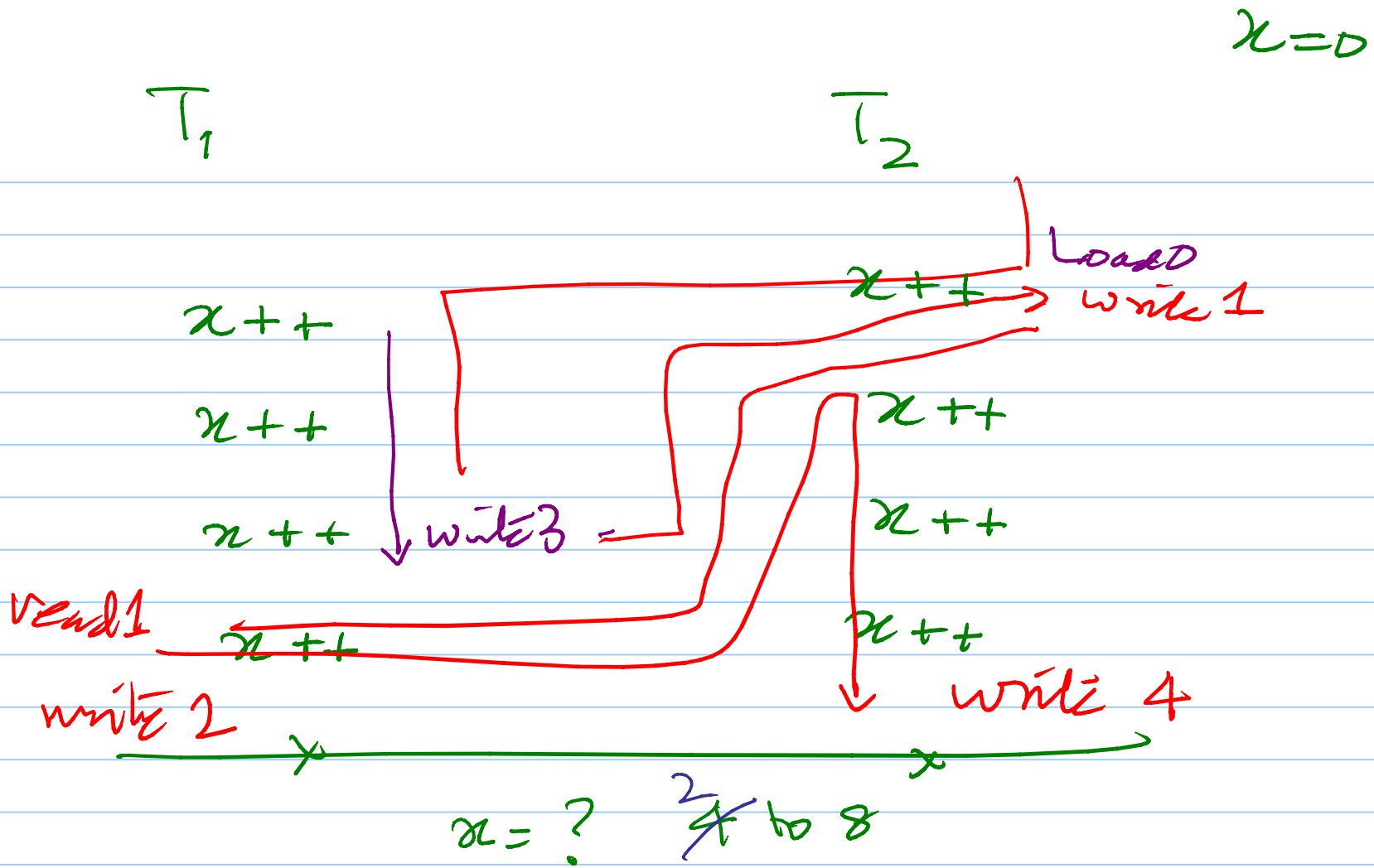
CSE 330: Operating Systems

Class: 12 **Date:** 2/15

Spring 2018

Note Title

- Concurrency
- Semaphores
  - → Dijkstra
    - → usage → critical sections
      - → synchronization


x=0
T1
T2
x++
x++
x++
write3
x++
write1
x++
x++
x++
read1
x++
write 2
write 4
x = ?
2 to 8

How to implement threads

- ~~use pthreads library~~

- from scratch
  - → need PCBs
    - → build context switching
  - → need scheduler
  - → need queues

Goal

- user level threads
- non-preemptive scheduling

use a process & make it run threads

main

{

start thread(f1)

start thread(f2)

start_thread(f)

{ allocate a ~~PCB~~ q_item that contains a PCB

a stack

init PCB to point to(f)

put ~~PCB~~ q_item into ReadyQ

thread_id = counter++

global

}

ucontext library

ucontext_t ← predefined struct
that contains a PCB

&-item → struct { next, prev → ptr to &item
ucontext_t pcb;
int threadid;
}

initialize PCB

malloc(8192) → stackP

8192 → stacksize

initTCB (&item, f, stackP, stacksize)

{ put zeros into &item)

getcontext (&&item → PCB)

&item → PCB.uc_stack.ss_sp = stackP.

&item → PCB - stack.ss.size = stacksize

makecontext (&&item → PCB, f, 0);

}

the swapcontext routine

library routine → swapcontext (from , to)

from → ptr to PCB

to → ptr to PCB

main → { start thread , startthread, Run)

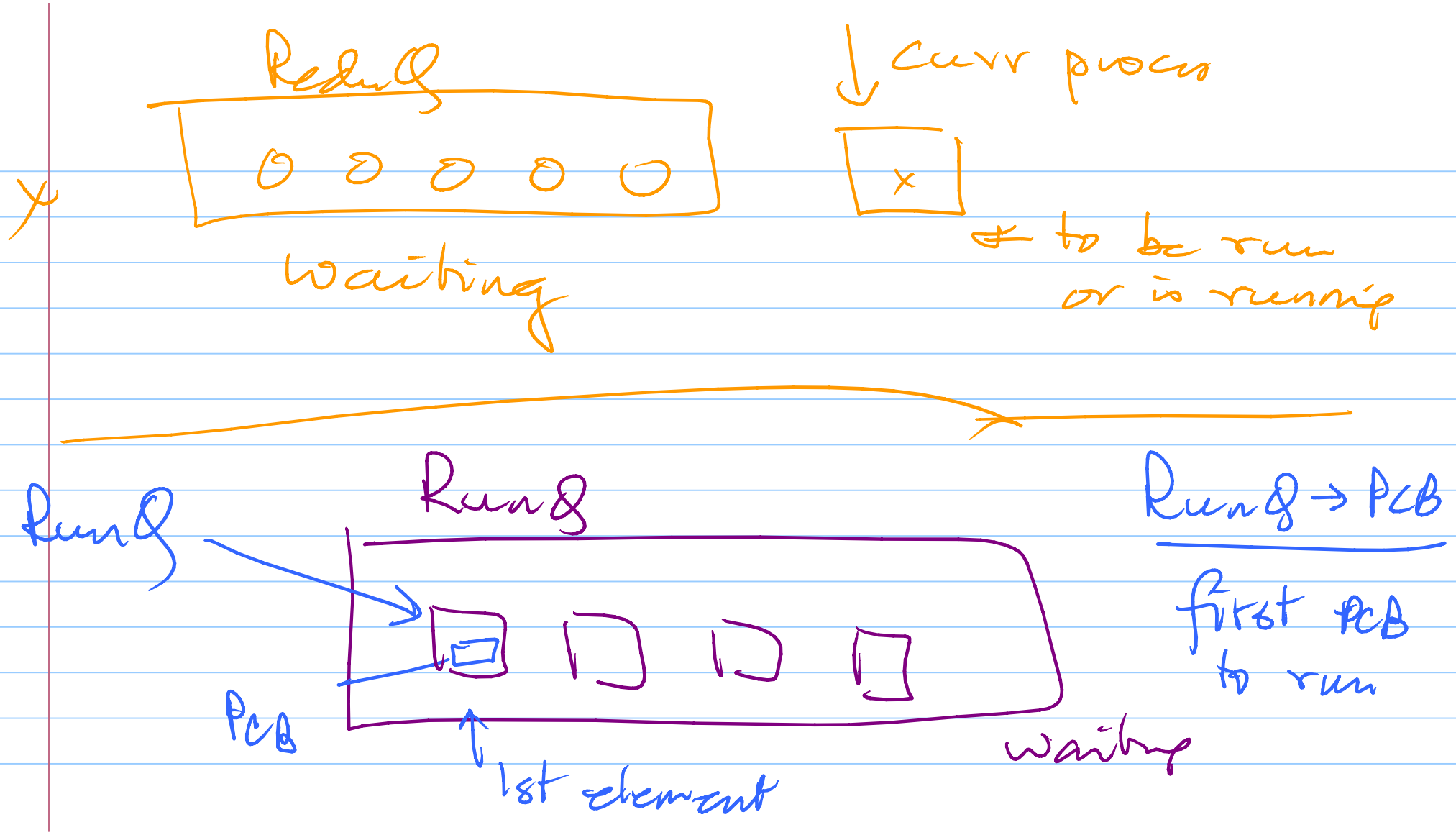
Run () { ucontext_t parent;

→ getcontext (&parent)

→ swapcontext (&parent, &(kernel → RCB));

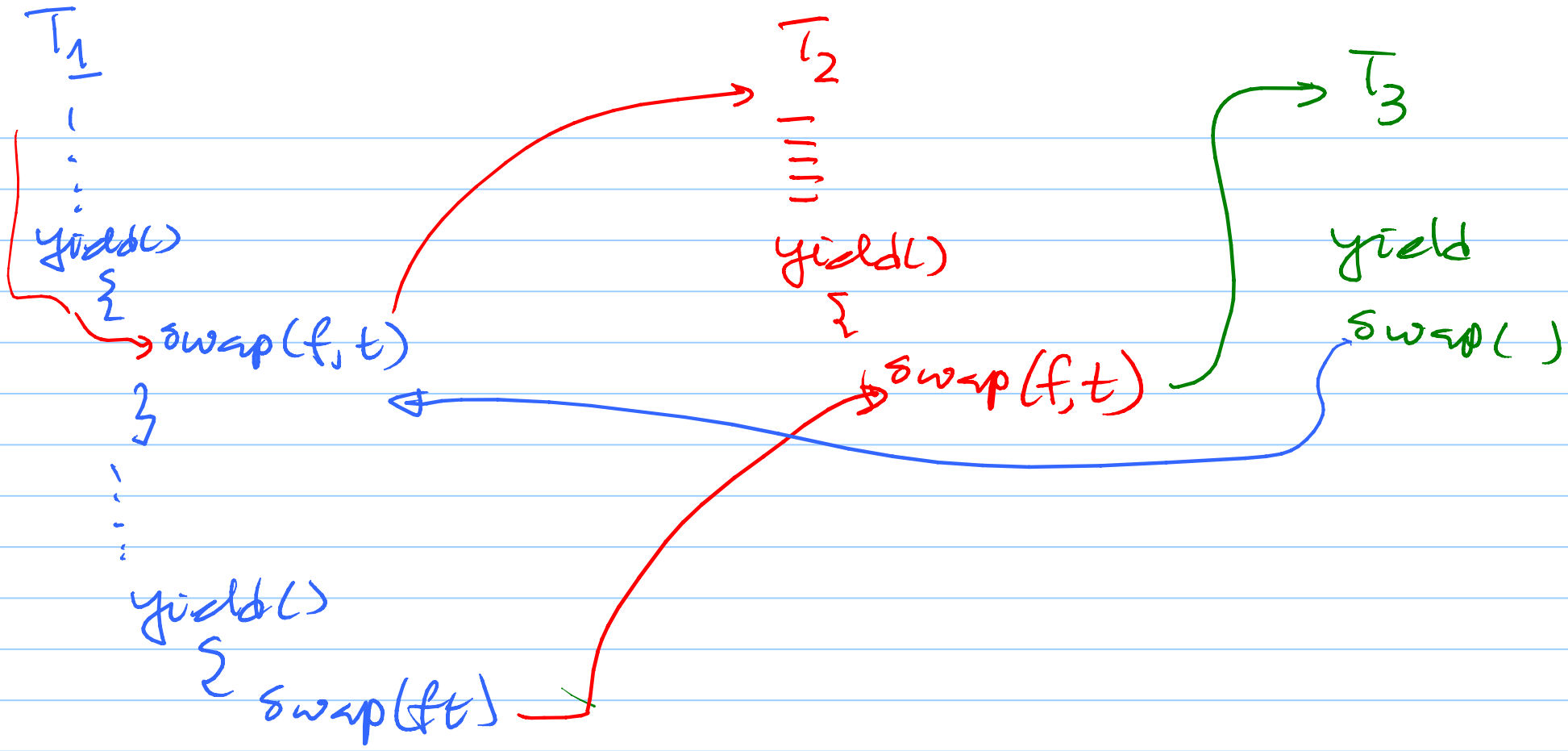
f → runs for ever

first PCB


Ready
curr proc
x
waiting
to be run
or is running
Run Q
Run Q
Run Q → PCB
first PCB
to run
PCB
1st element
waiting

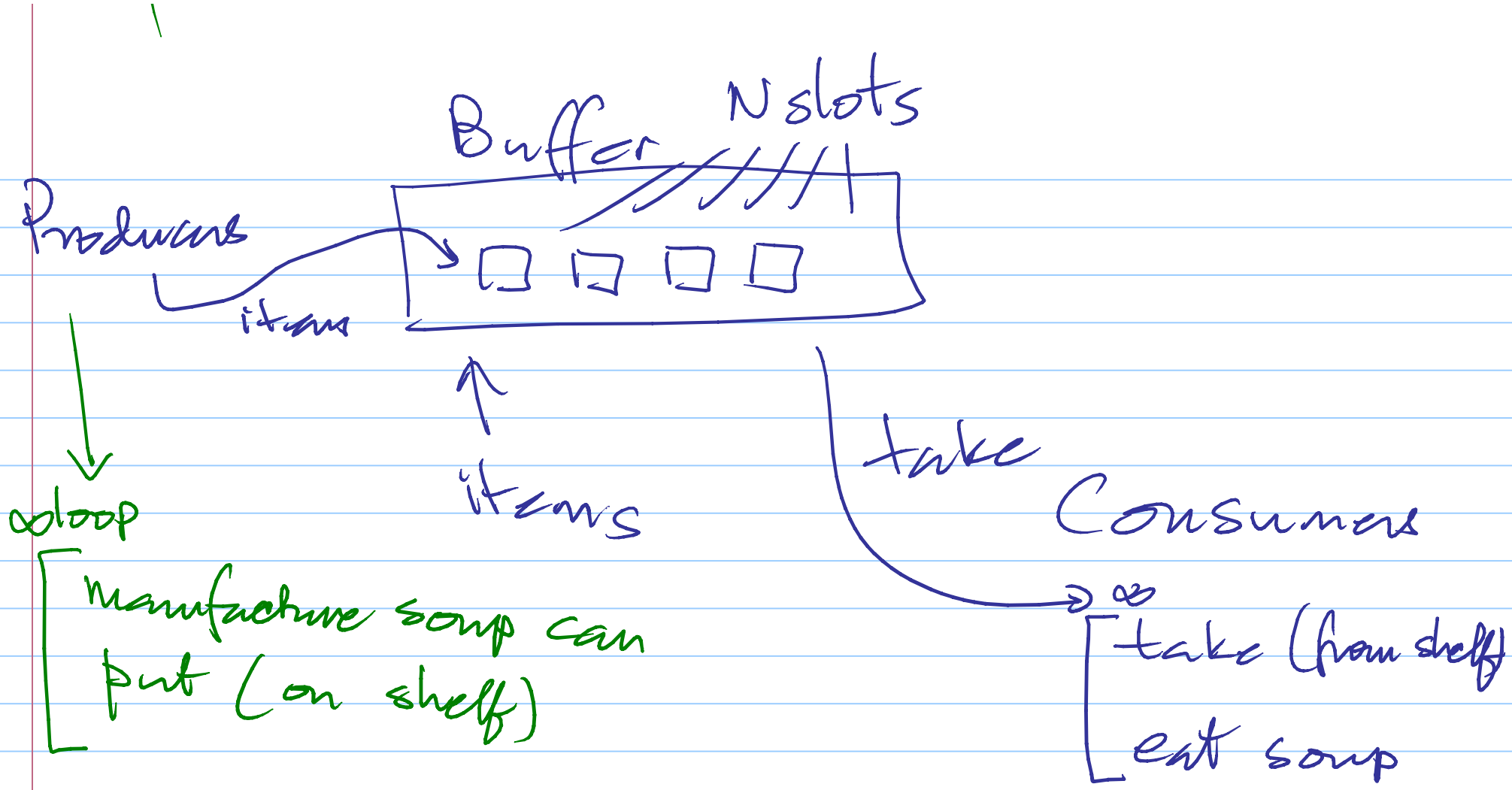
```
f() { while(1) { print
                 sleep(1);
                 yield(); } }
```

```
yield() { from = RunQ -> PCB
          RotateQ(&RunQ)
          to = RunQ -> PCB
          swapcontext(from, to);
        }
```


