

Mutual Exclusion with Busy Waiting: Peterson's algorithm

```
#define N 2

int turn;
int interested[N];

void enter_region(int process) {
    int other = 1 - process;
    interested[process] = TRUE;
    turn = process;
    while(turn==process && interested[other]==TRUE) ;
}

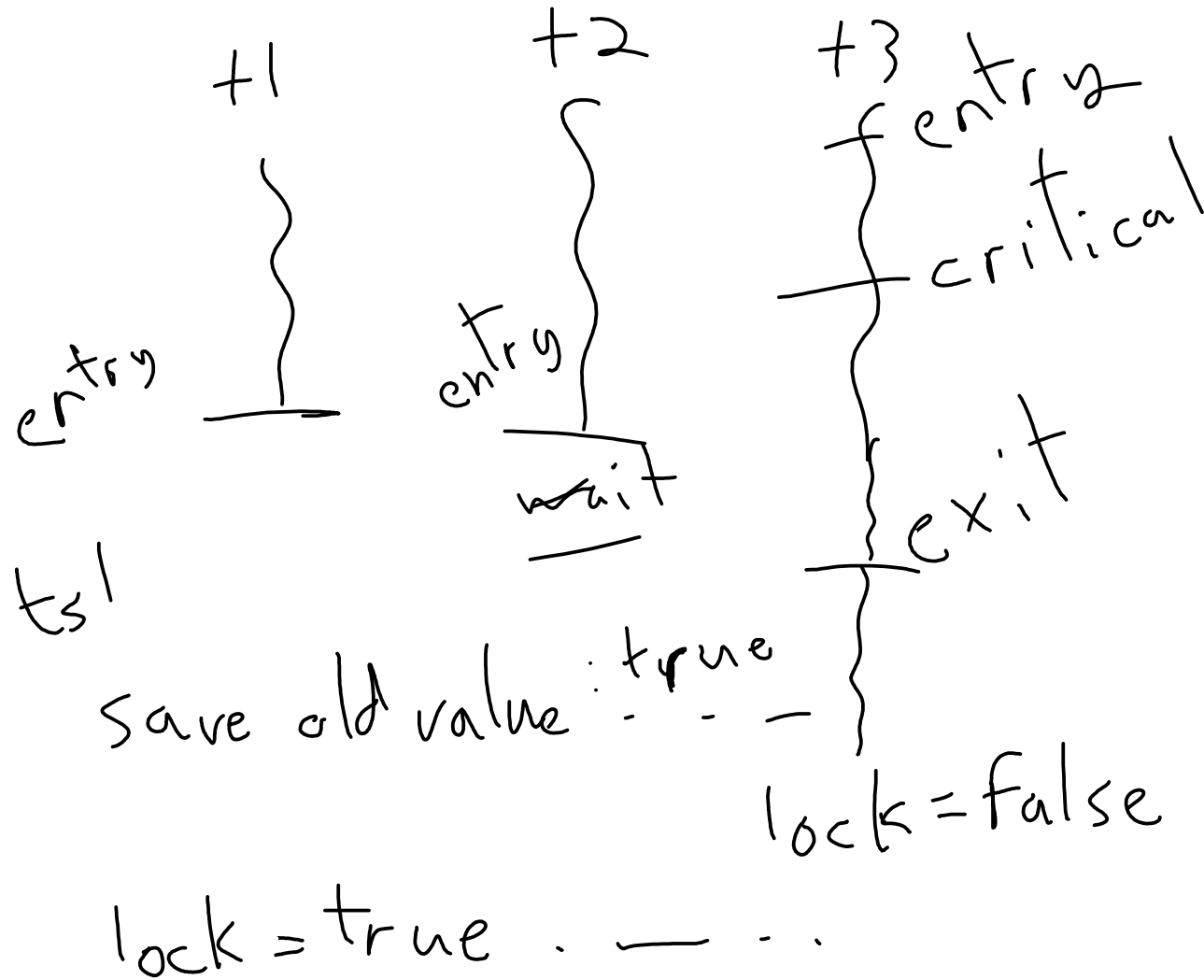
void leave_region(int process) {
    interested[process] = FALSE;
}
```

