

sched

Scm → {count, q}

initscm(s, v) { s.count = v
Initq(s.q) }

```
P(S) { S.count --
        if (S.count < 0) {
                curr = delQ(RunQ)
                next = RunQ
                AddQ(S.Q, curr)
                SwapContext(curr,
                            next);
        }
```



```
V(s) {  s.count ++
        if (s.count <= 0) {
                AddQ(RunQ, DelQ(s.Q))
        };
yield();
}
```

?

Monitors & conditions → they do not monitor anything / not conditions

→ class or abstract data type or name of a class

→ variable that has no value

```
monitor x {
  local variables
  condition c1, c2, c3

  method m1(arg) {                    }

  method m2(arg) {                    }
}
```

```
x.m1( )
x.m2( )
```

Monitor Semantics

- all methods in a monitor execute in one critical section
  - only 1 thread in a monitor (multiple threads can call but not execute)

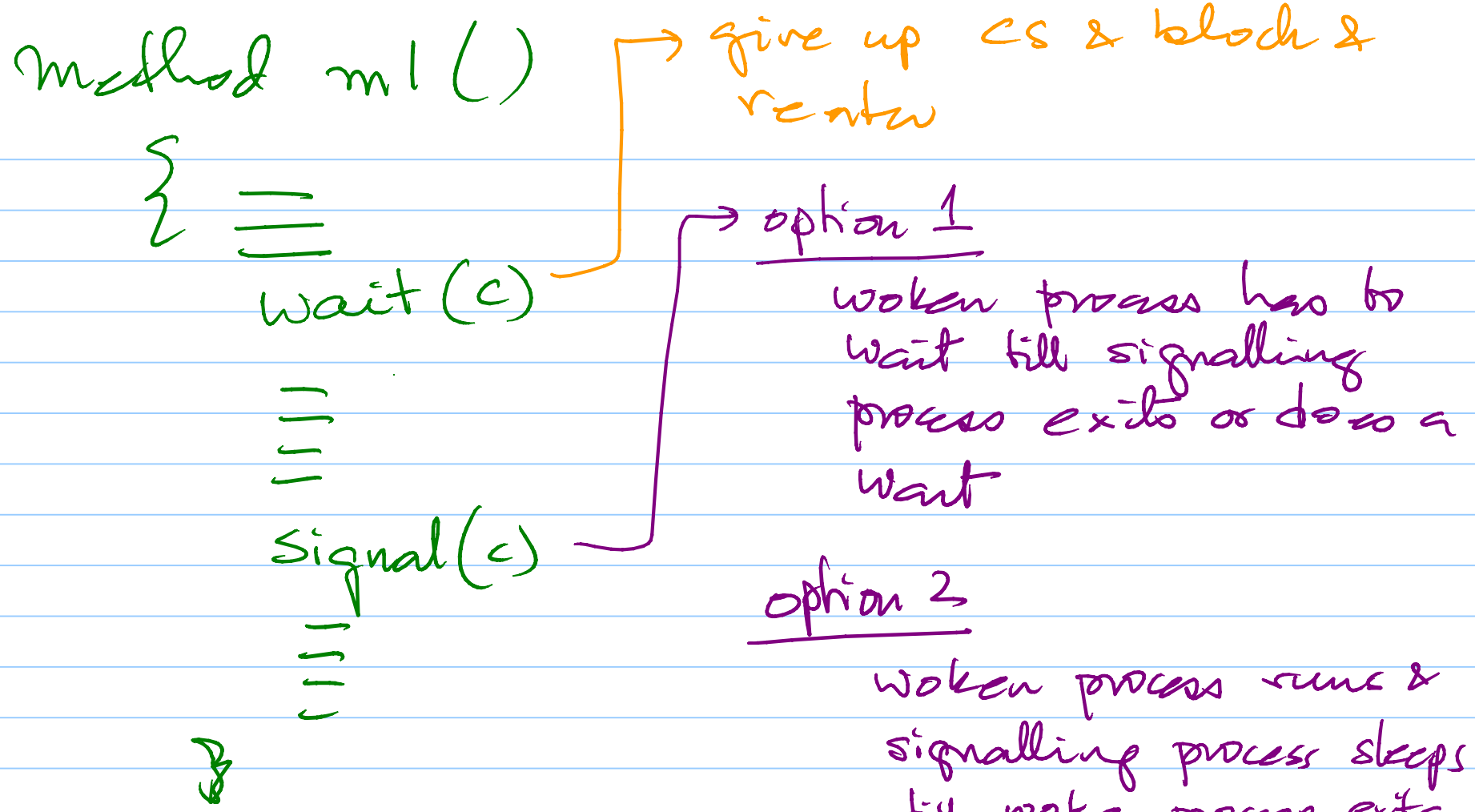
Condition variables support 2 methods

wait (cond) → step out of monitor & block; when awaken, re-enter monitor

unconditional block →

signal (cond) → wake* one process up if there are any blocked on the cond.

only used in monitor method


method m1()
{
wait(c)
signal(c)
}
give up cs & block & reenter
option 1
woken process has to wait till signalling process exits or does a wait
option 2
woken process runs & signalling process sleeps till woken process exits or it waits