T1
yield()
{
swap(f, t)
}
yield()
{
swap(ft)
T2
yield()
{
swap(f,t)
T3
yield
swap( )

Semaphore programs (classical process coordination problems)

- Bounded Buffer (aka Producer Consumer)
- Readers & writers
- Dining Philosophers

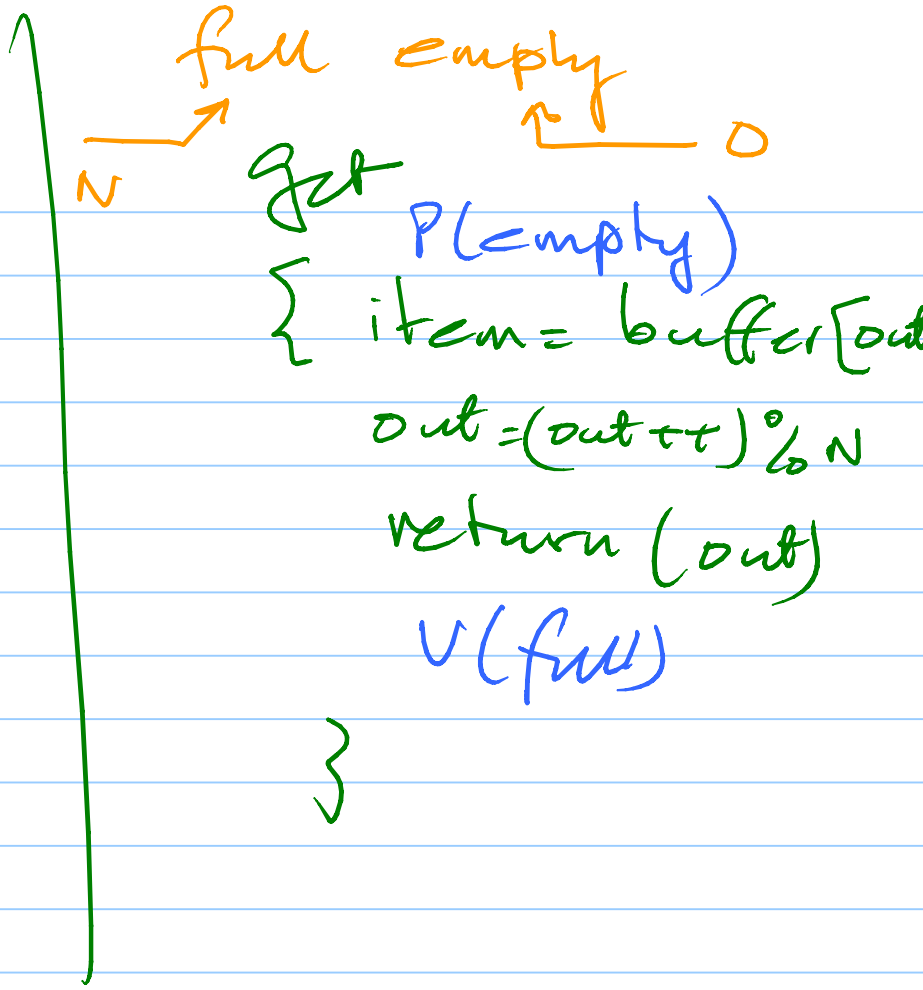

Buffer
N slots
Producers
items
loop
manufacture soup can
put (on shelf)
items
take
Consumers
take (from shelf)
eat soup

1 prod, 1 cons

item type

```
int buffer[N]
int in, out = 0
```

```
put(item)
{   P(full)
    buffer[in] = item;
    in = (in++) % N
    V(empty)
}
```



full
empty
full
in
in
out