Instructions

test and set


atomic

TS L



Semaphores

integer

operations

P

down

sleep

acquire

V

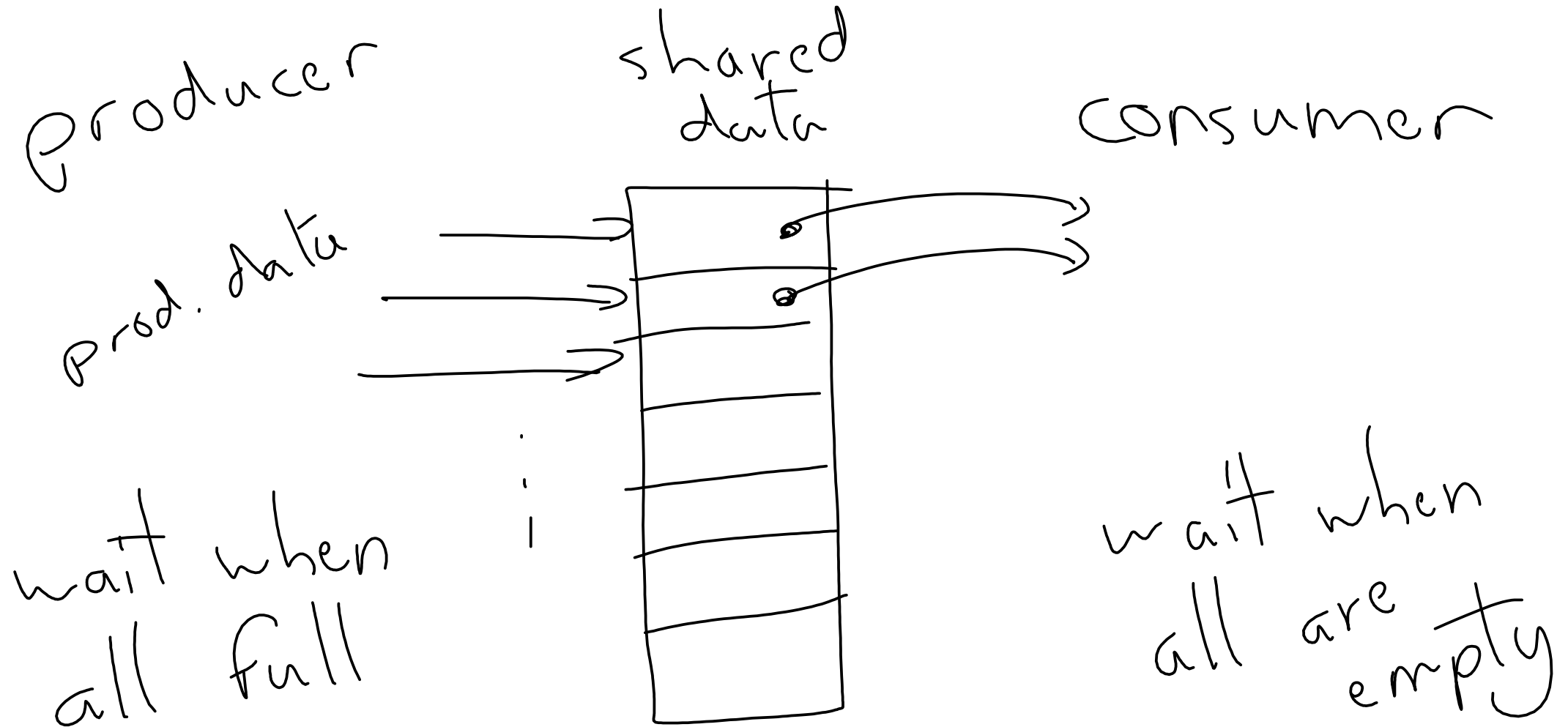
up

wake

release

of permits

Producer/Consumer (Bounded Buffer)



Semaphores: Producer-Consumer (1 of 5)

```
#define N 100

typedef int sema;
sema mutex=1;
sema empty=N, full=0;
```

```
void producer(void) {
    int item;
    while(TRUE) {
        item = produce_item();
        [
            down(&empty);
            down(&mutex);
        ]
        insert_item(item);
        [
            up(&mutex);
            up(&full);
        ]
    }
}
```

```
void consumer(void) {
    int item;
    while(TRUE) {
        [
            down(&full);
            down(&mutex);
        ]
        item = remove_item();
        [
            up(&mutex);
            up(&empty);
        ]
        consume_item(item);
    }
}
```

remainder
critically entry
exit

entry
critical
exit

Readers/Writers Problem

Readers/Writers (1 of 6)

- N processes access (i.e., read or write) some shared data
- At any given time: R readers **or** 1 writer allowed. Basic solution:

